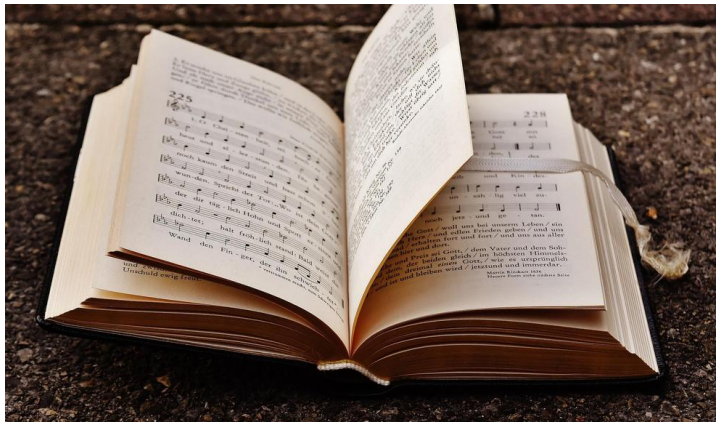


Chapter 8:
he On-chip Network

Background Required to Understand this Chapter

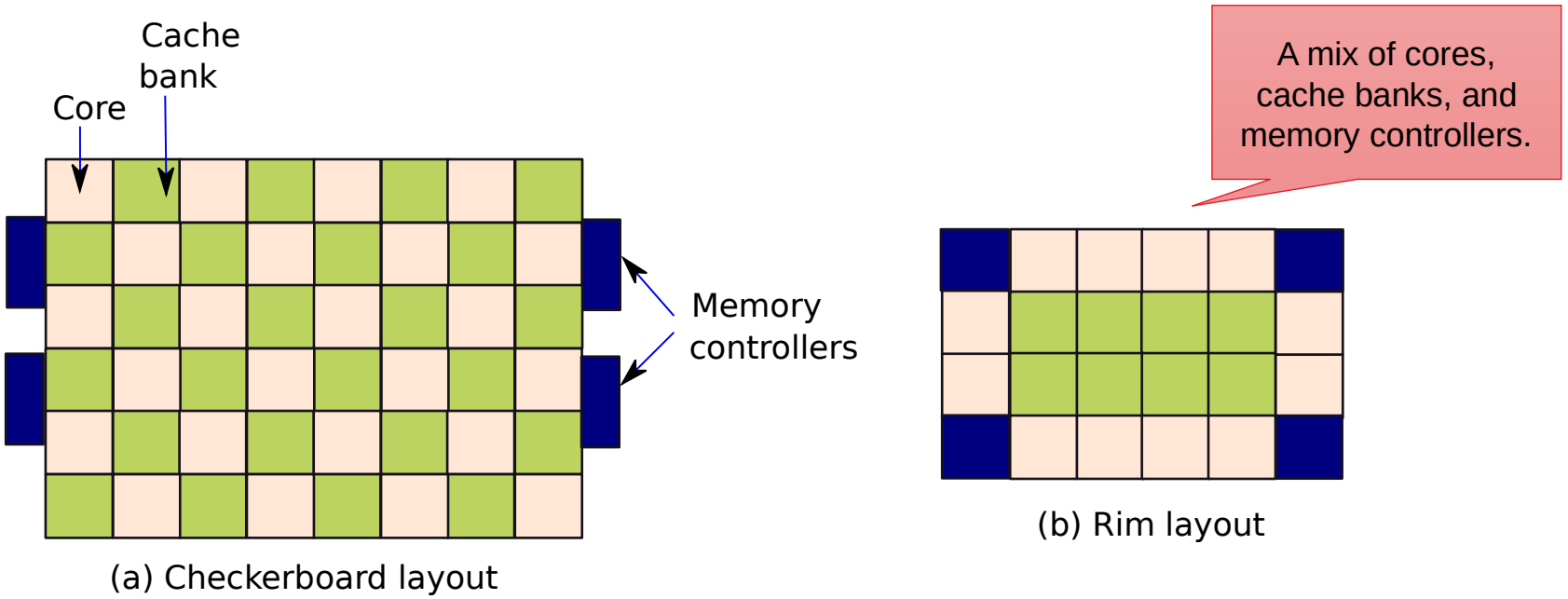
- Pipelining
- Caches
- Basic graph theory

Chapter 2 and 7

Contents

1. Overview of an NoC
2. Message Transmission
3. Routing
4. Design of a Router
5. Non-Uniform Cache Architectures
6. Performance Aspects

Layout of a Modern Chip



Different layouts are suitable for different kinds of workloads.

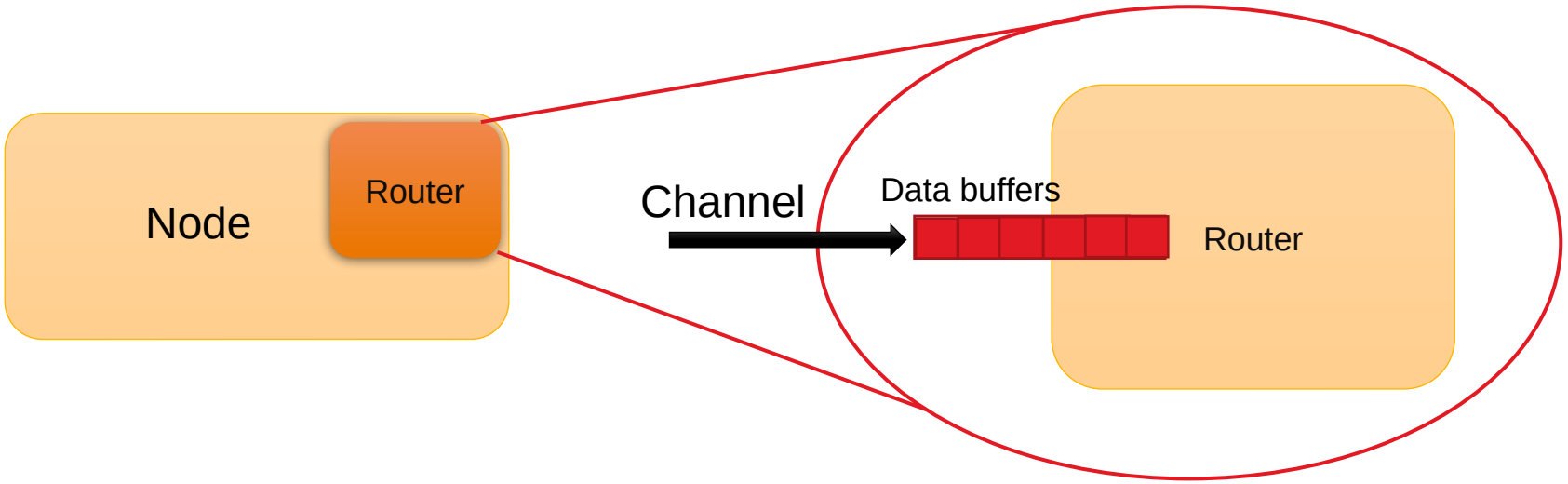
Definition of a Node

Any object that can send or receive a message.

- Core
- Memory Controller
- Cache Bank
- Accelerator

Router

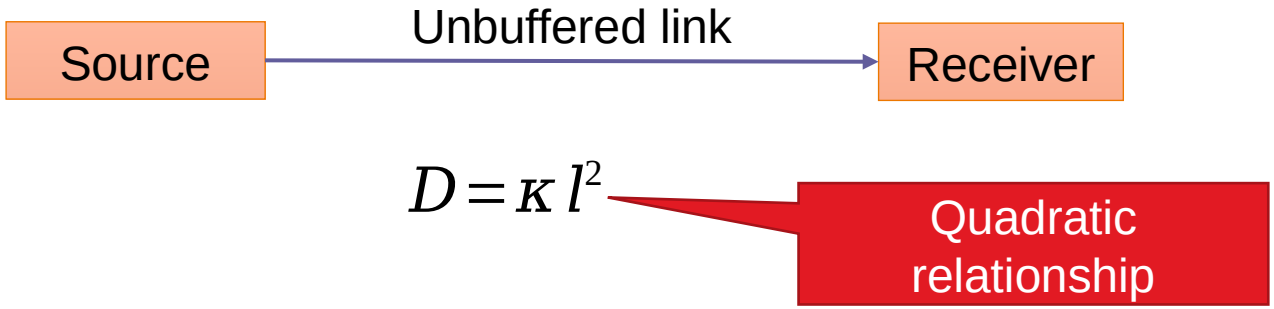
Every communicating node has a router that orchestrates the communication on its behalf.



Key Question

How do the routers communicate with each other?

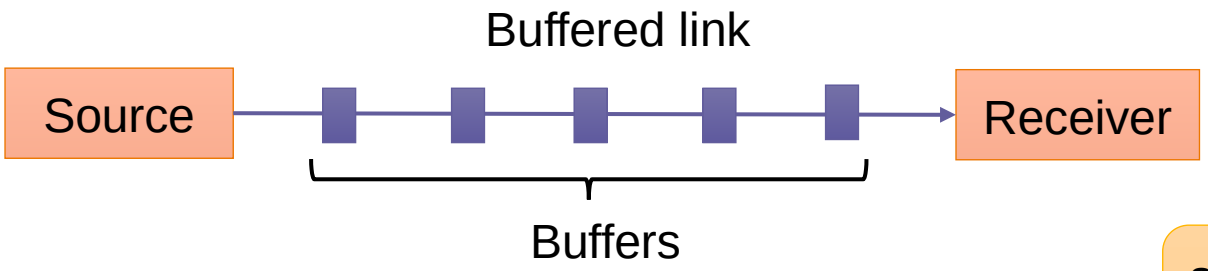
Connection between two routers with a single link.



- D is the delay
- l is the length of the wire

Long wires are very slow

Buffered Wires

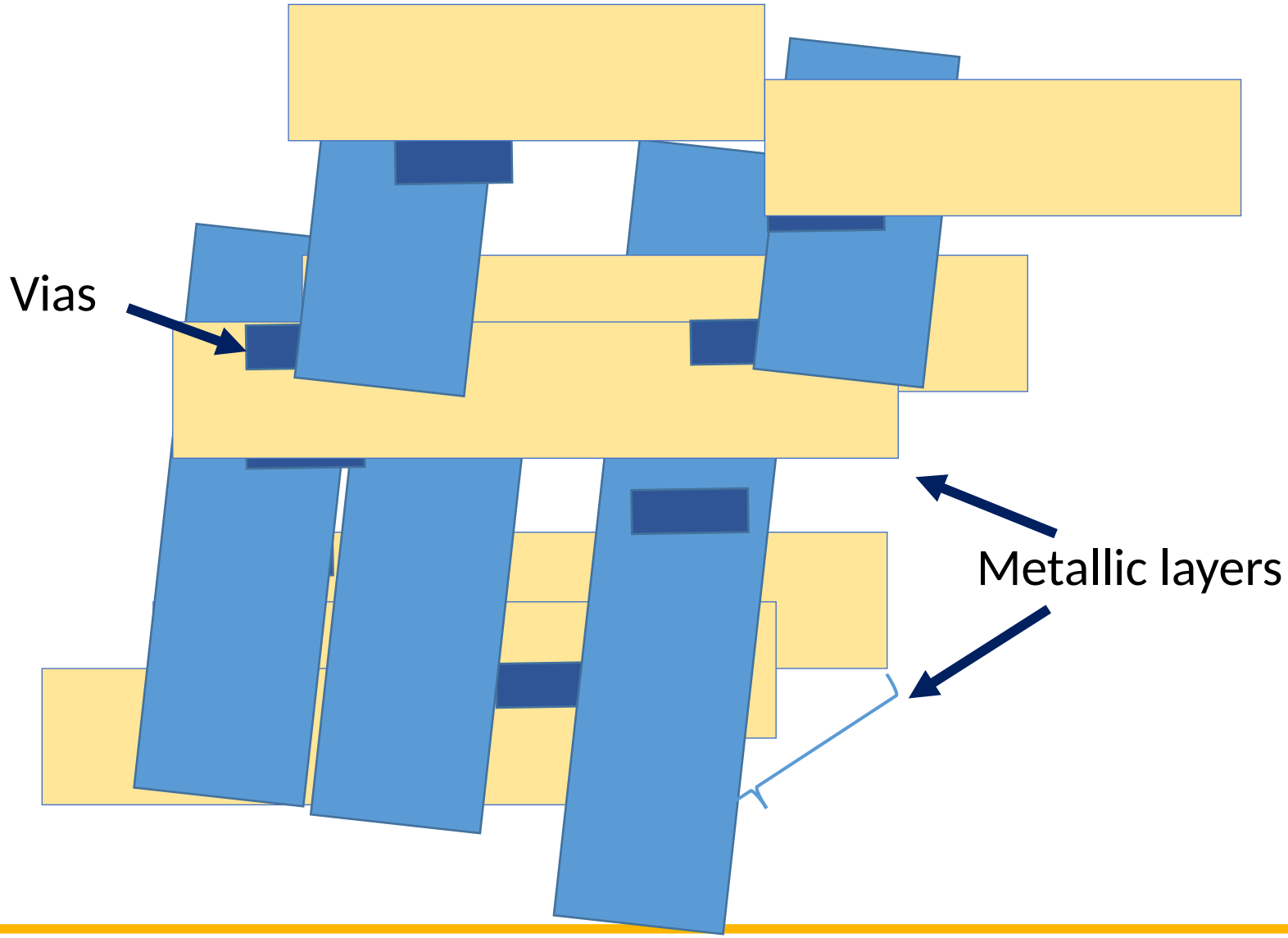


s = number of segments
d = delay of a buffer

$$D = 2\sqrt{\kappa d}.l - d$$

Delay is linear in terms of the length

Multi-layer Interconnects



Types of Interconnects in Silicon Chips

1. Local wires ⎵ These are used to connect transistors within the same functional unit.

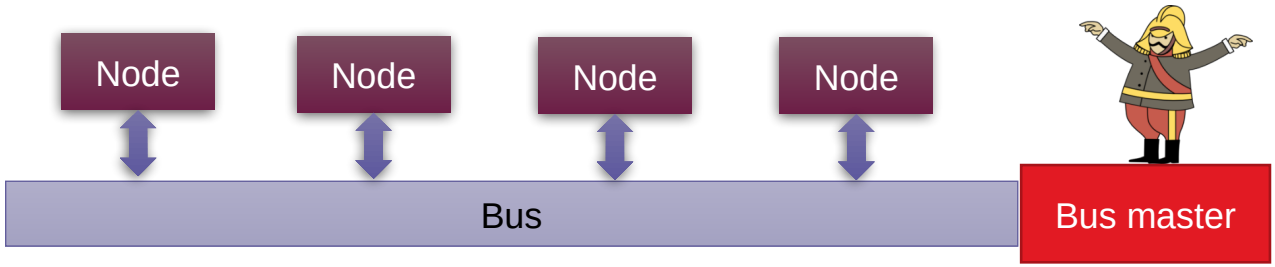
2. Intermediate wires ⎵ Used to connect transistors across adjoining functional units.

3. Global wires ⎵ Used to connect transistors across the chip.

Most intermediate and global wires are typically buffered.

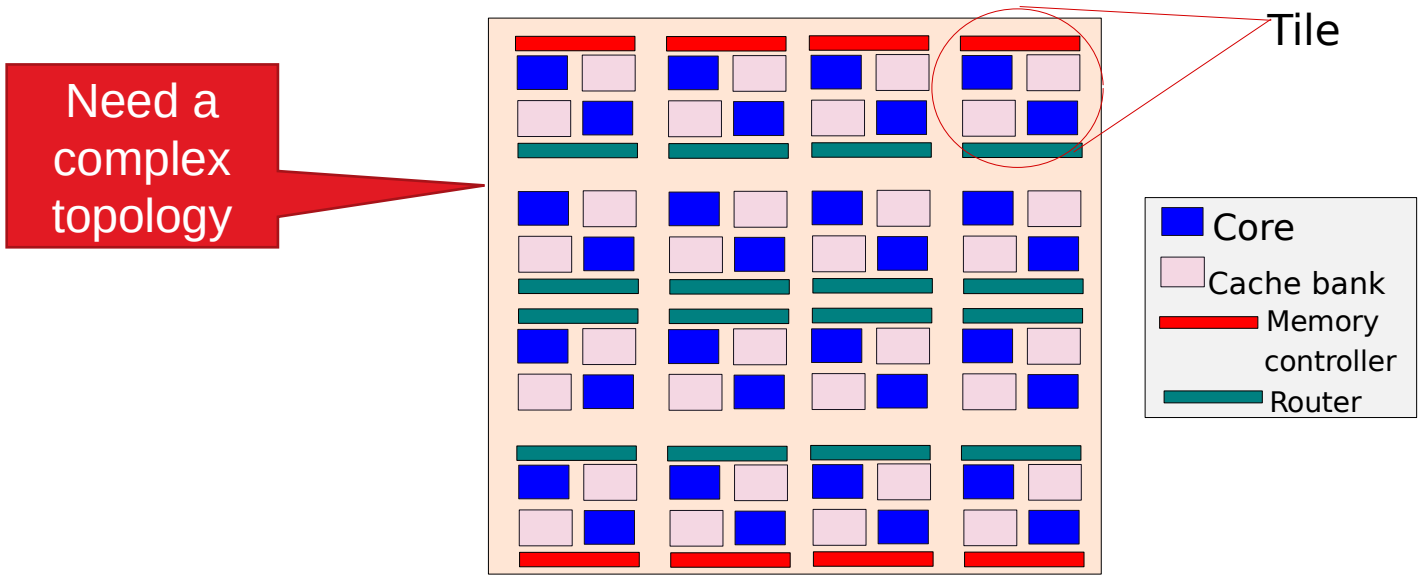
Simple yet Naïve Solution

Connect the routers using a bus (single set of copper wires)



- The bus master needs to grant permission to a node to transmit a message.
- It becomes a point of contention.
- Slow and not scalable

- Layout of a multicore processor



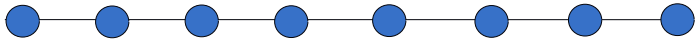
Create a Network on Chip (NoC)

- Arrange a set of nodes as tiles
- A router sends and receives all the messages for its tile
- A router also forwards messages originating at other routers to their destination
- Routers are referred to as nodes. Adjacent nodes are connected with links.
- The routers and links form the on chip network, or NoC.

- Bisection Bandwidth
  - Number of links that need to be snapped to divide an NoC into two equal parts (ignore small additive constants). Indicative of path diversity between two nodes. More is the path diversity more is the congestion tolerance.
- Diameter
  - Maximum optimal distance between any two pair of nodes (again ignore small additive constants)
- **Aim** : Maximise bisection bandwidth, minimise diameter

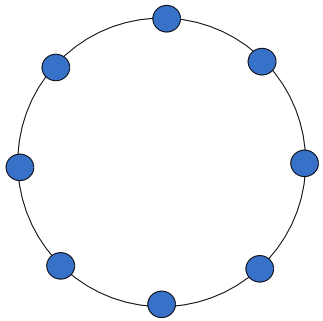
Example Topologies

Chain and Ring



Chain

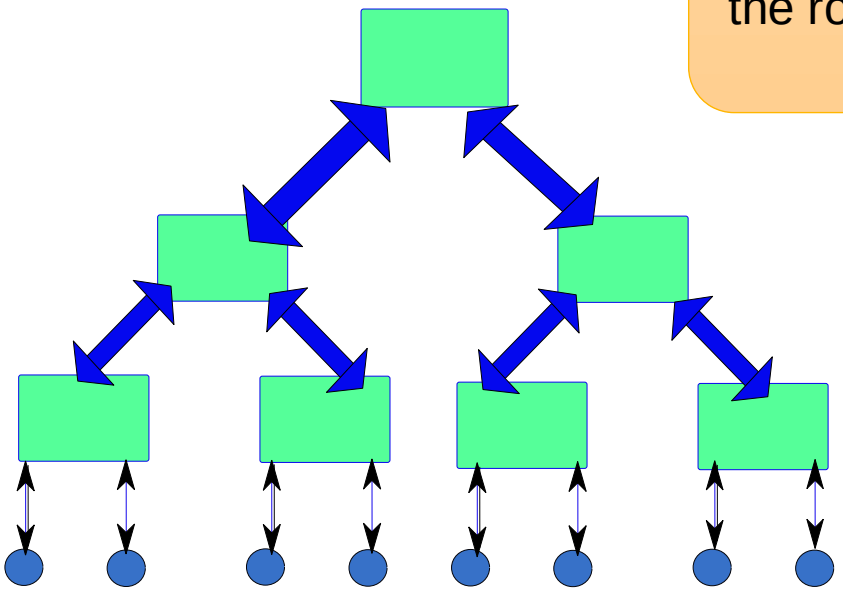
No path diversity



Ring

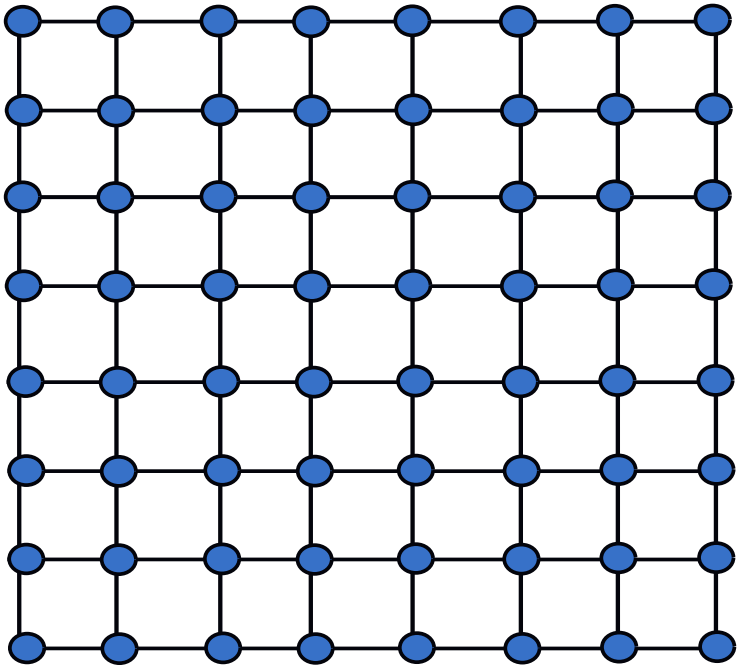
Limited path diversity

Fat Tree



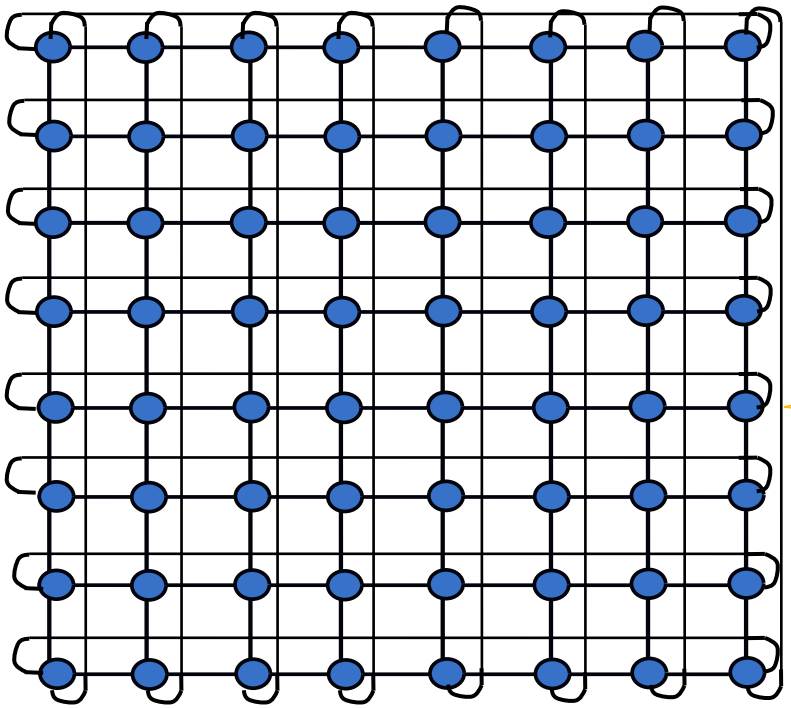
Links get wider towards the root. This ensures that the root does not become a bottleneck

Mesh



Every node has a degree of 4 other than at the rims and corners

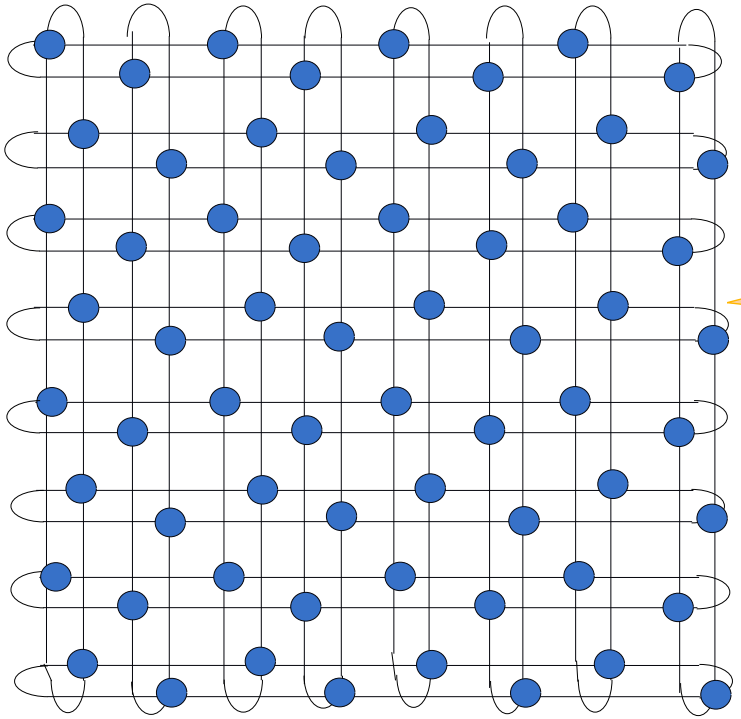
Torus



Every node has a degree of 4

The long links cause an issue.

