Interconnection Networks

Nima Honarmand

(Largely based on slides by Prof. Thomas Wenisch, University of Michigan)

Interconnection Networks

- What holds our parallel machines together - at the core of parallel computer architecture
- Shares basic concept with LAN/WAN, but very different trade-offs due to very different time scale/requirements

Different Scales of Networks (1/3)

- On-Chip Networks
  - Interconnect within a single chip
- Devices are micro-architectural elements: caches, directories, processor cores
- Currently, designs with 10s of devices are common
  - Ex: IBM Cell, Intel multicores, Tile processors
- Projected systems with 100s of devices on the horizon
- Proximity: millimeters

Different Scales of Networks (2/3)

- System-Area Networks
  - Interconnects within one “machine”
    - Interconnect in a multi-processor system
    - Interconnect in a supercomputer
- Hundreds to thousands of devices interconnected
  - IBM Blue Gene/L supercomputer (64K nodes, each with 2 processors)
- Maximum interconnect distance
  - Fraction to tens of meters (typical)
  - a few hundred meters (some)
    - InfiniBand: 120 Gbps over a distance of 300m

Different Scales of Networks (3/3)

- Local-Area Networks
  - Interconnect autonomous computer systems
  - Machine room or throughout a building or campus
  - Hundreds of devices interconnected (1,000s with bridging)
  - Maximum interconnect distance
    - few metres to tens of kilometers
  - Example (most popular): Ethernet, with 10 Gbps over 40Km
- Wide-Area Networks
  - Interconnect systems distributed across the globe
  - Internetworking support is required
  - Many millions of devices interconnected
  - Maximum interconnect distance
    - many thousands of kilometers

We are concerned with On-Chip and System-Area Networks

ICN Design Considerations (1/3)

- Application requirements
  - Number of terminals or ports to support
  - Peak bandwidth of each terminal
  - Average bandwidth of each terminal
  - Latency requirements
  - Message size distribution
  - Expected traffic patterns
  - Required quality of service
  - Required reliability and availability
- Job of an interconnection network is to transfer information from source node to dest. node in support of network transactions that realize the application
  - latency as small as possible
  - as many concurrent transfers as possible
  - cost as low as possible

ICN Design Considerations (2/3)

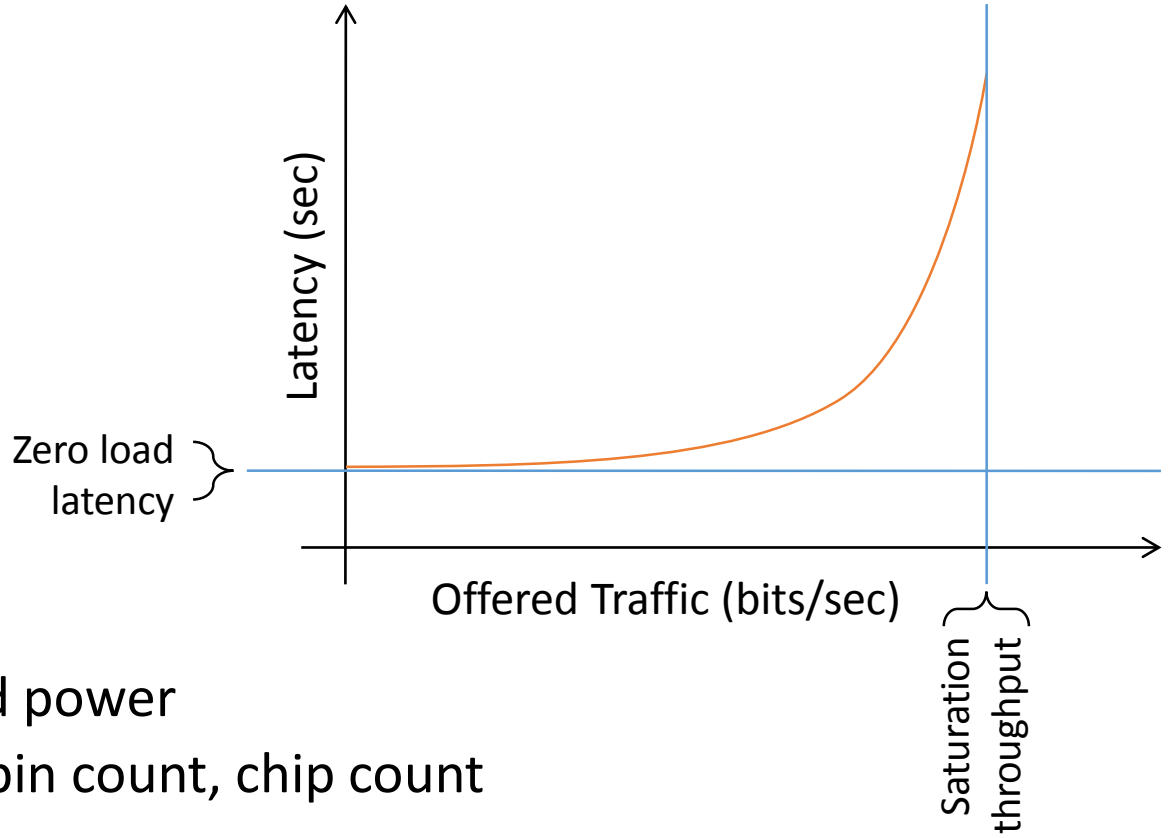
- Example requirements for a coherent processor-memory interconnect
  - Processor ports 1-2048
  - Memory ports 1-4096
  - Peak BW 8 GB/s
  - Average BW 400 MB/s
  - Message Latency 100 ns
  - Message size 64 or 576 bits
  - Traffic pattern arbitrary
  - Quality of service none
  - Reliability no message loss
  - Availability 0.999 to 0.99999

ICN Design Considerations (3/3)

- Technology constraints
  - Signaling rate
  - Chip pin count (if off-chip networking)
  - Area constraints (typically for on-chip networking)
  - Chip cost
  - Circuit board cost (if backplane boards needed)
  - Signals per circuit board
  - Signals per cable
  - Cable cost
  - Cable length
  - Channel and switch power constraints
  - …

Performance and Cost

- Main Performance figures: latency and throughput



- Main cost factors
  - On-Chip: area and power
  - Off-Chip: wiring, pin count, chip count

Basic Definitions

- An interconnection network is a graph of ***nodes*** inter-connected using ***channels***

- ***Node***: a vertex in the network graph
  - ***Terminal*** nodes: where messages originate and terminate
  - ***Switch*** (***router***) nodes: forward messages from in ports to out ports
  - ***Switch degree***: number of in/out ports per switch

- ***Channel***: an edge in the graph
  - *i.e.*, an ordered pair (x,y) where x and y are nodes
  - Channel = link (transmission medium) + transmitter + receiver
  - ***Channel width***: w = number of bits transferred per cycle
  - ***Phit*** (physical unit or digit): data transferred per cycle
  - ***Signaling rate***: f = number of transfer cycles per second
  - ***Channel bandwidth***: $b = w \times f$

Basic Definitions

- ***Path*** (or ***route***): a sequence of channels, connecting a ***source*** node to a ***destination*** node
- ***Minimal Path***: a path with the minimum number of channels between a source and a destination
  - R_{xy} = set of all minimal paths from x to y
- ***Network Diameter***: Longest minimal path over all (source, destination) pairs

Basic Communication Latency

- $\text{Time}(n)_{src\text{-}dest}$ = overhead + routing delay + channel occupancy + contention delay
- occupancy = $(n + n_e) / b$
  - n = size of data
  - n_e= size of packet overhead
  - b = channel bandwidth
- Routing delay
  - function of routing distance and switch delay
  - depends on topology, routing algorithm, switch design, etc.
- Contention
  - Given channel can only be occupied by one message
  - Affected by topology, switching strategy, routing algorithm

Off-chip vs. On-chip ICNs

- Off-chip: I/O bottlenecks
  - Pin-limited bandwidth
  - Inherent overheads of off-chip I/O transmission
- On-chip
  - Wiring constraints
    - Metal layer limitations
    - Horizontal and vertical layout
    - Short, fixed length
    - Repeater insertion limits routing of wires
      - Avoid routing over dense logic
      - Impact wiring density
  - Power
    - Consume 10-15% or more of die power budget
  - Latency
    - Different order of magnitude
    - Routers consume significant fraction of latency

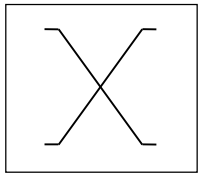
Main Aspects of an ICN

- **Topology**
  - Static arrangement of channels and nodes in a network
- **Routing**
  - Determines the set of paths a message/packet can follow
- **Flow control**
  - Allocating network resources (channels, buffers, etc.) to packets and managing contention
- **Switch microarchitecture**
  - Internal architecture of a network switch
- **Network interface**
  - How to interface a terminal with a switch
- **Link architecture**
  - Signaling technology and data representation on the channel

Topology

Types of Topologies

- We focus on switched topologies
  - Alternatives: bus and crossbar
- ***Bus***
  - Connects a set of components to a single shared channel
  - Effective broadcast medium
- ***Crossbar***
  - Directly connects ***n*** inputs to ***m*** outputs without intermediate stages
  - Fully connected, single hop network
  - Typically used as an internal component of switches
  - Can be implemented using physical switches (in old telephone networks) or multiplexers (far more common today)



Types of Topologies

- ***Direct***
  - Each router is associated with a terminal node
  - *All* routers are sources and destinations of traffic
- ***Indirect***
  - Routers are distinct from terminal nodes
  - Terminal nodes can source/sink traffic
  - Intermediate nodes switch traffic between terminal nodes

Metrics for Comparing Topologies

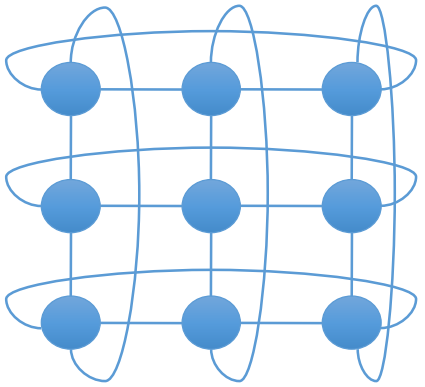
- **Switch degree**
  - Proxy for switch complexity
- **Hop count** (average and worst case)
  - Proxy for network latency
- **Maximum channel load**
  - A proxy for hotspot load
- **Bisection bandwidth**
  - Proxy for maximum traffic a network can support under a uniform traffic pattern
- **Path Diversity**
  - Provides routing flexibility for *load balancing* and *fault tolerance*
  - Enables better congestion avoidance

Detour: Cut and Bisection

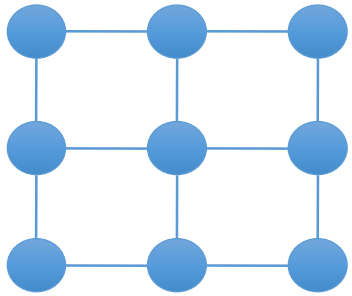
- ***Cut***: a set of channels that partitions the set of all nodes into two disjoint sets, N_1 and N_2
- ***Bisection***: a cut that partitions the network nearly in half
  - bisecting set of nodes: $|N_2| \leq |N_1| \leq |N_2| + 1$
  - and, set of terminals: $|N_2 \cap T| \leq |N_1 \cap T| \leq |N_2 \cap T| + 1$
- ***Bisection bandwidth***: minimum bandwidth over all bisections of the network

Tori and Meshes

- Examples of direct networks
- ***Torus***: k-ary n-cube
  - An n-dimensional grid with k nodes in each dimension
  - k^n nodes; degree = $2n$ (n channels per dim)
  - Each node is connected to its immediate neighbors in the grid
  - Edge nodes in each dimension are also connected
  - k is called the ***radix***
- **Mesh**: k-ary n-mesh
  - Like a torus with no channel between edge nodes



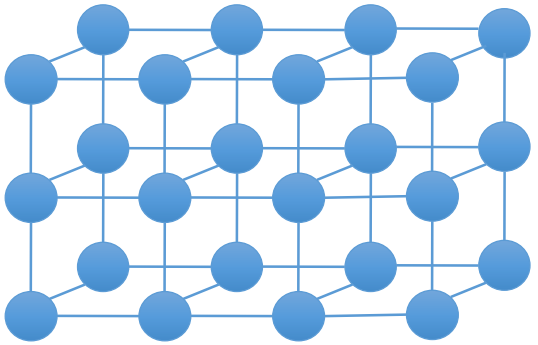
3-ary 2-cube



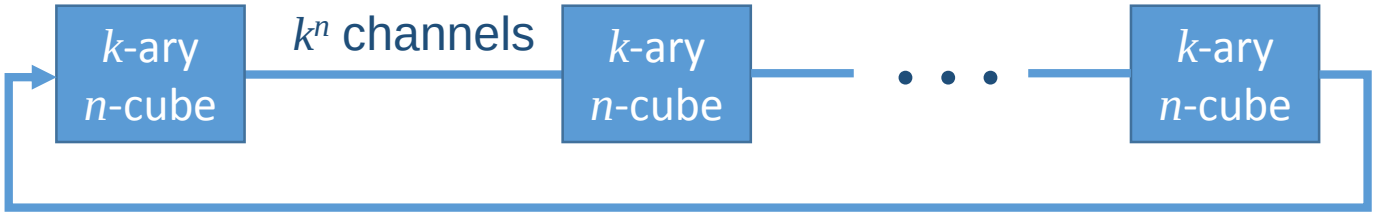
3-ary 2-mesh

Tori and Meshes

- n-cubes can have different radices in different dimensions
  - Example: 2 in Y, 3 in Z and 4 in X
- Very regular: can construct an $n+1$-dim cube by taking k n-dim cubes, arranging them in an array and connecting the corresponding nodes of neghibors



2,3,4-ary 3-cube

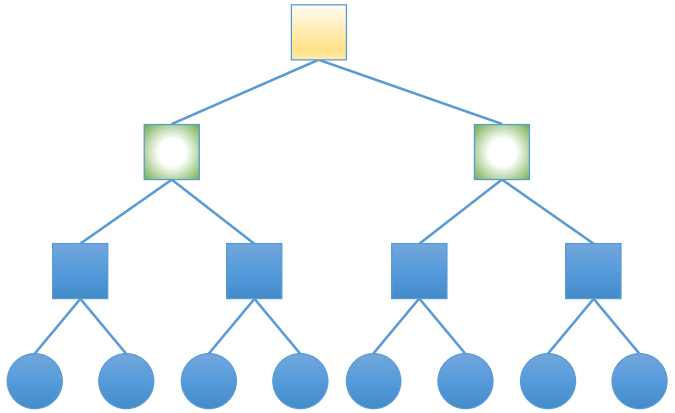


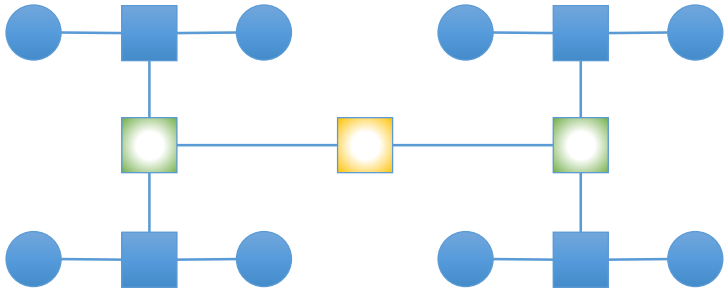
Tori and Meshes

- Famous topologies in this family
  - ***Ring***: k-ary 1-cube
  - 2D and 3D ***grids***
  - ***Hypercube***: 2-ary (binary)s n-cube
- 1D or 2D map well to planar substrate for on-chip
- 3D is easy to build in 3D spaces (*e.g.*, a supercomputer)
- Tori are edge symmetric
  - ✓ Good for load balancing
- Removing wrap-around links for mesh loses edge symmetry
  - ✘ More traffic concentrated on center channels
- Good path diversity
- Exploit locality for near-neighbor traffic
  - Important for many scientific computations

Tree

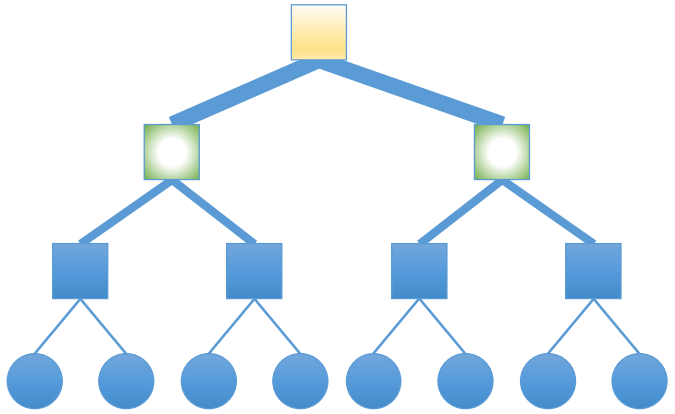
- Diameter and average distance logarithmic
  - k-ary tree, height = $\log_k N$
  - address specified d-vector of radix k coordinates describing path down from root
- Route up to common ancestor and down
- Bisection BW?

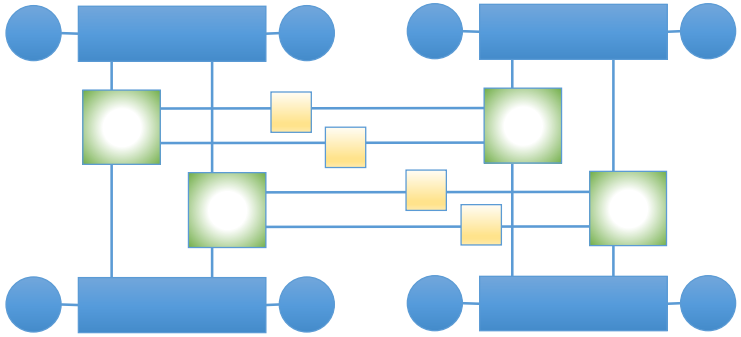




Fat Tree

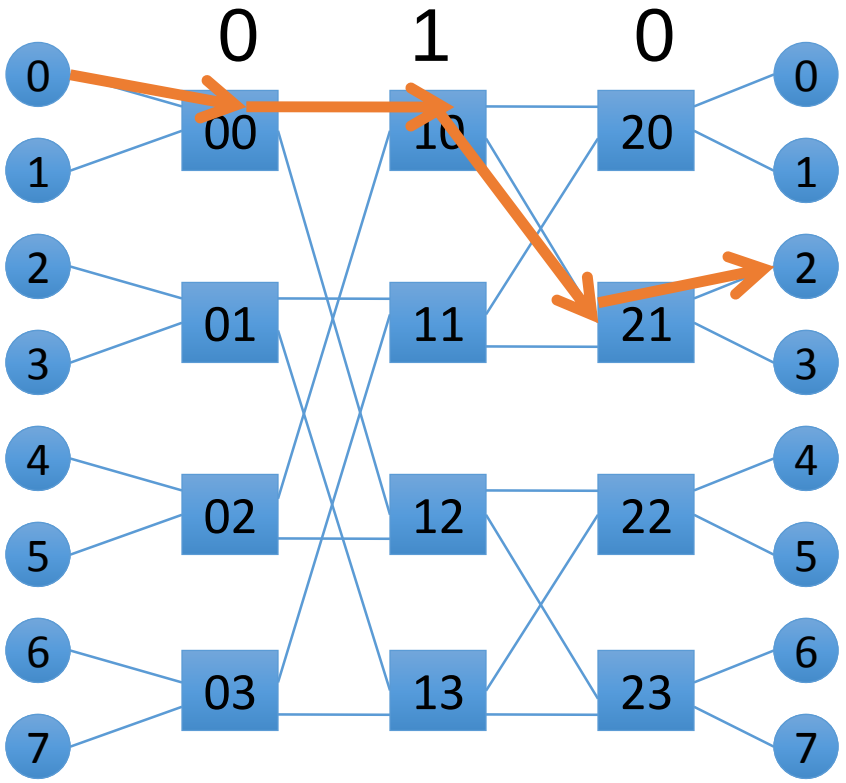
- Bandwidth remains constant at each level
  - Bisection BW scales with number of terminals
- Unlike tree in which bandwidth decreases closer to root
- Fat links can be implemented with increasing the BW (uncommon) or number of channels (more common)





Butterfly (1/3)

- Indirect network
- k-ary n-fly: k^n terminals
  - k: input/output degree of each switch
  - n: number of stages
  - Each stage has k^{n-1} k-by-k switches
- Example routing from 000 to 010
  - Dest address used to directly route packet
  - j^{th} bit used to select output port at stage j

