Course Review

Chapter 1 (6th chapter1)

Fundamentals of Computer Design

★CPU Performance

In order to determine the effect of a design change on the performance experienced by the user, we can use the following relation:

$$CPU\ Execution\ Time = CPU\ Clock\ Cycles \times Clock\ Period$$

Alternatively,

$$CPU\ Execution\ Time = \frac{CPU\ Clock\ Cycles}{Clock\ Rate}$$

Clearly, we can reduce the execution time of a program by either reducing the number of clock cycles required or the length of each clock cycle.

Instruction Count and CPI

- Instruction Count for a program
  - Determined by program, ISA and compiler
- Average cycles per instruction（CPI）
  - Determined by CPU hardware
  - If different instructions have different CPI
    - Average CPI affected by instruction mix

$$CPI = \frac{CPU\ Clock\ Cycles}{Instruction\ Count}$$

$$CPU\ Clock\ Cycles = Instructions\ for\ a\ Program \times Average\ Clock\ Cycles\ Per\ Instruction$$

$$CPU\ Time = Instruction\ Count \times CPI \times Clock\ Period$$

$$CPU\ Time = \frac{Instruction\ Count \times CPI}{Clock\ Rate}$$

Amdahl's Law ★

Amdahl's Law states that the performance improvement to be gained from using some faster mode of execution is limited by the fraction of the time the faster mode can be used.

Amdahl's Law depends on two factors:

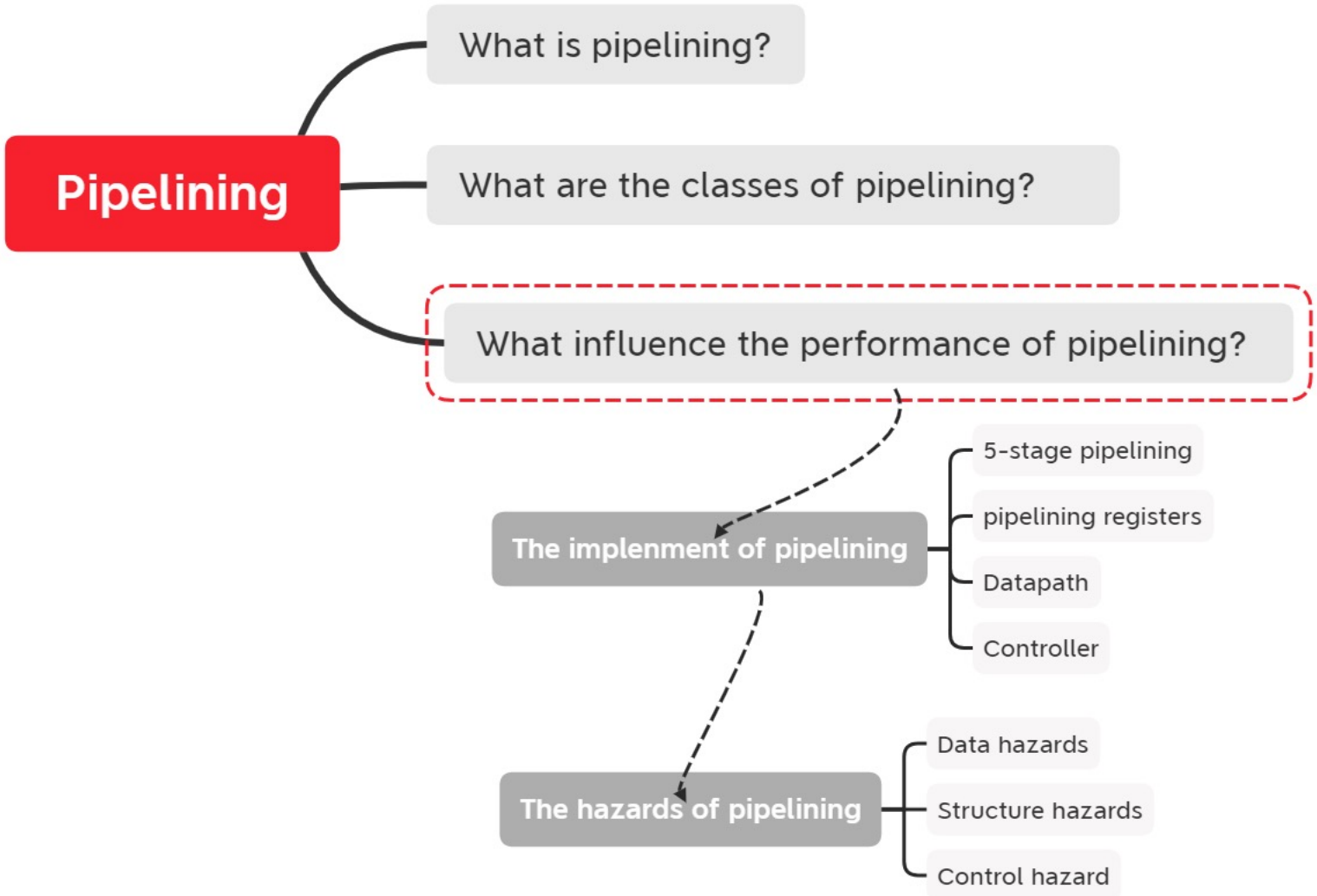
- The fraction of the time the enhancement can be exploited.
- The improvement gained by the enhancement while it is exploited.

$$Improved\ Execution\ Time = \frac{Affected\ Execution\ Time}{Amount\ of\ Improvement} + Unaffected\ Execution\ Time$$

Make the common case fast!

Chapter 2 (6th Appendix C)

Pinelining


Pipelining
What is pipelining?
What are the classes of pipelining?
What influence the performance of pipelining?
The implenment of pipelining
5-stage pipelining
pipelining registers
Datapath
Controller
The hazards of pipelining
Data hazards
Structure hazards
Control hazard

Instruction Set Architecture

ISA

- actual programmer-visible instruction set
- the boundary between software and hardware

ISA: Class

- Most are **general-purpose register architectures** with operands of either registers or memory locations
- Two popular versions

