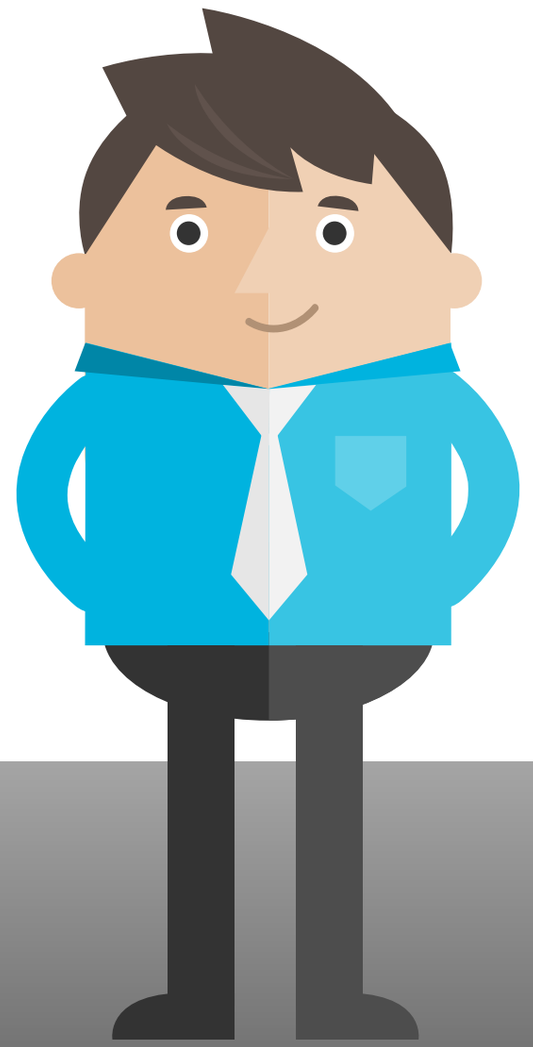
Performance Matters

Emery Berger & Charlie Curtsinger*
College of Information and Computer Sciences

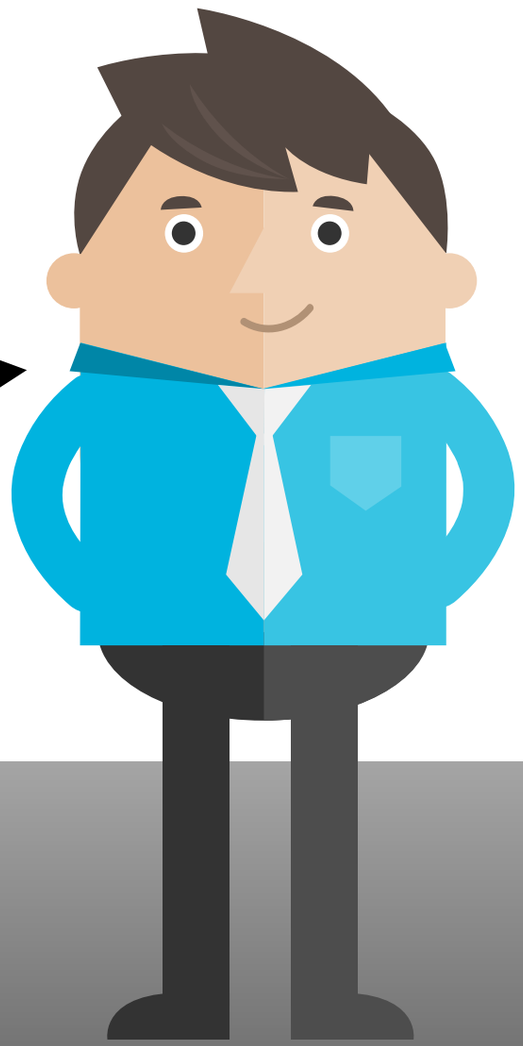
UMassAmherst

*now at *Grinnell College*

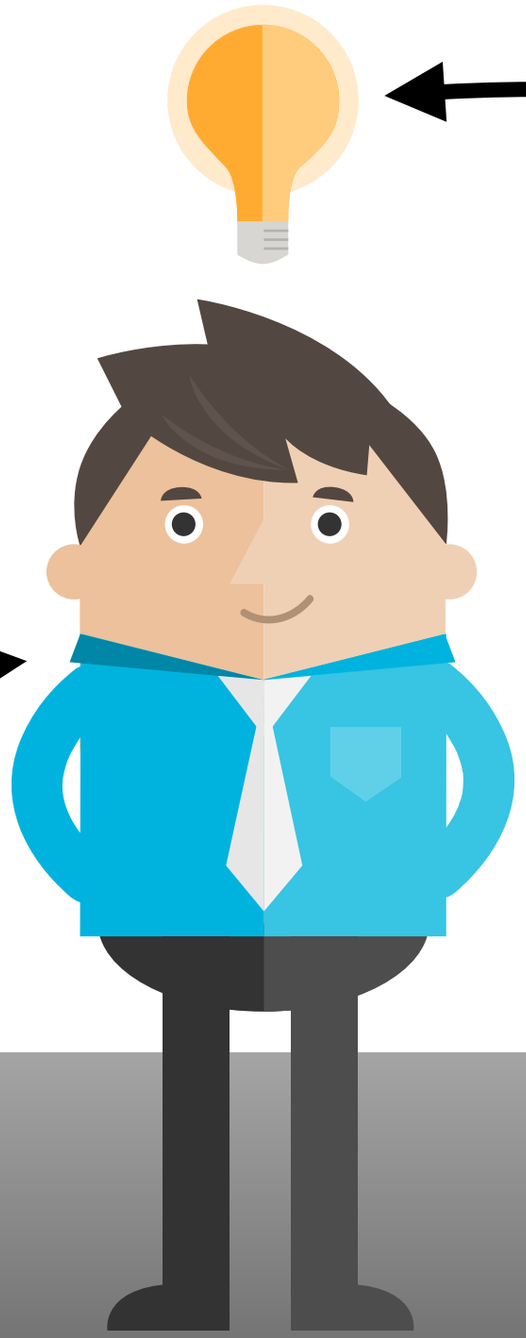
A few months ago,
in a valley far, far away...



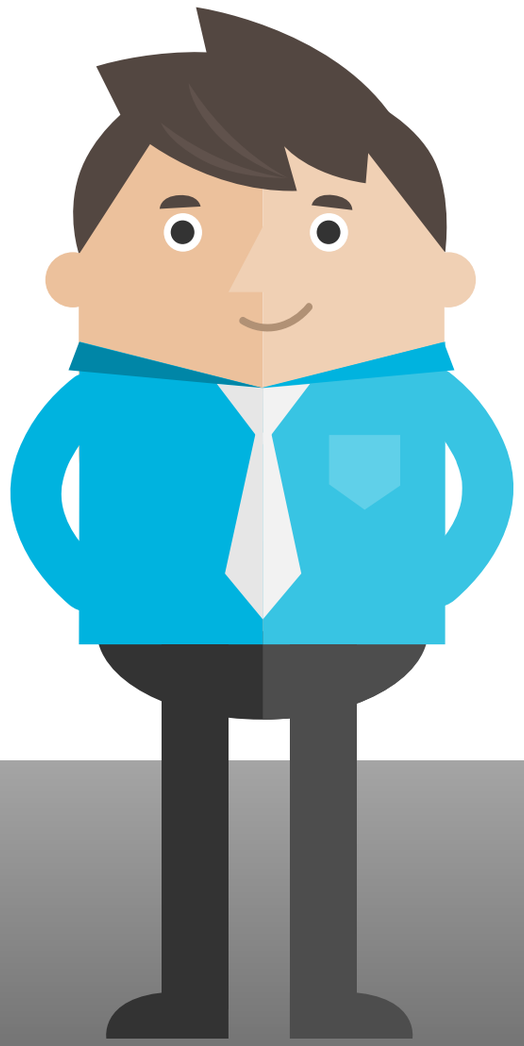
This is Bob.

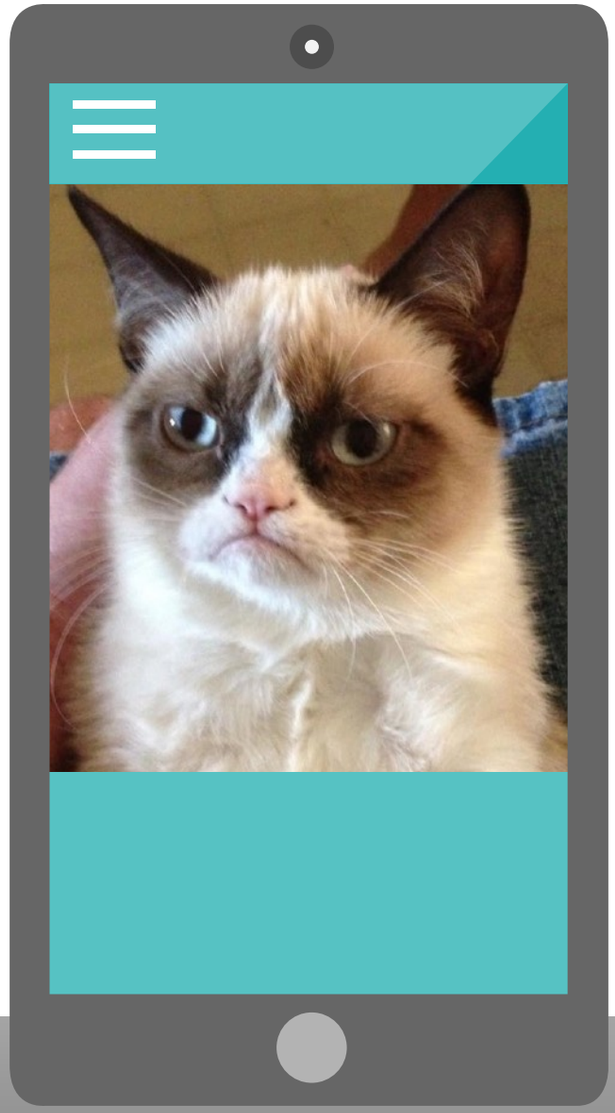


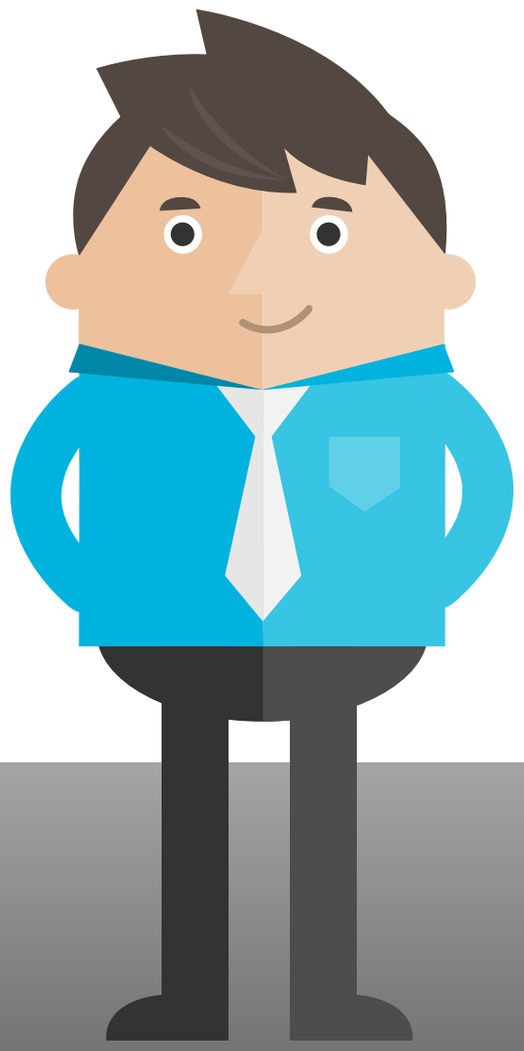
This is Bob.

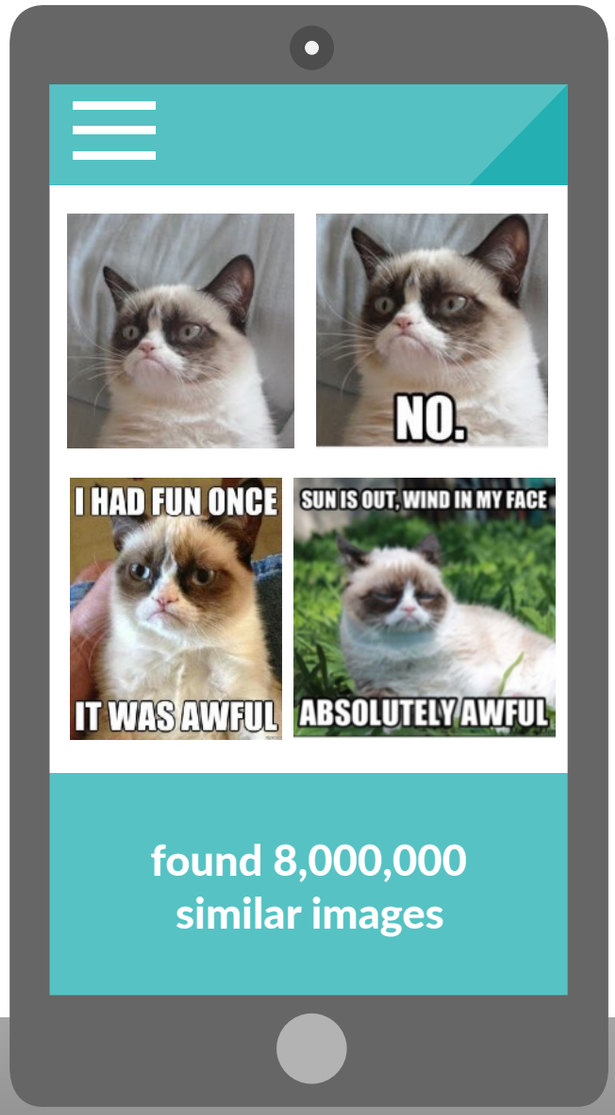


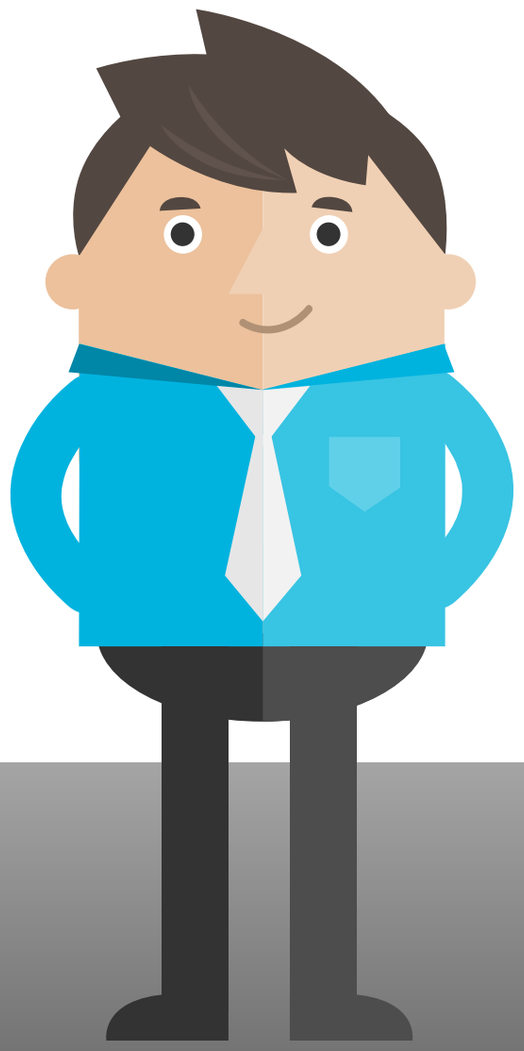
Bob has an idea for a startup.





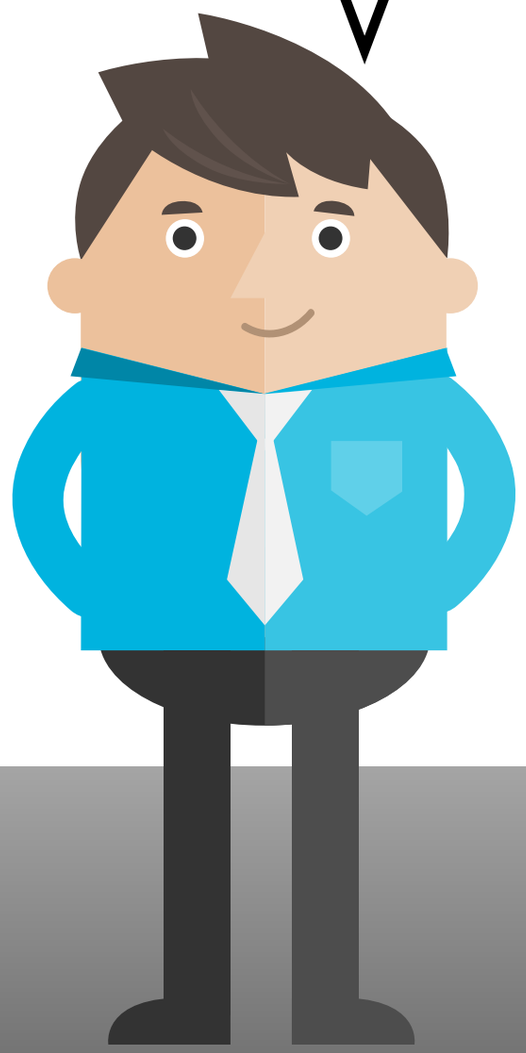



NO.
I HAD FUN ONCE
IT WAS AWFUL
SUN IS OUT, WIND IN MY FACE
ABSOLUTELY AWFUL
found 8,000,000
similar images

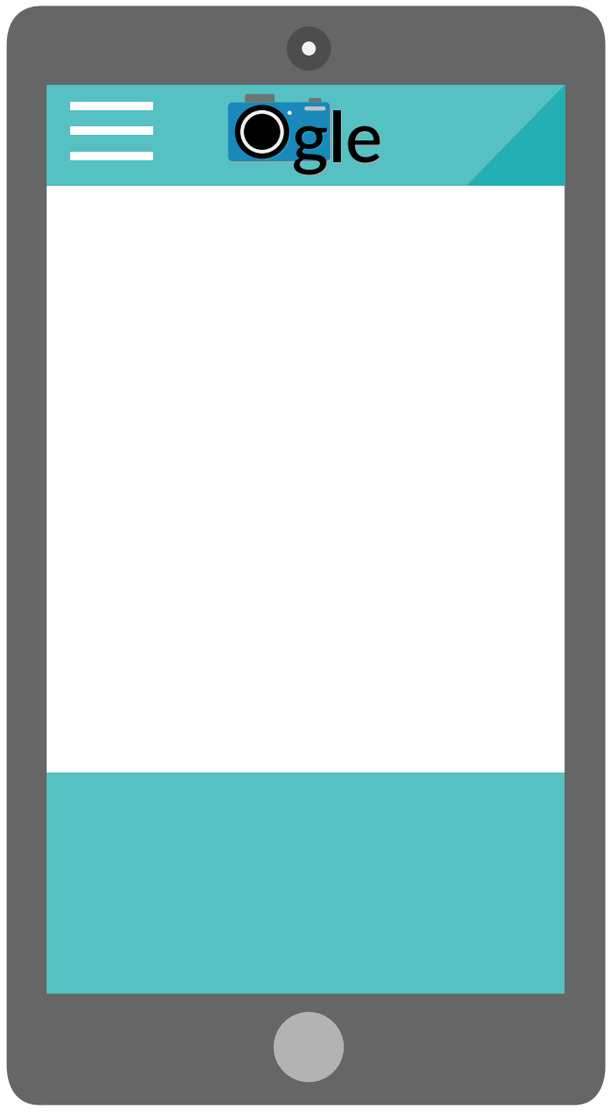



NO.
I HAD FUN ONCE
IT WAS AWFUL
SUN IS OUT, WIND IN MY FACE
ABSOLUTELY AWFUL
found 8,000,000
similar images

It’s going to totally disrupt image search.

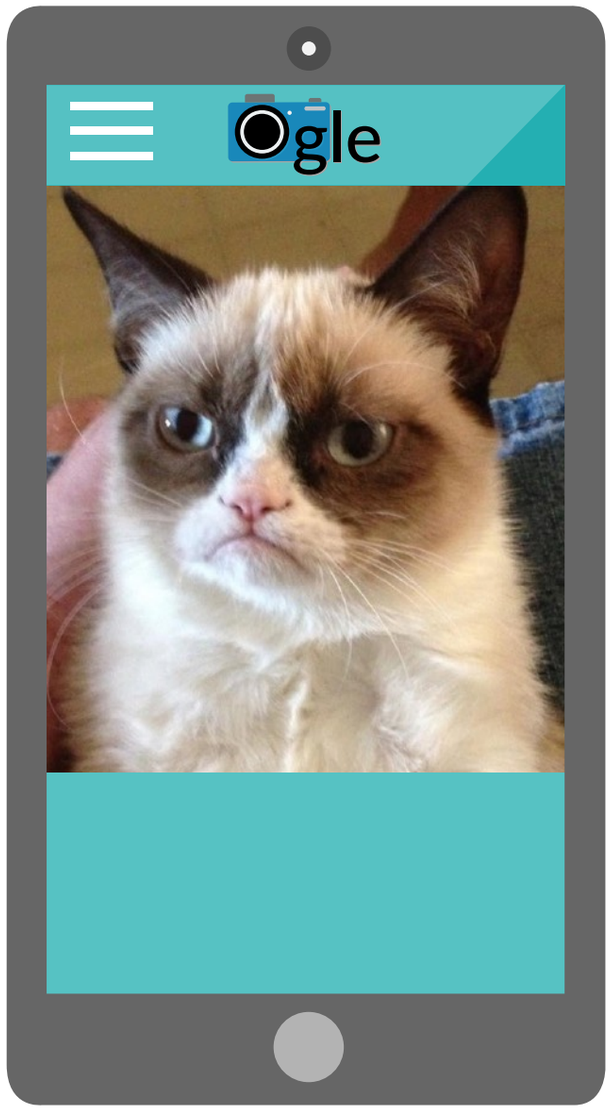


The Prototype Ogle



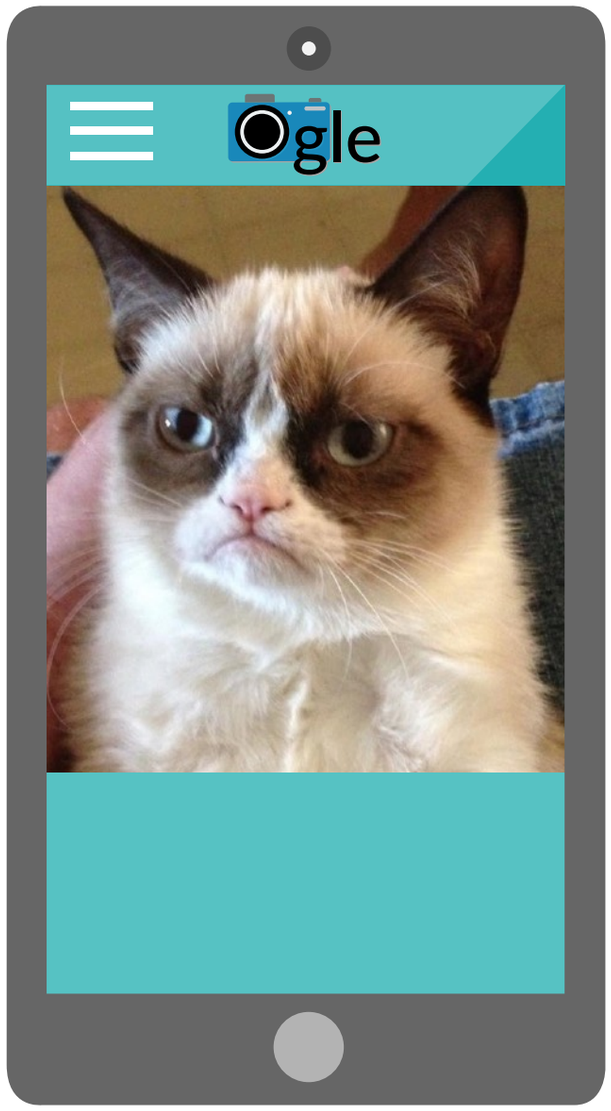
The Prototype Ogle

Take a picture

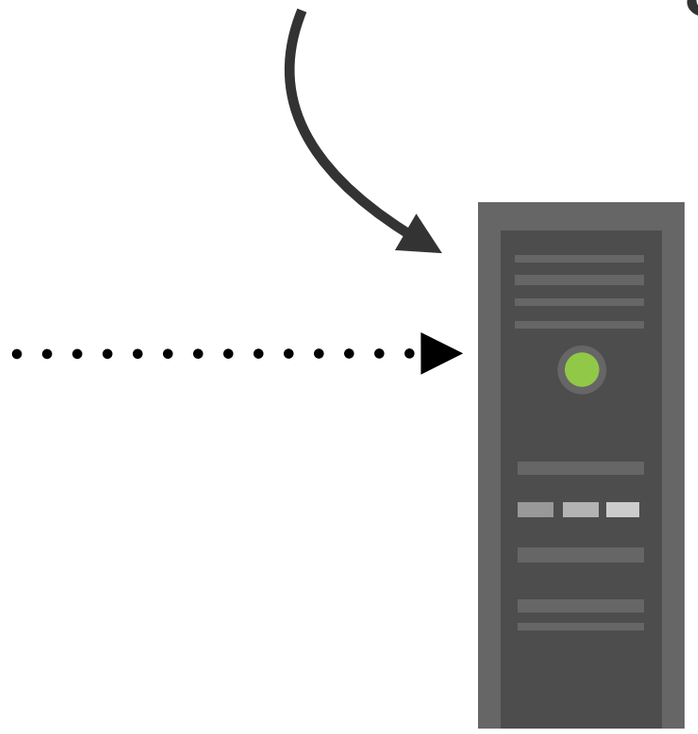


The Prototype Ogle

Take a picture

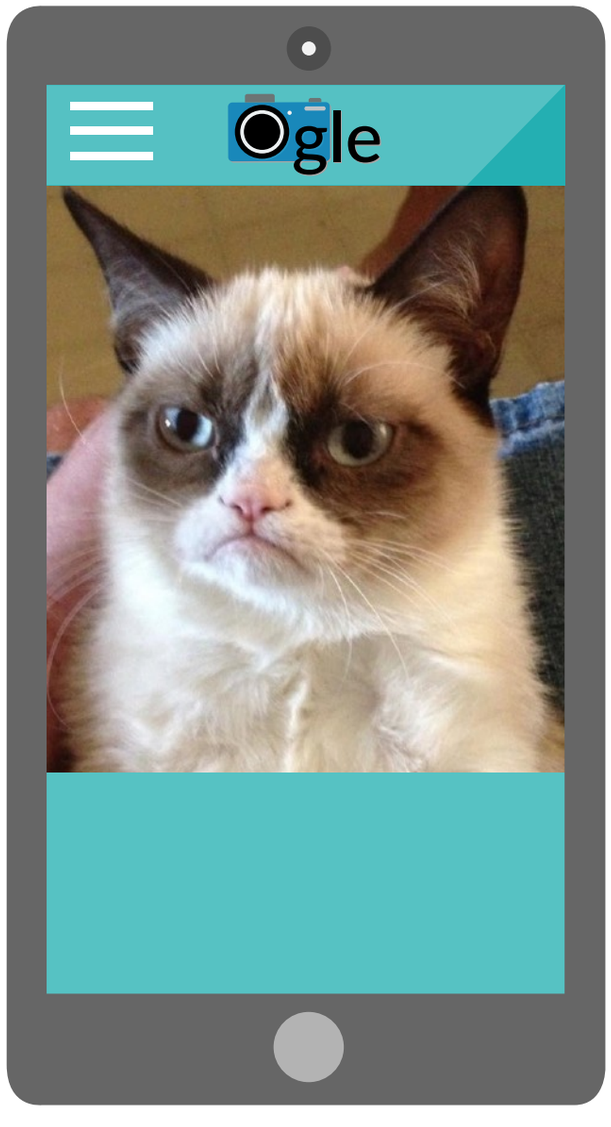


Send it to Ogle

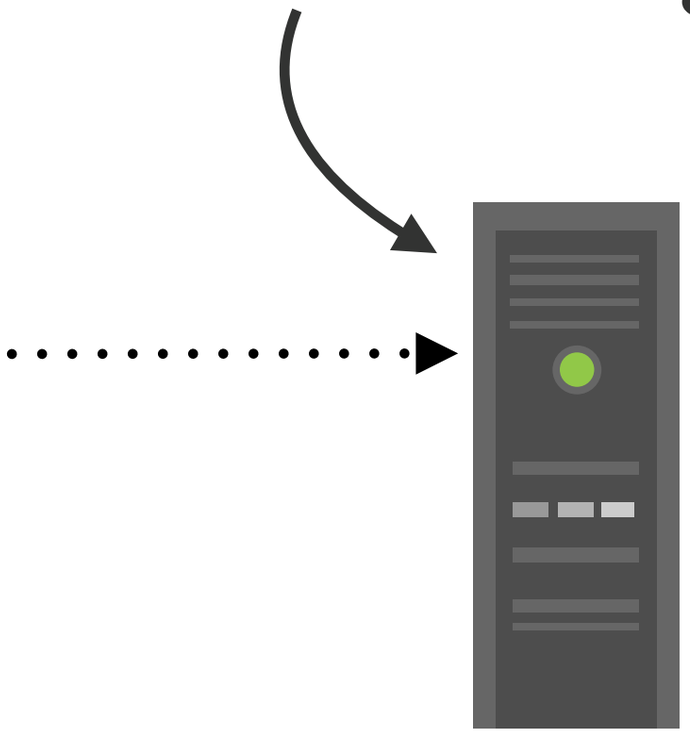


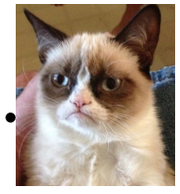
The Prototype Ogle

Take a picture

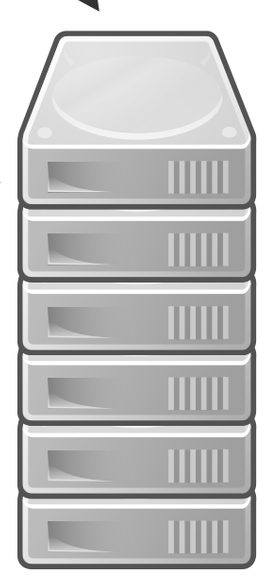


Send it to Ogle



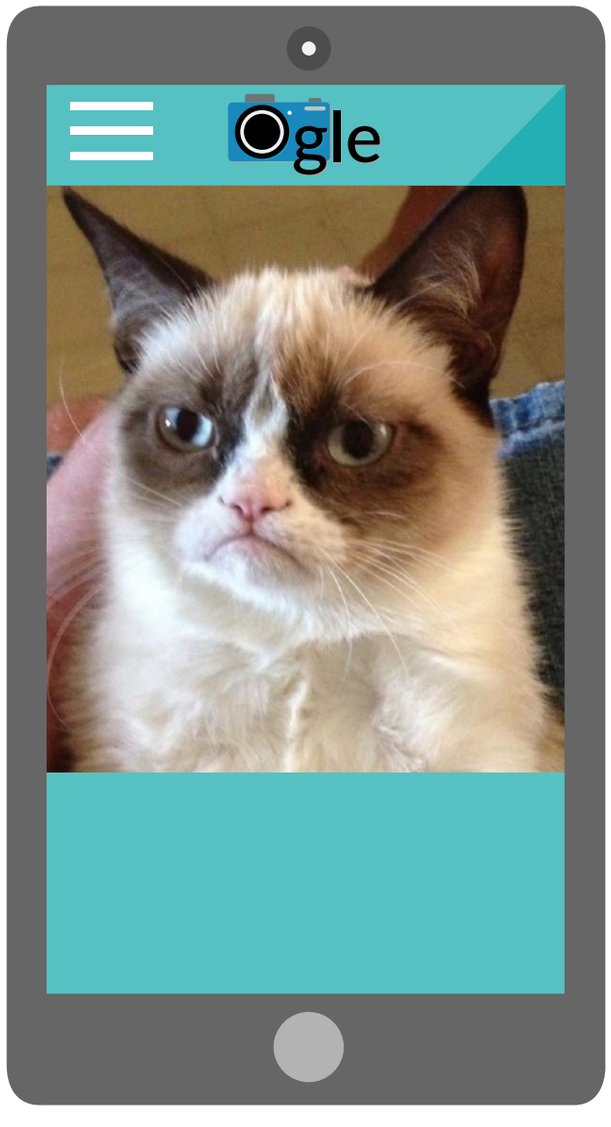


Add it to the database



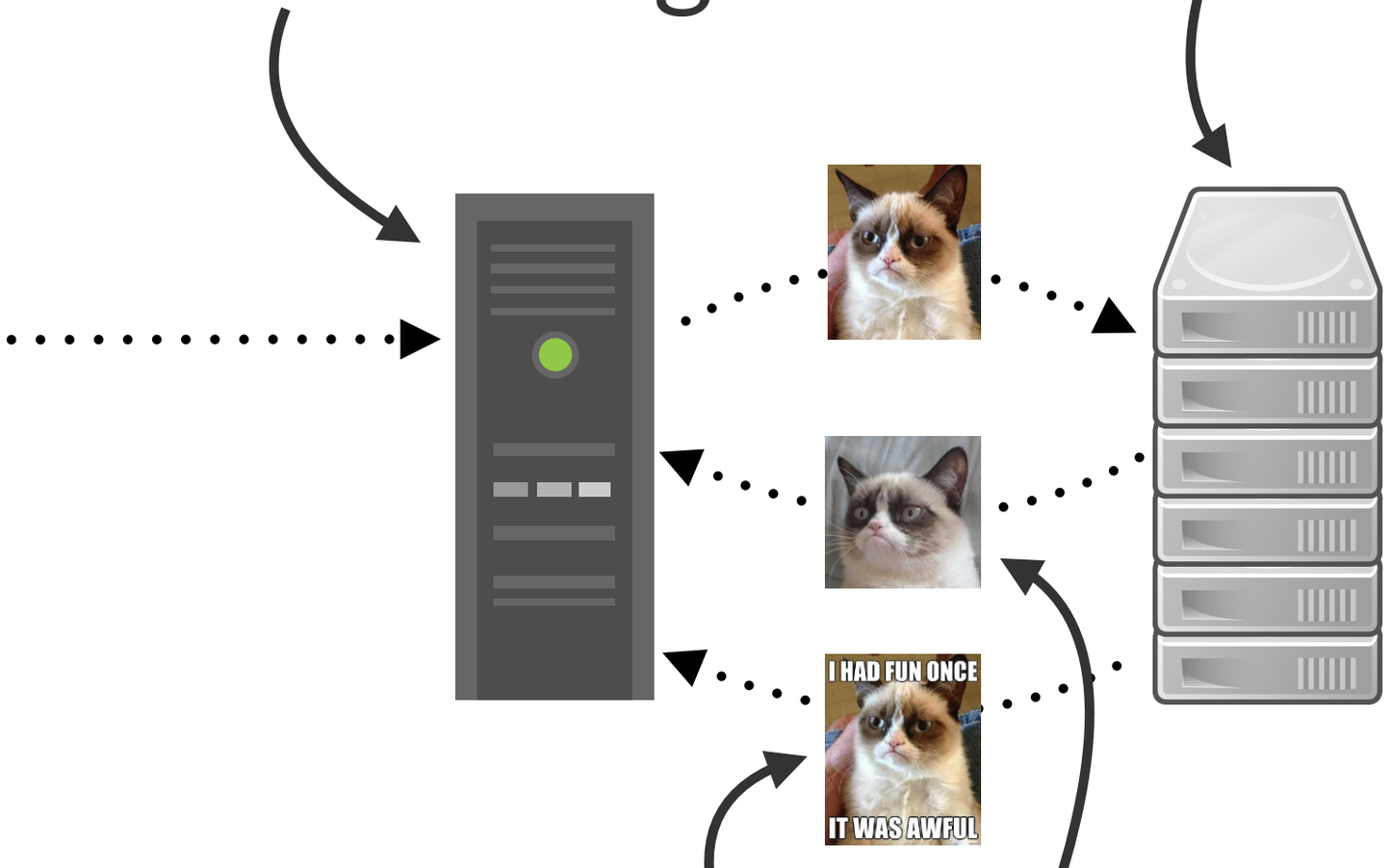
The Prototype Ogle

Take a picture



Send it to Ogle

Add it to the database



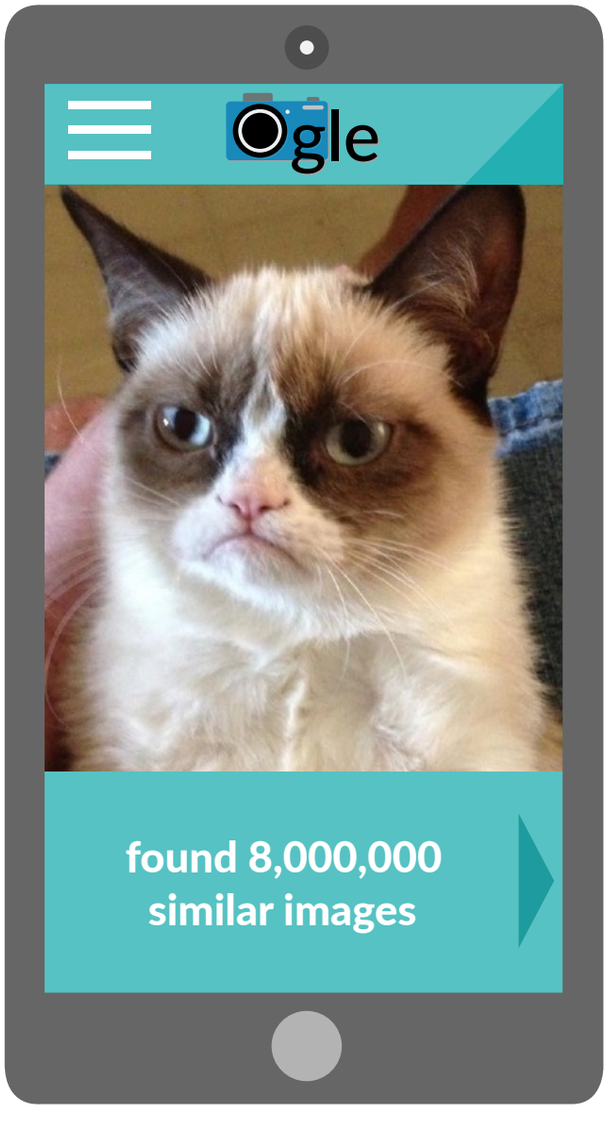
Find similar pictures

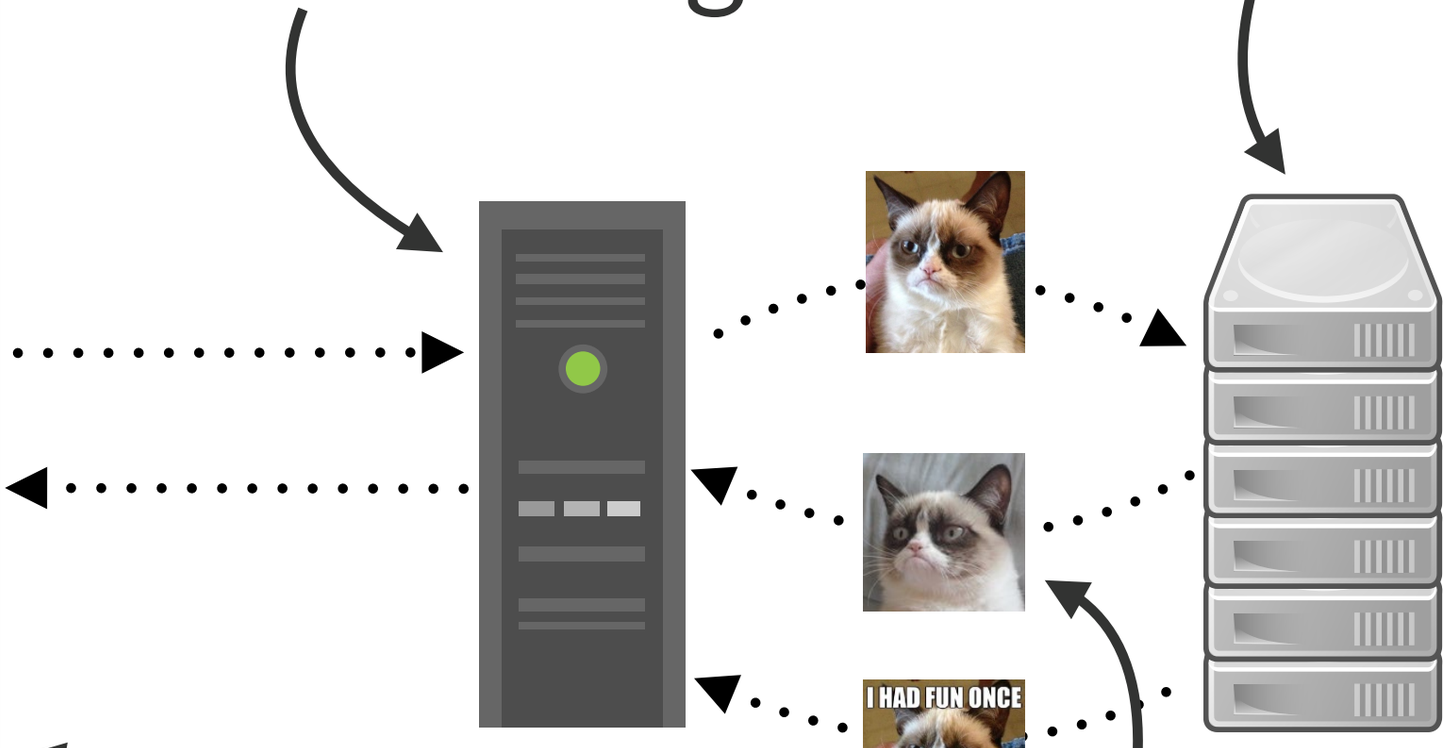
The Prototype Ogle

Take a picture

Send it to Ogle

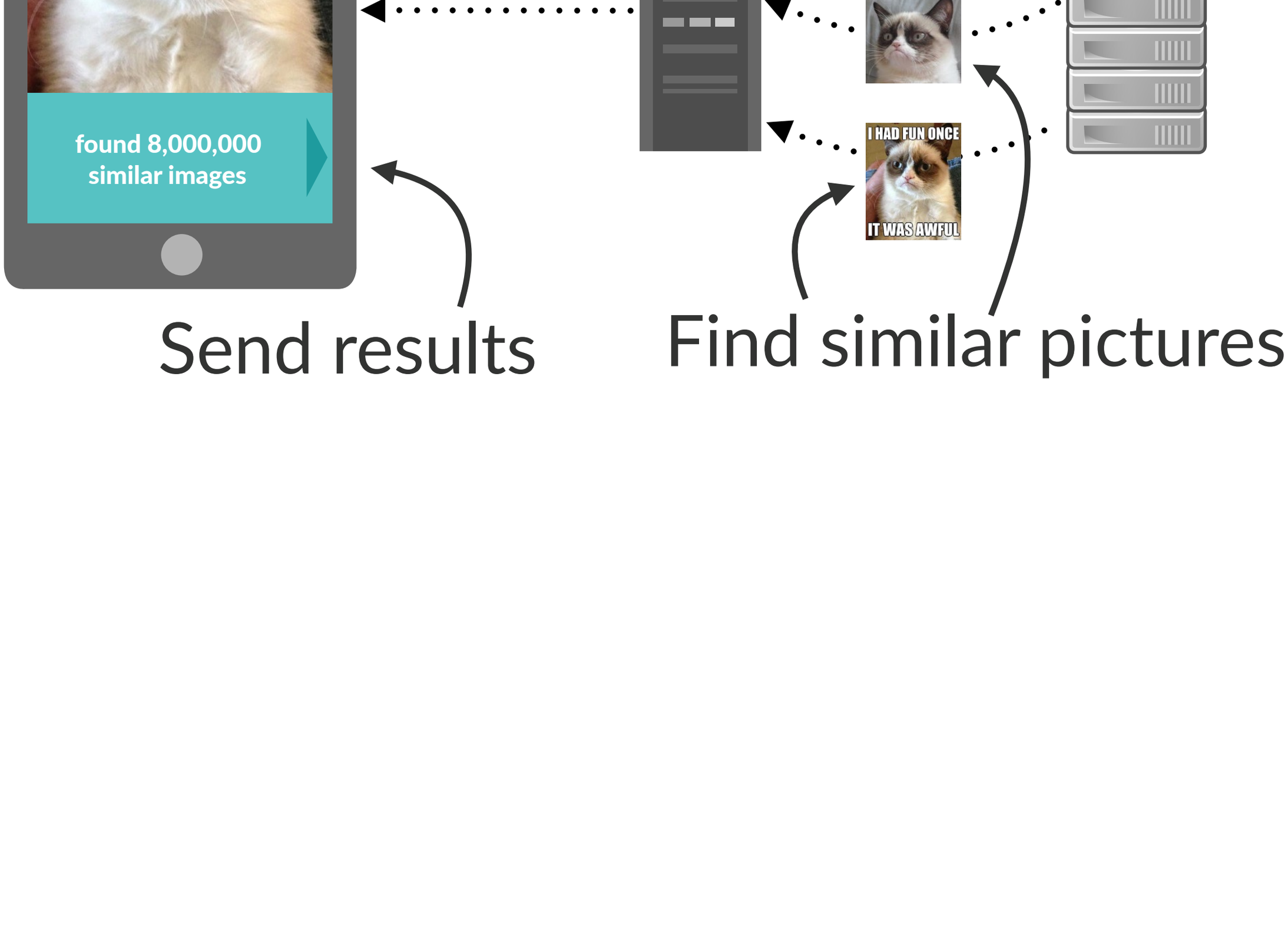
Add it to the database

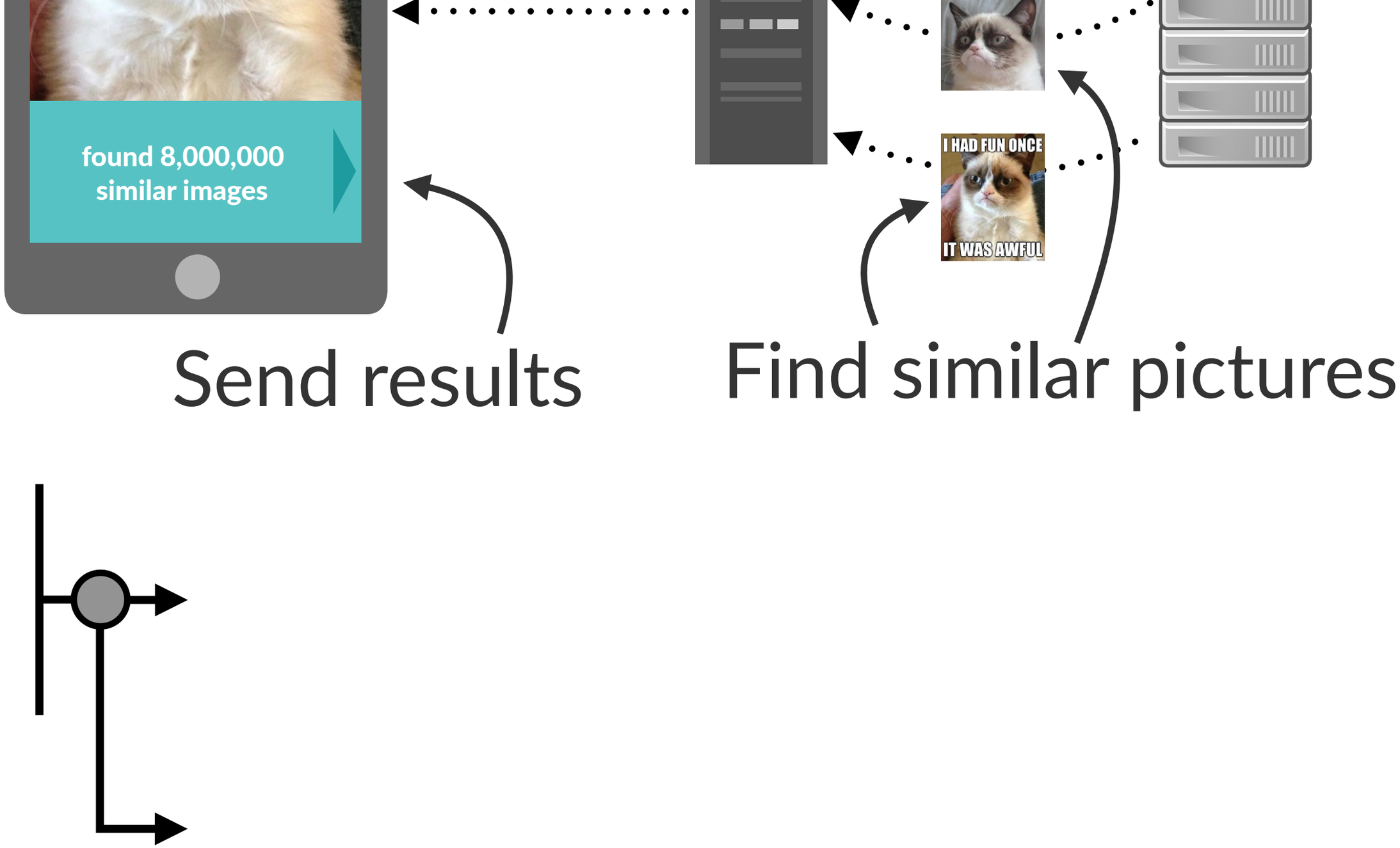




Send results

Find similar pictures


found 8,000,000
similar images
I HAD FUN ONCE
IT WAS AWFUL
Send results
Find similar pictures


found 8,000,000
similar images
I HAD FUN ONCE
IT WAS AWFUL
Send results
Find similar pictures

found 8,000,000
similar images

I HAD FUN ONCE

IT WAS AWFUL

Send results

Find similar pictures

compress

found 8,000,000
similar images

I HAD FUN ONCE

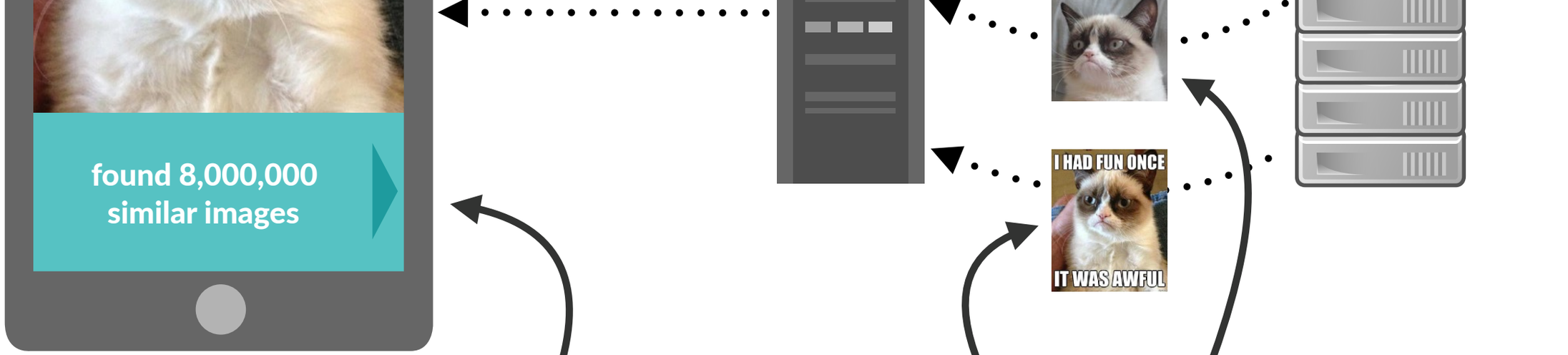
IT WAS AWFUL

Send results

Find similar pictures

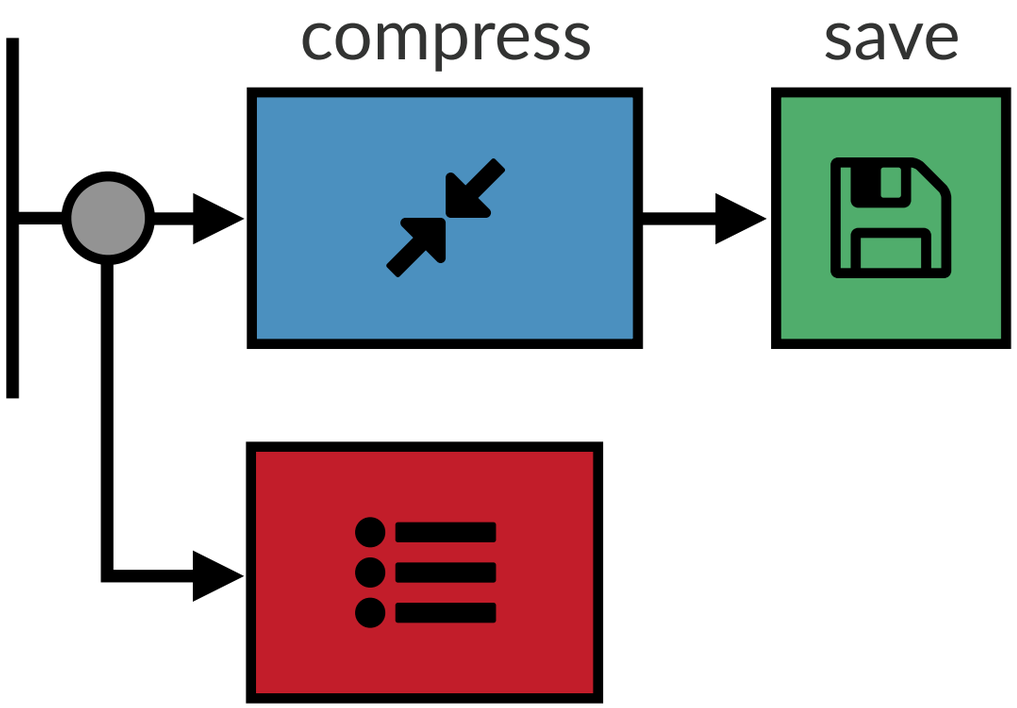
compress

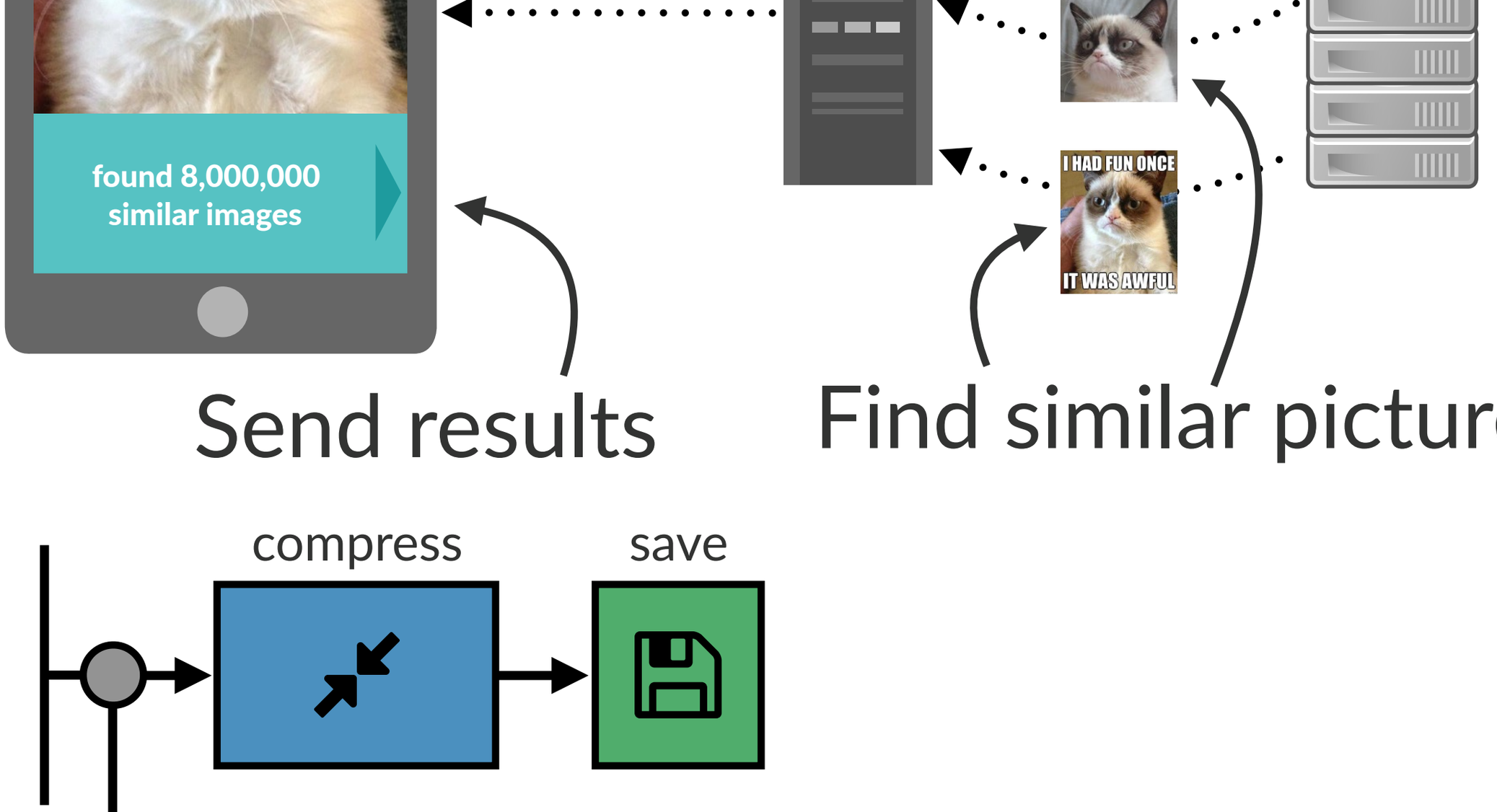
save



Send results

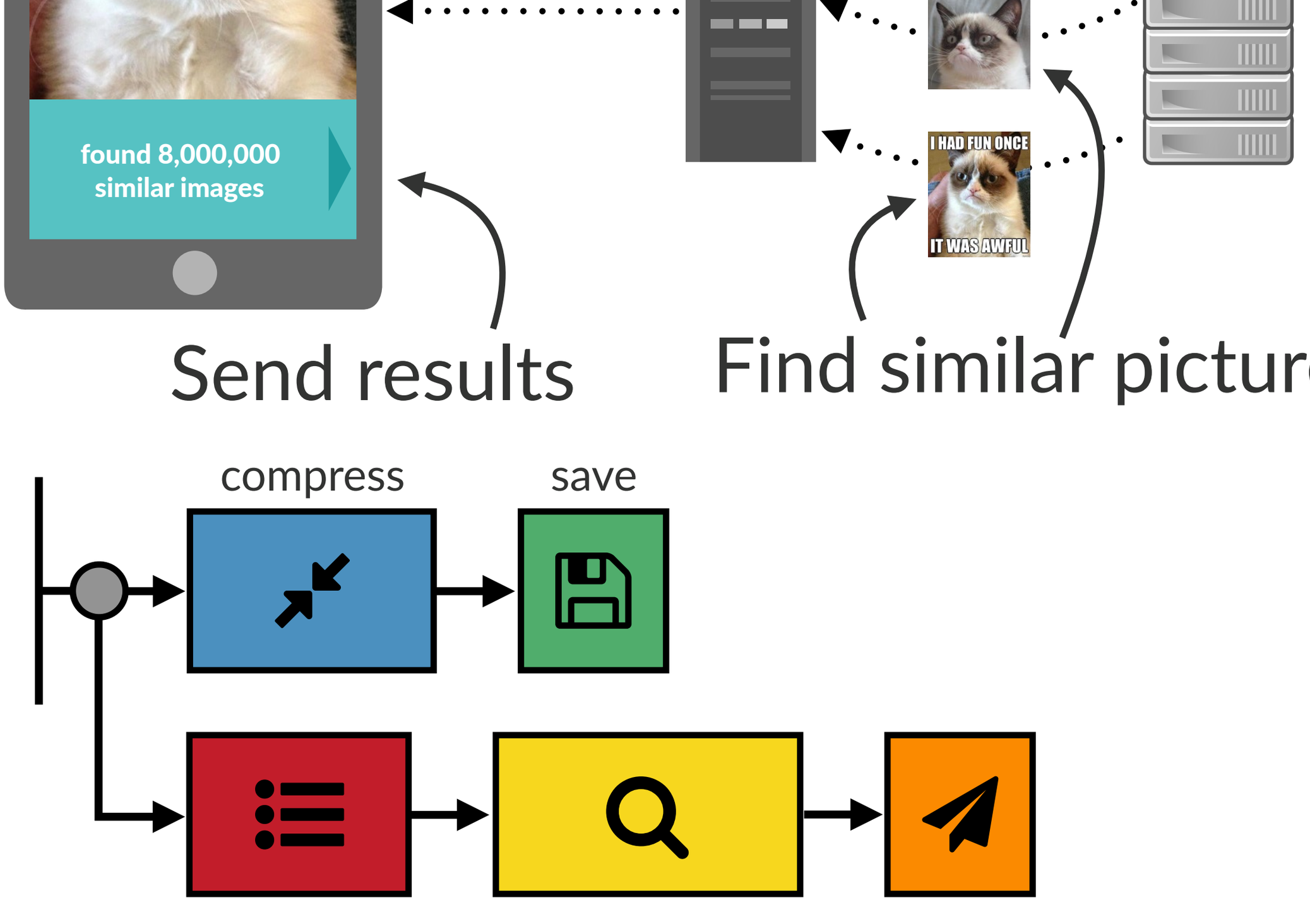
Find similar pictures





Send results

Find similar pictures


found 8,000,000
similar images
I HAD FUN ONCE
IT WAS AWFUL
Send results
Find similar pictures

compress
save
feature
extraction
search for
similar pix
send
results

found 8,000,000
similar images

I HAD FUN ONCE

IT WAS AWFUL

Send results

Find similar pictures

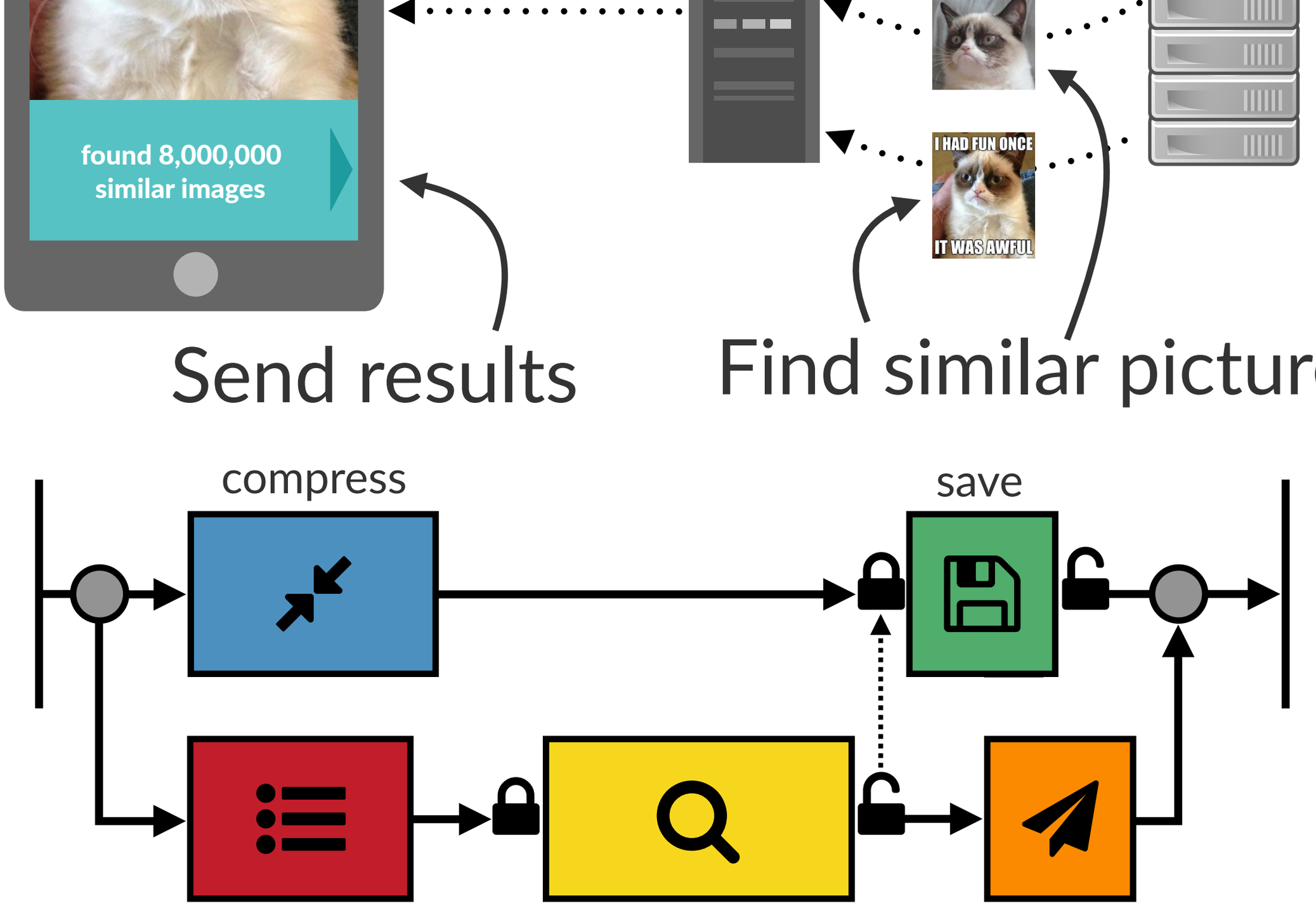
compress

save

feature
extraction

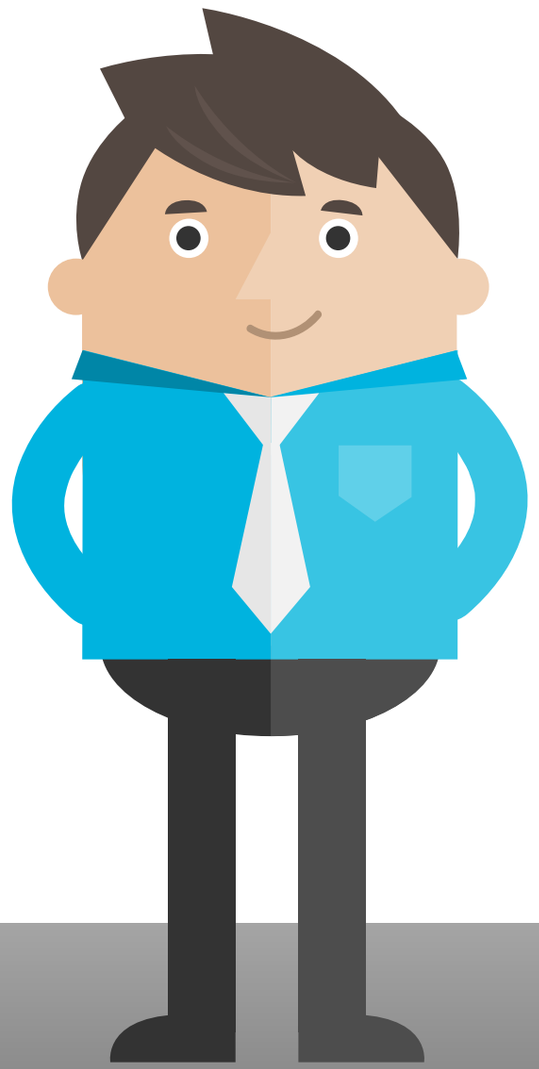
search for
similar pix

send
results

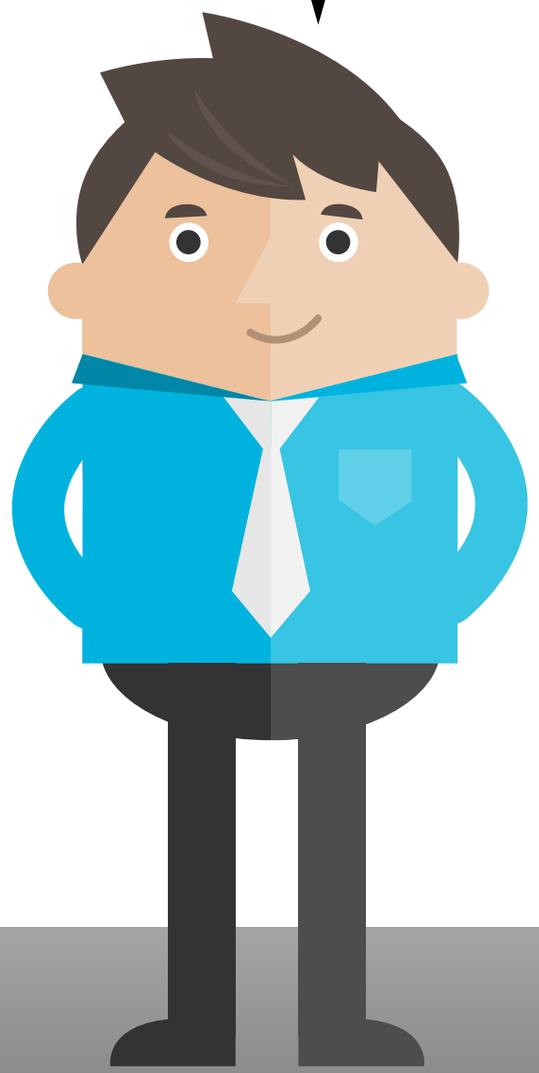

found 8,000,000
similar images
I HAD FUN ONCE
IT WAS AWFUL
Send results
Find similar pictures

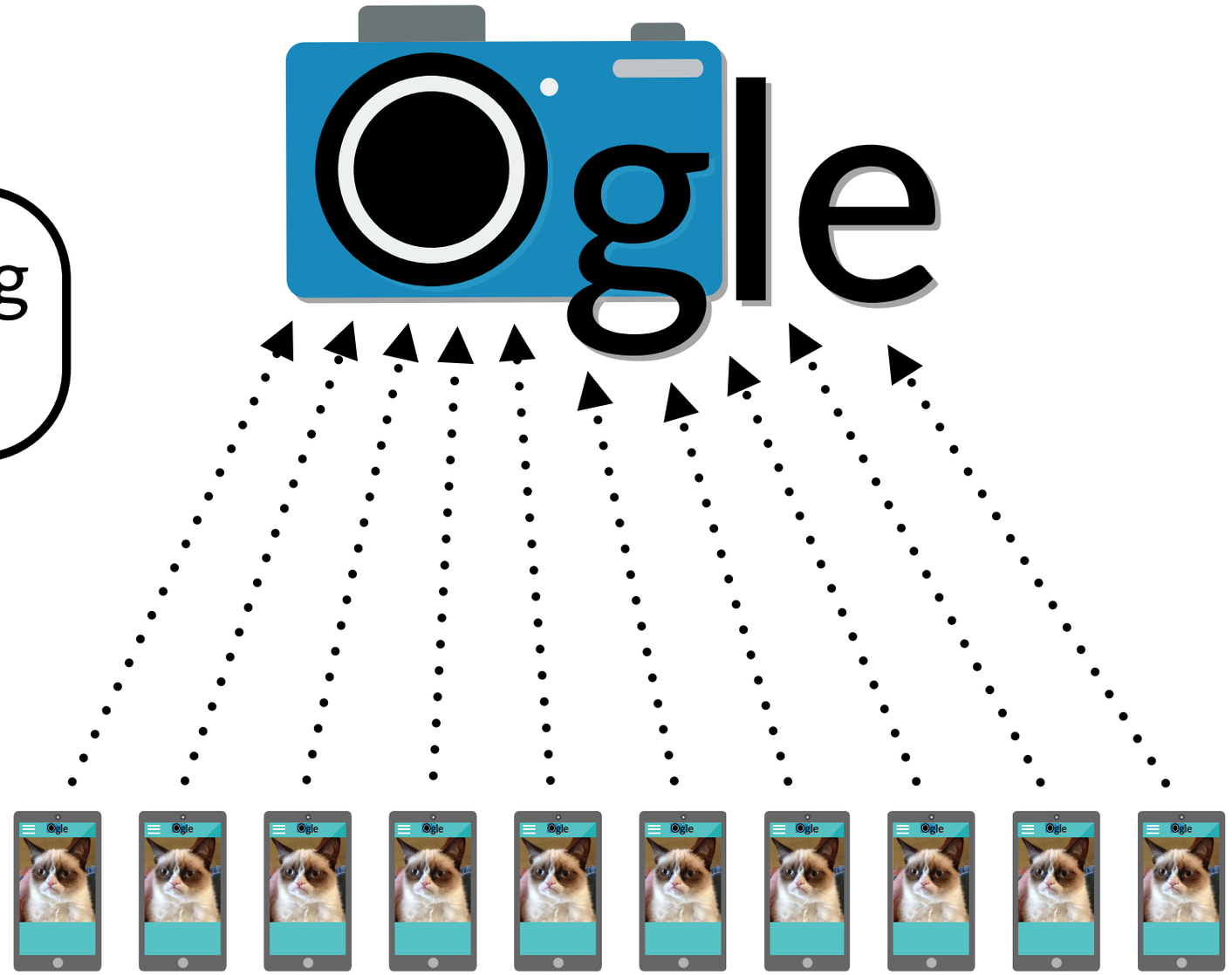
compress
save
feature
extraction
search for
similar pix
send
results

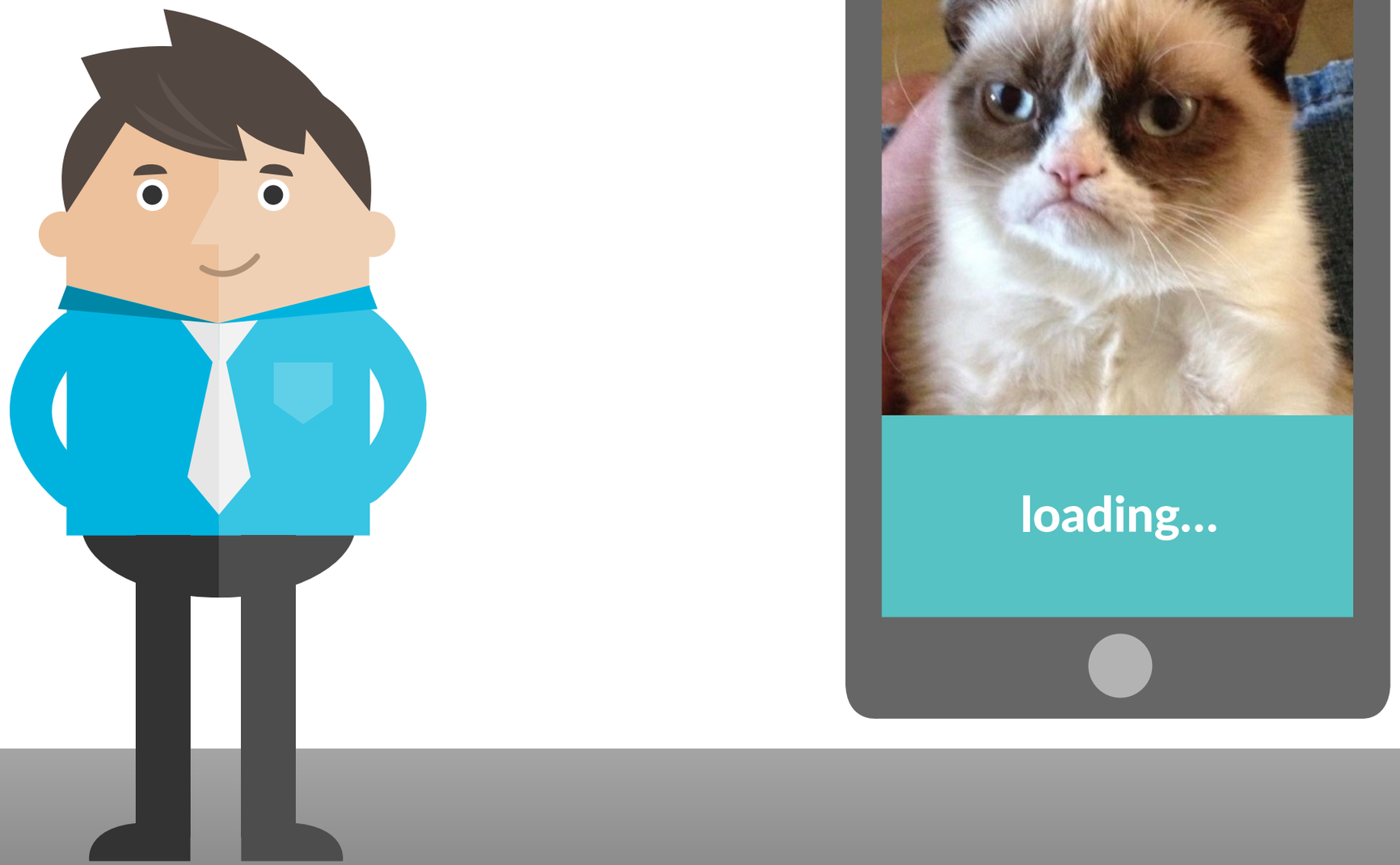
Ogle



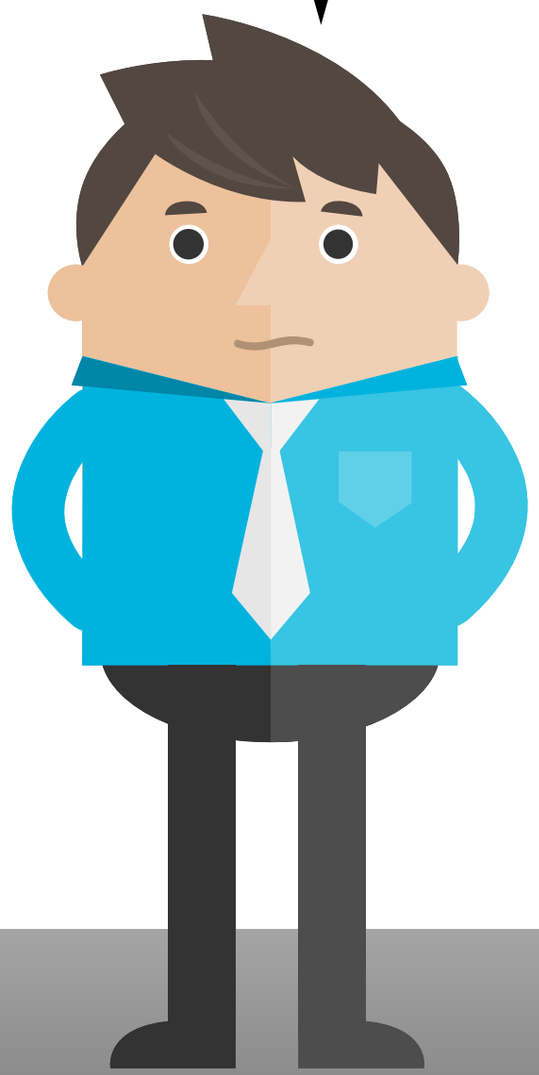
Ogle is totally disrupting image search.

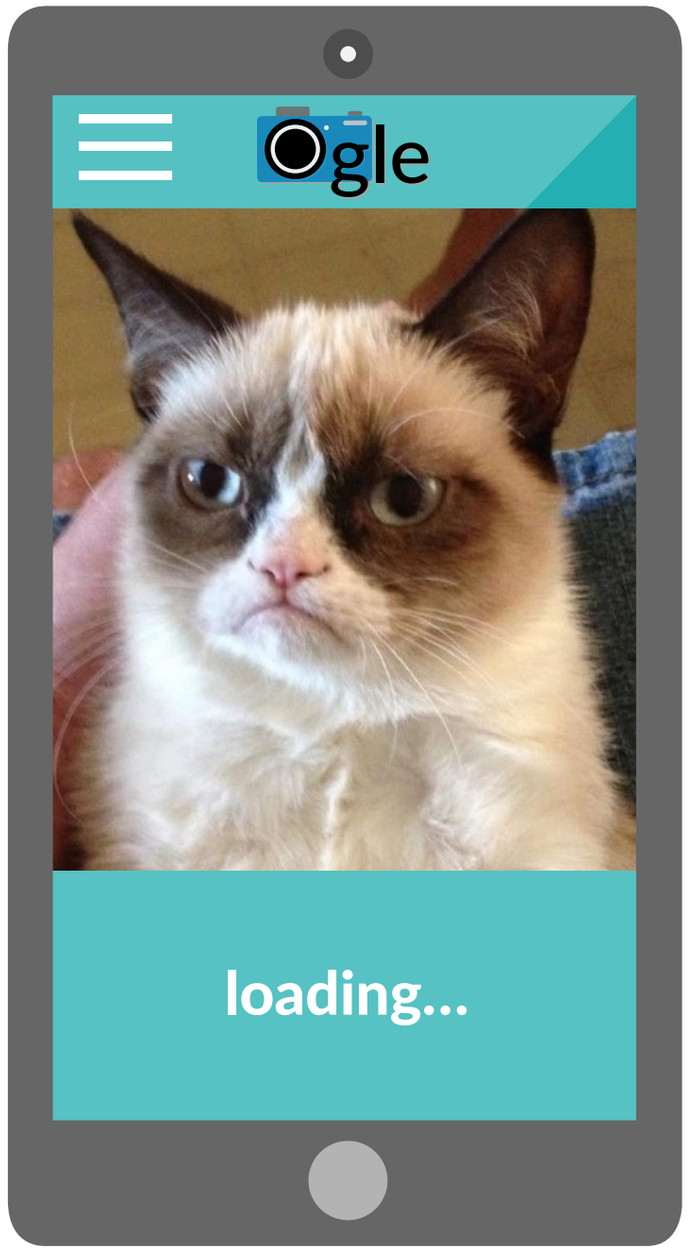


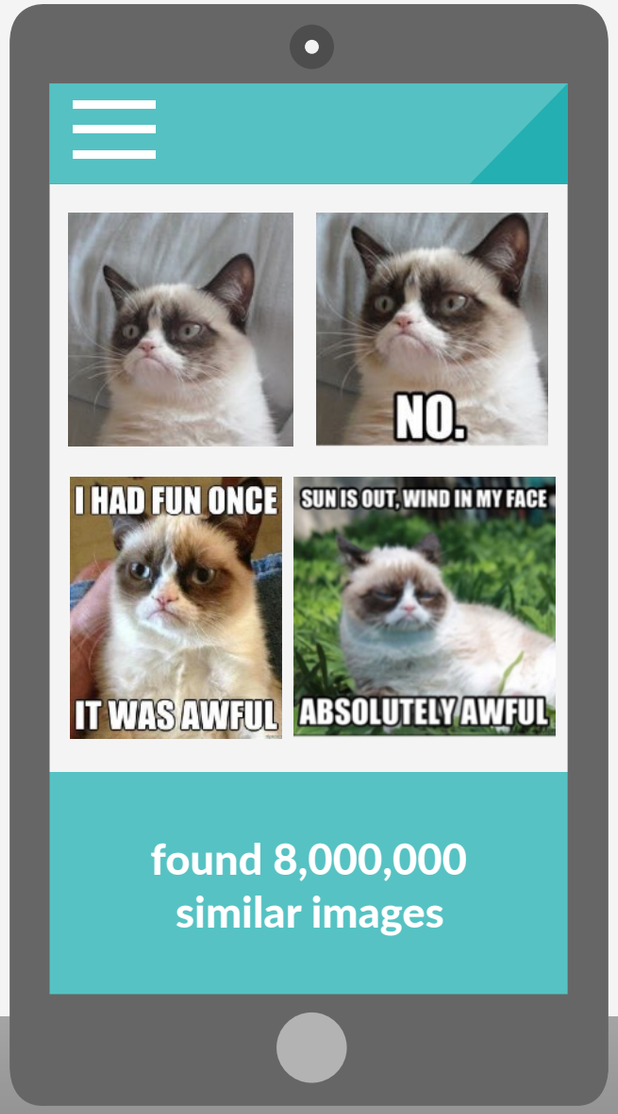



gle
loading...

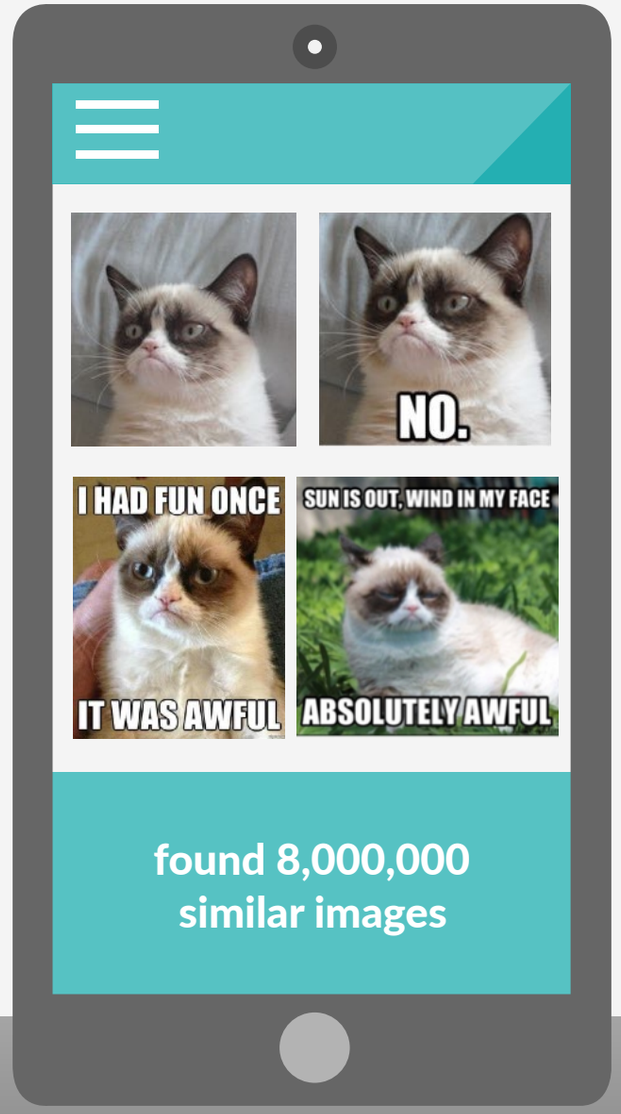
Ogle is too slow!

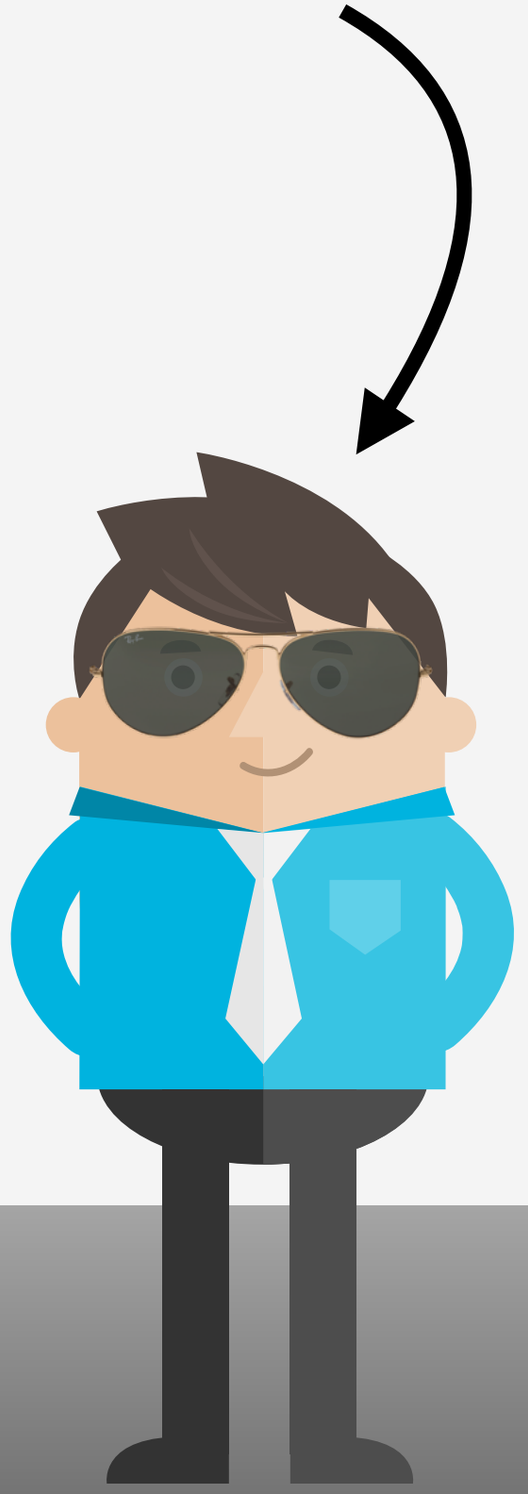




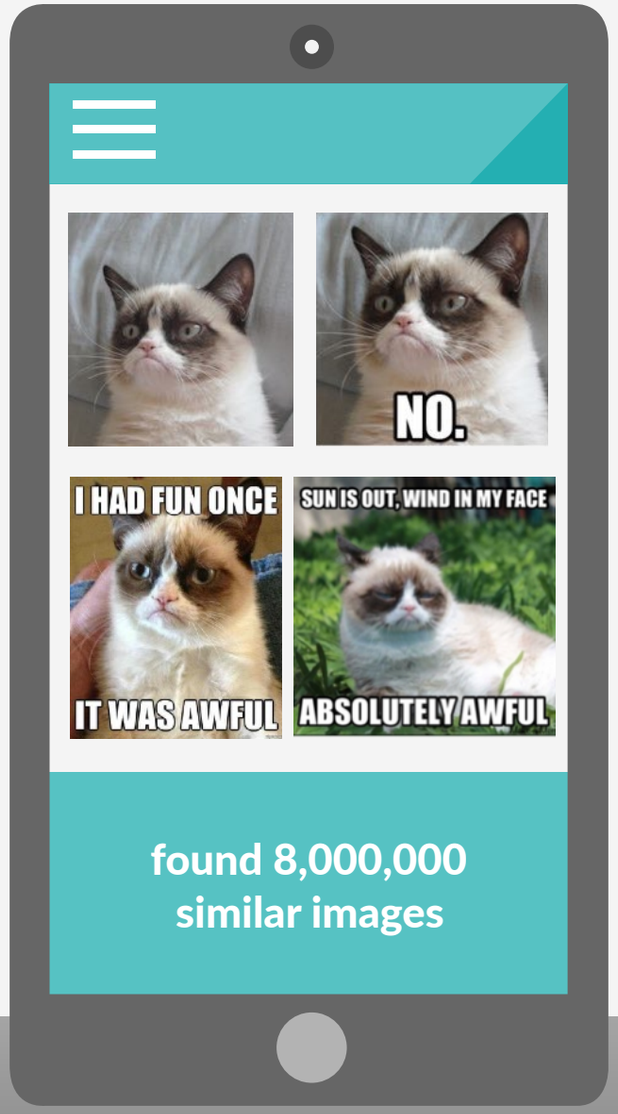

NO.
I HAD FUN ONCE
SUN IS OUT, WIND IN MY FACE
IT WAS AWFUL
ABSOLUTELY AWFUL
found 8,000,000
similar images

This is also Bob.



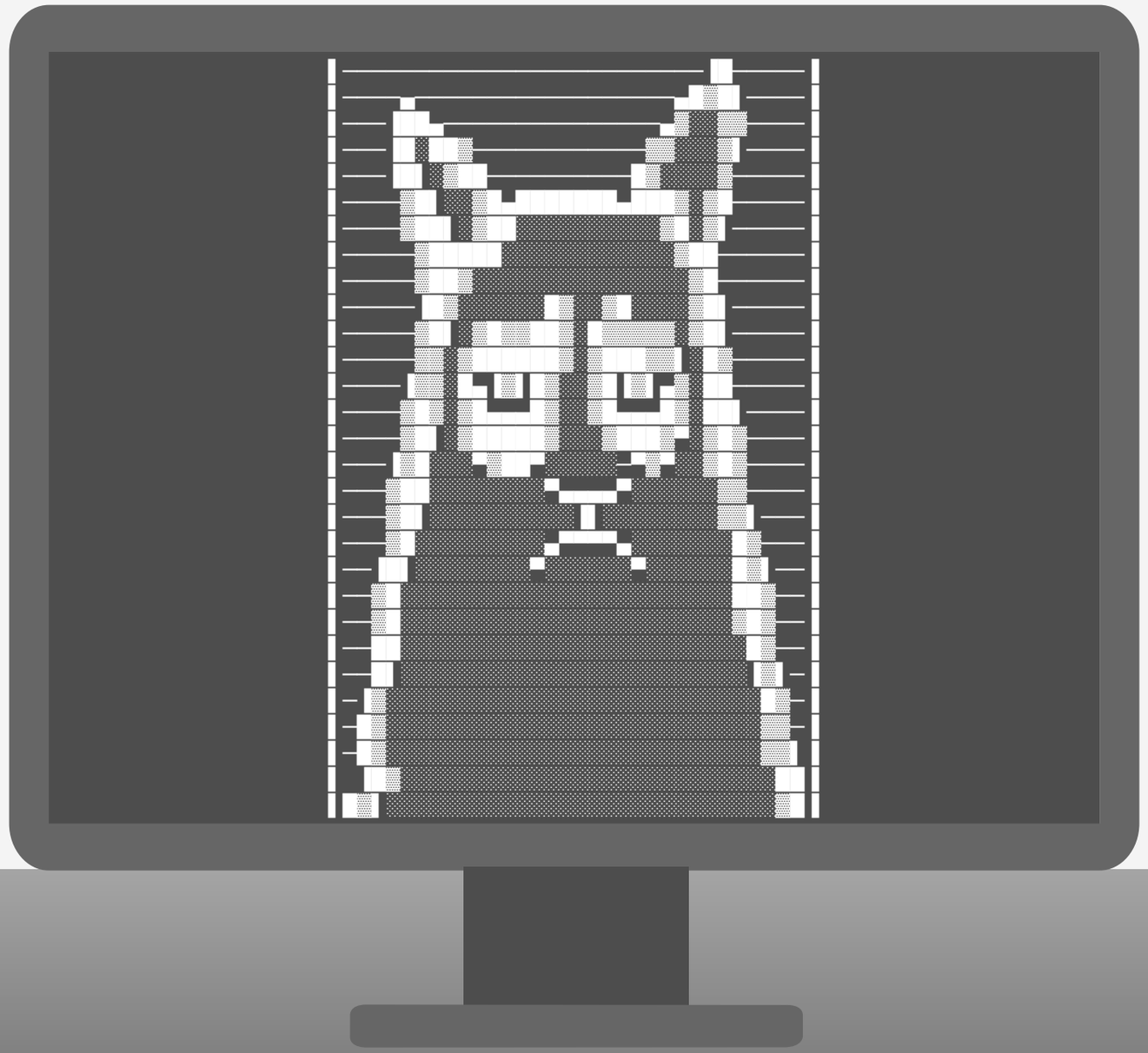


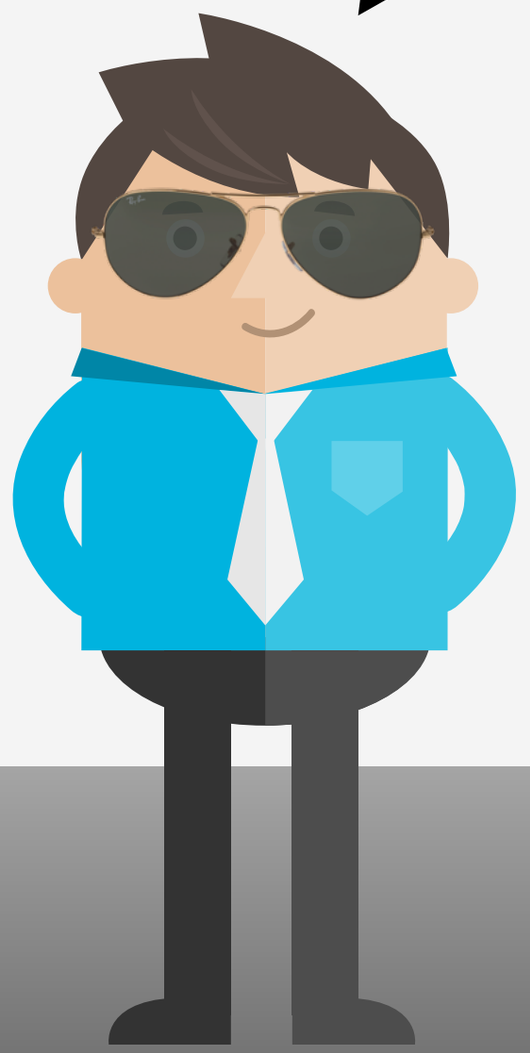
This is also Bob.
in the ‘80s

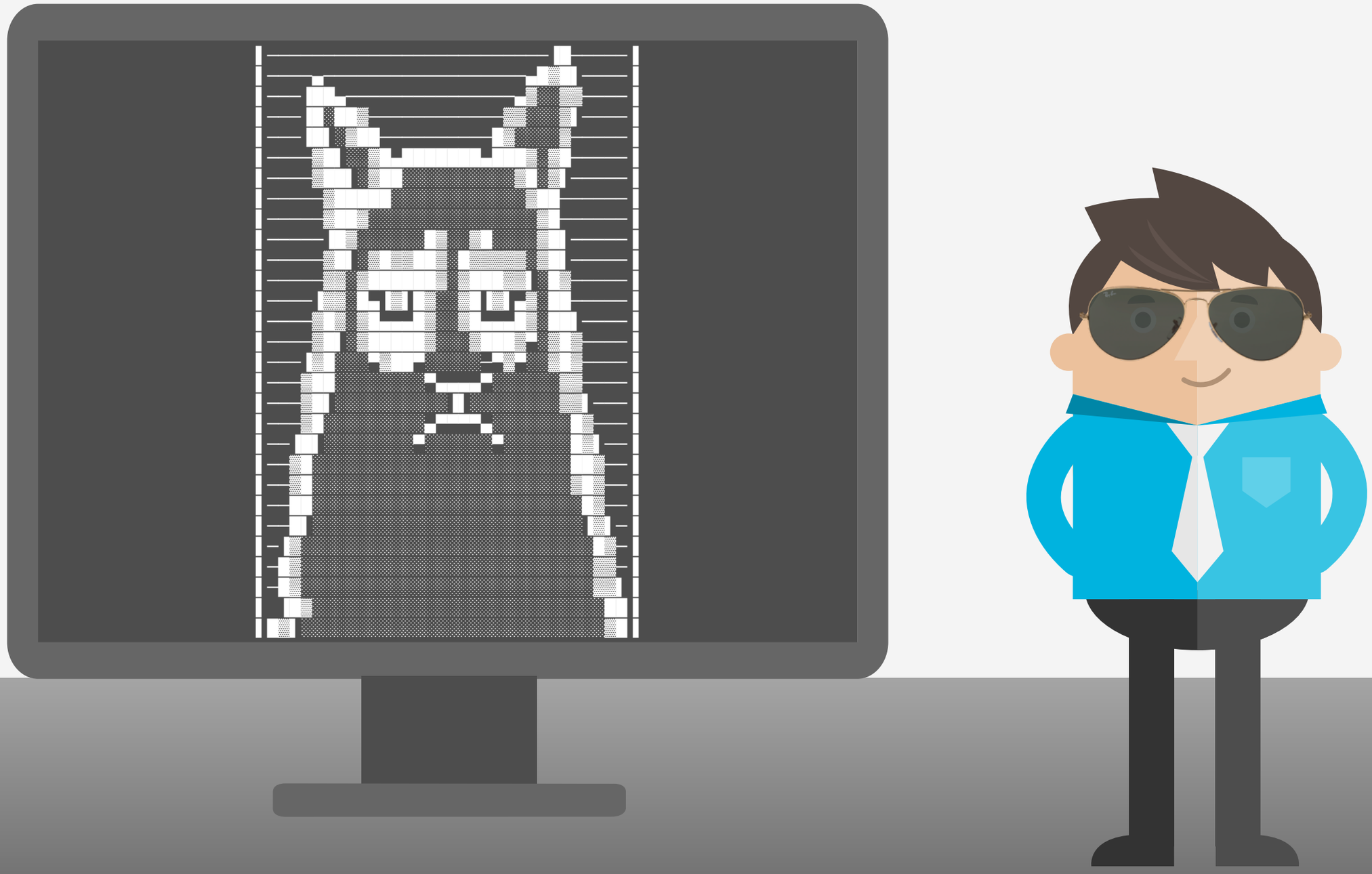


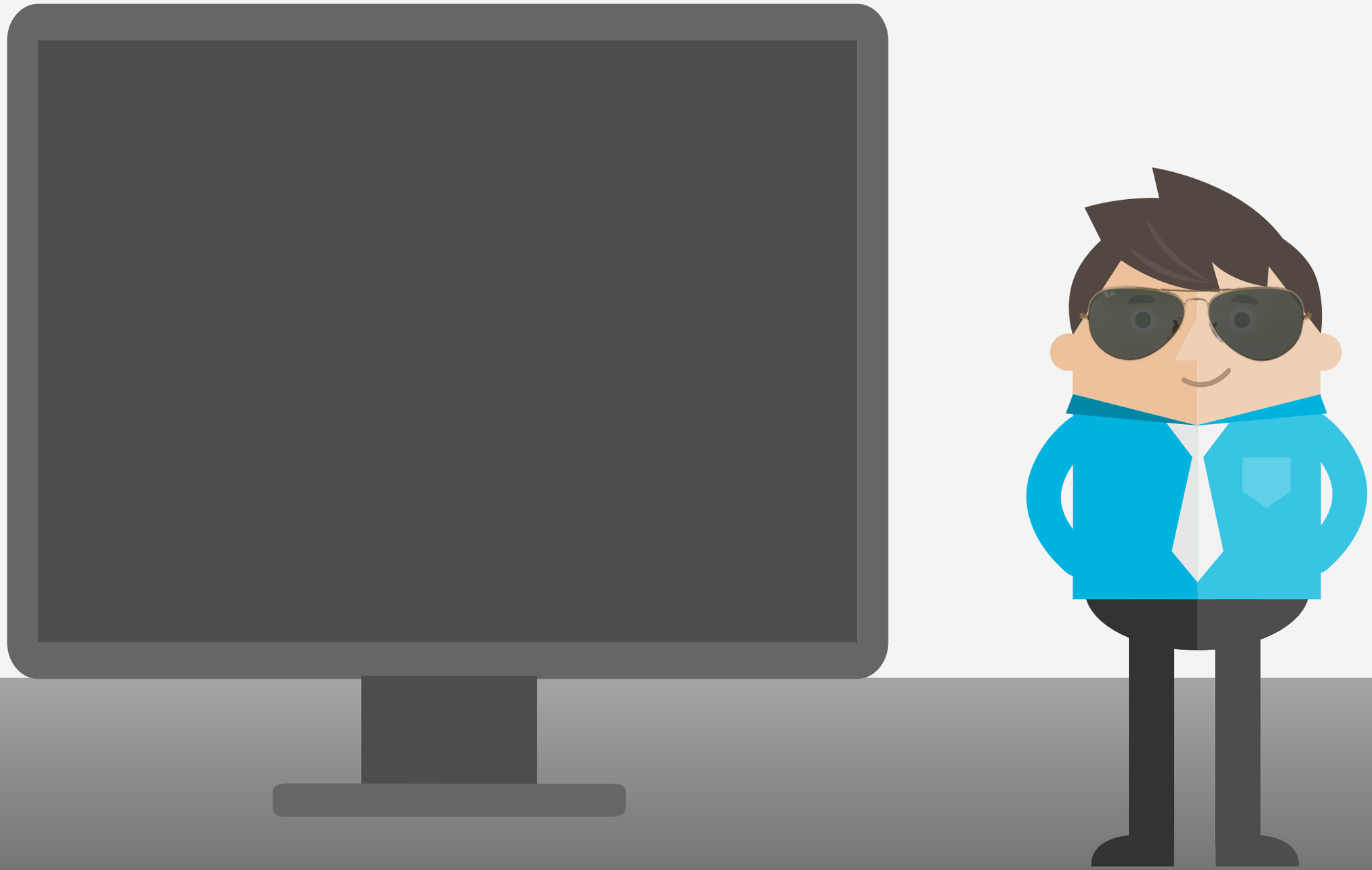
This is also Bob.
in the ‘80s

This is also Bob.
in the ‘80s

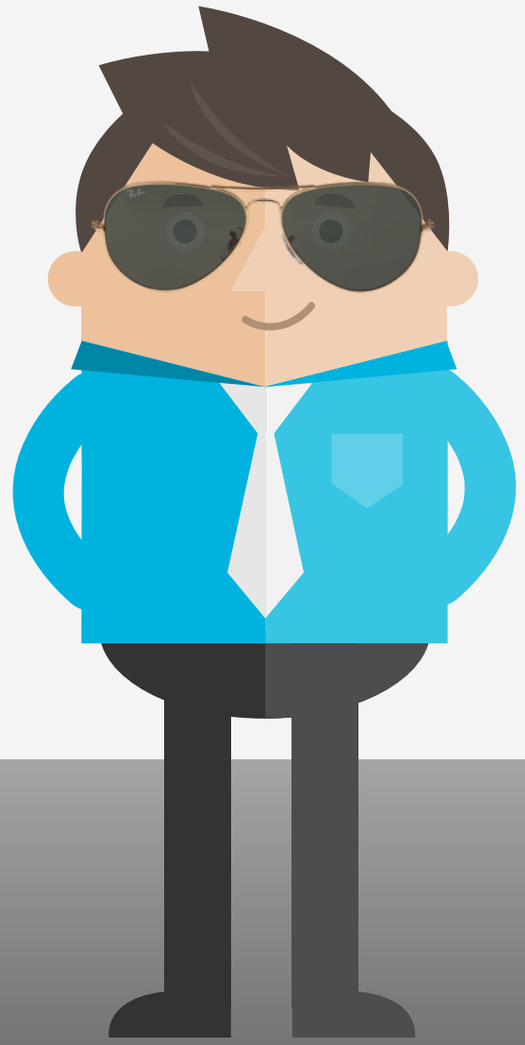




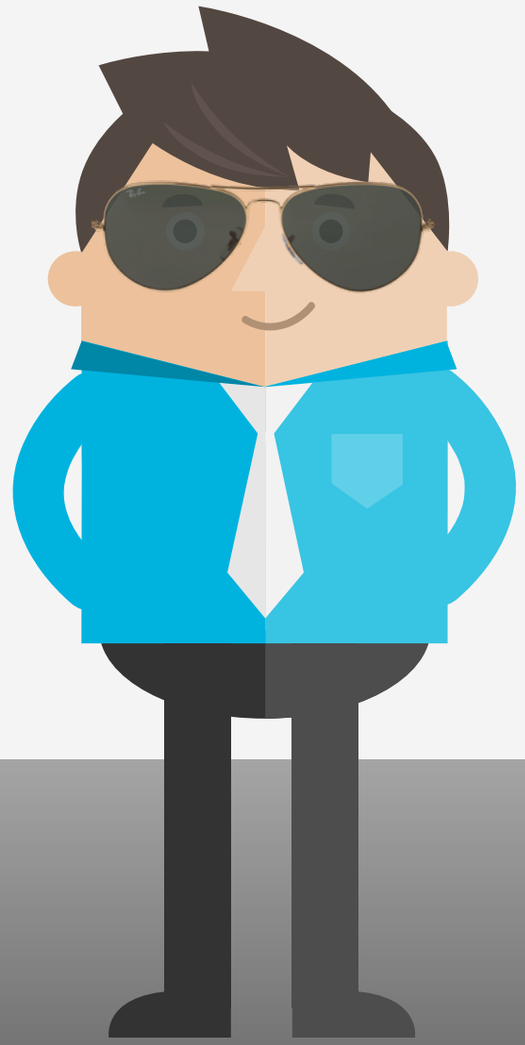




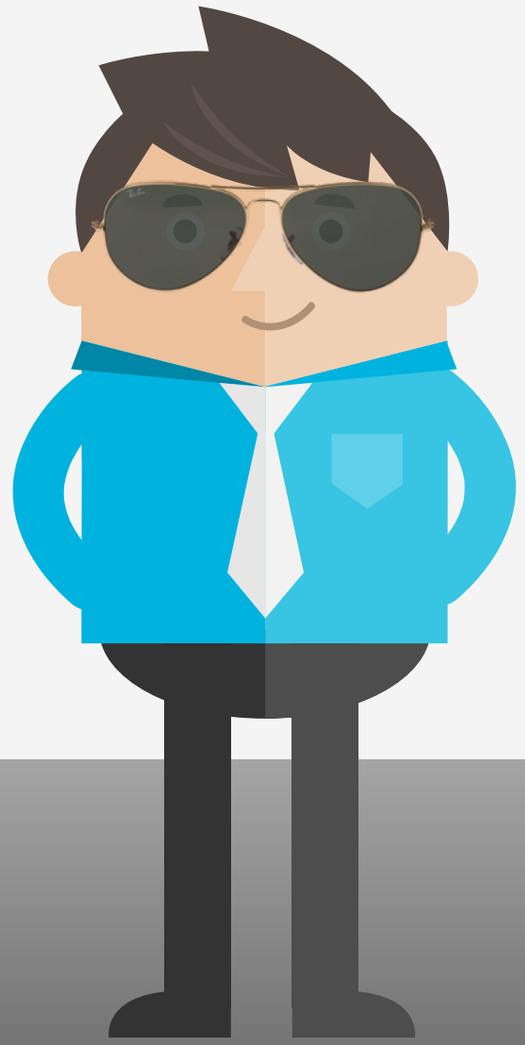
Searching database...



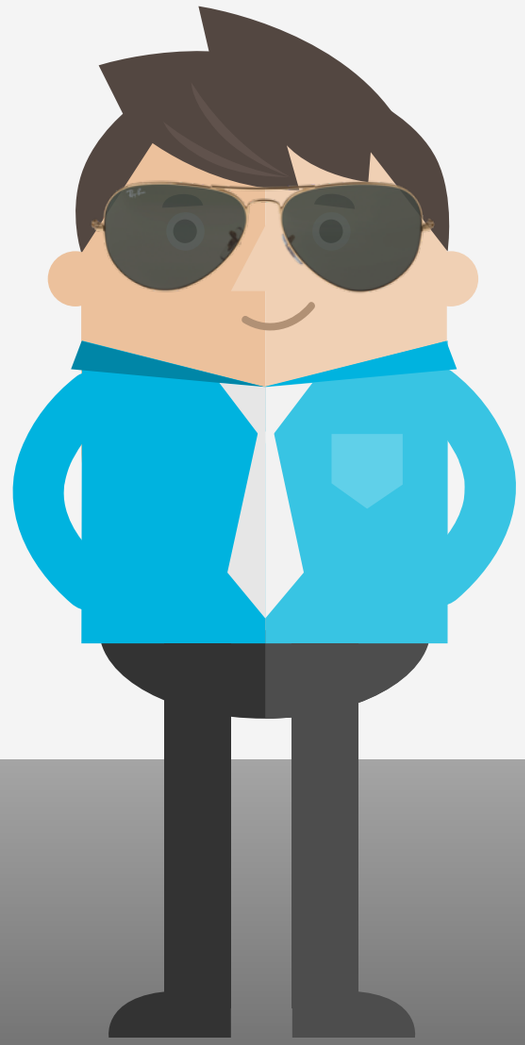
```
Searching database...

found 4 results:
```



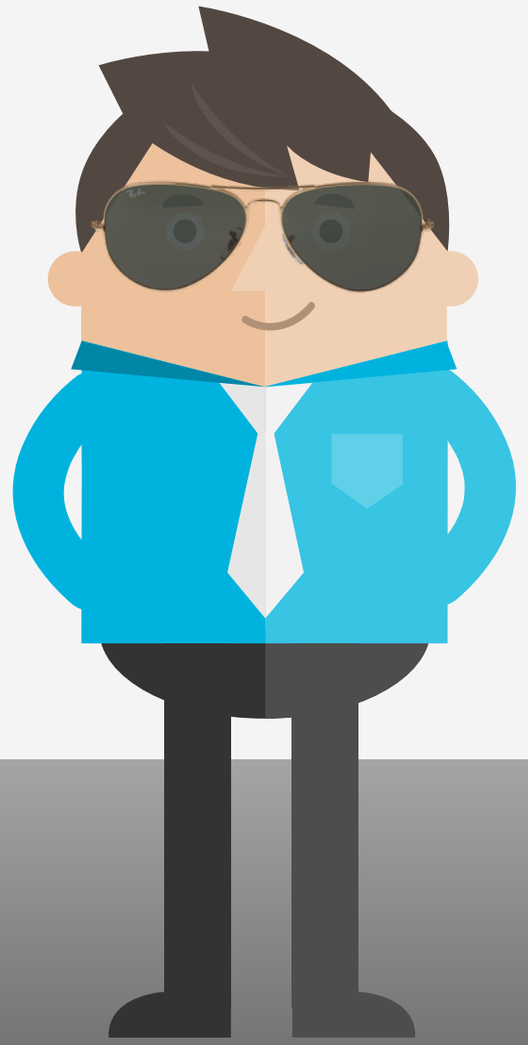
```
Searching database...

found 4 results:
catisgumpy.bmp
```



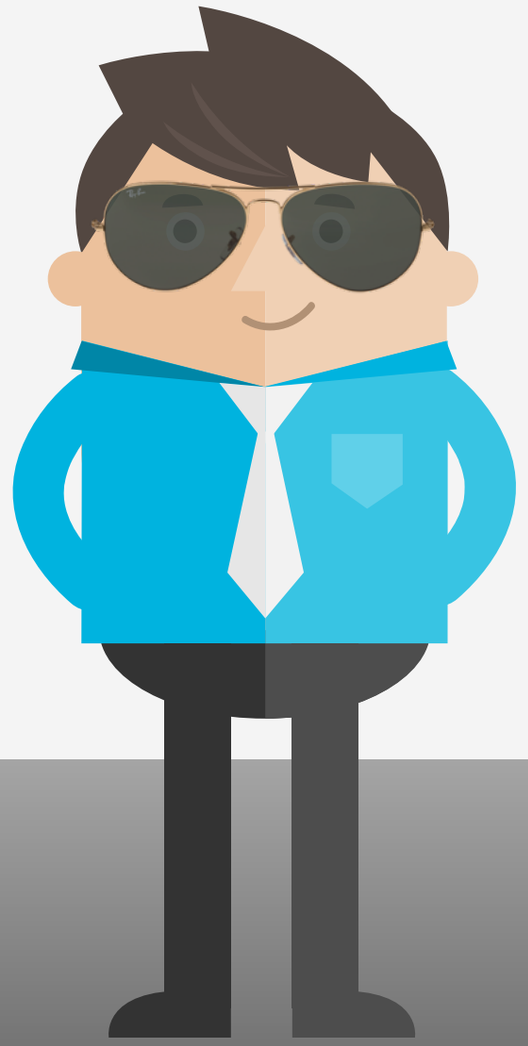
```
Searching database...

found 4 results:
catisgumpy.bmp
funisawful.bmp
```



```
Searching database...

found 4 results:
catisgumpy.bmp
funisawful.bmp
catdoesnotlikefun.bmp
```

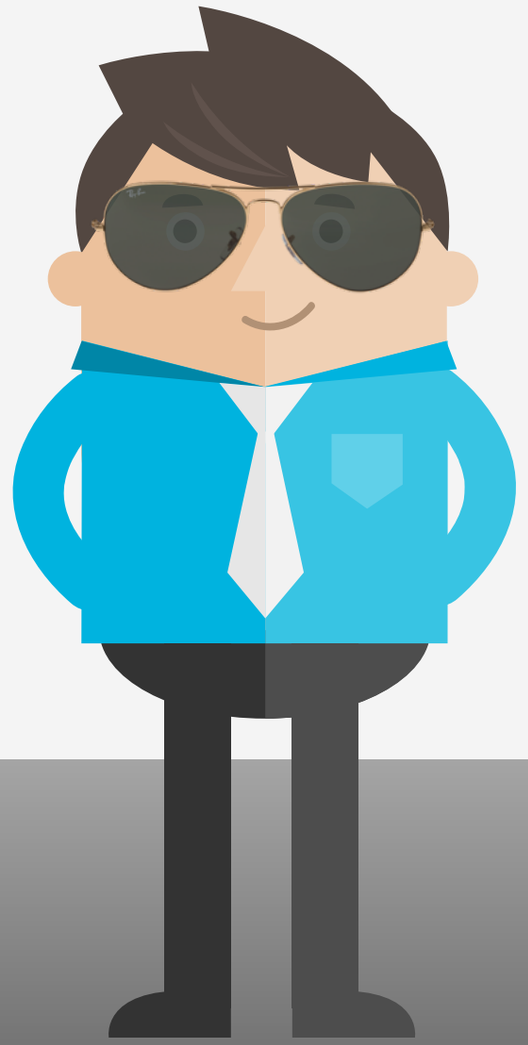


```
Searching database...

found 4 results:
catisgumpy.bmp
funisawful.bmp
catdoesnotlikefun.bmp
from_dad.bmp
```



GRAPHOSEARCH

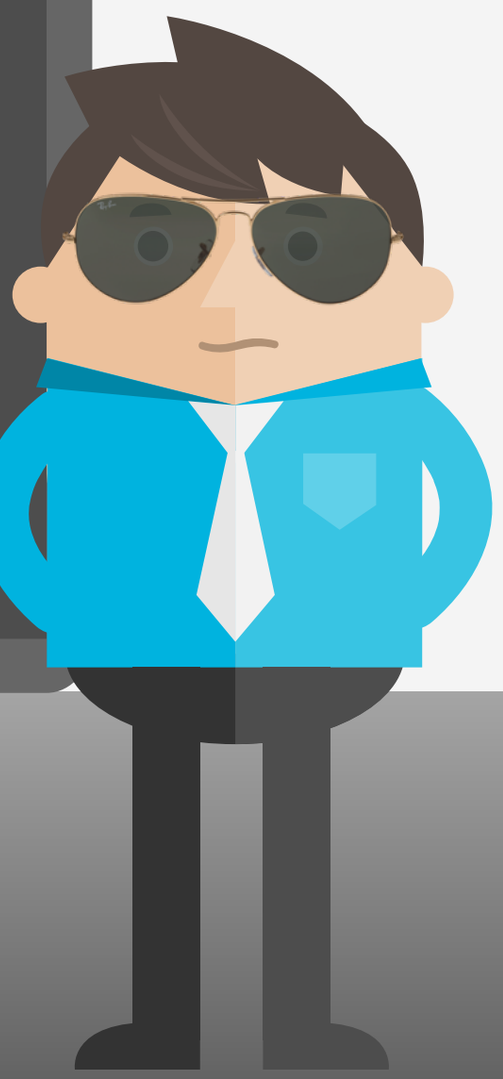
```
Searching database...

found 4 results:
catisgumpy.bmp
funisawful.bmp
catdoesnotlikefun.bmp
from_dad.bmp
```



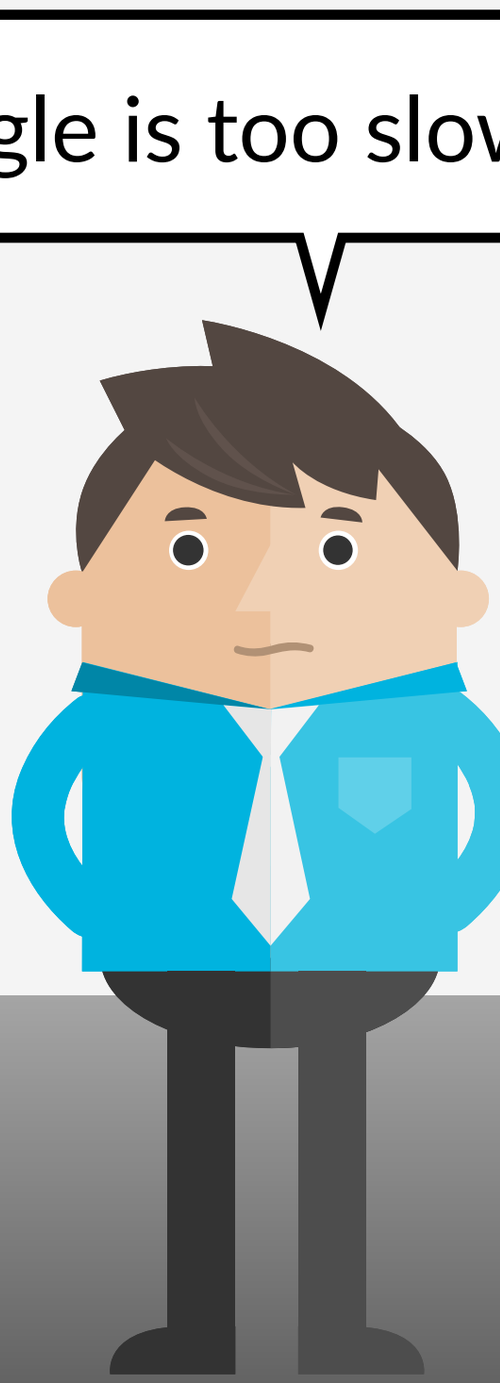
ase . . .

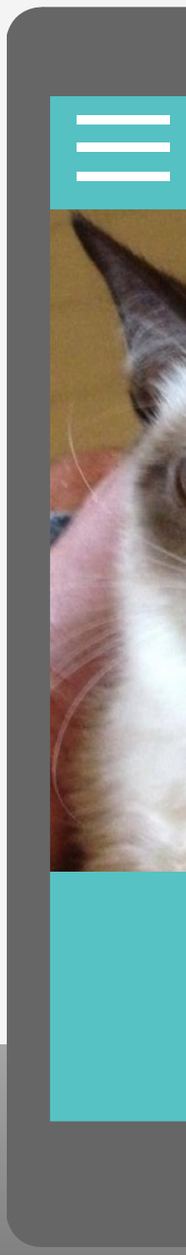
:

un . bmp



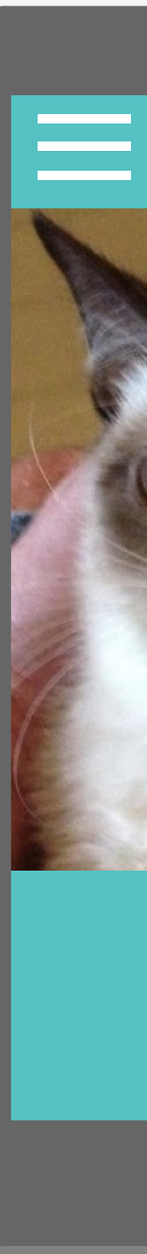
Ogle is too slow!





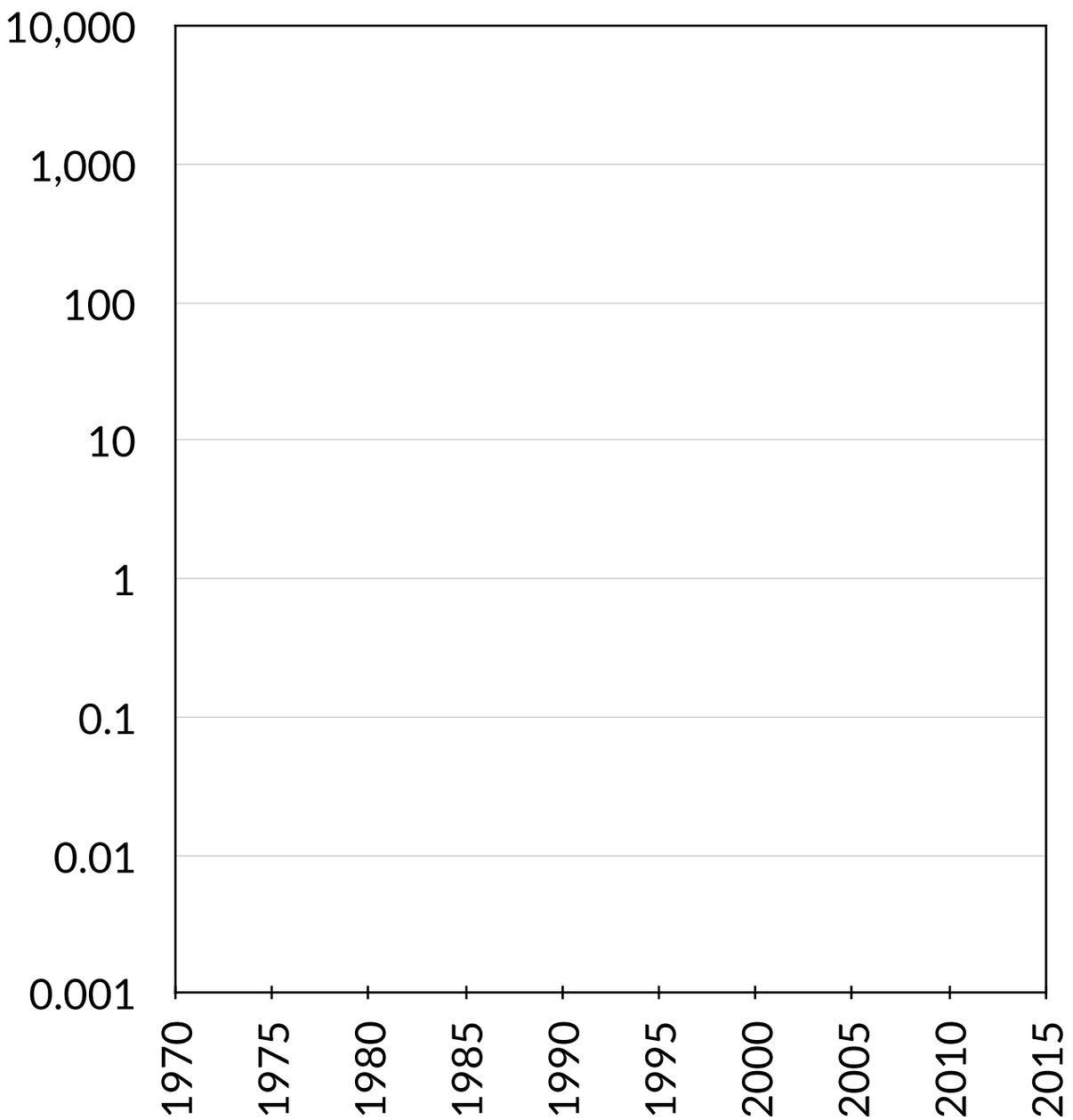
Graphosearch is too slow!

Ogle is too slow!

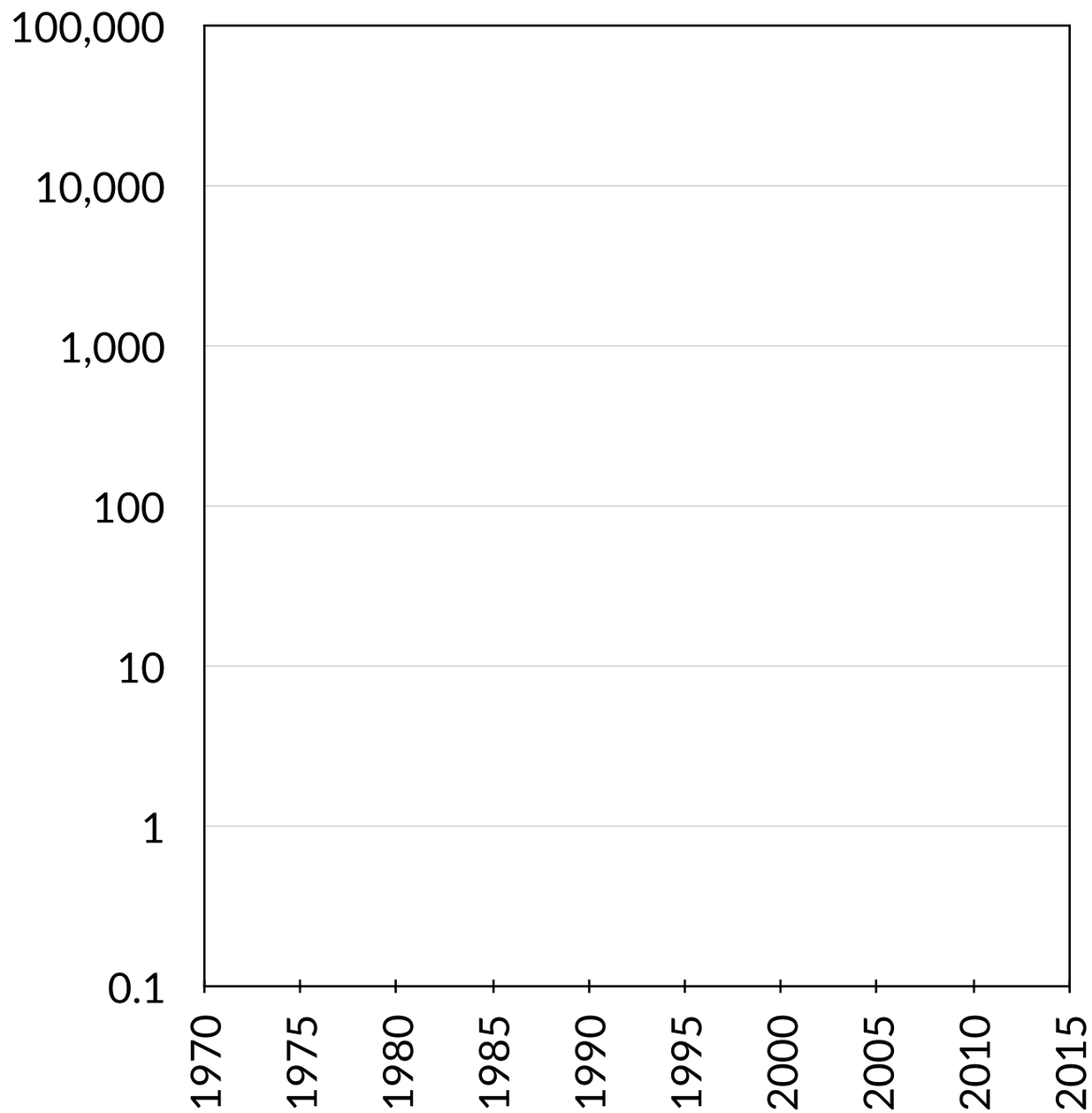


Performance used to be easy

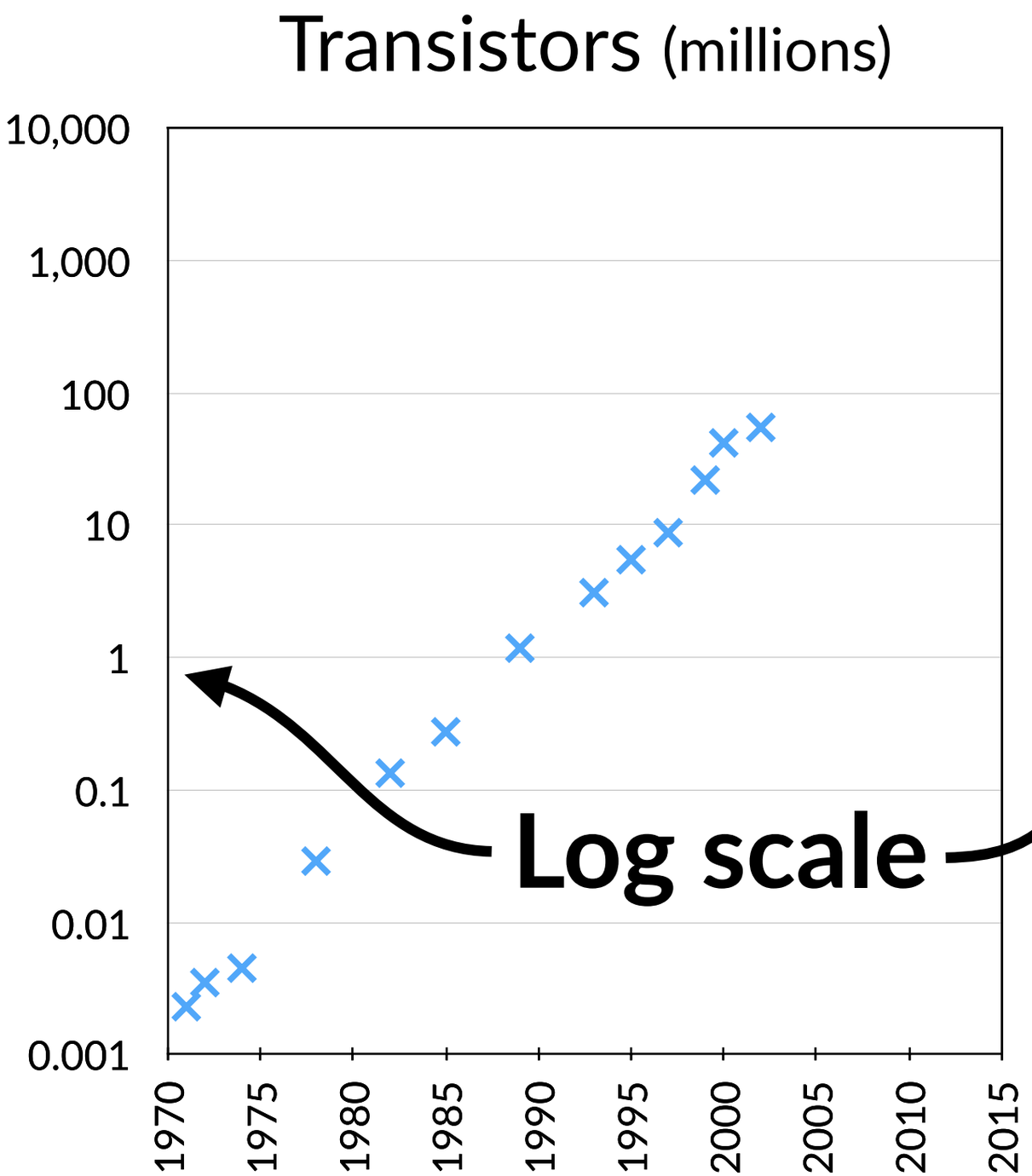
Transistors (millions)



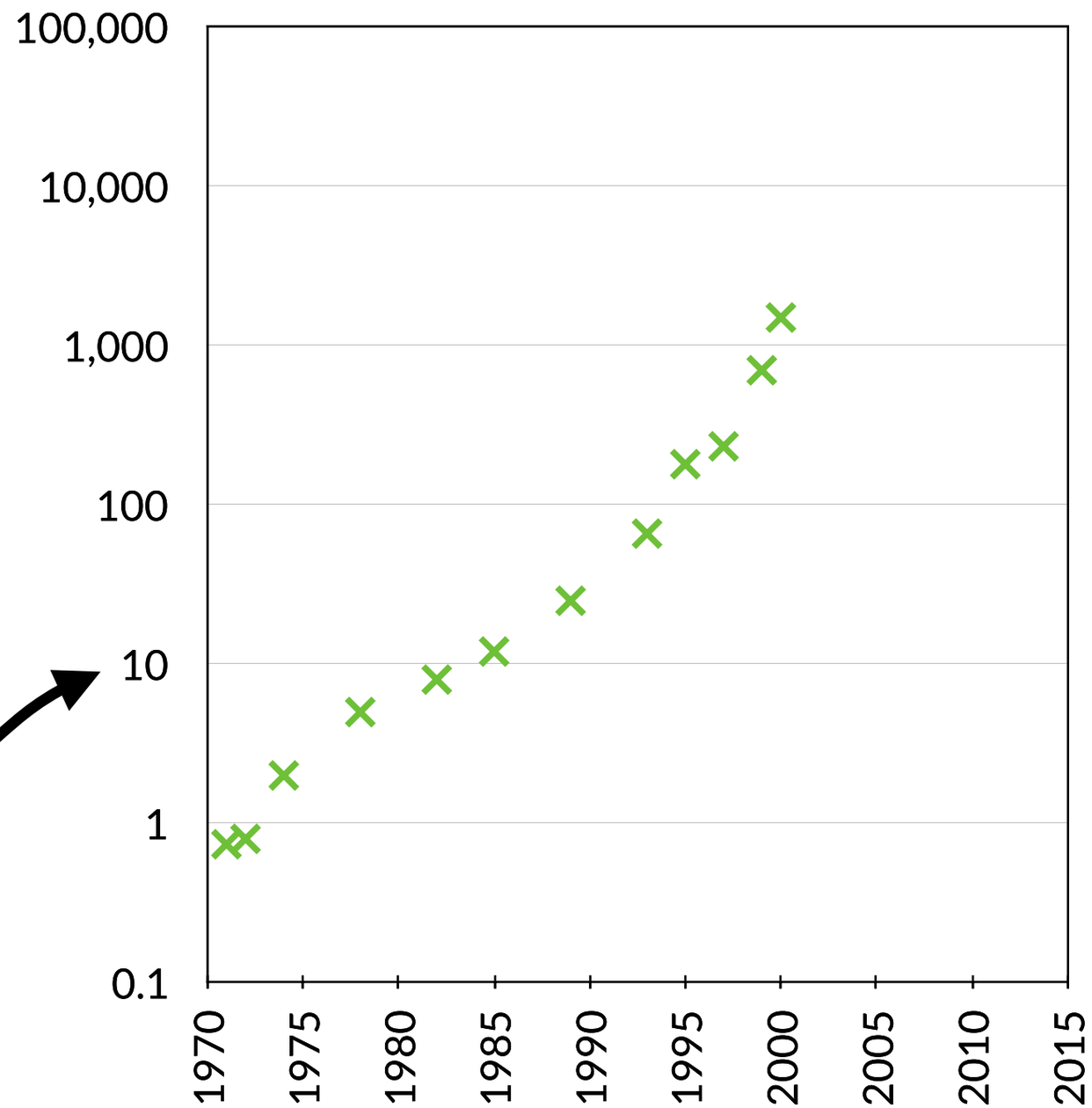
Clock Speed (MHz)



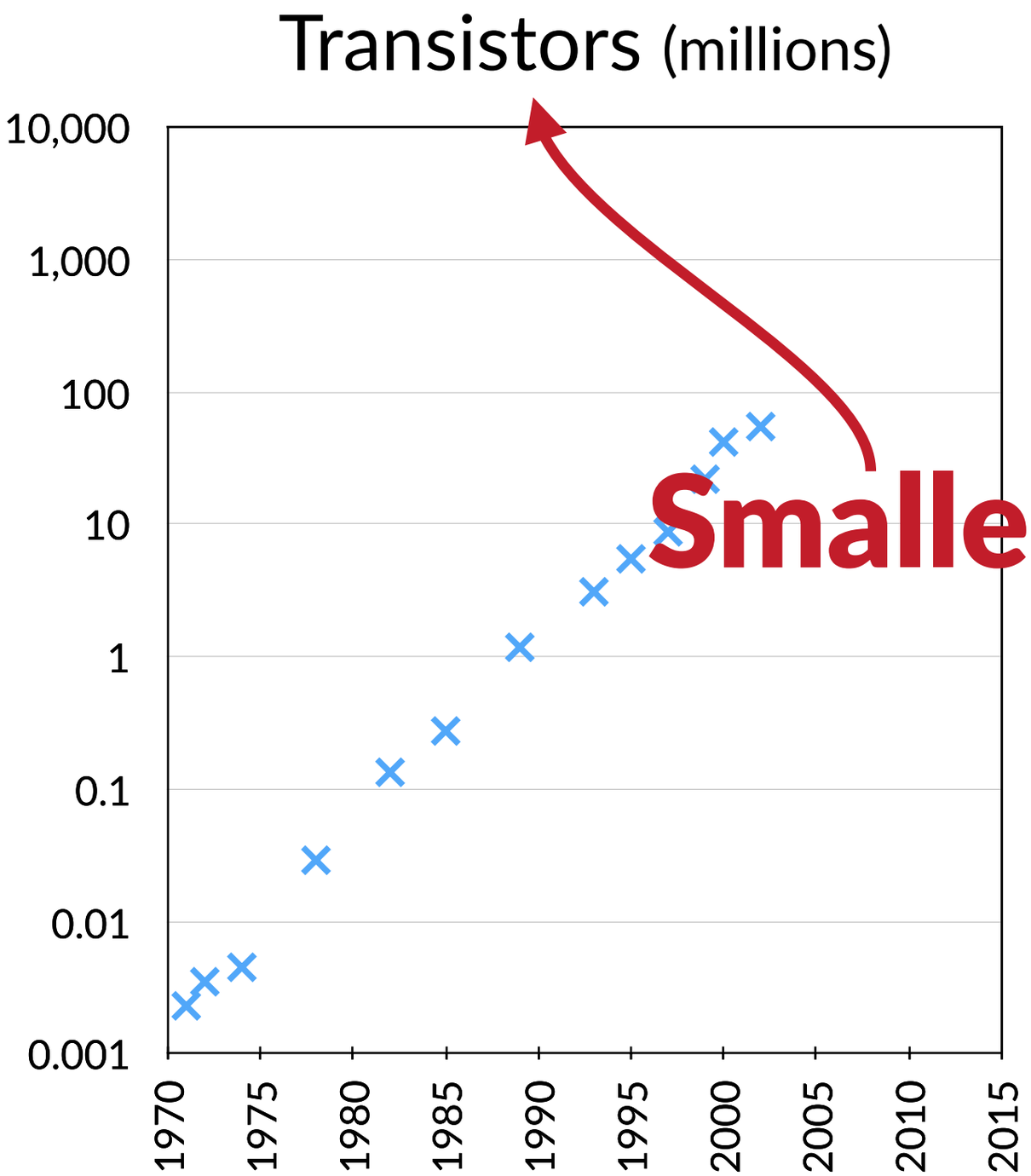
Performance used to be easy

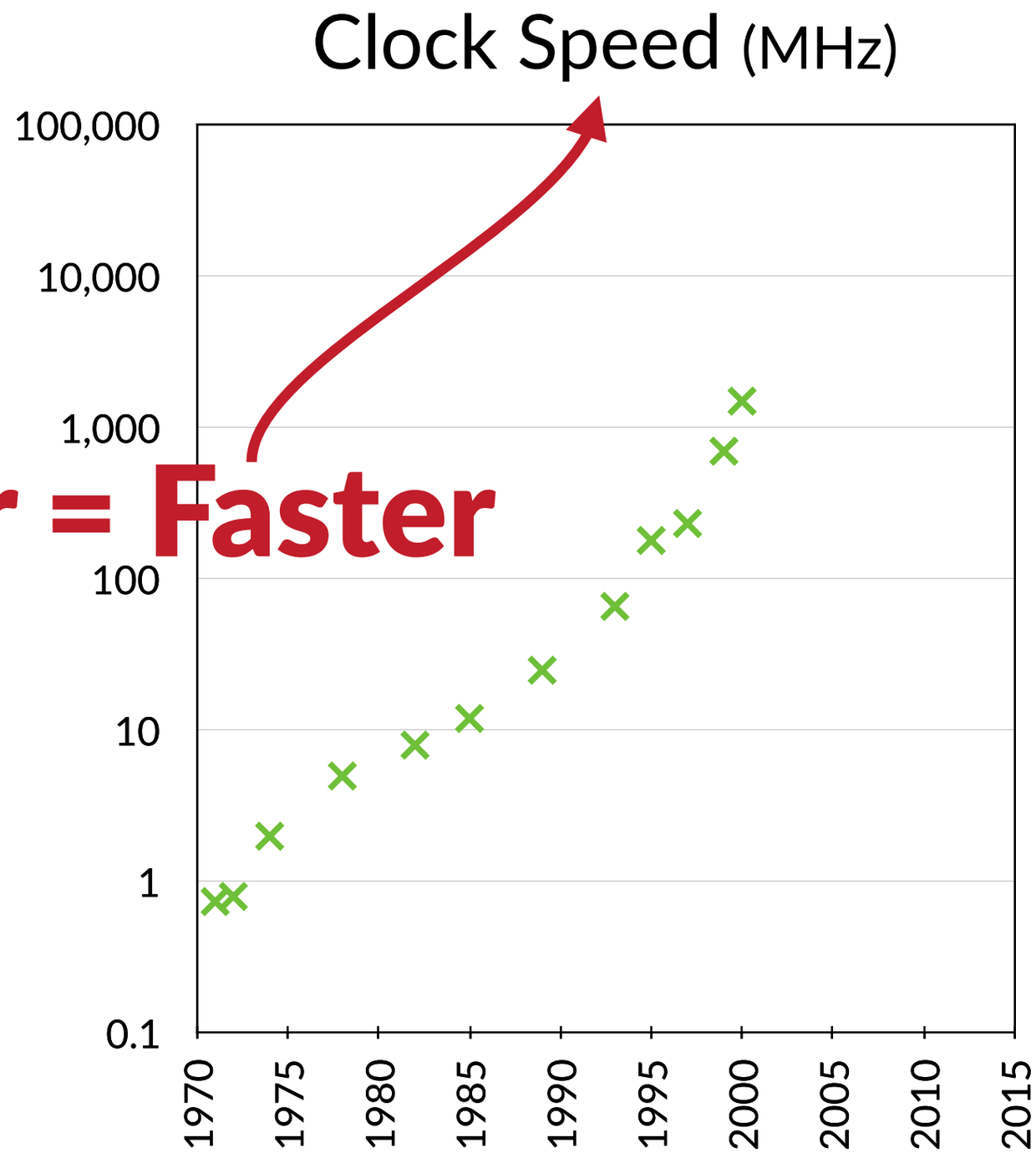


Log scale



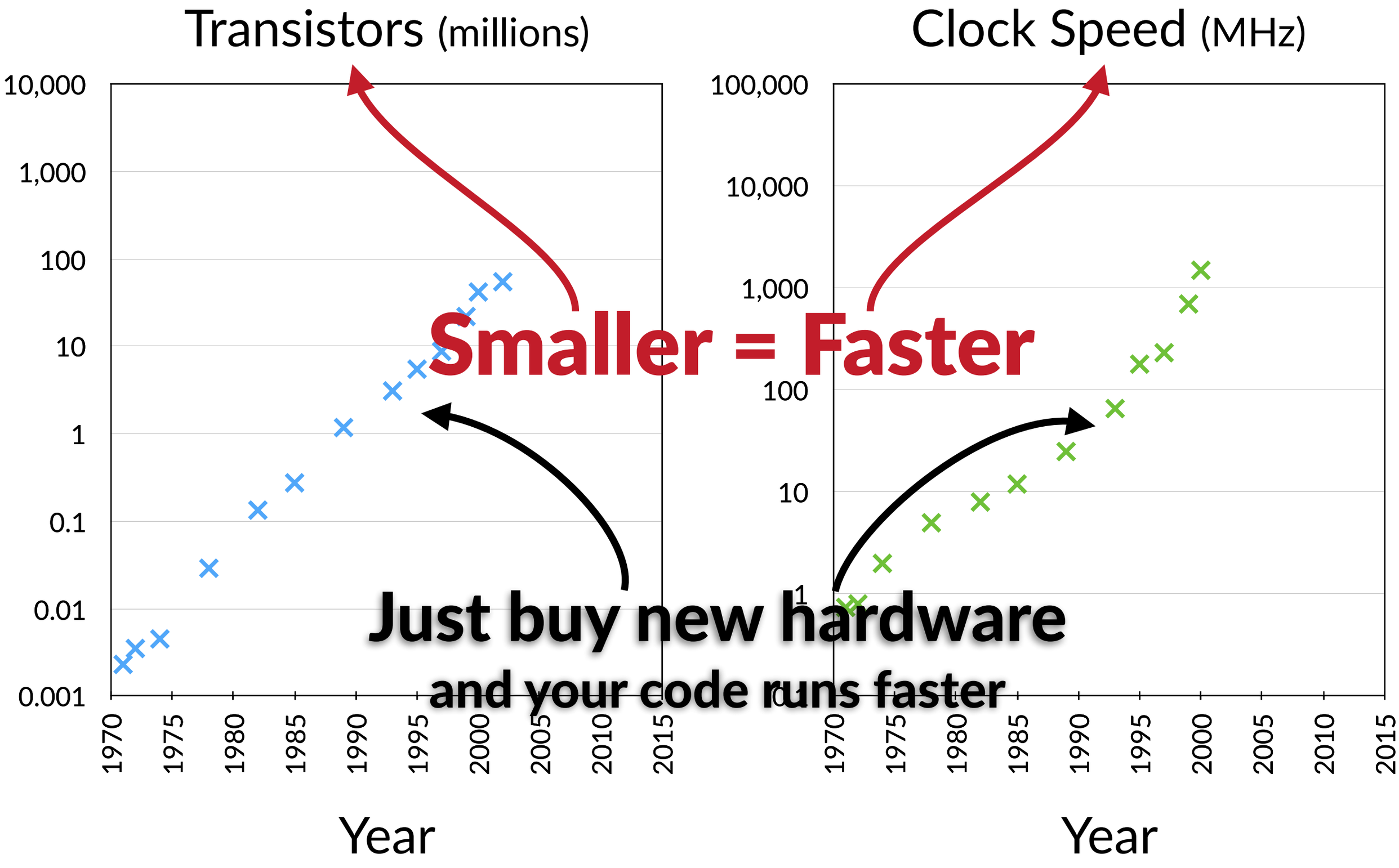
Performance used to be easy





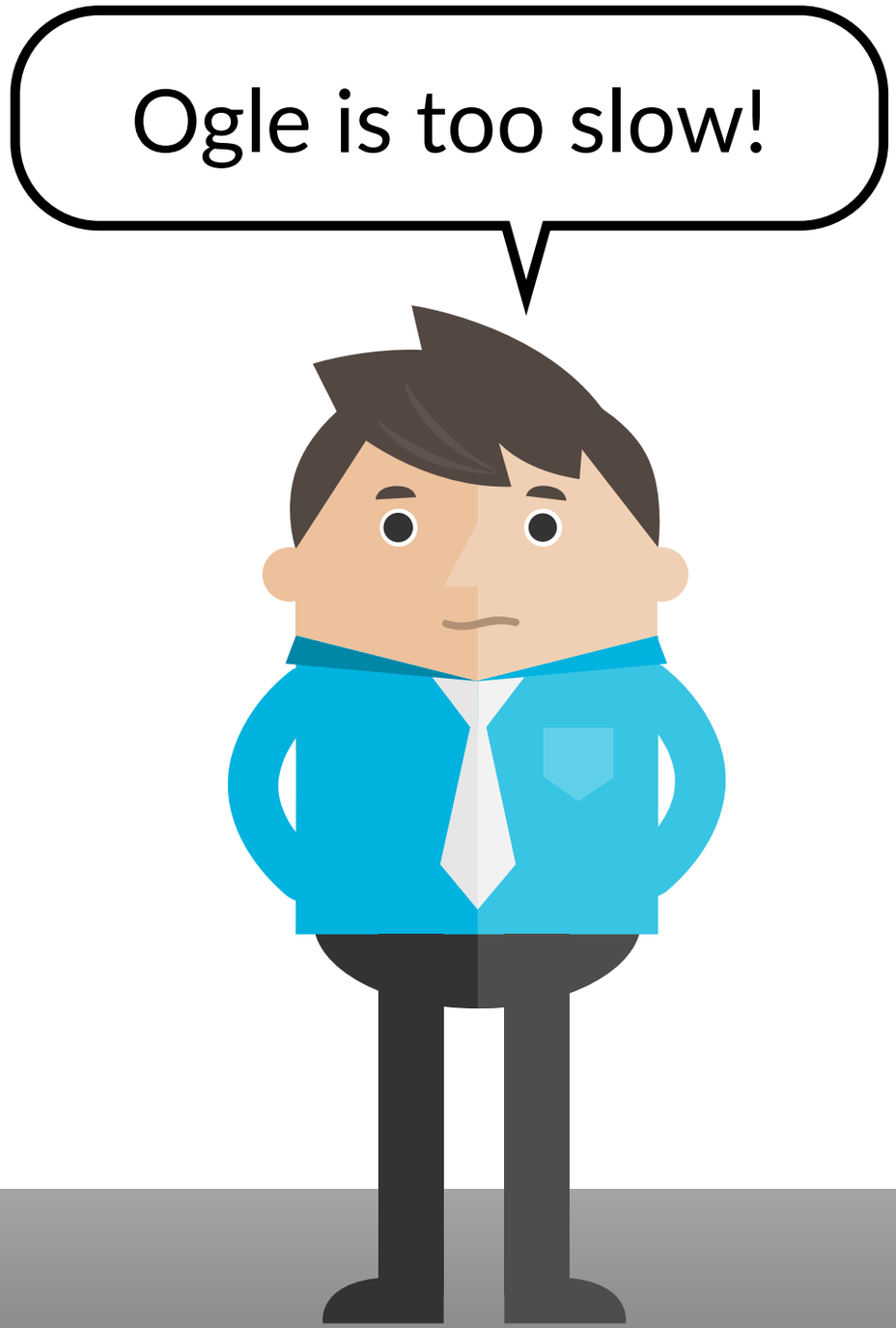
Smaller = Faster

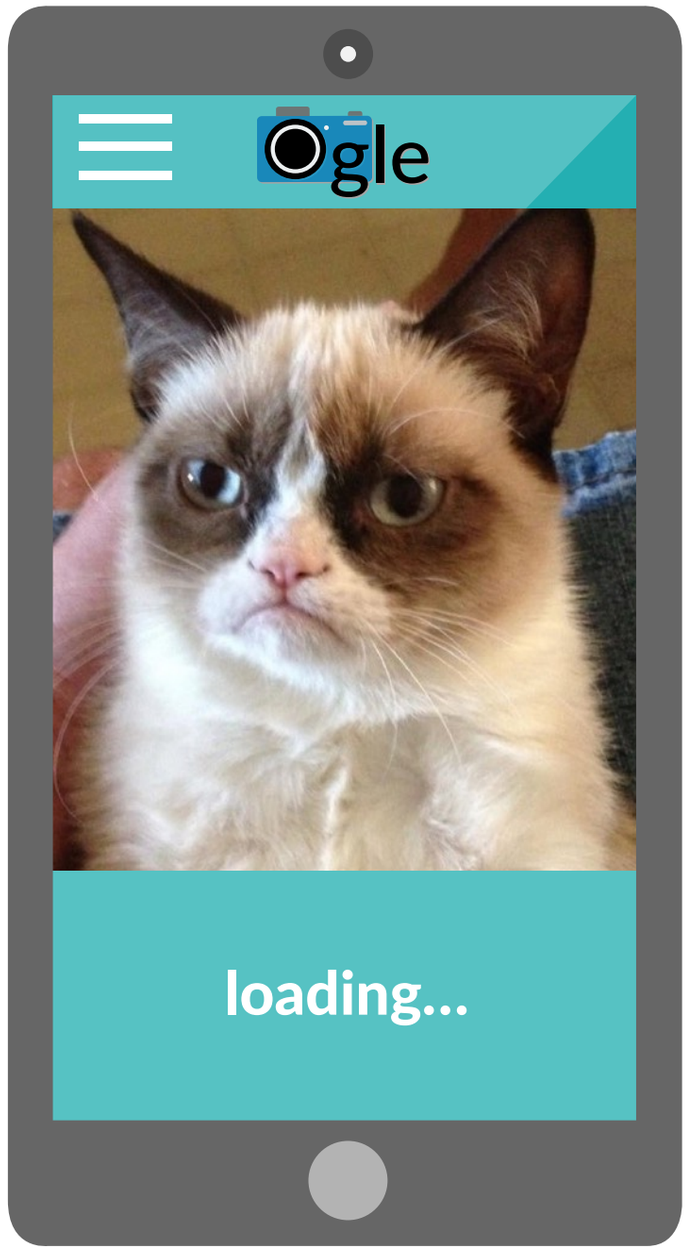
Performance used to be easy



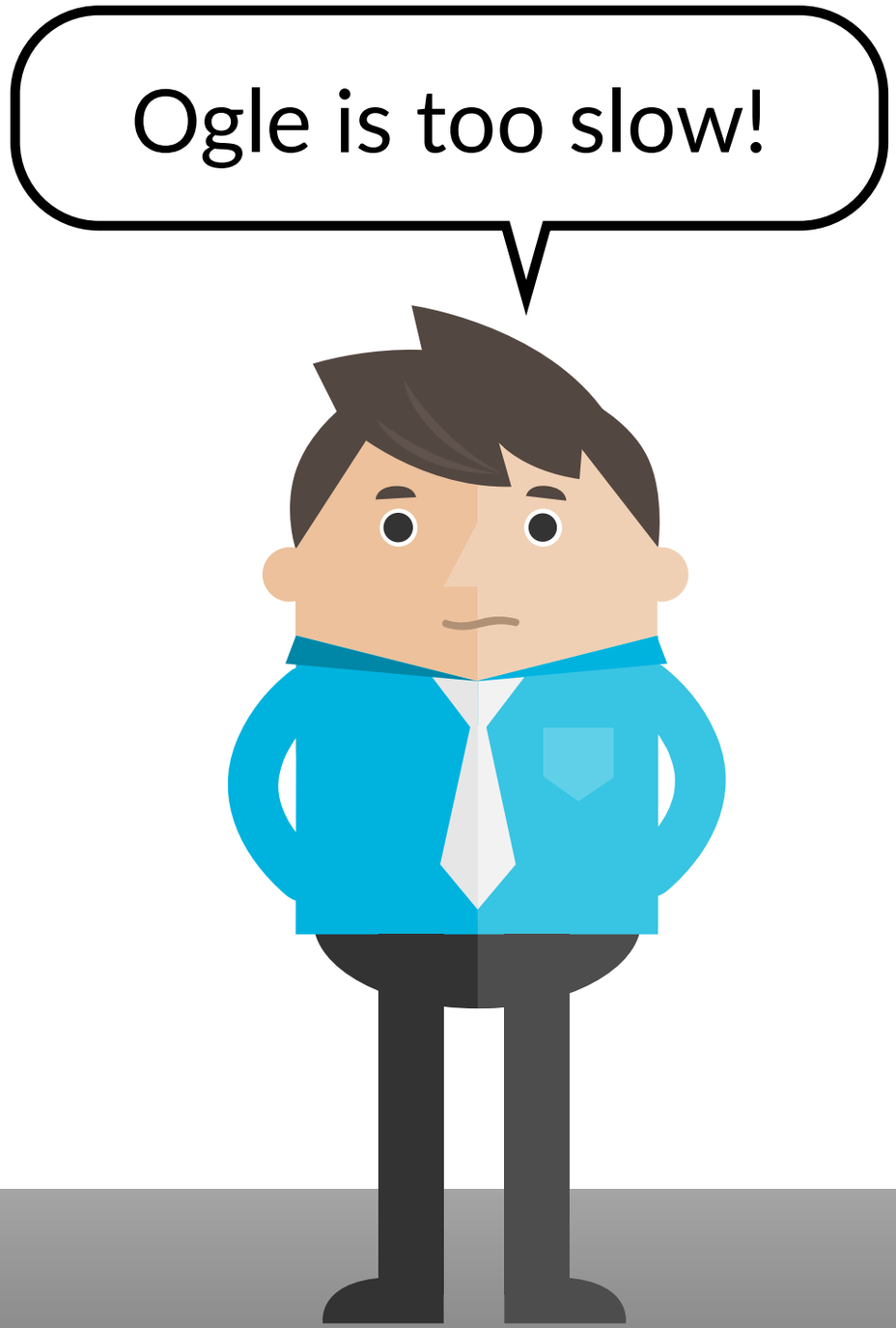
Performance analysis in the ‘80s

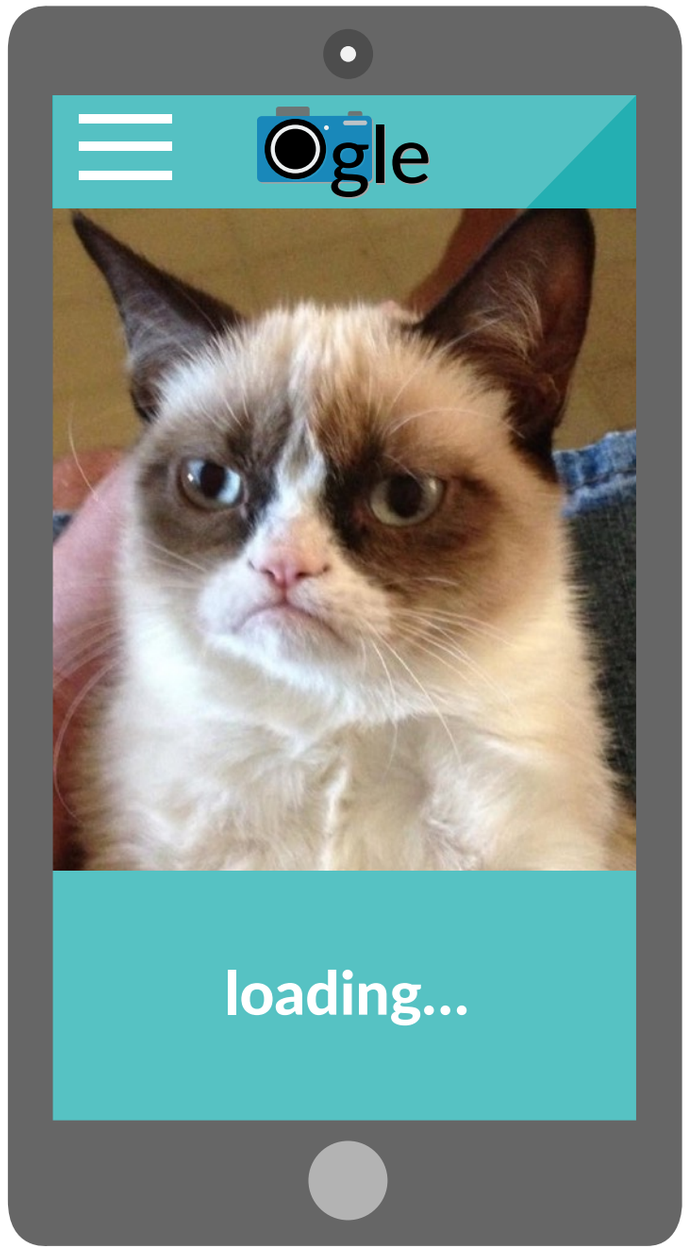
Back to the present...



