CSE 6220/CX 4220

Introduction to HPC

Lecture 19: Programming Models / Architectures and Interconnect Networks

Helen Xu

hxu615@gatech.edu

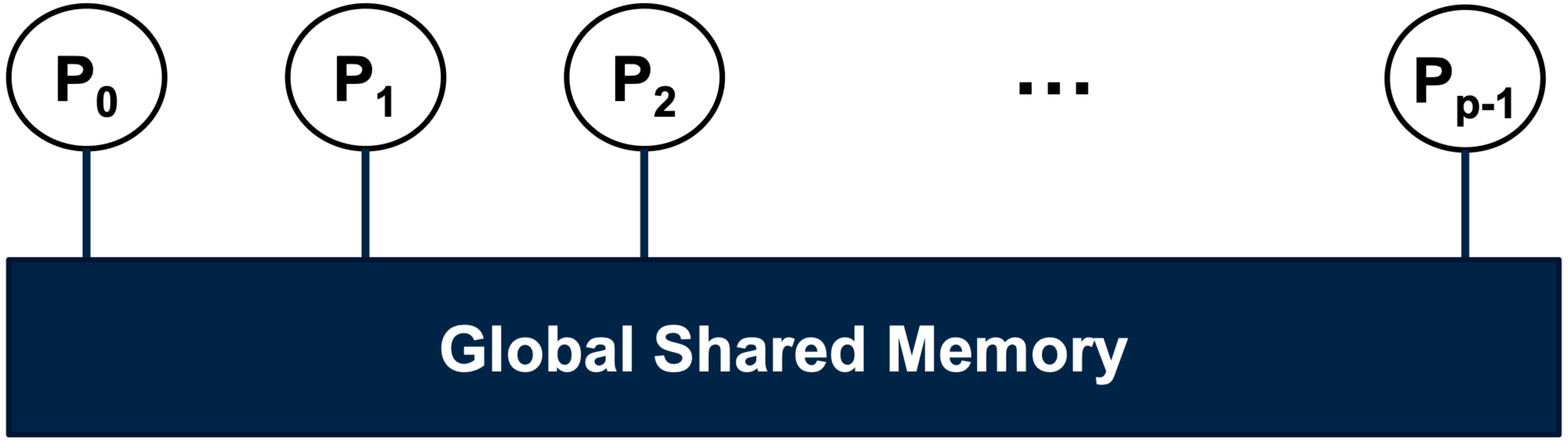
Georgia Tech College of Computing
School of Computational Science and Engineering

Slide credits to Prof. Srinivas Aluru

Parallel Models / Architectures

First half - revisiting shared-memory models for idealized parallel processors

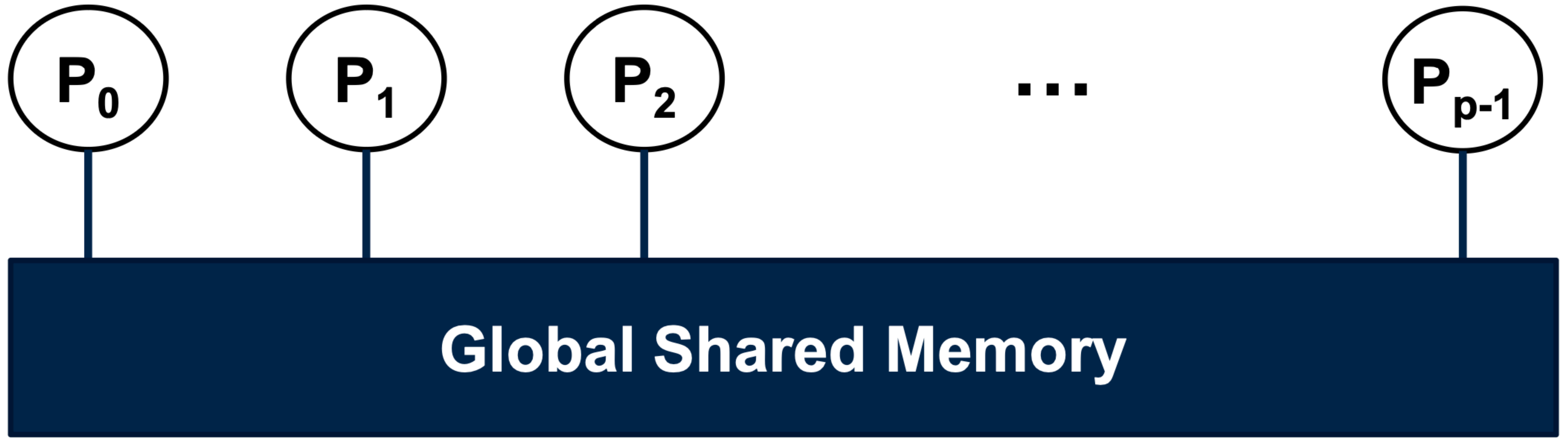
The first model was the **Parallel Random Access Model** (PRAM) - before real parallel computers were built



Architecture of an Ideal Parallel Computer

A natural extension of the Random Access Machine (RAM) serial architecture is the **Parallel Random Access Machine**, or PRAM.

- PRAMs consist of **unlimited processors** and a **global memory of unbounded size** that is uniformly accessible to all processors.
- Processors share a common clock but may execute different instructions in each cycle.
- Each processor can simultaneously undertake $O(1)$ work in $O(1)$ time.
- Each processor can access any memory location in $O(1)$ time.



PRAM Model

Depending on how simultaneous memory accesses are handled, PRAMs can be divided into four subclasses.

- Exclusive-read, exclusive-write (EREW) PRAM.
- Concurrent-read, exclusive-write (CREW) PRAM.
- Concurrent-read, concurrent-write (CRCW) PRAM.
- Exclusive-read, concurrent-write (ERCW) PRAM.

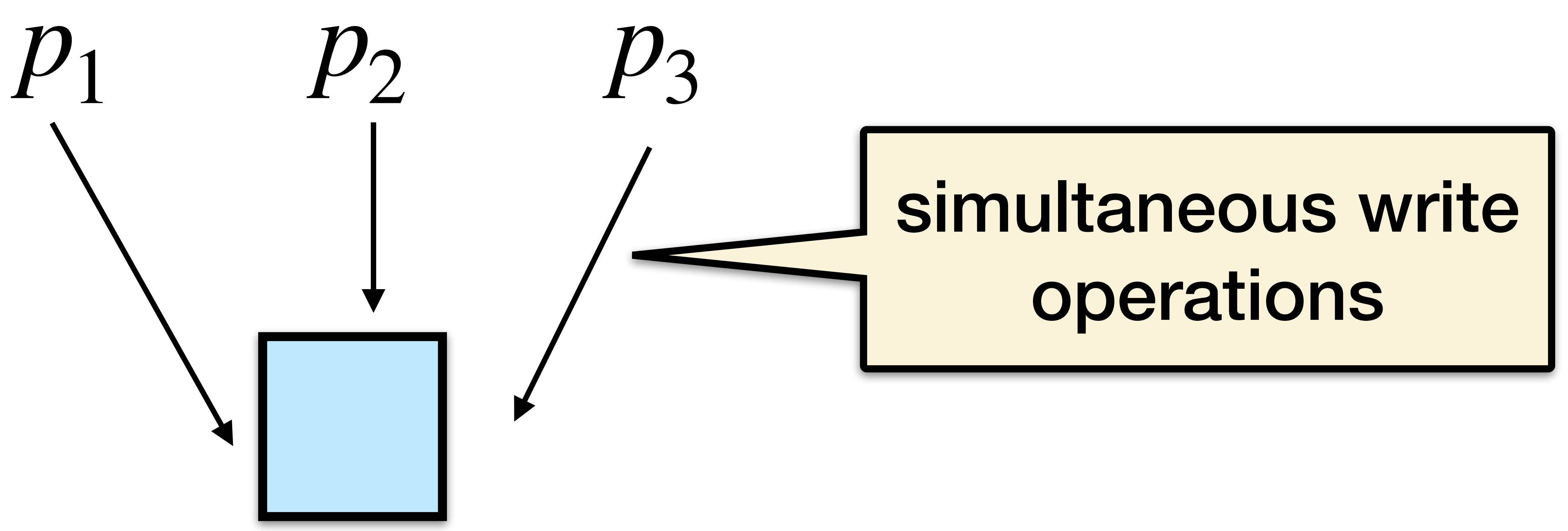
Fourth one is for completeness but not used much.

The first three are the most popular - listed in **increasing levels of power**

Concurrent Writes

What does **concurrent write** mean, anyway?

- Common: write only if all values are identical.
- Priority: follow a predetermined priority order (e.g., min, max, etc.).
- Sum: Write the sum of all data items.
- Arbitrary: write the data from a randomly selected processor.



Example: Sum p numbers on PRAM

What is the time complexity of **adding** p numbers if we have:

Concurrent-read, concurrent-write (CRCW) PRAM: Sum model

- All processors write to the sum variable concurrently
- $O(1)$

Exclusive-read, exclusive-write (EREW) PRAM

- Similar to the tree-like algorithms we have done so far
- $O(\log p)$

Problem: Compute logical OR on PRAM

What is the time complexity of **computing logical OR** of p Booleans if we have:

Concurrent-read, concurrent-write (CRCW) PRAM: Common model (only write identical values

- Start by writing 0 in the result, then a processor writes 1 iff it has 1
- $O(1)$

Exclusive-read, exclusive-write (EREW) PRAM

- Similar to the tree-like algorithms we have done so far
- $O(\log p)$

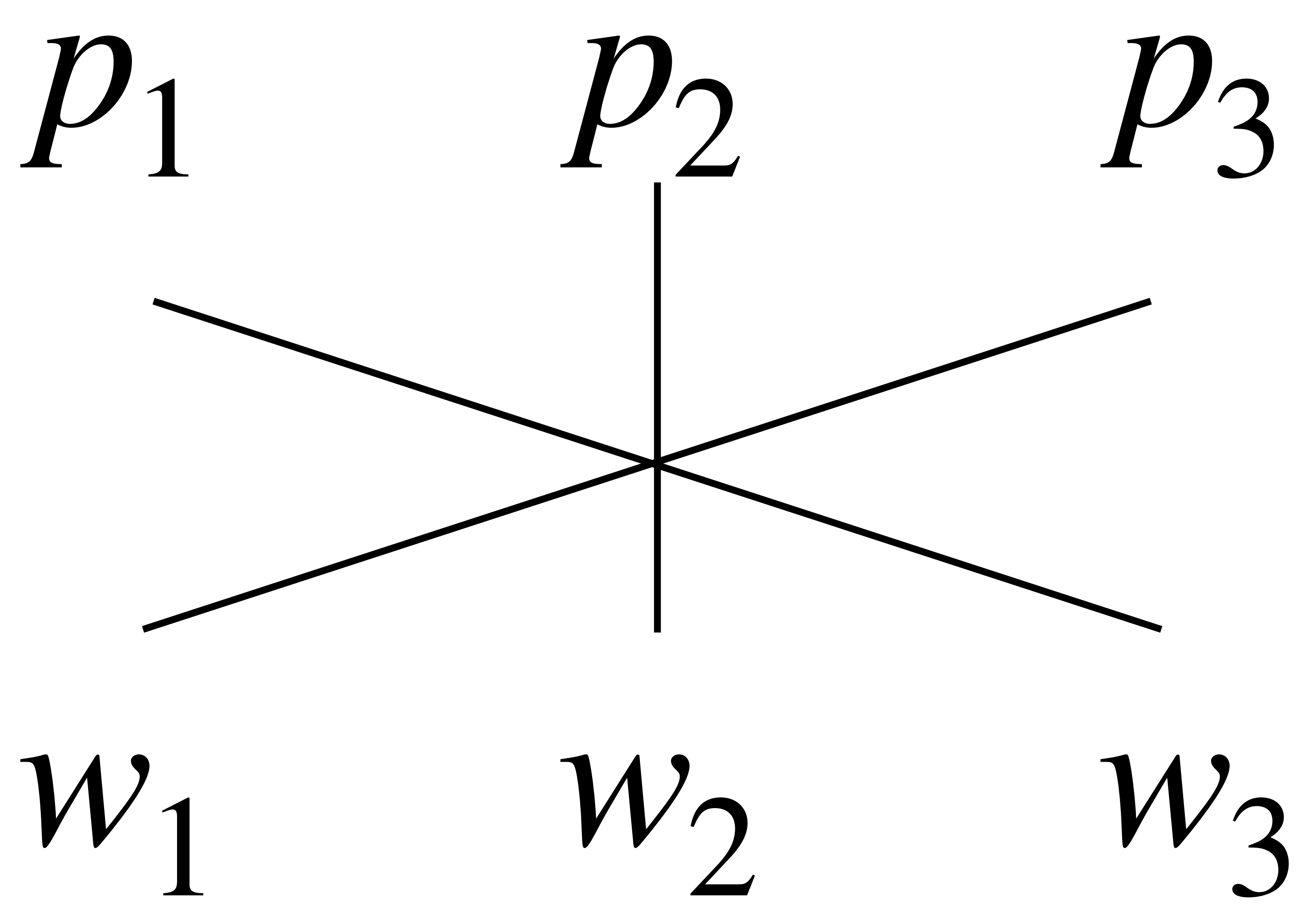
Complexity Theory for Parallel Algorithms

