

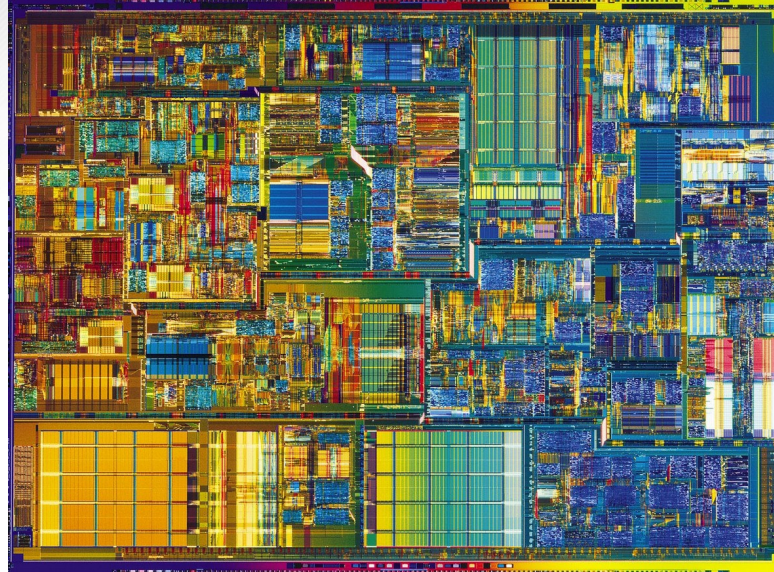


# **SPIN and DRAIN: A new approach to address deadlocks in interconnection networks**

**Paul V. Gratz**

*Computer Engineering and Systems Group  
Department of Electrical and Computer Engineering  
Department of Computer Science and Engineering  
Texas A&M University*

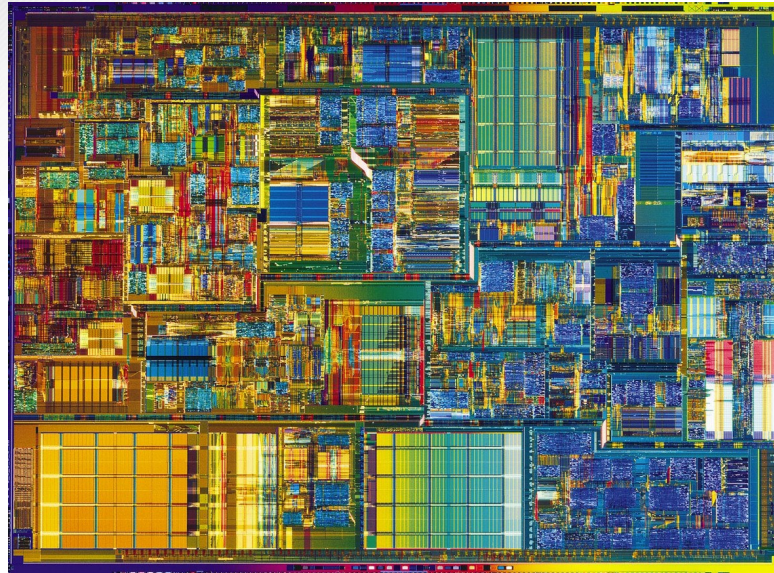
# First, a history lesson...



Intel Pentium 4  
Circa 2001

- 4-way Out-of-Order Superscalar
- High performance branch predictor
- Execution trace cache
- “Hyper”-pipelining
- “Hyper”-threading
- **Double-pumped ALU**
- 126 instructions in flight
- 6-uOps issue/cycle
- SSE Instructions
- 96 (48)-entry Load (Store) buffer
- 2\*128-entry regfile

# First, a history lesson...



Intel Pentium 4  
Circa 2001

- 4-way Out-of-Order Superscalar
- High performance branch predictor
- Execution trace cache
- “Hyper”-pipelining
- “Hyper”-threading

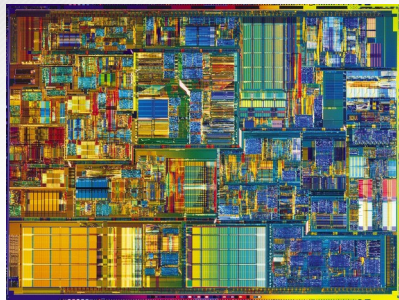
*Deep Speculation*

- Double-pumped ALU
- 126 instructions in flight
- 6-uOps issue/cycle
- SSE instructions
- 96 (48)-entry Load (Store) buffer
- 2\*128-entry regfile

*High Power*

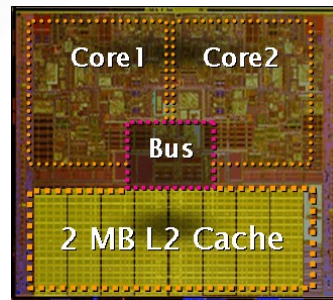
# First, a history lesson...

2001



Pentium 4

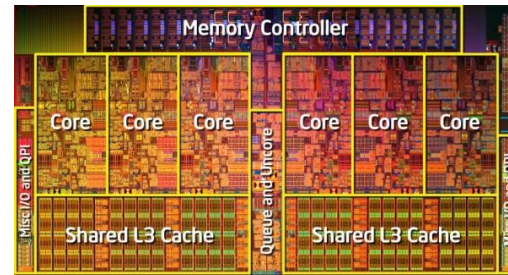
2006



Core 2 Duo

- Dual-core
- Pentium III-like microarchitecture
- 2-level memory hierarchy
- Simple bus interconnect

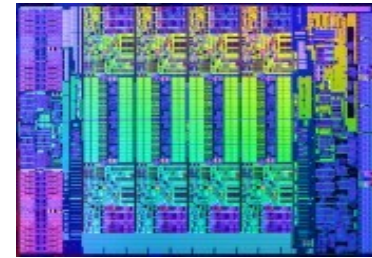
2008



Core i7 (Nehalem)

- Six-core
- Similar microarchitecture to Core-2
- 3-level memory hierarchy
- Crossbar interconnect

2014

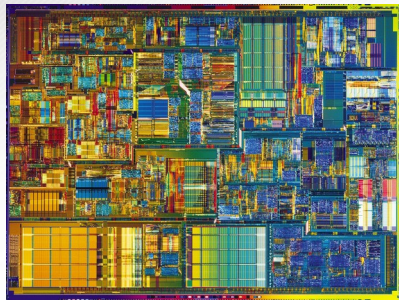


Core i7 (Haswell)

- Eight-core
- Incrementally advanced microarchitecture
- 3-level memory hierarchy
- Hierarchical interconnect

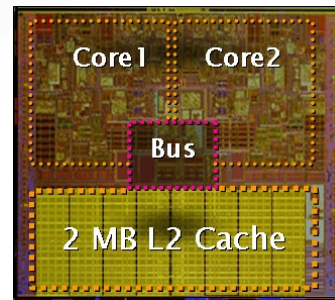
# First, a history lesson...

2001



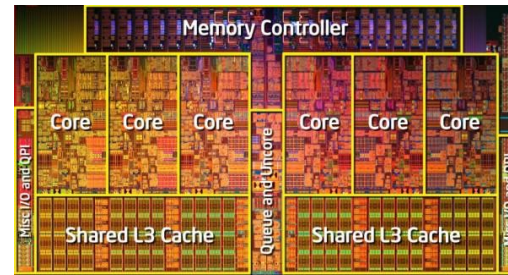
Pentium 4

2006



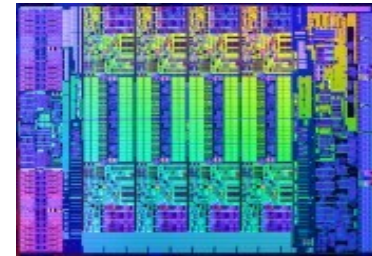
Core 2 Duo

2008



Core i7 (Nehalem)

2014



Core i7 (Haswell)

## VLSI Trends:

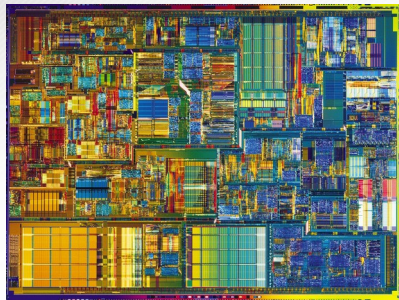
- Continued transistor scaling
- Constrained Power and Energy budgets

- Six-core
- Similar microarchitecture to Core-2
- 3-level memory hierarchy
- Crossbar interconnect

- Eight-core
- Incrementally advanced microarchitecture
- 3-level memory hierarchy
- Hierarchical interconnect

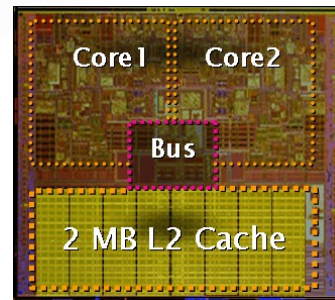
# First, a history lesson...

