

# Computer Architecture

## Lecture 22: Interconnects II

Prof. Onur Mutlu

ETH Zürich

Fall 2017

20 December 2017

# Summary of Last Lecture

---

- Interconnection Network Basics

# Today and Tomorrow

---

- Interconnection Networks Wrap-Up
- Research in Computer Architecture
- Course Logistics
  - Final Lab
  - Final Exam
  - Past Exams and Homeworks
- Discussion session tomorrow
  - Exam Questions
  - Bring Questions

# Interconnection Networks



# Readings

---

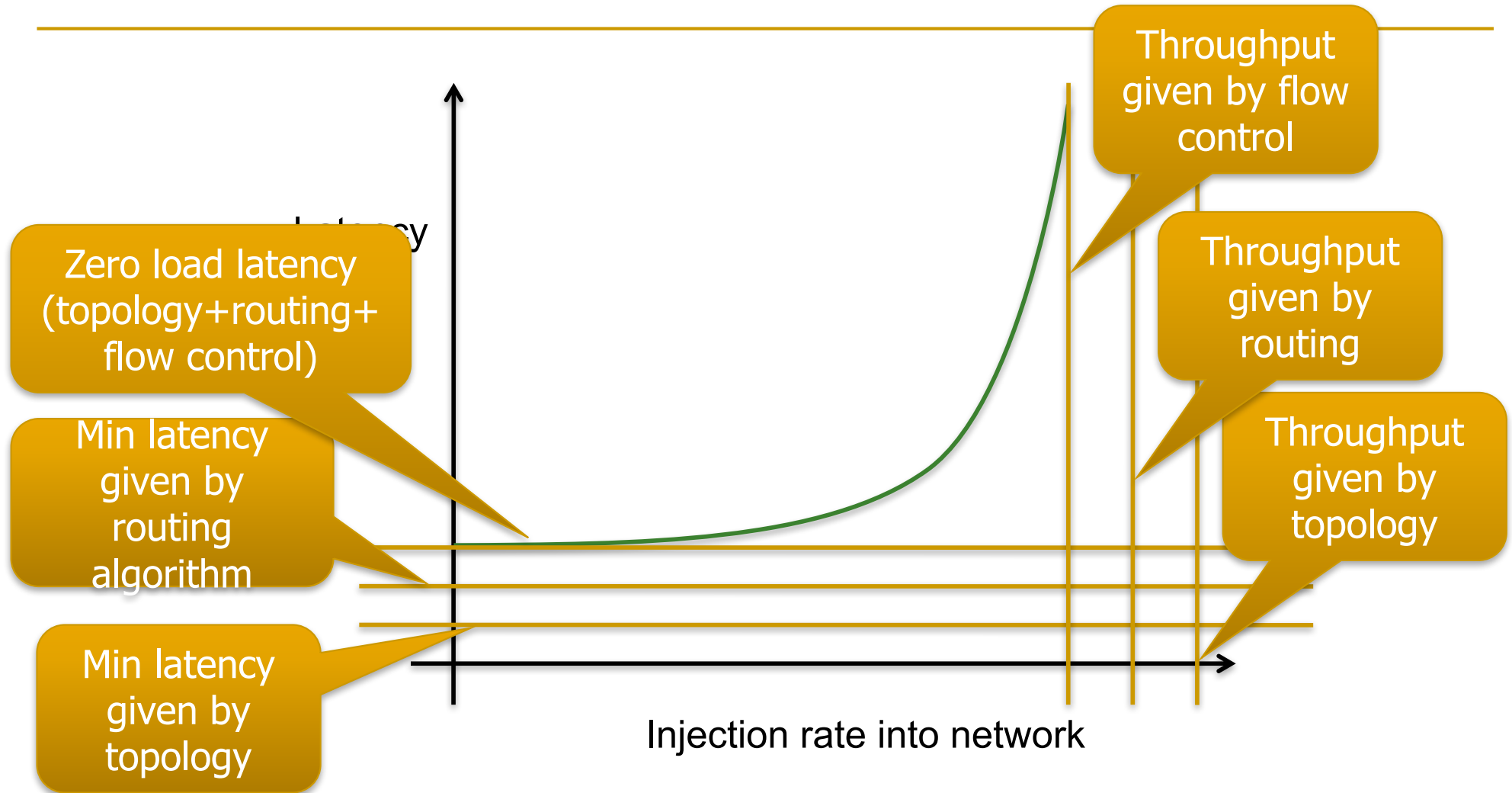
## ■ Required

- Moscibroda and Mutlu, “[A Case for Bufferless Routing in On-Chip Networks](#),” ISCA 2009.

## ■ Recommended

- Das et al., “[Application-Aware Prioritization Mechanisms for On-Chip Networks](#),” MICRO 2009.

# Review: Interconnection Network Performance



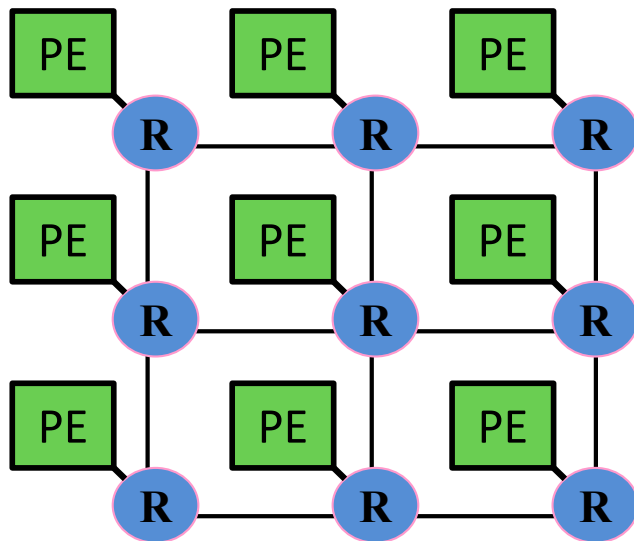
# Review: Network Performance Metrics

---

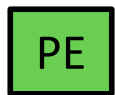
- Packet latency
- Round trip latency
- Saturation throughput
- Application-level performance: system performance
  - Affected by interference among threads/applications

# Buffering and Routing in On-Chip Networks

# On-Chip Networks



Router

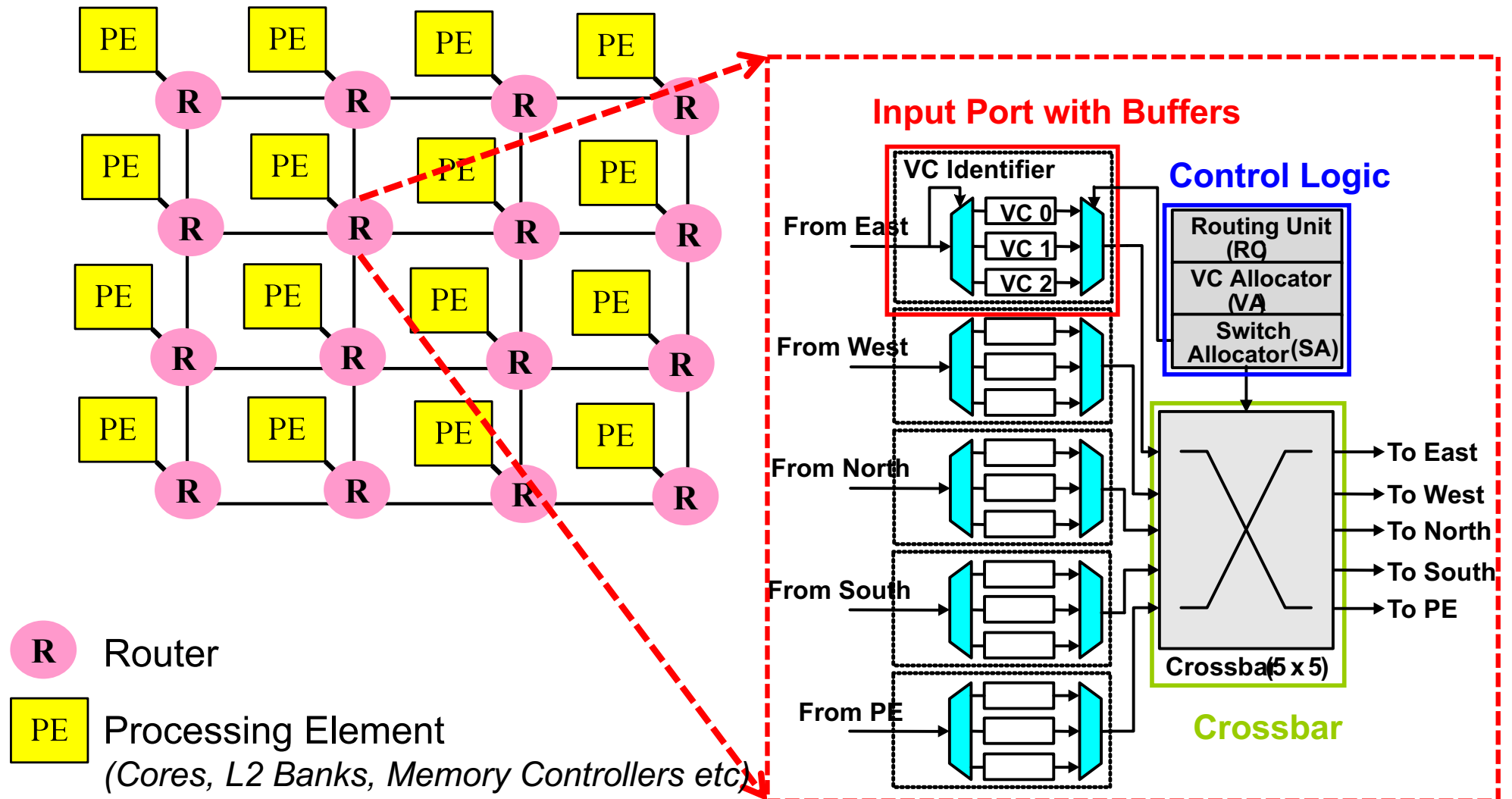


Processing Element

(Cores, L2 Banks, Memory Controllers, etc)

- Connect **cores, caches, memory controllers, etc**
  - Buses and crossbars are not scalable
- **Packet switched**
- **2D mesh:** Most commonly used topology
- Primarily serve **cache misses** and **memory requests**

# On-chip Networks



# On-Chip vs. Off-Chip Interconnects

---

## ■ On-chip advantages

- ❑ Low latency between cores
- ❑ No pin constraints
- ❑ Rich wiring resources
- Very high bandwidth
- Simpler coordination

## ■ On-chip constraints/disadvantages

- ❑ 2D substrate limits implementable topologies
- ❑ Energy/power consumption a key concern
- ❑ Complex algorithms undesirable
- ❑ Logic area constrains use of wiring resources

# On-Chip vs. Off-Chip Interconnects (II)

---

- Cost
  - Off-chip: Channels, pins, connectors, cables
  - On-chip: Cost is storage and switches (wires are plentiful)
  - Leads to networks with many wide channels, few buffers
  
- Channel characteristics
  - On chip short distance → low latency
  - On chip RC lines → need repeaters every 1-2mm
    - Can put logic in repeaters
  
- Workloads
  - Multi-core cache traffic vs. supercomputer interconnect traffic



# On-Chip vs. Off-Chip Tradeoffs

---

- George Nychis, Chris Fallin, Thomas Moscibroda, Onur Mutlu, and Srinivasan Seshan,  
**"On-Chip Networks from a Networking Perspective: Congestion and Scalability in Many-core Interconnects"**  
*Proceedings of the 2012 ACM SIGCOMM Conference (**SIGCOMM**), Helsinki, Finland, August 2012. Slides (pptx)*

## On-Chip Networks from a Networking Perspective: Congestion and Scalability in Many-Core Interconnects

George Nychis<sup>†</sup>, Chris Fallin<sup>†</sup>, Thomas Moscibroda<sup>§</sup>, Onur Mutlu<sup>†</sup>, Srinivasan Seshan<sup>†</sup>

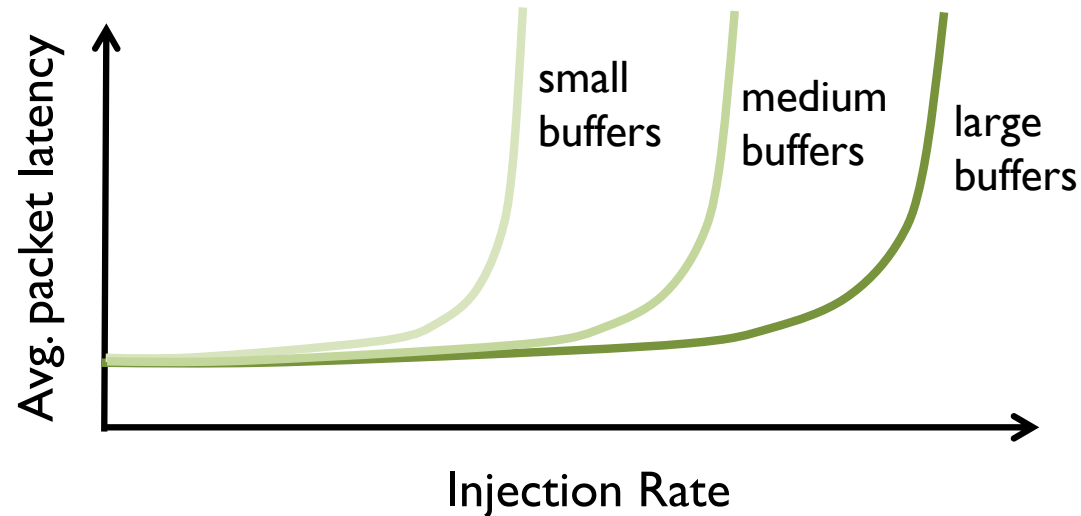
<sup>†</sup> Carnegie Mellon University  
{gnychis,cfallin,onur,srini}@cmu.edu

<sup>§</sup> Microsoft Research Asia  
moscitho@microsoft.com

# Buffers in NoC Routers

---

- Buffers are necessary for high network throughput  
→ buffers increase total available bandwidth in network



# Buffers in NoC Routers

---

- Buffers are necessary for high network throughput  
→ buffers increase total available bandwidth

- Buffers consume significant chip area
  - Dynamic energy
  - Static energy
- Buffers require significant **chip area**

e.g., in TRIPS prototype chip, input buffers occupy 75% of total on-chip network area [Gratz et al, ICCD' 06]

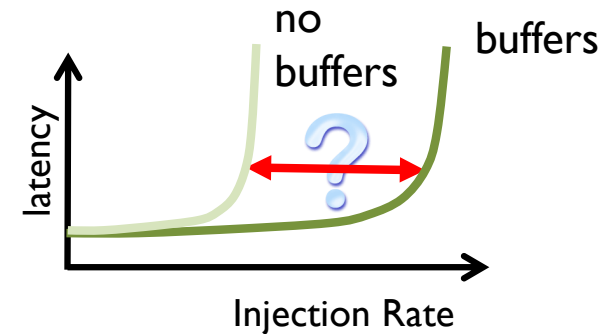
Can we get rid of buffers...?



# Going Bufferless...?

- How much throughput do we lose?

→ How is latency affected?



- Up to what **injection rates** can we use bufferless routing?

→ Are there **realistic scenarios** in which NoC is operated at injection rates below the threshold?

- Can we achieve **energy reduction**?

→ If so, how much...?

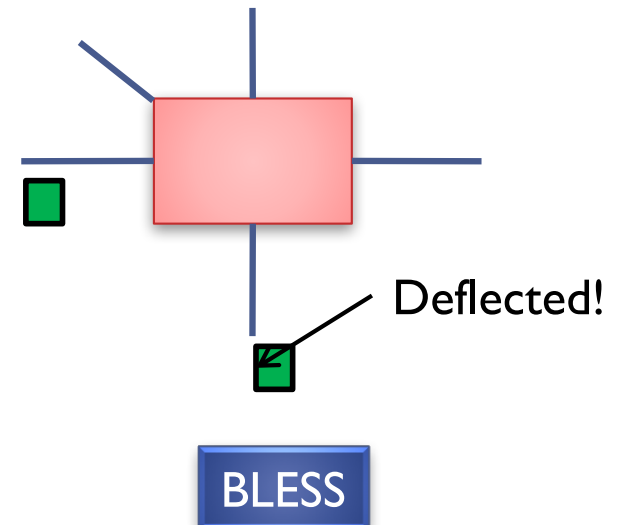
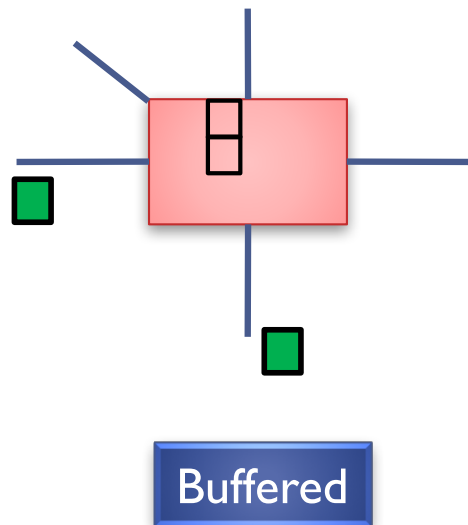
- Can we reduce **area, complexity**, etc...?



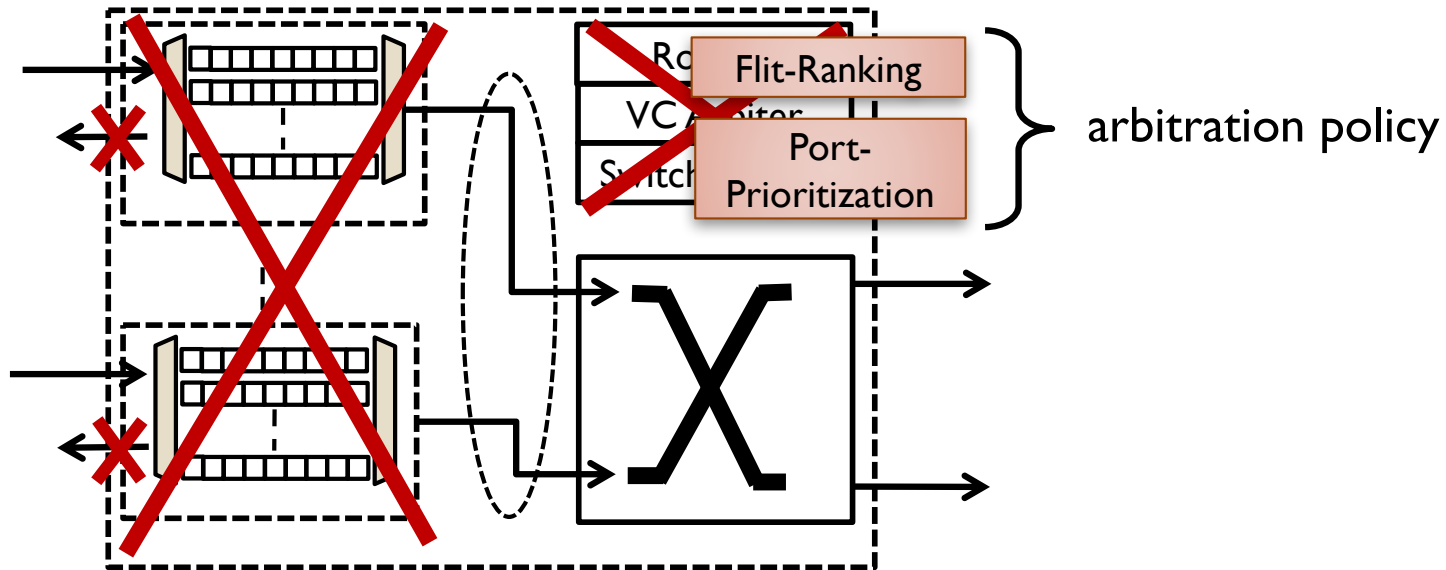
Answers in  
our paper!

# BLESS: Bufferless Routing

- Always forward *all* incoming flits to some output port
- If no productive direction is available, send to another direction
- → packet is deflected
- → **Hot-potato routing** [Baran' 64, etc]



# BLESS: Bufferless Routing



Flit-Ranking

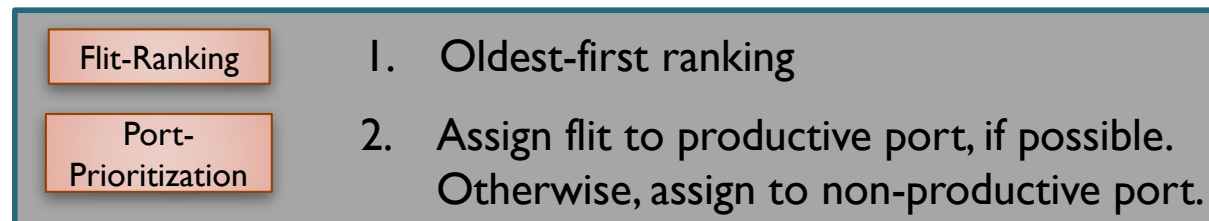
1. Create a ranking over all incoming flits

Port-Prioritization

2. For a given flit in this ranking, find the best free output-port  
Apply to each flit in order of ranking

# FLIT-BLESS: Flit-Level Routing

- Each flit is routed independently.
- **Oldest-first arbitration** (other policies evaluated in paper)



- **Network Topology:**
  - Can be applied to most topologies (Mesh, Torus, Hypercube, Trees, ...)
    - 1) #output ports , #input ports at every router
    - 2) every router is reachable from every other router
- **Flow Control & Injection Policy:**
  - Completely **local**, inject whenever input port is free
- **Absence of Deadlocks:** every flit is always moving
- **Absence of Livelocks:** with oldest-first ranking



# BLESS: Advantages & Disadvantages

---

## Advantages

- No buffers
- Purely local flow control
- Simplicity
  - no credit-flows
  - no virtual channels
  - simplified router design
- No deadlocks, livelocks
- Adaptivity
  - packets are deflected around congested areas!
- Router latency reduction
- Area savings

## Disadvantages

- Increased latency
- Reduced bandwidth
- Increased buffering at receiver
- Header information at each flit
- Oldest-first arbitration complex
- QoS becomes difficult

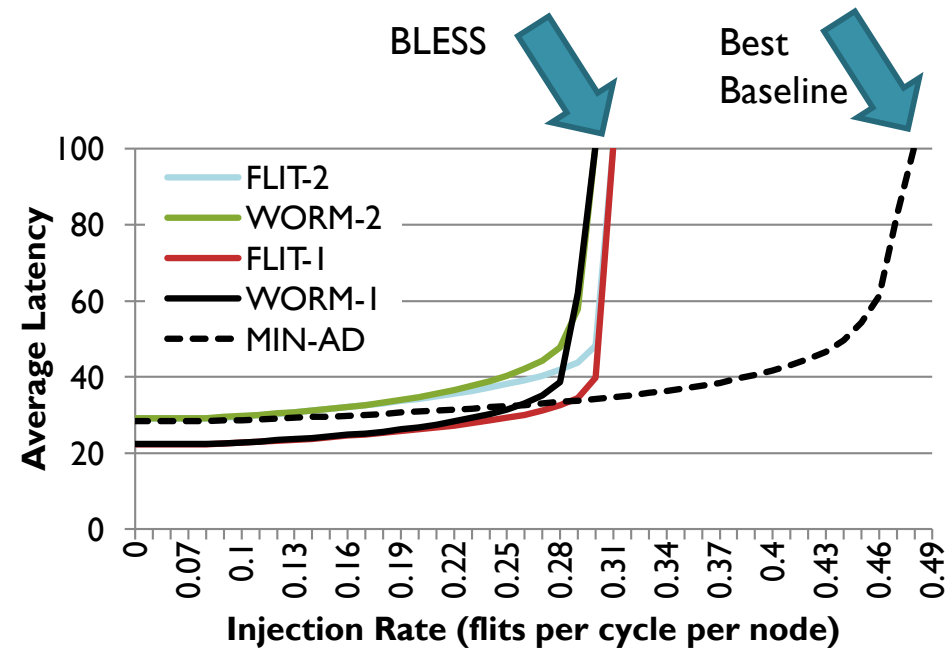
Impact on energy...?






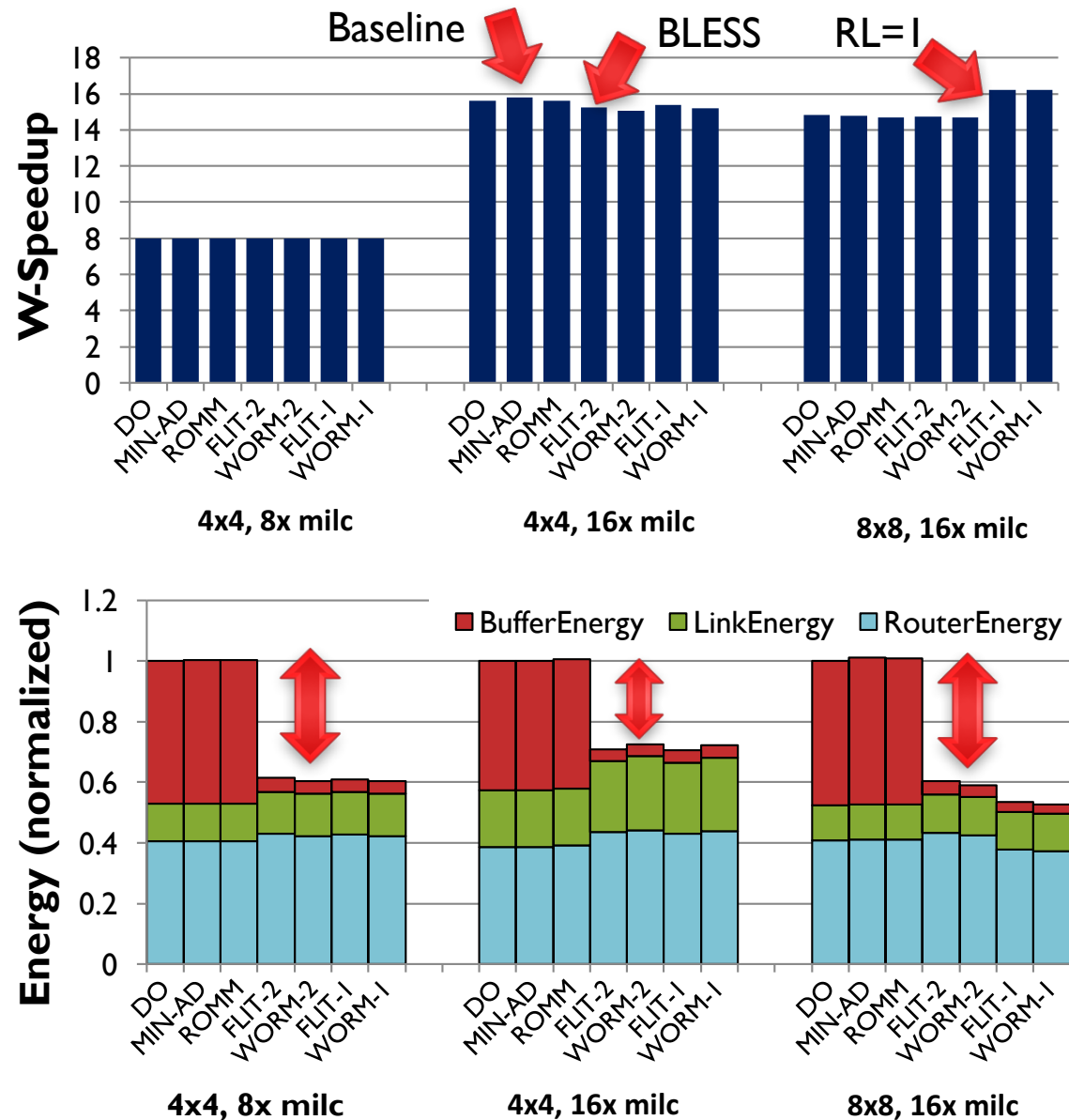
# Evaluation – Synthetic Traces

- First, the bad news ☹️
- Uniform random injection
- BLESS has **significantly lower saturation throughput** compared to buffered baseline.



# Evaluation – Homogenous Case Study

- **milc** benchmarks  
(moderately intensive)
- **Perfect caches!**
- Very little performance degradation with BLESS  
(less than 4% in dense network)
- With router latency 1, BLESS can even outperform baseline  
(by ~10%)
- **Significant energy improvements**  
(almost 40%) 



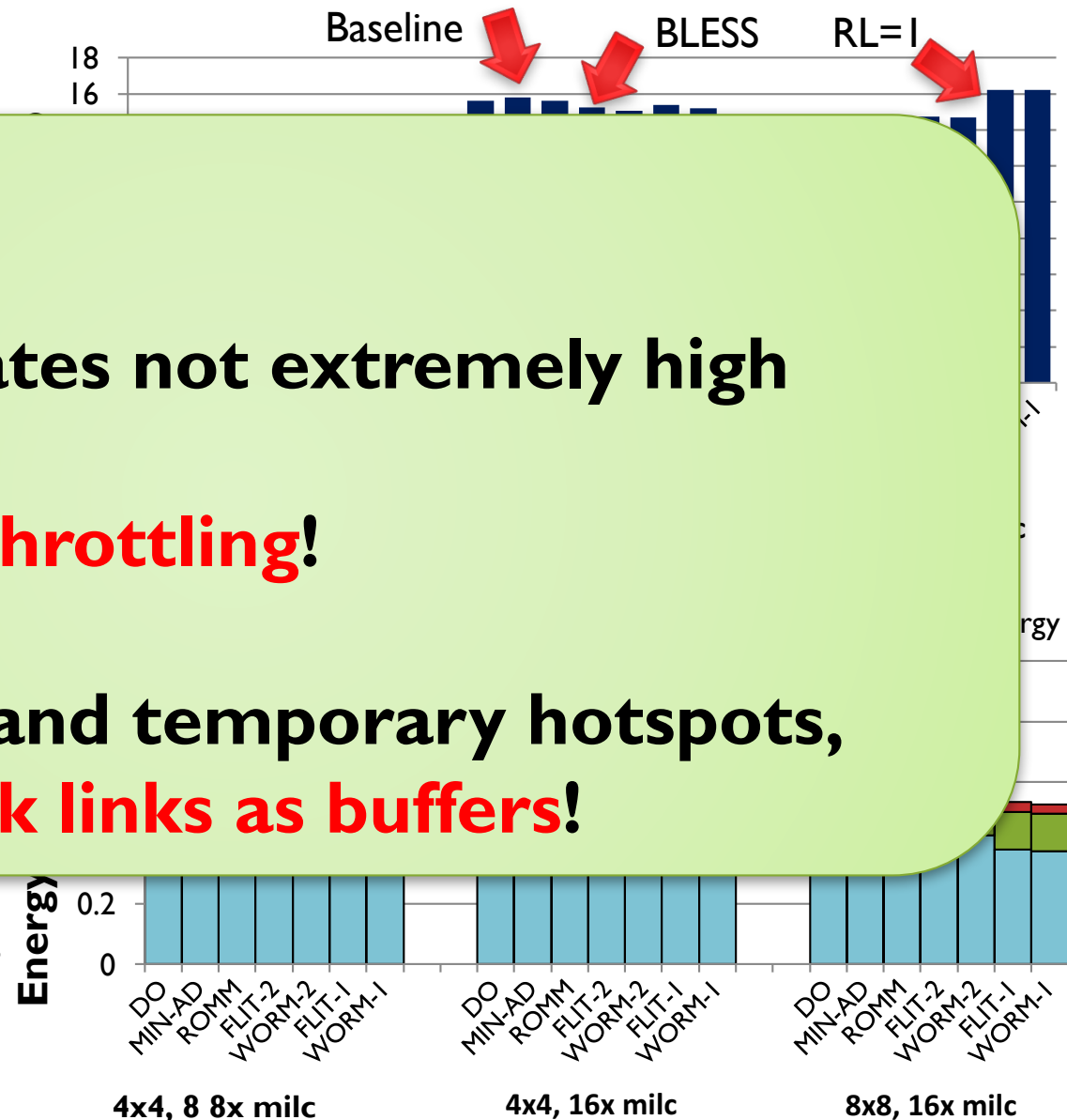
# Evaluation – Homogenous Case Study

- **mile benchmarks**

## Observations:

- 1) Injection rates not extremely high on average  
→ **self-throttling!**
- 2) For bursts and temporary hotspots,  
**use network links as buffers!**

- **Significant energy improvements (almost 40%)**



# BLESS Conclusions

---

- For a very wide range of applications and network settings, buffers are not needed in NoC
  - Significant energy savings (32% even in dense networks and perfect caches)
  - Area-savings of 60%
  - Simplified router and network design (flow control, etc...)
  - Performance slowdown is minimal (can even increase!)

➤ A strong case for a **rethinking of NoC design!**

- **Future research:**
  - Support for quality of service, different traffic classes, energy-management, etc...

# Bufferless Routing in NoCs

---

- Moscibroda and Mutlu, “A Case for Bufferless Routing in On-Chip Networks,” ISCA 2009.
  - [https://users.ece.cmu.edu/~omutlu/pub/bless\\_isca09.pdf](https://users.ece.cmu.edu/~omutlu/pub/bless_isca09.pdf)

## A Case for Bufferless Routing in On-Chip Networks

Thomas Moscibroda  
Microsoft Research  
moscitho@microsoft.com

Onur Mutlu  
Carnegie Mellon University  
onur@cmu.edu

# Issues In Bufferless Deflection Routing

---

- Livelock
- Resulting Router Complexity
- Performance & Congestion at High Loads
- Quality of Service and Fairness
- Chris Fallin, Greg Nazario, Xiangyao Yu, Kevin Chang, Rachata Ausavarungnirun, and Onur Mutlu,  
**"Bufferless and Minimally-Buffered Deflection Routing"**  
*Invited Book Chapter in Routing Algorithms in Networks-on-Chip, pp. 241-275, Springer, 2014.*

# Low-Complexity Bufferless Routing

---

- Chris Fallin, Chris Craik, and Onur Mutlu,  
**"CHIPPER: A Low-Complexity Bufferless Deflection Router"**  
*Proceedings of the 17th International Symposium on High-Performance Computer Architecture (HPCA)*, pages 144-155, San Antonio, TX, February 2011. Slides (pptx)  
An extended version as SAFARI Technical Report, TR-SAFARI-2010-001, Carnegie Mellon University, December 2010.

## CHIPPER: A Low-complexity Bufferless Deflection Router

Chris Fallin                      Chris Craik                      Onur Mutlu  
cfallin@cmu.edu    craik@cmu.edu    onur@cmu.edu

Computer Architecture Lab (CALCM)  
Carnegie Mellon University



# CHIPPER: A Low-complexity Bufferless Deflection Router

Chris Fallin, Chris Craik, and Onur Mutlu,

**"CHIPPER: A Low-Complexity Bufferless Deflection Router"**

*Proceedings of the 17th International Symposium on High-Performance Computer Architecture (**HPCA**), pages 144-155, San Antonio, TX, February 2011. Slides (pptx)*

**SAFARI** Carnegie Mellon



# Motivation

---

- Recent work has proposed **bufferless deflection routing** (BLESS [Moscibroda, ISCA 2009])
  - **Energy savings:**  $\sim 40\%$  in total NoC energy
  - **Area reduction:**  $\sim 40\%$  in total NoC area
  - **Minimal performance loss:**  $\sim 4\%$  on average
  - **Unfortunately: unaddressed complexities in router**
    - ➔ long critical path, large reassembly buffers
- **Goal:** obtain these benefits while simplifying the router in order to **make bufferless NoCs practical.**

# Problems that Bufferless Routers Must Solve

---

## 1. Must provide **livelock freedom**

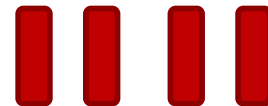
→ A packet should not be deflected forever

## 2. Must **reassemble packets** upon arrival

**Flit:** atomic routing unit

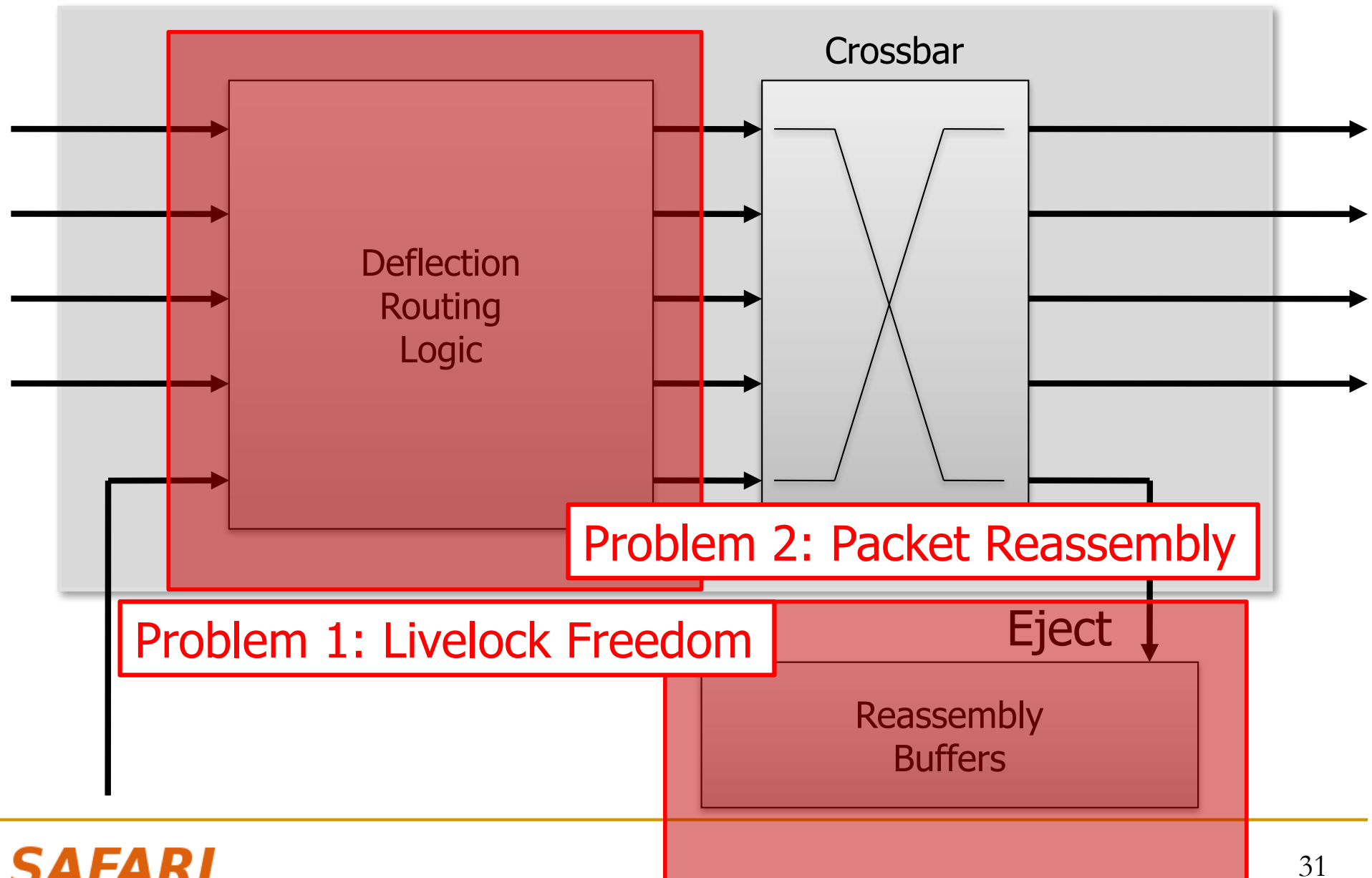


**Packet:** one or multiple flits



0 1 2 3

# A Bufferless Router: A High-Level View



# Complexity in Bufferless Deflection Routers

---

## 1. Must provide livelock freedom

Flits are sorted by age, then assigned in age order to output ports

→ **43% longer critical path than buffered router**

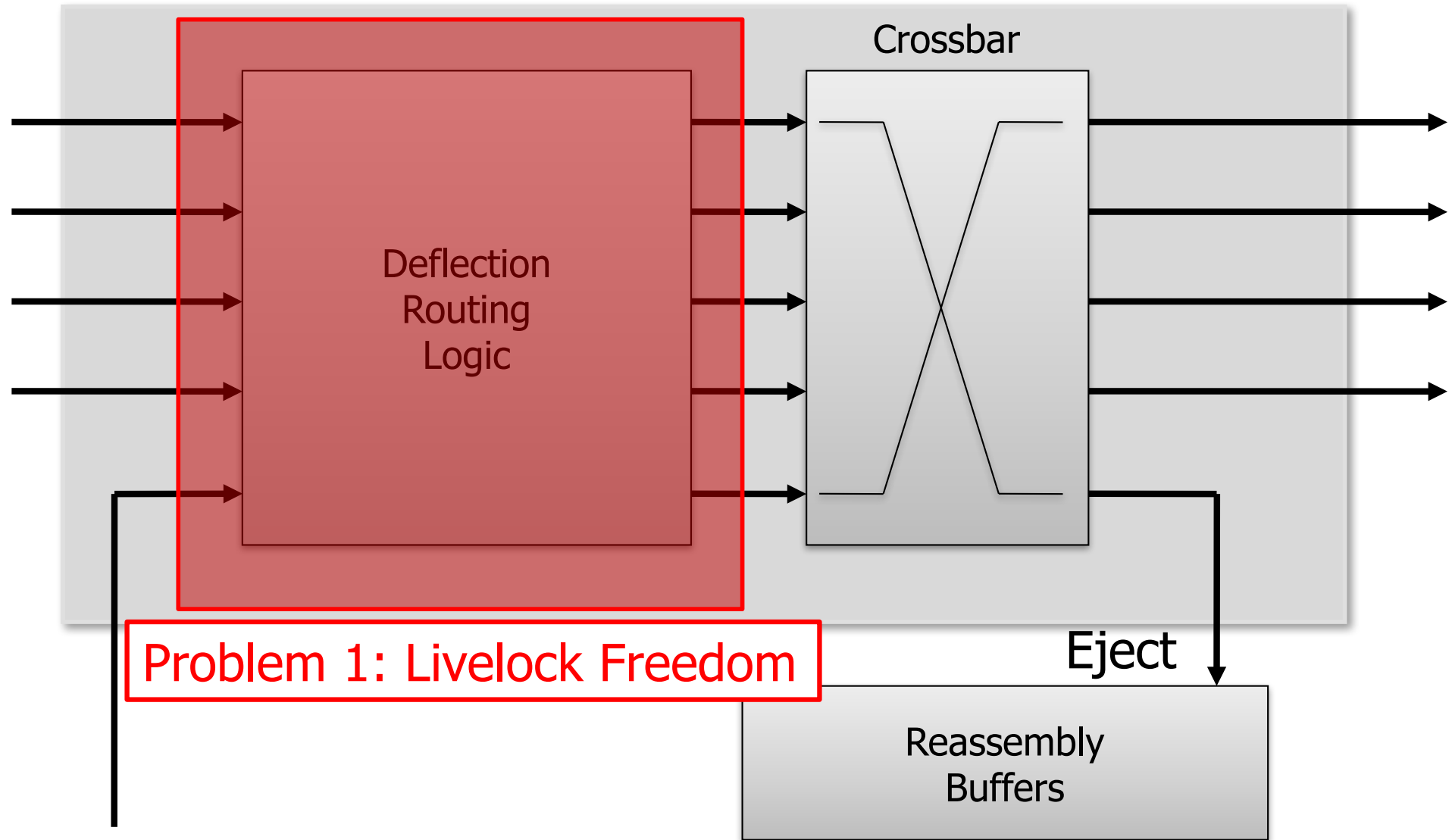
## 2. Must reassemble packets upon arrival

Reassembly buffers must be sized for worst case

→ **4KB per node**

(8x8, 64-byte cache block)

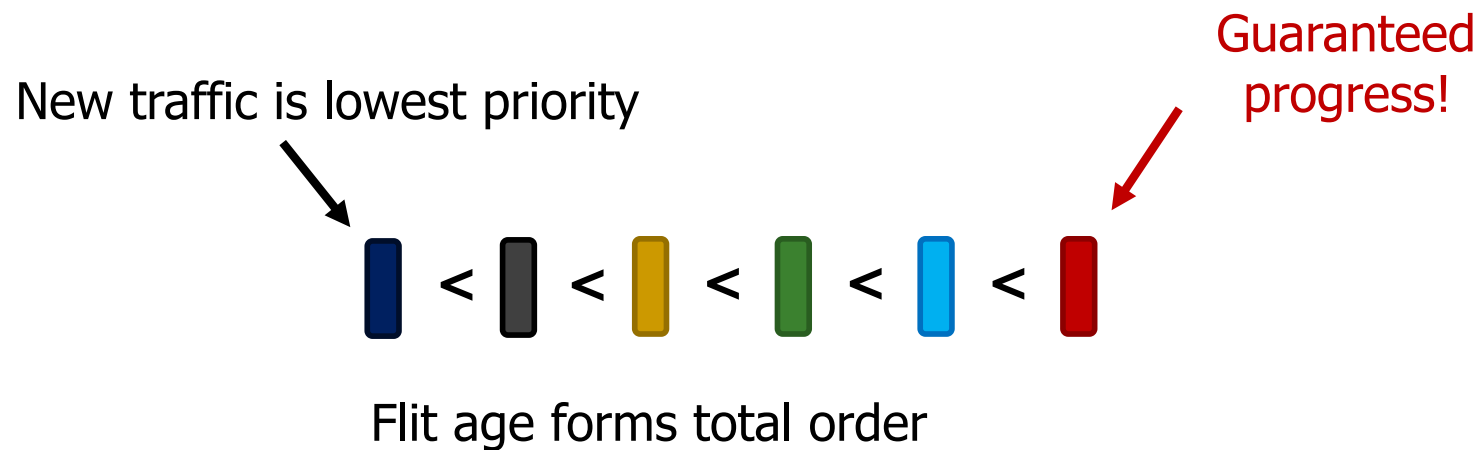
# Problem 1: Livelock Freedom



# Livelock Freedom in Previous Work

---

- What stops a flit from deflecting forever?
- All flits are **timestamped**
- **Oldest flits** are assigned their desired ports
- **Total order among flits**

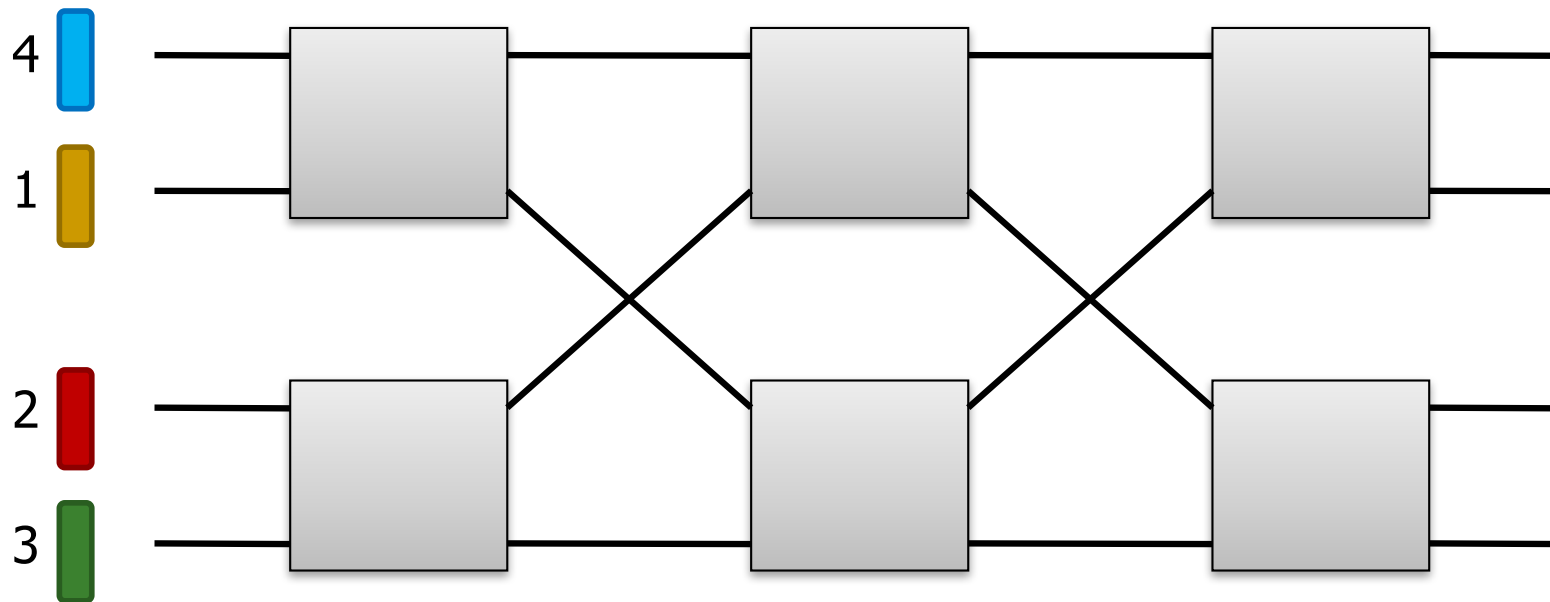


- But what is the **cost** of this?

# Age-Based Priorities are Expensive: Sorting

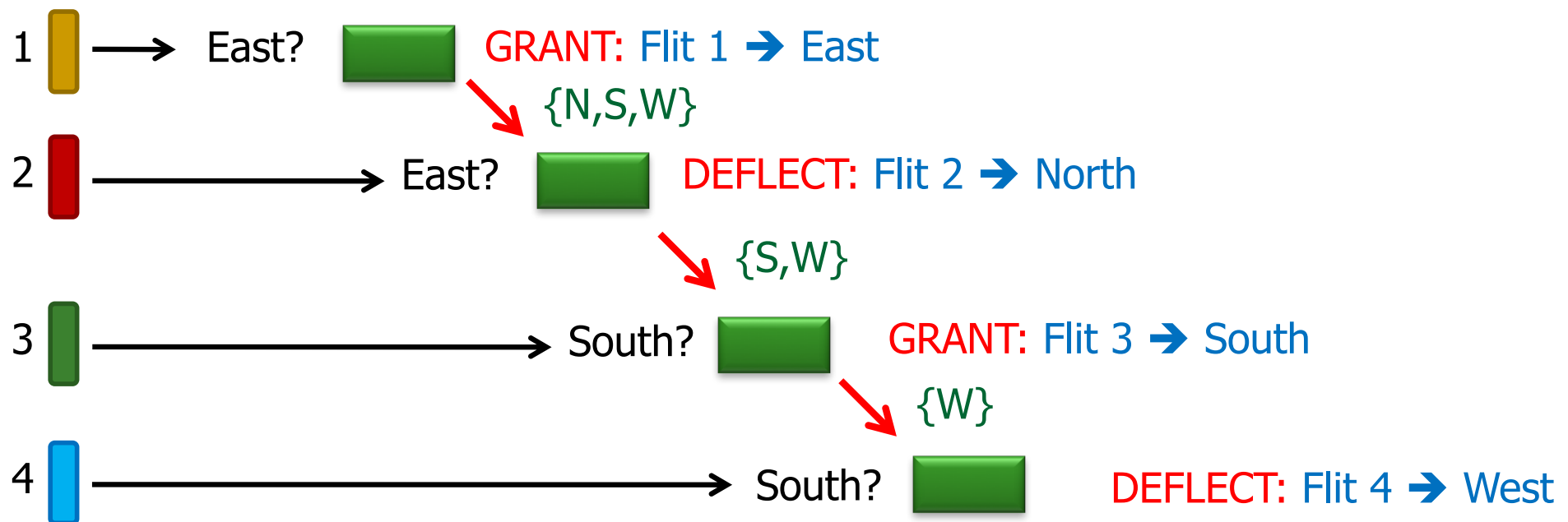
---

- Router must **sort flits by age**: long-latency sort network
  - **Three comparator stages** for 4 flits



# Age-Based Priorities Are Expensive: Allocation

- After sorting, flits assigned to output ports in priority order
- Port assignment of younger flits depends on that of older flits
  - **sequential dependence** in the port allocator

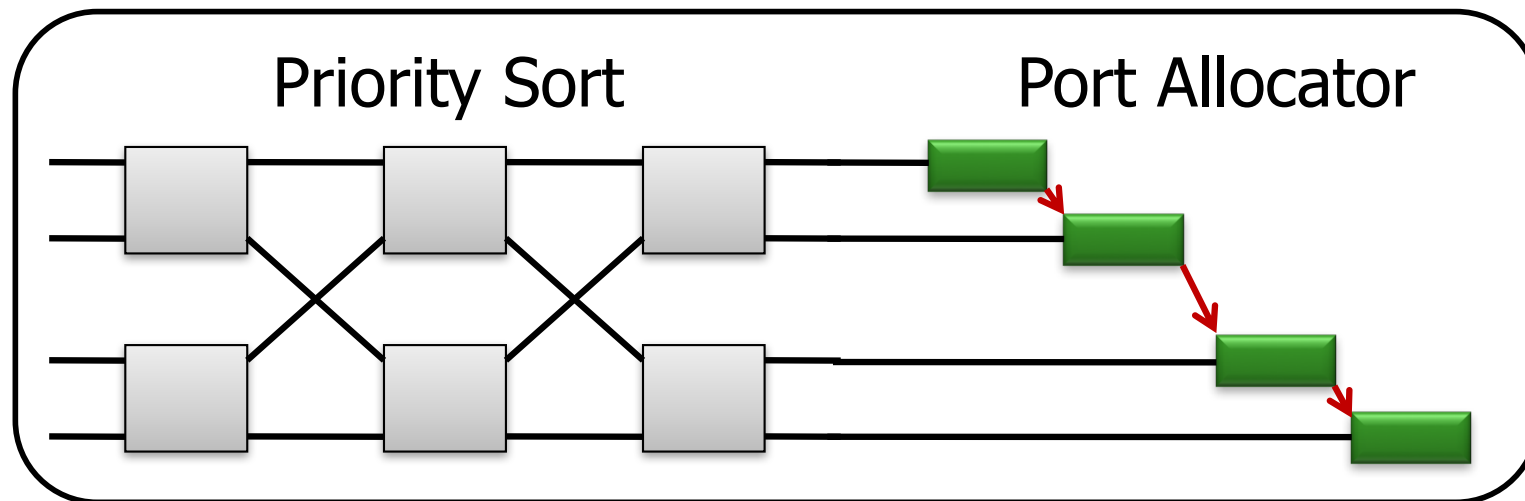


Age-Ordered Flits



# Age-Based Priorities Are Expensive

- Overall, **deflection routing logic** based on **Oldest-First** has a **43% longer critical path** than a buffered router



- Question: is there a cheaper way to route while guaranteeing livelock-freedom?

