Perceptron Learning for Cache Management

Elvira Teran

School of Engineering
Texas A&M International University

January 15th, 2019

Last Level Caches

- Slow memory
  - Off-chip main memory might be >200 times as slow as the processor
    - E.g. an array access might take 200 times longer than an addition
- Mitigate high latency to main memory
- Limited capacity
- Speculate which data must be kept in faster on chip memory
- Cache policy efficiency impacts performance

Reuse Prediction

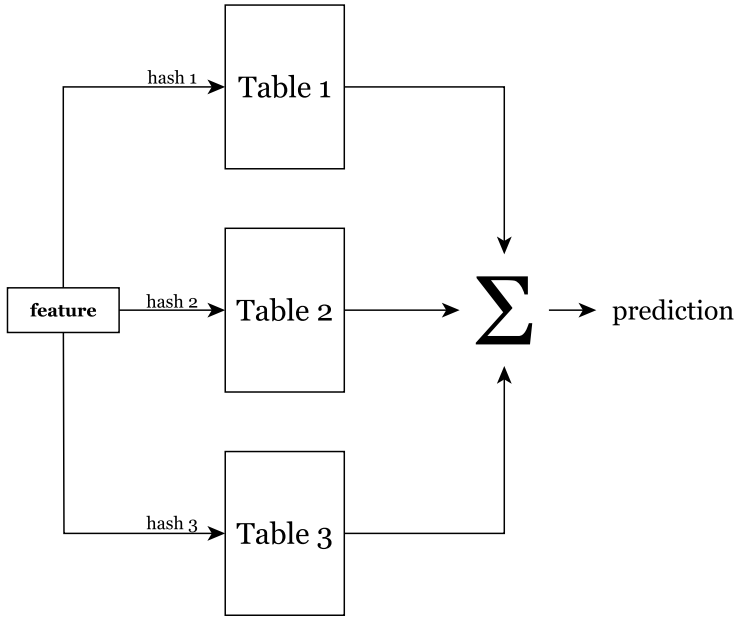
- *Dead block prediction*
- Predict whether a block will be reused again before eviction
- Use features to find correlations between past behavior and future accesses
- Use:
  - Drive placement, replacement and bypass
- Accuracy from these features in LLC has been problematic

Perceptron-Based Techniques

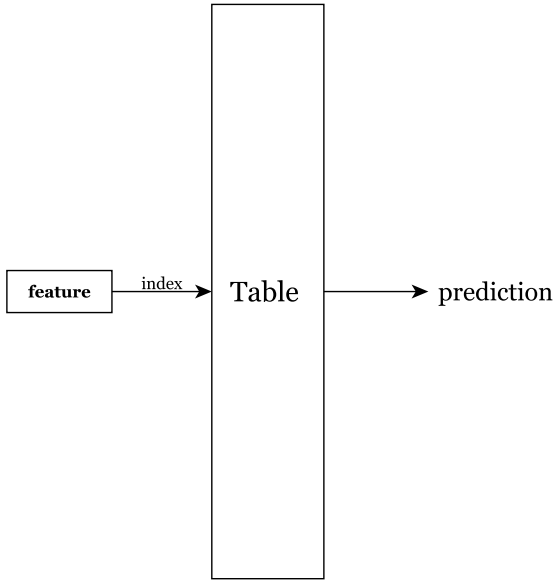
- *Perceptron learning for reuse prediction* *[MICRO 2016]*
- *Multiperspective reuse prediction* *[MICRO 2017]*
- Contributions
  - Perceptron learning in cache management
  - Combine multiple (many) features to improve the accuracy
  - Placement, replacement and bypass optimization
  - Higher accuracy yields better performance

Related Work

- Multiple features correlate to last access:
  - Trace of PCs, last PC, bits from memory address, time/reference count, etc.



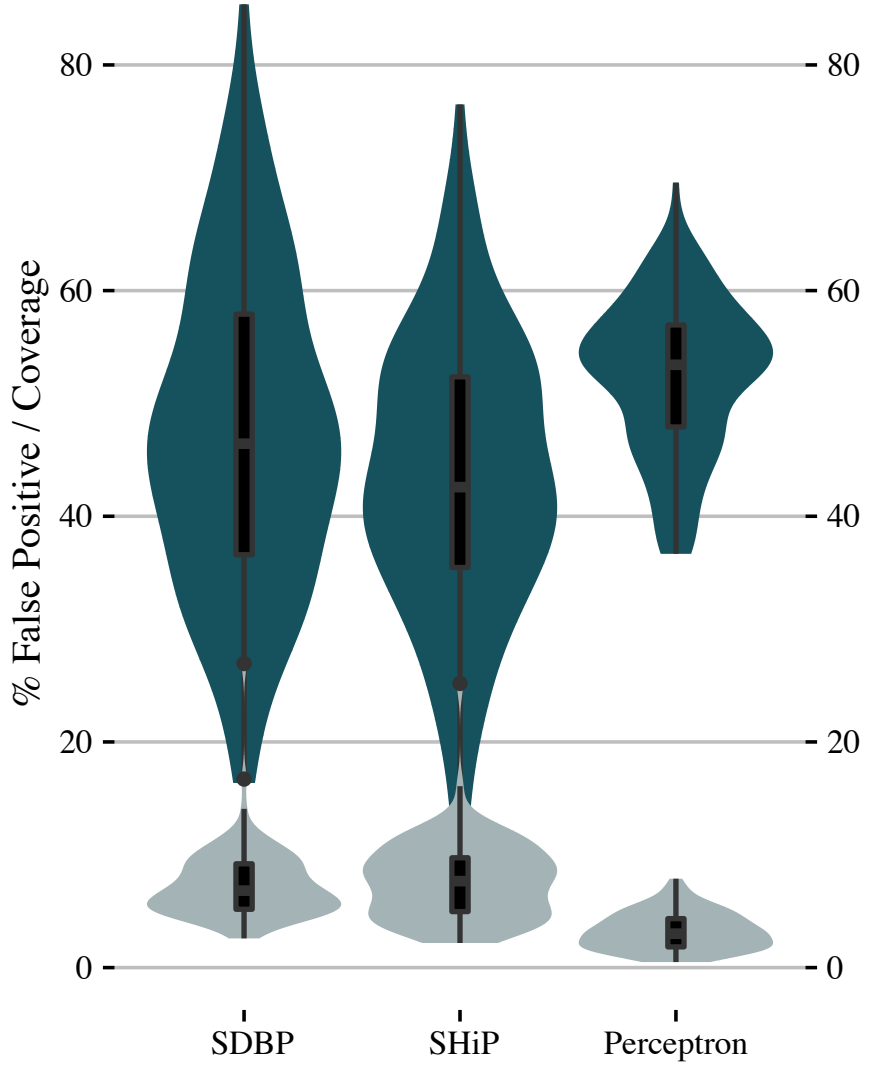
Sampling Dead Block Predictor (SDBP) *[Khan et al.]*



Signature-based Hit Predictor (SHiP)*[Wu et al.]*

Perceptron Learning for Reuse Prediction

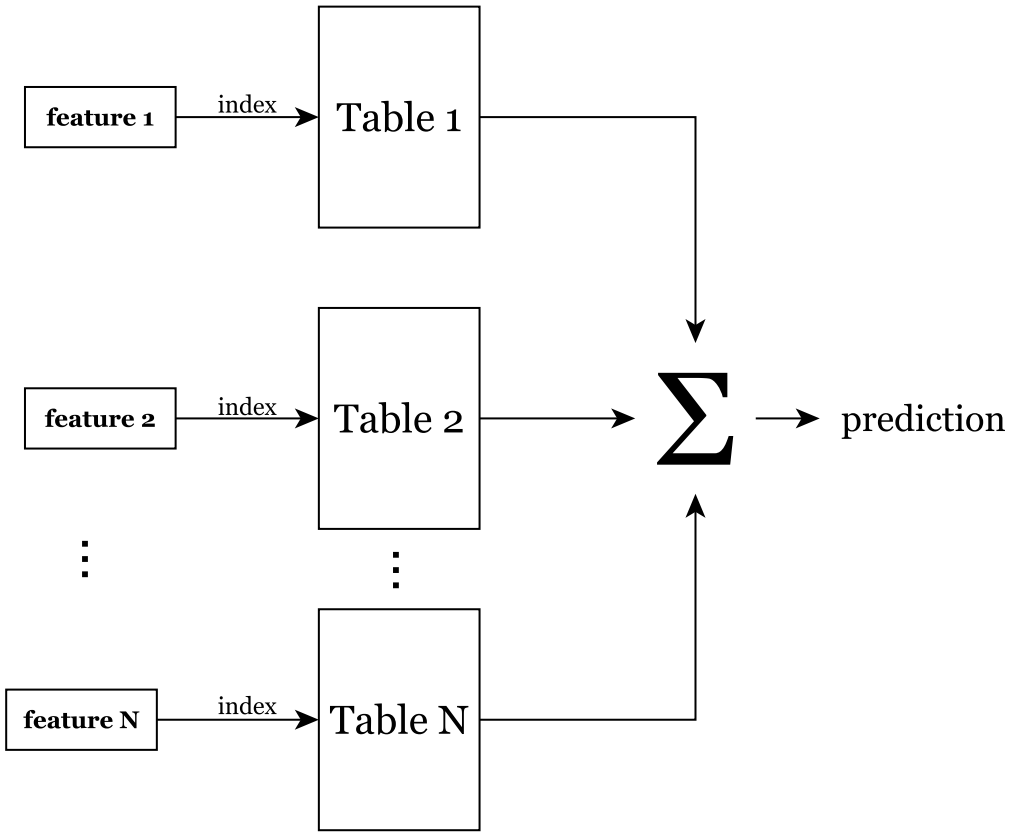
- Coverage rate:
  - SDBP: 47.2%
  - SHiP: 43.2%
  - Perceptron: 52.4%
- False positive rate:
  - SDBP: 7.4%
  - SHiP: 7.7%
  - Perceptron: 3.2%



