Computer Architecture: Interconnects (Part II)

Michael Papamichael

Carnegie Mellon University

Material partially based on Onur Mutlu's 18-742 lecture slides from Spring 2010

Announcements

- Reviews due **today (November 1)**
  - Moscibroda and Mutlu, "A Case for Bufferless Routing in On-Chip Networks," ISCA 2009.
  - Fallin et al., "CHIPPER: A Low-Complexity Bufferless Deflection Router," HPCA 2011.

- Project milestones on **November 6**

Readings

- Required
  - Dally, "Route Packets, Not Wires: On-Chip Interconnection Networks," DAC 2001.
  - Das et al., "Application-Aware Prioritization Mechanisms for On-Chip Networks," MICRO 2009.
  - Chang et al., "HAT: Heterogeneous Adaptive Throttling for On-Chip Networks," SBAC-PAD 2012.
  - Fallin et al., "MinBD: Minimally-Buffered Deflection Routing for Energy-Efficient Interconnect," NOCS 2012.

- Please see website for more recommended readings

Agenda

- **Terminology review**
- More on interconnection networks
  - Network properties and performance metrics
  - Buffering and flow control
  - Router design and pipeline options

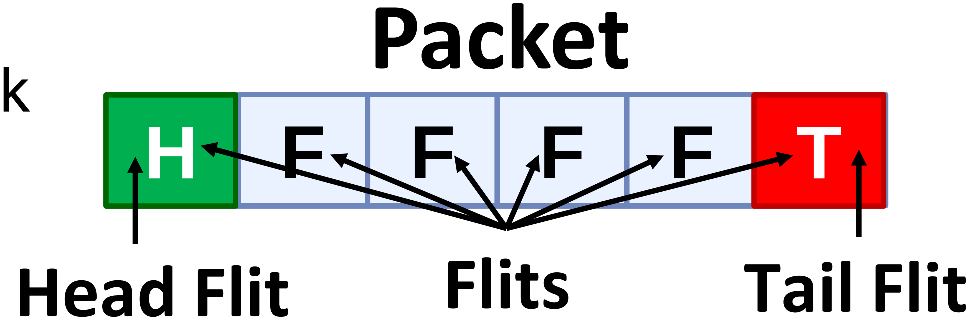
- Research on NoCs
  - The problem with packet scheduling
  - Application-aware packet scheduling
  - Aergia: Latency slack-based packet scheduling
  - Bufferless networks

Basic Terminology (review)

- **Topology**
  - Specifies way switches are wired
- **Routing (algorithm)**
  - How does a message get from source to destination
- **Buffering and Flow Control**
  - Managing and communicating buffer space
- **Switch/router**
  - Connects fixed set of inputs to fixed set of outputs
- **Channel**
  - A single logical connection between routers/switches

Basic Terminology (review)

- **Node**
  - A switch/router or client/endpoint that is part of the network
- **Message**
  - Unit of transfer for network's clients (processors, memory)
- **Packet**
  - Unit of transfer for network
- **Flit**
  - Flow control digit
  - Unit of flow control within network



Agenda

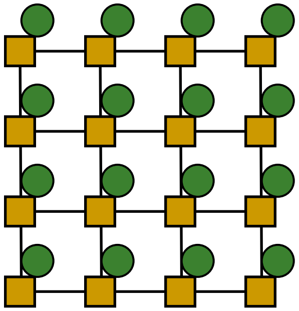
- Terminology review
- **More on interconnection networks**
  - **Network properties and performance metrics**
  - Buffering and flow control
  - Router design and pipeline options
- Research on NoCs
  - The problem with packet scheduling
  - Application-aware packet scheduling
  - Aergia: Latency slack-based packet scheduling
  - Bufferless networks

Properties of a Topology/Network

- **Regular or Irregular**
  - Regular if topology is regular graph (e.g. ring, mesh).

- **Routing Distance**
  - Number of links/hops along route

- **Diameter**
  - Maximum routing distance



diameter = 6

- **Average Distance**
  - Average number of hops across all valid routes

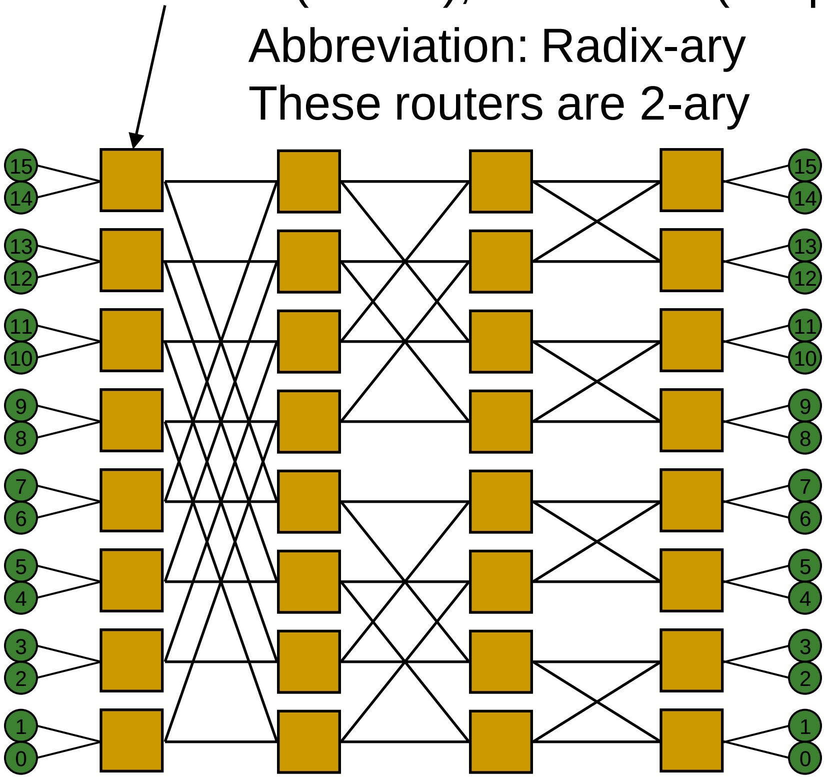
Properties of a Topology/Network

- **Direct or Indirect Networks**
  - Endpoints sit “inside” (direct) or “outside” (indirect) the network
  - E.g. mesh is direct; every node is both endpoint and switch

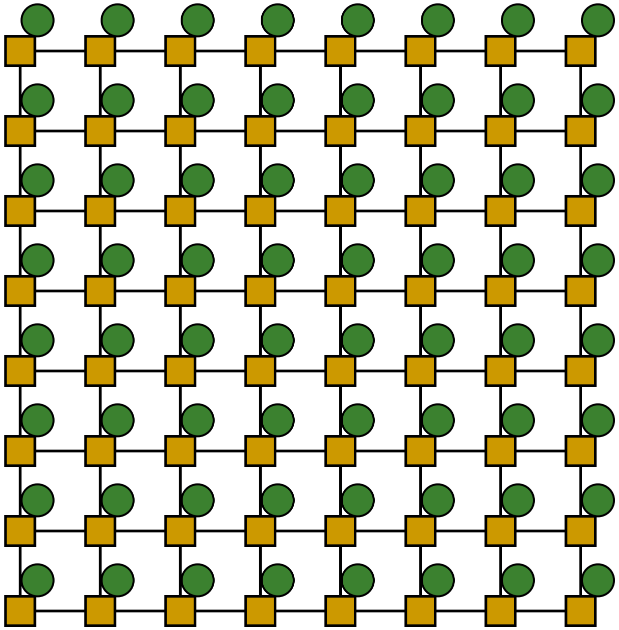
Router (switch), Radix of 2 (2 inputs, 2 outputs)

Abbreviation: Radix-ary

These routers are 2-ary



Indirect



Direct

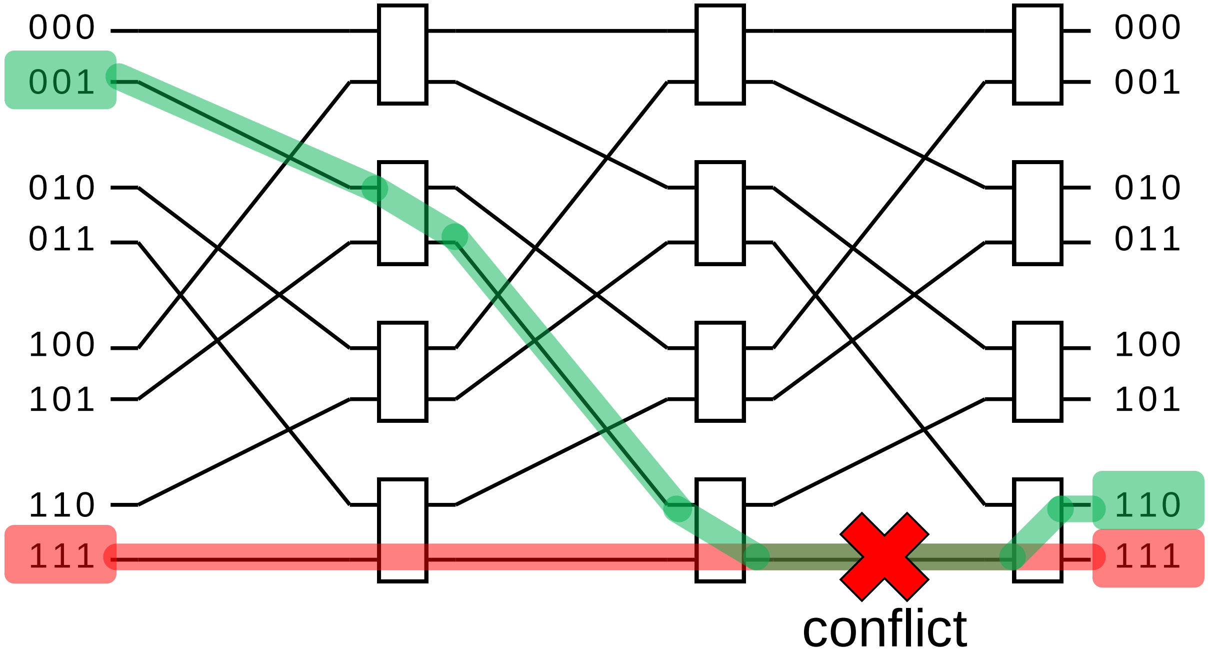
Properties of a Topology/Network

- **Bisection Bandwidth**
  - Often used to describe network performance
  - Cut network in half and sum bandwidth of links severed
    - (Min # channels spanning two halves) * (BW of each channel)
  - Meaningful only for recursive topologies
  - Can be misleading, because does not account for switch and routing efficiency

- **Blocking vs. Non-Blocking**
  - If connecting any permutation of sources & destinations is possible, network is <u>non-blocking</u>; otherwise network is <u>blocking</u>.
  - <u>Rearrangeable non-blocking</u>: Same as non-blocking but might require rearranging connections when switching from one permutation to another.

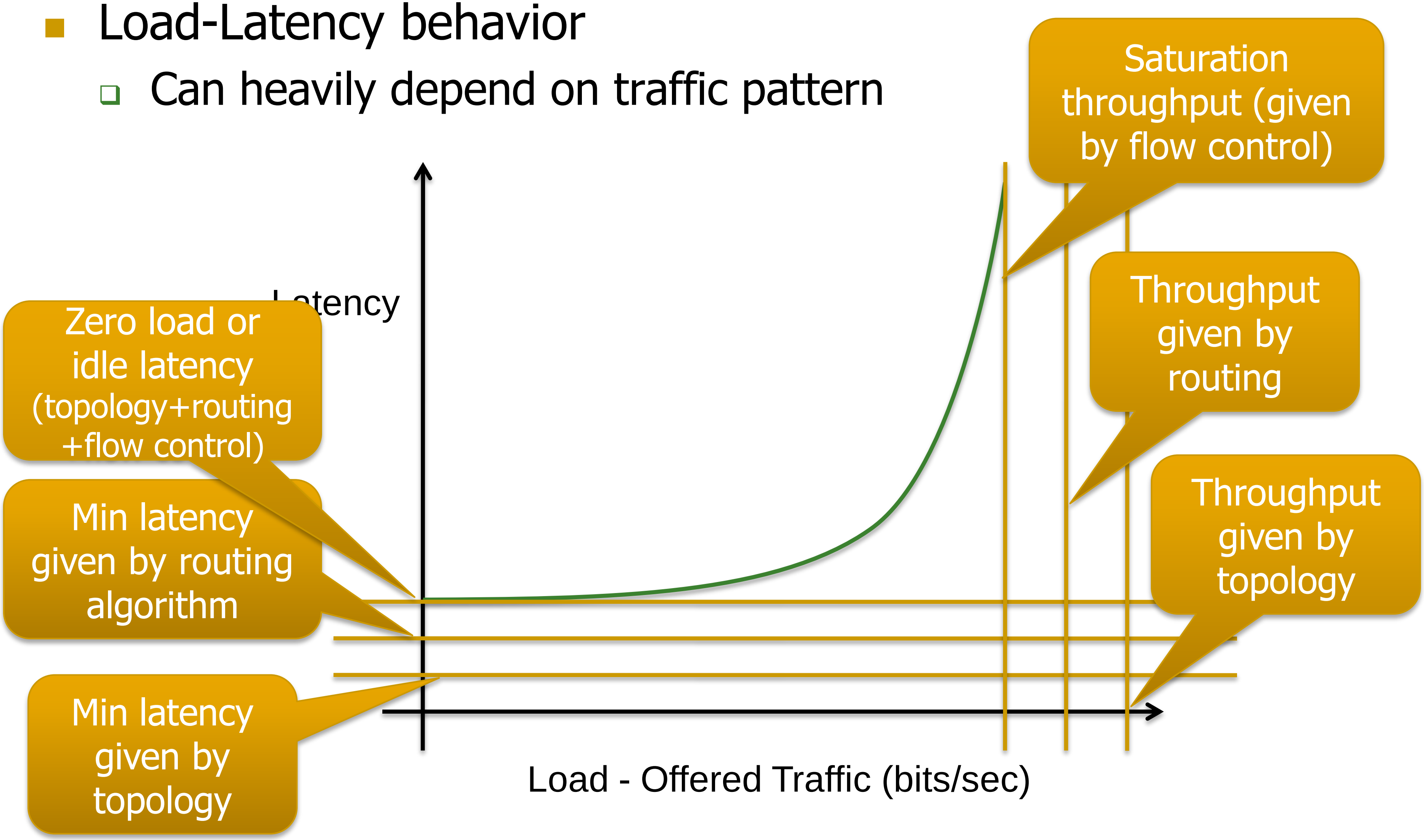
Blocking vs. Non-Blocking Example

- What type of topology/network is this?
  - Multistage Logarithmic (Omega)
- Is this blocking or non-blocking?
  - Blocking



Interconnection Network Performance

- Load-Latency behavior
  - Can heavily depend on traffic pattern



Zero load or idle latency (topology+routing +flow control)

Min latency given by routing algorithm

Min latency given by topology

Saturation throughput (given by flow control)

Throughput given by routing

Throughput given by topology

Ideal Latency

- Ideal latency
  - Solely due to wire delay between source and destination

$$T_{ideal} = \frac{D}{v} + \frac{L}{b}$$

  - D = Manhattan distance
  - L = packet size
  - b = channel bandwidth
  - v = propagation velocity

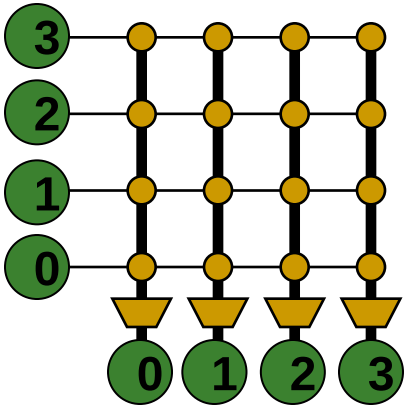
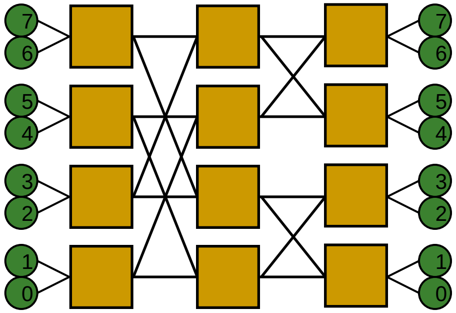
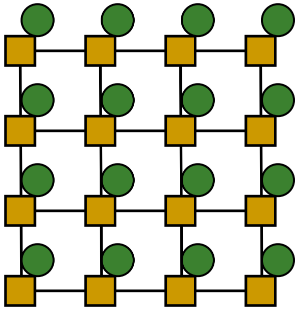
Actual Latency

- Dedicated wiring impractical
  - Long wires segmented with insertion of routers

$$T_{actual} = \frac{D}{v} + \frac{L}{b} + H \cdot T_{router} + T_c$$

  - D = Manhattan distance
  - L = packet size
  - b = channel bandwidth
  - v = propagation velocity
  - H = hops
  - T_{router} = router latency
  - T_c = latency due to contention

Review

		 7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0	
Topology	**Crossbar**	**Multistage Logarith.**	**Mesh**
Direct/Indirect	**Indirect**	**Indirect**	**Direct**
Blocking/ Non-blocking	**Non-blocking**	**Blocking** (this particular one)	**Blocking**
Cost	**O(N^2)**	**O(NlogN)**	**O(N)**
Latency	**O(1)**	**O(logN)**	**O(sqrt(N))**

Agenda

- Terminology review
- **More on interconnection networks**
  - Network properties and performance metrics
  - **Buffering and flow control**
  - Router design and pipeline options

- Research on NoCs
  - The problem with packet scheduling
  - Application-aware packet scheduling
  - Aergia: Latency slack-based packet scheduling
  - Bufferless networks

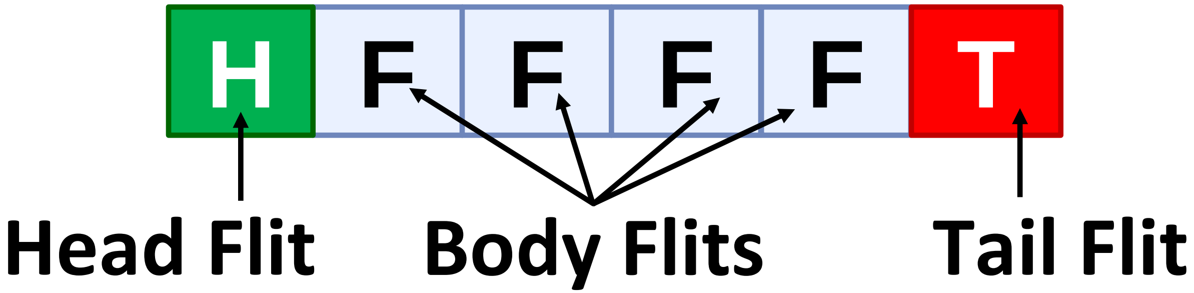
Circuit vs. Packet Switching (review)

- Circuit switching sets up full path
  - Establish route then send data
  - (no one else can use those links)
  - faster and higher bandwidth
  - setting up and bringing down links slow

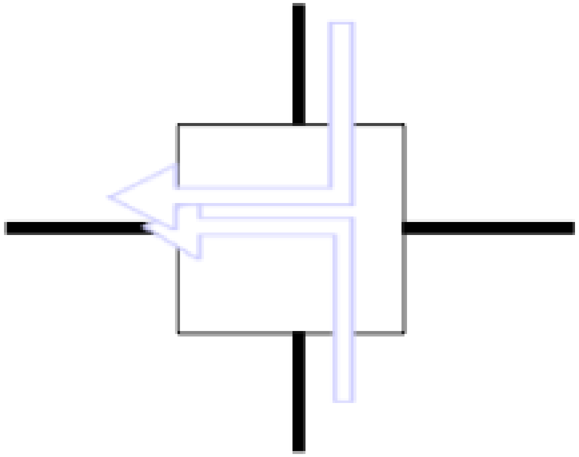
- Packet switching routes per packet
  - Route each packet individually (possibly via different paths)
  - if link is free can use
  - potentially slower (must dynamically switch)
  - no setup, bring down time

Packet Switched Networks: Packet Format

- **Header**
  - routing and control information
  - at start so router can start forwarding early
- **Payload/Body**
  - carries data (non HW specific information)
  - can be further divided (framing, protocol stacks…)
- **Tail**
  - contains control information, e.g. error code
  - at end of packet so it can be generated on the way out



Handling Contention



- Two packets trying to use the same link at the same time
- What do you do?
  - Buffer one
  - Drop one
  - Misroute one (deflection)
- Assume buffering for now

Flow Control Methods

- **Circuit switching**
- **Store and forward** (Packet based)
- **Virtual cut through** (Packet based)
- **Wormhole** (Flit based)

Circuit Switching Revisited

- Resource allocation granularity is high

- Idea: Pre-allocate resources across multiple switches for a given "flow"
- Need to send a probe to set up the path for pre-allocation

+ No need for buffering

+ No contention (flow's performance is isolated)

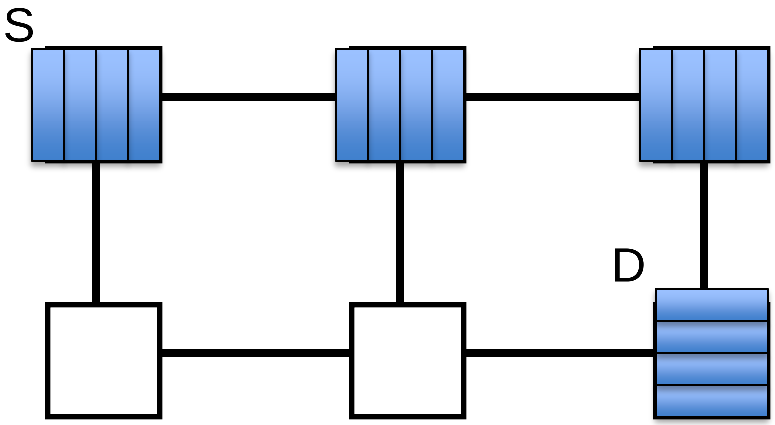
+ Can handle arbitrary message sizes

- Lower link utilization: two flows cannot use the same link

- Handshake overhead to set up a "circuit"

Store and Forward Flow Control

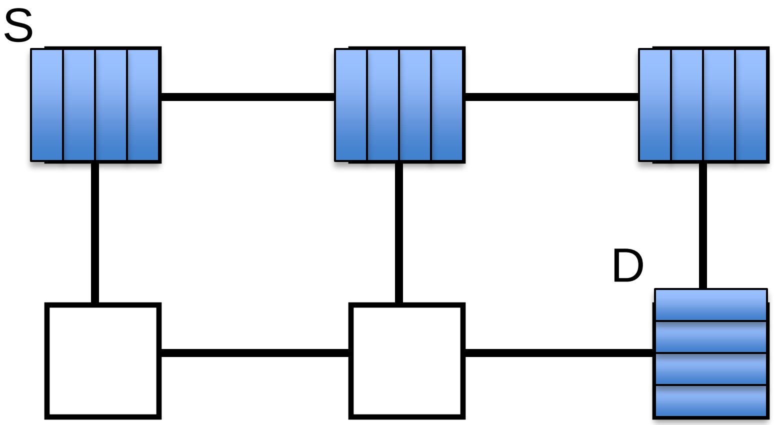
- Packet based flow control
- Store and Forward
  - Packet copied entirely into network router before moving to the next node
  - Flow control unit is the entire packet
- Leads to high per-packet latency
- Requires buffering for entire packet in each node



Can we do better?