Folded Torus



Problem of long links gets solved.

High-Radix Topologies

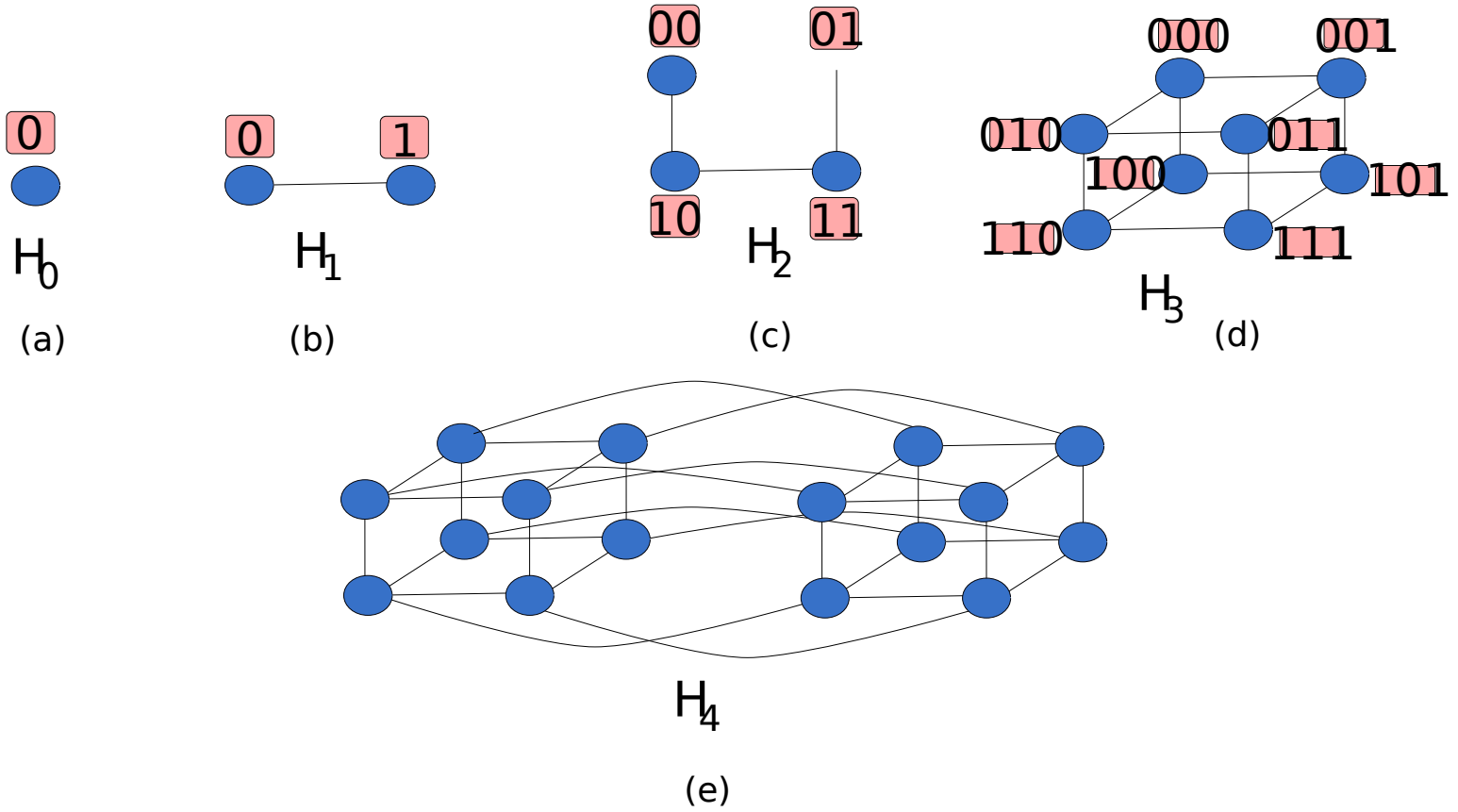
Nodes are connected to more than 4 neighbors

- Additional path diversity
- Low diameter
- Hard to implement it in an NoC

Much easier to implement in a cluster computer

- Hypercubes
- Clos network
- Butterfly

Hypercube


0
H_0
(a)
0
1
H_1
(b)
00
01
10
11
H_2
(c)
000
001
010
011
100
101
110
111
H_3
(d)
H_4
(e)

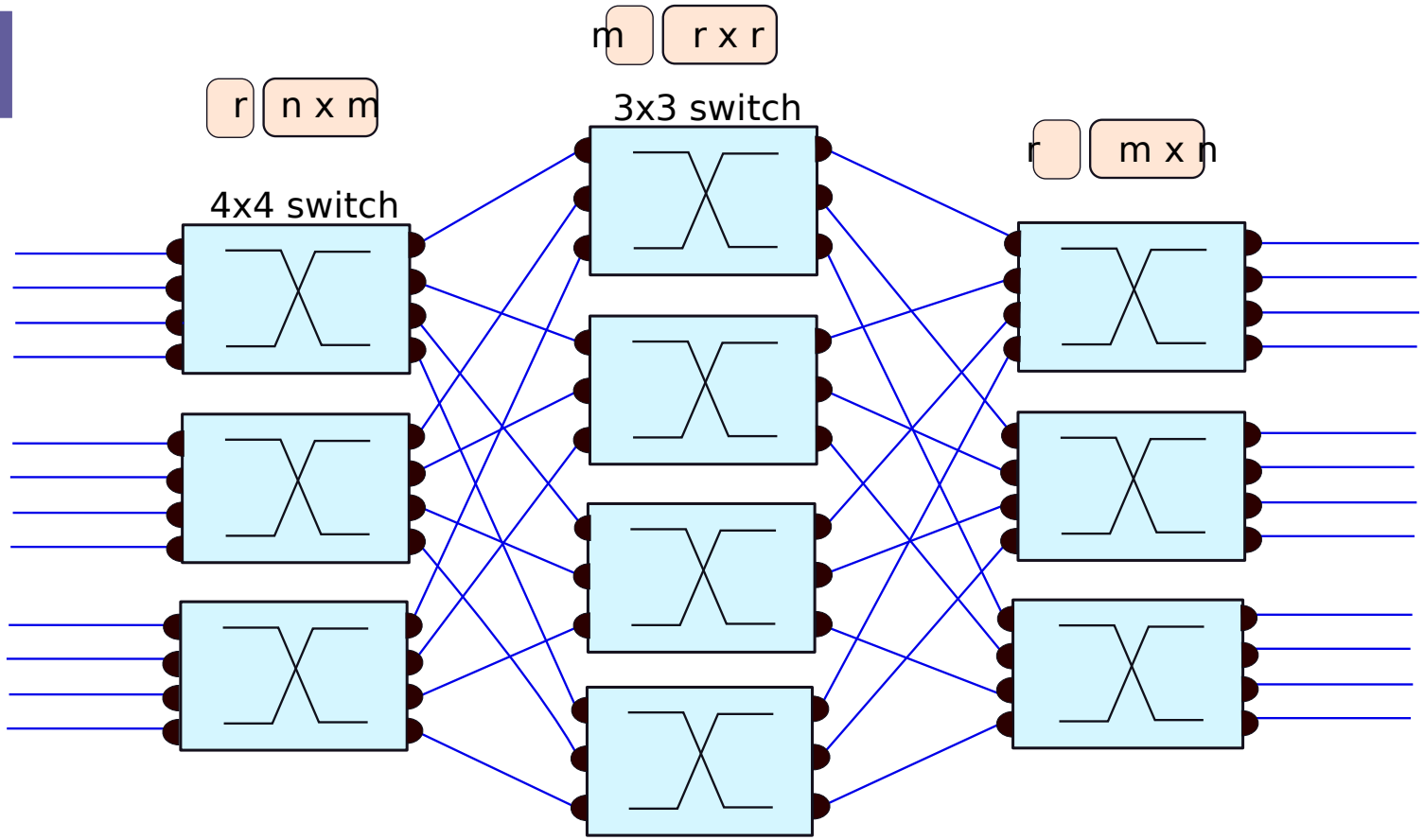
Properties of a Hypercube

* If a hypercube has *N* nodes
  * Diameter $= \log_2(N)$
  * Bisection bandwidth $= N/2$
* It is a recursive data structure
  * Connect two copies of an order (N-1) hypercube
    * by connecting corresponding nodes together.

Clos Network (*nr* inputs and outputs)

Three-layer network

r = 3
m = 4
n = 4

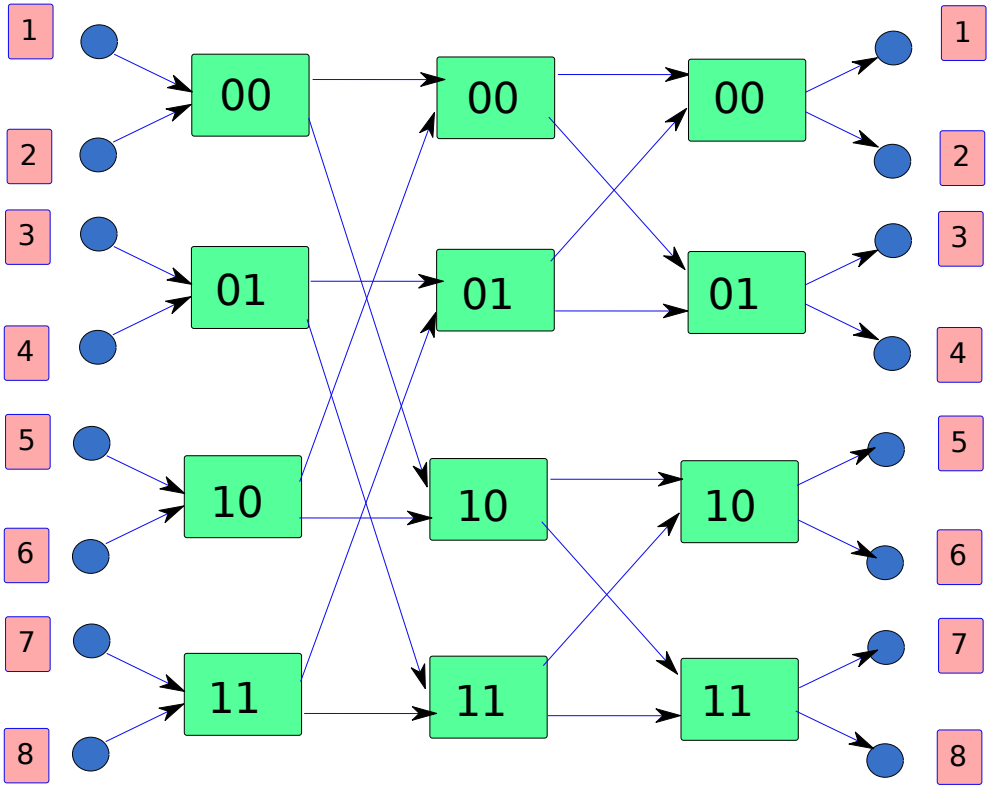


Properties of a Clos Network

* If m >= n, we can always connect an unused input terminal to an unused output terminal by rearranging the rest of the connections.
* If m >= 2n – 1, we can do the same without rearranging the rest of the connections.

Butterfly

Only uses 2x2 switches



Efficient implementations with high-radix routers exist.

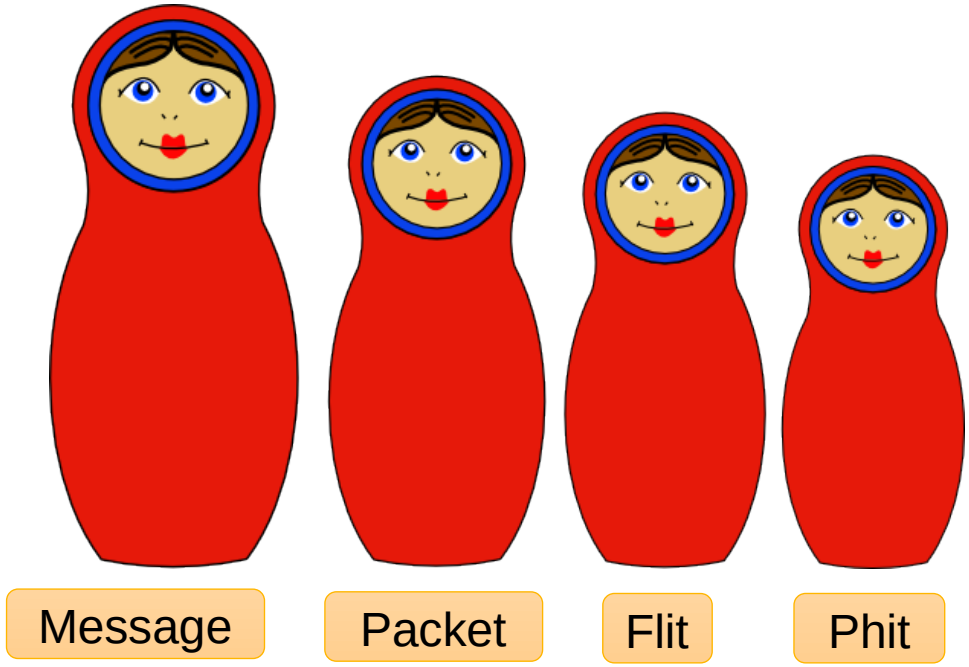
Summary

Topology	#Switches	#Links	Diameter	Bisection Bandwidth
Chain	0	N-1	N-1	1
Ring	0	N	N/2	2
Fat Tree	N-1	2N-2	2 log(N)	N/2*
Mesh	0	2N - 2√N	2√N – 2	√N
Torus	0	2N	√N	2√N
Folded Torus	0	2N	√N	2√N
Hybercube	0	N log(N)/2	log(N)	N/2
Butterfly	N log(N)/2	N + N log(N)	log(N) + 1	N/2
* Assume that the size of each link is equal to the number of leaves in its subtree				

Contents

1. Overview of an NoC
2. Message Transmission
3. Routing
4. Design of a Router
5. Non-Uniform Cache Architectures
6. Performance Aspects

Hierarchy of messages sent in a NoC



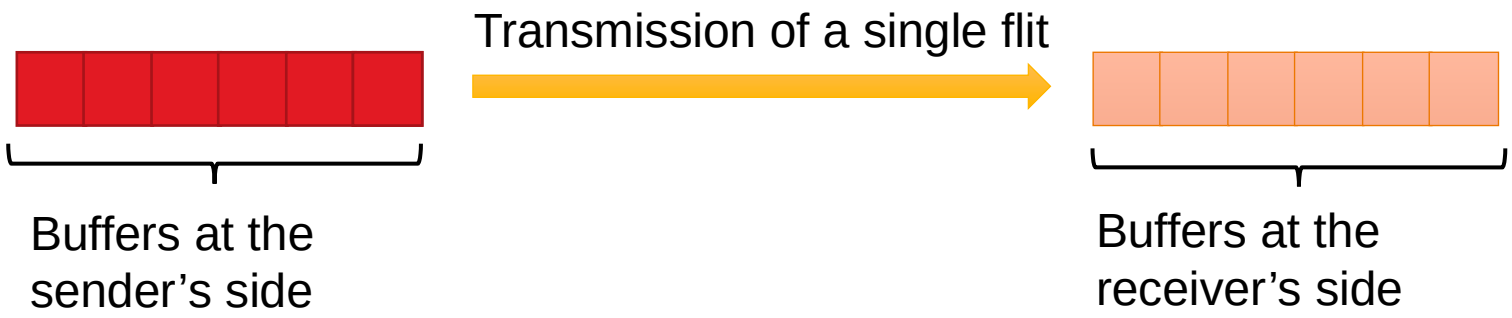
Description

- Message ␣ Set of bytes sent from a sending node to a receiving node
- Packet ␣ A message is broken into multiple packets.
  - All the bytes in a packet take the same route.
- Flit ␣ Flow control digit: size is 8 bytes or 16 bytes. An atomic unit of bytes. A router never breaks a flit.

Head flit	Body flit	Body flit	Tail flit

- Phit ␣ Physical digit. While transmitting a flit, it might take multiple cycles. If we can send 32 bits at a time, then transmitting a 128-bit flit will take 4 cycles. In each cycle we send a 32-bit phit.

Flow Control



- What do we do if we do not have enough buffer space at the receiver's side?
- We are not allowed to drop a flit.
- Have some information at the sender about the free buffer space available at the receiver.

Credit Based Flow Control

- The sender (A) maintains an estimate of the number of free buffers at the receiver (B).
- If the sender thinks that the receiver has enough free buffers, then only it sends a flit.

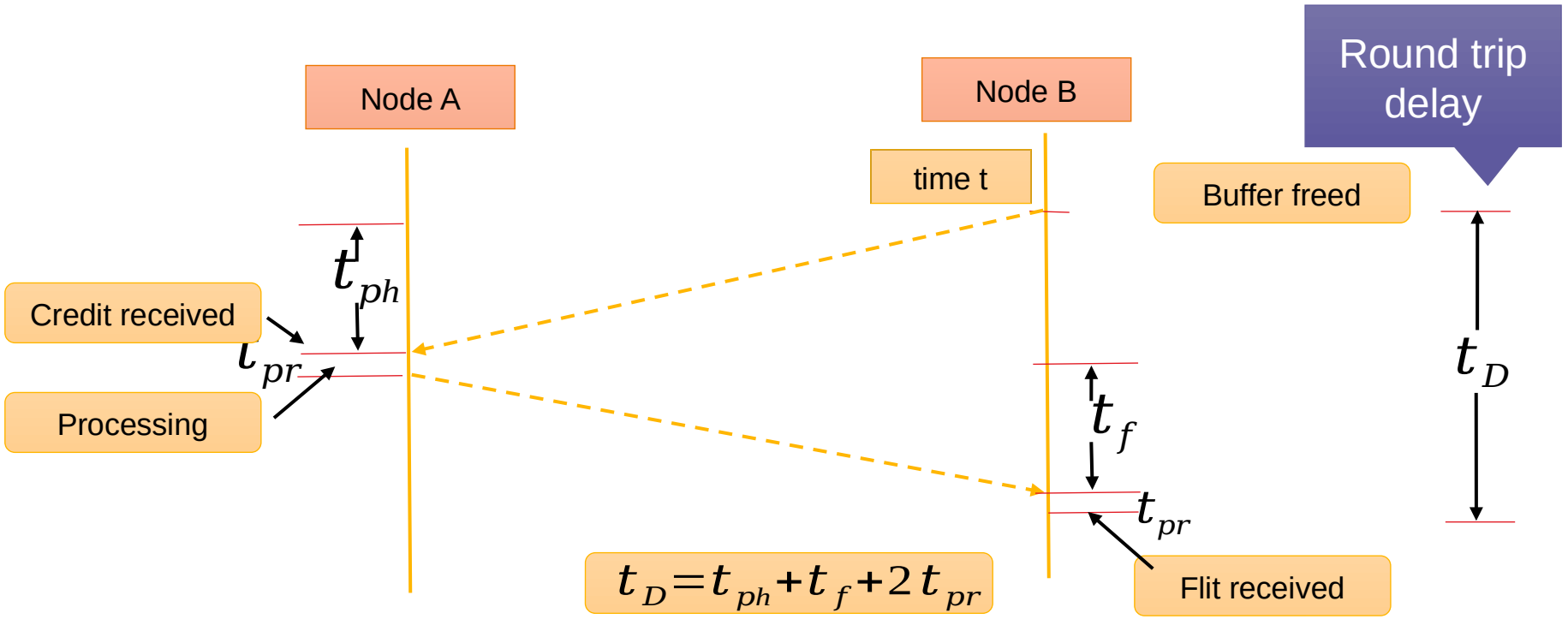
Term	Meaning
	Time it takes a single phit across the A ▭ B link
	Time it takes to send the flit from A ▭ B
	Time it takes to process a message.
	Total credit round trip delay

Assumptions

The routers are clock synchronized.

- Time is measured in the units of “router cycles”. All delays are an integral number of router cycles.
- We first receive a message, then process it, and then add it to a buffer.
- All the status messages (credits, on, off, etc.) are 1 phit each

Process of Sending Credits



Properties of Credit Based Flow Control

- From a buffer getting freed to receiving the next flit (because of the credit that was sent), it takes units of time.
- We can receive a maximum of flits in this time period.
- If the size of the buffer at *B* is at least () flits, then *A* will not stall (assuming *B* can send credit messages as soon as possible)

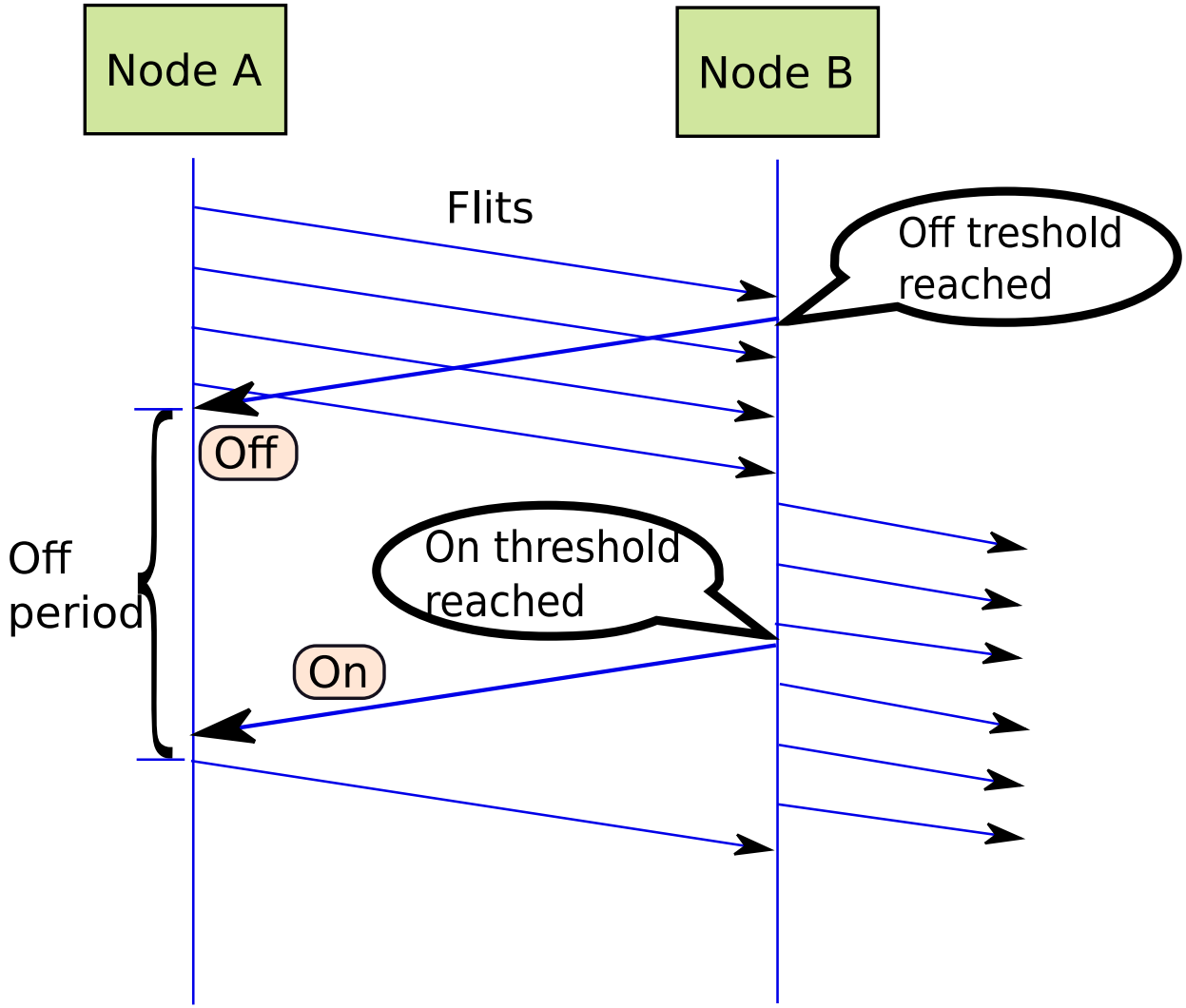
Disadvantage

- We need to send a credit message, whenever a buffer becomes free.
- Not power efficient