2001



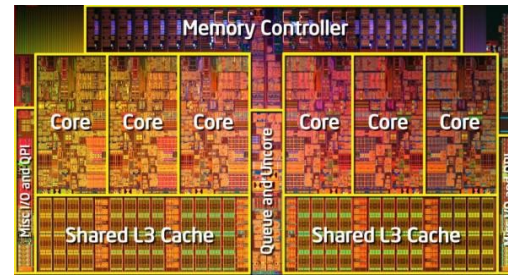
Pentium 4

2006



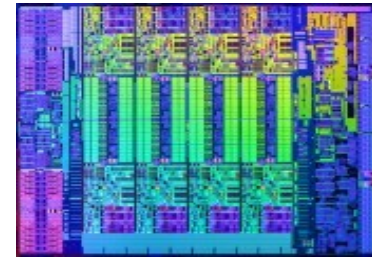
Core 2 Duo

2008



Core i7 (Nehalem)

2014



Core i7 (Haswell)

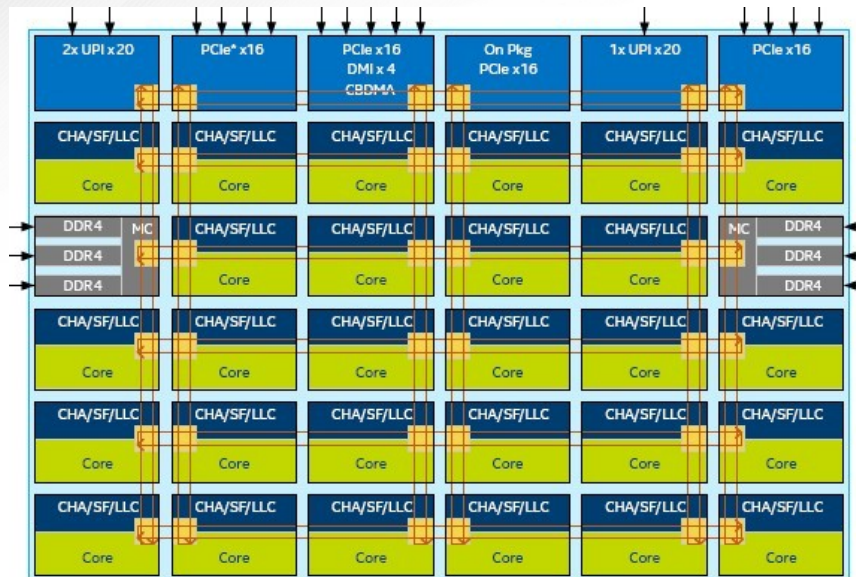
## VLSI Trends:

- Continued transistor scaling
- Constrained Power and Energy budgets

## Outcomes:

- Core counts increasing
  - Core complexity stalled (mostly)
- Memory hierarchy system complexity increasing
- Interconnect between cores increasingly critical

Era of Chip-multiprocessors (CMPs): Complexity moves from the cores up the memory system hierarchy and interconnect.

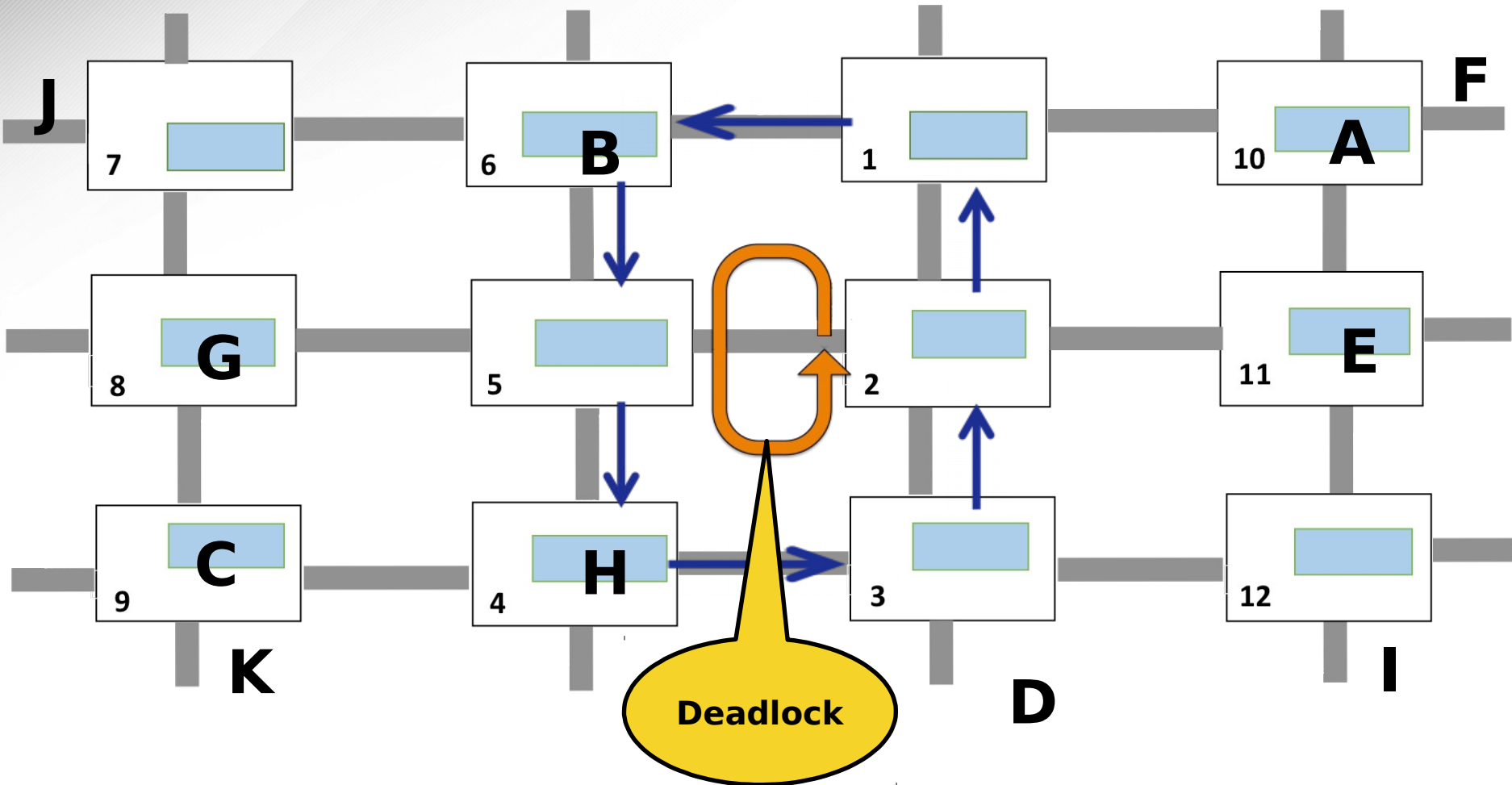


Intel Skylake Xeon XCC (2017)

- Increasing core counts
- Multi-level hierarchies
  - Private lower levels
  - Large, shared last-level
- Multi-threaded apps
  - Data shared via caches
  - Synchronization critical

- All is not well:
  - Caches don't help for first access to a particular location
    - (A focus of much of my other work)
  - Huge latencies for shared data/synchronization due to coherence
    - ***Interconnect between cores an increasingly critical design component***

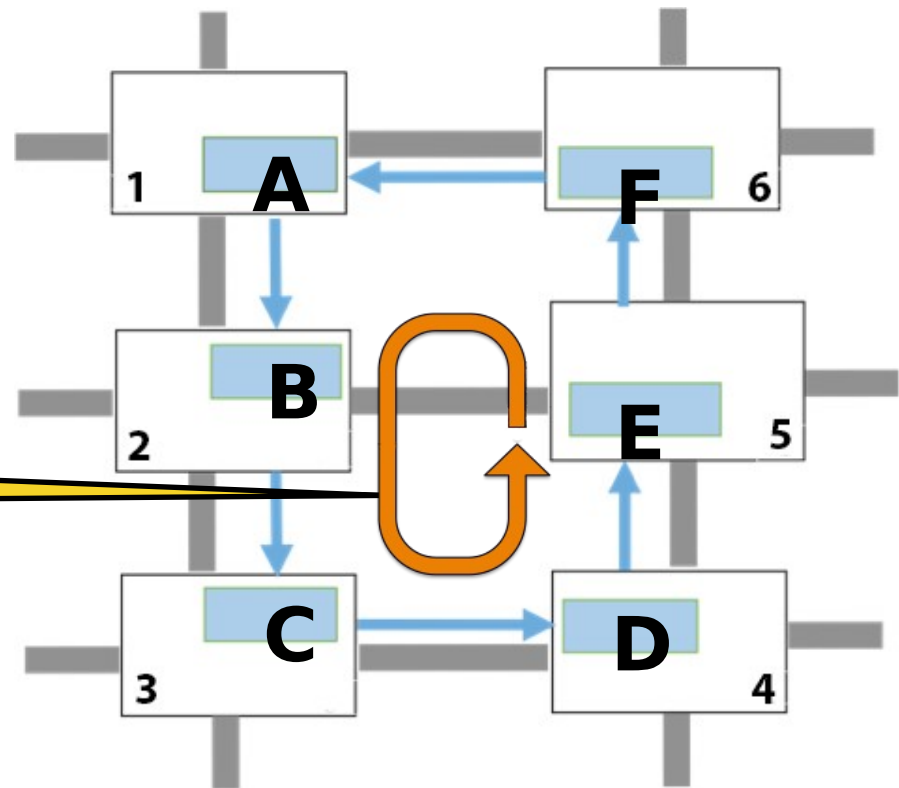
# Network Routing





# Routing Deadlocks

- A Routing Deadlock is a *cyclic buffer dependency chain* that renders forward progress impossible.



