Chapter 5

Large and Fast: Exploiting Memory Hierarchy

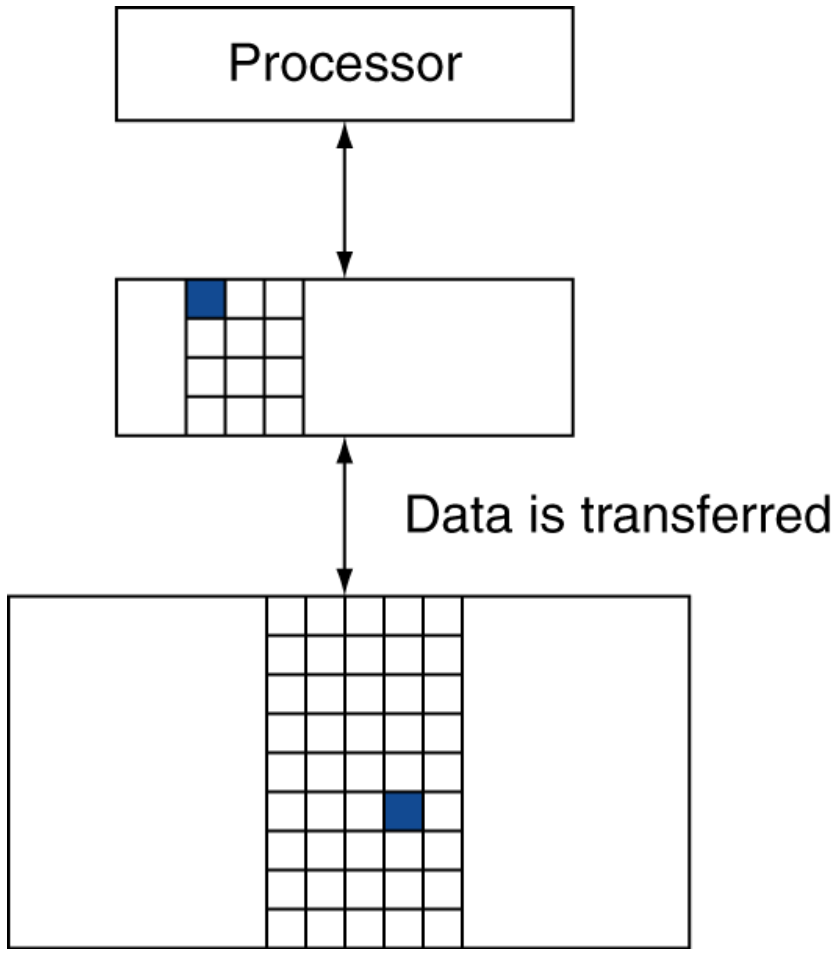
Principle of Locality

- Programs access a small proportion of their address space at any time
- Temporal locality
  - Items accessed recently are likely to be accessed again soon
  - e.g., instructions in a loop, induction variables
- Spatial locality
  - Items near those accessed recently are likely to be accessed soon
  - E.g., sequential instruction access, array data

Taking Advantage of Locality

- Memory hierarchy
- Store everything on disk
- Copy recently accessed (and nearby) items from disk to smaller DRAM memory
  - Main memory
- Copy more recently accessed (and nearby) items from DRAM to smaller SRAM memory
  - Cache memory attached to CPU

Memory Hierarchy Levels



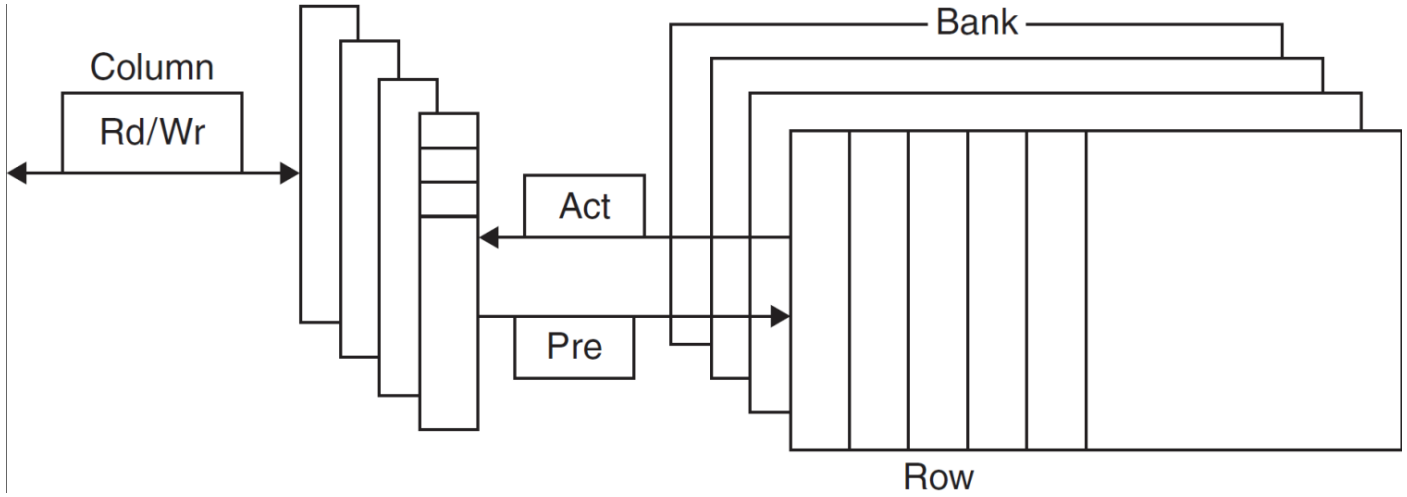
- Block (aka line): unit of copying
  - May be multiple words
- If accessed data is present in upper level
  - Hit: access satisfied by upper level
    - Hit ratio: hits/accesses
- If accessed data is absent
  - Miss: block copied from lower level
    - Time taken: miss penalty
    - Miss ratio: misses/accesses
      = 1 – hit ratio
  - Then accessed data supplied from upper level

Memory Technology

- Static RAM (SRAM)
  - 0.5ns – 2.5ns, $500 – $1000 per GiB
- Dynamic RAM (DRAM)
  - 50ns – 70ns, $3 – $6 per GiB
- Magnetic disk
  - 5ms – 20ms, $0.01 – $0.02 per GiB
- Ideal memory
  - Access time of SRAM
  - Capacity and cost/GB of disk

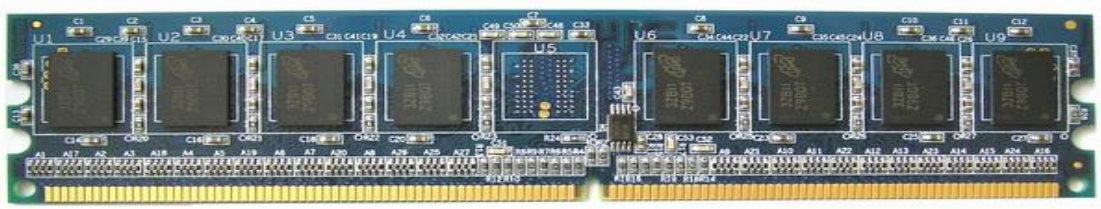
DRAM Technology

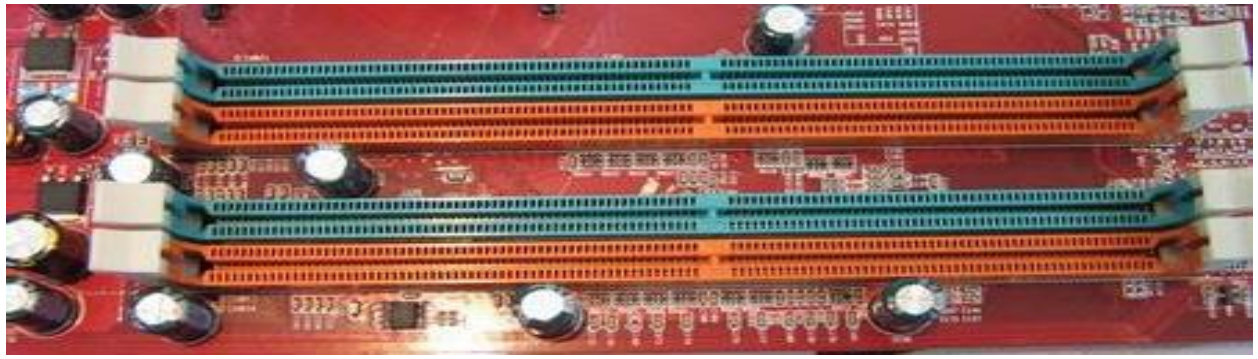
- Data stored as a charge in a capacitor
  - Single transistor used to access the charge
  - Must periodically be refreshed
    - Read contents and write back
    - Performed on a DRAM “row”



Advanced DRAM Organization

- Bits in a DRAM are organized as a rectangular array
  - DRAM accesses an entire row
  - Burst mode: supply successive words from a row with reduced latency
- Double data rate (DDR) DRAM
  - Transfer on rising and falling clock edges
- Quad data rate (QDR) DRAM
  - Separate DDR inputs and outputs



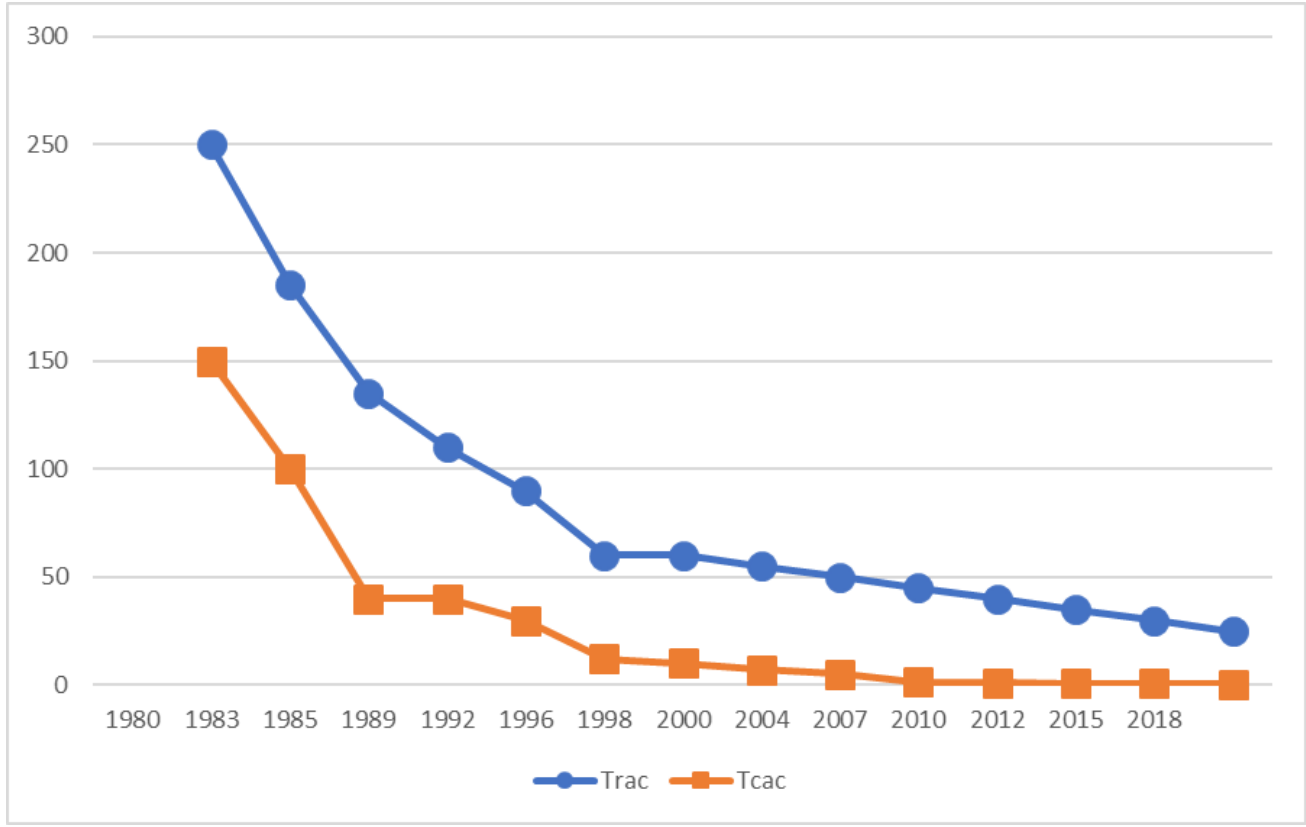


主存的主要性能指标

- 按字节连续编址，每个存储单元为1个字节（8个二进位）
- 存储容量：所包含的存储单元的总数（单位：MiB或GiB）
- 存取时间T_A：从CPU送出内存单元的地址开始，到主存读出数据并送到CPU（或者是把CPU数据写入主存）所需要的时间（单位是ns，1 ns = 10^{-9} s），分读取时间和写入时间
- 存储周期T_{MC}：连读两次访问存储器所需的最小时间间隔，它应等于存取时间加上下一次存取开始前所要求的附加时间，因此，T_{MC}比T_A大（因为存储器读出放大器、驱动电路等都需要有一段稳定恢复时间，所以读出后不能立即进行下一次访问）。

