Scalable Cache Coherence & Interconnection Networks

Cosmin E. Oancea
cosmin.oancea@diku.dk

Department of Computer Science (DIKU)
University of Copenhagen

October 2022 PMPH Lecture Slides

Course Organization

W	HARDWARE		SOFTWARE	LAB/CUDA
1	Trends Vector Machine	⟵	List HOM (Map-Reduce)	Intro & Simple Map Programming
2	In Order Processor	⟶ ⟵	VLIW Instr Scheduling	Scan & Reduce
3	Cache Coherence		Reasoning About Parallelism	Sparse Vect Matrix Mult
4	Interconnection Networks		Case Studies & Optimizations	Transpose & Matrix Matrix Mult
5	Memory Consistency		Optimising Locality	Sorting & Profiling & Mem Optimizations
6	OoO, Spec Processor		Thread-Level Speculation	Project Work

Three narative threads: the path to complex & good design:

- Design Space tradeoffs, constraints, common case, trends.
- Reasoning: from simple to complex, Applying Concepts.

1. Scalable Shared Memory Systems

2. Interconnection Networks (IN)
   - Design Space, Hardware Components, Communication Models
   - Switching Strategies: Circuit and Packet Switching
   - Latency And Bandwidth Models
   - Indirect Interconnect-Network Topologies
   - Direct Interconnect-Network Topologies
   - Routing Algorithms and Deadlock Avoidance

Bus-Based SMPs Do Not Scale

Ideally, memory bandwidth should scale linearly with the number of nodes, and memory latency should remain constant.

Bus-Based Multiprocessors are inherently NON-Scalable:

- Bus Bandwidth capped by `(# of bus wires)`$\times$`clockrate`, and
- actually decreases as more noded are added, because the wire length and the load on them increases with the number of nodes.

- Dance Hall: memory latency about the same for any access, but as the # of procs goes up, the latency of all accesses increases.

Bus-Based SMPs Do Not Scale

Ideally, memory bandwidth should scale linearly with the number of nodes, and memory latency should remain constant.

Bus-Based Multiprocessors are inherently NON-Scalable:

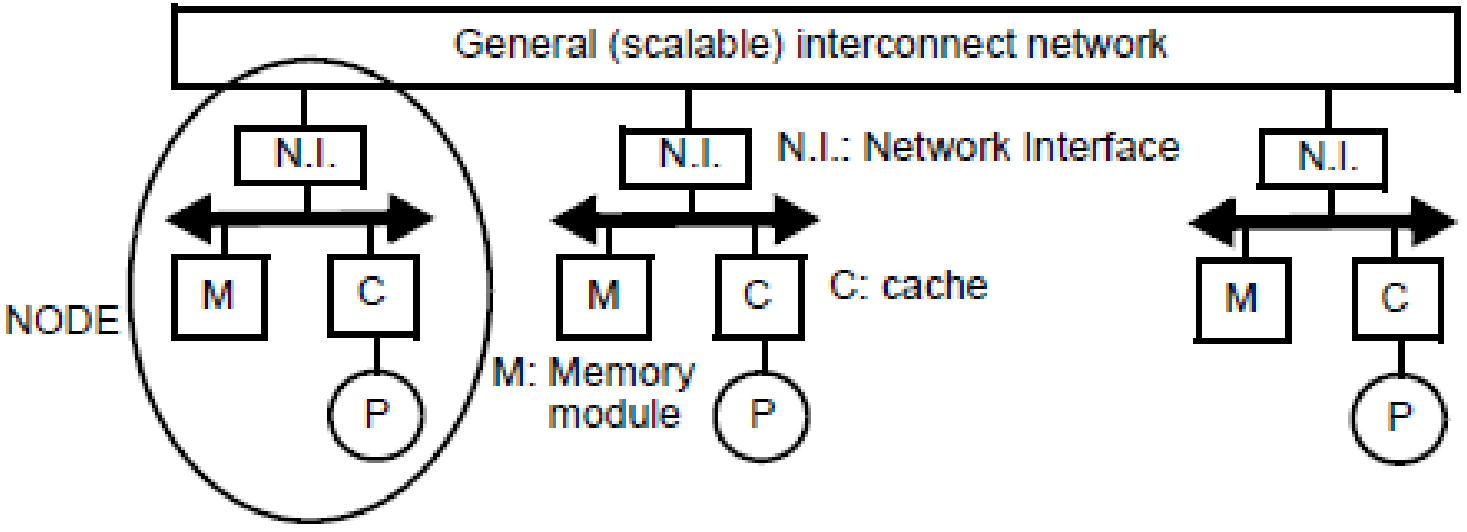
- Bus Bandwidth capped by `(# of bus wires)`×`clockrate`, and
- actually decreases as more noded are added, because the wire length and the load on them increases with the number of nodes.

- Dance Hall: memory latency about the same for any access, but as the # of procs goes up, the latency of all accesses increases.

- Snoopy (Broadcast) Based Protocols are NOT Scalable, because involve *all* nodes in *all* coherence transactions.

cc-NUMA: Cache Coherent Non-Uniform Mem Arch

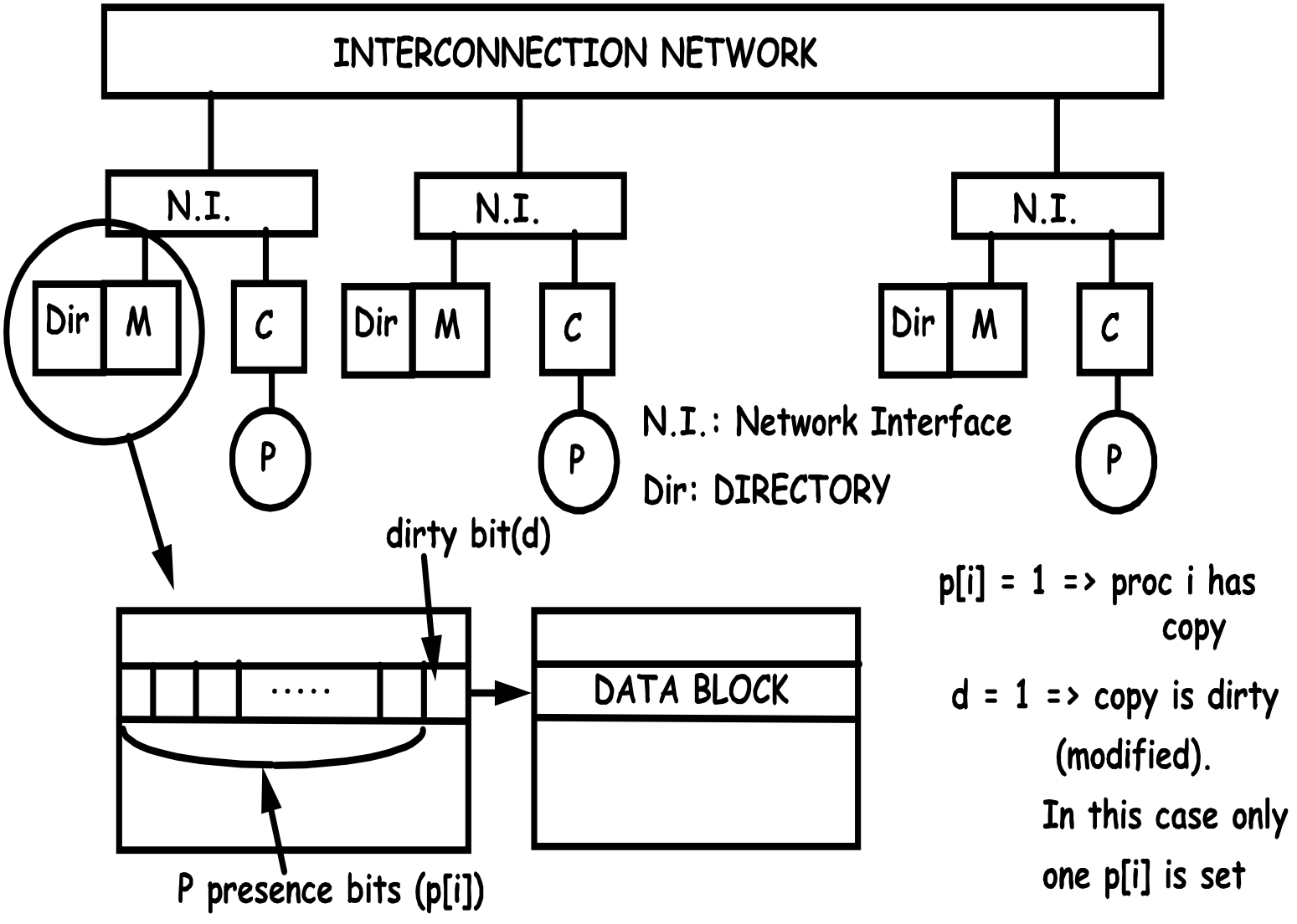
Def. Scalable: memory bandwidth should grow linearly and memory access latency should grow sub-linearly with the number of nodes.



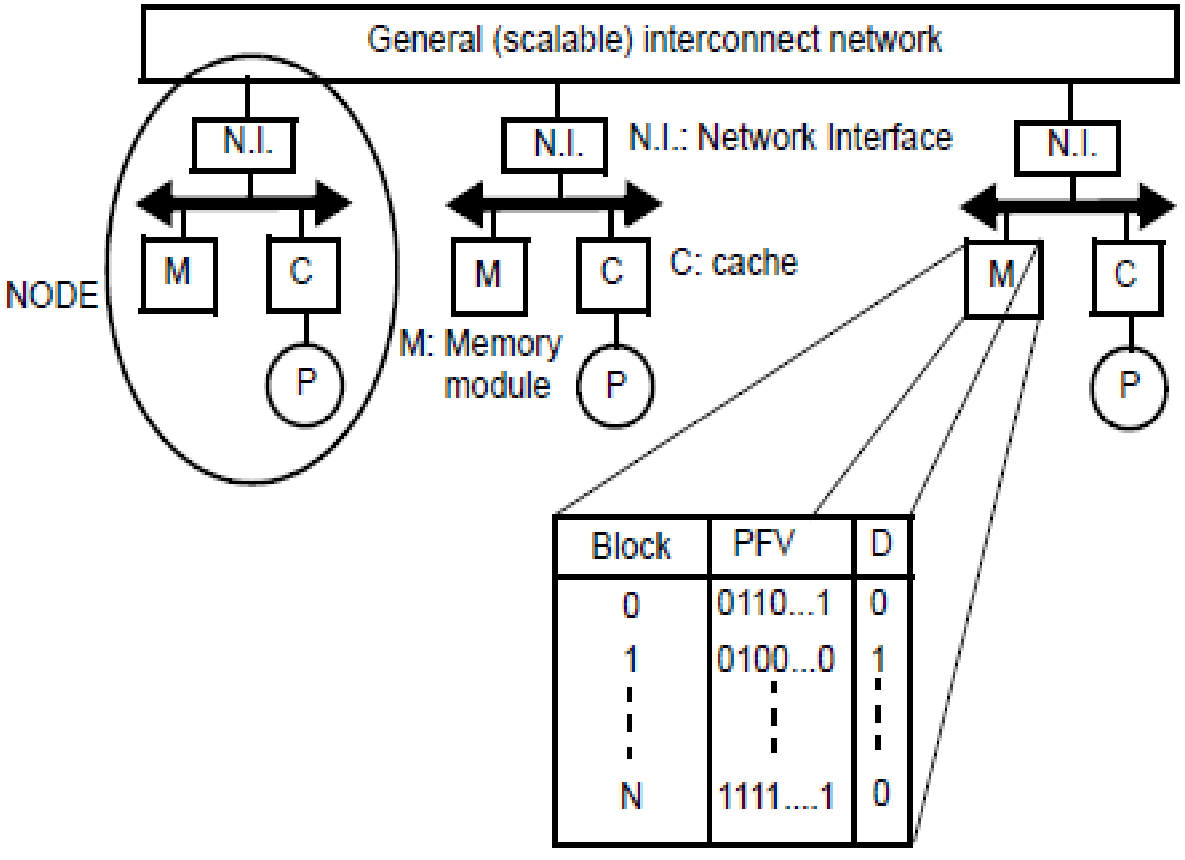
cc-NUMA because latency of a local mem module is $<$ of a remote:

- Memory partitioned into banks & distributed across nodes to leverage locality. Can be Main Mem or shared cache (NUCA).
- Uses a scalable interconnect to accomodate bandwidth,
- Uses scalable directory protocols to keep track of sharing.

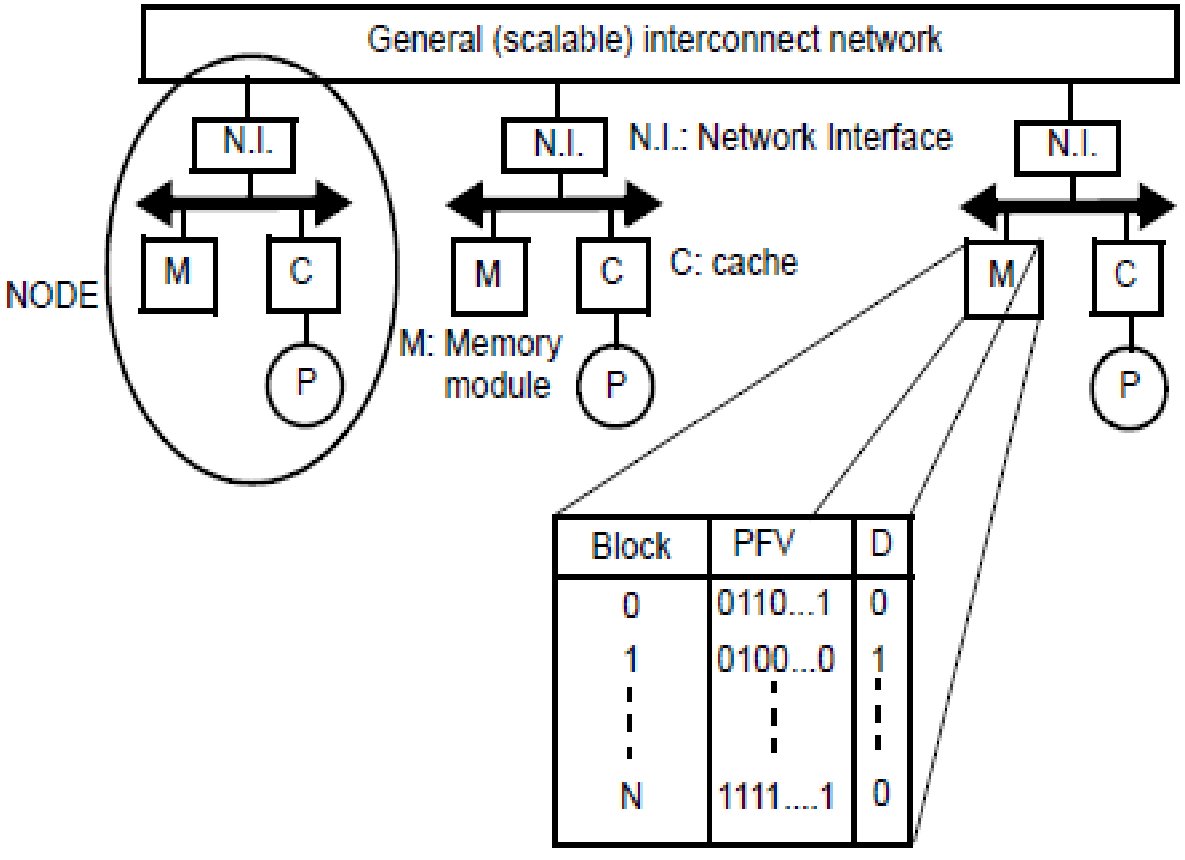
Hardware Structure for Baseline Directory Protocol



Presence-Flag Vector Scheme



Presence-Flag Vector Scheme



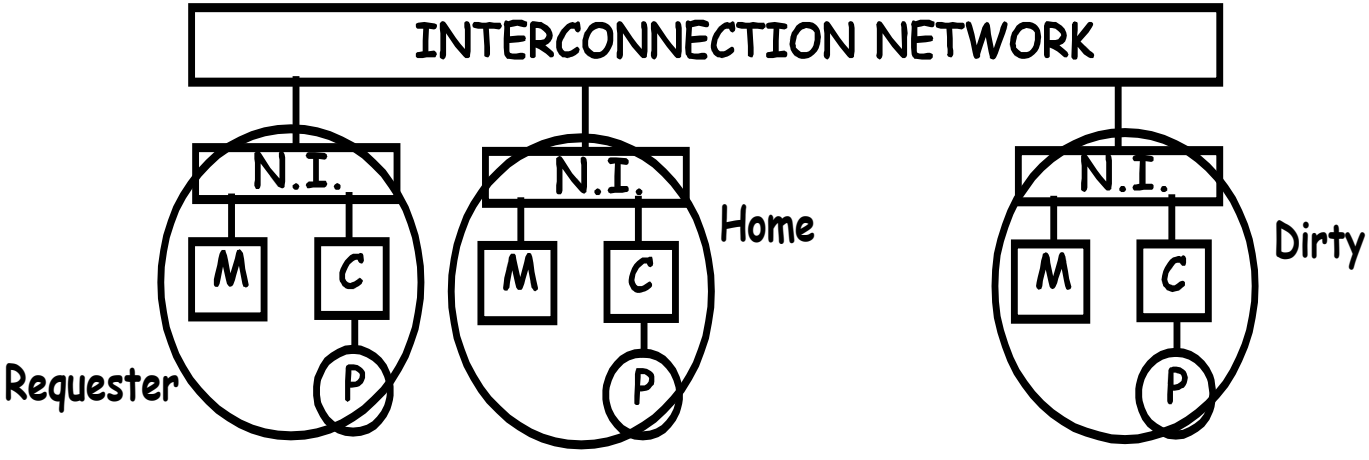
A memory block can be in two states: Clean or Dirty. Example:

- Block 1 is cached by processor 2 only and is Dirty, i.e., memory is stale and the only valid copy is at a remote node.
- Block N is cached by all procs and is Clean (mem is up to date).

