4. Networks

in parallel computers

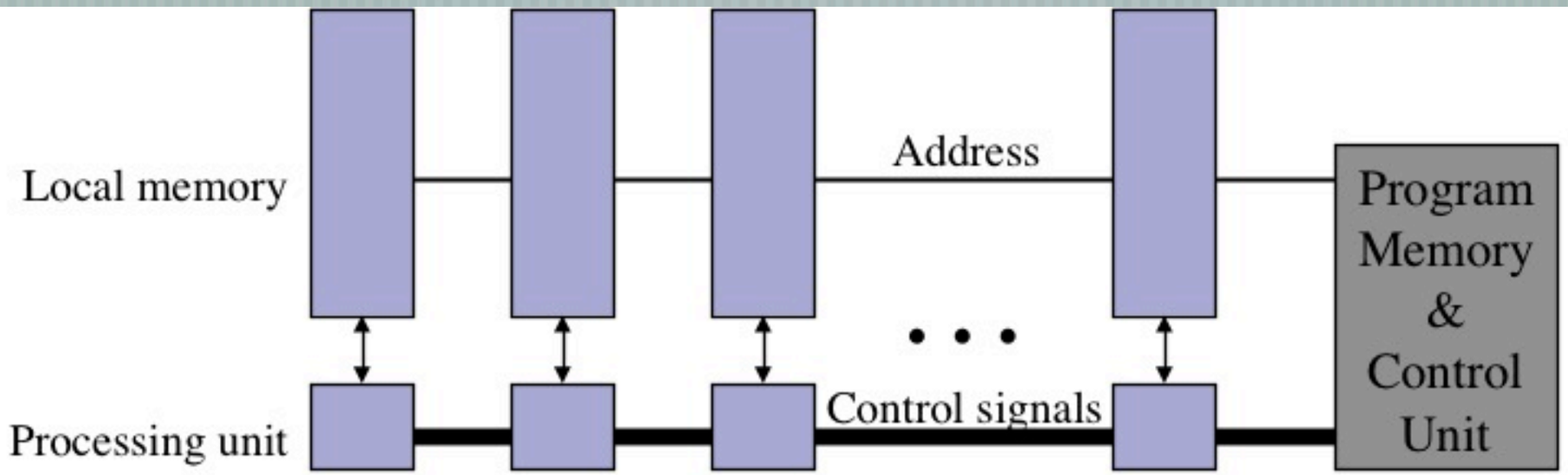
Advances in Computer Architecture

System architectures for parallel computers

Control organization

- Single Instruction stream Multiple Data stream (**SIMD**)
  - All processors execute the same instruction using one control unit and many data paths with control signals broadcast to each datapath from the control unit
  - Conditional execution possible: PE activity can be masked based on condition register
- Multiple Instruction Stream Multiple data Stream (**MIMD**)
  - Typically a collection of microprocessors connected by a bus or network:
    - it could be a group of PCs loosely interconnected by the internet
  - or a rack of processors more tightly interconnected with a high speed bus or network

SIMD Computer



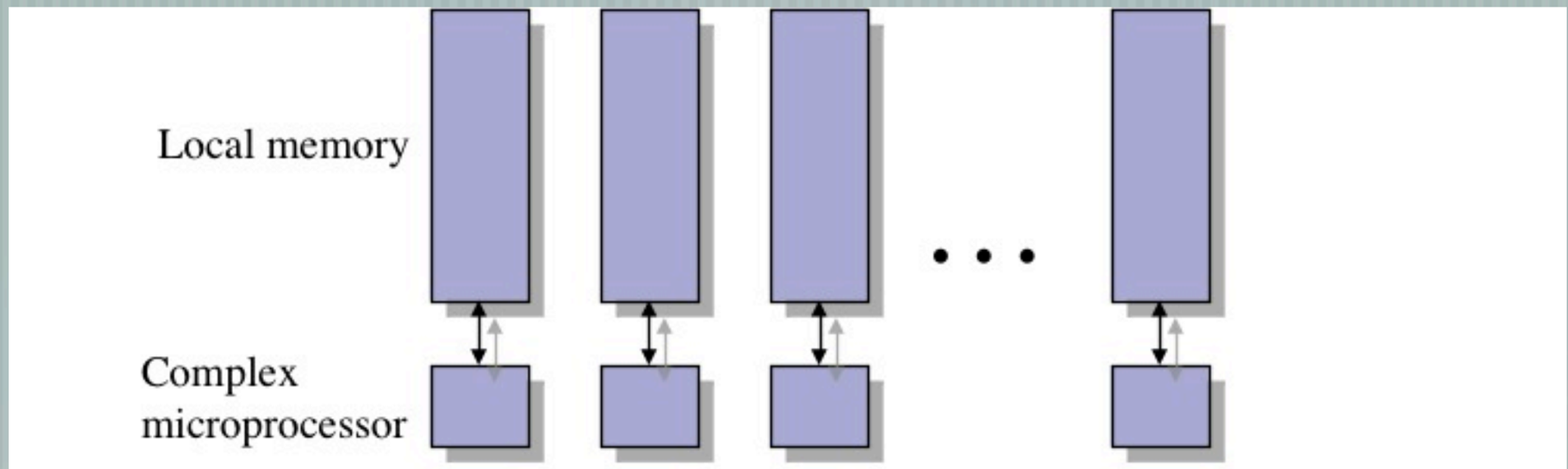
Processing unit (PU) is often very simple (e.g. 1 bit)

Local address modification expensive for a single bit processor

Scalability limited by broadcast delay of address and control to all PUs

Note that no data communication are shown between processing units but will be regular connections between processors

MIMD Computer



Use off-the-shelf microprocessors - economies of scale

Each processor is independent but will communicate with other processors

The granularity of inter-processor communications determines generality

Note that no data communications are shown between microprocessors this may be regular or ad-hoc (as in the internet)

Memory organization

- A **distributed memory** organization partitions the memory between the different processors where each processor "owns" a bank
  - only one processor can access each bank of memory
  - data is shared between processors by the exchange of messages managed in software or hardware
  - messages are routed to processors based on a processor addressing scheme (e.g. IP)
- A **shared memory** organization allows each processor to access any word of data anywhere in memory
  - data is accessed by address regardless of its location
  - shared memory is not exclusive to bus-based multi-processors
  - shared memory can be implemented over packet-addressed networks
  - access times to different locations in memory may not be uniform - NUMA

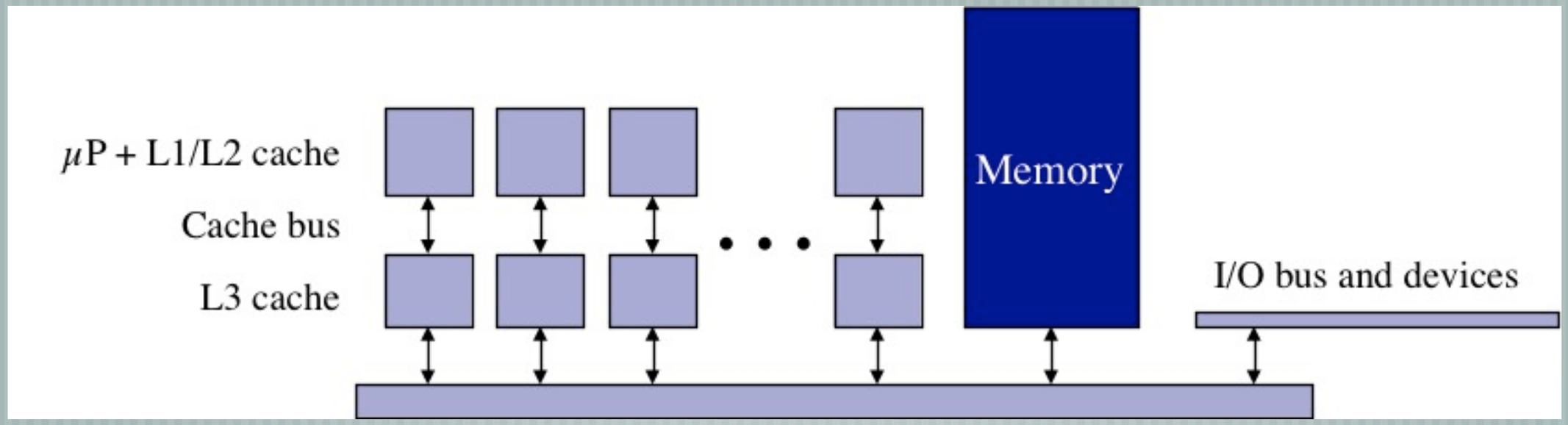
Typical MIMD examples

- Multi-processor PCs + multi-cores use bus-connected shared memory or central switch for memory channels
  - typically with 2-16 processors
  - they are symmetric and have uniform access to memory (UMA)
- Symmetric Multi-Processor (SMP) servers use switch/bus-connected shared memory, multiple chips
  - typically 32-128 processors
  - they also have uniform but slower access to memory
- Massively Parallel Processors (MPPs) use network connected memory which can be distributed or shared
  - Typically 256 to hundreds of thousands in supercomputers
  - They have non-uniform access to memory if it is shared as accesses are implemented using message passing
  - There may be many orders of magnitude difference between access times to L1 and remote memory

Communication architecture

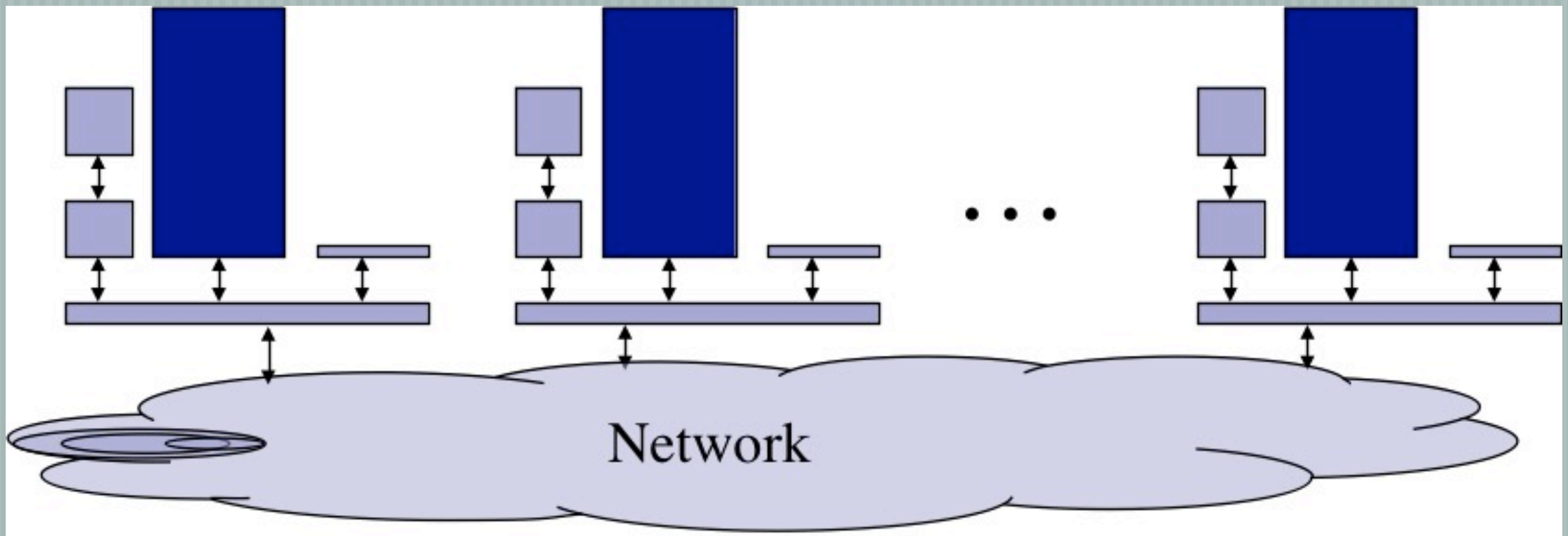
- There are two ways in which microprocessors may be connected to form parallel computers
- By shared **Bus** - giving shared memory multi-processors
  - e.g. MP PCs, SMPs
- By **Network** - giving distributed or shared memory
  - e.g. SMPs and MPPs
- It is common for architectures to use a combination of the two
  - e.g. bus-based SMPs interconnected by a network

Symmetric Multi-Processors



- Processors in an SMP have a symmetric view on the shared memory
- May be at any level of the memory hierarchy, L1:L2, L2:l3, L3:main
- Communication is by shared memory reading and writing a given address ... snoopy protocol for memory coherence between processor caches
- Synchronisation is usually implemented in software and requires an indivisible operation (e.g. test and set on a memory location)
- A global synchronisation mechanism is usually implemented on the bus