- **P-completeness:** Given any polynomial number of processors - can it be solved in polylog time?
- Every algorithm that is **NP-hard serially is also NP-hard in parallel** - by reducing the parallel algorithm to a serial algorithm
  - Given a polynomial number of processors, time would only go up polynomially to run it on one processor.
- There are some problems that are polynomially solvable serially but **hard to parallelize** - e.g., some forms of depth-first search

Physical Complexity of PRAM Computer

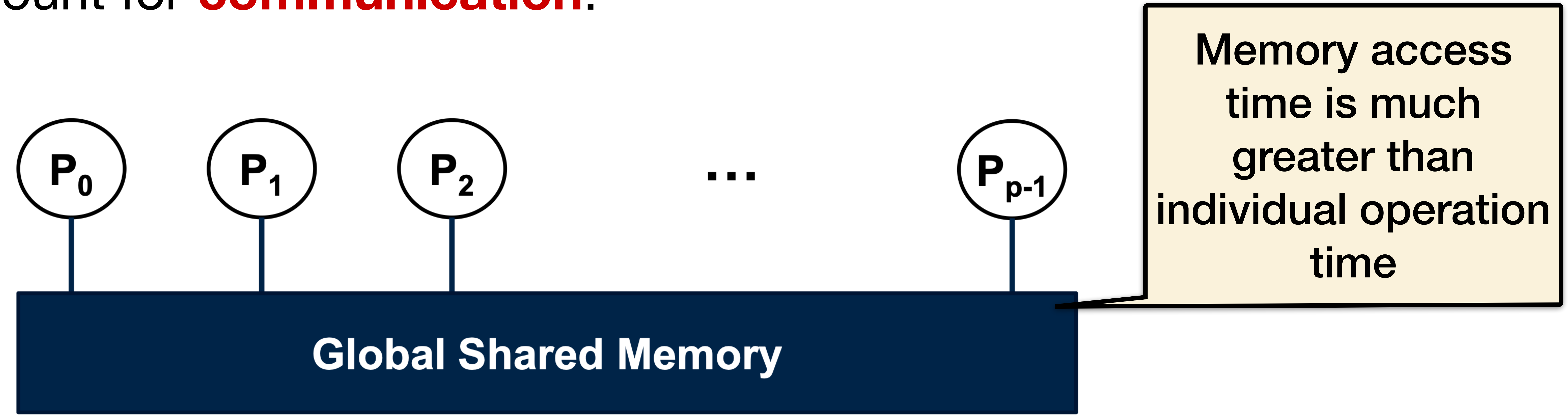
Processors and memories are **connected via switches**.

- Since these switches must operate in $O(1)$ time at the level of words, for a system of p processors and m words, the switch complexity is $O(mp)$.
- Clearly, for meaningful values of p and m, **a true PRAM is not realizable**.



PRAM Limitations

Many of the original PRAM algorithms didn't work well in practice because they didn't account for **communication**.

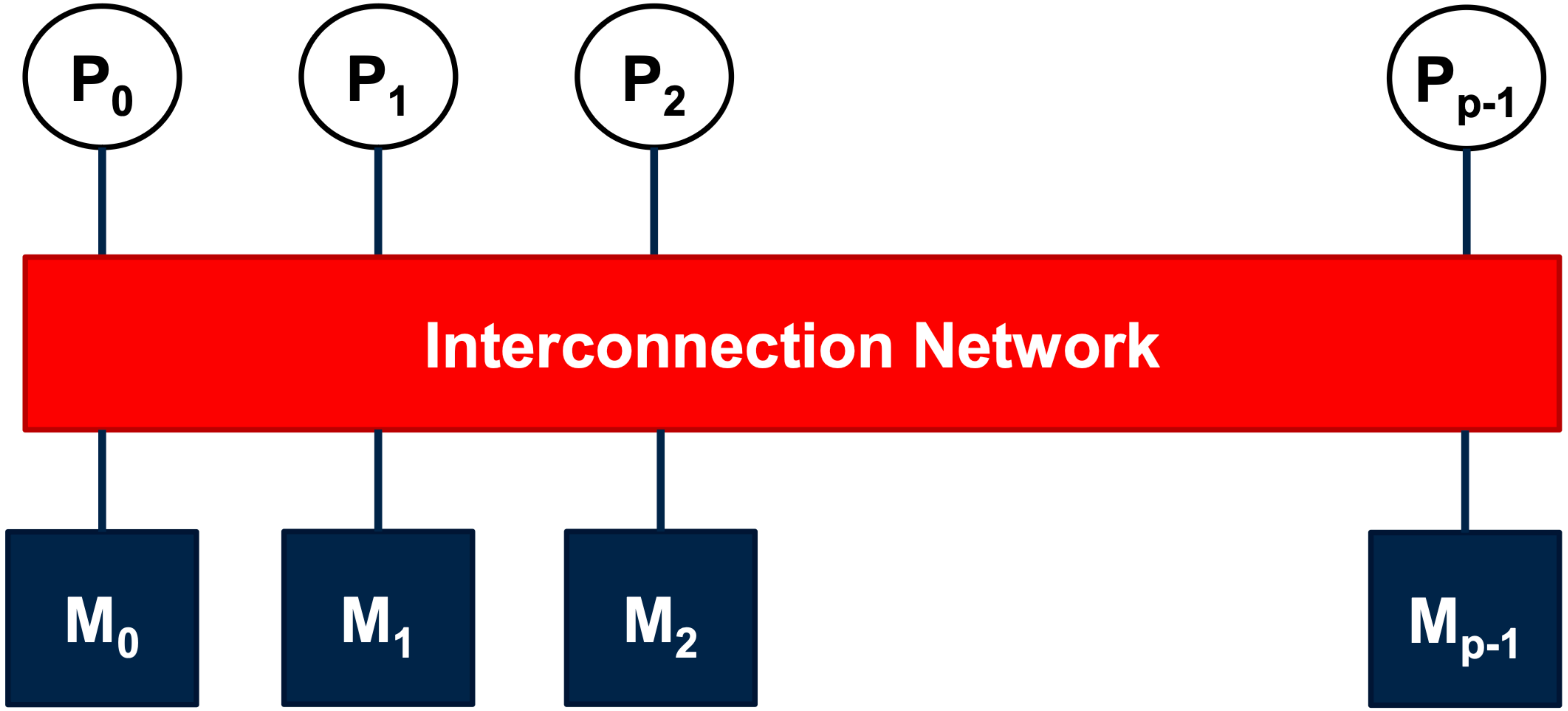


P_0
P_1
P_2
...
P_{p-1}
Global Shared Memory
Memory access time is much greater than individual operation time

The PRAM model is still useful as a **first pass at designing** a parallel algorithm, since other models are more complex.

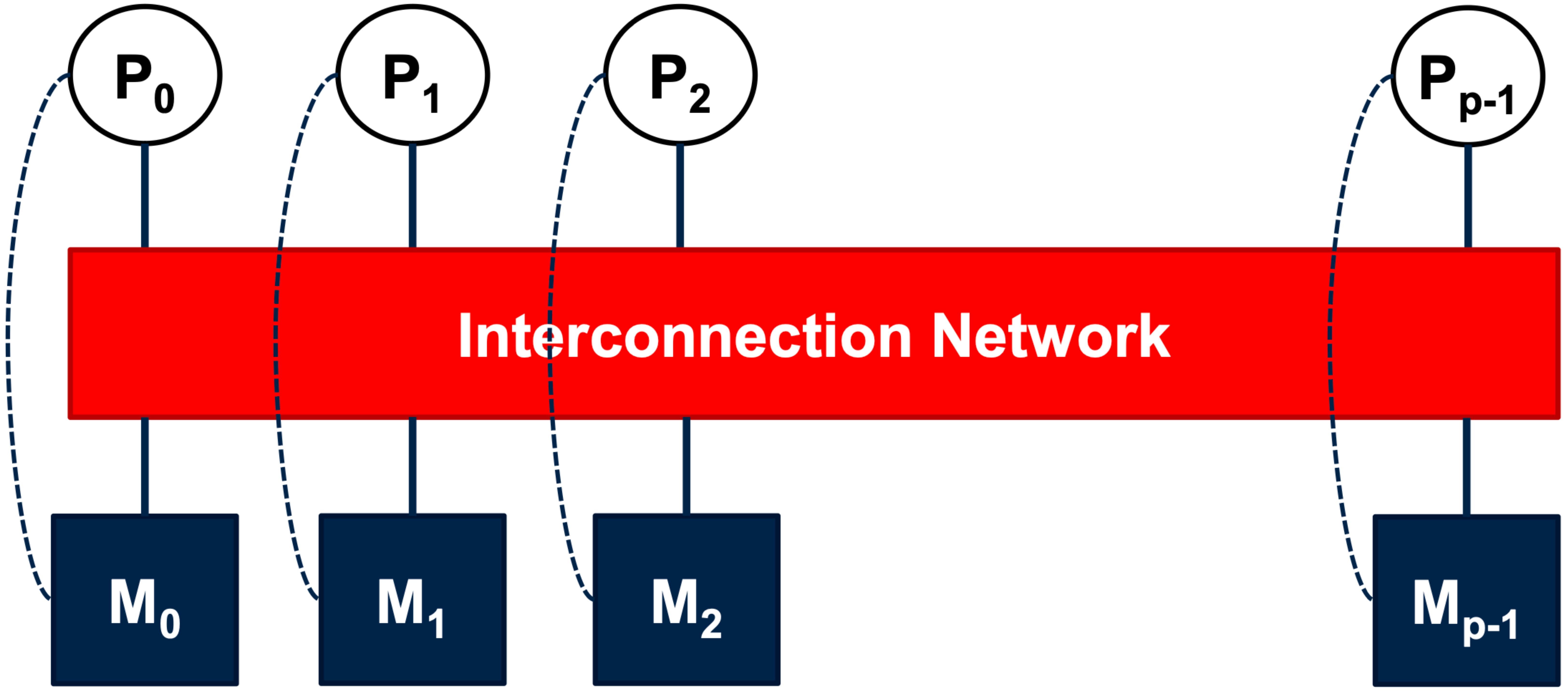
Networked Shared Memory Model

- **Interconnection network** between processors and memory modules (might be larger than one element)



