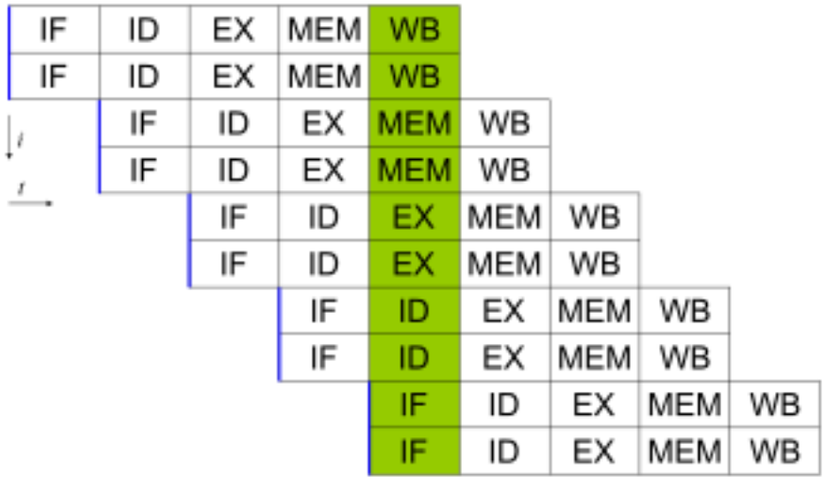
Multiple Issue (1)

Multiple issue processors replicate functional units and try to simultaneously execute different instructions in a program.

```
1 for (i = 0; i < 1000; i++)
2       z[i] = x[i] + y[i];
```

Multiple Issue (2)

static multiple issue - functional units are scheduled at compile time.

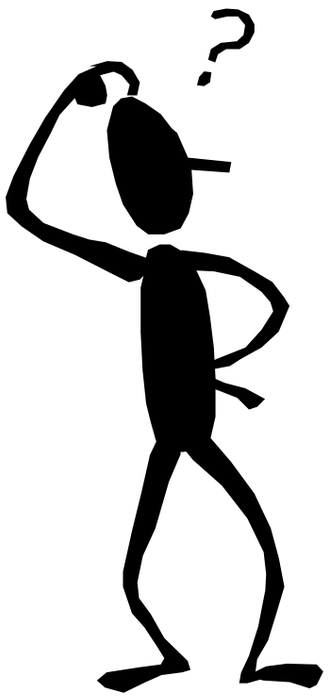


dynamic multiple issue – functional units are scheduled at run-time.

superscalar

Speculation (1)

In order to make use of multiple issue, the system must find instructions that can be executed simultaneously.



In speculation, the compiler or the processor makes a guess about an instruction, and then executes the instruction on the basis of the guess.

Speculation (2)

```
1 z = x + y;
2 if (z > 0)
3         w = x;
4 else
5         w = y;
```

If the system speculates incorrectly,
it must go back and recalculate w = y.

Hardware multithreading (1)

There aren't always good opportunities for simultaneous execution of different instructions.

Ex. Fibonacci number with dynamic programming

Hardware multithreading provides a means for systems to continue doing useful work when the task being currently executed has stalled.

Ex., the current task has to wait for data to be loaded from memory.

Hardware multithreading (2)

Fine-grained - the processor switches between threads after each instruction, skipping threads that are stalled.

Pros: potential to avoid wasted machine time due to stalls.

Cons: a thread that's ready to execute a long sequence of instructions may have to wait to execute every instruction.

Hardware multithreading (3)

Coarse-grained - only switches threads that are stalled waiting for a time-consuming operation to complete.

Pros: switching threads doesn't need to be nearly instantaneous.