DRAM Generations

Year	Capacity	$/GiB
1980	64 Kib	$6,480,000
1983	256 Kib	$1,980,000
1985	1 Mib	$720,000
1989	4 Mib	$128,000
1992	16 Mib	$30,000
1996	64 Mib	$9,000
1998	128 Mib	$900
2000	256 Mib	$840
2004	512 Mib	$150
2007	1 Gib	$40
2010	2 Gib	$13
2012	4 Gib	$5
2015	8 Gib	$7
2018	16 Gib	$6



DRAM Performance Factors

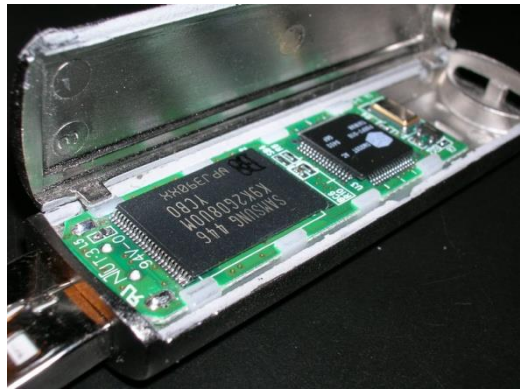
- Row buffer
  - Allows several words to be read and refreshed in parallel
- Synchronous DRAM
  - Allows for consecutive accesses in bursts without needing to send each address
  - Improves bandwidth
- DRAM banking
  - Allows simultaneous access to multiple DRAMs
  - Improves bandwidth

CPU与存储器之间的通信方式

- 异步方式（读操作）过程（需握手信号）
  - CPU送地址到地址线，主存进行地址译码
  - CPU发读命令，然后等待存储器发回“完成”信号
  - 主存收到读命令后开始读数，完成后发“完成”信号给CPU
  - CPU接收到“完成”信号，从数据线取数
  - 写操作过程类似
- 同步方式
  - CPU和主存由统一时钟信号控制，无需应答信号
  - 主存总是在确定的时间内准备好数据
  - CPU送出地址和读命令后，总是在确定的时间取数据
  - 存储器芯片必须支持同步方式

Flash Storage

- Nonvolatile semiconductor storage
  - 100× – 1000× faster than disk
  - Smaller, lower power, more robust
  - But more $/GiB (between disk and DRAM)

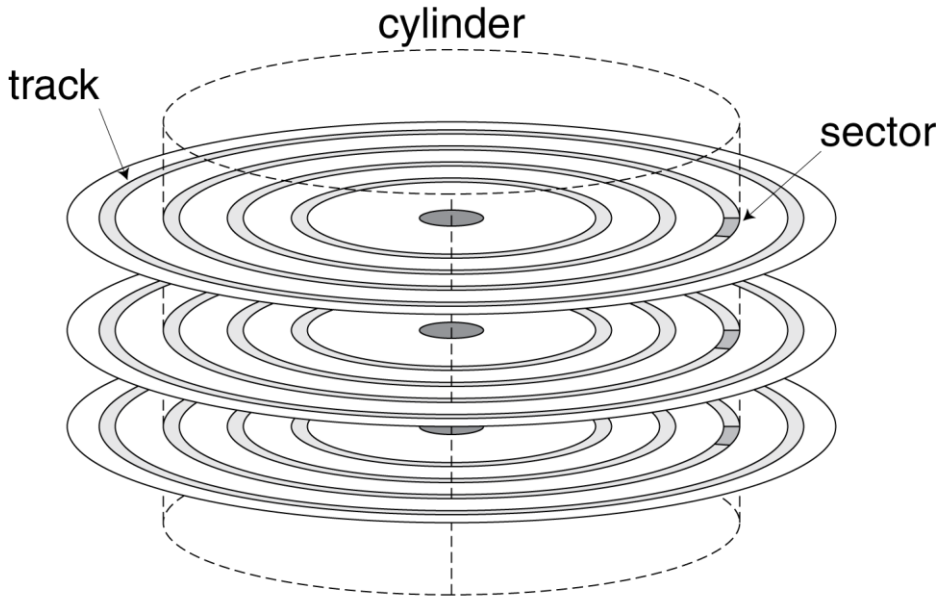


Flash Types

- NOR flash: bit cell like a NOR gate
  - Random read/write access
  - Used for instruction memory in embedded systems
- NAND flash: bit cell like a NAND gate
  - Denser (bits/area), but block-at-a-time access
  - Cheaper per GB
  - Used for USB keys, media storage, …
- Flash bits wears out after 100,000 of accesses
  - Not suitable for direct RAM or disk replacement
  - Wear leveling: remap data to less used blocks

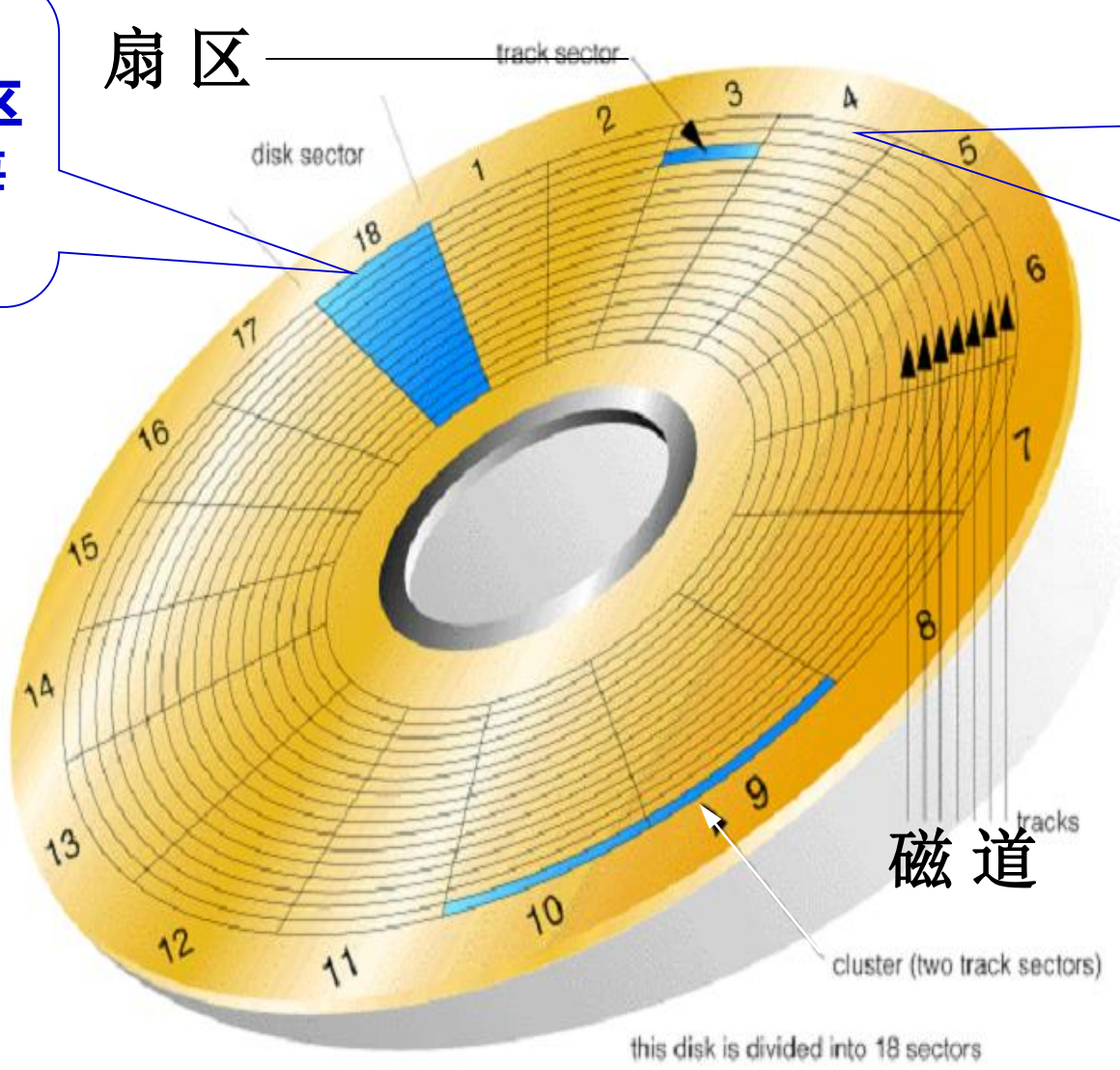
Disk Storage

- Nonvolatile, rotating magnetic storage



磁盘的磁道和扇区

每个磁道被划分为若干段（段又叫**扇区**），每个扇区的存储容量为512字节。每个扇区都有一个编号



磁盘表面被分为许多同心圆，每个同心圆称为一个**磁道**。每个磁道都有一个编号，最外面的是0磁道

早期扇区大小是512字节。但现在已经迁移到更大、更高效的4096字节扇区，通常称为**4K扇区**。国际硬盘设备与材料协会（IDEMA）将之称为高级格式化（磁盘格式化操作指在盘面上划分磁道和扇区并填写相关信息）。

现代磁盘磁道上的位密度相同，所以，外道上的扇区数比内道上扇区数多，使整个磁盘的容量提高

Disk Sectors and Access

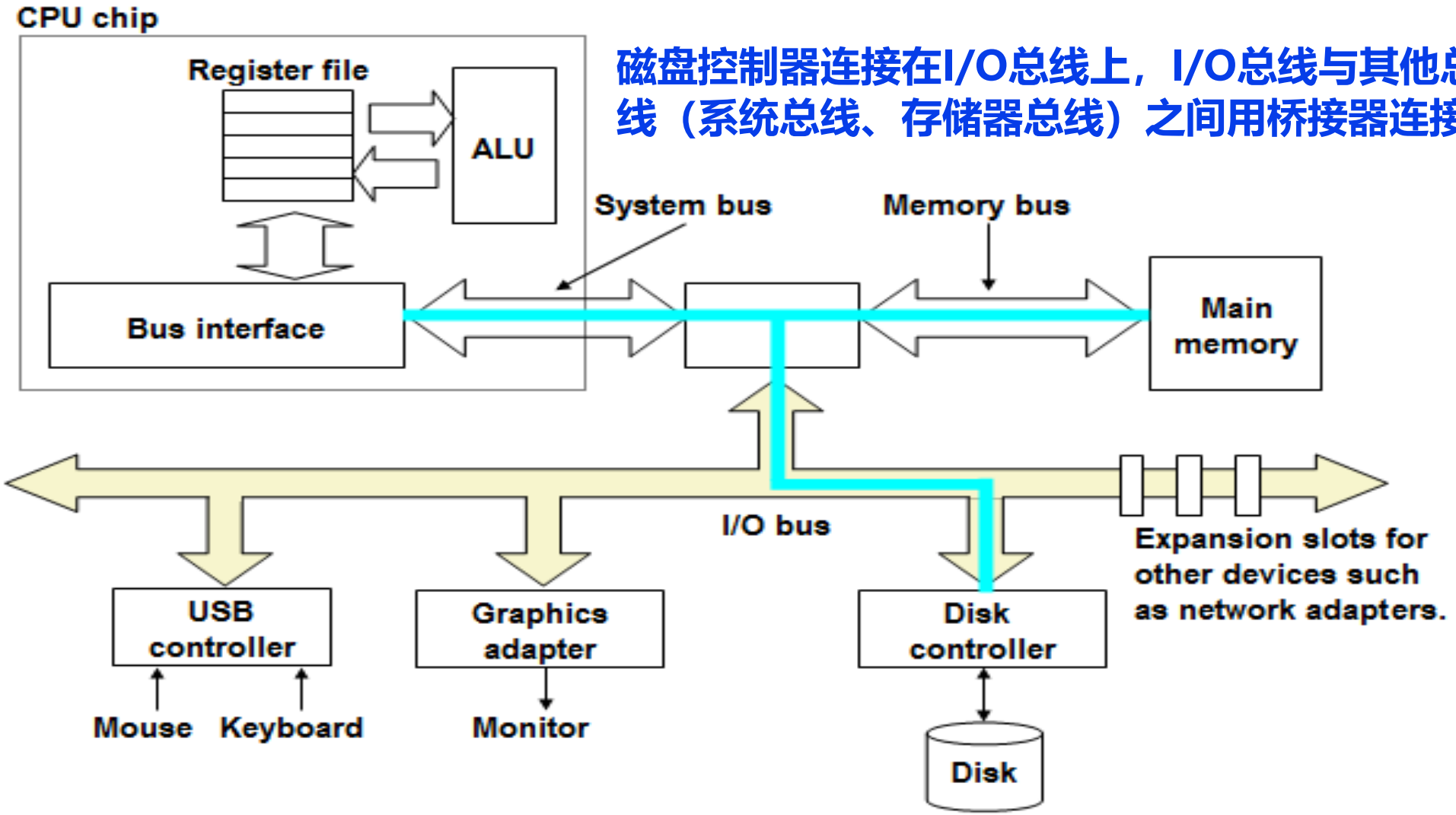
- Each sector records
  - Sector ID
  - Data (512 bytes, 4096 bytes proposed)
  - Error correcting code (ECC)
    - Used to hide defects and recording errors
- Access to a sector involves
  - Queuing delay if other accesses are pending
  - Seek: move the heads
  - Rotational latency
  - Data transfer
  - Controller overhead