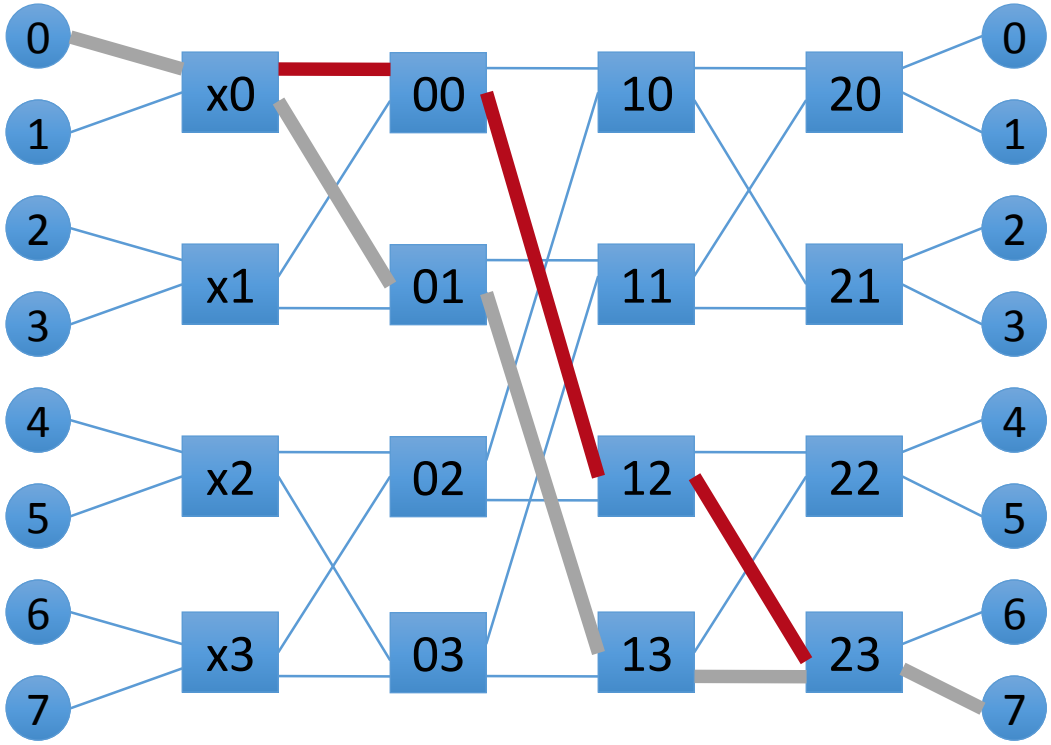


2-ary 3-fly

2 port switch, 3 stages

Butterfly (2/3)

- No path diversity $|R_{xy}| = 1$
- Can add extra stages for diversity
  - Increases network diameter

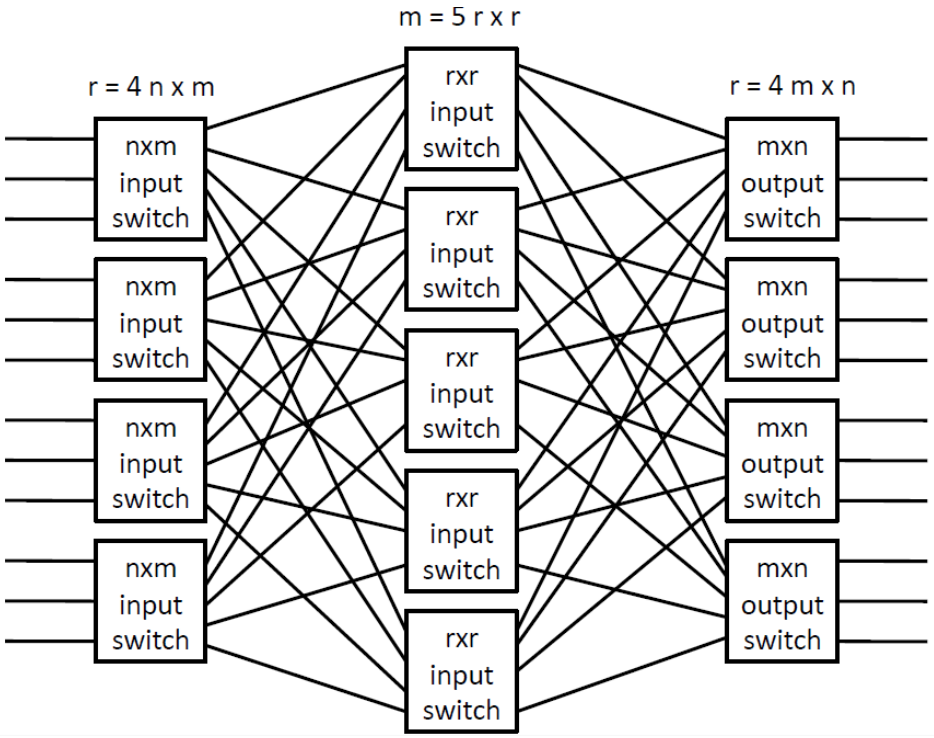


Butterfly (3/3)

- Hop Count = $\log_k N + 1$
- Does **not** exploit **locality**
  - Hop count same regardless of location
- Switch Degree = $2k$
- Requires long wires to implement

Clos Network (1/2)

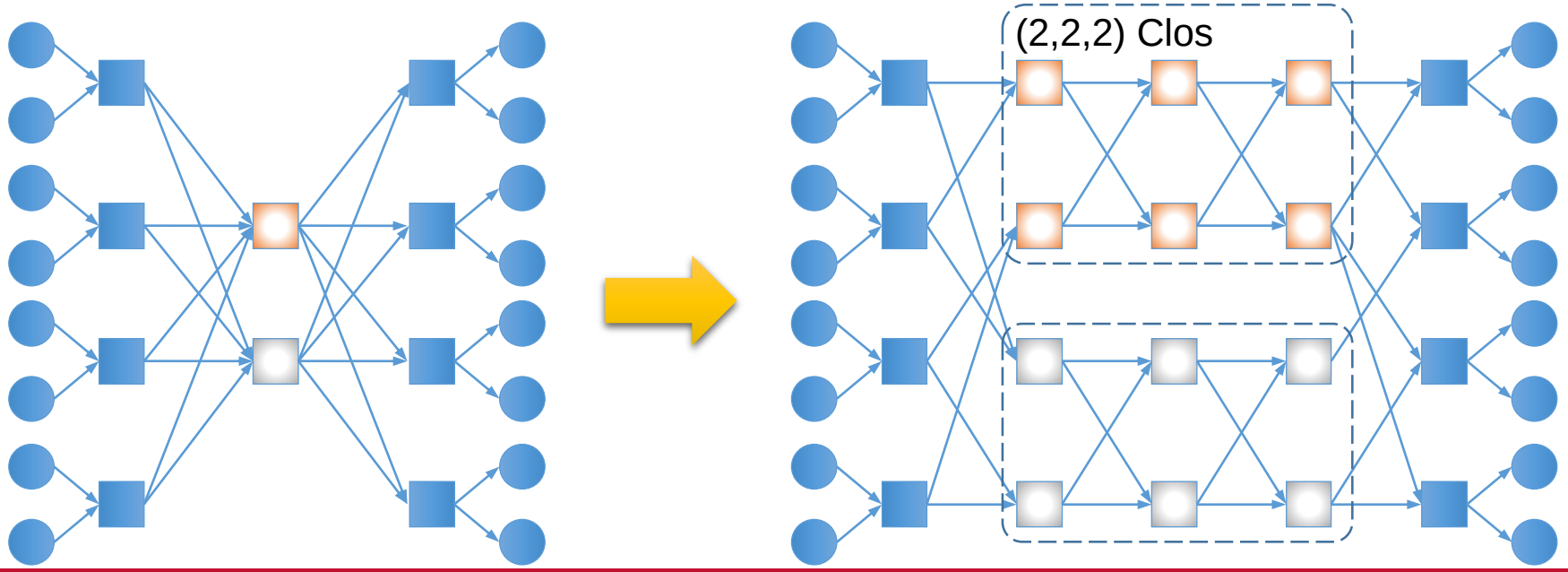
- 3-stage Clos
  - Input switches
  - Output switches
  - Middle switches
- Parameters
  - m: # of middle switches
  - n: in/out degree of edge switches
  - r: # of input/output switches



3-stage Clos network with $m = 5, n = 3, r = 4$

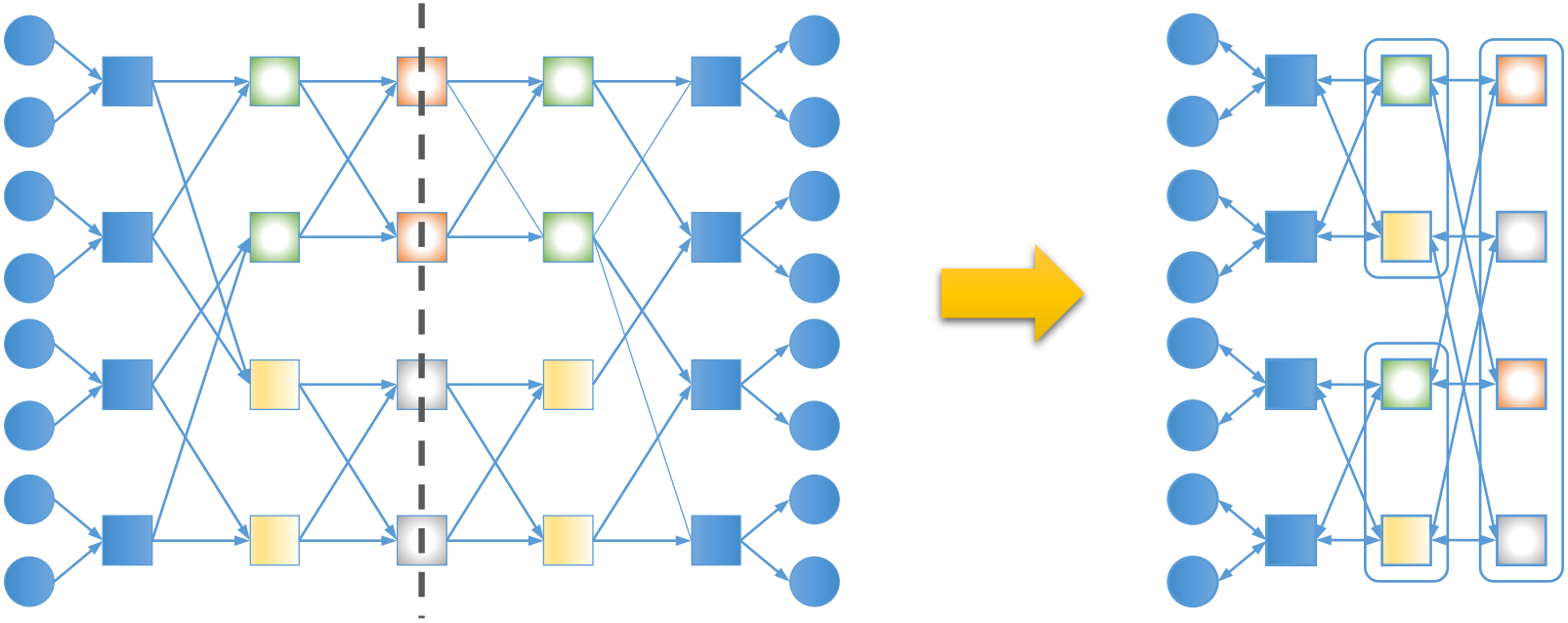
Clos Network (2/2)

- Provides path diversity
  - $|R_{xy}| = m$ (number of middle switches)
  - One path through every middle switch
- Can increase # of stages (and diversity) by replacing the middle stage with another clos network



Folding Clos Networks

- Can fold the network *along the middle stage* to share input/output switches



- The right-hand side is a fat tree
  - Alternative impl. w/ more links instead of high-BW links

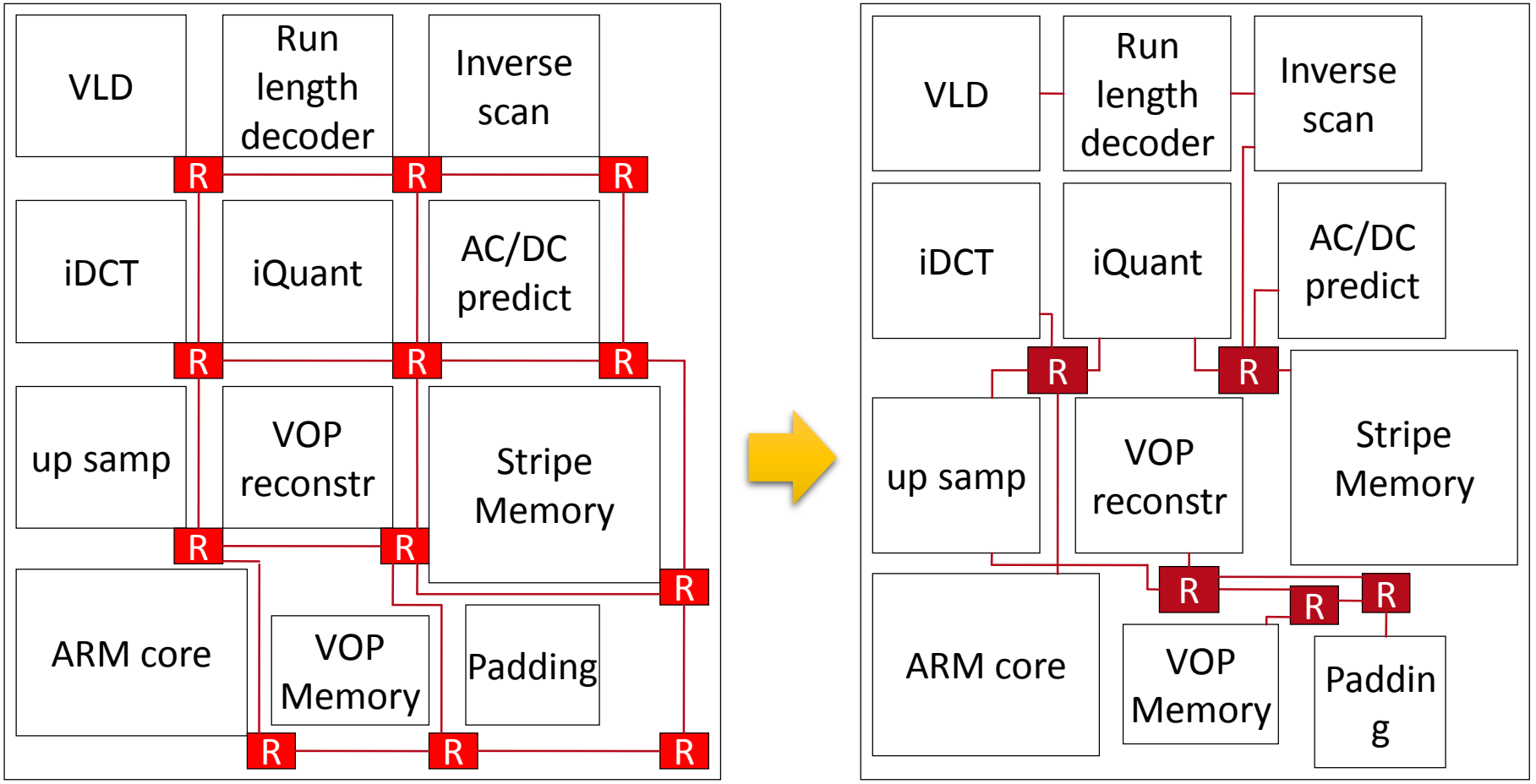
And Other Topologies…

- Many other topologies with different properties discussed in the literature
  - Omega networks
  - Benes networks
  - Bitonic networks
  - Flattened Butterfly
  - Dragonfly
  - Cube-connected cycles
  - HyperX
  - …
- However, these are typically special purpose and not used in general purpose hardware

Irregular Topologies

- Common in MPSoC (Multiprocessor System-on-Chip) designs
- MPSoC design leverages wide variety of IP blocks
  - Regular topologies may not be appropriate given heterogeneity
  - Customized topology
    - Often more power efficient and deliver better performance
- Customize based on traffic characterization
  - Often synthesized using automatic tools

Irregular Topology Example



Flow Control

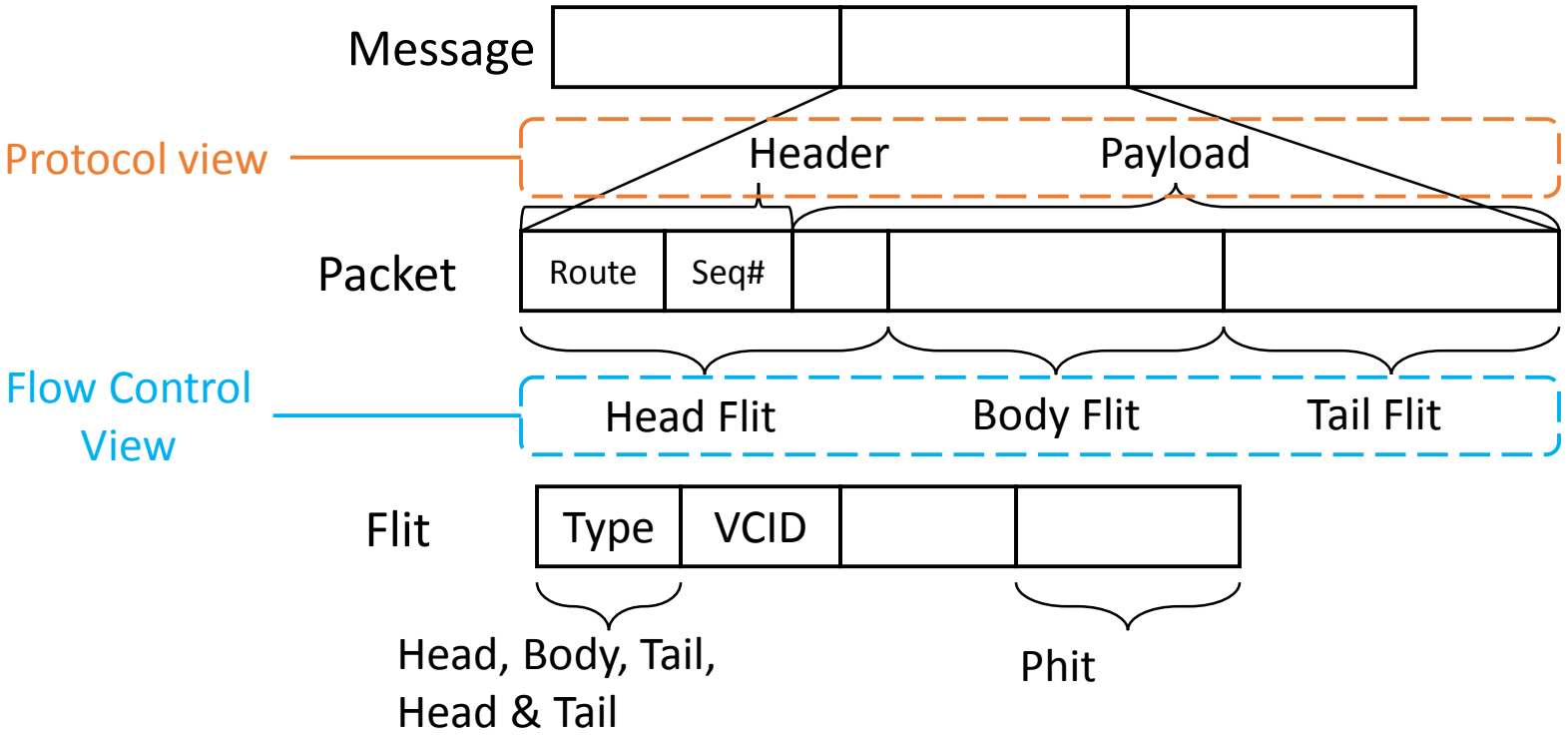
Flow Control Overview

- Flow Control: determine allocation of resources to messages as they traverse network
  - Buffers and links
  - Significant impact on throughput and latency of network

Flow Control Units:

- ***Message***: composed of one or more packets
  - If message size is <= maximum packet size only one packet created
- ***Packet***: composed of one or more flits
- ***Flit***: flow control digit
- ***Phit***: physical digit
  - Subdivides flit into chunks = to link width

Flow Control Overview



- Packet contains destination/route information
  - Flits may not → all flits of a packet must take same route