Cut through Flow Control

- Another form of packet based flow control
- Start forwarding as soon as header is received and resources (buffer, channel, etc) allocated
  - Dramatic reduction in latency
- Still allocate buffers and channel bandwidth for full packets



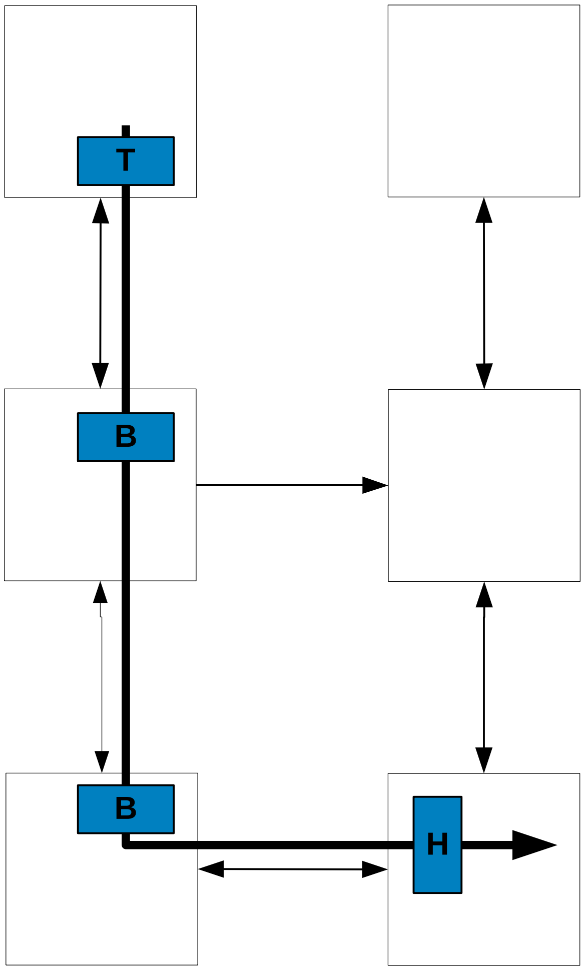
- What if packets are large?

Cut through Flow Control

- What to do if output port is blocked?
- Lets the tail continue when the head is blocked, absorbing the whole message into a single switch.
  - Requires a buffer large enough to hold the largest packet.
- Degenerates to store-and-forward with high contention

- **Can we do better?**

Wormhole Flow Control



- Packets broken into (potentially) smaller flits (buffer/bw allocation unit)
- Flits are sent across the fabric in a *wormhole fashion*
  - Body follows head, tail follows body
  - Pipelined
  - If head blocked, rest of packet stops
  - Routing (src/dest) information only in head

- How does body/tail know where to go?
- Latency almost independent of distance for long messages

Wormhole Flow Control

- **Advantages over "store and forward" flow control**

  + Lower latency

  + More efficient buffer utilization

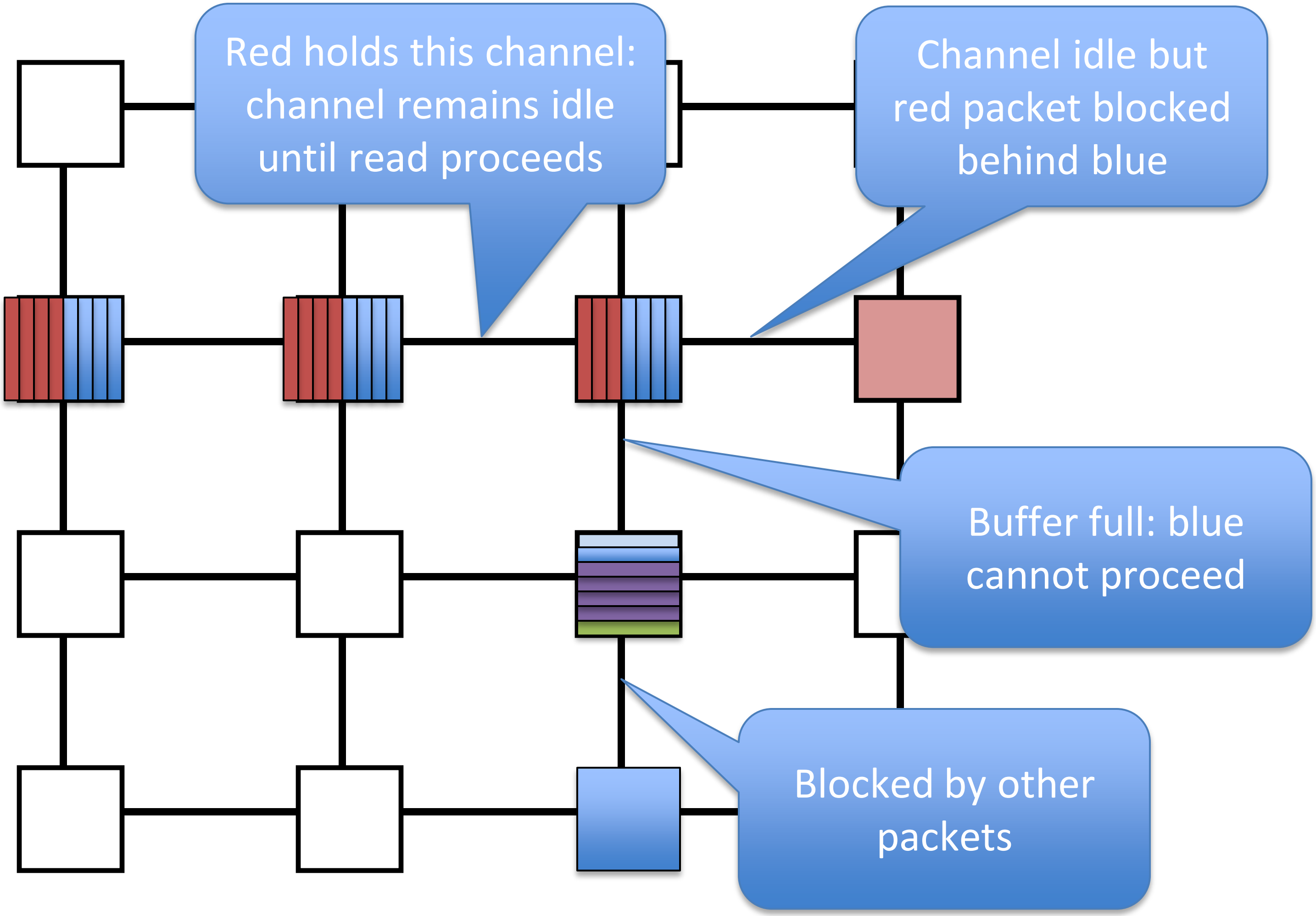
- **Limitations**

  - Occupies resources across multiple routers

  - Suffers from **head of line blocking**

    - if head flit cannot move due to contention, another worm cannot proceed even though links may be idle

Head of Line Blocking



Red holds this channel: channel remains idle until read proceeds

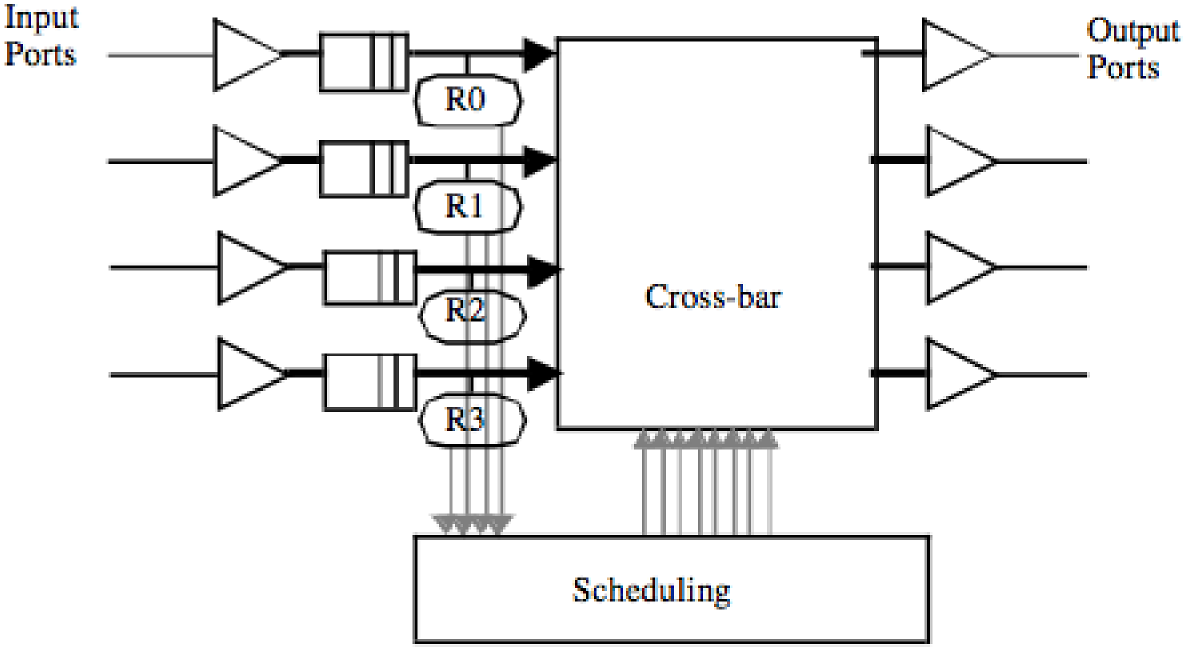
Channel idle but red packet blocked behind blue

Buffer full: blue cannot proceed

Blocked by other packets

Head of Line Blocking

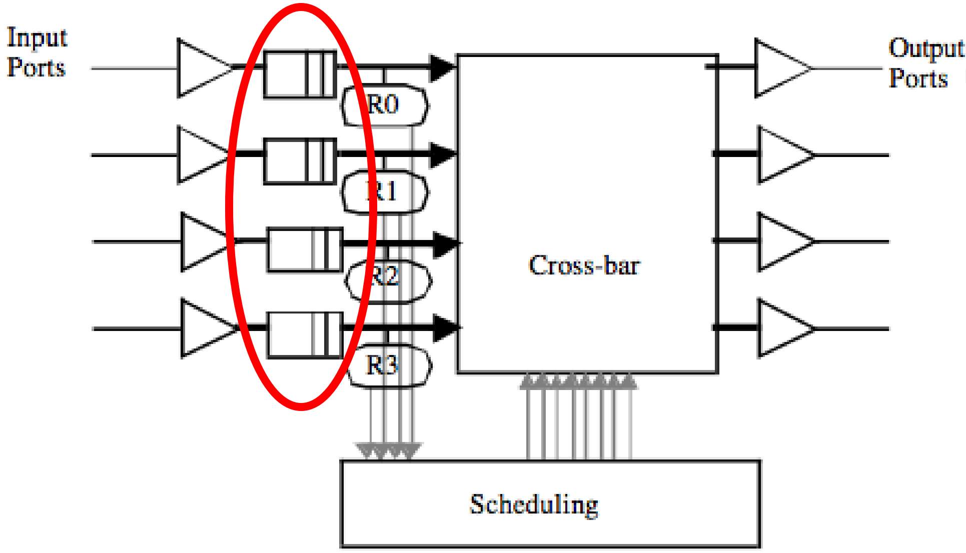
- A worm can be before another in the router input buffer
- Due to FIFO nature, the second worm cannot be scheduled even though it may need to access another output port

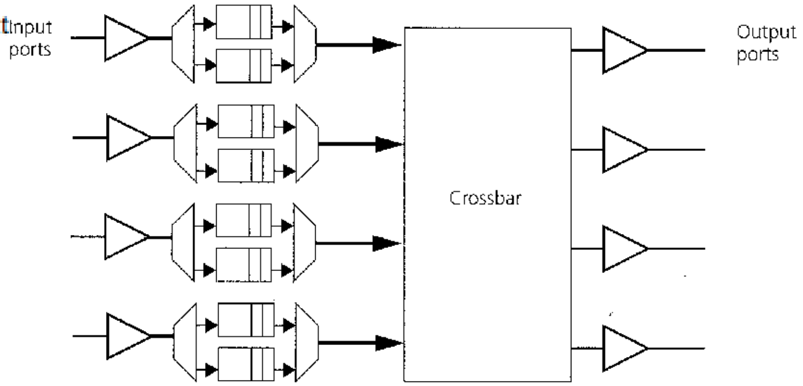


Karo et al., "Input Versus Output Queuing on a Space-Division Packet Switch," IEEE Transactions on Communications 1987

Virtual Channel Flow Control

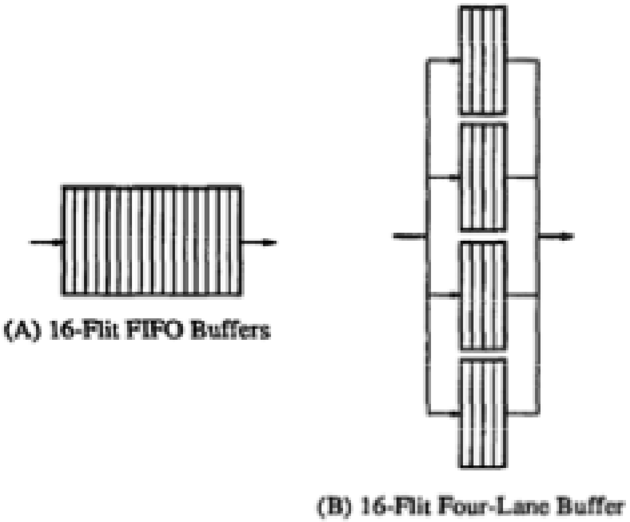
- Idea: Multiplex multiple channels over one physical channel
- Divide up the input buffer into multiple buffers sharing a single physical channel
- Dally, "Virtual Channel Flow Control," ISCA 1990.





Virtual Channel Flow Control

- Idea: Multiplex multiple channels over one physical channel
- Divide up the input buffer into multiple buffers sharing a single physical channel
- Dally, "Virtual Channel Flow Control," ISCA 1990.

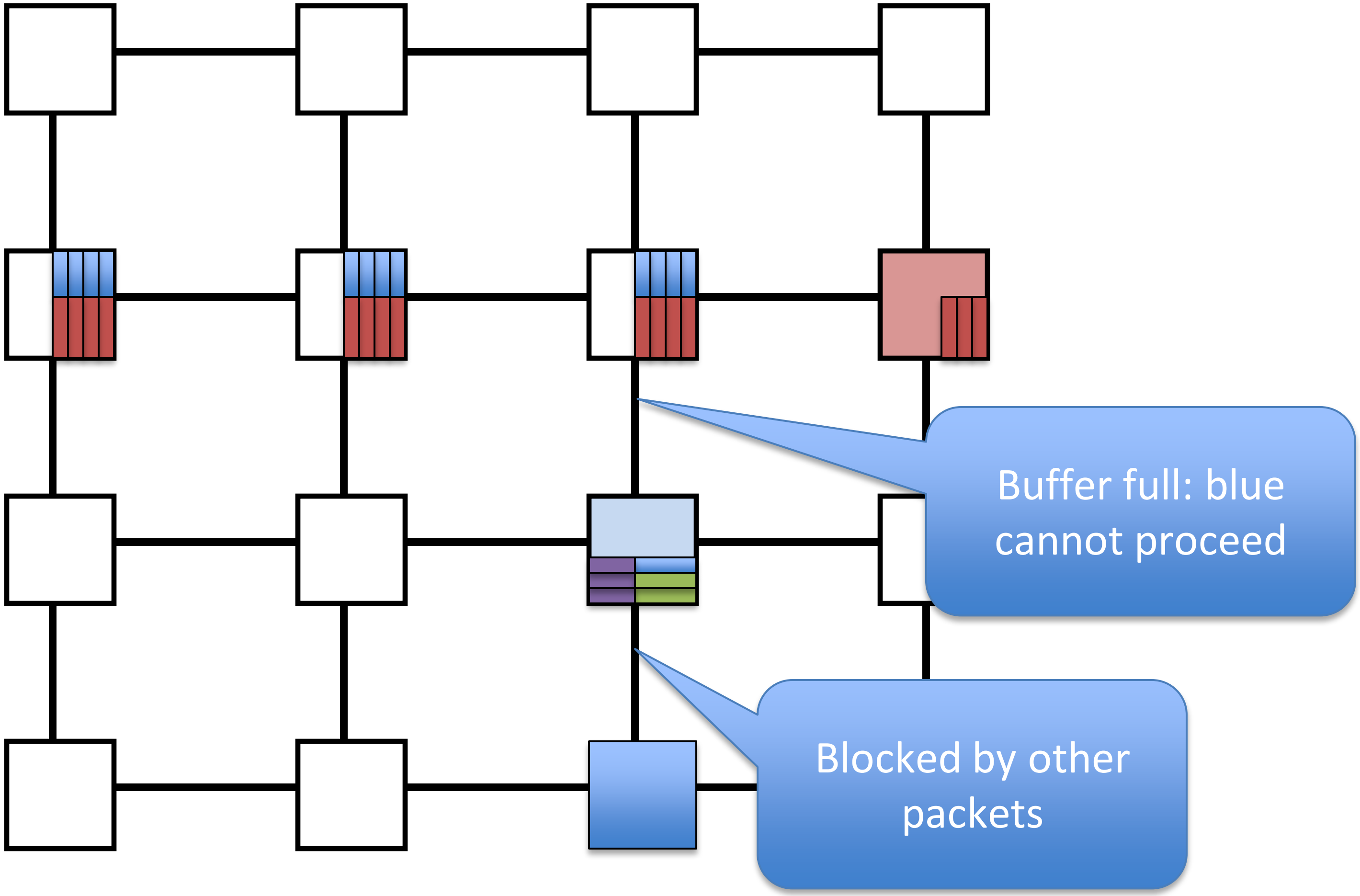


(A) 16-Flit FIFO Buffers

(B) 16-Flit Four-Lane Buffer

Figure 5: (A) Conventional nodes organize their buffers into FIFO queues restricting routing. (B) A network using virtual-channel flow control organizes its buffers into several independent lanes.

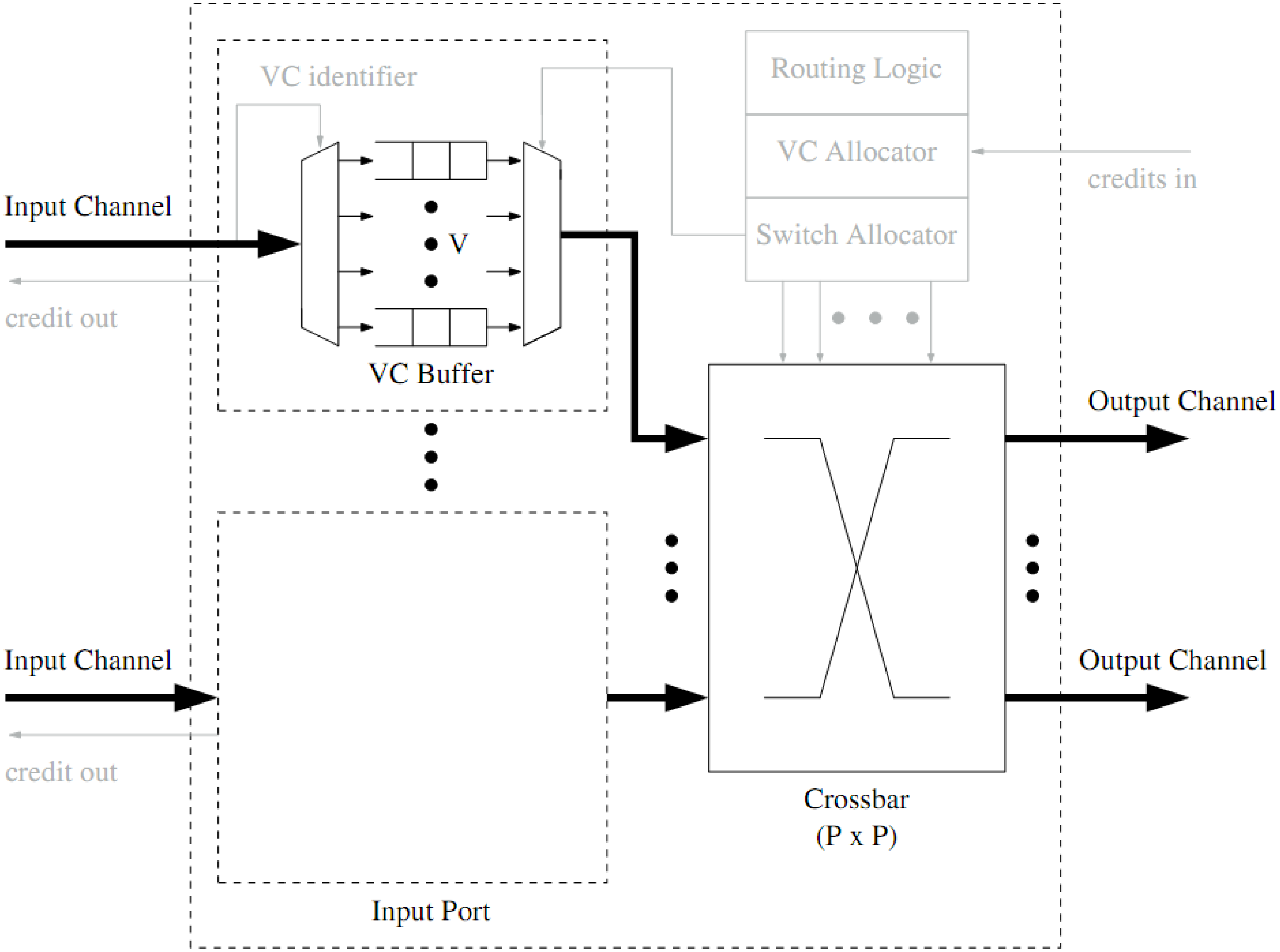
Virtual Channel Flow Control



Buffer full: blue cannot proceed

Blocked by other packets

A Modern Virtual Channel Based Router



Other Uses of Virtual Channels

- Deadlock avoidance
  - Enforcing switching to a different set of virtual channels on some “turns” can break the cyclic dependency of resources
    - Enforce order on VCs
  - Escape VCs: Have at least one VC that uses deadlock-free routing. Ensure each flit has fair access to that VC.
  - Protocol level deadlock: Ensure address and data packets use different VCs → prevent cycles due to intermixing of different packet classes

