CO 05 Memory Hierarchy

1 Memory Intro

- SRAM
  - volatile
- DRAM
  - volative
- Flash
  - nonvolatile
- Magnetic Disk
  - noevolatile
  - cylinder, plate, track, sector, head
  - plate 是双面的

磁盘计算 >

Disk Access Example

- **Given**
  - 512B sector, 15,000rpm, 4ms average seek time, 100MB/s transfer rate, 0.2ms controller overhead, idle disk
- **Average read time**
  - 4ms seek time
    + ½ / (15,000/60) = 2ms rotational latency
    + 512 / 100MB/s = 0.005ms transfer time
    + 0.2ms controller delay
    = 6.2ms
- **If actual average seek time is 1ms**
  - Average read time = 3.2ms

磁盘的主要瓶颈在于 rotation time

2 Memory Hierarchy Intro

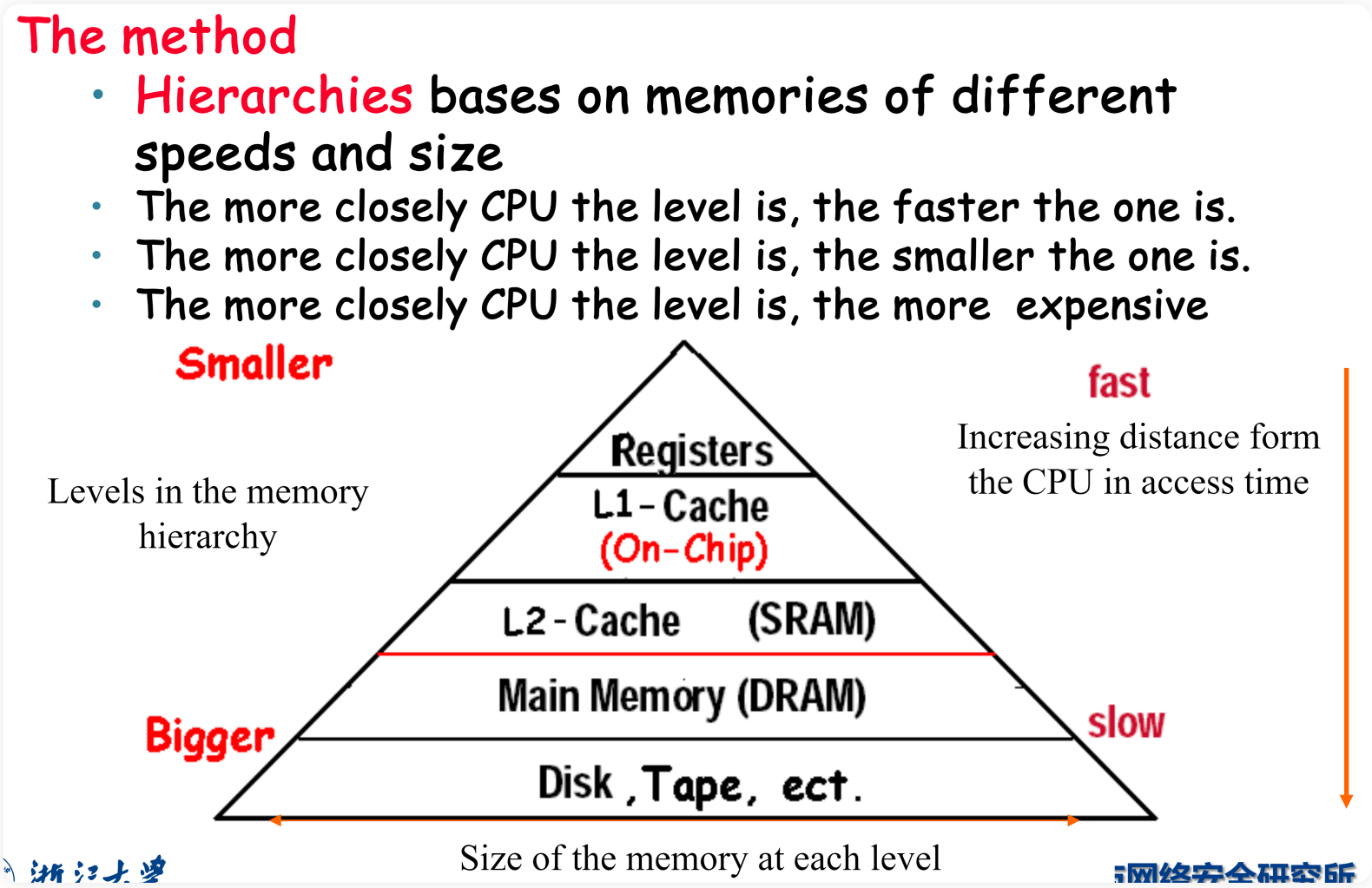
- **Temporal locality** 时间局部性
  - 近期访问的数据会在短时间内再次访问
  - e.g. 循环中的指令、计数器变量
- **Spatial locality** 空间局部性
  - 近期访问的数据旁边的数据会马上被访问
  - e.g. 数组遍历、sequential inst access

Taking Advantage of Locality

- 预测并从磁盘取到 DRAM
- 预测并从 DRAM 取到 SRAM
- *Large and fast*

2.1 Terms

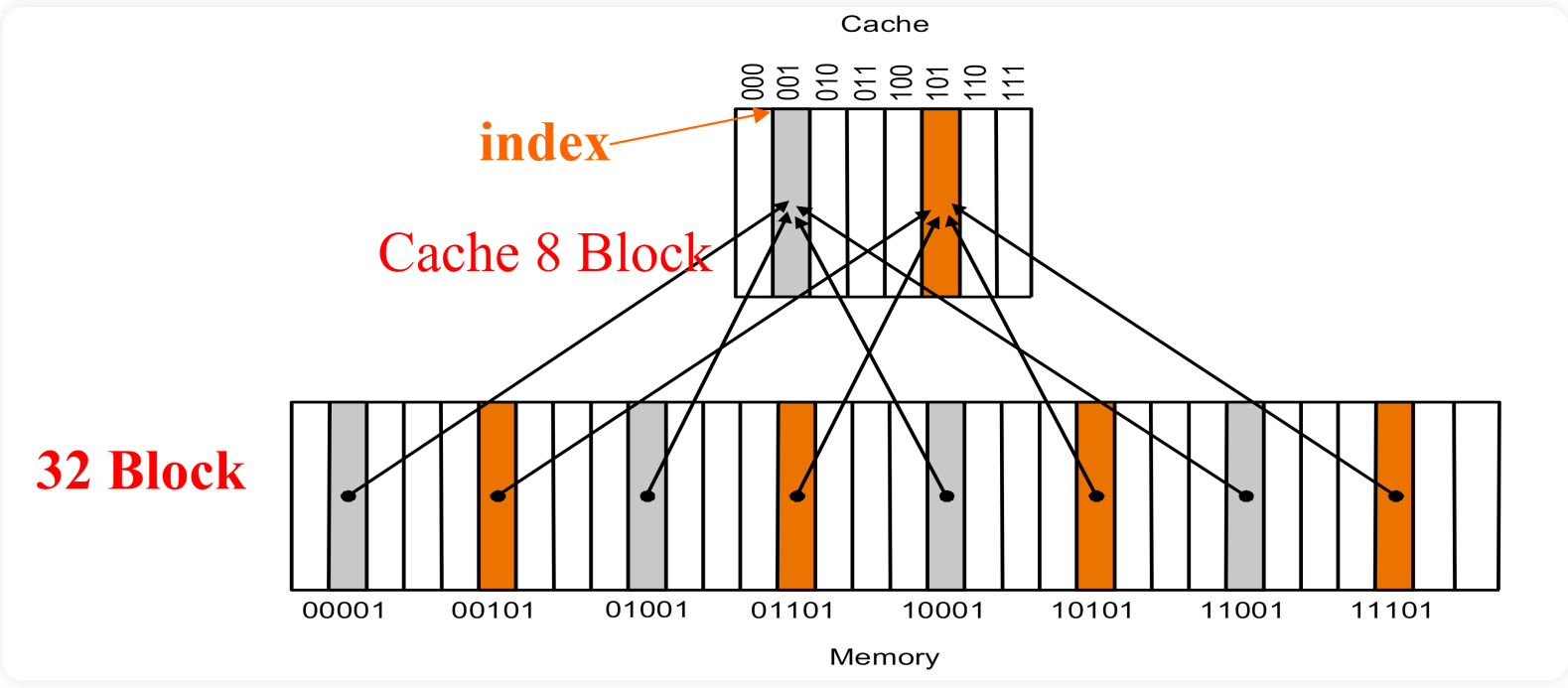
- Block(line): 在不同内存之间搬运的最小单元，一定是 word 的 2^N，因为需要占用总线
- Hit: access the upper level and succeeds
- Miss: access the upper level and fails
- Hit Time: 访问命中的延时，包括了判断是否 hit 的时间
- Miss Penalty: miss 后，从 lower level 搬运数据到 upper level，以及到 CPU 的时间



