18-742 Fall 2012
Parallel Computer Architecture
Lecture 21: Interconnects IV

Prof. Onur Mutlu
Carnegie Mellon University
10/29/2012

New Review Assignments

- Were Due: Sunday, October 28, 11:59pm.
  - Das et al., "Aergia: Exploiting Packet Latency Slack in On-Chip Networks," ISCA 2010.
  - Dennis and Misunas, "A Preliminary Architecture for a Basic Data Flow Processor," ISCA 1974.

- Due: Tuesday, October 30, 11:59pm.
  - Arvind and Nikhil, "Executing a Program on the MIT Tagged-Token Dataflow Architecture," IEEE TC 1990.

- Due: Thursday, November 1, 11:59pm.
  - Patt et al., "HPS, a new microarchitecture: rationale and introduction," MICRO 1985.
  - Patt et al., "Critical issues regarding HPS, a high performance microarchitecture," MICRO 1985.

Other Readings

- Dataflow
  - Gurd et al., "The Manchester prototype dataflow computer," CACM 1985.
  - Lee and Hurson, "Dataflow Architectures and Multithreading," IEEE Computer 1994.

- Restricted Dataflow
  - Patt et al., "HPS, a new microarchitecture: rationale and introduction," MICRO 1985.
  - Patt et al., "Critical issues regarding HPS, a high performance microarchitecture," MICRO 1985.
  - Sankaralingam et al., "Exploiting ILP, TLP and DLP with the Polymorphous TRIPS Architecture," ISCA 2003.
  - Burger et al., "Scaling to the End of Silicon with EDGE Architectures," IEEE Computer 2004.

Project Milestone I Meetings

- Please come to office hours for feedback on
  - Your progress
  - Your presentation

Last Lectures

- Transactional Memory (brief)

- Interconnect wrap-up

- Project Milestone I presentations

Today

- More on Interconnects Research

- Start Dataflow

Research in Interconnects

Research Topics in Interconnects

- Plenty of topics in interconnection networks. Examples:

- Energy/power efficient and proportional design
- Reducing Complexity: Simplified router and protocol designs
- Adaptivity: Ability to adapt to different access patterns
- QoS and performance isolation
  - Reducing and controlling interference, admission control
- Co-design of NoCs with other shared resources
  - End-to-end performance, QoS, power/energy optimization
- Scalable topologies to many cores, heterogeneous systems
- Fault tolerance
- Request prioritization, priority inversion, coherence, …
- New technologies (optical, 3D)

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?

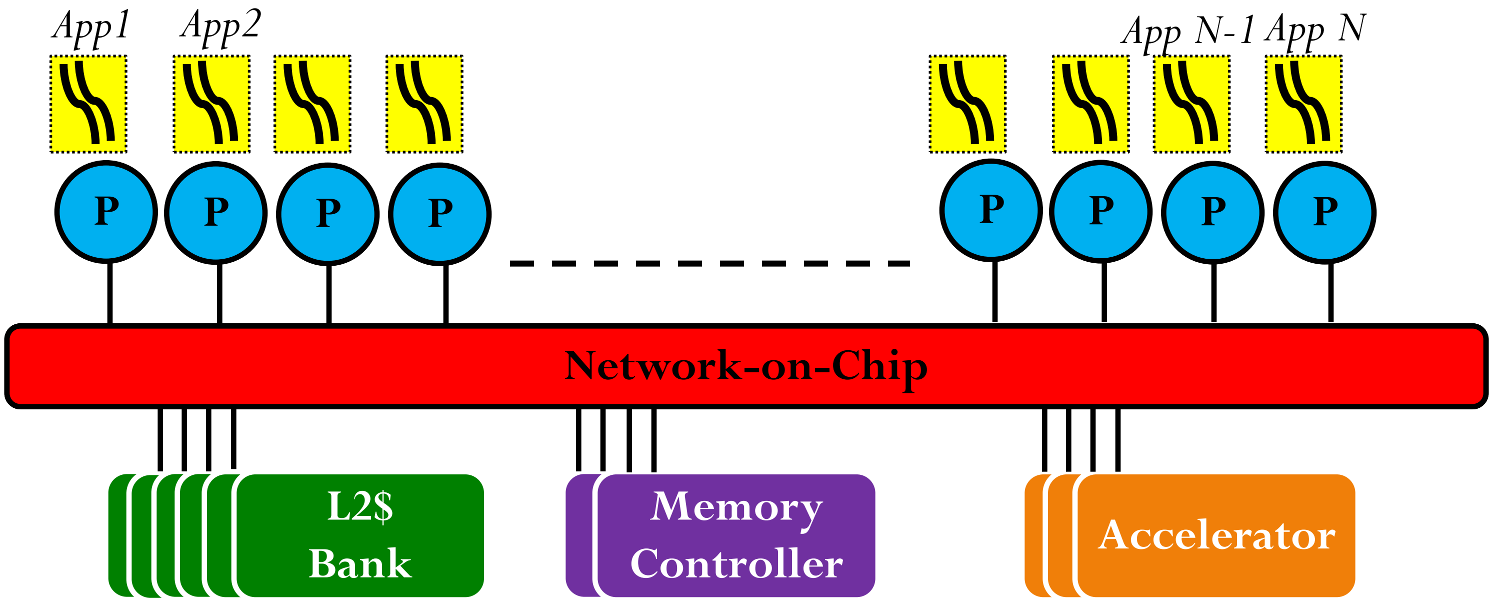
- Common strategies
  - Round robin across virtual channels
  - Oldest packet first (or an approximation)
  - Prioritize some virtual channels over others

- Better policies in a multi-core environment
  - Use application characteristics

Application-Aware Packet Scheduling

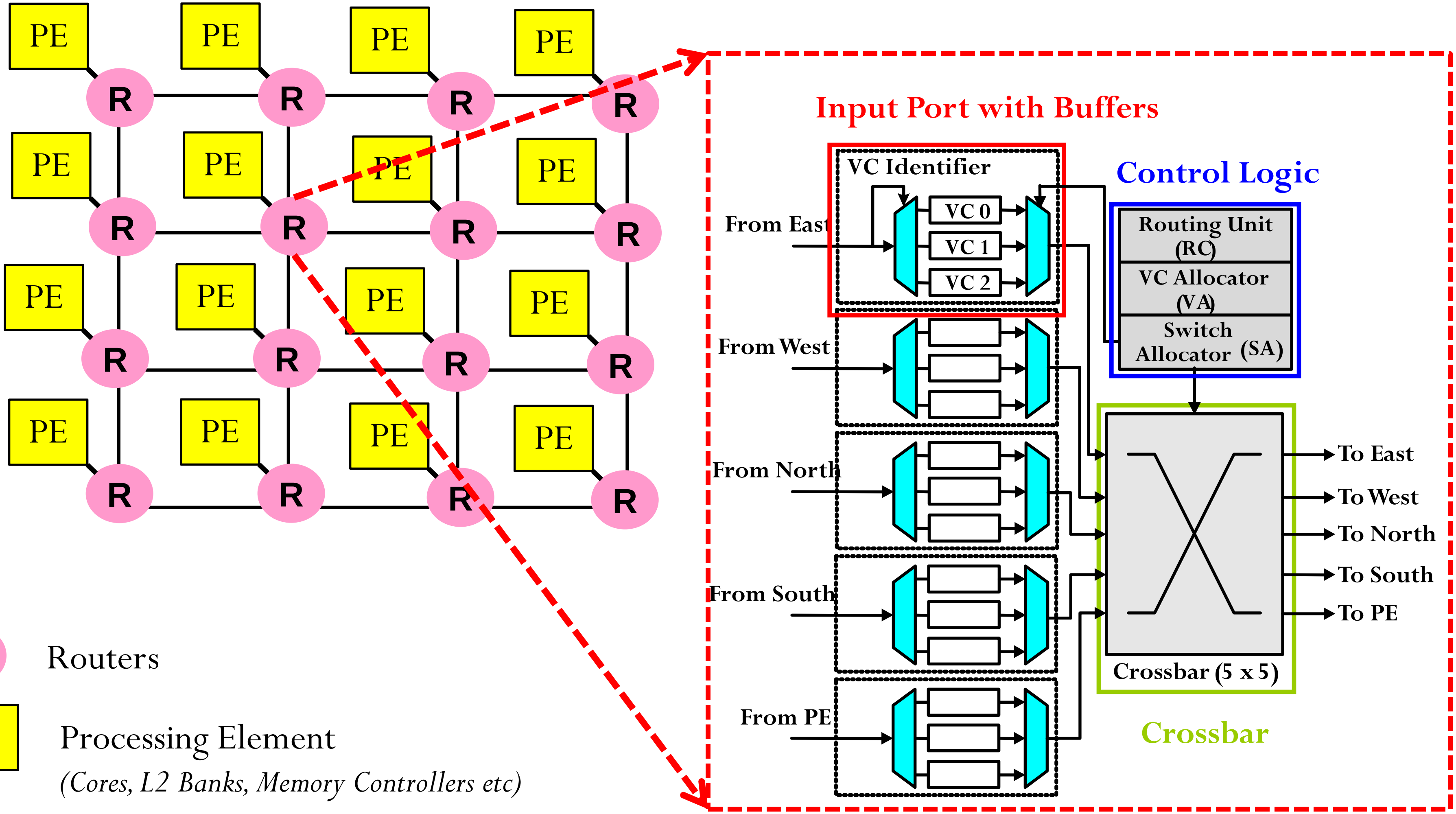
Das et al., "Application-Aware Prioritization Mechanisms for On-Chip Networks," MICRO 2009.

The Problem: Packet Scheduling



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

The Problem: Packet Scheduling

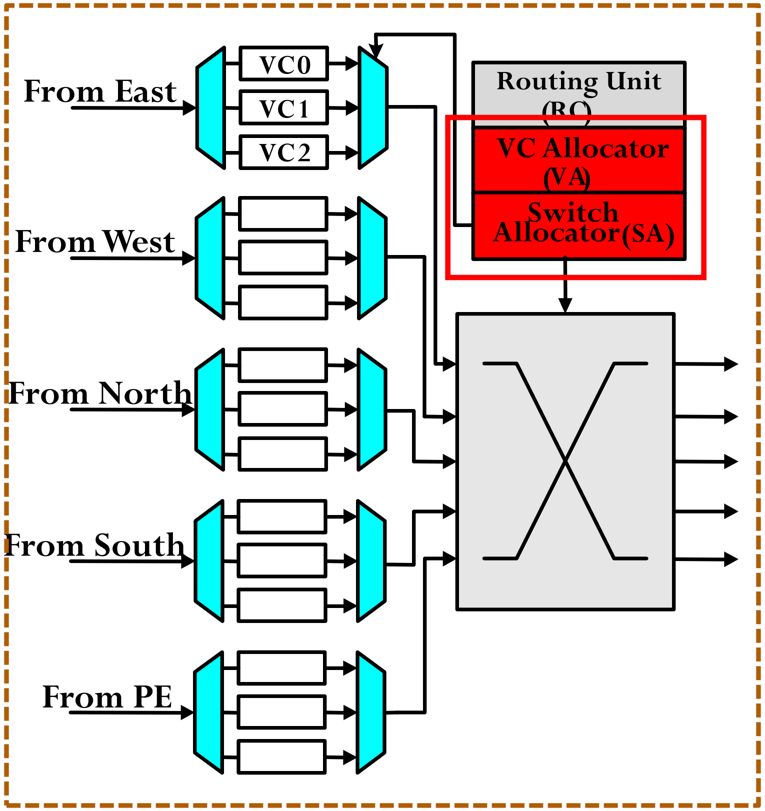


R Routers

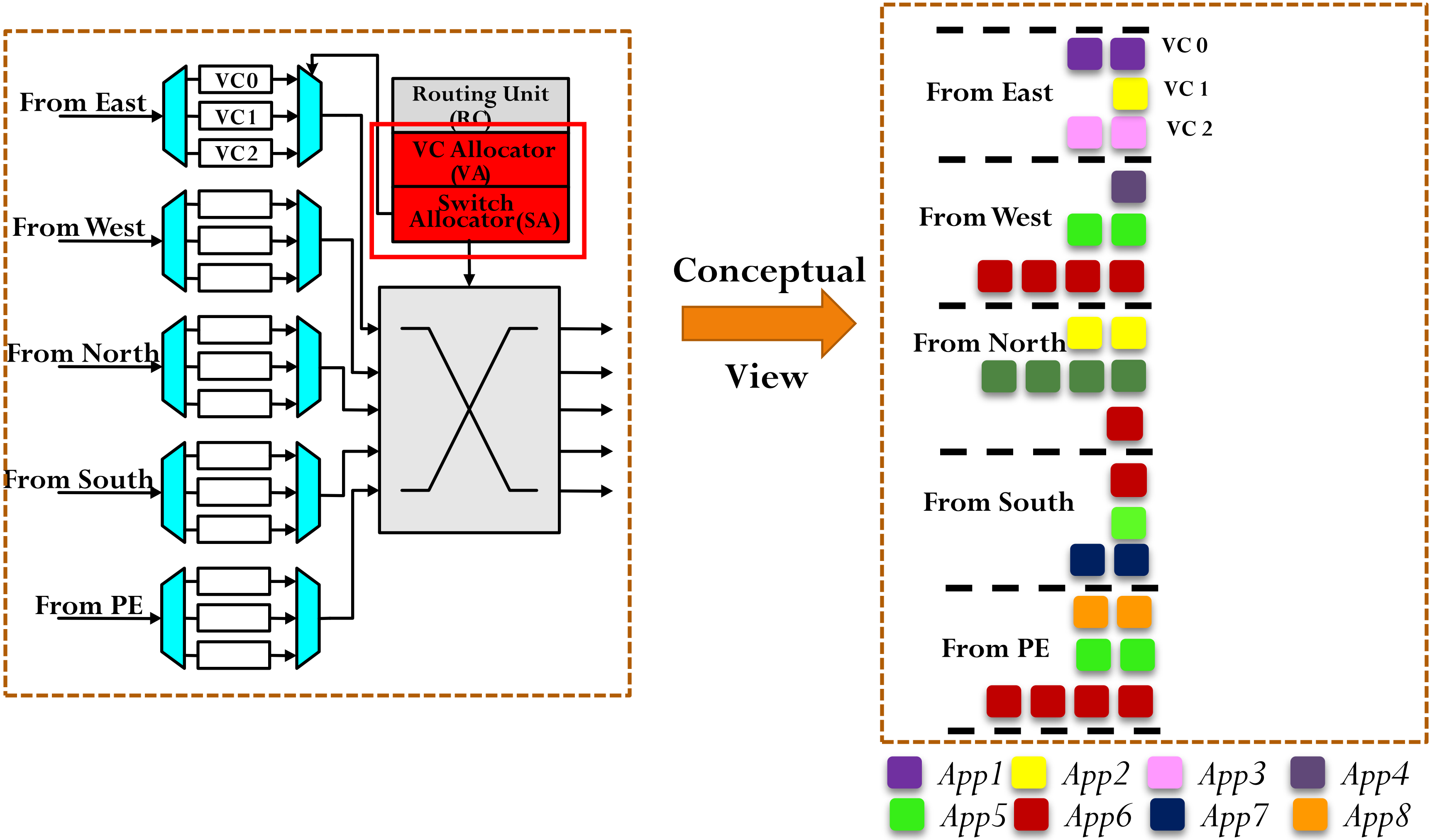
PE Processing Element

(Cores, L2 Banks, Memory Controllers etc)

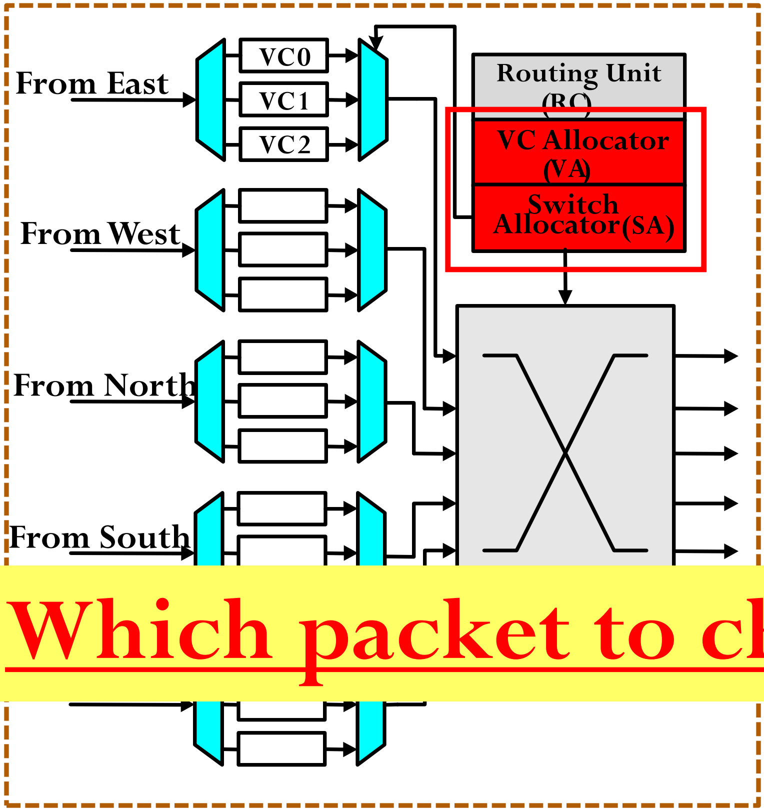
The Problem: Packet Scheduling



The Problem: Packet Scheduling

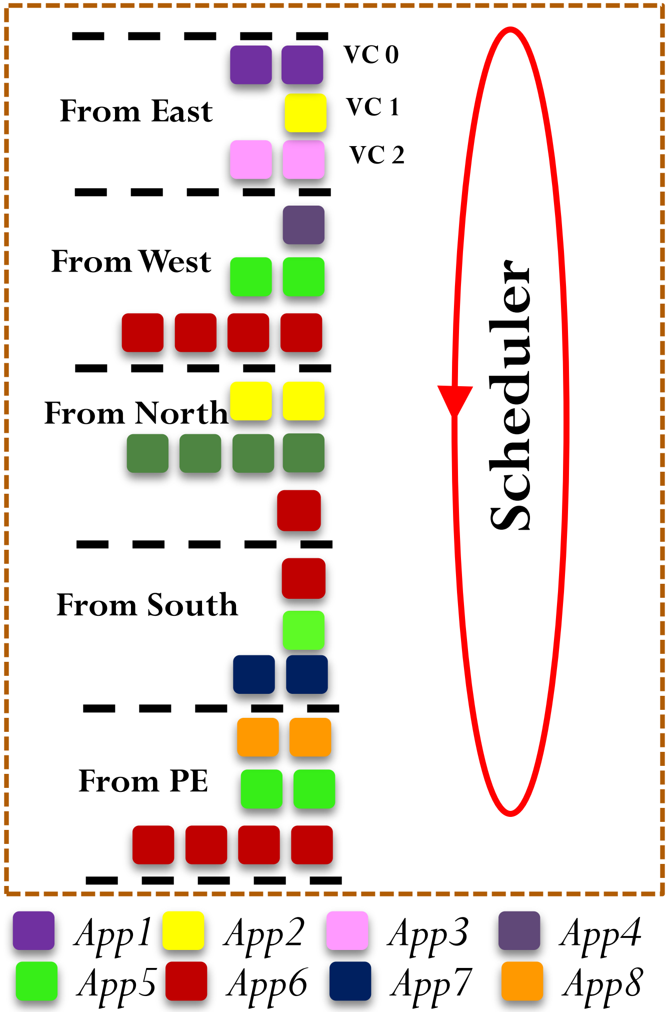


The Problem: Packet Scheduling



Which packet to choose?