Perceptron Learning for Reuse Prediction

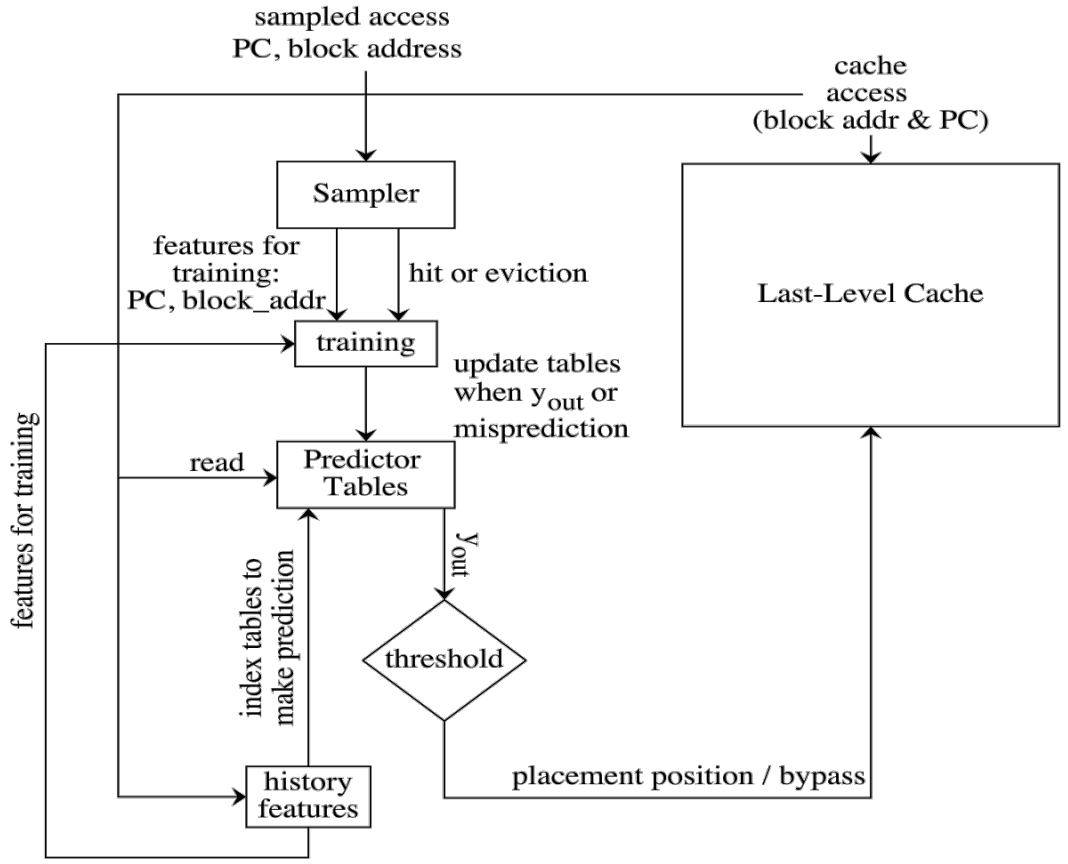
- Combine multiple features $F_{1..n}$
- Each feature indexes different table $T_{1..n}$
- y_{out} = sum of counters from tables
- Predict not reuse if $y_{out} > \tau$
- Sampler provides training data
- Perceptron Learning Rule:

if mispredict or $|y_{out}| < \theta$ then
  for $i \in 1..n$
    h = hash(F_i)
    if block is dead T[h]++
    else T[h]--



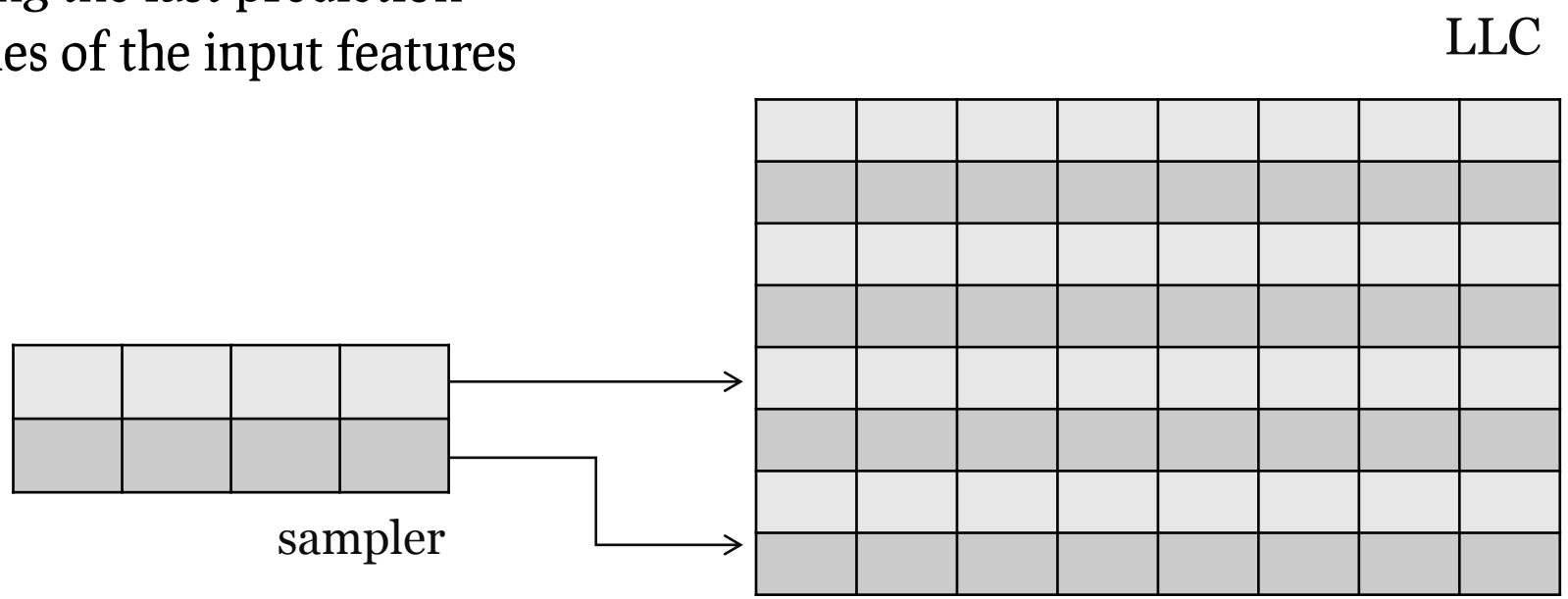
Perceptron Organization

- 6 tables
- 256 entries
- 6-bit weights
- Per-core vectors
- Features:
  - ✓ Pc_0
  - ✓ $Pc_1 >> 1$
  - ✓ $Pc_2 >> 2$
  - ✓ $Pc_3 >> 3$
  - ✓ Tag of current block >> 4
  - ✓ Tag of current block >> 7