- Prioritization of traffic classes
  - Some virtual channels can have higher priority than others

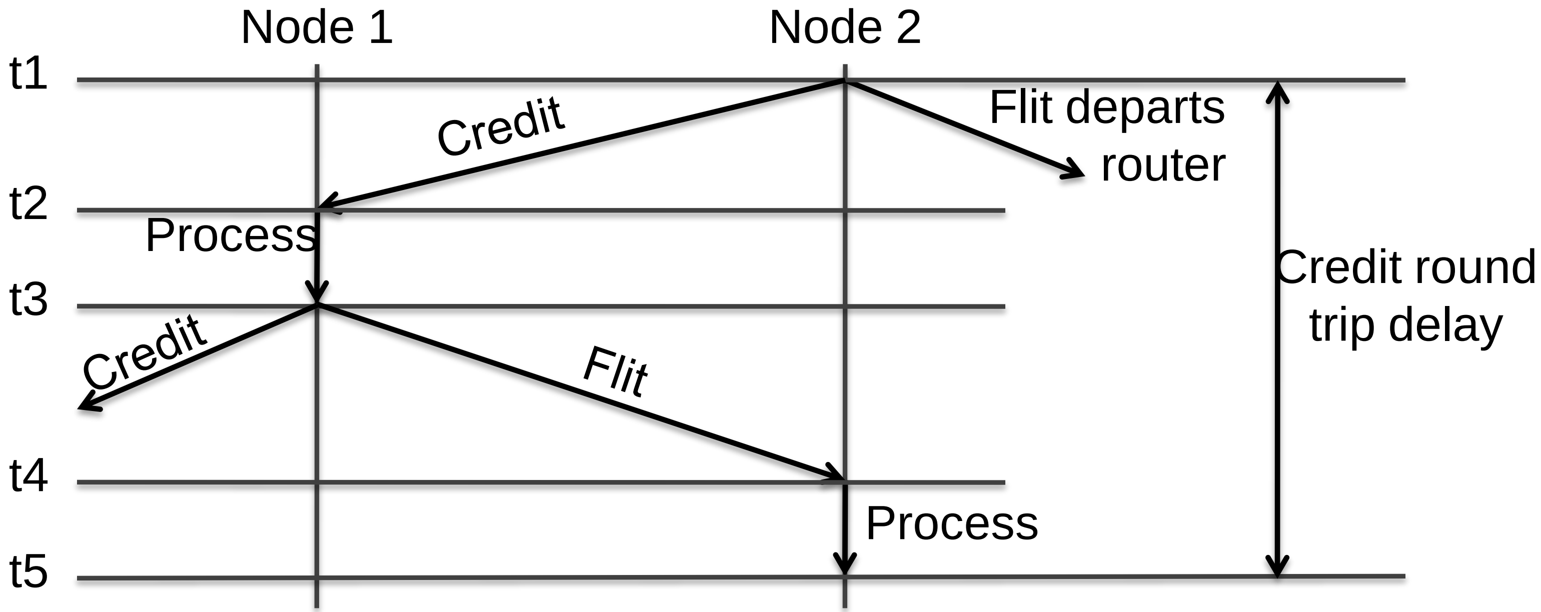
Communicating Buffer Availability

- Credit-based flow control
  - Upstream knows how many buffers are downstream
  - Downstream passes back credits to upstream
  - Significant upstream signaling (esp. for small flits)

- On/Off (XON/XOFF) flow control
  - Downstream has on/off signal to upstream

- Ack/Nack flow control
  - Upstream optimistically sends downstream
  - Buffer cannot be deallocated until ACK/NACK received
  - Inefficiently utilizes buffer space

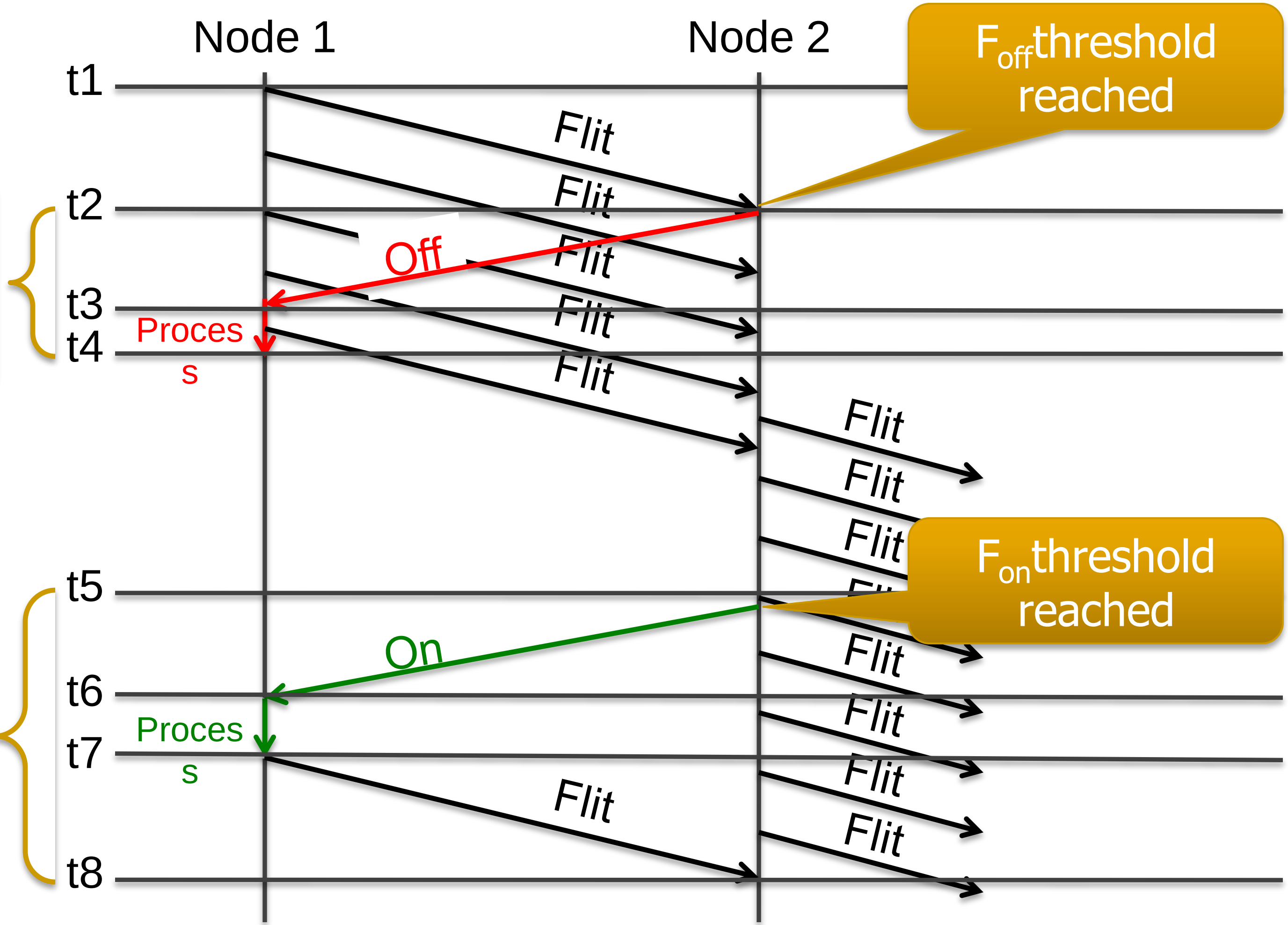
Credit-based Flow Control



- Round-trip credit delay:
  - Time between when buffer empties and when next flit can be processed from that buffer entry
- Significant throughput degradation if there are few buffers
- Important to size buffers to tolerate credit turn-around

On/Off (XON/XOFF) Flow Control

- Downstream has on/off signal to upstream



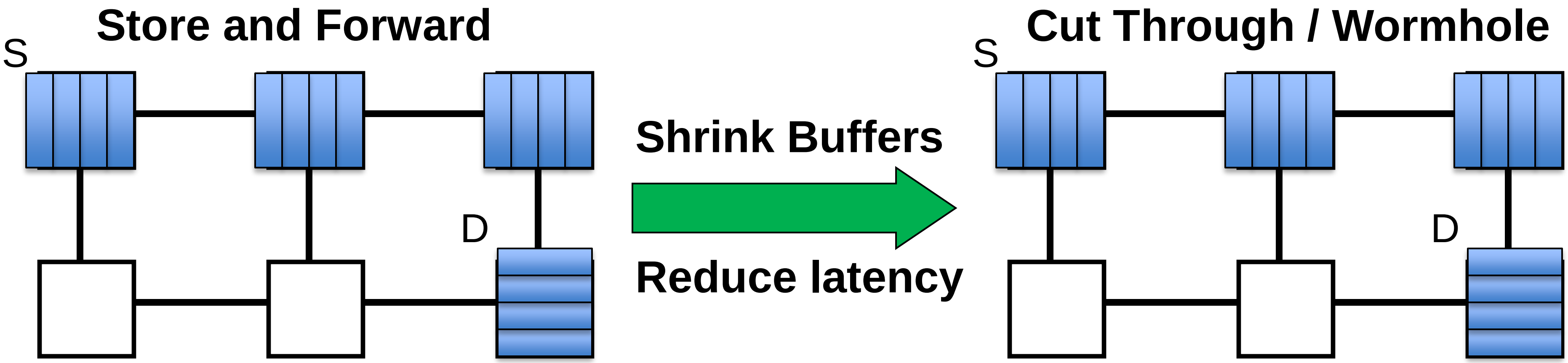
F_{off} threshold reached

F_{off} set to prevent flits arriving before t4 from overflowing

F_{on} threshold reached

F_{on} set so that Node 2 does not run out of flits between t5 and t8

Review: Flow Control



Store and Forward

Shrink Buffers

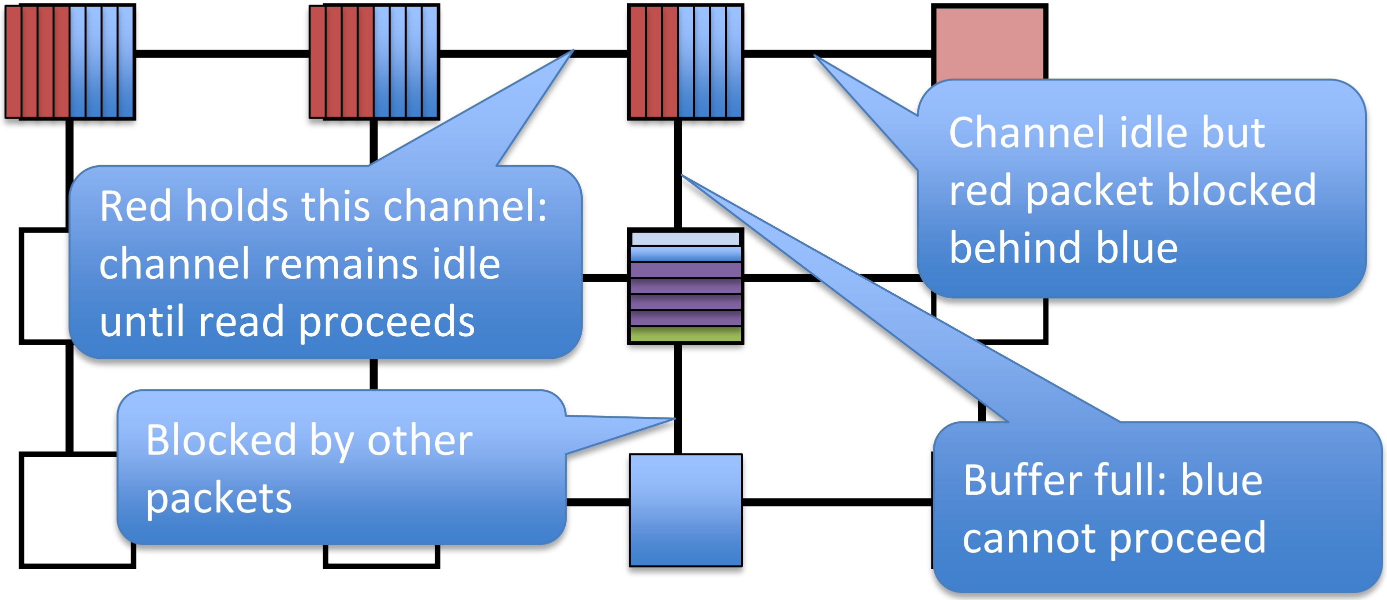
Reduce latency

Cut Through / Wormhole

Any other issues?

Head-of-Line Blocking

Use Virtual Channels



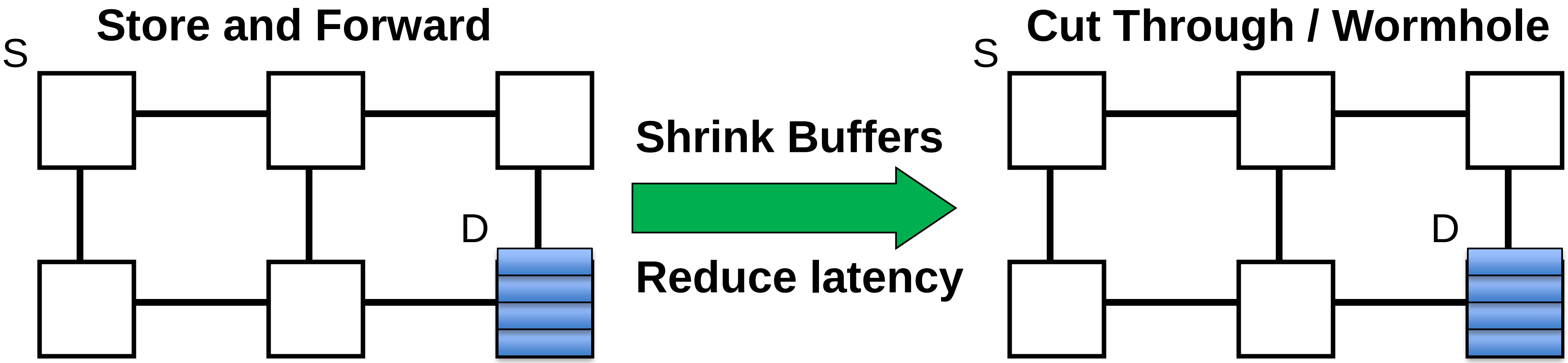
Red holds this channel: channel remains idle until read proceeds

Channel idle but red packet blocked behind blue

Blocked by other packets

Buffer full: blue cannot proceed

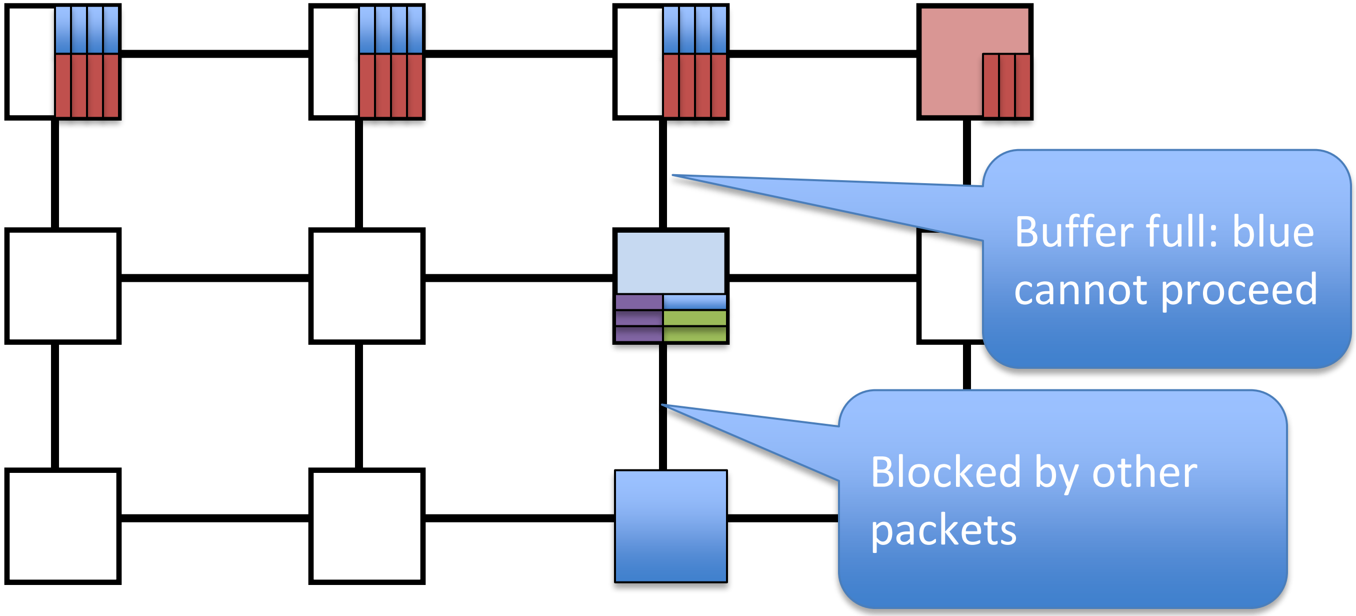
Review: Flow Control



Any other issues?

Head-of-Line Blocking

Use Virtual Channels


Buffer full: blue cannot proceed

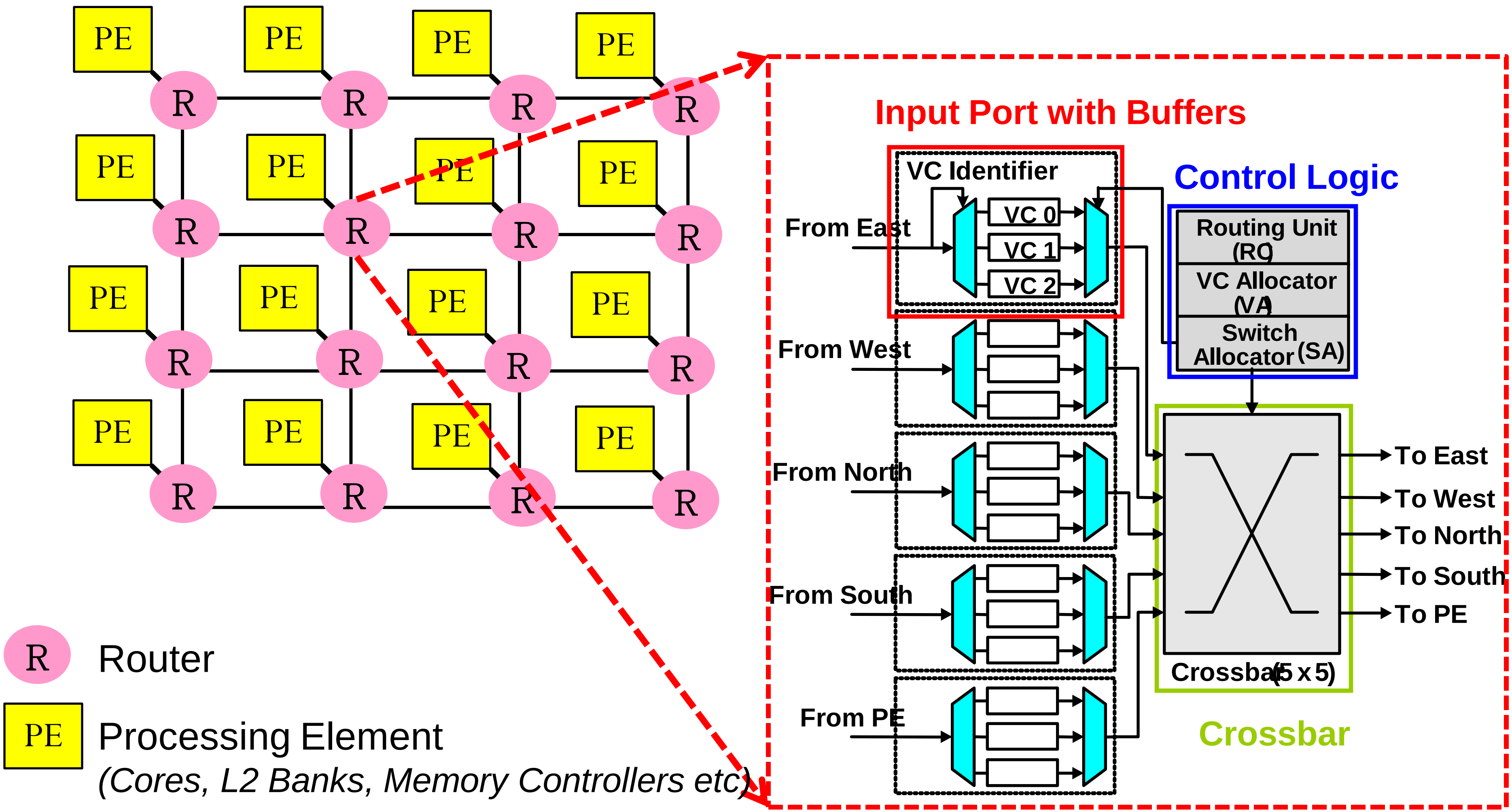
Blocked by other packets

Agenda

- Terminology review
- **More on interconnection networks**
  - Network properties and performance metrics
  - Buffering and flow control
  - **Router design and pipeline options**

- Research on NoCs
  - The problem with packet scheduling
  - Application-aware packet scheduling
  - Aergia: Latency slack-based packet scheduling
  - Bufferless networks

On-chip Networks



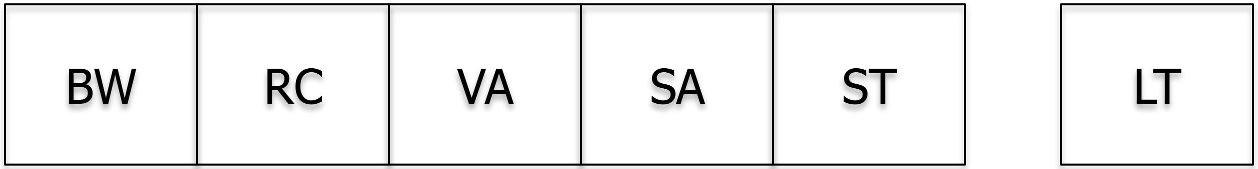
R Router

PE Processing Element
(Cores, L2 Banks, Memory Controllers etc)

Router Design: Functions of a Router

- Buffering (of flits)
- Route computation
- Arbitration of flits (i.e. prioritization) when contention
  - Called packet scheduling
- Switching
  - From input port to output port
- Power management
  - Scale link/router frequency

Router Pipeline



BW RC VA SA ST LT

- Five logical stages
  - BW: Buffer Write
  - RC: Route computation
  - VA: Virtual Channel Allocation
  - SA: Switch Allocation
  - ST: Switch Traversal

