



Parallel Hardware and Interconnects

Section 2.3
Bryan Mills, PhD

Spring 2017

Summary

- Pthreads
 - Simple threading (ie strassen)
 - Shared Data and Critical Sections
 - Busy-Wait
 - Locks (mutex)
 - Semaphore
 - Read/Write Locks

Supercomputer Etiquette

- Always use your credentials to login.
- Do not run parallel jobs on login nodes!!
- Do not abuse allocations.
 - Only use for class assignments
 - Never request more nodes that assignment specifies
 - Never request more than 10 minutes in your submission script.
 - Never request more than 32 cores in interactive session
 - Never request more than an hour of interactive session.
 - Always exit interactive sessions when done.

Flynn's Taxonomy

classic von Neumann

<p>SISD</p> <p>Single instruction stream Single data stream</p>	<p>(SIMD)</p> <p>Single instruction stream Multiple data stream</p>
<p>MISD</p> <p>Multiple instruction stream Single data stream</p>	<p>(MIMD)</p> <p>Multiple instruction stream Multiple data stream</p>

not covered

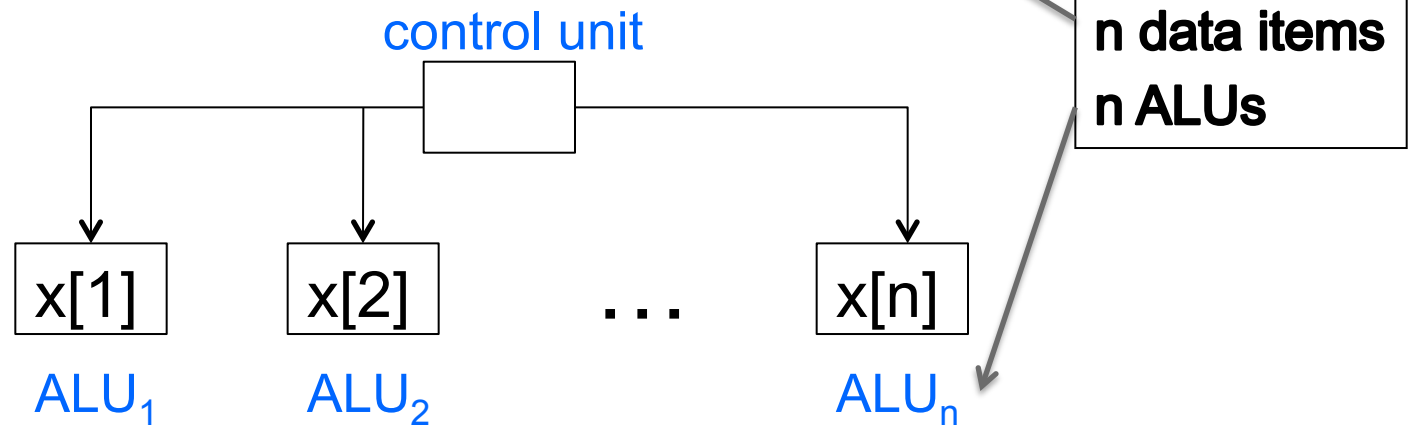
Pthreads, OpenMP, MPI

SIMD

- **Single Instruction, Multiple Data**
 - Parallelism achieved by dividing data among the processors.
- One Instruction is repeatedly applied to different data.
- Data parallelism

SIMD example


```
for (i = 0; i < n; i++)  
    x[i] += y[i];
```



SIMD

- What if we don't have as many ALUs as data items?
- Divide the work and process iteratively.
- Ex. $m = 4$ ALUs and $n = 15$ data items.

Round3	ALU ₁	ALU ₂	ALU ₃	ALU ₄
1	X[0]	X[1]	X[2]	X[3]
2	X[4]	X[5]	X[6]	X[7]
3	X[8]	X[9]	X[10]	X[11]
4	X[12]	X[13]	X[14]	



SIMD drawbacks

- All ALUs are required to execute the same instruction, or remain idle.
- In classic design, they must also operate synchronously.
- The ALUs have no instruction storage.
- Efficient for large data parallel problems, but not other types of more complex parallel problems.

Vector processors

- Example of SIMD
- Operate on arrays or vectors of data while conventional CPU's operate on individual data elements or scalars.
 - Also known as array processors
- Vector registers.
 - Capable of storing a vector of operands and operating simultaneously on their contents.
- Vectorized and pipelined functional units.
 - The same operation is applied to each element in the vector (or pairs of elements).

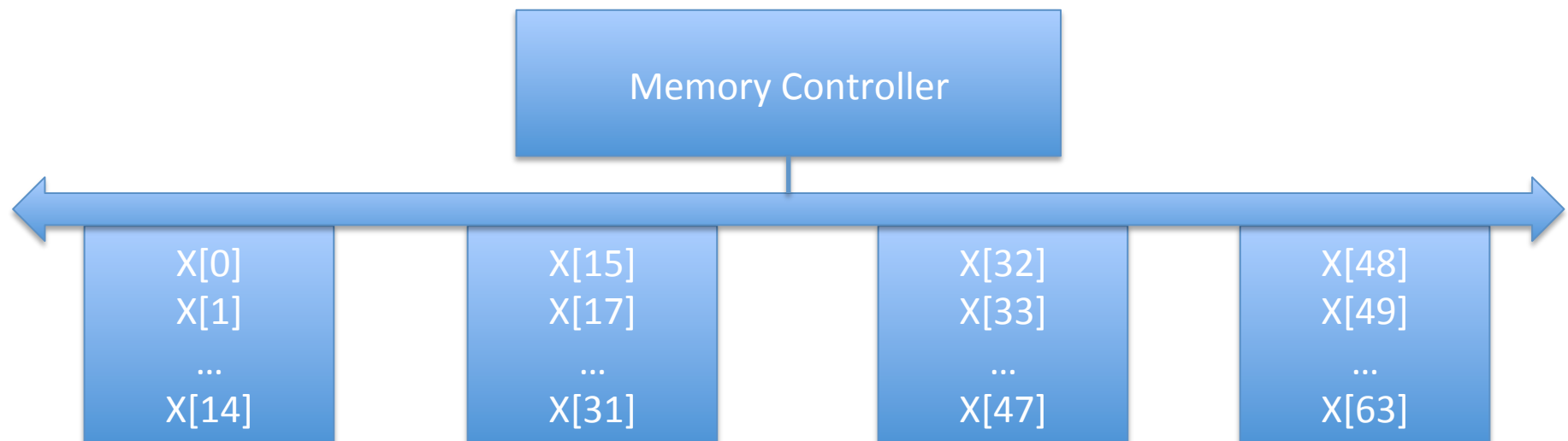
Vector processors

	0	1	2	3	4	5	6	7	8	9	10	11
x	3	7	0	1	4	0	0	4	5	3	1	0
y	2	4	2	1	8	3	9	5	5	1	2	1
	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
result x	5	11	2	2	12	3	9	9	10	4	3	1

```
for (i = 0; i < n; i++)  
    x[i] += y[i];
```

Interleaved Memory

- Multiple “banks” of memory, which can be accessed more or less independently.
- Distribute elements of a vector across multiple banks, so reduce or eliminate delay in loading/storing successive elements.
- Strided memory access and hardware scatter/gather.
 - The program accesses elements of a vector located at fixed intervals.



Vector processors

Pros

- Fast and Easy to use.
- Vectorizing compilers are good at identifying code to exploit.
- Compilers also can provide information about code that cannot be vectorized.
- High memory bandwidth.
- Uses every item in a cache line.



Cons

- They don't handle irregular data structures as well as other parallel architectures.
- A very finite limit to their ability to handle ever larger problems. ([scalability](#))

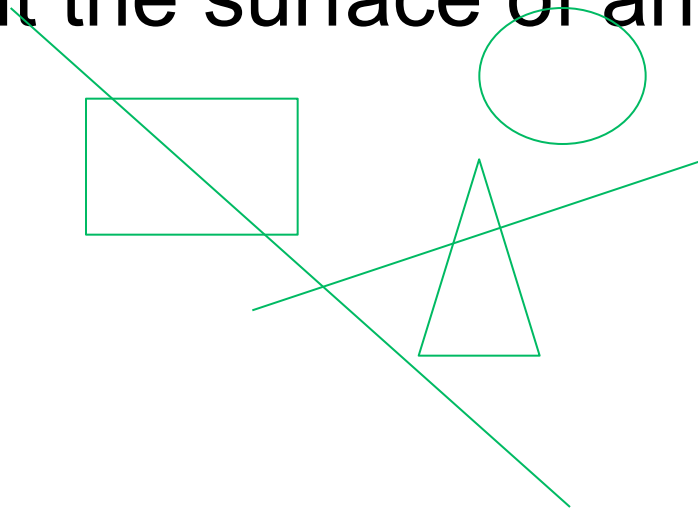


Vector Processors

- Born out of supercomputers in the 60s and became very popular in the 80s and 90s.
- Several modern CPUs make some use of vector processors or have similar techniques
 - Cell processor (IBM, Sony, Toshiba)
 - Combines one general CPU with 8 vector processors
 - Used in Playstation 3

Graphics Processing Units (GPU)

- Real time graphics application programming interfaces or API's use points, lines, and triangles to internally represent the surface of an object.



GPUs

- A graphics processing pipeline converts the internal representation into an array of pixels that can be sent to a computer screen.
- Several stages of this pipeline (called **shader functions**) are programmable.
 - Typically just a few lines of C code.



GPUs

- Shader functions are also implicitly parallel, since they can be applied to multiple elements in the graphics stream.
- GPU' s can often optimize performance by using SIMD parallelism.
- The current generation of GPU' s use SIMD parallelism.
 - Although they are not pure SIMD systems.

Flynn's Taxonomy

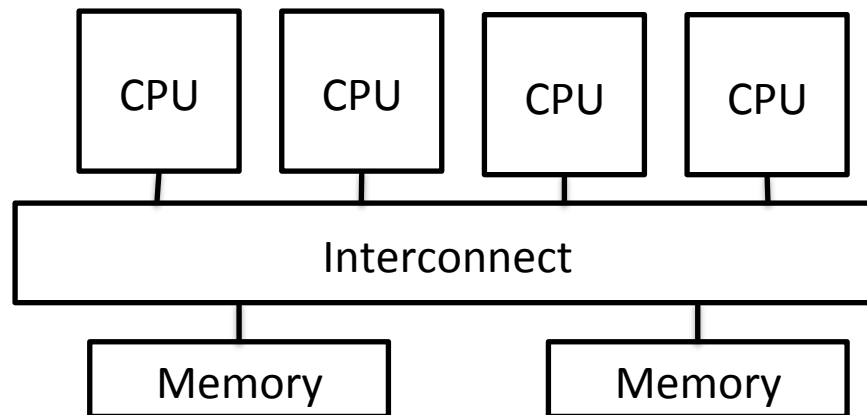
<i>classic von Neumann</i> SISD Single instruction stream Single data stream	<i>Vector (kinda GPUs)</i> (SIMD) Single instruction stream Multiple data stream
MISD Multiple instruction stream Single data stream <i>not covered</i>	(MIMD) Multiple instruction stream Multiple data stream <i>Pthreads, OpenMP, MPI</i>

MIMD

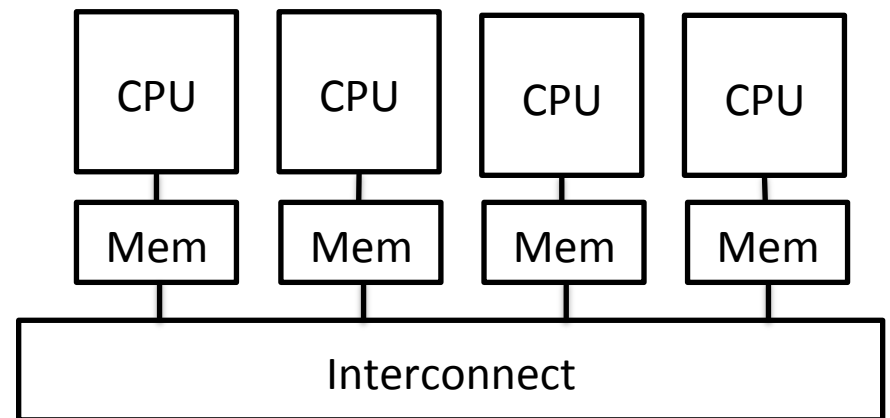
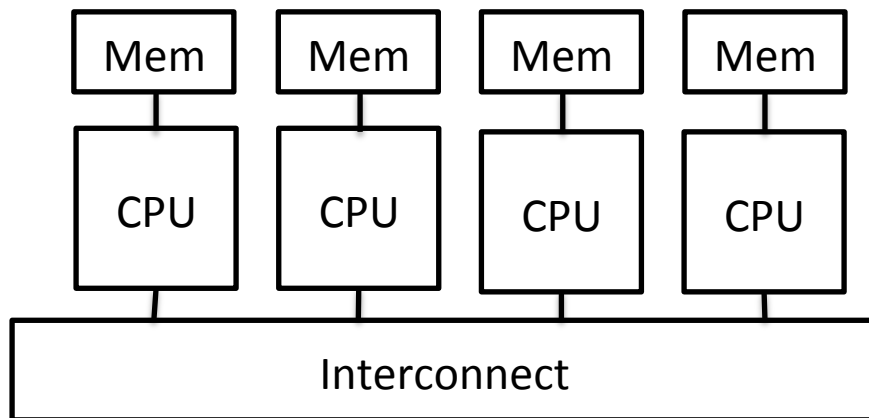
- **Multiple Instruction, Multiple Data**
 - Supports multiple simultaneous instruction streams operating on multiple data streams.
 - Typically consist of a collection of fully independent processing units or cores, each of which has its own control unit and its own ALU.
- Different types of MIMDs depending on memory architecture and address space