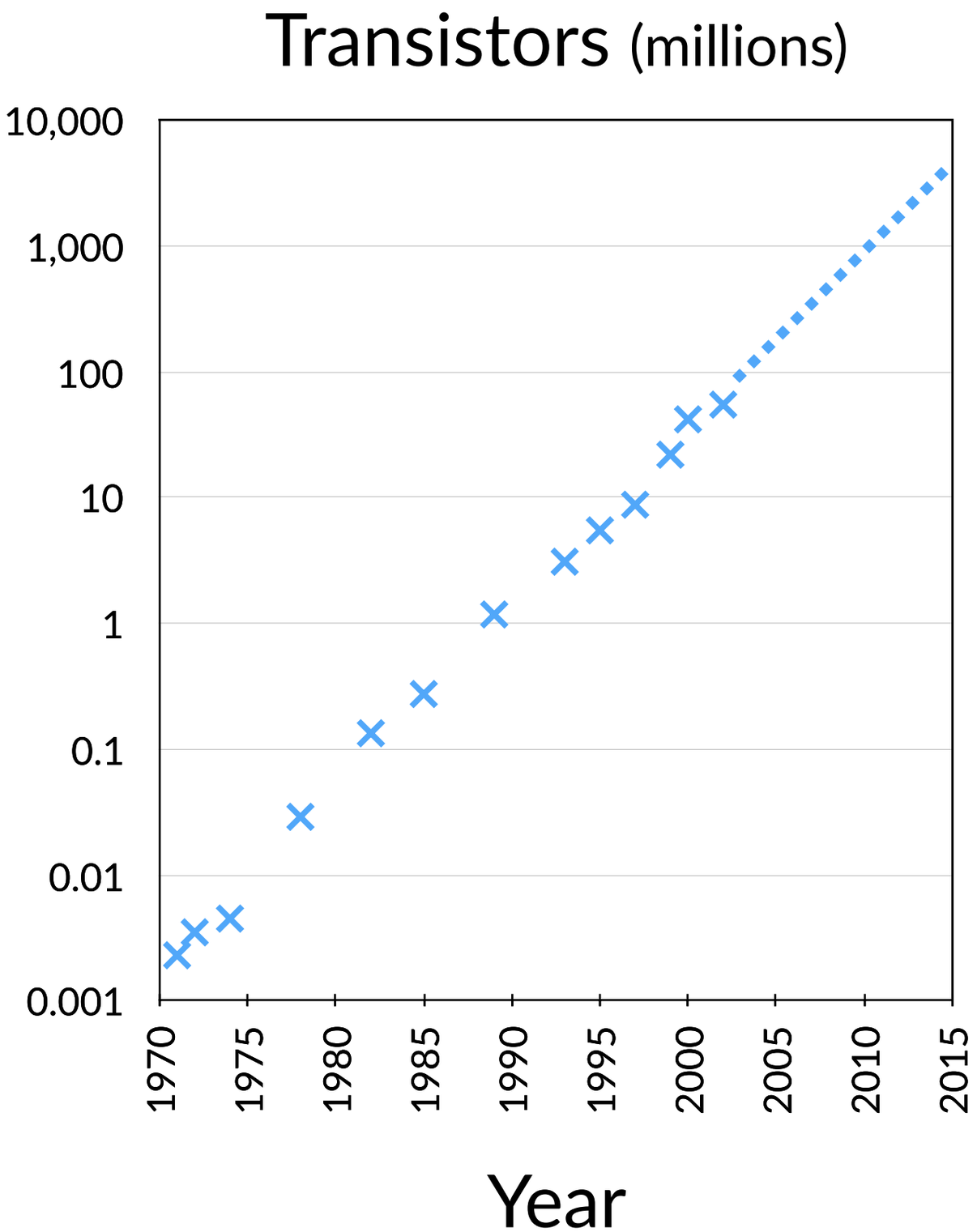


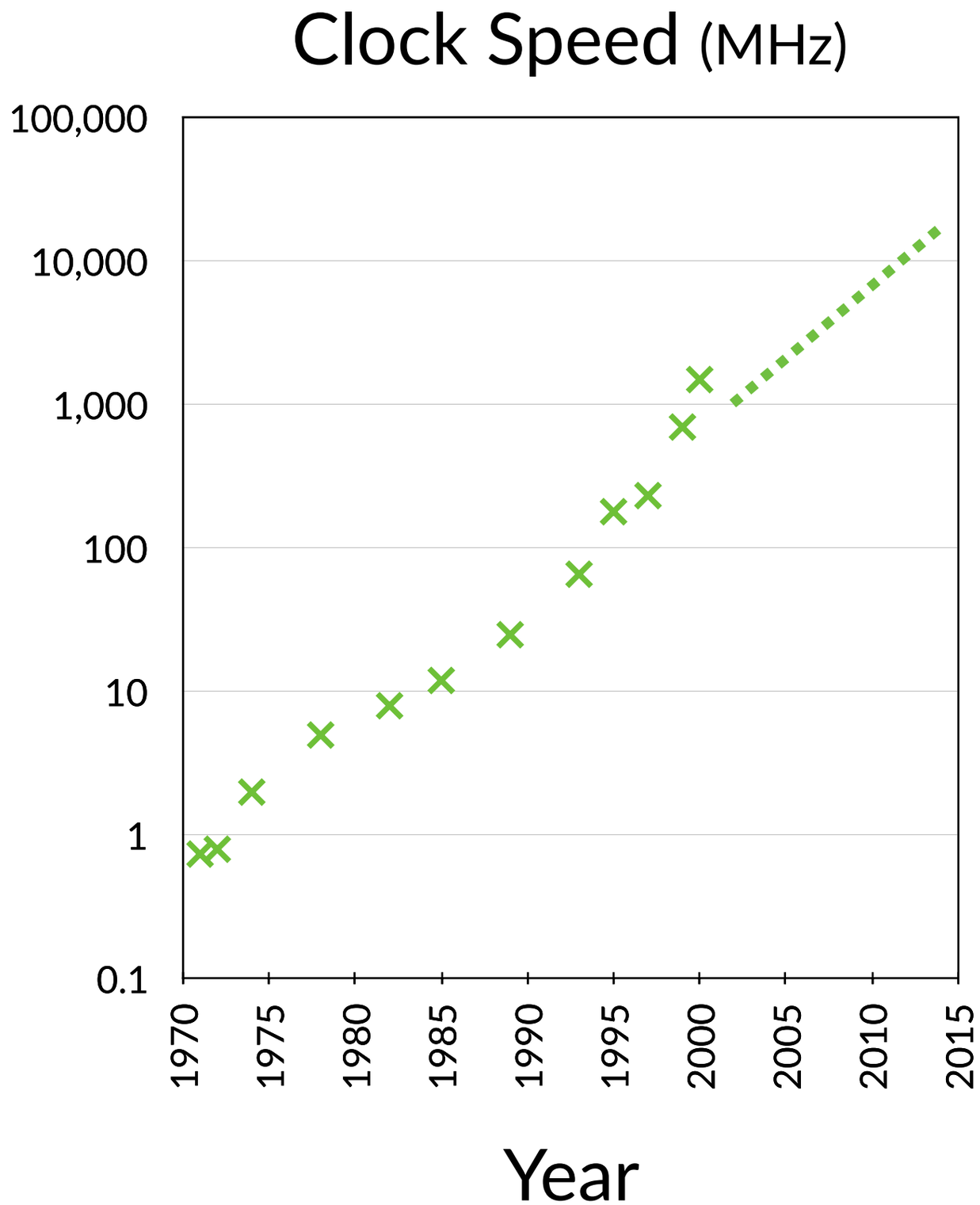
Back to the present...



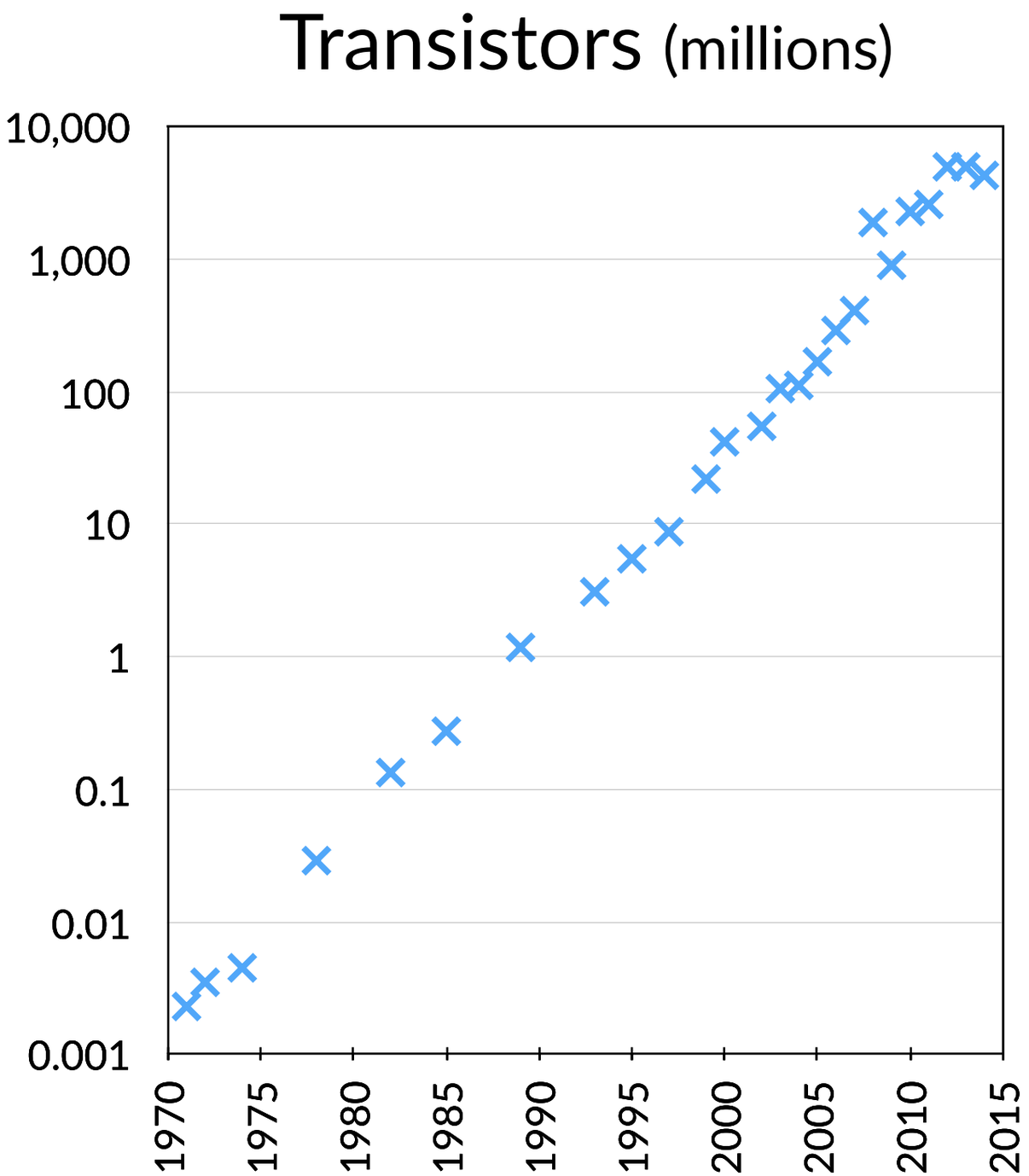


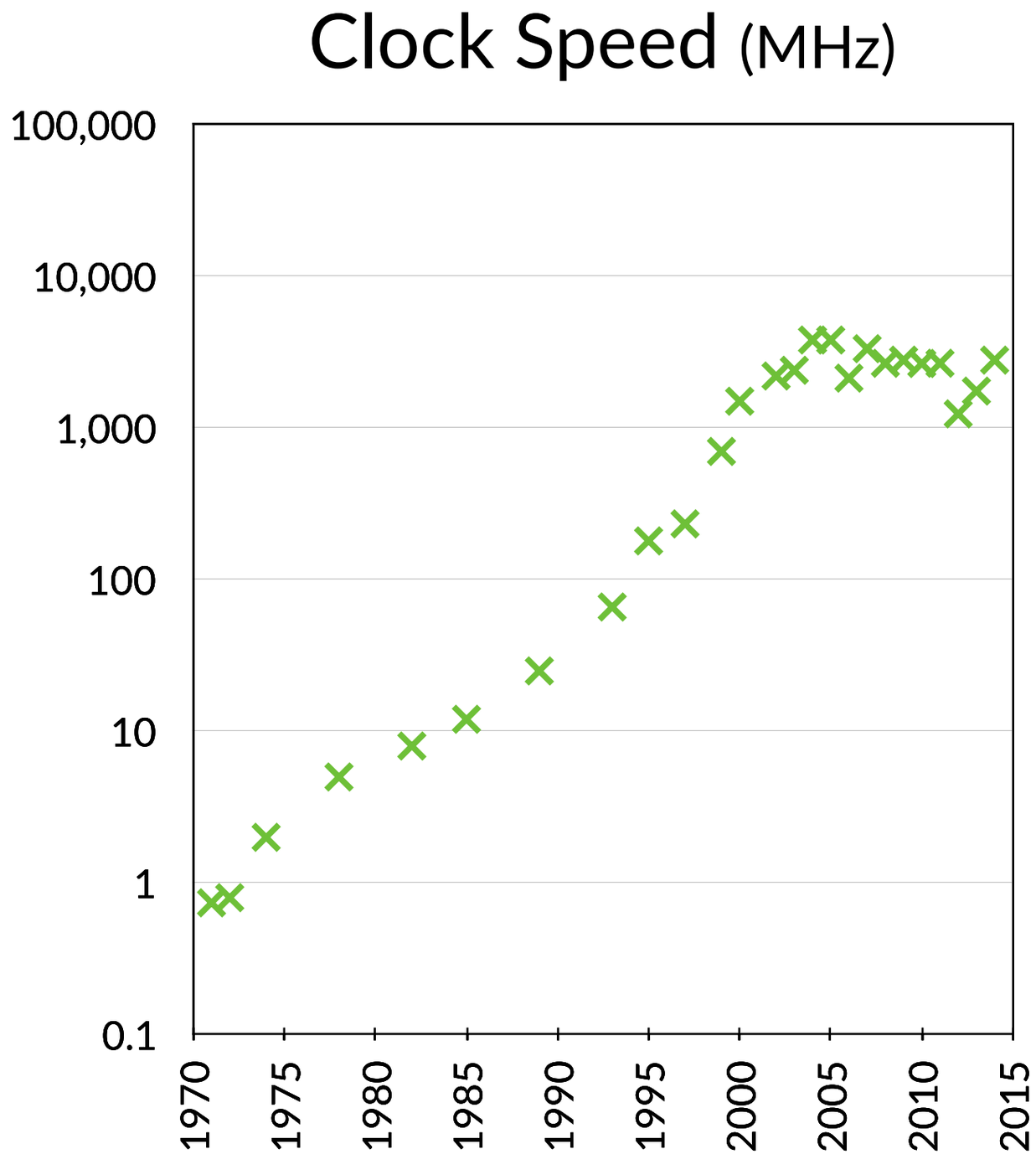
Performance not easy anymore



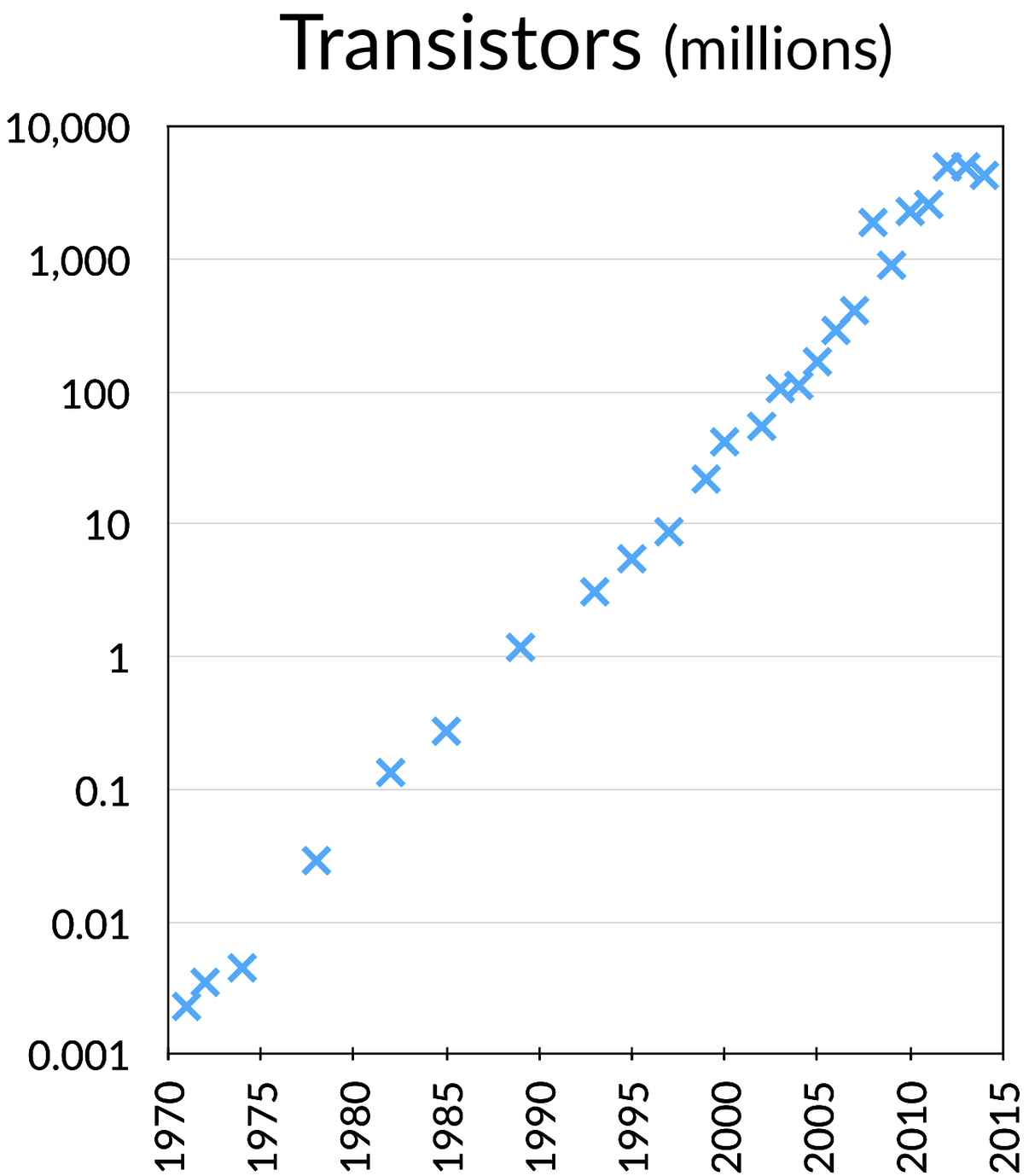


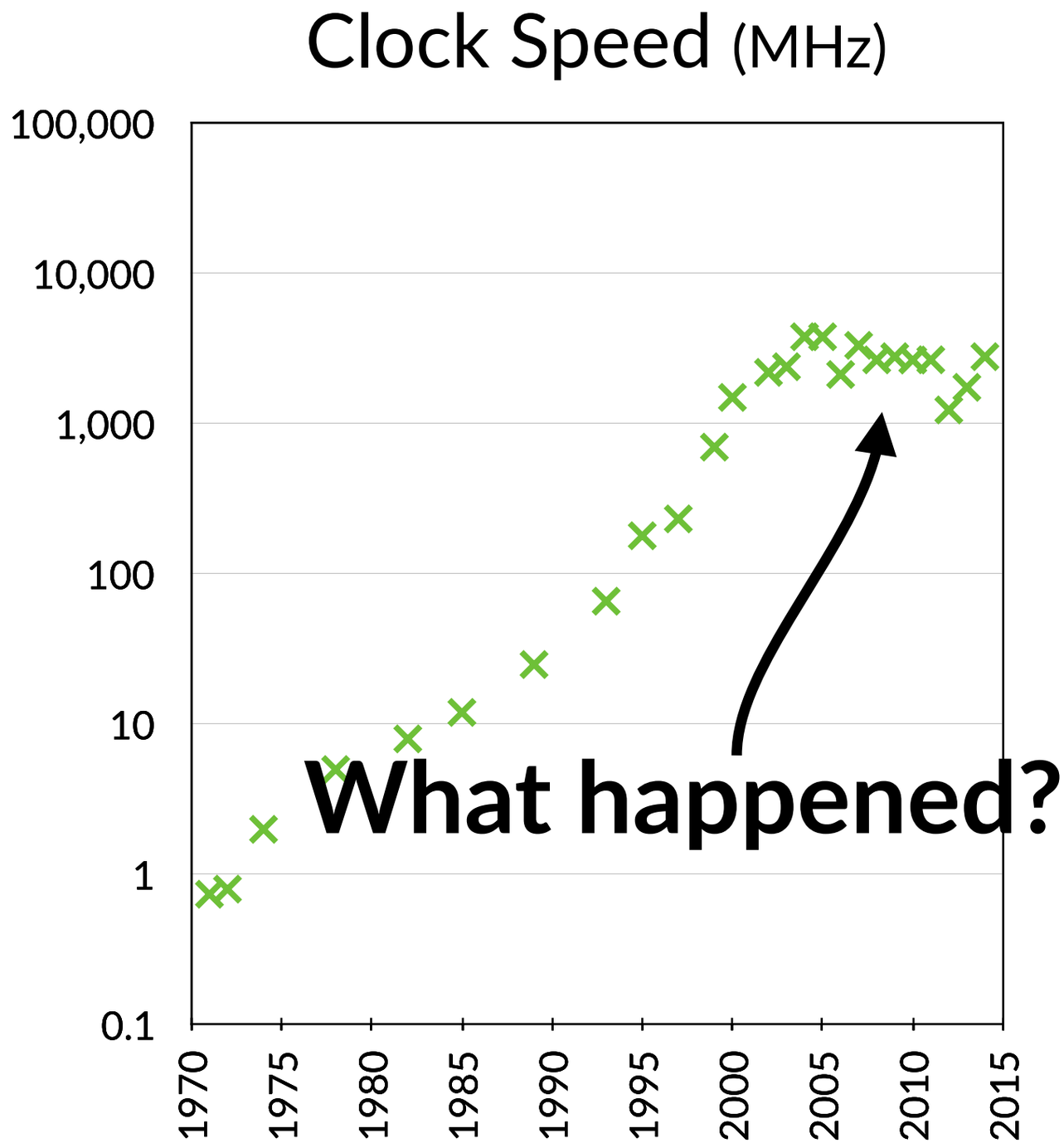
Performance not easy anymore



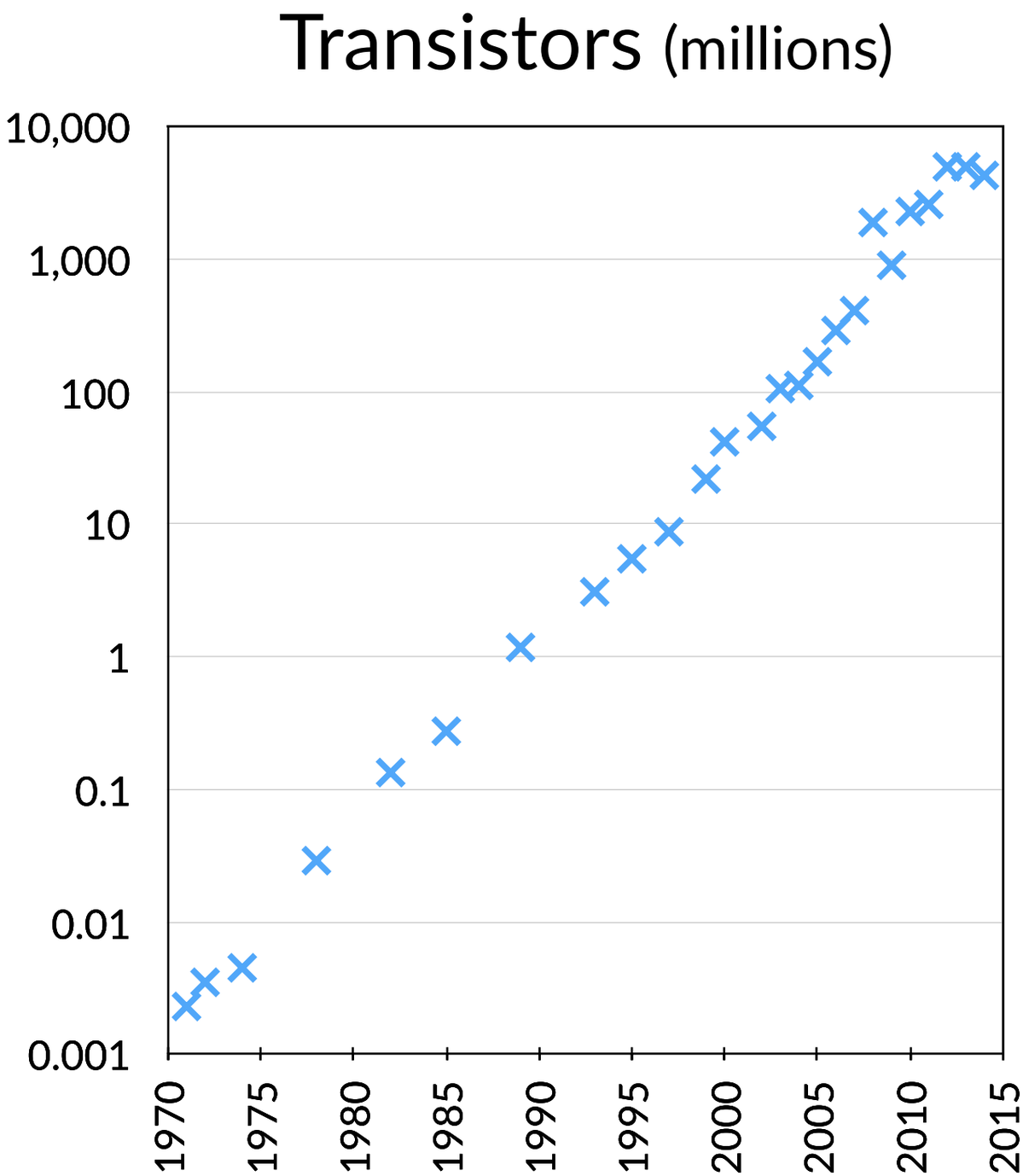


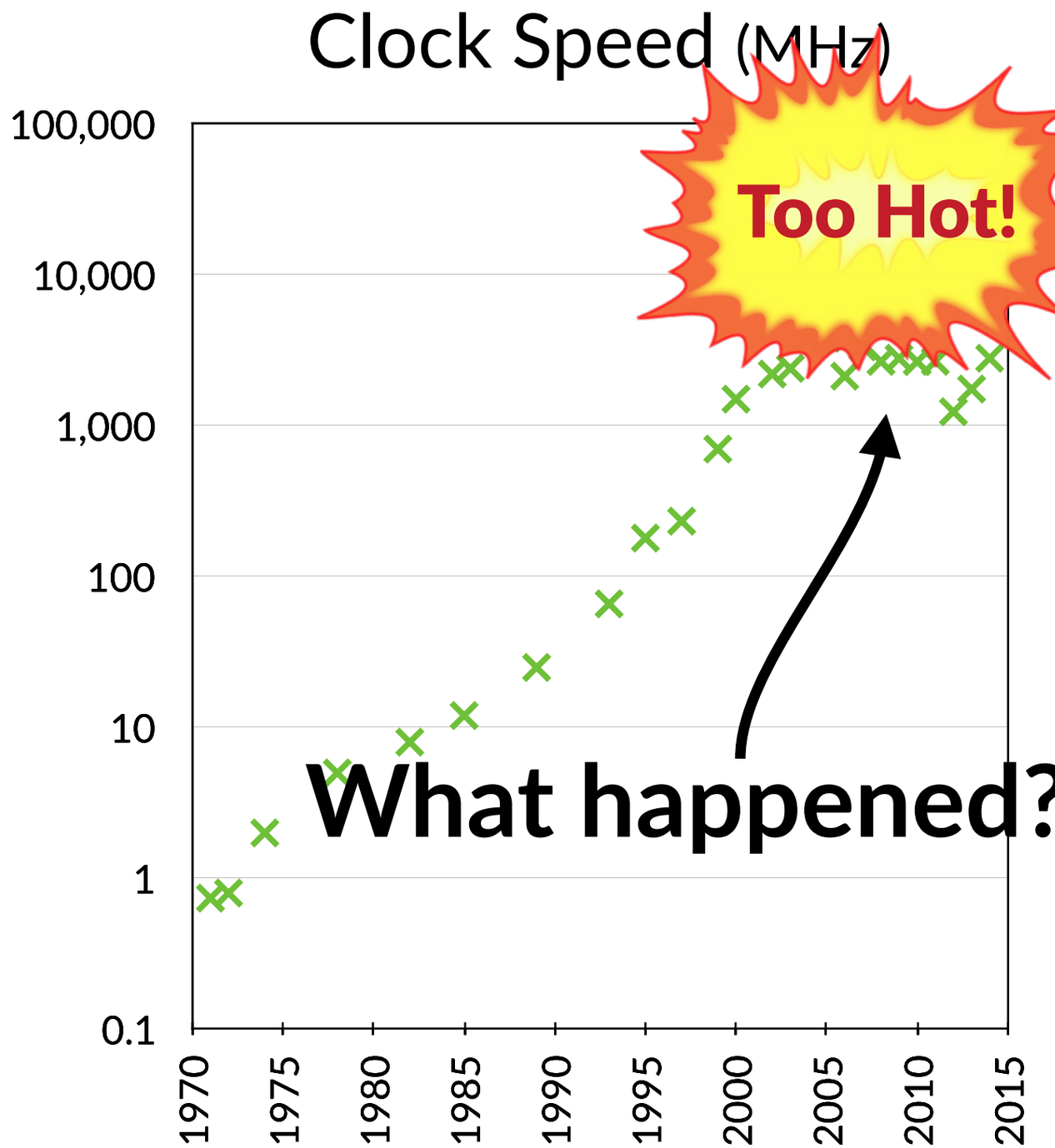
Performance not easy anymore



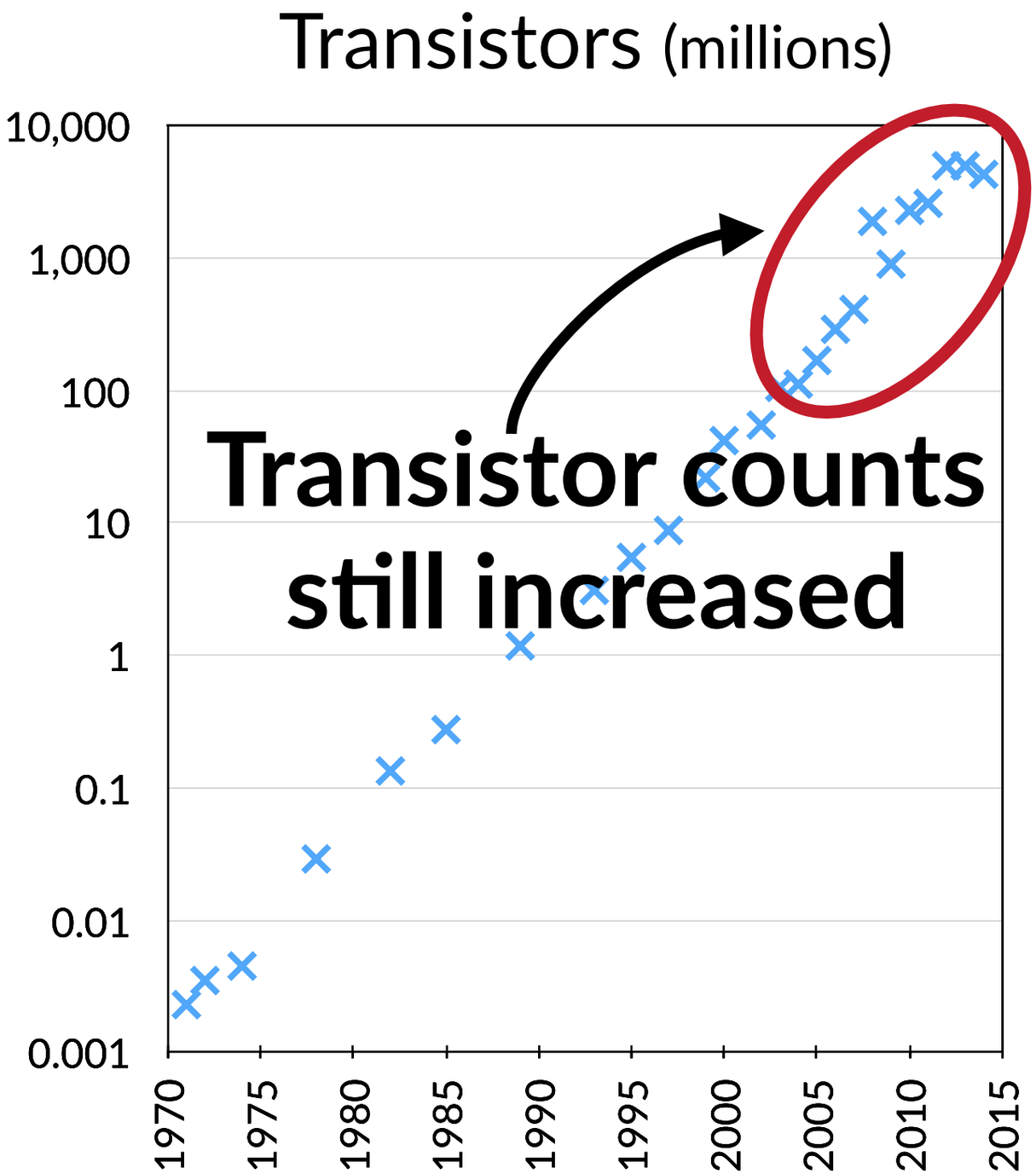


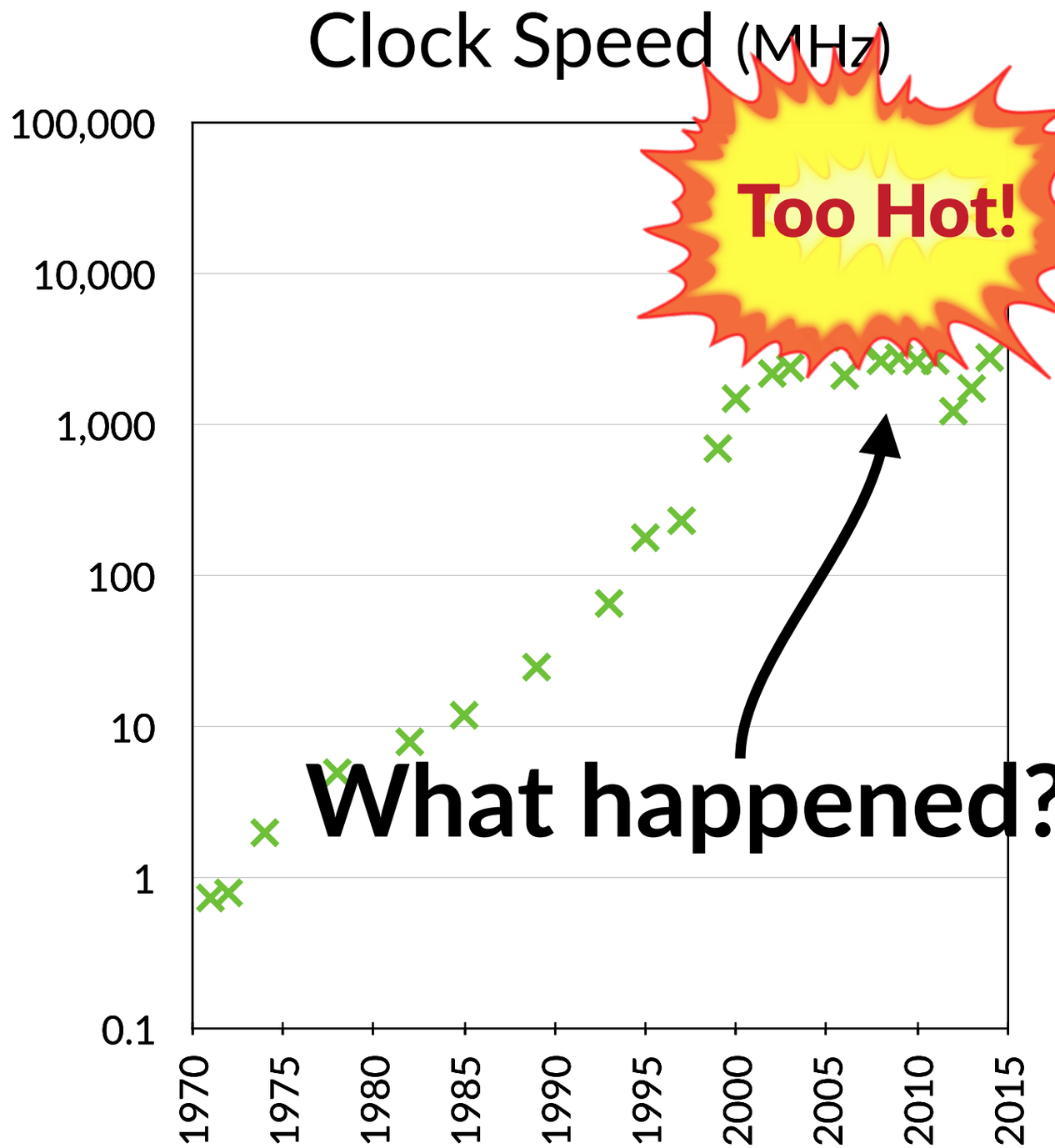
Performance not easy anymore



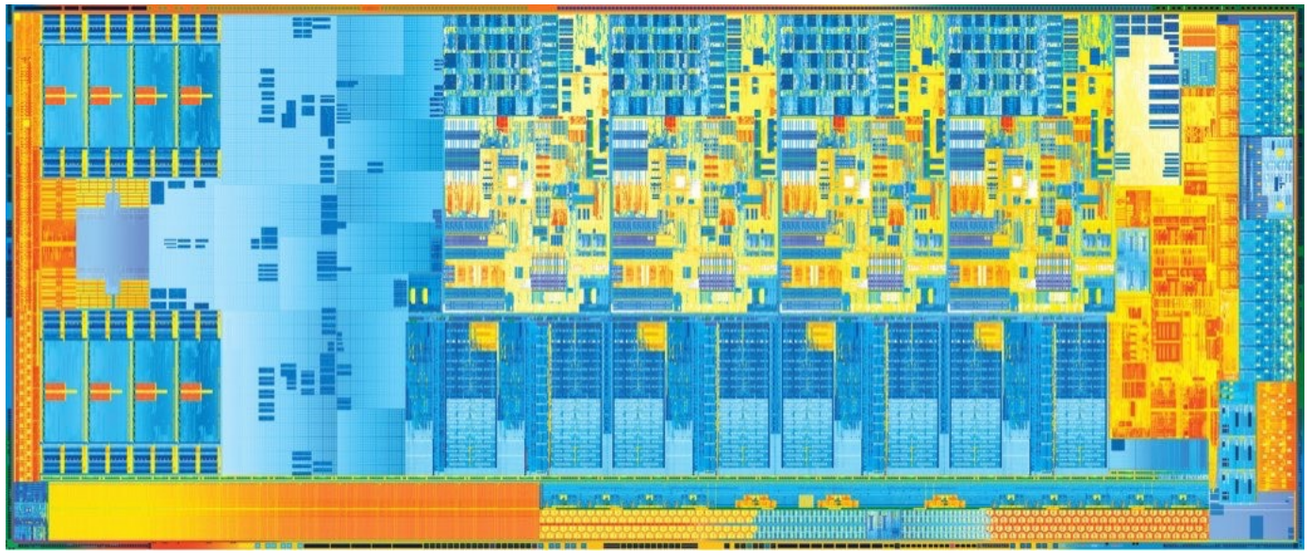


Performance not easy anymore



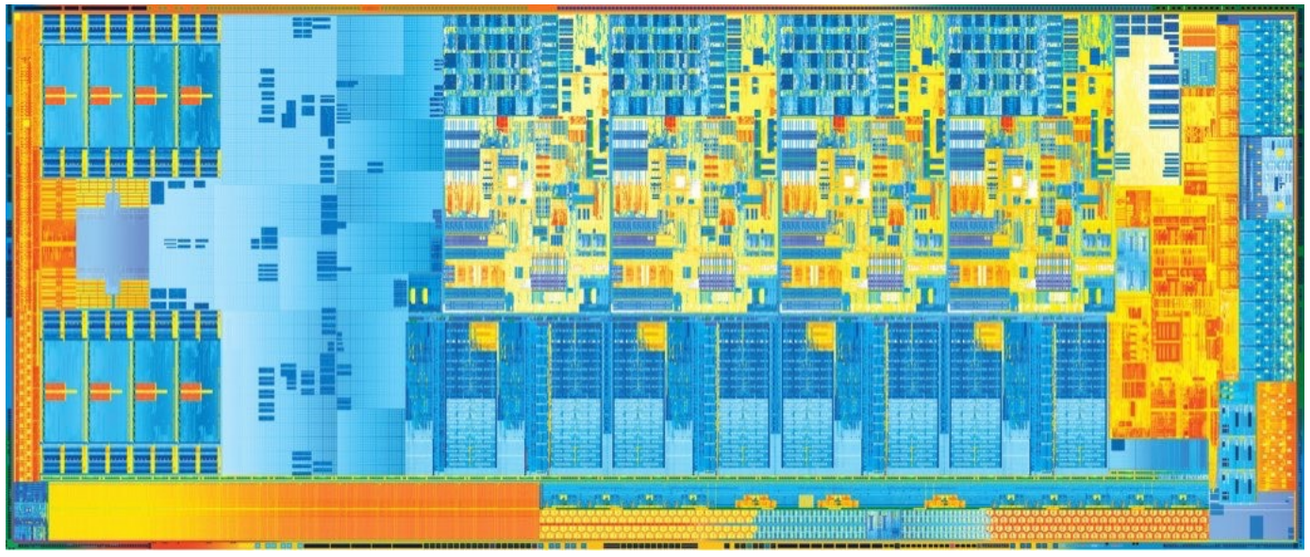


Where do all those transistors go?



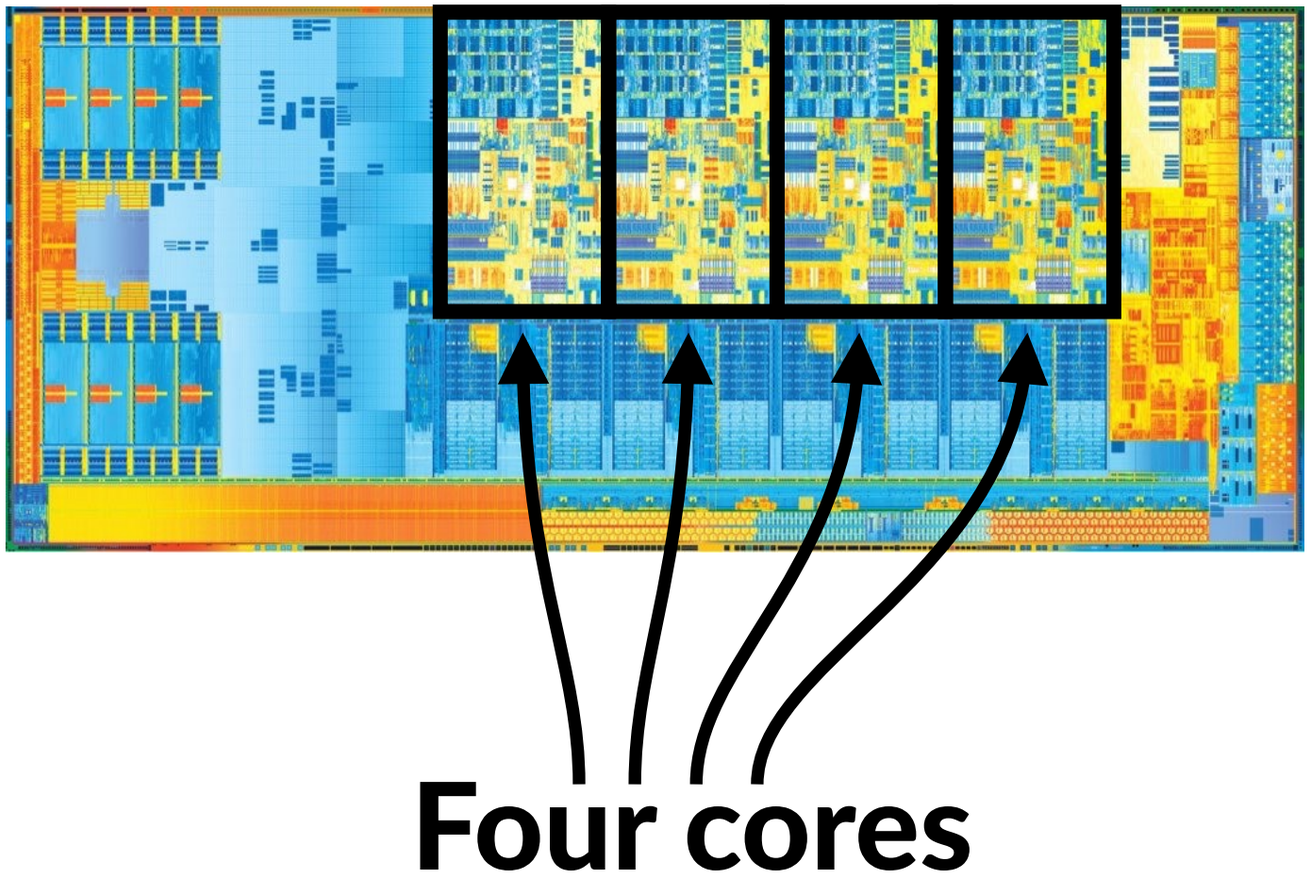
Where do all those transistors go?

Instead of faster processors, we just get more.



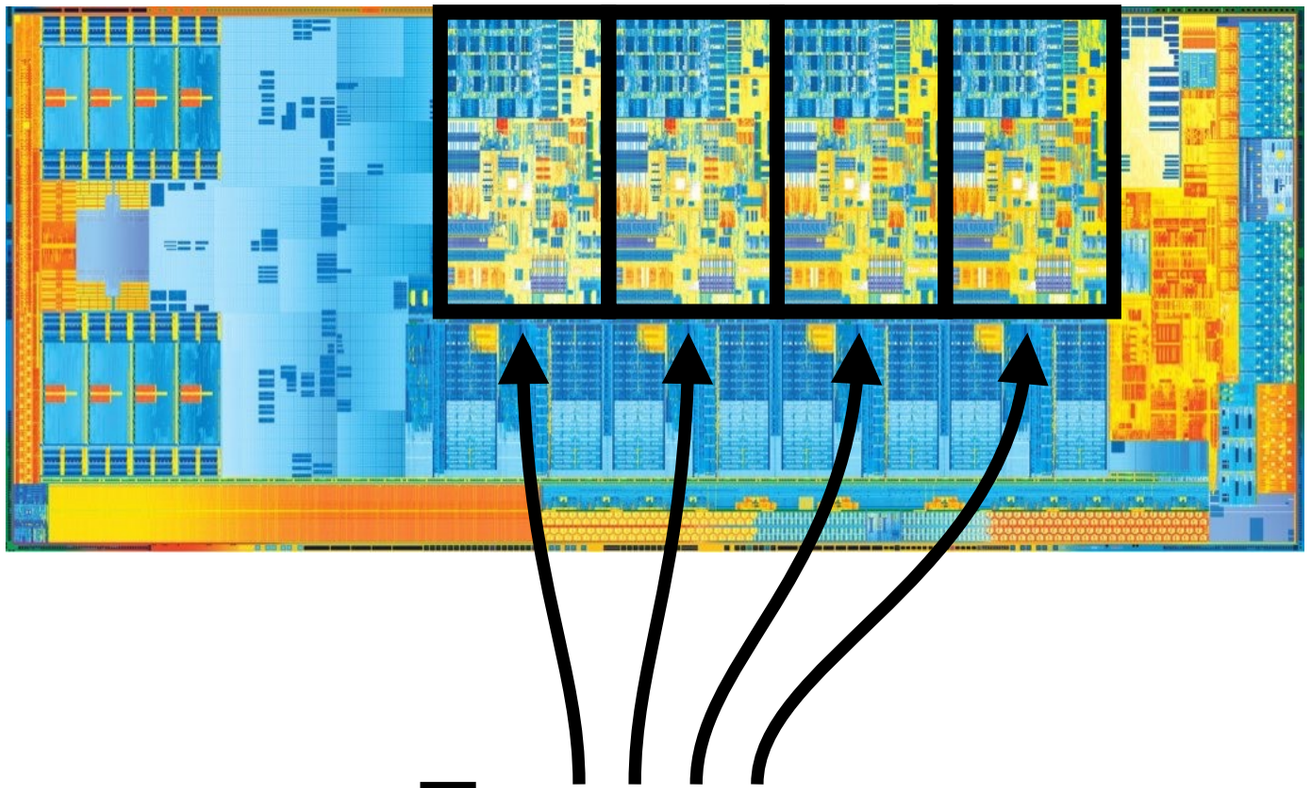
Where do all those transistors go?

Instead of faster processors, we just get more.



Where do all those transistors go?

Instead of faster processors, we just get more.

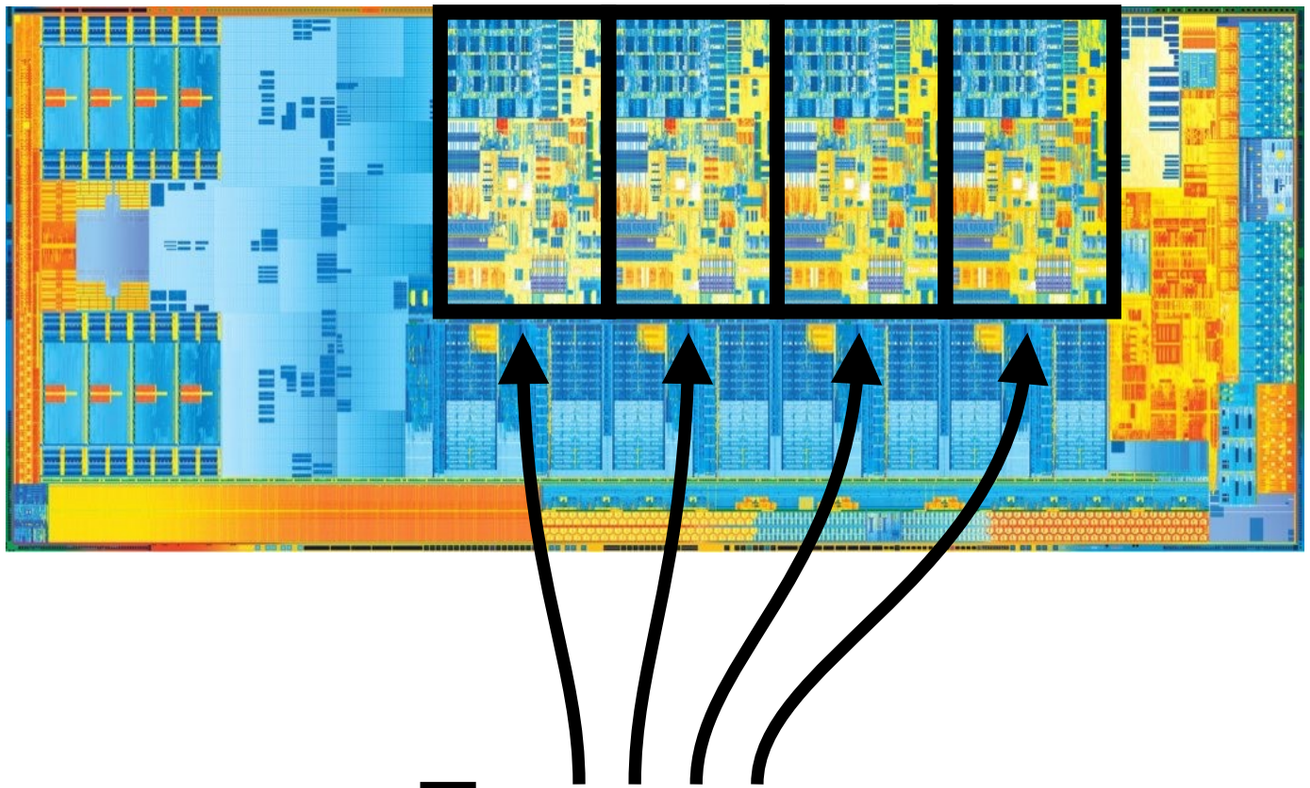


Four cores

Really just four separate processors

Where do all those transistors go?

Instead of faster processors, we just get more.



Four cores

Really just four separate processors

Like having four Pentium Pros from 1995

●●●●● AT&T LTE 12:22 PM 98%

Updates

Update All

Pending Updates

Inbox by Gmail - the…
34.1 MB
Apr 22, 2015

UPDATE

• Bug fixes and performance improvements

TripAdvisor Hotels Fl…
55.9 MB
What's New ▾

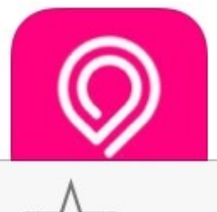
UPDATE

Yelp
46.1 MB
What's New ▾

UPDATE

Google Drive - free…
43.0 MB
What's New ▾

UPDATE



Gogobot - City Guid…
56.3 MB
What's New ▾

UPDATE

Featured Top Charts Near Me Search Updates 7

●●●●● AT&T LTE 12:22 PM 98%

Updates Update All

Pending Updates

Google Drive - free…
43.0 MB
Apr 22, 2015
UPDATE

• Bug fixes and performance improvements

Gogobot - City Guid…
56.3 MB
What's New ▾
UPDATE

Updated April 22, 2015

Microsoft OneNote for…
97.7 MB
What's New ▾
OPEN

WhatsApp Messenger
56.9 MB
What's New ▾
OPEN

Inbox by Gi
34.1 MB
Apr 22, 2015

• Bug fixes and perfor

TripAdvisor
55.9 MB
What's New

Yelp
46.1 MB
What's New

Google Driv
43.0 MB
What's New

Gogobot -
56.3 MB

Featured Top Charts

●●●●● AT&T LTE 12:22 PM 98%

Updates Update All

Pending Updates

Inbox by Gmail
34.1 MB
Apr 22, 2015
• Bug fixes and performance enhancements

TripAdvisor
55.9 MB
What's New

Yelp
46.1 MB
What's New

Google Drive
43.0 MB
What's New

Gogobot - City Guide
56.3 MB
What's New

Featured Top Charts Ne

●●●●● AT&T LTE 12:22 PM 98%

Updates Update All

Pending Updates

Google Drive
43.0 MB
Apr 22, 2015
• Bug fixes and performance enhancements

Gogobot - City Guide
56.3 MB
What's New

Updated April 22

Microsoft OneNote
97.7 MB
What's New

WhatsApp Messenger
56.9 MB
What's New

●●●●● AT&T LTE 12:22 PM 98%

Updates Update All

Pending Updates

Gogobot - City Guid…
56.3 MB
Apr 23, 2015
UPDATE

- Apple Watch: just get a shiny new Apple Watch? You're in luck! Add the new Gogobot app to your Apple watch to discover the best places to eat, play and stay near you!

- Bug fixes and performance enhancements!

Updated April 22, 2015

Microsoft OneNote for…
97.7 MB
What's New ▾
OPEN

WhatsApp Messenger
56.9 MB
OPEN

●●●●● AT&T LTE 12:22 PM 98%

Updates Update All

Pending Updates

Inbox by Gmail
34.1 MB
Apr 22, 2015
• Bug fixes and performance improvements

TripAdvisor
55.9 MB
What's New

Yelp
46.1 MB
What's New

Google Drive
43.0 MB
What's New

Gogobot - Travel Guides
56.3 MB
What's New

Featured Top Charts Ne

●●●●● AT&T LTE 12:22 PM 98%

Updates Update All

Pending Updates

Google Drive
43.0 MB
Apr 22, 2015
• Bug fixes and performance improvements

Gogobot - Travel Guides
56.3 MB
What's New

Updated April 22

Microsoft OneNote
97.7 MB
What's New

WhatsApp Messenger
56.9 MB
What's New

●●●●● AT&T LTE 12:22 PM 98%

Updates Update All

Pending Updates

Gogobot - Travel Guides
56.3 MB
Apr 23, 2015
- Apple Watch: Watch? You're in app to your Apple Watch places to eat, p

- Bug fixes and performance improvements

Updated April 22

Microsoft OneNote
97.7 MB
What's New

WhatsApp Messenger
56.9 MB
What's New

●●●●● AT&T LTE 12:23 PM 98%

Updates Update All

Updated April 22, 2015

Fly Delta
51.5 MB
Apr 21, 2015
OPEN

What's New in Fly Delta 3.3
• Introducing Fly Delta for Apple Watch – Keep track of time to departure and arrival along with gate and baggage carousel information
• Any eBoarding pass(es) you save to Passbook will automatically be available on Apple Watch
• Fixed an issue which caused the app to quit unexpectedly before boarding if a seat had not yet been assigned
• Fixed an issue which caused some customers to see an error message when attempting to view/change seats
• Several other bug fixes and performance improvements

AP Mobile
23.3 MB
OPEN

●●●●● AT&T LTE 12:23 PM 98%

Update All

●●●●● AT&T LTE 12:23 PM 98%

Updates Update All

Updated April 22, 2015

AP Mobile
23.3 MB
Apr 20, 2015
OPEN

* Support for Apple Watch! - Stay up-to-date with breaking news notifications and force touch to save a story to your phone.
* Improved story sharing experience through AP's Bigstory.ap.org site.
* Bug fixes and the usual performance enhancements.

Thanks for your support!

TripIt - Travel Organize…
15.6 MB
What's New ▾
OPEN

Updated April 20, 2015

Movies by Flixster, wit…
49.5 MB
What's New ▾
OPEN

●●●●● AT&T LTE 12:23 PM 98%

Updates Update All

Updated April 22, 2015

Fly Delta
51.5 MB
Apr 21, 2015
OPEN

What's New in Fly Delta 3.3
• Introducing Fly Delta for Apple Watch – Keep track of time to departure and arrival along with gate and baggage carousel information
• Any eBoarding pass(es) you save to Passbook will automatically be available on Apple Watch
• Fixed an issue which caused the app to quit unexpectedly before boarding if a seat had not yet been assigned
• Fixed an issue which caused some customers to see an error message when attempting to view/change seats
• Several other bug fixes and performance improvements

AP Mobile
23.3 MB
OPEN

●●●●● AT&T LTE 12:23 PM 97%

Updates

Update All

Updated April 20, 2015

Google Play Music
13.1 MB
Apr 20, 2015
OPEN

Crash fixes and general performance improvements.

Starbucks
33.7 MB
What's New ▾
OPEN

Shazam
41.4 MB
What's New ▾
OPEN

Instagram
20.5 MB
What's New ▾
OPEN

Updated April 18, 2015

●●●●● AT&T LTE 12:2

Upd

Updated April 22, 201

AP Mobile
23.3 MB
Apr 20, 2015

* Support for Apple Wa
with breaking news no
touch to save a story t
* Improved story sharir
AP's Bigstory.ap.org si
* Bug fixes and the usu
enhancements.

Thanks for your suppo

TripIt - Trave
15.6 MB
What's New ▾

Updated April 20, 201

Movies by F
49.5 MB
What's New ▾

Pendi

• Bug

23 PM 98%

dates Update All

15

OPEN

ta 3.3
for Apple Watch – Keep
ture and arrival along
e carousel information
(es) you save to
tically be available on

caused the app to quit
oarding if a seat had

caused some
error message when
ange seats
es and performance

OPEN

●●●●● AT&T LTE 12:24 PM 97%

Updates

Update All

Updated April 16, 2015

Kindle – Read Books,…
77.3 MB
Apr 15, 2015
OPEN

Performance and Stability Improvements

Updated April 13, 2015

Facebook
51.2 MB
What's New ▾
OPEN

Updated April 12, 2015

Evernote
63.7 MB
What's New ▾
OPEN

Updated April 11, 2015

Amazon Local – Resta…
10.6 MB
What's New ▾
OPEN

Featured 7 OPEN

●●●●● AT&T LTE 12:2

Upd

Updated April 22, 201

AP Mobile
23.3 MB
Apr 20, 2015

* Support for Apple Wa
with breaking news no
touch to save a story t
* Improved story sharir
AP's Bigstory.ap.org si
* Bug fixes and the usu
enhancements.

Thanks for your suppo

TripIt - Trave
15.6 MB
What's New ▾

Updated April 20, 201

Movies by F
49.5 MB
What's New ▾

●●●●● AT&T LTE

Updated April

Goog
13.1 M
Apr 2(

Crash fixes and
improvements.

Starb
33.7 M
What'

Shaz
41.4 M
What'

Insta
20.5 M
What'

Updated April

98%

date All

OPEN

h – Keep
along
nation

ole on

o to quit
t had

hen

ance

●●●●● AT&T LTE 12:24 PM 97%

Updates

Updated April 13, 2015

Facebook
51.2 MB
Apr 13, 2015
OPE

Thanks for using Facebook! To make our ap better for you, we bring updates to the App Store every 2 weeks. You can update the ap automatically (without checking back here) going to Settings > iTunes & App Store > Automatic Downloads and turning on Updates.

Every update of our Facebook app includes improvements for speed and reliability. As other new features become available, we'll highlight those for you in the app.

Updated April 12, 2015

Evernote
63.7 MB
What's New ▾
OPE

Updated April 11, 2015

●●●●● AT&T LTE 12:24 PM 97%

Updated Apri

Kin
77.3
Apr

Performance a

Updated Apri

Fac
51.2
Wha

Updated Apri

Eve
63.7
Wha

Updated Apri

Am
10.6
Wha

●●●●● AT&T LTE

Updated April 2

Goog
13.1 M
Apr 20

Crash fixes and improvements.

Starb
33.7 M
What's

Shaz
41.4 M
What's

Insta
20.5 M
What's

Updated April 1

●●●●● AT&T LTE

Upd

Updated April 22, 201

AP Mobile
23.3 MB
Apr 20, 2015

* Support for Apple Wa with breaking news no touch to save a story t
* Improved story sharir AP's Bigstory.ap.org si
* Bug fixes and the usu enhancements.

Thanks for your suppo

TripIt - Trave
15.6 MB
What's New ▾

Updated April 20, 201

Movies by F
49.5 MB
What's New ▾

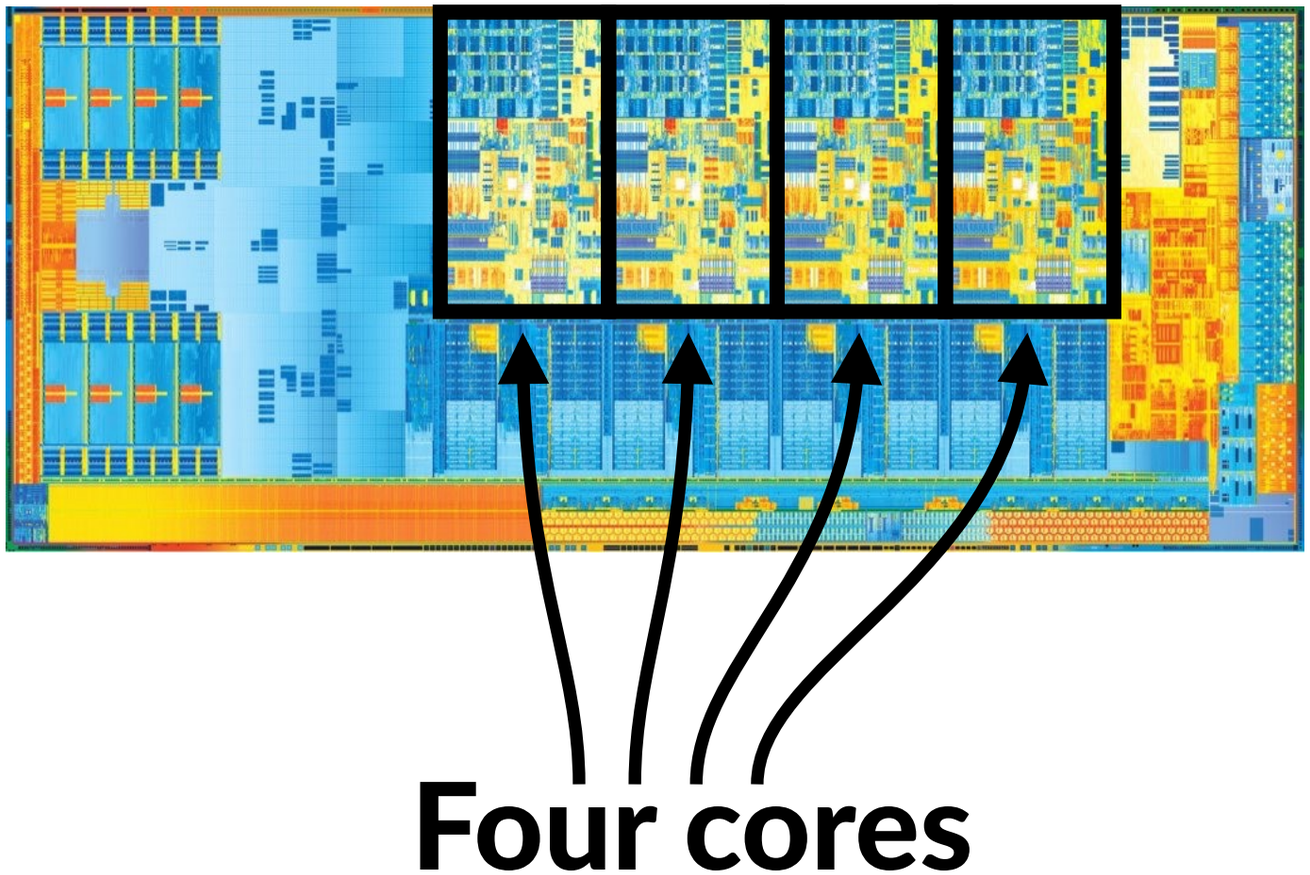
Pendi

• Bug

Featured

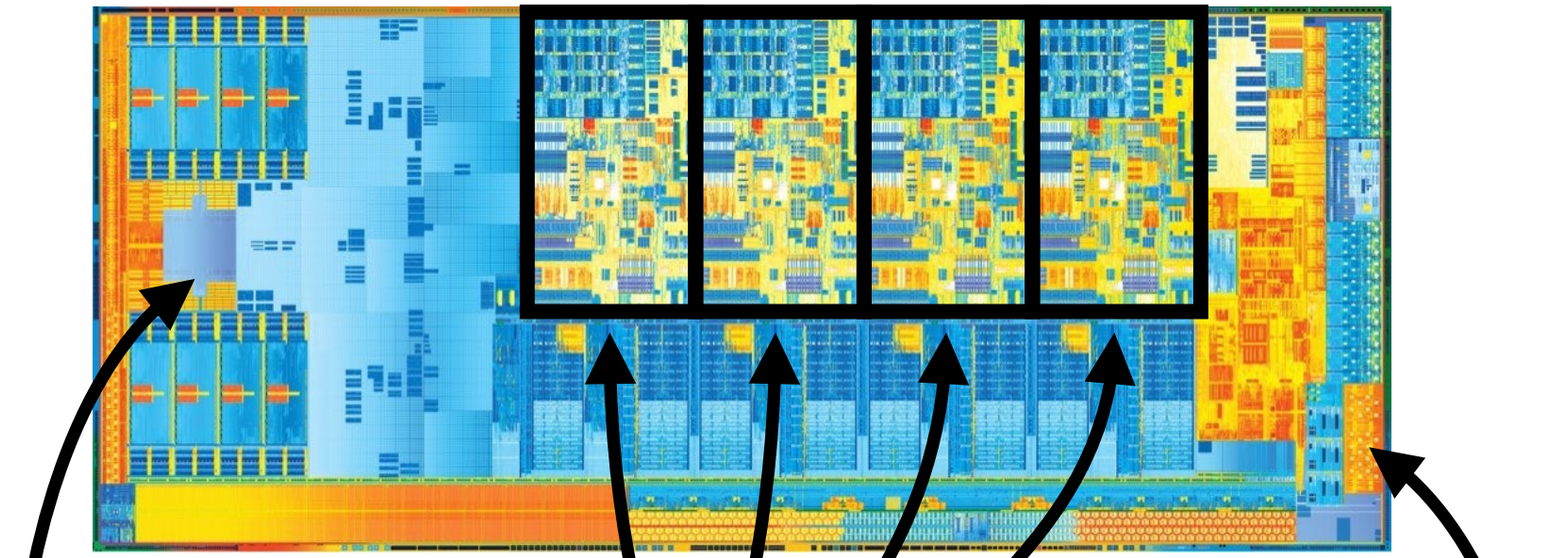
Where do all those transistors go?

Instead of faster processors, we just get more.



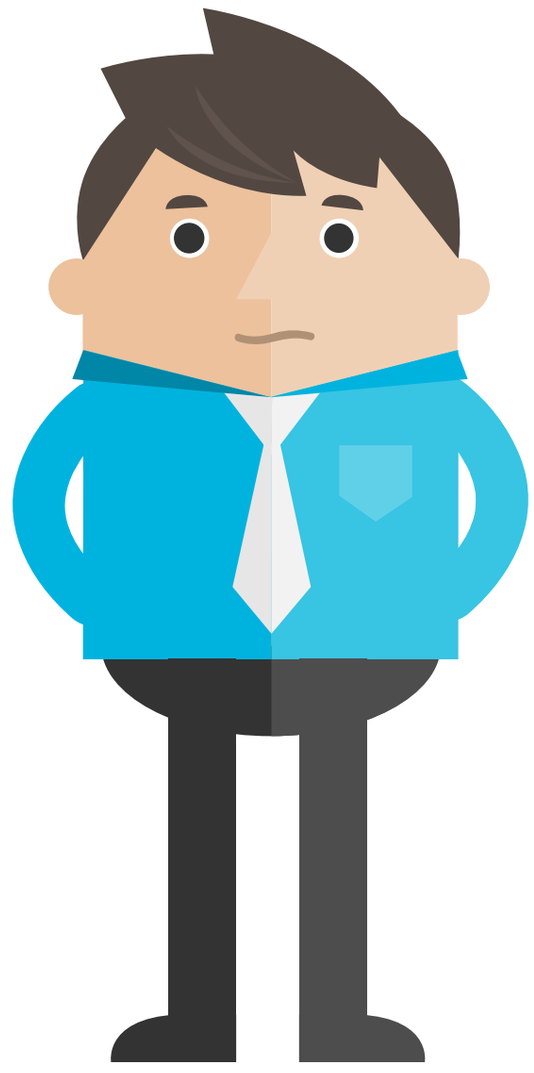
Where do all those transistors go?

Instead of faster processors, we just get more.



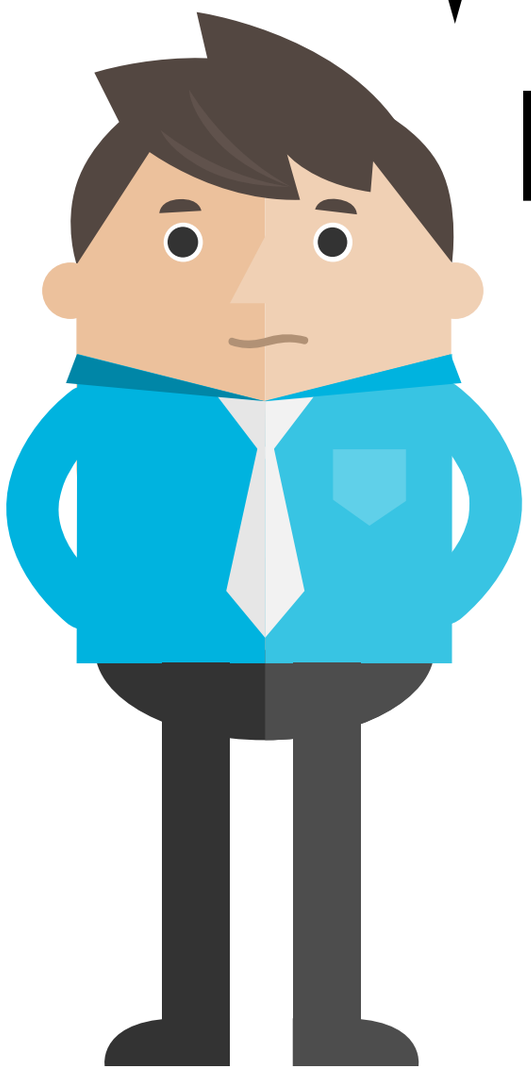
Four cores

and a lot of extra “stuff”




Why is this so hard?

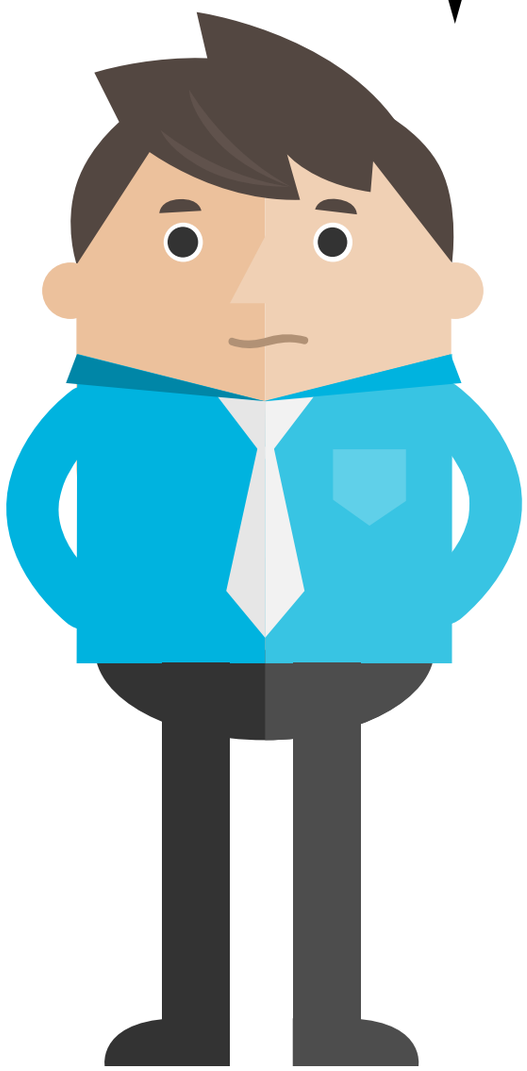
Why is this so hard?



Performance Optimization

current approaches out of steam

Why is this so hard?



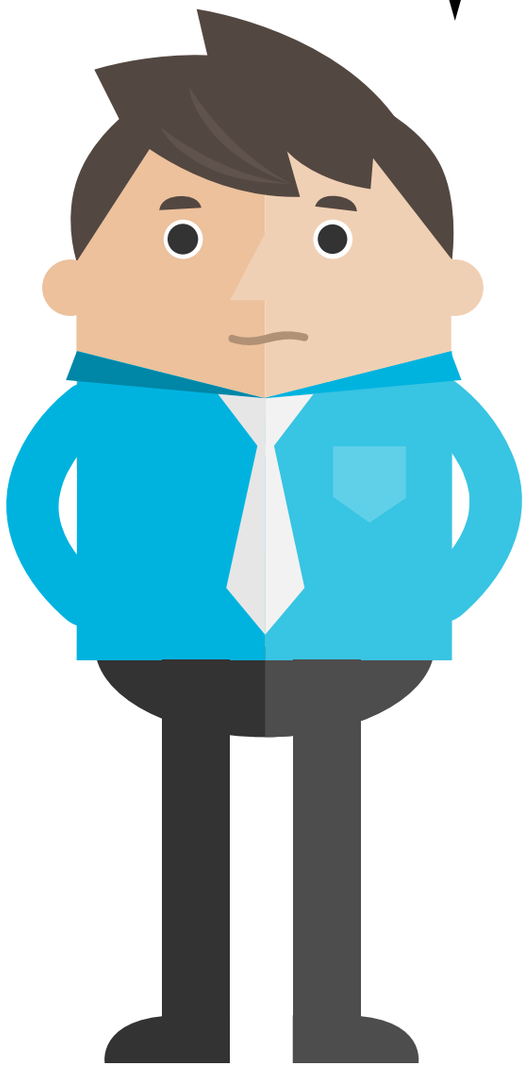
Performance Analysis

current approaches unsound

Performance Profiling

current approaches ineffective

Why is this so hard?



Performance Analysis

current approaches unsound

STABILIZER

We've been doing it wrong

We've been doing it wrong

Memory layout affects performance

We've been doing it wrong

Memory layout affects performance

changing a program will change its layout

We've been doing it wrong

Memory layout affects performance

changing a program will change its layout

no way to measure effect of change in isolation

We've been doing it wrong

Memory layout affects performance

changing a program will change its layout

no way to measure effect of change in isolation

STABILIZER eliminates the effect of layout

We've been doing it wrong

Memory layout affects performance

changing a program will change its layout

no way to measure effect of change in isolation

STABILIZER eliminates the effect of layout

enables sound performance evaluation

We've been doing it wrong

Memory layout affects performance

changing a program will change its layout

no way to measure effect of change in isolation

STABILIZER eliminates the effect of layout

enables sound performance evaluation

Case Studies

We've been doing it wrong

Memory layout affects performance

changing a program will change its layout

no way to measure effect of change in isolation

STABILIZER eliminates the effect of layout

enables sound performance evaluation

Case Studies

evaluation of LLVM's optimizations with STABILIZER

We've been doing it wrong

Memory layout affects performance

changing a program will change its layout

no way to measure effect of change in isolation

STABILIZER eliminates the effect of layout

enables sound performance evaluation

Case Studies

evaluation of LLVM's optimizations with STABILIZER

Unsound performance evaluation

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}


 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
    asm("isync");
}
```

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

Unsound performance evaluation

A

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}


 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
    asm("isync");
}
```

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

Unsound performance evaluation

A

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}


 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    size_t meaning_of_life=42;
    for (size_t i = 0; i < size; i += 32) {
        asm("icbi 0,%0" : : "r"(p));
        p += 32;
    }
    for (size_t i = 16; i < size; i += 32) {
        asm("icbi 0,%0" : : "r"(p));
        p += 32;
    }
    asm("isync");
}
```

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

Unsound performance evaluation

A

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}


 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    size_t meaning_of_life=42;
    for (size_t i = 0; i < size; i += 32) {
        asm("icbi 0,%0" : : "r"(p));
        p += 32;
    }
    for (size_t i = 16; i < size; i += 32) {
        asm("icbi 0,%0" : : "r"(p));
        p += 32;
    }
    asm("isync");
}
```

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

Unsound performance evaluation

A

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}


 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    size_t meaning_of_life=42;
    for (size_t i = 0; i < size; i += 32) {
        asm("icbi 0,%0" : : "r"(p));
        p += 32;
    }
    for (size_t i = 16; i < size; i += 32) {
        asm("icbi 0,%0" : : "r"(p));
        p += 32;
    }
    asm("isync");
}
```

Unsound performance evaluation

A

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    size_t meaning_of_life=42;
    for (size_t i = 0; i < size; i += 32) {
        asm("icbi 0,%0" : : "r"(p));
        p += 32;
    }
    for (size_t i = 16; i < size; i += 32) {
        asm("icbi 0,%0" : : "r"(p));
        p += 32;
    }
    asm("isync");
}
```

```
void setTimer(int msec) {
    struct itimerval timer;

    timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
    timer.it_value.tv_usec = 1000 * (msec % 1000);
    timer.it_interval.tv_sec = 0;
    timer.it_interval.tv_usec = 0;

    setitimer(ITIMER_REAL, &timer, 0);
}
```

Unsound performance evaluation

A

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    size_t meaning_of_life=42;
    for (size_t i = 0; i < size; i += 32) {
        asm("icbi 0,%0" : : "r"(p));
        p += 32;
    }
    for (size_t i = 16; i < size; i += 32) {
        asm("icbi 0,%0" : : "r"(p));
        p += 32;
    }
    asm("isync");
}
```

```
void setTimer(int msec) {
    struct itimerval timer;

    timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
    timer.it_value.tv_usec = 1000 * (msec % 1000);
    timer.it_interval.tv_sec = 0;
    timer.it_interval.tv_usec = 0;

    setitimer(ITIMER_REAL, &timer, 0);
}
```

Unsound performance evaluation

A

A′

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}
```

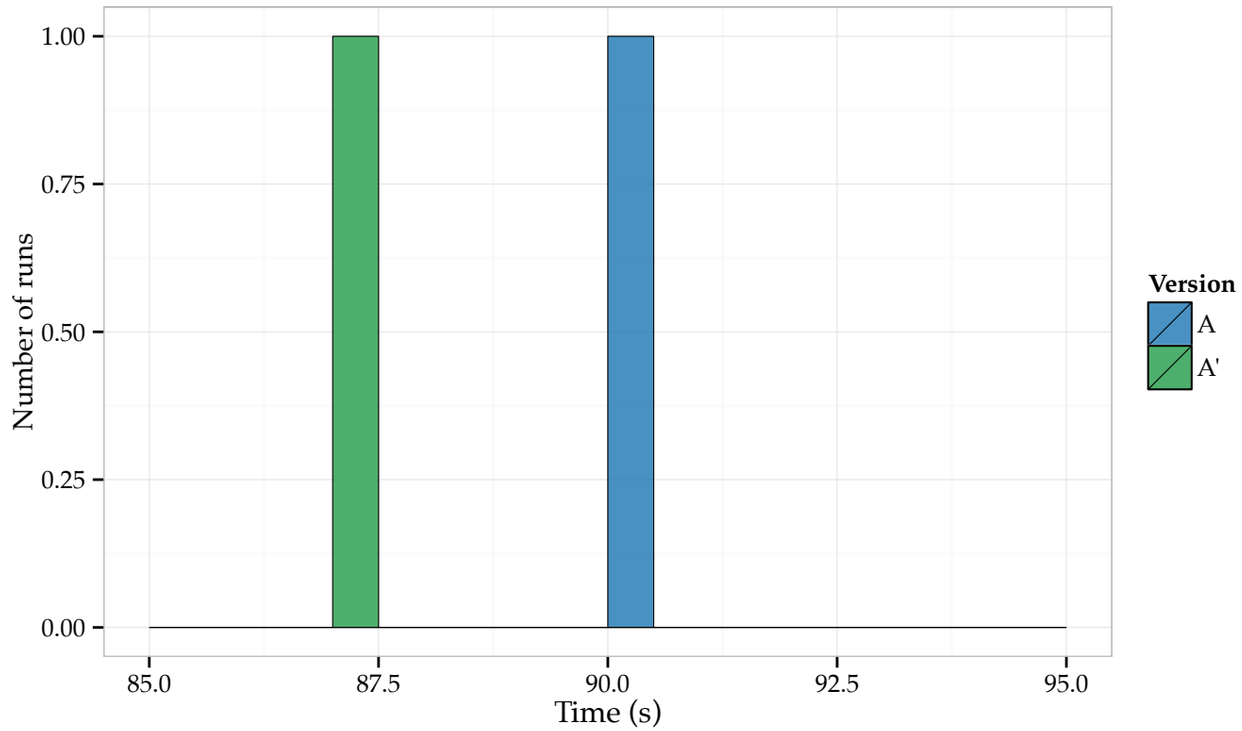
```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    size_t meaning_of_life=42;
    for (size_t i = 0; i < size; i += 32) {
        asm("icbi 0,%0" : : "r"(p));
        p += 32;
    }
    for (size_t i = 16; i < size; i += 32) {
        asm("icbi 0,%0" : : "r"(p));
        p += 32;
    }
    asm("isync");
}
```

```
void setTimer(int msec) {
    struct itimerval timer;

    timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
    timer.it_value.tv_usec = 1000 * (msec % 1000);
    timer.it_interval.tv_sec = 0;
    timer.it_interval.tv_usec = 0;

    setitimer(ITIMER_REAL, &timer, 0);
}
```

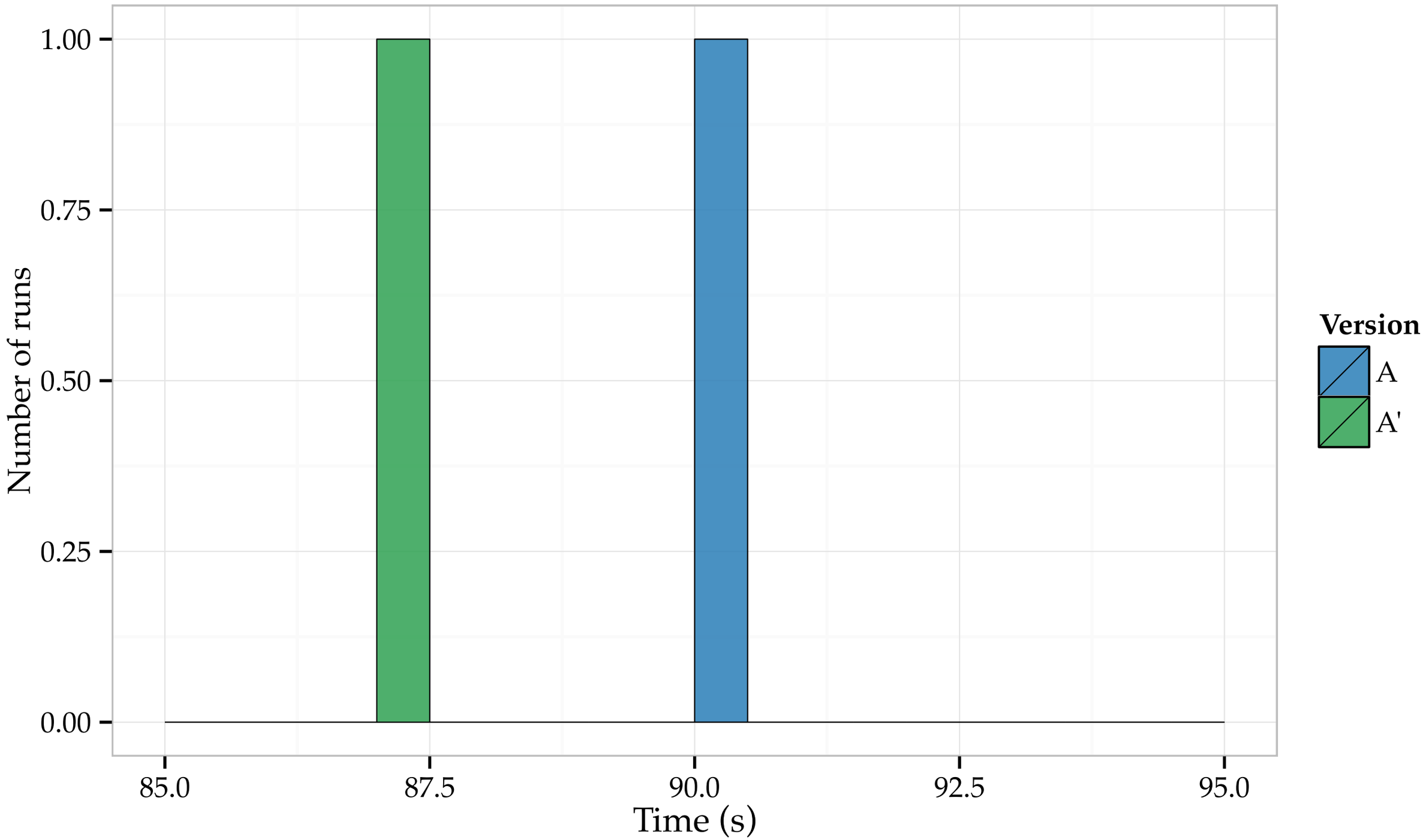
Which is faster?

A ×1

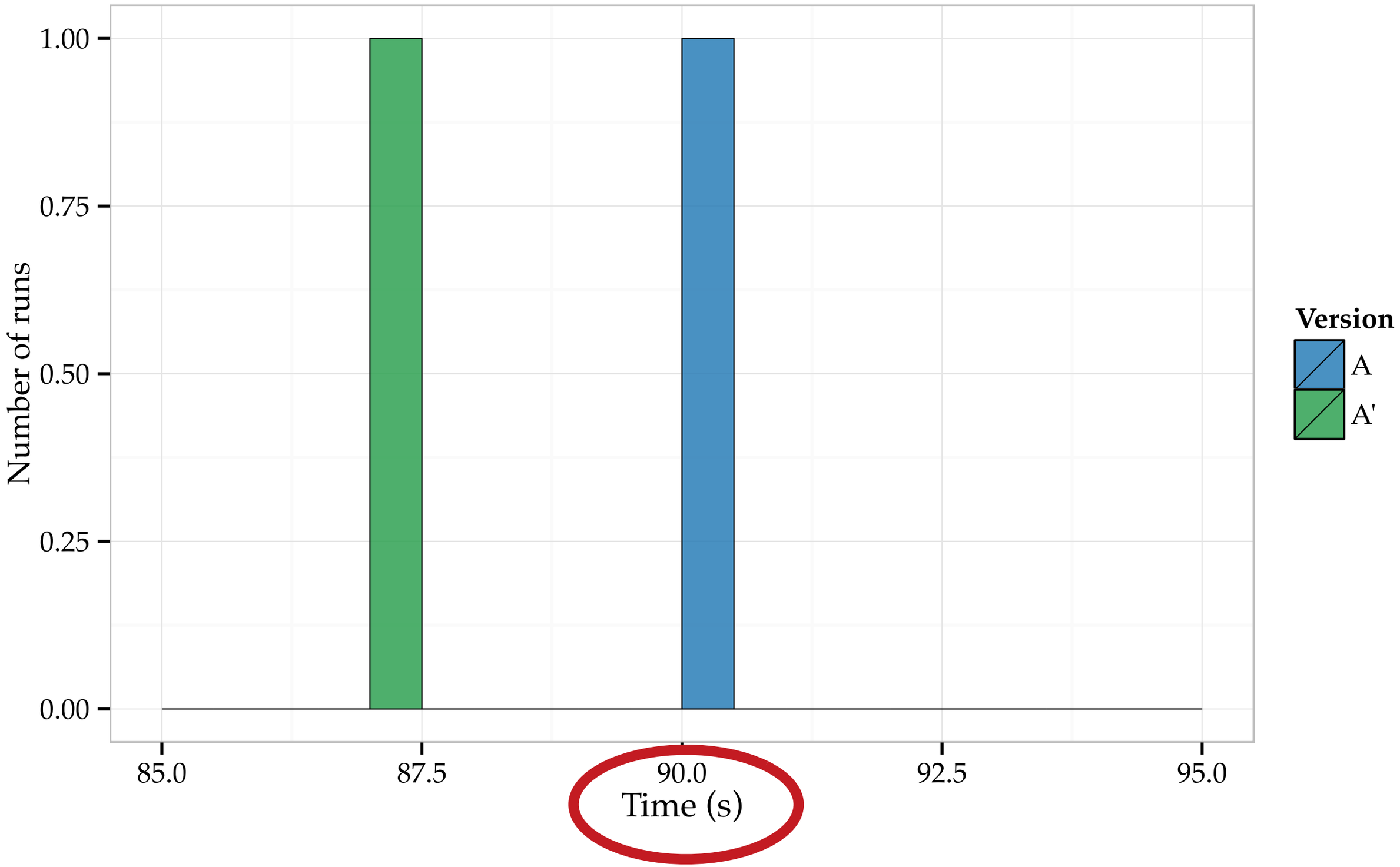
A′ ×1



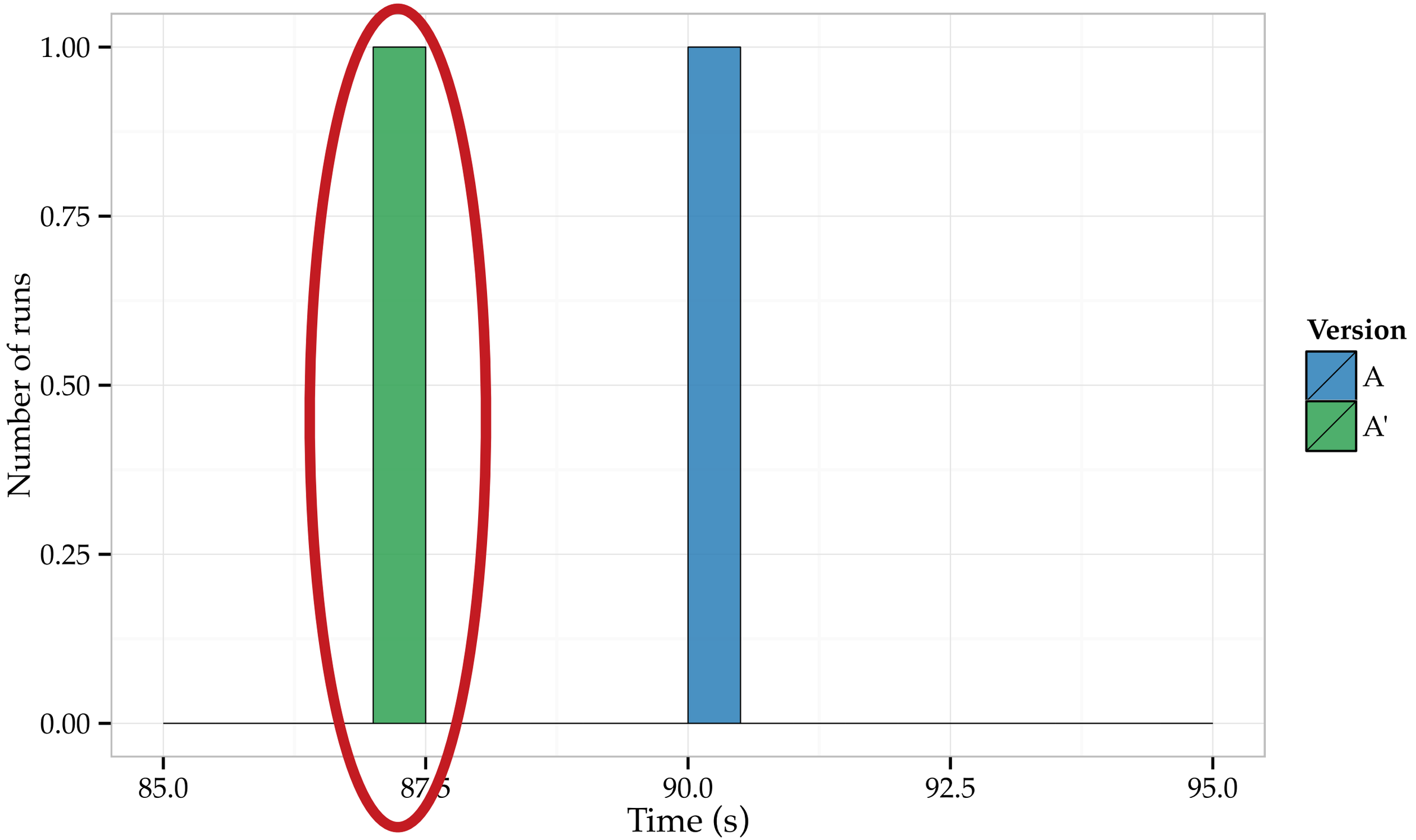
Is A′ faster than A?



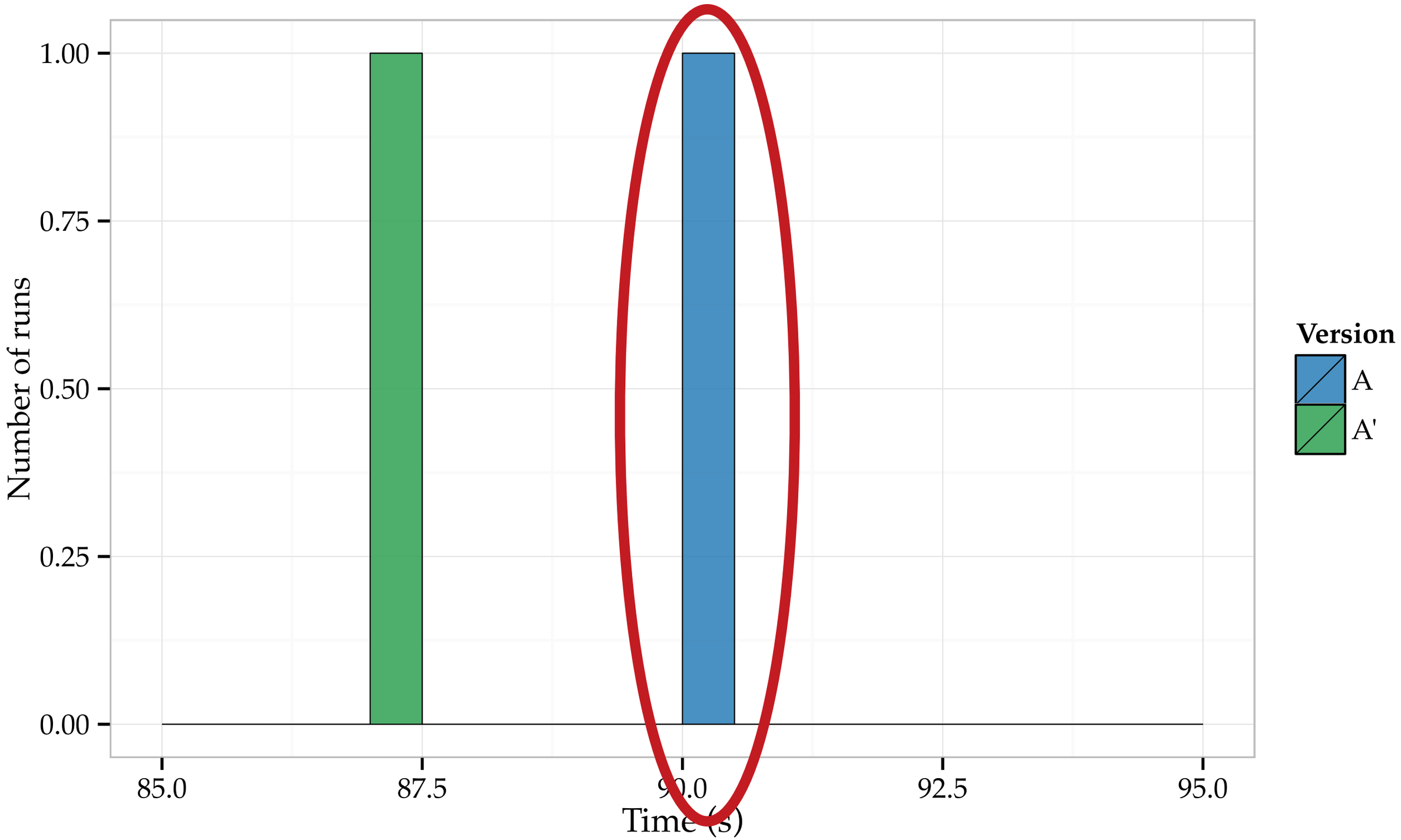
Is A′ faster than A ?



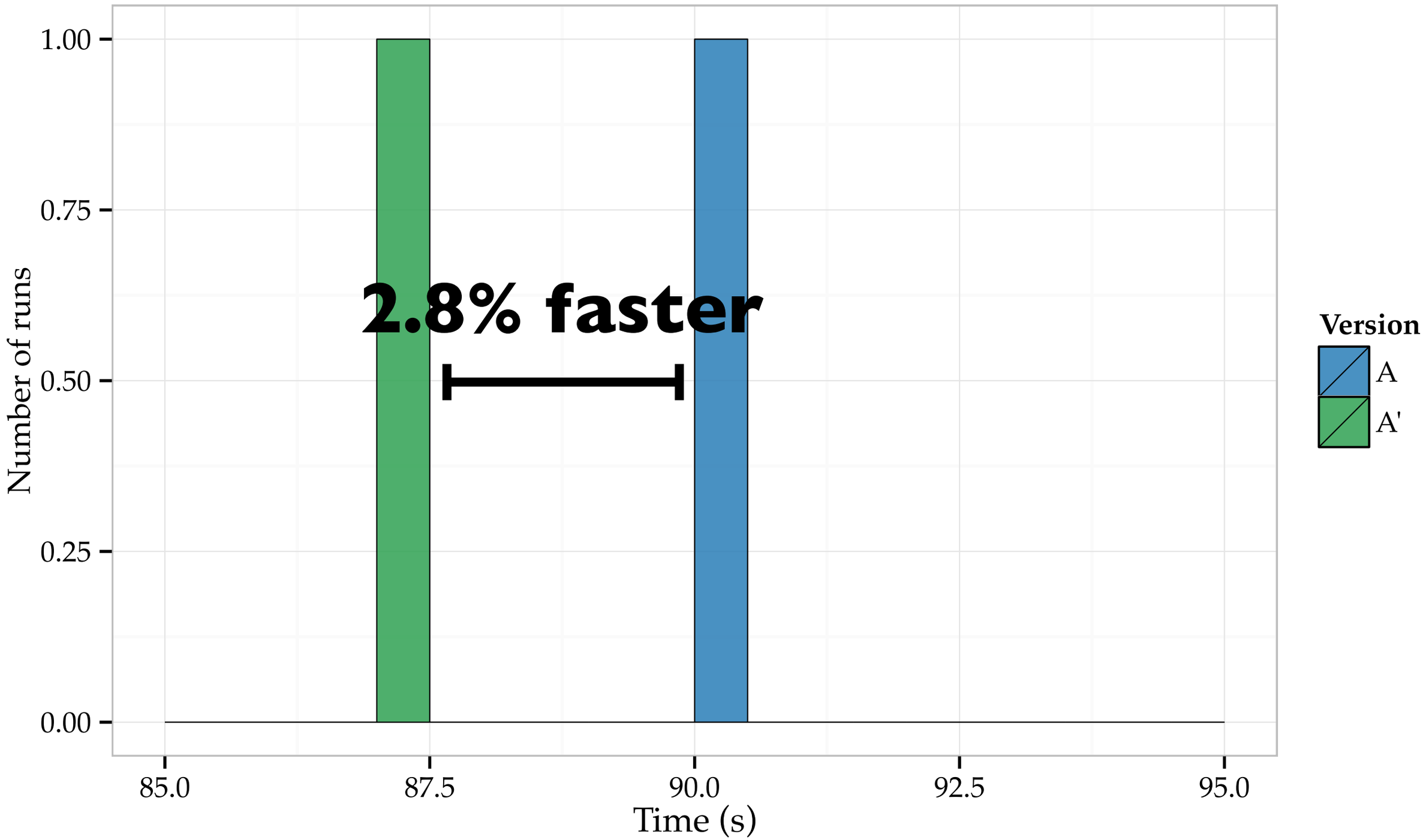
Is A′ faster than A?



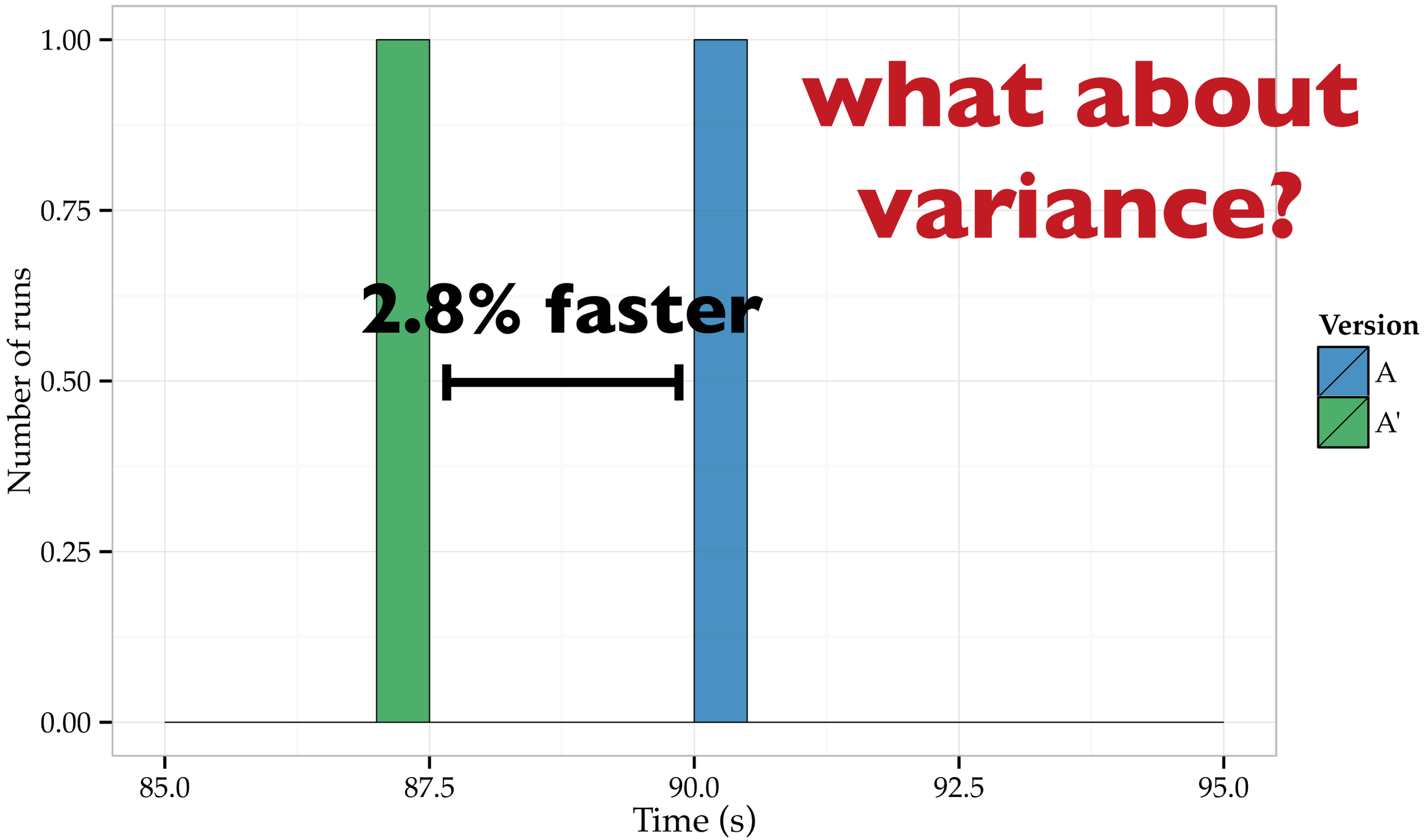
Is A′ faster than A?



Is A′ faster than A?

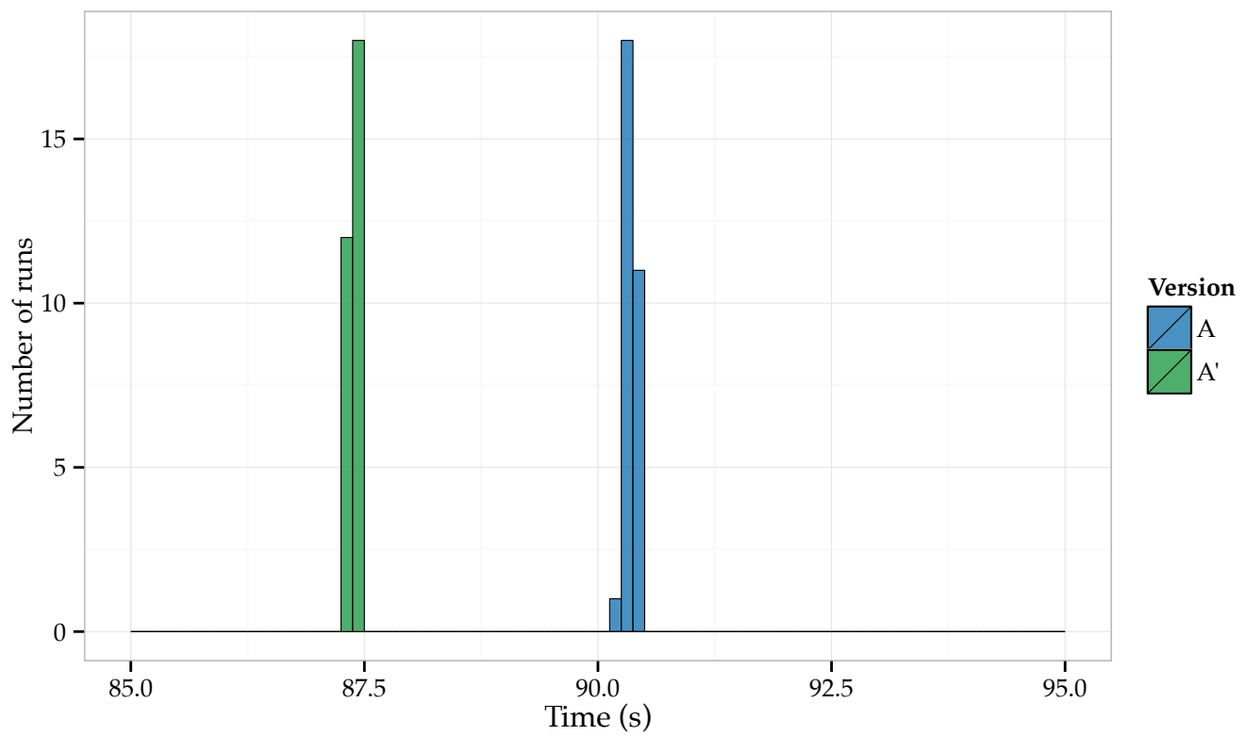


Is A′ faster than A ?

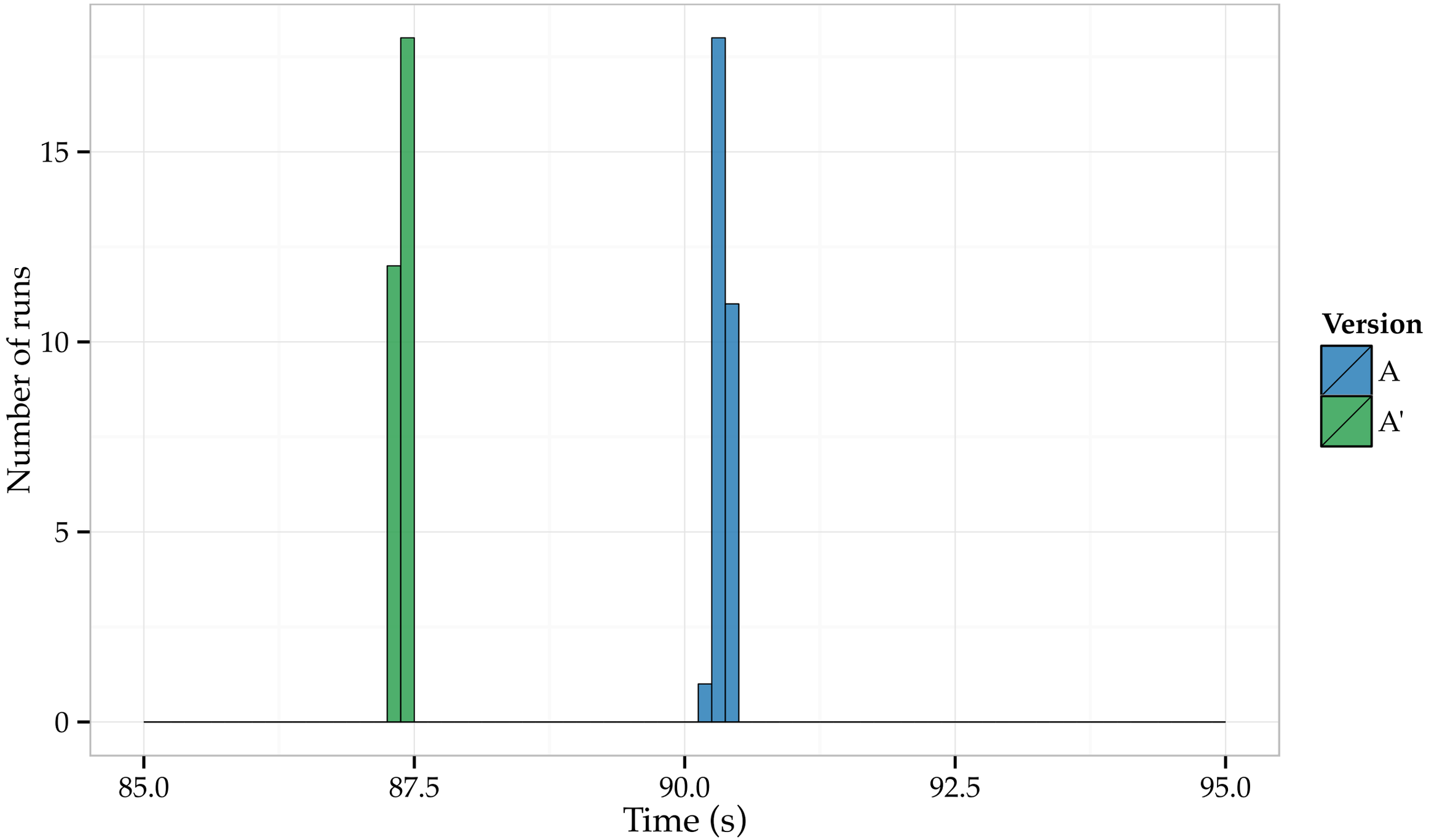


Which is faster?

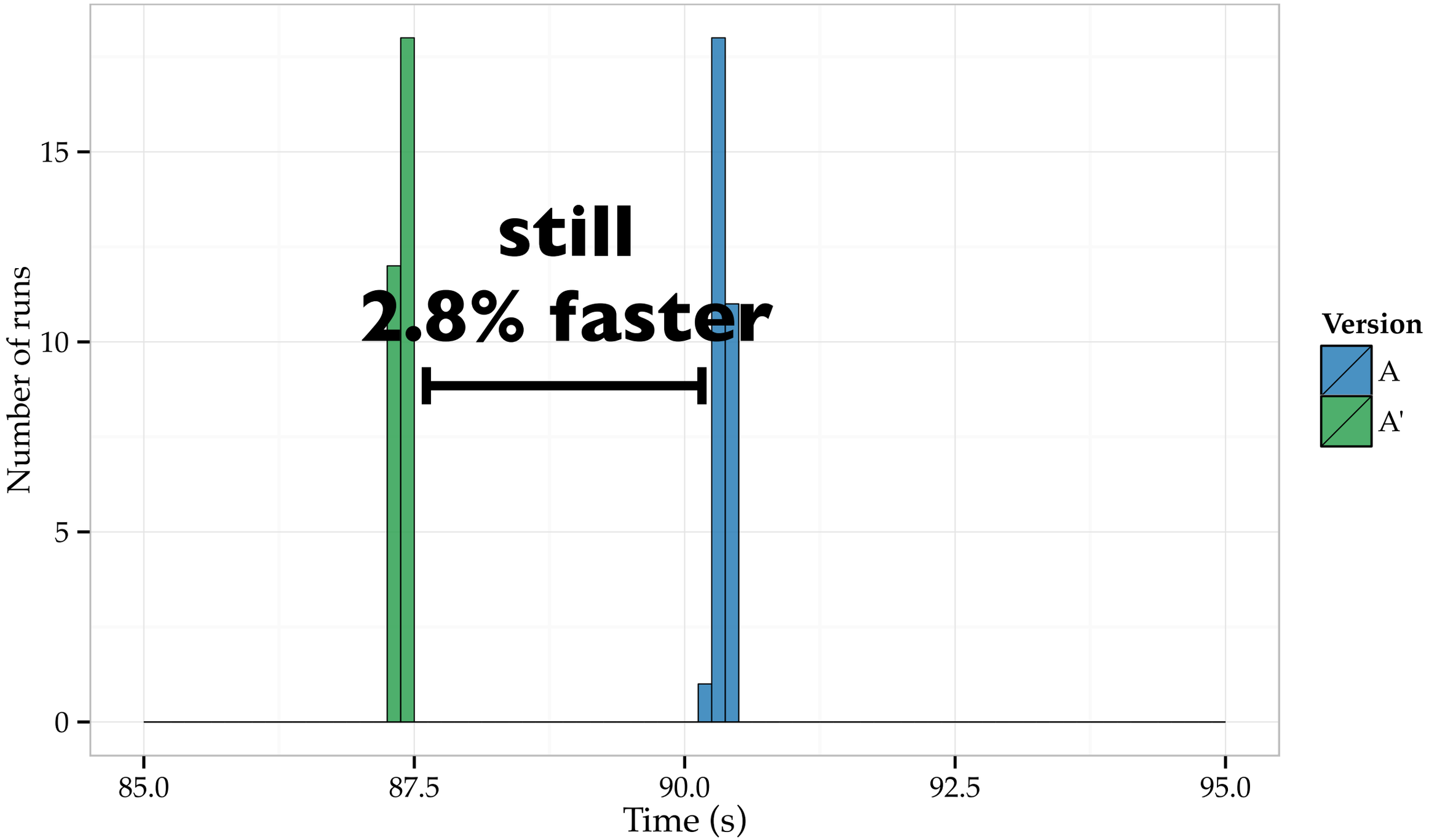
A ×30 A′ ×30



Is A′ faster than A?



Is A′ faster than A ?



Why is A′ faster than A ?

Why is A′ faster than A ?

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
}
```

```
void setTimer(int msec) {
    struct itimerval timer;

    timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
    timer.it_value.tv_usec = 1000 * (msec % 1000);
    timer.it_interval.tv_sec = 0;
    timer.it_interval.tv_usec = 0;

    setitimer(ITIMER_REAL, &timer, 0);
}
```

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}


void setTimer(int msec) {
    struct itimerval timer;

    timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
    timer.it_value.tv_usec = 1000 * (msec % 1000);
    timer.it_interval.tv_sec = 0;
    timer.it_interval.tv_usec = 0;

    setitimer(ITIMER_REAL, &timer, 0);
}
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
    asm("isync");
}
```

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

Why is A′ faster than A ?

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
}
```

```
 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

Was it the code change?

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}


 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
    asm("isync");
}
```

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

Why is A′ faster than A ?

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
}
```

```
void setTimer(int msec) {
    struct itimerval timer;

    timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
    timer.it_value.tv_usec = 1000 * (msec % 1000);
    timer.it_interval.tv_sec = 0;
    timer.it_interval.tv_usec = 0;

    setitimer(ITIMER_REAL, &timer, 0);
}
```

Was it the code change?

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}
```

```
void setTimer(int msec) {
    struct itimerval timer;

    timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
    timer.it_value.tv_usec = 1000 * (msec % 1000);
    timer.it_interval.tv_sec = 0;
    timer.it_interval.tv_usec = 0;

    setitimer(ITIMER_REAL, &timer, 0);
}
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
    asm("isync");
}
```

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

Why is A′ faster than A ?

Or was it the new layout?

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
}
```

```
void setTimer(int msec) {
    struct itimerval timer;

    timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
    timer.it_value.tv_usec = 1000 * (msec % 1000);
    timer.it_interval.tv_sec = 0;
    timer.it_interval.tv_usec = 0;

    setitimer(ITIMER_REAL, &timer, 0);
}
```

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}


void setTimer(int msec) {
    struct itimerval timer;

    timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
    timer.it_value.tv_usec = 1000 * (msec % 1000);
    timer.it_interval.tv_sec = 0;
    timer.it_interval.tv_usec = 0;

    setitimer(ITIMER_REAL, &timer, 0);
}
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
    asm("isync");
}
```

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

Why is A′ faster than A ?

Or was it the new layout?

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
}
```

```
void setTimer(int msec) {
    struct itimerval timer;

    timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
    timer.it_value.tv_usec = 1000 * (msec % 1000);
    timer.it_interval.tv_sec = 0;
    timer.it_interval.tv_usec = 0;

    setitimer(ITIMER_REAL, &timer, 0);
}
```

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}

void setTimer(int msec) {
    struct itimerval timer;

    timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
    timer.it_value.tv_usec = 1000 * (msec % 1000);
    timer.it_interval.tv_sec = 0;
    timer.it_interval.tv_usec = 0;

    setitimer(ITIMER_REAL, &timer, 0);
}
```

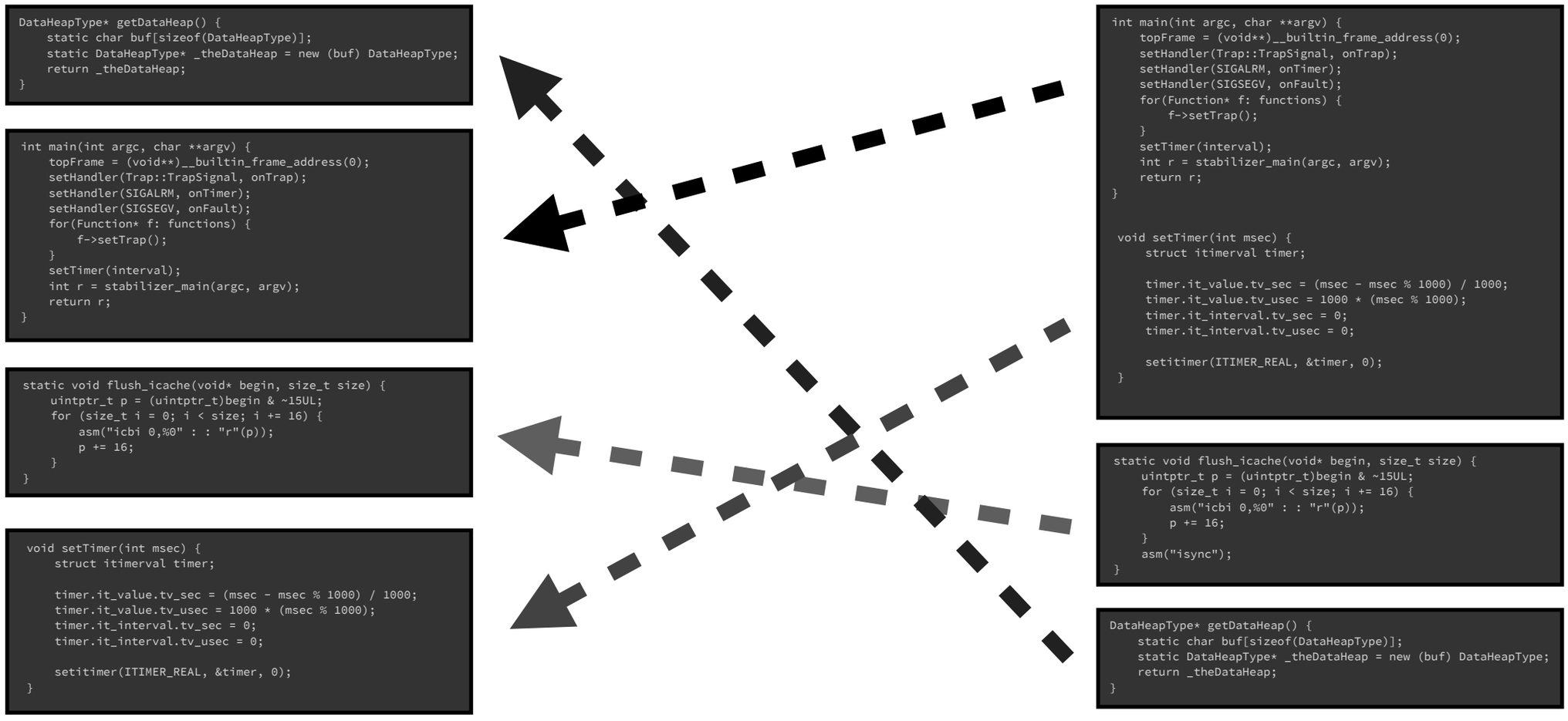
```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
    asm("isync");
}
```

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

Why is A′ faster than A ?

Layout biases measurement

Mytkowicz et al. (ASPLOS'09)



Layout biases measurement

Mytkowicz et al. (ASPLOS'09)

Layout biases measurement

Mytkowicz et al. (ASPLOS'09)

Link Order

Changes function addresses

Layout biases measurement

Mytkowicz et al. (ASPLOS'09)

Link Order

Changes function addresses

Environment Variable Size

Moves the program stack

Layout biases measurement

Mytkowicz et al. (ASPLOS'09)

Link Order

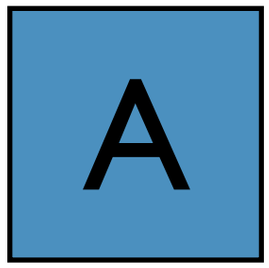
Changes function addresses

Larger than the impact of -O3

Environment Variable Size

Moves the program stack

Blame the cache




```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}


 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
    asm("isync");
}
```

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

Blame the cache

A

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}


 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

map to same cache set

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
    asm("isync");
}
```

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

Blame the cache

A

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}


 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
    asm("isync");
}
```

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

map to same cache set **conflict**

Blame the cache

A

A′

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}


 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
    asm("isync");
}
```

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
}
```

```
 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

Blame the cache

A

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}


 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
    asm("isync");
}
```

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

A′

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
}
```

```
 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

Blame the cache

A

A′

map to same set

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}


 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
    asm("isync");
}
```

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
}
```

```
 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

Blame the cache

A

A′

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}


 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
    asm("isync");
}
```

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

map to same set

Nothing here

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
}
```

```
 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

Blame the cache

A

A′

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}


 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
    asm("isync");
}
```

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

map to same set

Nothing here

no conflict

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
}
```

```
 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

Blame the cache

A

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}


 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

```
 static void flush_icache(void* begin, size_t size) {
     uintptr_t p = (uintptr_t)begin & ~15UL;
     for (size_t i = 0; i < size; i += 16) {
         asm("icbi 0,%0" : : "r"(p));
         p += 16;
     }
     asm("isync");
 }
```

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

A′

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

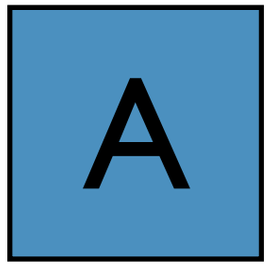
```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
}
```

```
 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

Blame the cache

or branch predictor




```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}


 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

```
 static void flush_icache(void* begin, size_t size) {
     uintptr_t p = (uintptr_t)begin & ~15UL;
     for (size_t i = 0; i < size; i += 16) {
         asm("icbi 0,%0" : : "r"(p));
         p += 16;
     }
     asm("isync");
 }
```

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}
```

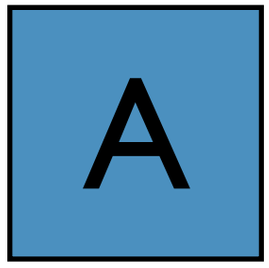
```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
}
```

```
 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

Blame the cache

or TLB

or branch predictor




```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}


 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

```
 static void flush_icache(void* begin, size_t size) {
     uintptr_t p = (uintptr_t)begin & ~15UL;
     for (size_t i = 0; i < size; i += 16) {
         asm("icbi 0,%0" : : "r"(p));
         p += 16;
     }
     asm("isync");
 }
```

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
}
```

```
 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

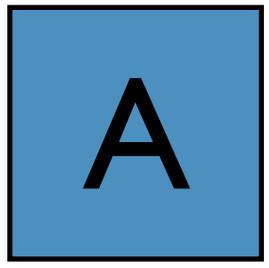
Blame the cache

or TLB

or branch predictor

or branch

target predictor




```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}


 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
    asm("isync");
}
```

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
}
```

```
 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

Blame the cache

or TLB

or branch predictor

or branch

target predictor

or prefetcher

A

A'

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}


 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
    asm("isync");
}
```

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

```
DataHeapType* getDataHeap() {
    c      [sizeof(DataHeapType)];
    sta  DataHeapType* _theDataHeap = new (buf) DataHeapType;
    re     _theDataHeap;
}
```

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
}
```

```
 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

Blame the **hash**

A

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}


 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

```
 static void flush_icache(void* begin, size_t size) {
     uintptr_t p = (uintptr_t)begin & ~15UL;
     for (size_t i = 0; i < size; i += 16) {
         asm("icbi 0,%0" : : "r"(p));
         p += 16;
     }
     asm("isync");
 }
```

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

A′

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}
```

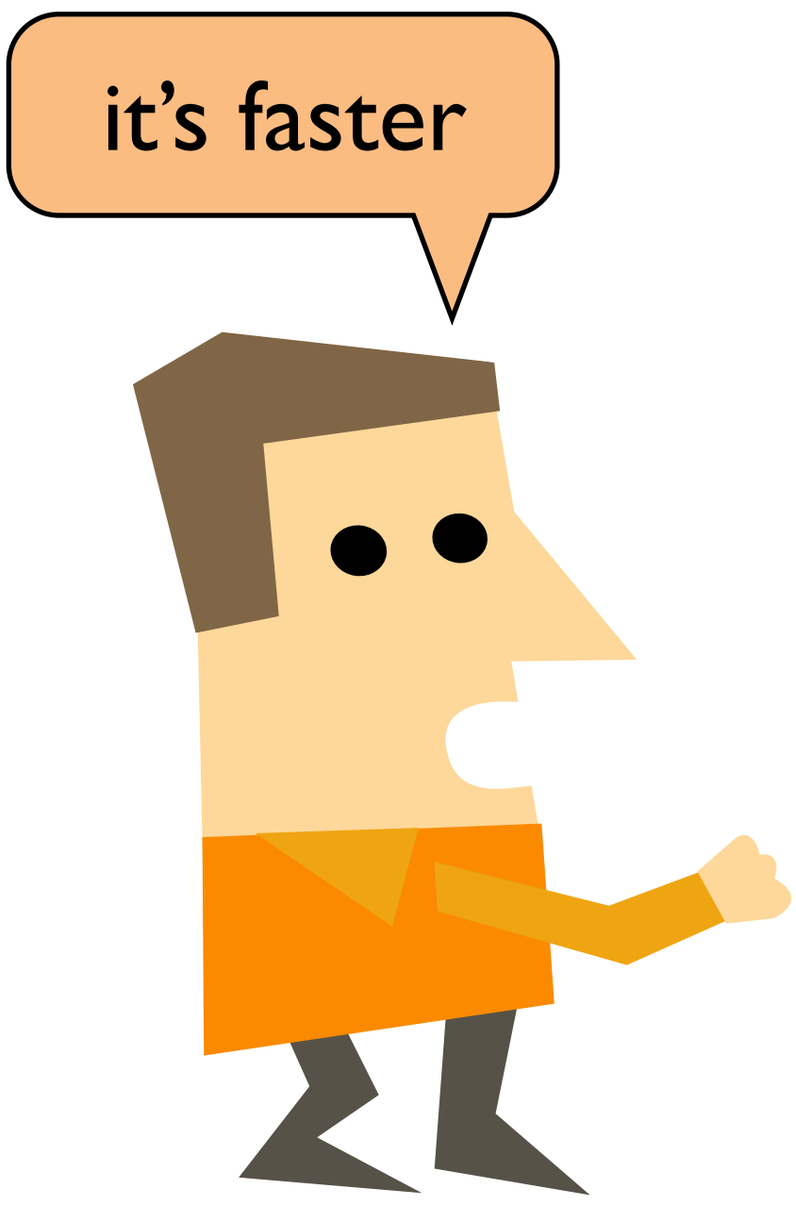
```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
}
```

```
 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

Is A′ faster than A ?

