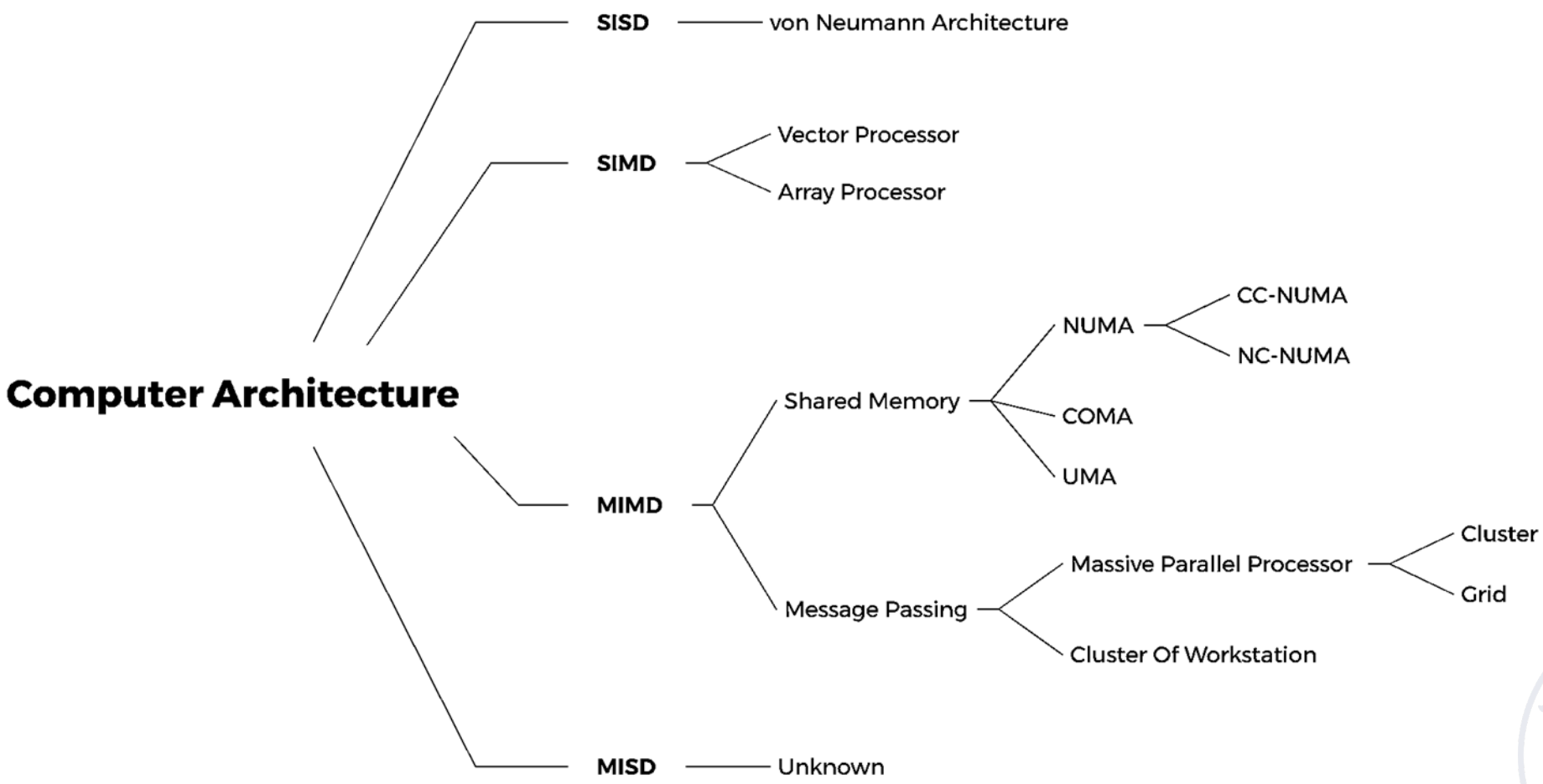
Chapter 5

DLP and TLP

Flynn


Computer Architecture
SISD
von Neumann Architecture
SIMD
Vector Processor
Array Processor
MIMD
Shared Memory
NUMA
CC-NUMA
NC-NUMA
COMA
UMA
Message Passing
Massive Parallel Processor
Cluster
Grid
Cluster Of Workstation
MISD
Unknown

From ILP to TLP

- Thread-level parallelism is identified at a high level by software system or programmer;
- The threads consist of hundreds to millions of instructions that may be executed in parallel.

From TLP to MIMD Architecture

- TLP implies the existence of multiple program counters.
- Thus TLP is exploited primarily through MIMDs

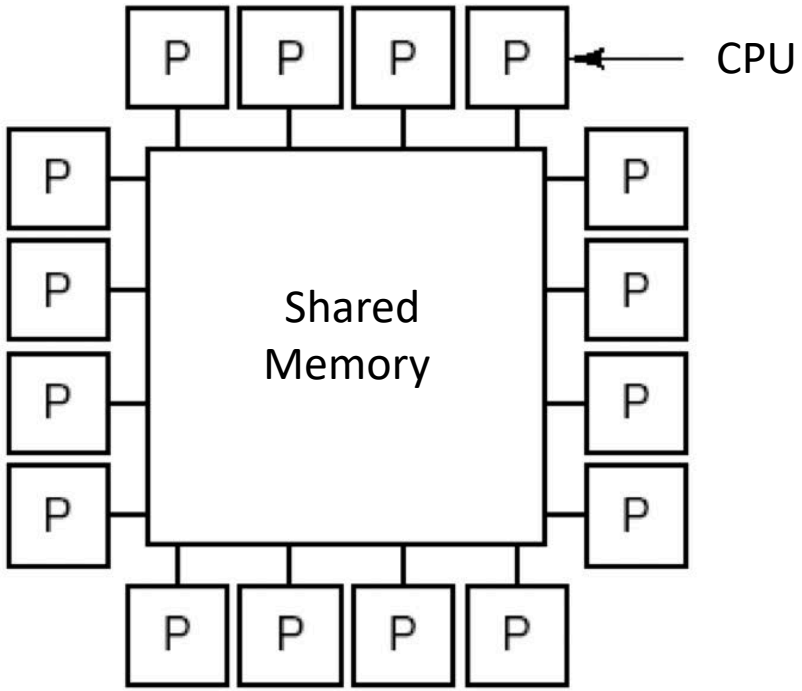
MIMD Architecture

- Multi-processor system——based on shared memory
  - There is only a unique address space in the system, and all processors share this address space.
  - A unique address space does not mean that there is only one memory physically. The shared address space can be realized by a physically shared memory, or can be realized by a distributed memory with the support of hardware and software.
- Multi-computer system——based on message passing
  - Each processor has its own memory, which can only be accessed by the processor and cannot be directly accessed by other processors. This type of memory is called local memory or private memory.
  - When processor A needs to send data to processor B, A sends the data to B in the form of a message.

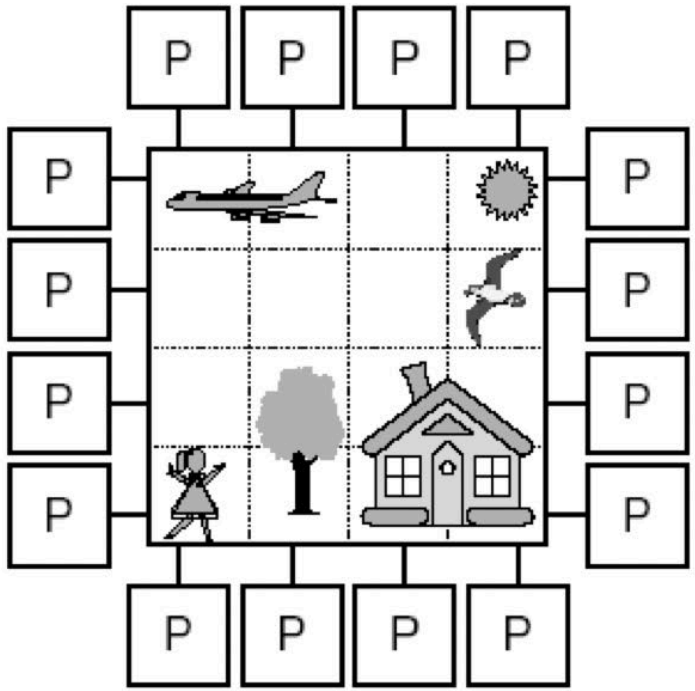
MIMD Architecture

- Multi-processor system——based on shared memory
  - There is only a unique address space in the system, and all processors share this address space.
  - A unique address space does not mean that there is only one memory physically. The shared address space can be realized by a physically shared memory, or can be realized by a distributed memory with the support of hardware and software.
- Multi-computer system——based on message passing
  - Each processor has its own memory, which can only be accessed by the processor and cannot be directly accessed by other processors. This type of memory is called local memory or private memory.
  - When processor A needs to send data to processor B, A sends the data to B in the form of a message.

Shared Memory System



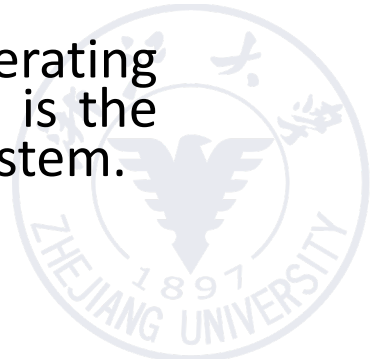
(a) A shared memory system with 16 CPUs



(b) A pictures is divided into 16 parts which are processed by different CPUs

Shared Memory System

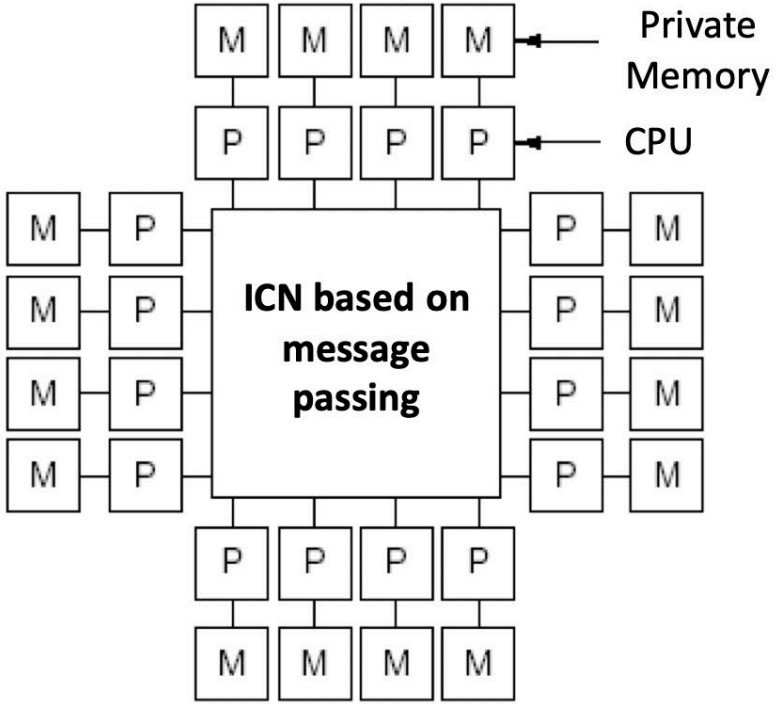
- There are multiple CPUs and all CPUs share the same virtual address space mapped to the shared physical memory. Multiprocessor systems are sometimes referred to as **Shared Memory System**.
- From a software perspective, multiprocessor systems are easy to expand. Any processor can access memory by executing LOAD/STORE instructions. Two processors can communicate in a very simple way, as long as one processor writes data into the memory and the other processor reads the data from the memory.
- Multiprocessor systems also have disks, network adapters, and other input/output devices. If in a system, each CPU has equal access to all memory modules and input/output devices, and these CPUs are interchangeable in the operating system, then the system is a symmetric multiprocessor system **SMP** (Symmetric Multi-processor).
- There is only one operating system in a multiprocessor system, and the operating system is responsible for managing a series of tables. This single system image is the main feature that distinguishes a multiprocessor system from a multicomputer system.



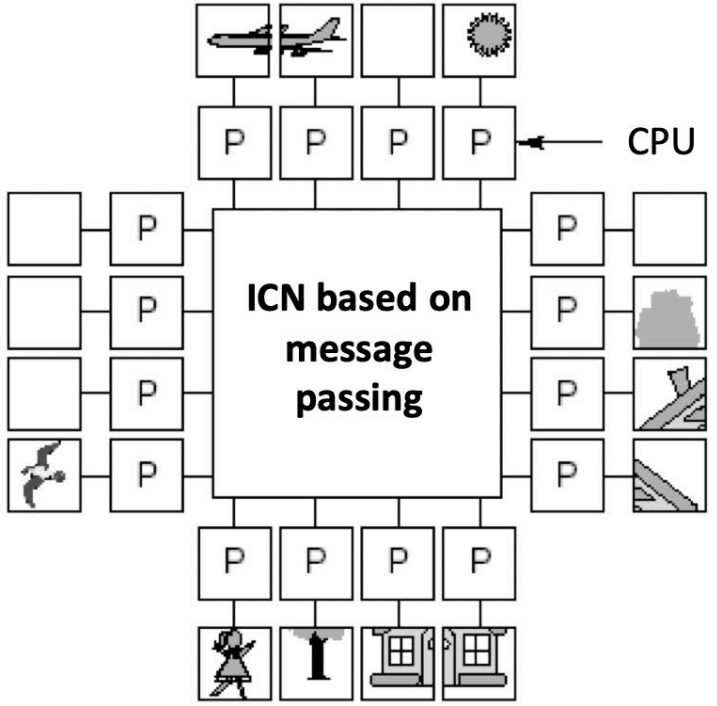
MIMD Architecture

- Multi-processor system——based on shared memory
  - There is only a unique address space in the system, and all processors share this address space.
  - A unique address space does not mean that there is only one memory physically. The shared address space can be realized by a physically shared memory, or can be realized by a distributed memory with the support of hardware and software.
- Multi-computer system——based on message passing
  - Each processor has its own memory, which can only be accessed by the processor and cannot be directly accessed by other processors. This type of memory is called local memory or private memory.
  - When processor A needs to send data to processor B, A sends the data to B in the form of a message.