Conceptual View



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

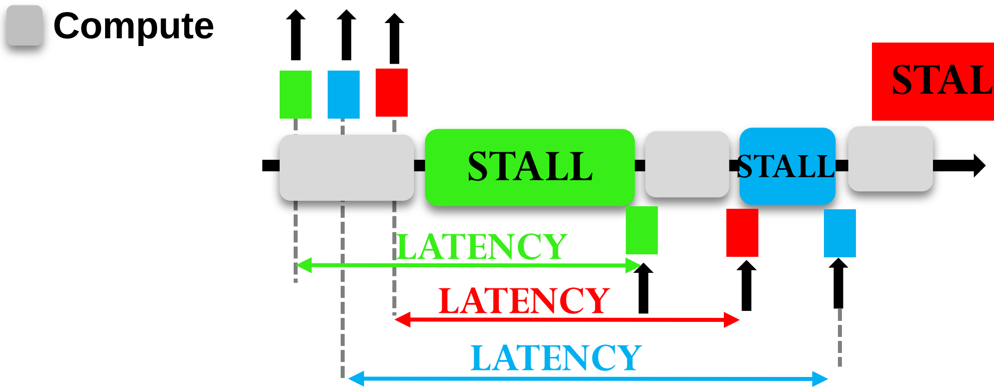
- Applications have different criticality with respect to the network
  - Some applications are network latency sensitive
  - Some applications are network latency tolerant

- 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 STC**
  - Shortest Job First Principle (SJF)
    - **Lower network load leads to higher STC**
  - Average Memory Access Time
    - **Higher memory access time leads to higher STC**

STC Principle 1 {MLP}

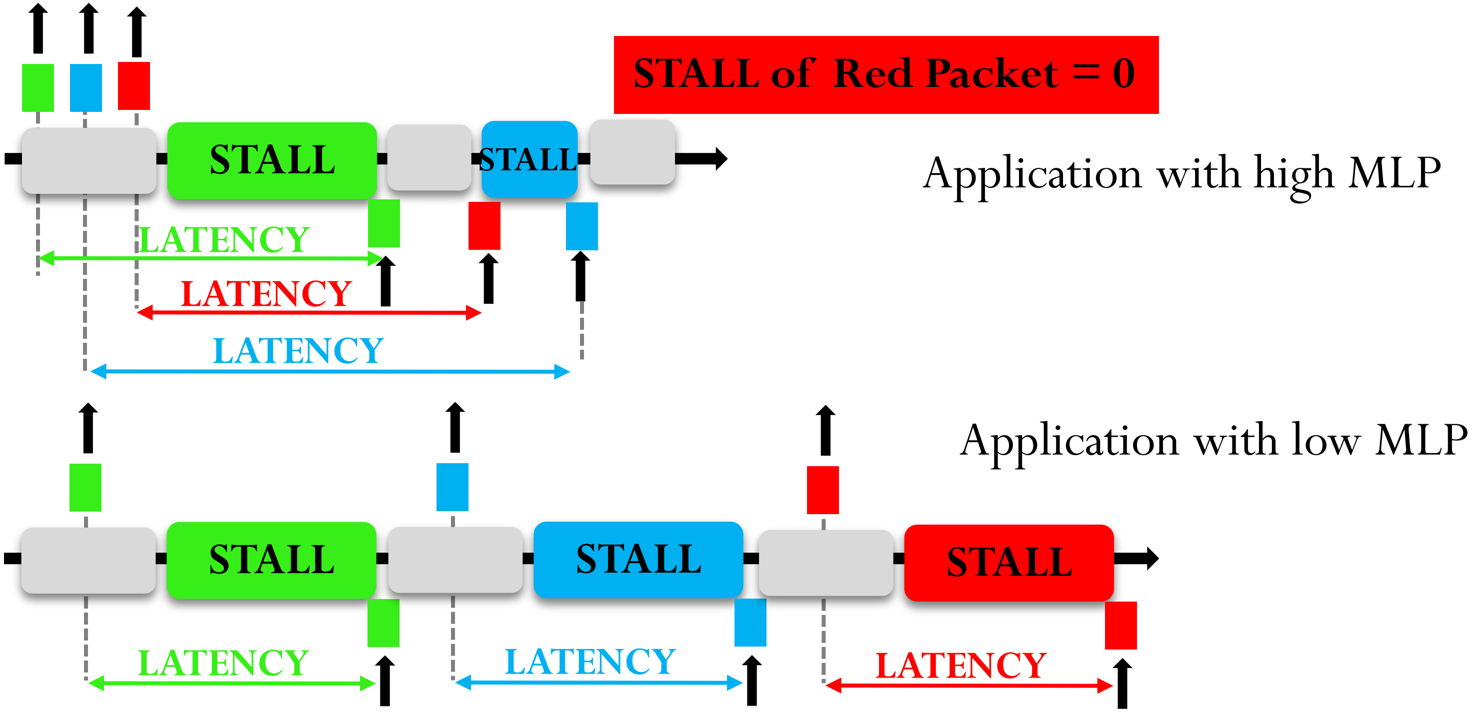


STALL of Red Packet = 0

Application with high MLP

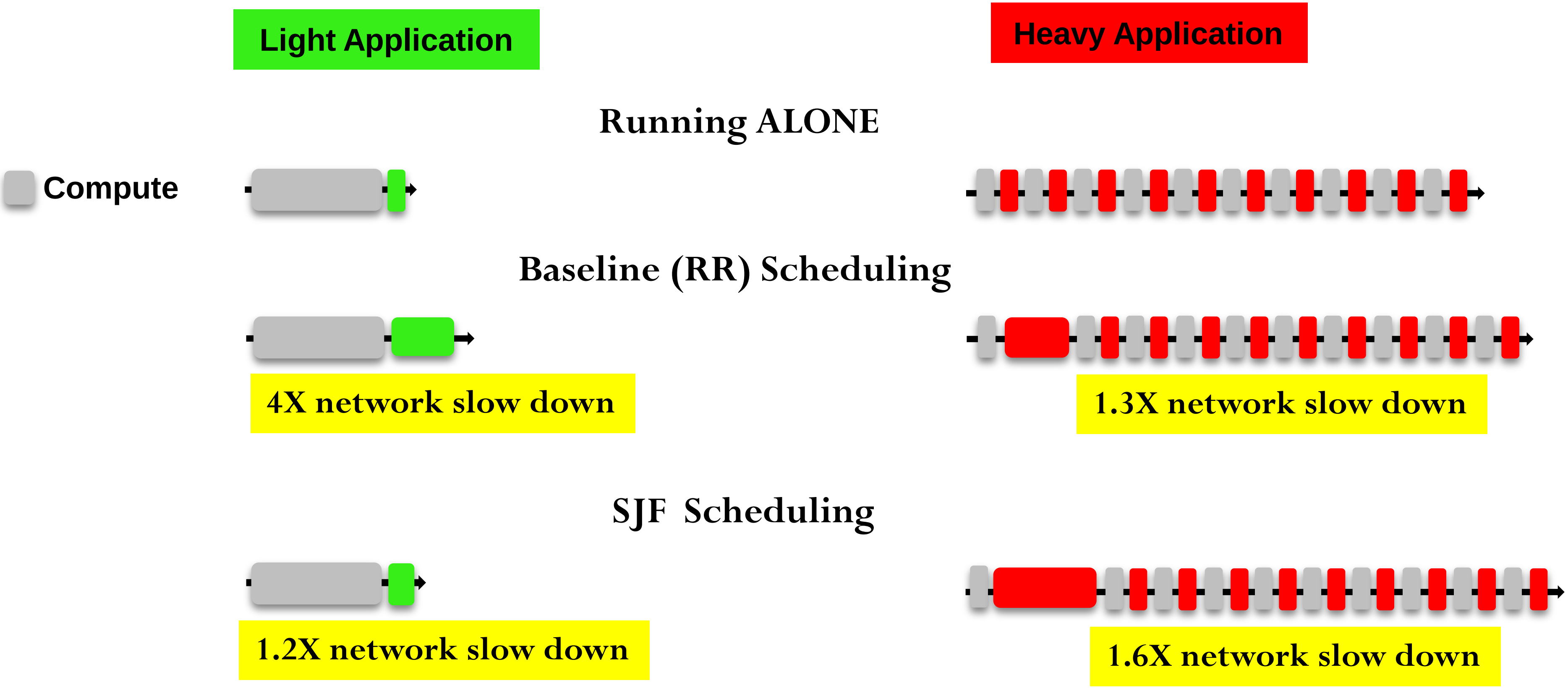
- Observation 1: **Packet Latency != Network Stall Time**

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

▨ **Compute**

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 stall time 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 Stall Time Criticality (STC).
- Explored many heuristics for quantifying STC (Details & analysis in paper)
  - 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 ranking 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

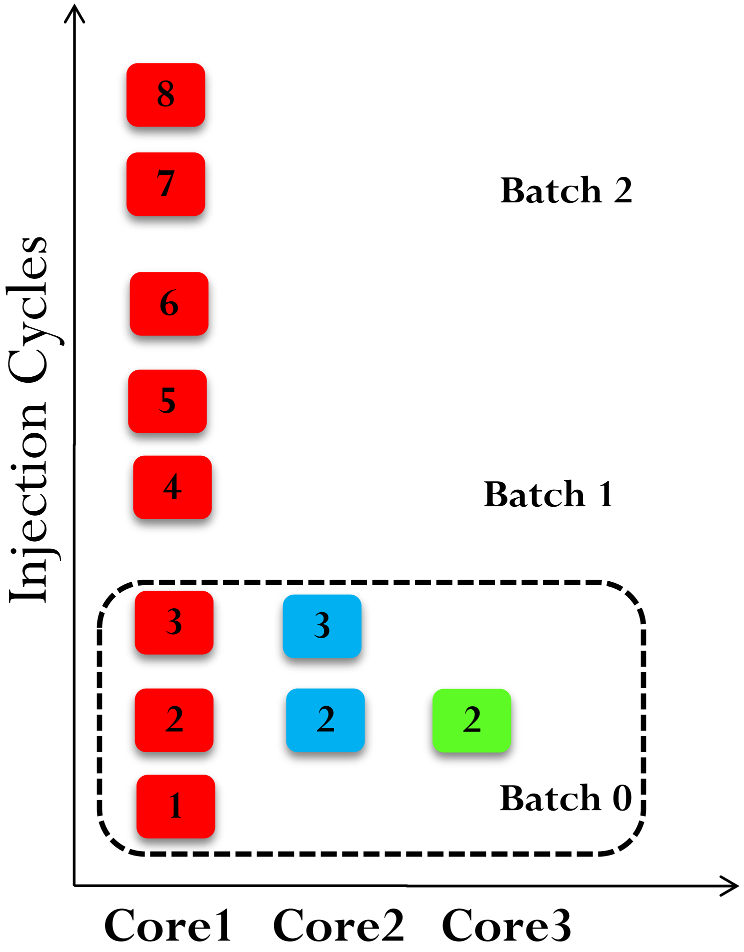
Component 2: Batching

- 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**
- Alternative batching policies explored in paper
- 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

- Before injecting a packet into the network, it is tagged by
  - 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 the same across all routers