Deadlock

# Routing Deadlocks

- A **Routing Deadlock** is a cyclic buffer dependency chain that renders forward progress impossible.
  - Renders the chip non-functional.
- Deadlocks are a fundamental problem in both off-chip and on-chip interconnection networks.
  - Deadlocks are hard to detect during functional verification.
    - Manifest after a long use time.
    - Depend on : traffic pattern, injection rate, congestion.
- Existing approaches to prevent deadlocks require high hardware costs or impose significant performance penalties
  - Need a low cost solution for functional correctness !!

**Focus of this work is to achieve low-cost, high performance deadlock freedom**

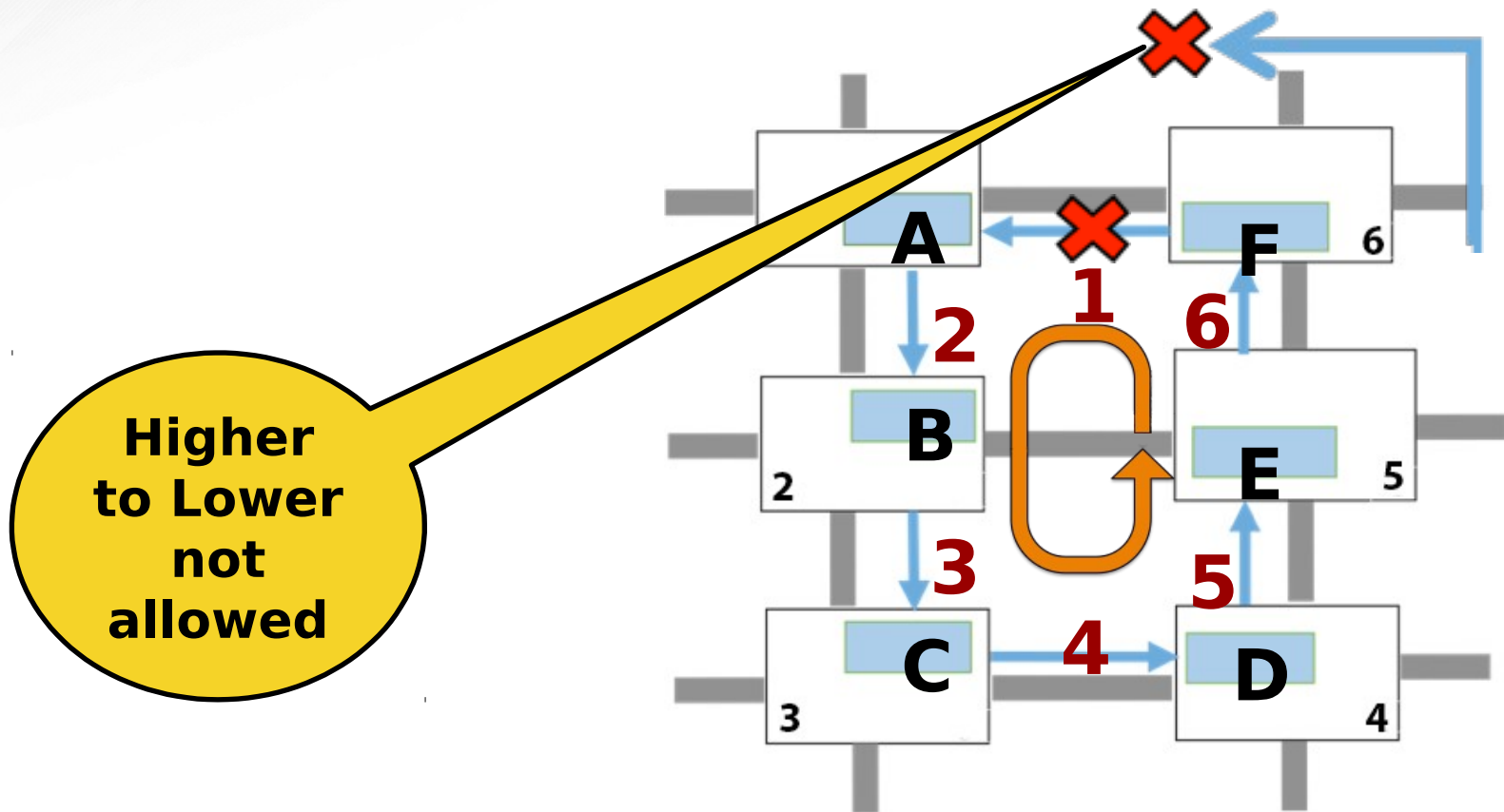


# Outline

- Introduction
- Background: Routing Deadlocks
  - Dally's Theory
  - Duato's Theory
  - Flow Control Routing
  - Deflection Routing
- **SPIN** : **S**ynchronized **P**rogress in **I**nterconnection **N**etworks
- **DRAIN** : **D**eadlock **R**emoval for **A**rbitrary **I**rregular **N**etworks
- Conclusion

# Solution I: Dally's Theory

- A ***strict order*** in acquisition of links and/or buffers ensures a cyclic dependency can never be created.

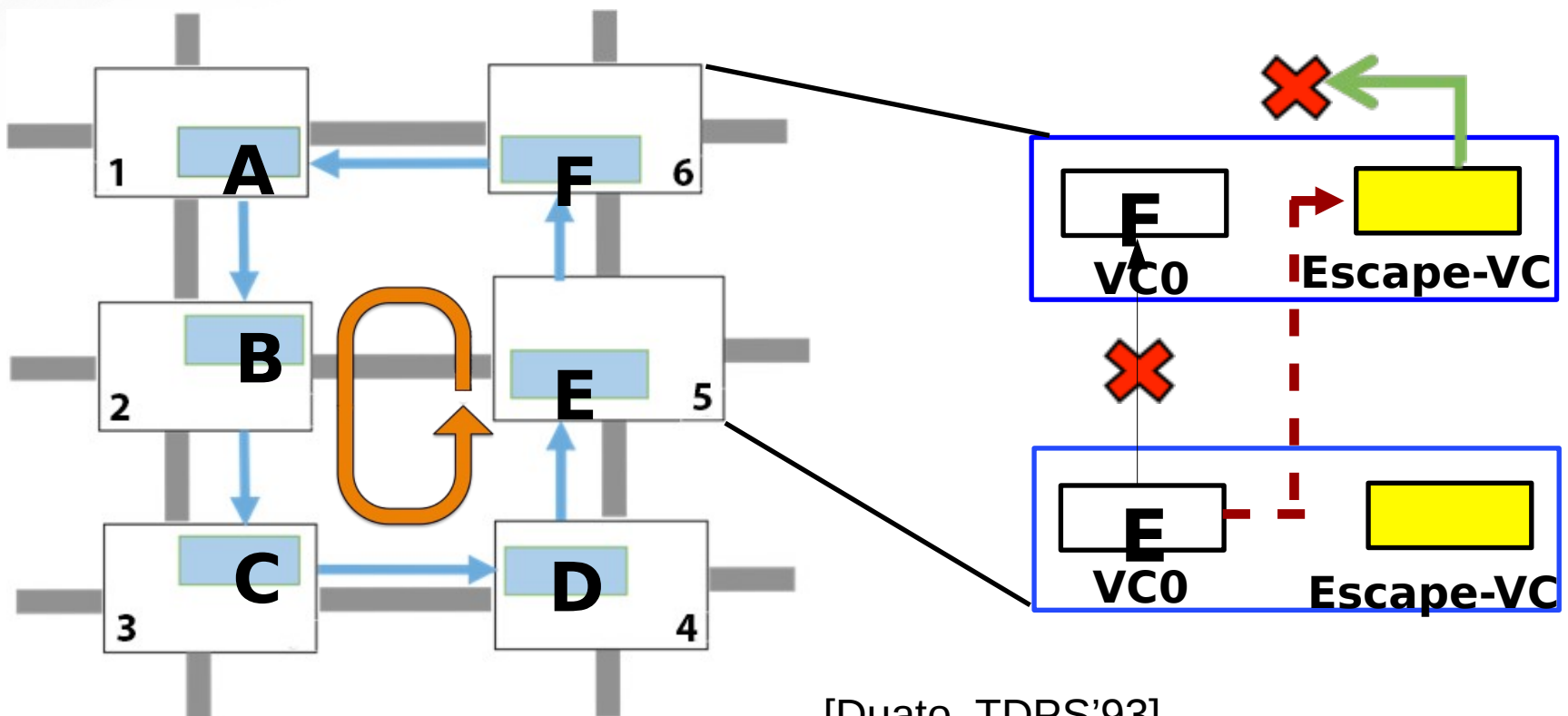


# Solution I: Dally's Theory

- A ***strict order*** in acquisition of links and/or buffers ensures a cyclic dependency can never be created.
- ***Implementations:*** Turn model [Glass and Ni, ISCA'92]  
XY routing, Up-Down routing [Schroeder et al, ICCP'91]
- Limitations:
  - ***Routing Restrictions:*** Increased Latency, Throughput loss, Energy overhead
  - Require large no. of VCs for fully adaptive routing.