Memory Architecture

Pthreads are
an example



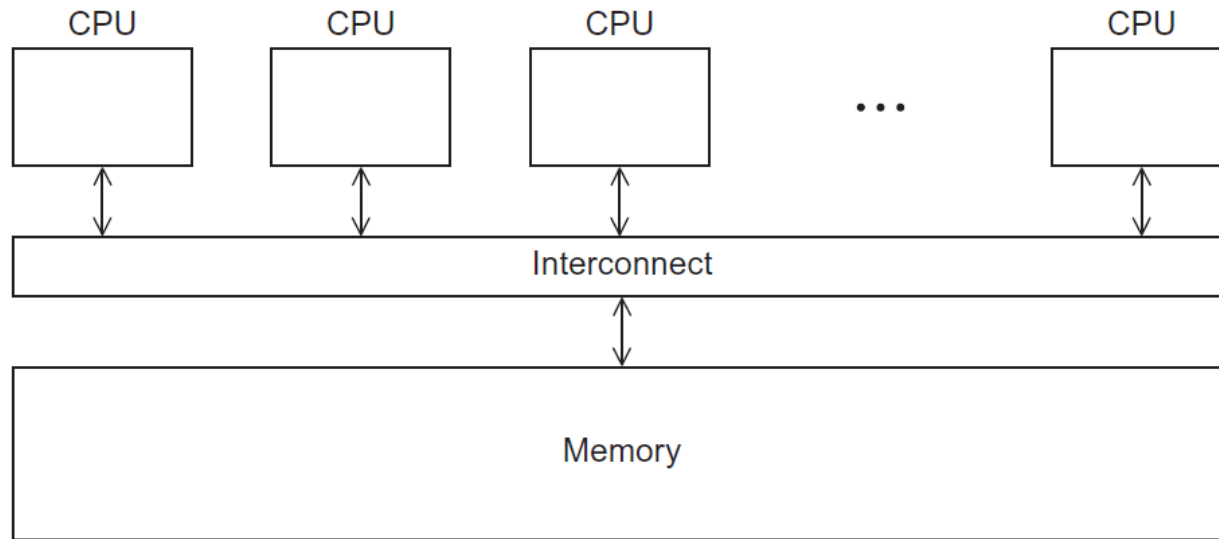
Global, shared memory architecture (Symmetric Multi-Processors – SMP)



Distributed memory architecture

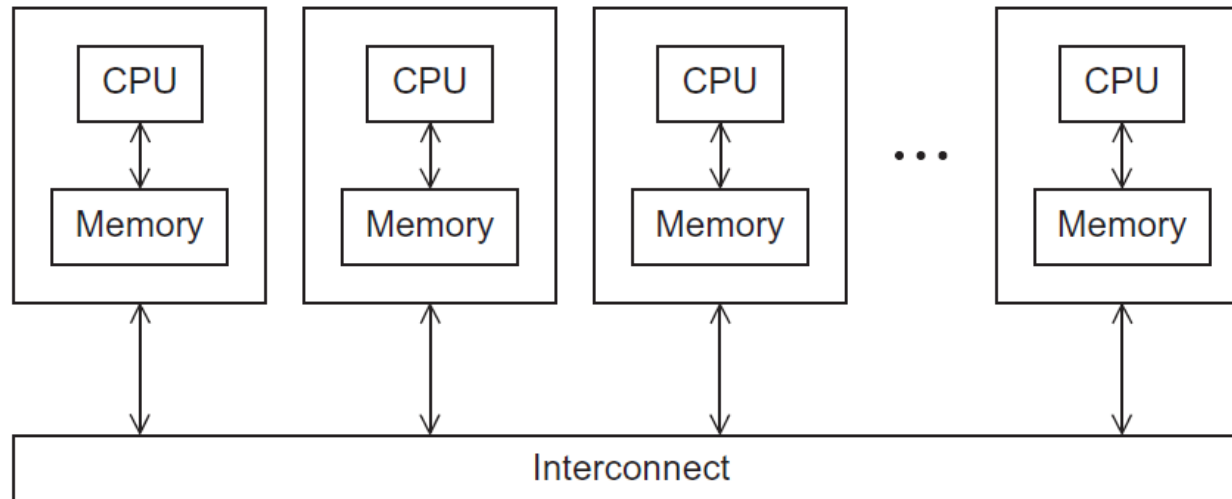
MPI

Shared Memory Addresses



- Most widely available shared memory systems use one or more multicore processors.
 - (multiple CPU's or cores on a single chip)
- Each CPU accesses the memory using the same address
 - Address XXXX on CPU₁ is same on CPU₂

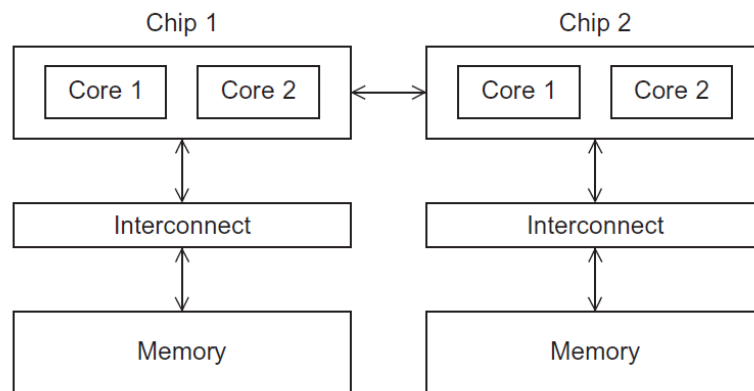
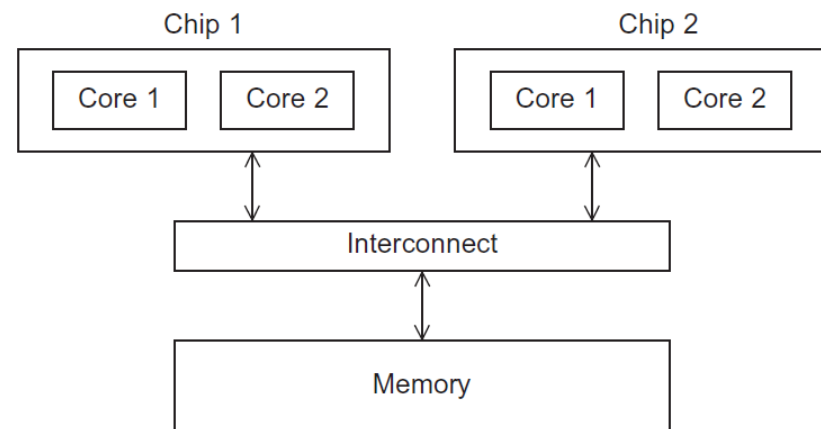
Distributed Memory Addresses



- Clusters
 - A collection of commodity systems
 - Connected by a commodity interconnection network.
- Nodes of a cluster are individual computations units joined by a communication network.
- Requires Message Passing between individual computational units.

Multicore Memory Access

- Uniform Memory Access (UMA)
 - Time to access all the memory locations will be the same for all the cores.



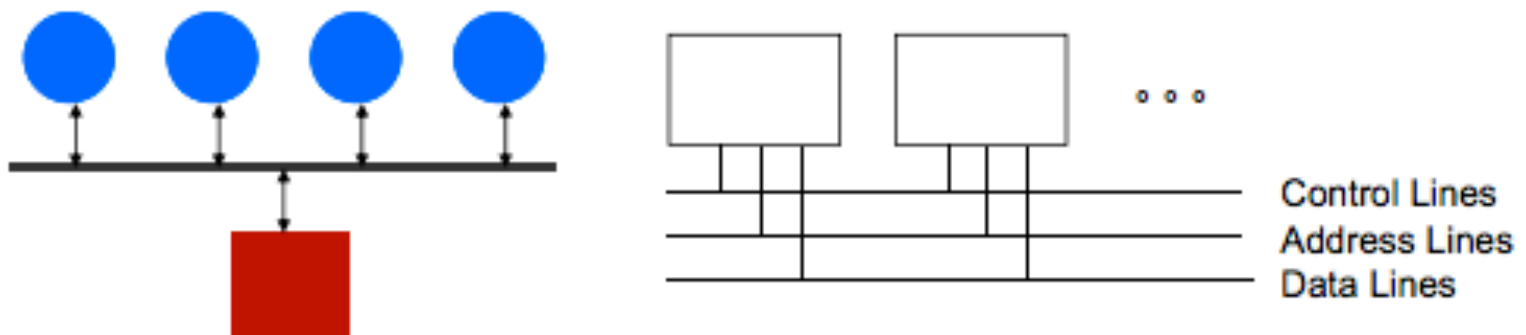
- Non-uniform Memory Access (NUMA)
 - A memory location a core is directly connected to can be accessed faster than a memory location that must be accessed through another chip.

Interconnection networks

- Affects performance of both distributed and shared memory systems.
- Two categories:
 - Shared memory interconnects
 - Distributed memory interconnects

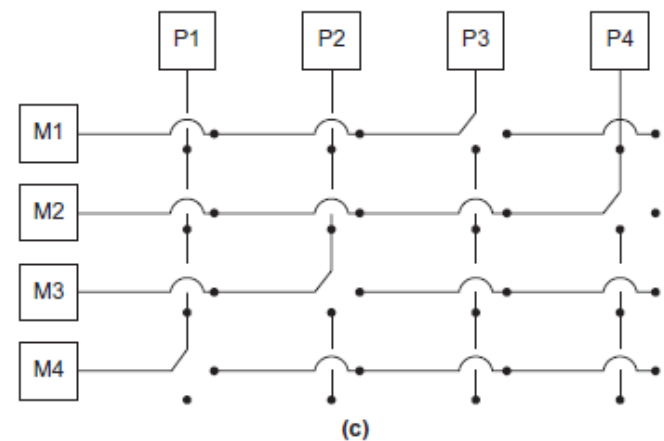
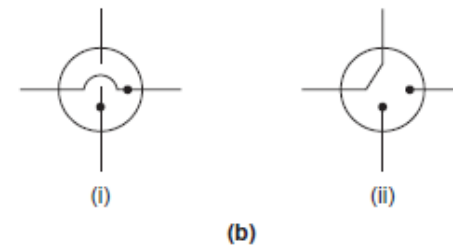
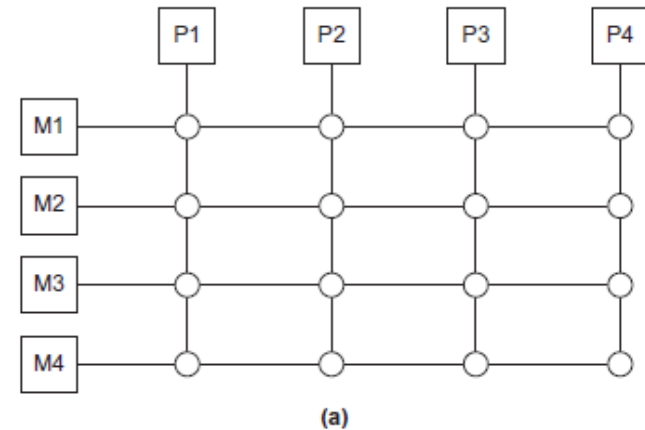
Bus Interconnect

- A collection of parallel communication wires together with some hardware that controls access to the bus.
- Communication wires are shared by the devices that are connected to it.
- As the number of devices connected to the bus increases, contention for use of the bus increases, and performance decreases.



Crossbar

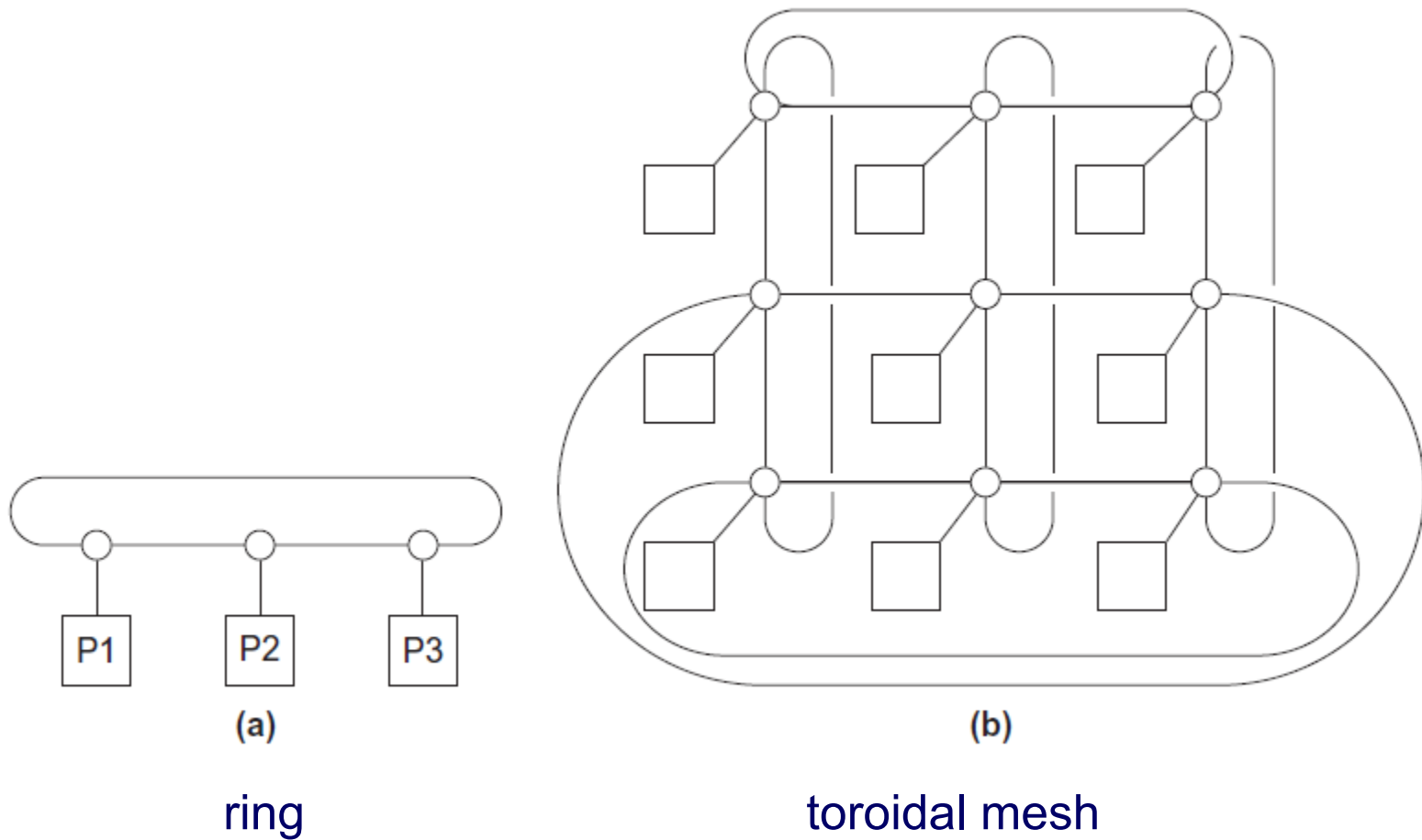
- Allows simultaneous communication among different devices.
- Faster than buses.
- But the cost of the switches and links is relatively high.
- Simultaneous memory accesses by the processors



Distributed memory interconnects

- Two groups
 - Direct interconnect
 - Each switch is directly connected to a processor memory pair, and the switches are connected to each other.
 - Indirect interconnect
 - Switches may not be directly connected to a processor.

