

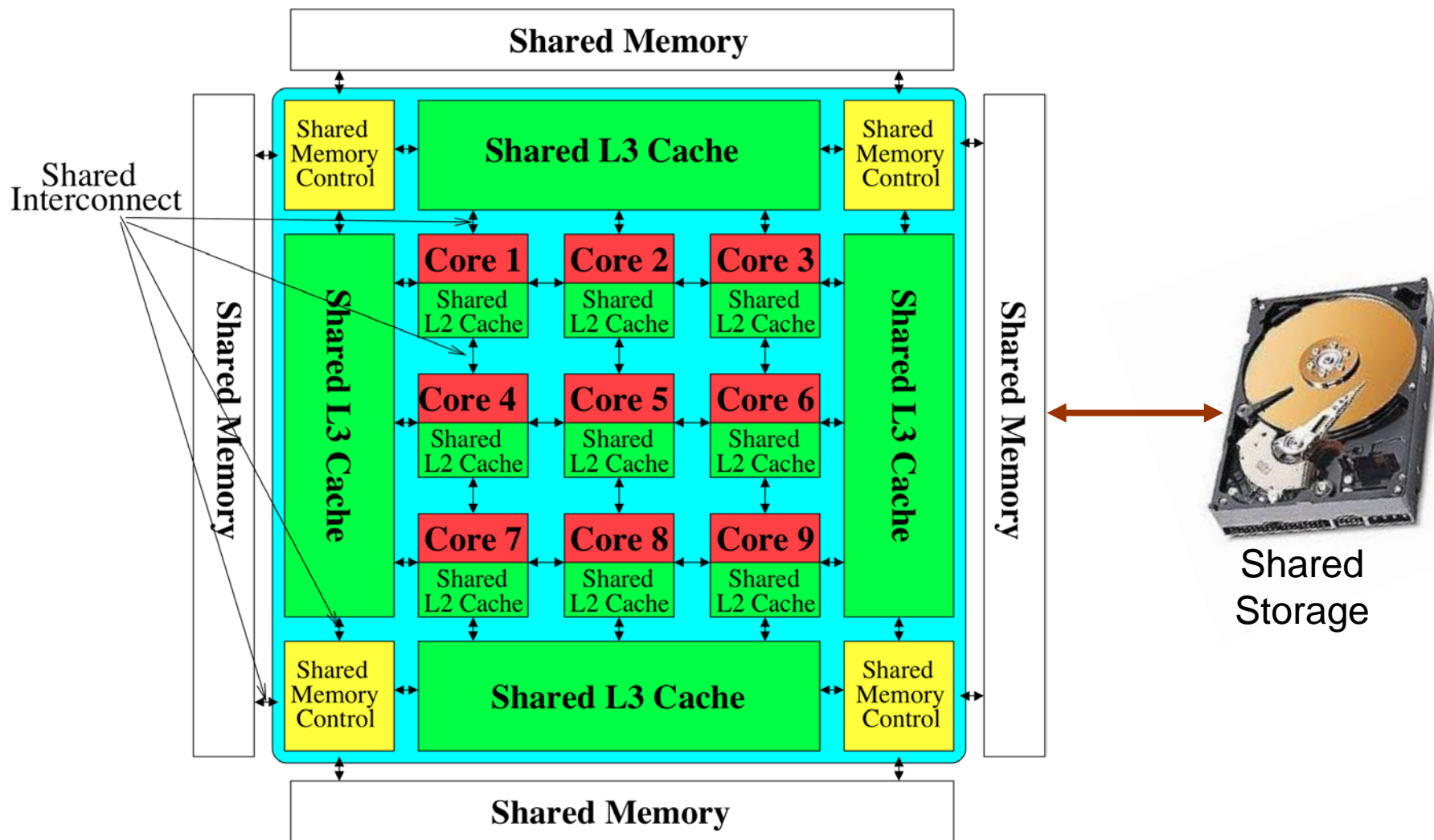
Computer Architecture:
Interconnects:
Off-Chip and On-Chip

15-740

Carnegie Mellon University

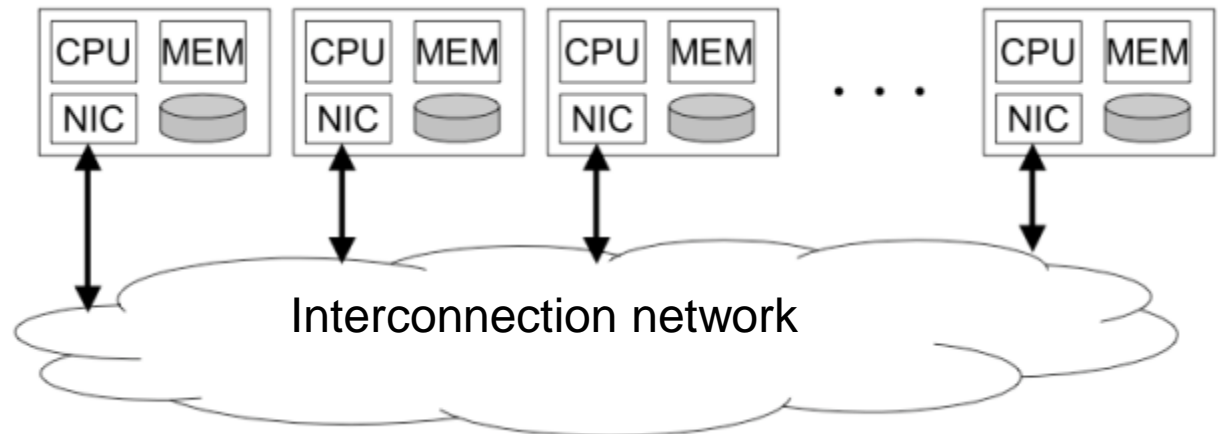
Interconnect Basics

Interconnect in a Multi-Core System



Where Is Interconnect Used?

- To connect components
- Many examples
 - Processors and processors
 - Processors and memories (banks)
 - Processors and caches (banks)
 - Caches and caches
 - I/O devices



Why Is It Important?

- Affects the scalability of the system
 - How large of a system can you build?
 - How easily can you add more processors?
- Affects performance and energy efficiency
 - How fast can processors, caches, and memory communicate?
 - How long are the latencies to memory?
 - How much energy is spent on communication?

Interconnection Network Basics

- Topology
 - Specifies the way switches are wired
 - Affects routing, reliability, throughput, latency, cost
- Routing (algorithm)
 - How does a message get from source to destination
 - Static or adaptive
- Buffering and Flow Control
 - What do we store within the network?
 - Entire packets, parts of packets, etc?
 - How do we throttle during oversubscription?
 - Tightly coupled with routing strategy

Topology

- Bus (simplest)
- Point-to-point connections (ideal and most costly)
- Crossbar (less costly)
- Ring
- Tree
- Omega
- Hypercube
- Mesh
- Torus
- Butterfly
- ...

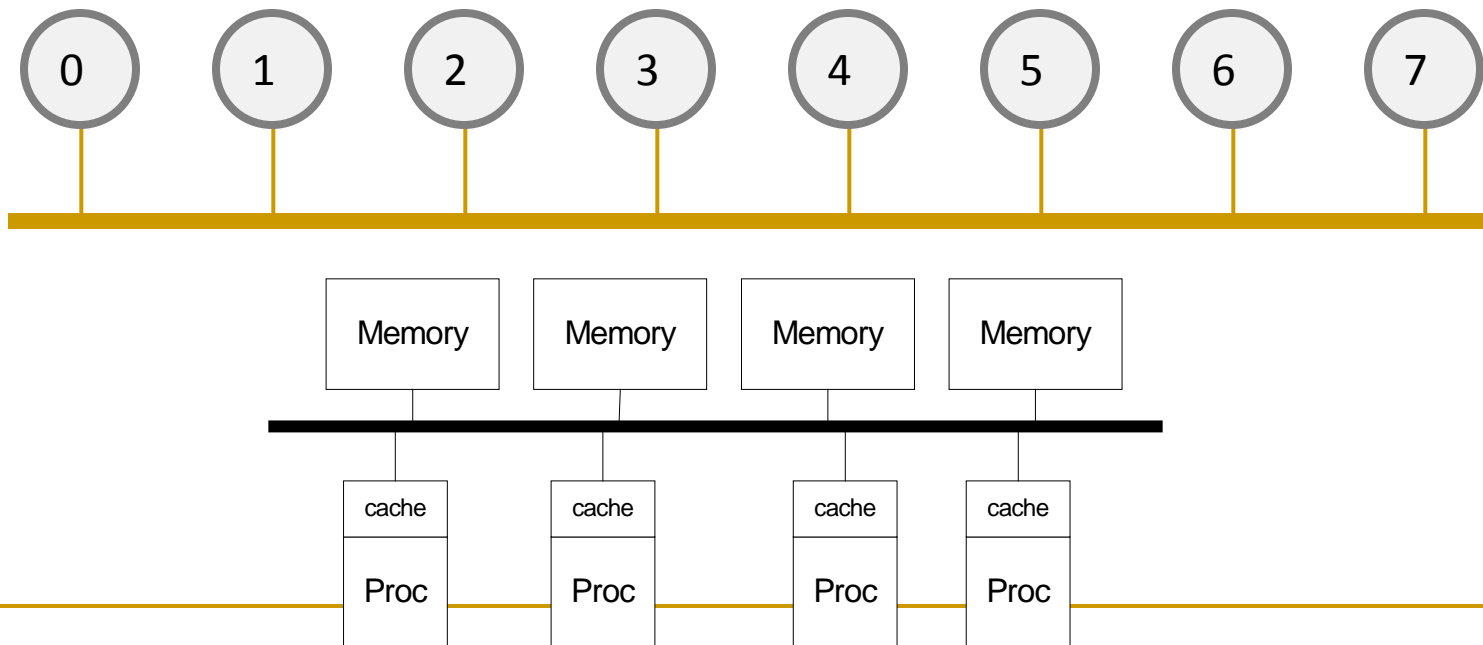
Metrics to Evaluate Interconnect Topology

- Cost
- Latency (in hops, in nanoseconds)
- Contention

- Many others exist you should think about
 - Energy
 - Bandwidth
 - Overall system performance

Bus

- + Simple
- + Cost effective for a small number of nodes
- + Easy to implement coherence (snooping and serialization)
- Not scalable to large number of nodes (limited bandwidth, electrical loading → reduced frequency)
- High contention → fast saturation

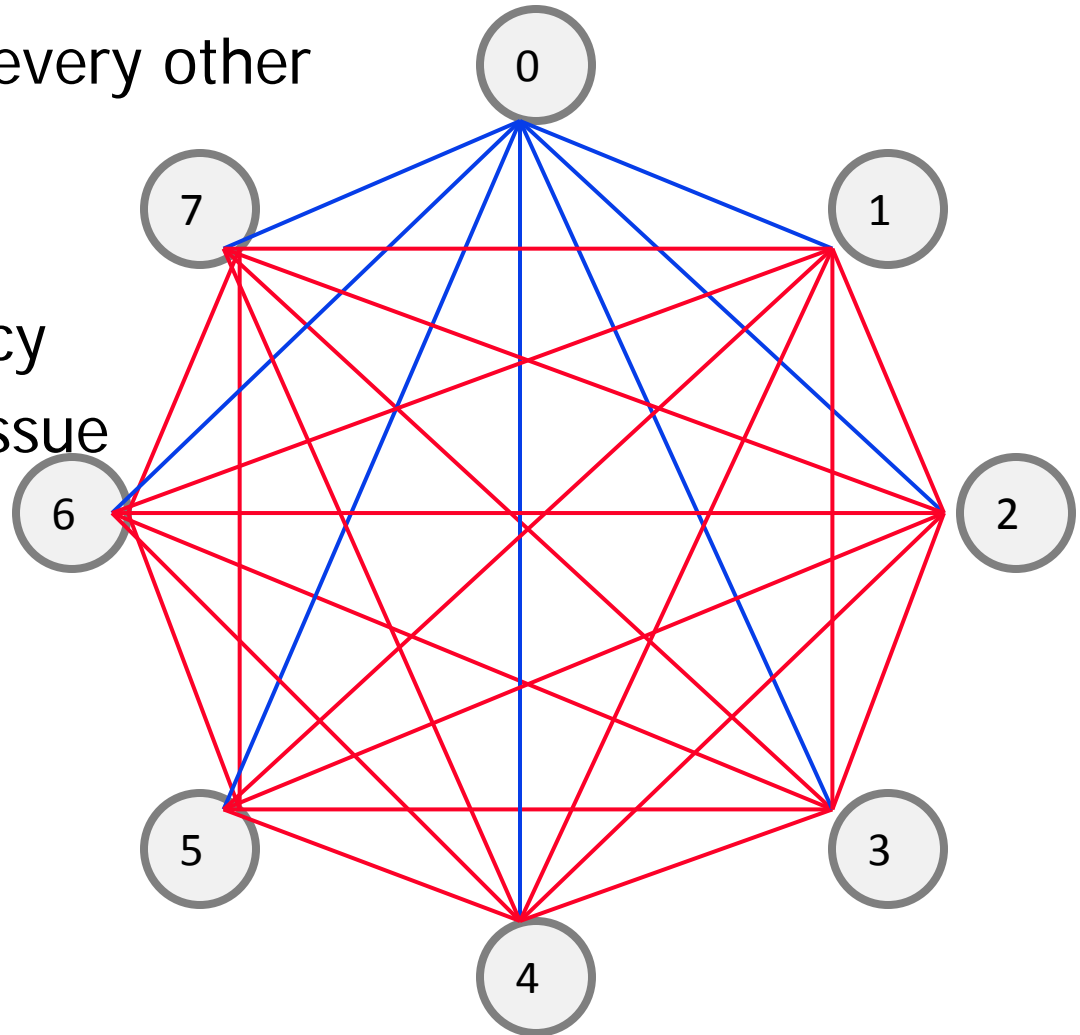


Point-to-Point

Every node connected to every other

- + Lowest contention
- + Potentially lowest latency
- + Ideal, if cost is not an issue

- Highest cost
- $O(N)$ connections/ports per node
- $O(N^2)$ links
- Not scalable
- How to lay out on chip?



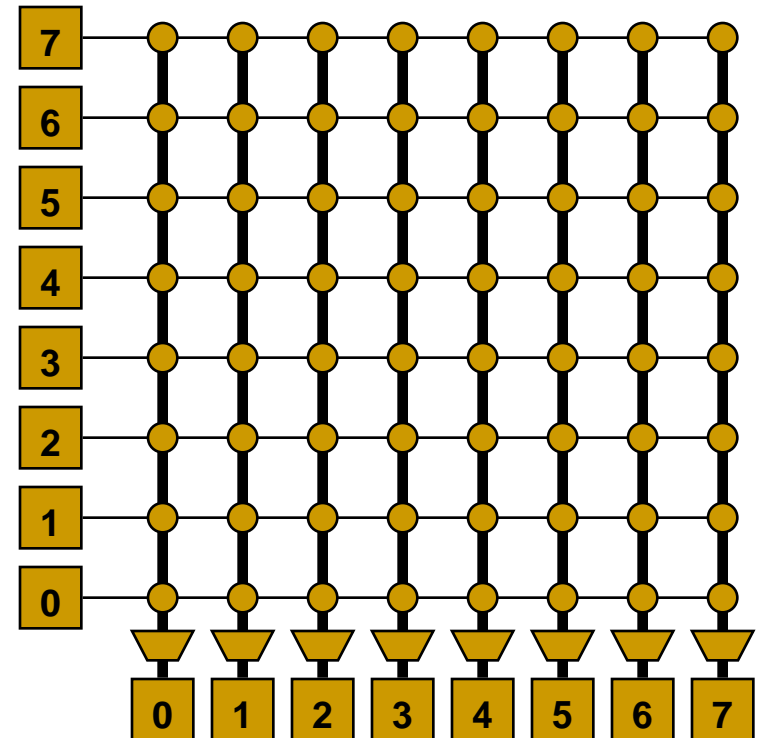
Crossbar

- Every node connected to every other (non-blocking) except only one can be using the connection at any given time
- Enables concurrent sends to non-conflicting destinations
- Good for small number of nodes