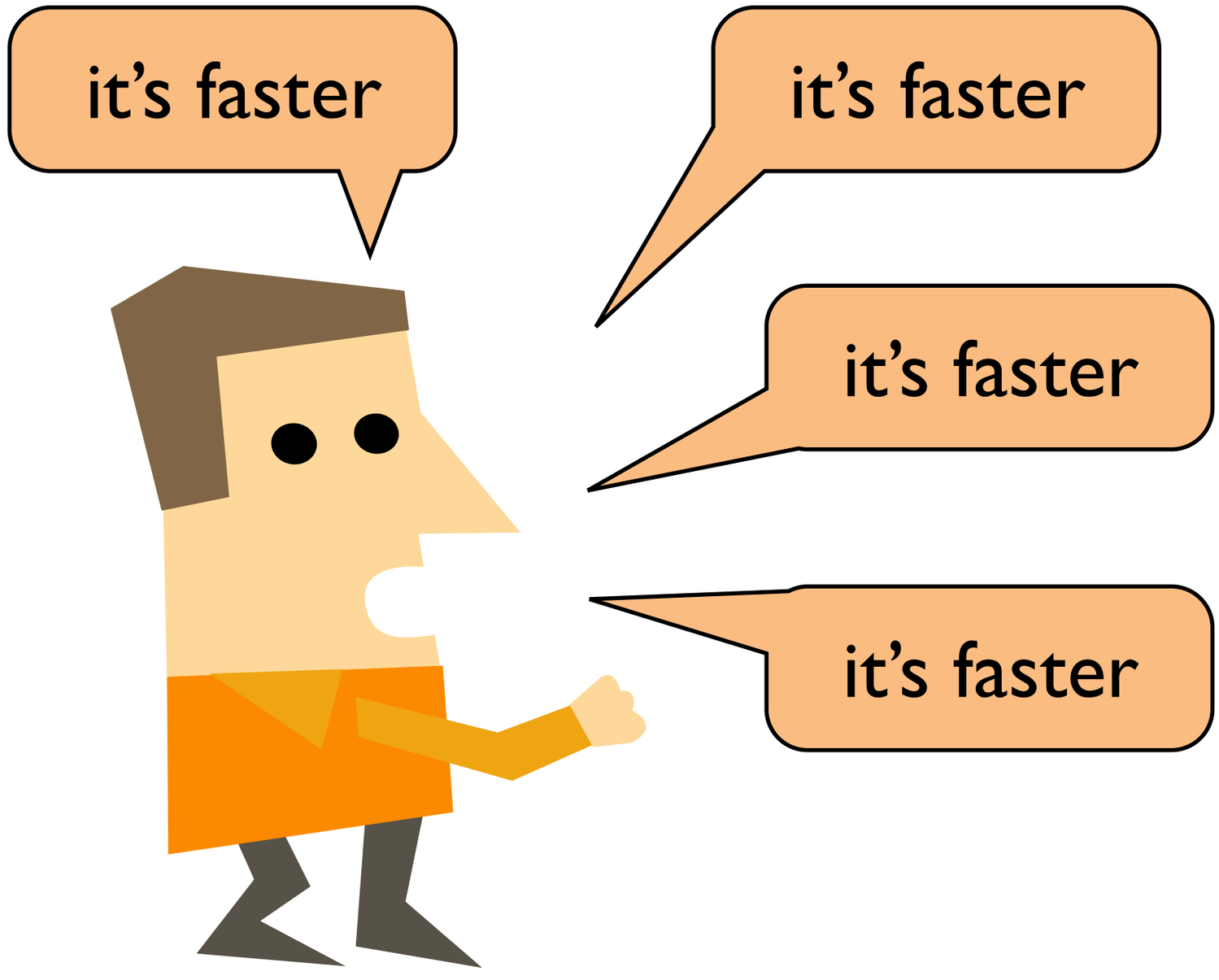
Is A′ faster than A ?

Let’s do a poll

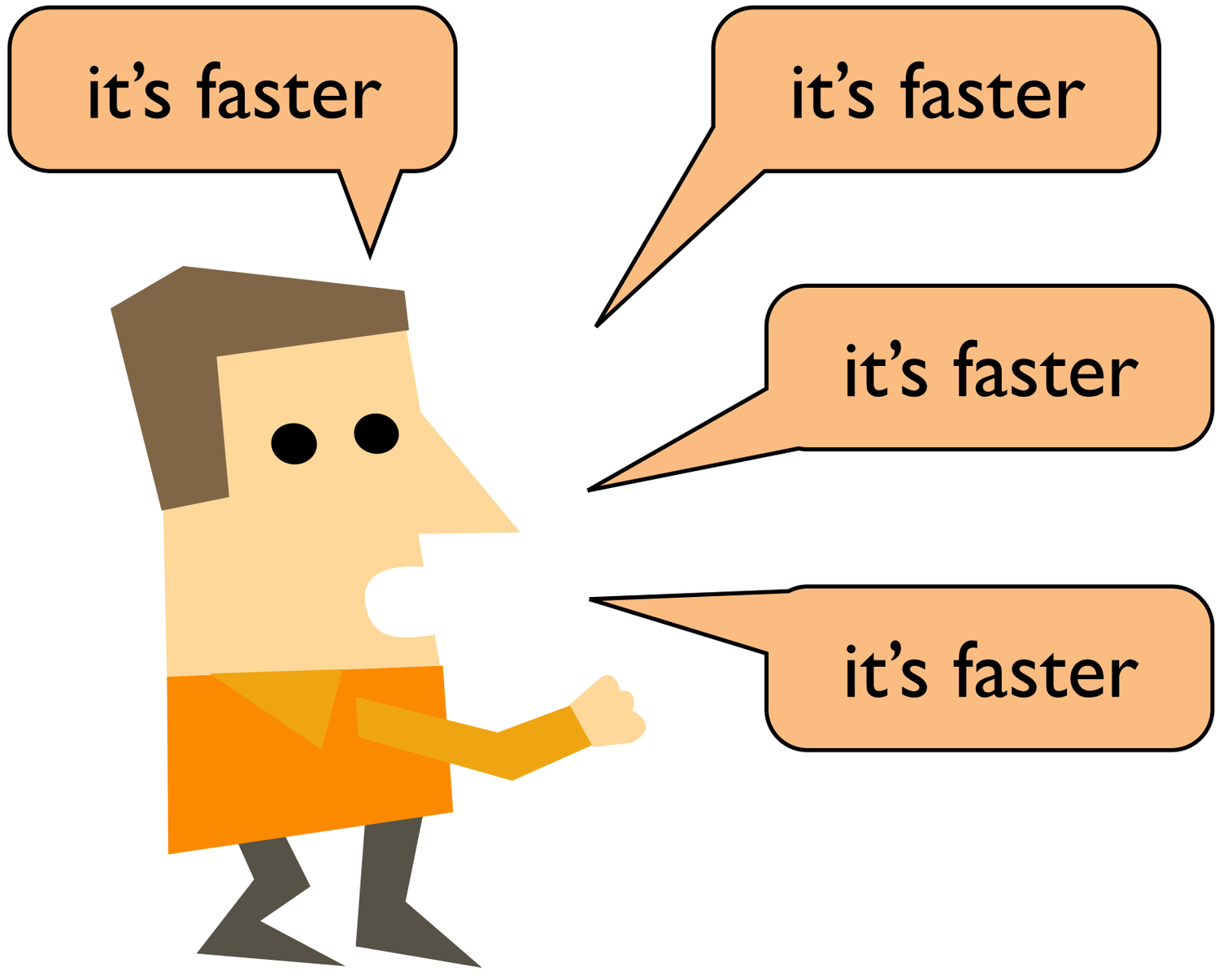


Is A′ faster than A ?

Let's do a poll

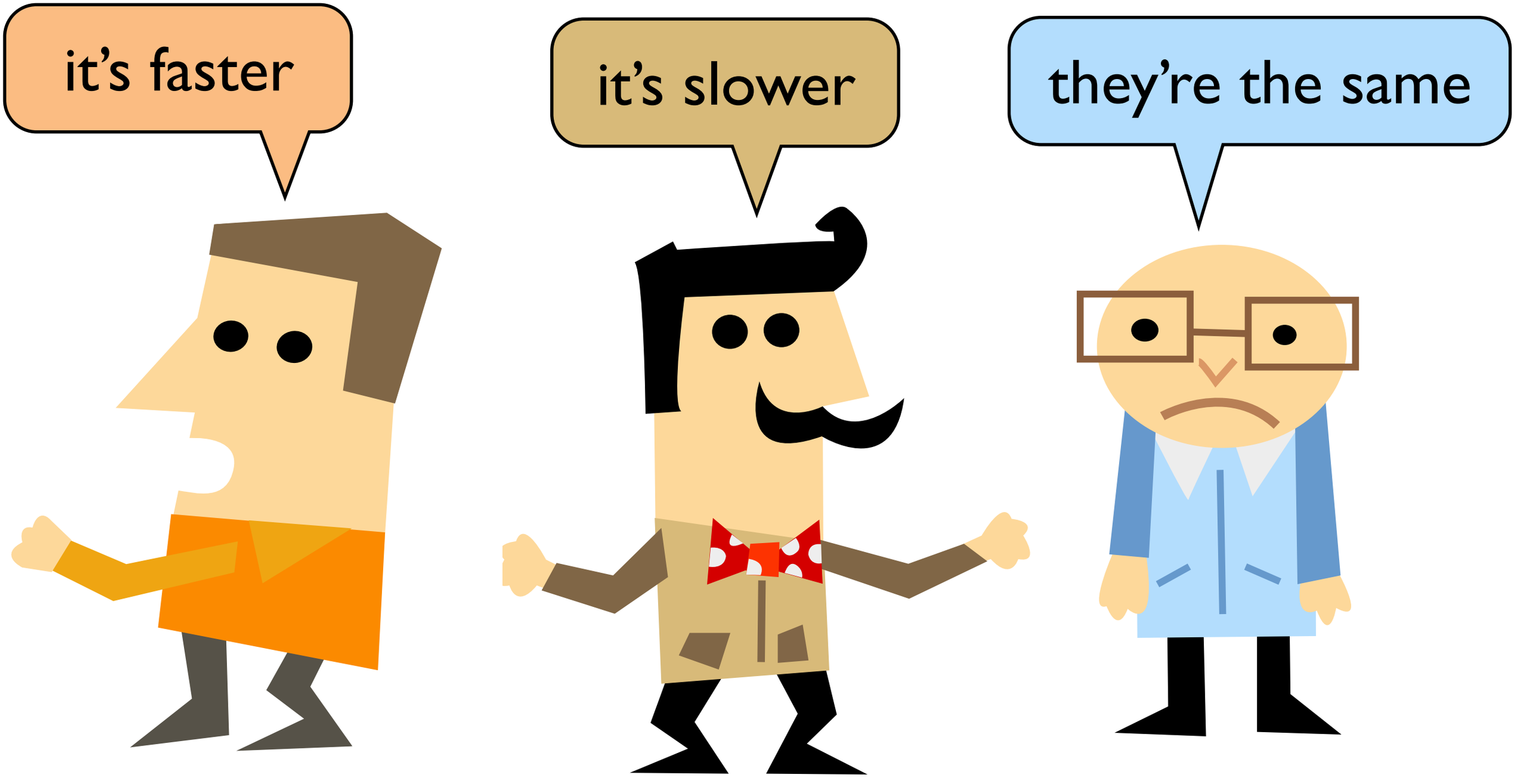


Do we trust this?

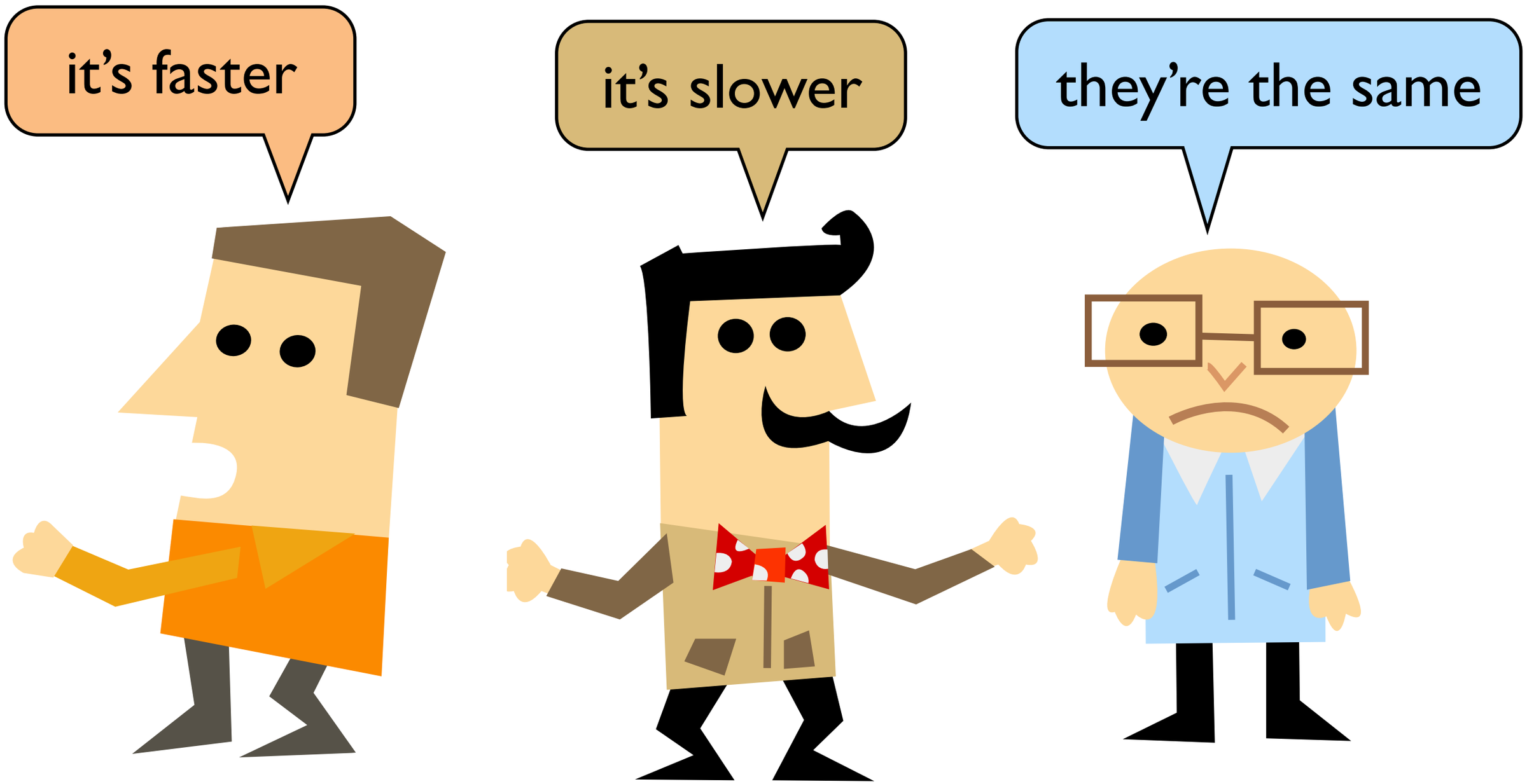


Is A′ faster than A ?

Is A′ faster than A ?

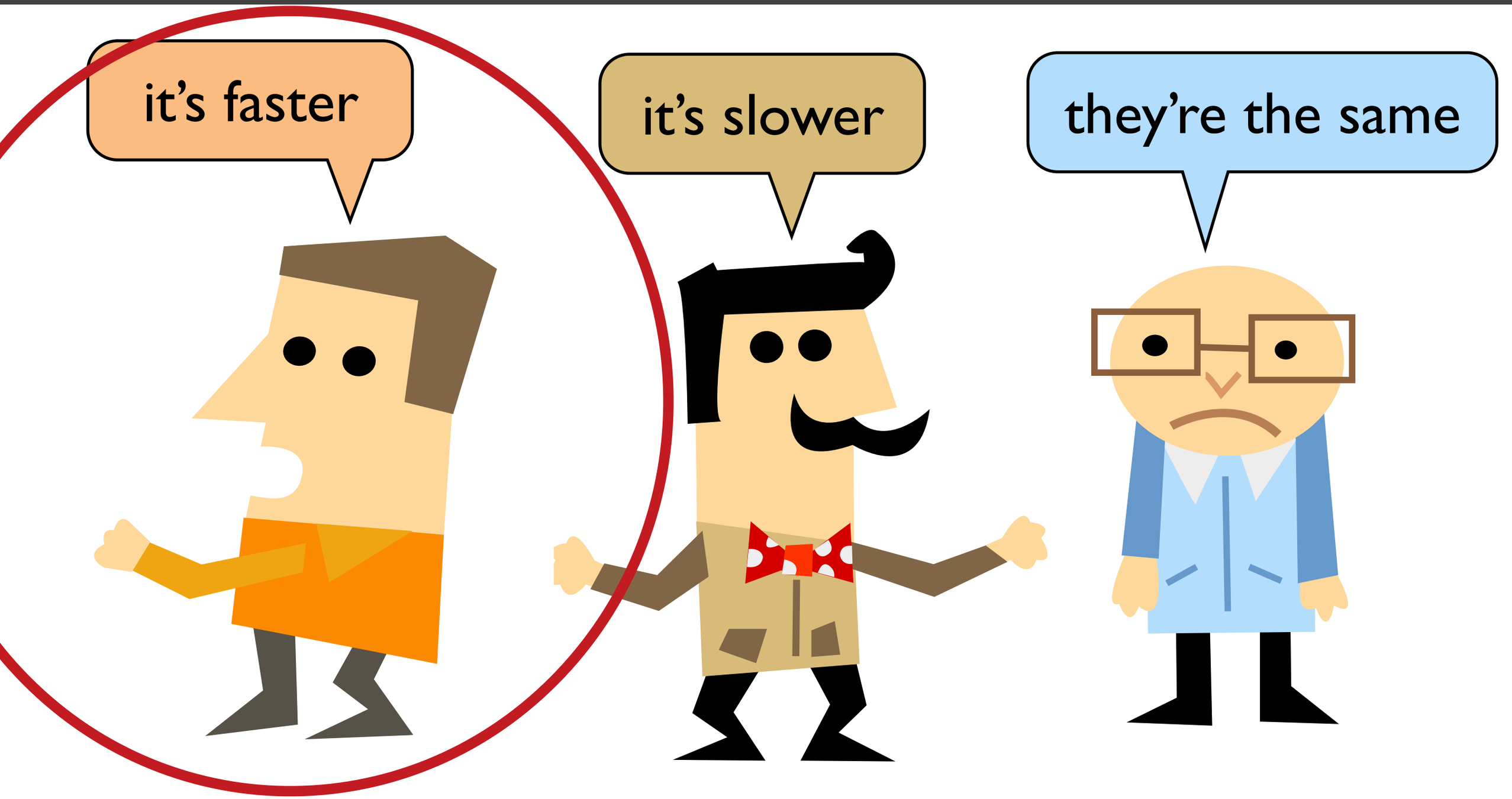


But it ran faster!



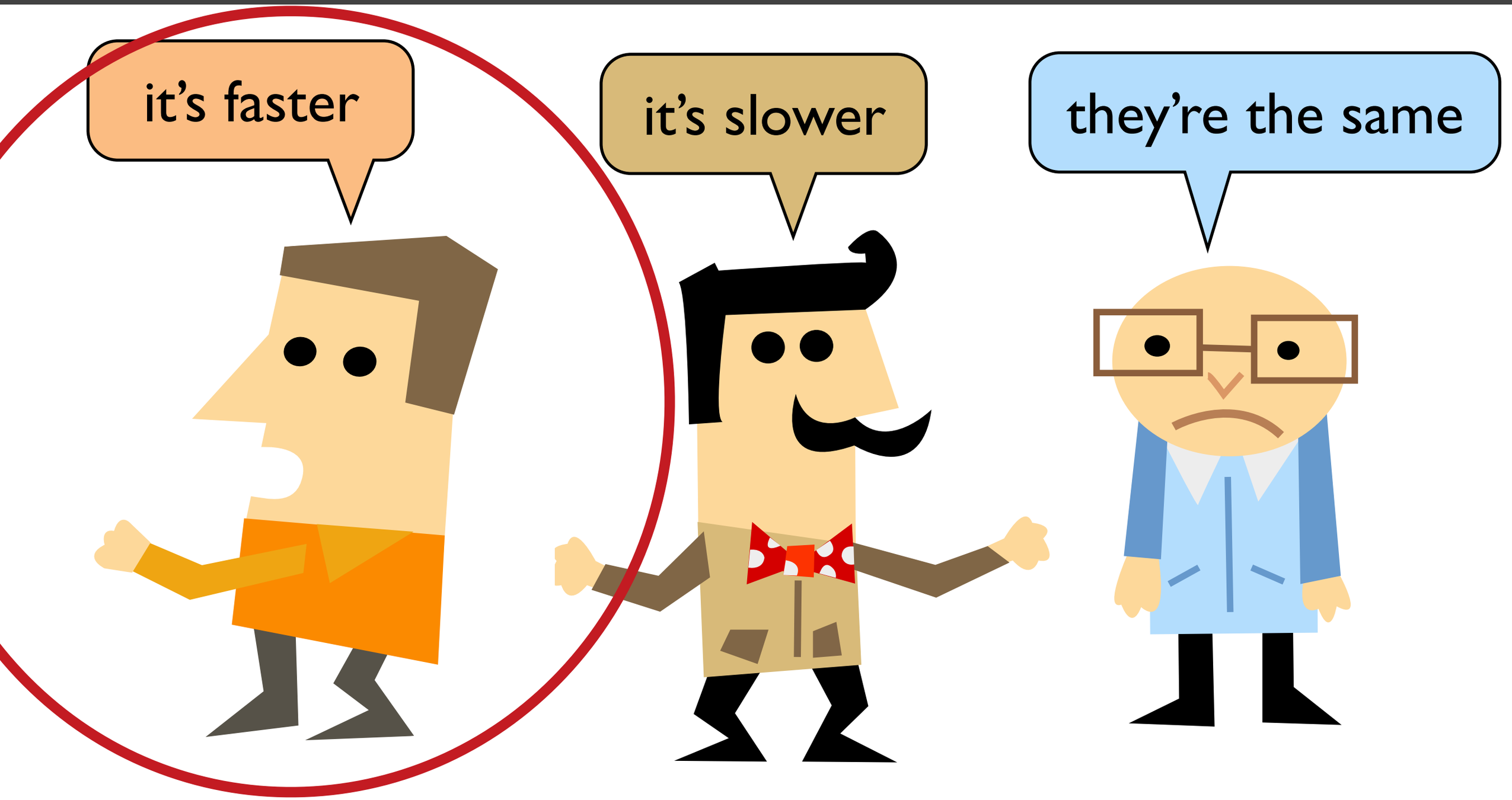
But it ran faster!

What if we only talk to Bob?



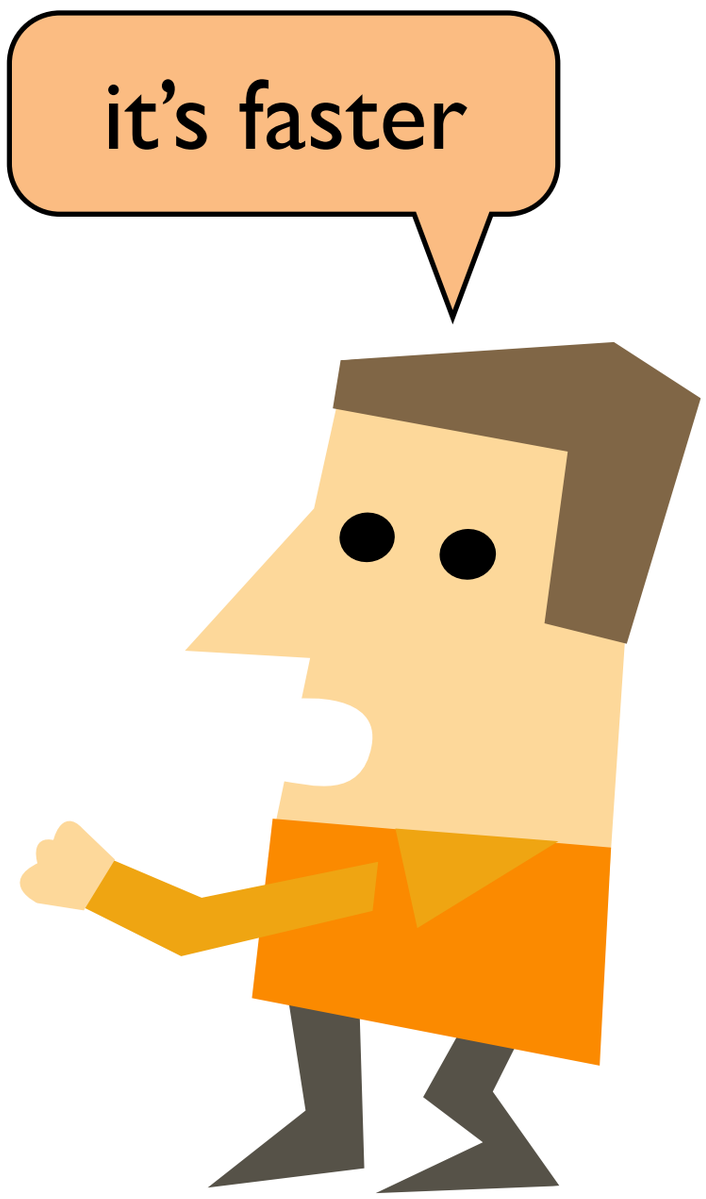
But it ran faster!

What if we only use this layout?



But it ran faster!

What if we only use this layout?



But it ran faster!

What if we only use this layout?

Upgrade `libc`

But it ran faster!

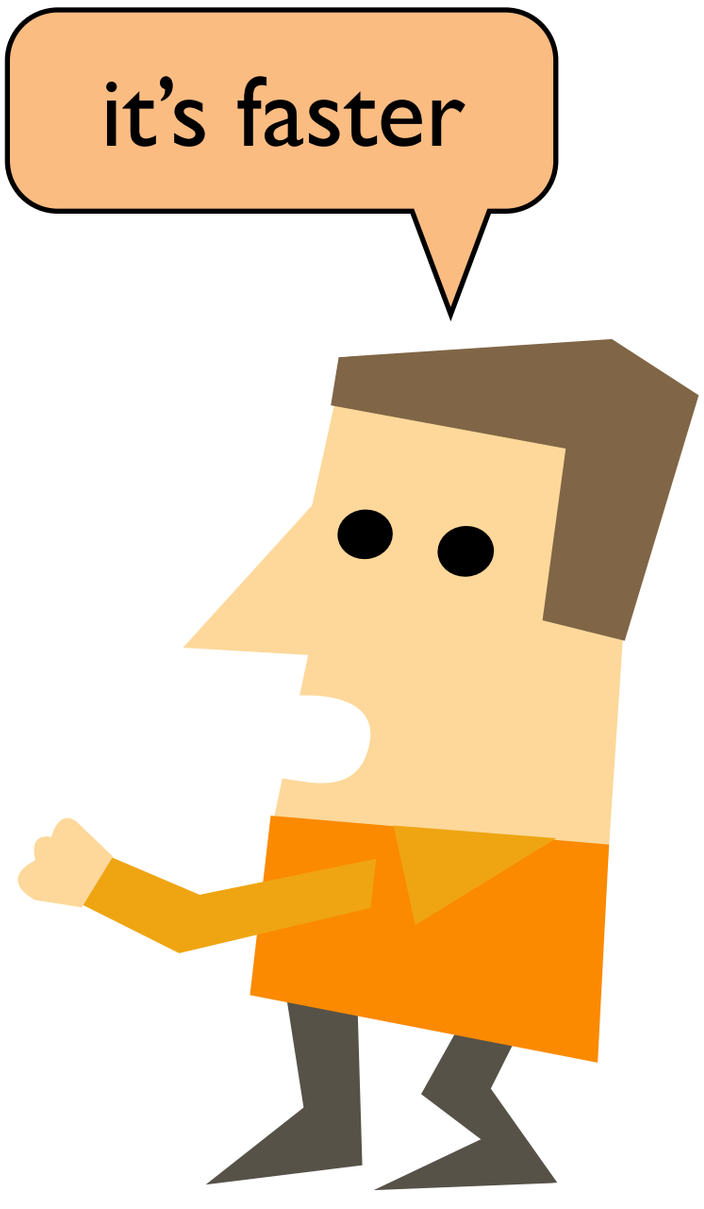
What if we only use this layout?

Upgrade `libc`

Changes layout

But it ran faster!

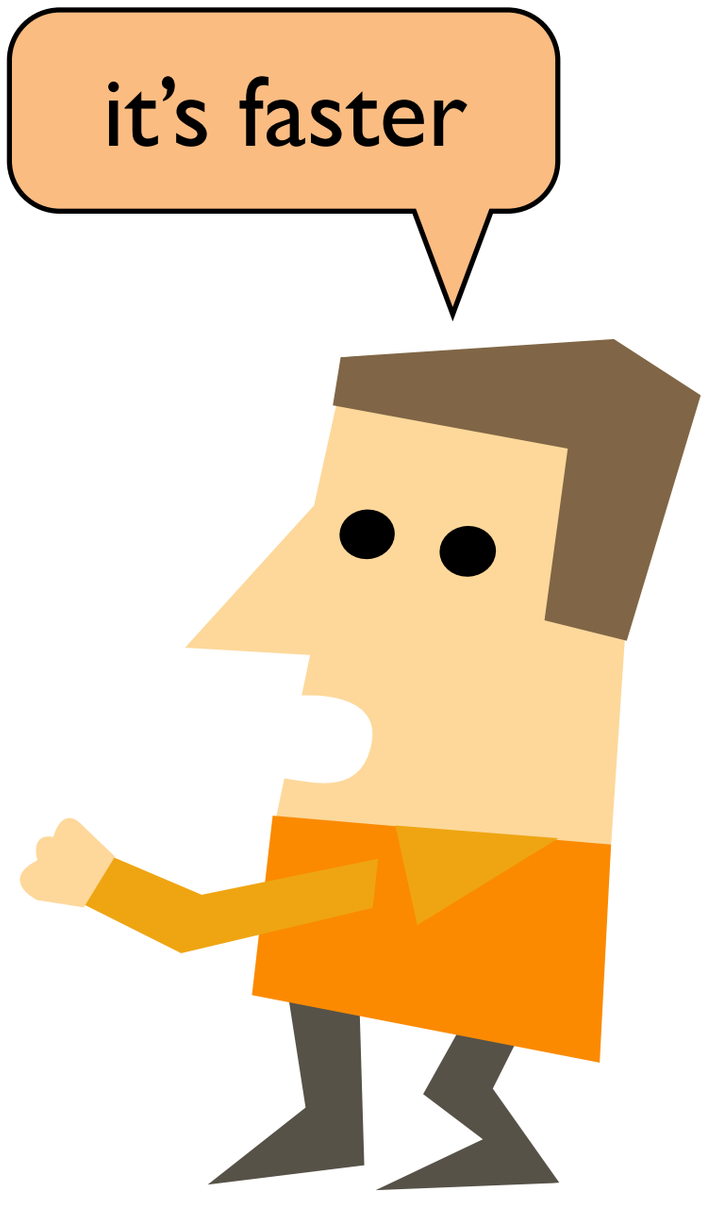
What if we only use this layout?



Change Username

But it ran faster!

What if we only use this layout?

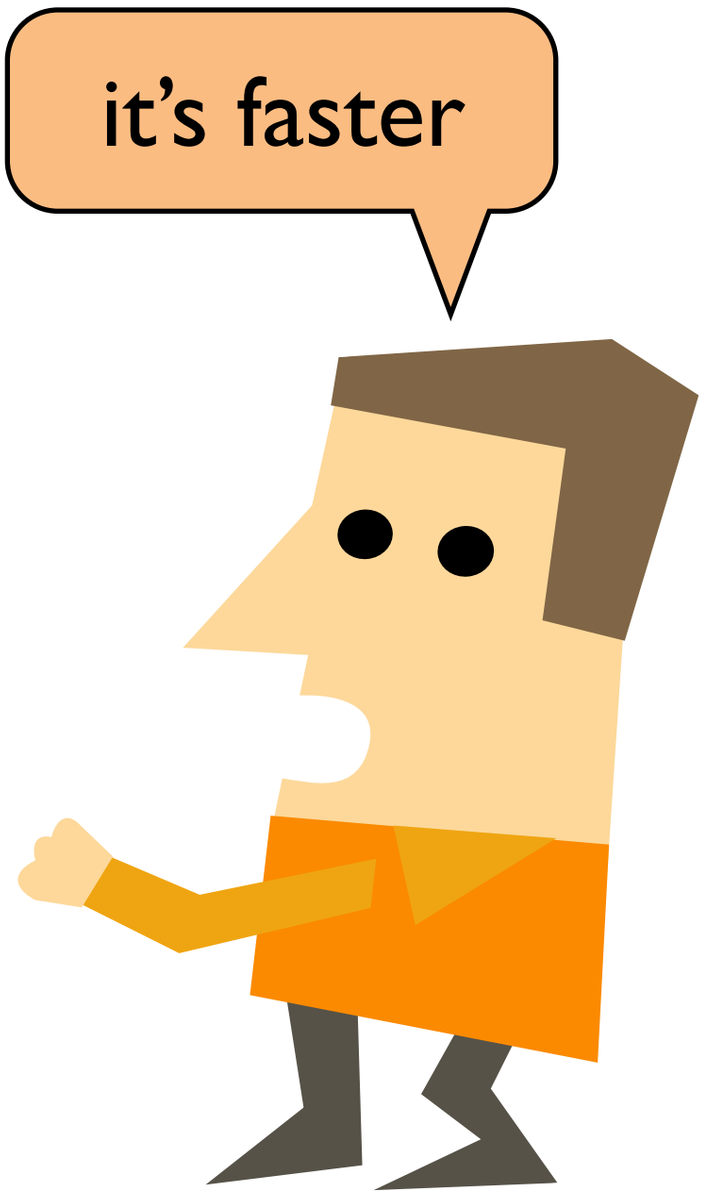


Change Username

Changes layout

But it ran faster!

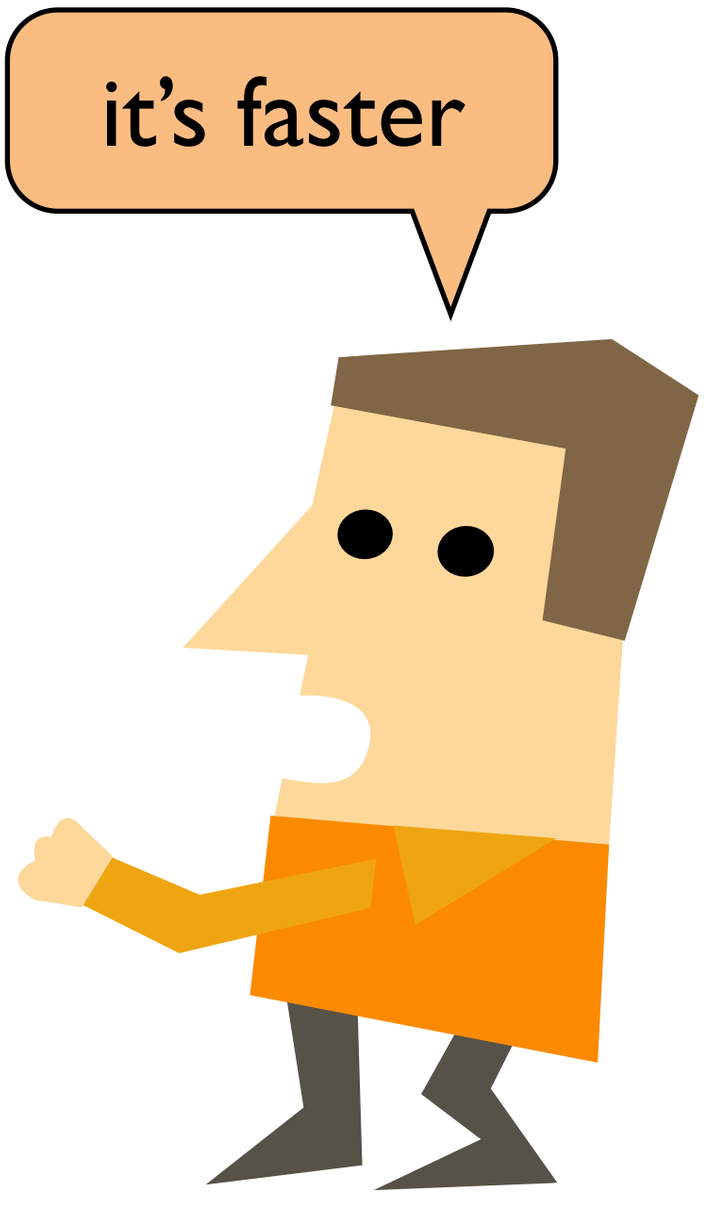
What if we only use this layout?



Run in a new directory

But it ran faster!

What if we only use this layout?



Run in a new directory

Changes layout

But it ran faster!

What if we only use this layout?

Layout is Brittle

But it ran faster!

What if we only use this layout?

Layout is Brittle

Layout biases measurement

Mytkowicz et al. (ASPLOS'09)

But it ran faster!

What if we only use this layout?

Layout is Brittle

Layout biases measurement

Mytkowicz et al. (ASPLOS'09)

Can we eliminate the effect of layout?

But it ran faster!

What if we only use this layout?

Layout biases measurement

Can we eliminate the effect of layout?

YES

Memory layout affects performance

makes performance evaluation difficult

STABILIZER eliminates the effect of layout

enables sound performance evaluation

Case Studies

evaluation of LLVM's optimizations with STABILIZER

Memory layout affects performance

makes performance evaluation difficult

STABILIZER eliminates the effect of layout

enables sound performance evaluation

Case Studies

evaluation of LLVM's optimizations with STABILIZER

STABILIZER

randomizes layout

Layout biases measurement

STABILIZER

randomizes layout

function addresses

Layout biases measurement

STABILIZER

randomizes layout

function addresses stack frame sizes

Layout biases measurement

STABILIZER

randomizes layout

function addresses stack frame sizes

heap allocations

Layout biases measurement

STABILIZER

repeatedly randomizes layout

function addresses stack frame sizes

heap allocations

Layout biases measurement

STABILIZER

during execution

repeatedly randomizes layout

function addresses

stack frame sizes

heap allocations

Layout biases measurement

STABILIZER

repeatedly randomizes layout

function addresses stack frame sizes

heap allocations

~~**Layout biases measurement**~~

STABILIZER

repeatedly randomizes layout

function addresses stack frame sizes

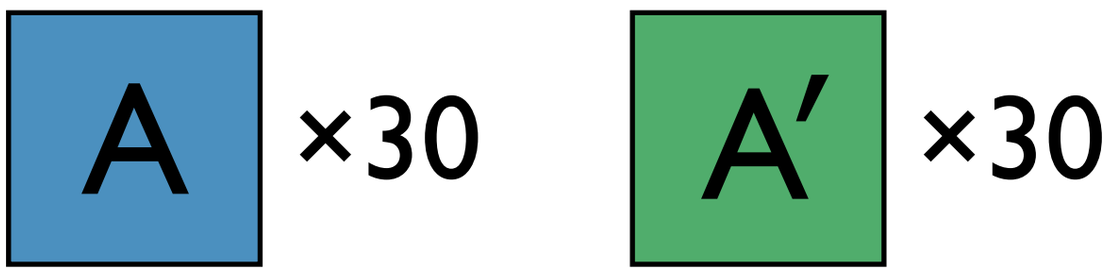
heap allocations

~~**Layout biases measurement**~~

a completely random layout

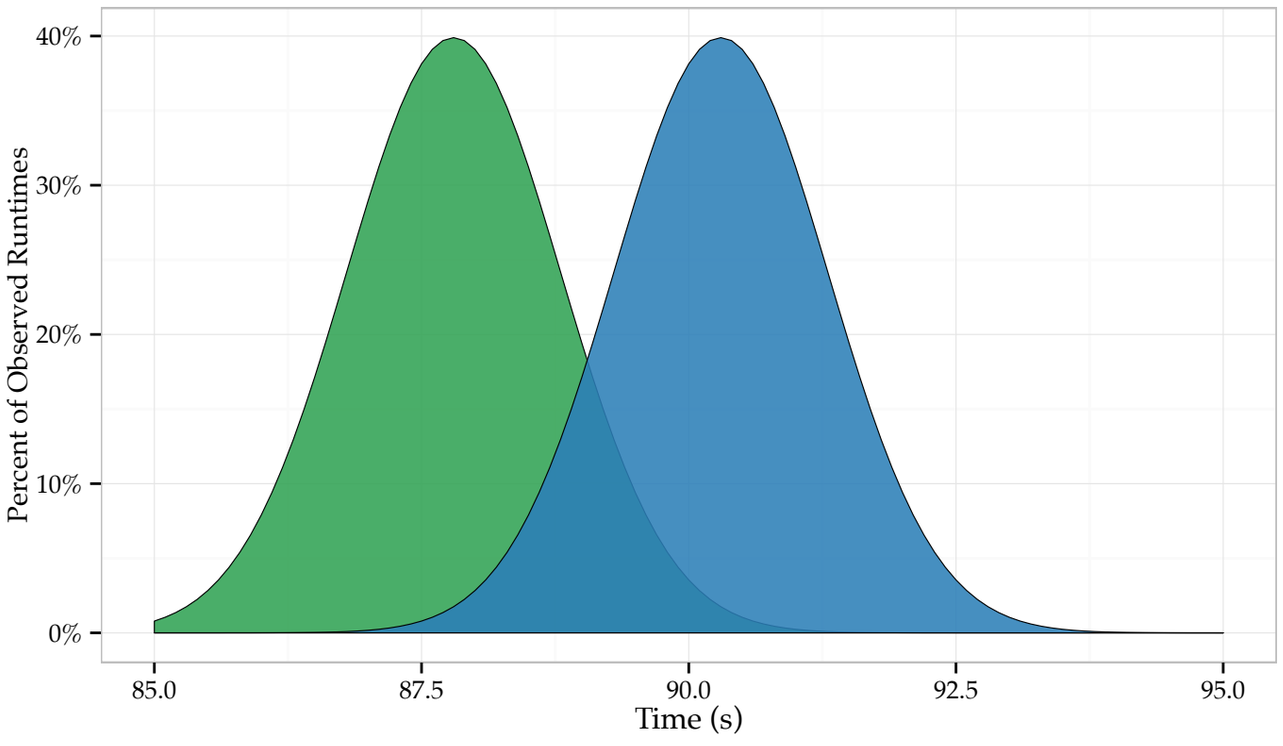
cannot bias results

Sound Performance Evaluation

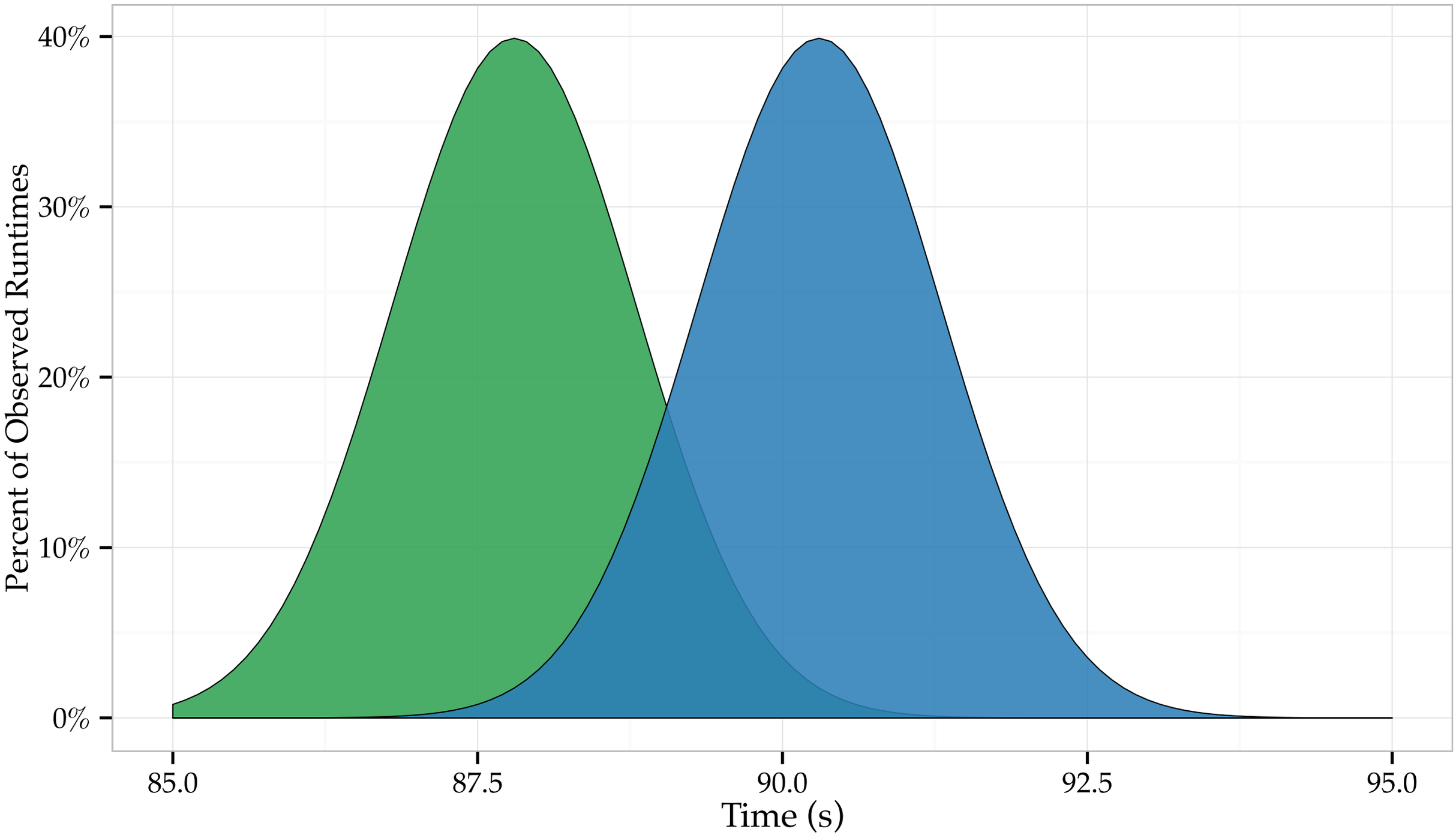


Sound Performance Evaluation

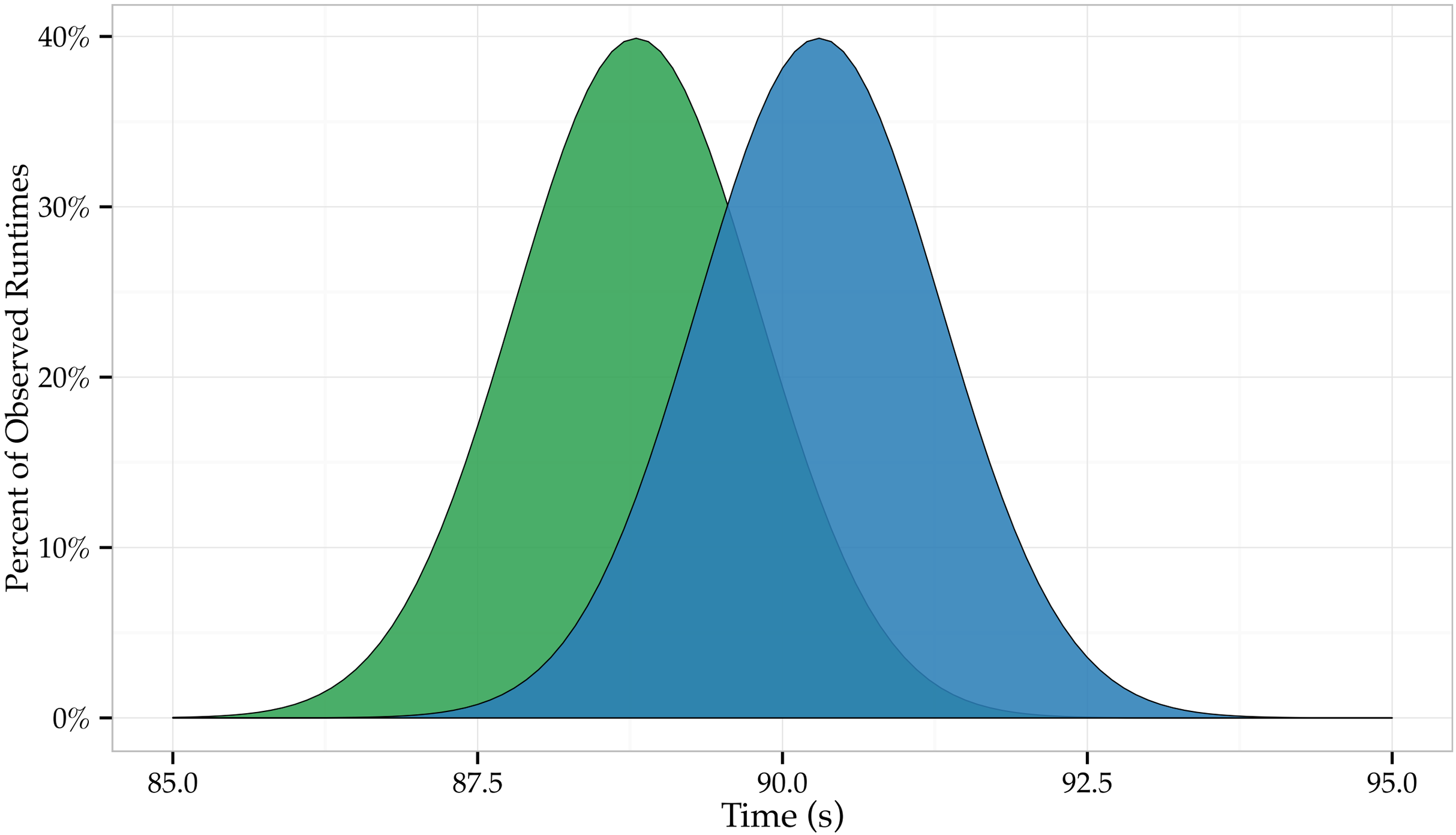
A ×30 A′ ×30



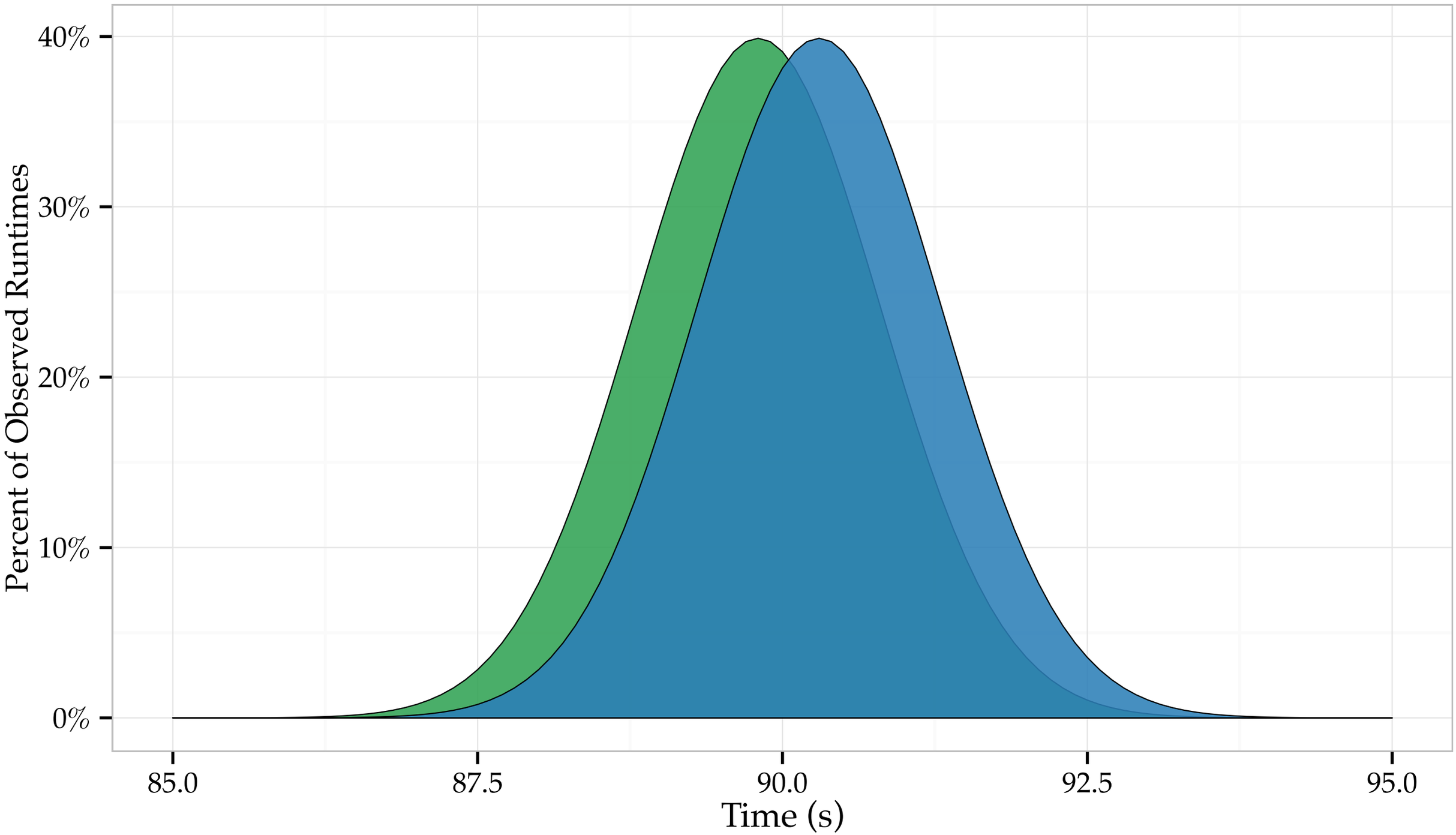
Is A′ faster than A?



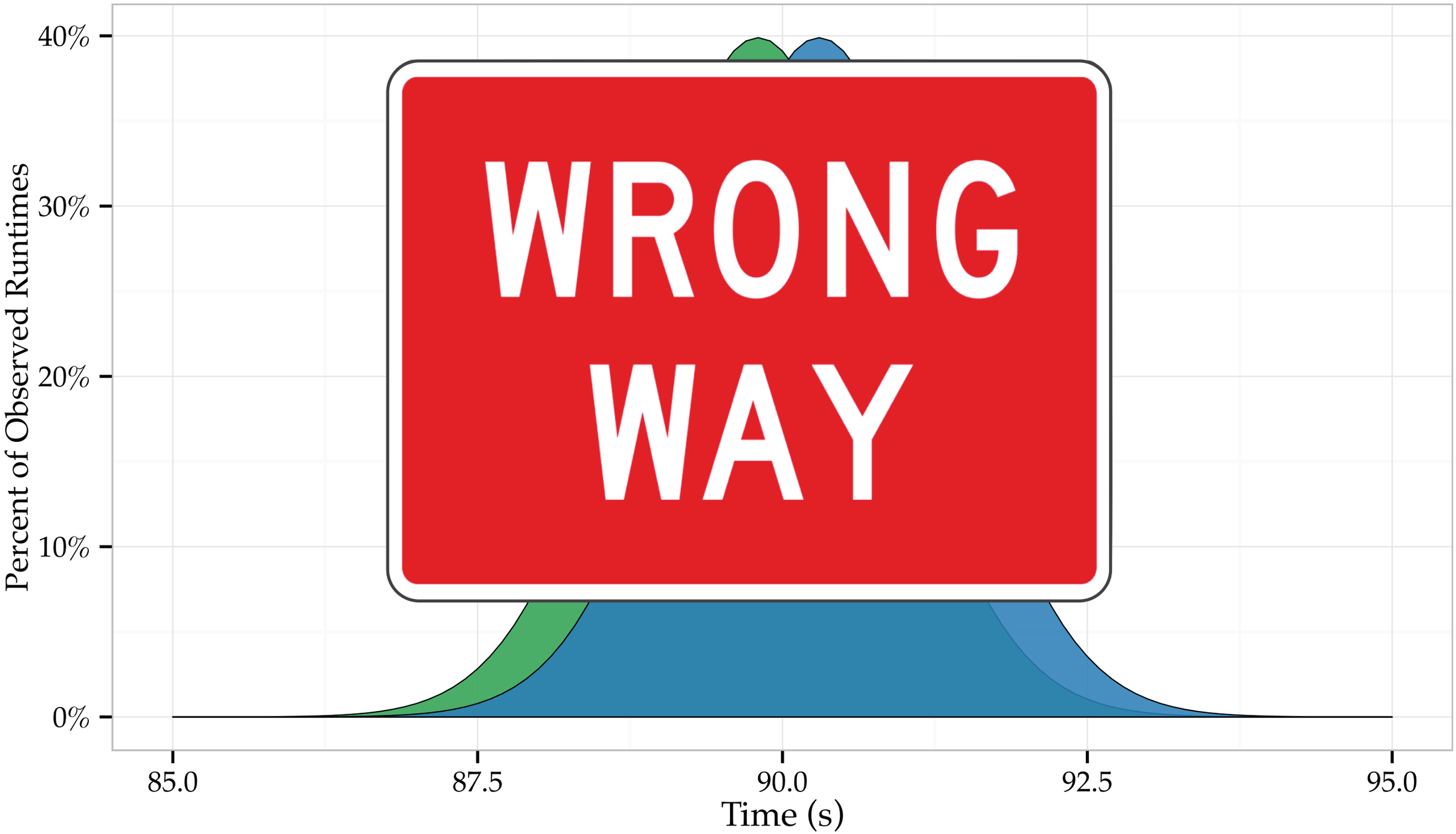
Is A′ faster than A?



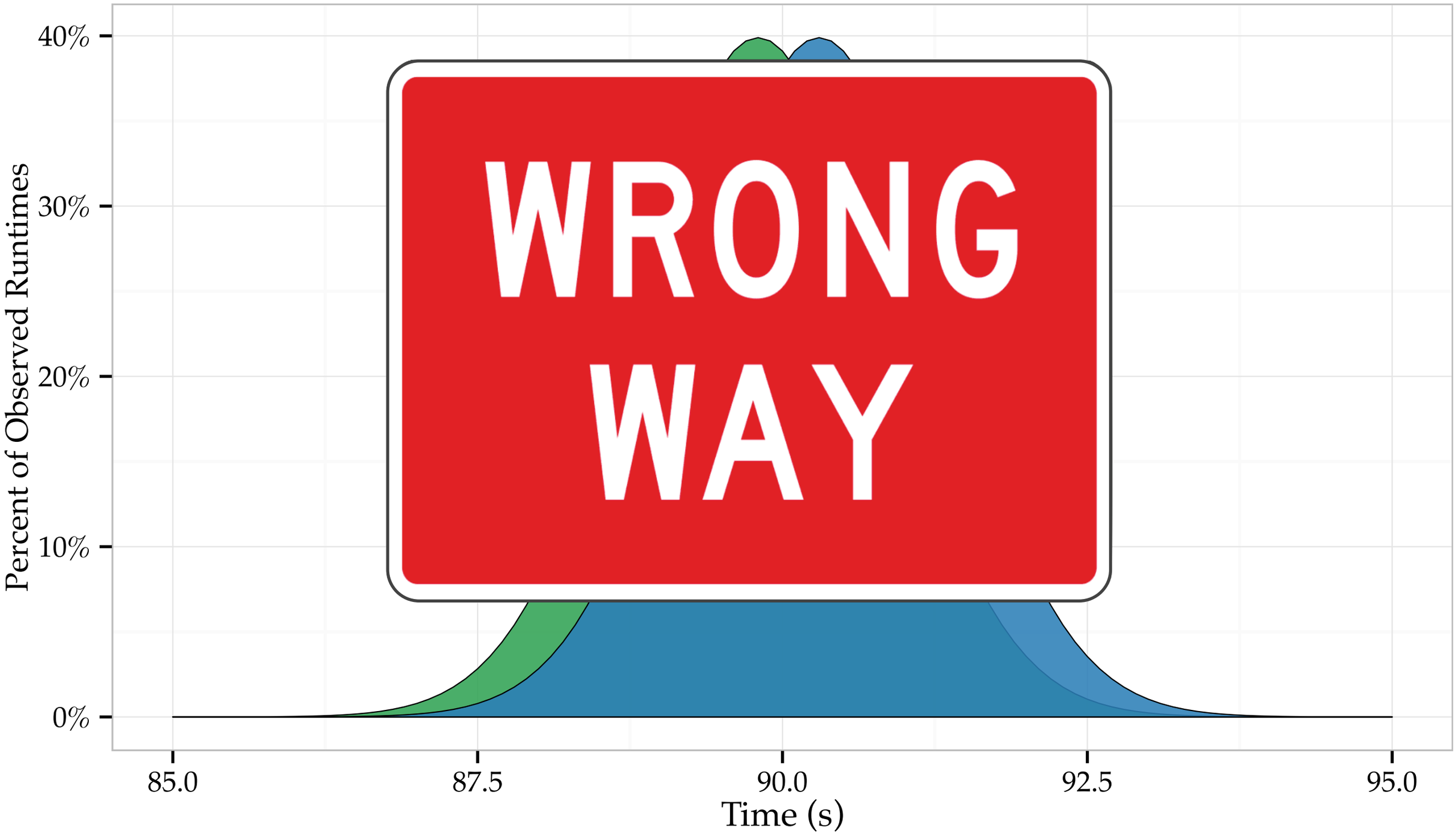
Is A′ faster than A?



Is A′ faster than A?

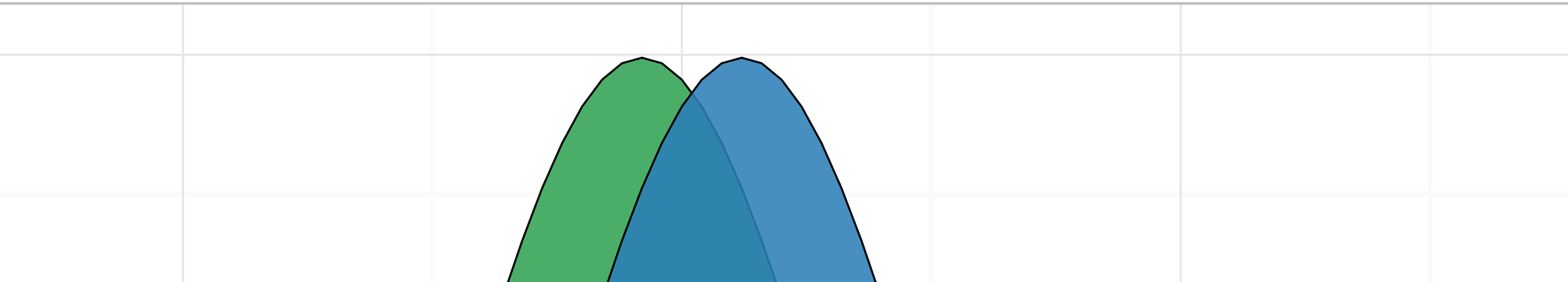


Is A′ faster than A?



The Statistical Approach

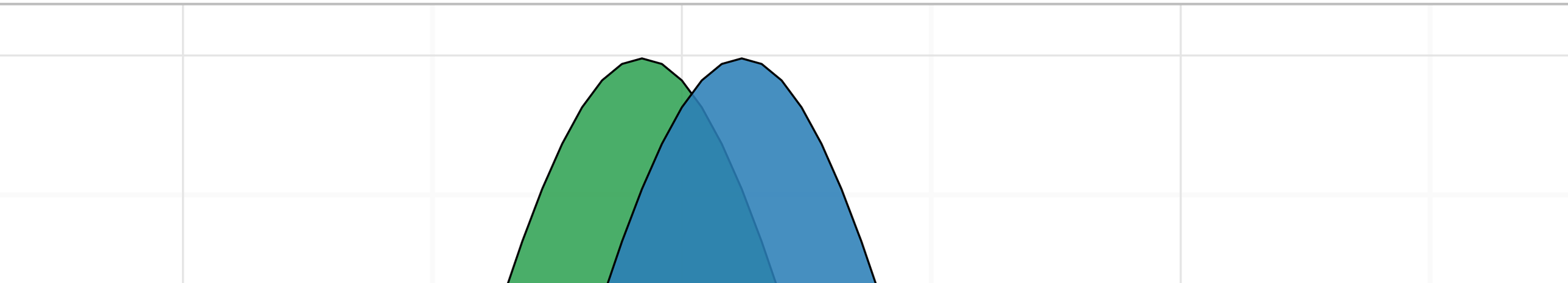
If A′ = A



The Statistical Approach

hypothesis testing

If A′ = A

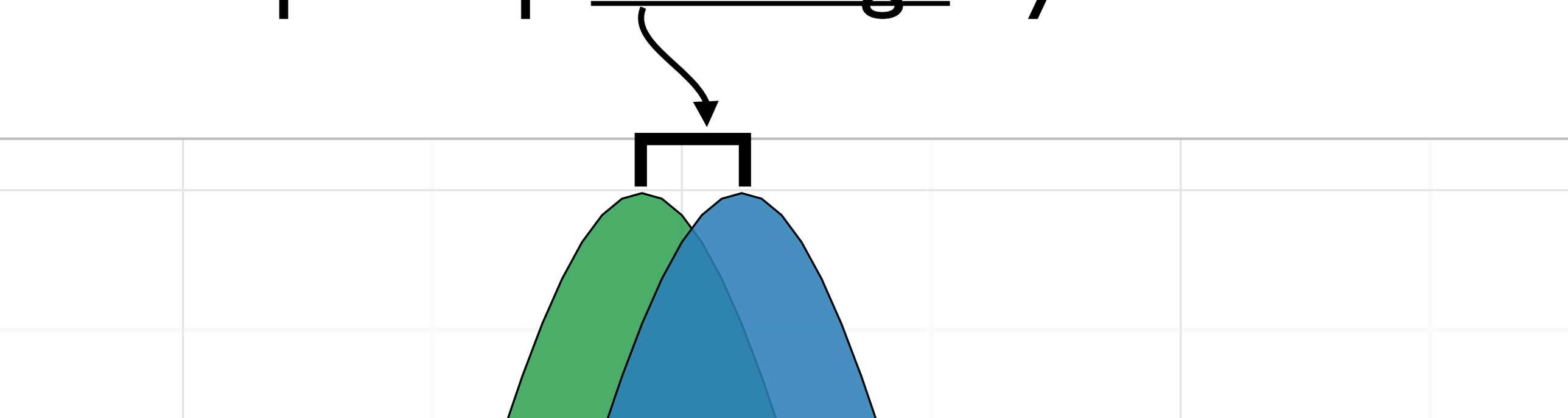


The Statistical Approach

hypothesis testing

If A′ = A

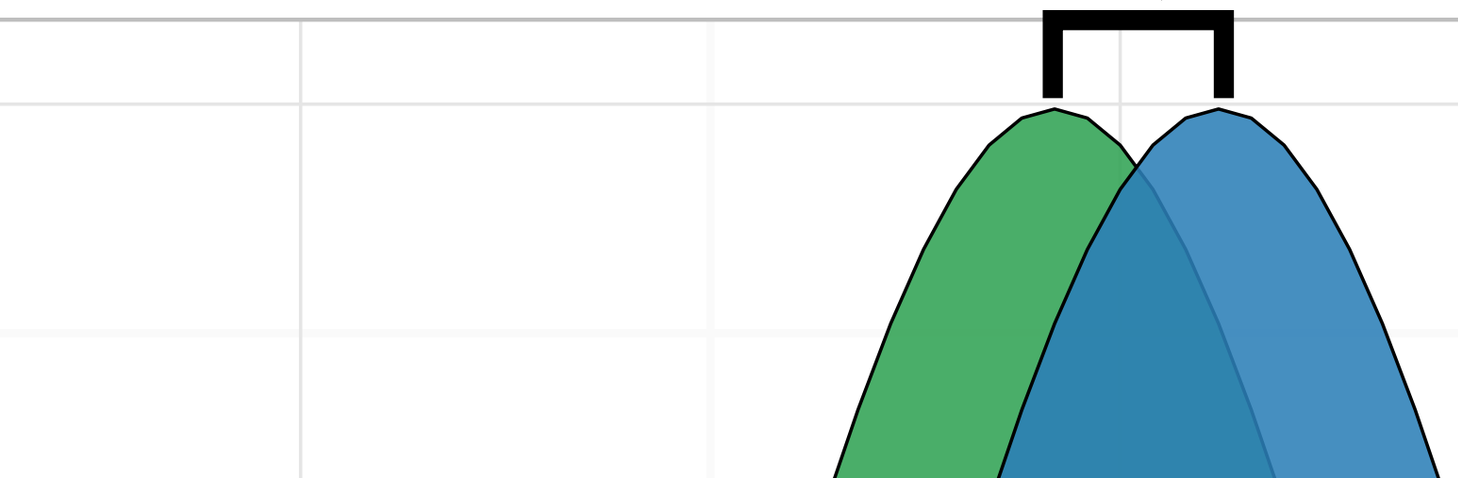
what is the probability of measuring a speedup this large by chance?



If A′ = A

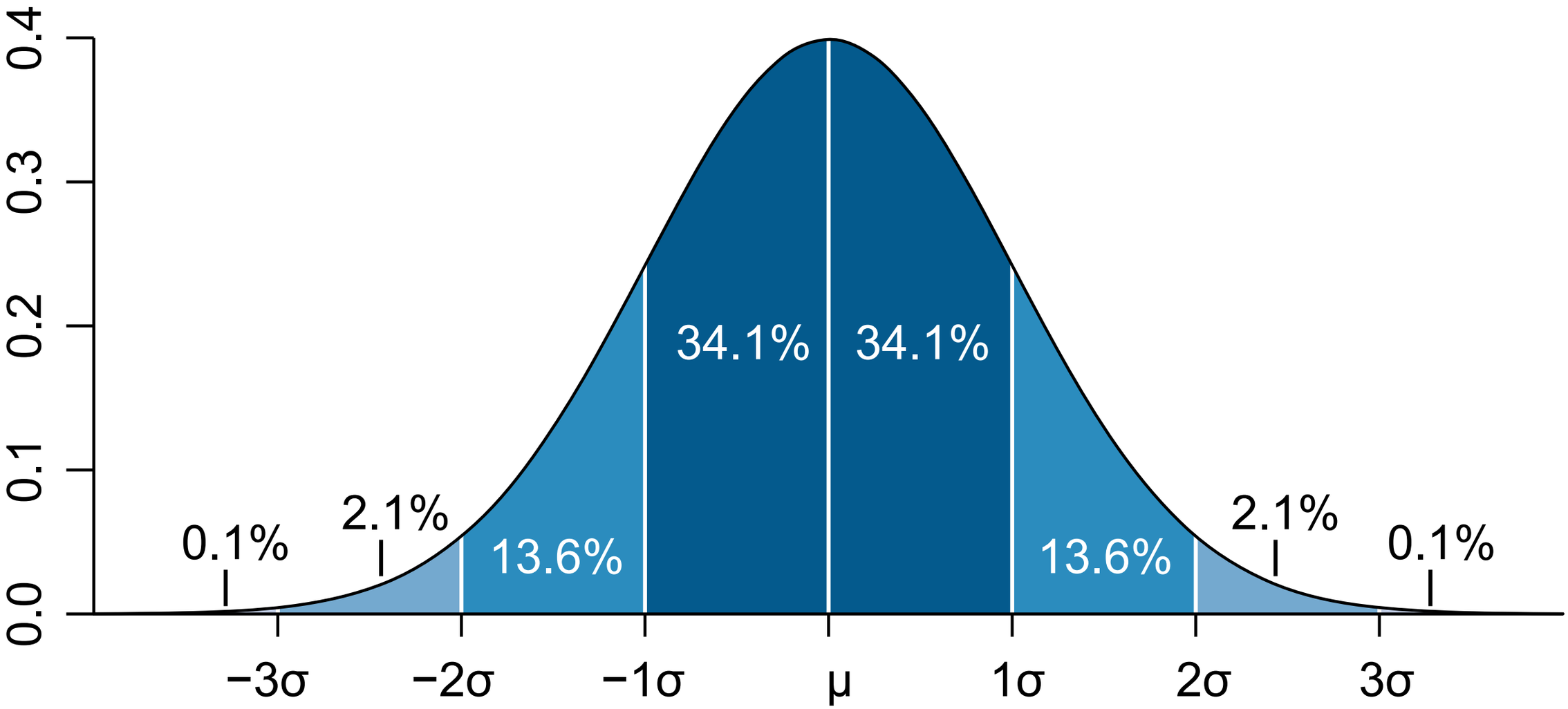
easy to compute for the normal distribution

what is the probability of measuring a speedup this large by chance?



If A' = A

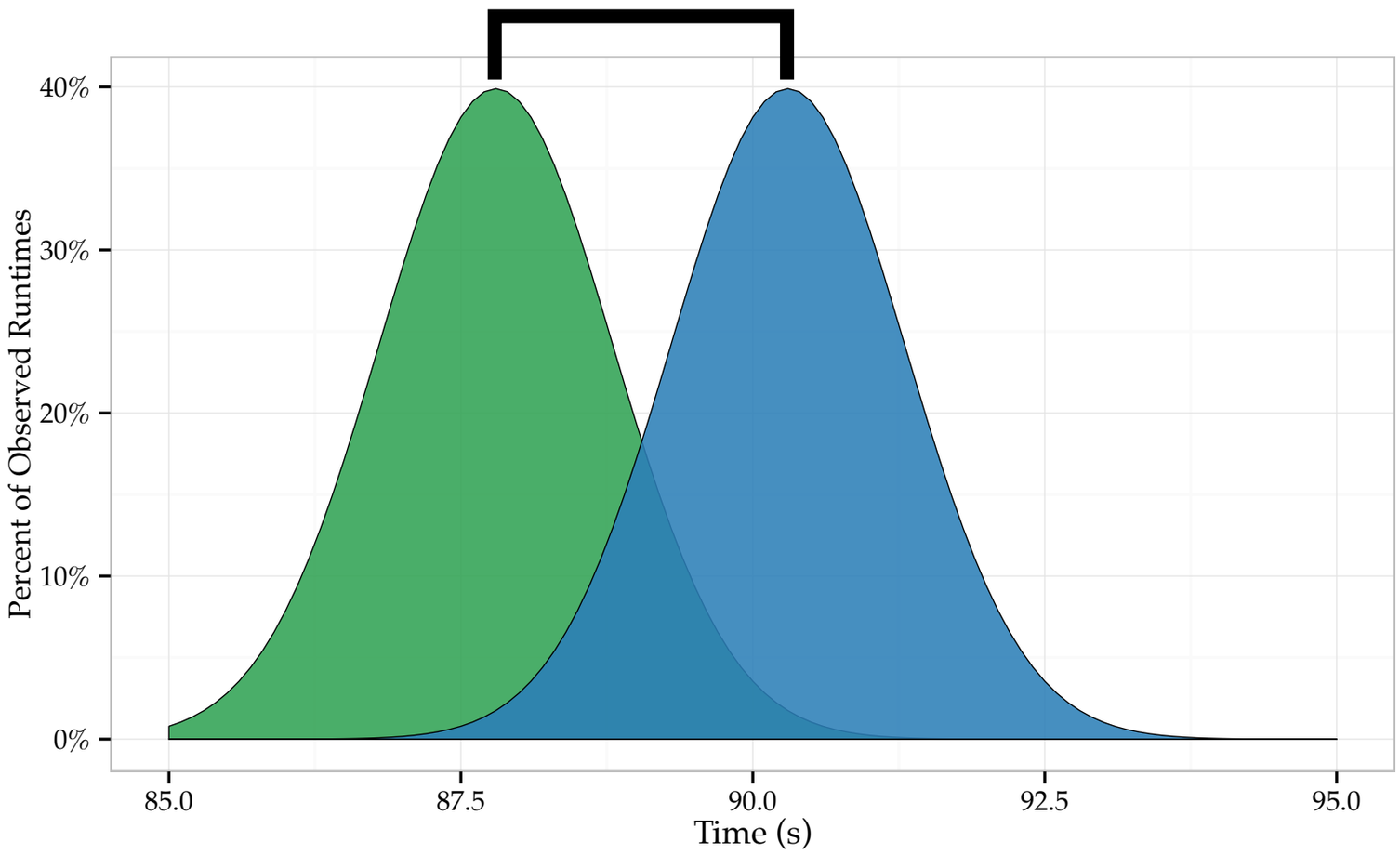
easy to compute for the normal distribution



STABILIZER

repeatedly randomizes layout

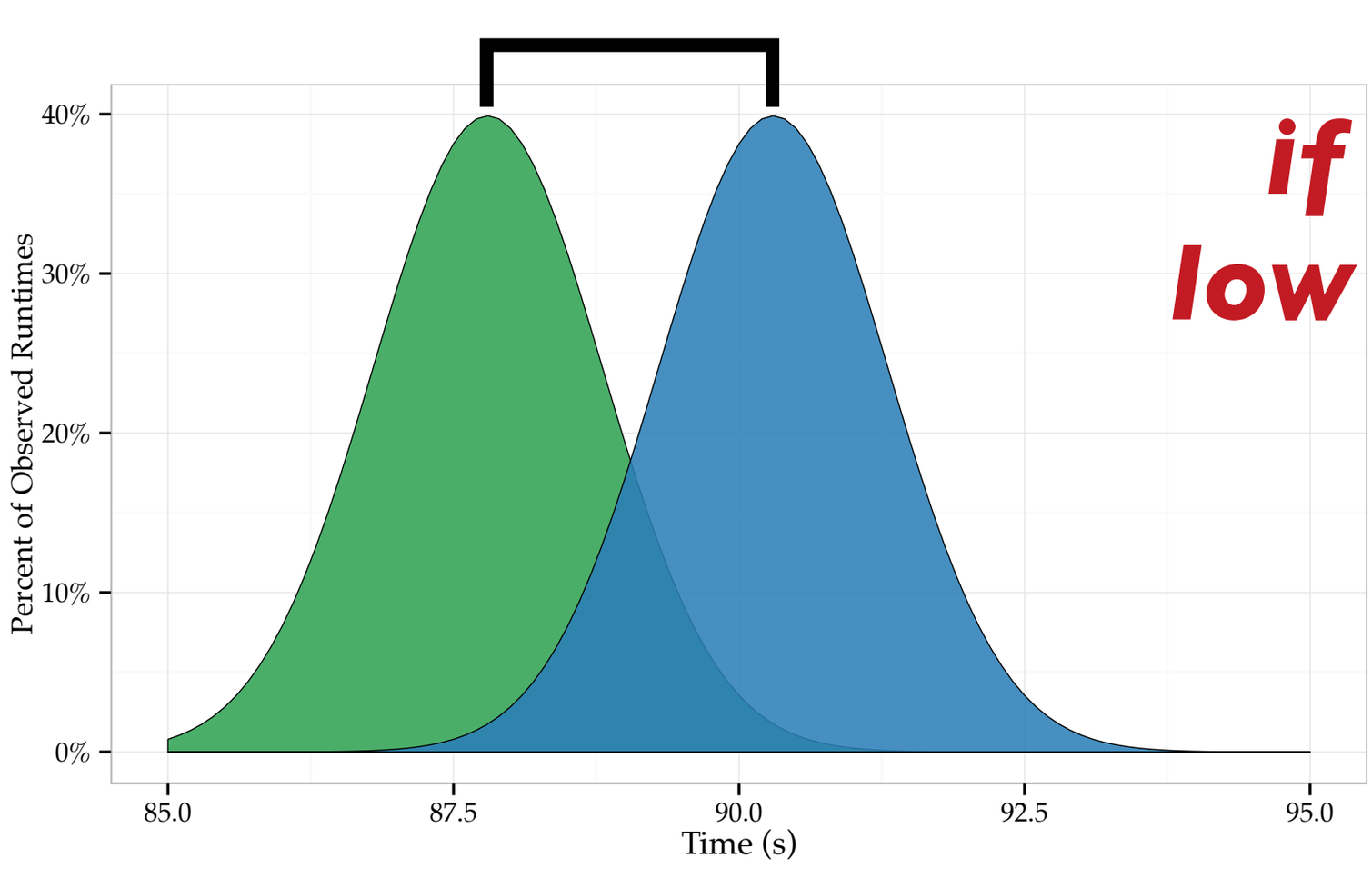
what is the probability of measuring a speedup this large by chance?



STABILIZER

repeatedly randomizes layout

what is the probability of measuring a speedup this large by chance?



if there is a low probability

STABILIZER

repeatedly randomizes layout

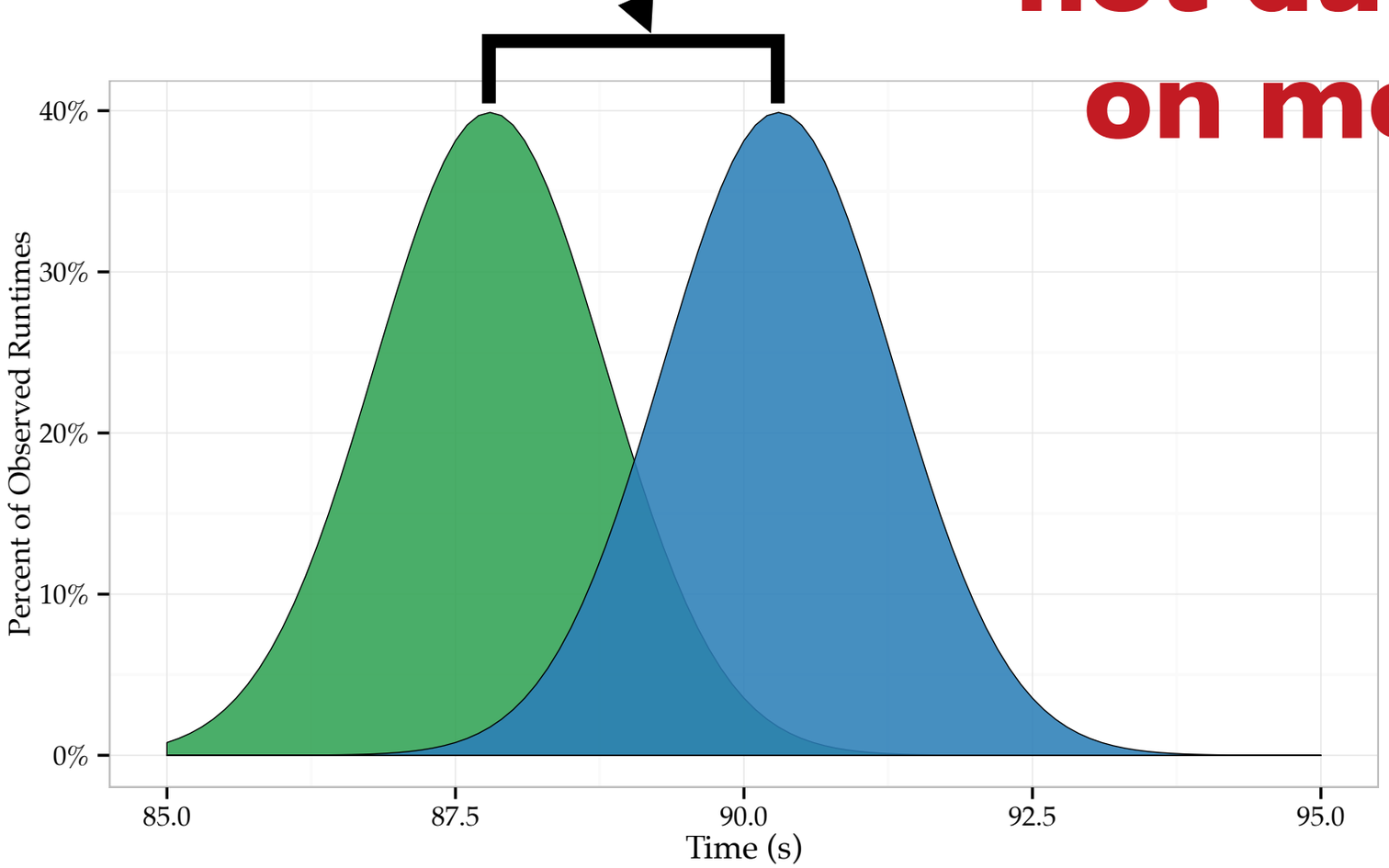
this speedup is real

STABILIZER

repeatedly randomizes layout

this speedup is real

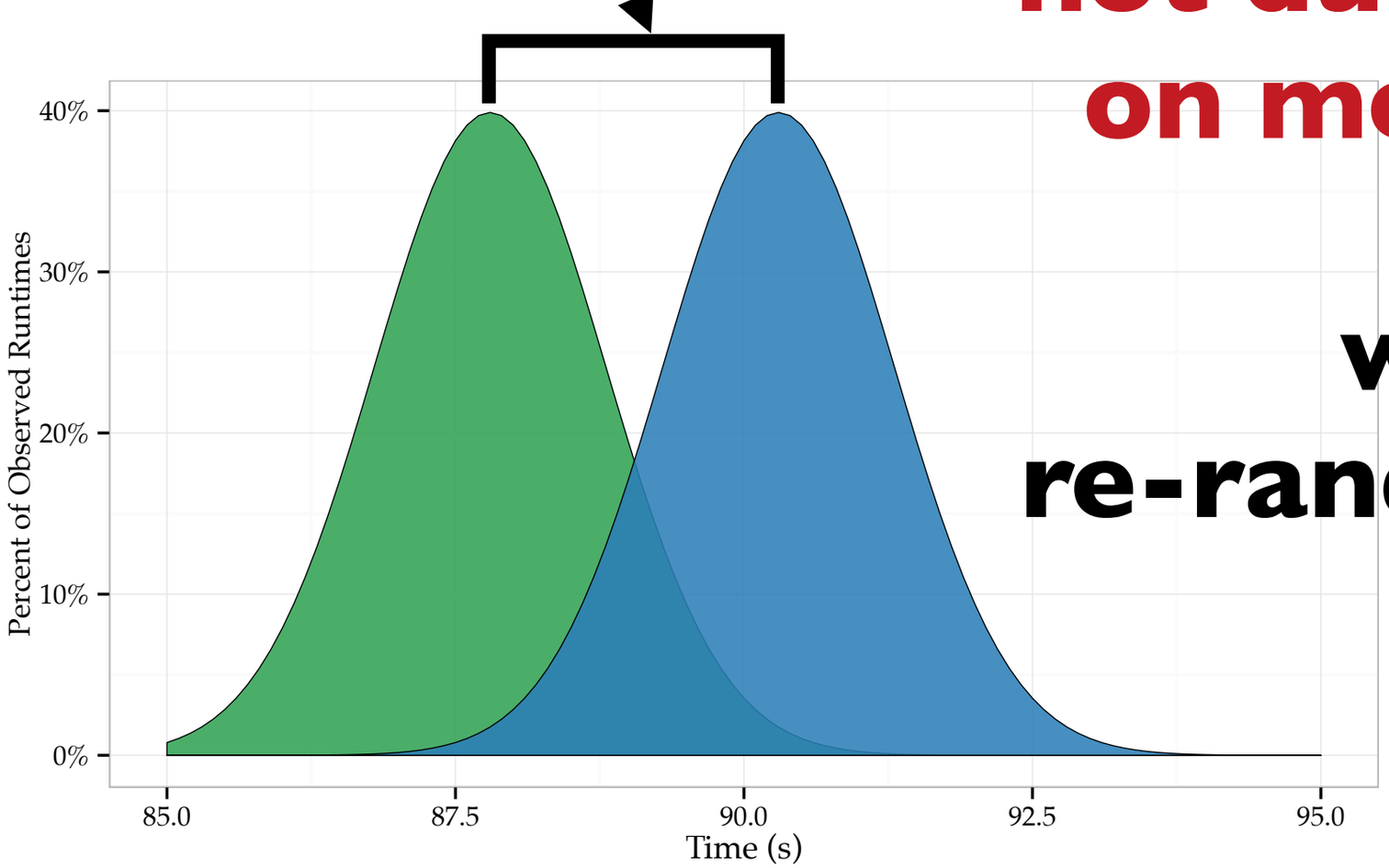
not due to the effect on memory layout



STABILIZER

repeatedly randomizes layout

this speedup is real

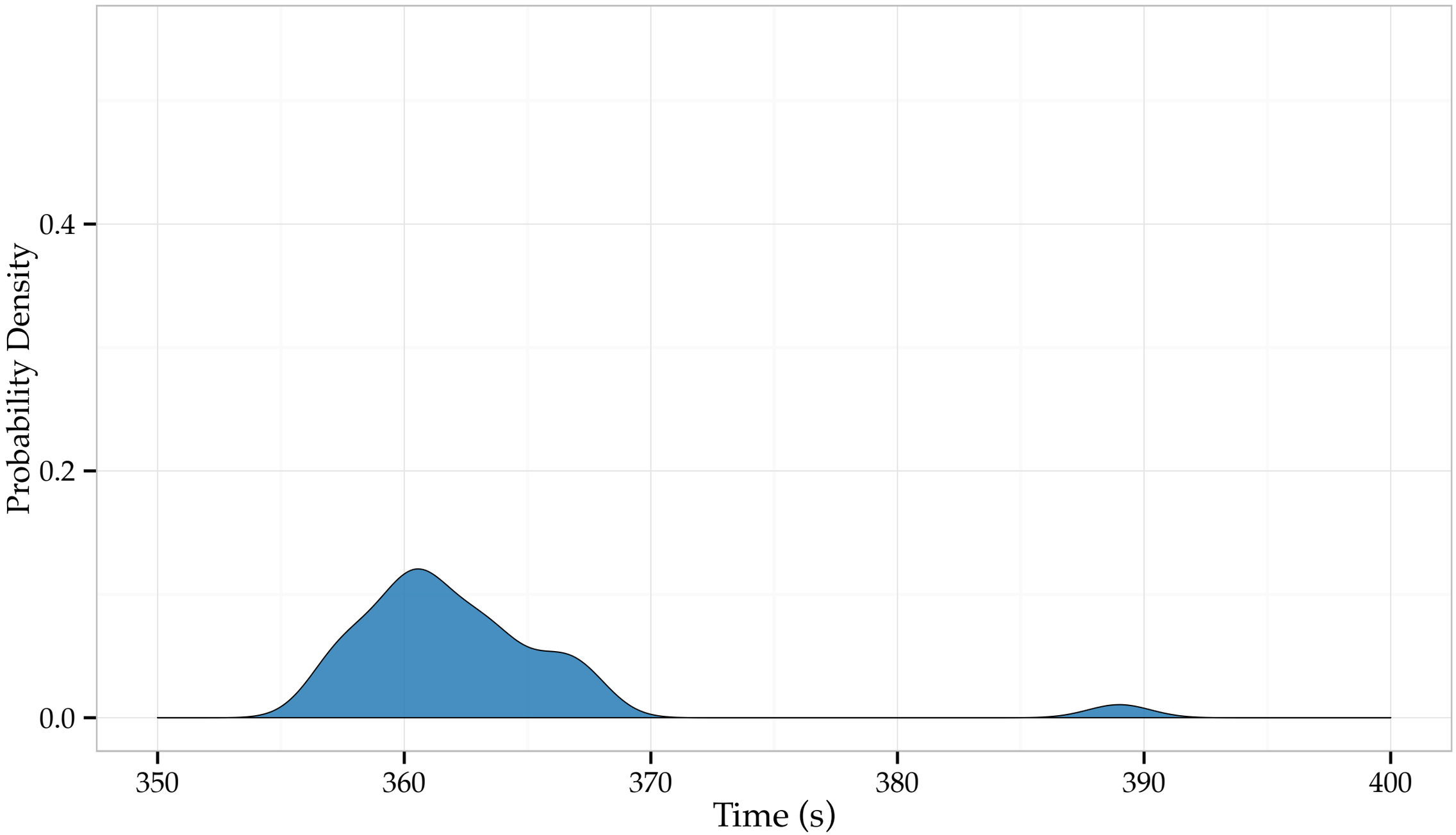


not due to the effect on memory layout

what does re-randomization do?

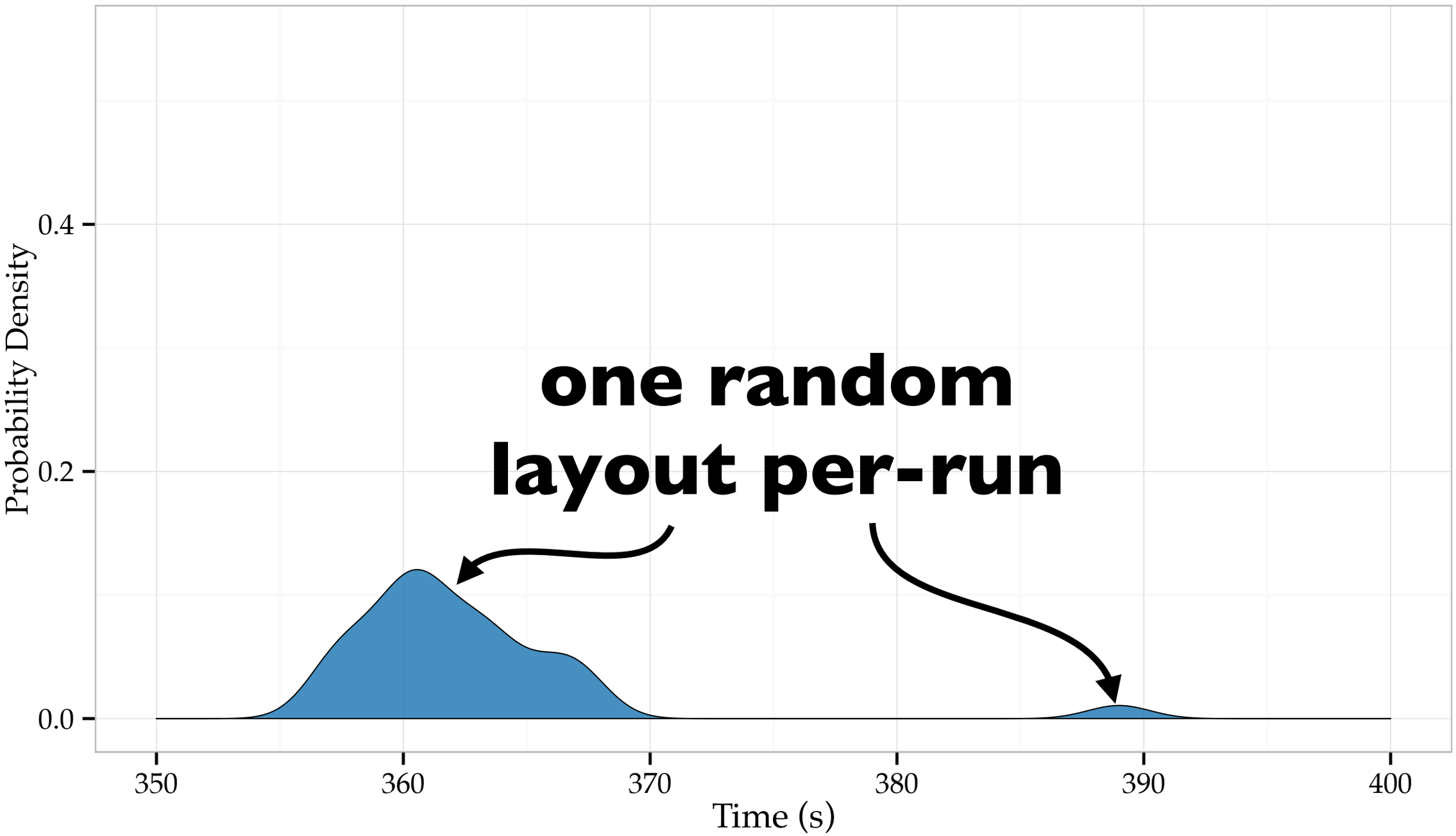
STABILIZER

repeatedly randomizes layout



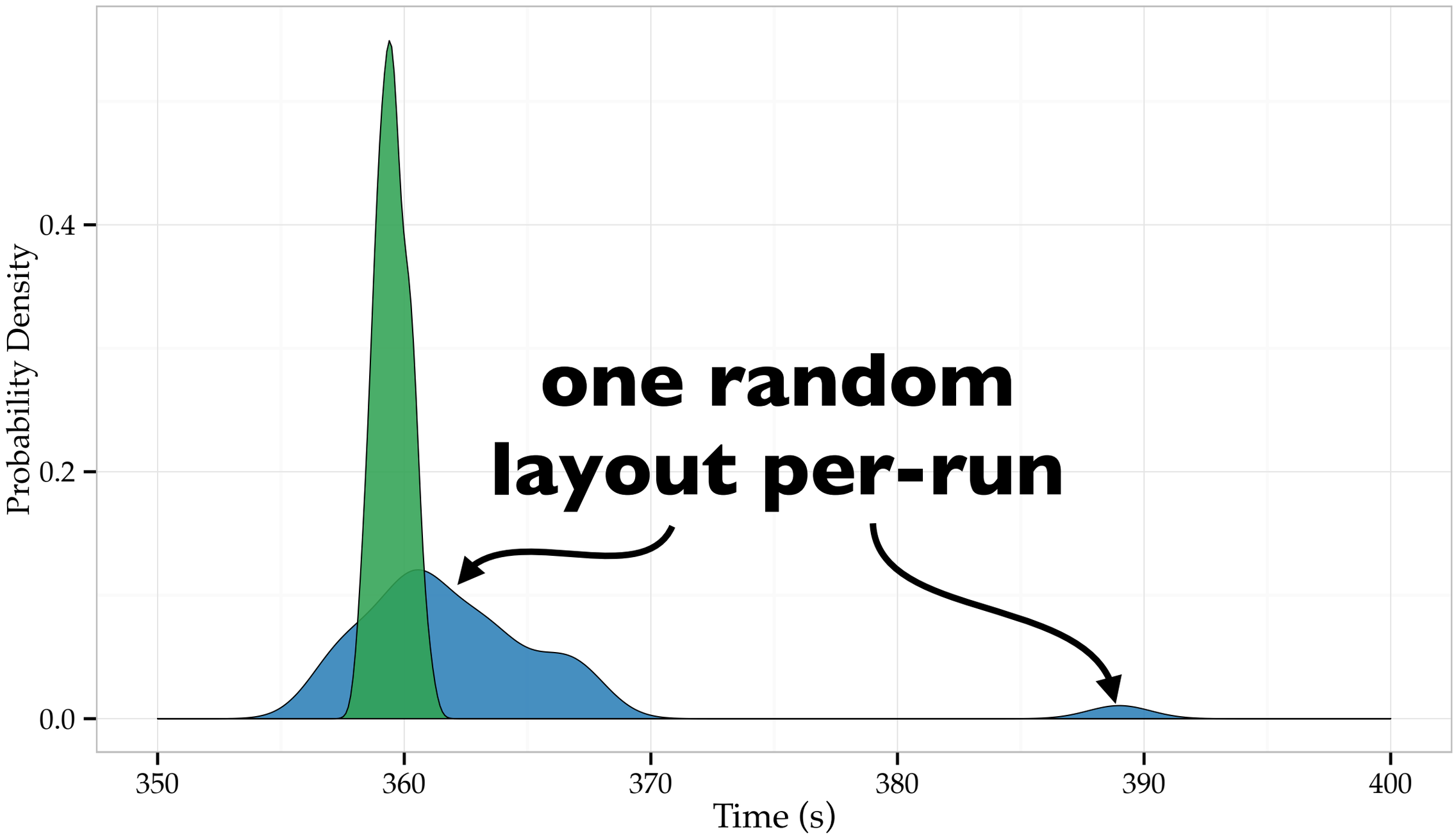
STABILIZER

repeatedly randomizes layout



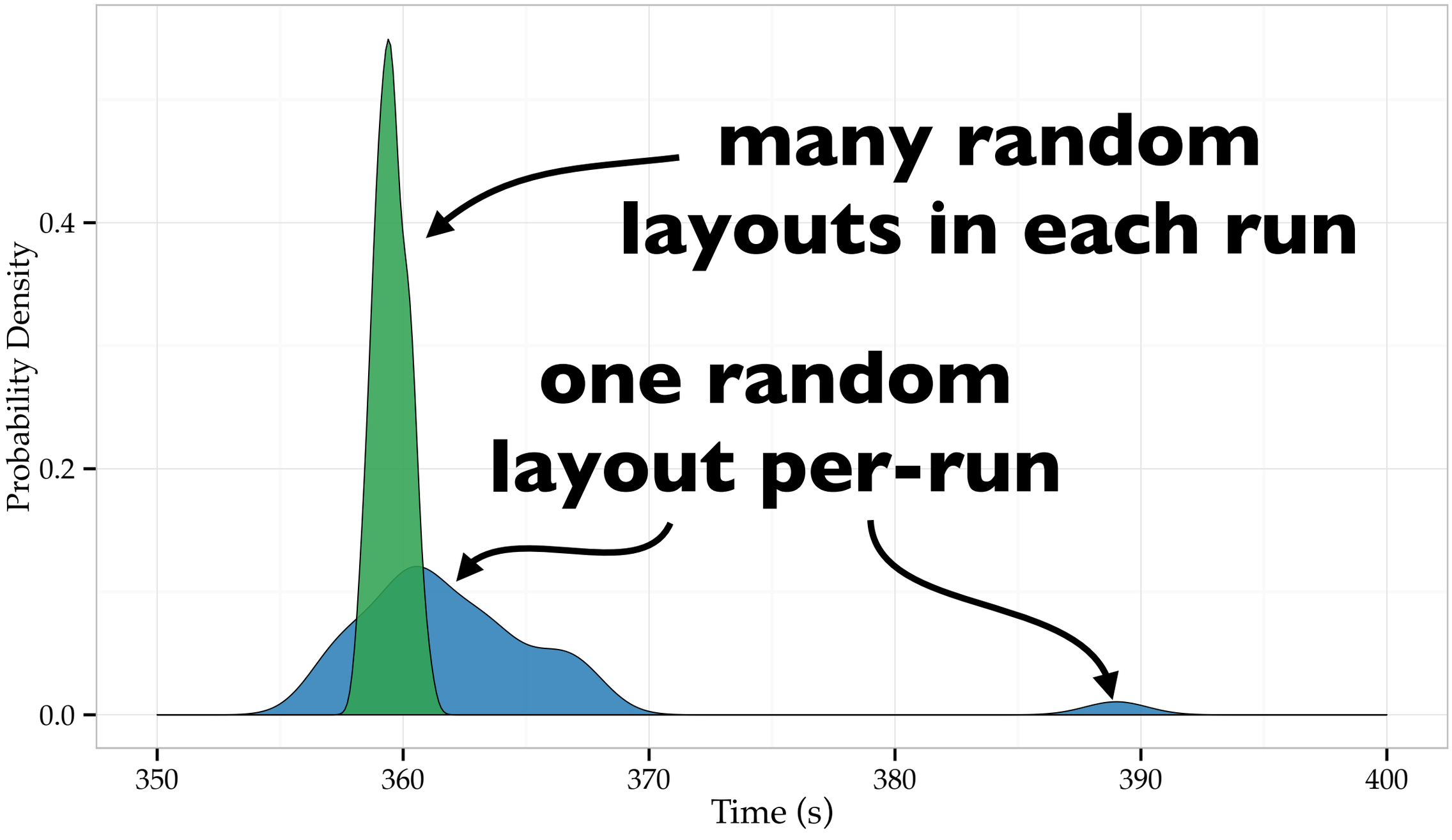
STABILIZER

repeatedly randomizes layout



STABILIZER

repeatedly randomizes layout



STABILIZER

repeatedly randomizes layout

STABILIZER

repeatedly randomizes layout

STABILIZER generates a new random layout every ½ second

STABILIZER

repeatedly randomizes layout

STABILIZER generates a new random layout every ½ second

Total execution time is the sum of all periods

STABILIZER

repeatedly randomizes layout

STABILIZER generates a new random layout every ½ second

Total execution time is the sum of all periods

The sum

STABILIZER

repeatedly randomizes layout

STABILIZER generates a new random layout every ½ second

Total execution time is the sum of all periods

The sum of a sufficient number

STABILIZER

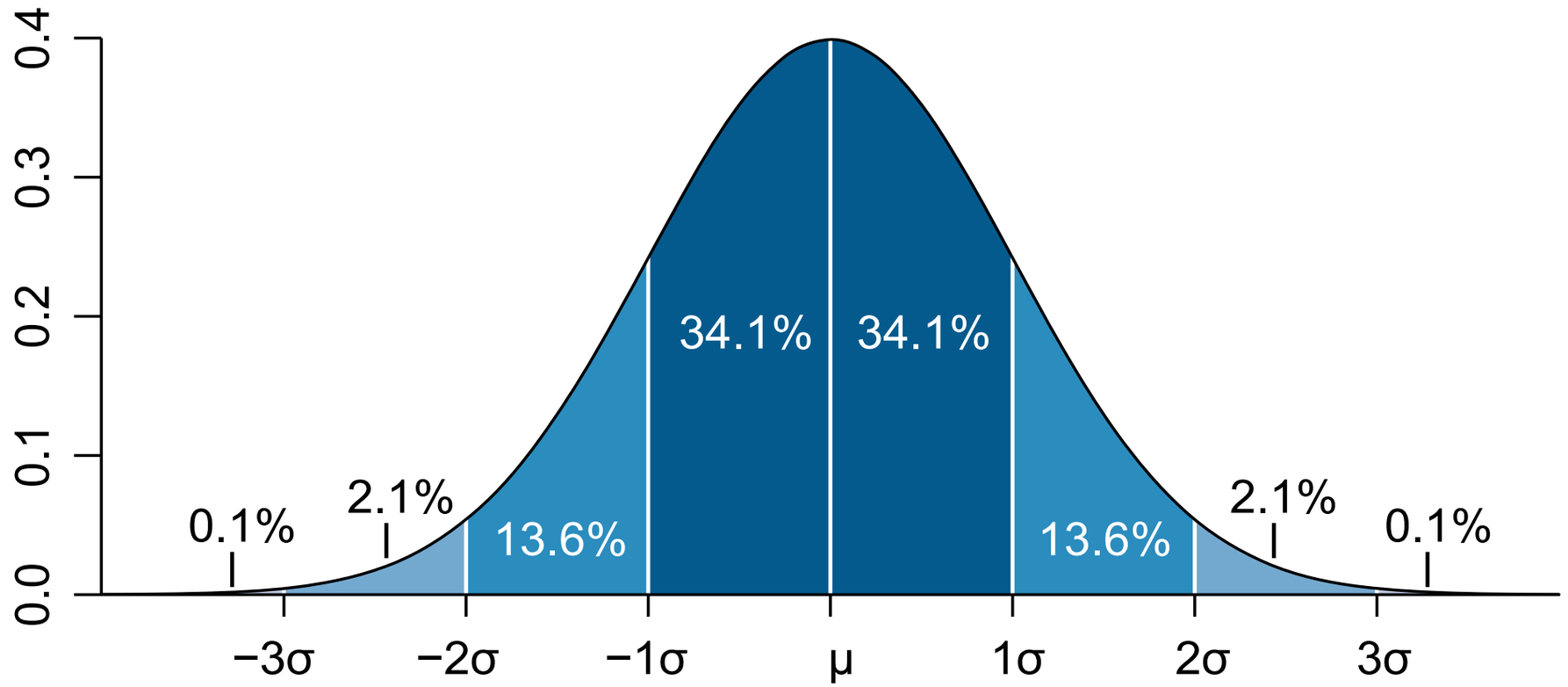
repeatedly randomizes layout

STABILIZER generates a new random layout every ½ second

Total execution time is the sum of all periods

The sum of a sufficient number of independent, identically distributed random variables

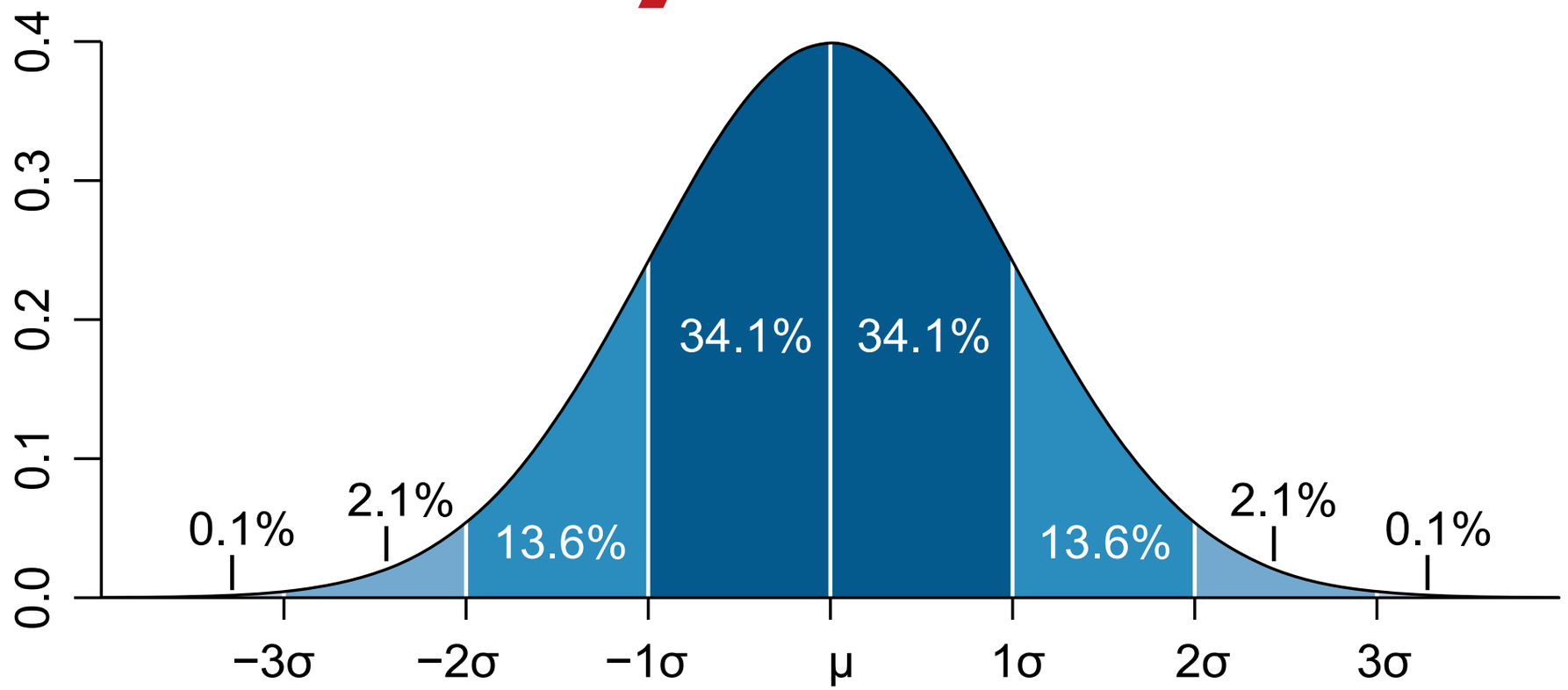
Central Limit Theorem



The sum of a sufficient number of independent, identically distributed random variables is approximately normally distributed.

Central Limit Theorem

execution times are normally distributed



The sum of a sufficient number of independent, identically distributed random variables is approximately normally distributed.

Memory layout affects performance

makes performance evaluation difficult

STABILIZER eliminates the effect of layout

enables sound performance evaluation

Case Studies

evaluation of LLVM's optimizations with STABILIZER

Memory layout affects performance

makes performance evaluation difficult

STABILIZER eliminates the effect of layout

enables sound performance evaluation

Case Studies

evaluation of LLVM's optimizations with STABILIZER

Case Studies

evaluation of LLVM's optimizations with STABILIZER

Case Studies

evaluation of LLVM's optimizations with STABILIZER

on each benchmark

Case Studies

evaluation of LLVM's optimizations with STABILIZER

on each benchmark across the whole benchmark suite

Case Studies

evaluation of LLVM's optimizations with STABILIZER

on each benchmark

across the whole benchmark suite

first, build benchmarks with STABILIZER

Build programs with STABILIZER

Build programs with STABILIZER

```
> szc main.c
```

Build programs with STABILIZER

```
> szc main.c
```

Build programs with STABILIZER

```
> szc -Rcode main.c
```

Build programs with STABILIZER

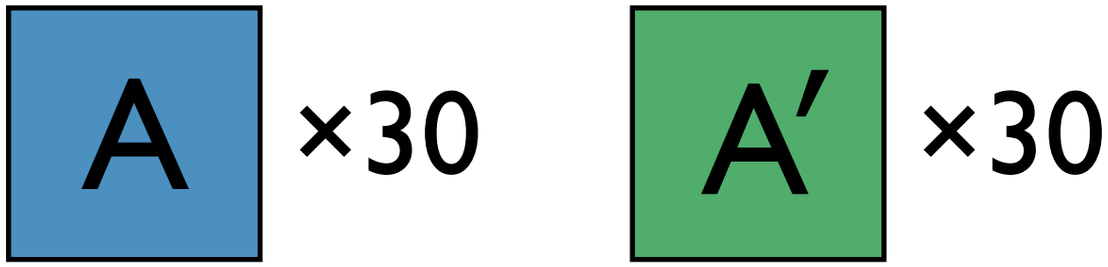
```
> szc -Rcode -Rheap -Rstack main.c
```

Build programs with STABILIZER

```
> szc -Rcode -Rheap -Rstack main.c
```

now run the benchmarks

Run benchmarks as usual



Run benchmarks as usual

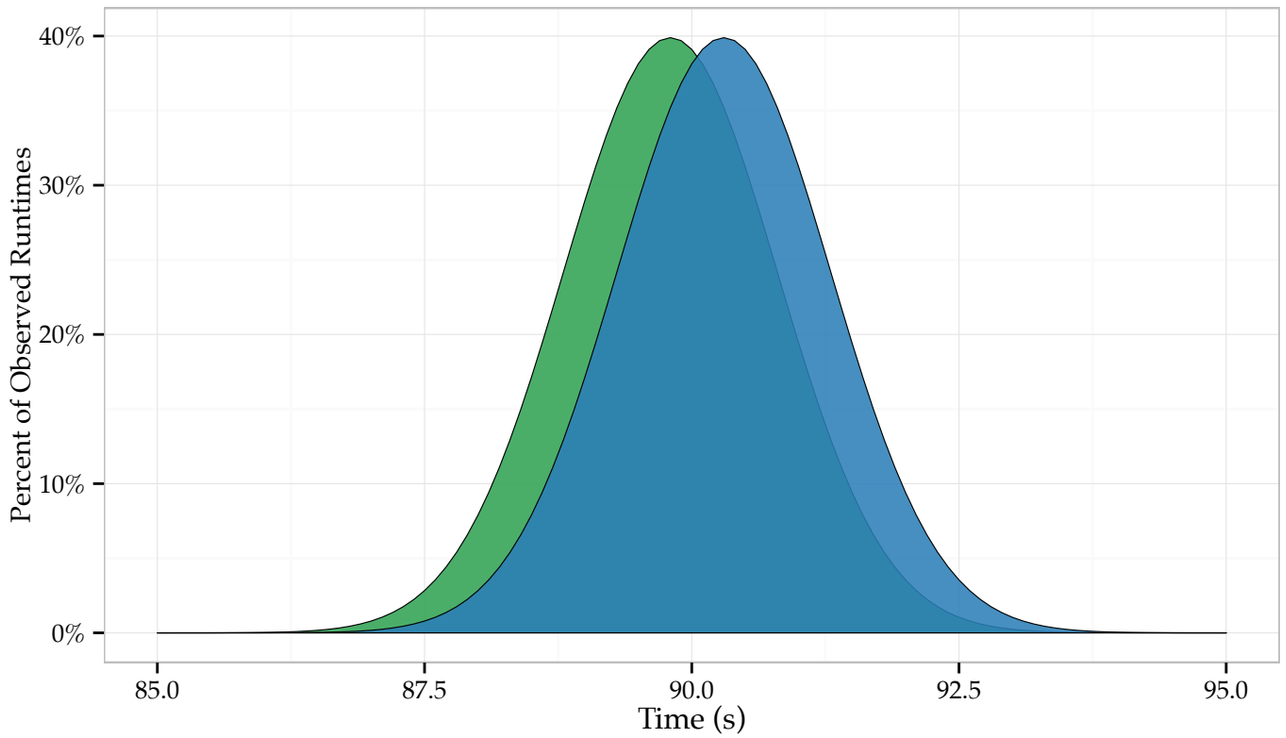
A ×30 A′ ×30

drop the results into R

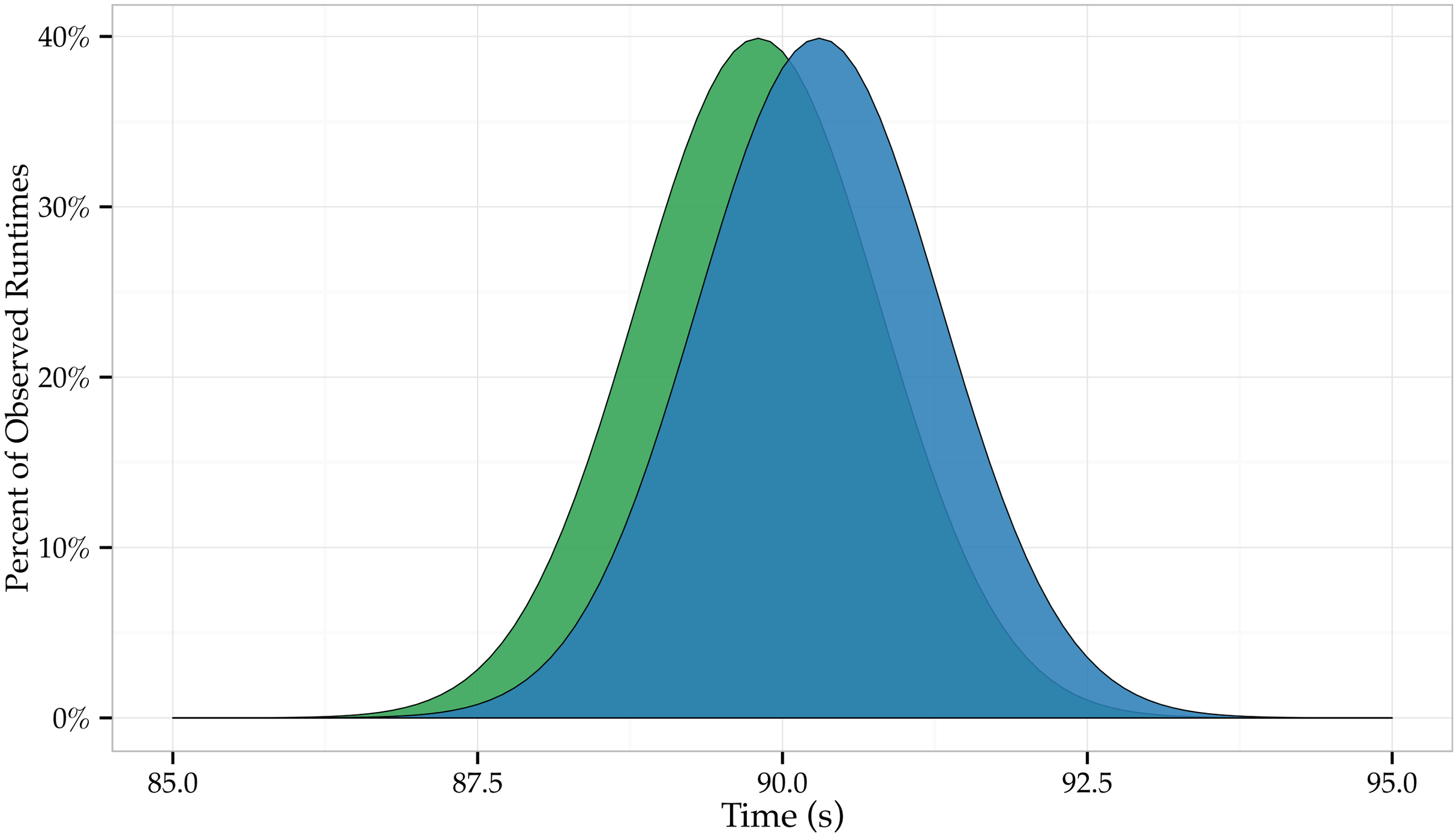
Run benchmarks as usual

A ×30 A′ ×30

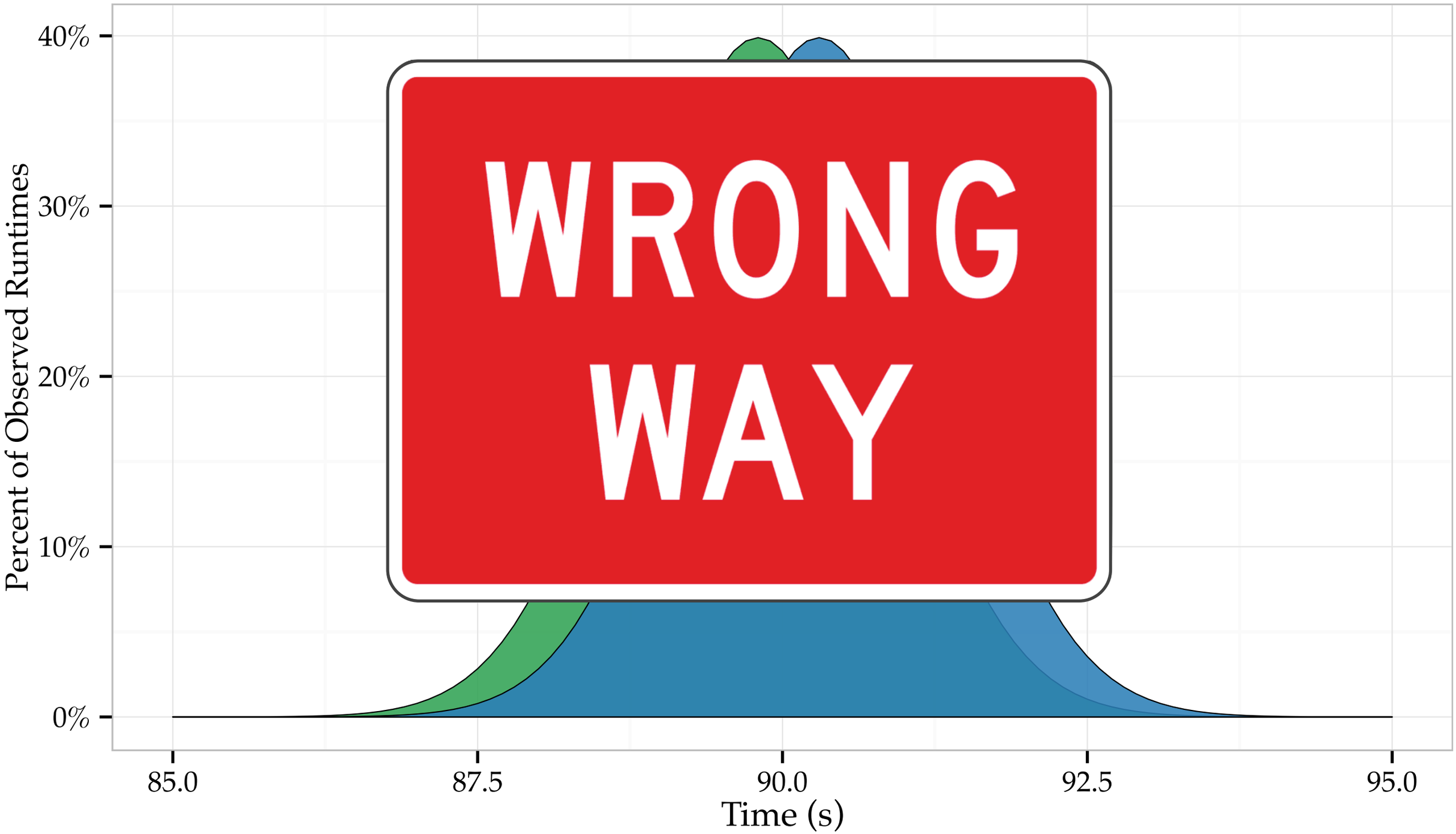
drop the results into R



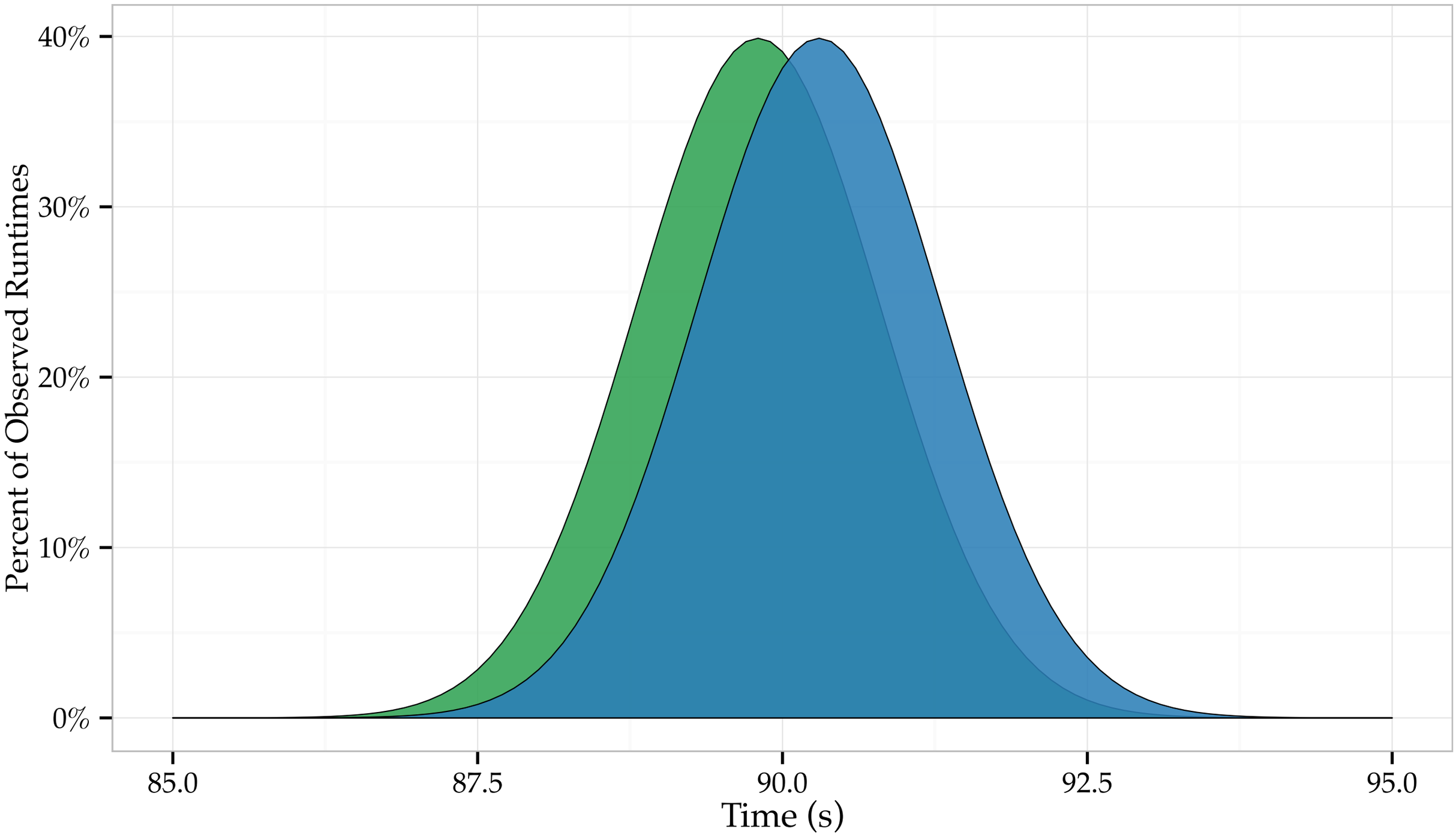
Is A′ faster than A?



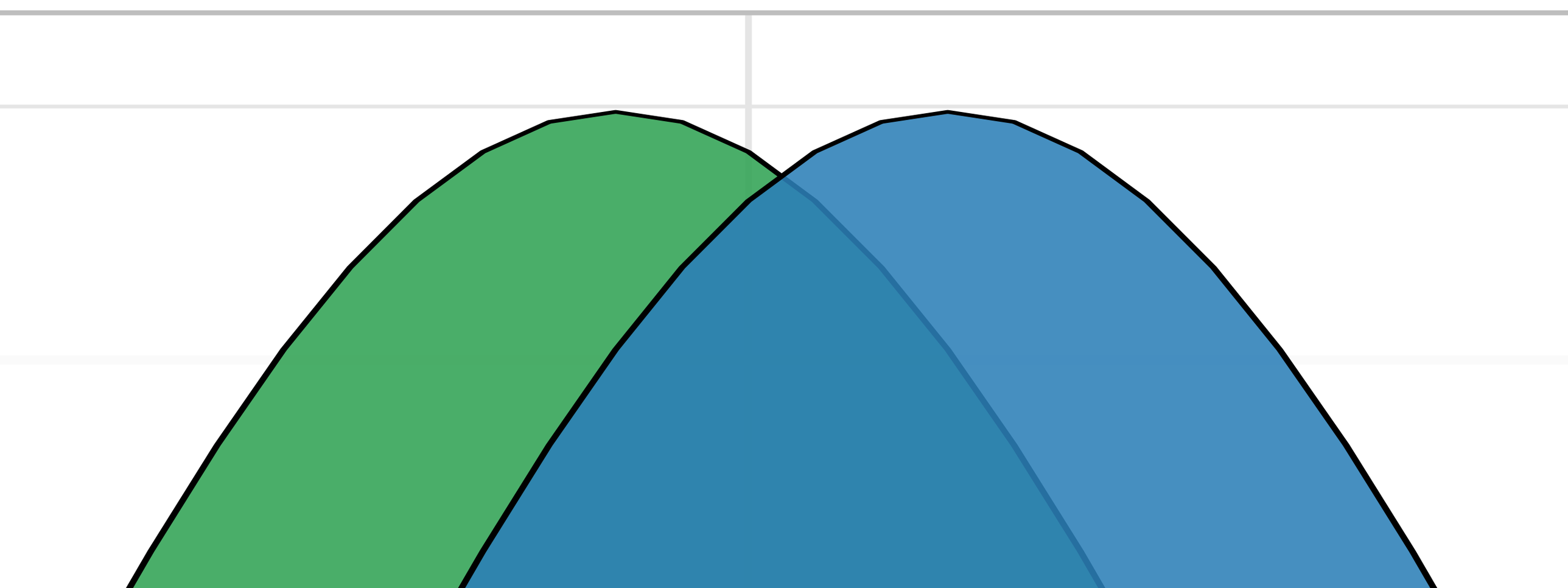
Is A′ faster than A?



If A′ = A

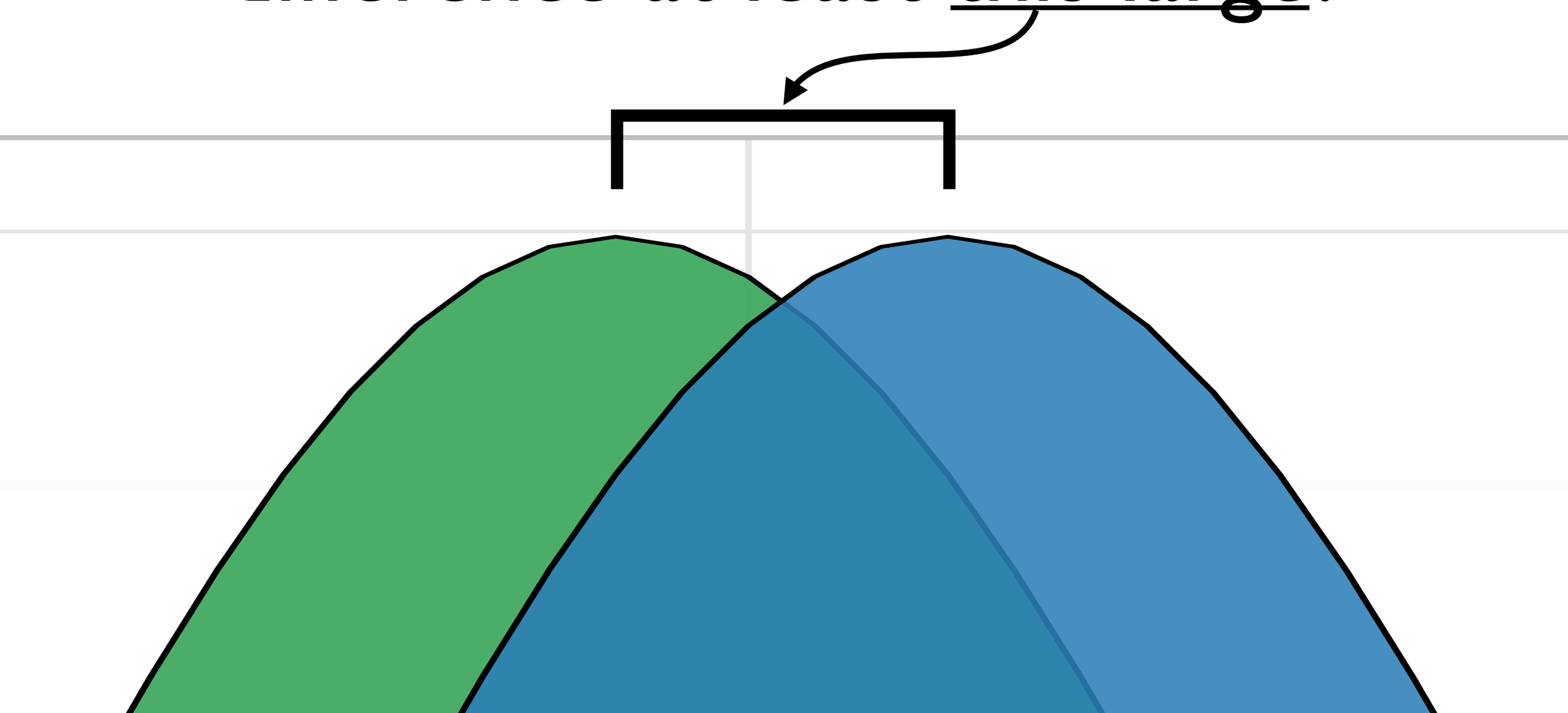

40%
30%
20%
10%
0%
Percent of Observed Runtimes
85.0
87.5
90.0
92.5
95.0
Time (s)

If A' = A



If A′ = A

what is the *probability* of measuring a difference at least this large?



The Student's t-test

If A′ = A

what is the *probability* of measuring a difference at least this large?

The Student's t-test

`t.test(times.A', times.A)`

If A′ = A

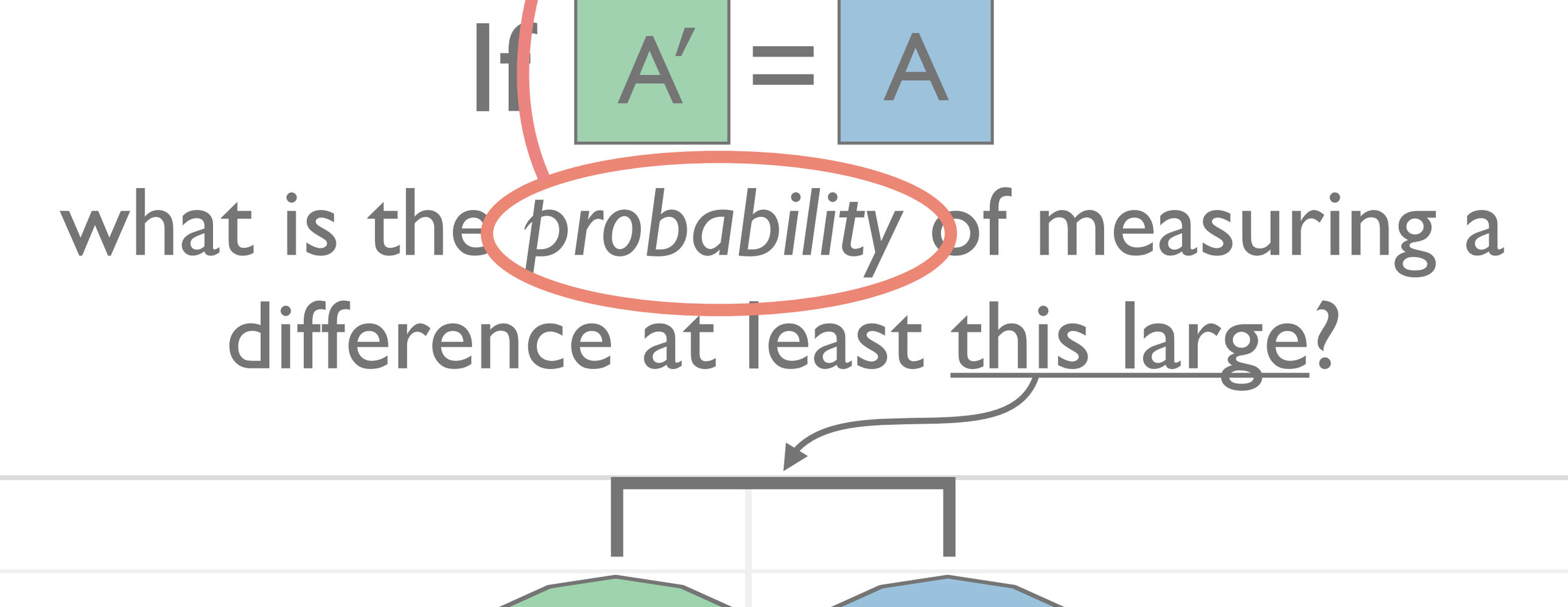
what is the *probability* of measuring a difference at least this large?

The Student's t-test

If p-value

If A′ = A

what is the *probability* of measuring a difference at least this large?



The Student's t-test

If p-value ≤ 5%

If A′ = A

what is the *probability* of measuring a difference at least this large?

The Student's t-test

If p-value $\leq$ 5%

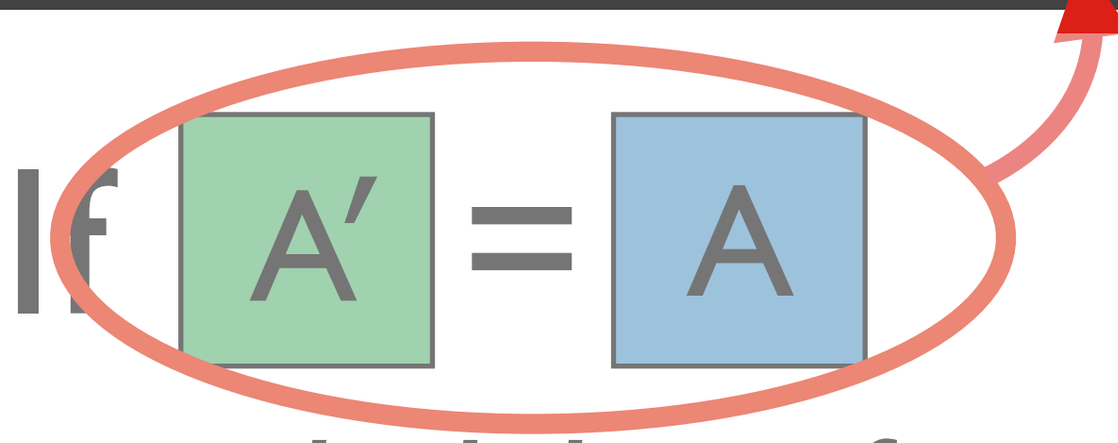
95% Confidence

If A′ = A

what is the *probability* of measuring a difference at least this large?

The Student's t-test

If p-value ≤ 5%
we reject the null hypothesis



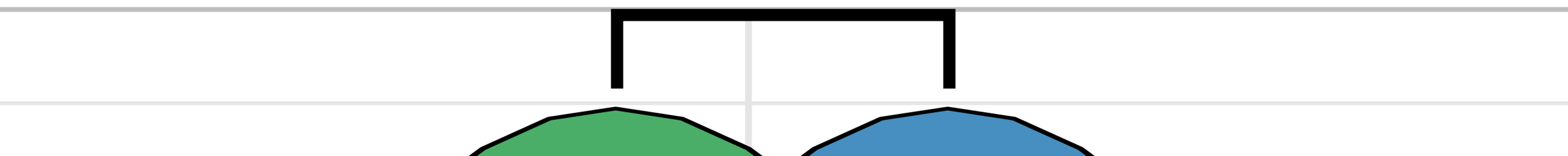
what is the *probability* of measuring a difference at least this large?