Massively parallel processors



Single processor node, connected with networks

Scalability

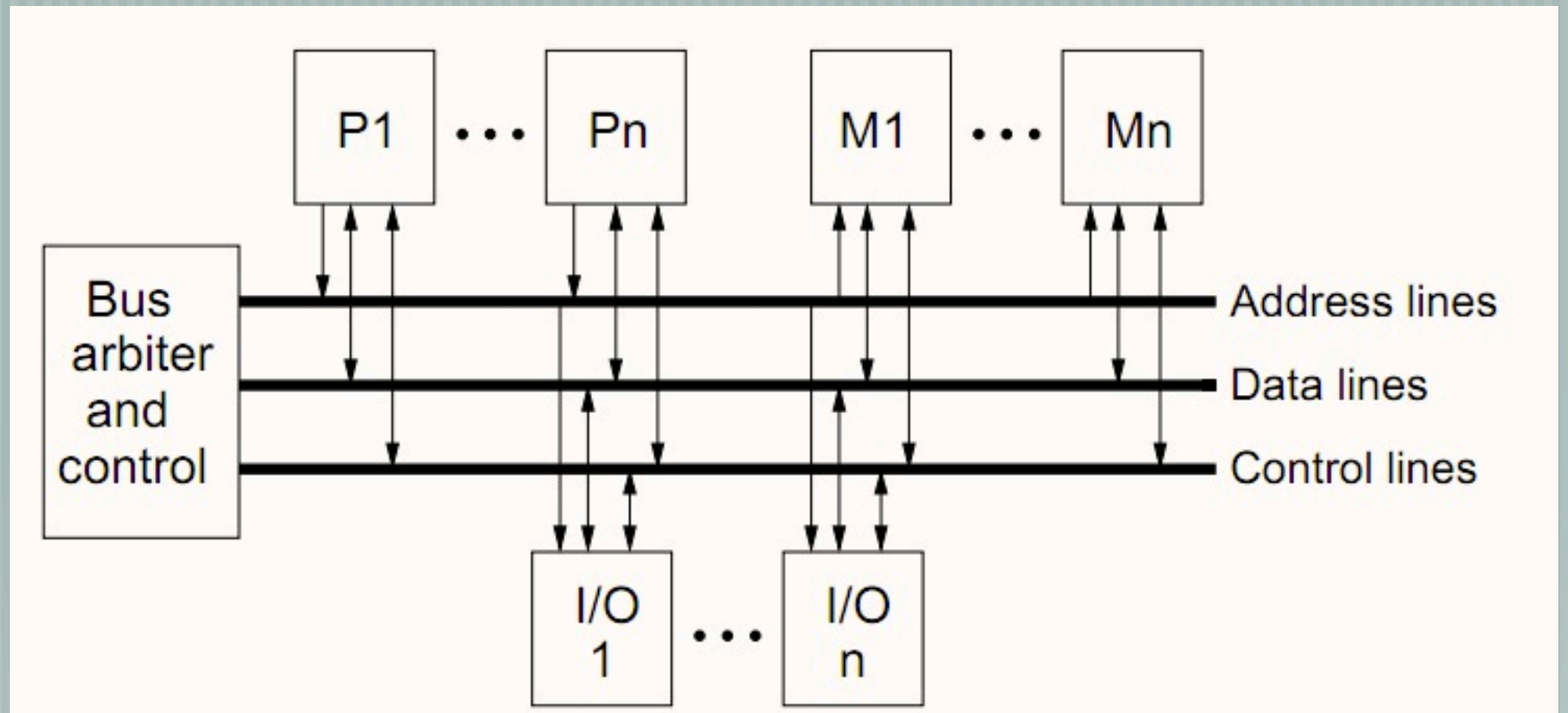
- The goal in a parallel computer is to scale processor performance linearly with the number of processors used
  - processor performance is generally scalable but communication and synchronization are not!
- A bus has a constant performance so it scales very badly
  - adding processors eventually saturates the bandwidth and congestion gives decreasing performance with concurrency
  - The throughput is constant for P processors

Scalability

- A fully connected switch (X-bar) scales throughput with concurrency
  - e.g. for P processors concurrent busses can be established between any of the processors with throughput of O(P)
- The problem with fully connected switches is that the costs grow very quickly i.e. $O(P^2)$ for P processors
  - Theoretically, at best we can get O(P) throughput with O(logP) delay and at a cost of O(PlogP) - e.g. hypercube network

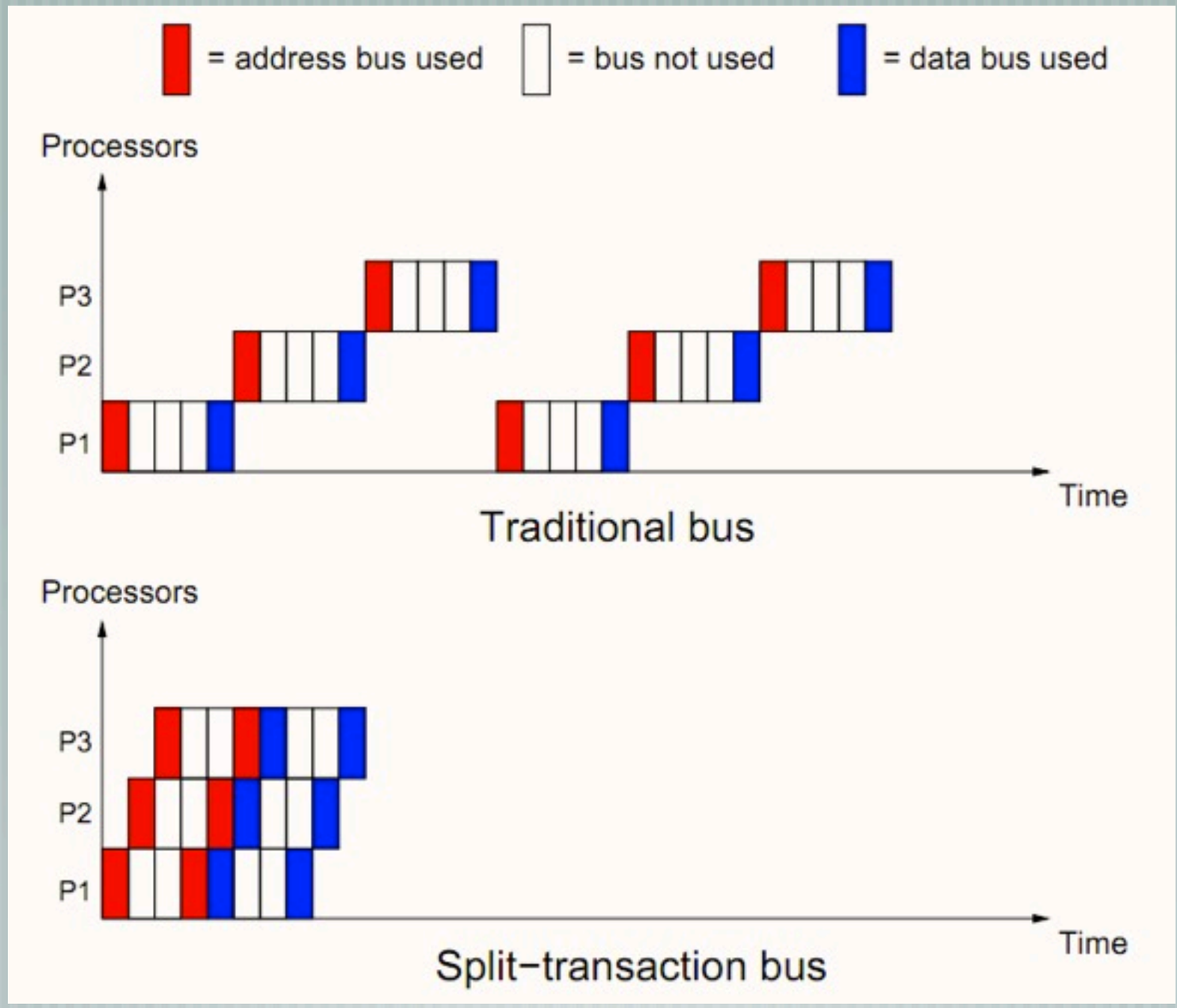
Buses

Bus architectures

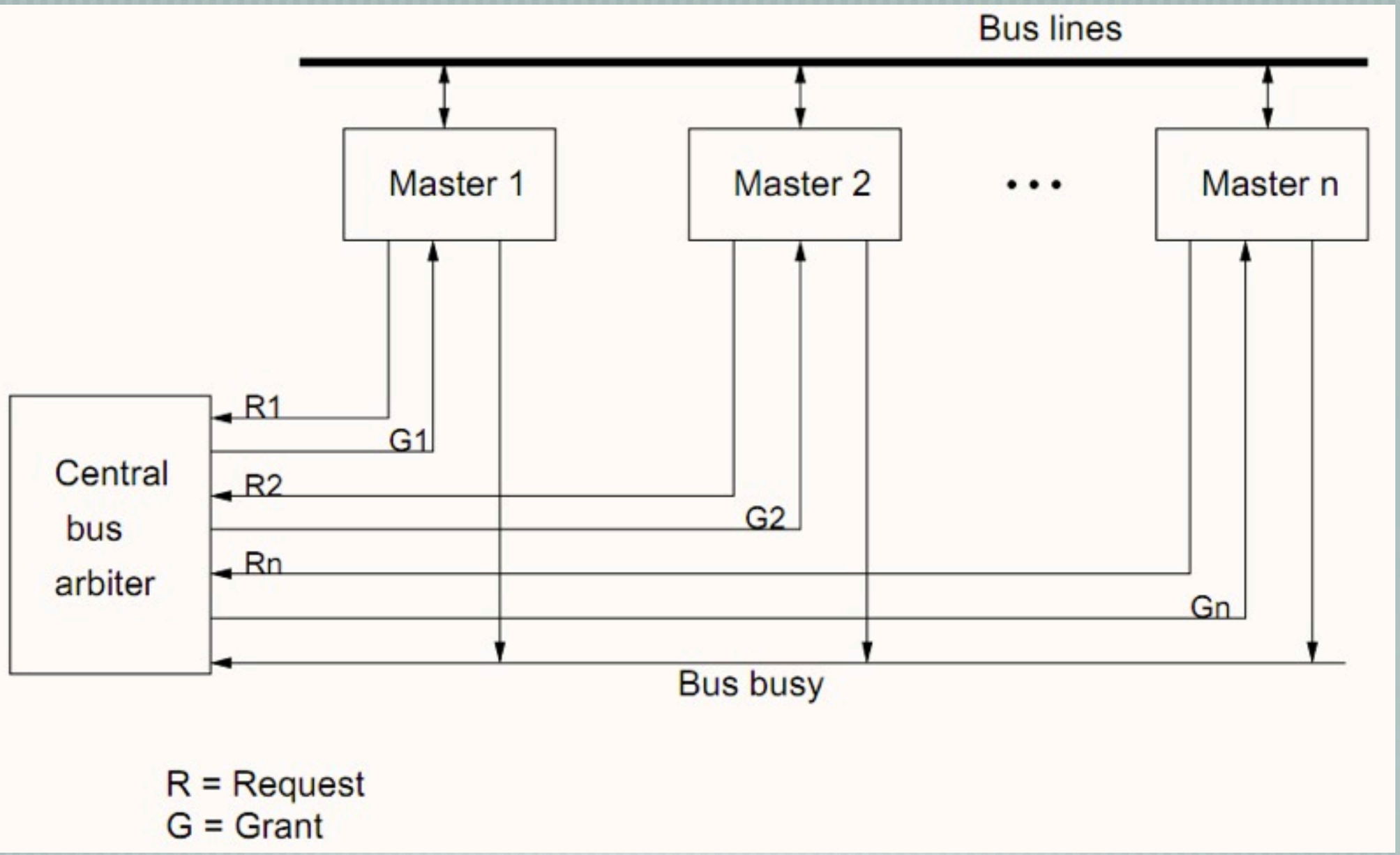


- In traditional busses, address and data lines may be time-multiplexed
- When there are more **bus-masters**, arbitration is required

Traditional vs split-transaction

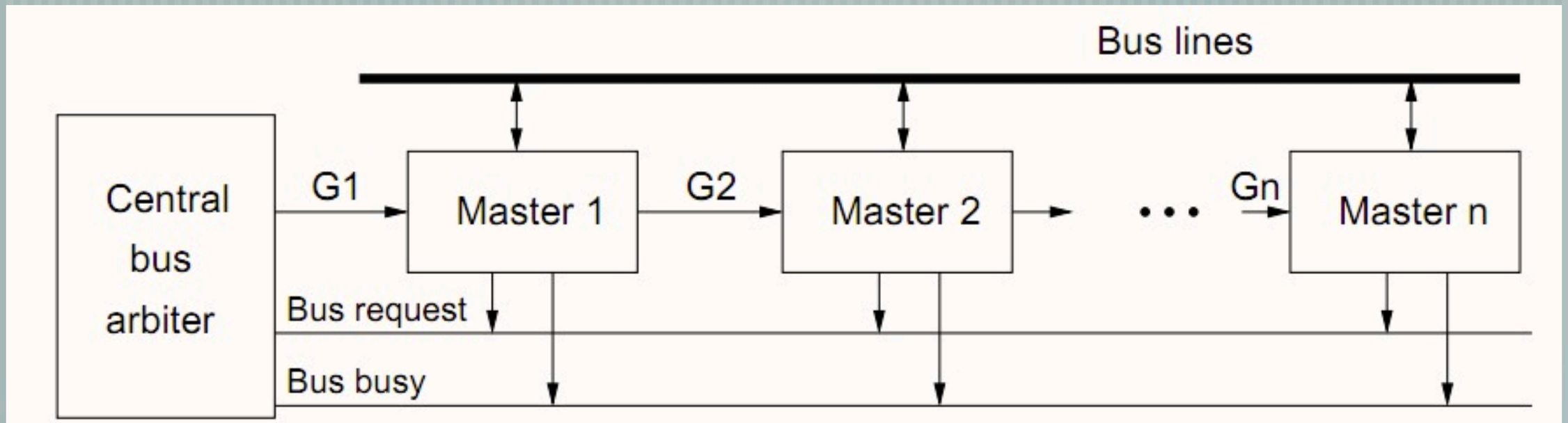


Central arbitration



flexible + efficient, expensive, not scalable

Daisy-chain arbitration



higher latencies, less fairness, cheaper

Networks for parallel computers

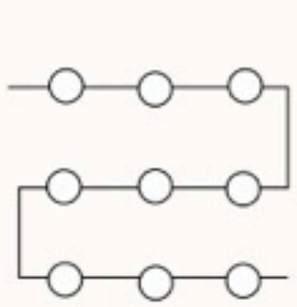
Bandwidth and latency

- **Bandwidth of a channel**
  - number of bits per second that can be passed over a channel
  - usually defined in terms of the channel width and the channel cycle
  - a *w* bit channel at *f* MHz has a bandwidth of *f*w* Mbps
- **Latency of a channel** - serialization latency
  - the time required to pass a message of a given size over a channel
  - a message of *m* bits in the above channel requires *m/ (f*w)* secs