Multi-computer system based on message passing



(a) Every CPU owns its private memory



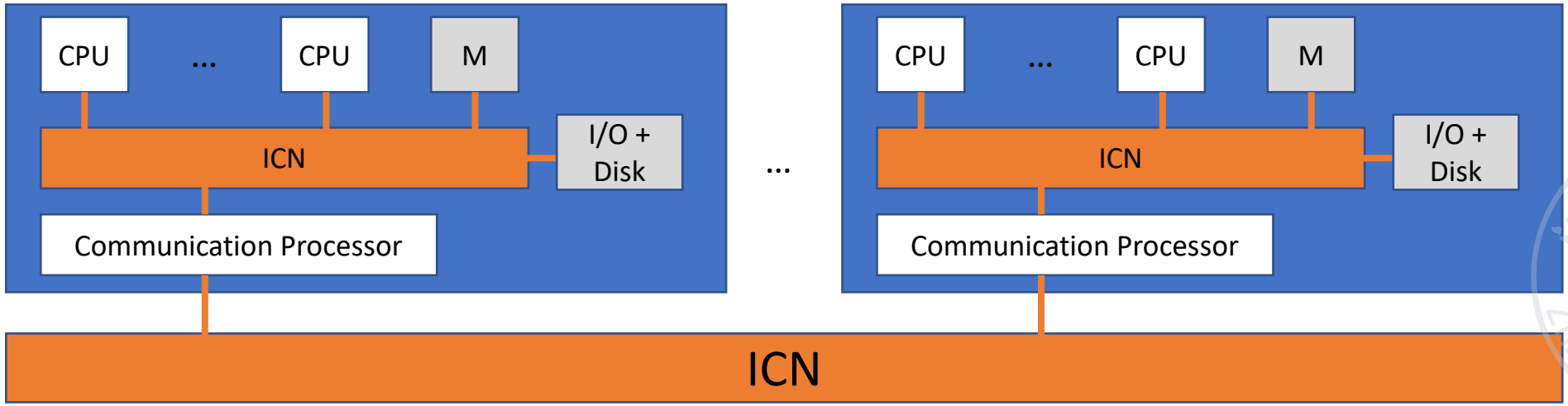
(b) A pictures is divided into 16 parts which are processed and stored separately

Multi-computer system based on message passing

- In a multi-computer architecture, each CPU has its own private memory, which can only be used by itself and other CPUs cannot access it. Each CPU has its own independent physical address space.
- The multi-computer system has good scalability and can reach a larger scale than the multi-processor system.
- The feature of shared memory without hardware implementation in multi-computer systems also greatly affects its software architecture. The CPU in a multi-computer system cannot communicate by reading and writing shared memory. Communication in the system is achieved by using an interconnection network to pass messages. The programming of a multi-computer system is much more complicated than that of a multi-processor system.
- Each node in a multi-computer system consists of one or more CPUs, RAMs, magnetic disks, and other input/output devices and communication processors. And each node has an operating system, at least the core part of the operating system.

Multi-computer system based on message passing

- General multi-computer architecture
  - Each node is composed of one or more CPUs, RAMs, disks and other input/output devices and communication processors.
  - The communication processors are connected to each other through an interconnection network. A variety of different topologies, switching strategies and path finding algorithms can be used.



MIMD Architecture

- Different memory access models of MIMD multiprocessor system
  - Uniform Memory Access (UMA)
  - Non Uniform Memory Access (NUMA)
  - Cache Only Memory Access (COMA)

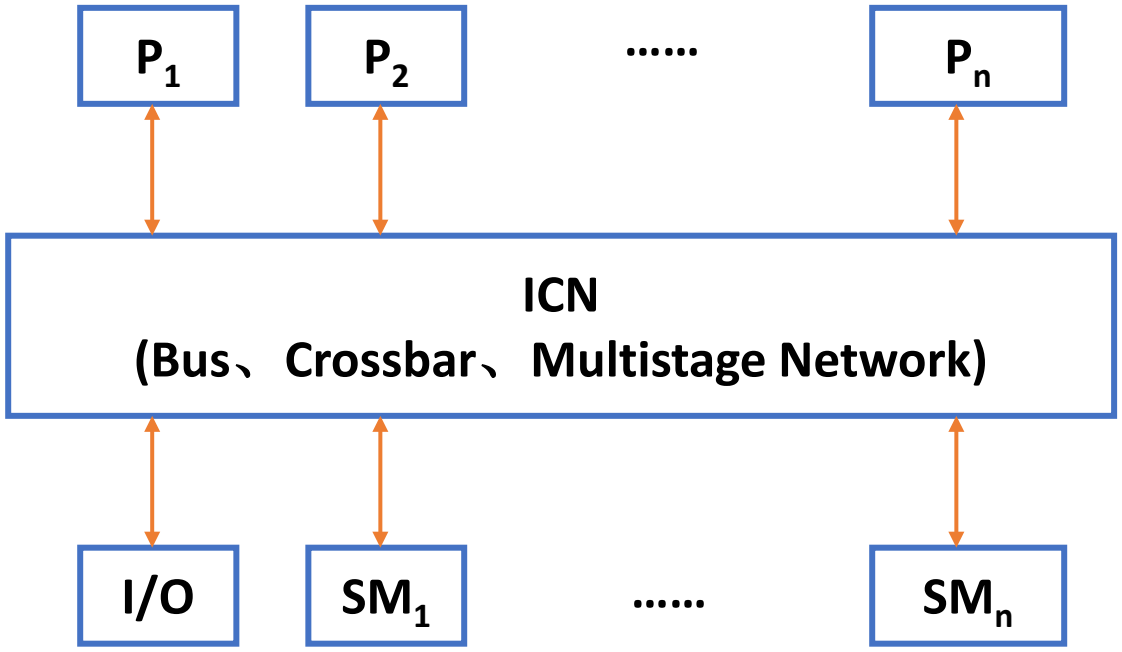
- Further division of MIMD multi-computer system
  - Massively Parallel Processors (MPP)
  - Cluster of Workstations(COW)

MIMD Architecture

- Different memory access models of MIMD multiprocessor system
  - Uniform Memory Access (UMA)
  - Non Uniform Memory Access (NUMA)
  - Cache Only Memory Access (COMA)

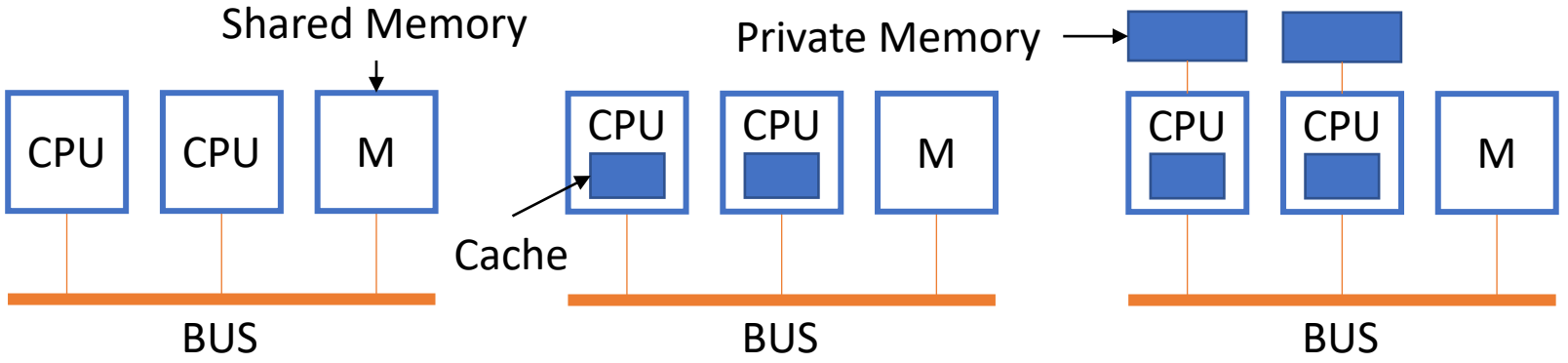
- Further division of MIMD multi-computer system
  - Massively Parallel Processors (MPP)
  - Cluster of Workstations(COW)

UMA



UMA Multi-processor system

- UMA system features
  - Physical memory is uniformly shared by all processors
  - It takes the same time for all processors to access any memory word
  - Each processor can be equipped with private cache or private memory
- UMA Multiprocessor System Based on Bus

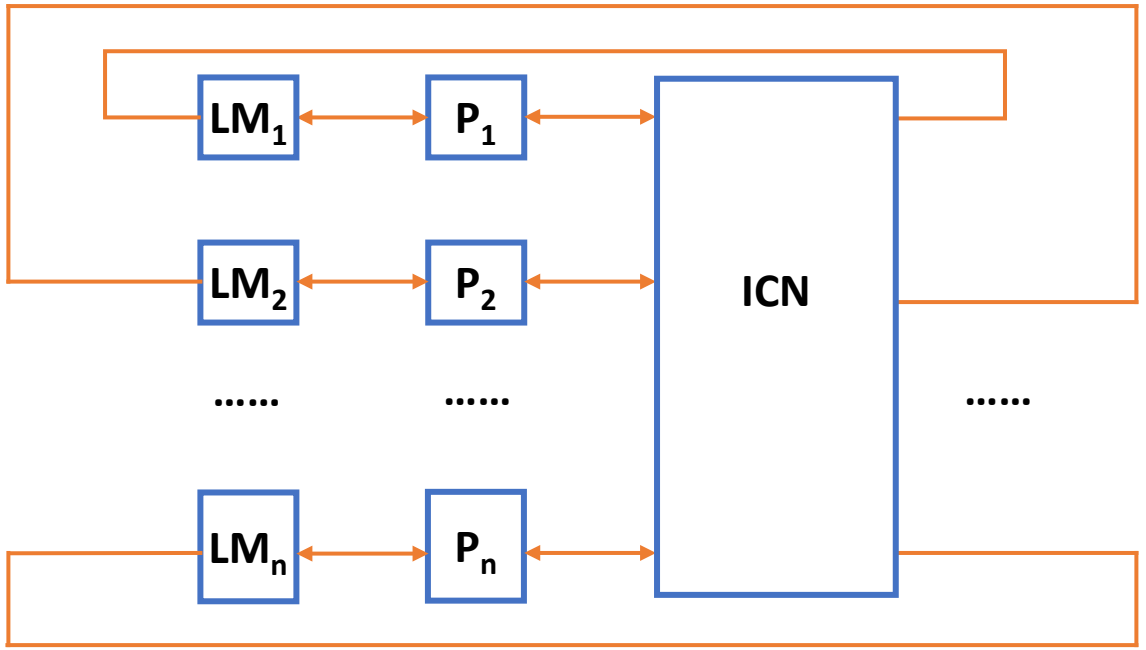


MIMD Architecture

- Different memory access models of MIMD multiprocessor system
  - Uniform Memory Access (UMA)
  - Non Uniform Memory Access (NUMA)
  - Cache Only Memory Access (COMA)

- Further division of MIMD multi-computer system
  - Massively Parallel Processors (MPP)
  - Cluster of Workstations(COW)