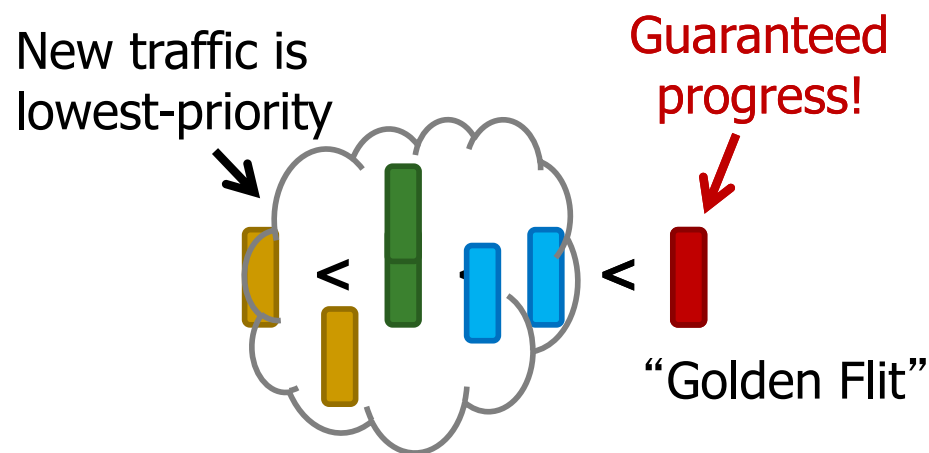
# Solution: Golden Packet for Livelock Freedom

---

- What is *really necessary* for livelock freedom?

**Key Insight:** No total order. it is enough to:

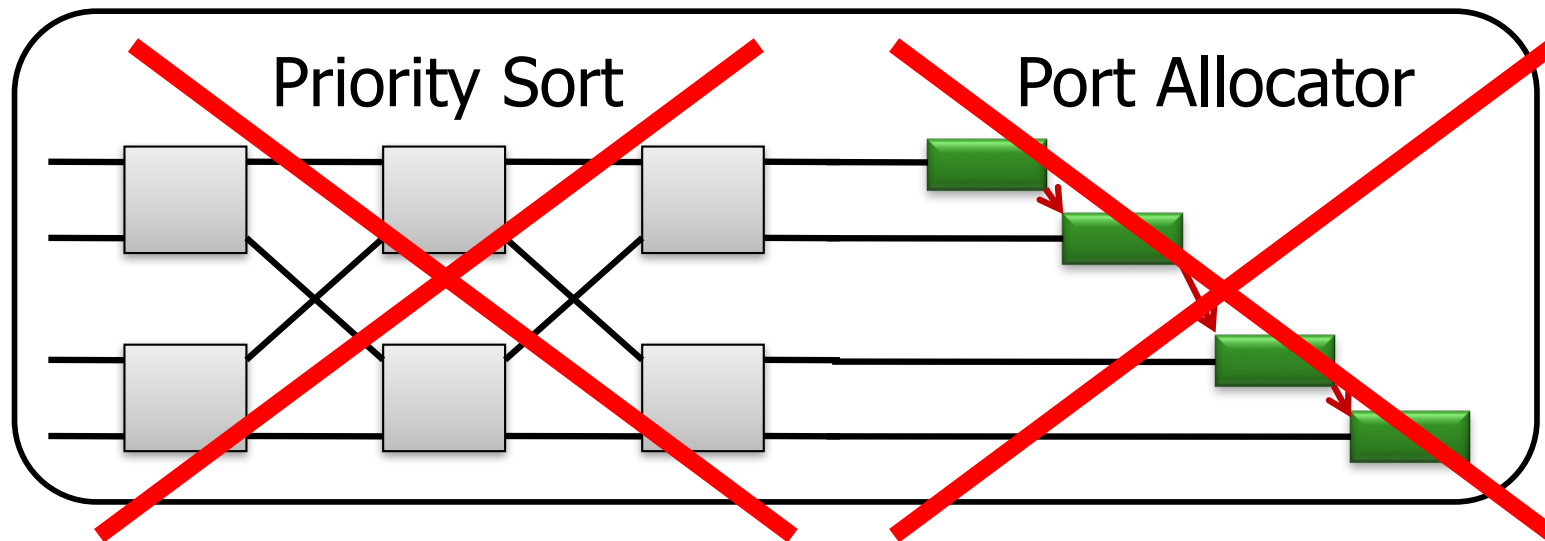
1. Pick one flit to **prioritize until arrival**
2. Ensure any flit is **eventually** picked



Flit age forms total order  
**partial ordering is sufficient!**

# What Does Golden Flit Routing Require?

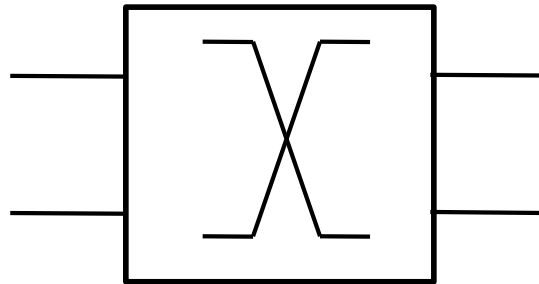
- Only **need** to properly route the Golden Flit
- **First Insight:** no need for full sort
- **Second Insight:** no need for sequential allocation



# Golden Flit Routing With Two Inputs

---

- Let's route the Golden Flit in a two-input router first



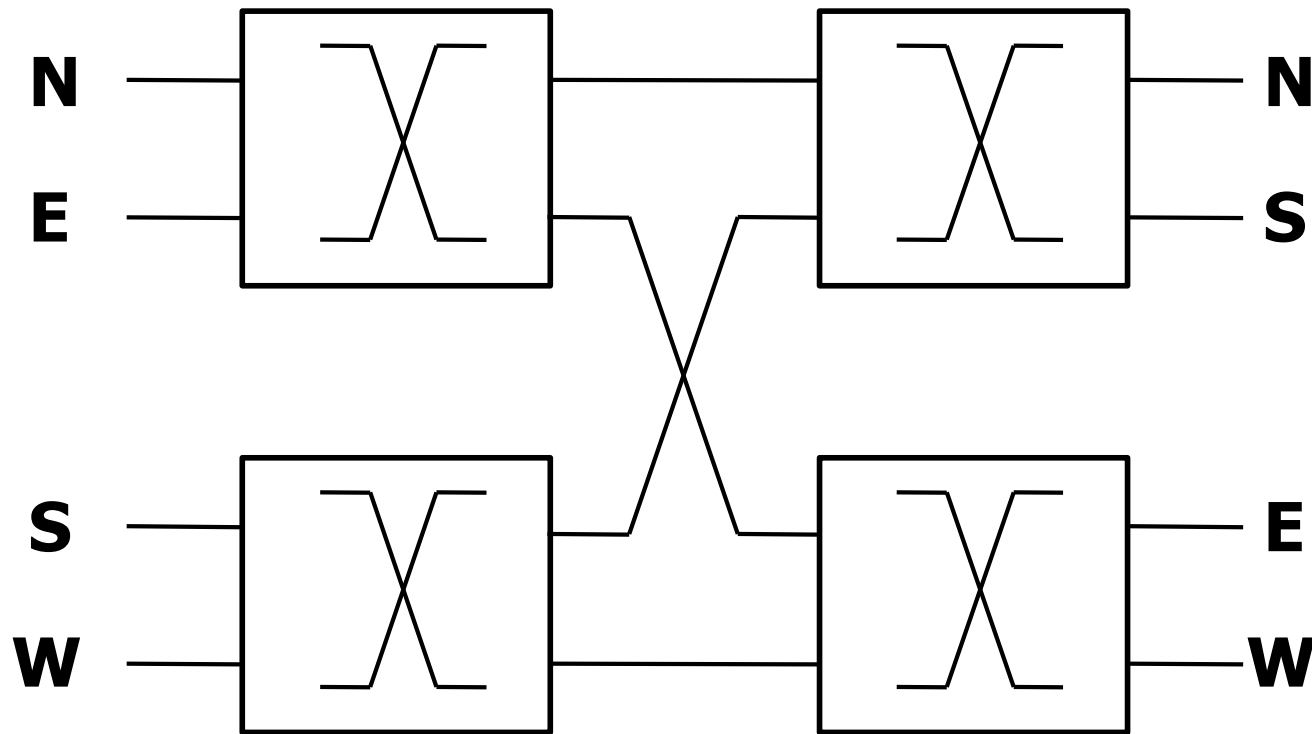
- **Step 1:** pick a “winning” flit: Golden Flit, else random
- **Step 2:** steer the winning flit to its desired output and deflect other flit

➔ **Golden Flit is always routed toward its destination**

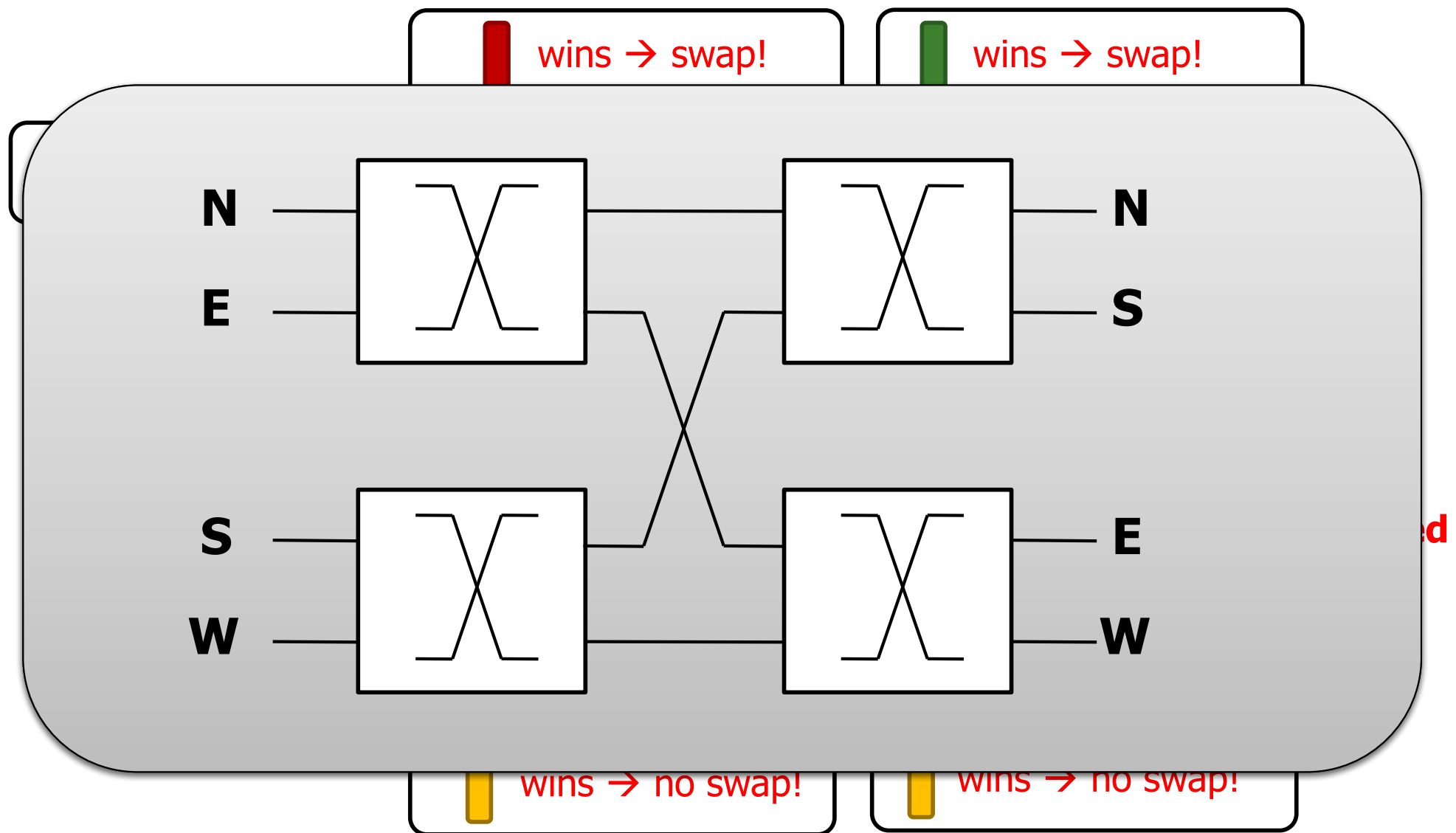
# Golden Flit Routing with Four Inputs

---

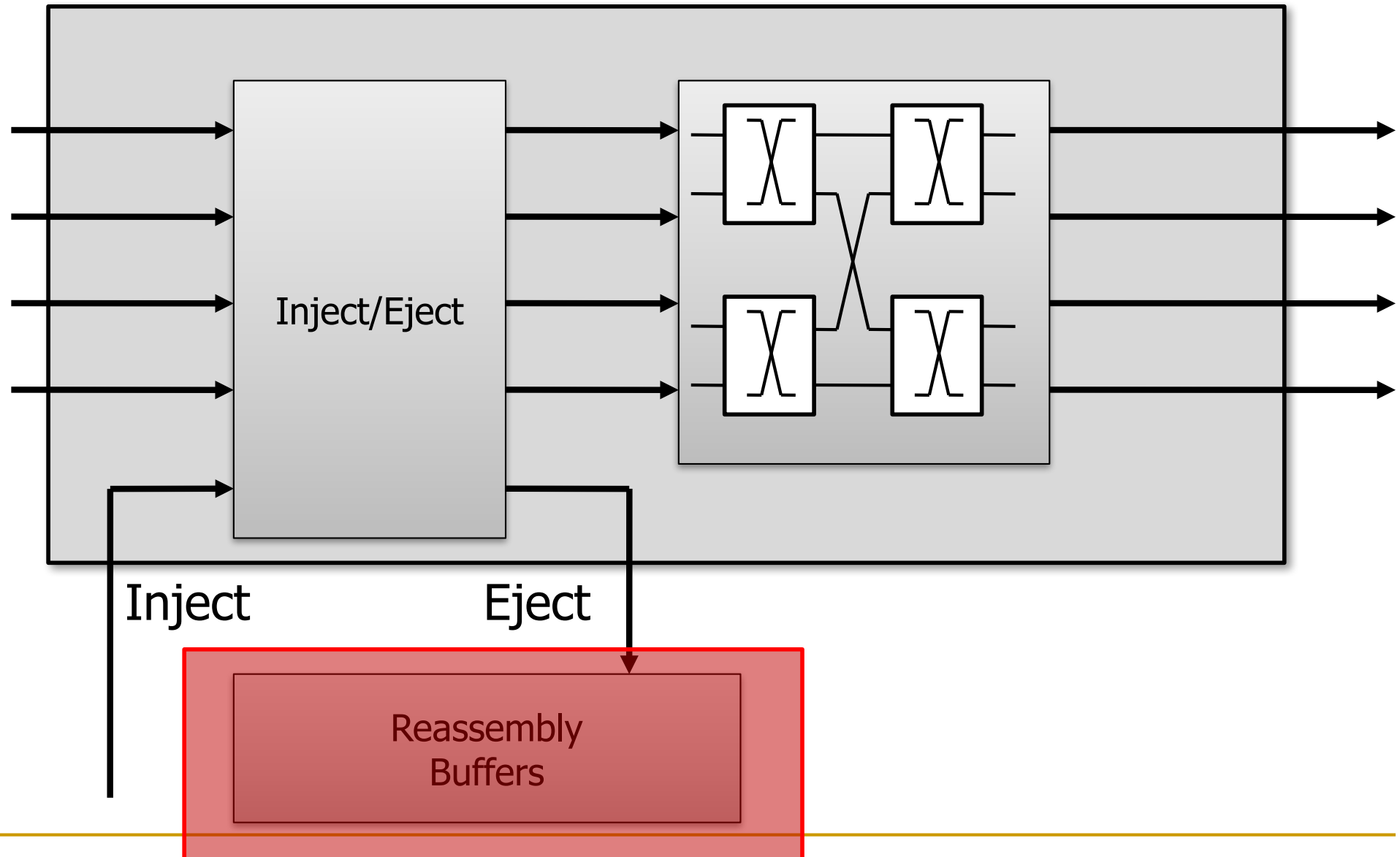
- Each block makes decisions **independently!**
  - **Deflection is a distributed decision**



# Permutation Network Operation



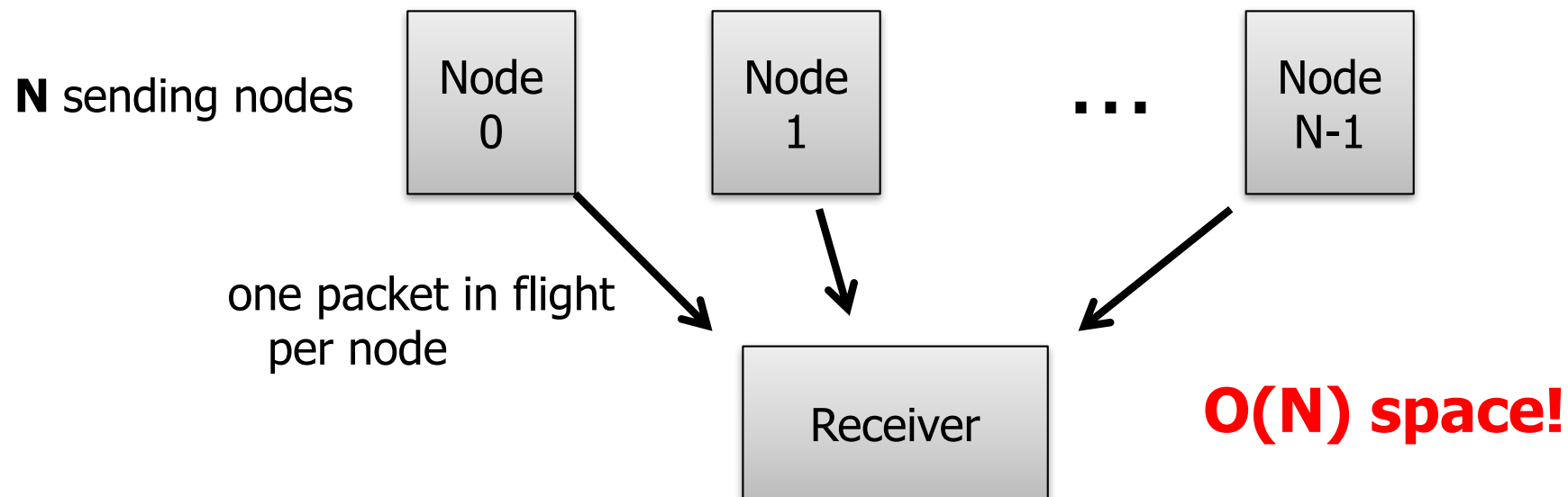
## Problem 2: Packet Reassembly



# Reassembly Buffers are Large

---

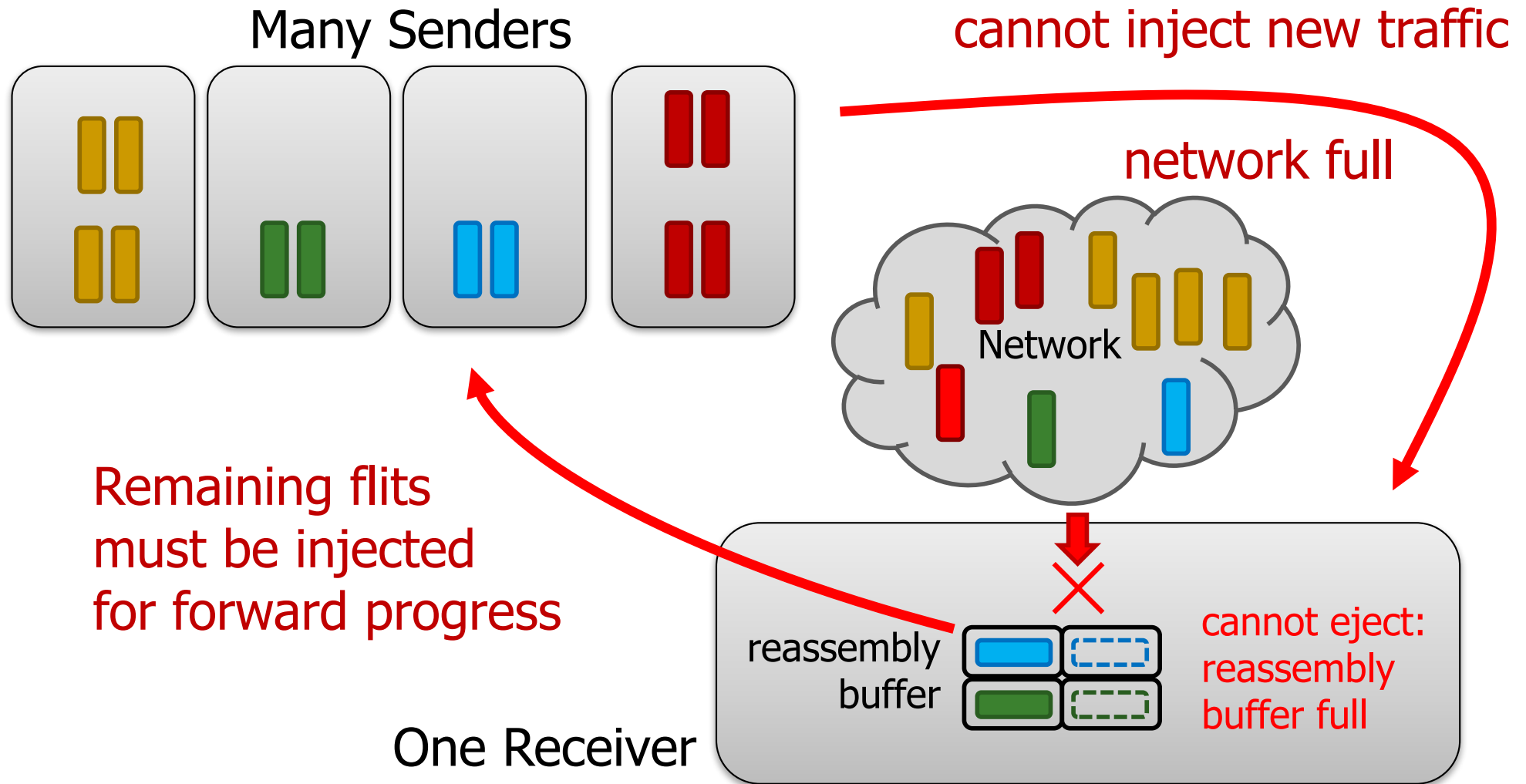
- **Worst case:** every node sends a packet to one receiver
- Why can't we **make reassembly buffers smaller?**





# Small Reassembly Buffers Cause Deadlock

- What happens when reassembly buffer is too small?

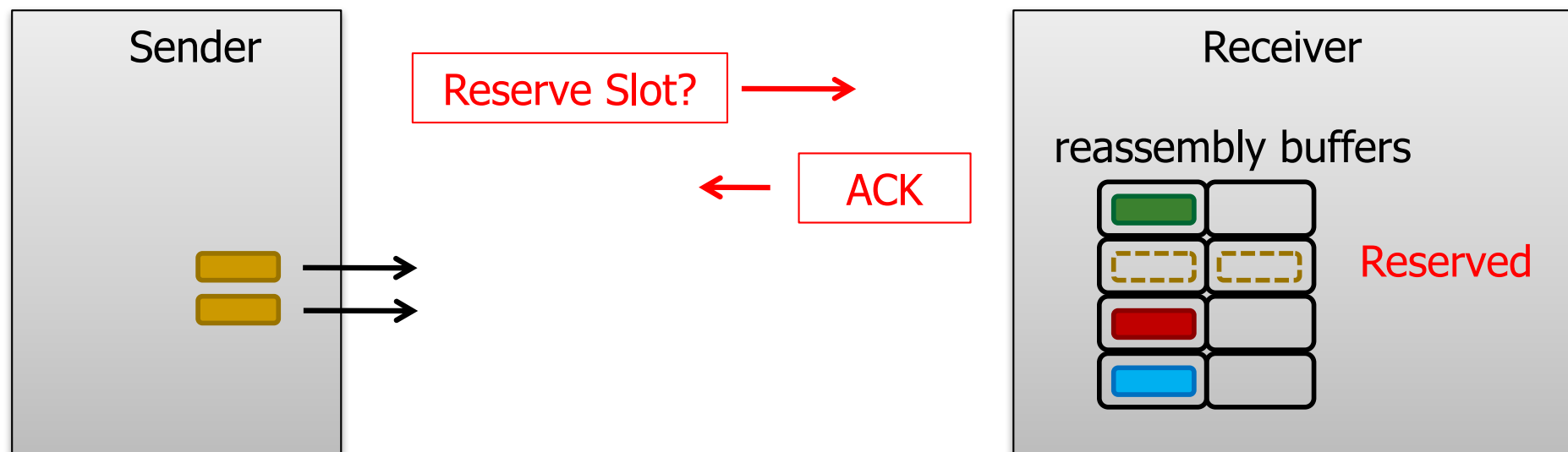


# Reserve Space to Avoid Deadlock?

- What if every sender **asks permission** from the receiver before it sends?

➔ **adds additional delay** to every request

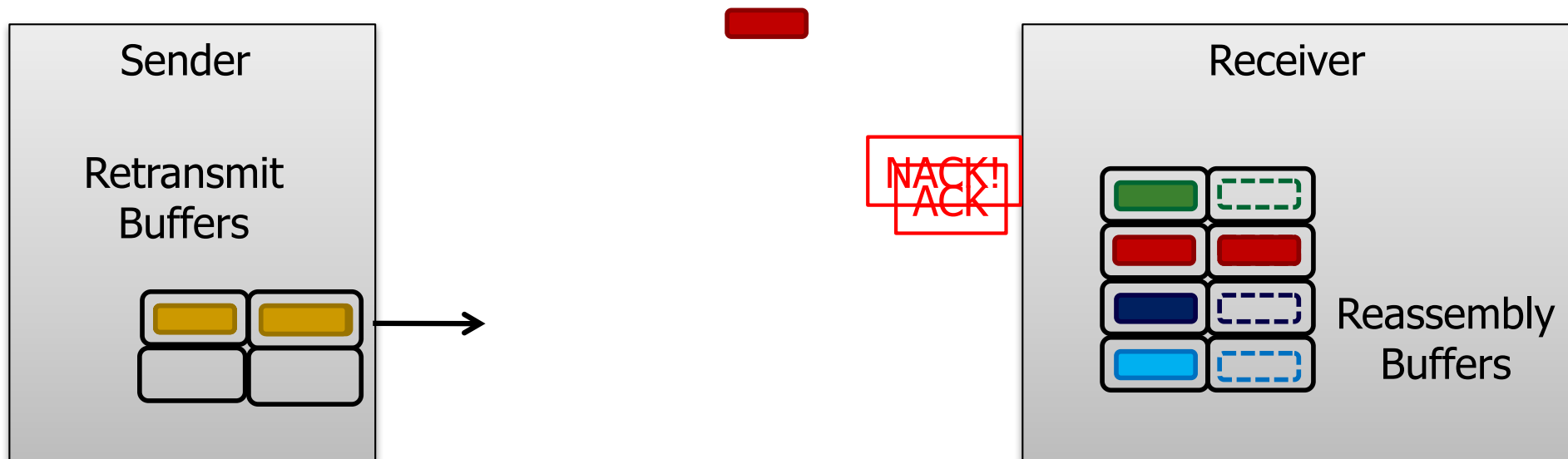
- ➔ 1. Reserve Slot
- ➔ 2. ACK
- ➔ 3. Send Packet



# Escaping Deadlock with Retransmissions

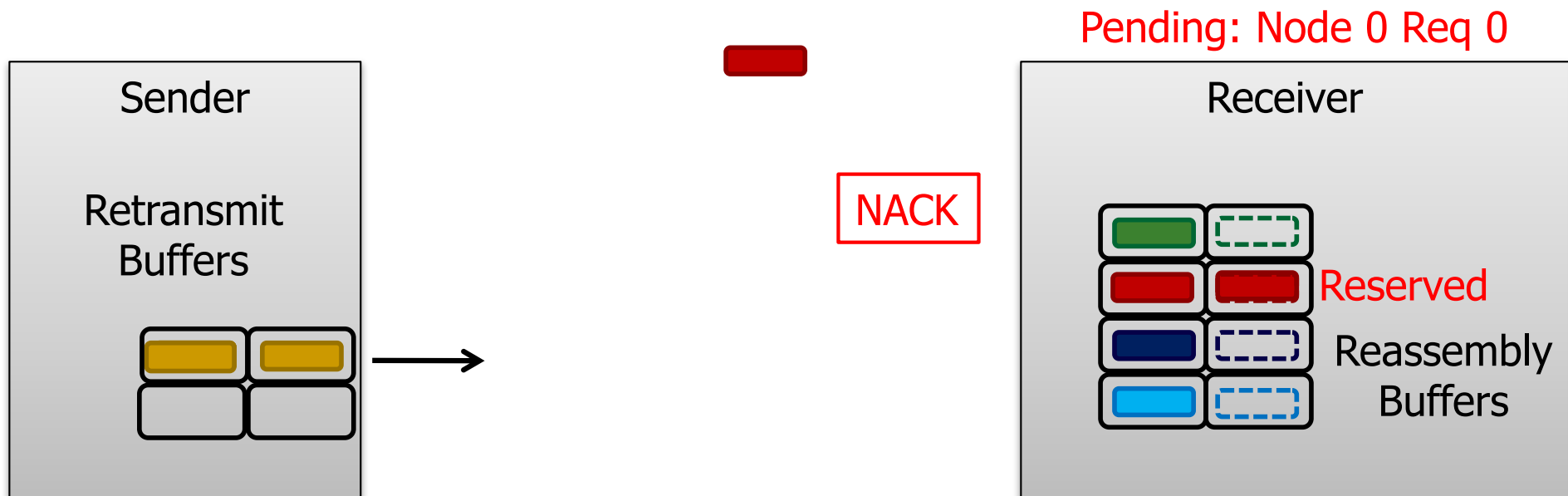
- Sender is optimistic instead: assume buffer is free
  - If not, receiver **drops** and NACKs; sender **retransmits**

- **no additional delay in best case**
  - **transmit buffering overhead for all packets**
  - **potentially many retransmits**
1. Send (2 flits)
  2. Drop, NACK
  3. Other packet completes
  4. Retransmit packet
  5. ACK
  6. Sender frees data

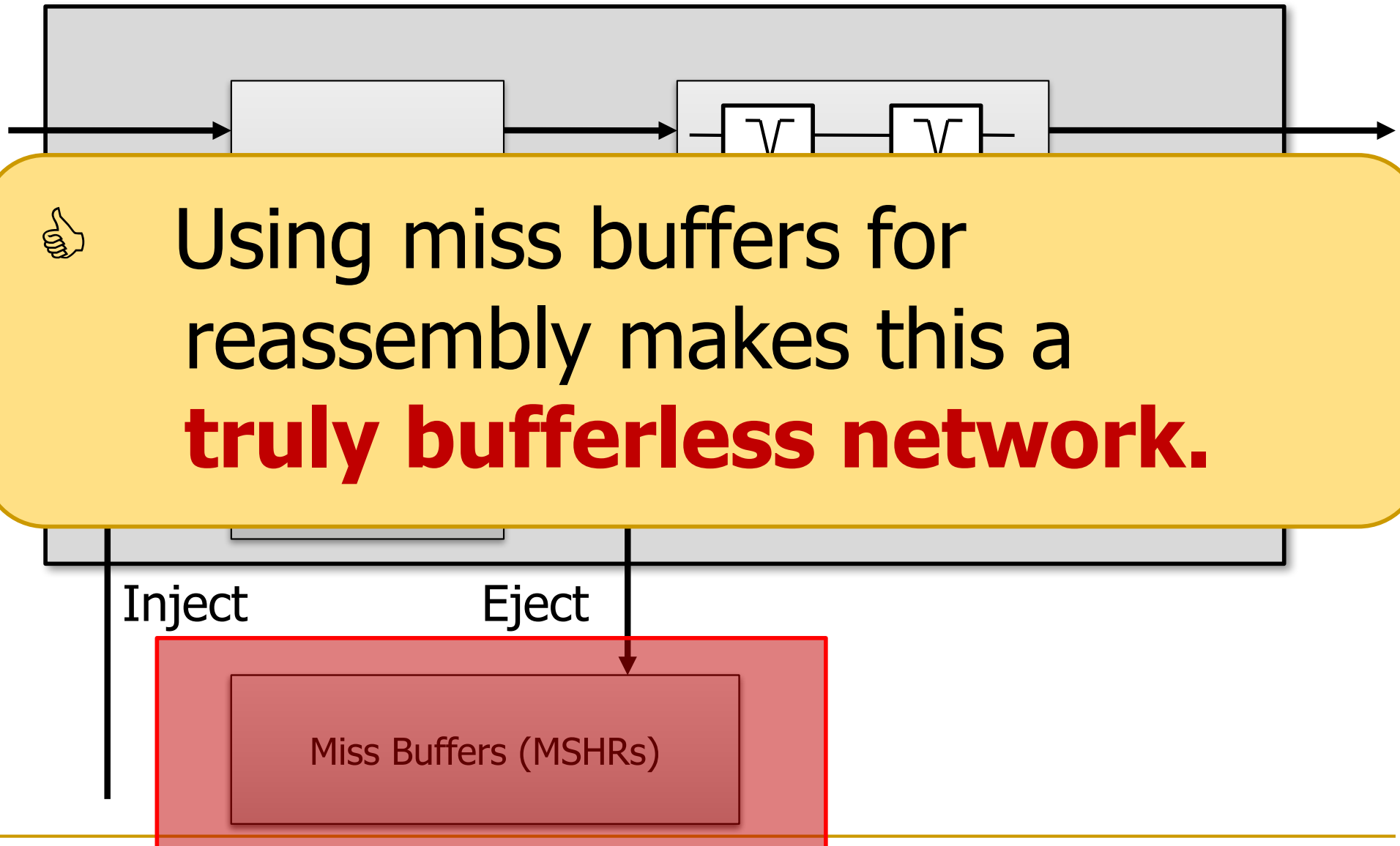


# Solution: Retransmitting Only Once

- **Key Idea:** Retransmit only **when space becomes available**.
  - Receiver **drops packet** if full; **notes** which packet it drops
  - When space frees up, receiver **reserves space** so retransmit is successful
  - Receiver notifies sender to retransmit

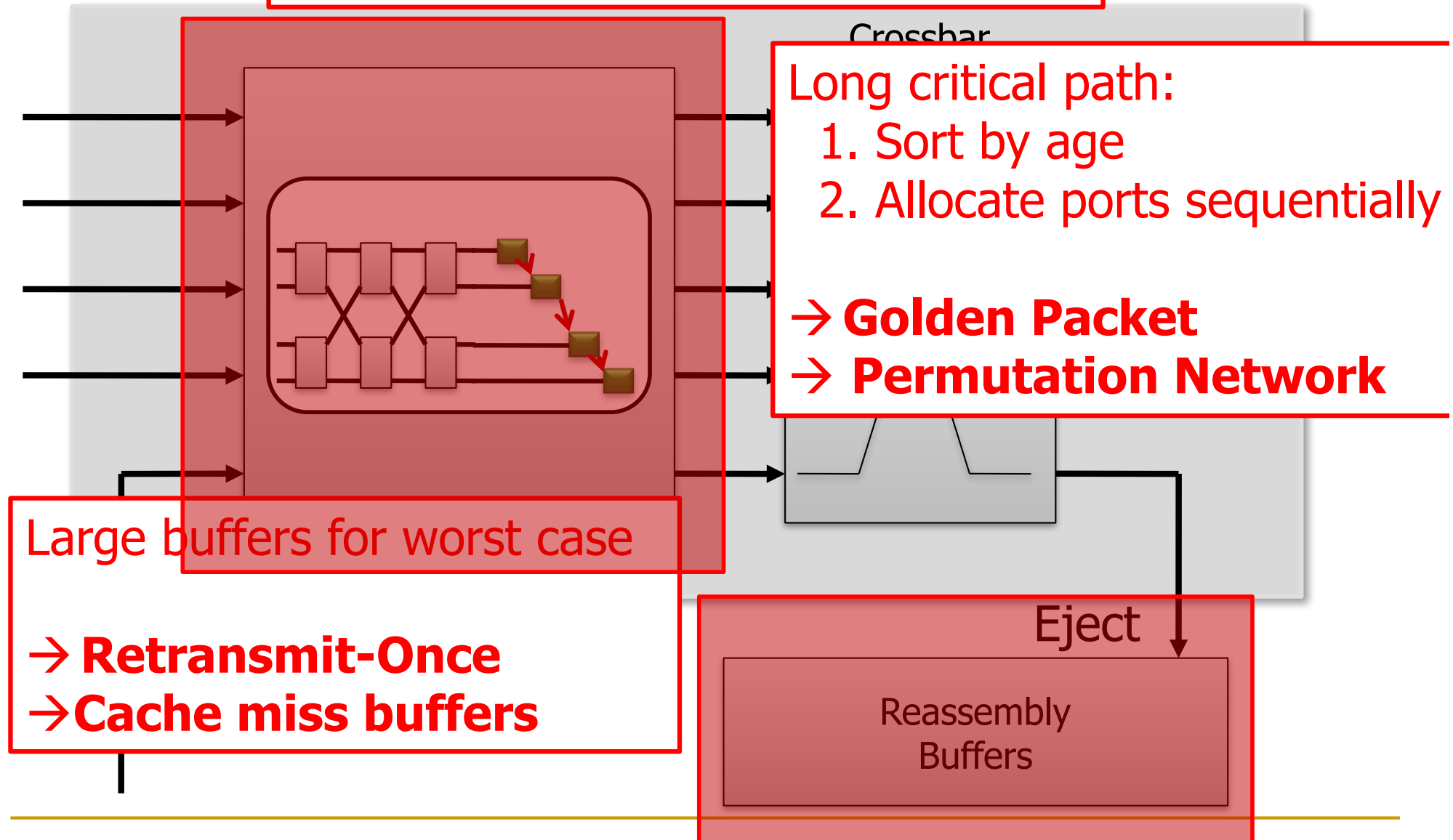


# Using MSHRs as Reassembly Buffers

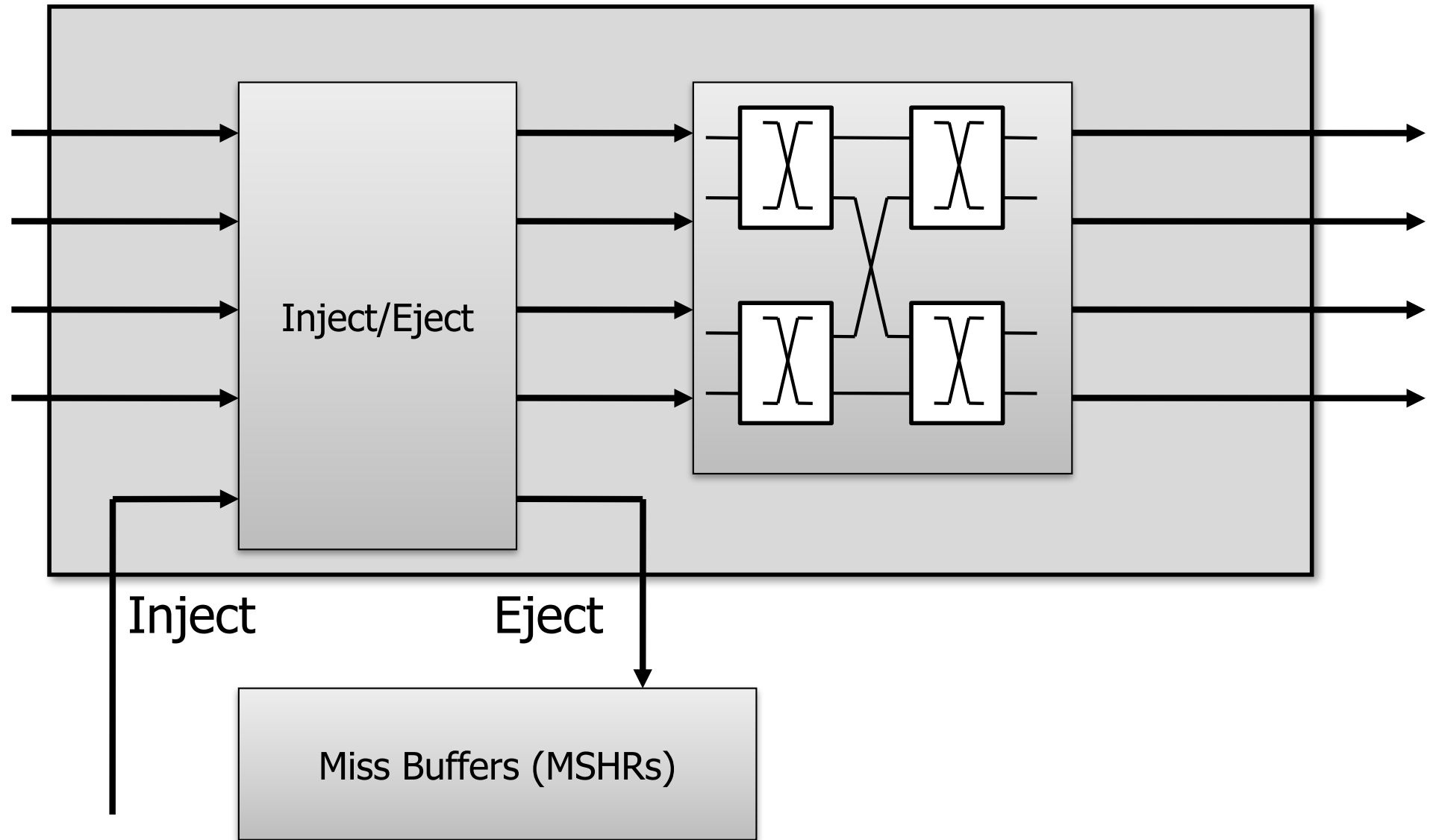


# CHIPPER: Cheap Interconnect Partially-Permuting Router

## Baseline Bufferless Deflection Router



# CHIPPER: Cheap Interconnect Partially-Permuting Router



---

# EVALUATION



# Methodology

---

- **Multiprogrammed** workloads: CPU2006, server, desktop
  - 8x8 (64 cores), 39 homogeneous and 10 mixed sets
- **Multithreaded** workloads: SPLASH-2, 16 threads
  - 4x4 (16 cores), 5 applications
- **System configuration**
  - **Buffered** baseline: 2-cycle router, 4 VCs/channel, 8 flits/VC
  - **Bufferless** baseline: 2-cycle latency, FLIT-BLESS
  - Instruction-trace driven, closed-loop, 128-entry OoO window
  - 64KB L1, **perfect L2 (stresses interconnect)**, XOR mapping

# Methodology

---

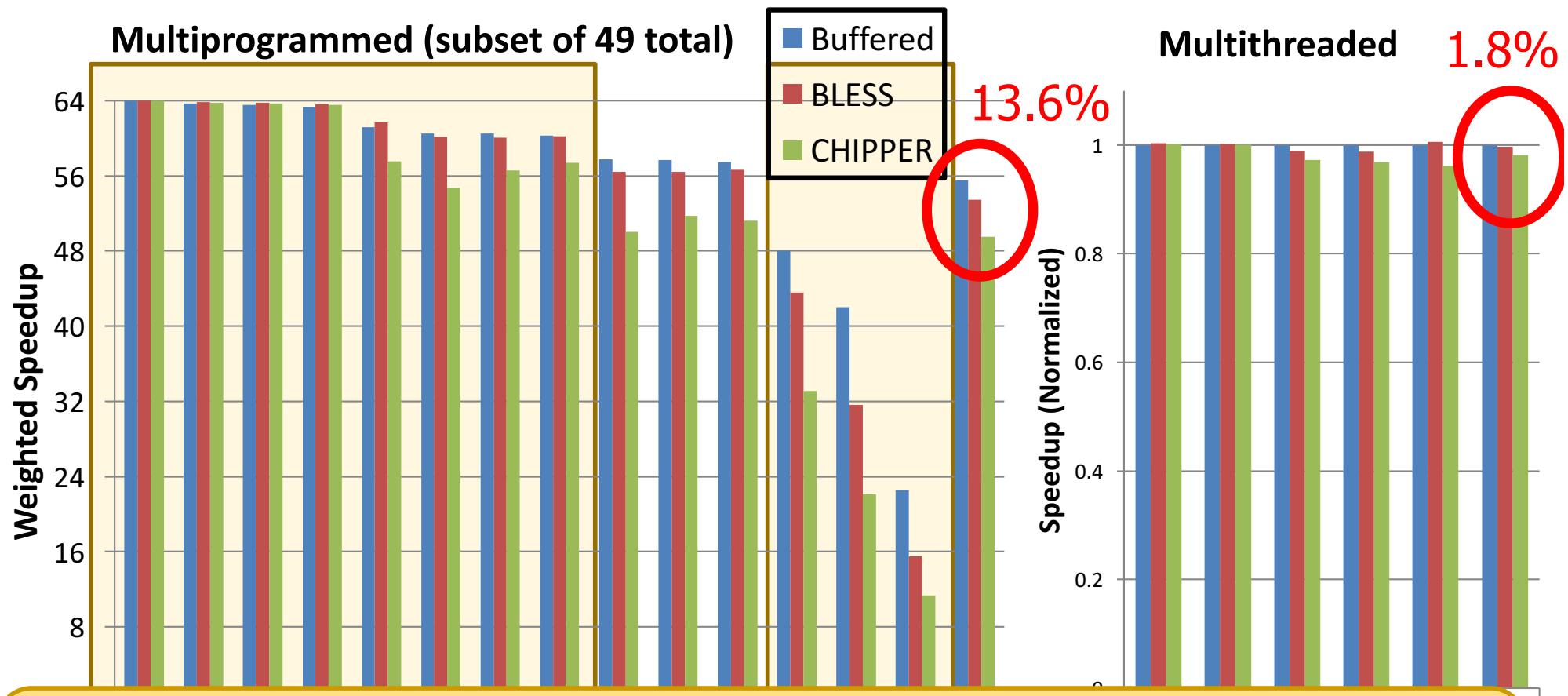
## ■ **Hardware modeling**

- Verilog models for CHIPPER, BLESS, buffered logic
  - Synthesized with commercial 65nm library
- ORION for crossbar, buffers and links

## ■ **Power**

- Static and dynamic power from hardware models
- Based on event counts in cycle-accurate simulations

# Results: Performance Degradation

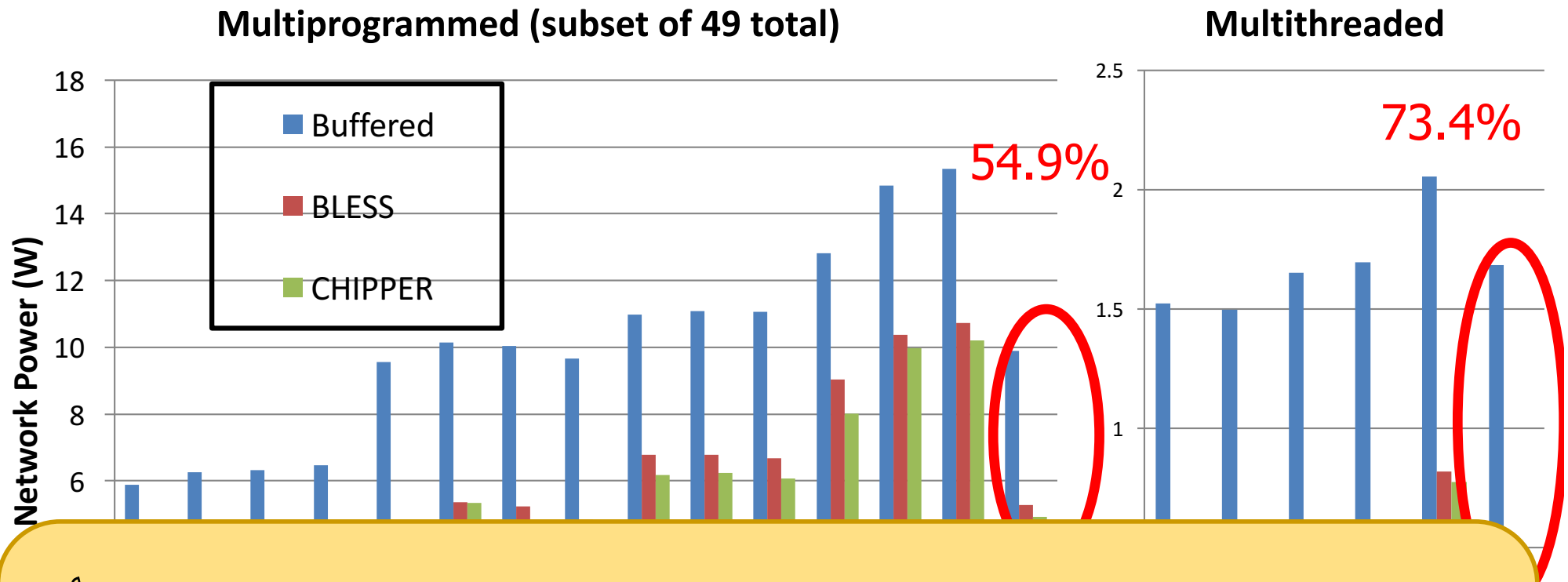


Minimal loss for low-to-medium-intensity workloads

5.0%

49.8%

# Results: Power Reduction

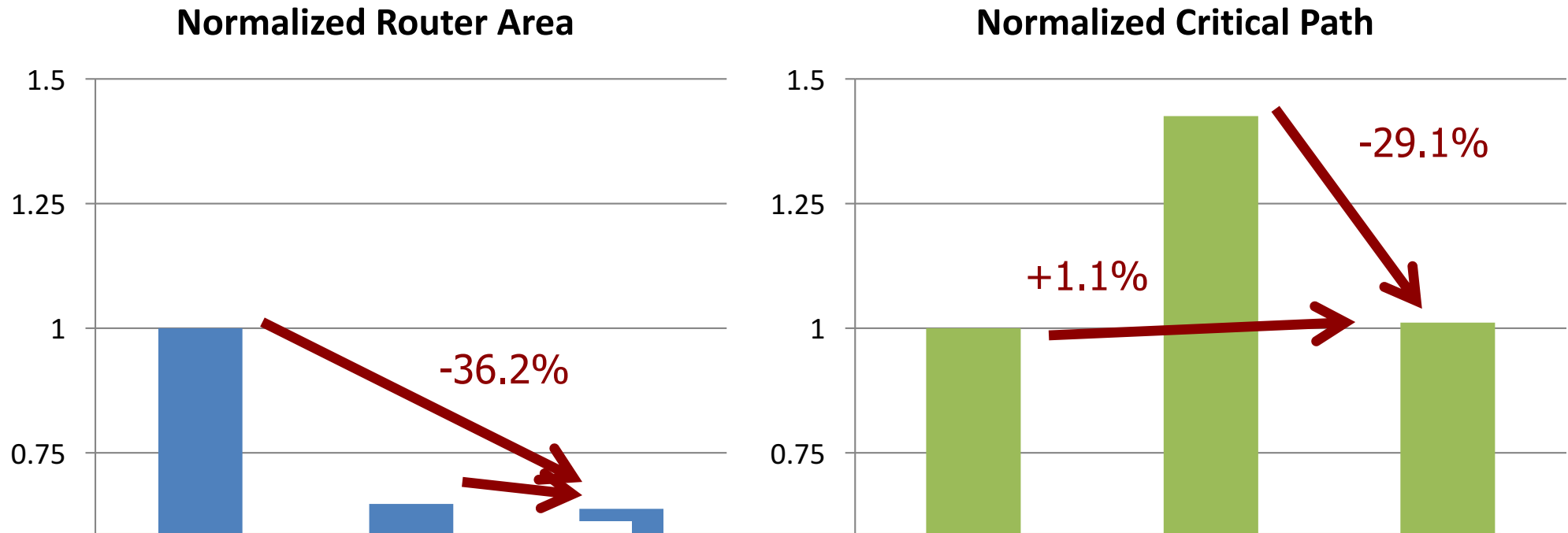


Removing buffers → majority of power savings



Slight savings from BLESS to CHIPPER

# Results: Area and Critical Path Reduction



**CHIPPER maintains area savings** of BLESS



Critical path **becomes competitive** to buffered

# Conclusions

---

- Two key issues in bufferless deflection routing
  - livelock freedom and packet reassembly
- Bufferless deflection routers were high-complexity and impractical
  - Oldest-first prioritization → long critical path in router
  - No end-to-end flow control for reassembly → prone to deadlock with reasonably-sized reassembly buffers
- CHIPPER is a new, practical bufferless deflection router
  - Golden packet prioritization → short critical path in router
  - Retransmit-once protocol → deadlock-free packet reassembly
  - Cache miss buffers as reassembly buffers → truly bufferless network
- CHIPPER frequency comparable to buffered routers at much lower area and power cost, and minimal performance loss

# More on CHIPPER

---

- Chris Fallin, Chris Craik, and Onur Mutlu,  
**"CHIPPER: A Low-Complexity Bufferless Deflection Router"**  
*Proceedings of the 17th International Symposium on High-Performance Computer Architecture (HPCA)*, pages 144-155, San Antonio, TX, February 2011. Slides (pptx)  
An extended version as SAFARI Technical Report, TR-SAFARI-2010-001, Carnegie Mellon University, December 2010.

