CSCI 5451: Introduction to Parallel Computing

Lecture 3: Parallel Architectures (cont'd)

computer science
& engineering

University of Minnesota
Driven to Discover®

Lecture Overview

- ❑ Recap
- ❑ Multi-Stage Networks (in-depth)
- ❑ Network Topology
- ❑ Memory & Moving Bits
    - Memory Basics (Bandwidth, Caching, Latency, Prefetching, etc.)
    - Cache Coherence

Lecture Overview

- ❑ **Recap**
- ❑ Multi-Stage Networks (in-depth)
- ❑ Network Topology
- ❑ Memory & Moving Bits
    - Memory Basics (Bandwidth, Caching, Latency, Prefetching, etc.)
    - Cache Coherence

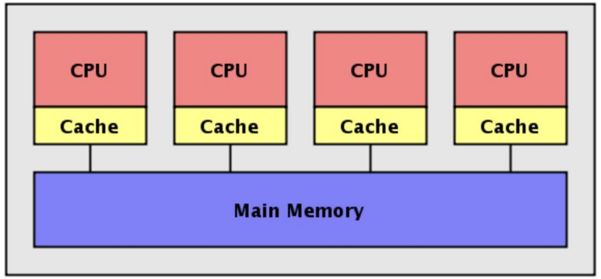
Logical Parallel Hardware Models

Flynn's Taxonomy

Data streams

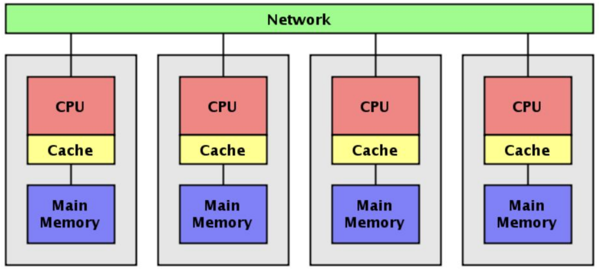
Instructions streams	single	multiple
single	SISD	SIMD
multiple	MISD	MIMD

Shared



Source: Kaminsky/Parallel Java

Distributed

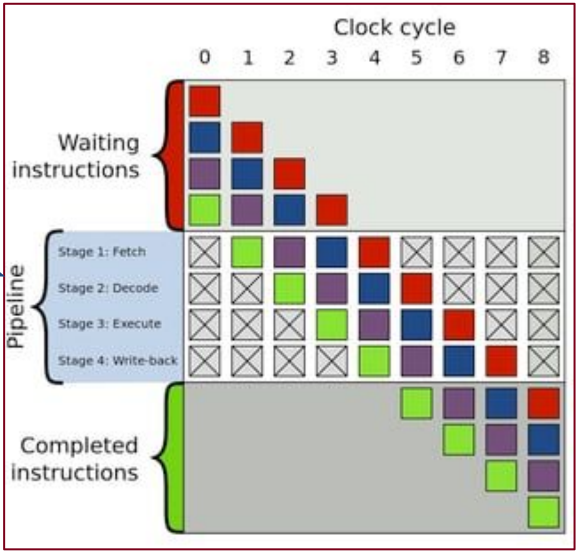


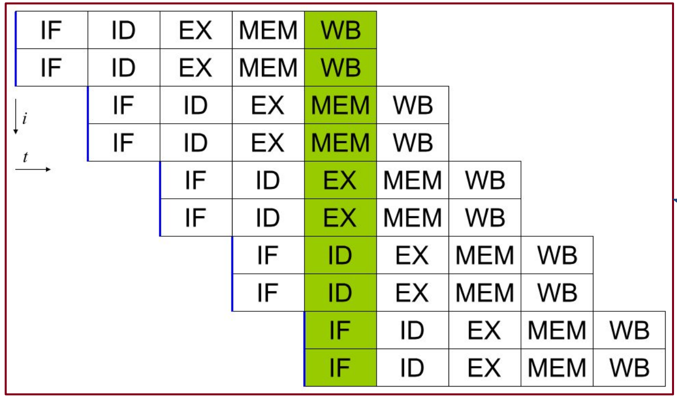
Source: Kaminsky/Parallel Java

Superscalar Execution (Implicit Parallelism)

Recap

Pipelining





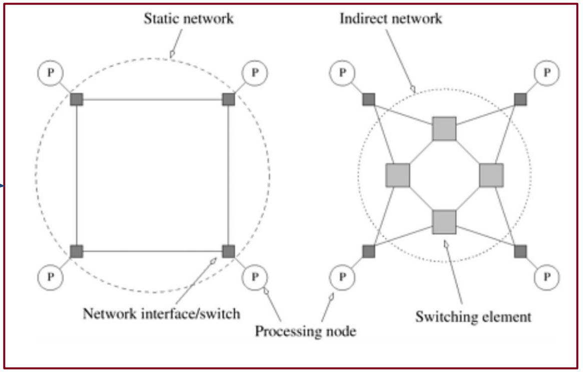
IF	ID	EX	MEM	WB				
IF	ID	EX	MEM	WB				
	IF	ID	EX	MEM	WB			
	IF	ID	EX	MEM	WB			
		IF	ID	EX	MEM	WB		
		IF	ID	EX	MEM	WB		
			IF	ID	EX	MEM	WB	
			IF	ID	EX	MEM	WB	
				IF	ID	EX	MEM	WB
				IF	ID	EX	MEM	WB

2-Way Superscalar Execution

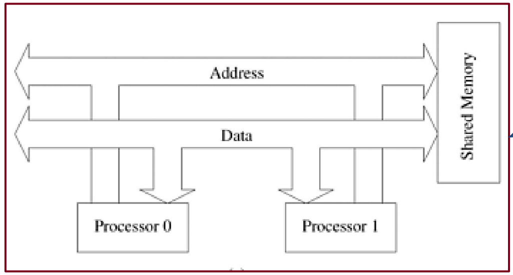
Network Topology Choices

Recap

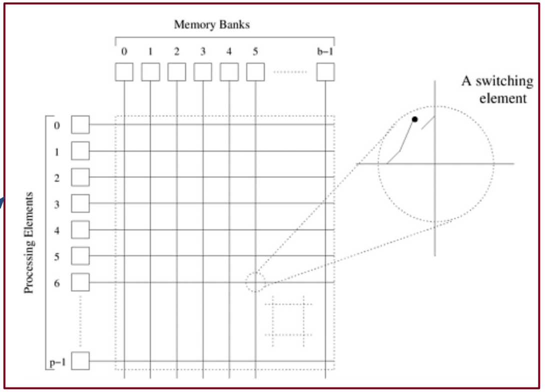
Static vs. Dynamic



Bus



Crossbar



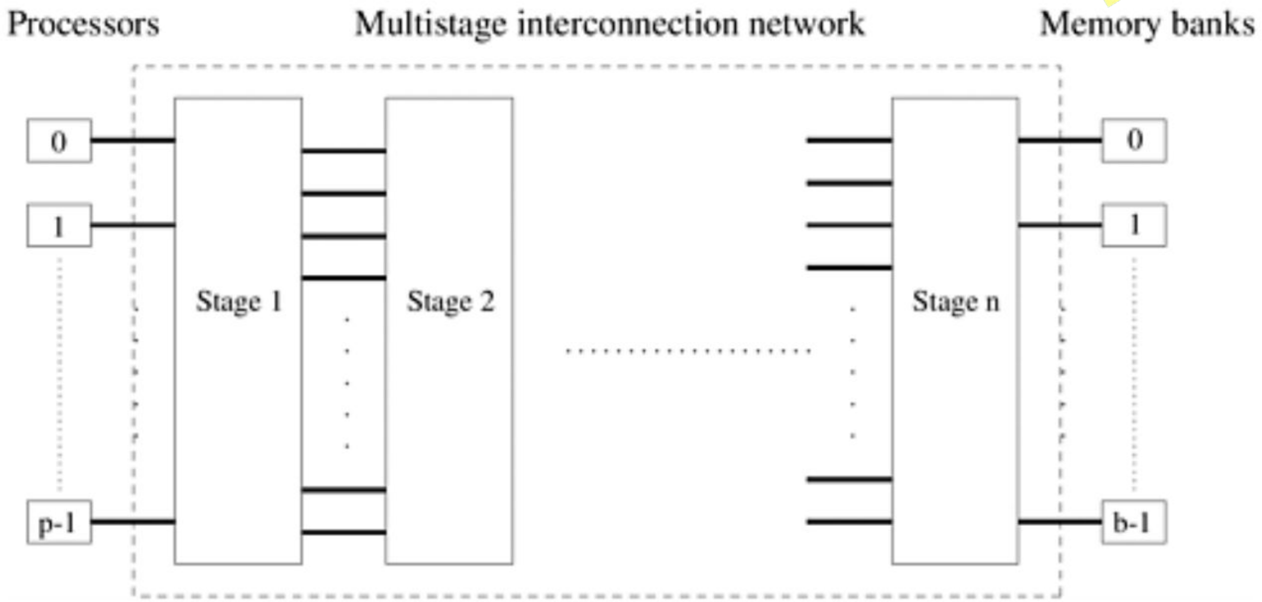
Lecture Overview

- ❑ Recap
- ❑ **Multi-Stage Networks (in-depth)**
- ❑ Network Topology
- ❑ Memory & Moving Bits
  - Memory Basics (Bandwidth, Caching, Latency, Prefetching, etc.)
  - Cache Coherence