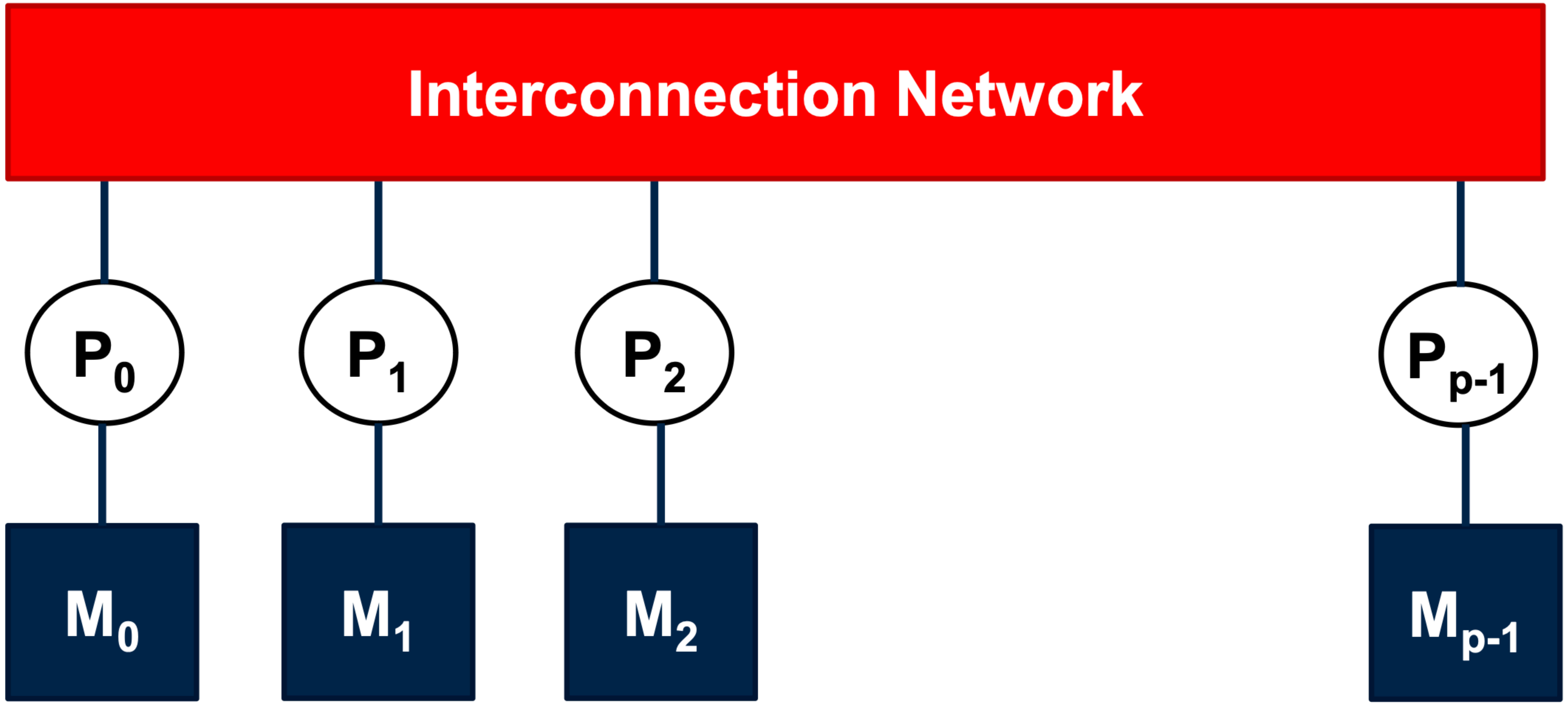
Networked Shared Memory Model

One type of non-uniform memory access (NUMA) - **private link between each processor and its related memory module** (bypassing the interconnect).



Networked Distributed-Memory Model

Another way to view the previous architecture is the distributed-memory model we have been working with so far.



Distinguishing between shared and distributed memory

There are multiple levels at which we can specify whether we are using shared or distributed memory:

- Algorithm (depends on the model)
- Programming language / abstraction (e.g., MPI)
- Actual hardware/architecture

Abstractions

Assume p processors with a memory of m per processor

Total Memory = $mp = M$

	Shared	Distributed
Algorithm	$O(1)$ time to access memory	$O(1)$ time to access to only local memory
Programming	$< 0 \ldots M-1 >$	$< 0 \ldots p-1, 0 \ldots m-1 >$

How to support a shared-memory programming abstraction on top of distributed memory?

Abstractions

How to support shared memory programming abstraction on top of a distributed memory?

$< i > \rightarrow < i / m, i \% m >$

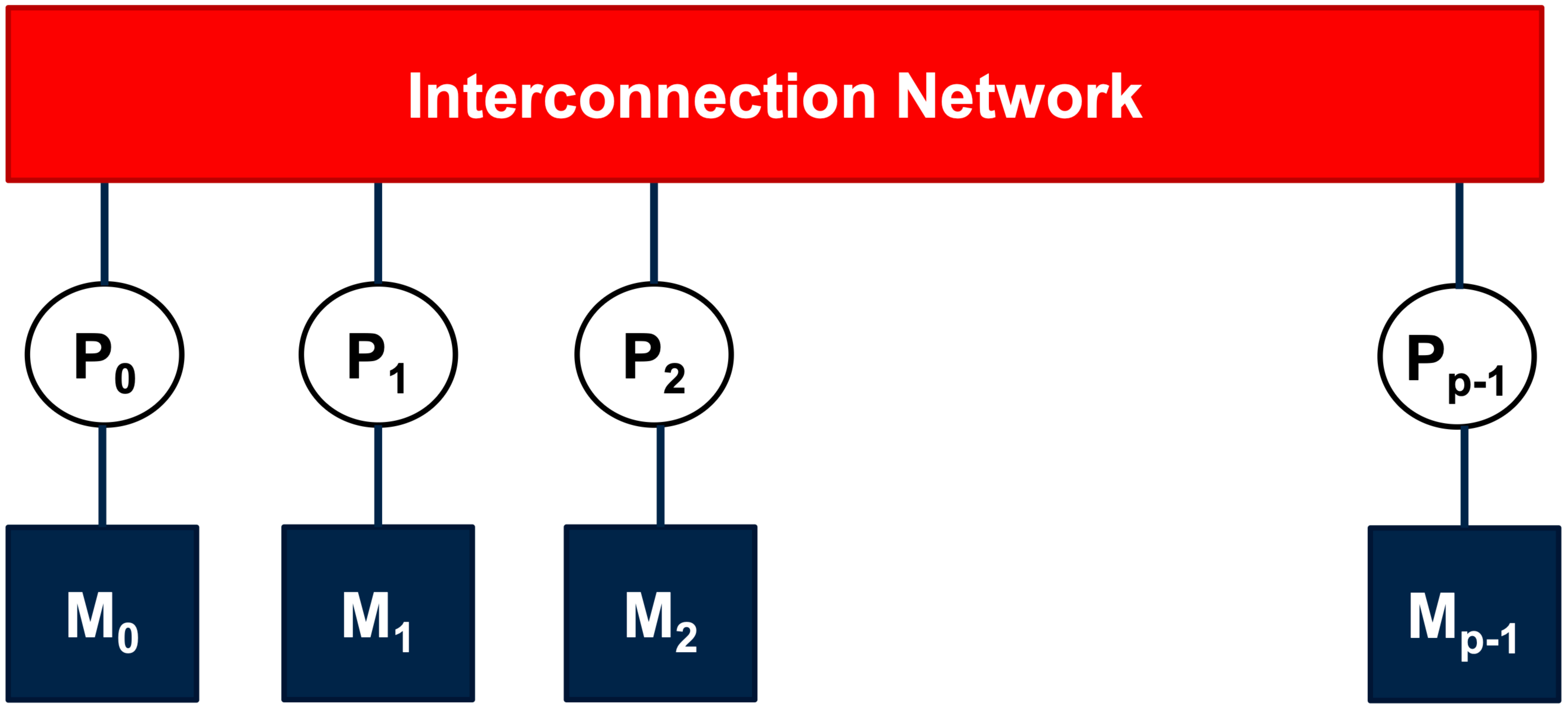
	Shared	Distributed
Algorithm	$O(1)$ time to access memory	$O(1)$ time to access to only local memory
Programming	$< 0 \ldots M - 1 >$	$< 0 \ldots p - 1, 0 \ldots M - 1 >$
Architecture	✗	✓

Example

Consider the following simple problem - every processor tries to read the same global memory location (e.g., a) and set their own local variable (e.g., x).

- In CRCW/CREW, it would take O(1) time.
- In practice, it would actually take something like O(p) time since processors have to queue up at memory locations.
- To resolve the issue - declare array of size p, one processor reads a, writes it to another location, then 2 processors read those and write a to 2 more locations, etc. Then all p processors set x = a from different slots - O(log p)

What should the interconnect network be?



Interconnection Networks for Parallel Computers

Interconnection networks carry data between processors and to memory.

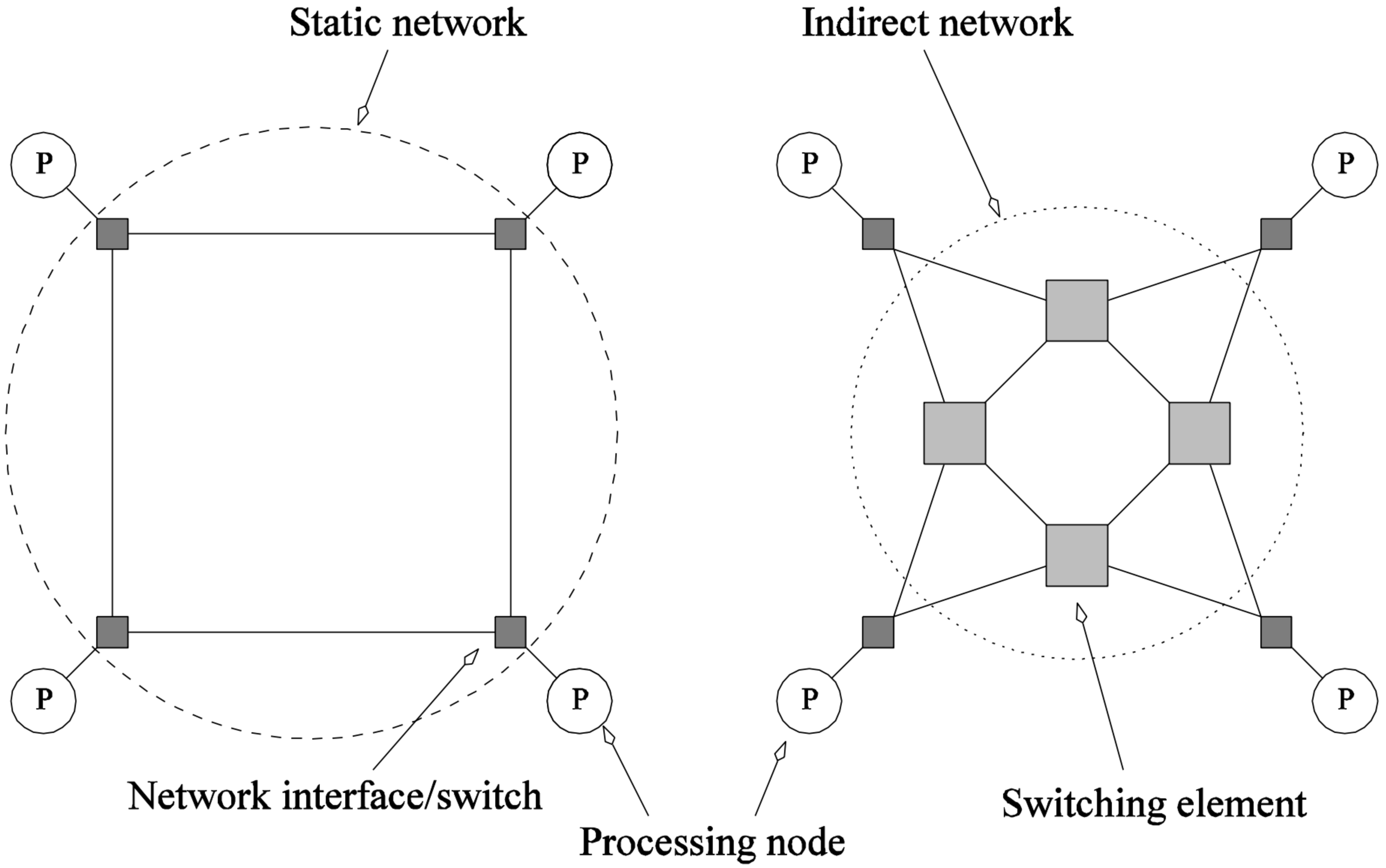
Interconnects are made of switches and links (wires, fiber).

Interconnects are classified as ***static*** or ***dynamic***.

Static networks consist of **point-to-point communication links** among processing nodes and are also referred to as ***direct*** networks.

Dynamic networks are built using **switches and communication links**. Dynamic networks are also referred to as **indirect networks**.

Static and Dynamic Interconnection Networks



Evaluating Static Networks

Distance*: between a pair of processors is the number of links on a shortest path connecting the two.*

Diameter: The largest of distances between pairs of processors.
Ideal Diameter: $\Theta(1)$

Bisection Width (BW): The minimum number of wires you must cut to divide the network into two subnetwork with $p/2$ processors each.