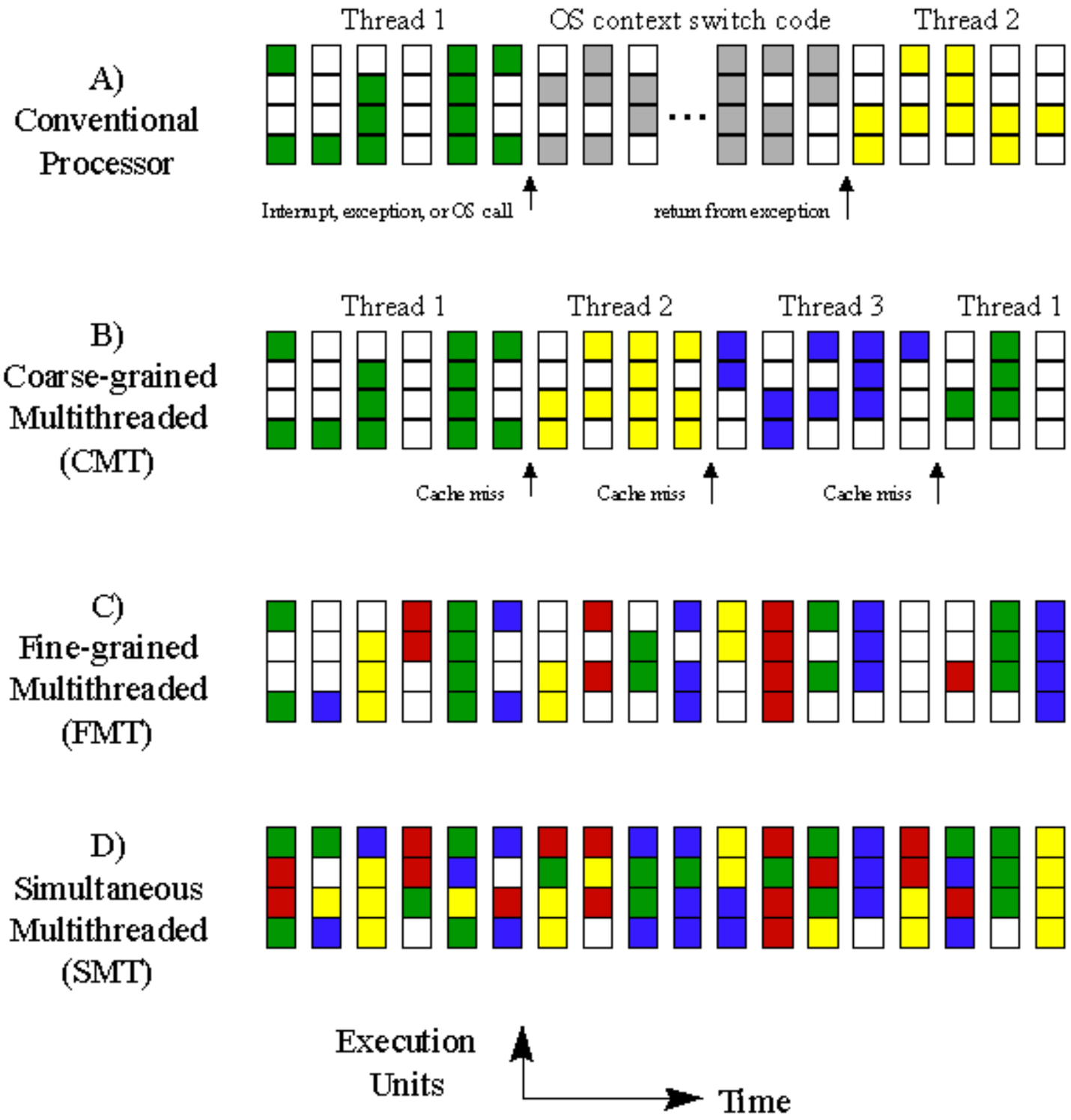
Cons: the processor can be idled on shorter stalls, and thread switching will also cause delays.

Hardware multithreading (4)

Simultaneous multithreading (SMT) - a variation on fine-grained multithreading.

Allows multiple threads to make use of the multiple functional units.

Hardware multithreading (5)