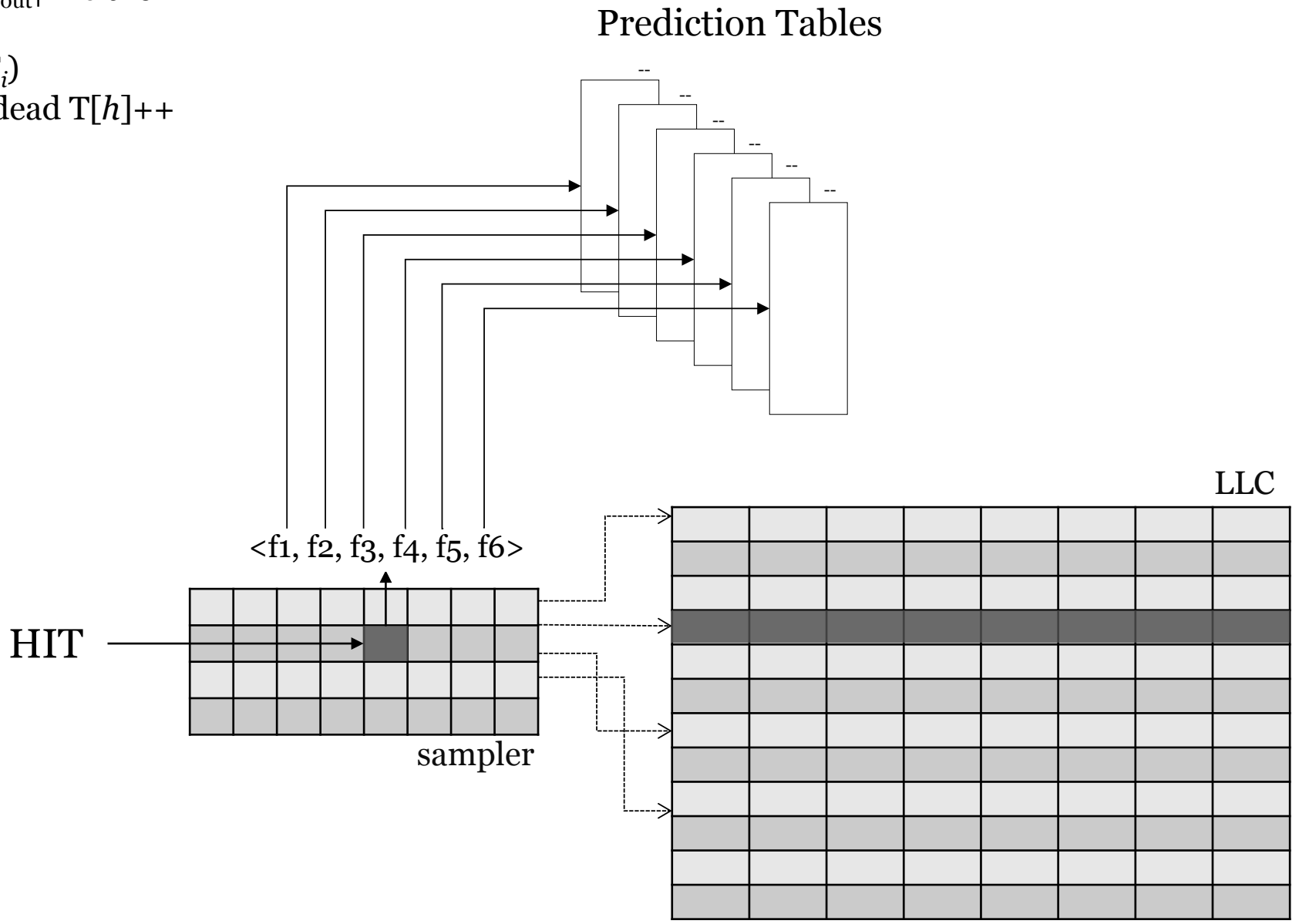
Training with Sampler

- Set sampling
- Sampler entry:
  - Partial tag (15 bits)
  - y_{out} representing the last prediction
  - Sequence of hashes of the input features
  - LRU bits



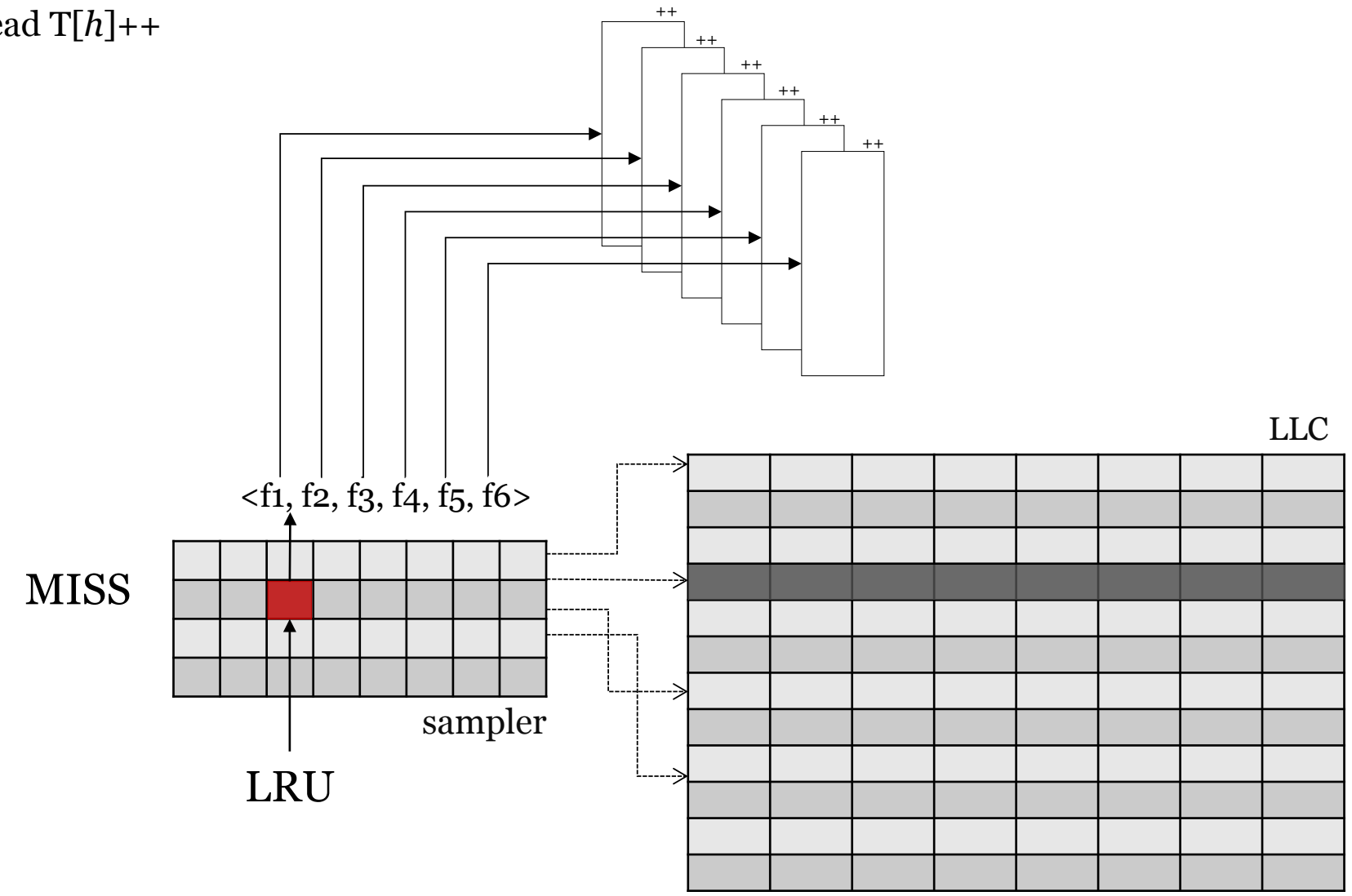
TRAINING

if mispredict or $|y_{out}| < \theta$ then
    for $i \in 1..n$
        h = hash(F_i)
        if block is dead T[h]++
        else T[h]--