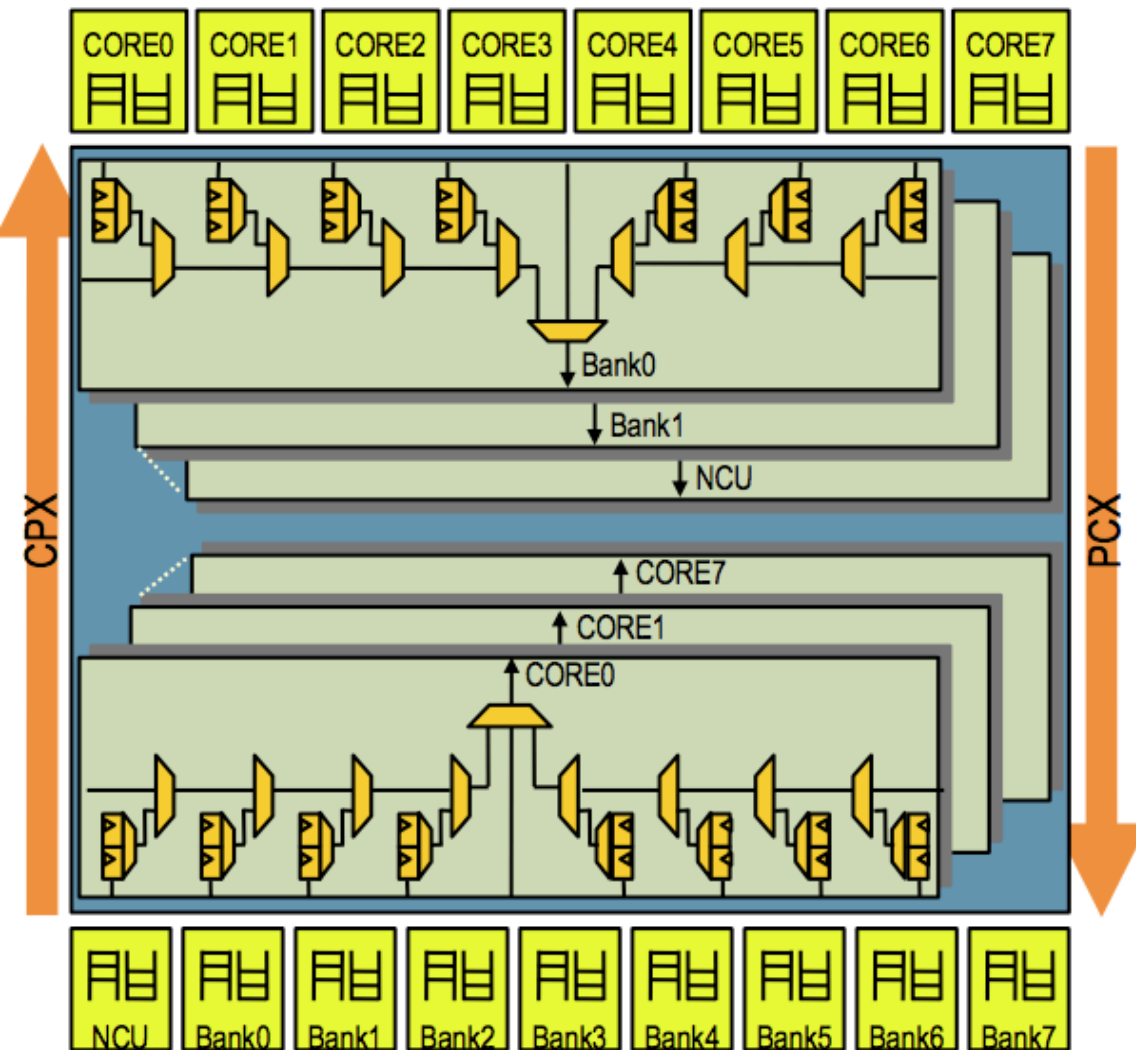
- + Low latency and high throughput
- Expensive
- Not scalable $\rightarrow O(N^2)$ cost
- Difficult to arbitrate as N increases

Used in core-to-cache-bank networks in

- IBM POWER5
- Sun Niagara I/II

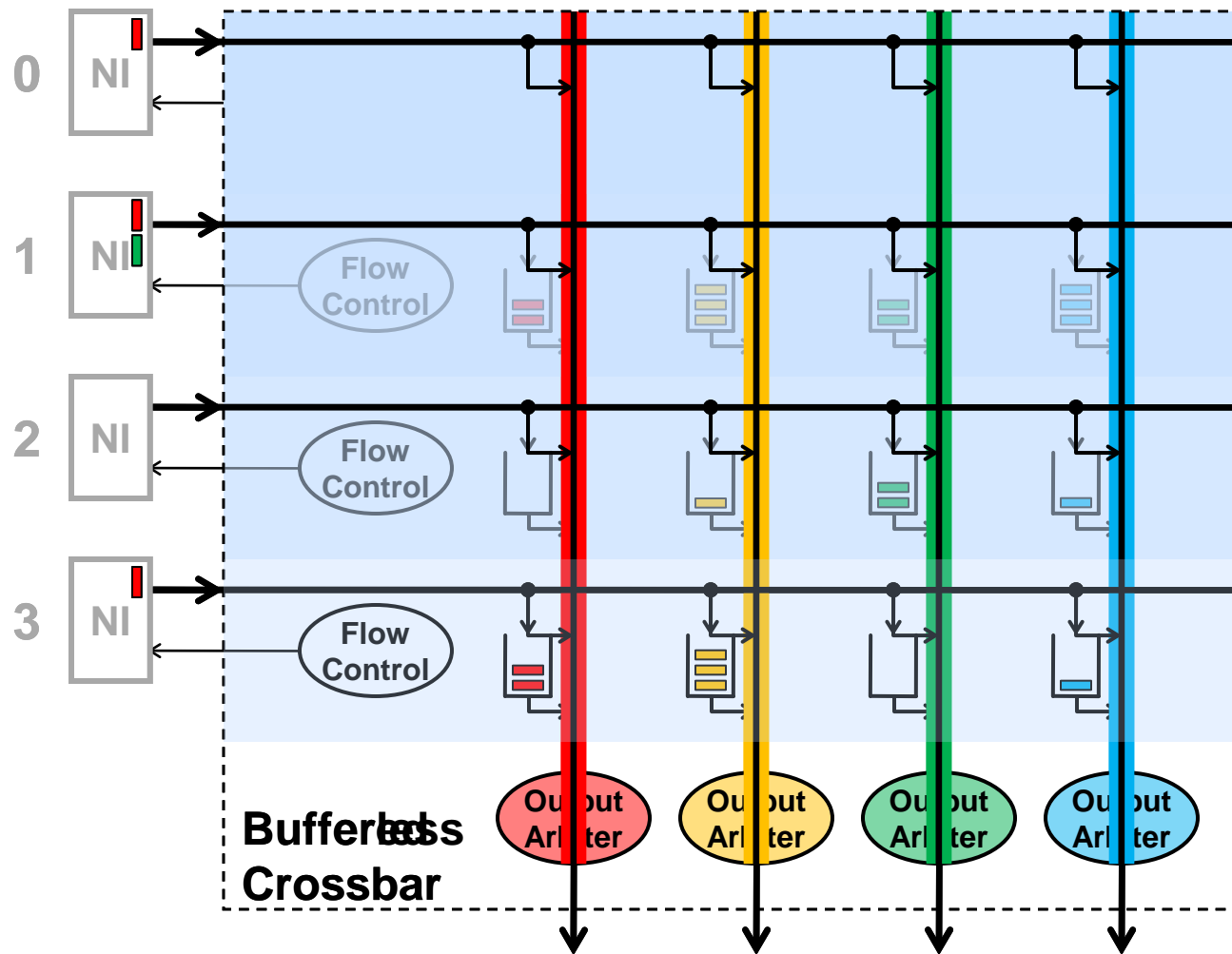


Sun UltraSPARC T2 Core-to-Cache Crossbar



- High bandwidth interface between 8 cores and 8 L2 banks & NCU
- 4-stage pipeline: req, arbitration, selection, transmission
- 2-deep queue for each src/dest pair to hold data transfer request

Buffered Crossbar



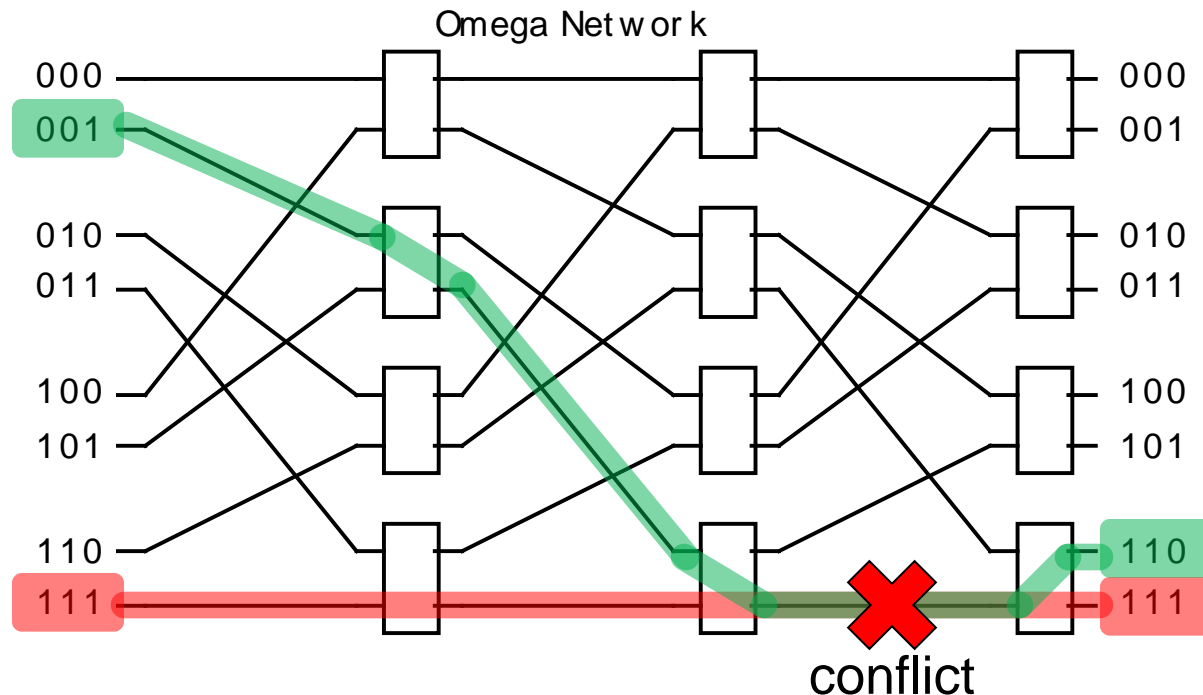
- + Simpler arbitration/scheduling
- + Efficient support for variable-size packets
- Requires N^2 buffers

Can We Get Lower Cost than A Crossbar?

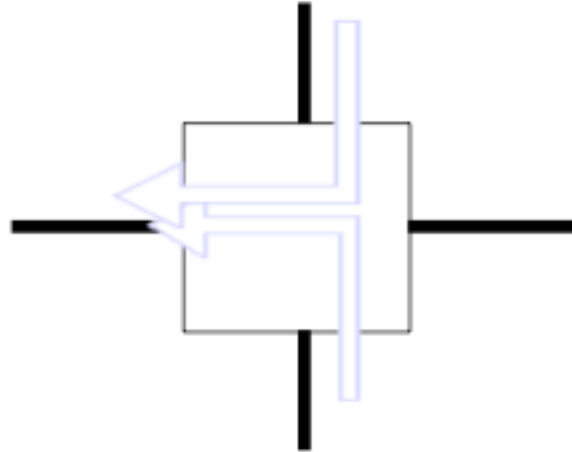
- Yet still have low contention?
- Idea: Multistage networks

Multistage Logarithmic Networks

- Idea: Indirect networks with multiple layers of switches between terminals/nodes
- Cost: $O(N \log N)$, Latency: $O(\log N)$
- Many variations (Omega, Butterfly, Benes, Banyan, ...)
- Omega Network:



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

Aside: Circuit vs. Packet Switching

- **Circuit switching** sets up full path
 - Establish route then send data
 - (no one else can use those links)



- **Packet switching** routes per packet
 - Route each packet individually (possibly via different paths)
 - if link is free, any packet can use it



Aside: Circuit vs. Packet Switching

- **Circuit switching** sets up full path
 - Establish route then send data
 - (no one else can use those links)
 - + faster arbitration
 - setting up and bringing down links takes time
- **Packet switching** routes per packet
 - Route each packet individually (possibly via different paths)
 - if link is free, any packet can use it
 - potentially slower --- must dynamically switch
 - + no setup, bring down time
 - + more flexible, does not underutilize links

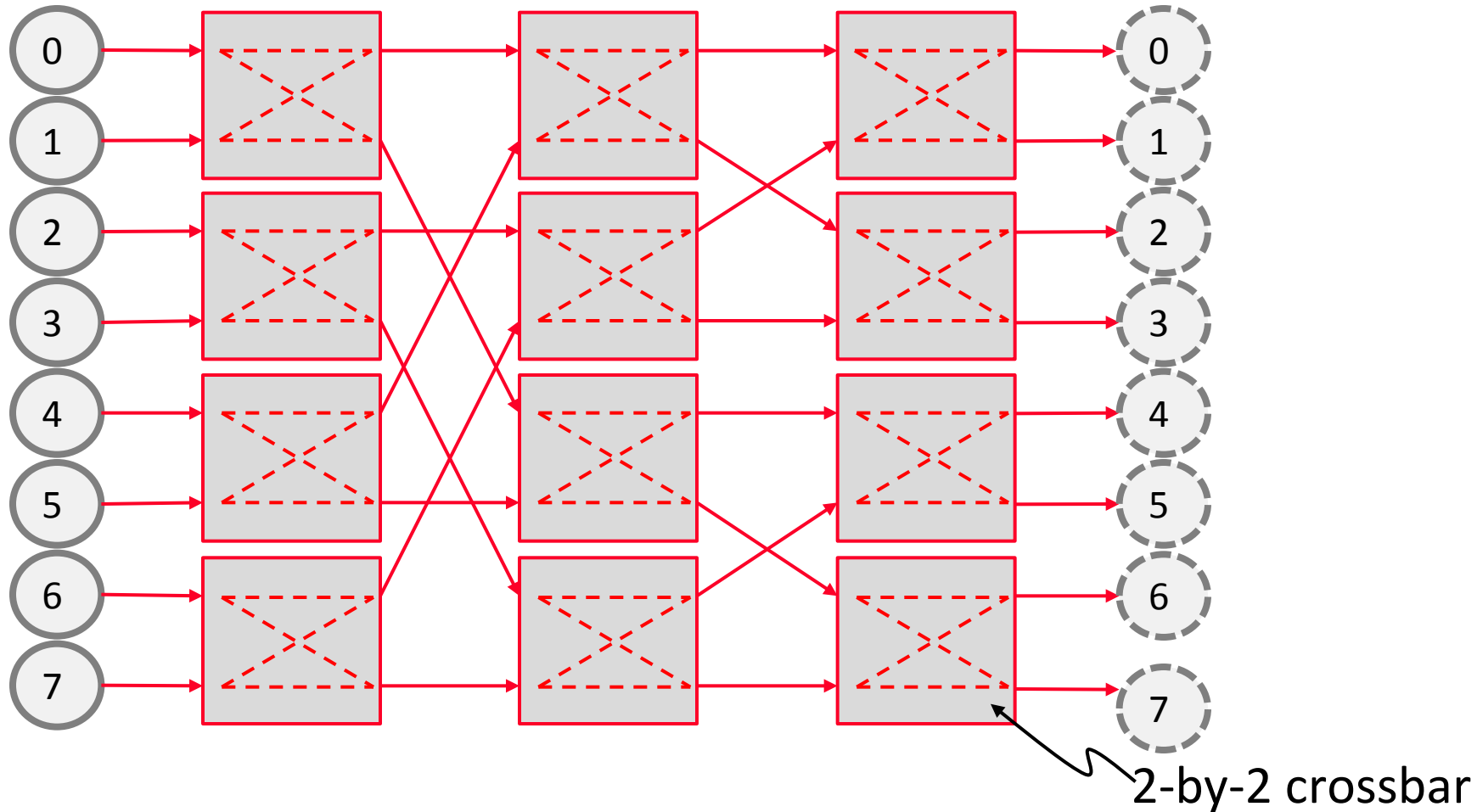


Switching vs. Topology

- Circuit/packet switching choice independent of topology
- It is a higher-level protocol on how a message gets sent to a destination