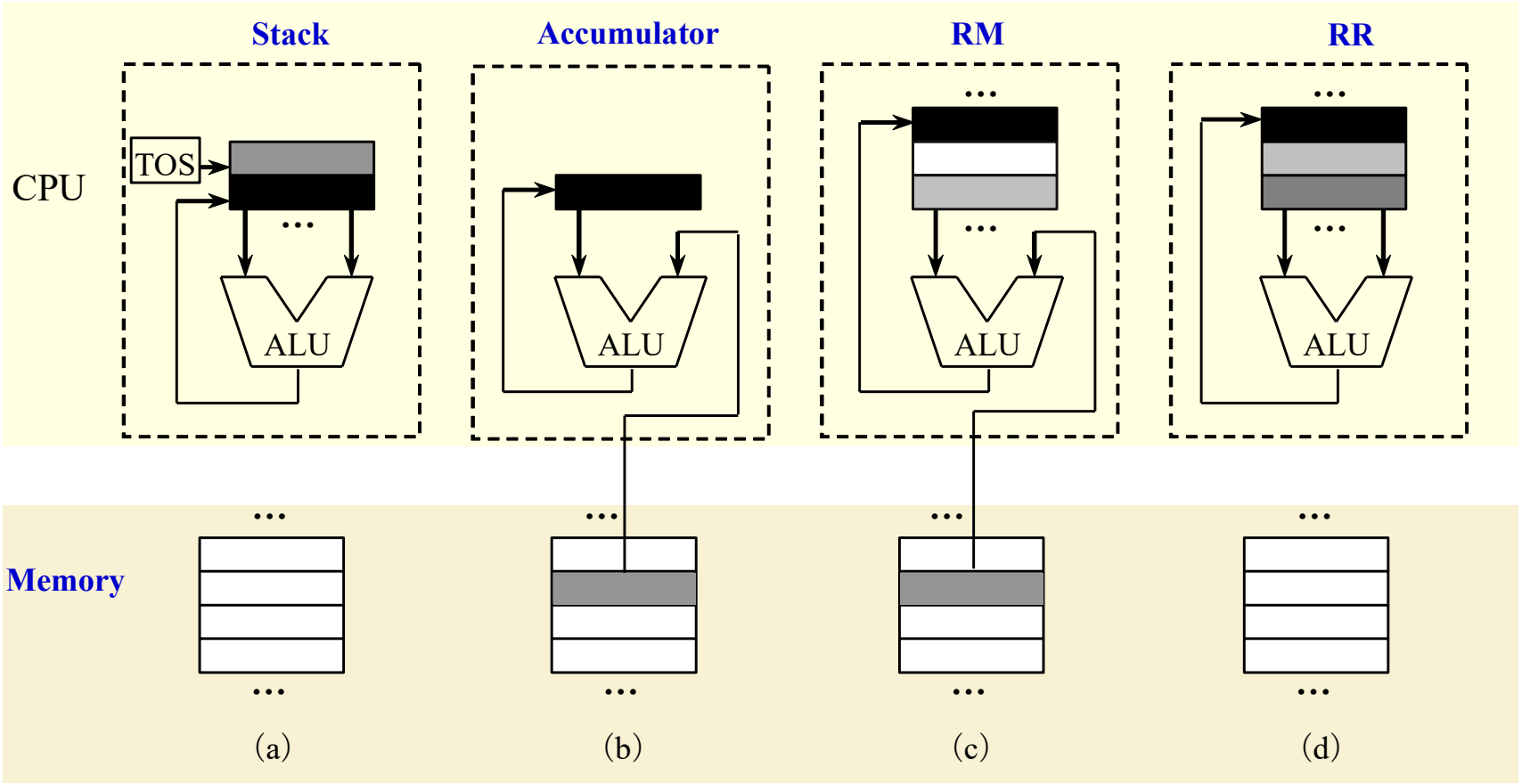
  register-memory ISA: e.g., 80x86

  many instructions can access memory

  load-store ISA: e.g., ARM, MIPS,RISC-V

  only load or store instructions can access memory

GPR Classification



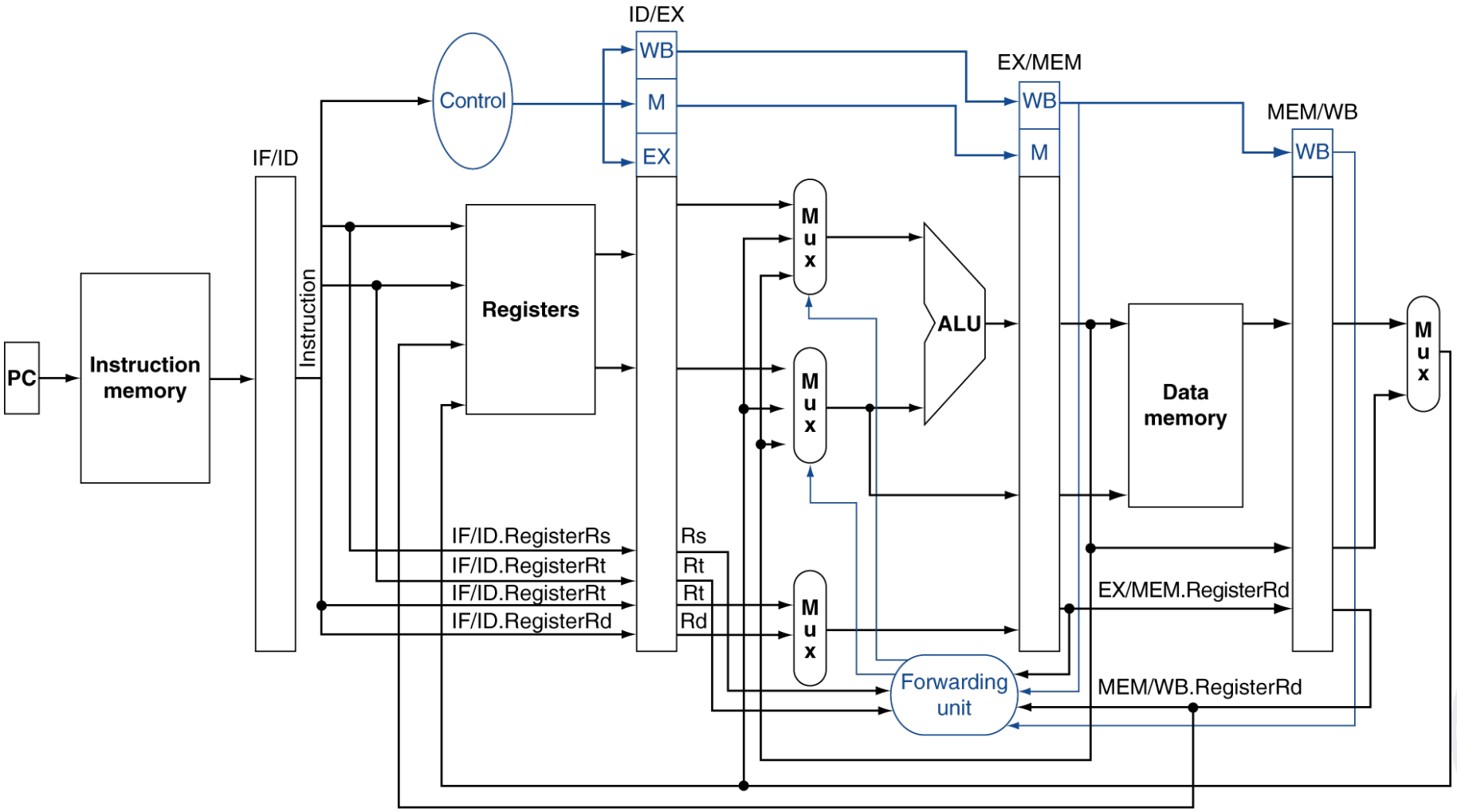
The Four ISA Design Principles

1. Simplicity favors regularity
   - Consistent *instruction size, instruction formats, data formats*
   - Eases implementation by simplifying hardware, leading to higher performance
2. Smaller is faster
   - Fewer bits to access and modify
   - Use the register file instead of slower memory
3. Make the common case fast
   - e.g. Small constants are common, thus small immediate fields should be used.
4. Good design demands good compromises
   - Compromise with special formats for important exceptions
   - e.g. A long jump (beyond a small constant)

Pipeline Summary

- Pipelining improves performance by increasing instruction throughput
  - Executes multiple instructions in parallel
  - Each instruction has the same latency
- Subject to hazards
  - Structure, data, control
- Instruction set design affects complexity of pipeline implementation

Pipelined Datapath ★



The Classic Five-Stage Pipeline for a RISC Processor

- A Simple Implementation of RISC-V
- Instructions Dependences
- Pipeline Hazards

Dependences are a property of *programs.*

- Data Dependences
- Name Dependences
  - Anti-dependence
  - Output-dependence
- Control Dependences

- Data Hazards
  - RAW
  - WAR
  - WAW
- Branch Hazards
- Structural Hazards

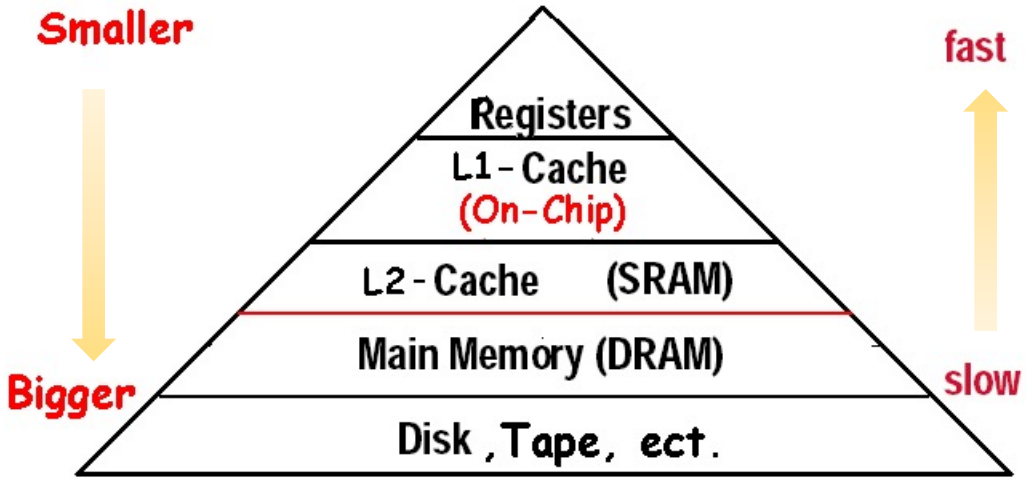
Hazard are properties of the *pipeline organization.*

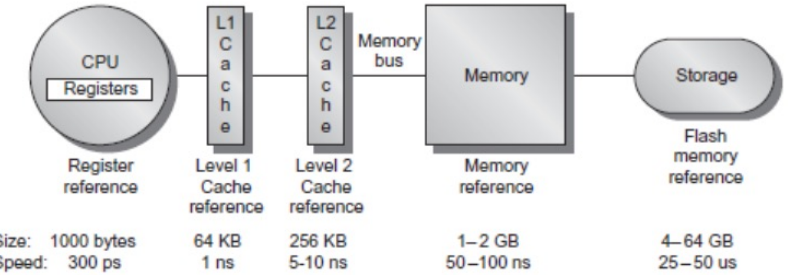
Chapter 3 (6th chapter2 & Appendix B)

Memory Hierarchy

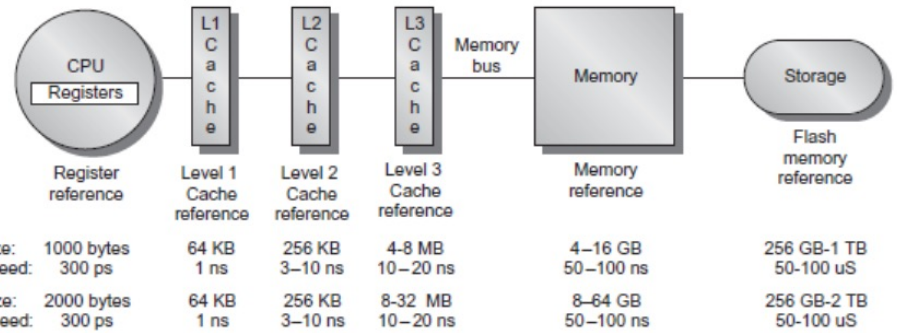
What is a cache?