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

磁盘存储器的连接

磁盘控制器连接在I/O总线上，I/O总线与其他总线（系统总线、存储器总线）之间用桥接器连接



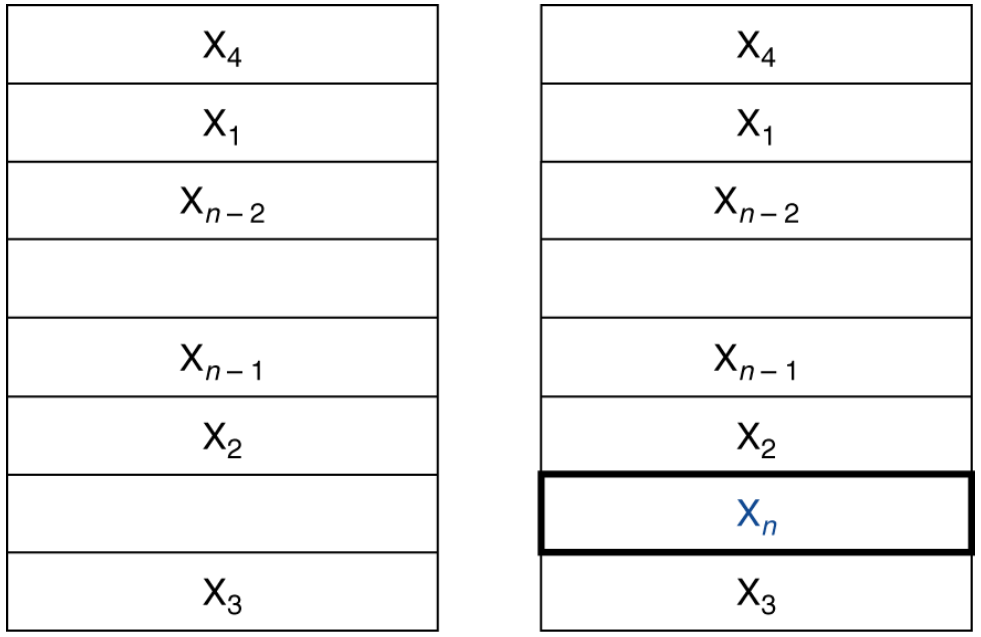
磁盘的最小读写单位是扇区，磁盘按成批数据交换方式进行读写，采用直接存储器存取（DMA，Direct Memory Access）方式进行数据输入输出，需用专门的DMA接口来控制外设与主存间直接数据交换，数据不通过CPU。通常把专门用来控制总线进行DMA传送的接口硬件称为DMA控制器

Disk Performance Issues

- Manufacturers quote average seek time
  - Based on all possible seeks
  - Locality and OS scheduling lead to smaller actual average seek times
- Smart disk controller allocate physical sectors on disk
  - Present logical sector interface to host
  - SCSI, ATA, SATA
- Disk drives include caches
  - Prefetch sectors in anticipation of access
  - Avoid seek and rotational delay

Cache Memory

- Cache memory
  - The level of the memory hierarchy closest to the CPU
- Given accesses X_1, …, X_{n-1}, X_n

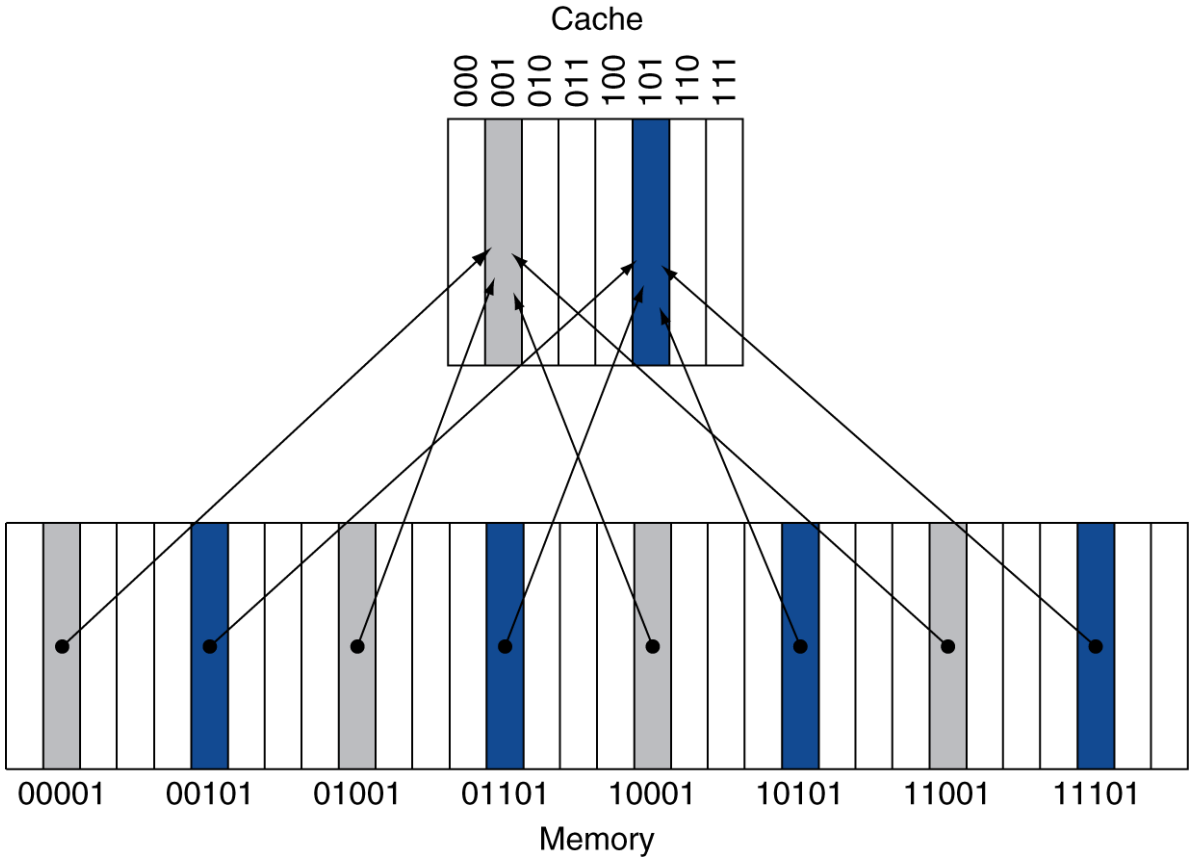


a. Before the reference to X_n b. After the reference to X_n

- How do we know if the data is present?
- Where do we look?

Direct Mapped Cache

- Location determined by address
- Direct mapped: only one choice
  - (Block address) modulo (#Blocks in cache)



- #Blocks is a power of 2
- Use low-order address bits

Tags and Valid Bits

- How do we know which particular block is stored in a cache location?
  - Store block address as well as the data
  - Actually, only need the high-order bits
  - Called the tag
- What if there is no data in a location?
  - Valid bit: 1 = present, 0 = not present
  - Initially 0

Cache Example

- 8-blocks, 1 word/block, direct mapped
- Initial state

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
22	10 110	Miss	110

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	**Y**	**10**	**Mem[10110]**
111	N		

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
26	11 010	Miss	010

Index	V	Tag	Data
000	N		
001	N		
010	**Y**	**11**	**Mem[11010]**
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
22	10 110	Hit	110
26	11 010	Hit	010

Index	V	Tag	Data
000	N		
001	N		
010	Y	11	Mem[11010]
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
16	10 000	Miss	000
3	00 011	Miss	011
16	10 000	Hit	000

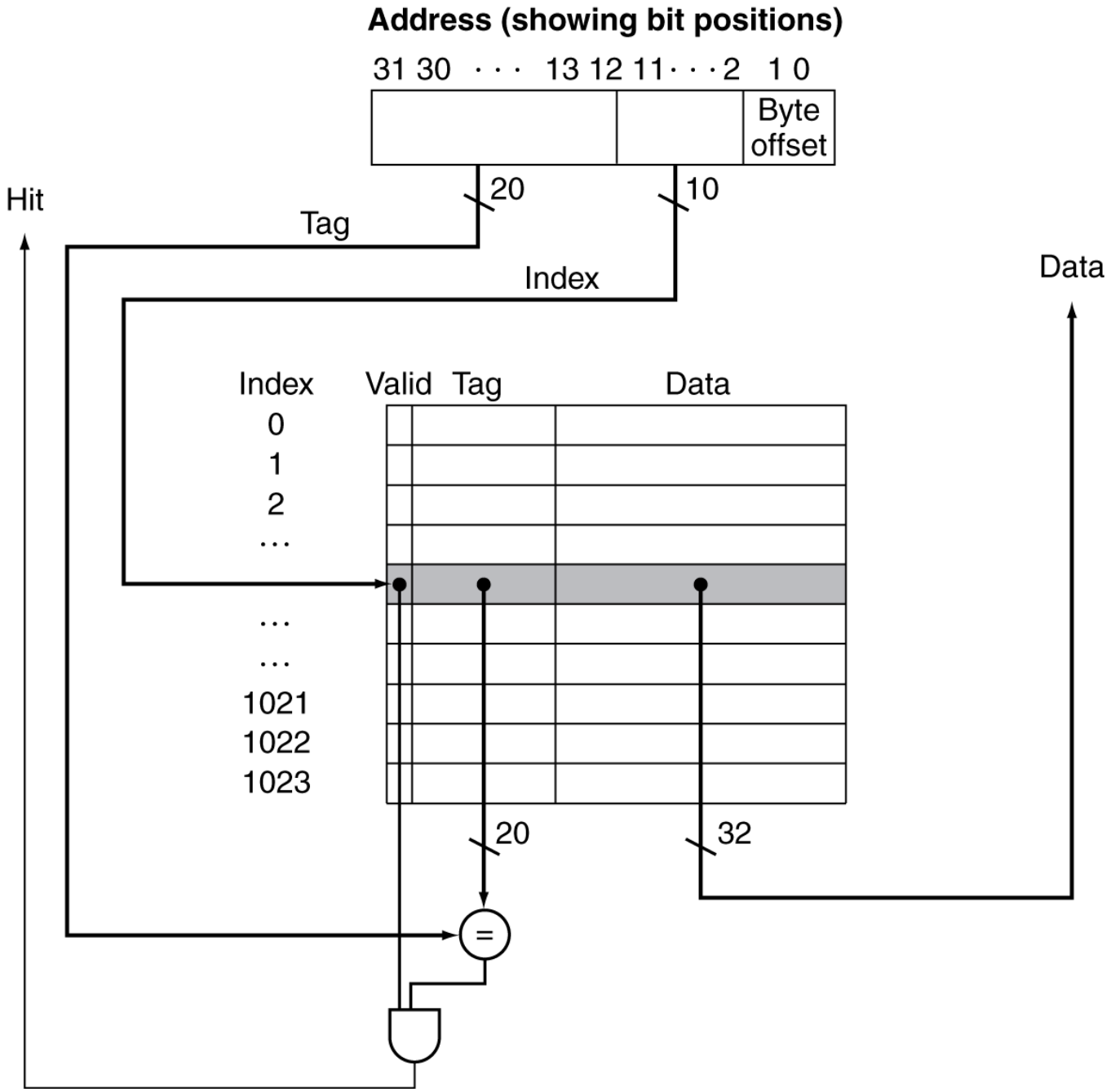
Index	V	Tag	Data
000	**Y**	**10**	**Mem[10000]**
001	N		
010	Y	11	Mem[11010]
011	**Y**	**00**	**Mem[00011]**
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
18	10 010	Miss	010

Index	V	Tag	Data
000	Y	10	Mem[10000]
001	N		
010	**Y**	**10**	**Mem[10010]**
011	Y	00	Mem[00011]
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Address Subdivision