> **L1 Cache 和 L2 Cache 都是 SRAM，但是延迟有区别**
>
> - L1 Cache 主要目的是减少 hit time，容量较小，寻址更快
> - L2 Chche 主要目的是增加 hit rate (避免读内存导致 CPU stall)，容量较大，寻址更慢

3 The basics of Cache

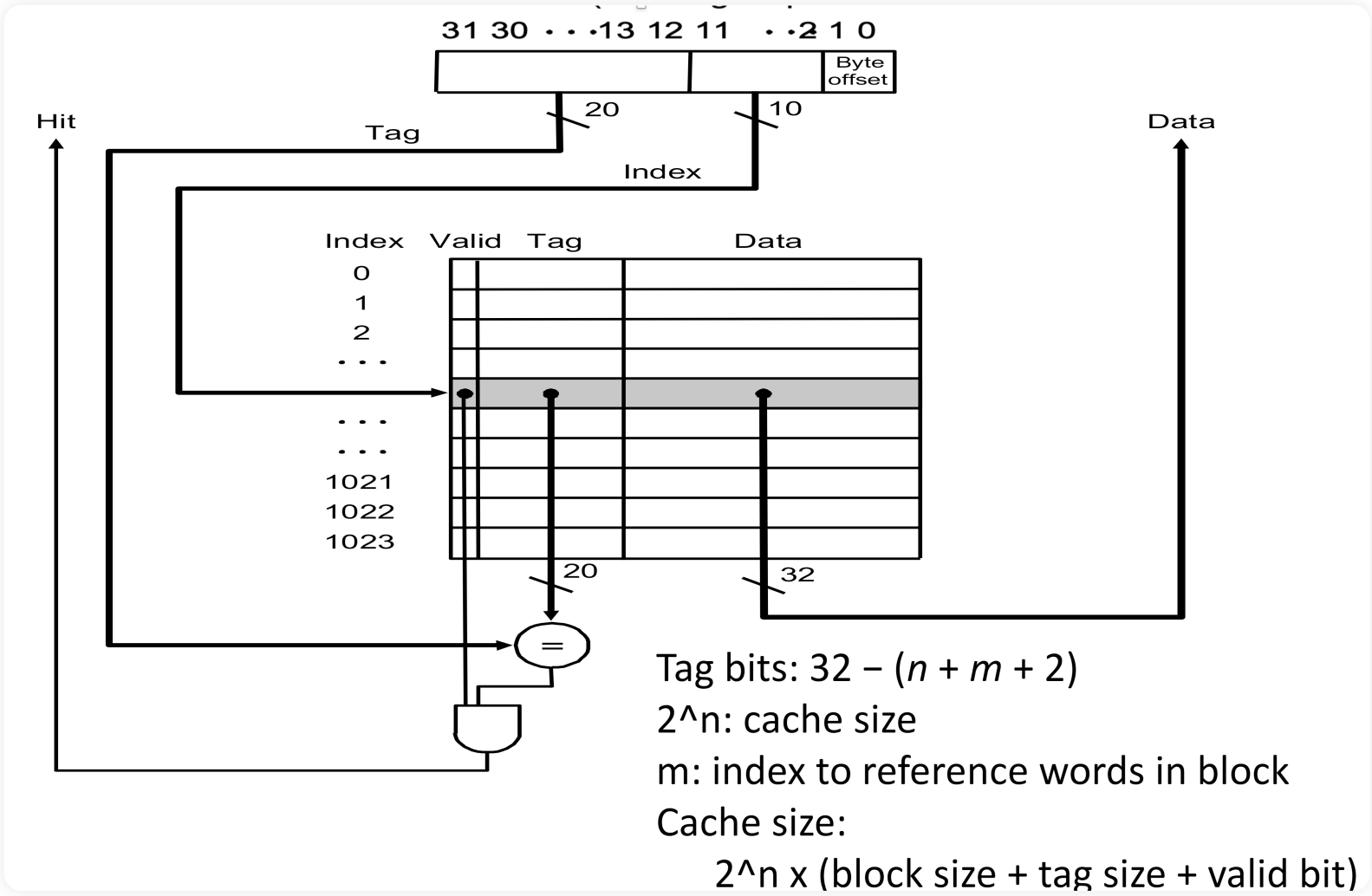
3.1 Direct Mapped Cache

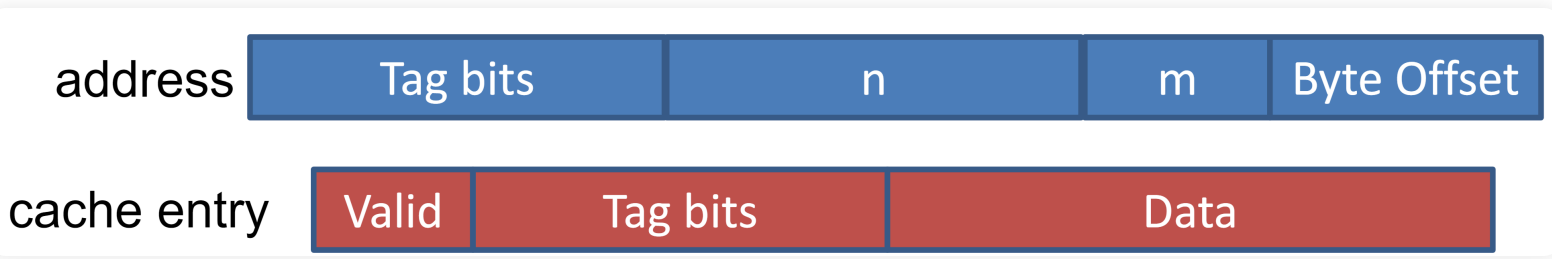


- 在 cache 中的地址是 (block address) mod (#blocks in cache)，*实际上就是取最低的几位*
- $\text{mem addr} = \text{block addr} \times \text{block size} + \text{block offset}$
- tag: block addr 除掉 cache index 的高几位
  - $\#\text{tag bits} = 32-$
- valid bits: 标记 cache block 是否有效，避免未初始化访问
- pros and cons

- pro: 寻址快
- con: 可能造成竞争，*e.g. 对同在一个 cache index 的两个数据的连续修改，每一次都会 miss*

3.1.1 相关计算





- address: | tag bits | cache block index | cache block word offset | word byte offset (2 bits) |
- cache entry: | valid bit | tag bits | block data |

Example >

Example

- How many total bits are required for a direct-mapped cache 16KB of data and 4-word blocks, assuming a 32-bit address?

Answer

16KB=4KWord=2^{12} words
One block=4 words = 2^{2} words
Number of blocks (index bit) = $2^{12} \div 2^{2} = 2^{10}$ blocks
Data bits of block =4×32=128 bits
Tag bits = address – index-block size =32-10–2-2 =18 bits
Valid bit = 1 bit

Total Cache size = $2^{10} \times (128+18+1) = 2^{10} \times 147 = 147$ Kbits
= 18.4KB
It is about 1.15 times as many as needed just for the data

Example

Consider a cache with 64 blocks and a block size of 16 bytes.
What block number does byte address 1200 map to?

