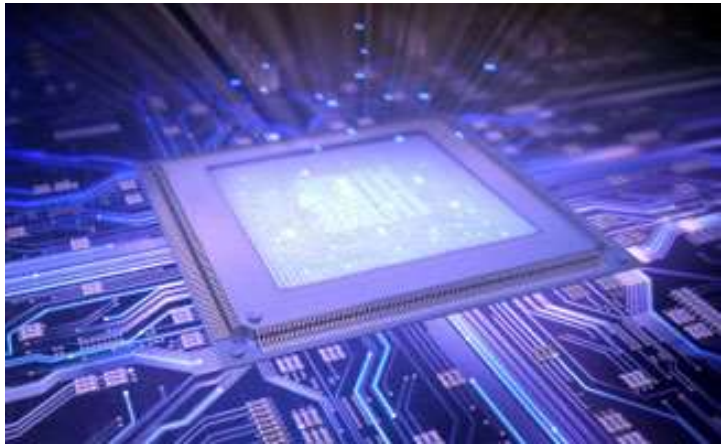
Parallel Computing

Parallel Hardware: Advanced

Mohamed Zahran (aka Z)
mzahran@cs.nyu.edu
http://www.mzahran.com

Some slides are adopted from:

- G. Barlas book
- P. Pacheco book



Last lecture we looked at techniques to exploit ILP (Instruction Level Parallelism)

- Pipelining
- Superscalar
- Out-of-order execution
- Speculative execution
- Simultaneous Multithreading (aka Hyperthreading technology)

All the above require very little, if at all, work from the side of the programmer to make use of.

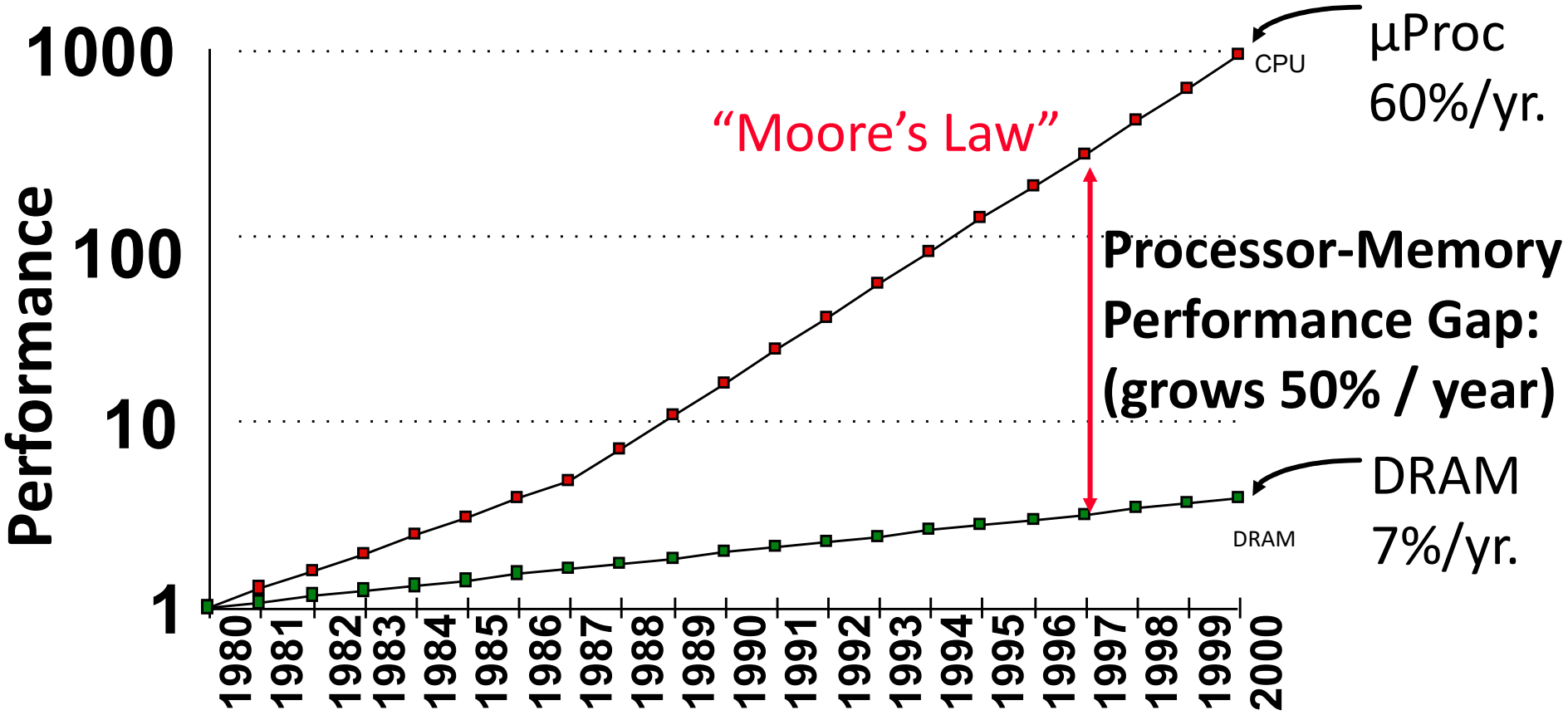
Computer Technology … Historically

- Memory
  - DRAM capacity: 2x / 2 years (since '96);
    64x size improvement in last decade.
- Processor
  - Speed 2x / 1.5 years (since '85); → BUT!!
    100X performance in last decade.
- Disk
  - Capacity: 2x / 1 year (since '97)
    250X size in last decade.

Memory Wall Is Here to Stay

- Memory access is still a big problem in parallel machines.
- Cache coherence (as we will see later) also has large negative effect on performance.

Memory Wall

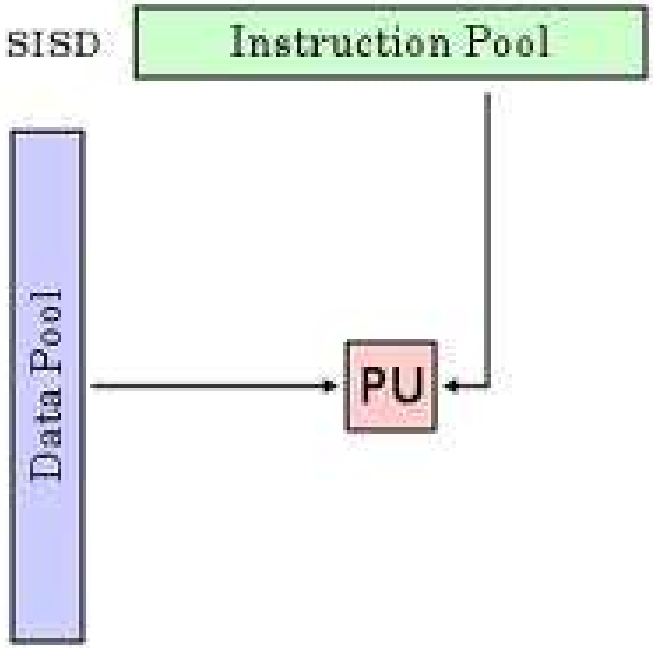


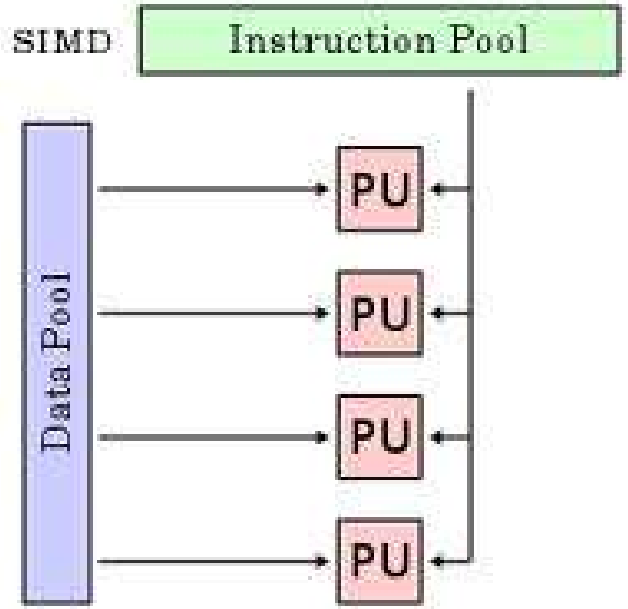
Most of the single core performance loss is on the memory system!