# Solution II: Duato's Theory

- *Adds buffers* to create a *deadlock free escape path* that can be used to avoid/recover from deadlocks.
- *Implementation:* turn restrictions in escape-VC.



## Solution II: Duato's Theory

- **Adds buffers** to create a **deadlock free escape path** that can be used to avoid/recover from deadlocks.
- **Implementation:** Requires an extra “escape” VC in each router. Turn restrictions in escape-VC.
- Limitations:
  - **Energy** and **Area overhead** of escape VCs.
  - Additional routing tables/logic for routing within escape-VC.

- **Solution III: Flow Control**
  - **Restrict injection** when no. of empty buffers fall below a threshold
  - **Implementation:** Bubble Flow Control [Carrion et al., HIPC'97]
  - Limitation: Implementation Complexity, Throughput Loss.
  
- **Solution IV: Deflection Routing**
  - Assign every flit to some output port even if they get **misrouted**.
  - **Implementation:** BLESS [Moscibroda and Mutlu, ISCA'09]  
CHIPPER [Fallin et al., HPCA'11]
  - Limitation: **Livelocks**, non-minimal routing



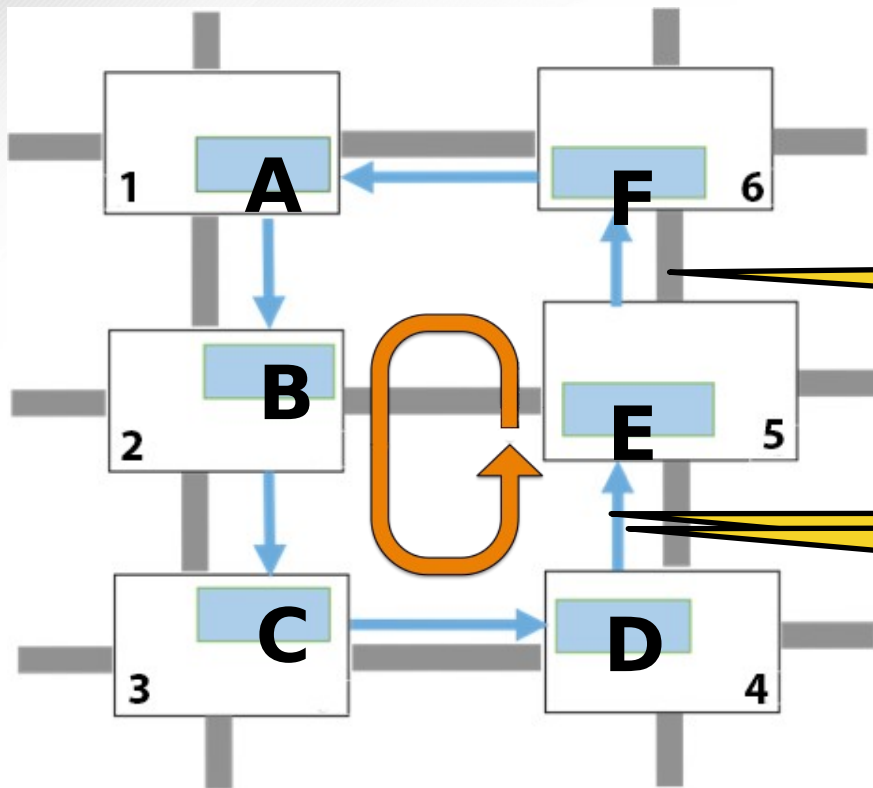
# Deadlock Freedom Theories

Metric Theory	Acyclic CDG not Required	No Packet Injection Restrictions	Livelock Free	VC cost for Mesh Routing		Topology Indepen- dent
				Minimal	Adaptive	
Dally	✗	✓	✓	1	6	✗
Duato	<div style="background-color: yellow; border: 2px solid black; padding: 10px; text-align: center;"> <p>Can we do <u>better</u> ??</p> </div>					✗
Flow Control						✓
Deflection Routing	✓	✗	✗	✗	1	✓
SPIN	✓	✓	✓	1	1	✓

- Introduction
- Background: Routing Deadlocks
- **SPIN** : **S**ynchronized **P**rogress in **I**nterconnection **N**etworks
  - Key idea
  - Case study: implementation/microarch
  - **FAvORS**: **F**ully **A**daptive, **O**ne-vc **R**outing with **S**PIN
  - Evaluation
- **DRAIN** : **D**eadlock **R**emoval for **A**rbitrary **I**rregular **N**etworks
- Conclusion

# SPIN : Key Idea

- Deadlocks are the result of a *lack of coordination* not a lack of resources.



## What if:

We **coordinate** the movement of every packet at a given time ??

spin complete

Deadlock

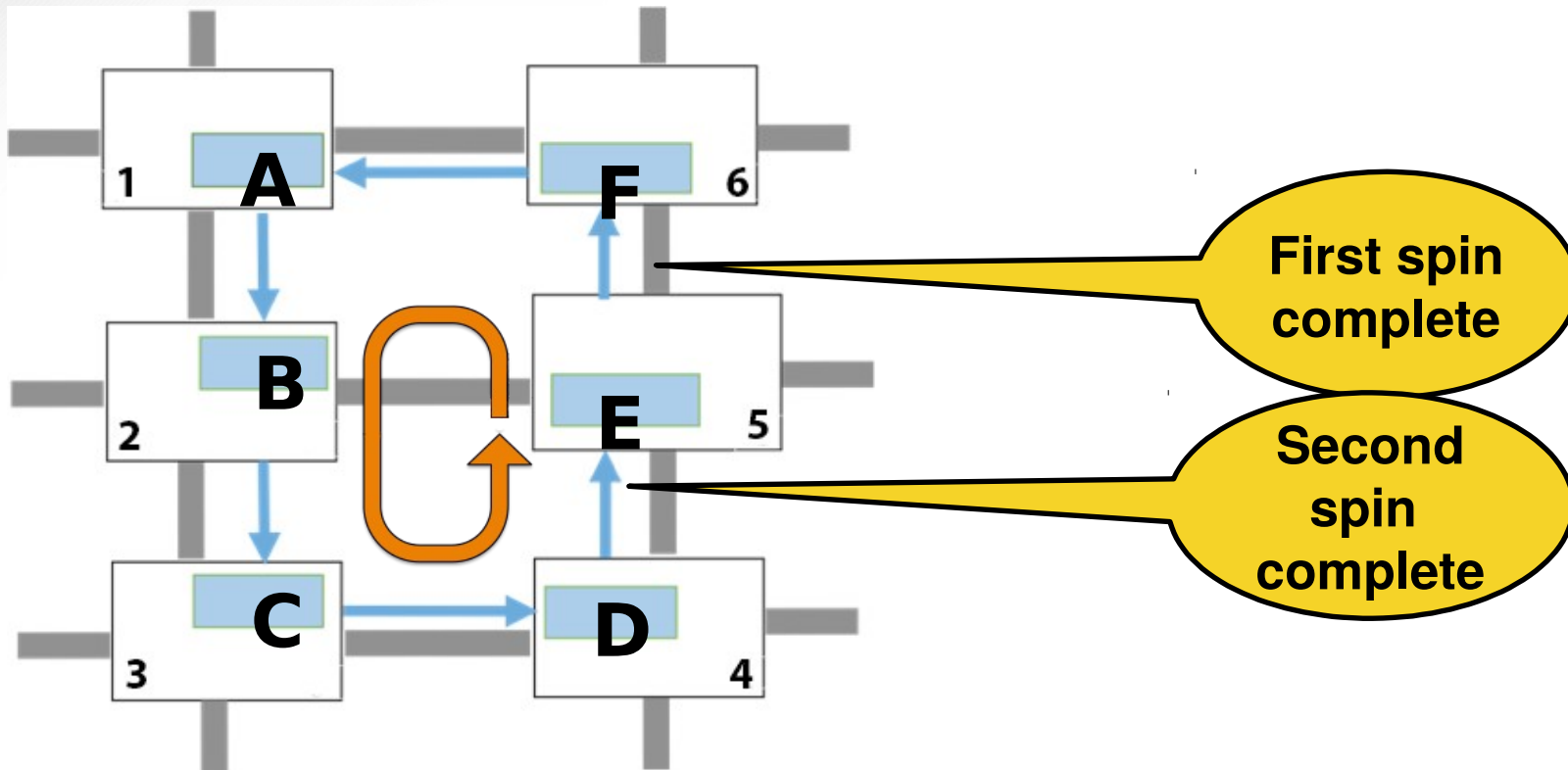
Simultaneous Synchronized Movement

- *Simultaneous Synchronized Movement* of all deadlocked packets in the loop is called a *spin*.