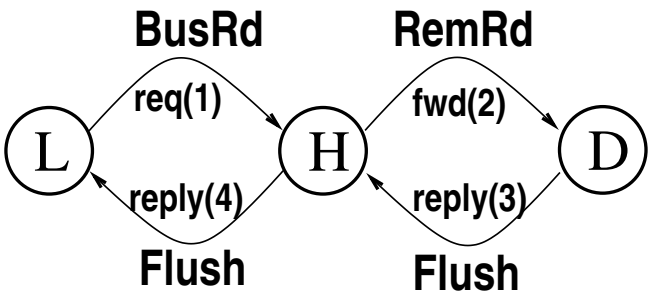
cc-NUMA Protocols

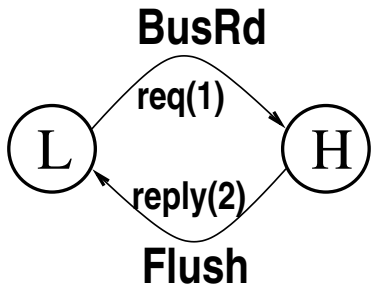
Protocol is similar to MSI-Invalidate (or MSI-Update or MESI), but without broadcast. Instead uses only the (protocol) agents:

- Home Node (H) is the node where the memory block and its directory entry reside,
- Local Node (L) or requester is the node initiating the request,
- Remote Node (R) is any other node participating in transaction:
  - Dirty Node (D) is the node holding the latest modified copy,
  - Shared Nodes (S) are the nodes holding a shared copy.
- Home may be the same as Local or Dirty.

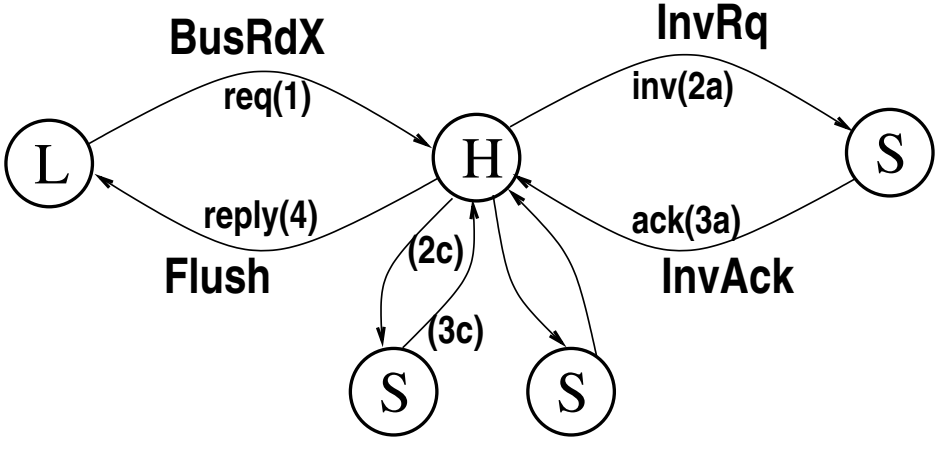


MSI Invalidate in ccNUMA



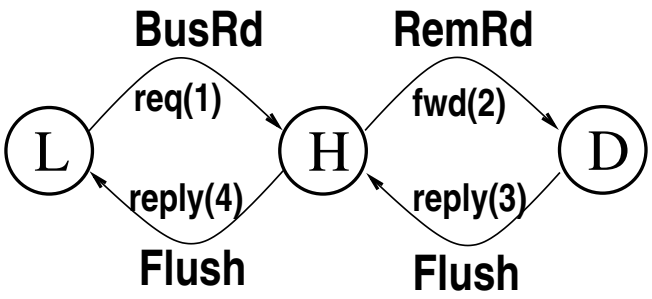


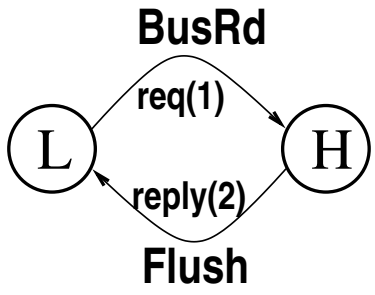
(1) Read Miss on Dirty & Clean Blocks



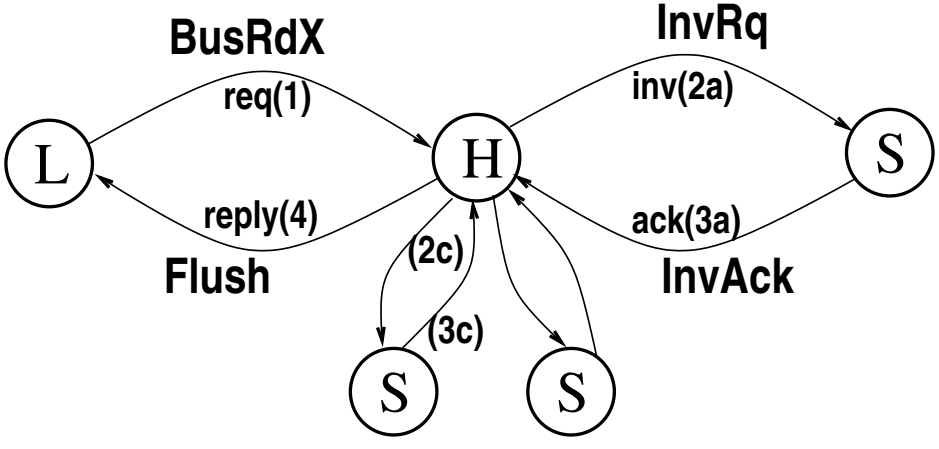
(2) Write Miss on Shared Block

MSI Invalidate in ccNUMA





(1) Read Miss on Dirty & Clean Blocks



(2) Write Miss on Shared Block

- Read Miss on Dirty Block: local node send *bus-read* request to host ⇒ Since memory (host) is stale, host sends *remote-read* request to dirty node, i.e., the only one set in PFV entry ⇒ D sends copy to H ⇒ H updates memory and PFV, and sends copy to local (4 hops).
- Read Miss on Clean Block: H sends its valid copy to L (2 hops).
- Write Miss on Shared Block: L signals H ⇒ H sends *invalidations* to all sharing nodes (found in `PFV`) ⇒ When host receives the acks from all S nodes it updates `PFV` (clears all but dirty L) and flushes block to L. (4 hops if parallel invs).
- Write Miss on Dirty Block: dirty node flushes value to L via H.

Block Eviction & Race Conditions

When a block is evicted upon replacement:

- If dirty then OK, i.e., H automatically notified by L's write-back.
- If no dirty then silent eviction is still correct, but affects perform: Tradeoff between the overhead of keeping PFV accurate and the overhead of processing useless invalidations.

Race Conditions: same-block transactions must not interleave. How?

Block Eviction & Race Conditions

When a block is evicted upon replacement:

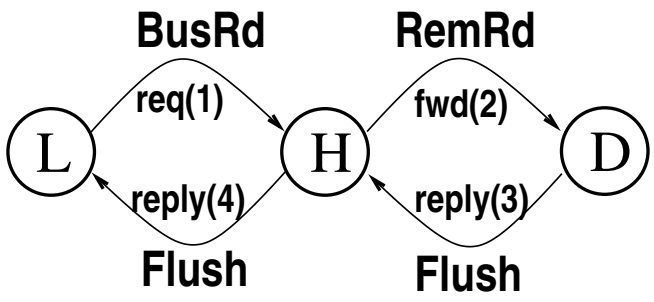
- If dirty then OK, i.e., H automatically notified by L's write-back.
- If no dirty then silent eviction is still correct, but affects perform: Tradeoff between the overhead of keeping PFV accurate and the overhead of processing useless invalidations.

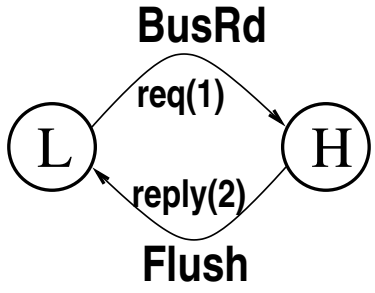
Race Conditions: same-block transactions must not interleave. How?

1. Home node is the central arbiter.
2. Use a lock bit per entry to signal in-progress transaction.
3. If block-bit set and another transaction then:
   Queueing incomming requests until buffer full then NACK $\Rightarrow$
   reduced latency and bandwidth and simple overflow treatment.

When should the Lock Bit be cleared/turned off?

When to Turn Off the Lock Bit?

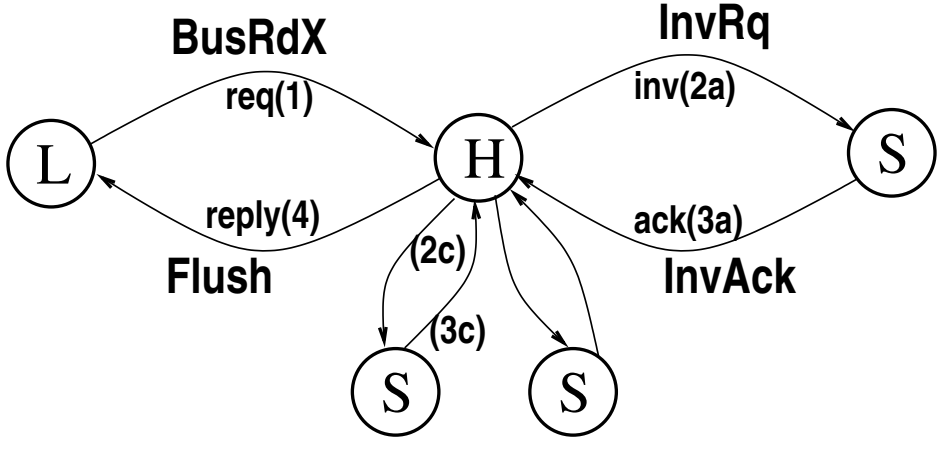




Example using Fig (2) then (1):

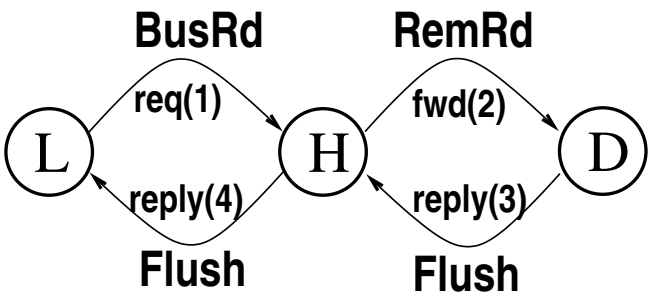
- The latest time the lock bit can be turned off is just before H `Flushes` to L. Not good enough:

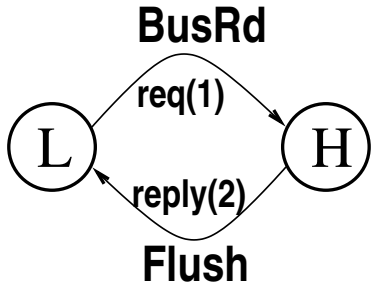
(1) Read Miss on Dirty & Clean Blocks



(2) Write Miss on Shared Block

When to Turn Off the Lock Bit?

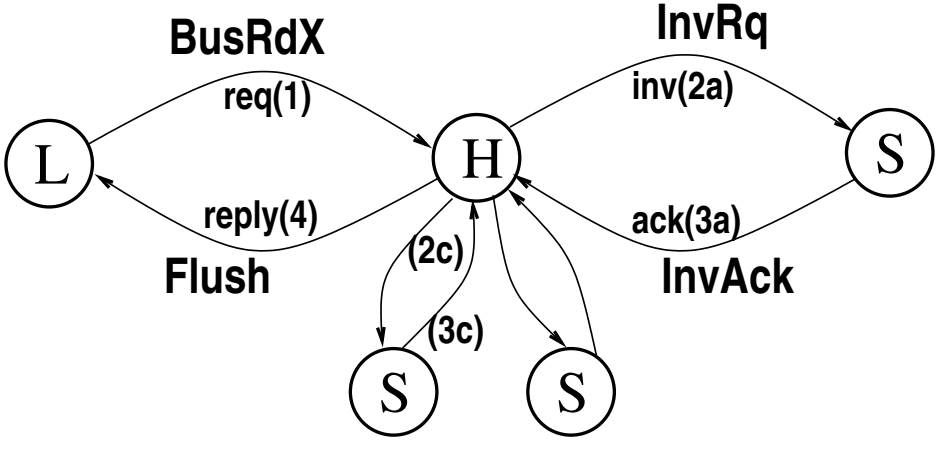




(1) Read Miss on Dirty & Clean Blocks

Example using Fig (2) then (1):

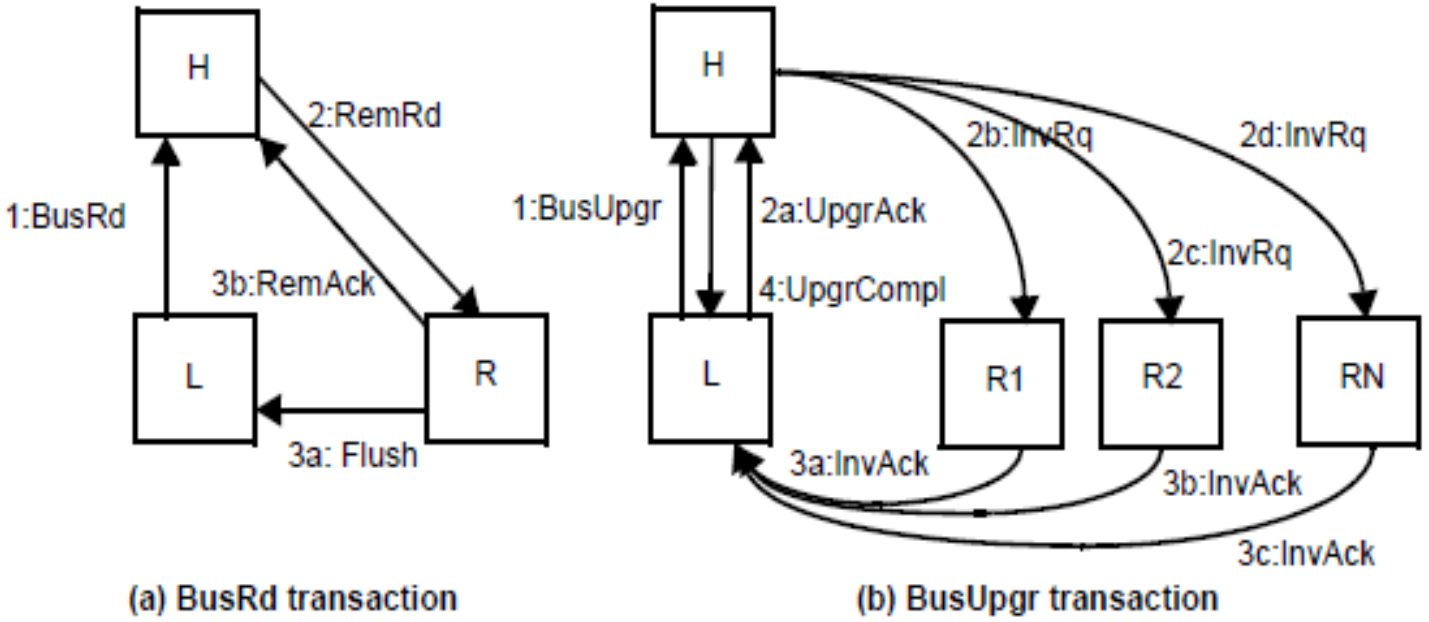
- The latest time the lock bit can be turned off is just before H `Flushes` to L. Not good enough:
- L is now marked dirty. Assume H receives a read request on the same block and forwards it to L, which is now the dirty node.
- If `RemRd` reaches L before the `Flush` from H Then failure because L does not have the copy yet.