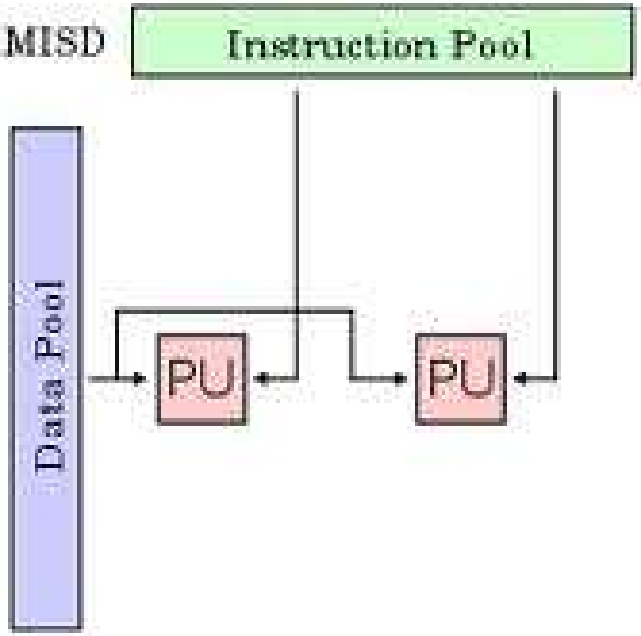
Flynn's Taxonomy

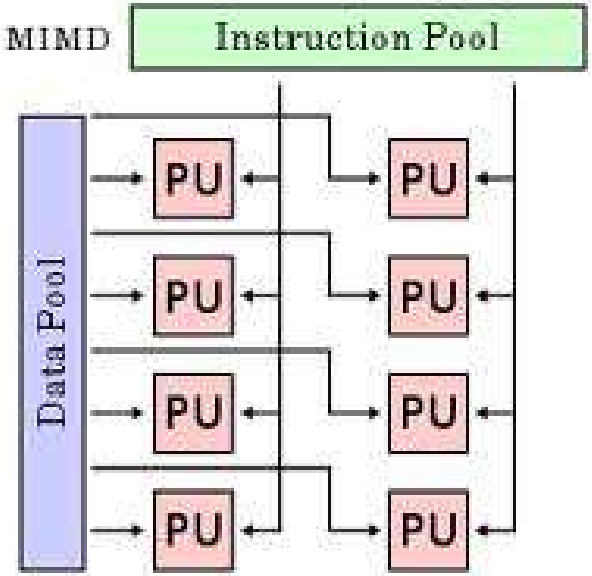
classic von Neumann

SISD Single instruction stream Single data stream	(SIMD) Single instruction stream Multiple data stream
MISD Multiple instruction stream Single data stream	(MIMD) Multiple instruction stream Multiple data stream


SISD
Instruction Pool
Data Pool
PU


SIMD
Instruction Pool
Data Pool
PU
PU
PU
PU

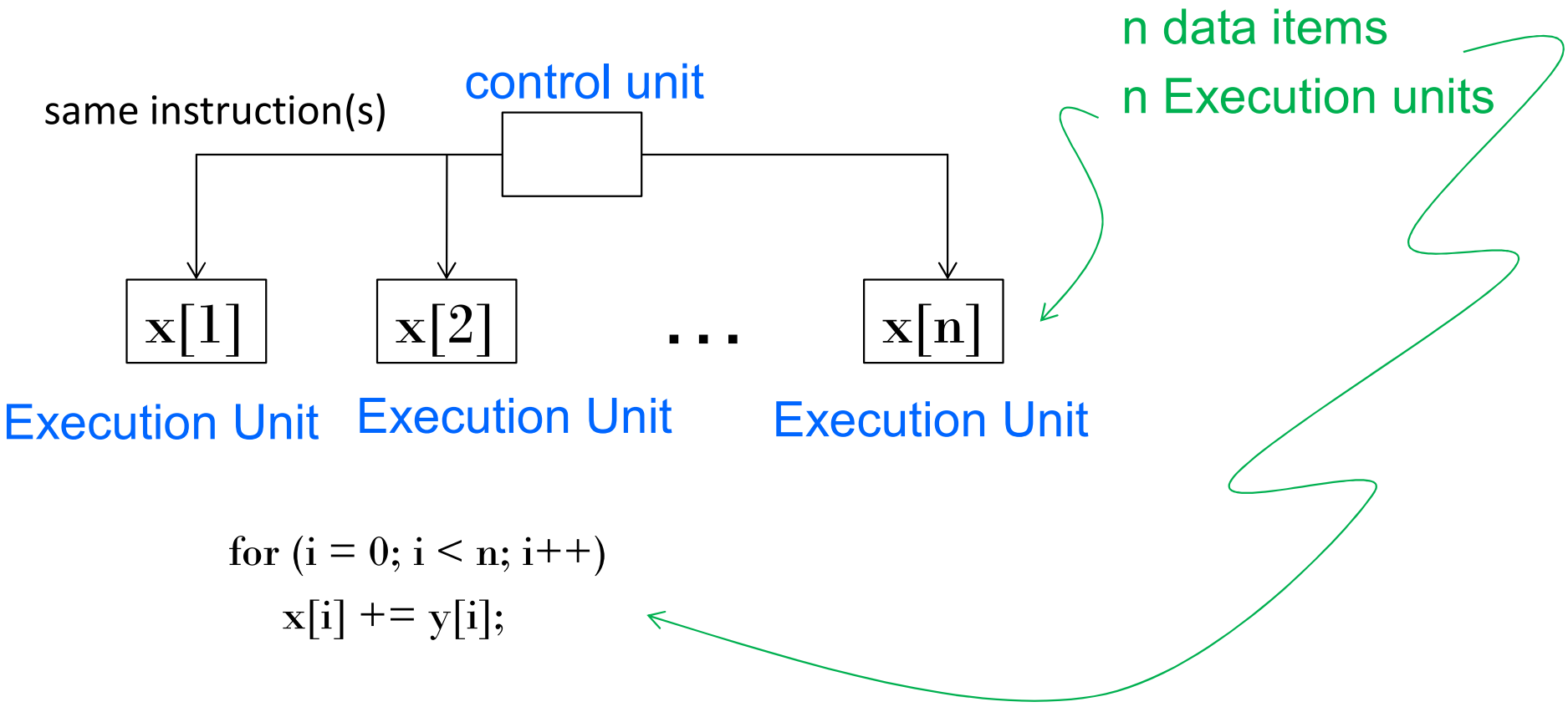

MISD
Instruction Pool
Data Pool
PU
PU


MIMD
Instruction Pool
Data Pool
PU
PU
PU
PU
PU
PU
PU
PU

SIMD

- Parallelism achieved by dividing data among the processors.
- Applies the same instruction (or group of instructions) to multiple data items at once.
- Called data parallelism.
- Example:
  - GPUs
  - vector processors

SIMD example




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

Naming convention: Execution units are most often called Arithmetic and Logic Units (ALUs)

SIMD

- What if we don't have as many ALUs as data items?
- Divide the work and process iteratively.
- Example 4 ALUs and 15 data items.

Round	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]	

ALU = Arithmetic and Logic Unit (i.e. the execution unit that executes the instructions)

SIMD drawbacks

- All ALUs are required to execute the same instruction(s) or remain idle.
- In classic design, they must also operate synchronously (i.e. together at the same time).
- Efficient for large data parallel problems, but not other types of more complex parallel problems.