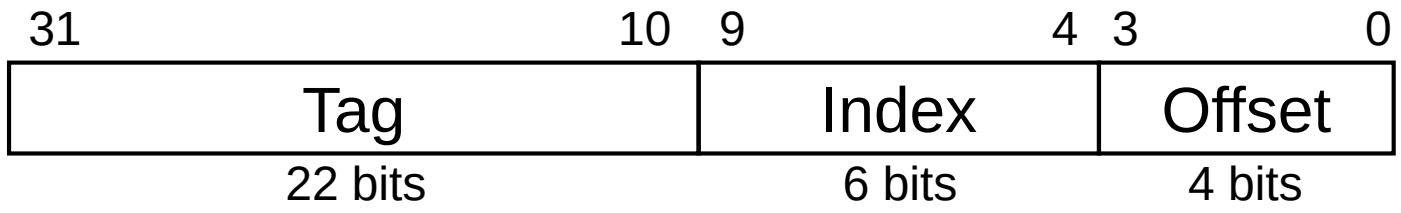


Cache大小

- Suppose
  - The cache size is 2^n blocks, so n bits are used for the index
  - The block size is 2^m words (2^{m+2} bytes), so m bits are used for the word within the block, and two bits are used for the byte part of the address
- 标记字段的大小是32-(n+m+2)
- 直接映射cache的总位数是: 2^n(数据大小+标记字段大小+有效位大小）= $2^n \cdot (32 \cdot 2^m + (32-n-m-2)+1)$
- 对于上图，n=10，m=0，则数据占用4KiB，标记和有效位字段占 $2^{10} \cdot 21$=2.625KiB，通常还是称为4KiB Cache

Example: Larger Block Size

- 64 blocks, 16 bytes/block
  - To what block number does address 1200 map?
- Block address = $\lfloor 1200/16 \rfloor = 75$
- Block number = 75 modulo 64 = 11



Block Size Considerations

- Larger blocks should reduce miss rate
  - Due to spatial locality
- But in a fixed-sized cache
  - Larger blocks $\Rightarrow$ fewer of them
    - More competition $\Rightarrow$ increased miss rate
  - Larger miss penalty
    - Early restart and critical-word-first can help

Cache Misses

- On cache hit, CPU proceeds normally
- On cache miss
  - Stall the CPU pipeline
  - Fetch block from next level of hierarchy
  - Instruction cache miss
    - Restart instruction fetch
  - Data cache miss
    - Complete data access

Write-Through

- On data-write hit, could just update the block in cache
  - But then cache and memory would be inconsistent
- Write through: also update memory
- But makes writes take longer
  - e.g., if base CPI = 1, 10% of instructions are stores, write to memory takes 100 cycles
    - Effective CPI = 1 + 0.1×100 = 11
- Solution: write buffer
  - Holds data waiting to be written to memory
  - CPU continues immediately
    - Only stalls on write if write buffer is already full

Write-Back

- Alternative: On data-write hit, just update the block in cache
  - Keep track of whether each block is dirty
- When a dirty block is replaced
  - Write it back to memory
  - Can use a write buffer to allow replacing block to be read first

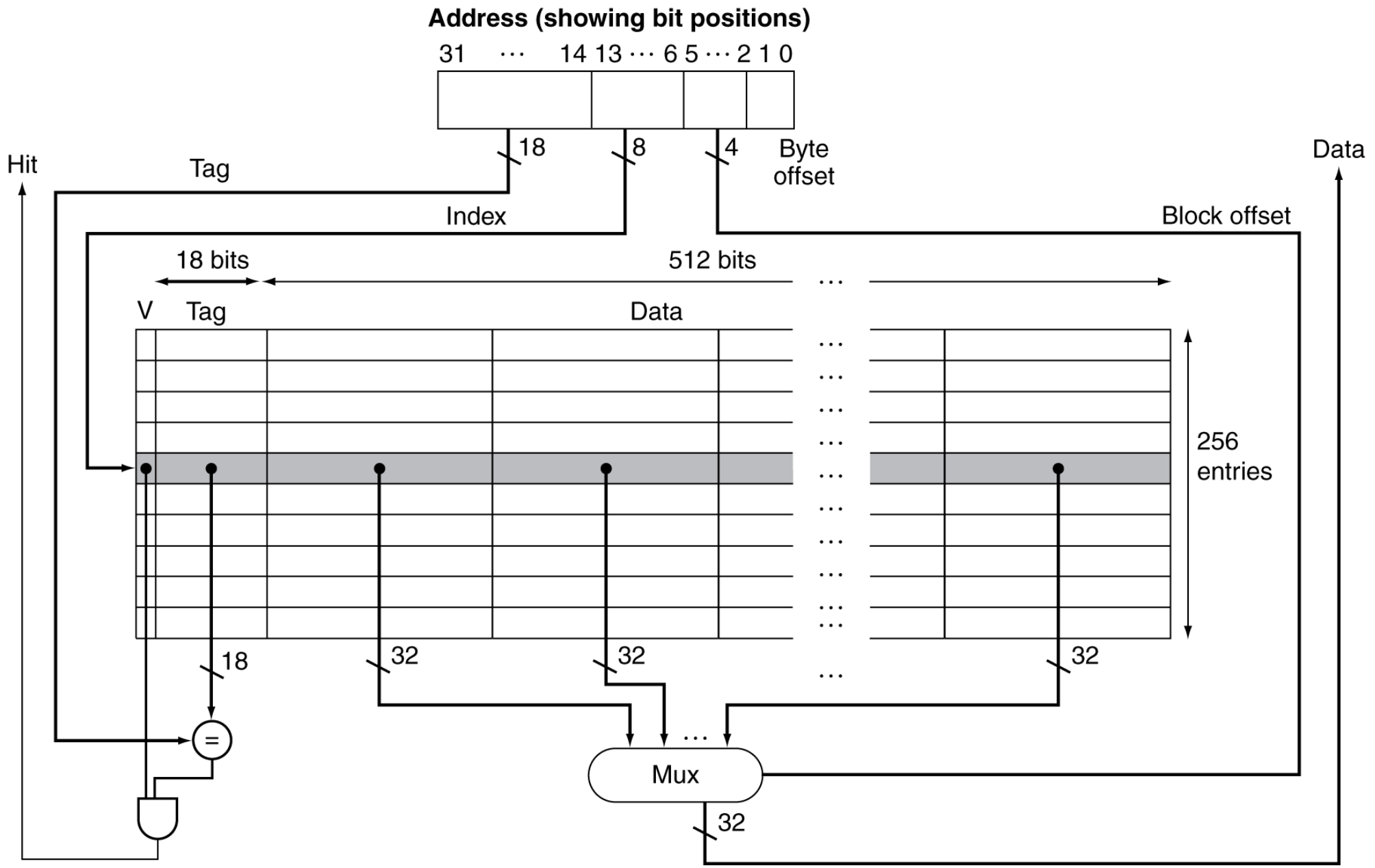
Write Allocation

- What should happen on a write miss?
- Alternatives for write-through
  - Write allocate: fetch the block
  - No write allocate: don’t fetch the block
    - Since programs often write a whole block before reading it (e.g., initialization)
- For write-back
  - Usually fetch the block

Example: Intrinsity FastMATH

- Embedded MIPS processor
  - 12-stage pipeline
  - Instruction and data access on each cycle
- Split cache: separate I-cache and D-cache
  - Each 16KB: 256 blocks × 16 words/block
  - D-cache: write-through or write-back
- SPEC2000 miss rates
  - I-cache: 0.4%
  - D-cache: 11.4%
  - Weighted average: 3.2%

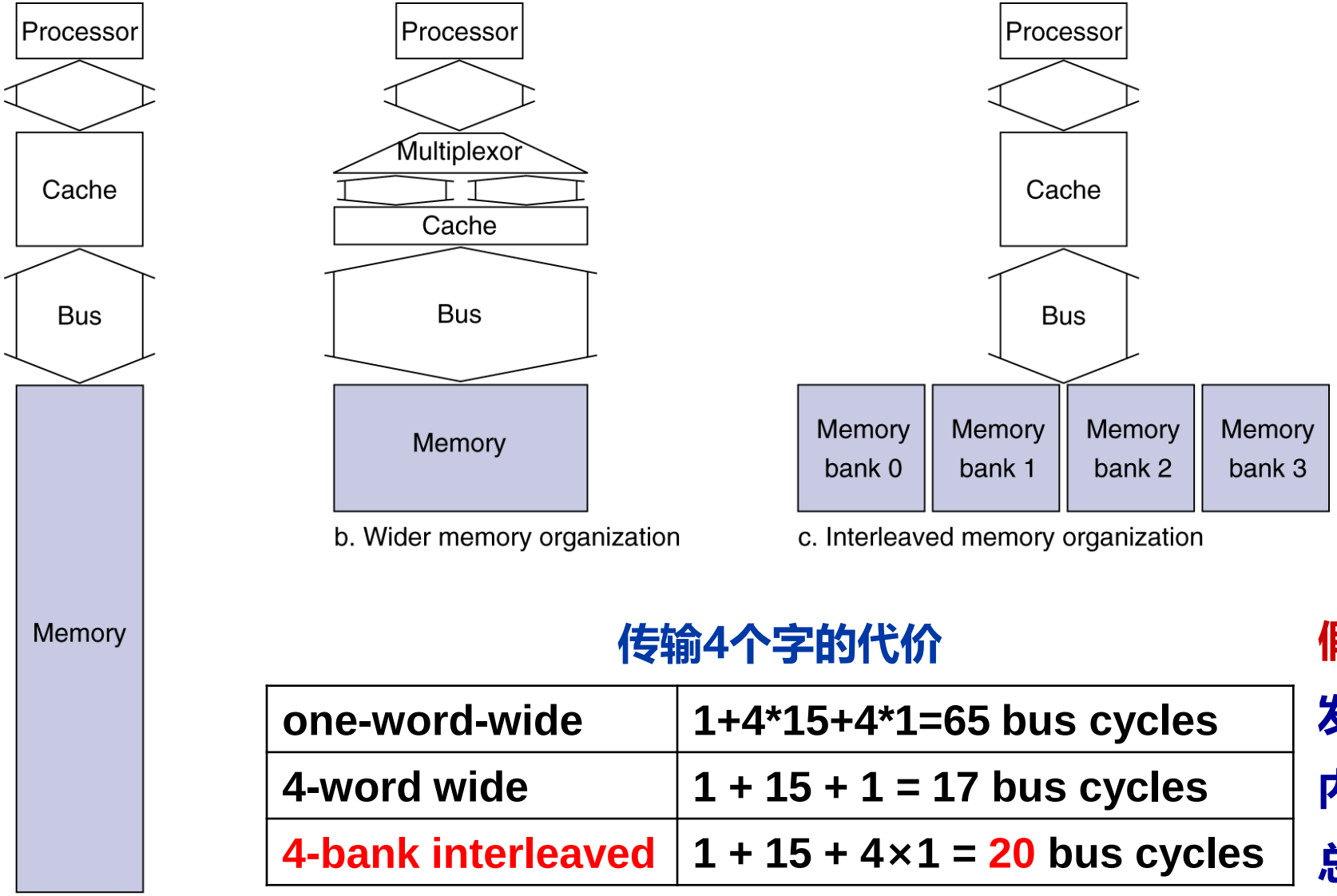
Example: Intrinsity FastMATH



Main Memory Supporting Caches

- Use DRAMs for main memory
  - Fixed width (e.g., 1 word)
  - Connected by fixed-width clocked bus
    - Bus clock is typically slower than CPU clock
- Example cache block read
  - 1 bus cycle for address transfer
  - 15 bus cycles per DRAM access
  - 1 bus cycle per data transfer
- For 4-word block, 1-word-wide DRAM
  - Miss penalty = 1 + 4×15 + 4×1 = 65 bus cycles
  - Bandwidth = 16 bytes / 65 cycles = 0.25 B/cycle

设计支持Cache的存储器系统



a. One-word-wide memory organization

b. Wider memory organization

c. Interleaved memory organization

传输4个字的代价

one-word-wide	1+4*15+4*1=65 bus cycles
4-word wide	1 + 15 + 1 = 17 bus cycles
4-bank interleaved	1 + 15 + 4×1 = 20 bus cycles

假定存储器访问过程：

发送地址到内存：1个总线时钟

内存访问时间：15个总线时钟

总线上传送一个字：1个总线时钟

Measuring Cache Performance

- Components of CPU time
  - Program execution cycles
    - Includes cache hit time
  - Memory stall cycles
    - Mainly from cache misses
- With simplifying assumptions:

$$\text{Memory stall cycles}$$

$$= \frac{\text{Memory accesses}}{\text{Program}} \times \text{Miss rate} \times \text{Miss penalty}$$

$$= \frac{\text{Instructio ns}}{\text{Program}} \times \frac{\text{Misses}}{\text{Instructio n}} \times \text{Miss penalty}$$