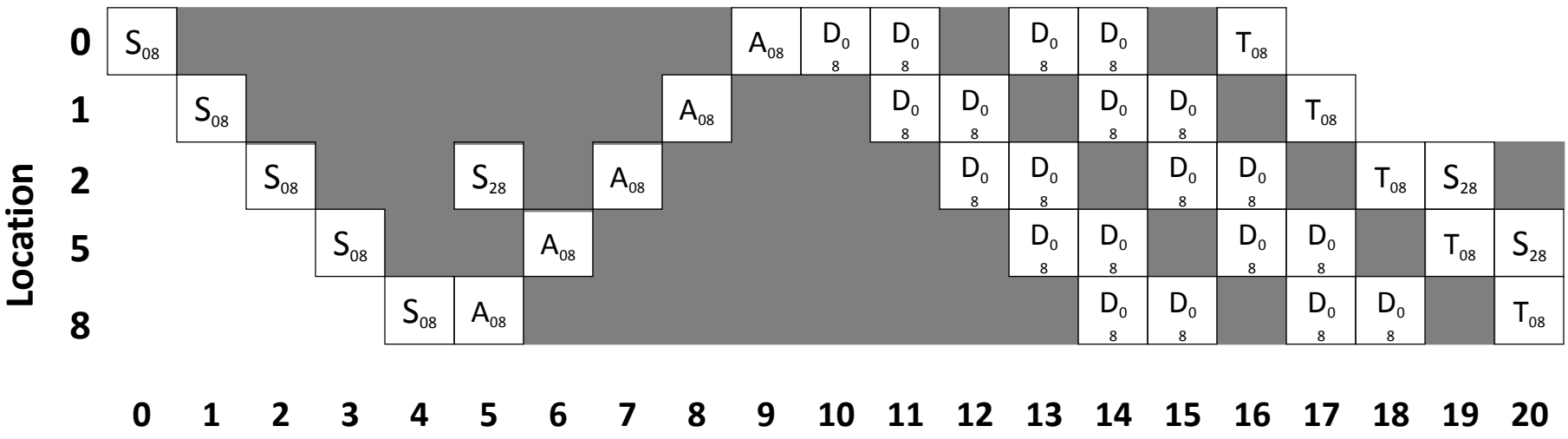
Switching

- Different flow control techniques based on granularity
- **Message-based**: allocation made at message granularity (circuit-switching)
- **Packet-based**: allocation made to whole packets
- **Flit-based**: allocation made on a flit-by-flit basis

Message-Based Flow Control

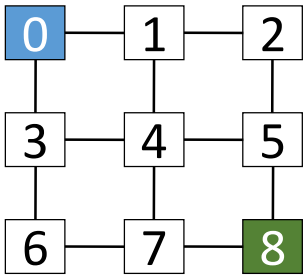
- Coarsest granularity
- **Circuit Switching**
  - Pre-allocates resources across multiple hops
    - Source to destination
    - Resources = links (buffers not necessary)
  - Probe sent into network to reserve resources
  - Message does not need per-hop routing or allocation once probe sets up circuit
- Good for transferring large amounts of data
- No other message can use resources until transfer is complete
  - Throughput can suffer due setup and hold time for circuits
  - Links are idle until setup is complete

Time-Space Diagram: Circuit-Switching



Time to setup+ack circuit from 0 to 8

Time setup from 2 to 8 is blocked



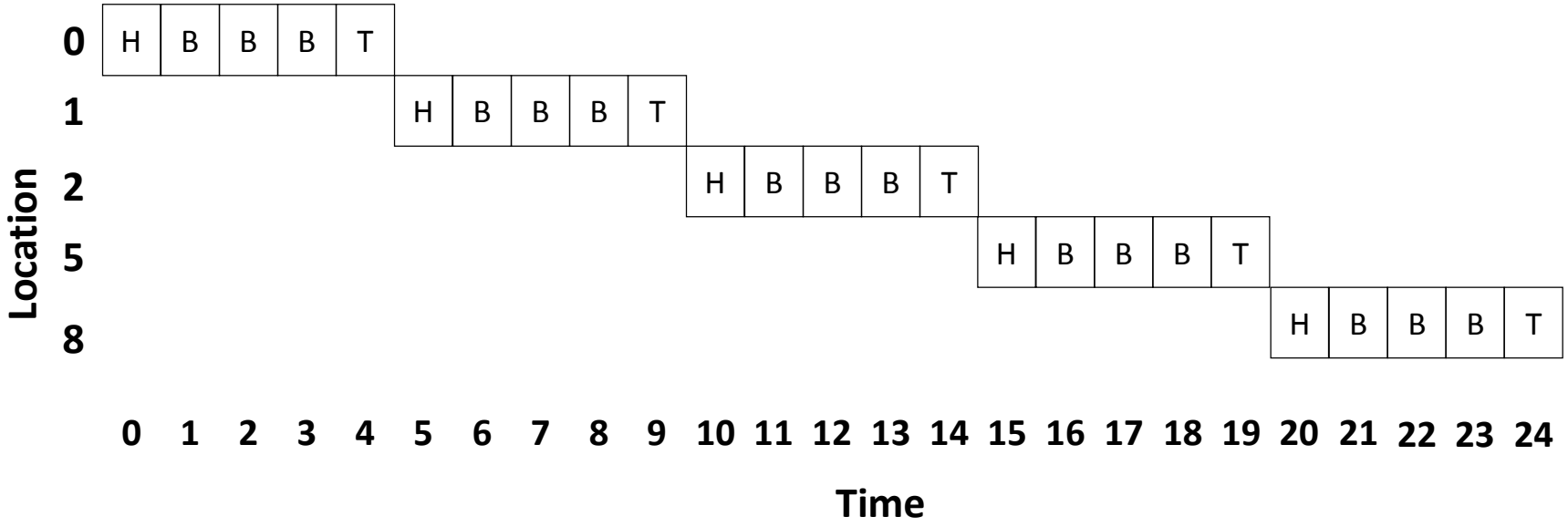
Packet-based Flow Control

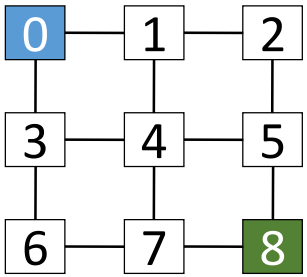
- Break messages into packets
- Interleave packets on links
  - Better utilization
- Requires per-node buffering to store in-flight packets
- Two types of packet-based techniques
  - Store & Forward
  - Virtual Cut-Through

Store & Forward (S&F)

- Links and buffers are allocated to entire packet
- Head flit waits at router until entire packet is received (Store) before being forwarded to the next hop (Forward)
- Not suitable for on-chip
  - Requires buffering at each router to hold entire packet
    - Packet cannot traverse link until buffering allocated to entire packet
  - Incurs high per-hop latency (pays serialization latency at each hop)

Time-Space Diagram: S&F

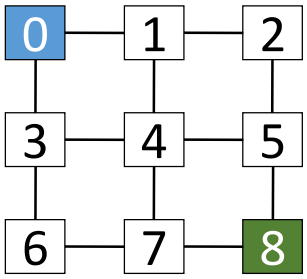


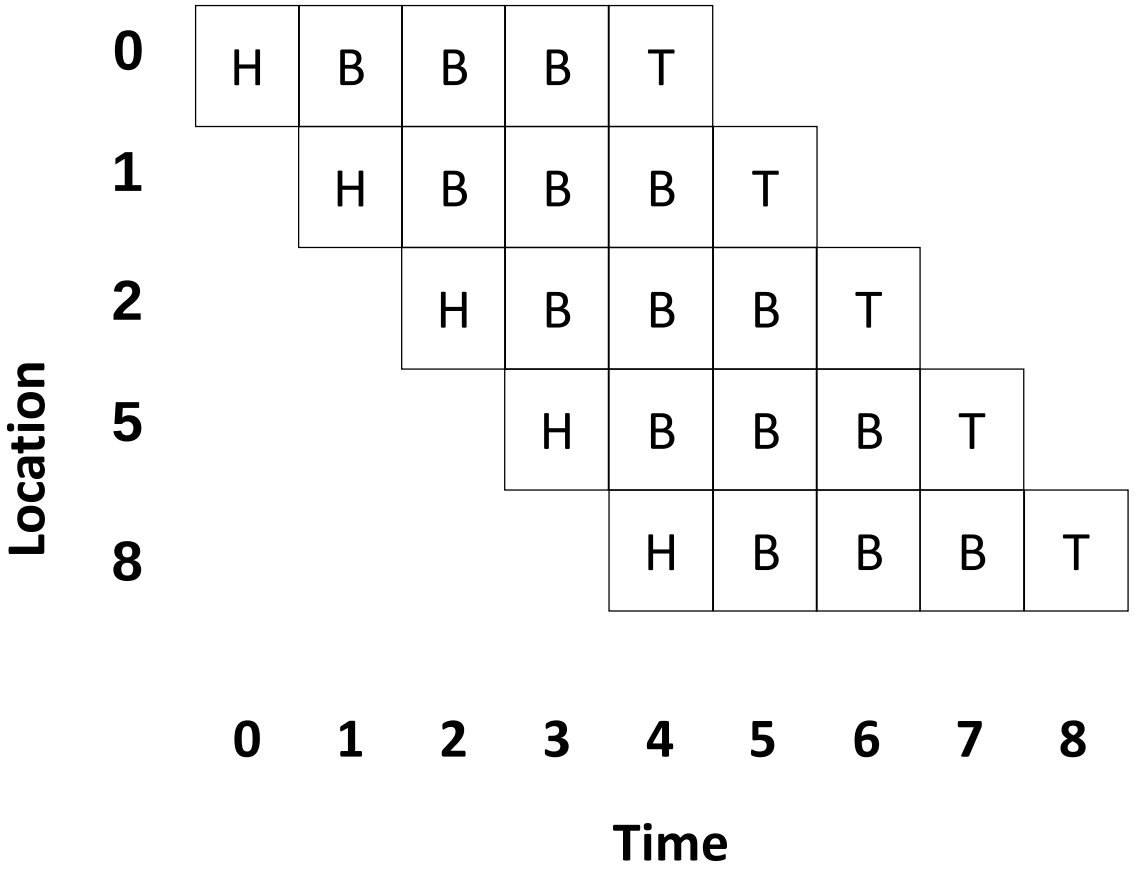


Virtual Cut-Through (VCT)

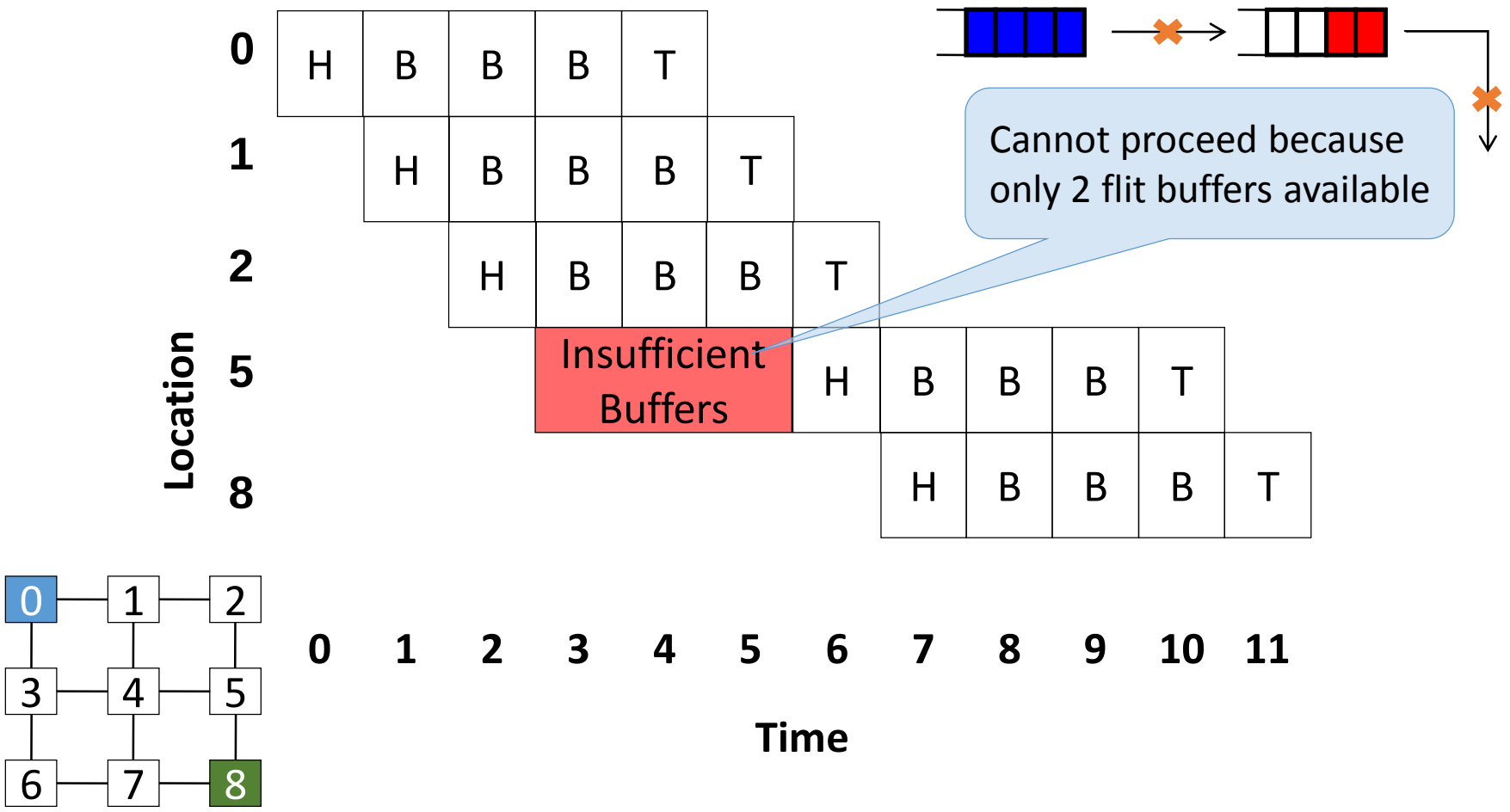
- Links and Buffers allocated to entire packets
- Flits can proceed to next hop before tail flit has been received by current router
  - Only if next router has enough buffer space for entire packet
- Reduces the latency significantly compared to Store & Forward
- Still requires large buffers
  - Unsuitable for on-chip

Time-Space Diagram: VCT





Time-Space Diagram: VCT (2)

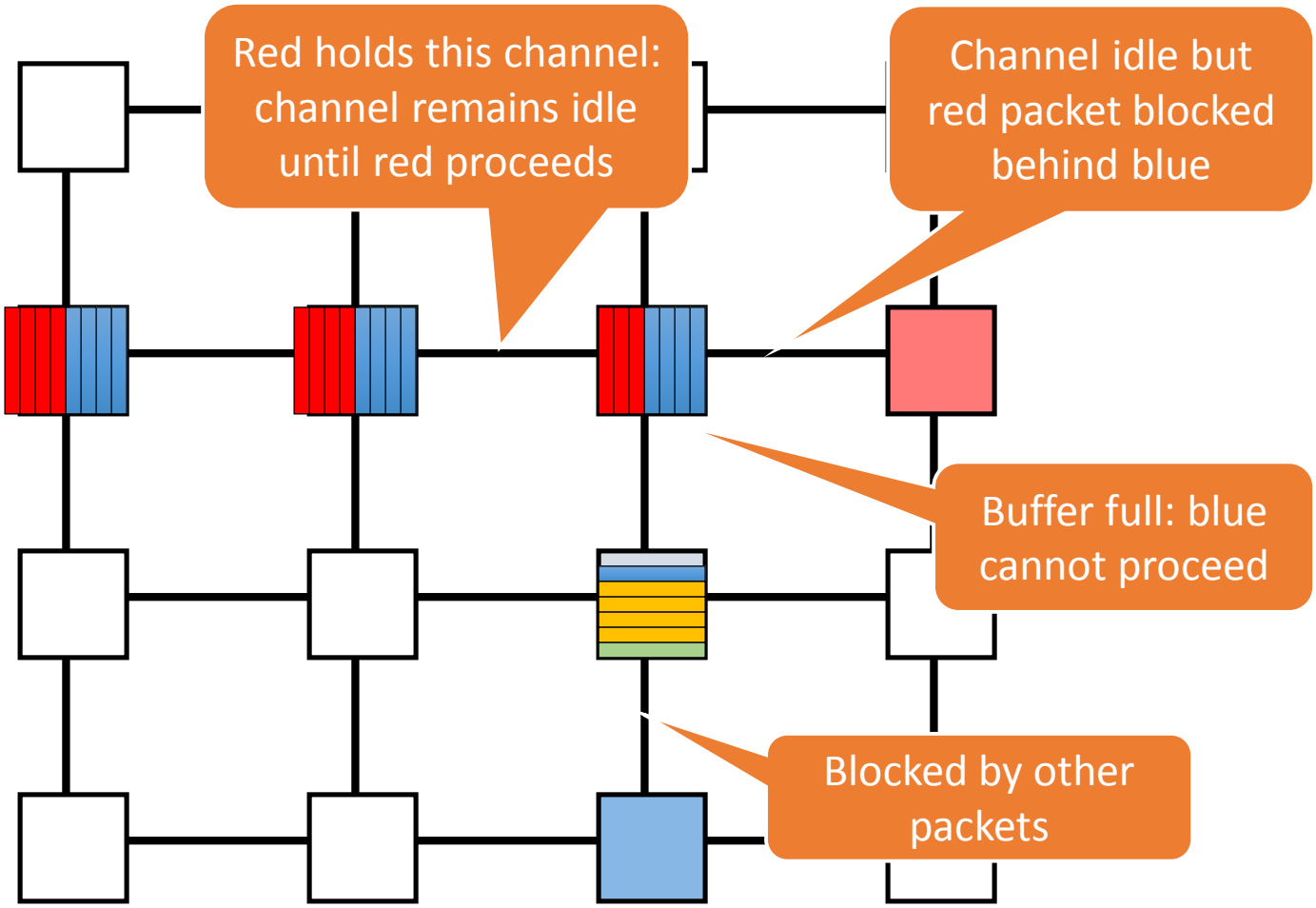


Flit-Level Flow Control

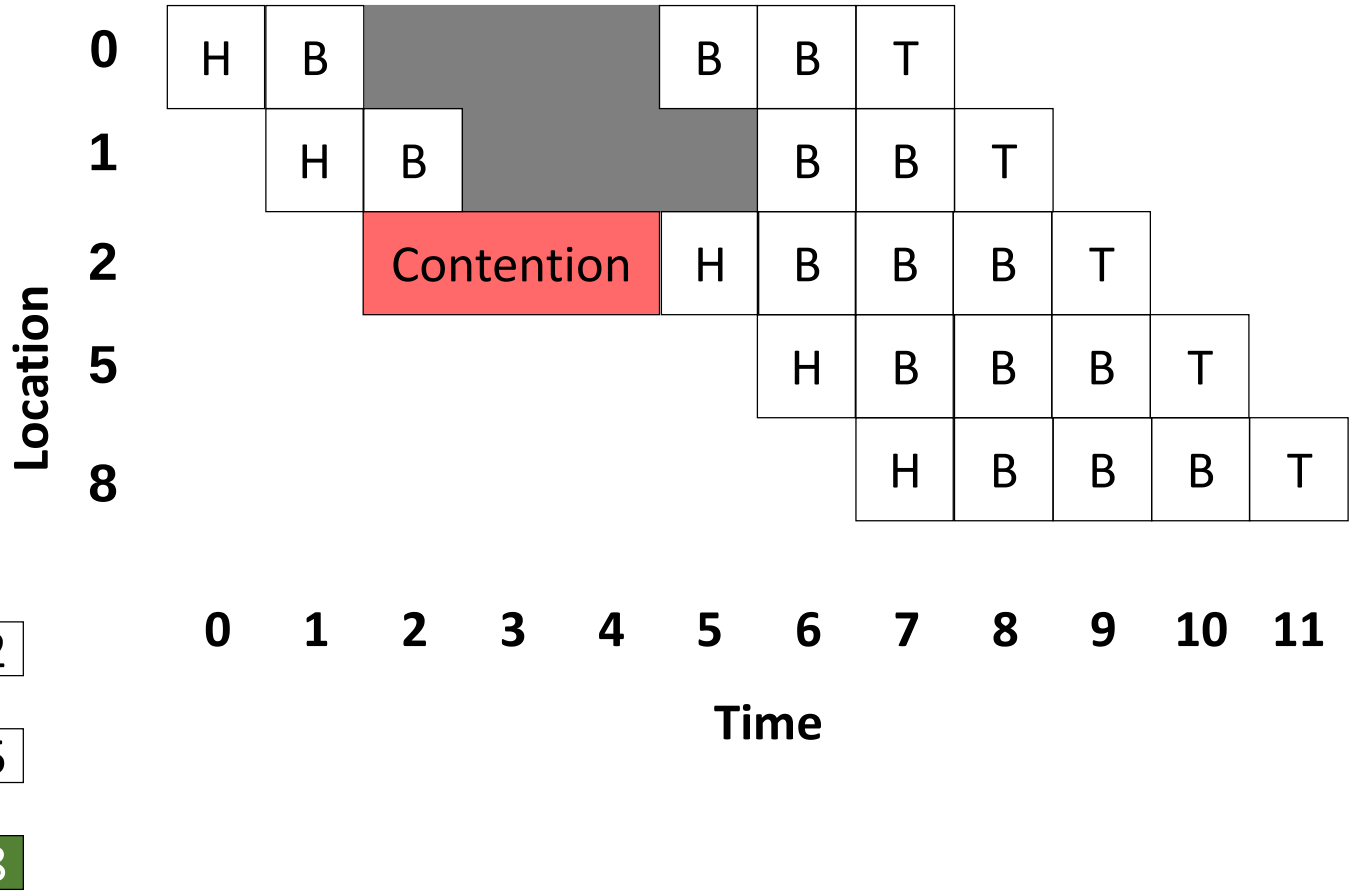
- Flit can proceed to next router when there is buffer space available for that flit
  - Improves over SAF and VCT by allocating buffers on a flit-by-flit basis
  - Help routers meet tight area/power constraints
- Called **Wormhole Flow Control**
  - ✓ More efficient buffer utilization (good for on-chip)
  - ✓ Low latency
  - ✗ Poor link utilization: if head flit becomes blocked, all links spanning length of packet are idle
    - Cannot be re-allocated to different packet
    - Suffers from **head of line** (HOL) blocking