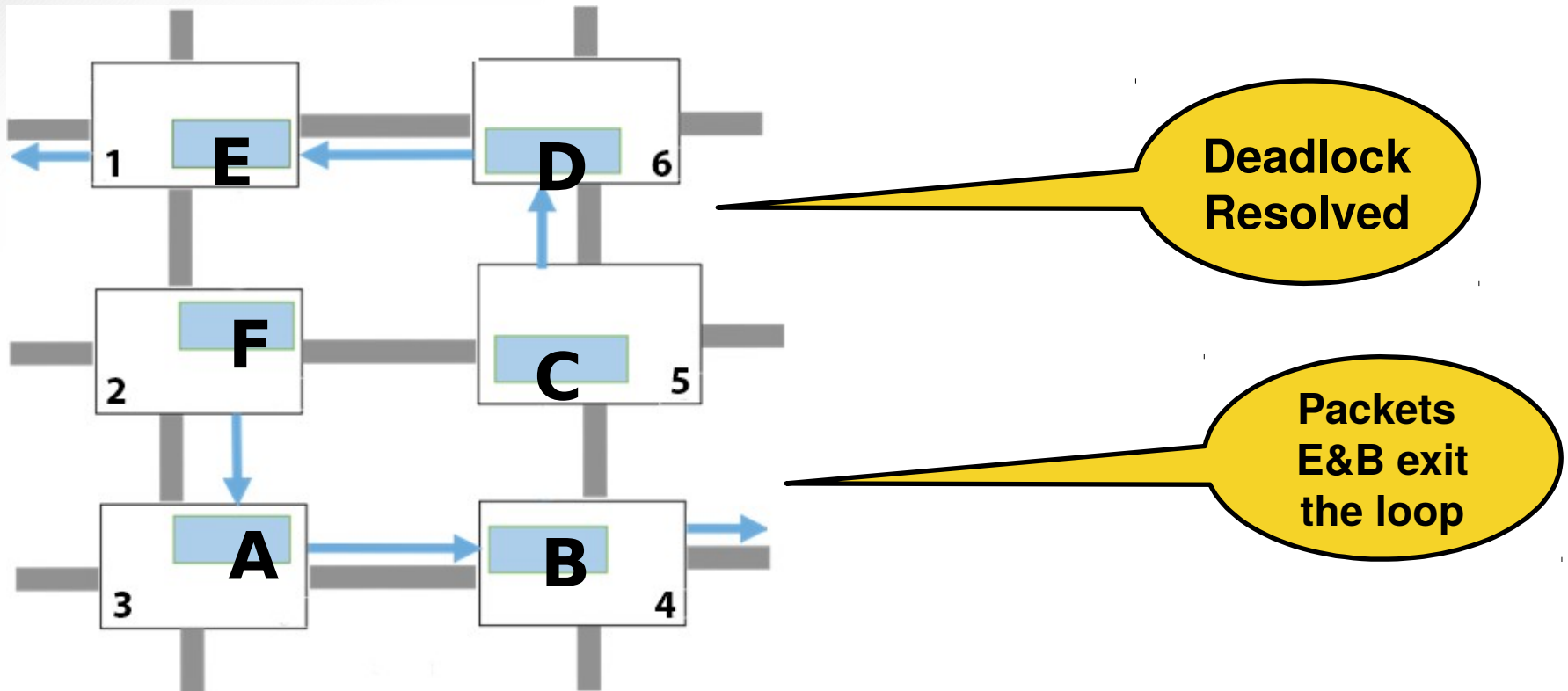
# SPIN : Key Idea

- ***Simultaneous Synchronized Movement*** of all deadlocked packets in the loop is called a ***spin***.
  - Each spin leads to ***one hop of forward movement*** for all deadlocked packets.
  - One spin may not resolve the deadlock. If so, spin can be repeated
  - Deadlock is ***guaranteed to be resolved*** in a ***finite*** number of spins [proof in paper, Sec. III]

# SPIN : Key Idea




# SPIN : Key Idea



# SPIN: Implementation

- SPIN is a generic deadlock freedom theory that can have *multiple implementations*.
- We choose a *recovery approach* as *deadlocks* are *rare* scenarios (See Sec. II-F).
- *Our Implementation*:
  - Detect the Deadlock.
  - Coordinate a time for spin.
  - Execute the spin.

## Example : Detect Deadlocks

- Use *counters*.
- Placed at *every node* at design time.
  - Optimize by exploiting topology symmetry (See Static Bubble).
- If packet does not leave in *threshold time* (configurable), it indicates a *potential deadlock*.
- *Counter expired?*  Send *probe* to verify deadlock.