Multi-Stage Networks

Recap

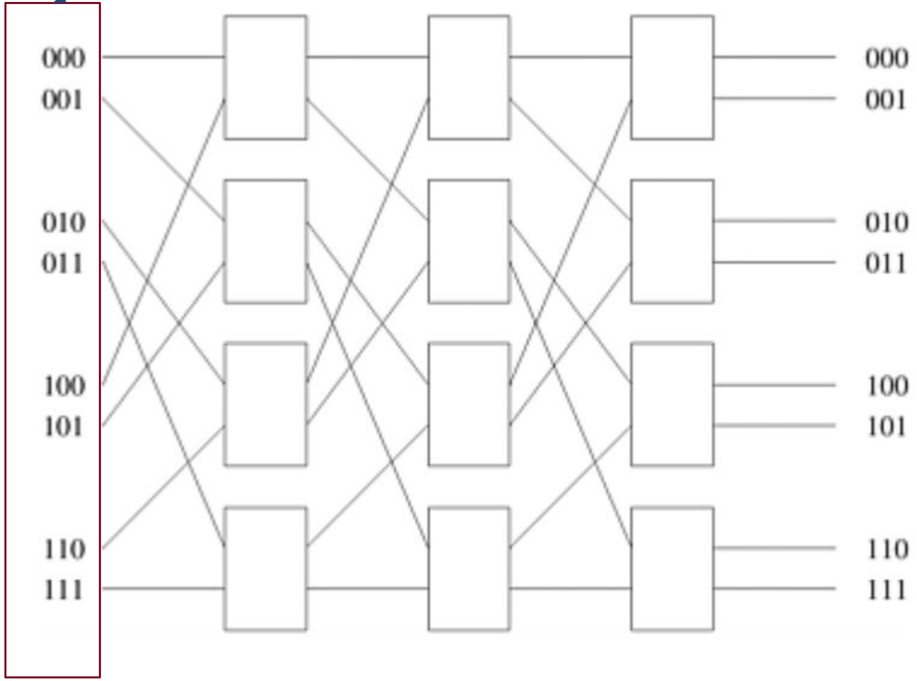
- Shuffles connections at each stage
- Worse than Bus on price, better than Crossbar
- Worse than Crossbar on scalable performance, better than Bus
- Uses log(p) stages



Omega Network

p processors

Processors

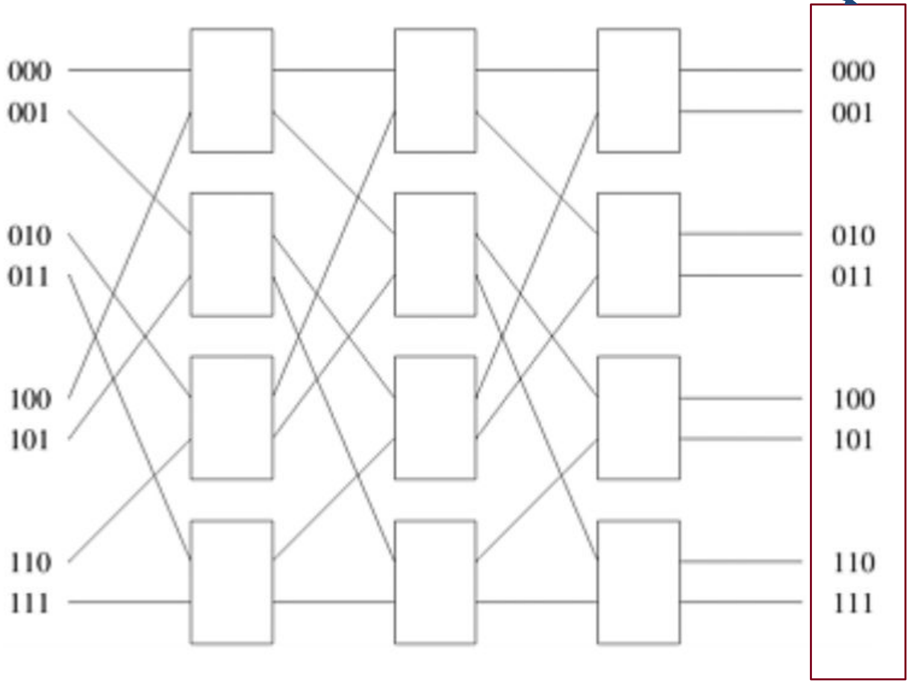


Omega Network

b banks

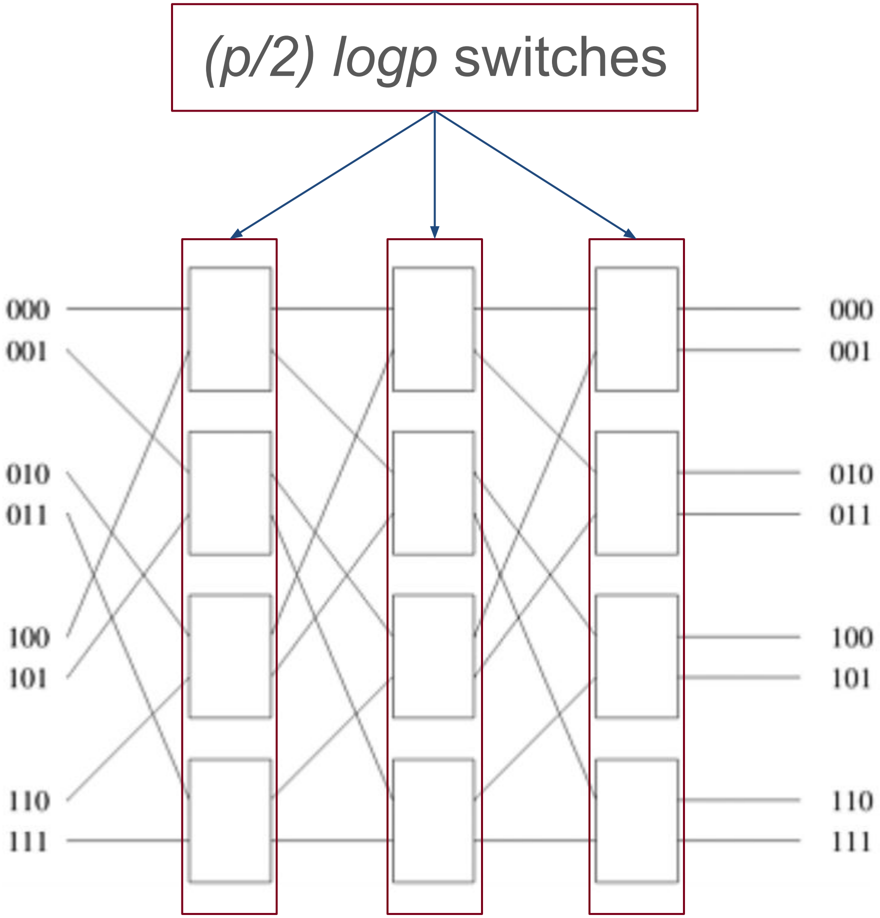
Memory Banks

We make the simplifying assumption that the number of banks is equal to the number of processors