Parallel Hardware

Jun Li, Department of Computer Science, CUNY Queens College

Flyn's Taxonomy

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

classic von Neumann

not covered

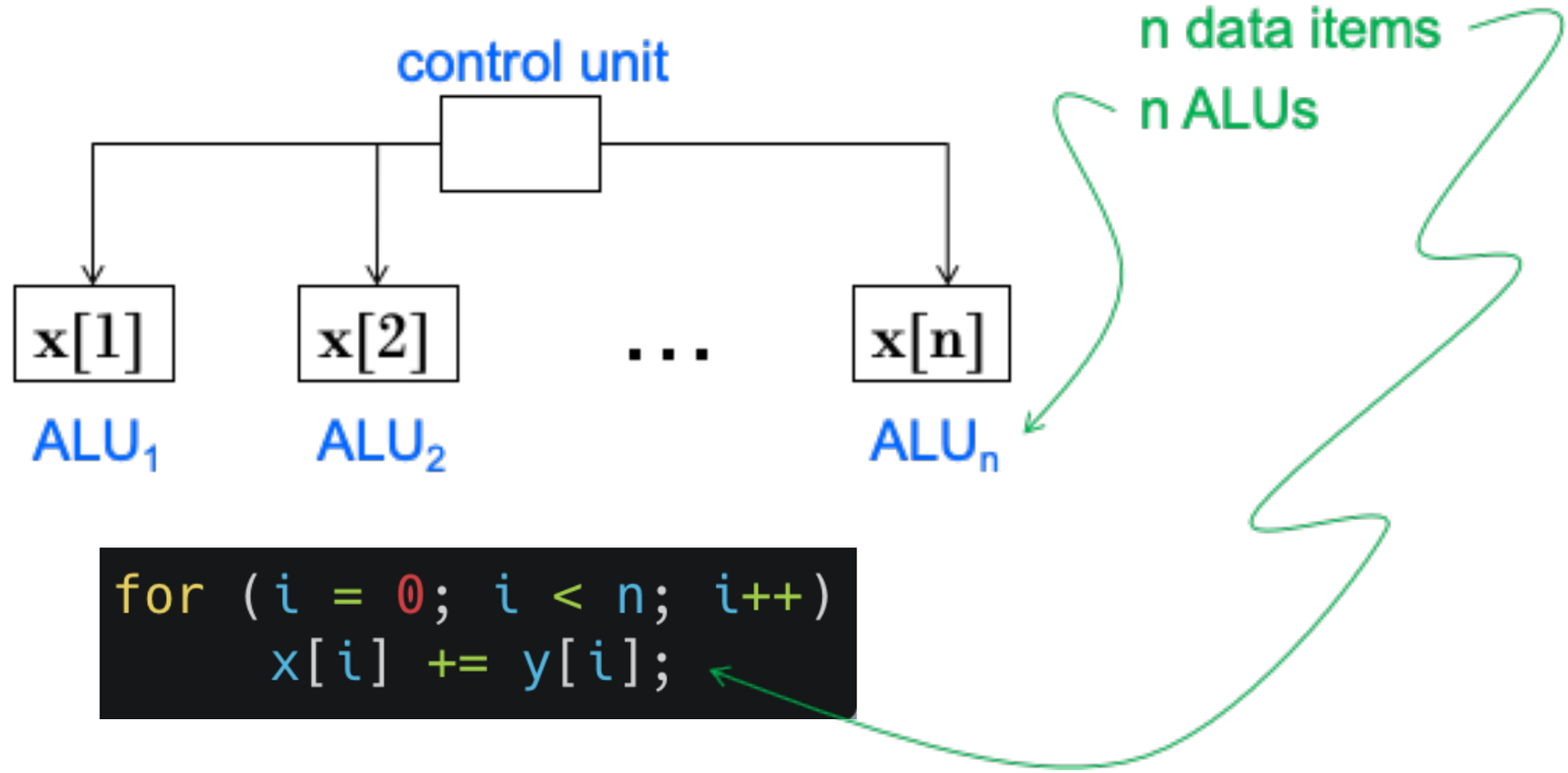
SIMD

Parallelism achieved by dividing data among the processors.

Applies the same instruction to multiple data items.

Called 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 (1)

Operate on arrays or vectors of data while conventional CPU's operate on individual data elements or scalars.

Vector registers.

Capable of storing a vector of operands and operating simultaneously on their contents.

Vector processors (2)

Vectorized and pipelined functional units.

The same operation is applied to each element in the vector (or pairs of elements).

Vector instructions.

Operate on vectors rather than scalars.

Vector processors (3)

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.

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.

High memory bandwidth.

Uses every item in a cache line.

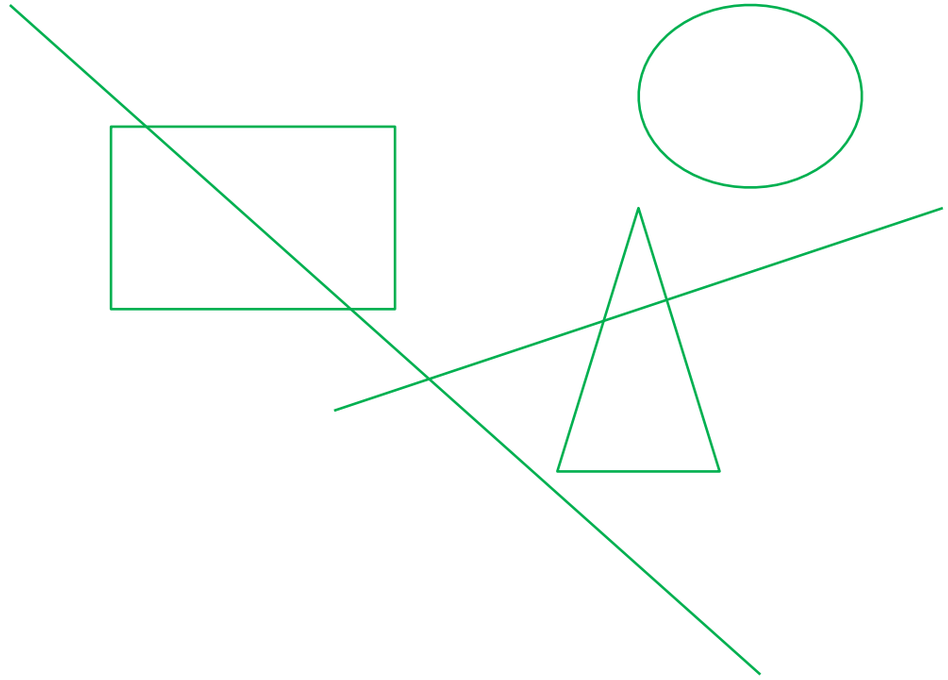
Vector processors - Cons

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)

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.