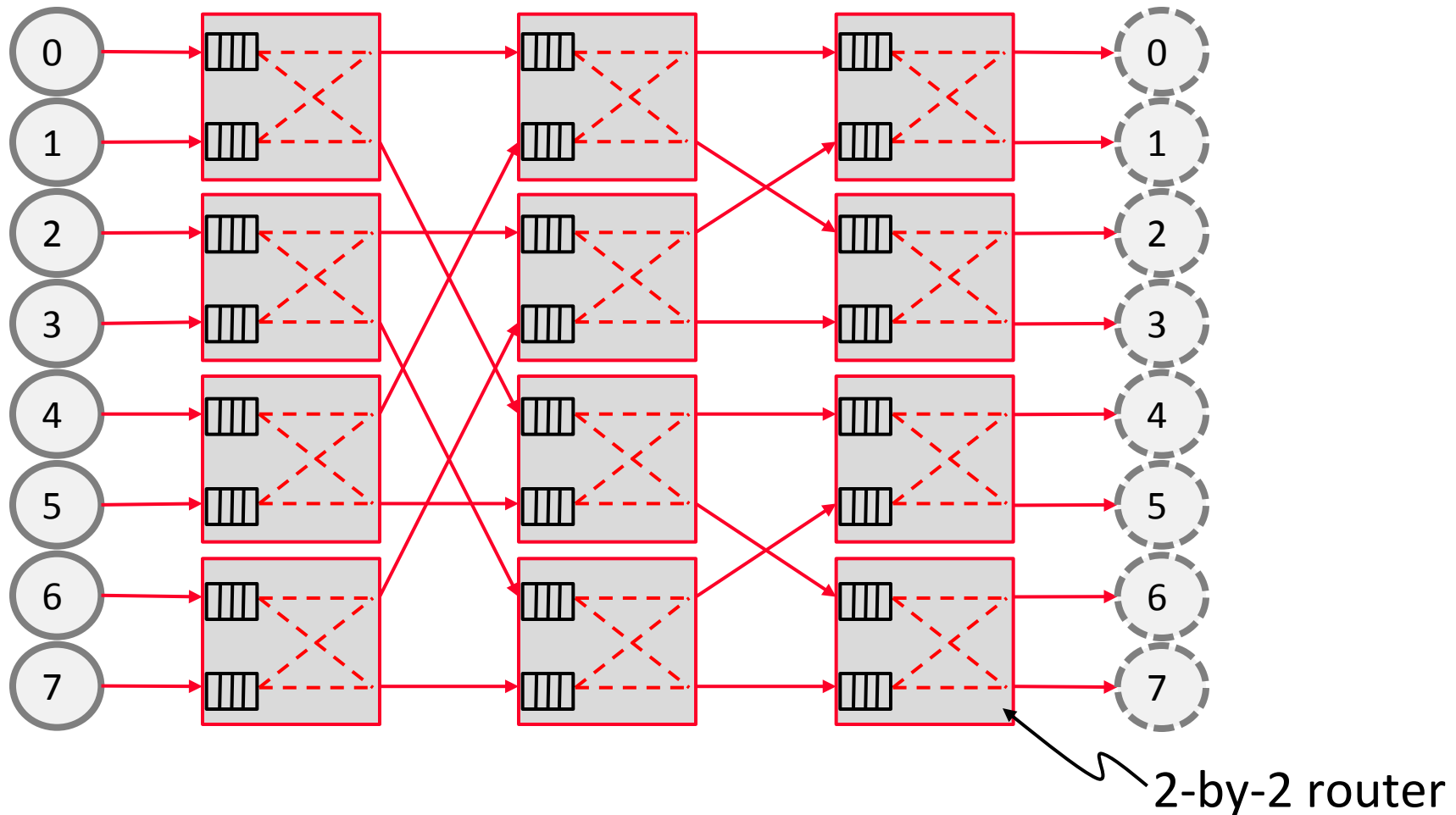
- However, some topologies are more amenable to circuit vs. packet switching

Multistage Circuit Switched



- More restrictions on feasible concurrent Tx-Rx pairs
- But more scalable than crossbar in cost, e.g., $O(N \log N)$ for Butterfly

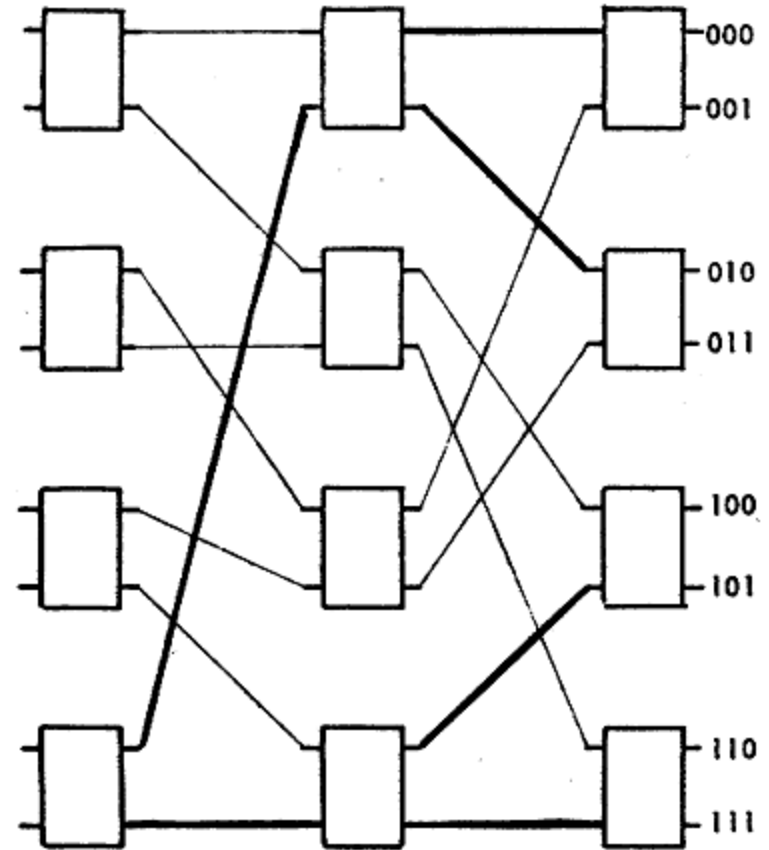
Multistage Packet Switched



- Packets “hop” from router to router, pending availability of the next-required switch and buffer

Another Example: Delta Network

- Single path from source to destination
- Does not support all possible permutations
- Proposed to replace costly crossbars as processor-memory interconnect
- Janak H. Patel , “[Processor-Memory Interconnections for Multiprocessors](#),” ISCA 1979.



8x8 Delta network

Another Example: Omega Network

- Single path from source to destination
- All stages are the same
- Used in NYU Ultracomputer
- Gottlieb et al. “[The NYU Ultracomputer-designing MIMD, shared-memory parallel machine,](#)” ISCA 1982.

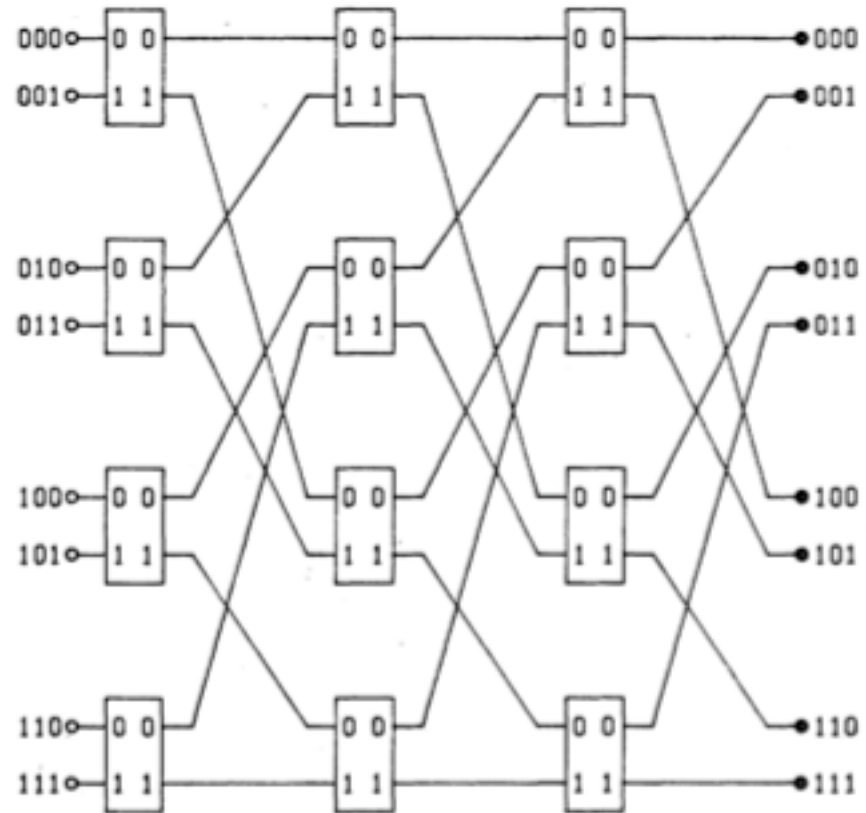
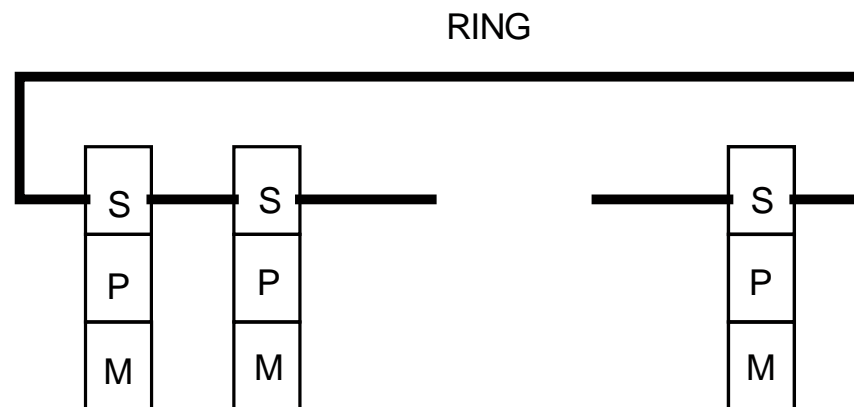


Fig. 2. Omega-network ($N = 8$).

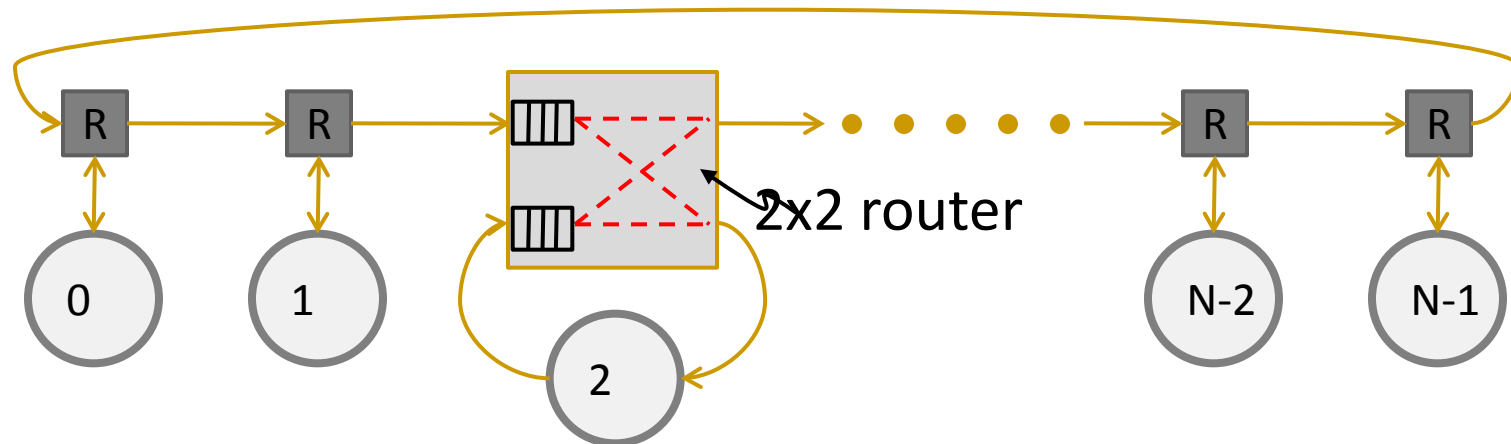
Ring

- + Cheap: $O(N)$ cost
- High latency: $O(N)$
- Not easy to scale
 - Bisection bandwidth remains constant

Used in Intel Haswell, Intel Larrabee, IBM Cell, many commercial systems today



Unidirectional Ring



- Simple topology and implementation
 - ❑ Reasonable performance if N and performance needs (bandwidth & latency) still moderately low
 - ❑ $O(N)$ cost
 - ❑ $N/2$ average hops; latency depends on utilization

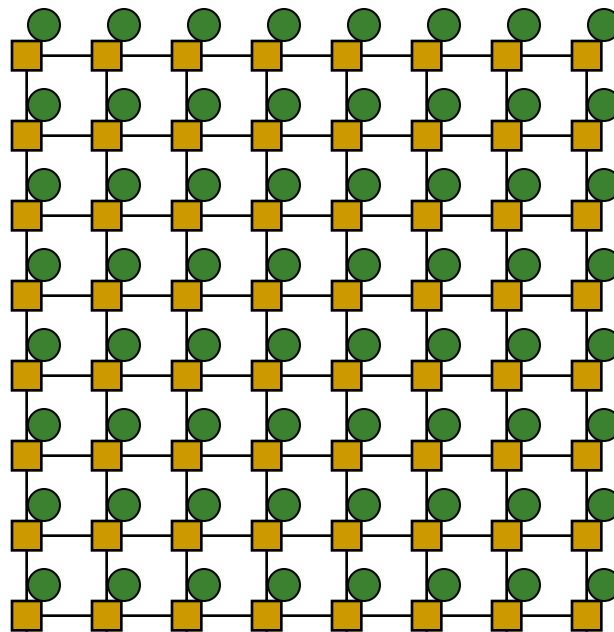
Bidirectional Rings

- + Reduces latency
- + Improves scalability
- Slightly more complex injection policy (need to select which ring to inject a packet into)

Mesh

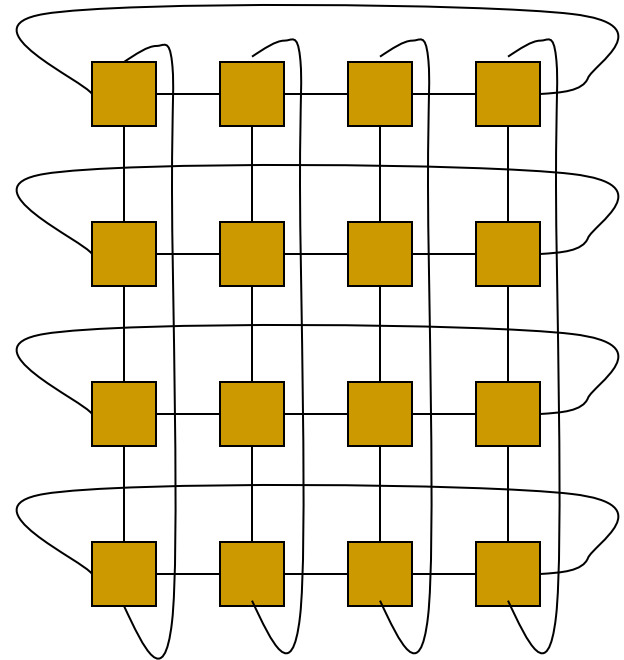
- $O(N)$ cost
- Average latency: $O(\sqrt{N})$
- Easy to layout on-chip: regular and equal-length links
- Path diversity: many ways to get from one node to another

- Used in Tileria 100-core
- And many on-chip network prototypes



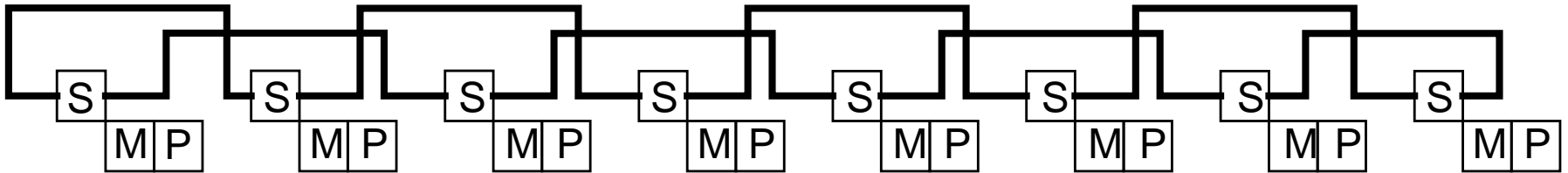
Torus

- Mesh is not symmetric on edges: performance very sensitive to placement of task on edge vs. middle
- Torus avoids this problem
- + Higher path diversity (and bisection bandwidth) than mesh
- Higher cost
- Harder to lay out on-chip
 - Unequal link lengths



Torus, continued

- Weave nodes to make inter-node latencies \sim constant



Trees

Planar, hierarchical topology

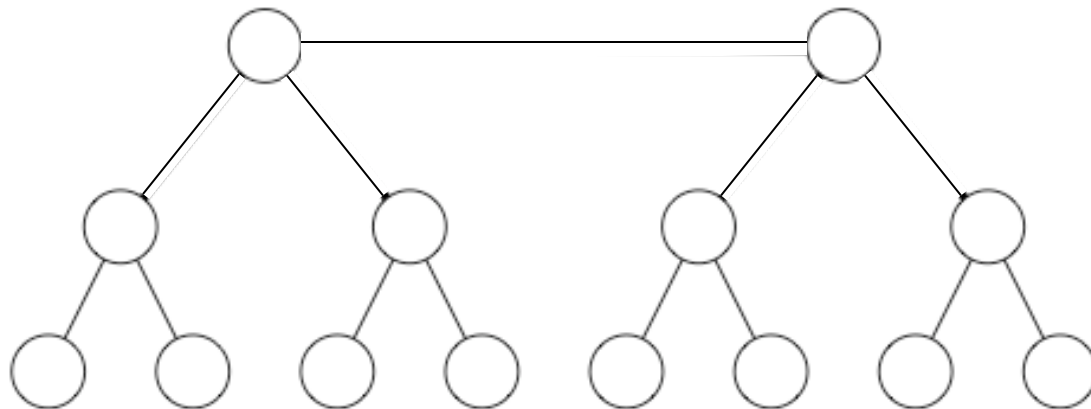
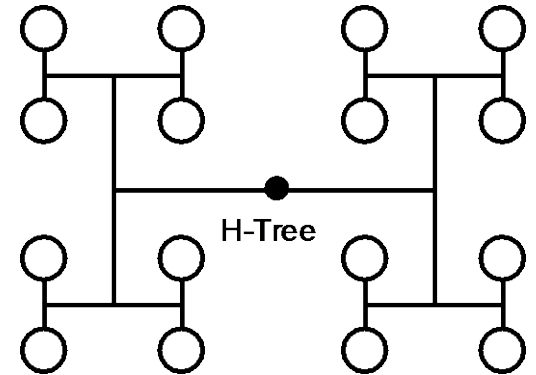
Latency: $O(\log N)$