Ideal Bisection Width: $\Theta(p)$

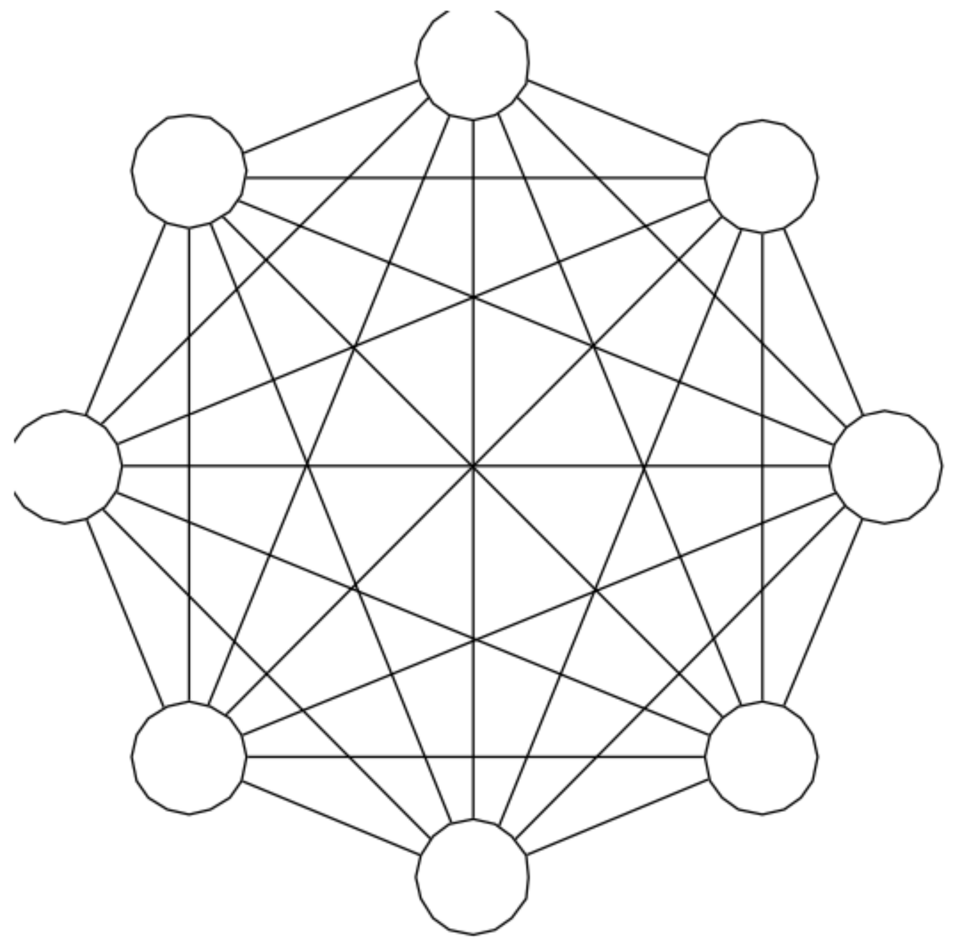
Evaluating Static Networks

***Cost**:* The number of links or switches (whichever is asymptotically higher) is a meaningful measure of the cost. However, a number of other factors, such as the ability to layout the network, the length of wires, etc., also factor in to the cost.

Ideal Cost: $O(p)$

Network Topologies: Completely Connected Network

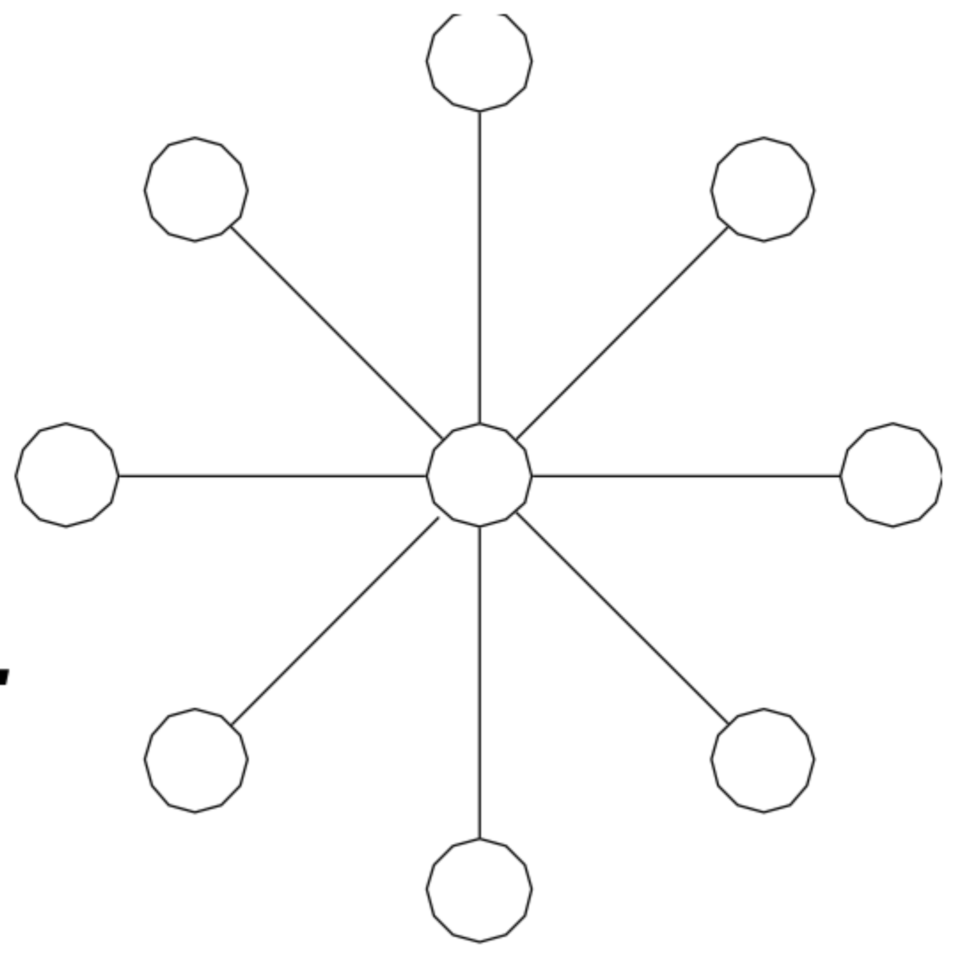
- Each processor is connected to every other processor.



- Diameter = 1
- BW = $\frac{p}{2} \times \frac{p}{2} = \frac{p^2}{4} = \Theta(p^2)$
- Number of Links per node: $p - 1$
- Cost : The number of links in the network scales as $O(p^2)$.
- While the performance scales very well, the hardware complexity is not realizable for large values of p.

Network Topologies: Star Connected Network

- Every node is connected only to a common node at the center.



- Diameter = 2
- Distance between any pair of nodes is $O(1)$.

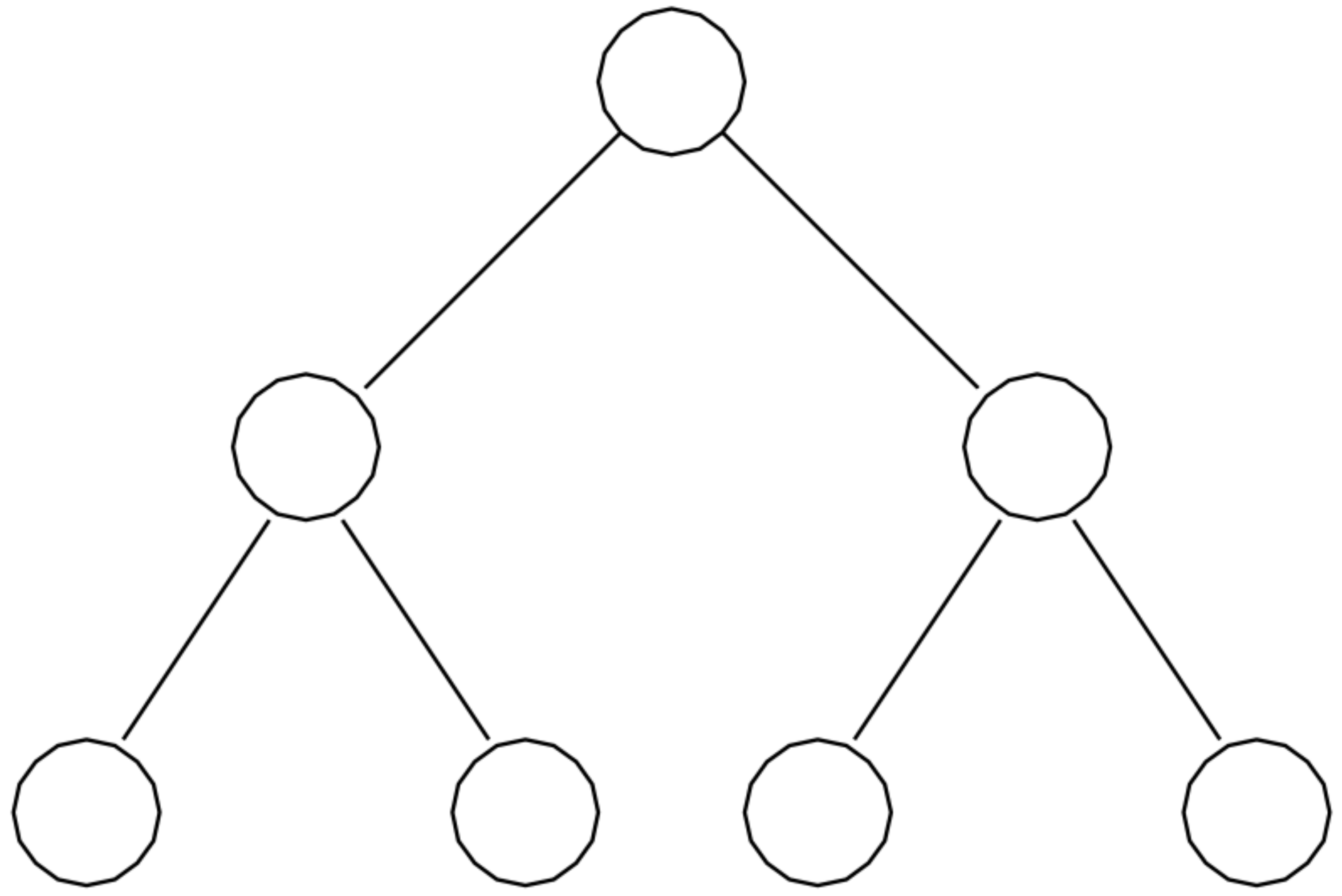
- However, the central node becomes a bottleneck.

- BW = 1

- Number of Links per node: maximum of $p-1$
- Cost : $p-1$

Network Topologies: Tree Properties

- The distance between any two nodes is no more than $2 \log p$.
- Links higher up the tree potentially carry more traffic than those at the lower levels.
- Trees can be laid out in 2D with no wire crossings. This is an attractive property of trees.



Fat Trees

Fat trees [Leiserson 85] double links (and bandwidth) at each level (starting at leaves and going up) to handle more traffic on the way up.

Example supercomputers include the Connection Machine CM-5, Summit and Sierra (until recently, the two fastest), etc.