(Block address) modulo (Number of cache blocks)

Answer

Where the address of the block is

First: get BLOCK Address

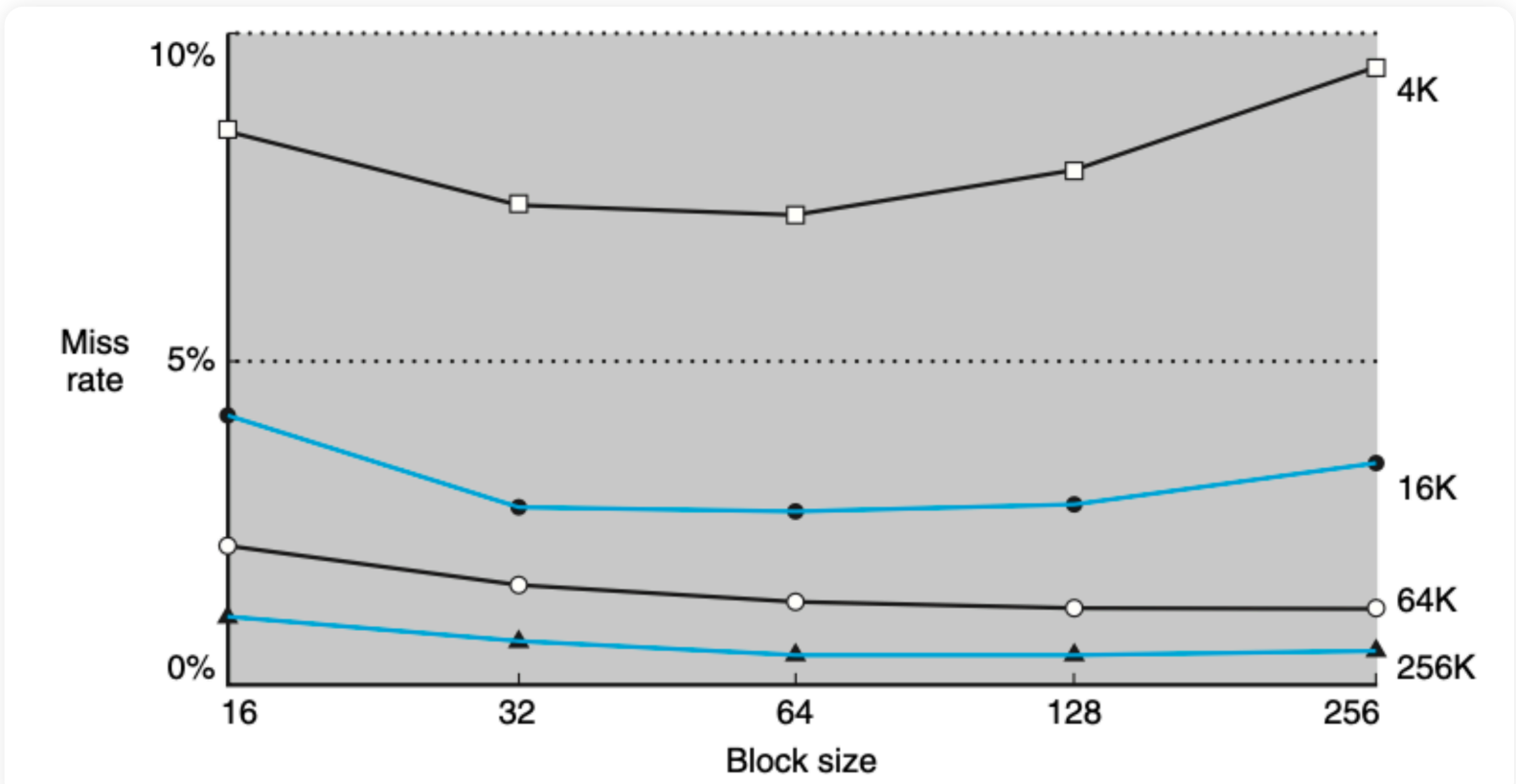
$$\left\lfloor \frac{\text{Byte address}}{\text{Bytes per block}} \right\rfloor = \left\lfloor \frac{1200}{16} \right\rfloor = 75$$

Notice!!! **75 modulo 64 =11** Then: get INDEX

$$\left\lfloor \frac{\text{Byte address}}{\text{Bytes per block}} \right\rfloor \times \text{Byte per block} \longleftrightarrow \left\lfloor \frac{\text{Byte address}}{\text{Bytes per block}} \right\rfloor \times \text{Byte per block} + (\text{Byte per block} - 1)$$

Here: 1200 ⟷ 1215

3.2 Miss Rate vs Block Size



- 总容量越大，miss rate 越低
- 容量一定
  - 若 size 太小，无法利用空间局部性，e.g. 一个数组遍历，要分成更多段来读取到 cache
  - 若 size 太大，无法兼顾分散访问

> ⚠ **Attention**
>
> - 设计的目标不只是降低 miss rate，而是关注总的 penalty
> - 如果 block 较大，可能会 miss rate 比较低，但是读取搬运的开销更大

3.3 Hit and Misses

3.3.1 Read miss

- miss 的 penalty 近似
  - inst cache miss
  - data cache miss
- 处理 miss
  1. 保留原始的 PC
  2. main memory read (in multiple cycles)
  3. write to cache, update tag/valid

4. 从原始的 PC 重新执行，这次一定会 hit

3.3.2 Write

- write hits: different strategies
  - write-back: 只更新 cache，不更新低级存储
    - pros: 速度更快
    - cons: inconsistent，数据不一致
  - write-through: 写穿透，同步更新低级存储
    - cons: 慢
- write miss: 读取到缓存，再进行 write hit 操作

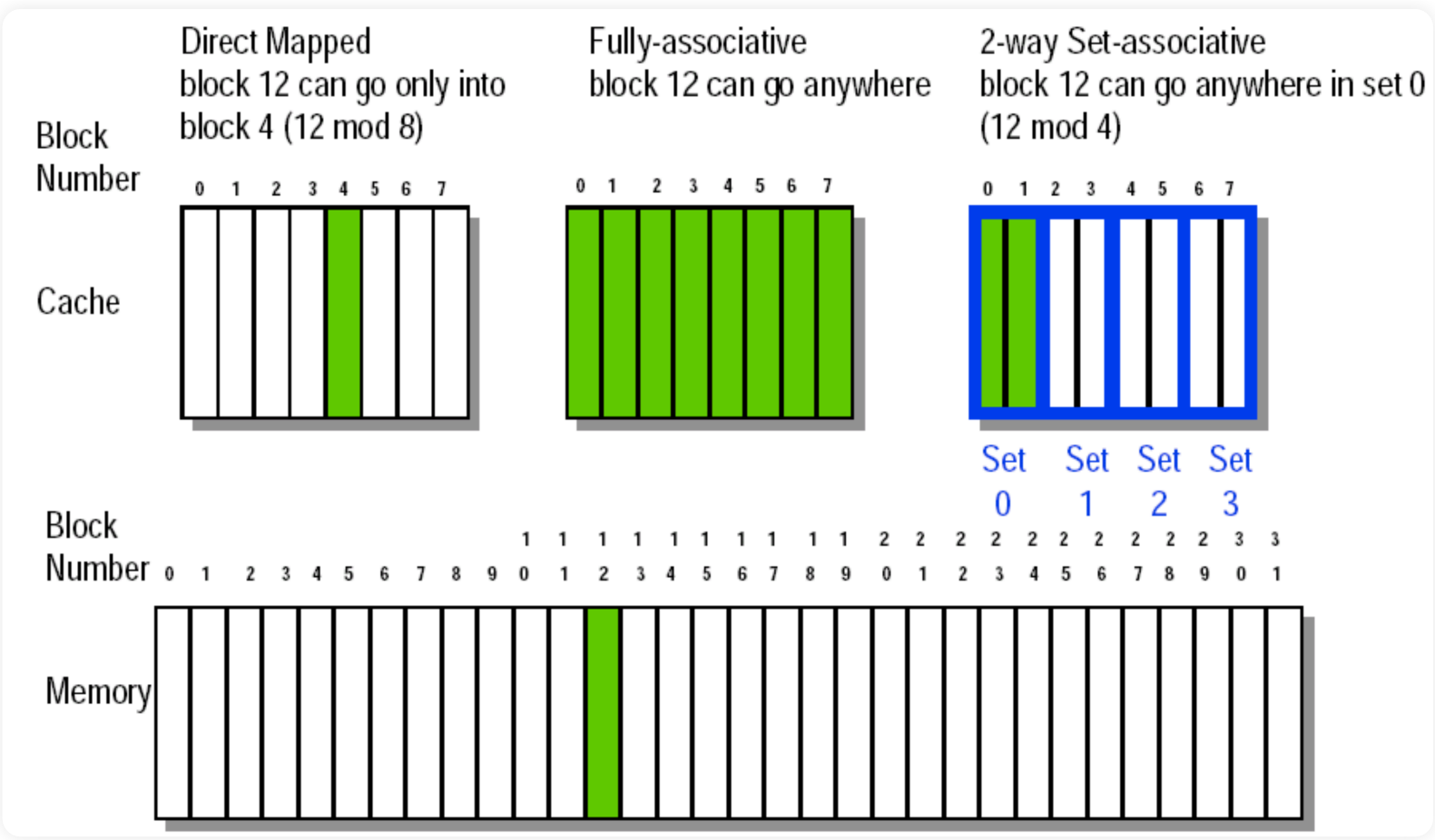
> **为什么一定要将整个 block 读取到 cache 再写入?**
>
> 因为与 main memory 的通信一定是以 block 为单位的，cpu 只能在 cache 中执行 word 级别操作，所以一定要保证整个 block 都在 cache 中，否则 main memory 可能会被覆盖