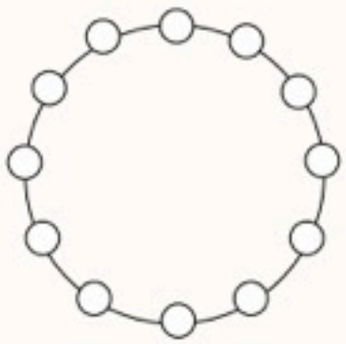
Network topology

- **Physical topology**: how the wires are connected
- **Logical topology**: what nodes can be considered as neighbors for the purpose of sending/receiving messages
- Cost intimately related to topology:
  - more wires, more cost
  - more neighbors, more communication, more performance

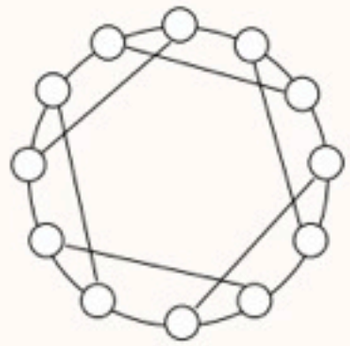
Example topologies



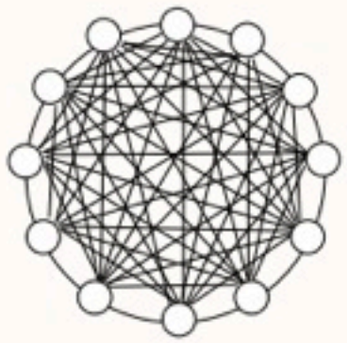
Linear array



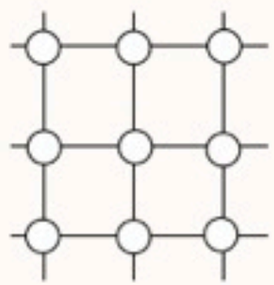
Ring



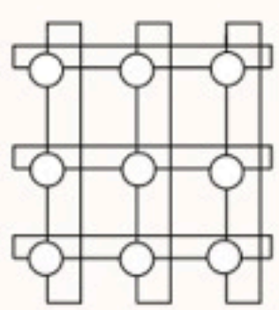
Chordal ring of degree 3



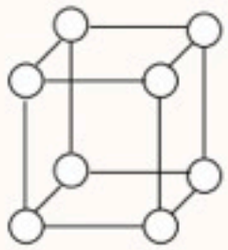
Completely connected



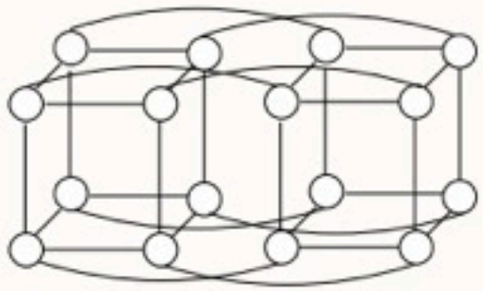
Mesh



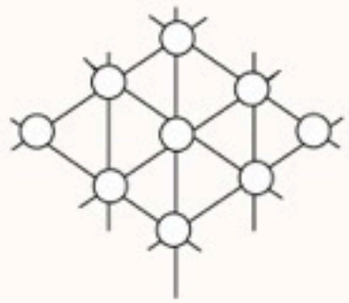
Torus



3−Hypercube

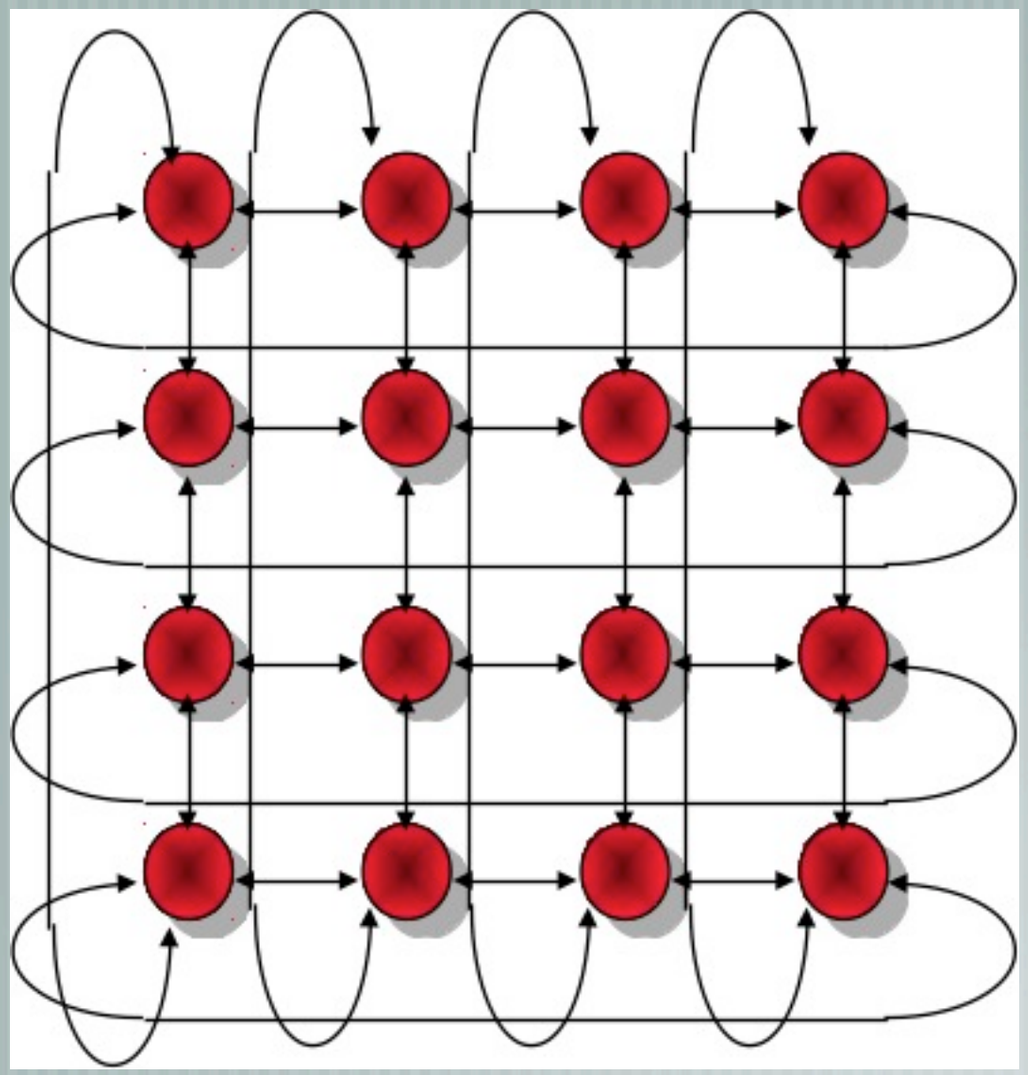


4−Hypercube



Systolic array

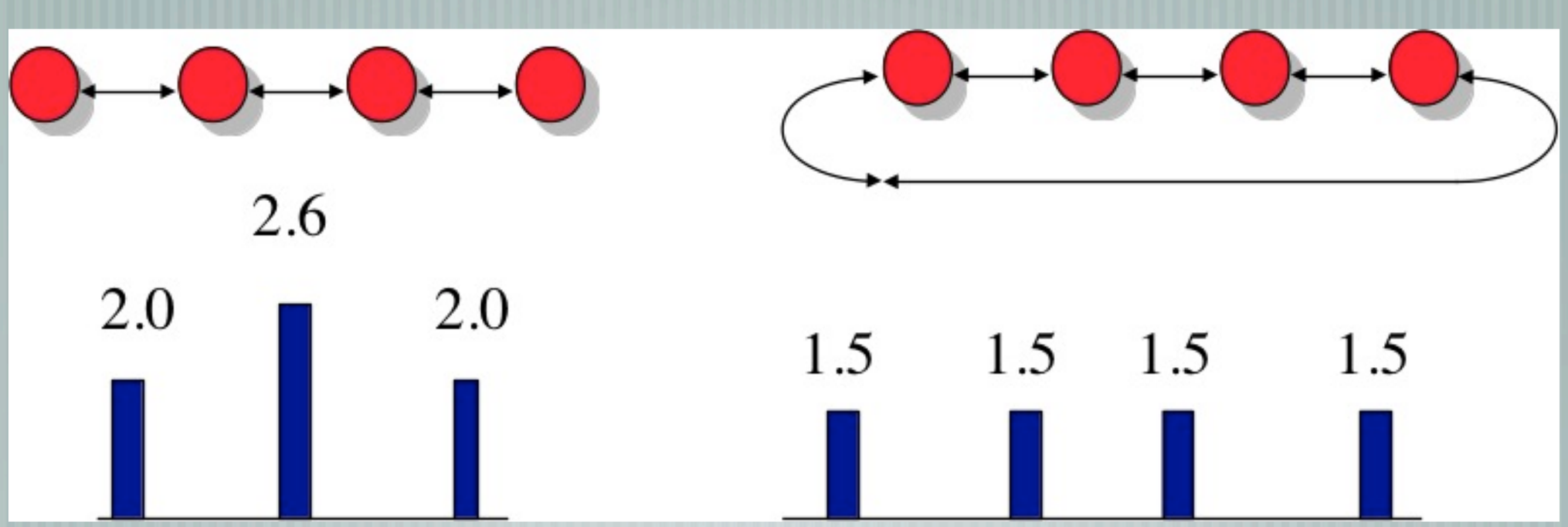
Toroidal topologies



By providing wrap around connections on each edge the distance measure is reduced by a factor of 2. Moreover the network also becomes symmetrical.

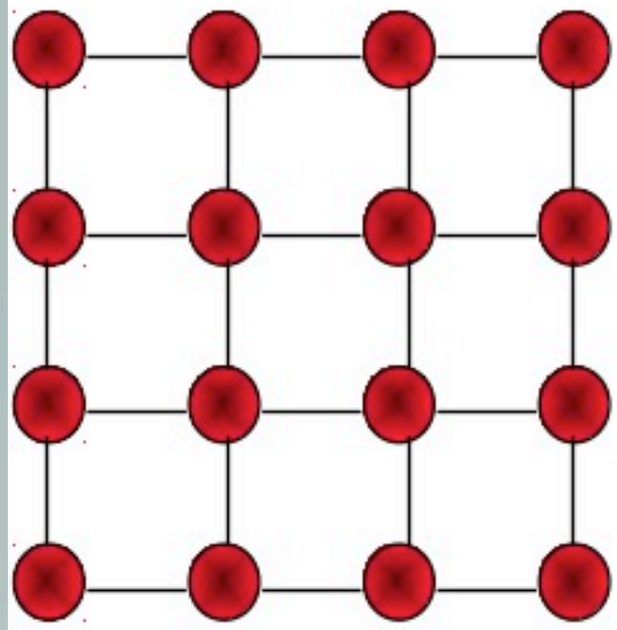
Whereas a mesh suffers from congestion at its centre, a torroidal network has an even distribution of traffic.