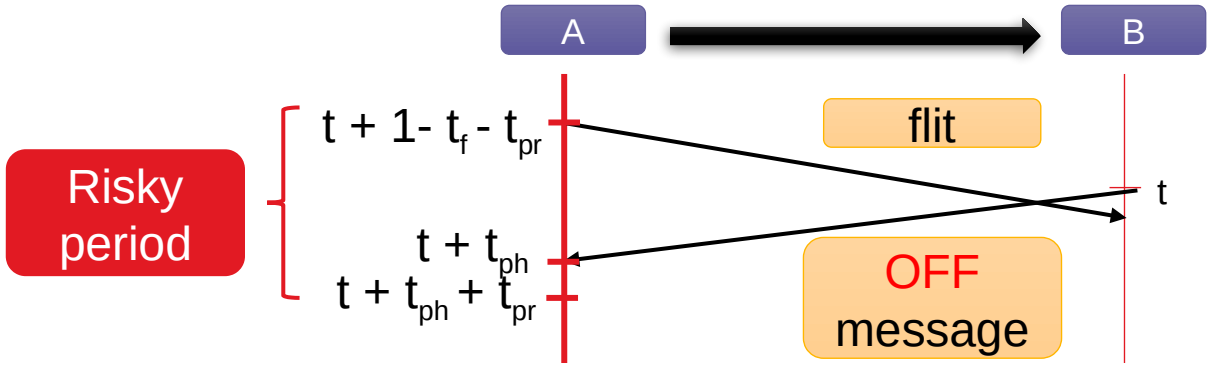
Improvement: On-Off Flow Control

- **Channel**: A ⊔ B
- Improvement over normal credit-based flow control
- Off-threshold (N_{OFF})
- If we have N_{OFF} free buffers in B, then it immediately sends a message to A to stop sending
- On-threshold (N_{ON})
  - If the number of free buffers in B is equal to N_{ON} , then it sends a message to A to start sending

Graphical Description



Analysis: Off message



How long is the risky period (during which packets are sent to *B*), after *B* has sent the OFF message?

$$(t + t_{ph} + t_{pr}) - (t + 1 - t_f - t_{pr}) + 1 = t_{ph} + t_f + 2t_{pr} = t_D$$

After sending the ***OFF*** message, B can receive t_D/t_f flits

→ It needs to have enough buffer space available

The equation for N_{OFF}

Because we cannot drop flits, we have:

$$N_{OFF} \geq \frac{t_D}{t_f}$$

Analysis: ON message

Necessary condition:

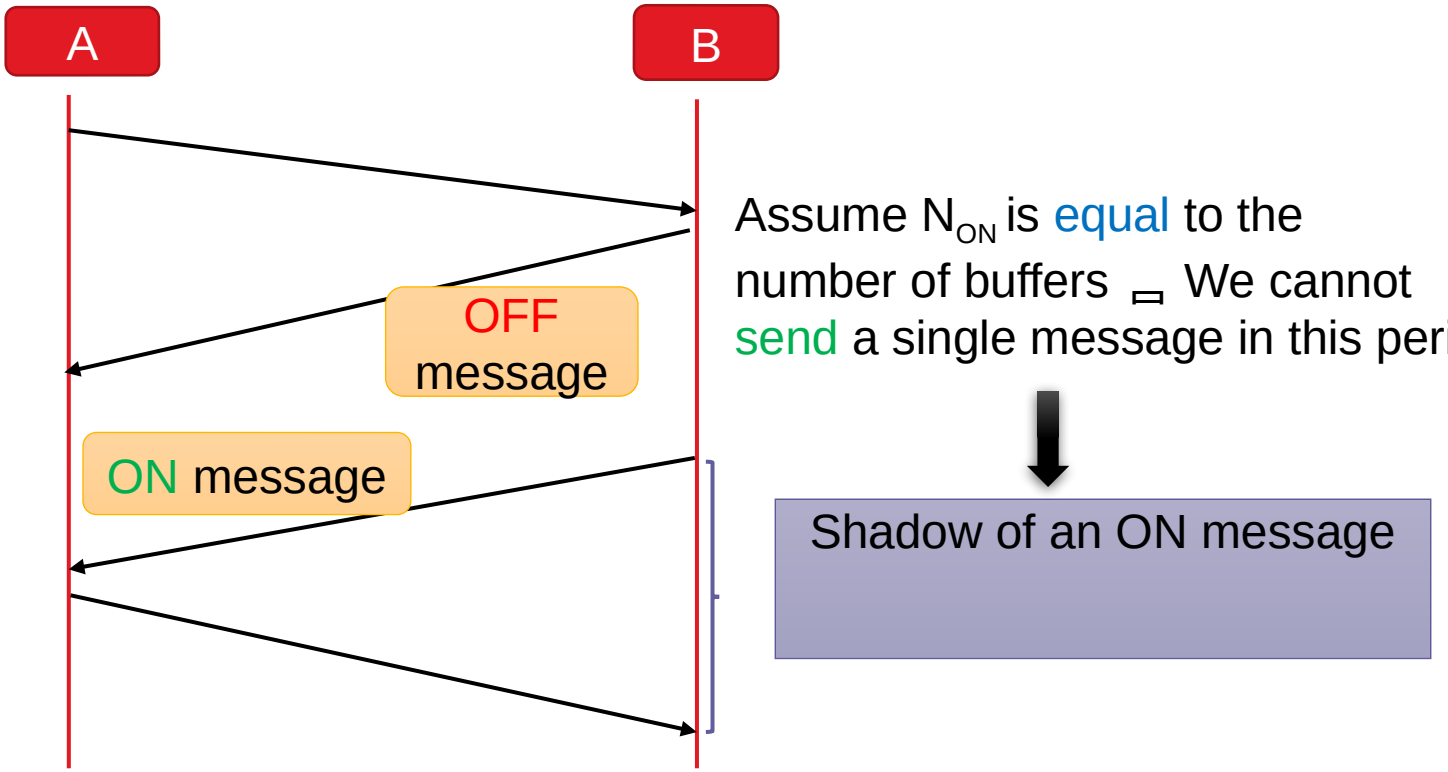
$$N_{ON} > N OFF$$

If $N_{ON} = N_{OFF} + 1$

- After we receive 1 message, we will have to send an OFF message
- Very inefficient

To avoid frequent ON and OFF messages, there should be a sizeable gap between the thresholds, such that we can transmit in one go without frequent stop-and-start messages.

Better estimate for N_{ON}



This means that *N* (number of buffers) should be far greater than N_{ON}

$N - N_{ON}$ is the number of buffers with messages. It should be more than a threshold.

Better estimate for N_{ON} - II

If *B* has *N* flit buffers and the following conditions hold:

We can keep transmitting flits in the shadow of an ON message. We will have enough flits.

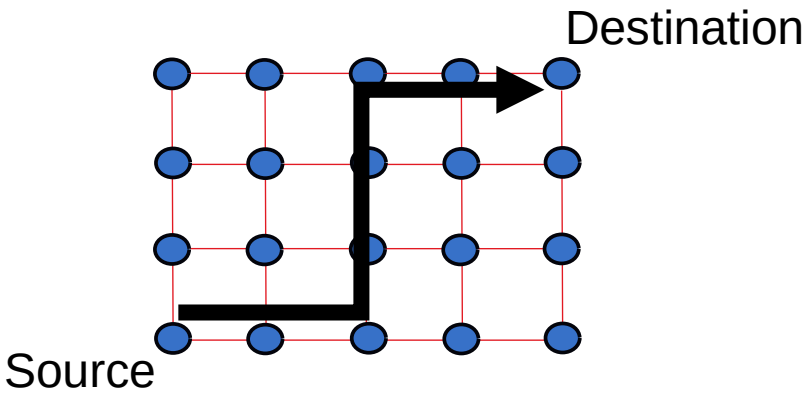
We thus have:

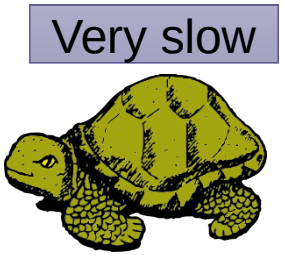
$$N \geq N_{ON} + \frac{t_D}{t_f} > N_{OFF} + \frac{t_D}{t_f} \geq \frac{t_D}{t_f} + \frac{t_D}{t_f} = 2\frac{t_D}{t_f}$$

$$N > 2\frac{t_D}{t_f}$$

Transmission of Messages on the NoC

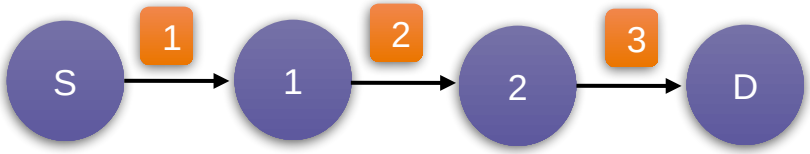
Circuit Switching



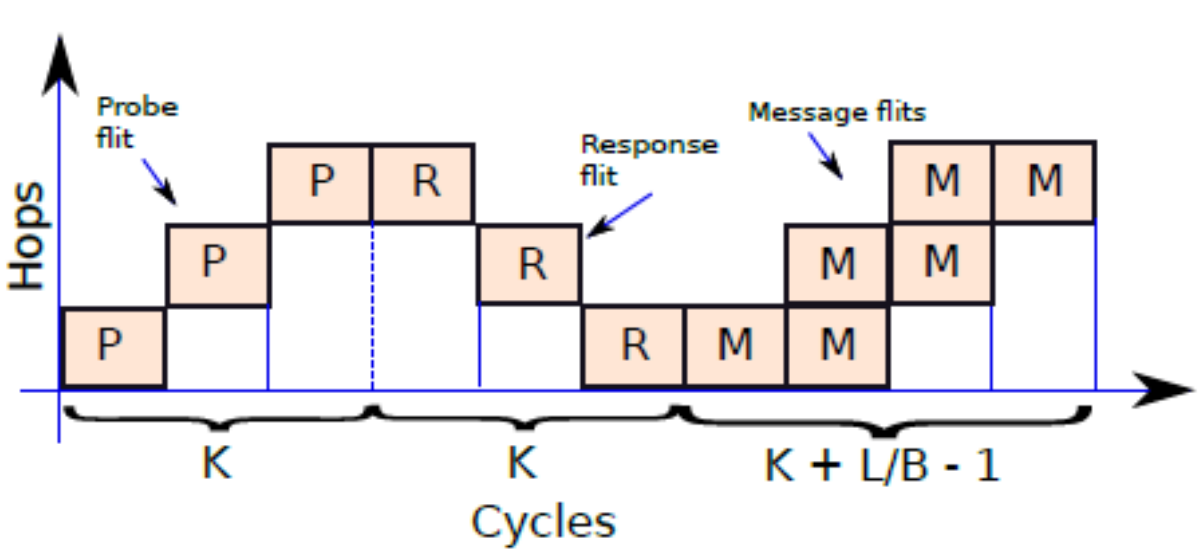


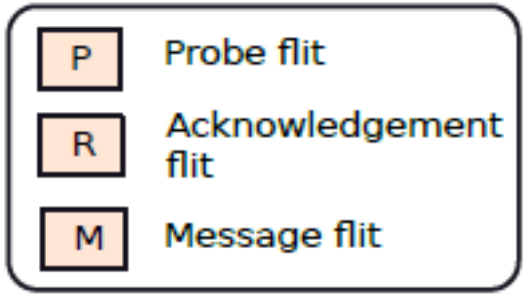
- Reserve buffer space in the routers along the way
- Once the entire path is reserved, then send a message
- After sending the message, clear the buffers

Space-time diagram



L = Length of the message in flits
K = Number of hops/ cycles to reach the destination (assumption: 1 cycle per hop)
B = channel bandwidth: flits per cycle





K = 3, L = 2, B = 1

Total time:
3K + L/B - 1

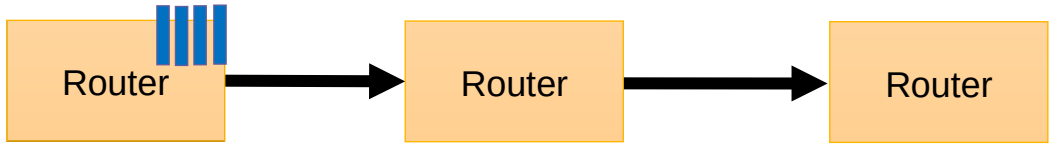
Advantages and Disadvantages

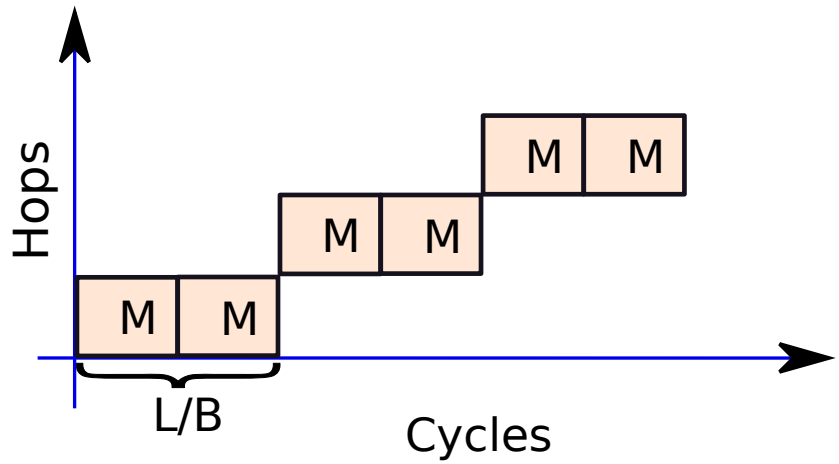
- Once a circuit is established, we can transfer as may bytes as we want.
- This is good for bulk transfers.

- Large overheads for a single transfer.
- Locks up resources for more time than the actual requirement.

Packet based Flow Control: Store and Forward (SAF)

Send a packet to a router (set of flits), store the entire packet, and then forward it to the next router.



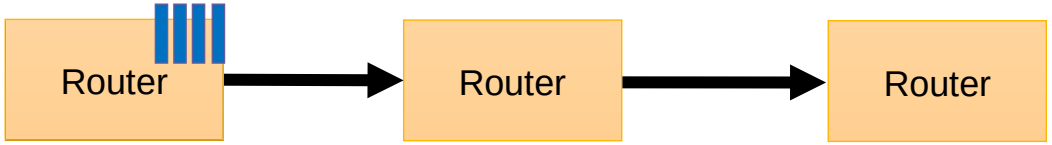


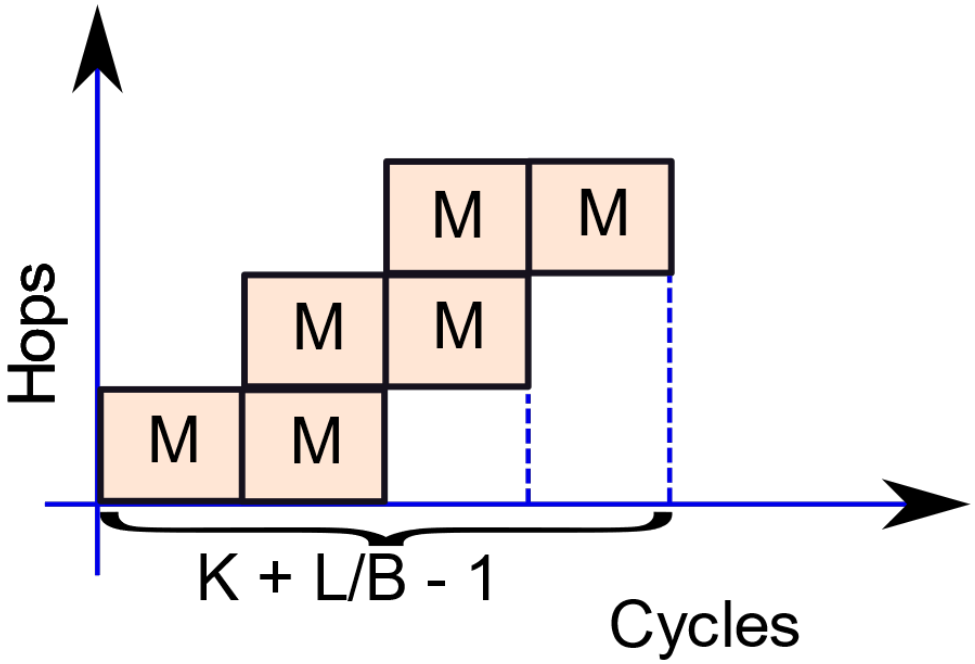
K * L / B cycles

SAF Protocol

- When the flits are being sent, we wait for the entire packet to be buffered.
- This is a waste of time. We can send flits to the next router on the way.
- **Analysis**: Total time required $\sqsupset$ K * L/B

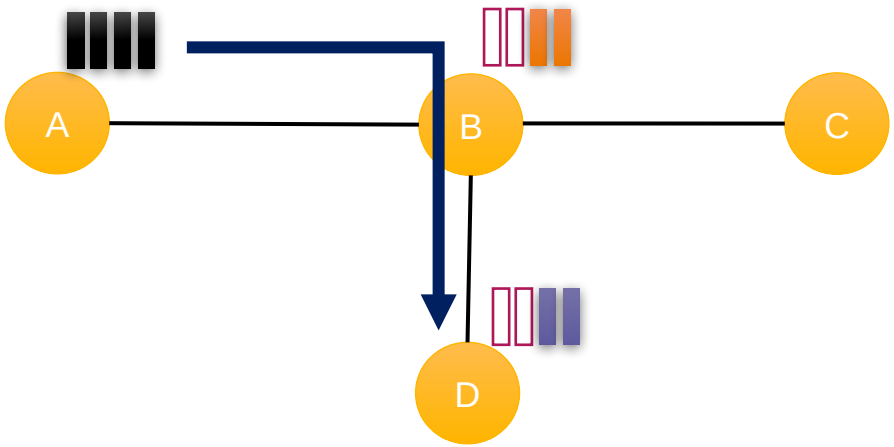
Virtual Cut-Through Routing (VCT)





Even though we send flits before the entire packet is buffered, we need to ensure that there is space for the entire packet.

Problems in the VCT Scheme



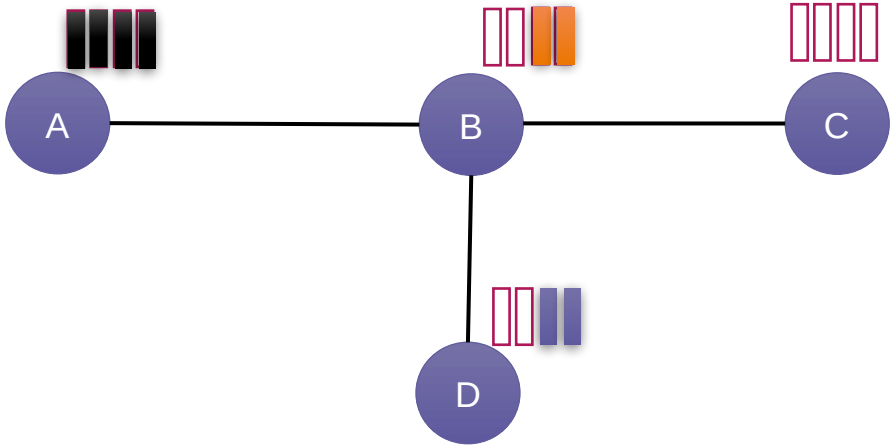
- Flit from packet 1 destined to C
- Flit from packet 2 destined to D
- Flit from packet 3 destined to a node beyond D

We cannot send any flit from packet 1 till the router at B is totally empty because we need to buffer the entire packet.

Comparison of Circuit Switching with Packet-level Switching

Scheme	Time in cycles
Circuit Switching	3K + L/B – 1
Store and Forward (SAF)	K * L/B
Virtual Cut Through (VCT)	K + L/B - 1

Solution: Buffer at the Flit Level

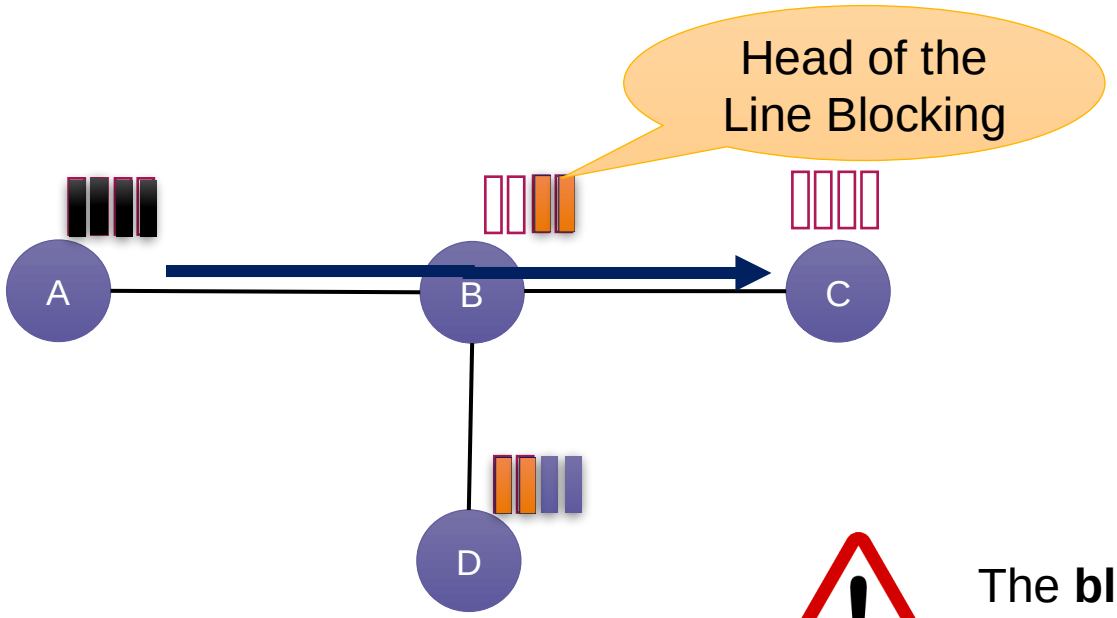


Wormhole Flow Control

- We do not have to wait for space to be created to buffer the entire packet.
- The flow control is at the level of flits.

- Does not solve all our problems.

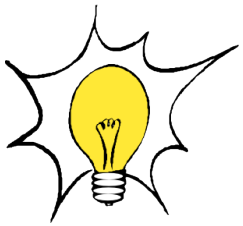
Problem with Wormhole Routing



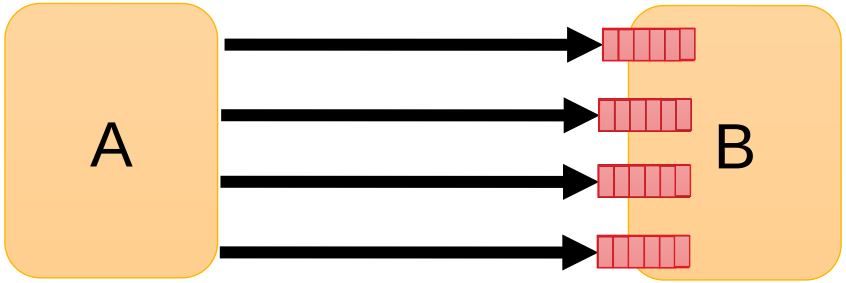
Flit from packet 1
Route: A ␣ B ␣ C

The **black** flits can go to C, however, the **orange** flits are blocking them.

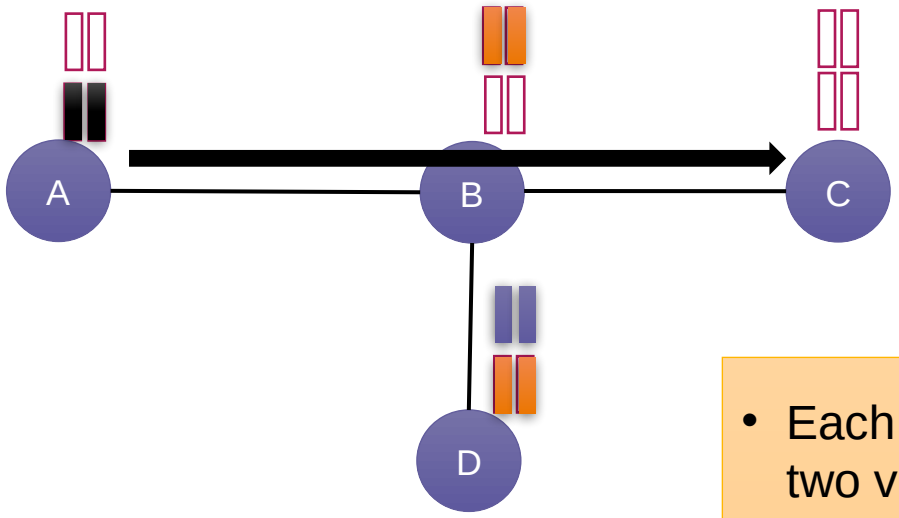
Solution: Virtual Channels (VCs)



Multiplex a single physical channel among different virtual channels.
Have a separate buffer queue for each virtual channel.