4 Set associative

> Reducing cache misses by more flexible placement of blocks

> **Intro: Fully associative**
>
> - direct mapped: 可能导致竞争，利用率不高
> - fully associative: block can go anywhere in cache
>   - cons: 查找非常慢

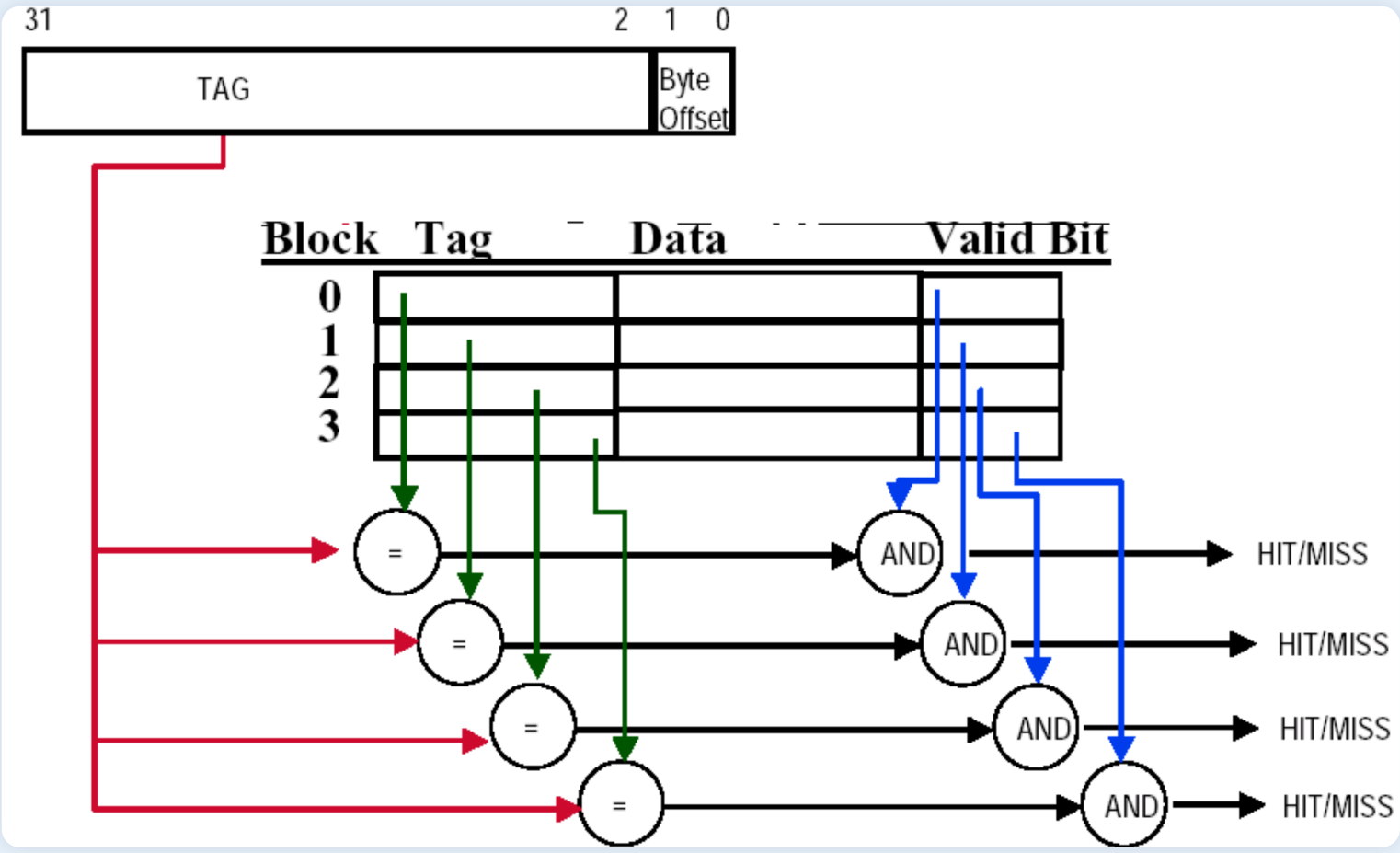
- block can go anywhere within its set
- n-way set associative 指的是一个 set 的 block 容量为 n
  - direct mapped = 1-way set associative
  - fully associative = n-way set



4.1 Block Identification

- | Tag | Index | Offset |
  - offset: block 内部的 byte offset
  - index: select sets
  - tag: 剩余的物理地址
  - tag + index = **block address**

fully-associative



Note

- associative identification 可以用电路并行实现，但是电路规模太大也不行

4.2 Block Replacement

- random
- LRU(Least-recently used)
  - 使用表或增加 cache block unit 宽度来记录
- FIFO(First in, first out)

4.2.1 LRU

- 使用 1-bit，每 T 个周期都清零，能够表示至少 T 个周期内没有用过
- % 事实上，会使用类似二叉树的方法，每个 internal node 会保存一个 bit，表示最新访问路径，**并不总是能找到最优解**
  - e.g. leaf[0:3]，按照 12304 的顺序访问，访问 4 的时候会替换 2

5 Write Strategy

> also written to main memory?

- write-through cache
  - 总是写入到 main memory
  - 可以随意丢弃 block
- write-back cache
  - 需要额外的 dirty bit，标记 block 是否有修改；如果有，覆盖前需要写回
  - $ lower bandwidth: cache block 可能会多次访问、修改 *since data often overwritten multiple times*