## CHIPPER: A Low-complexity Bufferless Deflection Router

Chris Fallin                      Chris Craik                      Onur Mutlu  
cfallin@cmu.edu    craik@cmu.edu    onur@cmu.edu

Computer Architecture Lab (CALCM)  
Carnegie Mellon University

# Minimally-Buffered Deflection Routing

---

- Chris Fallin, Greg Nazario, Xiangyao Yu, Kevin Chang, Rachata Ausavarungnirun, and Onur Mutlu,  
**"MinBD: Minimally-Buffered Deflection Routing for Energy-Efficient Interconnect"**  
*Proceedings of the 6th ACM/IEEE International Symposium on Networks on Chip (NOCS)*, Lyngby, Denmark, May 2012. [Slides \(pptx\)](#) [\(pdf\)](#)

## MinBD: Minimally-Buffered Deflection Routing for Energy-Efficient Interconnect

Chris Fallin, Greg Nazario, Xiangyao Yu<sup>†</sup>, Kevin Chang, Rachata Ausavarungnirun, Onur Mutlu

Carnegie Mellon University  
{cfallin,gnazario,kevincha,rachata,onur}@cmu.edu

<sup>†</sup>Tsinghua University & Carnegie Mellon University  
yxythu@gmail.com



# “Bufferless” Hierarchical Rings

---

- Ausavarungnirun et al., “Design and Evaluation of Hierarchical Rings with Deflection Routing,” SBAC-PAD 2014.
  - [http://users.ece.cmu.edu/~omutlu/pub/hierarchical-rings-with-deflection\\_sbacpad14.pdf](http://users.ece.cmu.edu/~omutlu/pub/hierarchical-rings-with-deflection_sbacpad14.pdf)
- Discusses the design and implementation of a mostly-bufferless hierarchical ring

## Design and Evaluation of Hierarchical Rings with Deflection Routing

Rachata Ausavarungnirun   Chris Fallin   Xiangyao Yu†   Kevin Kai-Wei Chang

Greg Nazario   Reetuparna Das§   Gabriel H. Loh‡   Onur Mutlu

Carnegie Mellon University   §University of Michigan   †MIT   ‡Advanced Micro Devices, Inc.

# “Bufferless” Hierarchical Rings (II)

---

- Rachata Ausavarungnirun, Chris Fallin, Xiangyao Yu, Kevin Chang, Greg Nazario, Reetuparna Das, Gabriel Loh, and Onur Mutlu,  
**"A Case for Hierarchical Rings with Deflection Routing: An Energy-Efficient On-Chip Communication Substrate"**  
***Parallel Computing (PARCO)***, to appear in 2016.
  - [arXiv.org version](https://arxiv.org/abs/1602.04888), February 2016.

Achieving both High Energy Efficiency  
and High Performance in On-Chip Communication  
using Hierarchical Rings with Deflection Routing

Rachata Ausavarungnirun   Chris Fallin   Xiangyao Yu<sup>†</sup>   Kevin Kai-Wei Chang  
Greg Nazario   Reetuparna Das<sup>§</sup>   Gabriel H. Loh<sup>‡</sup>   Onur Mutlu  
Carnegie Mellon University   §University of Michigan   <sup>†</sup>MIT   <sup>‡</sup>AMD

# Summary of Six Years of Research

---

- Chris Fallin, Greg Nazario, Xiangyao Yu, Kevin Chang, Rachata Ausavarungnirun, and Onur Mutlu,  
**"Bufferless and Minimally-Buffered Deflection Routing"**  
*Invited Book Chapter in Routing Algorithms in Networks-on-Chip, pp. 241-275, Springer, 2014.*

## Chapter 1

# Bufferless and Minimally-Buffered Deflection Routing

Chris Fallin, Greg Nazario, Xiangyao Yu, Kevin Chang, Rachata Ausavarungnirun, Onur Mutlu

# On-Chip vs. Off-Chip Tradeoffs

---

- George Nychis, Chris Fallin, Thomas Moscibroda, Onur Mutlu, and Srinivasan Seshan,  
**"On-Chip Networks from a Networking Perspective: Congestion and Scalability in Many-core Interconnects"**  
*Proceedings of the 2012 ACM SIGCOMM Conference (**SIGCOMM**), Helsinki, Finland, August 2012. Slides (pptx)*

## On-Chip Networks from a Networking Perspective: Congestion and Scalability in Many-Core Interconnects

George Nychis<sup>†</sup>, Chris Fallin<sup>†</sup>, Thomas Moscibroda<sup>§</sup>, Onur Mutlu<sup>†</sup>, Srinivasan Seshan<sup>†</sup>

<sup>†</sup> Carnegie Mellon University  
{gnychis,cfallin,onur,srini}@cmu.edu

<sup>§</sup> Microsoft Research Asia  
moscitho@microsoft.com

# Packet Scheduling

---

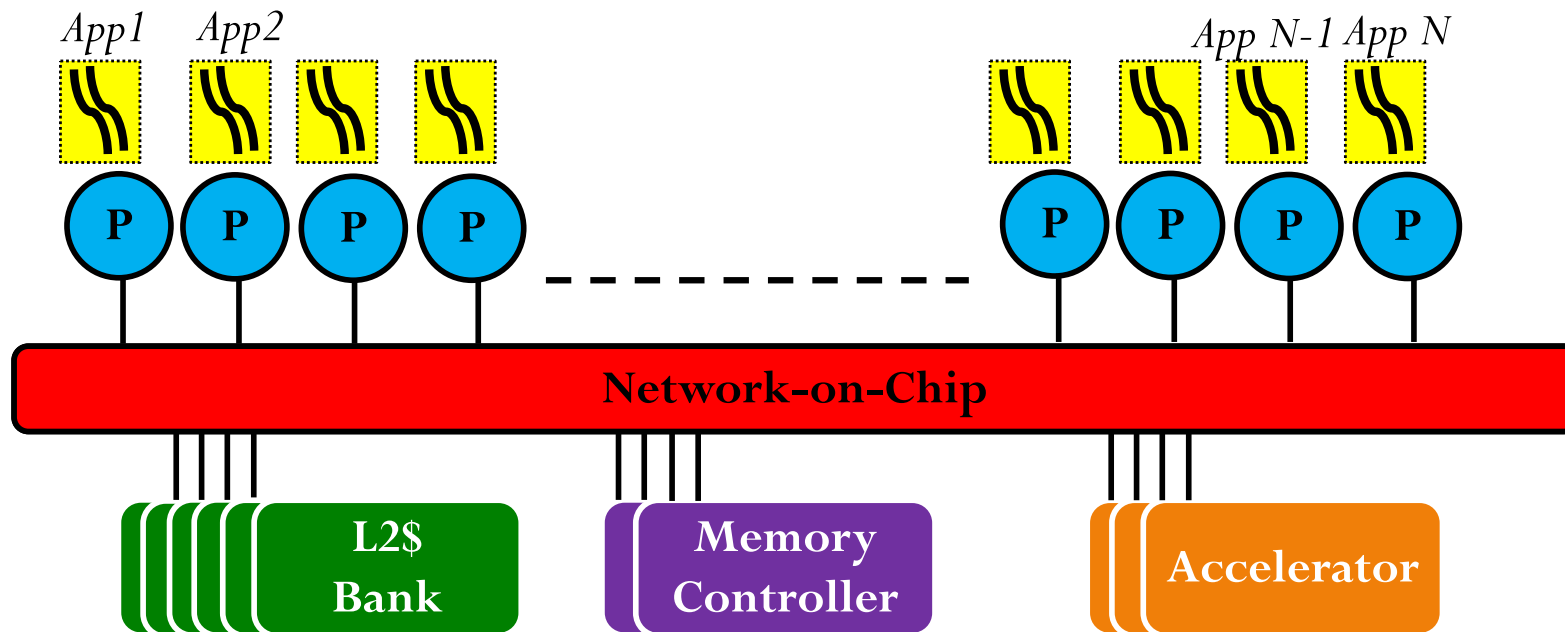
- Which packet to choose for a given output port?
  - ❑ Router needs to prioritize between competing flits
  - ❑ Which input port?
  - ❑ Which virtual channel?
  - ❑ Which application's packet?
  
- Common strategies
  - ❑ Round robin across virtual channels
  - ❑ Oldest packet first (or an approximation)
  - ❑ Prioritize some virtual channels over others
  
- Better policies in a multi-core environment
  - ❑ Use application characteristics

# Application-Aware Packet Scheduling

Das et al., "Application-Aware Prioritization Mechanisms for On-Chip Networks," MICRO 2009.

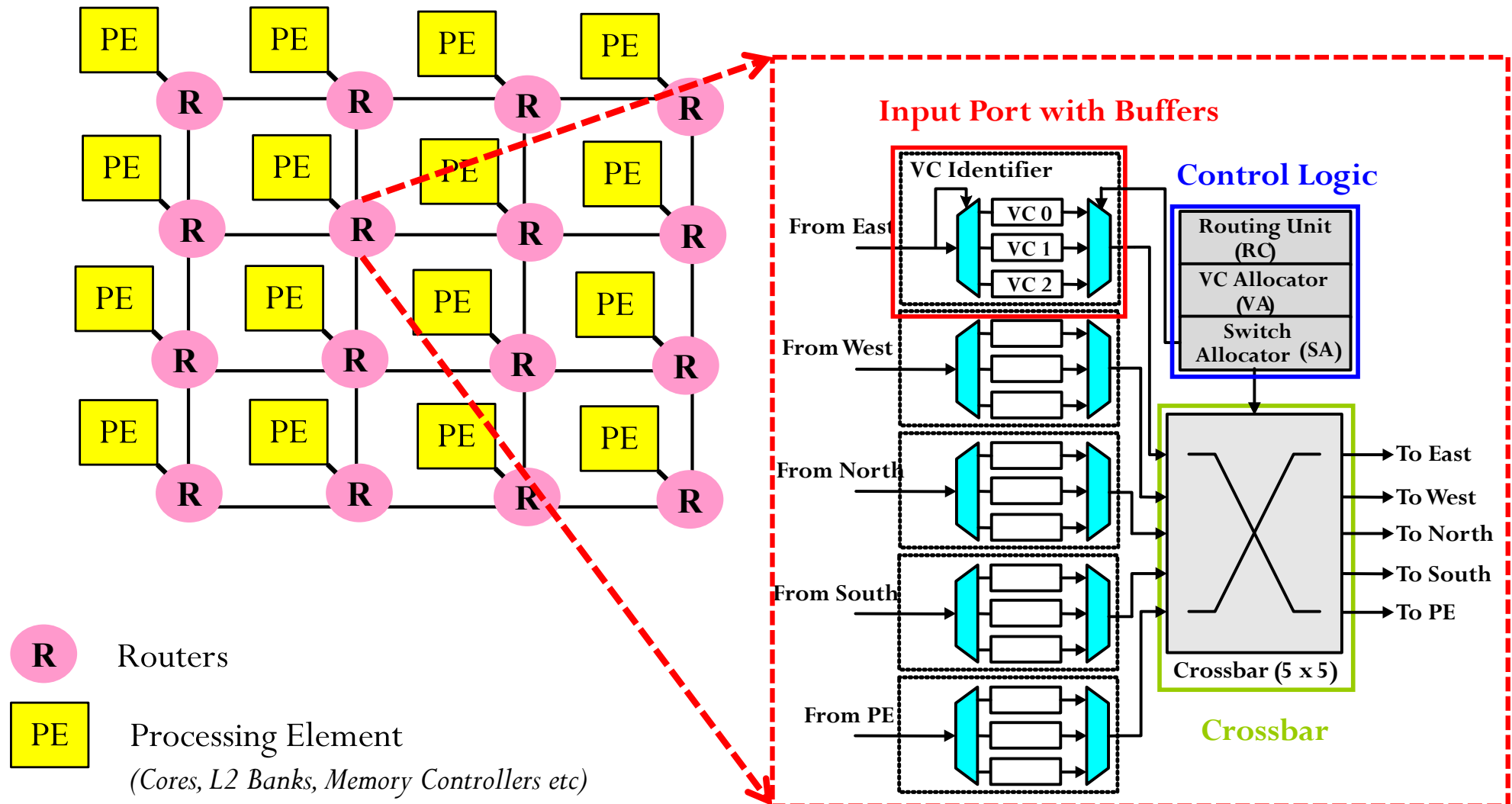
# The Problem: Packet Scheduling

---



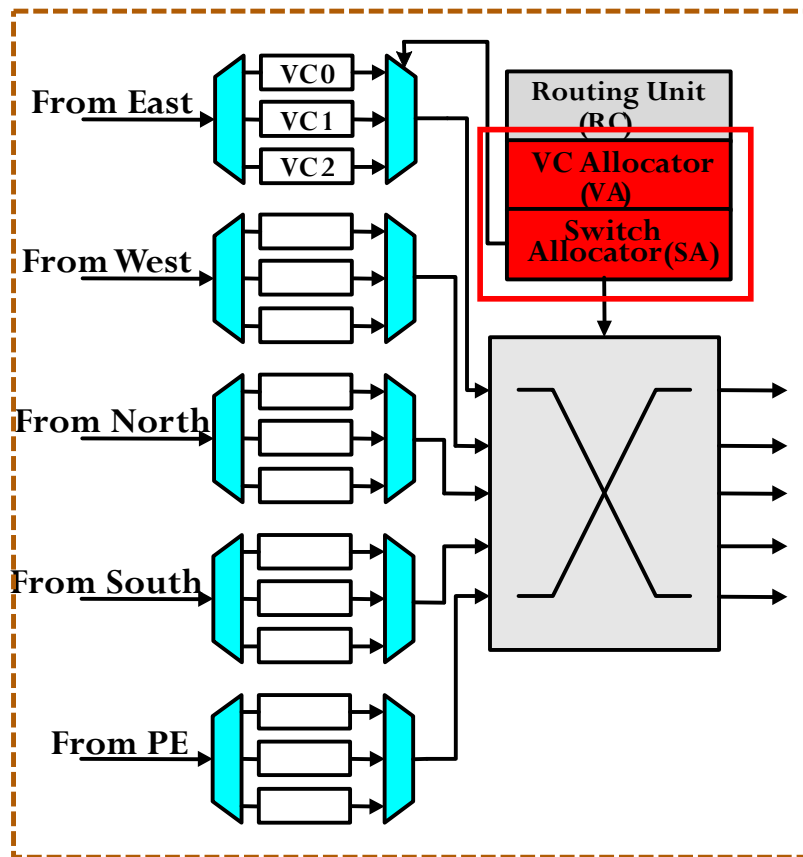
Network-on-Chip is a **critical** resource  
shared by **multiple applications**

# The Problem: Packet Scheduling

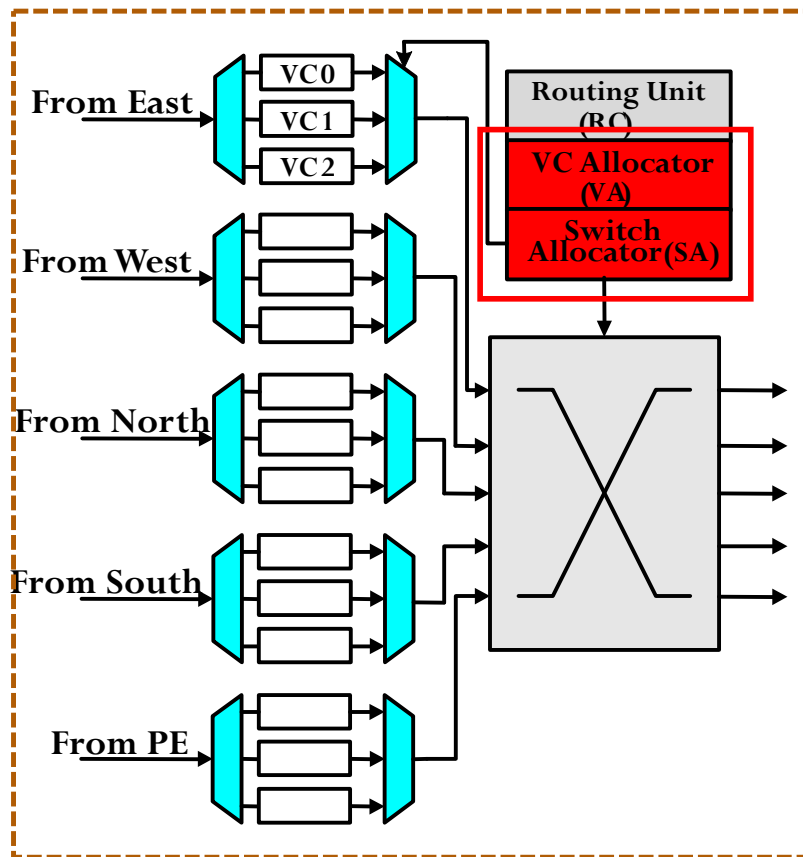




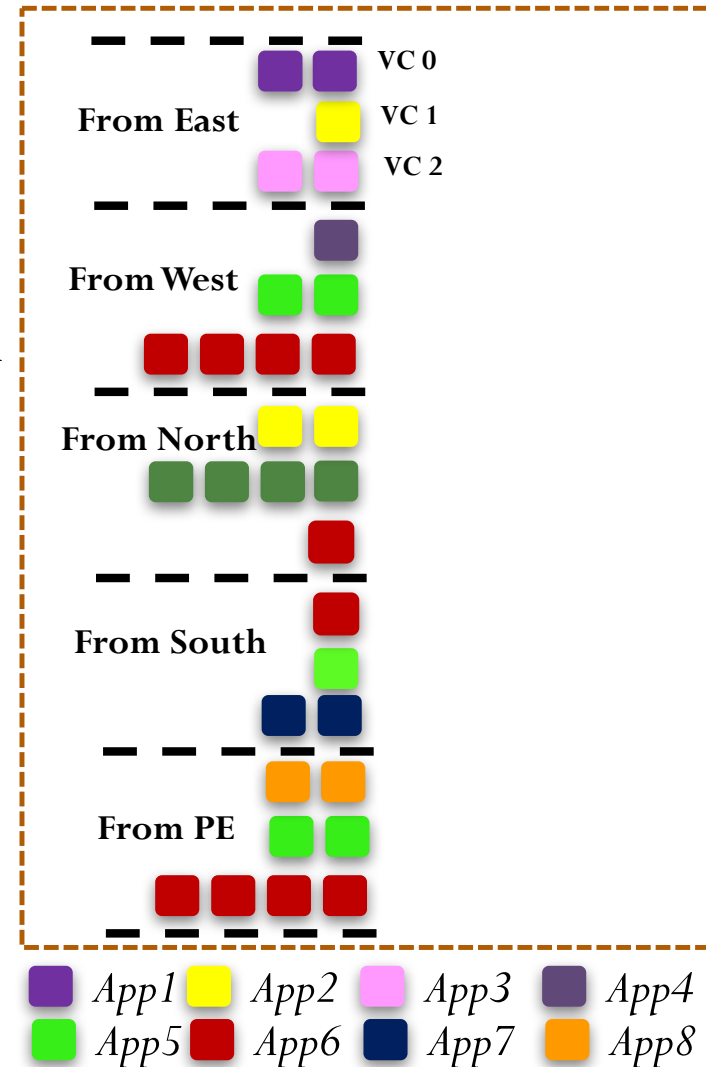
# The Problem: Packet Scheduling



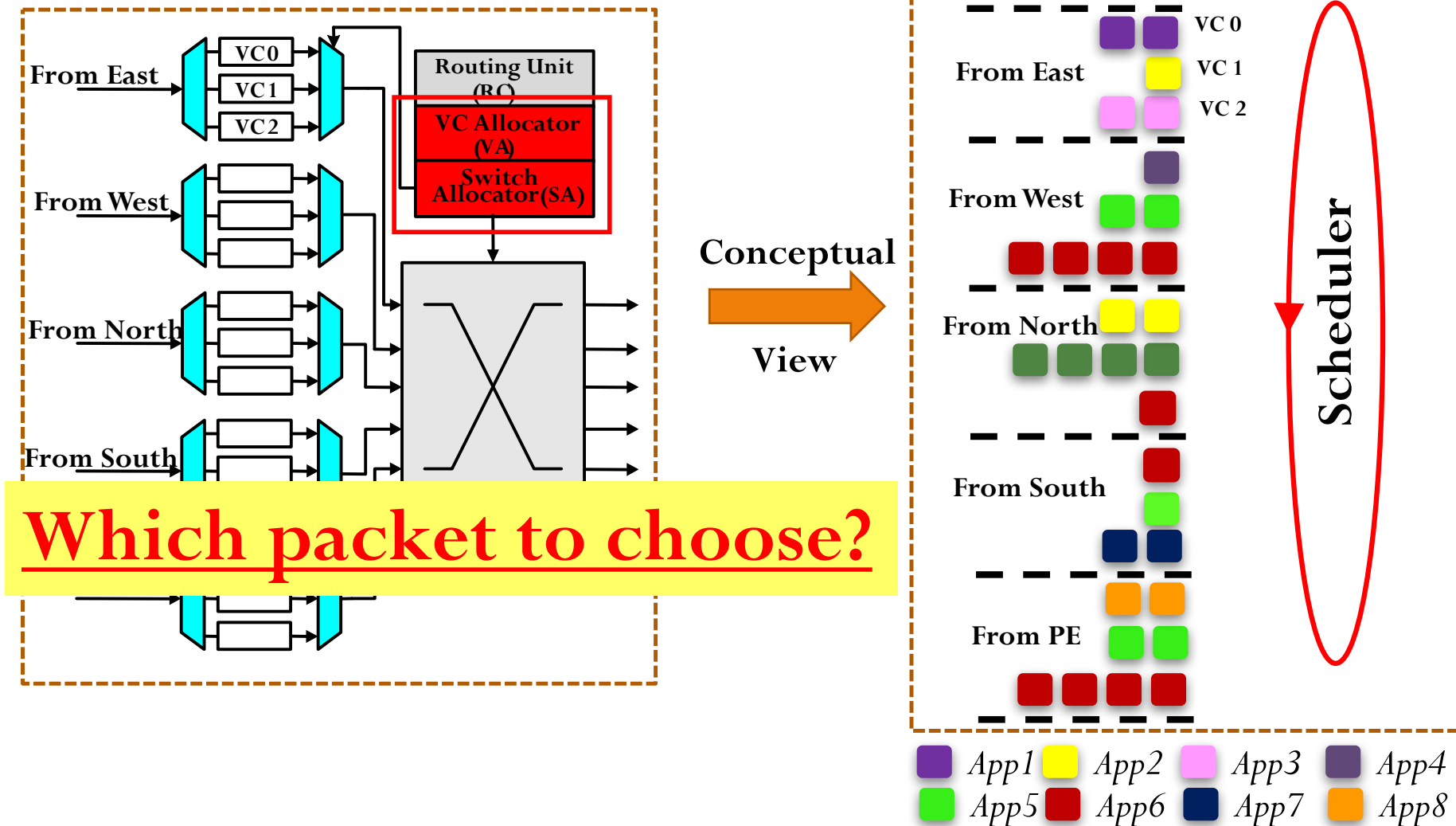
# The Problem: Packet Scheduling



Conceptual  
View



# The Problem: Packet Scheduling

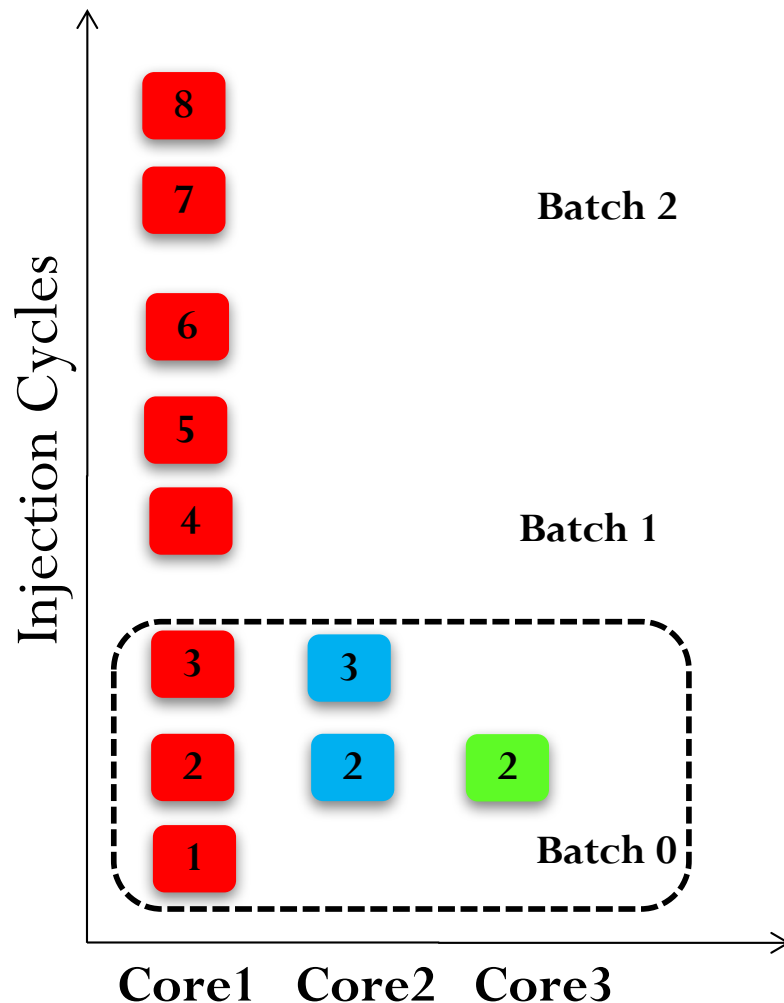


# The Problem: Packet Scheduling




---

- Existing scheduling policies
  - Round Robin
  - Age
- Problem 1: **Local** to a router
  - Lead to contradictory decision making between routers: packets from one application may be prioritized at one router, to be delayed at next.
- Problem 2: **Application oblivious**
  - Treat all applications packets equally
  - But applications are heterogeneous
- **Solution** : Application-aware global scheduling policies.

# STC Scheduling Example

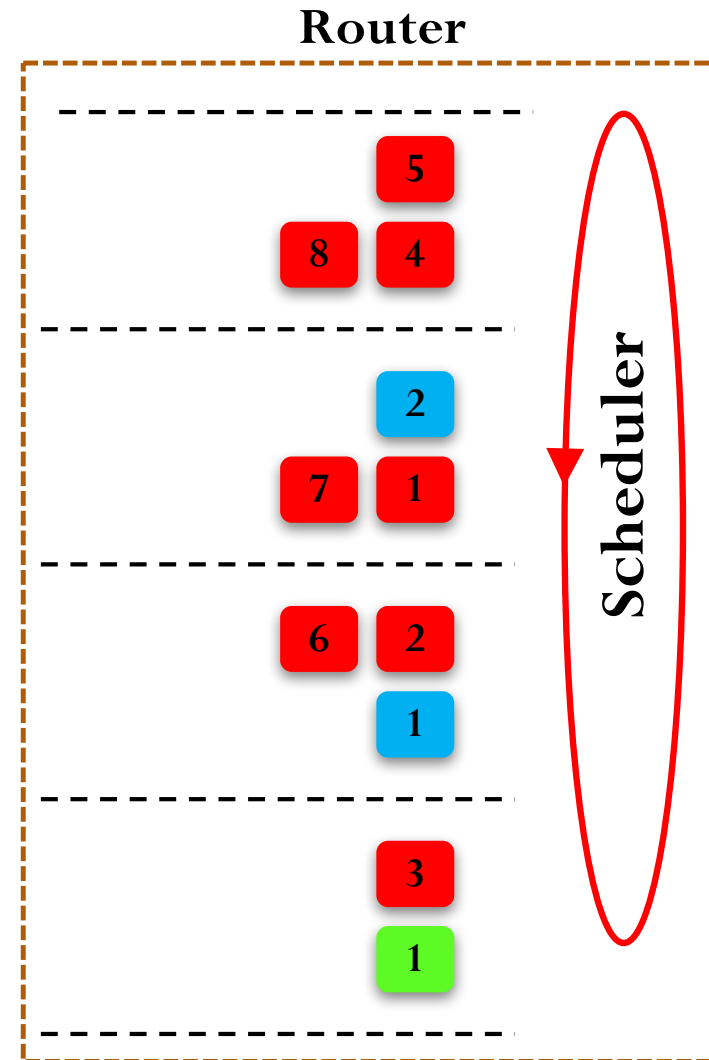
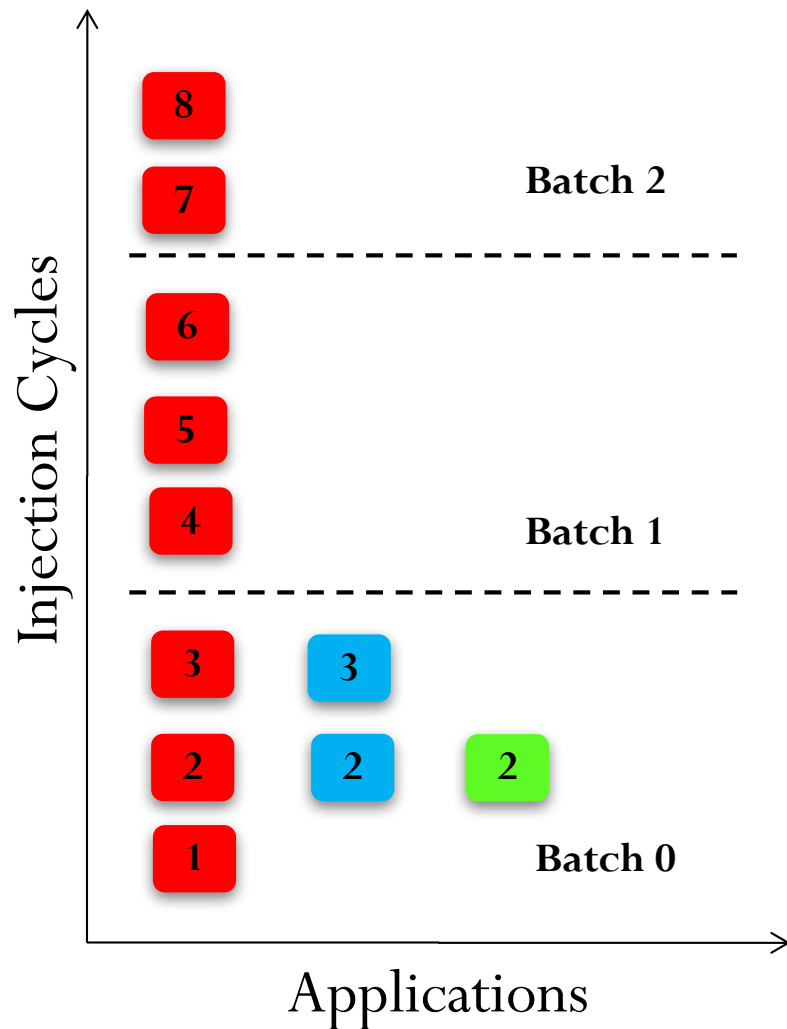


Batching interval length = 3 cycles

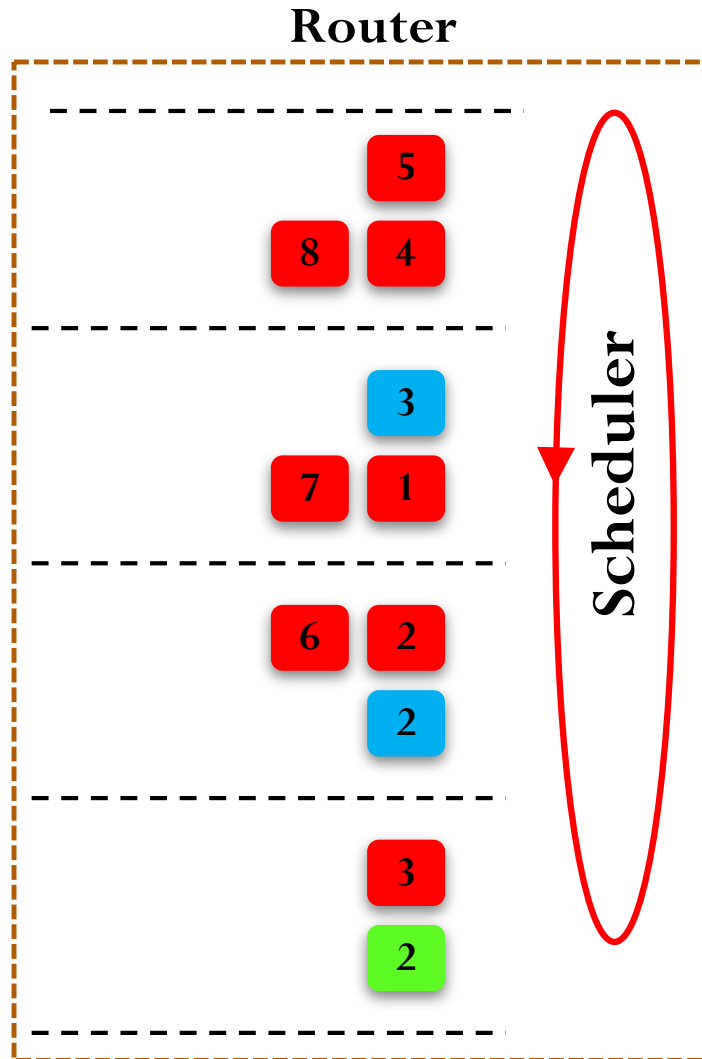
Ranking order =  >  > 

**Packet Injection Order at Processor**

# STC Scheduling Example

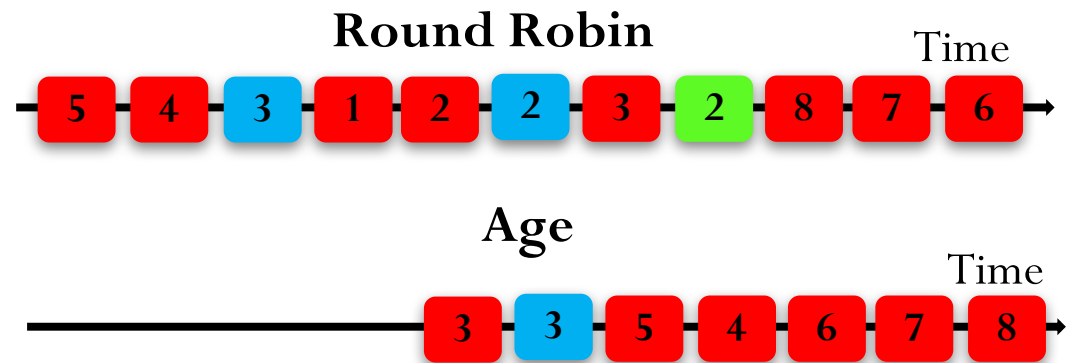
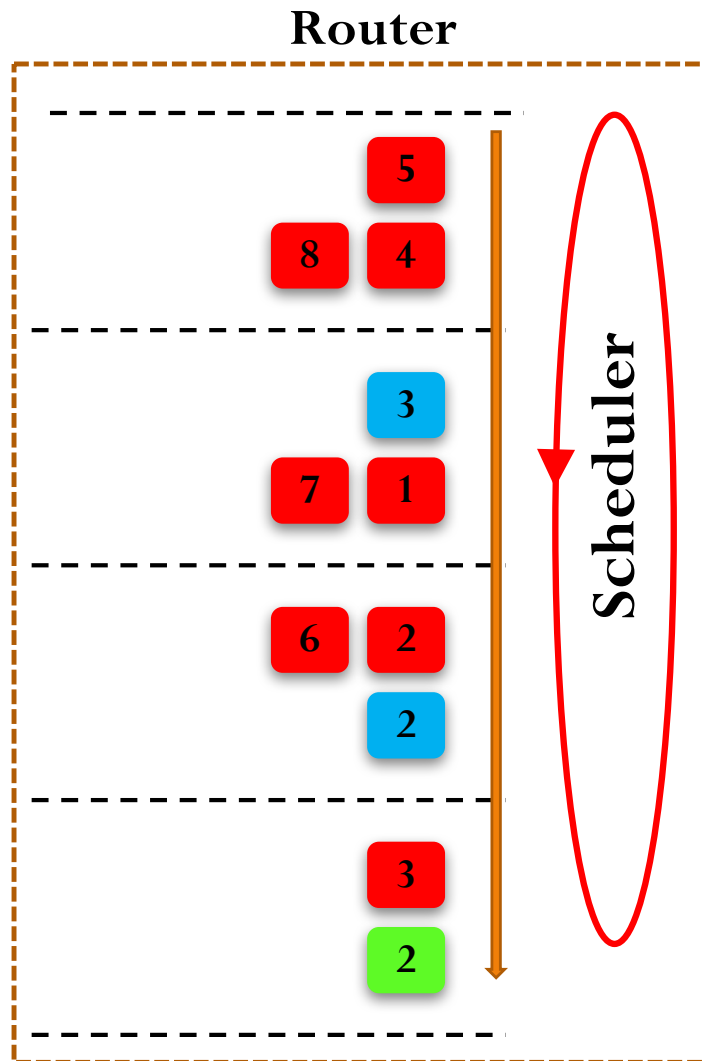


# STC Scheduling Example



STALL CYCLES				Avg
RR	8	6	11	8.3
Age				
STC				

# STC Scheduling Example



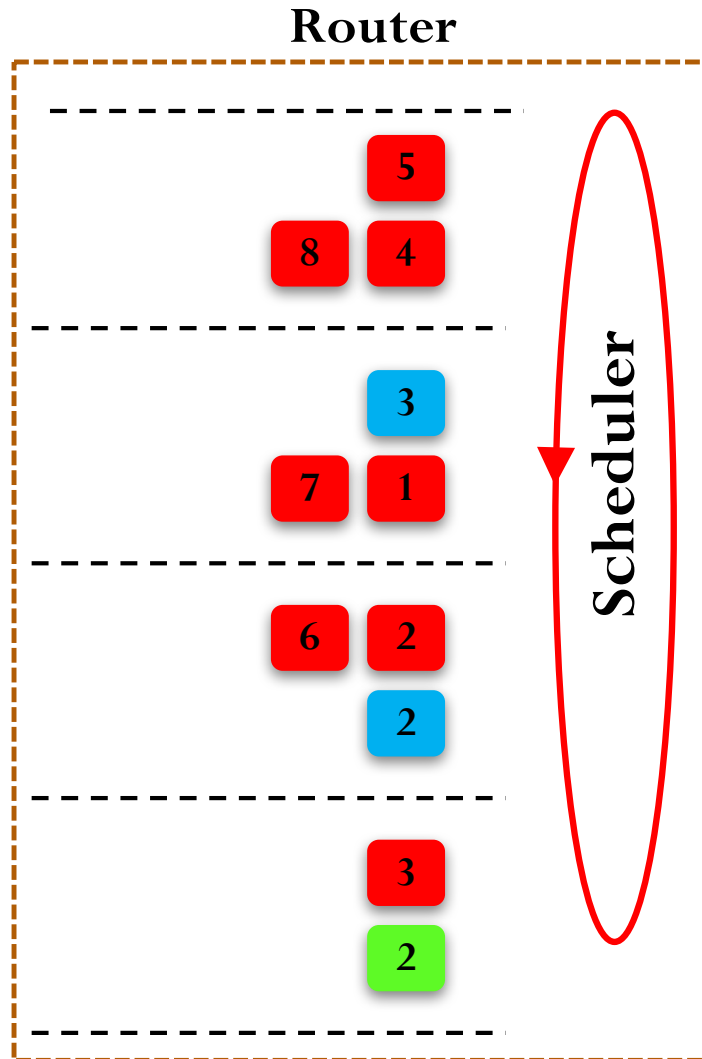
STALL CYCLES				Avg
RR	8	6	11	8.3
Age	4	6	11	7.0
STC				



Ranking order



# STC Scheduling Example



Round Robin



Age



STC



STALL CYCLES				Avg
RR	8	6	11	8.3
Age	4	6	11	7.0
STC	1	3	11	5.0

# Application-Aware Prioritization in NoCs

---

- Das et al., “Application-Aware Prioritization Mechanisms for On-Chip Networks,” MICRO 2009.
  - [https://users.ece.cmu.edu/~omutlu/pub/app-aware-noc\\_micro09.pdf](https://users.ece.cmu.edu/~omutlu/pub/app-aware-noc_micro09.pdf)

## Application-Aware Prioritization Mechanisms for On-Chip Networks

Reetuparna Das<sup>§</sup> Onur Mutlu<sup>†</sup> Thomas Moscibroda<sup>‡</sup> Chita R. Das<sup>§</sup>

<sup>§</sup>Pennsylvania State University  
{rdas,das}@cse.psu.edu

<sup>†</sup>Carnegie Mellon University  
onur@cmu.edu

<sup>‡</sup>Microsoft Research  
moscitho@microsoft.com

# Slack-Based Packet Scheduling

---

- Reetuparna Das, Onur Mutlu, Thomas Moscibroda, and Chita R. Das, **"Aergia: Exploiting Packet Latency Slack in On-Chip Networks"** *Proceedings of the 37th International Symposium on Computer Architecture (ISCA)*, pages 106-116, Saint-Malo, France, June 2010. [Slides \(pptx\)](#)

## Aergia: Exploiting Packet Latency Slack in On-Chip Networks

Reetuparna Das<sup>§</sup>   Onur Mutlu<sup>†</sup>   Thomas Moscibroda<sup>‡</sup>   Chita R. Das<sup>§</sup>  
<sup>§</sup>Pennsylvania State University   <sup>†</sup>Carnegie Mellon University   <sup>‡</sup>Microsoft Research  
{rdas,das}@cse.psu.edu   onur@cmu.edu   moscitho@microsoft.com

# Low-Cost QoS in On-Chip Networks (I)

---

- Boris Grot, Stephen W. Keckler, and Onur Mutlu,  
**"Preemptive Virtual Clock: A Flexible, Efficient, and Cost-effective QOS Scheme for Networks-on-Chip"**  
*Proceedings of the 42nd International Symposium on Microarchitecture (**MICRO**), pages 268-279, New York, NY, December 2009. Slides (pdf)*

## Preemptive Virtual Clock: A Flexible, Efficient, and Cost-effective QOS Scheme for Networks-on-Chip

Boris Grot

Department of Computer Sciences  
The University of Texas at Austin  
{bgrot, skeckler}@cs.utexas.edu

Stephen W. Keckler

<sup>†</sup>Computer Architecture Laboratory (CALCM)  
Carnegie Mellon University  
onur@cmu.edu

Onur Mutlu<sup>†</sup>

# Low-Cost QoS in On-Chip Networks (II)

---

- Boris Grot, Joel Hestness, Stephen W. Keckler, and Onur Mutlu,  
**"Kilo-NOC: A Heterogeneous Network-on-Chip Architecture for Scalability and Service Guarantees"**  
*Proceedings of the 38th International Symposium on Computer Architecture (ISCA), San Jose, CA, June 2011. Slides (pptx)*

## Kilo-NOC: A Heterogeneous Network-on-Chip Architecture for Scalability and Service Guarantees

Boris Grot<sup>1</sup>  
bgrot@cs.utexas.edu

Joel Hestness<sup>1</sup>  
hestness@cs.utexas.edu

Stephen W. Keckler<sup>1,2</sup>  
skeckler@nvidia.com

Onur Mutlu<sup>3</sup>  
onur@cmu.edu

<sup>1</sup>The University of Texas at Austin  
Austin, TX

<sup>2</sup>NVIDIA  
Santa Clara, CA

<sup>3</sup>Carnegie Mellon University  
Pittsburgh, PA

# Computer Architecture

## Lecture 22: Interconnects II

Prof. Onur Mutlu

ETH Zürich

Fall 2017

20 December 2017

We did not cover the following slides in lecture.  
These are for benefit.

# MinBD: Minimally-Buffered Deflection Routing for Energy-Efficient Interconnect

Chris Fallin, Greg Nazario, Xiangyao Yu, Kevin Chang, Rachata  
Ausavarungnirun, and Onur Mutlu,

**"MinBD: Minimally-Buffered Deflection Routing for Energy-Efficient  
Interconnect"**

*Proceedings of the 6th ACM/IEEE International Symposium on Networks on  
Chip (NOCS), Lyngby, Denmark, May 2012. Slides (pptx) (pdf)*

**SAFARI** Carnegie Mellon University



# Bufferless Deflection Routing

---

- **Key idea:** Packets are never buffered in the network. When two packets contend for the same link, one is **deflected**.
- Removing **buffers** yields significant benefits
  - Reduces **power** (CHIPPER: reduces NoC power by 55%)
  - Reduces **die area** (CHIPPER: reduces NoC area by 36%)
- But, at **high network utilization** (load), bufferless deflection routing causes **unnecessary link & router traversals**
  - Reduces **network throughput** and application performance
  - Increases **dynamic power**
- **Goal:** Improve **high-load performance** of low-cost deflection networks by reducing the deflection rate.