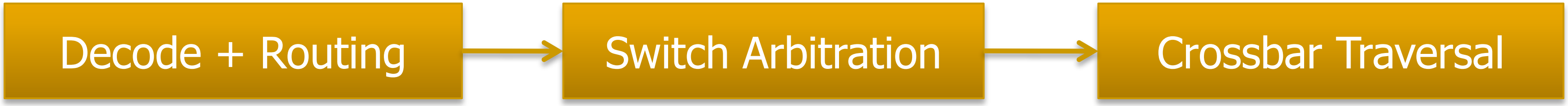
  - LT: Link Traversal

Wormhole Router Timeline

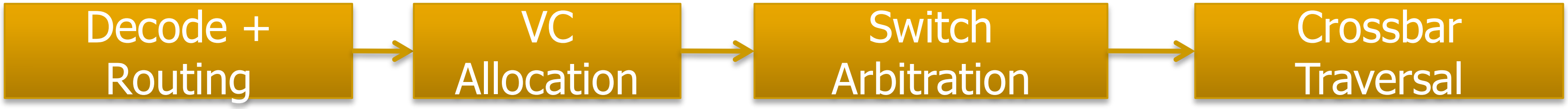
Head	BW	RC	VA	SA	ST	LT			
Body 1		BW	☒	☒	SA	ST	LT		
Body 2			BW	☒	☒	SA	ST	LT	
Tail				BW	☒	☒	SA	ST	LT

- Route computation performed once per packet
  - Virtual channel allocated once per packet
- Body and tail flits inherit this information from head flit

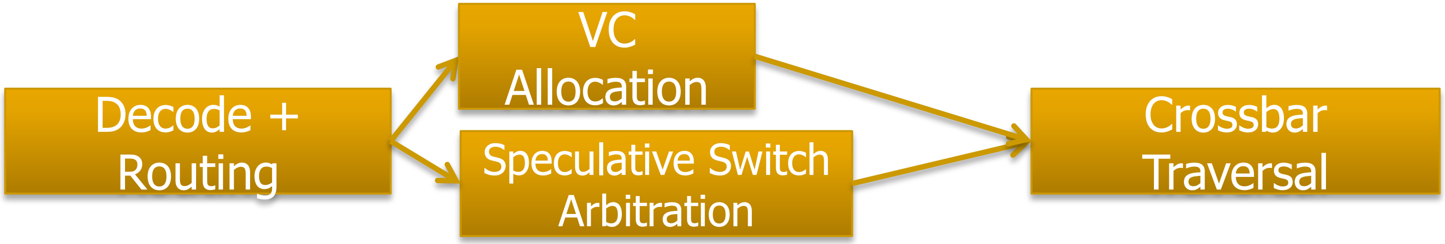
Dependencies in a Router


Decode + Routing
Switch Arbitration
Crossbar Traversal

Wormhole Router


Decode + Routing
VC Allocation
Switch Arbitration
Crossbar Traversal

Virtual Channel Router


Decode + Routing
VC Allocation
Speculative Switch Arbitration
Crossbar Traversal

Speculative Virtual Channel Router

- Dependence between output of one module and input of another
  - Determine critical path through router
  - Cannot bid for switch port until routing performed

Pipeline Optimizations: Lookahead Routing

- At current router perform routing computation for next router
  - Overlap with BW

BW RC	VA	SA	ST	LT

  - Precomputing route allows flits to compete for VCs immediately after BW
  - RC decodes route header
  - Routing computation needed at next hop
    - Can be computed in parallel with VA

- Galles, "Spider: A High-Speed Network Interconnect," IEEE Micro 1997.

Pipeline Optimizations: Speculation

- Assume that Virtual Channel Allocation stage will be successful
  - Valid under low to moderate loads
- Entire VA and SA in parallel

BW RC	VA SA	ST	LT

- If VA unsuccessful (no virtual channel returned)
  - Must repeat VA/SA in next cycle
- Prioritize non-speculative requests

Pipeline Optimizations: Bypassing

- When no flits in input buffer
  - Speculatively enter ST
  - On port conflict, speculation aborted

VA RC Setup	ST	LT

  - In the first stage, a free VC is allocated, next routing is performed and the crossbar is setup

Agenda

- Terminology review
- More on interconnection networks
  - Network properties and performance metrics
  - Buffering and flow control
  - Router design and pipeline options

- **Research on NoCs**
  - **The problem with packet scheduling**
  - Application-aware packet scheduling
  - Aergia: Latency slack-based packet scheduling
  - Bufferless networks

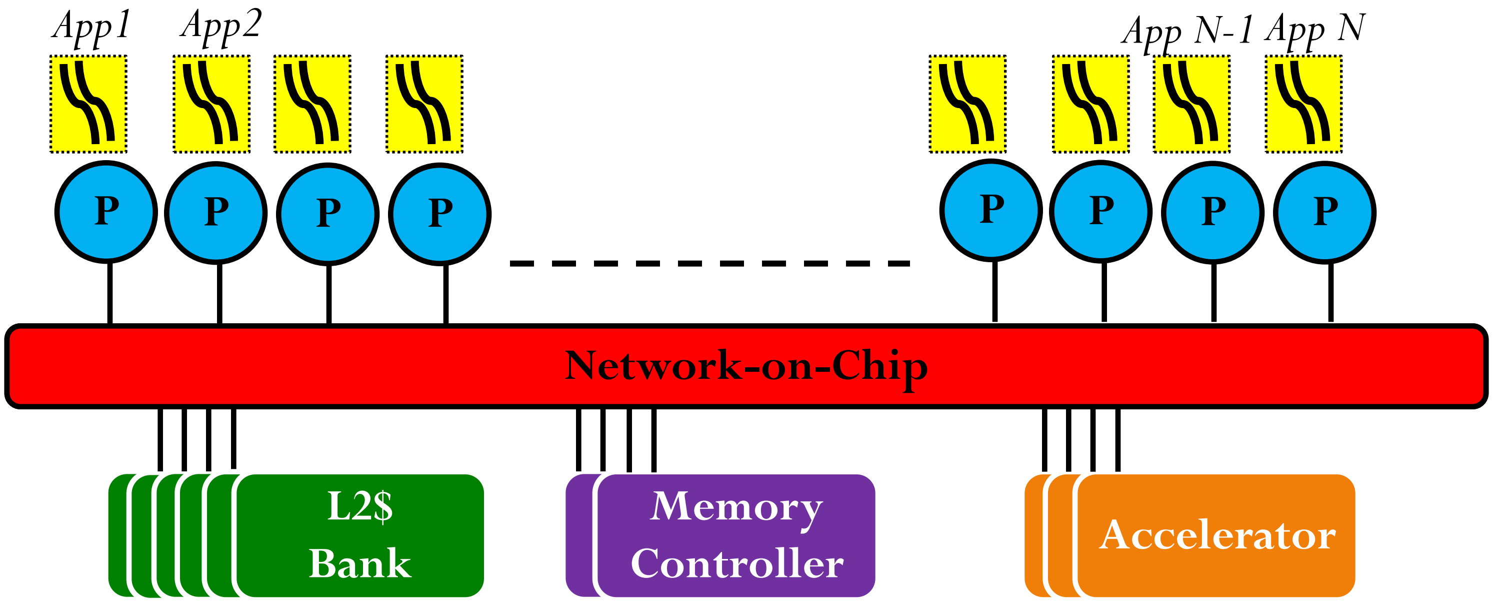
Packet Scheduling

- 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

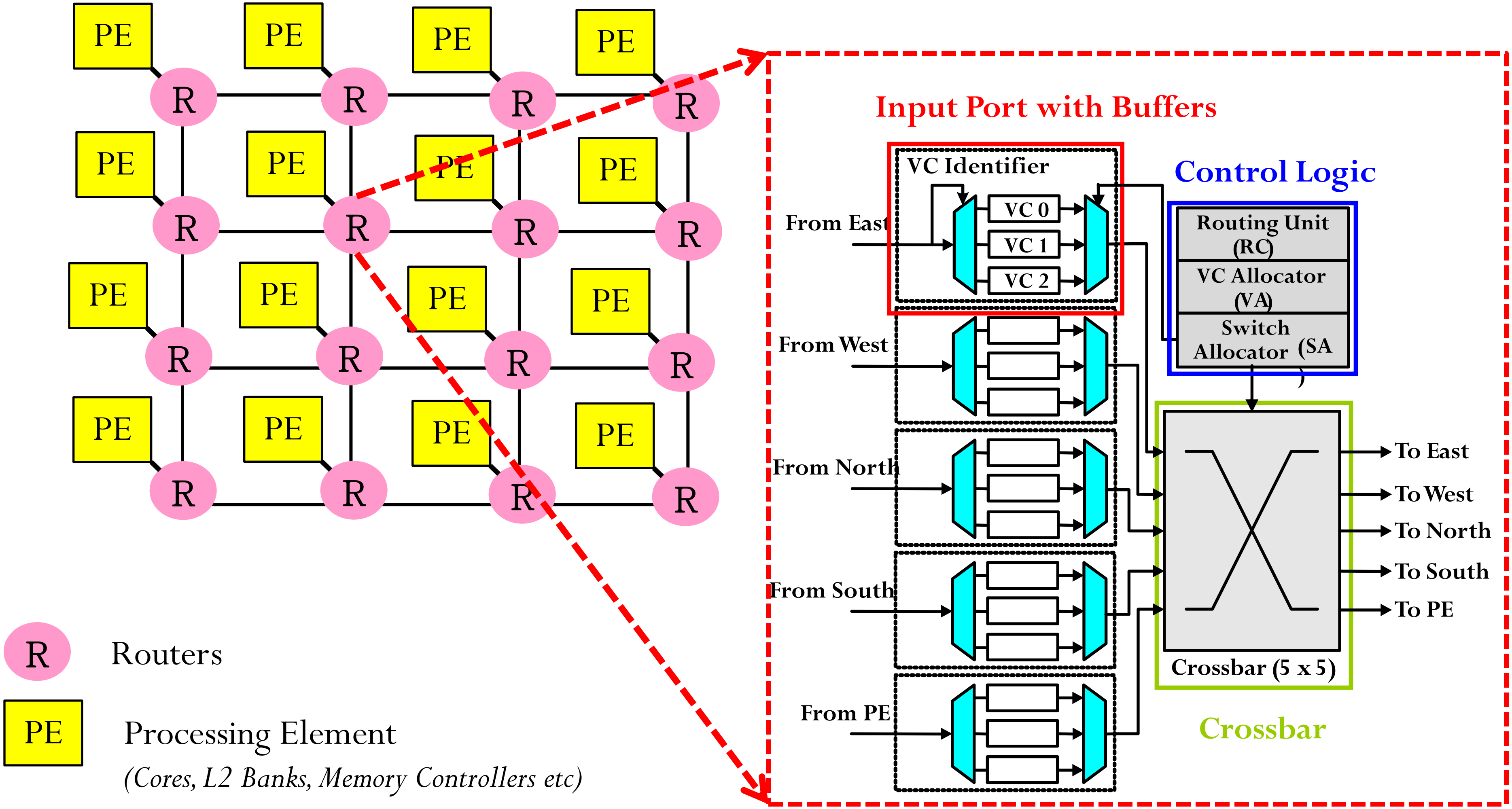
- Better policies in a multi-core environment
  - Use application characteristics

The Problem: Packet Scheduling



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

The Problem: Packet Scheduling

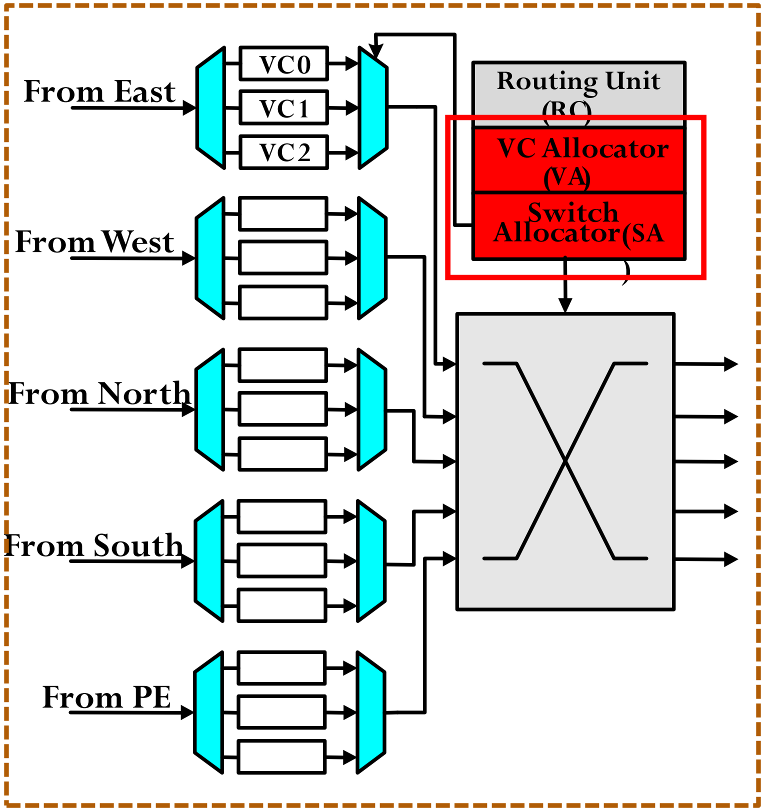


R Routers

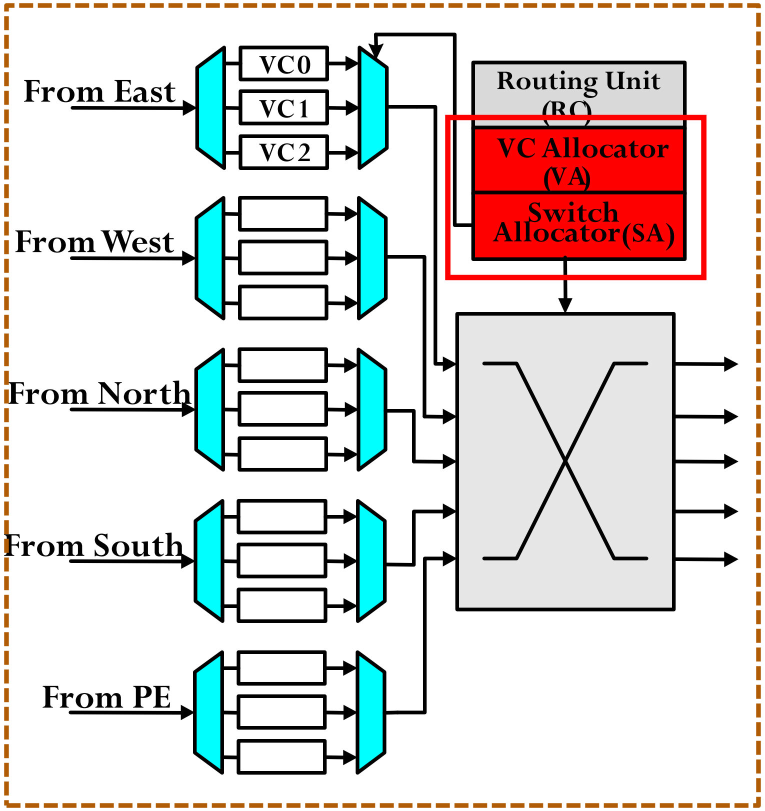
PE Processing Element

(Cores, L2 Banks, Memory Controllers etc)

The Problem: Packet Scheduling

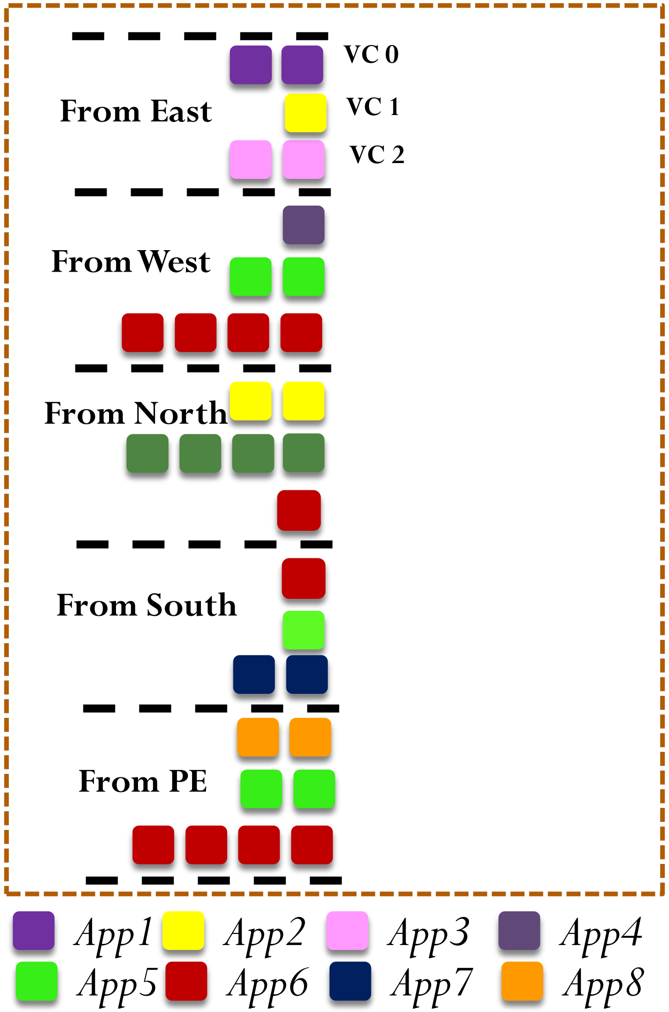


The Problem: Packet Scheduling

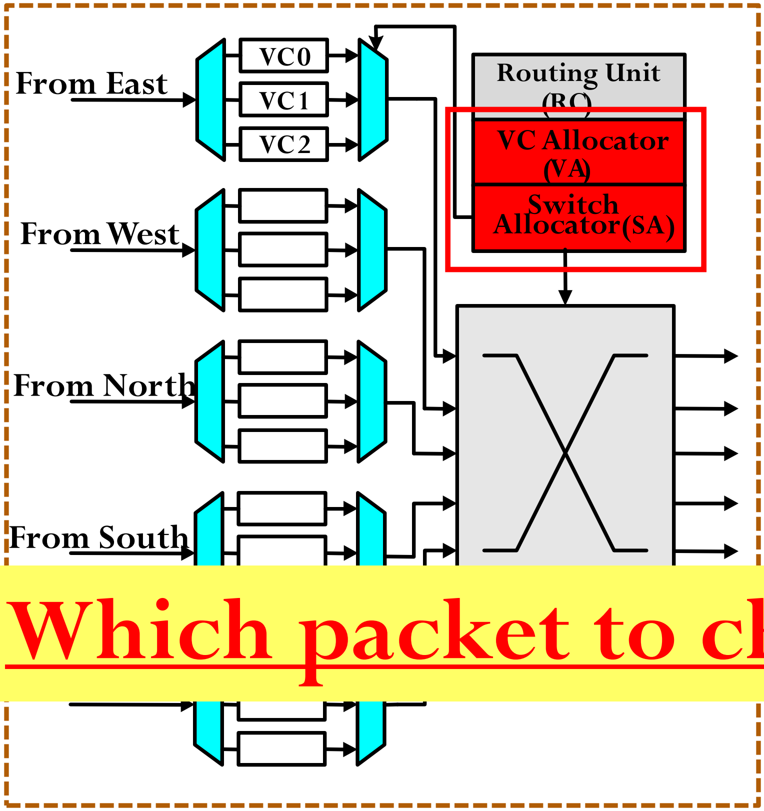


Conceptual

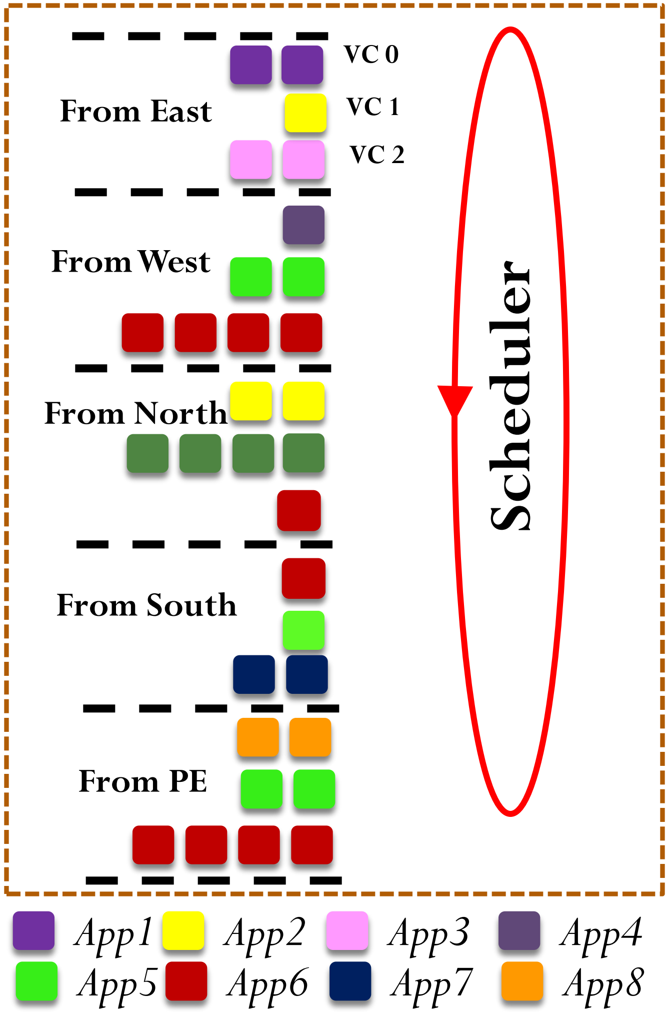
View



The Problem: Packet Scheduling



Conceptual View



Which packet to choose?

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.