SIMD Example: Vector processors

- Processors execute instructions where operands are vectors instead of individual data elements or scalars.
- This needs:
  - Vector registers
    - Capable of storing a vector of operands and operating simultaneously on their contents.
  - Vectorized execution units
    - The same operation is applied to each element in the vector (or pairs of elements)

Vector processors - Pros

- Fast
- Easy to use.
- Vectorizing compilers are good at identifying code to exploit.
- Compilers also can provide information about code that cannot be vectorized.
  - Helps the programmer re-evaluate code.
- Makes the best use of memory bandwidth.
  - Bringing a bunch of values is better use for the bandwidth than bringing a value at a time.
- Uses every item in a cache line.

Vector processors - Cons

- They don't handle irregular data structures.

- A very finite limit to their ability to handle ever larger problems. (scalability)
  - The machine has a finite number of vectorized execution units.
  - The machine has a finite number of vectorized registers.

MIMD

- 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.
- Example: multicore processors, multiprocessor systems

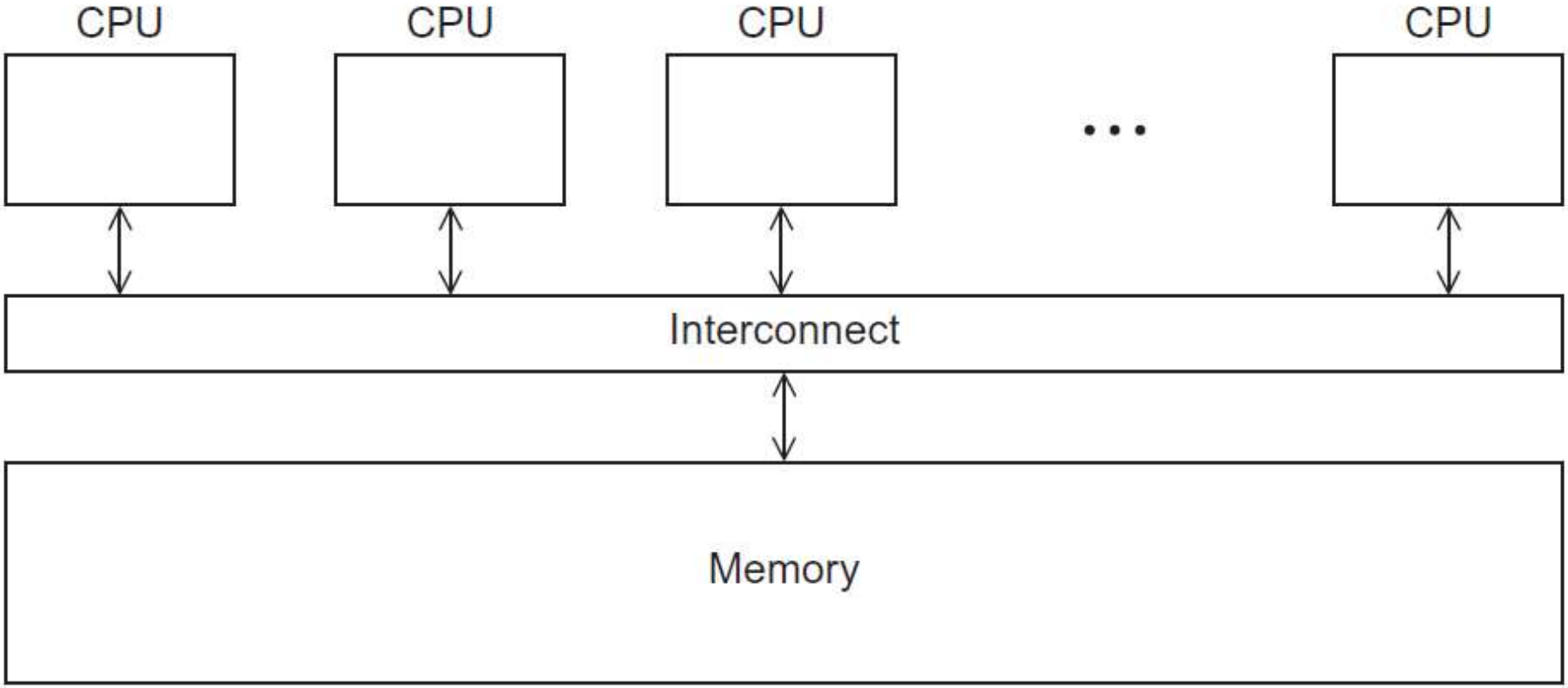
Flynn's classification is based on how instructions and data are used.

Now, let's classify MIMD based on how memory is designed?

Shared Memory System

- A collection of autonomous processors is connected to a memory system via an interconnection network.
- Each processor can access each memory location.
- The processors usually communicate implicitly by accessing data shared in memory.

Shared Memory System

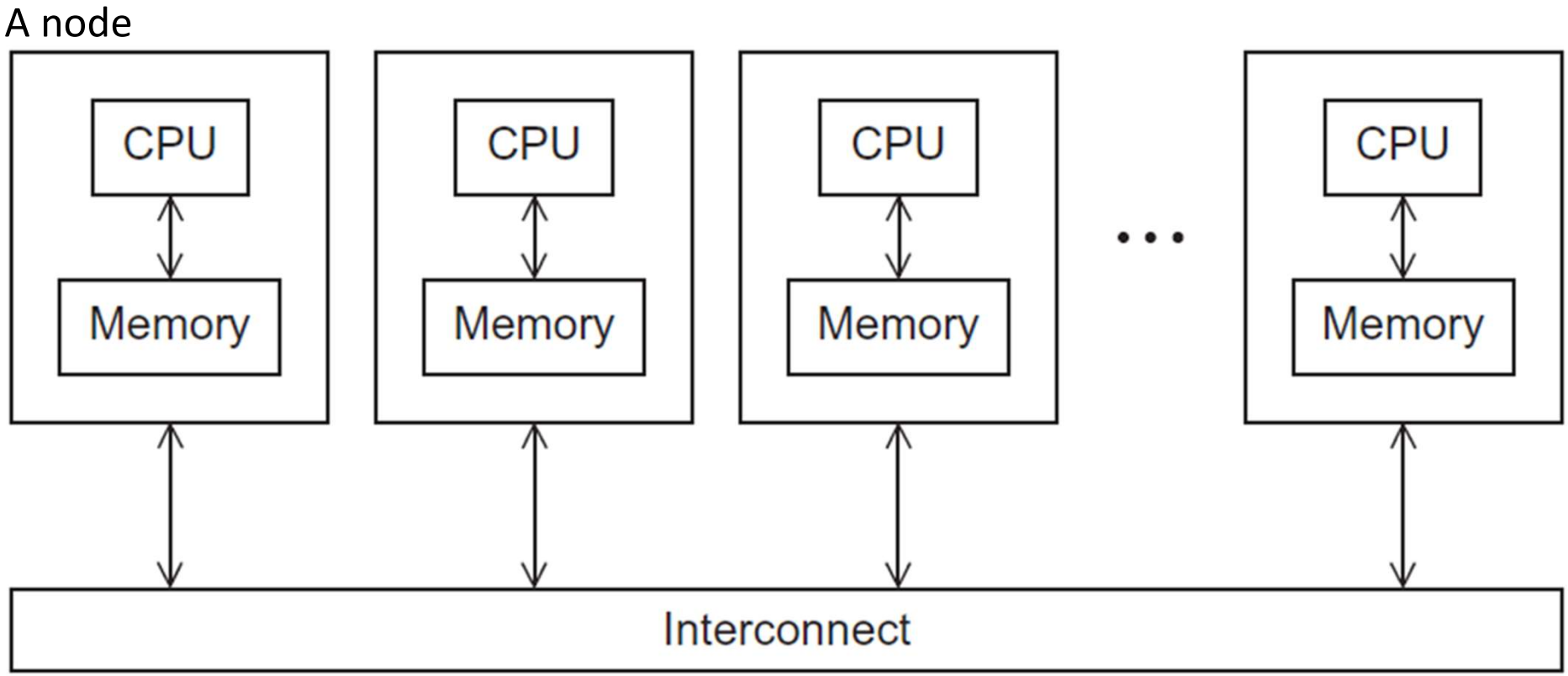


Suppose that one CPU wants to access addr1, and another CPU wants addr2, will they both see the same memory access delay?