Omega Network

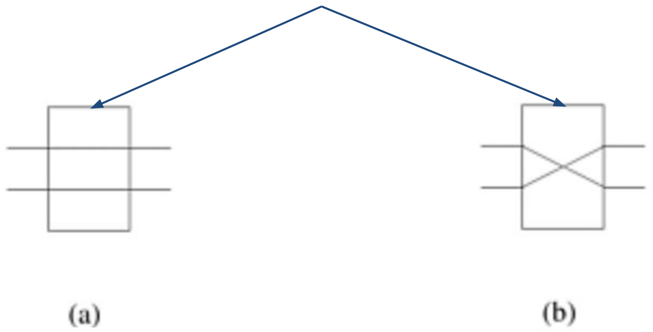
Switches

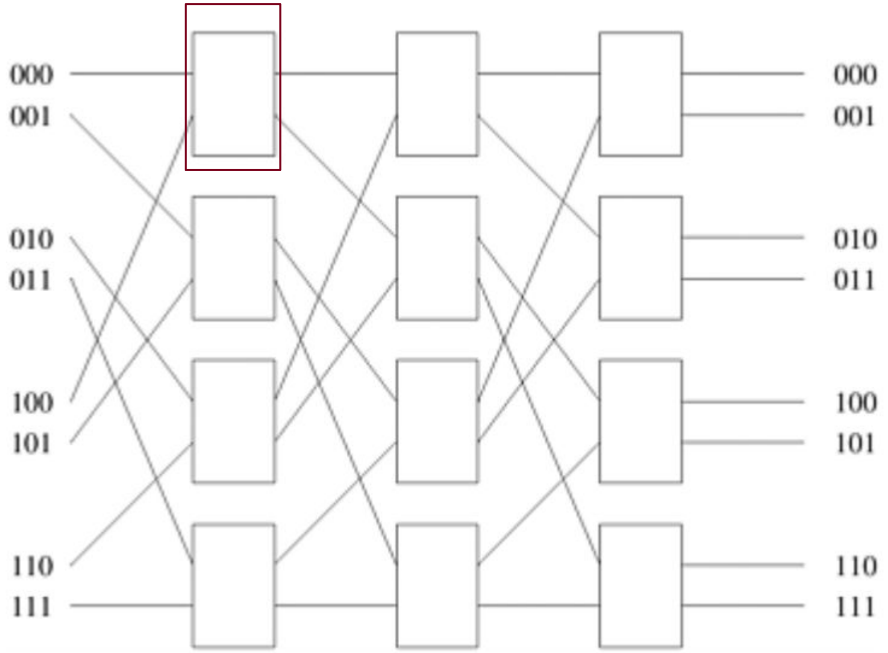


Omega Network

Switch Structure

Switches have 2 inputs, 2 outputs → Inputs can be routed in either direction of the outputs





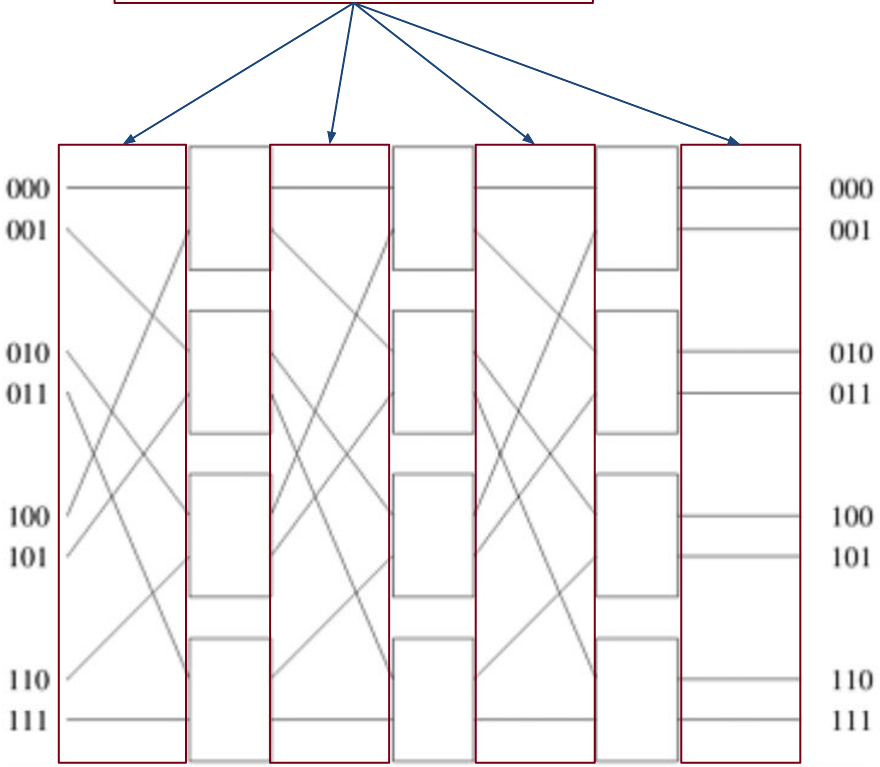
Omega Network

Physical Link Structure

For all but the final layer of links, connecting links follows the below function (where each switch contains 2 inputs). *j* is the output index and *i* is the input

$$j = \begin{cases} 2i, & 0 \le i \le p/2 - 1 \\ 2i + 1 - p, & p/2 \le i \le p - 1 \end{cases}$$

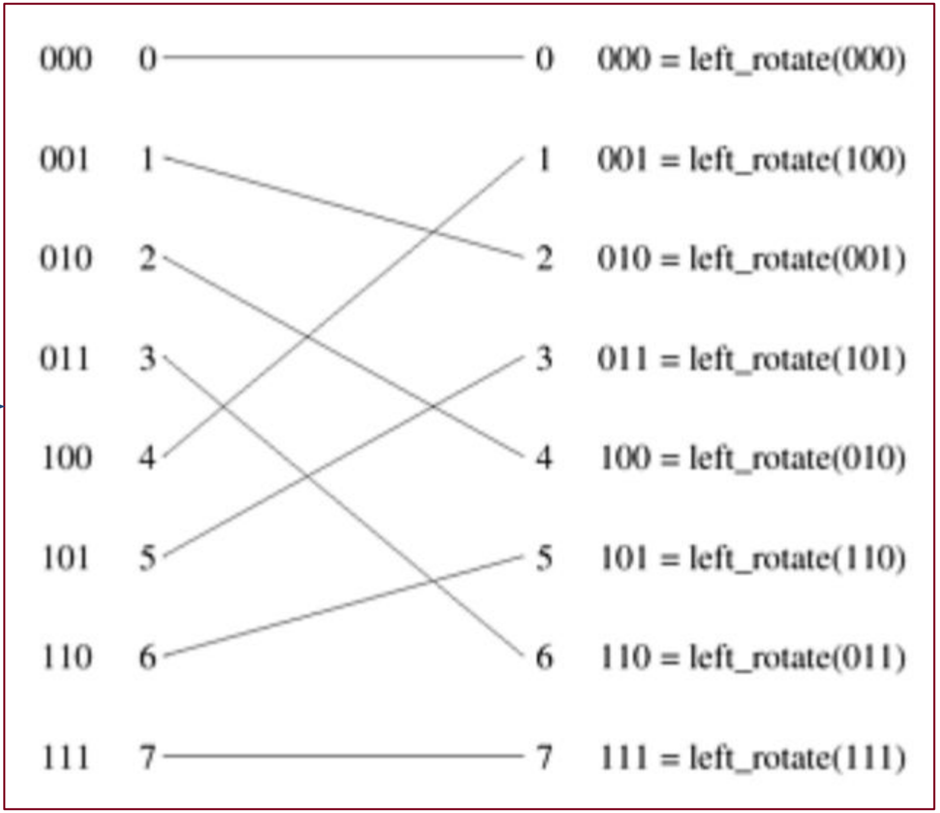
p(logp + 1) Links



Omega Network

This interconnection pattern is also called a

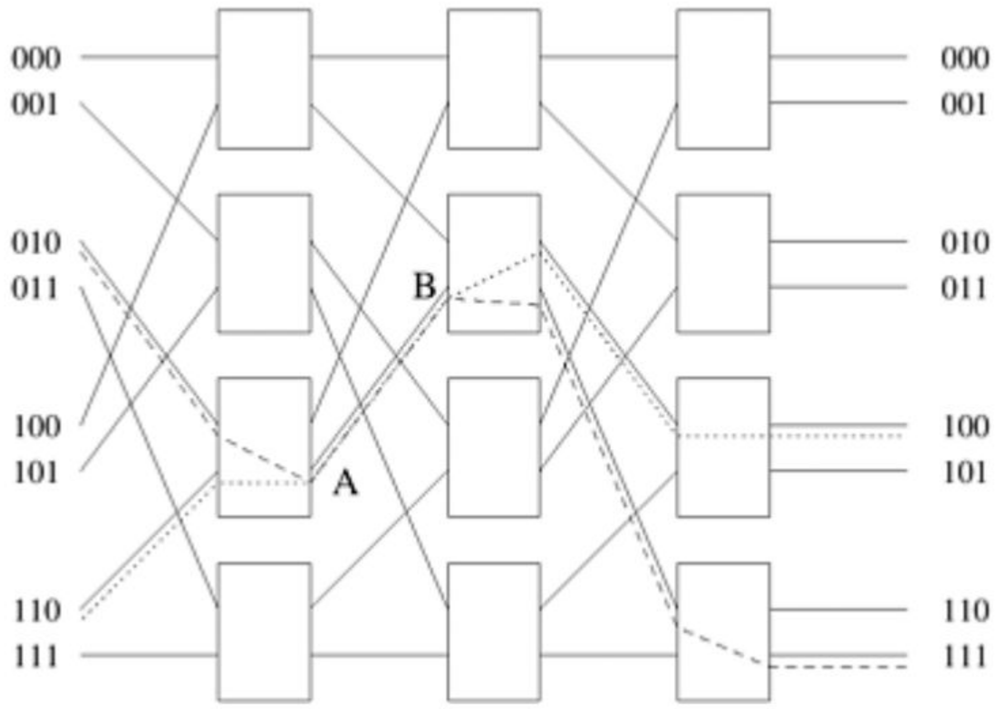
Perfect Shuffle



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)