STC Scheduling Example

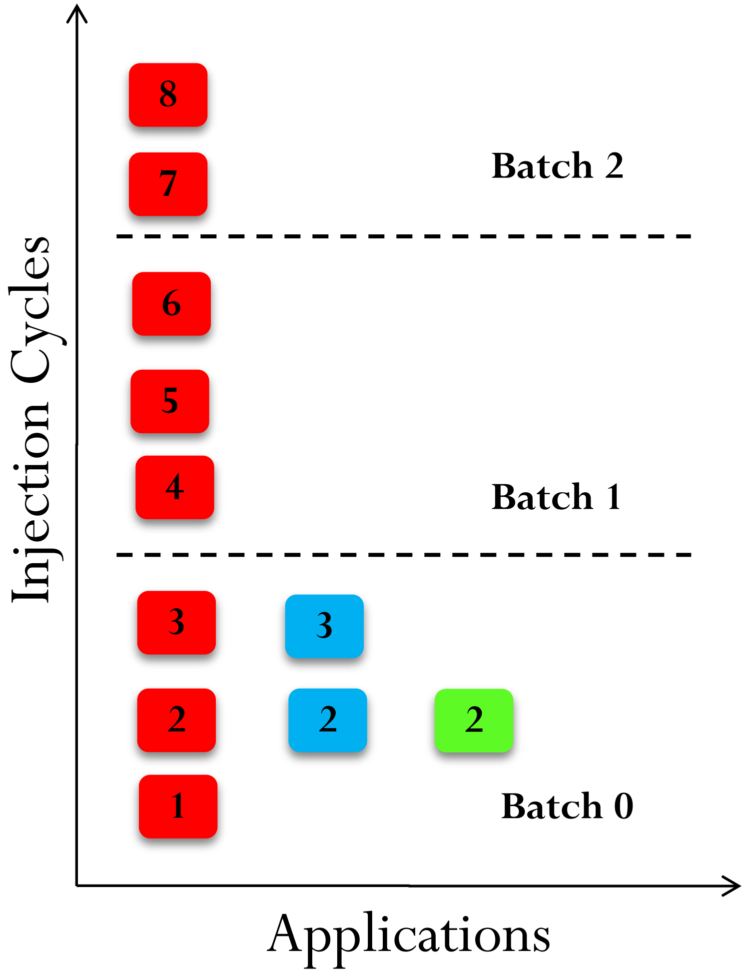


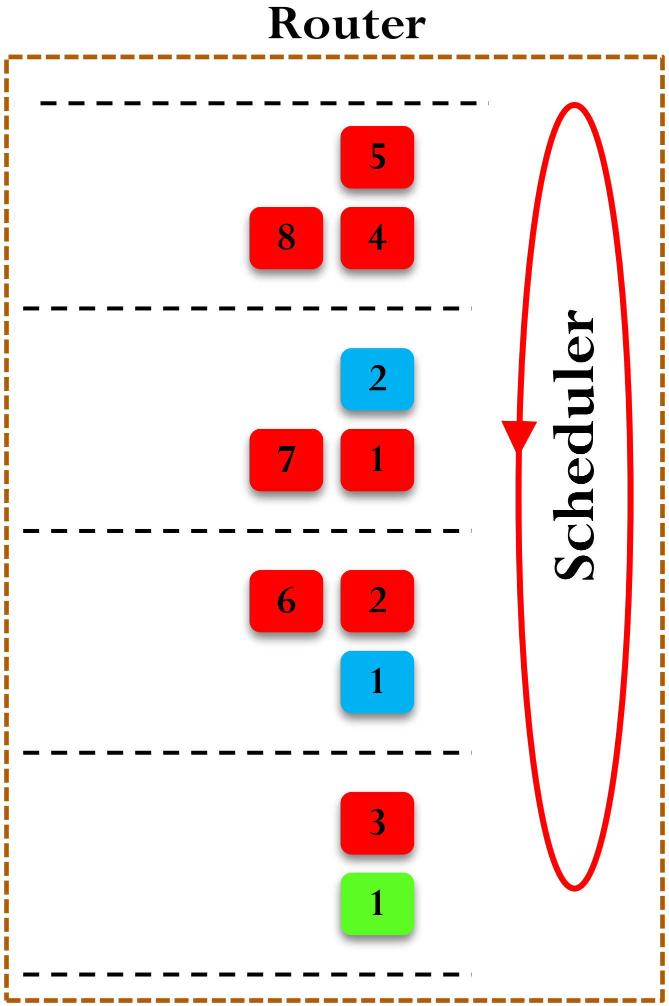
Packet Injection Order at Processor

Batching interval length = 3 cycles

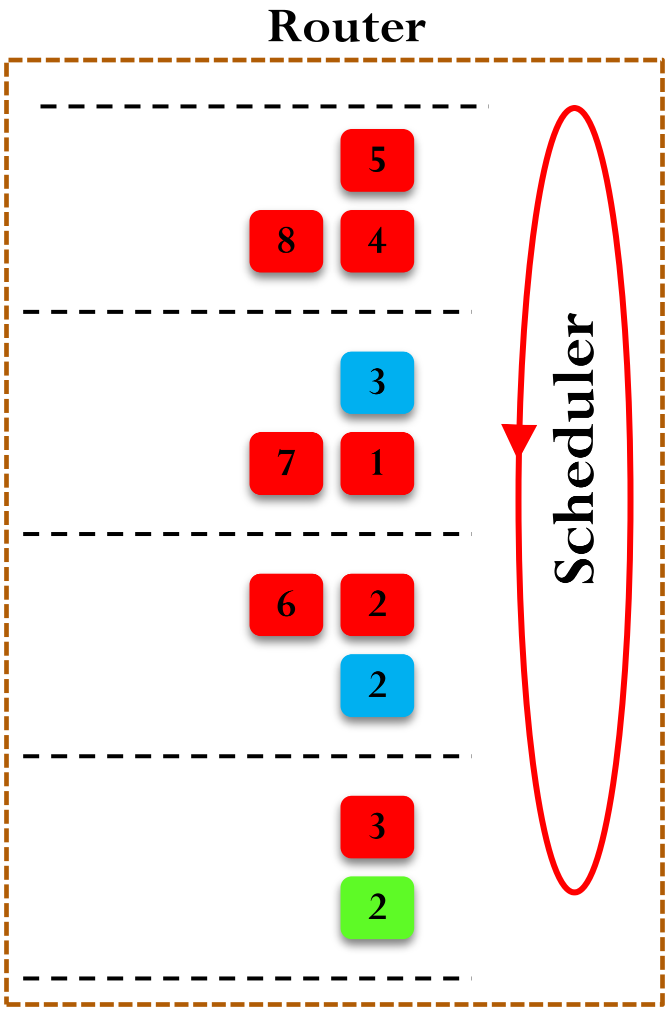
Ranking order = 🟩 > 🟦 > 🟥

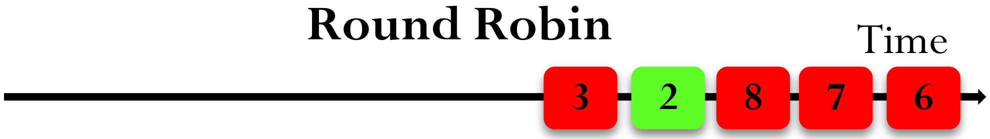
STC Scheduling Example





STC Scheduling Example

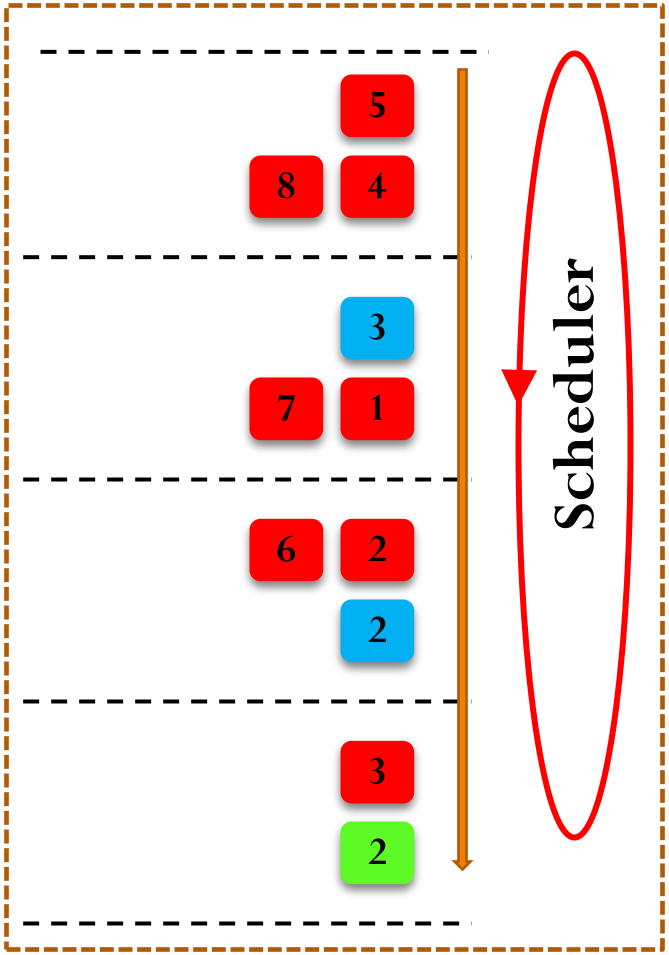


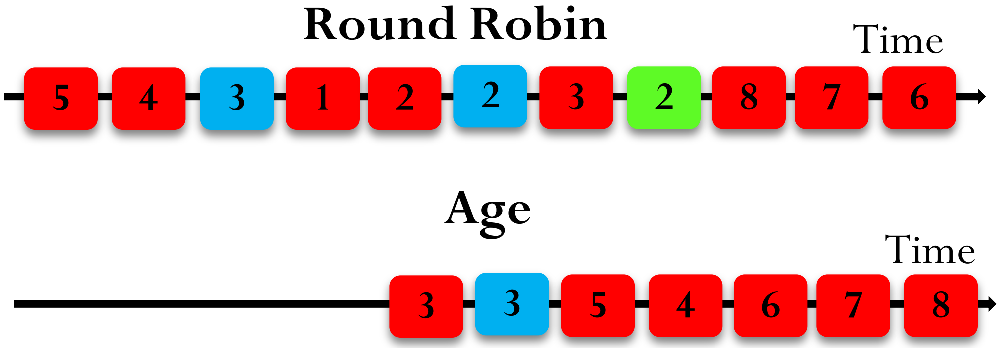


	STALL CYCLES			Avg
RR	8	6	11	8.3
Age				
STC				

STC Scheduling Example

Router

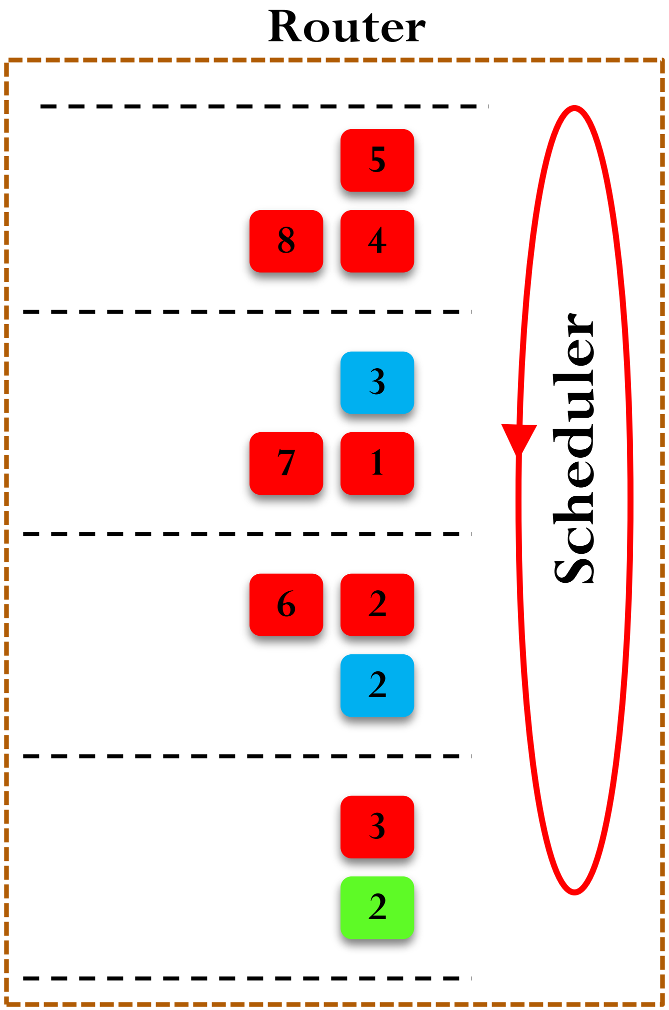


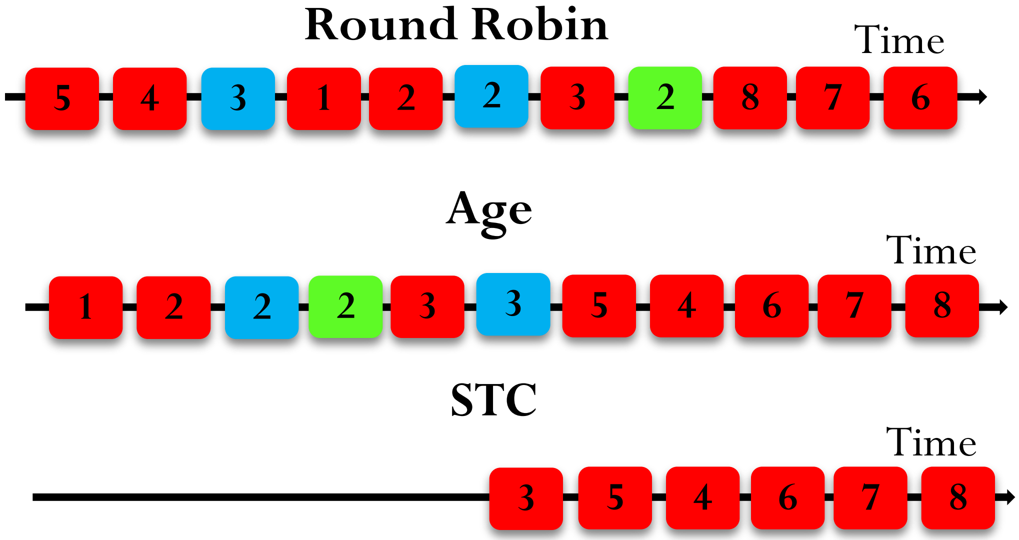


	STALL CYCLES			Avg
RR	8	6	11	8.3
Age	4	6	11	7.0
STC				

Ranking order 🟩 > 🟦 > 🟥

STC Scheduling Example





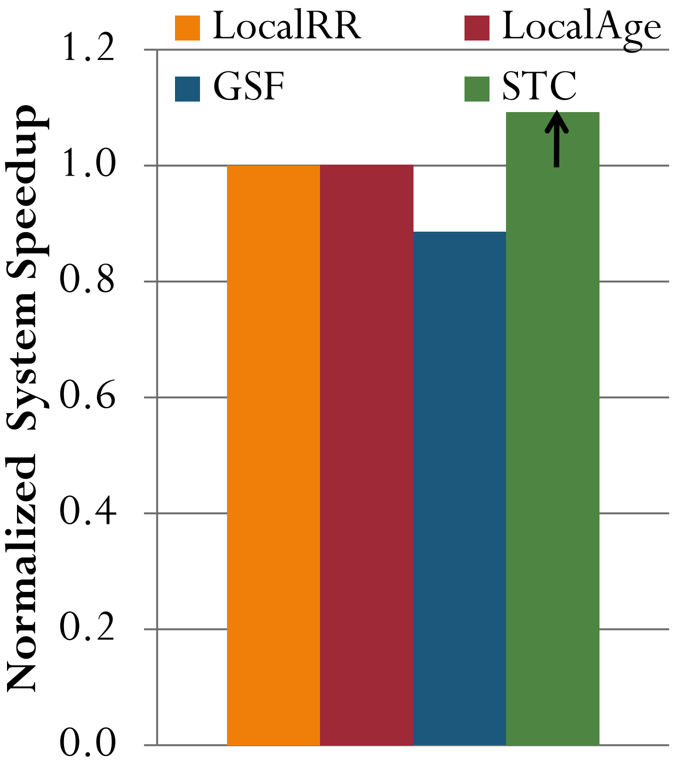
	STALL CYCLES			Avg
RR	8	6	11	8.3
Age	4	6	11	7.0
STC	1	3	11	5.0

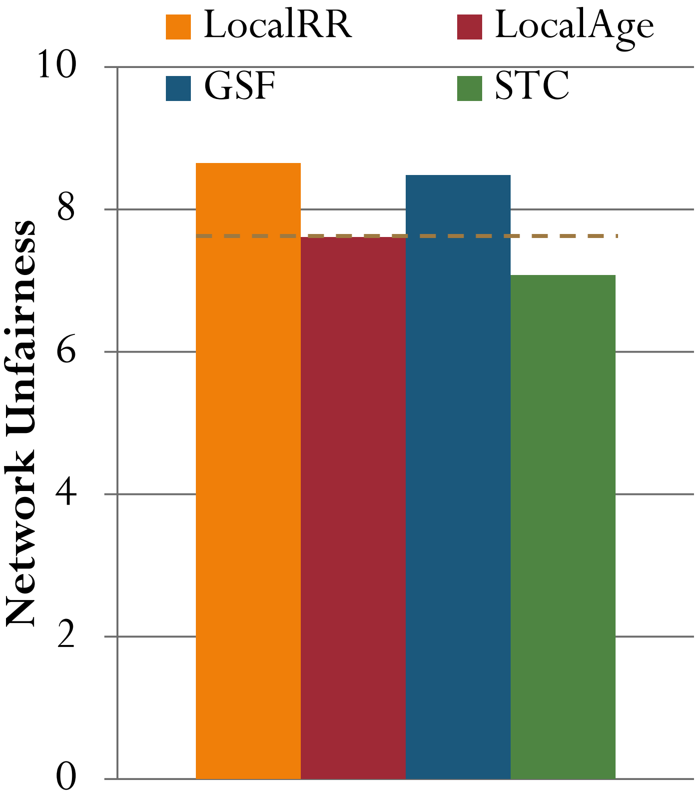
Qualitative Comparison

- **Round Robin & Age**
  - 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 & stall till oldest batch completes and frees up fresh credits.
    - Underutilization of network resources

System Performance

- STC provides 9.1% improvement in weighted speedup over the best existing policy{averaged across 96 workloads}
- Detailed case studies in the paper





Slack-Driven Packet Scheduling

Das et al., "Aergia: Exploiting Packet Latency Slack in On-Chip Networks," ISCA 2010.

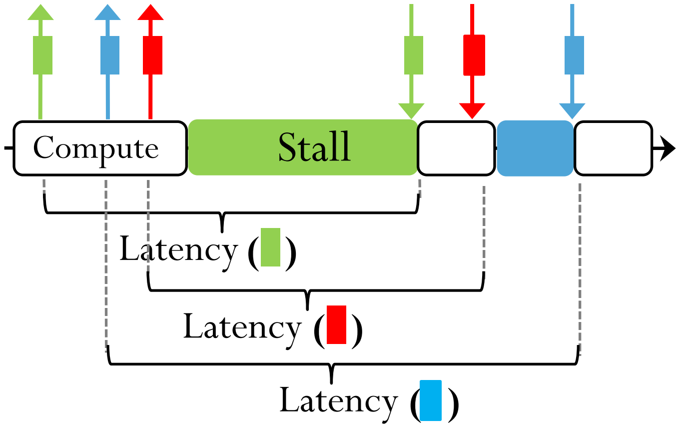
Packet Scheduling in NoC

- Existing scheduling policies
  - Round robin
  - Age
- Problem
  - Treat all packets equally
  - Application-oblivious

All packets are not the same…!!!

- Packets have different criticality
  - Packet is critical if latency of a packet affects application's performance
  - Different criticality due to memory level parallelism (MLP)

MLP Principle



Stall (🟥) = 0

Packet Latency != 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
- Aérgia
  - Concept of Slack
  - Estimating Slack
- Evaluation
- Conclusion

Slack of Packets

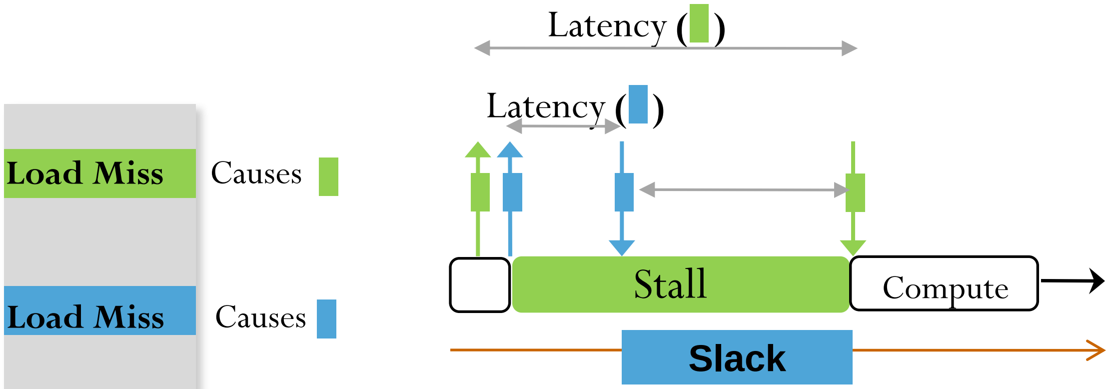
- What is slack of a packet?
  - Slack of a packet is number of cycles it can be delayed in a router without (significantly) reducing application's performance
  - Local network slack
- Source of slack: Memory-Level Parallelism (MLP)
  - Latency of an application's packet hidden from application due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

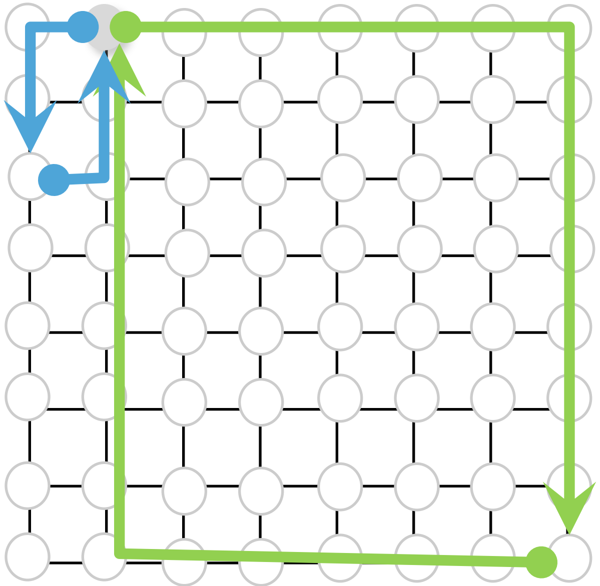
Concept of Slack

Instruction Window

Execution Time

Network-on-Chip





🟦 returns earlier than necessary

Slack (🟦) = Latency (🟩) – Latency (🟦) = 26 – 6 = 20 hops

Packet(🟦) can be delayed for available slack cycles without reducing performance!

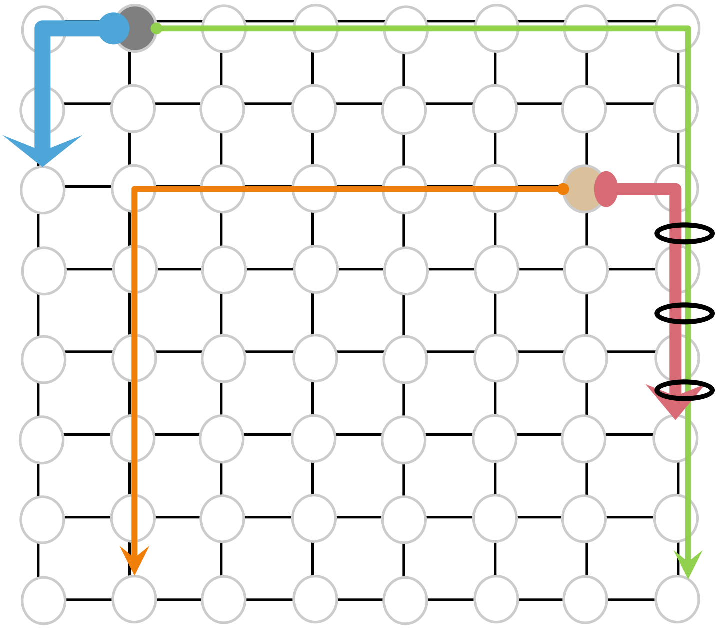
Prioritizing using Slack

Core A

Load Miss	**Causes** 🟩
Load Miss	**Causes** 🟦

Core B

Load Miss	**Causes** 🟧
Load Miss	**Causes** 🟥



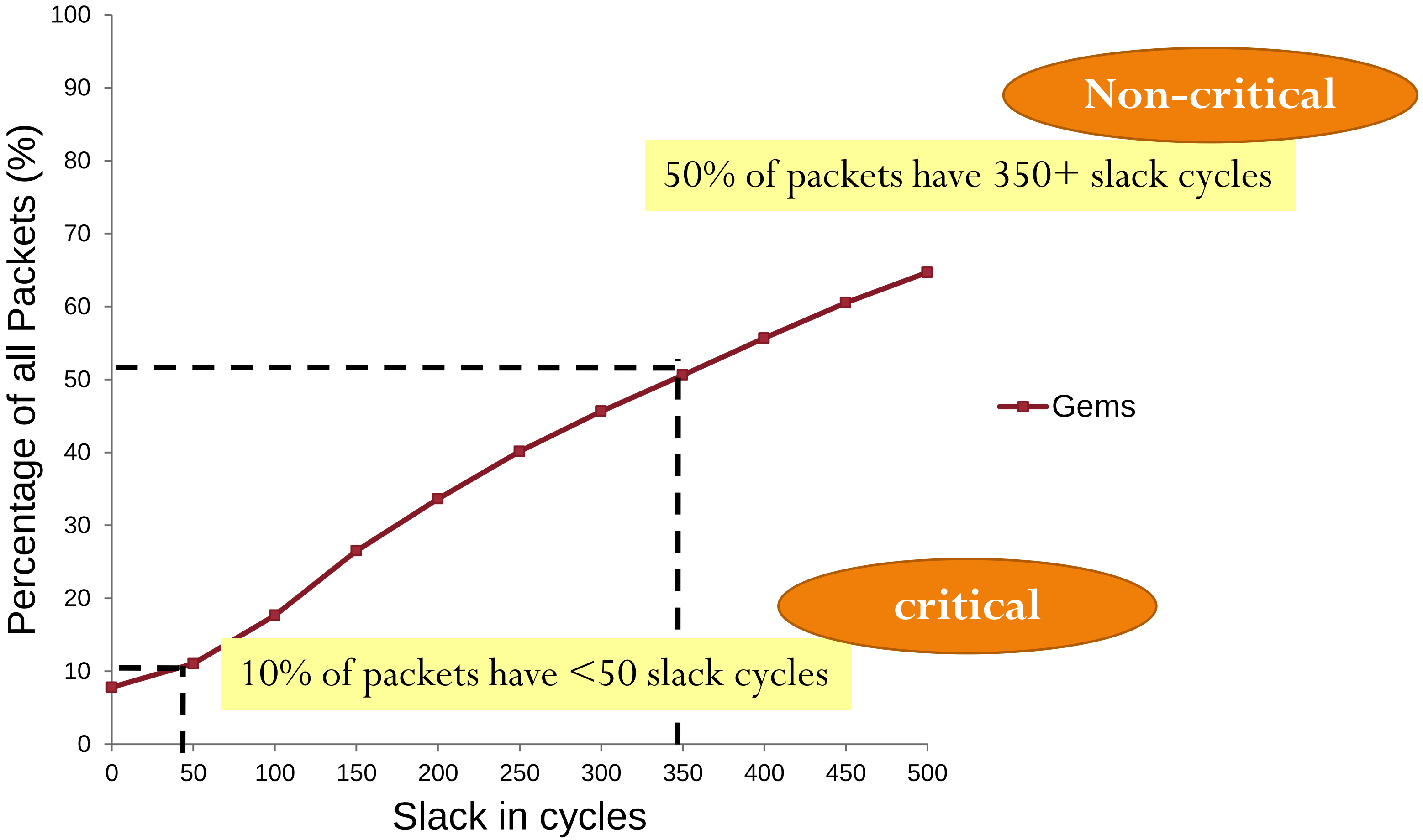
Packet	**Latency**	**Slack**
🟩	13 hops	0 hops
🟦	3 hops	10 hops

⬭ Interference at 3 hops

Slack(🟥) > Slack (🟩)