The Student's t-test

If p-value ≤ 5%
we reject the null hypothesis

$A' \neq A$

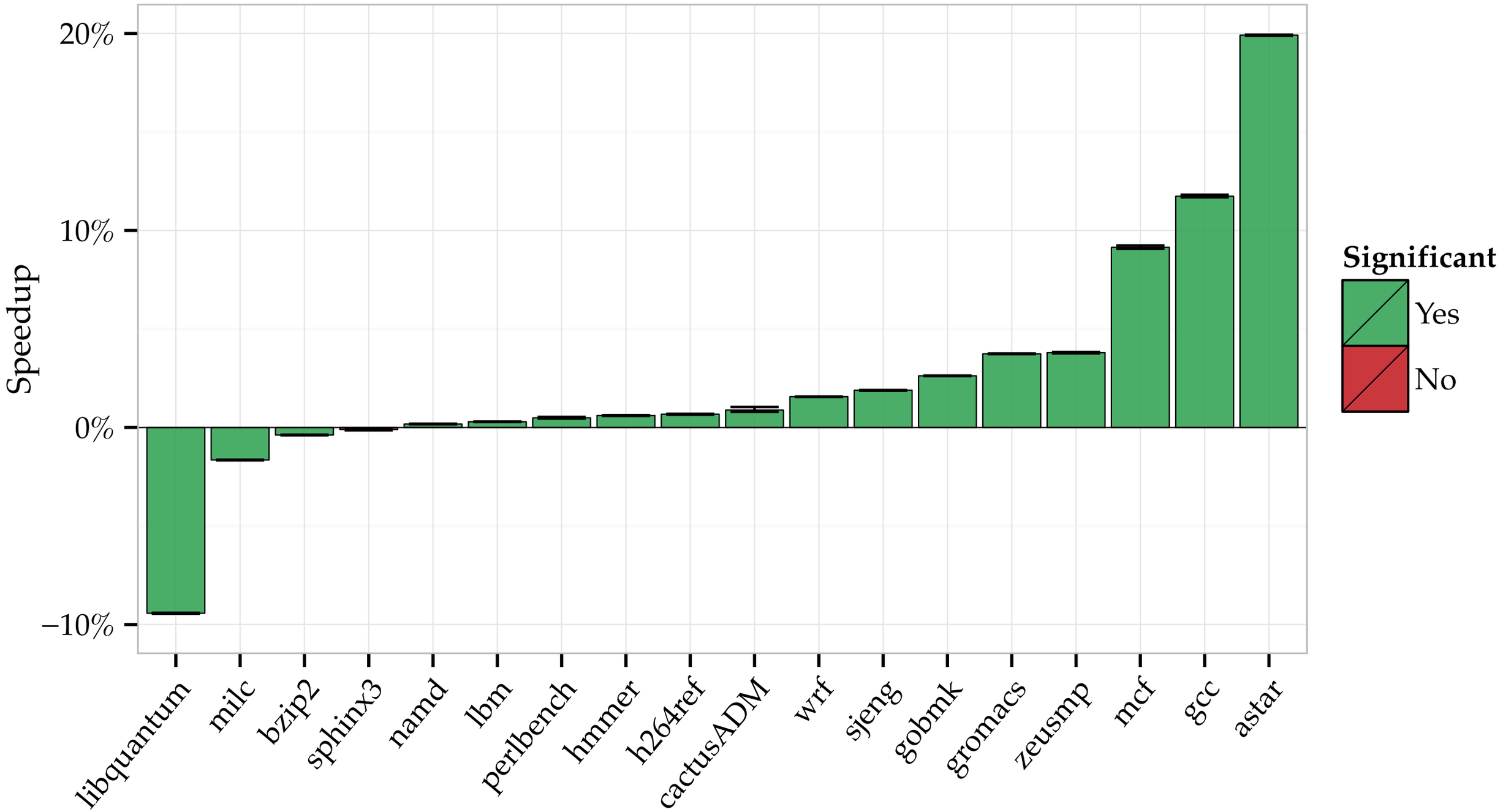
Random chance not responsible for the measured difference



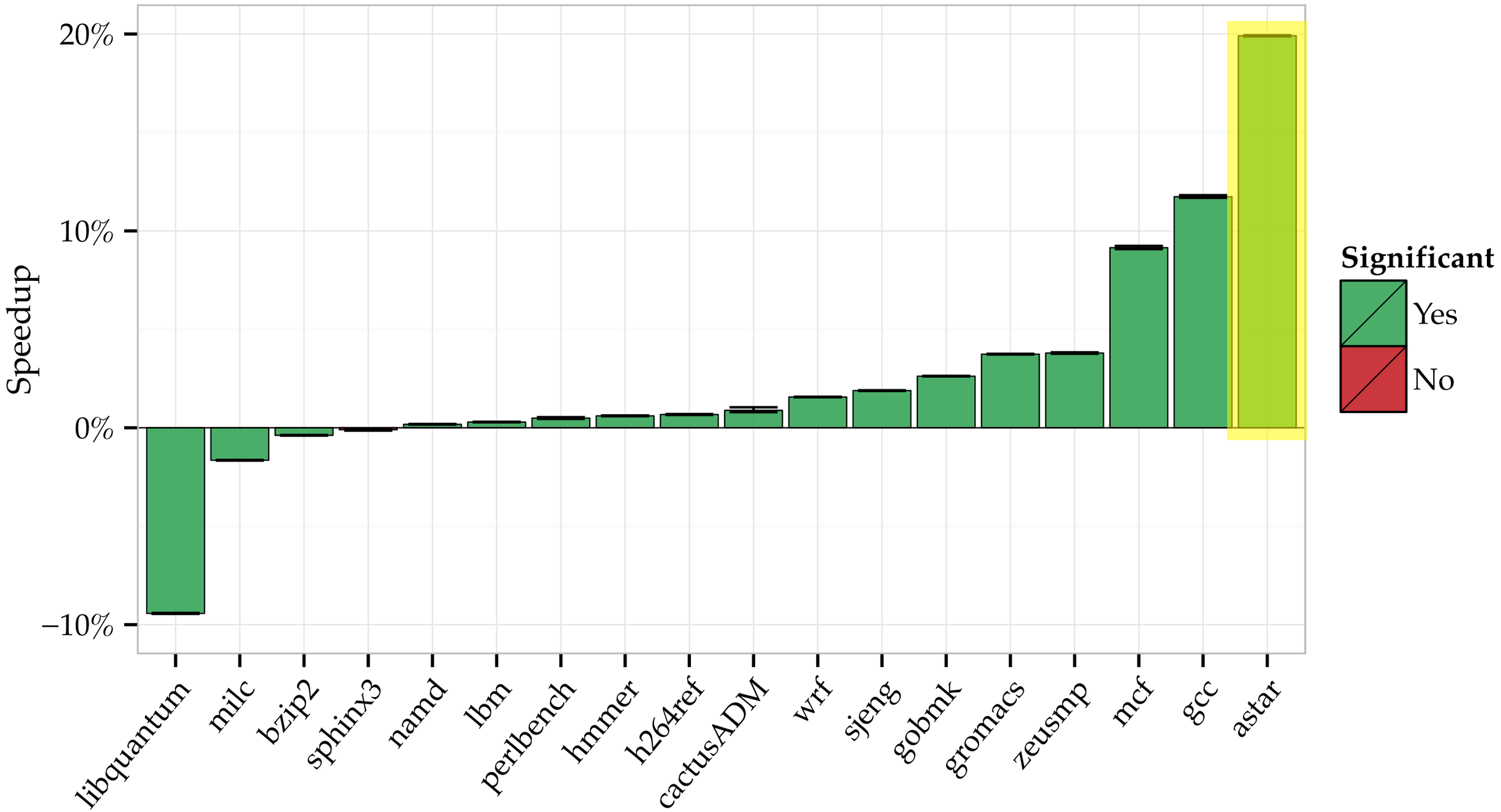
The difference is real

$$A' \neq A$$

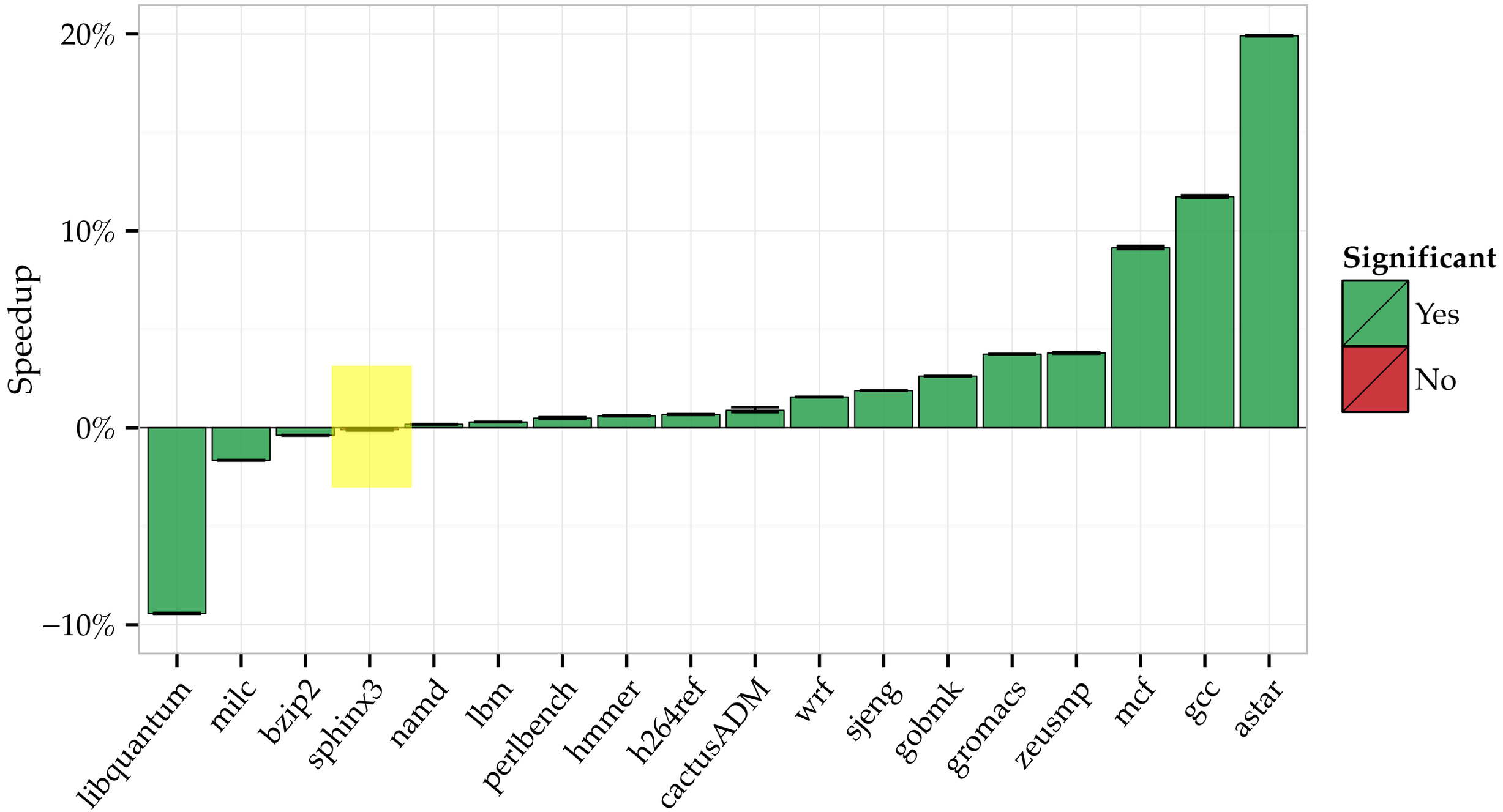
Speedup of -O2 over -O1



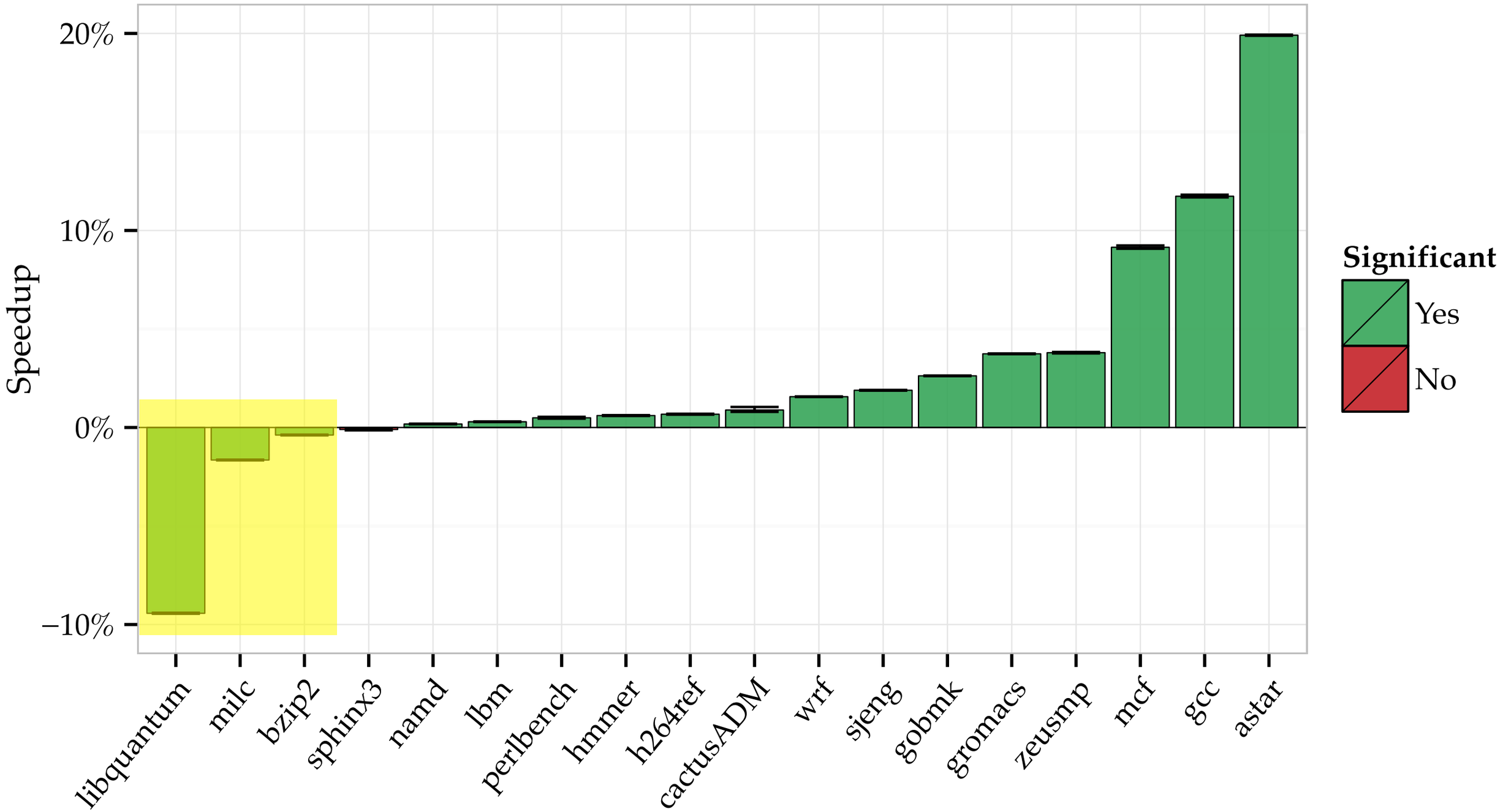
Speedup of -O2 over -O1



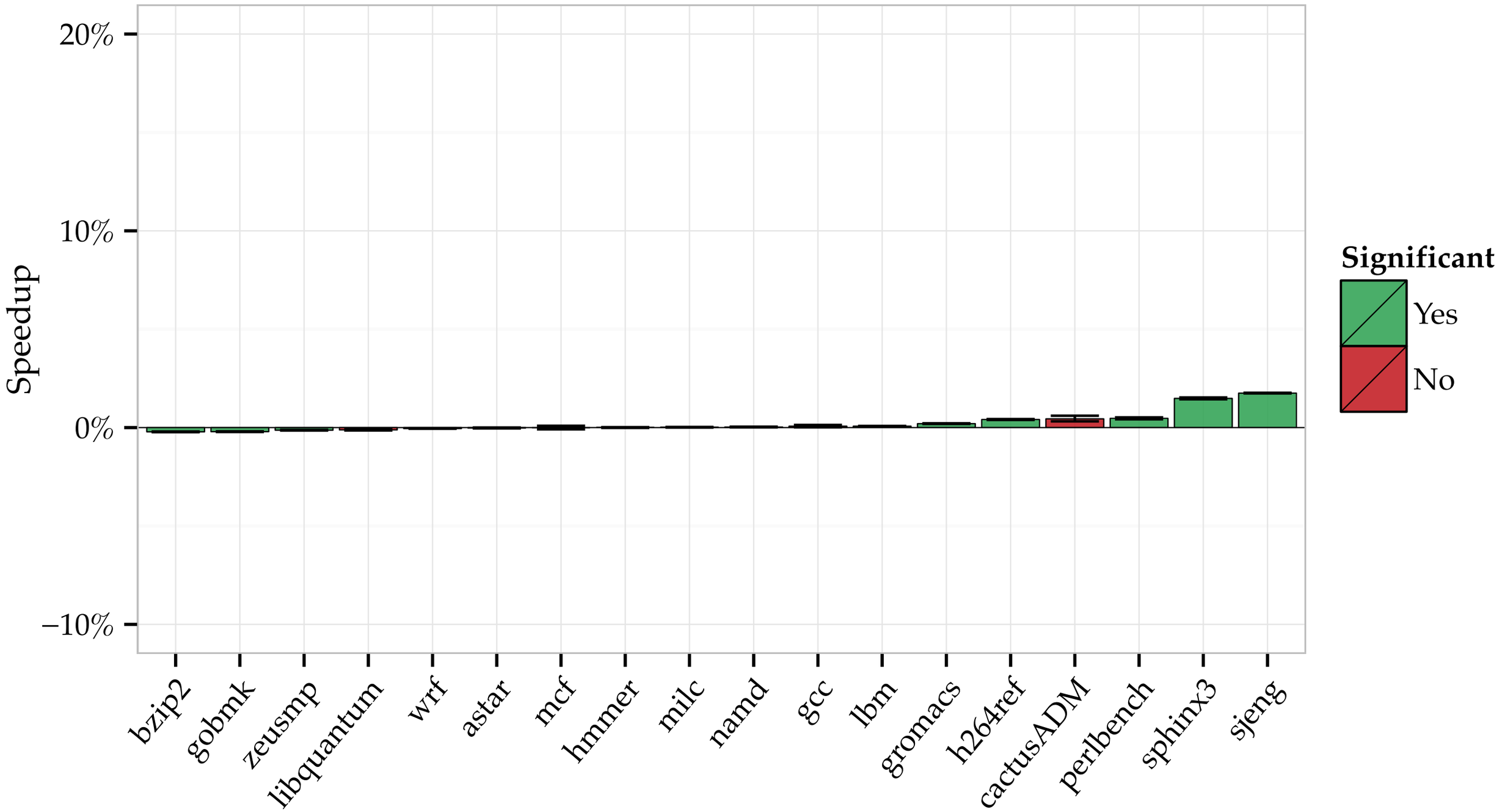
Speedup of -O2 over -O1



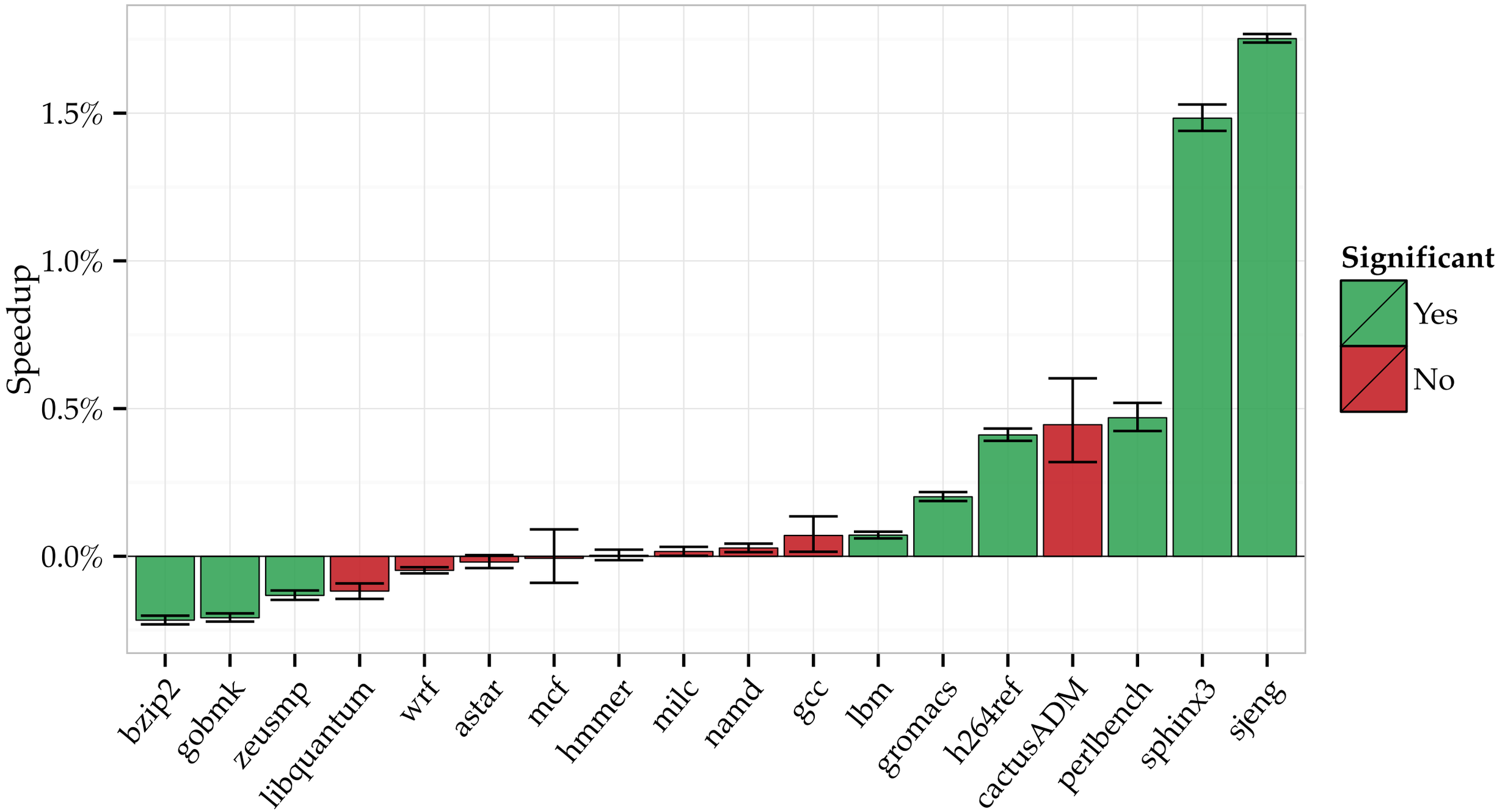
Speedup of -O2 over -O1



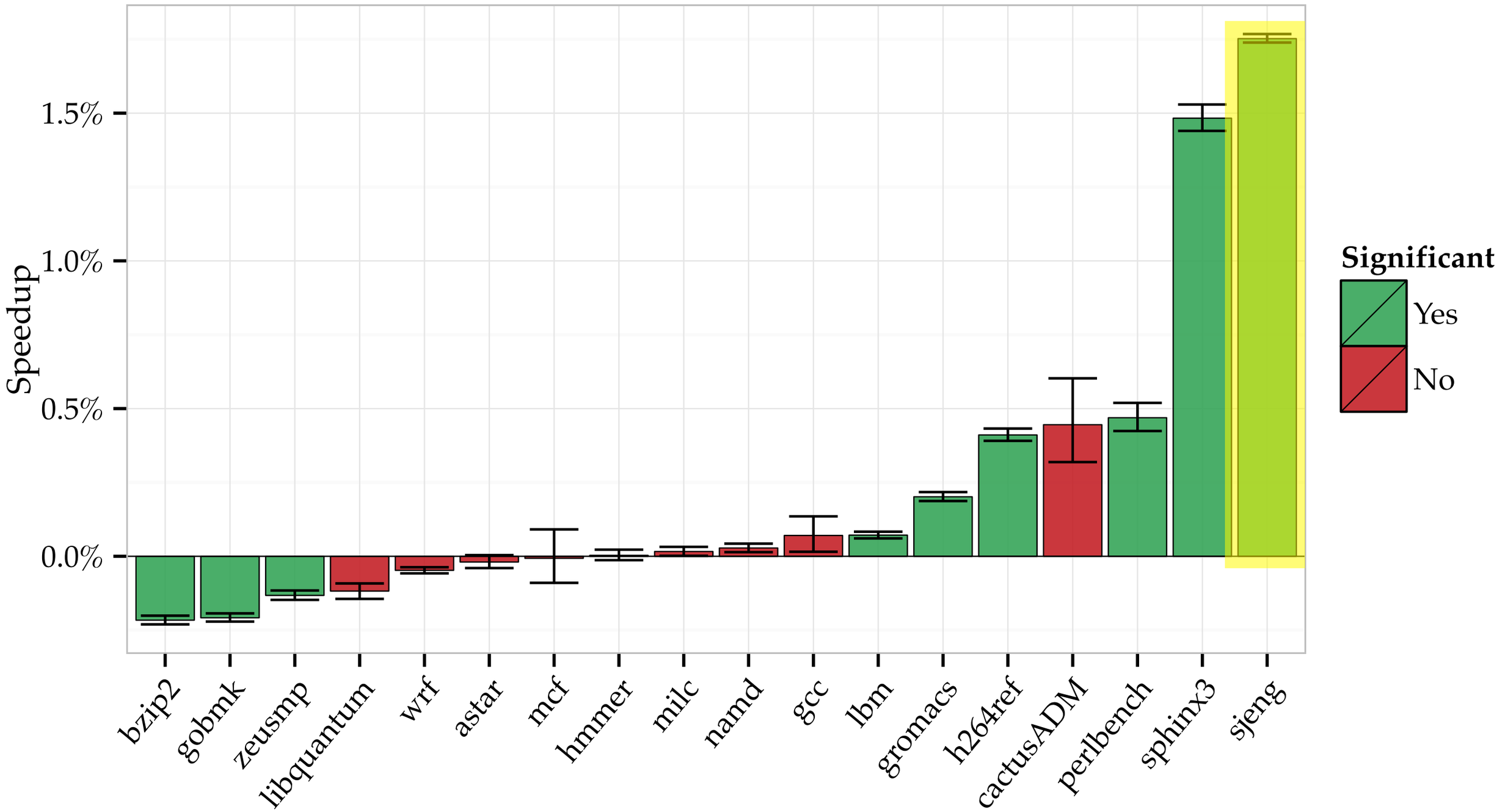
Speedup of -O3 over -O2



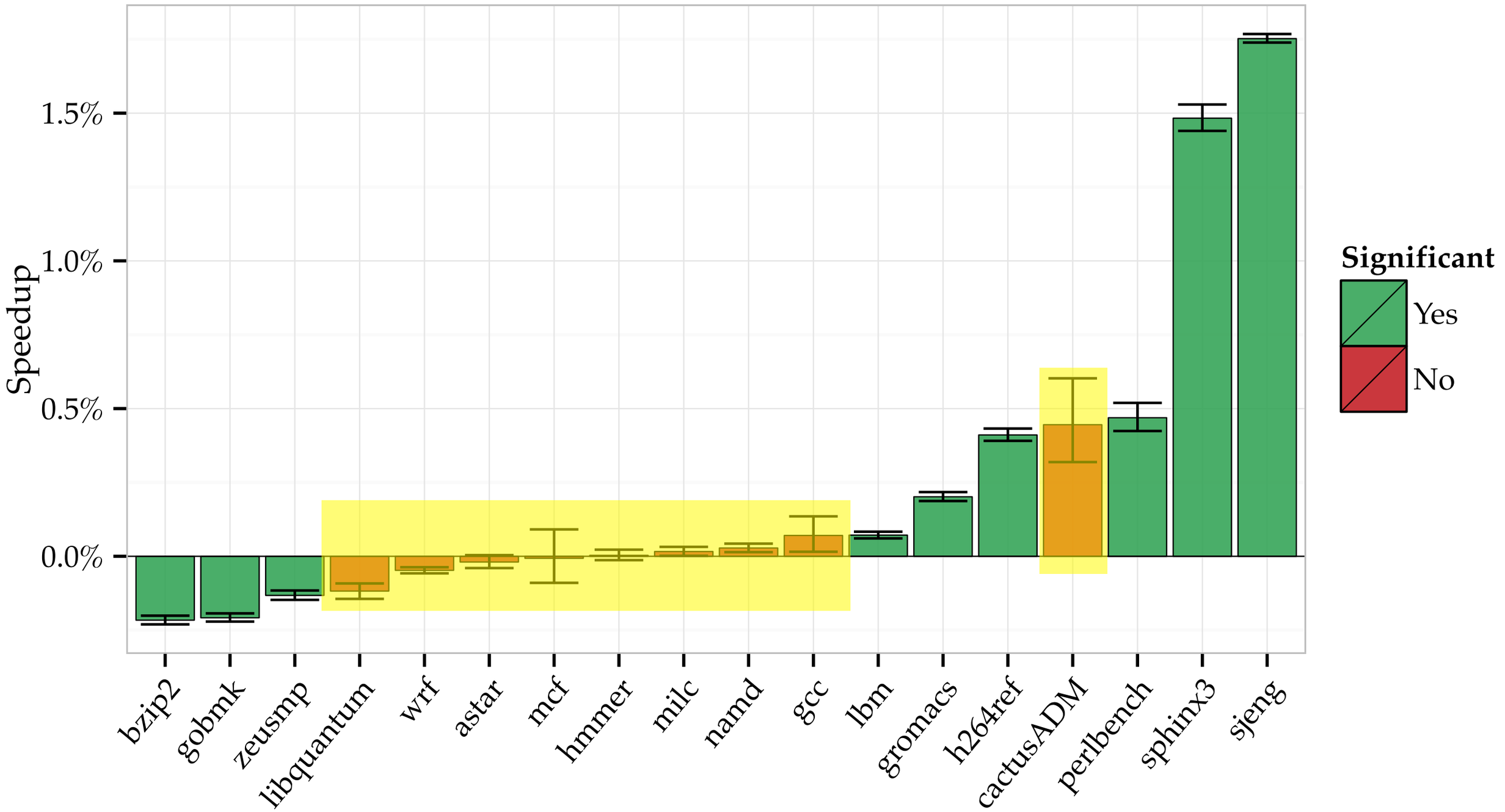
Speedup of -O3 over -O2



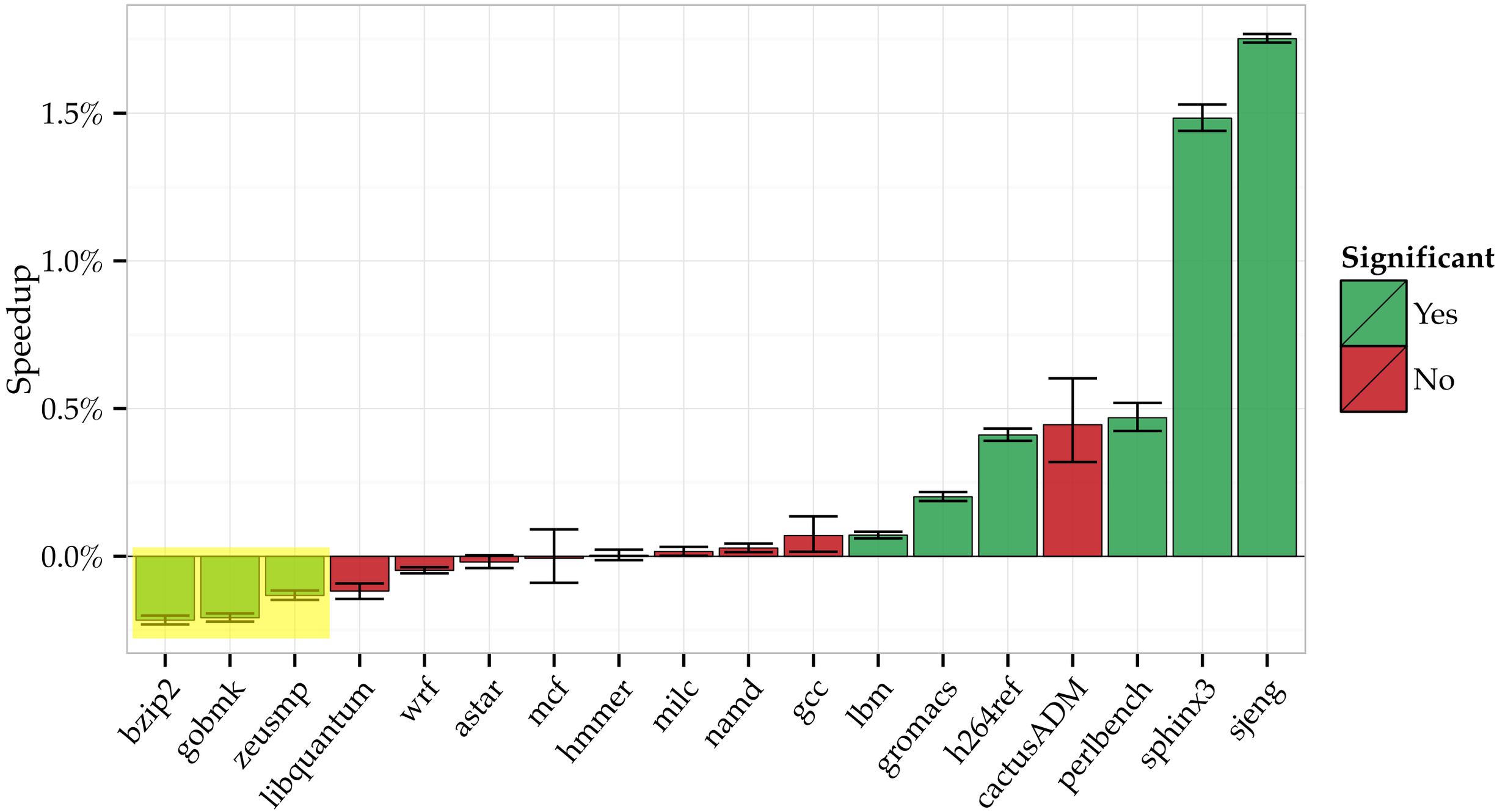
Speedup of -O3 over -O2



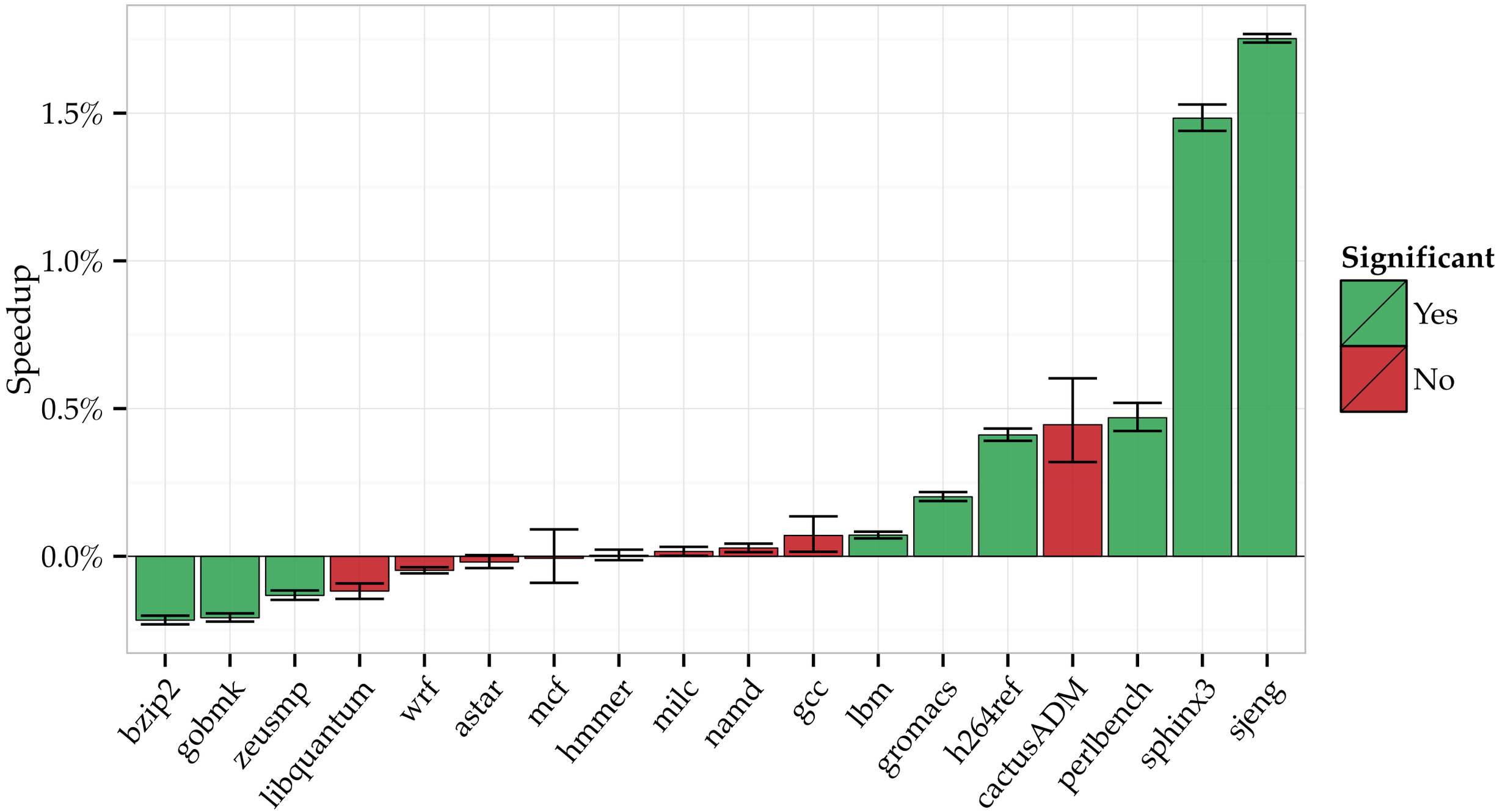
Speedup of -O3 over -O2



Speedup of -O3 over -O2



What do the results mean?



Comparing optimizations

-O2 ×30

-O3 ×30

Comparing optimizations

-O2 ×30

lbm ×30

-O3 ×30

lbm ×30

Comparing optimizations

-O2 ×30

lbm ×30

astar ×30

-O3 ×30

lbm ×30

astar ×30

Comparing optimizations

-O2 ×30

lbm ×30

astar ×30

-O3 ×30

lbm ×30

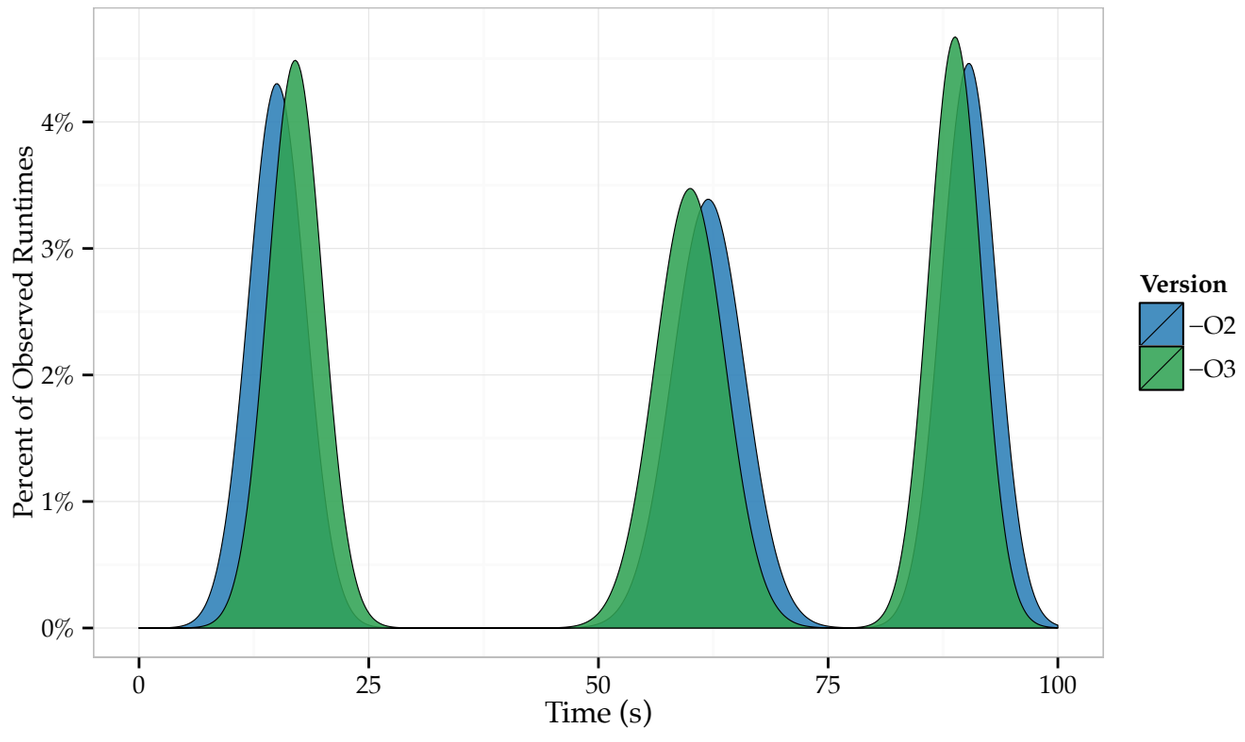
astar ×30

•••

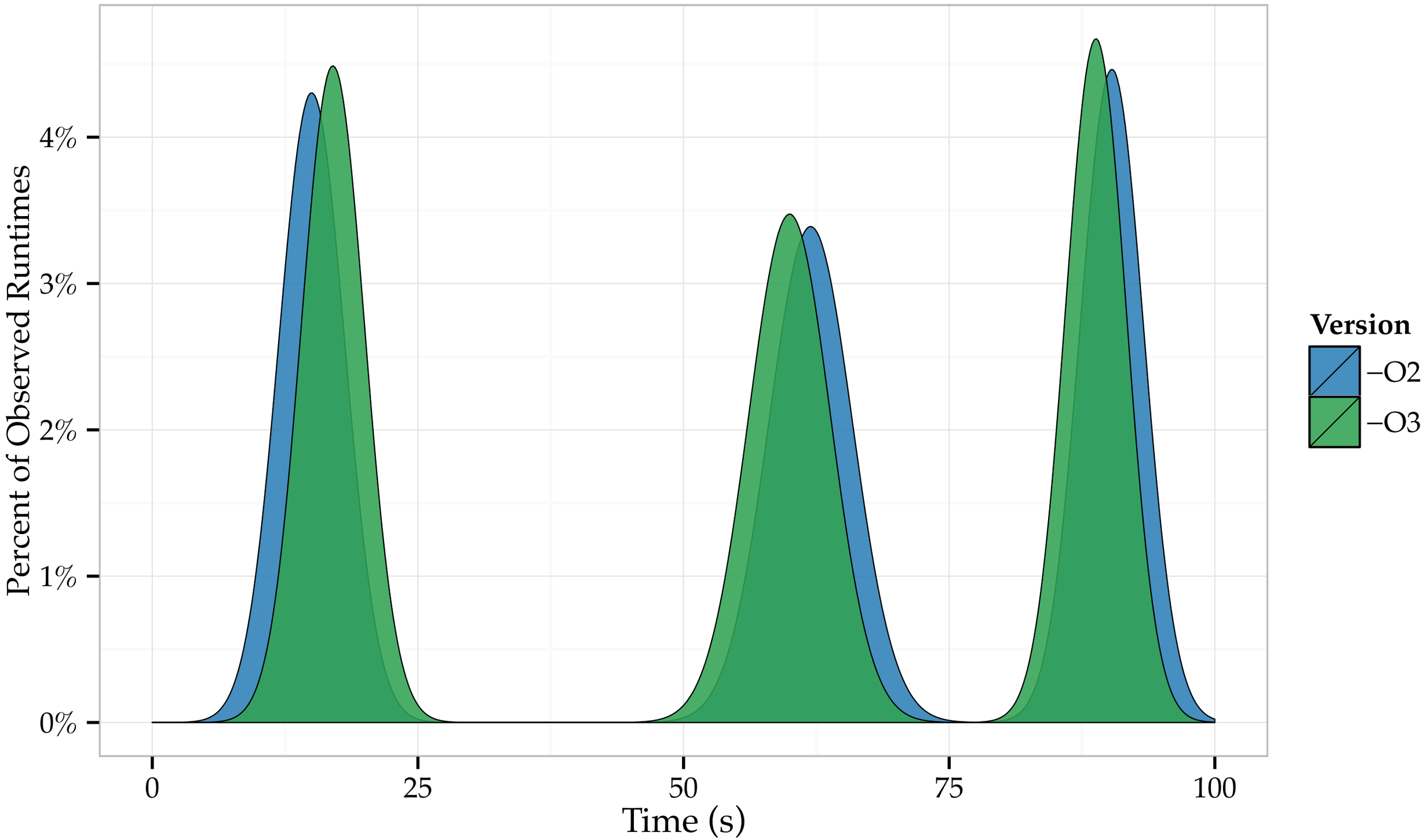
Comparing optimizations

-O2 ×30

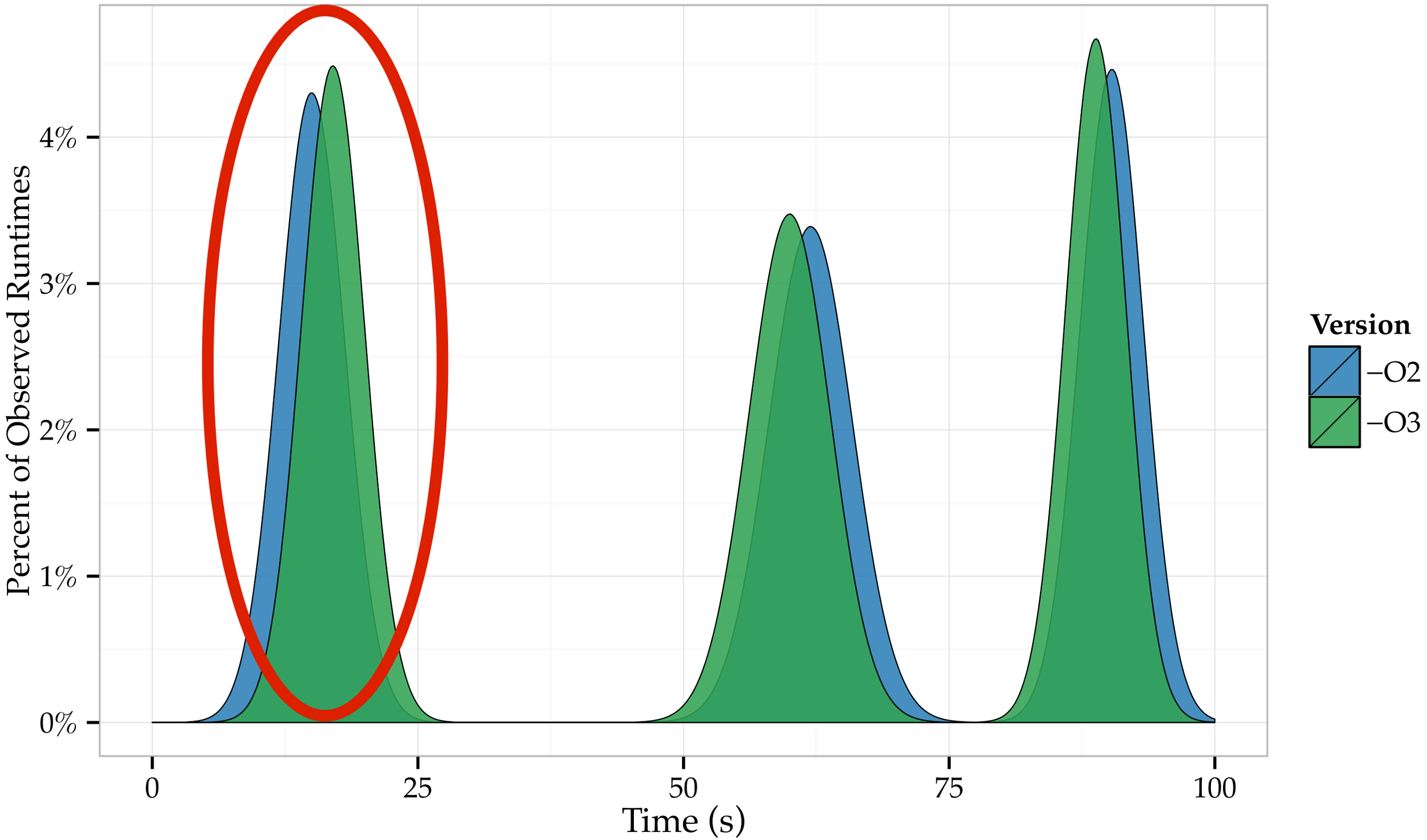
-O3 ×30



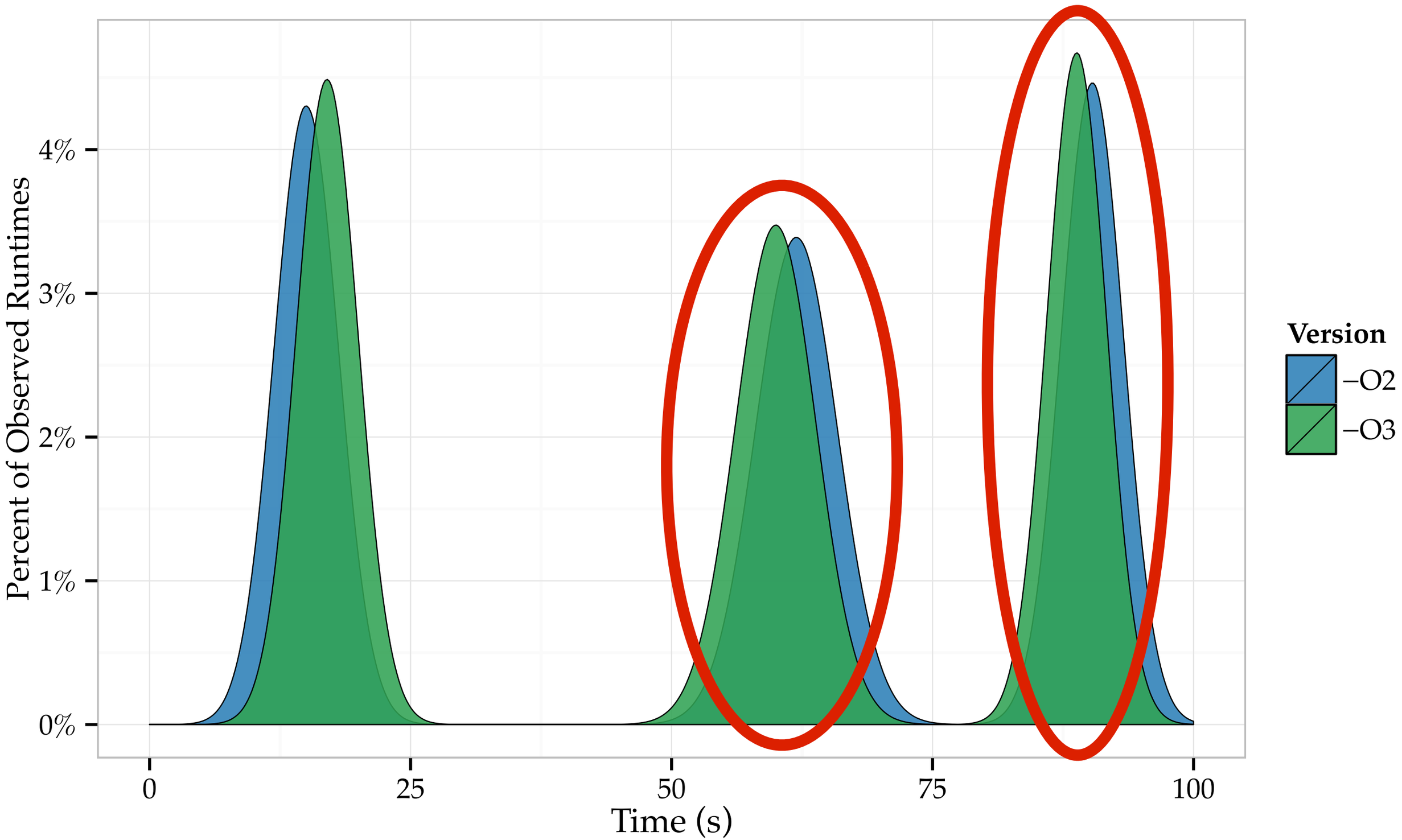
Is -O3 faster than -O2?



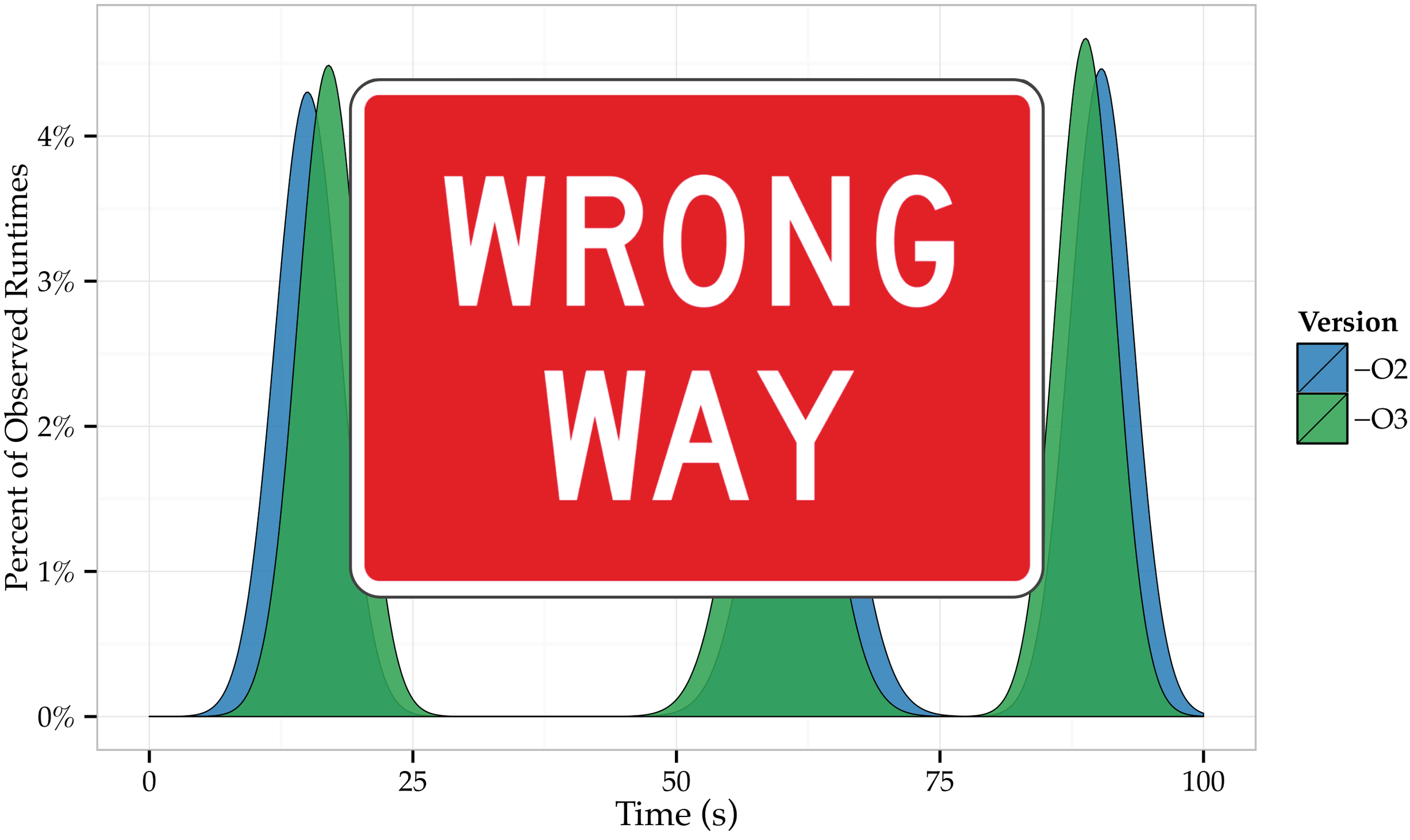
Is -O3 faster than -O2?



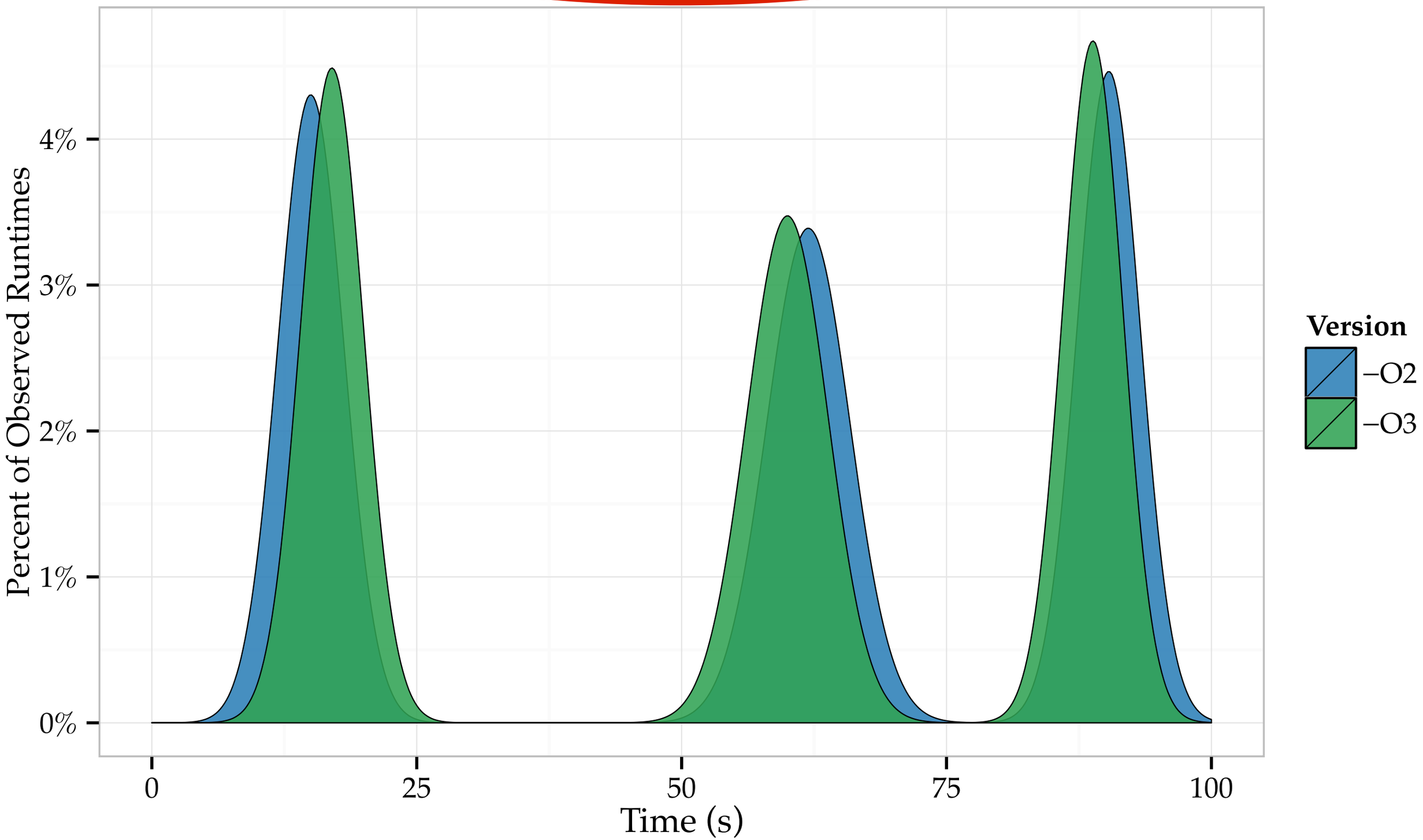
Is -O3 faster than -O2?



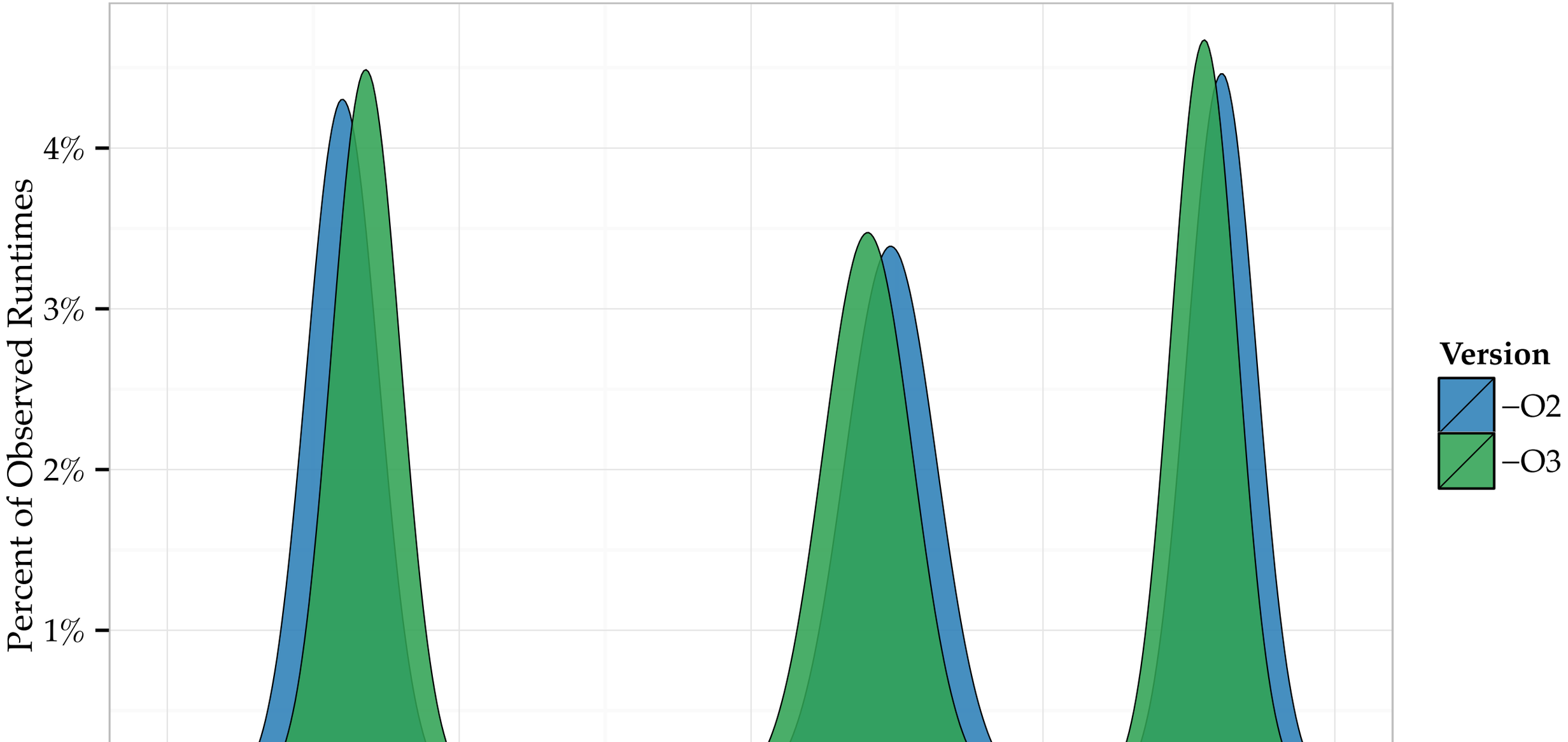
Is -O3 faster than -O2?



If -O3 = -O2

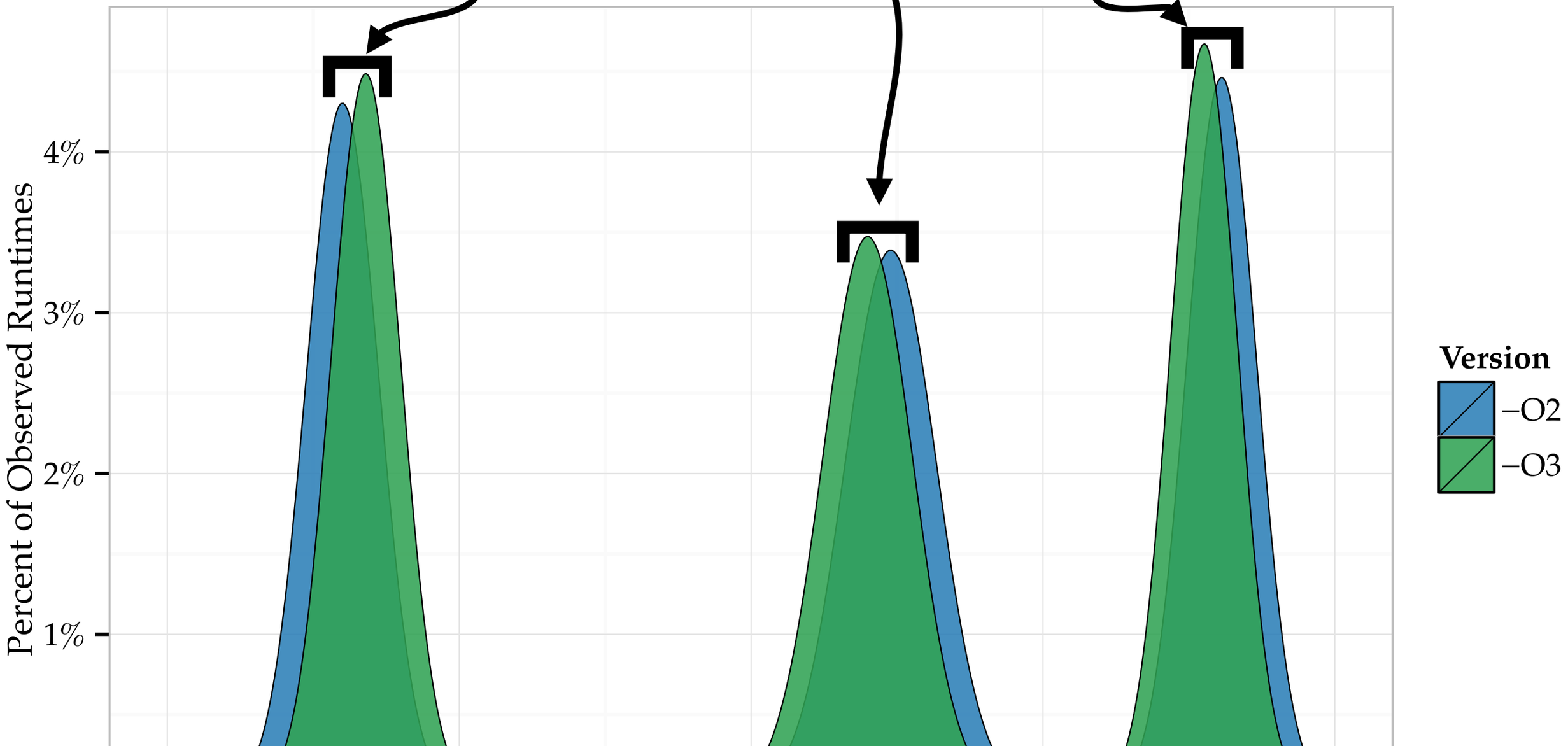

Percent of Observed Runtimes
4%
3%
2%
1%
0%
0
25
50
75
100
Time (s)
Version
–O2
–O3

If -O3 = -O2



If -O3 = -O2

what is the probability of measuring these differences?



Analysis of Variance

If -O3 = -O2

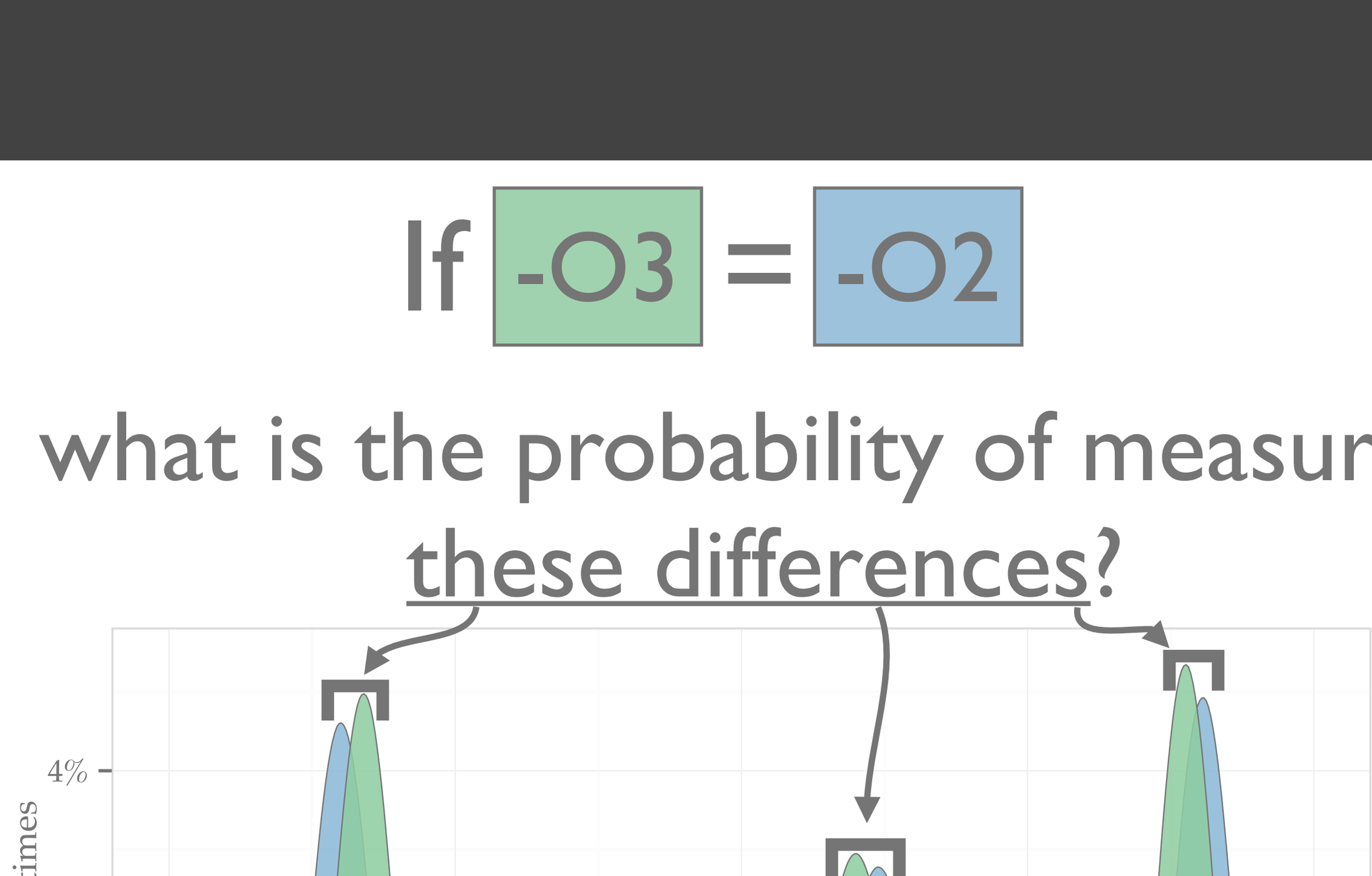
what is the probability of measuring these differences?

Analysis of Variance

`aov(time~opt+Error(benchmark/opt), times)`

If -O3 = -O2

what is the probability of measuring these differences?



Analysis of Variance

If p-value

If -O3 = -O2

what is the probability of measuring these differences?

Analysis of Variance

If p-value ≤ 5%

If -O3 = -O2

what is the probability of measuring these differences?

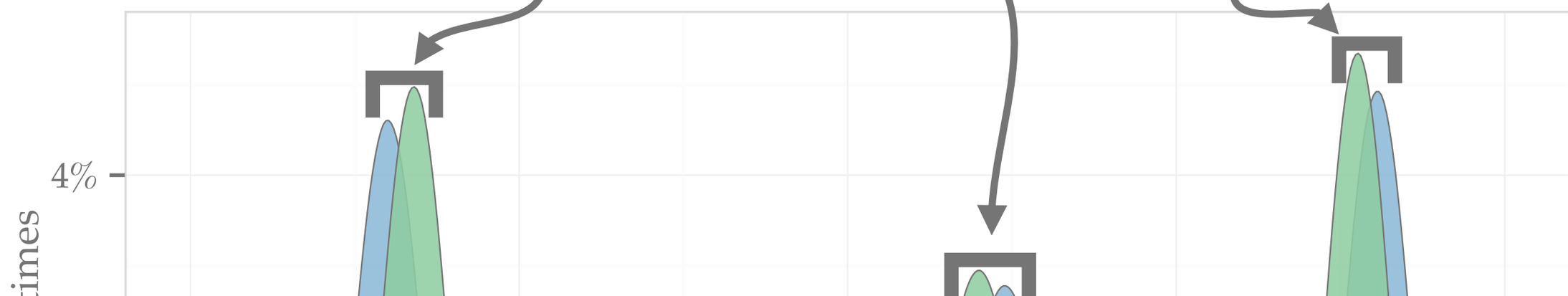
Analysis of Variance

If p-value ≤ 5%

we reject the null hypothesis

If -O3 = -O2

what is the probability of measuring these differences?



Analysis of Variance

If p-value ≤ 5%

we reject the null hypothesis

-O3 vs -O2

Analysis of Variance

If p-value ≤ 5%

we reject the null hypothesis

-O3 vs -O2

p-value = 26.4%

Analysis of Variance

If p-value ≤ 5%
we reject the null hypothesis

-O3 vs -O2

p-value = 26.4%

one in four experiments will show an effect that does not exist!

Analysis of Variance

If p-value ≤ 5%

we reject the null hypothesis

-O3 vs **-O2**

p-value = 26.4%

fail to reject the null hypothesis

Analysis of Variance

If p-value ≤ 5%

we reject the null hypothesis

The effect of **-O3** over **-O2** is indistinguishable from noise

Analysis of Variance

If p-value ≤ 5%

we reject the null hypothesis

The effect of -O3 over -O2 is indistinguishable from noise

Did STABILIZER hide the effect?

Memory layout affects performance

makes performance evaluation difficult

STABILIZER eliminates the effect of layout

random layout enables sound performance evaluation

Case Studies

showed that -O3 does not have a statistically significant effect across our benchmarks

Memory layout affects performance

makes performance evaluation difficult

STABILIZER eliminates the effect of layout

random layout enables sound performance evaluation

Case Studies

showed that -O3 does not have a statistically significant effect across our benchmarks

Memory layout affects performance

makes performance evaluation difficult

STABILIZER eliminates the effect of layout

random layout enables sound performance evaluation

Case Studies

showed that -O3 does not have a statistically significant effect across our benchmarks

STABILIZER

Memory layout affects performance

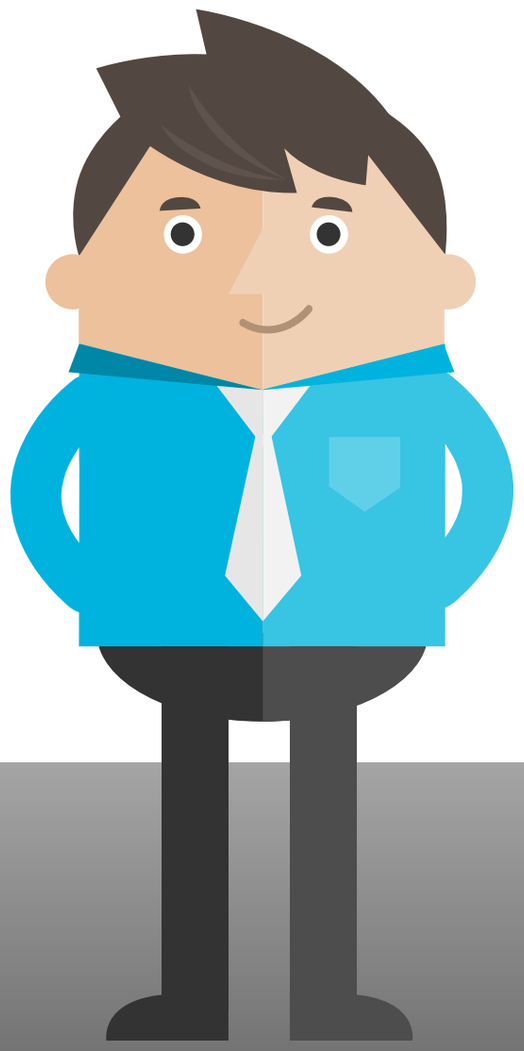
makes performance evaluation difficult

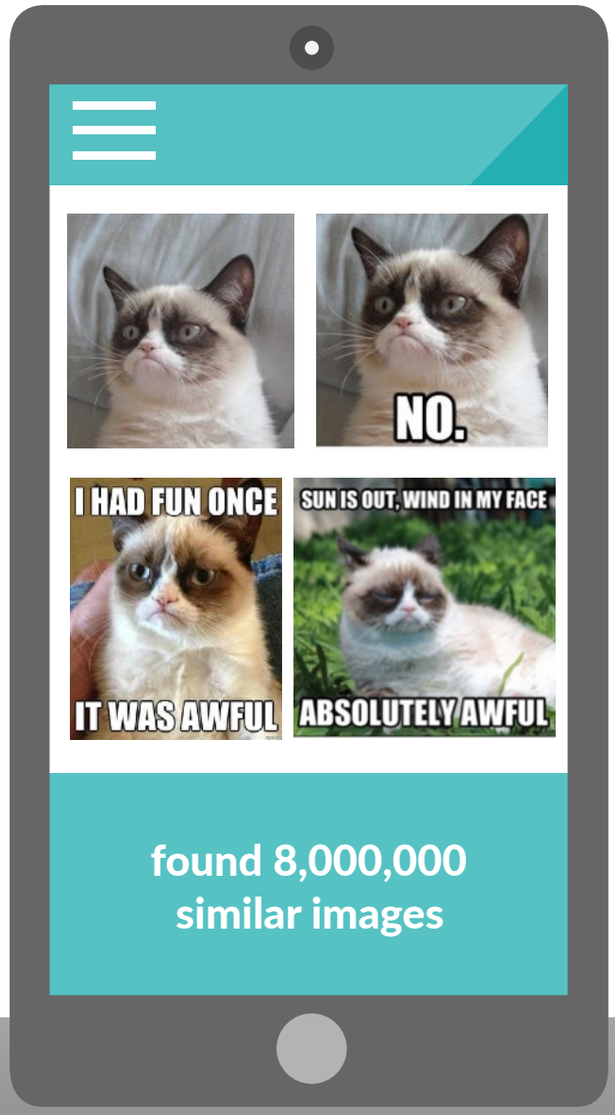
STABILIZER eliminates the effect of layout

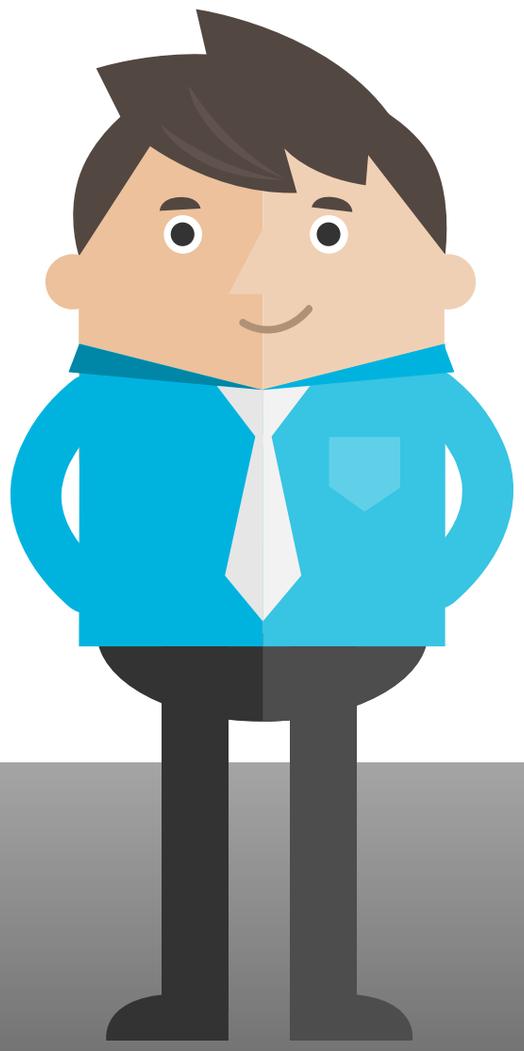
random layout enables sound performance evaluation

Case Studies

showed that -O3 does not have a statistically significant effect across our benchmarks

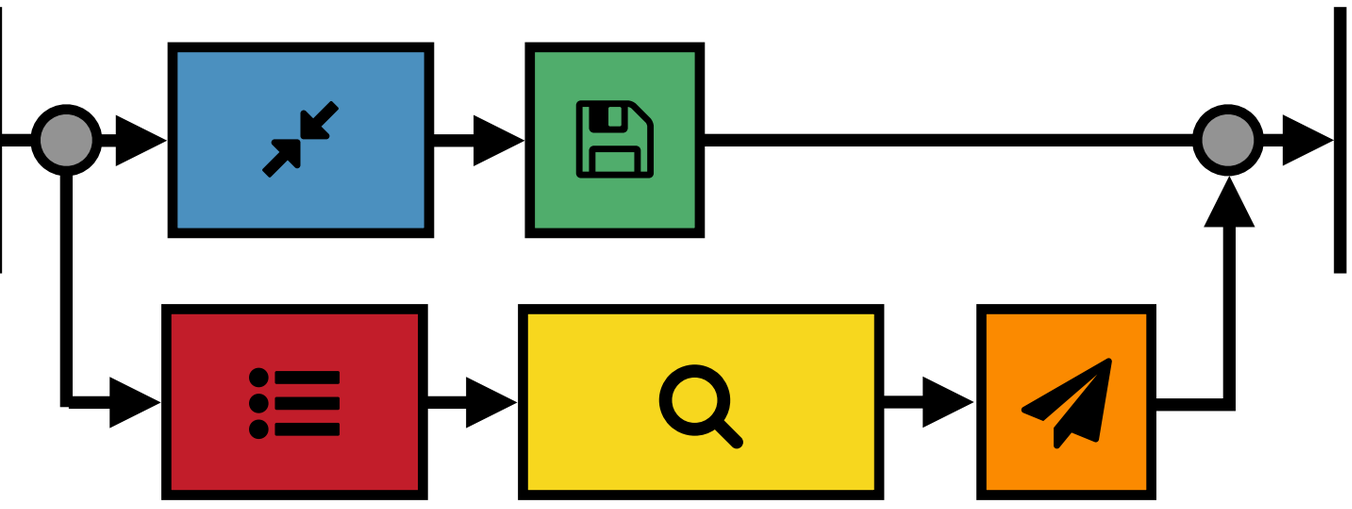



NO.
I HAD FUN ONCE
IT WAS AWFUL
SUN IS OUT, WIND IN MY FACE
ABSOLUTELY AWFUL
found 8,000,000
similar images

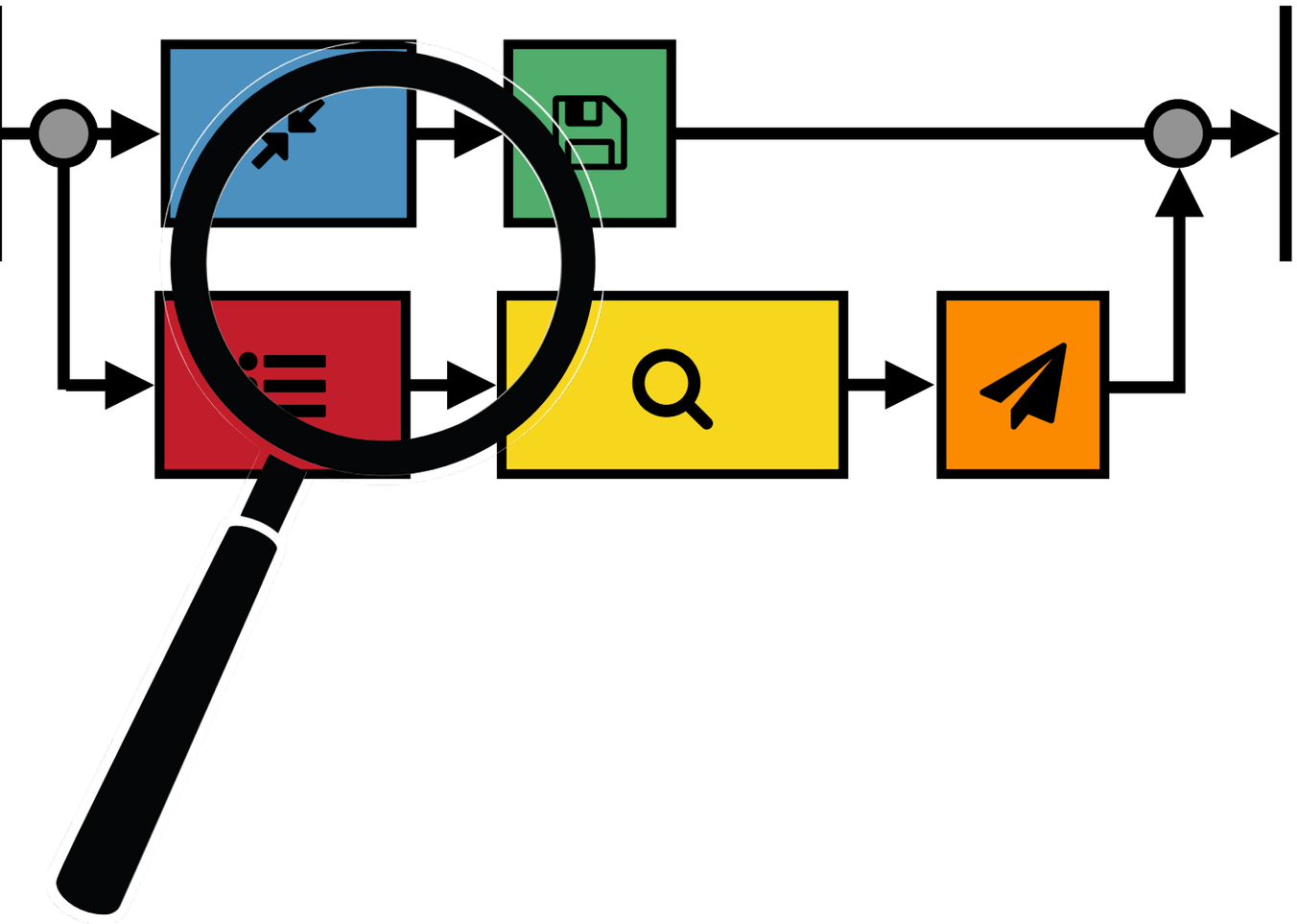



NO.
I HAD FUN ONCE
IT WAS AWFUL
SUN IS OUT, WIND IN MY FACE
ABSOLUTELY AWFUL
found 8,000,000
similar images

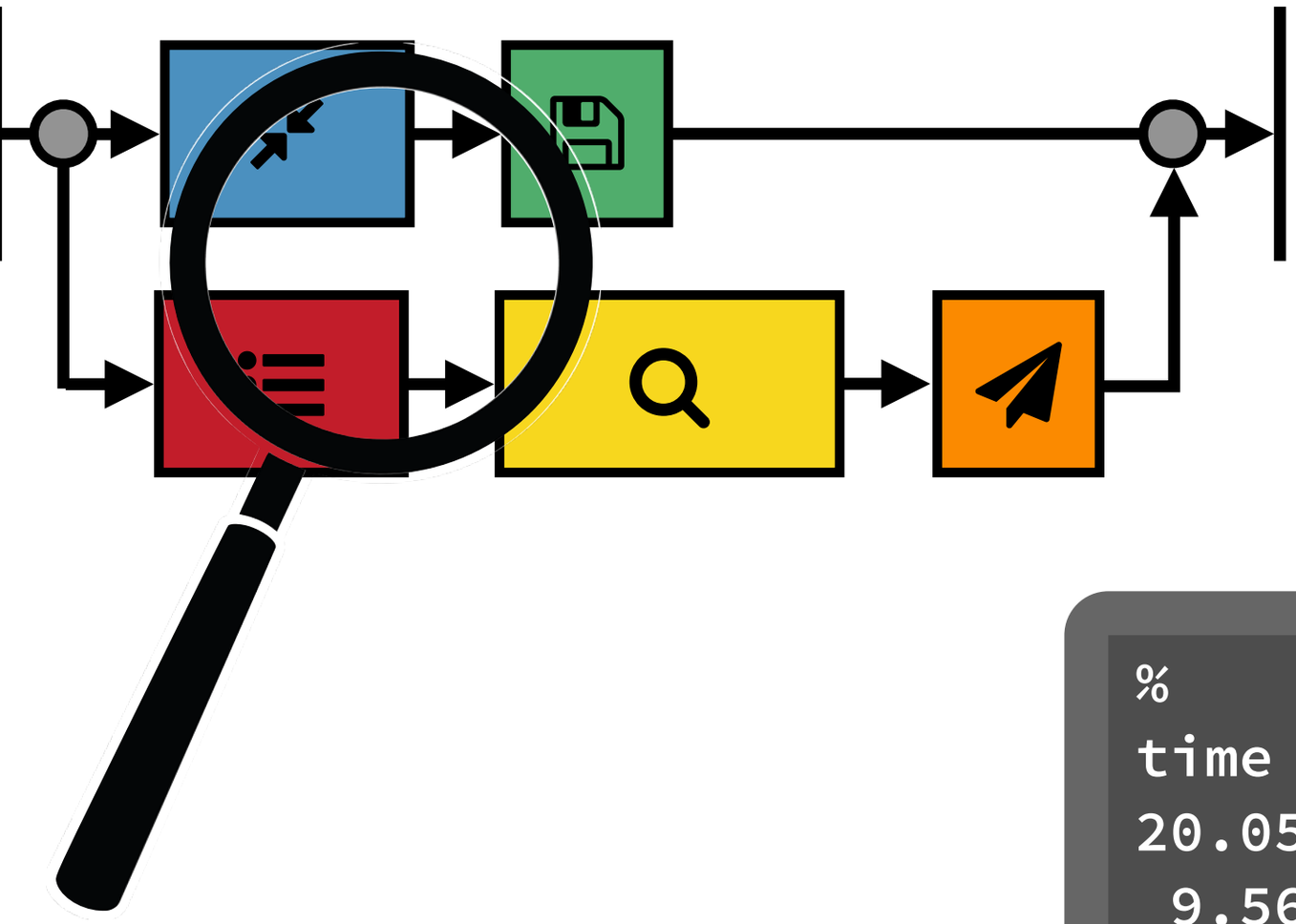
Software Profilers



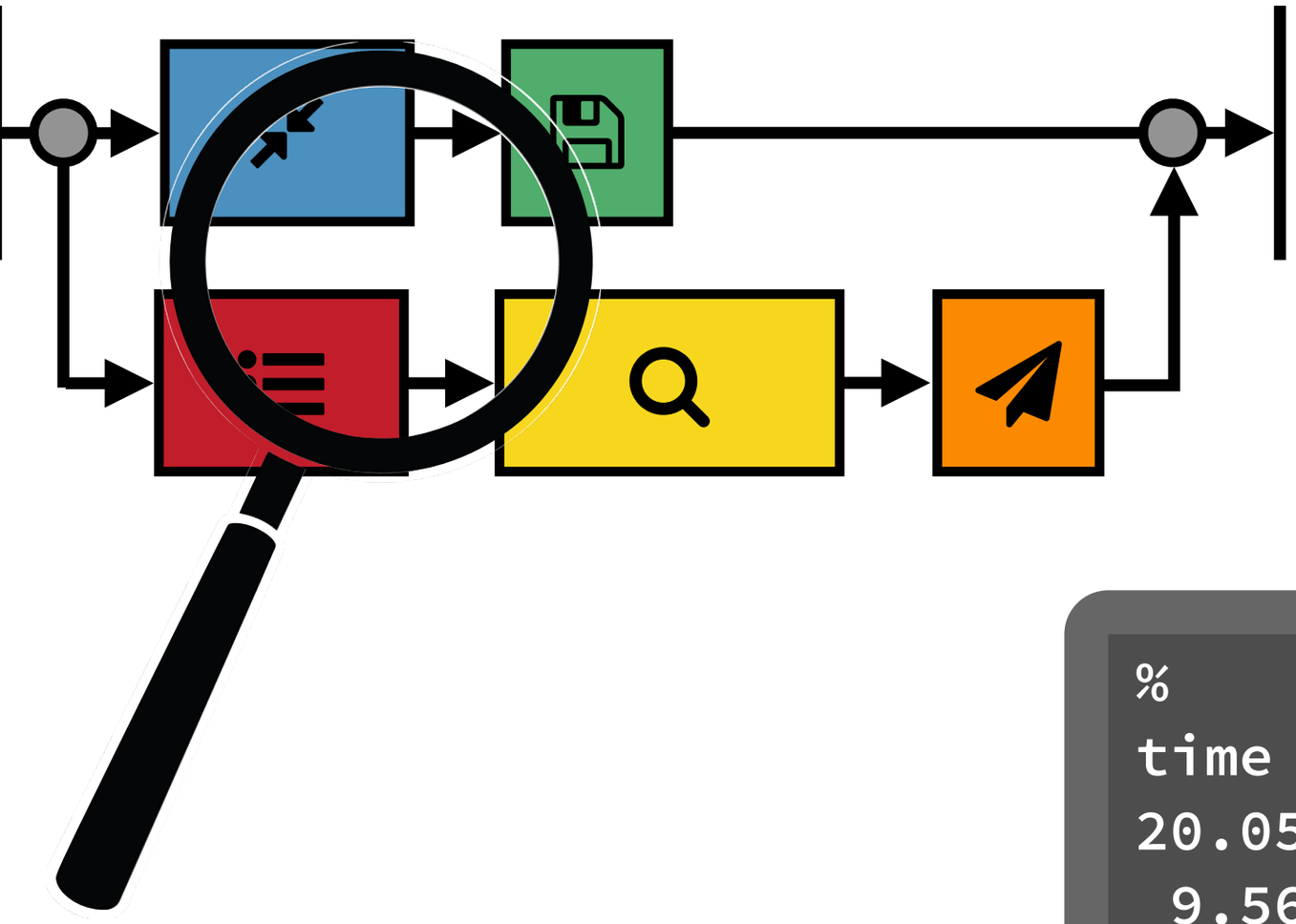
Software Profilers



Software Profilers



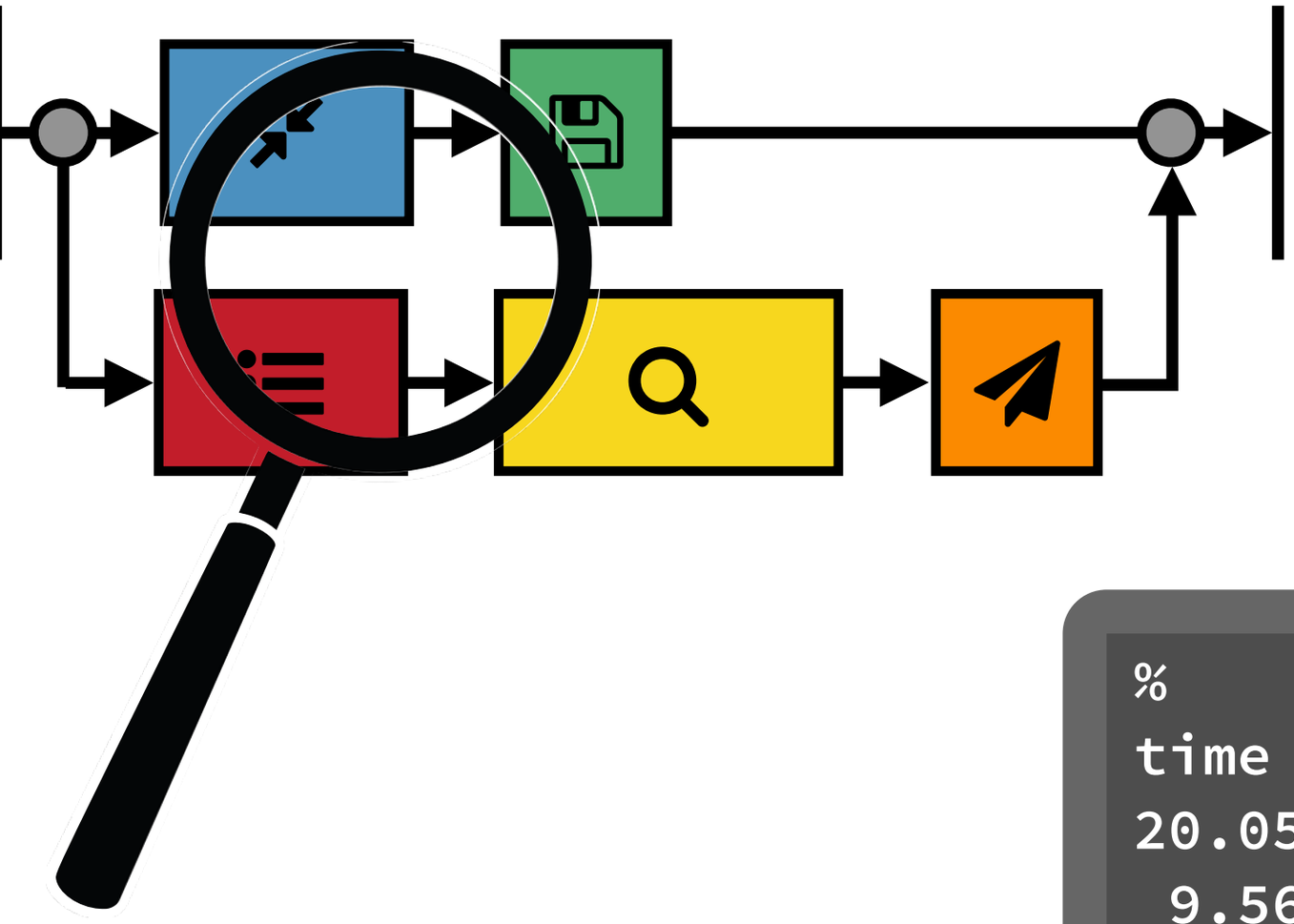
Software Profilers



Number of calls to each function

```
%       cumulative
time    seconds     calls     name
20.05    8.02       1
 9.56    3.82       1
19.95    7.98       1
45.19   11.31       1
 5.25    2.10       1
```

Software Profilers

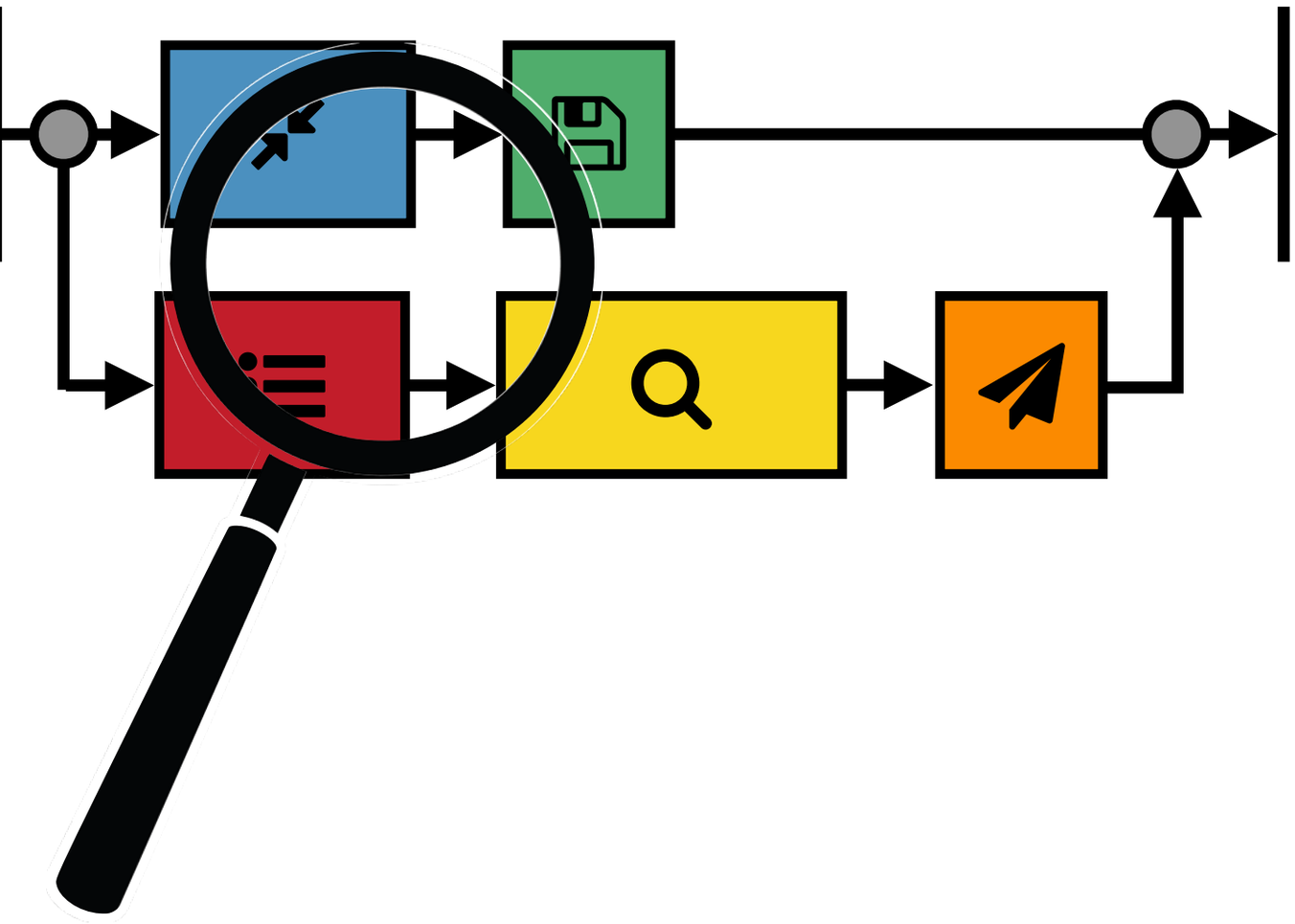


Number of calls to each function

Runtime for each function

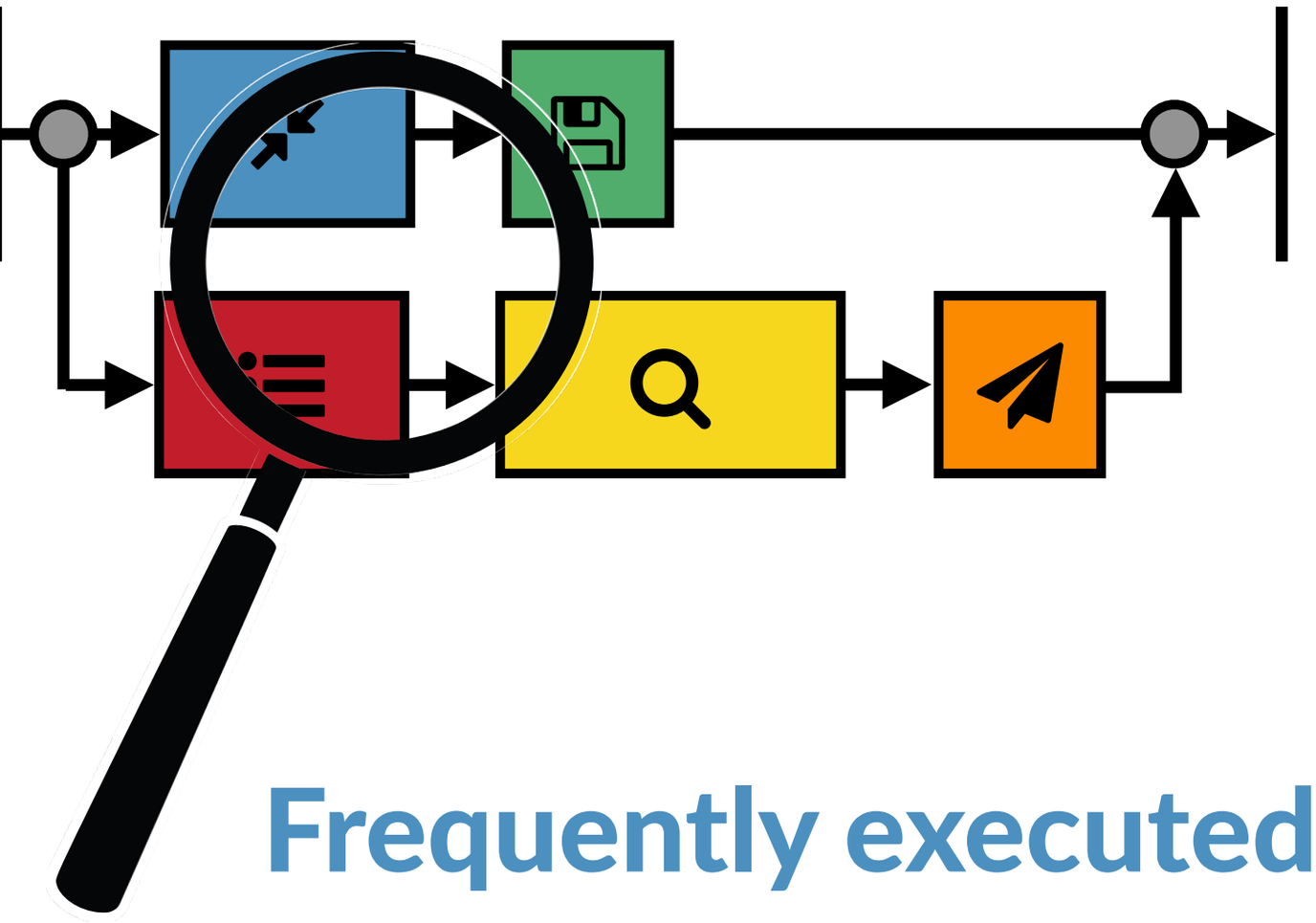
```
%       cumulative
time    seconds     calls     name
20.05    8.02       1
 9.56    3.82       1
19.95    7.98       1
45.19   11.31       1
 5.25    2.10       1
```

Software Profilers

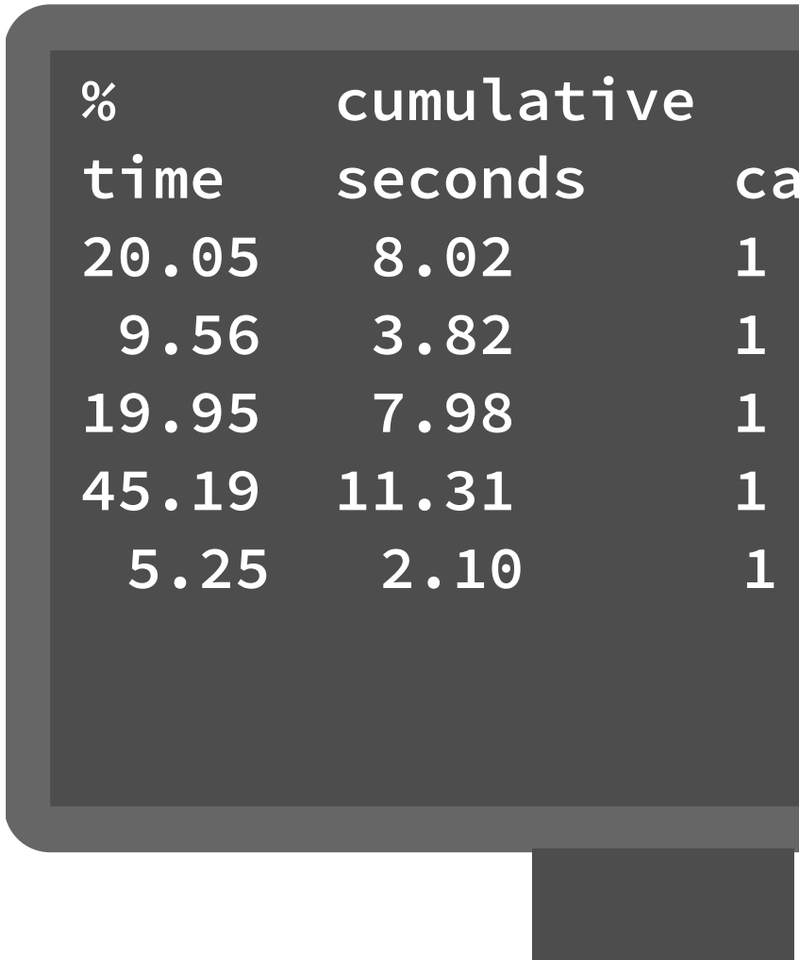


```
%       cumulative
time    seconds     ca
20.05    8.02       1
 9.56    3.82       1
19.95    7.98       1
45.19   11.31       1
 5.25    2.10       1
```

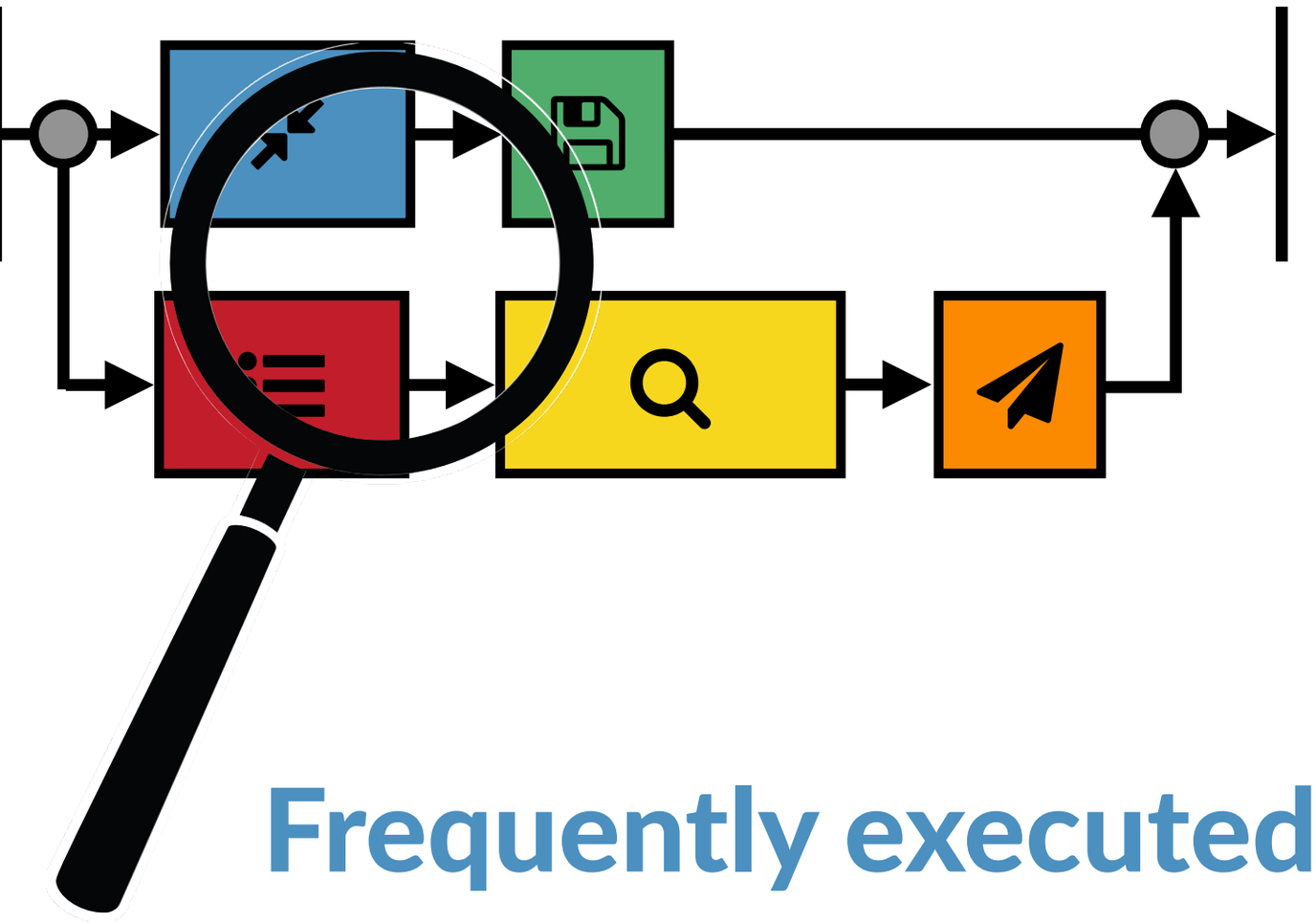
Software Profilers



Frequently executed code

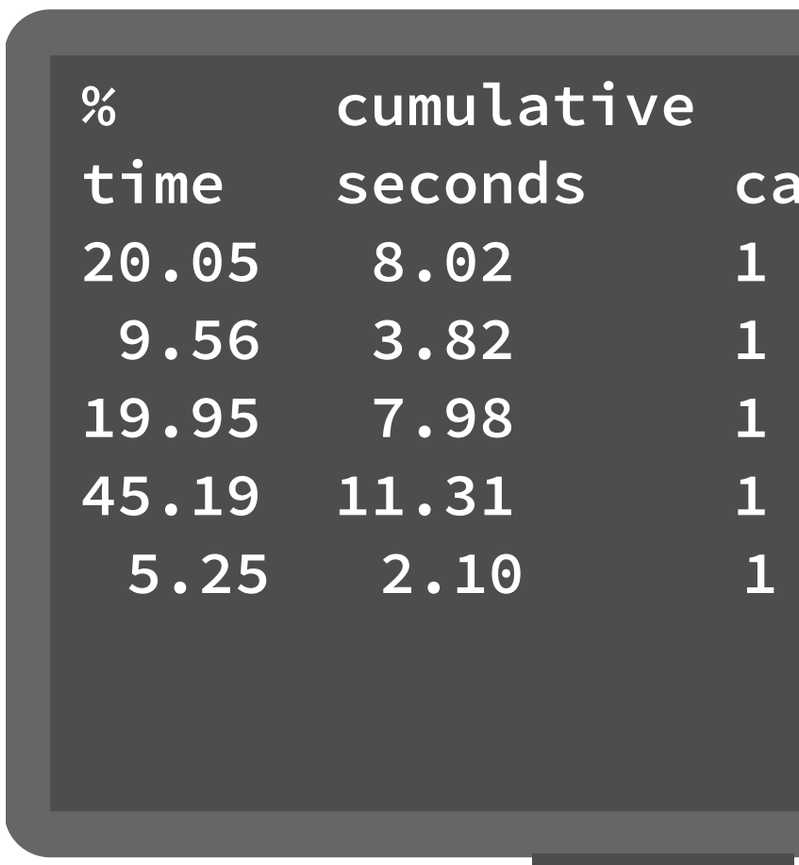


Software Profilers



Frequently executed code

Code that runs for a long time

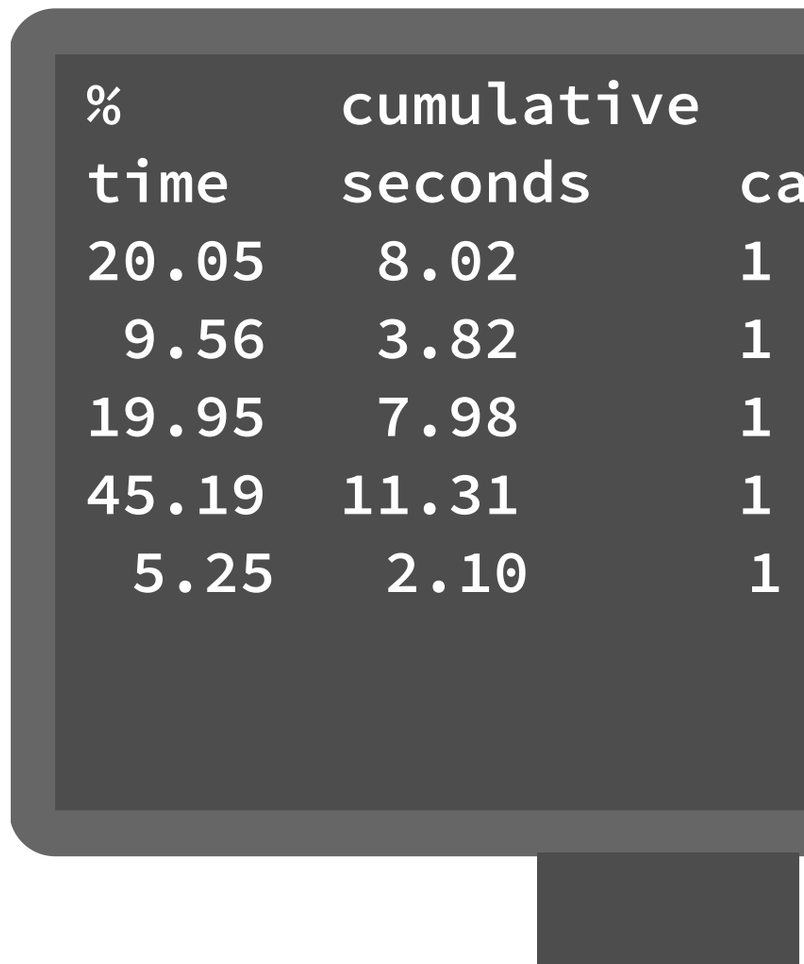


Software Profilers

Are these places where Bob should focus on performance?

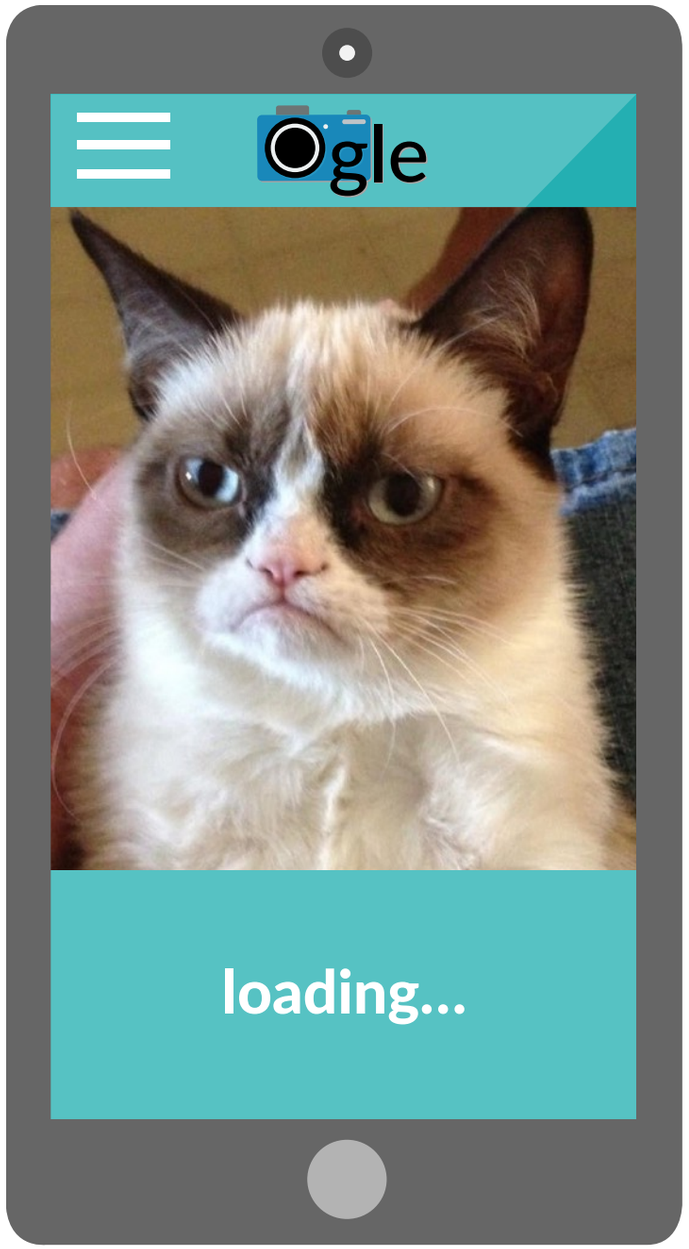
Frequently executed code

Code that runs for a long time

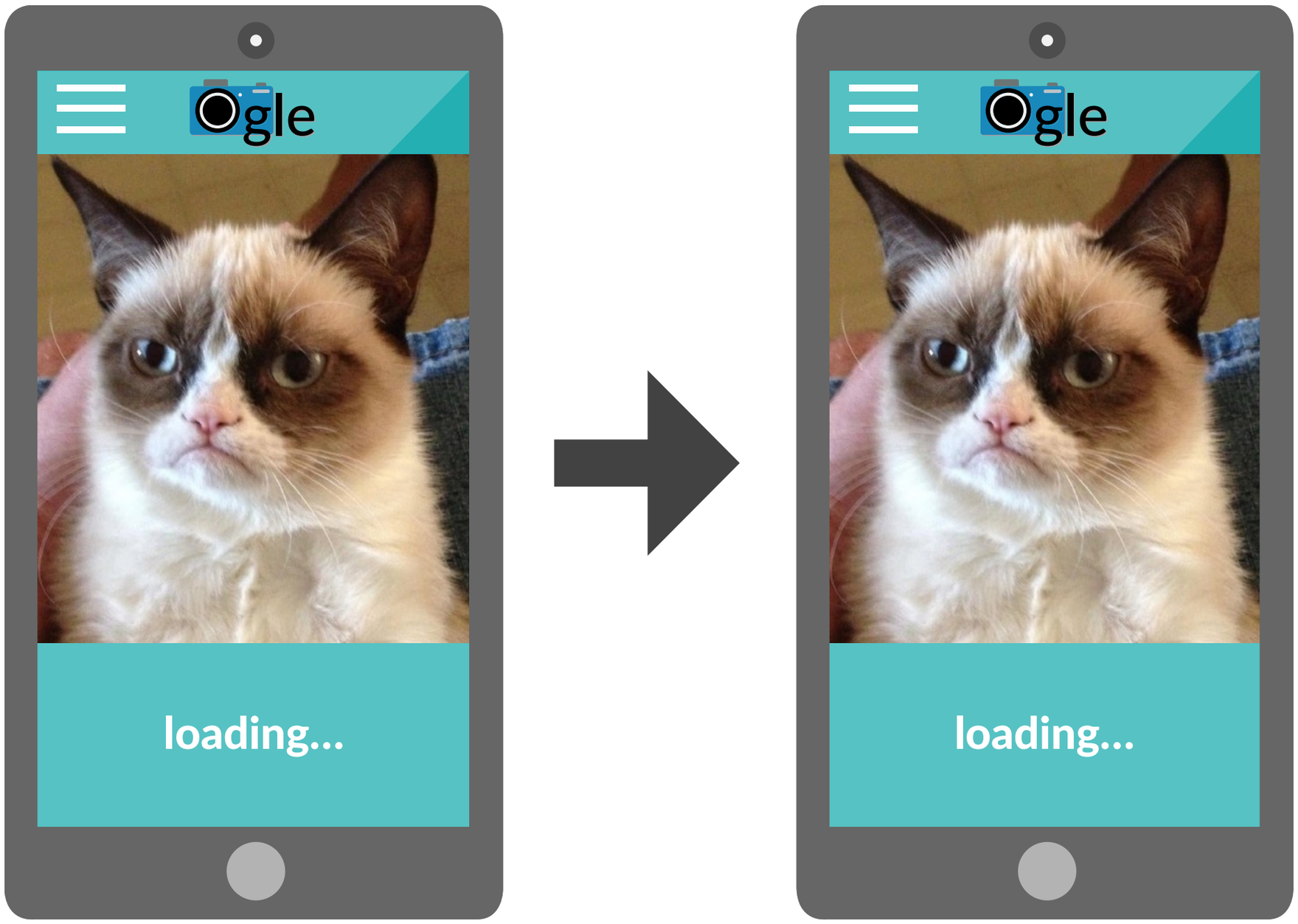


Would this speed up Ogle?

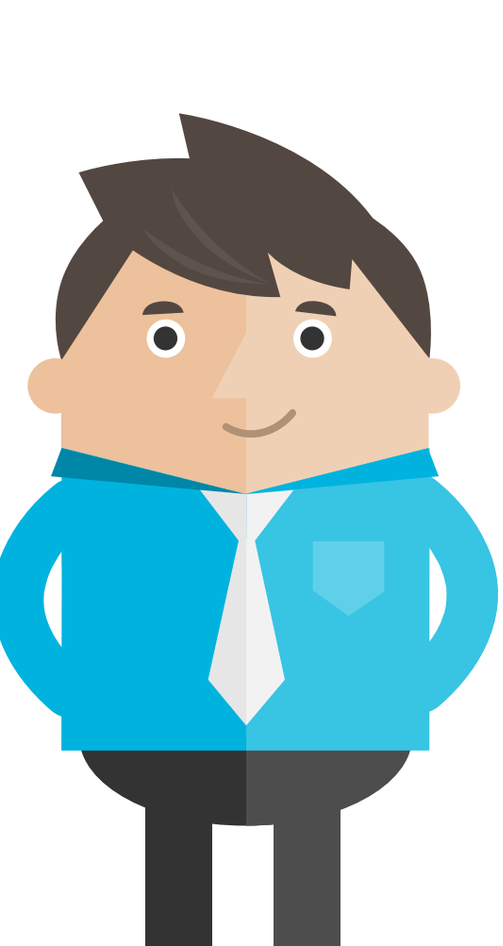
Would this speed up Ogle?

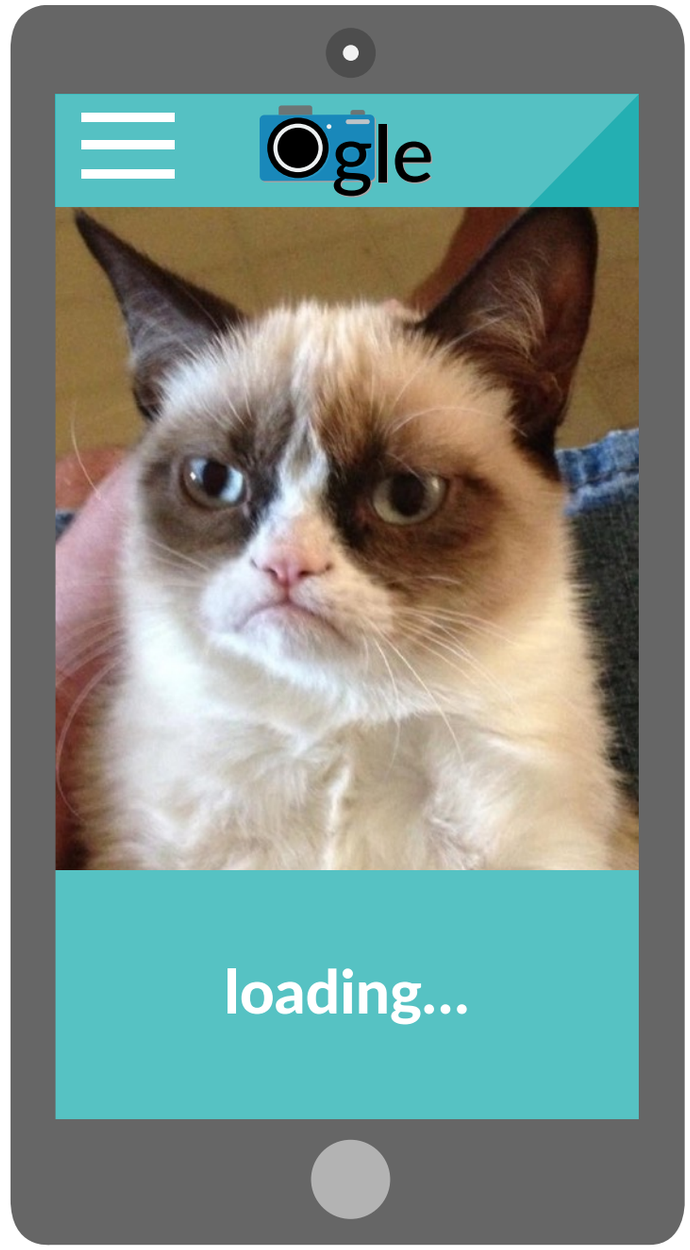


Would this speed up Ogle?



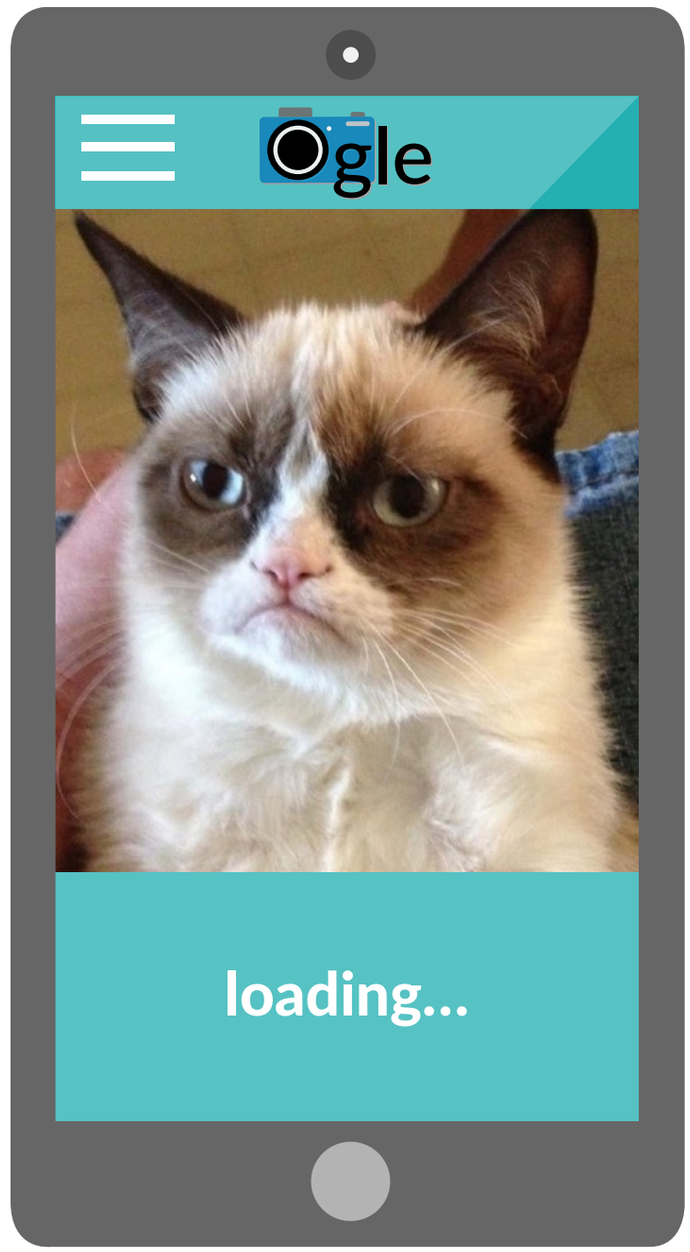
Would this speed up Ogle?





Would this speed up Ogle?

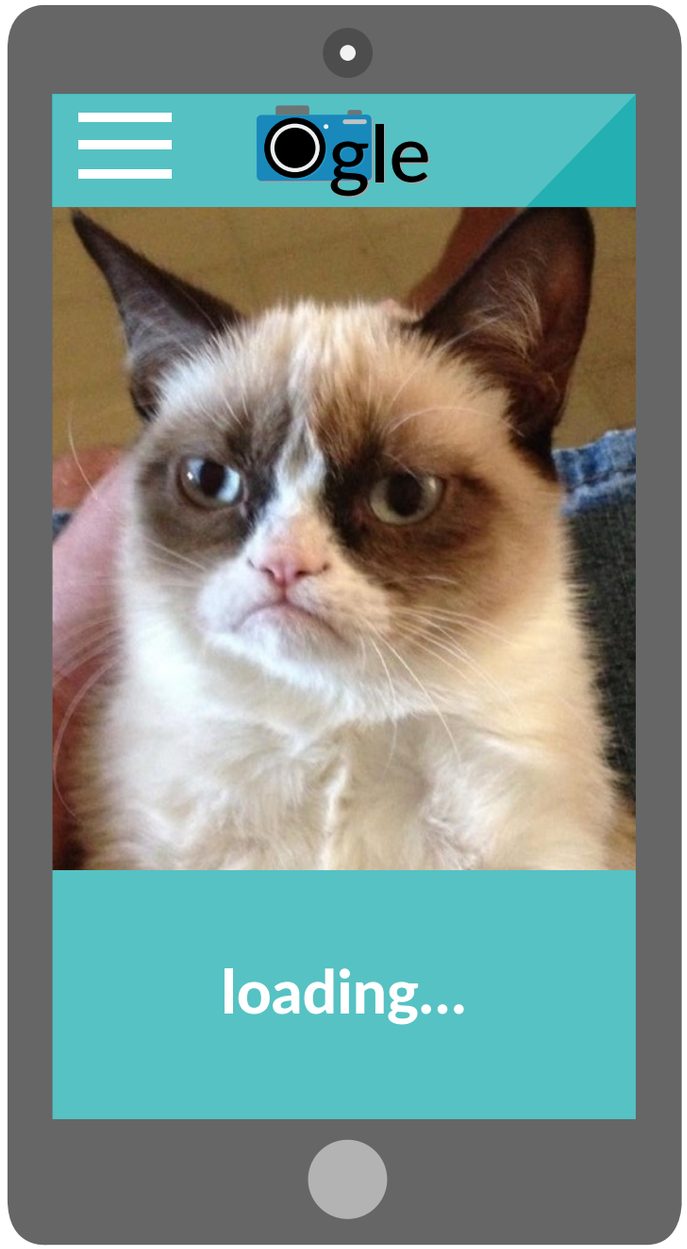
Frequently executed code



Would this speed up Ogle?

Frequently executed code

Code that runs for a long time

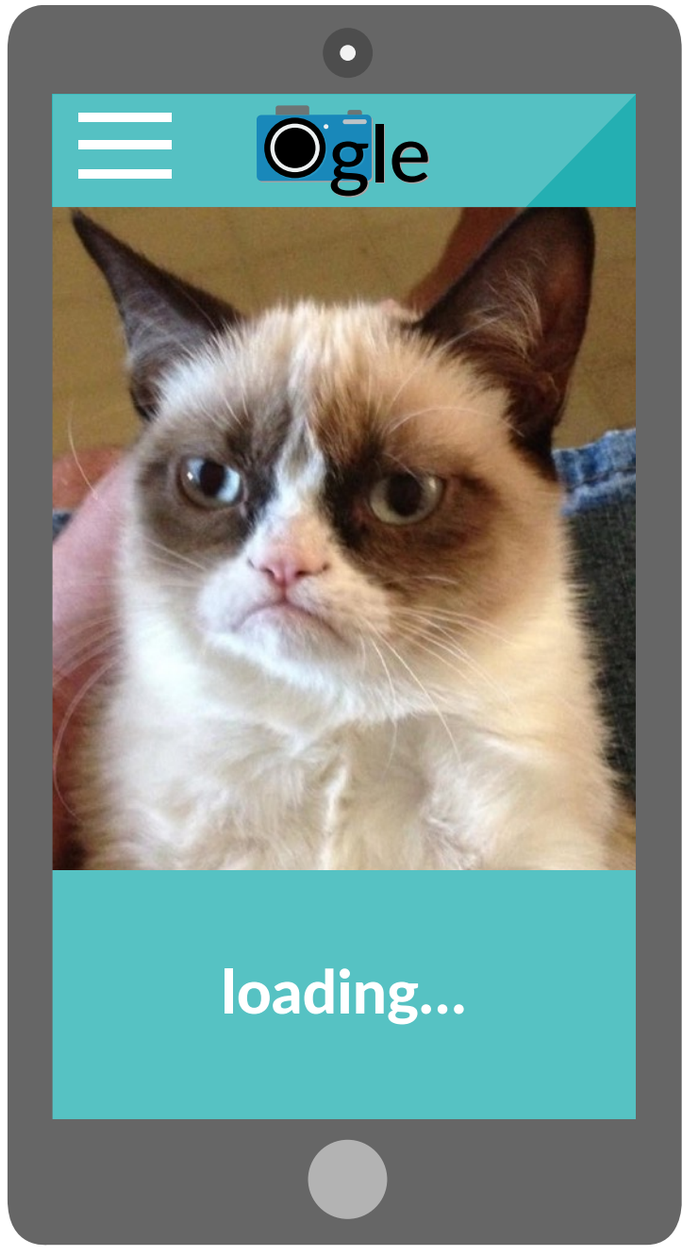


Would this speed up Ogle?

Frequently executed code

Code that runs for a long time

Profilers do a bad job finding important code in parallel programs.

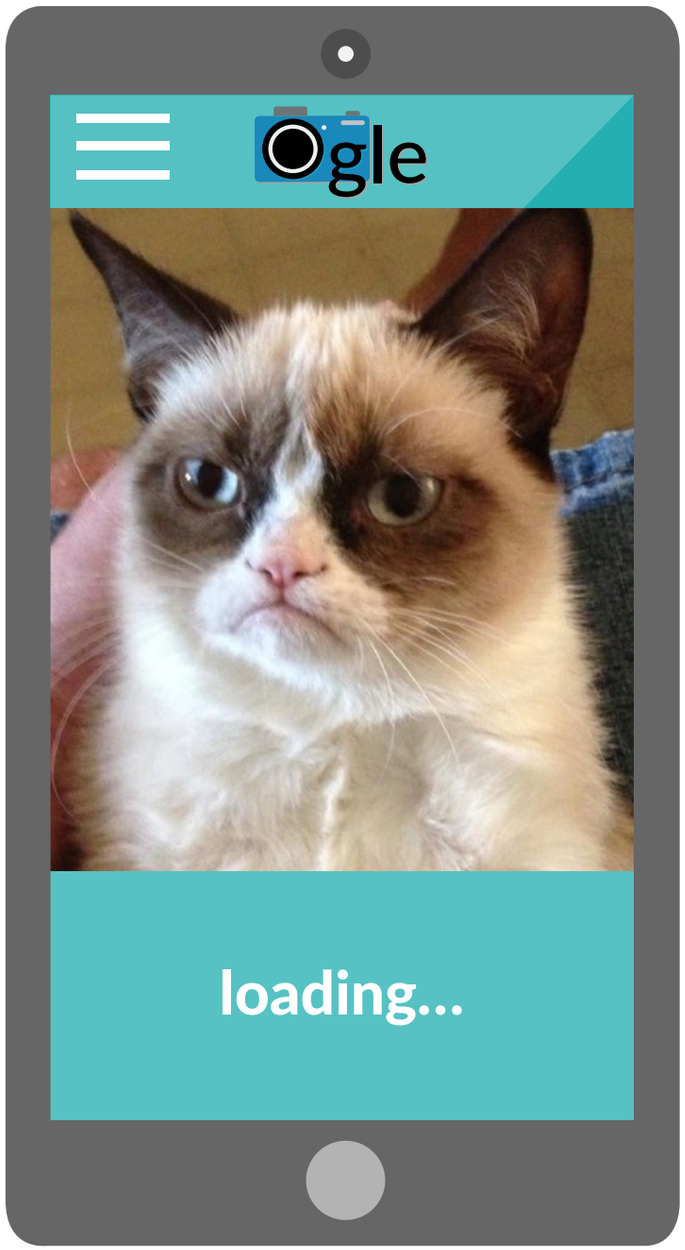


Would this speed up Ogle?

Frequently executed code

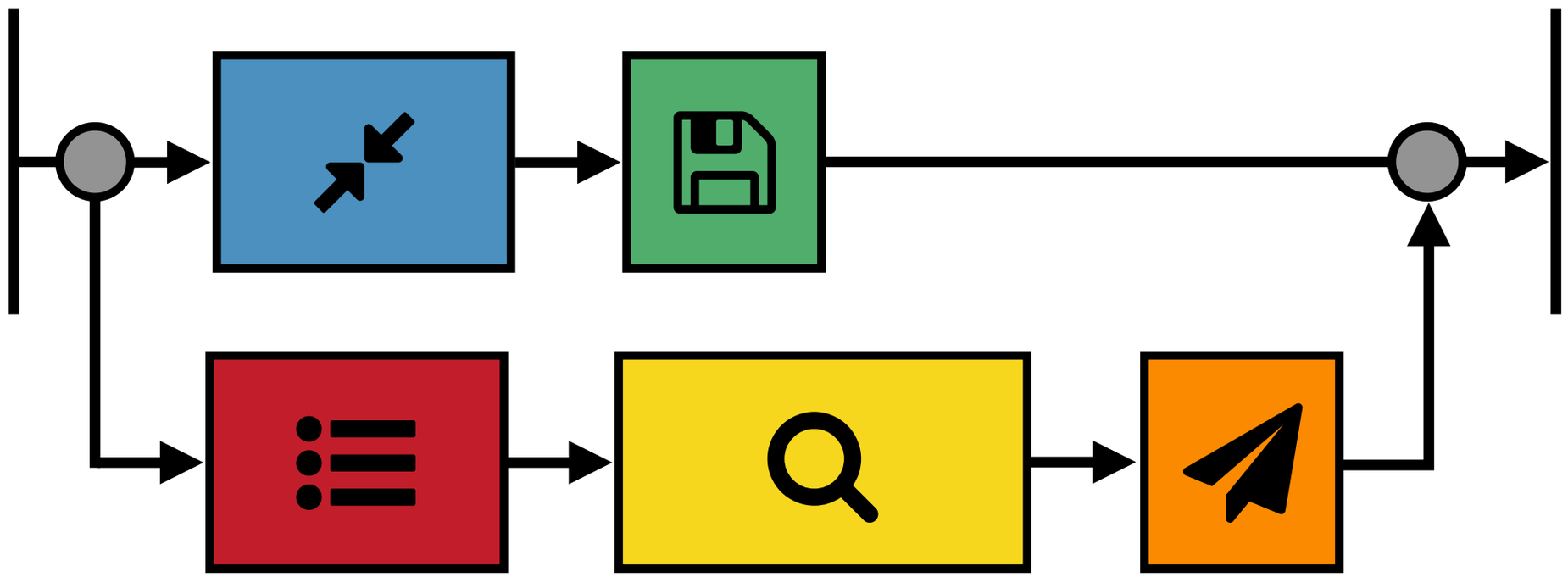
Code that runs for a long time

Profilers do a bad job finding important code in parallel programs.

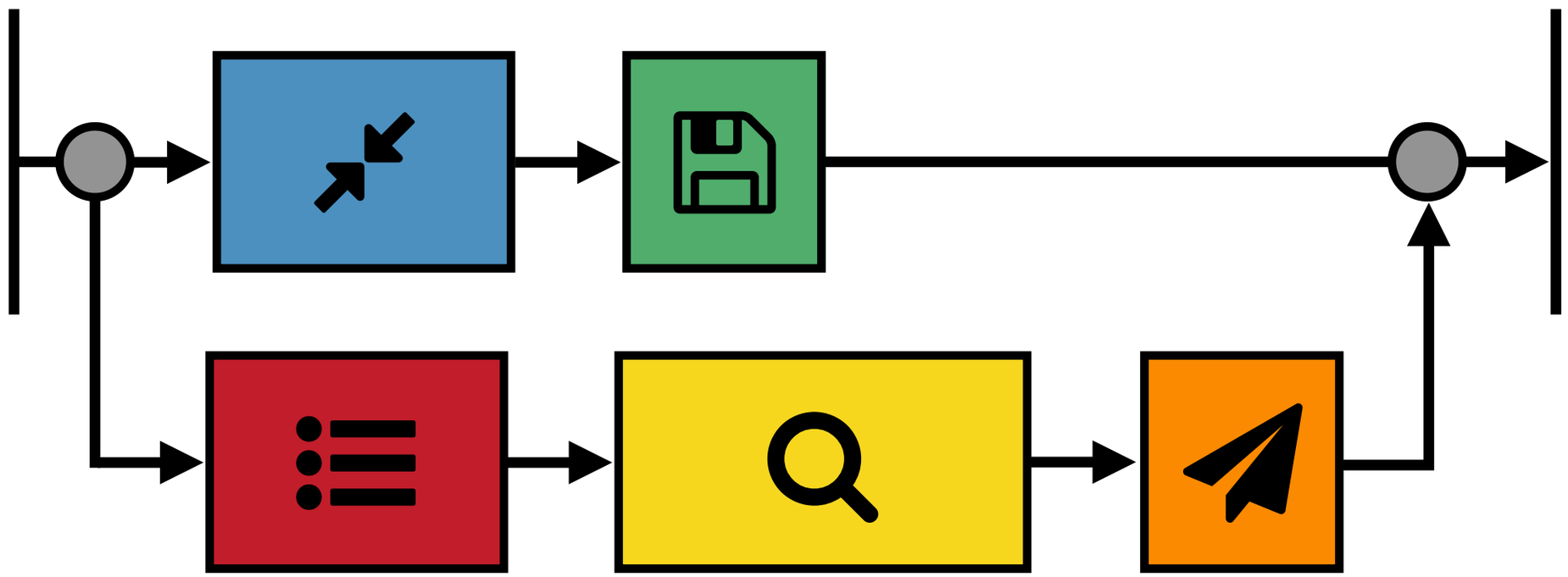


We need better tools.

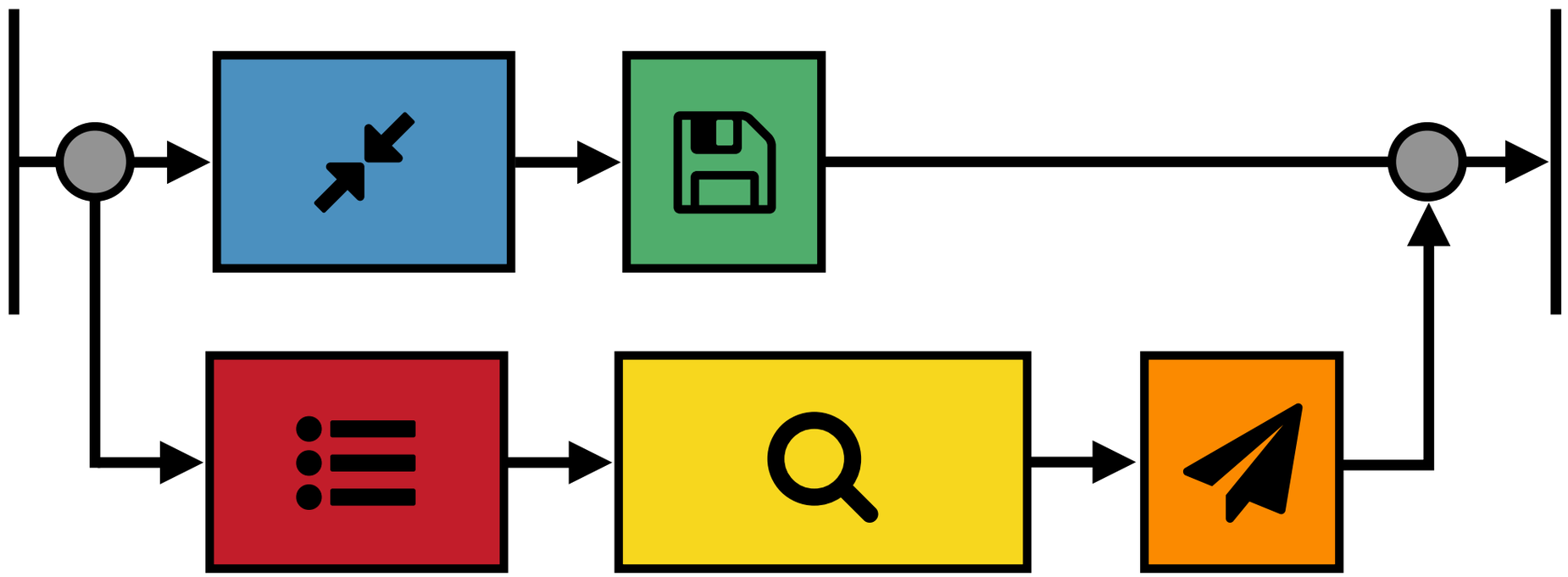
What *would* speed up Ogle?