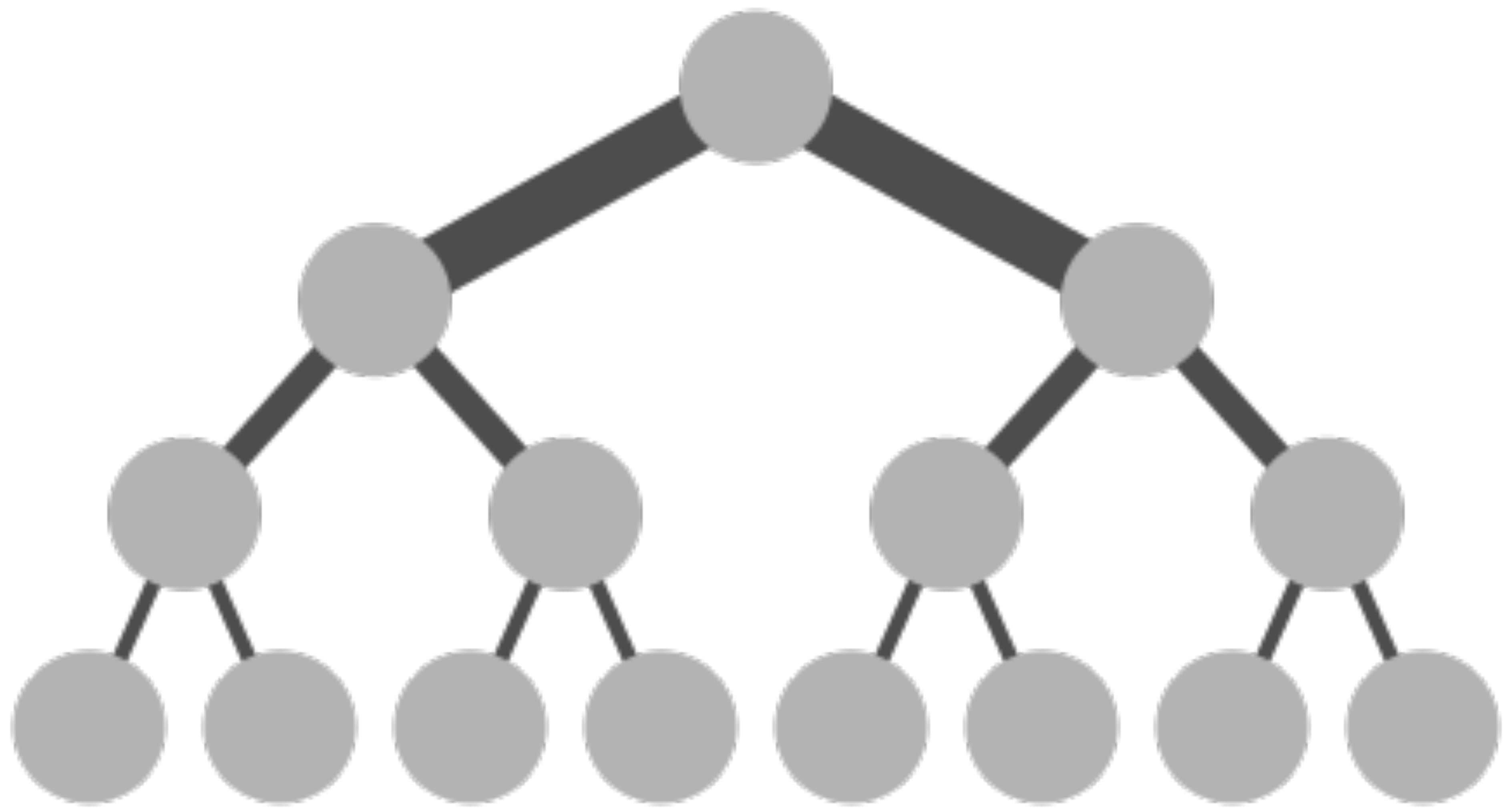
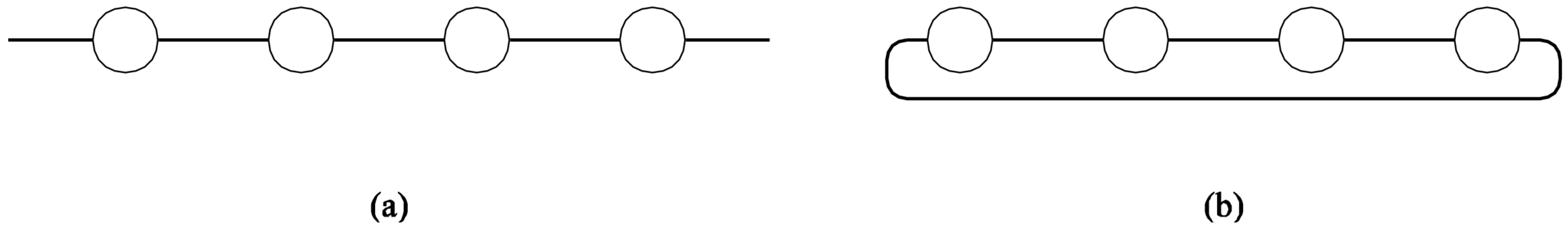


Image from https://en.wikipedia.org/wiki/Fat_tree

Network Topologies - Linear Arrays / Rings

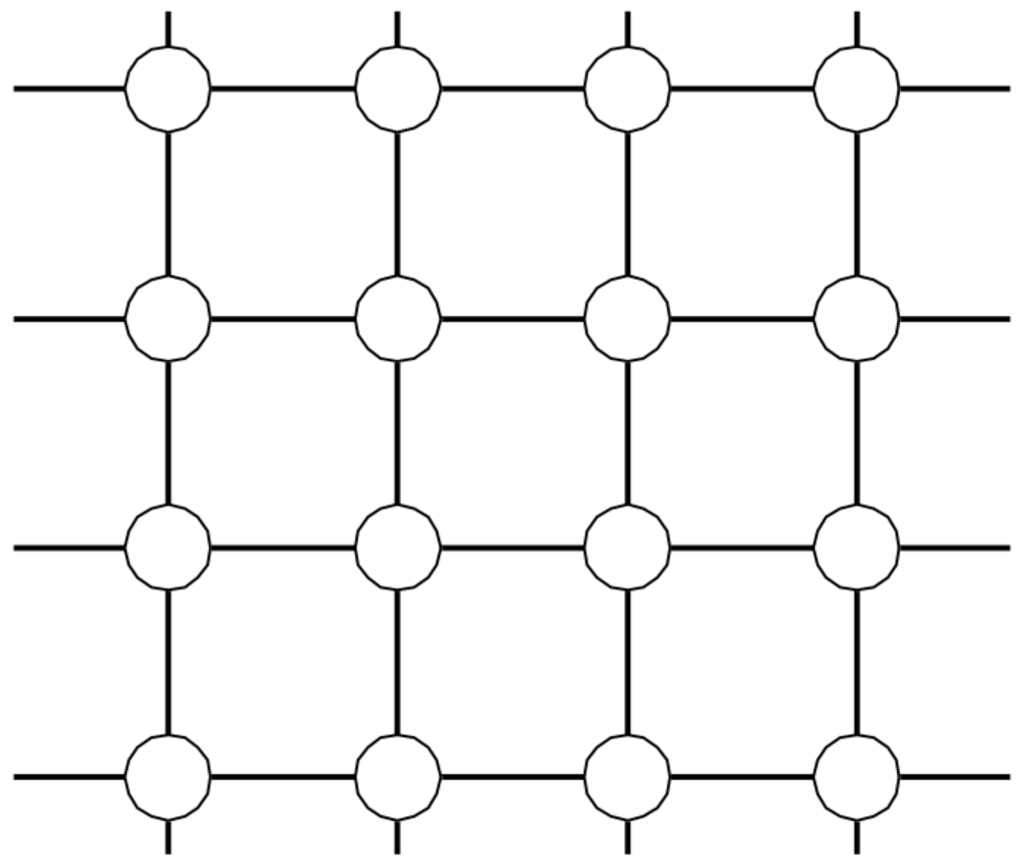
In a linear array, each node has two neighbors, one to its left and one to its right. If the nodes at either end are connected, we refer to it as a 1-D torus or a ring.

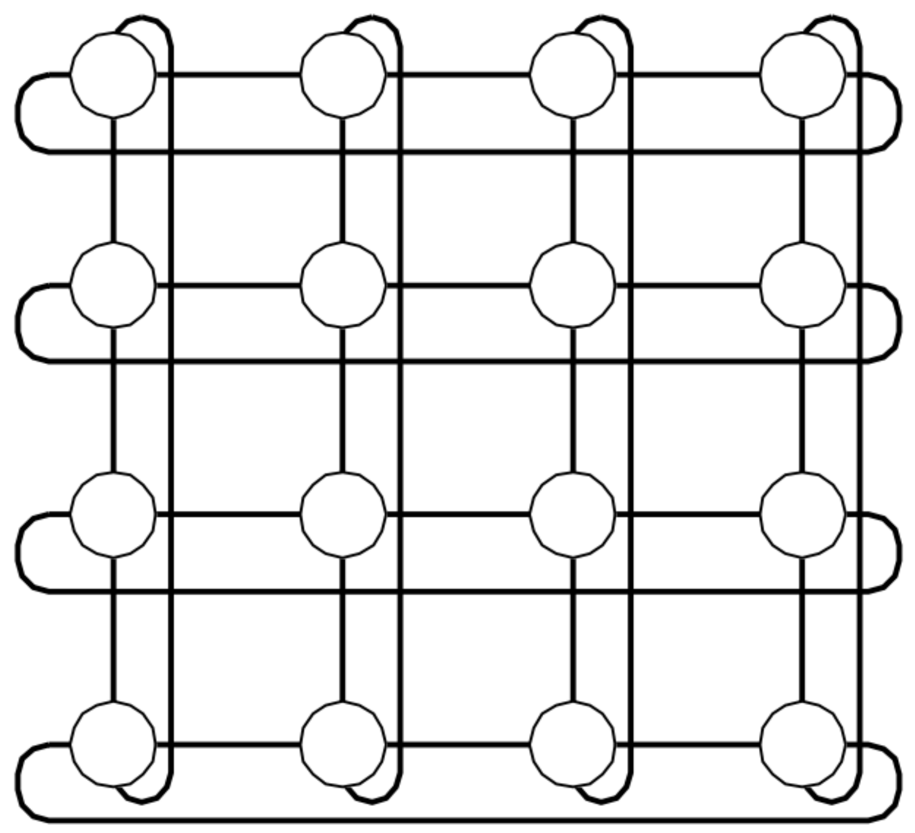


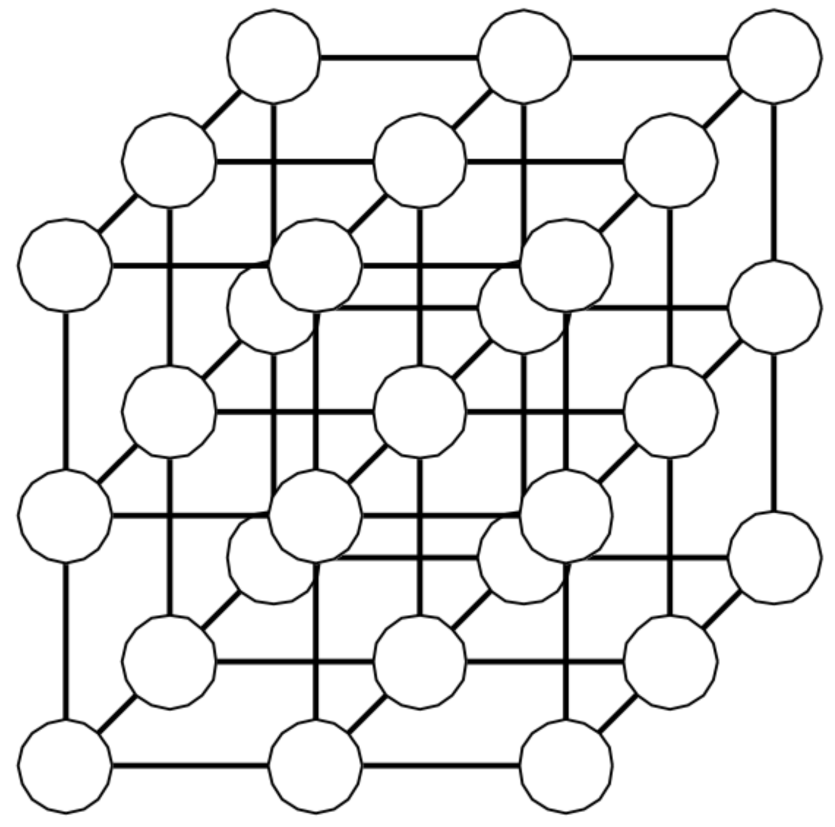
Linear arrays: (a) with no wraparound links; (b) with wraparound link.

Network Topologies: 2D and 3D Meshes

- A generalization to 2 dimensions has nodes with 4 neighbors
- A further generalization to d dimensions has nodes with $2d$ neighbors.


(a)


(b)


(c)

Two and three dimensional meshes:
(a) 2-D mesh with no wraparound;
(b) 2-D mesh with wraparound link (2-D torus); and
(c) a 3-D mesh with no wraparound.

d dimension k-ary mesh $p = k^d$

Evaluating Static Networks

Down is good

Up is good

Down is good

Down is good

Network	Diameter	Bisection Width	Max. Links per Node	Cost (total no. of links)
Completely-connected	1	$p^2/4$	$p-1$	$p(p-1)/2$
Star	2	1	$p-1$	$p-1$
Linear array	$p-1$	1	2	$p-1$
Complete binary tree	$2\log((p+1)/2)$	1	3	$p-1$
2-D mesh, no wraparound	$2(\sqrt{p}-1)$	$\sqrt{p}$	4	$2(p-\sqrt{p})$
2-D wraparound mesh (torus)	$2\lfloor\sqrt{p}/2\rfloor$	$2\sqrt{p}$	4	$2p$
Wraparound *k*-ary *d*-dimensional mesh (*p*= k^d)	$d\lfloor k/2\rfloor$ $d\left\lfloor p^{1/d}/2\right\rfloor$	$2k^{d-1}$ $2p^{1-\frac{1}{d}}$	$2d$	dp

How To Maximize Mesh Dimensionality?