Prioritize 🟩

Slack in Applications



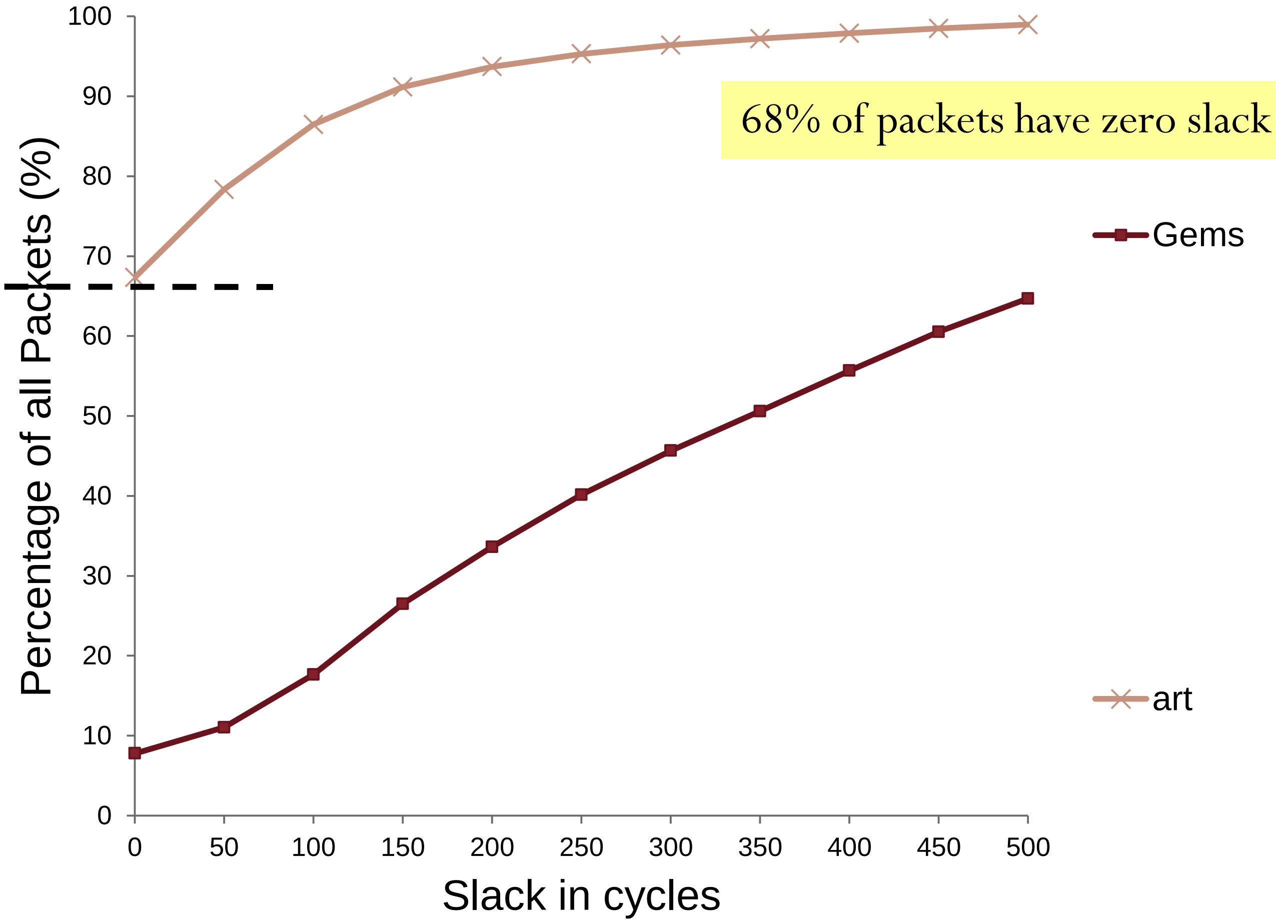
Non-critical

50% of packets have 350+ slack cycles

critical

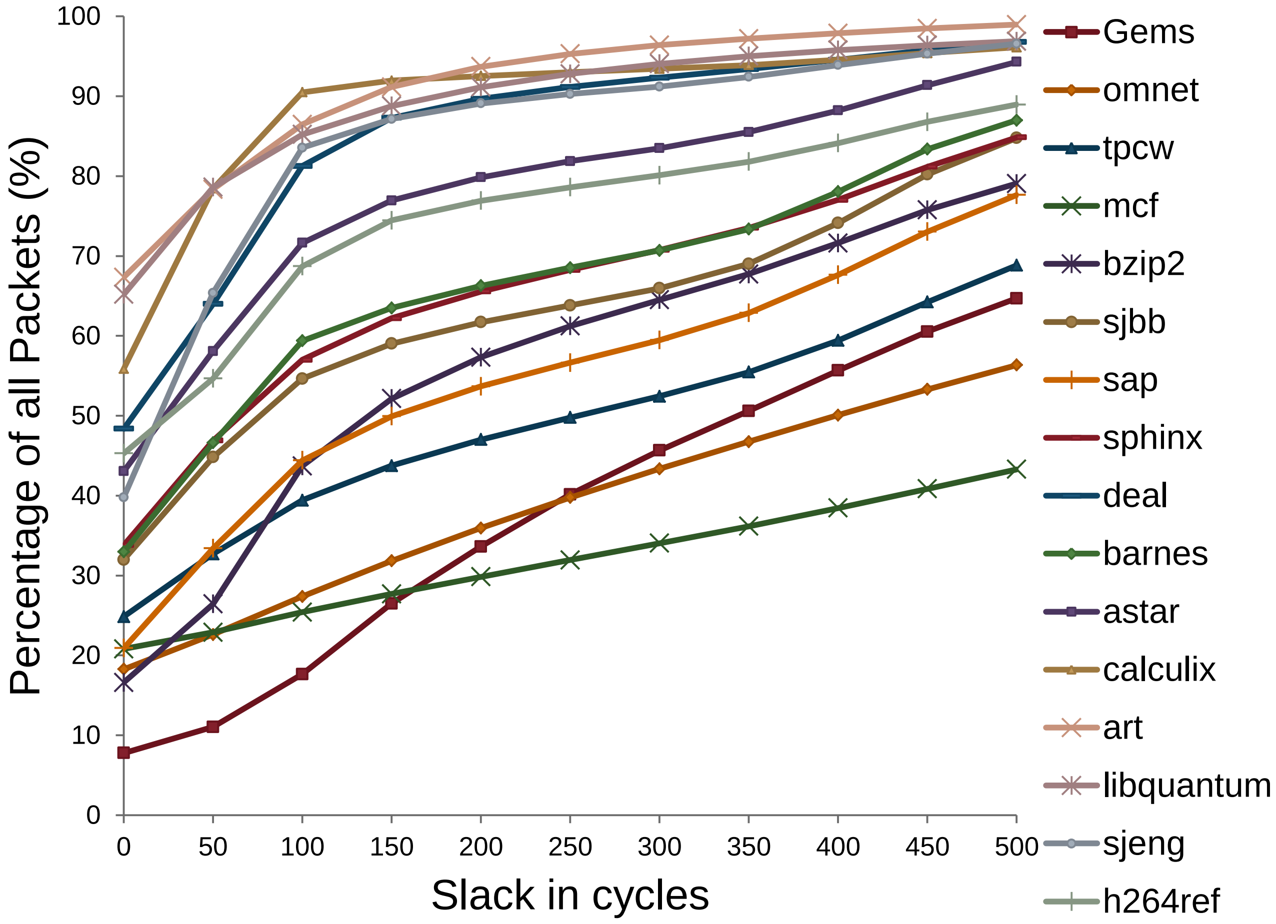
10% of packets have <50 slack cycles

Slack in Applications

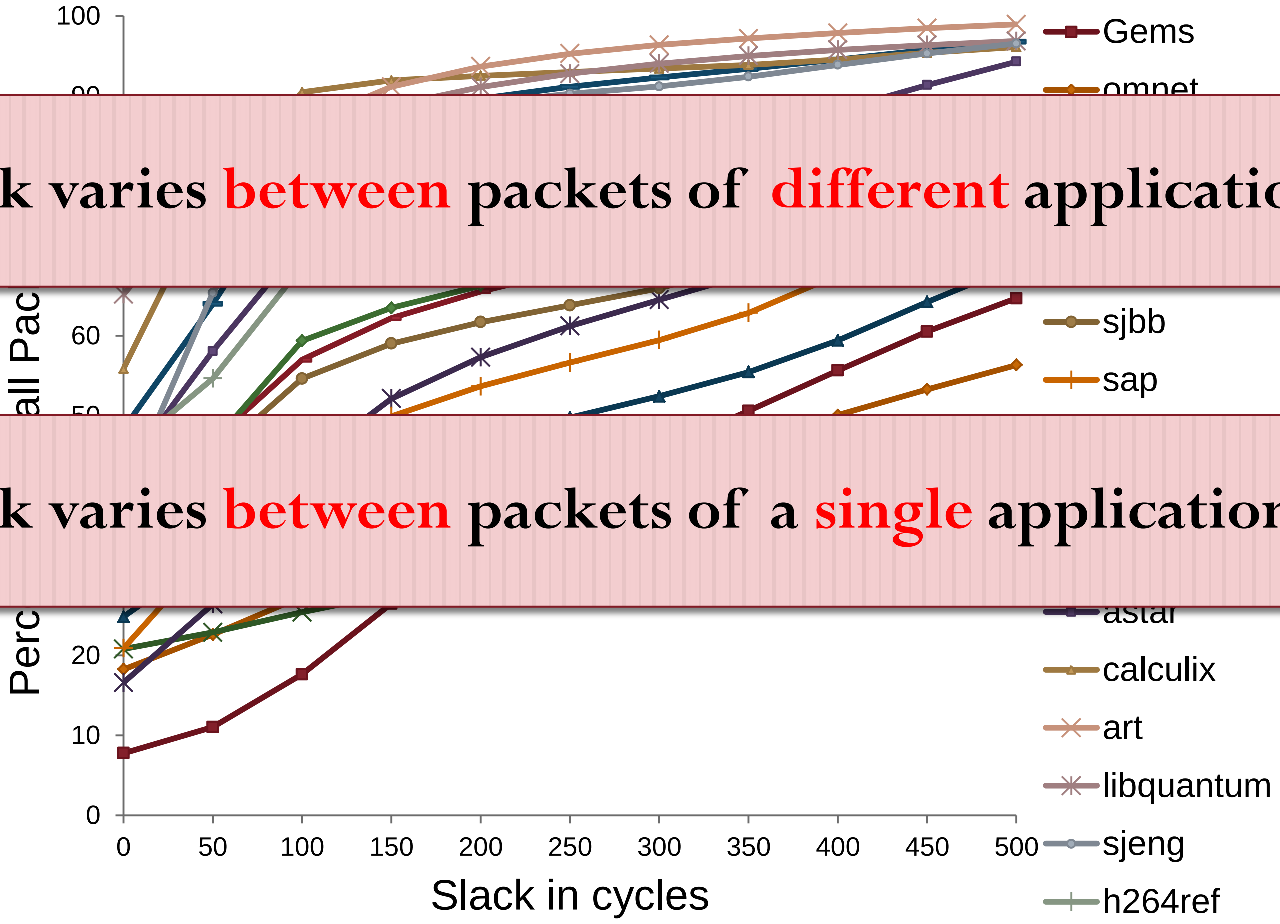


68% of packets have zero slack cycles

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
- Aérgia
  - 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 >= “T” threshold then next load is a L2 miss.
- Number of miss predecessors?
    - List of outstanding L2 Misses
- Hops estimate?
    - Hops => $\Delta X + \Delta Y$ 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
- Aérgia
  - Concept of Slack
  - Estimating Slack
- Evaluation
- Conclusion

Evaluation Methodology

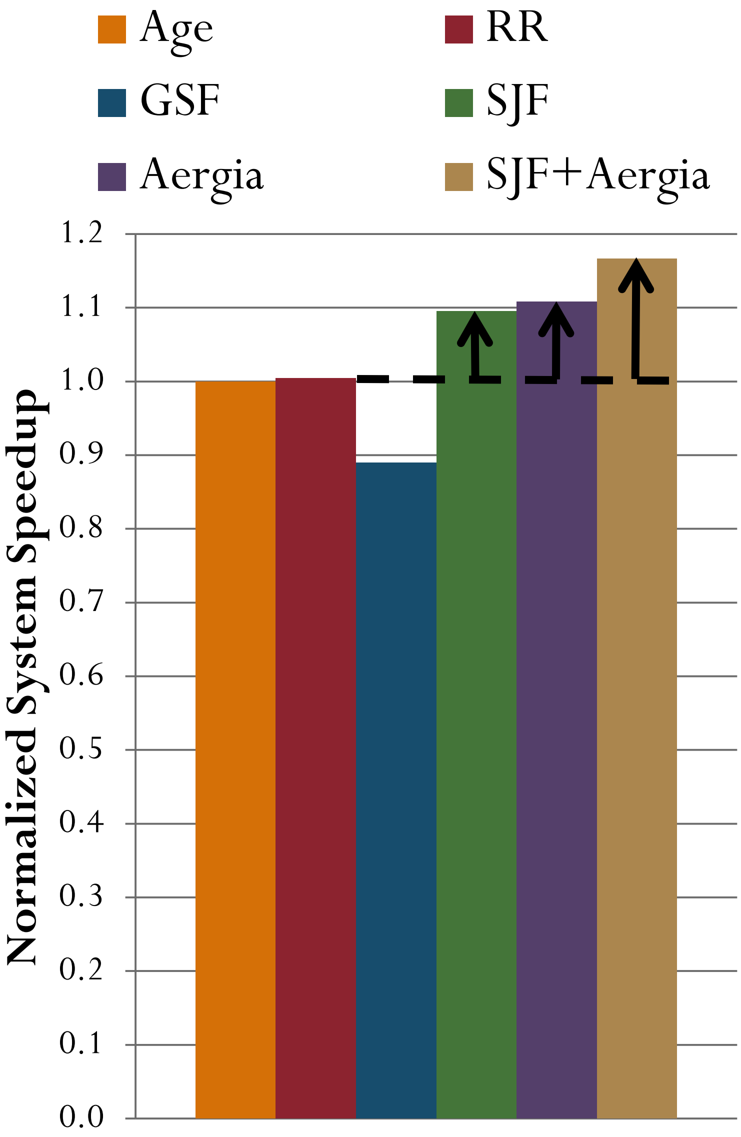
- 64-core system
  - x86 processor model based on Intel Pentium M
  - 2 GHz processor, 128-entry instruction window
  - 32KB private L1 and 1MB per core shared L2 caches, 32 miss buffers
  - 4GB DRAM, 320 cycle access latency, 4 on-chip DRAM controllers
- Detailed Network-on-Chip model
  - 2-stage routers (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

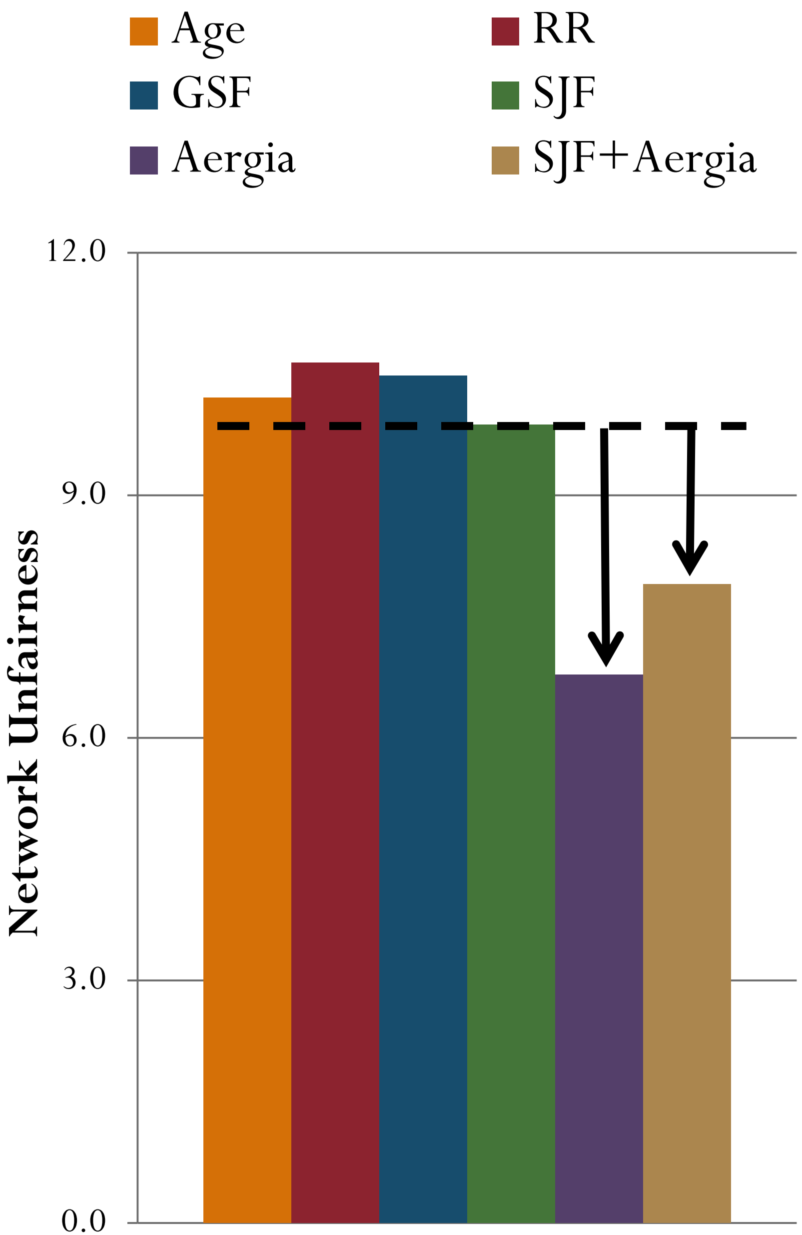
System Performance

- 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



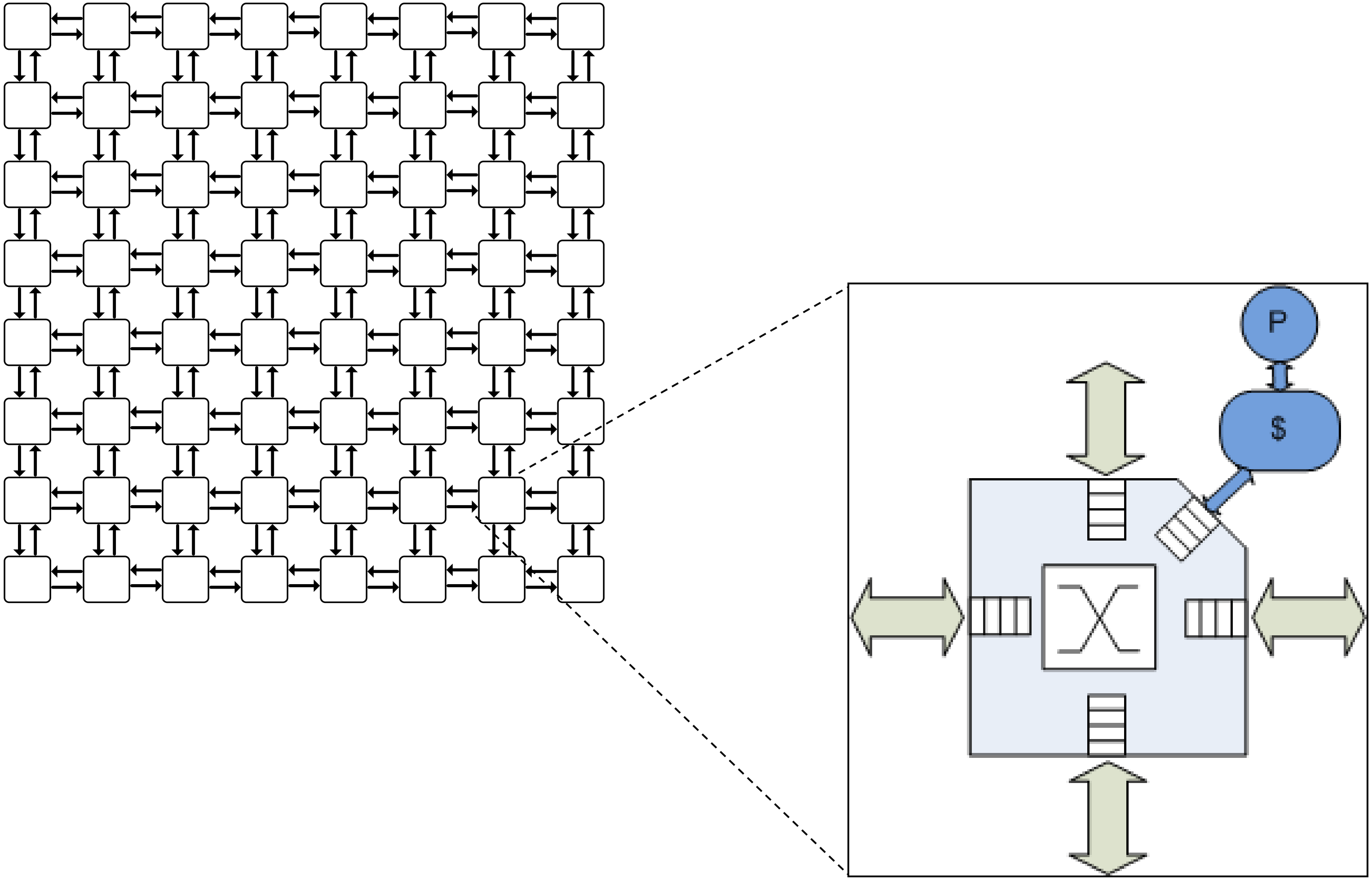
Conclusions & Future Directions

- 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%

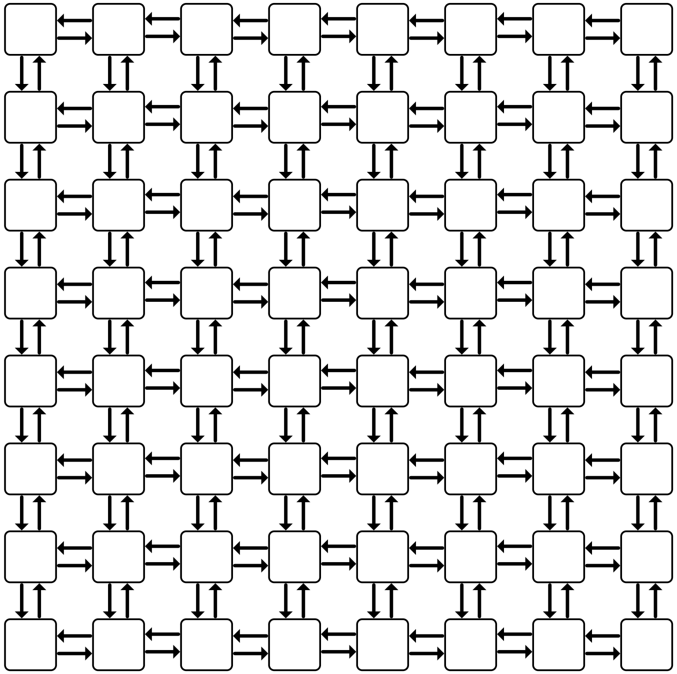
Express-Cube Topologies

Grot et al., “Express Cube Topologies for On-Chip Interconnects” ” HPCA 2009.