- Small, fast storage used to improve average access time to slow memory.
- In computer architecture, almost everything is a cache!
  - Registers “a cache” on variables – software managed
  - First-level cache a cache on second-level cache
  - Second-level cache a cache on memory
  - Memory a cache on disk (virtual memory)
  - TLB a cache on page table
  - Branch-prediction a cache on prediction information?

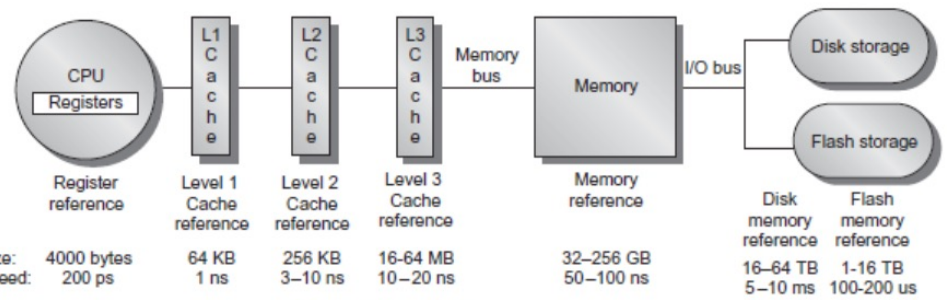




(A) Memory hierarchy for a personal mobile device



(B) Memory hierarchy for a laptop or a desktop



(C) Memory hierarchy for server

Four Questions for Memory Hierarchy Designers

Caching is a general concept used in processors, operating systems, file systems, and applications.

There are **Four Questions** for Memory Hierarchy Designers

- Q1: Where can a block be placed in the upper level/main memory?

(Block placement)

  - Fully Associative, Set Associative, Direct Mapped
- Q2: How is a block found if it is in the upper level/main memory?

(Block identification)

  - Tag/Block
- Q3: Which block should be replaced on a (Virtual Memory) miss?

(Block replacement)

  - Random, LRU,FIFO
- Q4: What happens on a write?

(Write strategy)

  - Write Back or Write Through (with Write Buffer)

Q4: Write Strategy ★

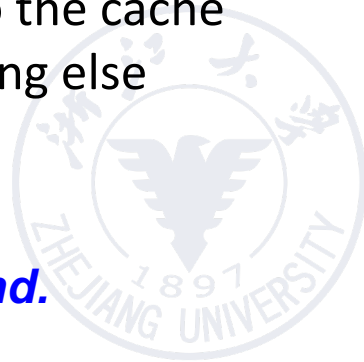
- When data is written into the cache (on a store), is the data also written to main memory?
  - If the data is written to memory, the cache is called a write-through cache
    - Can always discard cached data - most up-to-date data is in memory
    - Cache control bit: only a valid bit
    - memory (or other processors) always have latest data
  - If the data is NOT written to memory, the cache is called a write-back cache
    - Can't just discard cached data - may have to write it back to memory
    - Cache control bits: both valid and dirty bits
    - much lower bandwidth, since data often overwritten multiple times

- Write-through adv: Read misses don't result in writes, memory hierarchy is consistent and it is simple to implement.
- Write back adv: Writes occur at speed of cache and main memory bandwidth is smaller when multiple writes occur to the same block.

Summary

Read Cache	Write Cache ★	
	hit	miss
Read through	***Write-through*** • write is done synchronously both to the cache and to the backing store • the data is written to memory	***No-write allocate (write around)*** • The block is only written to main memory • It is not stored in the cache.
Read allocate	***Write-back*** • writing is done only to the cache • the data is NOT written to memory	***Write allocate*** • The block is loaded into the cache on a miss before anything else occurs.

In general, write-back caches use write-allocate , and write-through caches use write-around.



How to Improve

Hence, there are more than 20 cache optimizations into four categories:

$$AMAT = HitTime + MissRate \times MissPenalty$$

1.Reduce the miss penalty

——multilevel caches, critical word first, read miss before write miss, merging write buffers, and victim caches

2. Reduce the miss rate

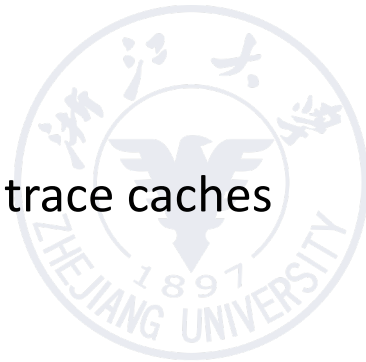
——larger block size, large cache size, higher associativity, way prediction and pseudo-associativity, and compiler optimizations

3. Reduce the miss penalty and miss rate via parallelism

——non-blocking caches, hardware prefetching, and compiler prefetching

4. Reduce the time to hit in the cache

——small and simple caches, avoiding address translation, pipelined cache access, and trace caches



Average Memory Access Time ★

- Average Memory Access Time

$$\text{Average Memory Access Time} = \frac{\text{Whole accesses time}}{\text{All memory accesses in program}}$$

$$= \frac{\text{Accesses time on hitting+ Accesses time on miss}}{\text{All memory accesses in program}}$$

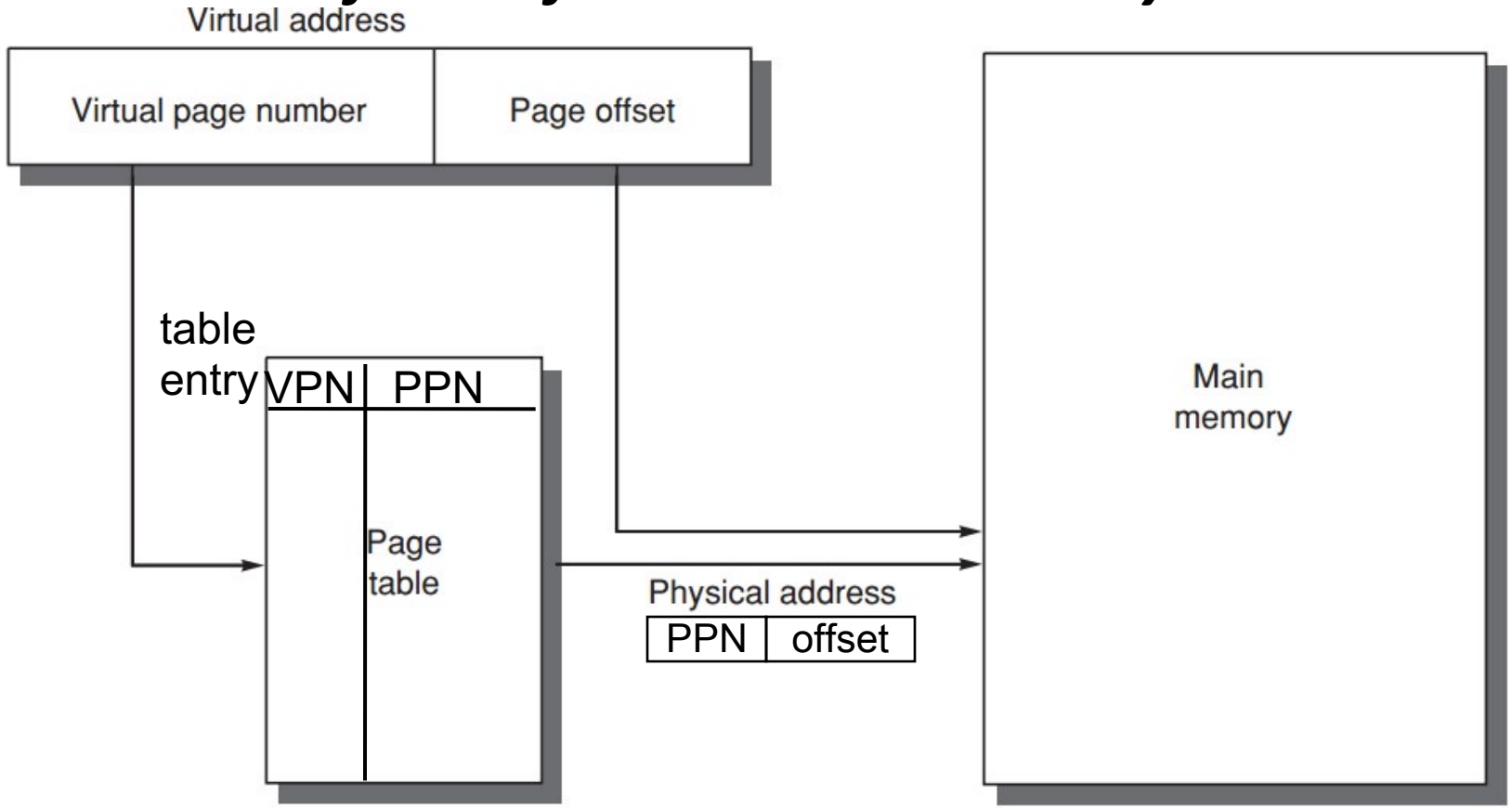
$$= \text{Hit time} + (\text{Miss Rate} \times \text{Miss Penalty})$$

$$= \left(HitTime_{Inst} + MissRate_{Inst} \times MissPenalty_{Inst}\right) \times Inst\%$$
$$\left(HitTime_{Data} + MissRate_{Data} \times MissPenalty_{Data}\right) \times Data\%$$

$$CPUtime = IC \times \left(\frac{AluOps}{Inst} \times CPI_{AluOps} + \frac{MemAccess}{Inst} \times AMAT \right) \times CycleTime$$