# Outline: This Talk

---

- **Motivation**
- **Background:** Bufferless Deflection Routing
- **MinBD:** Reducing Deflections
  - Addressing Link Contention
  - Addressing the Ejection Bottleneck
  - Improving Deflection Arbitration
- **Results**
- **Conclusions**

# Outline: This Talk

---

- Motivation
- **Background:** Bufferless Deflection Routing
- **MinBD:** Reducing Deflections
  - Addressing Link Contention
  - Addressing the Ejection Bottleneck
  - Improving Deflection Arbitration
- Results
- Conclusions

# Issues in Bufferless Deflection Routing

---

- **Correctness:** Deliver all packets without **livelock**
  - **CHIPPER<sup>1</sup>: Golden Packet**
  - Globally prioritize one packet until delivered
- **Correctness:** Reassemble packets without **deadlock**
  - **CHIPPER<sup>1</sup>: Retransmit-Once**
- **Performance:** Avoid performance degradation at **high load**
  - **MinBD**

---

**SAFARI**  
<sup>1</sup> Fallin et al., “CHIPPER: A Low-complexity Bufferless Deflection Router”, HPCA

# Key Performance Issues

---

- 1. Link contention:** no buffers to hold traffic → any link contention causes a deflection  
→ use side buffers
- 2. Ejection bottleneck:** only one flit can eject per router per cycle → simultaneous arrival causes deflection  
→ eject up to 2 flits/cycle
- 3. Deflection arbitration:** practical (fast) deflection arbiters deflect unnecessarily  
→ new priority scheme (silver flit)

# Outline: This Talk

---

- **Motivation**
- **Background:** Bufferless Deflection Routing
- **MinBD:** Reducing Deflections
  - Addressing Link Contention
  - Addressing the Ejection Bottleneck
  - Improving Deflection Arbitration
- **Results**
- **Conclusions**

# Outline: This Talk

---

- **Motivation**
- **Background:** Bufferless Deflection Routing
- **MinBD:** Reducing Deflections
  - Addressing Link Contention
  - Addressing the Ejection Bottleneck
  - Improving Deflection Arbitration
- **Results**
- **Conclusions**

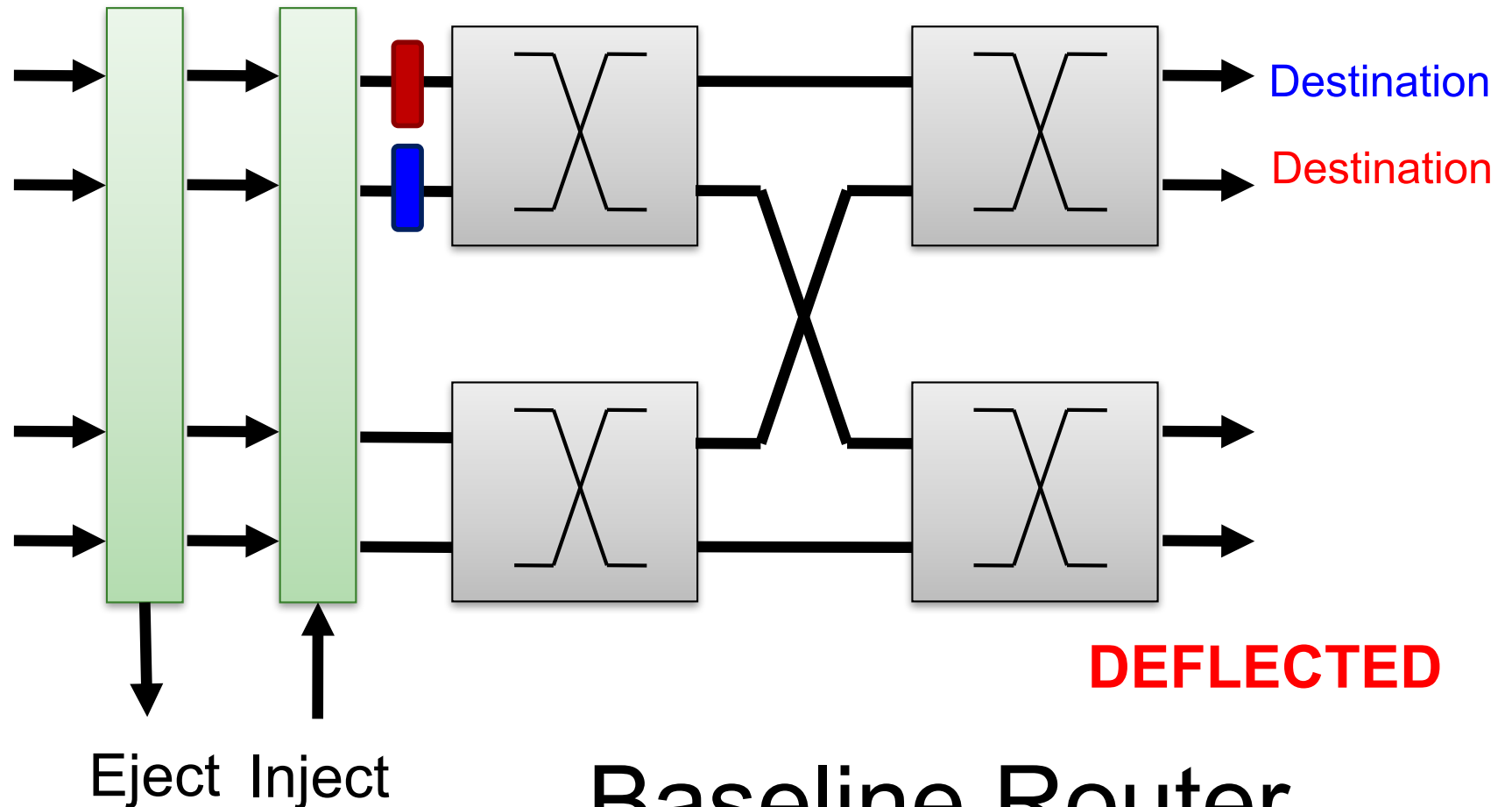
# Addressing Link Contention

---

- **Problem 1:** Any link contention causes a deflection
- **Buffering** a flit can avoid deflection on contention
- But, **input buffers** are expensive:
  - All flits are buffered on every hop → **high dynamic energy**
  - Large buffers necessary → **high static energy** and **large area**
- **Key Idea 1:** add a **small buffer** to a bufferless deflection router to buffer **only** flits that **would have been deflected**

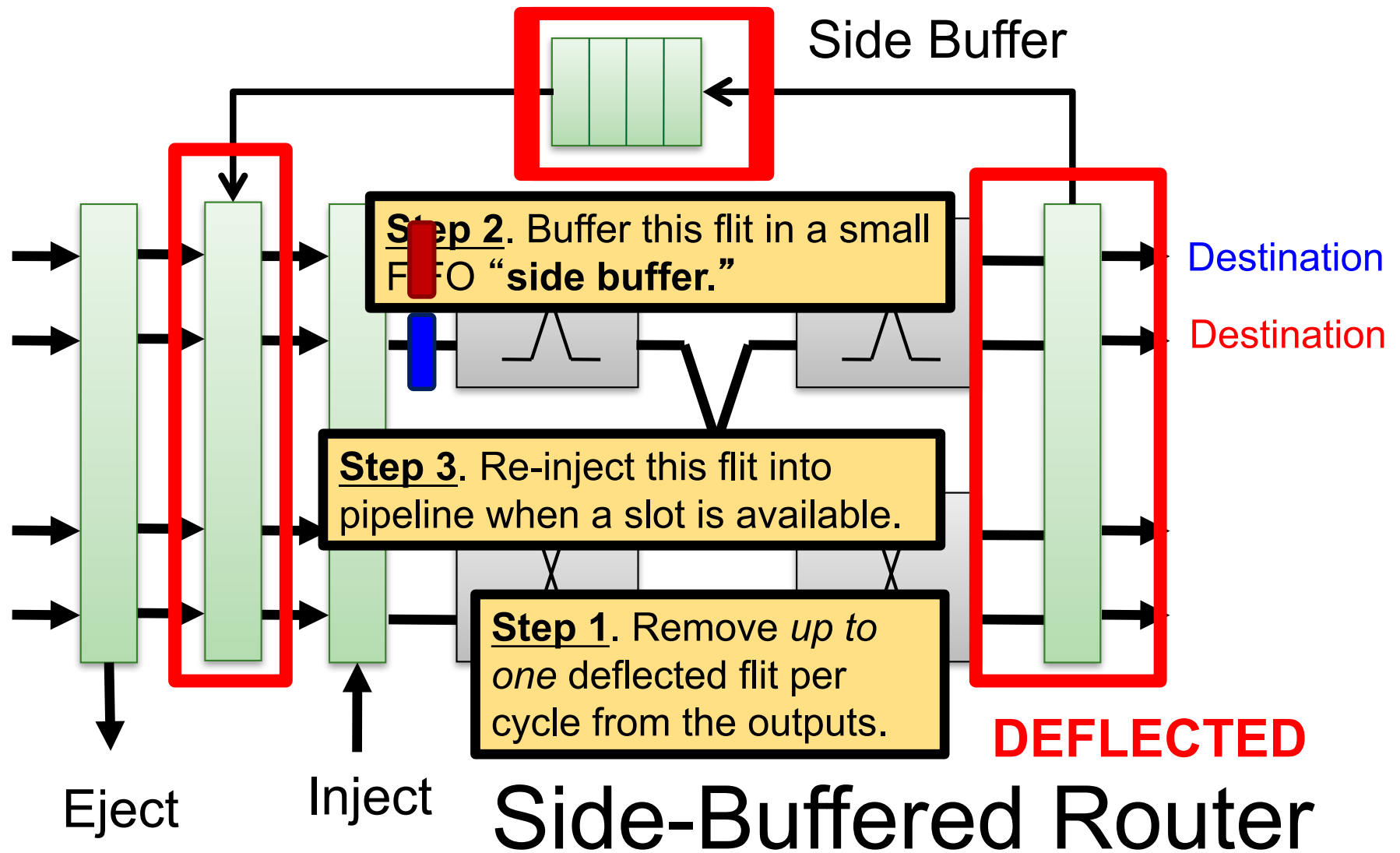


# How to Buffer Deflected Flits



Baseline Router

# How to Buffer Deflected Flits



# Why Could A Side Buffer Work Well?

---

- Buffer some flits and deflect other flits at **per-flit level**
  - Relative to **bufferless routers**, **deflection rate reduces** (need not deflect all contending flits)
    - 4-flit buffer reduces deflection rate by **39%**
  - Relative to **buffered routers**, **buffer is more efficiently used** (need not buffer all flits)
    - similar performance with **25%** of buffer space

# Outline: This Talk

---

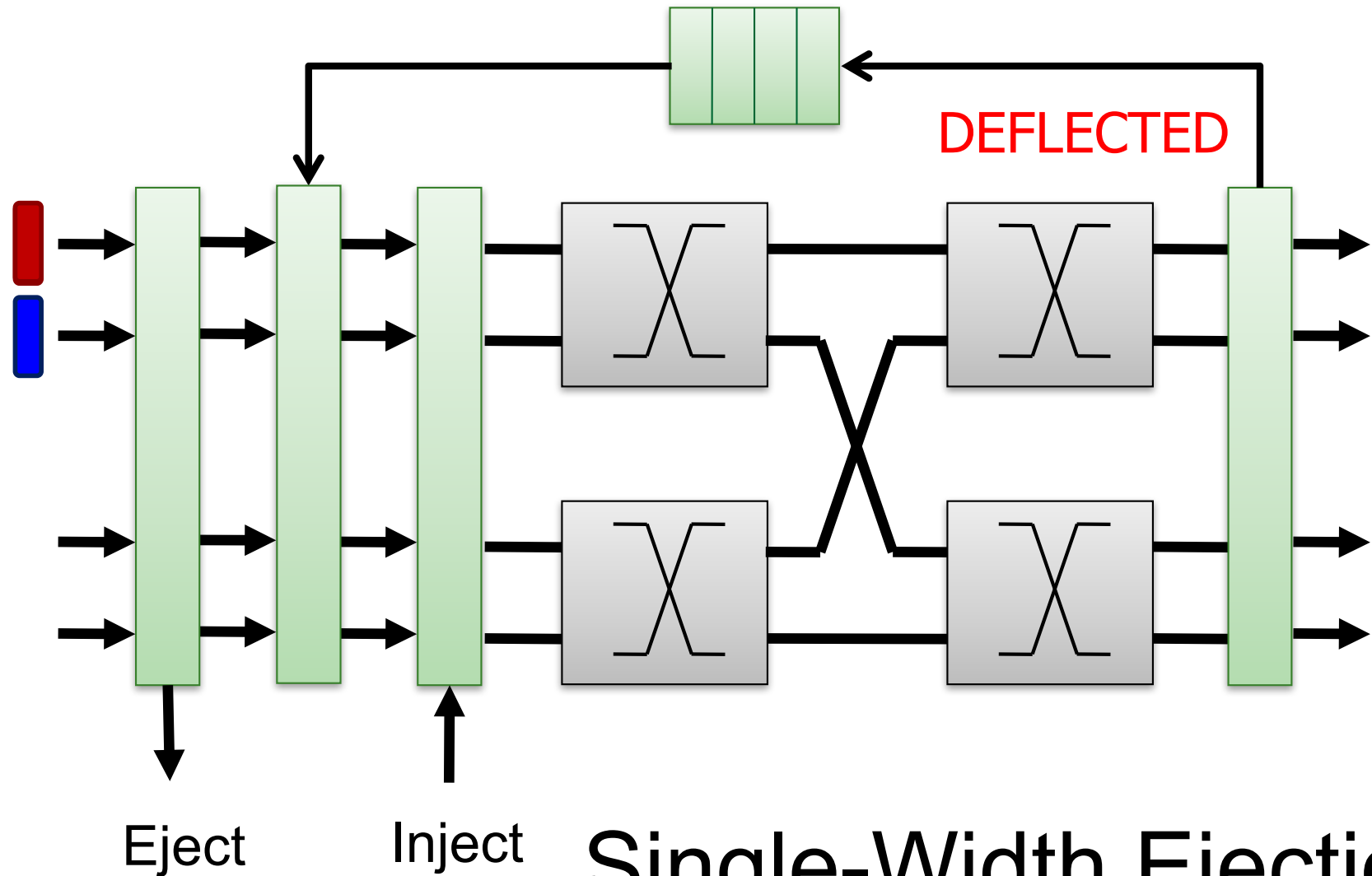
- **Motivation**
- **Background:** Bufferless Deflection Routing
- **MinBD:** Reducing Deflections
  - Addressing Link Contention
  - Addressing the Ejection Bottleneck
  - Improving Deflection Arbitration
- **Results**
- **Conclusions**

# Addressing the Ejection Bottleneck

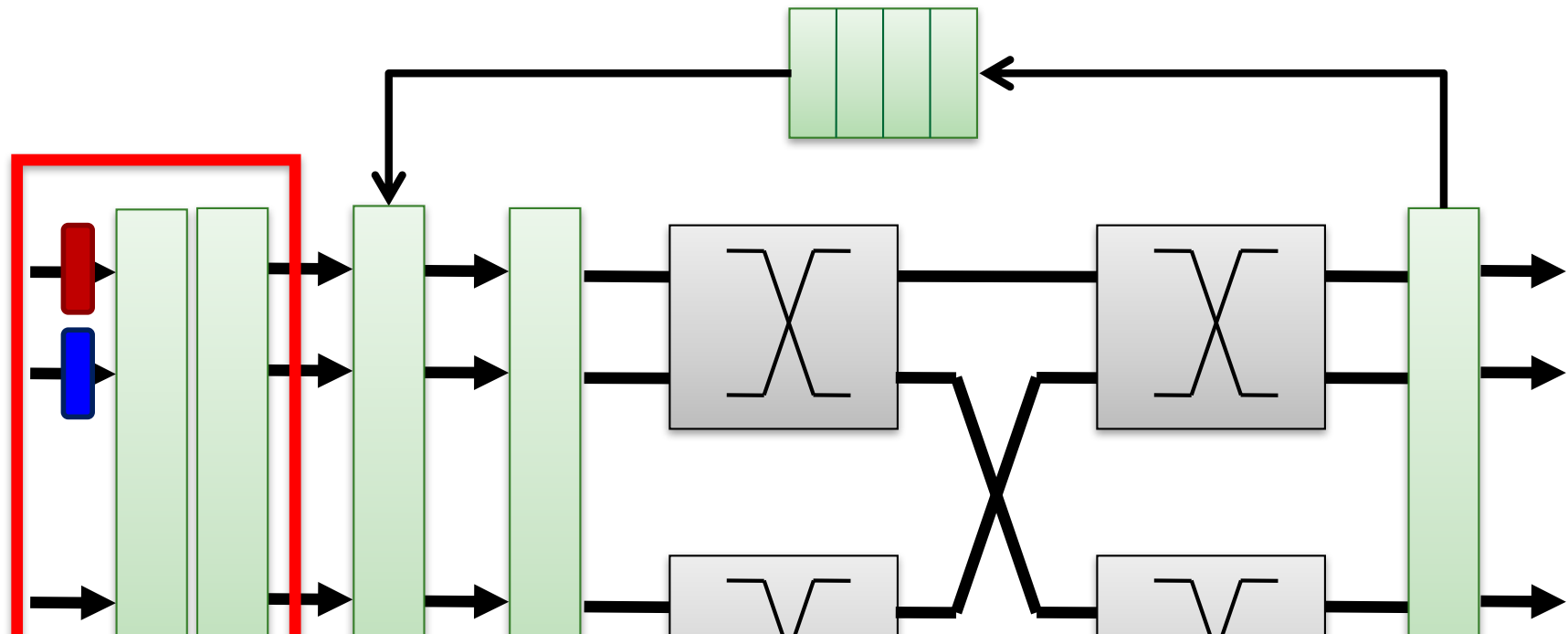
---

- **Problem 2:** Flits deflect unnecessarily because only one flit can **eject** per router per cycle
- In 20% of all ejections,  $\geq 2$  flits could have ejected
  - all but one flit must **deflect and try again**
  - these deflected flits cause additional contention
- Ejection width of 2 flits/cycle reduces **deflection rate 21%**
- **Key idea 2:** Reduce deflections due to a single-flit ejection port by allowing **two flits** to eject per cycle

# Addressing the Ejection Bottleneck



# Addressing the Ejection Bottleneck



For fair comparison, **baseline routers** have dual-width ejection for perf. (not power/area)

Eject

Inject

Dual-Width Ejection

# Outline: This Talk

---

- **Motivation**
- **Background:** Bufferless Deflection Routing
- **MinBD:** Reducing Deflections
  - Addressing Link Contention
  - Addressing the Ejection Bottleneck
  - Improving Deflection Arbitration
- **Results**
- **Conclusions**



# Improving Deflection Arbitration

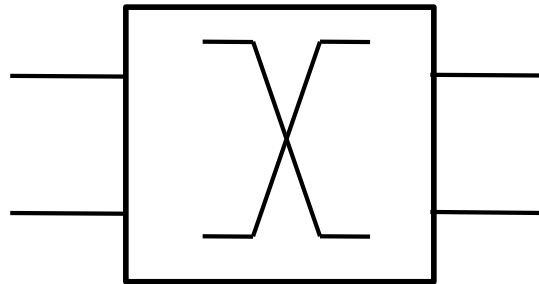
---

- **Problem 3:** Deflections occur unnecessarily because fast arbiters must use simple priority schemes
- Age-based priorities (several past works): full priority order gives fewer deflections, but requires slow arbiters
- State-of-the-art deflection arbitration (Golden Packet & two-stage permutation network)
  - Prioritize one packet globally (**ensure forward progress**)
  - Arbitrate other flits randomly (**fast critical path**)
- Random common case leads to uncoordinated arbitration

# Fast Deflection Routing Implementation

---

- Let's route in a two-input router first:



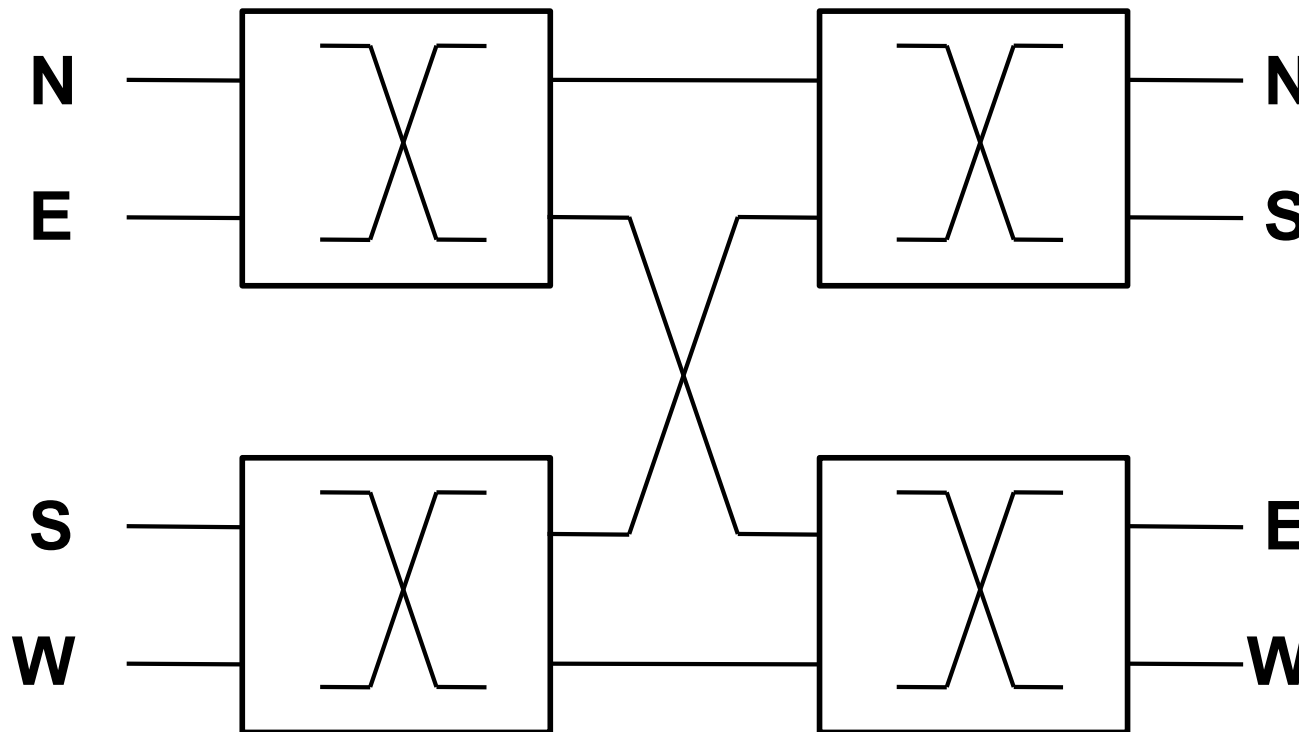
- **Step 1:** pick a “winning” flit (Golden Packet, else random)
- **Step 2:** steer the winning flit to its desired output and deflect other flit

➔ **Highest-priority flit always routes to destination**

# Fast Deflection Routing with Four Inputs

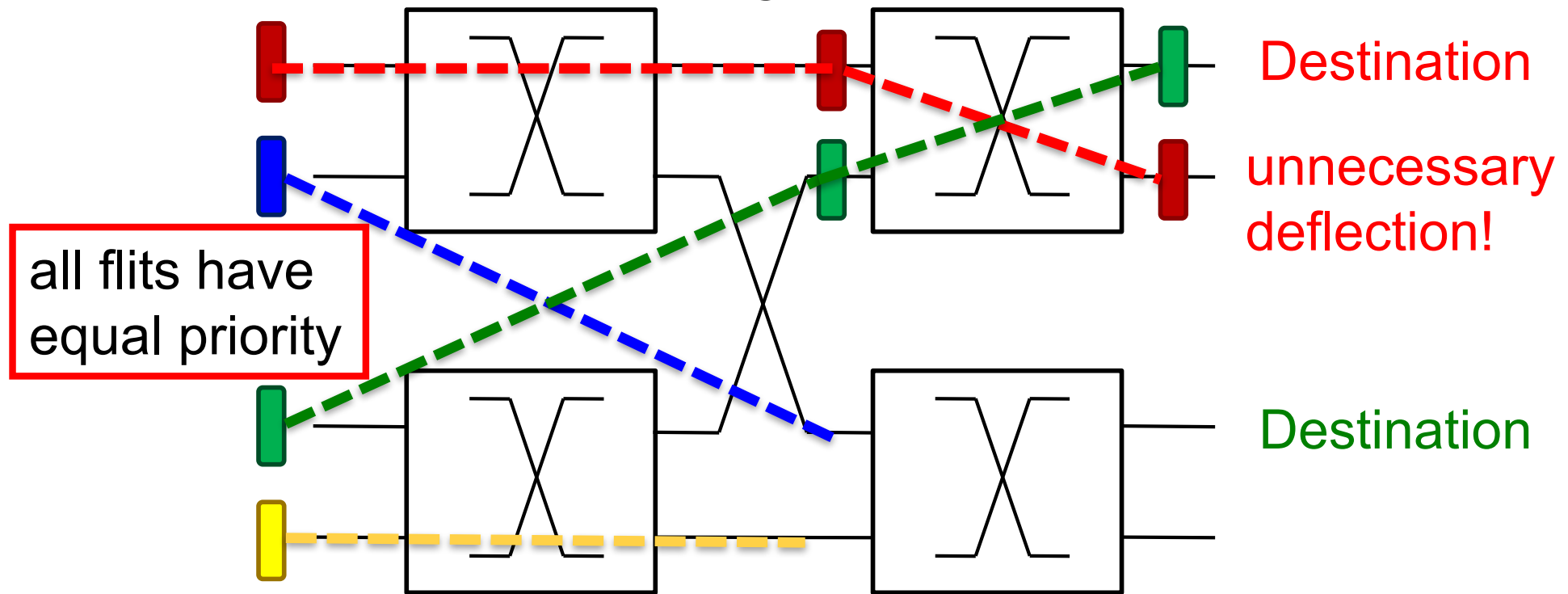
---

- Each block makes decisions **independently**
  - **Deflection is a distributed decision**



# Unnecessary Deflections in Fast Arbiters

- How does lack of coordination cause unnecessary deflections?
  1. No flit is golden (pseudorandom arbitration)
  2. Red flit wins at first stage
  3. Green flit loses at first stage (must be deflected now)
  4. Red flit loses at second stage; Red and Green are deflected



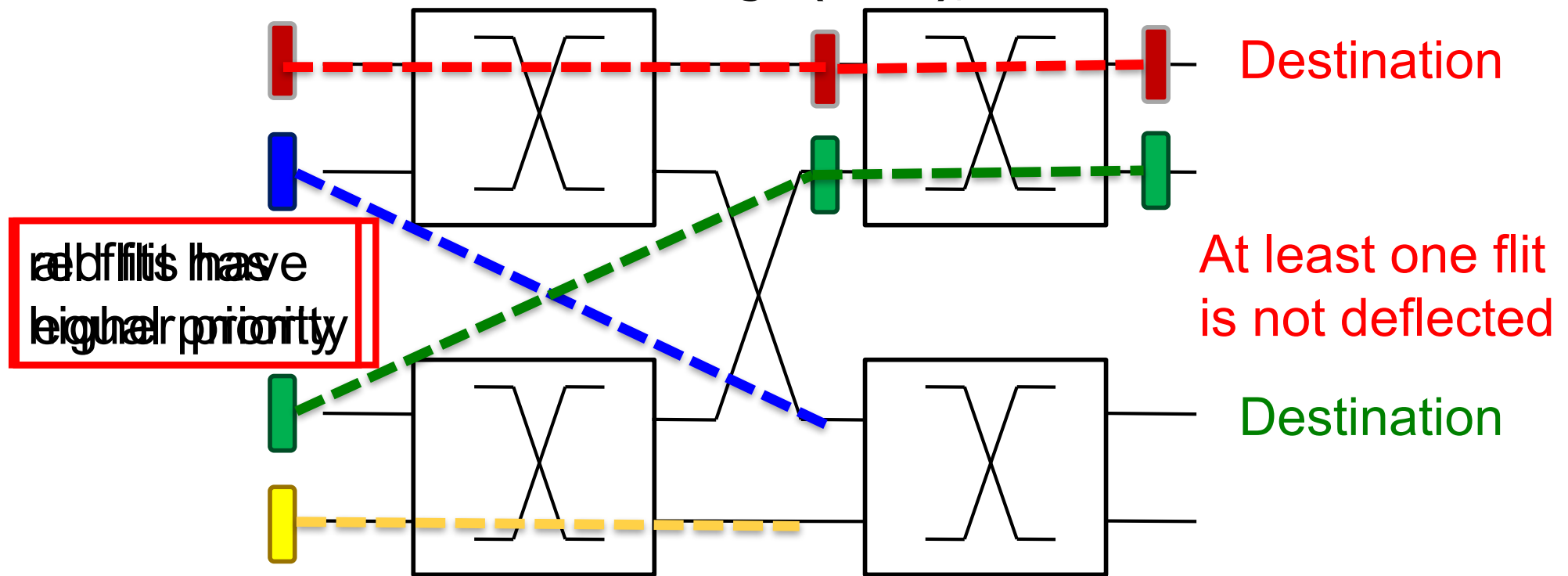
# Improving Deflection Arbitration

---

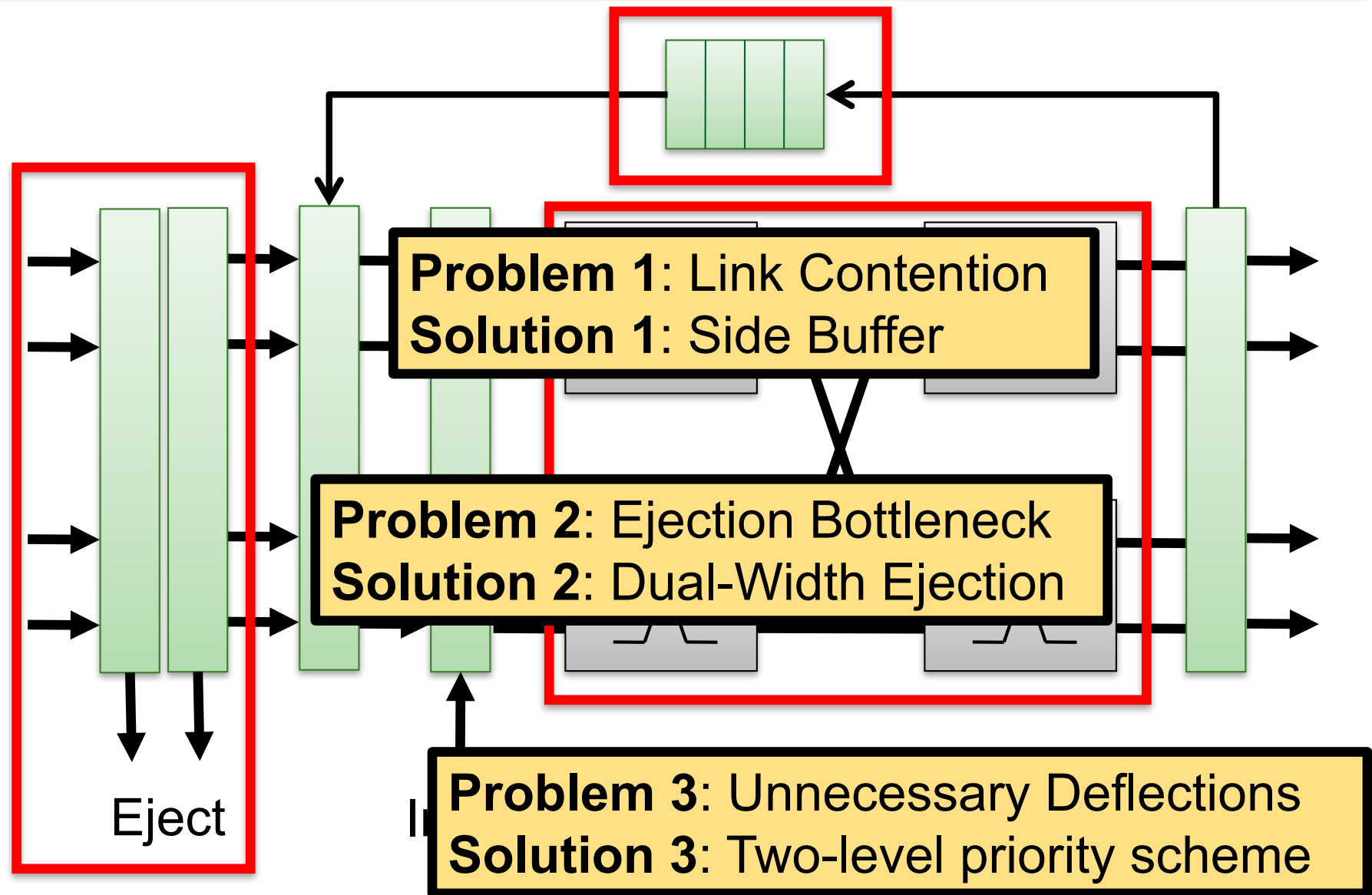
- **Key idea 3: Add a priority level** and prioritize one flit to ensure at least **one flit is not deflected in each cycle**
- **Highest priority:** one **Golden Packet** in network
  - Chosen in static round-robin schedule
  - **Ensures correctness**
- **Next-highest priority:** one **silver flit** per router per cycle
  - Chosen pseudo-randomly & local to one router
  - **Enhances performance**

# Adding A Silver Flit

- Randomly picking a silver flit ensures **one flit is not deflected**
  - No flit is golden but Red flit is silver
  - Red flit wins at first stage (silver)
  - Green flit is deflected at first stage
  - Red flit wins at second stage (silver); not deflected



# Minimally-Buffered Deflection Router



# Outline: This Talk

---

- **Motivation**
- **Background:** Bufferless Deflection Routing
- **MinBD:** Reducing Deflections
  - Addressing Link Contention
  - Addressing the Ejection Bottleneck
  - Improving Deflection Arbitration



# Outline: This Talk

---

- **Motivation**
- **Background:** Bufferless Deflection Routing
- **MinBD:** Reducing Deflections
  - Addressing Link Contention
  - Addressing the Ejection Bottleneck
  - Improving Deflection Arbitration
- **Results**
- **Conclusions**

# Methodology: Simulated System

---

## ■ **Chip Multiprocessor Simulation**

- ❑ **64-core** and **16-core** models
- ❑ **Closed-loop** core/cache/NoC cycle-level model
- ❑ Directory cache coherence protocol (SGI Origin-based)
- ❑ 64KB L1, perfect L2 (stresses interconnect), XOR-mapping
- ❑ Performance metric: **Weighted Speedup**  
(similar conclusions from network-level latency)
- ❑ Workloads: multiprogrammed SPEC CPU2006
  - 75 randomly-chosen workloads
  - Binned into network-load categories by average injection rate

# Methodology: Routers and Network

---

- **Input-buffered** virtual-channel router
  - ❑ 8 VCs, 8 flits/VC [[Buffered\(8,8\)](#)]: large buffered router
  - ❑ 4 VCs, 4 flits/VC [[Buffered\(4,4\)](#)]: typical buffered router
  - ❑ 4 VCs, 1 flit/VC [[Buffered\(4,1\)](#)]: smallest deadlock-free router
  - ❑ All power-of-2 buffer sizes up to (8, 8) for perf/power sweep
- **Bufferless deflection** router: **CHIPPER**<sup>1</sup>
- **Bufferless-buffered hybrid** router: **AFC**<sup>2</sup>
  - ❑ Has input buffers and deflection routing logic
  - ❑ Performs coarse-grained (multi-cycle) mode switching
- **Common parameters**
  - ❑ 2-cycle router latency, 1-cycle link latency
  - ❑ 2D-mesh topology (16-node: 4x4; 64-node: 8x8)
  - ❑ Dual ejection assumed for baseline routers (for perf. only)

---

<sup>1</sup>Fallin et al., “CHIPPER: A Low-complexity Bufferless Deflection Router”, HPCA 2011.

<sup>2</sup>Jafri et al., “Adaptive Flow Control for Robust Performance and Energy”, MICRO 2010.

# Methodology: Power, Die Area, Crit. Path

---

## ■ **Hardware modeling**

- Verilog models for CHIPPER, MinBD, buffered control logic
  - Synthesized with commercial 65nm library
- ORION 2.0 for datapath: crossbar, muxes, buffers and links

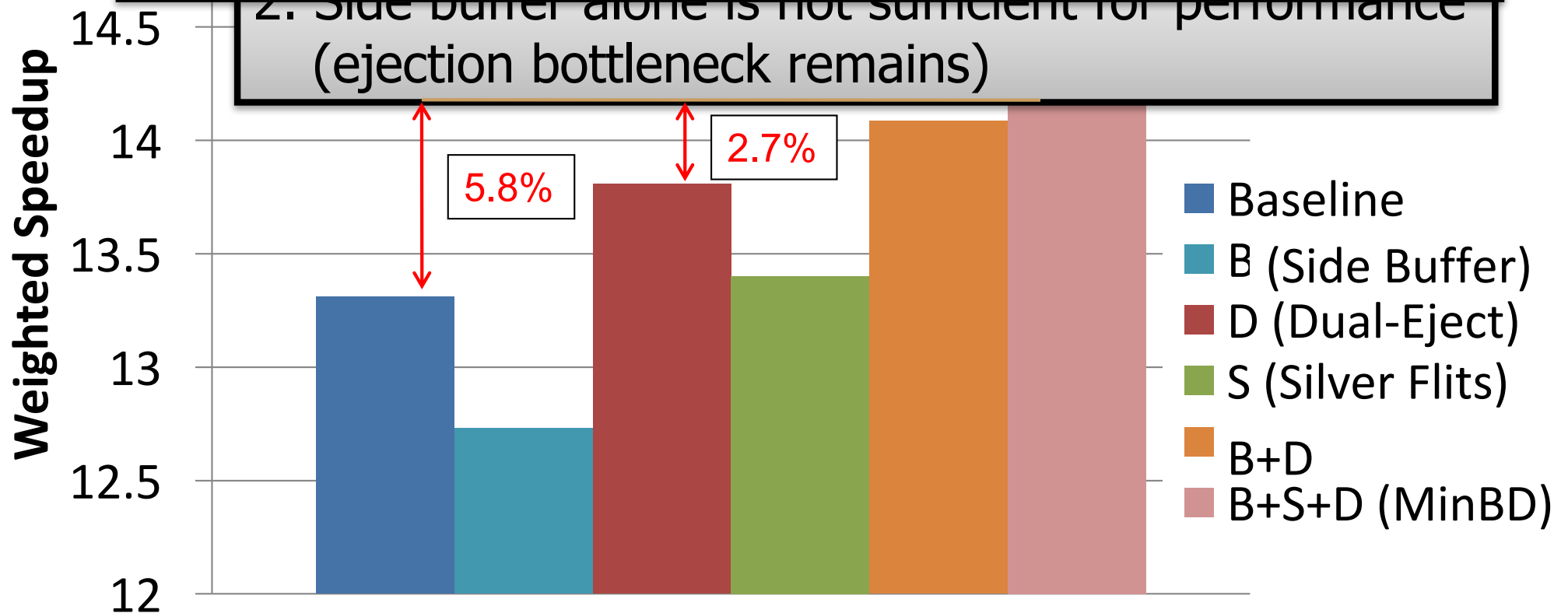
## ■ **Power**

- Static and dynamic power from hardware models
- Based on event counts in cycle-accurate simulations
- Broken down into buffer, link, other

# Reduced Deflections & Improved Perf.

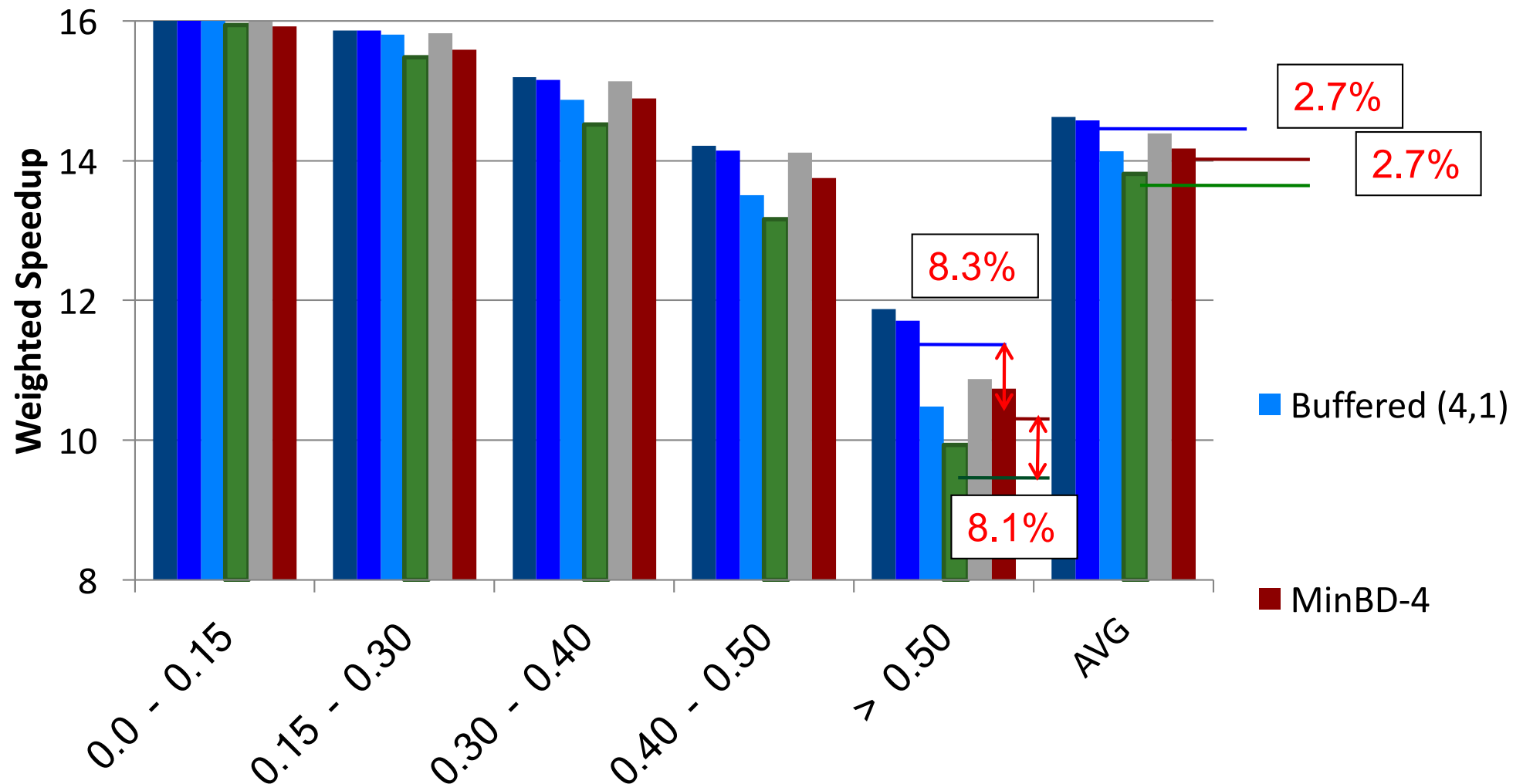
3. Overall, **5.8%** over baseline, **2.7%** over dual-eject by reducing deflections **64%** / **54%**

2. Side buffer alone is not sufficient for performance (ejection bottleneck remains)



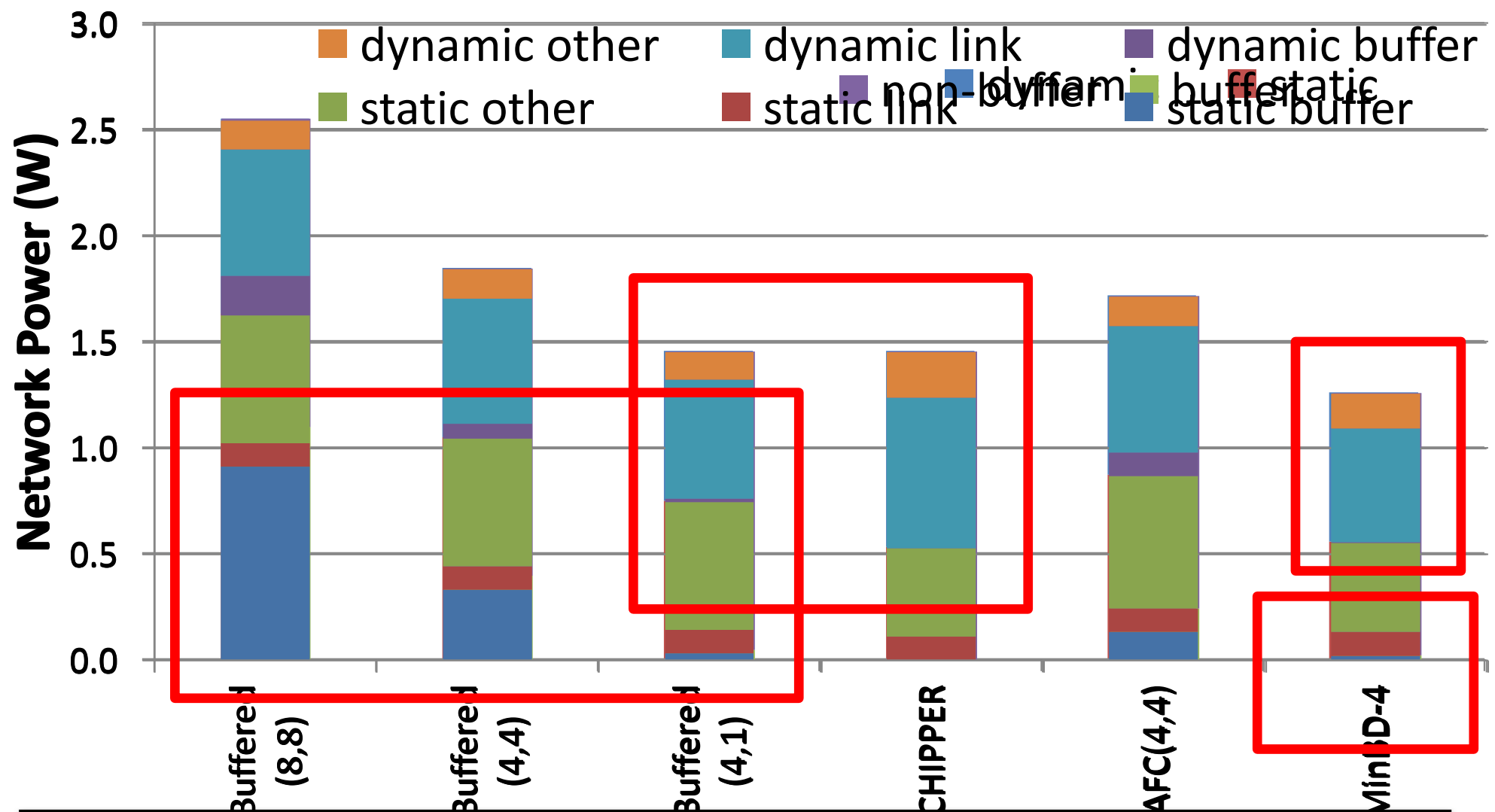
Deflection Rate	28%	17%	22%	27%	11%	10%
-----------------	-----	-----	-----	-----	-----	-----

# Overall Performance Results



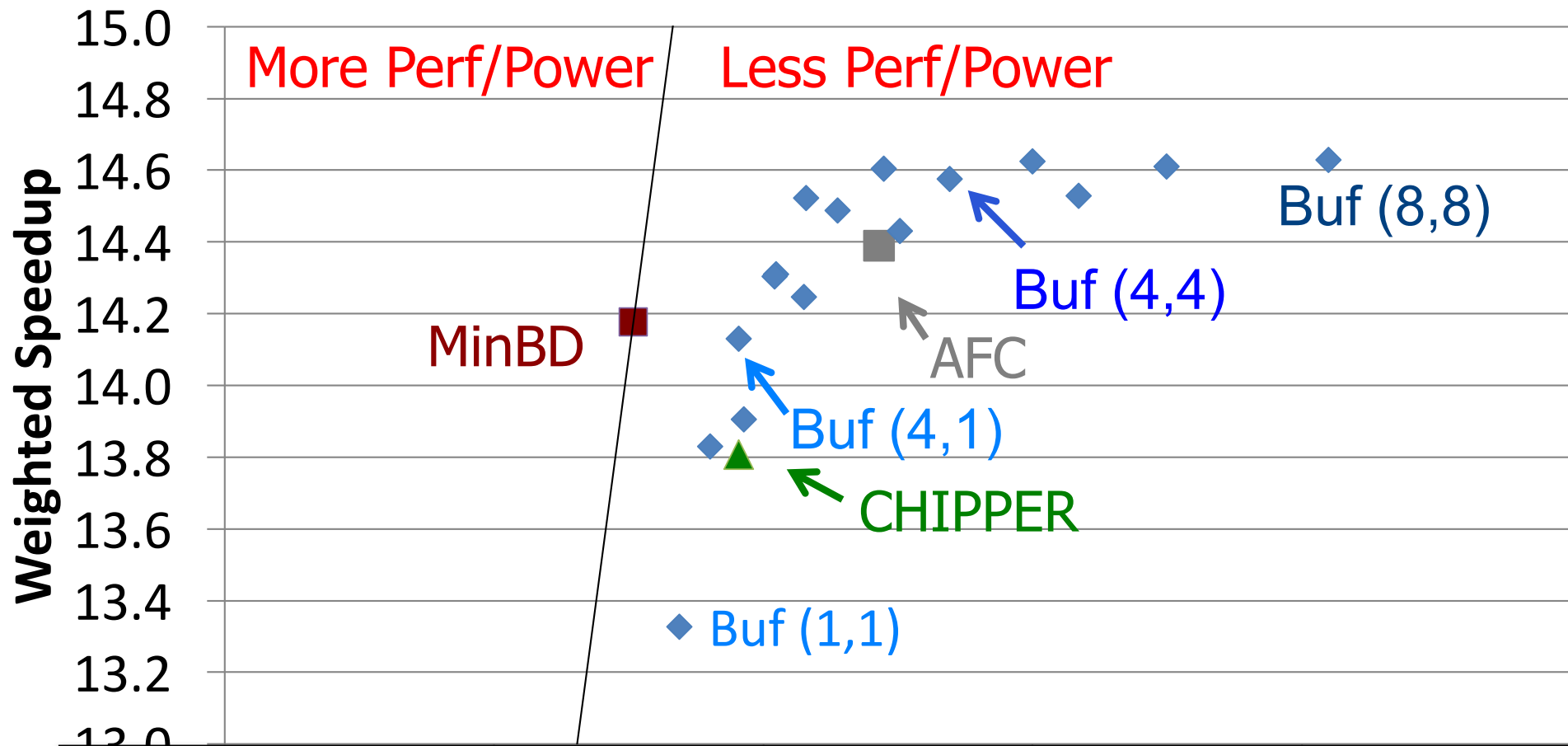
- Similar perf. to Buffered (4,1) @ 25% of buffering space
- Within **2.7%** of Buffered (4,4) (**8.3%** at high load)

# Overall Power Results



- Dynamic power increases with deflection routing
- Dynamic power reduces in MinBD relative to CHIPPER

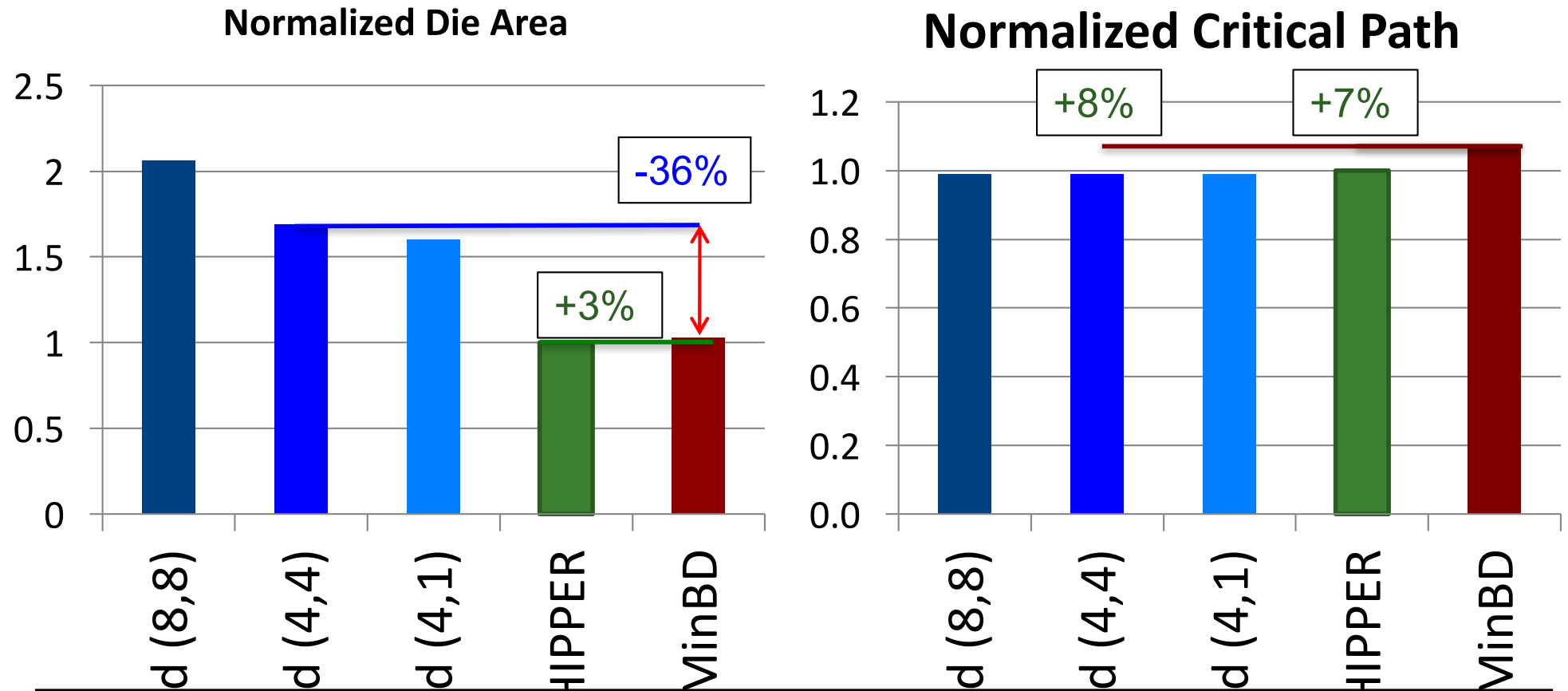
# Performance-Power Spectrum



- Most **energy-efficient** (perf/watt) of any evaluated network router design



# Die Area and Critical Path



- Only **3%** area increase over CHIPPER (4-flit buffer)
- Increases by **7%** over CHIPPER, **8%** over Buffered (4,4)