Cache Performance Example

- Given
  - I-cache miss rate = 2%
  - D-cache miss rate = 4%
  - Miss penalty = 100 cycles
  - Base CPI (ideal cache) = 2
  - Load & stores are 36% of instructions
- Miss cycles per instruction
  - I-cache: 0.02 × 100 = 2
  - D-cache: 0.36 × 0.04 × 100 = 1.44
- Actual CPI = 2 + 2 + 1.44 = 5.44
  - Ideal CPU is 5.44/2 =2.72 times faster

Average Access Time

- Hit time is also important for performance
- Average memory access time (AMAT)
  - AMAT = Hit time + Miss rate × Miss penalty
- Example
  - CPU with 1ns clock, hit time = 1 cycle,
    miss penalty = 20 cycles, cache miss rate = 5%
  - AMAT = 1 + 0.05 × 20 = 2ns
    - 2 cycles per instruction

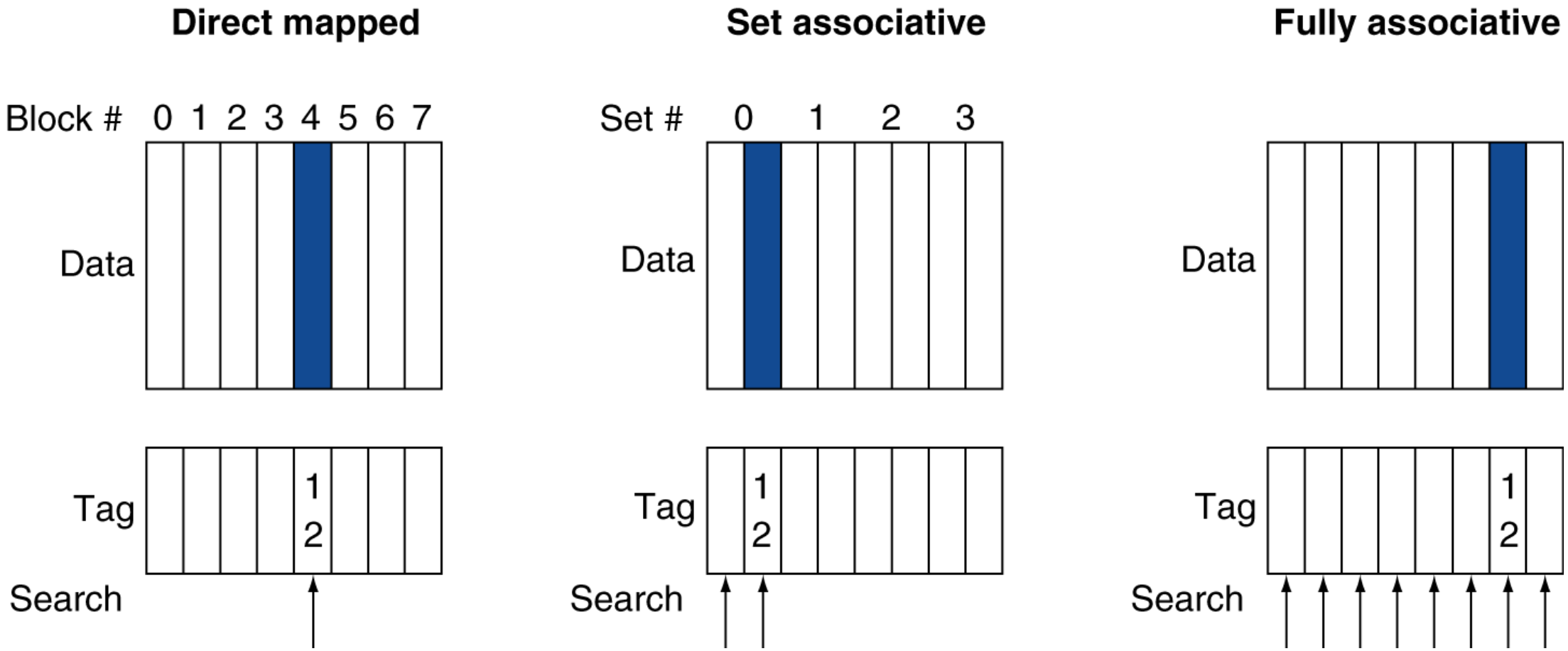
Performance Summary

- When CPU performance increased
  - Miss penalty becomes more significant
- Decreasing base CPI
  - Greater proportion of time spent on memory stalls
- Increasing clock rate
  - Memory stalls account for more CPU cycles
- Can’t neglect cache behavior when evaluating system performance

Associative Caches

- Fully associative
  - Allow a given block to go in any cache entry
  - Requires all entries to be searched at once
  - Comparator per entry (expensive)
- *n*-way set associative
  - Each set contains *n* entries
  - Block number determines which set
    - (Block number) modulo (#Sets in cache)
  - Search all entries in a given set at once
  - *n* comparators (less expensive)

Associative Cache Example



Spectrum of Associativity

- For a cache with 8 entries

One-way set associative (direct mapped)

Block	Tag	Data
0		
1		
2		
3		
4		
5		
6		
7		

Two-way set associative

Set	Tag	Data	Tag	Data
0				
1				
2				
3				

Four-way set associative

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

Eight-way set associative (fully associative)

Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data

Associativity Example

- Compare 4-block caches
  - Direct mapped, 2-way set associative, fully associative
  - Block access sequence: 0, 8, 0, 6, 8
- Direct mapped

Block address	Cache index	Hit/miss	Cache content after access			
			0	1	2	3
0	0	miss	**Mem[0]**			
8	0	miss	**Mem[8]**			
0	0	miss	**Mem[0]**			
6	2	miss	Mem[0]		**Mem[6]**	
8	0	miss	**Mem[8]**		Mem[6]	

Associativity Example

- 2-way set associative

Block address	Cache index	Hit/miss	Cache content after access			
			Set 0		Set 1	
0	0	miss	**Mem[0]**			
8	0	miss	Mem[0]	**Mem[8]**		
0	0	hit	**Mem[0]**	Mem[8]		
6	0	miss	Mem[0]	**Mem[6]**		
8	0	miss	**Mem[8]**	Mem[6]		

Associativity Example

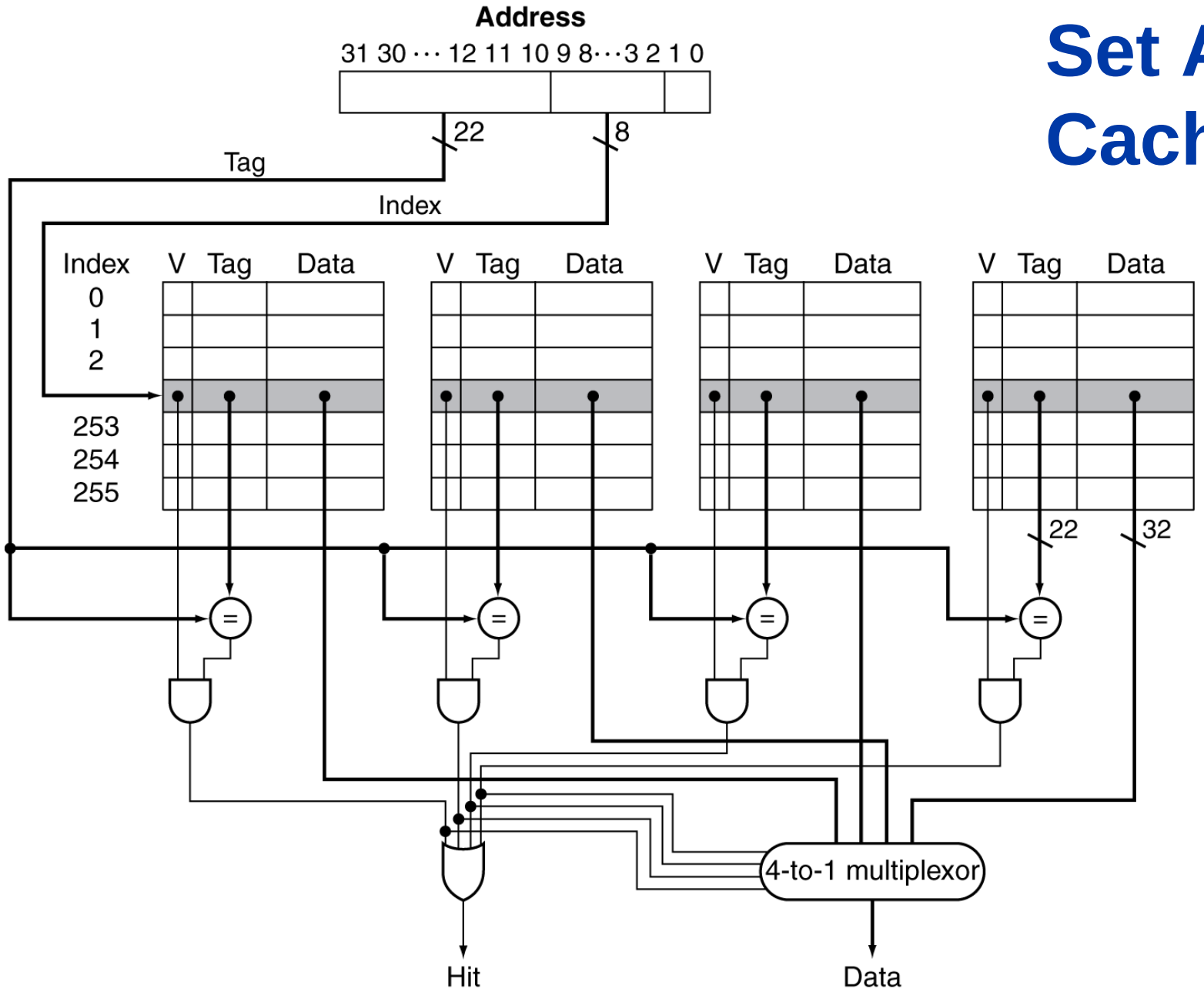
- Fully associative

Block address		Hit/miss	Cache content after access			
0		miss	**Mem[0]**			
8		miss	Mem[0]	**Mem[8]**		
0		hit	**Mem[0]**	Mem[8]		
6		miss	Mem[0]	Mem[8]	**Mem[6]**	
8		hit	Mem[0]	**Mem[8]**	Mem[6]	

How Much Associativity

- Increased associativity decreases miss rate
  - But with diminishing returns
- Simulation of a system with 64KB D-cache, 16-word blocks, SPEC2000
  - 1-way: 10.3%
  - 2-way: 8.6%
  - 4-way: 8.3%
  - 8-way: 8.1%
- 2-路或4-路组相联较常用。在较大容量的L2和L3 Cache中使用4-路以上
- 关联度越高，总的标记位数越多，额外空间开销越大!

Set Associative Cache Organization



Replacement Policy

- Direct mapped: no choice
- Set associative
  - Prefer non-valid entry, if there is one
  - Otherwise, choose among entries in the set
- Least-recently used (LRU)
  - Choose the one unused for the longest time
    - Simple for 2-way, manageable for 4-way, too hard beyond that
- Random
  - Gives approximately the same performance as LRU for high associativity

Multilevel Caches

- Primary cache attached to CPU
  - Small, but fast
- Level-2 cache services misses from primary cache
  - Larger, slower, but still faster than main memory
- Main memory services L-2 cache misses
- Some high-end systems include L-3 cache

Multilevel Cache Example

- Given
  - CPU base CPI = 1, clock rate = 4GHz
  - Miss rate/instruction = 2%
  - Main memory access time = 100ns
- With just primary cache
  - Miss penalty = 100ns/0.25ns = 400 cycles
  - Effective CPI = 1 + 0.02 × 400 = 9

Example (cont.)