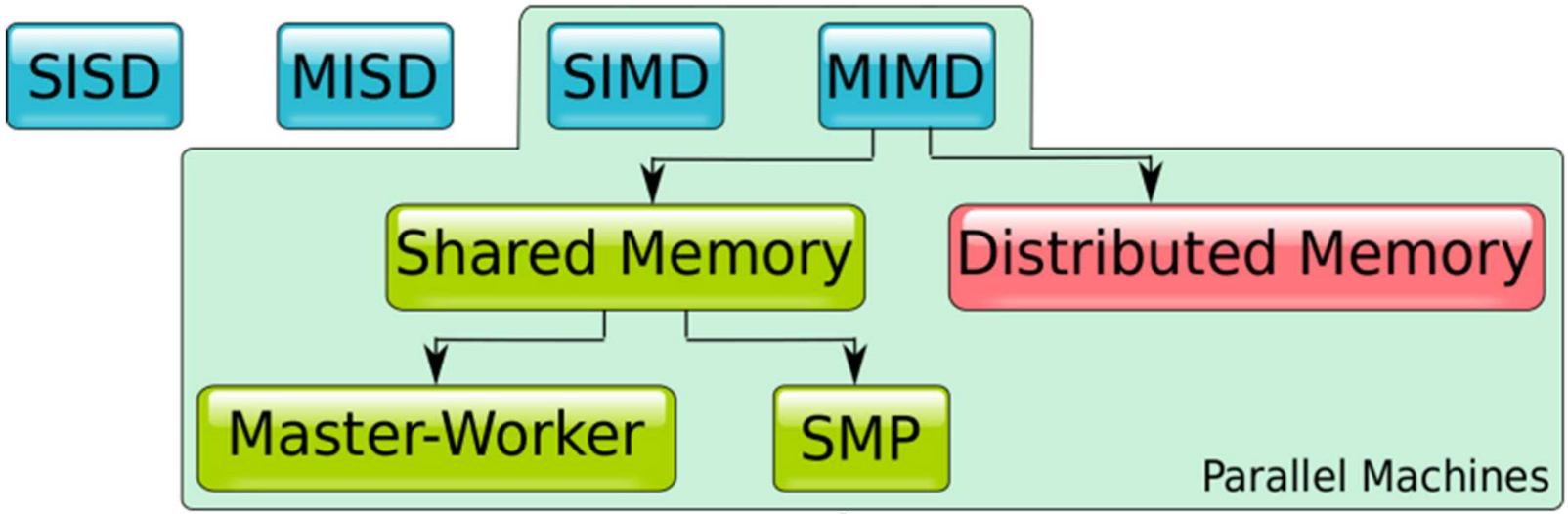
Hint: Banks!

Distributed Memory System

- A collection (cluster) of nodes
  - Connected by an interconnection network



Let's summarize that:



One node is more important than the others.

All nodes are the same.

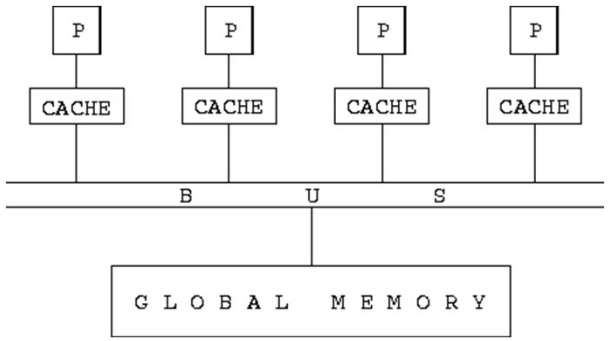
(SMP = Symmetric Multi-Processing)

A Brief discussion of Interconnection networks

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

Shared 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
    - Communication becomes unreliable due to noise.
    - Performance decreases.



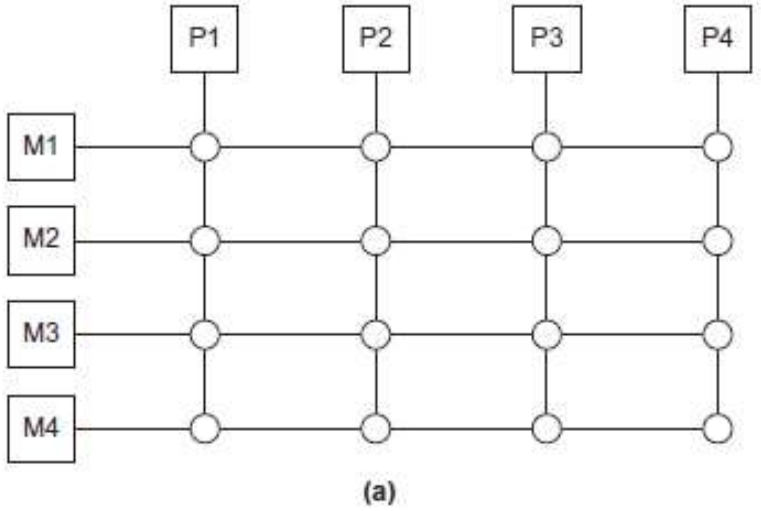
Shared memory interconnects

- Switched interconnect
  - Uses switches to control the routing of data among the connected devices.
  - Those switches are connected by wire forming network of some topology.
  - Example: Crossbar
    - Allows simultaneous communication among different devices.
    - Faster than buses.
    - But the cost of the switches and links is relatively high.

A Crossbar

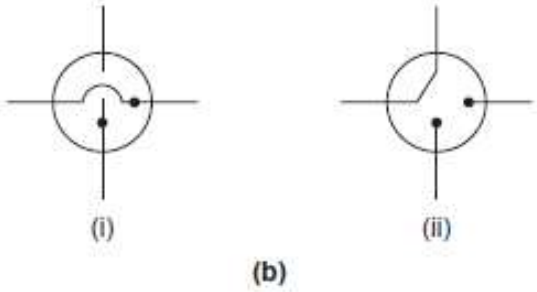
(a)

A crossbar switched network connecting 4 processors (P_i) and 4 memory modules (M_j)

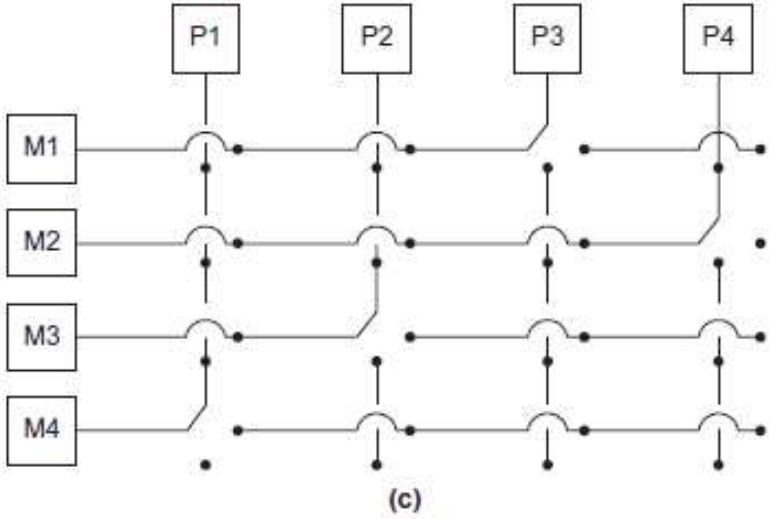


(b)