Good for local traffic

+ Cheap: $O(N)$ cost

+ Easy to Layout

- Root can become a bottleneck



Trees

Planar, hierarchical topology

Latency: $O(\log N)$

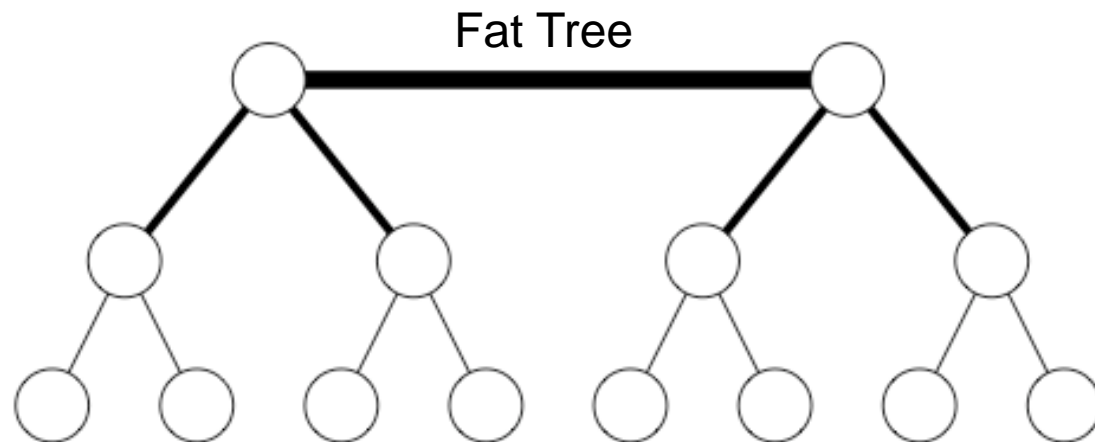
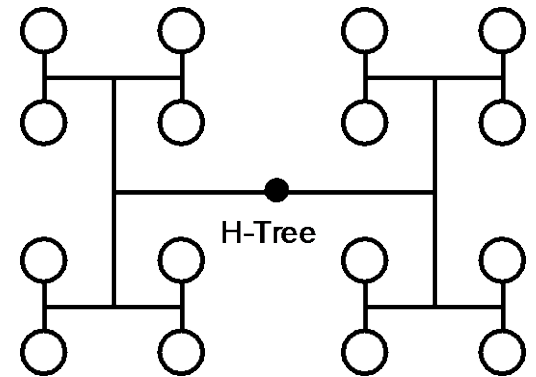
Good for local traffic

+ Cheap: $O(N)$ cost

+ Easy to Layout

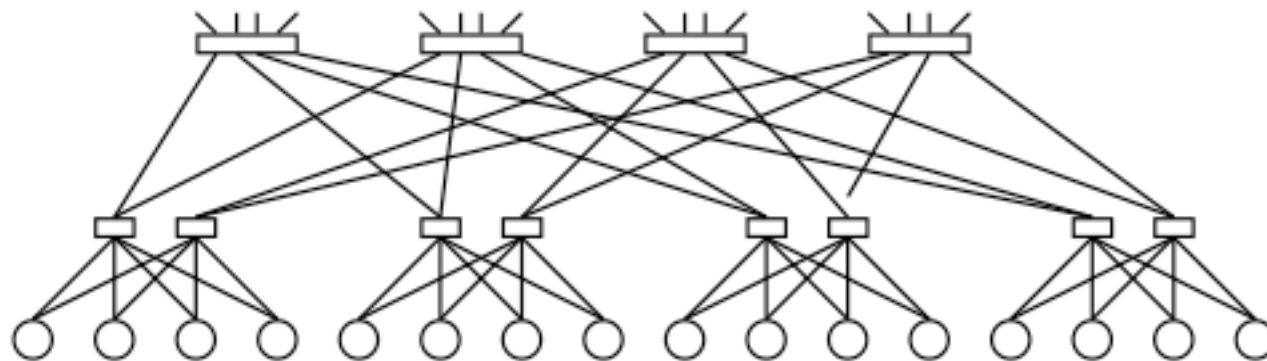
- Root can become a bottleneck

Fat trees avoid this problem (CM-5)



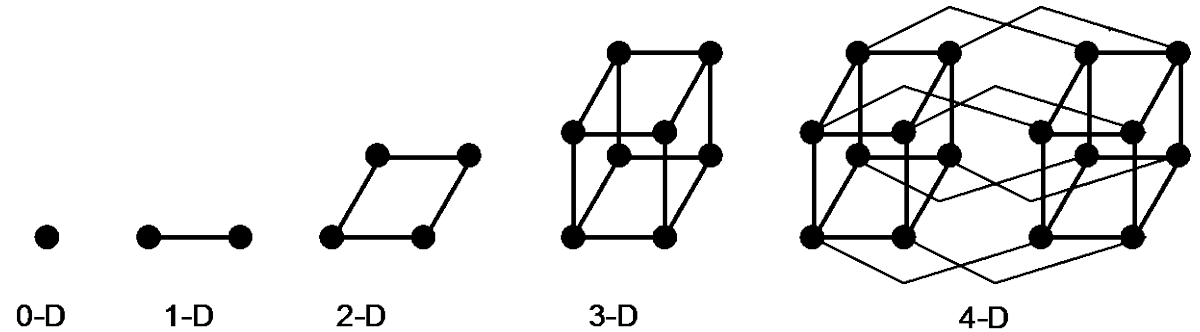
CM-5 Fat Tree

- Fat tree based on 4x2 switches
- Randomized routing on the way up
- Combining, multicast, reduction operators supported in hardware
 - Thinking Machines Corp., “[The Connection Machine CM-5 Technical Summary](#),” Jan. 1992.

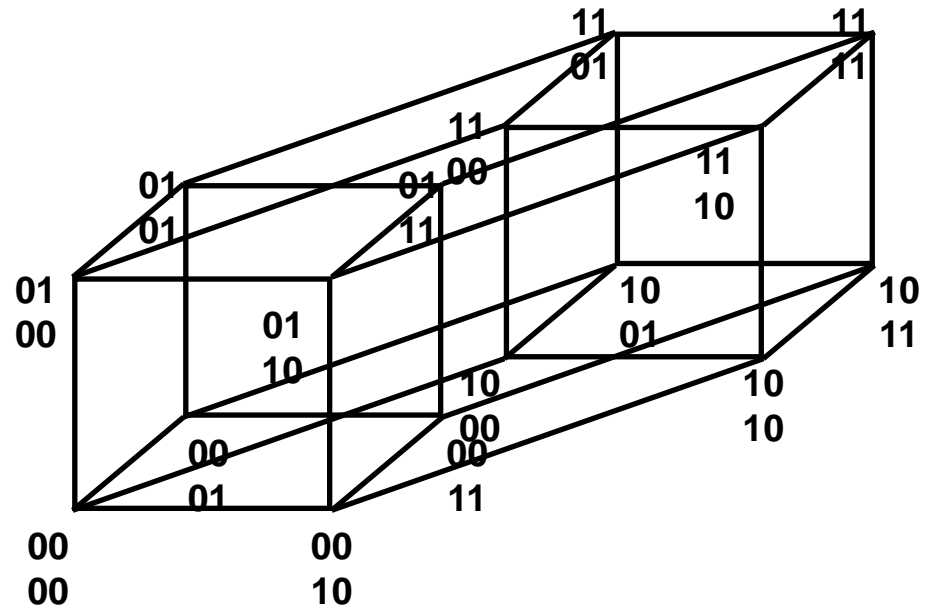


CM-5 Thinned Fat Tree

Hypercube

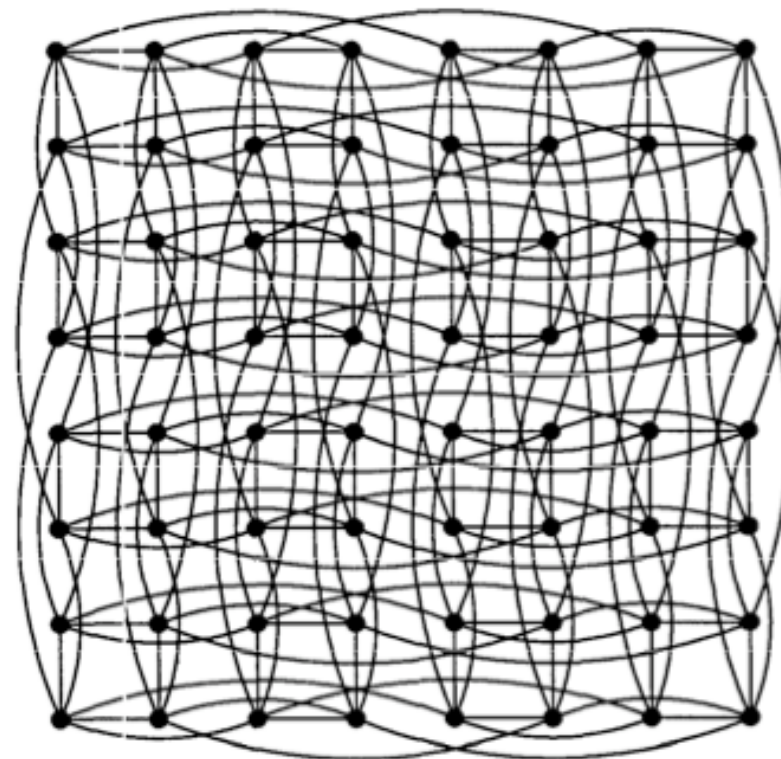
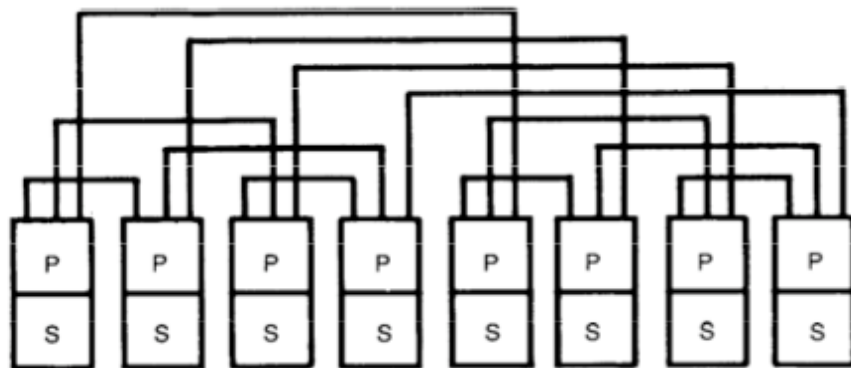


- Latency: $O(\log N)$
- Radix: $O(\log N)$
- #links: $O(N \log N)$
- + Low latency
- Hard to lay out in 2D/3D



Caltech Cosmic Cube

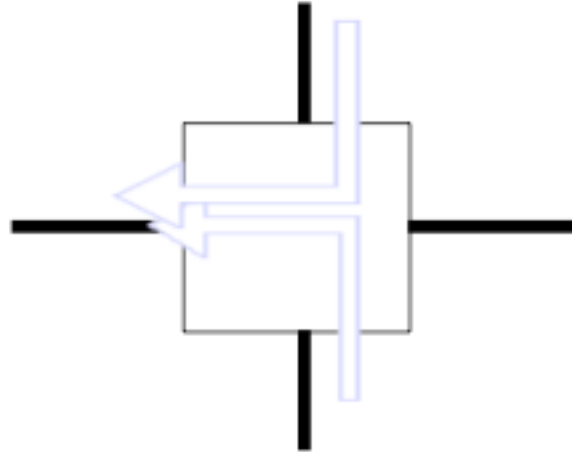
- 64-node message passing machine
- Seitz, “The Cosmic Cube,” CACM 1985.



A hypercube connects $N = 2^n$ small computers, called nodes, through point-to-point communication channels in the Cosmic Cube. Shown here is a two-dimensional projection of a six-dimensional hypercube, or binary 6-cube, which corresponds to a 64-node machine.

FIGURE 1. A Hypercube (also known as a binary cube or a Boolean n -cube)

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

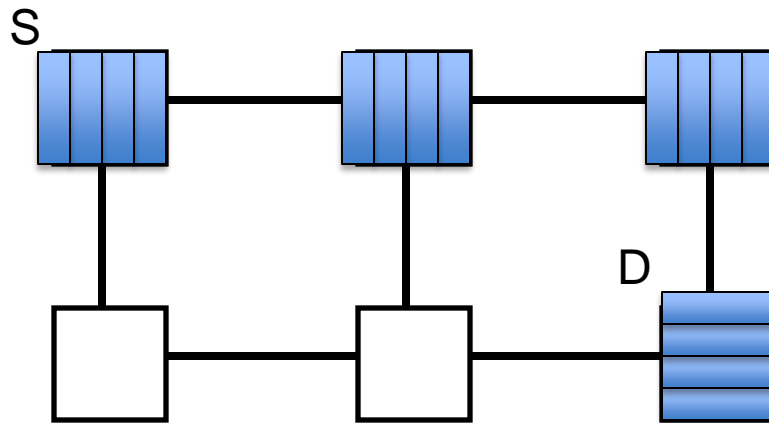
- **Circuit switching**
- **Store and forward** (Packet based)
- **Virtual cut through** (Packet based)
- **Wormhole** (Flit based)

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



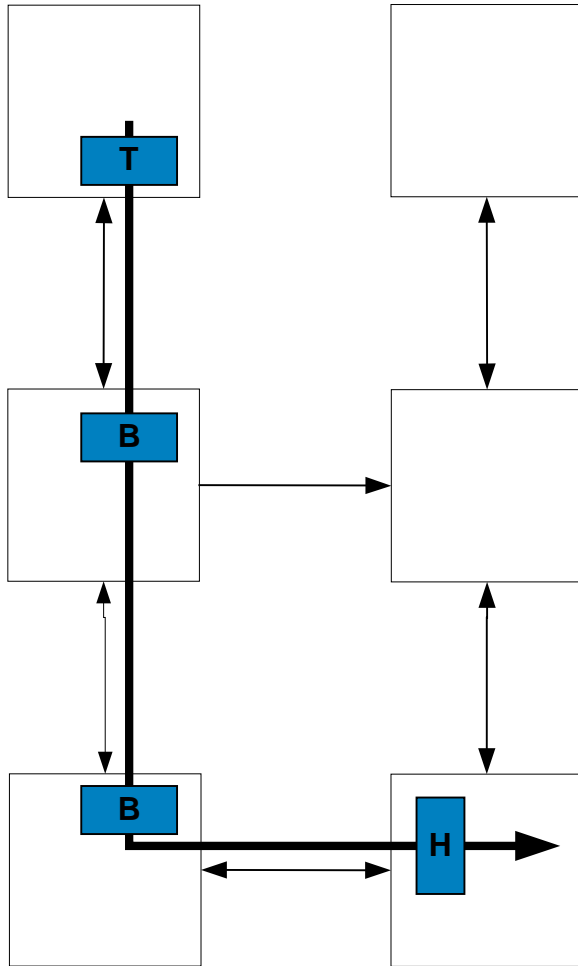
Can we do better?