Four Mem Hierarchy Q's

- **Q1.** ***Where can a block be placed in main memory?***
- **Q2.** ***How is a block found if it is in main memory?***



What is TLB

- **Translation lookaside buffer (TLB)**

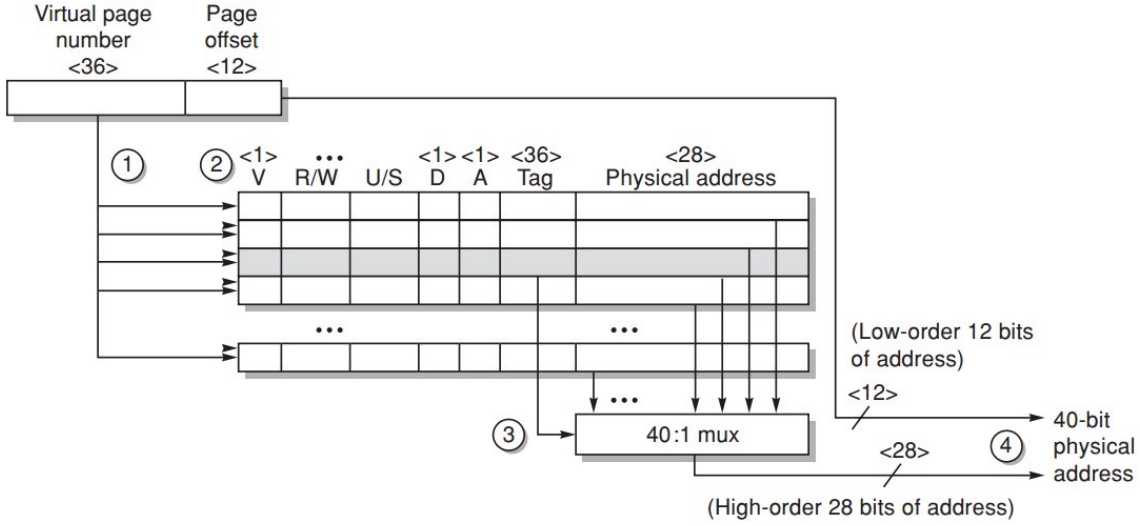
  a special **cache!**

  that keeps (prev) address translations

- TLB entry

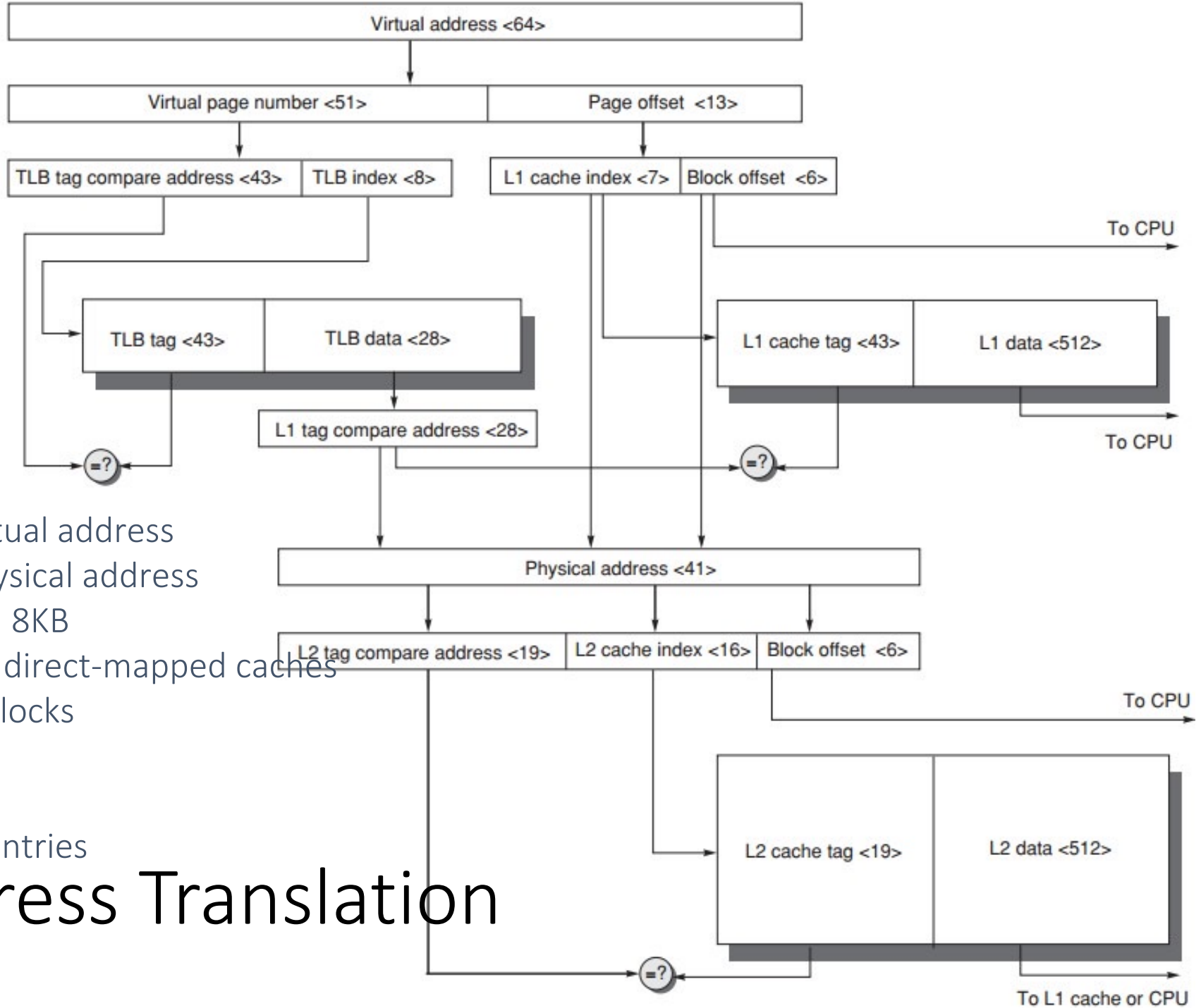
  --tag: portions of the virtual address;

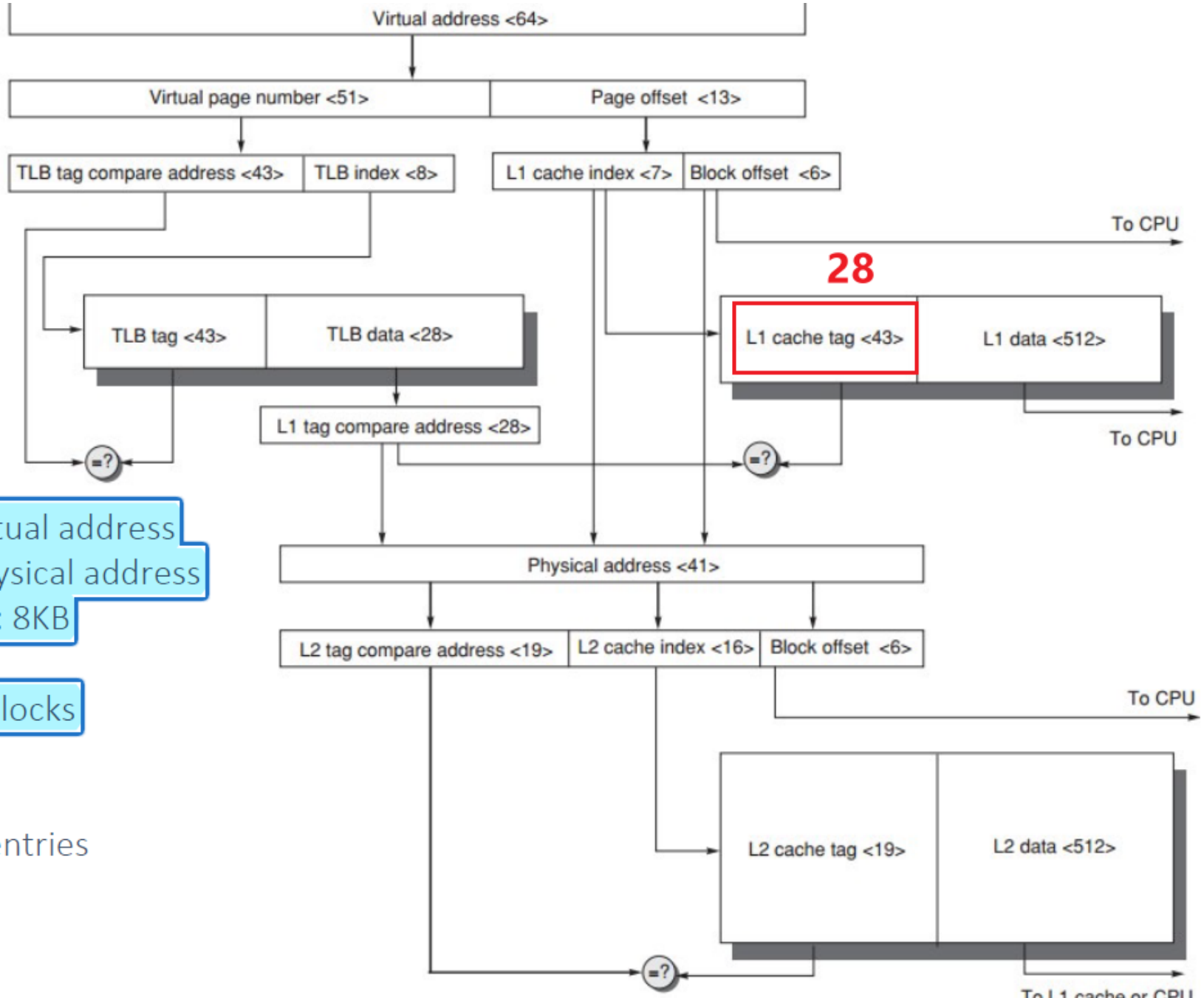
  --data: a physical page frame number, protection field, valid bit, use bit, dirty bit;