2-D Mesh

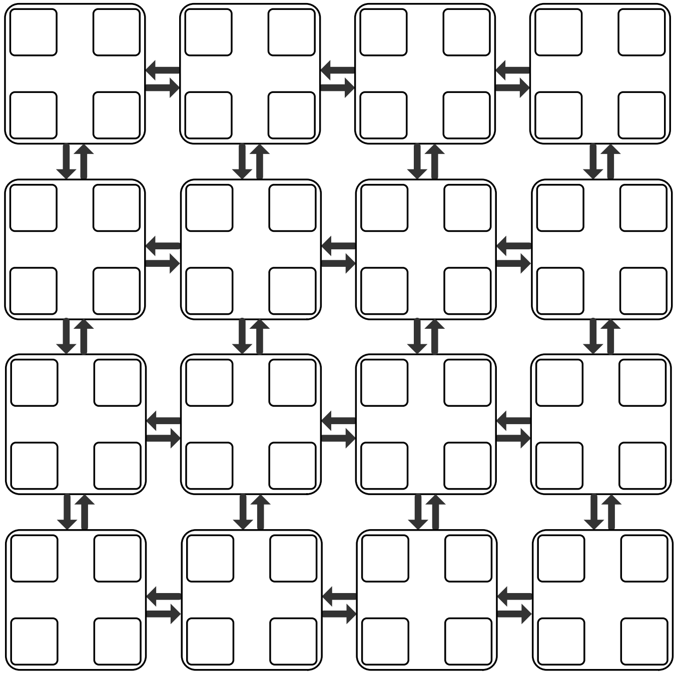


2-D Mesh



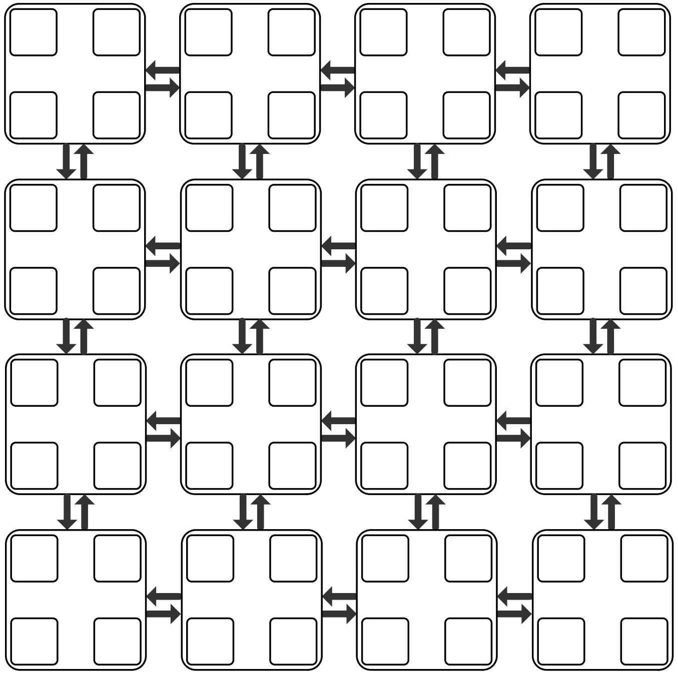
- Pros
  - Low design & layout complexity
  - Simple, fast routers
- Cons
  - Large diameter
  - Energy & latency impact

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



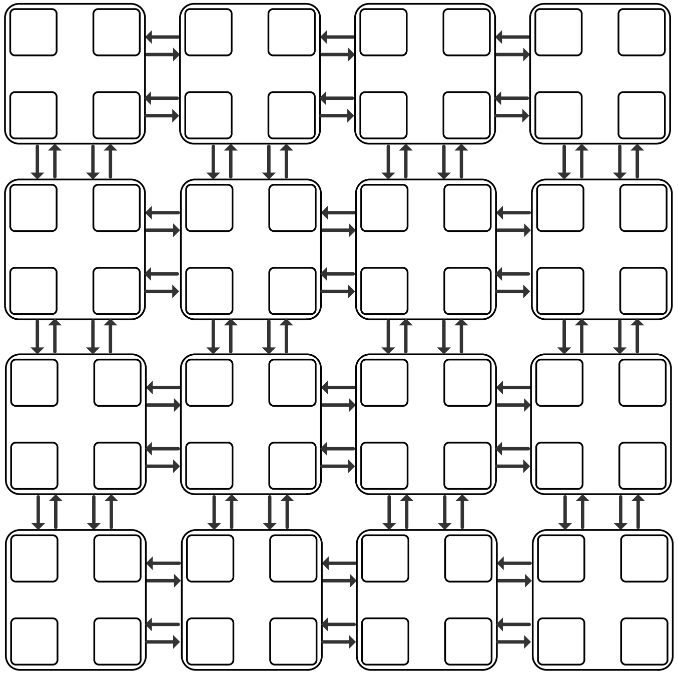
- Pros
  - Multiple *terminals* attached to a router node
  - Fast nearest-neighbor communication via the crossbar
  - Hop count reduction proportional to *concentration* degree
- Cons
  - Benefits limited by crossbar complexity

Concentration



- Side-effects
  - Fewer channels
  - Greater channel width

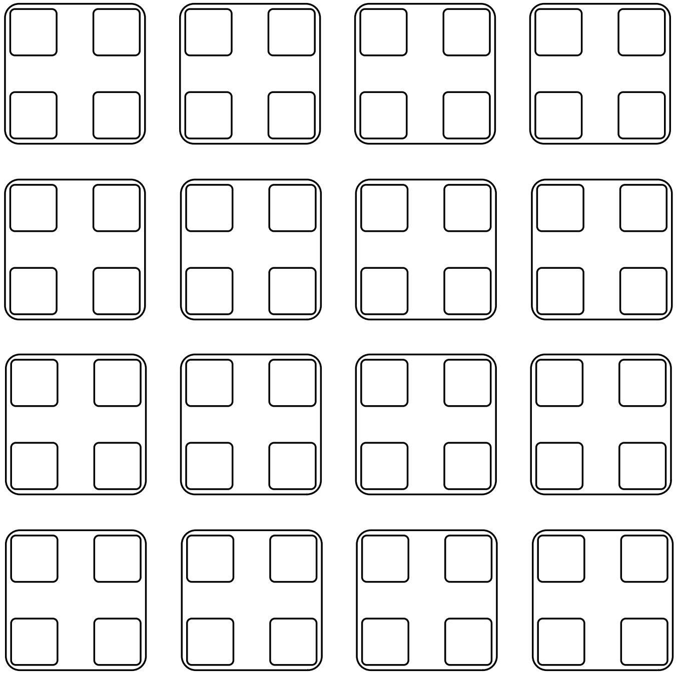
Replication



CMesh-X2

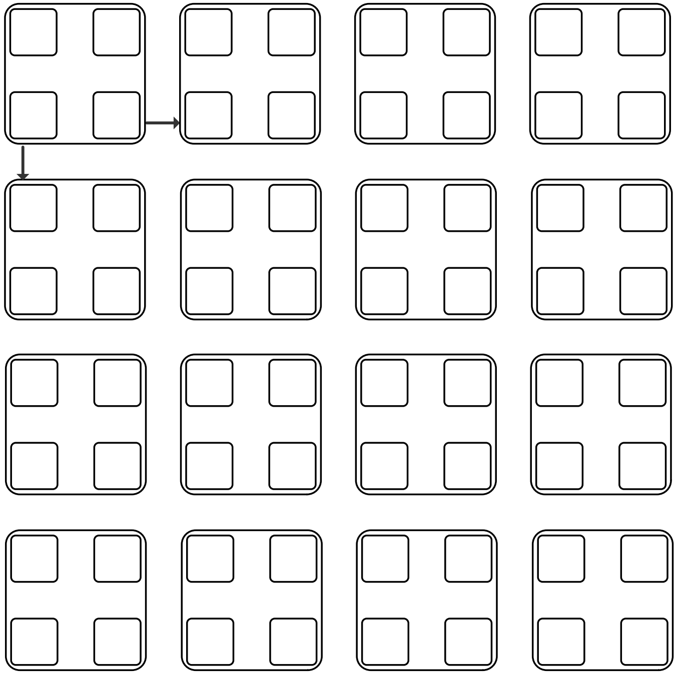
- Benefits
  - Restores bisection channel count
  - Restores channel width
  - Reduced crossbar complexity

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

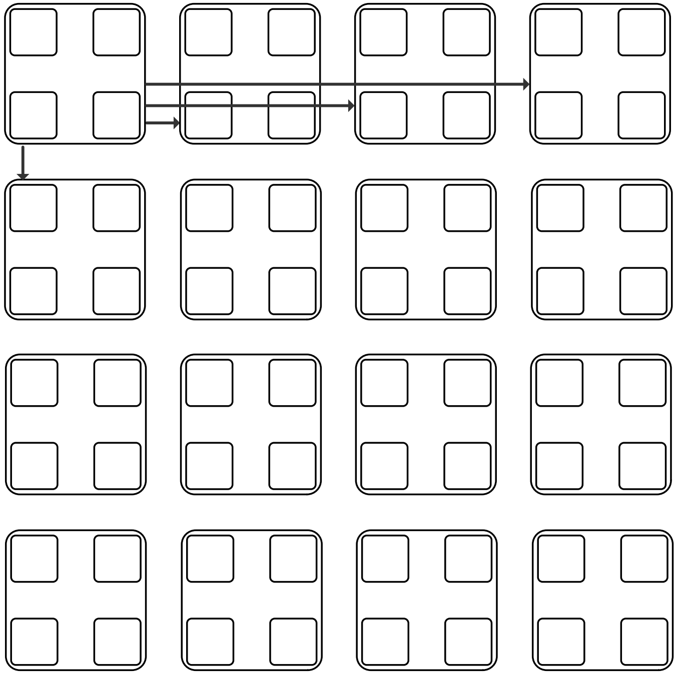


- Objectives:
  - Improve connectivity
  - Exploit the wire budget

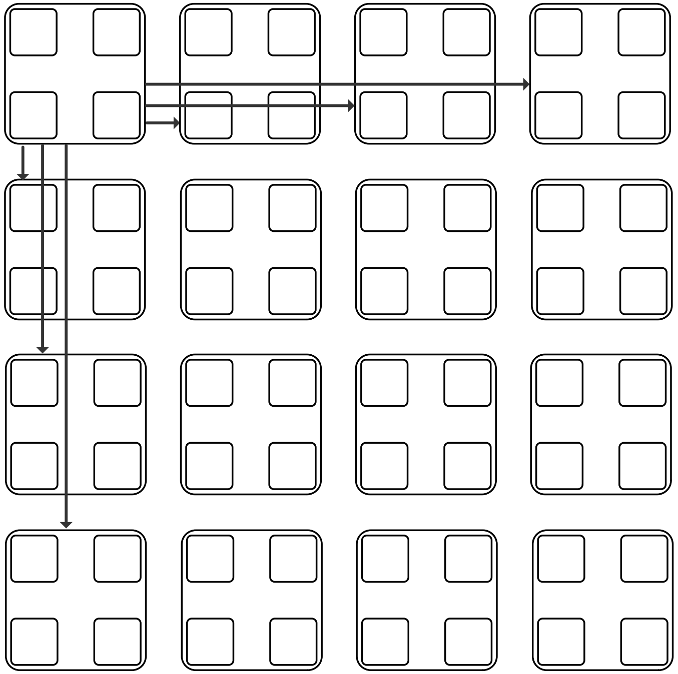
Flattened Butterfly *(Kim et al., Micro '07)*



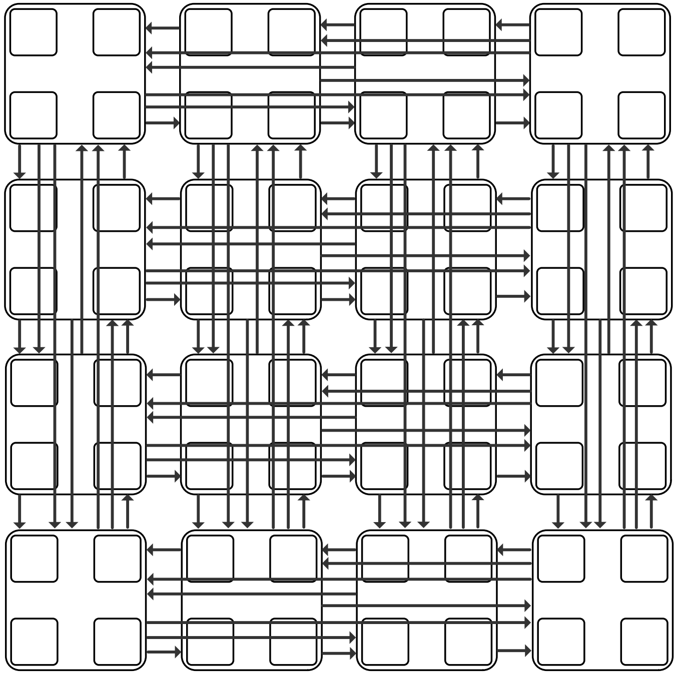
Flattened Butterfly *(Kim et al., Micro '07)*



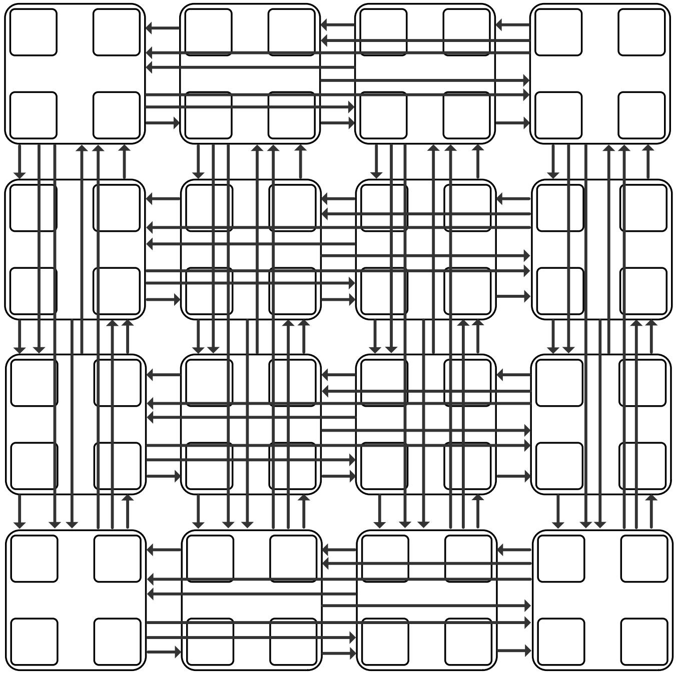
Flattened Butterfly *(Kim et al., Micro '07)*



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

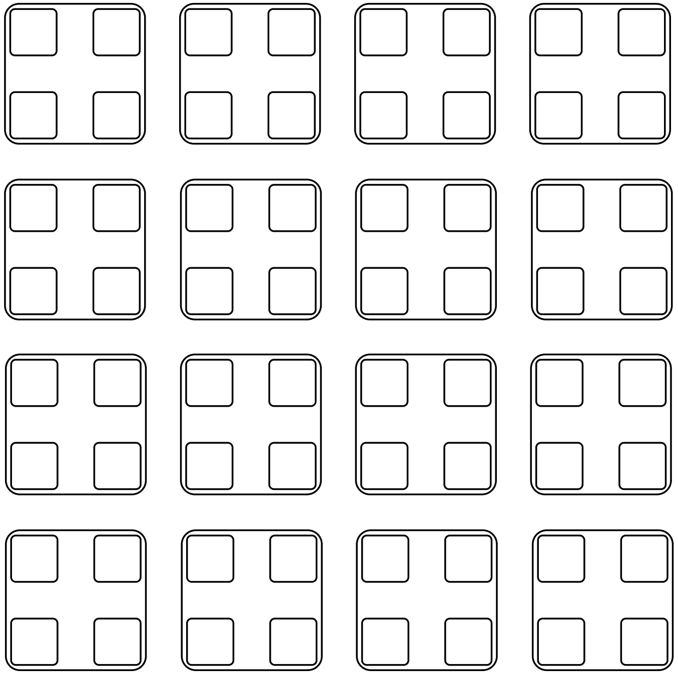


Flattened Butterfly *(Kim et al., Micro '07)*



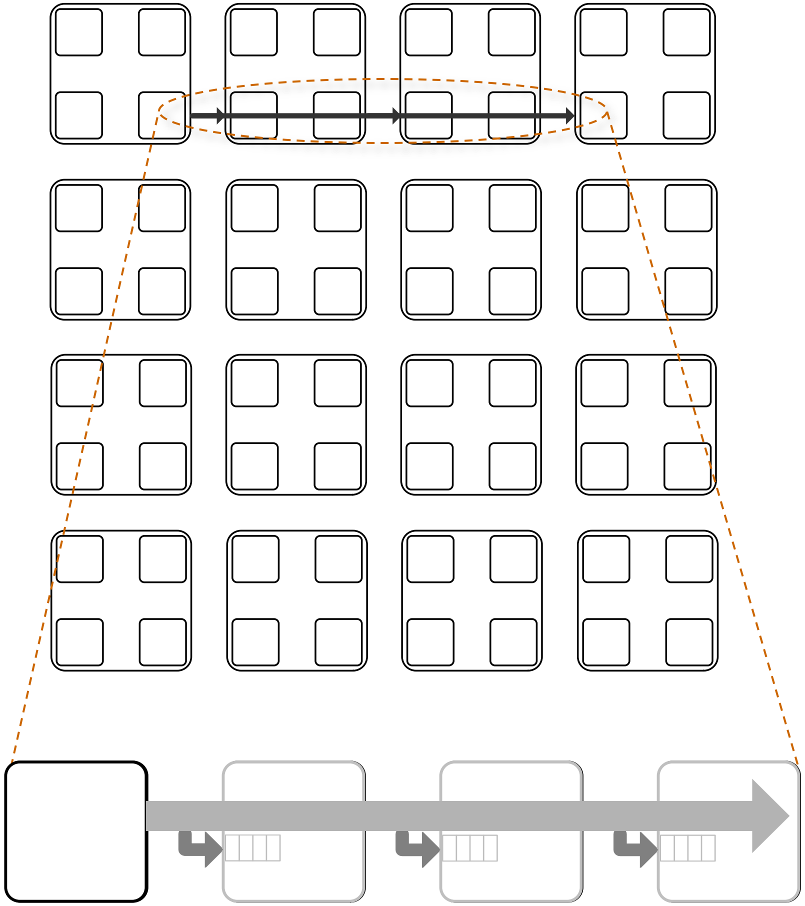
- Pros
  - Excellent connectivity
  - Low diameter: 2 hops
- Cons
  - High channel count: $k^2/2$ per row/column
  - Low channel utilization
  - Increased control (arbitration) complexity

Multidrop Express Channels (MECS)

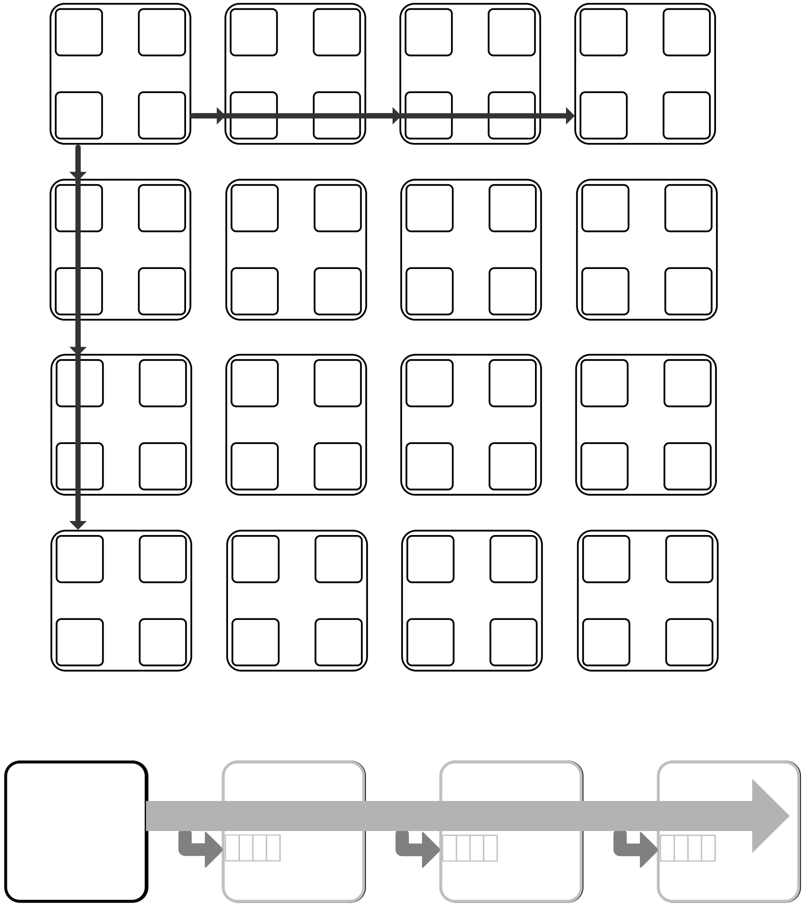


- Objectives:
  - Connectivity
  - More scalable channel count
  - Better channel utilization