Configuration of internal switches in a crossbar



(c) Simultaneous memory accesses by the processors

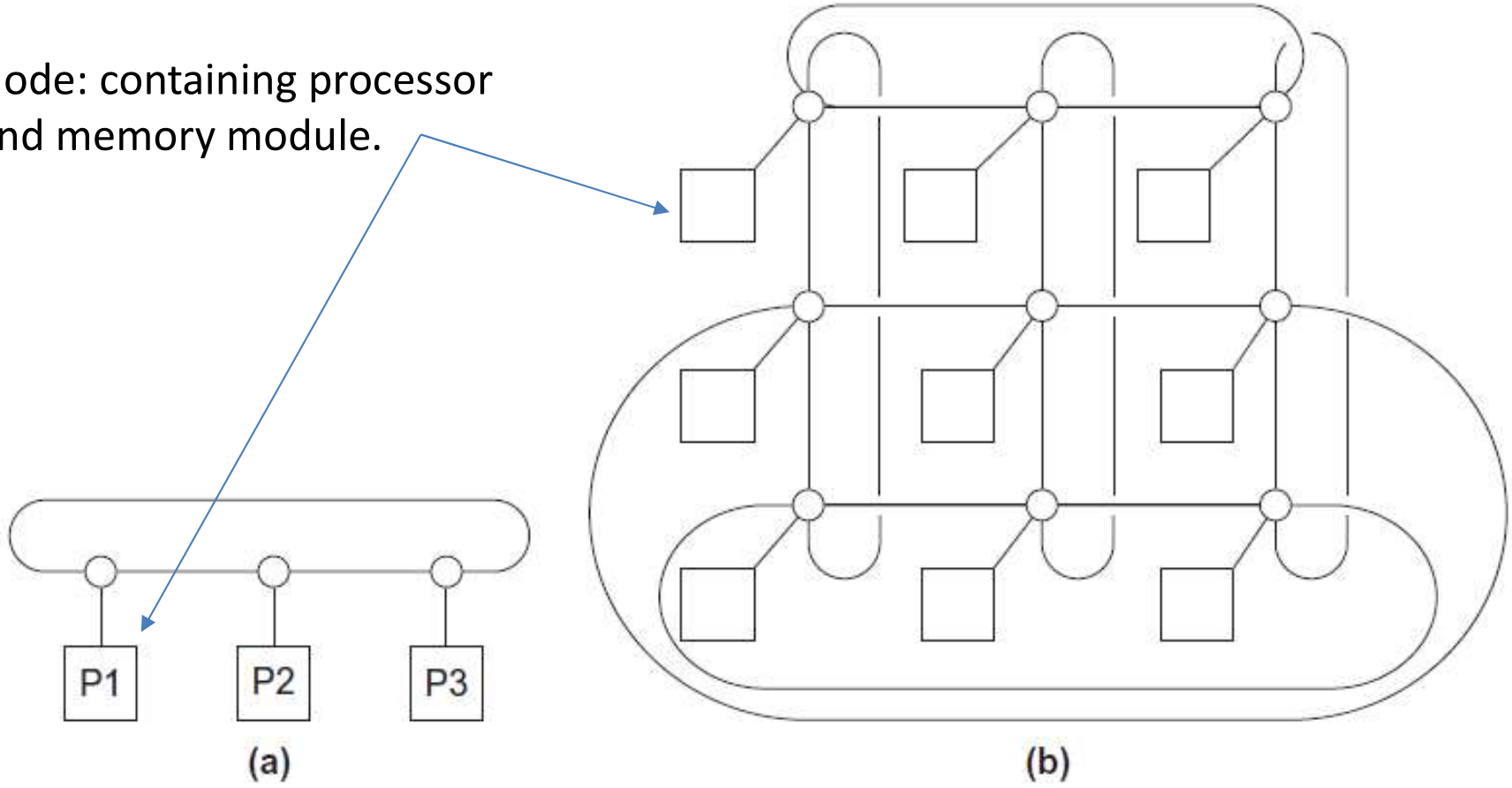


Distributed memory interconnects: Two Types

- 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: Examples

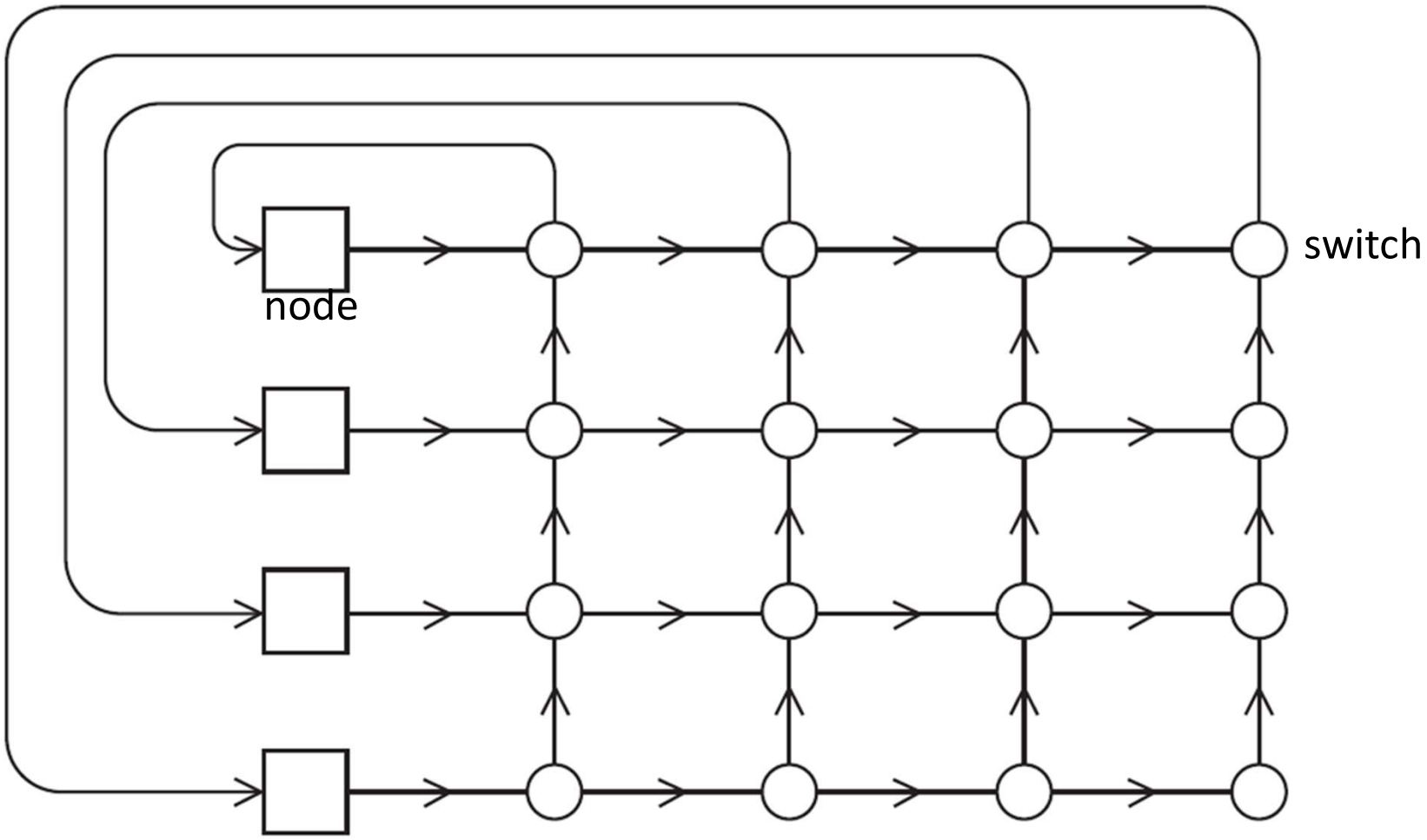
Node: containing processor and memory module.