Address Translation

64-bit virtual address
41-bit physical address
page size: 8KB
two-level direct-mapped caches
64-byte blocks
L1: 8KB
L2: 4MB
TLB 256 entries





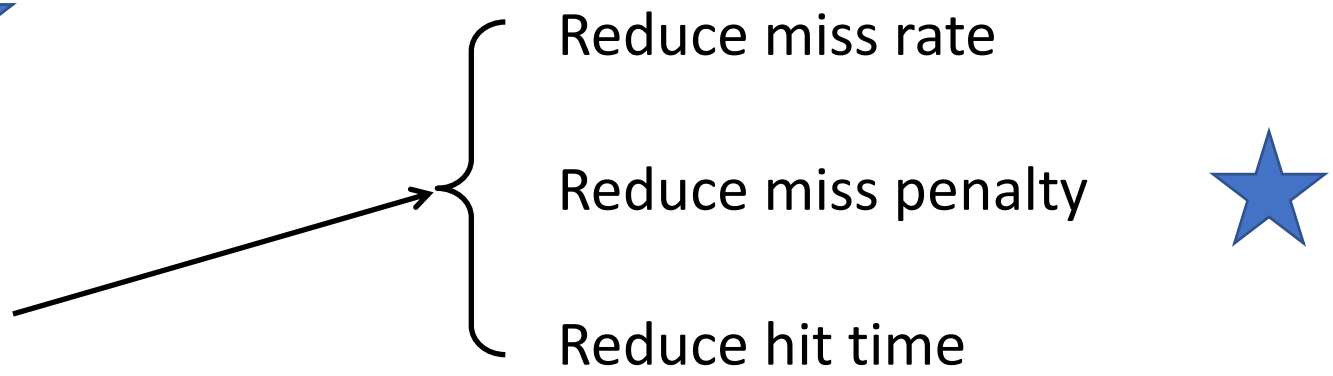
64-bit virtual address
41-bit physical address
page size: 8KB

64-byte blocks
L1: 8KB
L2: 4MB

TLB 256 entries

Summary

- Memory hierarchy
  - From single level to multi level
  - Evaluate the performance parameters of the storage system (average price per bit C; hit rate H; average memory access time T)
- Cache basic knowledge
  - Mapping rules
  - Access method
  - Replacement algorithm
  - Write strategy
  - Cache performance analysis



- Memory (the influence of memory organization structure on Cache failure rate)

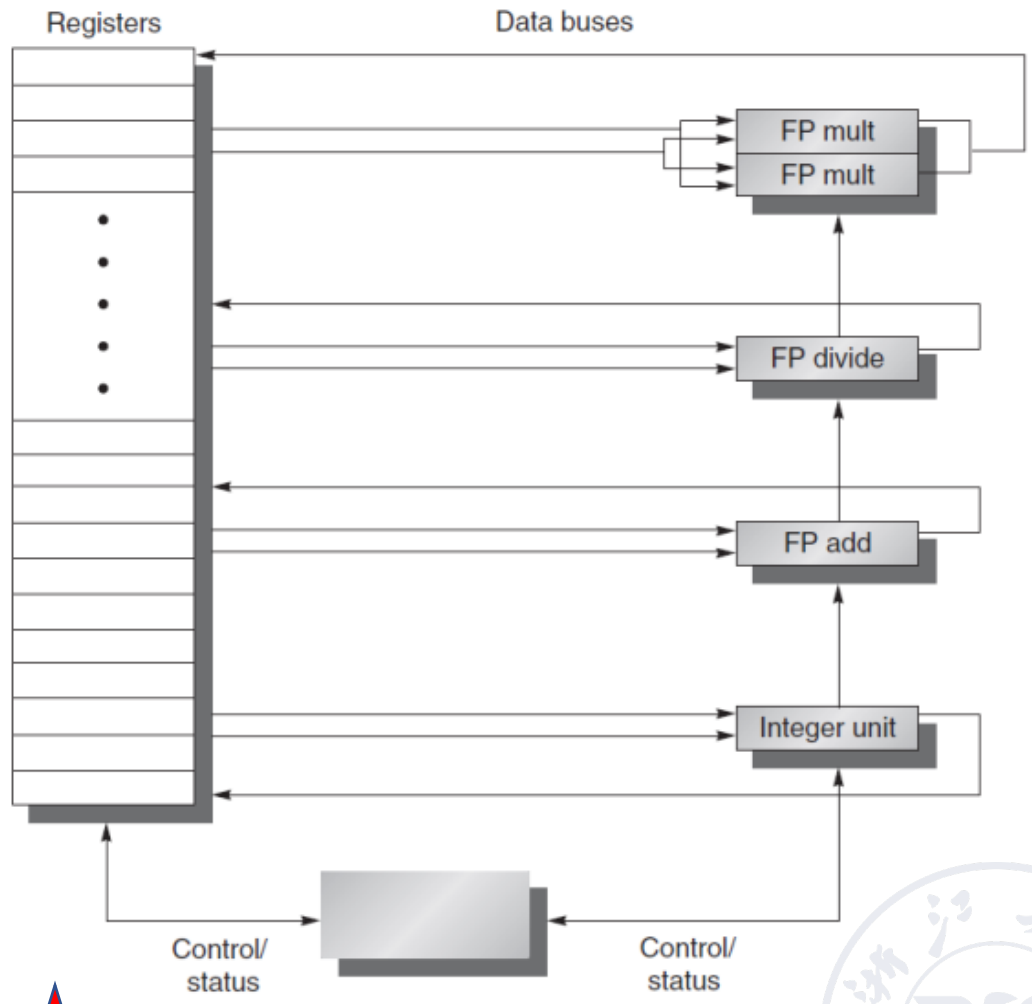
Chapter 4(chapter3)