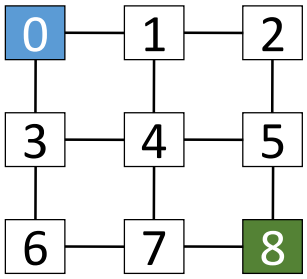
Wormhole Example

- 6-flit buffers per input port
- 2 4-flit packets
  - Red & blue



Time-Space Diagram: Wormhole

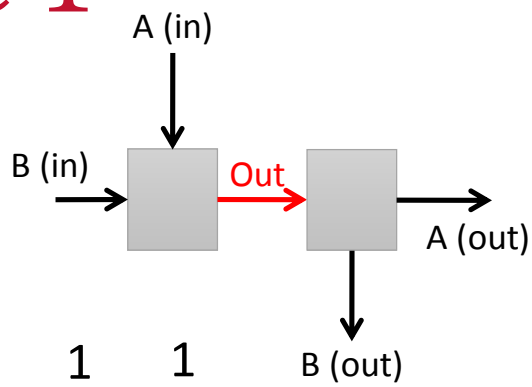




Virtual Channel Flow Control

- ***Virtual Channels***: multiple flit queues per input port
  - Share same physical link (channel)
- Used to combat HOL blocking in wormhole
  - Flits on different VC can pass blocked packet
  - Link utilization improved
- VCs first proposed for deadlock avoidance
  - We'll come back to this
- Can be applied to any flow control
  - First proposed with wormhole

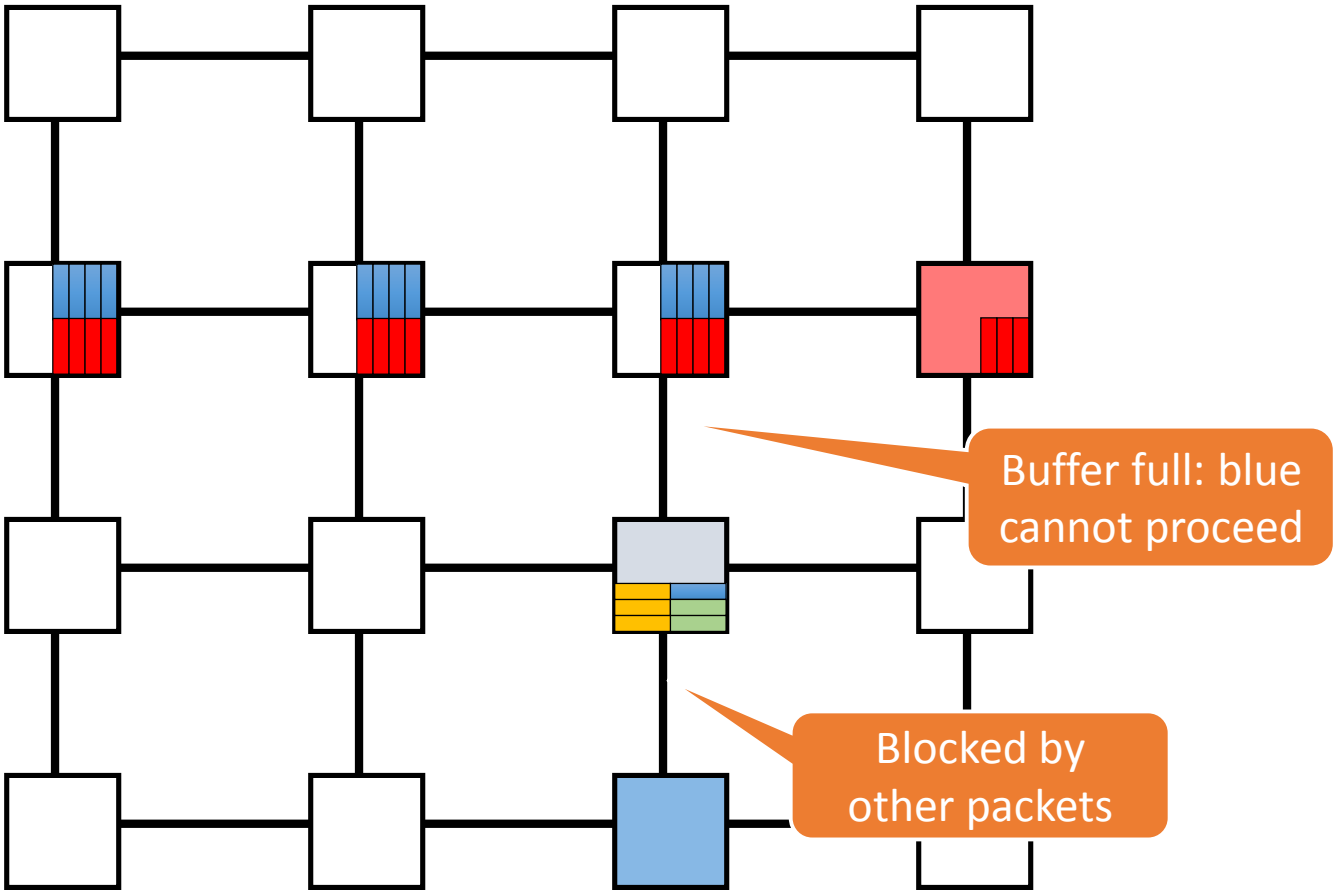
VC Flow Control – Example 1



A (in)	AH	A1	A2	A3	A4	A5		AT					
Occupancy	1	1	2	2	3	3	3	3	3	2	2	1	1
B (in)	BH	B1	B2	B3	B4		B5		BT				
Occupancy	1	2	2	3	3	3	3	3	3	3	2	2	1	1
Out	AH	BH	A1	B1	A2	B2	A3	B3	A4	B4	A5	B5	AT	BT
A (out)	AH		A1		A2		A3		A4		A5		AT	
B (out)		BH		B1		B2		B3		B4		B5		BT

VC Flow Control – Example 2

- 6-flit buffers per input port
- 3 flit buffers per VC



Summary of techniques

	Links	Buffers	Comments
Circuit-Switching	Messages	N/A (buffer-less)	Setup & Ack
Store and Forward	Packet	Packet	Head flit waits for tail
Virtual Cut Through	Packet	Packet	Head can proceed
Wormhole	Packet	Flit	HOL
Virtual Channel	Flit	Flit	Interleave flits of different packets

Buffer Backpressure

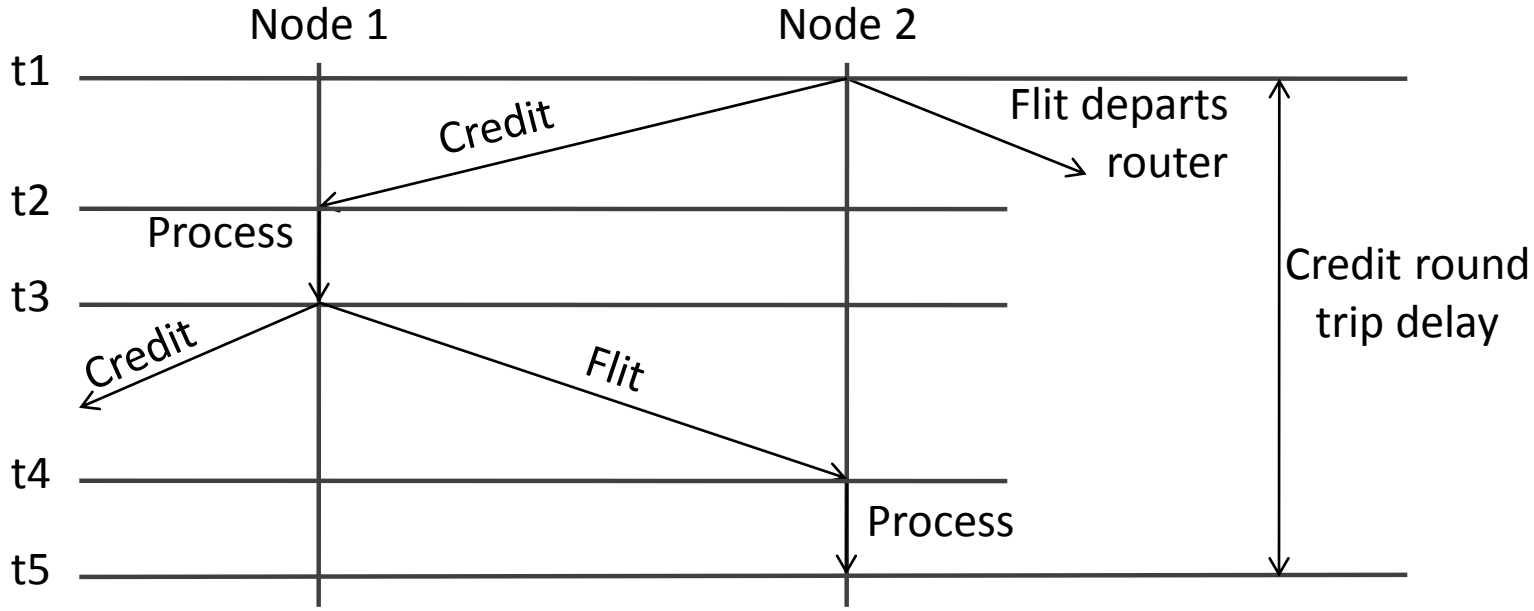
Buffer Backpressure

- Need mechanism to prevent buffer overflow
  - Avoid dropping packets
  - Upstream routers need to know buffer availability at downstream routers
- Significant impact on throughput achieved by flow control
- Two common mechanisms
  - Credits
  - On-off

Credit-Based Flow Control

- Upstream router stores credit counts for each downstream VC
- Upstream router
  - When flit forwarded
    - Decrement credit count
  - Count == 0, buffer full, stop sending
- Downstream router
  - When flit forwarded and buffer freed
    - Send credit to upstream router
    - Upstream increments credit count

Credit Timeline

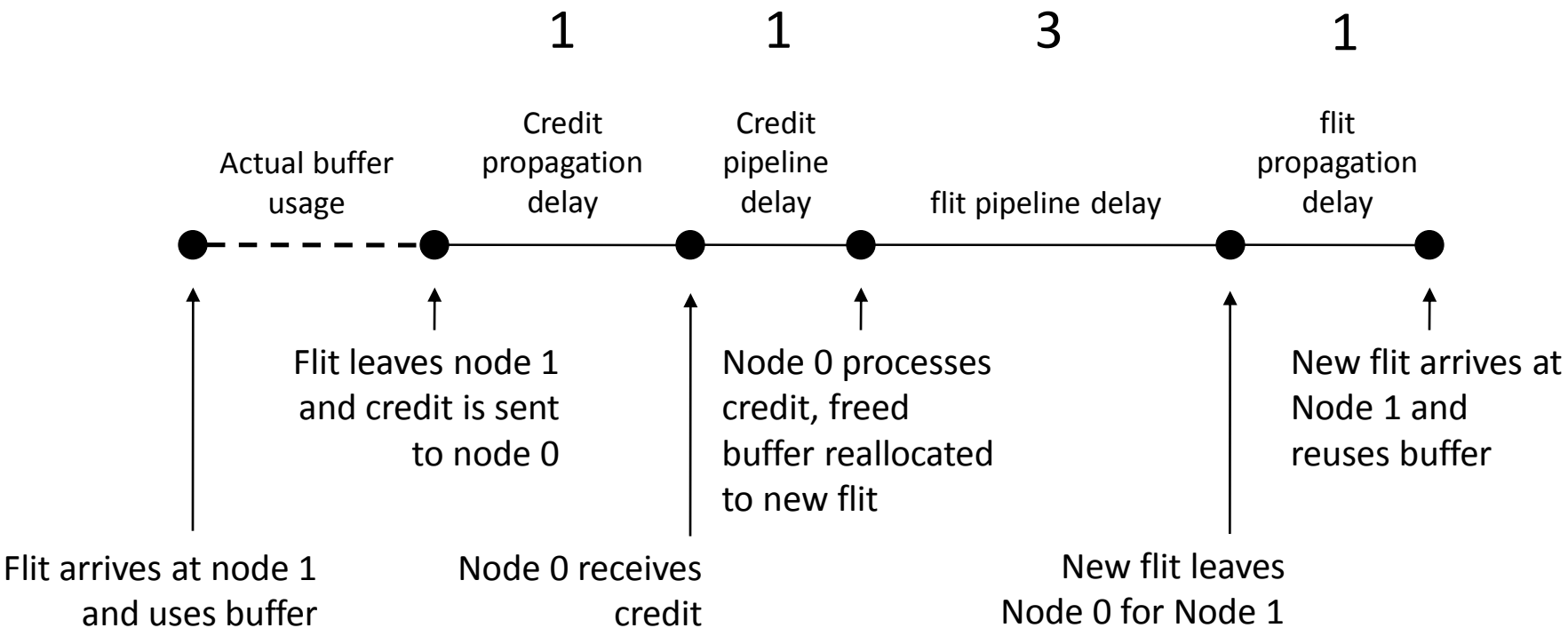


- Round-trip credit delay:
  - Time between when buffer empties and when next flit can be processed from that buffer entry
- Single entry buffer would result in significant throughput degradation
  - Important to size buffers to tolerate credit turn-around

Buffer Sizing

- Prevent backpressure from limiting throughput
  - Buffers must hold # of flits >= turnaround time
- Assume:
  - 1 cycle propagation delay for data and credits
  - 1 cycle credit processing delay
  - 3 cycle router pipeline
- At least 6 flit buffers

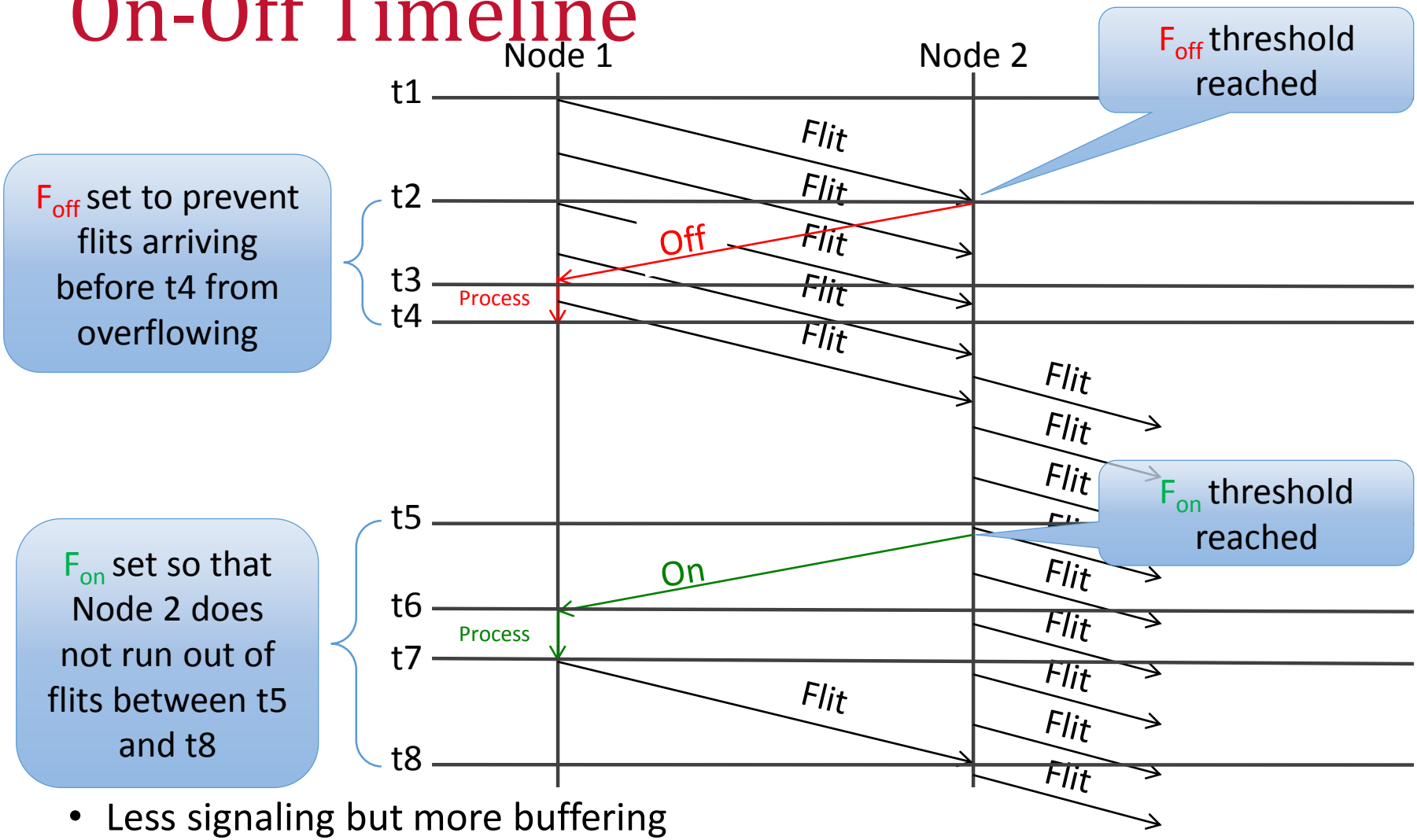
Actual Buffer Usage & Turnaround Delay



On-Off Flow Control

- Credit requires upstream signaling for every flit
- On-Off: decreases upstream signaling
  - Off signal: sent when number of free buffers falls below threshold F_{off}
  - On signal: sent when number of free buffers rises above threshold F_{on}

On-Off Timeline



- Less signaling but more buffering
  - On-chip buffers more expensive than wires

Flow Control Summary

- On-chip networks require techniques with lower buffering requirements
  - Wormhole or Virtual Channel flow control
- Avoid dropping packets in on-chip environment
  - Requires buffer backpressure mechanism
- Complexity of flow control impacts router micro-architecture

Routing

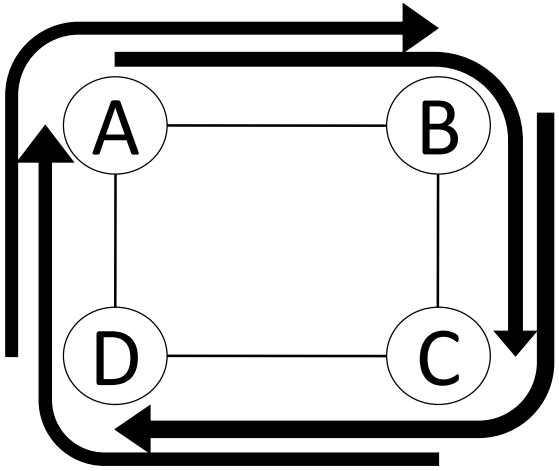
Routing Overview

- Discussion of topologies assumed ideal routing
- In practice…
  - Routing algorithms are not ideal
- Goal: distribute traffic **evenly** among paths
  - Avoid hot spots, contention
  - More balanced → closer throughput is to ideal
- Keep complexity in mind
  - Routing delay can become significant with complex routing mechanisms

Classifications of Routing Algorithms

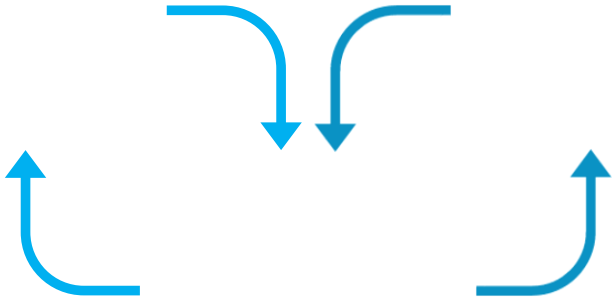
- ***Adaptivity***: does take network state (*e.g.*, congestion) into account?
  - Oblivious
    - Deterministic vs. non-deterministic
  - Adaptive
- ***Hop count***: are all allowed routes minimal?
  - Minimal
  - Non-minimal
- ***Routing decision***: where is it made?
  - Source routing
  - Per-hop routing
- ***Implementation***
  - Table
  - Circuit

Routing Deadlock

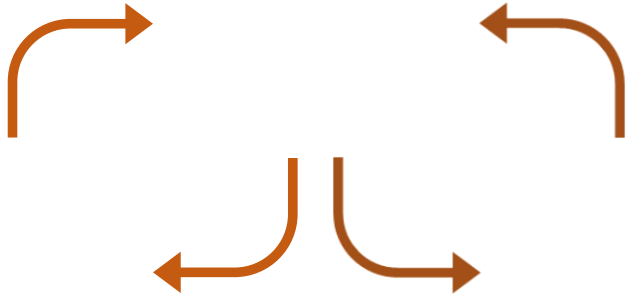


- Each packet is occupying a link and waiting for a link
- Without routing restrictions, a resource cycle can occur
  - Leads to deadlock
- To general ways to avoid
  - Deadlock-free routing: limit the set of turns the routing algorithm allows
  - Deadlock-free flow control: use virtual channels wisely
    - E.g., use ***Escape VCs***

Dimension Order Routing



Turns in X-Y routing

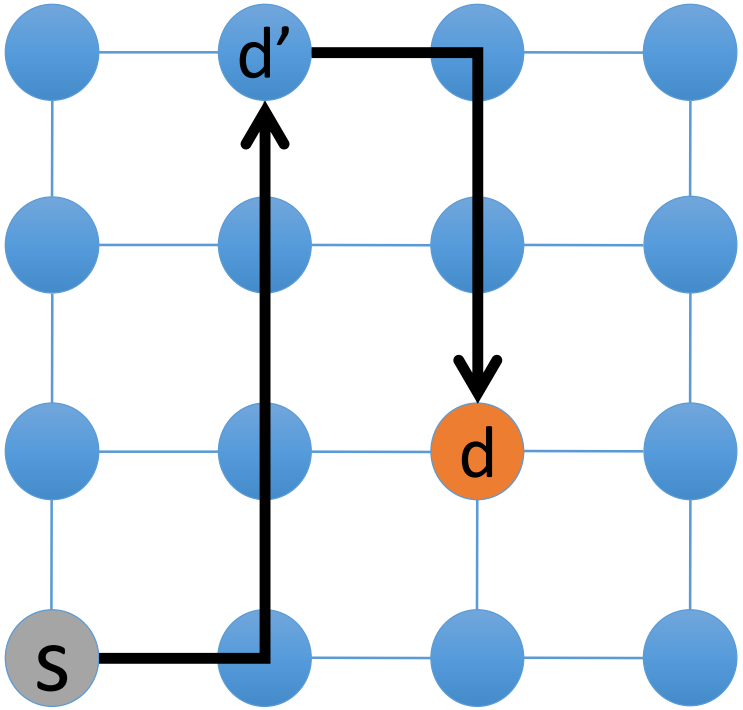


Turns in Y-X routing

- Traverse network dimension by dimension
  - X-Y routing: can only turn to Y dimension after finished X
  - Y-X routing: can only turn to X dimension after finished Y
- Deterministic and Minimal
  - Being deterministic implies oblivion but not often called so (term ***oblivious*** reserved for non-deterministic routing).