NUMA

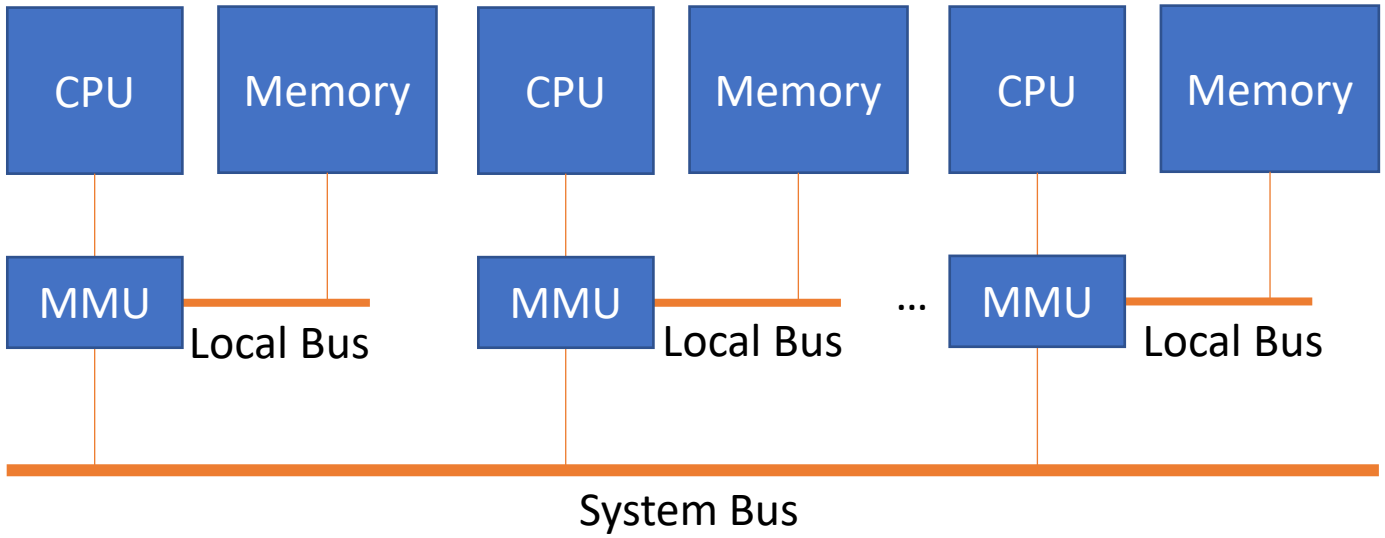


Shared local memory model LM: Local Memory P: Processor

NUMA

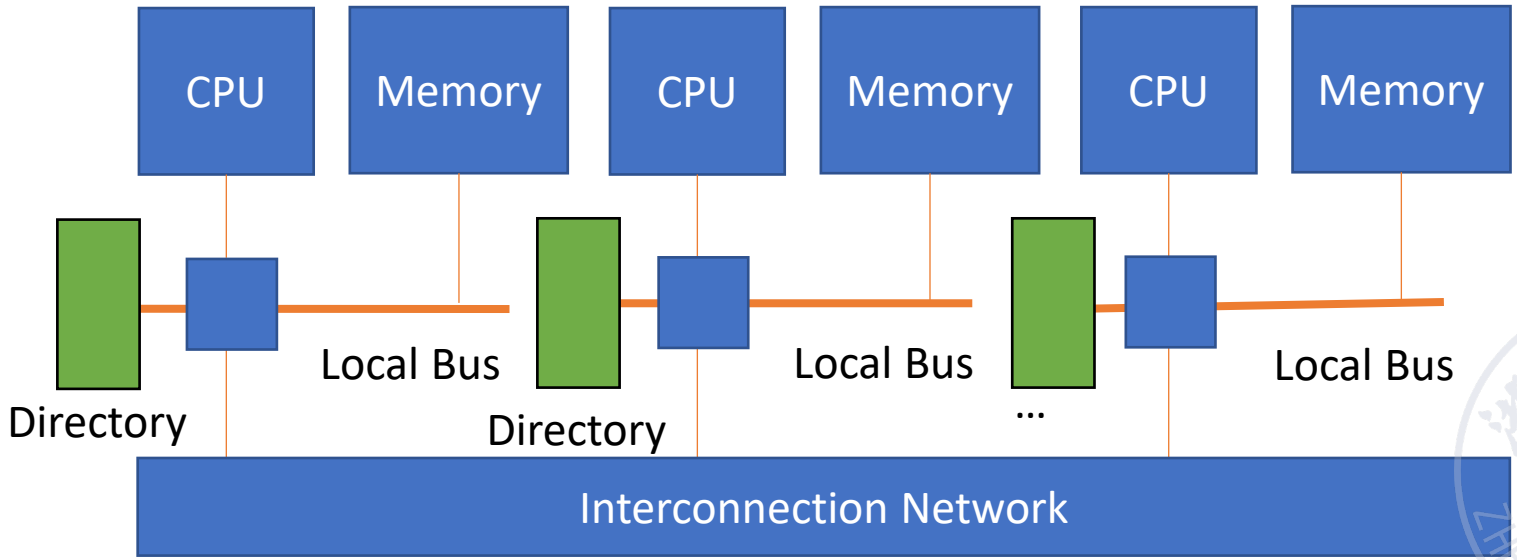
- NC-NUMA

  Non Cache
  Non-Uniform
  Memory Access



- CC-NUMA

  Coherent Cache
  Non-Uniform
  Memory Access



NUMA

- NUMA system features
  - All CPUs share an uniform address space
  - Use LOAD and STORE instructions to access remote memory
  - Access to remote memory is slower than access to local memory
  - The processor in the NUMA system can use cache

- NC-NUMA and CC-NUMA
  - The NUMA system without Cache is called the NC-NUMA multiprocessor system, which means that the remote memory access time is not hidden in this system. If Cache is used, the system will be called a CC-NUMA multiprocessor system.

MIMD Architecture

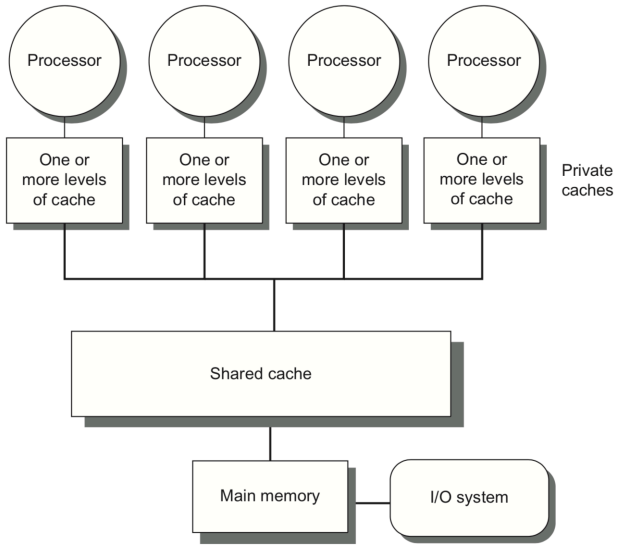
- Different memory access models of MIMD multiprocessor system
  - Uniform Memory Access (UMA)
  - Non Uniform Memory Access (NUMA)
  - Cache Only Memory Access (COMA)

- Further division of MIMD multi-computer system
  - Massively Parallel Processors (MPP)
  - Cluster of Workstations(COW)

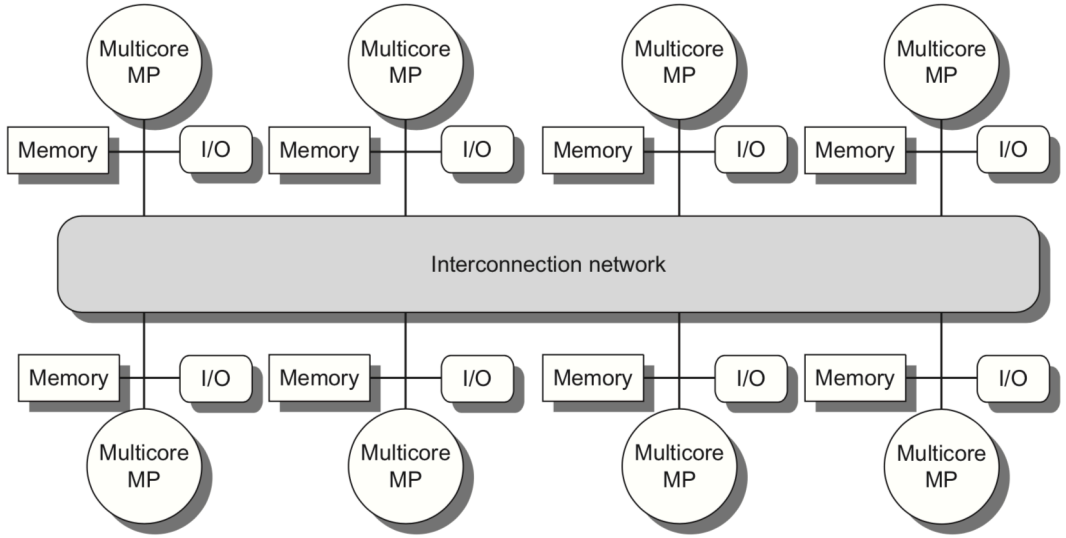
UMA and NUMA

- UMA is also called *symmetric (shared-memory) multiprocessors (SMP)* or *centralized shared-memory multiprocessors*.
- NUMA is called *distributed shared-memory multiprocessor (DSP)*.