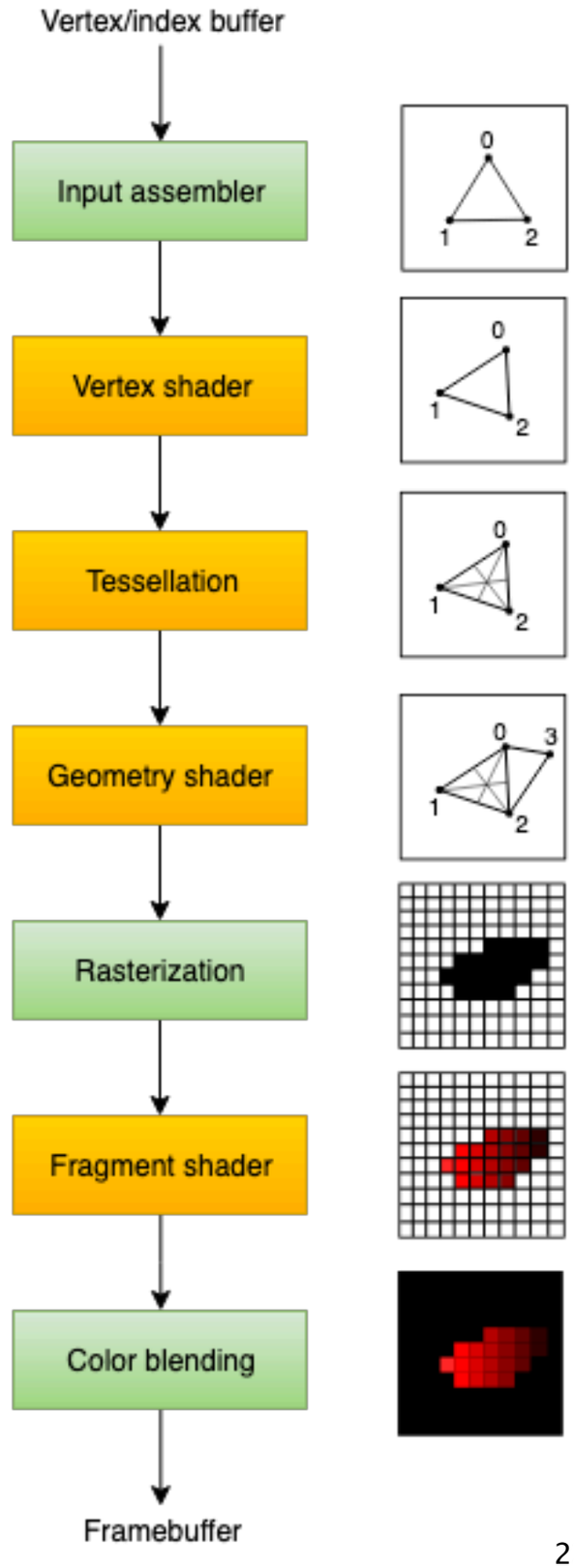


GPUs

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.

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.

Shared Memory System (1)

A collection of autonomous processors is connected to a memory system via an interconnection network.

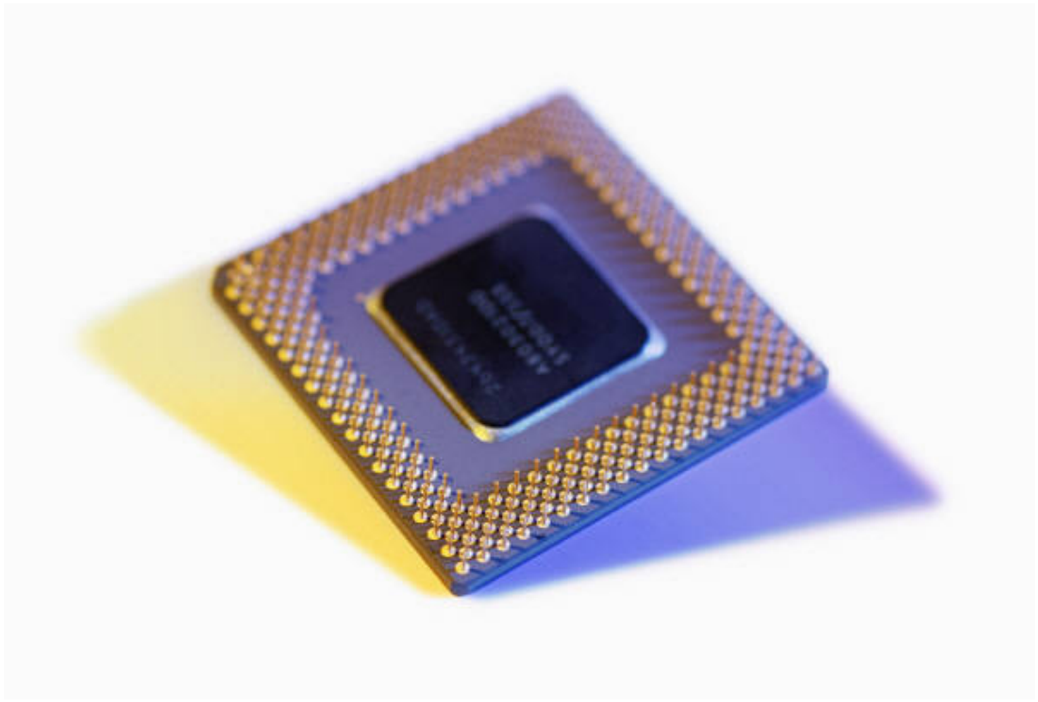
Each processor can access each memory location.

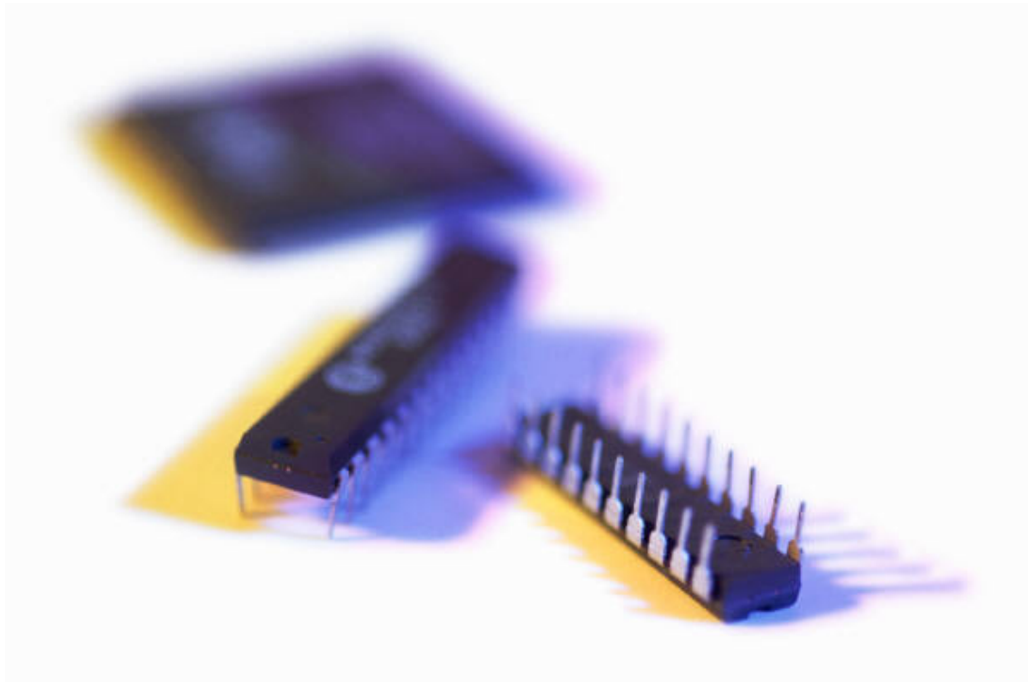
The processors usually communicate implicitly by accessing shared data structures.