- Now add L-2 cache
  - Access time = 5ns
  - Global miss rate to main memory = 0.5%
- Primary miss with L-2 hit
  - Penalty = 5ns/0.25ns = 20 cycles
- Primary miss with L-2 miss
  - Extra penalty = 400 cycles
- CPI = 1 + 0.02 × 20 + 0.005 × 400 = 3.4
- Performance ratio = 9/3.4 = 2.6

Multilevel Cache Considerations

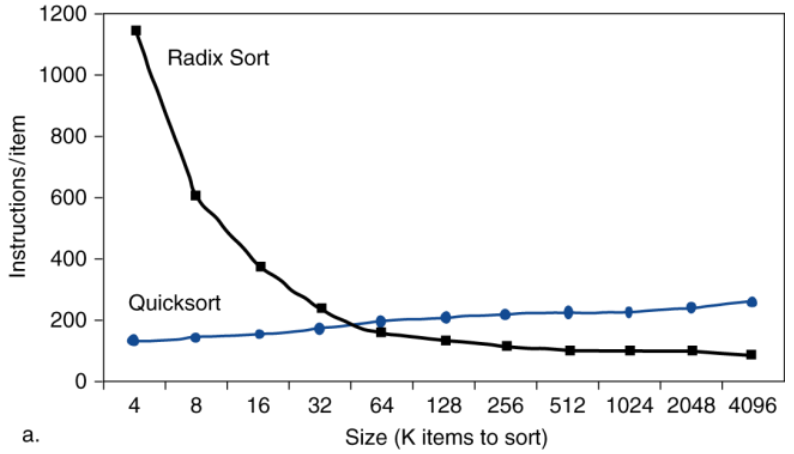
- Primary cache
  - Focus on minimal hit time
- L-2 cache
  - Focus on low miss rate to avoid main memory access
  - Hit time has less overall impact
- Results
  - L-1 cache usually smaller than L-2 cache
  - L-1 block size smaller than L-2 block size

Interactions with Advanced CPUs

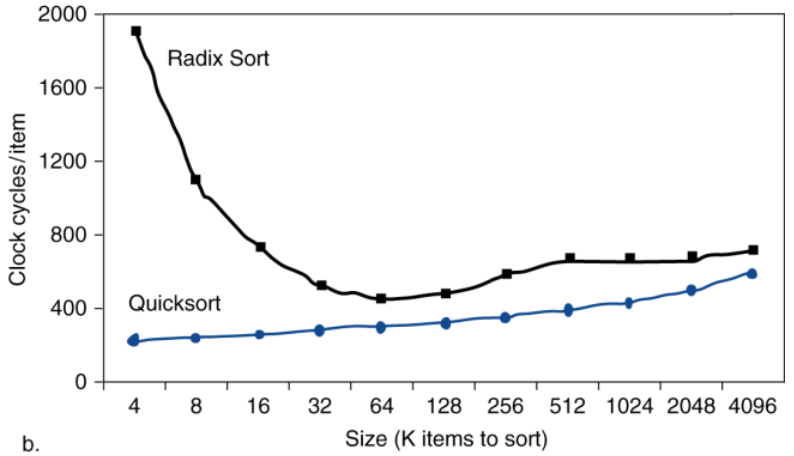
- Out-of-order CPUs can execute instructions during cache miss
  - Pending store stays in load/store unit
  - Dependent instructions wait in reservation stations
    - Independent instructions continue
- Effect of miss depends on program data flow
  - Much harder to analyse
  - Use system simulation

Interactions with Software

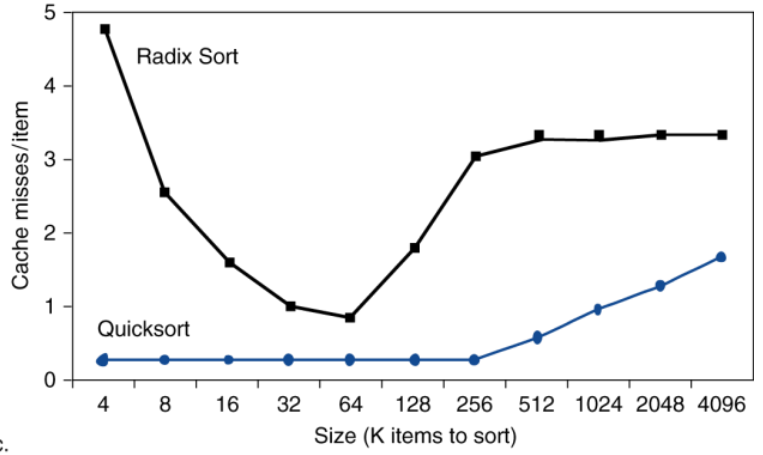
- Misses depend on memory access patterns
  - Algorithm behavior
  - Compiler optimization for memory access



a.



b.



c.

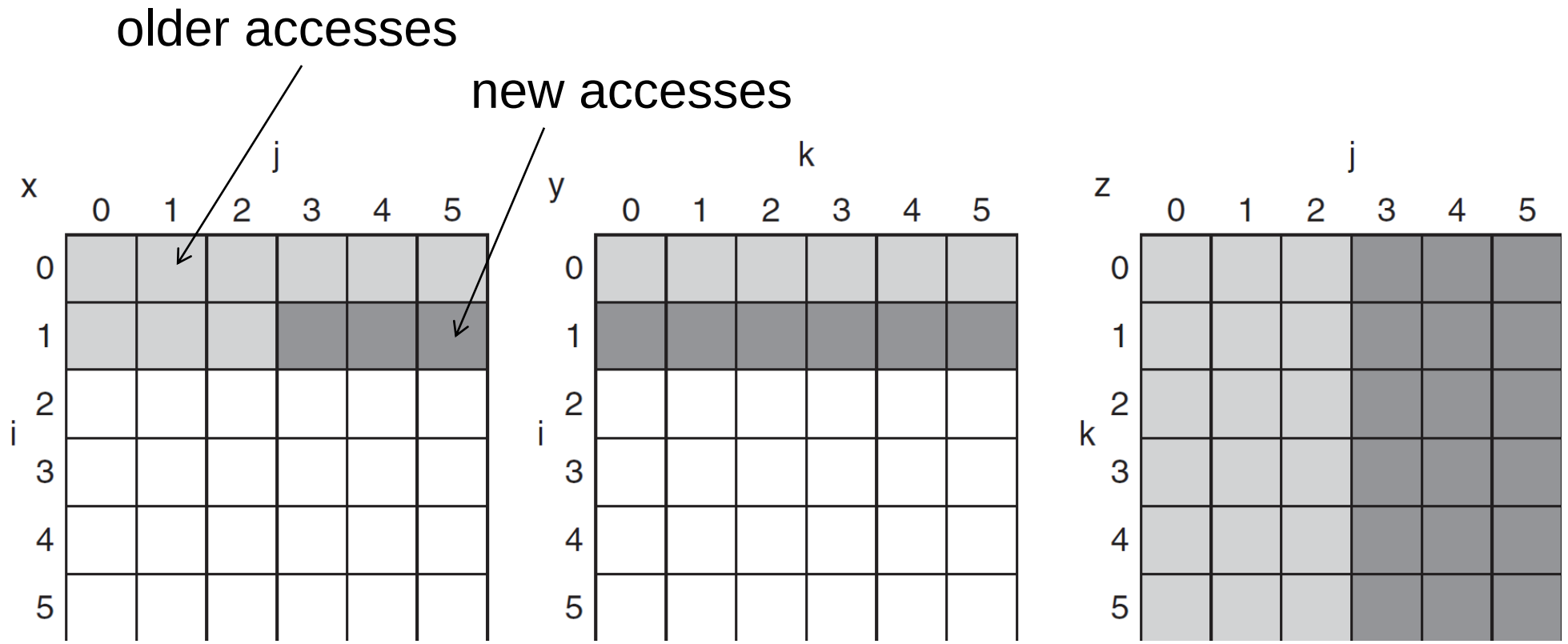
Software Optimization via Blocking

- Goal: maximize accesses to data before it is replaced
- Consider inner loops of DGEMM:

```
for (int j = 0; j < n; ++j)
{
  double cij = C[i+j*n];
  for( int k = 0; k < n; k++ )
    cij += A[i+k*n] * B[k+j*n];
  C[i+j*n] = cij;
}
```