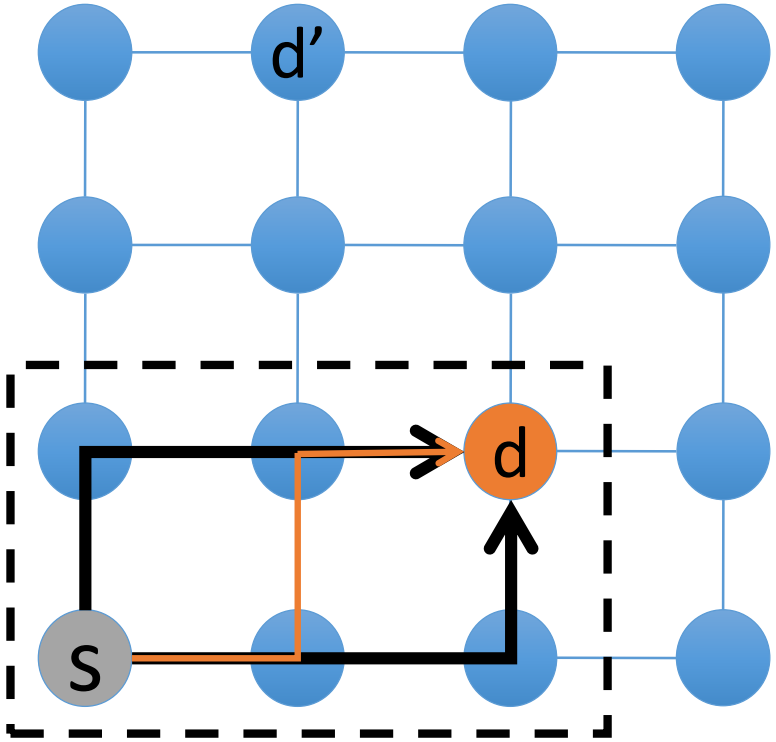
Valiant's Oblivious Routing Algorithm

- An oblivious algorithm
- To route from s to d
  - Randomly choose intermediate node d'
  - Route from s to d' and from d' to d
- Randomizes any traffic pattern
  - All patterns appear uniform random
  - Balances network load
- Non-minimal
- Destroys locality



Minimal Oblivious

- Valiant's: Load balancing but significant increase in hop count
- Minimal Oblivious: some load balancing, but use shortest paths
  - d' must lie within min quadrant
  - 6 options for d'
  - Only 3 different paths



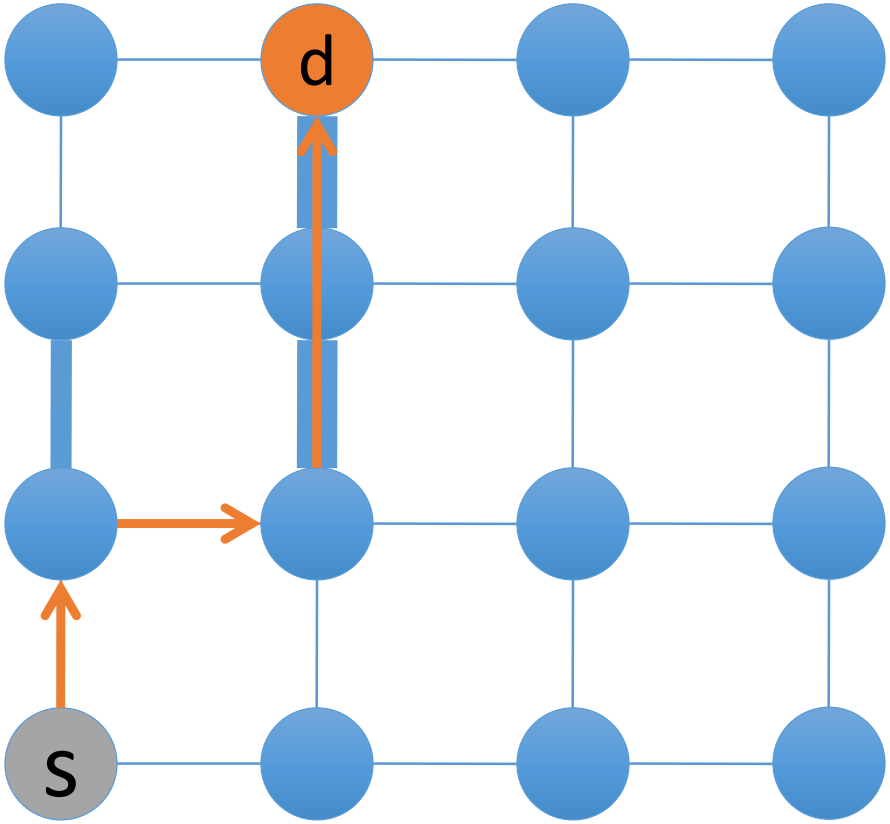
Oblivious Routing

- Valiant's and Minimal Adaptive
  - Deadlock free when used in conjunction with X-Y routing
- What if randomly choose between X-Y and Y-X routes?
  - Oblivious but not deadlock free!
- How to make it deadlock free?
  - Need 2 virtual channels
- Either version can be generalized to more than two phases
  - Choose more than one intermediate points

Adaptive

- Exploits path diversity
- Uses network state to make routing decisions
  - Buffer occupancies often used
  - Relies on flow control mechanisms, especially back pressure
- Local information readily available
  - Global information more costly to obtain
  - Network state can change rapidly
    - Use of local information can lead to non-optimal choices
- Can be minimal or non-minimal

Minimal Adaptive Routing

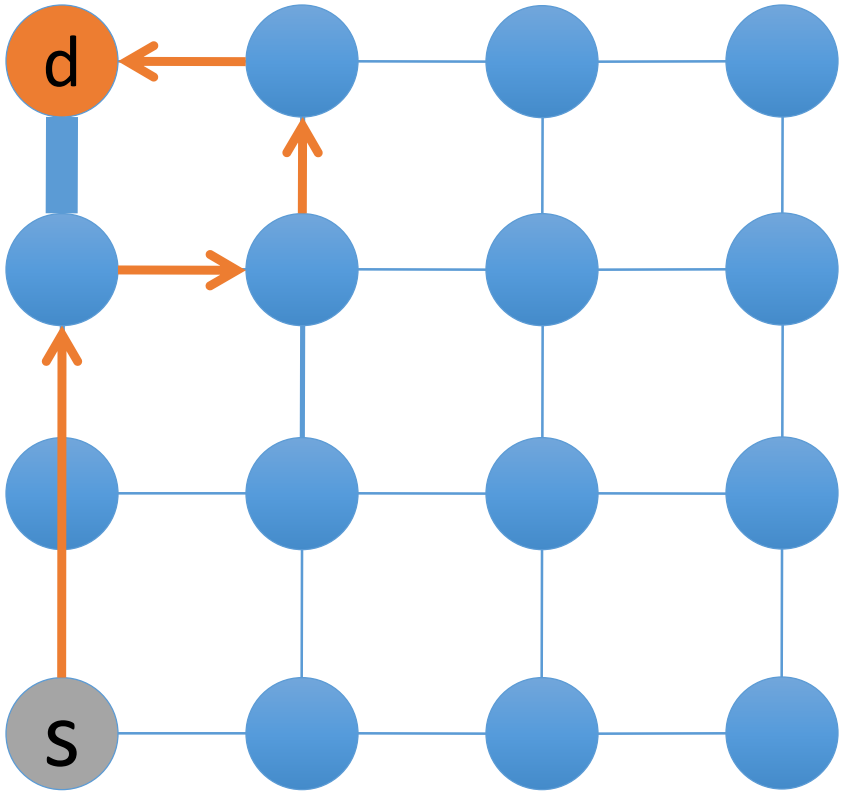


- Local info can result in sub-optimal choices

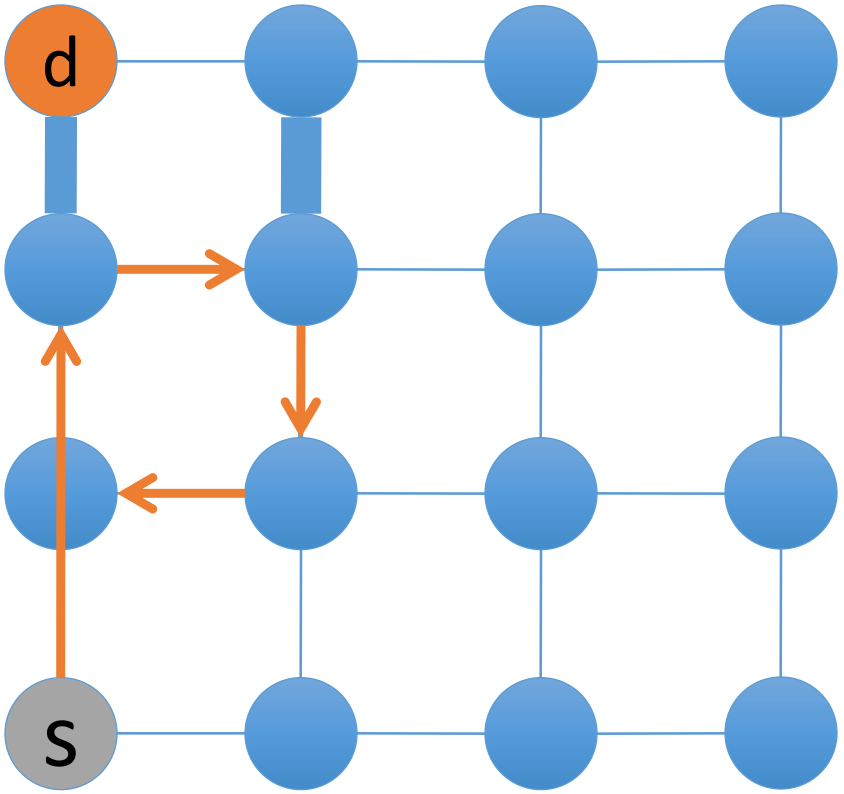
Non-minimal adaptive

- Fully adaptive
- Not restricted to take shortest path
- ***Misrouting***: directing packet along *non-productive* channel
  - Priority given to productive output
  - Some algorithms forbid U-turns
- Livelock potential: traversing network without ever reaching destination
  - Mechanism to guarantee forward progress
    - Limit number of misroutings

Non-minimal routing example

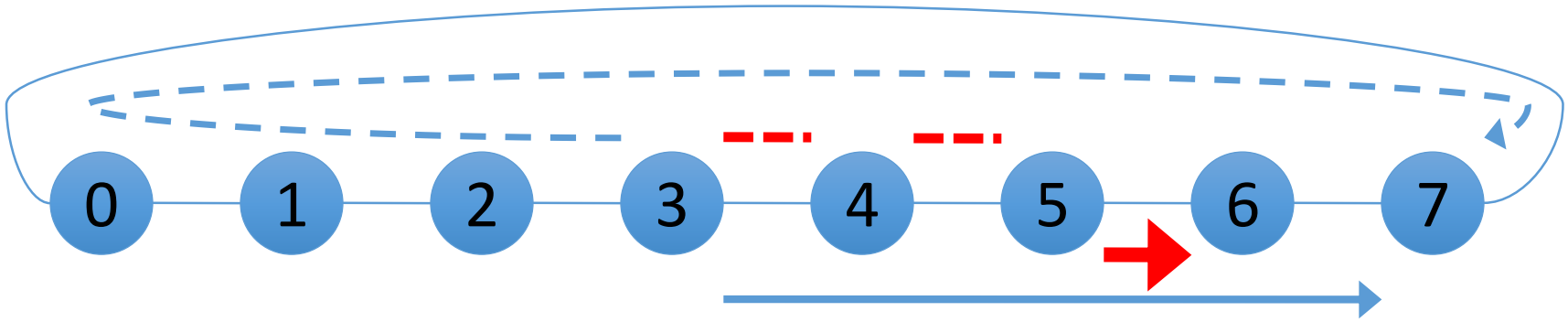


Longer path with potentially lower latency



Livelock: continue routing in cycle

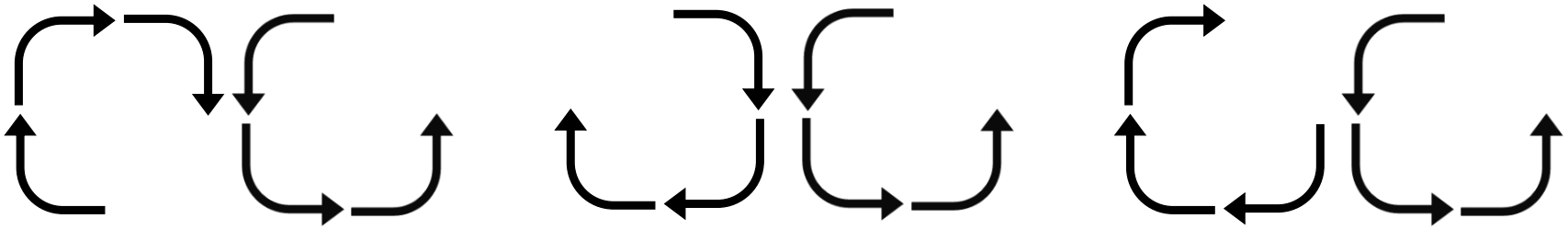
Adaptive Routing Example



- Should 3 route clockwise or counterclockwise to 7?
  - 5 is using all the capacity of link 5 → 6
- Queue at node 5 will sense contention but not at node 3
- Backpressure: allows nodes to indirectly sense congestion
  - Queue in one node fills up, it will stop receiving flits
  - Previous queue will fill up
- If each queue holds 4 packets
  - 3 will send 8 packets before sensing congestion

Adaptive Routing: Turn Model

- Successful adaptive routing requires path diversity
- Removing too many turns limits flexibility in routing
  - E.g., DOR eliminates 4 turns
    - N to E, N to W, S to E, S to W
- Question: how to ensure deadlock freedom while removing a minimum set of turns?
- Examples of valid turn models:

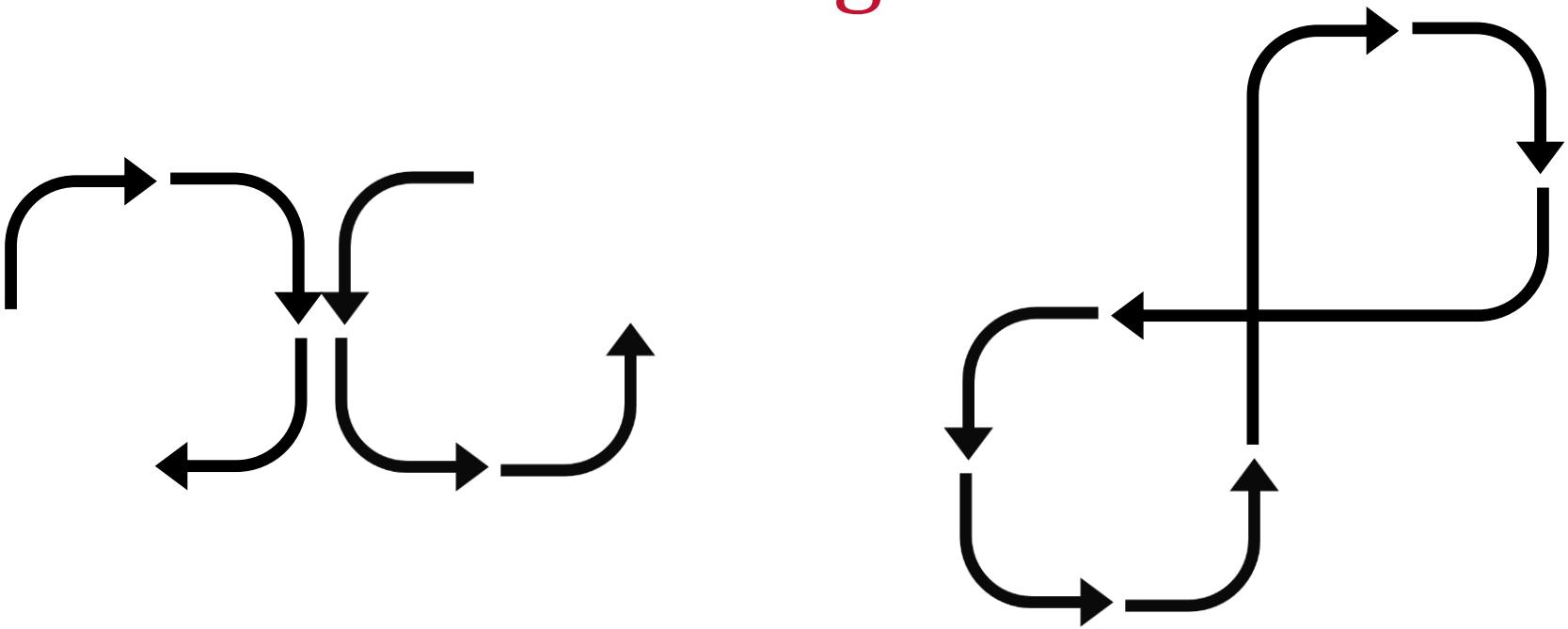


West first

North last

Negative first

Turn Model Routing Deadlock



- What about eliminating turns NW and WN?
- Not a valid turn elimination
  - Resource cycle results

→ Not all 2-removals result in valid turn models

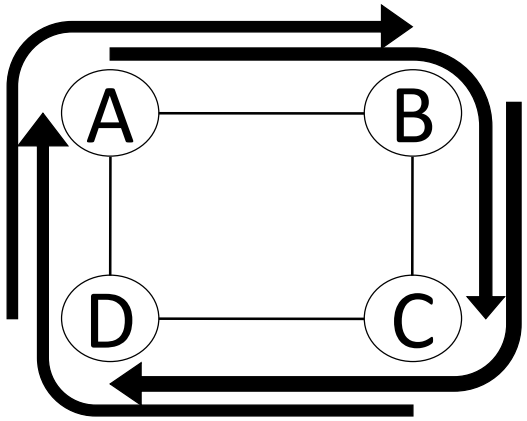
Deadlock Avoidance Using VCs

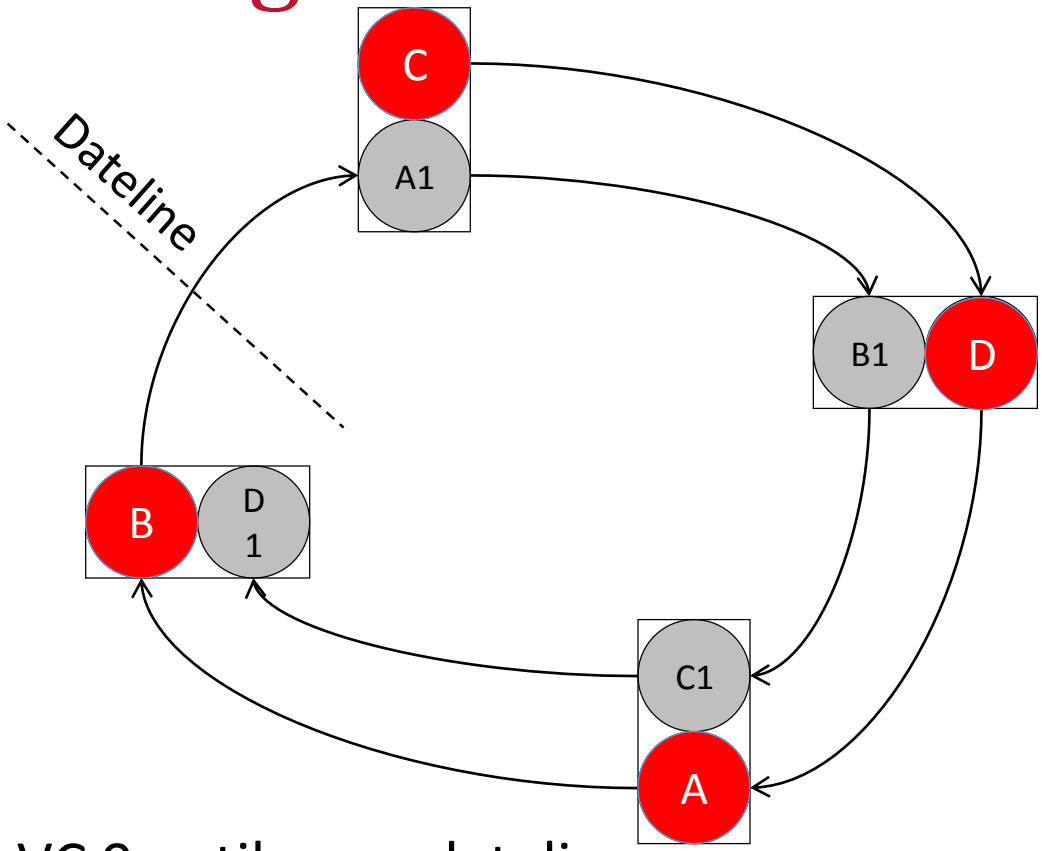
- Deadlock-free routing flow control to guarantee deadlock freedom give more flexible routing
  - VCs can break resource cycle if routing is not deadlock free
- Each VC is time-multiplexed onto physical link
  - Holding VC = holding VC's buffer queue not the physical link
- We'll consider two options:
  - VC ordering
  - Escape VCs

Here, we are using VCs to deal with routing deadlocks.