Shared Memory System (2)

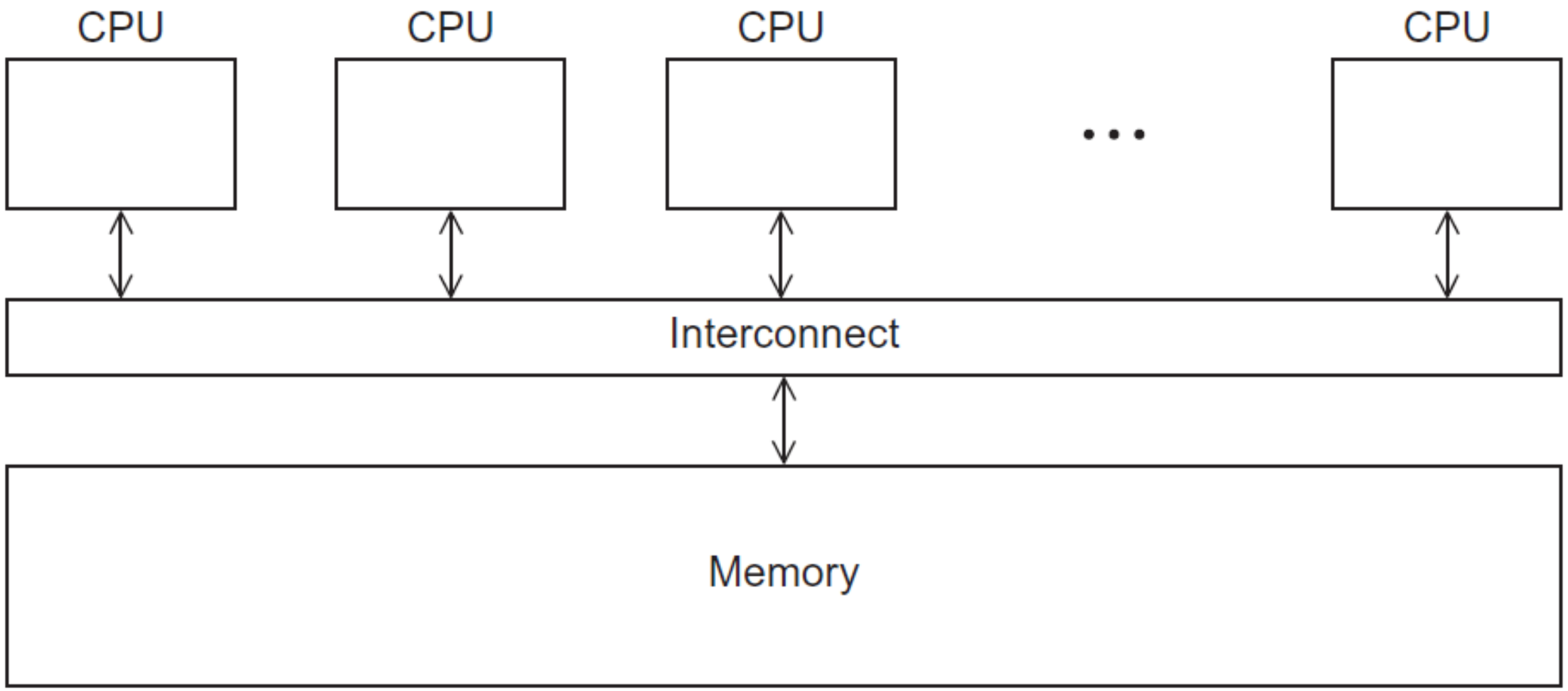
Most widely available shared memory systems use one or more multicore processors.