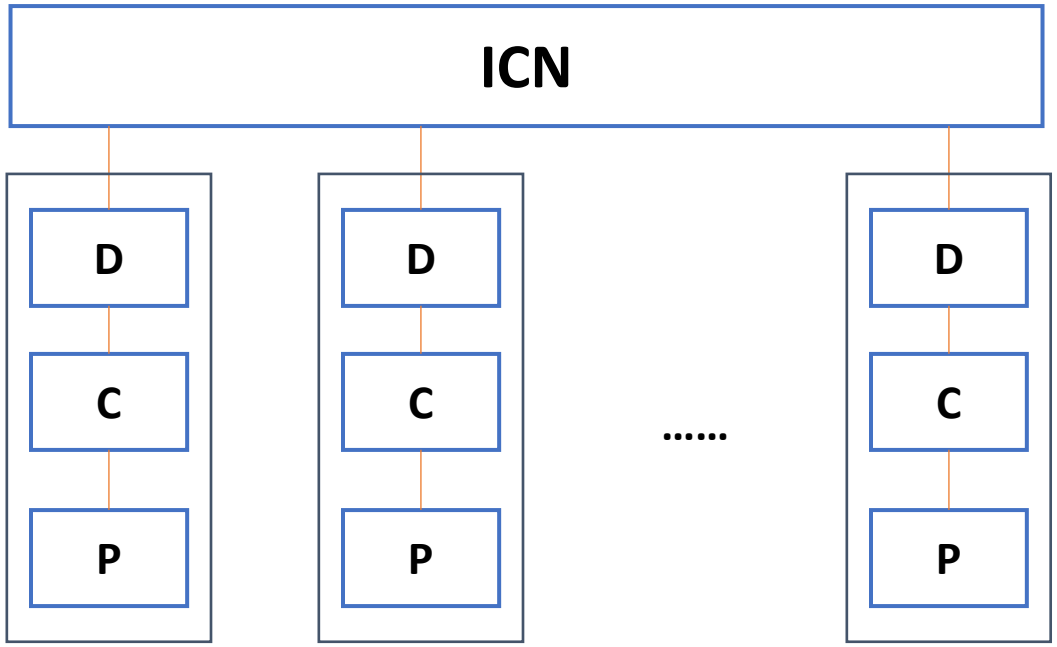
SMP



DSP



COMA

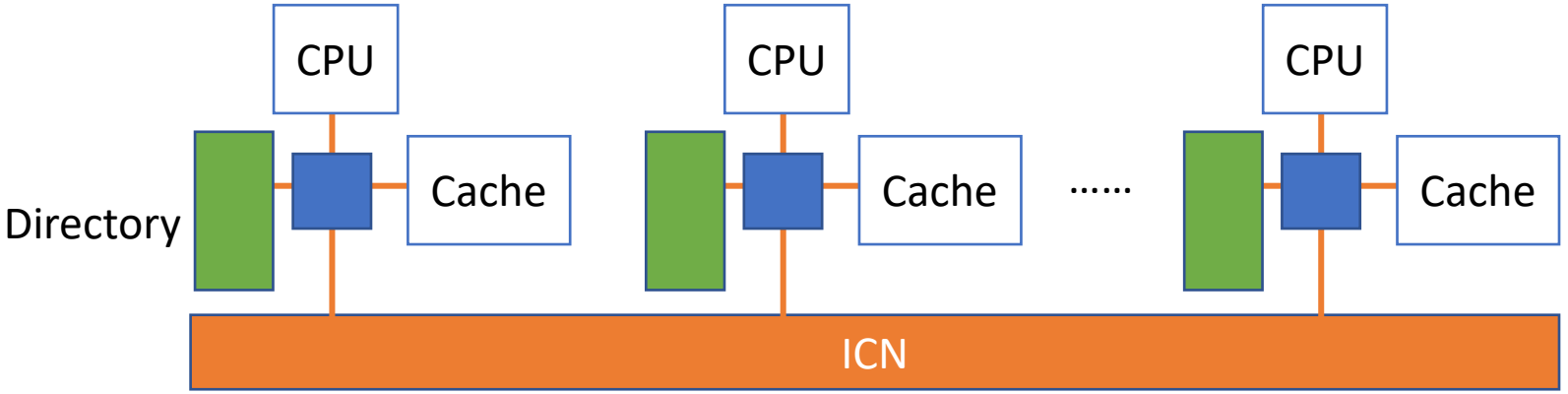


D: Cache Directory **C:** Cache **P:** Processor

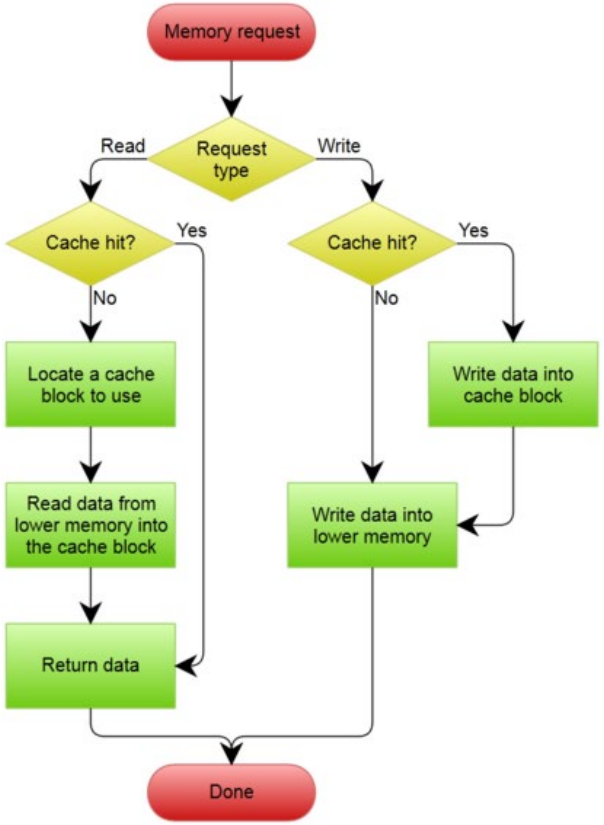
COMA

Characteristics of COMA

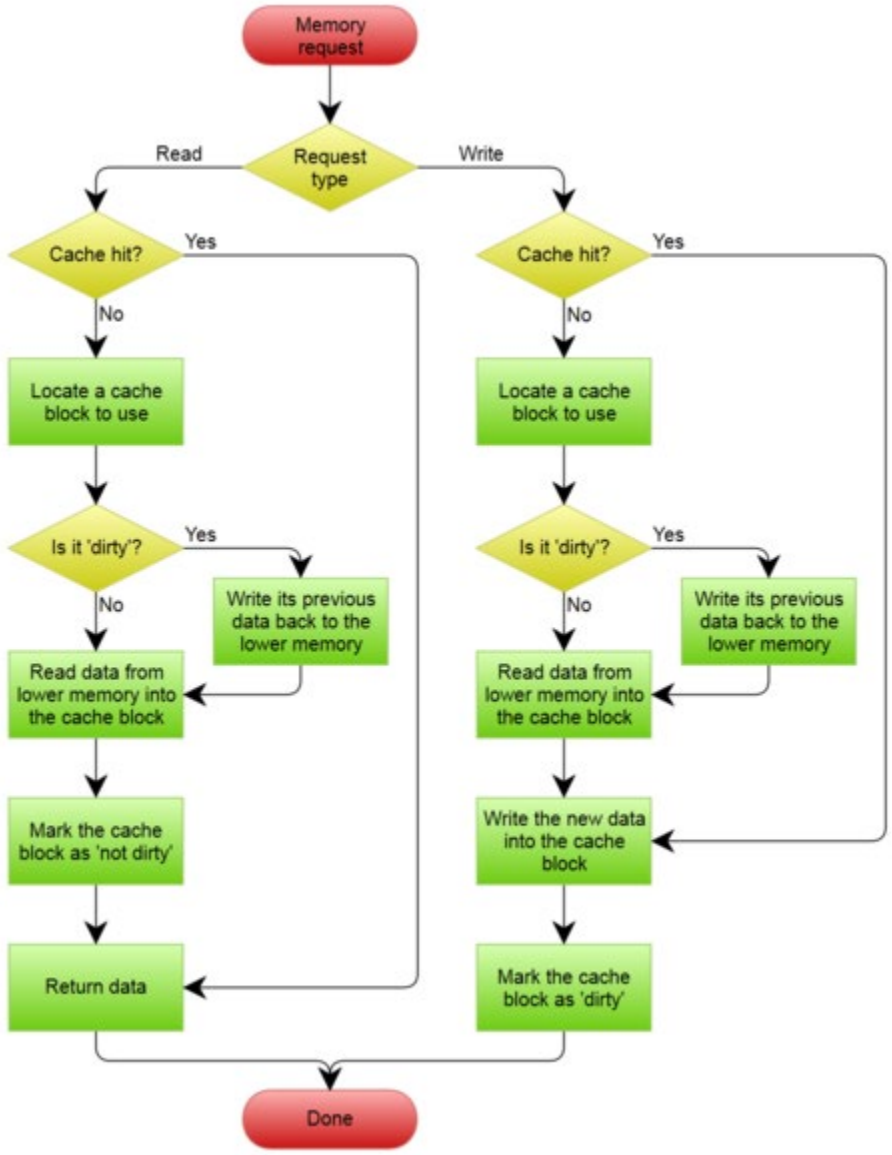
- COMA is a special case of NUMA. There is no storage hierarchy in each processor node, and all caches form a uniform address space.
- Use the distributed cache directory for remote cache access. When using COMA, the data can be allocated arbitrarily at the beginning, because it will eventually be moved to the place where it is used at runtime.



Cache Coherence



A Write-Through cache with No-Write Allocation



A Write-Back cache with Write Allocation

Cache Coherence

- Causes of Cache coherence problems
  - In modern parallel computers, processors often have Cache. One memory data may have multiple copies in the entire system. This leads to the Cache coherence problem.
- Cache coherence protocol
  - A set of rules implemented by Cache, CPU, and memory to prevent different versions of the same data from appearing in multiple Caches forms a Cache coherence protocol.
- Cache coherence protocols can generally be divided into two categories
  - Bus snooping protocol
  - Directory based protocol

Consistency and Coherence

- Coherence
  - All reads by any processor must return the most recently written value
  - Writes to the same location by any two processors are seen in the same order by all processors

- Consistency
  - When a written value will be returned by a read
  - If a processor writes location A followed by location B, any processor that sees the new value of B must also see the new value of A

Cache Coherence

- For UMA: *Snoopy coherence protocols*
  - In the snoopy coherence protocols, all processors snoop the bus. When a processor modifies the data in the private cache, it broadcasts invalid information or updated data on the bus to invalidate or update other copies.

- For NUMA: *Directory protocol*
  - The directory protocol uses a directory to record which processors in the system have copies of certain storage blocks in the cache. When a processor wants to write a shared block, it sends an invalid signal to those processors that have copies of the block through the directory in a "point-to-point” way, so that all other copies are invalidated.

Snoopy Coherence Protocols

- Write-through Cache Coherency Protocol
  - While writing the data in the cache line, the content in the corresponding memory is also modified, and the data in the memory is kept up to date at any time.

- Write-back Cache consistency protocol
  - The write operation does not directly write to the memory. On the contrary, when the cache line is modified, a certain bit in the cache is set to indicate that the data in the cache line is correct but the data in the memory is out of date. Of course, the line will eventually be written back to memory, but it may be after multiple write operations.

Write-through Cache Coherency Protocol

- Four situations when the monitoring cache performs read and write operations according to this protocol

	Local Request	Remote Request
Read Miss	Access data from memory	
Read Hit	Use local cache data	
Write Miss	Modify data in memory	
Write Hit	Modify cache and memory	Invalidate the cache item

- There are many changes in the basic protocol of write direct Cache consistency
  - Whether to use **Update Strategy** or **Invalidate Strategy** for remote write hits
  - Whether to transfer the corresponding word into the cache when the cache write is missing, this is whether to use the **Write-allocate Policy**.

Write Invalidation Protocol

three block states ***(MSI protocol)***

- ***I*nvalid**
- ***S*hared**

  indicates that the block in the private cache is potentially shared
- ***M*odified**

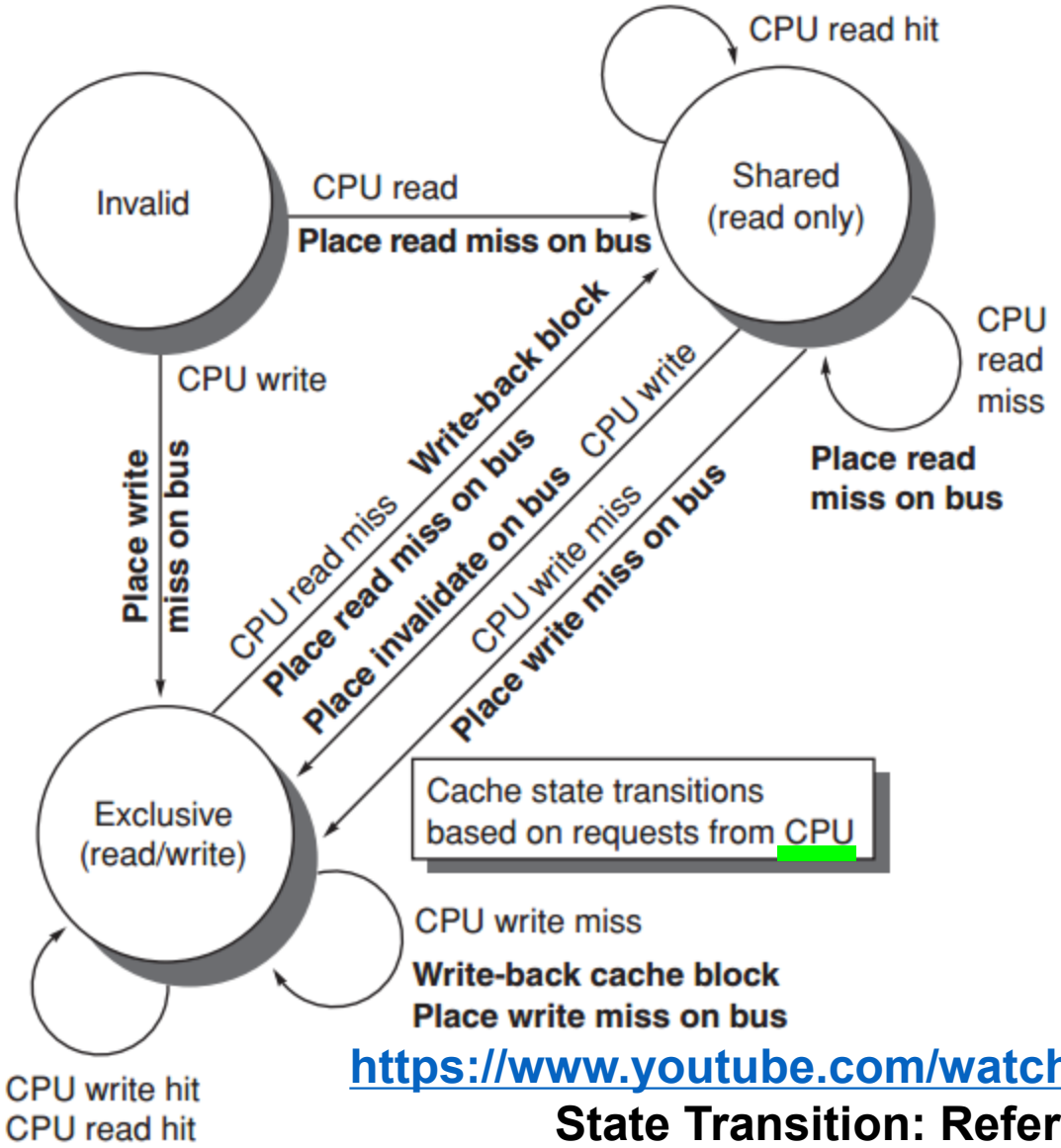
  indicates that the block has been updated in the private cache;

  implies that the block is **exclusive**

Write Invalidation Protocol

Request	Source	State of addressed cache block	Type of cache action	Function and explanation
Read hit	processor	shared or modified	normal hit	Read data in cache.
Read miss	processor	invalid	normal miss	Place read miss on bus.
Read miss	processor	shared	replacement	Address conflict miss: place read miss on bus.
Read miss	processor	modified	replacement	Address conflict miss: write back block, then place read miss on bus.
Write hit	processor	modified	normal hit	Write data in cache.
Write hit	processor	shared	coherence	Place invalidate on bus. These operations are often called upgrade or *ownership* misses, since they do not fetch the data but only change the state.
Write miss	processor	invalid	normal miss	Place write miss on bus.
Write miss	processor	shared	replacement	Address conflict miss: place write miss on bus.
Write miss	processor	modified	replacement	Address conflict miss: write back block, then place write miss on bus.
Read miss	bus	shared	no action	Allow memory to service read miss.
Read miss	bus	modified	coherence	Attempt to share data: place cache block on bus and change state to shared.
Invalidate	bus	shared	coherence	Attempt to write shared block; invalidate the block.
Write miss	bus	shared	coherence	Attempt to write block that is shared; invalidate the cache block.
Write miss	bus	modified	coherence	Attempt to write block that is exclusive elsewhere: write back the cache block and make its state invalid.

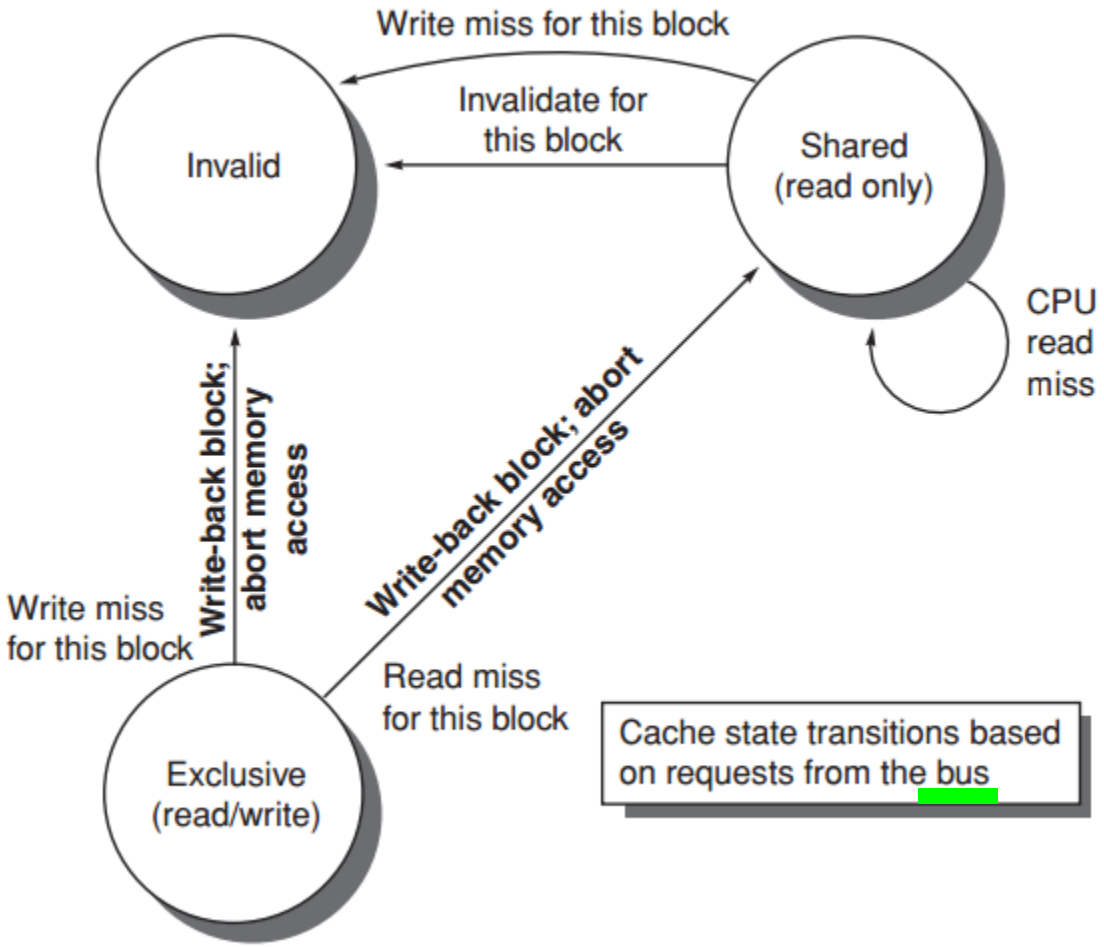
Write Invalidation Protocol



https://www.youtube.com/watch?v=gAUVAel-2Fg

State Transition: Refer to previous table

Write Invalidation Protocol



MSI Extensions

- **MESI**

  exclusive: indicates when a cache block is resident only in a single cache but is clean

  exclusive->read by others->shared

  exclusive->write->modified

  add figures?