Link load under uniform random traffic

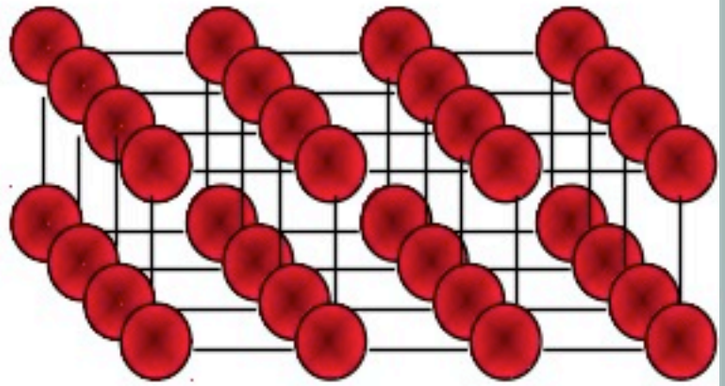


Multi-dimensional topologies

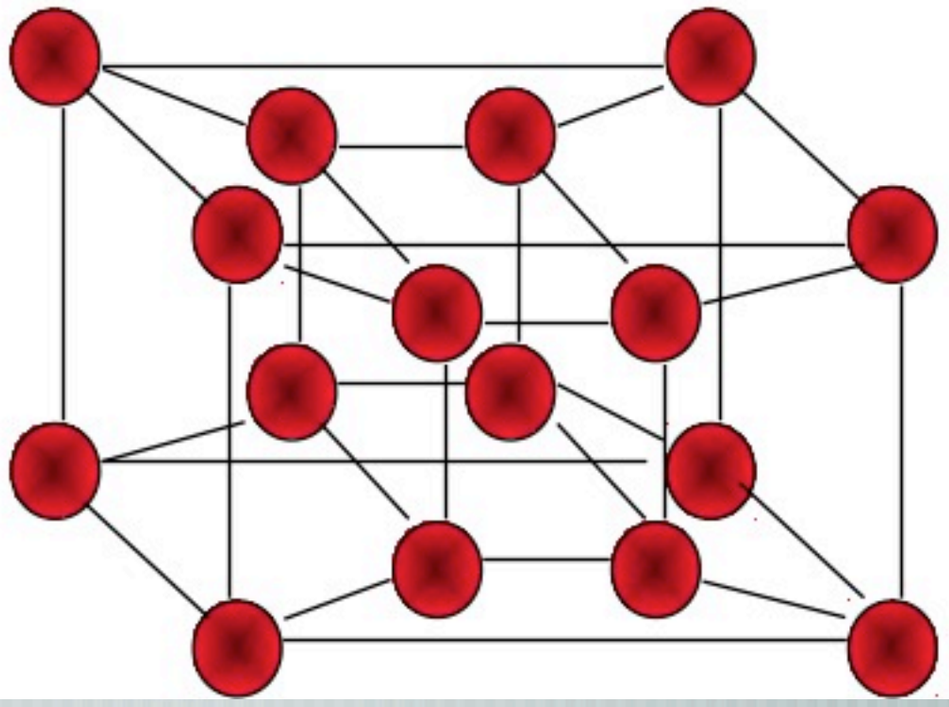
2-D Grids



3-D Cubes



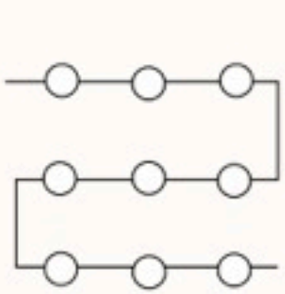
Hypercubes



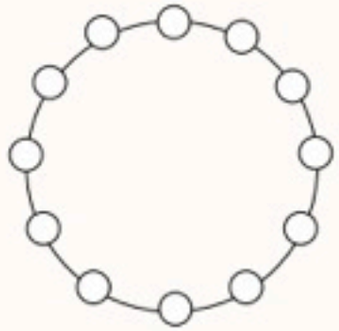
Metrics related to topology

- **Node degree**: number of channels connected to one node
  - 2D Mesh: degree 4, ring: degree 2
- **Diameter**: maximum shortest path between two nodes
  - Ring: N/2, 2D Mesh: $2N^{1/2}$
- **Bisection width**: when the network is cut into two equal halves, the minimum number of channels along the cut
  - This determines the bandwidth from one half to another

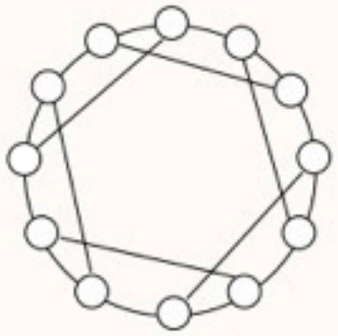
Topologies and metrics



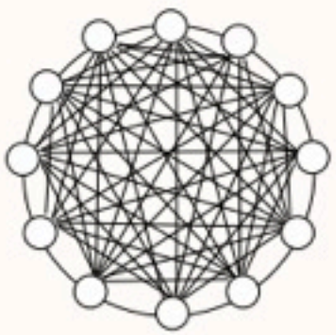
Linear array



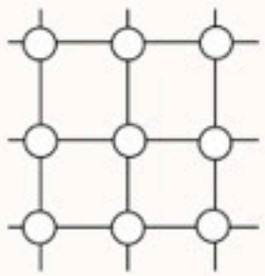
Ring



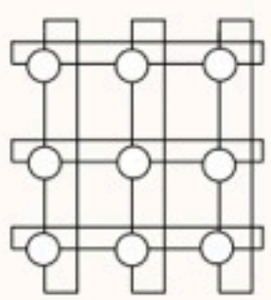
Chordal ring of degree 3



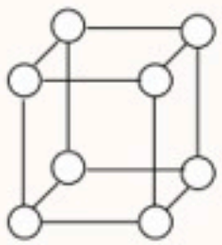
Completely connected



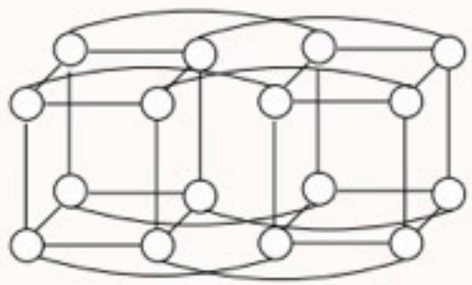
Mesh



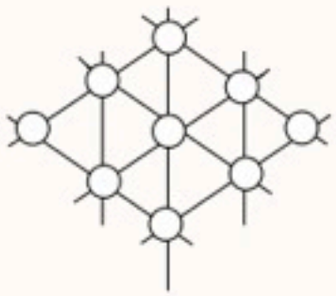
Torus



3−Hypercube



4−Hypercube



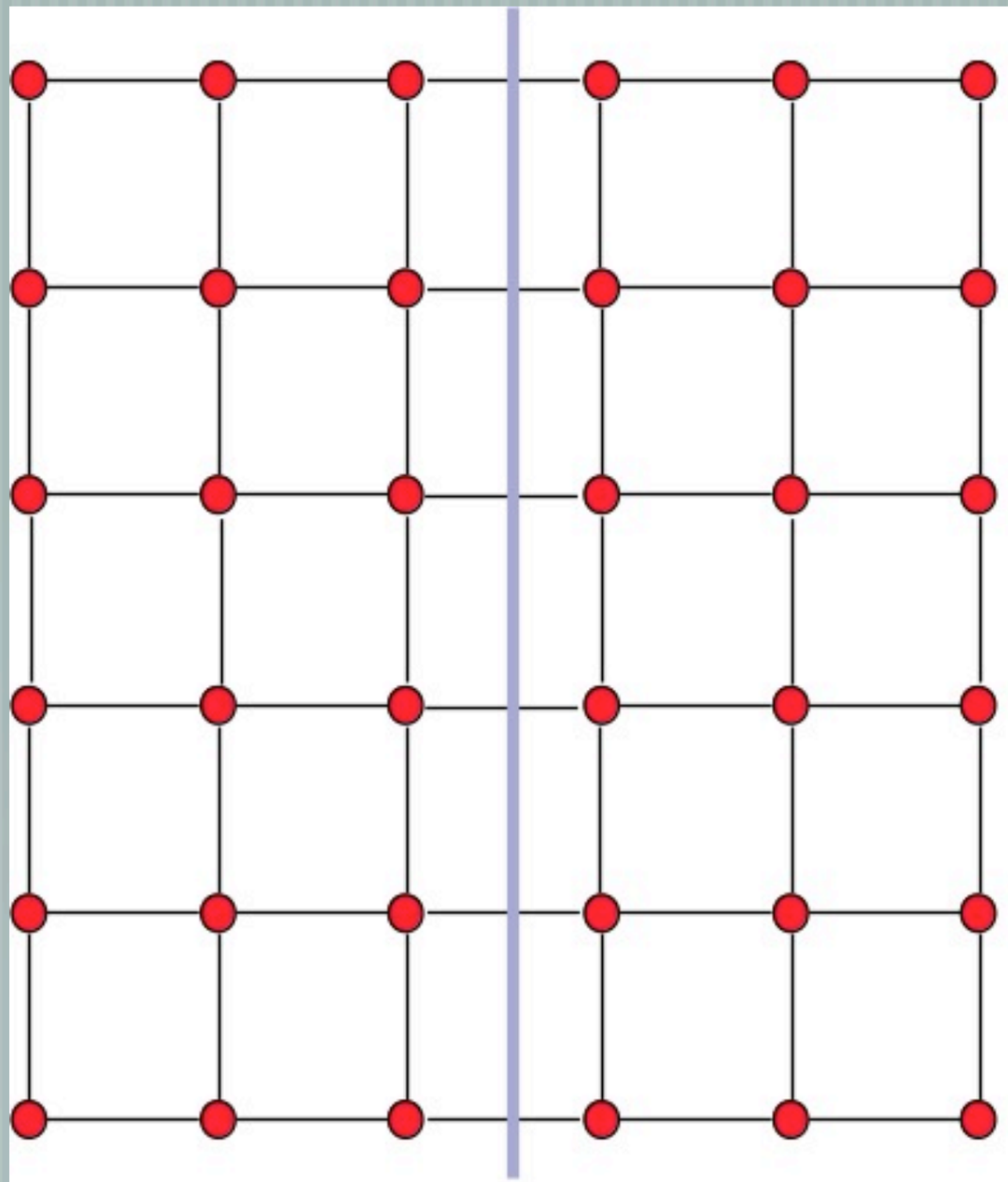
Systolic array

Net	Degree	Diameter	Bisection width
Ring	2	N/2	2
Fully conn.	N-1	1	$(N/2)^2$
2D mesh	4	$2N^{1/2}-1$	$N^{1/2}$
Hyper cube	$\log_2 N$	$\log_2 N$	N/2

Dimensionality vs. cost

- As the **dimensionality** of a network, D, increases **so does its degree** and the **cost of each node**
- **As the dimensionality increases** the distance measures of diameter and average distance, and hence **latency, decrease**
- A compromise normally results... but there are other factors
  - Wire length is an issue - 2-D and 3-D networks can be mapped in 3-D space but a hypercube always has non-local wires that can increase latency and decrease throughput

Bisection analysis: mesh is $O(n^{1/2})$



Local Traffic

Only nodes adjacent to the bisection contribute to the traffic across the bisection. This is scalable, i.e. as the network size is increased the traffic on the bisection remains constant.

Uniform and Random Traffic

Assume the capacity of each channel is C and the load placed on the network by each node is L. Consider only traffic across the bisection. Each node will contribute L/2 as the probability of a message crossing the bisection is 1/2. Therefore link capacity required for a given node load is:

$$Cn^{1/2} = nL/2$$

$$C=n^{1/2}L/2$$

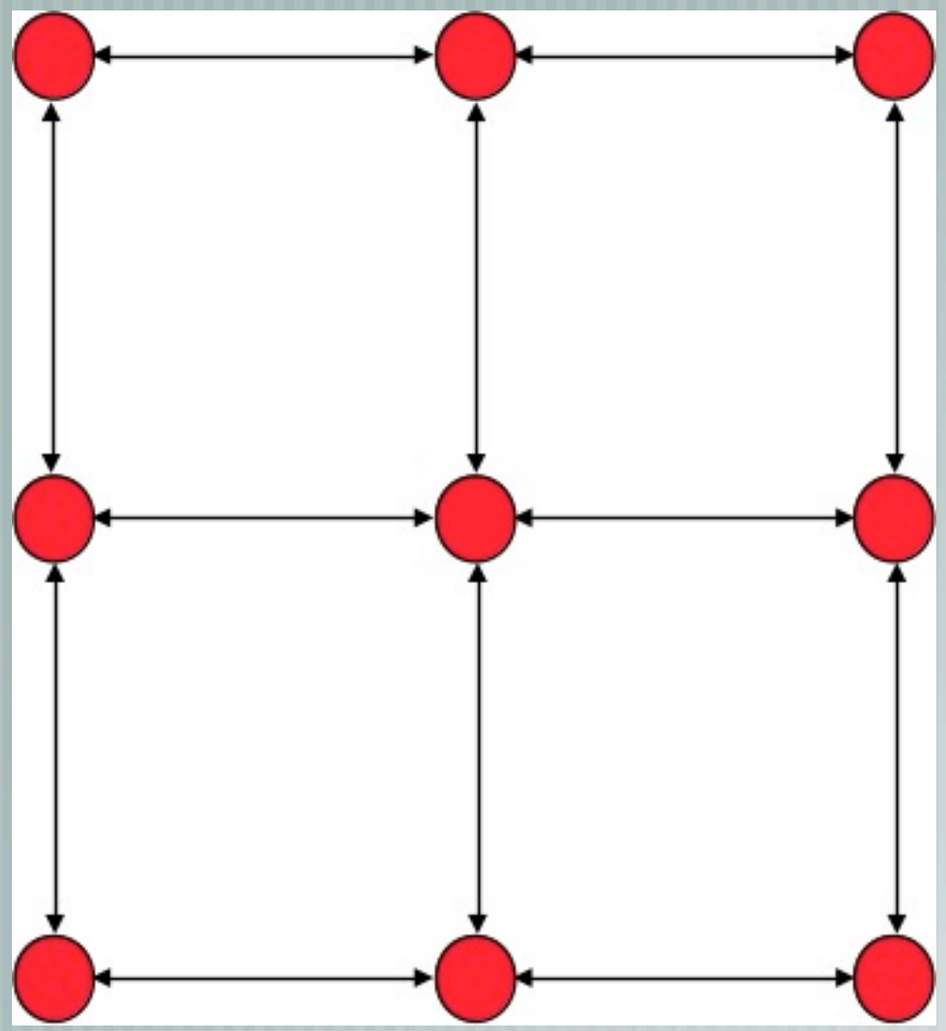
In a 2-D mesh for uniform random traffic the link bandwidth must scale by $n^{1/2}$ to support n processors

Dynamic vs static networks

- A **static** network provides a **pre-programmed set of connections**
  - messages are addressed to ports & ports connect to different nodes in the network

- What is the abstraction here?
  - one of permutations of data
  - message are exchanged over fixed topologies which is a low-level and application specific form of programming
  - This is typical of SIMD computers and some low-level DSP devices like the picoChip Array