ring

toroidal mesh

Indirect Interconnect: Examples



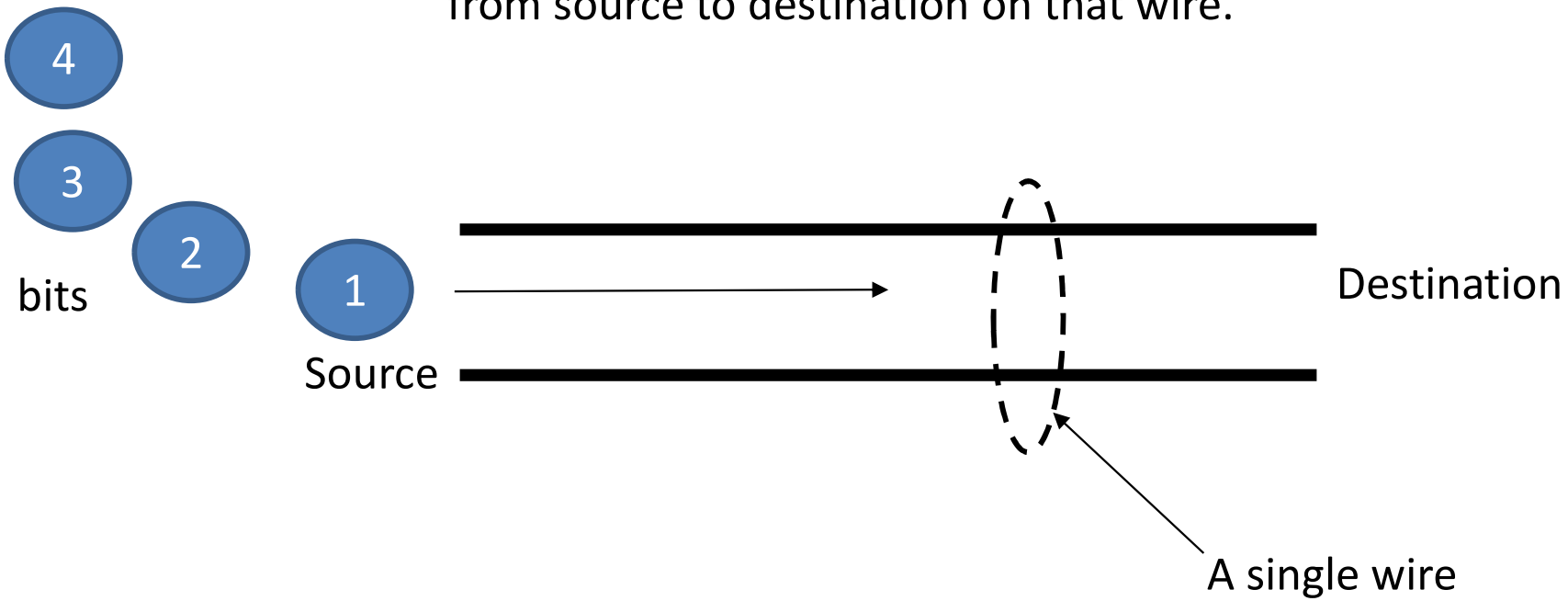
Crossbar Interconnect

Node: containing processor and memory module.

Some Definitions Related to Interconnection Networks

- 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 receiving the first byte.

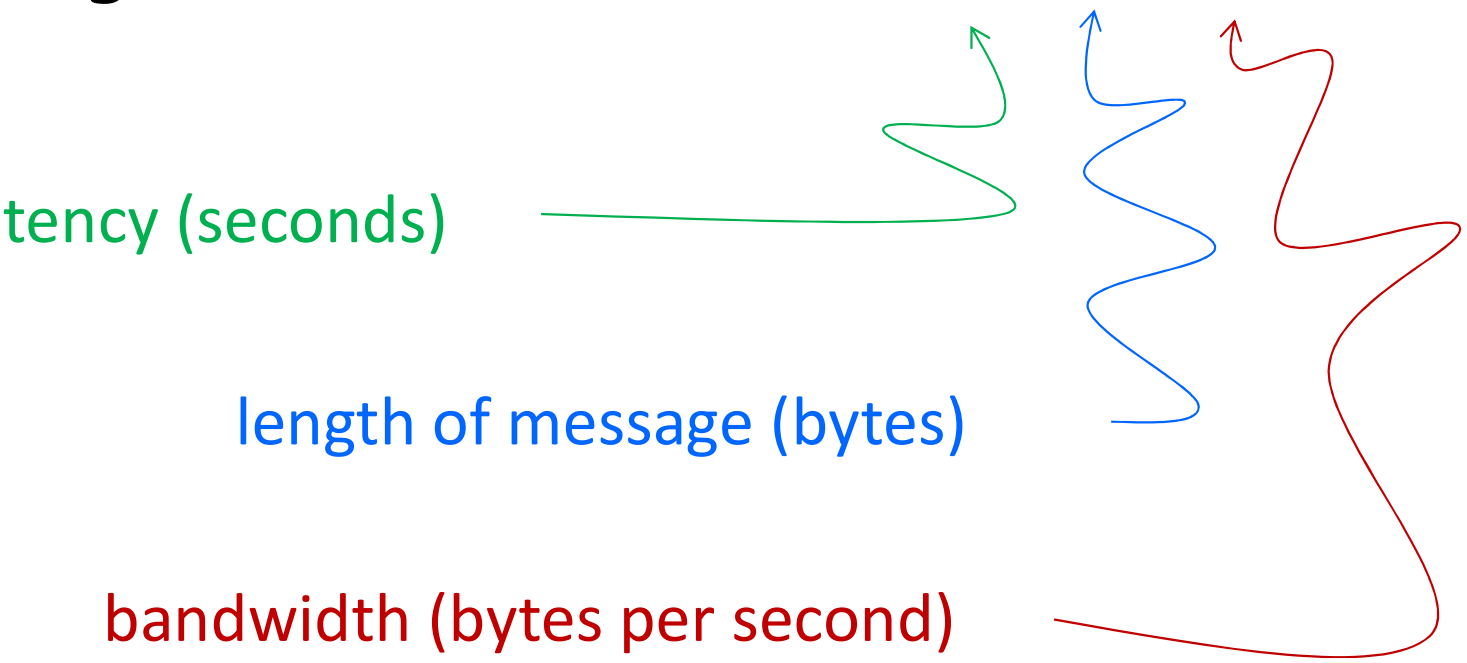
Example: Assume we have one wire, and 4 bits need to be sent from source to destination on that wire.



Latency: Time taken by bit 1 to reach destination starting from the source.

Bandwidth: how many bits are received at the destination at once per cycle. (with one wire, it is one)