(2) Write Miss on Shared Block

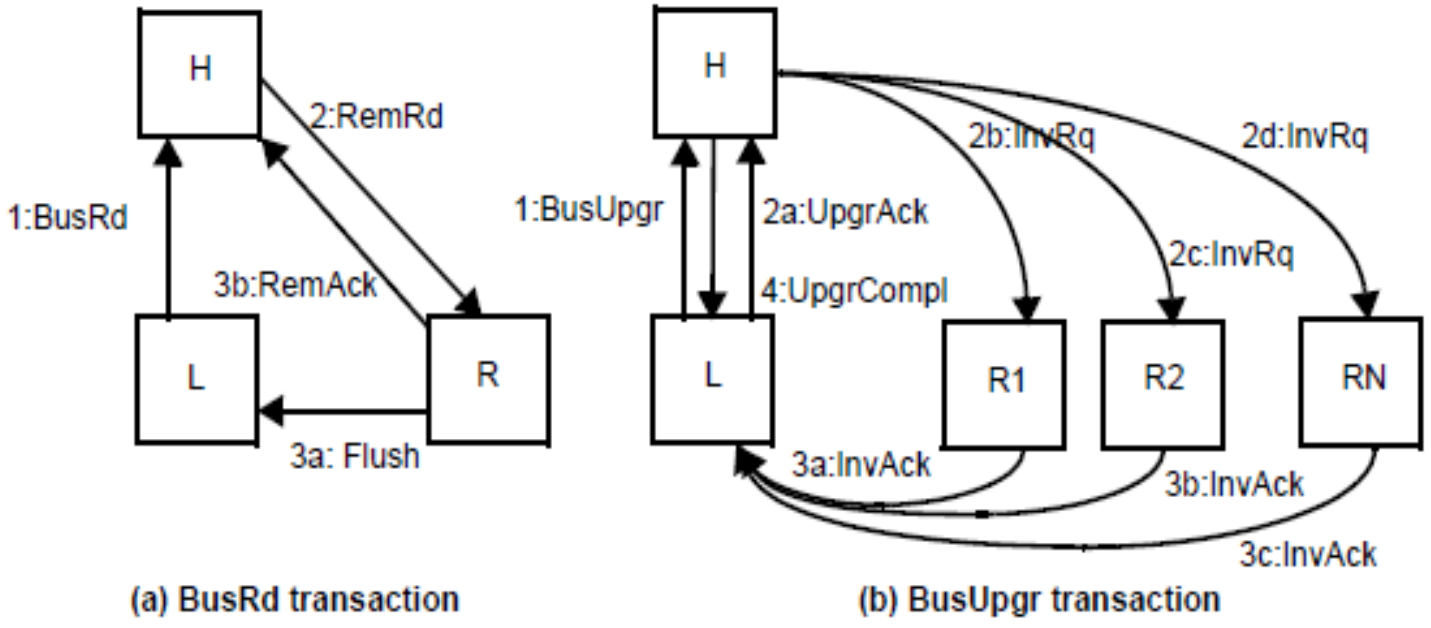
A Solution: L acknowledges to H the end of transaction, i.e., that it has received the `Flush`. After acknowledgement H can safely turn off the Lock Bit!

Standard DASH System: Optimizing Latencies



Baseline Protocol needs at worst 4 hops. Can make it in THREE:

Standard DASH System: Optimizing Latencies



Baseline Protocol needs at worst 4 hops. Can make it in THREE:

1. Request is sent to Home
2. H redirects request to remote nodes.
   In (b) H also sends L the # of remotes via `2a:UpgrAck`
3. Remote responds to L (L now knows when all have responded).
4. L notifies transaction completed, off the critical access path

Memory Requirements of Directory Protocols

Assume N nodes (processors), and M memory blocks per node.

Total Presence-Flag Vector (PFV) Directory Size: $M \times N \times N$
grows linearly with N, hence a serious scalability concern.
(N node directories, each with M entries, and each entry has N bits.)

One Alternative is Limited Pointer Protocol (LPP):

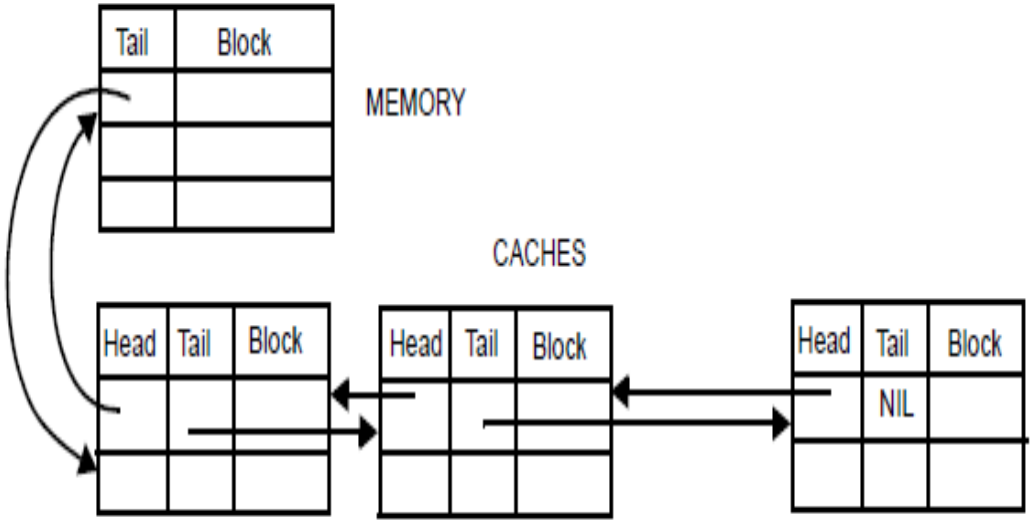
- Each entry maintains i pointers, each represented on $\log N$ bits (instead of N bits in PFV).
- Total LPP Directory Size: $M \times N \times i \times logN$
  grows logarithmically (sublinearly) with N (because i is constant)
- Failback: resort to broadcast when the directory pointers are exhausted.

Other Scalable Protocols

Coarse Vector Schemes: presence flags identify groups rather than individual nodes.

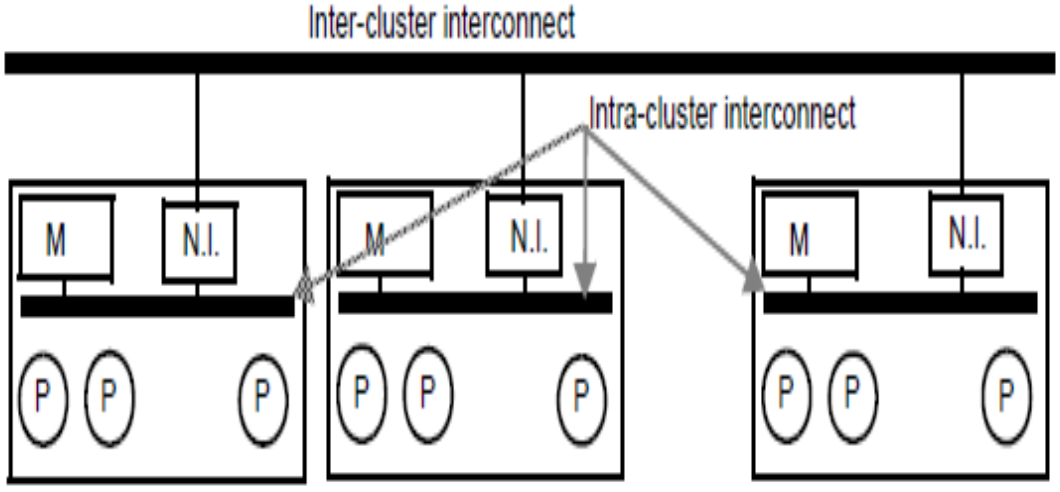
Directory Cache, i.e., at cache rather than main memory level ⇒ memory overhead is proportional to cache size (leverages locality).

Cache-Centric Directories, for example Scalable Coherent Interface:



- Directory Entry: Copies of the same memory block in different (private) caches are linked via a double-linked list (implemented in cache-hardware)
- Directory size proportional with the total (private) cache size, rather than to the main memory size as in PFV.
- Latency of `BusRdX/BusUpgr` larger than PFV because invalidations do not go in parallel (list requires sequential traversal).
- Bandwidth is comparable to PFV.

Hierarchical Systems



Instead of scaling in a flat configuration, one can form clusters in a hierarchical organization. This also reduces memory overhead. Relevant inside, as well as across chip multiprocs!

Coherence Options:

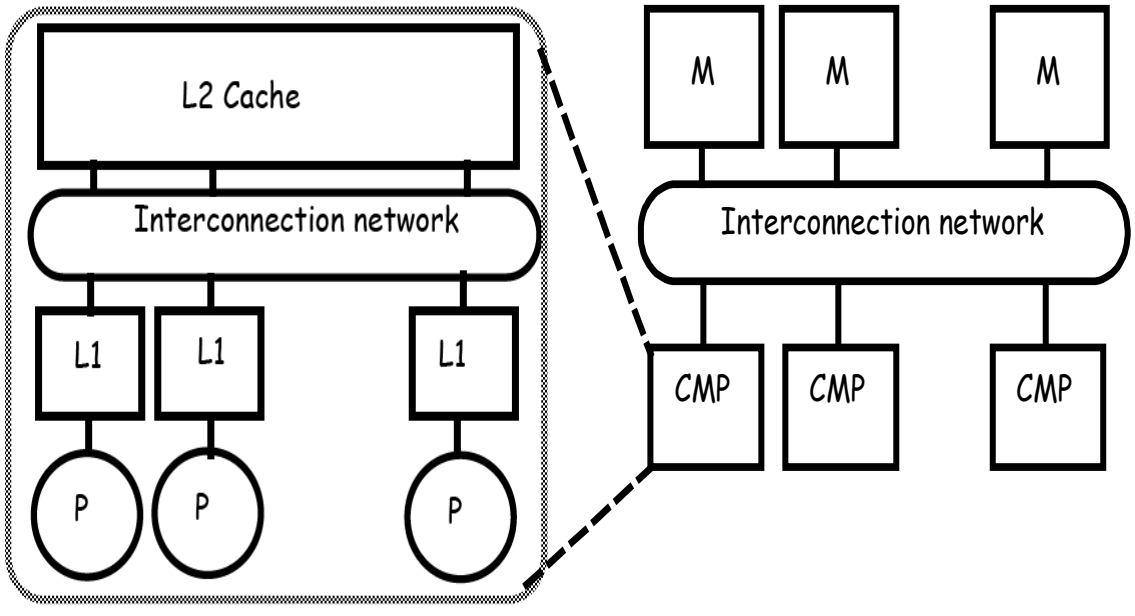
- Intra-Cluster Coherence: snoopy / directory
- Inter-Cluster Coherence: snoopy / directory
- Tradeoffs between memory overhead and performance to maintain coherence.

1. Scalable Shared Memory Systems

2. Interconnection Networks (IN)
   - Design Space, Hardware Components, Communication Models
   - Switching Strategies: Circuit and Packet Switching
   - Latency And Bandwidth Models
   - Indirect Interconnect-Network Topologies
   - Direct Interconnect-Network Topologies
   - Routing Algorithms and Deadlock Avoidance

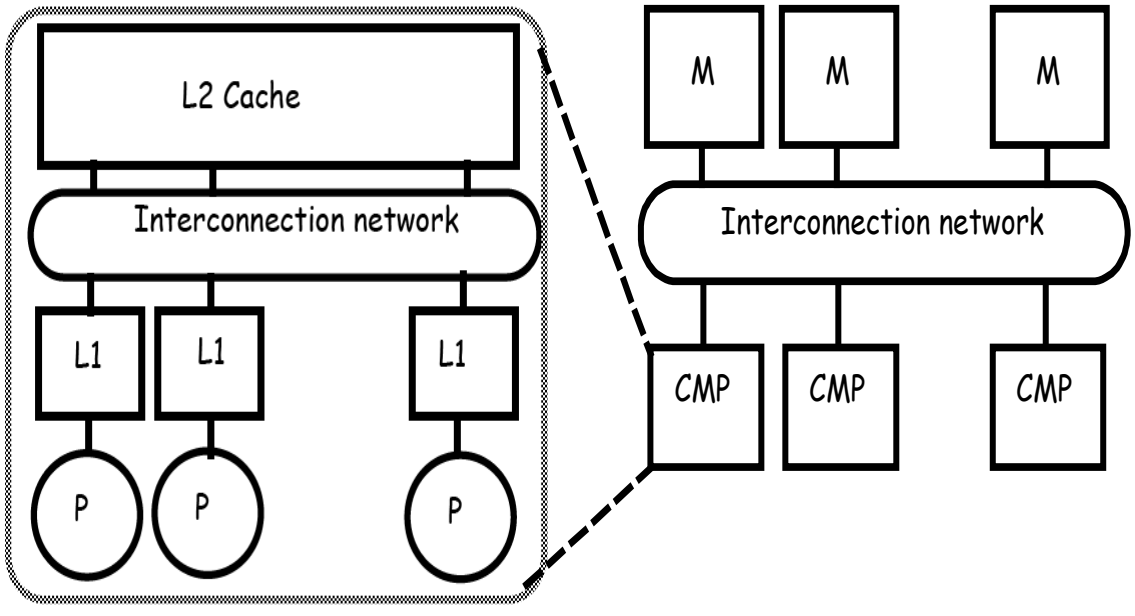
Parallel Computer Systems

Interconnect Networks (IN): Bringing data with low latency and high bandwidth (throughput) is paramount.



Parallel Computer Systems

Interconnect Networks (IN): Bringing data with low latency and high bandwidth (throughput) is paramount.