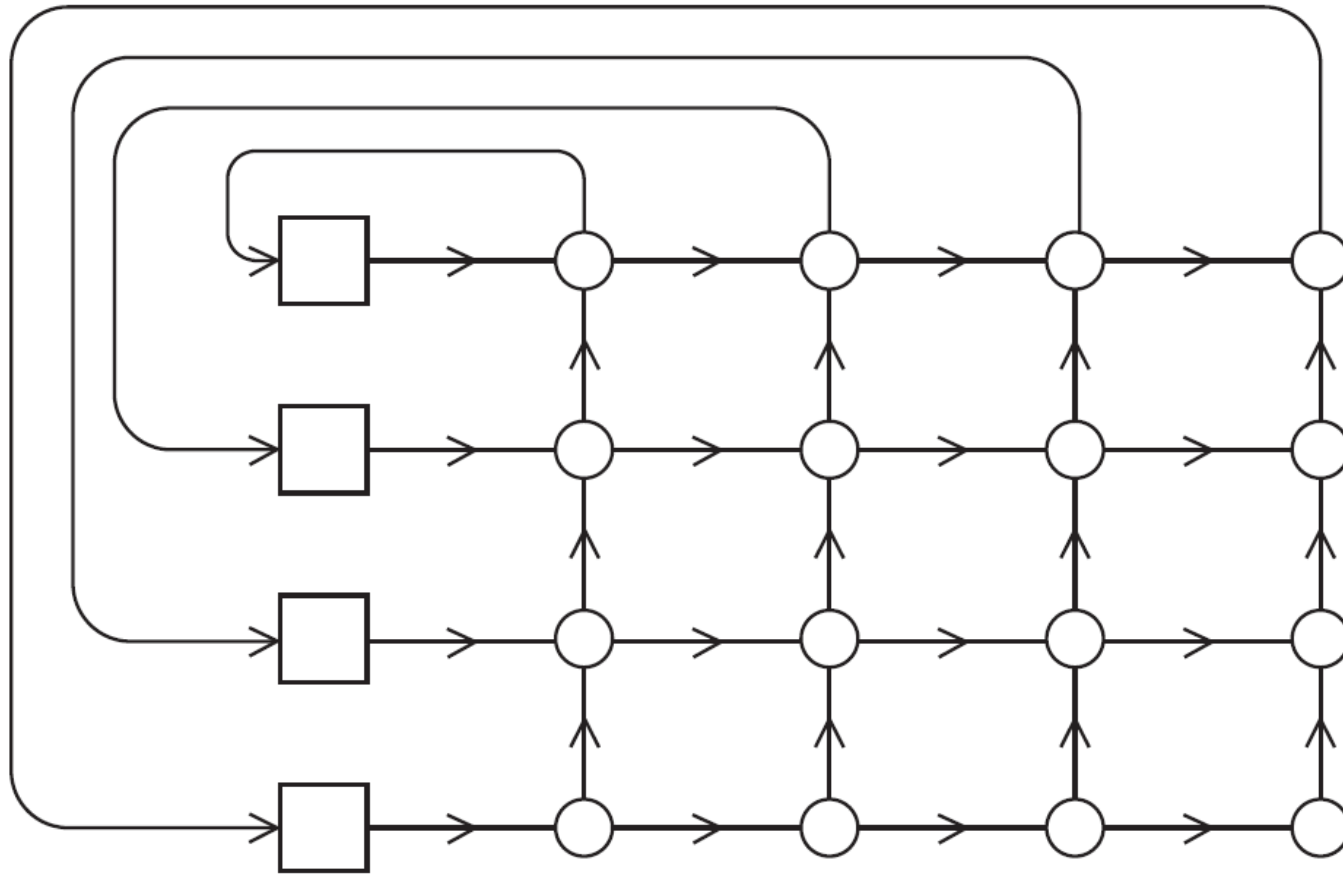
Direct interconnect



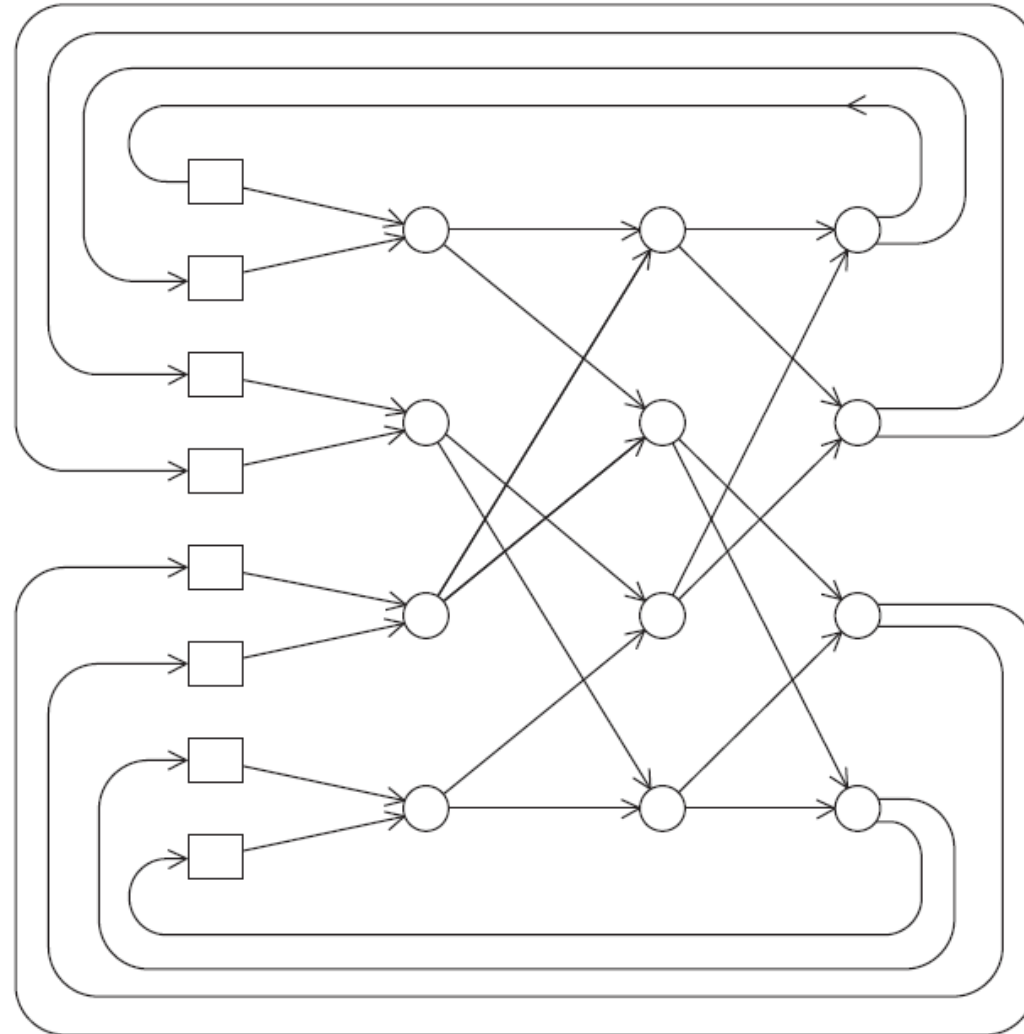
Indirect interconnects

- Simple examples of indirect networks:
 - Crossbar
 - Omega network
- Often shown with unidirectional links and a collection of processors, each of which has an outgoing and an incoming link, and a switching network.

Crossbar interconnect for distributed memory

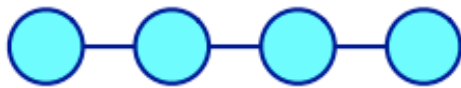


An omega network



Network Topologies

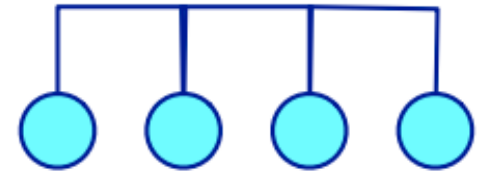
1D-mesh



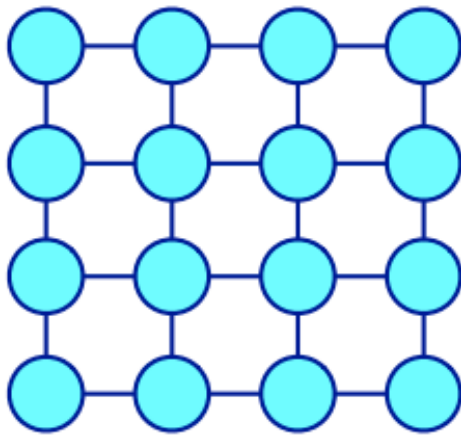
1D-torus (ring)



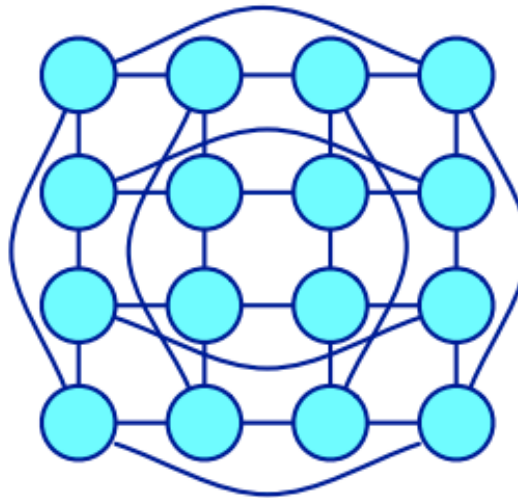
bus



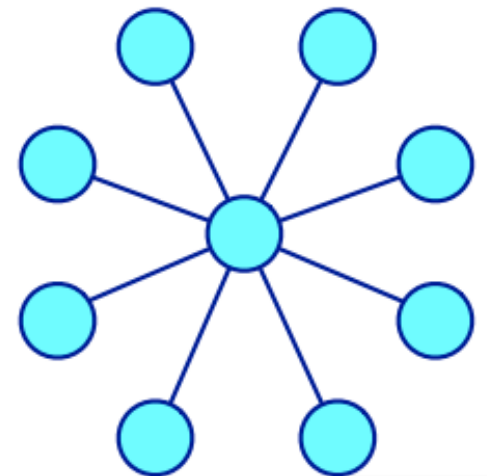
2D-mesh



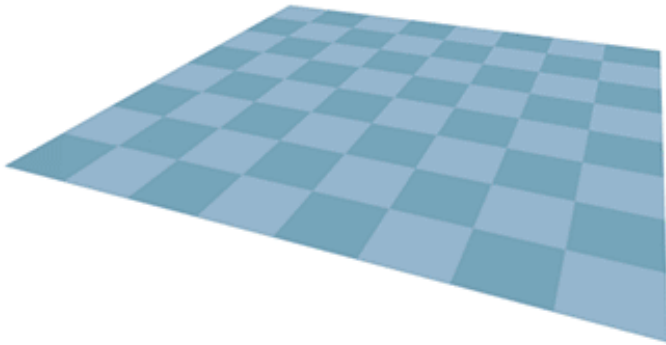
2D-torus



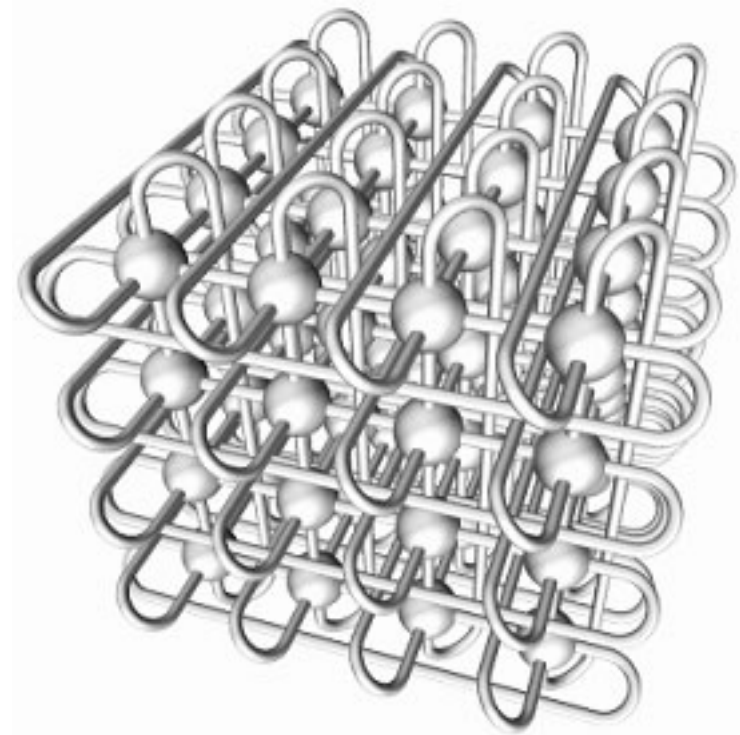
star



Torus



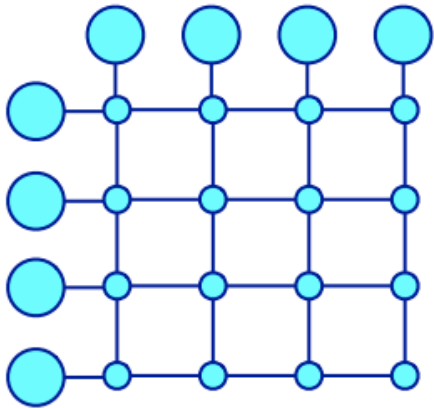
2D Animation [here](#)