  refer to https://www.youtube.com/watch?v=OLGEtXV4U3I

  MESI writes exclusive to modified silently, without broadcast on bus

MSI Extensions

- **MOESI**

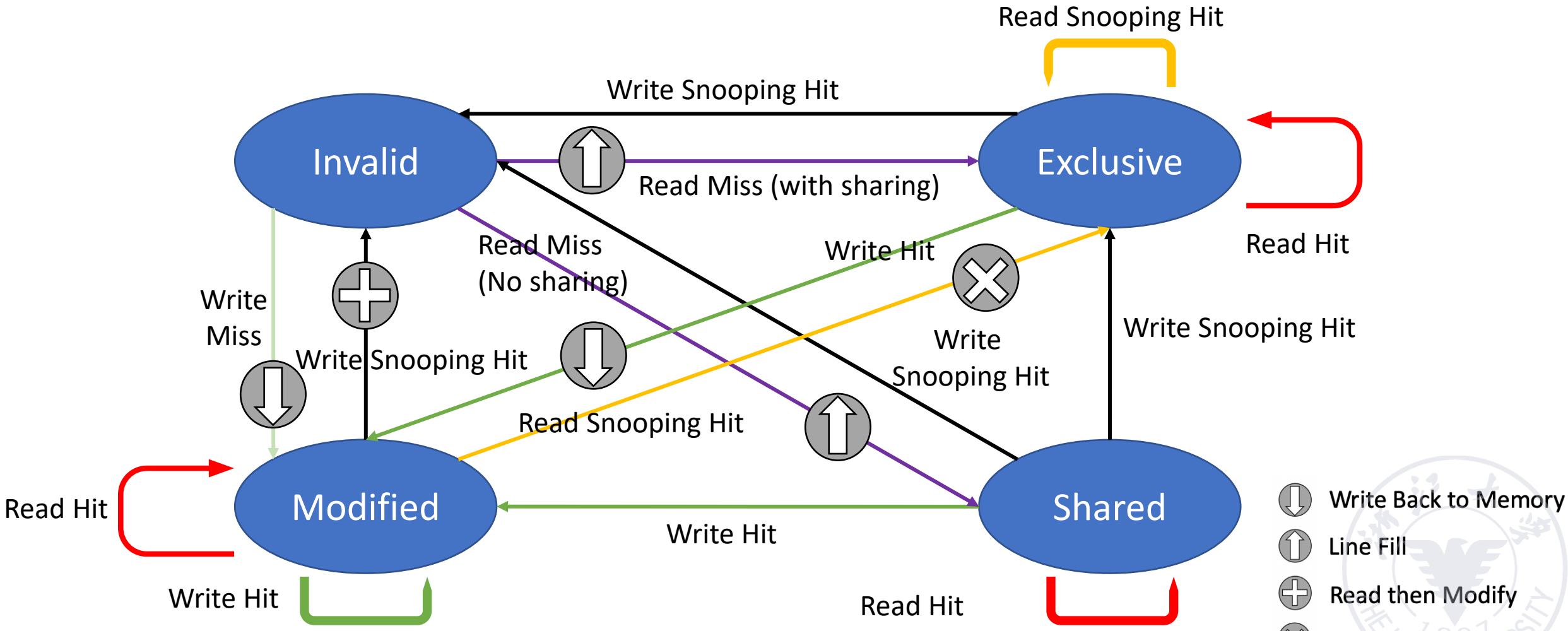
  owned: indicates that the associated block is owned by that cache and out-of-date in memory

  Modified -> Owned without writing the shared block to memory

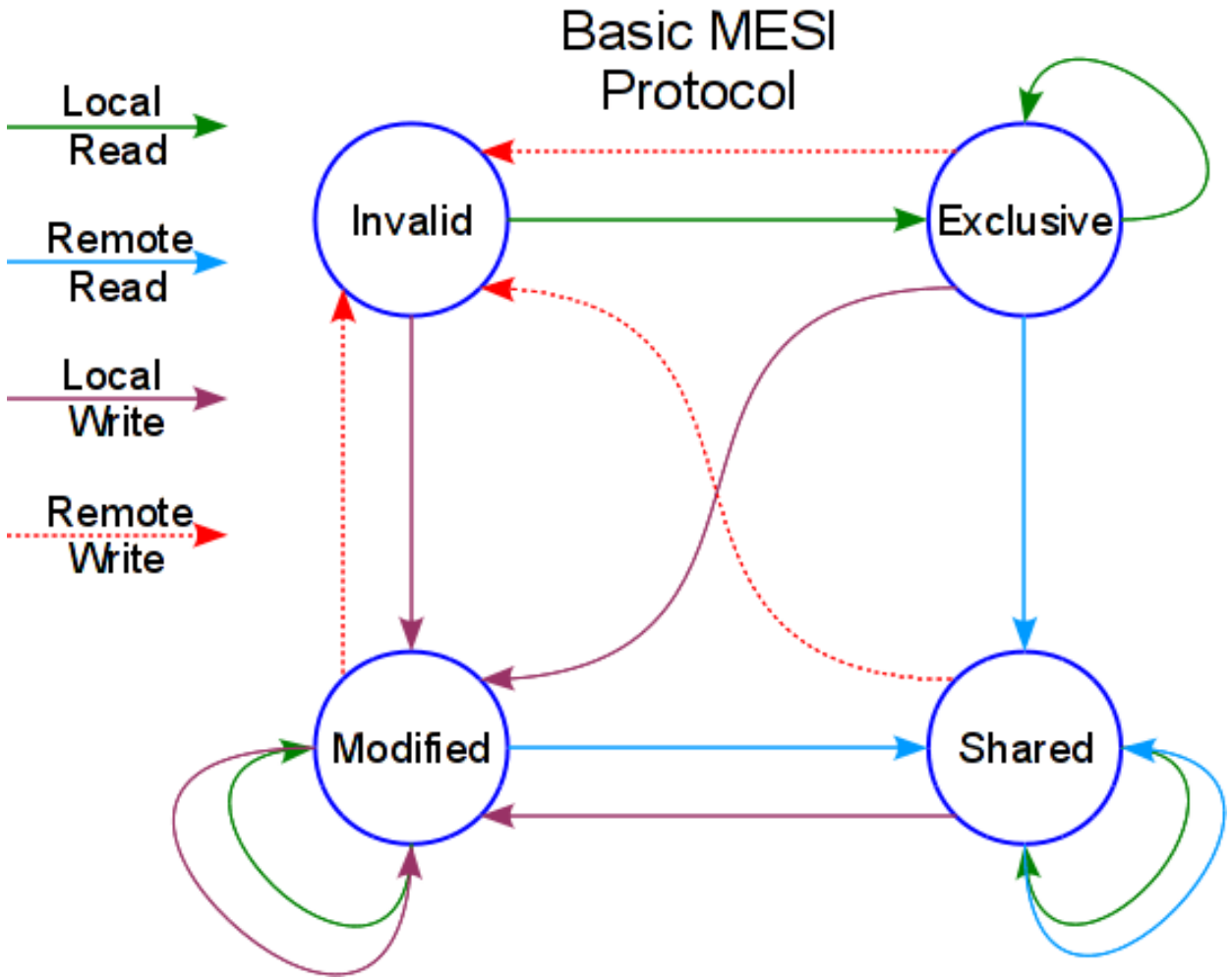
Write-back Cache Coherency Protocol

- MESI protocol: It is named after the initial letter of the four states used in the protocol. Each item in this protocol is in one of the following four states:
  - Invalid: The data contained in the cache item is invalid.
  - Shared: This row of data exists in multiple cache items, and the data in the memory is the latest.
  - Exclusive: No other cache items include this row of data, and the data in memory is the latest.
  - Modified: The data of the item is valid, but the data in the memory is invalid, and there is no copy of the data in other cache items.

MESI protocol state transition rules



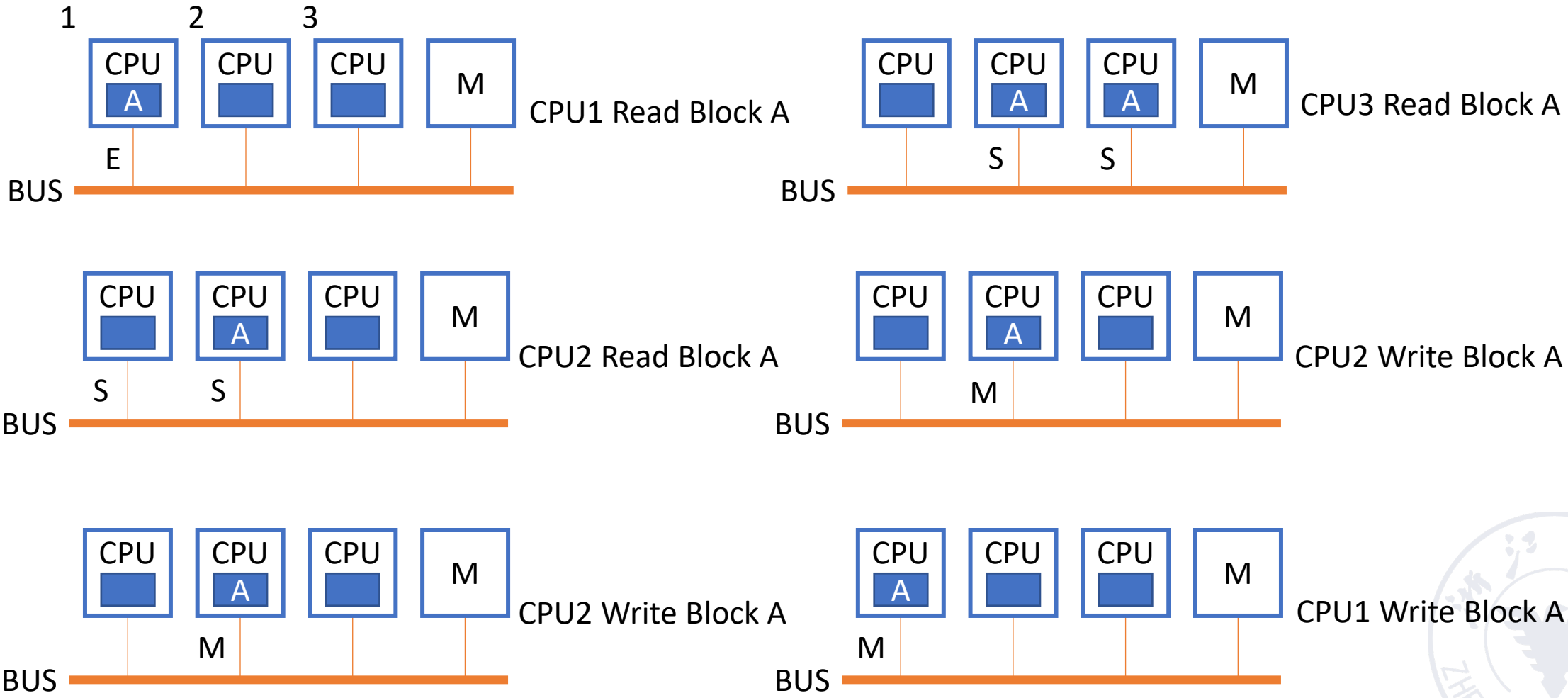
MESI protocol state transition rules



Different response to different bus activities

- Processor read request: the valid state (M/S/E) line read hits, the state remains unchanged. When the read is lost, a new line is allocated and data is read, and the state changes from I to S or E.
- Processor write request: valid state (M/S/E) line write hit, M/E becomes M, S becomes "unique" (other Cache shared copy is invalid) and then enters M state; when write is lost, no According to the write allocation method, write and save and then do not read in. According to the write allocation method, read this line first (I) and change to M state after modification. Both methods have the processing process of "unique" first (other Cache shared copies are invalid) .
- Cache read snooping hits: S state remains unchanged, E state becomes S state, M state preempts the bus, writes back to main memory, and becomes S state.
- Cache write snooping hit: the valid state (S/E) line changes to the I state, and after the M state preempts the bus and writes it back to the main memory, it changes to the I state.