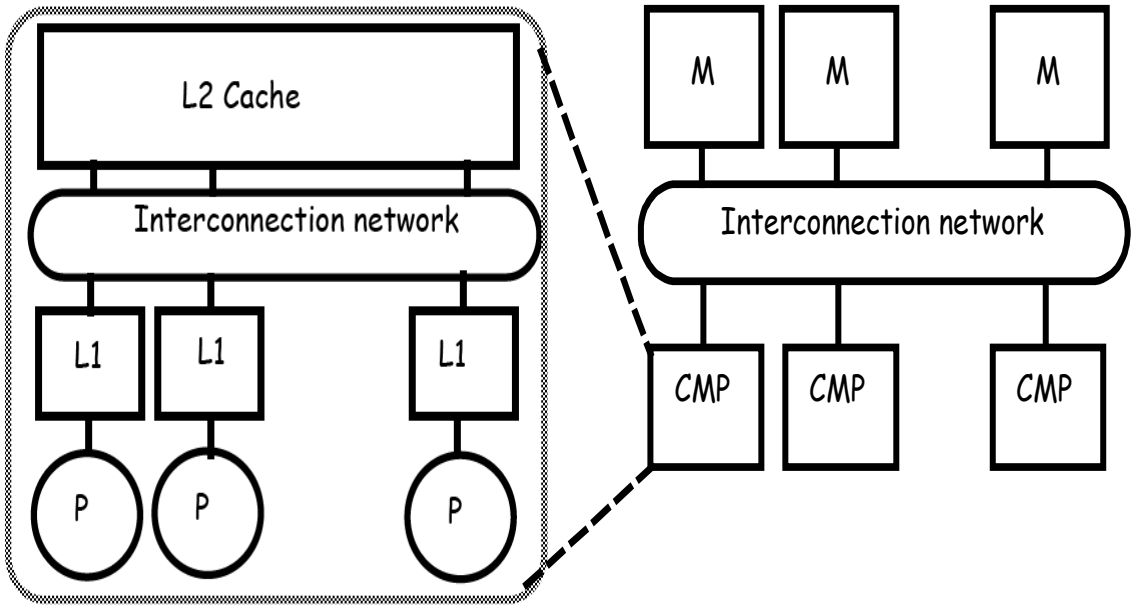


- IN between cores on each chip (OCN/NOC). Shared cache bandwidth increased by splitting cache into into banks, each connected to IN via a port. Important that IN bandwidth matches shared-cache bandwidth
- IN between processor chips SAN. Connects chips to MM (same ↑). LAN/WAN not studied: similar issues, but different tradeoffs, e.g., latency not critical.

Problem: meeting interconnection bandwidth and latency requirement within cost, e.g., silicon area, and power constraints.

Parallel Computer Systems

Interconnect Networks (IN): Bringing data with low latency and high bandwidth (throughput) is paramount.



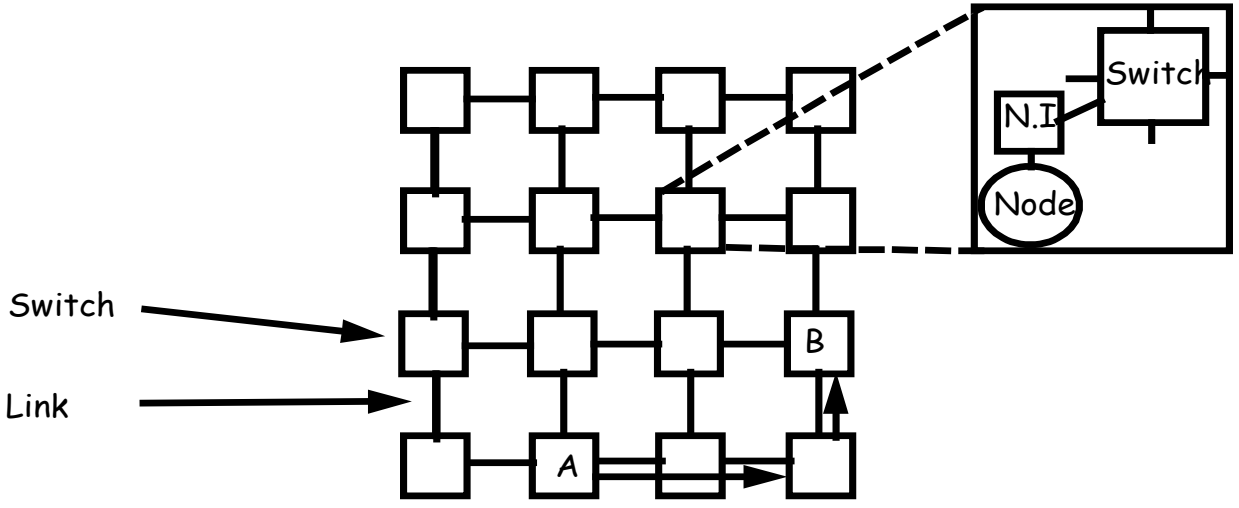
- IN between cores on each chip (OCN/NOC). Shared cache bandwidth increased by splitting cache into into banks, each connected to IN via a port. Important that IN bandwidth matches shared-cache bandwidth
- IN between processor chips SAN. Connects chips to MM (same ↑). LAN/WAN not studied: similar issues, but different tradeoffs, e.g., latency not critical.

Problem: meeting interconnection bandwidth and latency requirement within cost, e.g., silicon area, and power constraints.

For example: fully-connected interconnect best bandwidth & latency ⇒ smaller caches, fewer cores ⇒ not scalable.

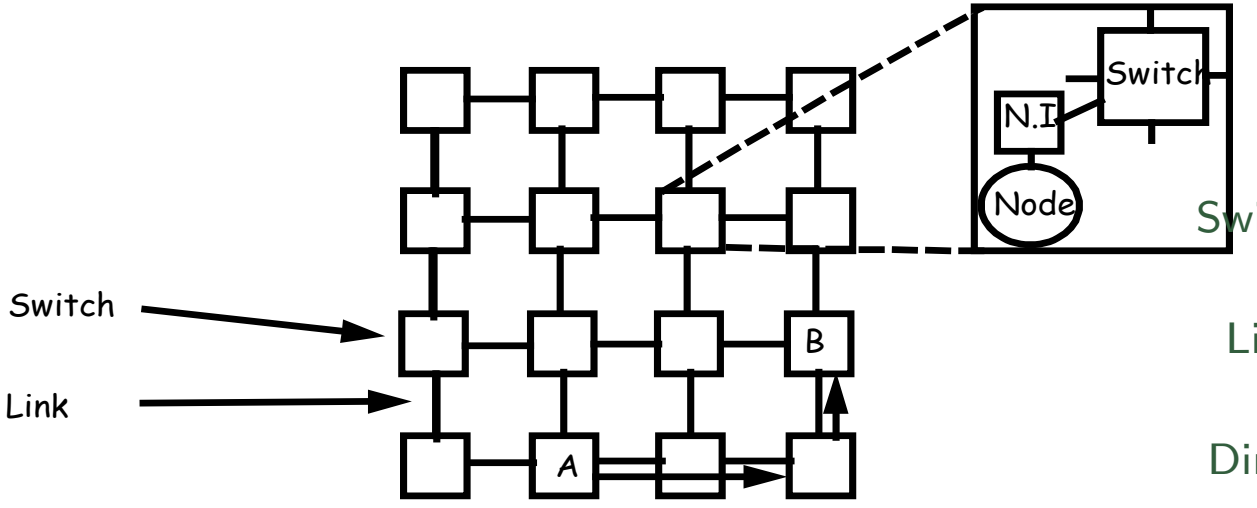
IN Components: Links & Switches (in a MESH)

IN connects nodes: cache/mem modules, CMPs, e.g., 4-by-4 MESH:



IN Components: Links & Switches (in a MESH)

IN connects nodes: cache/mem modules, CMPs, e.g., 4-by-4 MESH:



NI (network interface) connects switches to nodes.

Switch connects input to output ports.

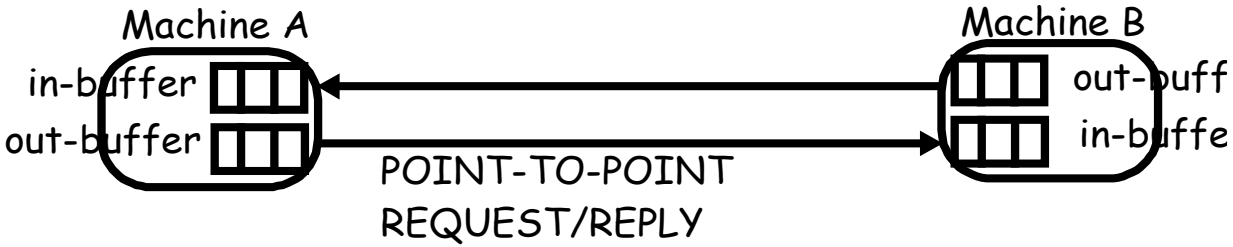
Links wires transferring signals between switches.

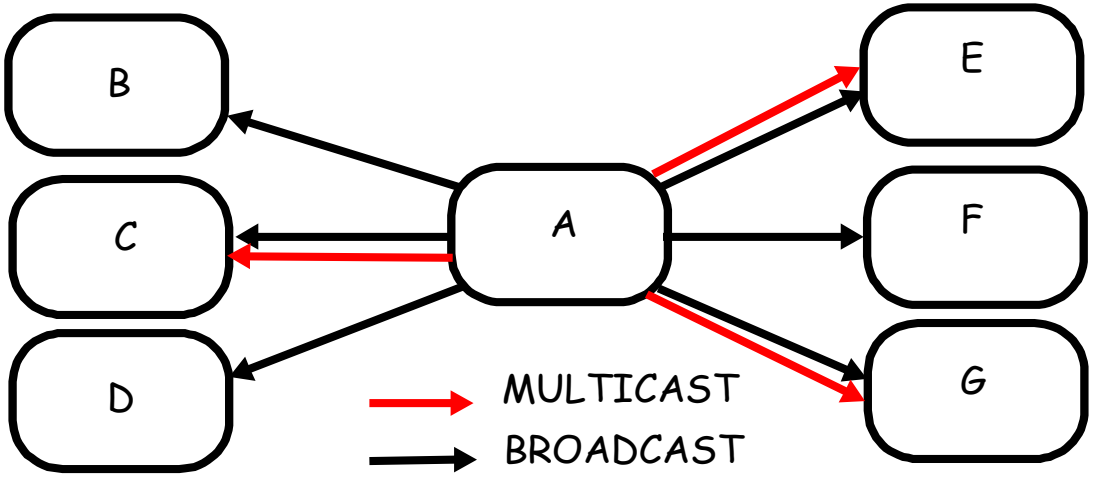
Direct IN (decentralized): to go from `A` to `B` hop from switch to switch.

Link width (`w`): # bits transf. in 1 clock cycle (`t`). Bandwidth=`w/t`. Synch (use same clock) vs Asynch (use handshake) communic.

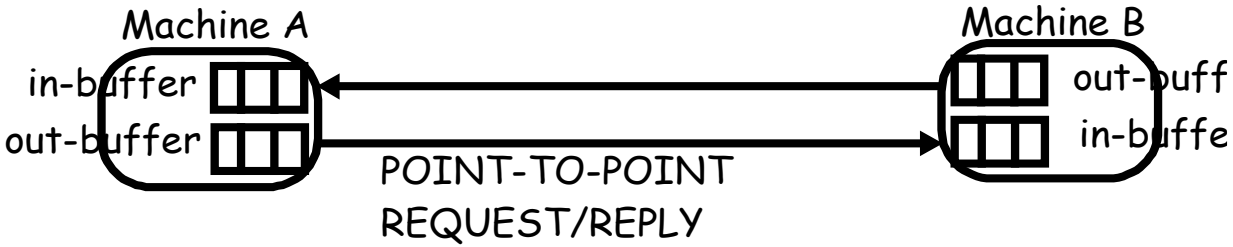
Switch n-by-n switch (input/output ports) ⇒ contention to output port ⇒ buffering at input or/and output port. Input/output ports (and buffers) often connected by a crossbar switch (later).

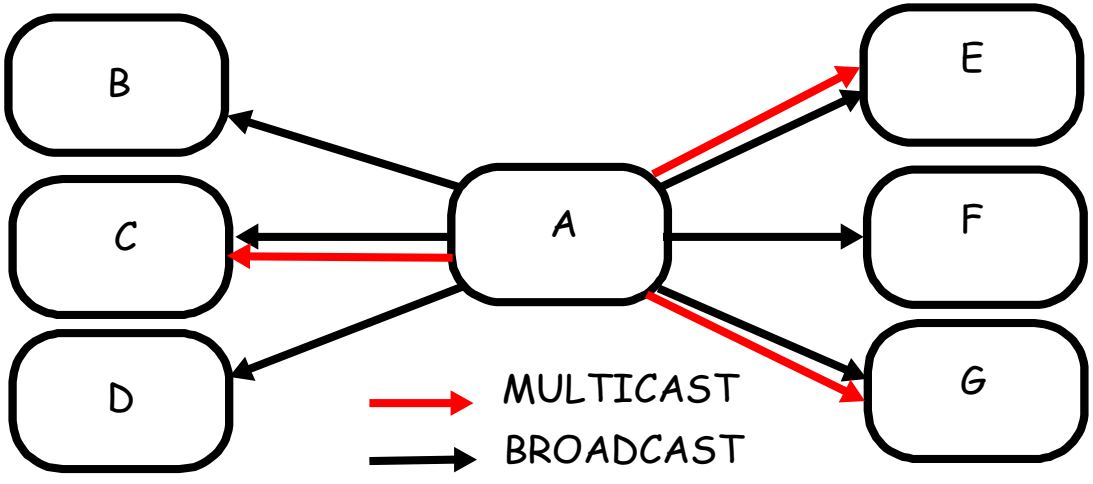
Communication Models





Communication Models





- point-to-point message transfer
- request-reply: request carries ID of sender
- multicast: one to many
- broadcast: one to all.

Messages and Packets

Messages contain the transferred information.

Messages broken into fixed-size packets, which are sent one by one:

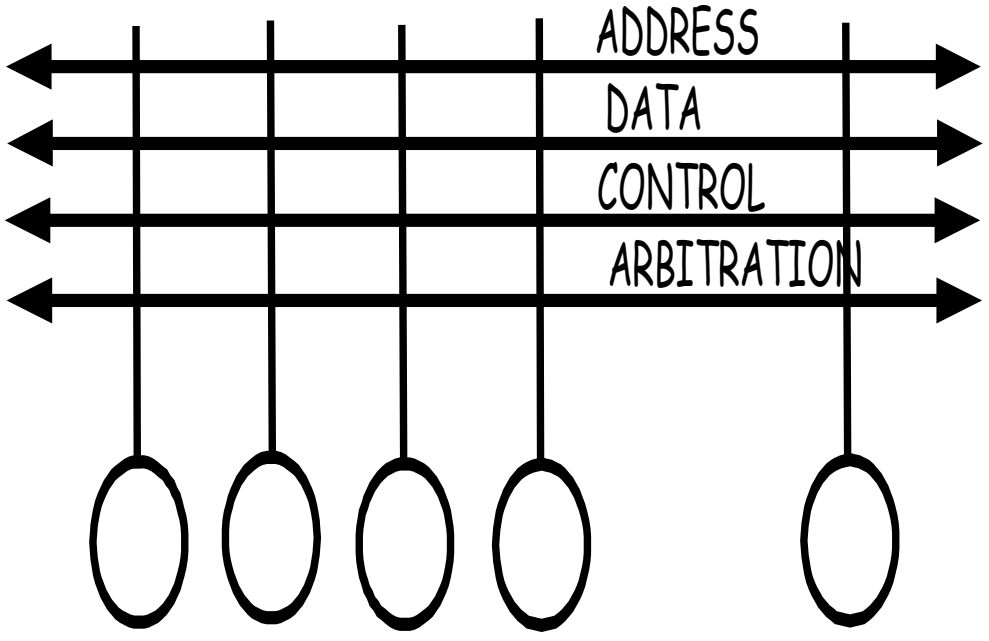
Messages and Packets

Messages contain the transferred information.