What *would* speed up Ogle?

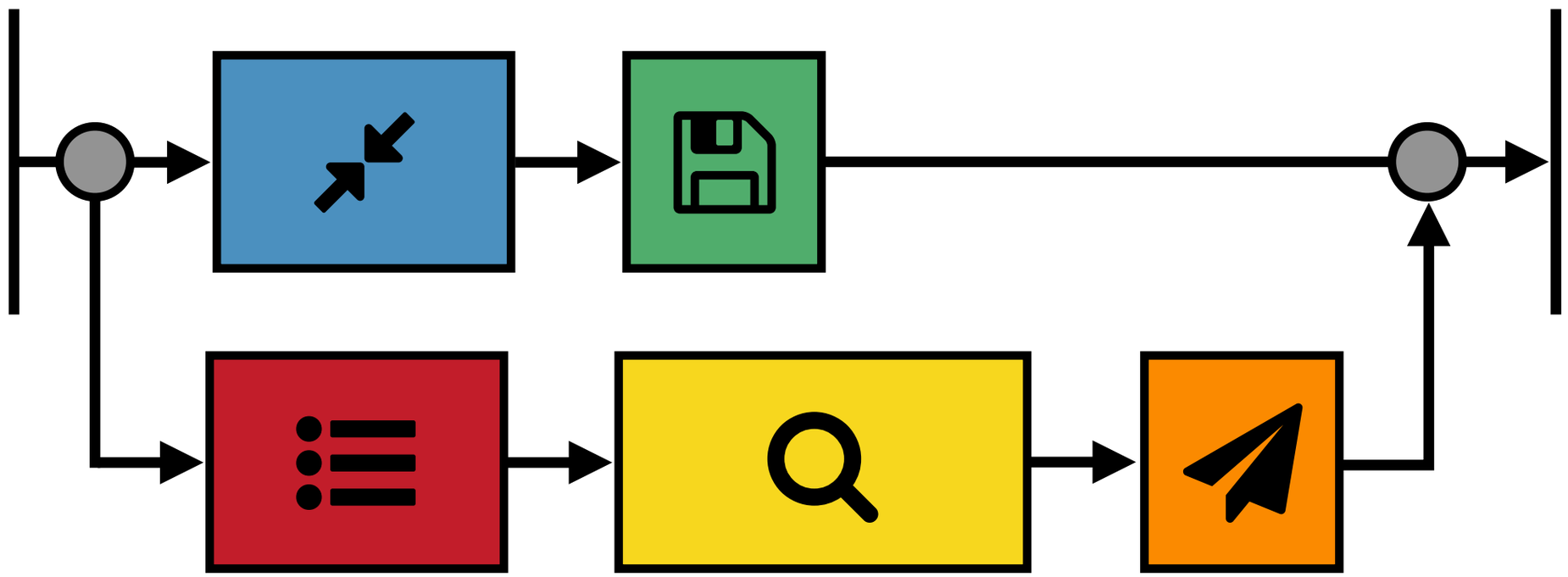


What *would* speed up Ogle?



Hey, look over here

What *would* speed up Ogle?



Hey, look over here

What would this information look like?

Causal Profile

Causal Profile

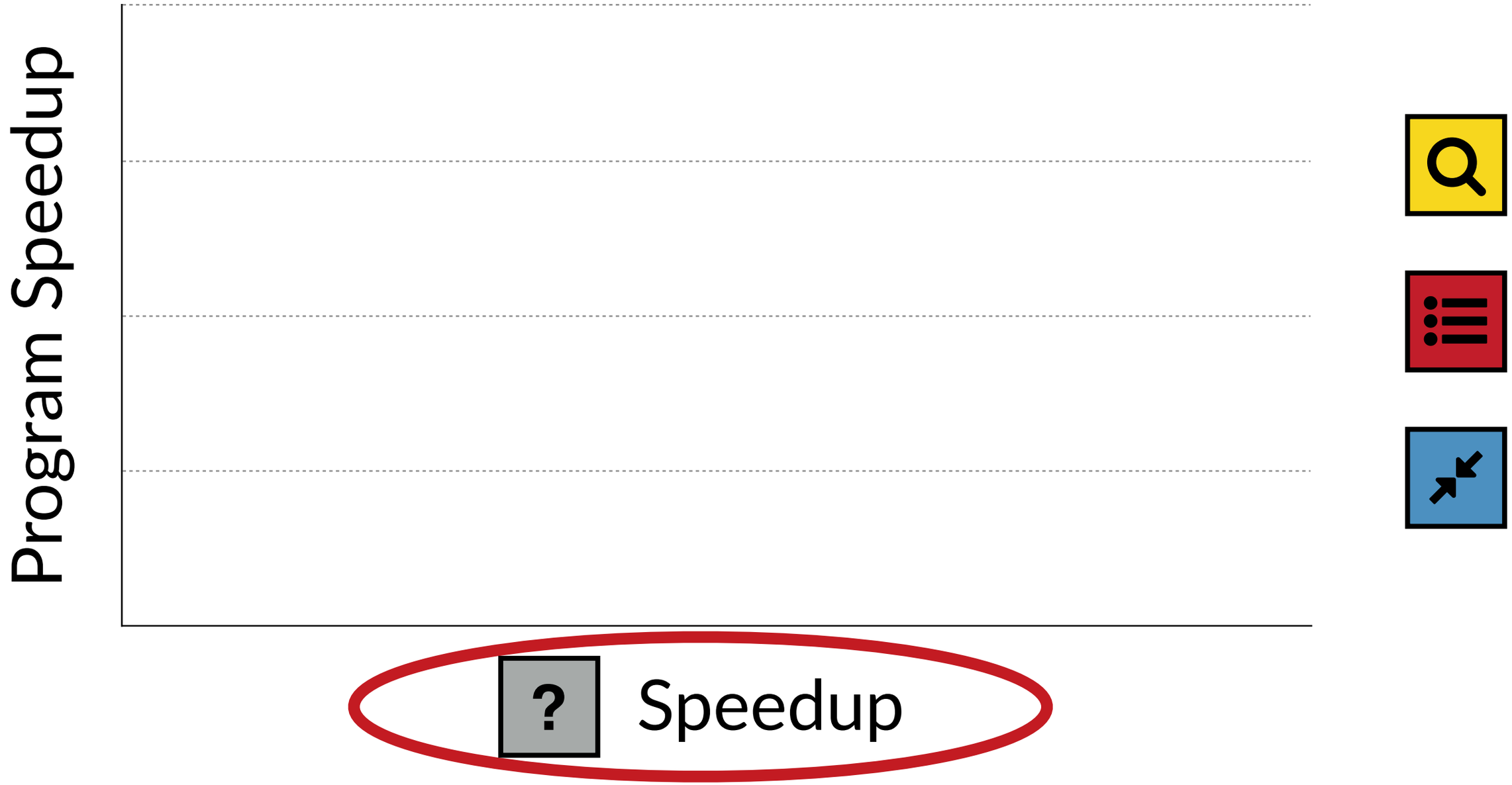
Tells you where optimizations
will make a difference

Causal Profile

Tells you where optimizations
will make a difference

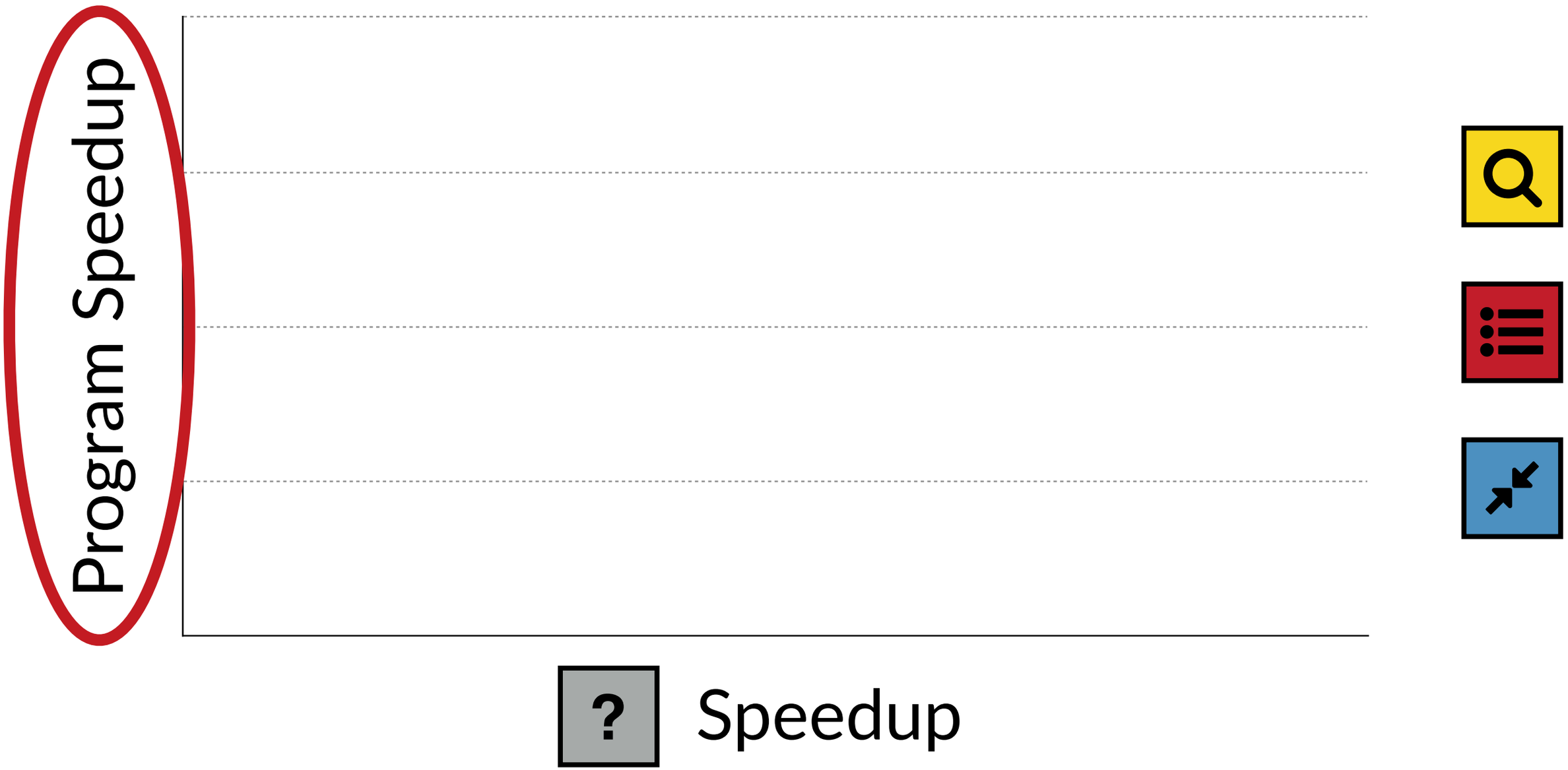
Causal Profile

Tells you where optimizations will make a difference



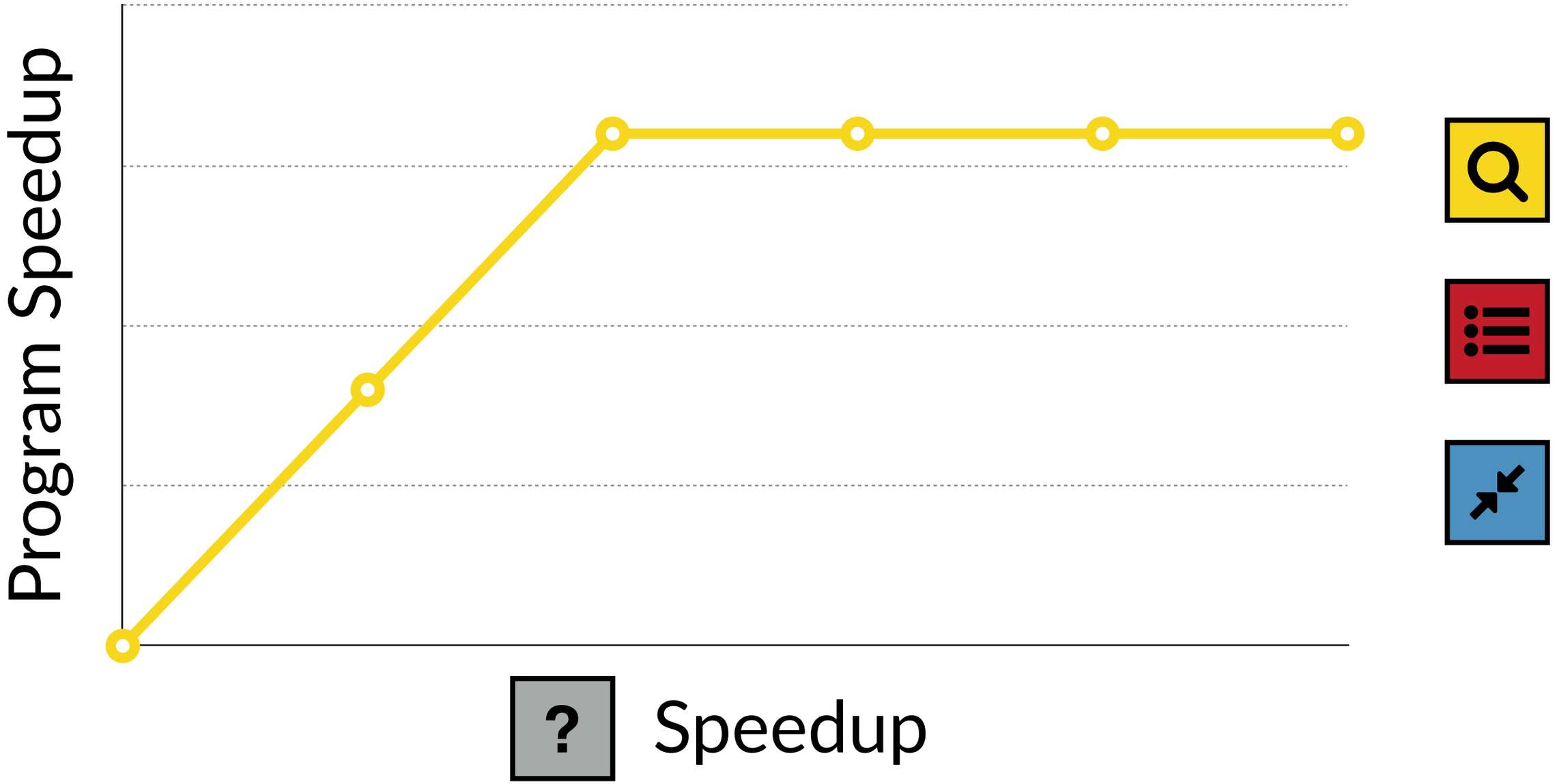
Causal Profile

Tells you where optimizations will make a difference



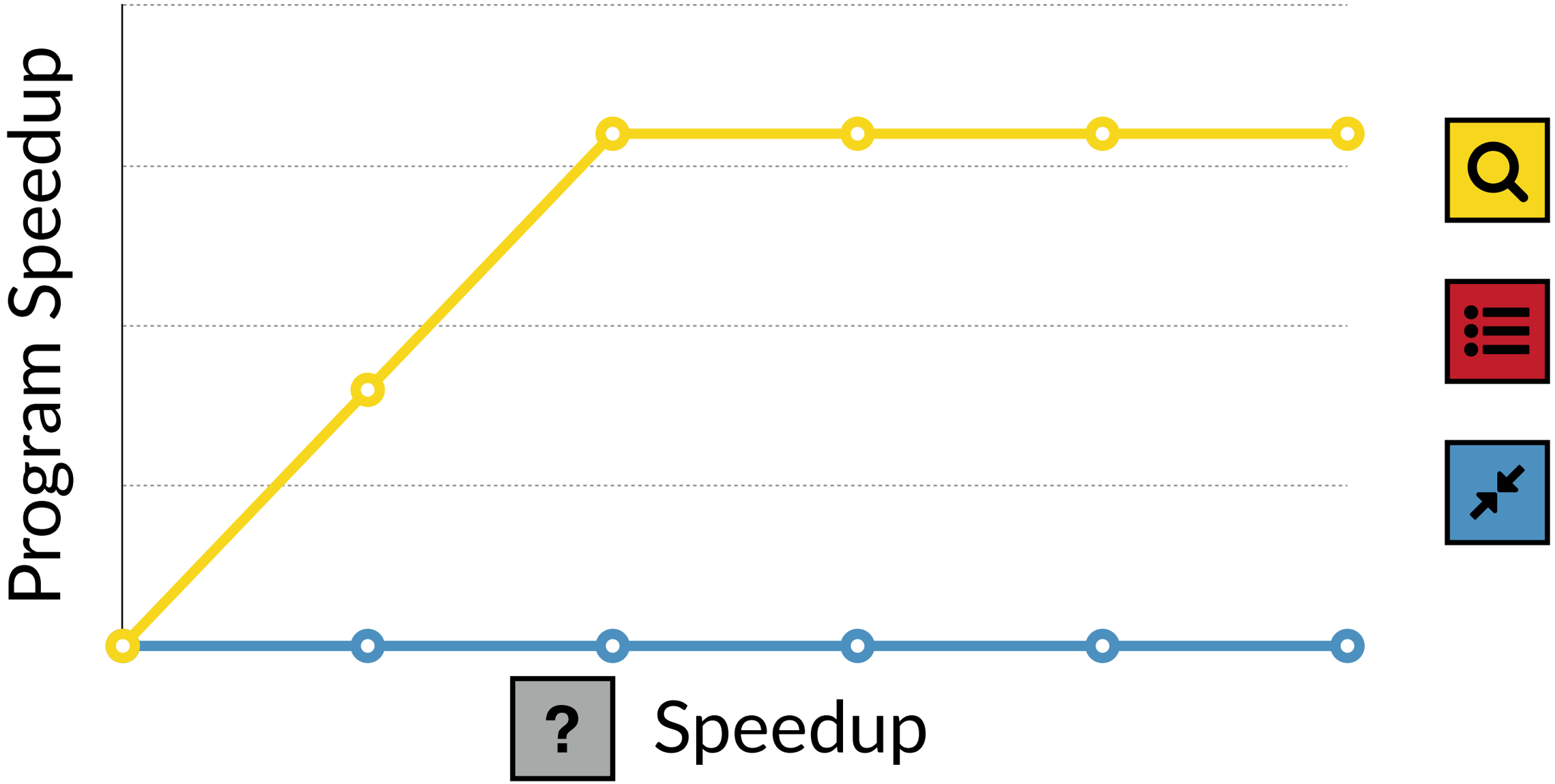
Causal Profile

Tells you where optimizations will make a difference



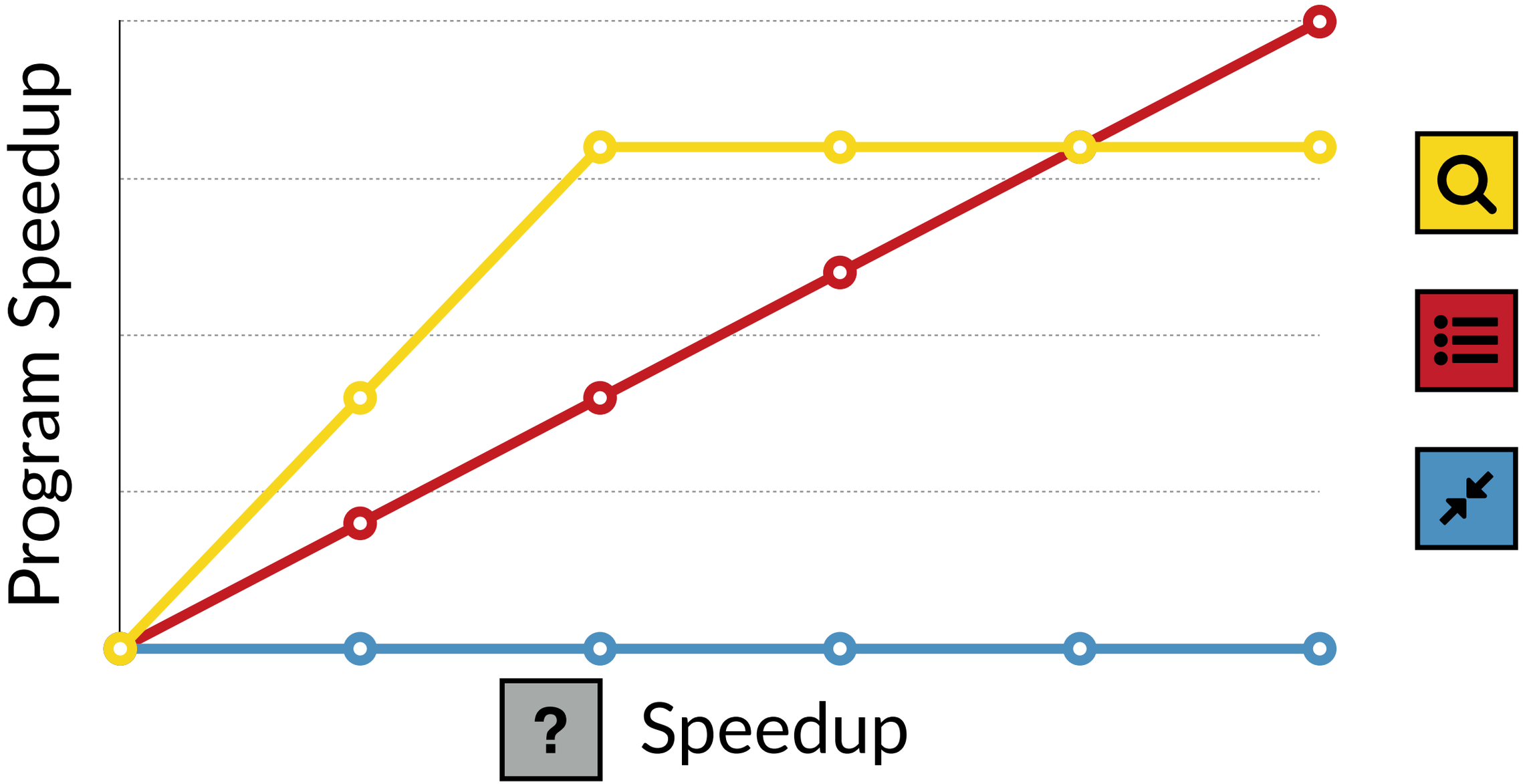
Causal Profile

Tells you where optimizations will make a difference

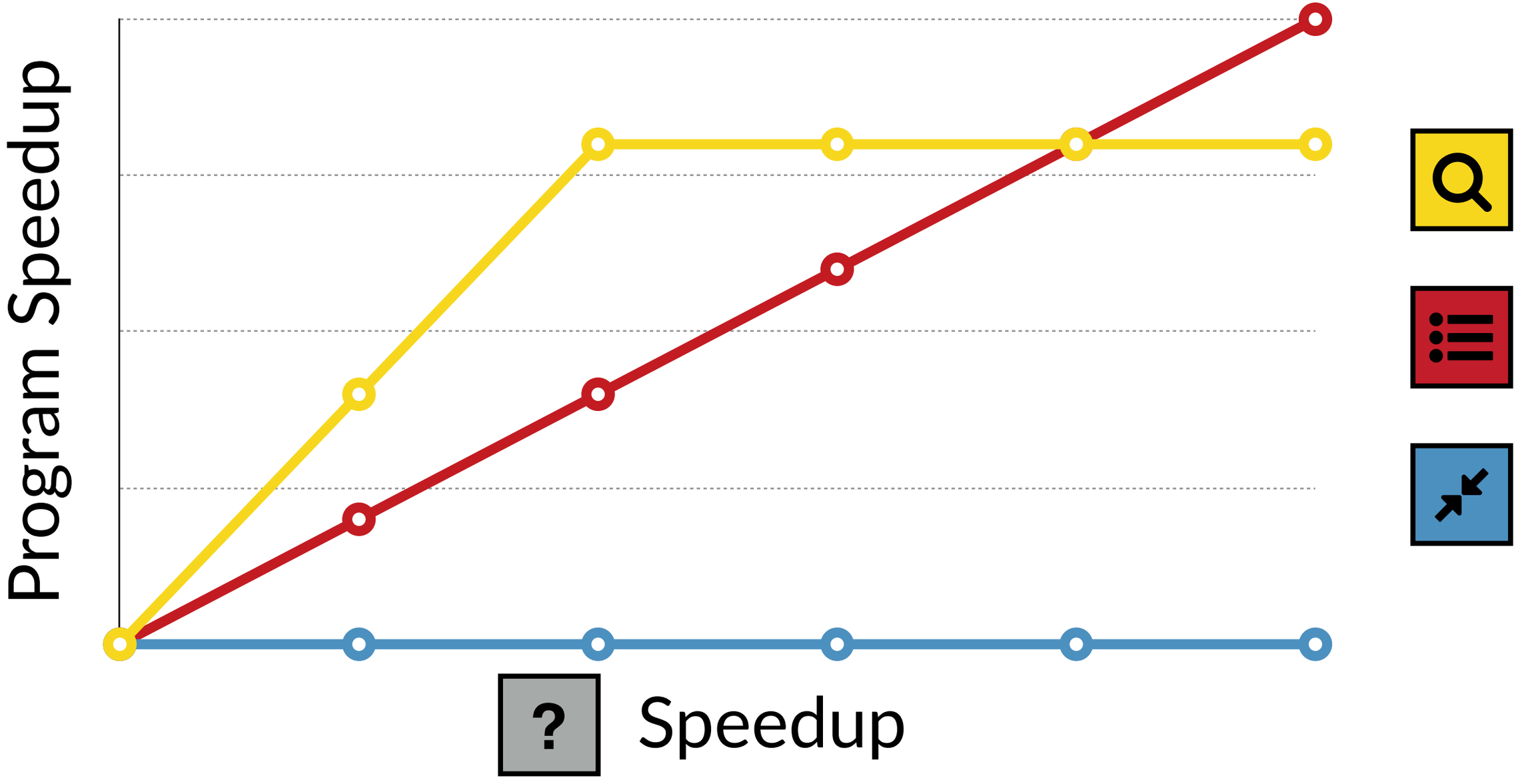


Causal Profile

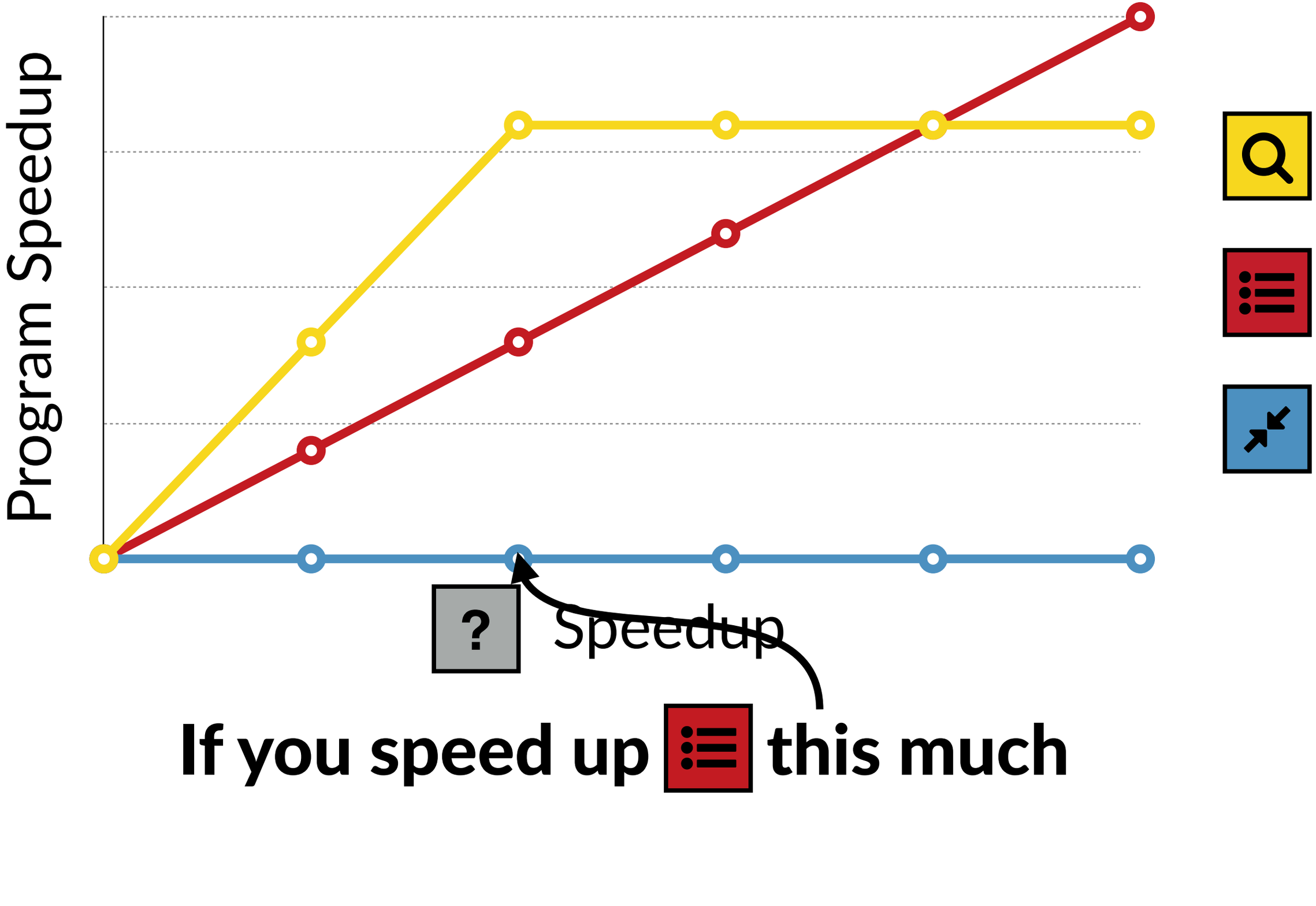
Tells you where optimizations will make a difference



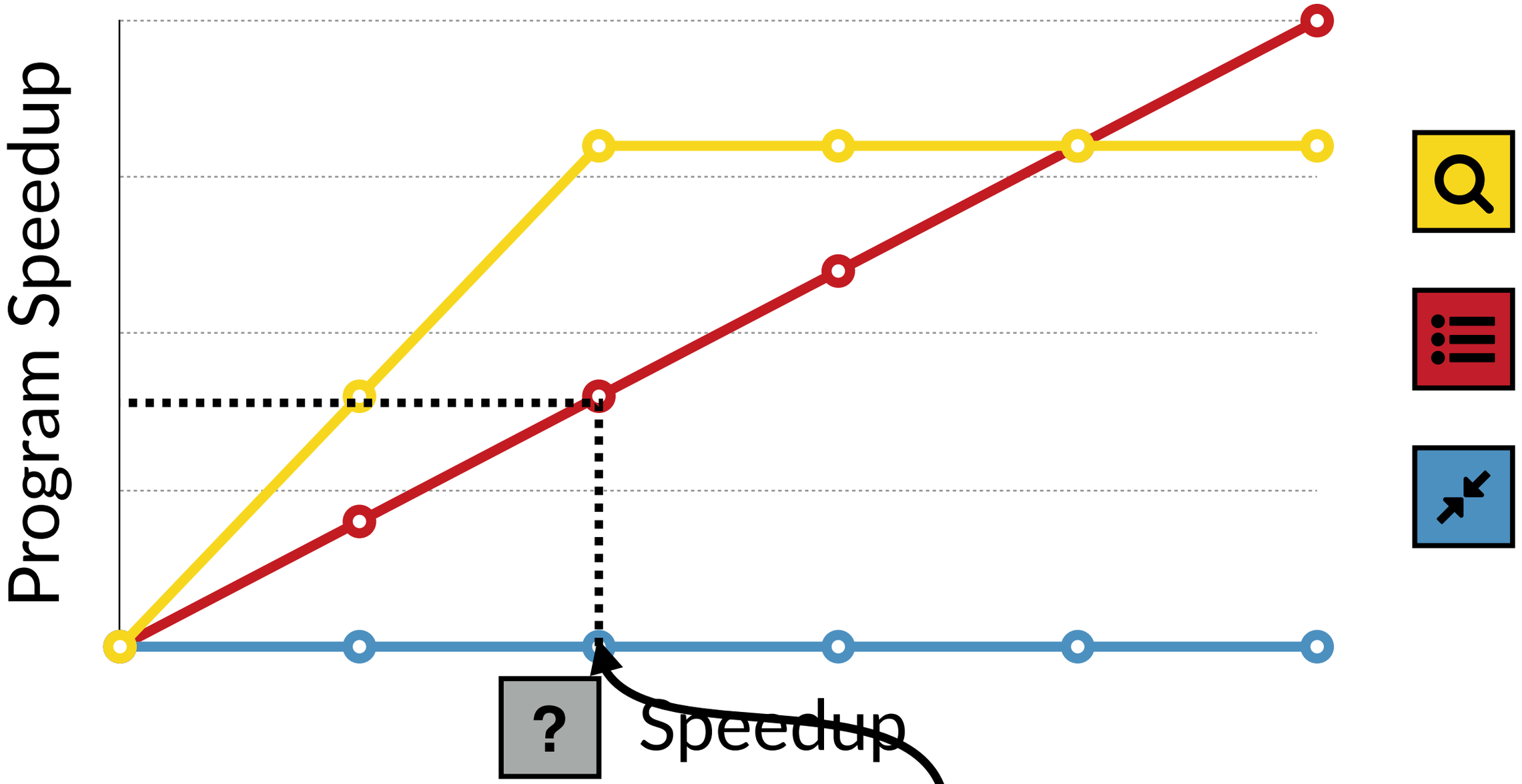
Causal Profile



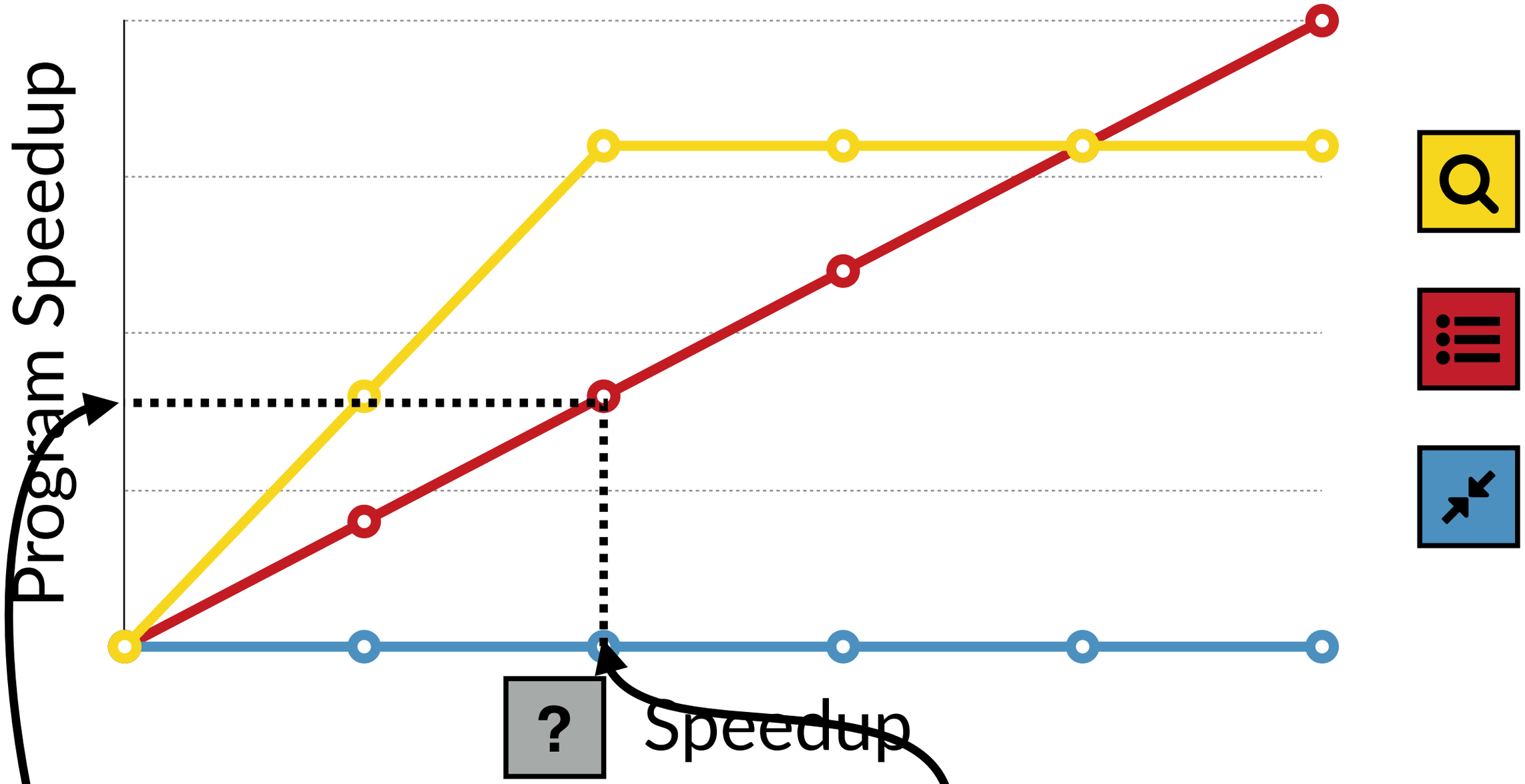
Causal Profile



Causal Profile

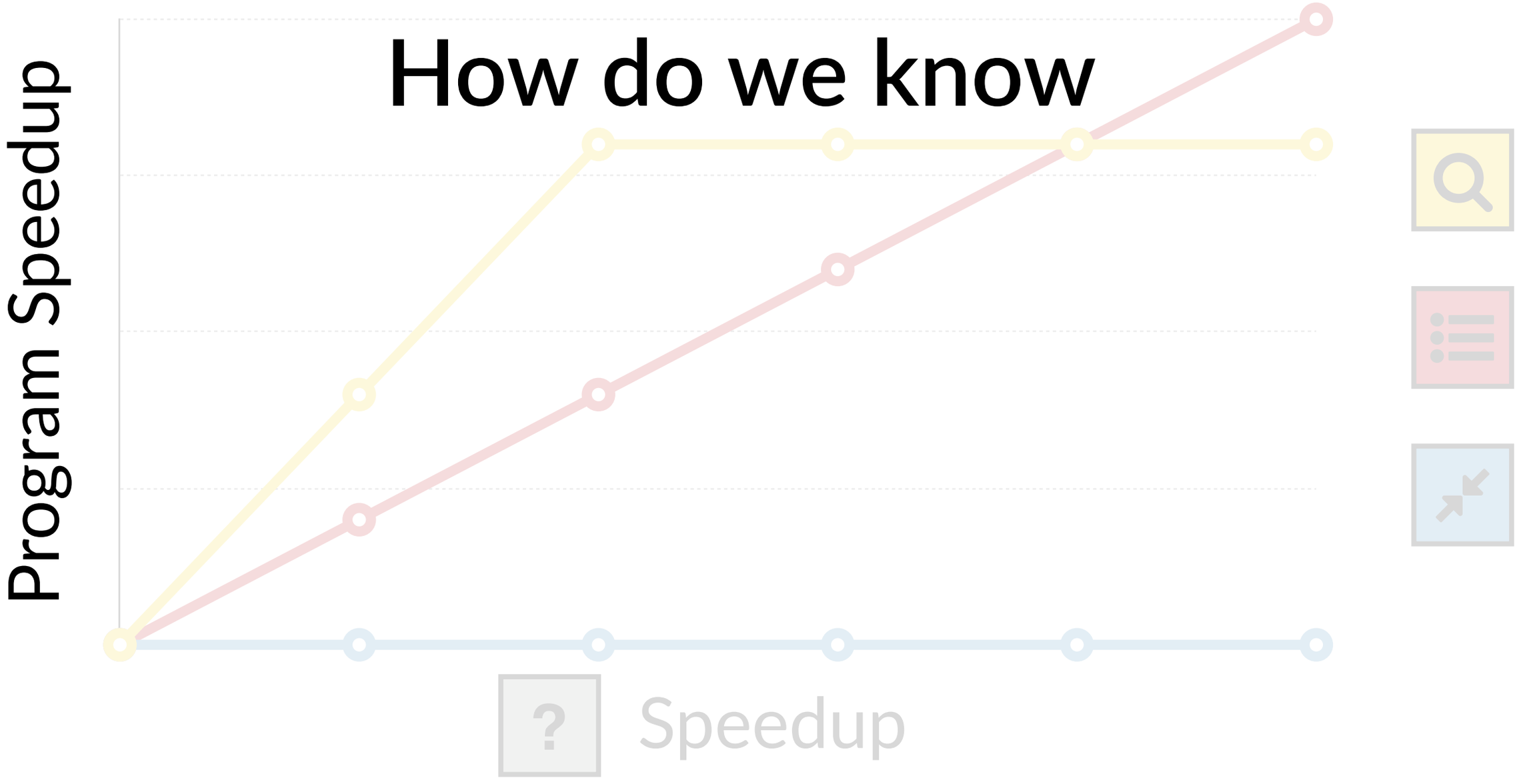


Causal Profile



Causal Profile

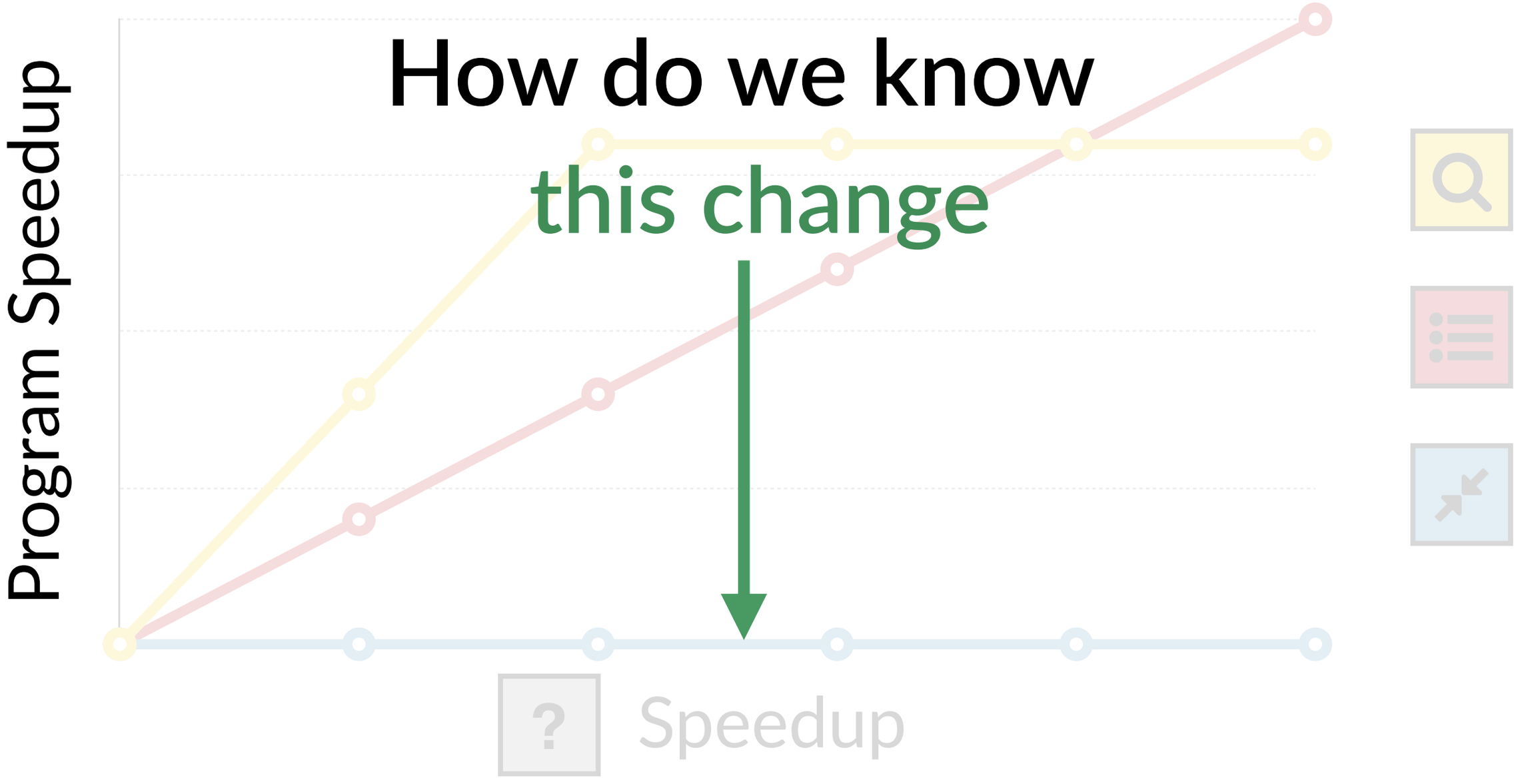
How do we know



Causal Profile

How do we know

this change

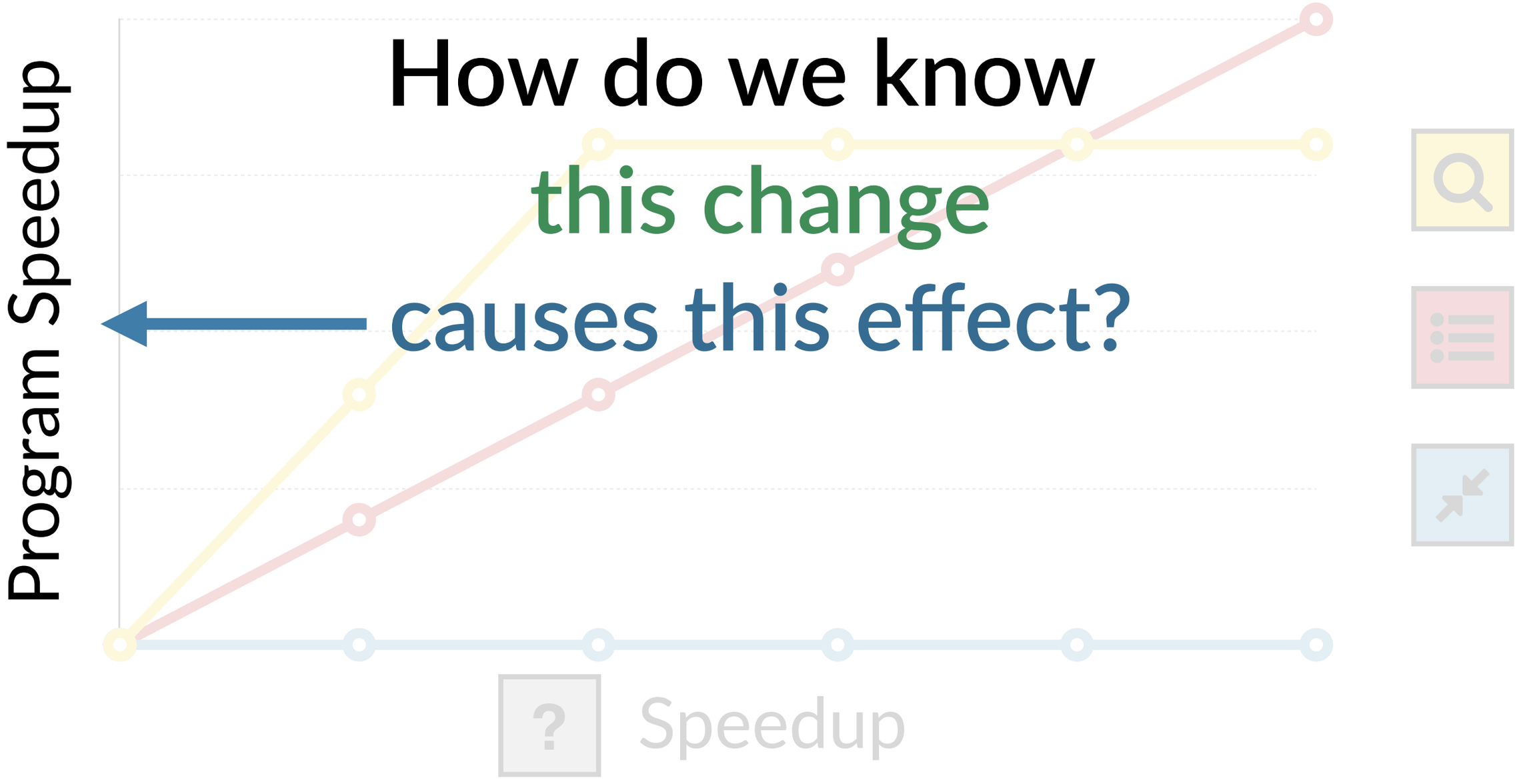


Causal Profile

How do we know

this change

causes this effect?

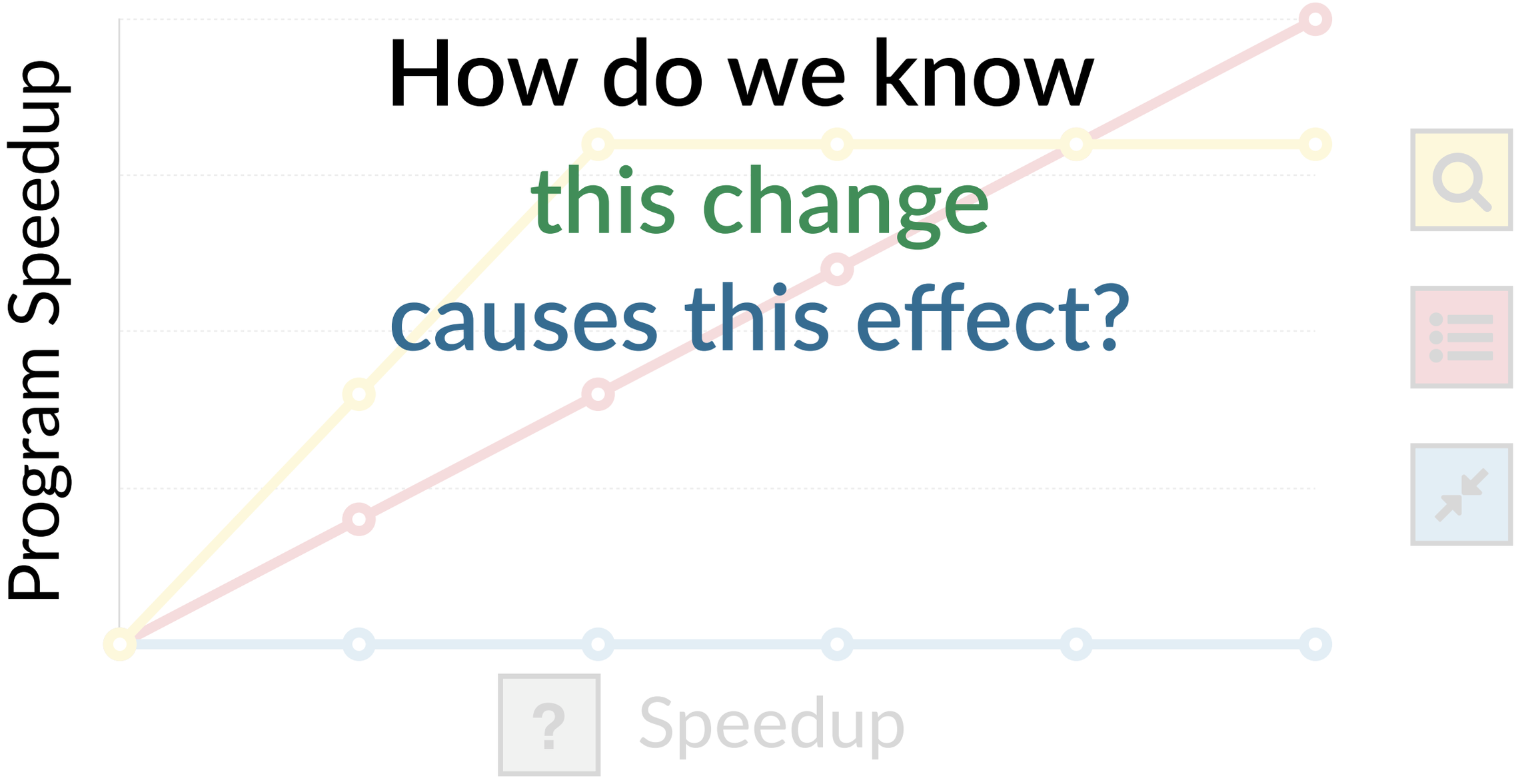


Causal Profile

How do we know

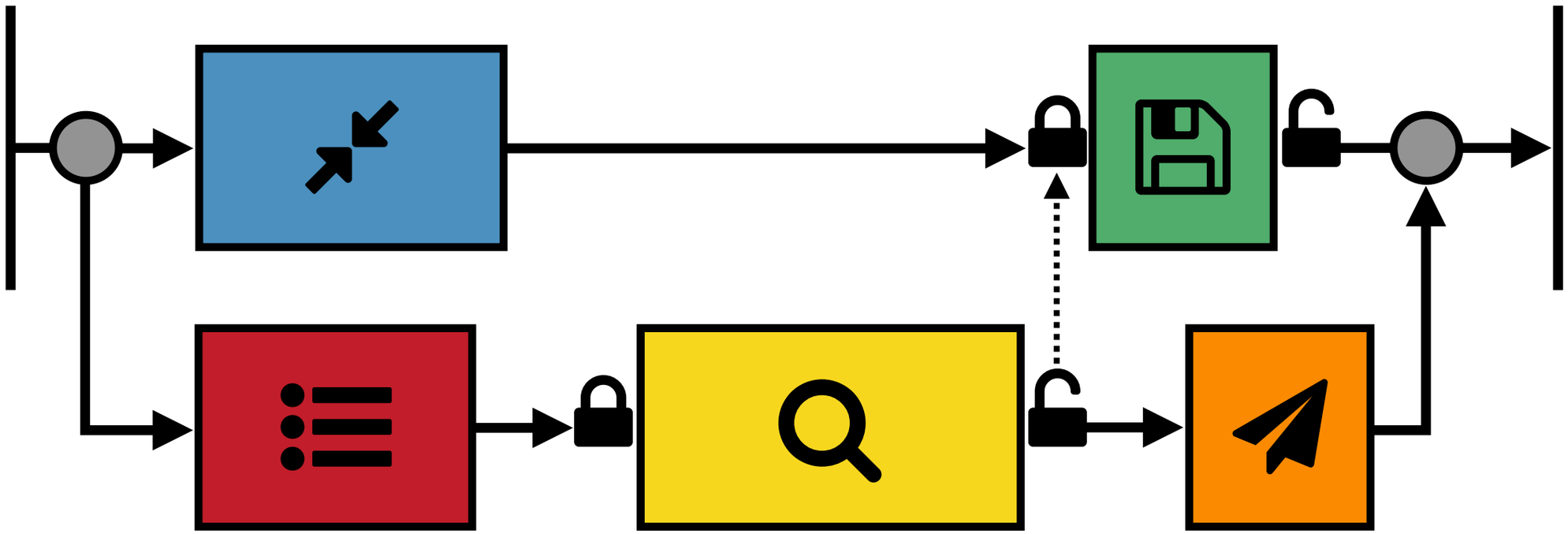
this change

causes this effect?



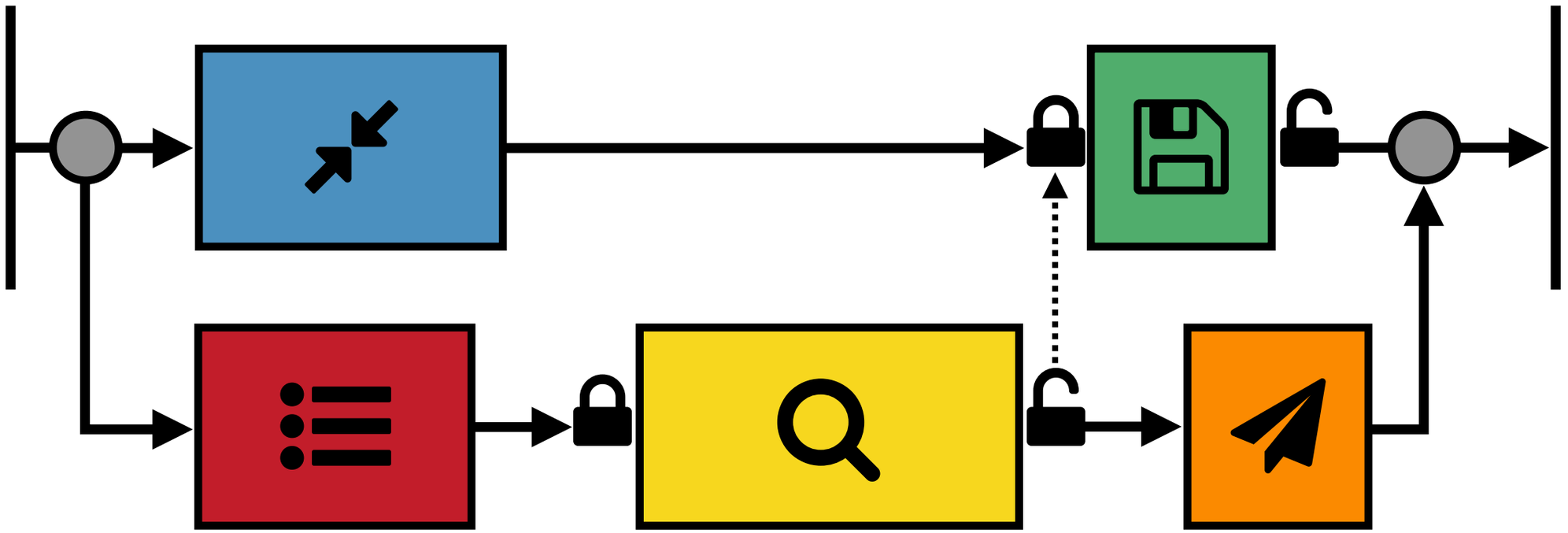
Run an experiment

Performance Experiments



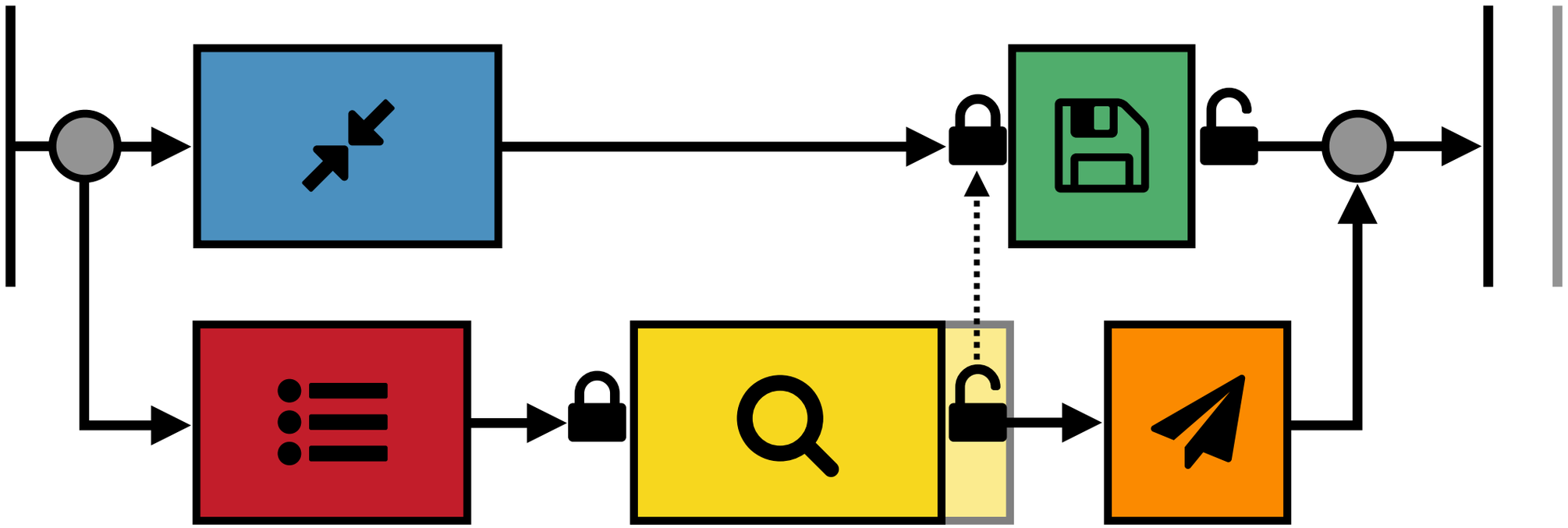
Performance Experiments

If we could magically speed up 🔍 …



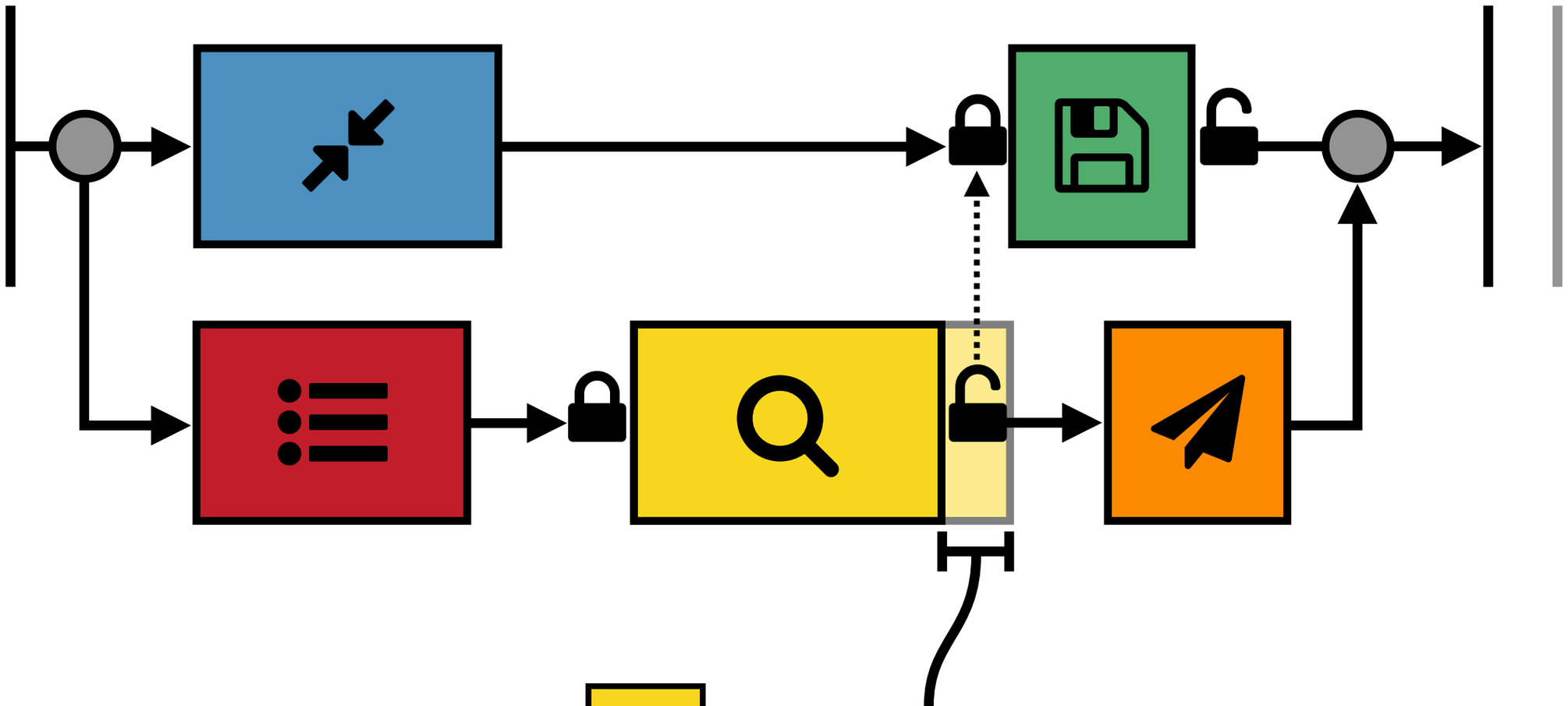
Performance Experiments

If we could magically speed up 🔍 …



Performance Experiments

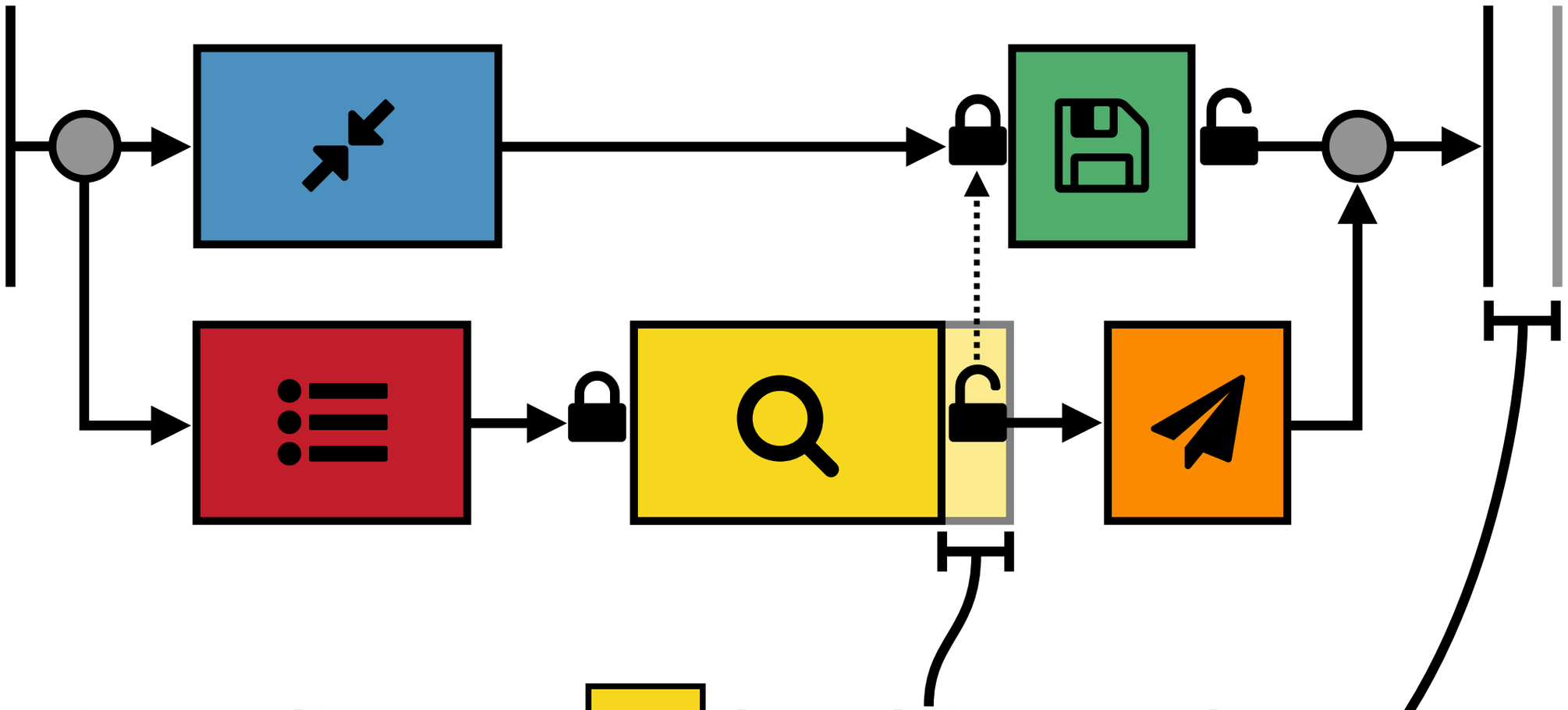
If we could magically speed up 🔍 …



Speeding up 🔍 by this much…

Performance Experiments

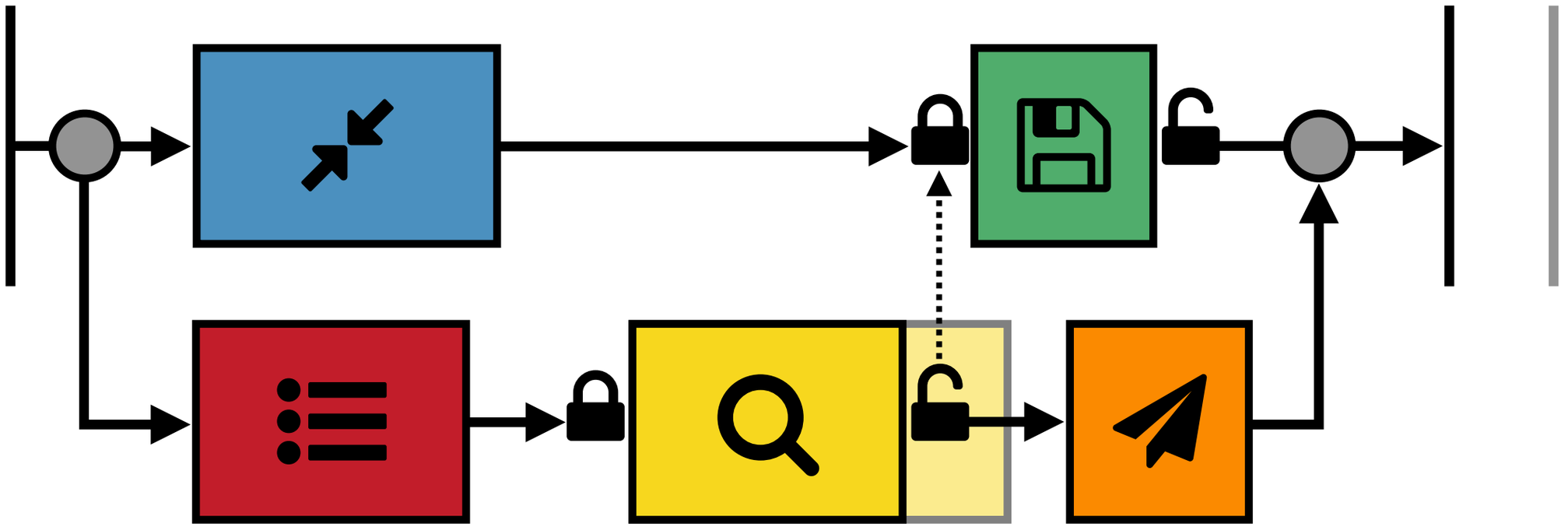
If we could magically speed up 🔍 …



Speeding up 🔍 by this much…
speeds up the program by this much.

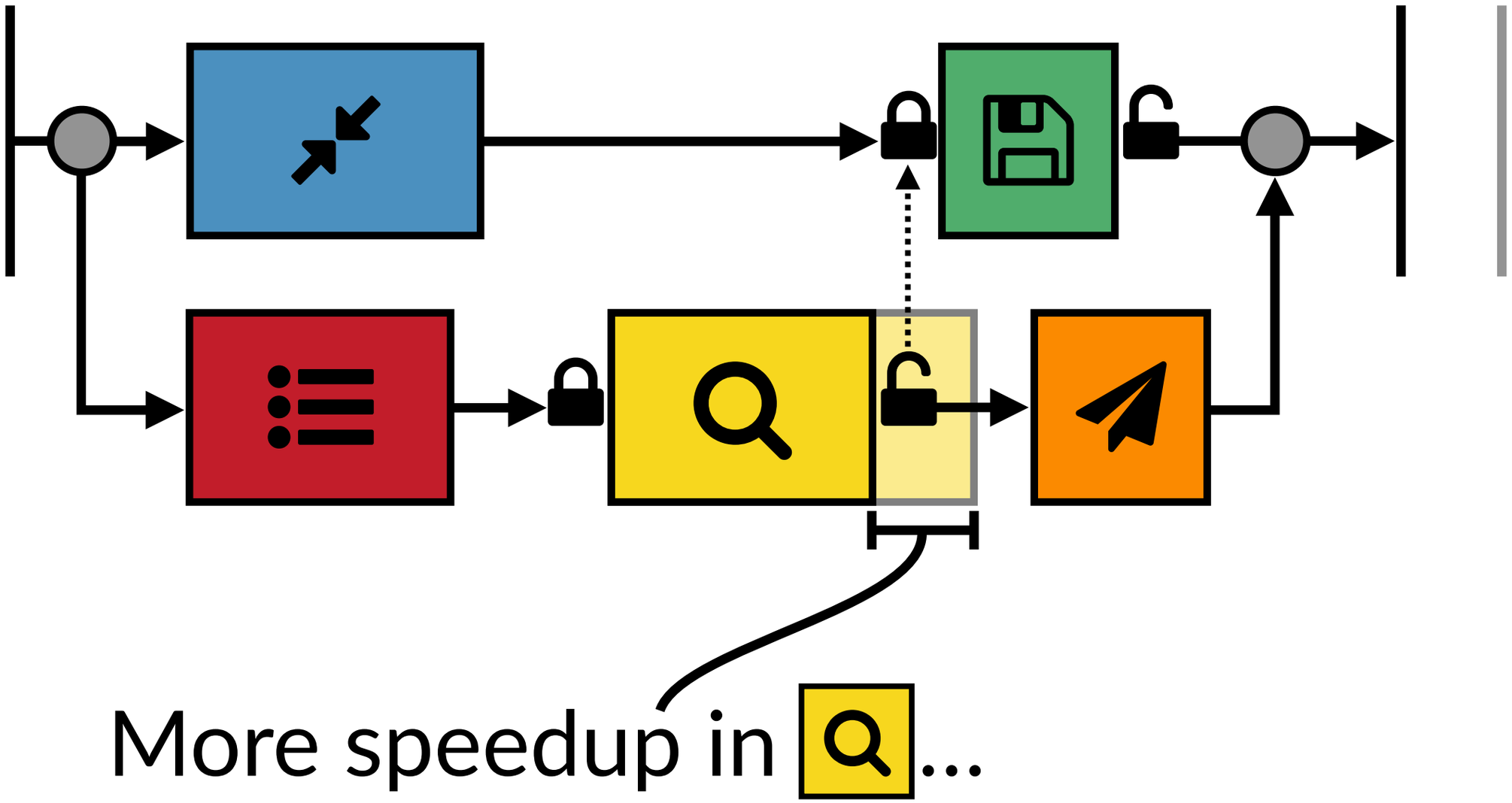
Performance Experiments

If we could magically speed up  ...



Performance Experiments

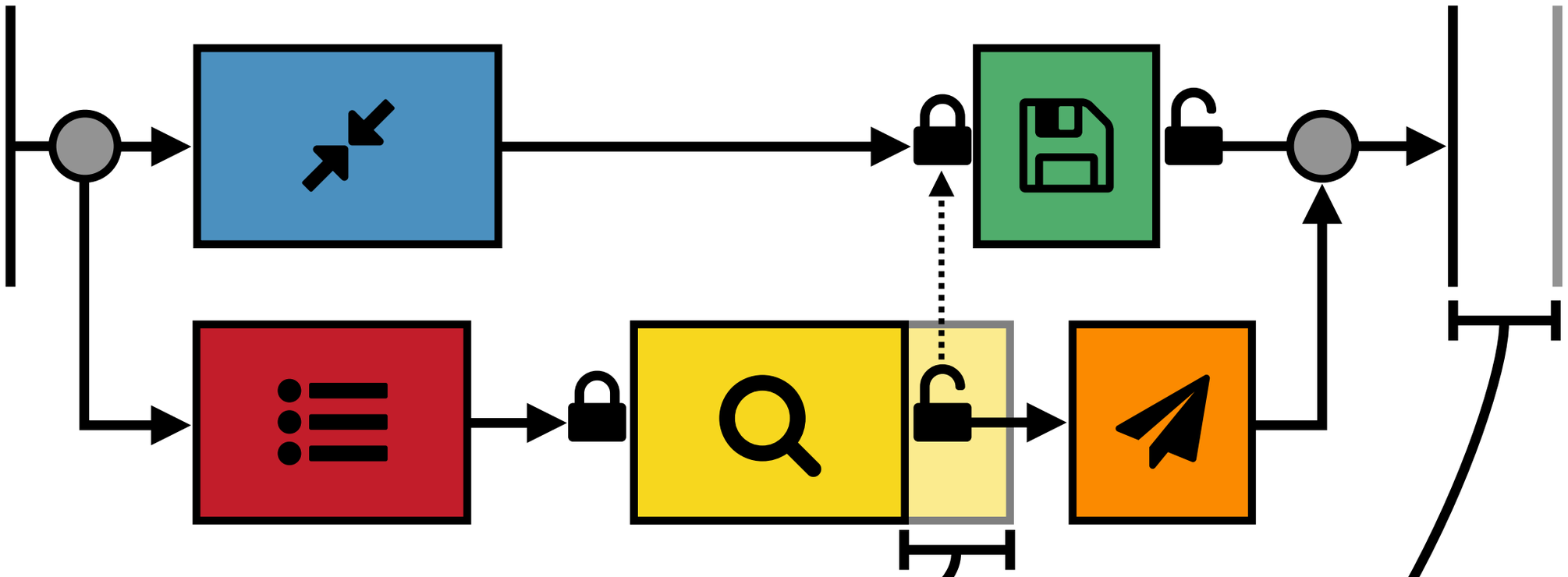
If we could magically speed up 🔍 …



More speedup in 🔍…

Performance Experiments

If we could magically speed up 🔍 …

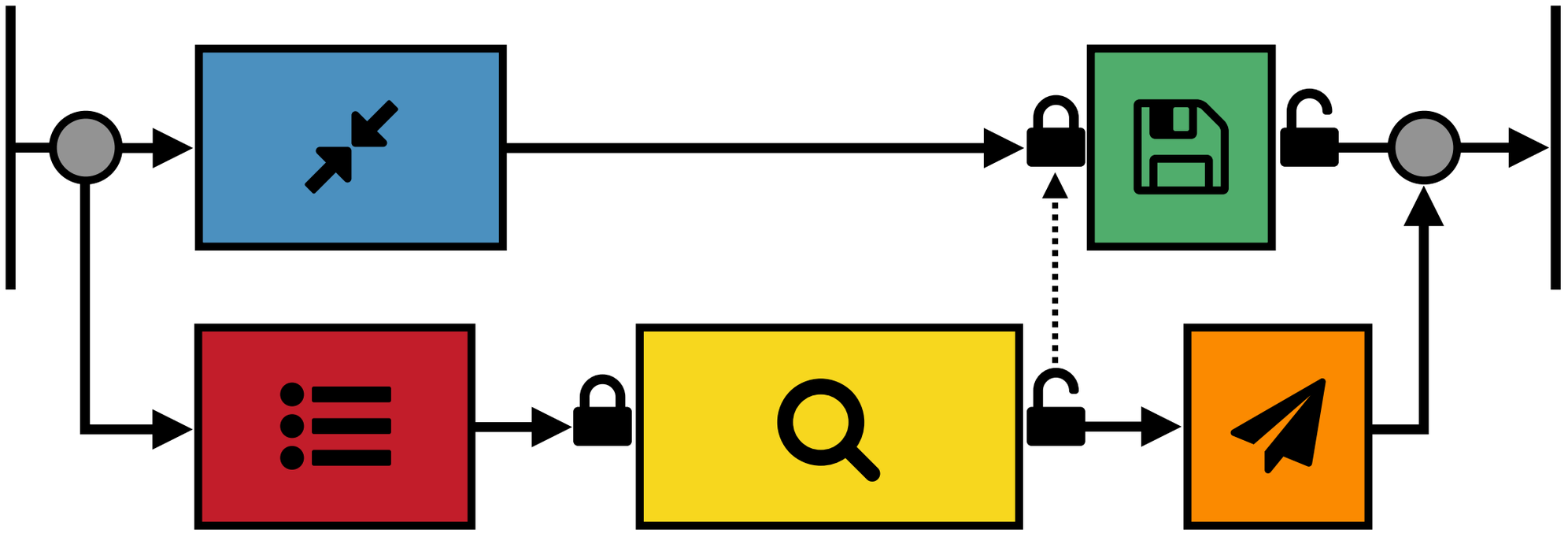


More speedup in 🔍 …

leads to a larger program speedup.

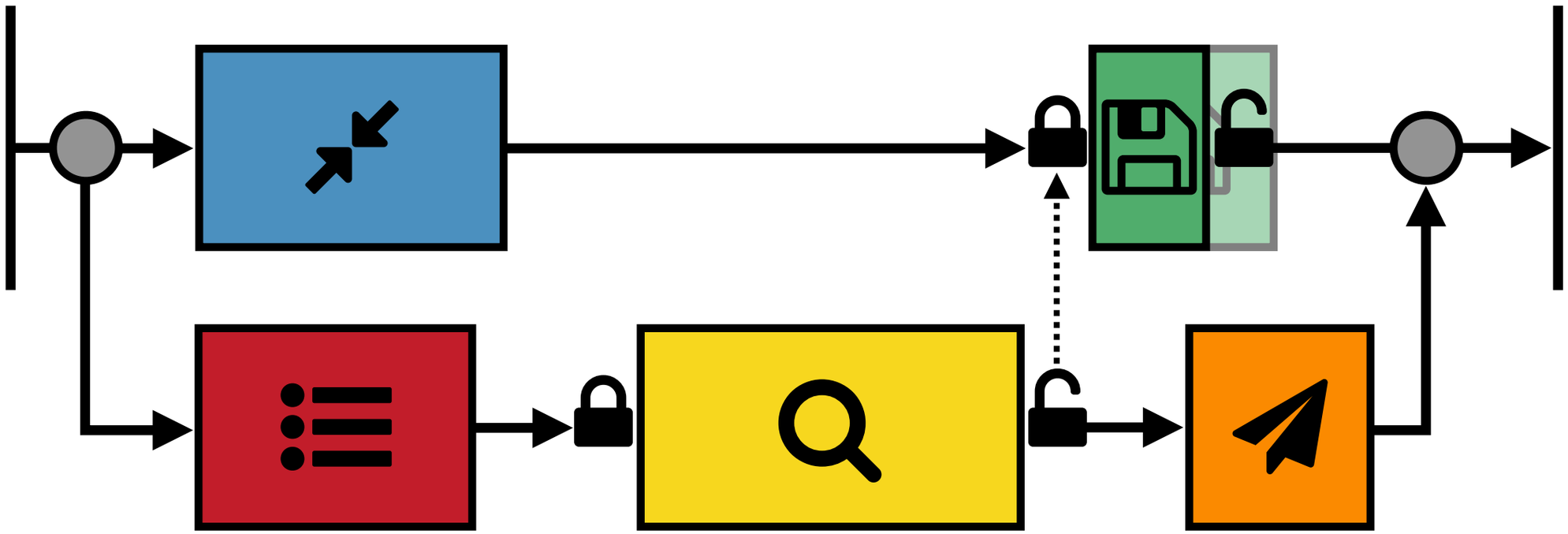
Performance Experiments

If we could magically speed up  ...



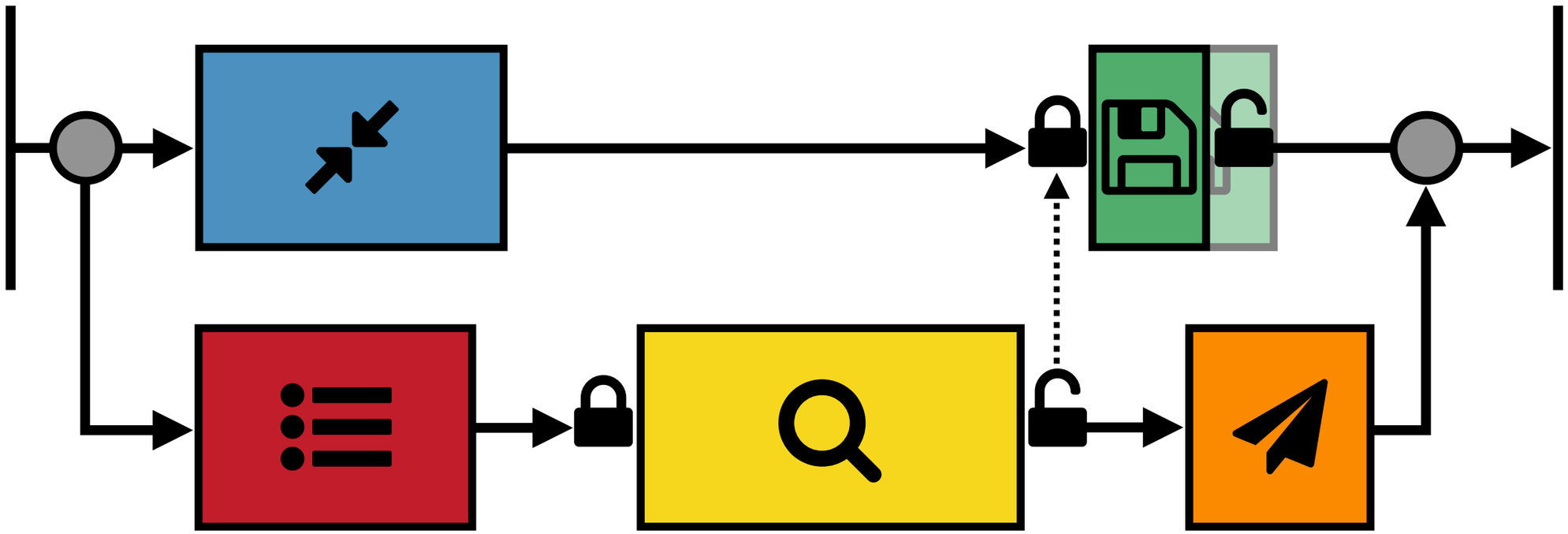
Performance Experiments

If we could magically speed up 💾 ...



Performance Experiments

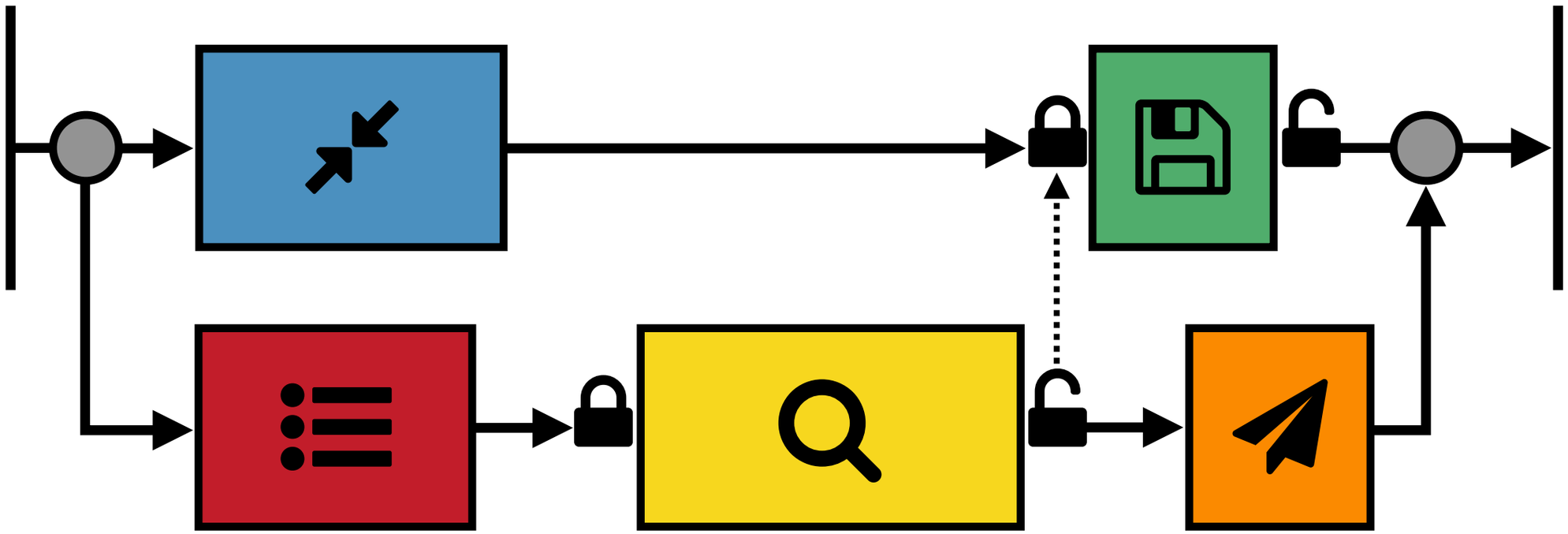
If we could magically speed up 💾 ...



No program speedup

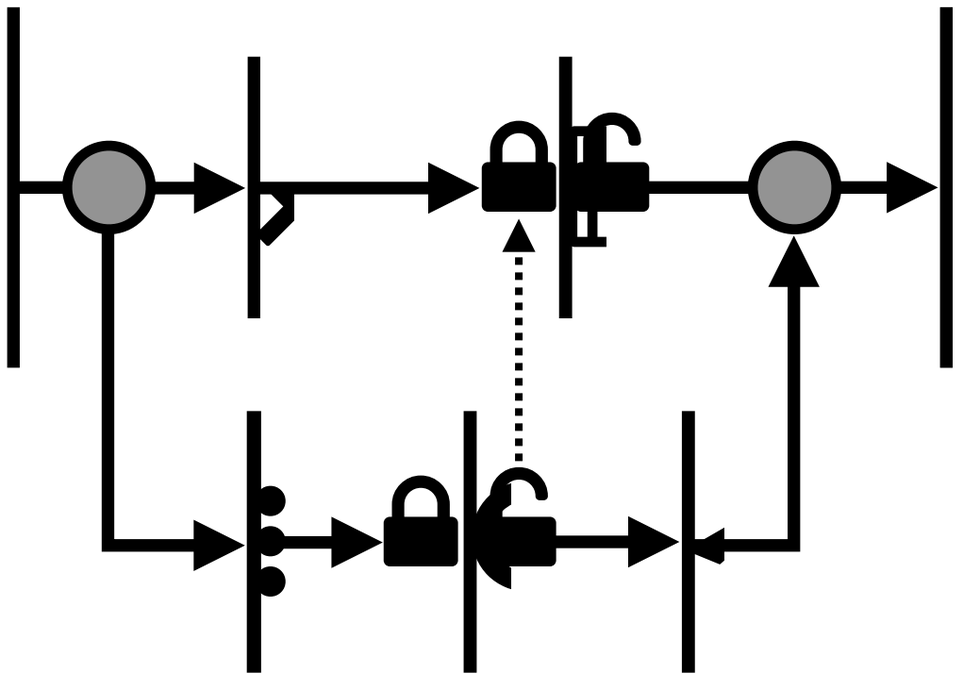
Performance Experiments

We’re going to have to do this without magic.



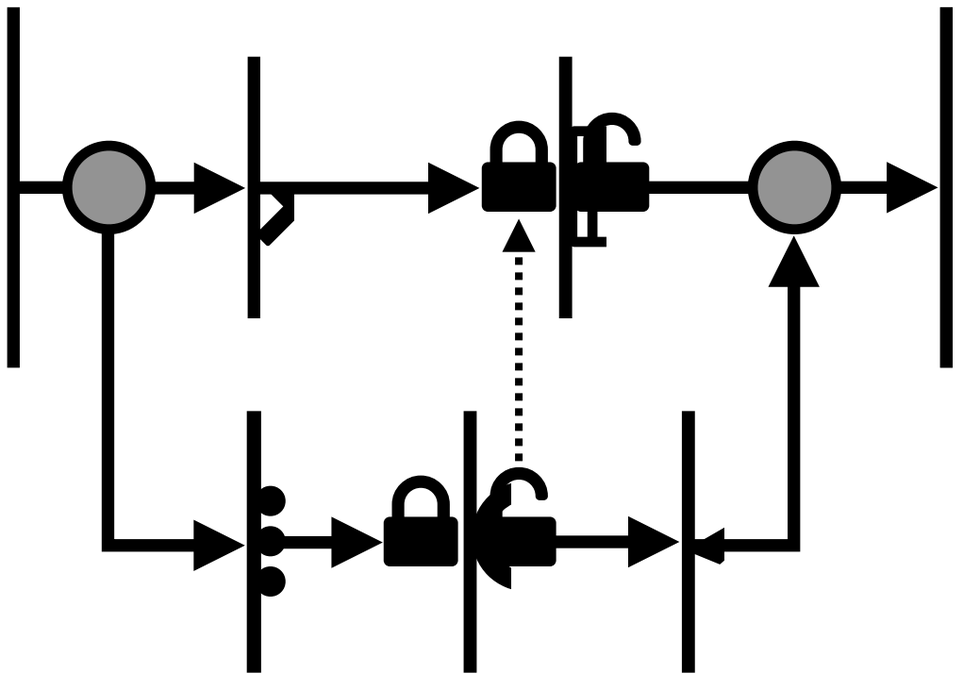
Performance Experiments

We’re going to have to do this without magic.



Performance Experiments

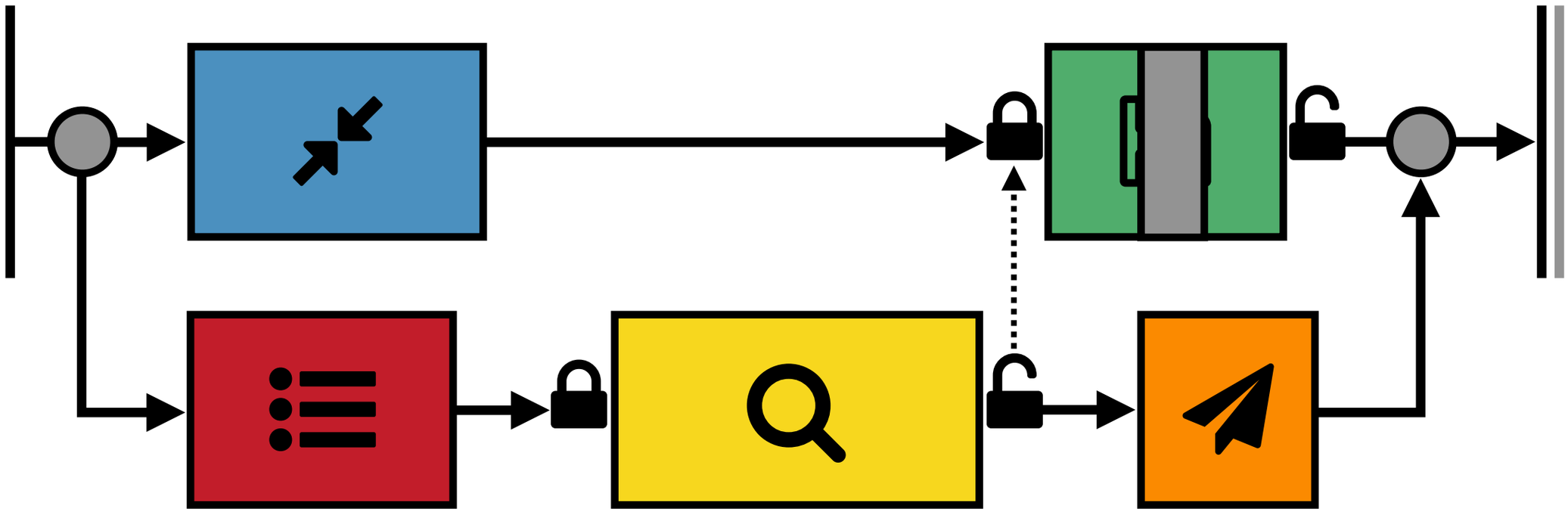
We’re going to have to do this without magic.



Otherwise we’d just do this.

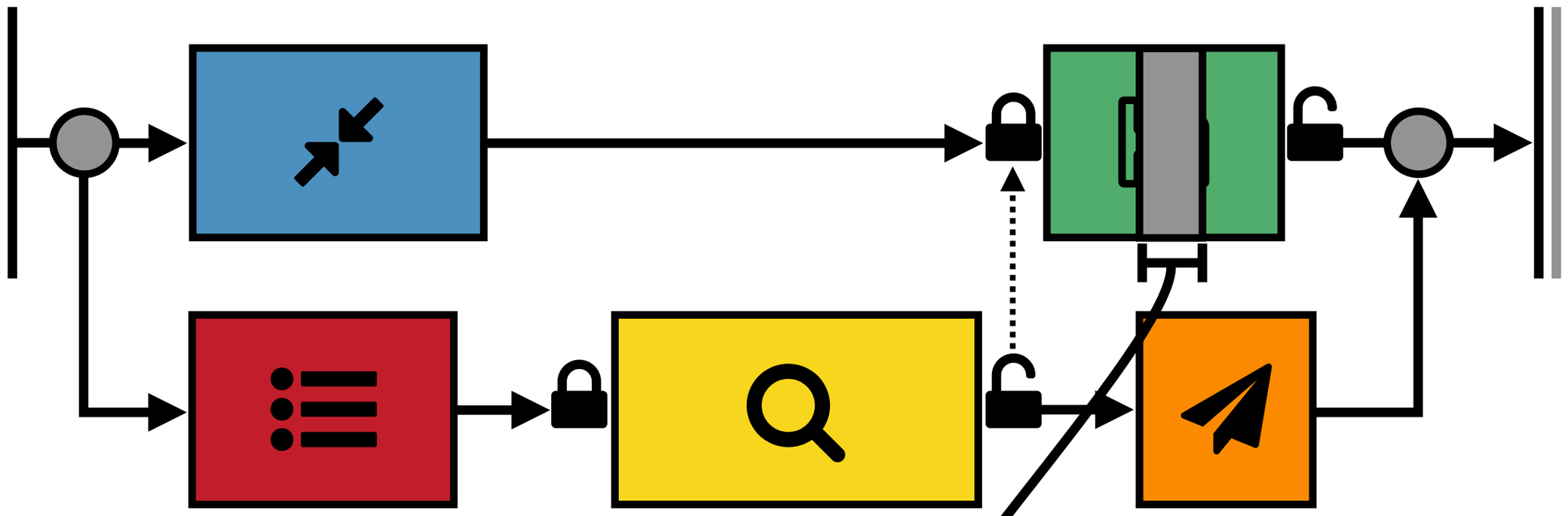
Virtual Speedup

“Speed up”  by slowing everything else down.



Virtual Speedup

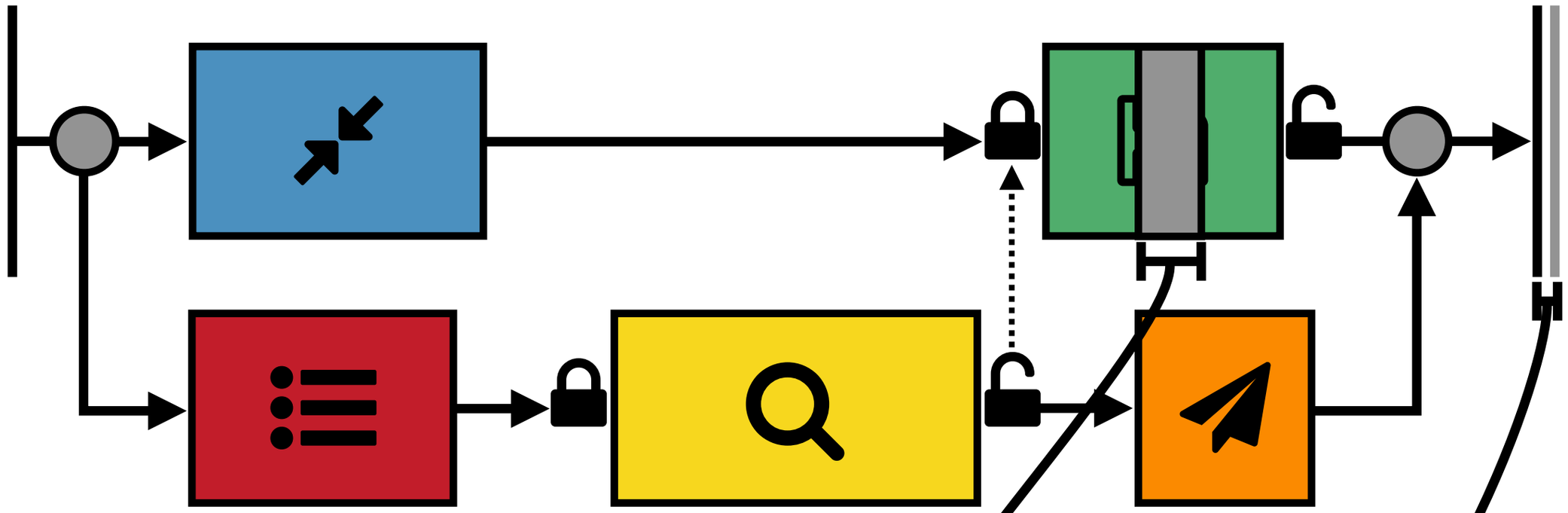
“Speed up”  by slowing everything else down.



Speeding up  by this much…

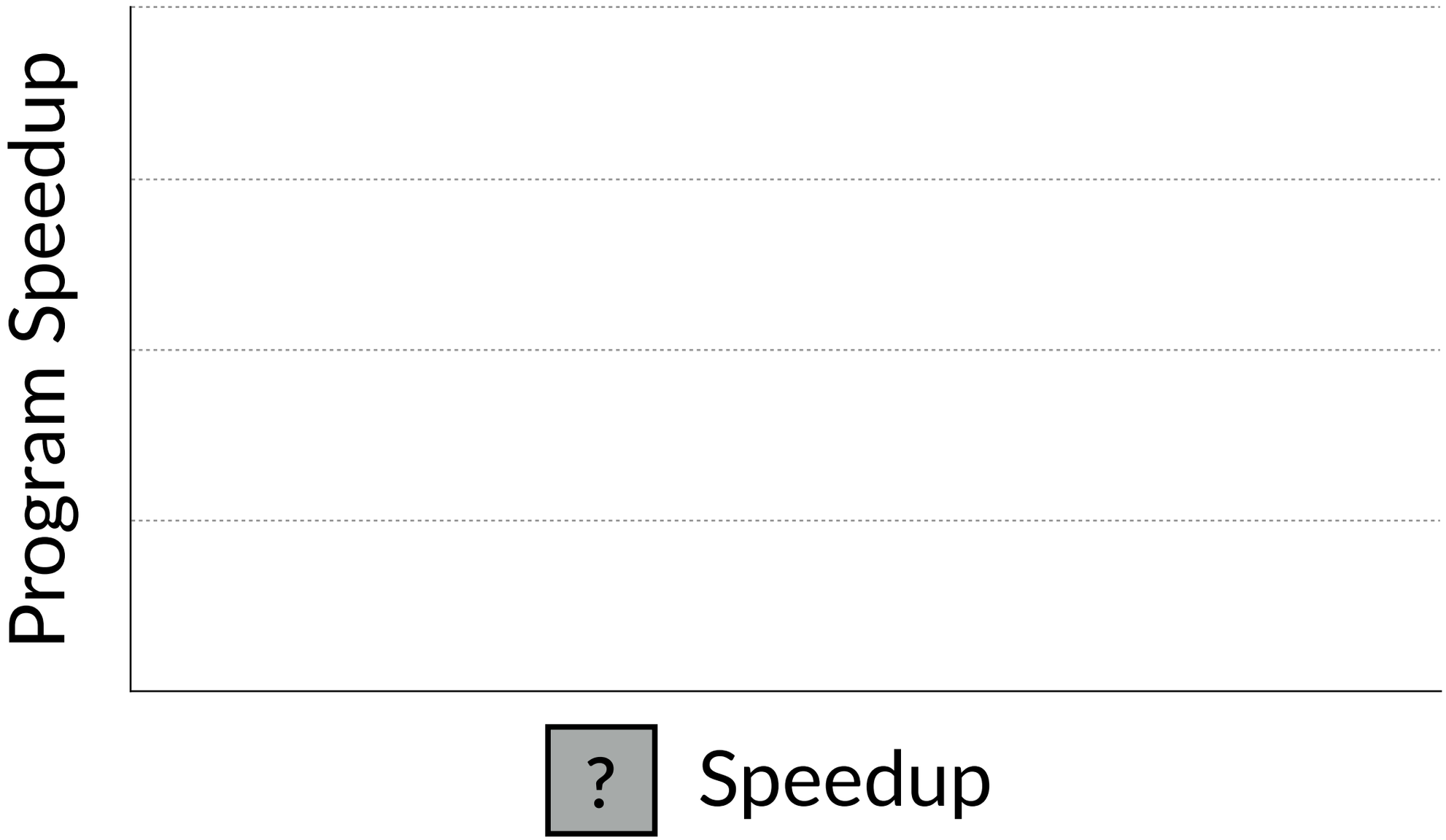
Virtual Speedup

“Speed up” by slowing everything else down.

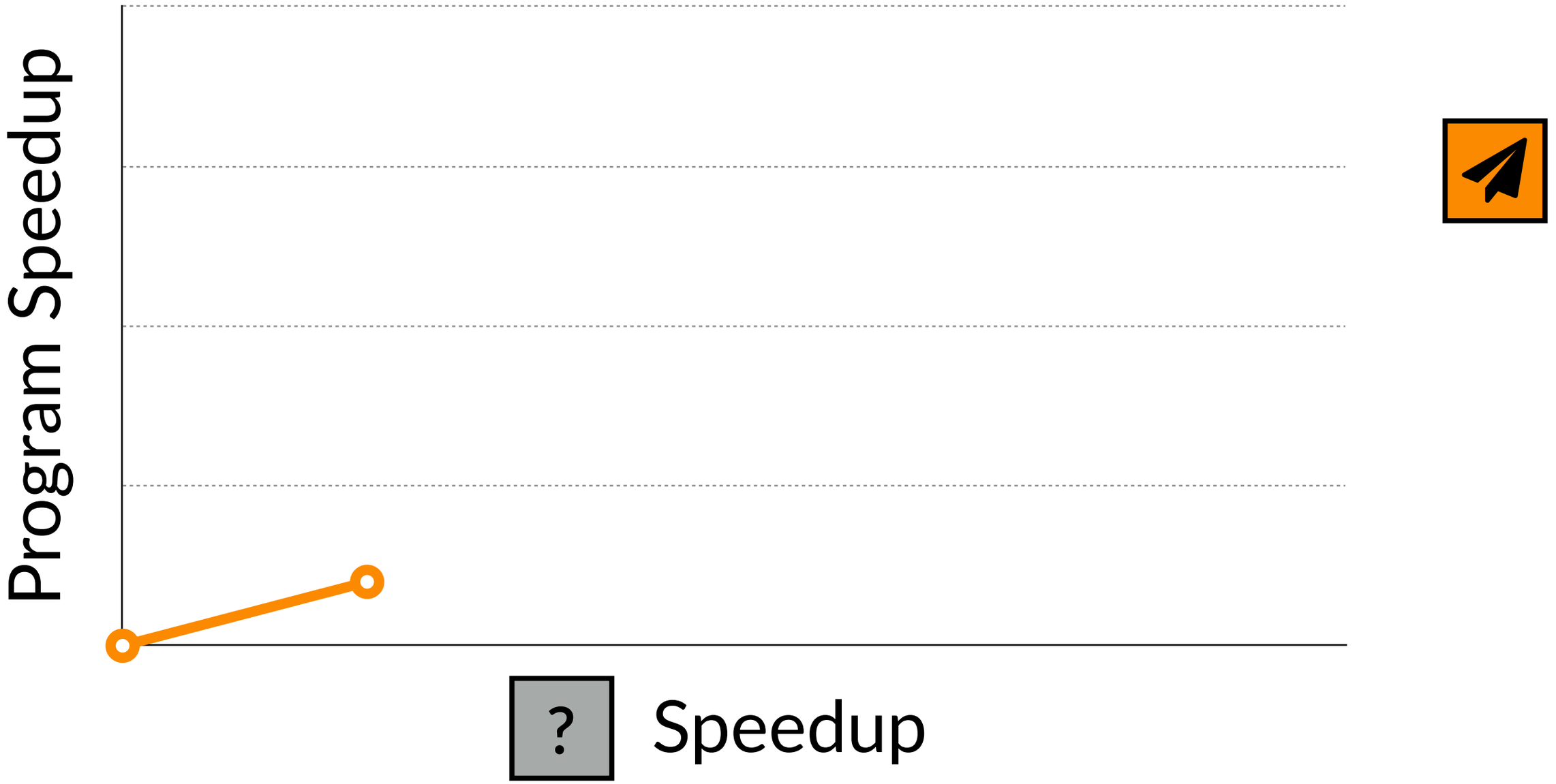


Speeding up by this much…
speeds up the program by this much.

Speedup Results

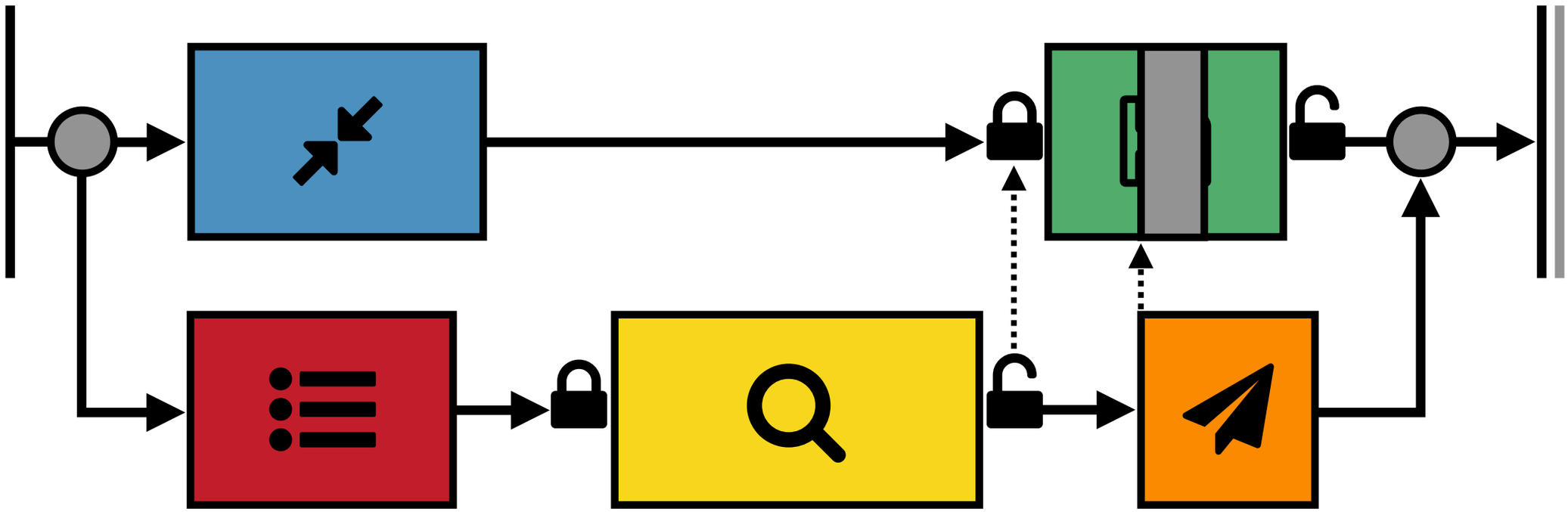


Speedup Results



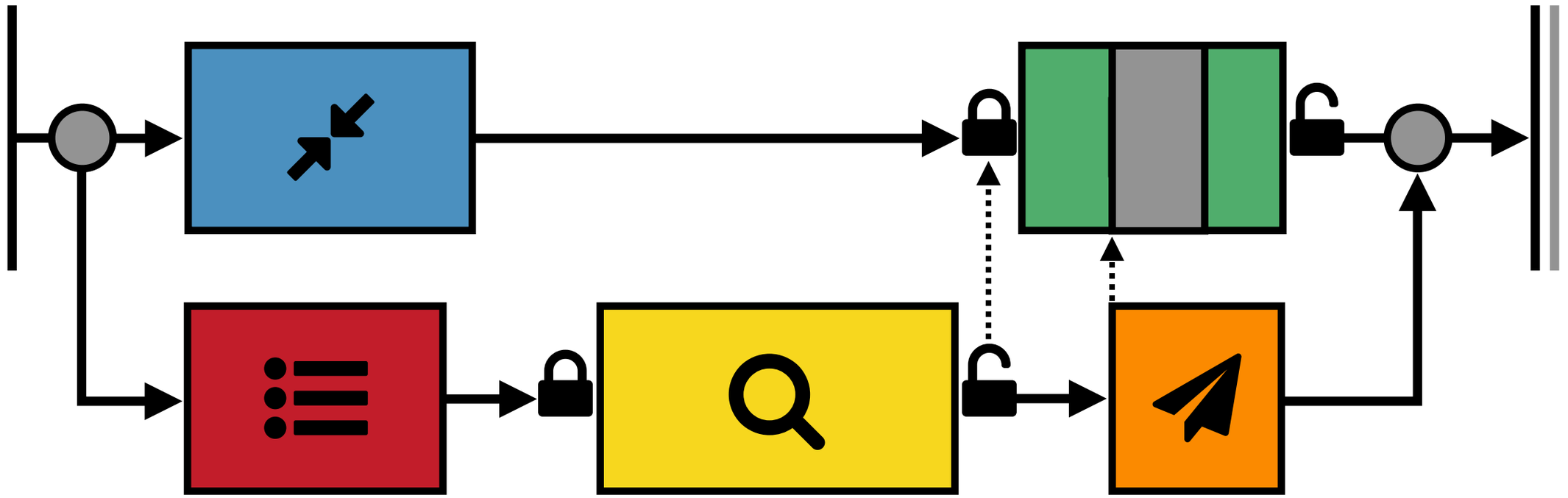
Virtual Speedup

“Speed up”  by slowing everything else down.



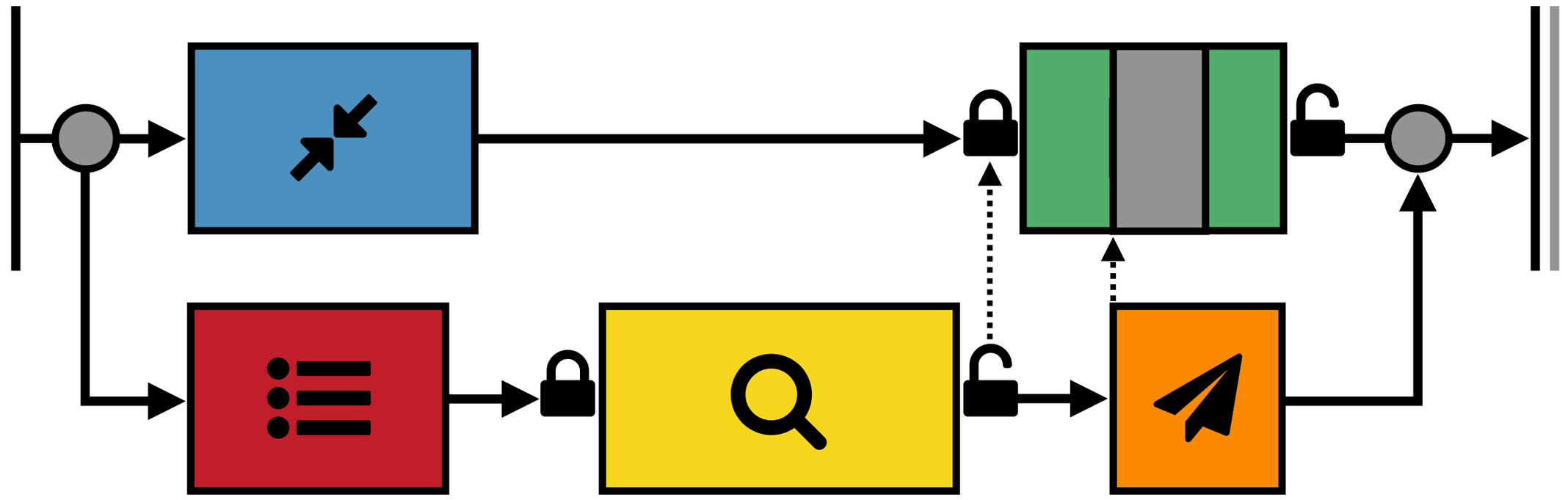
Virtual Speedup

“Speed up”  by slowing everything else down.



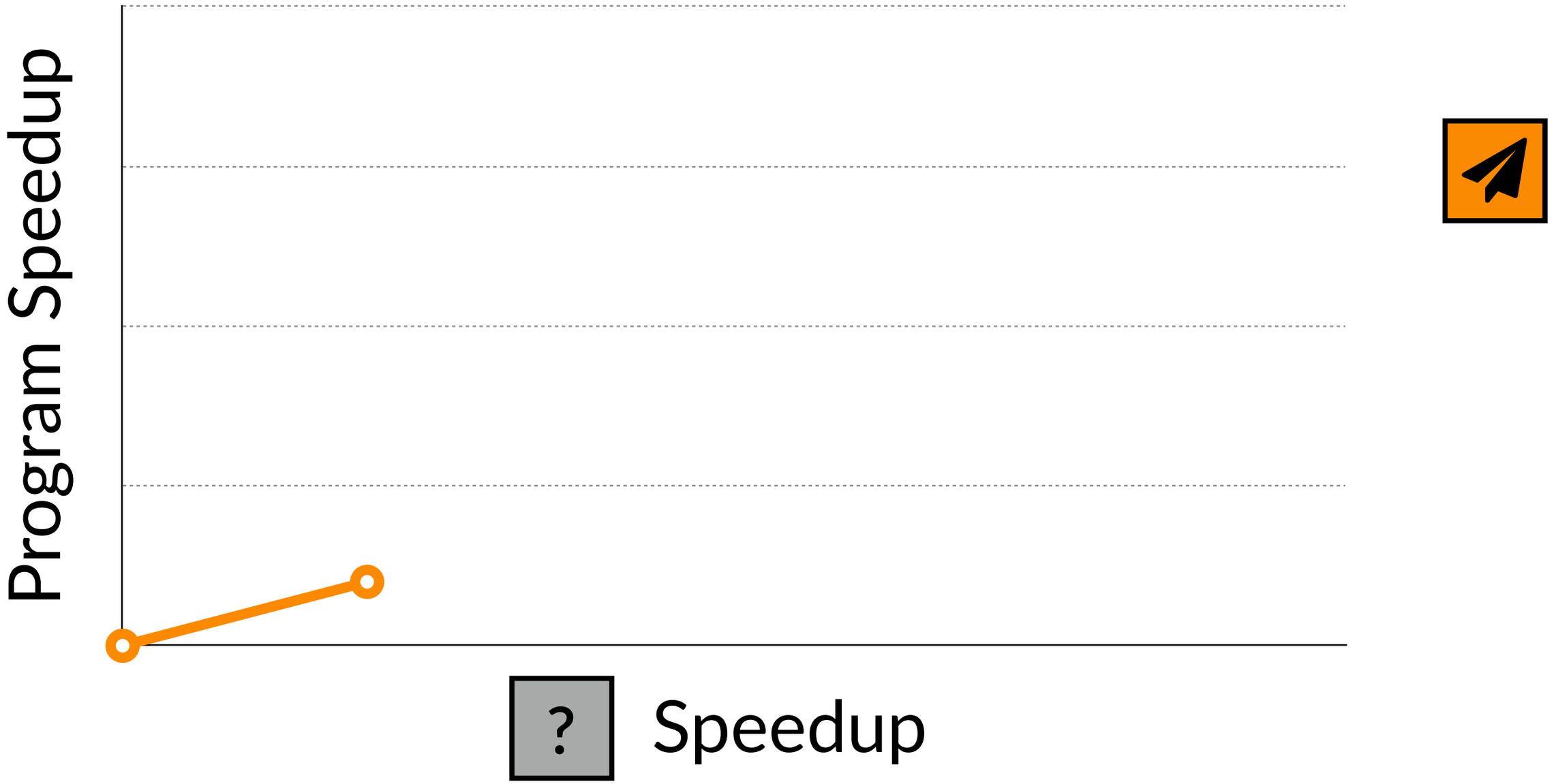
Virtual Speedup

“Speed up”  by slowing everything else down.

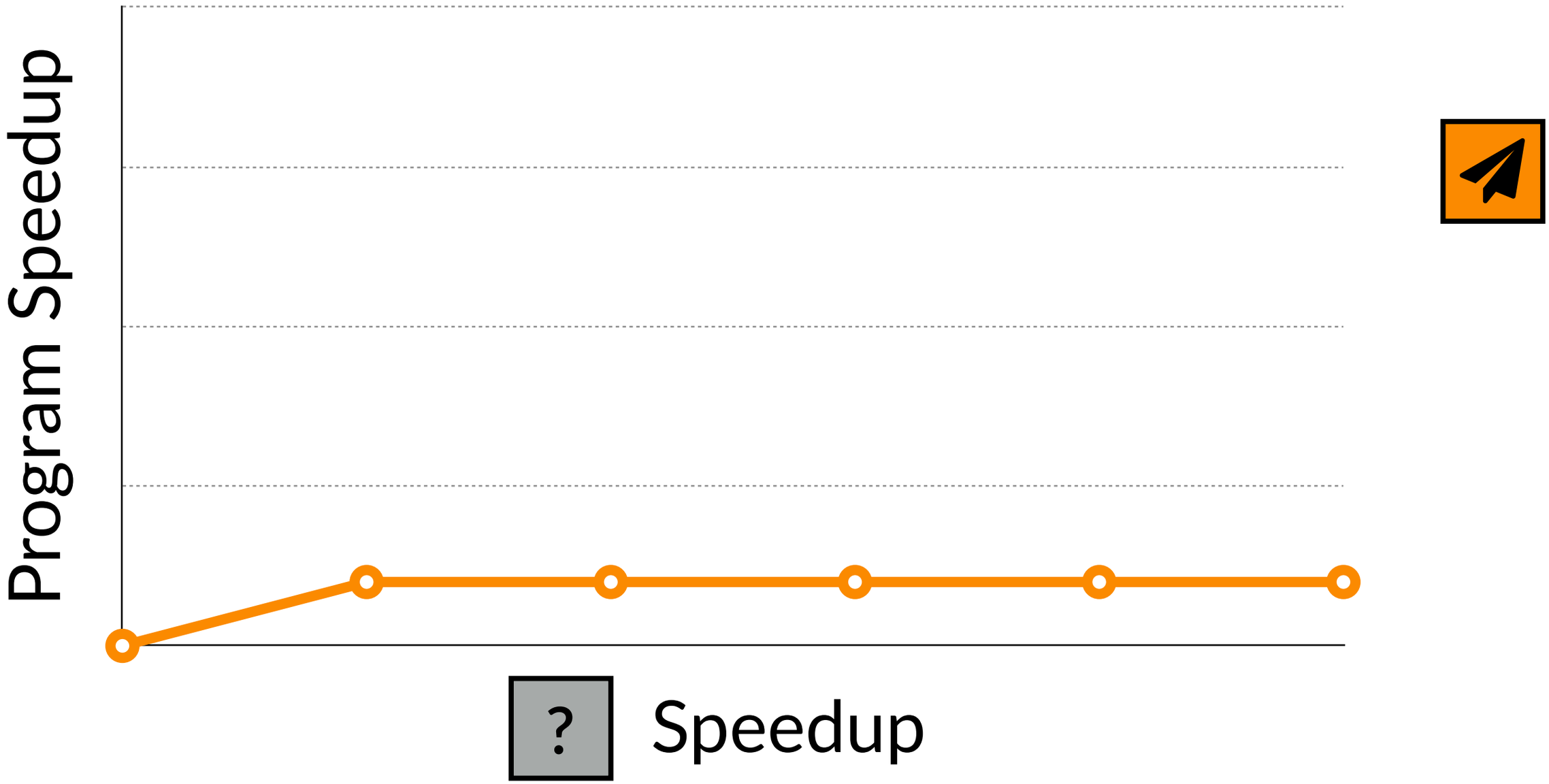


A larger speedup has no additional effect

Speedup Results

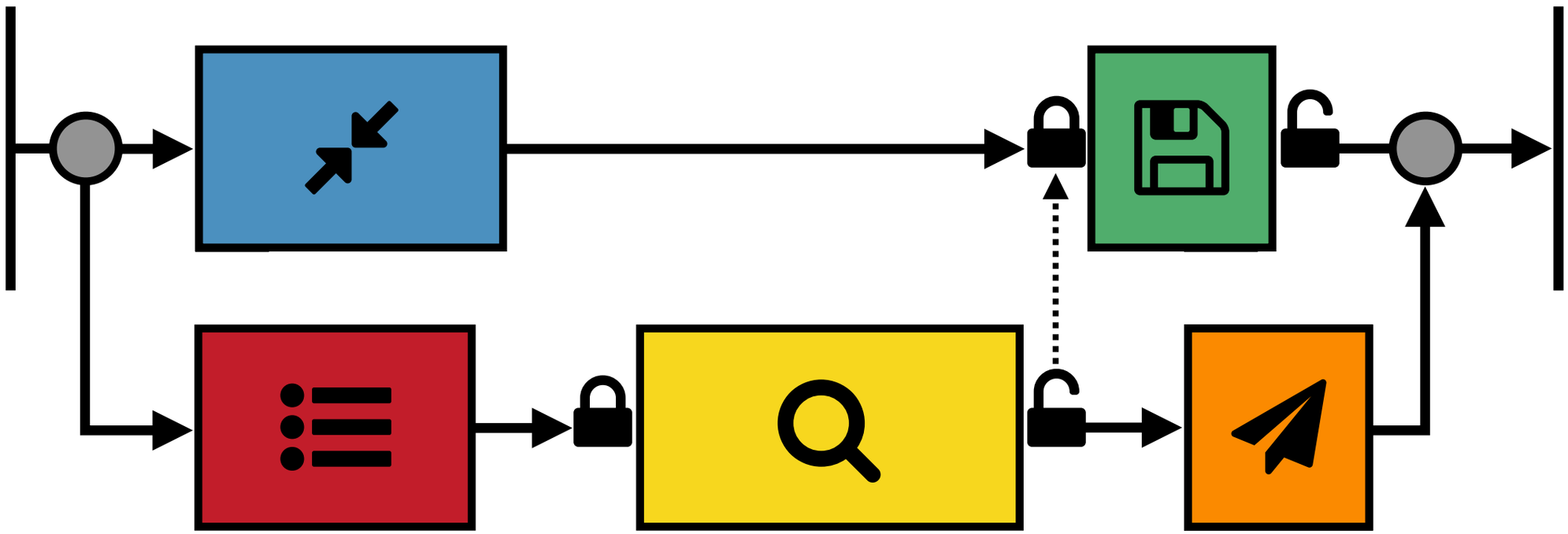


Speedup Results



Virtual Speedup

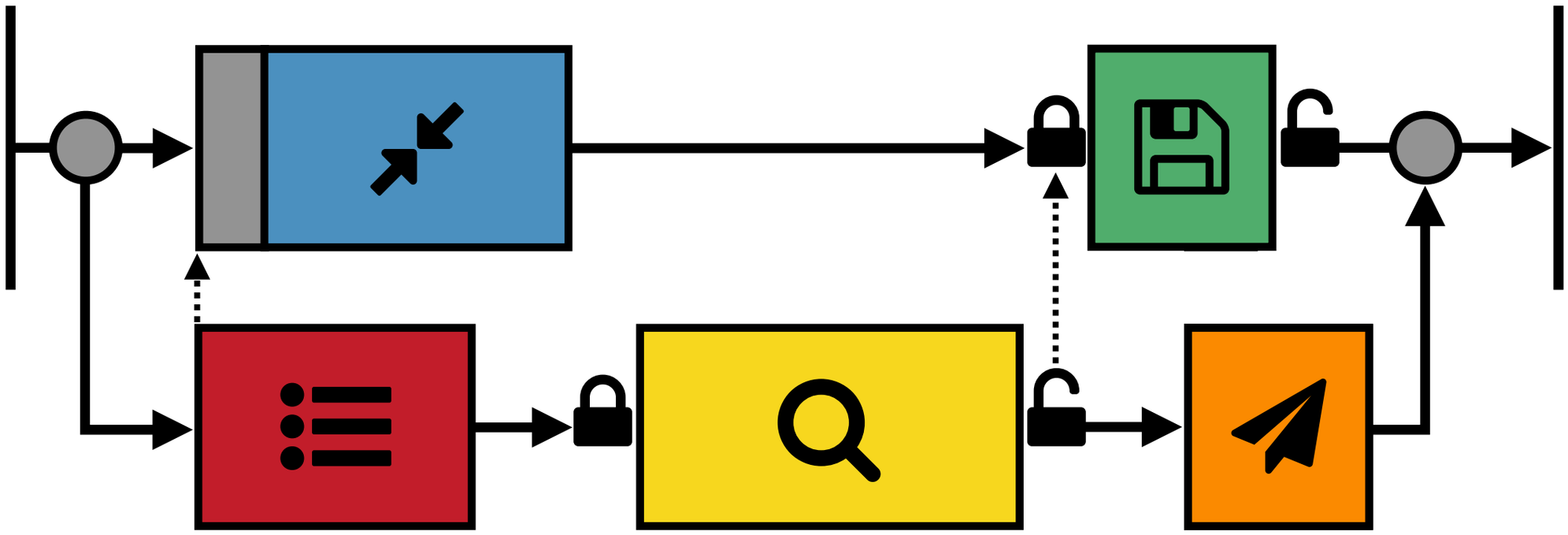
“Speed up”  by slowing everything else down.



Each time  runs, pause all other threads.

Virtual Speedup

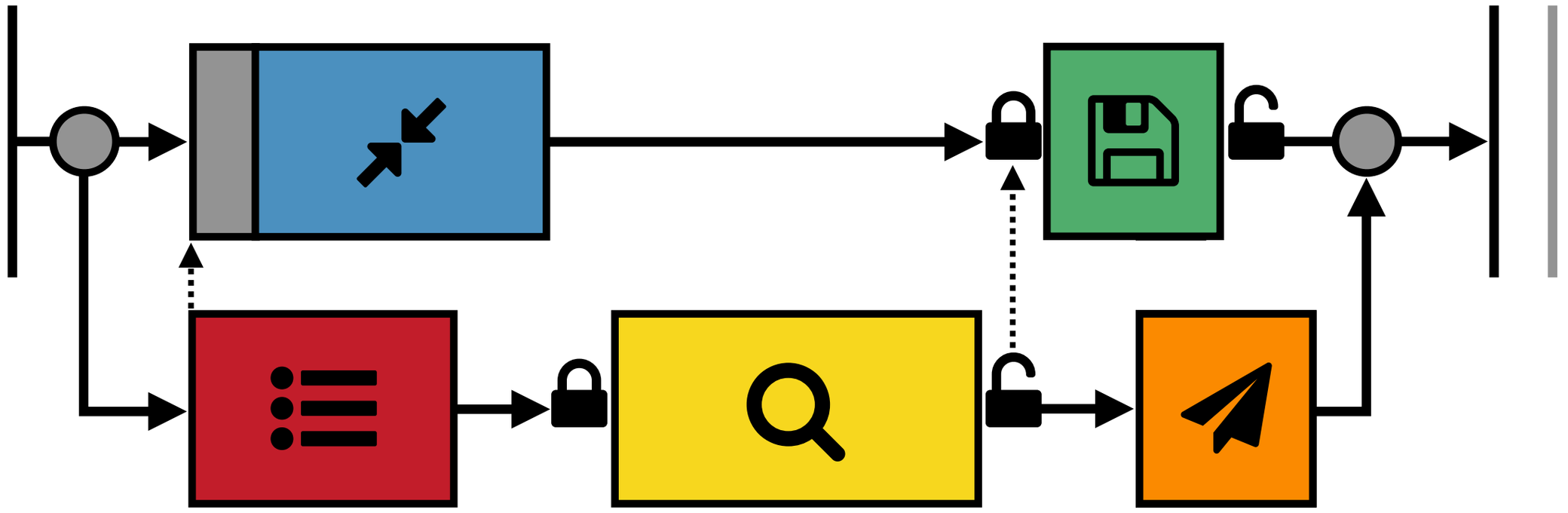
“Speed up”  by slowing everything else down.



Each time  runs, pause all other threads.

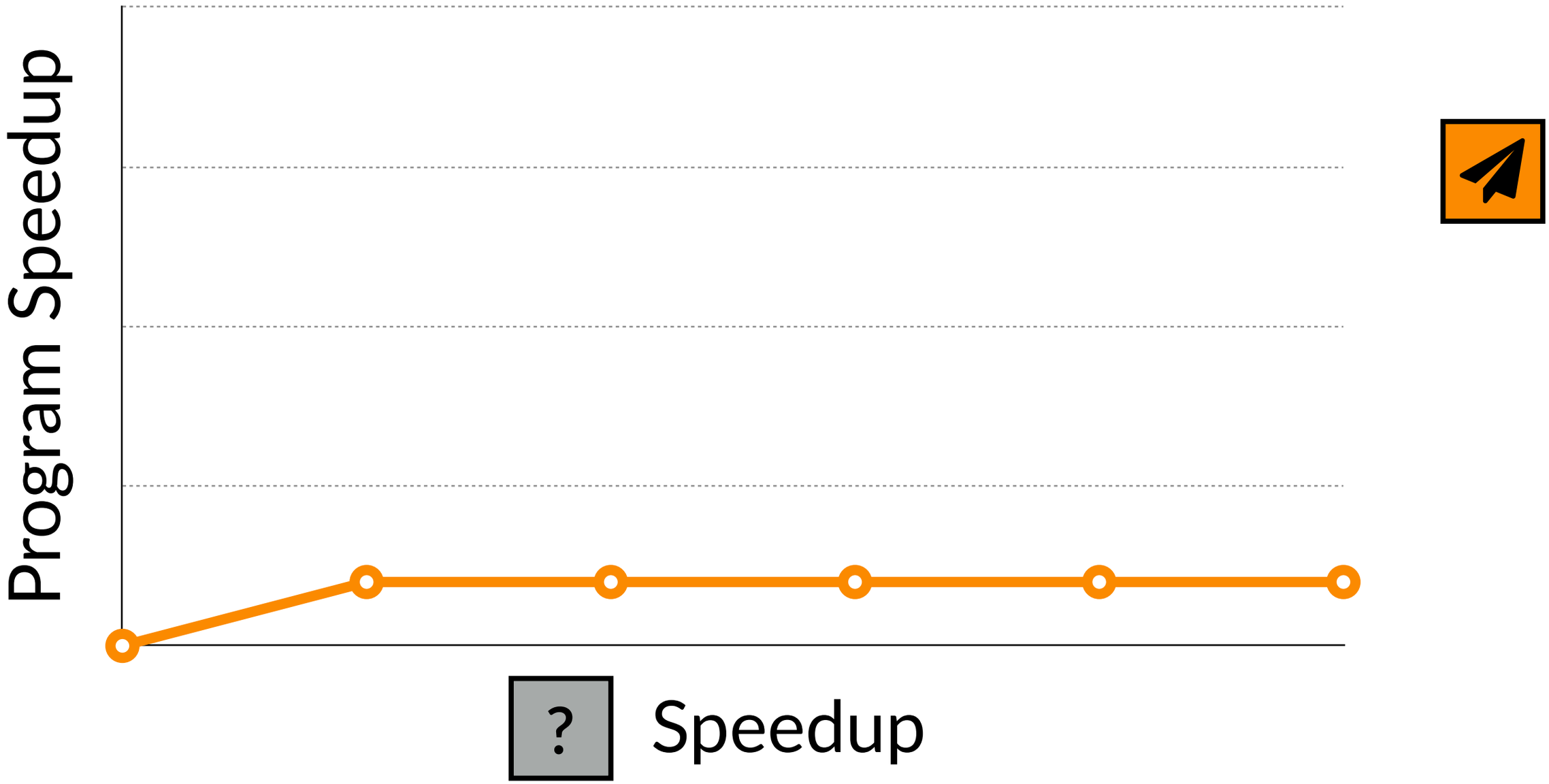
Virtual Speedup

“Speed up”  by slowing everything else down.

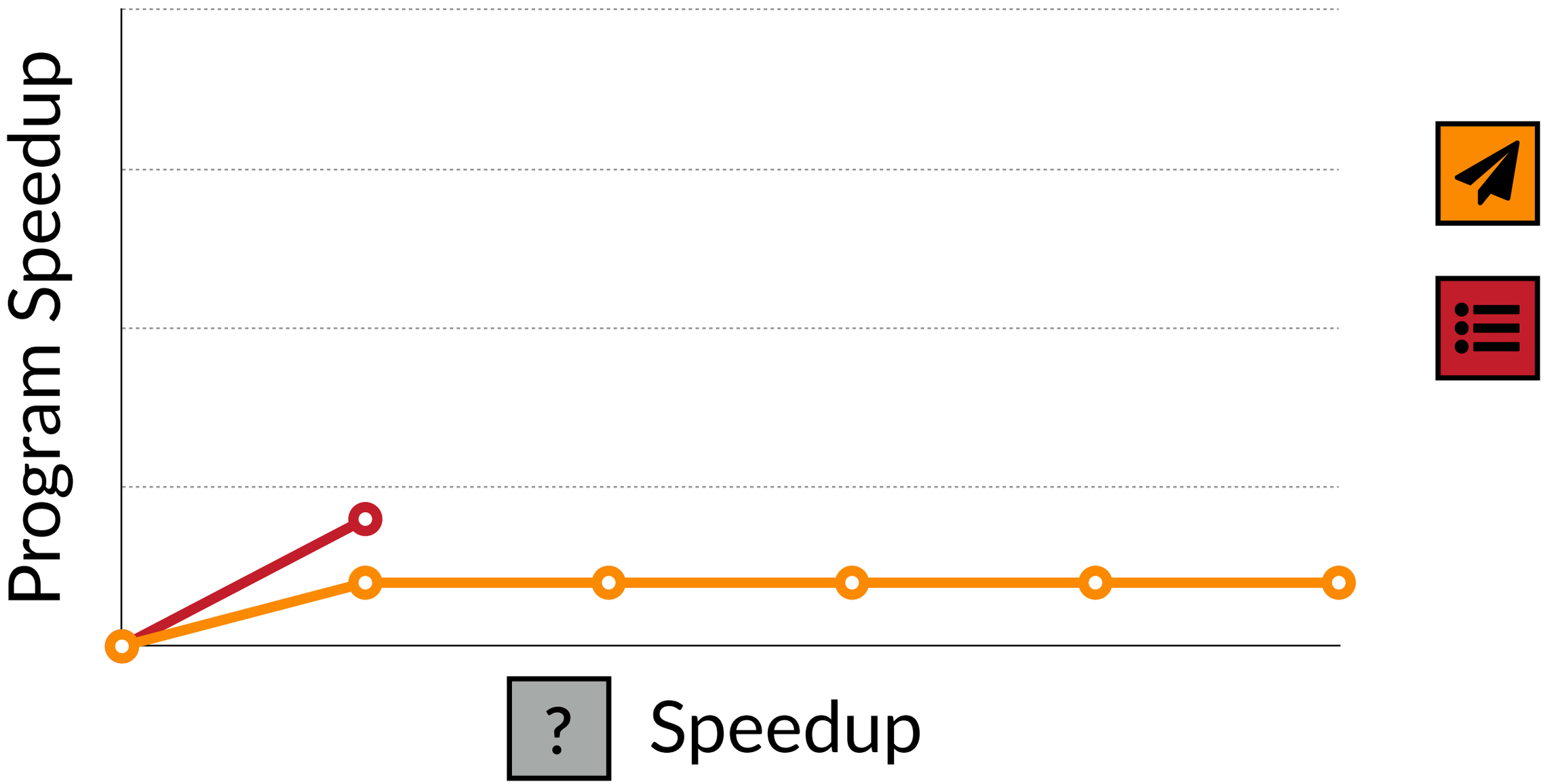


Each time  runs, pause all other threads.

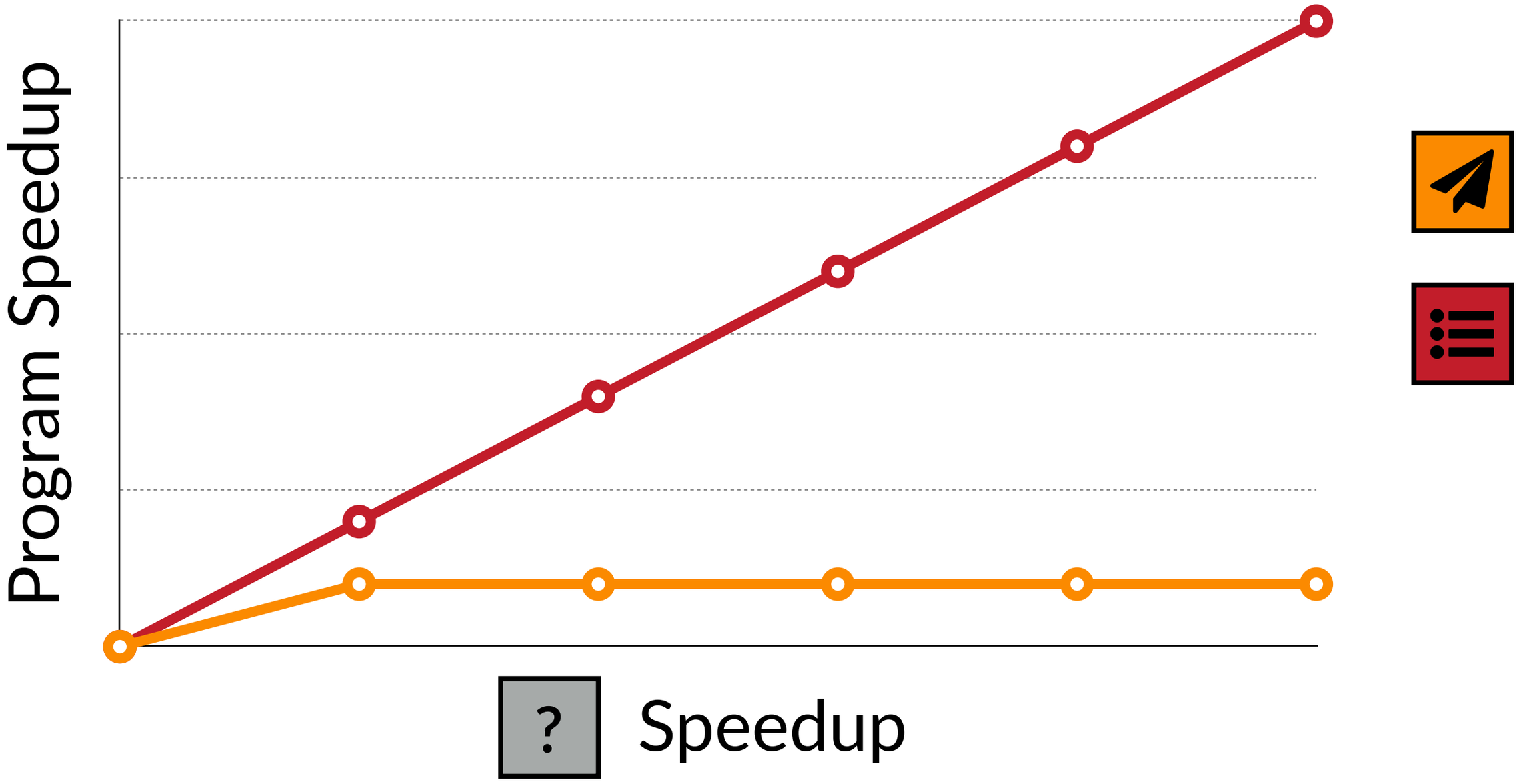
Speedup Results



Speedup Results

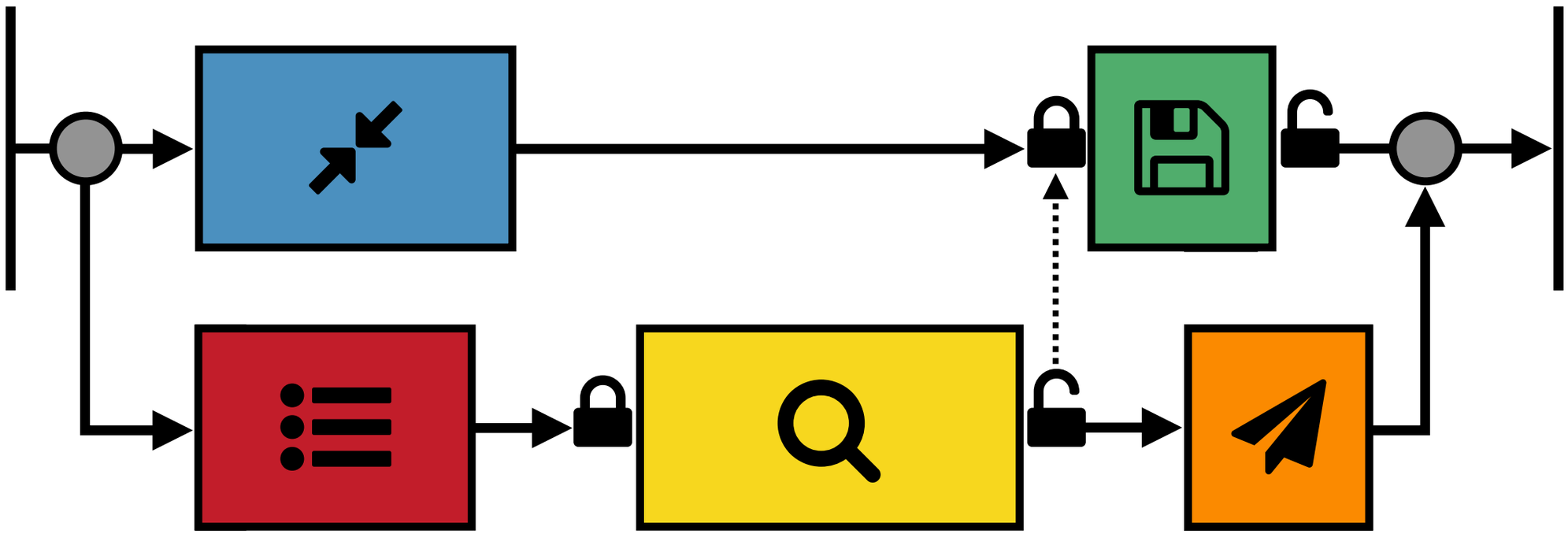


Speedup Results



Virtual Speedup

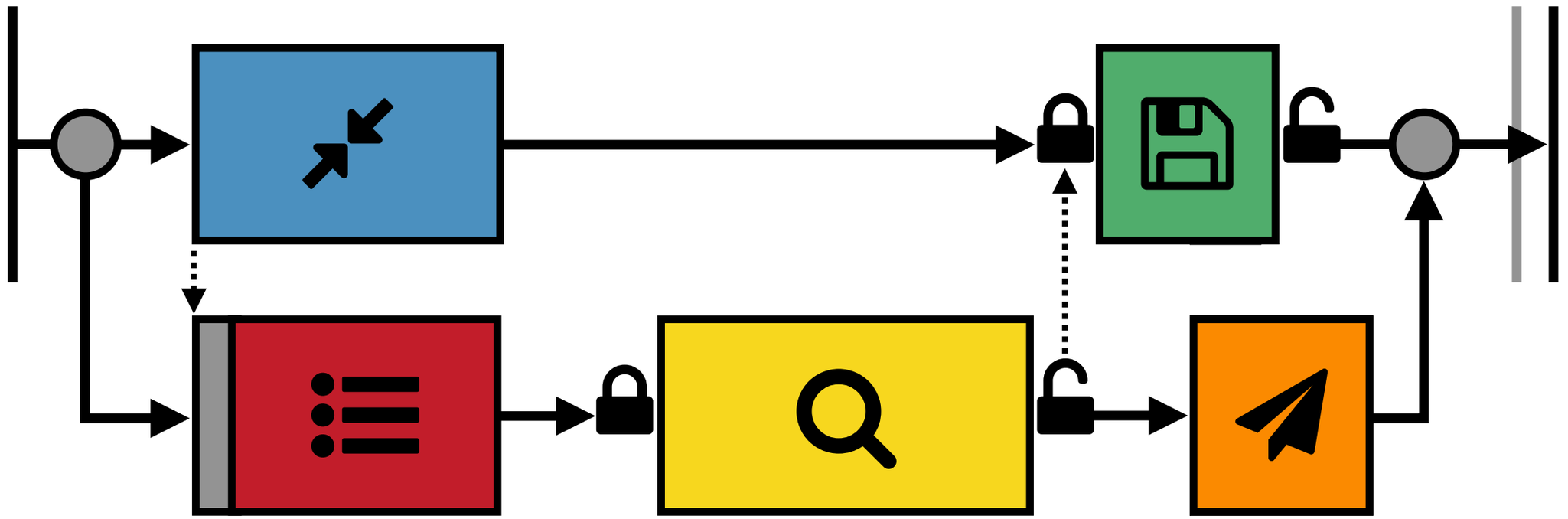
“Speed up”  by slowing everything else down.