Virtual Channels: Two per physical channel



- Each physical channel split into two virtual channels.
- No HoL blocking.

Virtual Channels: Pros and Cons

- There is no head of the line (HoL) blocking.
- Increases throughput significantly.
- We can multiplex a single physical channel among different virtual channels.

- Increases the complexity of the router.
- We now need to assign the virtual channels in a router (additional work).

Contents

1. Overview of an NoC
2. Message Transmission
3. Routing
4. Design of a Router
5. Non-Uniform Cache Architectures
6. Performance Aspects

Aim of Routing

Send a message from a sender to a receiver

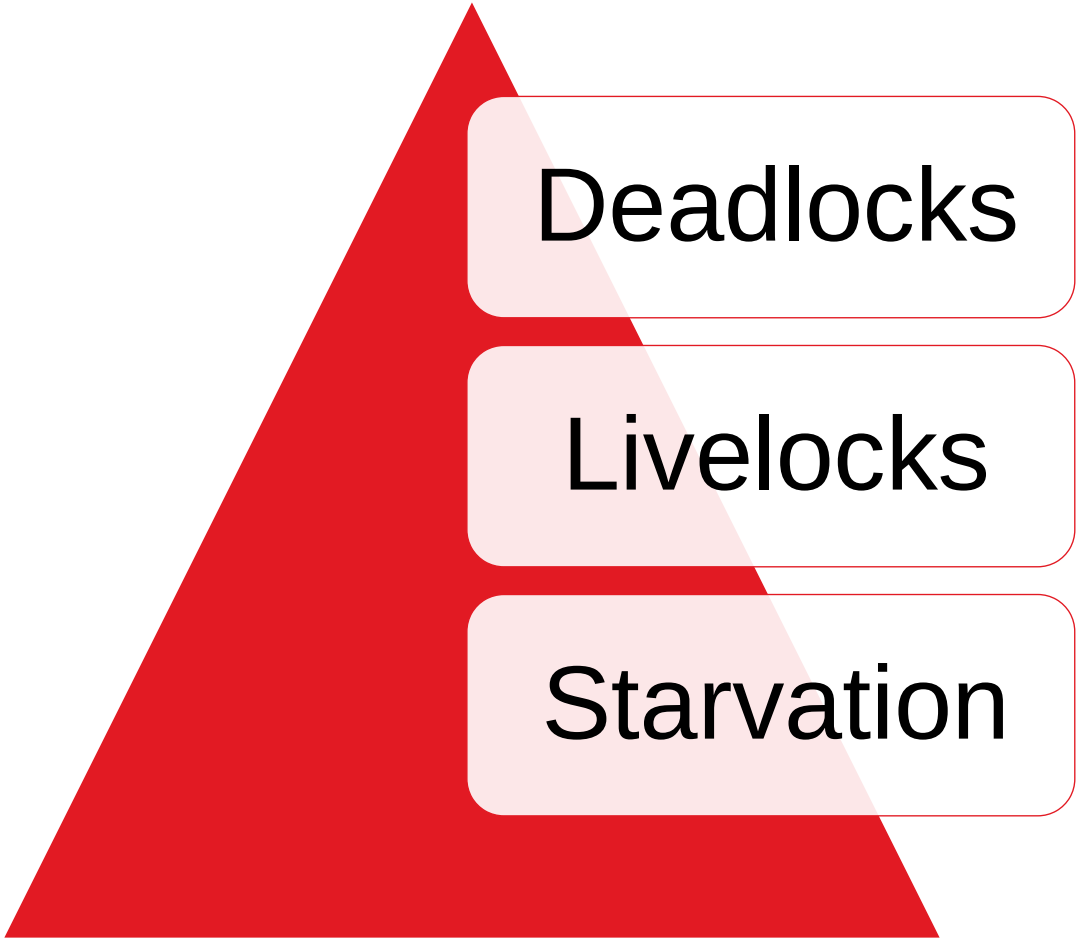
- Via a path of intermediate routers
- Good to have a choice of multiple routes ⎵ **path diversity**

What is a good route?

- Shortest path?
- Shortest time?

Good routes typically minimize the latency, and are also fair ⎵ do not unnecessarily penalize other packets

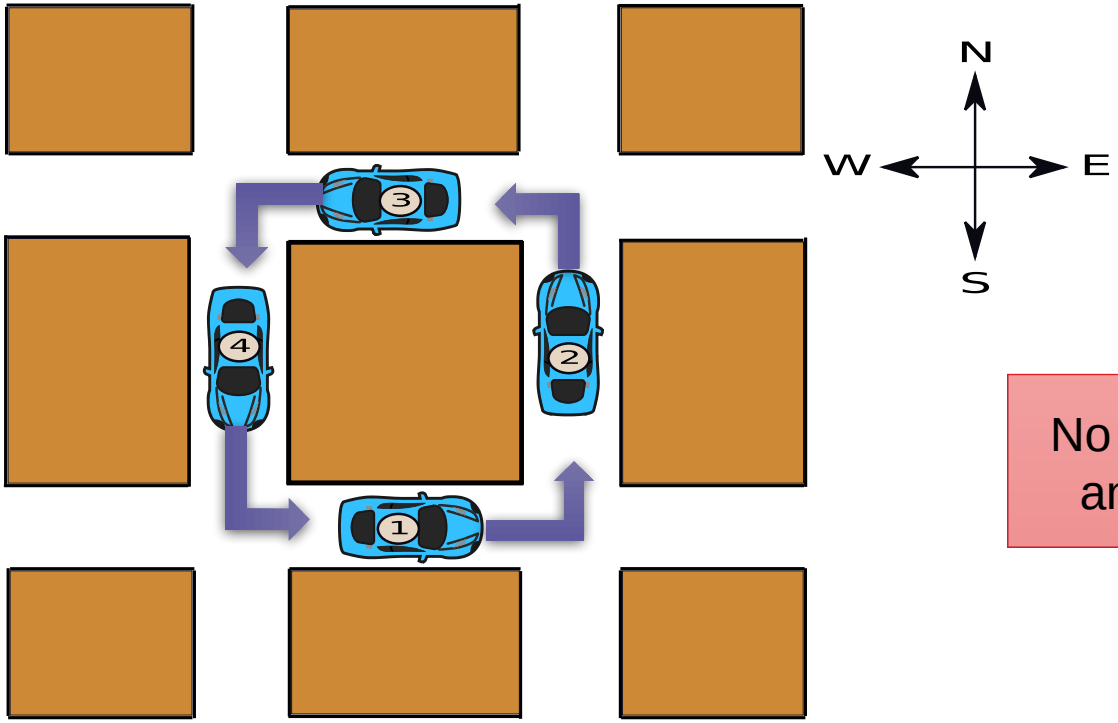
Problems to avoid …



Avoid Deadlocks

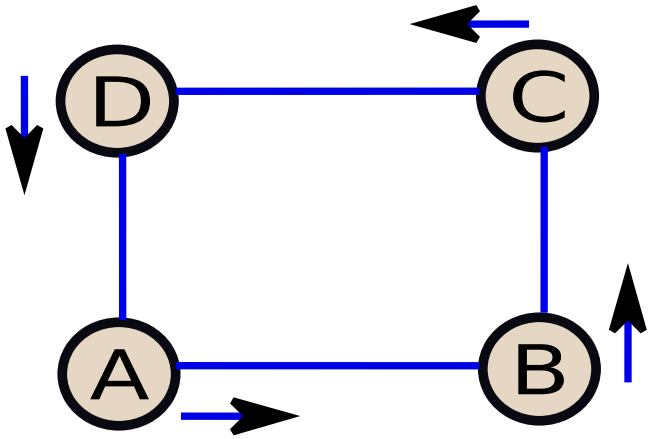
Photo by Jens Herrndorff on Unsplash

Meaning of a Deadlock

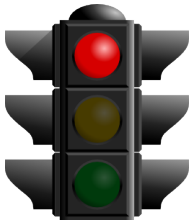


No car can make any progress.

Deadlock in an NoC



Assume a buffer size of 1 flit



No progress …

Livelock

- Flits continuously get transmitted, but they never reach their destination
- They keep moving in infinite loops
- These routing algorithms never terminate.

Starvation

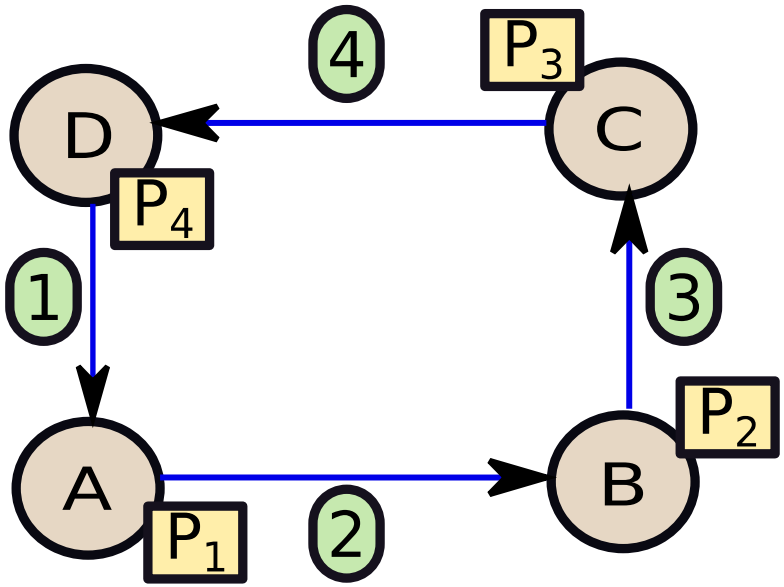
- A certain flit never makes progress (never moves towards its destination). It has the lowest priority, its router always transmits other flits.

Solution: Ageing

- Both the problems can be solved by implementing ageing. The longer a flit stays in a router, the higher is its priority.

Dealing with Deadlocks

- Deadlock in a circuit-switched system.

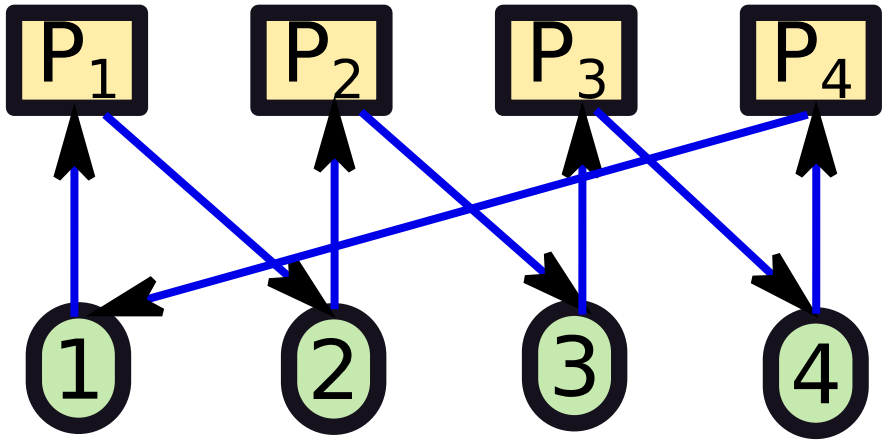


Packet	Source	Destination	Holds channel	Wants channel
P_1	D	B	1	2
P_2	A	C	2	3
P_3	B	D	3	4
P_4	C	A	4	1

Deadlock

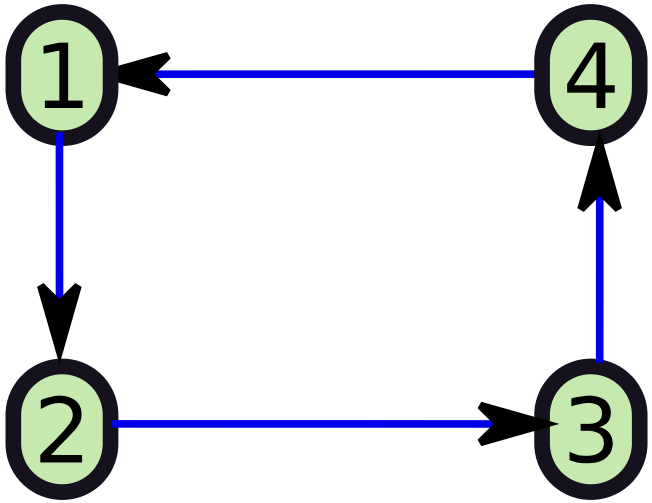
Resource Dependence Graph (RDG)

- There is an arrow from channel *j* to packet P_i, if the packet P_i holds the right to transmit on channel *j.*
- There is an arrow from P_i to channel *j*, if the packet P_i is waiting for channel *j* to become free.



A cycle in the RDG implies a deadlock.

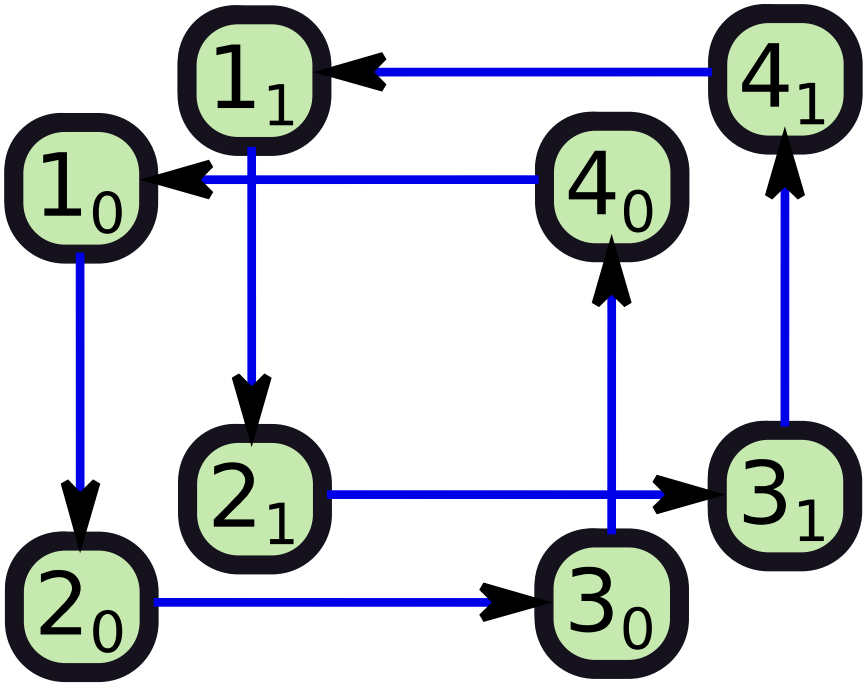
Channel Dependence Graph (simplified version of the RDG)



There is an arrow from Channel A to Channel B if the packet (message) that owns Channel A is waiting to get access to Channel B.

A cycle in the CDG implies a deadlock.

We can even have deadlocks with virtual channels ☹



1_0 ▭ 0^{th} VC of physical channel 1
1_1 ▭ 1^{st} VC of physical channel 1

Turn Graphs

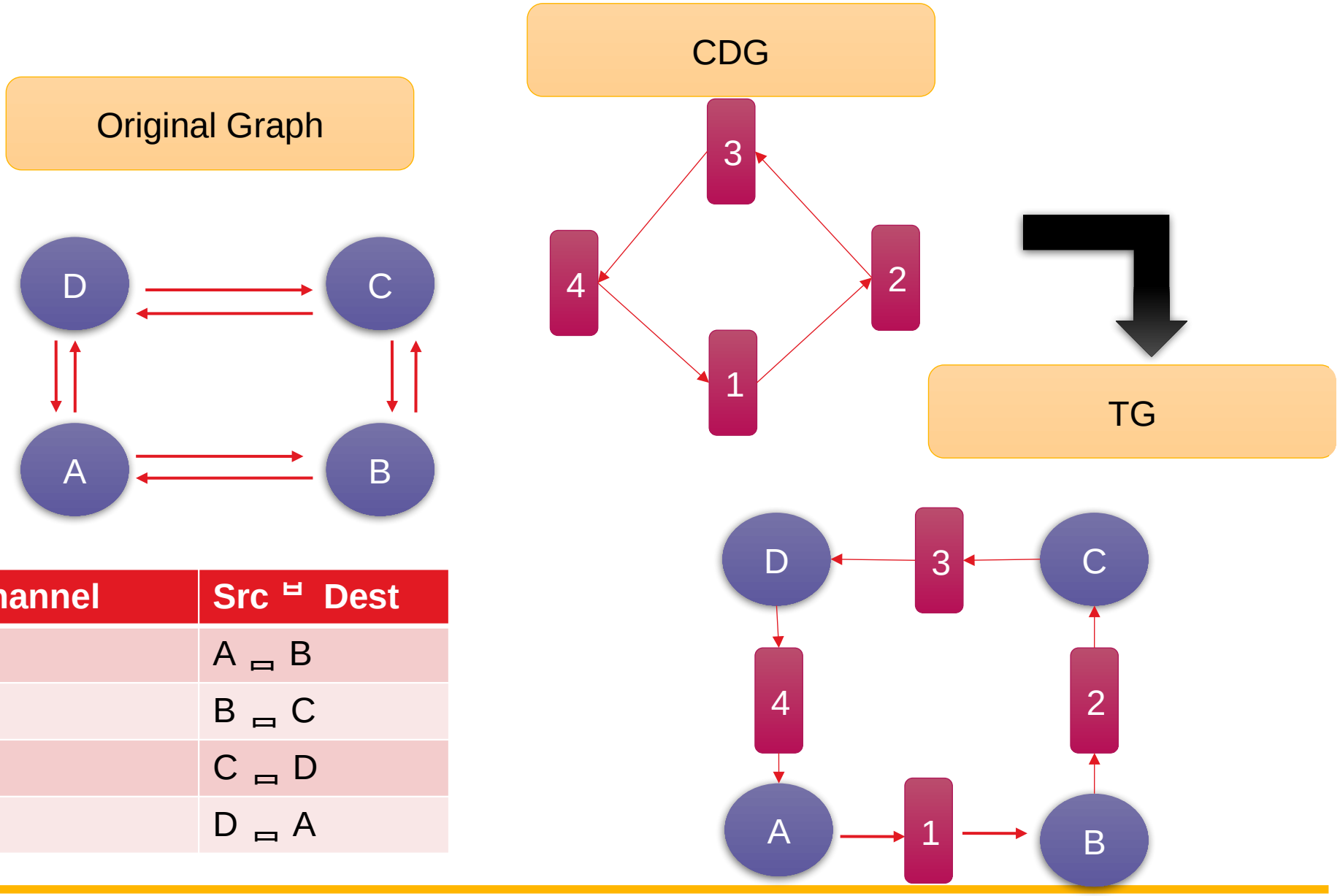
Disadvantage of CDGs and RDGs

- They lose the orientation information of the channels.

Basic Steps

- Consider a path *C* in the channel graph.
- The TG contains all the nodes and channels belonging to *C.* It preserves the orientations of the channels. It does not contain any other channel.
- We insert a new node called the *channel node* in the middle of each edge/channel.
- There is a path *C* in the CDG ⎵ There is a path having the same channel nodes in the TG (and vice versa)
- A cycle in the CDG ⎵ A cycle in the TG

Illustration of a TG (Turn Graph)



Channel	Src ⊔ Dest
1	A ⊔ B
2	B ⊔ C
3	C ⊔ D
4	D ⊔ A

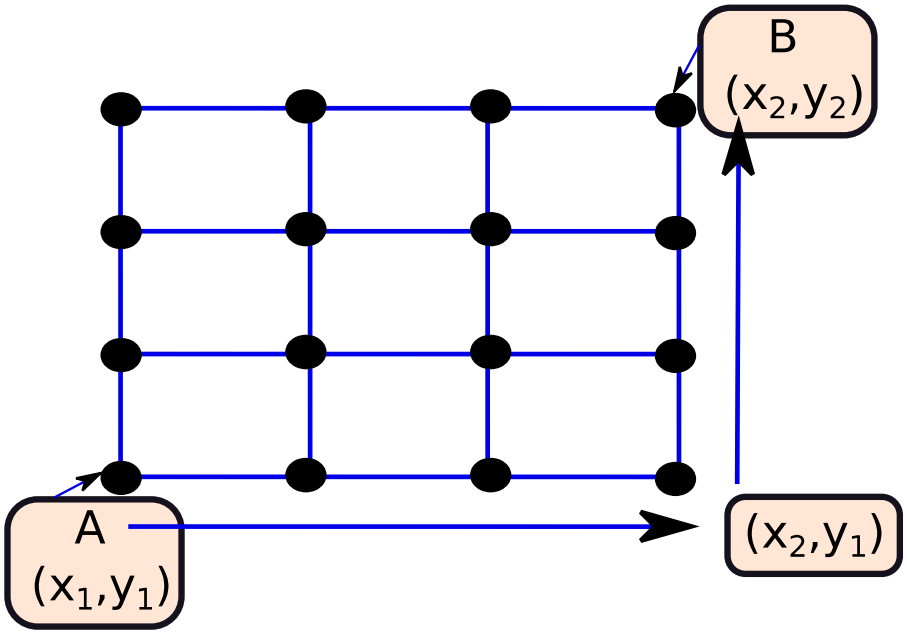
Properties of the TG

- Every edge in the CDG translates to either a set of collinear nodes in the TG, or a turn in the TG.
- Every cycle in the CDG is also a cycle in the TG.
- Every cycle in the CDG can be translated to a sequence of straight paths and turns in the corresponding TG

Aim:

Ensure that there are no cycles in the turn graph (TG).
Try to introduce routing protocols that are provably deadlock free (their TGs will never have cycles).

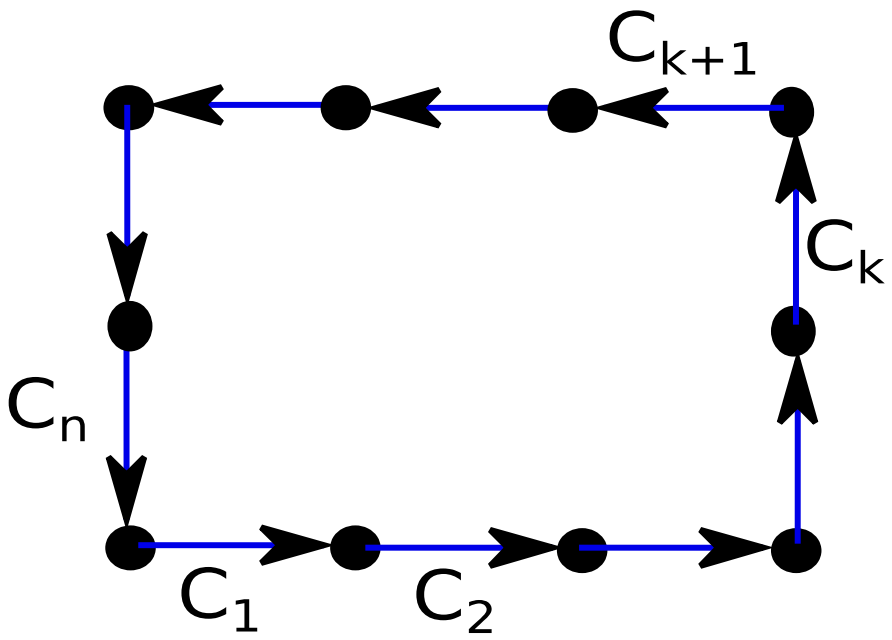
Dimension-Ordered Routing (X-Y Routing)



1. Route along the x-axis first
2. Then send the message along the y-axis

If we have *n* dimensions, we can order them, and route the message first along dimension 1, then dimension 2, and so on.

Assume that the TG has a cycle



Example: anti-clockwise cycle

1. We either have a clockwise cycle.
2. Or an anti-clockwise cycle.

The C_k C_{k+1} turn is not allowed. A packet is traversing in the y direction first, and then in the x direction.

Can be proven for all the other cases

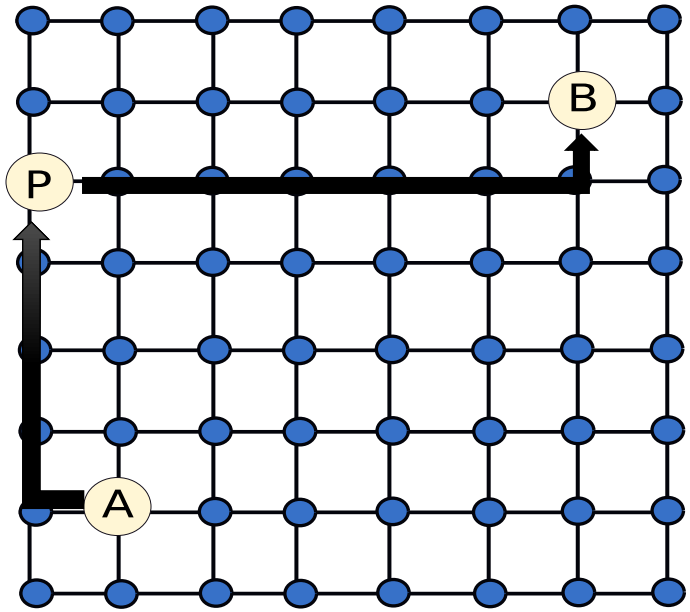
Hence, the TG does not have any cycles No deadlocks

Problems with X-Y routing

- It is simple.