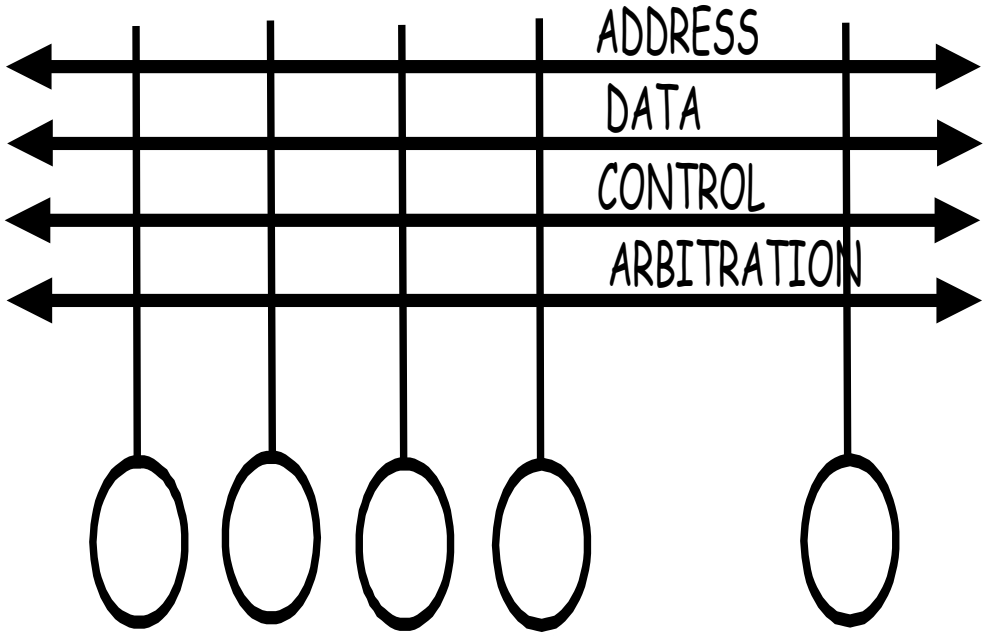
Messages broken into fixed-size packets, which are sent one by one:

- Payload: message data, not relevant to interconnect
- Error Code (ECC): error/detection correction info because packets can be lost or or corrupted, e.g., radiations or switch/link failures.
- Header/trailer: routing info + request/response type.
- Packet Envelope: Header + ECC + Trailer.

Eg Bus: indirect, low cost, shared, low bandwidth



Eg Bus: indirect, low cost, shared, low bandwidth



- Indirect (centralized): box with ports to all nodes exchanging info.
- Shared medium requires arbitration to grant *exclusive access*,
- Broadcast/Broadcall communication,
- Line Multiplexing, e.g., address, data.
- Pipelining, e.g., `arbitration` $\Rightarrow$ `address` $\Rightarrow$ `data`
- Split transaction vs Atomic (circuit-switched) Bus

1. Scalable Shared Memory Systems

2. Interconnection Networks (IN)
   - Design Space, Hardware Components, Communication Models
   - Switching Strategies: Circuit and Packet Switching
   - Latency And Bandwidth Models
   - Indirect Interconnect-Network Topologies
   - Direct Interconnect-Network Topologies
   - Routing Algorithms and Deadlock Avoidance

Switching Strategy

Defines how connections are established in the network.

Circuit (uninterrupted) Switching: establishes a route for the duration of the network service ⇒ avoids routing overhead in each switch ⇒ Trades off lower latency for reduced bandwidth of other procs.

- Example Remote Memory Read On Atomic Bus:
- Example Circuit Switching in MESH:
  Establish path in network, transmit packets/message, release path.
  Good when packets are sent continuously between two nodes.

Packet Switching: multiplex several services by sending packets with addresses ⇒ larger latency, higher bandwidth.

- Example: Remote Memory Access On Split-Transaction Bus.
- Example: Remote memory access on a MESH.

Switching Strategy

Two main Packet Switching Strategies:

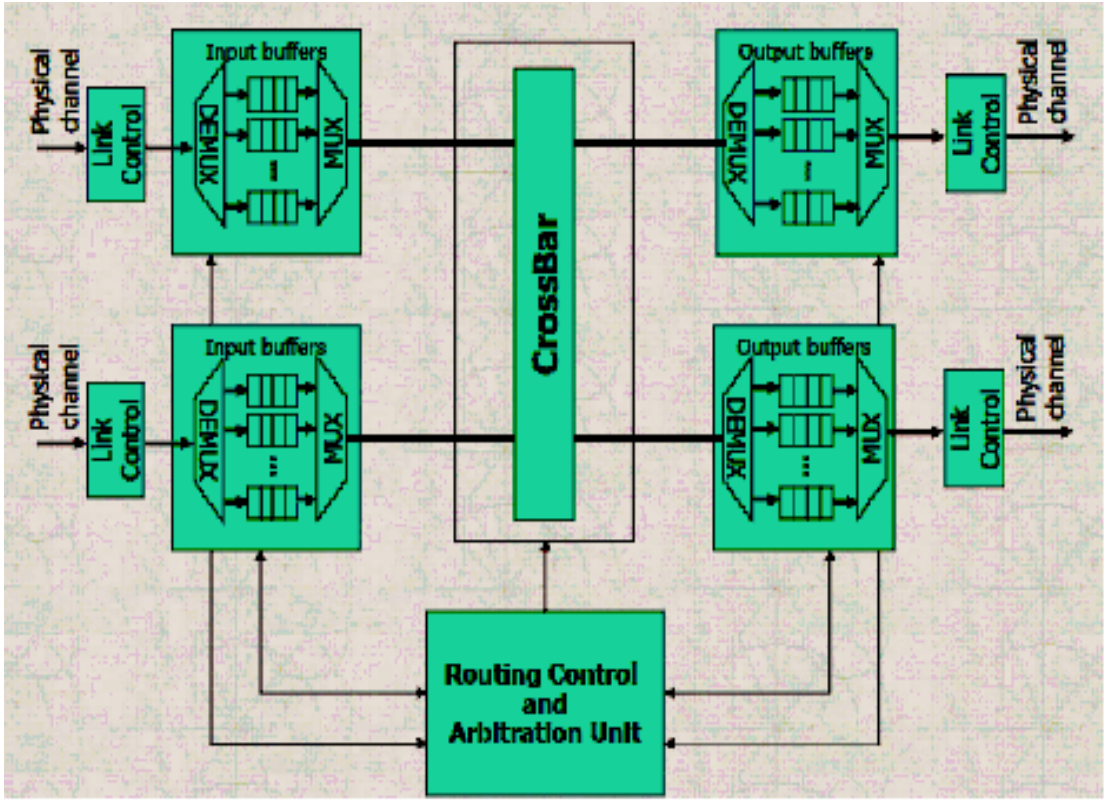
1. Store-and-Forward: packets are stored in buffers at each node, and cannot leave a node until the whole packet has arrived.
2. Cut-Through: packets can move through nodes in pipelined fashion, i.e., entire package moves through several nodes at once.

In practice we must deal with conflicts by stalling packets.

Two implementations of Cut-Through Packet Switching:

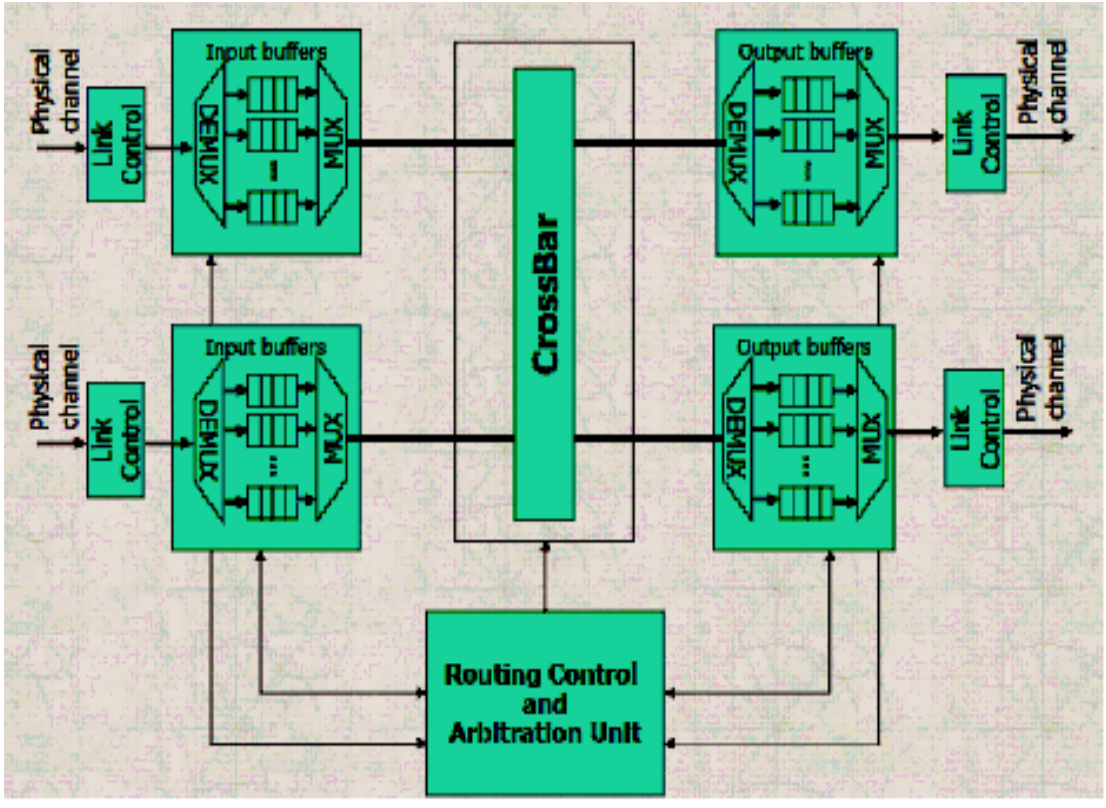
1. Virtual Cut-Through:
   - *Each node has enough buffering for the entire packet*, hence
   - in case of conflict the entire packet is buffered in a node.
   - When traffic is congested resembles store-and-forward.
2. Wormhole: Def. PHIT: link's width (# of bits per cycle)
   - FLIT (FlowControlUnit) consists of several consec PHITS of packet
   - *Each node has enough buffering for one* FLIT, the basic transfer unit subject to control flow. Virtual cut through: FIT is 1 package.
   - Saves buffering space (precious).

Switch Microarchitecture



From Duato and Pinkston in Hennessy and Paterson, 4th edition.

Switch Microarchitecture



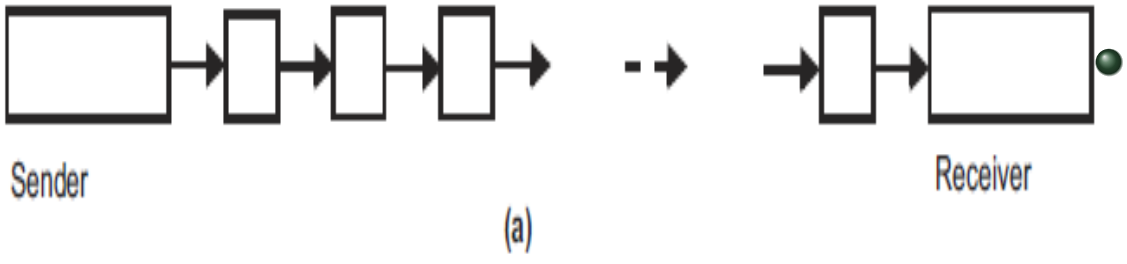
From Duato and Pinkston in Hennessy and Paterson, 4th edition.

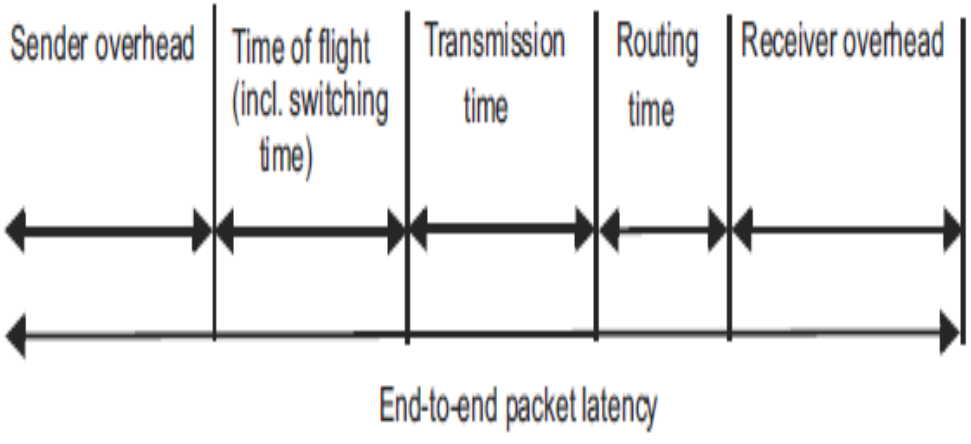
- Physical Channel ≡ LINK, Virtual Channel ≡ Buffers + LINK
- LINK is multiplexed among FLITs
- Input Buffering suffers from Head-Of-Line Blocking.

1. Scalable Shared Memory Systems

2. Interconnection Networks (IN)
   - Design Space, Hardware Components, Communication Models
   - Switching Strategies: Circuit and Packet Switching
   - Latency And Bandwidth Models
   - Indirect Interconnect-Network Topologies
   - Direct Interconnect-Network Topologies
   - Routing Algorithms and Deadlock Avoidance

Latency Models: End-to-End Packet Latency





- SenderOVerhead: creating the envelope & moving packet to NI.
- TimeOfFlight: time to send a bit from source to dest, when route is established and without conflicts (includes SwitchingTime).
- TransmissionTime: time to transfer a packet from source to dest, once the first bit has arrived @dest,e.g.,`PacketSize/PHITsize`.
- RoutingTime: time to set up switches (globally/locally).
- SwitchingTime: depends on switching strategy.
- ReceiverOVerhead: time to strip envelope and move packet in.

```
End-to-End Packet Latency = SenderOV + TimeOfFlight +
                            TransmissionTime +
                            RoutingTime + ReceiverOV
```

Latency (Continuation)

Packets of a Message Can Be Pipelined:

- Transfer pipeline has three stages:
  SendrOV→Transmission→ReceiverOV
- TotalMsgTime = Time for First Packet +
  (N-1) / Pipeline Throughput

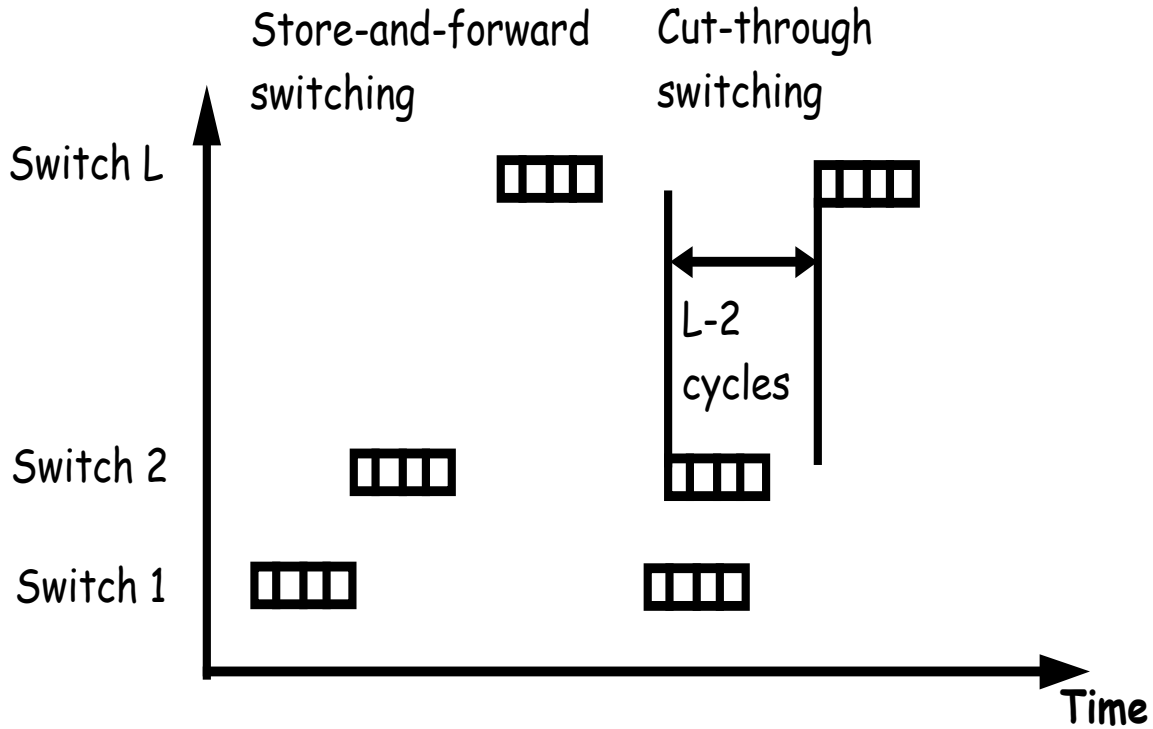
End-to-End Msg Latency=SenderOV + TimeOfFlight +
TransmissionTime + RoutingTime +
(N-1) × Max(SenderOV,
TransmissionTime,
ReceiverOV)

Latency for Switching Strategies

Circuit Switching:

- Route is set up first, `RoutingTime = L×R + TimeOfFlight`
- `R`: to set each switch, `L`: # of switches,
  `TimeOfFlight`: because I need to inform the node back.