Each time  runs, pause all other threads.

Virtual Speedup

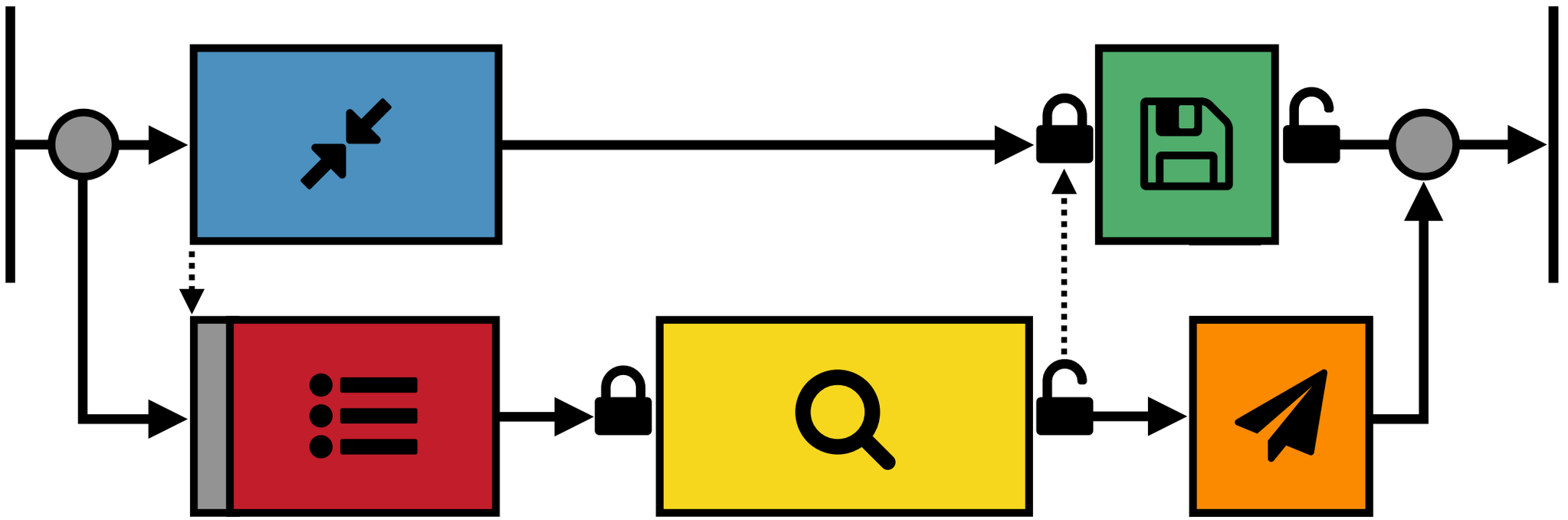
“Speed up”  by slowing everything else down.



Each time  runs, pause all other threads.

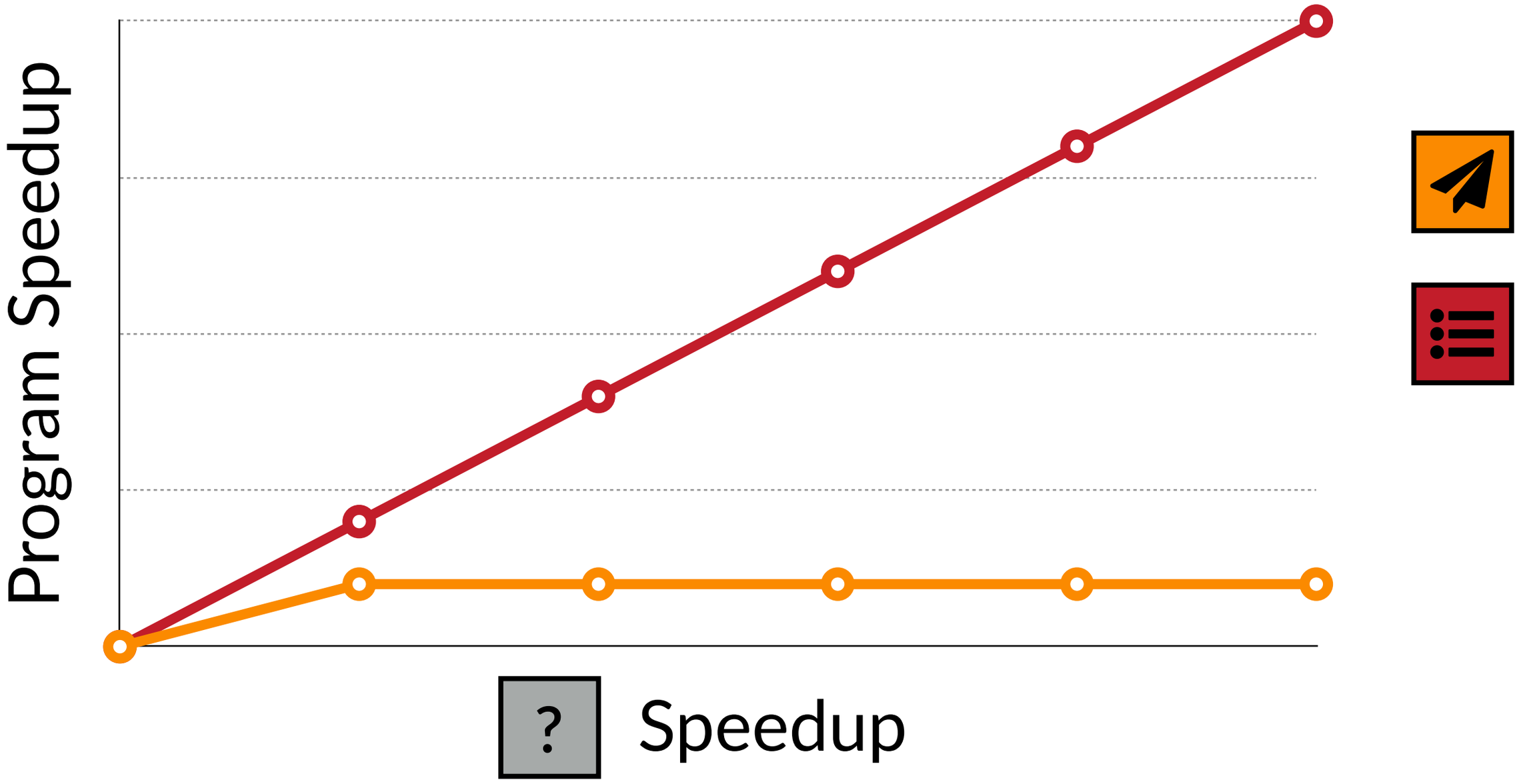
Virtual Speedup

“Speed up”  by slowing everything else down.

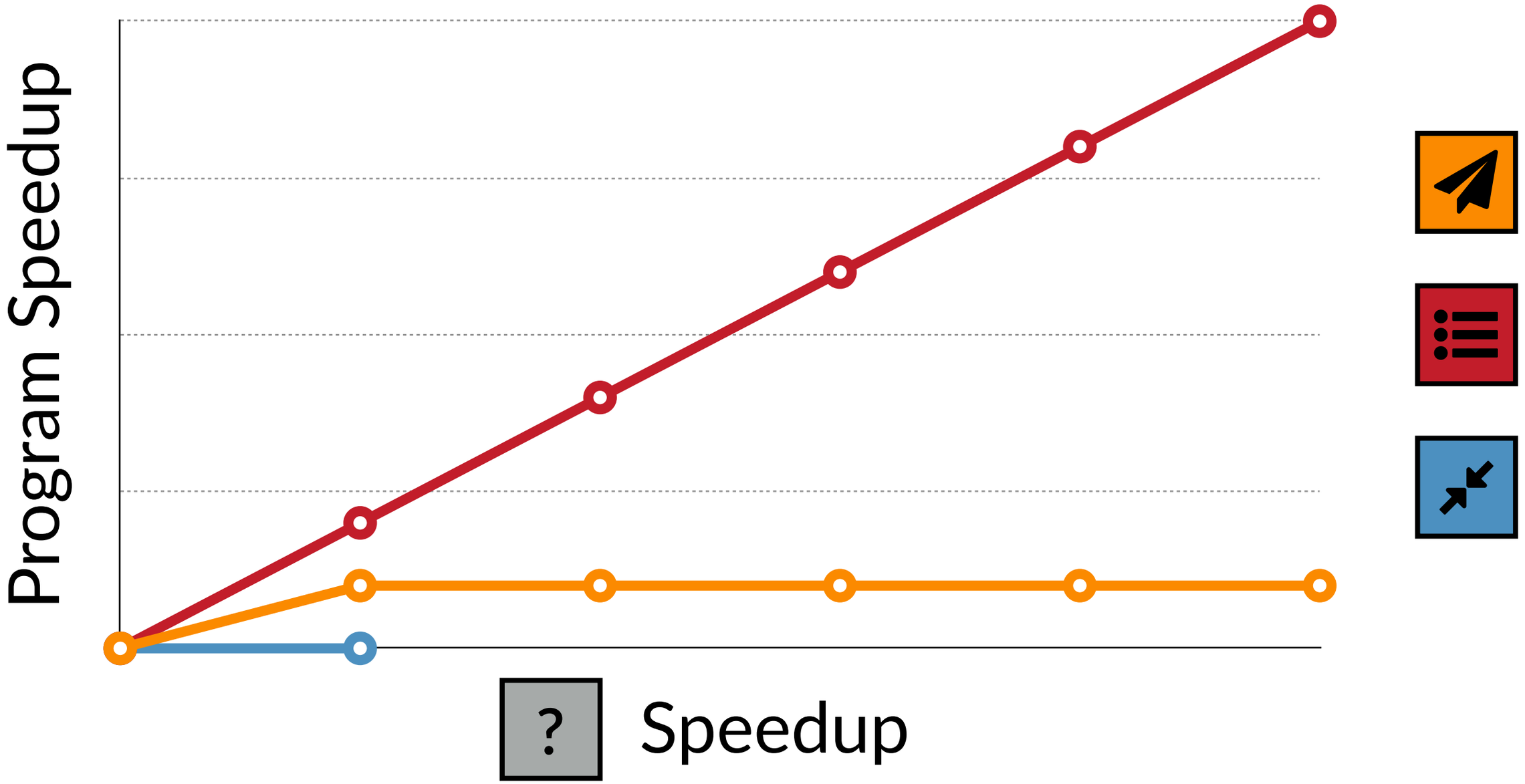


Each time  runs, pause all other threads.

Speedup Results

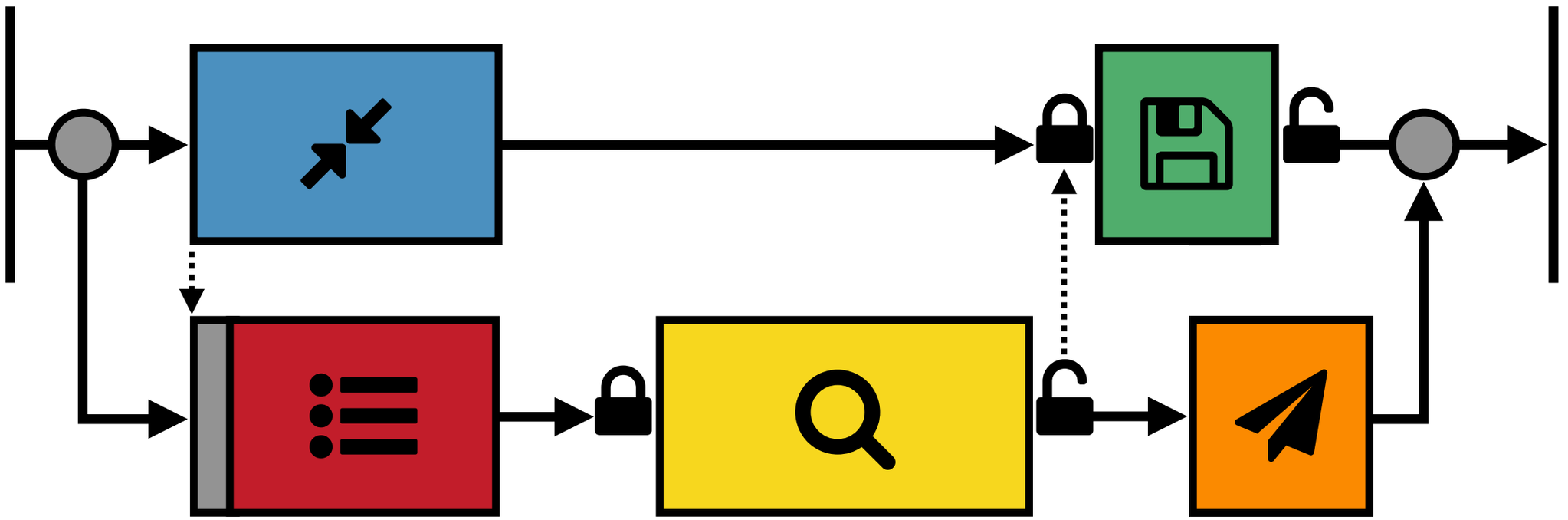


Speedup Results



Virtual Speedup

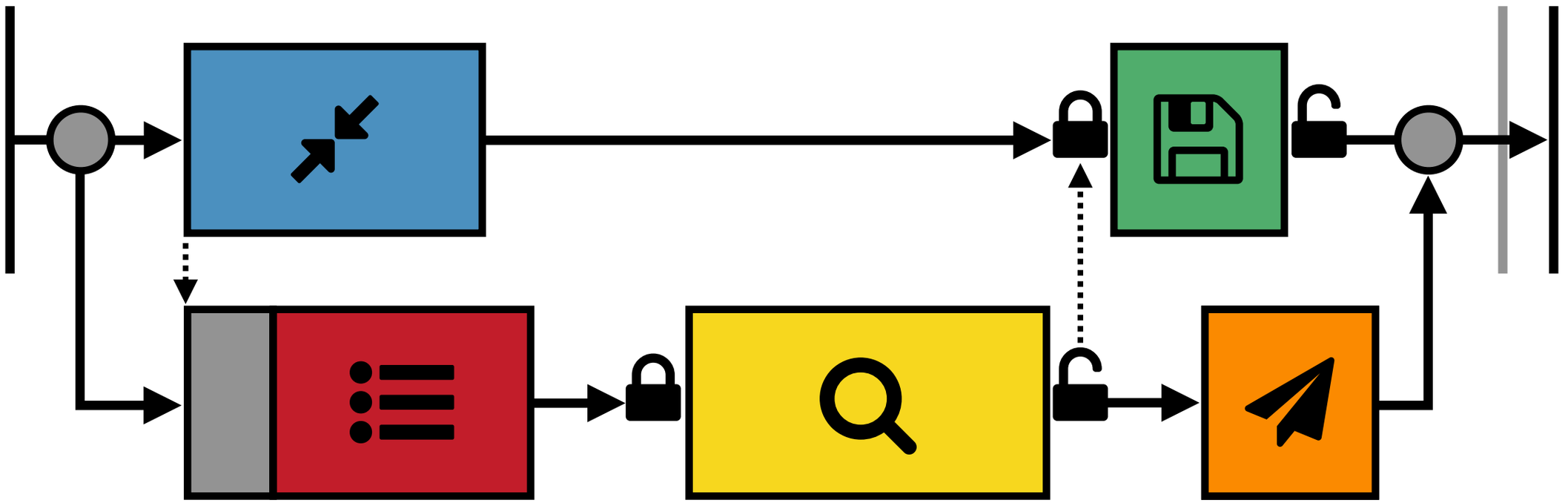
“Speed up”  by slowing everything else down.



Each time  runs, pause all other threads.

Virtual Speedup

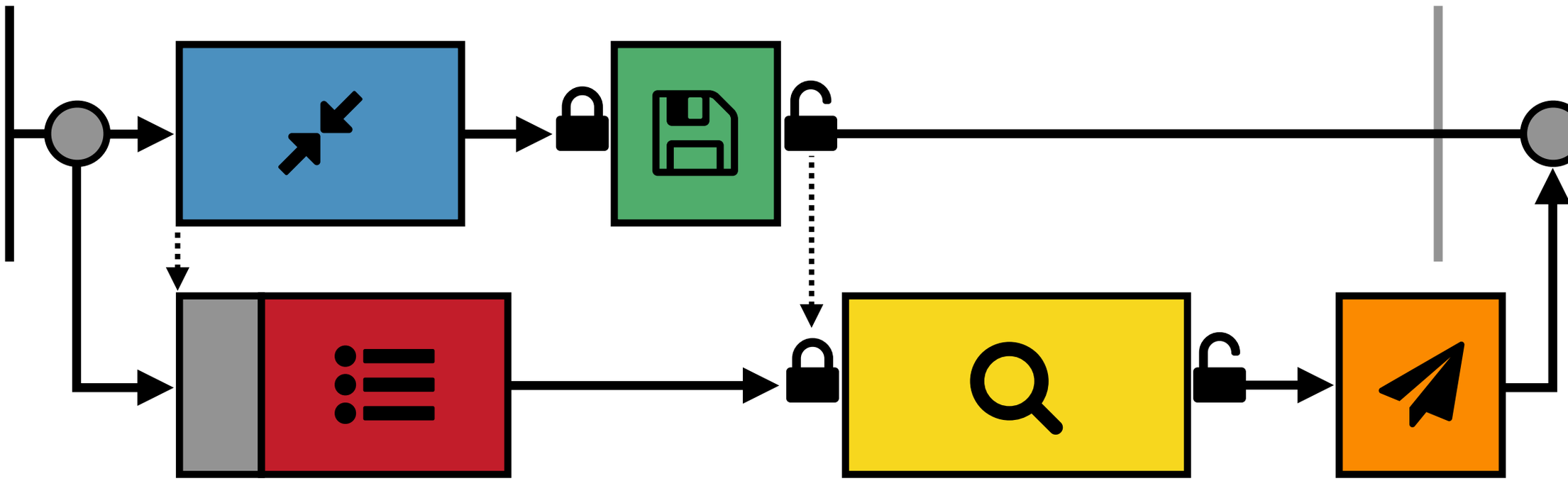
“Speed up”  by slowing everything else down.



Each time  runs, pause all other threads.

Virtual Speedup

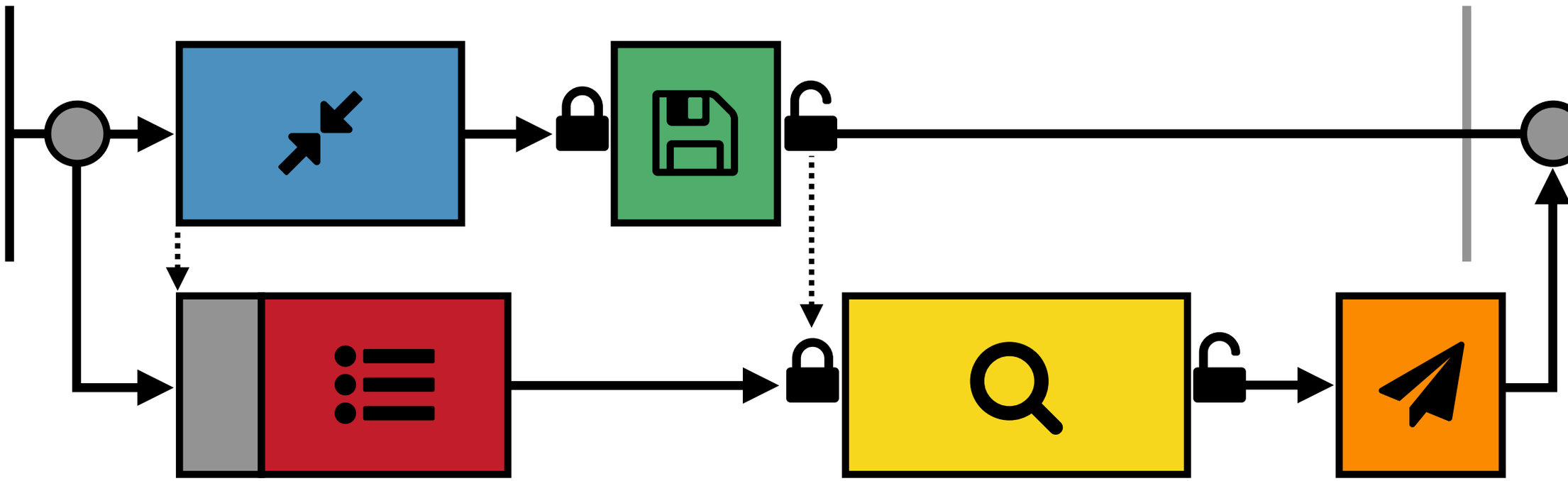
“Speed up”  by slowing everything else down.



Each time  runs, pause all other threads.

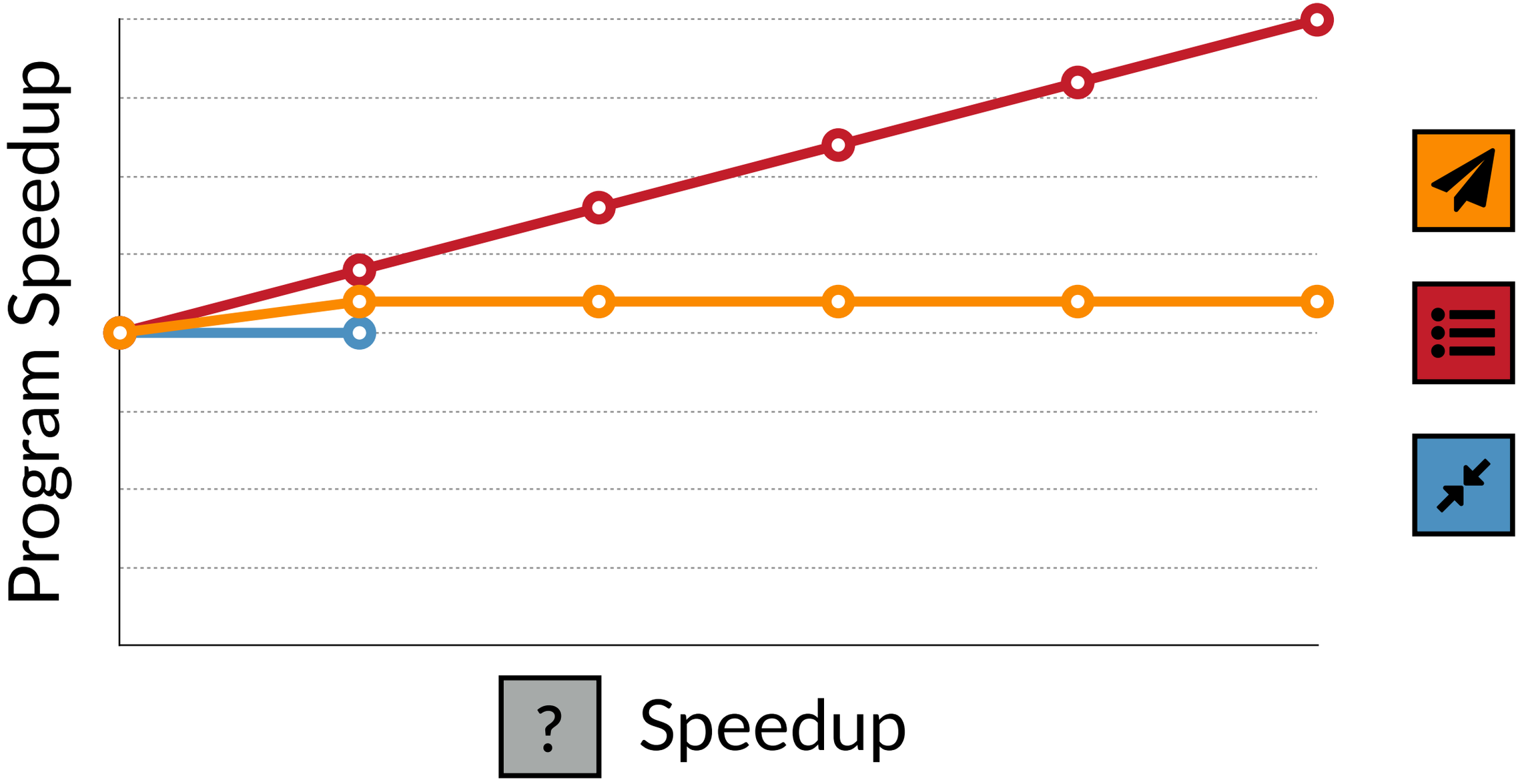
Virtual Speedup

“Speed up”  by slowing everything else down.

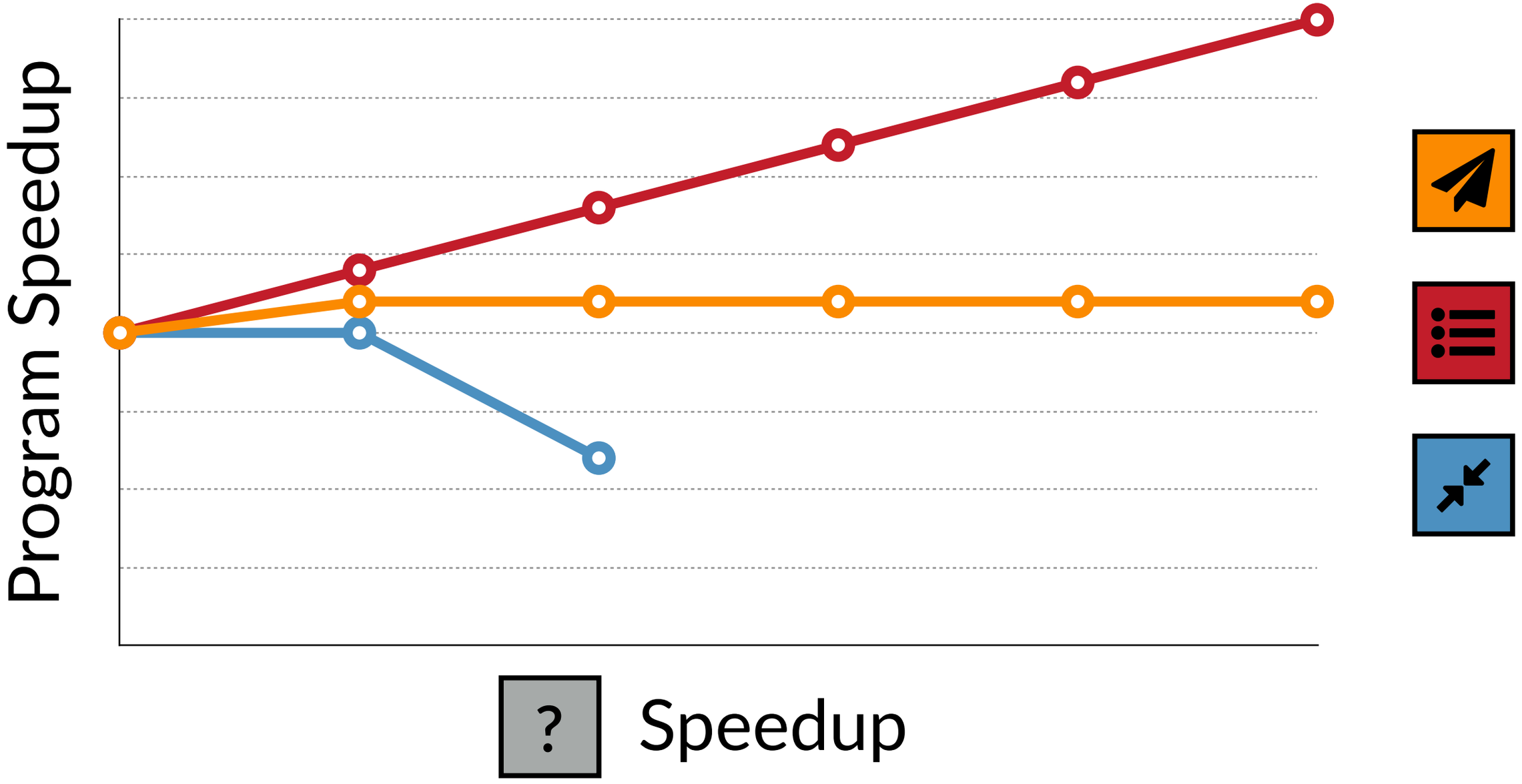


Each time  runs, pause all other threads.

Speedup Results

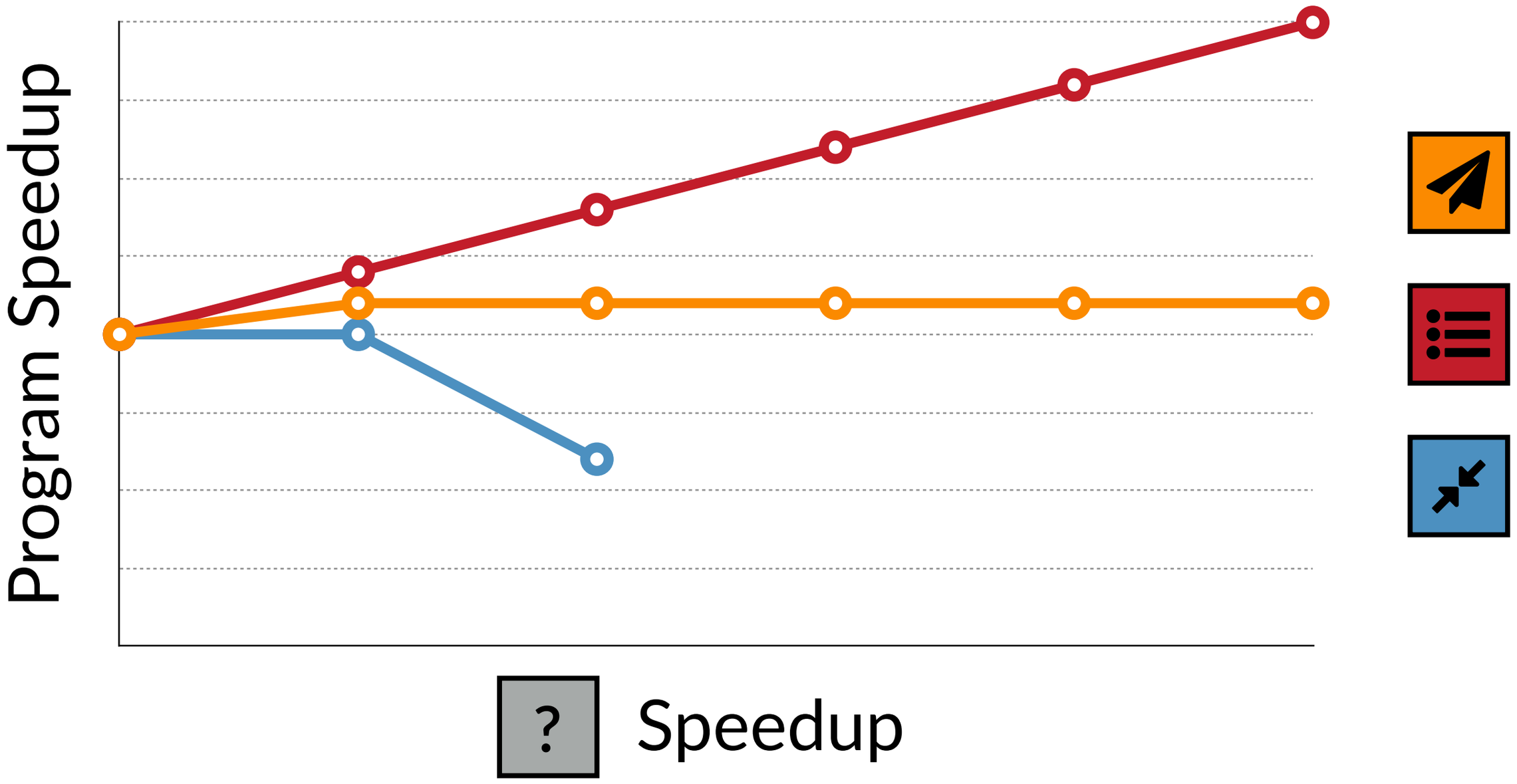


Speedup Results

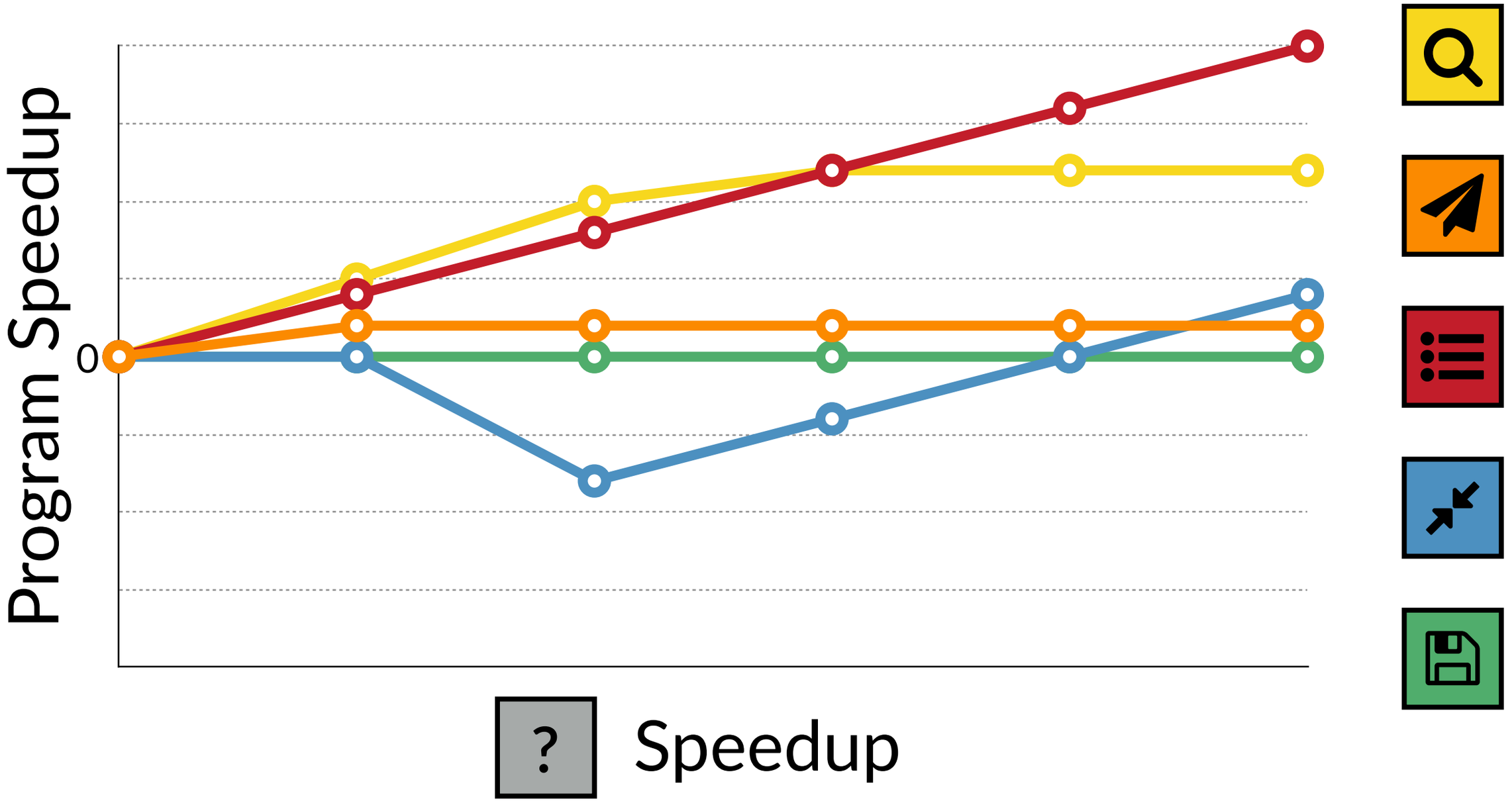


Speedup Results

Speeding up [shrink] slows the program down!



Speedup Results



Is runtime meaningful?

Take a picture

Send it to Ogle

Add it to the database

found 8,000,000 similar images

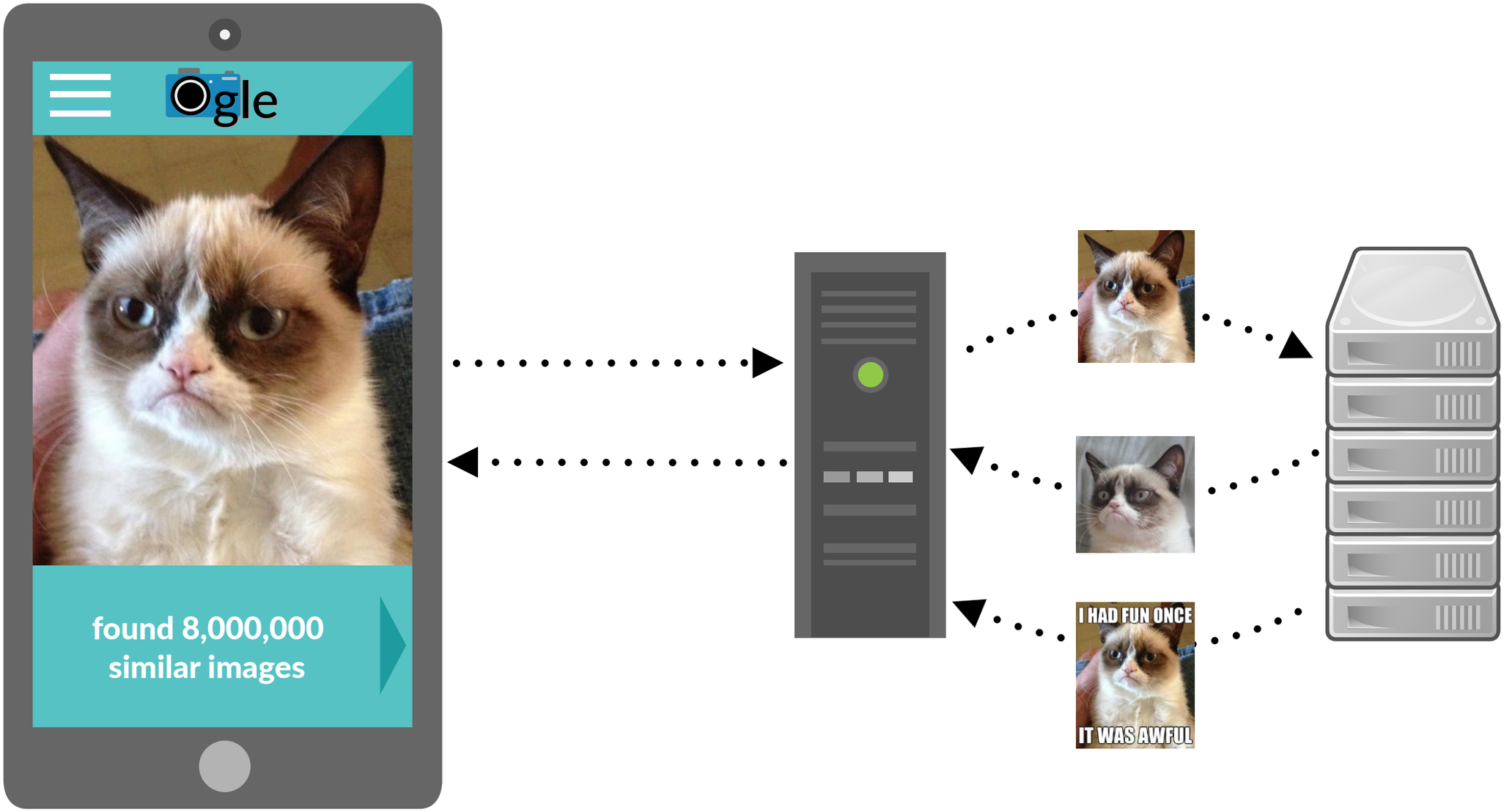
I HAD FUN ONCE

IT WAS AWFUL

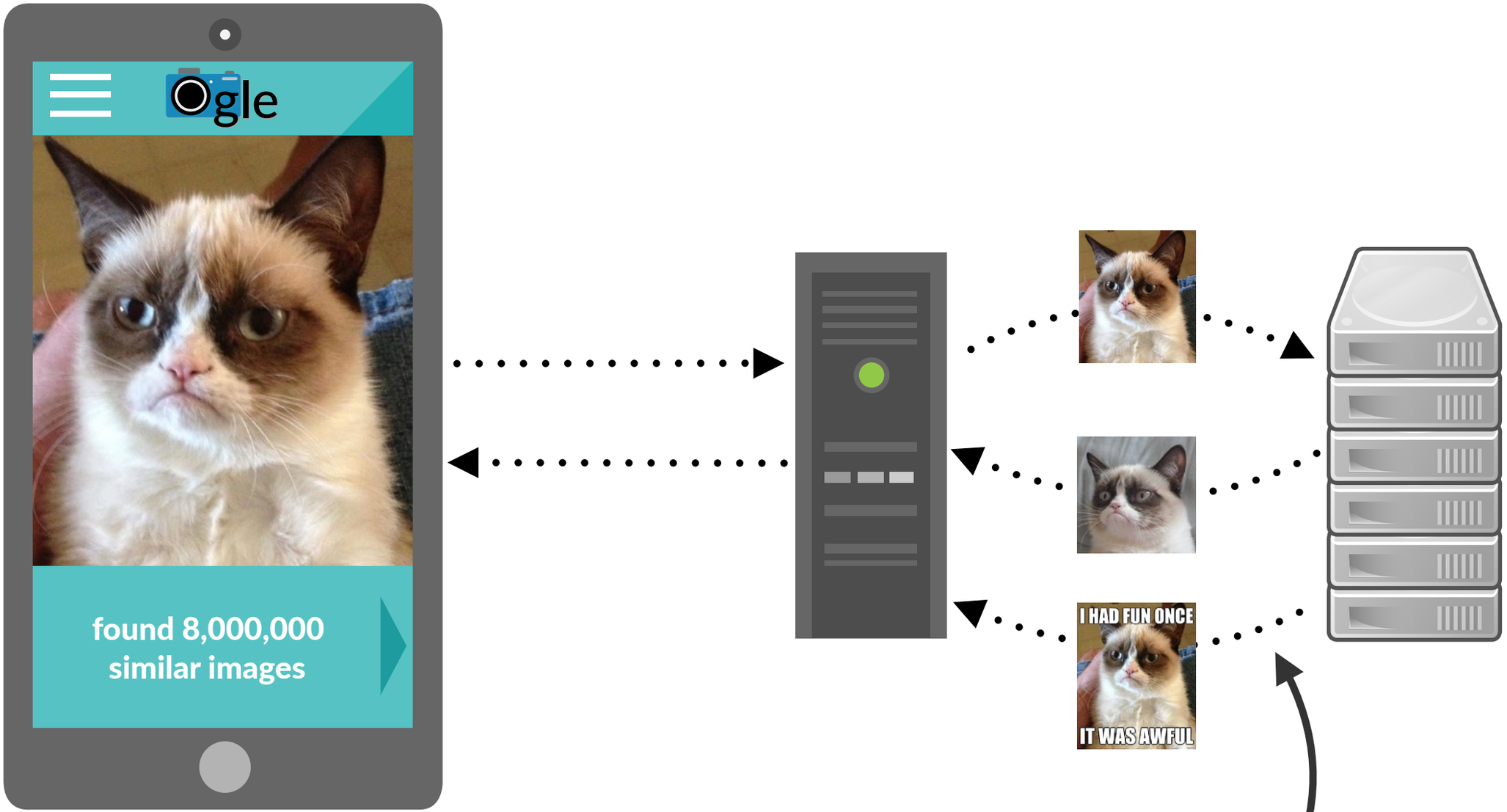
Send results

Find similar pictures

Is runtime meaningful?

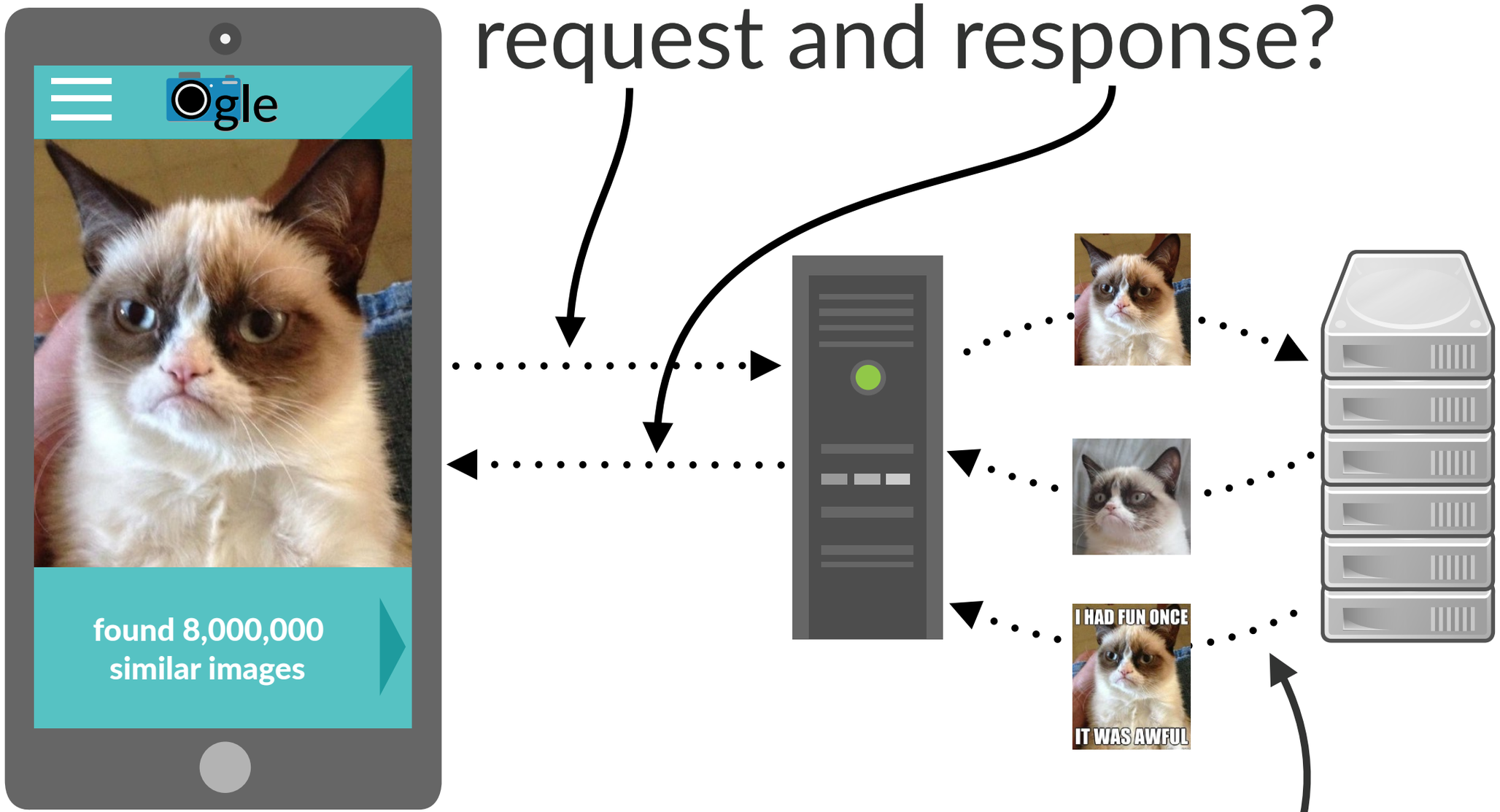


Is runtime meaningful?

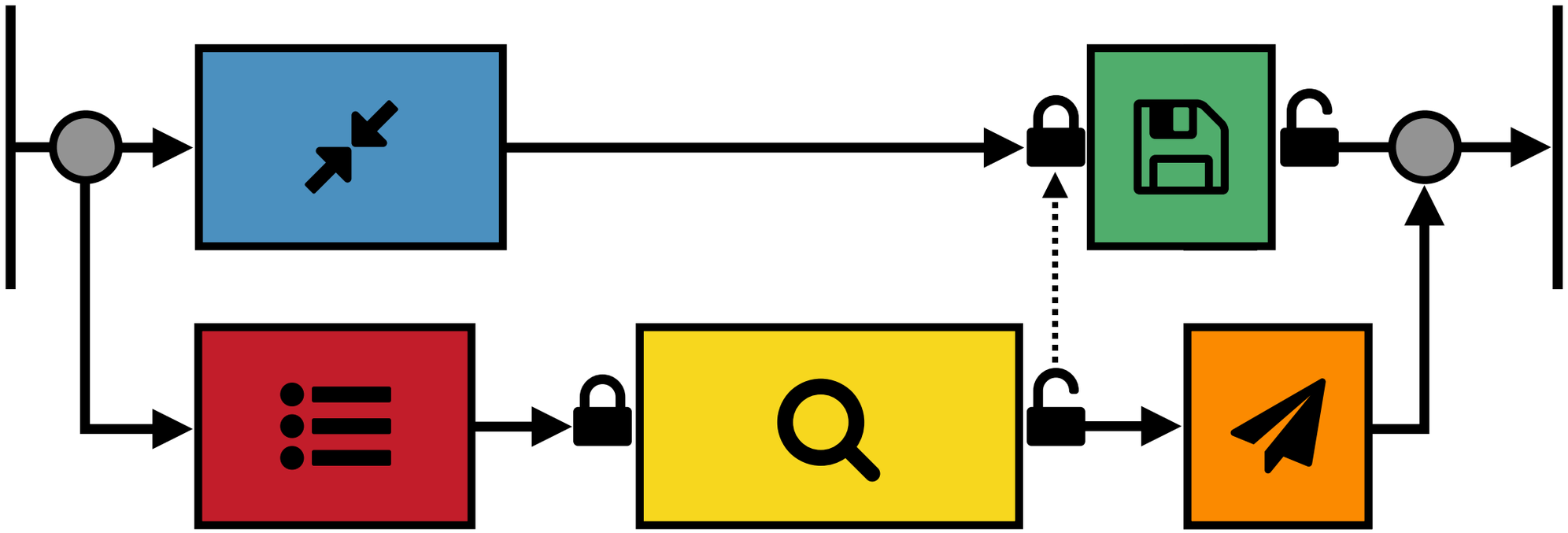


How fast do results come back?

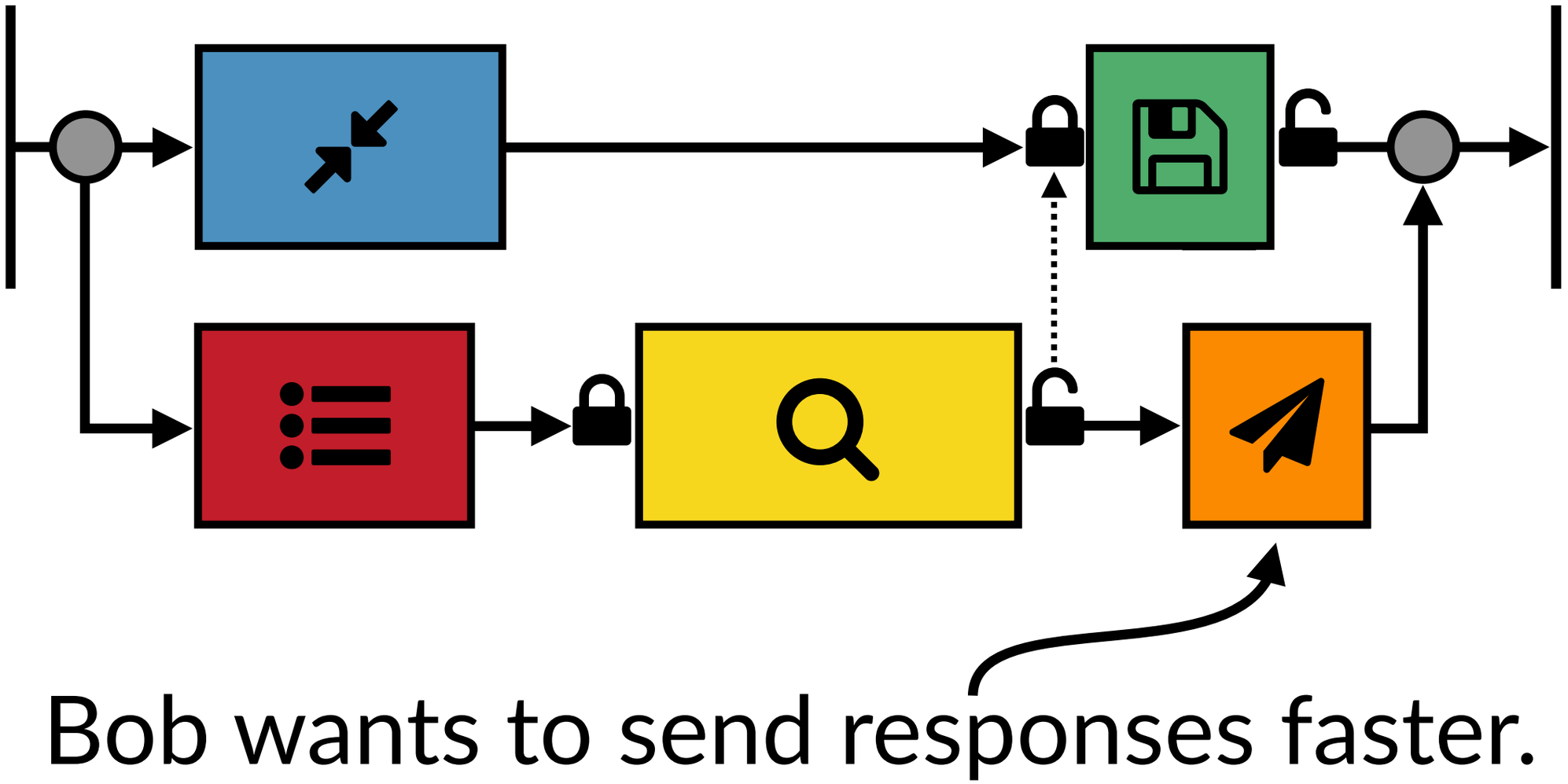
Is runtime meaningful?



Progress Points

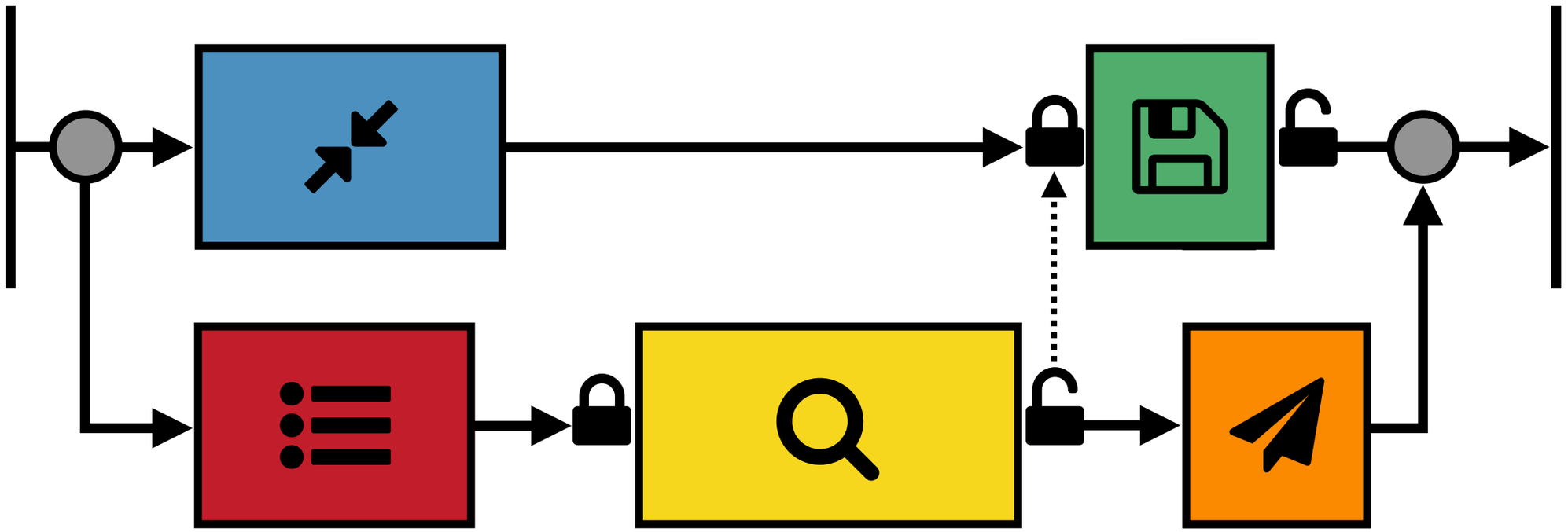


Progress Points



Bob wants to send responses faster.

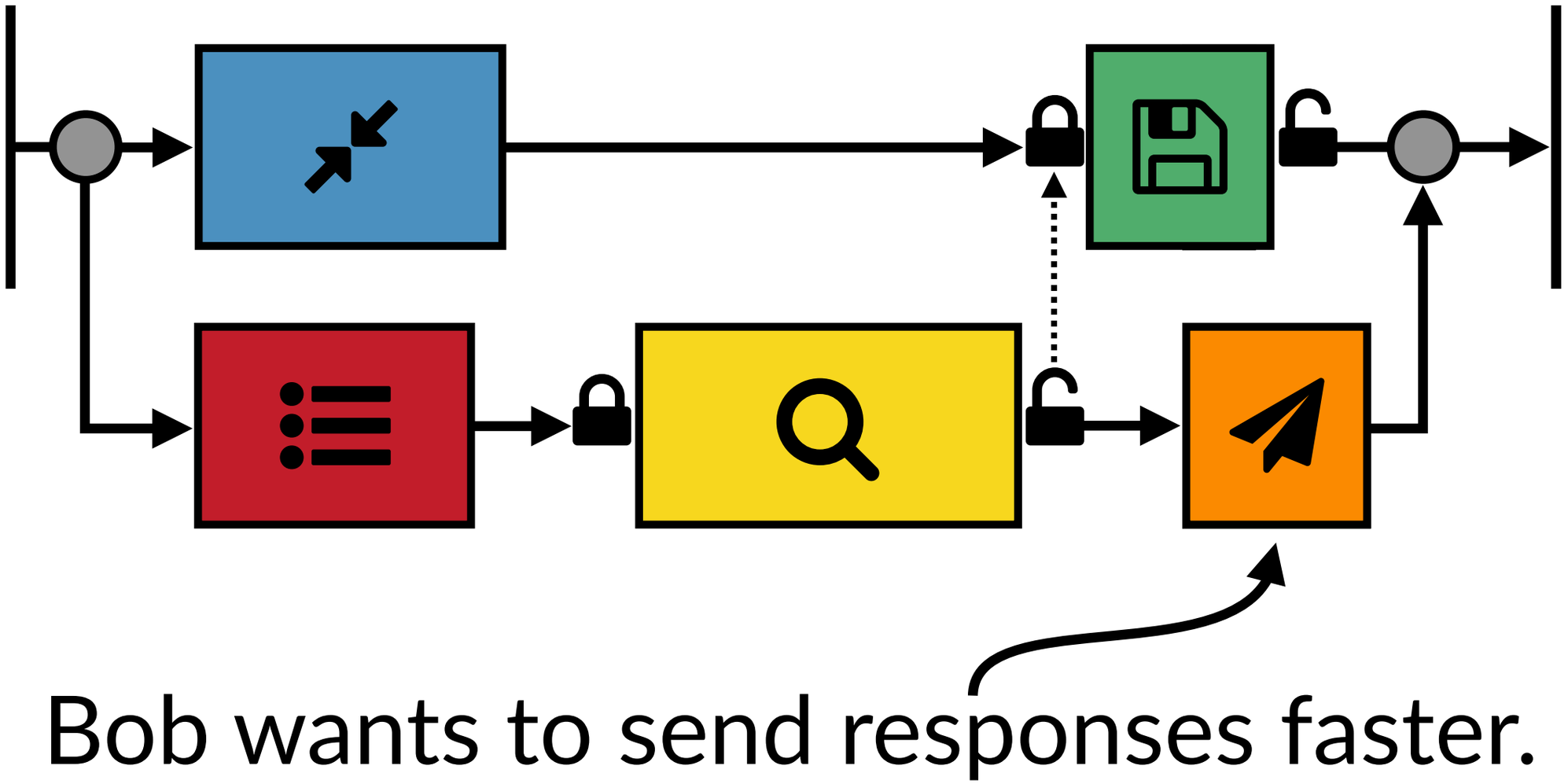
Progress Points



Bob wants to send responses faster.

He marks  as a *progress point*.

Progress Points

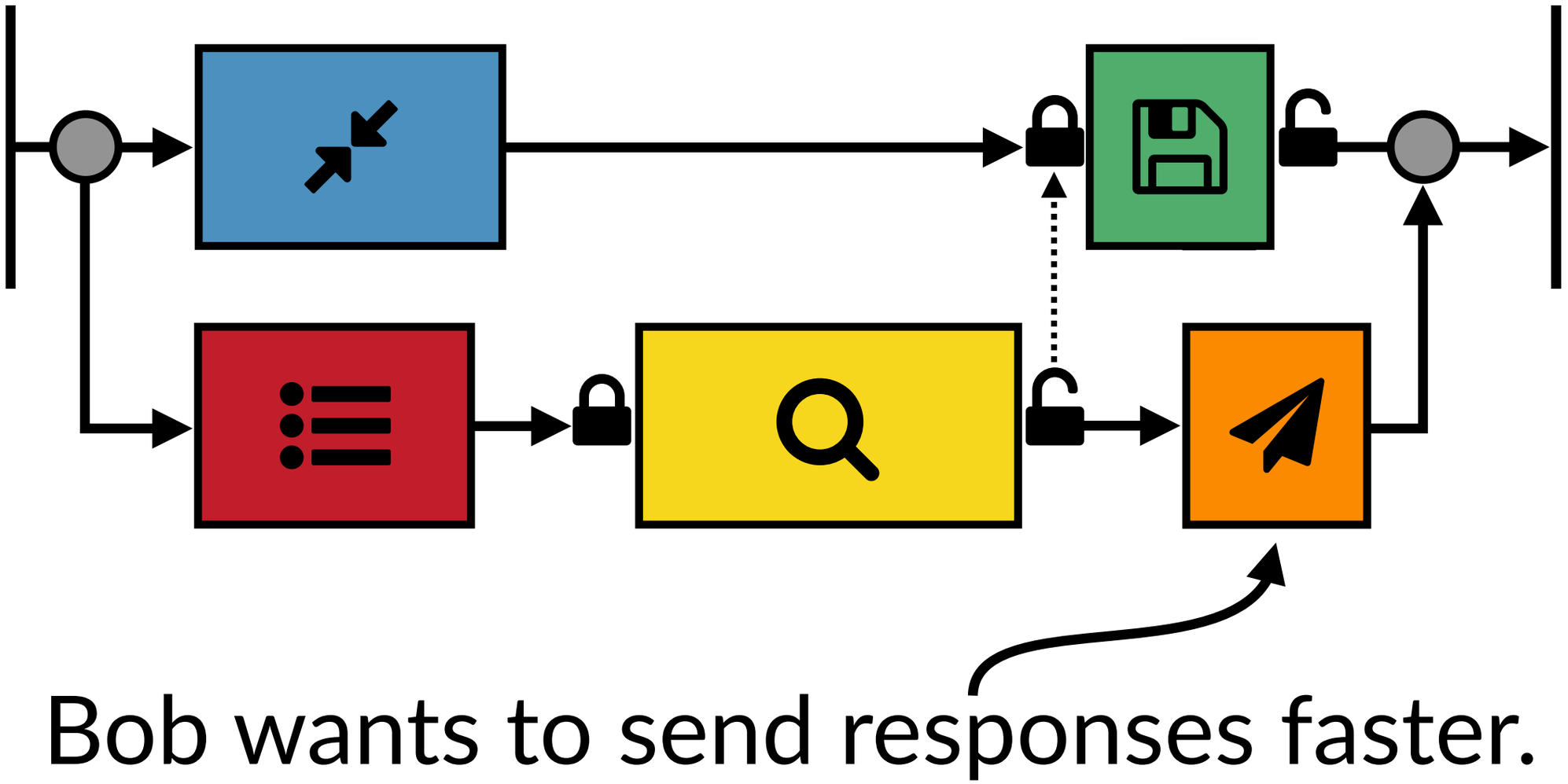


Bob wants to send responses faster.

He marks  as a *progress point*.

Each time this code runs…

Progress Points

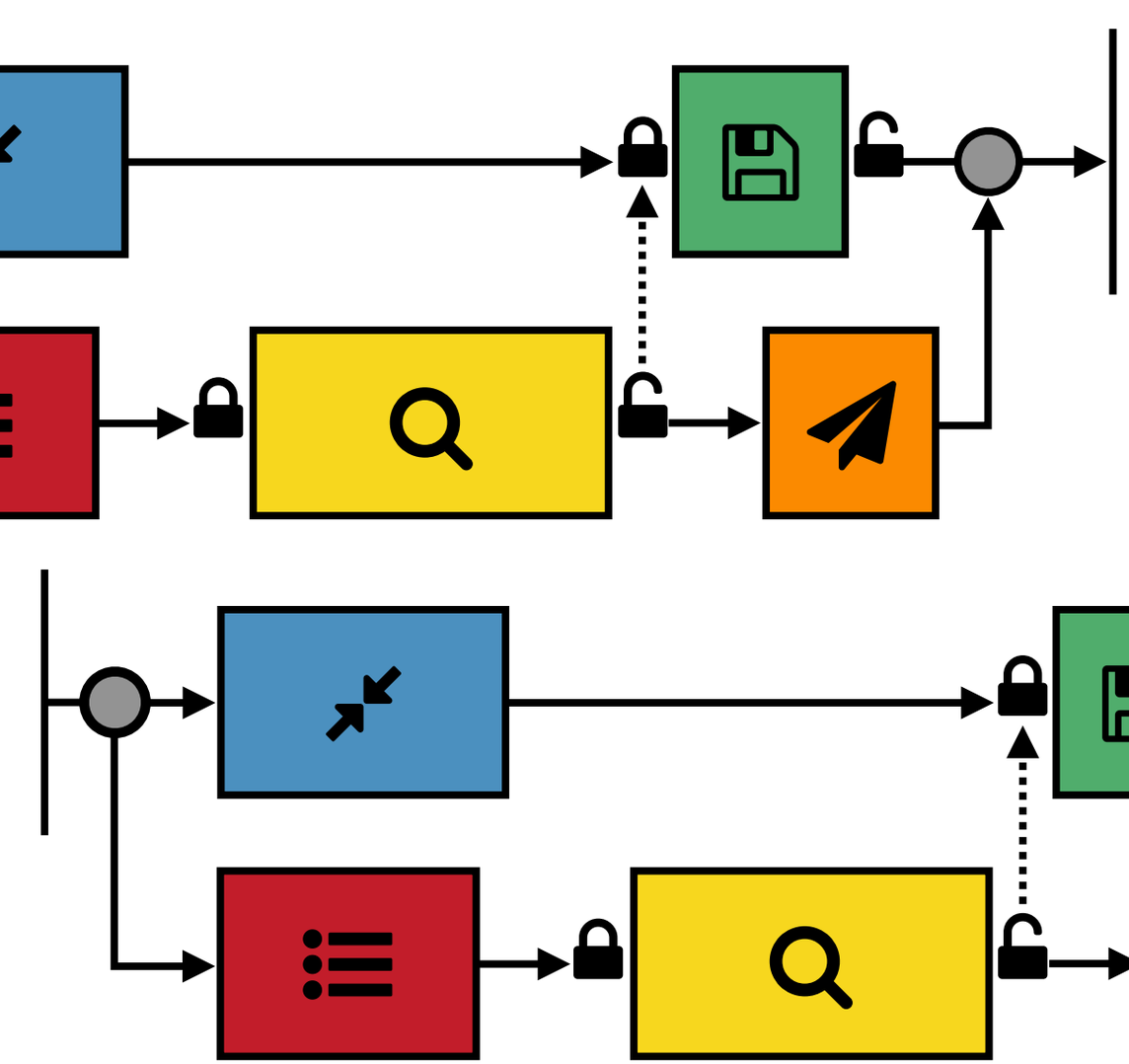


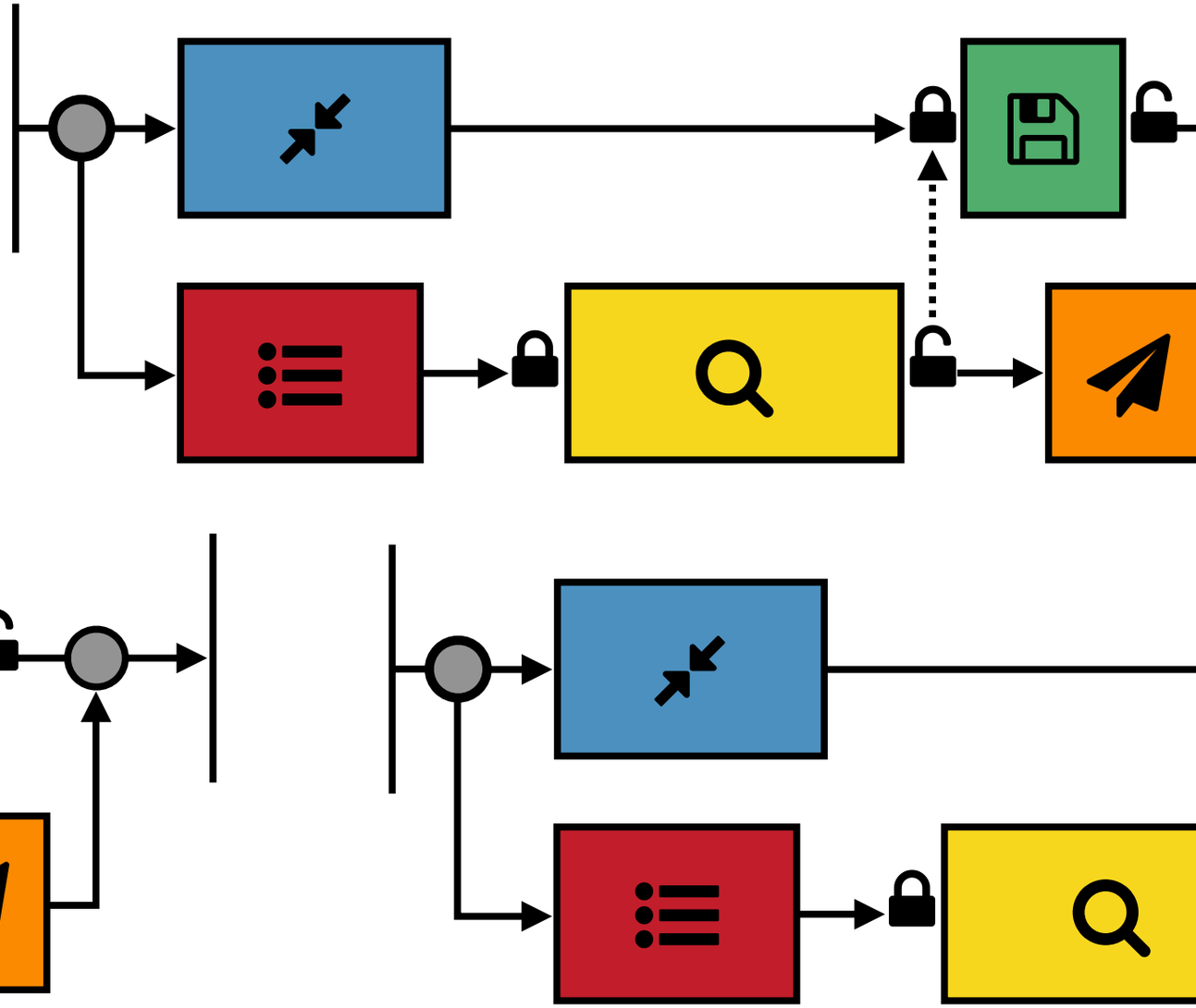
Bob wants to send responses faster.

He marks  as a *progress point*.

Each time this code runs…

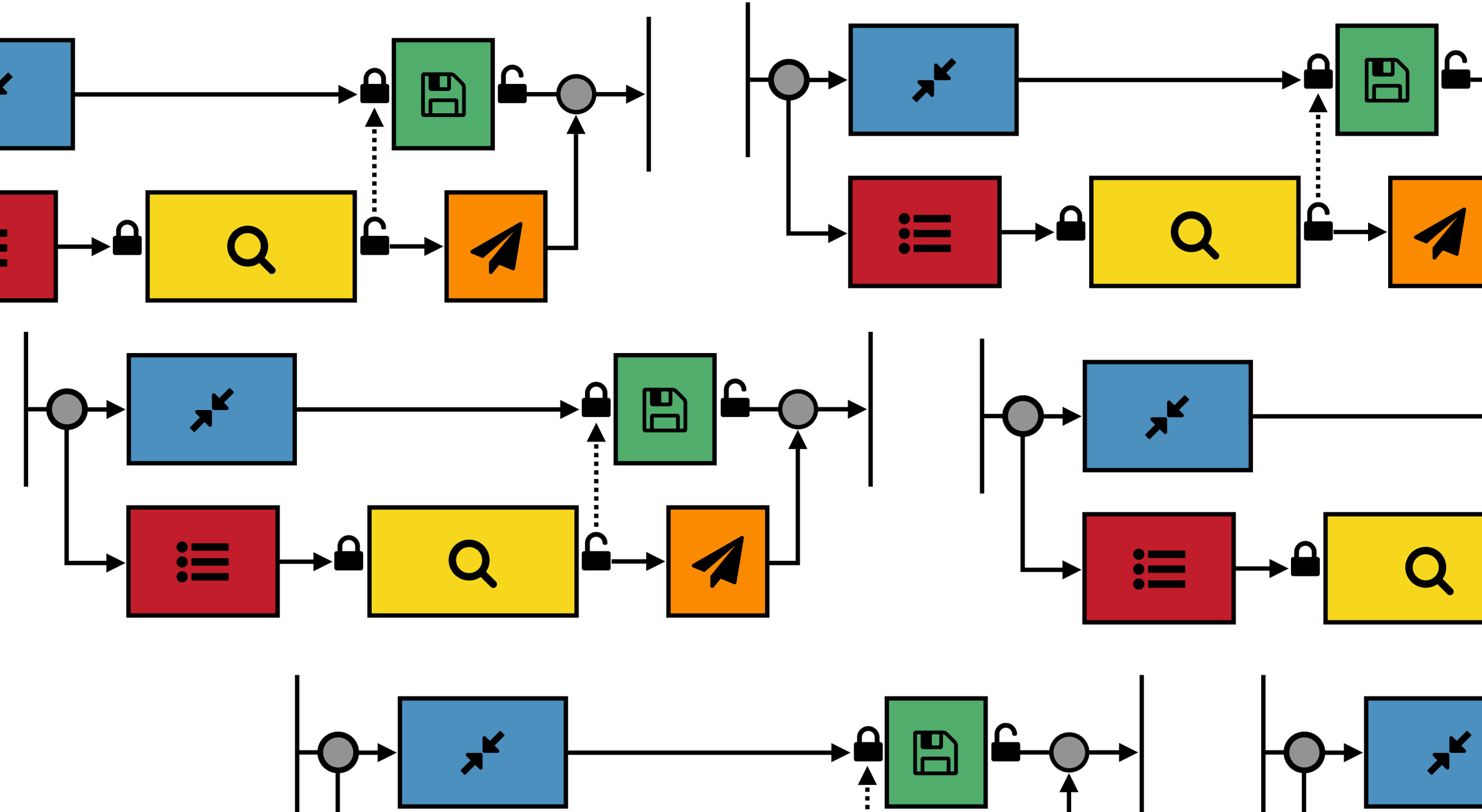
Progress Points





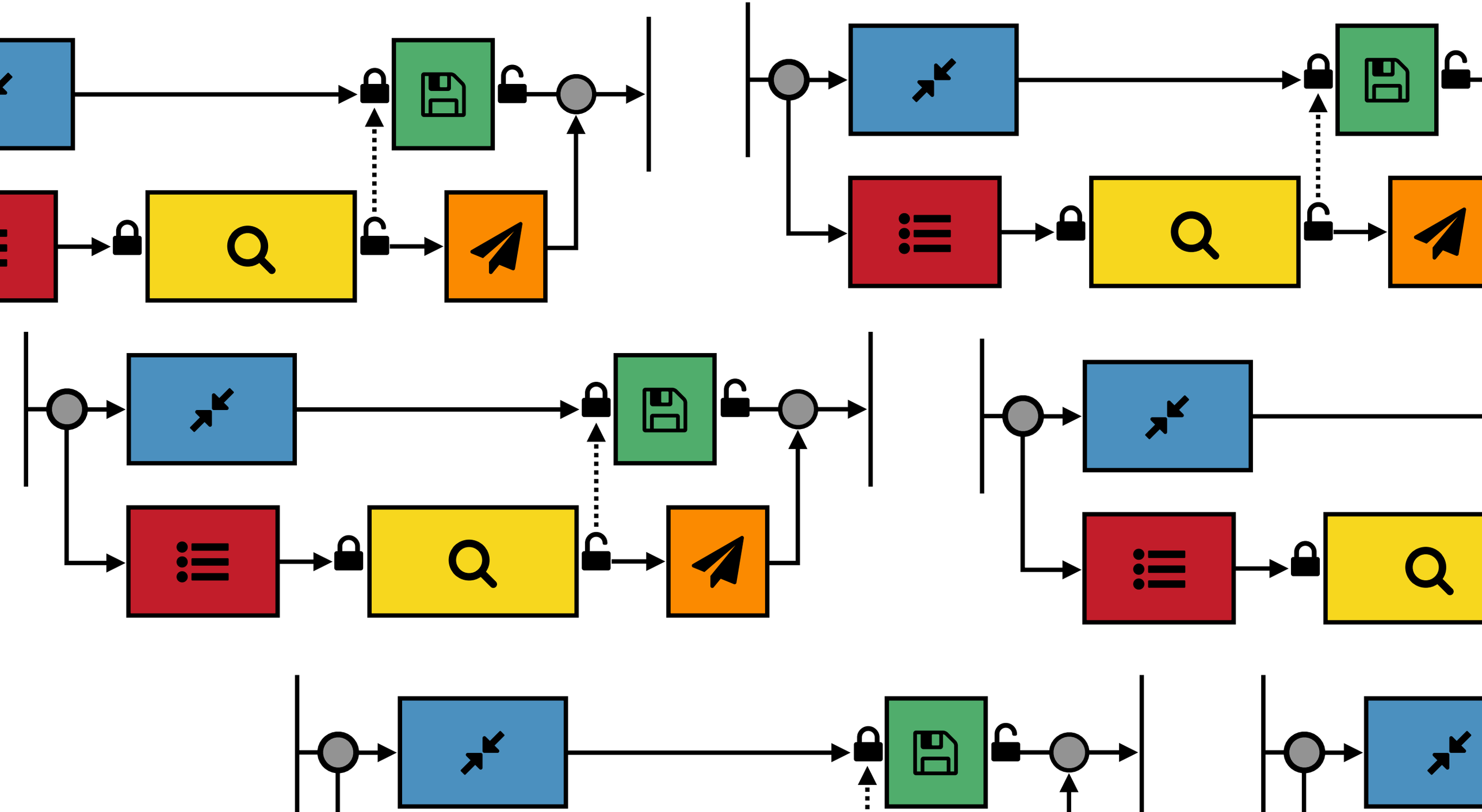
Progress Points

Many requests running for many users.

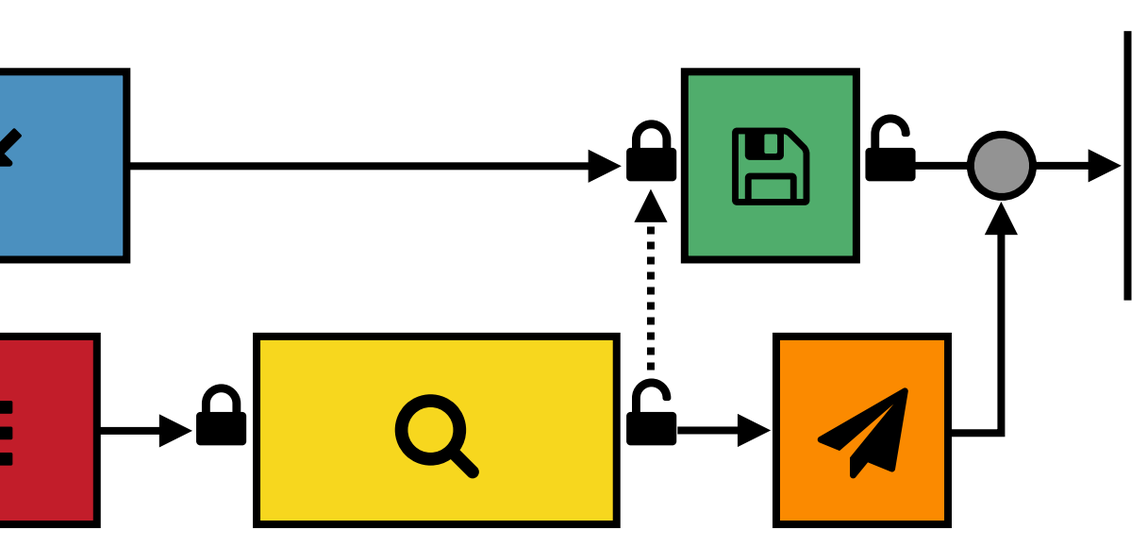


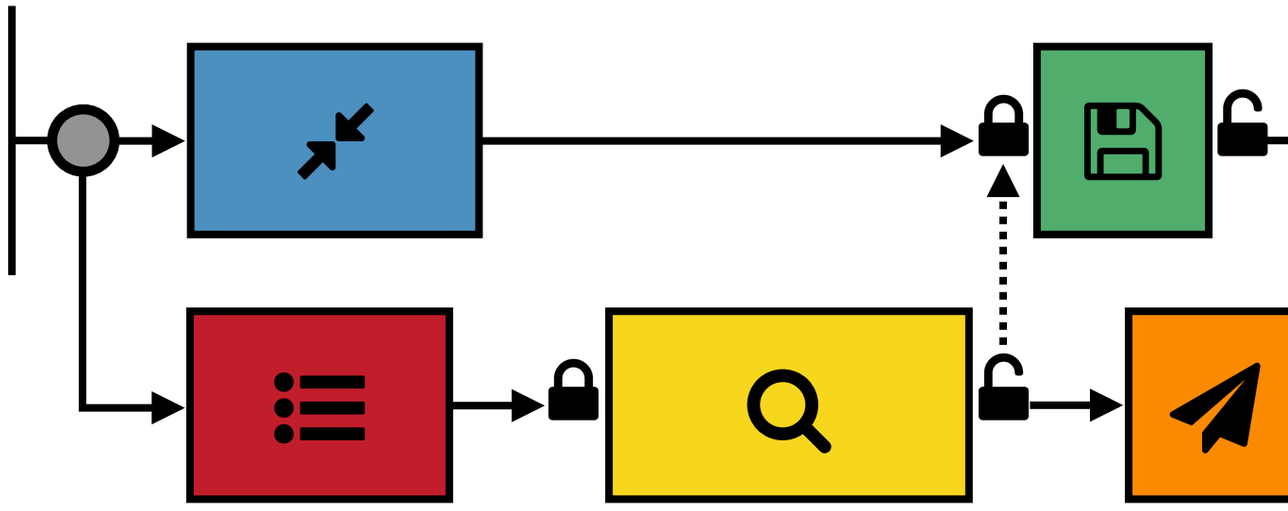
Progress Points

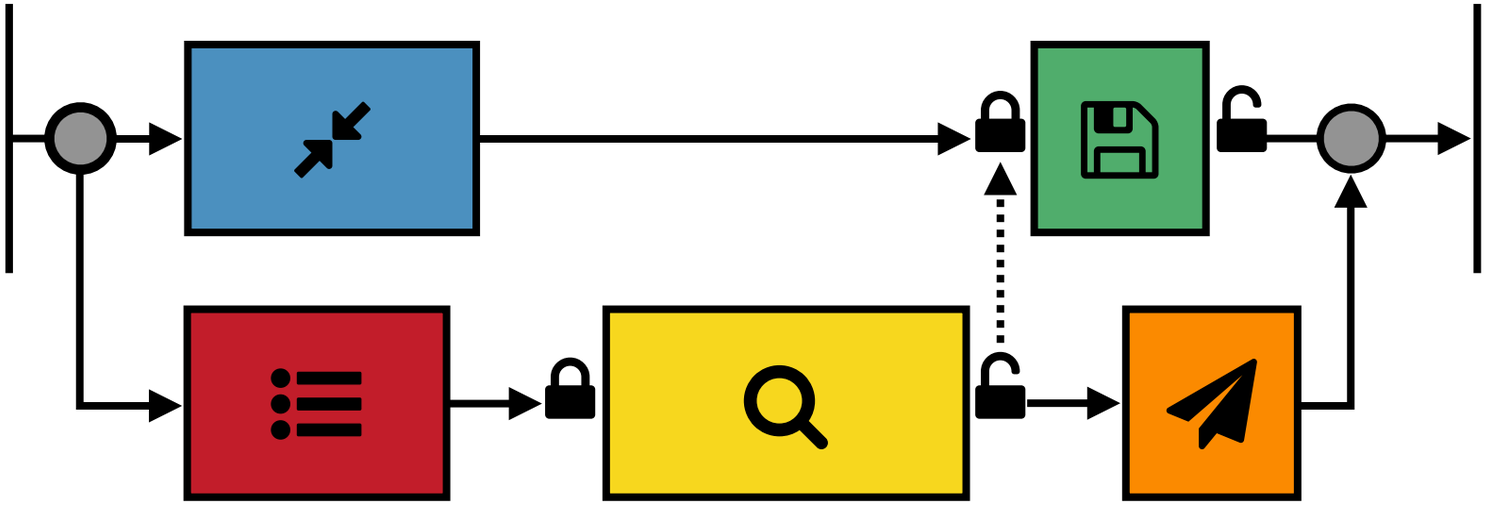
Many requests running for many users.

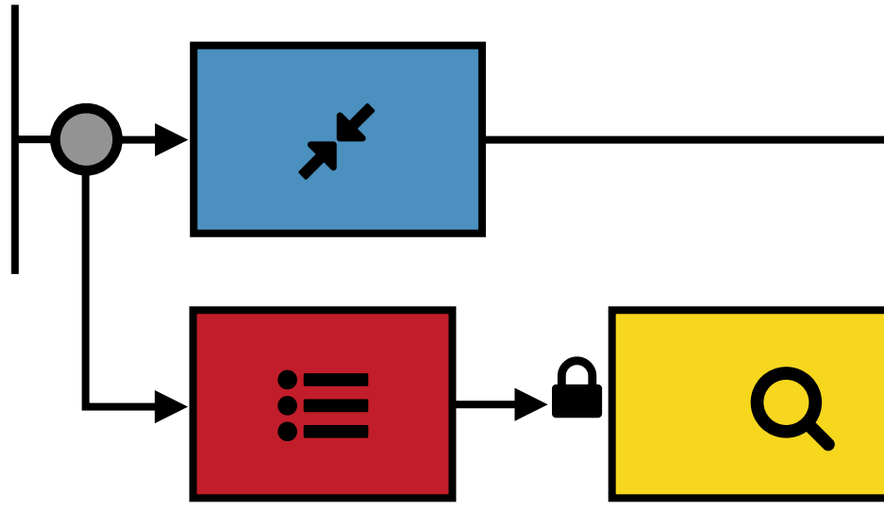


Progress Points



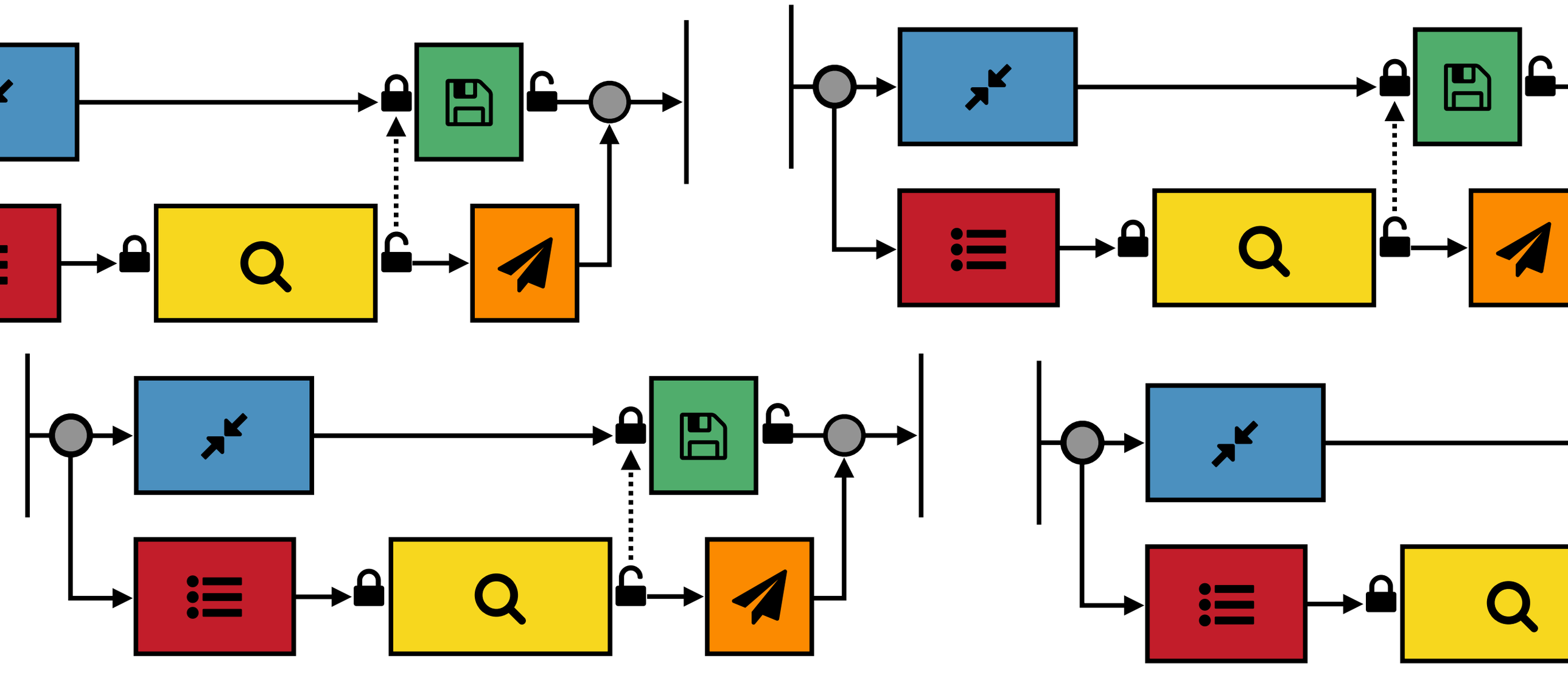






Progress Points

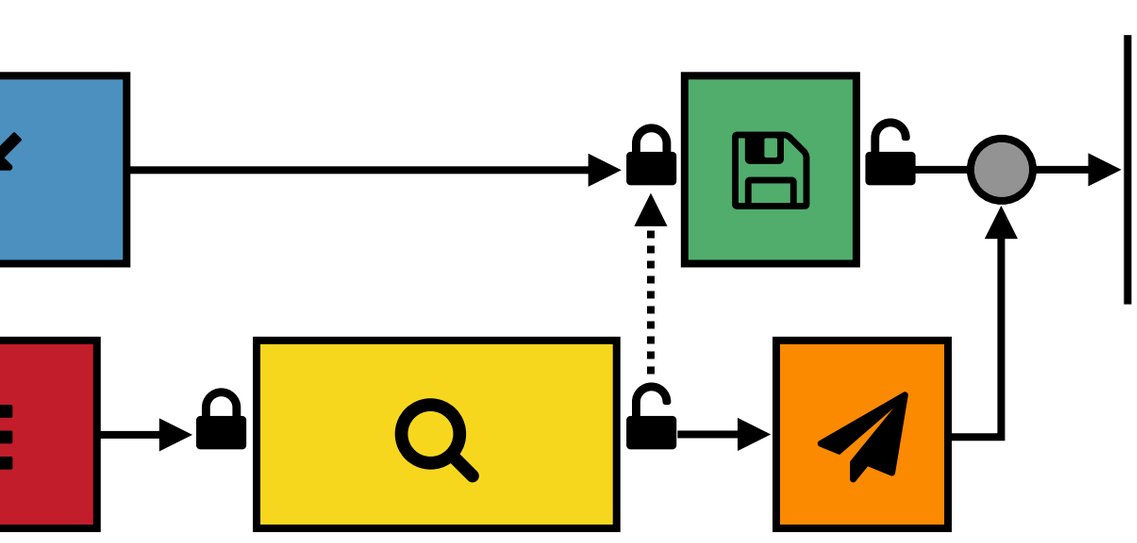
One progress point measures throughput.

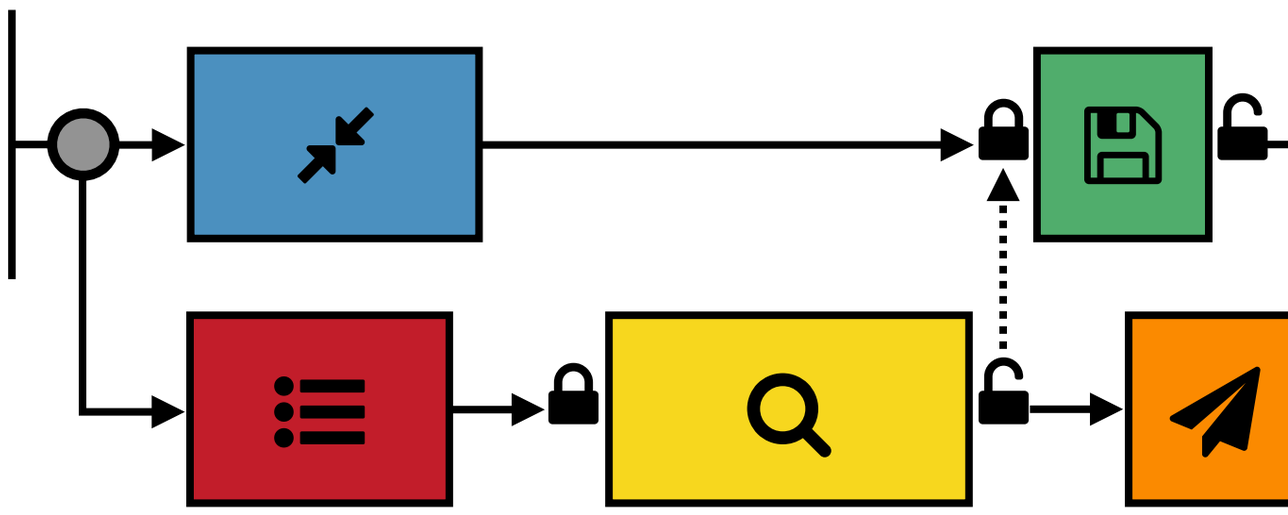


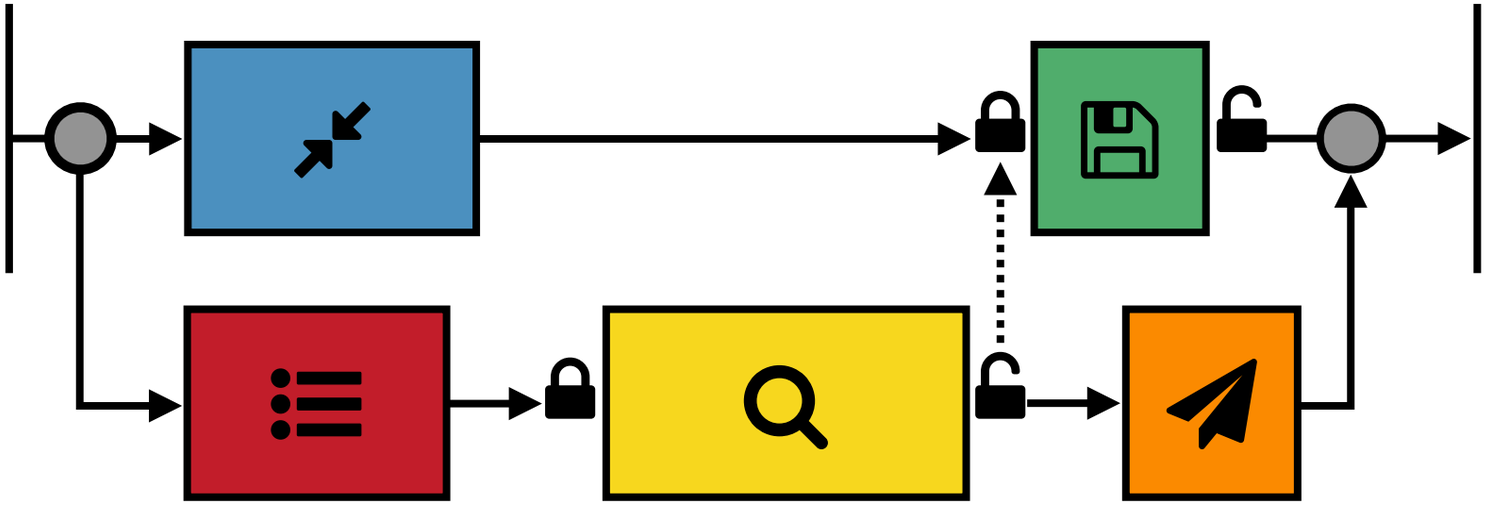
Progress Points

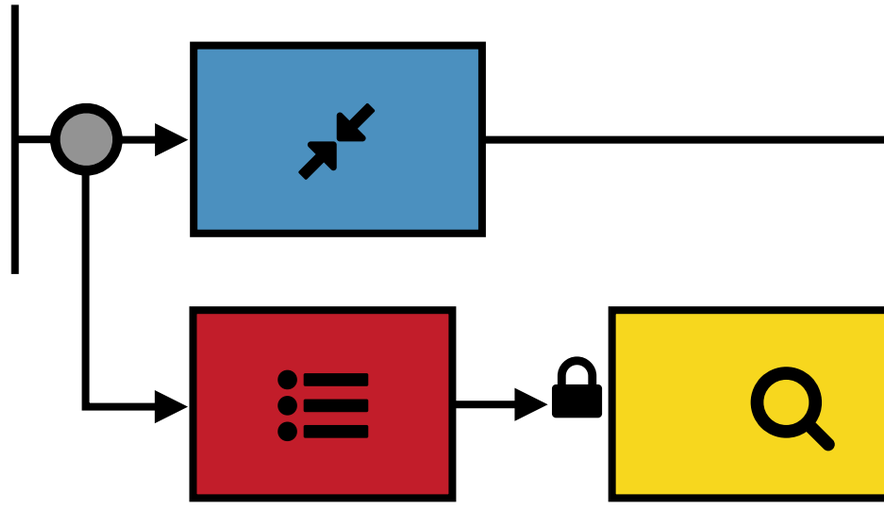
One progress point measures throughput.

If I speed up ?, how much faster do I run ?

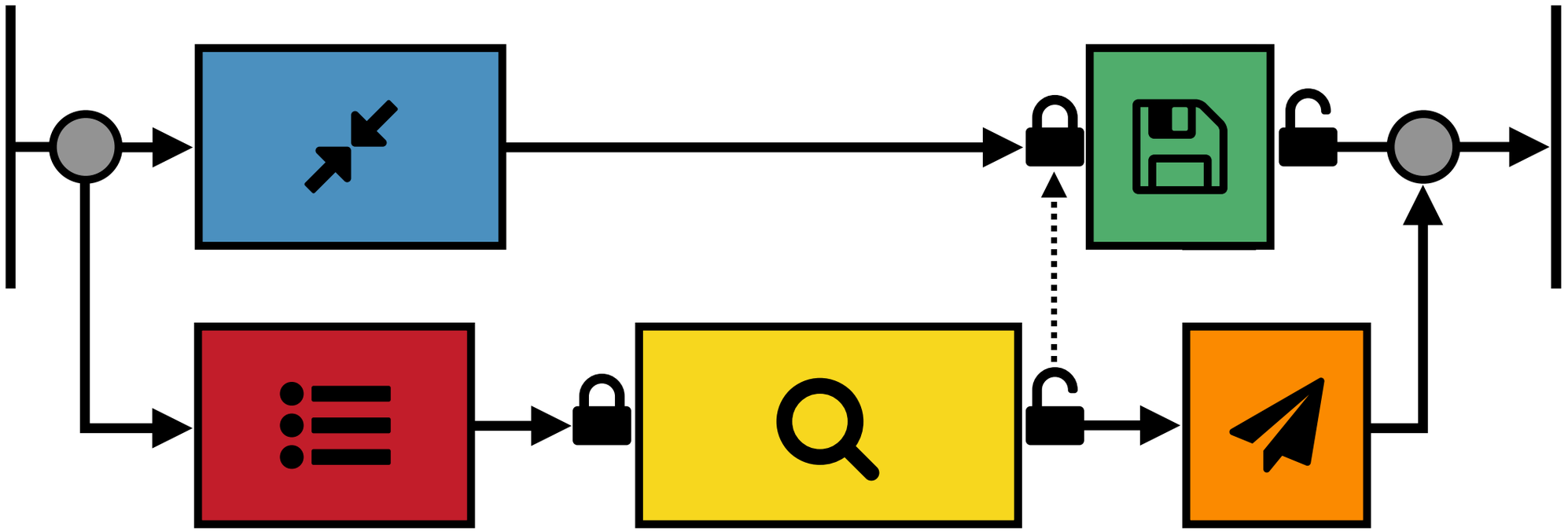




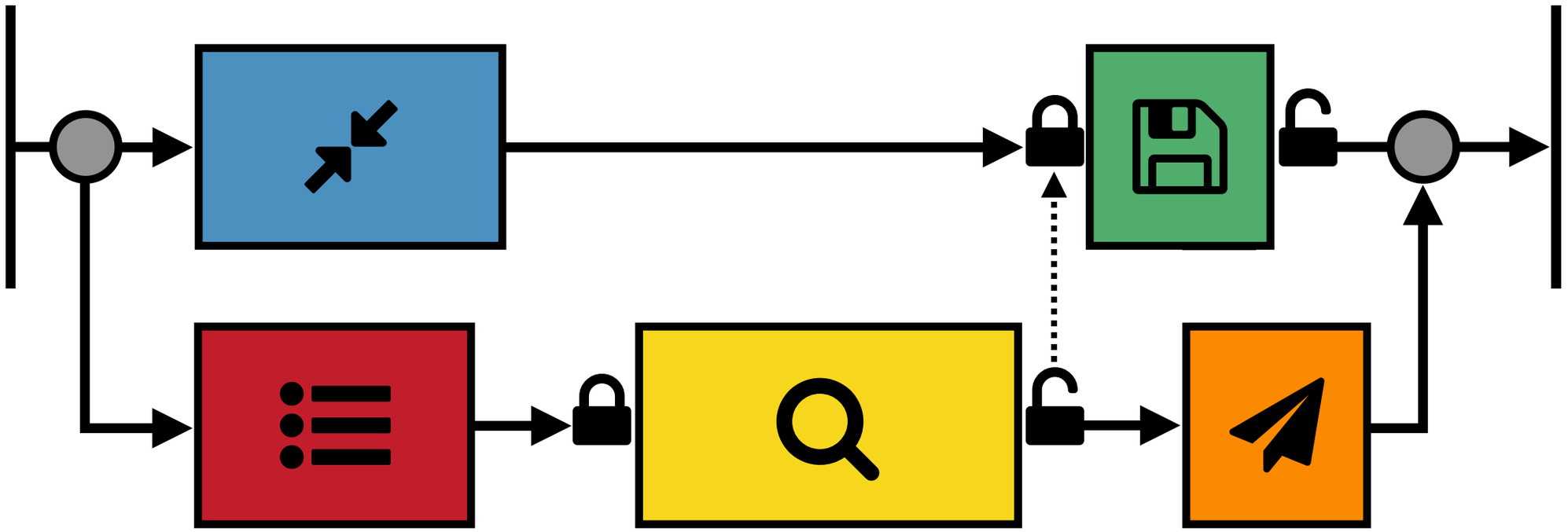




Progress Points

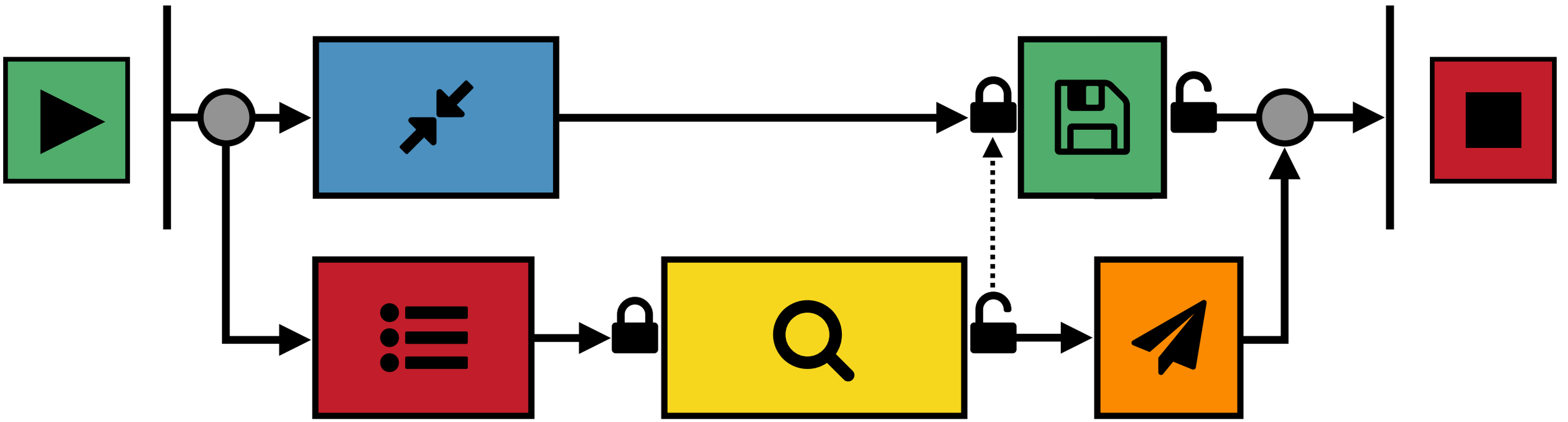


Progress Points



Bob wants to minimize response time.

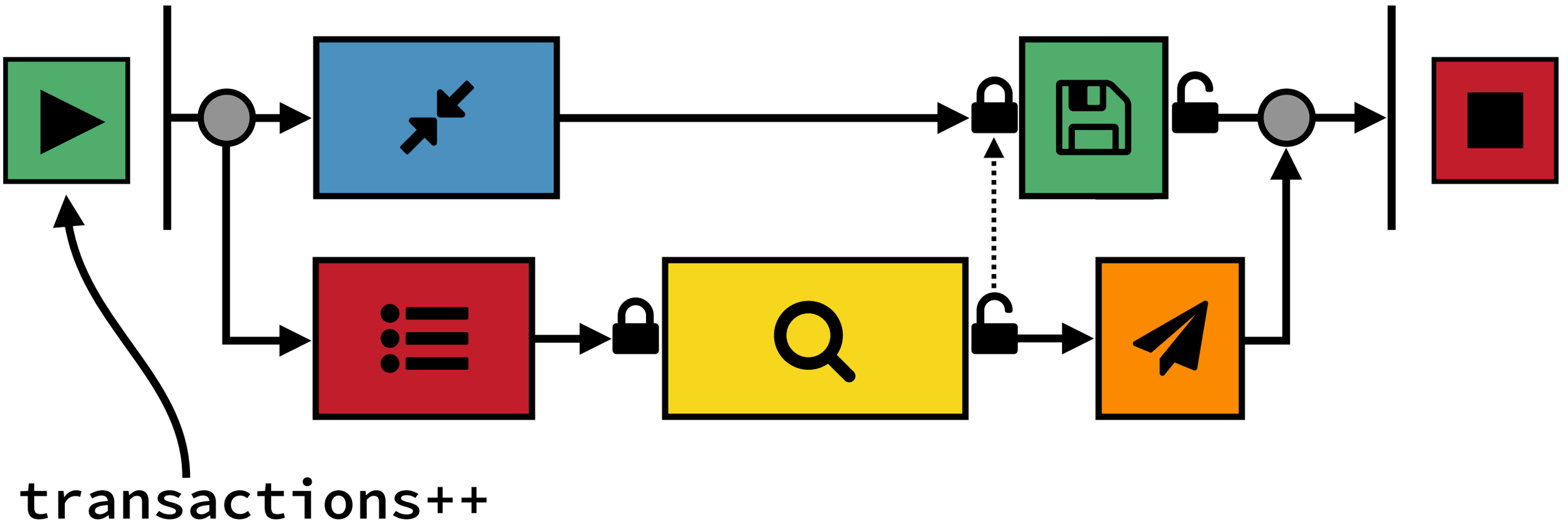
Progress Points



Bob wants to minimize response time.

He adds *latency progress points*.

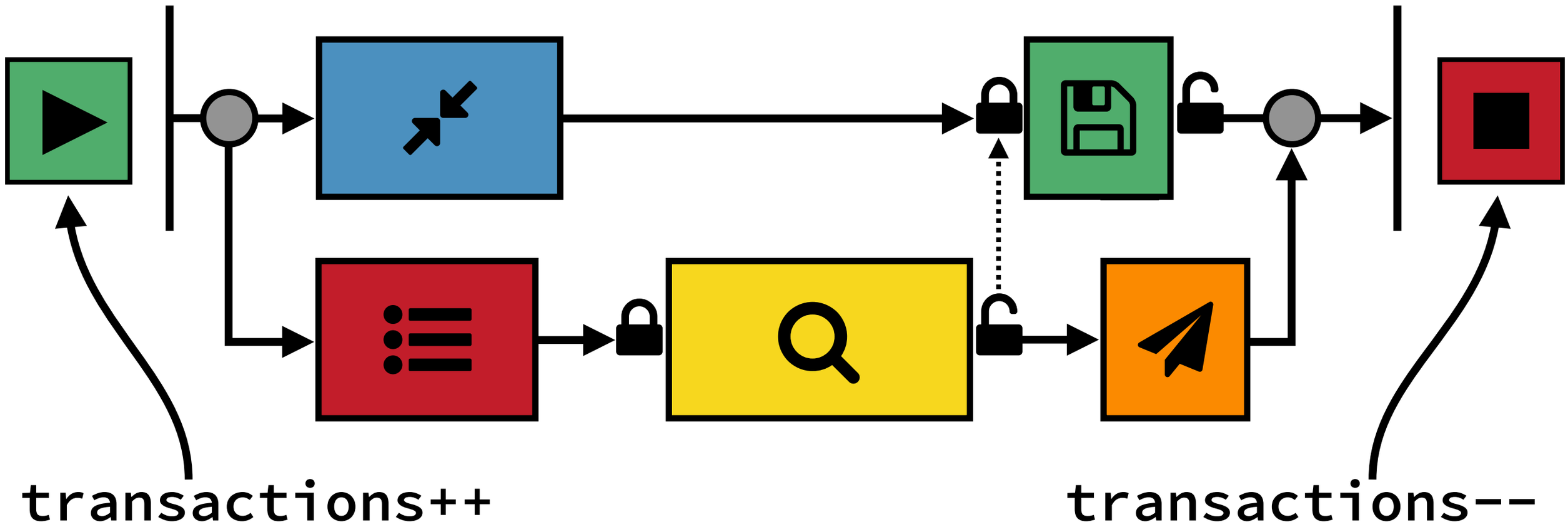
Progress Points



Bob wants to minimize response time.

He adds *latency progress points*.

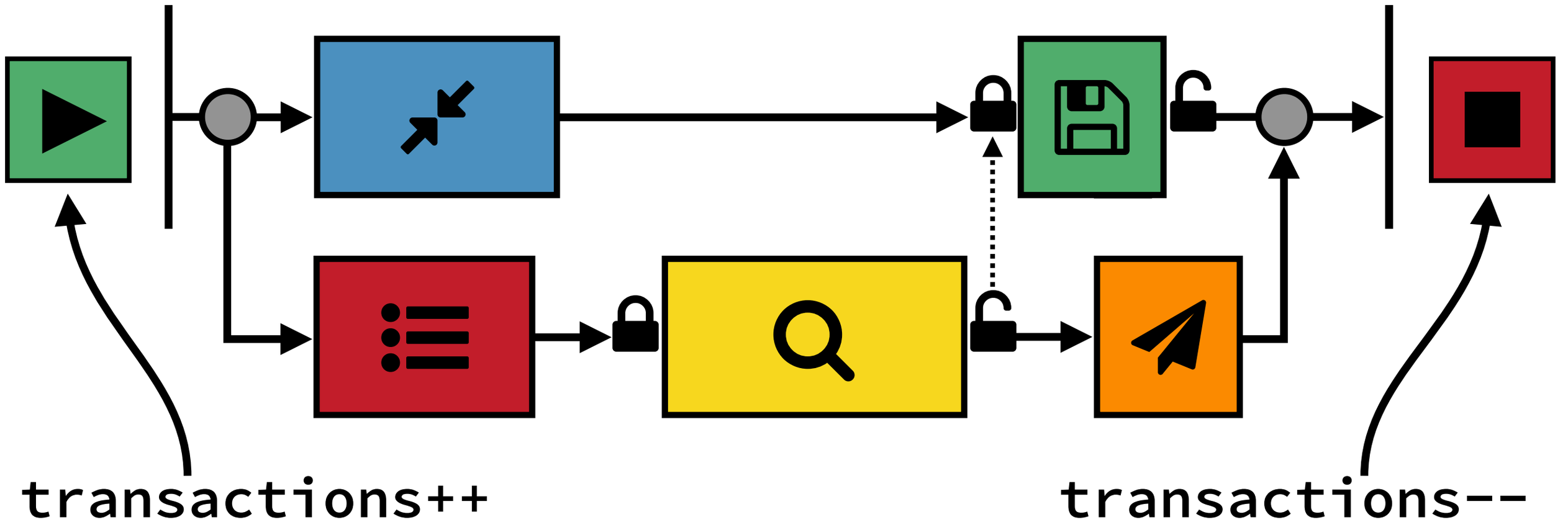
Progress Points



Bob wants to minimize response time.

He adds *latency progress points*.

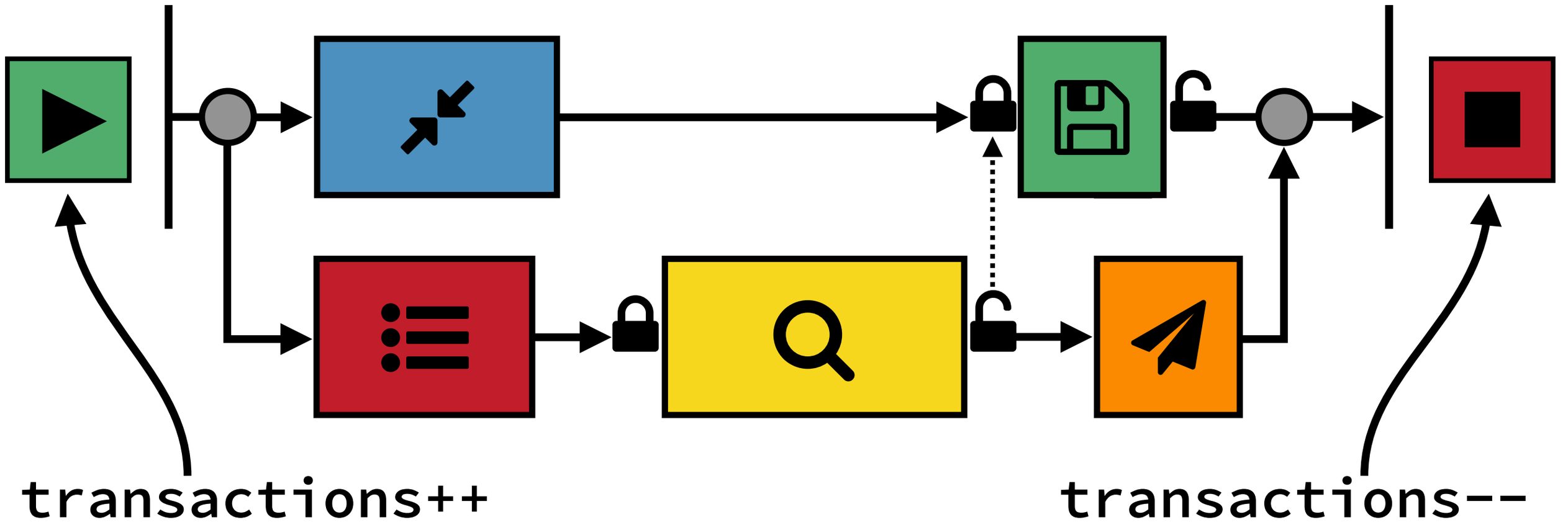
Progress Points



Bob wants to minimize response time.

Progress Points

Little's Law: W= L / λ

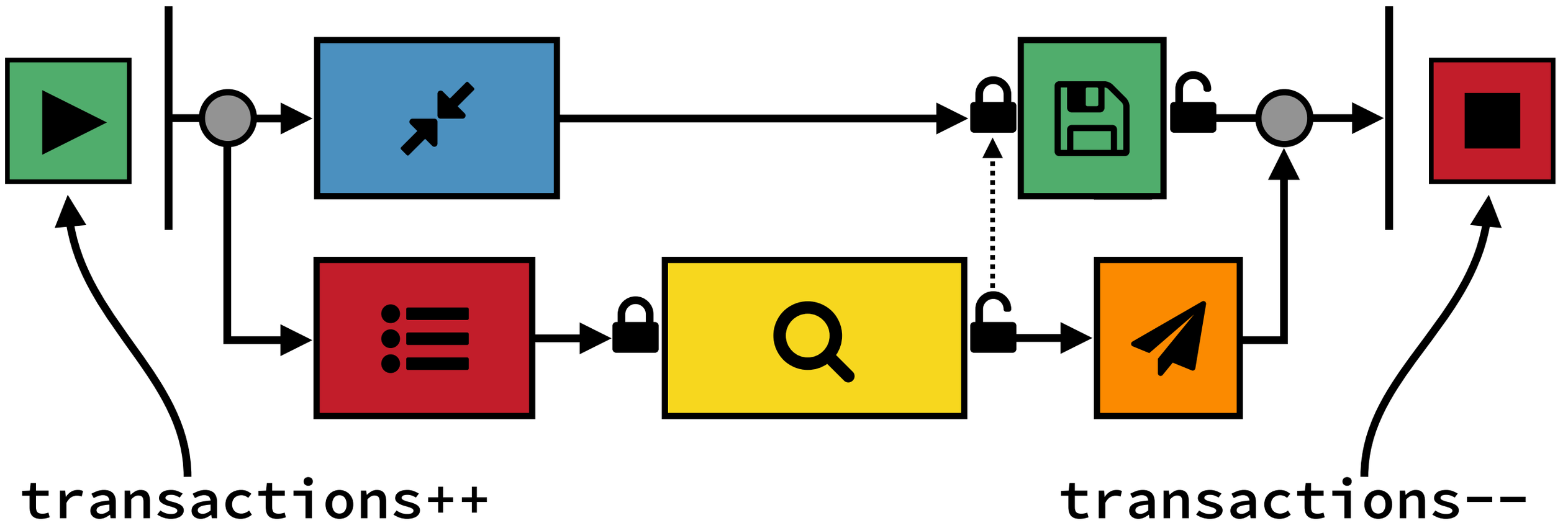


Bob wants to minimize response time.

Progress Points

Little's Law: W = L / λ

latency

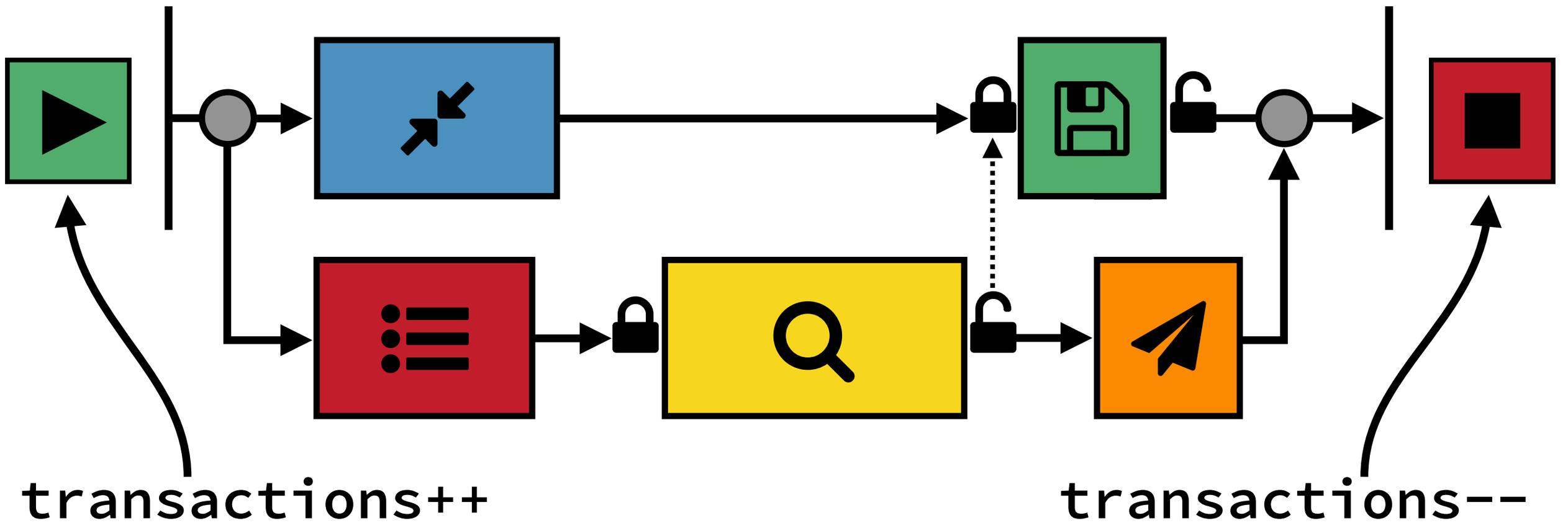


Bob wants to minimize response time.

Progress Points

Little's Law: W= L / λ

latency = `transactions`

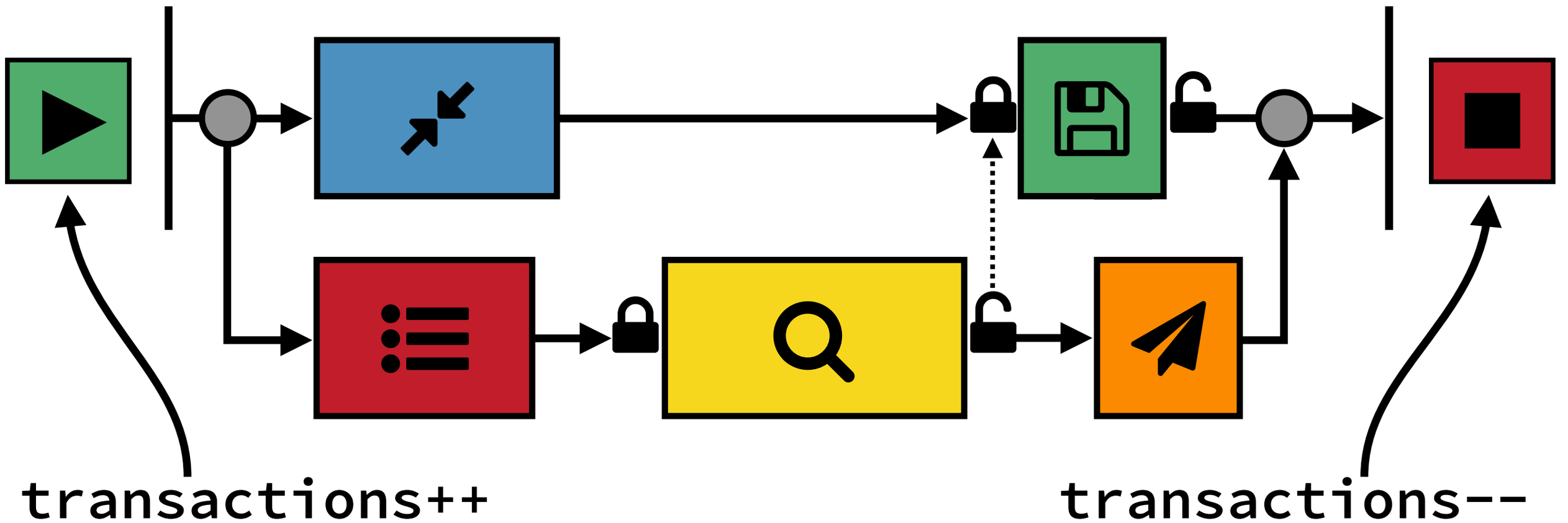


Bob wants to minimize response time.

Progress Points

Little's Law: W= L / λ

latency = `transactions` / throughput



Bob wants to minimize response time.