Using separate VCs for different message types (e.g., requests and responses in coherence protocols) to avoid protocol-level deadlocks is a different story.

Option 1: VC Ordering





- All message sent through VC 0 until cross dateline
- After dateline, assigned to VC 1
  - Cannot be allocated to VC 0 again

Option 2: Escape VCs

- Enforcing order lowers VC utilization
  - Previous example: VC 1 **underutilized**
- Escape VCs
  - Have onde VC that uses *deadlock free routing*
  - Example: VC 0 uses DOR, other VCs use arbitrary routing function
  - Access to VCs arbitrated fairly: packet always has chance of landing on escape VC

Routing Algorithm Implementation

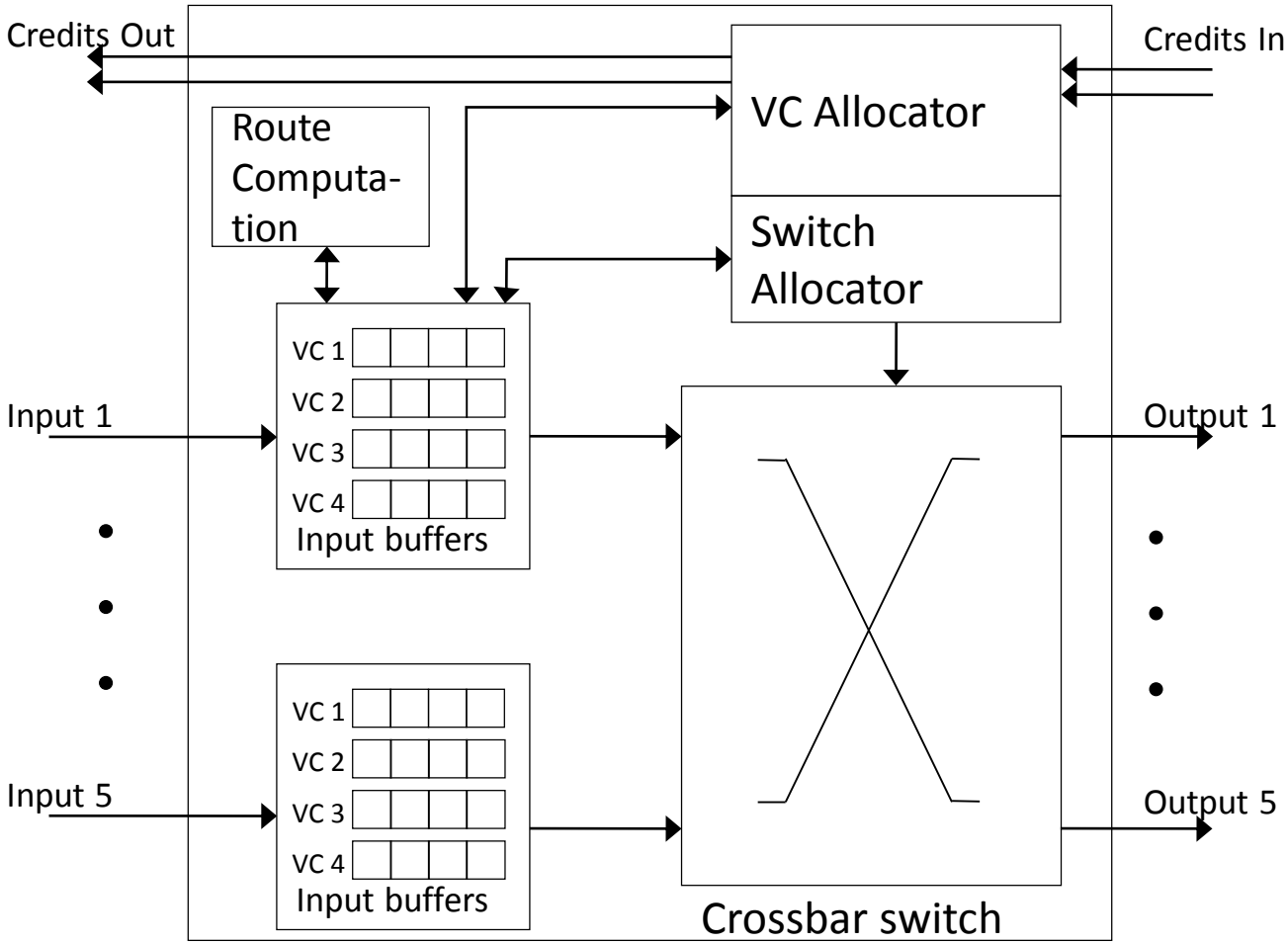
- **Source Tables**
  - Entire route specified at source
  - Avoids per-hop routing latency
  - Unable to adapt dynamically to network conditions
  - Support reconfiguration (not specific to topology)
  - Can specify multiple possible routes per destination
    - Select randomly or adaptively
- **Node Tables**
  - Store only next direction at each node
  - Smaller tables than source routing
  - Adds per-hop routing latency
  - Can specify multiple possible output ports per destination
- **Combinatorial circuits**
  - Simple (e.g., DOR): low router overhead
  - Specific to one topology and one routing algorithm
    - Limits fault tolerance

Router Microarchitecture

Router Microarchitecture Overview

- Focus on microarchitecture of Virtual Channel router
- Router complexity increase with bandwidth demands
  - Simple routers built when high throughput is not needed
    - Wormhole flow control, no virtual channels, DOR routing, unpipelined, …

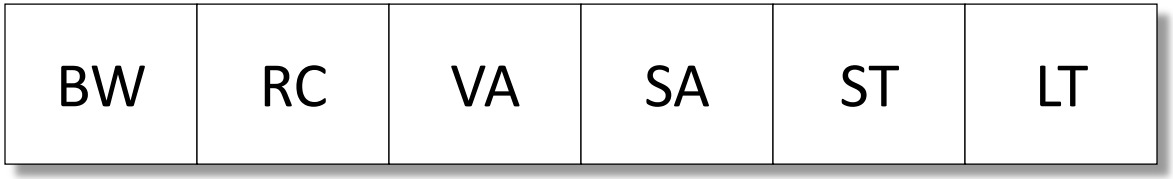
Virtual Channel Router



Router Components

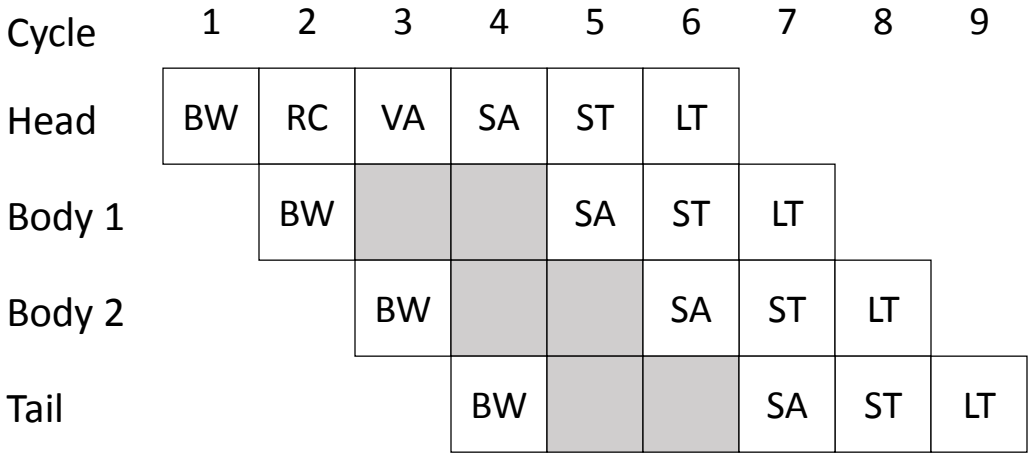
- Input buffers, route computation logic, virtual channel allocator, switch allocator, crossbar switch
- Most NoC routers are input buffered
  - Allows using single-ported memories
- Buffer store flits for duration in router

Baseline Router Pipeline



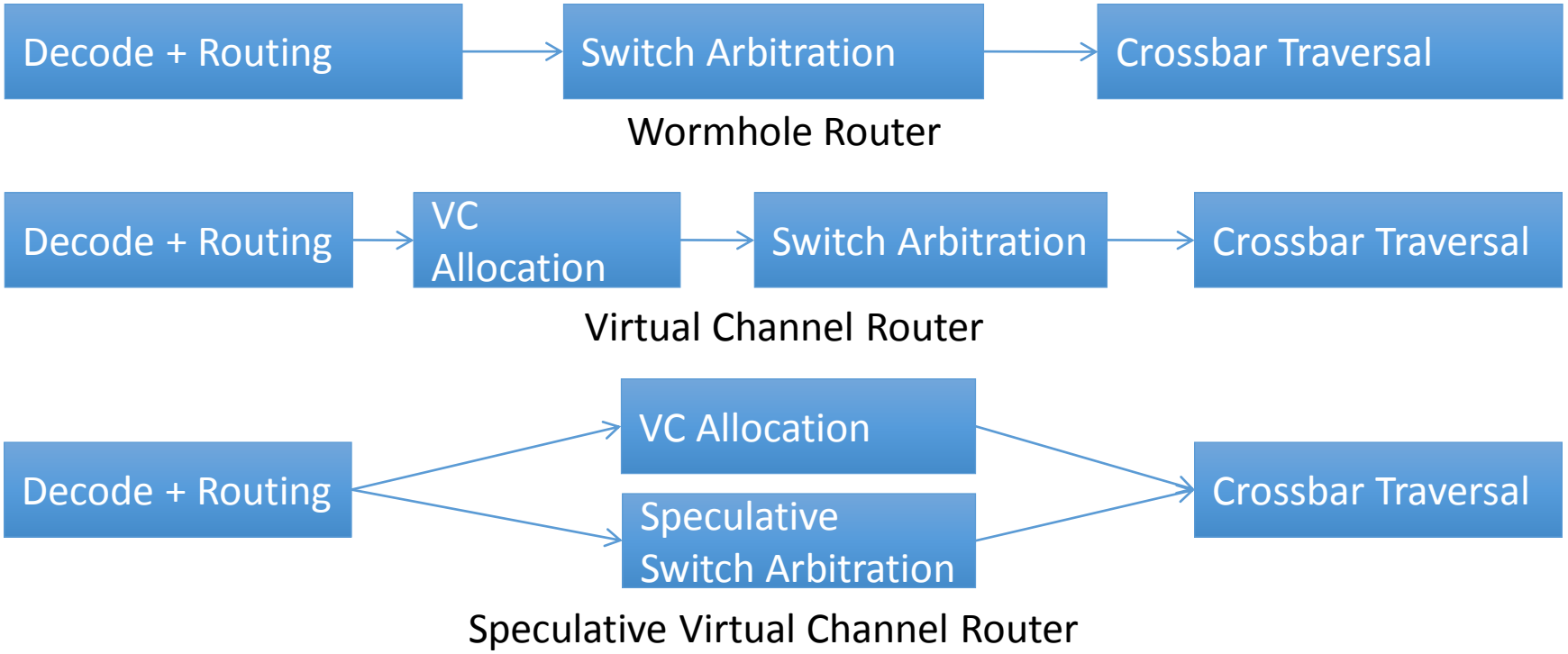
- Canonical *logical* router pipeline
  - Fit into physical stages based on target frequency and stage delays
  - **BW (Buffer Write)**: decode input VC and write to buffer
  - **RC (Route Computation)**: determine output port
  - **VA (VC Allocation)**: determine VC to use on the output port
  - **SA (Switch Allocation)**: arbitrate for crossbar in and out ports
  - **ST (Switch Traversal)**: once granted the output port, traverse the switch
  - **LT (Link Traversal)**: bon voyage!

Baseline Router Pipeline (2)



- Head flit goes through all 6 stages
- Body and Tail flits skip RC and VA
  - Route computation and VC allocation done only once per packet
  - Body and Tail flits inherit this info from the head flit
- Tail flit de-allocates the VC

Modules and Dependencies in Router



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

Router Pipeline Performance

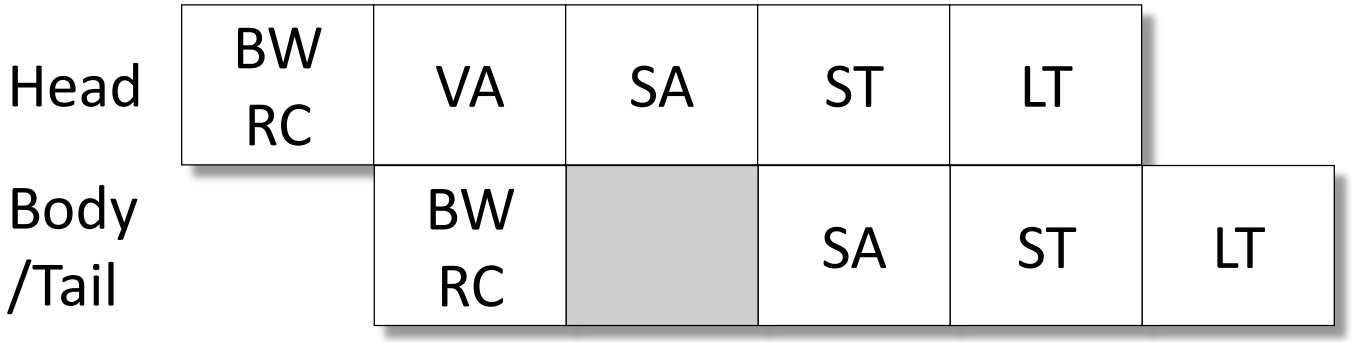
- Baseline (no load) delay

$$= (5\,cycles + link\ delay) \times hops + t_{serialization}$$

- Incurs routing delay, adding to message delay
  - Ideally, only pay link delay
- Also increases buffer turnaround time
  - Necessitates more buffers
  - Affects clock cycle time
- Techniques to reduce pipeline stages

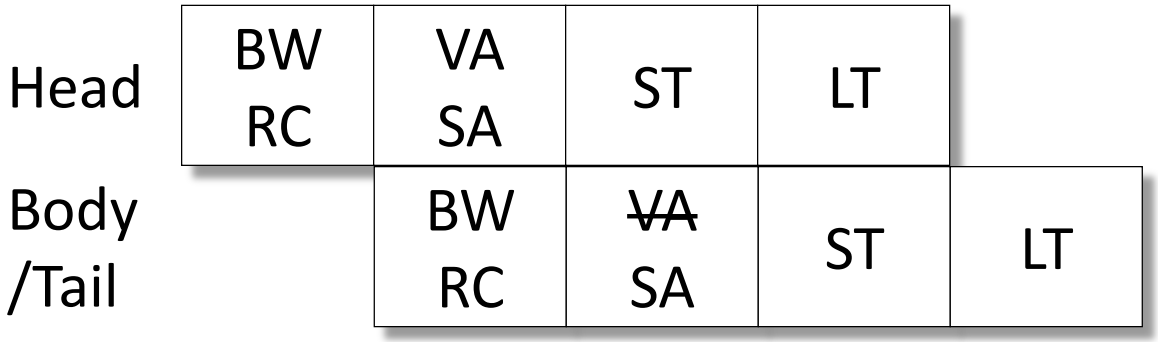
Optimizations: Lookahead Routing

- At current router perform routing computation for next router
  - Overlap with Buffer Write (BW)
  - Precomputing route allows flits to compete for VCs immediately after BW



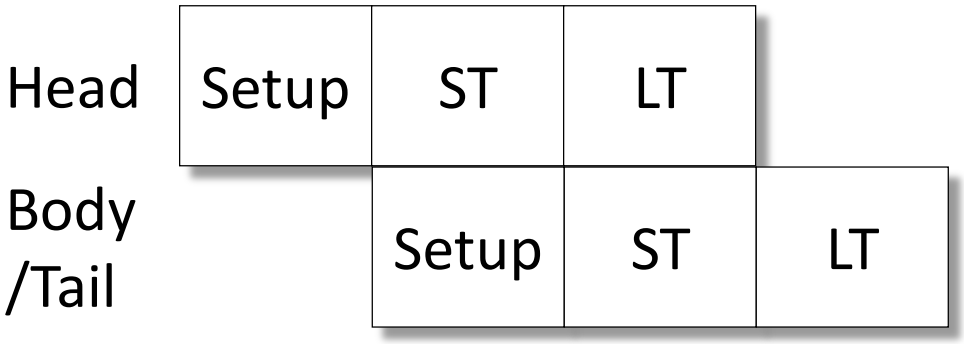
Pipeline Optimizations: Speculation

- Assume that VC Allocation will be successful
  - Valid under low to moderate loads
- Do VA and SA in parallel
- If VA unsuccessful (no virtual channel returned)
  - Must repeat VA/SA in next cycle
- Prioritize non-speculative requests
  - Body/tail flit already have VC info so they are not speculative