5.1 Write Stall

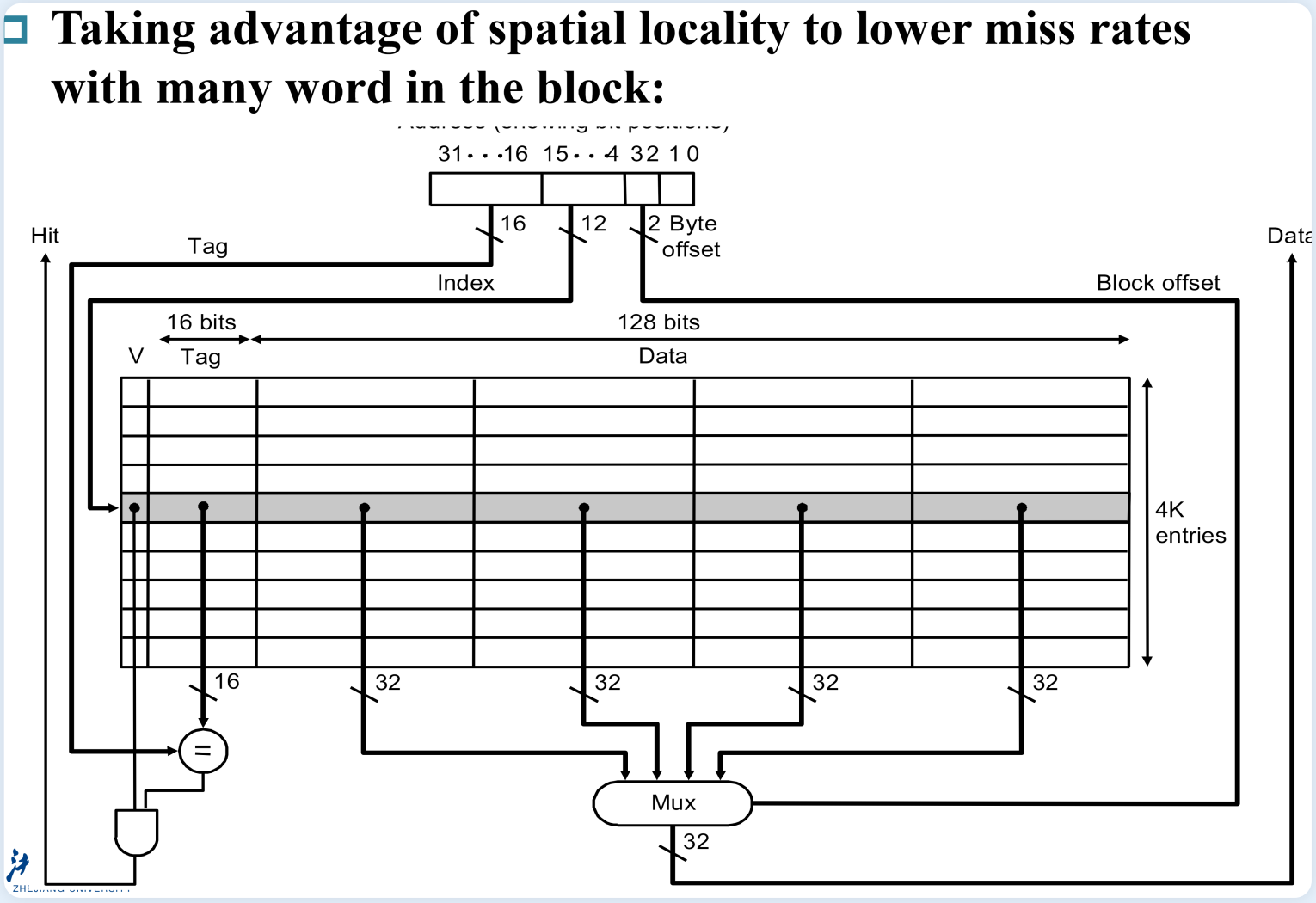
- write stall: 执行 **write through** 时 CPU 等待时间
- Write buffers: 写入 main memory 的数据缓存区
  - write-through 开销能够减小
  - ! buffer 有极小可能被写满

5.2 Write Misses

- Write allocate: 先把 block 加载到 cache

- ! 如果只需要修改 block 中的部分内容，但内存只能以 block 为单位读写，所以需要 write allocate
- Write around: 直接写入 main memory
  - e.g. 变量在 init 的时候置零，且不立即使用，那么采用两种策略的开销是一样的，因为都加载到 cache 一次

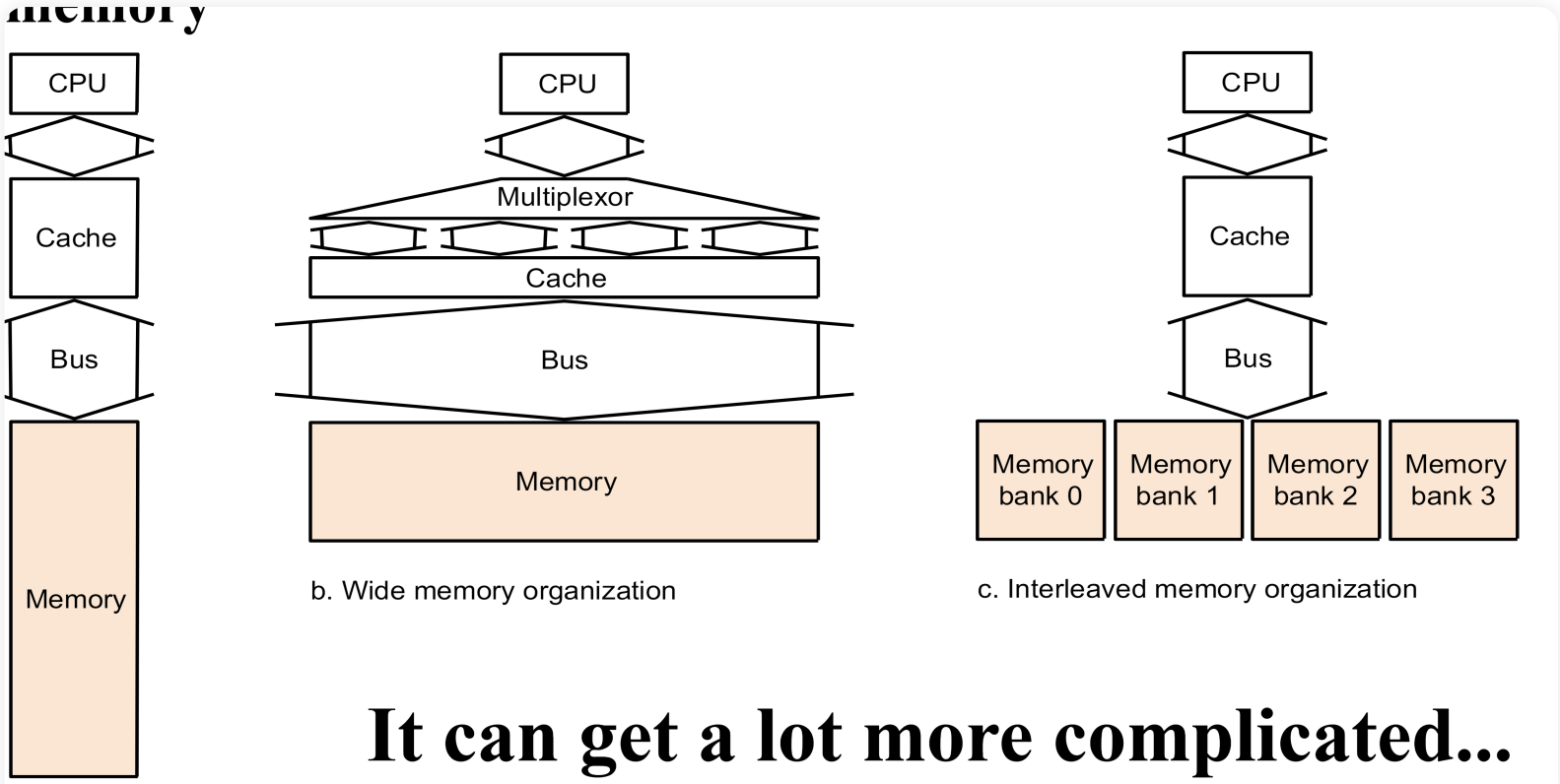
Note



m+2 实际上也可以合并，直接进行 block 内的 byte 寻址

6 Memory System

6.1 Memory Addressing



- Interleaved memory 的效果最好，CO_Ch5_1, p.60

6.2 Multi-level caches

⚠ Target

- 提升 high-level 的寻址速度
- 降低 SRAM 的 miss rate

Example >

- **Example:**
  - CPI of 1.0 on a 5GHz machine with a 2% miss rate, 100ns DRAM access
  - Adding 2nd level cache with 5ns access time decreases miss rate to 0.5%
- **Miss penalty to main memory is:** $\frac{\text{100ns}}{\text{0.2}}$ = 500 clock cycles
- **The CPI with one level of caching**

$$\textbf{Total CPI} = 1.0 + \textbf{Memory-stall cycles per instruction}$$

$$= 1.0 + 2\% \times 500 = 11.0$$

Miss penalty with levels of cache without access main memory

$$\frac{\text{5ns}}{\text{0.2}} = 25 \text{ clock cycles}$$

- **The CPI with Two level of cache with 0.5% miss rate for main memory**

$$\textbf{Total CPI} = 1.0 + \textbf{Primary stalls per instruction} + \textbf{Secondary stalls per instruction}$$