- X-Y routing has no path diversity. The route is fixed.
- The algorithm cannot adapt to congestion.

Oblivious Routing (Valiant's Algorithm)

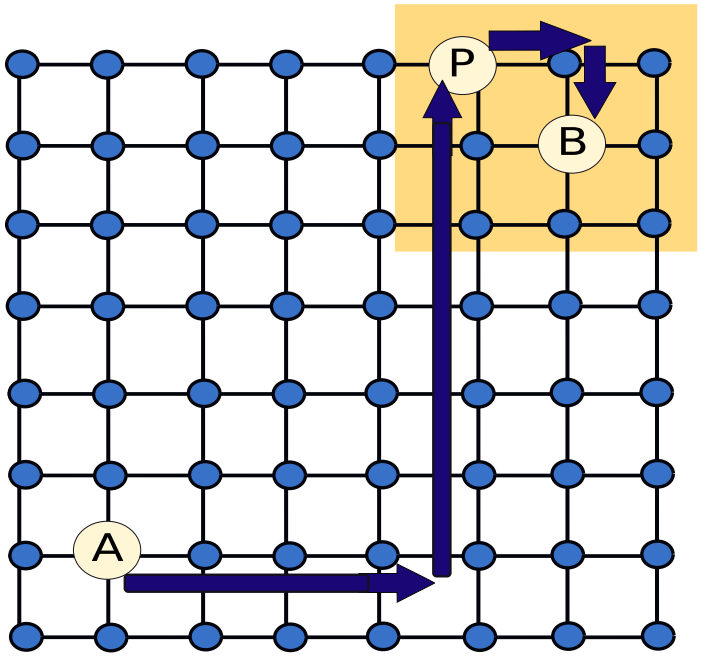


1. Choose a point *P* at random
2. Send the message from the source to *P*, and then from *P* to the destination.

\+ Deals with congestion much better by providing path diversity.

\- The routes can be very long and sub-optimal.

Minimally Oblivious Routing

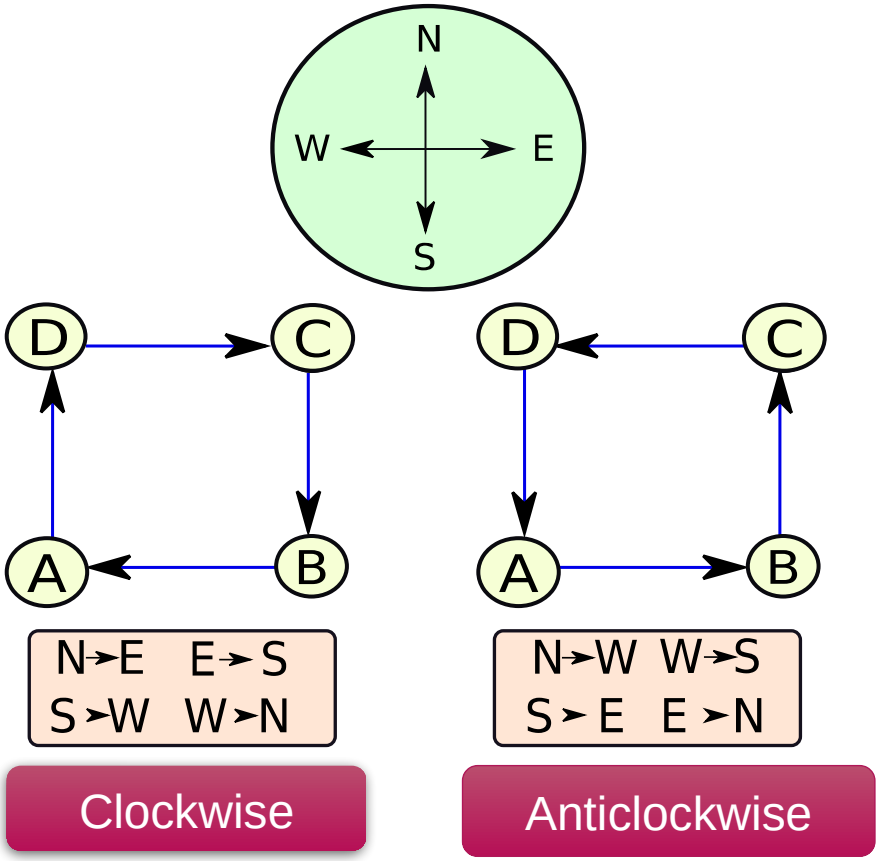


1. Create a window around B.
2. Send the message from the source to *P* (any node within the window), and then from *P* to the destination.

\+ The computed routes are not very long.

− Reduced path diversity.

Adaptive Routing: Theory of Turns

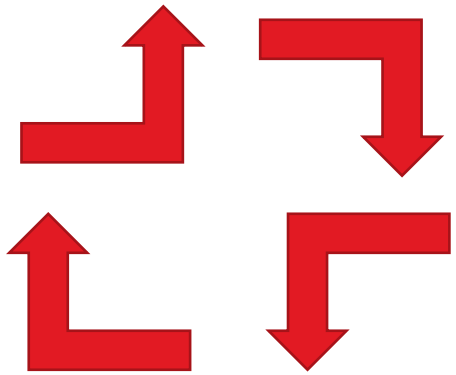


1. We can have two kinds of cycles: clockwise and anti-clockwise.
2. There are 8 possible turns.

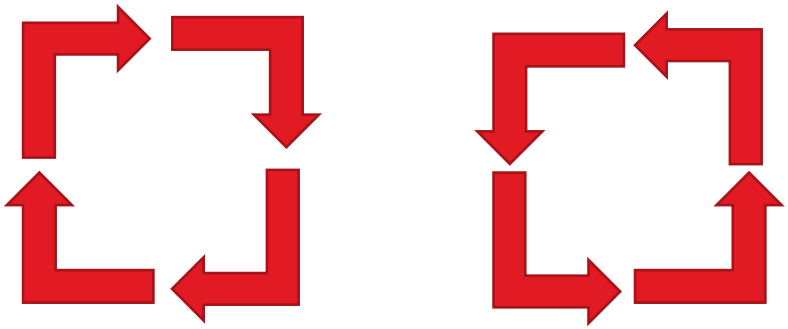
Out of these we can allow only 6 turns (at the most). We have to remove one turn from each cycle to prevent a cycle from forming in the **turn graph**.

Types of Routing Algorithms based on Turns

X-Y Routing



Only 4 allowed turns

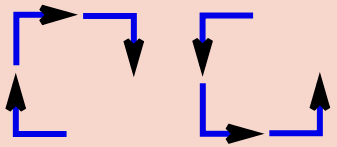
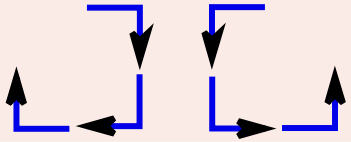
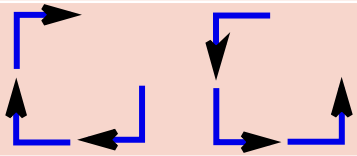


Disallow one of the turns in each cycle.

We have 4 choices per cycle:
(clockwise or anti-clockwise).
We thus have a total choice of 16 combinations.

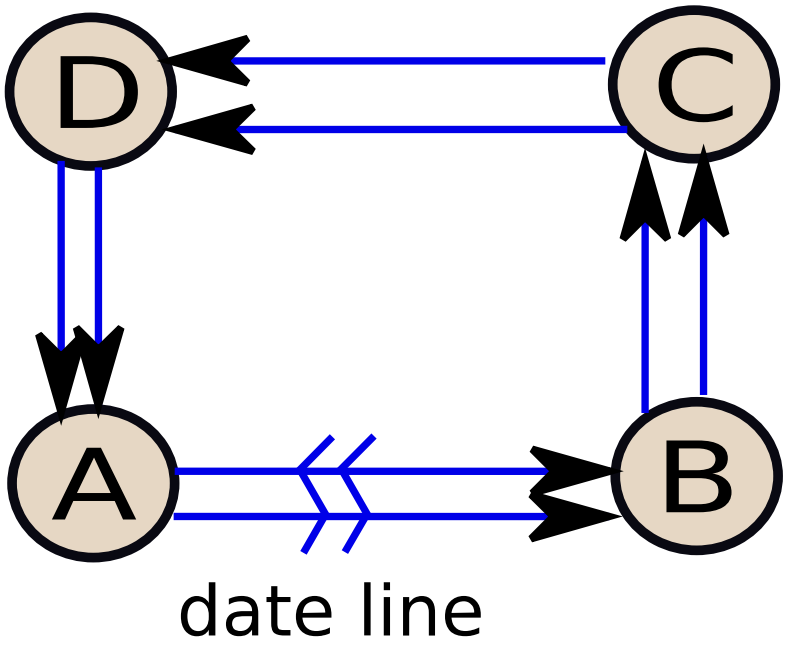
There are 16 such routing algorithms that eliminate one turn from each cycle

Examples of Popular Routing Algorithms

Name	Turns Disallowed	Allowed Turns
West-first	N ▭ W, S ▭ W	
North-last	N ▭ W, N ▭ E	
Negative-first	N ▭ W, E ▭ S	

High path diversity, and congestion avoidance capability

Date line based method where we have 2 VCs per physical channel



1. Inject the packet into VC 0
2. It keeps travelling in VCs numbered 0 till it crosses the date line.
3. Then it shifts to VCs labelled 1. It continues on these VCs till it reaches the destination.

There are no deadlocks.

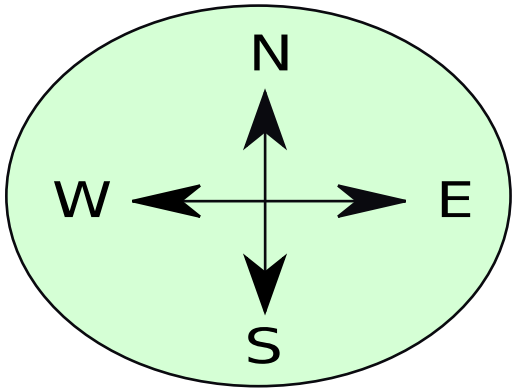
?

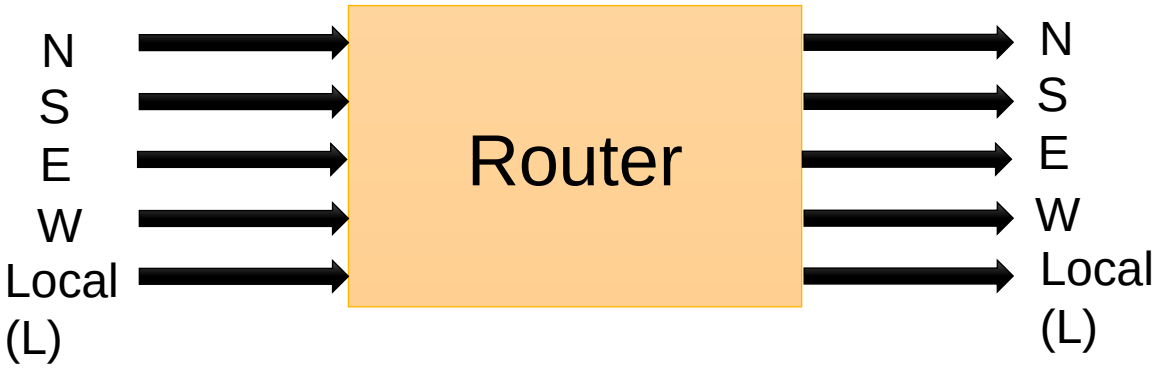
A flit on a VC numbered 1 will never wait for a VC numbered 0. A cyclic wait is not possible.

Contents

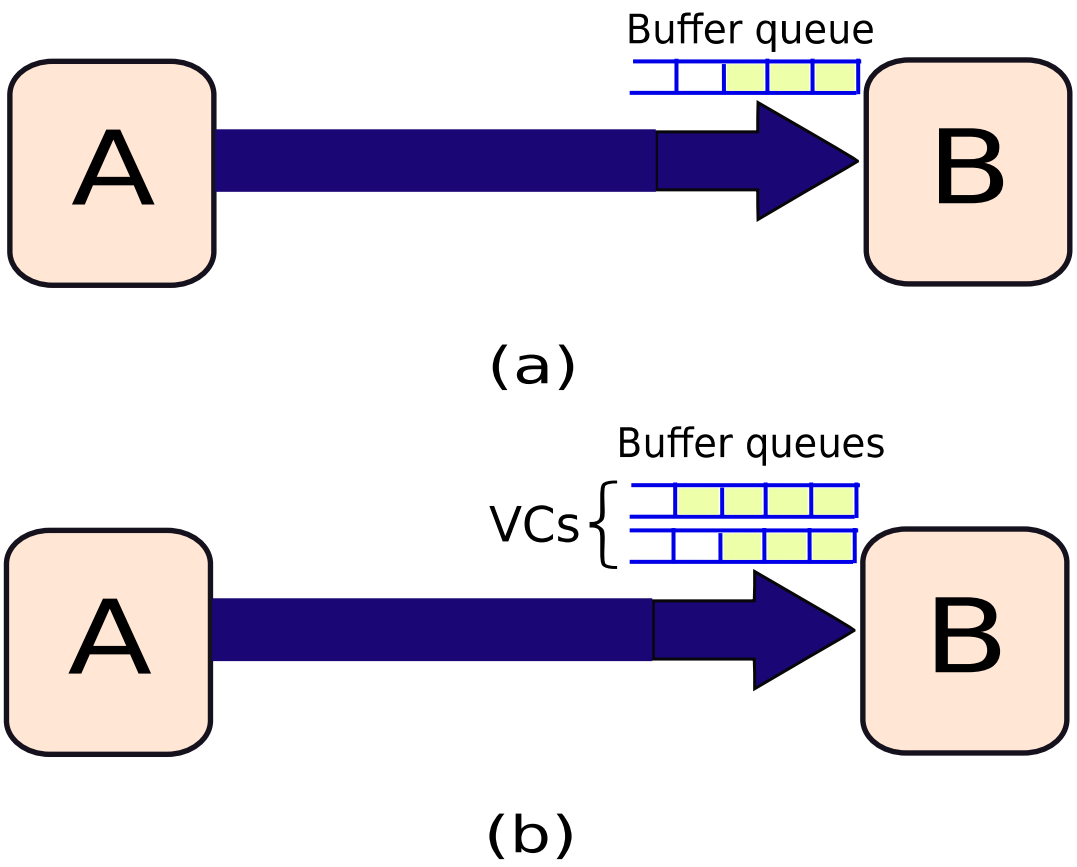
1. Overview of an NoC
2. Message Transmission
3. Routing
4. Design of a Router
5. Non-Uniform Cache Architectures
6. Performance Aspects

Interface



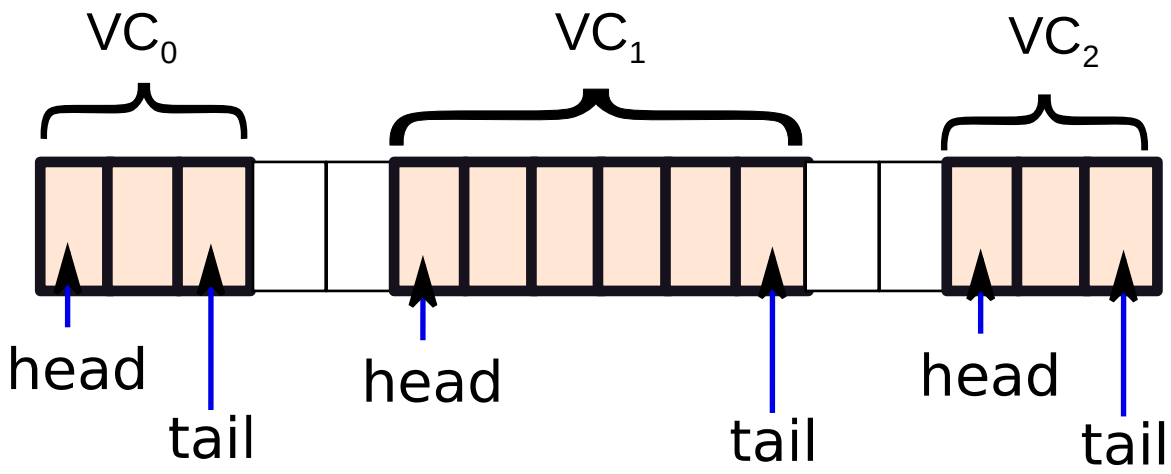


Input Buffers

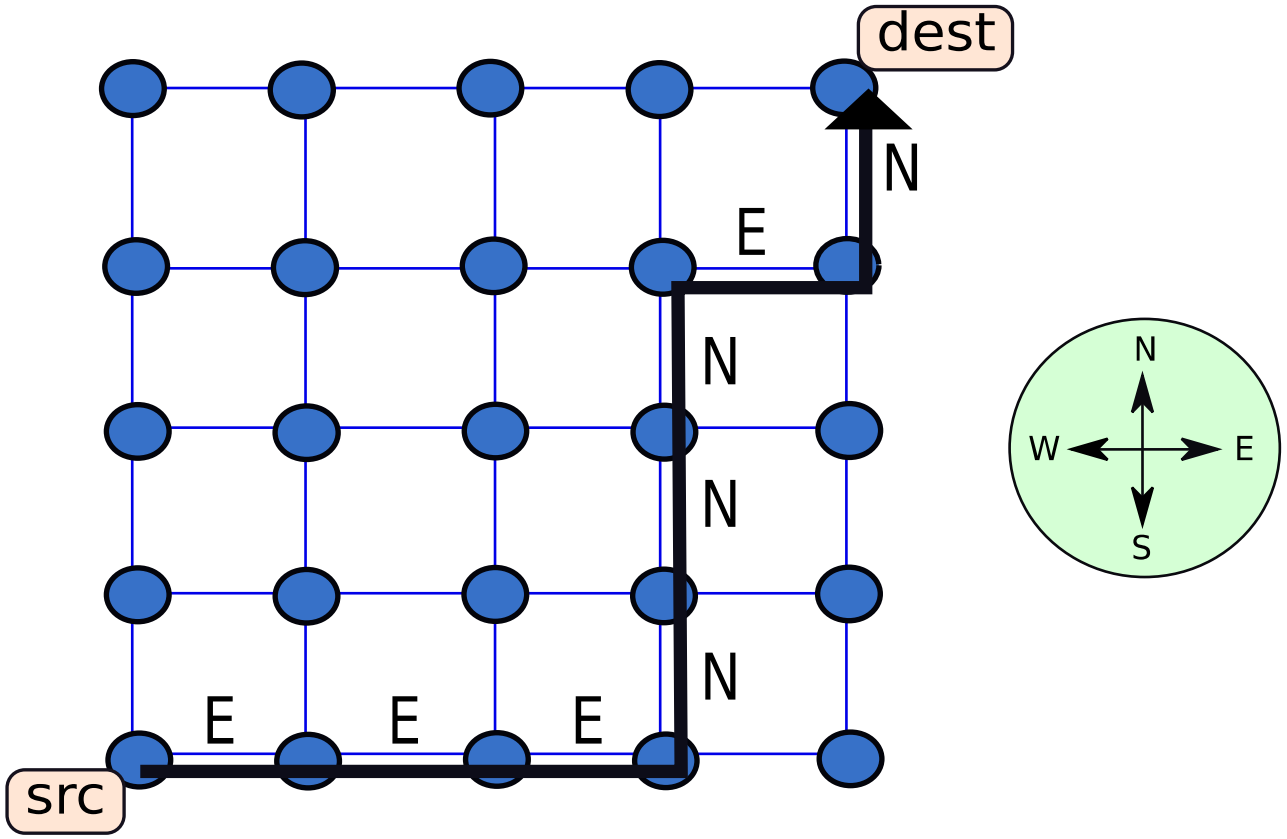


1. All the flits are buffered first.

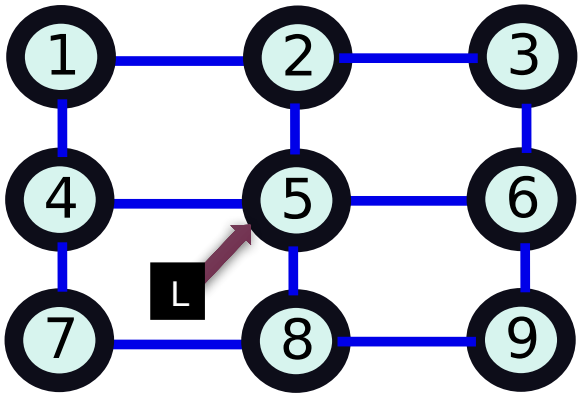
For multiple VCs (per physical channel) we can have a single, large buffer.



Route Computation



Example of a Routing Table



1. Each node maintains a routing table (node table). Specifies the next hop for the final destination.

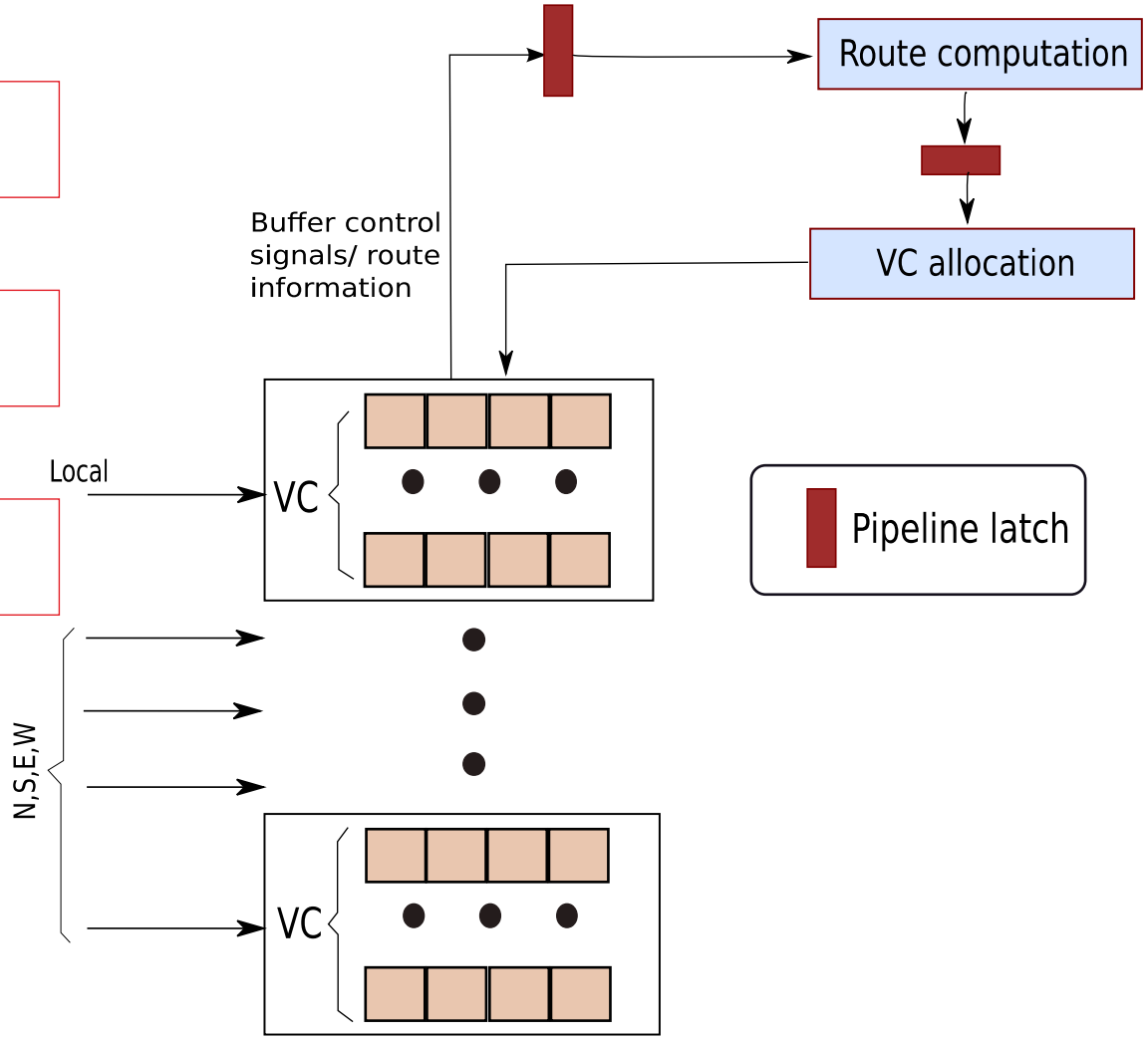
2. This table shows the routing decisions for the **North-last** algorithm at Node 5. Note that in some cases we have a choice of two routes.