Omega Network (Example)

Note: Only one possible path for A or B to go from their respective processors to the appropriate memory banks



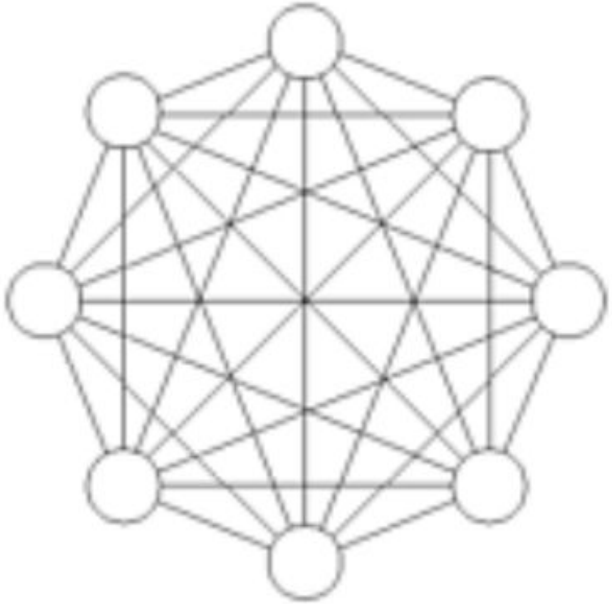
In Class Example:
4 processes to 4 banks Omega Network

Lecture Overview

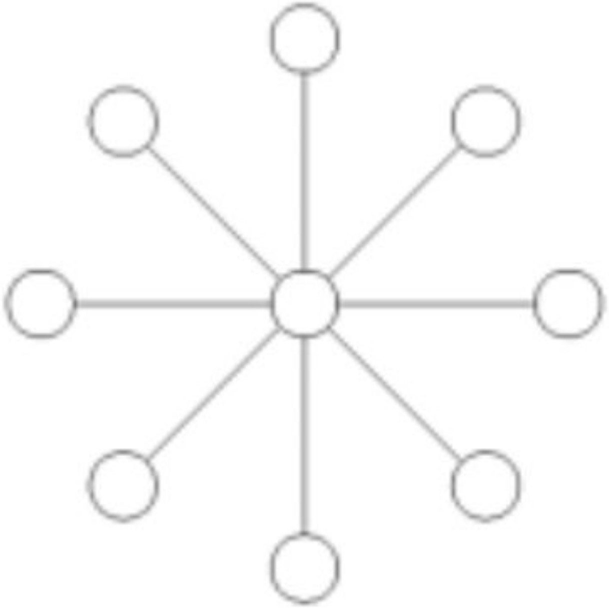
- ❑ Recap
- ❑ Multi-Stage Networks (in-depth)
- ❑ **Network Topology**
- ❑ Memory & Moving Bits
    - Memory Basics (Bandwidth, Caching, Latency, Prefetching, etc.)
    - Cache Coherence

How to connect processors to one another?

Processor Connections

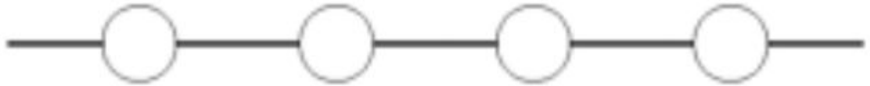


Fully Connected

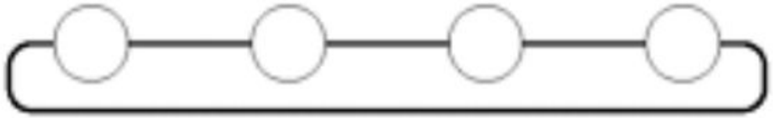


Star

Processor Connections

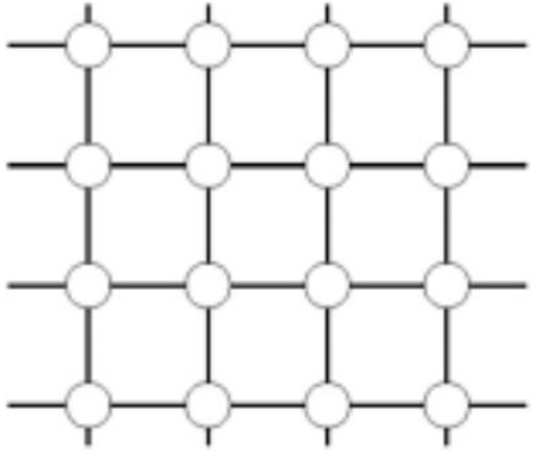


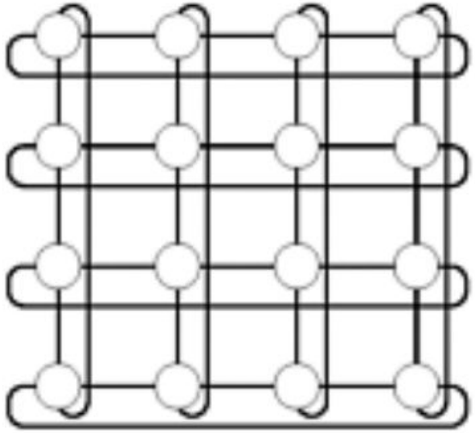
Linear Array
w/o Wraparound

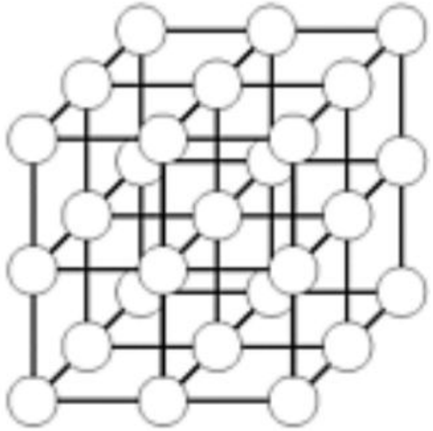


Linear Array
w/Wraparound

Processor Connections

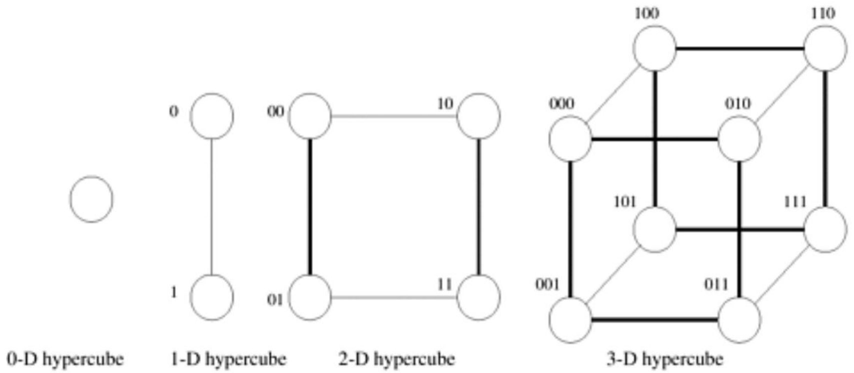

2-d Mesh
w/o Wraparound

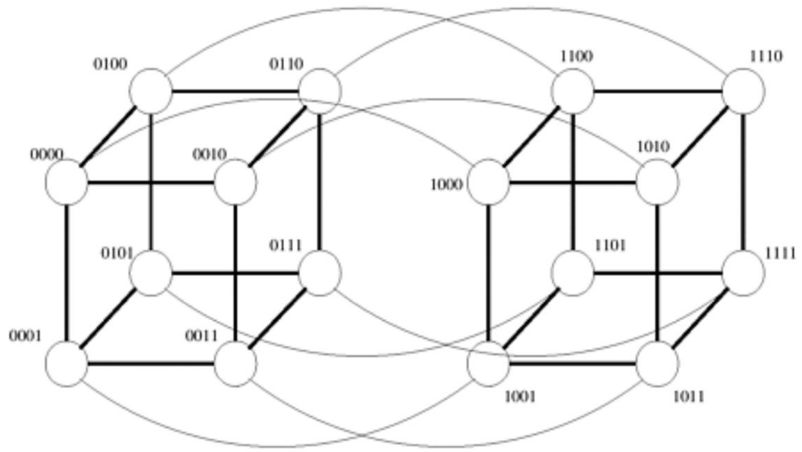

2-d Mesh
w/Wraparound
(2-d torus)