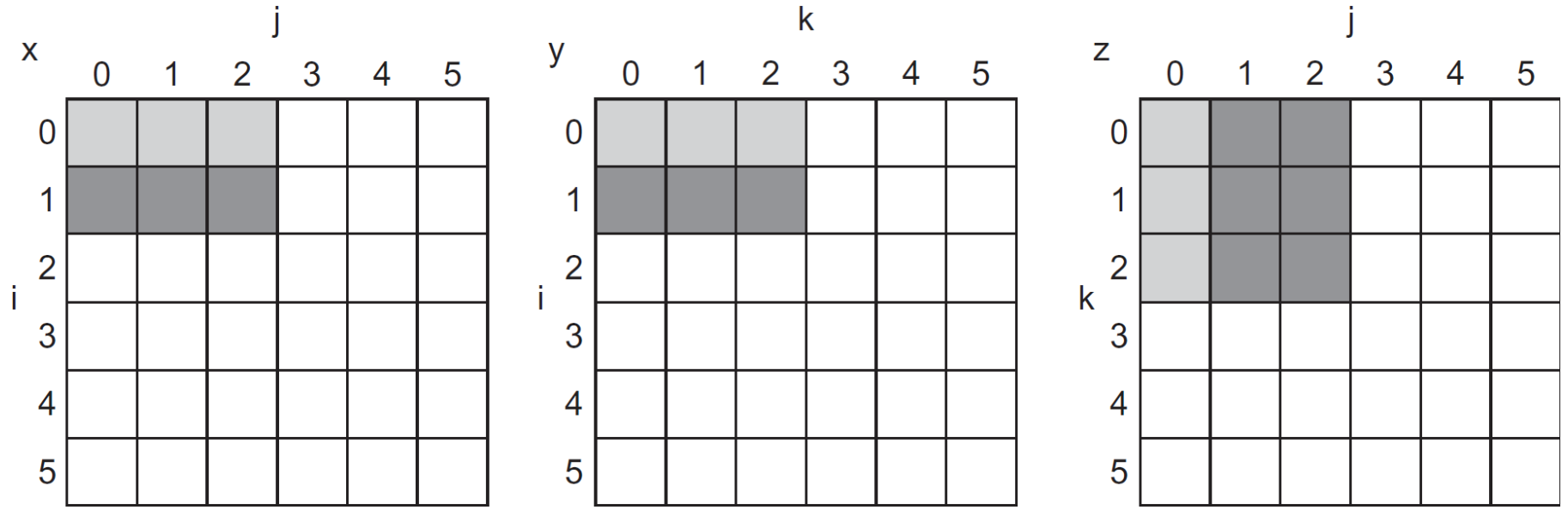
DGEMM Access Pattern

- C, A, and B arrays

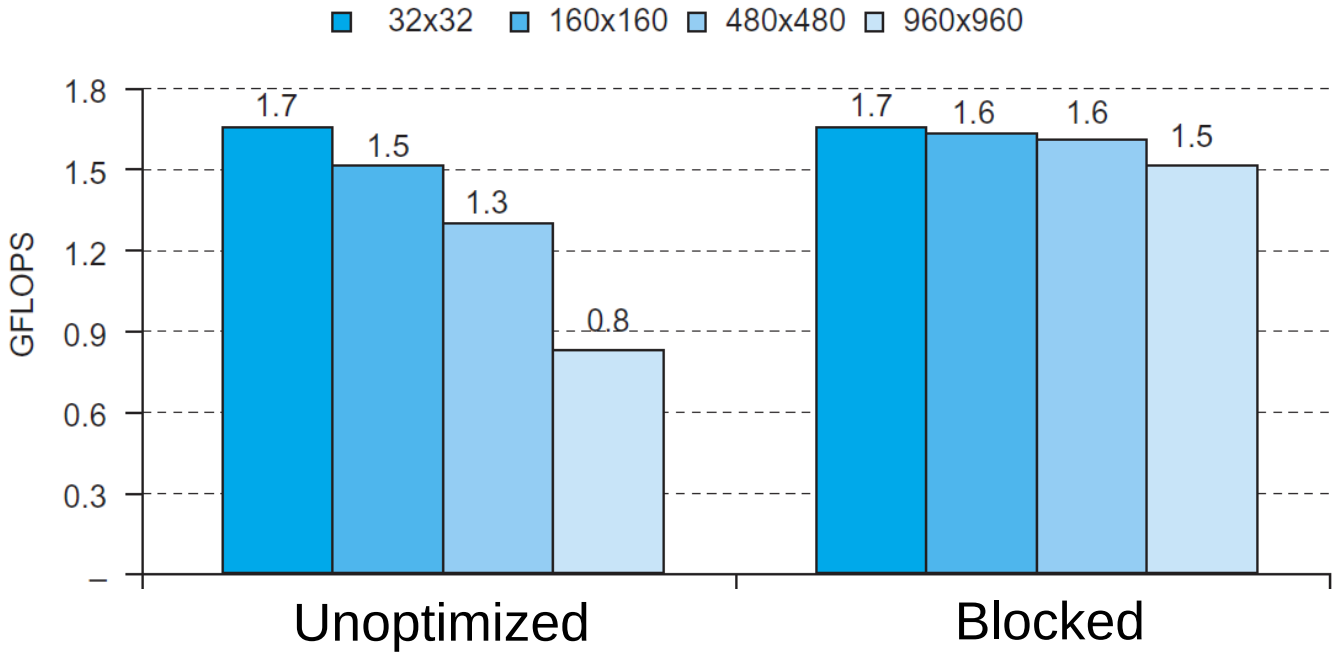


Cache Blocked DGEMM

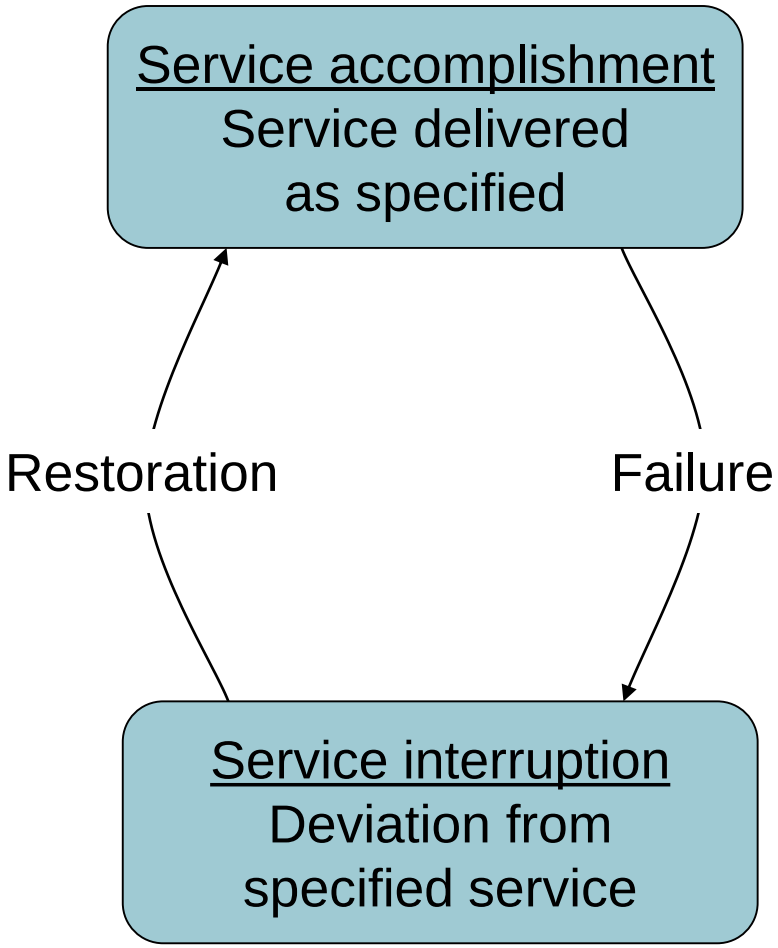
```
1 #define BLOCKSIZE 32
2 void do_block (int n, int si, int sj, int sk, double *A, double
3 *B, double *C)
4 {
5  for (int i = si; i < si+BLOCKSIZE; ++i)
6   for (int j = sj; j < sj+BLOCKSIZE; ++j)
7   {
8    double cij = C[i+j*n];/* cij = C[i][j] */
9    for( int k = sk; k < sk+BLOCKSIZE; k++ )
10    cij += A[i+k*n] * B[k+j*n];/* cij+=A[i][k]*B[k][j] */
11   C[i+j*n] = cij;/* C[i][j] = cij */
12  }
13 }
14 void dgemm (int n, double* A, double* B, double* C)
15 {
16  for ( int sj = 0; sj < n; sj += BLOCKSIZE )
17   for ( int si = 0; si < n; si += BLOCKSIZE )
18    for ( int sk = 0; sk < n; sk += BLOCKSIZE )
19     do_block(n, si, sj, sk, A, B, C);
20 }
```



Blocked DGEMM Access Pattern



Dependability



- Fault: failure of a component
  - May or may not lead to system failure

Dependability Measures

- Reliability: mean time to failure (MTTF)
- Service interruption: mean time to repair (MTTR)
- Mean time between failures
  - MTBF = MTTF + MTTR
- A related term is annual failure rate (AFR), which is simply the percentage of devices that would be expected to fail in a year for a given MTTF.
- When MTTF gets large it can be misleading, while AFR leads to better intuition

Example: MTTF vs. AFR of Disks

- Some disks today are quoted to have a 1,000,000-hour MTTF.
  - 1,000,000 hours is 1,000,000/(365 × 24) = 114 years
  - It seem they practically never fail
- Warehouse scale computers might have 50,000 servers. Assume each server has 2 disks. Use AFR to calculate how many disks we would expect to fail per year.
- One year is 365 × 24 = 8760 hours. A 1,000,000-hour MTTF means an AFR of 8760/1,000,000 = 0.876%. With 100,000 disks, we would expect 876 disks to fail per year, or on average more than 2 disk failures per day!

可用性

- Availability = MTTF / (MTTF + MTTR)
- Improving Availability
  - Increase MTTF: fault avoidance, fault tolerance, fault forecasting
  - Reduce MTTR: improved tools and processes for diagnosis and repair
- One shorthand is to quote the number of “nines of availability” per year

  One nine: 90%= > 36.5 days of repair/year

  Two nines: 99%= > 3.65 days of repair/year

  Three nines: 99.9%= > 526 minutes of repair/year

  Four nines: 99.99%= > 52.6 minutes of repair/year

  Five nines: 99.999%= > 5.26 minutes of repair/year

The Hamming SEC Code

- Hamming distance
  - Number of bits that are different between two bit patterns
- Minimum distance = 2 provides single bit error detection
  - E.g. parity code
- Minimum distance = 3 provides single error correction, 2 bit error detection

奇偶校验法的特点

- 奇偶校验码的Hamming距离d=2。
  - 两个数若有奇数位不同，则它们相应的校验位就不同；若有偶数位不同，则虽校验位相同，但至少有两位数据位不同。因而任意两个编码之间至少有两位不同。
- 只能发现奇数位出错，不能发现偶数位出错。
- 不能确定发生错误的位置，不具有纠错能力。
- 开销小，适用于校验1字节长的代码，故常被用于存储器读写检查或按字节传输过程中的数据校验，因为1字节长的代码发生错误时，1位出错的概率较大，2位以上出错则很少，所以可用奇偶校验法

Encoding SEC

- To calculate Hamming code:
  - Number bits from 1 on the left
  - All bit positions that are a power 2 are parity bits
  - Each parity bit checks certain data bits:

Bit position		1	2	3	4	5	6	7	8	9	10	11	12
Encoded date bits		p1	p2	d1	p4	d2	d3	d4	p8	d5	d6	d7	d8
Parity bit coverate	p1	X		X		X		X		X		X	
	p2		X	X			X	X			X	X	
	p4				X	X	X	X					X
	p8								X	X	X	X	X

Encoding SEC

- Bit 1 (0001_2) checks bits (1,3,5,7,9,11,…), which are bits where rightmost bit of address is 1 (0001_2, 0011_2, 0101_2, 0111_2, 1001_2, 1011_2,…).
- Bit 2 (0010_2) checks bits (2,3,6,7,10,11,14,15,…), which are the bits where the second bit to the right in the address is 1.
- Bit 4 (0100_2) checks bits (4–7, 12–15, 20–23,…), which are the bits where the third bit to the right in the address is 1.
- Bit 8 (1000_2) checks bits (8–15, 24–31, 40–47,…), which are the bits where the fourth bit to the right in the address is 1.

Note that each data bit is covered by two or more parity bits.

Bit position		1	2	3	4	5	6	7	8	9	10	11	12
Encoded date bits		p1	p2	d1	p4	d2	d3	d4	p8	d5	d6	d7	d8
Parity bit coverate	p1	X		X		X		X		X		X	
	p2		X	X			X	X			X	X	
	p4				X	X	X	X					X
	p8								X	X	X	X	X

Example: SEC

- Assume one byte data value is 10011010_2. First show the Hamming ECC code for that byte, and then invert bit 10 and show that the ECC code finds and corrects the single bit error.
- Leaving spaces for parity bits, the 12 bit pattern is _ _ 1_0 0 1_ 1 0 1 0.

Bit position		1	2	3	4	5	6	7	8	9	10	11	12
Encoded date bits		0	1	1	1	0	0	1	0	1	0	1	0
Parity bit coverate	p1	X		X		X		X		X		X	
	p2		X	X			X	X			X	X	
	p4				X	X	X	X					X
	p8								X	X	X	X	X

Example: SEC

- The final code word is 011100101010. Inverting bit 10 changes it to 011100101110.
- Parity bits 2 and 8 are incorrect. As 2 + 8 = 10, bit 10 must be wrong.

Bit position		1	2	3	4	5	6	7	8	9	10	11	12
Encoded date bits		0	1	1	1	0	0	1	0	1	1	1	0
Parity bit coverate	p1	X		X		X		X		X		X	
	p2		X	X			X	X			X	X	
	p4				X	X	X	X					X
	p8								X	X	X	X	X

SEC/DED

- We can make the minimum Hamming distance in a code be 4. This means we can correct single bit errors and detect double bit errors. The idea is to add a parity bit that is calculated over the whole word.
- Then the algorithm is to calculate parity over the ECC groups (H) as before plus one more over the whole group (P). There are four cases:
  - 1. H is even and P is even, no error.
  - 2. H is odd and P is odd, a correctable single error
  - 3. H is even and P is odd, a single error occurred in P bit
  - 4. H is odd and p4 is even, a double error occurred

校验码的位数

- 假定数据位数为d，校验码为p位。p位校验码所能表示的状态最多是2^p，每种状态可用来说明一种出错情况。
- 若只有一位错，则结果可能是：
  - 数据中某一位错 (d种可能)
  - 校验码中有一位错 (p种可能)
  - 无错 (1 种可能)

  } **d+p+1种情况**
- 则n和k必须满足下列关系：$2^p \geq 1+d+p$
- 例如，当数据有8位时，d=8，则$2^p \geq p+9$，所以p=4。类似地，数据长度为16时，p=5；数据长度为32时，p=6；数据长度为64时，p=7
- Single Error Correcting / Double Error Detecting (SEC/DED) is common in memory for servers today. Conveniently, eight byte data blocks can get SEC/DED with one more byte, which is why many DIMMs are 72 bits wide

Virtual Machines

- Host computer emulates guest operating system and machine resources
  - Improved isolation of multiple guests
  - Avoids security and reliability problems
  - Aids sharing of resources
- Virtualization has some performance impact
  - Feasible with modern high-performance comptuers
- Examples
  - IBM VM/370 (1970s technology!)
  - VMWare
  - Microsoft Virtual PC

Virtual Machine Monitor

- Maps virtual resources to physical resources
  - Memory, I/O devices, CPUs
- Guest code runs on native machine in user mode
  - Traps to VMM on privileged instructions and access to protected resources
- Guest OS may be different from host OS
- VMM handles real I/O devices
  - Emulates generic virtual I/O devices for guest

Example: Timer Virtualization

- In native machine, on timer interrupt
  - OS suspends current process, handles interrupt, selects and resumes next process
- With Virtual Machine Monitor
  - VMM suspends current VM, handles interrupt, selects and resumes next VM
- If a VM requires timer interrupts
  - VMM emulates a virtual timer
  - Emulates interrupt for VM when physical timer interrupt occurs