Destination	Direction
1	W
2	N
3	E
4	W
6	E
7	S | W
8	S
9	S | E

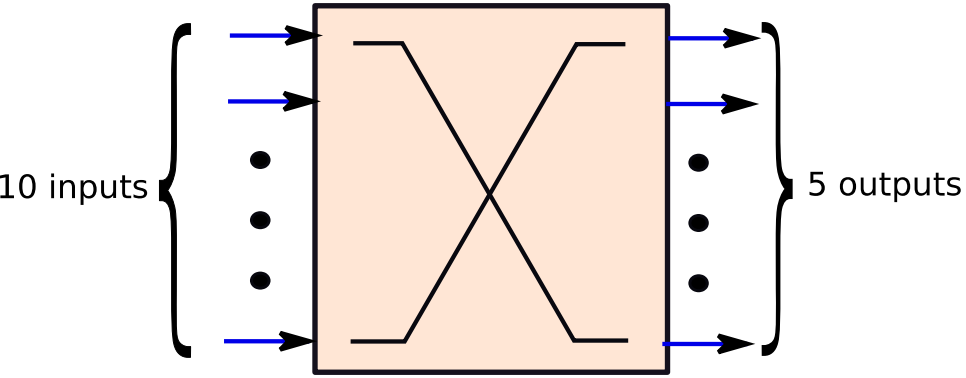
3. Make a decision based on congestion information. Take into account the delay incurred in sending flits the last time the channel was used.

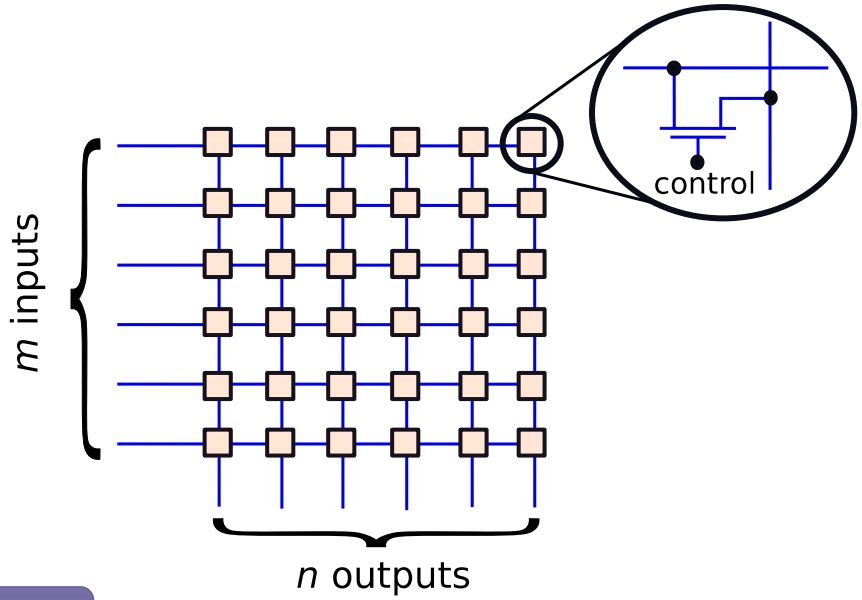
Router Pipeline till Now

- Buffer Write
- Route Computation
- VC Allocation in the physical channel



Next Step: Allocate switch ports





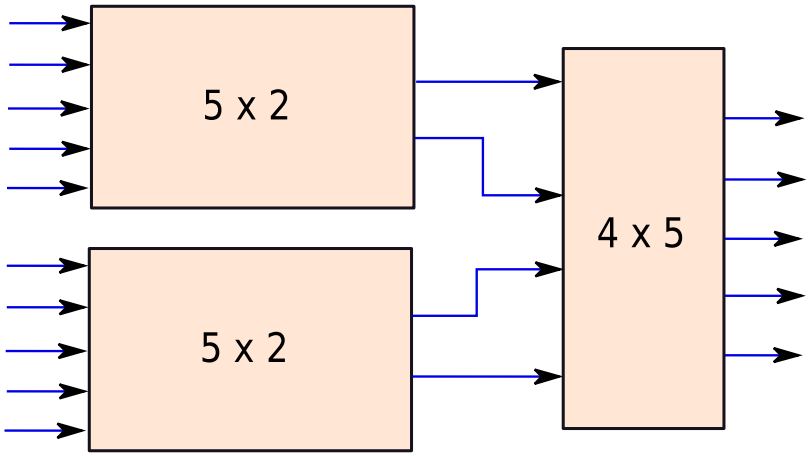
Design of an m x n switch

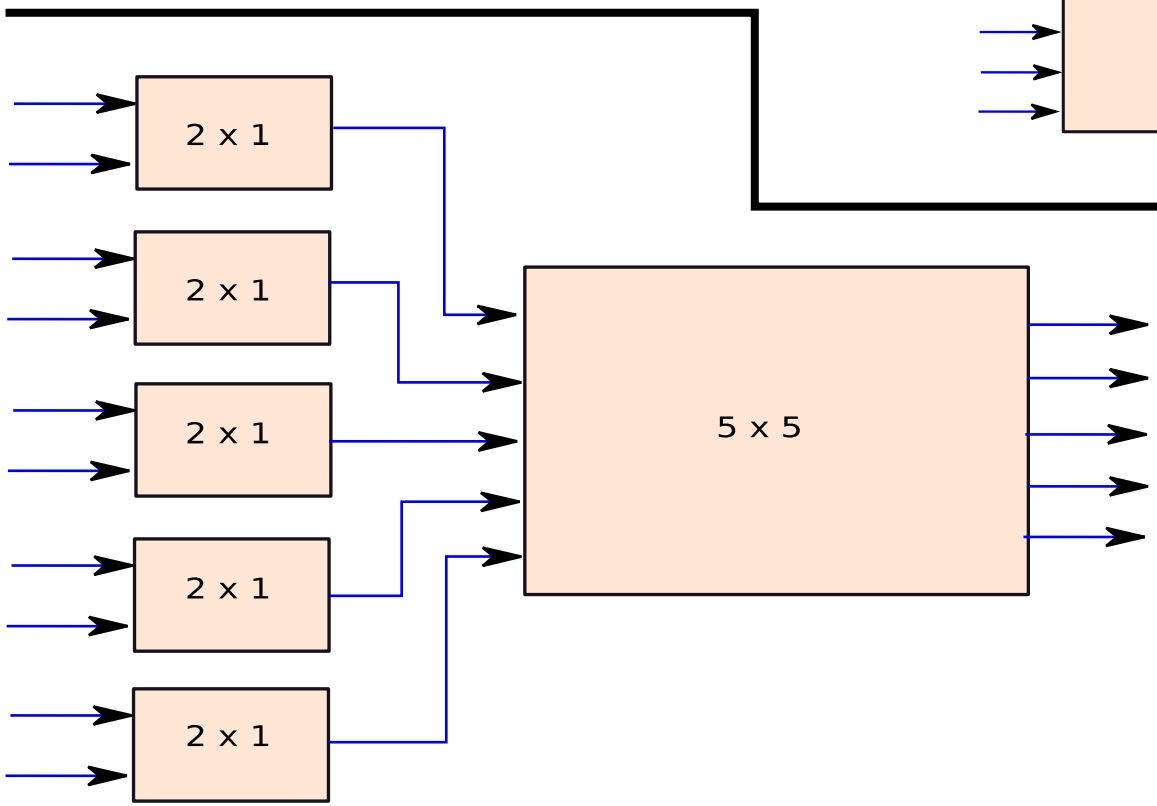
This part of the router's pipeline

Allocate Switch Ports	• Reserve a path to the output port • Fairness is important.
Switch Traversal	• Latency is critical.
Release Resources	• Once the tail flit leaves the router. Reclaim the VC.

Hierarchical Designs for Switches

Bigger switches





Smaller switches

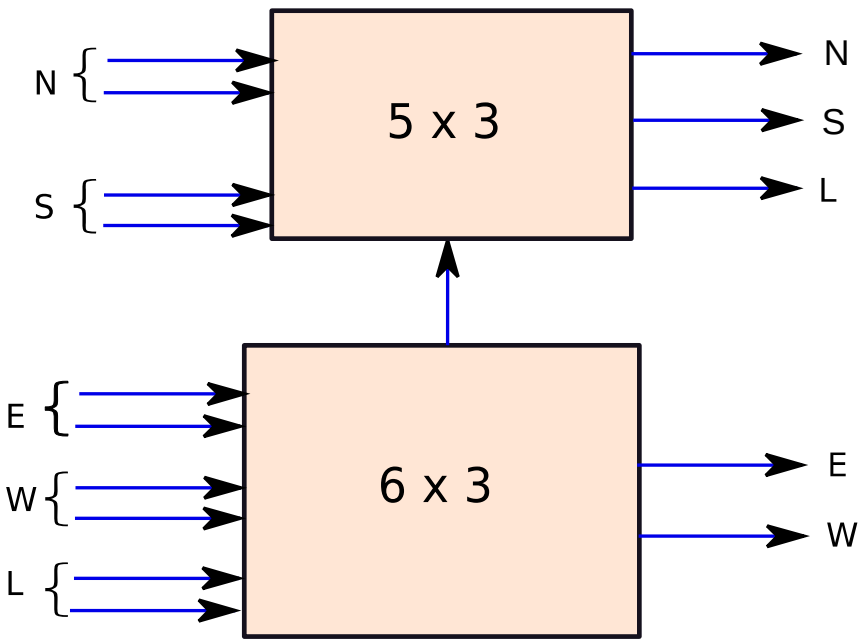
Area-Latency Tradeoffs

For an *m x n* switch:

1. Latency = m + n
2. Area = m * n

Design		Area cost	Latency cost
Level 1	**Level 2**		
10 x 5	--	50	15
Five 2 x 1	5 x 5	35	13
Two 5 x 2	4 x 5	40	16

Dimension Sliced Switch (Used for X-Y Routing)



Since no turns are allowed from (N/S) to (E/W), we can create two separate switches.

1. The lower router sends flits in a horizontal direction.
2. For any turn to the y-axis, it sends the flits to the upper router.
3. The upper router only routes flits on the y-axis, and delivers flits to the local tile.

Allocation and Arbitration

An arbiter chooses one out of N requests for resource allocation.

An allocator creates a one-to-one mapping between N requests and M resources

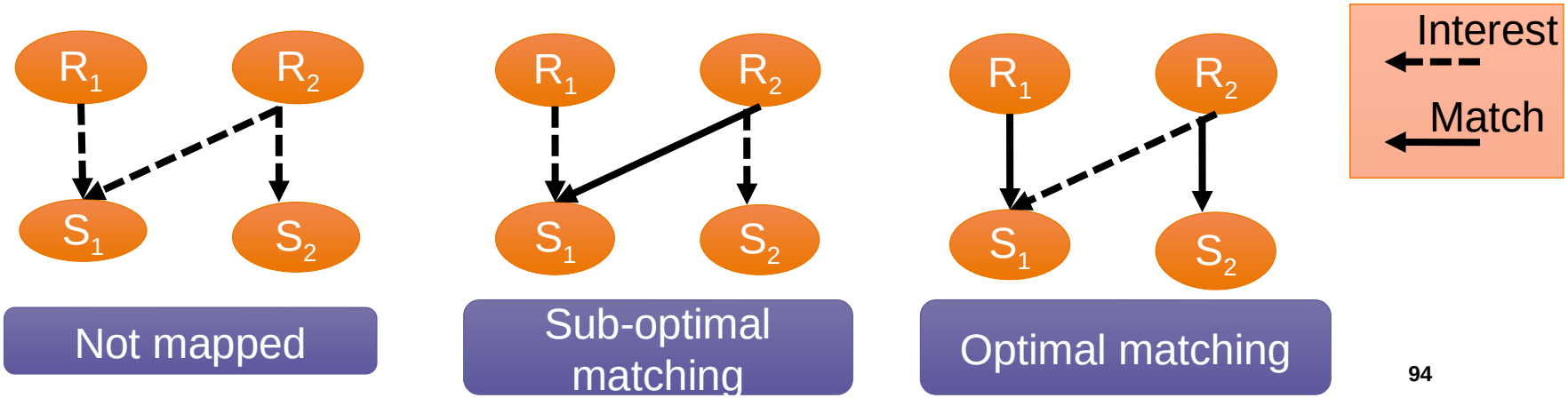
- Can either be created by chaining together arbiters.
- Or by creating a custom structure.

Matching between resources and requests

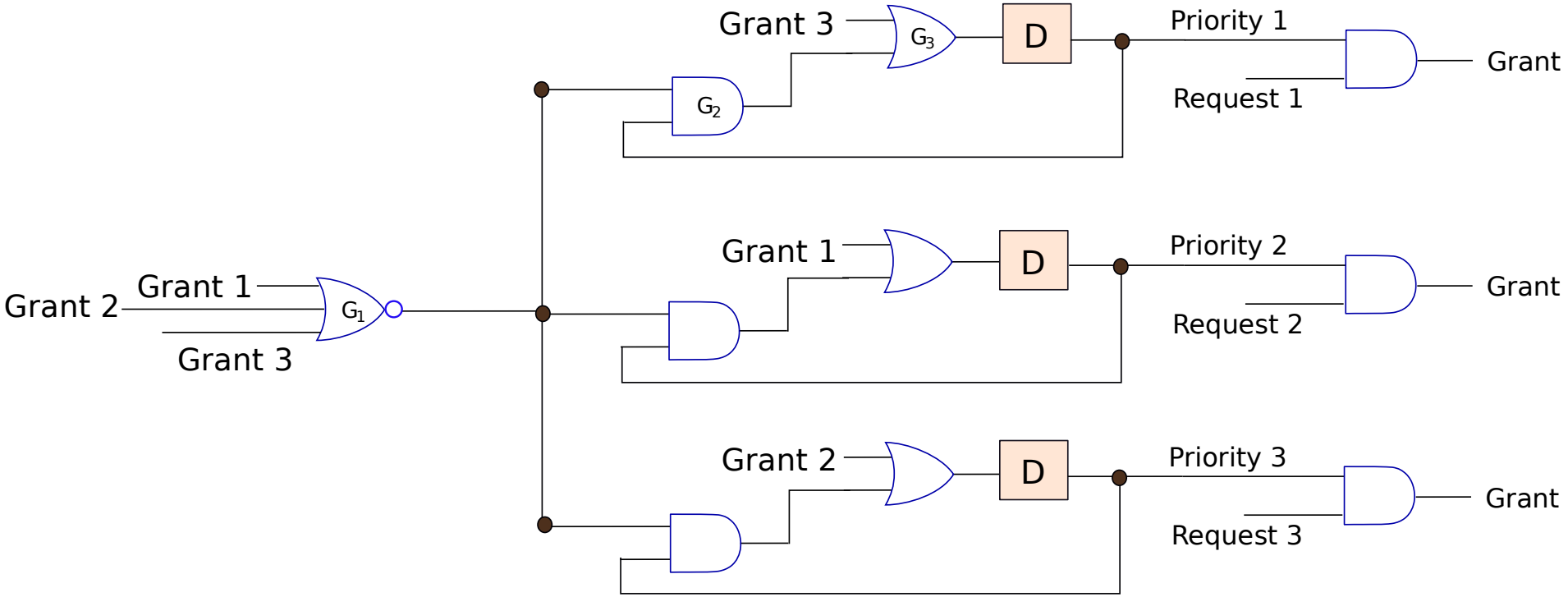
Condition 1: Every request is mapped to at most a single resource.

Condition 2: Every resource is mapped to at most a single request.

Condition 3: Request R_i is mapped to resource S_j if $f(R_i, S_j)$ is true.
The $f(R,S)$ function is true if request R is interested in resource S.



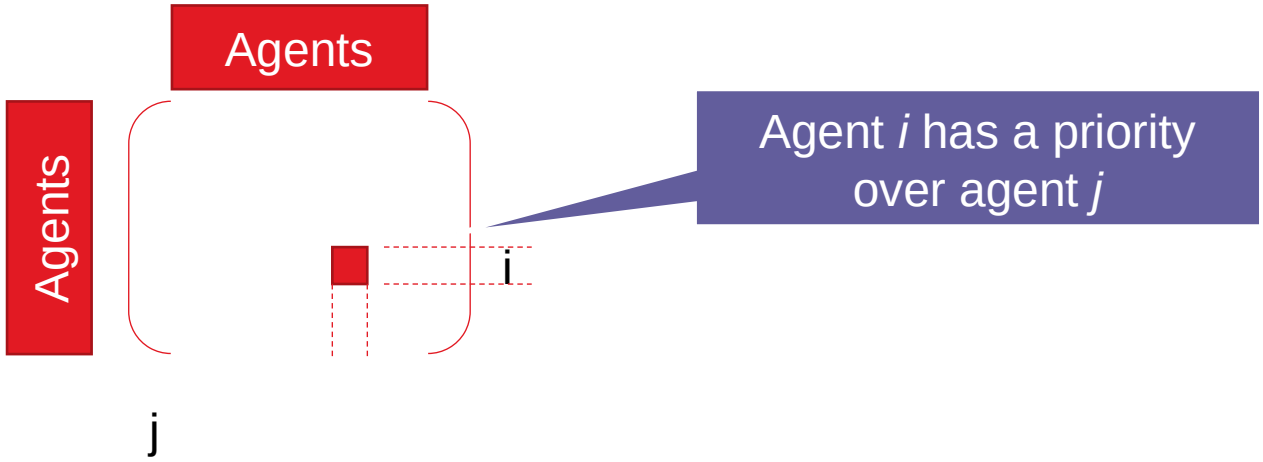
Round Robin Arbiter



- This is a simple scheme.

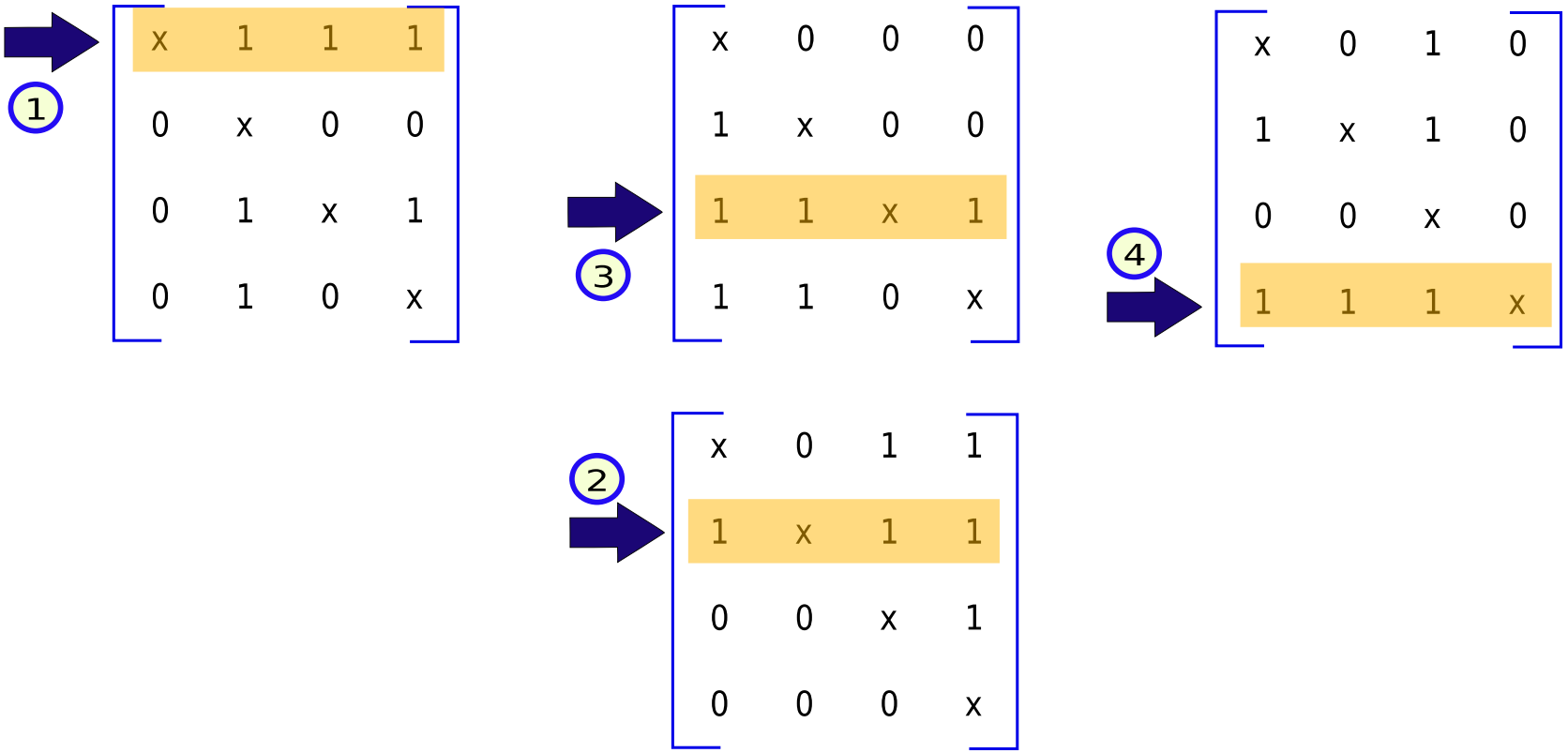
- No way of enforcing a priority between requests.
- If an agent is not interested, it is still given a chance.

Matrix Arbiter

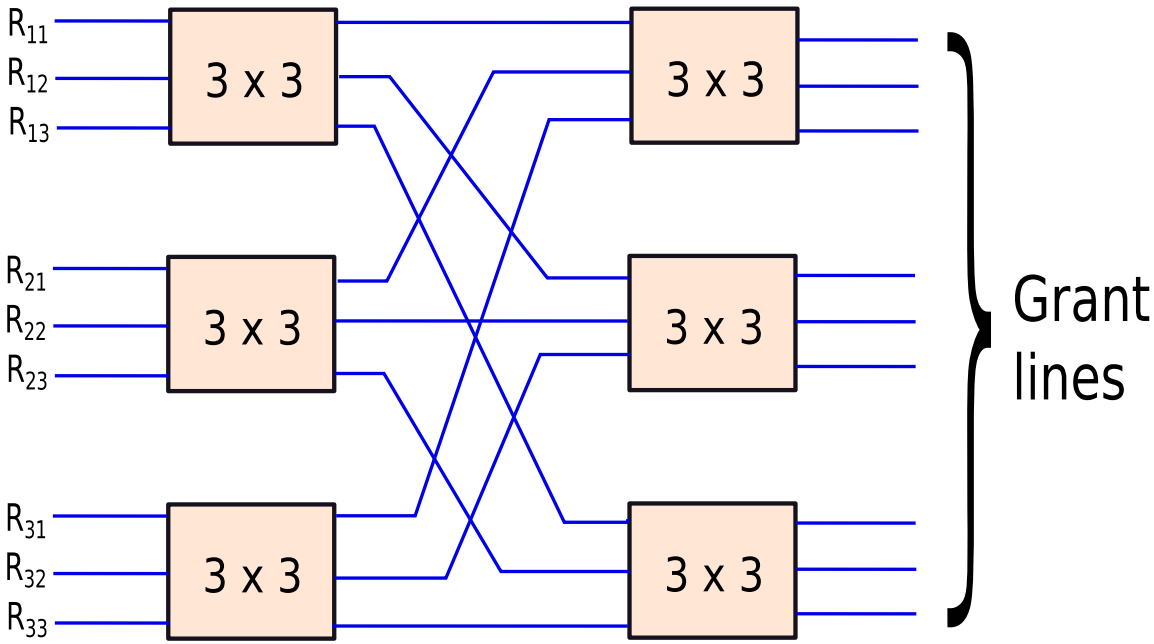


1. If a given agent is not interested, then it sets all the entries in its row to 0, and all the entries in its column to 1 (other than diagonal elements).
2. In every cycle the request is granted to the agent, which has 1s in all of the entries in its row other than the diagonal element.
3. Once the i^{th} agent is done servicing its request, it sets all the entries in its row to 0, and sets all the entries in its column to 1 (other than the diagonal elements). This means that it relegates itself to the lowest priority.