(multiple CPU's or cores on a single chip)

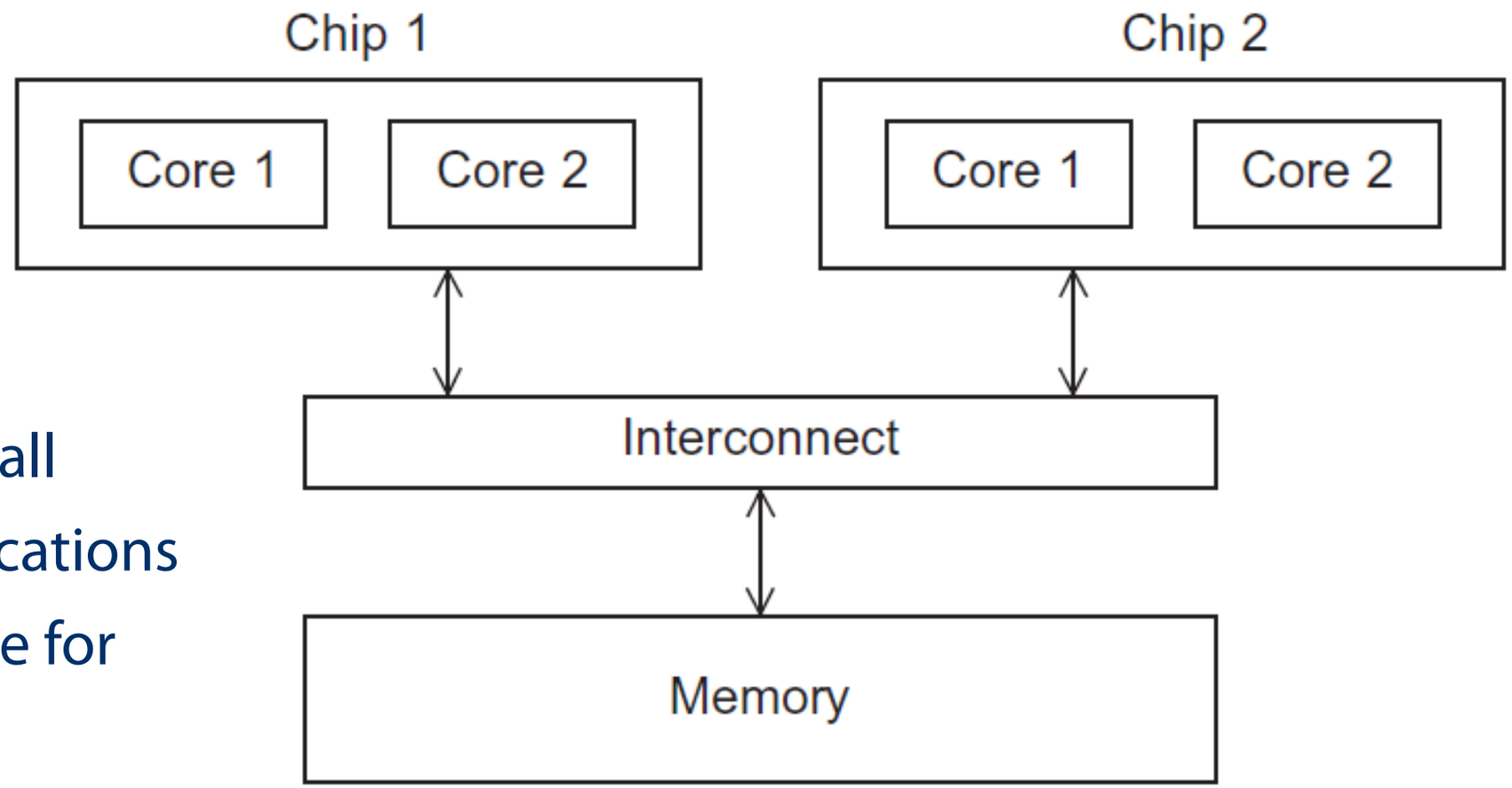




Shared Memory System

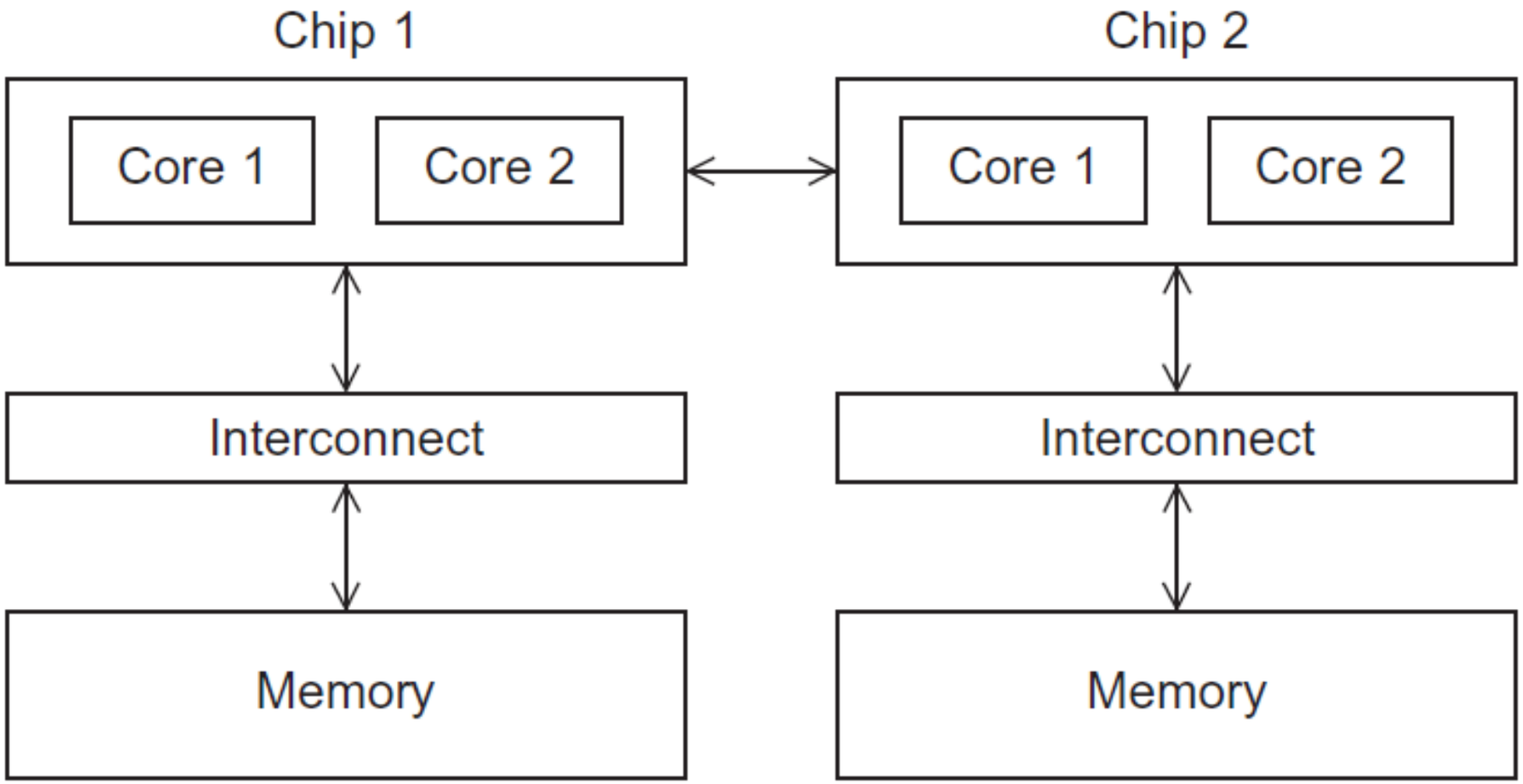


UMA multicore system



Time to access all the memory locations will be the same for all the cores.

NUMA multicore system



A memory location a core is directly connected to can be accessed faster than a memory location that must be accessed through another chip.