Agenda

- Terminology review
- More on interconnection networks
  - Network properties and performance metrics
  - Buffering and flow control
  - Router design and pipeline options

- **Research on NoCs**
  - The problem with packet scheduling
  - **Application-aware packet scheduling**
  - Aergia: Latency slack-based packet scheduling
  - Bufferless networks

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

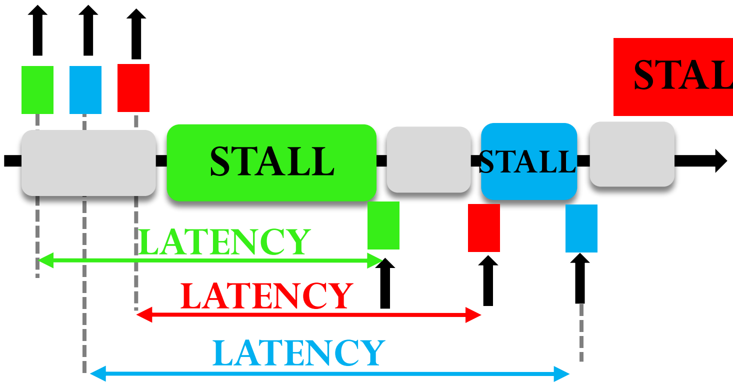
- Application's Stall Time Criticality (STC) can be measured by its average network stall time per packet (i.e. NST/packet)
  - Network Stall Time (NST) is number of cycles the processor stalls waiting for network transactions to complete

Motivation: Stall Time Criticality

- Why 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}

⬜ Compute

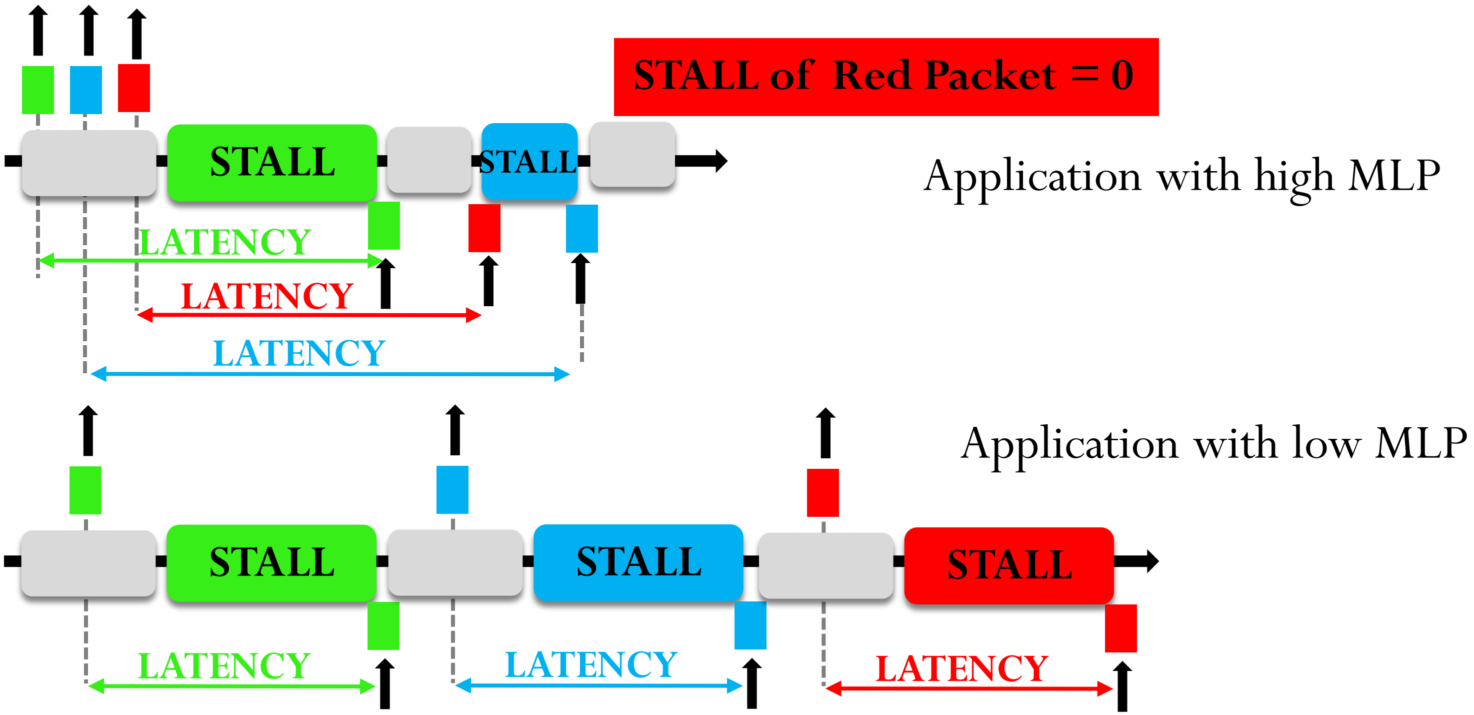


STALL of Red Packet = 0

Application with high MLP

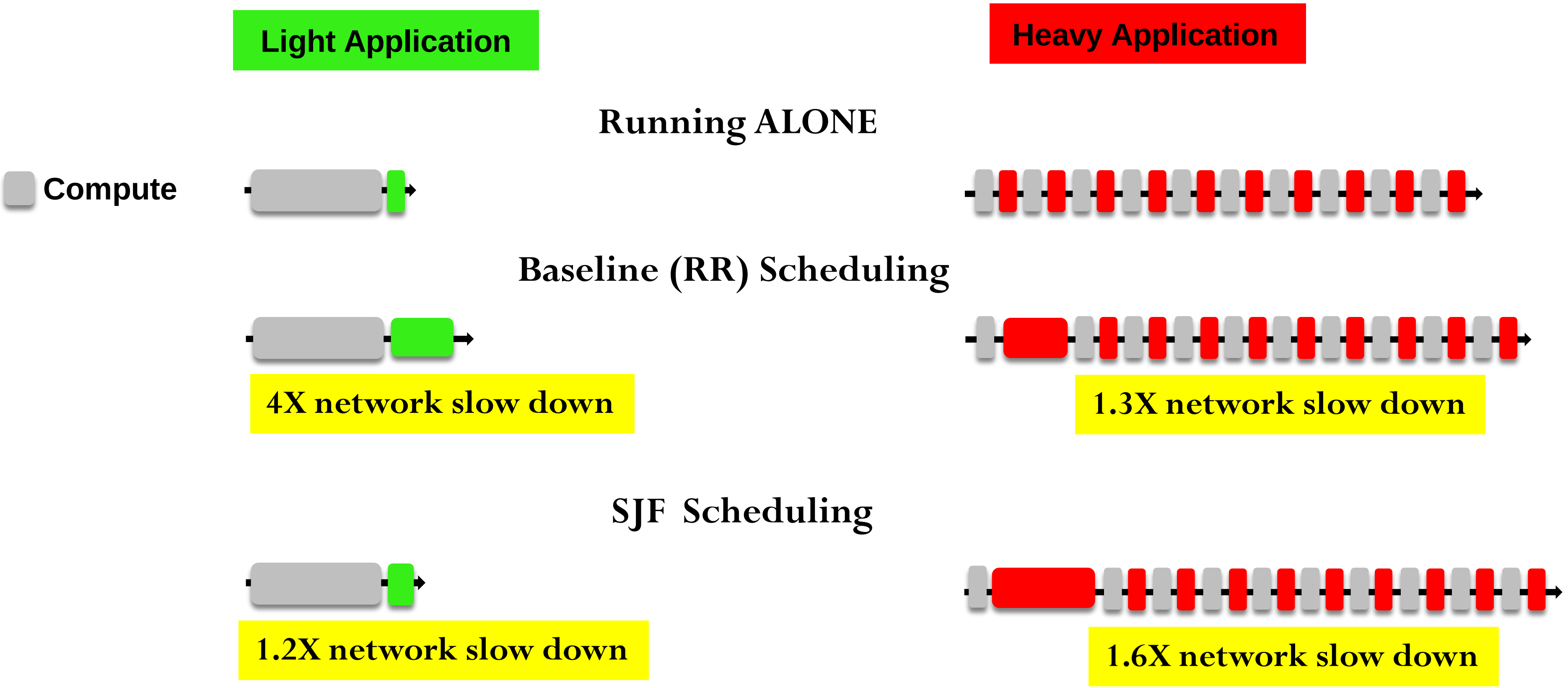
- Observation 1: **Packet Latency != Network Stall Time**

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

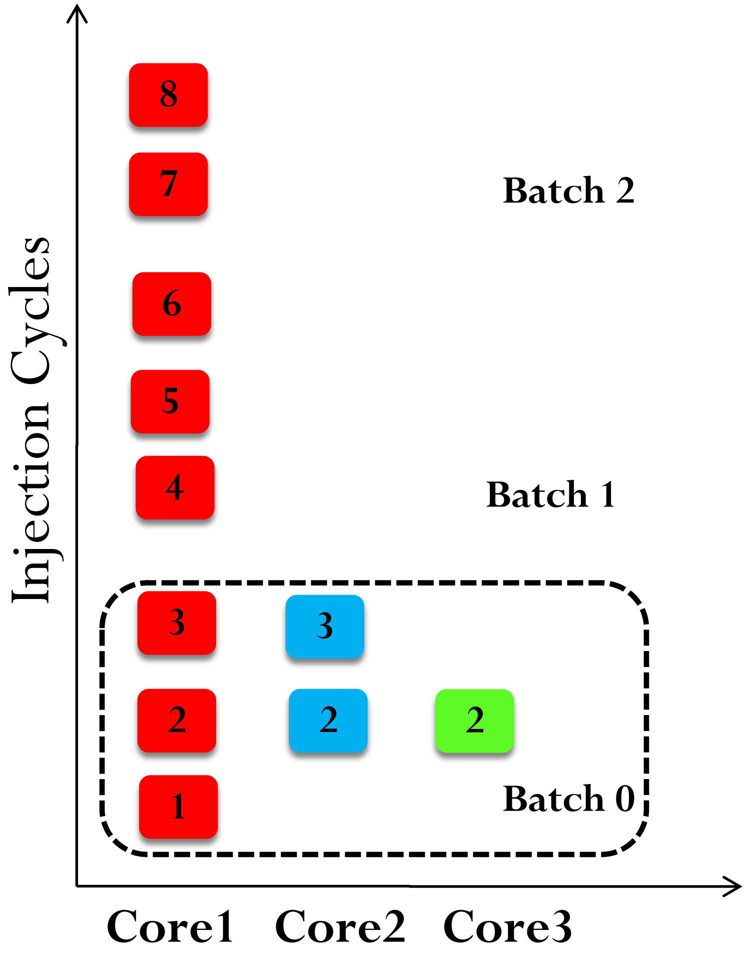
Component 2: Batching

- 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 same across all routers

STC Scheduling Example

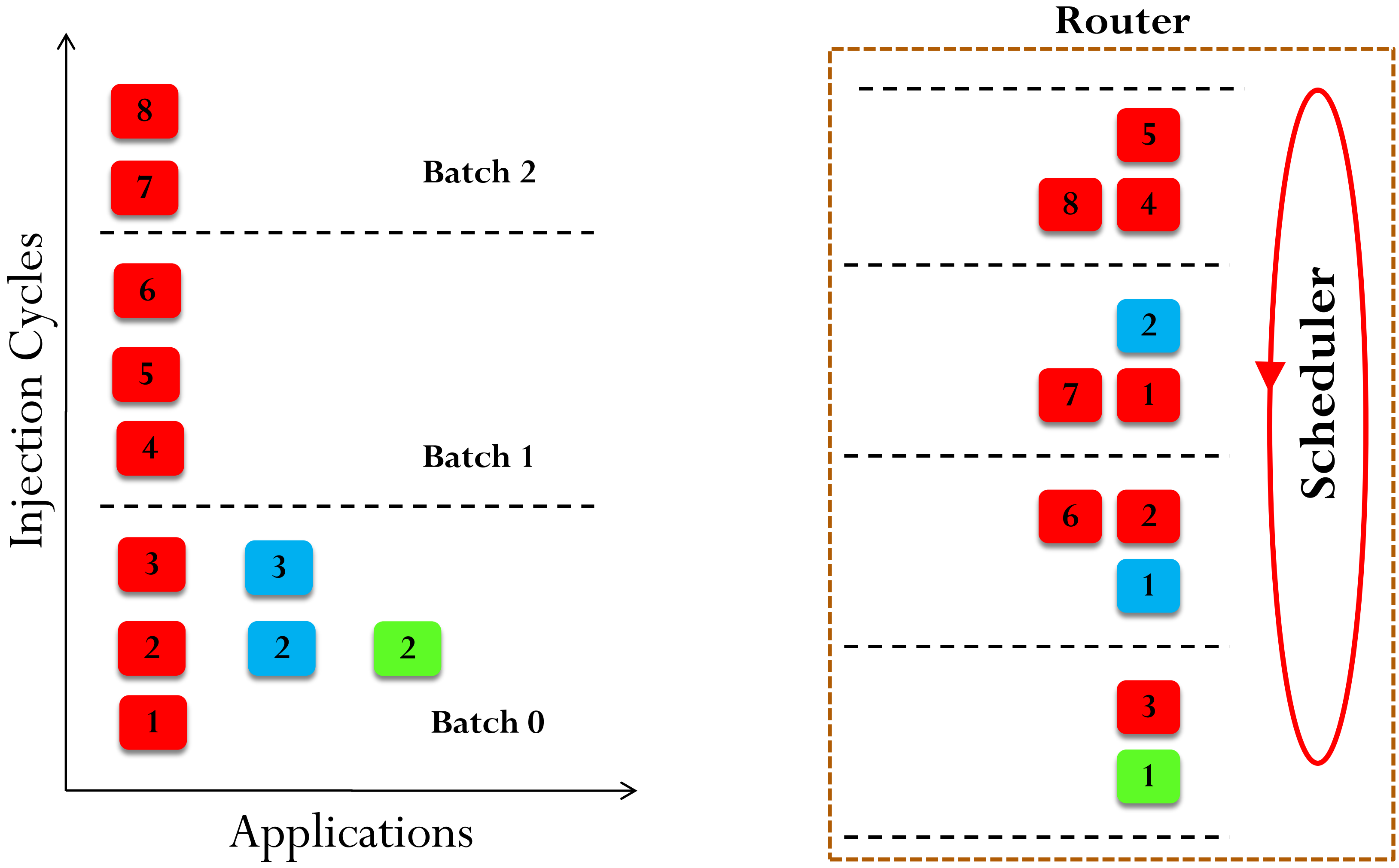


Packet Injection Order at Processor

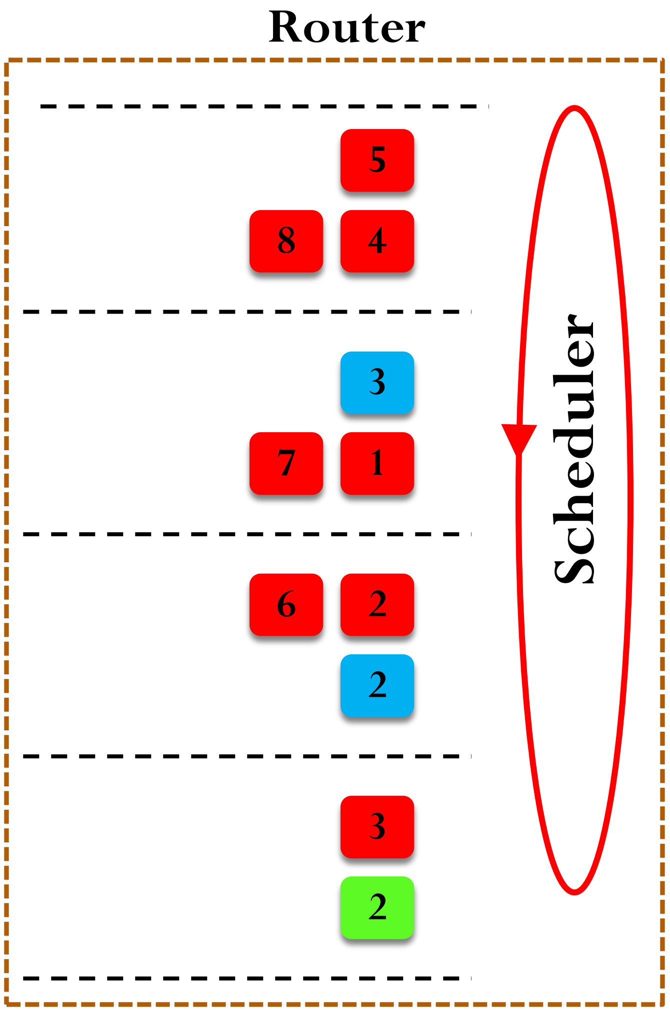
Batching interval length = 3 cycles

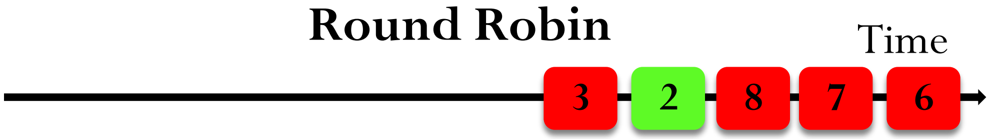
Ranking order = 🟩 > 🟦 > 🟥

STC Scheduling Example



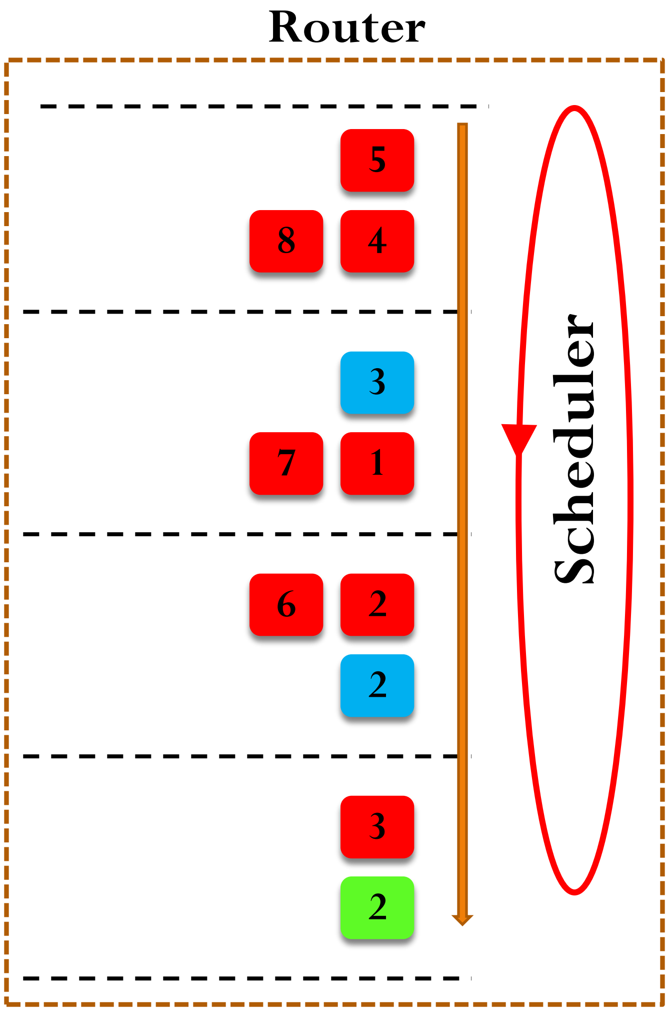
STC Scheduling Example

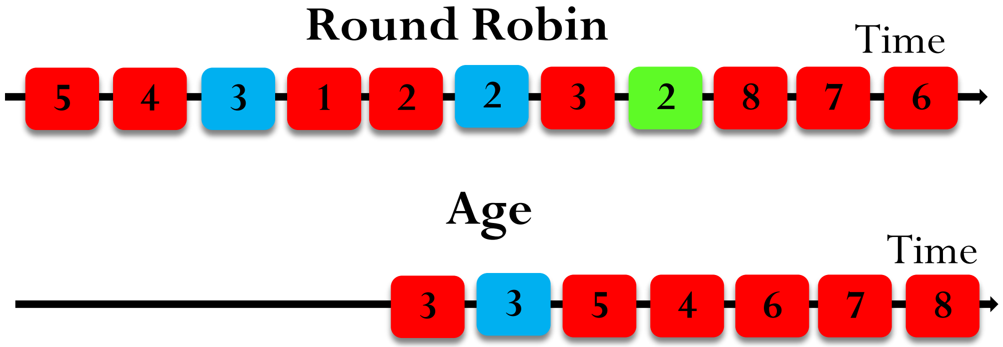




STALL CYCLES				Avg
RR	8	6	11	8.3
Age				
STC				

STC Scheduling Example

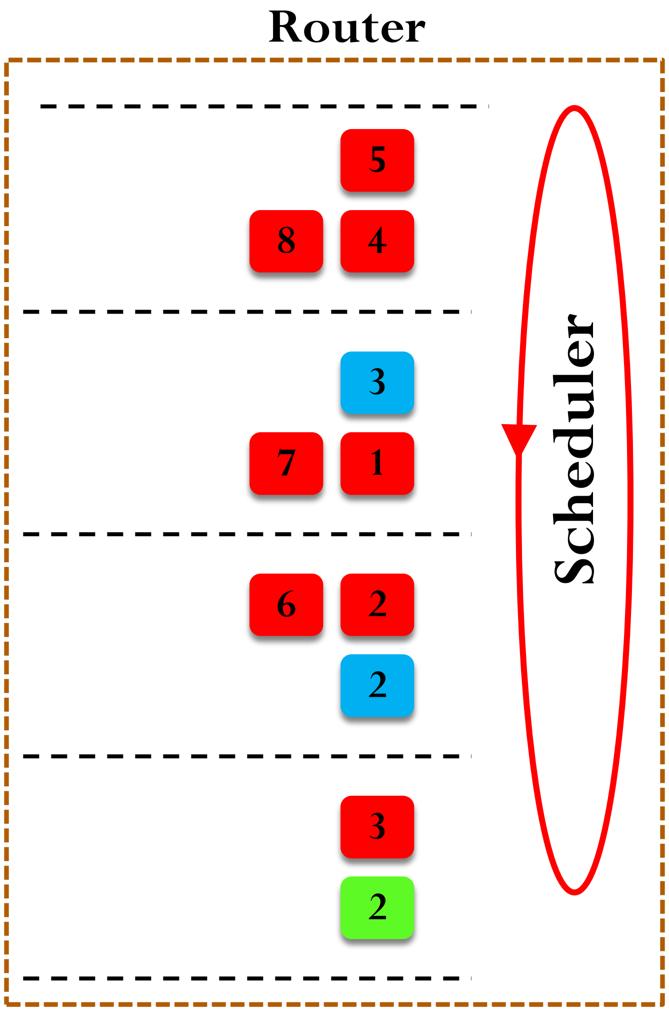


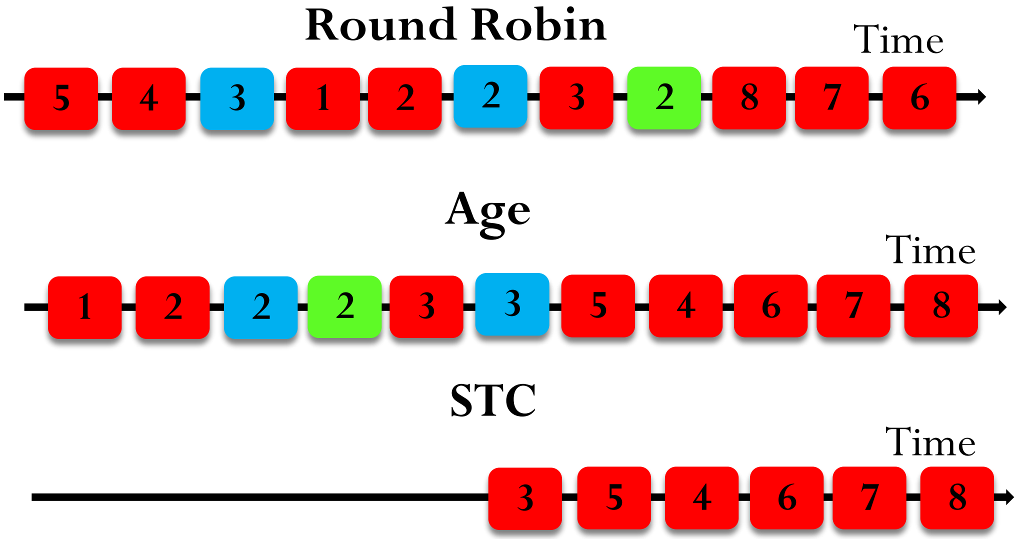


STALL CYCLES				Avg
RR	8	6	11	8.3
Age	4	6	11	7.0
STC				

Ranking order 🟩 > 🟦 > 🟥

STC Scheduling Example





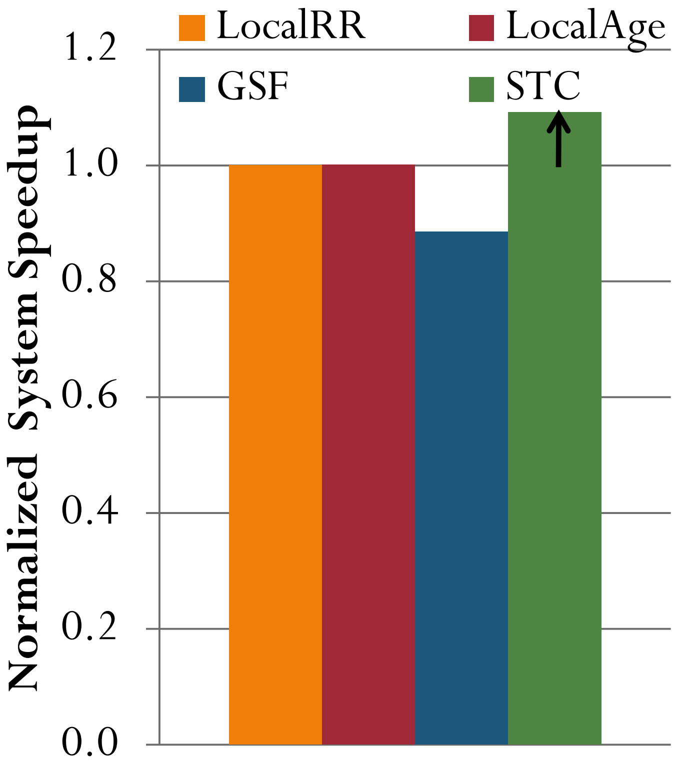
	STALL CYCLES			Avg
RR	8	6	11	8.3
Age	4	6	11	7.0
STC	1	3	11	5.0