Matrix Arbiter



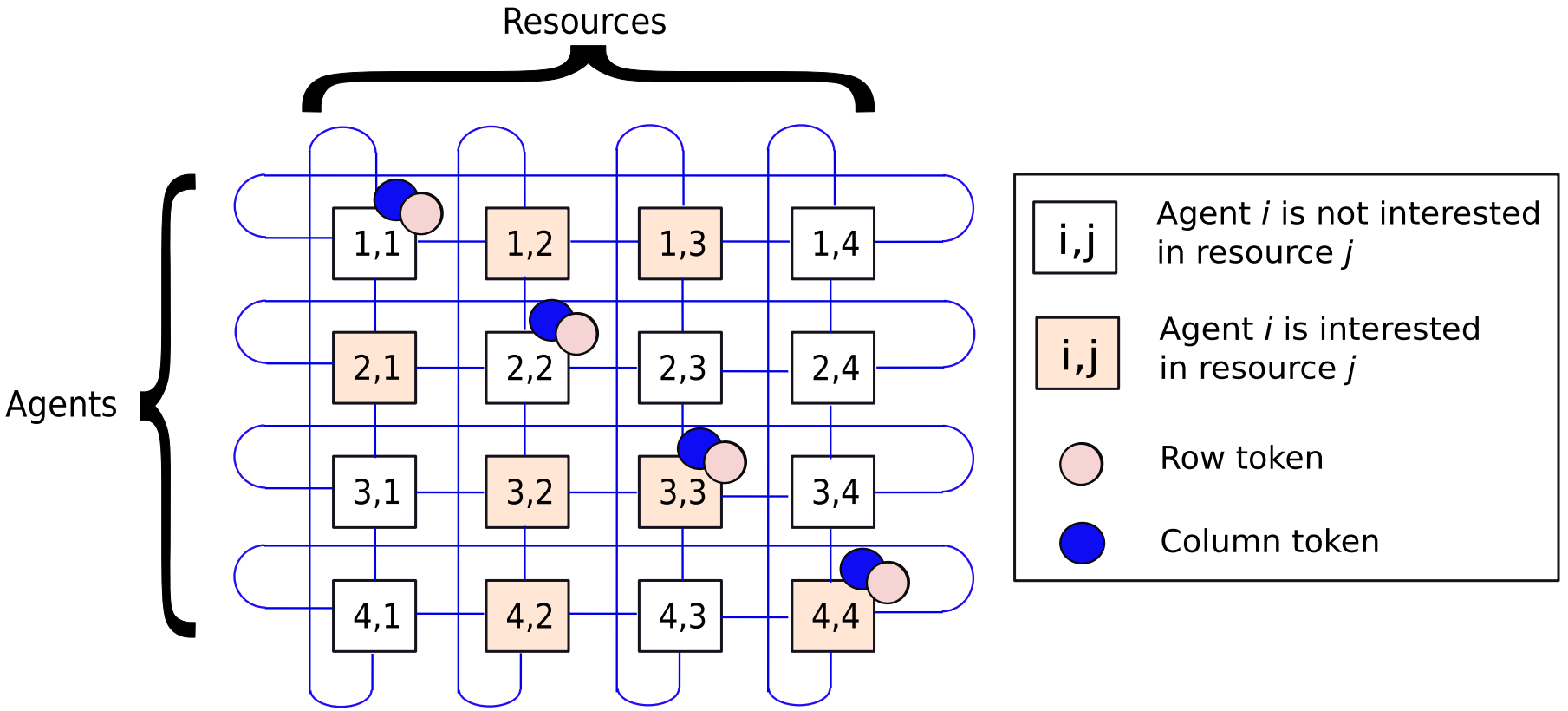
A Separable Allocator



R_{ij} = i^{th} agent is interested in the j^{th} resource

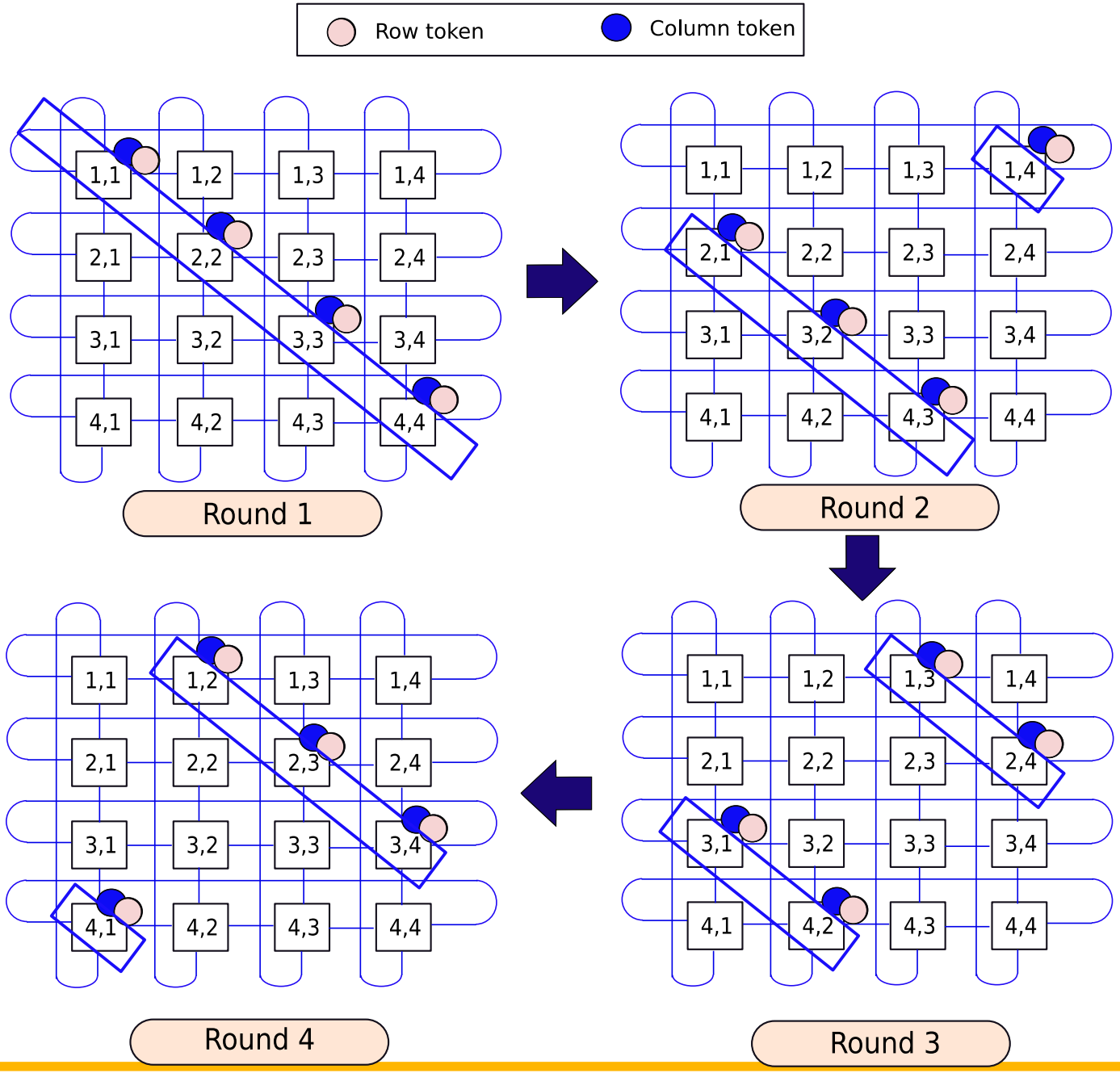
1. Possible that all the requests get selected for the same resource in the first column of switches.
2. Matching is not maximal.

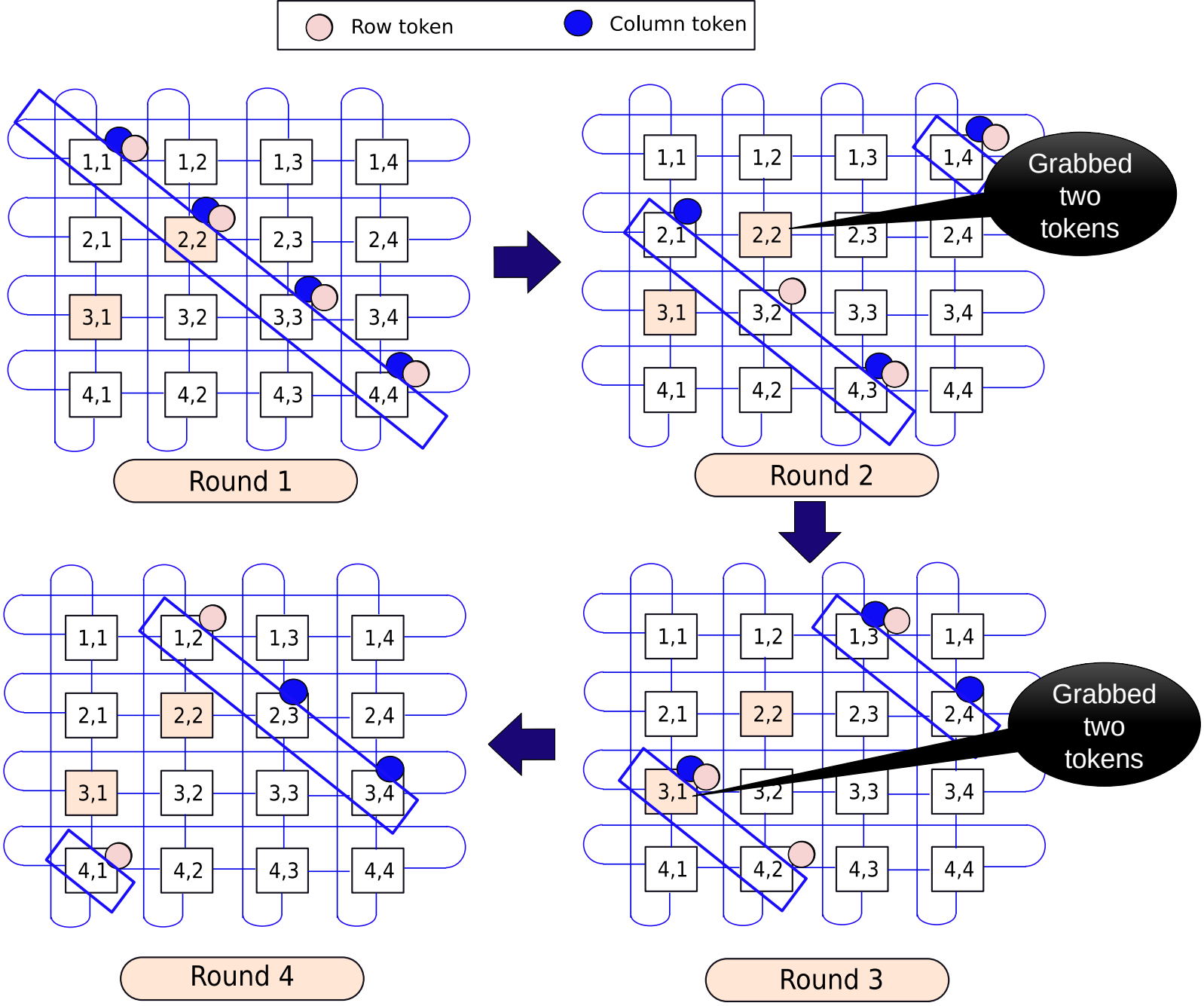
Wavefront Allocator



1. Start by giving a row and column token to each diagonal element.
2. Every round move the row token 1 step to the left, and the column token 1 step down (with wraparounds).

Example showing the movement of tokens.



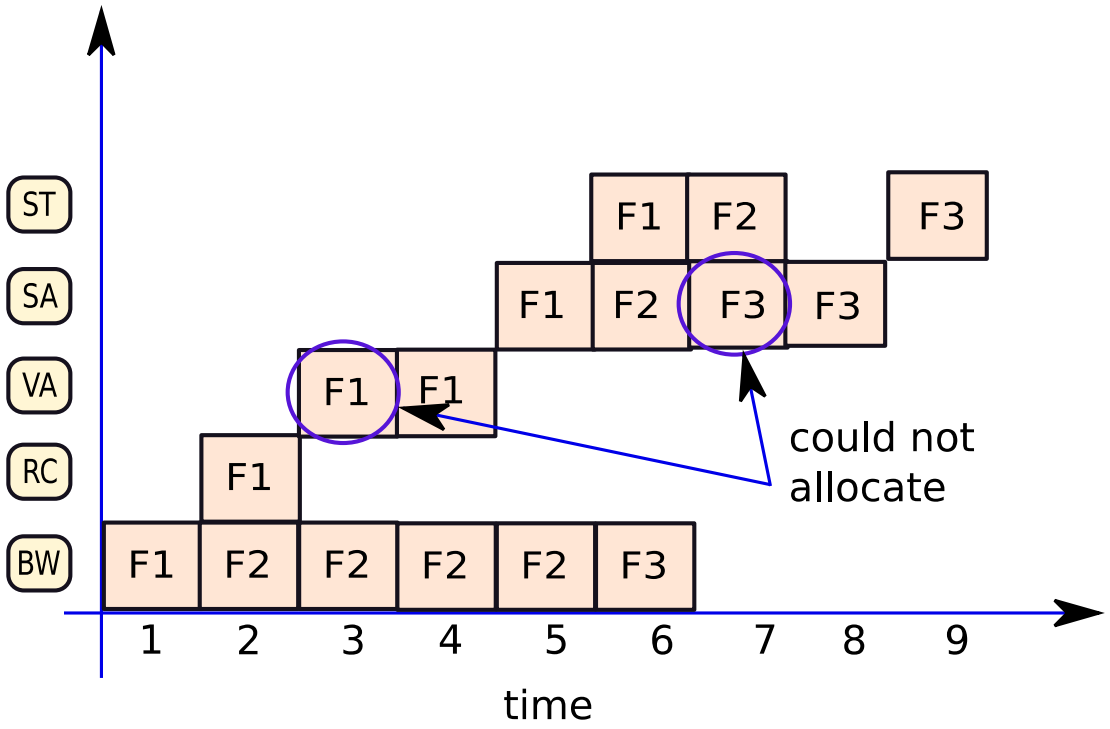

Row token
Column token
1,1
1,2
1,3
1,4
2,1
2,2
2,3
2,4
3,1
3,2
3,3
3,4
4,1
4,2
4,3
4,4
Round 1
Grabbed two tokens
Round 2
Round 3
Round 4

Properties of the Wavefront Allocator

- Once a token is absorbed, it is put out of circulation.
- This means that no cell in the same row (agent) or column (resource) can get both the tokens.
- This means that the matching that is produced is correct as per our criteria.

- It produces maximal matchings
- There is no matching that could have been done but was missed.
- Every cell gets the two tokens at some point of time, if no cell in the same row or column has been allotted its request. At that point of time that cell can be allotted the request.

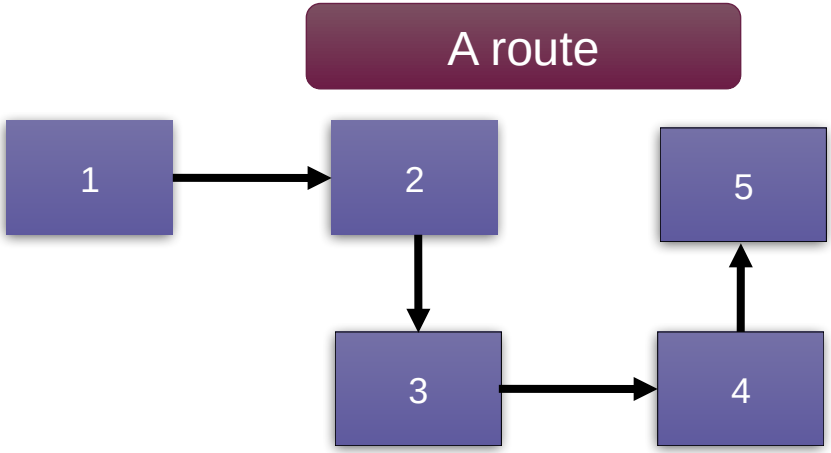
Router Pipeline Optimizations



Abbrev.	Name
BW	Buffer Write
RC	Route Computation
VA	VC Allocation
SA	Switch Allocation
ST	Switch Traversal

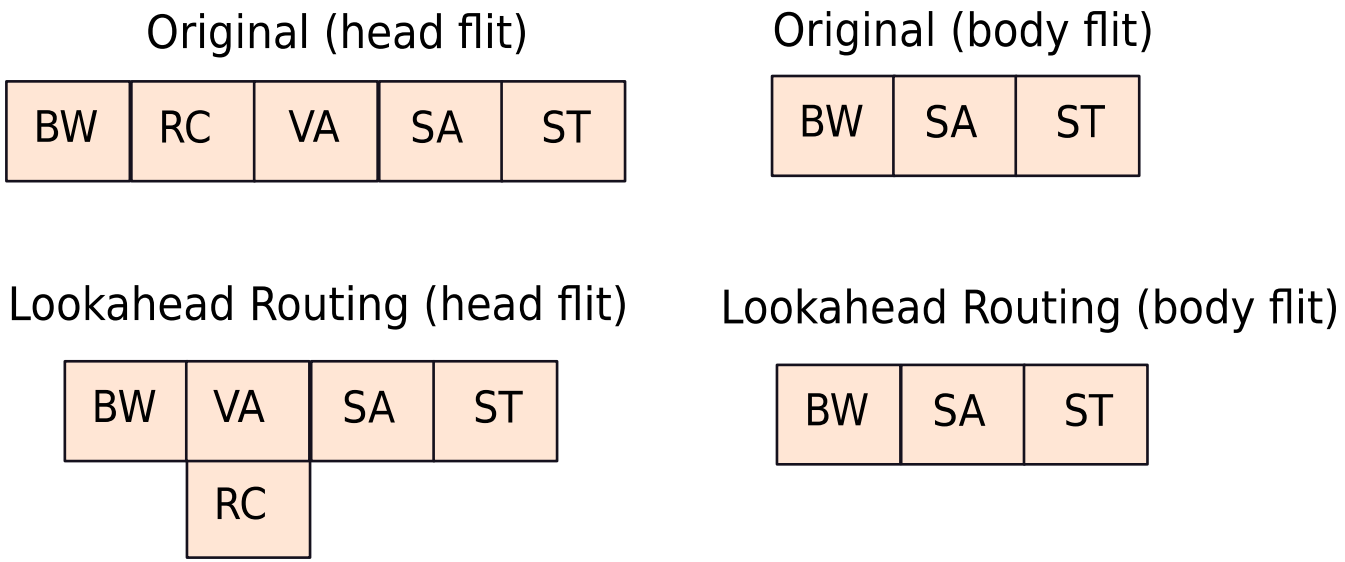
F1 is the head flit
Rest are body flits

Lookahead routing: Compute the route for the next hop.



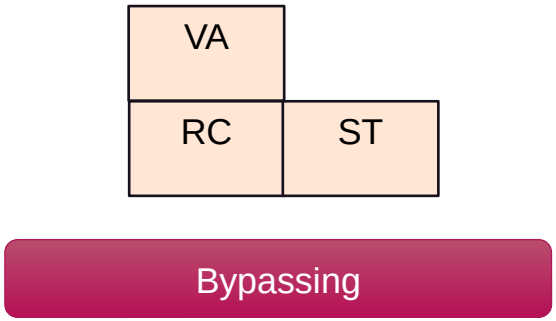
- Let's say that we already know that from Node 1 we will go to Node 2.
- Then when we are at Node 1, we compute the decision that Node 2 will take.
- Similarly, when we are at Node 2, we compute the decision that Node 3 will take.
- This routing decision is sent along with the packet.
- **Benefit**: Removes Route Computation from the critical path.

Pipeline diagrams



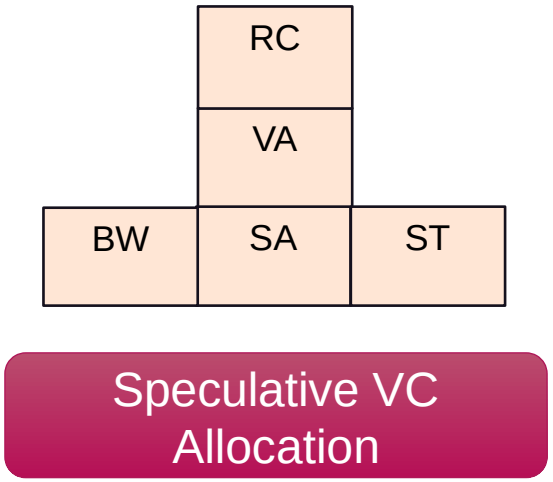
Bypassing

- If the router queues are empty
- Attempt to directly traverse the switch
- If there is a conflict, then resort to the conventional method



Speculative VC Allocation

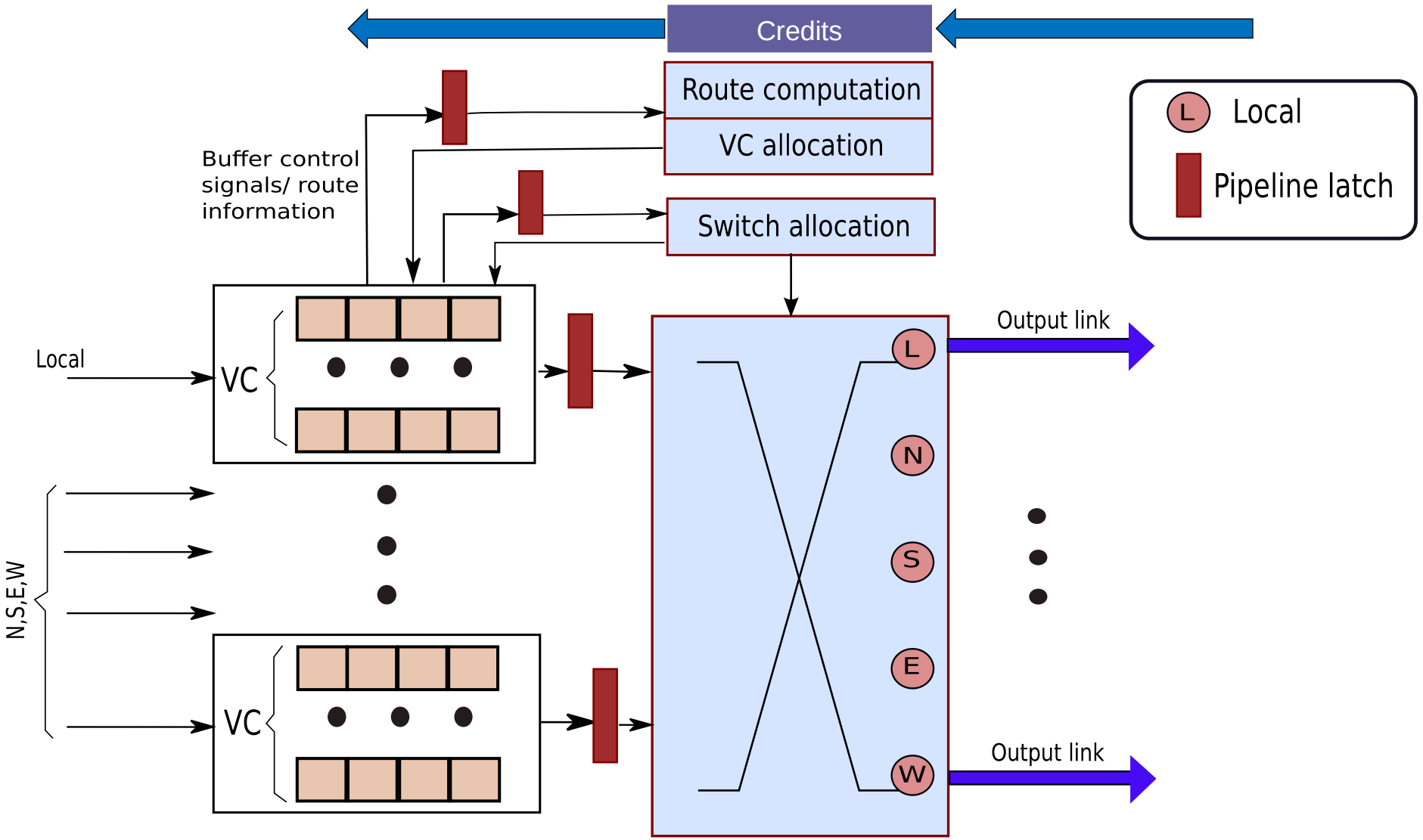
- Assume that we will find a free VC.
- After finding a free VC, send this information along with the head flit.
- If we do not find a VC, then follow the conventional method.



Late VC Selection

- Maintain a queue of free VCs with each outgoing link
- When a head flit traverses the switch, assign it a VC from the queue
- If a free VC is not available, cancel the process, and restart the conventional process

Design of the Router



Contents

1. Overview of an NoC
2. Message Transmission
3. Routing
4. Design of a Router
5. Non-Uniform Cache Architectures
6. Performance Aspects

Non-Uniform Caches

L2/L3 caches are large.