Distributed Memory System

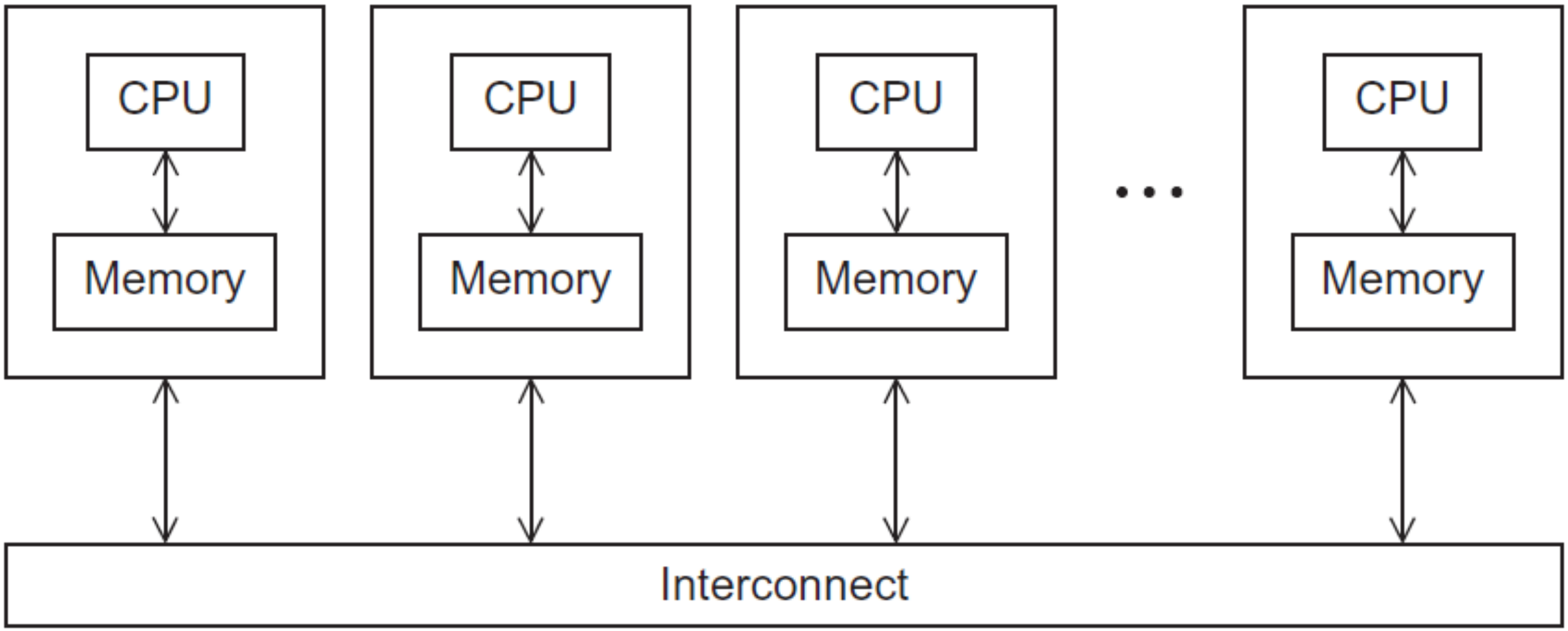
Clusters (most popular)

A collection of commodity systems.

Connected by a commodity interconnection network.

Nodes of a cluster are individual computations units joined by a communication network.

Distributed Memory System



Interconnection networks

Affects performance of both distributed and shared memory systems.