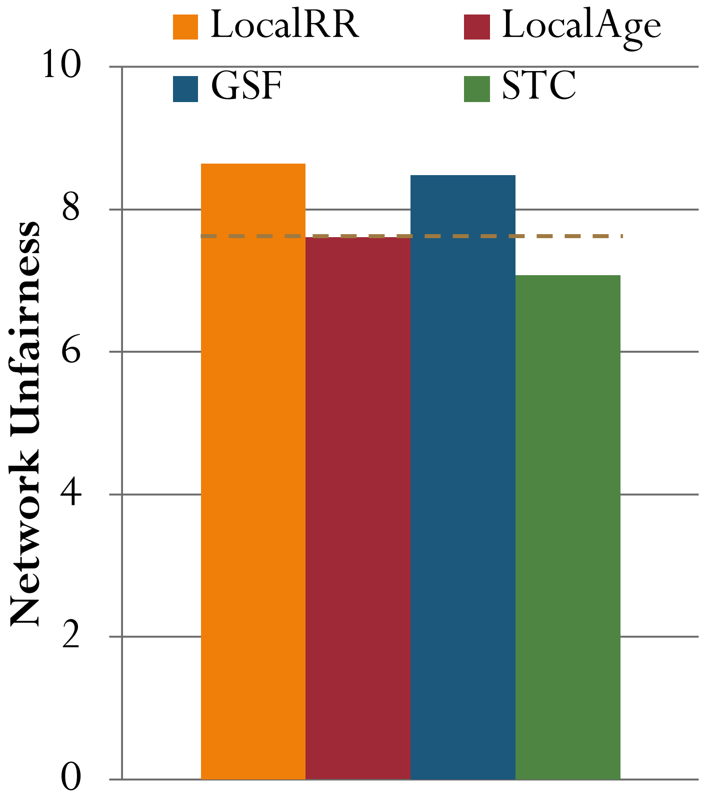
Qualitative Comparison

- **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





Agenda

- Terminology review
- More on interconnection networks
  - Network properties and performance metrics
  - Buffering and flow control
  - Router design and pipeline options
- **Research on NoCs**
  - The problem with packet scheduling
  - Application-aware packet scheduling
  - **Aergia: Latency slack-based packet scheduling**
  - Bufferless networks

Today

- Review (Topology & Flow Control)
- More on interconnection networks
  - Routing
  - Router design
  - Network performance metrics
  - On-chip vs. off-chip differences

- **Research on NoCs and packet scheduling**
  - The problem with packet scheduling
  - Application-aware packet scheduling
  - **Aergia: Latency slack based packet scheduling**

What is Aérgia?

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

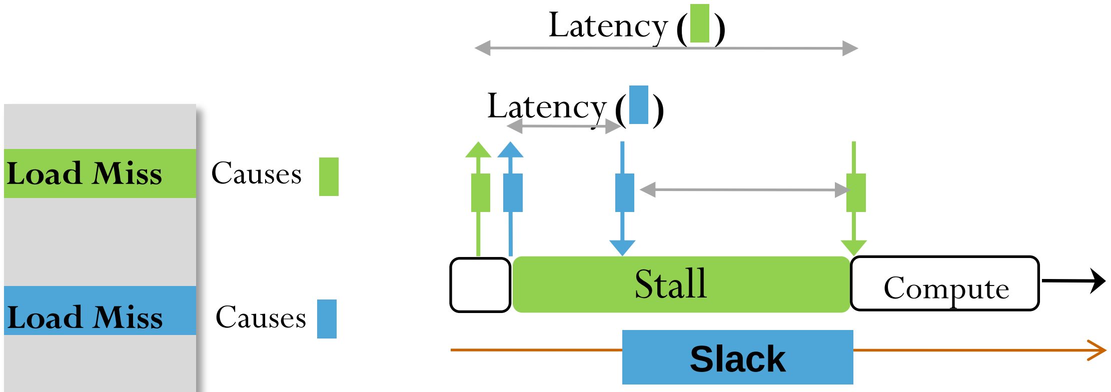
Slack of Packets

- What is slack of a packet?
  - Slack of a packet is number of cycles it can be delayed in a router without 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

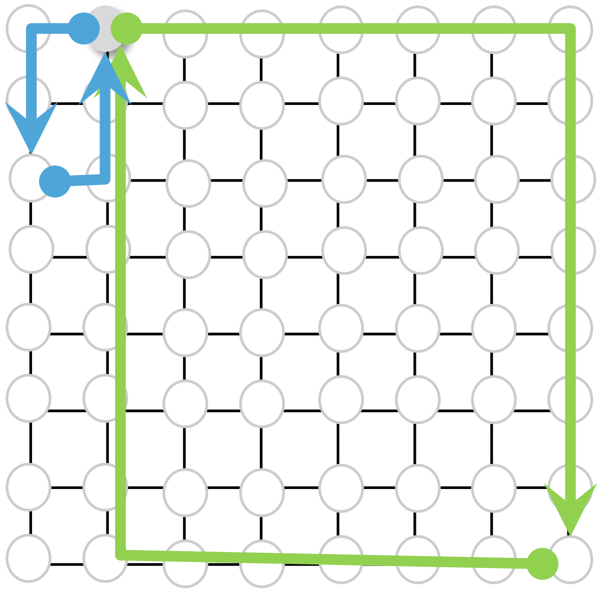
Instruction Window

Execution Time



🟦 **returns earlier than necessary**

Network-on-Chip



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

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

Prioritizing using Slack

Core A

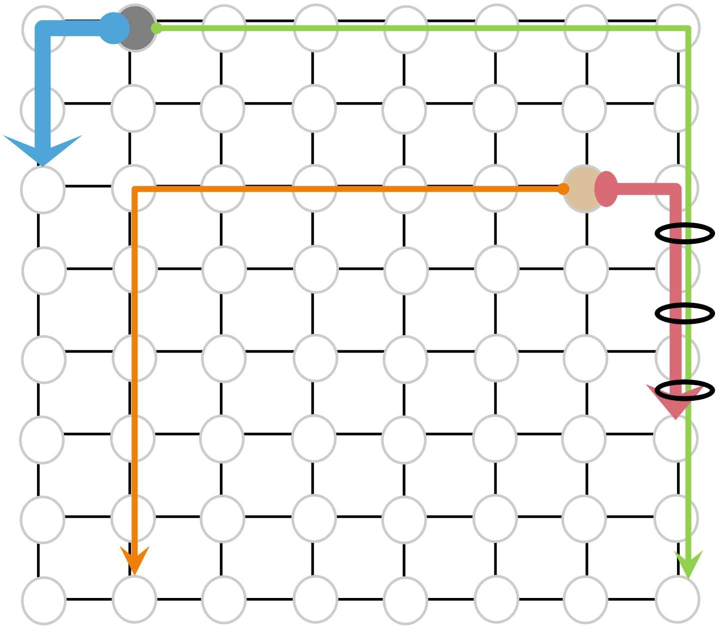
Load Miss **Causes** 🟩

Load Miss **Causes** 🟦

Core B

Load Miss **Causes** 🟧

Load Miss **Causes** 🟥



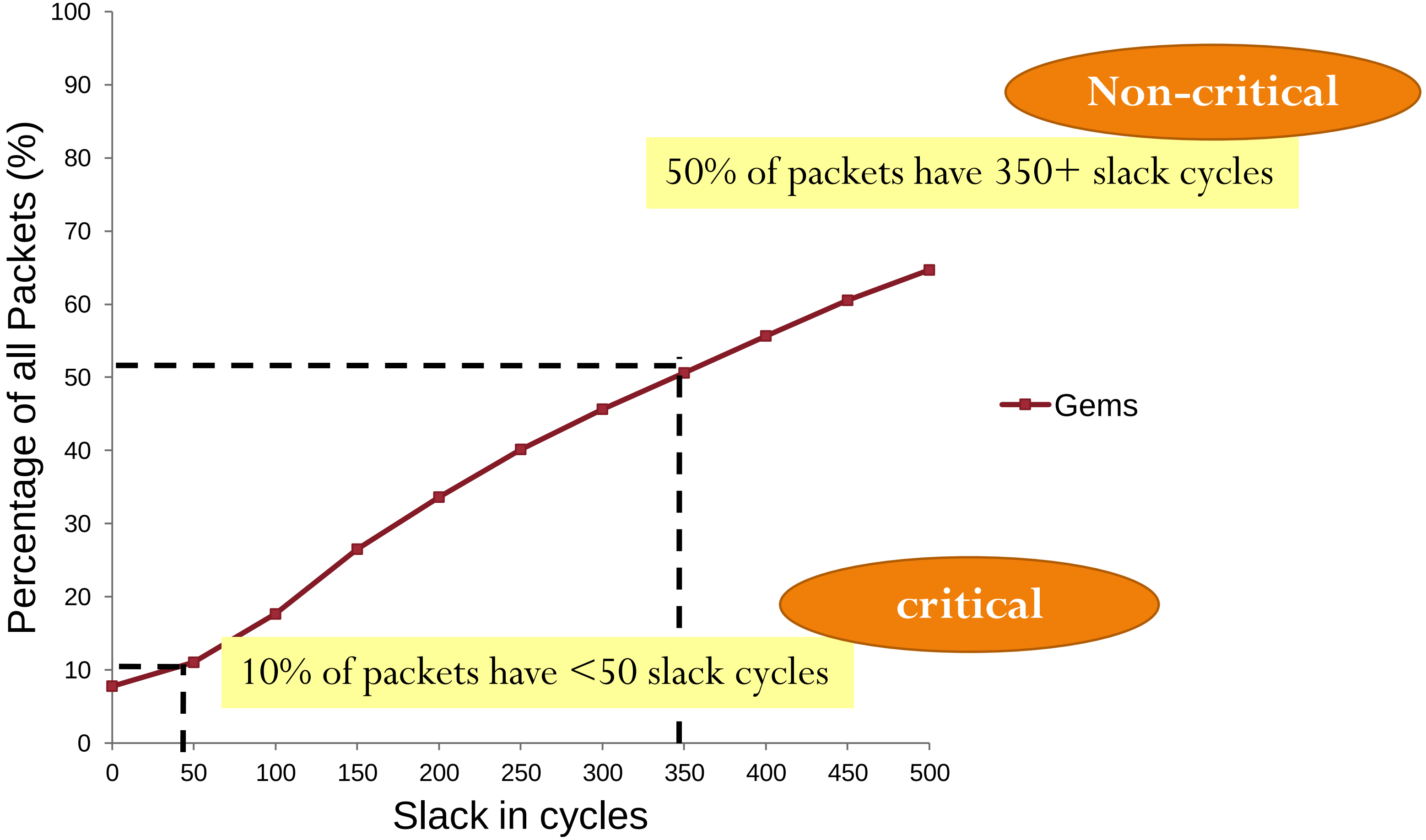
Packet	**Latency**	**Slack**
🟩	13 hops	0 hops
🟦	3 hops	10 hops

⬭ Interference at 3 hops

Slack(🟥) > Slack (🟩)