# Conclusions

---

- Bufferless deflection routing offers **reduced power & area**
  - But, high deflection rate hurts **performance at high load**
  - **MinBD** (Minimally-Buffered Deflection Router) introduces:
    - **Side buffer** to hold **only** flits that would have been deflected
    - **Dual-width ejection** to address ejection bottleneck
    - **Two-level prioritization** to avoid unnecessary deflections
  - MinBD yields **reduced power (31%) & reduced area (36%)** relative to **buffered** routers
  - MinBD yields **improved performance (8.1% at high load)** relative to **bufferless** routers → closes half of perf. gap
  - MinBD has the **best energy efficiency** of all evaluated designs with **competitive performance**
-

# More Readings

---

- Studies of congestion and congestion control in on-chip vs. internet-like networks
- George Nychis, Chris Fallin, Thomas Moscibroda, Onur Mutlu, and Srinivasan Seshan,  
**"On-Chip Networks from a Networking Perspective: Congestion and Scalability in Many-core Interconnects"**  
*Proceedings of the 2012 ACM SIGCOMM Conference (**SIGCOMM**), Helsinki, Finland, August 2012. Slides (pptx)*
- George Nychis, Chris Fallin, Thomas Moscibroda, and Onur Mutlu,  
**"Next Generation On-Chip Networks: What Kind of Congestion Control Do We Need?"**  
*Proceedings of the 9th ACM Workshop on Hot Topics in Networks (**HOTNETS**), Monterey, CA, October 2010. Slides (ppt) (key)*

# HAT: Heterogeneous Adaptive Throttling for On-Chip Networks

Kevin Chang, Rachata Ausavarungnirun, Chris Fallin, and Onur Mutlu,

"HAT: Heterogeneous Adaptive Throttling for On-Chip Networks"

*Proceedings of the 24th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, New York, NY, October 2012. Slides  
(pptx) (pdf)

Carnegie Mellon University

**SAFARI**

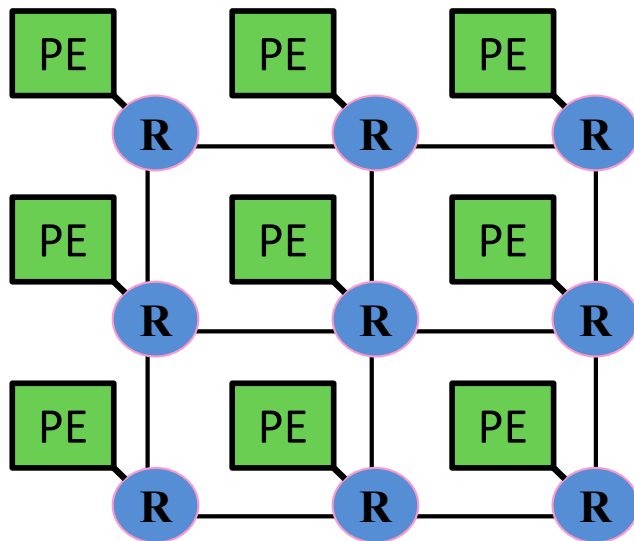
# Executive Summary

- **Problem:** Packets contend in on-chip networks (NoCs), causing congestion, thus reducing performance
- **Observations:**
  - 1) Some applications are more sensitive to network latency than others
  - 2) Applications must be throttled differently to achieve peak performance
- **Key Idea:** Heterogeneous Adaptive Throttling (HAT)
  - 1) Application-aware source throttling
  - 2) Network-load-aware throttling rate adjustment
- **Result:** Improves performance and energy efficiency over state-of-the-art source throttling policies

# Outline

- **Background and Motivation**
- Mechanism
- Prior Works
- Results

# On-Chip Networks



Router

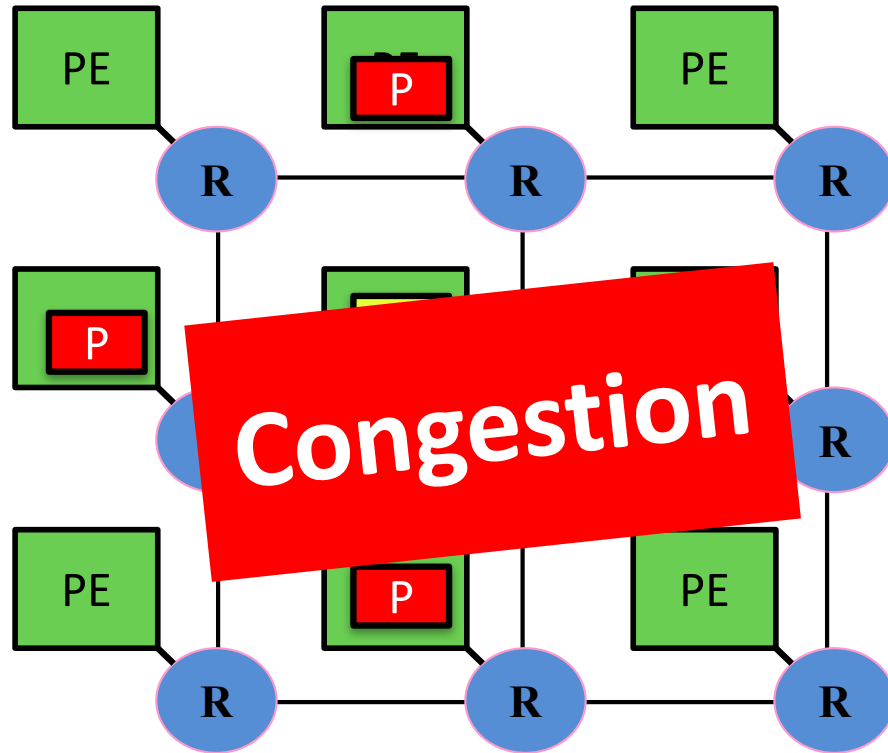


Processing Element

(Cores, L2 Banks, Memory Controllers, etc)

- Connect **cores, caches, memory controllers, etc**
- **Packet switched**
- **2D mesh:** Most commonly used topology
- Primarily serve **cache misses** and **memory requests**
- **Router designs**
  - Buffered: **Input buffers** to hold contending packets
  - Bufferless: **Misroute (deflect)** contending packets

# Network Congestion Reduces Performance



Limited shared resources  
(buffers and links)

- **Design constraints: power, chip area, and timing**

**Network congestion:**

↓ Network throughput

↓ Application performance





# Goal

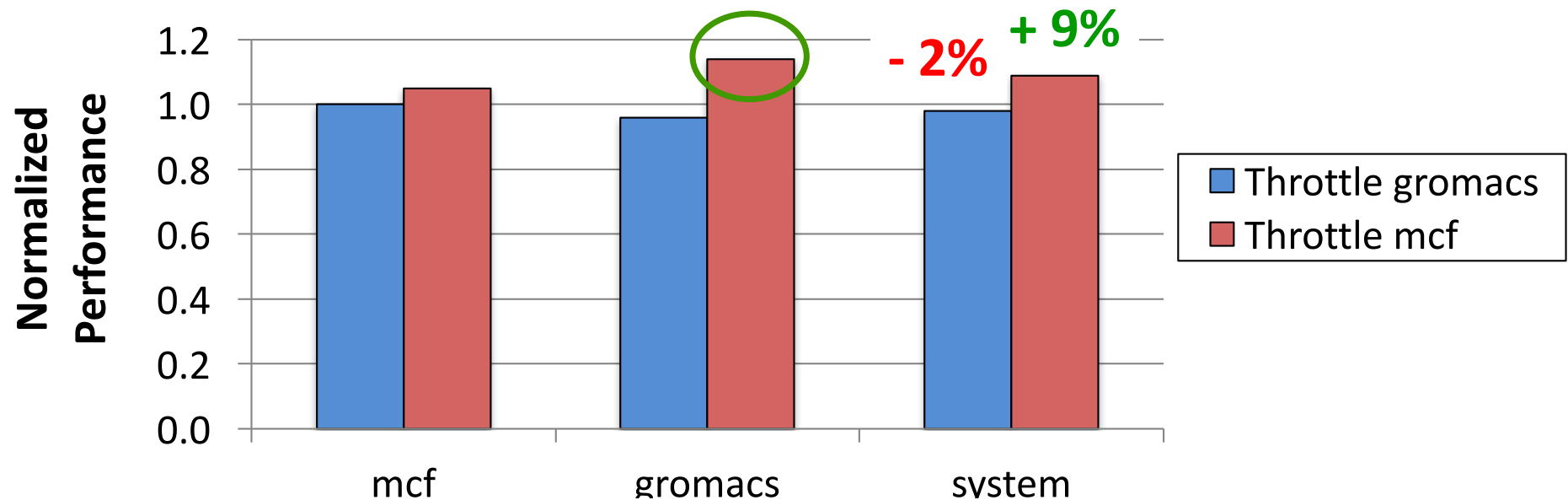
- Improve performance in a highly congested NoC
- Reducing network load decreases network congestion, hence improves performance
- **Approach: source throttling to reduce network load**
  - Temporarily delay new traffic injection
- **Naïve mechanism: throttle every single node**

# Key Observation #1

Different applications respond differently to changes in **network latency**

gromacs: network-**non**-intensive

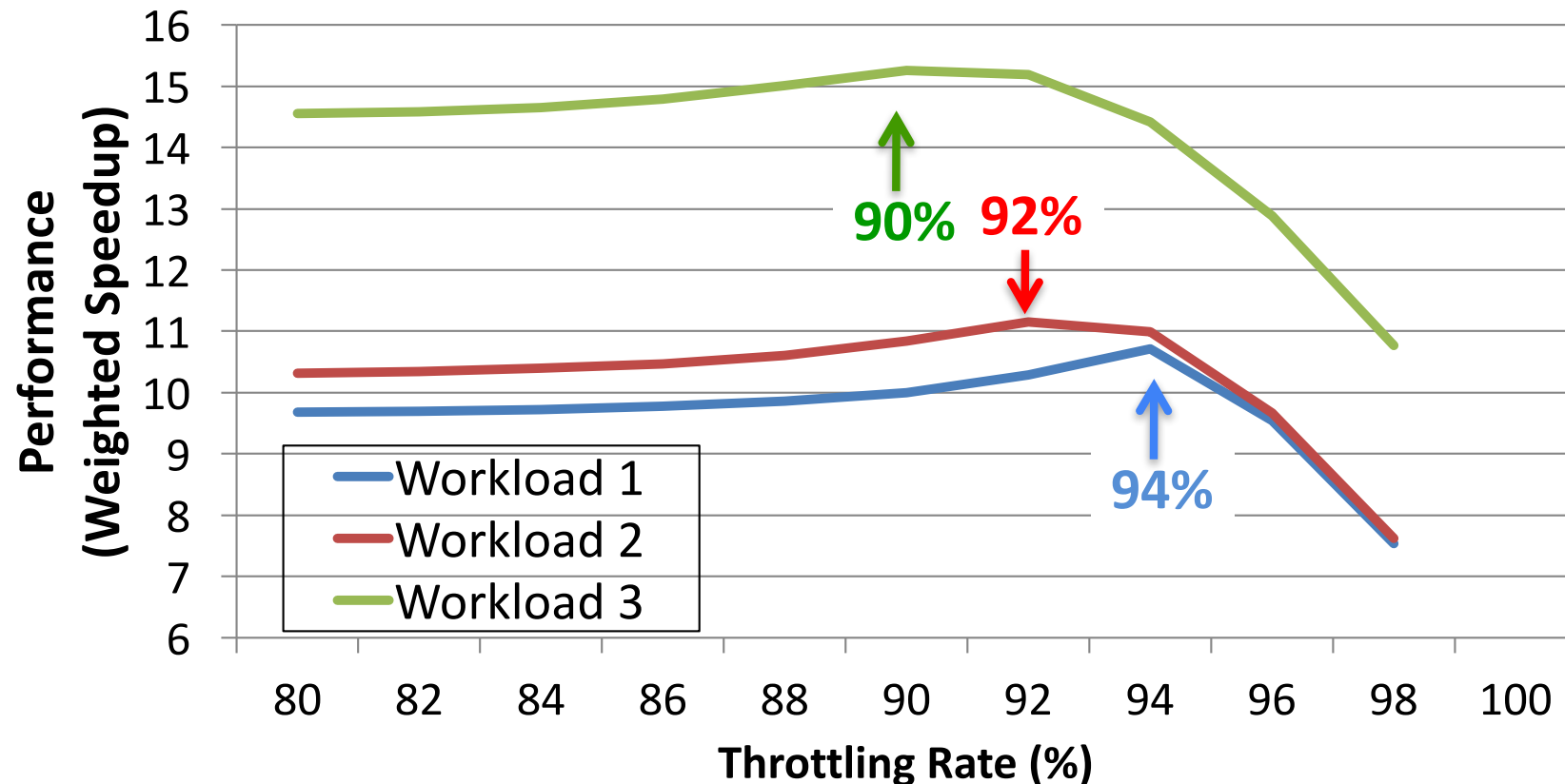
mcf: network-**intensive**



Throttling **network-intensive** applications benefits system performance more

# Key Observation #2

Different workloads achieve peak performance at different throttling rates



Dynamically adjusting throttling rate yields better performance than a single static rate

# Outline

- Background and Motivation
- **Mechanism**
- Prior Works
- Results

# Heterogeneous Adaptive Throttling (HAT)

## 1. Application-aware throttling:

Throttle **network-intensive** applications that interfere with **network-non-intensive** applications

## 2. Network-load-aware throttling rate adjustment:

**Dynamically** adjusts throttling rate to adapt to different workloads

# Heterogeneous Adaptive Throttling (HAT)

## 1. Application-aware throttling:

Throttle **network-intensive** applications that interfere with **network-non-intensive** applications

## 2. Network-load-aware throttling rate adjustment:

**Dynamically** adjusts throttling rate to adapt to different workloads

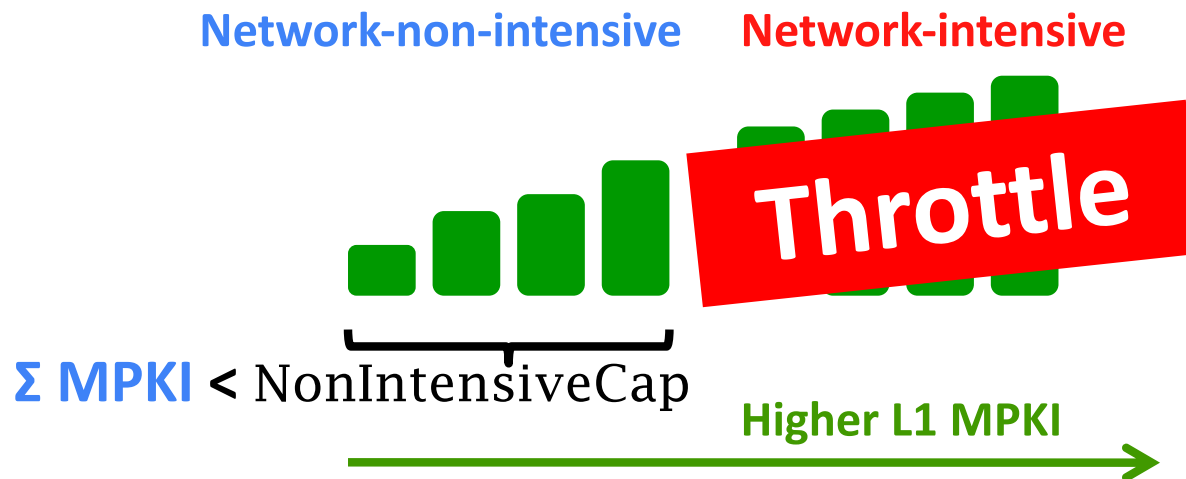
# Application-Aware Throttling

## 1. Measure Network Intensity

Use **L1 MPKI** (misses per thousand instructions) to estimate network intensity

## 2. Classify Application

Sort applications by L1 MPKI



## 3. Throttle network-intensive applications

# Heterogeneous Adaptive Throttling (HAT)

## 1. Application-aware throttling:

Throttle **network-intensive** applications that interfere with **network-non-intensive** applications

## 2. Network-load-aware throttling rate adjustment:

**Dynamically** adjusts throttling rate to adapt to different workloads



# Dynamic Throttling Rate Adjustment

- For a given **network design**, peak performance tends to occur at a **fixed network load point**
- **Dynamically** adjust throttling rate to achieve that network load point

# Dynamic Throttling Rate Adjustment

- **Goal:** maintain network load at a peak performance point

1. Measure network load
2. Compare and adjust throttling rate

If **network load** > **peak point**:

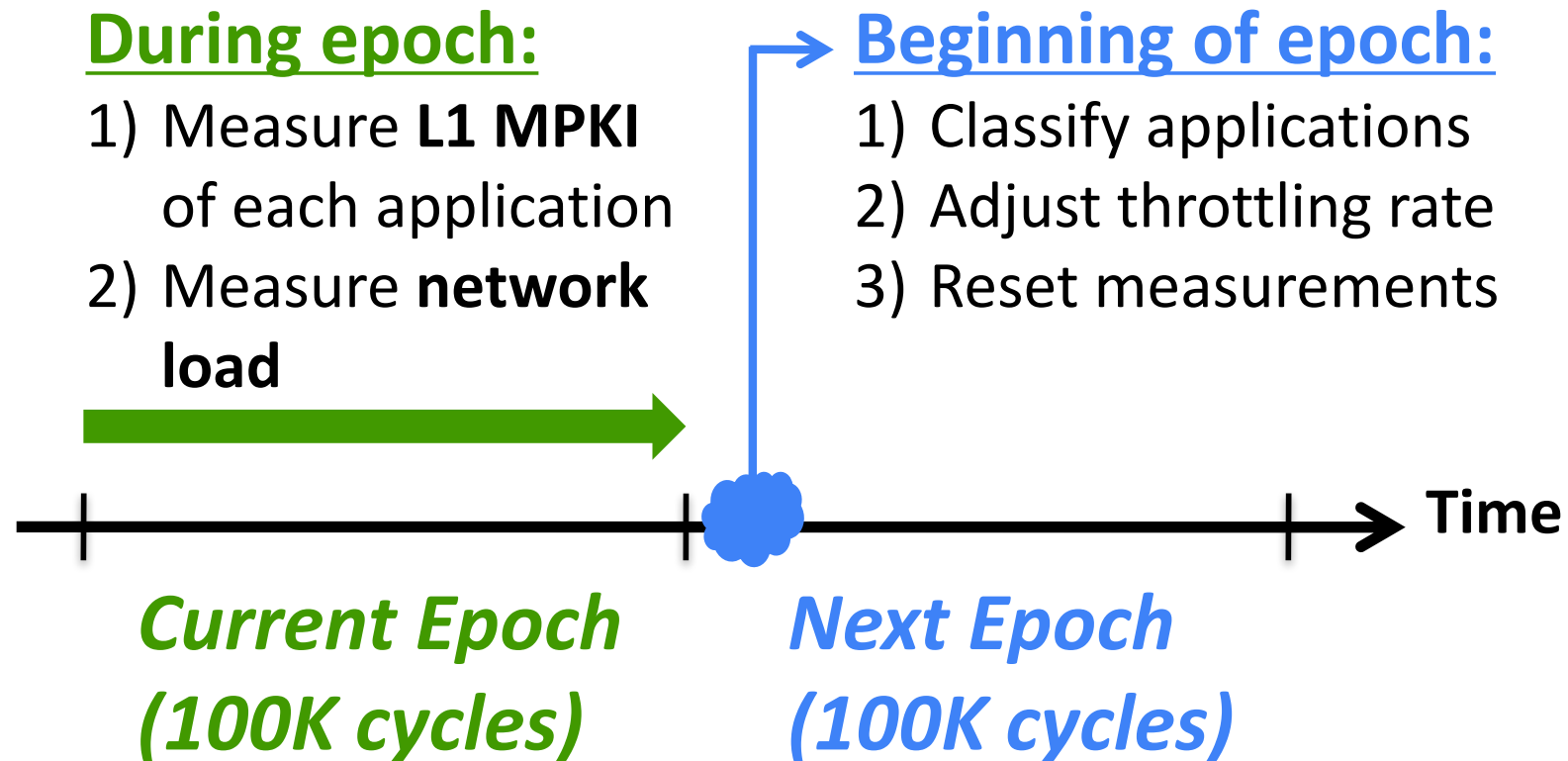
    Increase throttling rate

elif **network load** ≤ **peak point**:

    Decrease throttling rate

# Epoch-Based Operation

- Continuous **HAT** operation is expensive
- **Solution:** performs **HAT** at epoch granularity



# Outline

- Background and Motivation
- Mechanism
- **Prior Works**
- Results

# Prior Source Throttling Works

- **Source throttling for bufferless NoCs**

[Nychis+ Hotnets'10, SIGCOMM'12]

- Application-aware throttling based on starvation rate
- Does not adaptively adjust throttling rate
- “Heterogeneous Throttling”

- **Source throttling off-chip buffered networks**

[Thottethodi+ HPCA'01]

- Dynamically trigger throttling based on fraction of buffer occupancy
- Not application-aware: fully block packet injections of every node
- “Self-tuned Throttling”

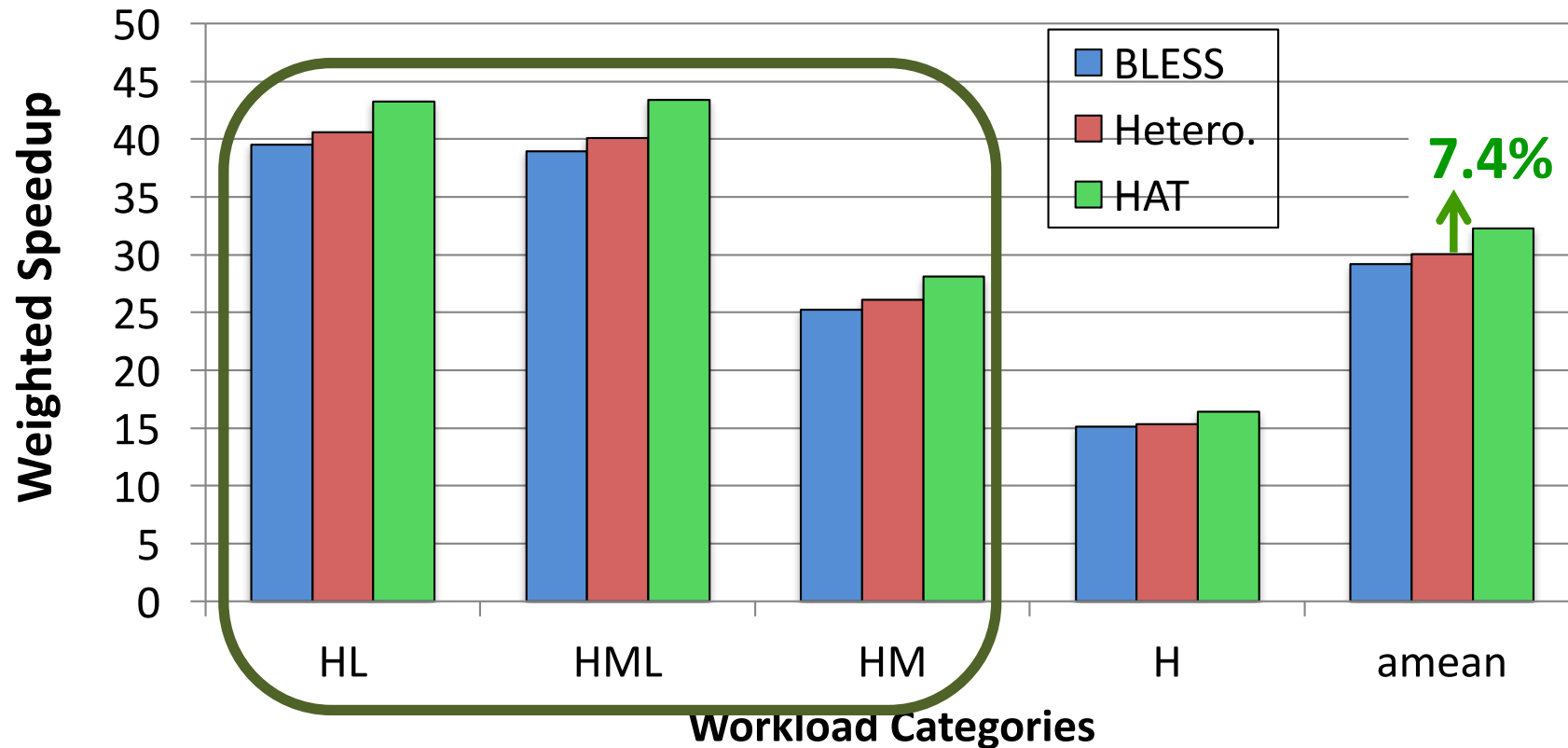
# Outline

- Background and Motivation
- Mechanism
- Prior Works
- **Results**

# Methodology

- **Chip Multiprocessor Simulator**
  - **64-node** multi-core systems with a **2D-mesh topology**
  - Closed-loop core/cache/NoC cycle-level model
  - 64KB L1, perfect L2 (always hits to stress NoC)
- **Router Designs**
  - **Virtual-channel buffered** router: 4 VCs, 4 flits/VC [Dally+ IEEE TPDS'92]
  - **Bufferless deflection** routers: **BLESS** [Moscibroda+ ISCA'09]
- **Workloads**
  - 60 multi-core workloads: SPEC CPU2006 benchmarks
  - Categorized based on their network intensity
    - Low/**M**edium/**H**igh intensity categories
- **Metrics:** Weighted Speedup (perf.), perf./Watt (energy eff.), and maximum slowdown (fairness)

# Performance: Bufferless NoC (BLESS)

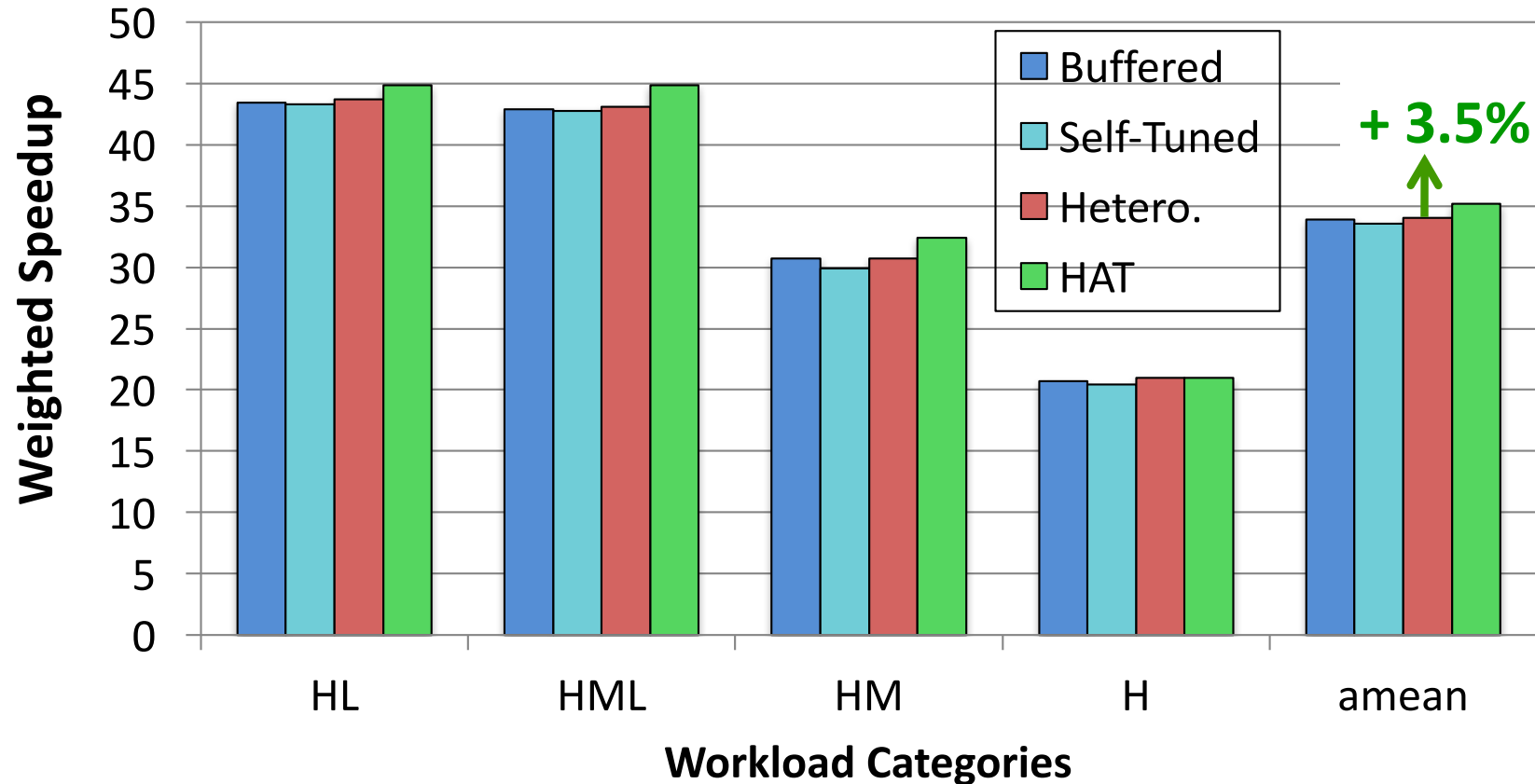


**HAT** provides better performance improvement than past work

Highest improvement on **heterogeneous** workload mixes  
- **L** and **M** are more **sensitive** to network latency



# Performance: Buffered NoC



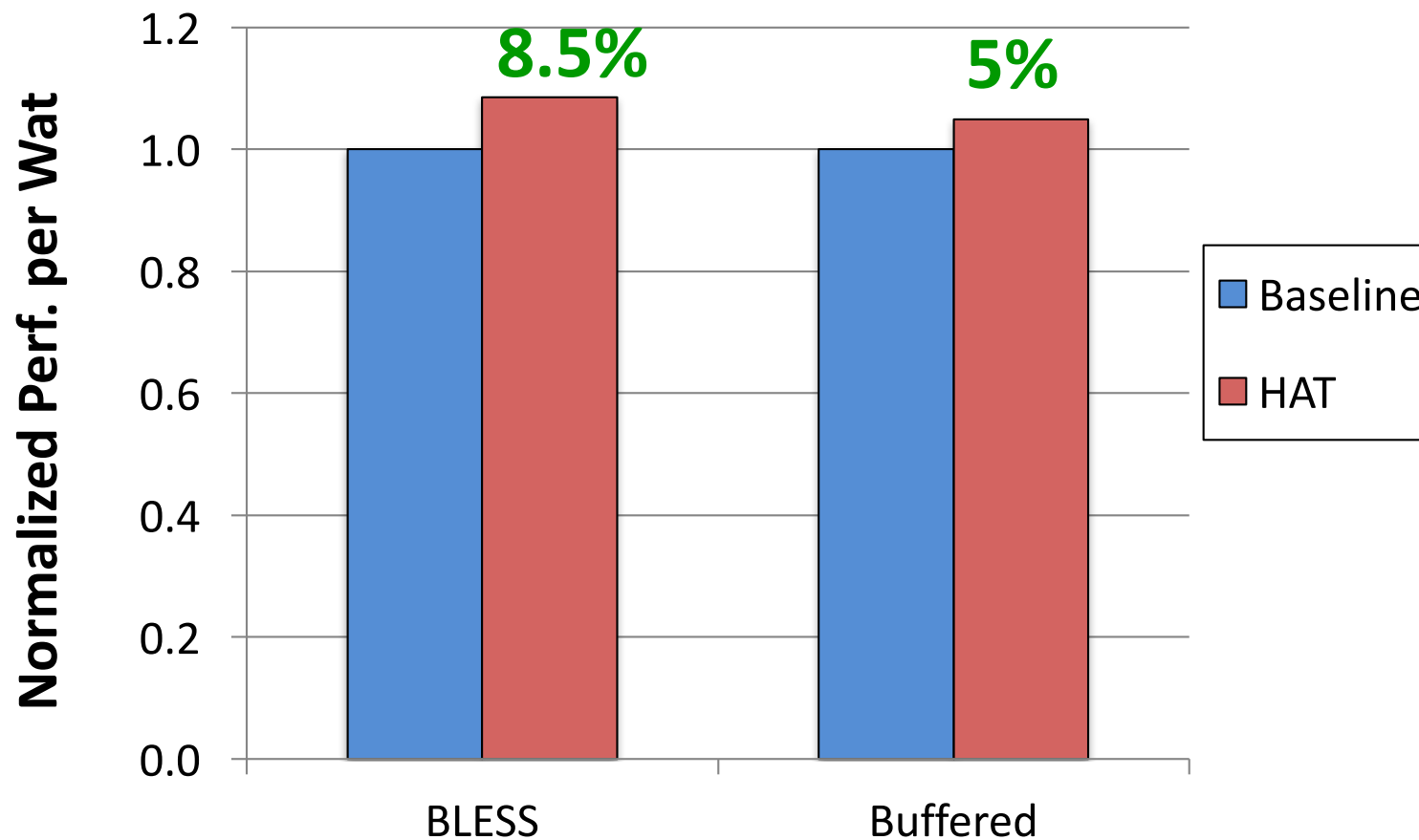
Congestion is much lower in Buffered NoC, but **HAT** still provides performance benefit

# Application Fairness



**HAT** provides better fairness than prior works

# Network Energy Efficiency



**HAT** increases energy efficiency by reducing congestion

# Other Results in Paper

- Performance on **CHIPPER**
- Performance on **multithreaded** workloads
- Parameters sensitivity sweep of **HAT**

# Conclusion

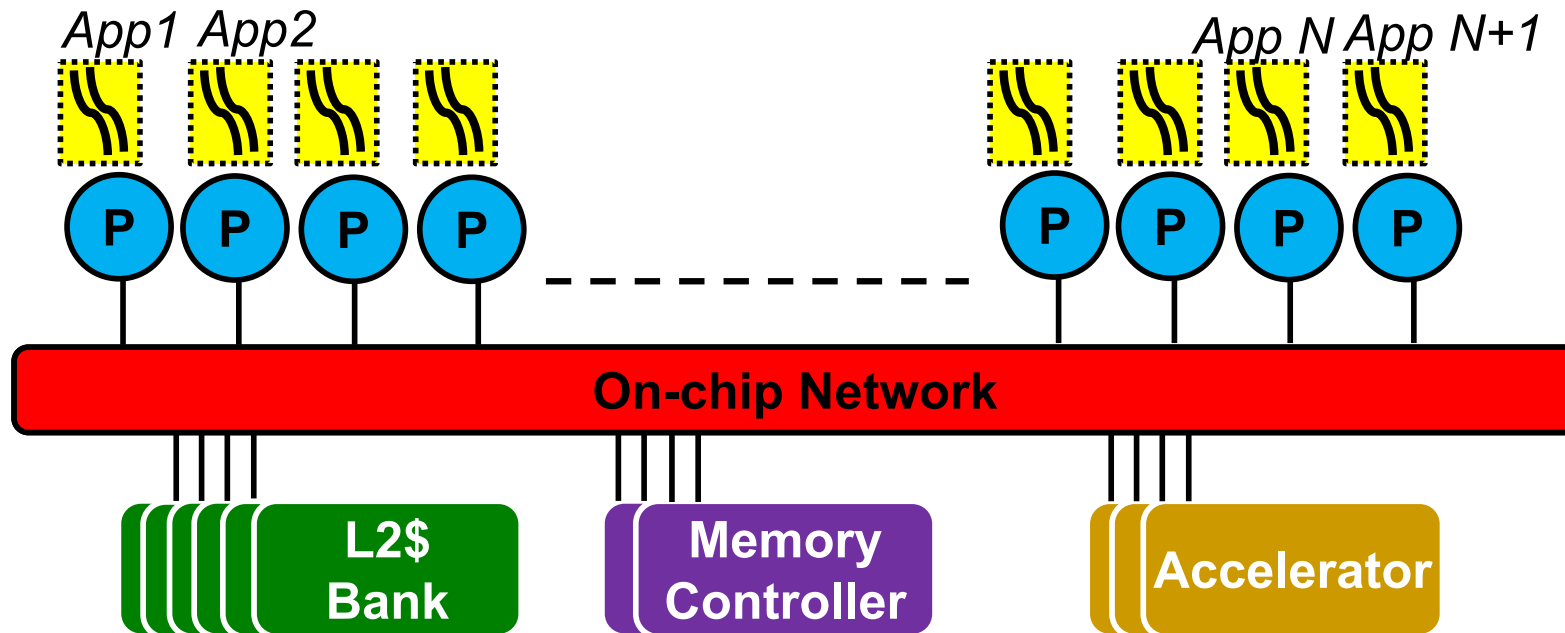
- **Problem:** Packets contend in on-chip networks (NoCs), causing congestion, thus reducing performance
- **Observations:**
  - 1) Some applications are more sensitive to network latency than others
  - 2) Applications must be throttled differently to achieve peak performance
- **Key Idea:** Heterogeneous Adaptive Throttling (HAT)
  - 1) Application-aware source throttling
  - 2) Network-load-aware throttling rate adjustment
- **Result:** Improves performance and energy efficiency over state-of-the-art source throttling policies

# Application-Aware Packet Scheduling

Reetuparna Das, Onur Mutlu, Thomas Moscibroda, and Chita R. Das,  
**"Application-Aware Prioritization Mechanisms for On-Chip Networks"**  
*Proceedings of the 42nd International Symposium on Microarchitecture*  
*(MICRO)*, pages 280-291, New York, NY, December 2009. Slides (pptx)

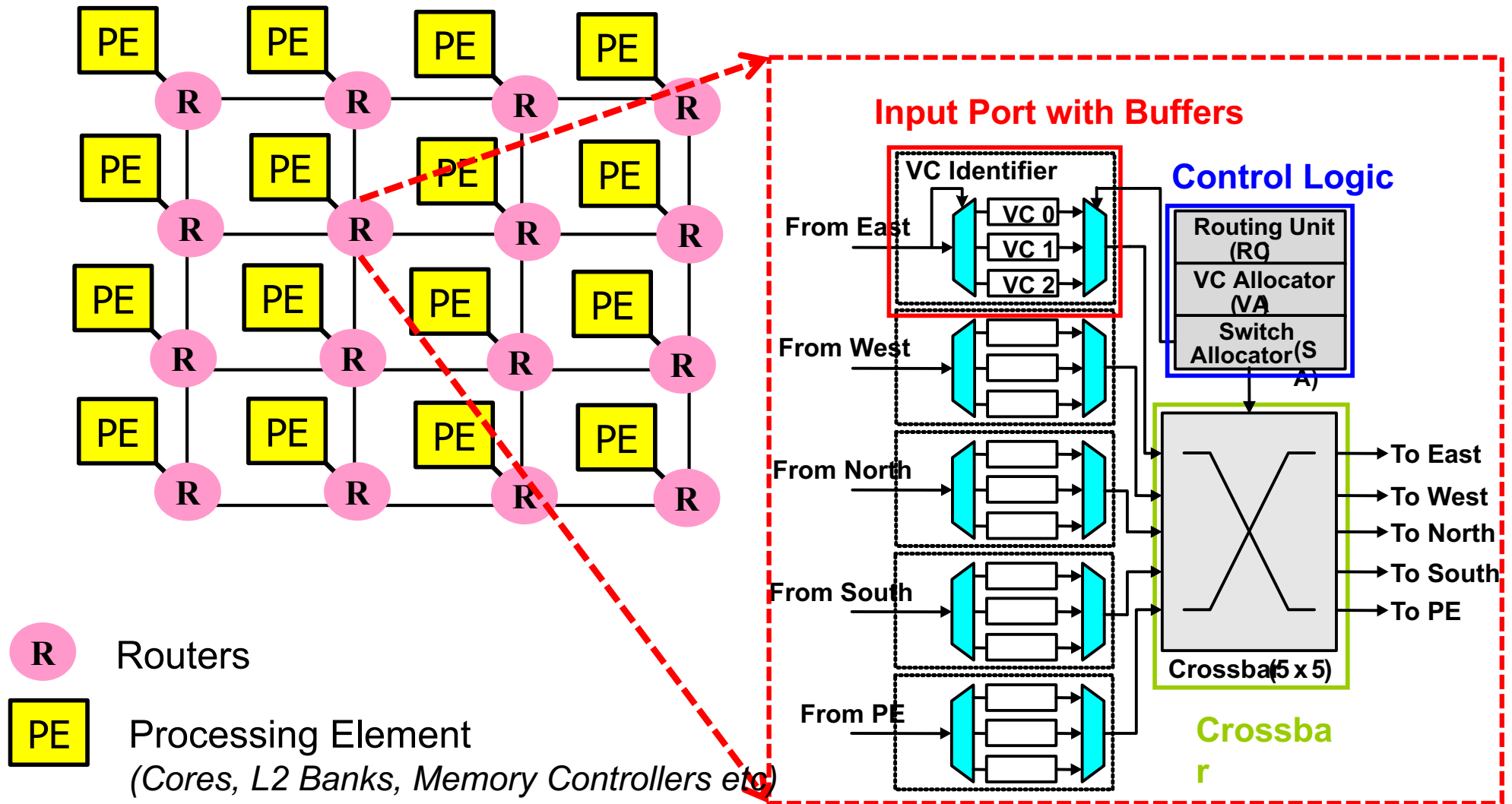
# The Problem: Packet Scheduling

---



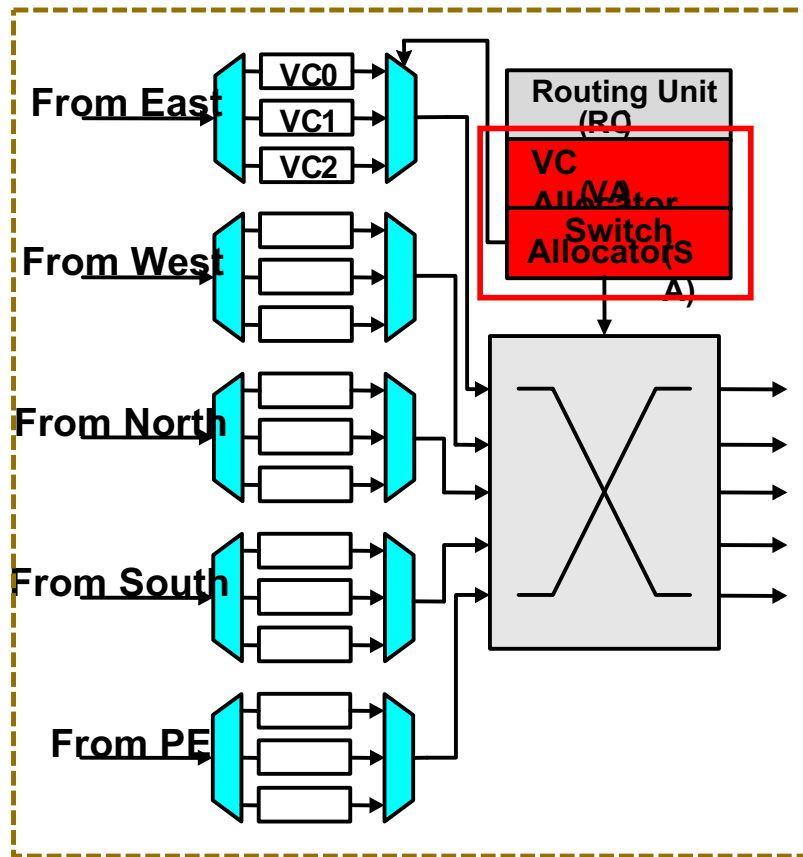
**On-chip Network is a critical resource  
shared by multiple applications**

# The Problem: Packet Scheduling

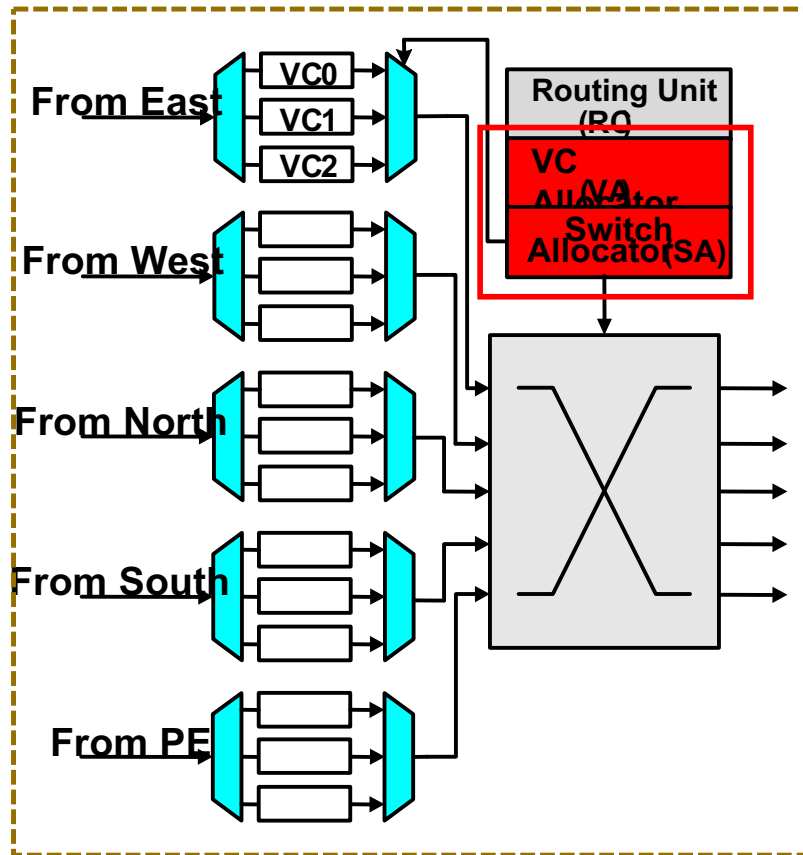




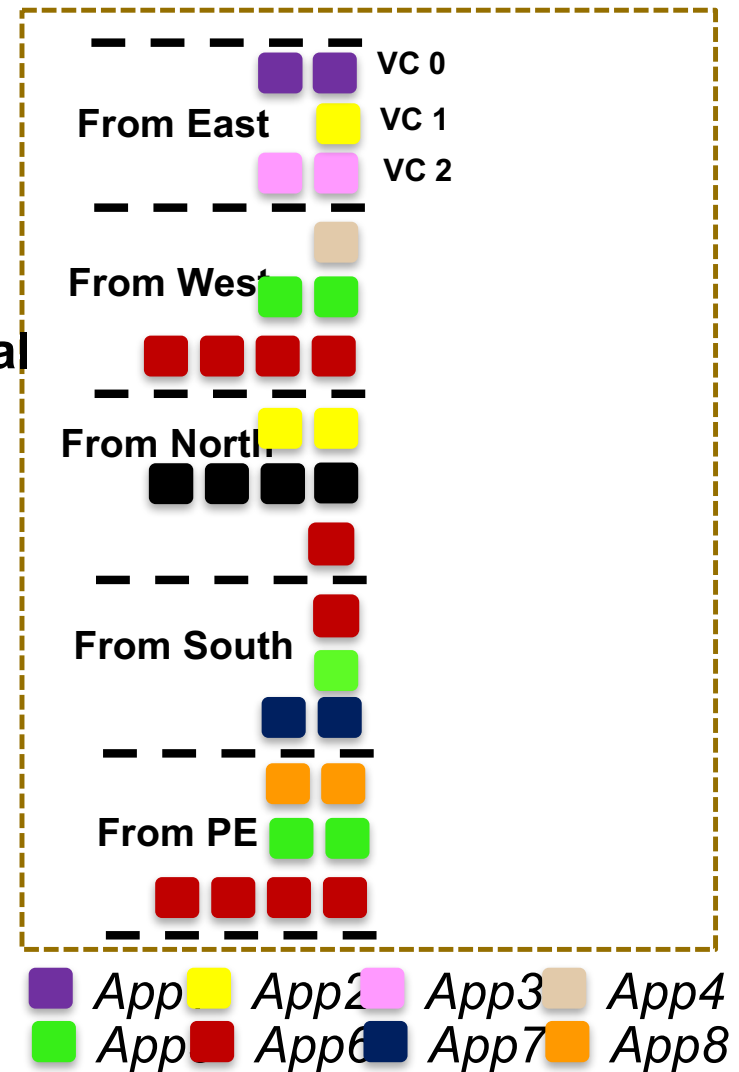
# The Problem: Packet Scheduling



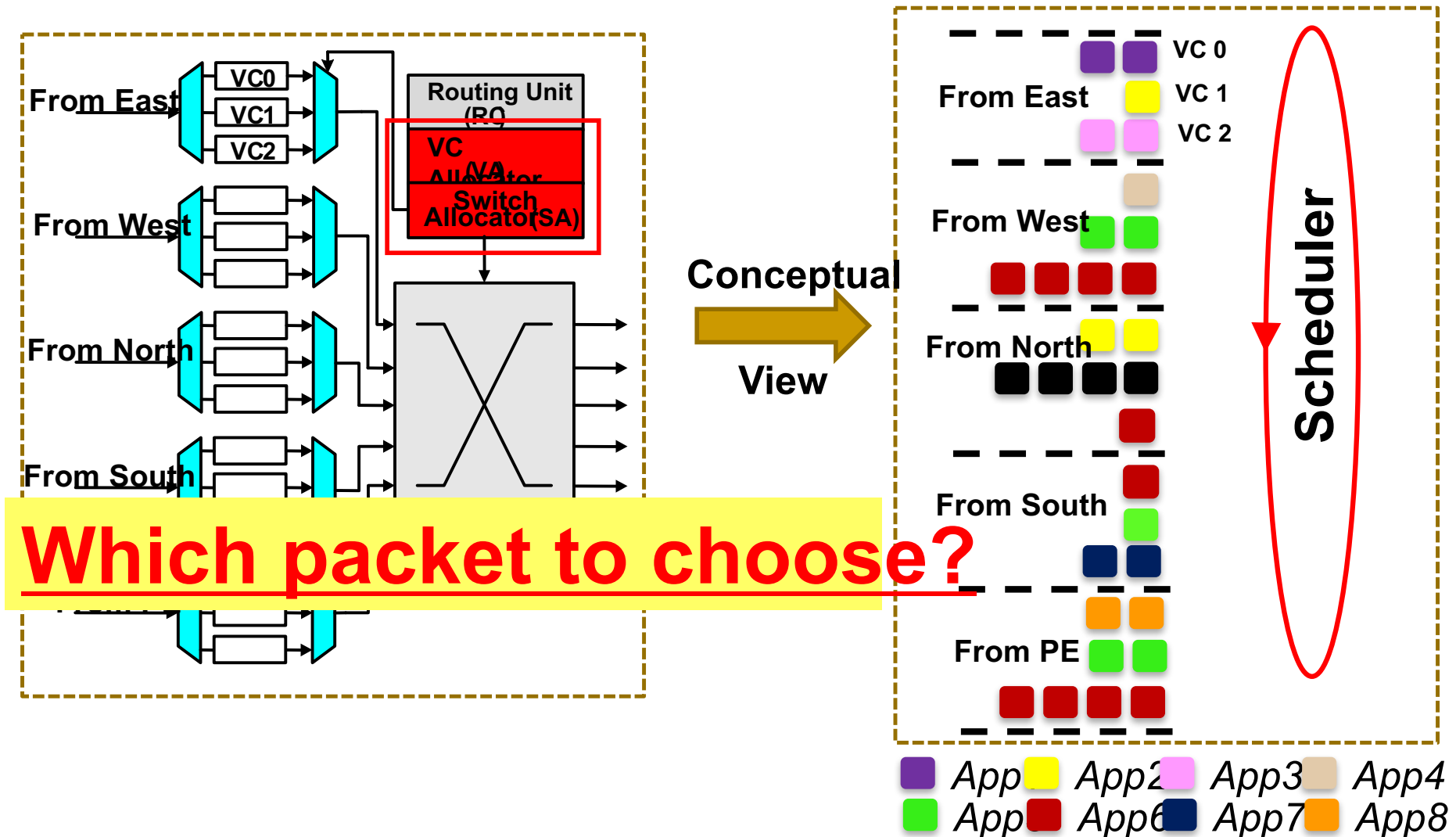
# The Problem: Packet Scheduling



Conceptual  
View



# The Problem: Packet Scheduling



# The Problem: Packet Scheduling

---

- Existing scheduling policies
  - Round Robin
  - Age
- Problem 1: **Local** to a router
  - Lead to contradictory decision making between routers:  
packets from one application may be prioritized at one router,  
to be delayed at next.
- Problem 2: **Application oblivious**
  - Treat all applications **packets equally**
  - But applications are heterogeneous
- **Solution**: Application-aware global scheduling policies.