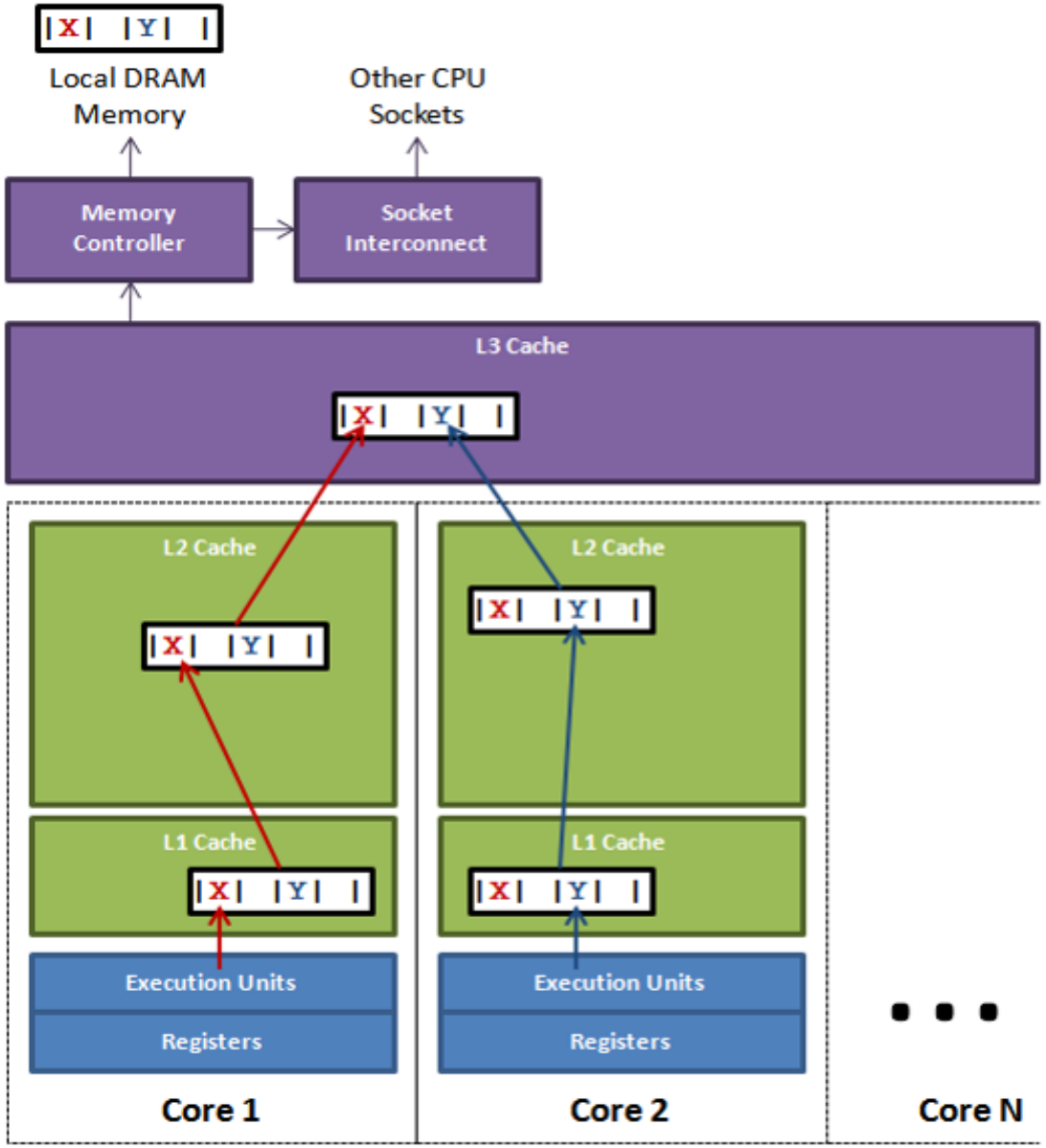
MESI protocol working process



False sharing

```
public class FalseSharingTest {

    public static void main(String[] args) throws InterruptedException
{
        testPointer(new Pointer());
    }

    private static void testPointer(Pointer pointer) throws
InterruptedException {
        long start = System.currentTimeMillis();
        Thread t1 = new Thread(() -> {
            for (int i = 0; i < 100000000; i++) {
                pointer.x++;
            }
        });

        Thread t2 = new Thread(() -> {
            for (int i = 0; i < 100000000; i++) {
                pointer.y++;
            }
        });

        t1.start();
        t2.start();
        t1.join();
        t2.join();

        System.out.println(System.currentTimeMillis() - start);
        System.out.println(pointer);
    }
}

class Pointer {
    volatile long x;
    volatile long y;
}
```

False sharing



Avoid False sharing

I.

```
class Pointer {
    volatile long x;
    long p1, p2, p3, p4, p5, p6, p7;
    volatile long y;
}
```

II.

```
class Pointer {
    MyLong x = new MyLong();
    MyLong y = new MyLong();
}

class MyLong {
    volatile long value;
    long p1, p2, p3, p4, p5, p6, p7;
}
```

III.

```
@sun.misc.Contended
class MyLong {
    volatile long value;
}
```

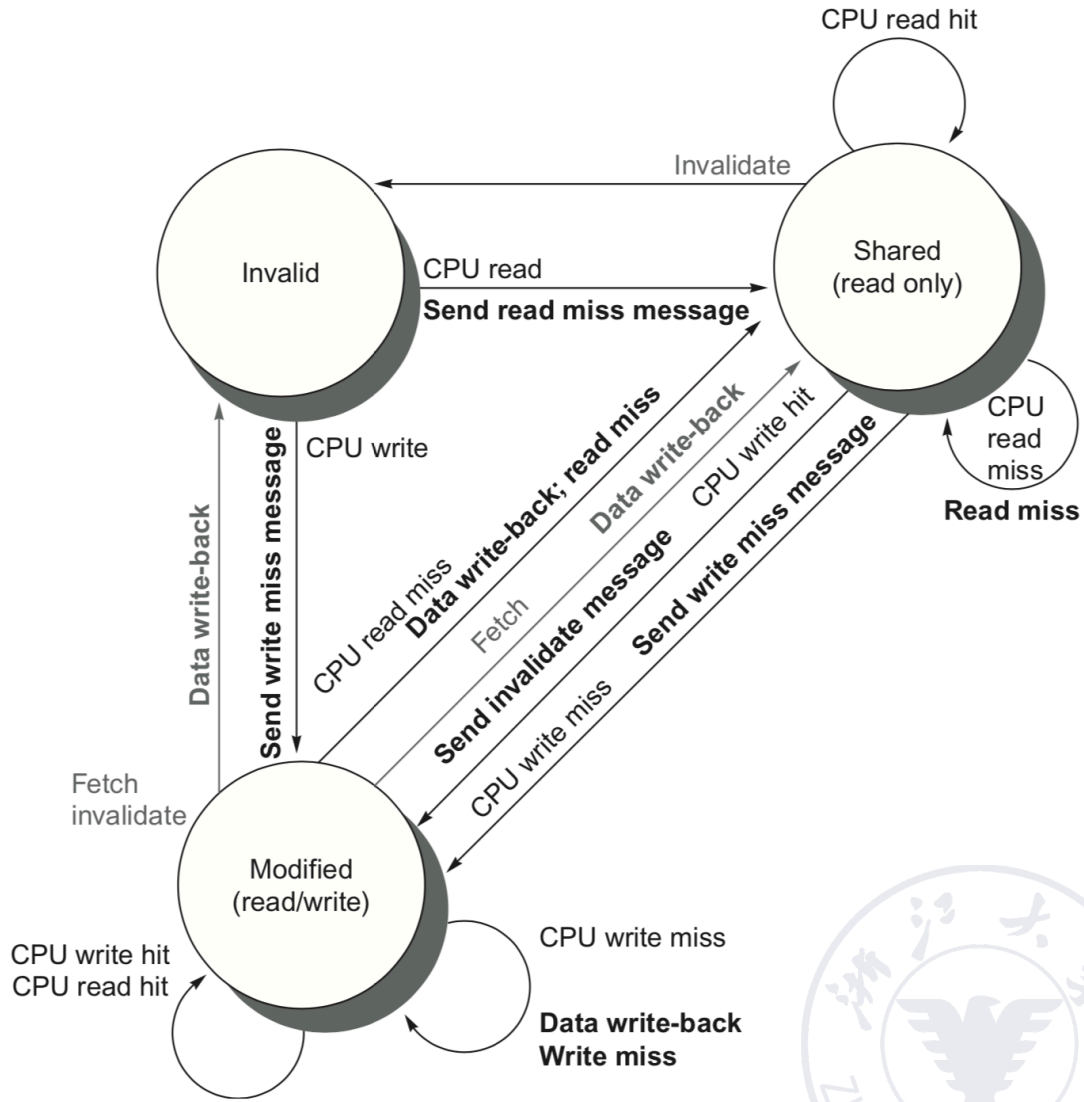
Directory protocol

- For each block, maintain state:
  - *Shared*
    - One or more nodes have the block cached, value in memory is up-to-date
    - Set of node IDs
  - *Invalid*
  - *Modified*
    - Exactly one node has a copy of the cache block, value in memory is out-of-date
    - Owner node ID
- Directory maintains block states and sends invalidation messages

Directory protocol

- State transition diagram for ***an individual cache block*** in a directory-based system.

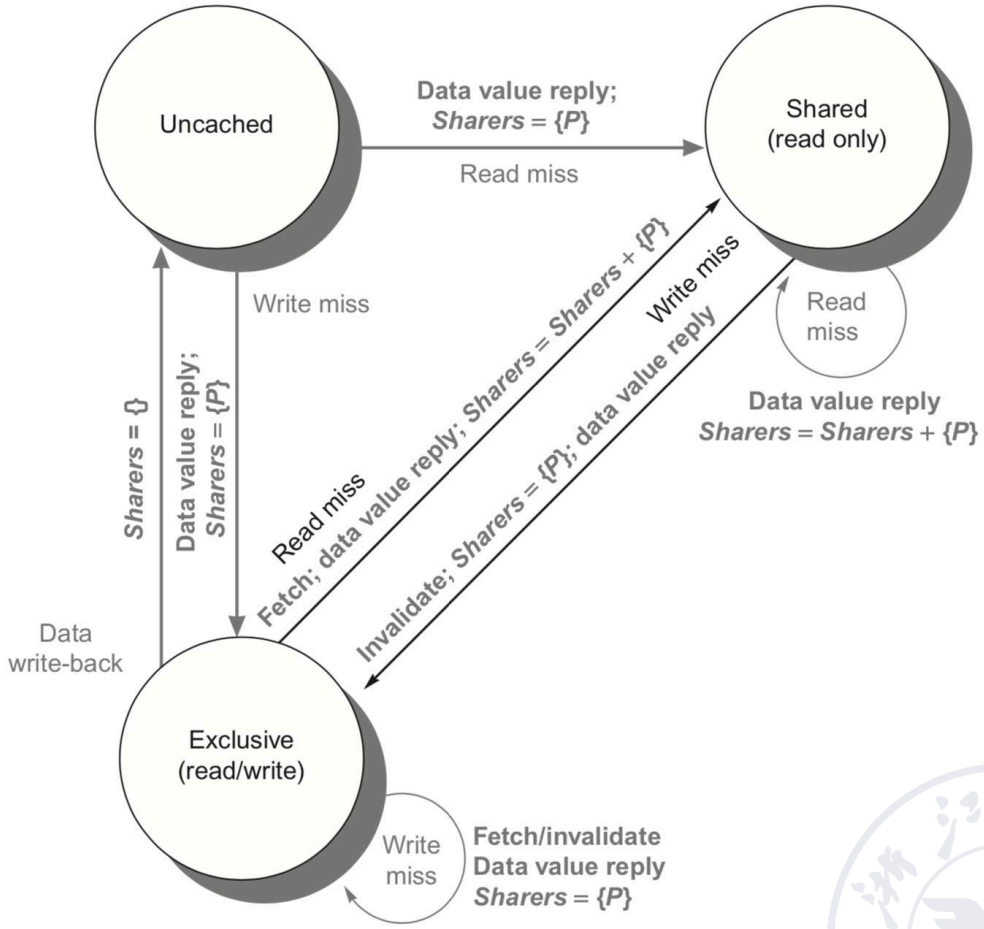
- Requests by the local processor are shown in black, and those from the home directory are shown in gray.



Directory protocol

- The state transition diagram for ***the directory*** has the same states and structure as the transition diagram for an individual cache.

- All actions are in gray because they are all externally caused. Bold indicates the action taken by the directory in response to the request.



Directory protocol

- For uncached block:
  - Read miss
    - Requesting node is sent the requested data and is made the only sharing node, block is now shared
  - Write miss
    - The requesting node is sent the requested data and becomes the sharing node, block is now exclusive
- For shared block:
  - Read miss
    - The requesting node is sent the requested data from memory, node is added to sharing set
  - Write miss
    - The requesting node is sent the value, all nodes in the sharing set are sent invalidate messages, sharing set only contains requesting node, block is now exclusive

Directory protocol

- For exclusive block:
  - Read miss
    - The owner is sent a data fetch message, block becomes shared, owner sends data to the directory, data written back to memory, sharers set contains old owner and requestor
  - Data write back
    - Block becomes uncached, sharer set is empty
  - Write miss
    - Message is sent to old owner to invalidate and send the value to the directory, requestor becomes new owner, block remains exclusive

Memory Consistency

- Example:

Processor 1:	Processor 2:
A=0	B=0
…	…
A=1	B=1
if (B==0) …	if (A==0) …

- Sequential consistency (reduces potential performance):
  - Result of execution should be the same as long as:
    - Accesses on each processor were kept in order
    - Accesses on different processors were arbitrarily interleaved

Relaxed Consistency Models

- Key idea: allow reads and writes to complete out of order, but to use synchronization operations to enforce ordering

Relaxed Consistency Models

- Rules:
  - X → Y
    - Operation X must complete before operation Y is done
    - Sequential consistency requires:
      - R → W, R → R, W → R, W → W
  - Relax W → R
    - “Total store ordering”
  - Relax W → W
    - “Partial store order”
  - Relax R → W and R → R
    - “Weak ordering” and “release consistency”

MIMD Architecture

- Different memory access models of MIMD multiprocessor system
  - Uniform Memory Access (UMA)
  - Non Uniform Memory Access (NUMA)
  - Cache Only Memory Access (COMA)

- Further division of MIMD multi-computer system
  - Massively Parallel Processors (MPP)
  - Cluster of Workstations(COW)