Prioritize 🟩

Slack in Applications



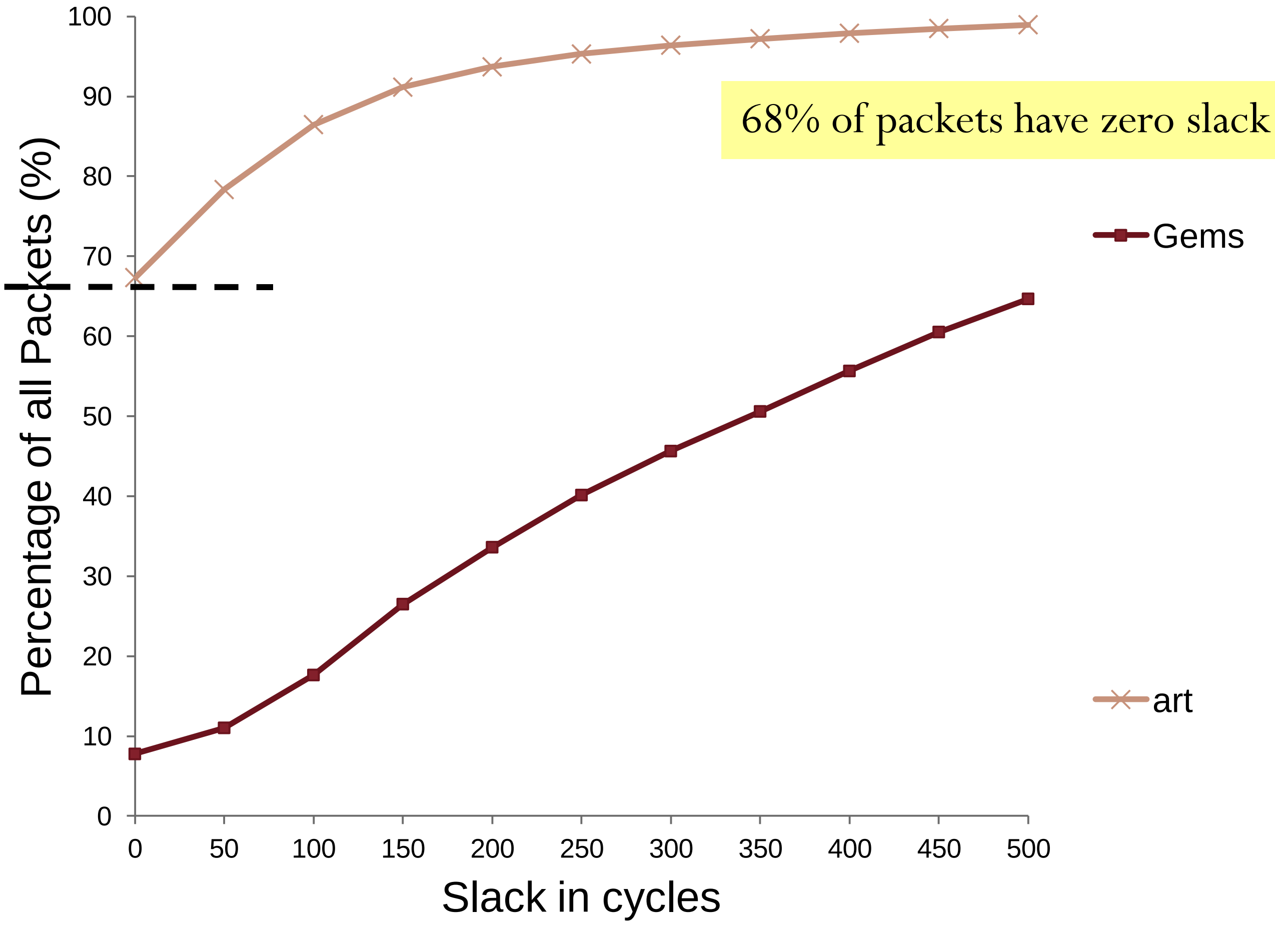
Non-critical

50% of packets have 350+ slack cycles

critical

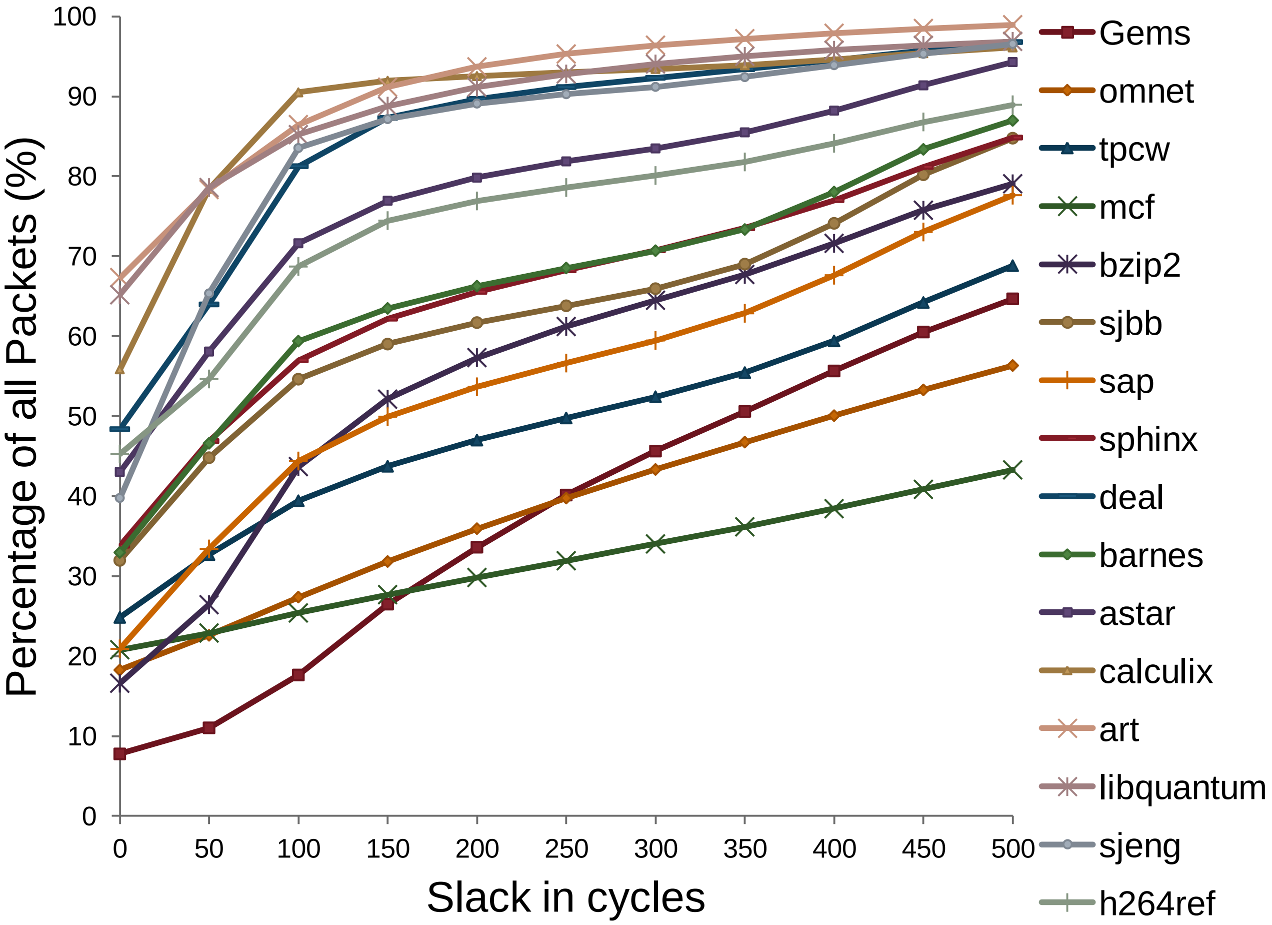
10% of packets have <50 slack cycles

Slack in Applications

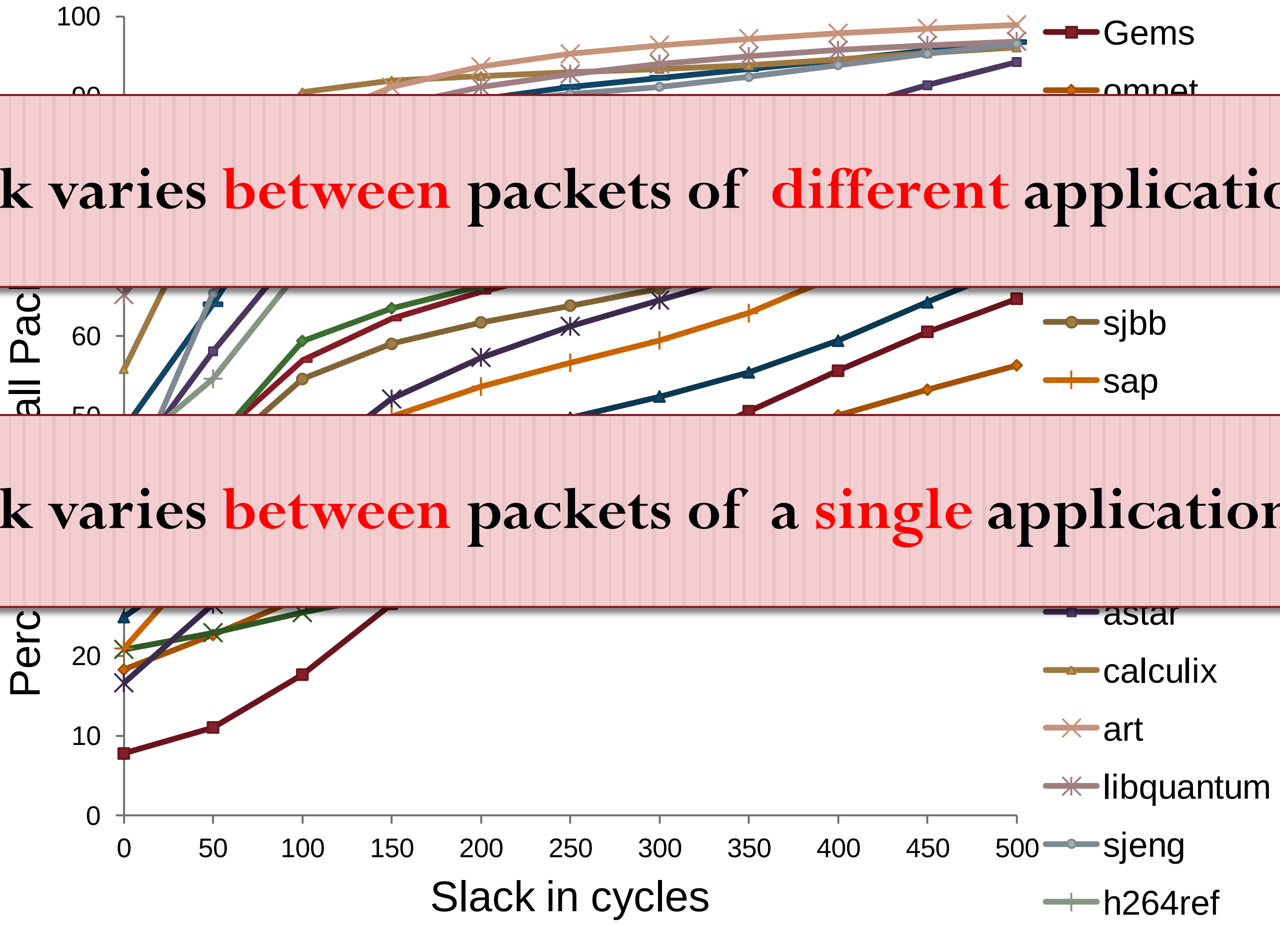


68% of packets have zero slack cycles

Diversity in Slack



Diversity in Slack



Slack varies between packets of different applications

Slack varies between packets of a single application

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

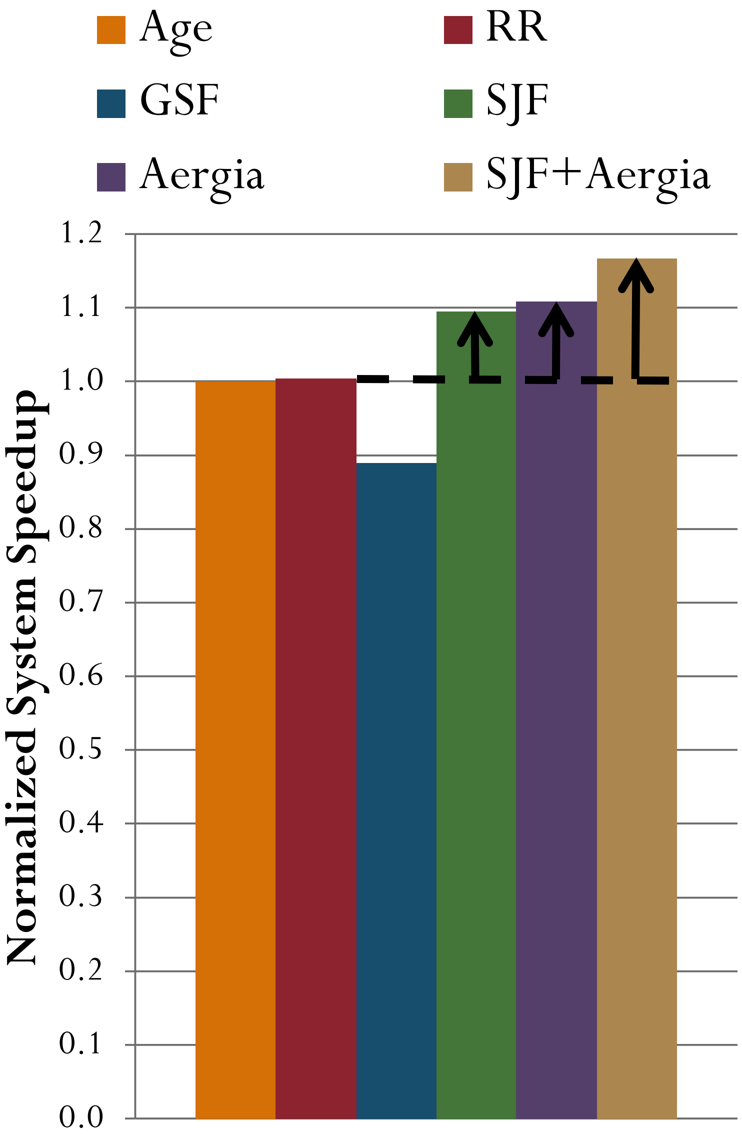
- Problem: Starvation
  - Prioritizing packets can lead to starvation of lower priority packets
- Solution: Time-Based Packet Batching
  - New batches are formed at every T cycles
  - Packets of older batches are prioritized over younger batches

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%



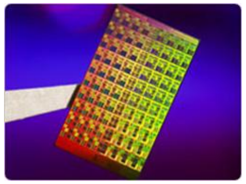
Agenda

- Terminology review
- More on interconnection networks
  - Network properties and performance metrics
  - Buffering and flow control
  - Router design and pipeline options

- **Research on NoCs**
  - The problem with packet scheduling
  - Application-aware packet scheduling
  - Aergia: Latency slack-based packet scheduling
  - **Bufferless networks**

On-Chip Networks (NoC)

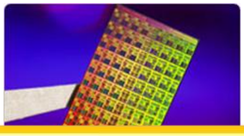
- Connect cores, caches, memory controllers, etc…
- Examples:
  - Intel 80-core Terascale chip
  - MIT RAW chip



- Design goals in NoC design:
  - High throughput, low latency
  - Fairness between cores, QoS, …
  - Low complexity, low cost
  - Power, low energy consumption

On-Chip Networks (NoC)

- Connect cores, caches, memory controllers, etc…
- Examples:
  - Intel 80-core Terascale chip
  - MIT RAW



- Design goals in
  - High throu
  - Fairness be
  - Low compl
  - Power, low

Energy/Power in On-Chip Networks

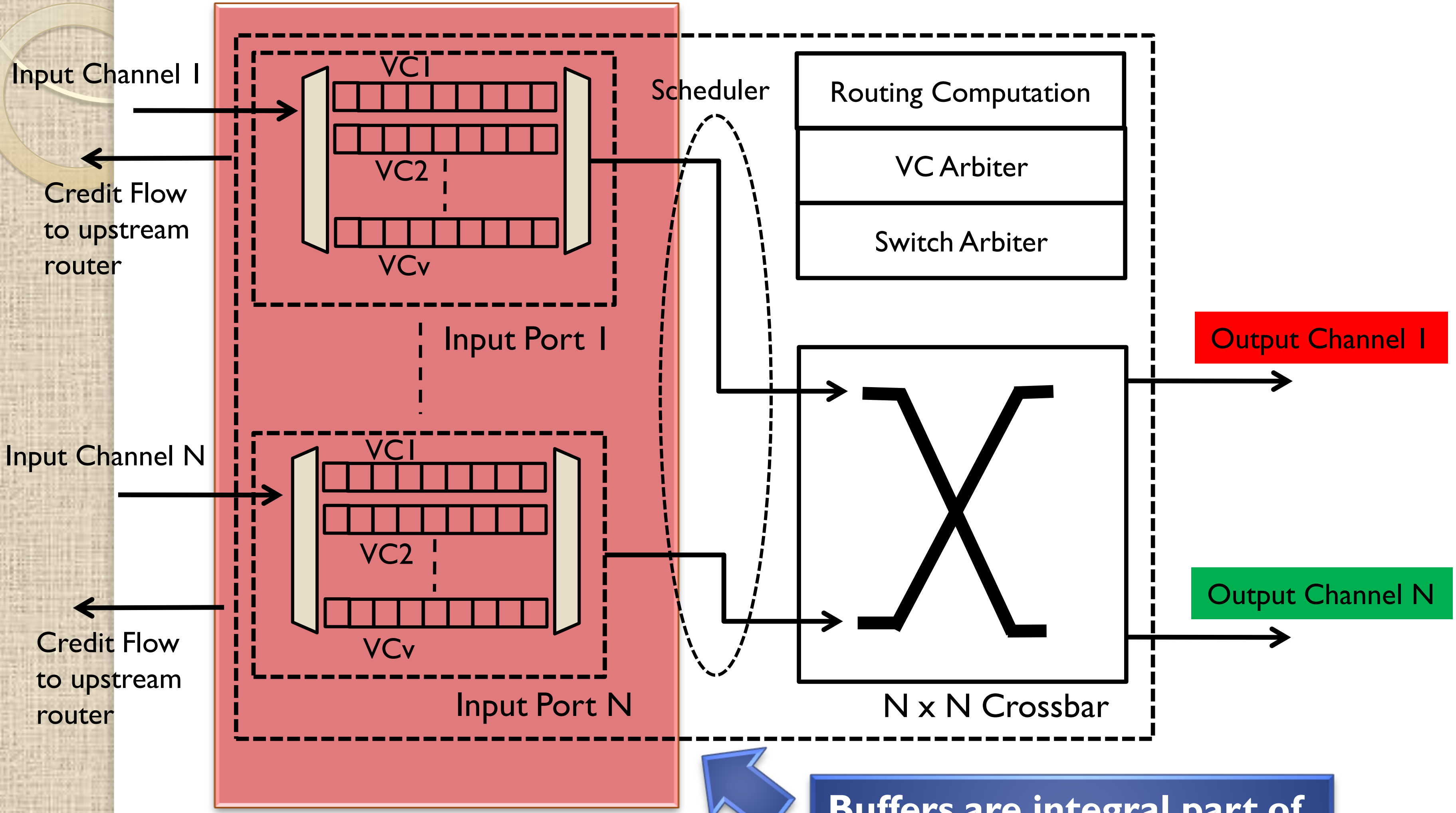
- Power is a key constraint in the design of high-performance processors
- NoCs consume substantial portion of system power
  - ~30% in Intel 80-core Terascale [IEEE Micro'07]
  - ~40% in MIT RAW Chip [ISCA'04]
- NoCs estimated to consume 100s of Watts [Borkar, DAC'07]

Current NoC Approaches

- Existing approaches differ in numerous ways:
  - Network topology [Kim et al, ISCA'07, Kim et al, ISCA'08 etc]
  - Flow control [Michelogiannakis et al, HPCA'09, Kumar et al, MICRO'08, etc]
  - Virtual Channels [Nicopoulos et al, MICRO'06, etc]
  - QoS & fairness mechanisms [Lee et al, ISCA'08, etc]
  - Routing algorithms [Singh et al, CAL'04]
  - Router architecture [Park et al, ISCA'08]
  - Broadcast, Multicast [Jerger et al, ISCA'08, Rodrigo et al, MICRO'08]

Existing work assumes existence of buffers in routers!

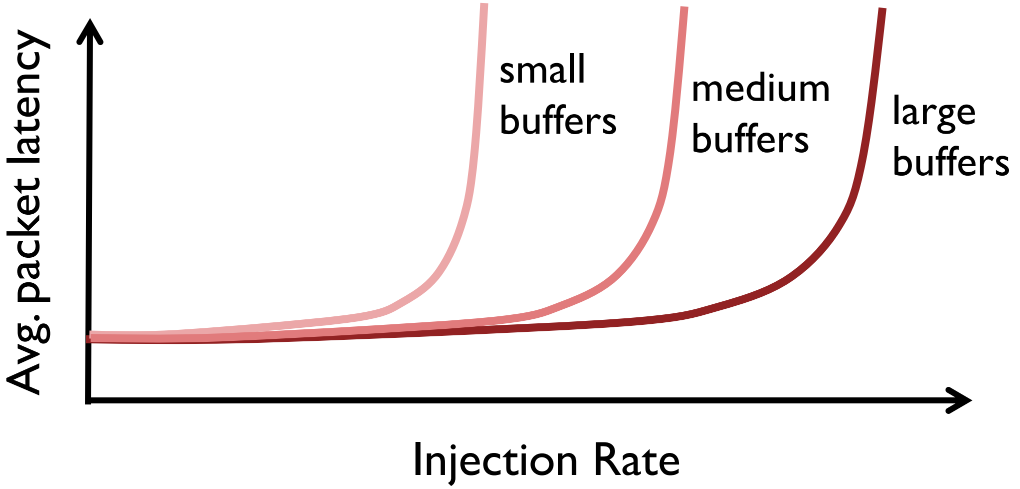
A Typical Router



Buffers are integral part of existing NoC Routers

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 networ 👍

  → buffers increase total availabl

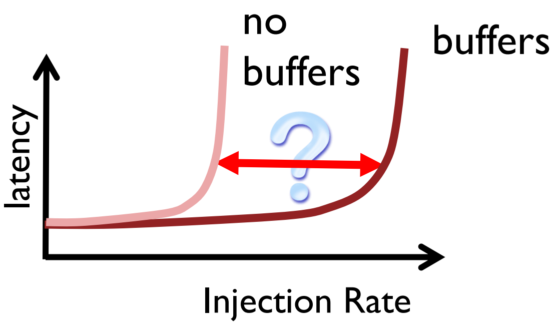
- Buffers consume sig 👎
  - Dynamic e
  - Stati d
- Buff cy 👎
  - ent
  - cation
  - low control
- quire significant chip area 👎
  - .g., in TRIPS prototype chip, input buffers occupy 75% of total on-chip network area [Gratz et al, ICCD'06]

Can we get rid of buffers…?

Going Bufferless…?

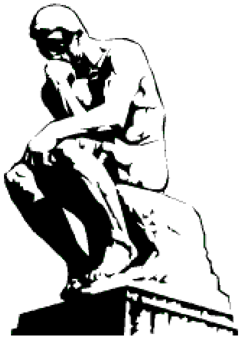
- How much throughput do we lose?

  → 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!

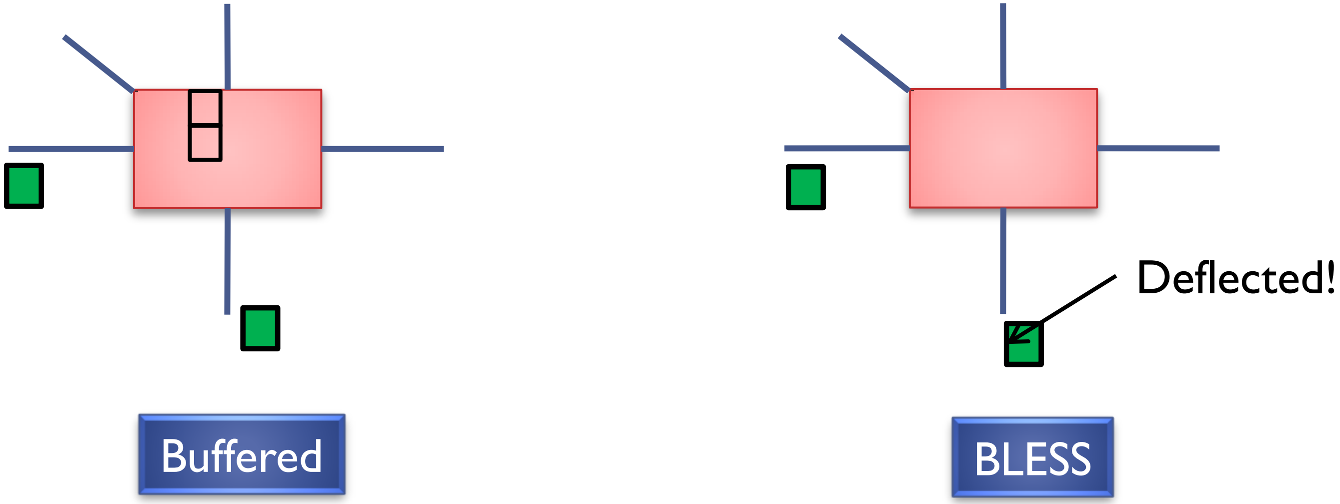
Overview

- Introduction and Background
- Bufferless Routing (BLESS)
  - FLIT-BLESS
  - WORM-BLESS
  - BLESS with buffers
- Advantages and Disadvantages
- Evaluations
- Conclusions

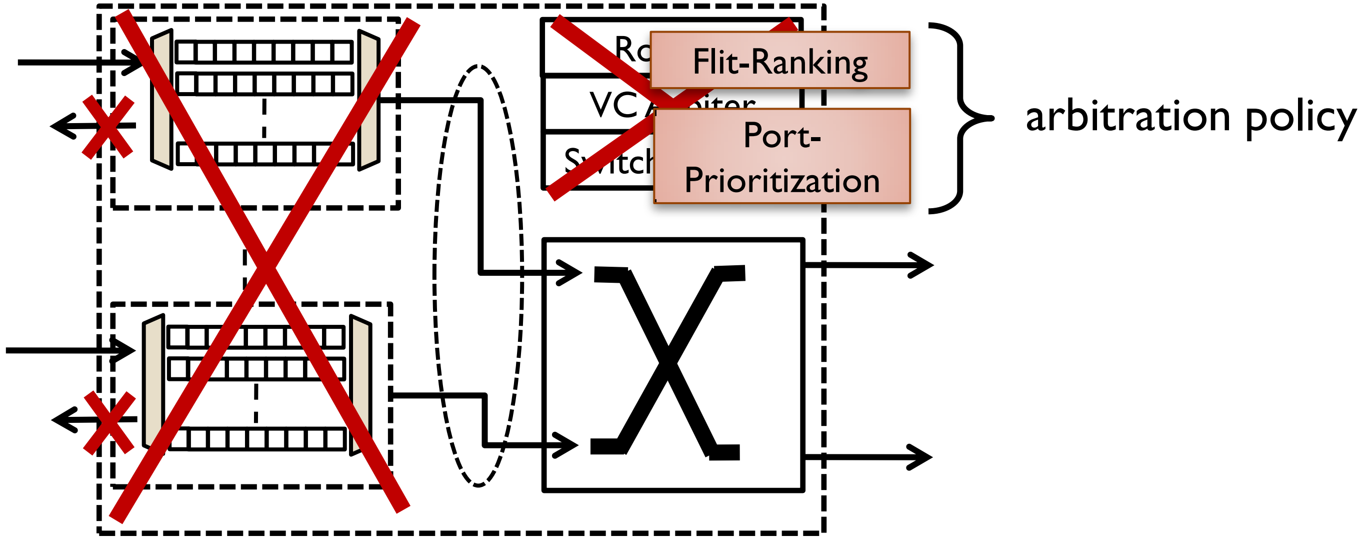
BLESS: Bufferless Routing

- 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

Thomas Moscibroda, Microsoft Research

FLIT-BLESS: Flit-Level Routing

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

Flit-Ranking	1. Oldest-first ranking
Port-Prioritization	2. Assign flit to productive port, if possible. Otherwise, assign to non-productive port.

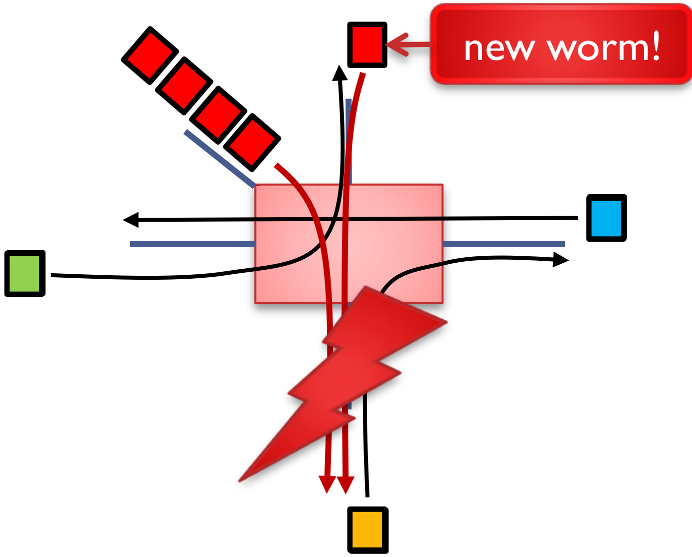
- Network Topology:
  → Can be applied to most topologies (Mesh, Torus, Hypercube, Trees, …)
  1) #output ports ¸ #input ports at every router
  2) every router is reachable from every other router
- Flow Control & Injection Policy:
  → Completely local, inject whenever input port is free
- Absence of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

WORM-BLESS: Wormhole Routing

- Potential downsides of FLIT-BLESS
  - Not-energy optimal (each flits needs header information)
  - Increase in latency (different flits take different path)
  - Increase in receive buffer size

- BLESS with wormhole routing…?
- Problems:
  - Injection Problem
    (not known when it is safe to inject)
  - Livelock Problem
    (packets can be deflected forever)

[Dally, Seitz'86]



WORM-BLESS: Wormhole Routing

Flit-Ranking

Port-Prioritization

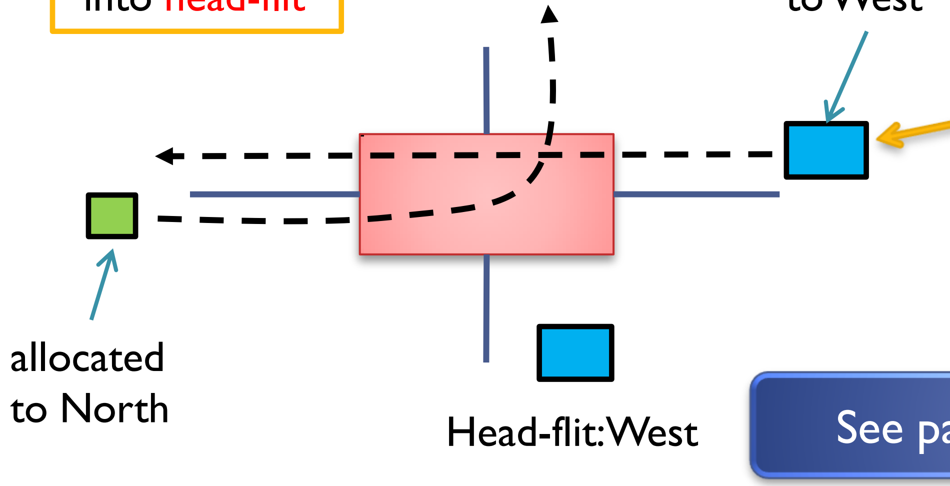
```
1. Oldest-first ranking
2. If flit is head-flit
     a) assign flit to unallocated, productive port
     b) assign flit to allocated, productive port
     c) assign flit to unallocated, non-productive port
     d) assign flit to allocated, non-productive port
   else,
     a) assign flit to port that is allocated to worm
```

Deflect worms if necessary!