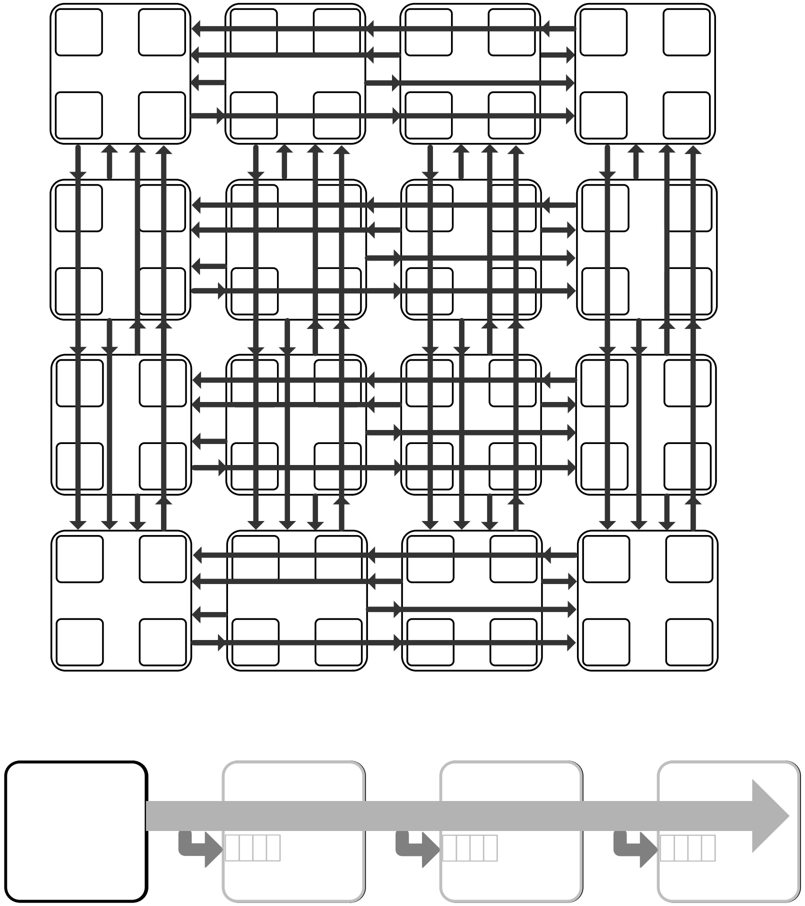
Multidrop Express Channels (MECS)



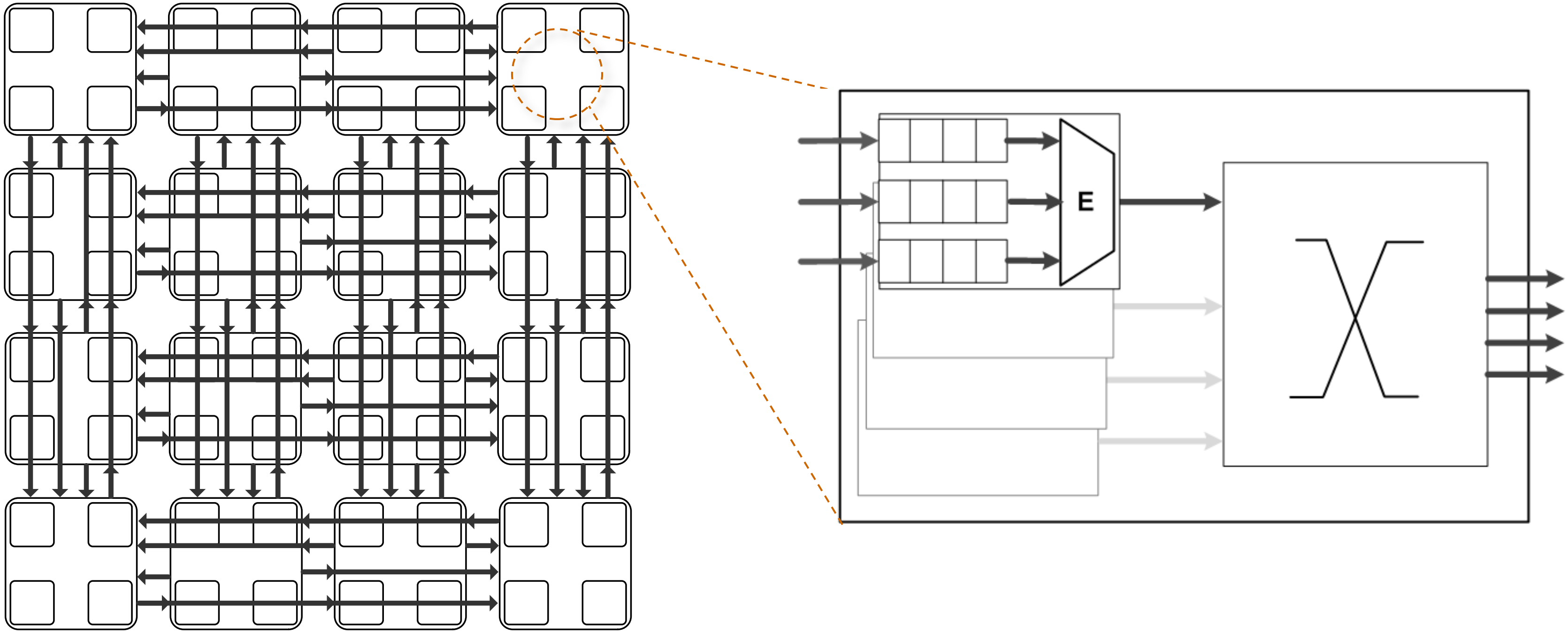
Multidrop Express Channels (MECS)



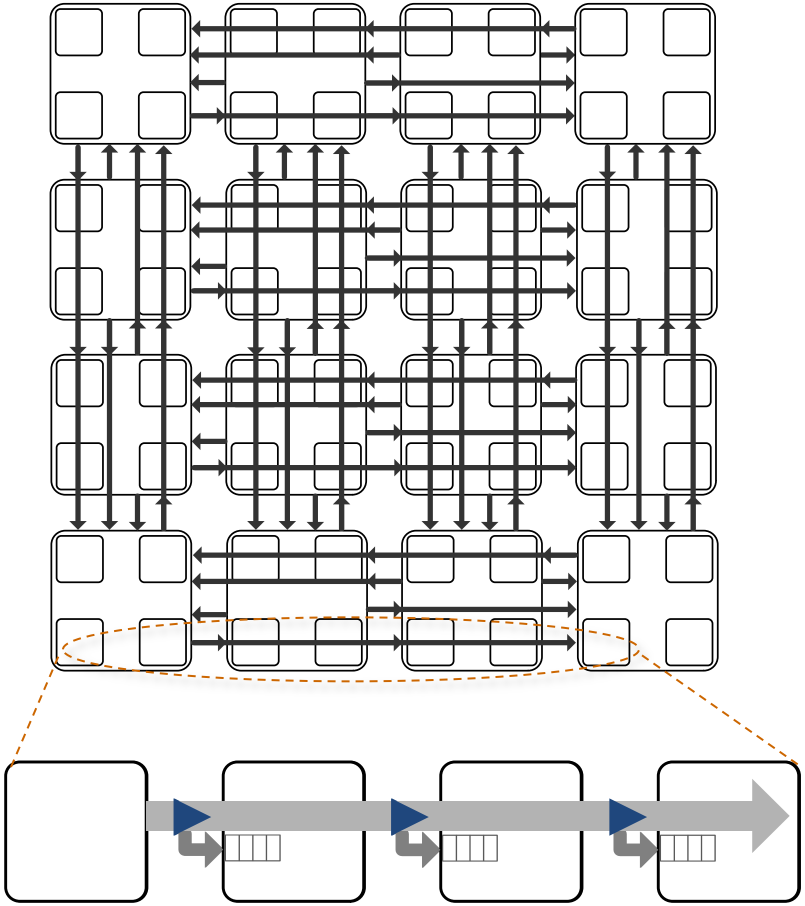
Multidrop Express Channels (MECS)



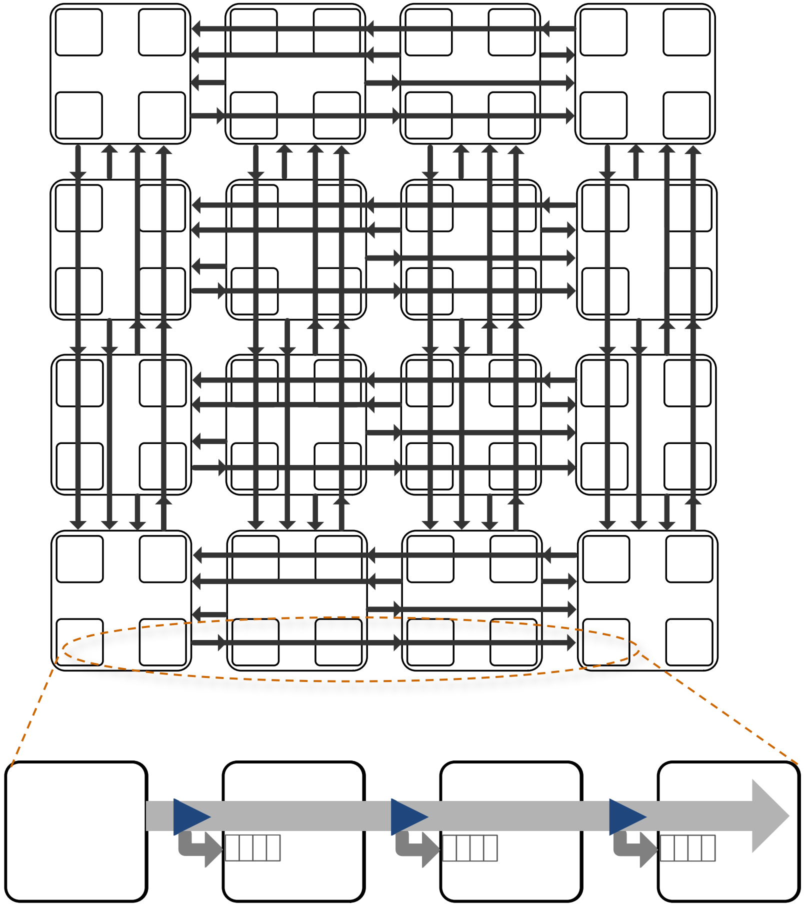
Multidrop Express Channels (MECS)



Multidrop Express Channels (MECS)

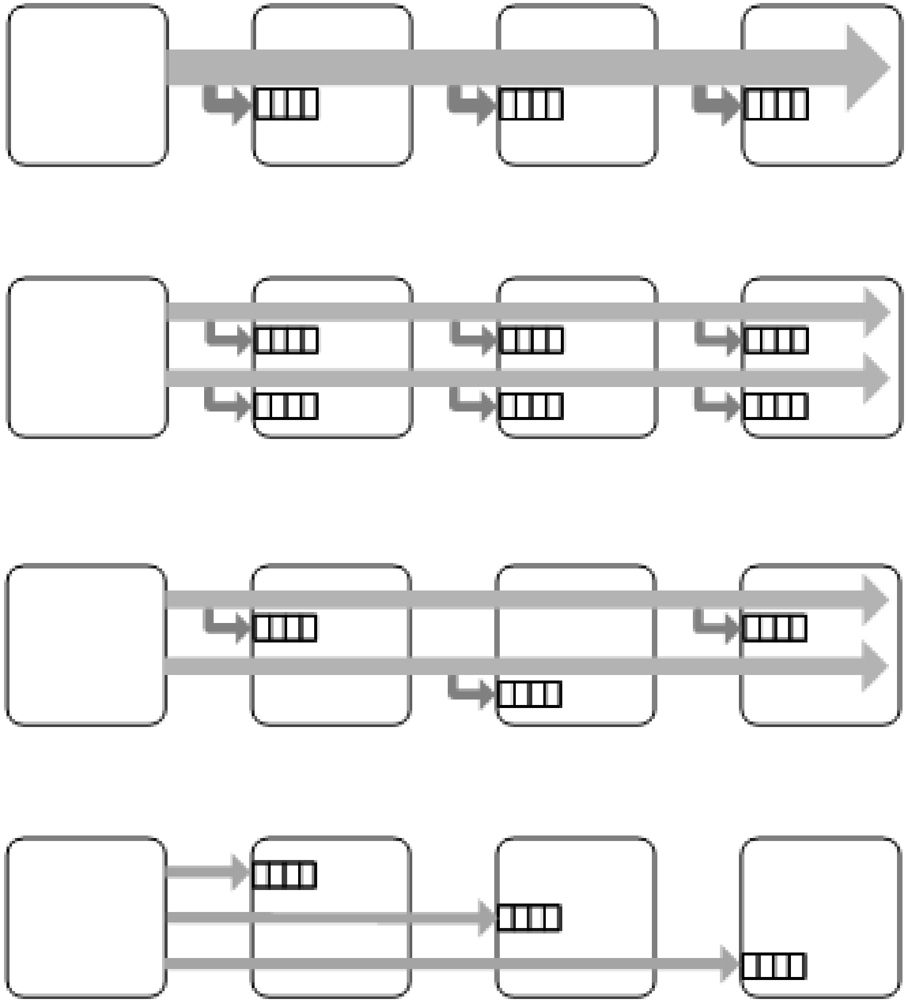


Multidrop Express Channels (MECS)