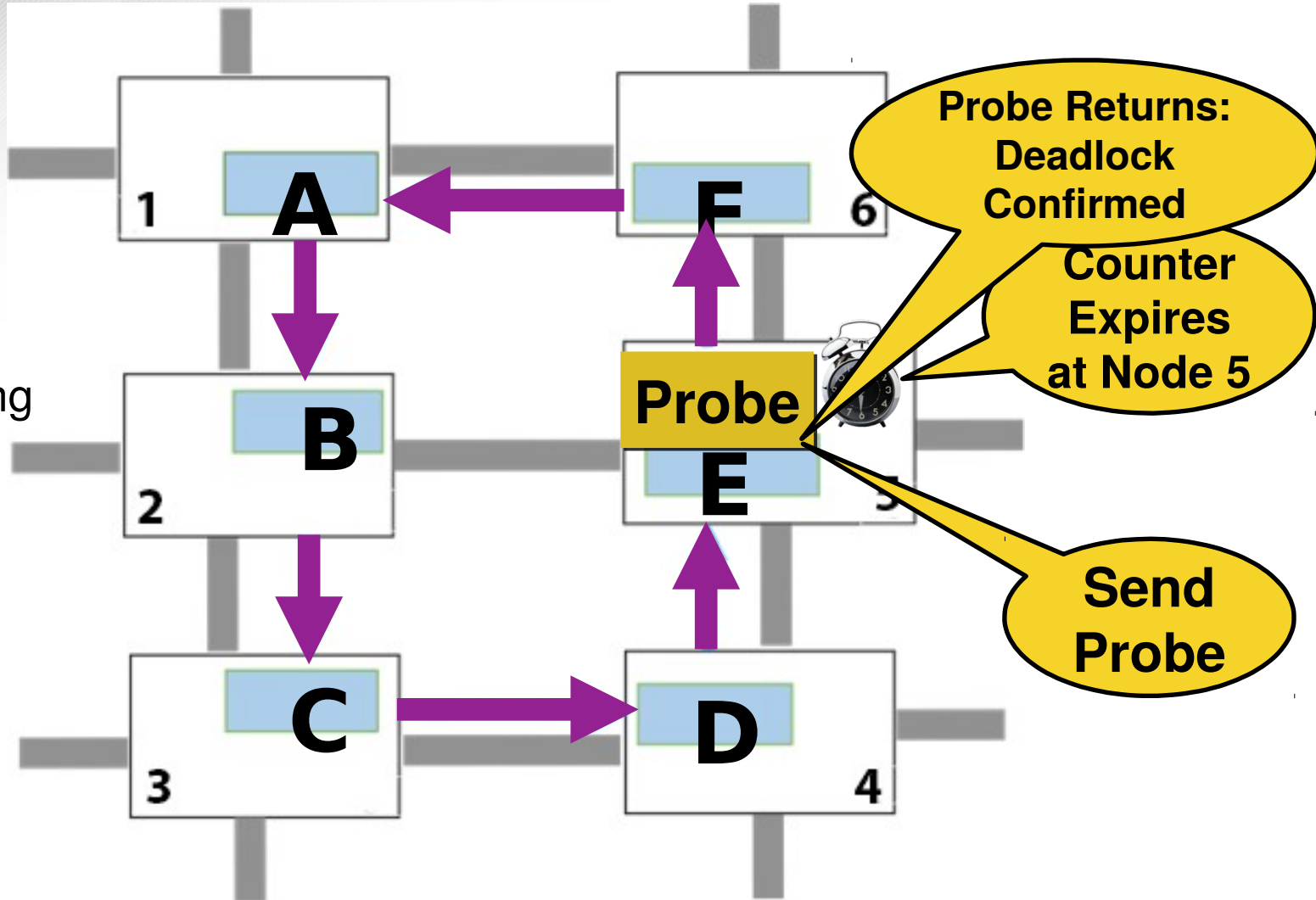


# Example : *Probe Msg.*

## 1. Deadlock Detection

2. Coordinating the spin.

3. Executing the spin.



# Example : *Probe Msg.*

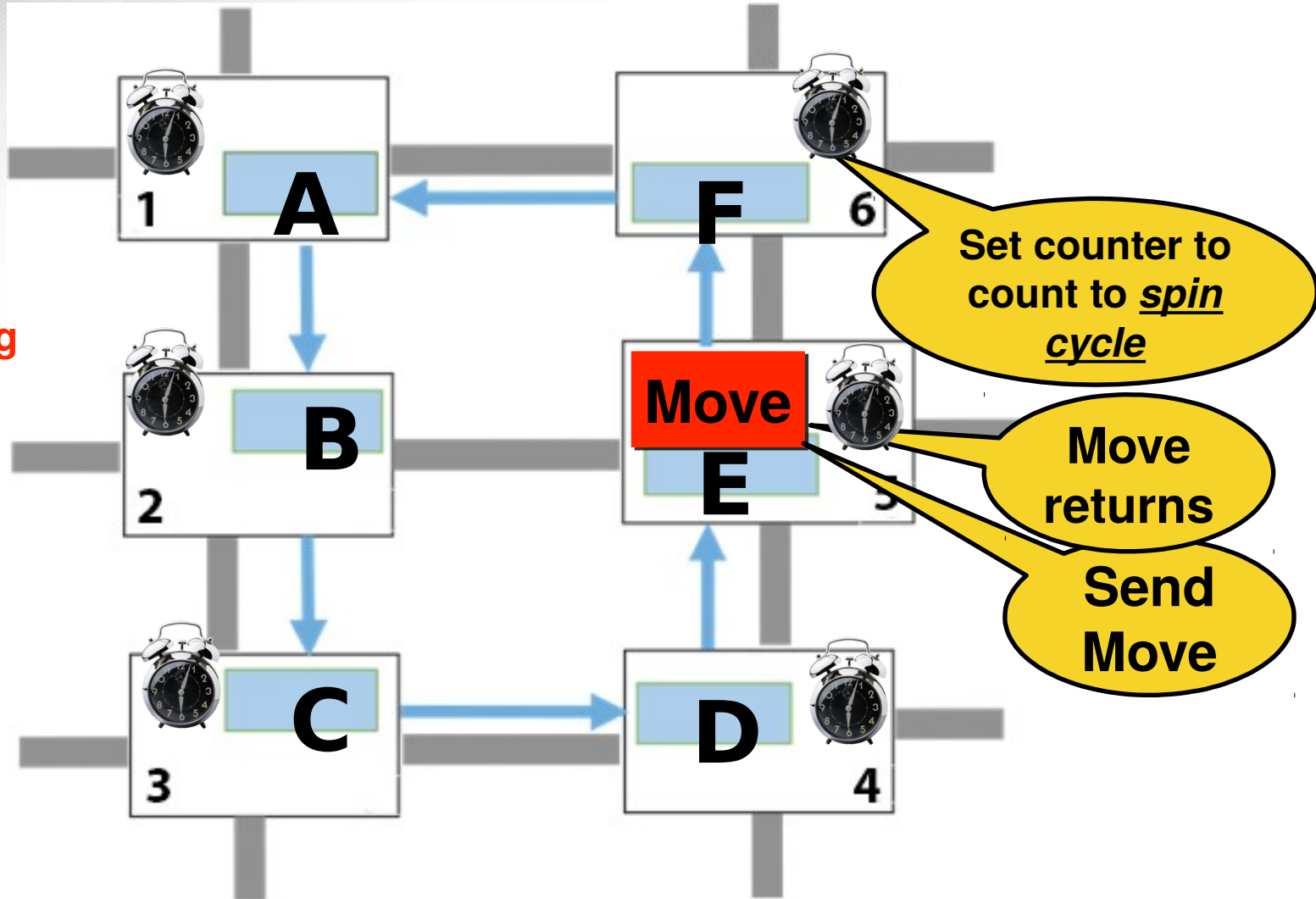
- ***Probe*** is a special message that ***tracks*** the ***buffer dependency***.
  - Uses “blocked” links, no extra network required
- ***Probe returns*** to sender:
  - Cyclic buffer dependence, hence ***deadlock***.
- Next, send a ***move*** msg. to convey the ***spin time***
  - Upon receiving move msg., router sets its ***counter*** to count to ***spin cyle***.

# Example : *Move Msg.*

1. Deadlock  
Detection

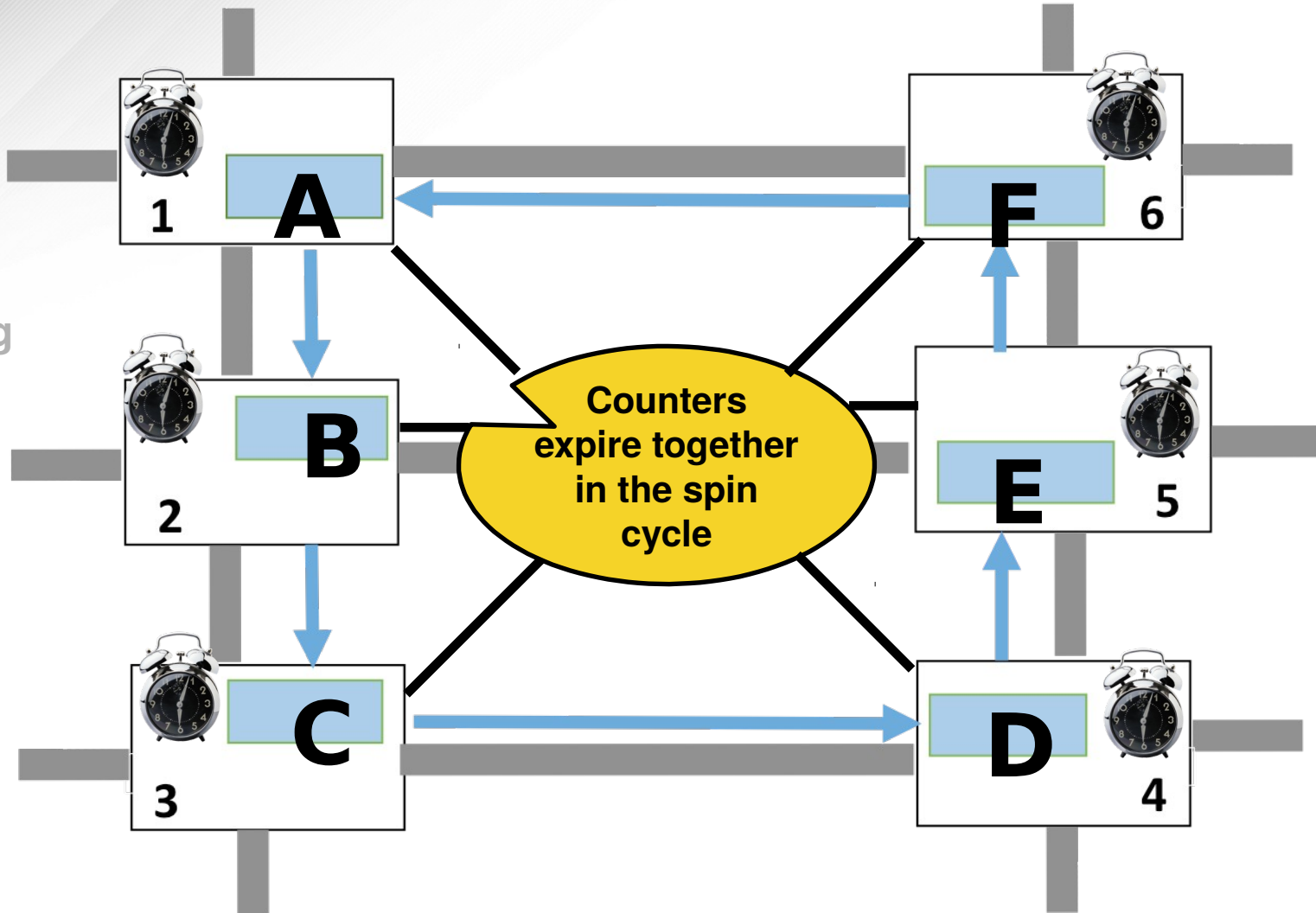
2. Coordinating  
the spin.