$p = 32$. How many dimensions can we make?

- 8 X 4 2-D mesh
- 4 X 4 X 2 3-D mesh
- 4 X 2 X 2 X 2 4-D mesh
- 2 X 2 X 2 X 2 5-D mesh
- Can't go any further! Also, torus just duplicates mesh links.

- To maximize dimensionality, choose variable number of dimensions based on p. For $p = 2^d$, choose $d = \log p$ as the number of dimensions.

Hypercube: Mesh of Maximum Dimensionality

d = log p ; $p = 2^d$

- Diameter = $d(p^{1/d} - 1) = d(2-1) = d = \log p$
- Bisection Width = $p^{1-1/d} = \dfrac{p}{p^{1/d}} = p/2$

How do we address processors in hypercube and find neighbors?

- It’s just a mesh (2 processors in each dimension).
- d-dimensional mesh has ranks described by d coordinates.
- To find neighbors along a dimension, hold all other dimensions fixed, and increment/decrement chosen dimension by 1.

Hypercube: Processor Representation and Neighbor Finding

d-dimensions. Coordinate along each dimension is 0 or 1.

Example: (1,0,0,1,0,1) in 6-dimensional hypercube.

Neighbor along a dimension:

- If coordinate is 0, neighbor is 1 (other neighbor doesn't exist)
- If coordinate is 1, neighbor is 0 (other neighbor doesn't exist)
- Every processor is a corner processor!

Since each coordinate is 0 or 1, just write them consecutively: (1,0,0,1,0,1) → 100101

Neighbor along a dimension: invert the bit

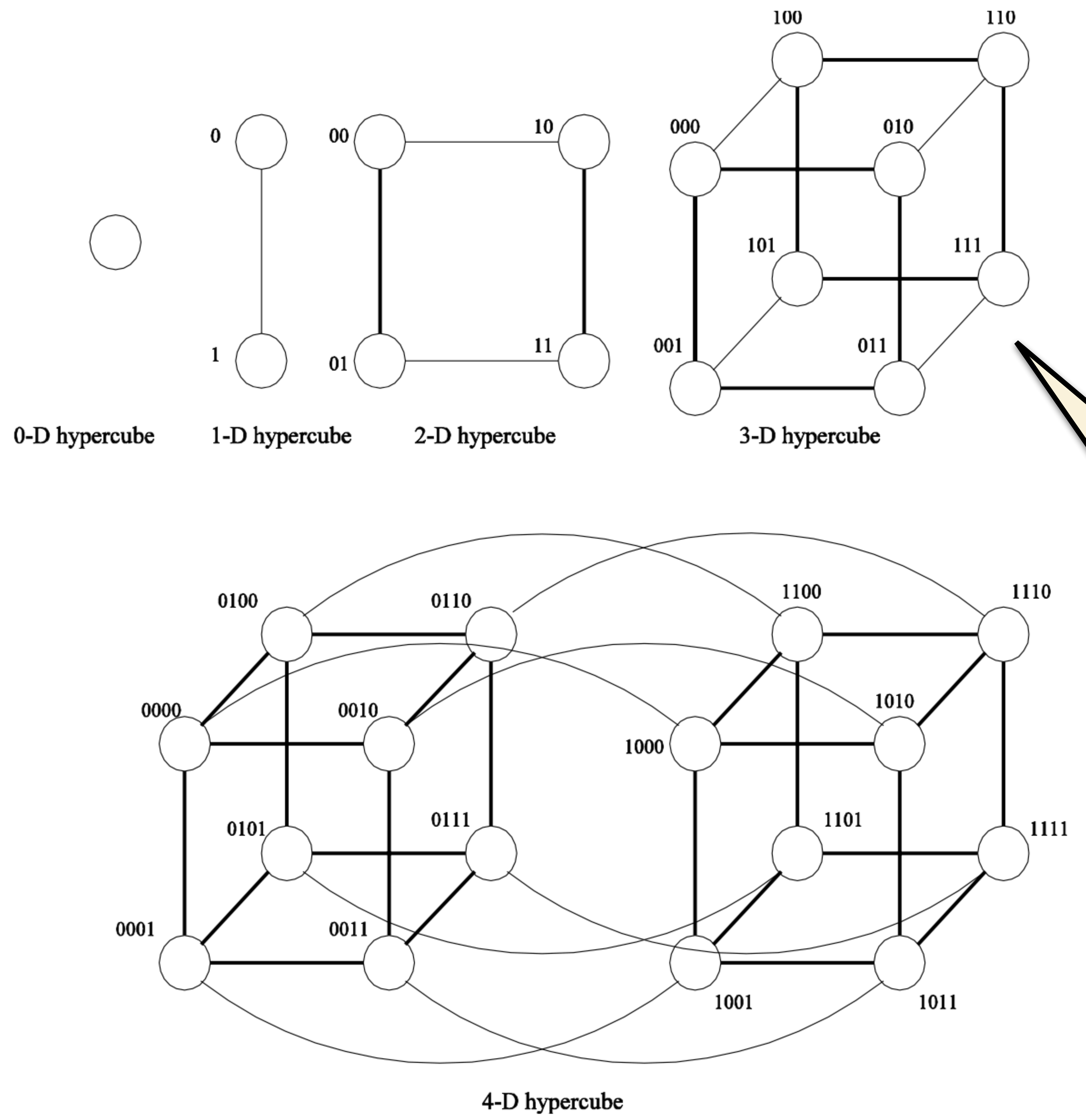
Network Topologies: Properties of Hypercubes

The distance between any two nodes is at most log p (diameter).

Each node has log p neighbors.

The distance between two nodes is given by the number of bit positions at which the two nodes differ.

Alternative View of Hypercubes



Construction of hypercubes from hypercubes of lower dimension

Difficulty of Constructing Hypercubes

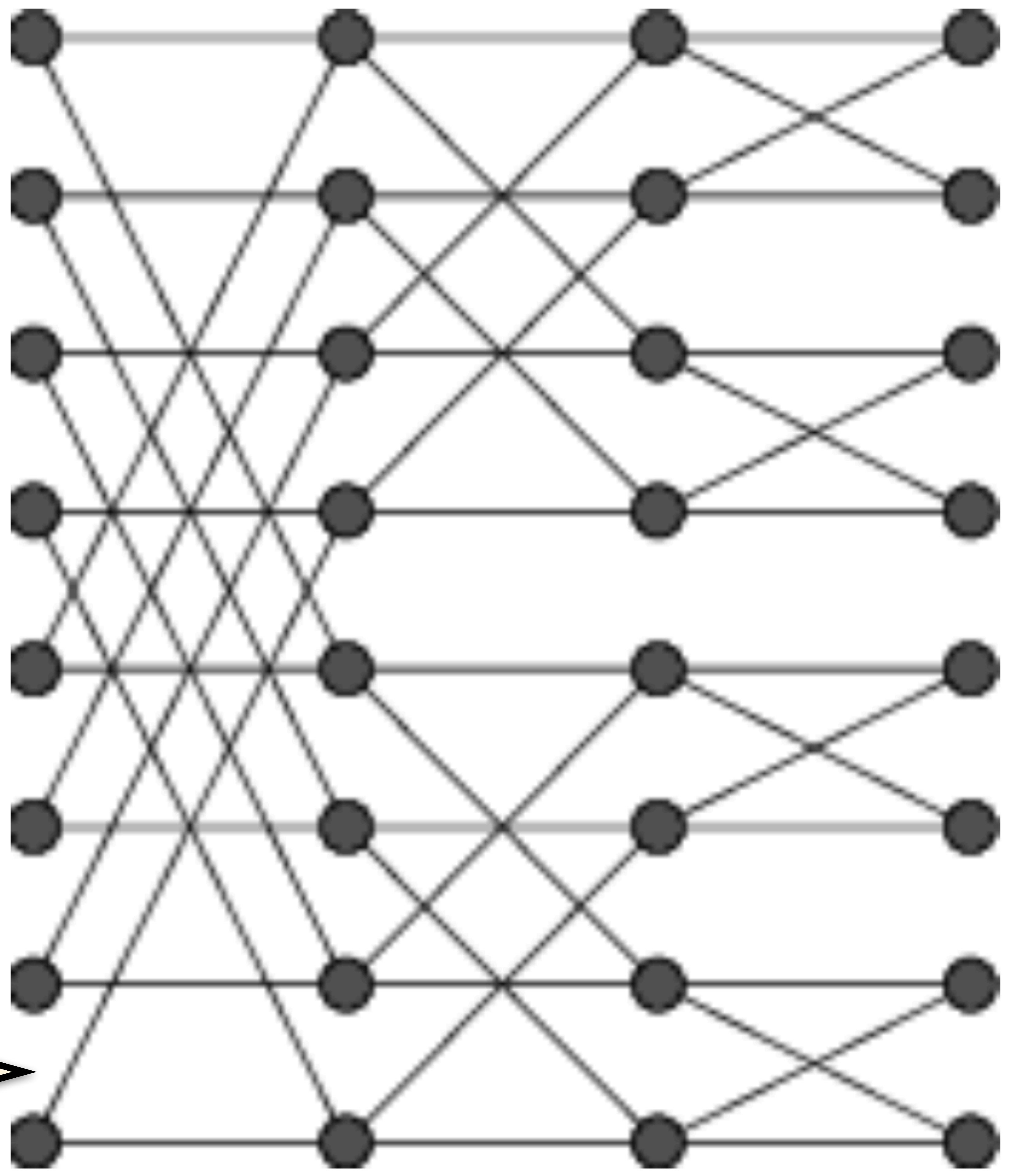
How do we **reuse nodes in smaller dimensional hypercubes** to build larger dimension hypercubes?

- In practice: **set maximum number of dimensions** allowed.

Hypercubes **can't be embedded in physical three-dimensional space** without crisscrossing links and non-uniform length cables.