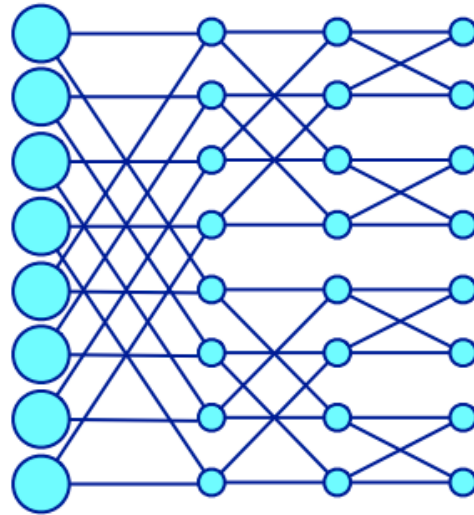
3d Torus

Network Topologies

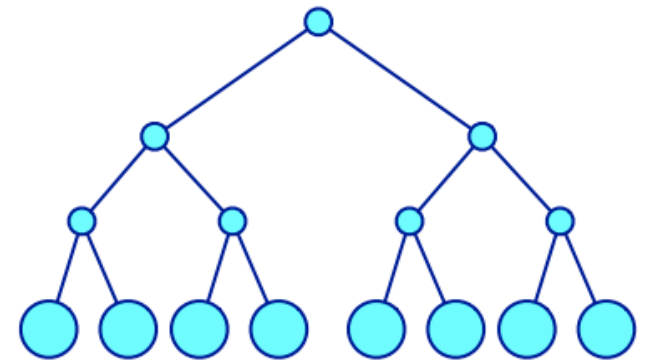
crossbar



butterfly



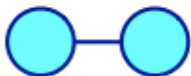
tree



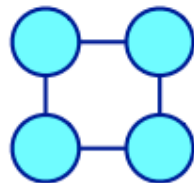
0D-cube



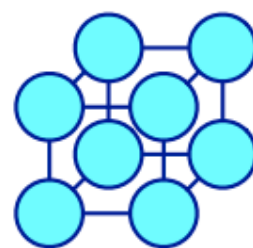
1D-cube



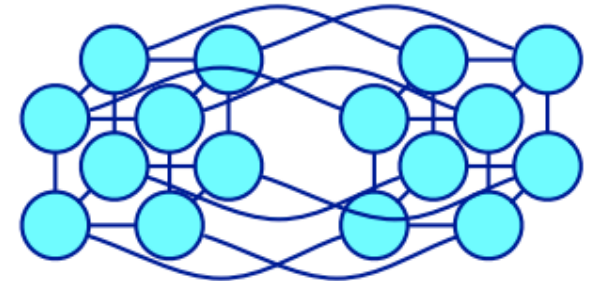
2D-cube



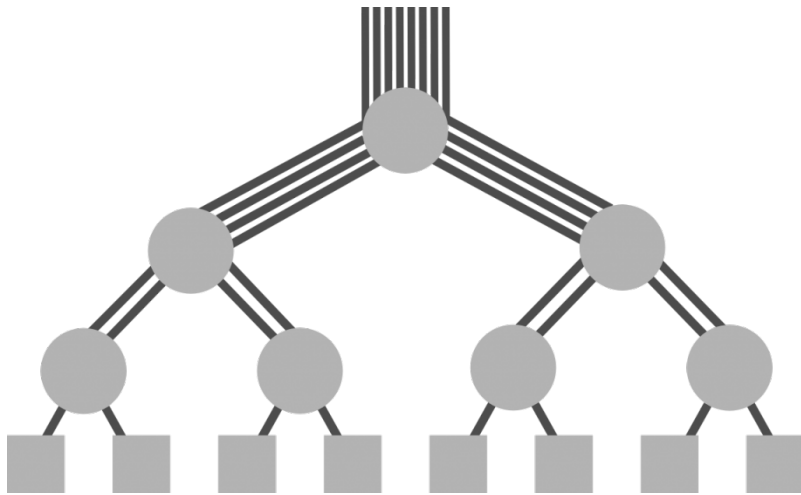
3D-cube



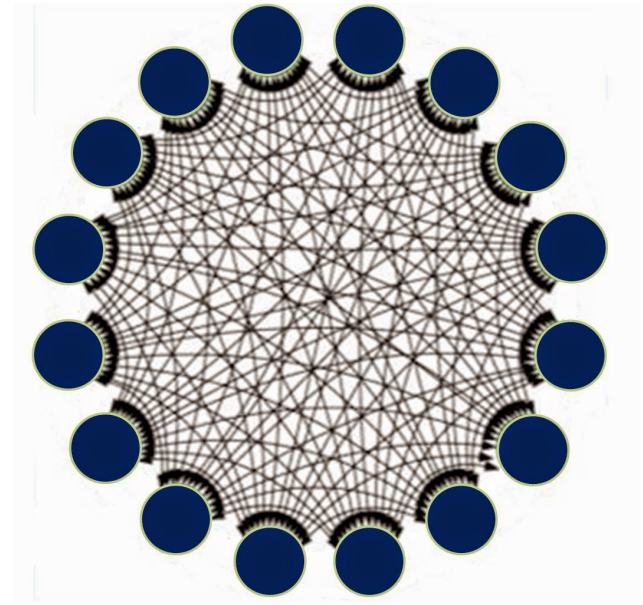
4D-cube



Network Topologies



Fat Tree



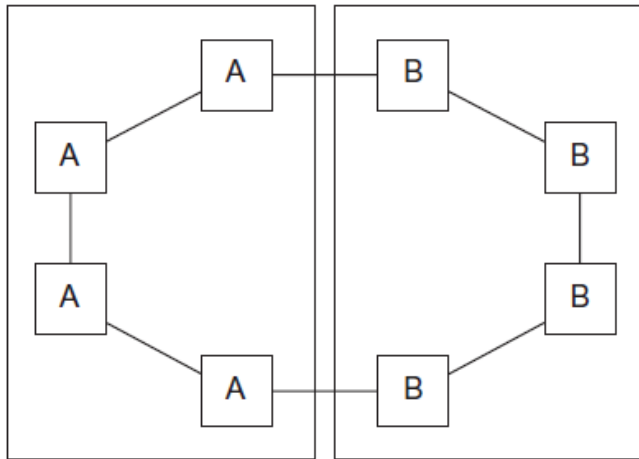
Dragon Fly

Interconnect Properties

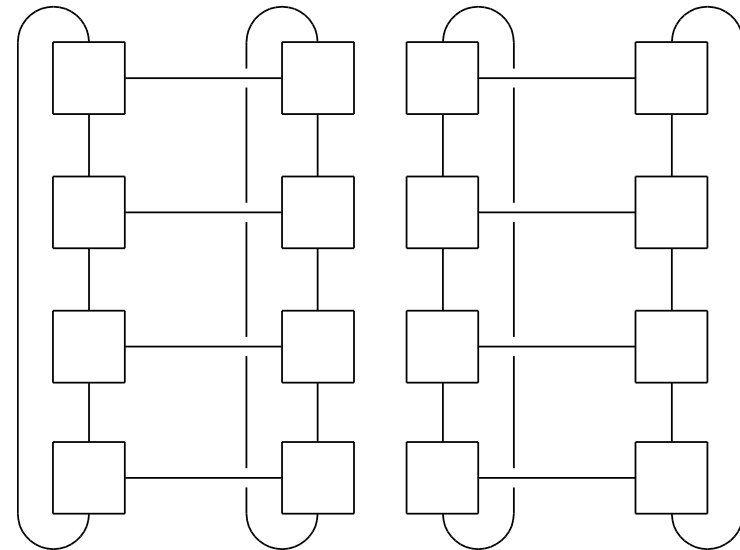
- The following characteristics of a network affect its performance and its feasibility.
 - **Degree**: maximum number of edges incident on any vertex. It determines the number of communication ports per node.
 - **Average distance**: average length of shortest paths between pairs of nodes.
 - **Diameter**: maximum distance between any pair of nodes. It determines the maximum communication delay between nodes.
 - **Bisection width**: smallest number of edges whose removal splits network into two disconnected, equally-sized parts. It determines the ability to support simultaneous global communication.
 - **Edge length**: maximum physical length of any wire. It may be constant or variable as number of nodes changes.

The bisections bandwidth

- A measure of network quality.
- Instead of counting the number of links joining the halves, it sums the bandwidth of the links.

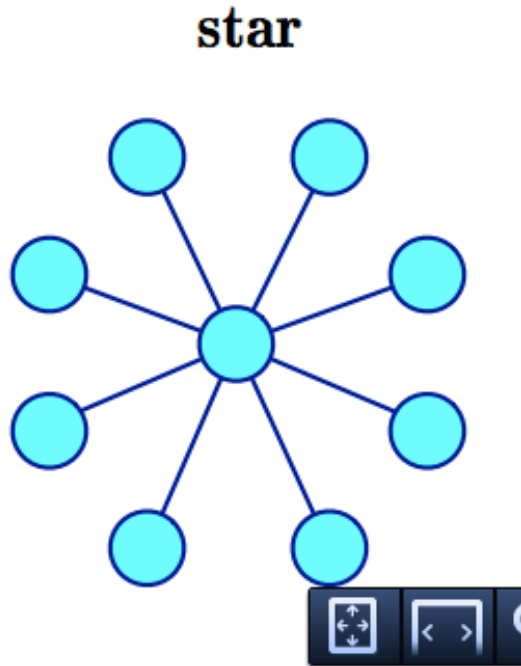


For a ring



For a toroidal mesh (torus)

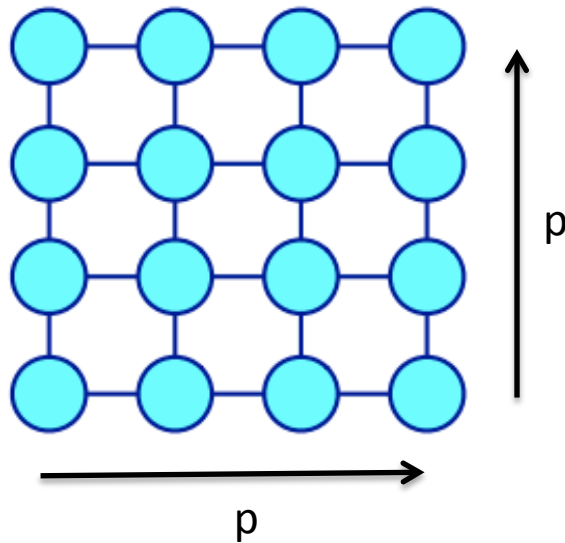
Example



- Nodes?
 - $p + 1$
- Degree?
 - p
- Diameter?
 - 2
- Bisection Width
 - 1
- Edge Length
 - Constant

Example

2D-mesh



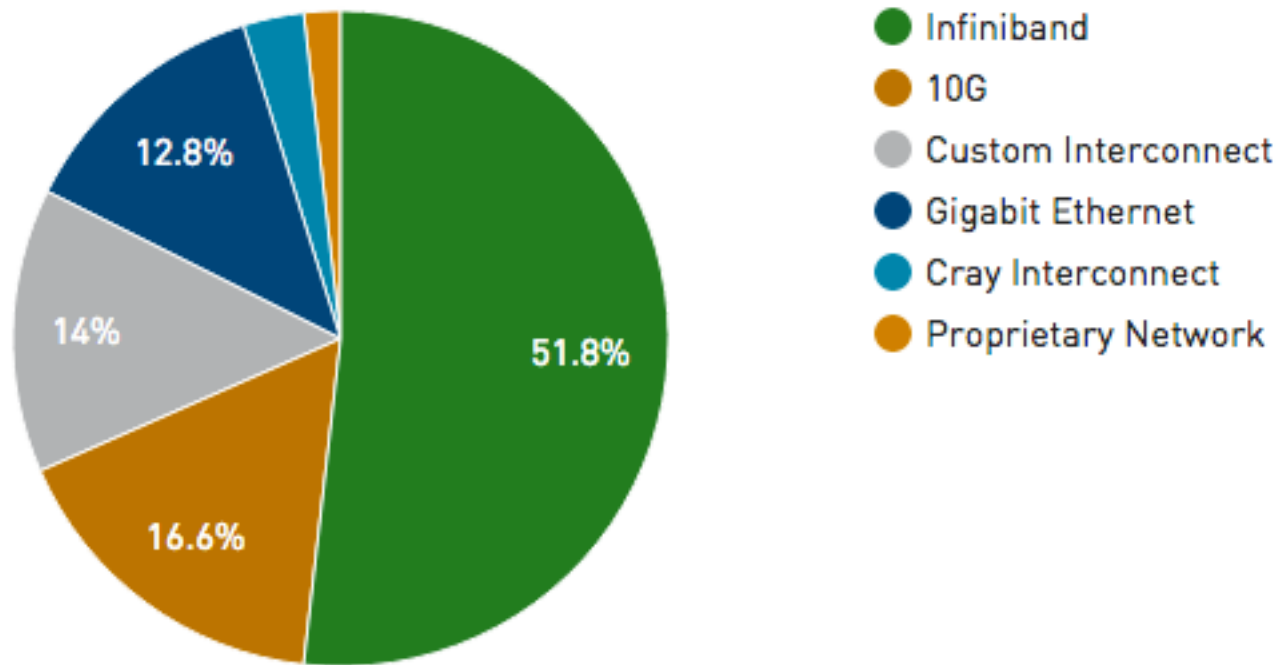
- Nodes?
 - p^2
- Degree?
 - 4
- Diameter?
 - $2(p-1)$
- Bisection Width
 - p
- Edge Length
 - Constant

Top 500

Rank	Machine	Location	Network
1	Sunway	China	Bi-section Tree
2	Tianhe-2 (MilkyWay)	China	Fat Tree
3	Titan	US - ORNL	3D-torus
4	Sequoia	US - LLNL	3D-torus
5	K Computer	Japan	6D-torus
6	Mira	US - Argonne	3D-torus
7	Piz Daint	Switzerland	Dragonfly
8	Shaheen II	Saudi Arabia	3D-torus
9	Stampede	US - Austin	Fat Tree
10	JUQUEEN	Germany	3D-torus
11	Vulcan	US - LLNL	3D-torus
12	CS-Storm	US - LANL	Fat Tree

Top 500

Interconnect Family System Share

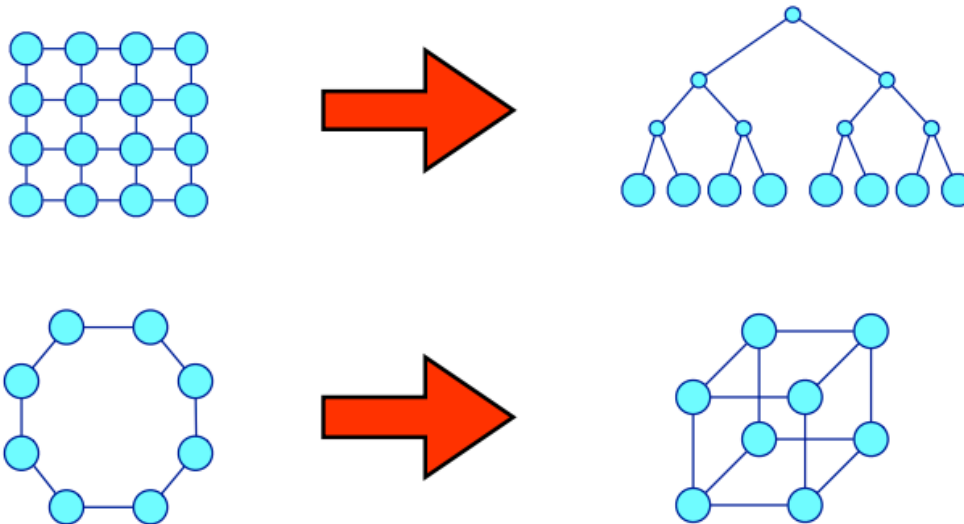


Multiple Interconnects

- Today's systems commonly have multiple interconnects
 - Intrepid (system at Argonne) has 5 interconnects
 - High Performance
 - Collective
 - Barrier
 - I/O
 - Service

Topology Mapping

- Find a mapping from one topology to a another topology
 - Often done in software and mapped onto a physical network graph.