Cut through Flow Control

- 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

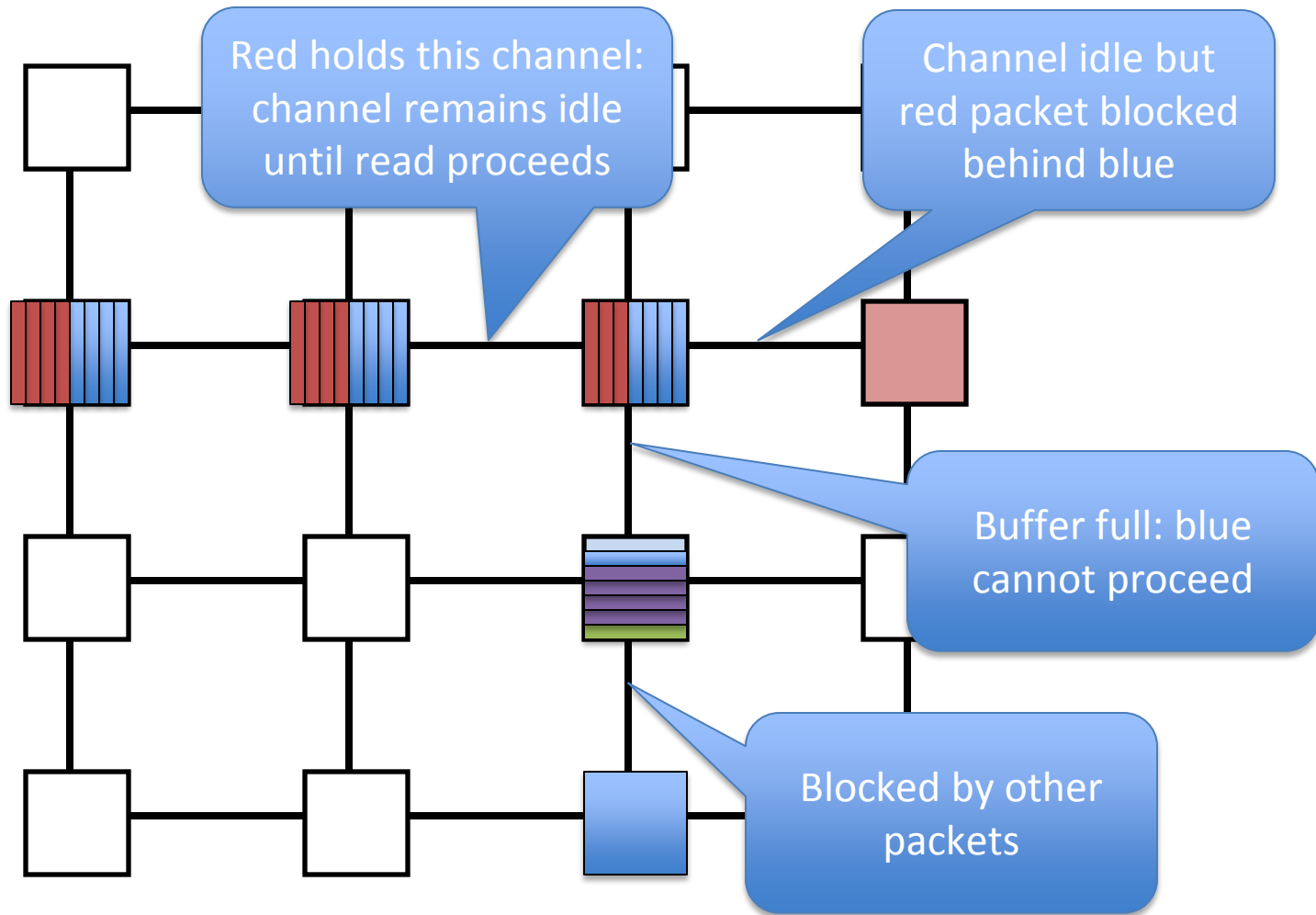
- **Advantages over “store and forward” flow control**

- + Lower latency
- + More efficient buffer utilization

- **Limitations**

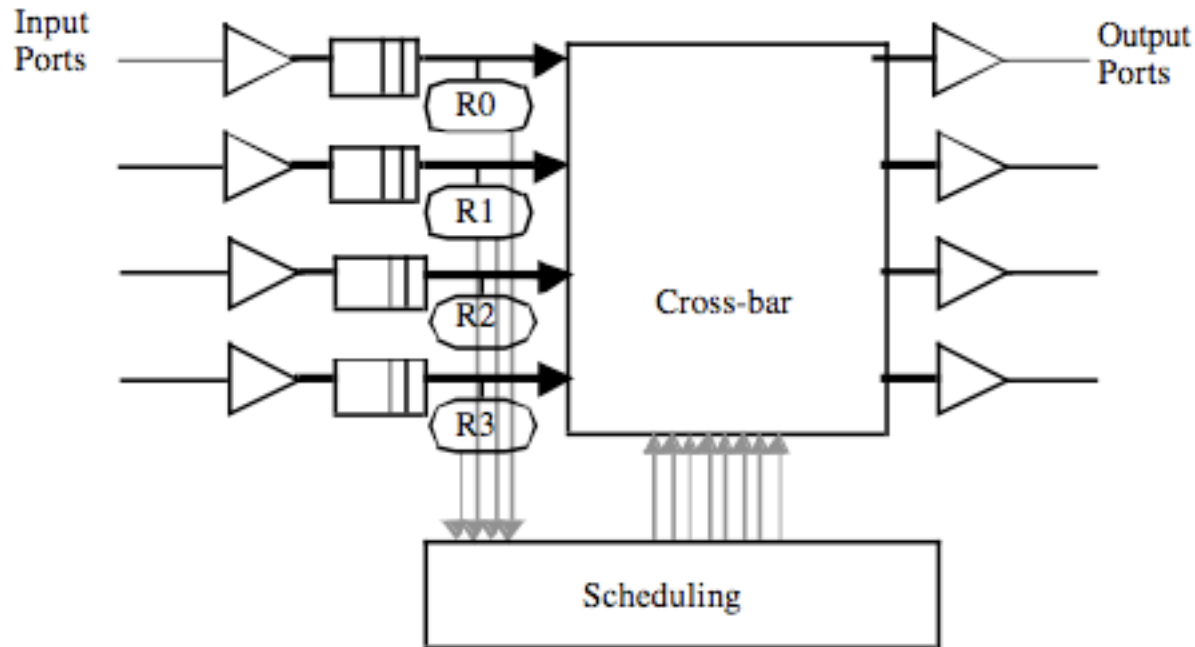
- Occupies resources across multiple routers
- Suffers from **head of line blocking**
 - if head flit cannot move due to contention, another worm cannot proceed even though links may be idle

Head of Line Blocking



Head of Line Blocking

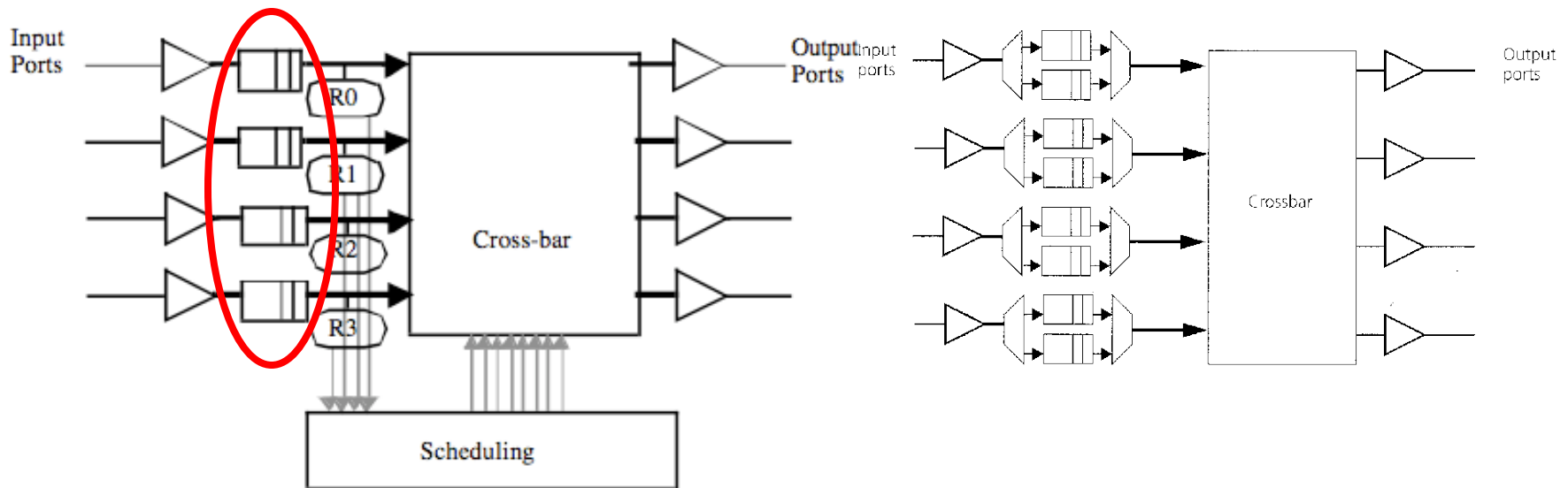
- A worm can be before another in the router input buffer
- Due to FIFO nature, the second worm cannot be scheduled even though it may need to access another output port



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.

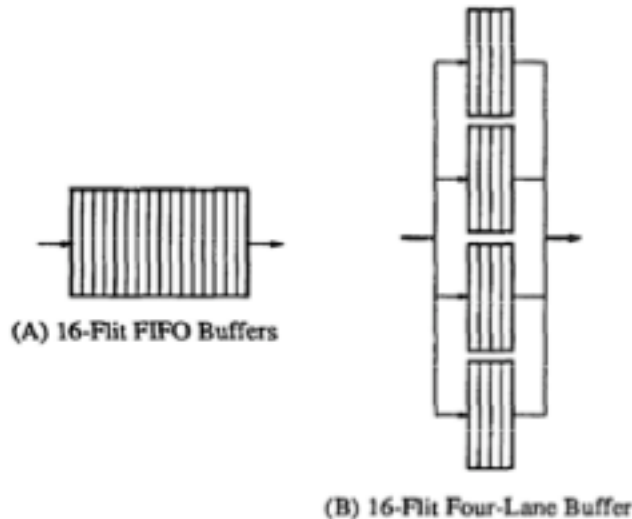
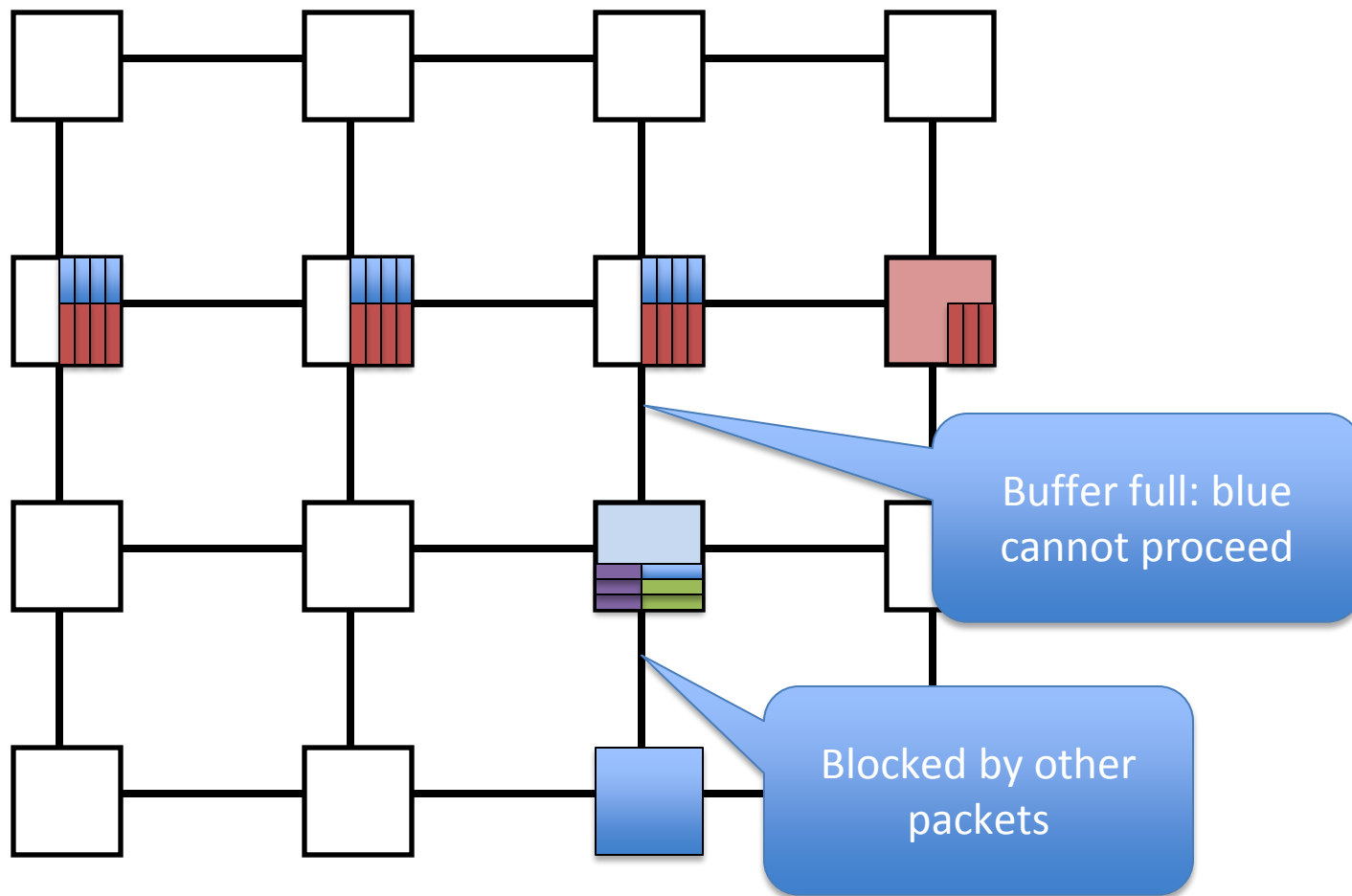
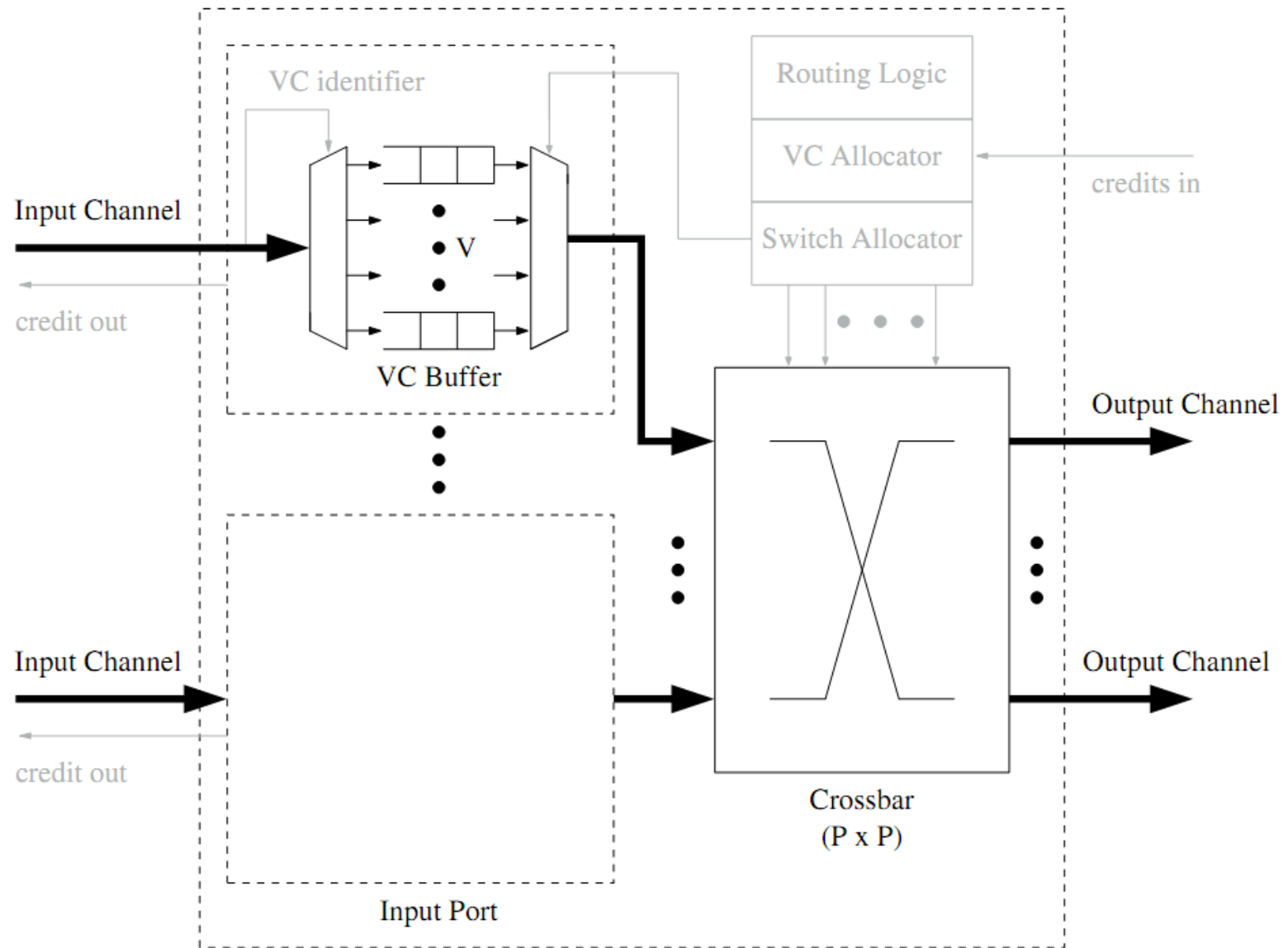


Figure 5: (A) Conventional nodes organize their buffers into FIFO queues restricting routing. (B) A network using virtual-channel flow control organizes its buffers into several independent lanes.