Example static network

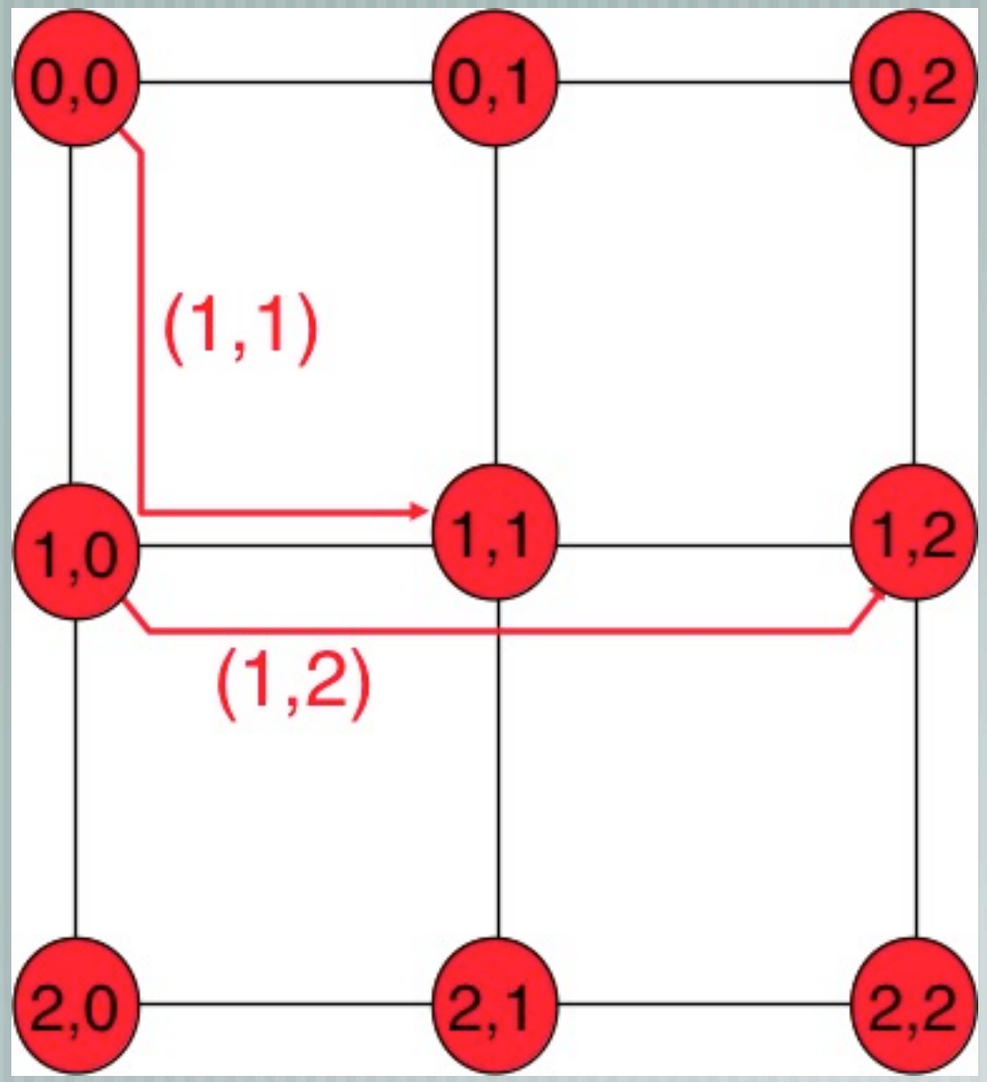


- Processor i,j in the grid may only send messages to processors:
  - i,j+1
  - i,j-1,
  - i+1,j,
  - i-1,j
- In a planar grid there are exceptions at the boundaries
- In a cyclic network (torus) there is uniformity or symmetry

Dynamic vs. static networks

- In a **dynamic** network each **message uniquely identifies its destination** by means of a header within the message
  - i.e., packet routing
  - the header addresses the node in the network to which the message is to be delivered, and possibly the manner in which it will interpreted there
- What is the abstraction here?
  - the abstraction here is well understood, one of address spaces
  - an address to a destination provides data or action
  - this can be abstracted and is able to support any programming paradigm

Example dynamic network



In this example, each node in the network has an address as does each message

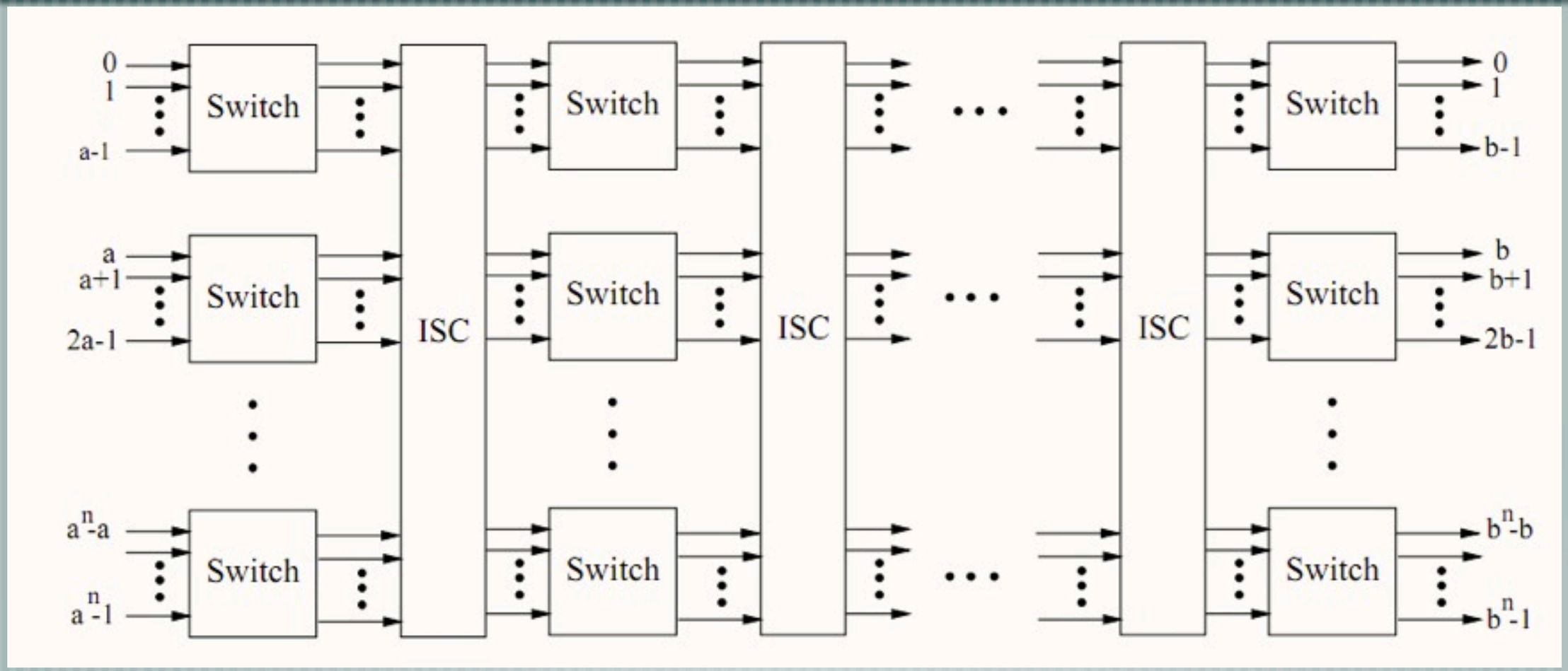
The processor dynamically computes an address and the nodes in the network route the message to its destination using a number of transfers or hops

Indirect networks

Direct vs. indirect networks

- **Direct networks** are characterized by **point-to-point connections** between nodes, typically fixed topology
- **Indirect networks** allow "remote" nodes to exchange messages, hopping through intermediate nodes
  - The physical links are shared by multiple point-to-point "virtual" neighbors
  - A bus is a special form of indirect network (1 physical channel, topology changes at every transaction)

Multistage networks - indirect

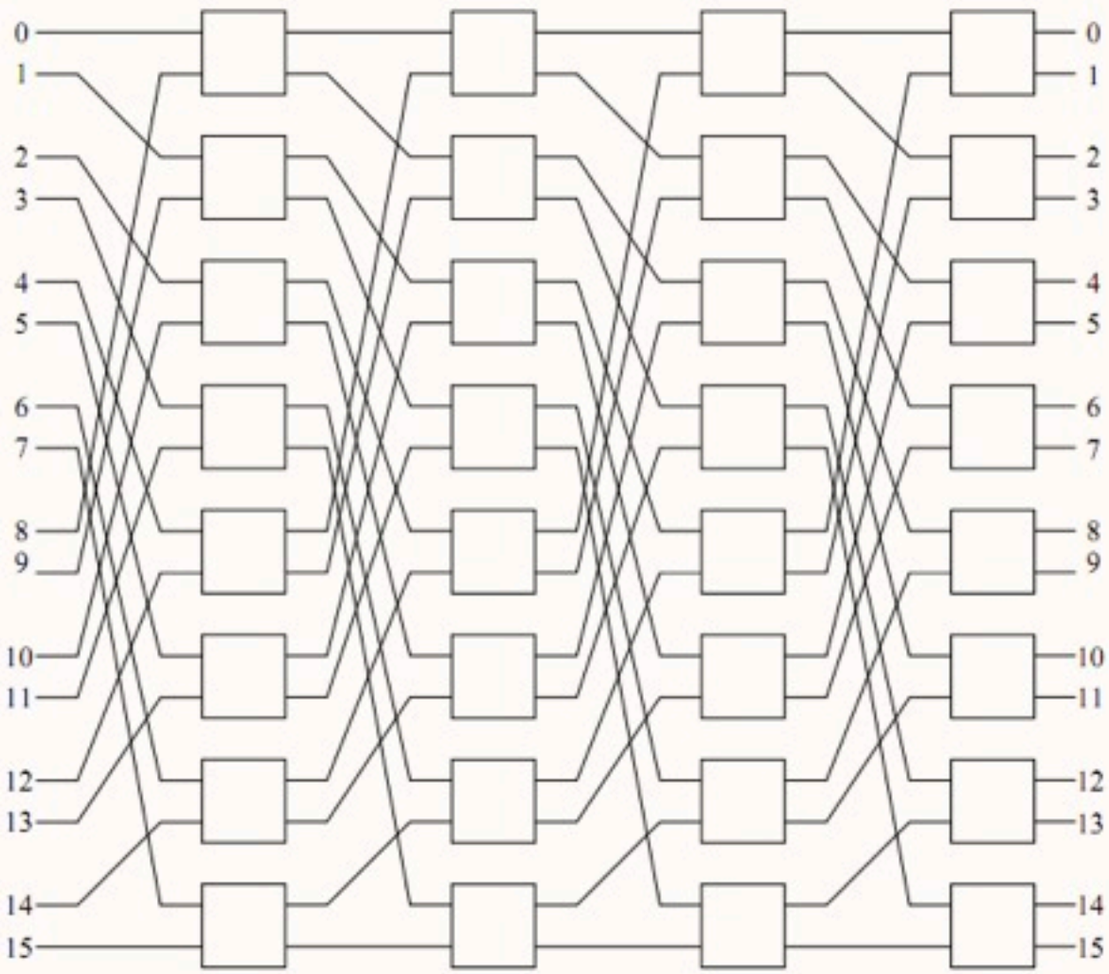


- A generalized Multistage Interconnection Network (MIN)
- built with a x b switches and a specific InterStage Connection (ISC) pattern

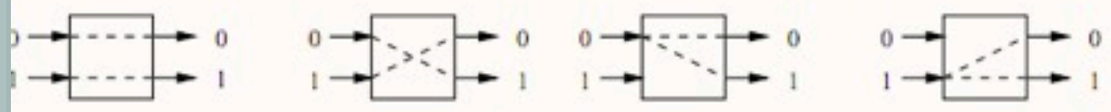
Multistage networks - indirect

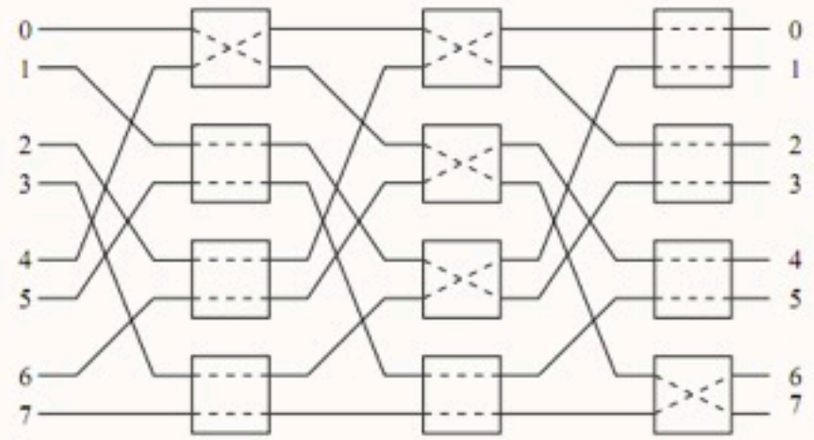
- Switch networks often use 2x2 switches
- N inputs require $\log_2 N$ stages of 2x2 switches
- Each stage requires N/2 switch modules
- The number of stages determines the delay of the network