- If we find data in a nearby bank ⇨ a couple of cycles
- If it is a remote bank ⇨ 20-50 cycles (depending on the size of the network and the degree of congestion)

Cache access times can be extremely variable depending on where the data is

- This is called a non-uniform cache (NUCA cache)
- Can we take advantage of this non-uniformity?
- ANSWER: Place frequently used data closer

S-NUCA (static NUCA)

Simplest scheme:

- Map cache blocks to cache banks
- Give a cache block ␣ find which cache bank it maps to

- Tradeoffs: Bank locality vs homogenizing bank accesses

Mapping 1

Tag	Bank id	Set id	Byte within a block

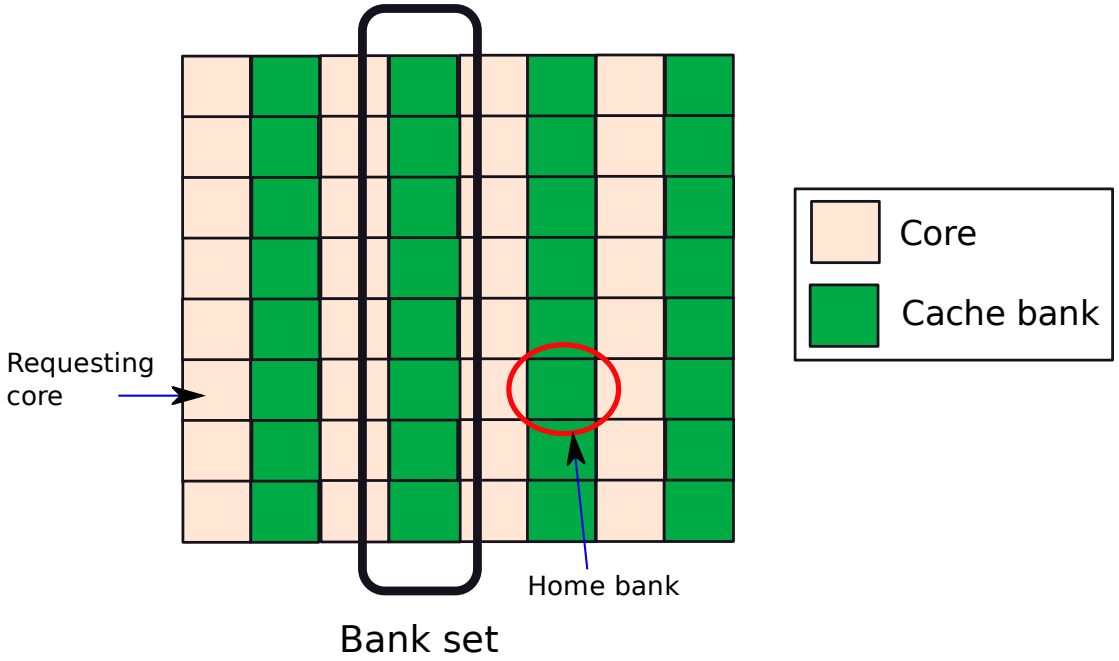
Mapping 2

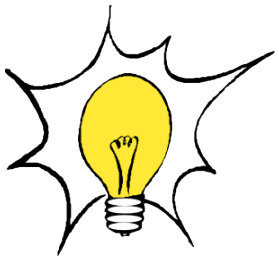
Tag	Set id	Bank id	Byte within a block

D-NUCA (Dynamic NUCA)

Let us now look at a method of bringing frequently used data closer

Bank set ⎵ sets of banks





Organize banks as columns (each column is a bank set)

Operation of the Protocol

Given an address ␣ Map it to a bank set

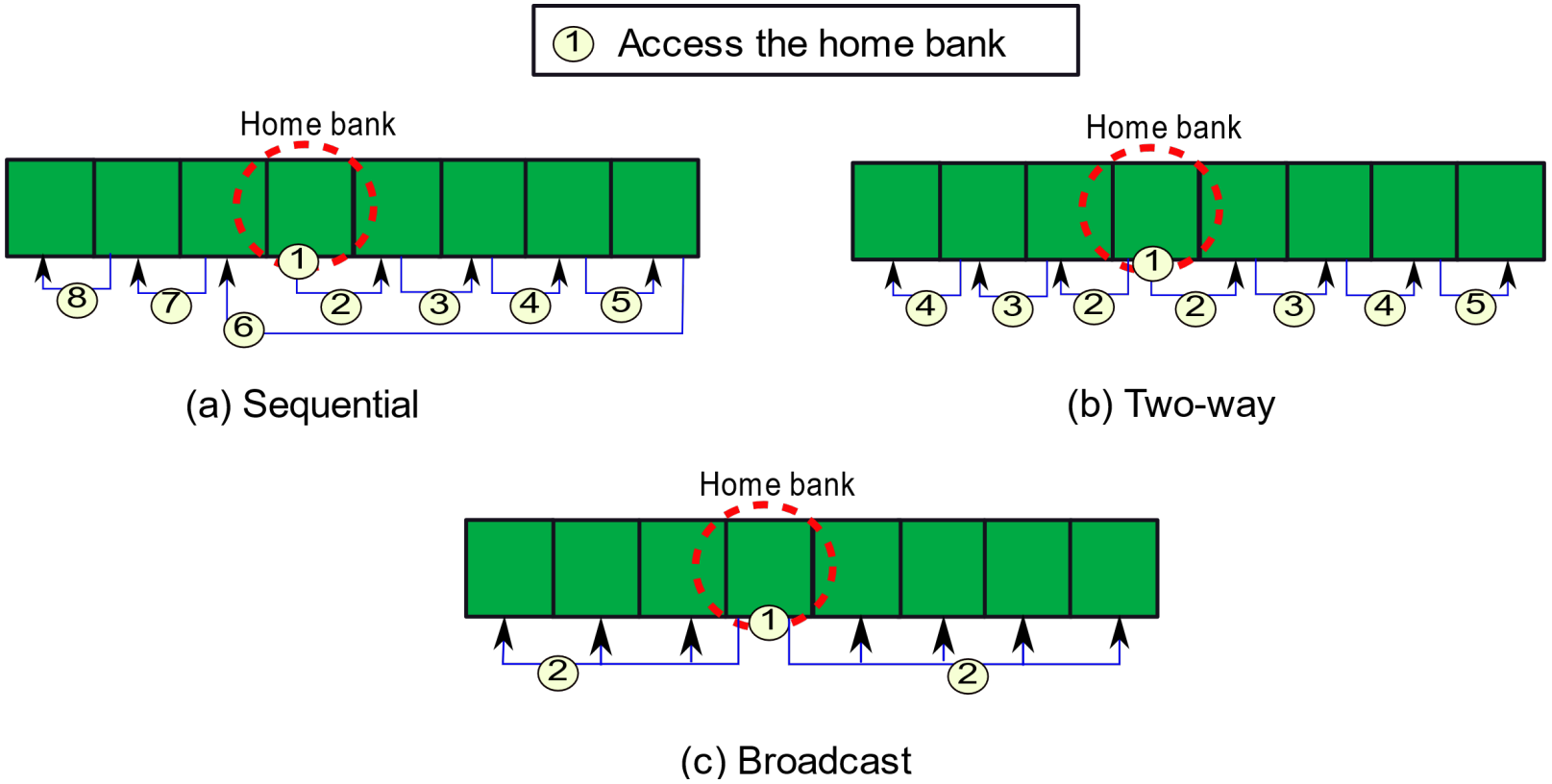
Search the bank set

- If we find the block, declare a cache hit
- Otherwise, declare a miss and fetch it from the lower level

Three search strategies

- Sequential
- Two-way
- Broadcast

Search Strategies: Start from the home bank (closest to the core in the bank set)



After a Hit/Miss Decision

Hit decision

- Send the block to the requesting core
- Promote the block ␣ Move it closer to the home bank (Migration)
- Bring it to a nearer bank (closer to the home bank)
- If there is an empty block in the set in the new bank ␣ no problem
- Otherwise swap the blocks
- Bringing blocks closer to the home bank (consistent with temporal locality)

Miss decision

- Fetch the block from the lower level
- Either insert it into the nearest bank or the farthest bank (depends on the strategy)

Eviction and Migration

Assume that a block needs to be evicted

- Solution 1: Move it to the lower level
- Solution 2: Move it away from the home bank. Need to tag each line with the id of the requesting core

Advantages

- Keeps data close to the requesting cores
- Brings data closer depending upon usage
- Gives a block multiple chances ⇒ Upon an eviction move it to bank slightly further away

Search policy, location, migration and eviction are important design decisions

Advanced Designs

Private data ⎵ Store it only within the bank in the tile

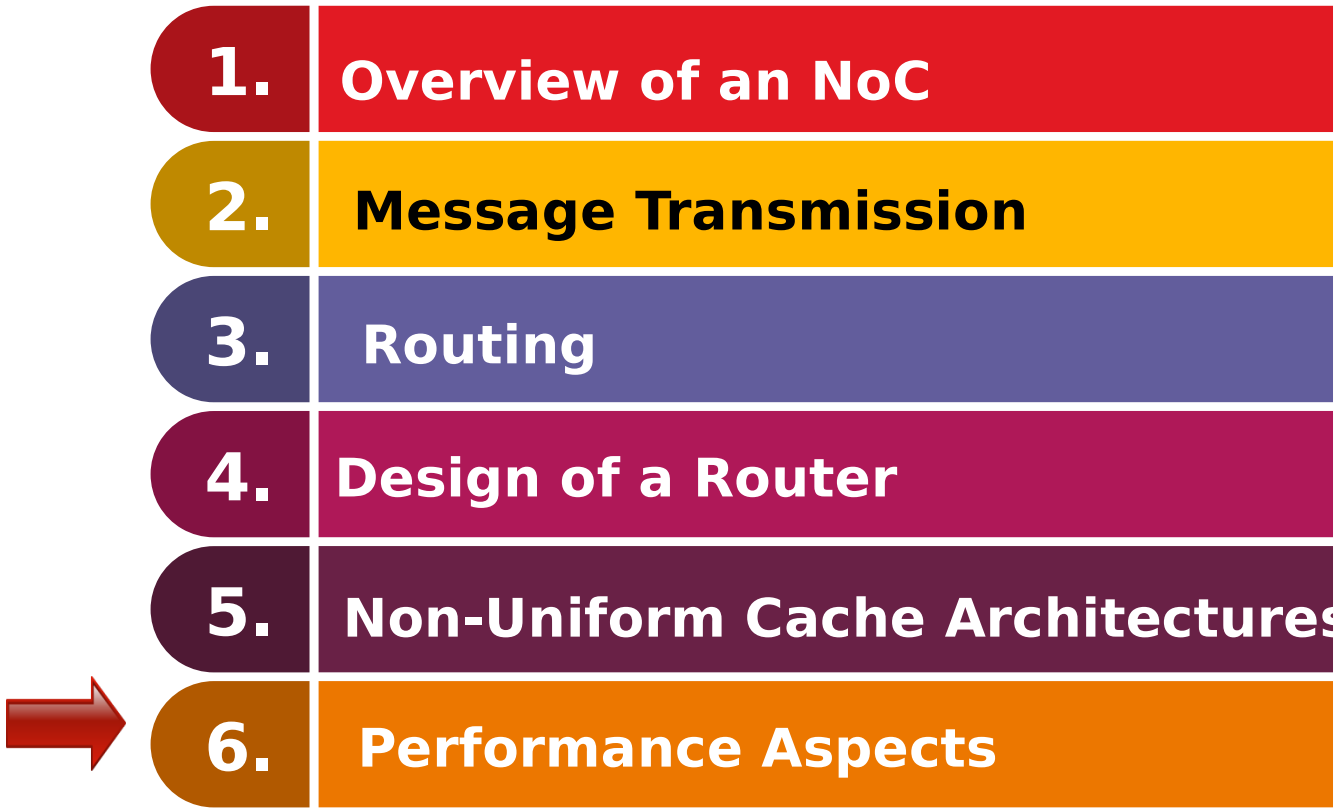
Shared data ⎵ Use SNUCA

Read-only data ⎵ Replicate

Advantages:

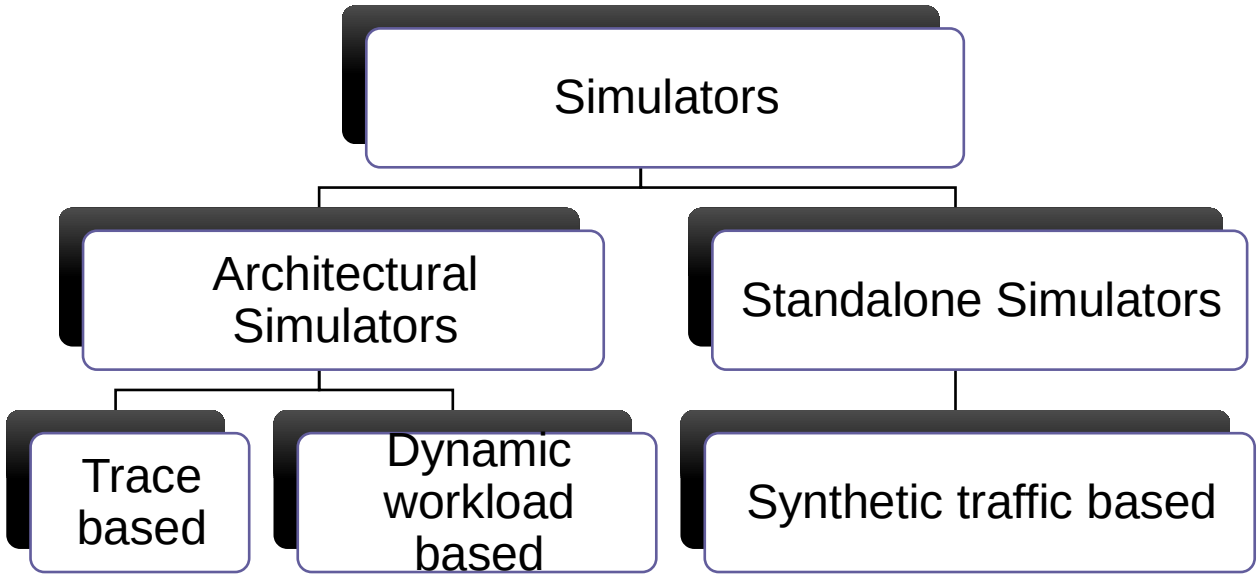
- Limited replication ensures good instruction hit rates
- Replicating all data is tricky ⎵ need to synchronize replicas

Contents


1.
Overview of an NoC
2.
Message Transmission
3.
Routing
4.
Design of a Router
5.
Non-Uniform Cache Architectures
6.
Performance Aspects

Performance Evaluation Techniques

- Evaluation Metrics
- Latency
- Throughput
- Energy or Power
- Area and routing complexity



Architectural Simulation

- Simulate the network routers and links.
- Get latency, throughput, and power/energy statistics.

Can .

Slow

simulate NoCs accurately for real workloads

Synthetic Traffic Generation

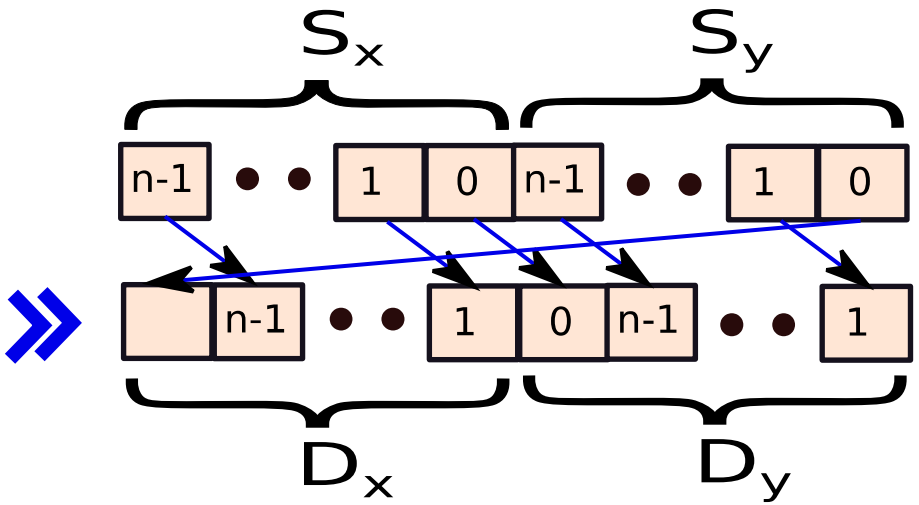
- Consider a mesh (for the sake of simplicity)
- Every node has an (x,y) coordinate
- Synthetic traffic: set of source ␣ destination pairs

Simple Patterns

Pattern	Src ␣ Dest	Remarks
Random	src ␣ random	Random traffic simulation
Bit-Complement		Used to test the NoC throughput
Transpose		Direction of traffic: perpendicular to the diagonal
Bit-Reverse	D[i] = S[2n – 1 – i]	Just reverse the bits. Used in FFTs

Bit-Rotation

Similar to a right shift.



(S_x, S_y)

(D_x, D_y)

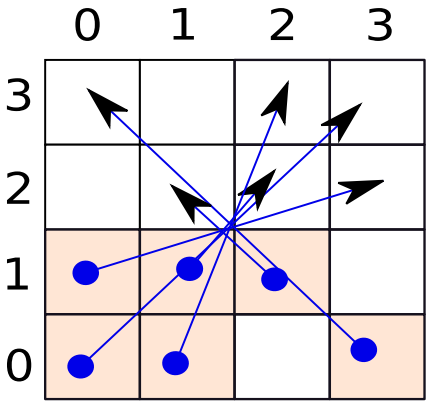
Shuffle and Tornado Patterns

Shuffle

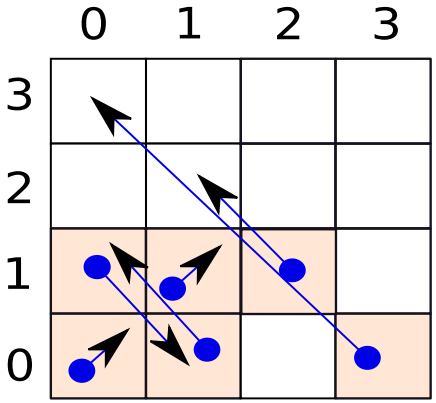
Similar to a left shift.

Pattern	Src ⊟ Dest	Remarks
Tornado		Seen while solving a system of differential equations

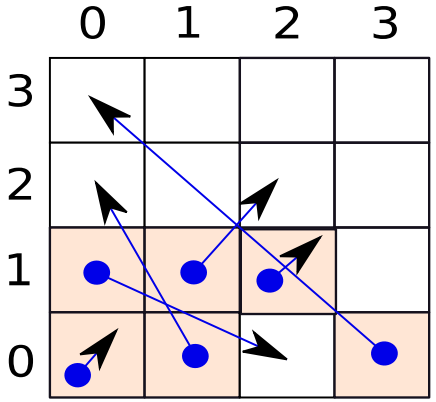
Faster approach: Synthetically generate traffic



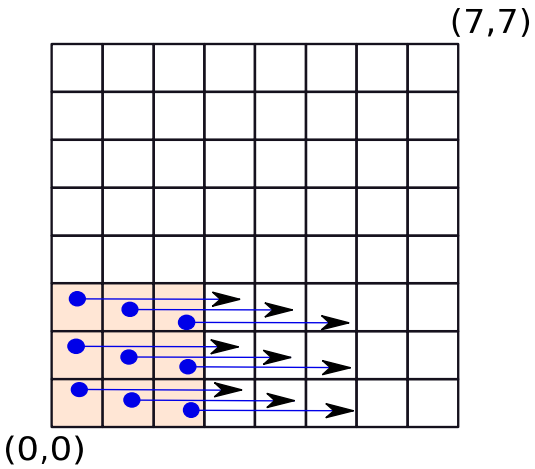
Bit-Complement



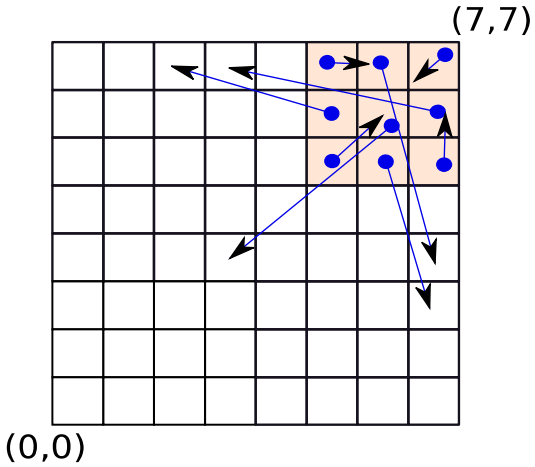
Transpose



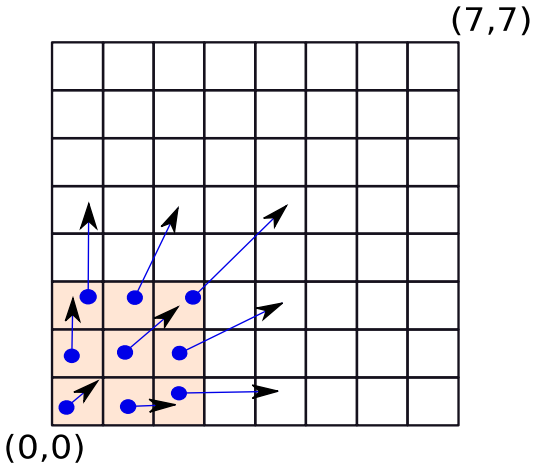
Bit-Reverse



Tornado

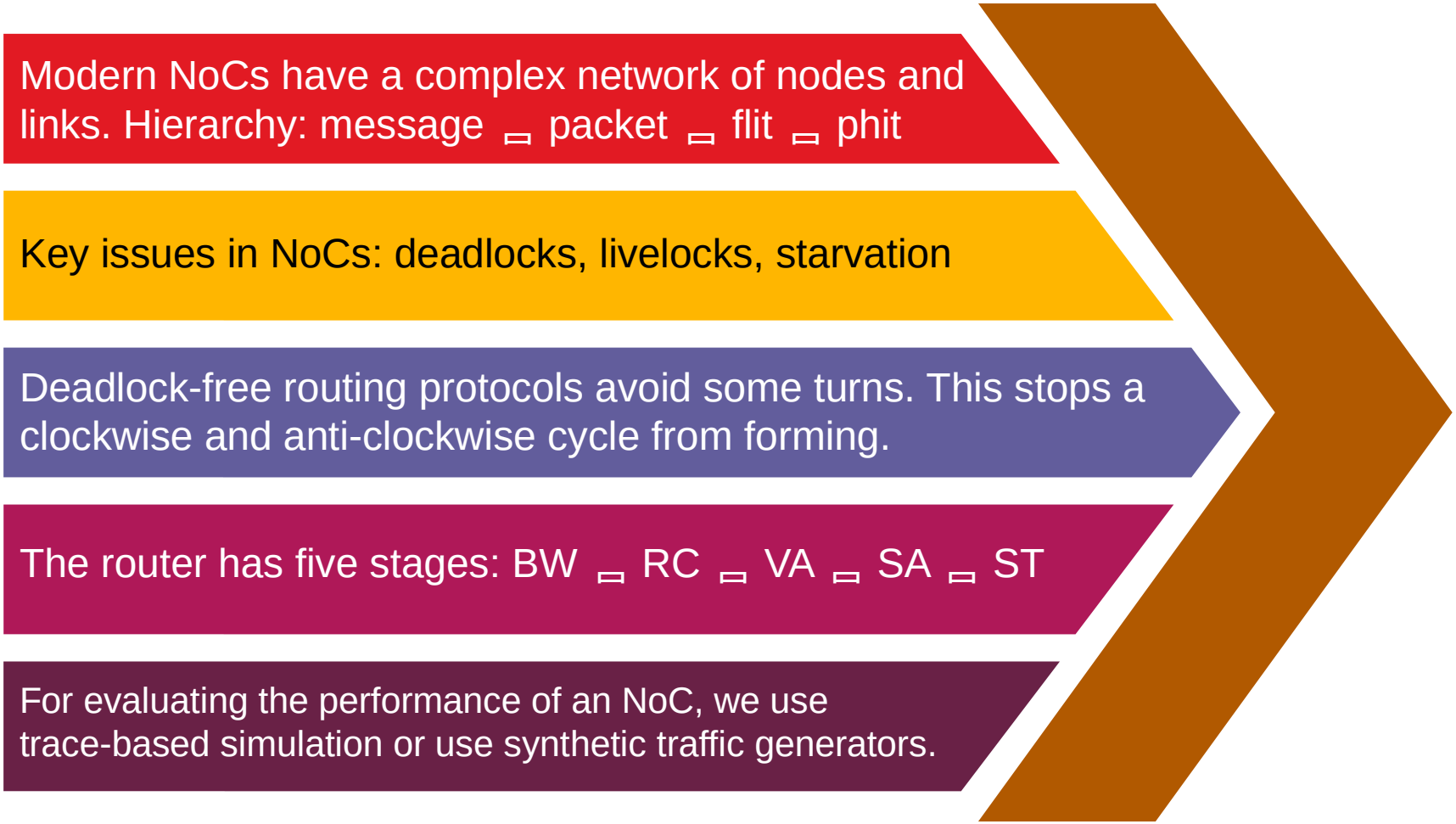


Bit-Rotation



Shuffle

Conclusion


Modern NoCs have a complex network of nodes and links. Hierarchy: message ▭ packet ▭ flit ▭ phit
Key issues in NoCs: deadlocks, livelocks, starvation
Deadlock-free routing protocols avoid some turns. This stops a clockwise and anti-clockwise cycle from forming.
The router has five stages: BW ▭ RC ▭ VA ▭ SA ▭ ST
For evaluating the performance of an NoC, we use trace-based simulation or use synthetic traffic generators.

The End