- Solution: spread the links over **multiple stages**

Static Butterfly Interconnection Network



p log p processors in total: p in each stage, log p stages

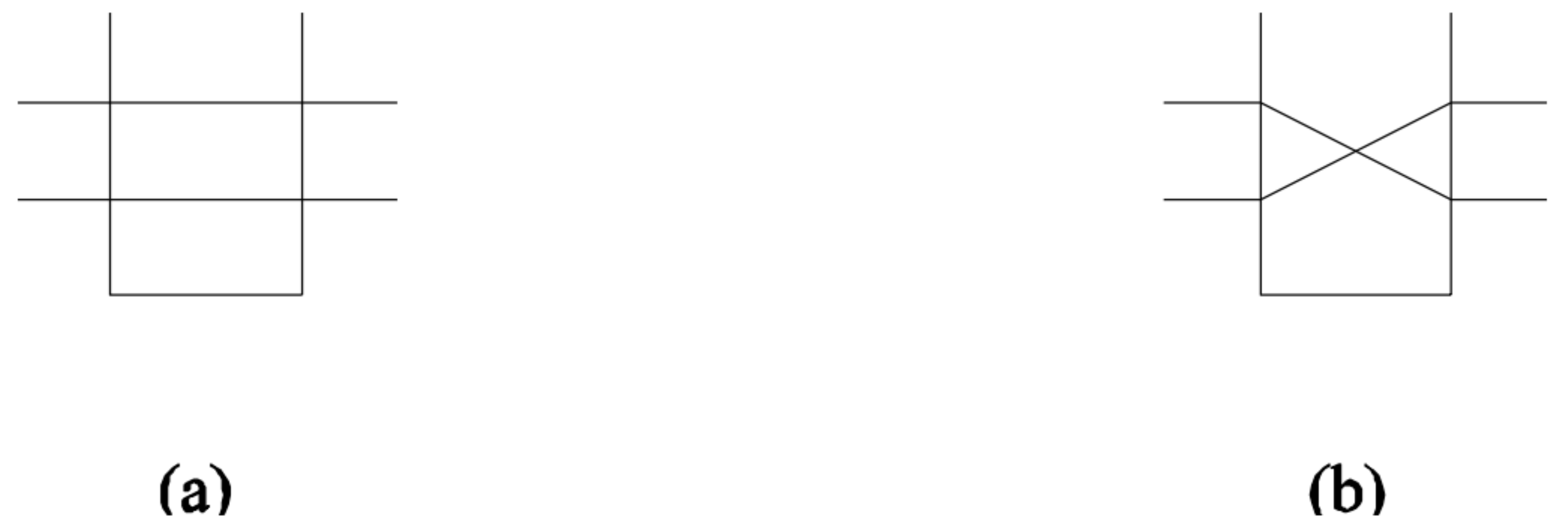
Cross links are hypercube links

Fixed number of links per node

Dynamic Multistage Interconnection Networks

Basic switching element is a 2X2 crossbar

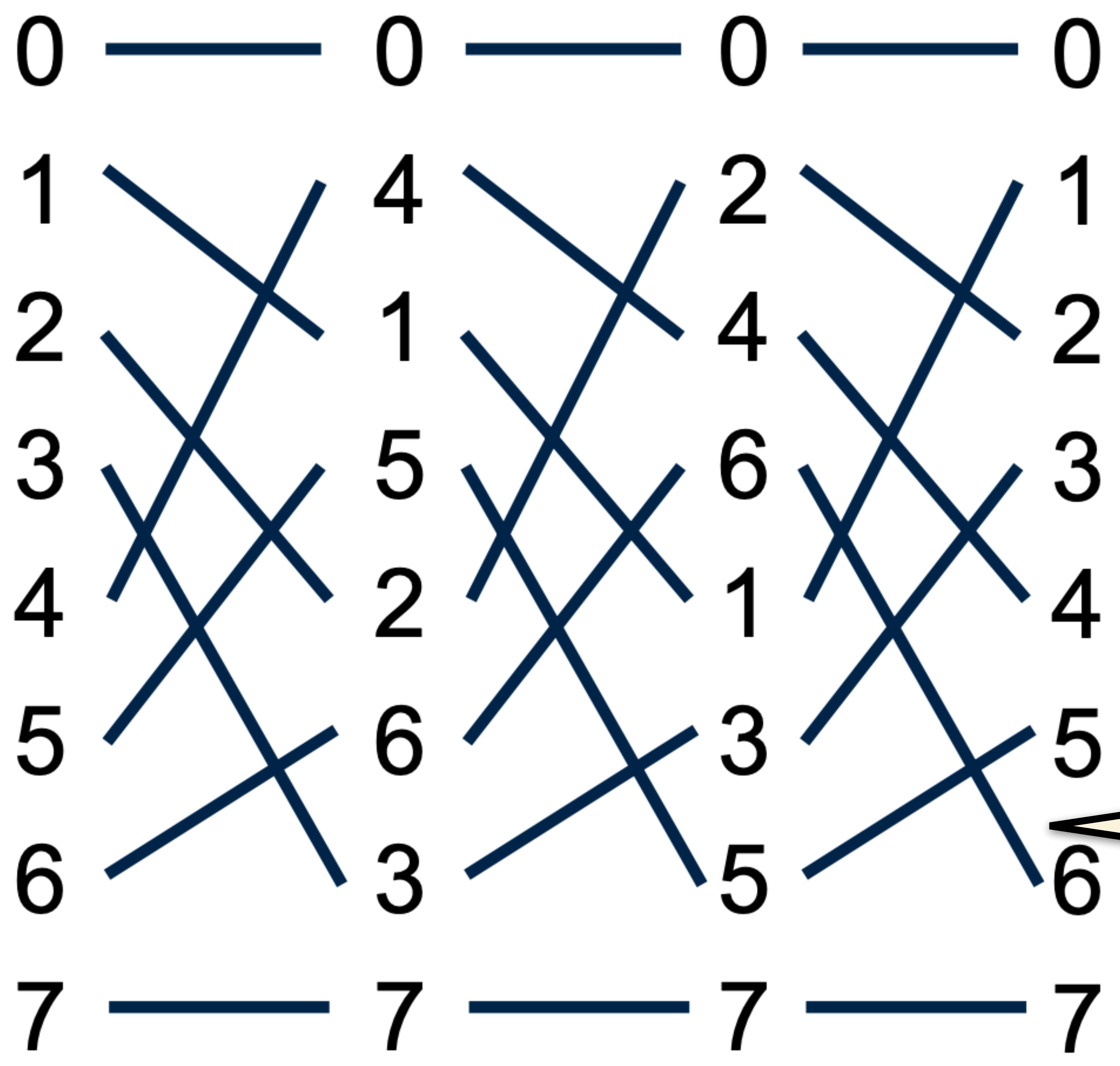
- A crossbar can connect any input to any output
- The switches operate in two modes – crossover or passthrough.



Two switching configurations of the 2 × 2 switch: (a) Pass-through; (b) Cross-over.

Perfect Shuffle

Goal: Route arbitrary hypercubic permutations efficiently



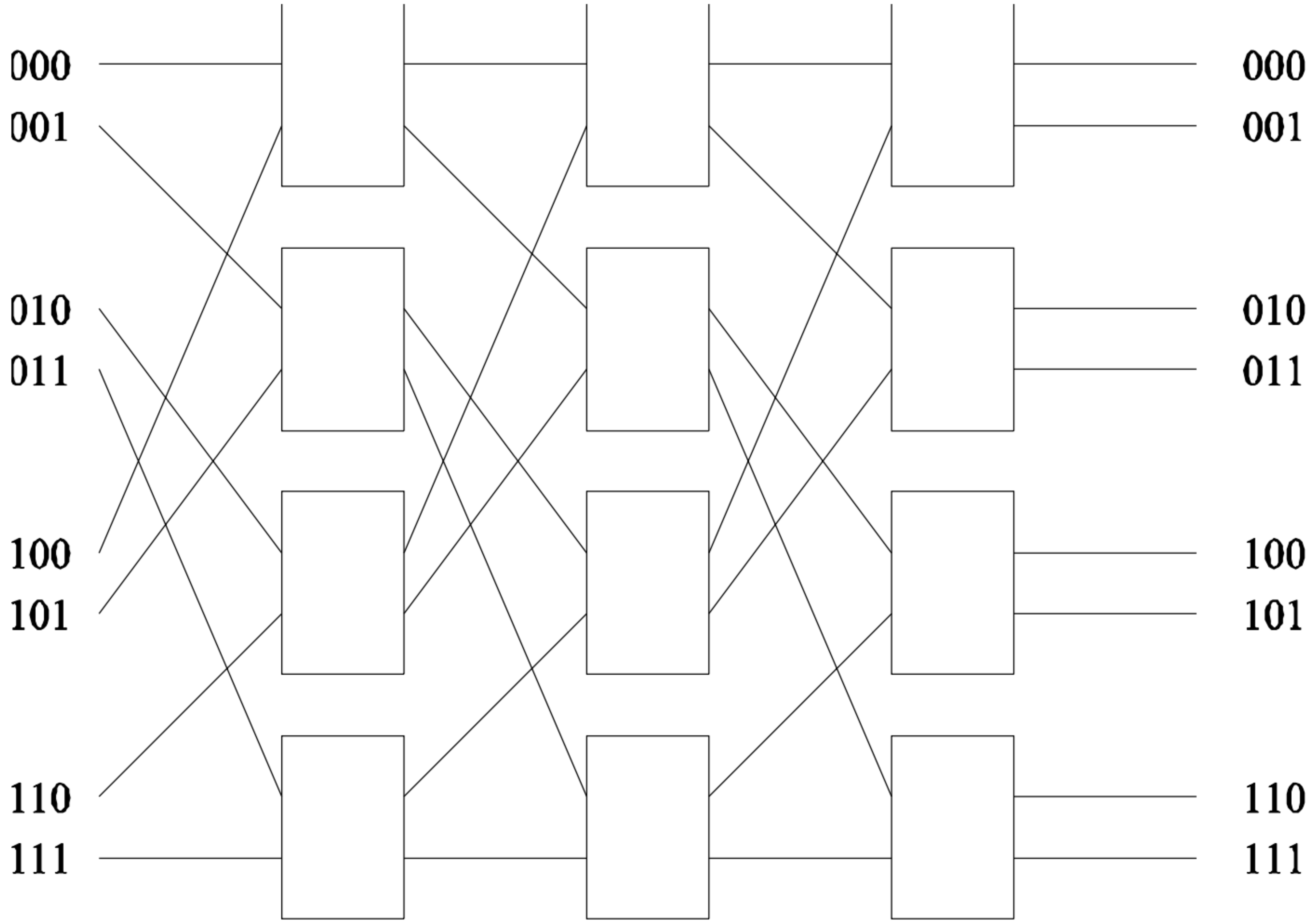
Communication pattern is the same across stages

L	R	L	R	L	R
0	4	0	2	0	1
1	5	4	6	2	3
2	6	1	3	4	5
3	7	5	7	6	7

take 1 from the left and 1 from the right in stages

Omega Network

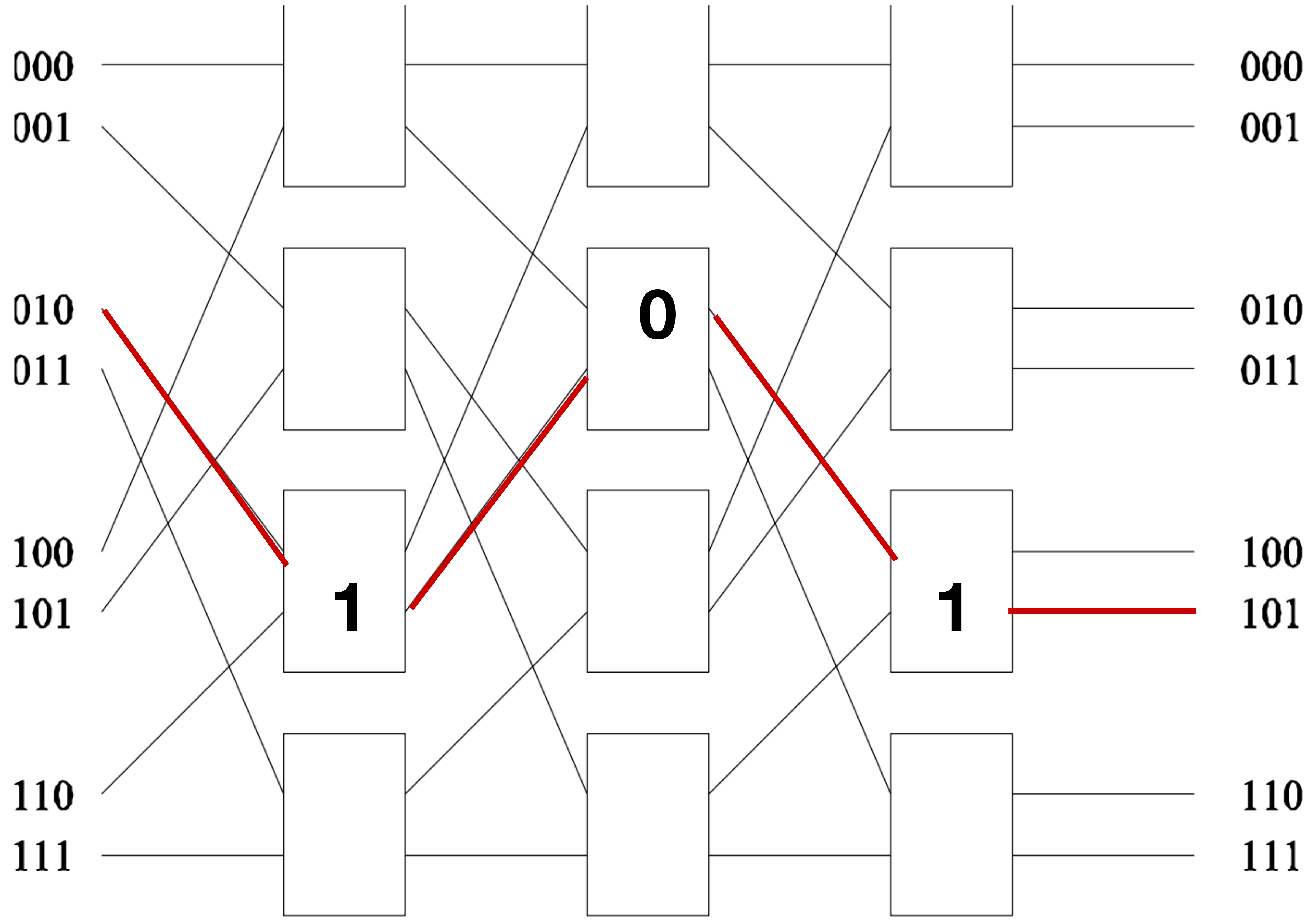
- An omega network has *p/2* × *log p* switching nodes, and the cost of such a network grows as O*(p log p).*