Virtual Channel Flow Control



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

Routing Algorithm

- Types
 - **Deterministic**: always chooses the same path for a communicating source-destination pair
 - **Oblivious**: chooses different paths, without considering network state
 - **Adaptive**: can choose different paths, adapting to the state of the network

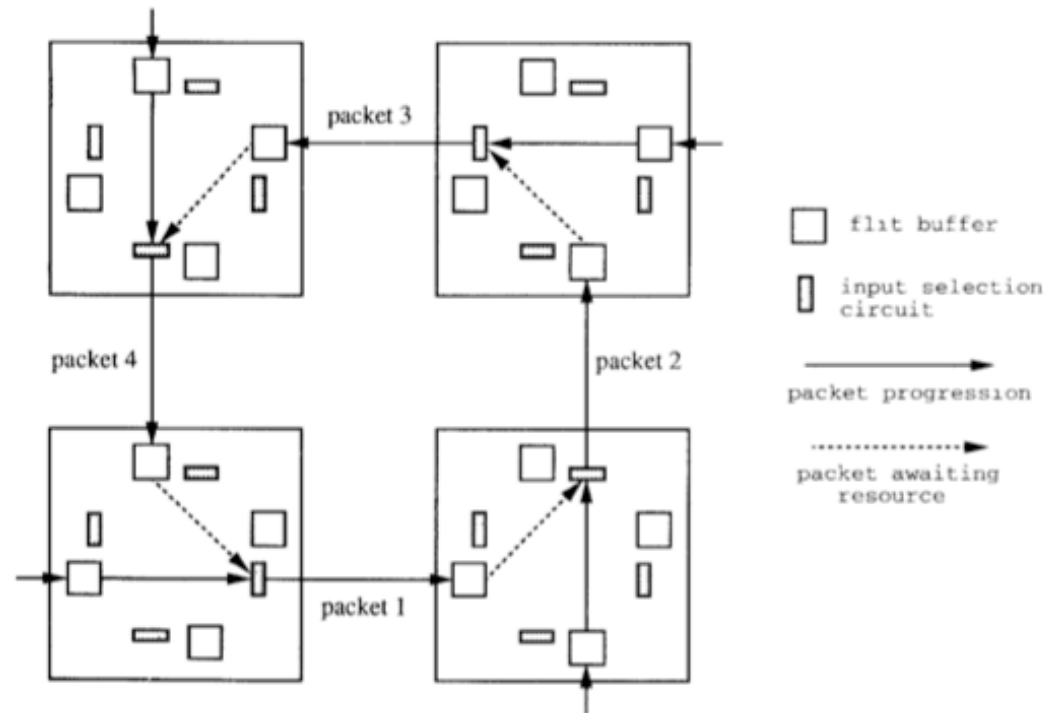
- How to adapt
 - Local/global feedback
 - Minimal or non-minimal paths

Deterministic Routing

- All packets between the same (source, dest) pair take the same path
 - Dimension-order routing
 - E.g., XY routing (used in Cray T3D, and many on-chip networks)
 - First traverse dimension X, then traverse dimension Y
- + Simple
- + Deadlock freedom (no cycles in resource allocation)
- Could lead to high contention
- Does not exploit path diversity

Deadlock

- No forward progress
- Caused by circular dependencies on resources
- Each packet waits for a buffer occupied by another packet downstream



Handling Deadlock

- Avoid cycles in routing
 - Dimension order routing
 - Cannot build a circular dependency
 - Restrict the “turns” each packet can take

- Avoid deadlock by adding more buffering (escape paths)

- Detect and break deadlock
 - Preemption of buffers

Turn Model to Avoid Deadlock

■ Idea

- Analyze directions in which packets can turn in the network
- Determine the cycles that such turns can form
- Prohibit just enough turns to break possible cycles

- Glass and Ni, “[The Turn Model for Adaptive Routing](#),” ISCA 1992.

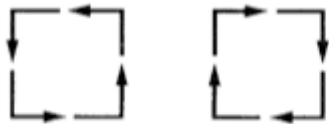


FIG. 2. The possible turns and simple cycles in a two-dimensional mesh.



FIG. 3. The four turns allowed by the xy routing algorithm.

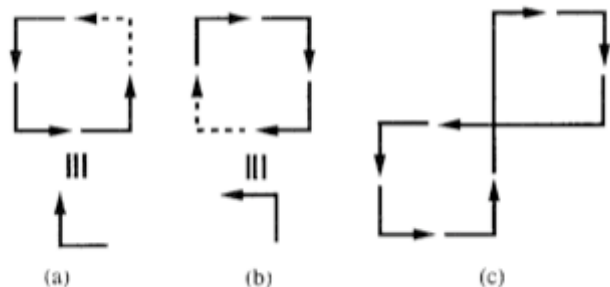


FIG. 4. Six turns that complete the cycles and allow deadlock.

Oblivious Routing: Valiant's Algorithm

- An example of oblivious algorithm
 - Goal: Balance network load
 - Idea: Randomly choose an intermediate destination, route to it first, then route from there to destination
 - Between source-intermediate and intermediate-dest, can use dimension order routing
- + Randomizes/balances network load
- Non minimal (packet latency can increase)
- Optimizations:
 - Do this on high load
 - Restrict the intermediate node to be close (in the same quadrant)

Adaptive Routing

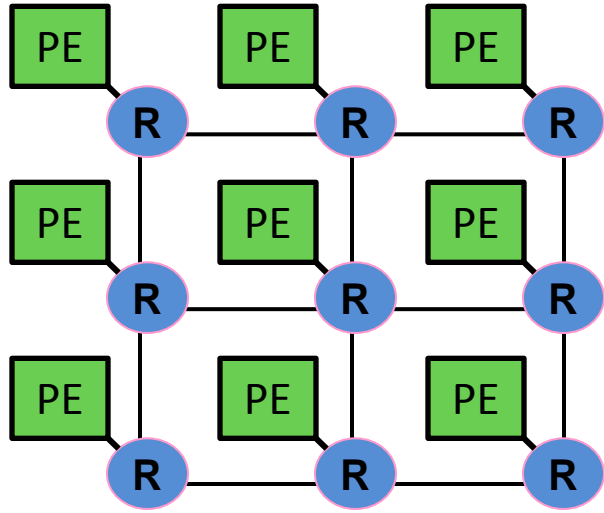
■ Minimal adaptive

- Router uses network state (e.g., downstream buffer occupancy) to pick which “productive” output port to send a packet to
 - Productive output port: port that gets the packet closer to its destination
- + Aware of local congestion
- Minimality restricts achievable link utilization (load balance)

■ Non-minimal (fully) adaptive

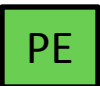
- “Misroute” packets to non-productive output ports based on network state
- + Can achieve better network utilization and load balance
- Need to guarantee livelock freedom

On-Chip Networks

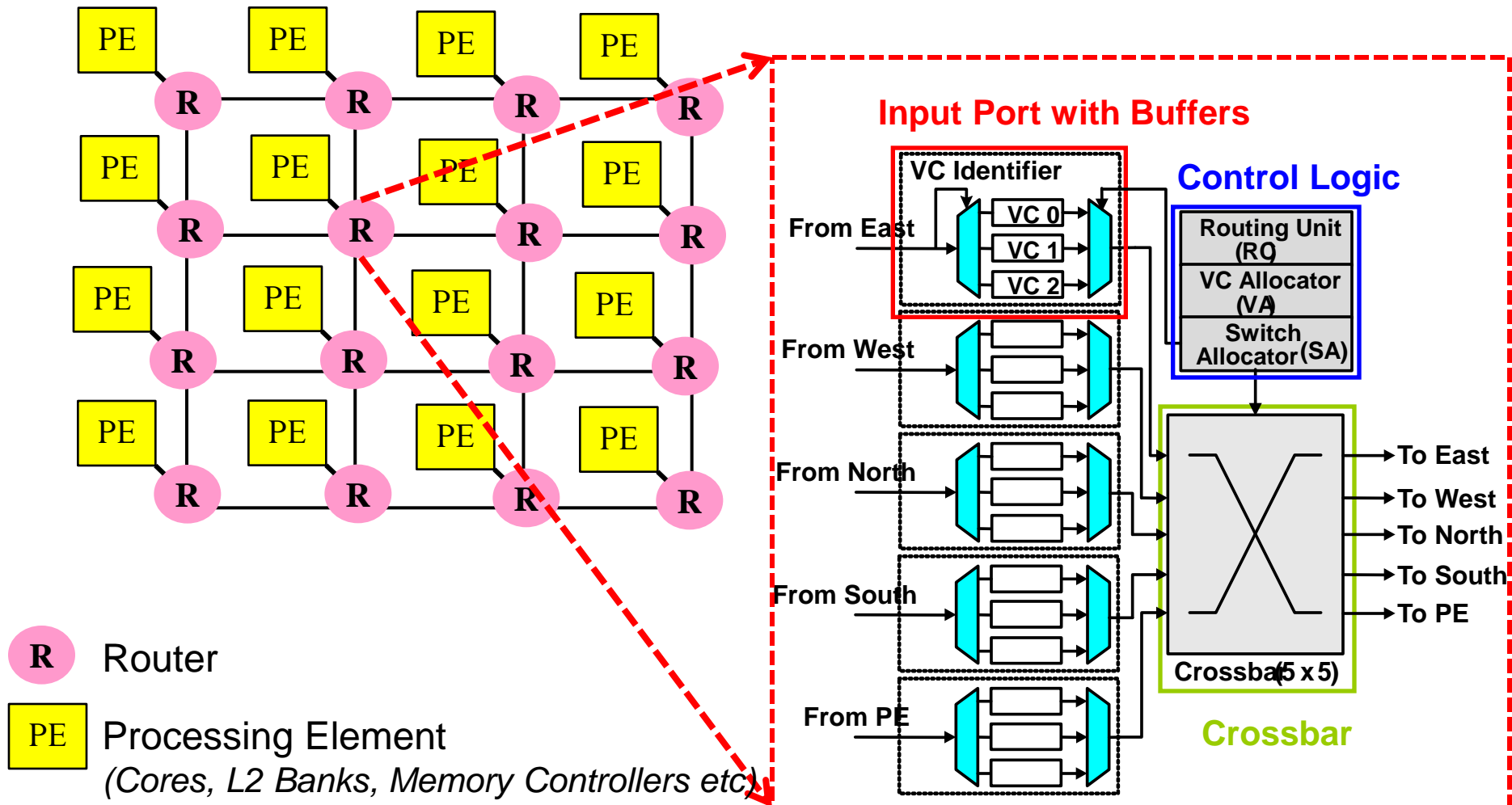


- Connect **cores, caches, memory controllers, etc**
 - Buses and crossbars are not scalable
- **Packet switched**
- **2D mesh:** Most commonly used topology
- Primarily serve **cache misses** and **memory requests**

 Router

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

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

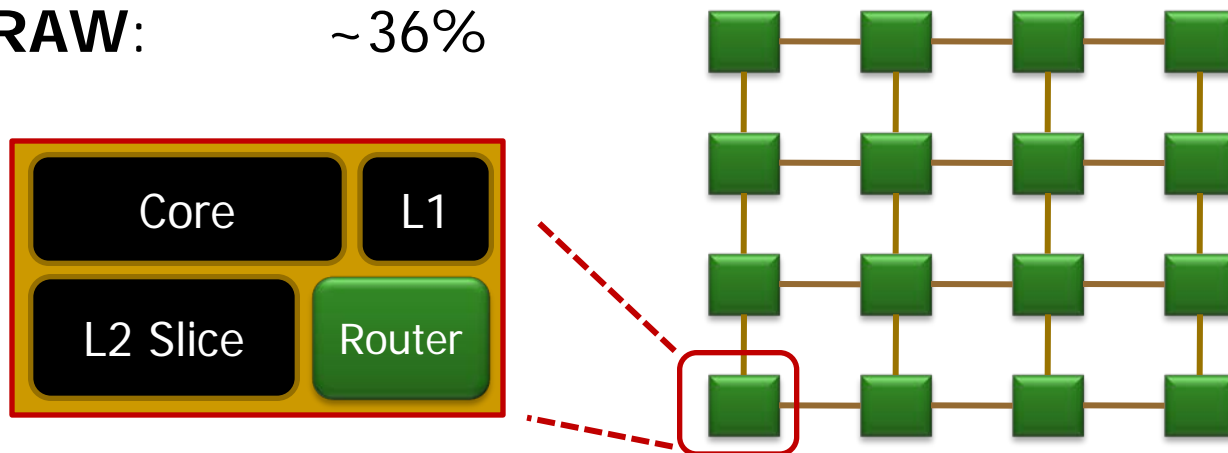
Motivation for Efficient Interconnect

- In many-core chips, on-chip interconnect (NoC) consumes **significant power**

Intel Terascale: ~28% of chip power

Intel SCC: ~10%

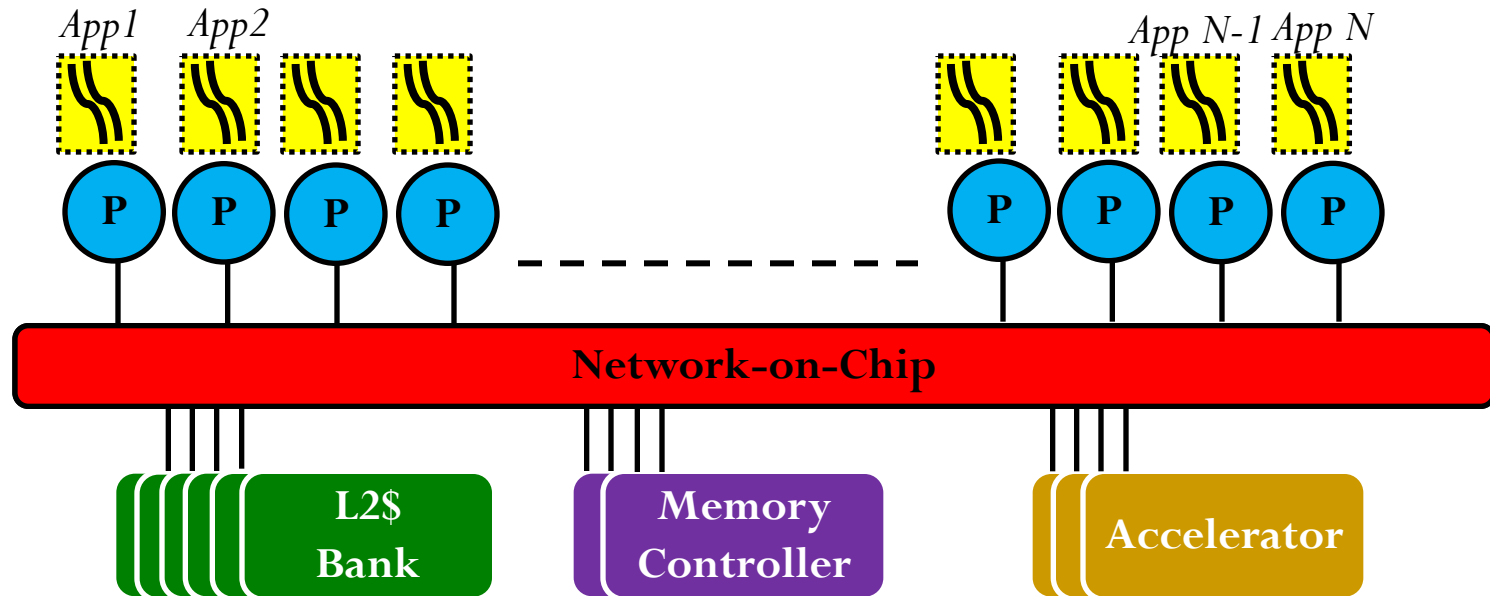
MIT RAW: ~36%



Packet Scheduling in Multicore?

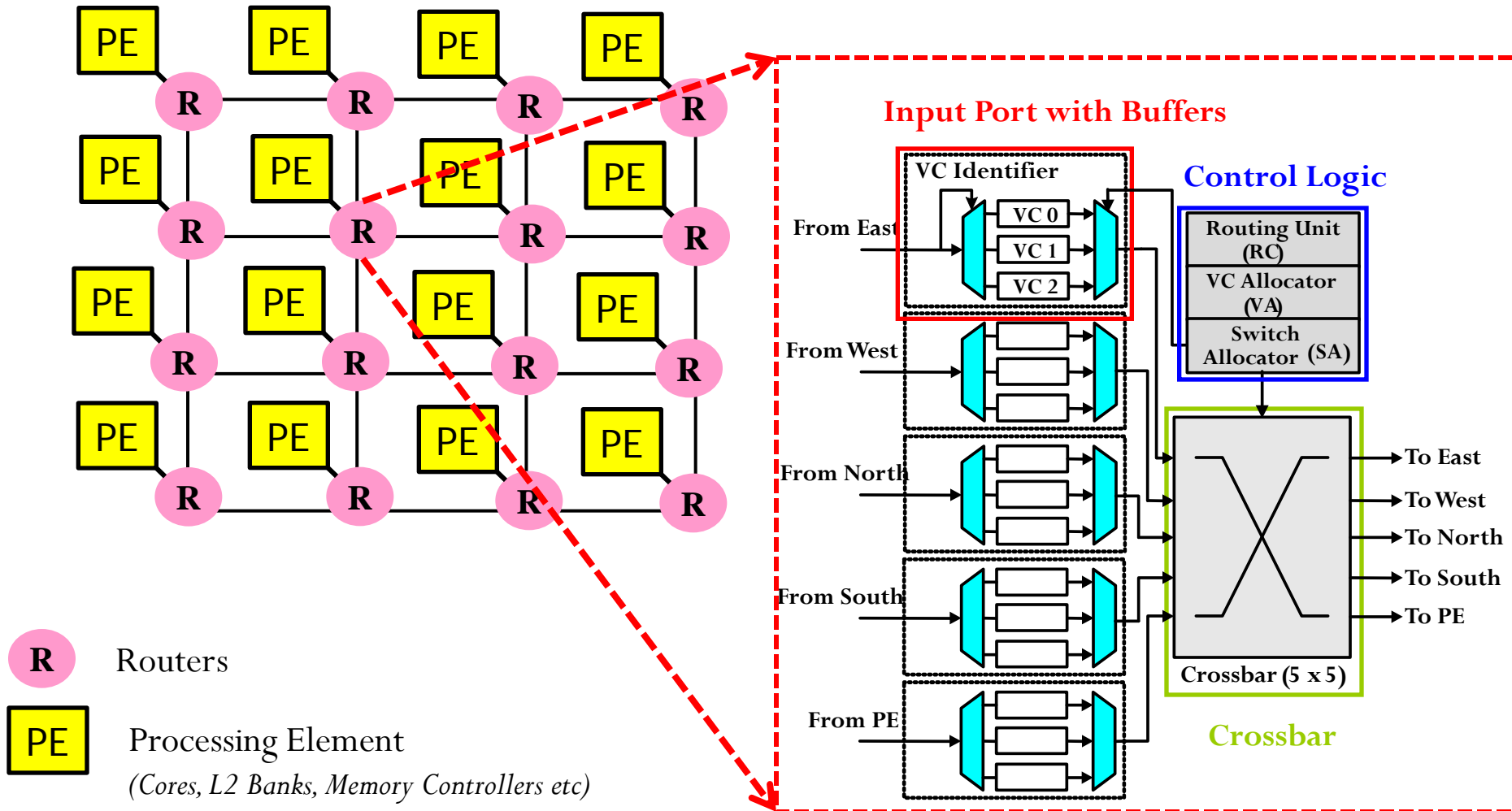
- Which packet to choose for a given output port?
 - ❑ Router needs to prioritize between competing flits
 - ❑ Which input port?
 - ❑ Which virtual channel?
 - ❑ Which application's packet?
- Common strategies
 - ❑ Round robin across virtual channels
 - ❑ Oldest packet first (or an approximation)
 -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