Omega networks - indirect

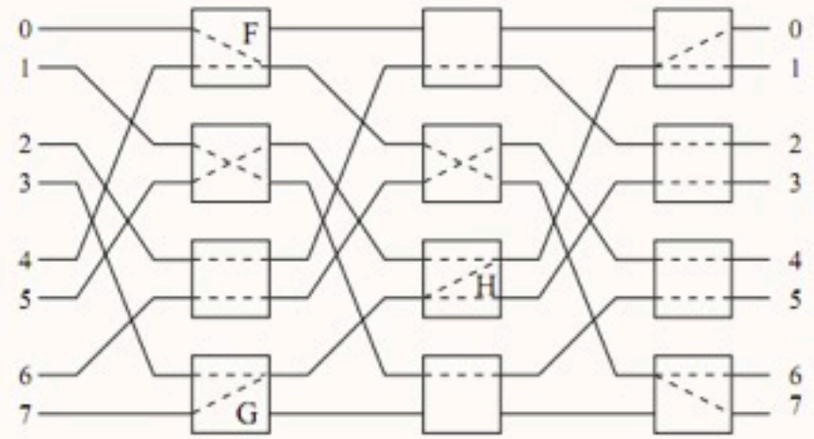


A 16x16 Omega network of 2x2 switches



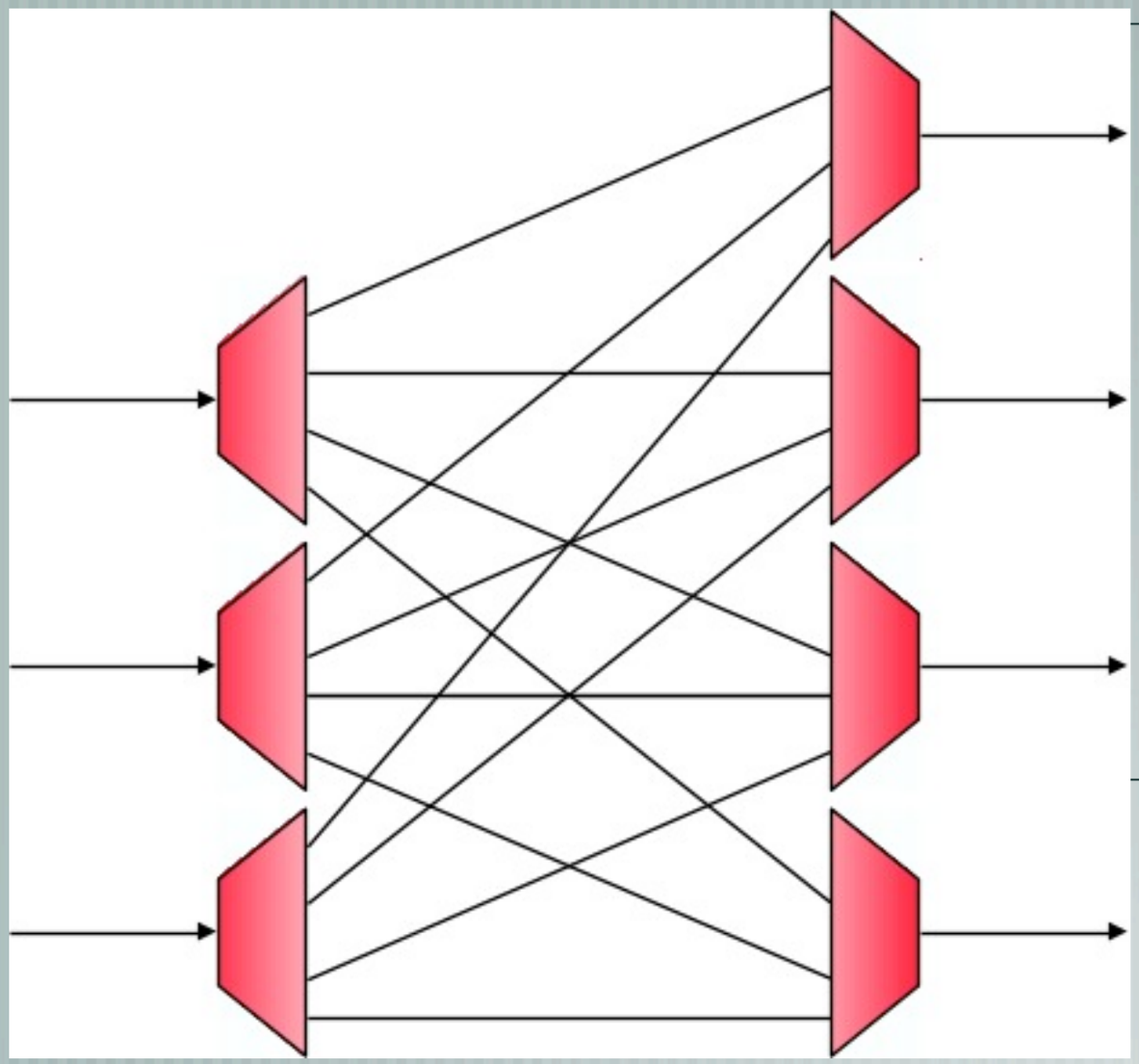


Permutation (0,7,6,4,2)(1,3)(5) without blocking



Permutation (0,6,4,7,3)(1,5)(2) blocked at switches F,G and H

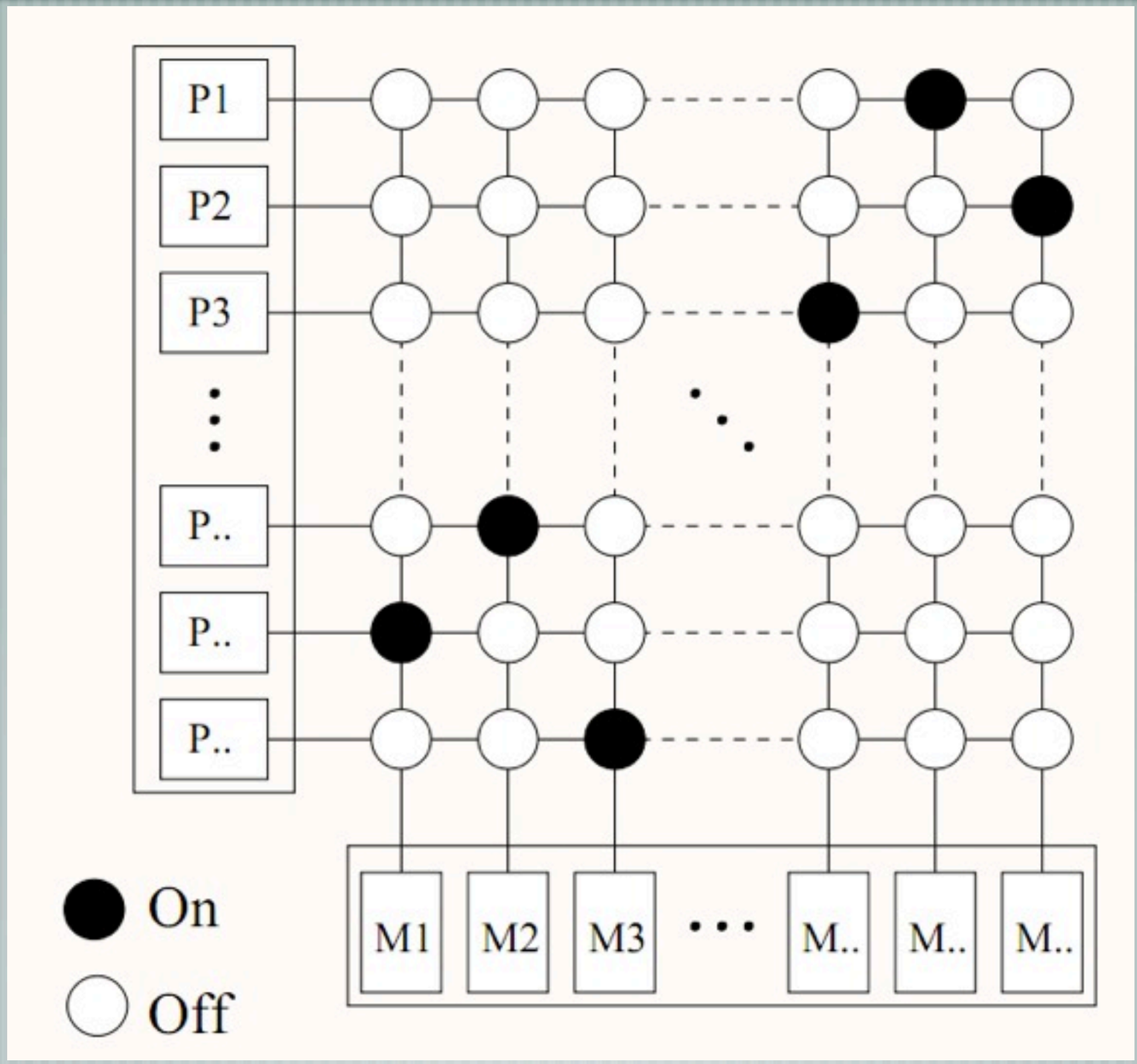
Crossbars (X-bars)



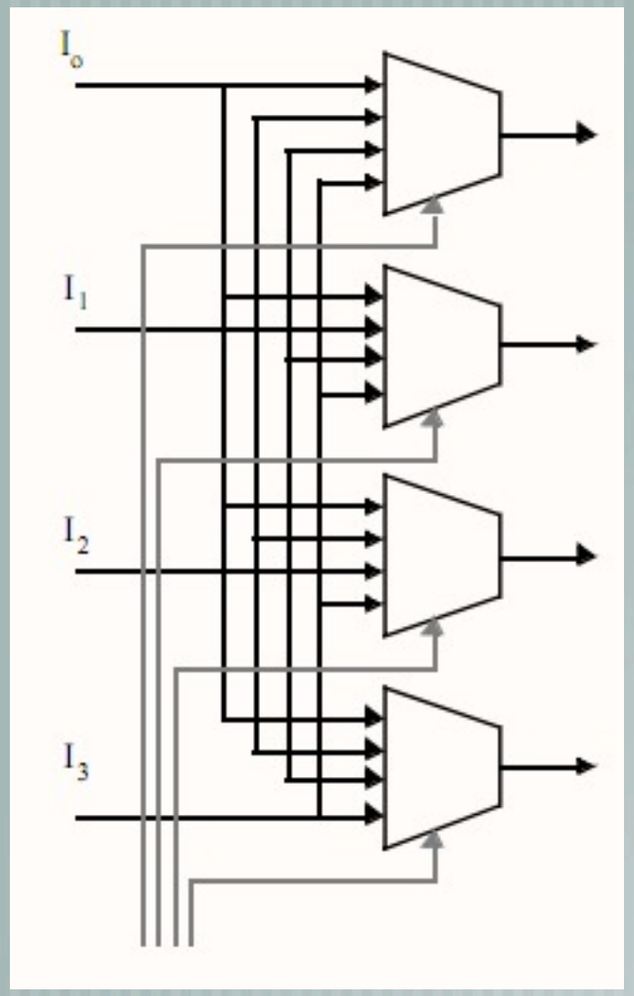
The X-bar connects every input to every output
It is **indirect**, yet has a **fully connected logical topology**

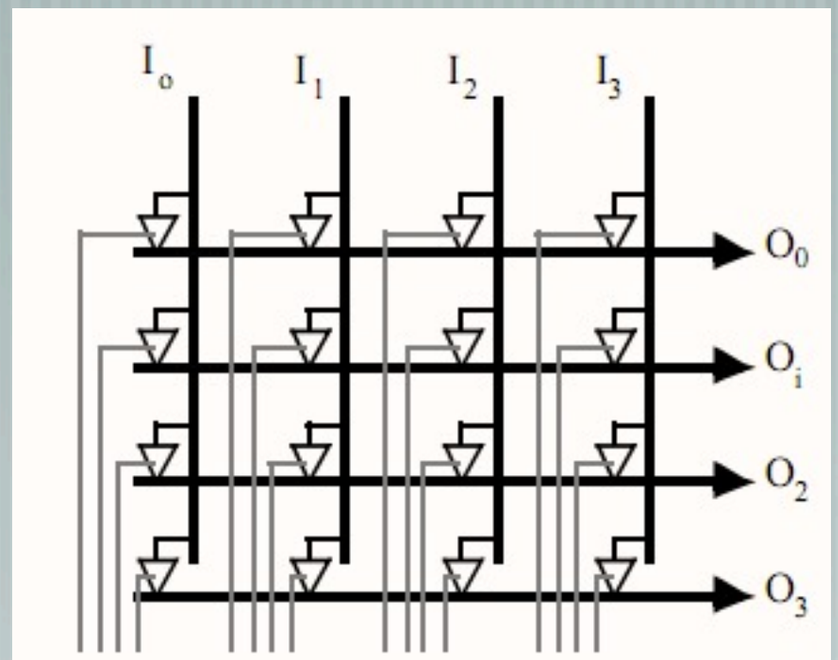
The cost of this network scales very badly as each node requires n*m channels

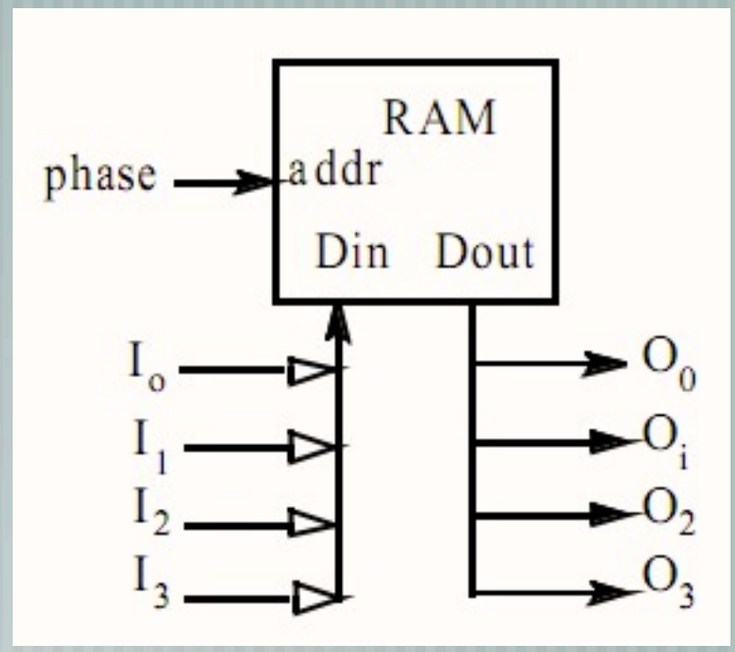
X-bar switches



X-bar switches







Scaling in indirect networks

Network characteristics	Bus	Multistage network	Crossbar switch
Min. latency	constant	$O(log_k n)$	constant
Bandwidth	$O(w)$	$O(w)$ to $O(nw)$	$O(w)$ to $O(nw)$
Wiring complexity	$O(w)$	$O(nwlog_k n)$	$O(n^2 w)$
Switching complexity	$O(n)$	$O(nlog_k n)$	$O(n^2)$
Connectivity and routing capability	Only one to one at a time	Some permutations and broadcast, if network unblocked	All permutations one at a time