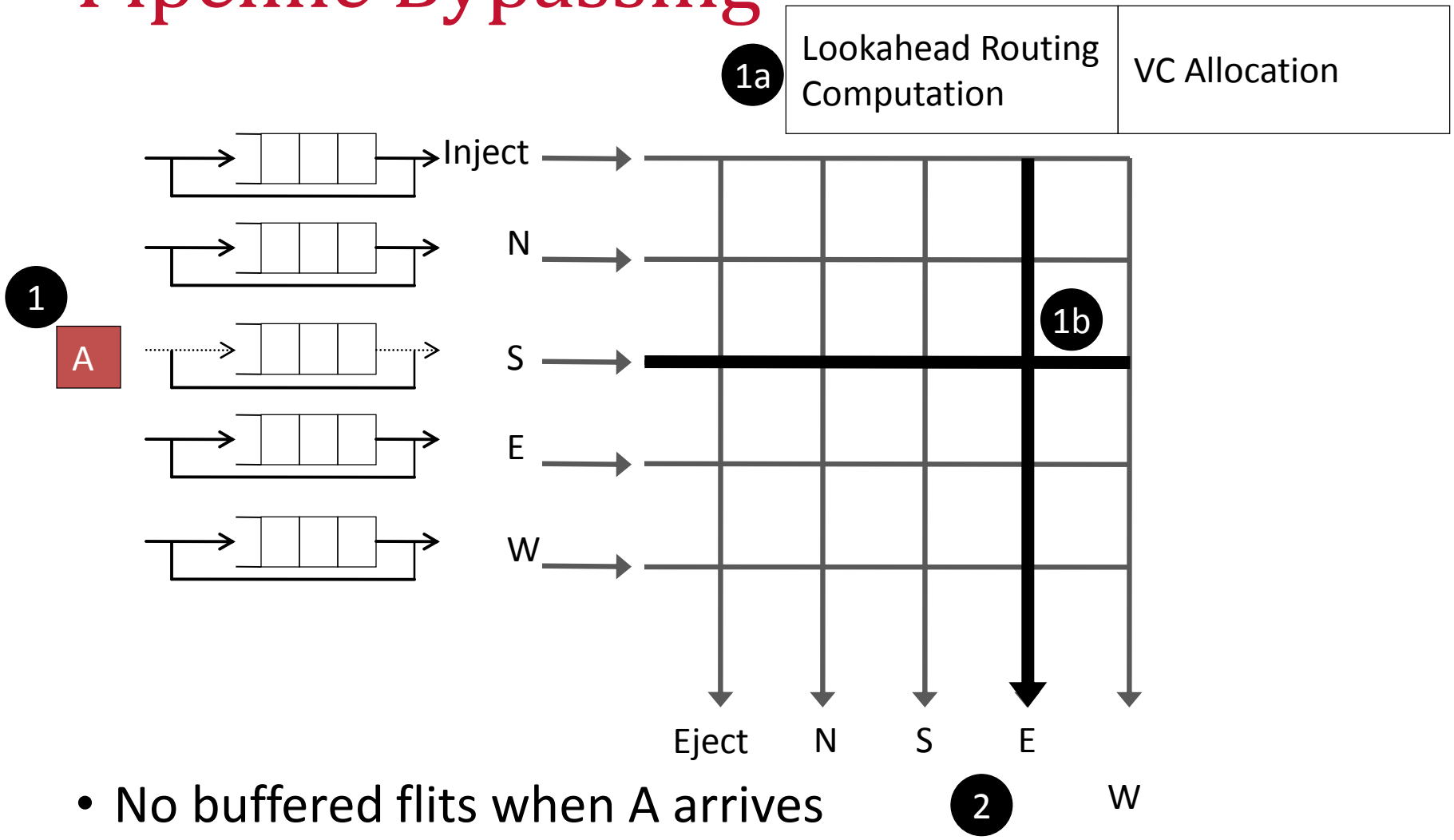


Pipeline Optimizations: Bypassing

- When no flits in input buffer
  - Speculatively enter ST
  - On port conflict, speculation aborted
  - In the first stage (setup)
    - Do lookahead routing → Just decode the head flit
    - Do SA and VA in parallel
    - Skip BW: do not write to buffer unless the speculation fails

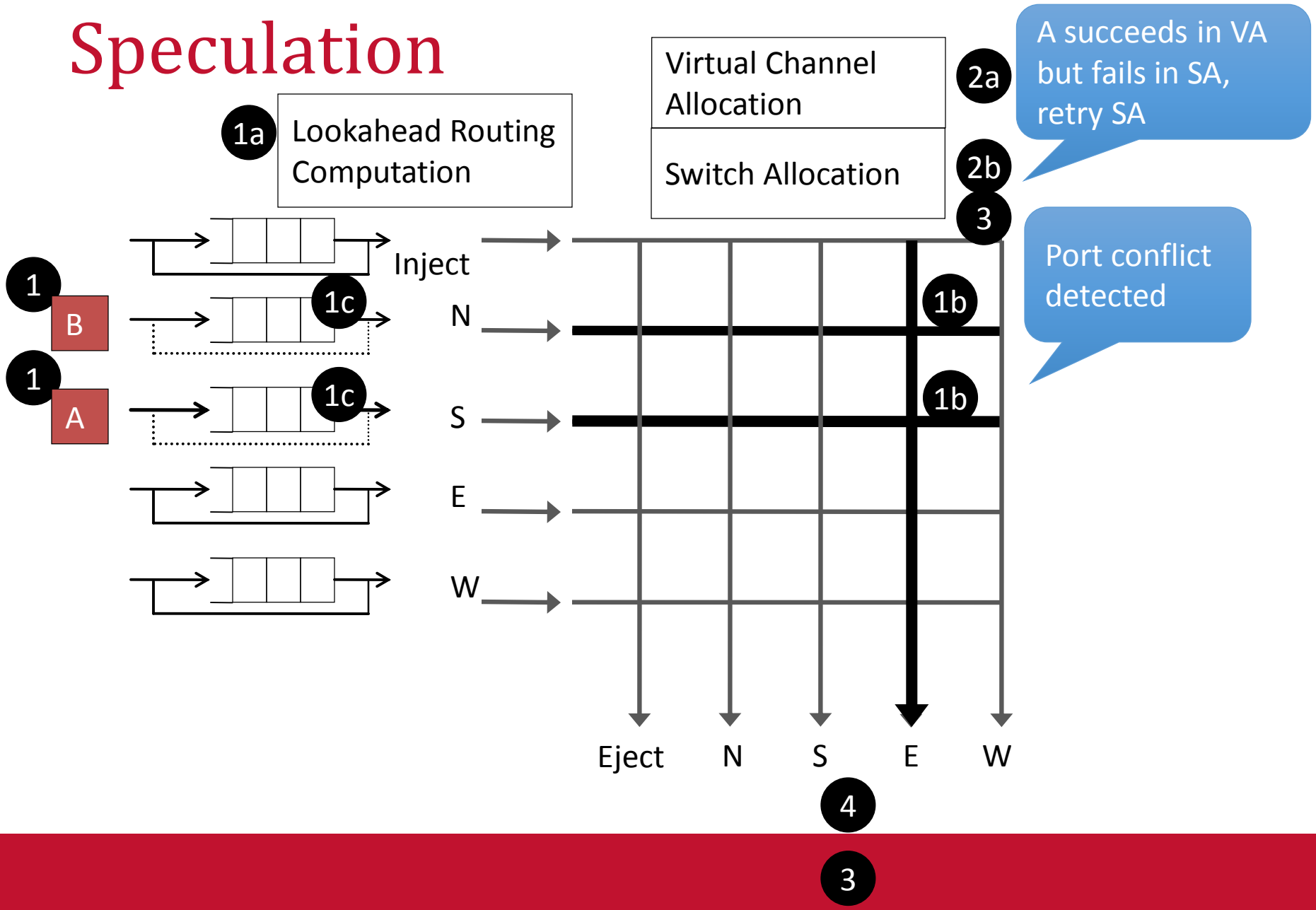


Pipeline Bypassing

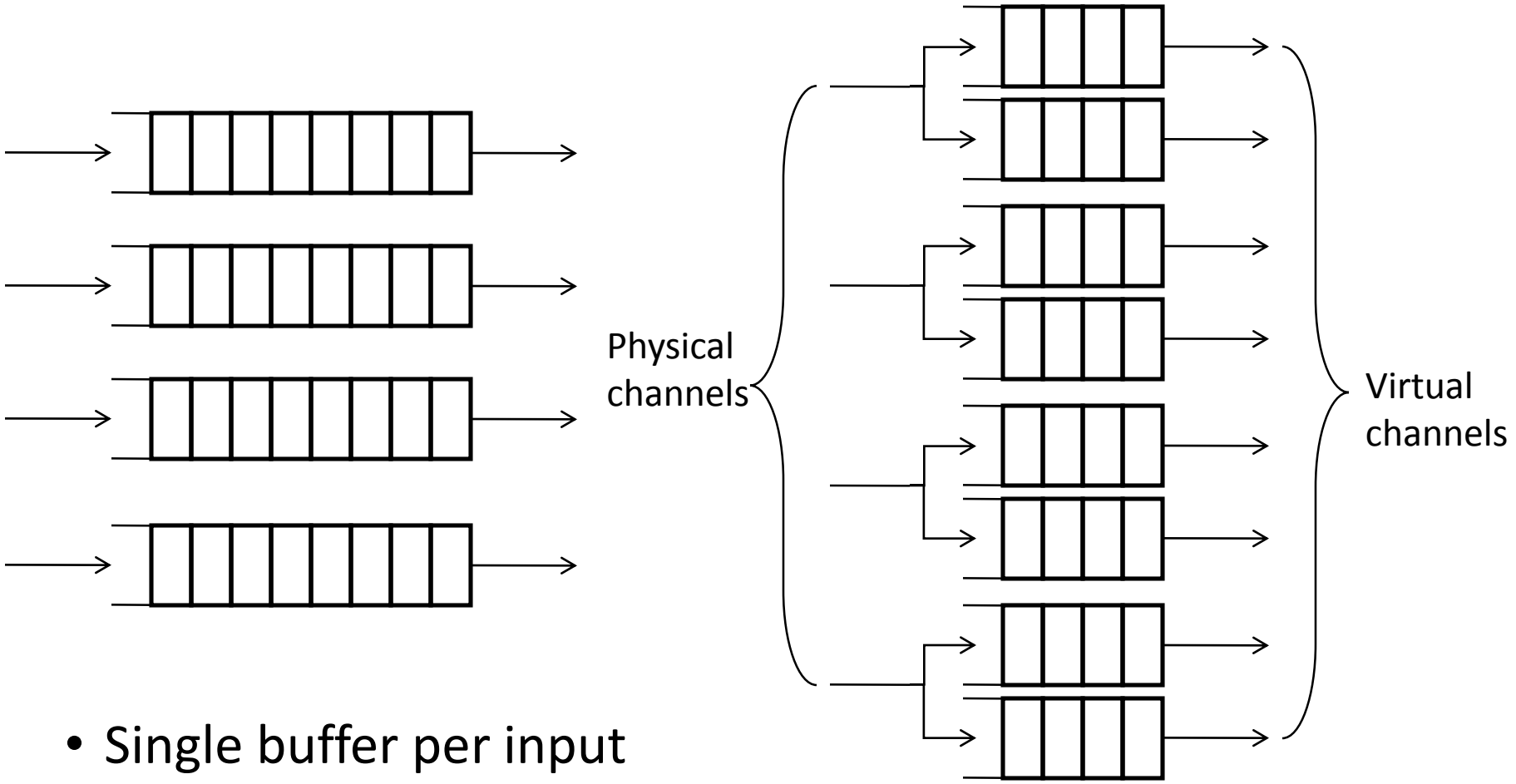


- No buffered flits when A arrives

Speculation

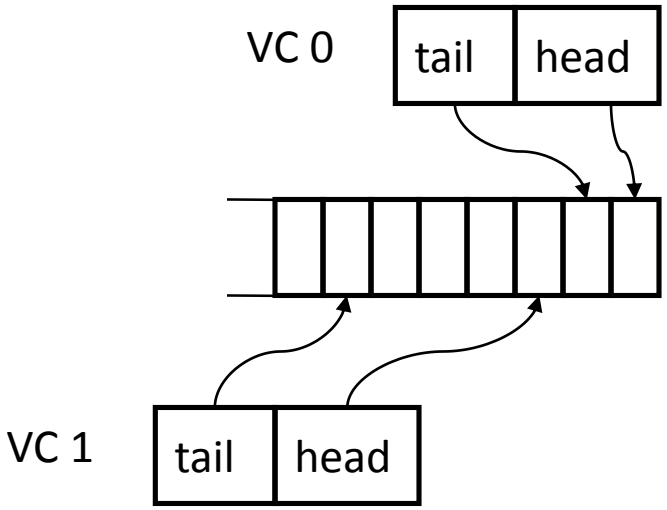


Buffer Organization



- Single buffer per input
- Multiple fixed length queues per physical channel

Buffer Organization



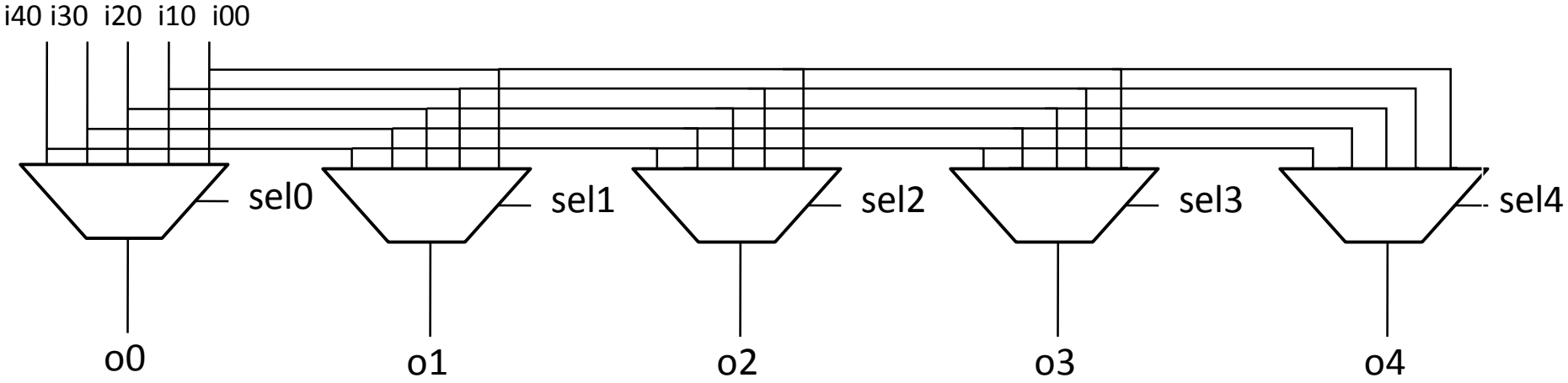
- Multiple variable length queues
  - Multiple VCs share a large buffer
  - Each VC must have minimum 1 flit buffer
    - Prevent deadlock
  - More complex circuitry

Buffer Organization

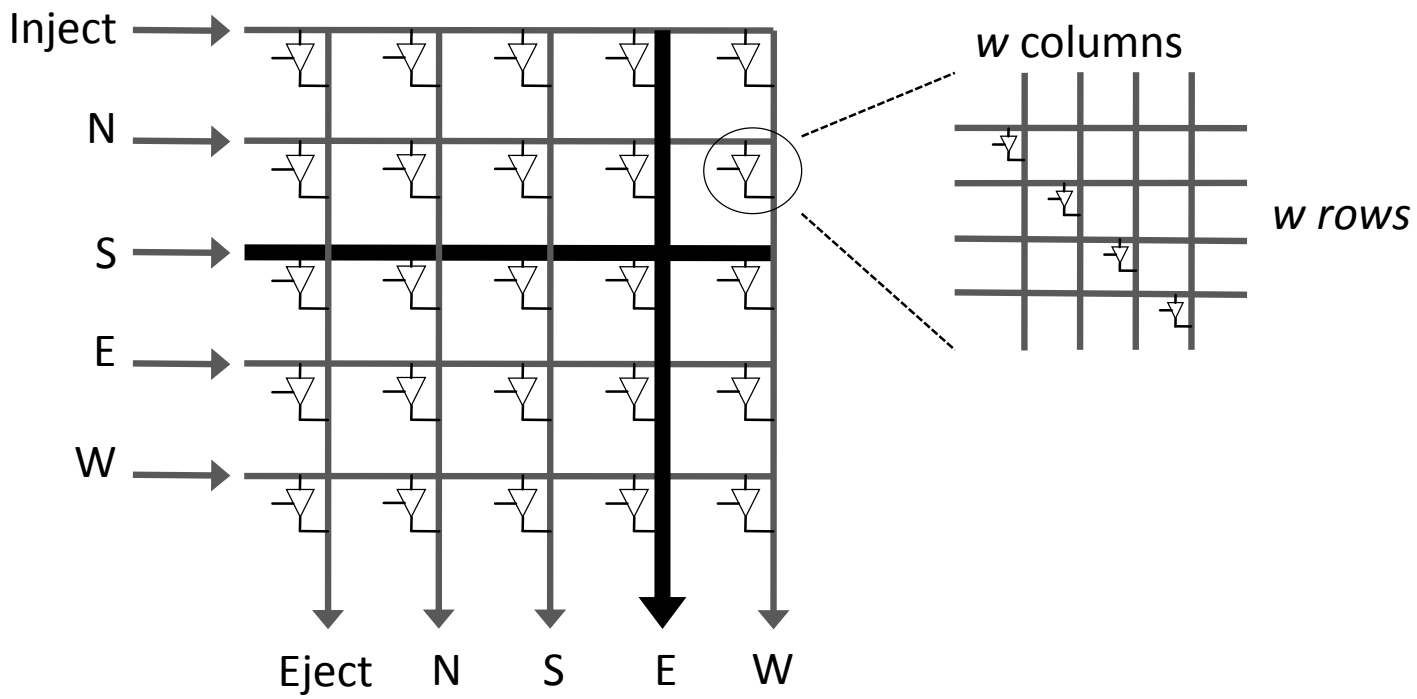
- Many shallow VCs or few deep VCs?
- More VCs ease HOL blocking
  - More complex VC allocator
- Light traffic
  - Many shallow VCs – underutilized
- Heavy traffic
  - Few deep VCs – less efficient, packets blocked due to lack of VCs

Crossbar Organization

- Heart of data path
  - Switches bits from input to output
- High frequency crossbar designs challenging
- Crossbar composed for many multiplexers
  - Common in low-frequency router designs

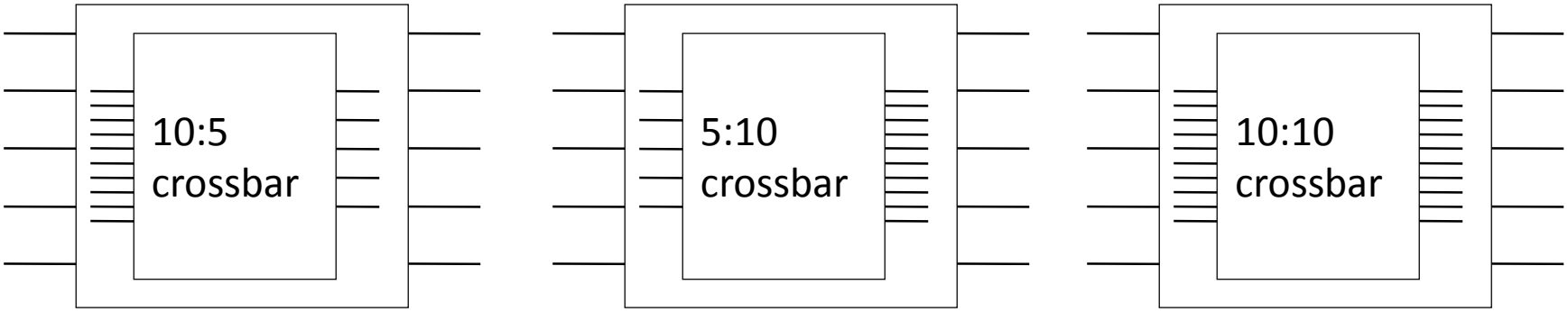


Crossbar Organization: Crosspoint



- Area and power scale at $O((pw)^2)$
  - p: number of ports (function of topology)
  - w: port width in bits (determines phit/flit size and impacts packet energy and delay)

Crossbar speedup



- Increase internal switch bandwidth
- Simplifies allocation or gives better performance with a simple allocator
  - More inputs to select from → higher probability each output port will be matched (used) each cycle
- Output speedup requires output buffers
  - Multiplex onto physical link

Arbiters and Allocators

- ***Allocator***: matches N requests to M resources
- ***Arbiter***: matches N requests to 1 resource
- Resources
  - VCs (for virtual channel routers)
  - Crossbar switch ports