# Motivation: Stall-Time Criticality

---

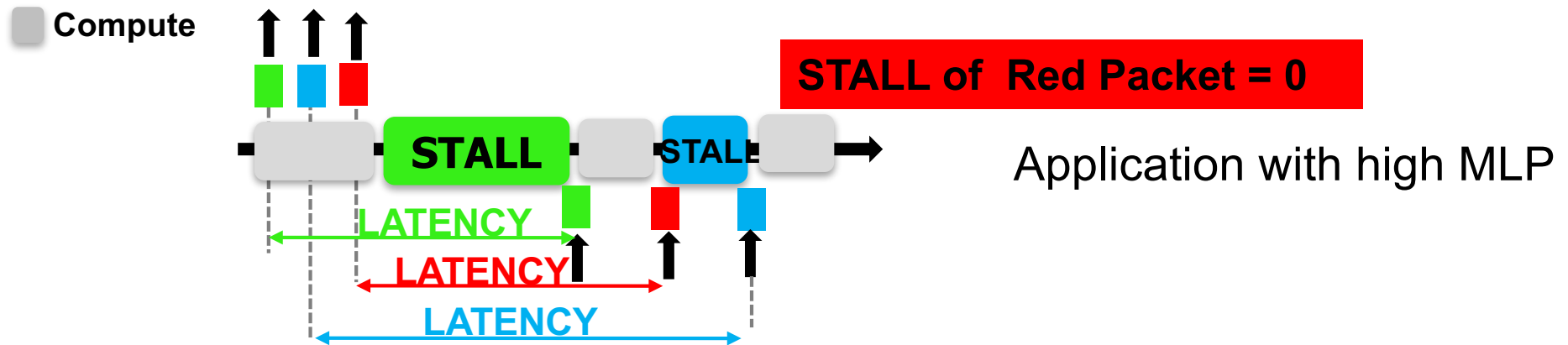
- Applications are **not homogenous**
- Applications have different **criticality** with respect to the **network**
  - Some applications are network latency sensitive
  - Some applications are network latency tolerant
- Application's **Stall Time Criticality (STC)** can be measured by its average network stall time per packet (**i.e. NST/packet**)
  - **Network Stall Time (NST)** is number of cycles the processor stalls waiting for network transactions to complete

# Motivation: Stall-Time Criticality

---

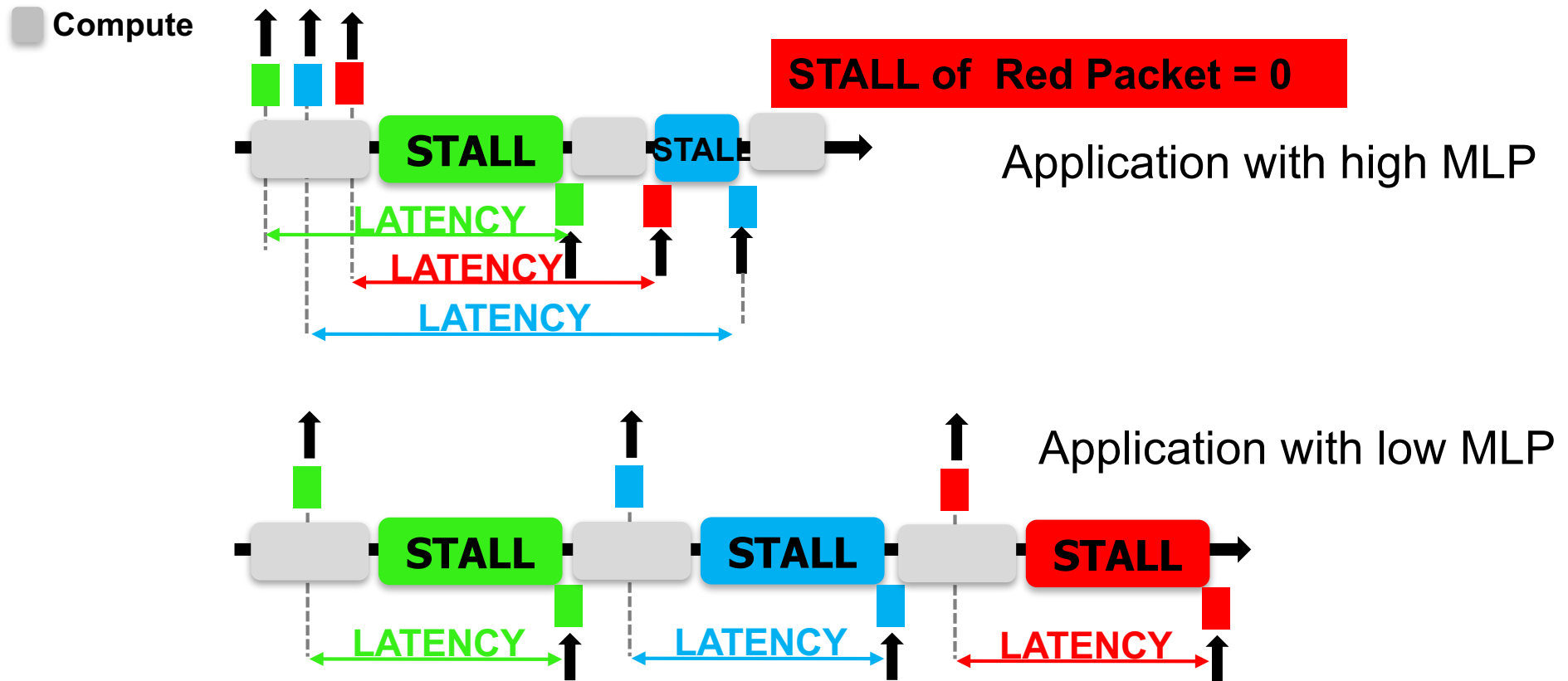
- Why do applications have different network stall time criticality (STC)?
  - Memory Level Parallelism (MLP)
    - **Lower MLP leads to higher criticality**
  - Shortest Job First Principle (SJF)
    - **Lower network load leads to higher criticality**

# STC Principle 1: MLP



- Observation 1: **Packet Latency  $\neq$  Network Stall Time**

# STC Principle 1: MLP



- Observation 1: **Packet Latency != Network Stall Time**
- Observation 2: A low MLP application's packets have higher criticality than a high MLP application's

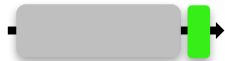


# STC Principle 2: Shortest-Job-First

Light Application

Heavy Application

Running ALONE



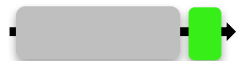
Baseline (RR) Scheduling



4X network slow down

1.3X network slow down

SJF Scheduling



1.2X network slow down

1.6X network slow down

Overall system throughput (weighted speedup) increases by 34%

# Solution: Application-Aware Policies

---

- Idea
  - Identify critical applications (i.e. network sensitive applications) and prioritize their packets in each router.
- Key components of scheduling policy:
  - Application Ranking
  - Packet Batching
- Propose low-hardware complexity solution

# Component 1: Ranking

---

- Ranking distinguishes applications based on Stall Time Criticality (STC)
- Periodically **rank** applications based on STC
- Explored many **heuristics** for estimating STC
  - Heuristic based on **outermost private cache Misses Per Instruction (L1-MPI)** is the most effective
  - **Low L1-MPI => high STC => higher rank**
- Why Misses Per Instruction (L1-MPI)?
  - Easy to Compute (low complexity)
  - Stable Metric (unaffected by interference in network)

# Component 1 : How to Rank?

---

- Execution time is divided into fixed “ranking intervals”
  - Ranking interval is 350,000 cycles
- At the end of an interval, each core calculates their L1-MPI and sends it to the Central Decision Logic (CDL)
  - CDL is located in the central node of mesh
- CDL forms a rank order and sends back its rank to each core
  - Two control packets per core every ranking interval
- Ranking order is a “partial order”
- Rank formation is not on the critical path
  - Ranking interval is significantly longer than rank computation time
  - Cores use older rank values until new ranking is available

# Component 2: Batching

---

- Problem: **Starvation**
  - Prioritizing a higher ranked application can lead to starvation of lower ranked application
- Solution: **Packet Batching**
  - Network packets are grouped into finite sized batches
  - **Packets of older batches are prioritized over younger batches**
- **Time-Based Batching**
  - New batches are formed in a periodic, synchronous manner across all nodes in the network, every  $T$  cycles

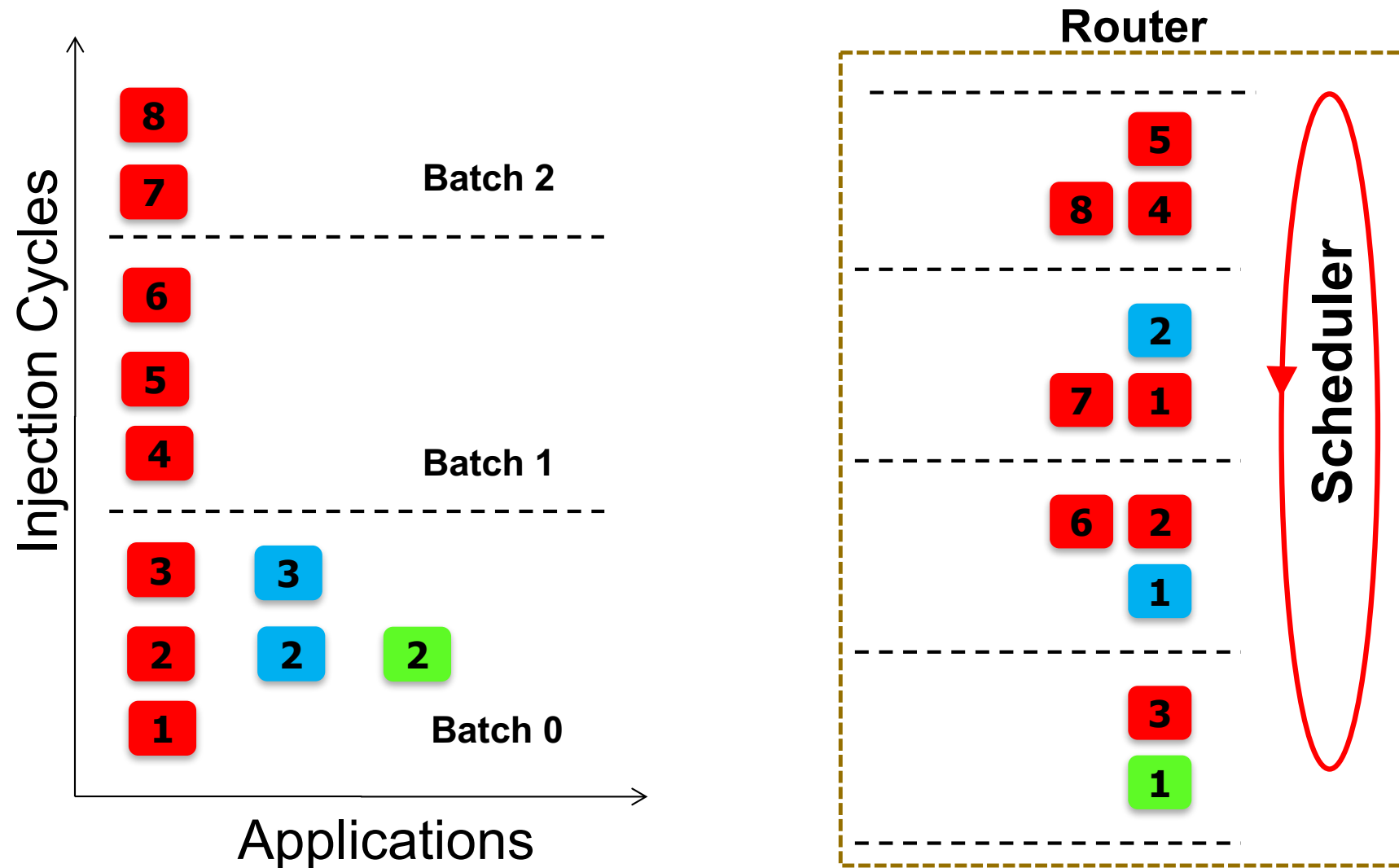
# Putting it all together: STC Scheduling

---

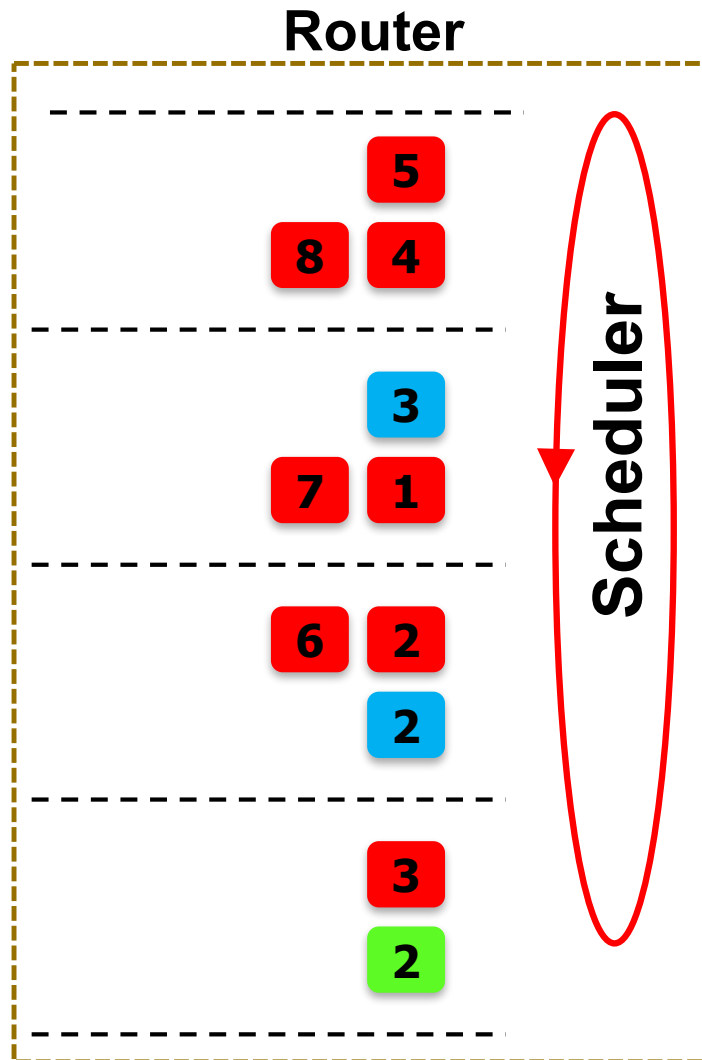
## Policy

- Before injecting a packet into the network, it is tagged with
  - Batch ID (*3 bits*)
  - Rank ID (*3 bits*)
- Three tier priority structure at routers
  - **Oldest batch first** (*prevent starvation*)
  - **Highest rank first** (*maximize performance*)
  - **Local Round-Robin** (*final tie breaker*)
- Simple hardware support: priority arbiters
- **Global coordinated scheduling**
  - Ranking order and batching order are same across all routers

# STC Scheduling Example



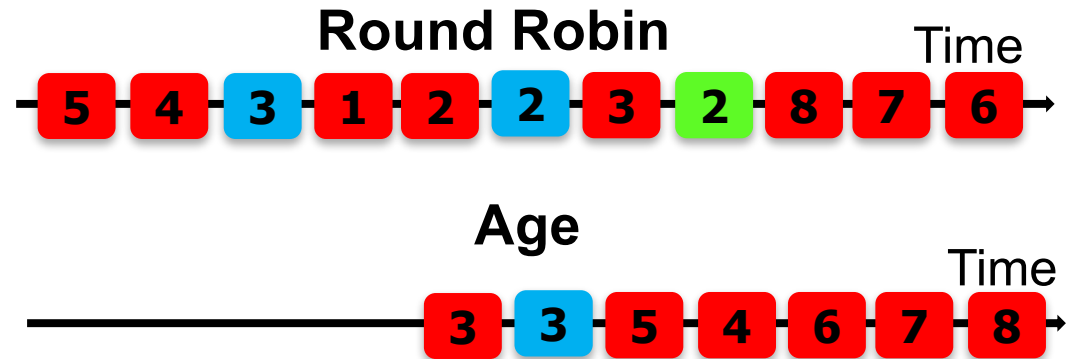
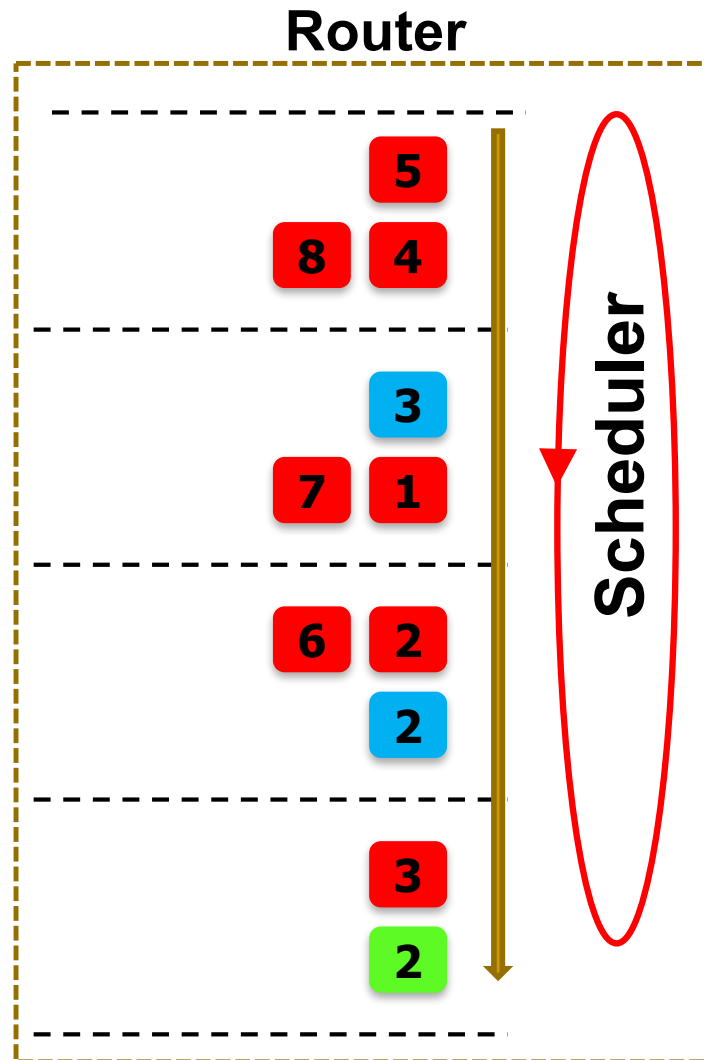
# STC Scheduling Example



STALL CYCLES				Avg
RR	8	6	11	8.3
Age				
STC				






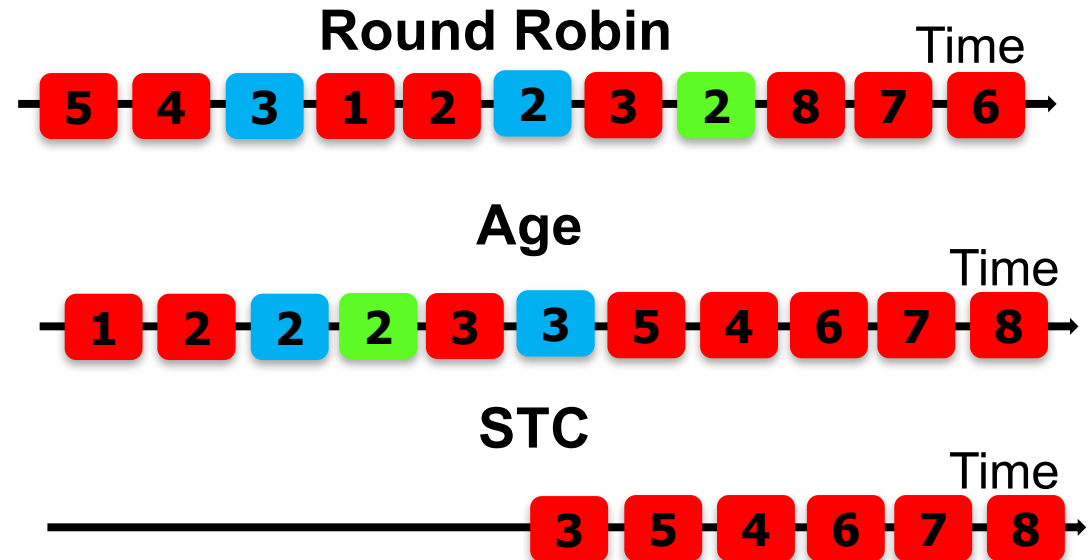
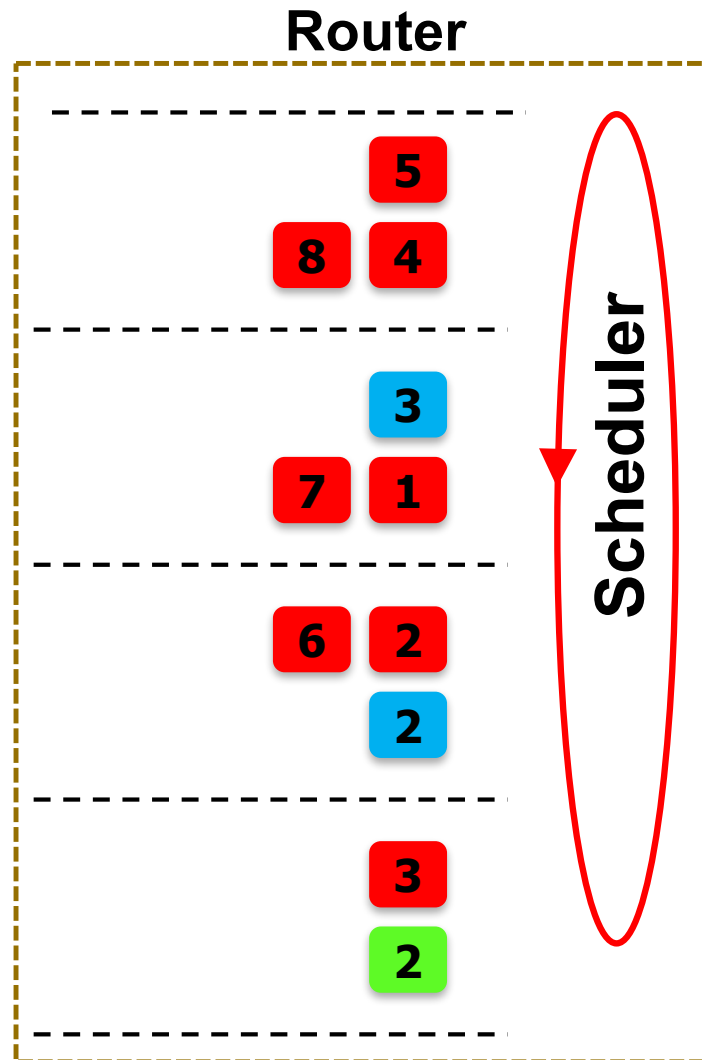
# STC Scheduling Example



STALL CYCLES				Avg
RR	8	6	11	8.3
Age	4	6	11	7.0
STC				

# STC Scheduling Example

Rank order   



STALL CYCLES				Avg
RR	8	6	11	8.3
Age	4	6	11	7.0
STC	1	3	11	5.0

# STC Evaluation Methodology

---

- 64-core system
  - ❑ x86 processor model based on Intel Pentium M
  - ❑ 2 GHz processor, 128-entry instruction window
  - ❑ 32KB private L1 and 1MB per core shared L2 caches, 32 miss buffers
  - ❑ 4GB DRAM, 320 cycle access latency, 4 on-chip DRAM controllers
- Detailed Network-on-Chip model
  - ❑ 2-stage routers (with speculation and look ahead routing)
  - ❑ Wormhole switching (8 flit data packets)
  - ❑ Virtual channel flow control (6 VCs, 5 flit buffer depth)
  - ❑ 8x8 Mesh (128 bit bi-directional channels)
- Benchmarks
  - ❑ Multiprogrammed scientific, server, desktop workloads (35 applications)
  - ❑ 96 workload combinations

# Comparison to Previous Policies

---

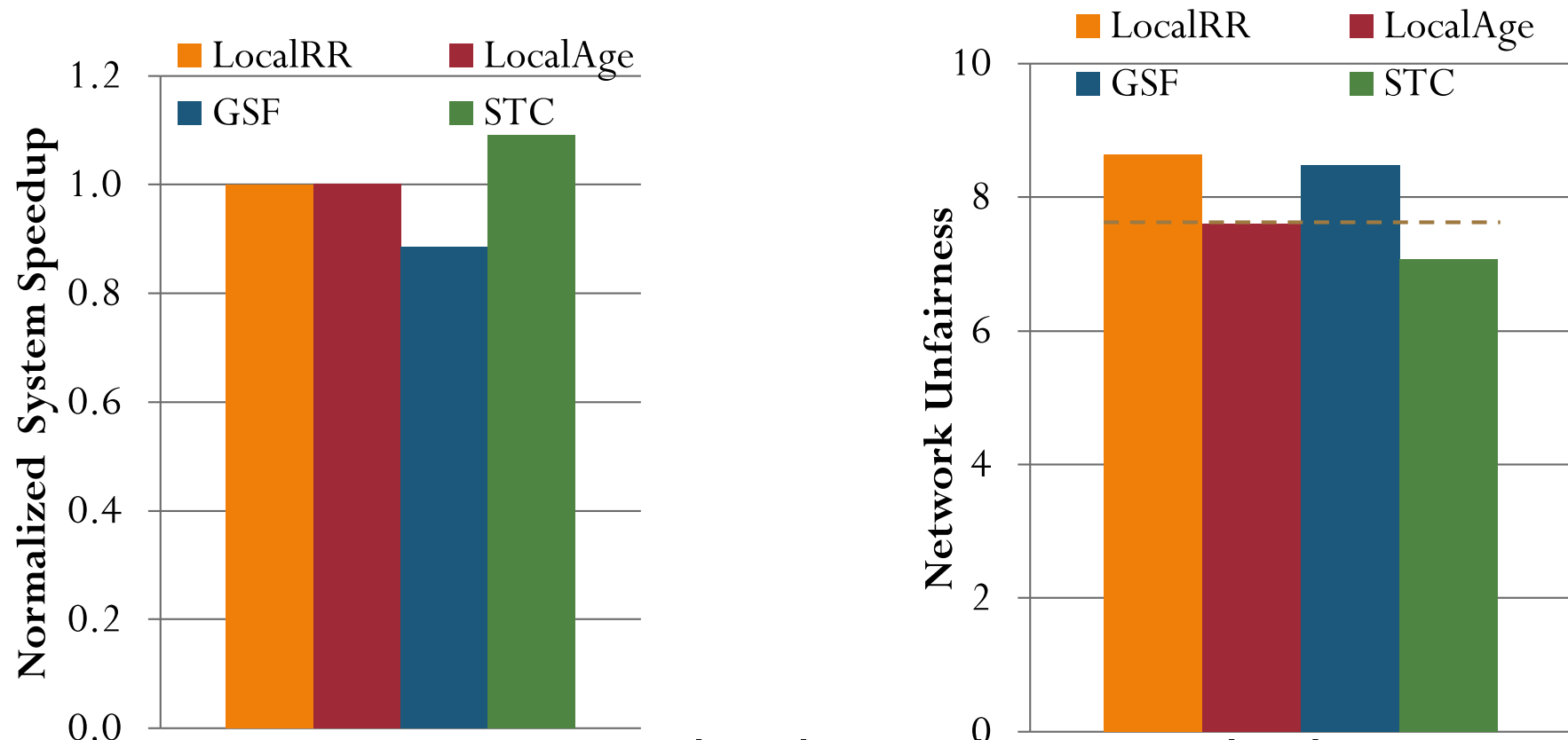
## ■ **Round Robin & Age (Oldest-First)**

- ❑ Local and application oblivious
- ❑ Age is biased towards heavy applications
  - heavy applications flood the network
  - higher likelihood of an older packet being from heavy application

## ■ **Globally Synchronized Frames (GSF)** [Lee et al., ISCA 2008]

- ❑ Provides **bandwidth fairness** at the expense of **system performance**
- ❑ Penalizes heavy and bursty applications
  - Each application gets equal and fixed quota of flits (credits) in each batch.
  - Heavy application quickly run out of credits after injecting into all active batches & stalls until oldest batch completes and frees up fresh credits.
  - Underutilization of network resources

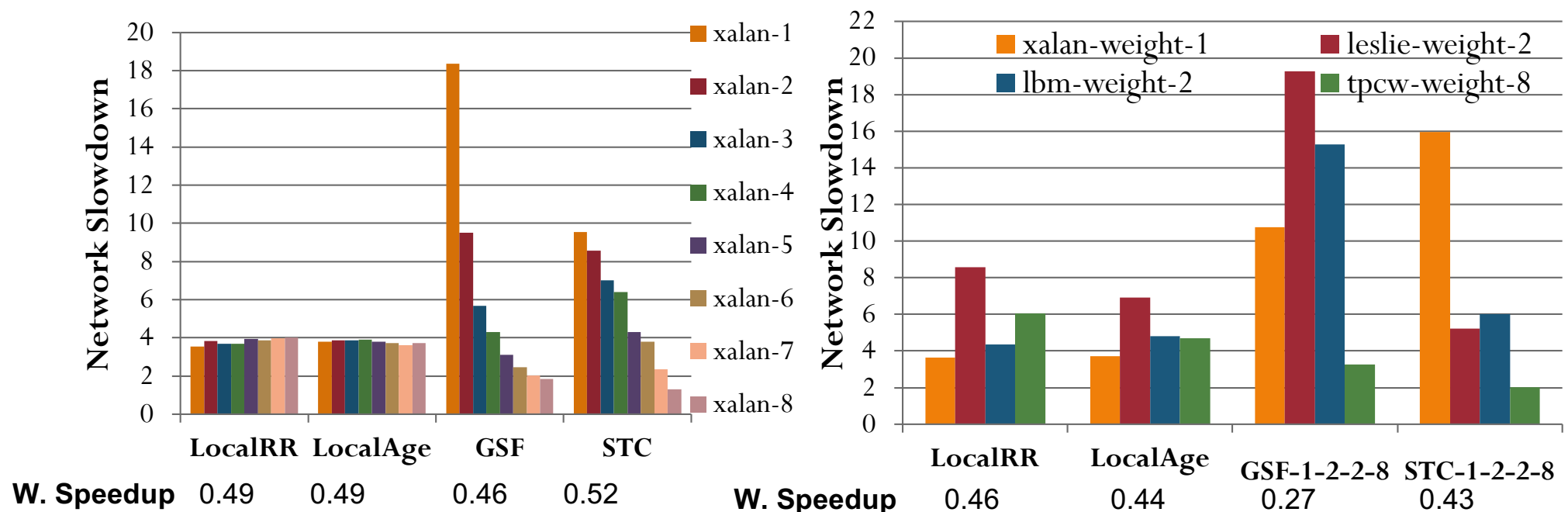
# STC System Performance and Fairness



- 9.1% improvement in weighted speedup over the best existing policy (averaged across 96 workloads)

# Enforcing Operating System Priorities

- Existing policies cannot enforce operating system (OS) assigned priorities in Network-on-Chip
- Proposed framework can enforce OS assigned priorities
  - Weight of applications => Ranking of applications
  - Configurable batching interval based on application weight



# Application Aware Packet Scheduling: Summary

---

- Packet scheduling policies critically impact performance and fairness of NoCs
- Existing packet scheduling policies are **local** and **application oblivious**
- STC is a new, **global, application-aware approach to packet scheduling** in NoCs
  - **Ranking:** differentiates applications based on their criticality
  - **Batching:** avoids starvation due to rank-based prioritization
- Proposed framework
  - provides **higher system performance and fairness** than existing policies
  - can **enforce OS assigned priorities** in network-on-chip

# Slack-Driven Packet Scheduling

Reetuparna Das, Onur Mutlu, Thomas Moscibroda, and Chita R. Das,  
**"Aergia: Exploiting Packet Latency Slack in On-Chip Networks"**  
*Proceedings of the 37th International Symposium on Computer Architecture*  
*(ISCA)*, pages 106-116, Saint-Malo, France, June 2010. Slides (pptx)



# Packet Scheduling in NoC

---

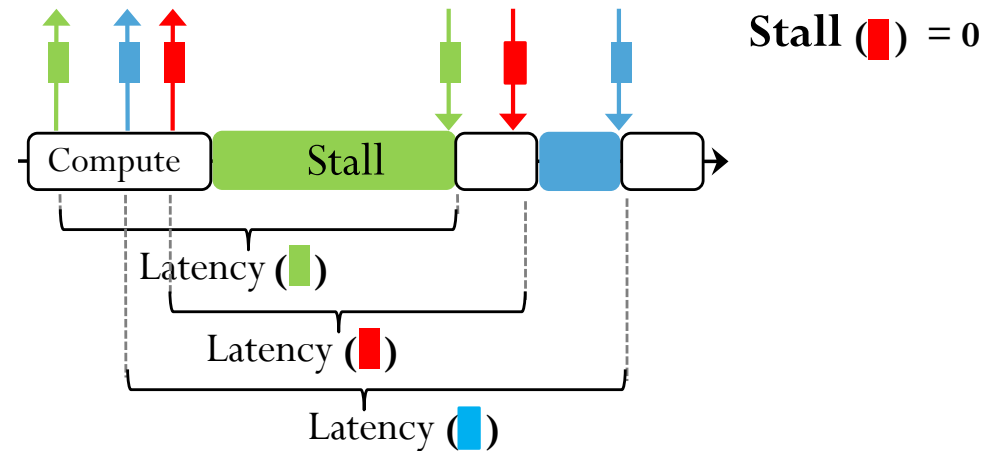
- Existing scheduling policies
  - Round robin
  - Age
- Problem
  - Treat all packets equally
  - Application-oblivious
- Packets have **different criticality**
  - Packet is critical if latency of a packet affects application's performance
  - Different criticality due to memory level parallelism (MLP)

All packets are not the same...!!!



# MLP Principle

---



Packet Latency  $\neq$  Network Stall Time

Different Packets have different criticality due to MLP

Criticality(■) > Criticality(■) > Criticality(■)

# Outline

---

- Introduction
  - Packet Scheduling
  - Memory Level Parallelism
- **Aérgia**
  - Concept of Slack
  - Estimating Slack
- Evaluation
- Conclusion

# What is Aérgia?

---



- Aérgia is the spirit of laziness in Greek mythology
- Some packets can afford to **slack**!

# Outline

---

- Introduction
  - Packet Scheduling
  - Memory Level Parallelism
- Aéria
  - Concept of Slack
  - Estimating Slack
- Evaluation
- Conclusion

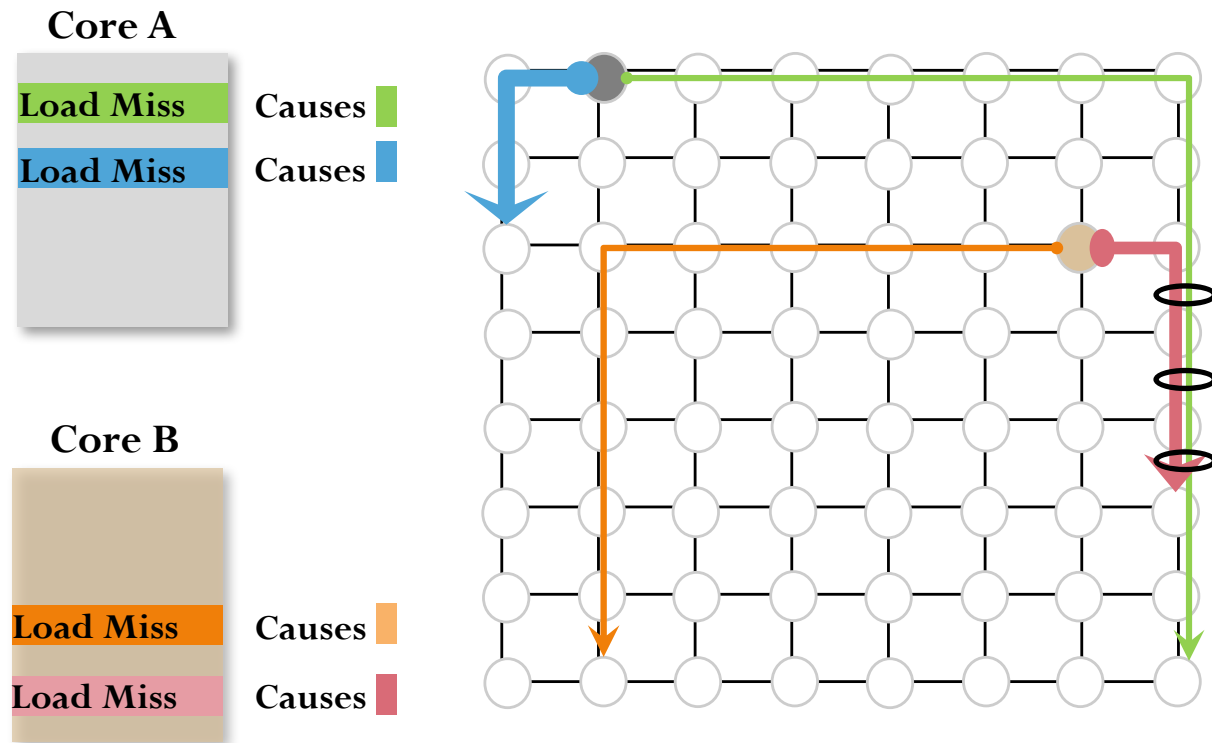
# Slack of Packets



---

- What is slack of a packet?
  - Slack of a packet is number of cycles it can be delayed in a router without (significantly) reducing application's performance
  - Local network slack
- Source of slack: Memory-Level Parallelism (MLP)
  - Latency of an application's packet hidden from application due to overlap with latency of pending cache miss requests
- Prioritize packets with lower slack





# Prioritizing using Slack



Packet	Latency	Slack
	13 hops	0 hops
	3 hops	10 hops

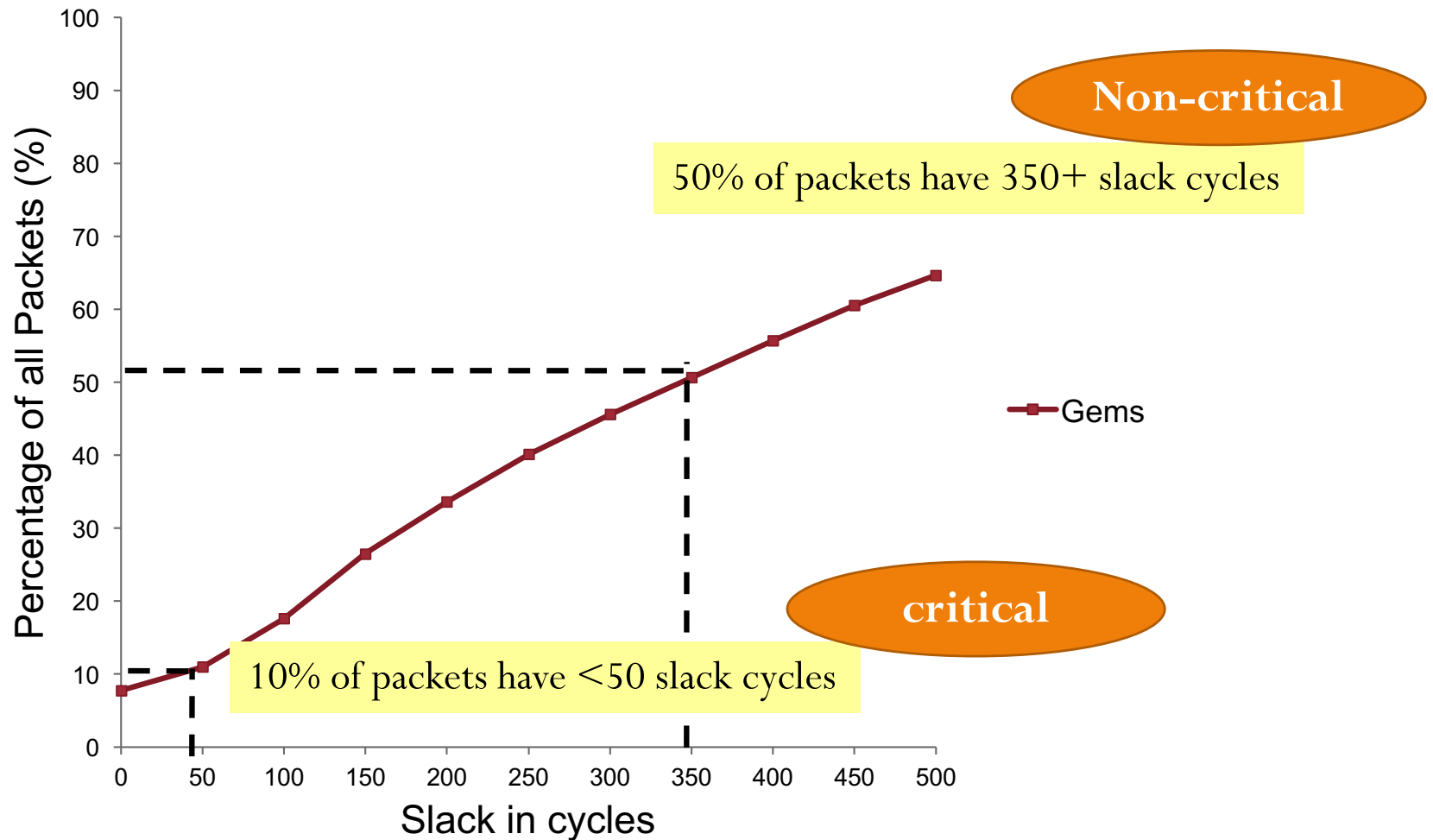
 Interference at 3 hops

Slack() > Slack ()

Prioritize 

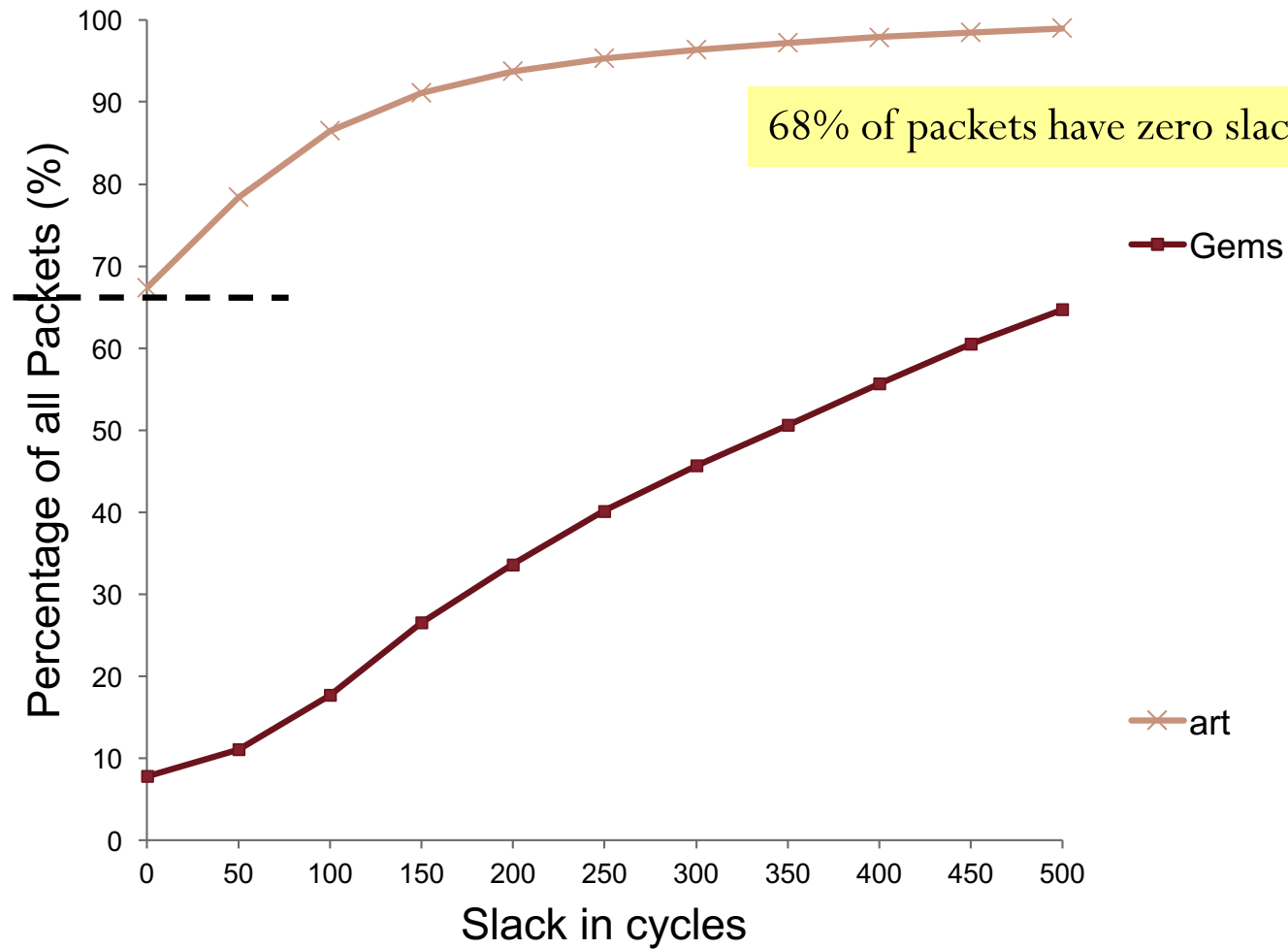


# Slack in Applications

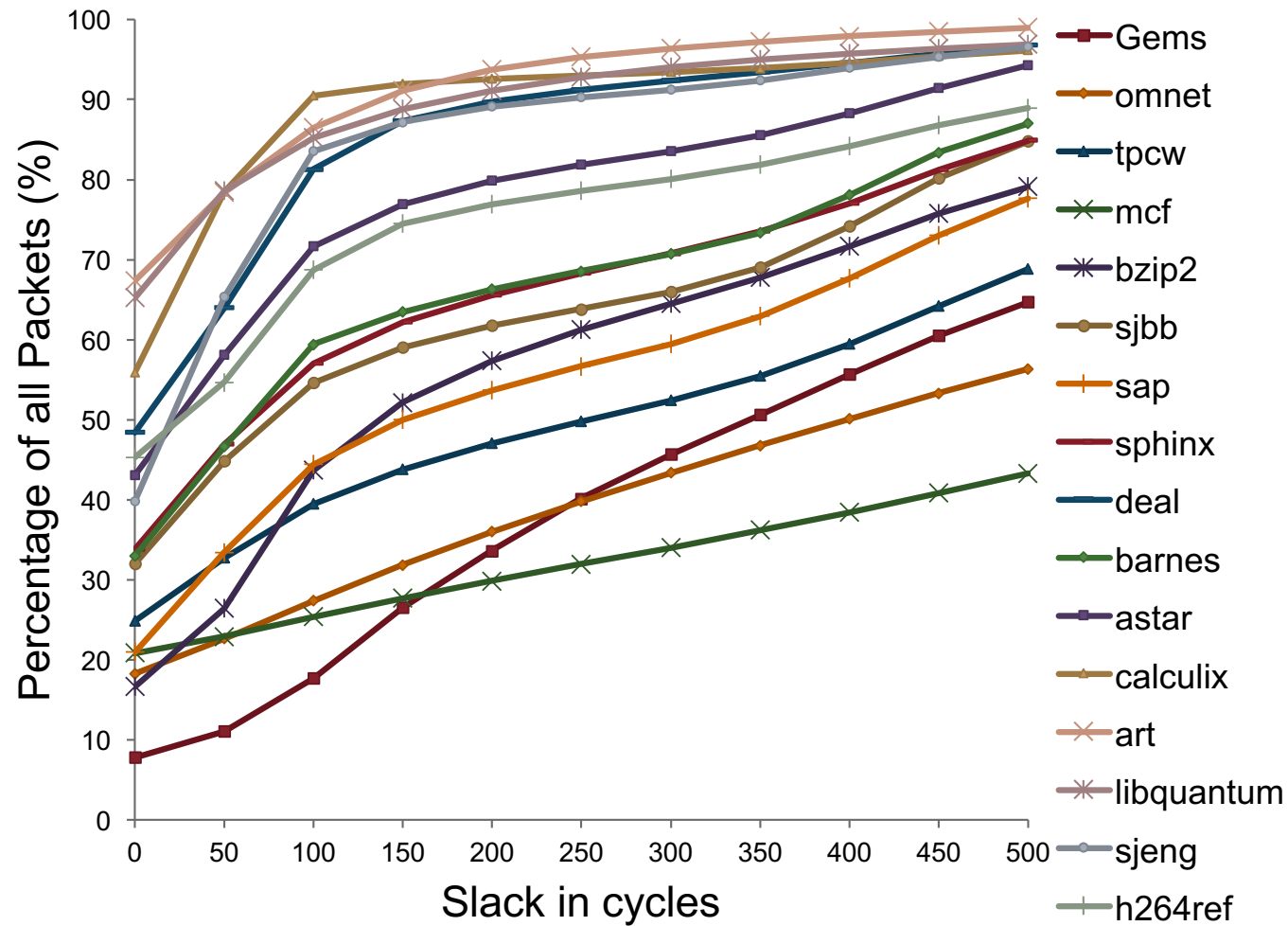


# Slack in Applications

---



# Diversity in Slack

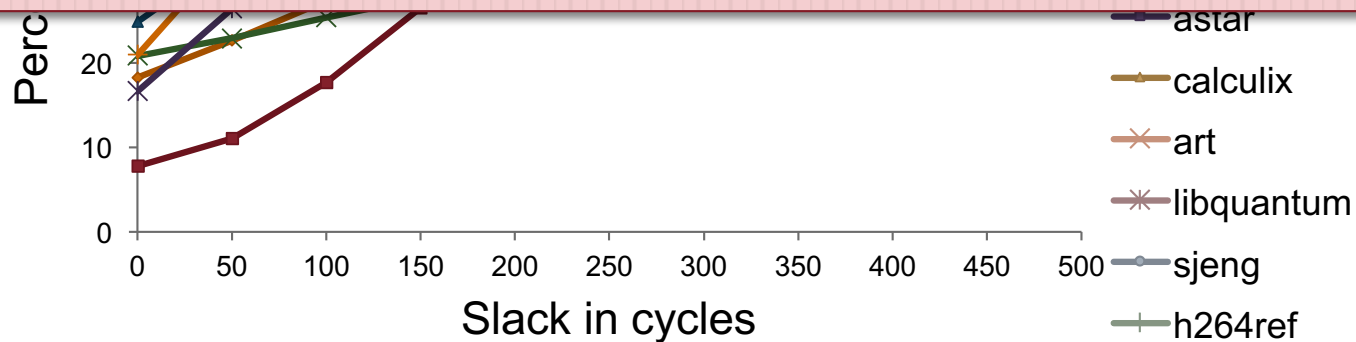


# Diversity in Slack

Slack varies **between** packets of **different** applications



Slack varies **between** packets of a **single** application



# Outline

---

- Introduction
  - Packet Scheduling
  - Memory Level Parallelism
- Aéria
  - Concept of Slack
  - Estimating Slack
- Evaluation
- Conclusion

# Estimating Slack Priority

---

**Slack (P)** = Max (Latencies of P's Predecessors) – Latency of P

**Predecessors(P)** are the packets of outstanding cache miss requests when P is issued

- Packet latencies not known when issued
- Predicting latency of any packet Q
  - Higher latency if Q corresponds to an L2 miss
  - Higher latency if Q has to travel farther number of hops

# Estimating Slack Priority

---

- Slack of P = Maximum Predecessor Latency – Latency of P

- Slack(P) = 

PredL2 (2 bits)	MyL2 (1 bit)	HopEstimate (2 bits)
--------------------	-----------------	-------------------------

**PredL2**: Set if any predecessor packet is servicing L2 miss

**MyL2**: Set if P is NOT servicing an L2 miss

**HopEstimate**: Max (# of hops of Predecessors) – hops of P

# Estimating Slack Priority

---

- How to predict L2 hit or miss at core?
  - *Global Branch Predictor* based L2 Miss Predictor
    - Use Pattern History Table and 2-bit saturating counters
  - *Threshold* based L2 Miss Predictor
    - If #L2 misses in “M” misses  $\geq$  “T” threshold then next load is a L2 miss.
- Number of miss predecessors?
  - List of outstanding L2 Misses
- Hops estimate?
  - Hops  $\Rightarrow \Delta X + \Delta Y$  distance
  - Use predecessor list to calculate slack hop estimate



# Starvation Avoidance

---

- Problem: **Starvation**
  - Prioritizing packets can lead to starvation of lower priority packets
- Solution: **Time-Based Packet Batching**
  - New batches are formed at every  $T$  cycles
  - Packets of older batches are prioritized over younger batches

# Putting it all together

---

- Tag header of the packet with priority bits before injection

**Priority (P) =**

Batch  
(3 bits)

PredL2  
(2 bits)

MyL2  
(1 bit)

HopEstimate  
(2 bits)

- Priority(P)?