- The goal is to retain the distances found in the software mapping to the one in the physical hardware.

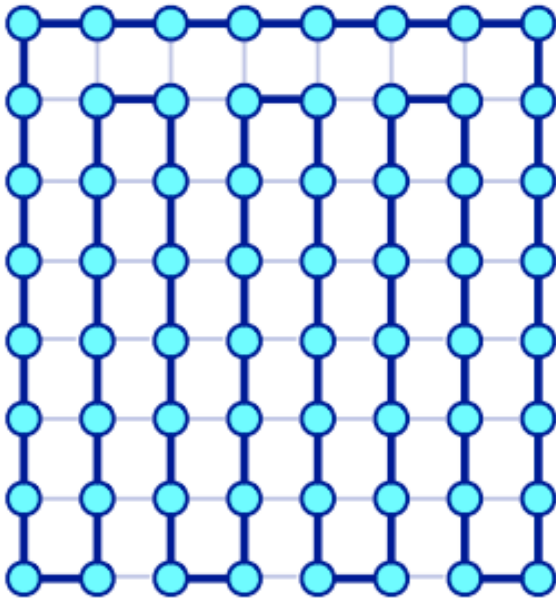
Graph Embedding Problem

- Function ϕ maps nodes in the source graph $G_s(V_s, E_s)$ to nodes in the target graph $G_t(V_t, E_t)$; $\phi: V_s \rightarrow V_t$
- An edge in G_s is mapped to a path in G_t
- **Load**: maximum number of nodes V_t mapped to the same node in V_s
- **Congestion**: maximum number of edges in E_s mapped to paths containing the same edge in E_t
- **Dilation**: maximum distance between any two nodes $\phi(u)$ and $\phi(v)$, such that $\langle \phi(u), \phi(v) \rangle$ is in E_s

Graph Embedding

- Finding optimal mapping ϕ is NP-complete; heuristic algorithms are used to find a good mapping.
- **Load** has to be as uniform as possible, to balance work across processors.
- Decreasing **congestion** improves the use of available bandwidth of network links.
- Decreasing **dilation** keeps nearest-neighbor communications in source graph as short as possible in target graph.
- The best embedding has load, congestion, and dilation equal to 1; it may not be always possible.

Graph Embedding Examples

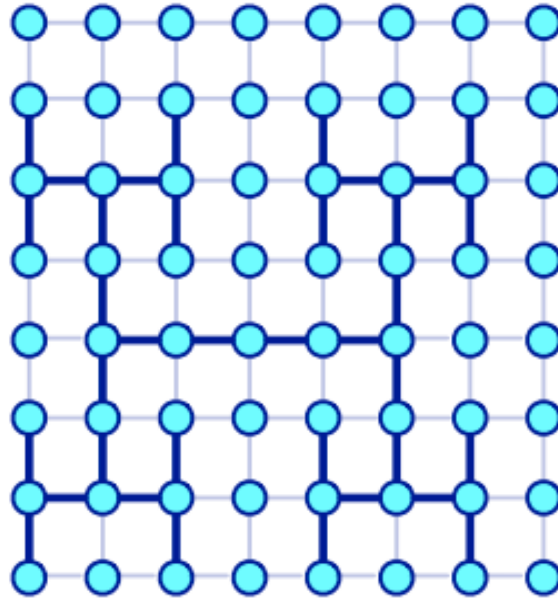


ring into 2D-mesh

load=1

congestion=1

dilation=1

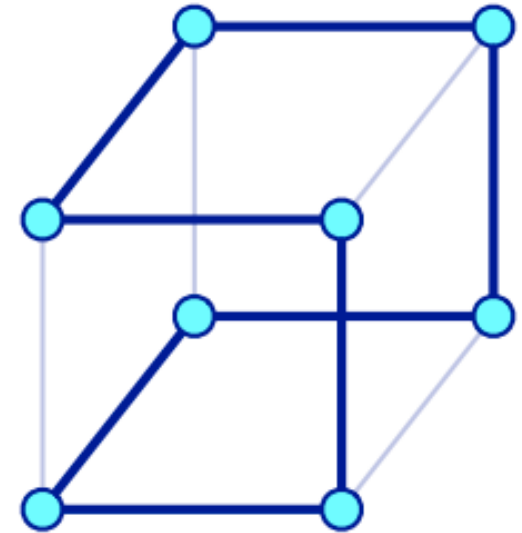


binary tree into 2D-mesh

load=1

congestion=1

dilation= $2^{\lceil (p-1)/2 \rceil - 1}$



ring into hypercube

load=1

congestion=1

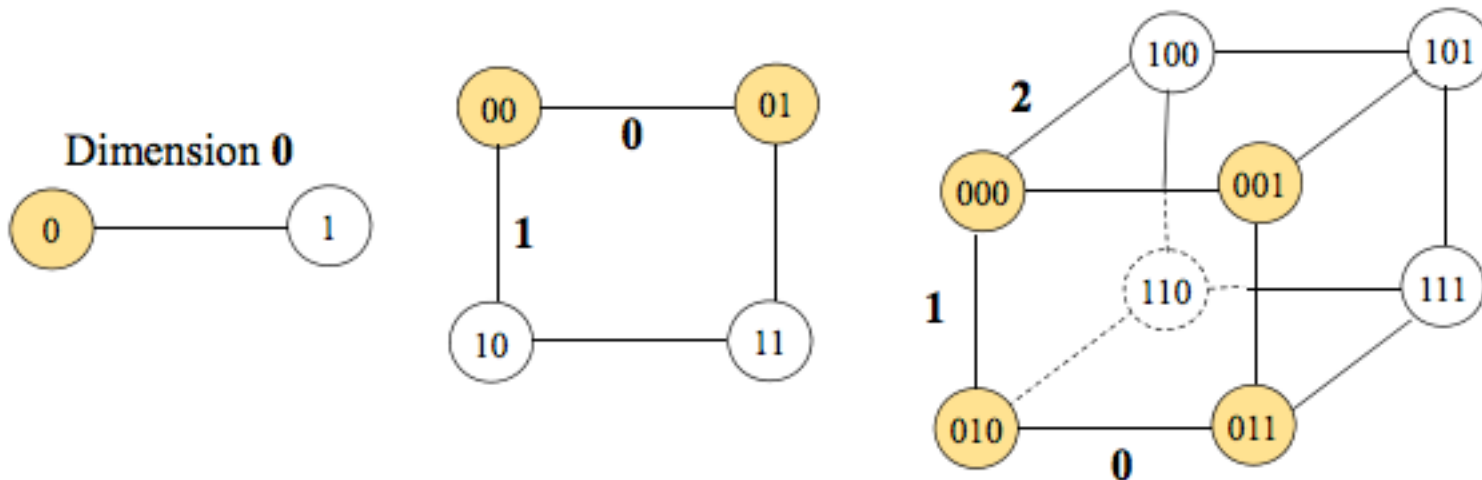
dilation=1

Graph Embedding Example

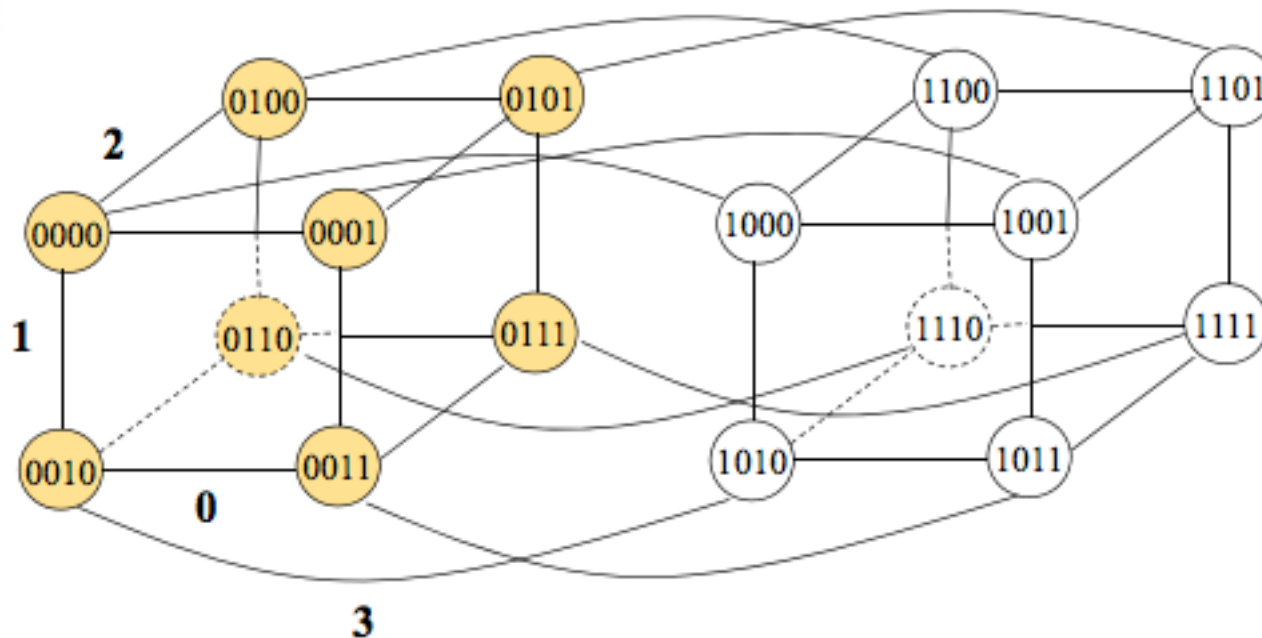
- Find a graph embedding for a star topology with 9 nodes into a 2D-mesh.
 - What is the **congestion** and **dilation**?

Hypercubes

- Connection with low diameter and a large bisection
- A hypercube of dimension d is built with two hypercubes of dimension $(d-1)$ [[animation](#)]



Hypercubes



- For a hypercube of dimension d , calculate
 - Number of nodes = 2^d
 - Degree
 - Diameter
 - Bisection
 - Edge Width

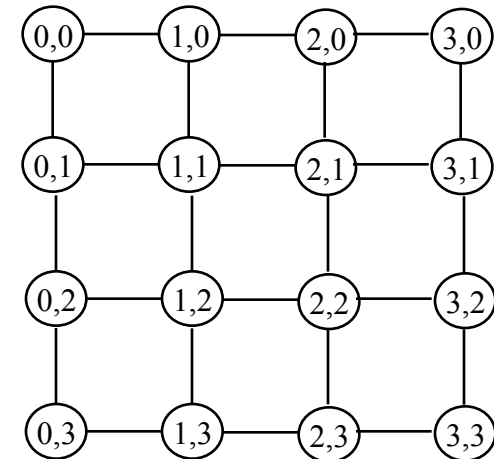
Routing

- Mechanism to send messages between nodes that are not directly connected.
- Properties of message routing algorithms:
 - *Minimal* or *non-minimal*, depending on whether shortest path is always taken.
 - Static or dynamic, depending on whether same path between a pair of nodes is always taken.
 - *Deterministic* or *randomized*, depending on whether path is chosen systematically or randomly.
 - *Circuit-switched* or *packet-switched*, depending on whether entire message goes along reserved path or is transferred in segments that may not all take the same path.

Routing messages in 2D mesh networks

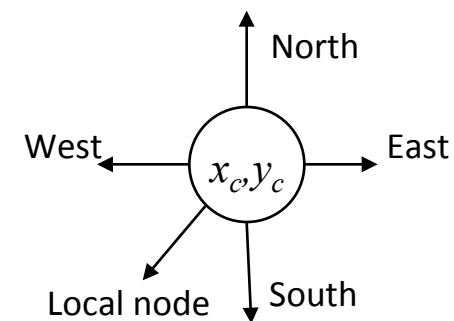
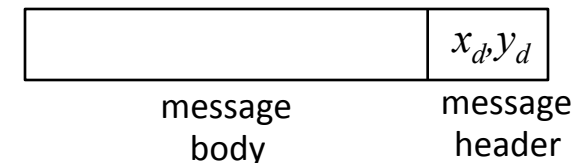
The problem:

- Assume that each switch in an $n \times n$ mesh is labeled by (x, y) , where $0 < x < n-1$ and $0 < y < n-1$.
- Assume also that each message has a header which contains the address, (x_d, y_d) , of its destination.
- A routing algorithm determines at any intermediate node, (x_c, y_c) , where to send the message next.