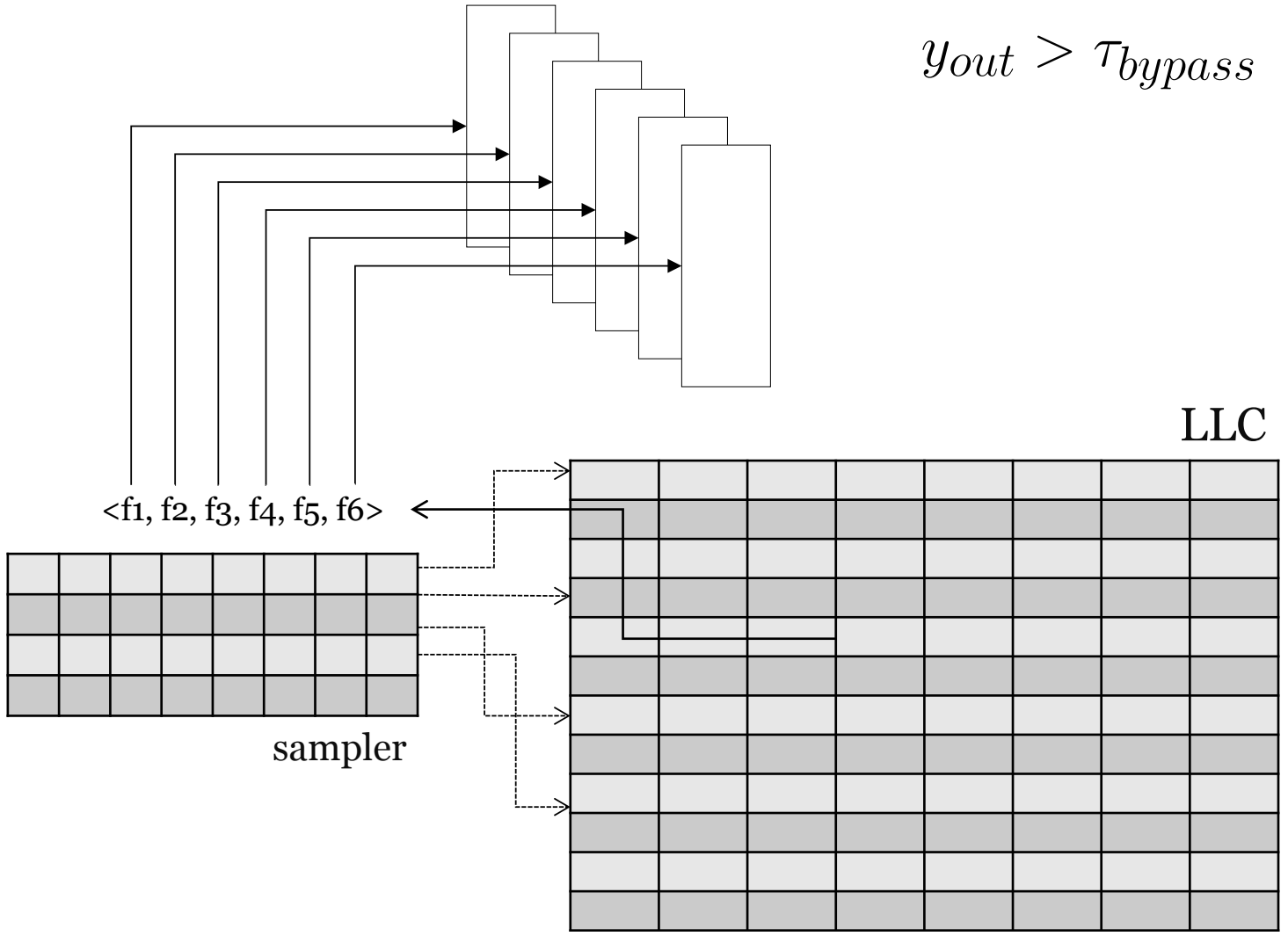


TRAINING

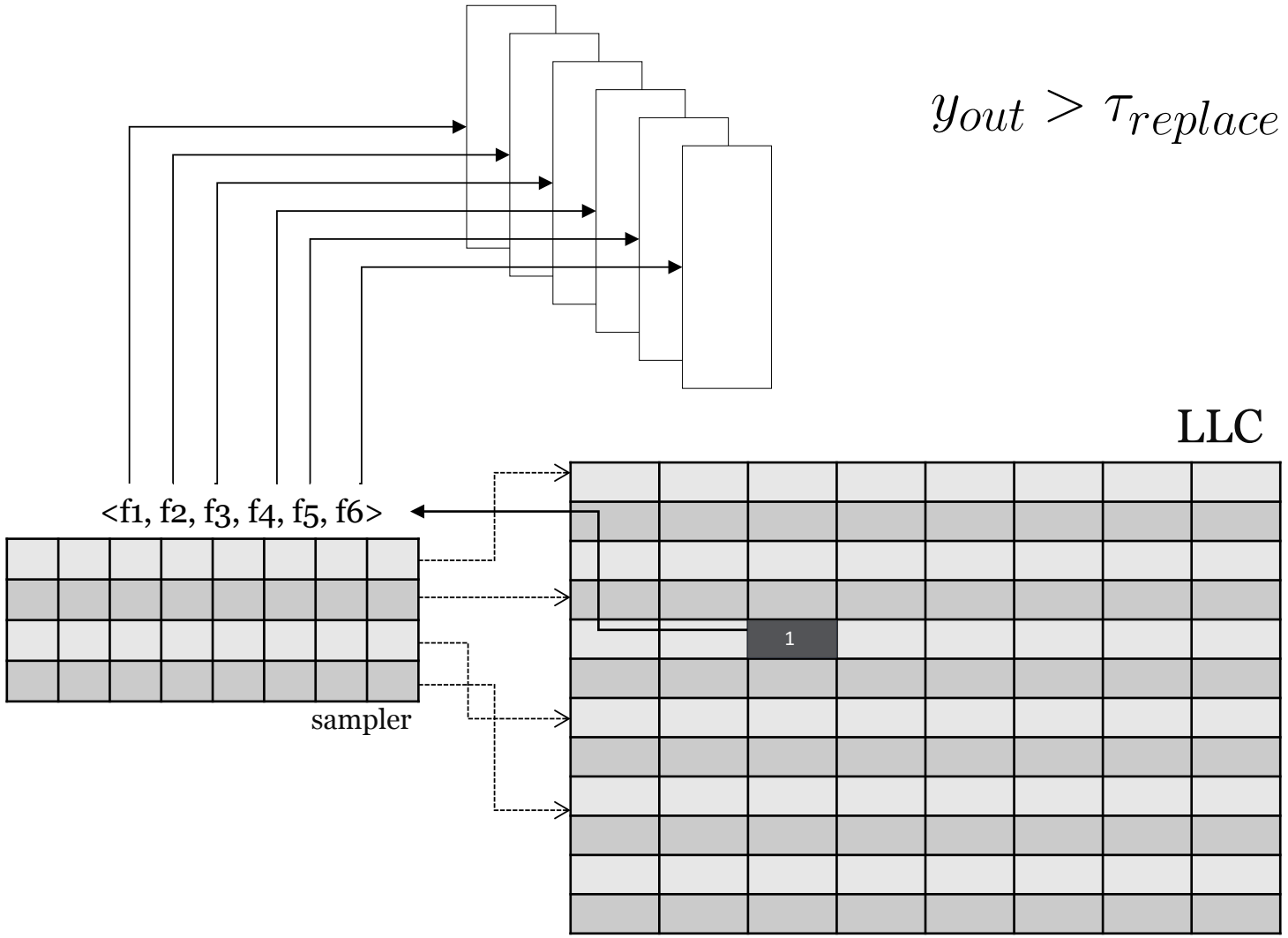
if mispredict or $|y_{out}| < \theta$ then
    for $i \in 1..n$
        h = hash(F_i)
        if block is dead T[h]++
        else T[h]--