- P's batch

*(highest priority)*

- P's Slack

- Local Round-Robin

*(final tie breaker)*

# Outline

---

- Introduction
  - Packet Scheduling
  - Memory Level Parallelism
- Aéria
  - Concept of Slack
  - Estimating Slack
- Evaluation
- Conclusion

# Evaluation Methodology

---

- 64-core system
  - x86 processor model based on Intel Pentium M
  - 2 GHz processor, 128-entry instruction window
  - 32KB private L1 and 1MB per core shared L2 caches, 32 miss buffers
  - 4GB DRAM, 320 cycle access latency, 4 on-chip DRAM controllers
- Detailed Network-on-Chip model
  - 2-stage routers (with speculation and look ahead routing)
  - Wormhole switching (8 flit data packets)
  - Virtual channel flow control (6 VCs, 5 flit buffer depth)
  - 8x8 Mesh (128 bit bi-directional channels)
- Benchmarks
  - Multiprogrammed scientific, server, desktop workloads (35 applications)
  - 96 workload combinations

# Qualitative Comparison

---

- **Round Robin & Age**

- Local and application oblivious
- Age is biased towards heavy applications

- **Globally Synchronized Frames (GSF)**

[Lee et al., ISCA 2008]

- Provides **bandwidth fairness** at the expense of **system performance**
- Penalizes heavy and bursty applications

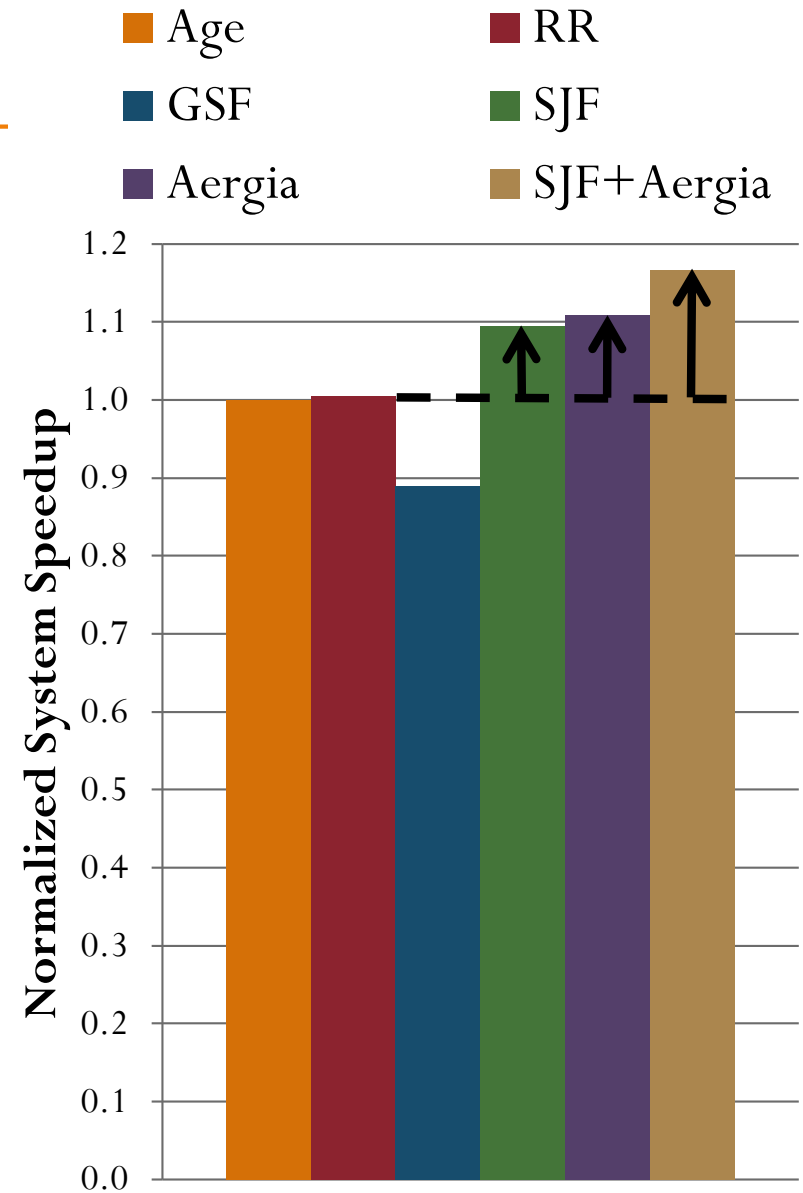
- **Application-Aware Prioritization Policies (SJF)**

[Das et al., MICRO 2009]

- **Shortest-Job-First Principle**
- Packet scheduling policies which prioritize network sensitive applications which inject lower load

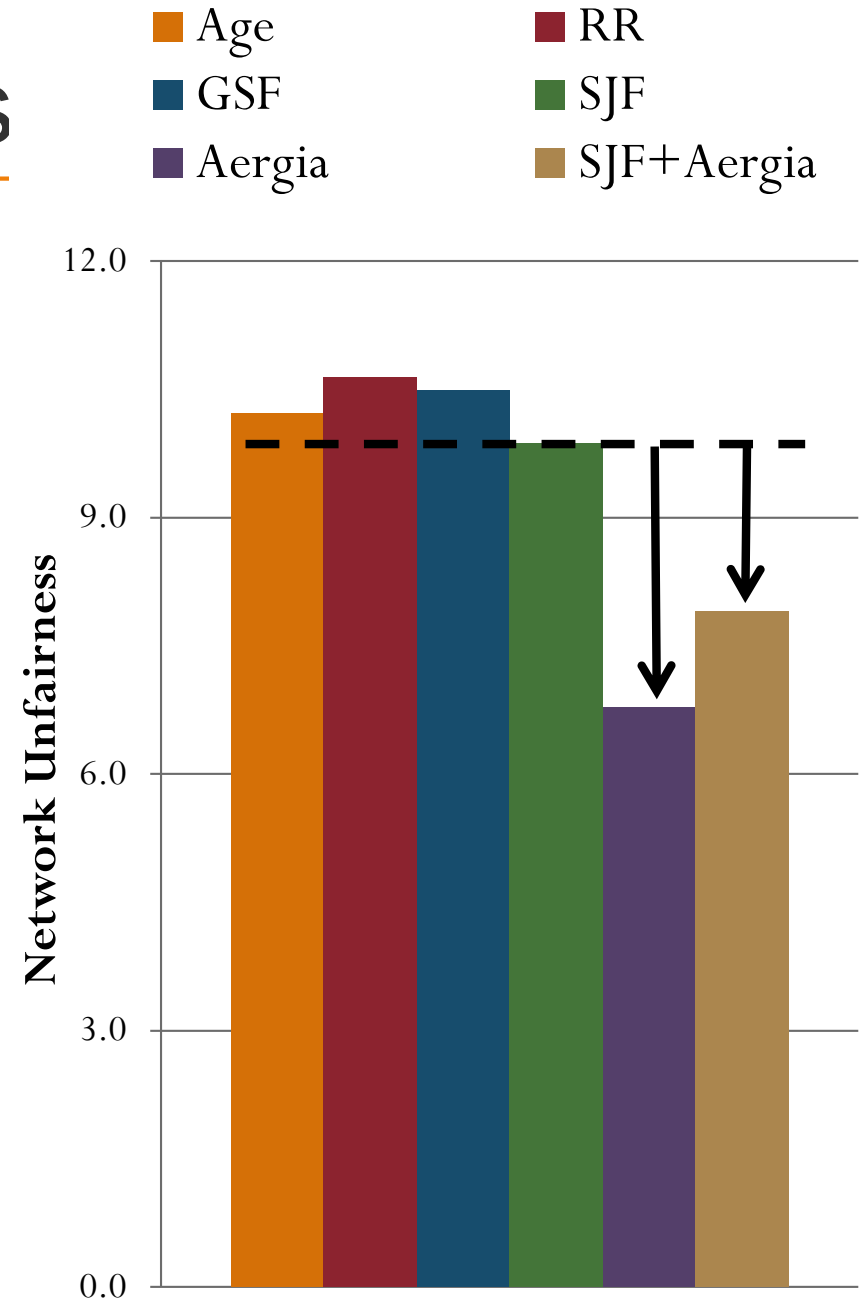
# System Performance

- SJF provides 8.9% improvement in weighted speedup
- Aérgia improves system throughput by 10.3%
- Aérgia+SJF improves system throughput by 16.1%



# Network Unfairness

- SJF does not imbalance network fairness
- Aergia improves network unfairness by 1.5X
- SJF+Aergia improves network unfairness by 1.3X



# Conclusions & Future Directions

---

- Packets have different criticality, yet existing packet scheduling policies **treat all packets equally**
- We propose a new approach to packet scheduling in NoCs
  - We define **Slack** as a key measure that characterizes the relative importance of a packet.
  - We propose **Aérgia** a novel architecture to accelerate low slack critical packets
- Result
  - Improves system performance: 16.1%
  - Improves network fairness: 30.8%

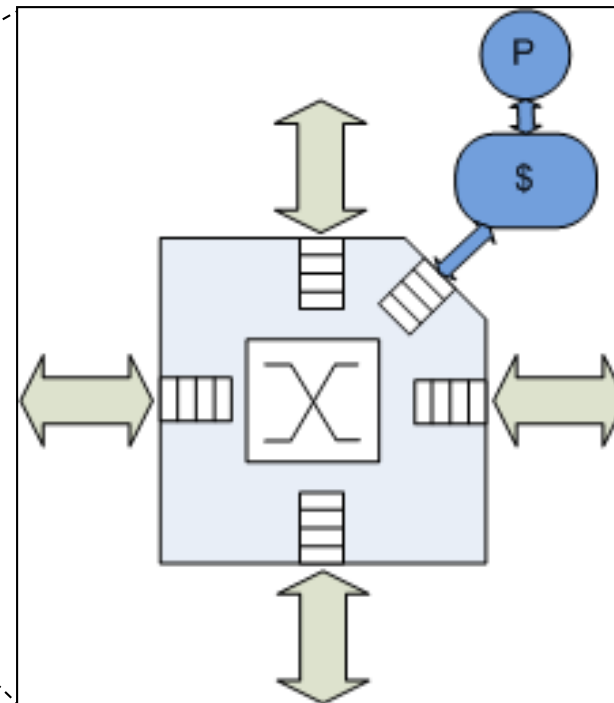
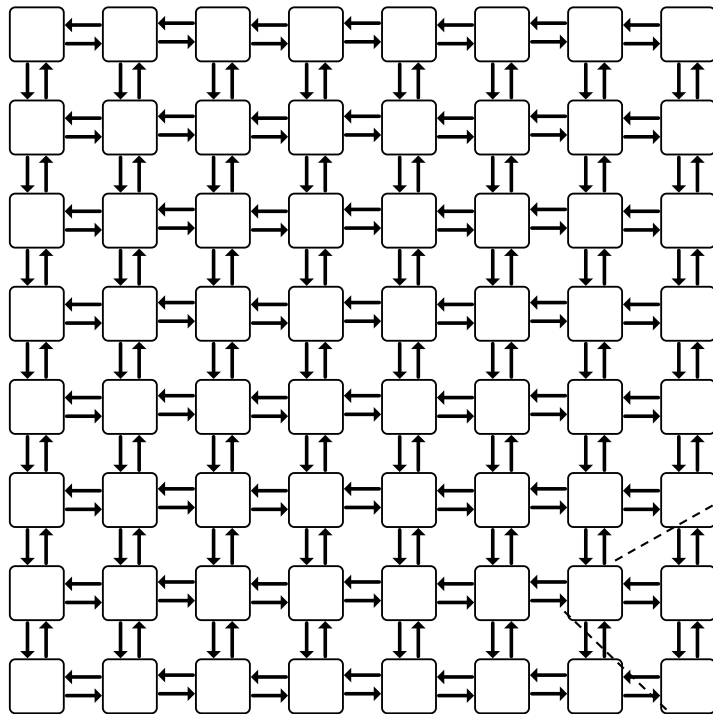


# Express-Cube Topologies

Boris Grot, Joel Hestness, Stephen W. Keckler, and Onur Mutlu,  
**"Express Cube Topologies for On-Chip Interconnects"**  
*Proceedings of the 15th International Symposium on High-Performance  
Computer Architecture (HPCA)*, pages 163-174, Raleigh, NC, February 2009.  
Slides (ppt)

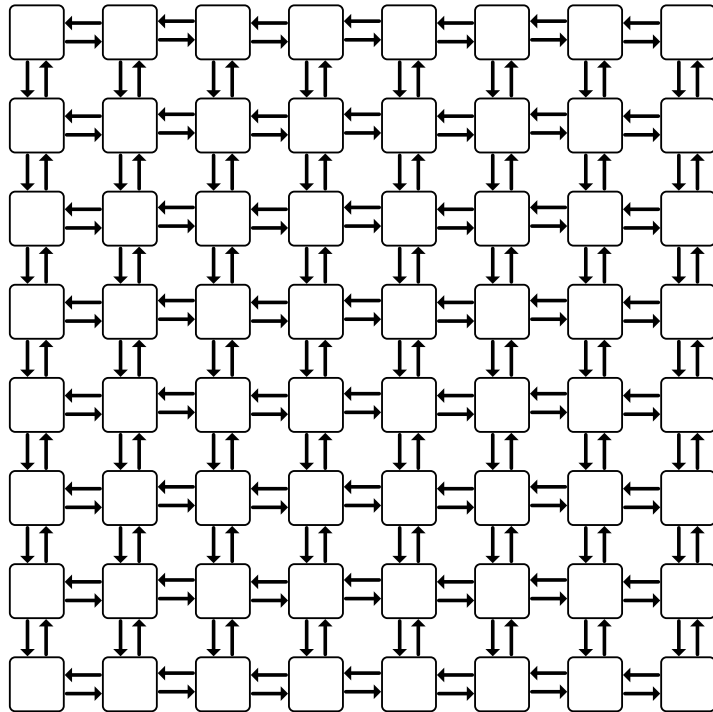
# 2-D Mesh

---



# 2-D Mesh

---



## □ Pros

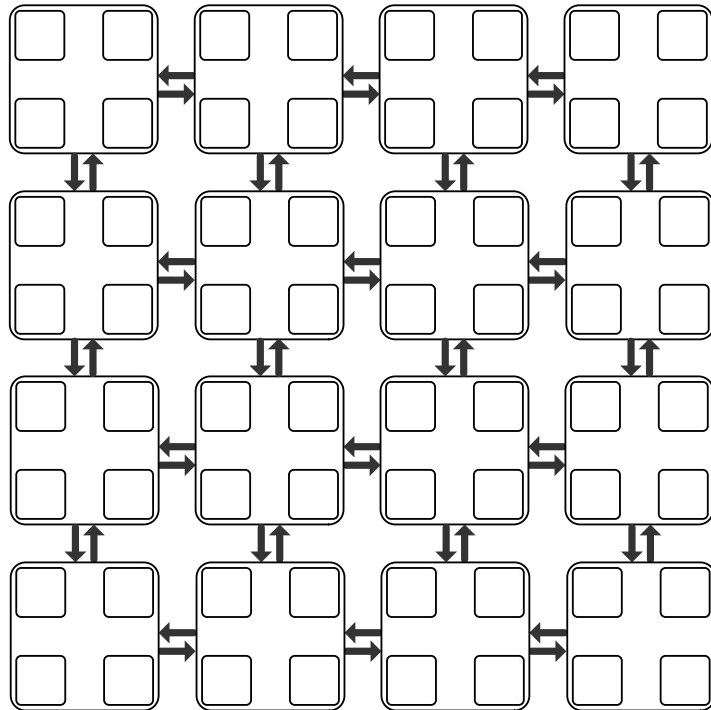
- Low design & layout complexity
- Simple, fast routers

## □ Cons

- Large diameter
- Energy & latency impact

# Concentration (*Balfour & Dally, ICS '06*)

---



## □ Pros

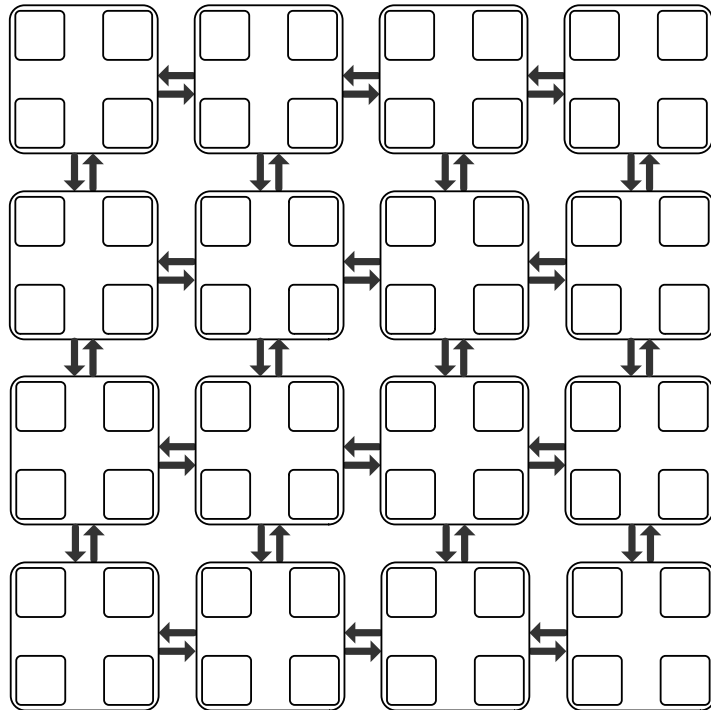
- Multiple *terminals* attached to a router node
- Fast nearest-neighbor communication via the crossbar
- Hop count reduction proportional to *concentration* degree

## □ Cons

- Benefits limited by crossbar complexity

# Concentration

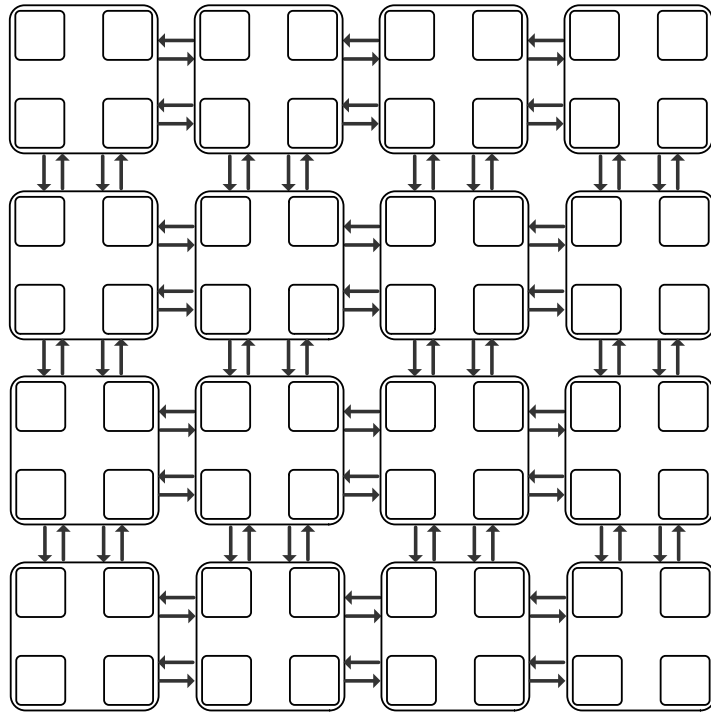
---



- Side-effects
  - Fewer channels
  - Greater channel width

# Replication

---



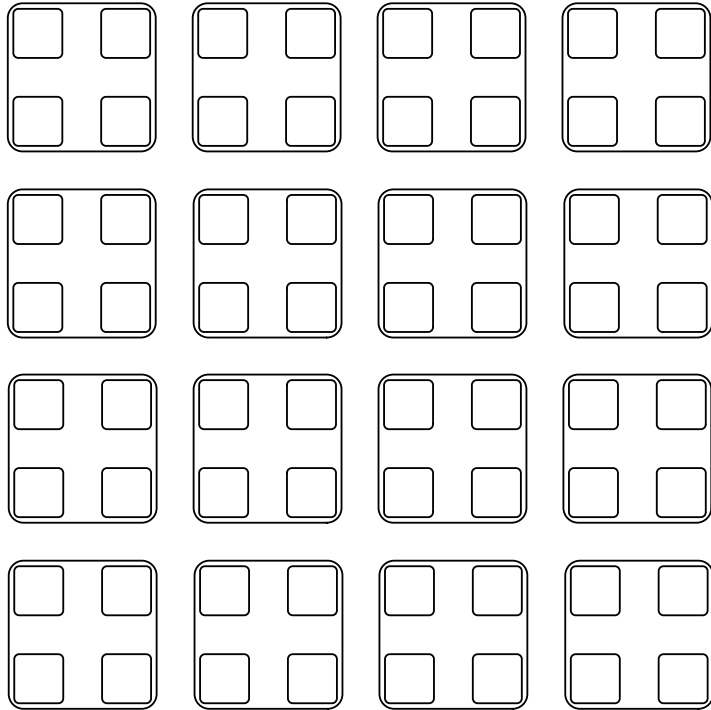
CMesh-X2

## ■ Benefits

- Restores bisection channel count
- Restores channel width
- Reduced crossbar complexity

# Flattened Butterfly (*Kim et al., Micro '07*)

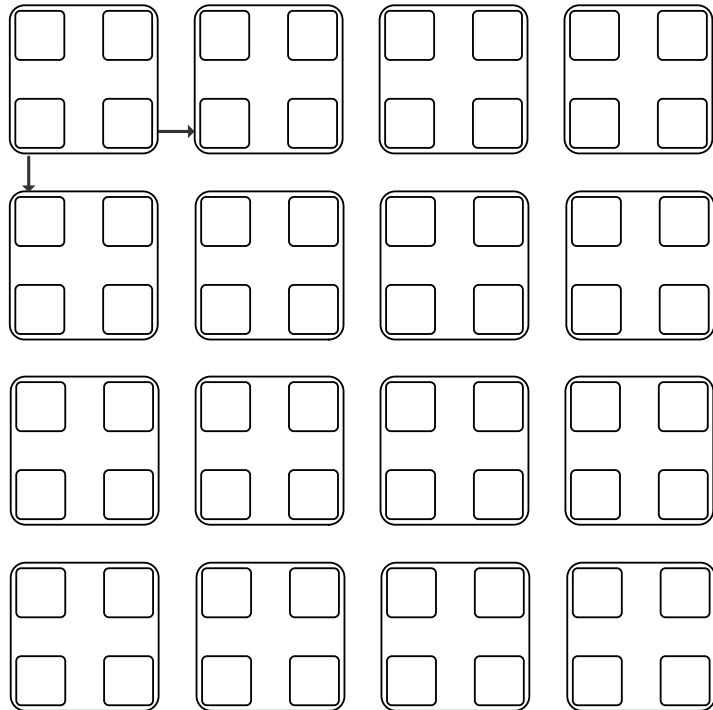
---



- Objectives:
  - Improve connectivity
  - Exploit the wire budget

# Flattened Butterfly (*Kim et al., Micro '07*)

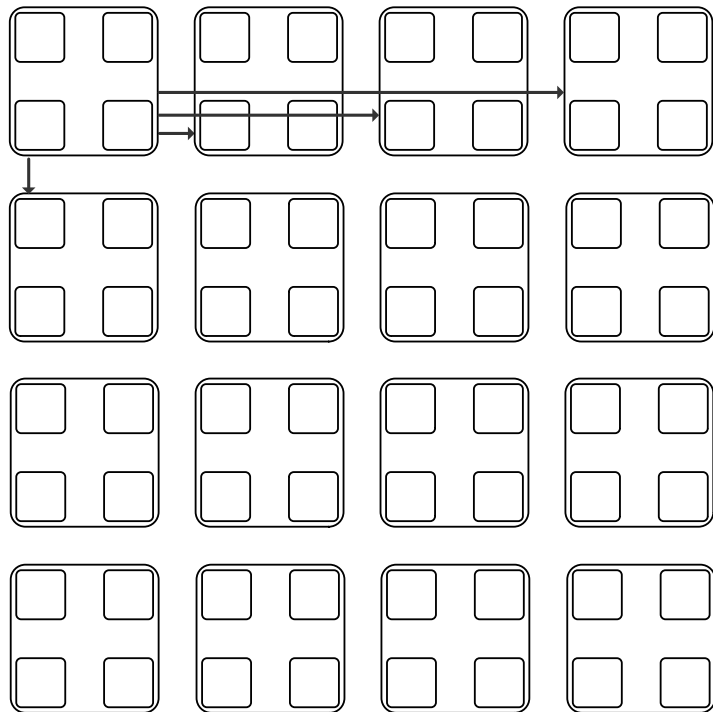
---





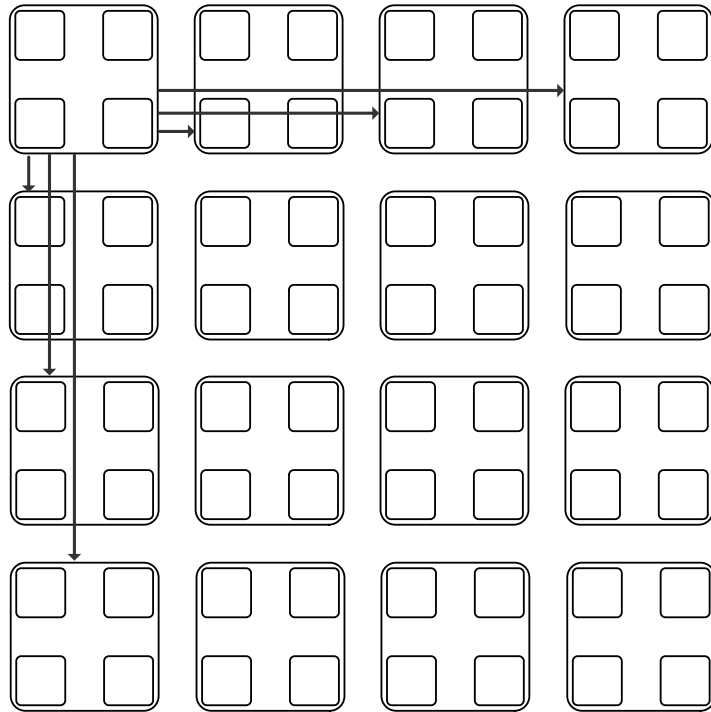
# Flattened Butterfly (*Kim et al., Micro '07*)

---



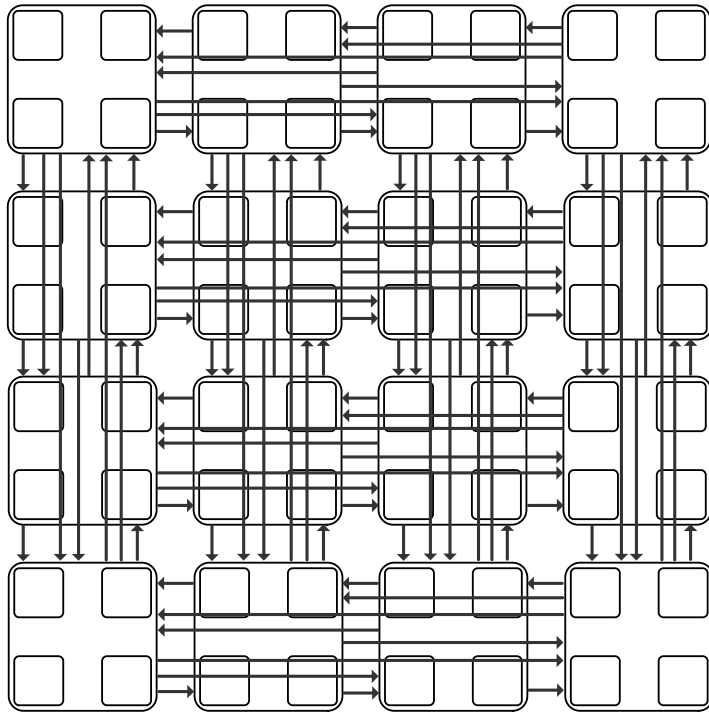
# Flattened Butterfly (*Kim et al., Micro '07*)

---



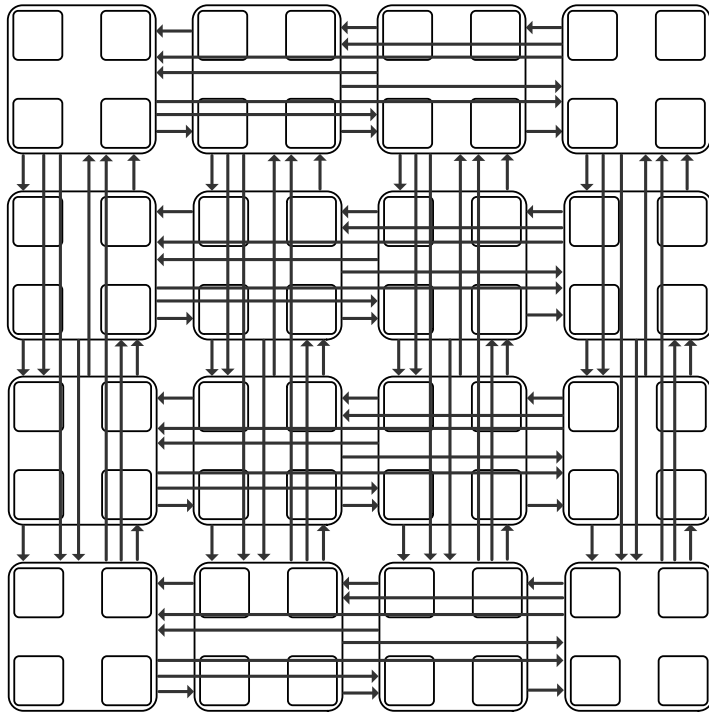
# Flattened Butterfly (*Kim et al., Micro '07*)

---



# Flattened Butterfly (*Kim et al., Micro '07*)

---



## □ Pros

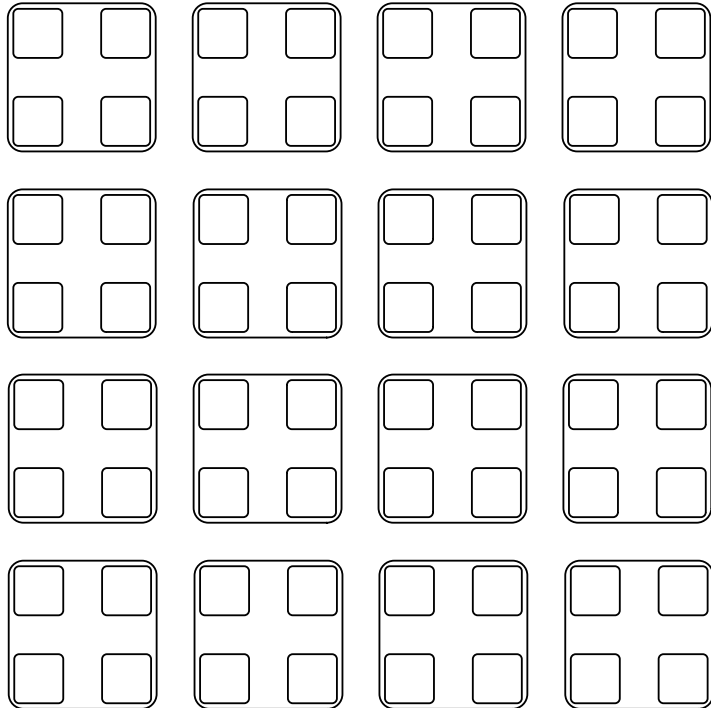
- Excellent connectivity
- Low diameter: 2 hops

## □ Cons

- High channel count:  $k^2/2$  per row/column
- Low channel utilization
- Increased control (arbitration) complexity

# Multidrop Express Channels (MECS)

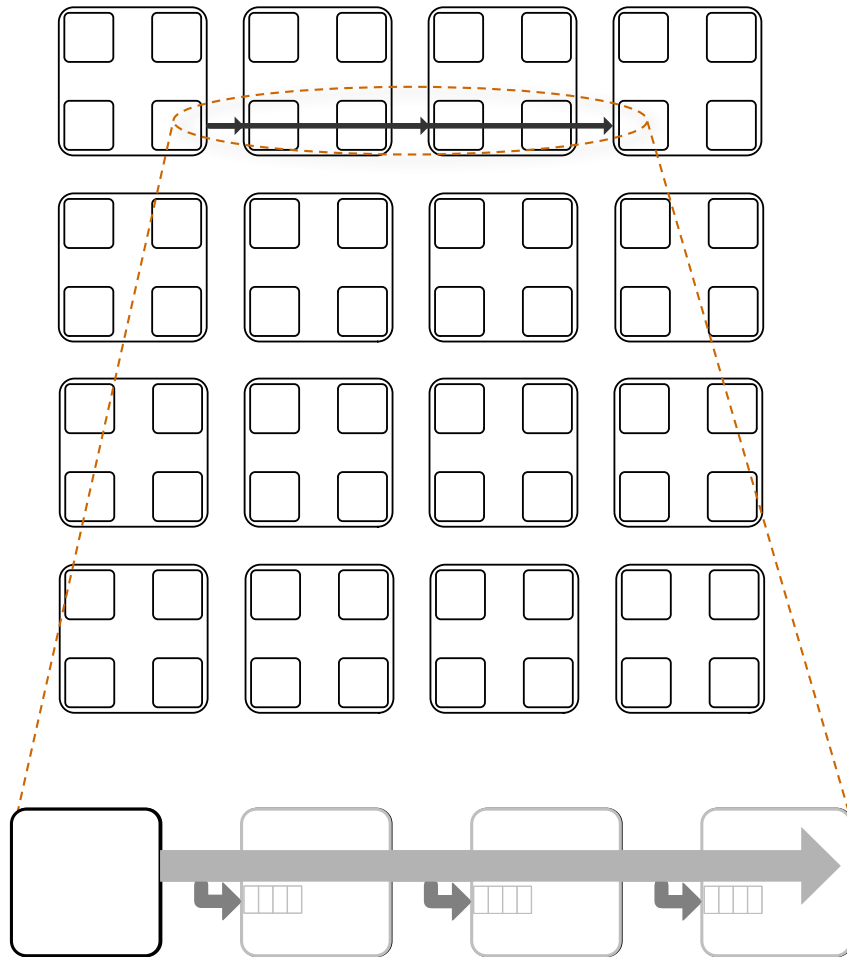
---



- Objectives:
  - Connectivity
  - More scalable channel count
  - Better channel utilization

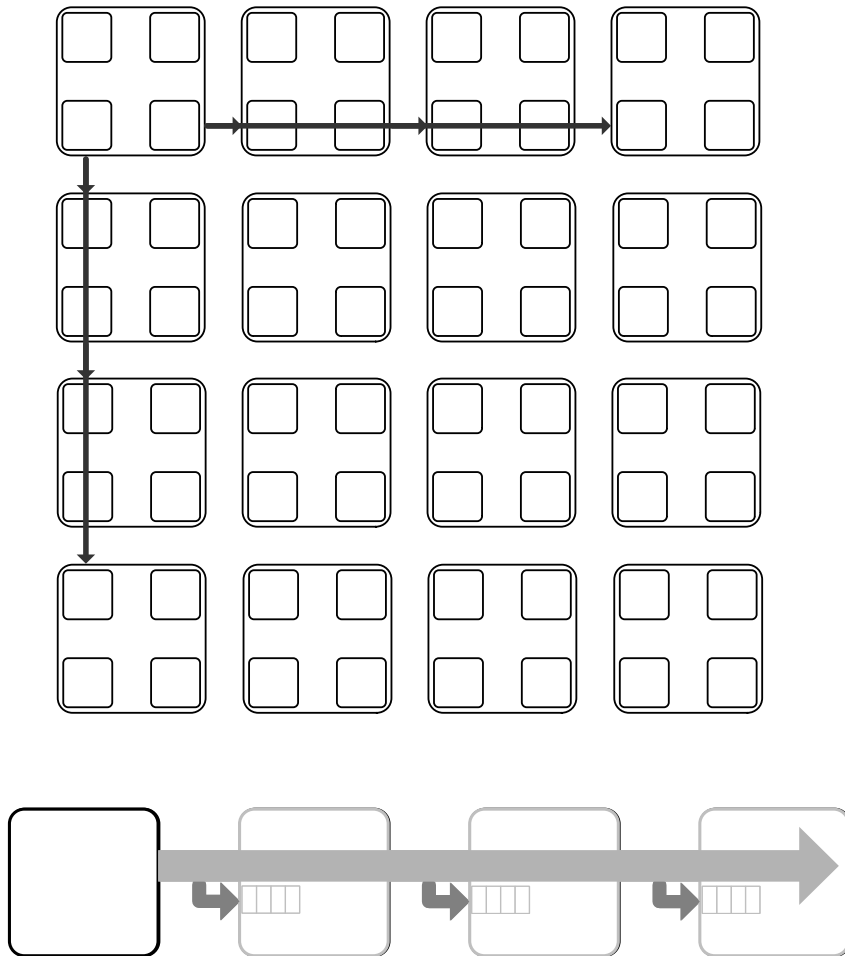
# Multidrop Express Channels (MECS)

---



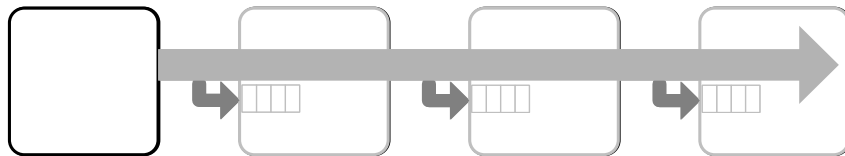
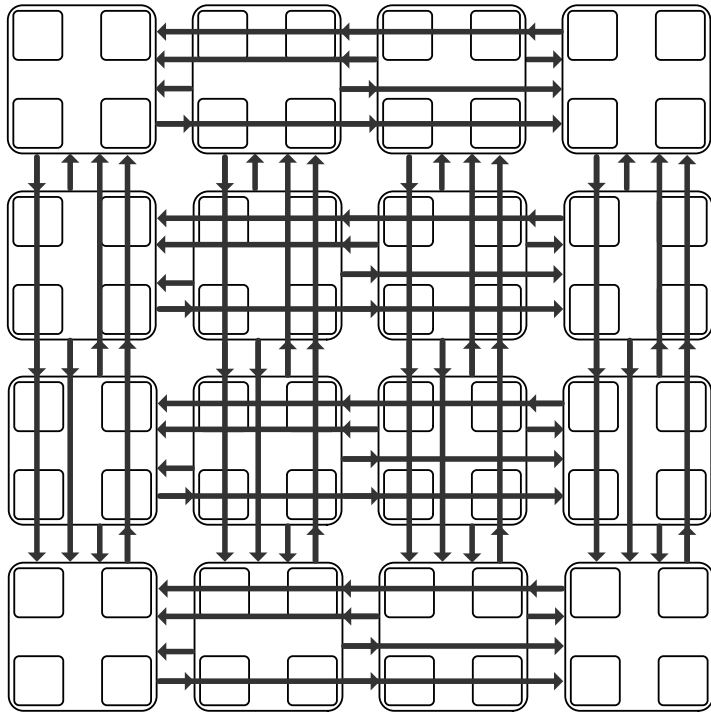
# Multidrop Express Channels (MECS)

---



# Multidrop Express Channels (MECS)

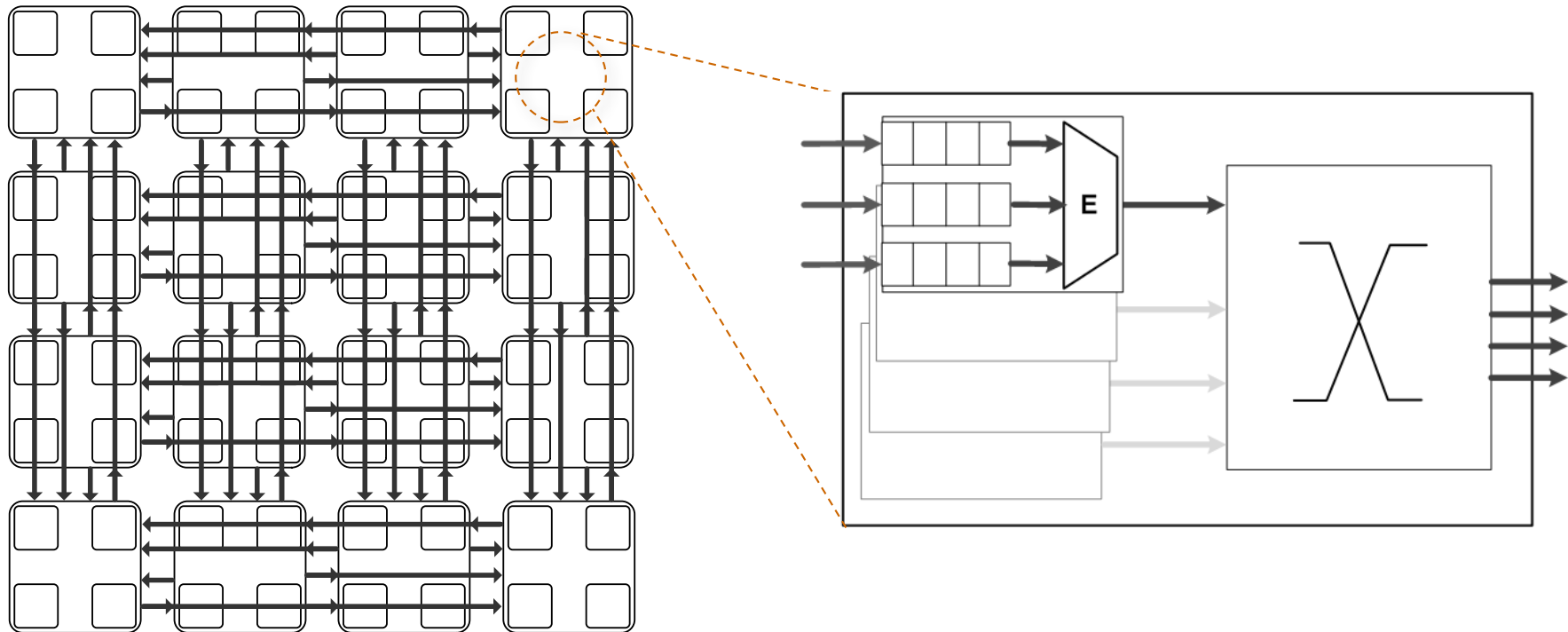
---





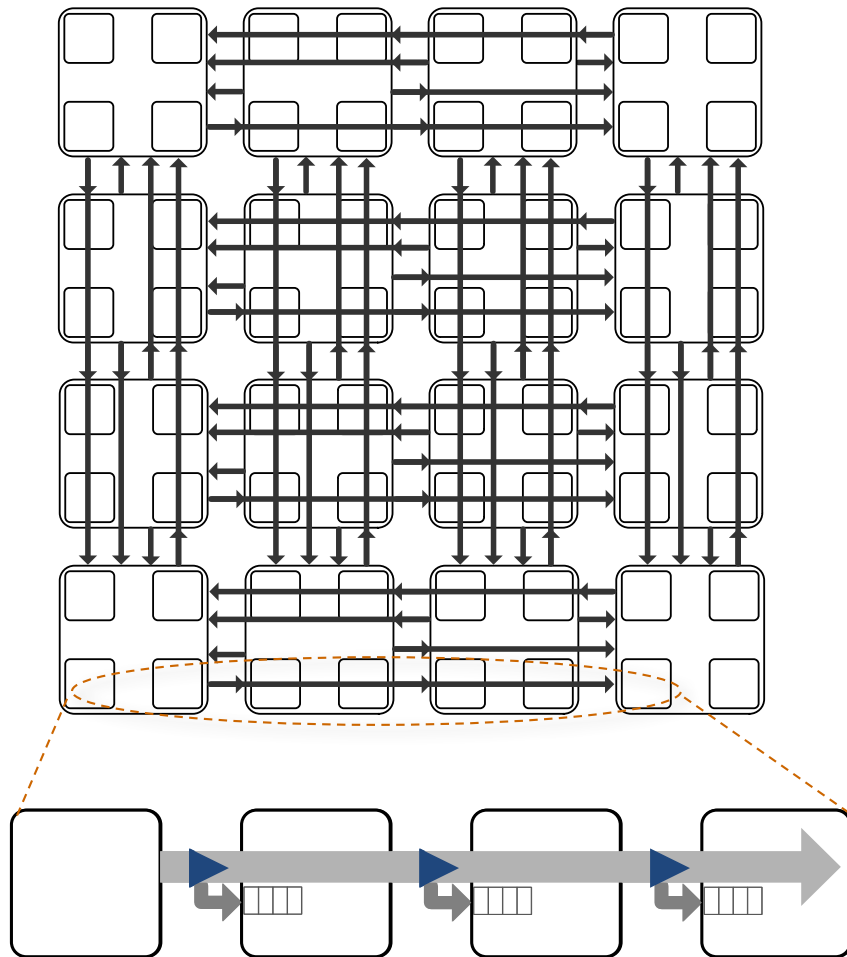
# Multidrop Express Channels (MECS)

---



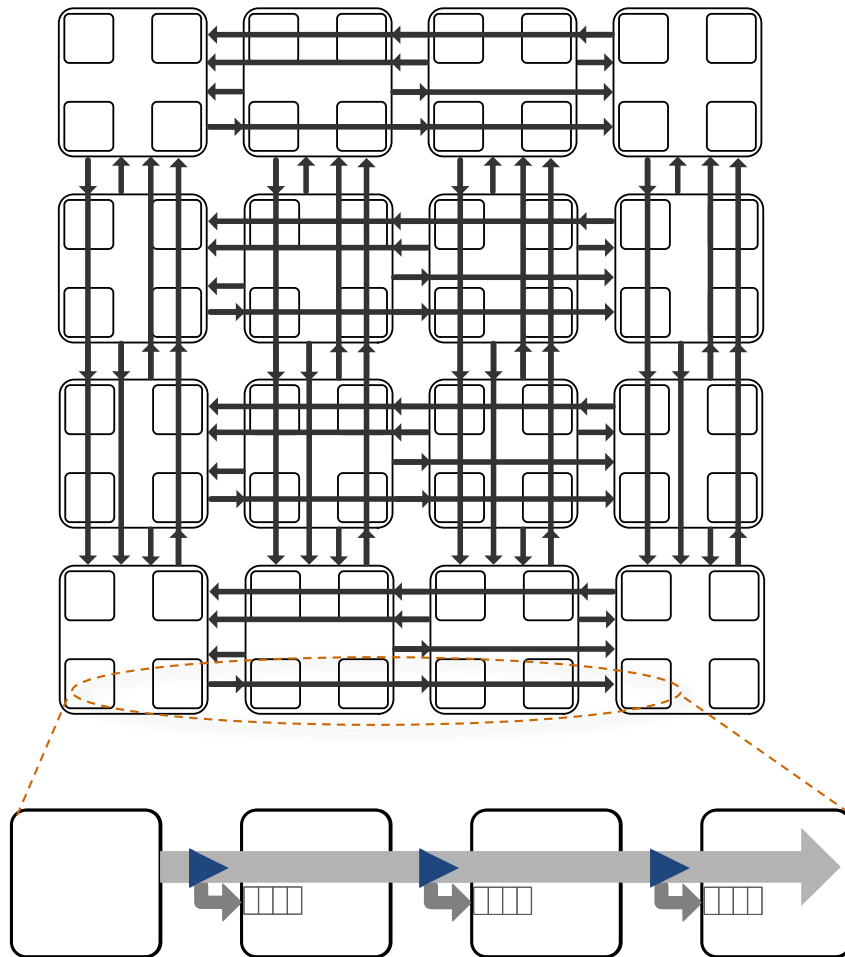
# Multidrop Express Channels (MECS)

---



# Multidrop Express Channels (MECS)

---



## □ Pros

- One-to-many topology
- Low diameter: 2 hops
- $k$  channels row/column
- Asymmetric

## □ Cons

- Asymmetric
- Increased control (arbitration) complexity

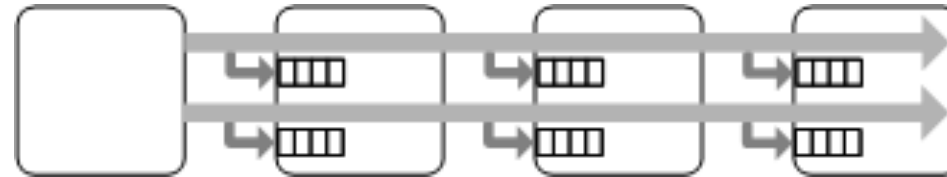
# Partitioning: a GEC Example

---

**MECS**



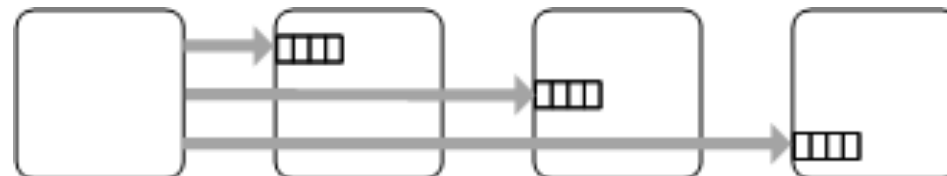
**MECS-X2**



**Partitioned  
MECS**



**Flattened  
Butterfly**



# Analytical Comparison

---

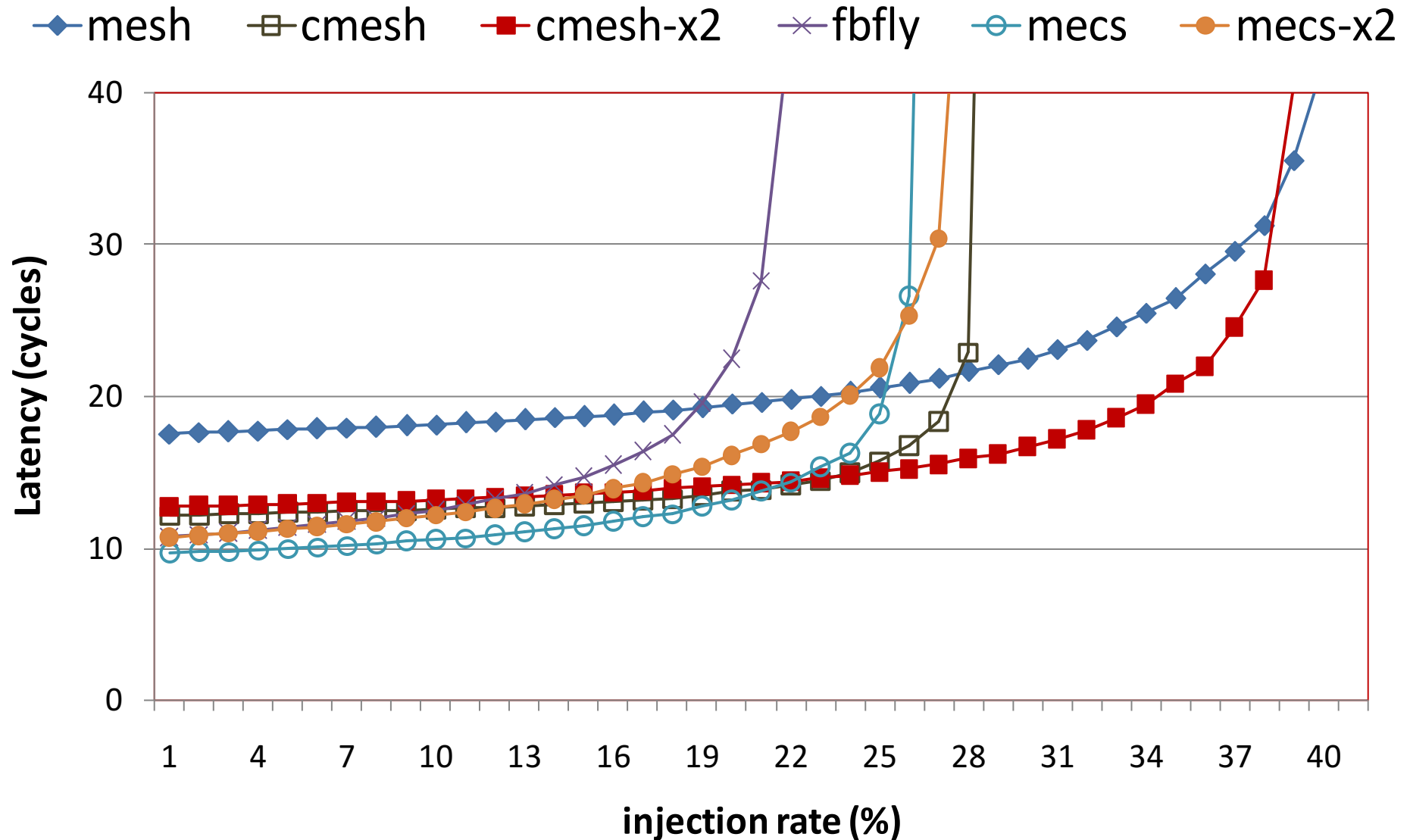
	CMesh		FBfly		MECS	
Network Size	64	256	64	256	64	256
Radix (conctr' d)	4	8	4	8	4	8
Diameter	6	14	2	2	2	2
Channel count	2	2	8	32	4	8
Channel width	576	1152	144	72	288	288
Router inputs	4	4	6	14	6	14
Router outputs	4	4	6	14	4	4