Message transmission time = l + n / b

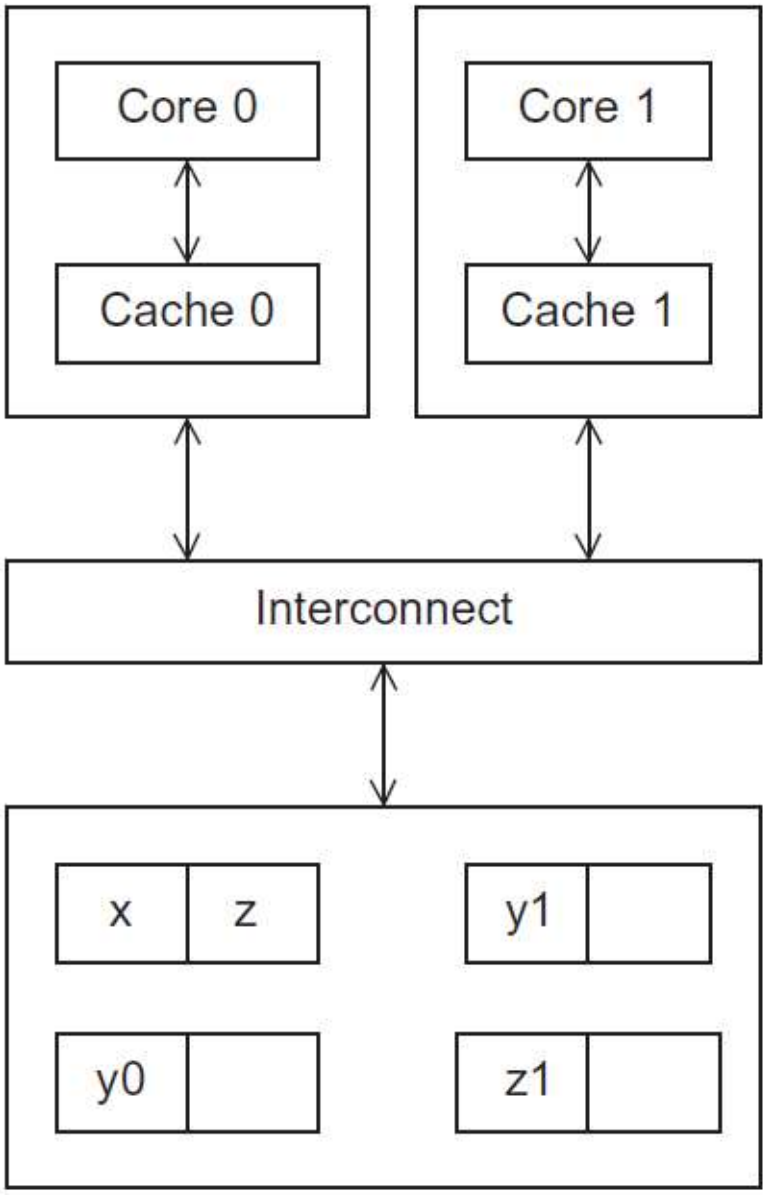


Between the processor/core and the memory modules, there is one or more levels of caches.

This introduces a big challenge.

Cache coherence

- Programmers have no control over caches and when they get updated.
- But programmers can write cache friendly code.



Cache coherence

y0 privately owned by Core 0
y1 and z1 privately owned by Core 1

x = 2; /* shared variable */

Time	Core 0	Core 1
0	`y0 = x;`	`y1 = 3*x;`
1	`x = 7;`	Statement(s) not involving x
2	Statement(s) not involving x	`z1 = 4*x;`

y0 eventually ends up = 2
y1 eventually ends up = 6
z1 = ???

Such a situation is a big mess and must not happen as it leads to buggy code.

This is where cache coherence comes to the rescue.

Snooping Cache Coherence

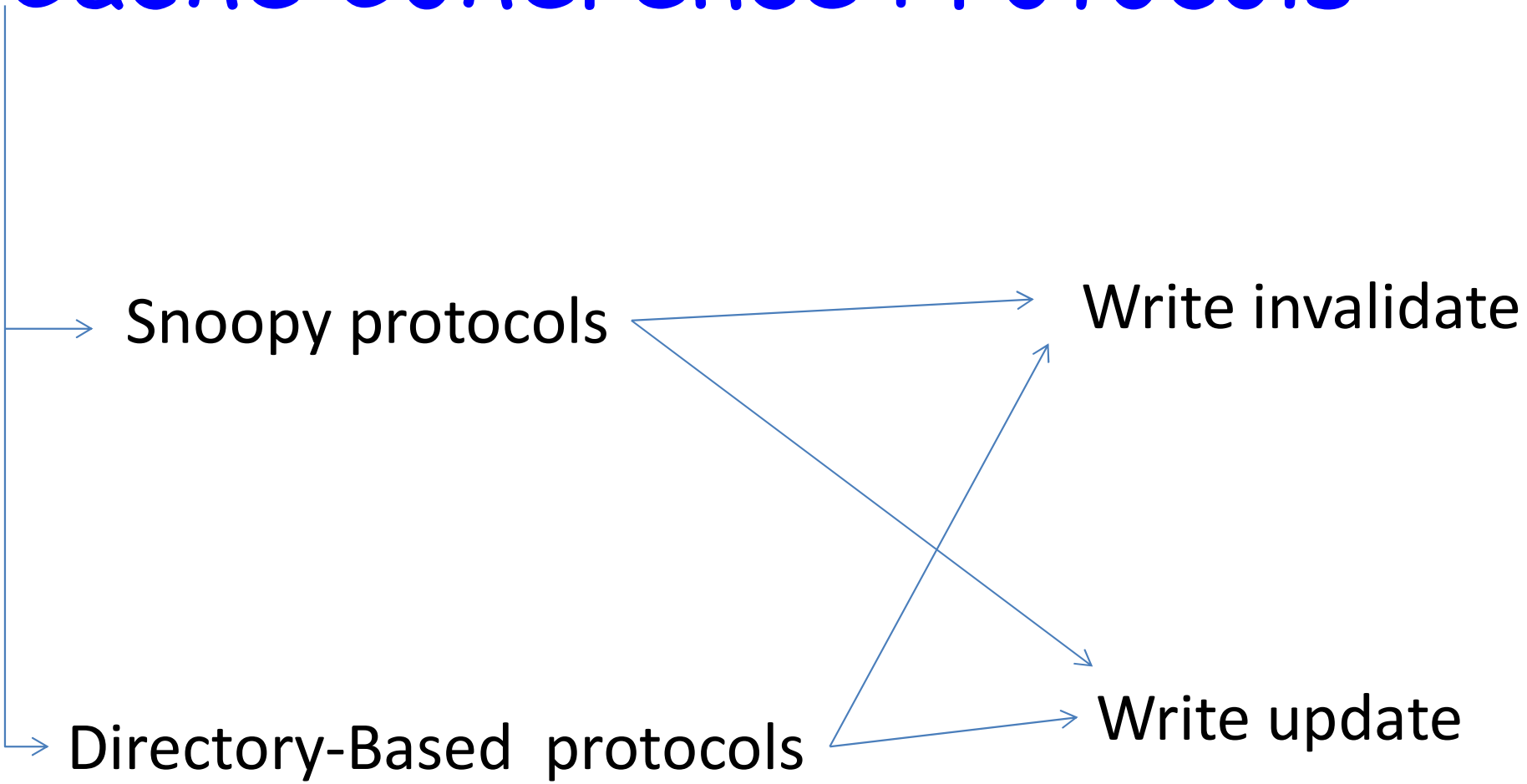
- The cores must share a bus .
- Any signal transmitted on the bus can be "seen" by all cores connected to the bus.
- When core 0, or any other core, updates the copy of x stored in its cache it also broadcasts this information across the bus.
- If another core is "snooping" the bus, it will see that x has been updated and it can mark its copy of x as invalid.

Directory Based Cache Coherence

- Uses a data structure called a directory that stores the status of each cache line.

- When a variable is updated, the directory is consulted, and the cache controllers of the cores that have that variable's cache line in their caches are invalidated.