Bypass Optimization



Replacement Optimization

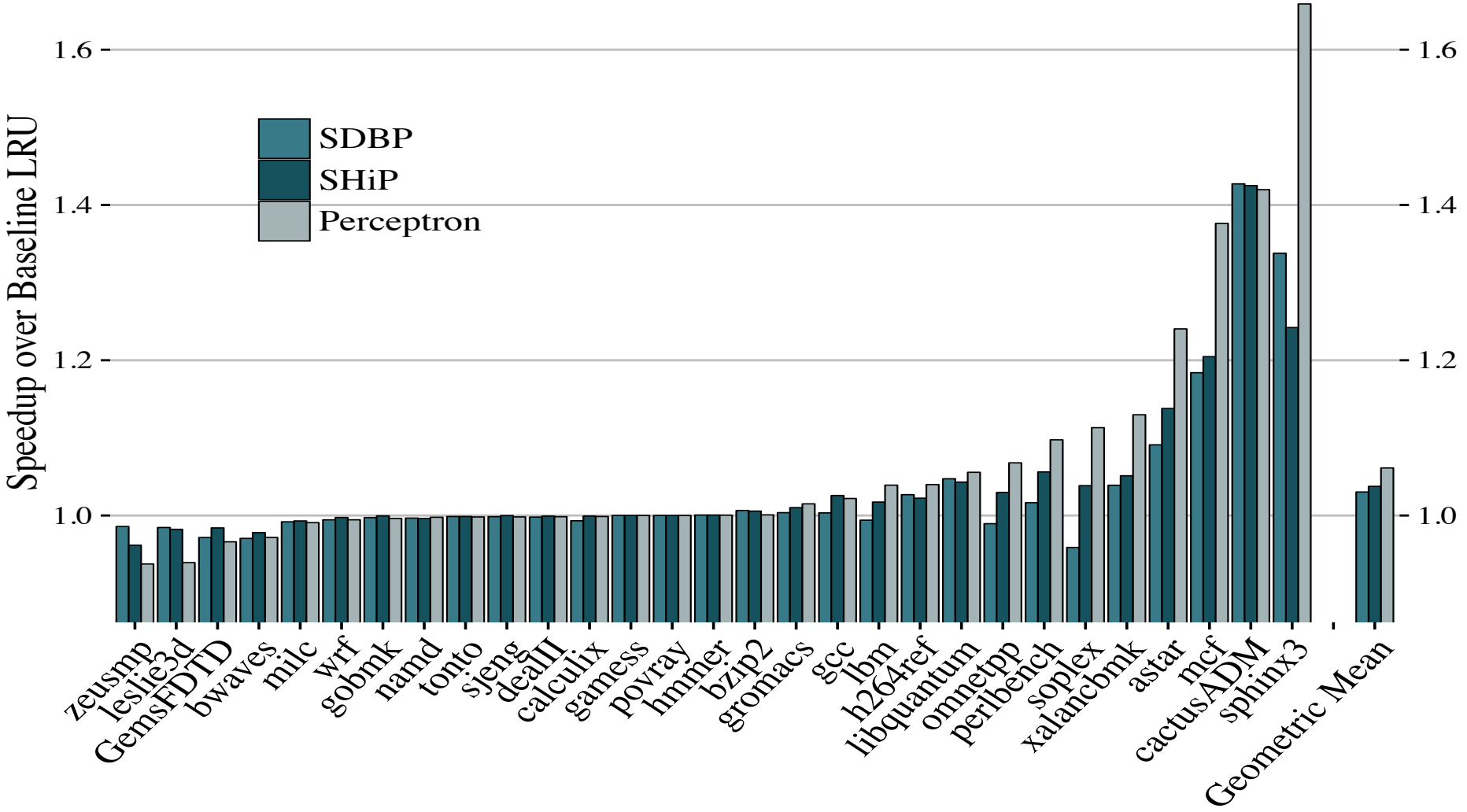


Experimental Methodology

- In-house simulator
- DL1: 32KB, 8-way
- UL2: 256KB, 8-way
- Single-thread : L3 4MB, 16-way
- Eight-core : L3 16MB, 16-way
- Stream Prefetcher
- SPEC CPU2006 benchmark suite
- Compared to:
  - SDBP *[Khan et al.]* and SHiP *[Wu et al.]*