ILP

Dynamic Scheduling

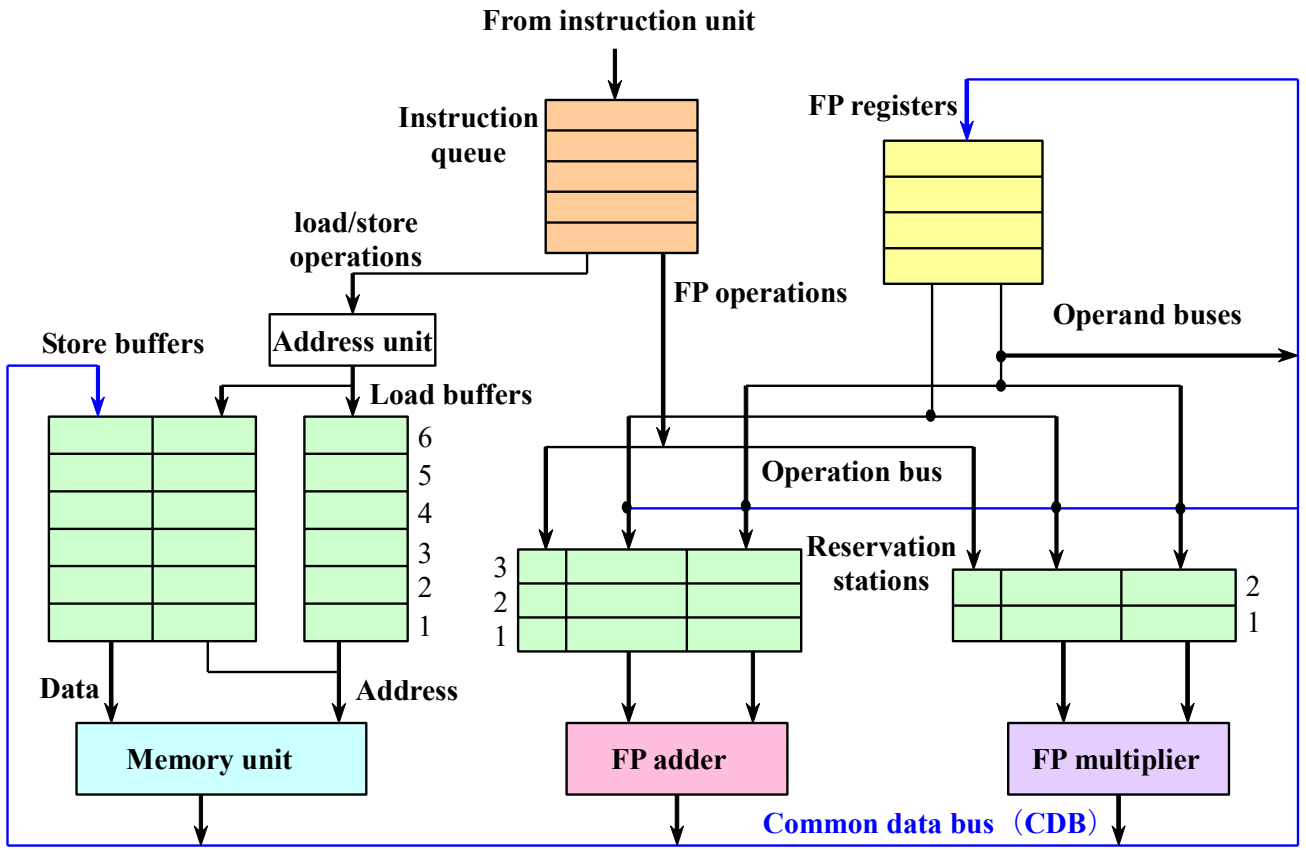
Idea: Dynamic Scheduling

Method: out-of-order execution



Dynamic Scheduling with a Scoreboard

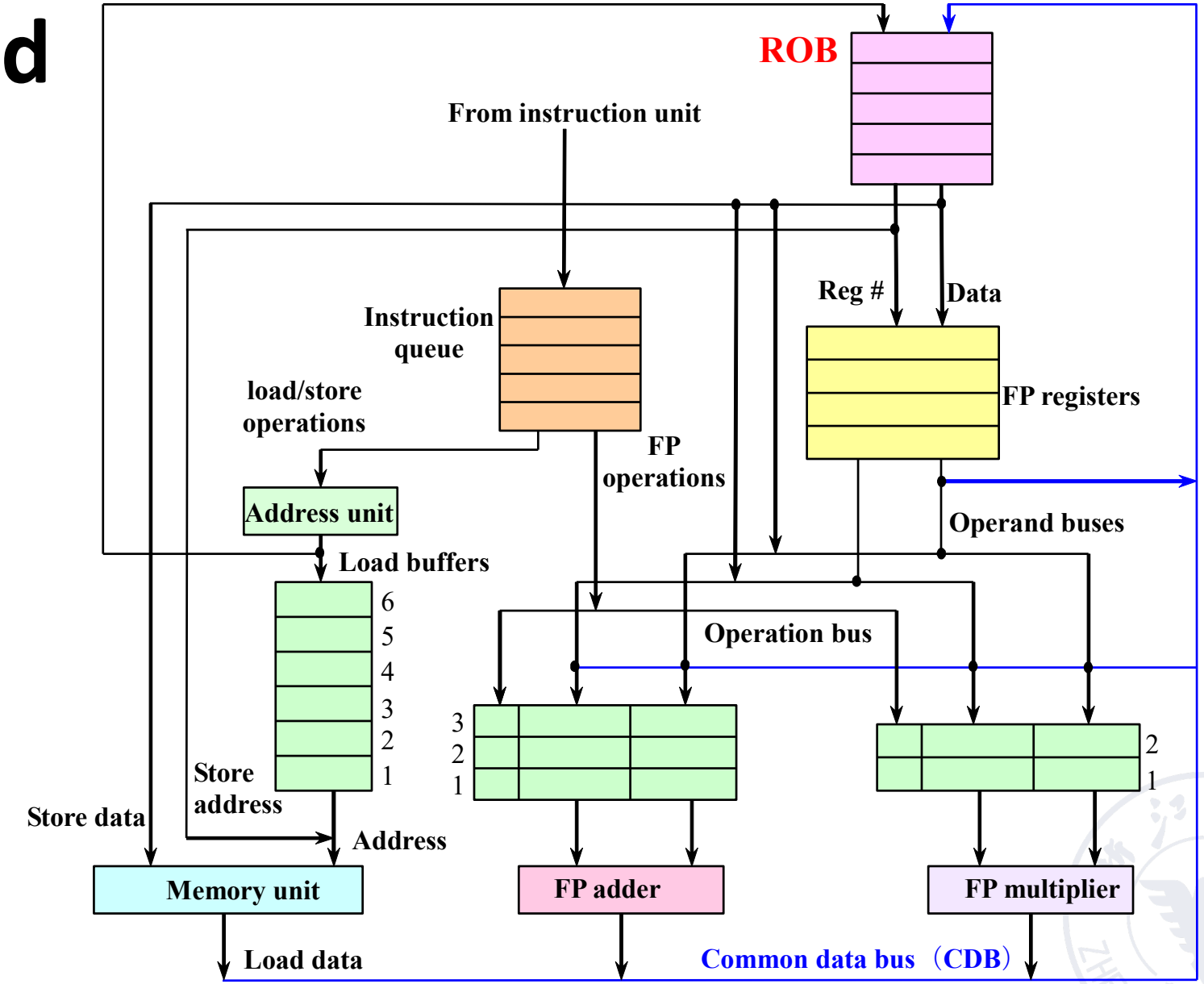
Tomasulo's Approach



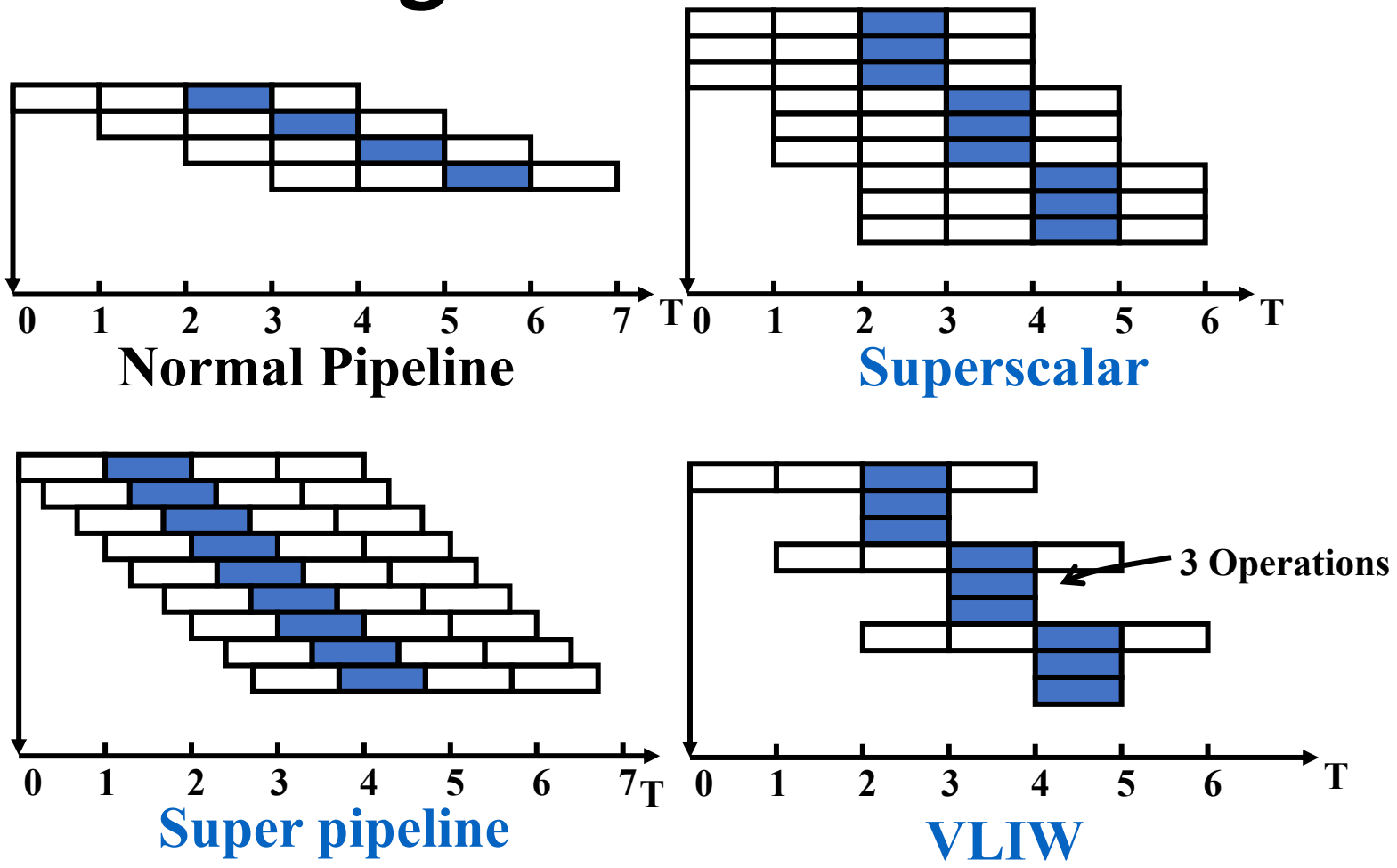
The basic structure of a floating-point unit using Tomasulo's algorithm

Hardware-Based Speculation ★

The basic structure of a FP unit using Tomasulo's algorithm and extended to handle speculation.