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, and performance decreases.

Shared memory interconnects

Switched interconnect

Uses switches to control the routing of data among the connected devices.

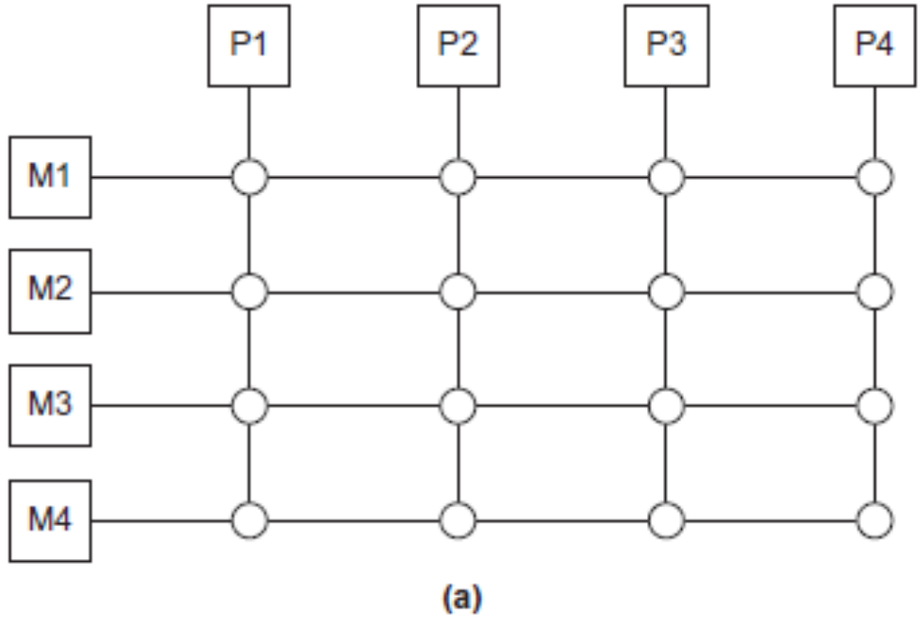
Crossbar

Allows simultaneous communication among different devices.

Faster than buses.

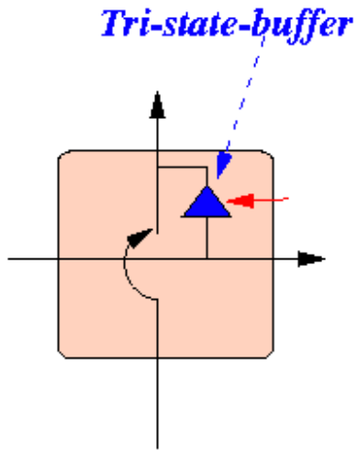
But the cost of the switches and links is relatively high.

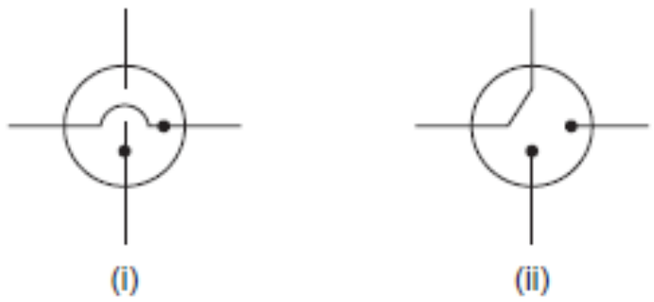
(a) crossbar switch connecting 4 processors (P_i) and 4 memory modules (M_j)



(a)

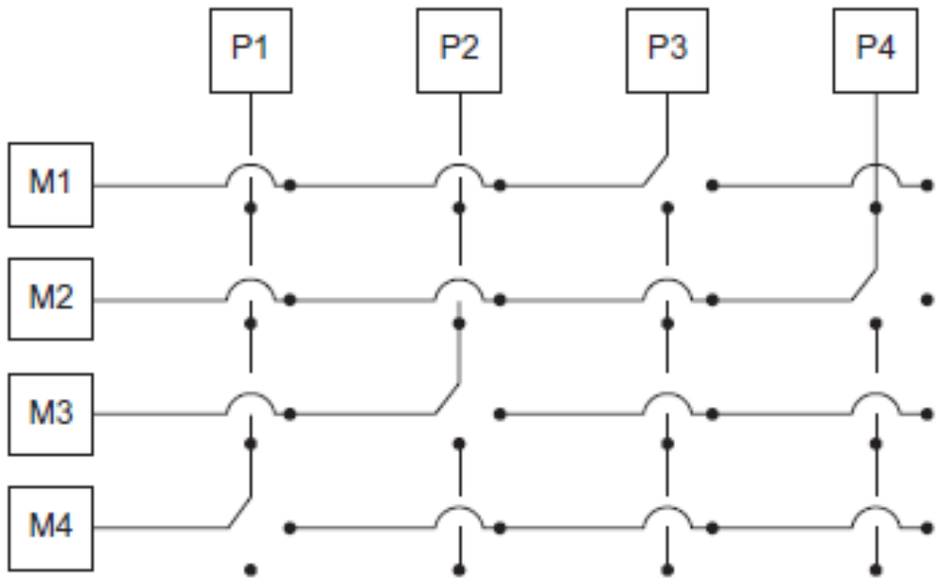
(b) Configuration of internal switches in a crossbar





(b)

(c) Simultaneous memory accesses by the processors



(c)