# Experimental Methodology

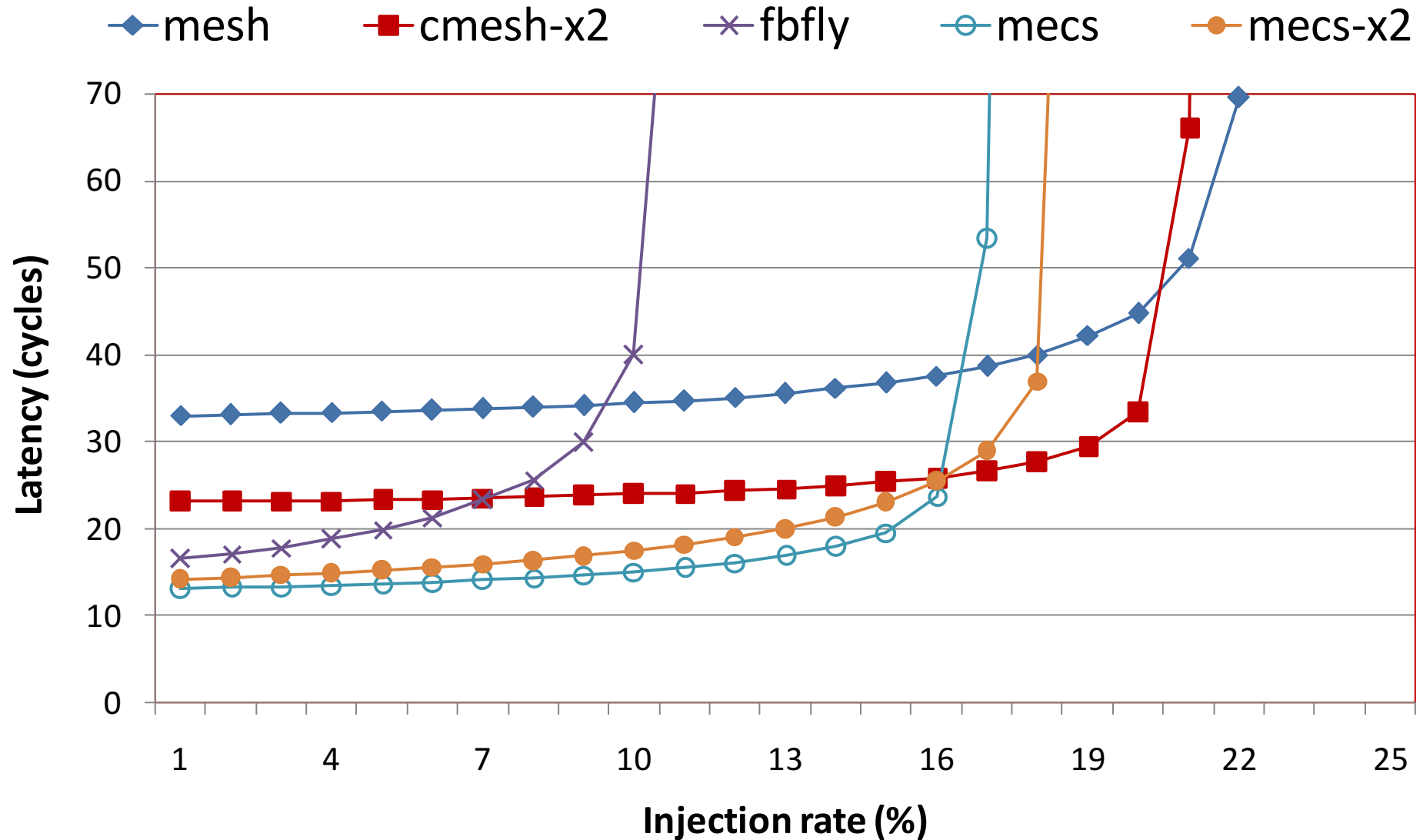
---

<b>Topologies</b>	Mesh, CMesh, CMesh-X2, FBFly, MECS, MECS-X2
<b>Network sizes</b>	64 & 256 terminals
<b>Routing</b>	DOR, adaptive
<b>Messages</b>	64 & 576 bits
<b>Synthetic traffic</b>	Uniform random, bit complement, transpose, self-similar
<b>PARSEC benchmarks</b>	Blackscholes, Bodytrack, Canneal, Ferret, Fluidanimate, Freqmine, Vip, x264
<b>Full-system config</b>	M5 simulator, Alpha ISA, 64 OOO cores
<b>Energy evaluation</b>	Orion + CACTI 6

# 64 nodes: Uniform Random

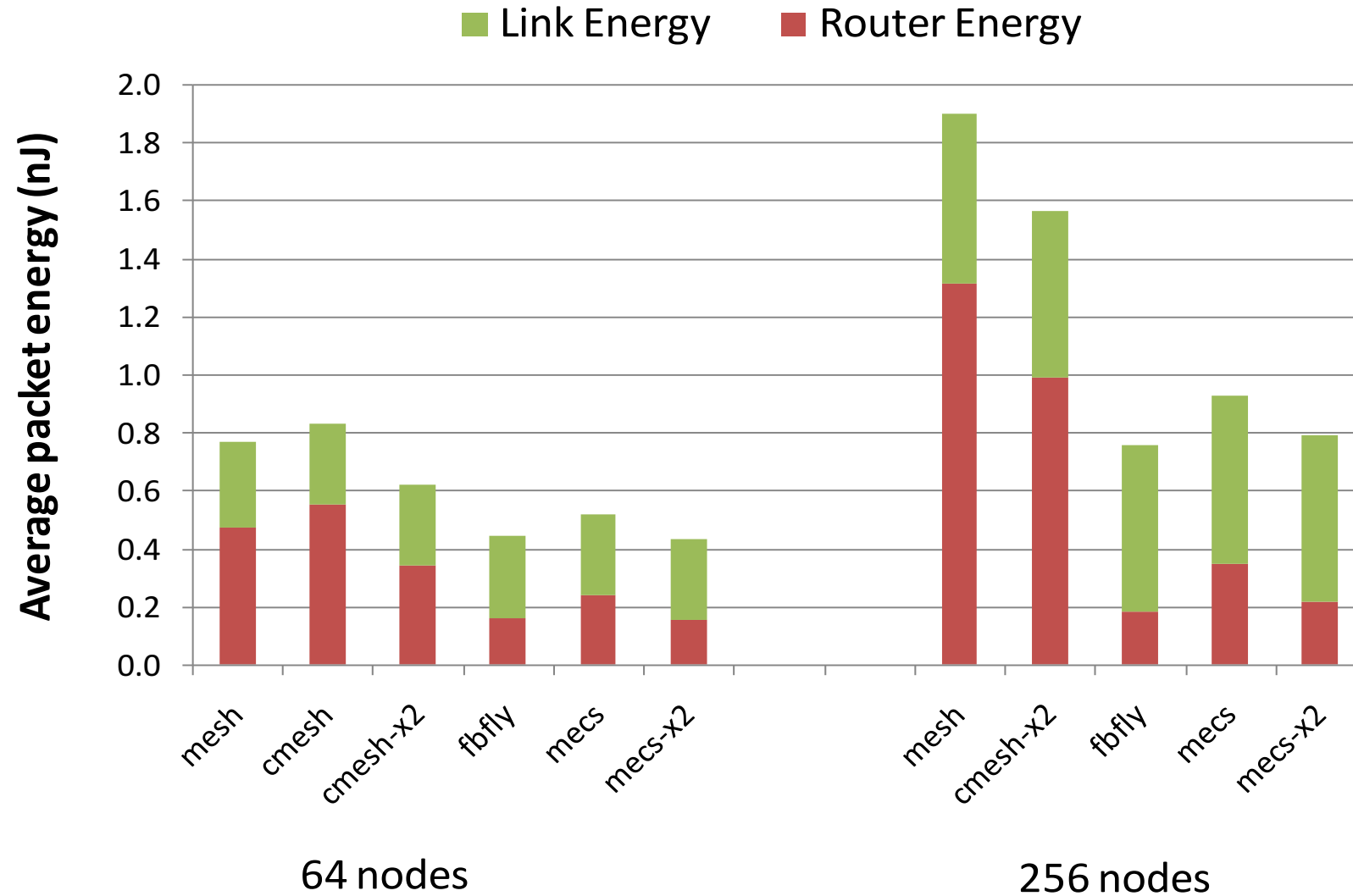


# 256 nodes: Uniform Random

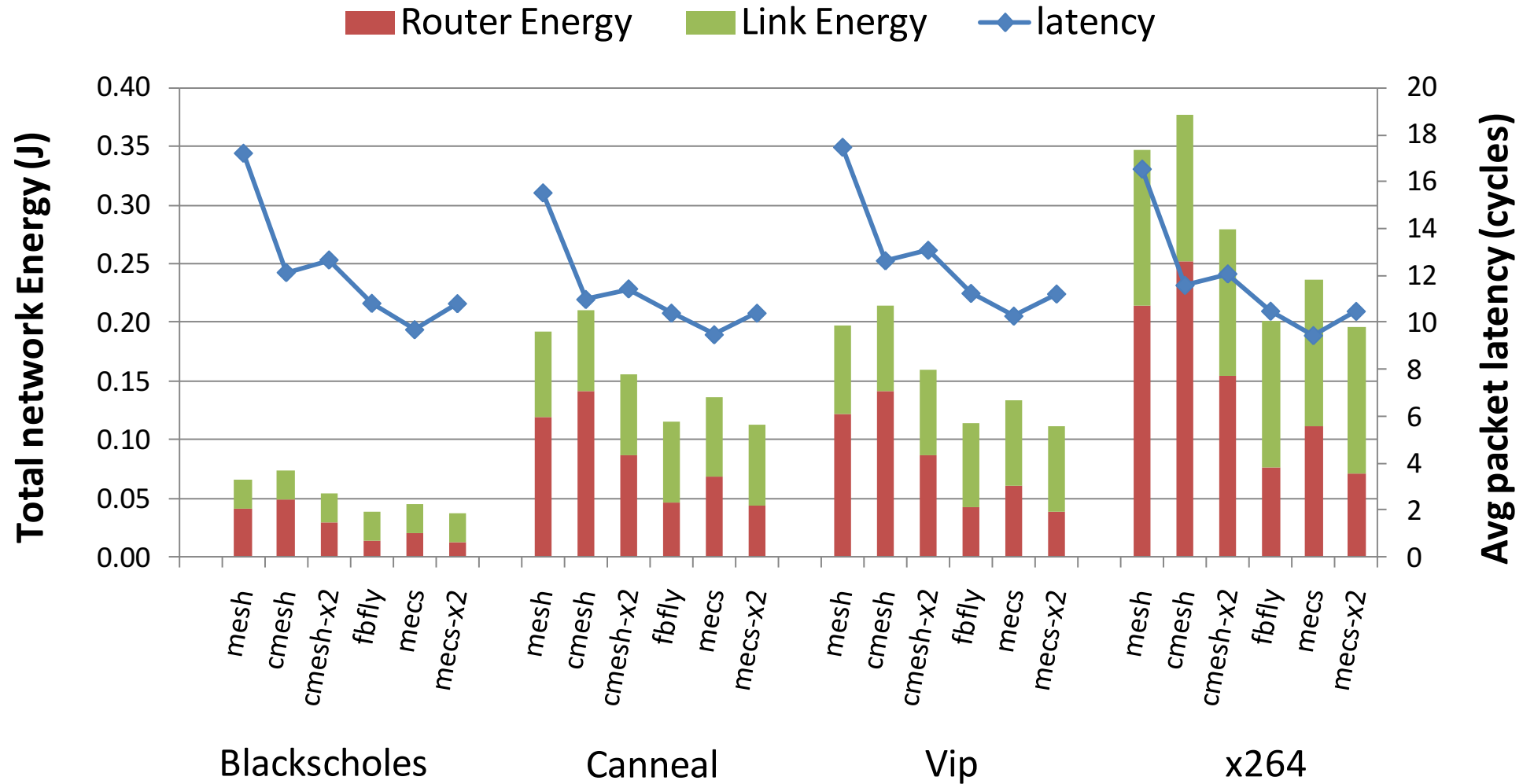




# Energy (100K pkts, Uniform Random)



# 64 Nodes: PARSEC



# Summary

---

## □ MECS

- A new one-to-many topology
- Good fit for planar substrates
- Excellent connectivity
- Effective wire utilization

## □ Generalized Express Cubes

- Framework & taxonomy for NOC topologies
- Extension of the k-ary n-cube model
- Useful for understanding and exploring on-chip interconnect options
- Future: expand & formalize

# Kilo-NoC: Topology-Aware QoS

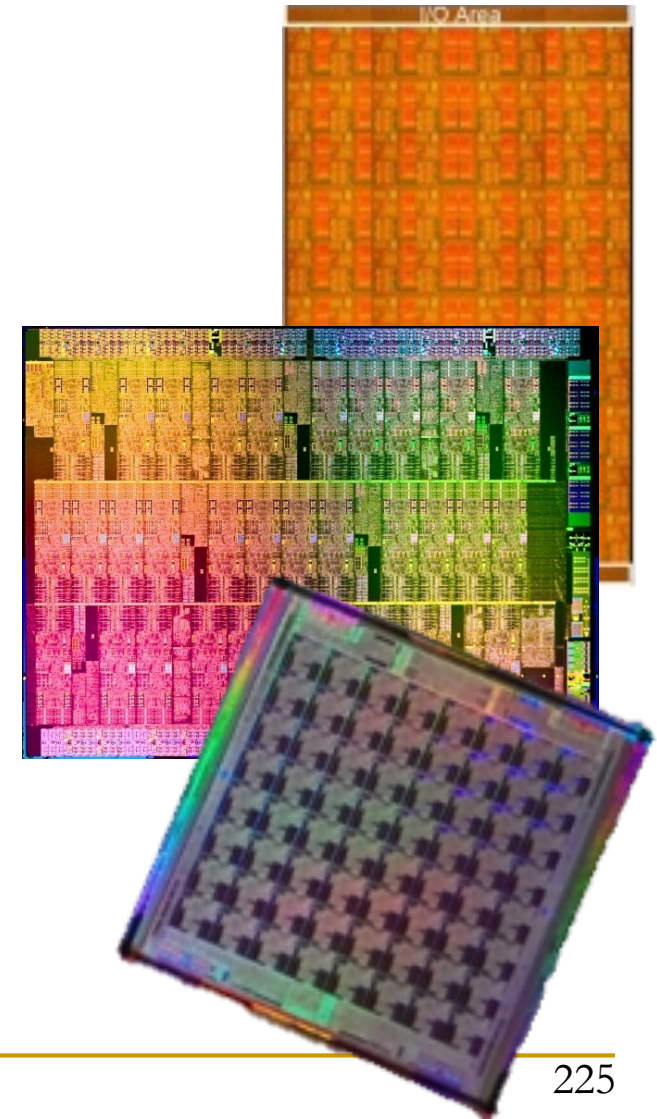
Boris Grot, Joel Hestness, Stephen W. Keckler, and Onur Mutlu,  
**"Kilo-NOC: A Heterogeneous Network-on-Chip Architecture for Scalability and Service Guarantees"**

*Proceedings of the 38th International Symposium on Computer Architecture (**ISCA**), San Jose, CA, June 2011. Slides (pptx)*

# Motivation

---

- Extreme-scale chip-level integration
  - Cores
  - Cache banks
  - Accelerators
  - I/O logic
  - Network-on-chip (NOC)
- 10-100 cores today
- 1000+ assets in the near future



# Kilo-NOC requirements

---

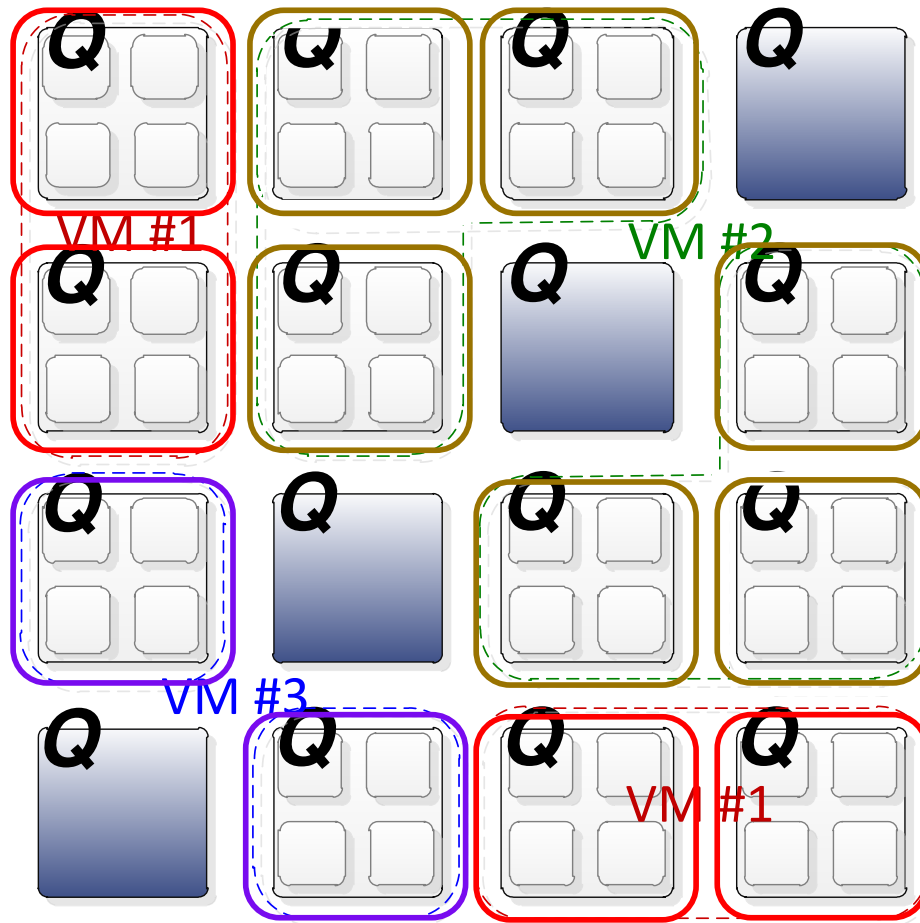
- High efficiency
  - Area
  - Energy
- Good performance
- Strong service guarantees (QoS)

# Topology-Aware QoS



---

- Problem: QoS support in each router is expensive (in terms of buffering, arbitration, bookkeeping)
  - E.g., Grot et al., “Preemptive Virtual Clock: A Flexible, Efficient, and Cost-effective QOS Scheme for Networks-on-Chip,” MICRO 2009.
- Goal: Provide QoS guarantees at low area and power cost
- Idea:
  - Isolate shared resources in a region of the network, support QoS within that area
  - Design the topology so that applications can access the region without interference

# Baseline QOS-enabled CMP

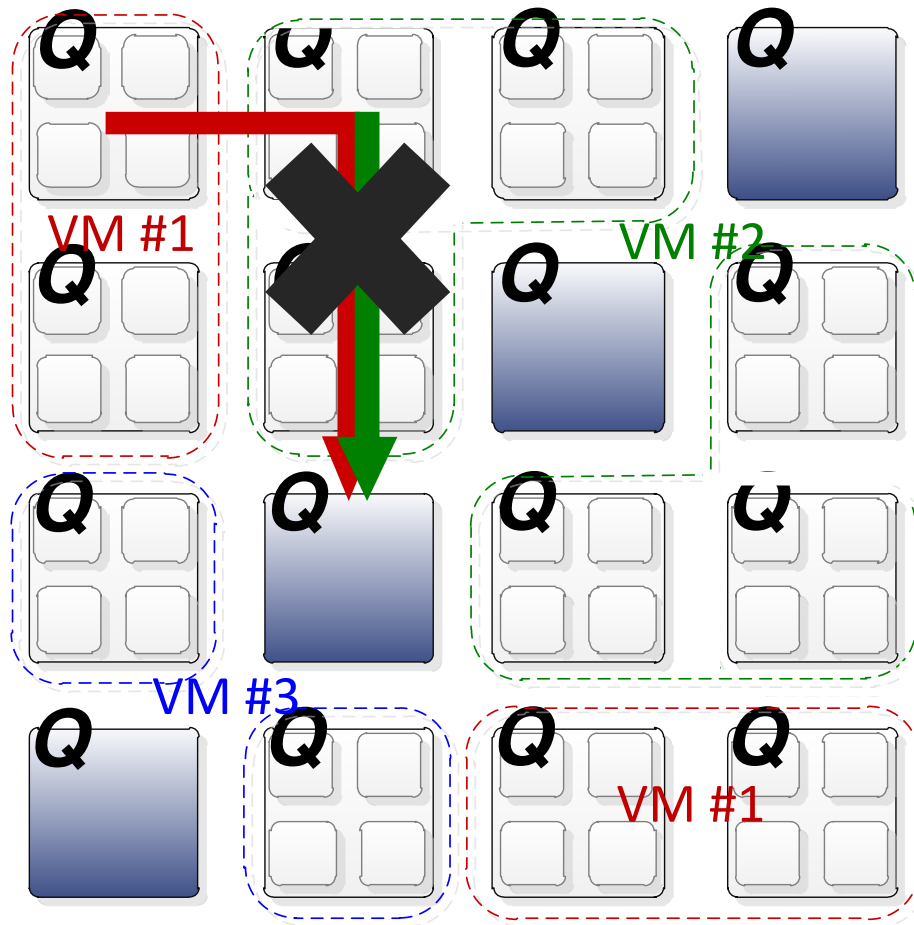


Multiple VMs  
sharing a die

-  Shared resources  
(e.g., memory controllers)
-  VM-private resources  
(cores, caches)
- Q** QOS-enabled router



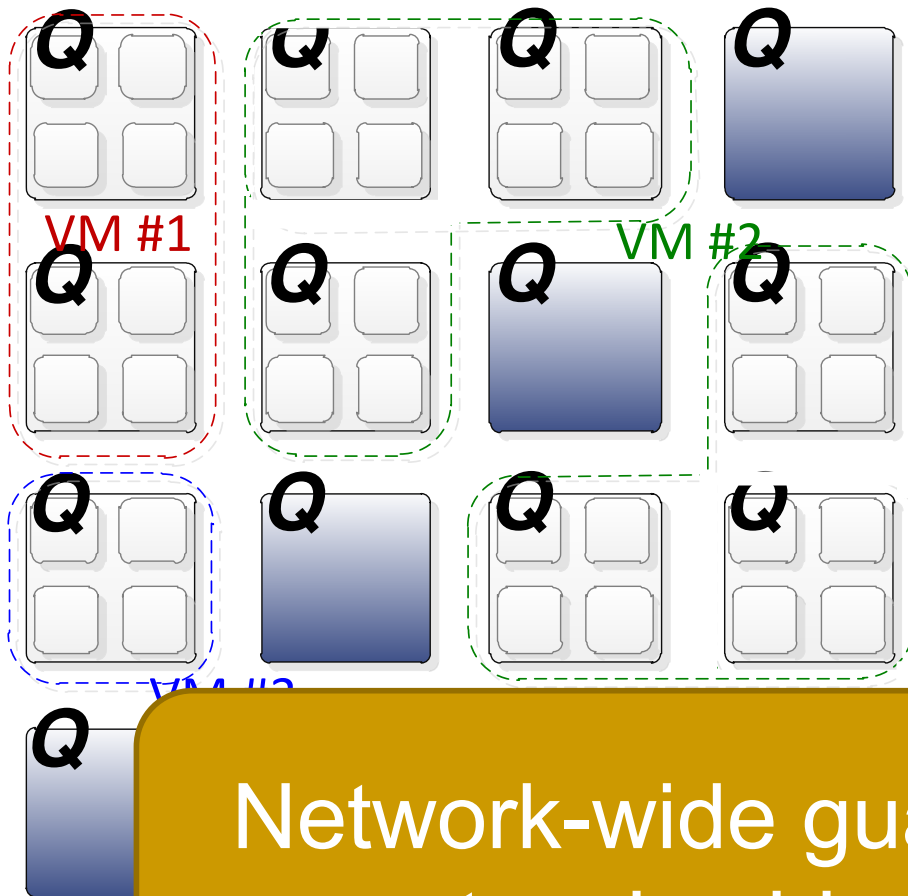
# Conventional NOC QOS



Contention scenarios:

- Shared resources
  - memory access
- Intra-VM traffic
  - shared cache access
- Inter-VM traffic
  - VM page sharing

# Conventional NOC QOS



Contention scenarios:

- Shared resources
  - memory access
- Intra-VM traffic
  - shared cache access
- Inter-VM traffic
  - VM page sharing

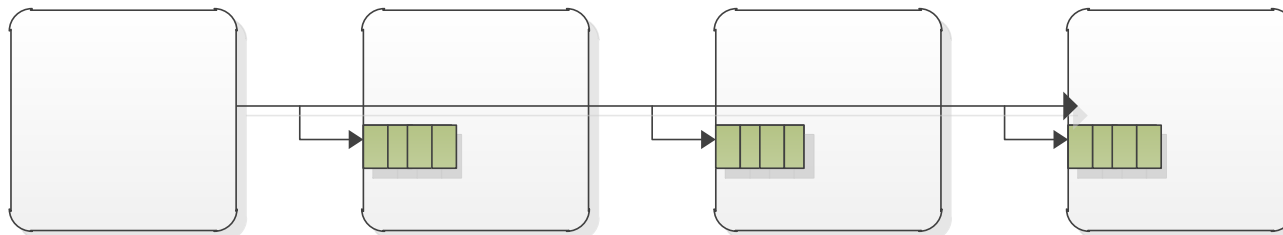
Network-wide guarantees *without* network-wide QOS support

# Kilo-NOC QOS

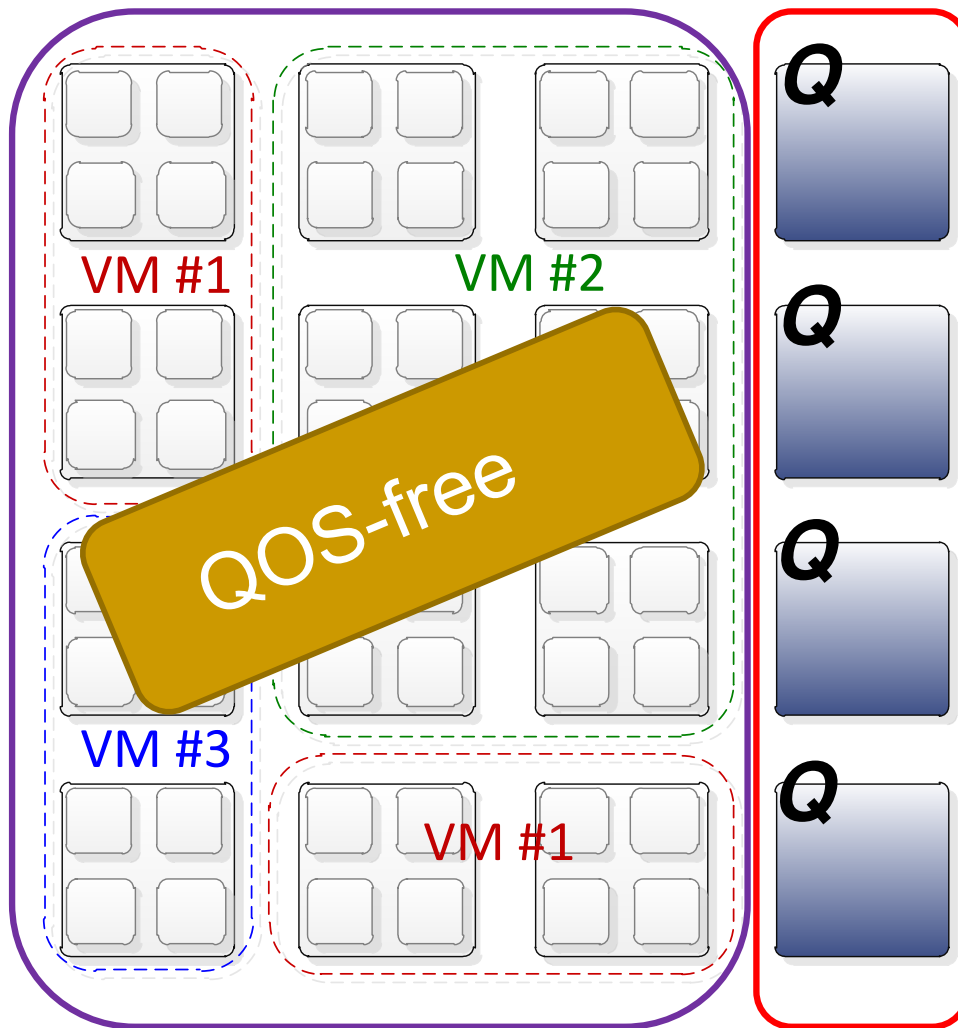
---

- Insight: leverage rich network connectivity
  - Naturally reduce interference among flows
  - Limit the extent of hardware QOS support
- Requires a low-diameter topology
  - This work: Multidrop Express Channels (MECS)

*Grot et al., HPCA  
2009*



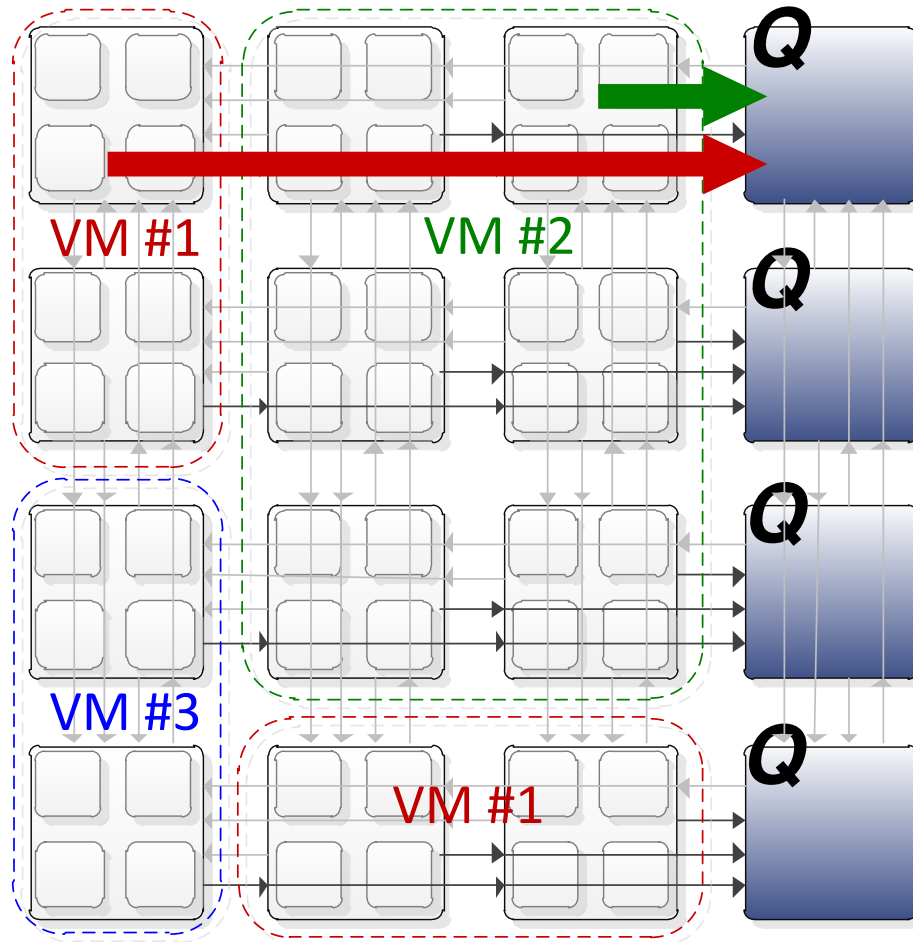
# Topology-Aware QOS



- Dedicated, QOS-enabled regions
  - Rest of die: QOS-free
- Richly-connected topology
  - Traffic isolation
- Special routing rules
  - Manage interference

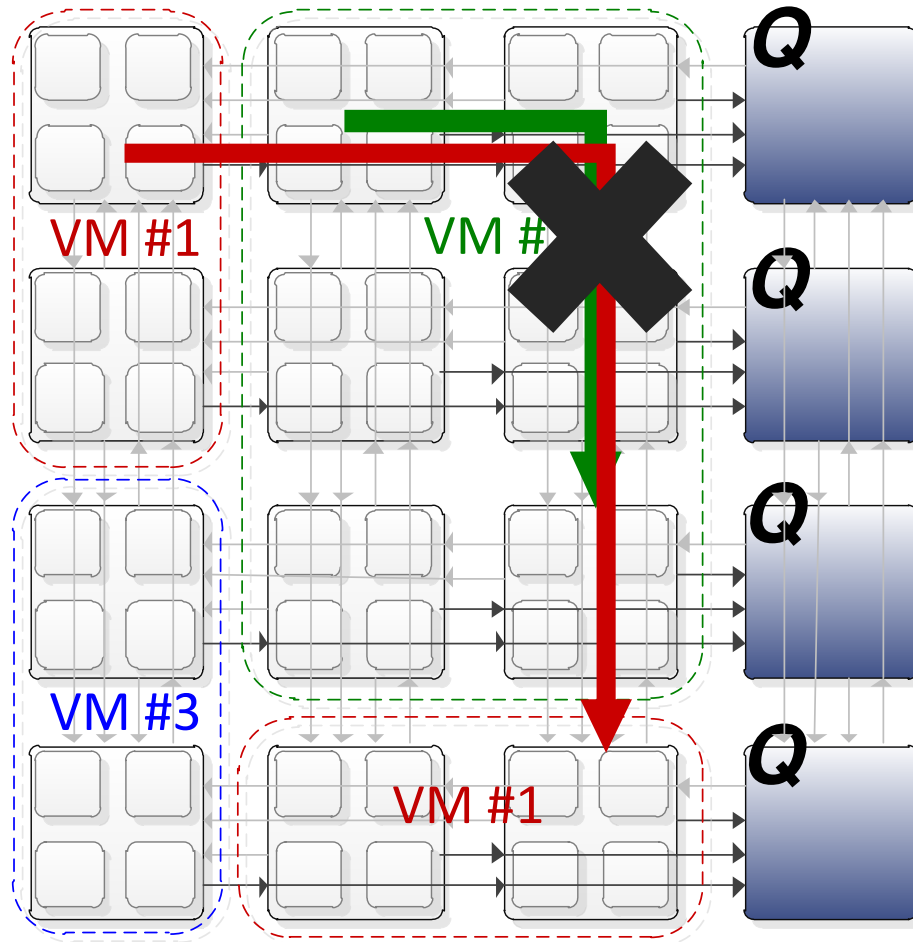
# Topology-Aware QOS

---



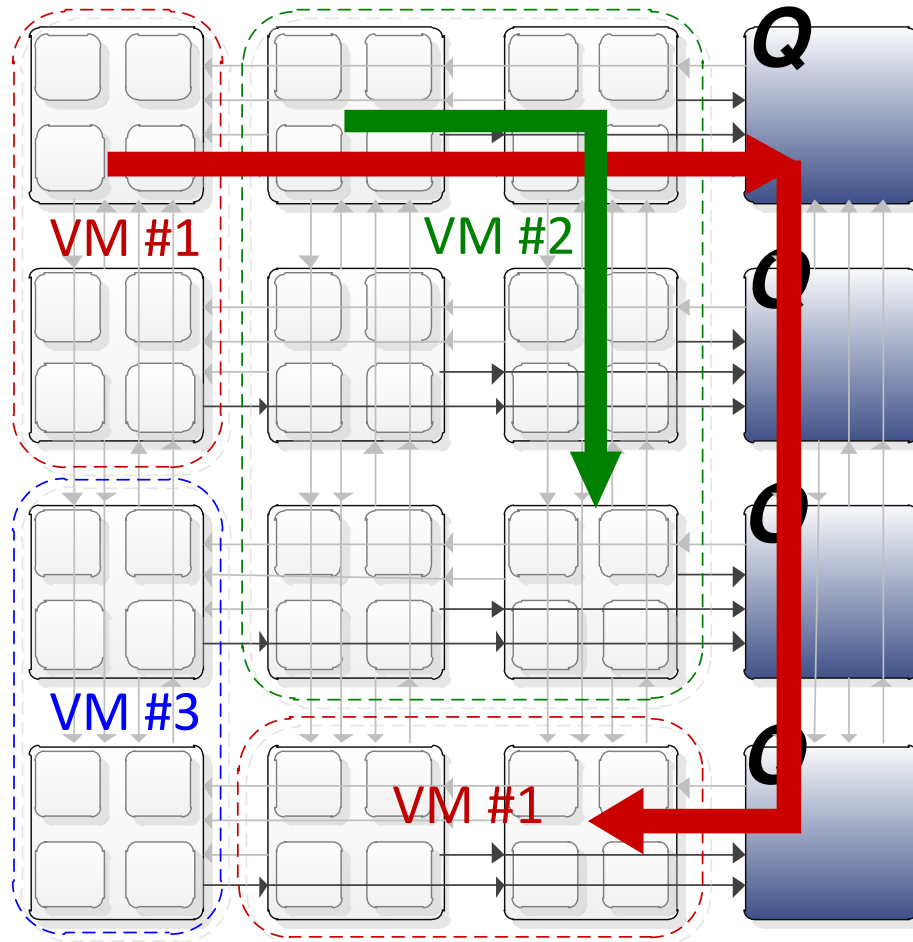
- Dedicated, QOS-enabled regions
  - Rest of die: QOS-free
- Richly-connected topology
  - Traffic isolation
- Special routing rules
  - Manage interference

# Topology-Aware QOS



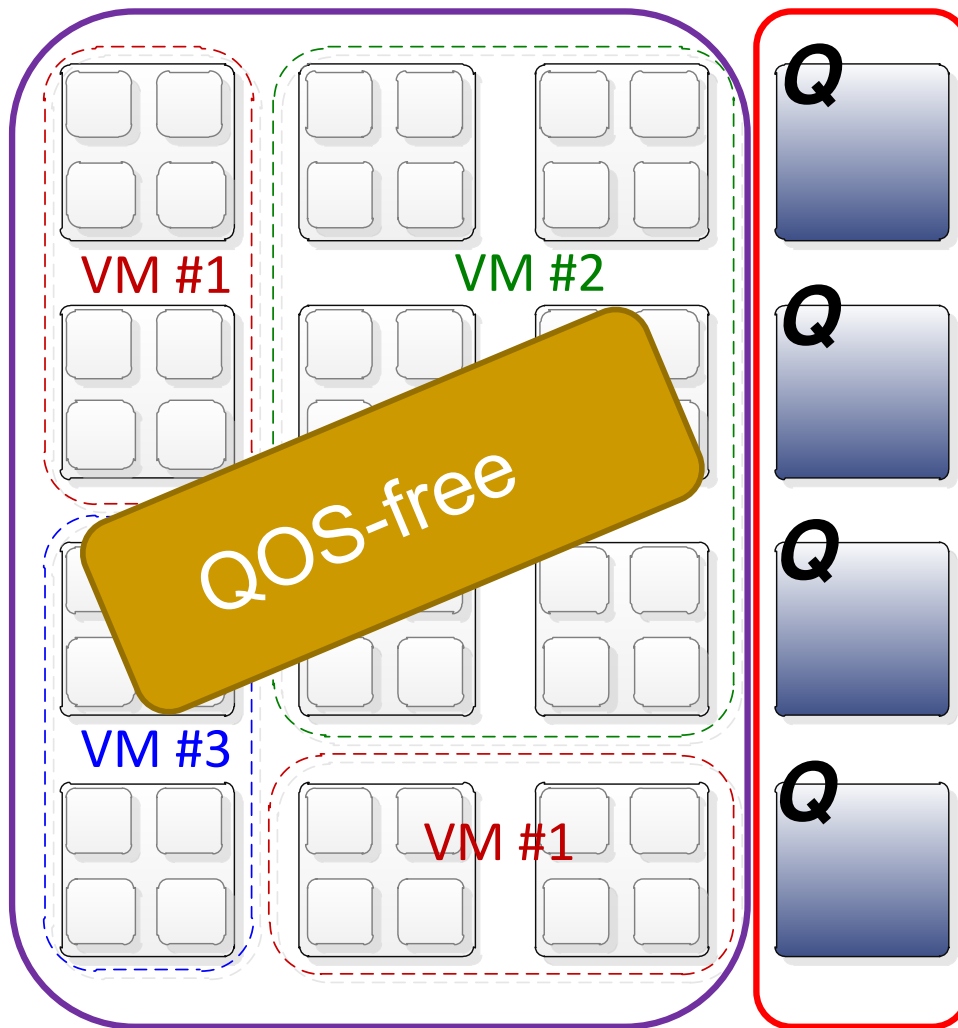
- Dedicated, QOS-enabled regions
  - Rest of die: QOS-free
- Richly-connected topology
  - Traffic isolation
- Special routing rules
  - Manage interference

# Topology-Aware QOS



- Dedicated, QOS-enabled regions
  - Rest of die: QOS-free
- Richly-connected topology
  - Traffic isolation
- Special routing rules
  - Manage interference

# Kilo-NOC view



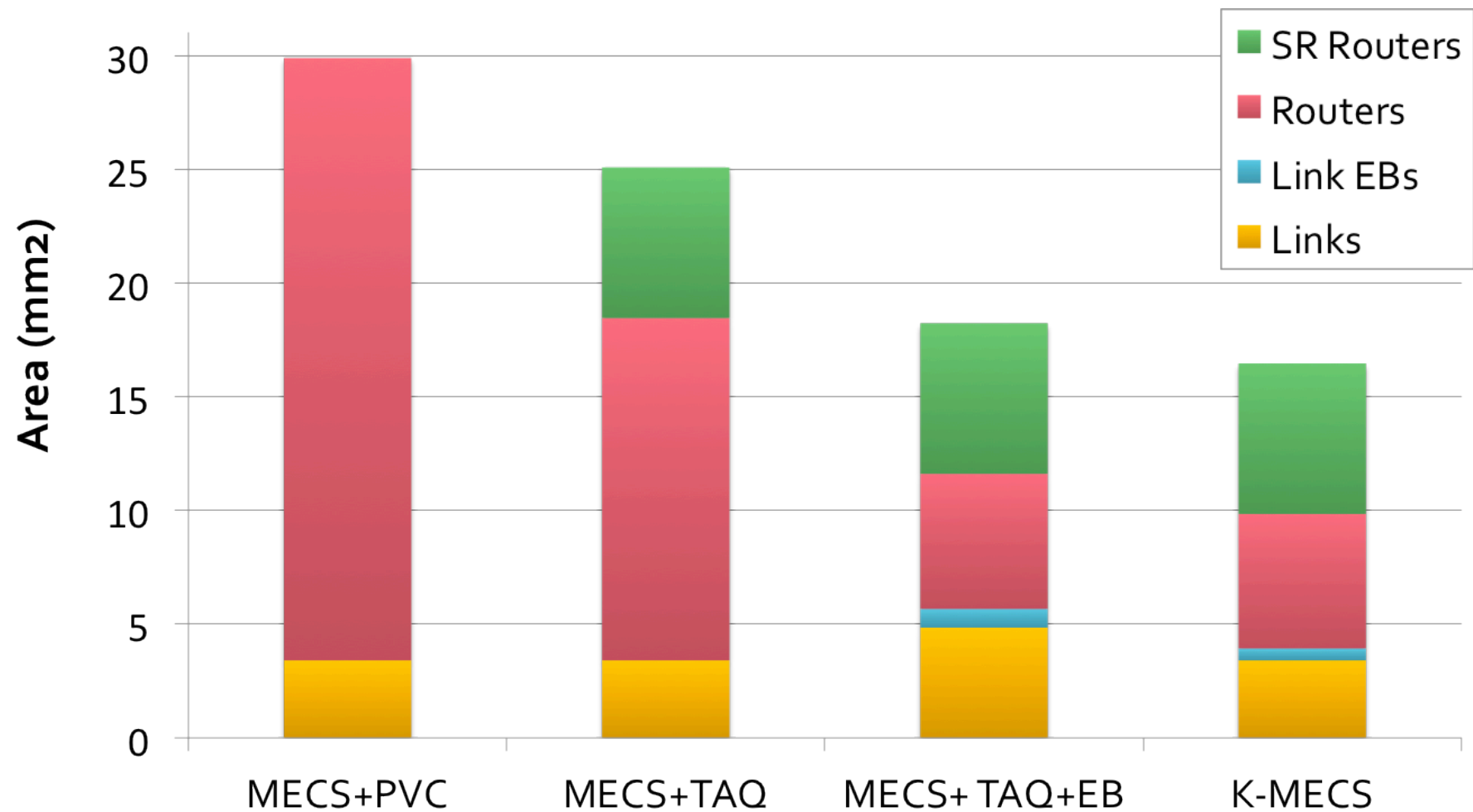
- Topology-aware QOS support
  - Limit QOS complexity to a fraction of the die
- Optimized flow control
  - Reduce buffer requirements in QOS-free regions



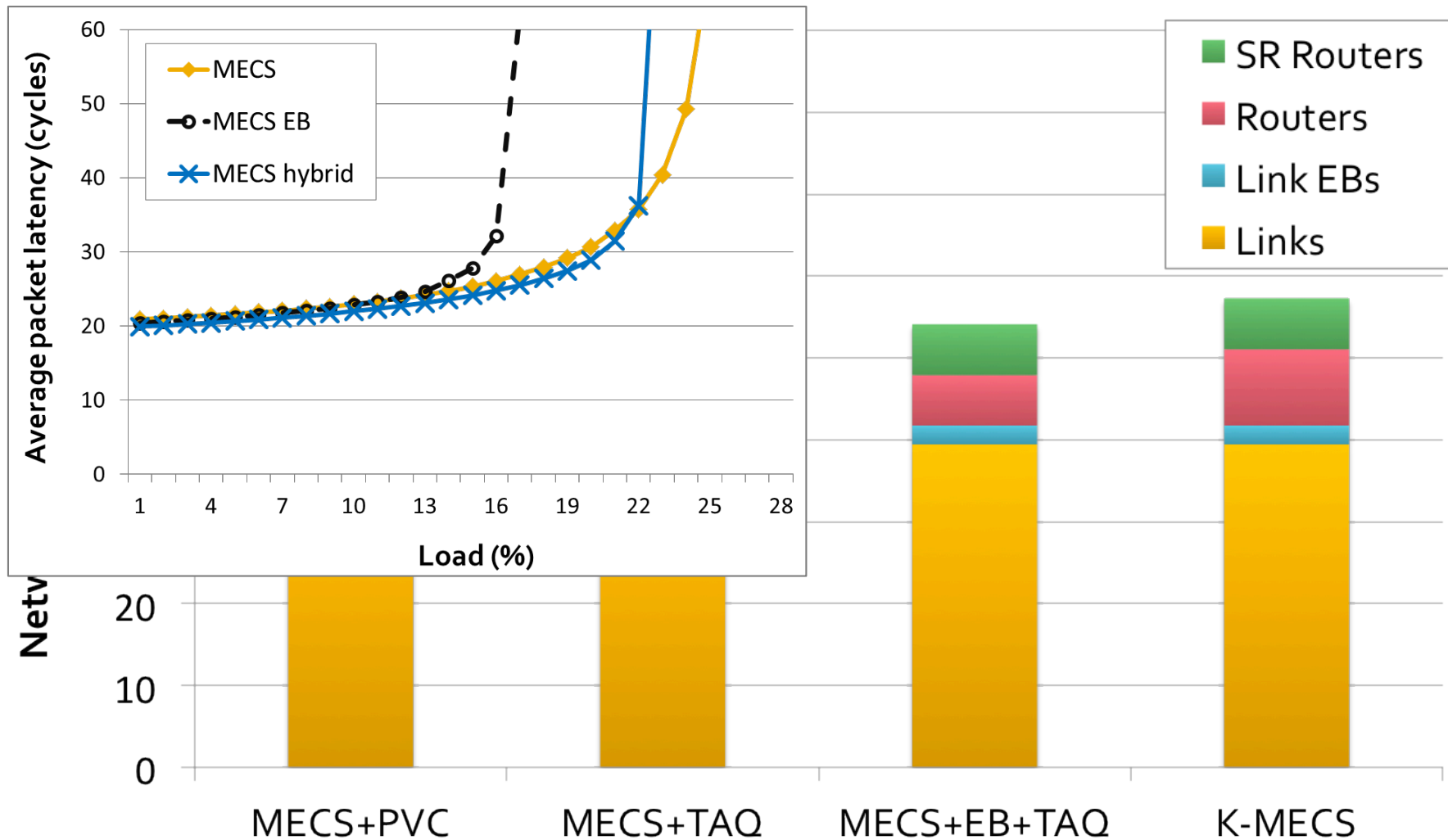
# Evaluation Methodology

Parameter	Value
Technology	15 nm
Vdd	0.7 V
System	1024 tiles: 256 concentrated nodes (64 shared resources)
<b>Networks:</b>	
MECS+PVC	VC flow control, QOS support (PVC) at each node
MECS+TAQ	VC flow control, QOS support only in shared regions
MECS+TAQ+EB	EB flow control outside of SRs, Separate <i>Request</i> and <i>Reply</i> networks
K-MECS	Proposed organization: TAQ + hybrid flow control

# Area comparison



# Energy comparison



# Summary

Kilo-NOC: a heterogeneous NOC architecture for kilo-node substrates

- Topology-aware QOS
  - Limits QOS support to a fraction of the die
  - Leverages low-diameter topologies
  - Improves NOC area- and energy-efficiency
  - Provides strong guarantees