Speedup Over LRU

Single-Threaded



SBDP: 3.5%

SHiP: 3.8%

Perceptron: 6.1%

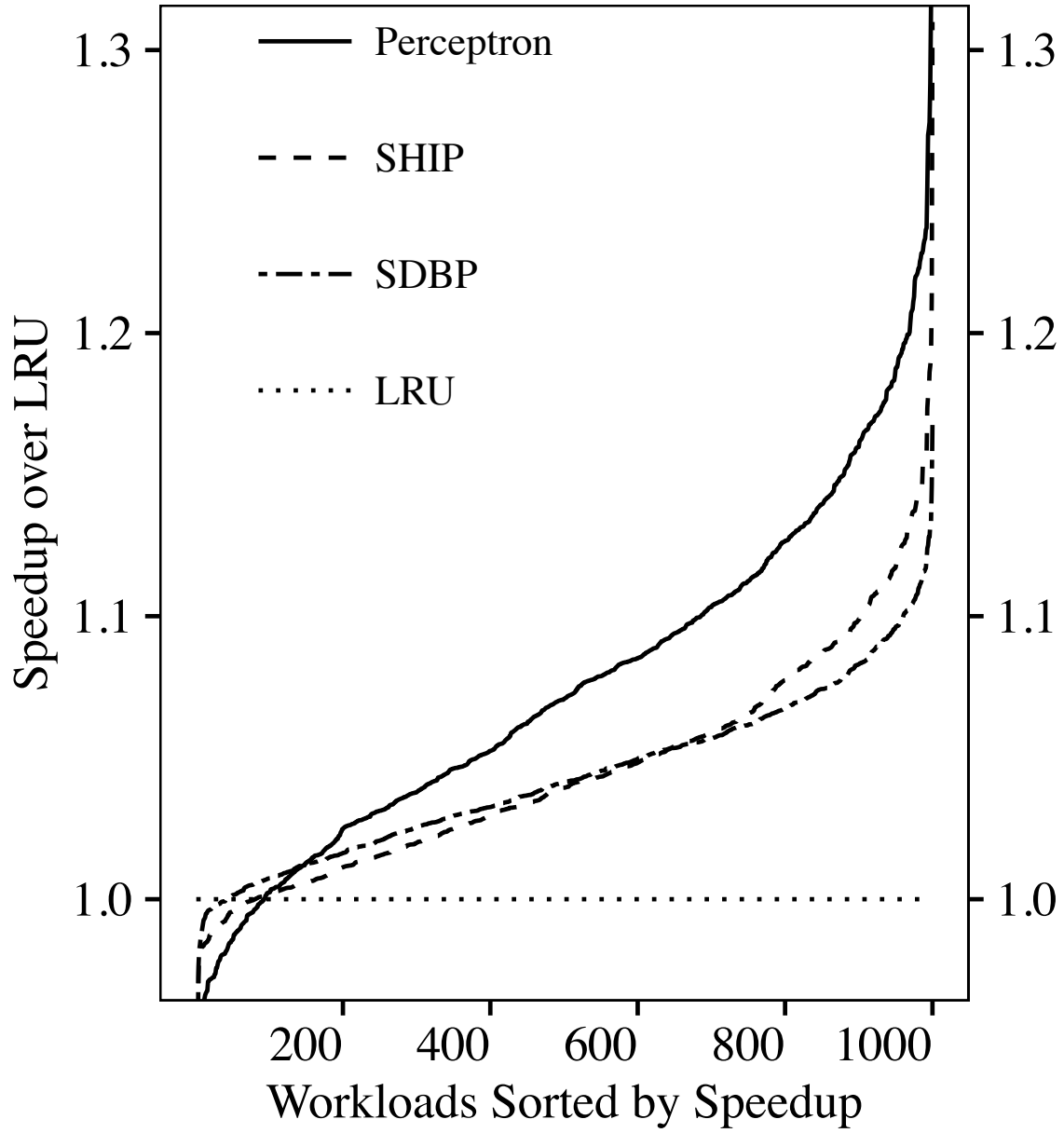
Eight-Core

Speedup

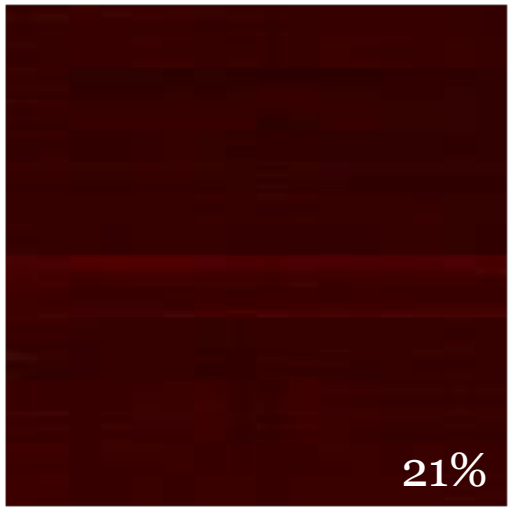
SDBP: 4.3%

SHiP: 4.4%

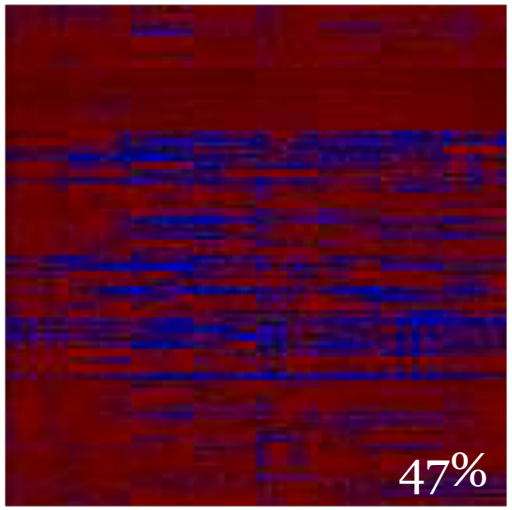
Perceptron: 7.4%



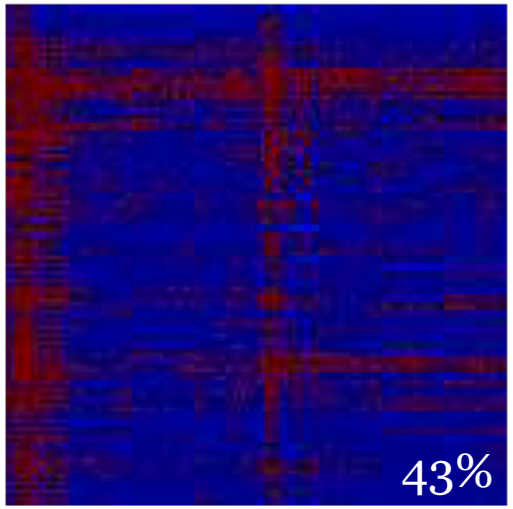
Cache Efficiency



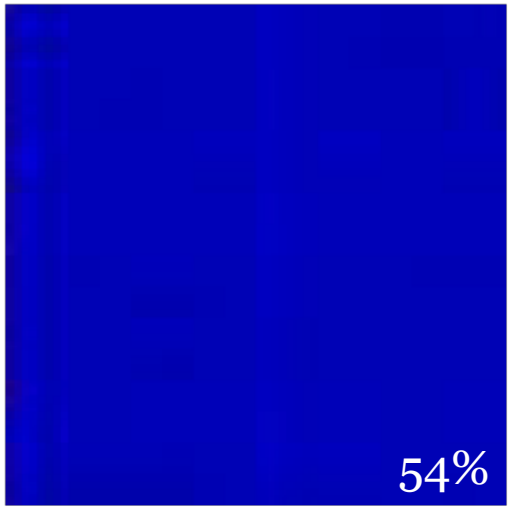
LRU



SDBP

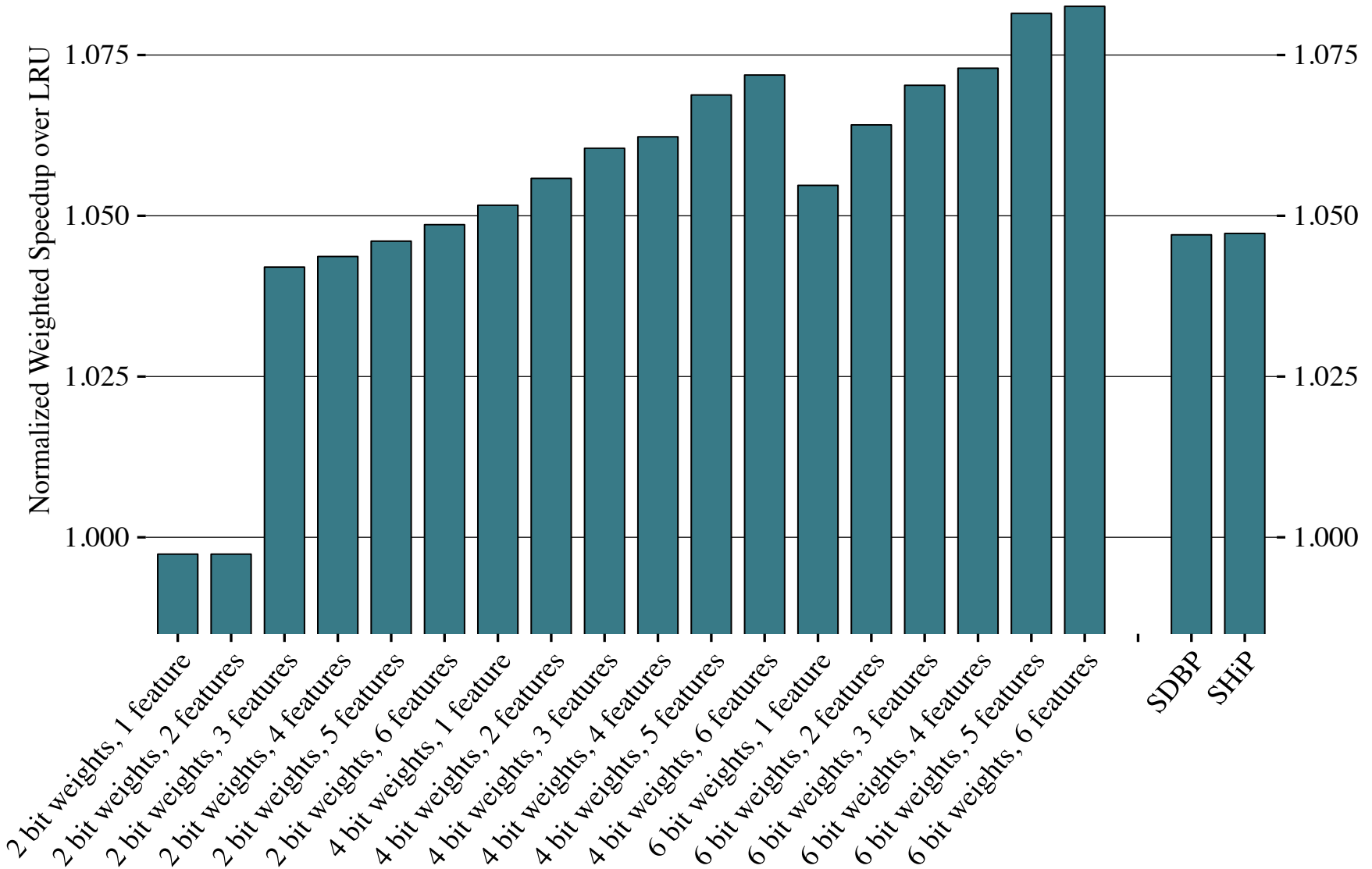


SHiP



Perceptron

Analysis


Normalized Weighted Speedup over LRU
1.075
1.050
1.025
1.000
2 bit weights, 1 feature
2 bit weights, 2 features
2 bit weights, 3 features
2 bit weights, 4 features
2 bit weights, 5 features
2 bit weights, 6 features
4 bit weights, 1 feature
4 bit weights, 2 features
4 bit weights, 3 features
4 bit weights, 4 features
4 bit weights, 5 features
4 bit weights, 6 features
6 bit weights, 1 feature
6 bit weights, 2 features
6 bit weights, 3 features
6 bit weights, 4 features
6 bit weights, 5 features
6 bit weights, 6 features
SDBP
SHiP

Multiperspective Reuse Prediction

- LLC block reuse is correlated with many different features
- We use a variety of features
- Use perceptron learning to predict block reuse
  - ~16 parameterized features
  - Chosen using stochastic search
  - Hashed perceptron organization
- Drive block placement, replacement, and bypass