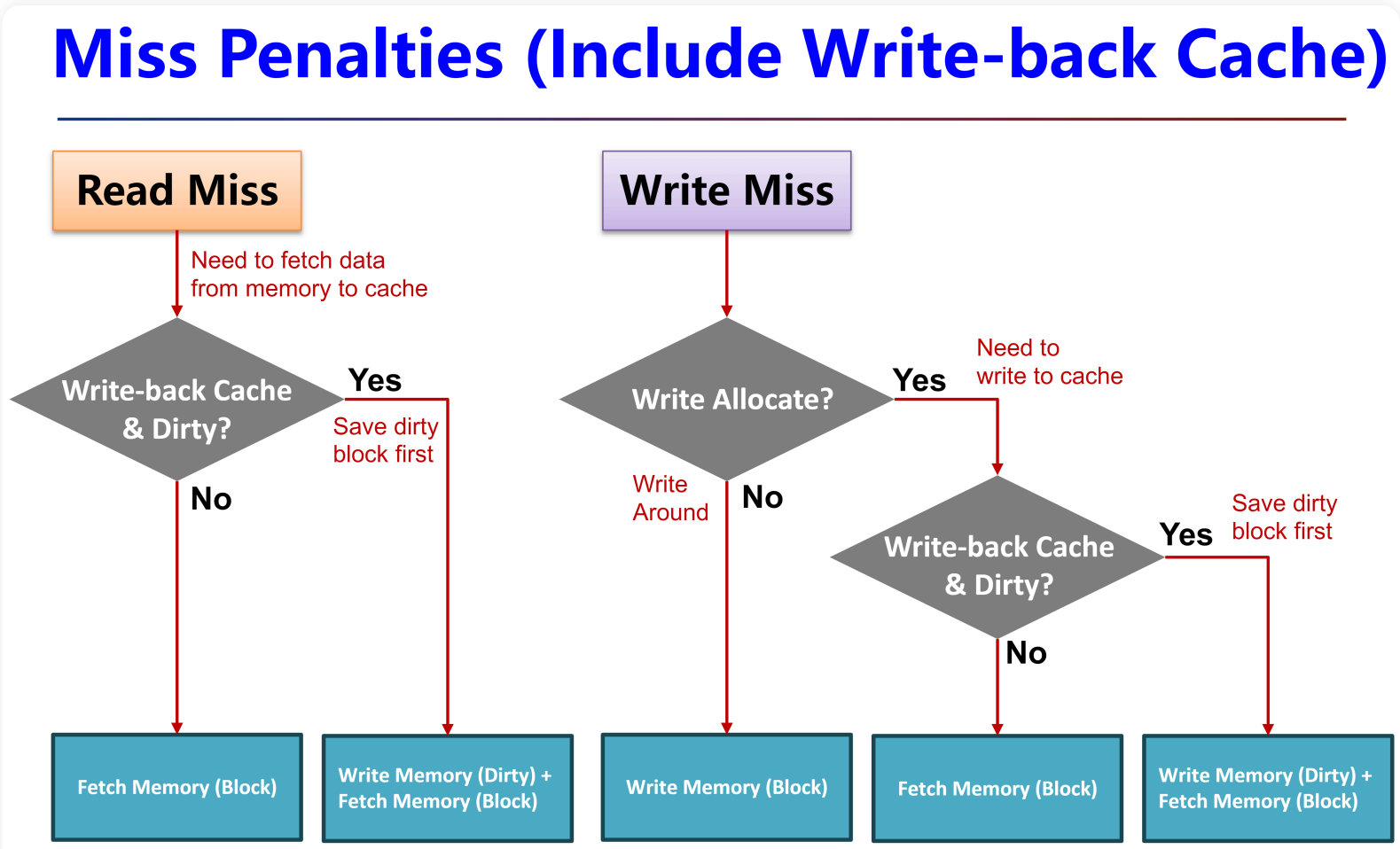
$$= 1 + 2\% \times 25 + 0.5\% \times 500$$

$$= 1.0 + 0.5 + 2.5 = 4.0$$

- **The processor with secondary cache is faster by**

$$\frac{11.0}{4.0} = 2.8$$

7 Measuring and imporving cache performance



- *If the write buffer stalls are small, we can safely ignore them. (No penalty on Write Memory)*
- *If the cache block size is one word, the write miss penalty is 0. (Except the block is dirty)*

- Average Memory Access Time(AMAT) = hit time + miss time = hit time + miss rate × miss penalty
- CPU Time = CPU exe. clock cycles + Mem-stall clock cycles

7.1 Mem-stall cycles

- $\text{Mem-stall cycles} = \#\text{mem inst.} \times \text{miss ratio} \times \text{miss penalty} = \text{Read-stall cycles} + \text{Write-stall cycles}$
  - $\text{Read-stall cycles} = \#\text{Read inst.} \times \text{Read miss rate} \times \text{Read miss penalty}$
  - $\text{Write-stall cycles} = (\#\text{Write inst.} \times \text{Write miss rate} \times \text{Write miss penalty}) + \text{Write buffer stalls}$
  - ! Read miss penalty = Write miss penalty，都多了个从 low level fetch 的过程
- If write buffer stalls are small, we can safely ignore them
- ? If the **cache block size** is one word, the **write miss penalty** is 0
  - 因为不需要再从 low-level 写到 high-level

7.2 Combine reads and writes

忽略 write buffer stalls，可以得到

$$\text{Mem-stall clock cycles} = \#\text{Mem access} \times \text{Miss rate} \times \text{Miss penalty}$$

7.3 计算题

Assume:

instruction cache miss rate	2%
data cache miss rate	4%
CPI without any memory stalls	2
miss penalty	100 cycles

The frequency of all loads and stores in gcc is 36%, as we see in Figure 3.26, on page 288.

Question: How faster a processor would run with a perfect cache?

Answer:

Instruction miss cycles = I×2%×100 =2.00I

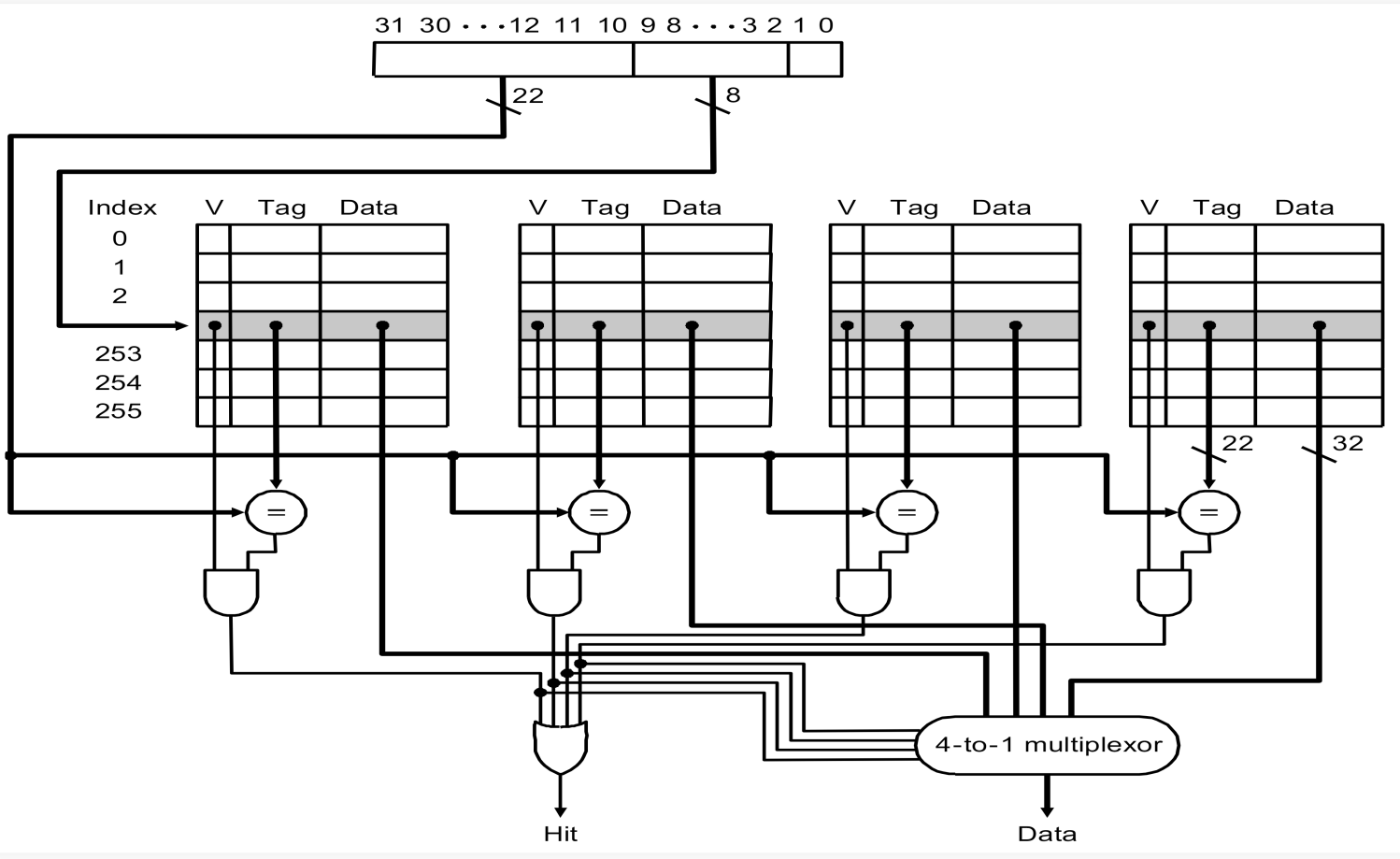
Data miss cycles = I×36%×4%×100 =1.44I

Total memory-stall cycles= 2.00I+ 1.44I =3.44 I

CPI with stall = CPI with perfect cache + total memory-stalls

= (2 + 3.44)I = 5.44I

> ⚠ Warning
>
> 如果 clock rate 变快一倍，**miss penalty 也会翻倍**



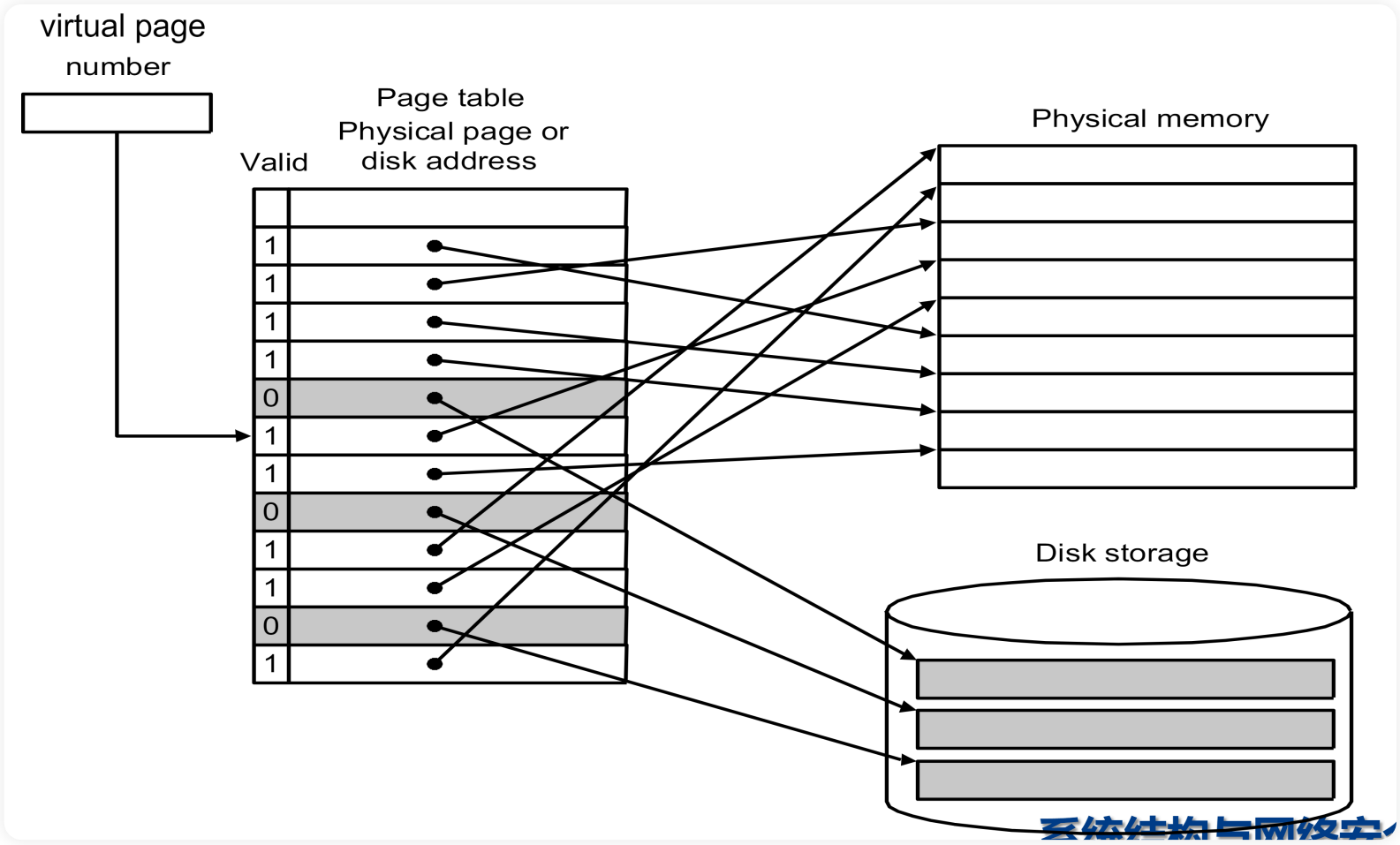
8 Virtual Memory

> ⚠ **Target**
>
> - 多线程内存共享
> - 内存安全，每个程序的虚拟地址可能有不同的映射，无法篡改
> - 将磁盘映射成内存，扩大可用内存

Pages: virtual memory blocks

- % #virtual pages > #physical pages, not really now
- **page faults**: 数据不在内存中，需要从 disk 读取
  - 由于 miss penalty 很大，所以 page 要设计的比较大 (e.g. 4KB)
  - 使用 LRU 来降低 miss rate
  - page fault 让软件来处理，因为 stall 的开销本身就很大
  - write-through 开销太大，write buffer 没有用处，只用 **write back**
- | virtual(physical) page number | page offset |

Page tables



- 使用 virutal page address 查表，得到 physical address
- 保存在内存中特殊的位置
- 一个 **page table register** 保存了对应程序的 page table 的基地址，不同的程序有各自的 page table，可以实现隔离
- valid bit 表示对应的数据是否在内存中

计算

Assume:

- Virtual address is 32 bits
- page size is 4KB
- Entry size is 4 Bytes

$$\textbf{Number of page table entries} = \frac{2^{32}}{2^{12}} = 2^{20}$$

$$\text{Size of page table} = 2^{20}\ \text{page table entries} \times 2^{2}\ \frac{\text{bytes}}{\text{page table entry}} = 4\text{MB}$$