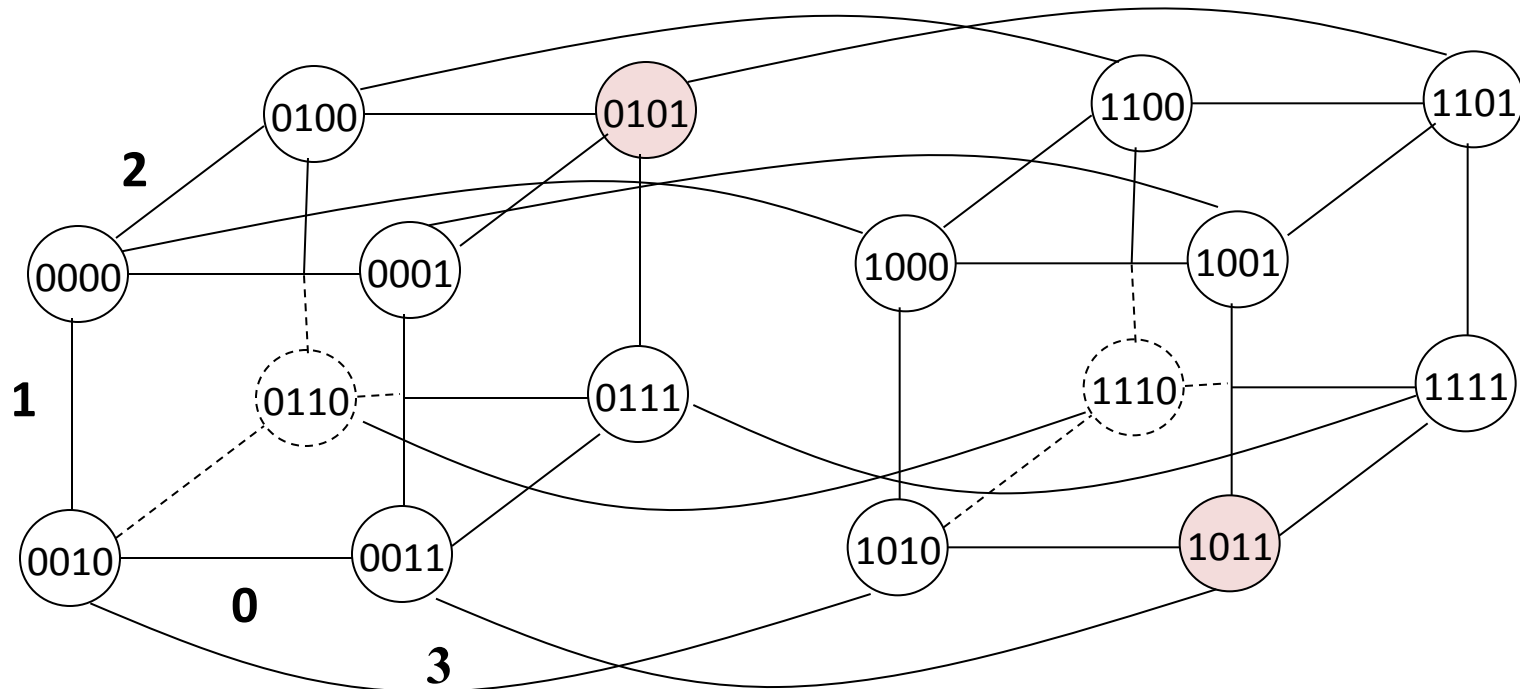
X-Y routing:

- 1) If $x_c < x_d$ then send the message to the East port
elseif $x_c > x_d$ then send it to the West port
- 2) If $y_c < y_d$ then send the message to the North port
elseif $y_c > y_d$ then send it to the South port
- 3) Deliver the message to the local node



Messages travel along the x-direction
before traveling along the y-direction

Dimension-order routing



Current Node = $CN = C_d, C_{(d-1)}, \dots, C_0$ (bit vector)

Destination Node = $DN = D_d, D_{(d-1)}, \dots, D_0$

- If $CN == DN$, keep the message
- Else { Find the largest k such that $d_k \neq c_k$;

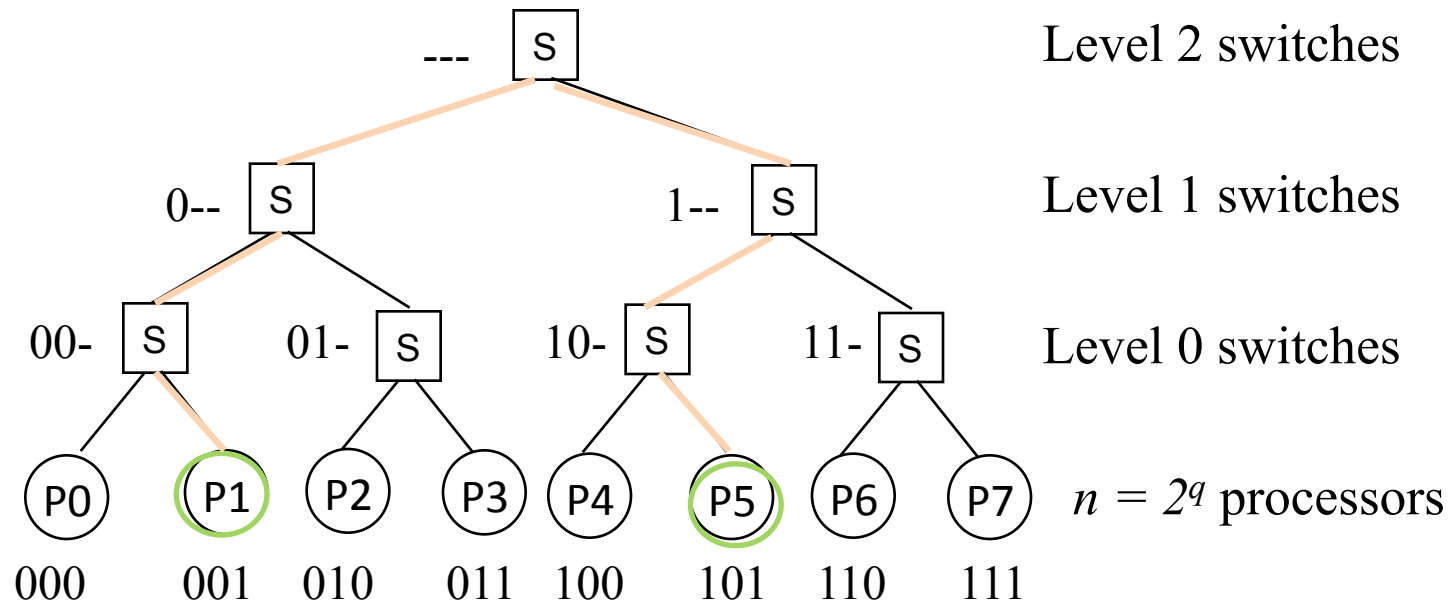
Send the message to the neighbor across dimension k }

Hypercube Communication

- Example of using hypercube for all-to-all communication.
 - Ex: sum numbers and share with all nodes

```
Hypercube(myid, input, logp, output) {  
    state = input;  
    for i = 0 to logp - 1 {  
        dest = myid XOR 2^i  
        send state to dest  
        rcv message from dest  
        state = Add(input, message)  
    }  
    output = state  
}
```

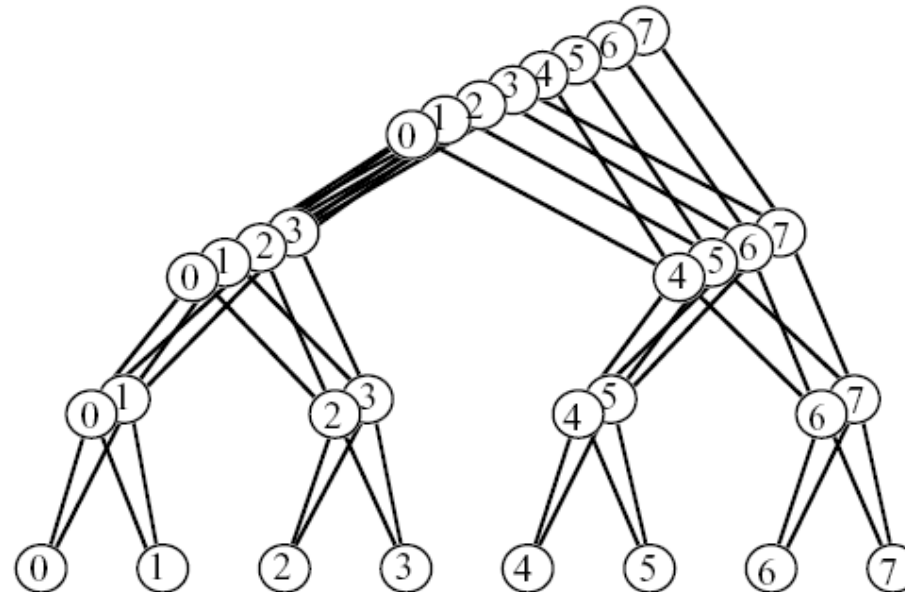
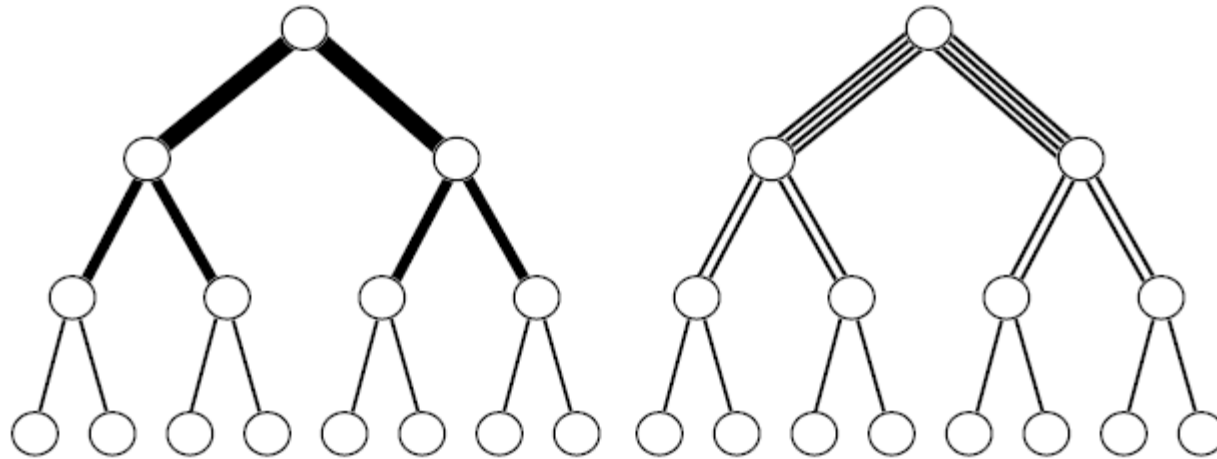
Tree Network topology



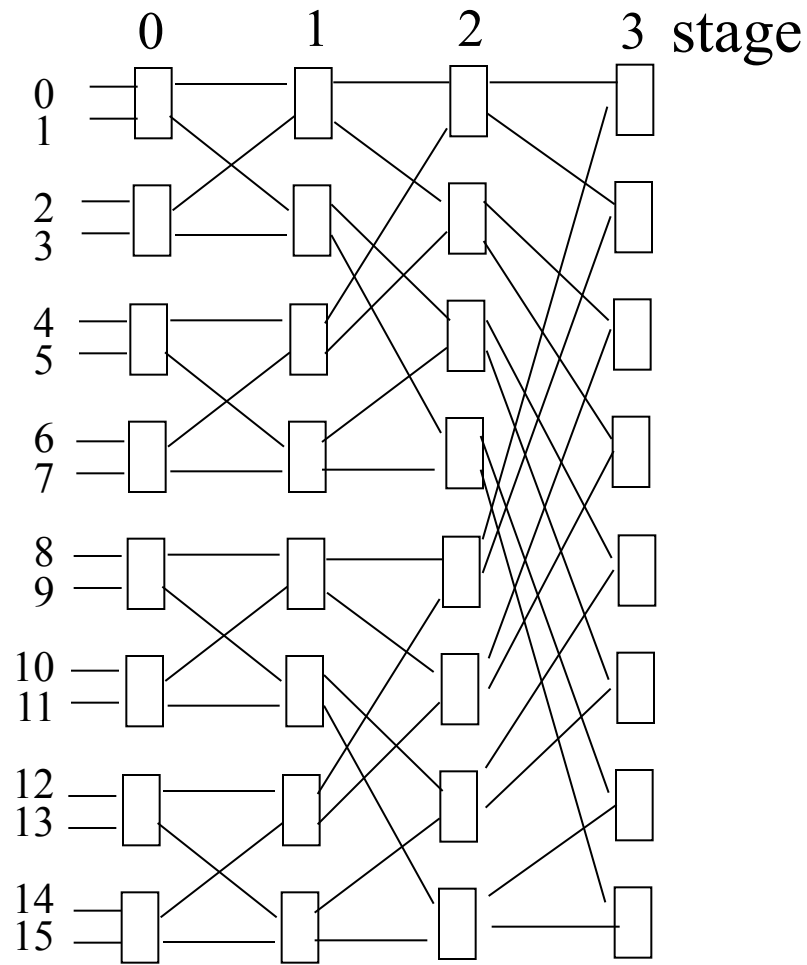
- The route between a source node, s_{q-1}, \dots, s_0 , and a destination node, d_{q-1}, \dots, d_0 , can be expressed as a sequence of up moves (U) followed by a sequence of right (R) and left (L) moves.
- Example: the route between 001 and 101 is UUURLR
- What is the bisection width of a tree with n leave nodes?