Exploiting ILP Using Multiple Issue and Static Scheduling



Chapter 5(chapter5.1-5.4)

DLP and TLP

Classes of Parallel Architectures

according to the parallelism
in the instruction and data streams
called for by the instructions:

SISD, SIMD, MISD, MIMD

SISD

- Single instruction stream
  single data stream
- uniprocessor
- Can exploit instruction-level parallelism

SIMD

- Single instruction stream
  multiple data stream
- The **same instruction** is executed by **multiple processors** using
  different data streams.
- Exploits data-level parallelism
- Data memory for each processor;
  whereas a single instruction memory and control processor.

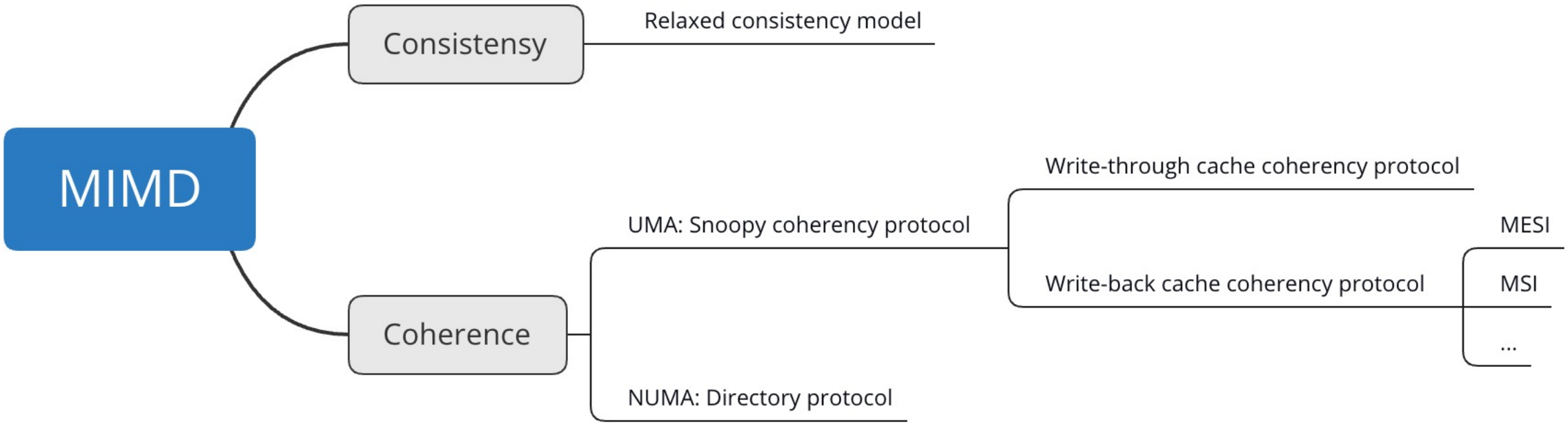
MISD

- Multiple instruction streams
  single data stream
- No commercial multiprocessor of this type yet

MIMD

- Multiple instruction streams

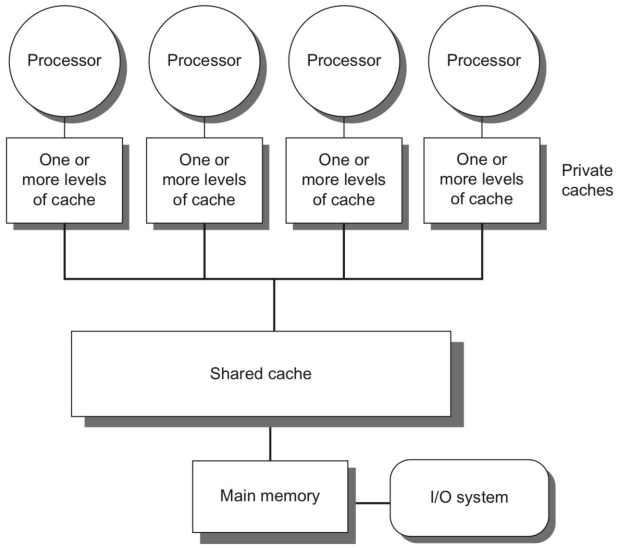
  multiple data streams
- Each processor fetches its own instructions and operates on its own data.
- Exploits task-level parallelism


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

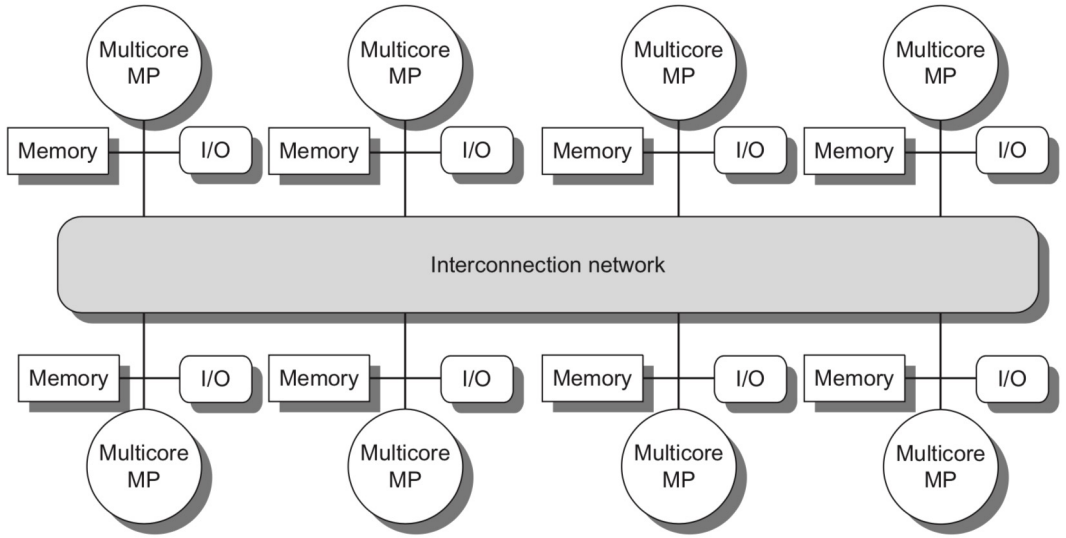
UMA and NUMA

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

SMP



DSP



Cache consistency and Cache coherence

- Cache coherence defines *the behavior of reads and writes to the same memory location.*

- Cache consistency defines *the behavior of reads and writes with respect to accesses to other memory locations.*

Cache Coherence Protocols

- **Directory based （ for DSP）**

  the sharing status of a particular block of physical memory is kept in one location, called directory

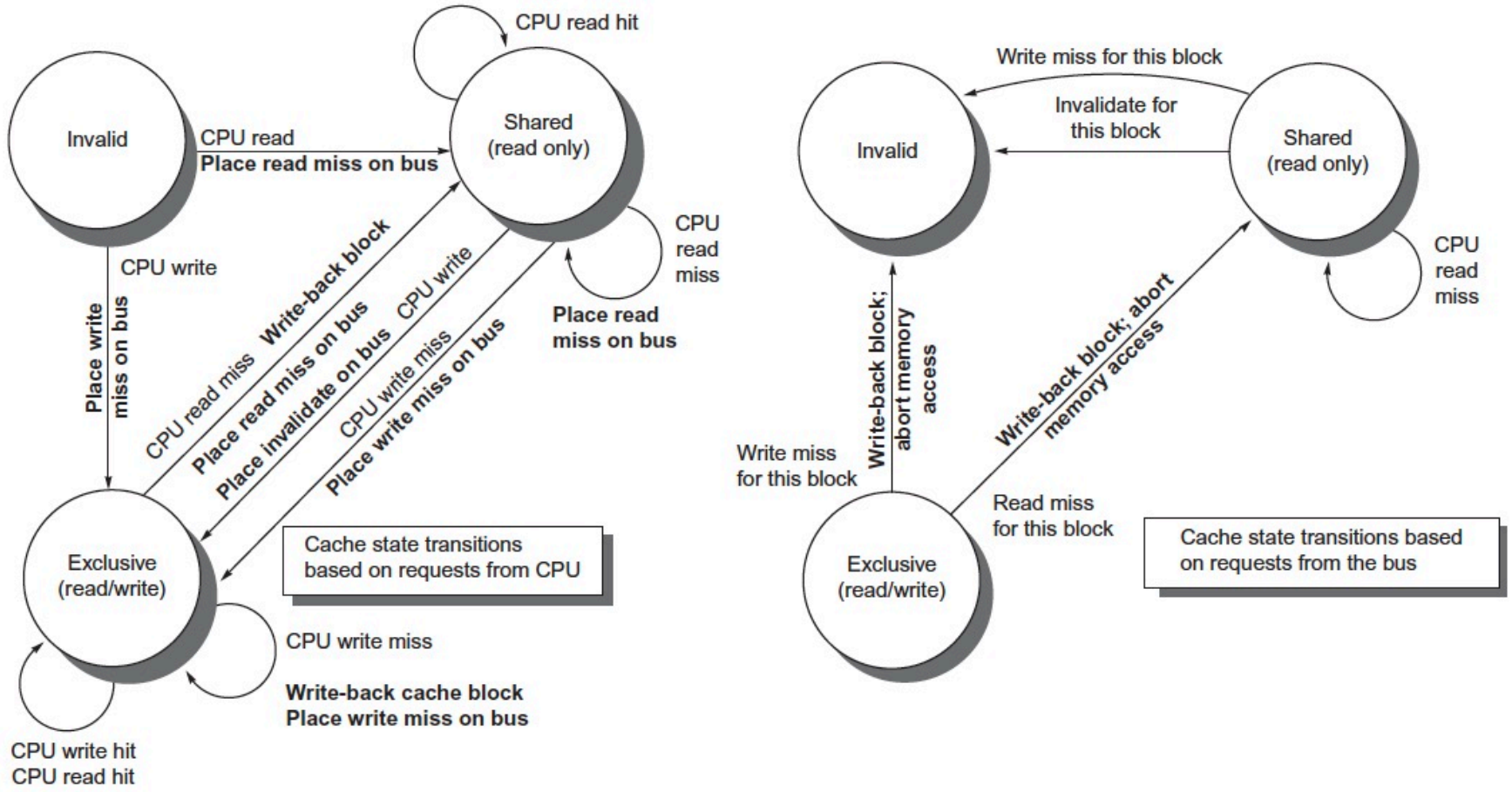
- **Snooping （ for SMP)**

  every cache that has a copy of the data from a block of physical memory could track the sharing status of the block