Coz: a Causal Profiler for Linux

```
>
```

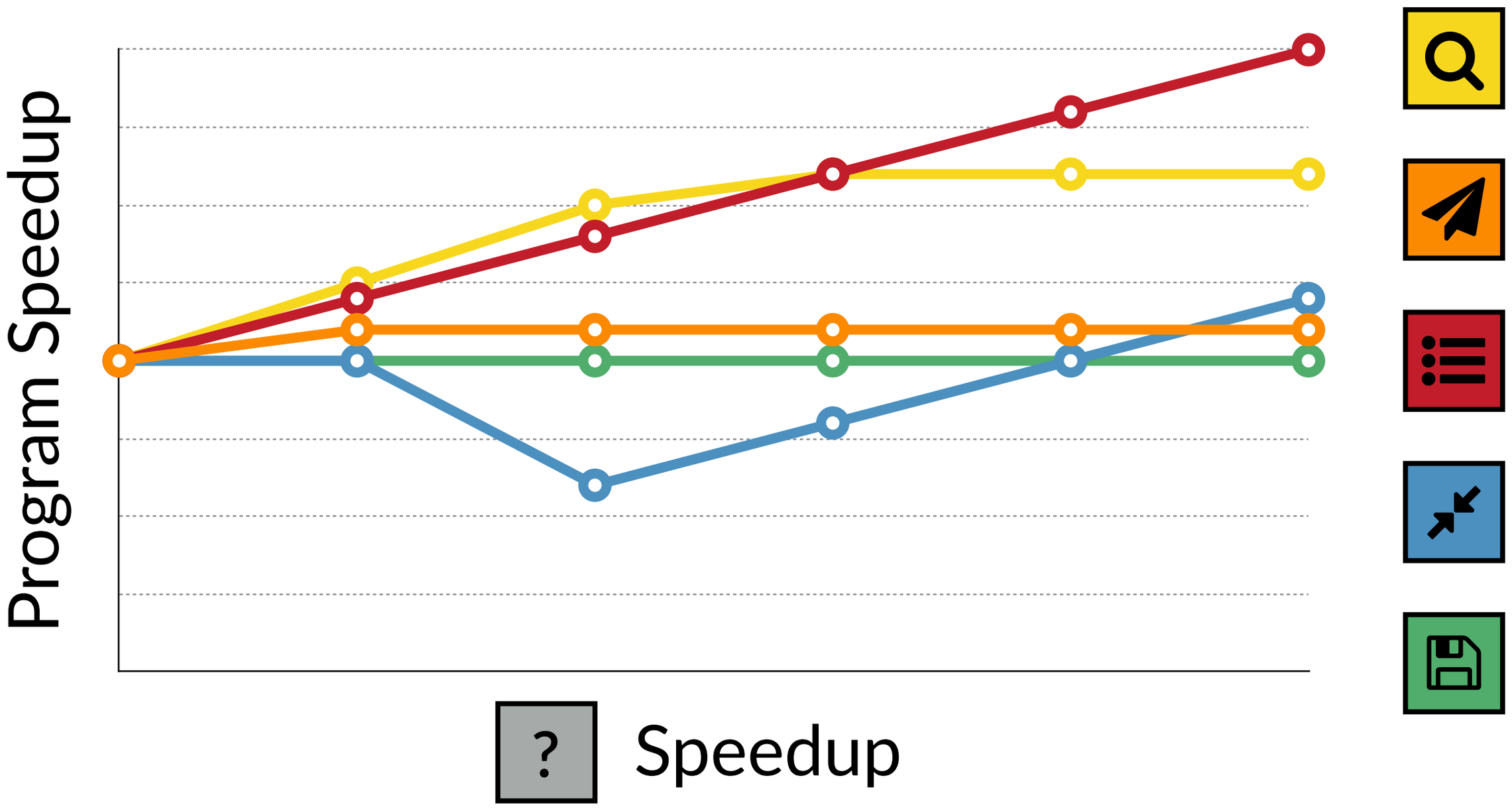
(coming to Debian, in Unstable now)

Coz: a Causal Profiler for Linux

```
> coz run --- ./some_program args
```

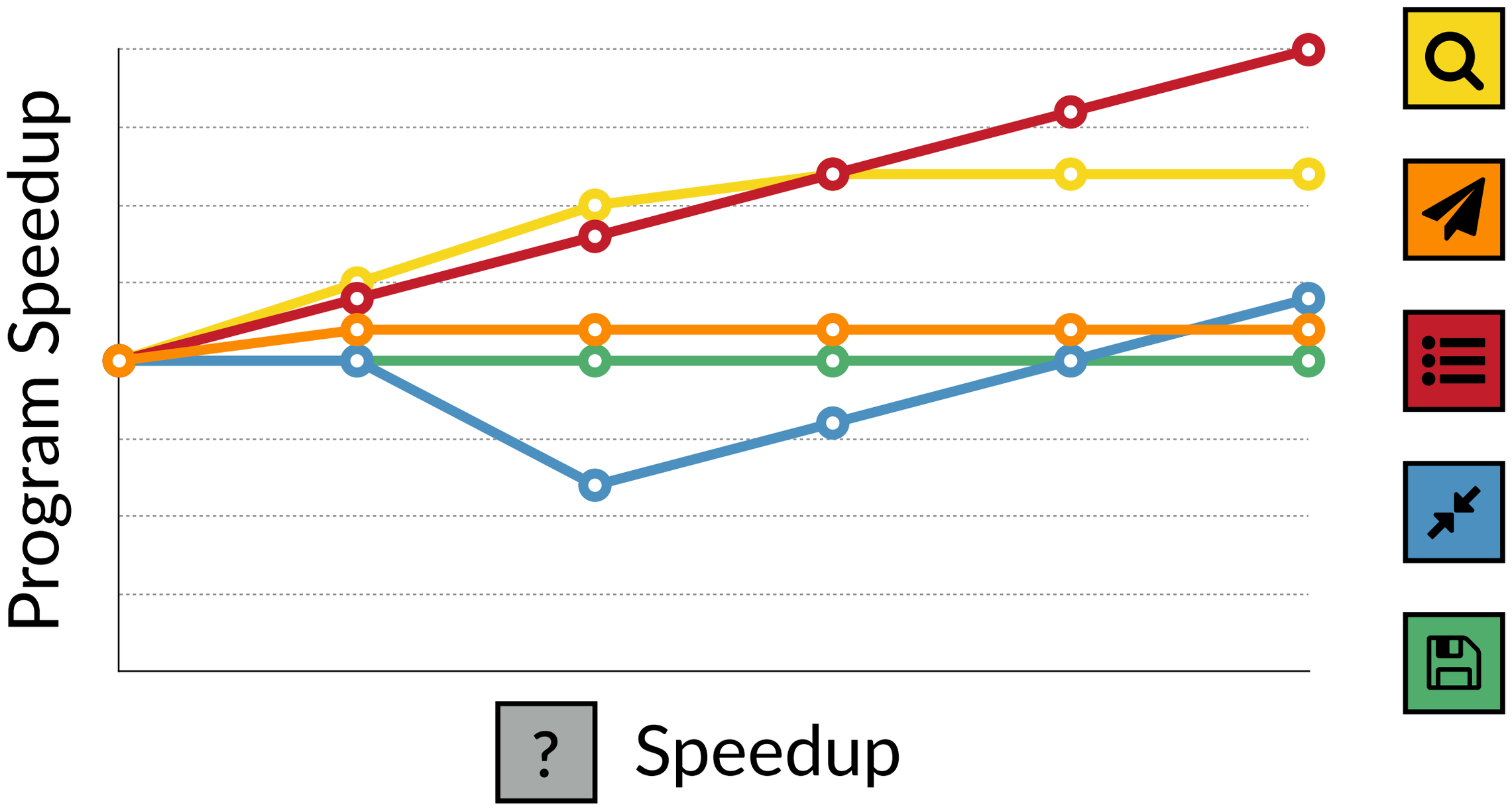
(coming to Debian, in Unstable now)

Coz Produces Causal Profiles

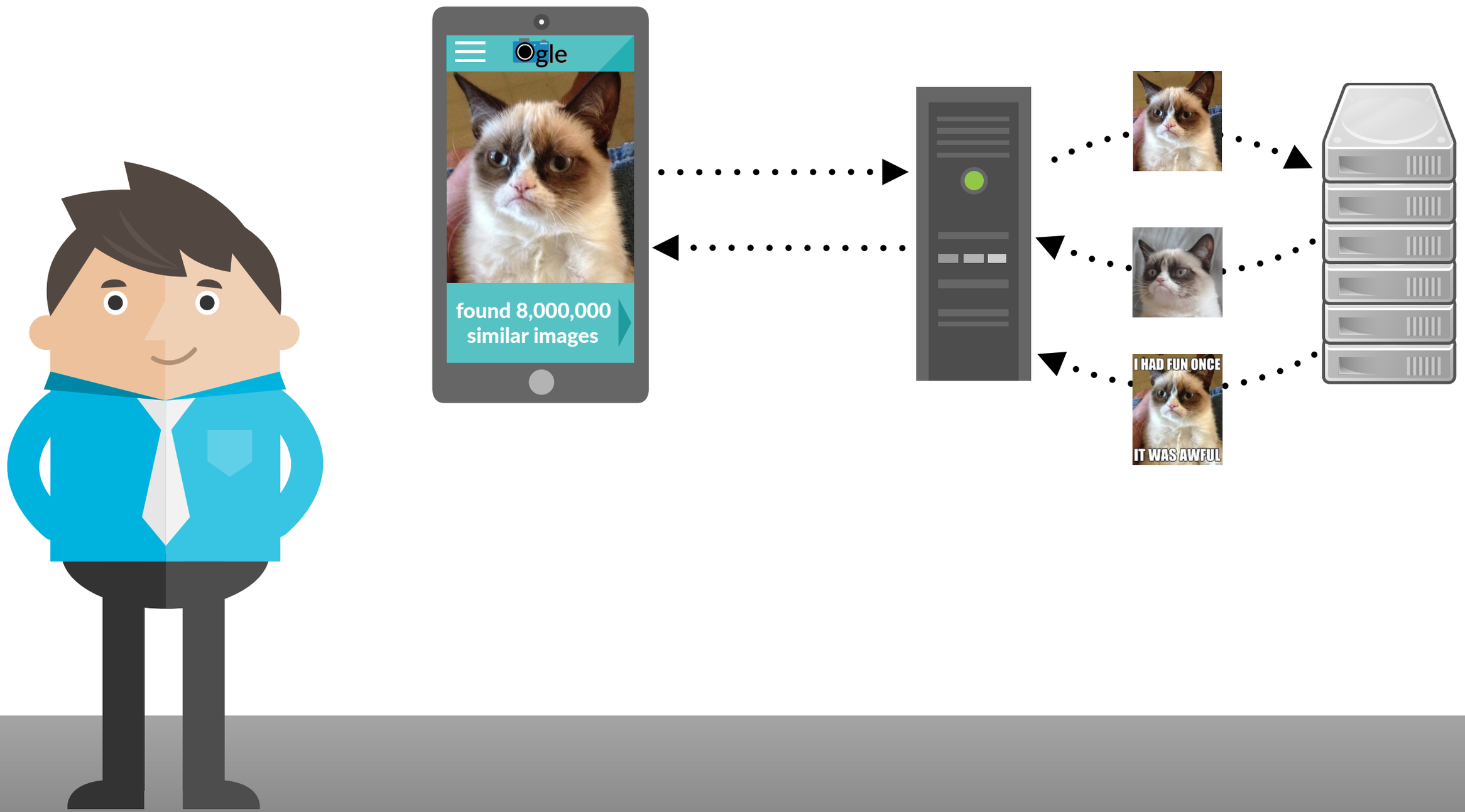


Coz Produces Causal Profiles

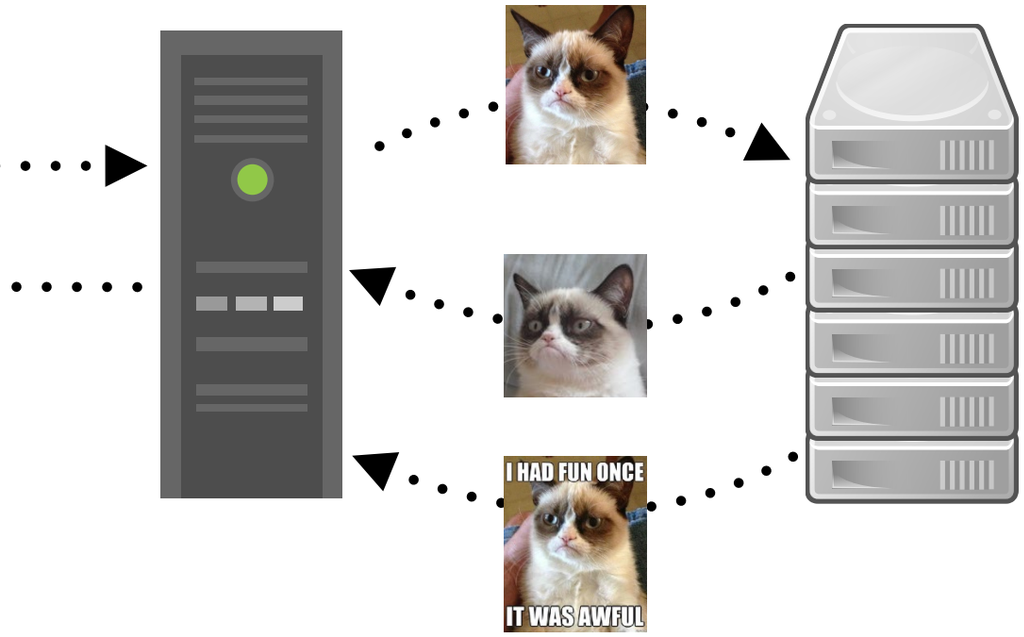
Let's use it to improve Ogle



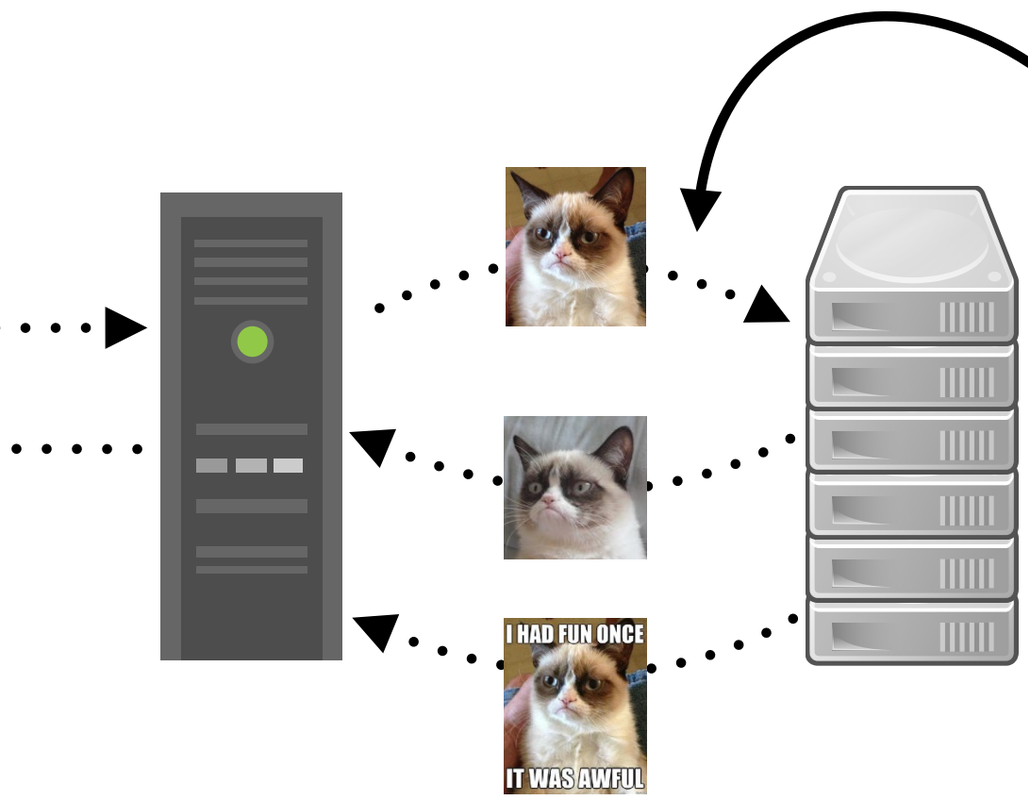
Using Causal Profiling on Ogle



Using Causal Profiling on Ogle

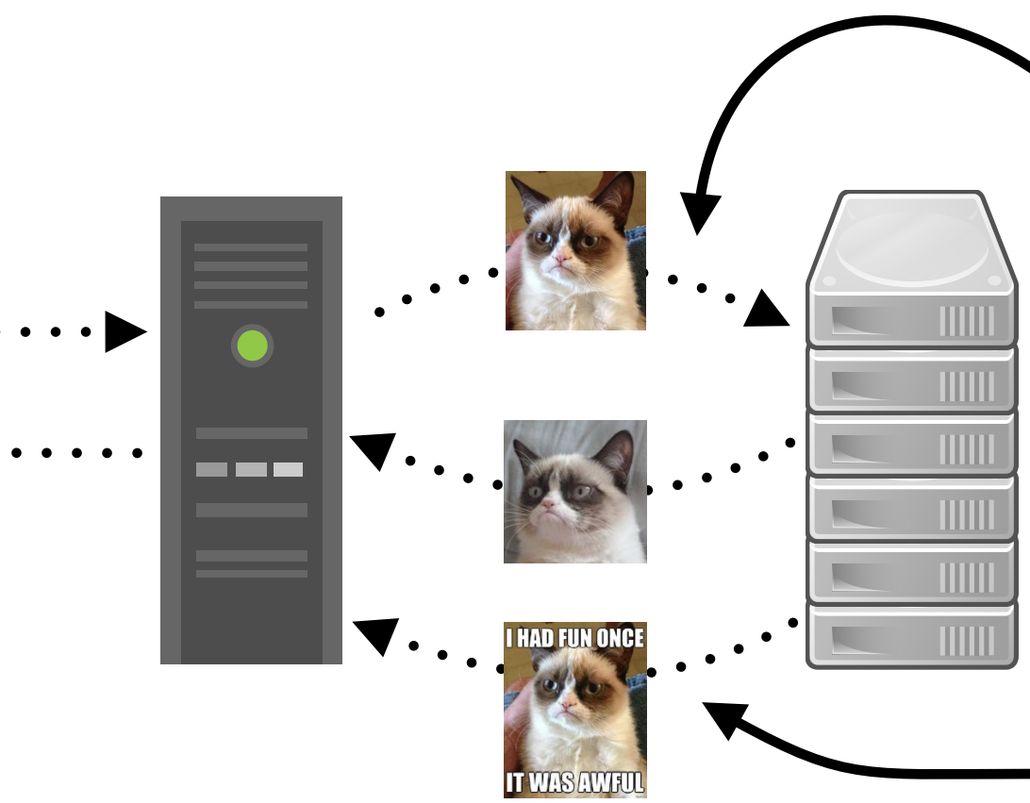


Using Causal Profiling on Ogle



dedup
compression

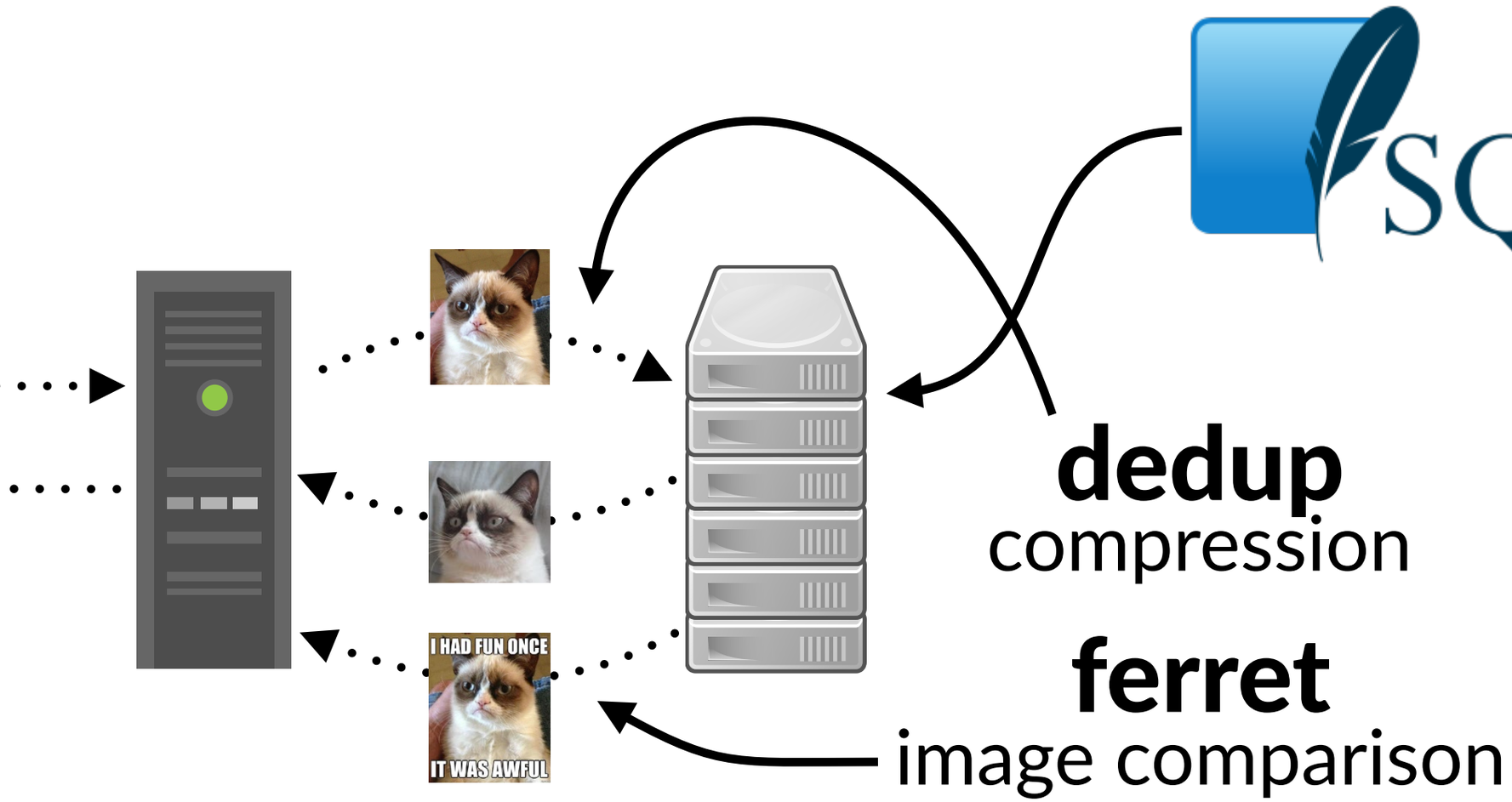
Using Causal Profiling on Ogle



dedup
compression

ferret
image comparison

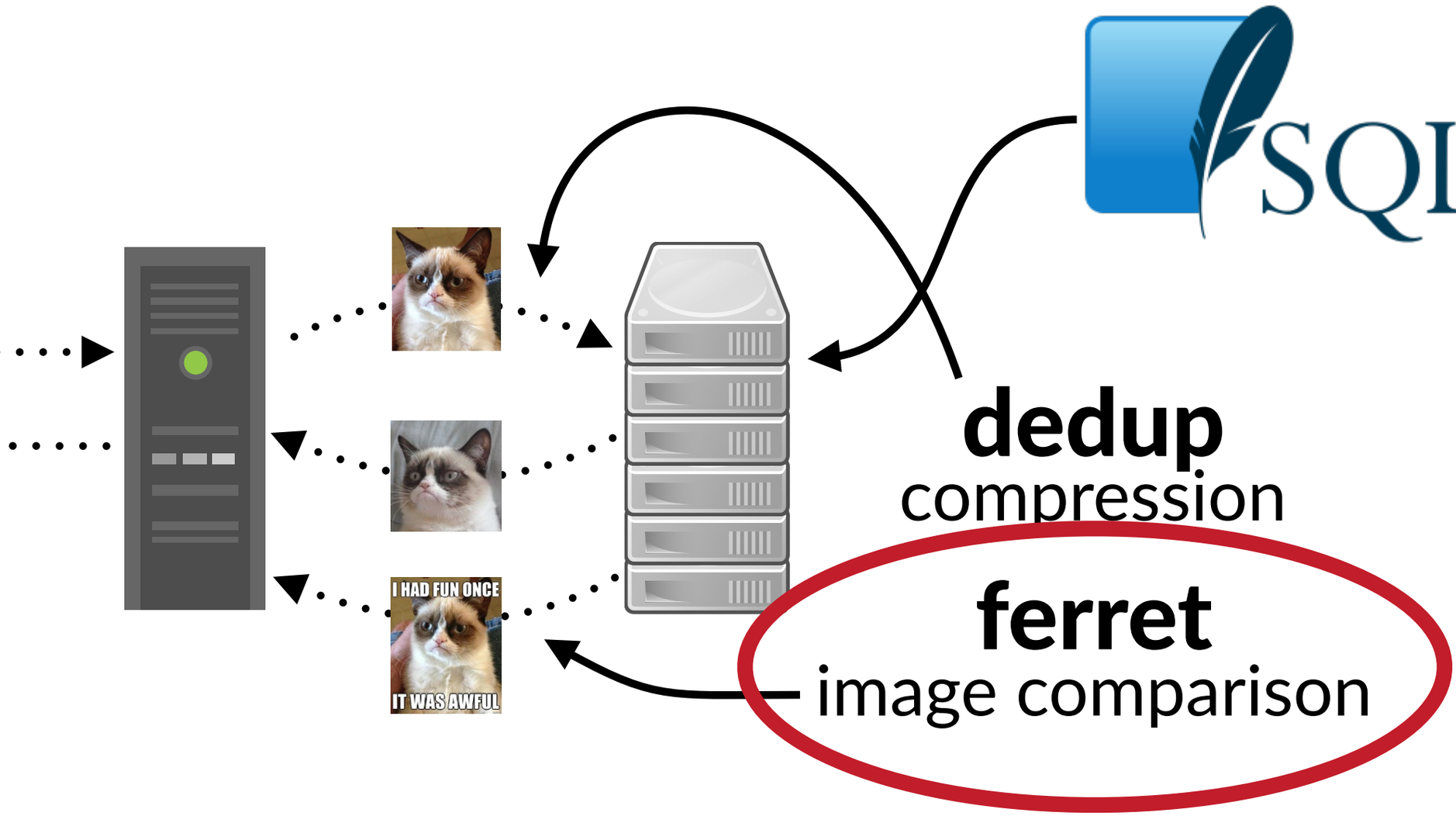
Using Causal Profiling on Ogle



dedup
compression

ferret
image comparison

Using Causal Profiling on Ogle

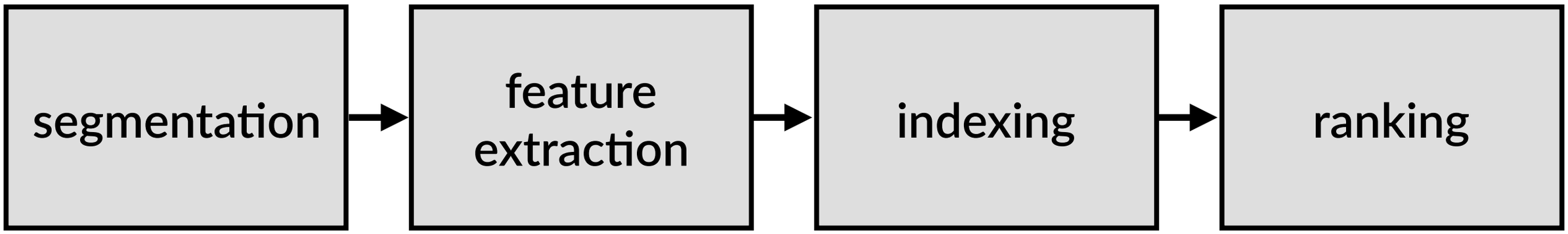


Ferret

image comparison

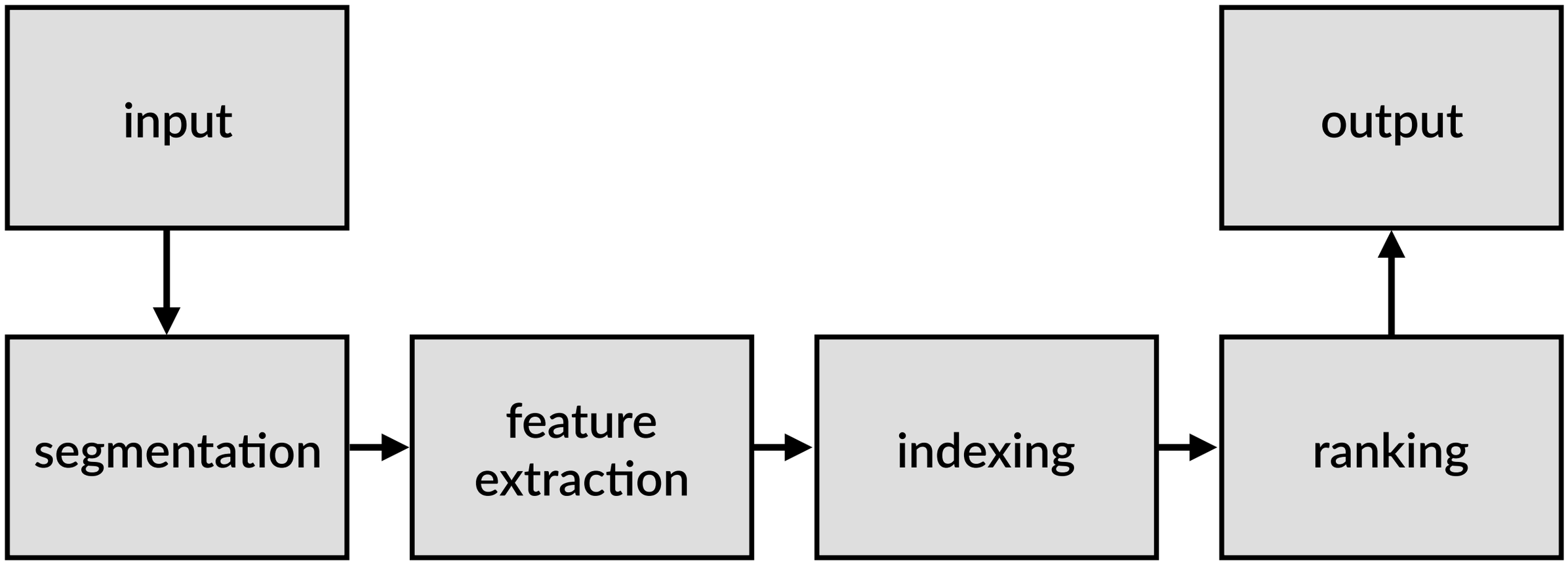
Ferret

image comparison



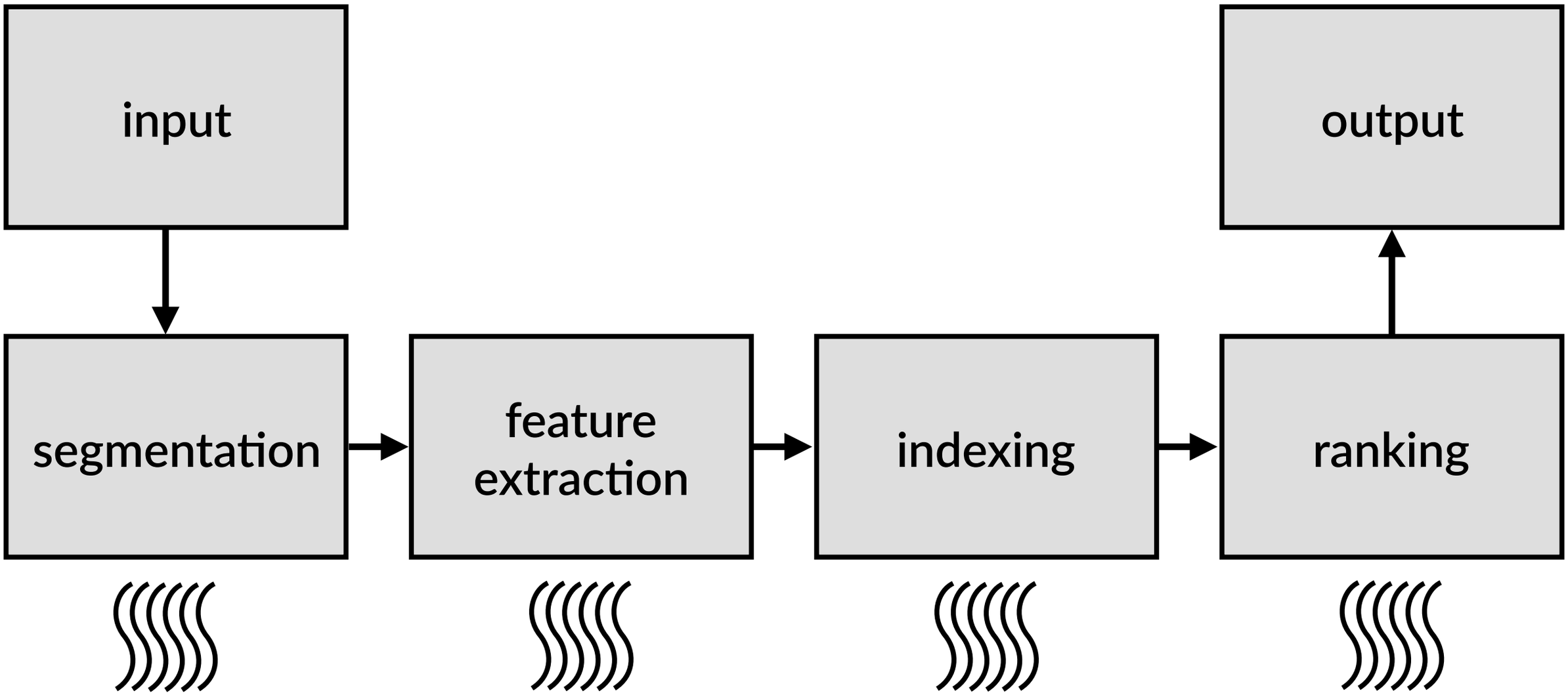
Ferret

image comparison

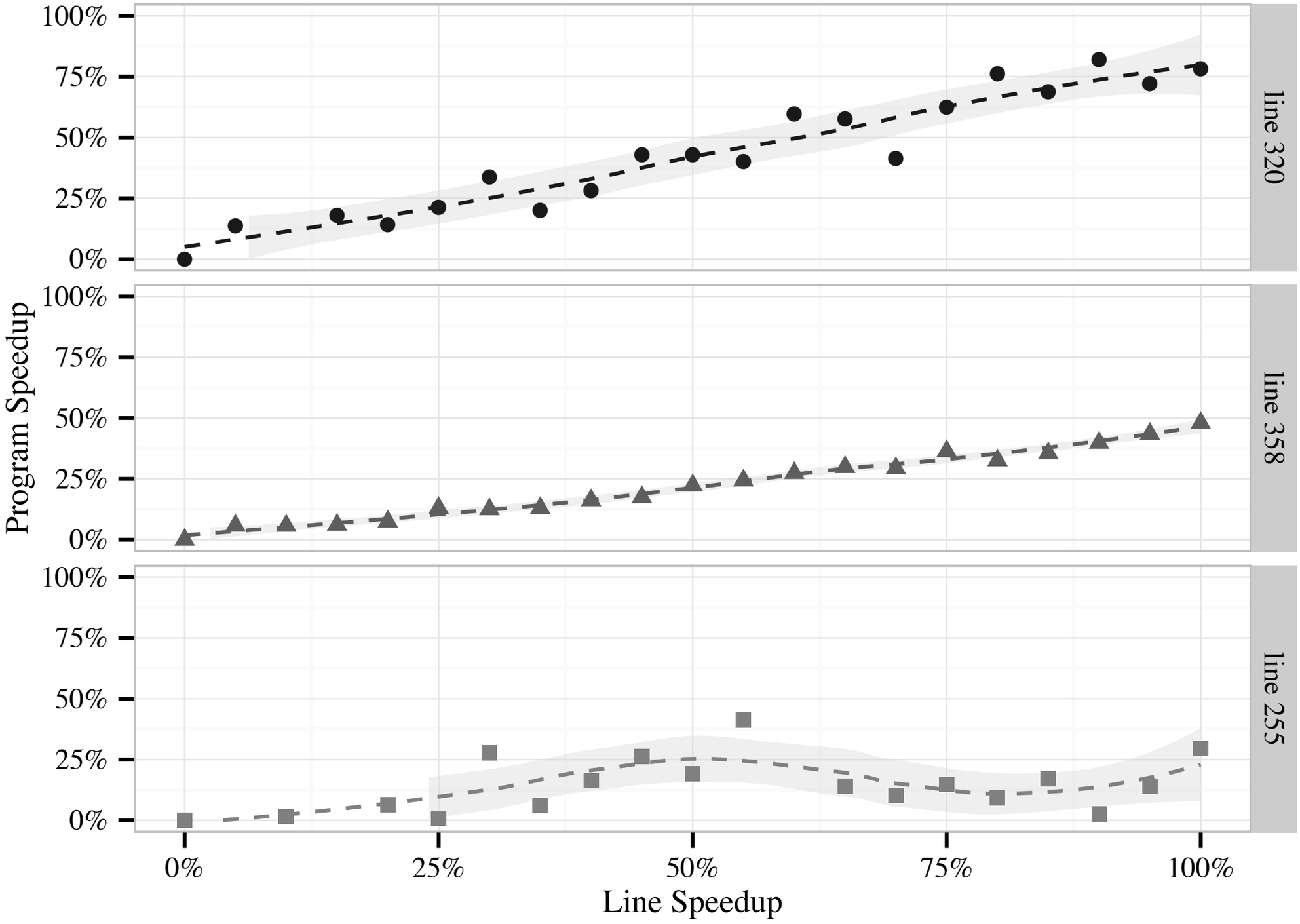


Ferret

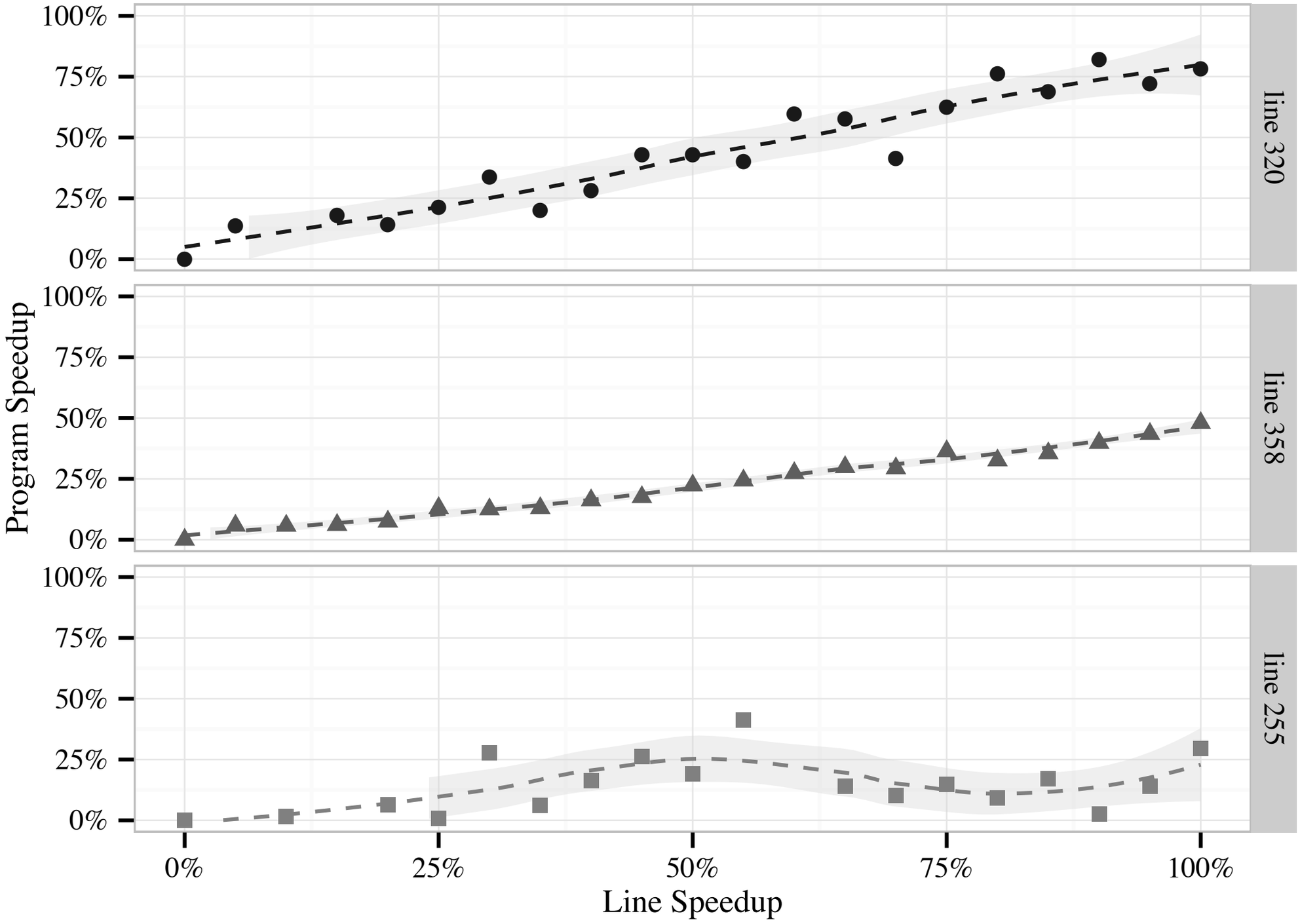
image comparison



Ferret


100%
75%
50%
25%
0%
line 320
line 358
line 255
Program Speedup
0%
25%
50%
75%
100%
Line Speedup
Line
line 320
line 358
line 255

Ferret

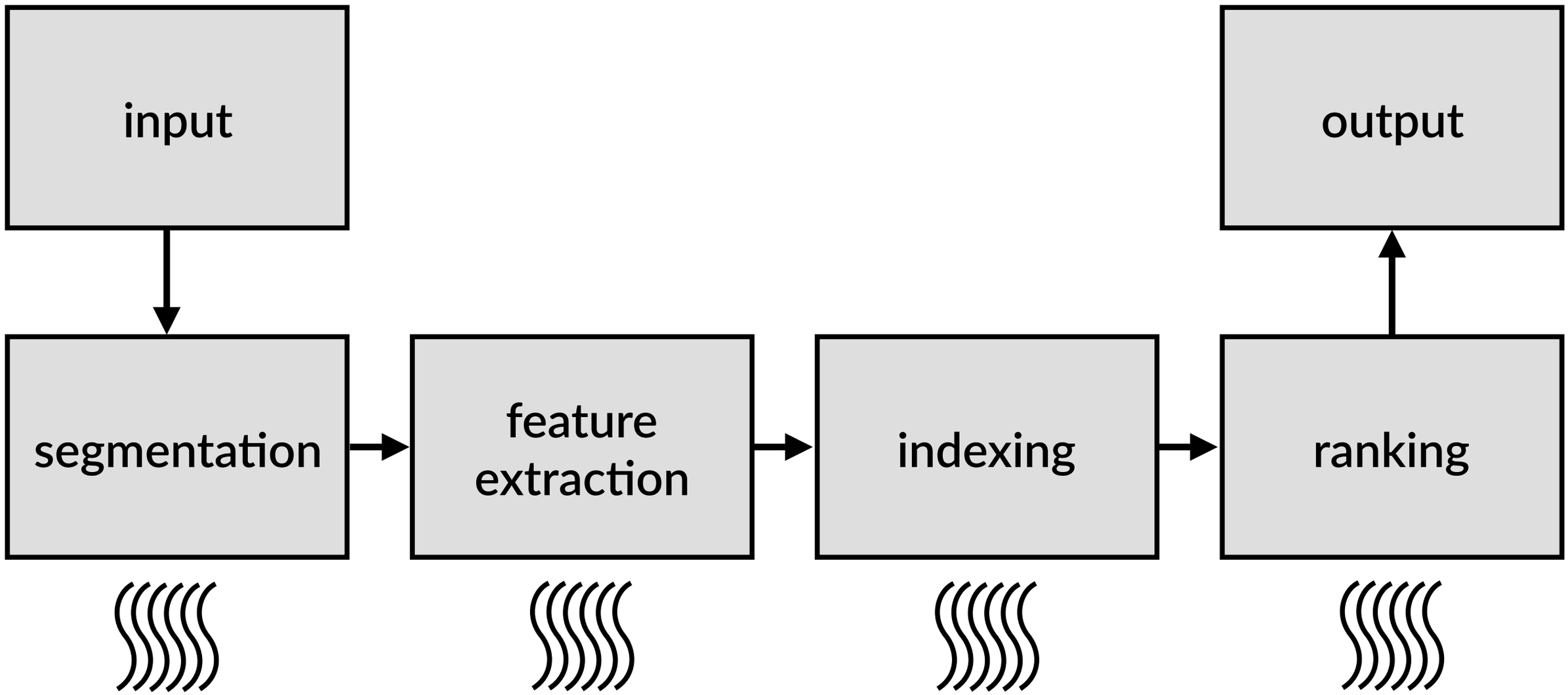


ranking

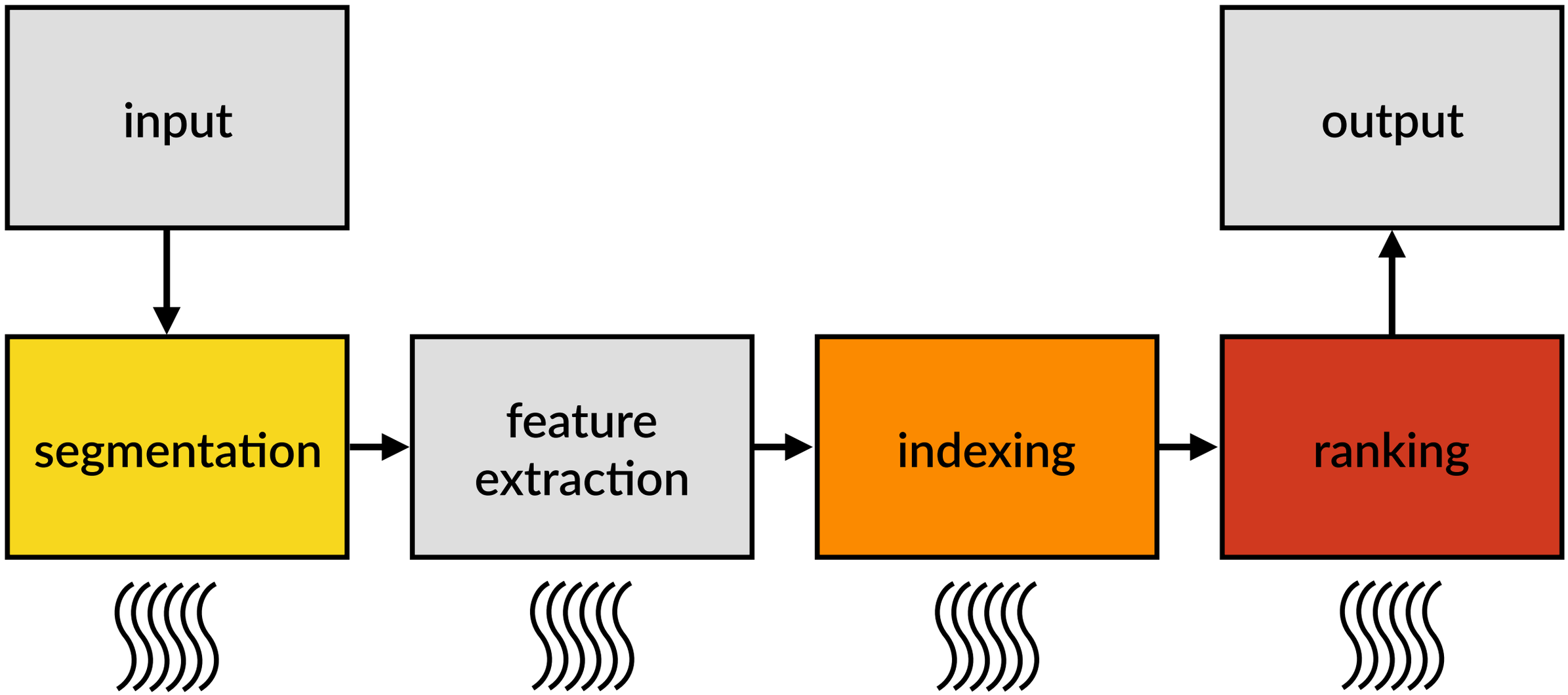
indexing

segmentation

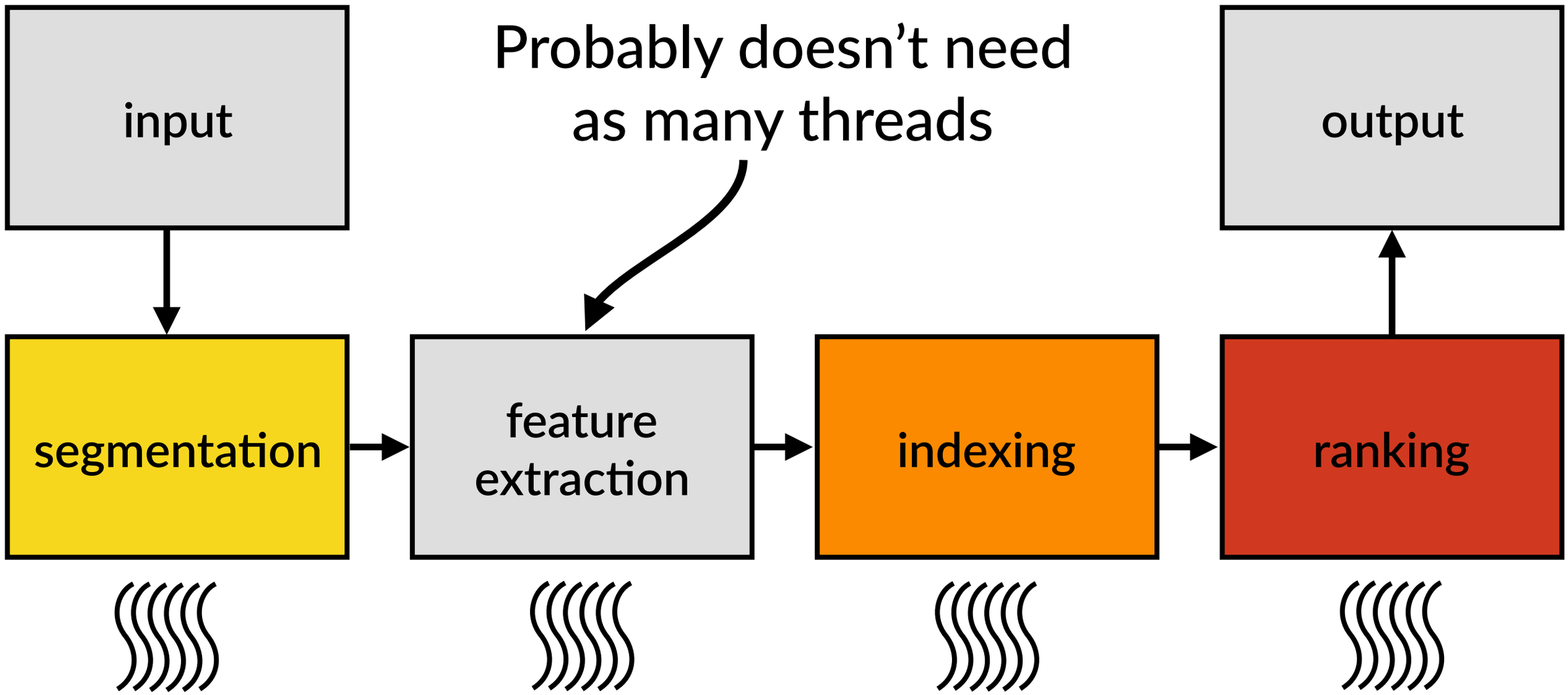
Ferret


input
segmentation
feature extraction
indexing
ranking
output

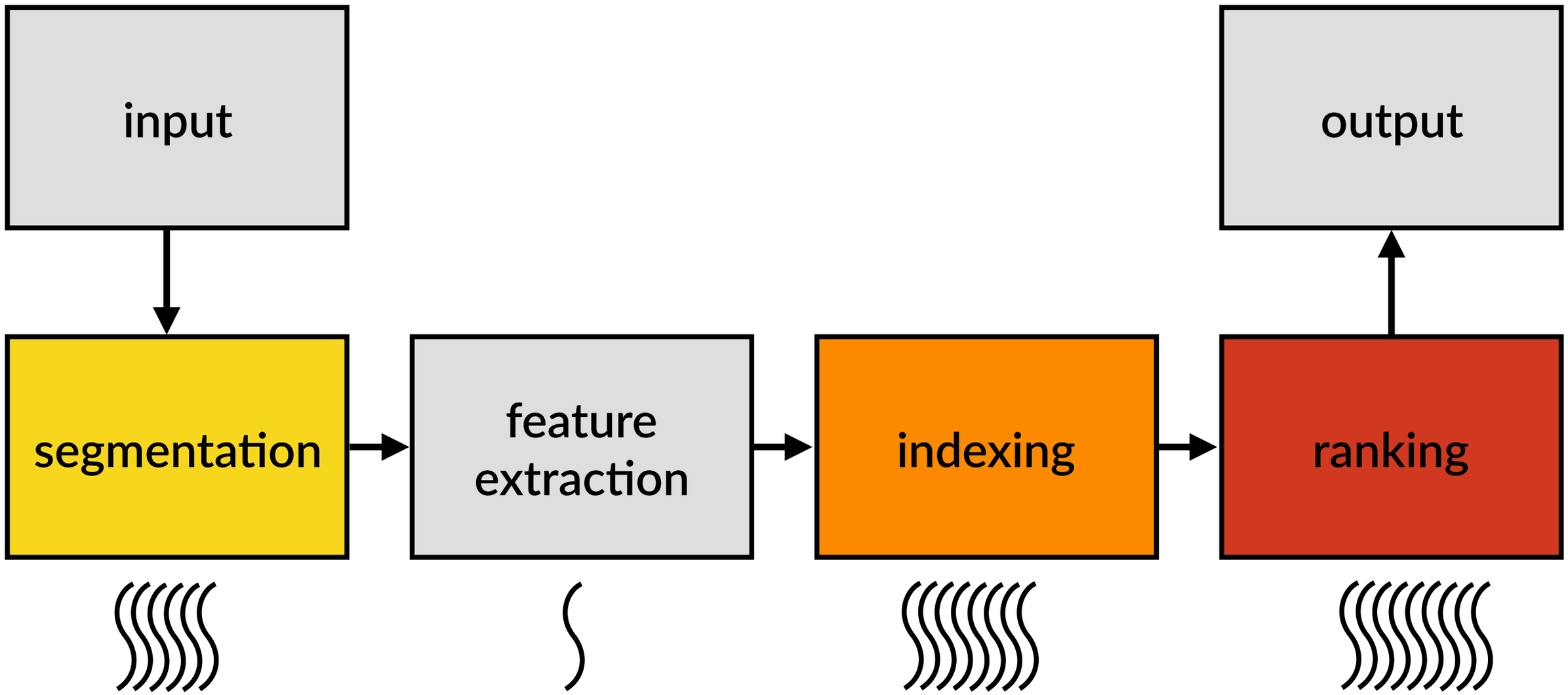
Ferret


input
output
segmentation
feature extraction
indexing
ranking

Ferret

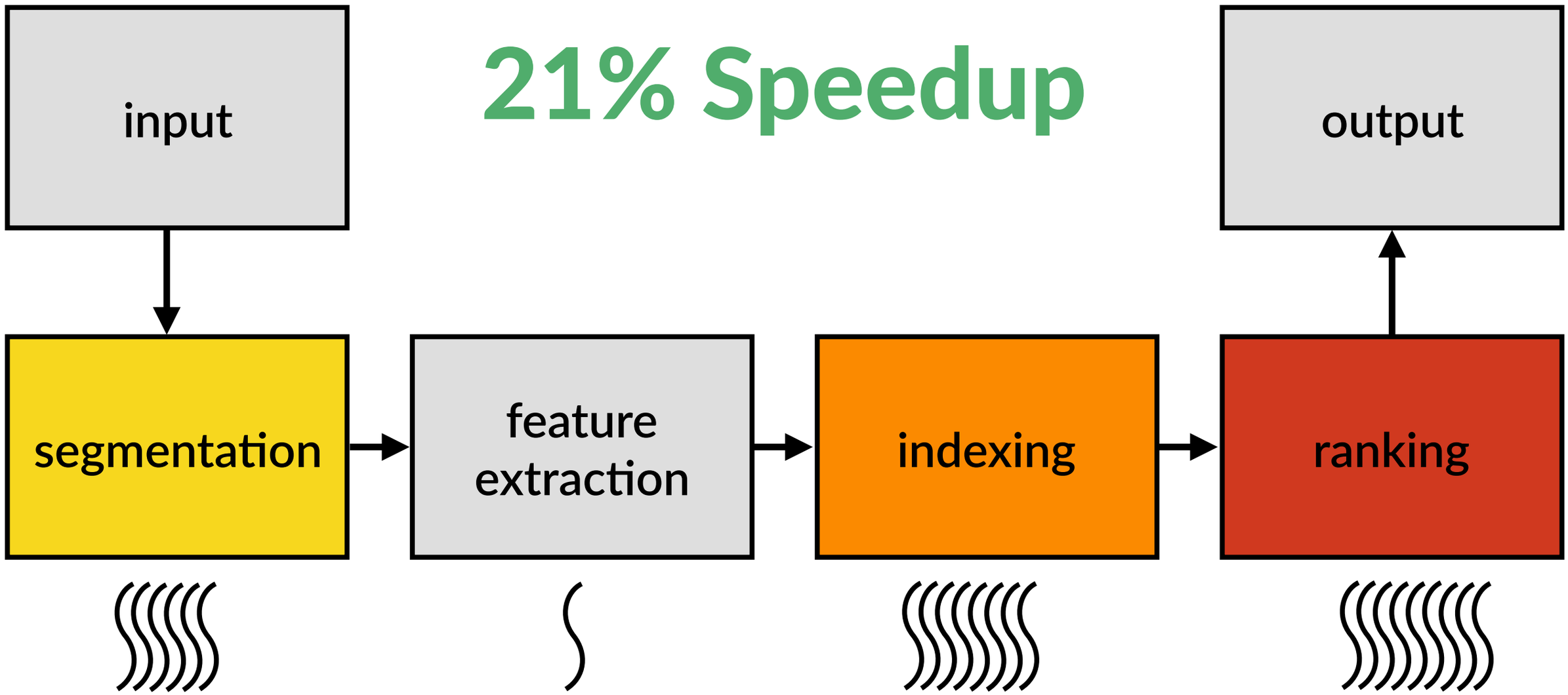

input
Probably doesn't need as many threads
output
segmentation
feature extraction
indexing
ranking

Ferret


input
output
segmentation
feature extraction
indexing
ranking

Ferret

21% Speedup



What did Causal Profiling predict?

What did Causal Profiling predict?

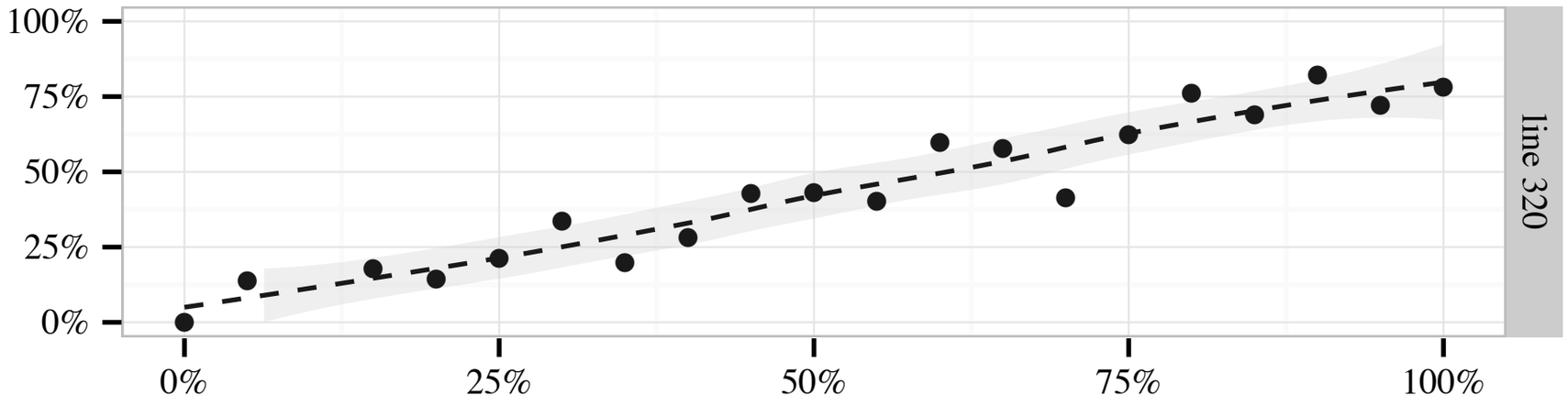
Increased from 16 to 22 threads

What did Causal Profiling predict?

Increased from 16 to 22 threads

27% increase in ranking throughput

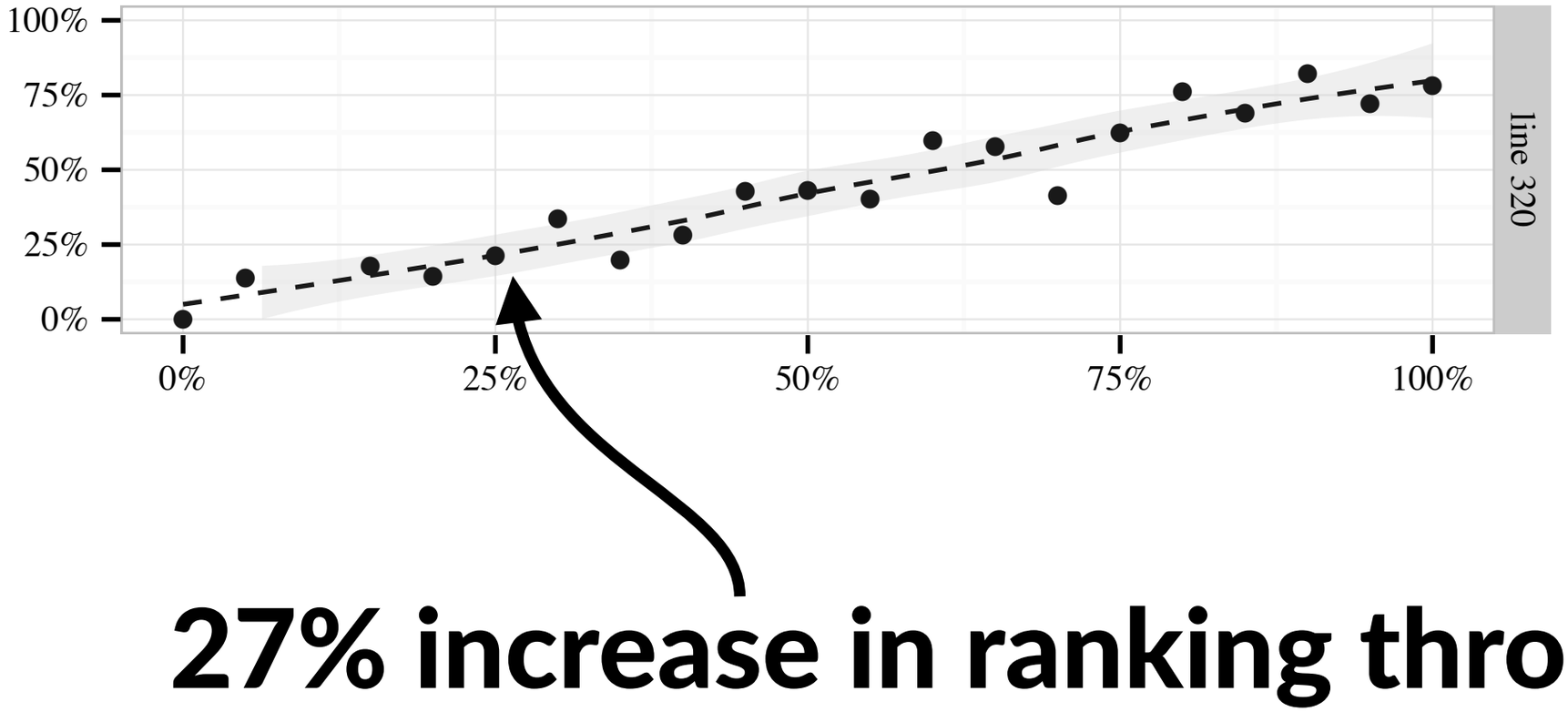
What did Causal Profiling predict?



ranking

27% increase in ranking throughput

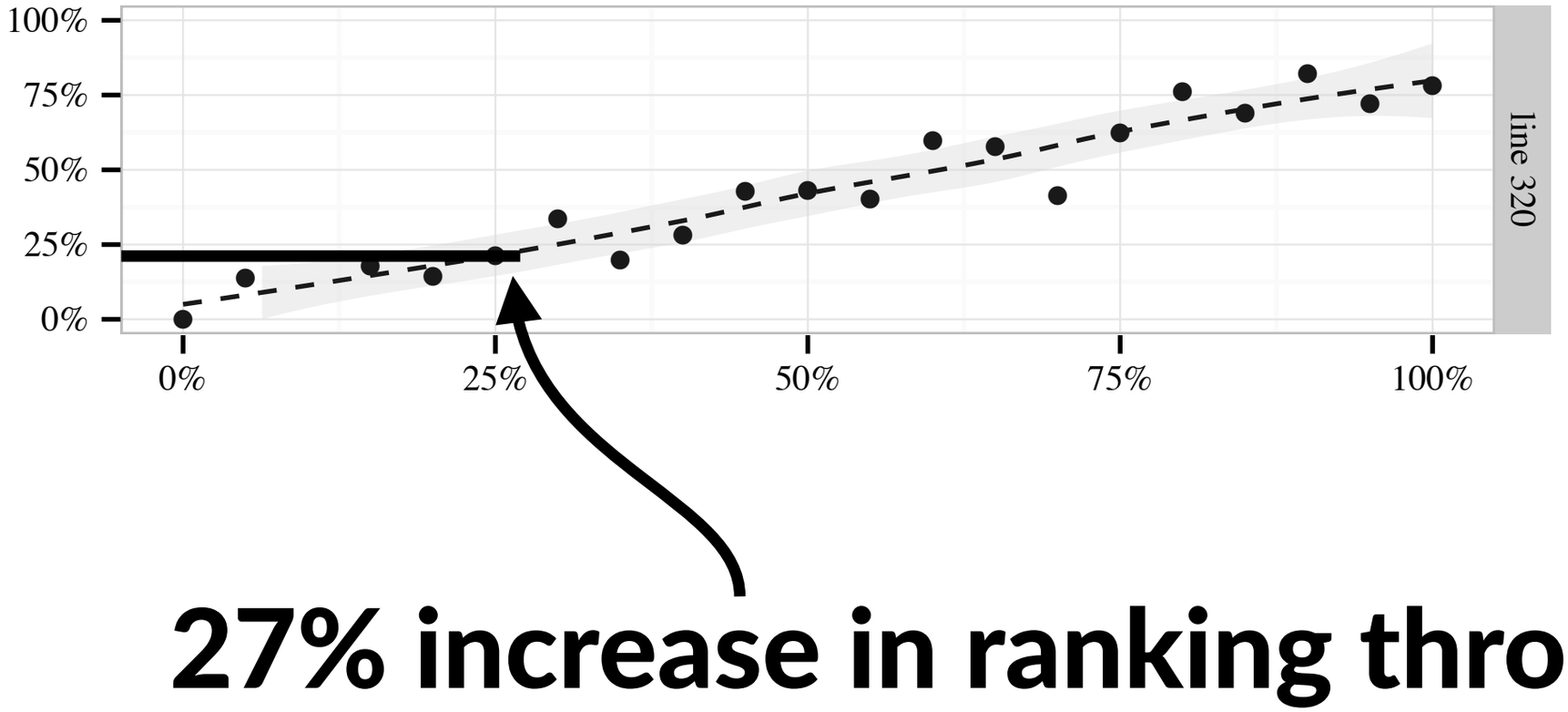
What did Causal Profiling predict?



ranking

27% increase in ranking throughput

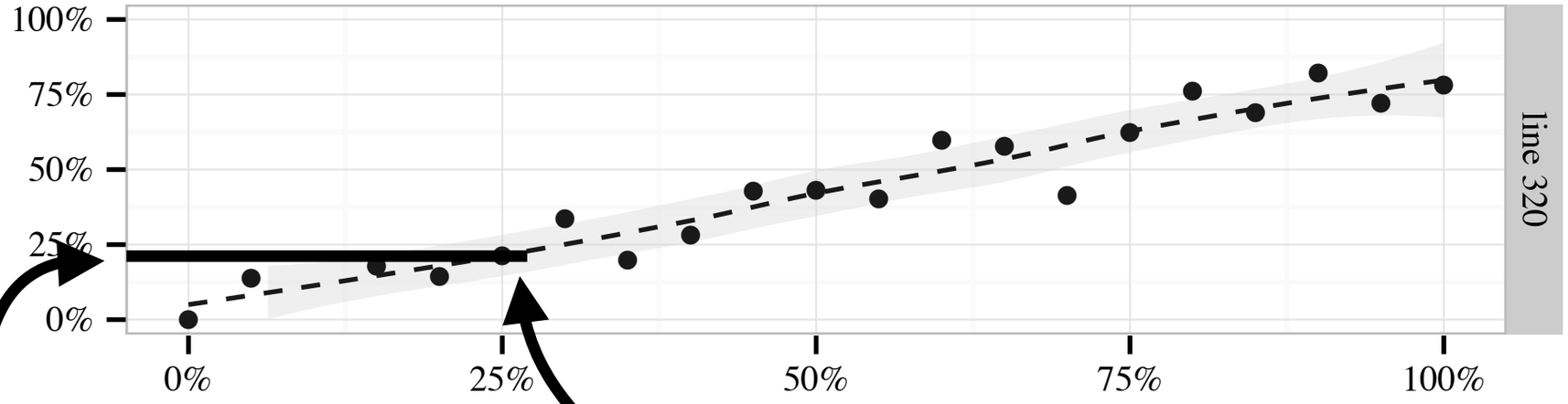
What did Causal Profiling predict?



ranking

27% increase in ranking throughput

What did Causal Profiling predict?

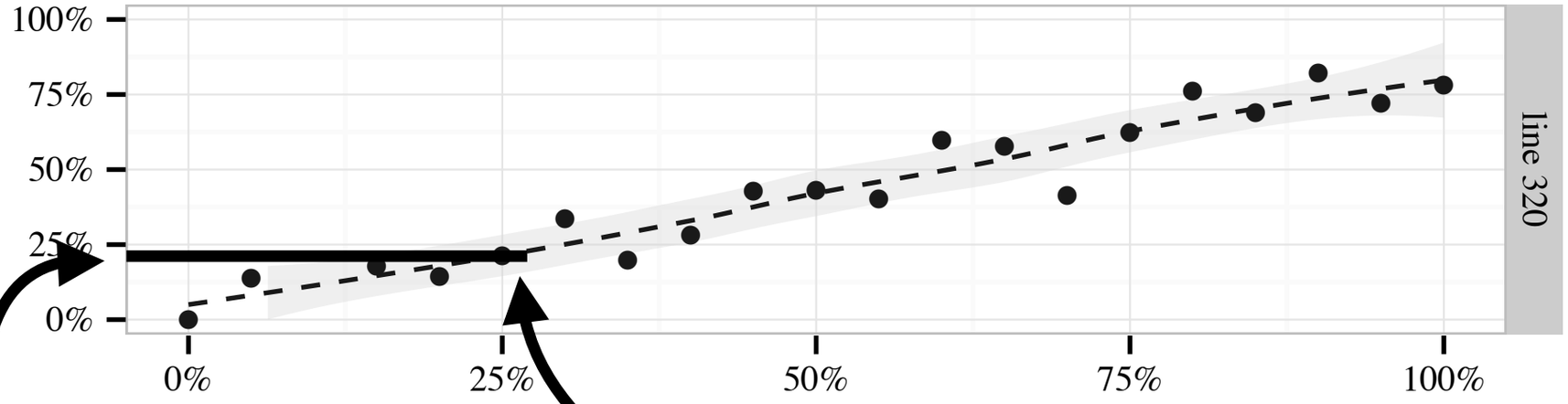


ranking

27% increase in ranking throughput

Causal Profiling predicted a 21% improvement

What did Causal Profiling predict?



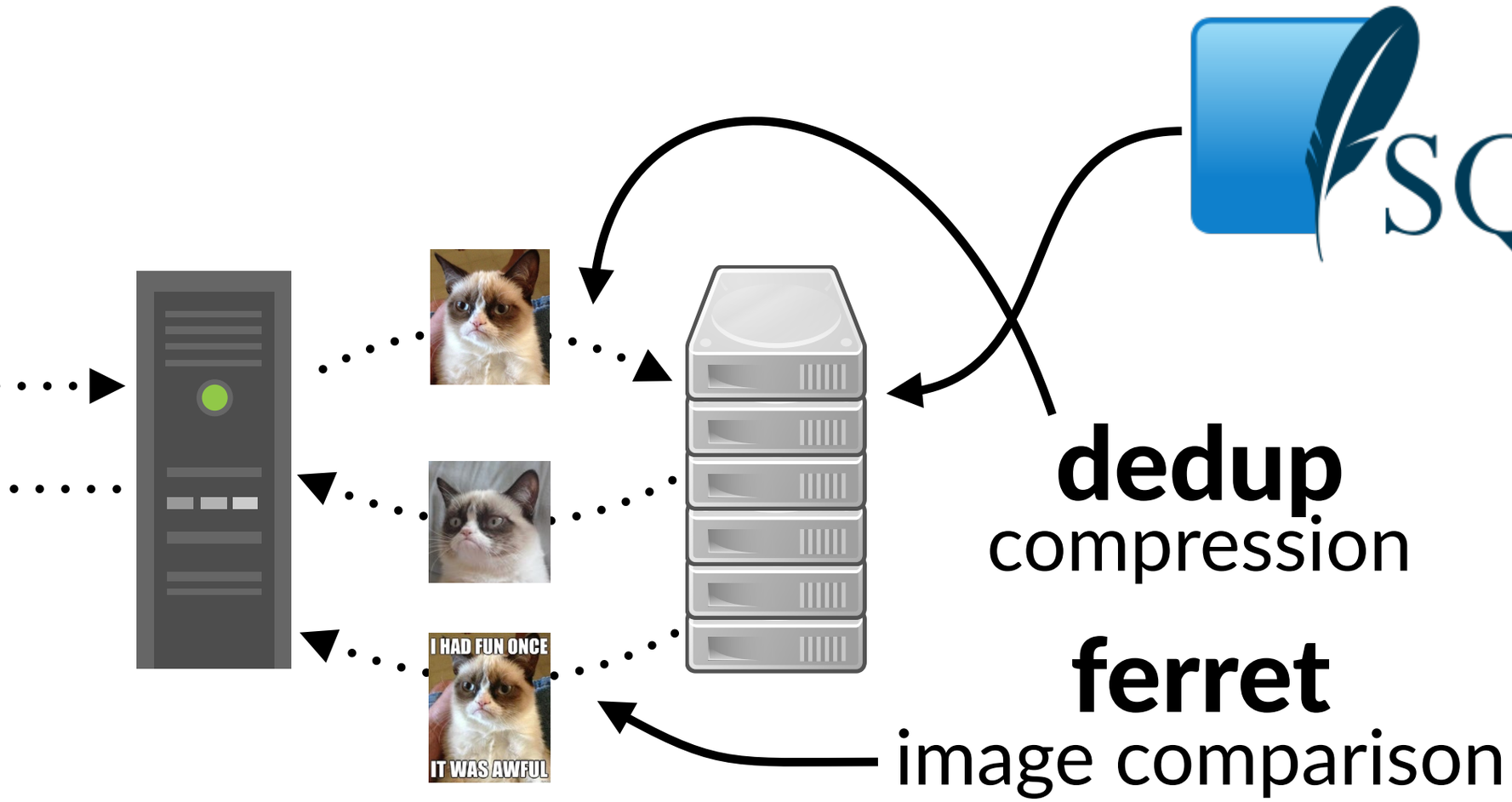
ranking

27% increase in ranking throughput

Causal Profiling predicted a 21% improvement

Exactly what we observed

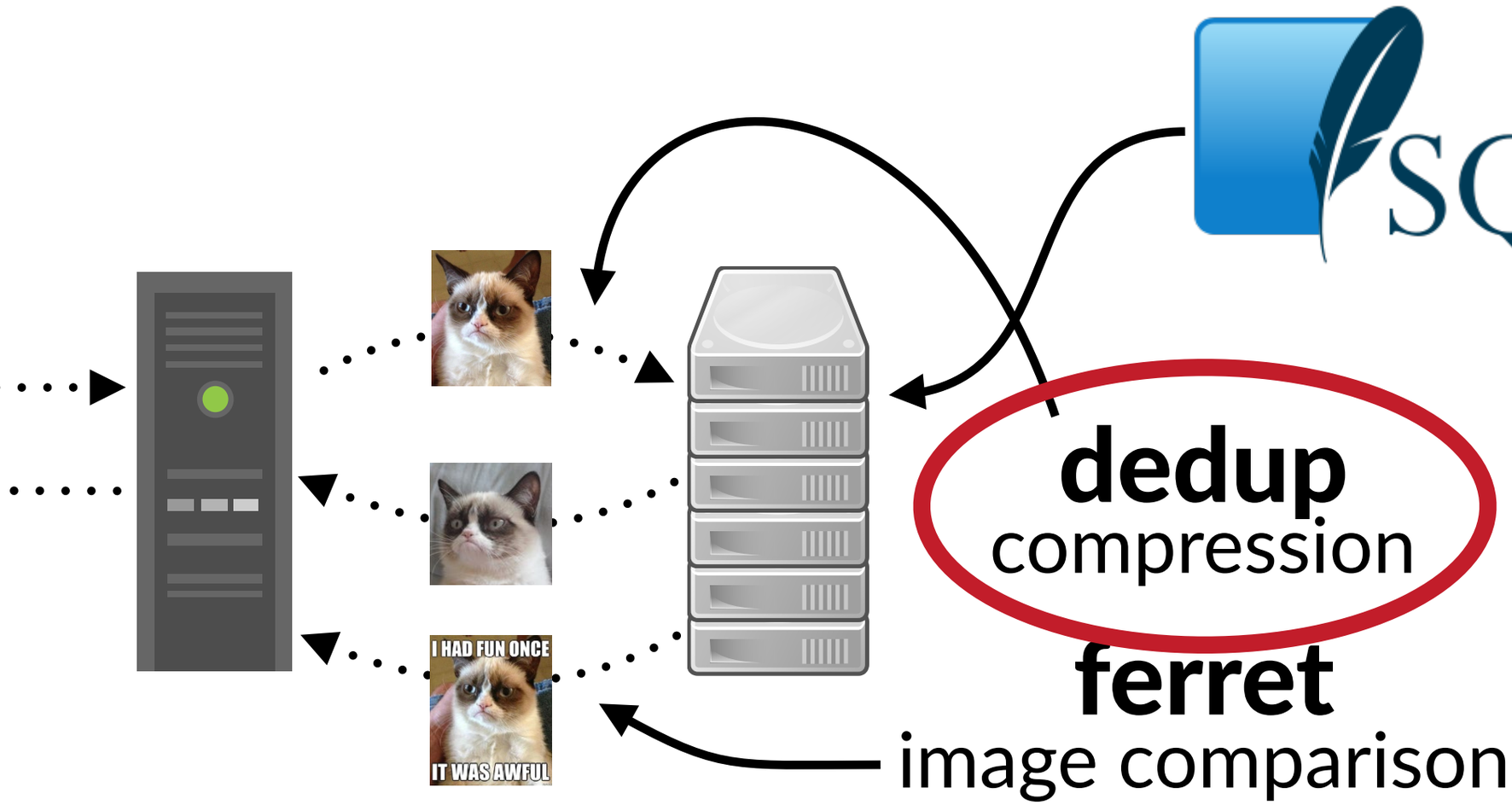
Using Causal Profiling on Ogle



dedup
compression

ferret
image comparison

Using Causal Profiling on Ogle

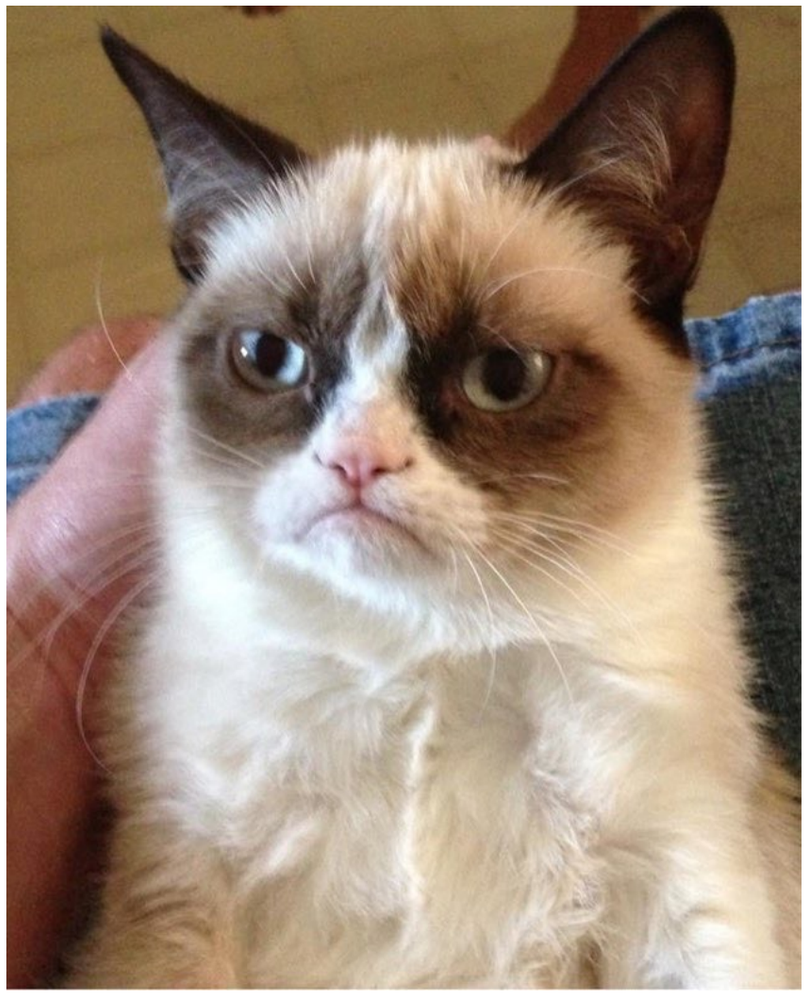


Dedup

Compression via deduplication

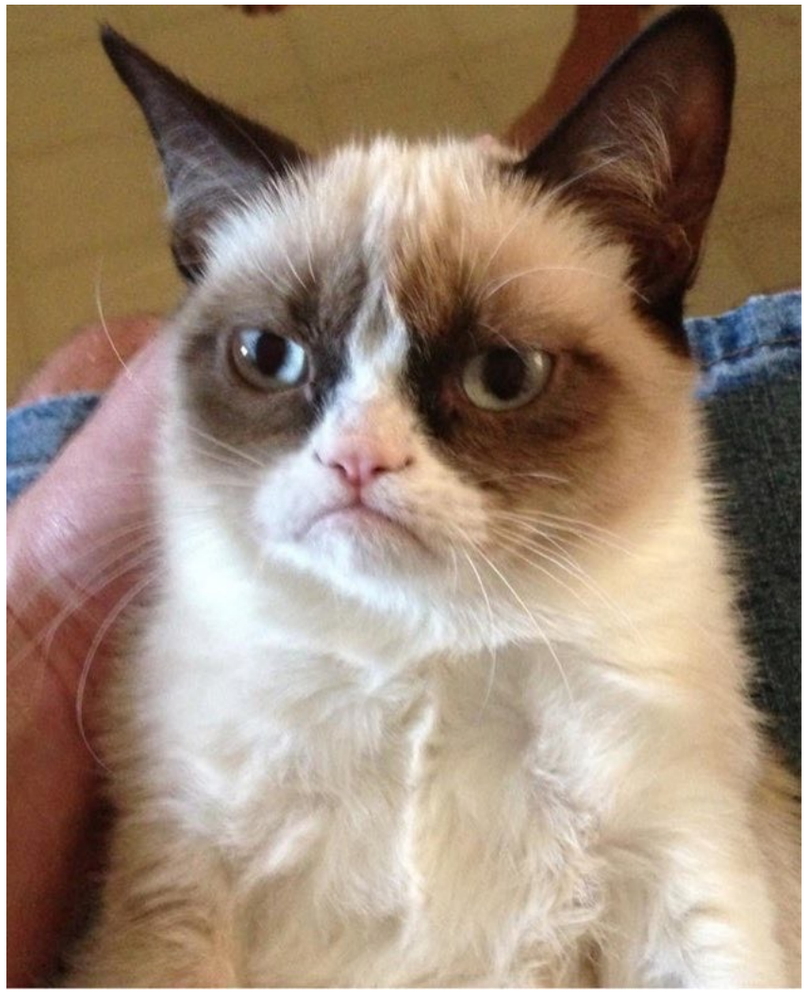
Dedup

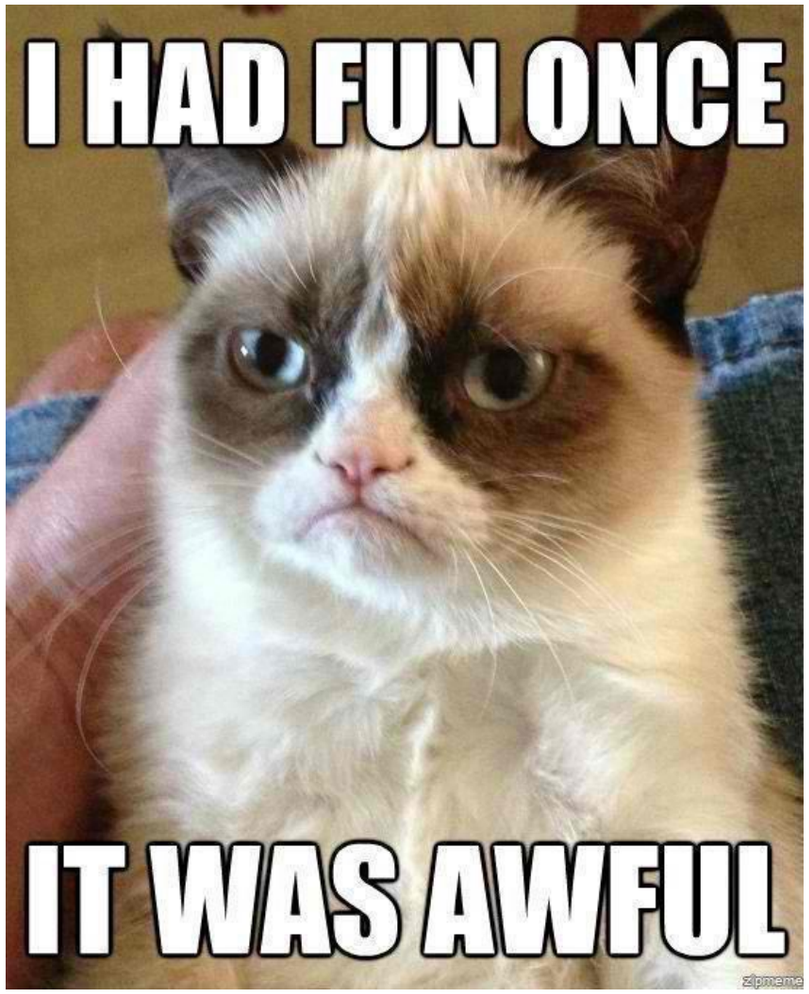
Compression via deduplication



Dedup

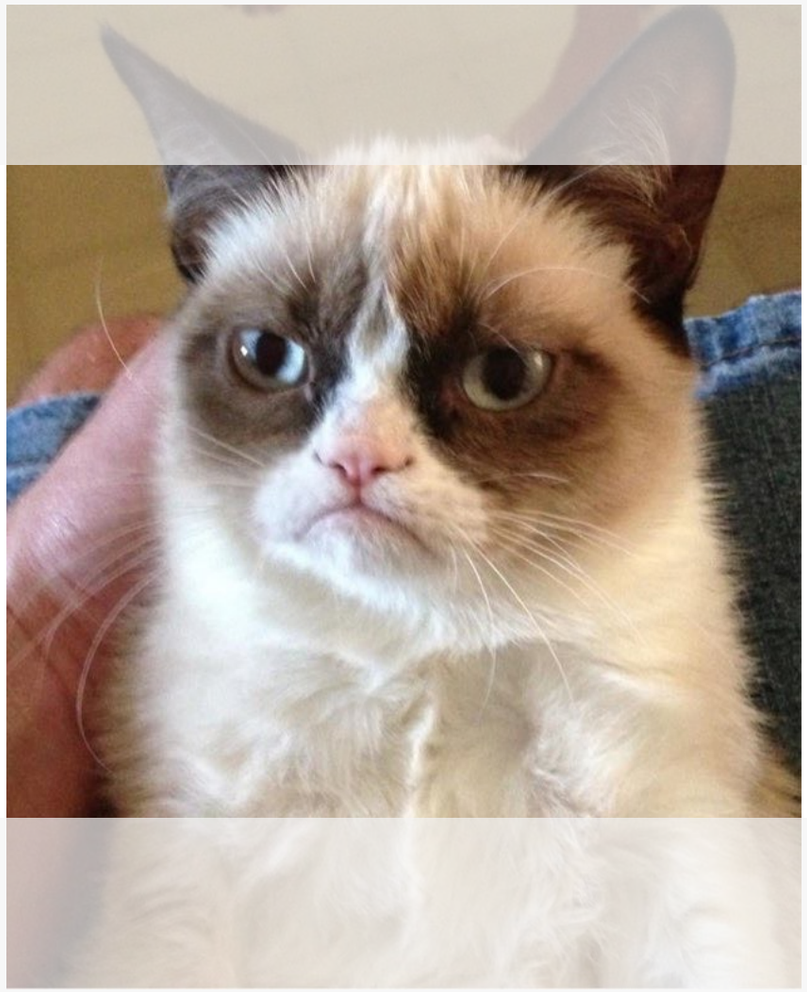
Compression via deduplication





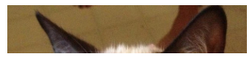
Dedup

Compression via deduplication



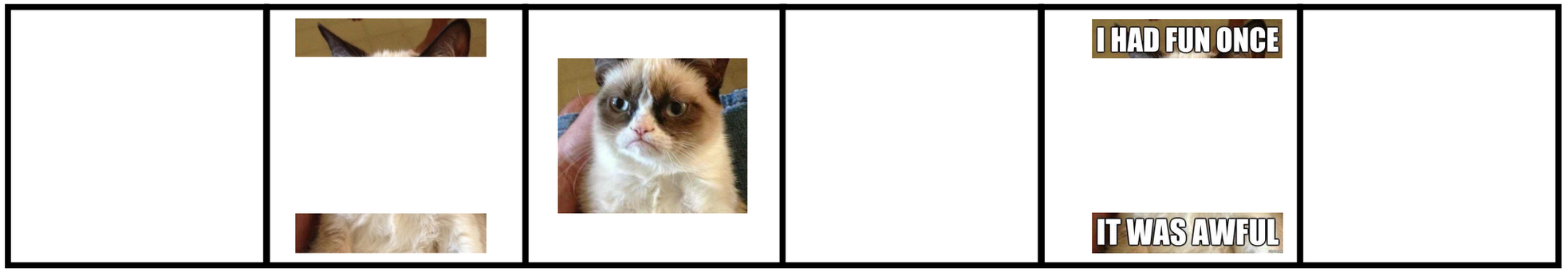
Dedup

Compression via deduplication



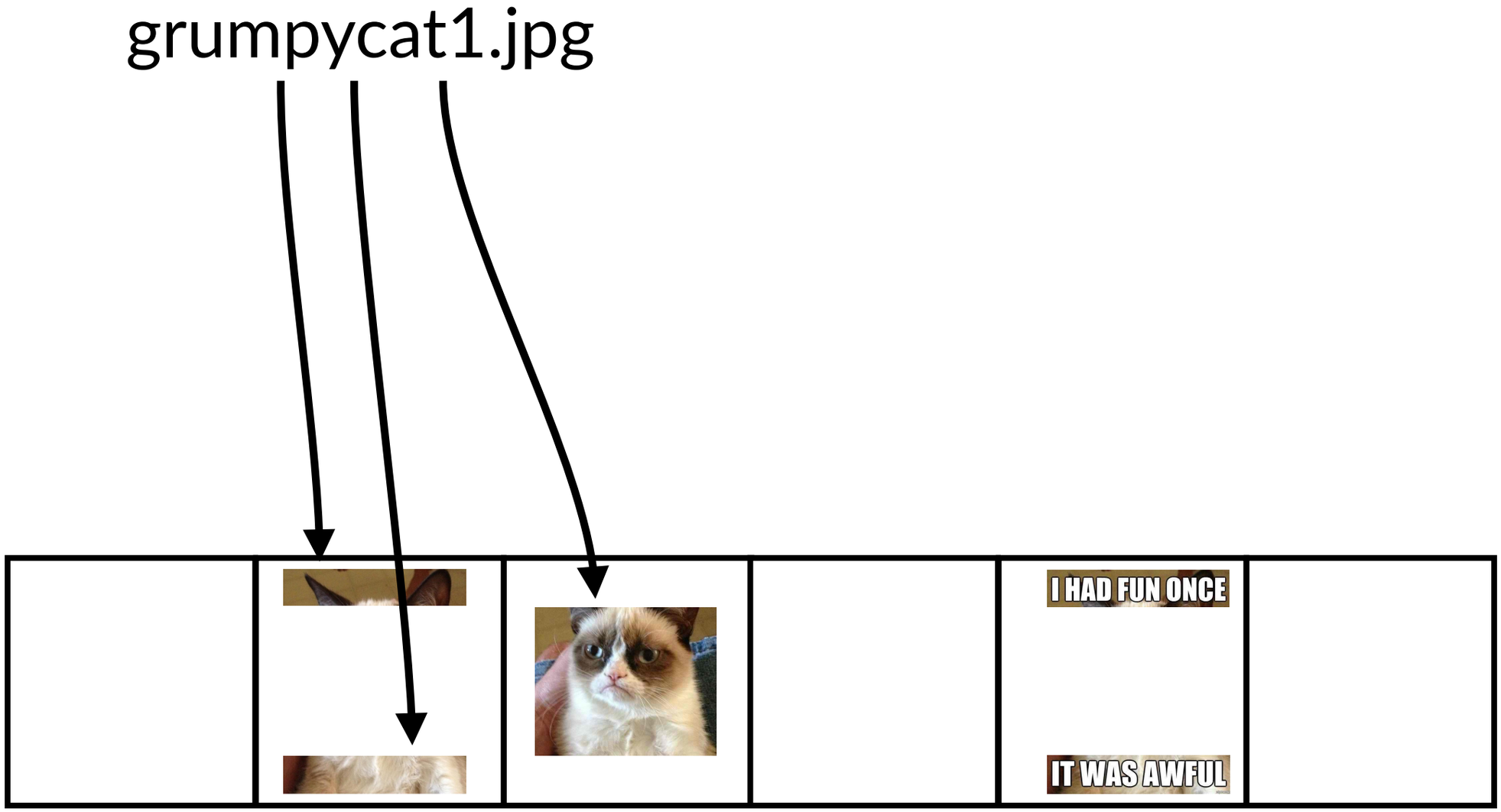
Dedup

Compression via deduplication



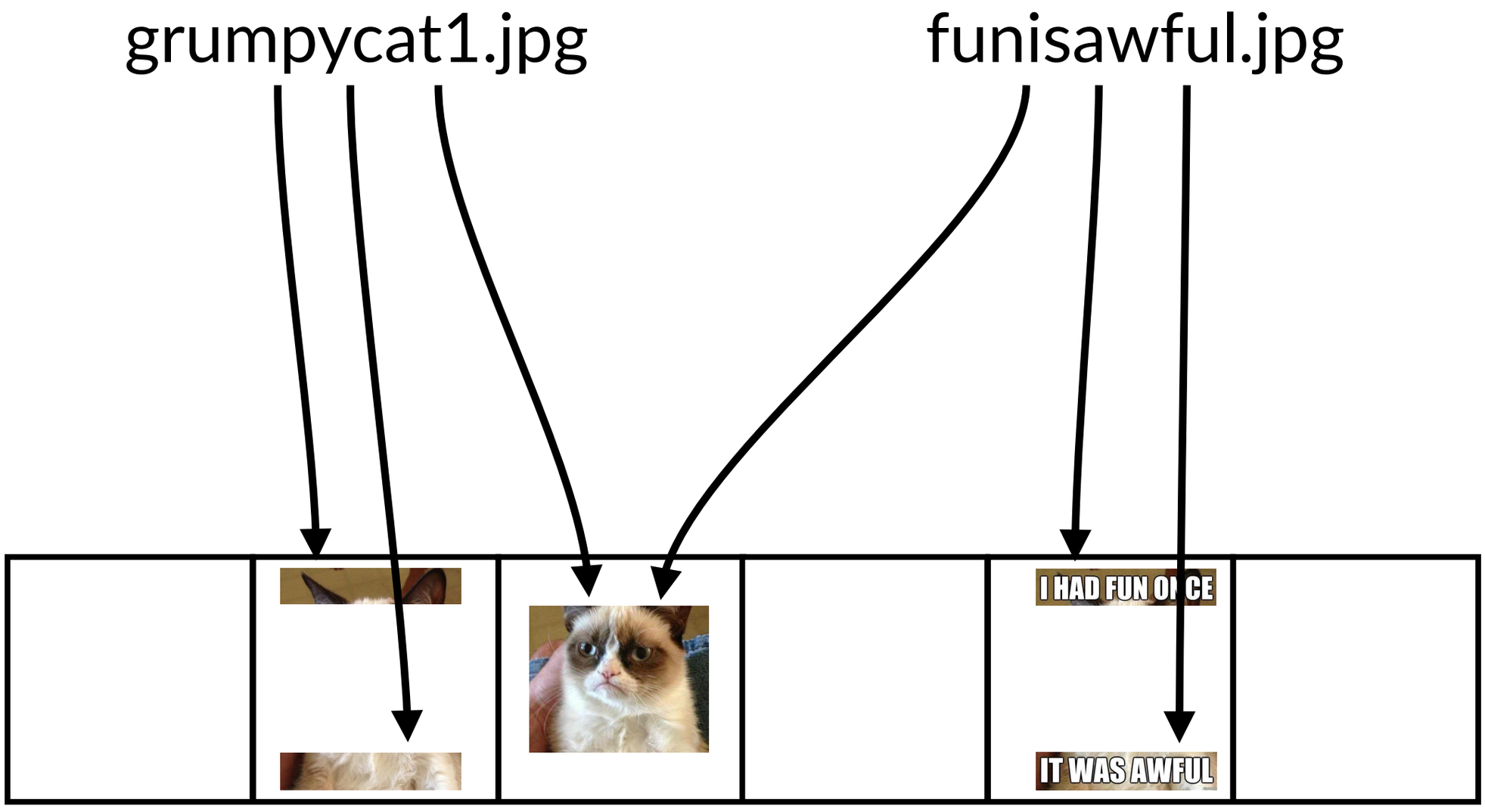
Dedup

Compression via deduplication



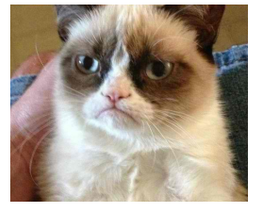
Dedup

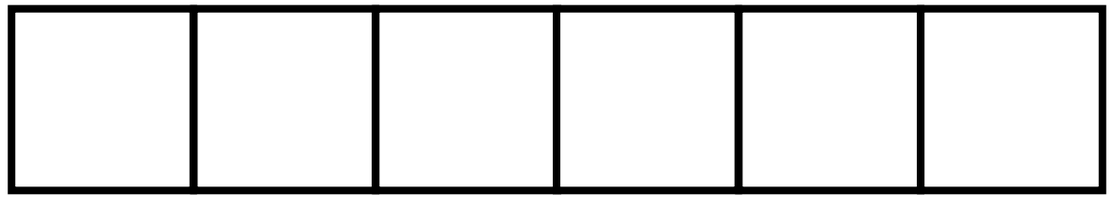
Compression via deduplication



Dedup

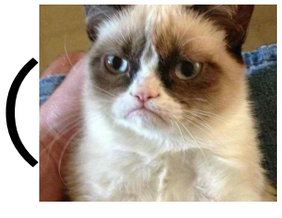
Compression via deduplication

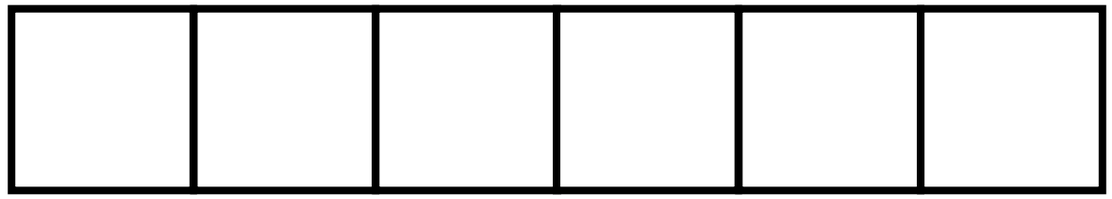




Dedup

Compression via deduplication

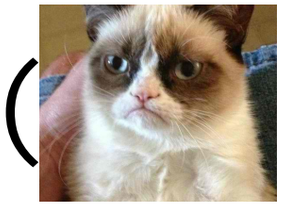
`hash_function(`  `)`



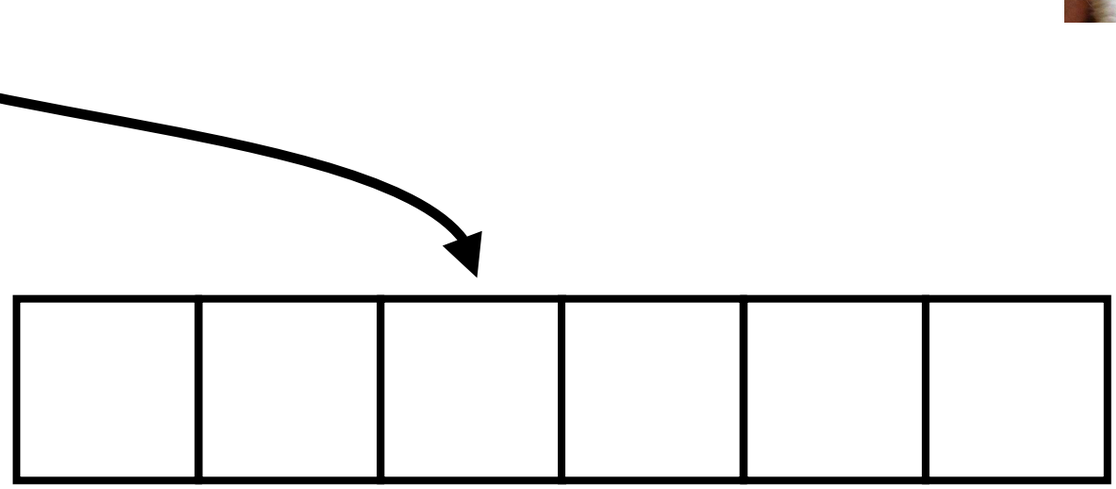
Dedup

Compression via deduplication

```
i = hash_function(
```



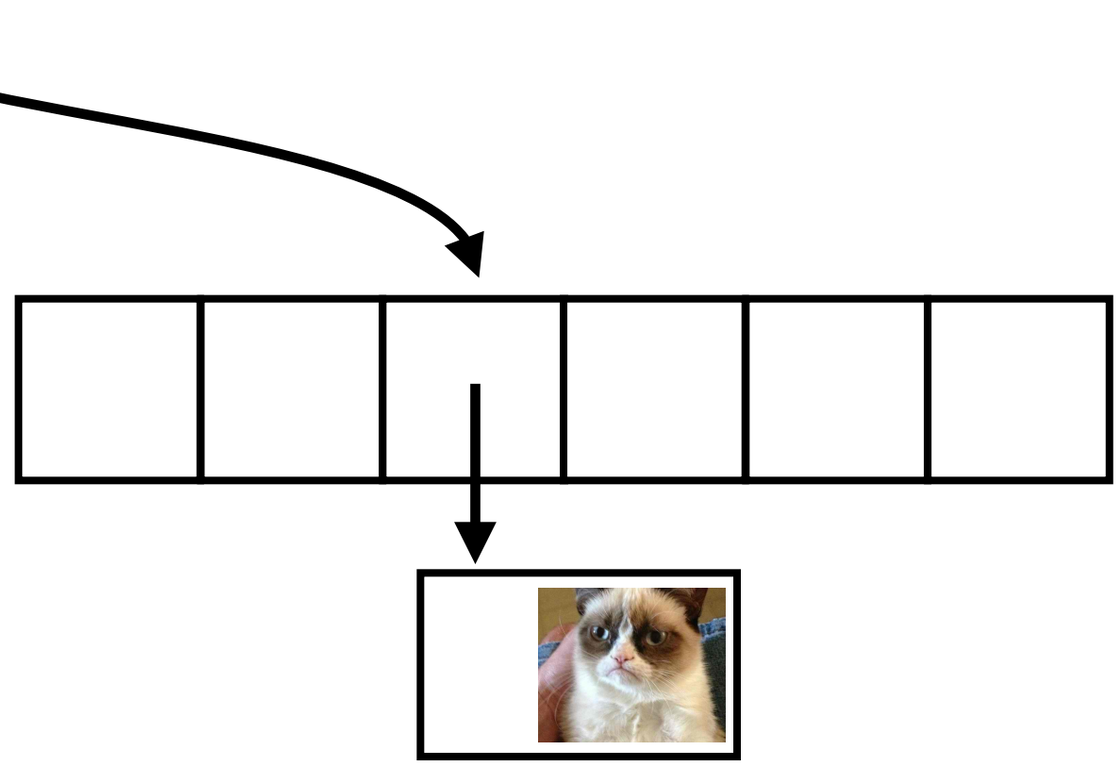
)



Dedup

Compression via deduplication

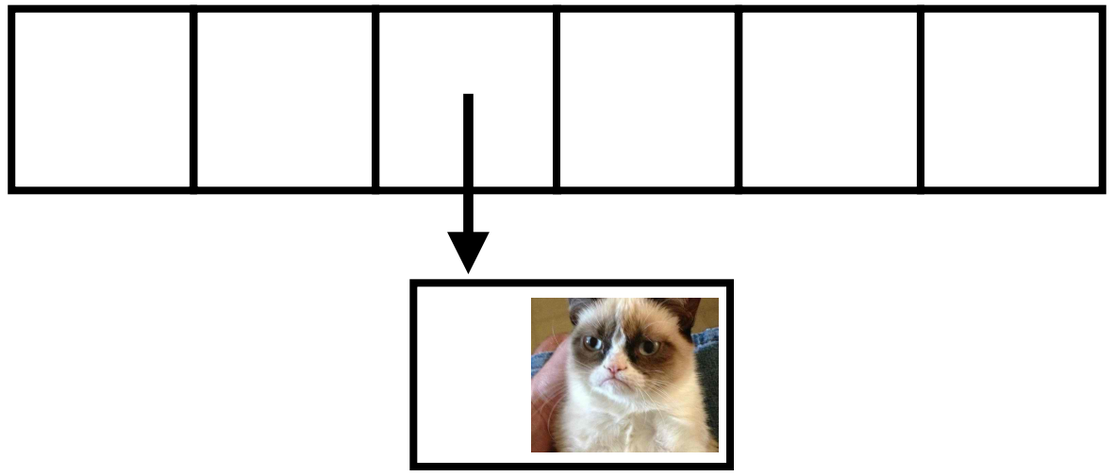
```
i = hash_function(    )
```



Dedup

Compression via deduplication

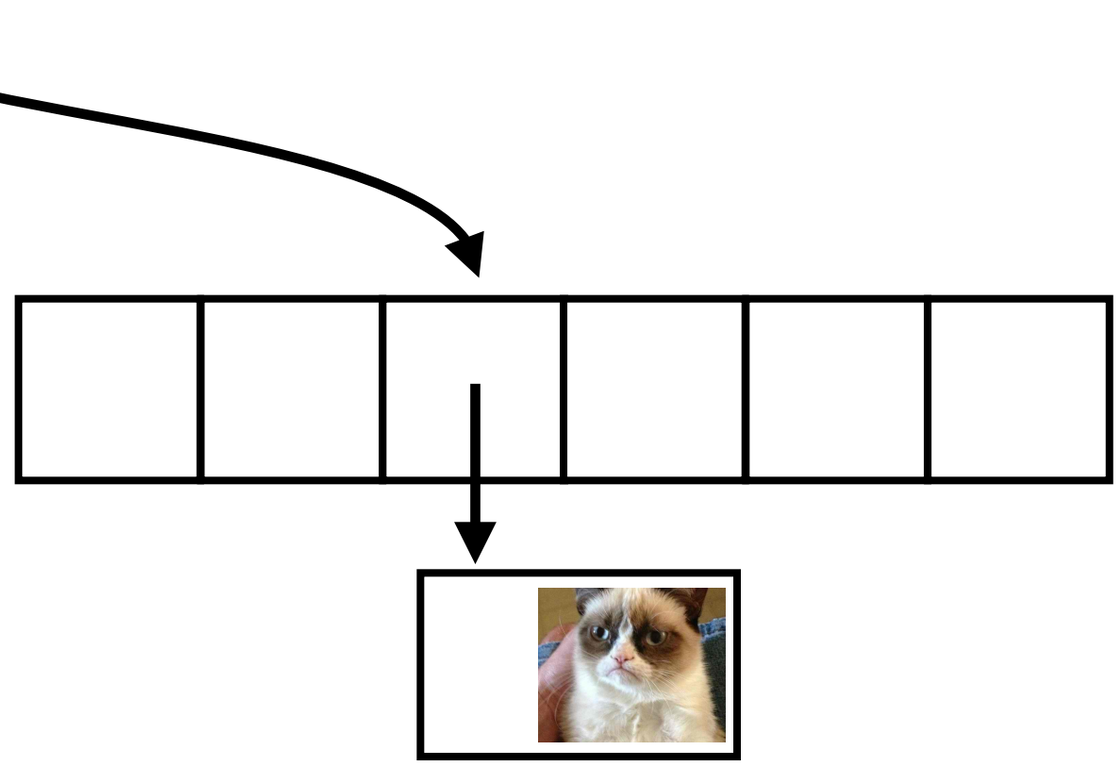
`hash_function(`  `)`



Dedup

Compression via deduplication

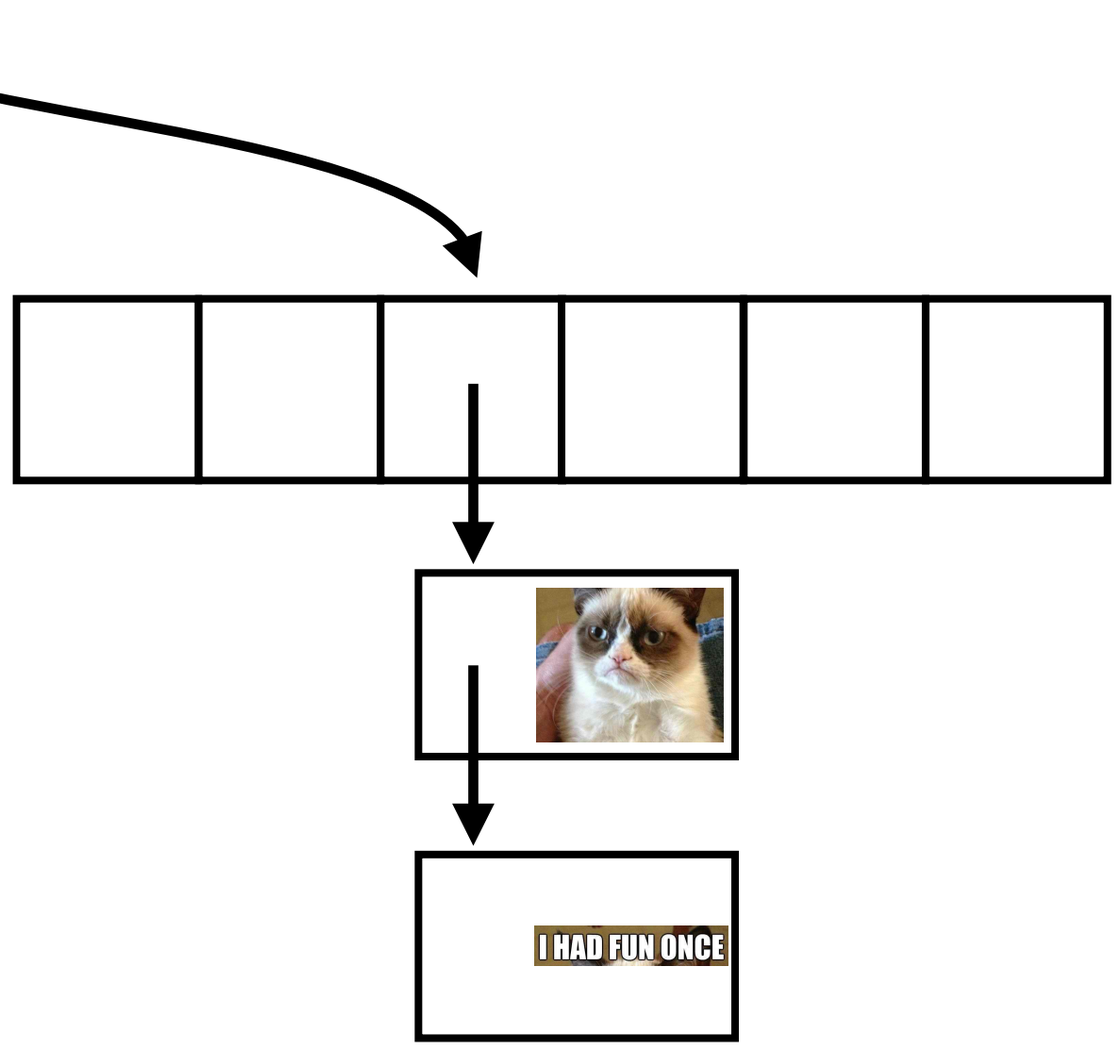
```
i = hash_function( I HAD FUN ONCE )
```



Dedup

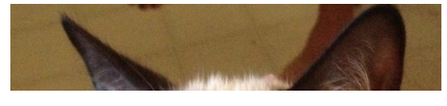
Compression via deduplication

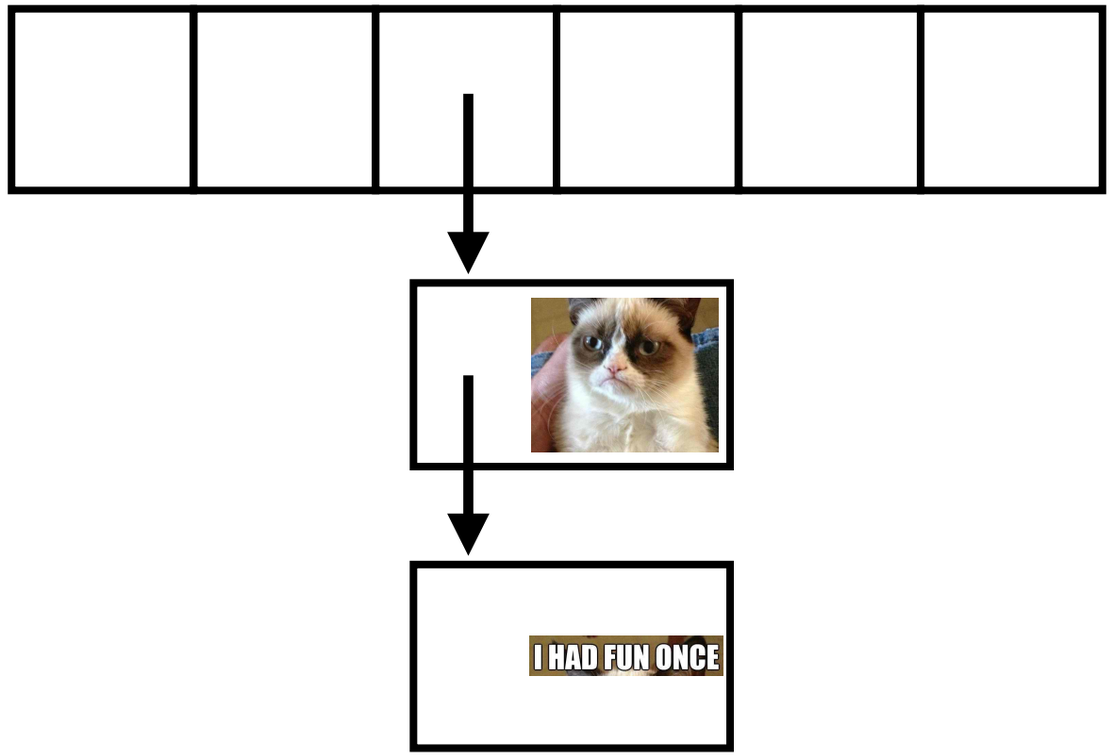
```
i = hash_function(          )
```



Dedup

Compression via deduplication

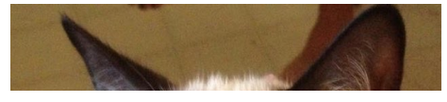
`hash_function(`  `)`

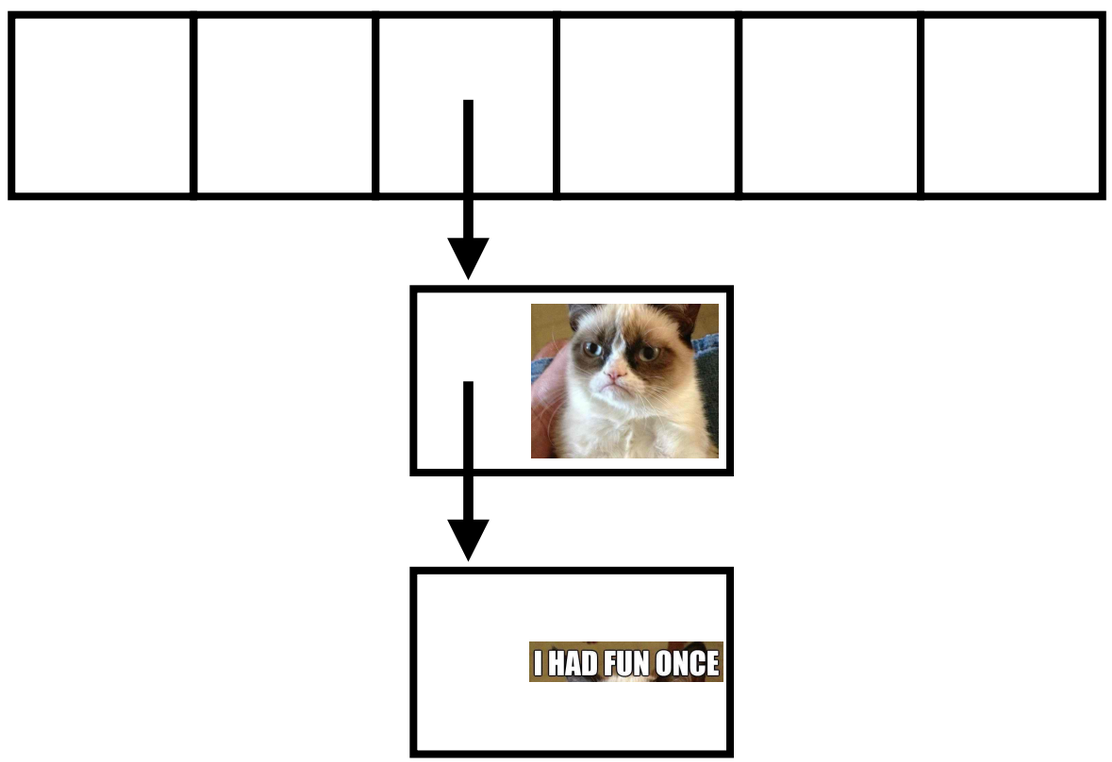


Dedup

Compression via deduplication

```
i = hash_function(
```

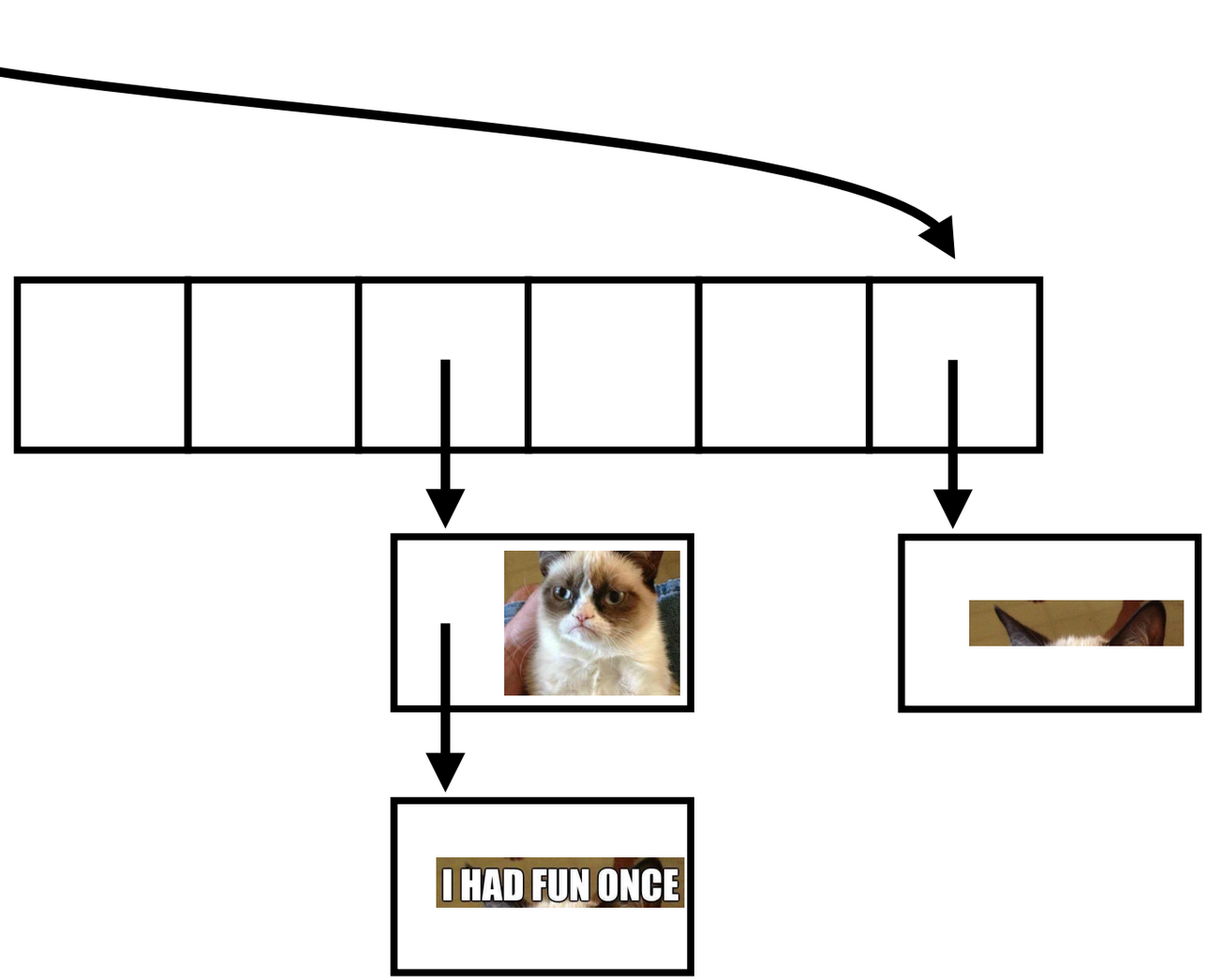




Dedup

Compression via deduplication

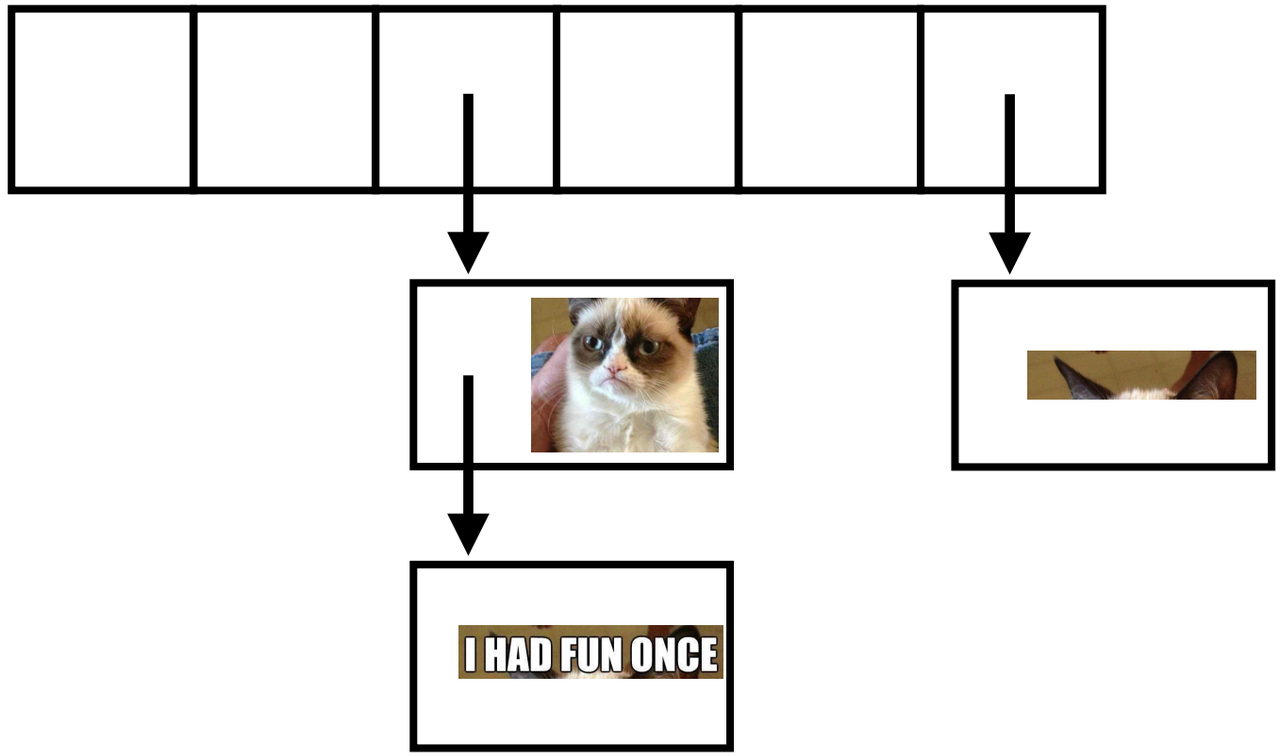
```
i = hash_function(          )
```



Dedup

Compression via deduplication

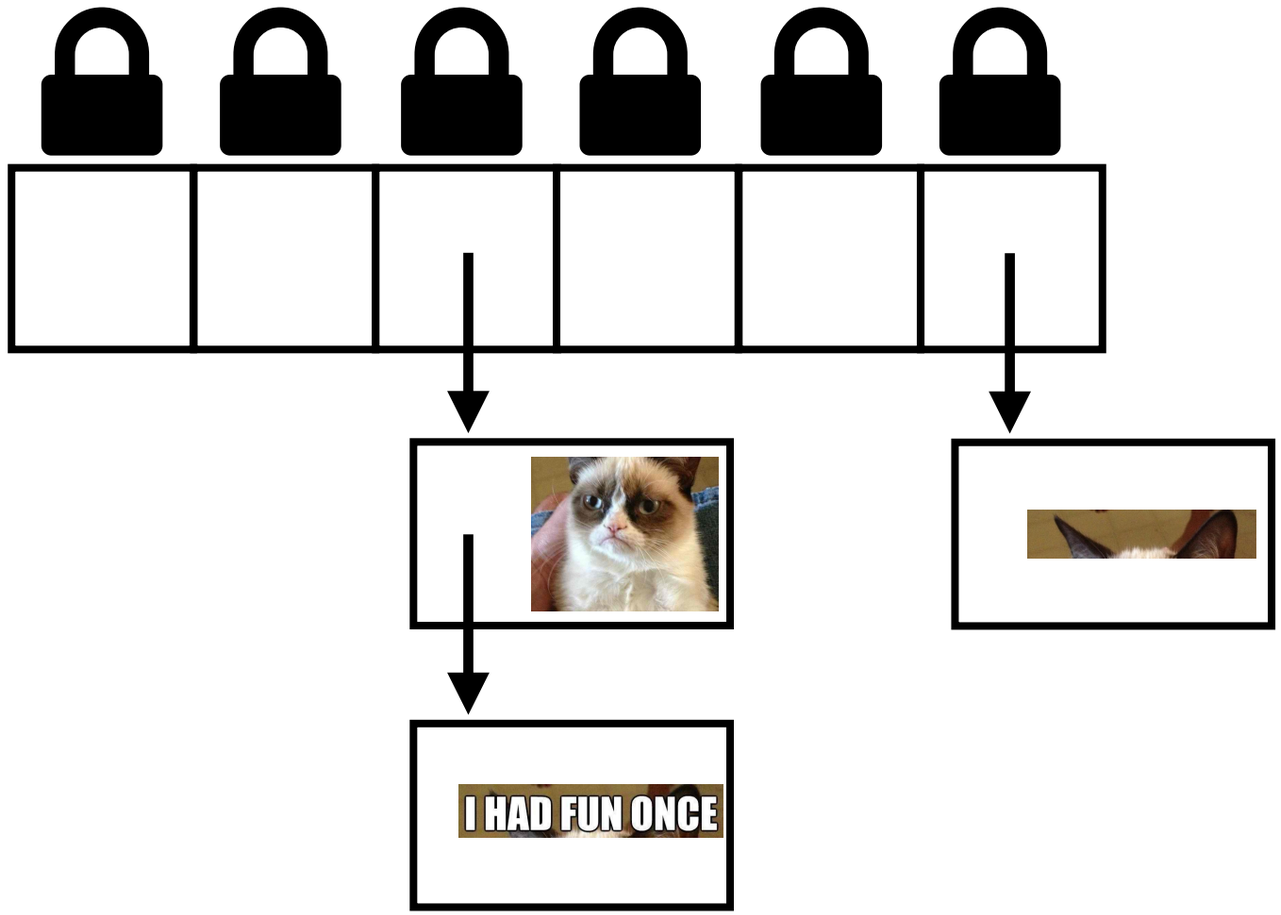
Hash table is accessed concurrently by many threads



Dedup

Compression via deduplication

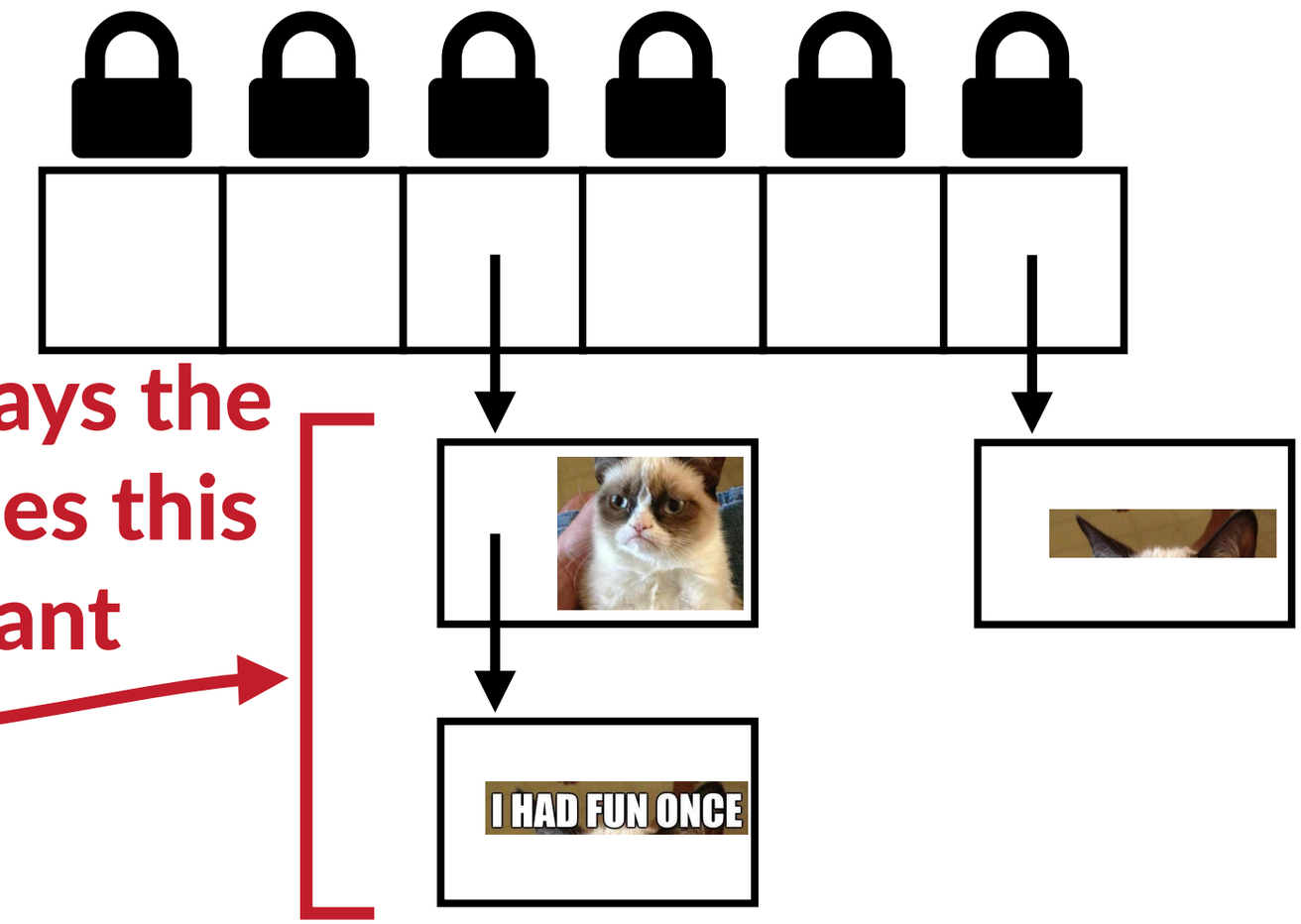
Hash table is accessed concurrently by many threads



Dedup

Compression via deduplication

Hash table is accessed concurrently by many threads

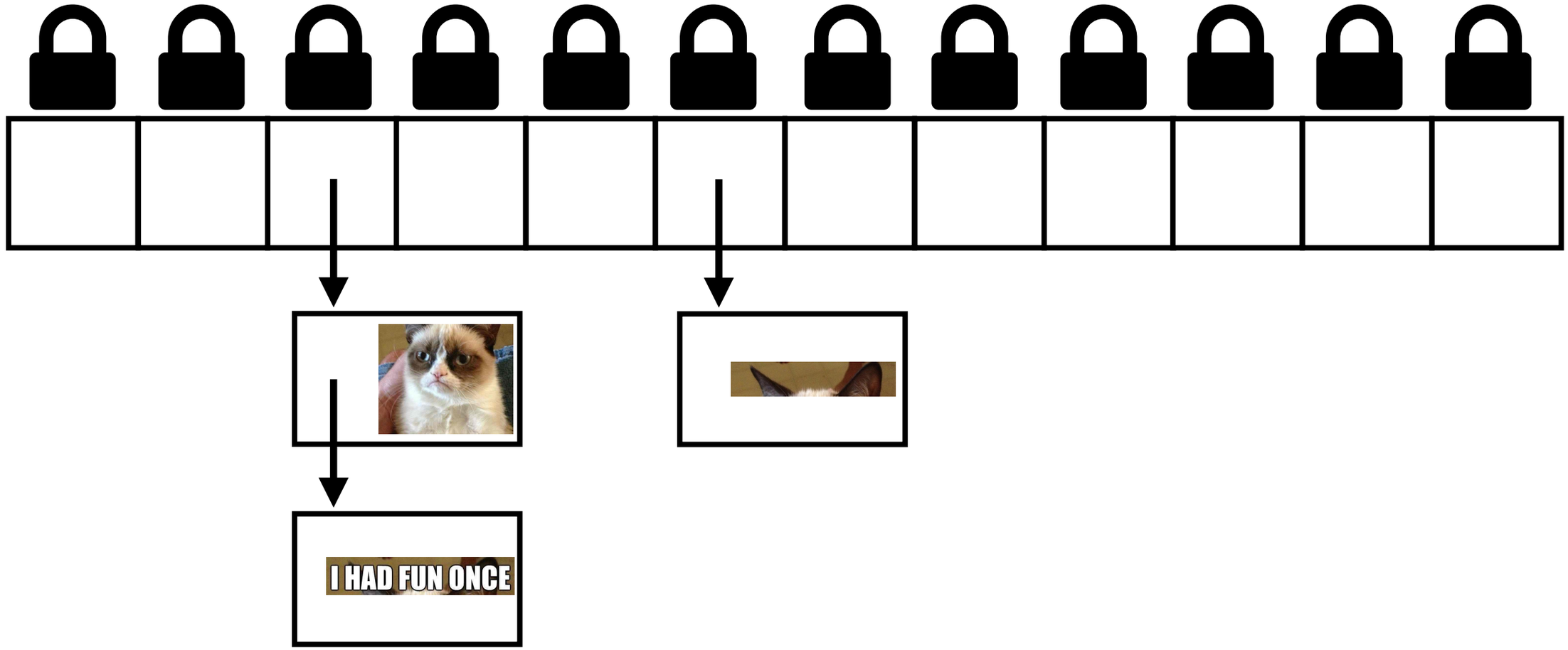


Causal Profiler says the loop that accesses this list is important

Dedup

Compression via deduplication

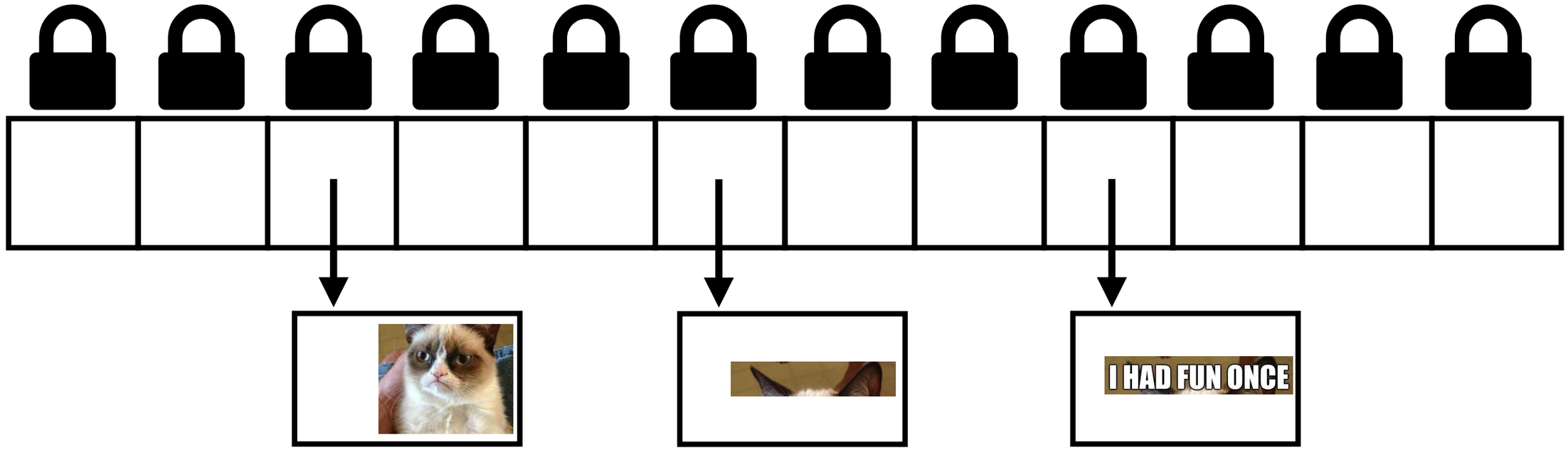
More hash buckets should lead to fewer collisions



Dedup

Compression via deduplication

More hash buckets should lead to fewer collisions

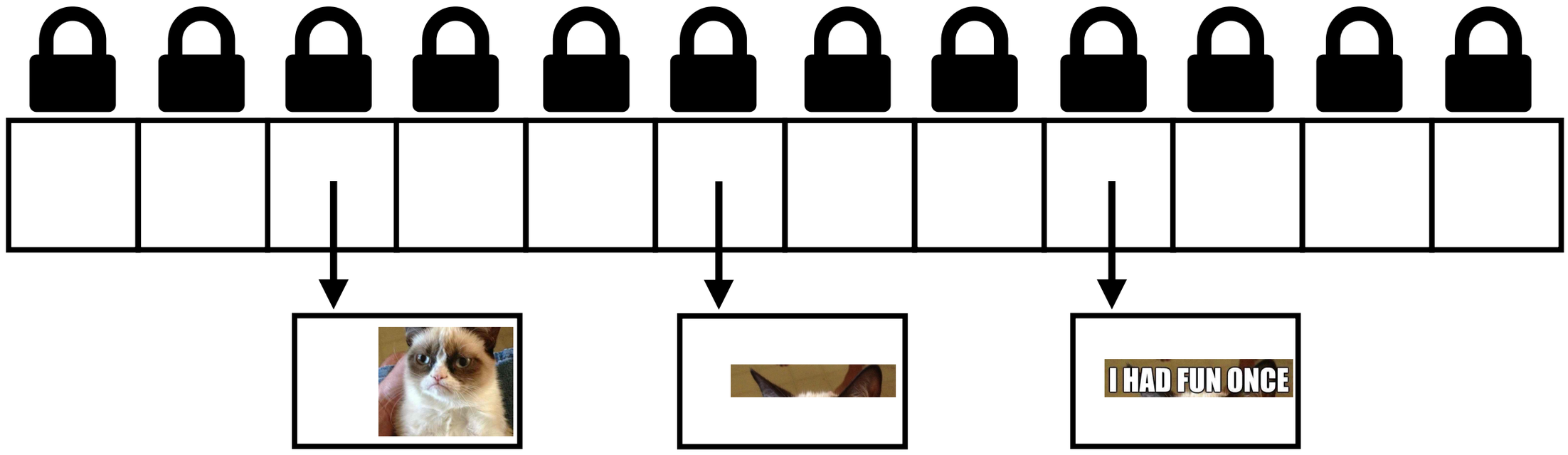


Dedup

Compression via deduplication

More hash buckets should lead to fewer collisions

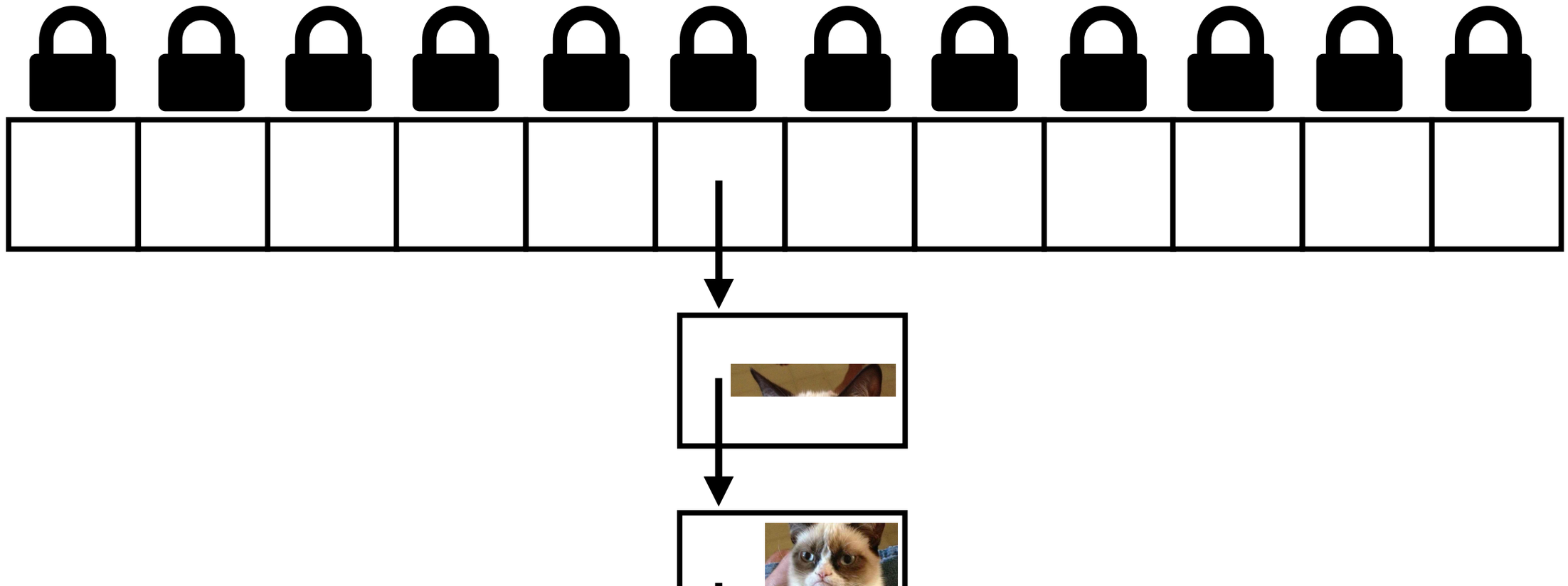
No performance improvement



Dedup

Compression via deduplication

What else could be causing collisions?

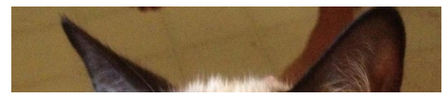


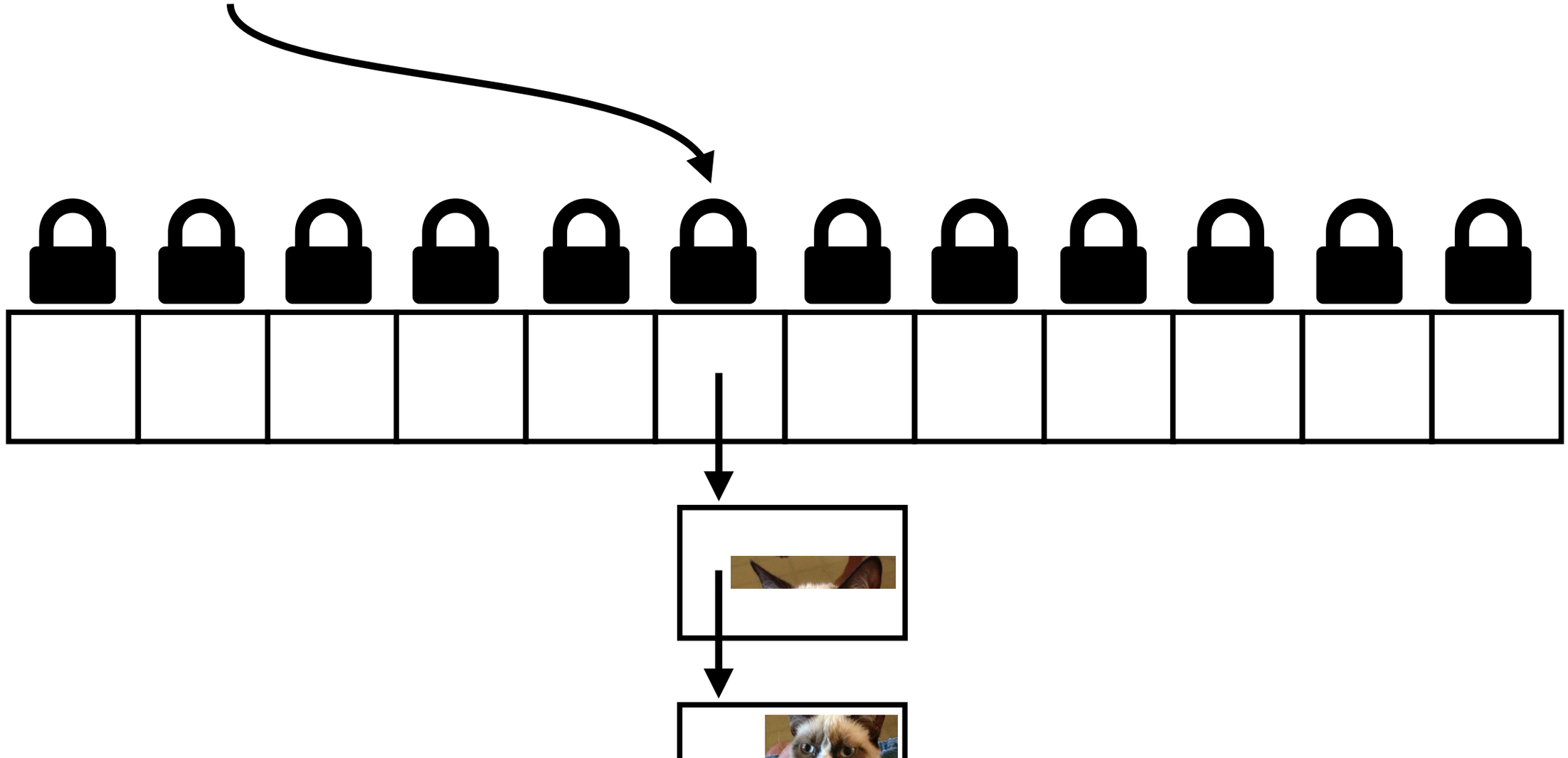
Dedup

Compression via deduplication

What else could be causing collisions?

```
i = hash_function(
```

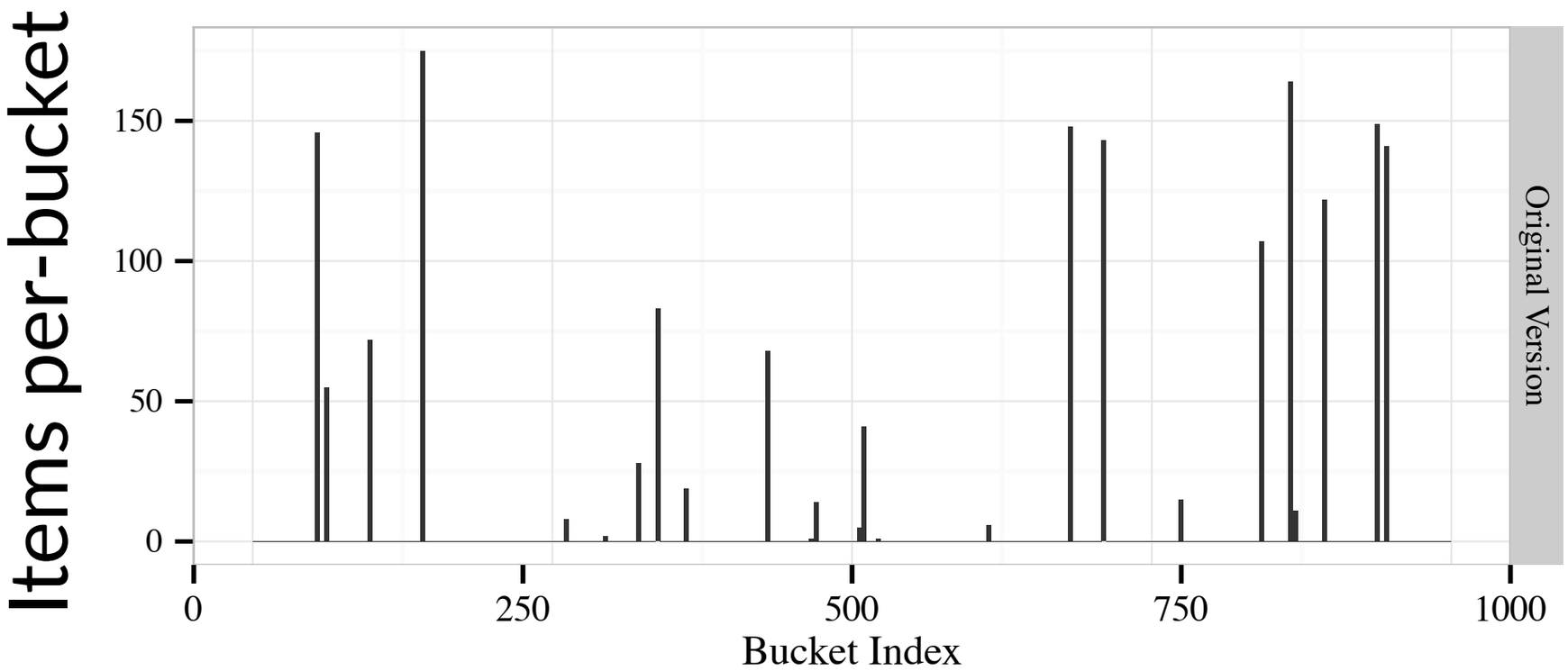




Dedup

Compression via deduplication

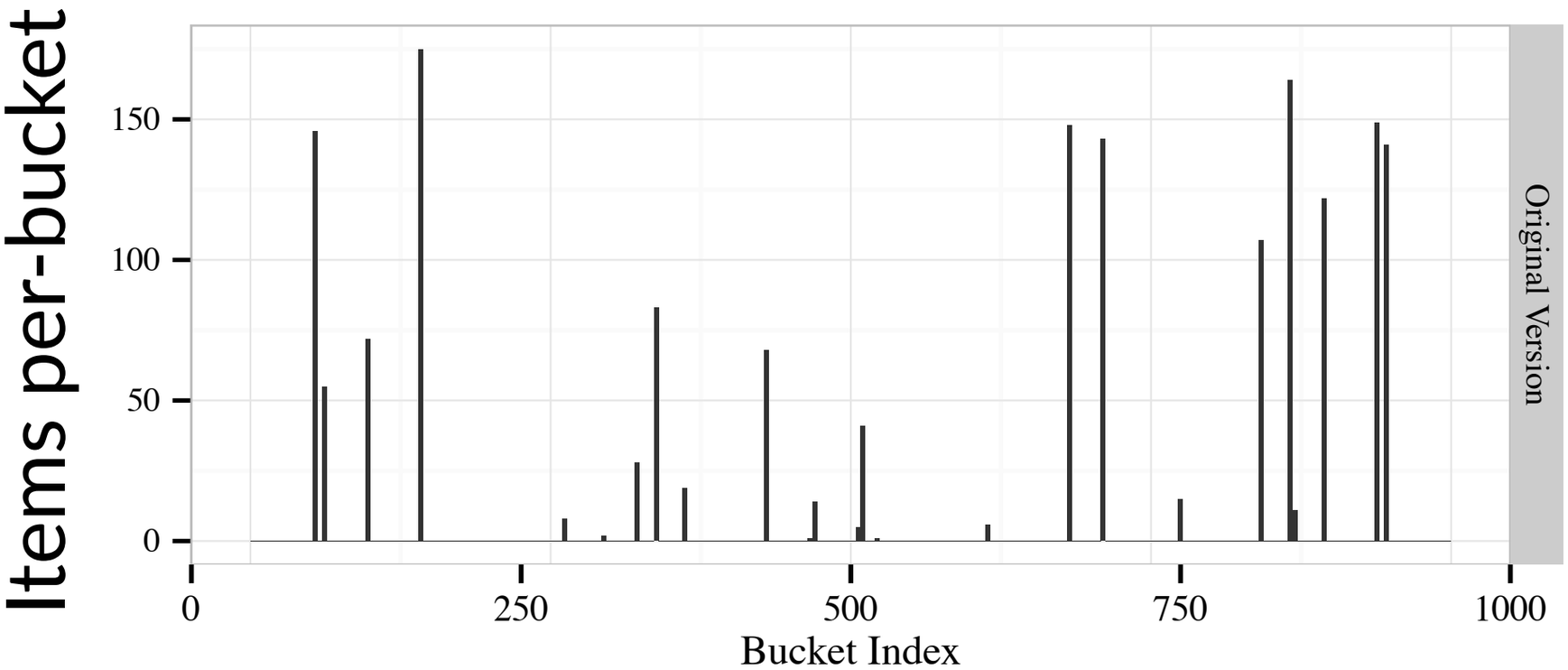
Horrible hash function!



Dedup

Compression via deduplication

Horrible hash function!