Routing in an Omega Network

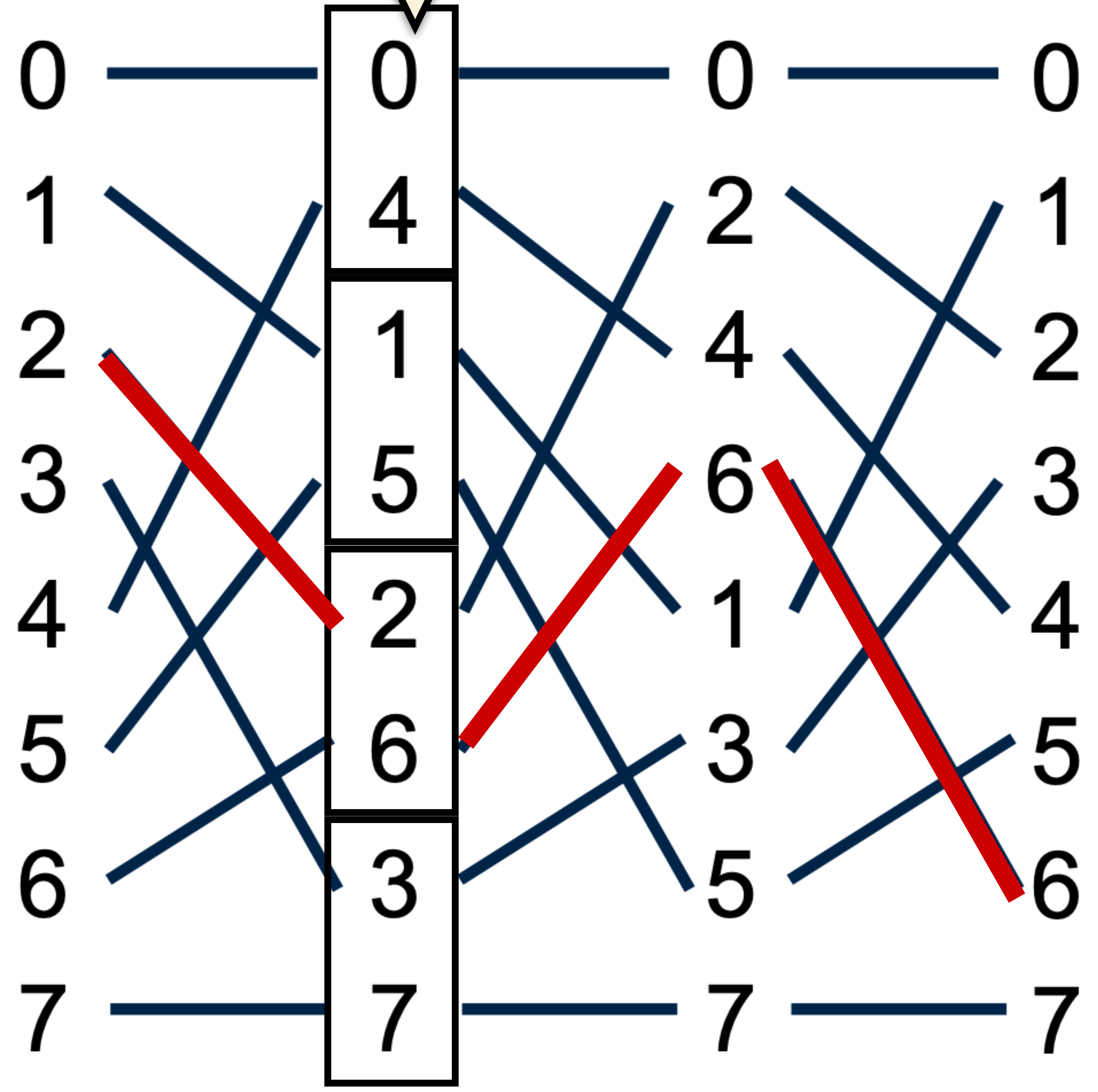
- Destination Tag Algorithm (DTA) – consider bits of destination processor from most significant to least significant
  - Bit is 0: connect to upper output
  - Bit is 1: connect to lower output

Example: 010 to 101



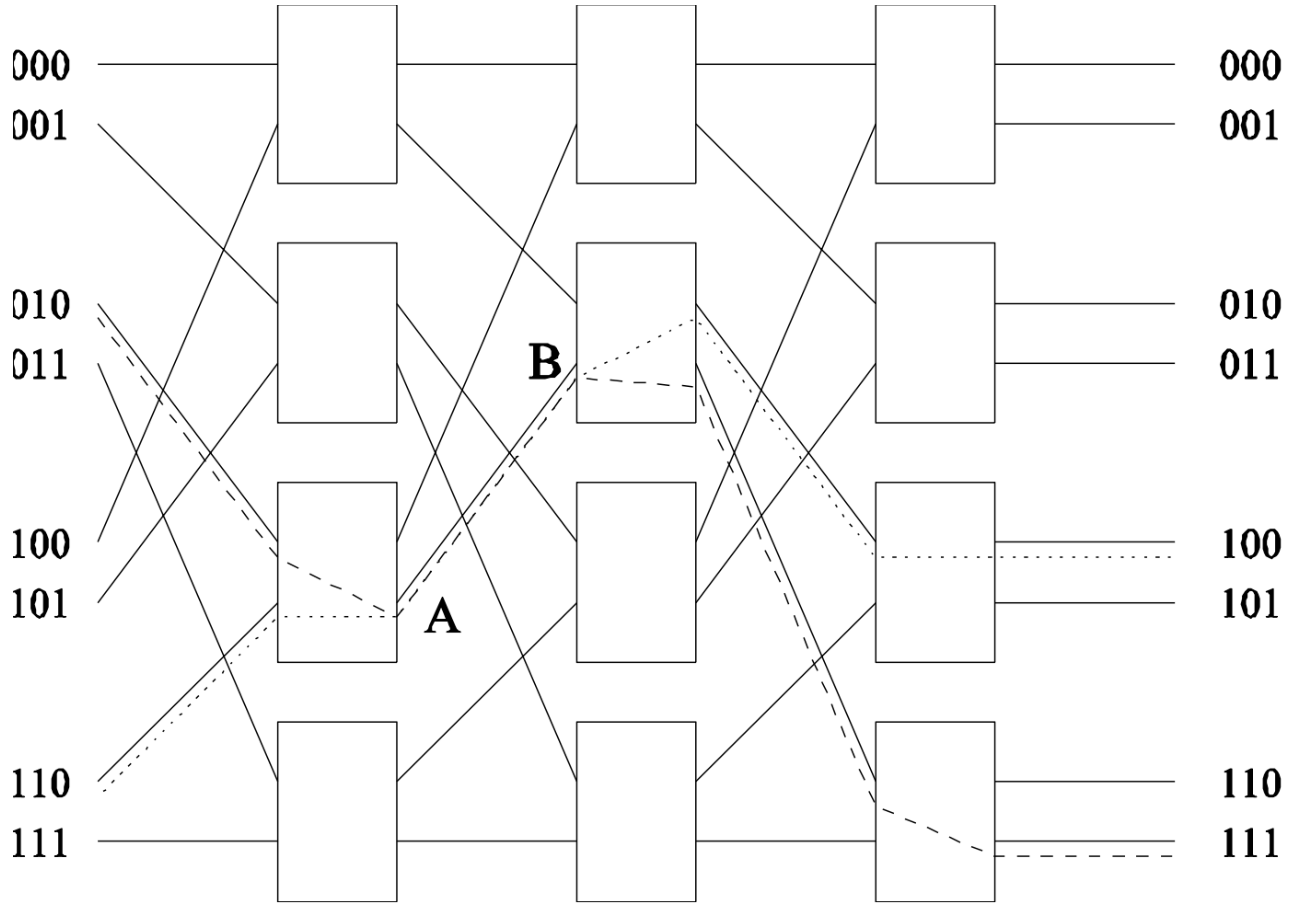
DTA based on Perfect Shuffle

Perfect shuffle brings together pairs of elements with flipped MSB



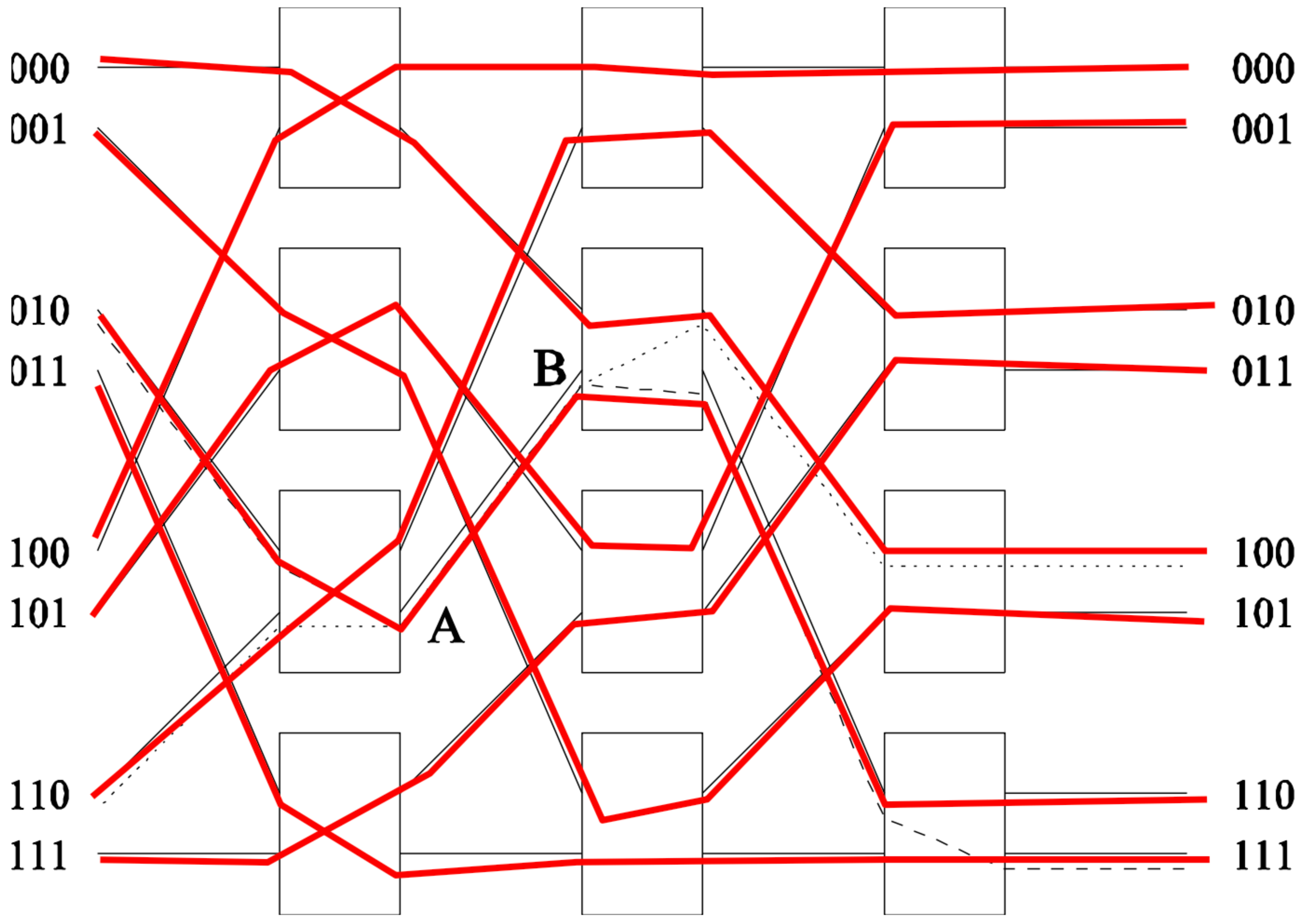
L	R	L	R	L	R
0	4	0	2	0	1
1	5	4	6	2	3
2	6	1	3	4	5
3	7	5	7	6	7

Not Every Permutation Can be Supported



An example of congestion in omega network: one of the messages 010 → 111 and 110 → 100 is blocked at link AB.

Omega Network - Hypercubic Permutations



Omega multistage interconnection network can route any hypercubic permutation without congestion.