The Problem: Packet Scheduling

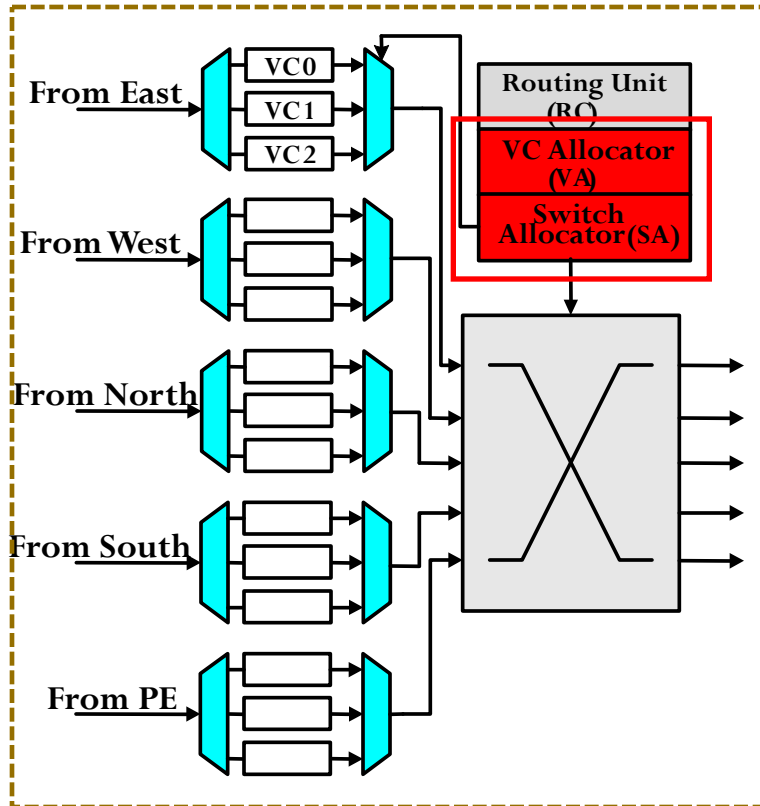


Network-on-Chip is a **critical** resource
shared by **multiple applications**

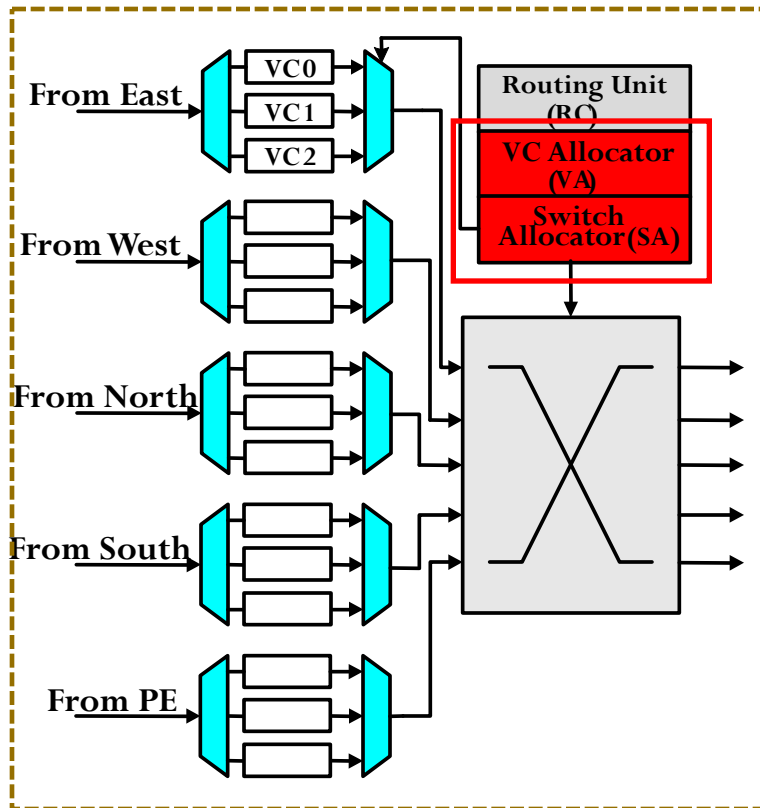
The Problem: Packet Scheduling



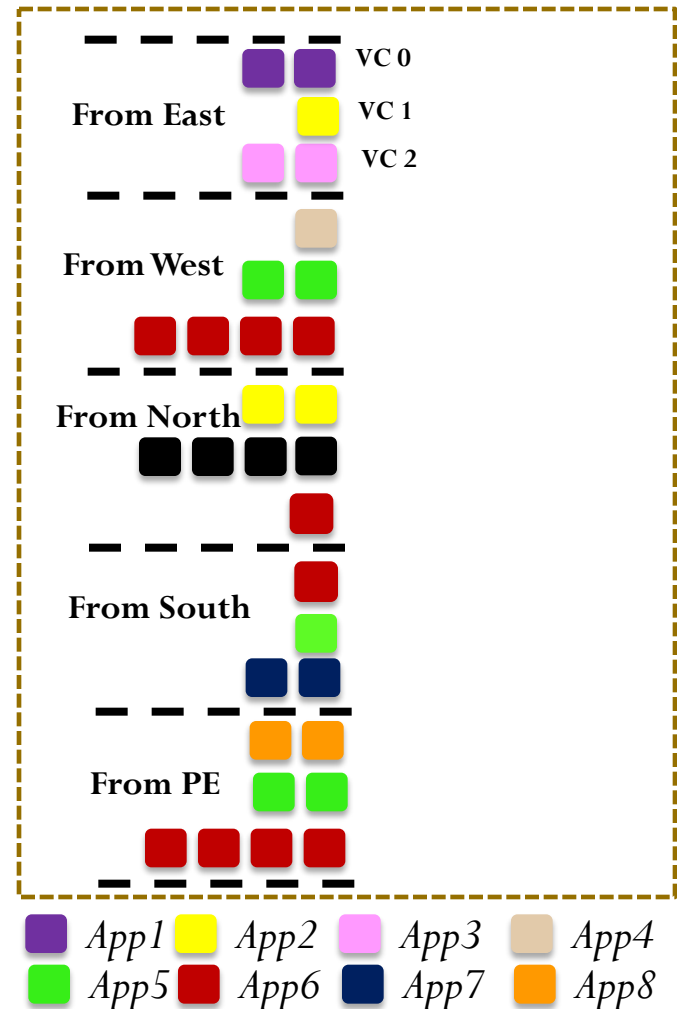
The Problem: Packet Scheduling



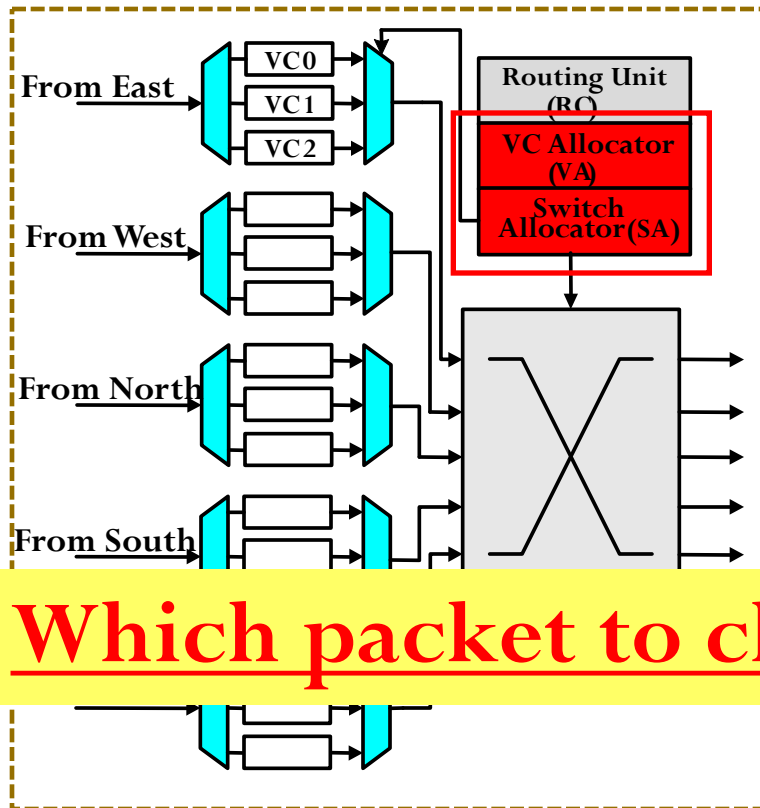
The Problem: Packet Scheduling



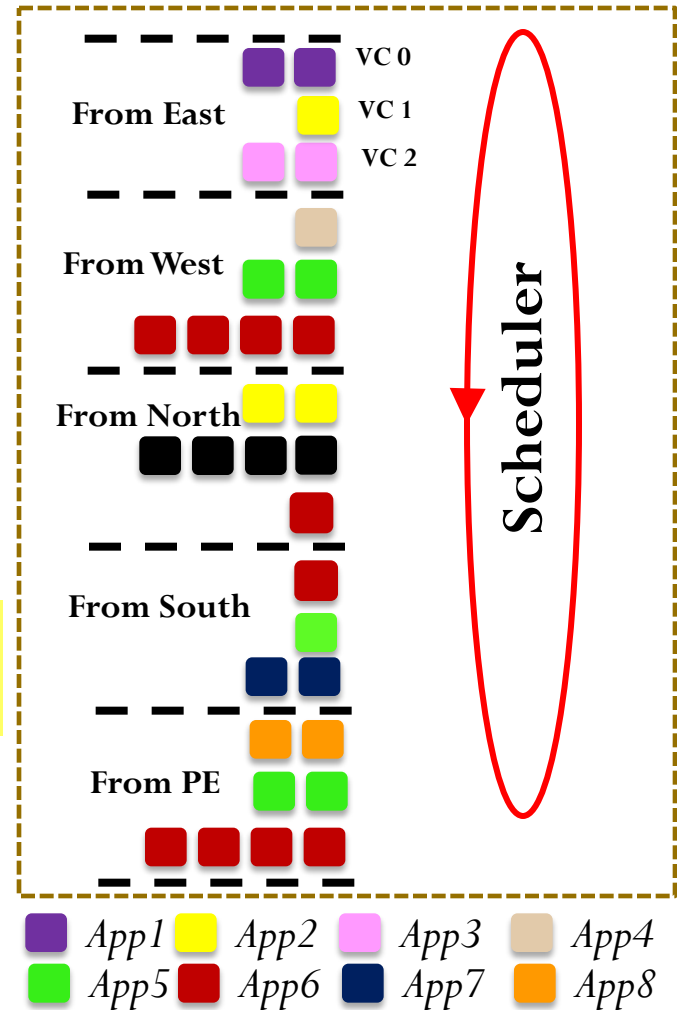
Conceptual
View



The Problem: Packet Scheduling



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.

Das, Mutlu, Moscibroda, and Das, "**Application-Aware Prioritization Mechanisms for On-Chip Networks,**" **MICRO 2009**

Motivation: Stall-Time Criticality

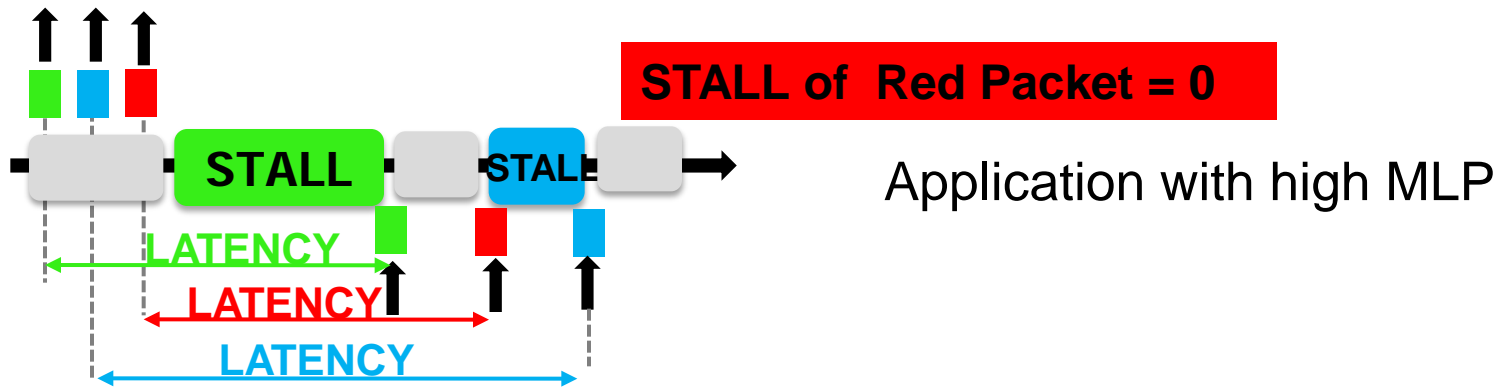
- Applications are **not homogenous**
- Applications have different **criticality** with respect to the **network**
 - Some applications are network latency sensitive
 - Some applications are network latency tolerant
- Application's **Stall Time Criticality (STC)** can be measured by its average network 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

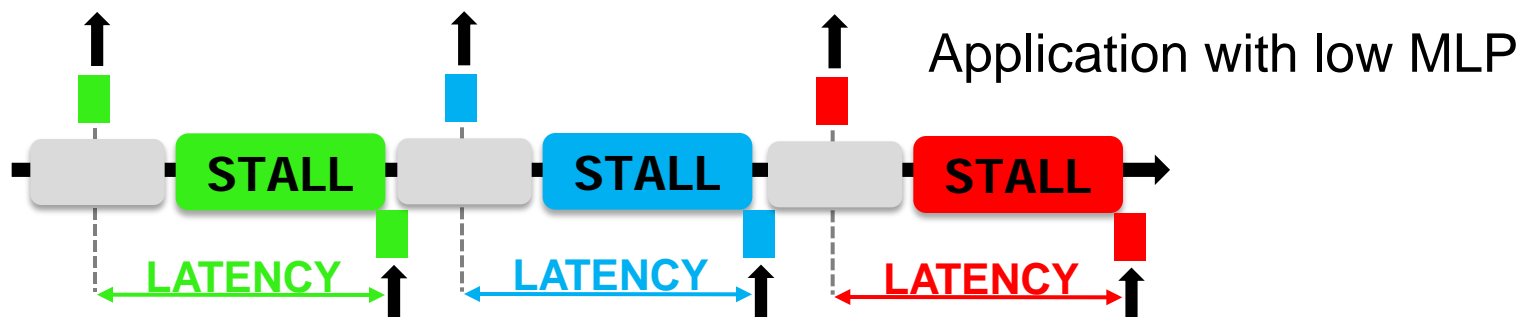
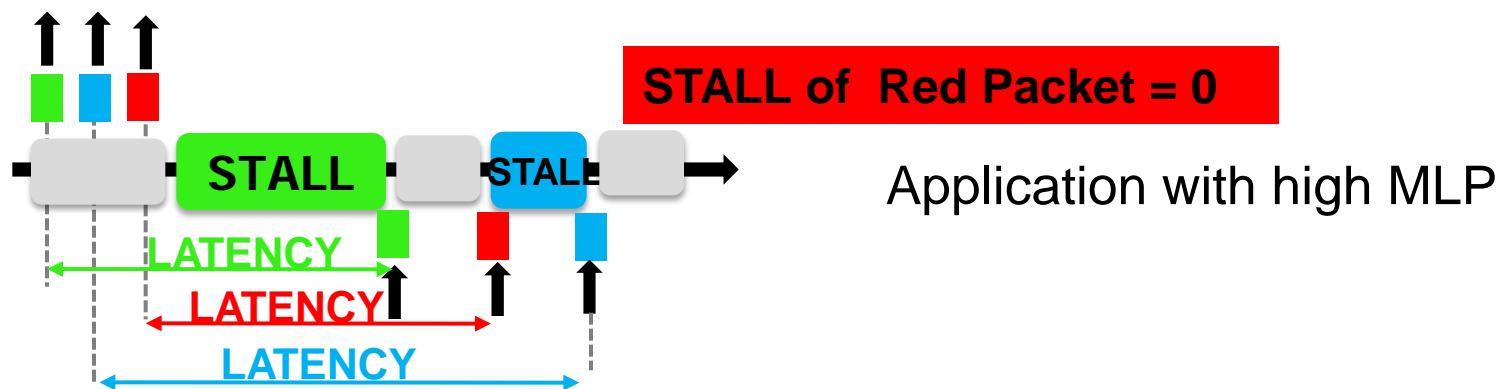
■ Compute