- 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

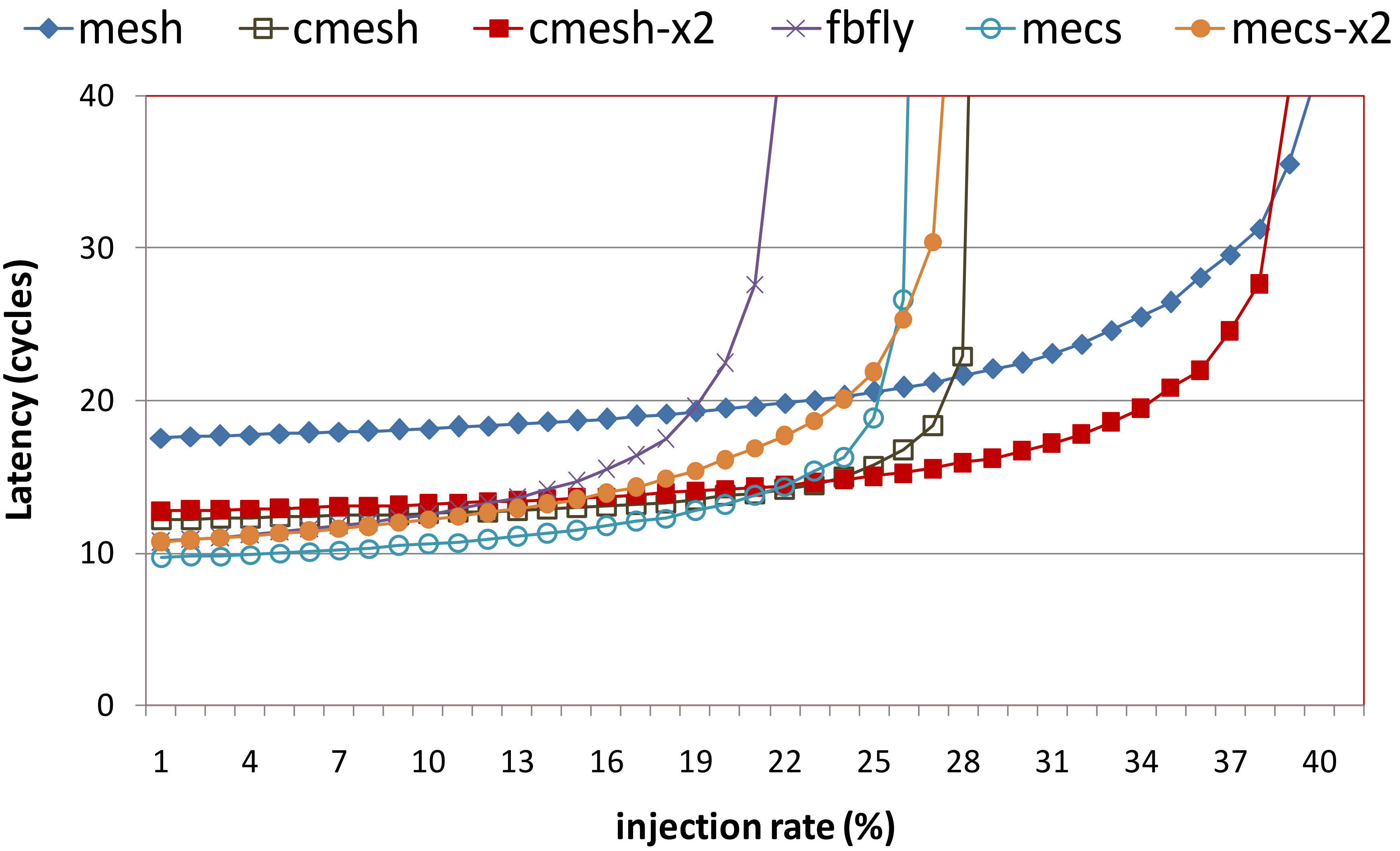
Analytical Comparison

	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

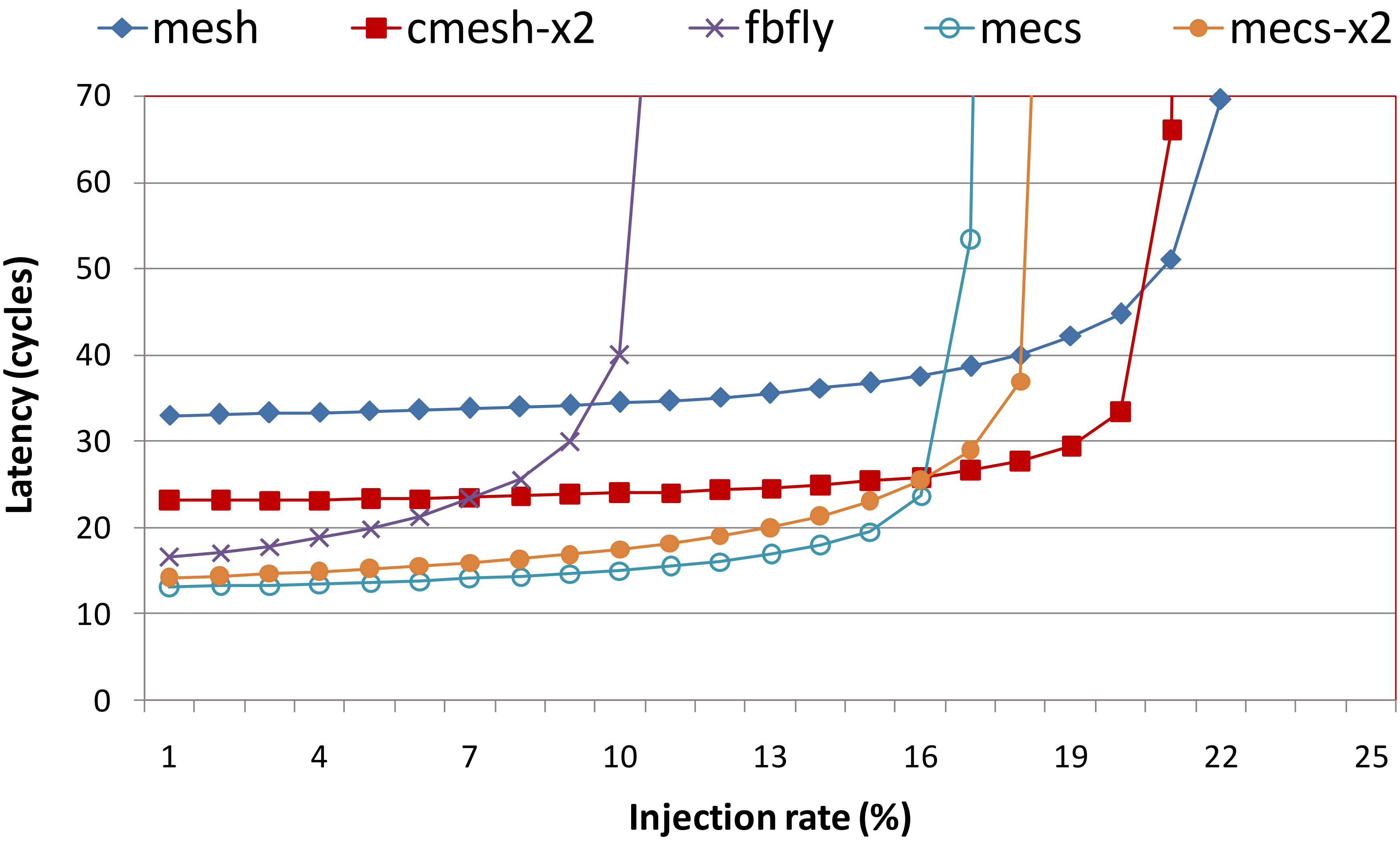
Experimental Methodology

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

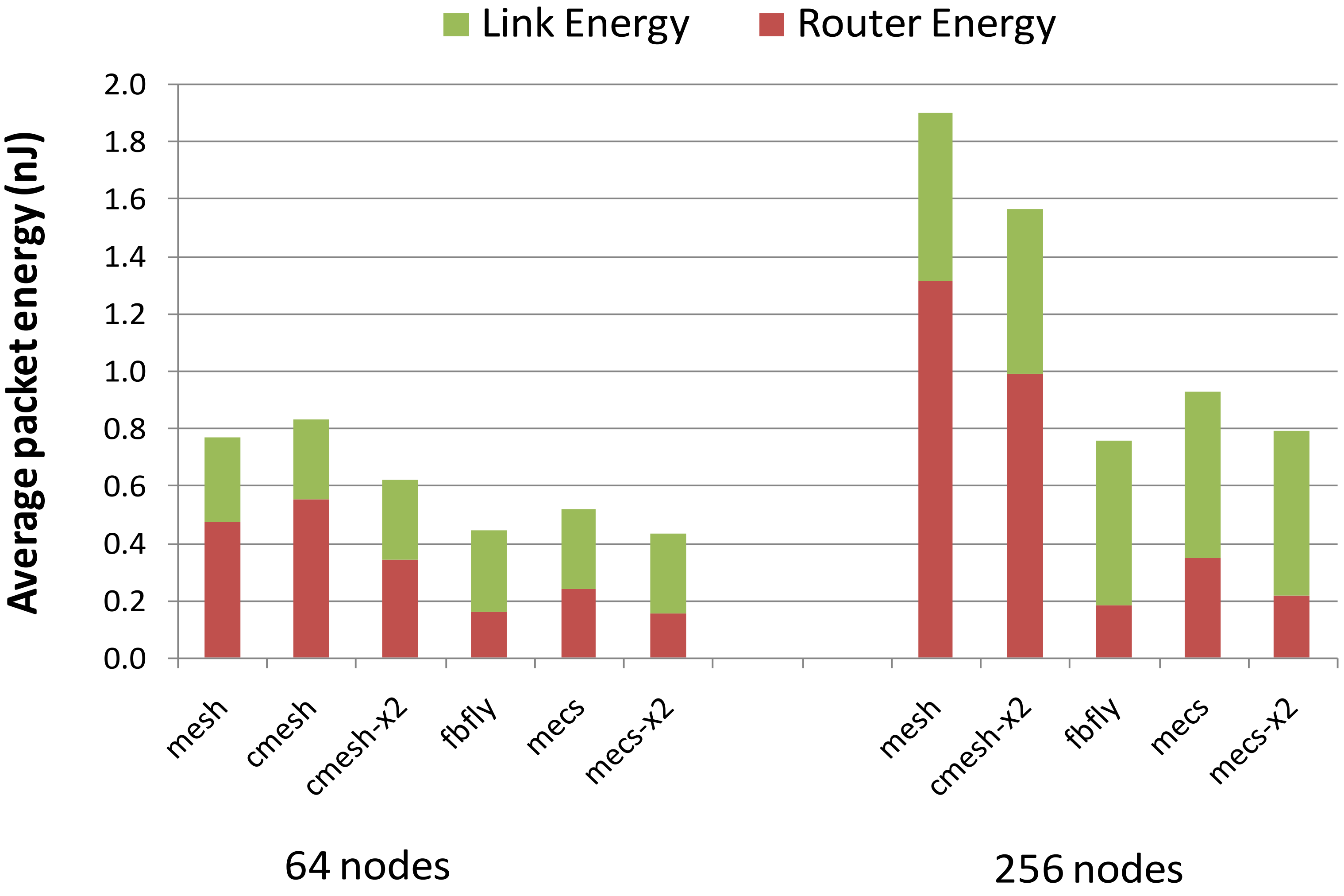
64 nodes: Uniform Random



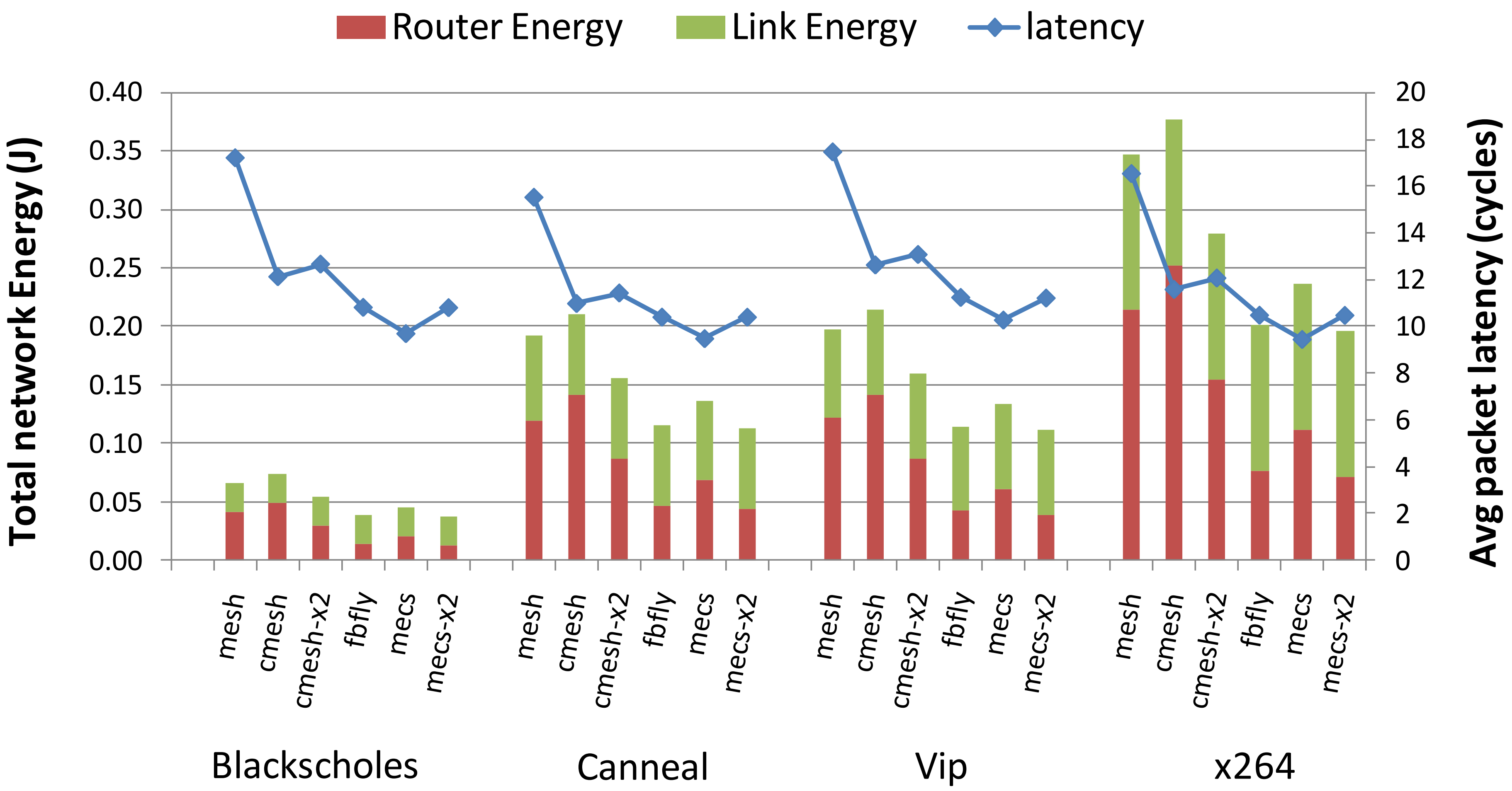
256 nodes: Uniform Random



Energy (100K pkts, Uniform Random)



64 Nodes: PARSEC



Summary

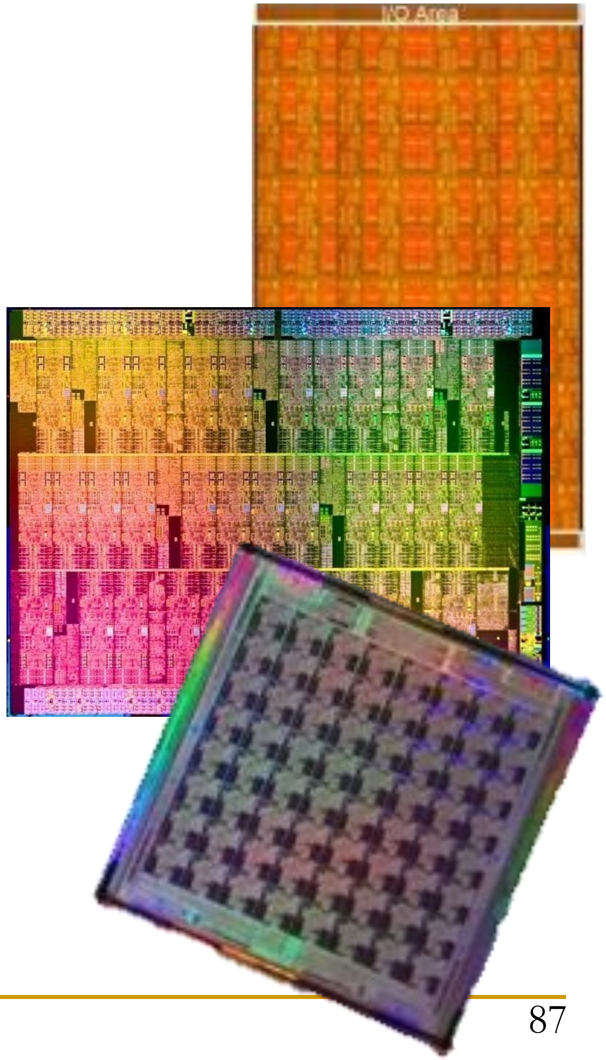
- MECS
  - A new one-to-many topology
  - Good fit for planar substrates
  - Excellent connectivity
  - Effective wire utilization
- Generalized Express Cubes
  - Framework & taxonomy for NOC topologies
  - Extension of the k-ary n-cube model
  - Useful for understanding and exploring on-chip interconnect options
  - Future: expand & formalize

Kilo-NoC: Topology-Aware QoS

Grot et al., "Kilo-NOC: A Heterogeneous Network-on-Chip Architecture for Scalability and Service Guarantees," ISCA 2011.

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)

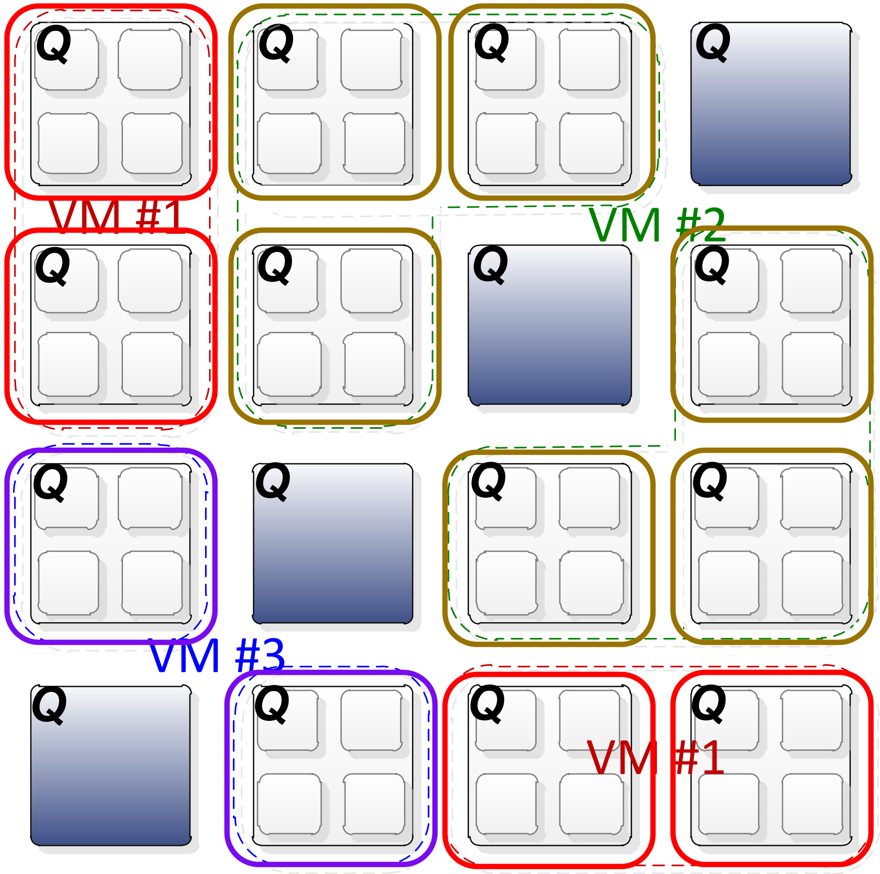
Topology-Aware QoS

- 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



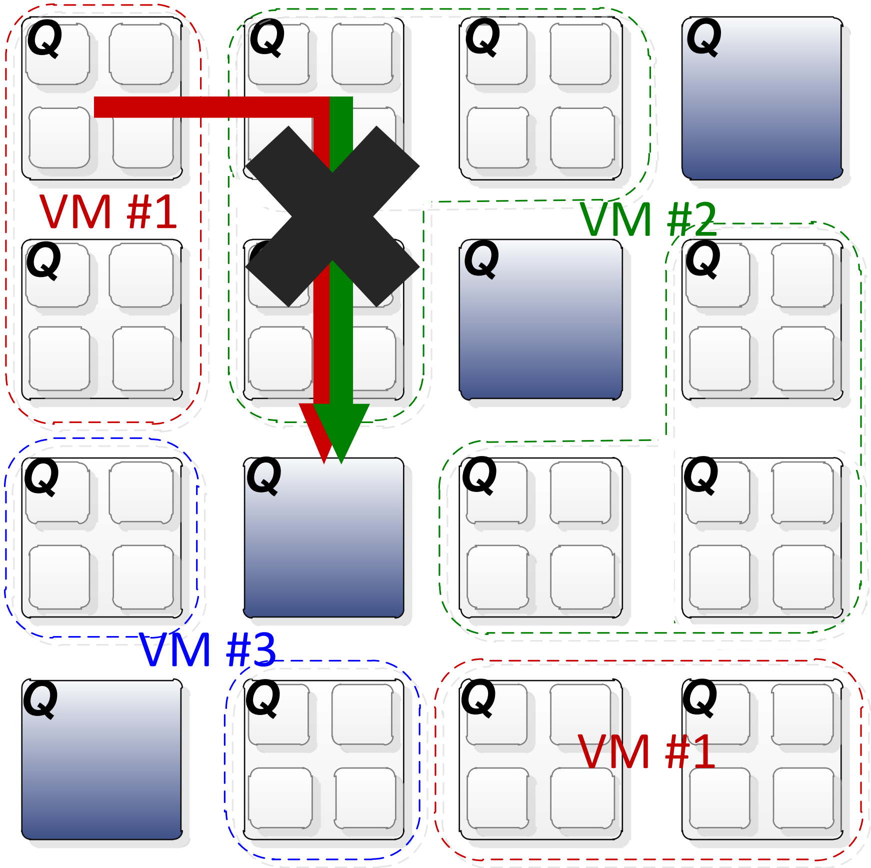
Multiple VMs
sharing a die

🟦 Shared resources
(e.g., memory controllers)

☐ VM-private resources
(cores, caches)

Q QOS-enabled router

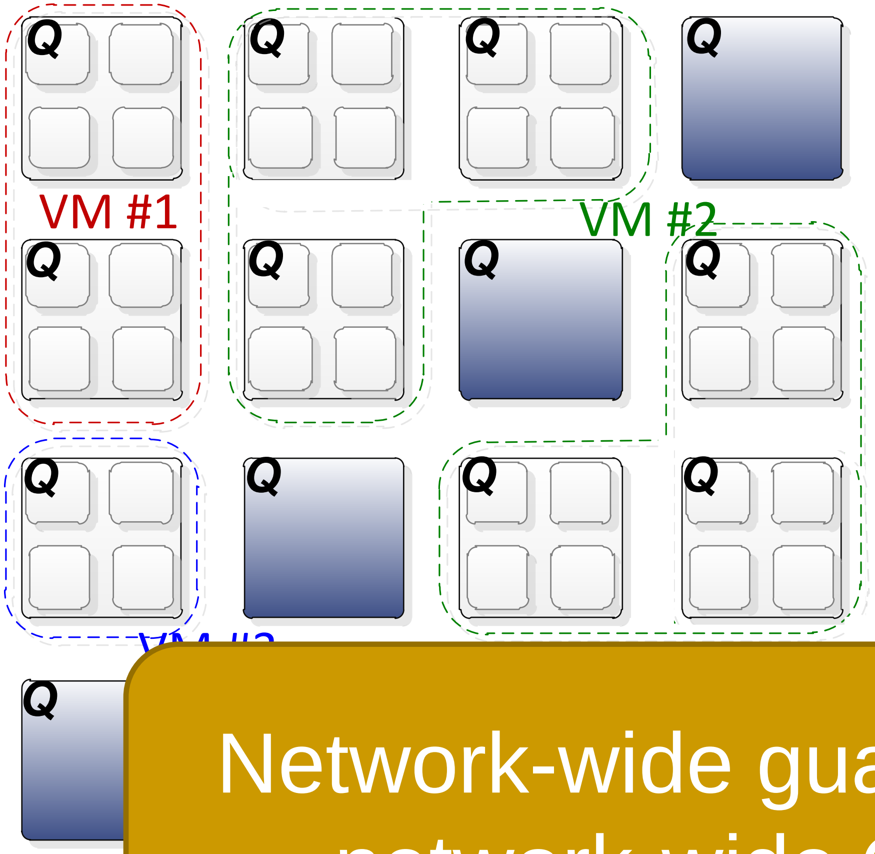
Conventional NOC QOS



Contention scenarios:

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

Conventional NOC QOS



Contention scenarios:

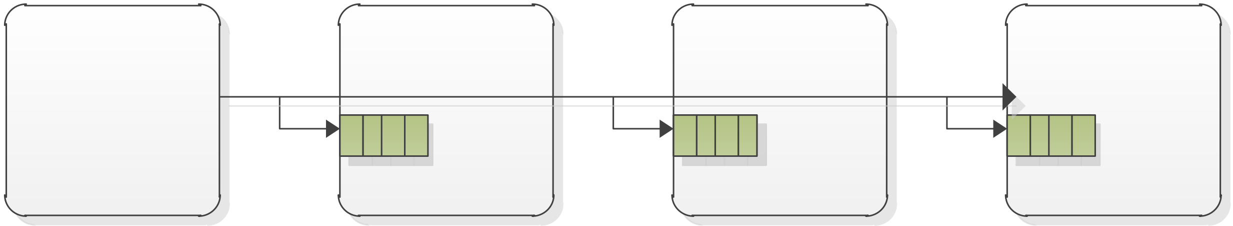
- Shared resources
  - ❑ memory access
- Intra-VM traffic
  - ❑ shared cache access
- Inter-VM traffic
  - ❑ VM page sharing