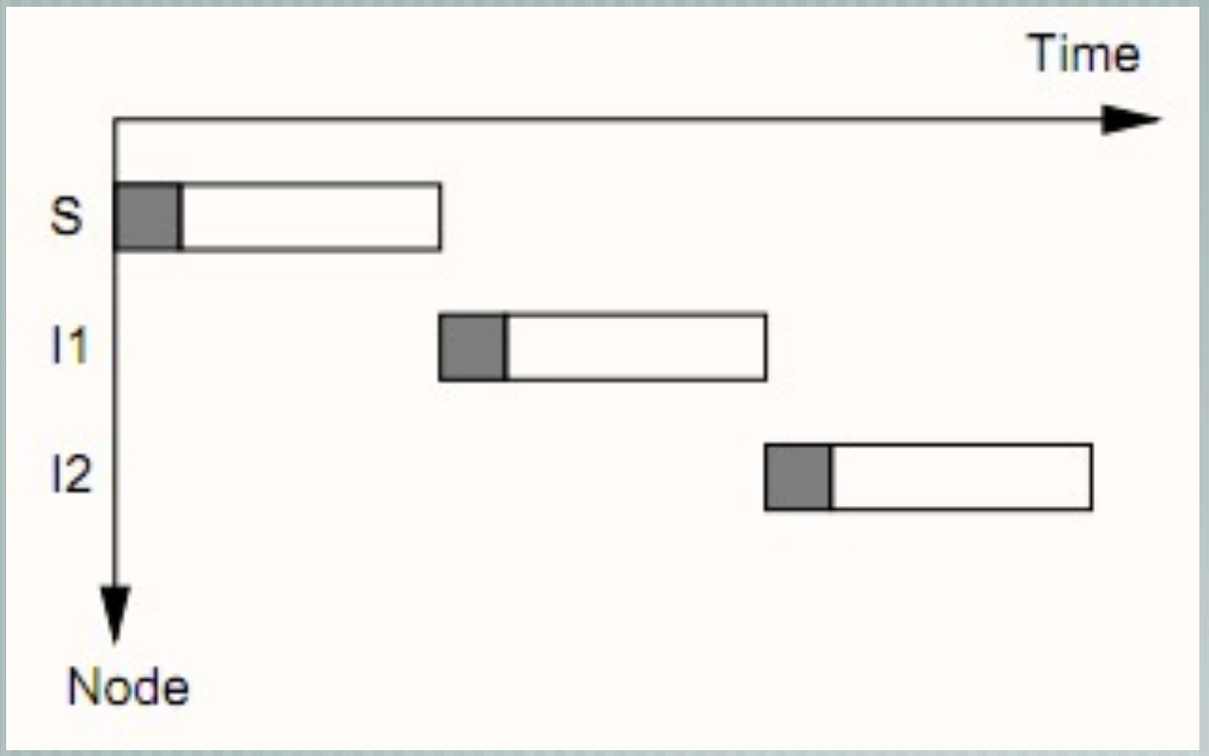
Direct vs. indirect

- **Direct** networks provide **more efficient communication**
  - at the expense of dedicated wires for every channel
- **Indirect** requires **larger latencies** due to hops
- **Indirect** requires **routing through the intermediate nodes**
  - Except for buses, once switched the distance is 1
  - Except for cross bars, only one switch required
  - Multistage: N hops where N is the number of stages

Network switching

- **Network switching**: how **channels** are established
  - **Circuit switching**: the channels are established for the entire communication (eg. old telephone system)
  - **Packet switching**: communication split into packets, channels established for individual packets
- Parallel computers have evolved from circuit to packet switching, which is more flexible, yet more costly

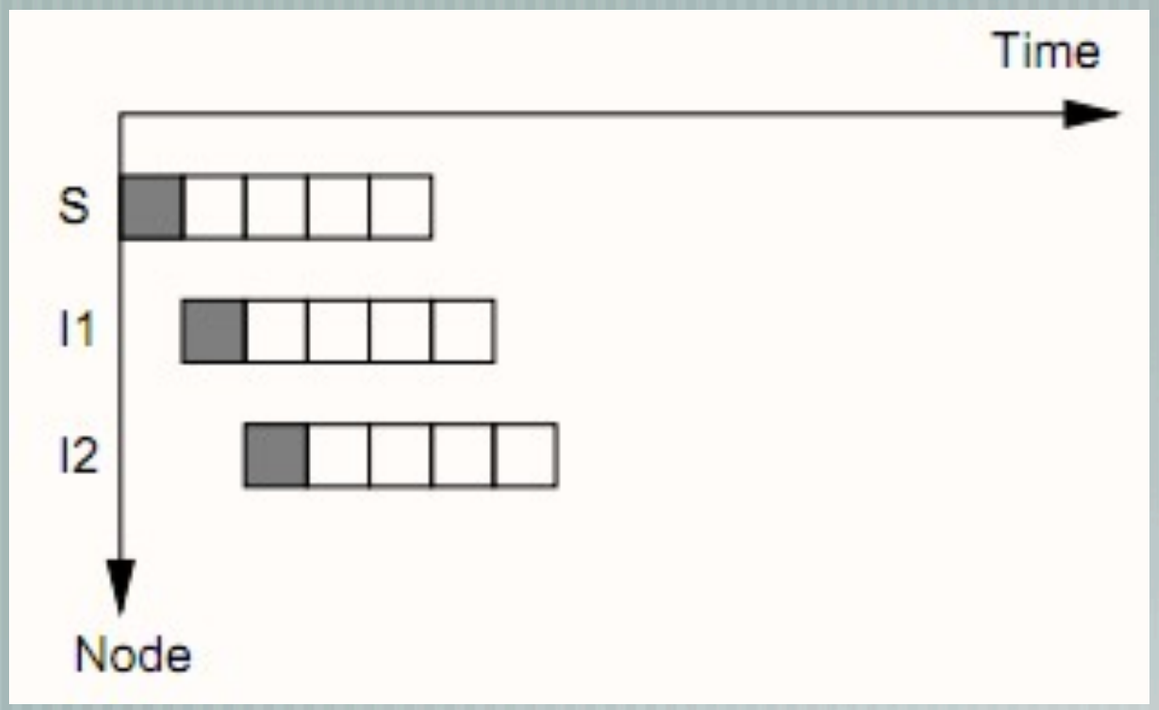
Packet switching



- **Store-and-forward**:
  - Packet is the smallest entity
  - Packet buffered at every node

$$T_{s\&f} = \frac{L}{W} \times D$$

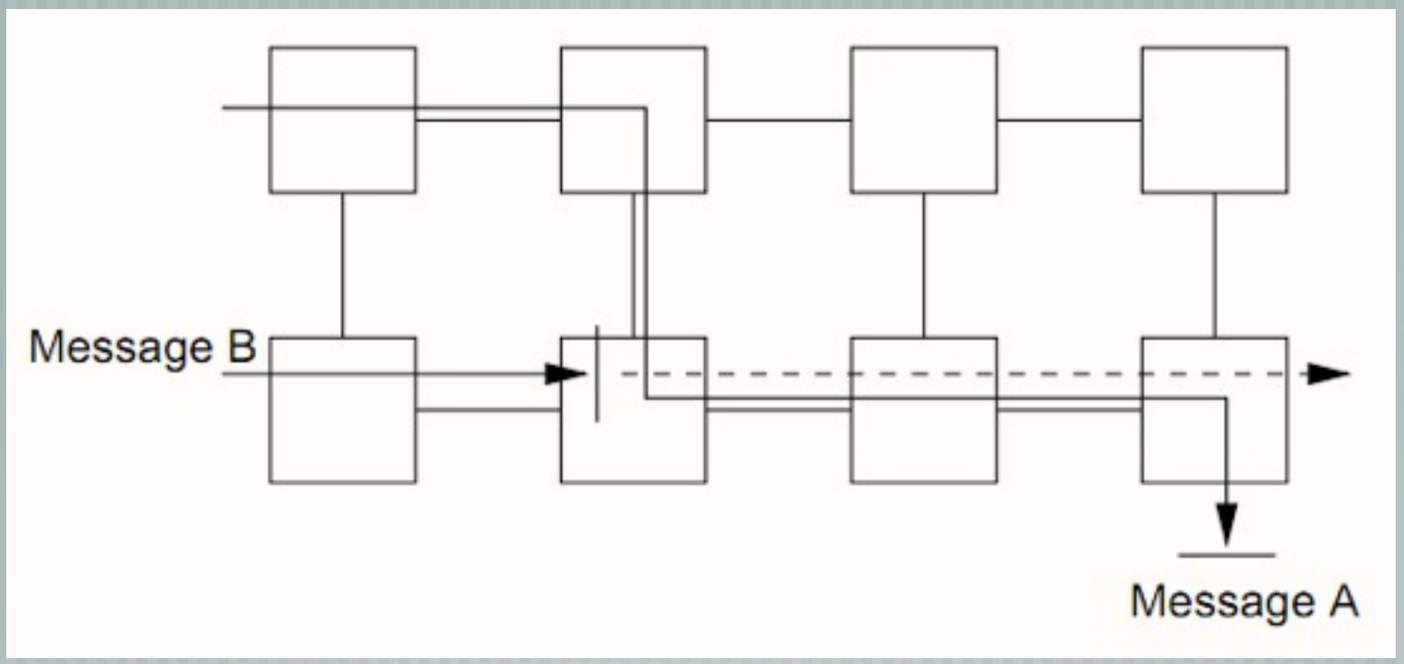
Packet switching



$$T_{wormhole} = \frac{L}{W} + \frac{F}{W}(D-1)$$

- **Wormhole routing**:
- Packet split in **flits**
- One packet can occupy multiple links
- A form of **pipelining**

Tree saturation



- Occurs when the head of the packet cannot advance, and tail flits are keeping channels busy
  - As the channels are blocked, other packets get stuck
  - This blocking behavior propagates with the structure of a tree
- Two solutions: **virtual channels** (cf later) and **virtual cut-through switching**

Virtual cut-through switching

- Mix of wormhole routing and store-and-forward:
  - **Flits advance as with wormhole**
  - Each node has a **buffer for an entire packet**
  - If the head cannot advance, all the tail flits can accumulate at the node where the head is blocked, freeing the other channels

Network routing

- **Network routing**: how data is **steered** through the channels
  - Which paths messages should take
  - Intimately depends on topology
    and how many nodes per link (direct vs. indirect)

Routing techniques

- Source-based routing:
  - Routers “eat” the head of a packet
  - Larger packets, no fault tolerance, no contention management
- Local routing: more complex, more flexible
- Routing may cause deadlocks
  - Buffer deadlocks (store and forward)
  - Channel deadlock (wormhole)

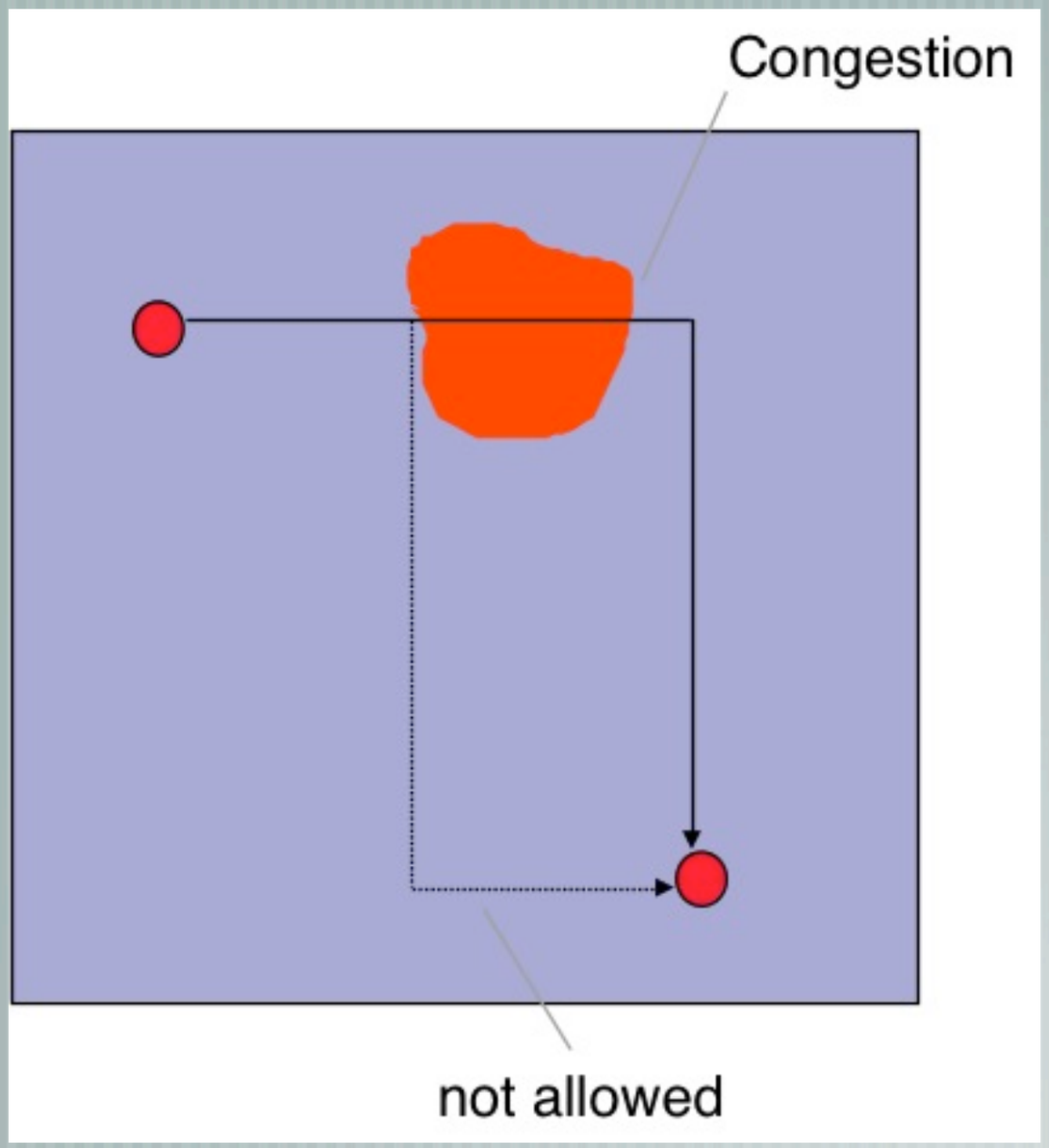
Deterministic vs. adaptive routing

- **Deterministic** (non-adaptive) routing : fixed path
  - Minimal and deadlock free
- **Adaptive routing** : exploits alternative paths
  - Less prone to contention and more fault-tolerant
  - Potential deadlocks
  - Reassembling of messages (out-of-order arrival of packets)