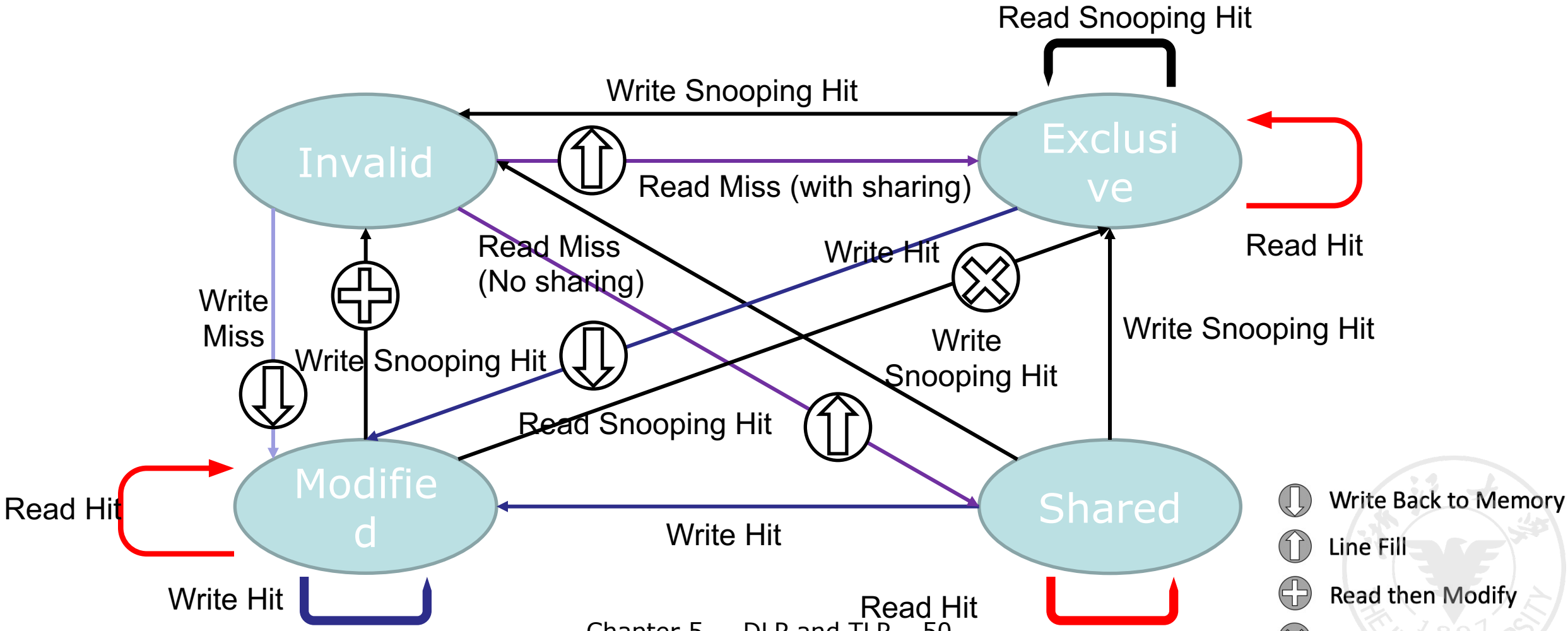
Snooping Cache Coherence Protocol

MSI protocol
- ✓ **Invalid**
- ✓ **Shared**
- ✓ **Modified**

implies that the block is **exclusive**



MESI protocol state transition rules



Directory-Based Cache Coherence Protocol

✓ **Shared**

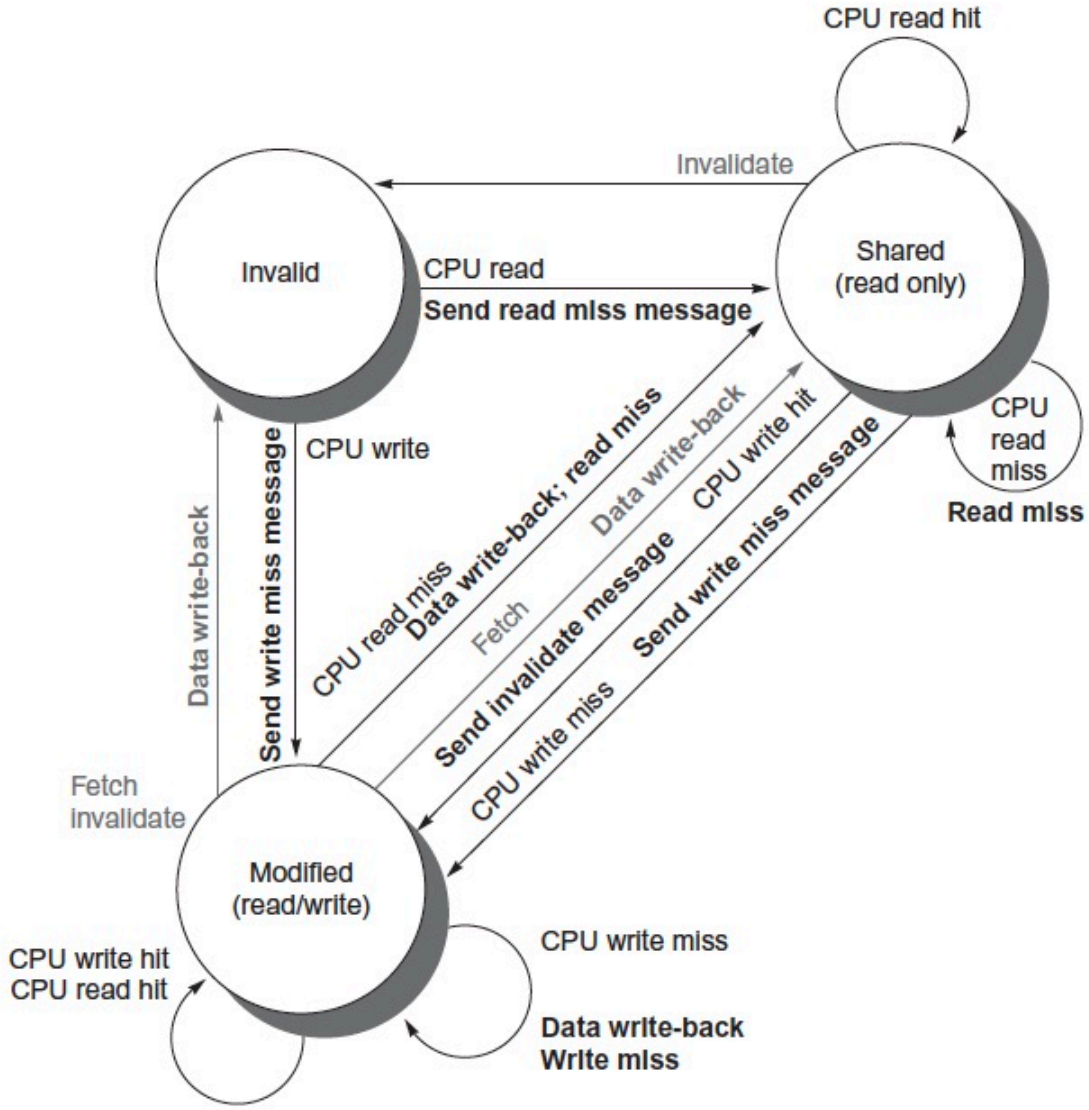
One or more nodes have the block cached, and the value in memory is up to date (as well as in all the caches).

✓ **Uncached**

No node has a copy of the cache block.

✓ **Modified**

Exactly one node has a copy of the cache block, and it has written the block, so the memory copy is out of date. The processor is called the owner of the block.



Directory-Based Cache Coherence Protocol

✓ **Shared**

One or more nodes have the block cached, and the value in memory is up to date (as well as in all the caches).

✓ **Uncached**

No node has a copy of the cache block.

✓ **Modified**

Exactly one node has a copy of the cache block, and it has written the block, so the memory copy is out of date. The processor is called the owner of the block.