3-d Mesh
w/o Wraparound

Processor Connections

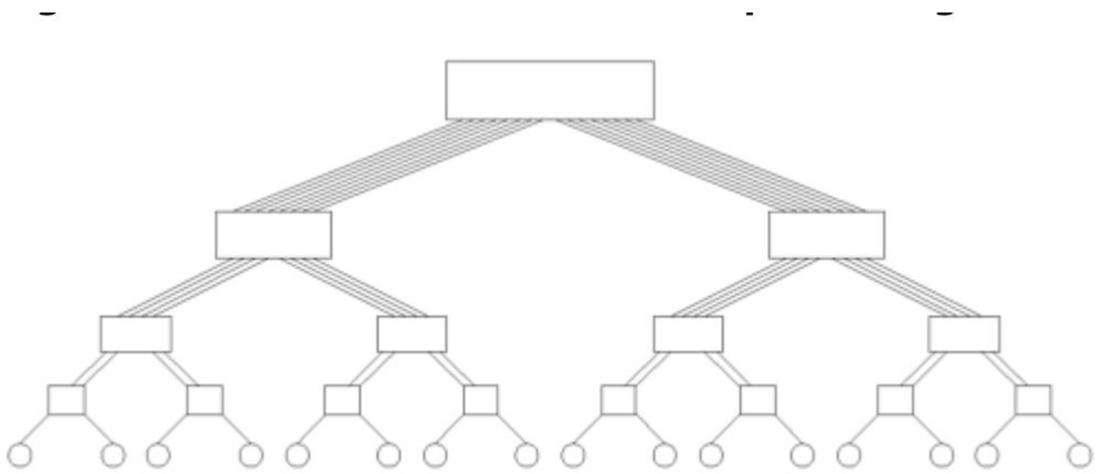
Hypercube Construction





Processor Connections

Fat Tree



Processor Connections in the Limit

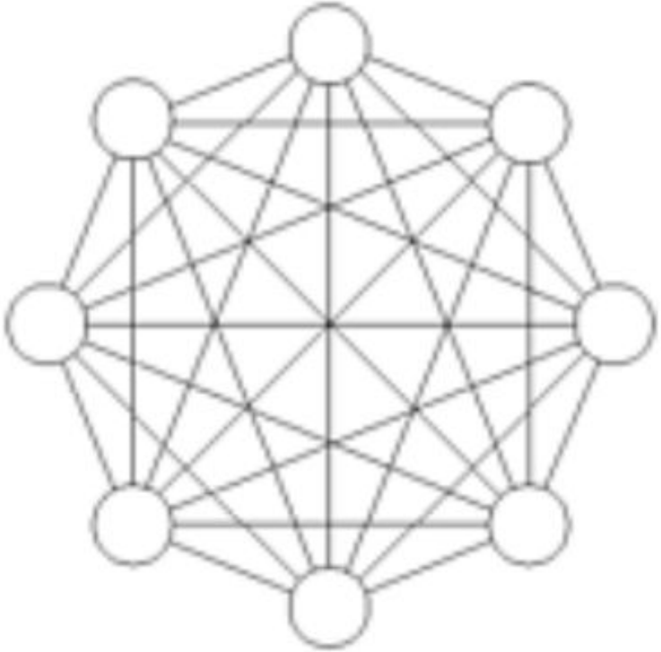
- What are the limits of a parallel computer? How complex do networks get in reality?
- Organizations (Companies/Governments)
  - Modern Supercomputers (Frontier (DoE), Aurora (DoE), Hyperion (Meta), Dojo (Tesla)) will not fit cleanly into the above categories
  - Nodes/switches/links/memory will be added/removed over time
  - Energy/Application requirements may change
- Distributed Computing (SETI@Home, PrimeIntellect, etc.)
  - Perform jobs at massive scale with consumer hardware
  - Users can volunteer their own hardware
  - Topologies are wildly different and change incredibly frequently

Network Evaluation

- ❑ Diameter → The maximum distance between any two nodes in the network, where the distance between any two nodes is defined as the shortest path between them
- ❑ Connectivity → Minimum number of links which must be cut to split the network in two
- ❑ Bisection width → Minimum number of links which must be cut to ensure that the network is partitioned into two equal halves.
- ❑ Cost (no of links) → How many links are required for this topology?

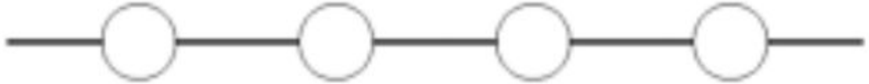
Example : Fully Connected

Network	Diameter	Bisection Width	Arc Connectivity	Cost (No. of links)
Completely-connected	1	$p^2/4$	$p-1$	$p(p-1)/2$



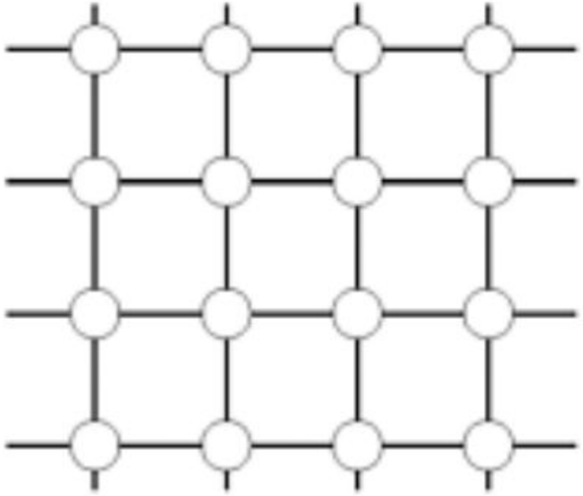
Example Linear Array w/o Wraparound

Network	Diameter	Bisection Width	Arc Connectivity	Cost (No. of links)
Linear array	$p-1$	1	1	$p-1$



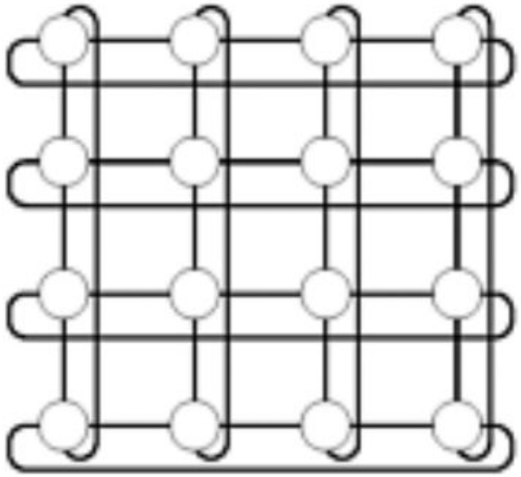
Example: 2-d Mesh w/o Wraparound

Network	Diameter	Bisection Width	Arc Connectivity	Cost (No. of links)
2-D mesh, no wraparound	$2(\sqrt{p}-1)$	$\sqrt{p}$	2	$2(p-\sqrt{p})$



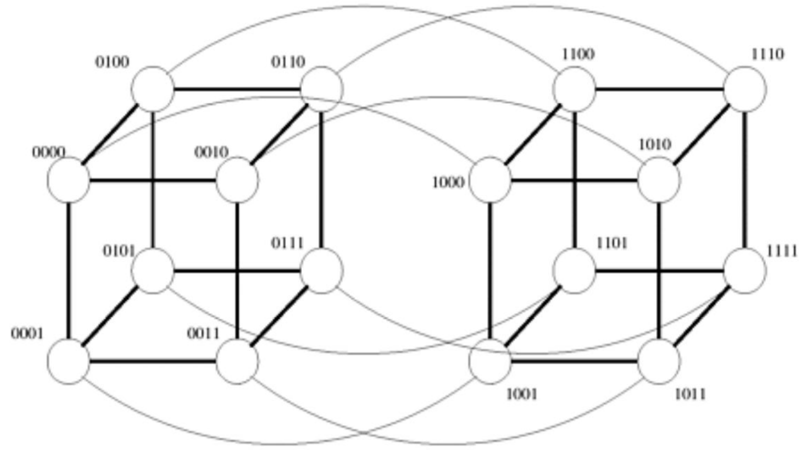
Example: 2-d Mesh w/Wraparound

Network	Diameter	Bisection Width	Arc Connectivity	Cost (No. of links)
2-D wraparound mesh	$2\lfloor\sqrt{p}/2\rfloor$	$2\sqrt{p}$	4	$2p$



Example: Hypercube

Network	Diameter	Bisection Width	Arc Connectivity	Cost (No. of links)
Hypercube	$\log p$	$p/2$	$\log p$	$(p \log p)/2$