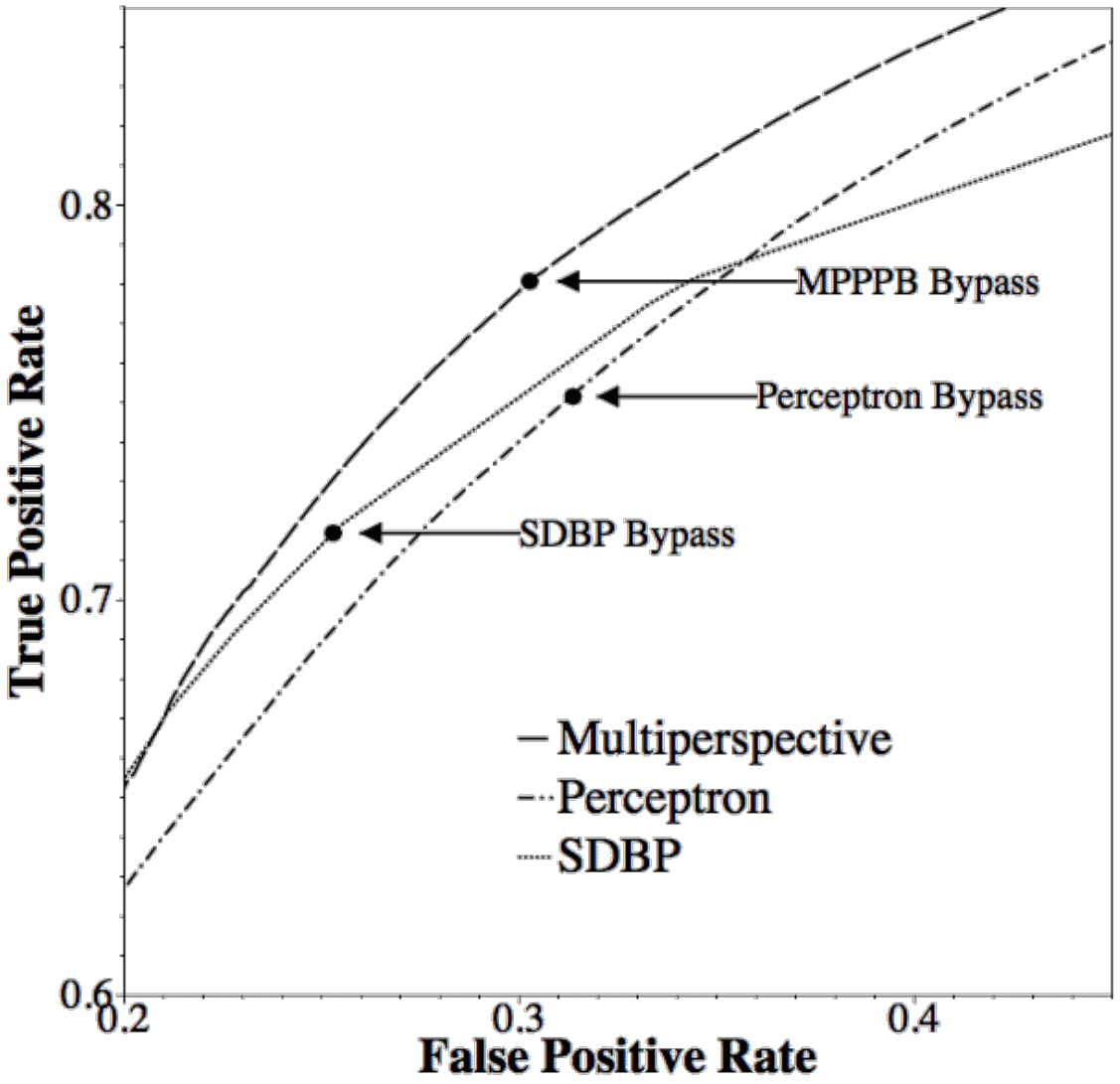
Multiperspective Approach

- Use multiple features instead of just PC and physical address
- Use stochastic search to find good features and parameters
- Drive a bypass, placement, and replacement optimization
  - Single-core policy built on our MDPP placement and promotion [HPCA 2016]
  - For multi-core the baseline policy is SRRIP [Jaleel et al. 2010]
- Multiperspective Placement Promotion and Bypass (MPPPB)

Variety of Features

- Each feature also has at least two parameters:
  - A (associativity) – position beyond which block is considered dead
  - X – if true, feature is XORed with most recent PC
- *pc*(A,B,E,W,X) – bits B to E of the Wth most recent PC
- *address*(A,B,E,X) – bits B to E of physical address
- *burst*(A,X) – 1 if access is to MRU block
- *insert*(A,X) – 1 if access is an insertion
- *lastmiss*(A,X) – 1 if last access to this set was a miss
- *offset*(A,X) – block offset from physical address
- *bias*(A,X) – 0 (tracks global tendency of blocks to be dead, an idea from our ASPLOS 2017 paper)

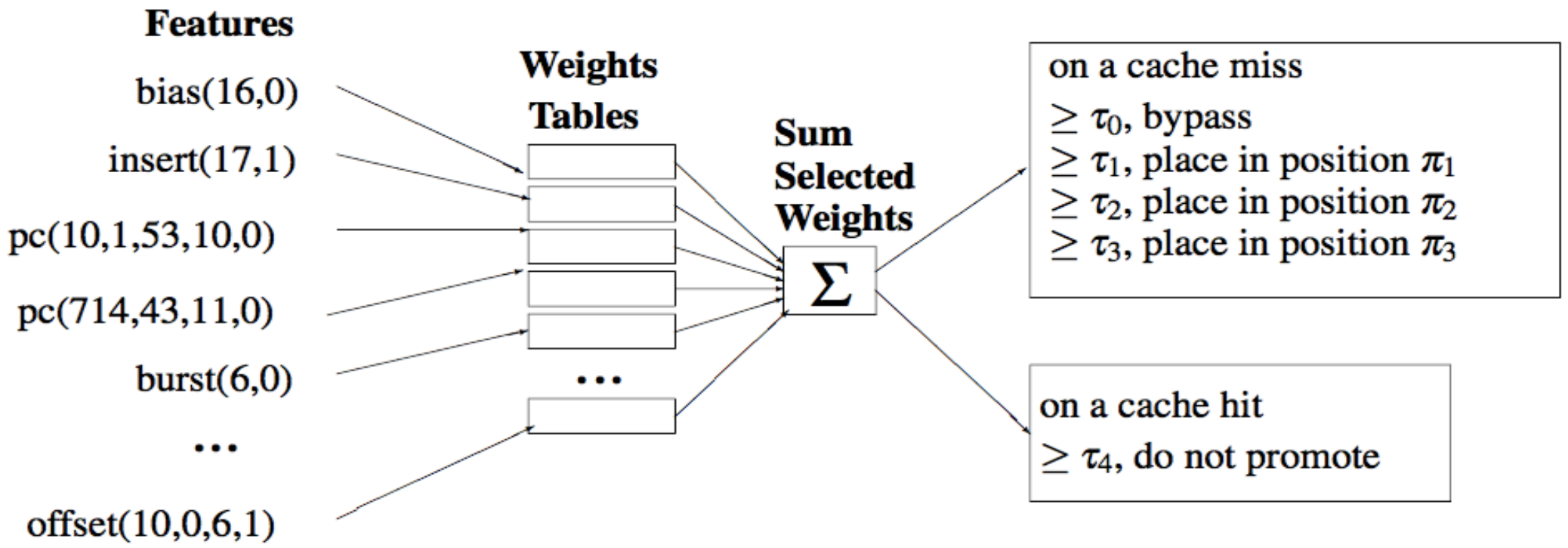
Improved Accuracy



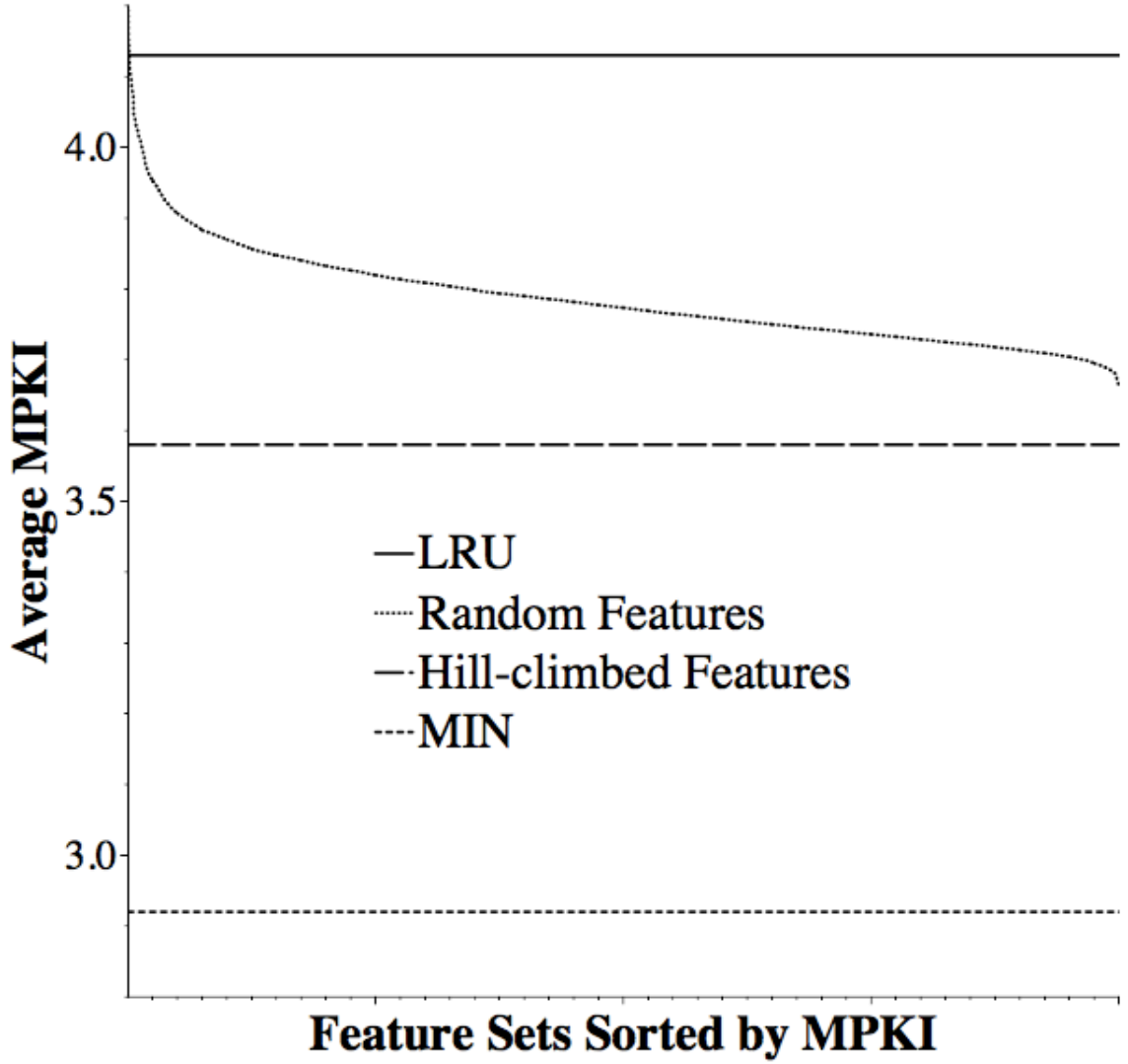
ROC curve; higher is better

Here, false positive rate is ratio of live blocks incorrectly classified as not reuse

Multiperspective Organization



Search for Features



Single-Thread Features

address(11,8,19,0)
address(16,8,16,0)
address(17,1,32,2)
bias(13,2)
bias(17,3)
burst(8,2)
insert(6,1)
insert(15,0)
insert(16,1)
lastmiss(15,2)
offset(14,0,7,2)
offset(8,1,6,2)
pc(6,4,11,2,2)
pc(6,5,48,0,3)
pc(6,5,77,4,1)
pc(17,6,20,0,1)

These features found with random search plus hill-climbing. Prefetching is enabled.