MPP

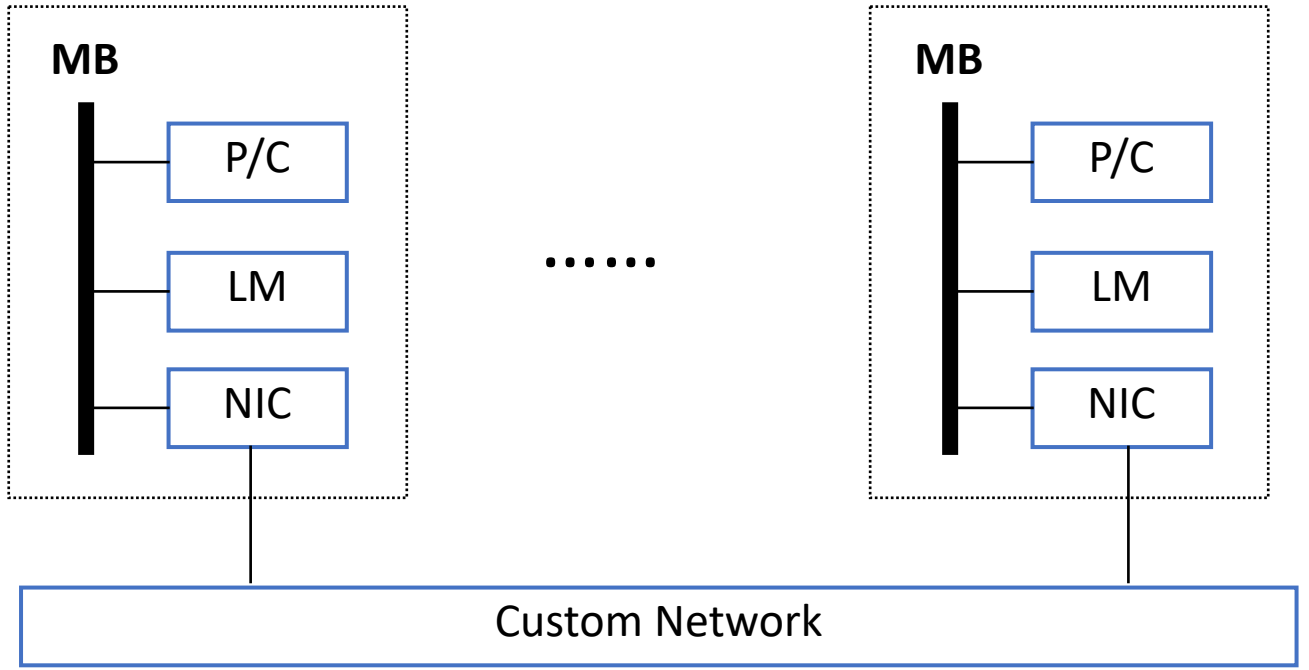
- The MPP system is a massively parallel computer system composed of hundreds of processors.
- In the past, it was mainly used for calculation-oriented occasions such as scientific calculation and engineering simulation, but it is also widely used in commercial and network applications.
- Development is difficult, the price is high, and the market is limited. It is a symbol of the country's comprehensive strength.

Characteristics of MPP

- MPP systems generally use standard commercial CPUs as their processors.
- The MPP system uses a high-performance private interconnection network, which can deliver messages with low latency and high bandwidth.
- The MPP system has powerful input/output capabilities.
- The MPP system is capable of special fault-tolerant processing.

MPP

LM： Local Memory
NIC： Network interface circuit
MB： Memory Bus



MIMD Architecture

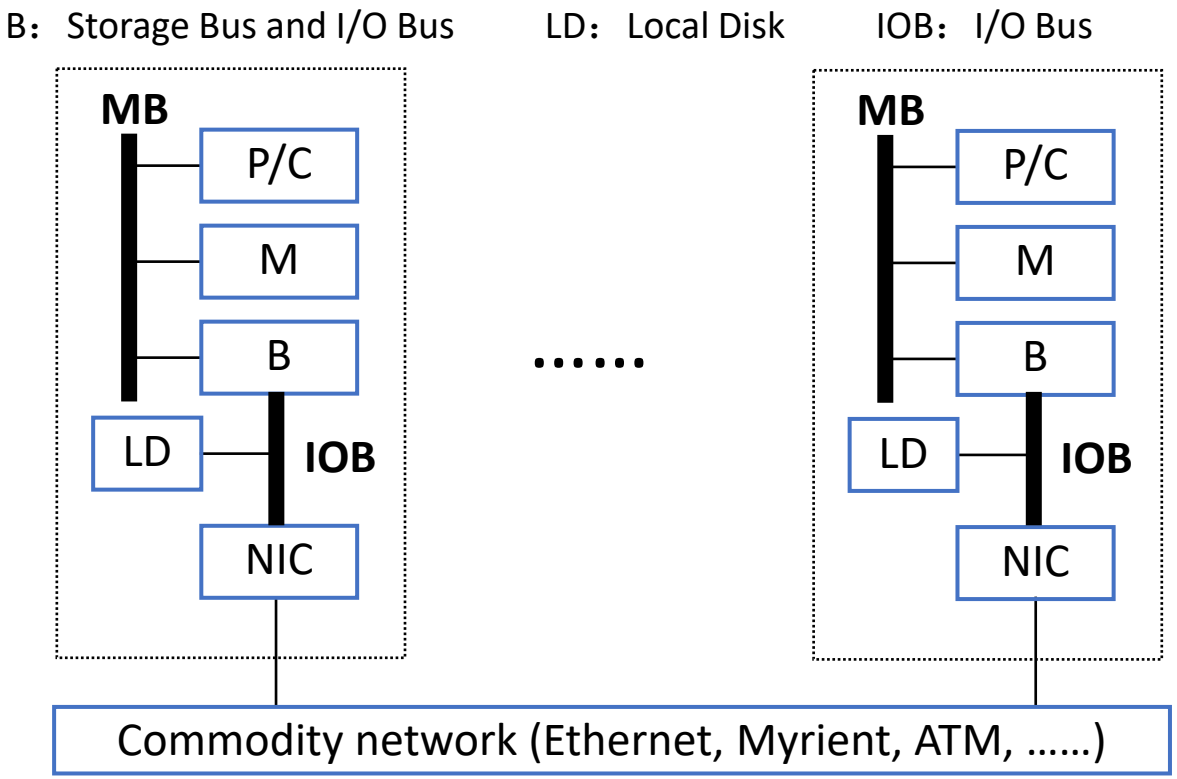
- Different memory access models of MIMD multiprocessor system
  - Uniform Memory Access (UMA)
  - Non Uniform Memory Access (NUMA)
  - Cache Only Memory Access (COMA)

- Further division of MIMD multi-computer system
  - Massively Parallel Processors (MPP)
  - Cluster of Workstations(COW)

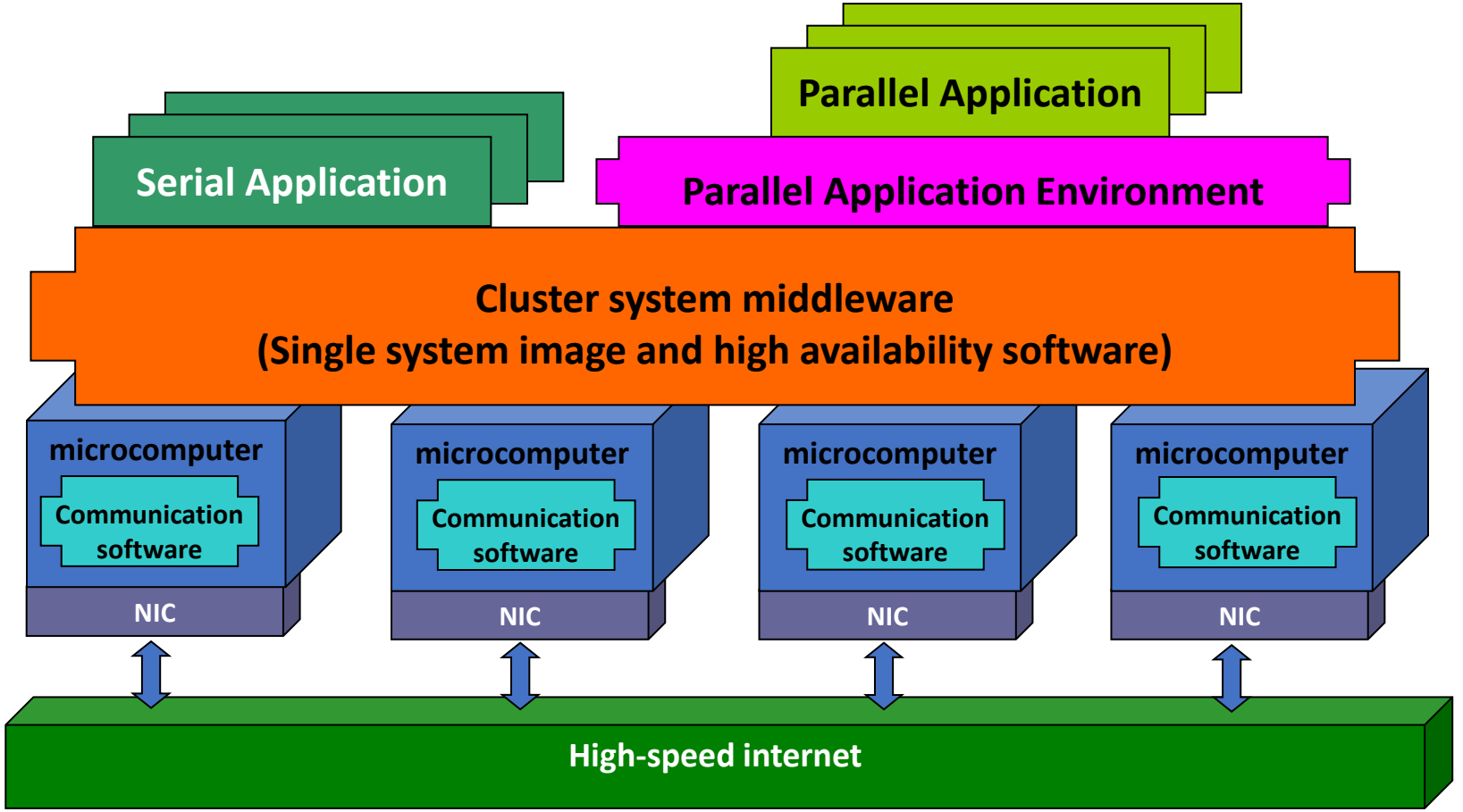
COW

- The COW system is composed of a large number of PCs or workstations connected together through a commercial network.
- COW can be assembled completely using commercially available components. These commercial components are mass-produced products, so they can achieve higher cost performance.
- There are two main types of COW that dominate: centralized and decentralized.

COW


B：Storage Bus and I/O Bus
LD：Local Disk
IOB：I/O Bus
MB
P/C
M
B
LD
IOB
NIC
.....
MB
P/C
M
B
LD
IOB
NIC
Commodity network (Ethernet, Myrient, ATM, ……)

COW Architecture

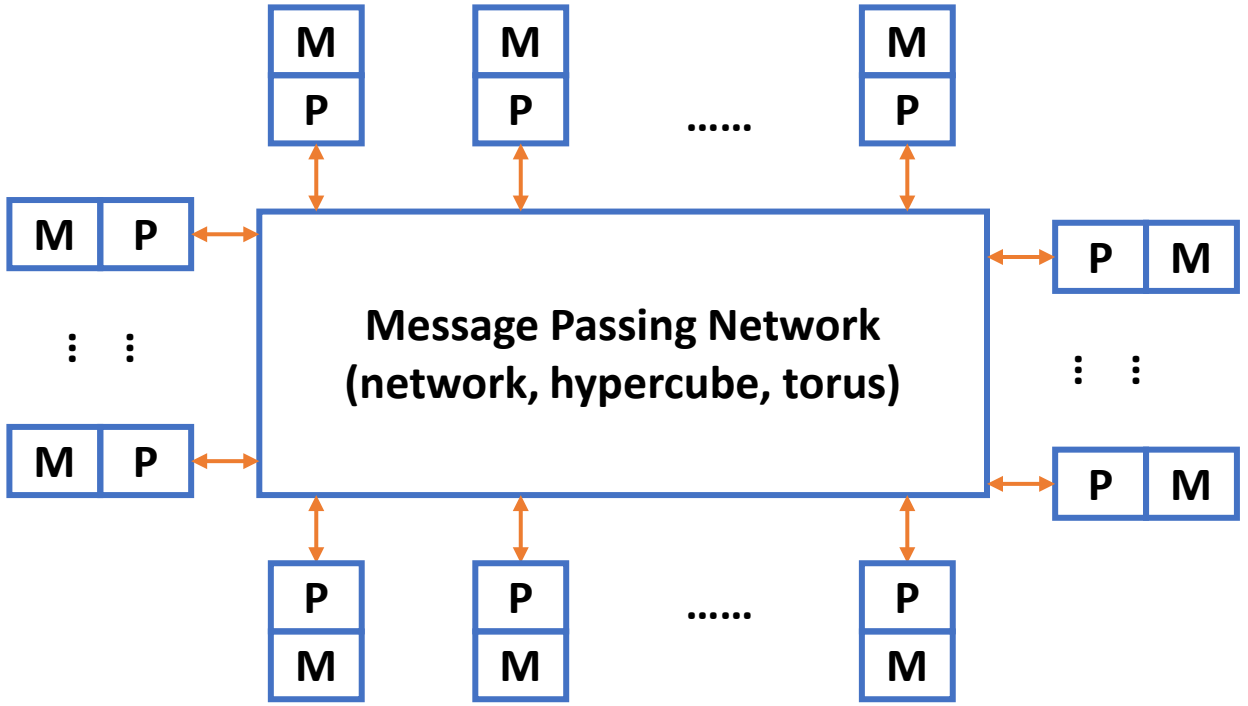


Difference between COW and MPP

- First, the nodes of COW are more complete computers, and computers can be homogeneous or heterogeneous. The nodes have their own disks and reside their own completed operating systems; and generally have a certain degree of autonomy. The nodal computer can still run without COW. However, MPP system nodes generally have no disks and only reside in the operating system kernel.
- Second, MPP uses the manufacturer's proprietary (or patented) high-speed communication network; COW generally uses publicly sold standard high-speed local area networks or system area networks, and the network is usually connected to the I/O bus of the node computer (Loosely coupled), while the MPP network interface is connected to the storage bus of the processing node (tightly coupled).