Cache Coherence Protocols

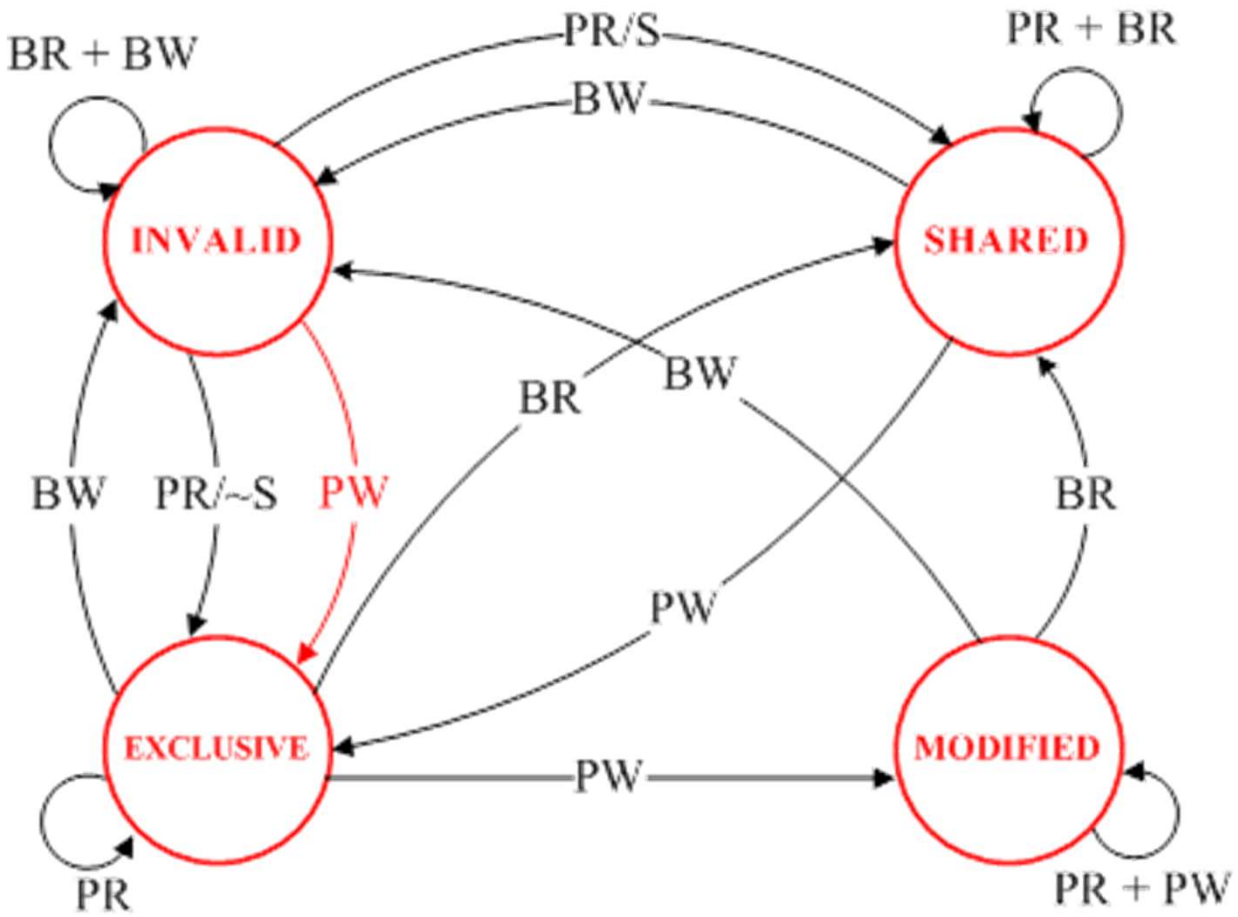


Directory-based is way more scalable than snoopy and hence is more widely used.

There are several coherence protocols

- MESI
- MSI
- MOESI
- …

Example: MESI Protocol

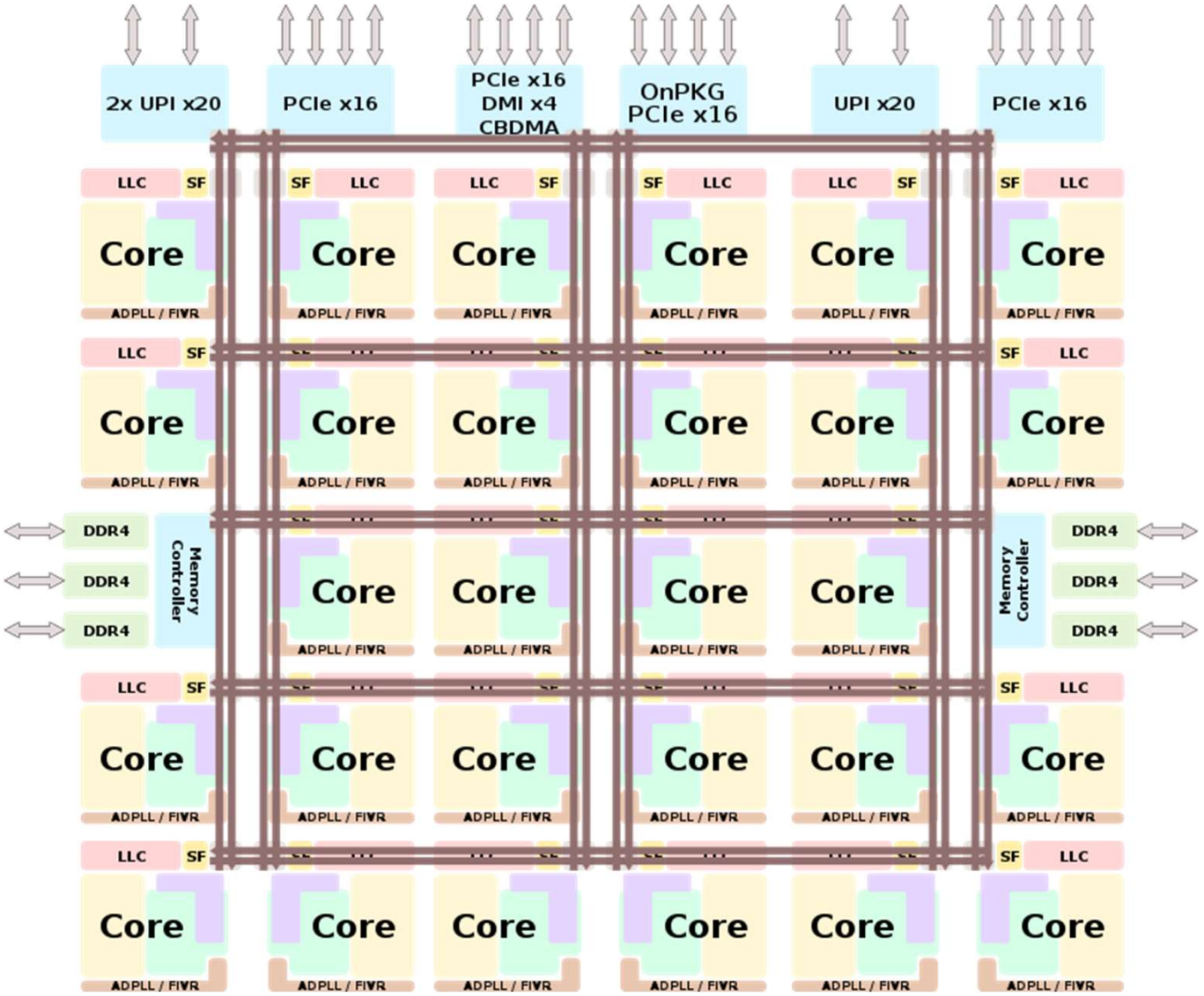


PR = processor read
PW = processor write
S/~S = shared/NOT shared

BR = observed bus read
BW = observed bus write

Examples from Real Life

Multicore Example
Intel Cascade Lake (e.g. Intel Xeon)



14 nm technology

L1D Cache:

- 32 KB/core
- 8-way set associative
- 64 sets, 64 B line size
- Write-back policy

L2 Cache:

- 1 MB/core
- 16-way set associative
- 64 B line size
- Write-back policy
- 14 cycles latency

L3 Cache:

- 1.375 MB/core
- 11-way set associative
- shared across all cores
- 50-70 cycles latency

Supercomputer Example: FUGAKU

#1 spot in Top500 supercomputer
(Nov 2021 list)

7,630,848 cores
(Processor Fujitsu A64FX 2.2GHz)
Total memory: 5,087,232 GB
158, 976 nodes

- 1 node = 1 CPU
- 2 nodes = CPU memory unit (CMU)
- 8 CMUs = Bunch of Blades (BoB)
- 3 BoBs = shelf
- 8 shelves = 1 rack
  - Some racks 4 shelves only.
- The whole machine → 432 racks

RMAX: 442,010 TFlop/s
RPEAK: 537,212 TFlop/s
Power: ~26,248 kW

OS: Red Hat Enterprise Linux

Conclusions

- The trend now is:
  - More cores per chip
  - Non-bus interconnect
  - NUMA and NUCA (Non-Uniform Memory/Cache Access)
- Communication and memory access are the two most expensive operations, NOT computations