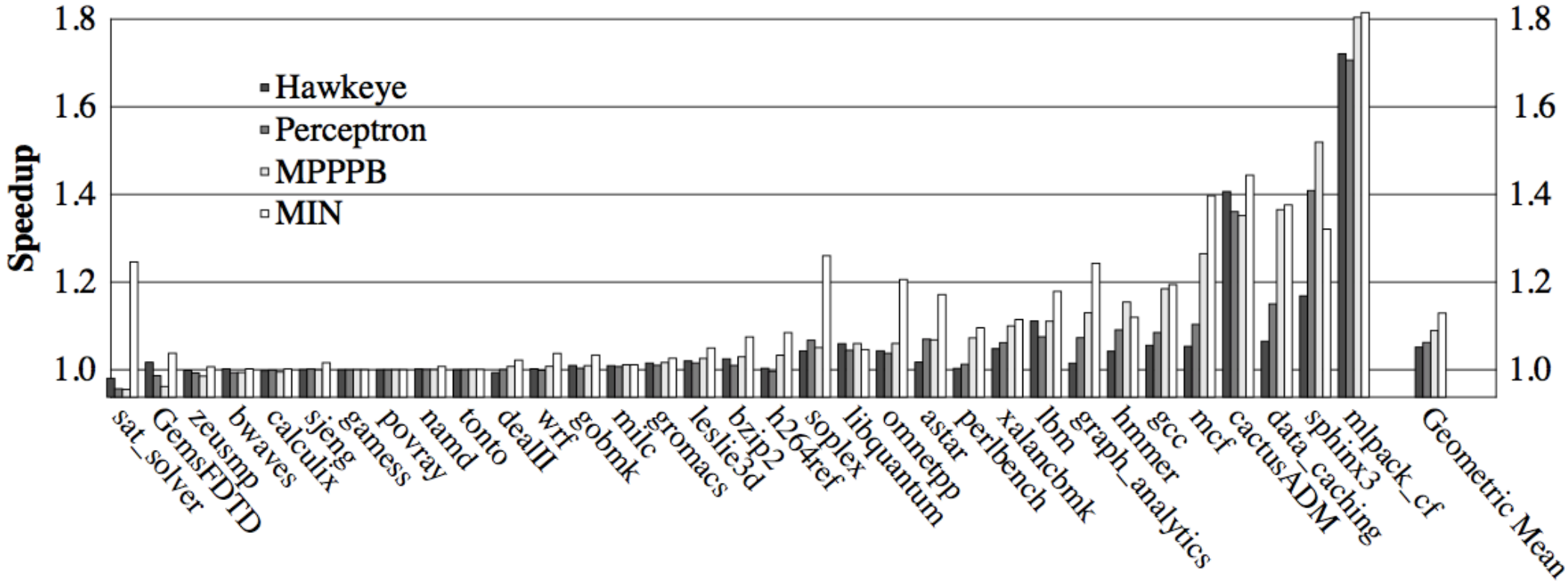
Multi-Programmed Features

address(9,2,13,2)
address(15,4,34,2)
address(16,3,14,2)
bias(13,3)
burst(16,2)
insert(6,0)
insert(10,2)
lastmiss(9,0)
offset(8,2,2,2)
offset(10,0,6,1)
offset(11,2,5,0)
offset(15,0,7,3)
pc(8,6,8,14,3)
pc(11,7,23,0,2)
pc(14,3,18,10,2)
pc(17,1,14,5,0)

Methodology

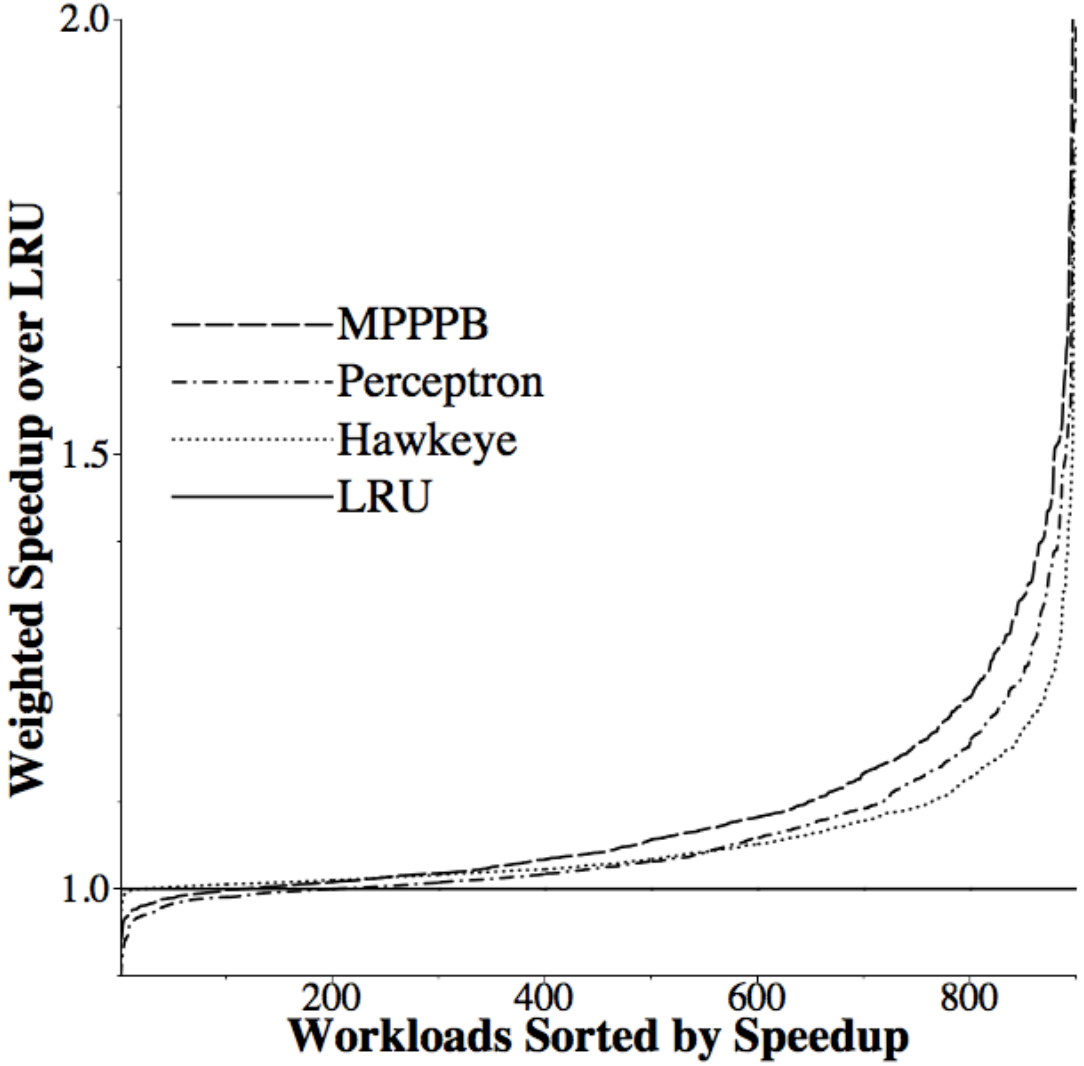
- Trace based simulator
- DL1: 32KB, 8-way
- UL2: 256KB, 8-way
- Single-thread : L3 2MB, 16-way
- Four-core : L3 8MB, 16-way
- Stream Prefetcher
- SPEC CPU 2006 and 2017 benchmark suites
- Compared to :
  - Hawkeye [Jain and Lin 2016], Perceptron [Us 2016], MIN [Bélády 1966]

Single-Thread Performance



geomean speedup:
Hawkeye 5.1%, Perceptron 6.3%, MPPPB 9.0%, MIN 12.8%

Multi-Programmed Performance



Geomean Weighted Speedup
Hawkeye: 5.2%
Perceptron: 5.8%
MPPPB: 8.3%

Thank you!
Questions?