Packets Switching: rout set up as package moves through switches



Latency for Switching Strategies (Cont.)

`PacketLatency = SenderOV + ToF (incl. SwitchingTime) + TransmissionTime + RoutingTime + ReceiverOV`

`R`: routing time per switch, `N`: # of phits, `L`: # of switches, `ToF`: time of flight

- Packet Latency for Circuit Switching
  - `SenderOV + 2×ToF + N + L×R + ReceiverOV`
  - `ToF = L` because there are L switches and 1 PHIT to switch.
- Packet Latency for Store and Forward
  - `SenderOV + ToF + N + L×R + ReceiverOV`
  - `ToF = L×N` because switching involves the whole packet in PHITs.
- Packet Latency for Cut Through
  - `SenderOV + ToF + N + L×R + ReceiverOV`
  - `ToF = L` as in circuit switching when traffic not congested.
  - `ToF = L×N` as in store and forward when traffic congested.
- Virtual Cut Through & Wormhole Cut Switching similar to circuit switching when traffic not congested, but offers better bandwidth. Differences reside in how they handle conflicts.

Bandwidth

- Bottlenecks Increase Latency, but Transfers are Pipelined:
  `Effective Bandwidth = PacketSize / MAX(SenderOV,ReceiverOV,TransmissionTime)`
- Network contention affects latency and effective bandwidth (not acounted in above formula).

1. Scalable Shared Memory Systems

2. Interconnection Networks (IN)
   - Design Space, Hardware Components, Communication Models
   - Switching Strategies: Circuit and Packet Switching
   - Latency And Bandwidth Models
   - Indirect Interconnect-Network Topologies
   - Direct Interconnect-Network Topologies
   - Routing Algorithms and Deadlock Avoidance

Latency and Bandwidth Models (for this section)

Measures of Latency:

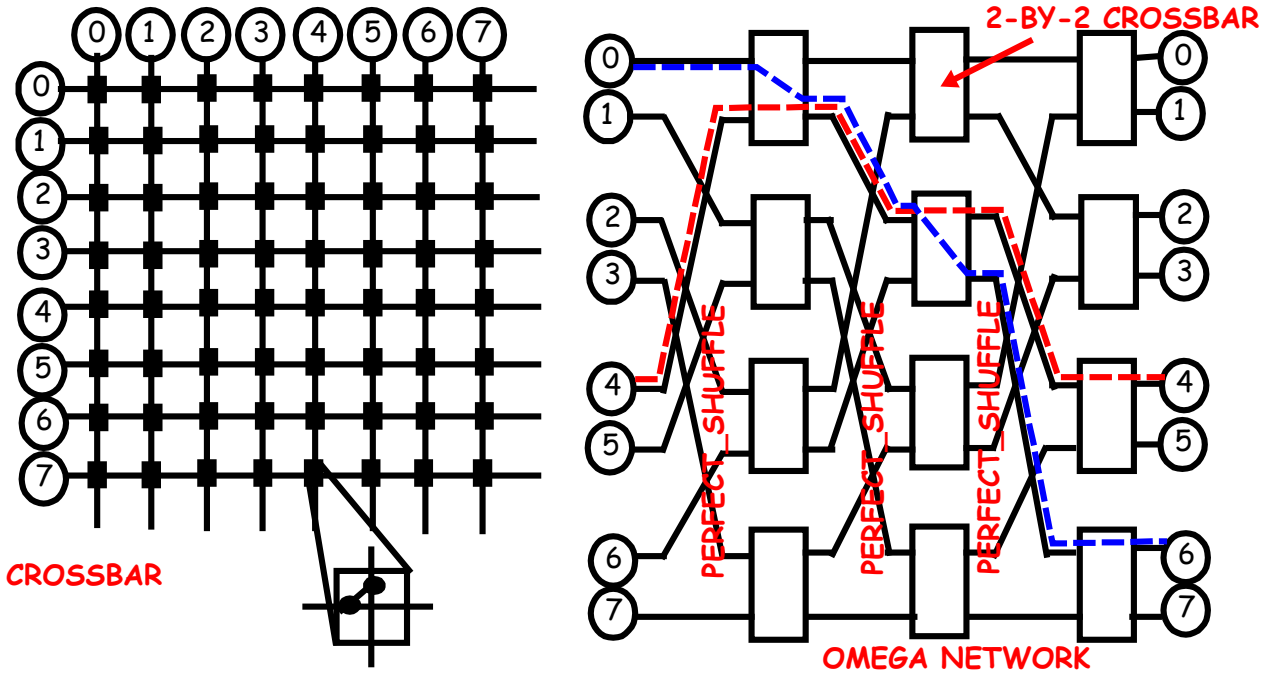
- Routing Distance: number of links traversed by a packet
- Average Routing Distance: average over all node pairs.
- Network Diameter: longest routing distance over all node pairs.

Measures of Bandwidth:

- Bisection Width:
  - network seen as a graph: vertices are switches and edges are links,
  - bisection is a cut through a minimum set of edges such that the cut divides the network graph into two isomorphic subgraphs.
  - Measures bandwidth when all nodes in one subgraph communicate only with nodes in the other subgraph.
- Aggregate Bandwidth: is the bandwidth across all links divided by the number of nodes.

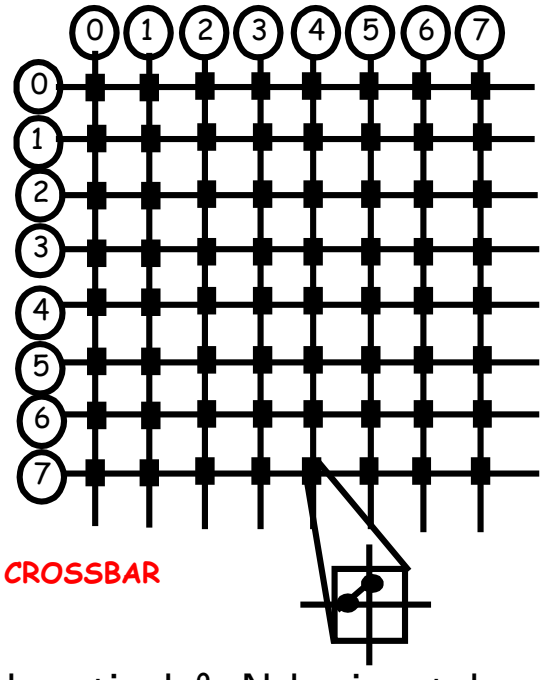
Crossbar Switch & Multistage IN (MIN)

Indirect Networks: centralized, typically connect different types of nodes, e.g., private with shared caches, cores to shared caches/mem.

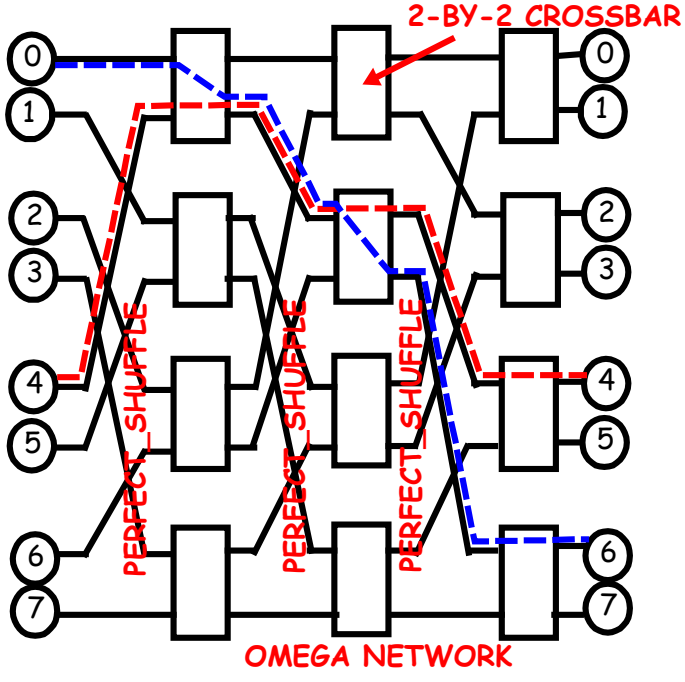


Crossbar Switch & Multistage IN (MIN)

Indirect Networks: centralized, typically connect different types of nodes, e.g., private with shared caches, cores to shared caches/mem.



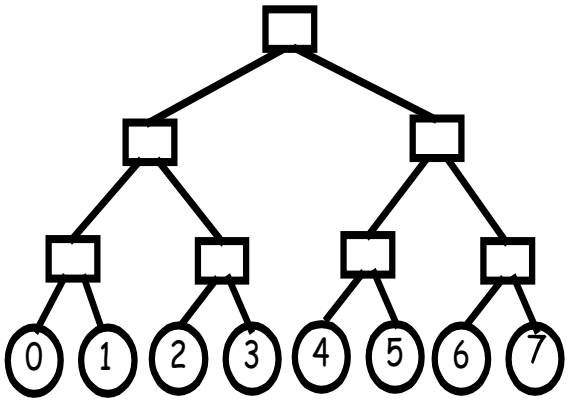
N×N Crossbar: N vertical & N horizontal buses. Routing Mechanism controls all N^2 crosspoints, each of them may connect two buses ⇒ unicast, multicast, and broadcast. Bandwidth scales linearly with N, but limited by bus length/load and cost/resource.



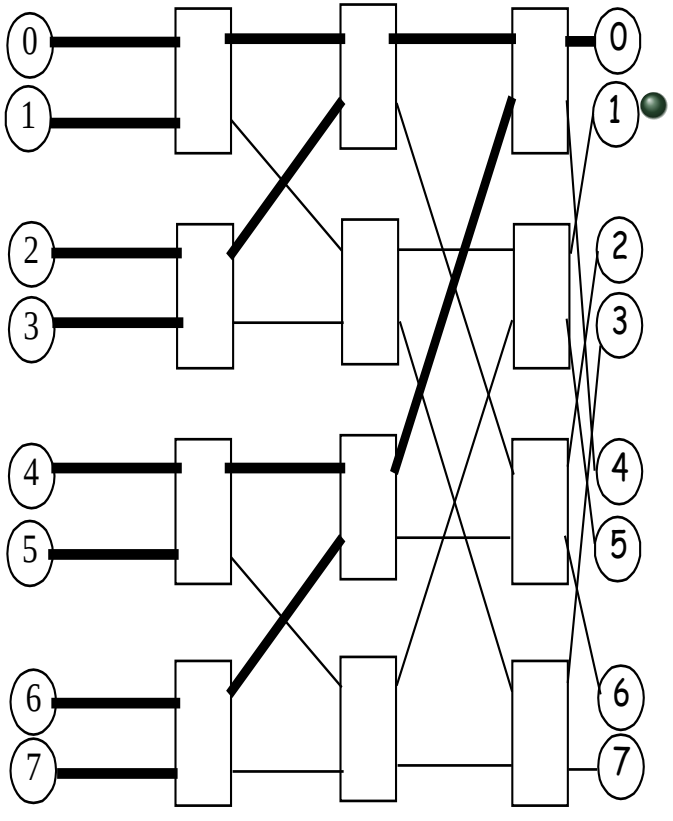
Use k×k crossbars as building blocks.
Perfect shuffle exchange (deck of cards)!
N×N nodes requires $(N \times \log_k N)/k$ crossbars, but a packet passes through $\log_k N$ crossbars!
MIN's cost scales better, but prone to contention.

Tree & Butterfly IN

Topology impacts the routes that share resources & cause conflicts.



BINARY TREE

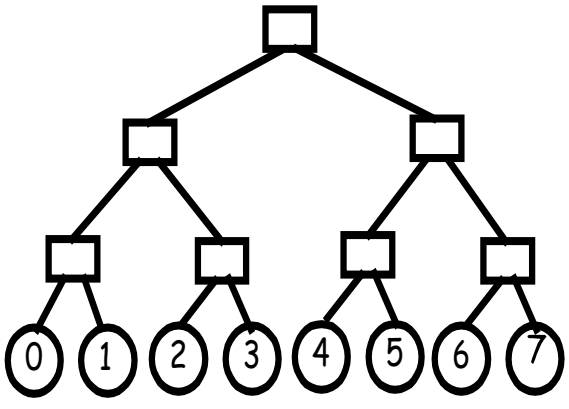


BUTTERFLY: EMBEDDED TREES

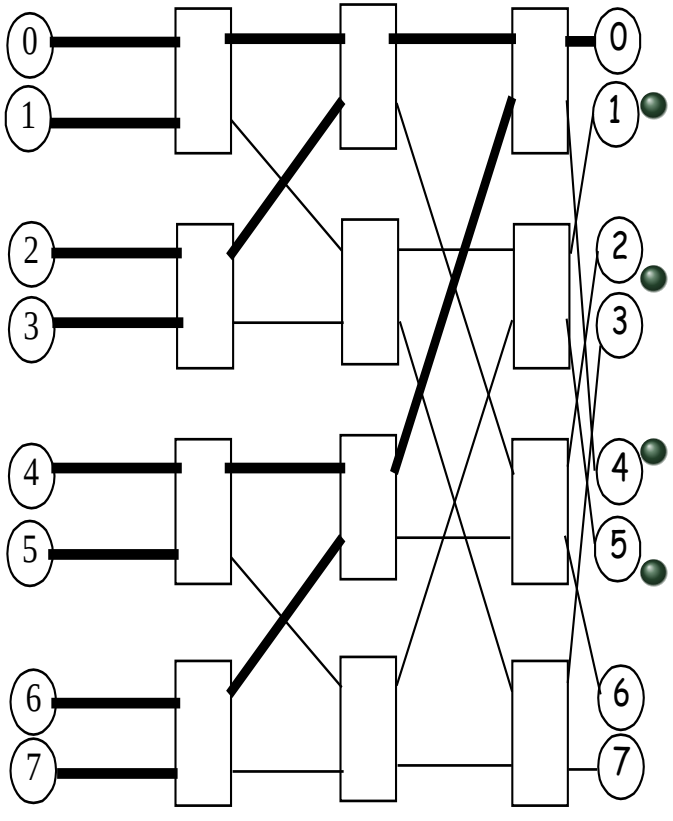
In the butterfly network: are there any trees that do not share resources?

Tree & Butterfly IN

Topology impacts the routes that share resources & cause conflicts.



BINARY TREE



BUTTERFLY: EMBEDDED TREES

In the butterfly network: are there any trees that do not share resources?

k-ary Tree connecting N nodes has depth $\log_k N$ & exploits locality but bisection width is 1.

Fat tree: the closer the link to root, the higher its bandwidth.

Butterfly Networks: is a MIN in which trees are embedded. There are as many tree roots as # of nodes N, but different trees may share resources.