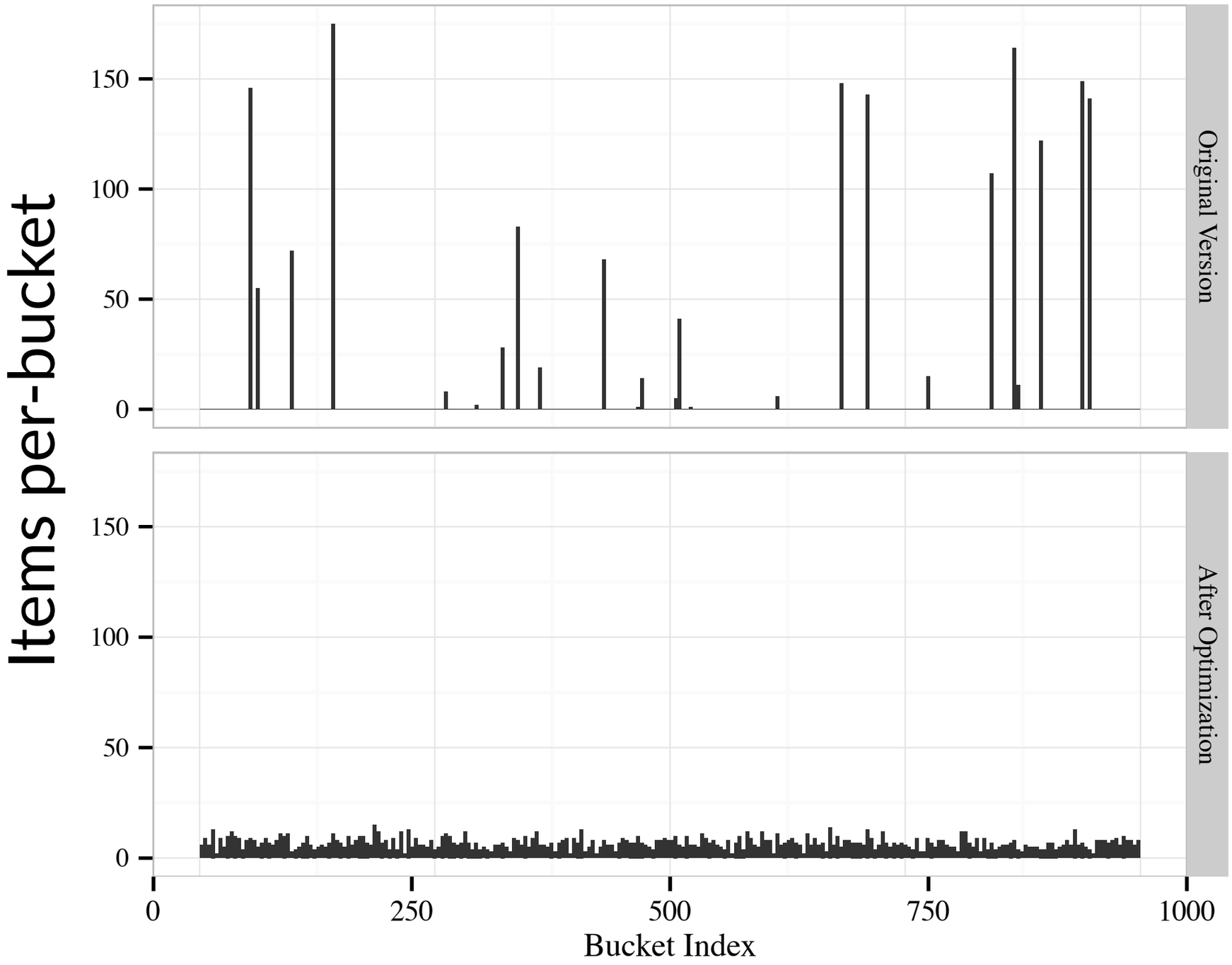


Bin Utilization

2.3%

Dedup

Compression via deduplication

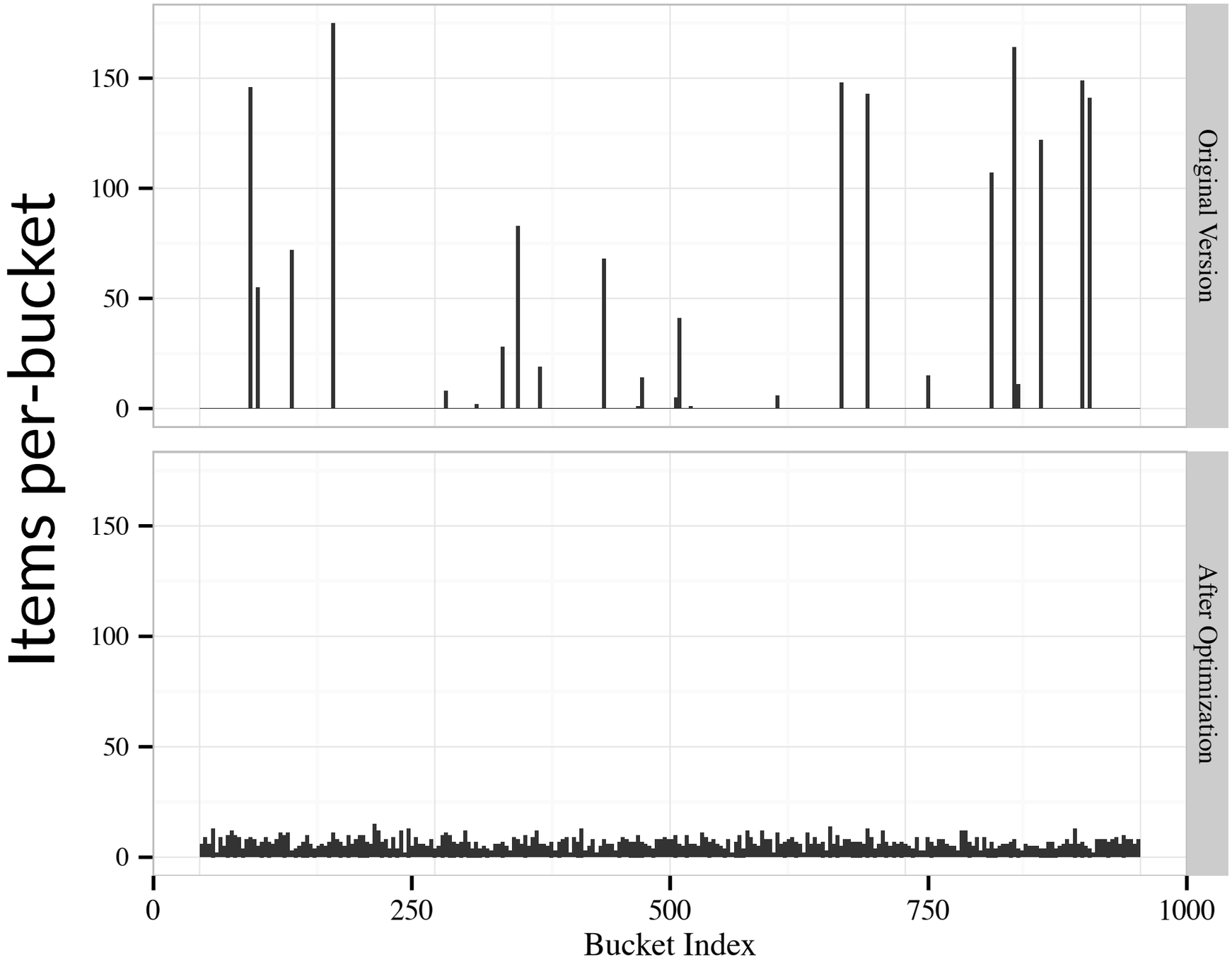


Bin Utilization

2.3%

Dedup

Compression via deduplication



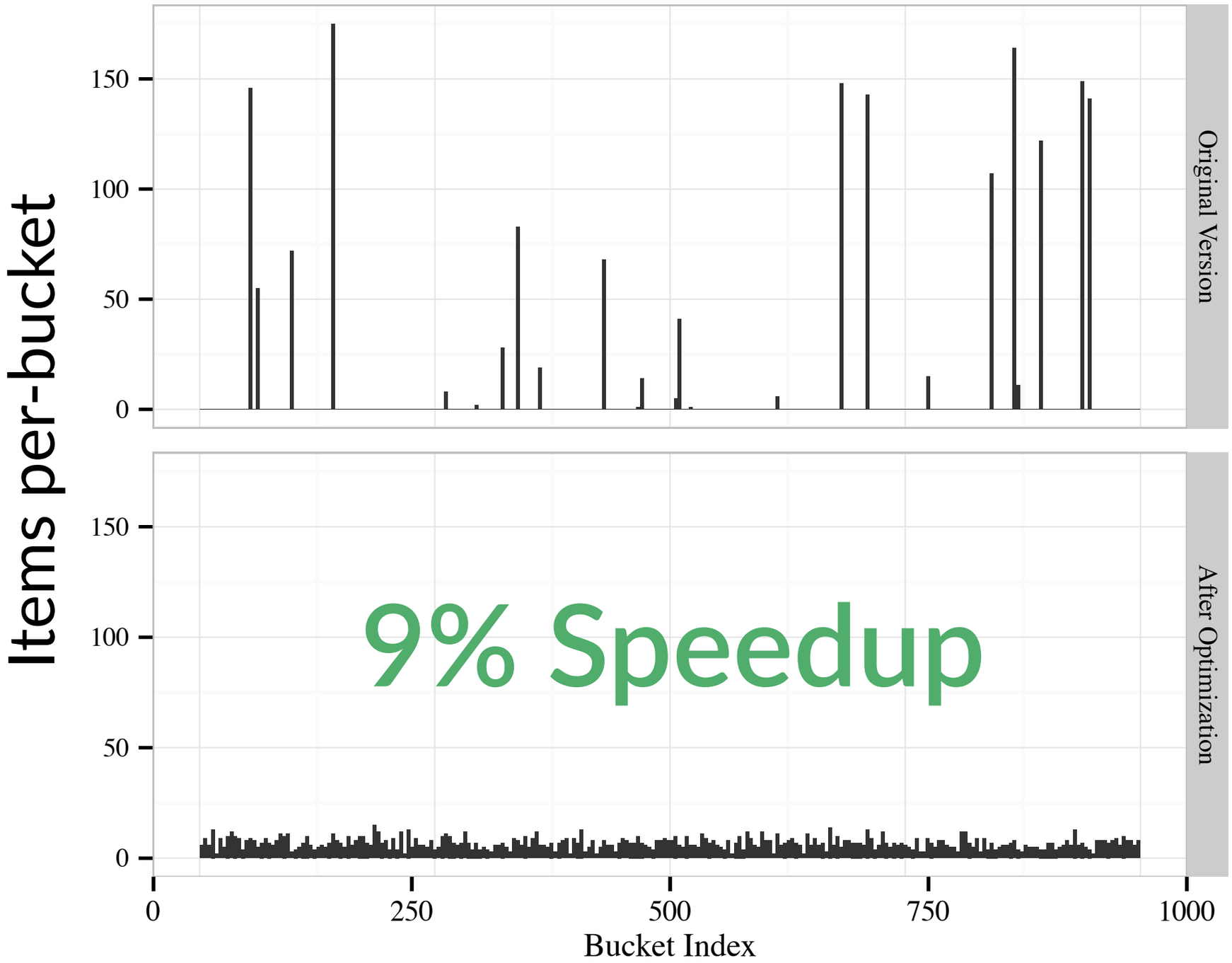
Bin Utilization

2.3%

82%

Dedup

Compression via deduplication



Bin Utilization

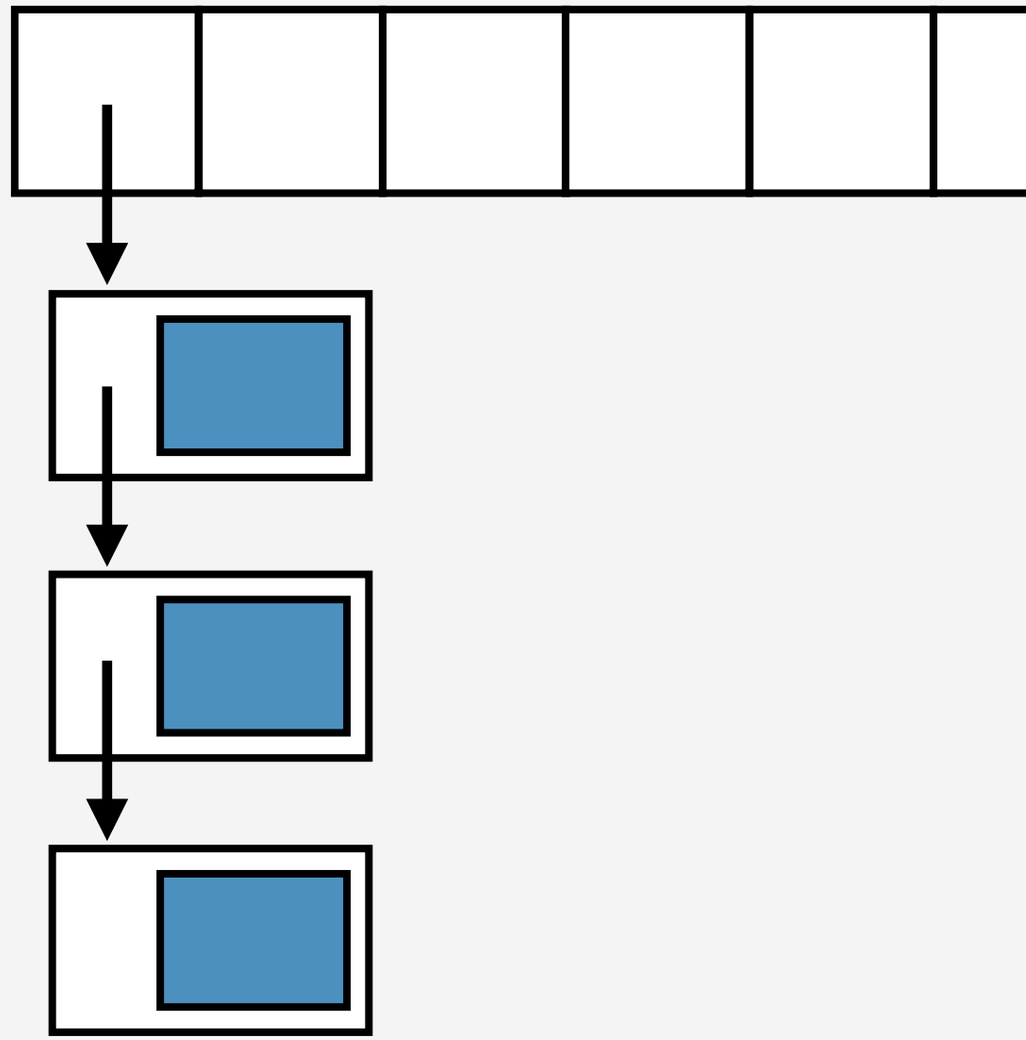
2.3%

82%

Dedup

Compression via deduplication

What did Causal Profiling predict?

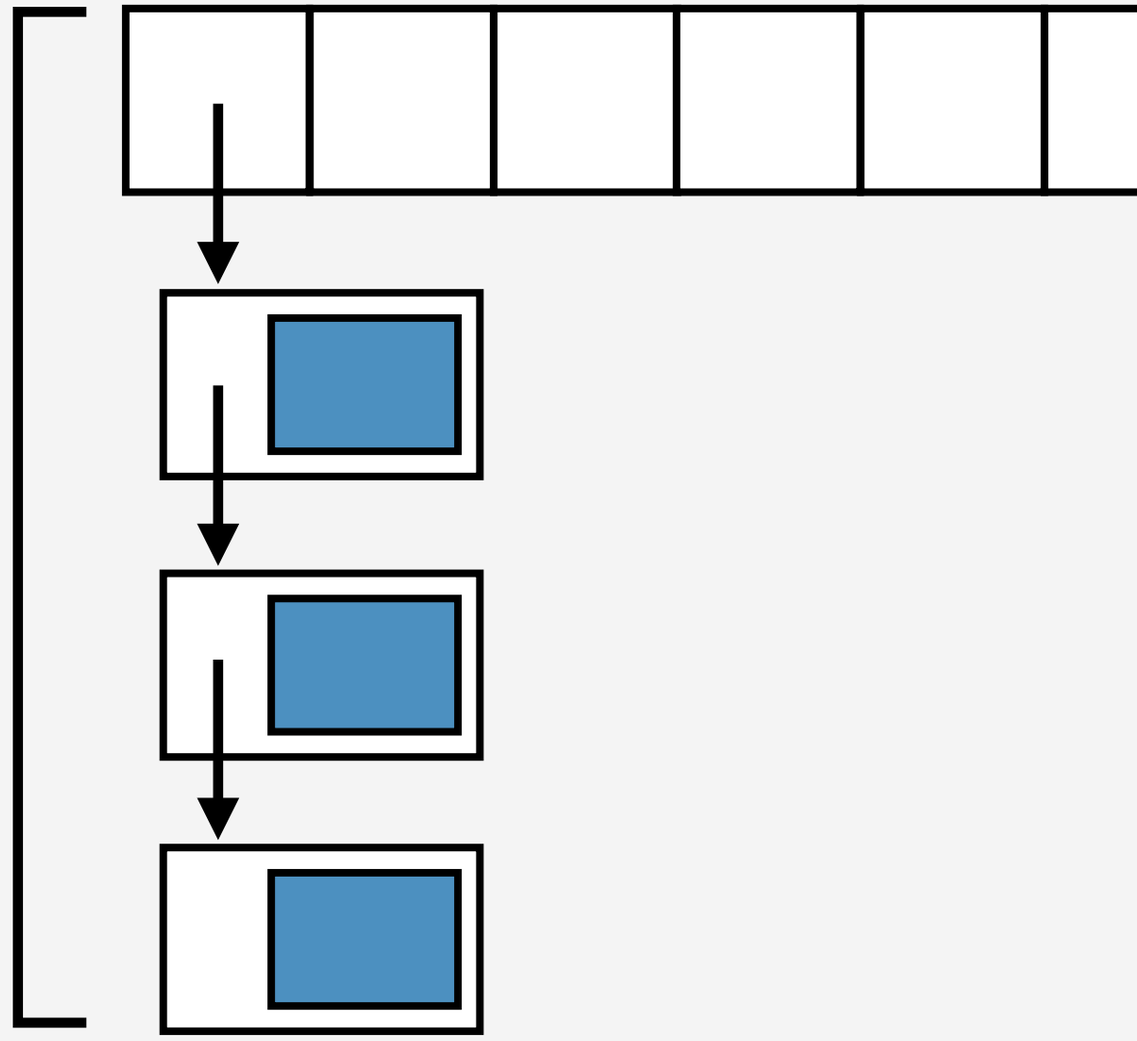


Dedup

Compression via deduplication

What did Causal Profiling predict?

Blocks per-bucket



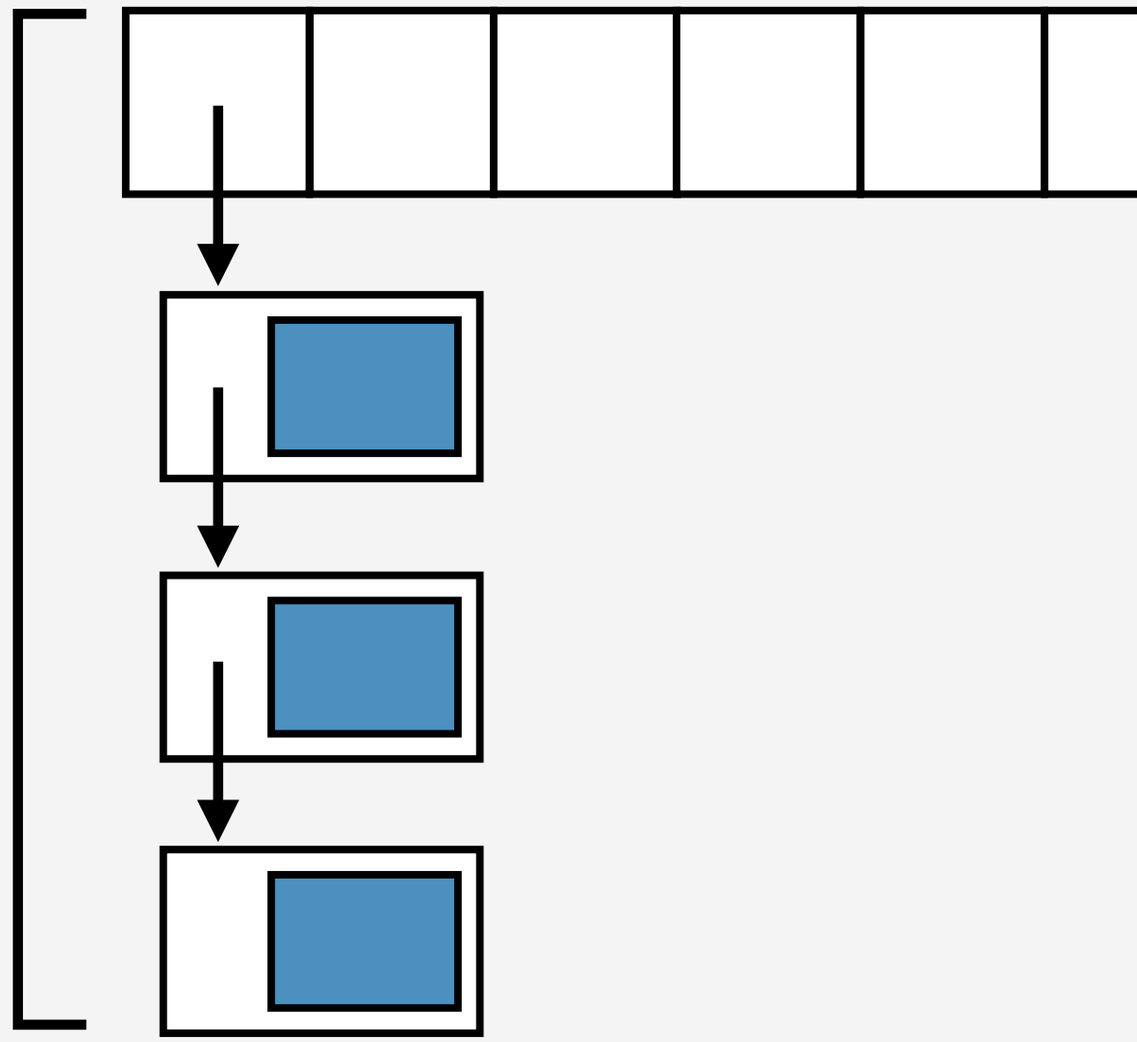
Dedup

Compression via deduplication

What did Causal Profiling predict?

Blocks per-bucket

Before: 76.7



Dedup

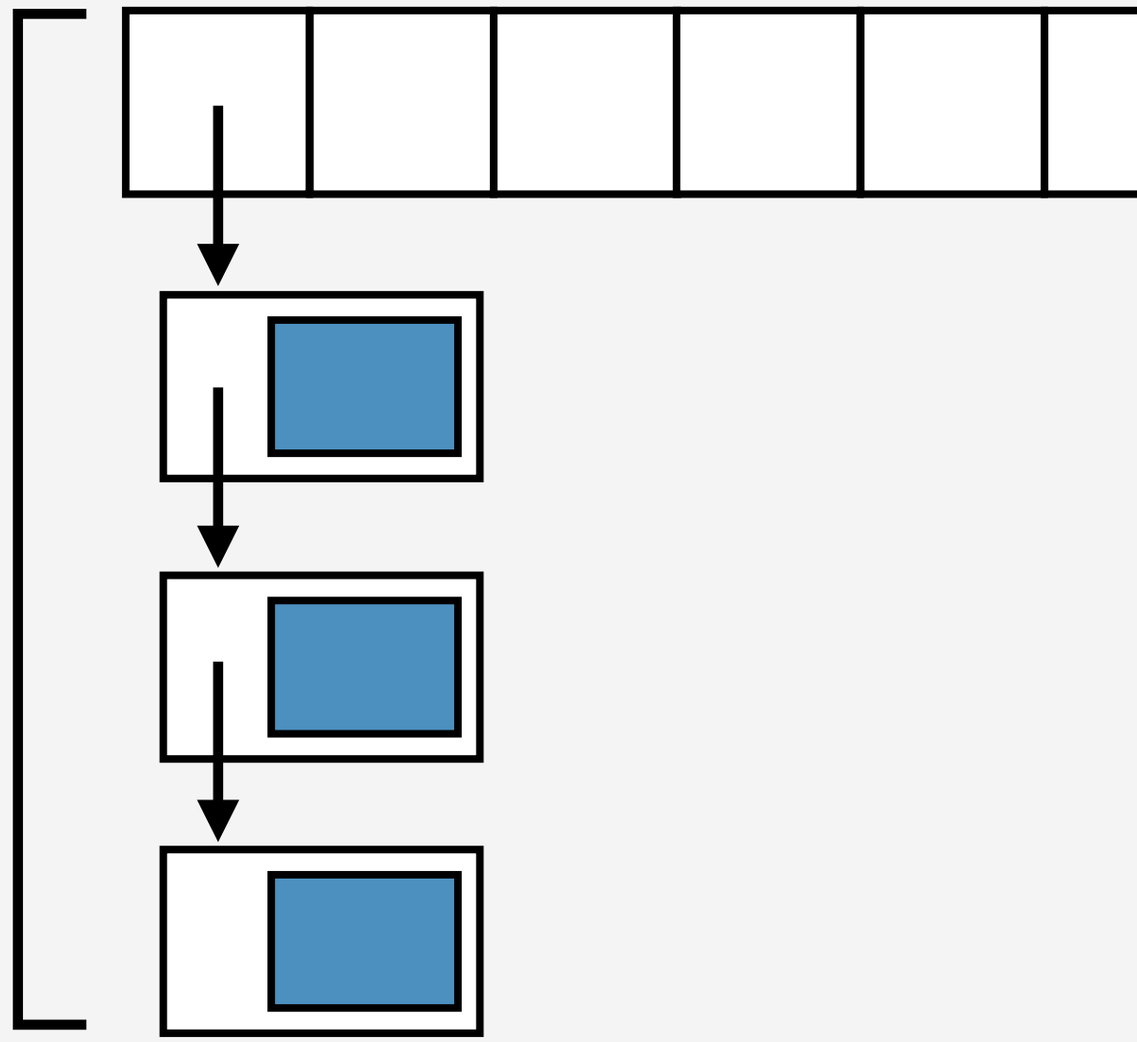
Compression via deduplication

What did Causal Profiling predict?

Blocks per-bucket

Before: 76.7

After: 2.09



Dedup

Compression via deduplication

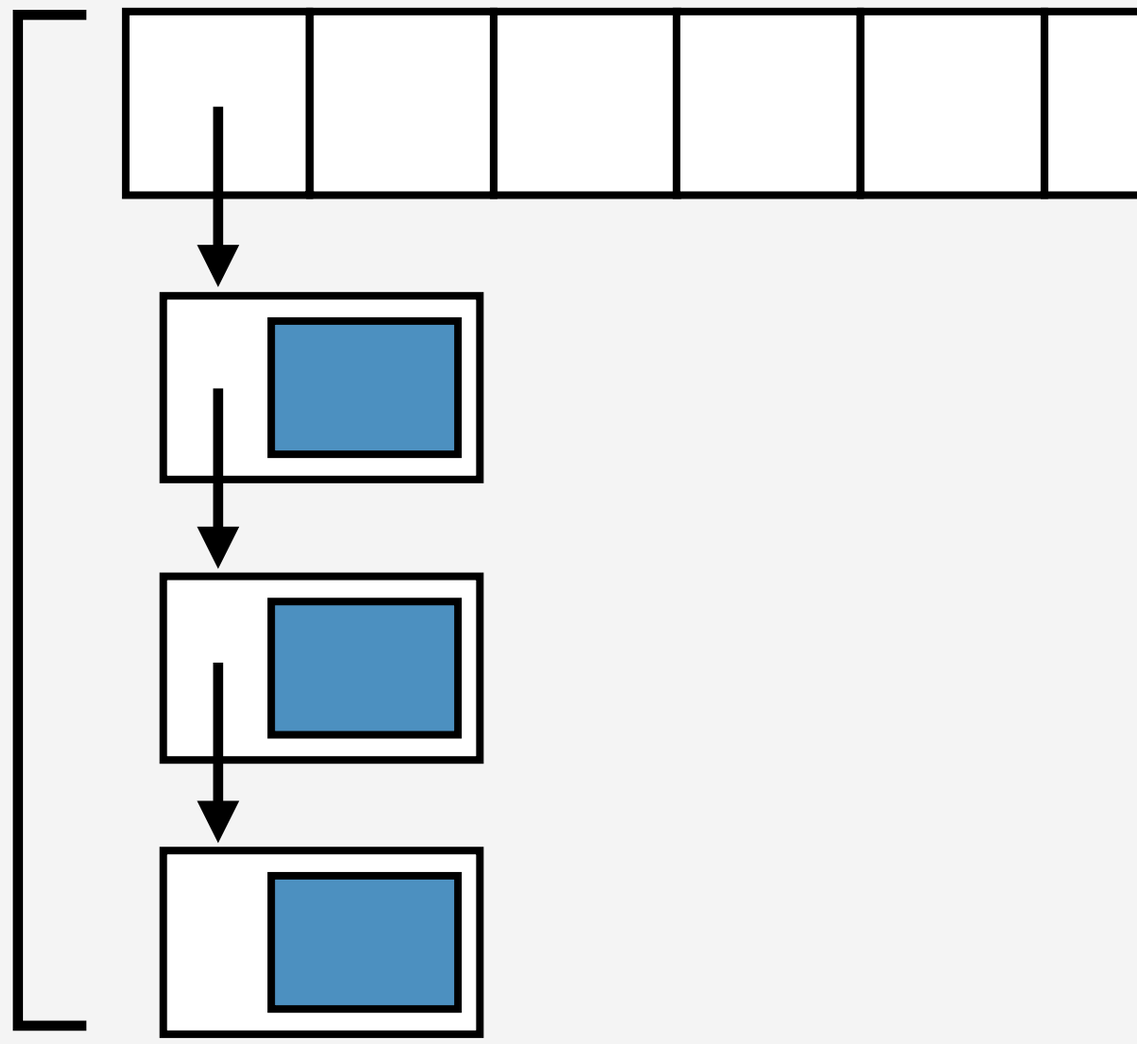
What did Causal Profiling predict?

Blocks per-bucket

Before: 76.7

After: 2.09

96% traversal speedup



Dedup

Compression via deduplication

What did Causal Profiling predict?

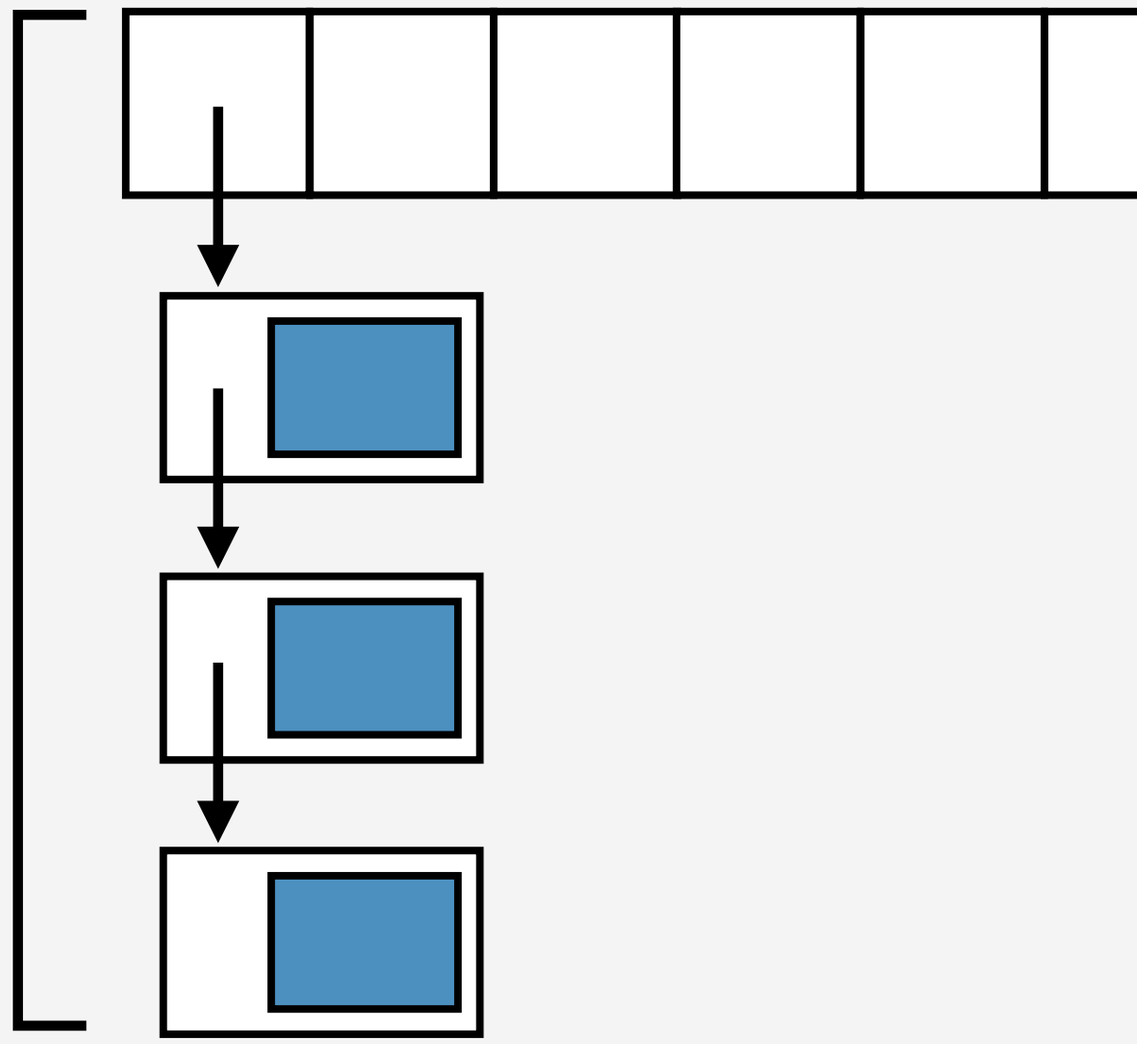
Blocks per-bucket

Before: 76.7

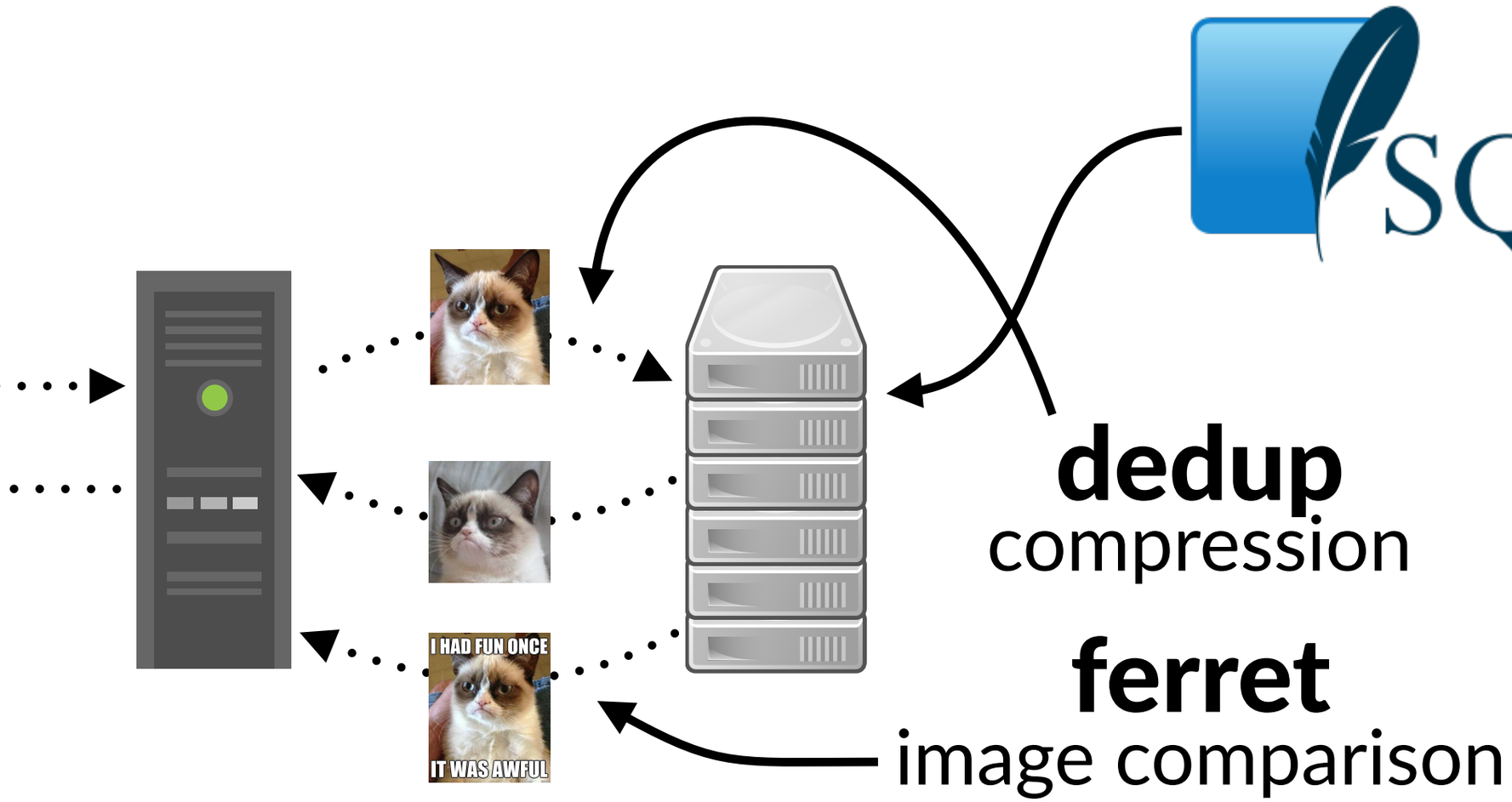
After: 2.09

96% traversal speedup

9% predicted speedup, exactly what we observed



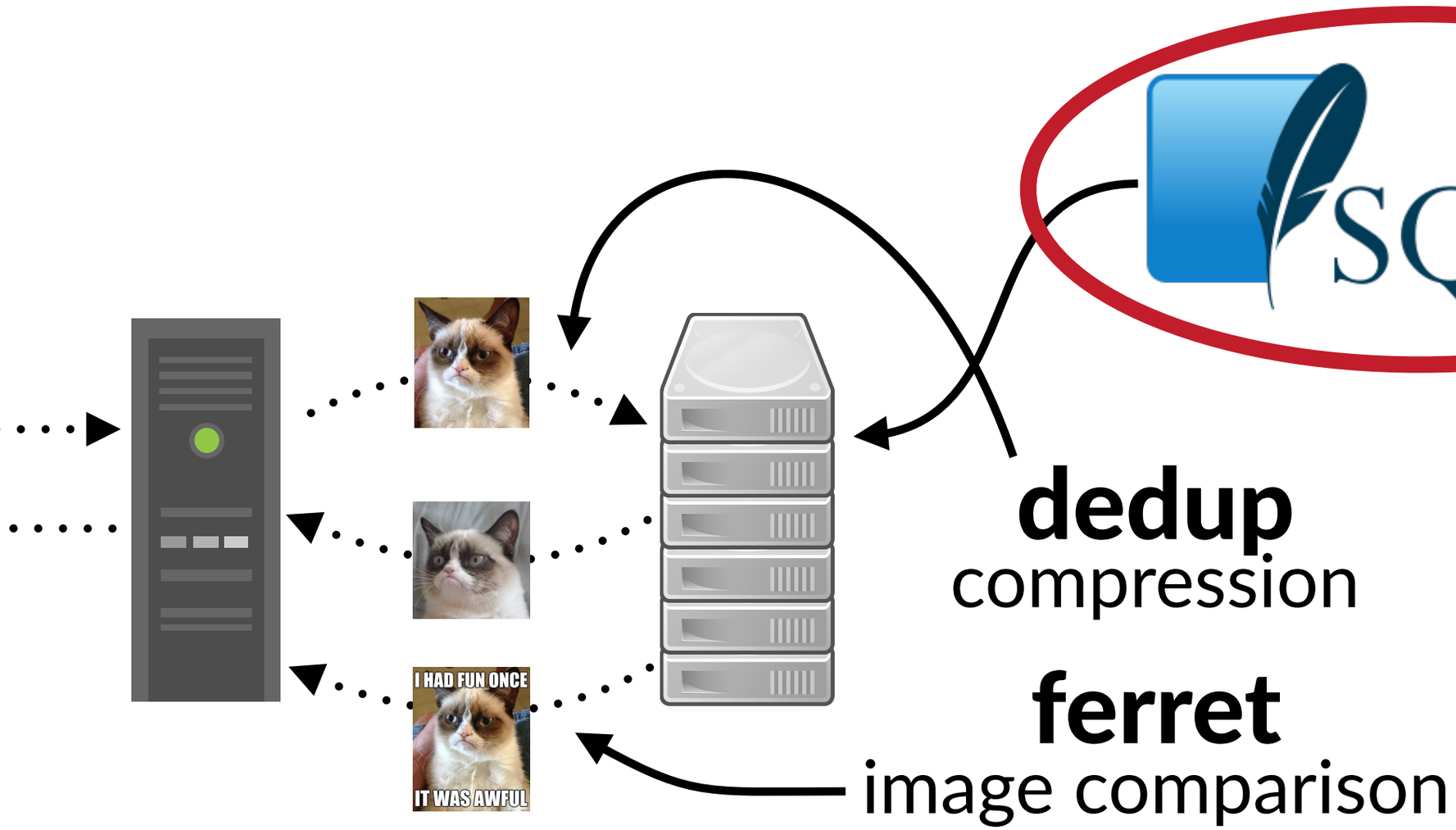
Using Causal Profiling on Ogle



dedup
compression

ferret
image comparison

Using Causal Profiling on Ogle



dedup
compression

ferret
image comparison


SQLite

Simple SQL Database

Simple SQL Database

```
#if THREAD_SAFE
```

Simple SQL Database

```
#if THREAD_SAFE
config_t global_config = {
```

Simple SQL Database

```
#if THREAD_SAFE
config_t global_config = {
  ...
```

Simple SQL Database

```
#if THREAD_SAFE
config_t global_config = {
    ...
    .unlock = pthread_mutex_unlock,
```

Simple SQL Database

```
#if THREAD_SAFE
config_t global_config = {
    ...
    .unlock = pthread_mutex_unlock,
    .getsize = sqlite_usable_size,
```

Simple SQL Database

```
#if THREAD_SAFE
config_t global_config = {
    ...
    .unlock = pthread_mutex_unlock,
    .getsize = sqlite_usable_size,
    .nextitem = sqlite_pagecache_next,
```

Simple SQL Database

```
#if THREAD_SAFE
config_t global_config = {
  ...
    .unlock = pthread_mutex_unlock,
    .getsize = sqlite_usable_size,
    .nextitem = sqlite_pagecache_next,
  ...
};
#endif
```

Simple SQL Database

```
void sqlite_unlock(lock* l) {
   global_config.unlock(l);
}
```

Simple SQL Database

```
void sqlite_unlock(lock* l) {
   global_config.unlock(l);
}
```

Indirect Call

Simple SQL Database

```
void sqlite_unlock(lock* l) {
   global_config.unlock(l);
}
```

Indirect Call

Cheap, but almost the same cost as `pthread_mutex_unlock`

Simple SQL Database

```
void sqlite_unlock(lock* l) {
   global_config.unlock(l);
}
```

Simple SQL Database

```
void sqlite_unlock(lock* l) {
  global_config.unlock(l);
}
```

```
void sqlite_getsize(void* p) {
  global_config.getsize(p);
}
```

Simple SQL Database

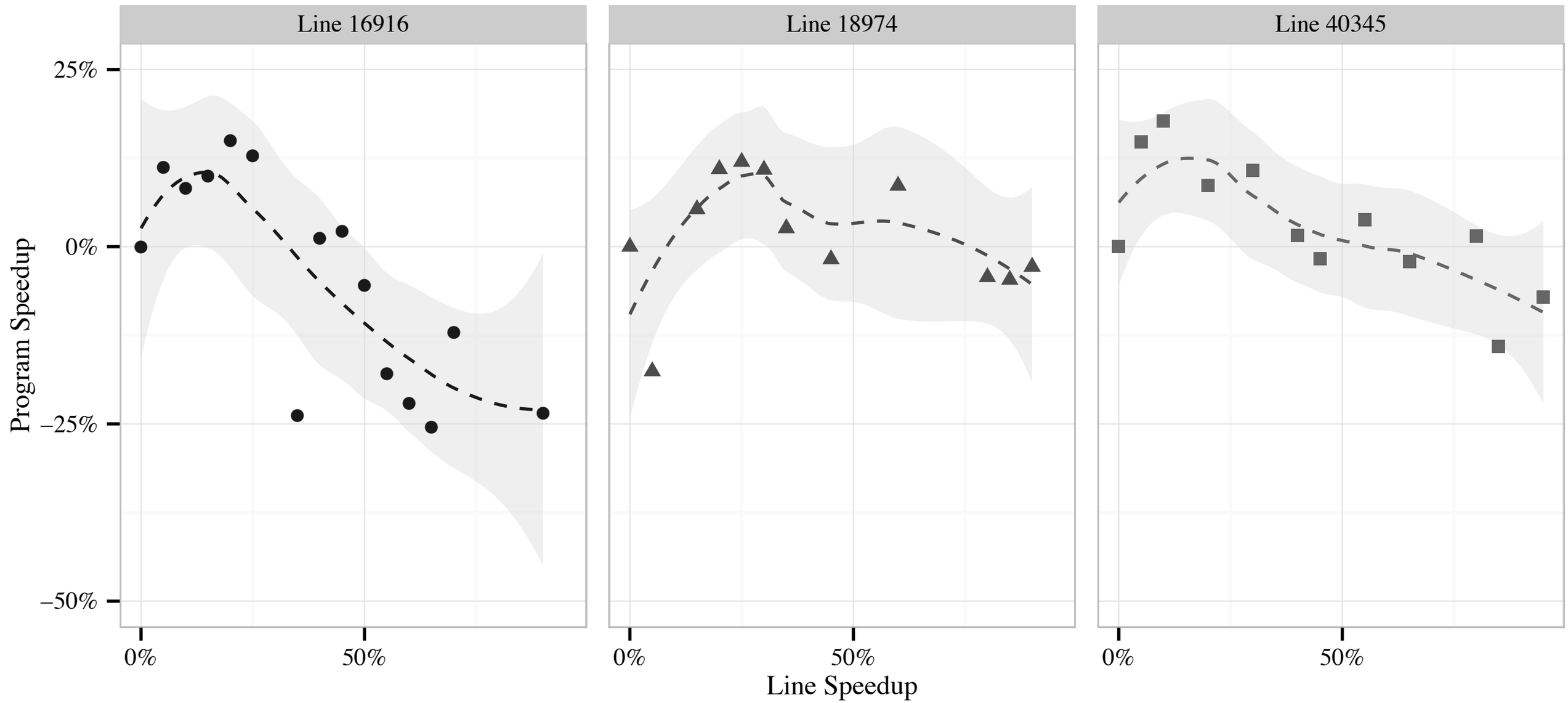
```
void sqlite_unlock(lock* l) {
  global_config.unlock(l);
}

void sqlite_getsize(void* p) {
  global_config.getsize(p);
}

void sqlite_nextitem(item* i) {
  global_config.nextitem(i);
}
```

Simple SQL Database

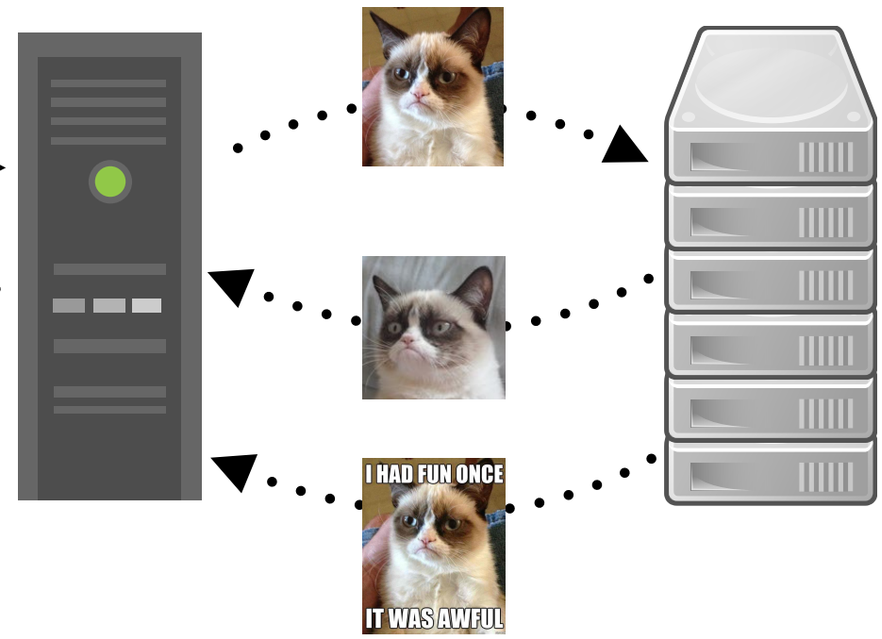
```
void sqlite_unlock(lock* l) {
  global_config.unlock(l);
}

void sqlite_getsize(void* p) {
  global_config.getsize(p);
}

void sqlite_nextitem(item* i) {
  global_config.nextitem(i);
}
```

Coz highlights these lines

Simple SQL Database



Using Causal Profiling on Ogle

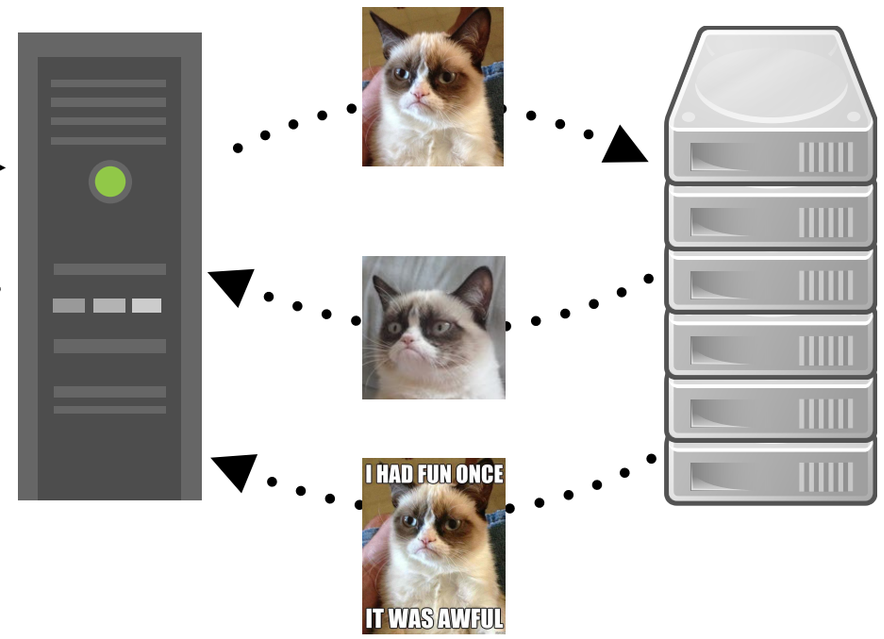


SQLite

dedup
compression

ferret
image comparison

Using Causal Profiling on Ogle

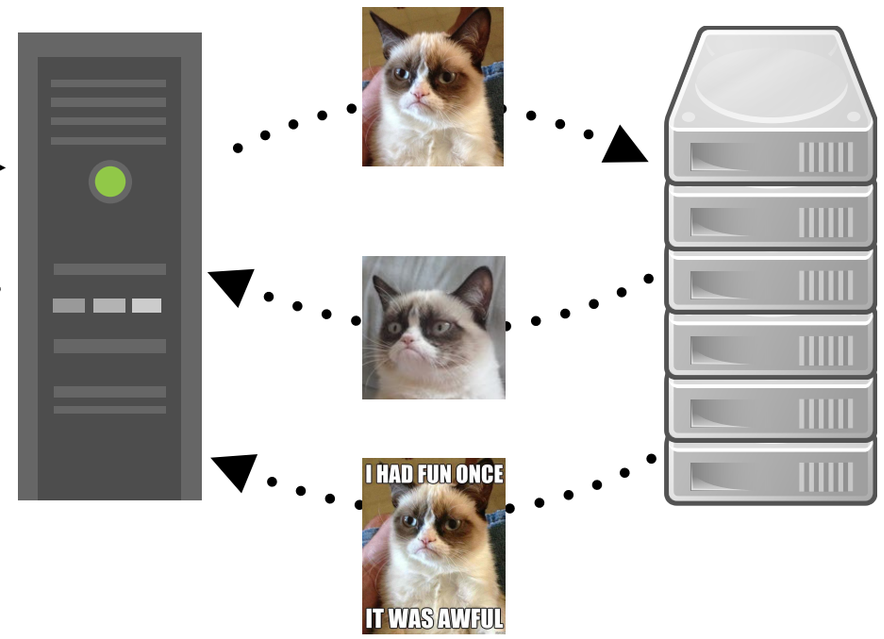


SQLite 25%

dedup
compression

ferret
image comparison

Using Causal Profiling on Ogle

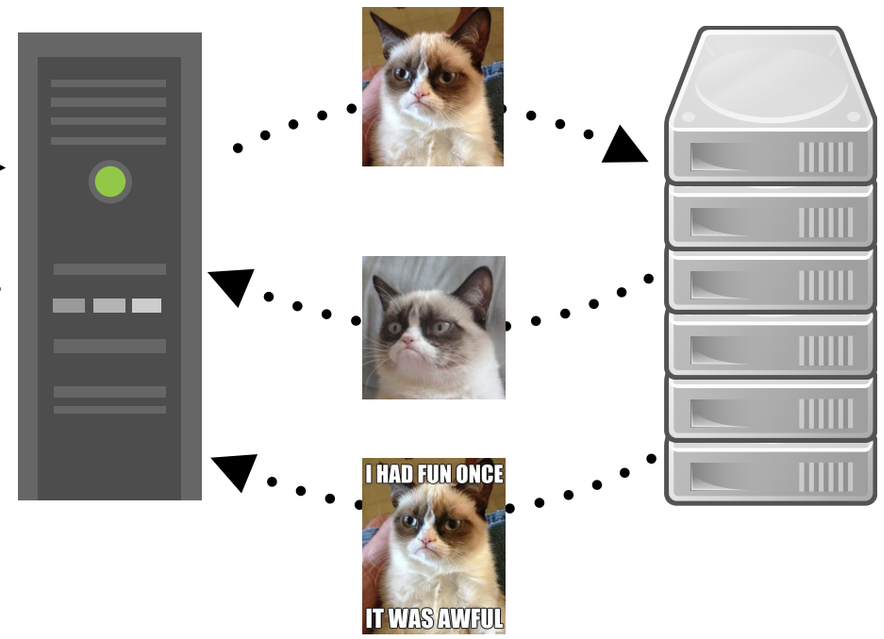


SQLite 25%

dedup
compression 9%

ferret
image comparison

Using Causal Profiling on Ogle



SQLite 25%

dedup
compression 9%

ferret
image comparison 21%

Summary of Optimizations

Benchmark	Speedup	Diff Size	Change Summary

Summary of Optimizations

Benchmark	Speedup	Diff Size	Change Summary
memcached	9.39%	-6, +2	removed unnecessary locks

Summary of Optimizations

Benchmark	Speedup	Diff Size	Change Summary
memcached	9.39%	-6, +2	removed unnecessary locks
sqlite	25.60%	-3, +3	removed DIY vtable implementation

Summary of Optimizations

Benchmark	Speedup	Diff Size	Change Summary
memcached	9.39%	-6, +2	removed unnecessary locks
sqlite	25.60%	-3, +3	removed DIY vtable implementation
blackscholes	2.56%	-61, +4	manual common subexpression elimination

Summary of Optimizations

Benchmark	Speedup	Diff Size	Change Summary
memcached	9.39%	-6, +2	removed unnecessary locks
sqlite	25.60%	-3, +3	removed DIY vtable implementation
blackscholes	2.56%	-61, +4	manual common subexpression elimination
dedup	8.95%	-3, +3	fixed degenerate hash function

Summary of Optimizations

Benchmark	Speedup	Diff Size	Change Summary
memcached	9.39%	-6, +2	removed unnecessary locks
sqlite	25.60%	-3, +3	removed DIY vtable implementation
blackscholes	2.56%	-61, +4	manual common subexpression elimination
dedup	8.95%	-3, +3	fixed degenerate hash function
ferret	21.27%	-4, +4	rebalanced pipeline thread allocation

Summary of Optimizations

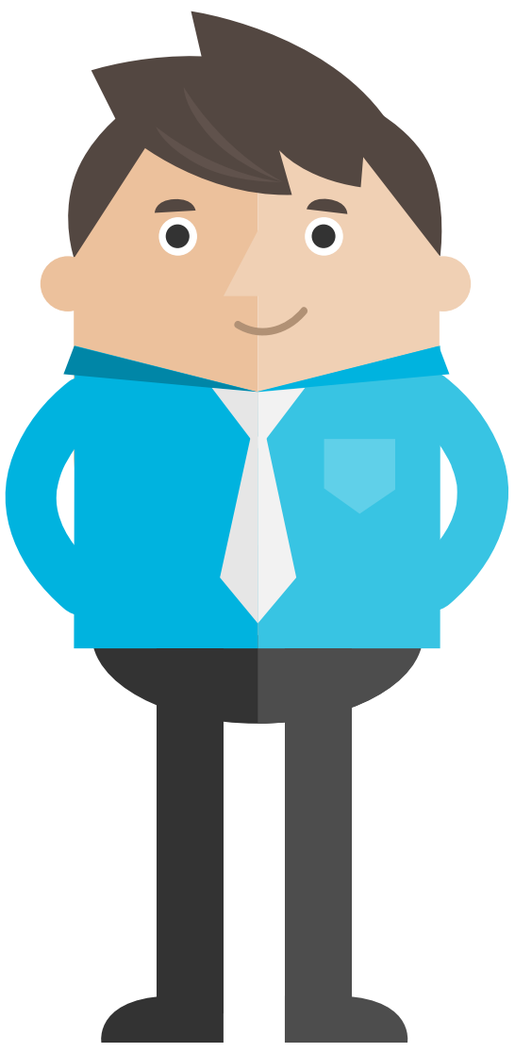
Benchmark	Speedup	Diff Size	Change Summary
memcached	9.39%	-6, +2	removed unnecessary locks
sqlite	25.60%	-3, +3	removed DIY vtable implementation
blackscholes	2.56%	-61, +4	manual common subexpression elimination
dedup	8.95%	-3, +3	fixed degenerate hash function
ferret	21.27%	-4, +4	rebalanced pipeline thread allocation
fluidanimate	37.50%	-1, +0	removed custom barrier with high contention

Summary of Optimizations

Benchmark	Speedup	Diff Size	Change Summary
memcached	9.39%	-6, +2	removed unnecessary locks
sqlite	25.60%	-3, +3	removed DIY vtable implementation
blackscholes	2.56%	-61, +4	manual common subexpression elimination
dedup	8.95%	-3, +3	fixed degenerate hash function
ferret	21.27%	-4, +4	rebalanced pipeline thread allocation
fluidanimate	37.50%	-1, +0	removed custom barrier with high contention
streamcluster	68.40%	-1, +0	removed custom barrier with high contention

Summary of Optimizations

Benchmark	Speedup	Diff Size	Change Summary
memcached	9.39%	-6, +2	removed unnecessary locks
sqlite	25.60%	-3, +3	removed DIY vtable implementation
blackscholes	2.56%	-61, +4	manual common subexpression elimination
dedup	8.95%	-3, +3	fixed degenerate hash function
ferret	21.27%	-4, +4	rebalanced pipeline thread allocation
fluidanimate	37.50%	-1, +0	removed custom barrier with high contention
streamcluster	68.40%	-1, +0	removed custom barrier with high contention
swaptions	15.80%	-10, +16	reordered loop nests



stabilizer-tool.org

coz-profiler.org

Backup Slides

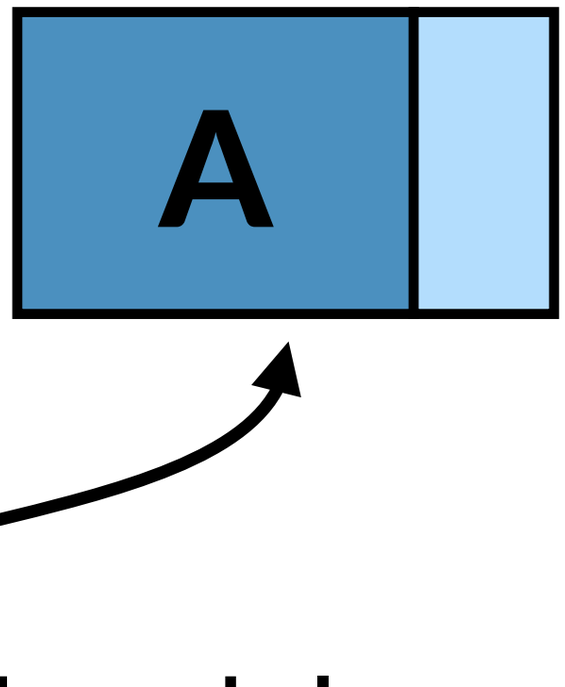
Testing Virtual Speedups

Testing Virtual Speedups

Two versions of A

Testing Virtual Speedups

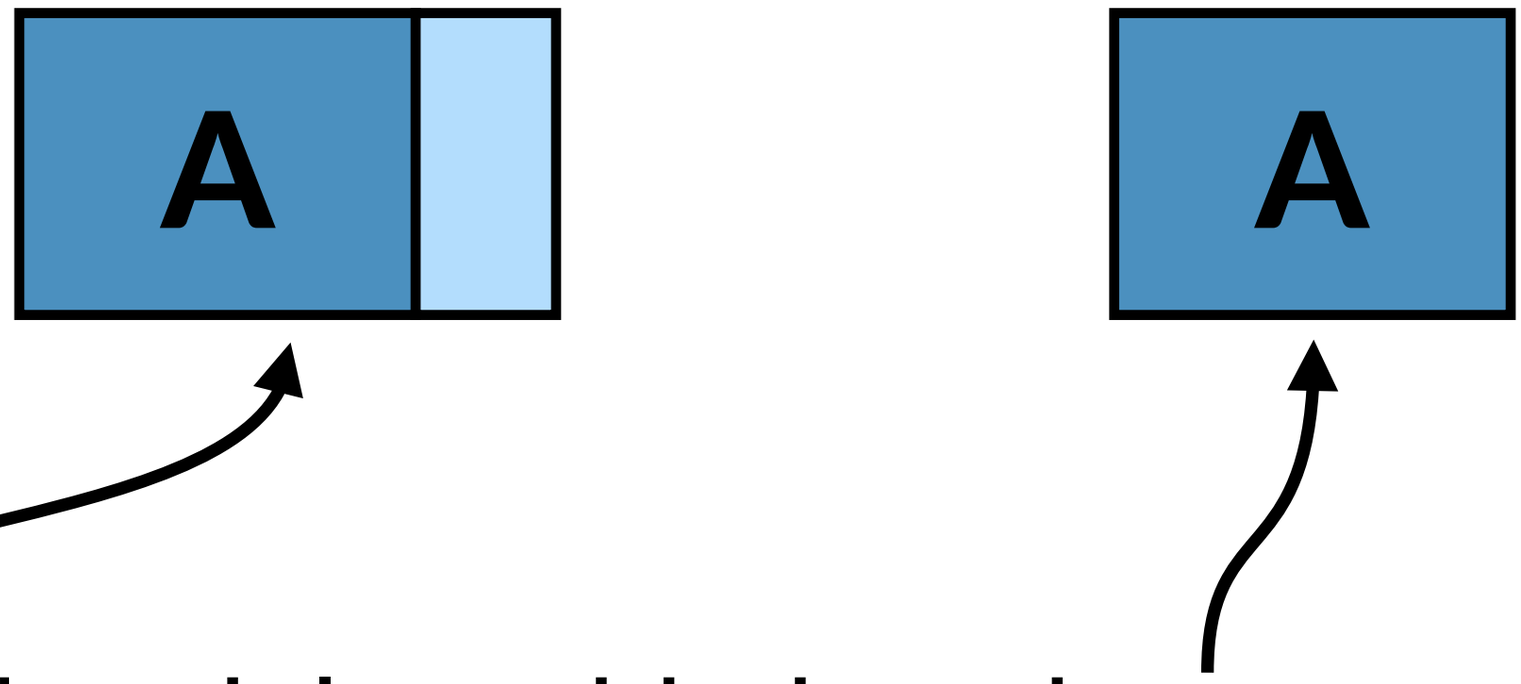
Two versions of A



One with a delay added

Testing Virtual Speedups

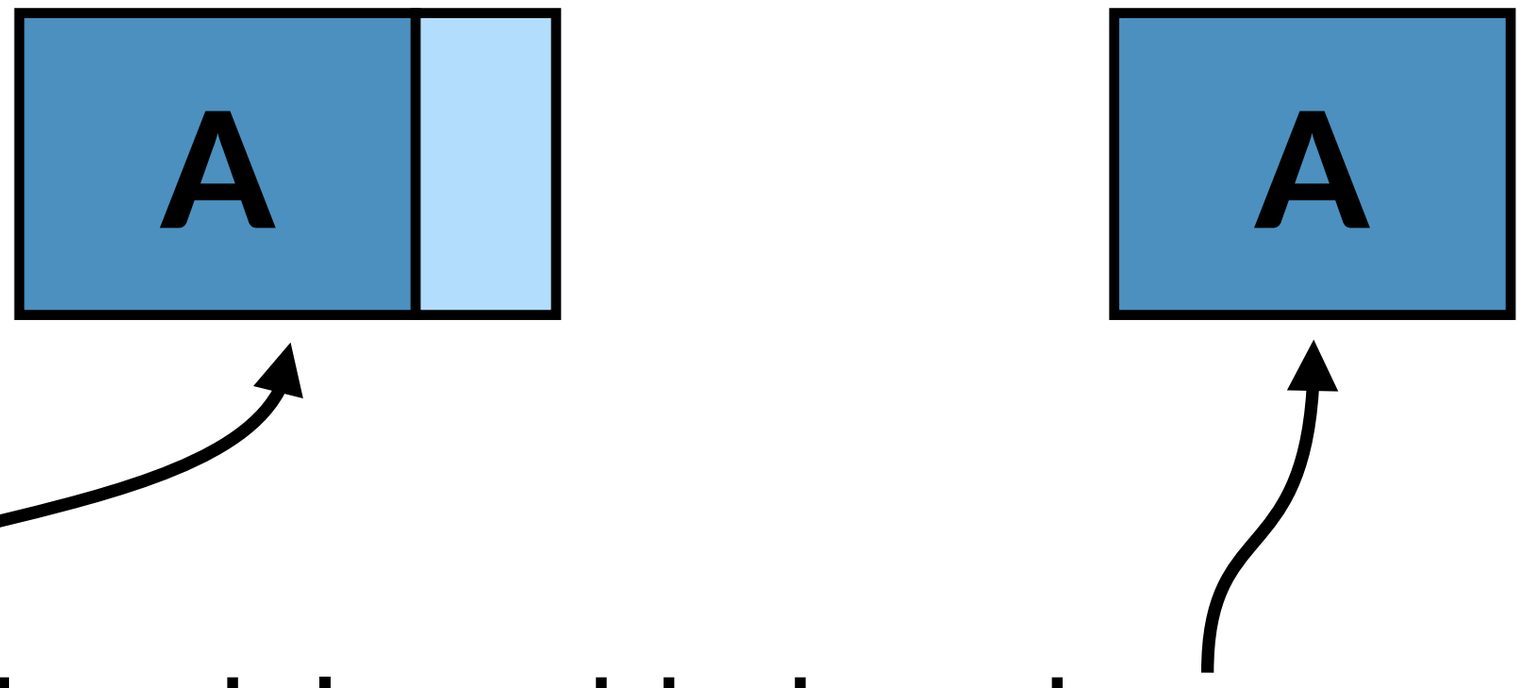
Two versions of A



One with a delay added, and one unmodified.

Testing Virtual Speedups

Two versions of A

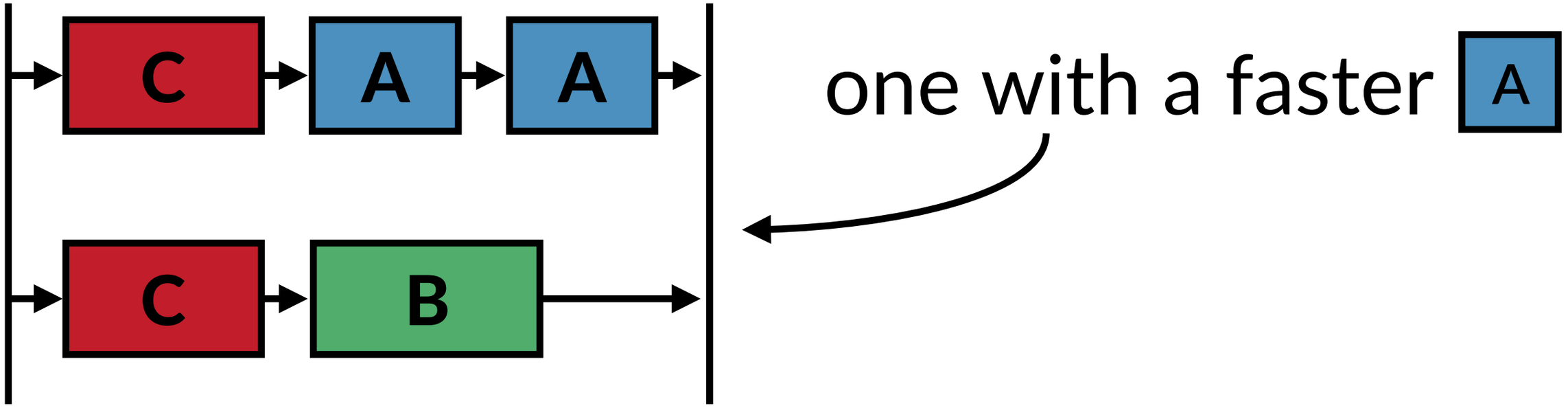


One with a delay added, and one unmodified.

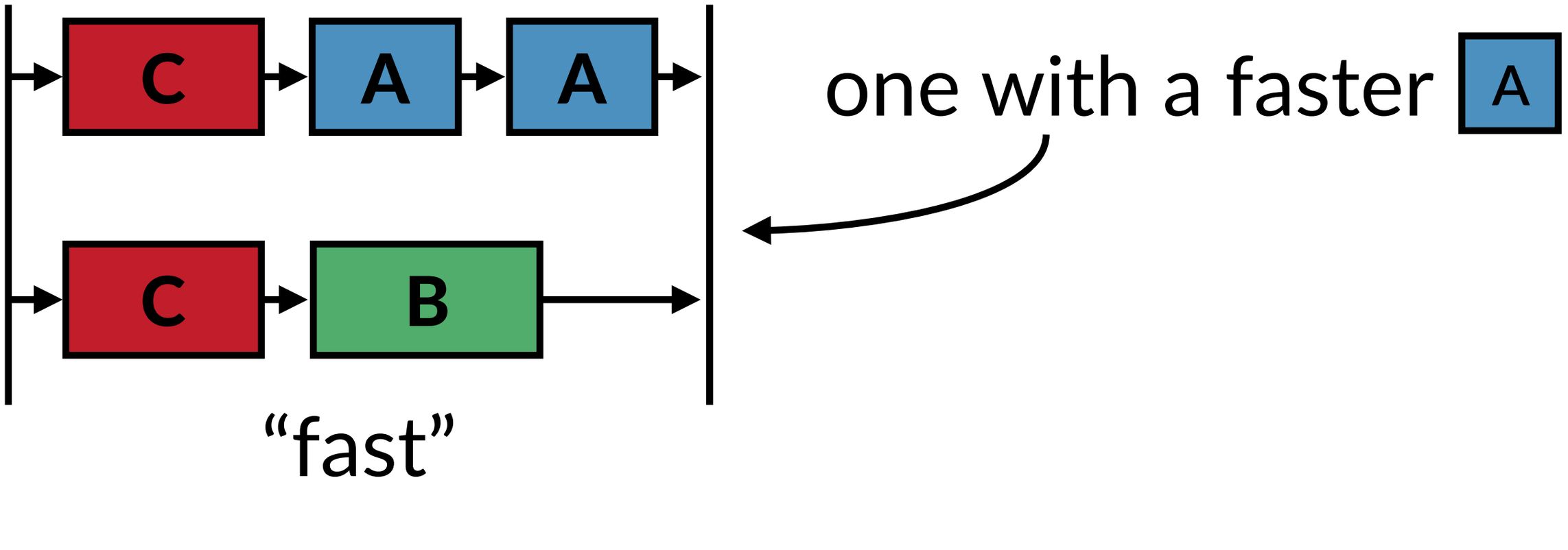
Two versions of our program

Two versions of our program

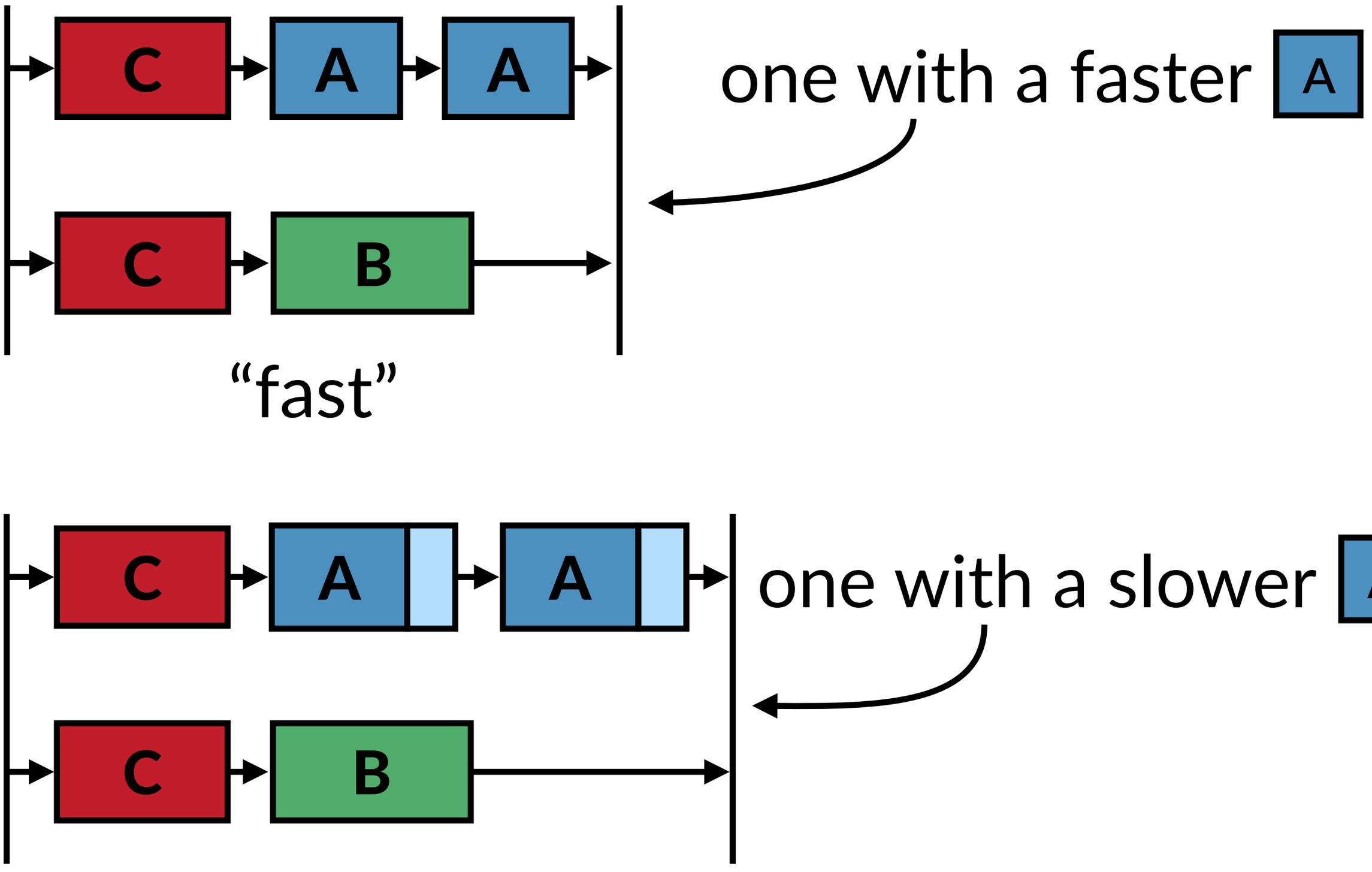
Two versions of our program



Two versions of our program

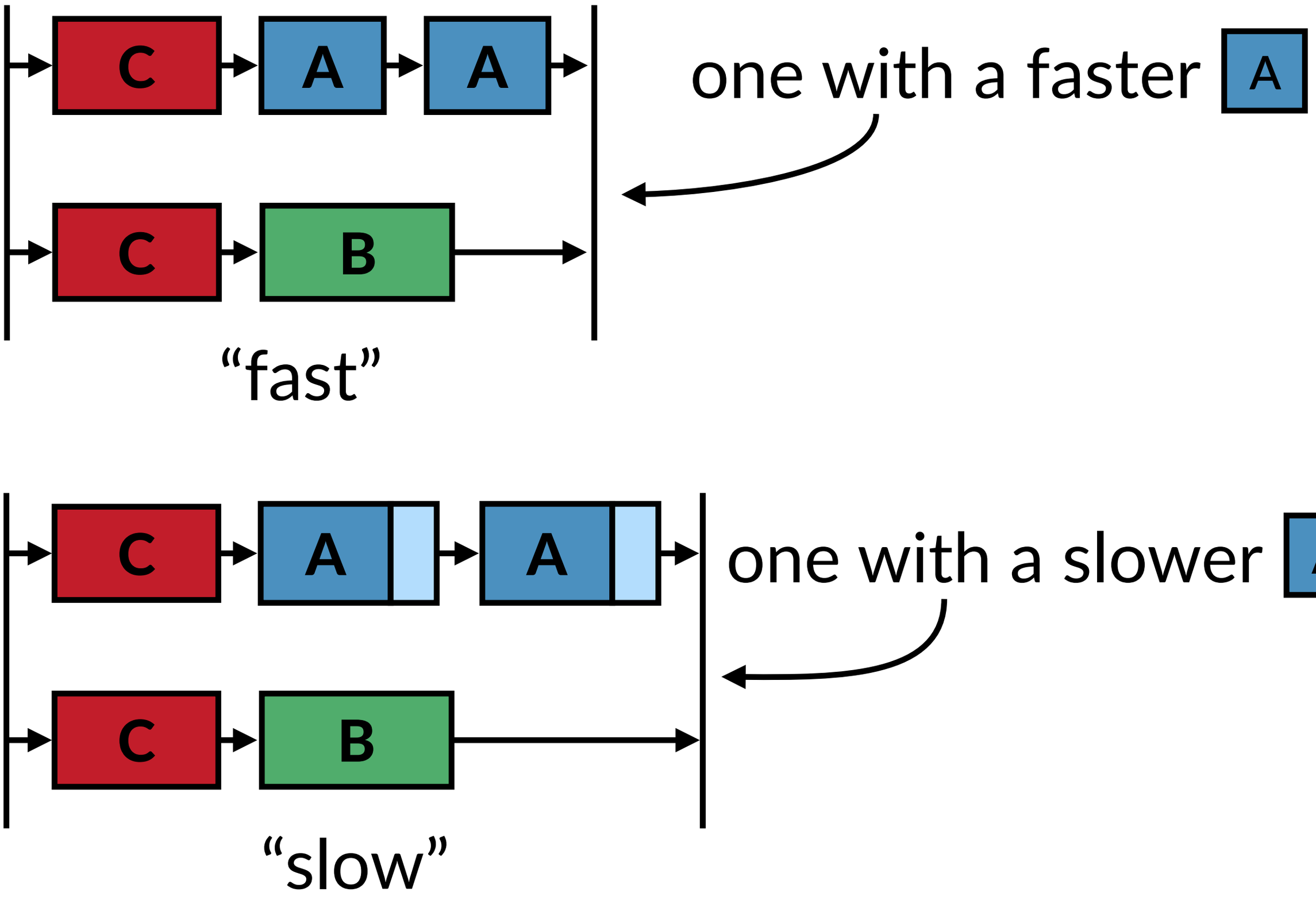


Two versions of our program

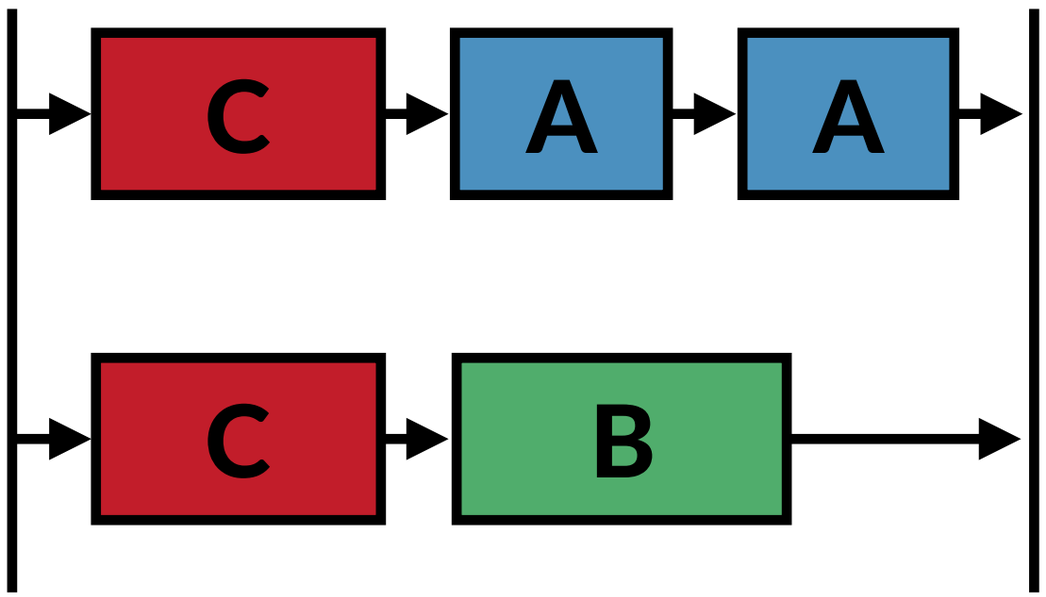


"fast"

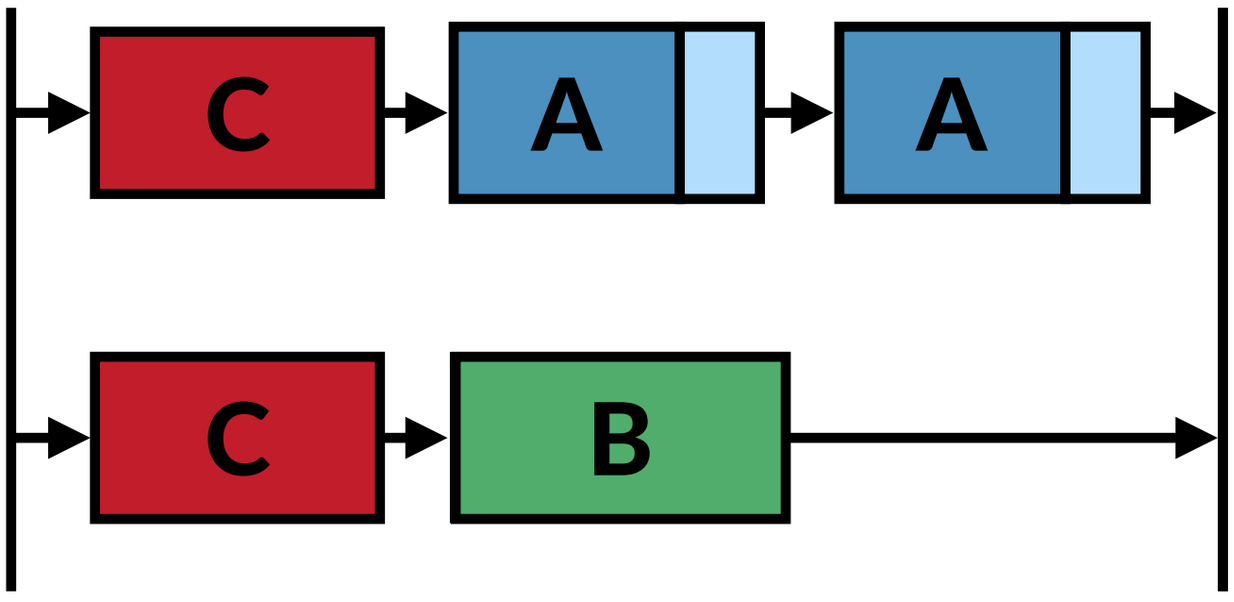
Two versions of our program



Two versions of our program

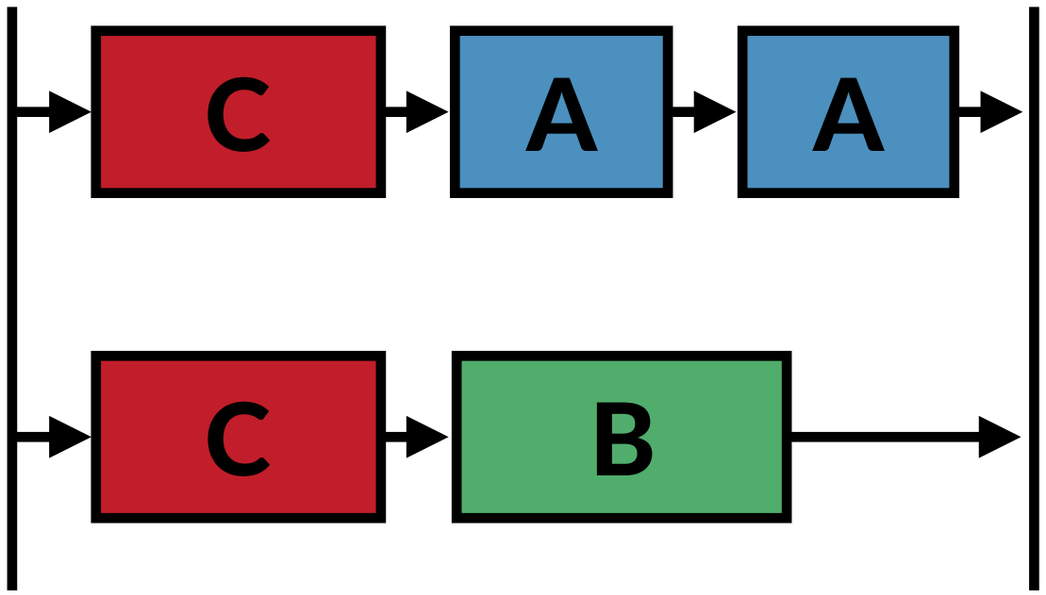


“fast”

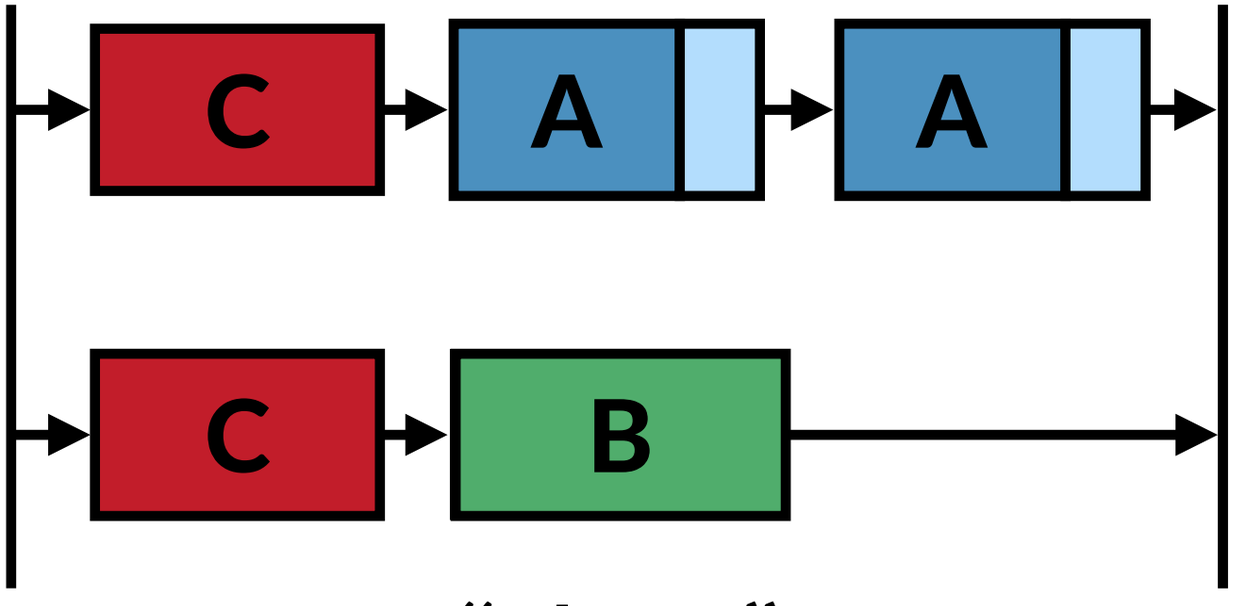


“slow”

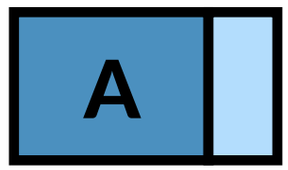
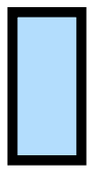
Two versions of our program



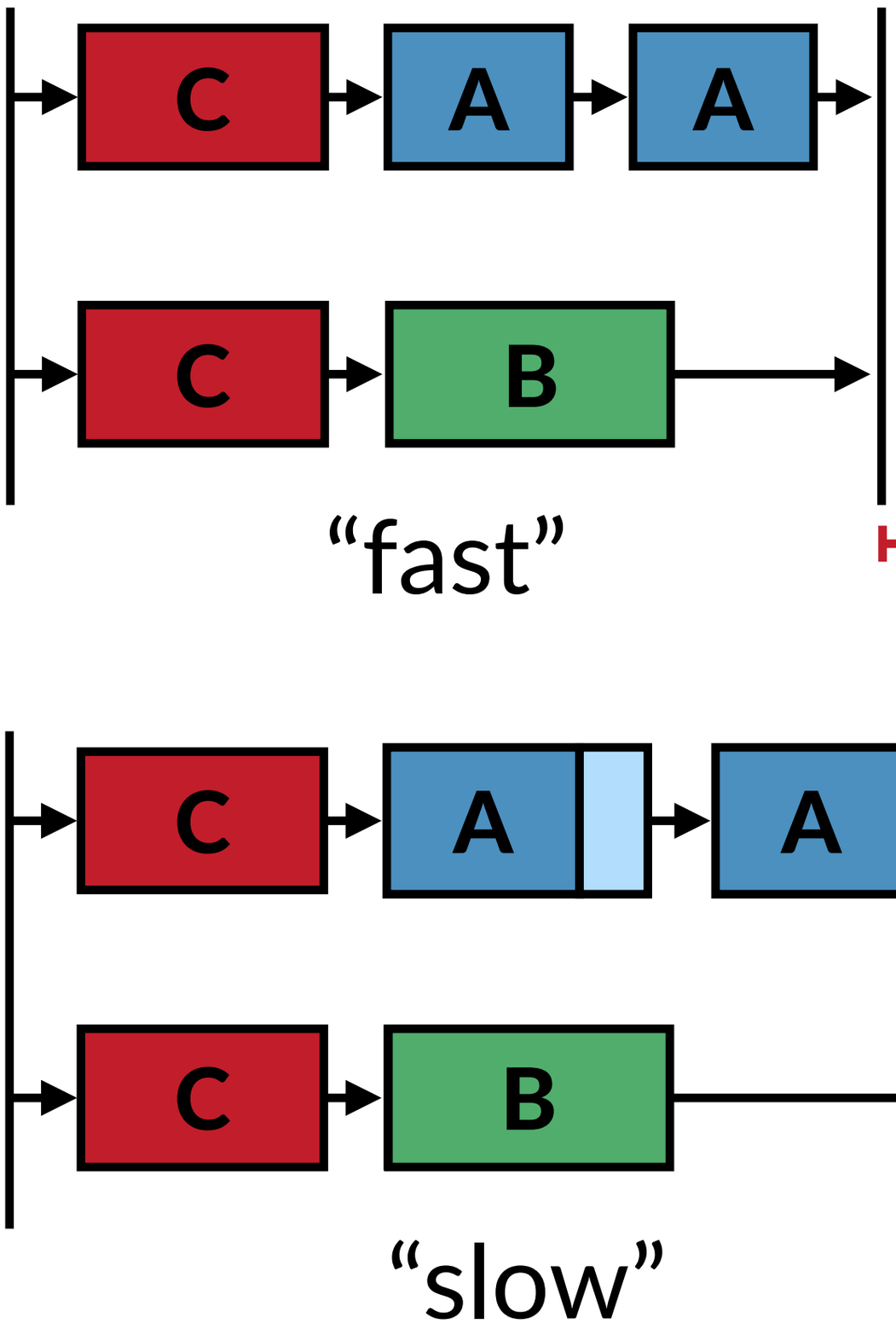
"fast"

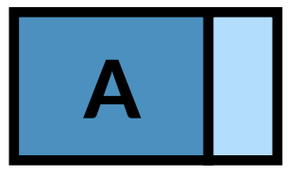
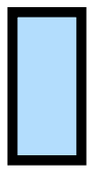


"slow"

Speeding up  by 

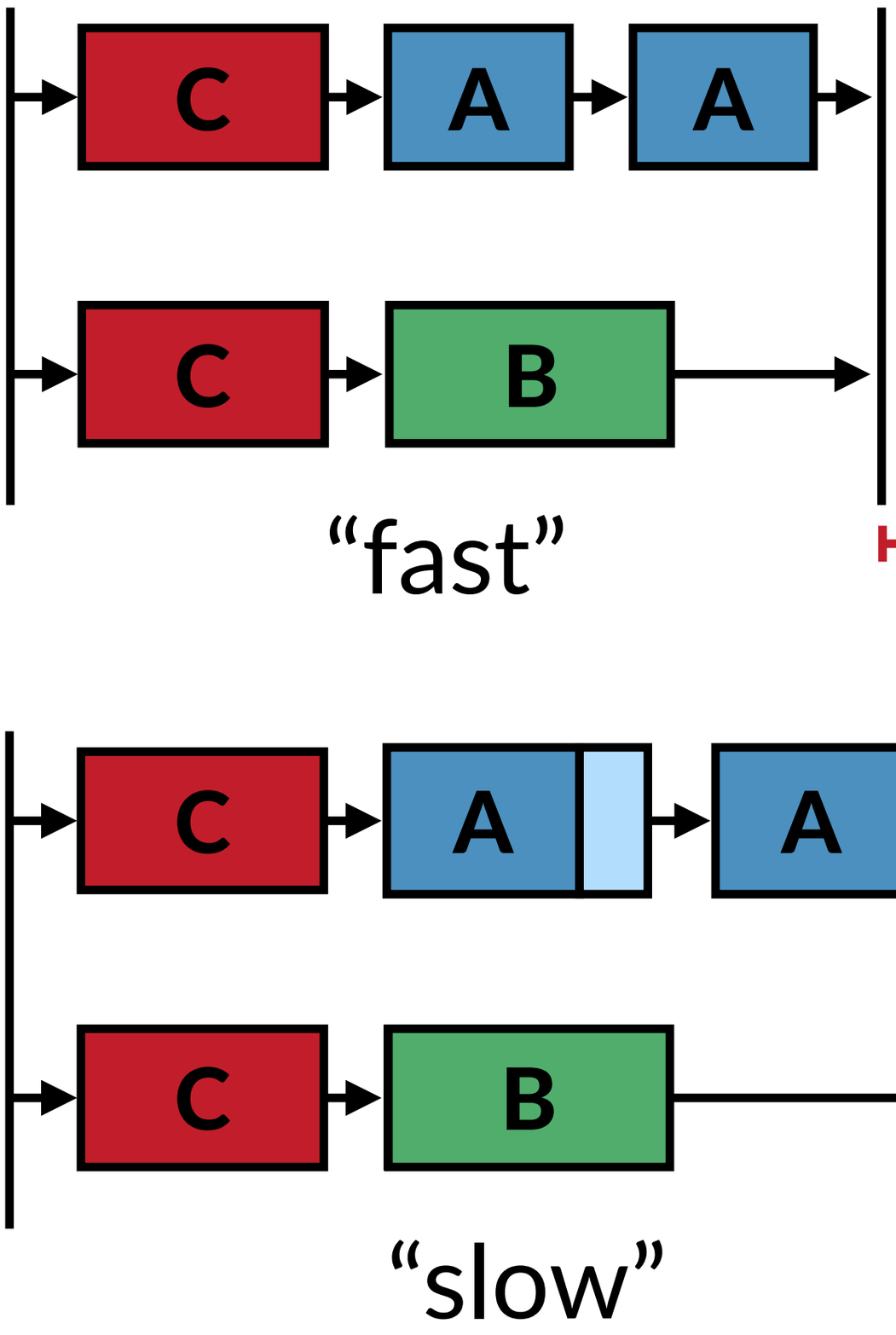
Two versions of our program



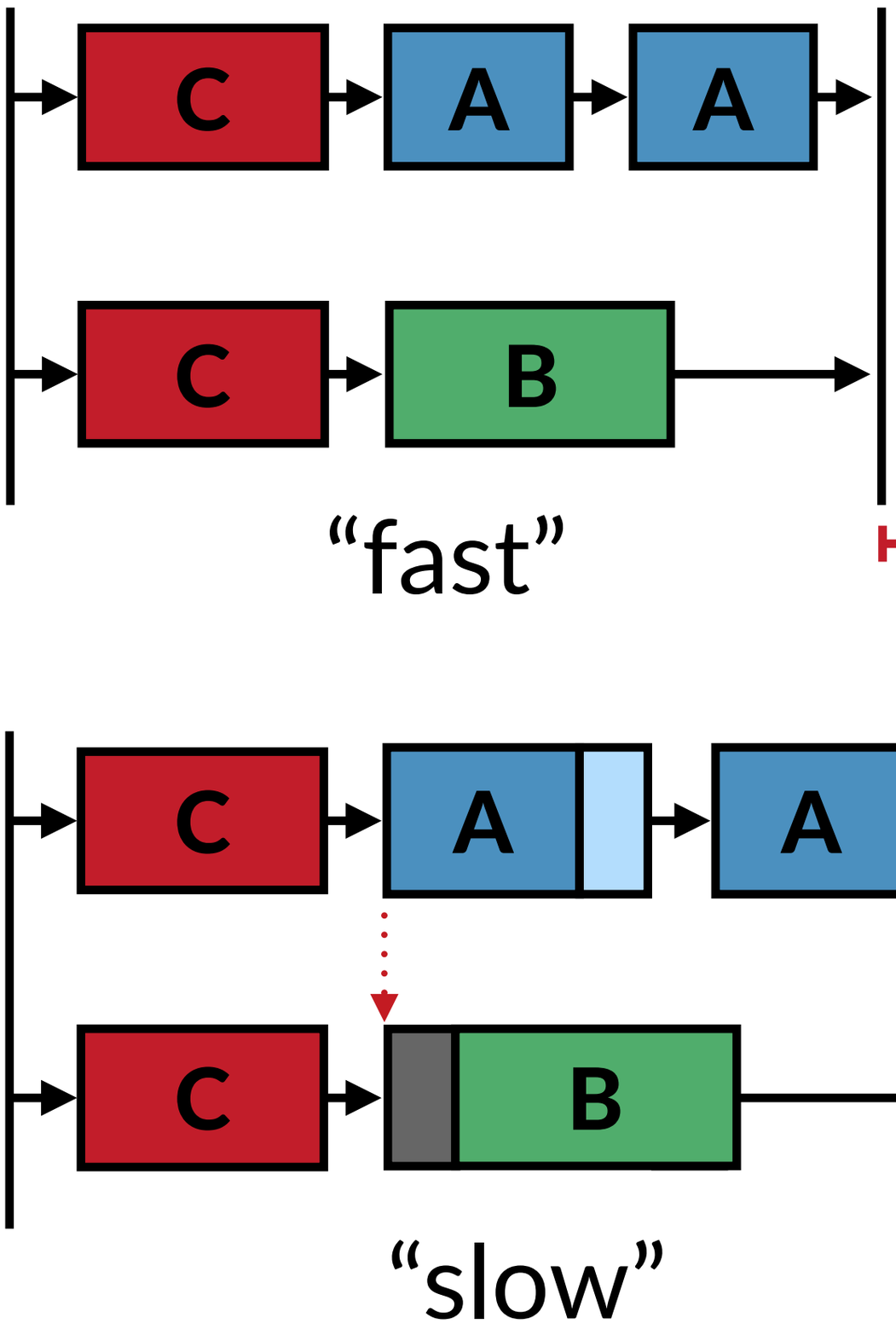
Speeding up  by 

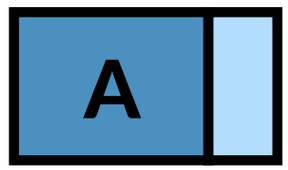
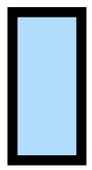
shortens execution by this much

Two versions of our program



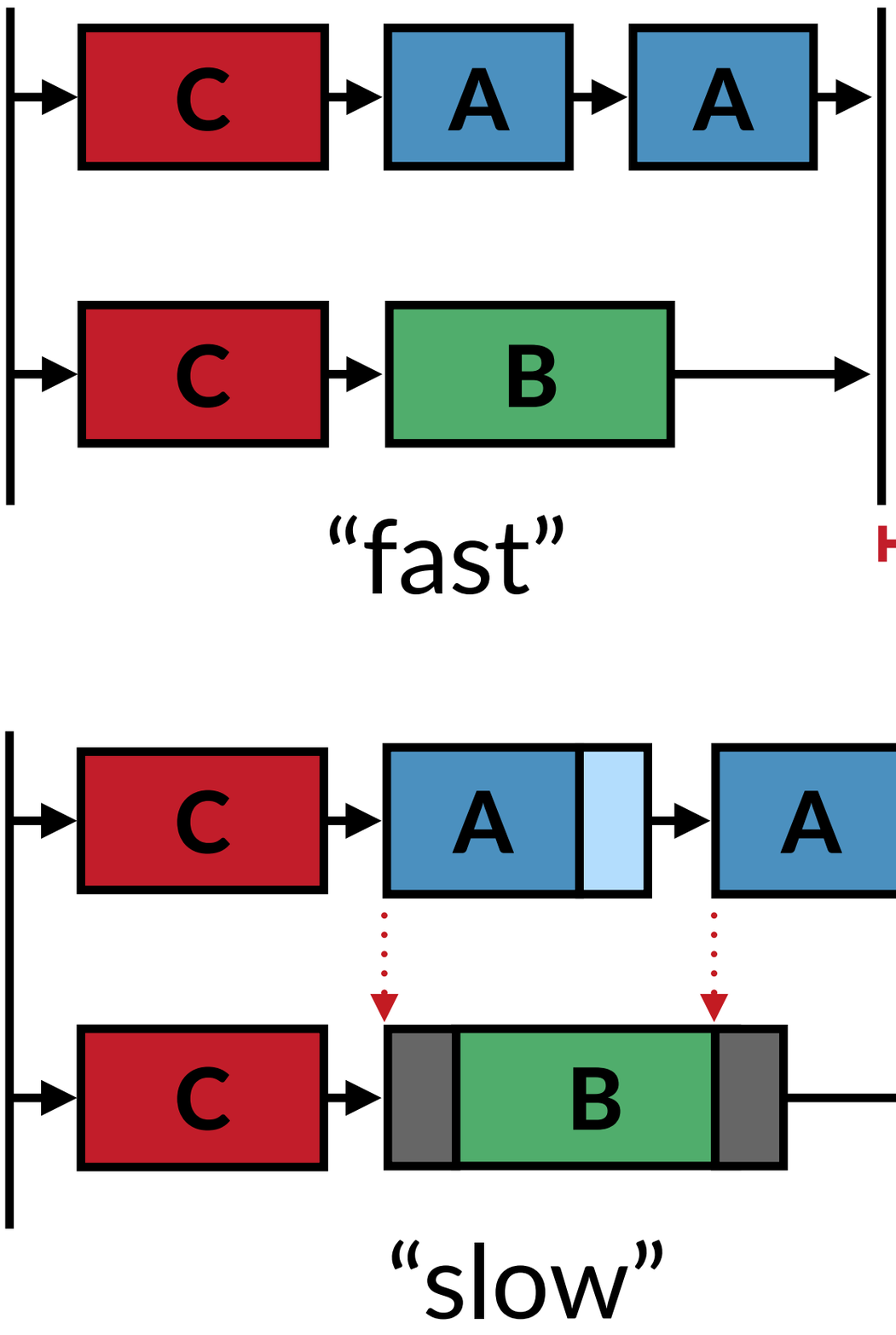
Two versions of our program

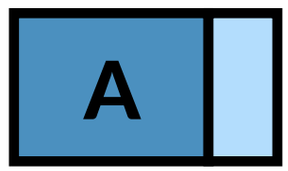
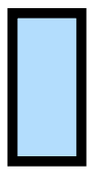


Speeding up  by 

shortens execution by this much

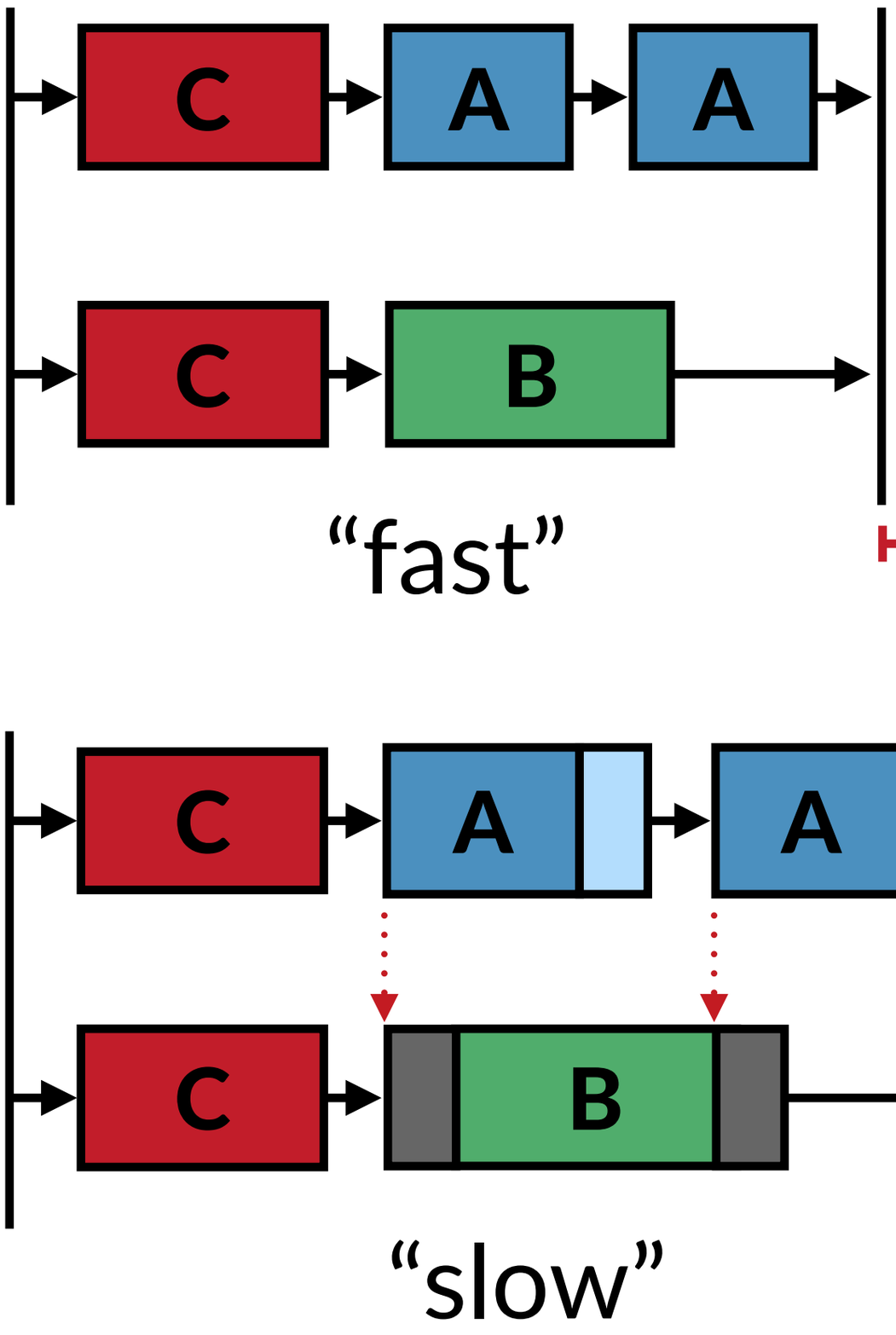
Two versions of our program

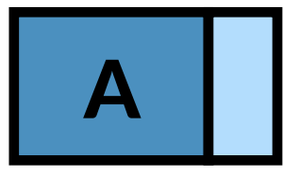
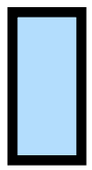


Speeding up  by 

shortens execution by this much

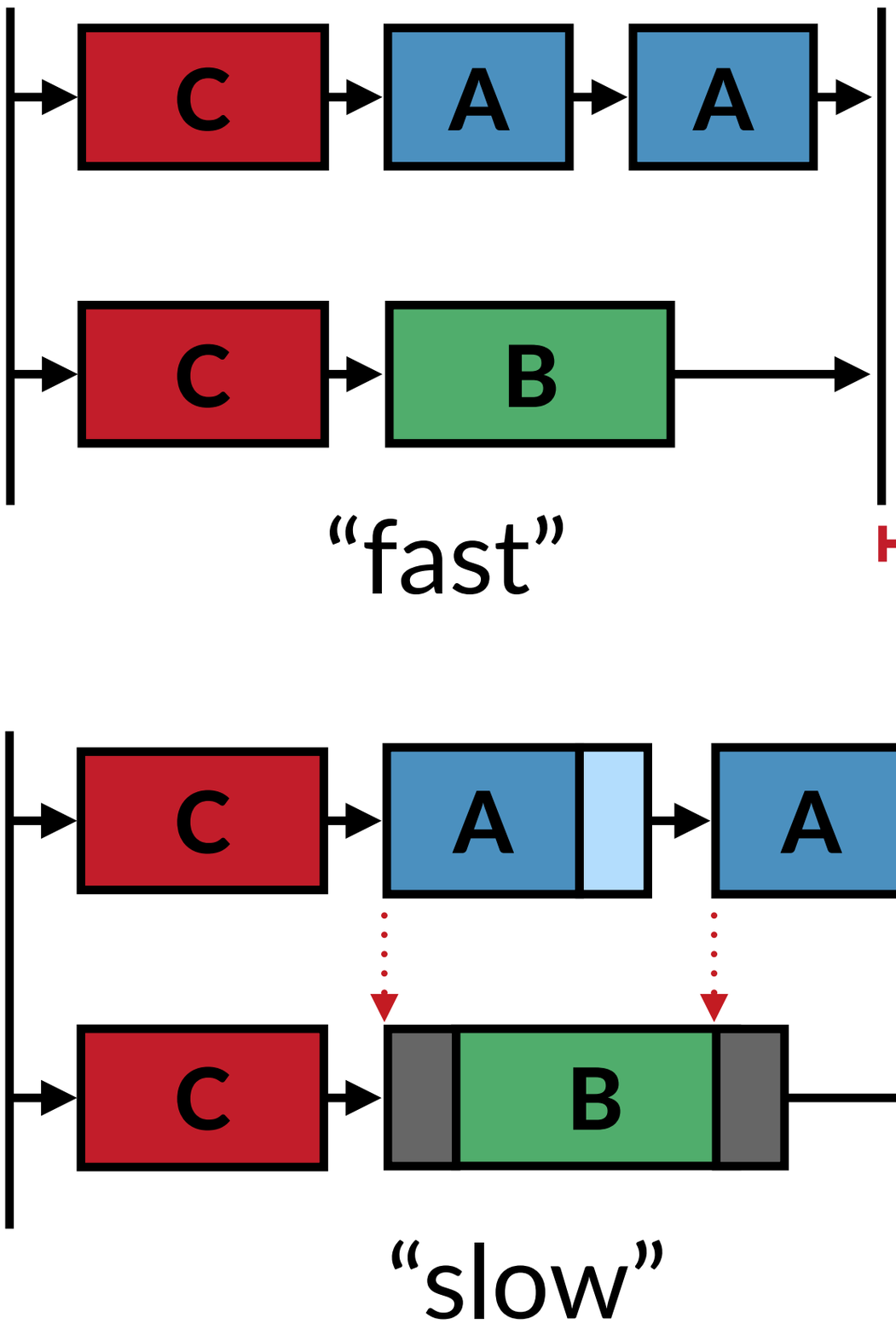
Two versions of our program

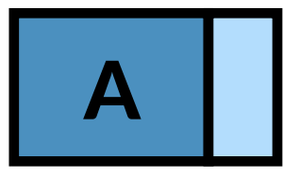
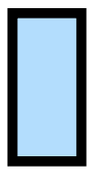


Speeding up  by 

shortens execution by this much

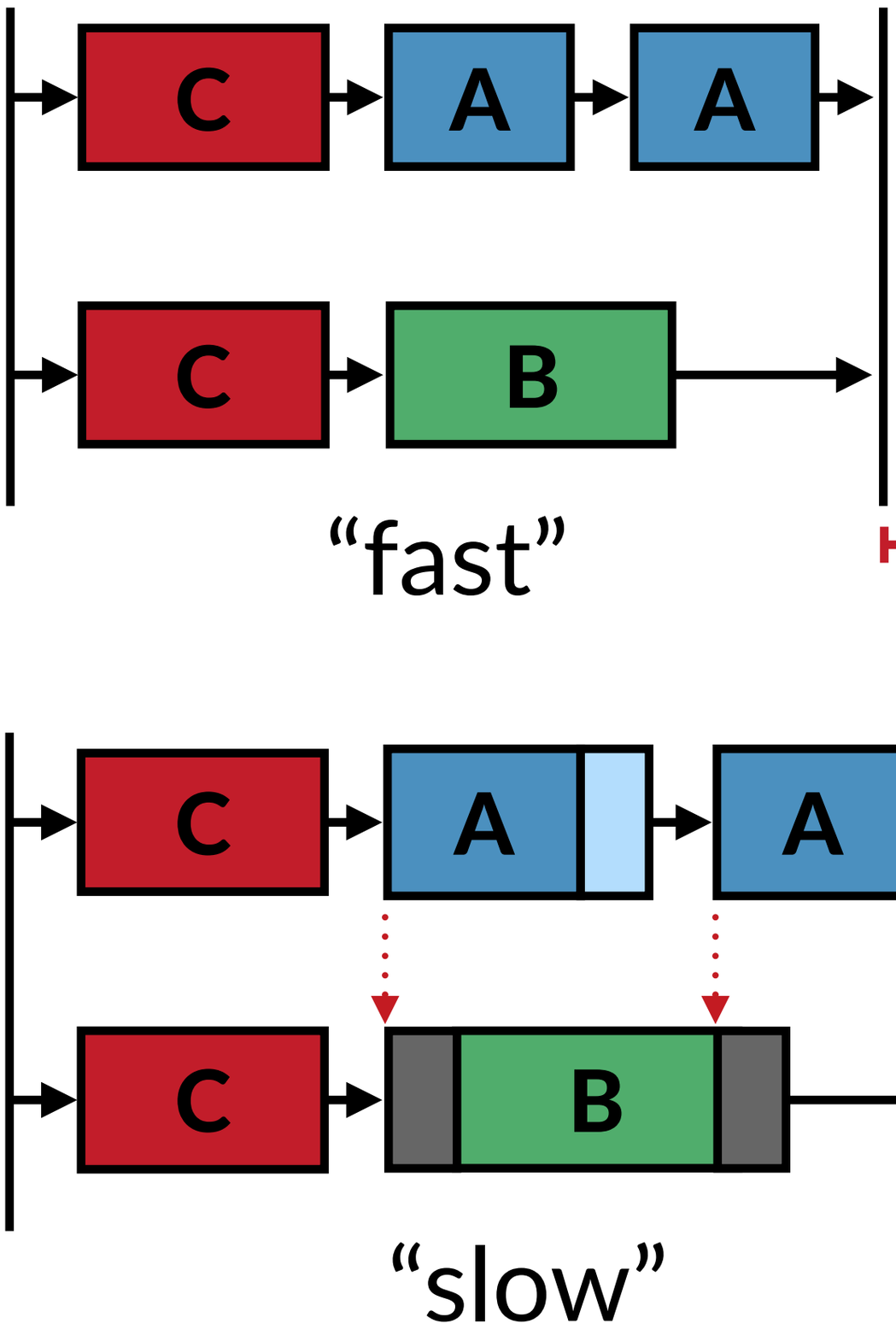
Two versions of our program



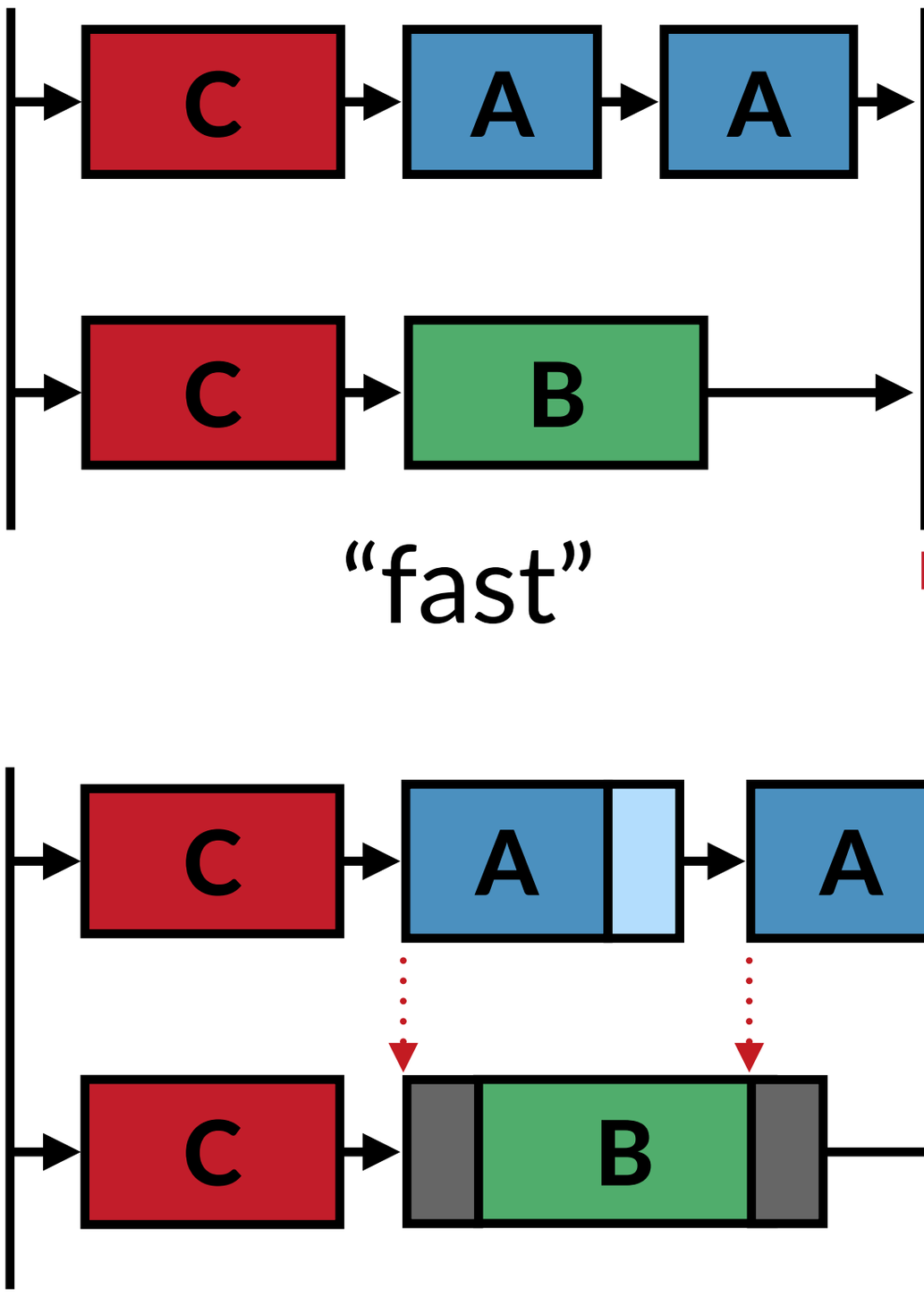
Speeding up  by 

shortens execution by this much

Two versions of our program



Two versions of our program



Speeding up A by

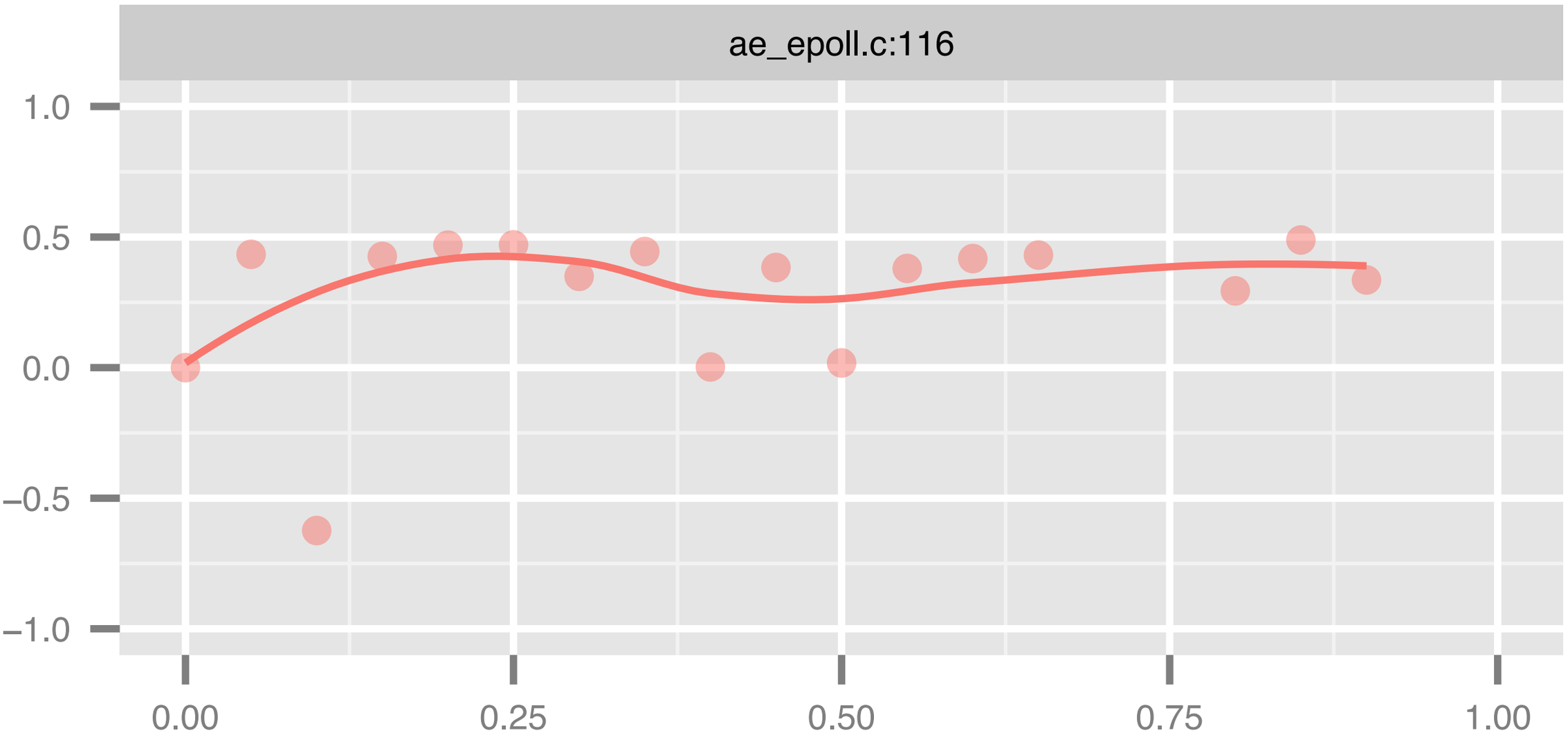
shortens execution by this much

virtually speeding up A

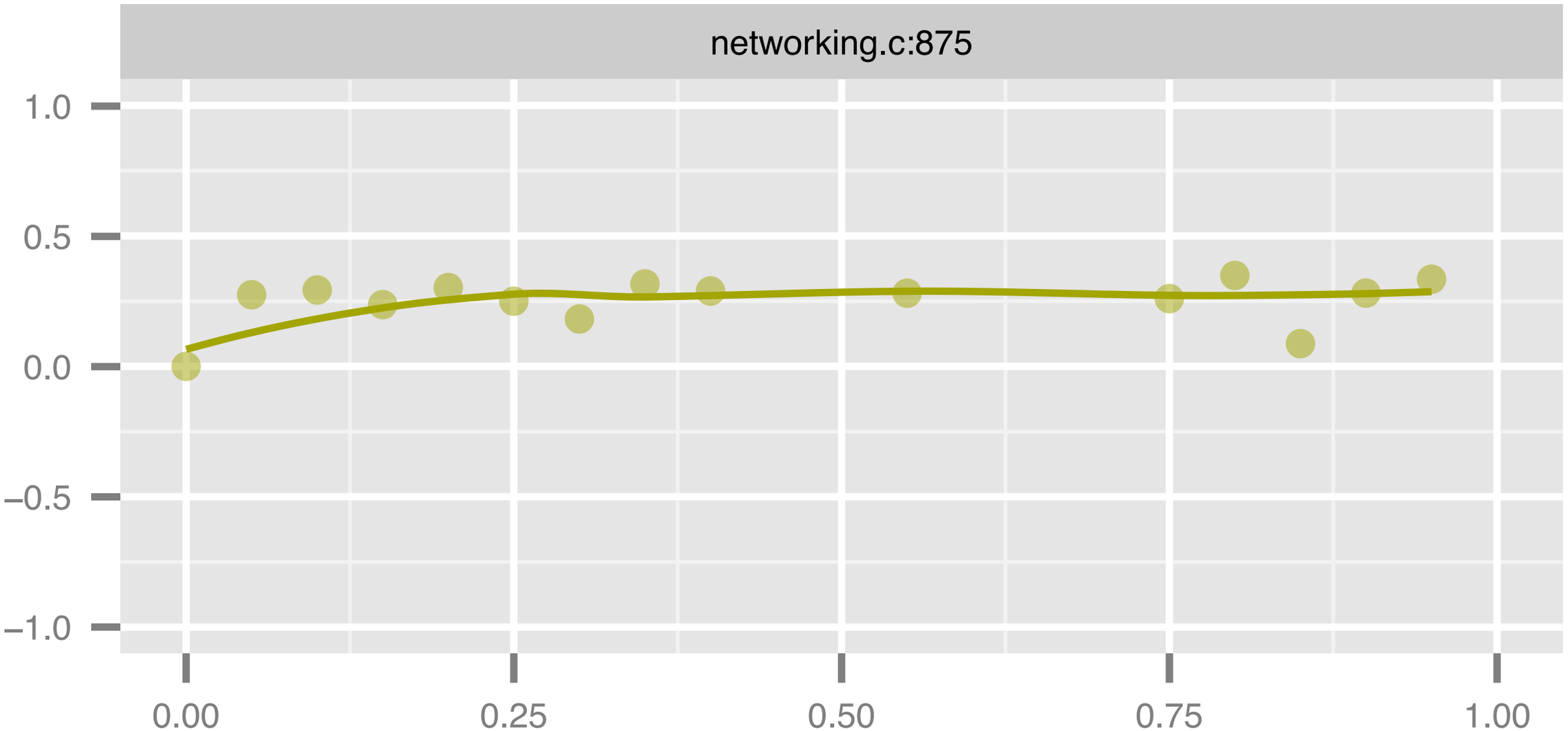
should have the same effect

Redis Results

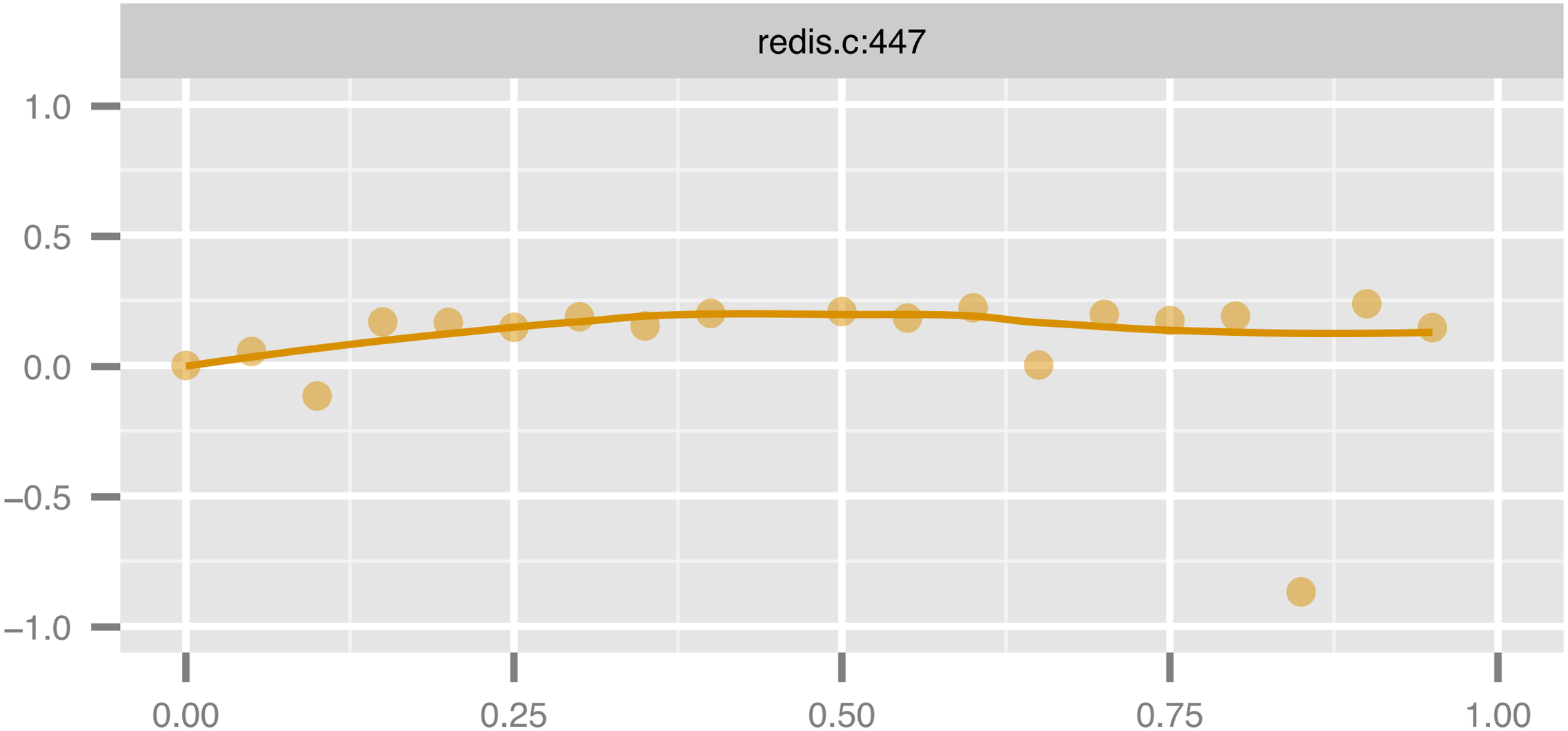
Waiting for accept response



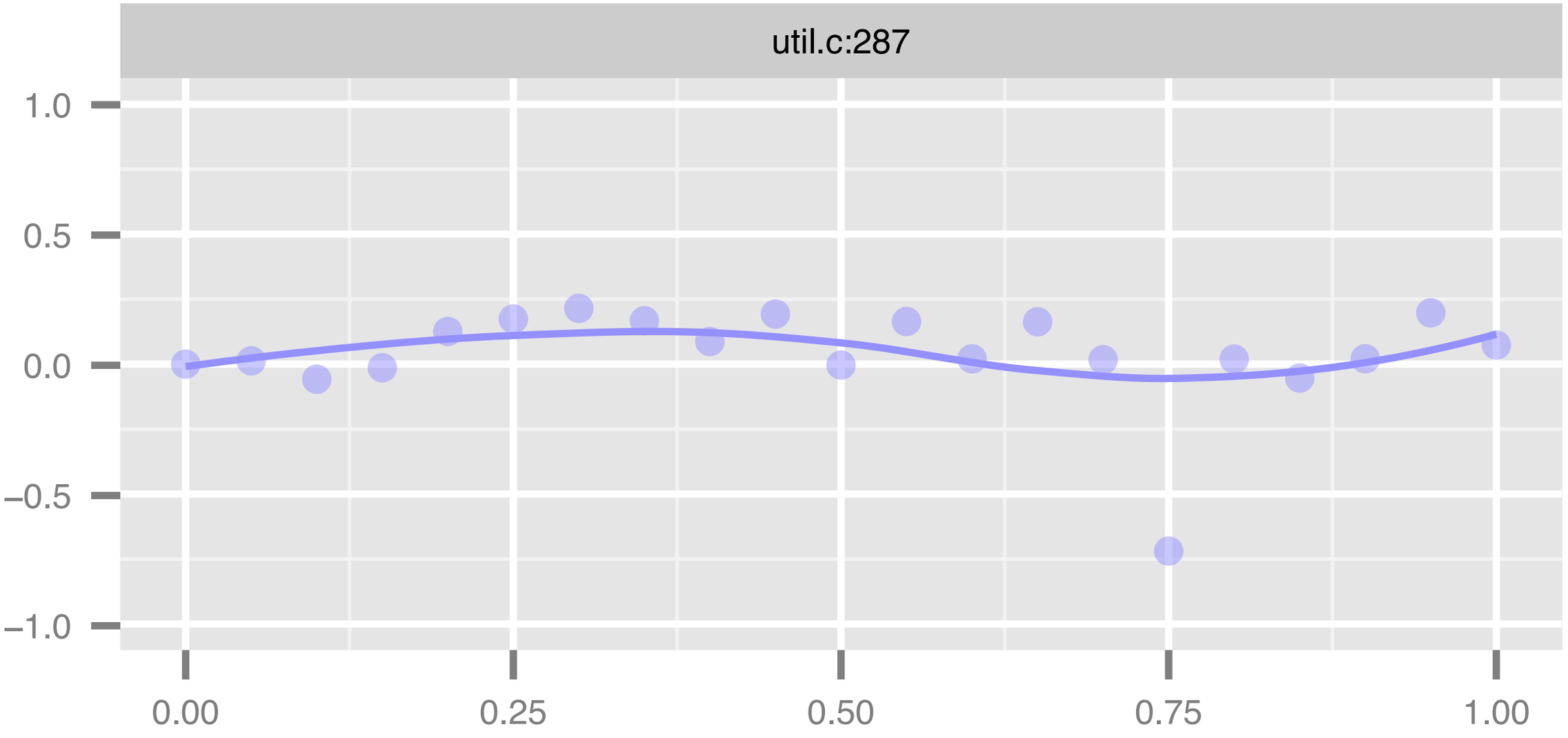
Cleaning up after a query



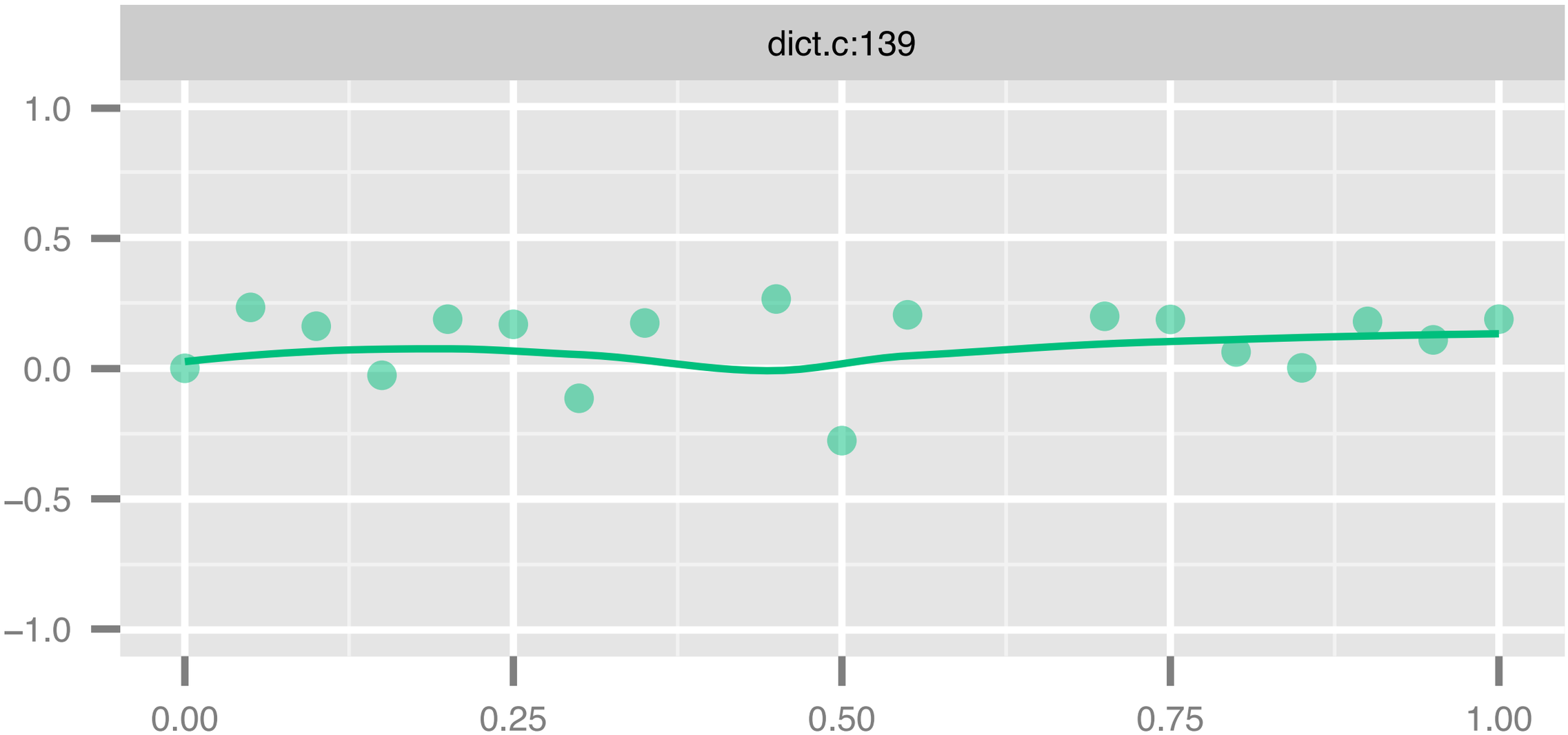
Case Insensitive String Comparison



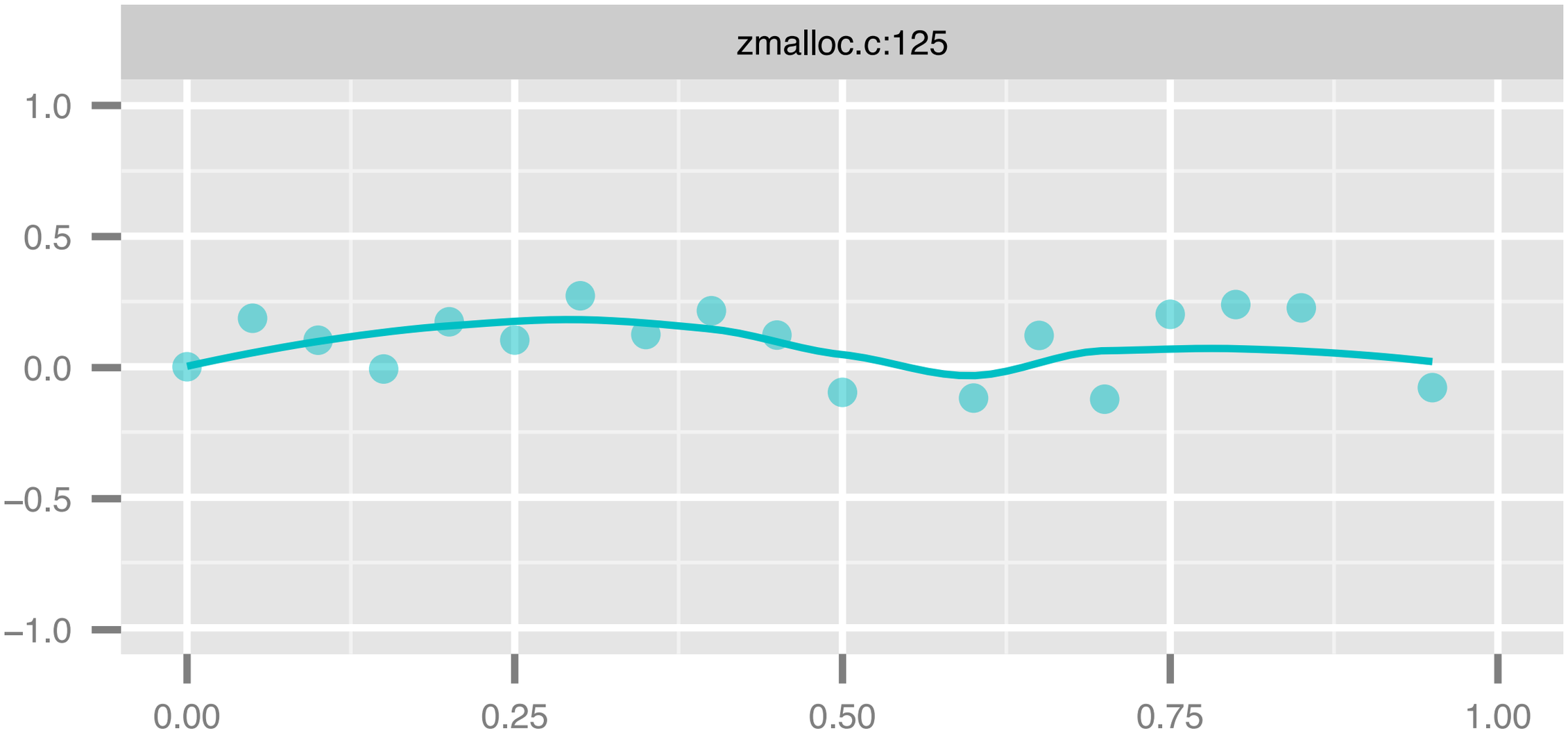
Custom Number to String Conversion



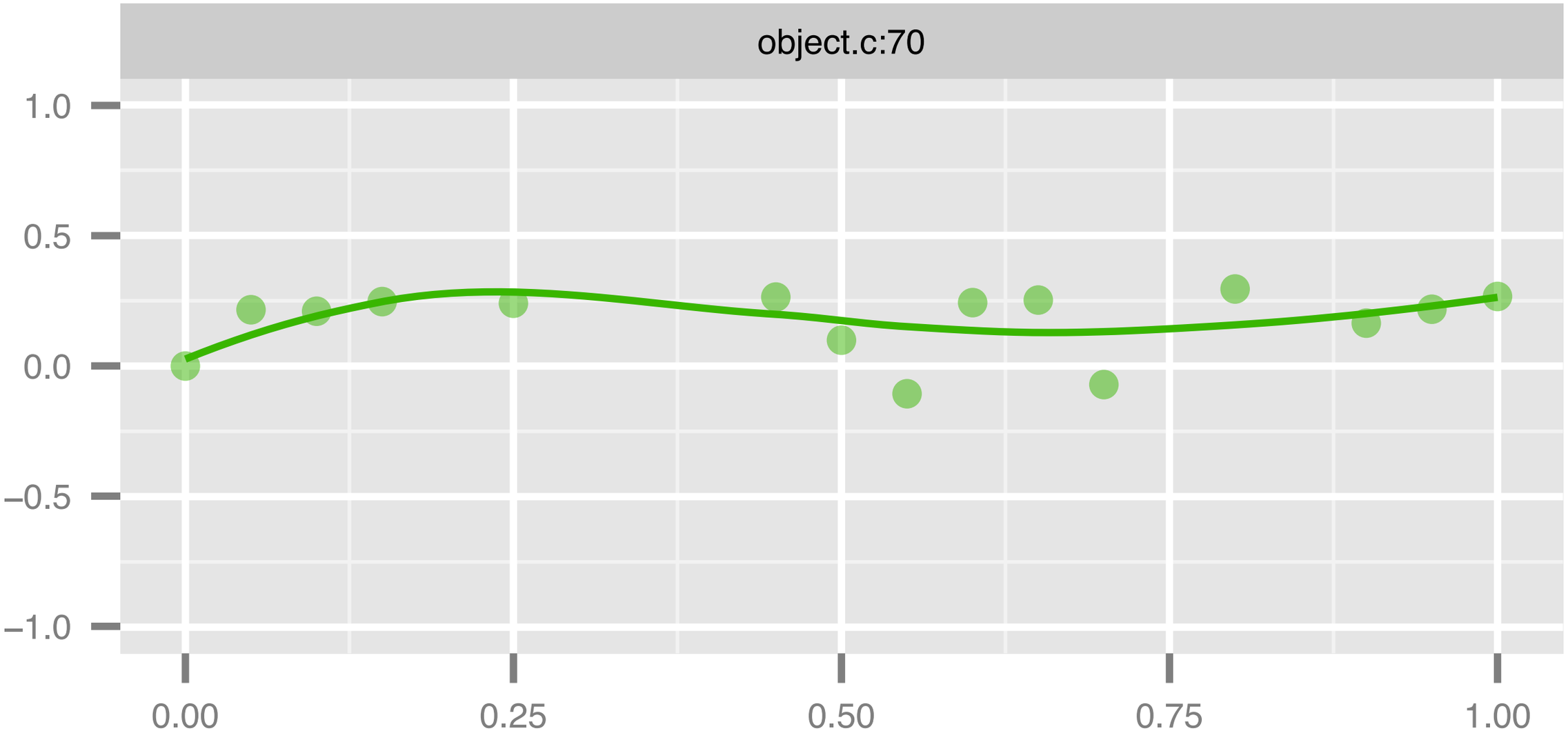
Frequently Executed Hash Function



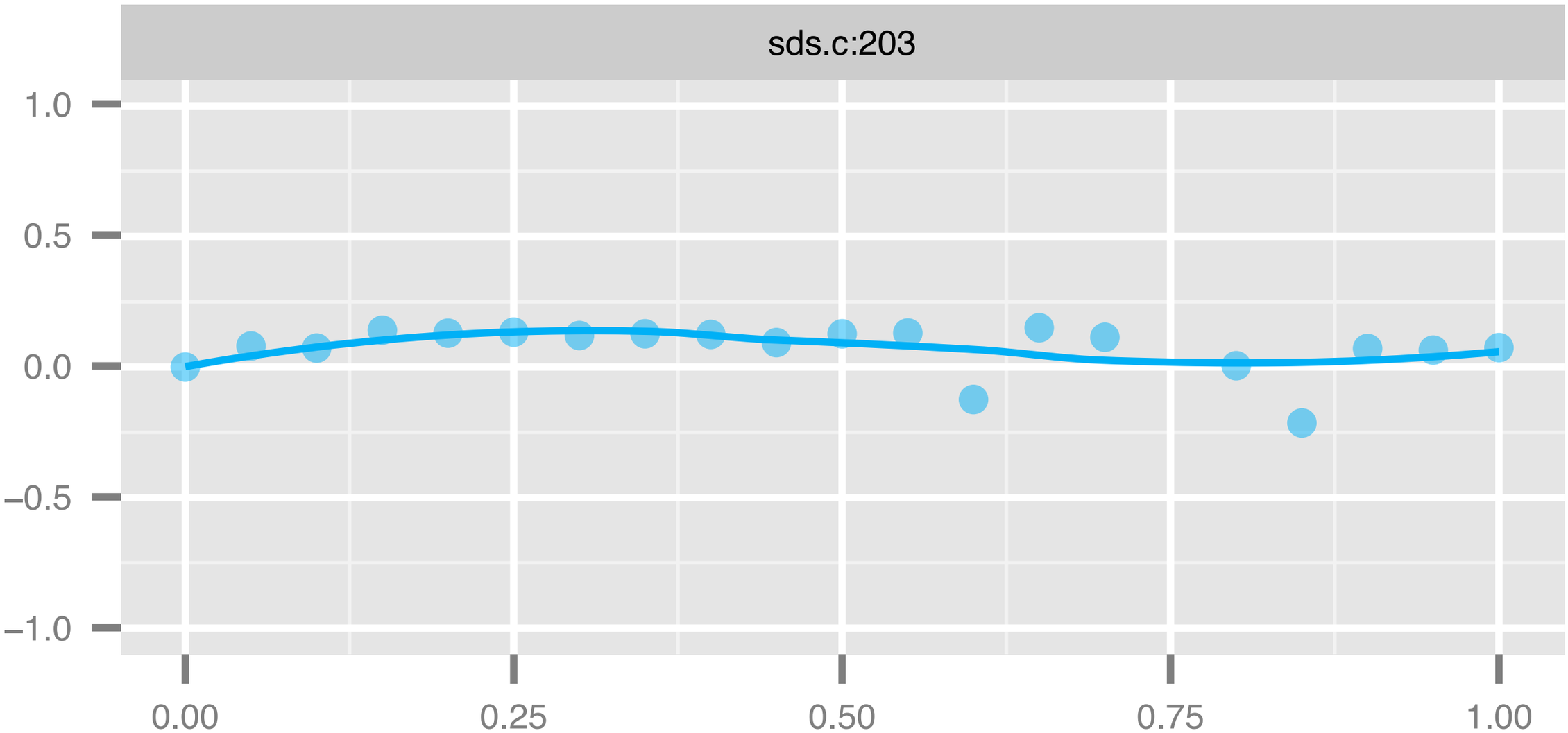
Allocating Memory for Custom String Representation



Setting up Custom String Representation



Changing Length of Custom String



“Pinning” Custom Strings