Dimension-order routing

- In multi-dimensional, non-cyclic networks, if routing is restricted then we can still maintain the one buffer per direction limit while avoiding cyclic dependencies
- The restriction is to route different dimensions in some fixed order, i.e. **route all packets in X before routing in Y** in a 2D mesh (XY-routing)

Dimension-order routing

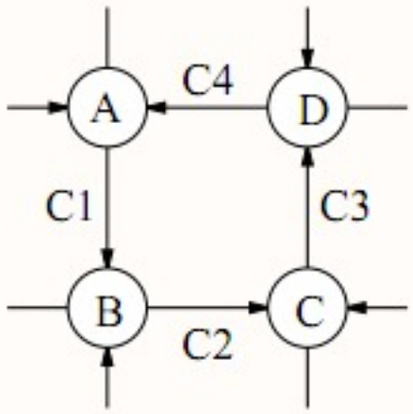


- With dimension-order routing there is just one path between any two nodes in a regular network and if there is congestion on-route it can not be avoided
- This is the one disadvantage of dimension-order routing

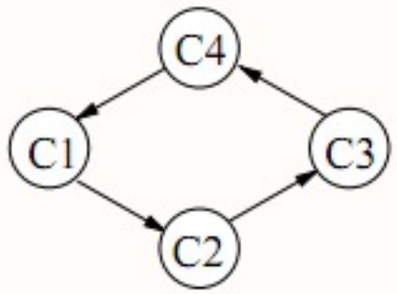
Deadlocks in dimension order routing

- In cyclic networks, rings, toruses, etc. dimension-order routing can still cause deadlock
  - cyclic dependencies can again arise due to wrap around in any dimension
- Possible to avoid this by providing a **second buffer**, aka another virtual channel
  - packets are transferred from one virtual channel (buffer) to another at some fixed point in the cycle, called the *date line*

Virtual channels



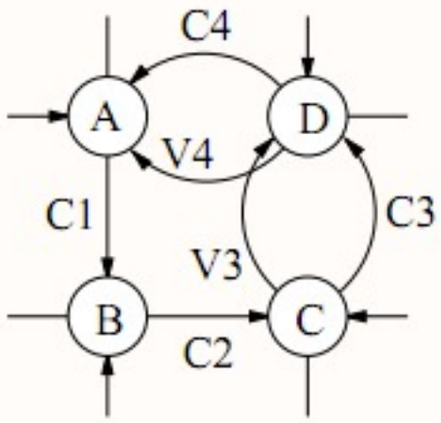
Channel deadlock



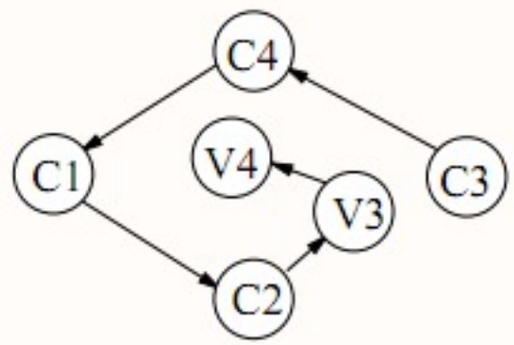
Channel-dependence graph containing a cycle

Deadlock occurs when there are cycles in the channel dependency graph

Virtual channels remove cycles hence deadlocks
Implemented with different buffers at nodes



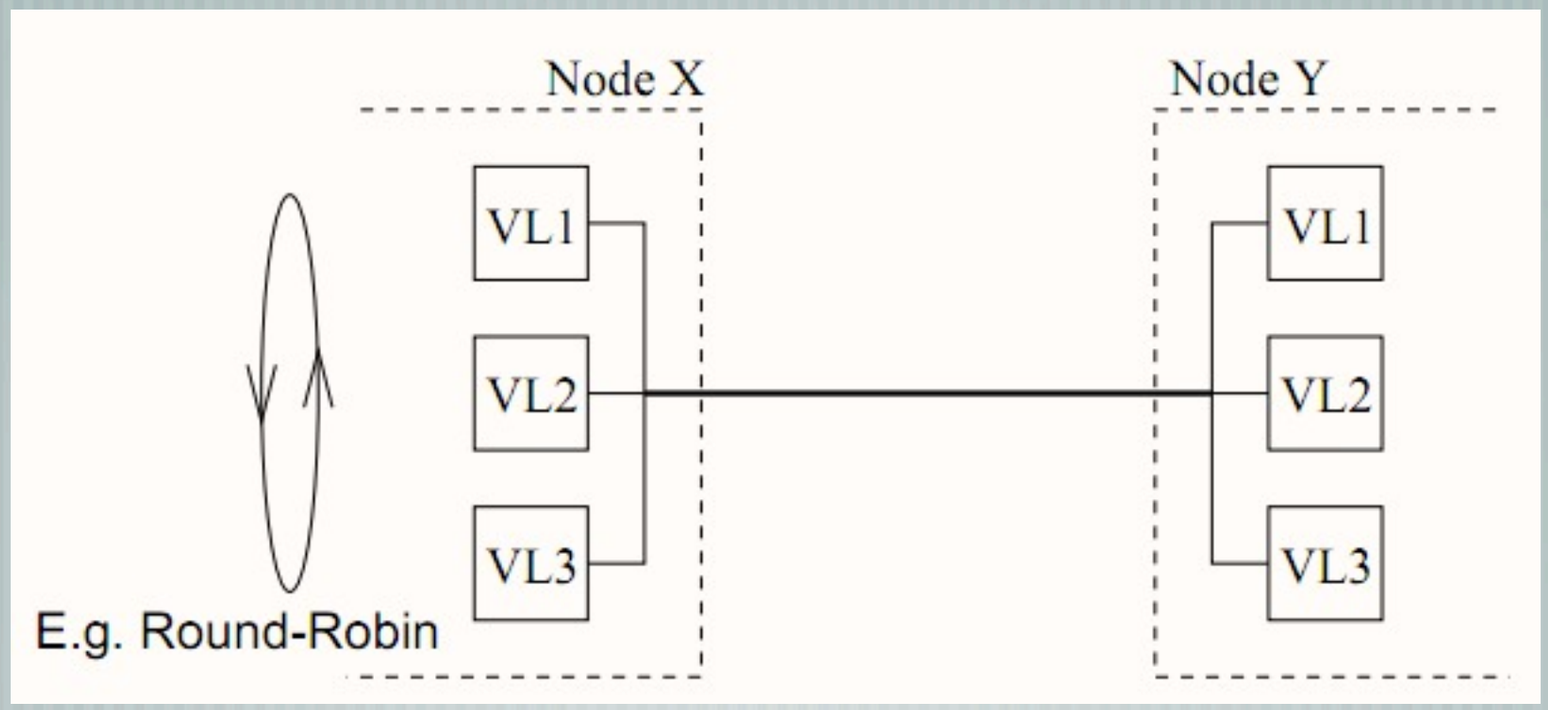
Adding two virtual channels (V3, V4)



Modified channel-dependence graph using V3 and V4

Virtual channels

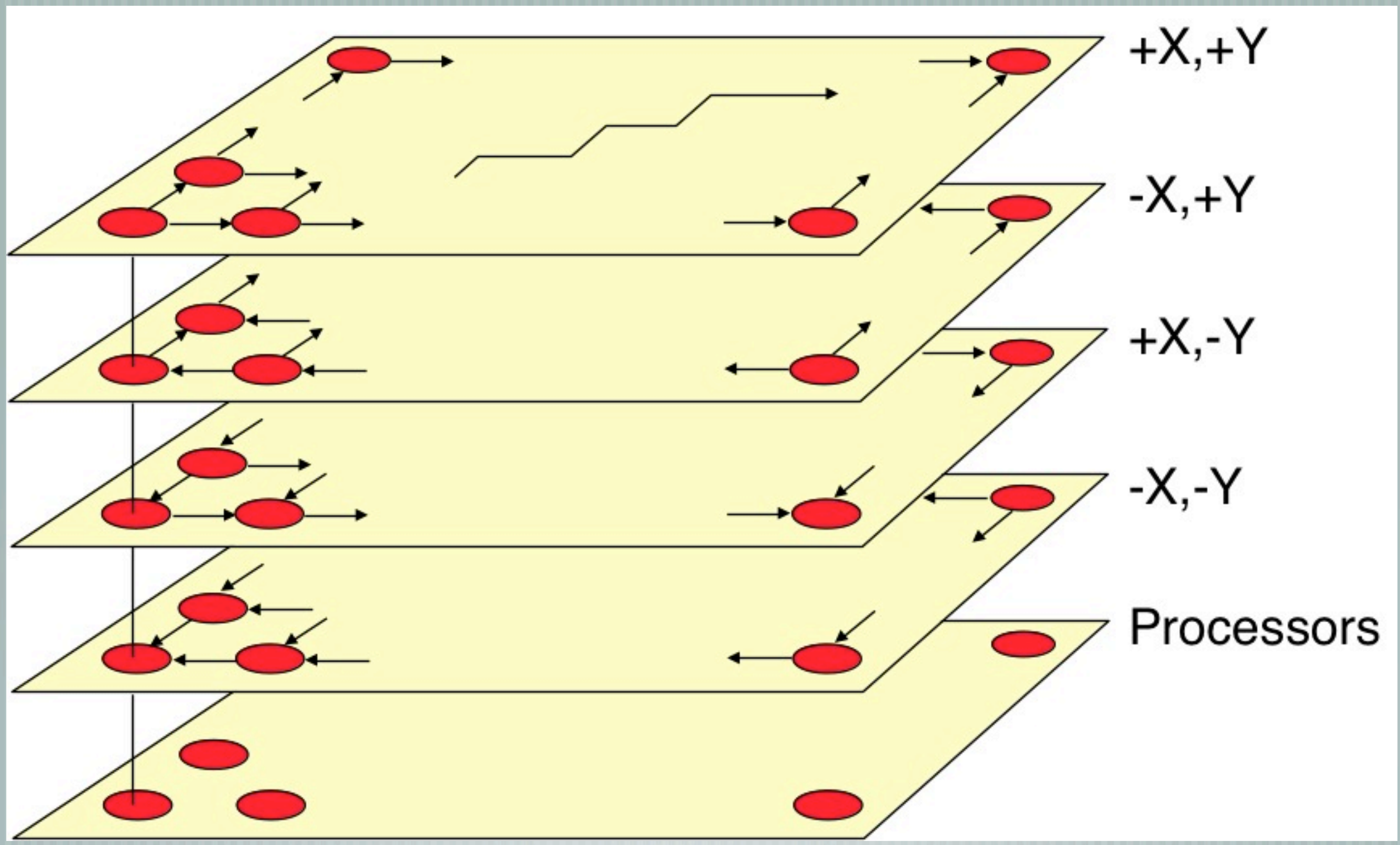
- Virtual channels are logical links between two nodes using their own buffers and multipexed over a single physical channel
- Virtual channels “break” dependency cycles



Adaptive routing with virtual networks

- Consider a 2-D Mesh Network
- One partition is that all messages are routed in one of four uni-directional grids, e.g.
  - +X,+Y +X,-Y -X,+Y -X,-Y
- Partition decided when message is constructed, based on source and destination address
- Restriction is that messages are always routed on shortest paths, they may adapt to congestion however between shortest paths in each grid

Virtual networks



Virtual channels

Advantages

- Increased network throughput
- Deadlock avoidance
- Virtual topologies
- Dedicated channels (e.g. debugging, monitoring)

Disadvantages

- Hardware cost (more buffers)
- Higher latency
- Incoming packets may be out-of-order

Summary

- Networks are not scalable
  - Either cost or cross-section may be but not both
- Networks can be designed so that both cost and latency grow slowly with scalable bisection
- Topology buffering and flow control are all important design parameters in network design
  - Buffering and virtual channels can control deadlock