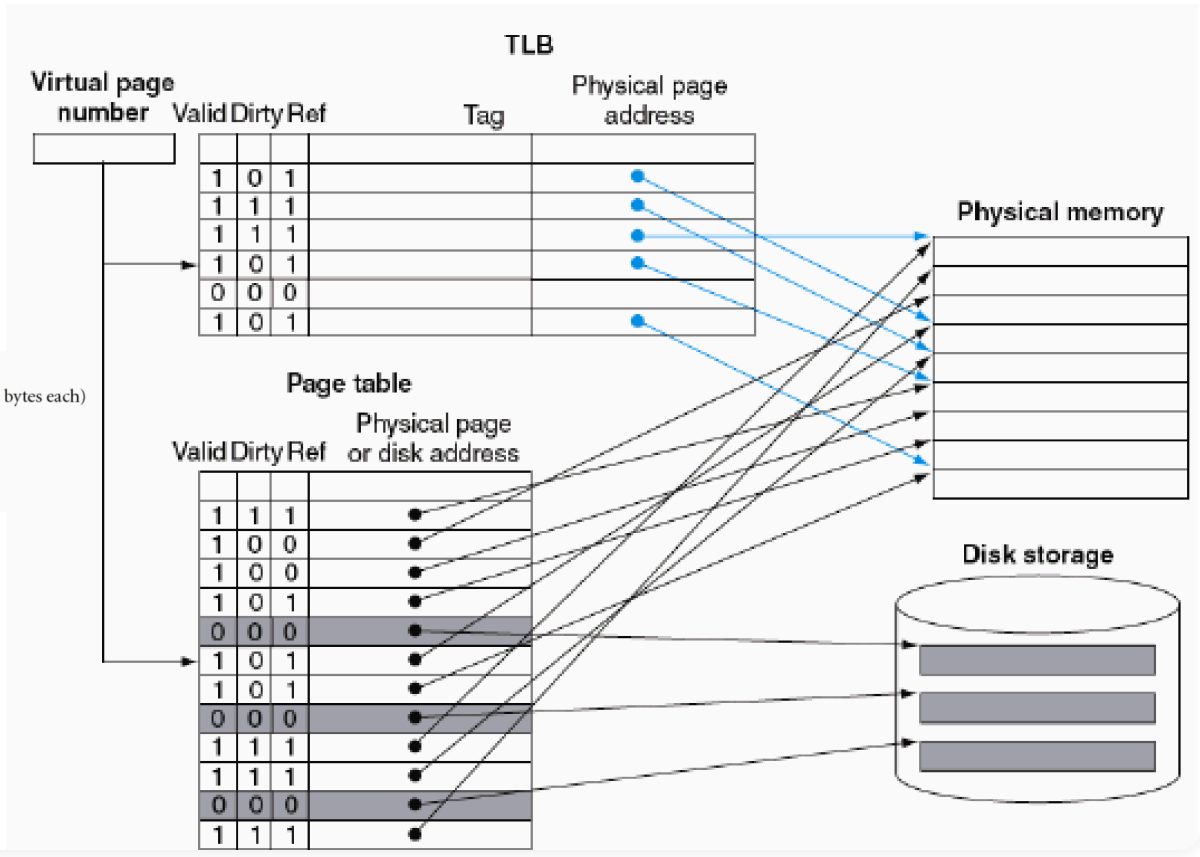
Page faults

- When the OS creates a process, it usually creates the space on idsk for all the pages of a process.
- page fault 发生时，OS 会根据 page table 找到对应的 disk 上的 page，并搬运到主存
- OS 也会实现 LRU

TLB(Translation-lookaside Buffer)

> a cache on the page table

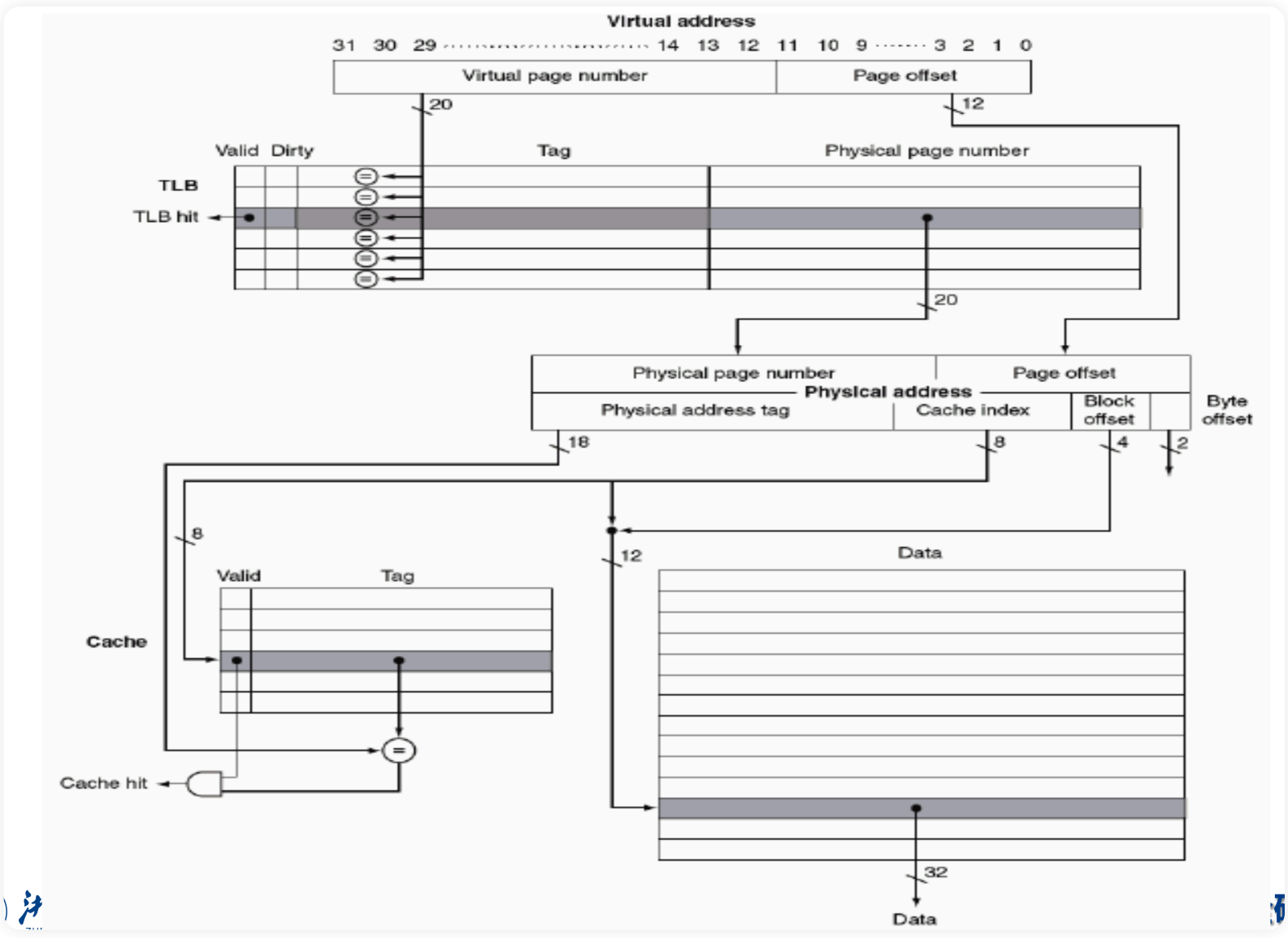
- **The TLB (Translation-lookaside Buffer) acts as Cache on the page table**
- **A cache for address translations: translation look aside buffer**



- TLB size: 16–512 entries
- Block size: 1–2 page table entries (typically 4–8 bytes each)
- Hit time: 0.5–1 clock cycle
- Miss penalty: 10–100 clock cycles
- Miss rate: 0.01%–1%

- 使用 tag 实现 associative
- reference bit 定期清零，实现简单的 LRU

Virtual address 访问 data 流程



1. Translation 过程：`virtual mem addr -> physical mem addr`
   - 在 TLB 中查找 virtual page number

    - 若 hit，得到 physical page number
    - 若 miss，需要查找 Page Table
        - 若 hit，得到 physical page number，*更新 TLB*
        - 若 miss(invalid)，page fault 操作，retry 时 hit，得到 physical page number，*更新 TLB*
2. Physical Memory Access：`physical mem addr -> data`
    - 查找是否在 cache 中
        - hit，返回数据
        - miss，查找更低级缓存或主存，*并向上更新*

> **Note**
>
> | TLB | Page table | Cache | Possible? If so, under what circumstance? |
> |---|---|---|---|
> | hit | hit | miss | Possible, although the page table is never really checked if TLB hits. |
> | miss | hit | hit | TLB misses, but entry found in page table; after retry, data is found in cache. |
> | miss | hit | miss | TLB misses, but entry found in page table; after retry, data misses in cache. |
> | miss | miss | miss | TLB misses and is followed by a page fault; after retry, data must miss in cache. |
> | hit | miss | miss | Impossible: cannot have a translation in TLB if page is not present in memory. |
> | hit | miss | hit | Impossible: cannot have a translation in TLB if page is not present in memory. |
> | miss | miss | hit | Impossible: data cannot be allowed in cache if the page is not in memory. |

- 如果数据不在 main memory 里，不可能 TLB hit 和 cache hit
- 有可能 TLB miss 但是 cache hit