Fat tree networks

Eliminates the bisection bottleneck of a binary tree

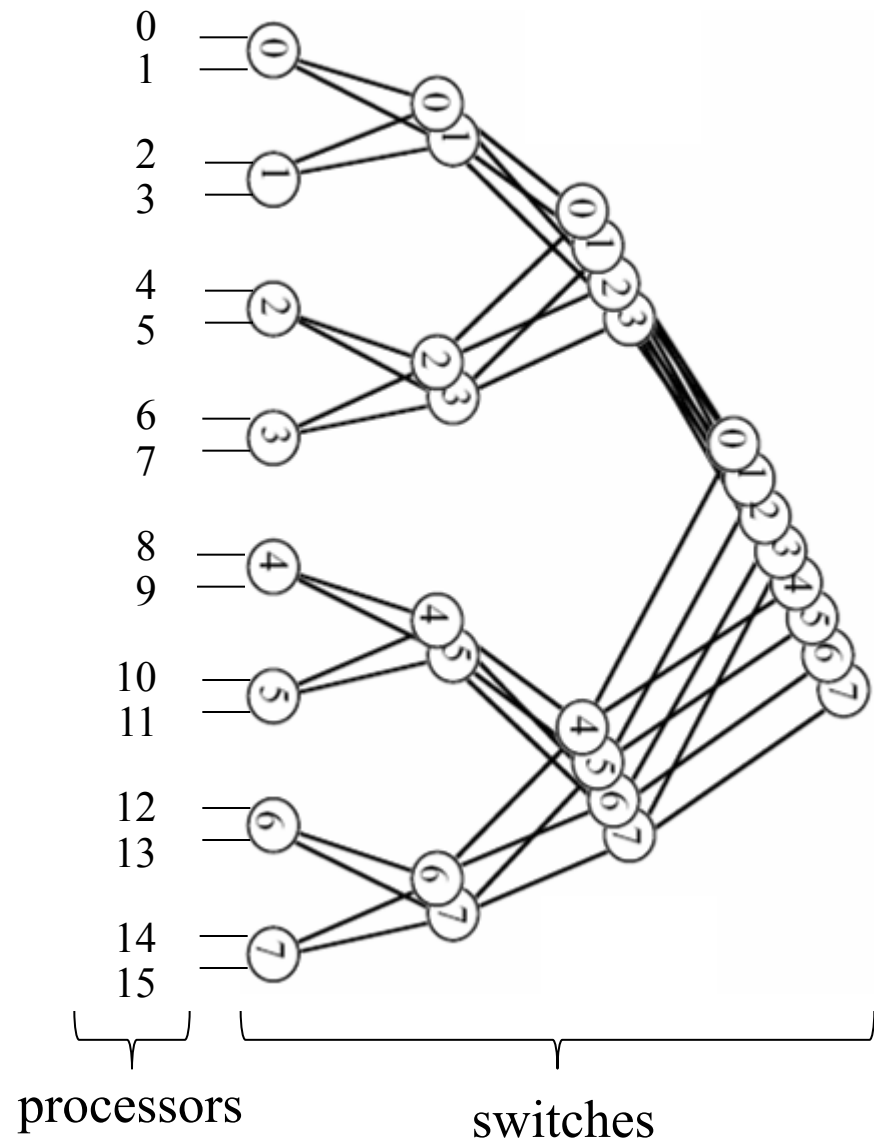


A 16-node fat tree network

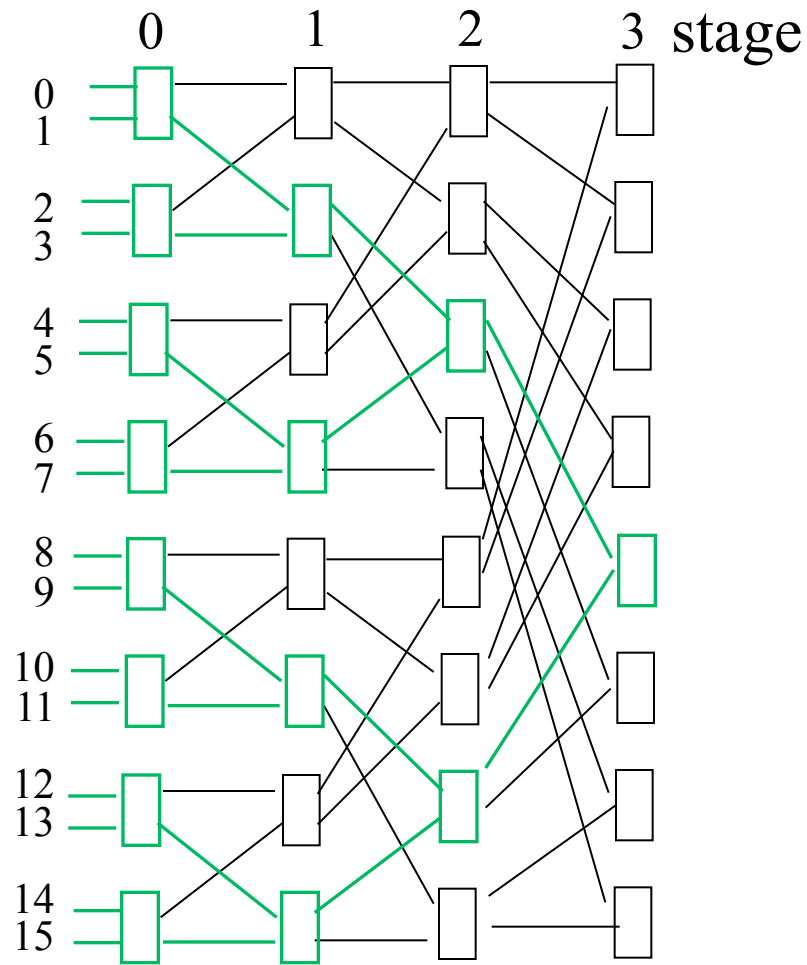


A fat tree networks using
2x2 bidirectional switches

Routing in fat trees??

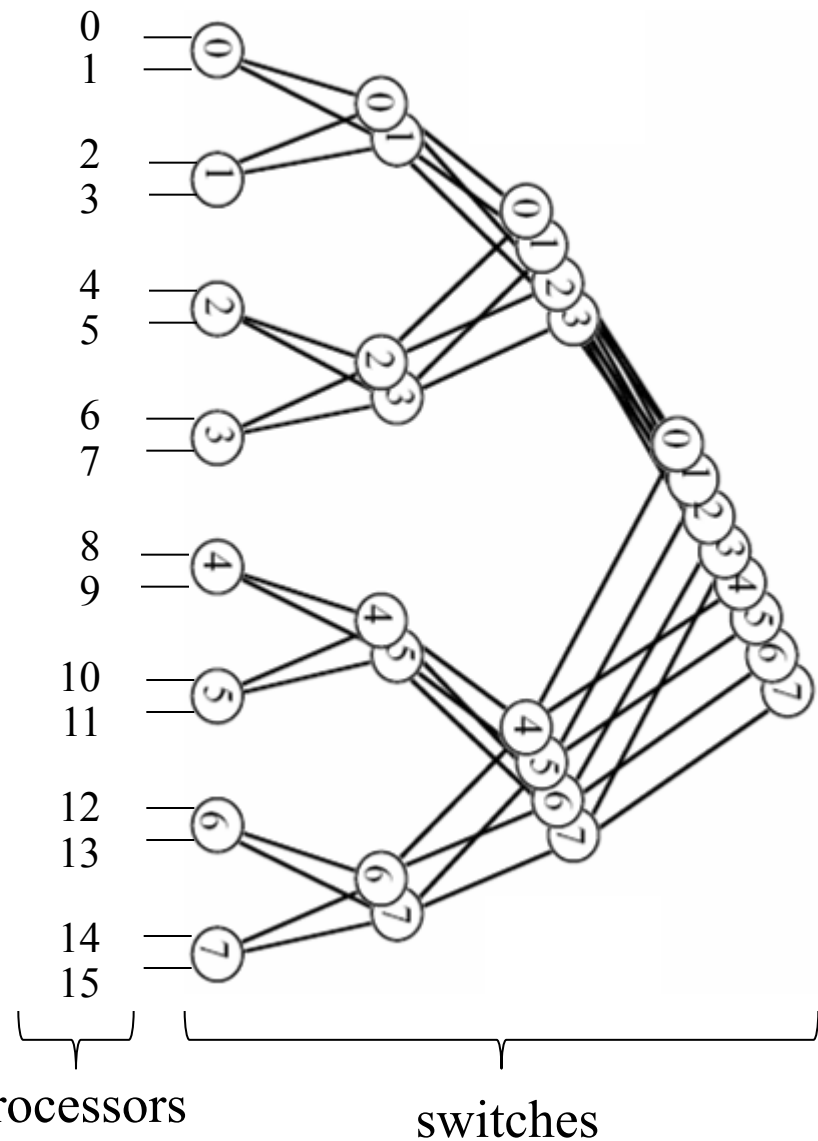


A 16-node fat tree network



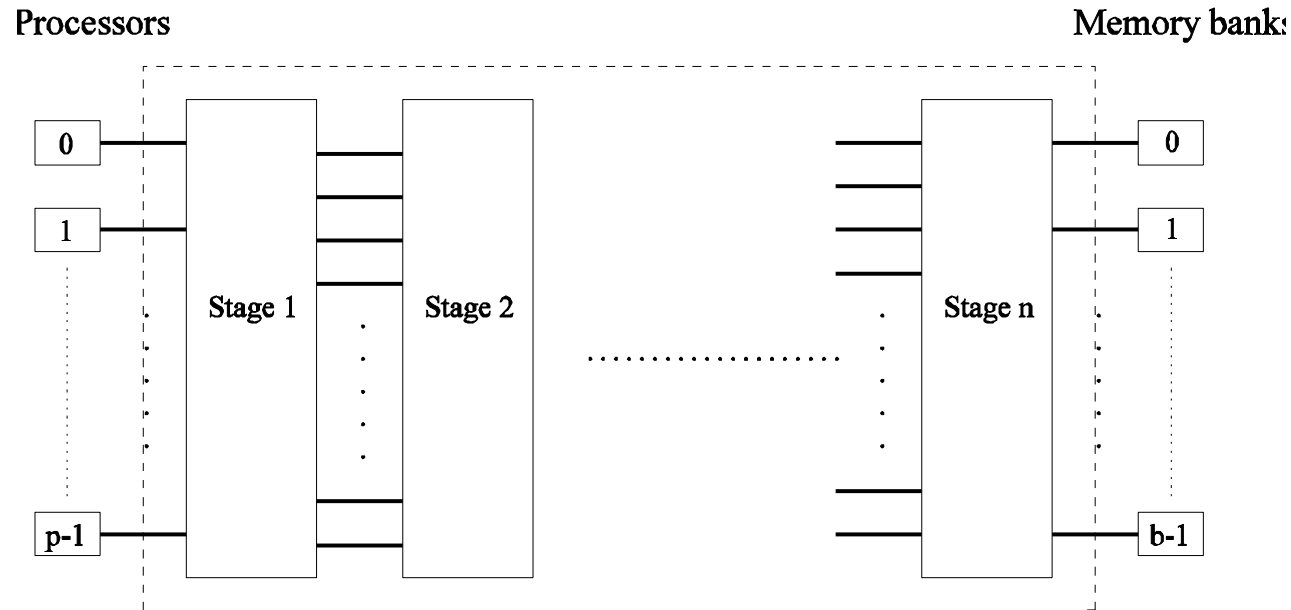
A fat tree networks using
2x2 bidirectional switches

Routing in fat trees??



Slide courtesy of Rami Melhem

Multistage Networks



The schematic of a typical multistage interconnection network.

Multistage Omega Network

A $n \times n$ Omega network consists of:

- $\log n$ stages,
- each stage has $n/2$, 2×2 switches
- Perfect shuffle connection between stages

i connects to $2i$ for $i = 0, \dots, n/2-1$

i connects to $2i+1-n$ for $i = p/2, \dots, n-1$

000 0 ————— 0 000 = left_rotate(000)

001 1 ————— 1 001 = left_rotate(100)

010 2 ————— 2 010 = left_rotate(001)

011 3 ————— 3 011 = left_rotate(101)

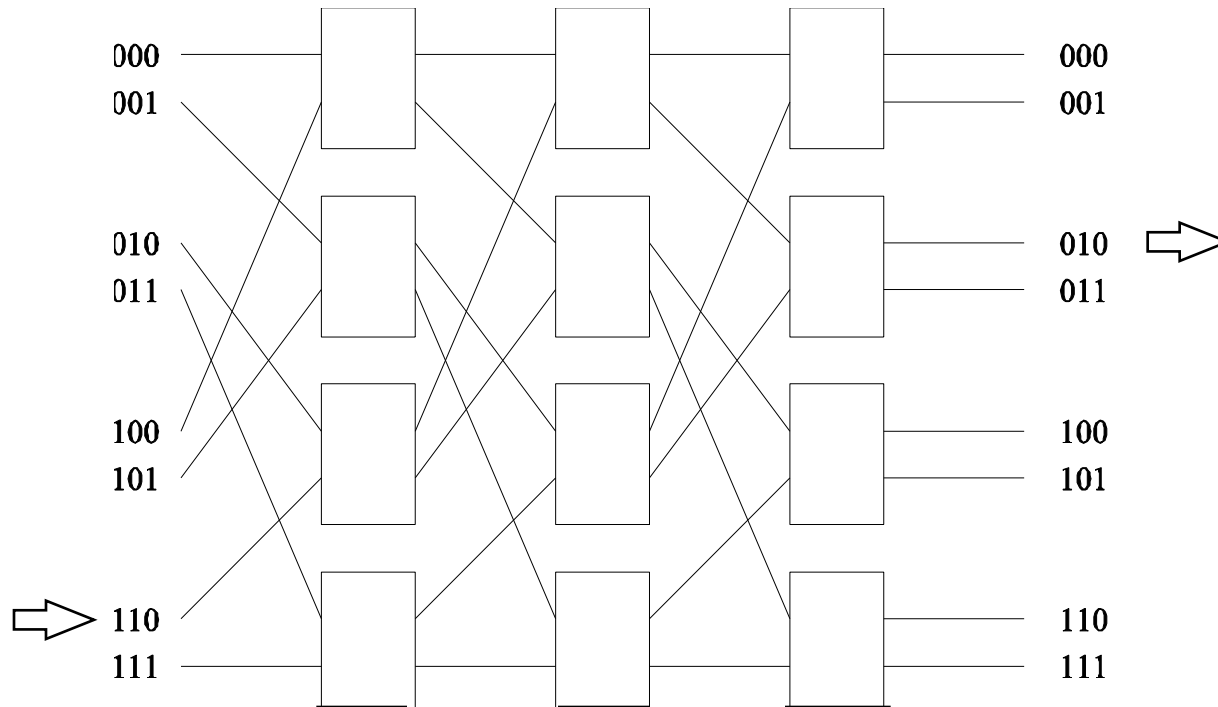
100 4 ————— 4 100 = left_rotate(010)