Multi-computer memory access model

- NORMA (No-Remote Memory Access)



Messaging package

- Programming for multiple computer systems requires special software, which generally includes library functions for handling inter-process communication and synchronization.
- There are many kinds of messaging systems, among which PVM and MPI are two popular messaging systems that can be used in most multi-computer systems.
  - PVM (parallel virtual machine): Parallel computing tool software jointly developed by Oak Ridge National Laboratory and several universities in the United States.
  - MPI (message passing interface): MPI is a message passing standard, which was gradually produced by the MPI committee in a series of meetings held from November 1992 to January 1994
- Both PVM and MPI can be used as a basis for easy redevelopment. PVM first appeared and has become the de facto standard in recent years, but now MPI has challenged the dominance of PVM.

Domain-Specific Architectures

- Moore’s Law enabled:
  - Deep memory hierarchy
  - Wide SIMD units
  - Deep pipelines
  - Branch prediction
  - Out-of-order execution
  - Speculative prefetching
  - Multithreading
  - Multiprocessing

- Objective:
  - Extract performance from software that is oblivious to architecture

Guidelines for DSAs

- Use dedicated memories to minimize data movement
- Invest resources into more arithmetic units or bigger memories
- Use the easiest form of parallelism that matches the domain
- Reduce data size and type to the simplest needed for the domain
- Use a domain-specific programming language

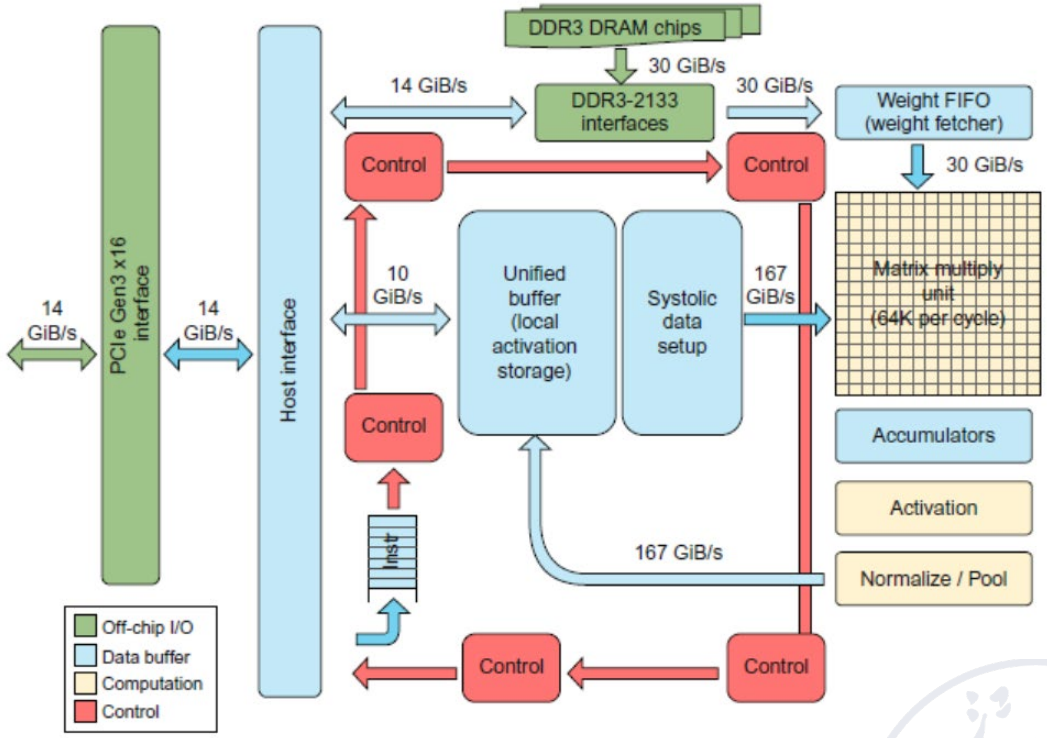
Example: Convolutional Neural Network

Abstraction in Computer Architecture View

- Batches:
  - Reuse weights once fetched from memory across multiple inputs
  - Increases operational intensity
- Quantization
  - Use 8- or 16-bit fixed point
- Summary:
  - Need the following kernels:
    - *Matrix-vector multiply*
    - *Matrix-matrix multiply*
    - *ReLU*
    - *Sigmoid*
    - *…*

Tensor Processing Unit

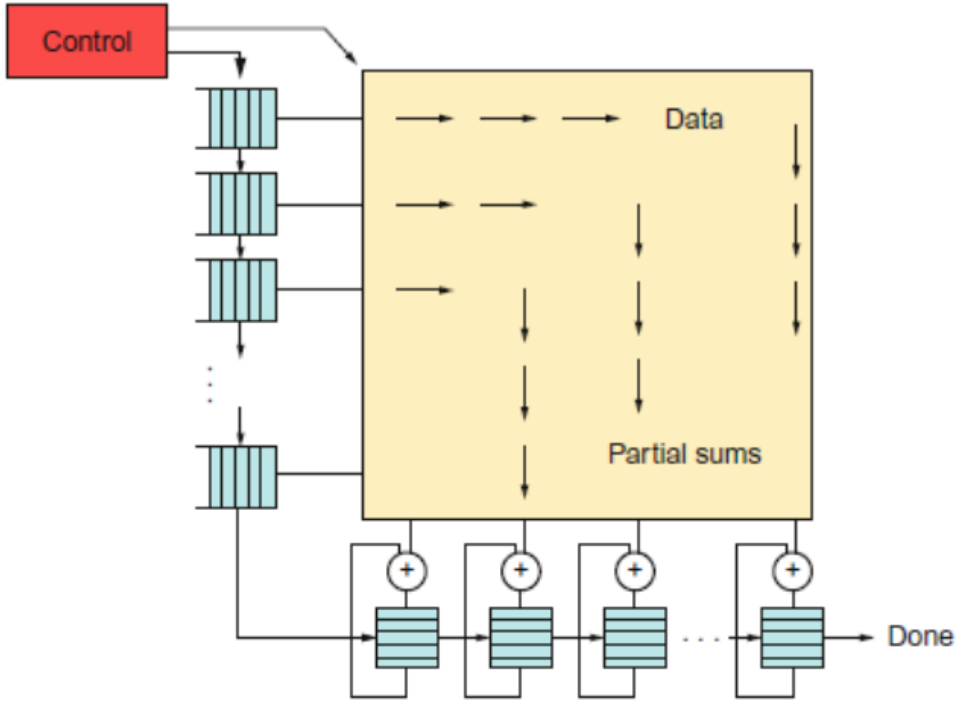
- Google's DNN ASIC
- 256 x 256 8-bit matrix multiply unit
- Large software-managed scratchpad
- Coprocessor on the PCIe bus

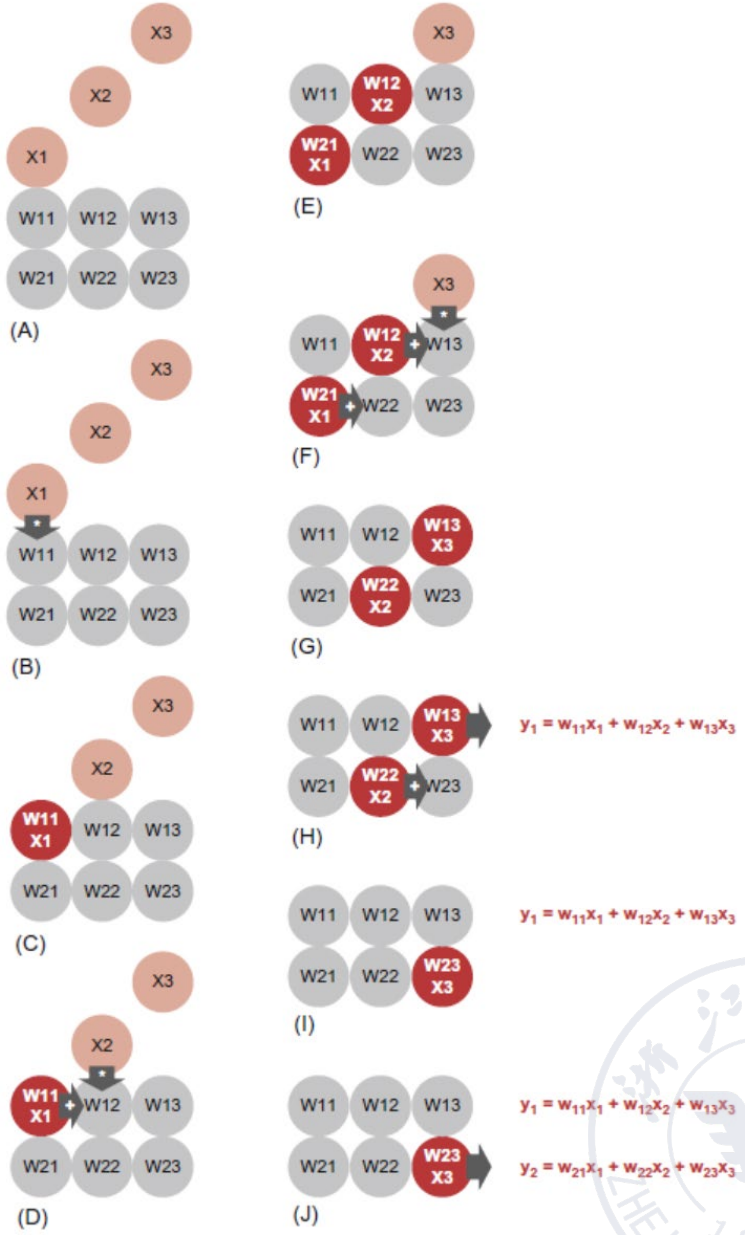


TPU ISA

- **Read_Host_Memory**
  - Reads memory from the CPU memory into the unified buffer
- **Read_Weights**
  - Reads weights from the Weight Memory into the Weight FIFO as input to the Matrix Unit
- **MatrixMatrixMultiply/Convolve**
  - Perform a matrix-matrix multiply, a vector-matrix multiply, an element-wise matrix multiply, an element-wise vector multiply, or a convolution from the Unified Buffer into the accumulators
  - takes a variable-sized B*256 input, multiplies it by a 256x256 constant input, and produces a B*256 output, taking B pipelined cycles to complete
- **Activate**
  - Computes activation function
- **Write_Host_Memory**
  - Writes data from unified buffer into host memory

Matrix Calculations in TPU

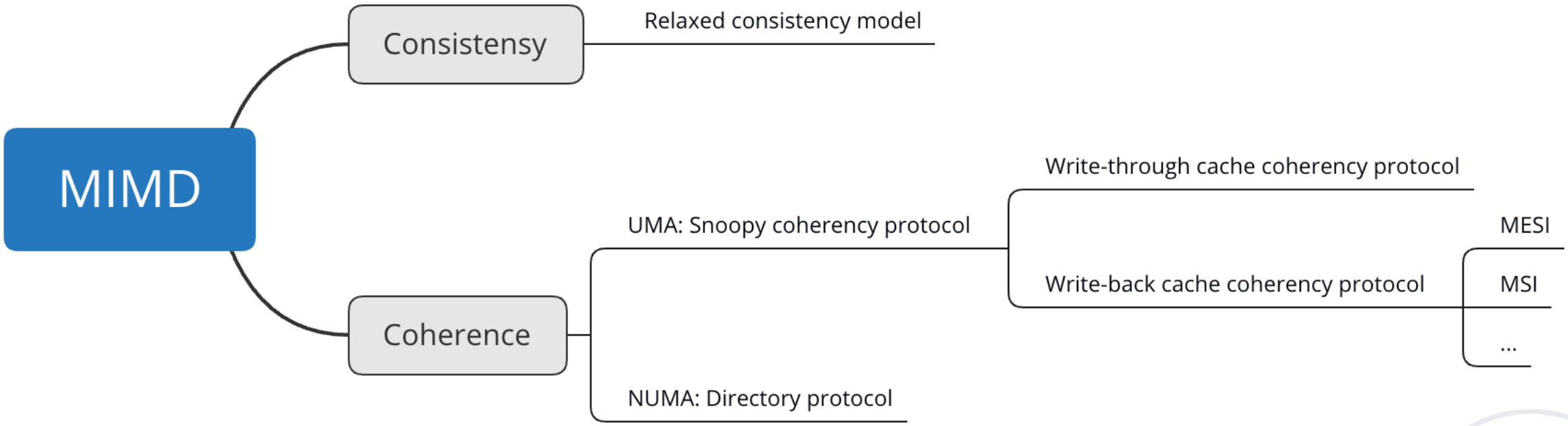




The TPU and the Guidelines

- Use dedicated memories
  - 24 MiB dedicated buffer, 4 MiB accumulator buffers
- Invest resources in arithmetic units and dedicated memories
  - 60% of the memory and 250X the arithmetic units of a server-class CPU
- Use the easiest form of parallelism that matches the domain
  - Exploits 2D SIMD parallelism
- Reduce the data size and type needed for the domain
  - Primarily uses 8-bit integers
- Use a domain-specific programming language
  - Uses TensorFlow

Summary


MIMD
Consistensy
Relaxed consistency model
Coherence
UMA: Snoopy coherency protocol
Write-through cache coherency protocol
MESI
Write-back cache coherency protocol
MSI
...
NUMA: Directory protocol

Summary

Read Cache	Write Cache	
	hit	miss
Read through	***Write-through*** write is done synchronously both to the cache and to the backing store	***Write allocate*** data at the missed-write location is loaded to cache, followed by a write-hit operation.
Read allocate	***Write-back*** writing is done only to the cache	***No-write allocate*** data at the missed-write location is not loaded to cache, and is written directly to the backing store