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

STC Principle 1: MLP

■ Compute



- Observation 1: **Packet Latency \neq 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



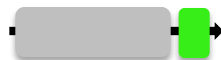
Baseline (RR) Scheduling



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

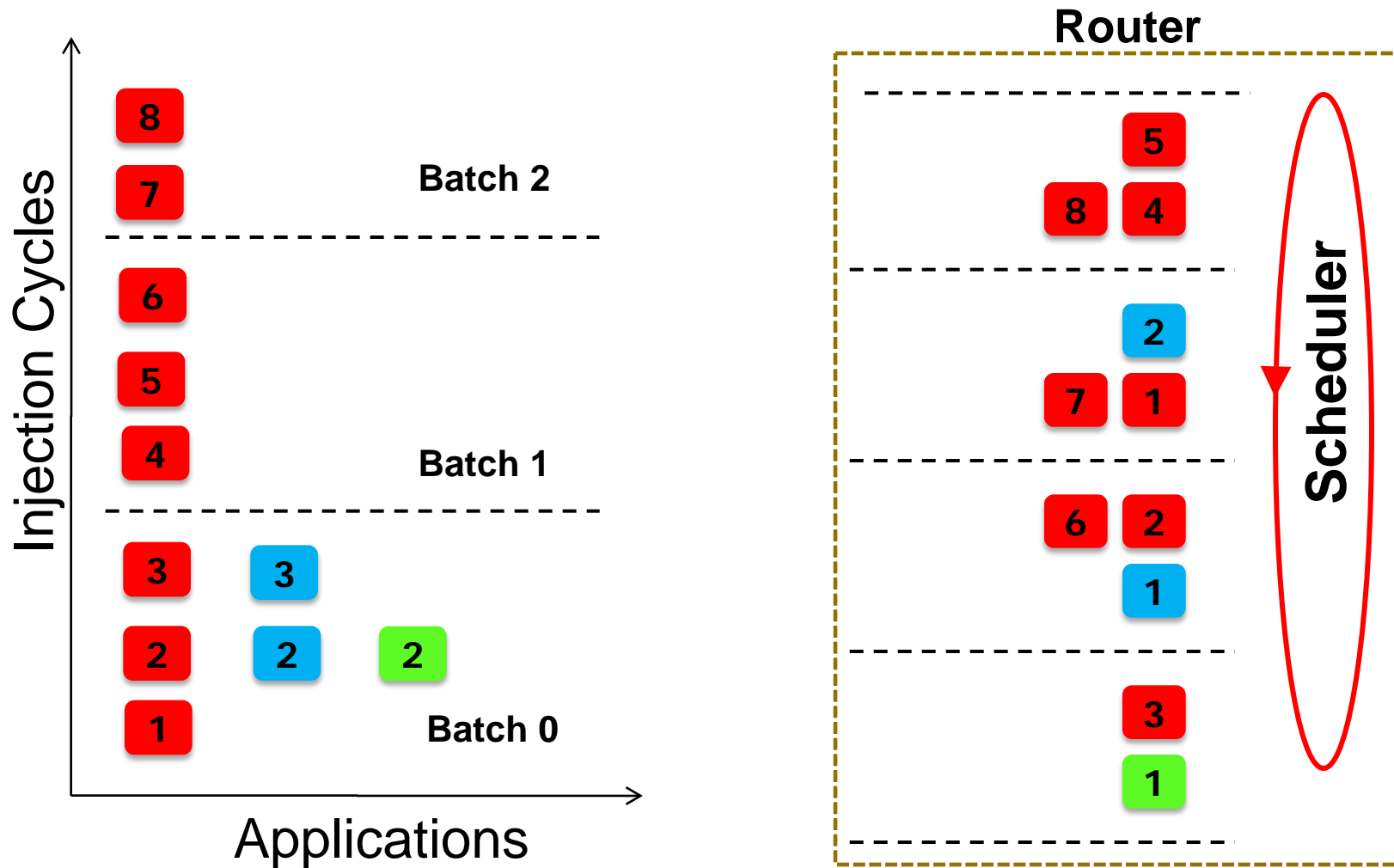
Component 2: Batching

- 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

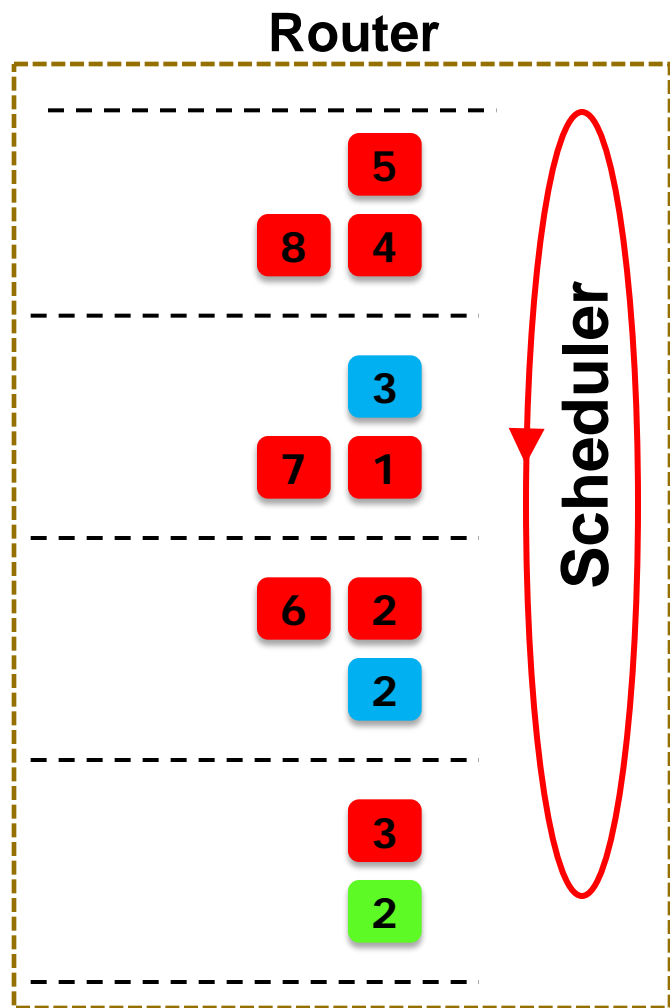
Putting it all together: STC Scheduling Policy

- 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

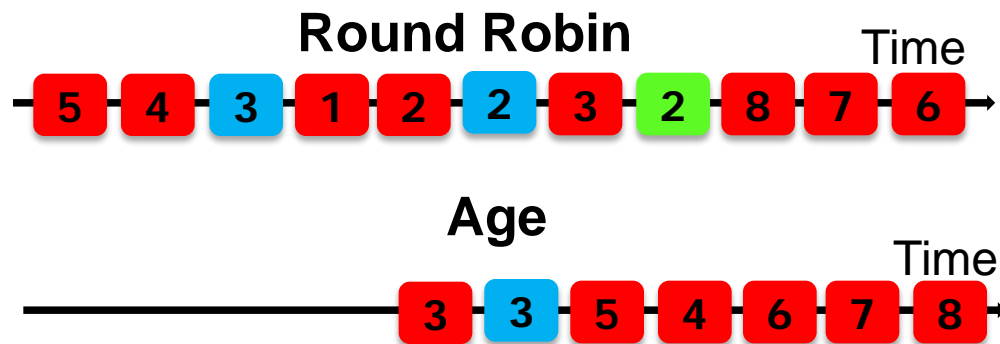
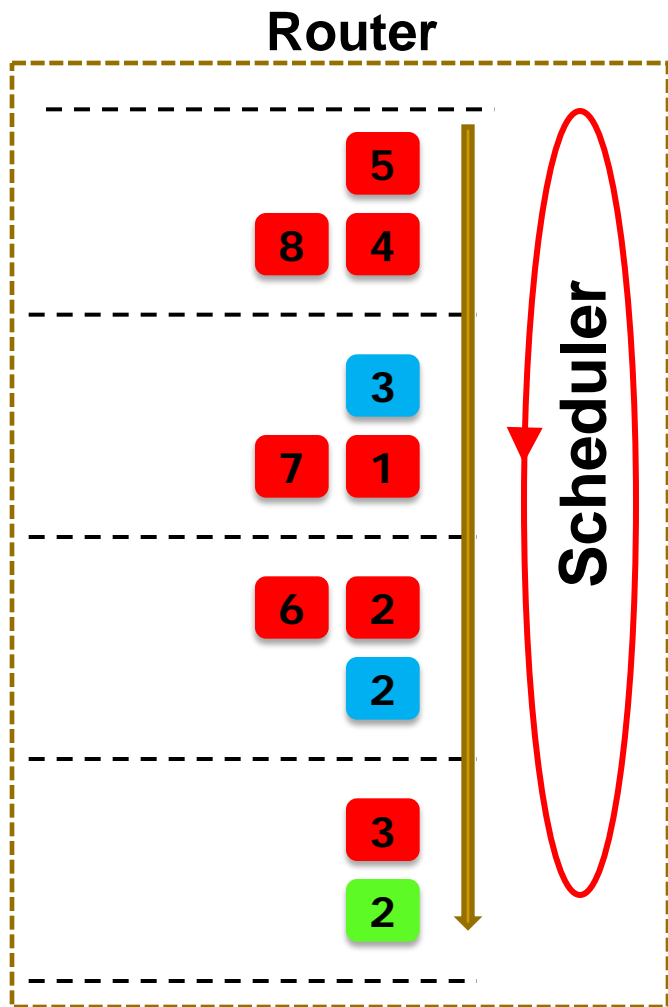


STC Scheduling Example



	STALL CYCLES			Avg
RR	8	6	11	8.3
Age				
STC				

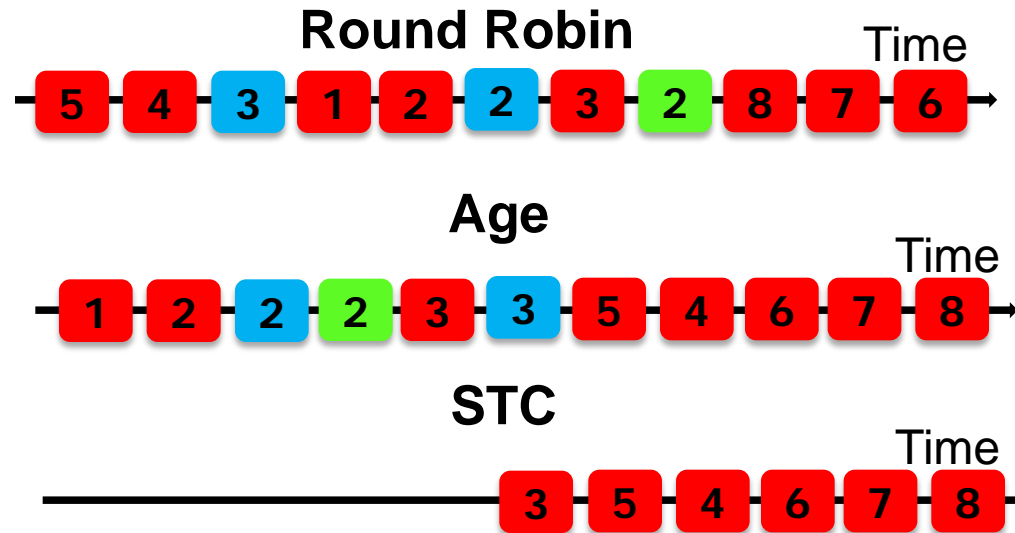
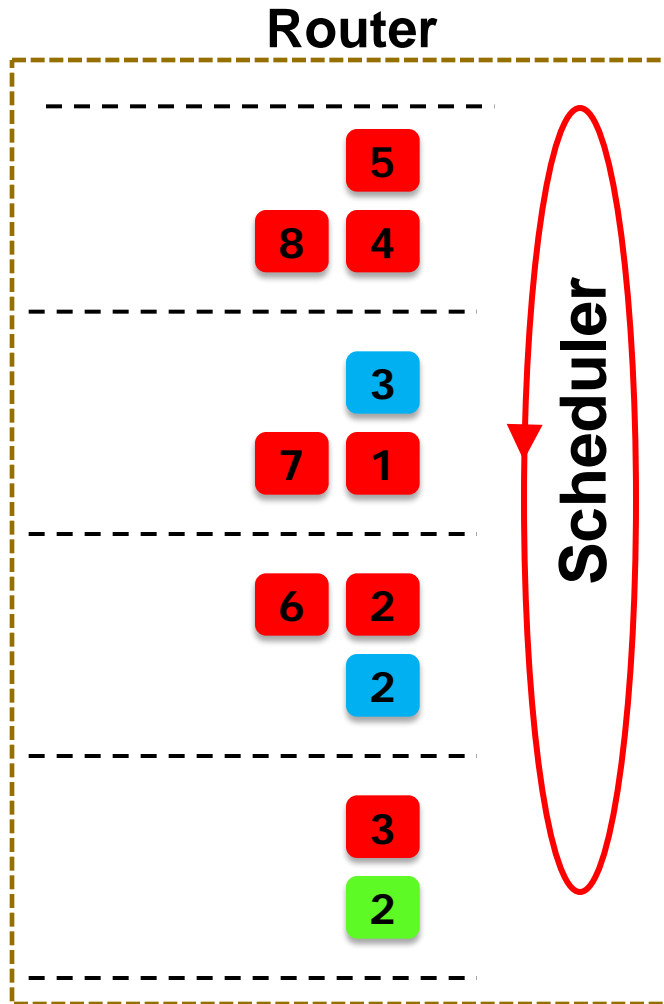
STC Scheduling Example



	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

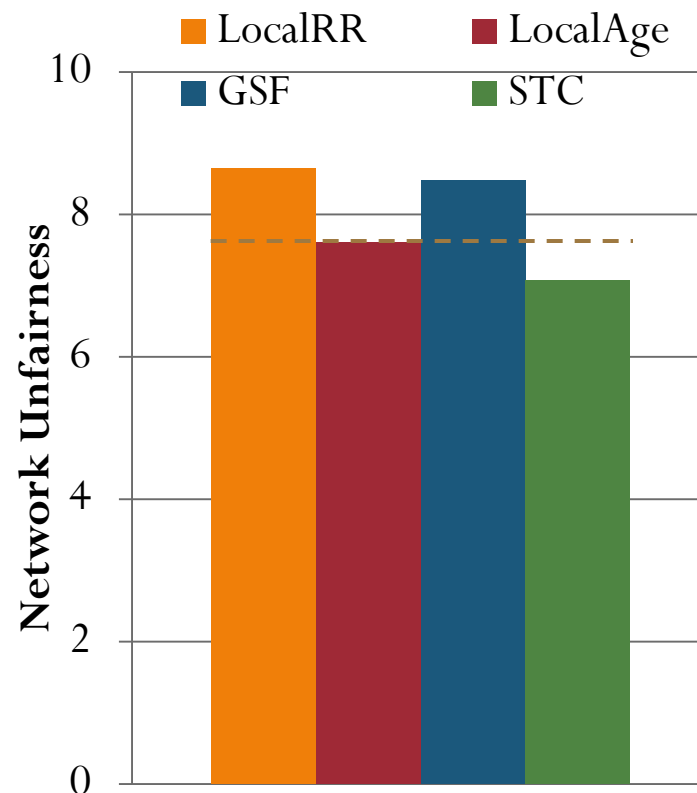
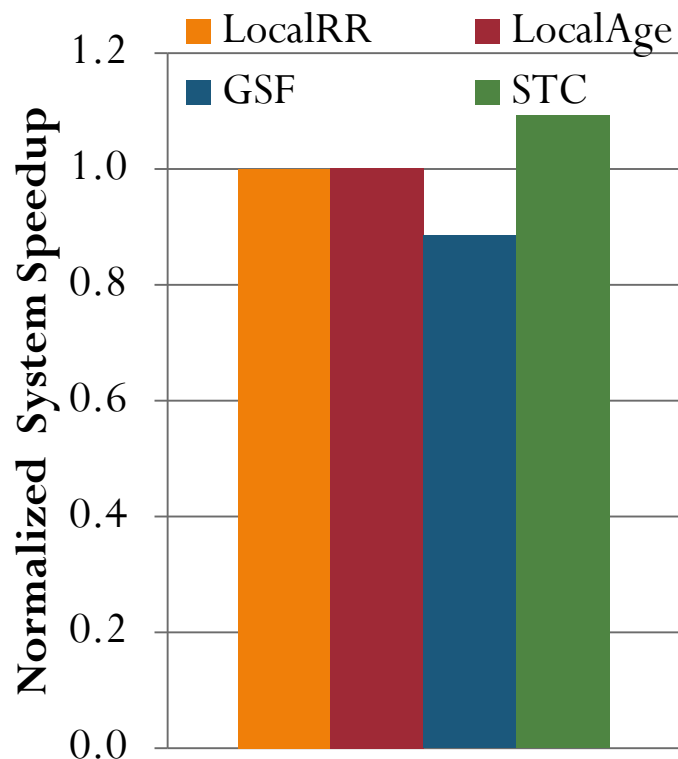
■ Round Robin & Age (Oldest-First)

- ❑ 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)

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