101 5 ————— 5 101 = left_rotate(110)

110 6 ————— 6 110 = left_rotate(011)

111 7 ————— 7 111 = left_rotate(111)

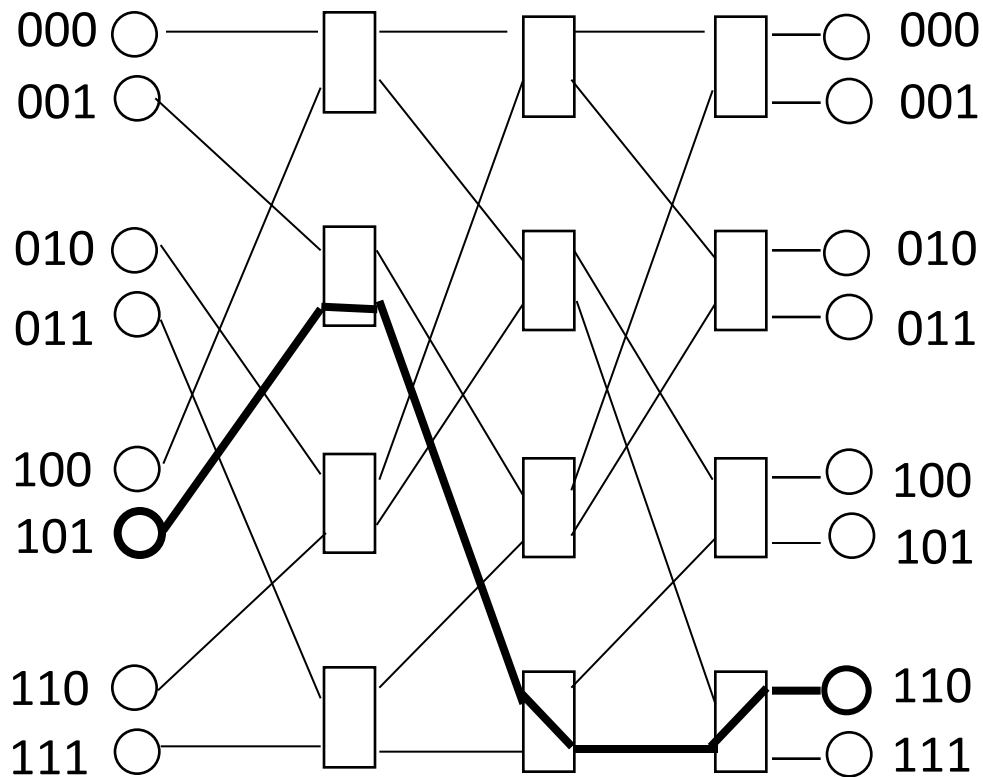
Multistage Omega Network



An 8x8 omega network

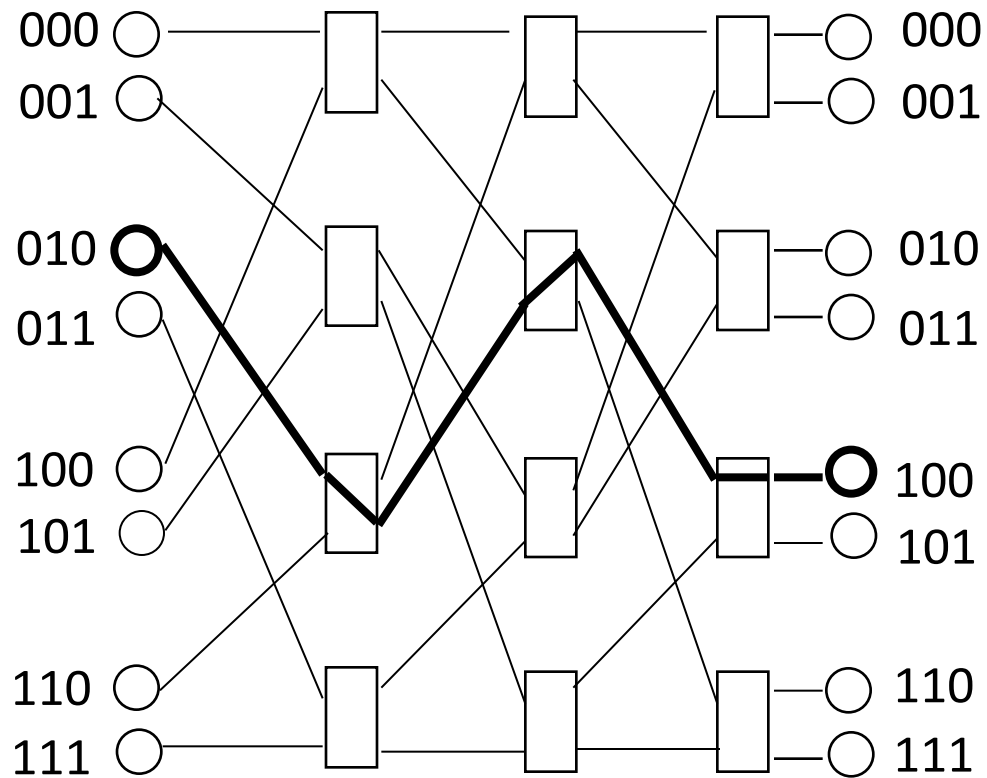
- cost grows as $n \log n$
- Unique route between a source, s , and a destination, d .

Routing in an OMEGA network



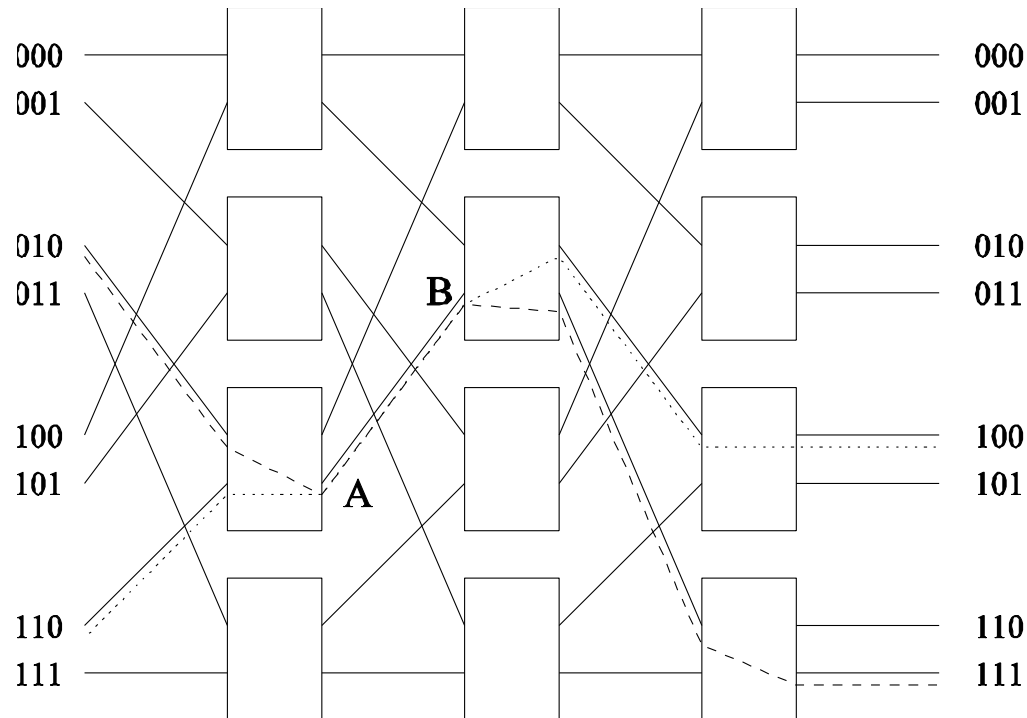
To route from source 101 to destination 110 = Lower>Lower>Upper
Configure network based upon destination alone, if 1 = Lower, if 0 = Upper
Works regardless of source 😊

Routing in an OMEGA network



Example: to route from source 010 to destination 100

The Omega Network is blocking

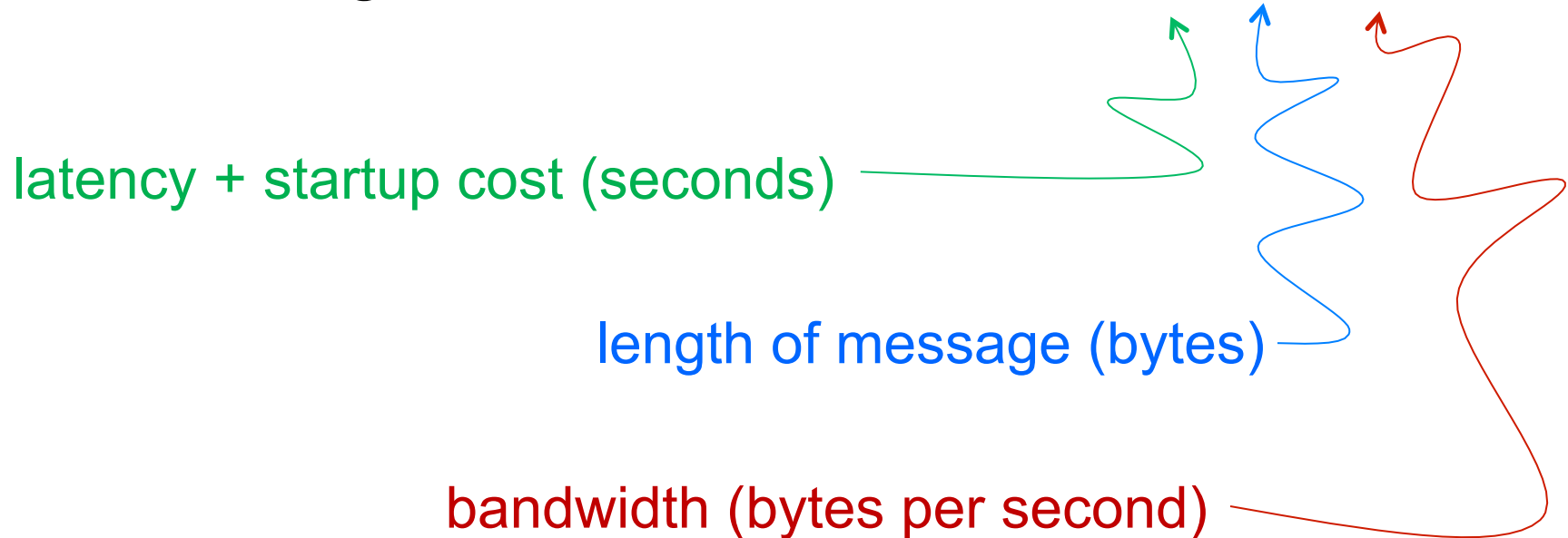


Example: one of the messages (010 to 111 or 110 to 100) is blocked at link AB.

More definitions

- Any time data is transmitted, we're interested in how long it will take for the data to reach its destination.
- **Latency**
 - The time that elapses between the source's beginning to transmit the data and the destination's starting to receive the first byte.
- **Bandwidth**
 - The rate at which the destination receives data after it has started to receive the first byte.

$$\text{Message transmission time} = l + n / b$$



Example: what is the transmission time for a message of length 256 Bytes if the network delay is 500 nsec and bandwidth is 100 MB/sec?

Routing Types

- *Store-and-forward*: entire message is received and stored at each node before being forwarded to next node on path:

$$T = (l + n/b)D$$

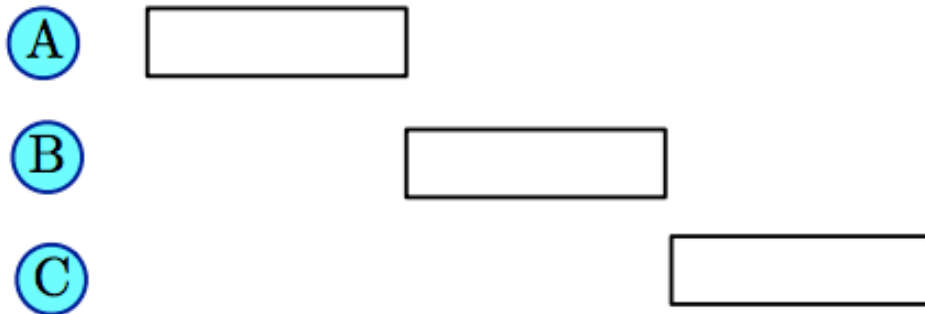
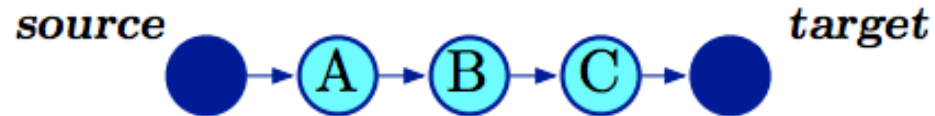
where D is the number of hops in path.

- *Cut-through (or wormhole)*: message broken into segments that are pipelined through network, with each segment forwarded as soon as it is received:

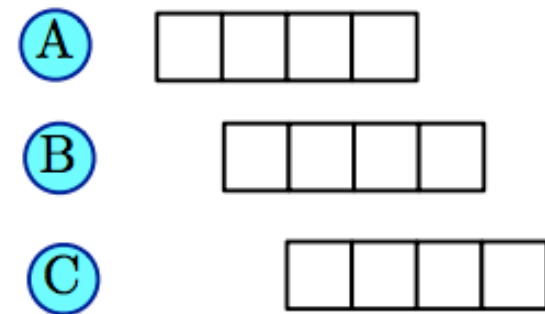
$$T = (l + n/b) + \gamma D$$

where γ is the incremental time per hop

Routing Types



store-and-forward



cut-through

Communication Concurrency

- A node may or may not be able to:
 - send a message while receiving another simultaneously on the same communication link.
 - send message on one link while receiving simultaneously on different link.
 - send or receive, or both, simultaneously on multiple links.
- Concurrency of the communication system has a high impact on performance. The time required for each step of communication is effectively multiplied by a factor.

Communication Overhead

- Communication might require processor time which means that could eat into your power and computation.