Static Network Evaluations

- Feel free to explore further on your own

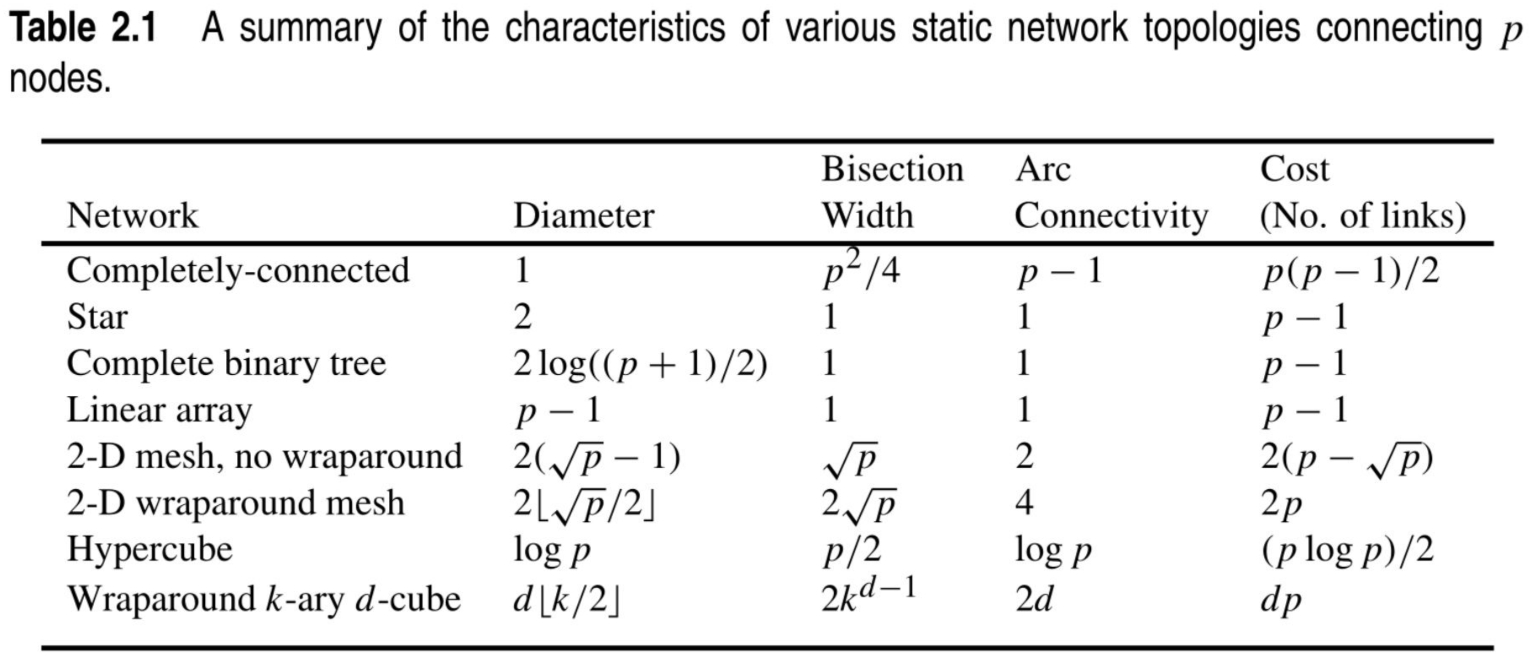


Table 2.1 A summary of the characteristics of various static network topologies connecting p nodes.

Network	Diameter	Bisection Width	Arc Connectivity	Cost (No. of links)
Completely-connected	1	$p^2/4$	$p-1$	$p(p-1)/2$
Star	2	1	1	$p-1$
Complete binary tree	$2\log((p+1)/2)$	1	1	$p-1$
Linear array	$p-1$	1	1	$p-1$
2-D mesh, no wraparound	$2(\sqrt{p}-1)$	$\sqrt{p}$	2	$2(p-\sqrt{p})$
2-D wraparound mesh	$2\lfloor\sqrt{p}/2\rfloor$	$2\sqrt{p}$	4	$2p$
Hypercube	$\log p$	$p/2$	$\log p$	$(p\log p)/2$
Wraparound k-ary d-cube	$d\lfloor k/2\rfloor$	$2k^{d-1}$	$2d$	dp

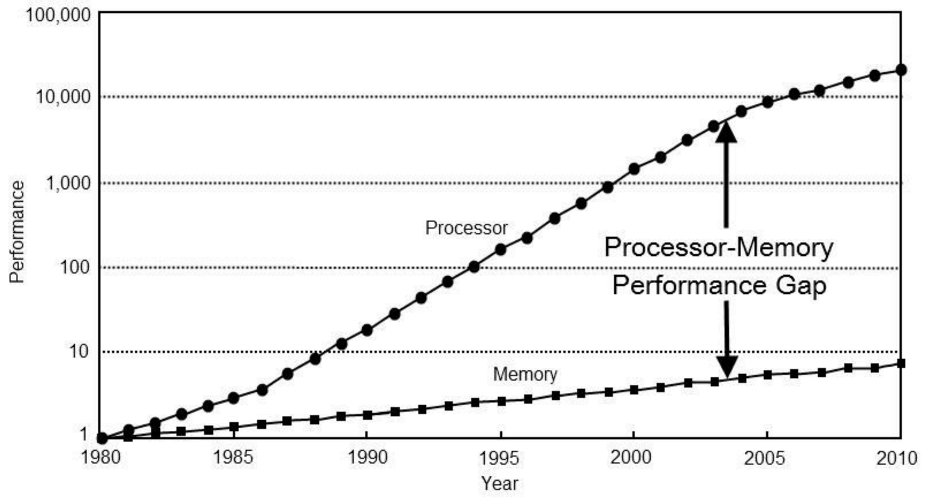
Lecture Overview

- ❑ Recap
- ❑ Multi-Stage Networks (in-depth)
- ❑ Network Topology
- ❑ **Memory & Moving Bits**
    - **Memory Basics (Bandwidth, Caching, Latency, Prefetching, etc.)**
    - Cache Coherence

With processors connected, how can we most intelligently handle memory?

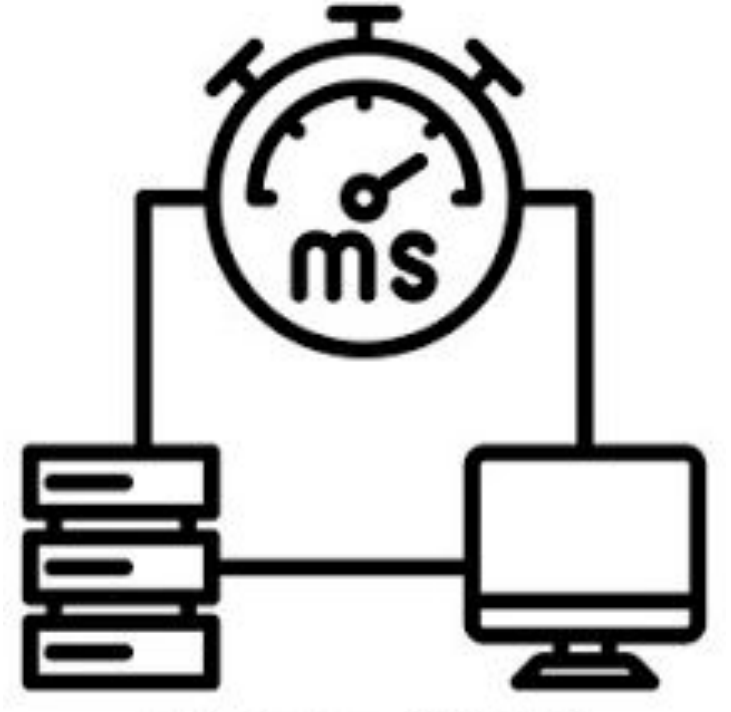
Bandwidth

- The rate data can be pumped from memory to the processor
- Analogous to the number of lanes on a highway which has no traffic
- Is not increasing nearly as fast as processor speeds $\longrightarrow$ Many bottlenecks in computing are memory-bound now