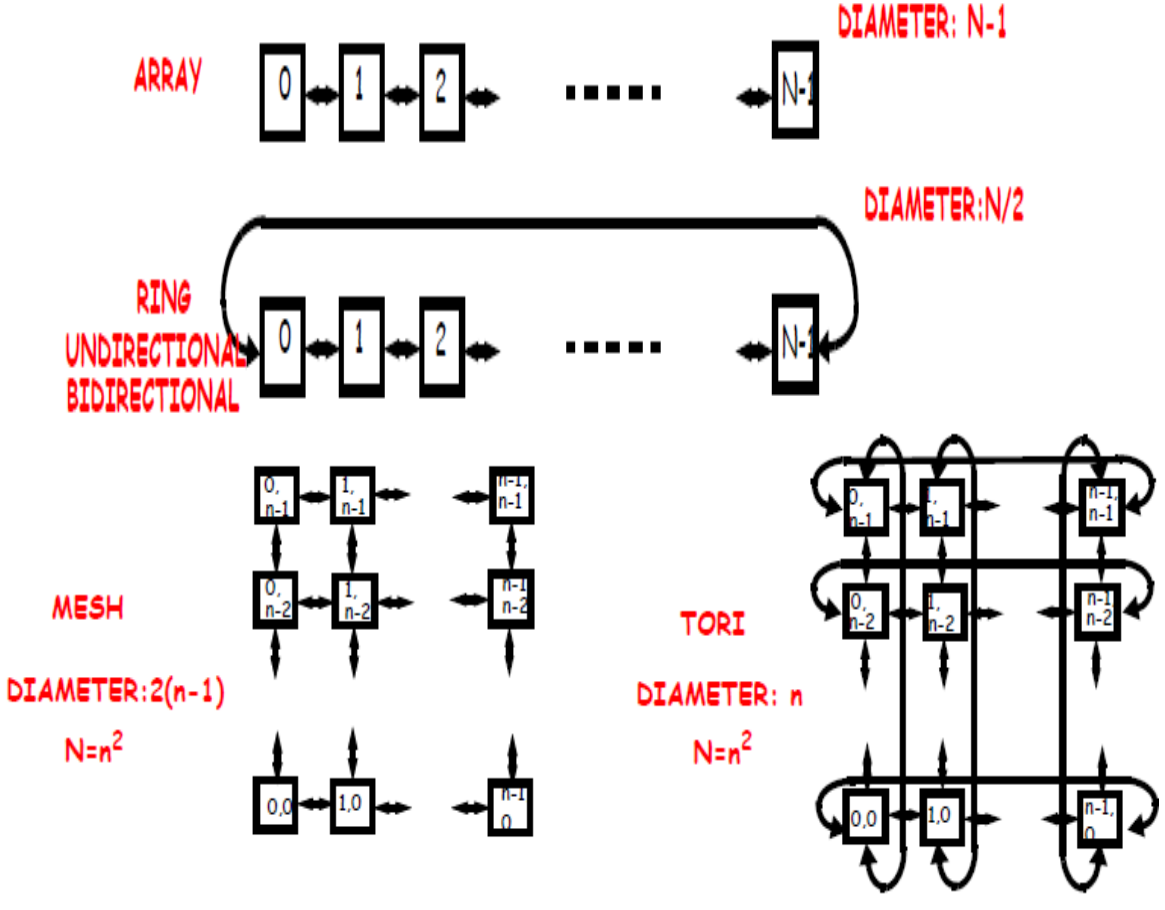
1. Scalable Shared Memory Systems

2. Interconnection Networks (IN)
   - Design Space, Hardware Components, Communication Models
   - Switching Strategies: Circuit and Packet Switching
   - Latency And Bandwidth Models
   - Indirect Interconnect-Network Topologies
   - Direct Interconnect-Network Topologies
   - Routing Algorithms and Deadlock Avoidance

Linear Array, Ring, Mesh and Tori

Direct (Decentralized) IN: nodes, typically of same type, are directly connected & integrated tightly with switches ⇒ exploit locality.

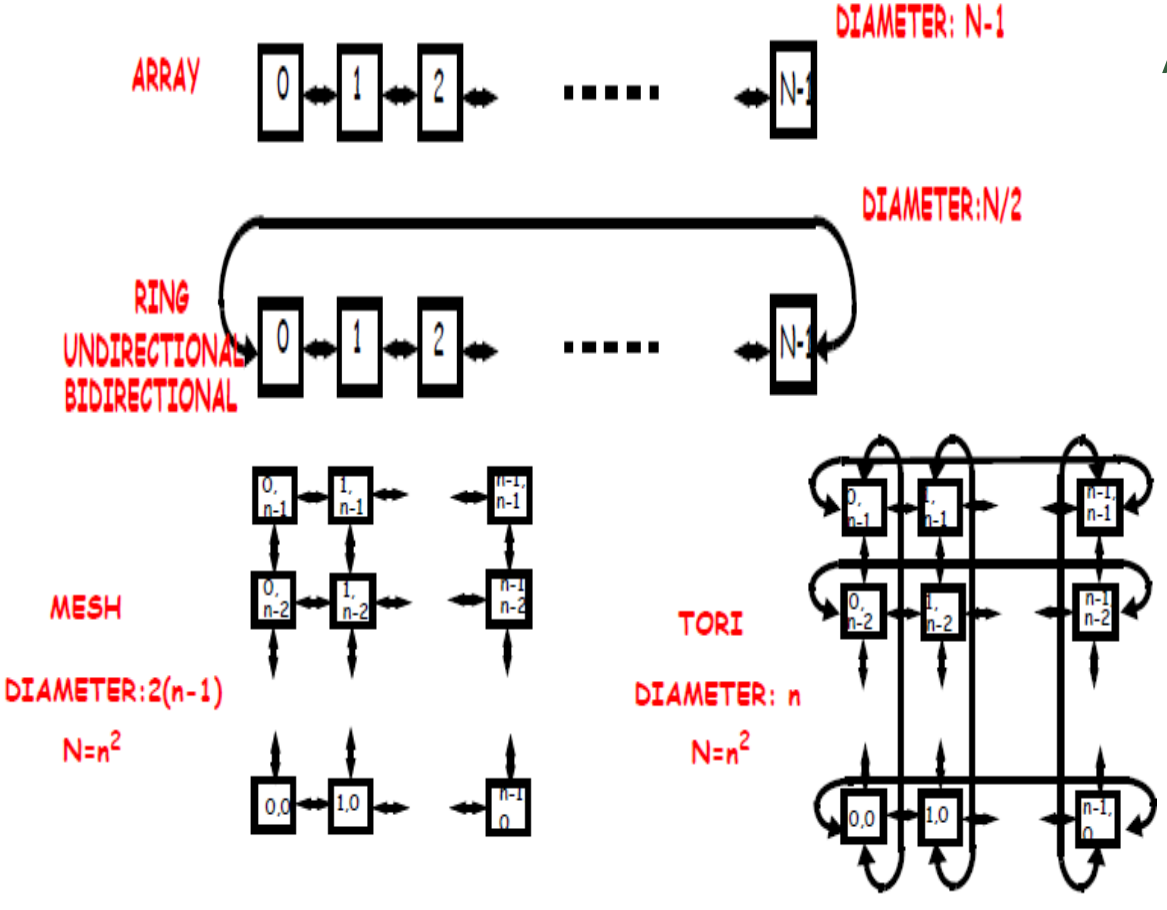
What is the diameter & bisection of a N=64-nodes Tori?



Linear Array, Ring, Mesh and Tori

Direct (Decentralized) IN: nodes, typically of same type, are directly connected & integrated tightly with switches ⇒ exploit locality.

What is the diameter & bisection of a N=64-nodes Tori?



Array diameter: $N-1$,
bisection: 1

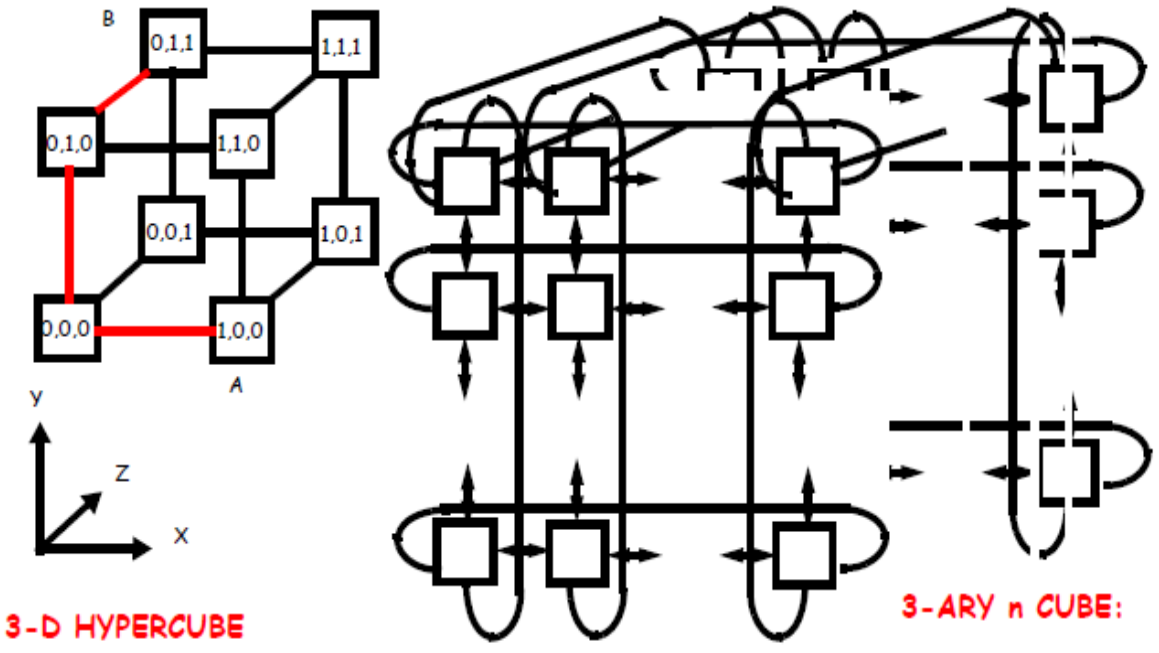
Ring diameter: $N/2$,
bisection: 2.

Mesh diameter: $2(n-1)$
bisection: n,
where $N = n^2$.

Tori diameter: n,
bisection: $2 \times n$,
where $N = n^2$.

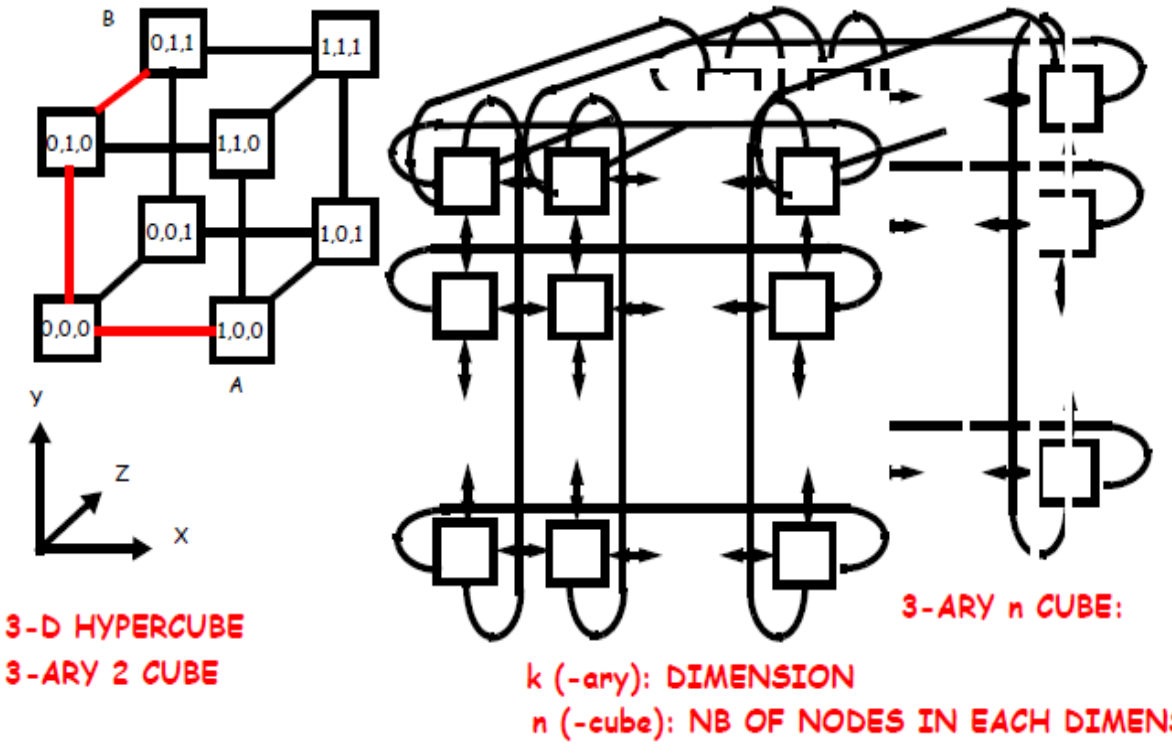
Hypercubes and k-ary n-cubes

Direct (Decentralized) IN: nodes, typically of same type, are directly connected & integrated tightly with switches ⇒ exploit locality.



Hypercubes and k-ary n-cubes

Direct (Decentralized) IN: nodes, typically of same type, are directly connected & integrated tightly with switches ⇒ exploit locality.



- Hypercube: an n-dim cube with 2 nodes in each dim. Connects $N=2^n$ nodes, with diameter: $\log N$ and bisection: $N/2$, but the cost (switch degree) of a node grows linear with n and hypercube difficult to lay out such that all links are equal!
- k-ary n-Cube, e.g., hypercube: n-ary 2-cube, n-by-n Tori: 2-ary n-cube. 3-ary n-cube: built from n Tori connected on top of each over.

Comparison Between Topologies

Interconnect Network	Switch Degree	Network Diameter	Bisection Width	Network Size
Crossbar Switch	N	1	N	N
Butterfly from k-by-k switches	k	$\log_k N$	N/2	N
k-ary Tree	k+1	$2\log_k N$	1	N
Linear Array	2	N-1	1	N
Ring	2	N/2	2	N
n-by-n mesh	4	2(n-1)	n	$N=n^2$
n-by-n tori	4	n	2n	$N=n^2$
k-dim hypercube	k	k	2^{k-1}	$N=2^k$
k-ary n-cube	2k	nk/2	$2n^{k-1}$	$N=n^k$

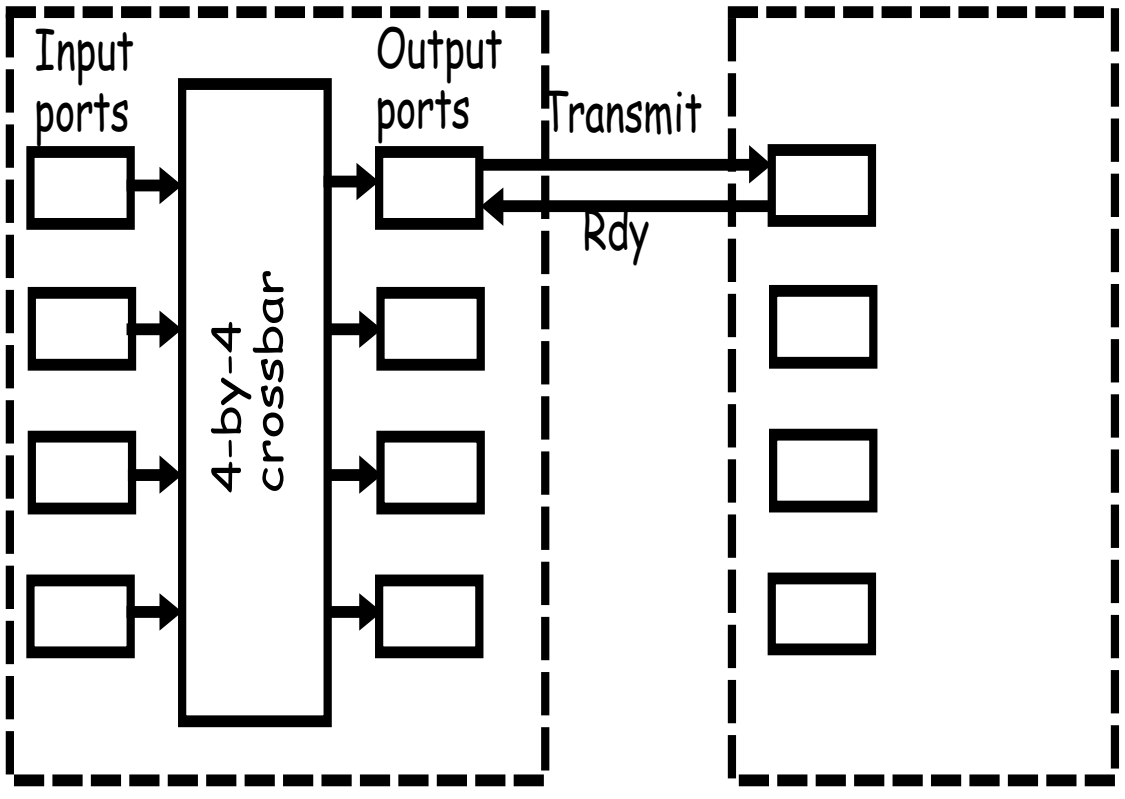
- Switch Degree: # of ports for each switch (switch complexity), approximates cost.
- Network Diameter: worst case routing distance between 2 nodes.
- Bisection Width: # of links cut in a bisection (measures potential bottlenecks).
- Network Size: # of nodes (We used # for number of).