3. Executing  
the spin.



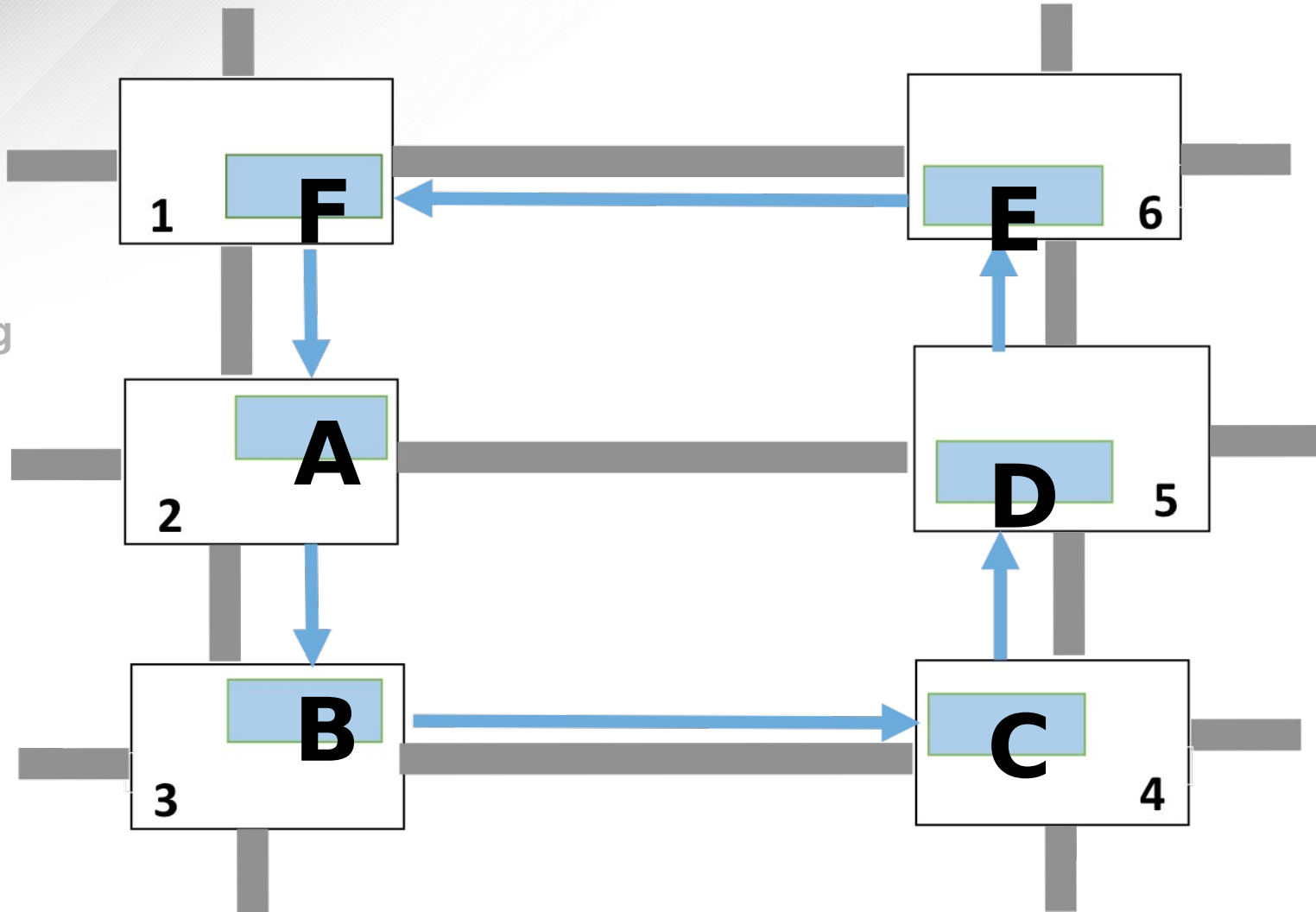
# Example : *spin*

1. Deadlock Detection
2. Coordinating the spin.
3. **Executing the spin.**



# Example : *spin*

1. Deadlock Detection
2. Coordinating the spin.
3. **Executing the spin.**



# SPIN Optimization

- Resolving a deadlock may require *multiple spins*
  - After spin, router can resume normal operation.
  - Counter expires again, process repeated.
- *Optimization:* send *probe\_move* after spin is complete.
  - probe\_move *checks* if *deadlock still exists* and if so, sets the time for the next spin.
- Details in paper (Sec. IV-B).

# Microarchitecture

- ***No additional links:*** Spl. Msgs. use the same links as regular flits.
  - Spl. Msgs. have higher priority in link usage over regular flits.
  - Links are idle during deadlocks (by definition).
- ***Bufferless Forwarding:*** Spl. Msgs. are not buffered anywhere (either forwarded or dropped).
- ***Distributed Design:*** any router can initiate the recovery.
- ***4% area overhead*** compared to traditional mesh router in 15nm.

# FAvORS Routing

- *SPIN* is the *first scheme* that enables *true one-VC* fully adaptive deadlock-free routing for *any topology*.
- **FAvORS** : **F**ully **A**daptive **O**ne-vc **R**outing with **SPIN**.
  - Algorithm has two flavors:
    - Minimal Adaptive
    - Non-minimal Adaptive
  - Route Selection Metrics:
    - Credit turn-around time
    - Hop Count
  - More details in paper (Sec. V).



## Network Configuration

<b>Simulator</b>	gem5 simulator + Garnet 2.0 Network model	
<b>Topologies</b>	8x8 Mesh	1024 node Off-chip Dragon-fly
<b>Link Latency</b>	1-cycle	Inter-group: 3-cycle Intra-group: 1-cycle
<b>Traffic</b>	Synthetic + Multi-threaded (PARSEC)	Synthetic

# Baselines

## 8x8 Mesh

Design	Routing Adaptivity	Minimal	Theory	Deadlock Freedom Type
<b>West-first Routing</b>	Partial	Yes	Dally	Avoidance
<b>Escape-VC</b>	Full	Yes	Duato	Avoidance
<b>Static-Bubble [6]</b>	Full	Yes	Flow-Control	Recovery

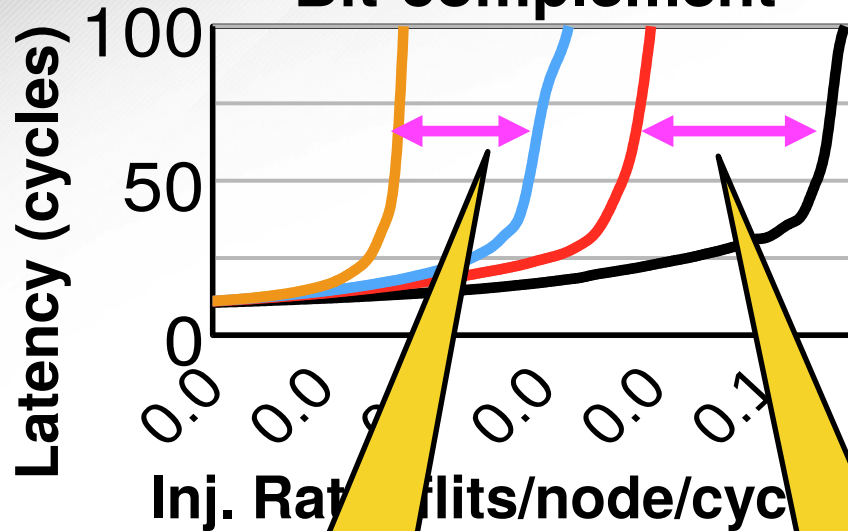
## 1024 Node Off-chip Dragon-fly

Design	Routing Adaptivity	Minimal	Theory	Deadlock Freedom Type
<b>UGAL [37]</b>	Full	No	Dally	Avoidance

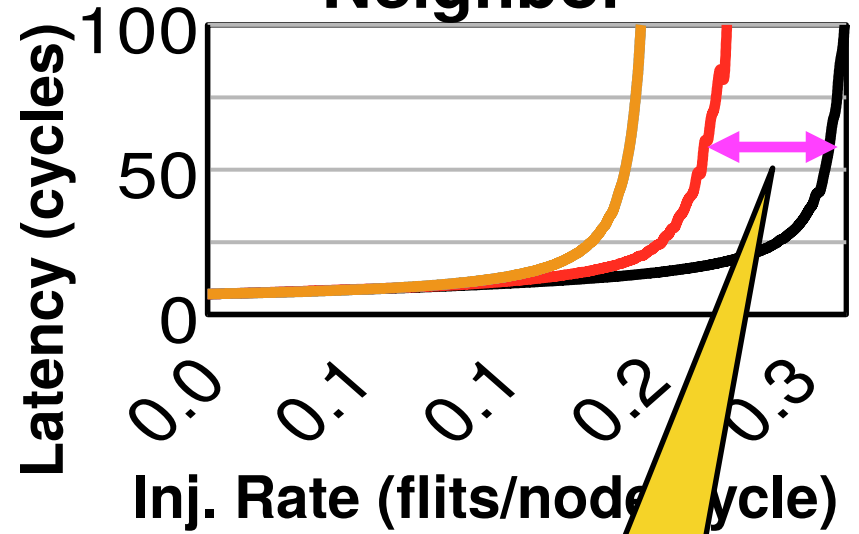
# Throughput

## 1024-node Off-chip Dragon-fly

### Bit-complement



### Neighbor



UGAL\_3V SPIN    
  UGAL\_3VC Dally    
  FAVORS\_NMin\_1VC SPIN    
  Minimal\_1VC SPIN