Latency

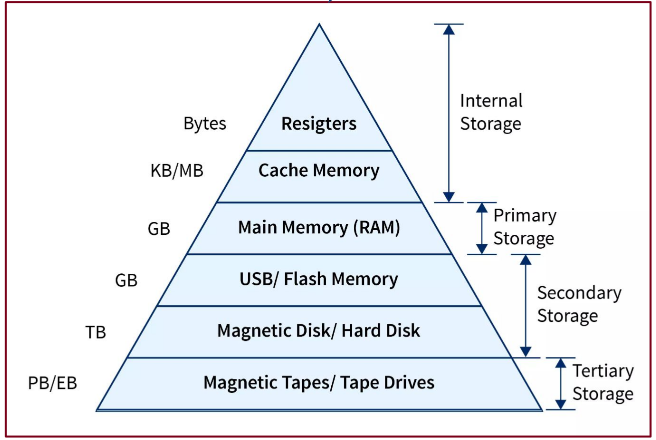
- ❑ How long it takes for a memory request to actually retrieve bits from memory
- ❑ Analogous to the time required to travel on a highway with no traffic between two exact points



Caching

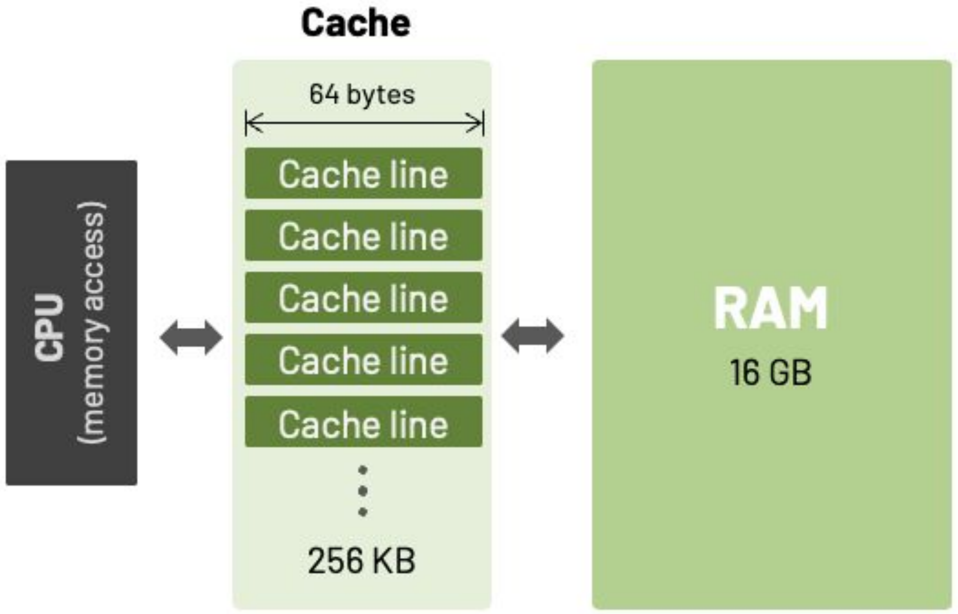
- ❑ Processor speed + DRAM speed mismatches have led to increasing use & complexity of caching
- ❑ Place smaller + faster memory closer/on the Processor
- ❑ Low-latency + high bandwidth storage
- ❑ Sits in between processor and main memory
- ❑ Typically multiple levels (L1, L2, L3)

Memory Hierarchy



Cache Lines

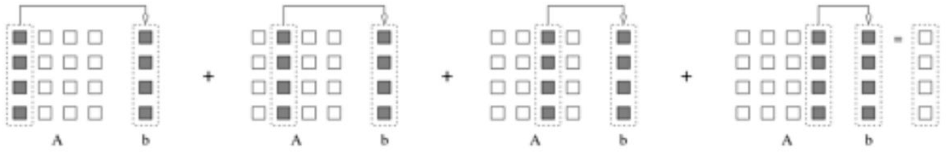
- Caches do not load single elements from memory
- They load *lines* from memory
- Typical lines are the same size across each level of the hierarchy
- Sizes ~64Bytes
- Loading a variable means pulling the whole line that variable is on from memory



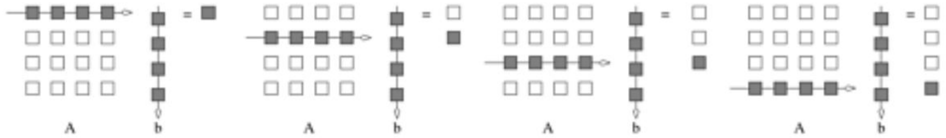
Exploiting Cache

The main method for exploiting the cache is to increase the hit ratio (i.e. The proportion of time)

- Temporal Locality → Repeatedly reference same data element multiple time in a small window
- Spatial Locality → Repeatedly reference nearby elements in memory (makes use of cache lines)



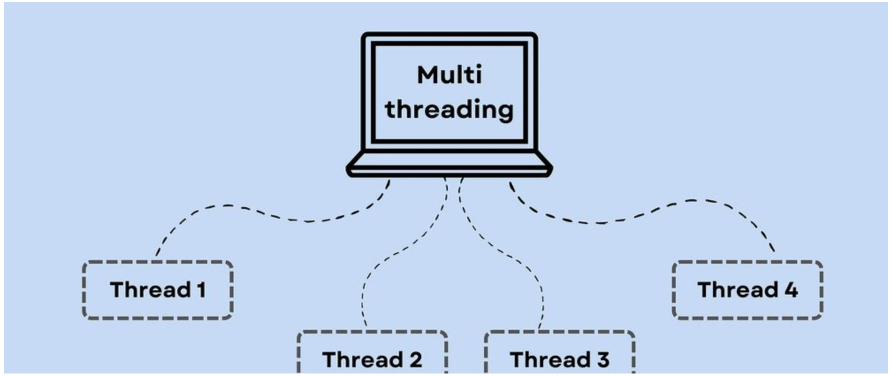
(a) Column major data access



(b) Row major data access.

Hiding Latency

- Prefetching - If we know in advance what we are going to be loading, then load it into cache earlier.
- Multithreading - Parallelize memory accesses (more feasible today as DRAM today usually permits a relatively large number of concurrent reads)



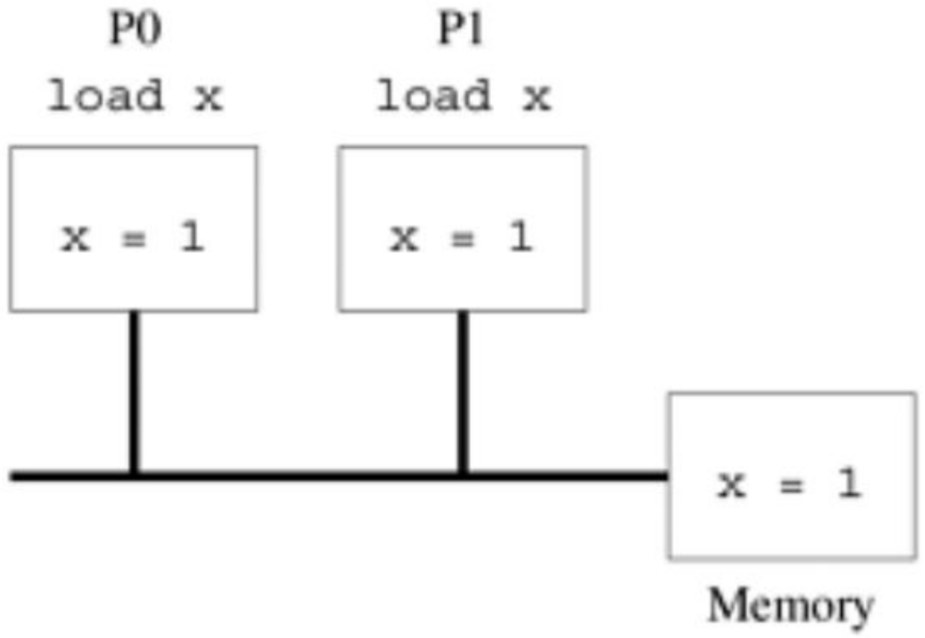
Lecture Overview

- ❑ Recap
- ❑ Multi-Stage Networks (in-depth)
- ❑ Network Topology
- ❑ **Memory & Moving Bits**
  - ○ Memory Basics (Bandwidth, Caching, Latency, Prefetching, etc.)
  - **○ Cache Coherence**

How do we manage states in shared memory systems so that each processes’ cache retains the same value?