Truncate worms if necessary!

At low congestion, packets travel routed as worms

Body-flit turns into head-flit



This worm is truncated!

& deflected!

See paper for details…

Thomas Moscibroda, Microsoft Research

BLESS with Buffers

- BLESS without buffers is extreme end of a continuum
- BLESS can be integrated with buffers
  - FLIT-BLESS with Buffers
  - WORM-BLESS with Buffers
- Whenever a buffer is full, it's first flit becomes must-schedule
- must-schedule flits must be deflected if necessary

See paper for details…

Overview

- Introduction and Background
- Bufferless Routing (BLESS)
  - FLIT-BLESS
  - WORM-BLESS
  - BLESS with buffers
- Advantages and Disadvantages
- Evaluations
- Conclusions

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

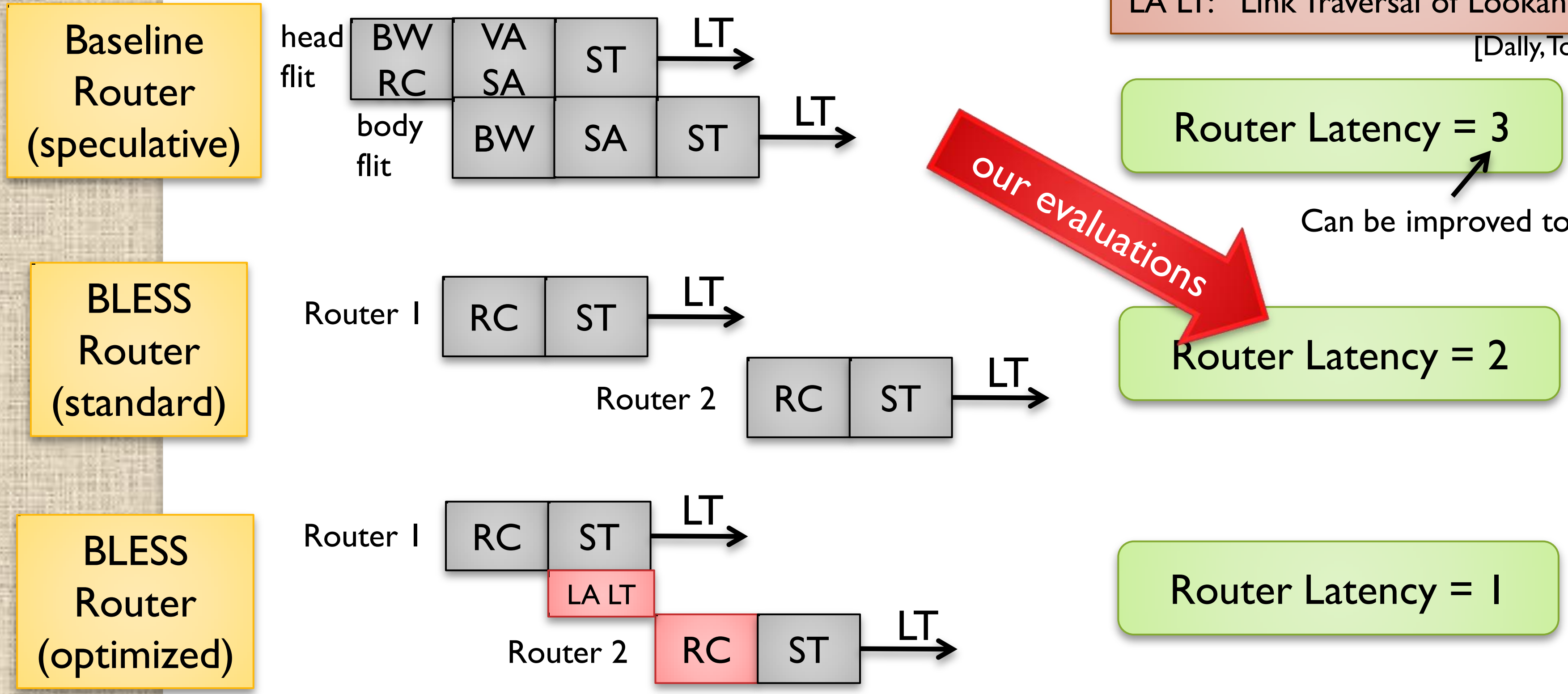
Impact on energy…?

Reduction of Router Latency

- BLESS gets rid of input buffers and virtual channels

BW: Buffer Write
RC: Route Computation
VA: Virtual Channel Allocation
SA: Switch Allocation
ST: Switch Traversal
LT: Link Traversal
LA LT: Link Traversal of Lookahead

[Dally,Towles'04]



Baseline Router (speculative)

head flit: BW RC | VA SA | ST | LT

body flit: BW | SA | ST | LT

Router Latency = 3

Can be improved to 2.

our evaluations

BLESS Router (standard)

Router 1: RC | ST | LT

Router 2: RC | ST | LT

Router Latency = 2

BLESS Router (optimized)

Router 1: RC | ST | LT

LA LT

Router 2: RC | ST | LT

Router Latency = 1

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

Extensive evaluations in the paper!

Impact on energy…?

Evaluation Methodology

- 2D mesh network, router latency is 2 cycles
  - 4x4, 8 core, 8 L2 cache banks (each node is a core or an L2 bank)
  - 4x4, 16 core, 16 L2 cache banks (each no
  - 8x8, 16 core, 64 L2 cache banks (each no
  - 128-bit wide links, 4-flit data packets, 1-f
  - For baseline configuration: 4 VCs per phys

Simulation is cycle-accurate
→ Models stalls in network and processors
→ Self-throttling behavior
→ Aggressive processor model

- Benchmarks
  - Multiprogrammed SPEC CPU2006 and Windows Desktop applications
  - Heterogeneous and homogenous application mixes
  - Synthetic traffic patterns: UR, Transpose, Tornado, Bit Complement
- x86 processor model based on Intel P
  - 2 GHz processor, 128-entry instruction w
  - 64Kbyte private L1 caches
  - Total 16Mbyte shared L2 caches; 16 MSHRs per bank
  - DRAM model based on Micron DDR2-800

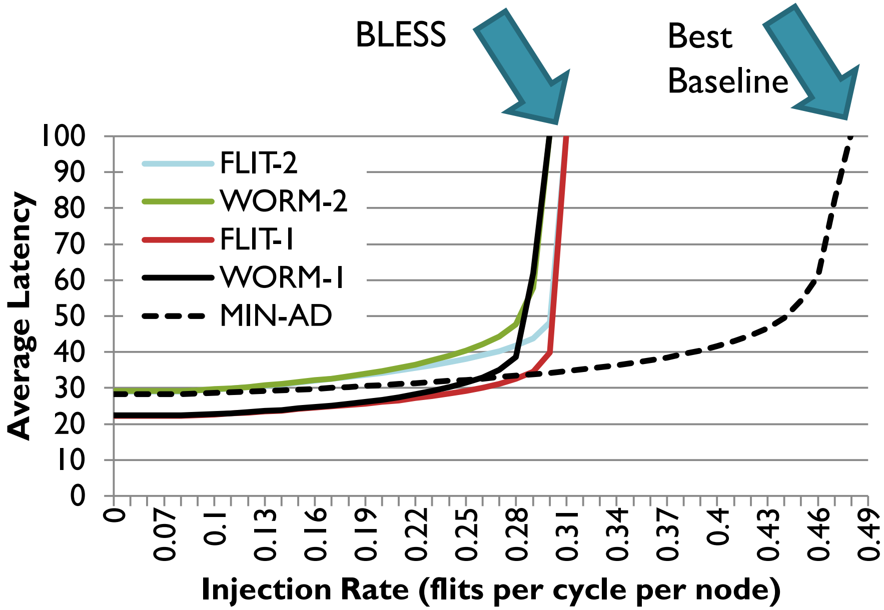
Most of our evaluations with perfect L2 caches
→ Puts maximal stress on NoC

Evaluation Methodology

- Energy model provided by Orion simulator [MICRO'02]
  - 70nm technology, 2 GHz routers at 1.0 V_{dd}
- For BLESS, we model
  - Additional energy to transmit header information
  - Additional buffers needed on the receiver side
  - Additional logic to reorder flits of individual packets at receiver
- We partition network energy into buffer energy, router energy, and link energy, each having static and dynamic components.
- Comparisons against non-adaptive and aggressive adaptive buffered routing algorithms (DO, MIN-AD, ROMM)

Evaluation – Synthethic Traces

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

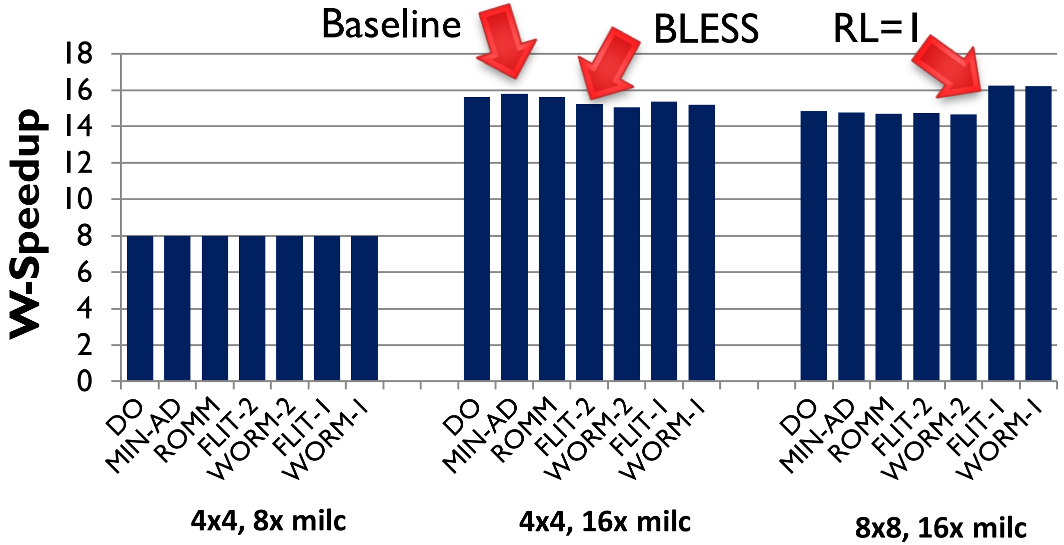

BLESS

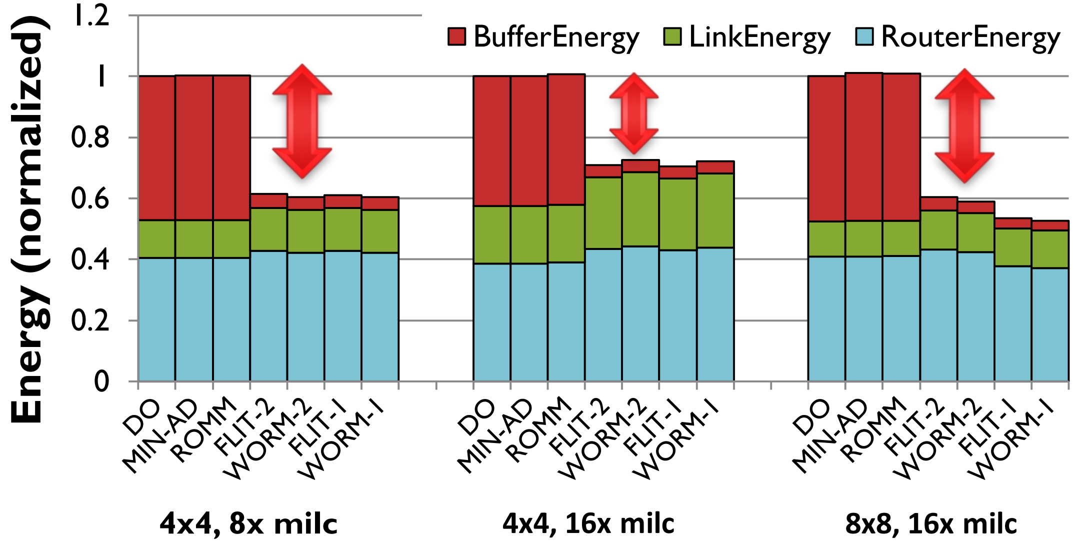
Best Baseline

Is BLESS doomed...?

Evaluation – Homogenous Case Study

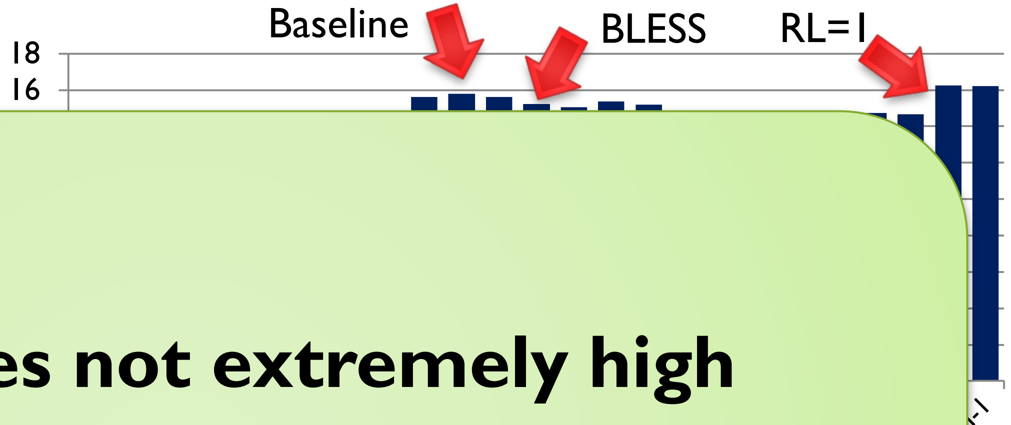
- milc benchmarks (moderately intensive)
- Perfect caches!
- Very little performance degradation with BLESS (less than 4% in dense network)
- With router latency 1, BLESS can even outperform baseline (by ~10%)
- Significant energy improvements (almost 40%)





Evaluation – Homogenous Case Study

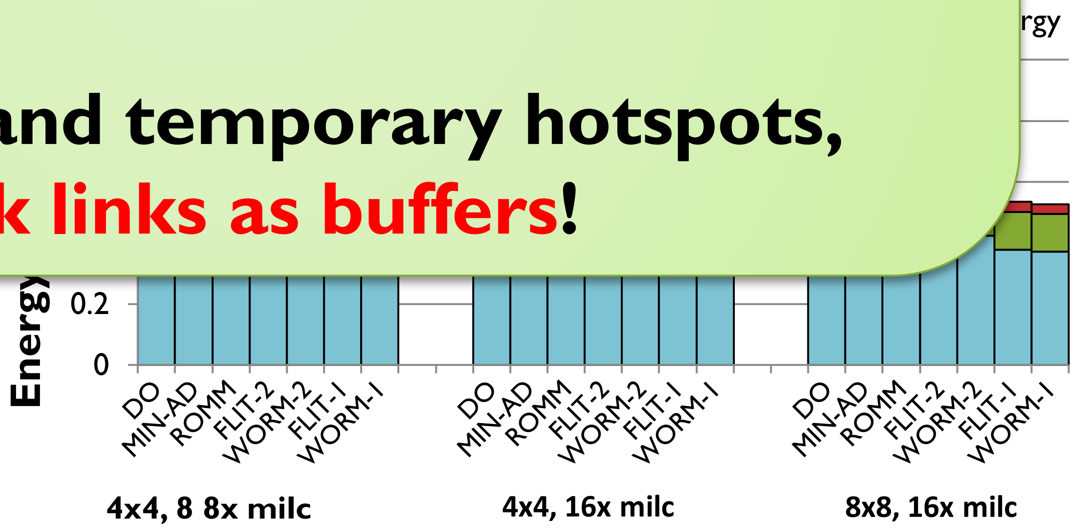
- milc benchmarks



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%)

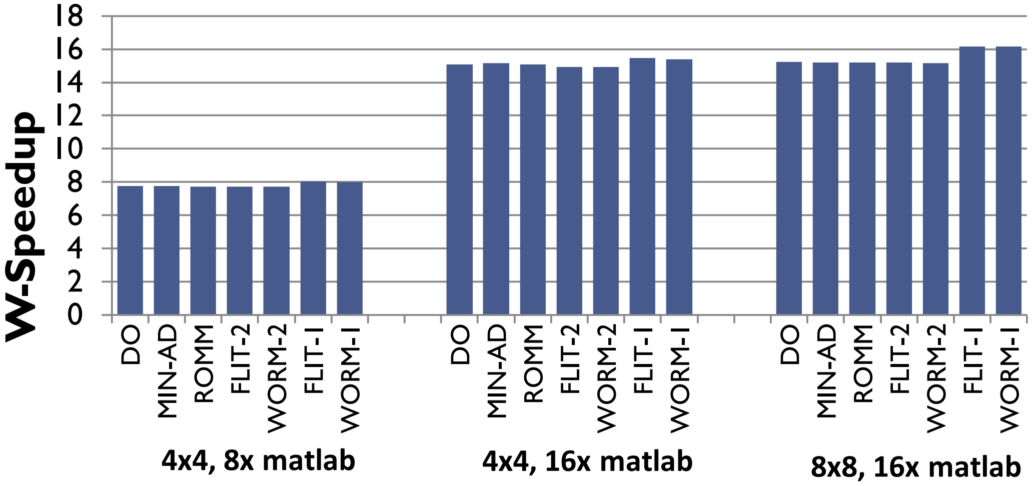
Evaluation – Further Results

See paper for details…

- BLESS increases buffer requirement at receiver by at most 2x
  → overall, energy is still reduced

- Impact of memory latency

  → with real caches, very little slowdown! (at most 1.5%)



Evaluation – Further Results

See paper for details…

- BLESS increases buffer requirement at receiver by at most 2x
  → overall, energy is still reduced

- Impact of memory latency
  → with real caches, very little slowdown! (at most 1.5%)

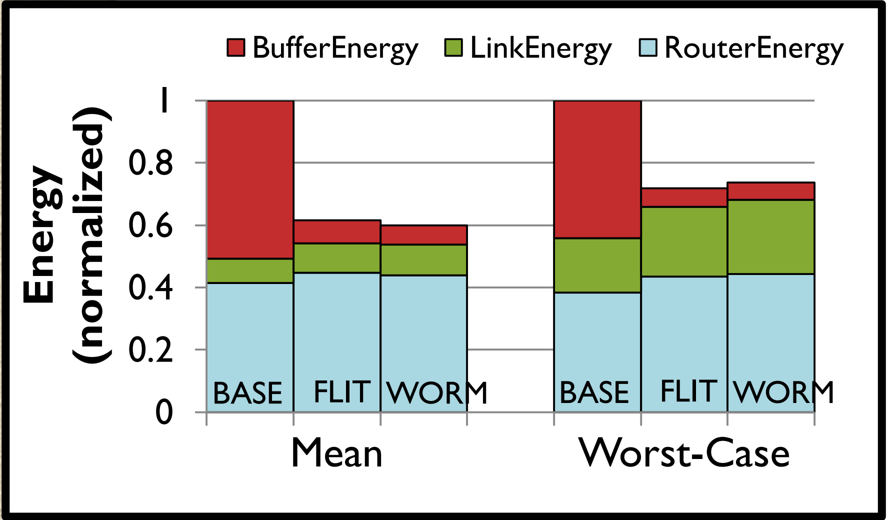
- Heterogeneous application mixes
  (we evaluate several mixes of intensive and non-intensive applications)
  → little performance degradation
  → significant energy savings in all cases
  → no significant increase in unfairness across different applications

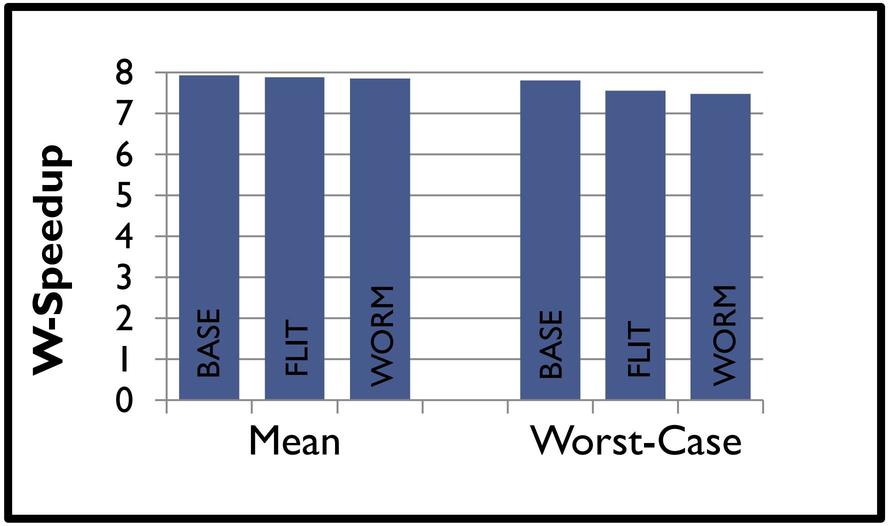
- Area savings: ~60% of network area can be saved!

Evaluation – Aggregate Results

- Aggregate results over all 29 applications

Sparse Network	Perfect L2		Realistic L2	
	Average	Worst-Case	Average	Worst-Case
∆ Network Energy	-39.4%	-28.1%	-46.4%	-41.0%
∆ System Performance	-0.5%	-3.2%	-0.15%	-0.55%





Thomas Moscibroda, Microsoft Research

Evaluation – Aggregate Results

- Aggregate results over all 29 applications

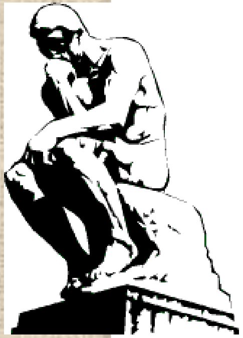
Sparse Network	Perfect L2		Realistic L2	
	Average	Worst-Case	Average	Worst-Case
∆ Network Energy	-39.4%	-28.1%	-46.4%	-41.0%
∆ System Performance	-0.5%	-3.2%	-0.15%	-0.55%

Dense Network	Perfect L2		Realistic L2	
	Average	Worst-Case	Average	Worst-Case
∆ Network Energy	-32.8%	-14.0%	-42.5%	-33.7%
∆ System Performance	-3.6%	-17.1%	-0.7%	-1.5%

Conclusion

- 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!



- We are currently working on future research.
  - Support for quality of service, different traffic classes, energy-management, etc…