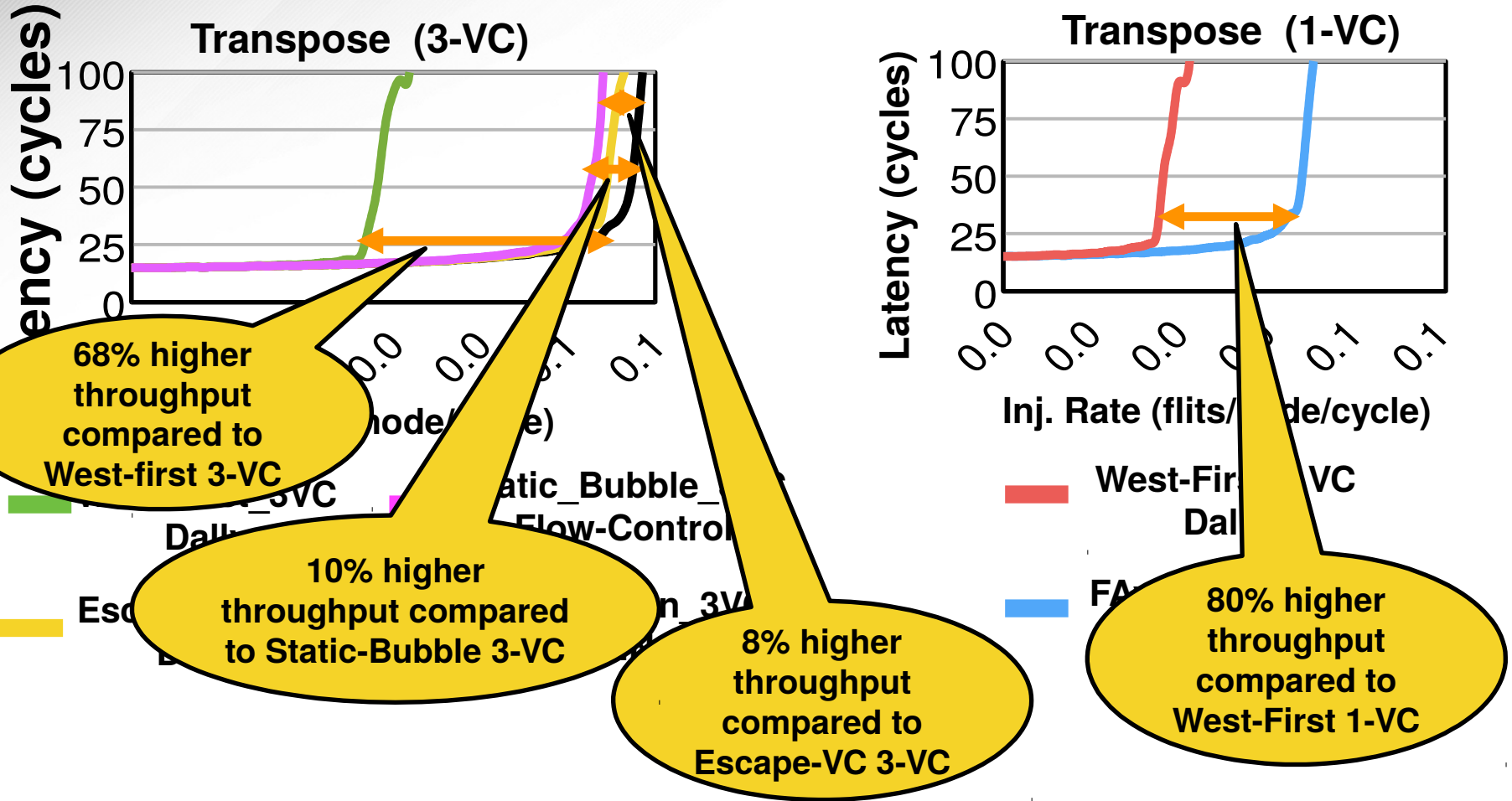
62% higher throughput compared to Minimal Routing 1-VC

50% higher throughput compared to UGAL\_Dally

25% higher throughput compared to UGAL\_Dally

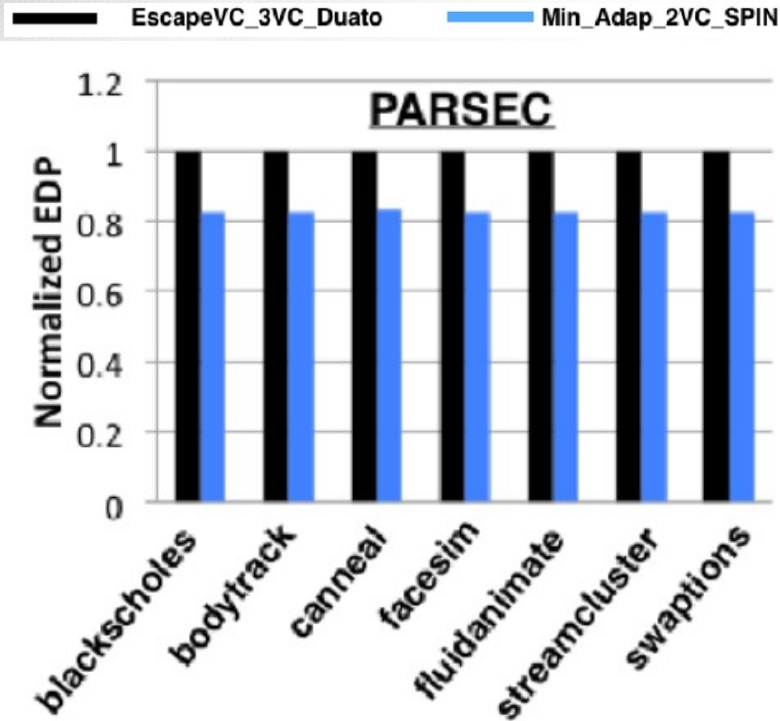
# Throughput

## 8x8 On-chip Mesh



# Energy Delay

3VC\_Duato vs.  
2VC\_SPIN Adaptive



- Runtime effectively equivalent
- **2VC\_SPIN 18% less energy and EDP**

# Summary

- **Deadlocks** are a **fundamental problem** in Interconnection Networks.
  - **SPIN** is a new deadlock freedom theory
  - **Simultaneous packet movement** for deadlock recovery
  - No routing restrictions or escape-VCs required
  - Enables **true one-VC fully adaptive routing** for any topology
- Salient Features of our Implementation:
  - **Scalable:** Distributed Deadlock Resolution
  - **Plug-n-Play:** topology agnostic
  - **68% higher** (Mesh) & **62% higher** (dragon-fly) saturation throughput.
- **Can we do better?**



# Outline

- Introduction
- Background: Routing Deadlocks
- *SPIN* : *S*ynchronized *P*rogress in *I*nterconnection *N*etworks
- *DRAIN* : *D*eadlock *R*emoval for *A*rbitrary *I*rregular *N*etworks
- Conclusion

- **SPIN** drawbacks
  - Somewhat complicated deadlock detection hardware
  - Built around Static Bubble<sub>[Ramrakhiani and Krishna,HPCA'17]</sub> framework
    - Difficult to adapt to wear-out induced network topology changes
- **DRAIN** : **D**eadlock **R**emoval for **A**rbitrary **I**rregular **N**etworks
  - No need for deadlock detection
  - Adapt to changing network topology



# DRAIN: Key Idea

- ***DRAIN*** : *D*eadlock *R*emoval for *A*rbitrary *I*rregular *N*etworks
  - **No need for deadlock detection**
    - Pre-emptively SPIN (***DRAIN***) the whole network
      - Deadlocks are rare so spin every >100K cycles
    - Deadlocks will be broken even if packets are mis-routed
  - **Adapt to changing network topology**
    - Link/router breaks change network topology to be spun
    - Developed algorithm to find ***minimum set of DRAIN paths*** which cover whole network
    - ***Regen DRAIN paths*** when link/router breaks
    - Route adaptively w/o worry of network deadlock

# Conclusion

- ***Deadlocks*** are a ***fundamental problem*** in Interconnection Networks.
  - ***SPIN*** and ***DRAIN*** represent a new approach to deadlock freedom
    - ***Deadlocks*** – not a lack of resources, a lack of coordination
    - ***Simultaneous packet movement*** for deadlock recovery
  - ***Low Overheads:*** No routing restrictions or escape-VCs required
  - ***High Performance:*** Equal or better performance with less hardware cost

# Acknowledgments

- ***SPIN*** Collaborators:
  - Aniruddh Ramrakhyani – Apple/Georgia Tech
  - Tushar Krishna – Georgia Tech
- ***DRAIN*** Collaborators:
  - Mayank Parasar – Georgia Tech
  - Joshua San Miguel – University of Wisconsin
  - Tushar Krishna – Georgia Tech
  - Natalie Enright-Jerger – University of Toronto