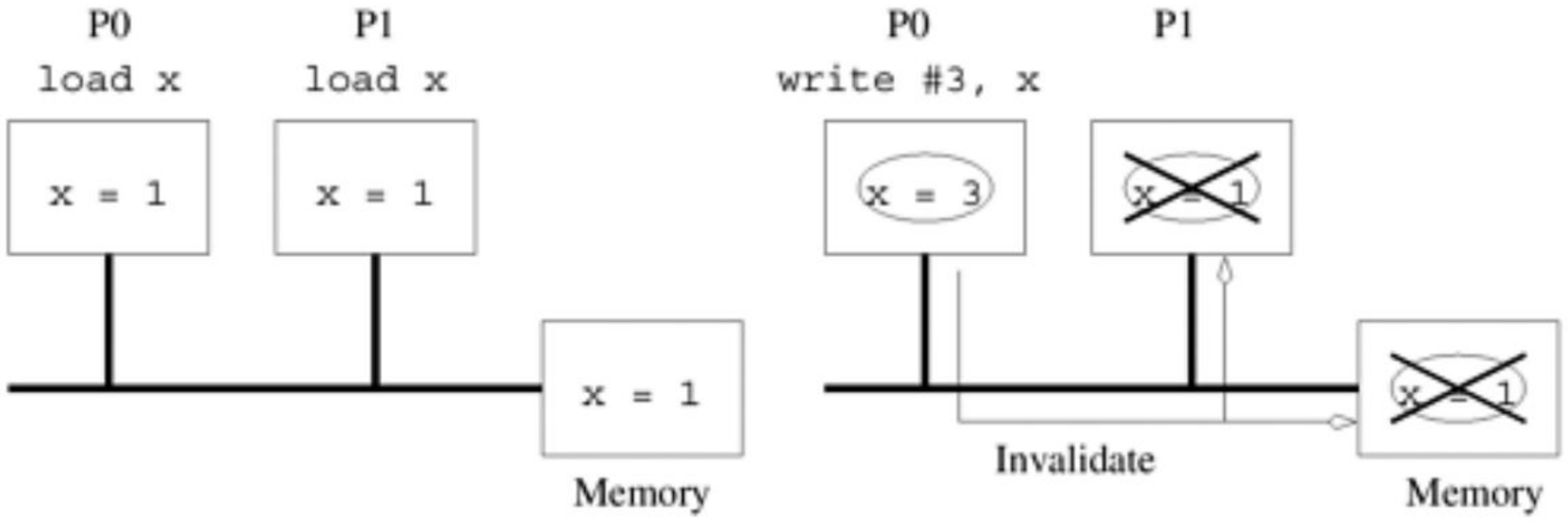
Cache Coherence

- There are three copies across three locations.
  - 1 in main memory
  - 1 in P0 cache
  - 1 in P1 cache
- They must all stay the same in the course of program execution.
- We ignore multiple levels of cache in this simple case here
- The problem is made more complex with multi-level cache



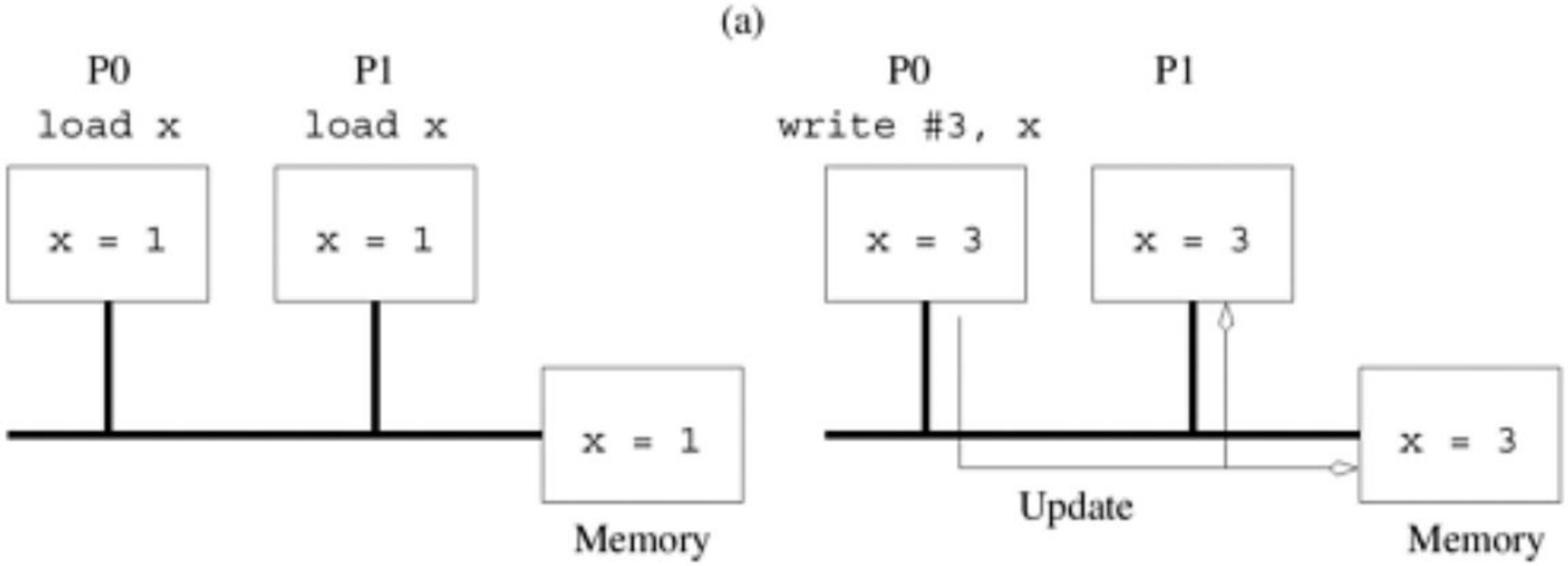
Invalidate Protocol

Similar to lazy loading → Cache lines are marked as invalid on other processors, but not updated until referenced



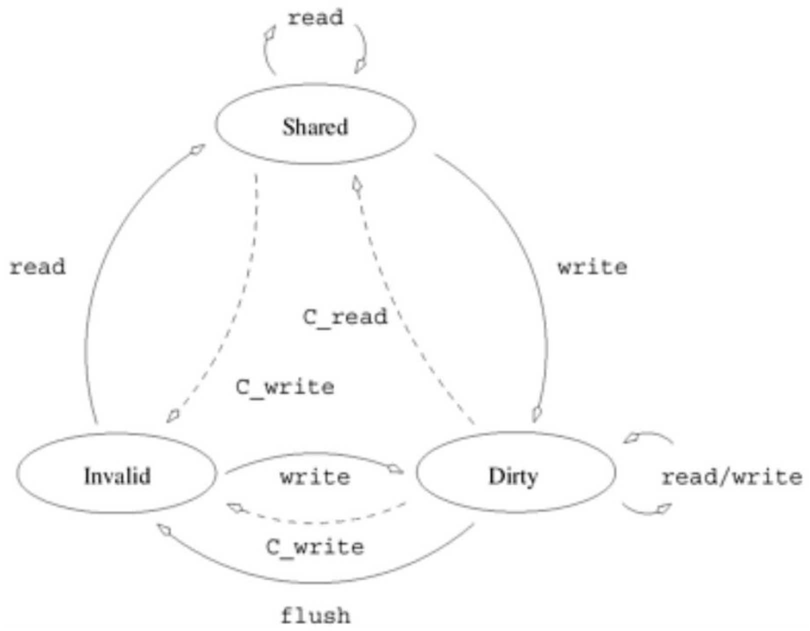
Update Protocol

Similar to eager loading → Cache lines are updated on all other processors as soon as they are updated on one



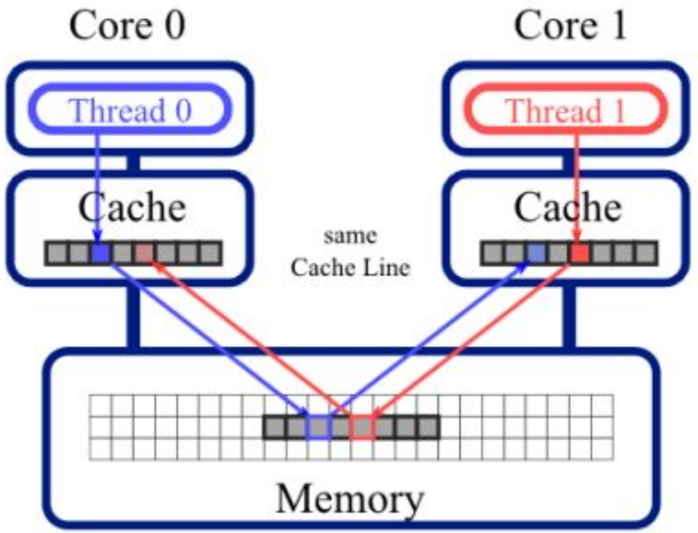
General Three State Protocol

- All cache lines have an a additional state attached to them on each processor
    - Shared → All variables the same
    - Dirty → Processor which updates the variable
    - Invalid → All other processors must mark their line as invalid
- Solid Lines → Processor actions
- Dashed Lines → Coherence actions



False Sharing

- Different variables are updated on the same cache line
- Can lead to excessive overhead even though different things are updated
- KEY TIP: Try to have your threads update different cache lines if possible on CPUs


Core 0
Core 1
Thread 0
Thread 1
Cache
Cache
same
Cache Line
Memory