1. Scalable Shared Memory Systems

2. Interconnection Networks (IN)
   - Design Space, Hardware Components, Communication Models
   - Switching Strategies: Circuit and Packet Switching
   - Latency And Bandwidth Models
   - Indirect Interconnect-Network Topologies
   - Direct Interconnect-Network Topologies
   - Routing Algorithms and Deadlock Avoidance

Flow Control

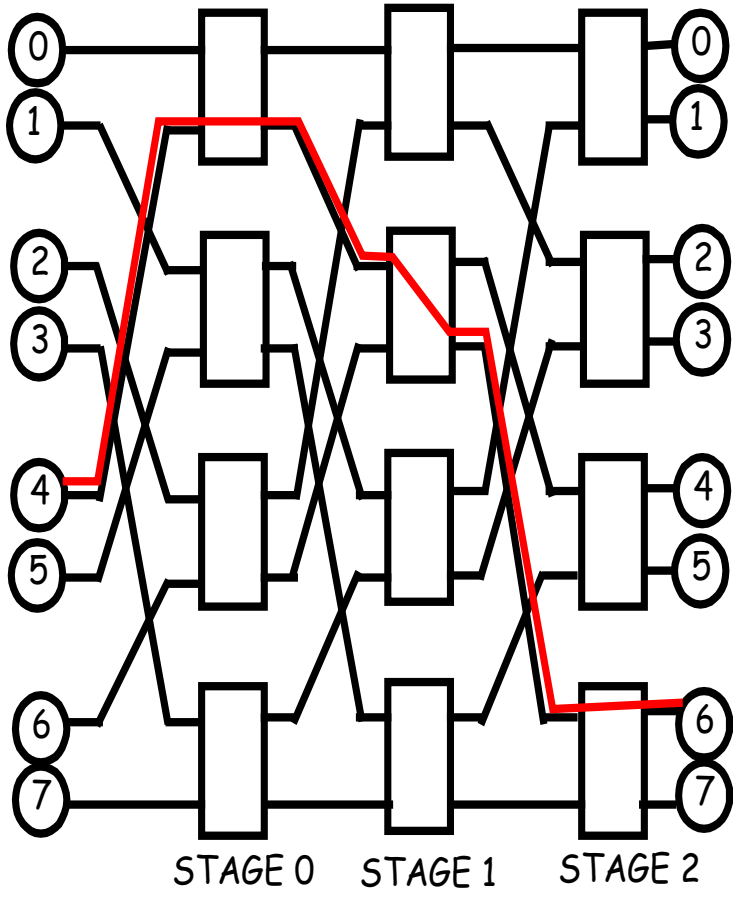
Refers to mechanisms to handle conflicts in switch-based networks.



- Buffers at input and output ports.
  Virtual Cut Through: buffer for entire packet.
  Wormhole: buffer for integral number of FLITs.
- Link-Level Flow Control, e.g., by handshake signal: Rdy indicates whether FLITs can be transmitted to destination.
- Hot Spot Contention and Tree Saturation.

Routing Algorithm for Omega (Shuffle) Network

Routing Algorithms: use source/destination address, are very simple
⇒ keep latency small. (Table driven approaches are prohibitive.)



USE THE DESTINATION ADDRESS
IN THIS CASE, 3 BITS <d2,d1,d0>

USE ith BIT OF THE DESTINATION (di)
TO SELECT UPPER OR LOWER OUTPUT
PORT FOR STAGE n-1-i

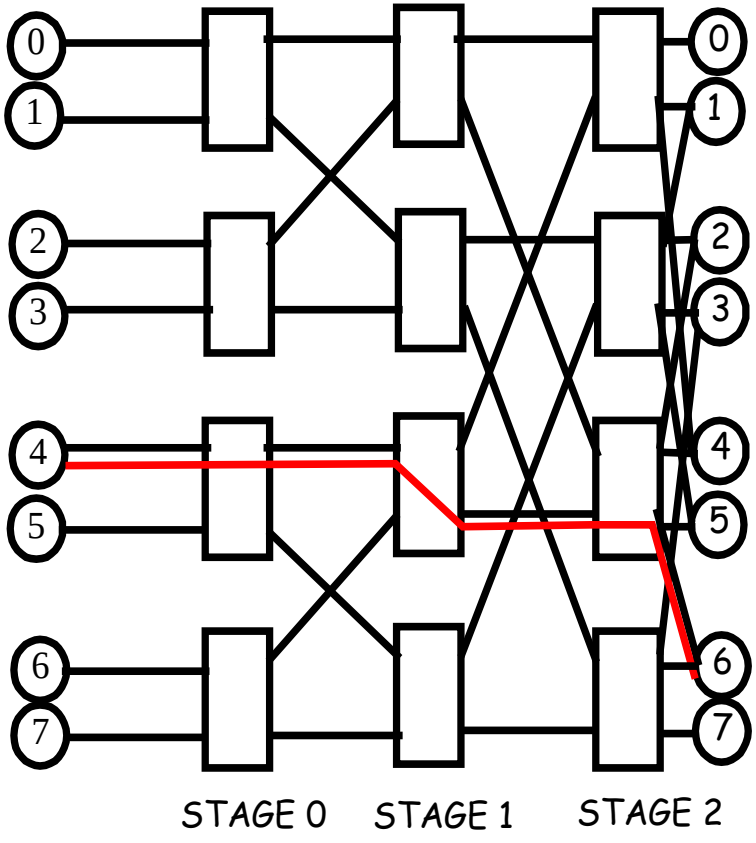
- 0 => UP
- 1 => DOWN

EXAMPLE: ROUTE FROM 4 TO 6

- DESTINATION ADDRESS IS 110
  - DOWN IN STAGE 0
  - DOWN IN STAGE 1
  - UP IN STAGE 2

Routing Algorithm for Butterfly Network

Routing Algorithms: use source/destination address, are very simple
$\Rightarrow$ keep latency small. (Table driven approaches are prohibitive.)



USE RELATIVE ADDRESS

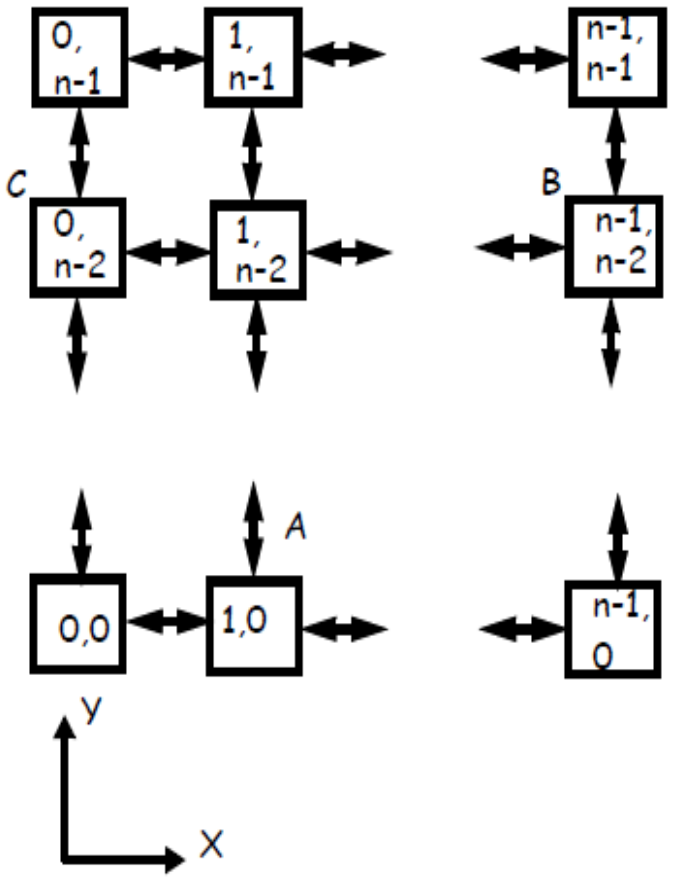
- BITWISE EXCLUSIVE OR OF SOURCE AND DESTINATION ADDRESSES TO FORM THE ROUTING ADDRESS
- IF BIT i IS ZERO, ROUTE STRAIGHT
- IF BIT i IS ONE, ROUTE ACROSS

EXAMPLE: ROUTE FROM 4 TO 6

- SOURCE: 100
- DESTINATION: 110
- EX-OR: 010

Routing Algorithm for Mesh Network

Dimension-order routing (deterministic).



NUMBER NODES SO THAT LOWER LEFT CORNER IS (0,0) AND UPPER RIGHT CORNER IS (n-1,n-1)

USE RELATIVE ADDRESS:

- $(X,Y)=(X_B-X_A, Y_B-Y_A)$
- ROUTE FIRST HORIZONTALLY
  - DECREMENT X
- WHEN X=0, ROUTE VERTICALLY
  - DECREMENT Y
  - UNTIL Y=0

EXAMPLE: MOVE PACKET FROM A TO B

- RELATIVE ADDRESS IS (X,Y)=(n-2,n-2)
- FIRST MOVE PACKET HORIZONTALLY
  - DECREMENT X BY 1 (RIGHT MOVE)
- WHEN X=0, MOVE PACKET VERTICALLY
  - DECREMENT Y BY 1 (UP MOVE)

TRY B TO C

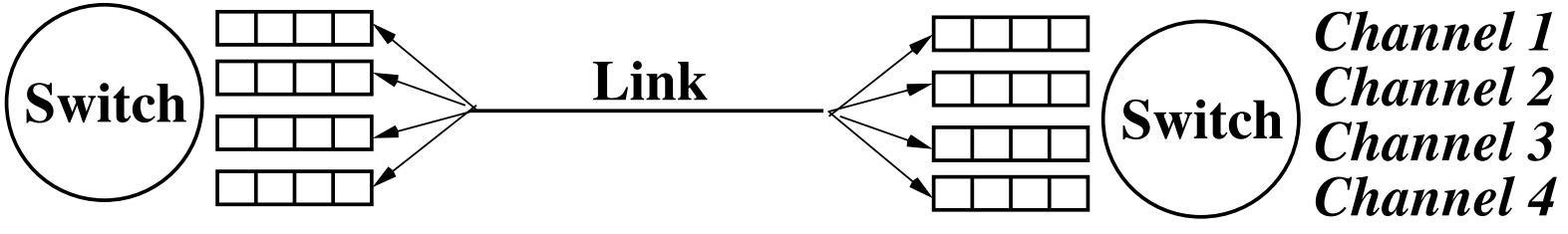
Hypercube: XOR (bit-wise exclusive OR) the source and destination address, then route in the order of non-zero dimensions.

Necessary Conditions for Deadlock

Assume a set of Agents accessing a Set of Resources.

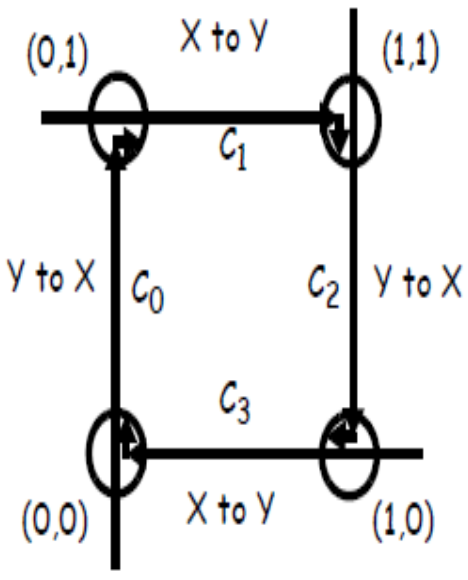
- Four Necessary Conditions For Deadlock see OpSys:
  1. Mutual Exclusion: only one agent can access the resource at a time
  2. No Preemption: once an agent acquired a resource, no mechanism can force it to relinquish it.
  3. Hold & Wait: agent holds acquired resources while waiting for others.
  4. Circular Wait: a set of agents wait on each other to acquire other's resources and no one can make any progress.

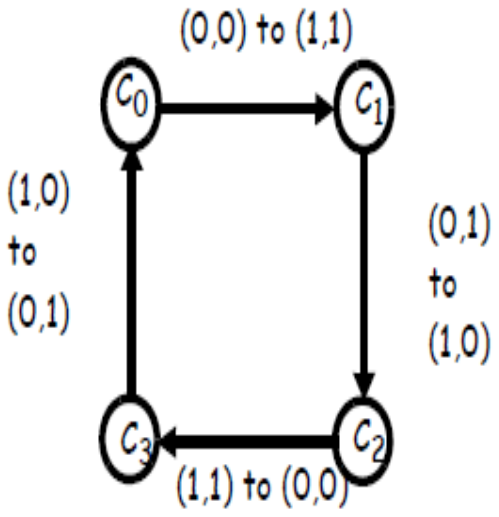
- In Network Case: agents are packets, resources are physical or logical channels.



Deadlock Detection for Mesh and Tori

Assume that packets are free to follow any route.



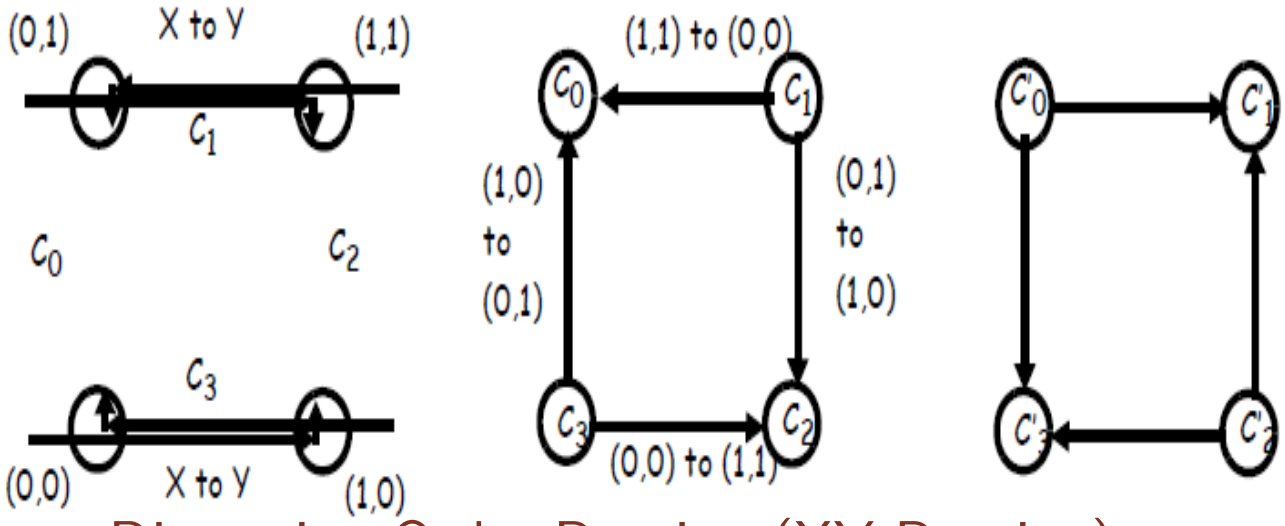


Each node tries in the same time to send a packet to the diagonally opposite node, e.g., `(0,0)` sends to `(1,1)`.

To avoid conflicts `(1,0)` uses first c_3 then c_0, `(0,0)` uses c_0 and c_1, etc.

The Resource Acquisition Graph or Channel-Dependency Graph on the right has cycles, hence circular wait, and as such deadlock is possible.

Deadlock Avoidance for Mesh and Tori



- Enforce Dimension Order Routing (XY Routing)
  - Packets move first Horizontally until they cannot anymore,
  - Then they move Vertically ⇒ No CYCLE, NO DEADLOCK.
- Problem: Contention for Channels ⇒ Wasted Bandwidth
  - Use virtual channels to increase routing flexibility & NO deadlock:
  - 2 buffers/switch ⇒ 2 sets of channels. One uses `XY`, the other `YX`.
  - Changing at most once from XY to YX routing ⇒ NO deadlock.
  - *West-First Routing:* packet is routed in the westward direction if destination node is west of source & NO deadlock. More flexible: only 2 turns are prohibited (in comparison to 4 for XY).