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

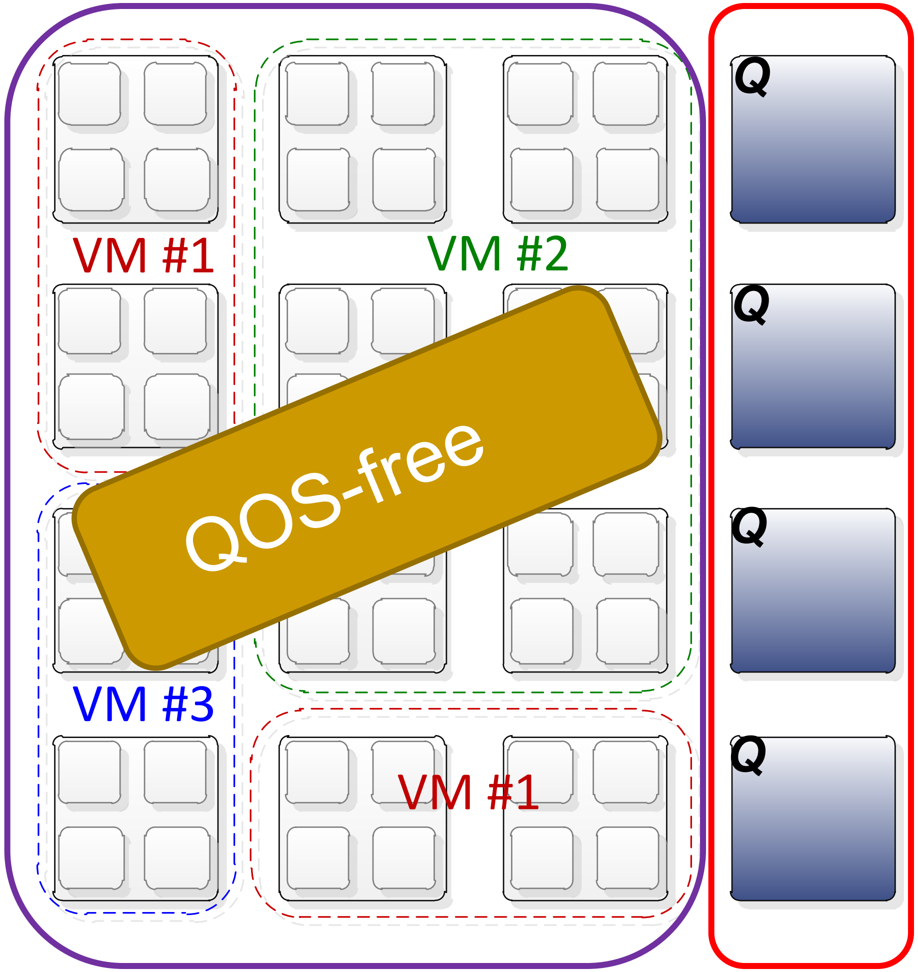
Kilo-NOC QOS

- 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

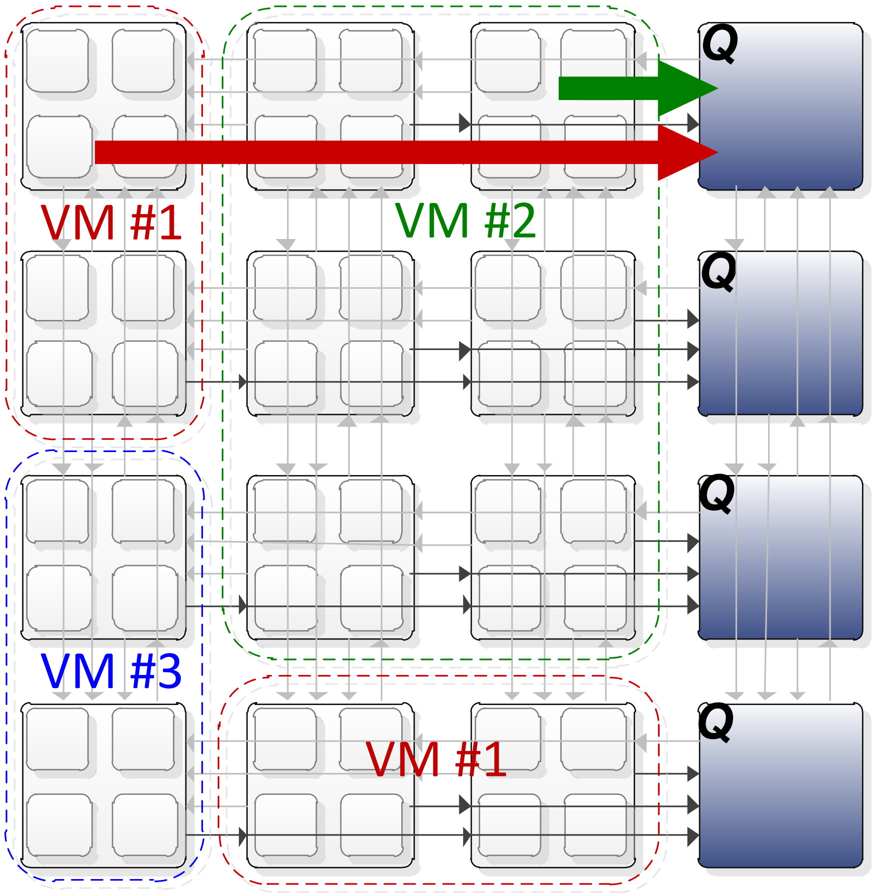


Topology-Aware QOS



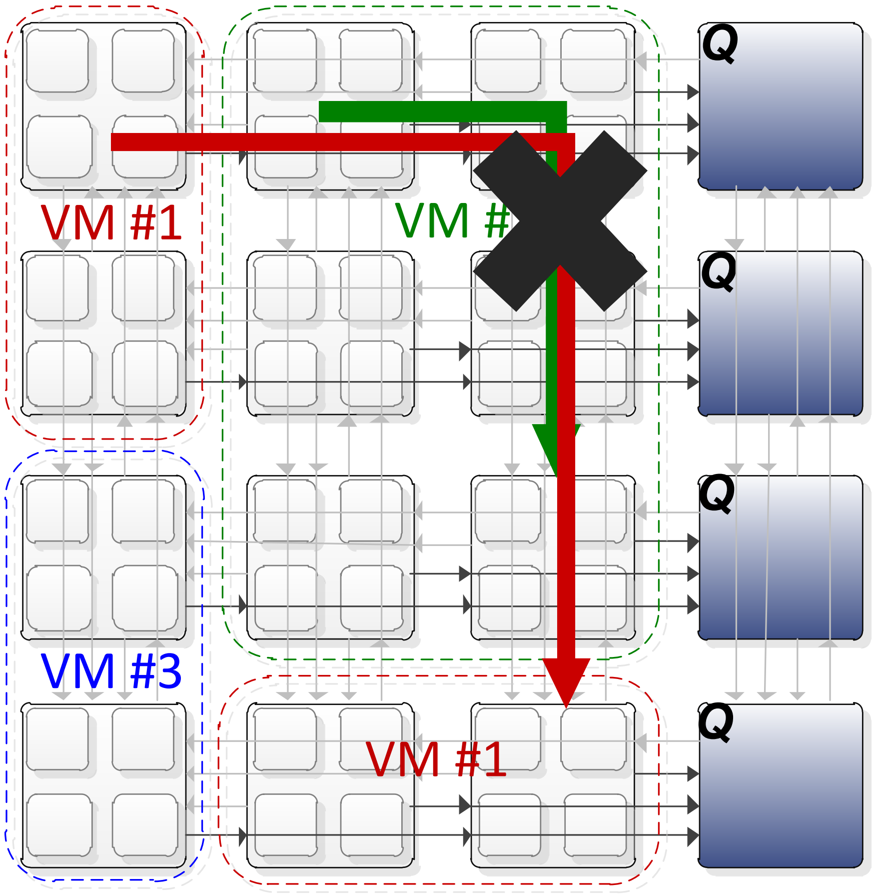
- Dedicated, QOS-enabled regions
    - Rest of die: QOS-free
- Richly-connected topology
    - Traffic isolation
- Special routing rules
    - Manage interference

Topology-Aware QOS



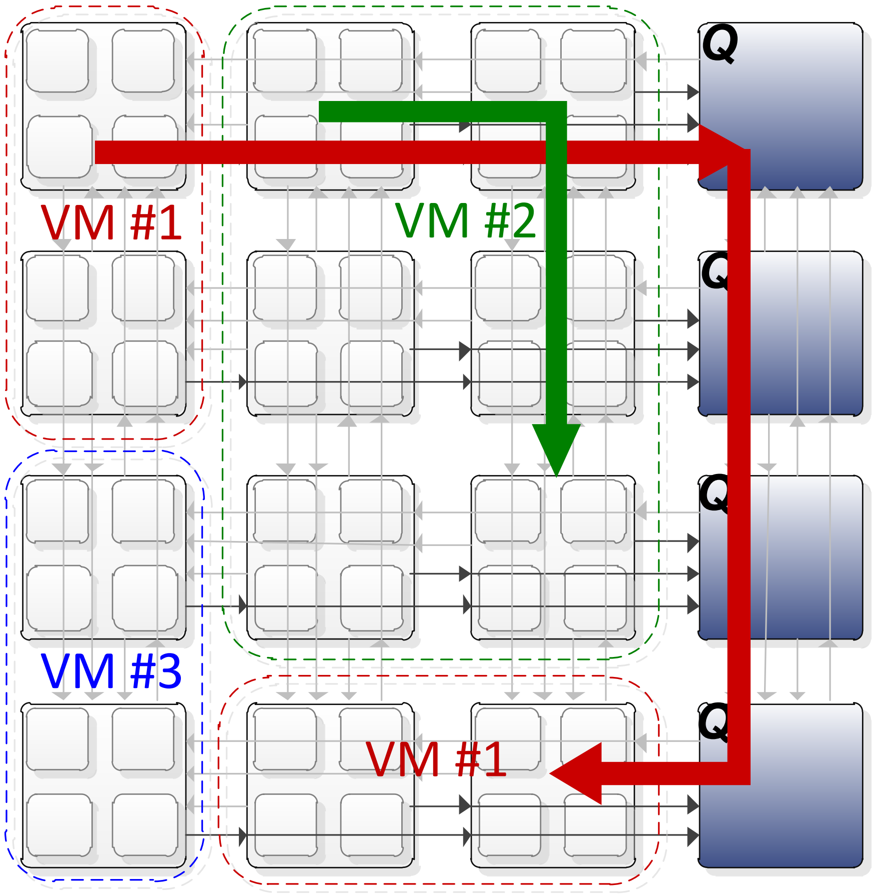
- Dedicated, QOS-enabled regions
  - Rest of die: QOS-free
- Richly-connected topology
  - Traffic isolation
- Special routing rules
  - Manage interference

Topology-Aware QOS



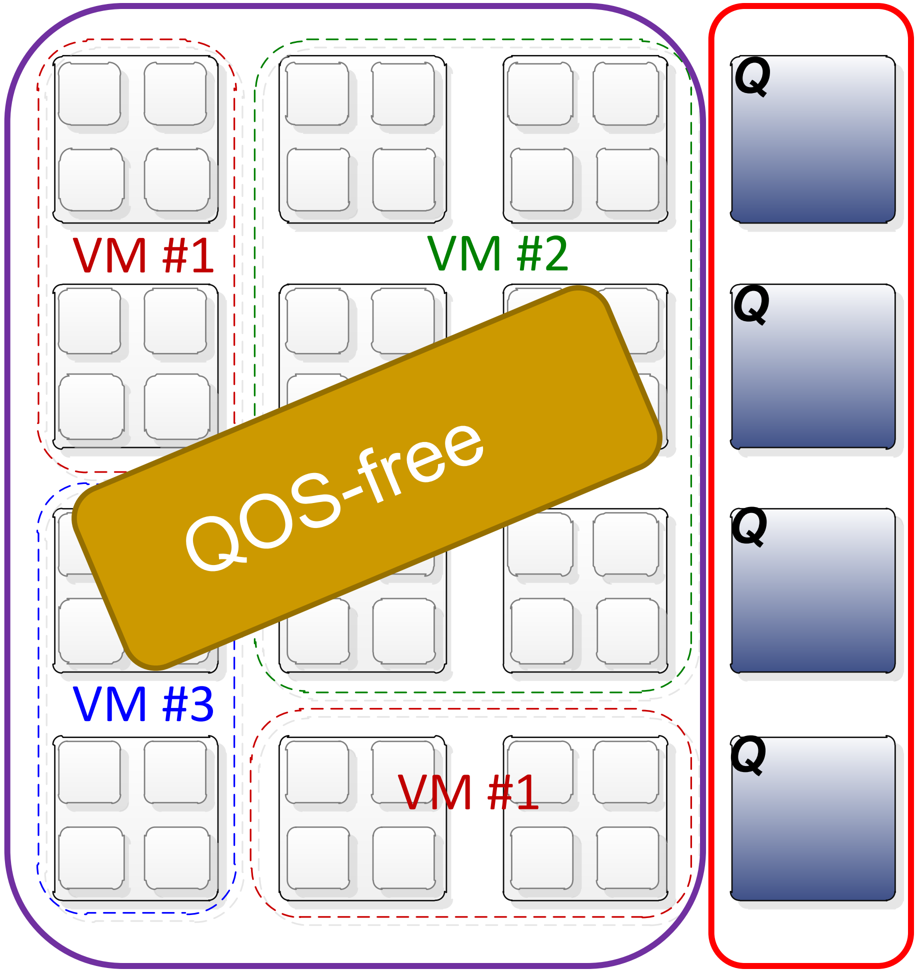
- Dedicated, QOS-enabled regions
  - Rest of die: QOS-free
- Richly-connected topology
  - Traffic isolation
- Special routing rules
  - Manage interference

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

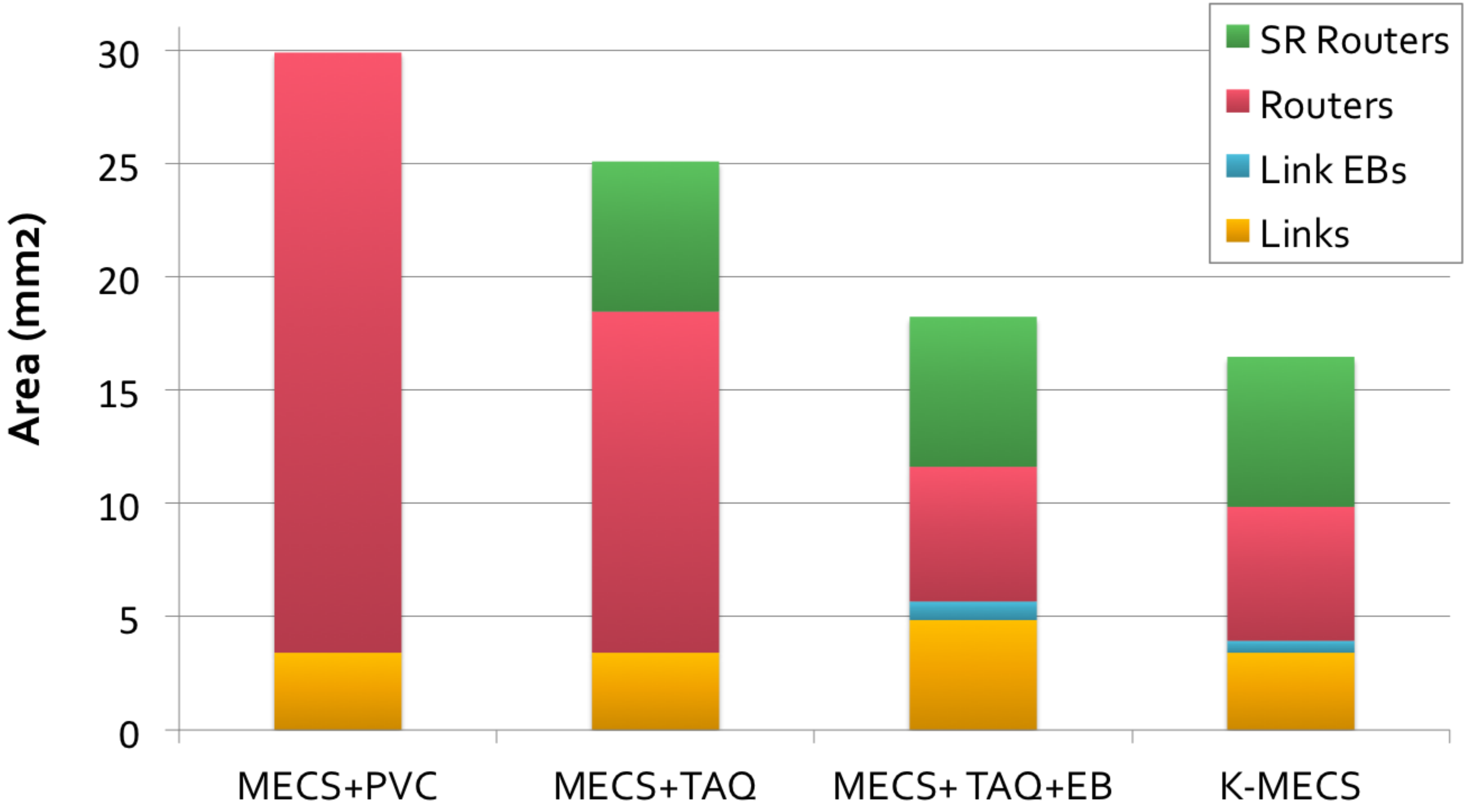


- Topology-aware QOS support
  - Limit QOS complexity to a fraction of the die
- Optimized flow control
  - Reduce buffer requirements in QOS-free regions

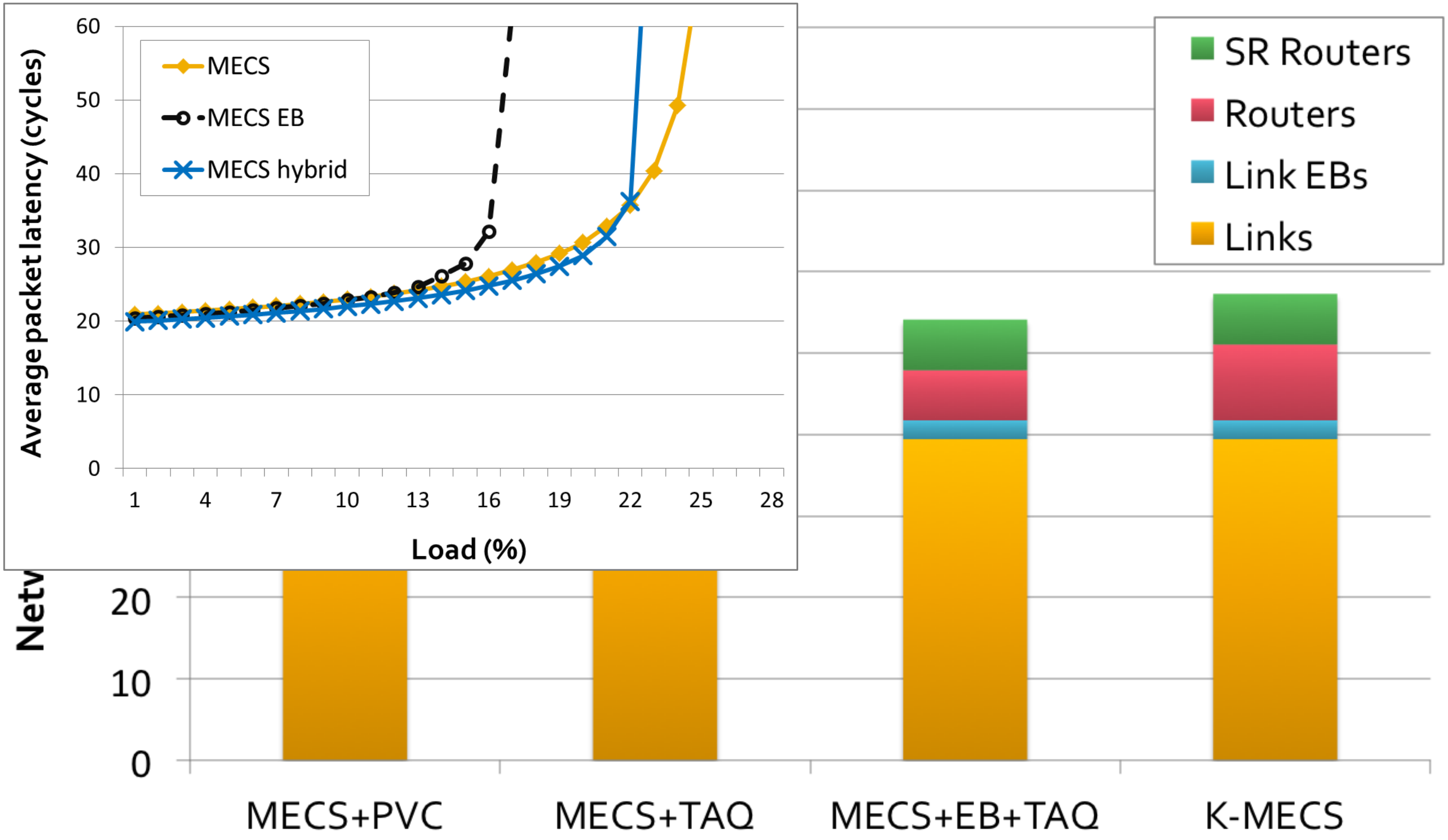
Evaluation Methodology

Parameter	Value
Technology	15 nm
Vdd	0.7 V
System	1024 tiles: 256 concentrated nodes (64 shared resources)
Networks:	
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 *Request* and *Reply* 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