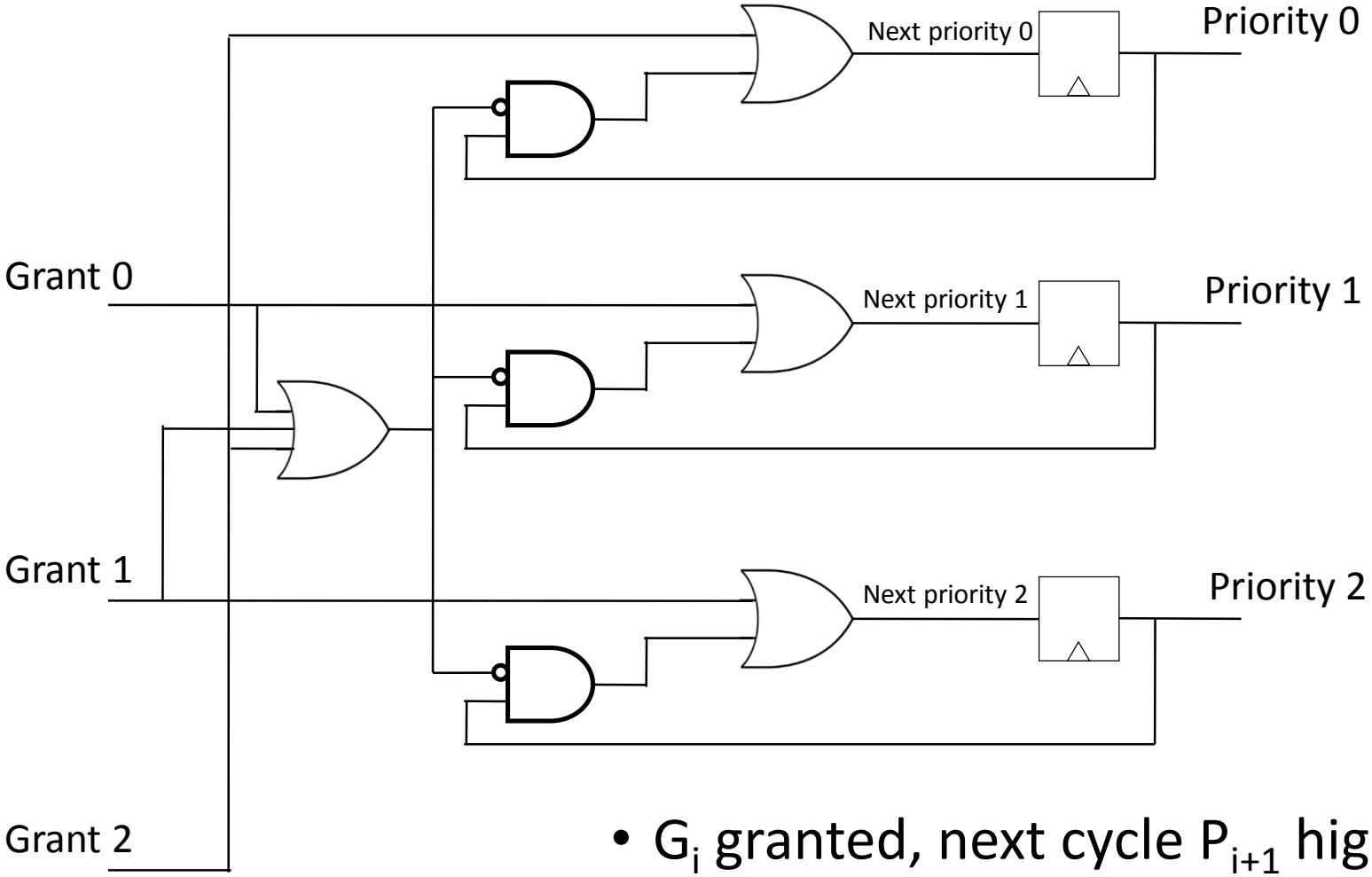
Arbiters and Allocators (2)

- VC allocator (VA)
  - Resolves contention for output virtual channels
  - Grants them to input virtual channels
- Switch allocator (SA)
  - Grants crossbar switch ports to input virtual channels
- Allocator/arbiter that delivers high matching probability translates to higher network throughput
  - Must also be fast and/or able to be pipelined

Round Robin Arbiter (1)

- Last request serviced given lowest priority
- Generate the next priority vector from current grant vector
- Exhibits fairness

Round Robin Arbiter (2)

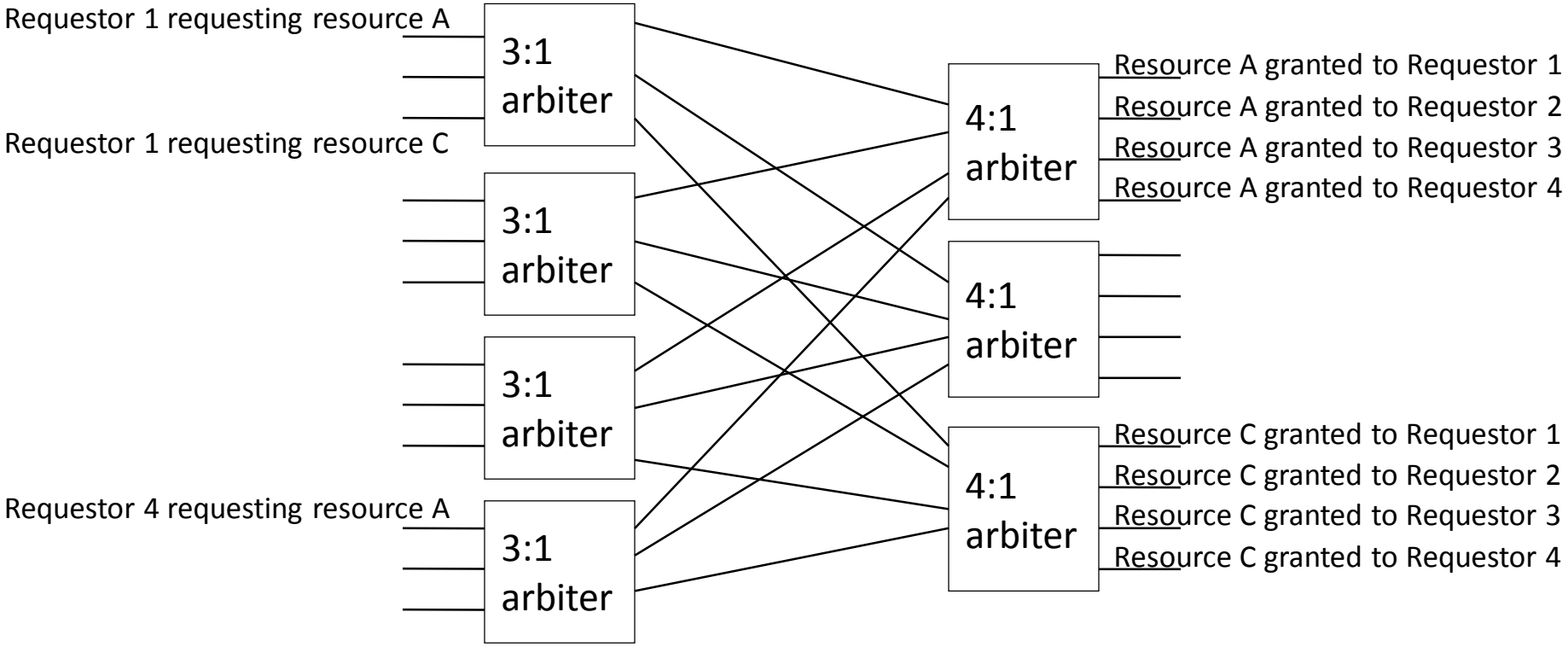


- G_i granted, next cycle P_{i+1} high

Separable Allocator

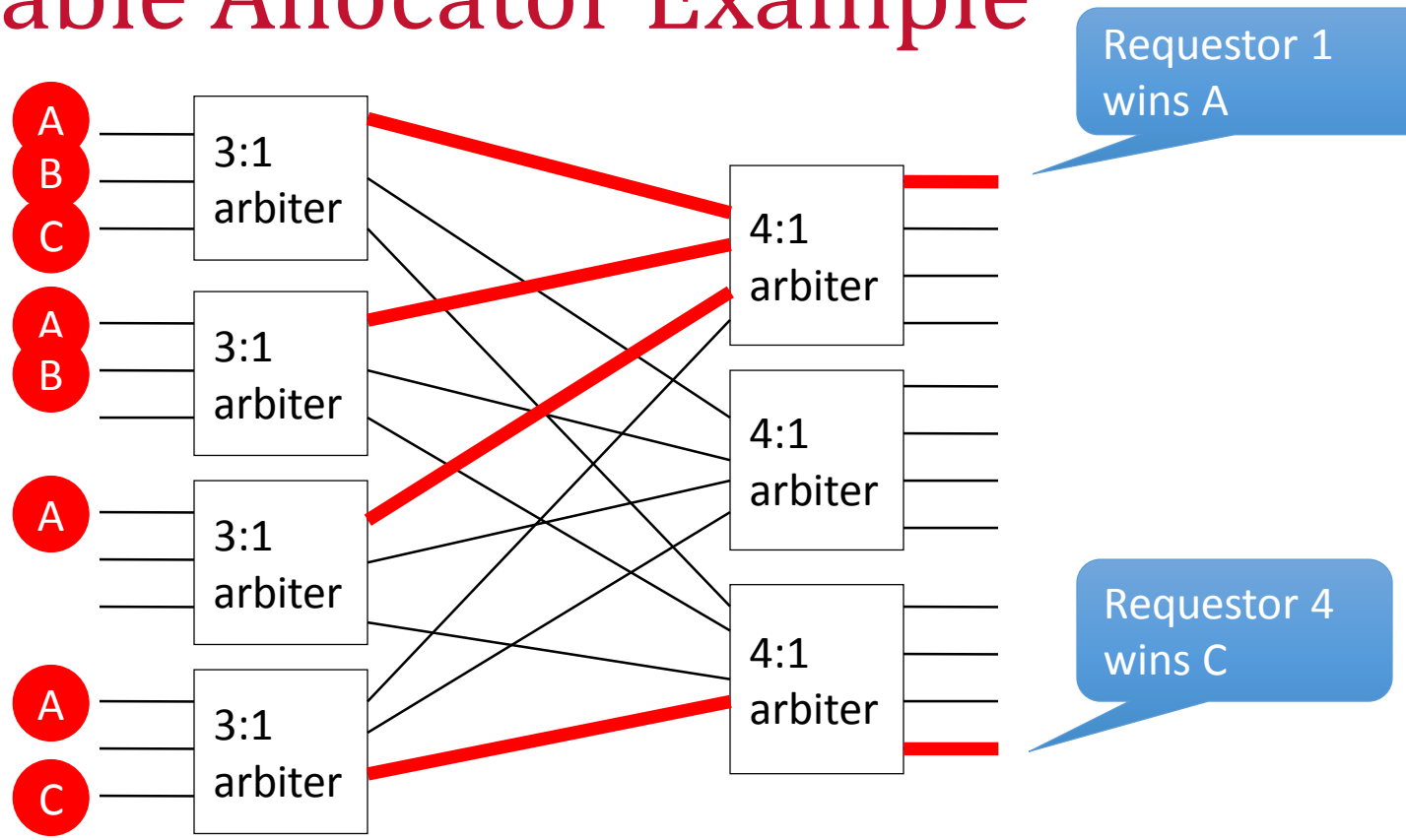
- Need for simple, pipeline-able allocators
- Allocator composed of arbiters
  - Arbiter chooses one out of N requests to a single resource
- Separable switch allocator
  - First stage: select single request at each input port
  - Second stage: selects single request for each output port

Separable Allocator



- A 3:4 allocator
- First stage: 3:1 – ensures only one grant for each input
- Second stage: 4:1 – only one grant asserted for each output

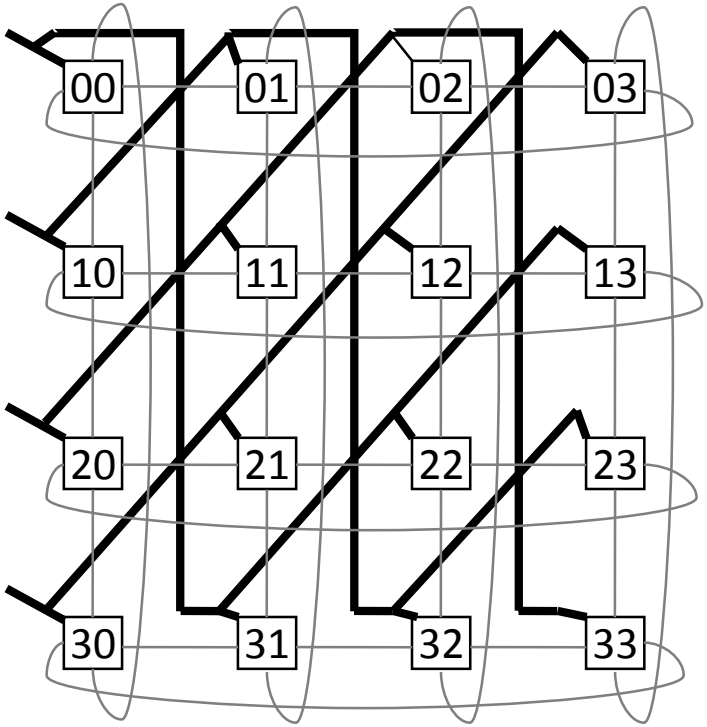
Separable Allocator Example



- 4 requestors, 3 resources
- Arbitrate locally among requests
  - Local winners passed to second stage

Wavefront Allocator

- Arbitrates among requests for inputs and outputs simultaneously
- Row and column tokens granted to diagonal group of cells
- If a cell is requesting a resource, it will consume row and column tokens
  - Request is granted
- Cells that cannot use tokens pass row tokens to right and column tokens down



front Allocator Example

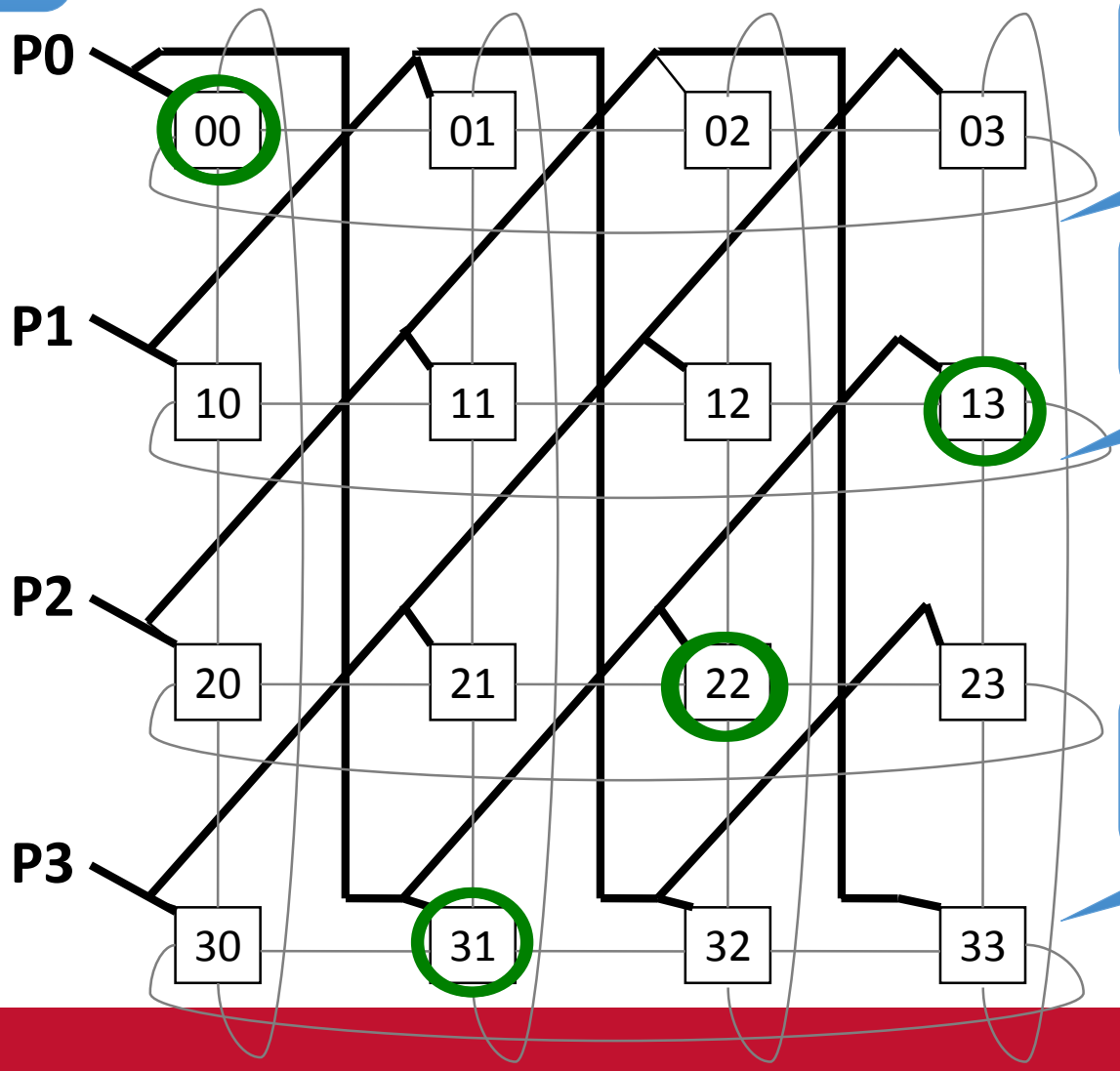
Tokens inserted at P0

A requesting Resources 0, 1 ,2

B requesting Resources 0, 1

C requesting Resource 0

D requesting Resources 0, 2

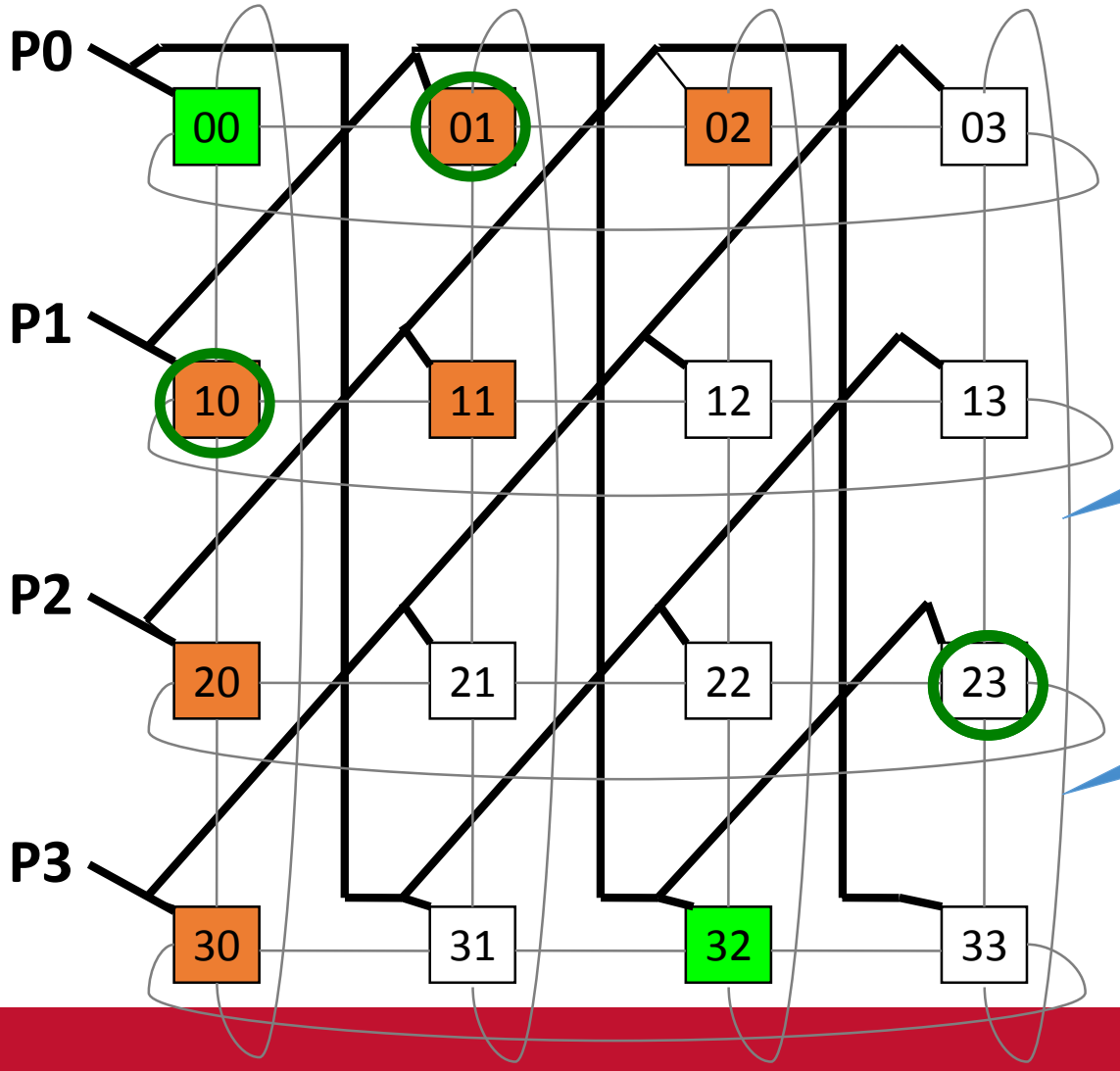


Entry [0,0] receives grant, consumes token

Remaining tokens pass down and right

[3,2] receives 2 tokens and is granted

Wavefront Allocator Example



VC Allocator Organization

- Depends on routing function
- If routing function returns single VC
  - VCA need to arbitrate between input VCs contending for same output VC
- If routing function returns multiple candidate VCs (for same physical channel)
  - Needs to arbitrate among ***v*** first stage requests before forwarding winning request to second stage

Adaptive Routing & Allocator Design

- Deterministic routing
  - Single output port
  - Switch allocator bids for output port
- Adaptive routing
  - Option 1: returns multiple candidate output ports
    - Switch allocator can bid for all ports
    - Granted port must match VC granted
  - Option 2: Return single output port
    - Reroute if packet fails VC allocation

Speculative VC Router

- Non-speculative switch requests must have higher priority than speculative ones
- Two parallel switch allocators
  - 1 for speculative, 1 for non-speculative
  - From output, choose non-speculative over speculative
- Possible for flit to succeed in speculative switch allocation but fail in VC allocation
  - Done in parallel
  - Speculation incorrect → Switch reservation is wasted
  - Body and Tail flits: non-speculative switch requests
    - Do not perform VC allocation → inherit VC from head flit