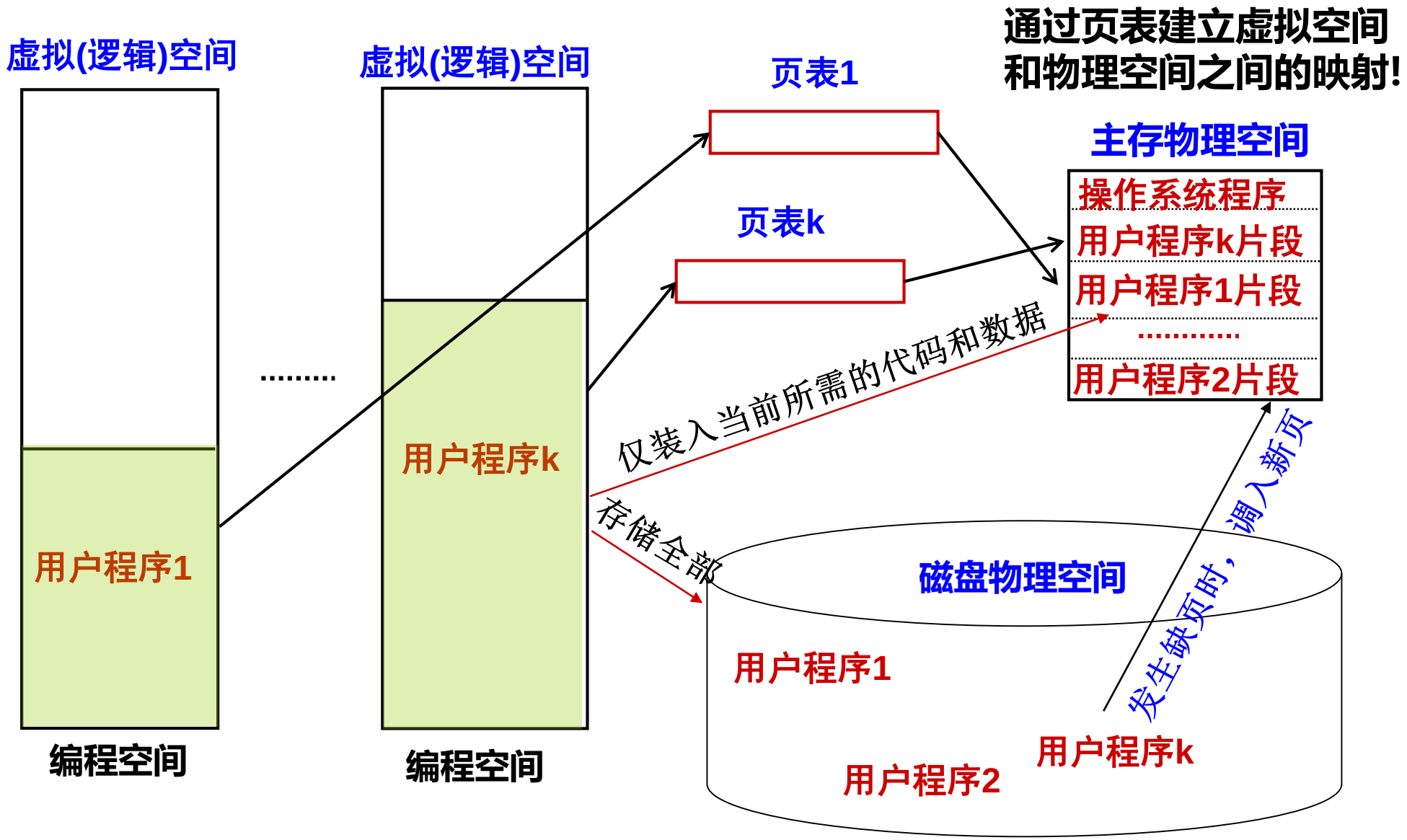
Instruction Set Support

- User and System modes
- Privileged instructions only available in system mode
  - Trap to system if executed in user mode
- All physical resources only accessible using privileged instructions
  - Including page tables, interrupt controls, I/O registers
- Renaissance of virtualization support
  - Current ISAs (e.g., x86) adapting

Virtual Memory

- Use main memory as a “cache” for secondary (disk) storage
  - Managed jointly by CPU hardware and the operating system (OS)
- Programs share main memory
  - Each gets a private virtual address space holding its frequently used code and data
  - Protected from other programs
- CPU and OS translate virtual addresses to physical addresses
  - VM “block” is called a page
  - VM translation “miss” is called a page fault

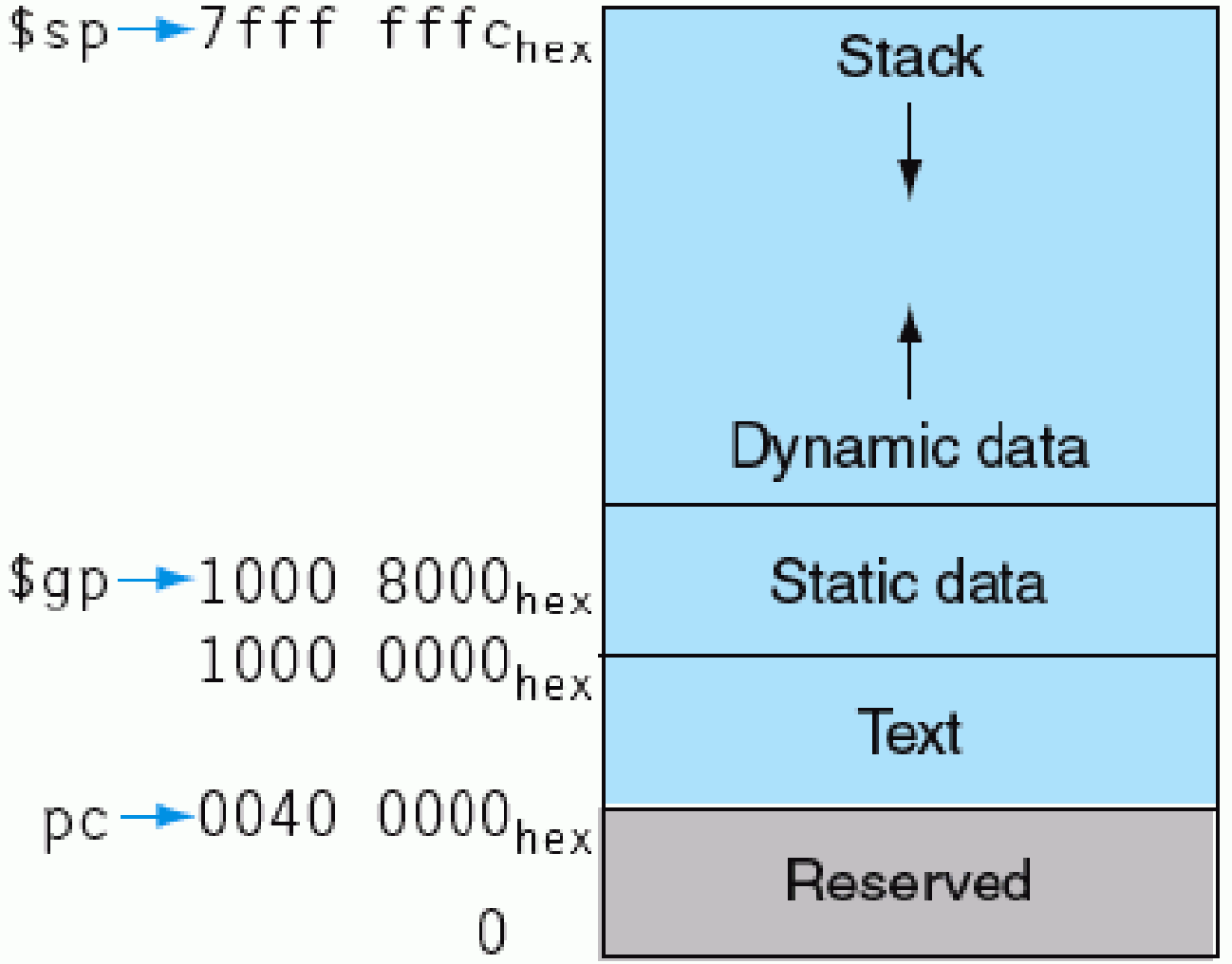
虚拟存储


通过页表建立虚拟空间
和物理空间之间的映射!
虚拟(逻辑)空间
虚拟(逻辑)空间
页表1
页表k
主存物理空间
操作系统程序
用户程序k片段
用户程序1片段
.....
用户程序2片段
用户程序1
.....
用户程序k
仅装入当前所需的代码和数据
存储全部
磁盘物理空间
用户程序1
用户程序2
用户程序k
发生缺页时，调入新页
编程空间
编程空间

MIPS程序和数据的存储器分配

问题：你知道一个程序在“编辑、编译、汇编、链接、装入”过程中的哪个环节确定了每条指令及其操作数的虚拟地址吗？

链接时确定虚拟地址；装入时生成页表以建立虚拟地址与物理地址之间的映射！

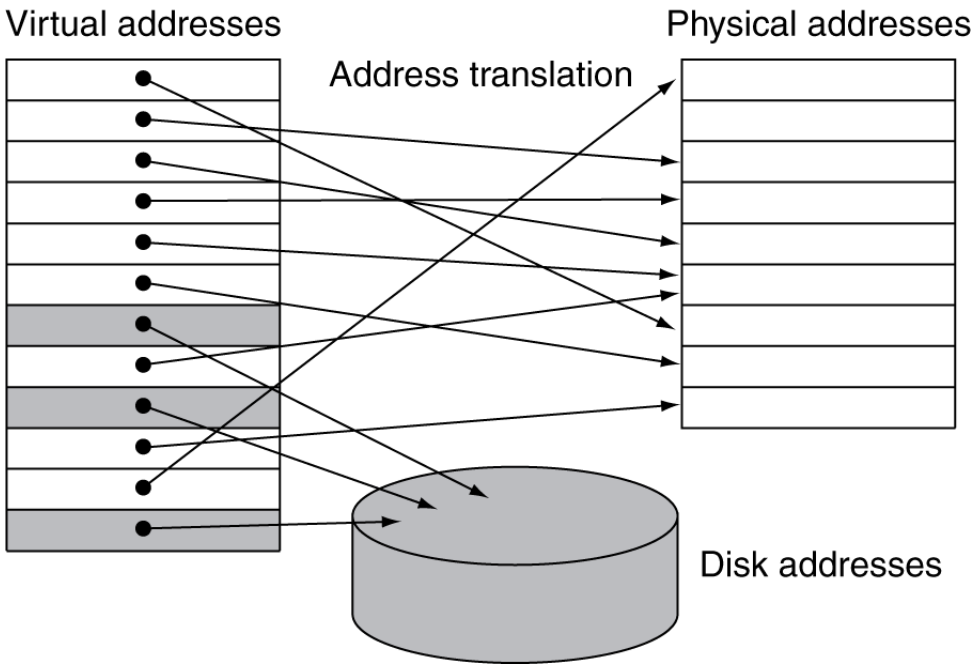


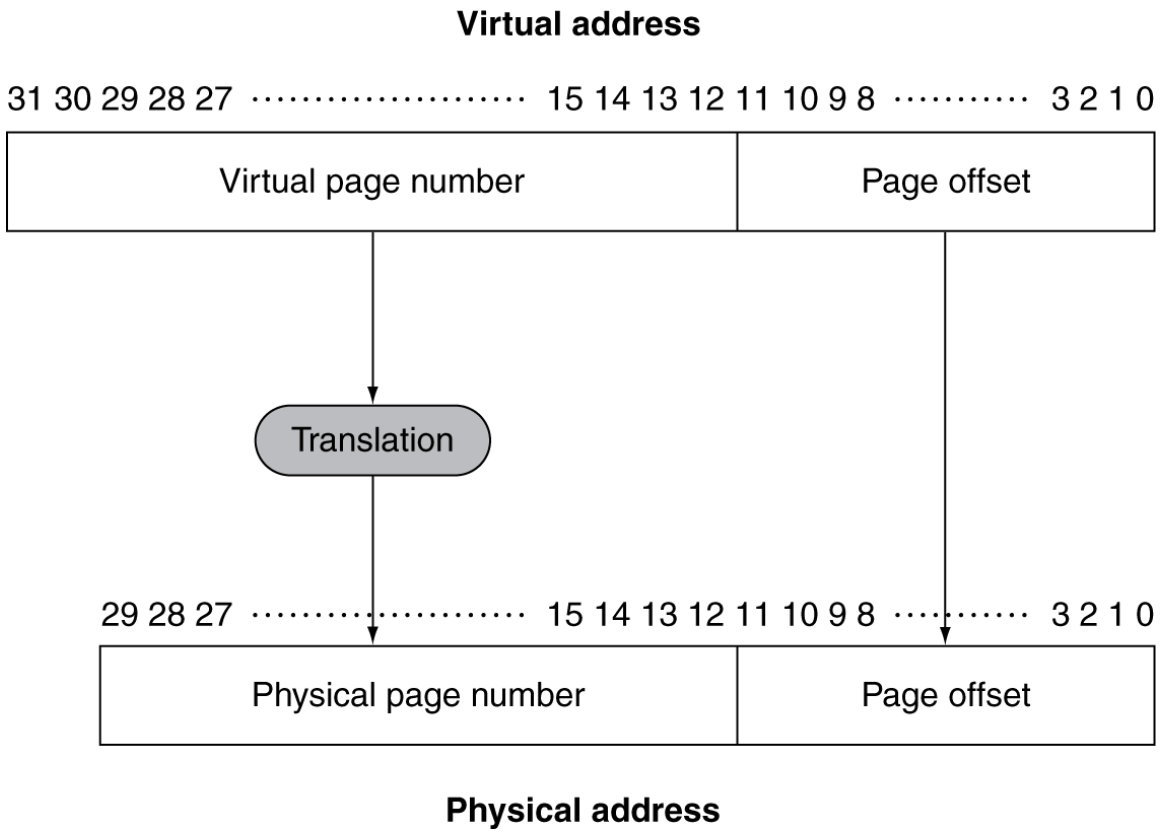
每个用户程序都有相同的虚拟地址空间！

这就是每个进程的虚拟（逻辑）地址空间！

Address Translation

- Fixed-size pages (e.g., 4K)



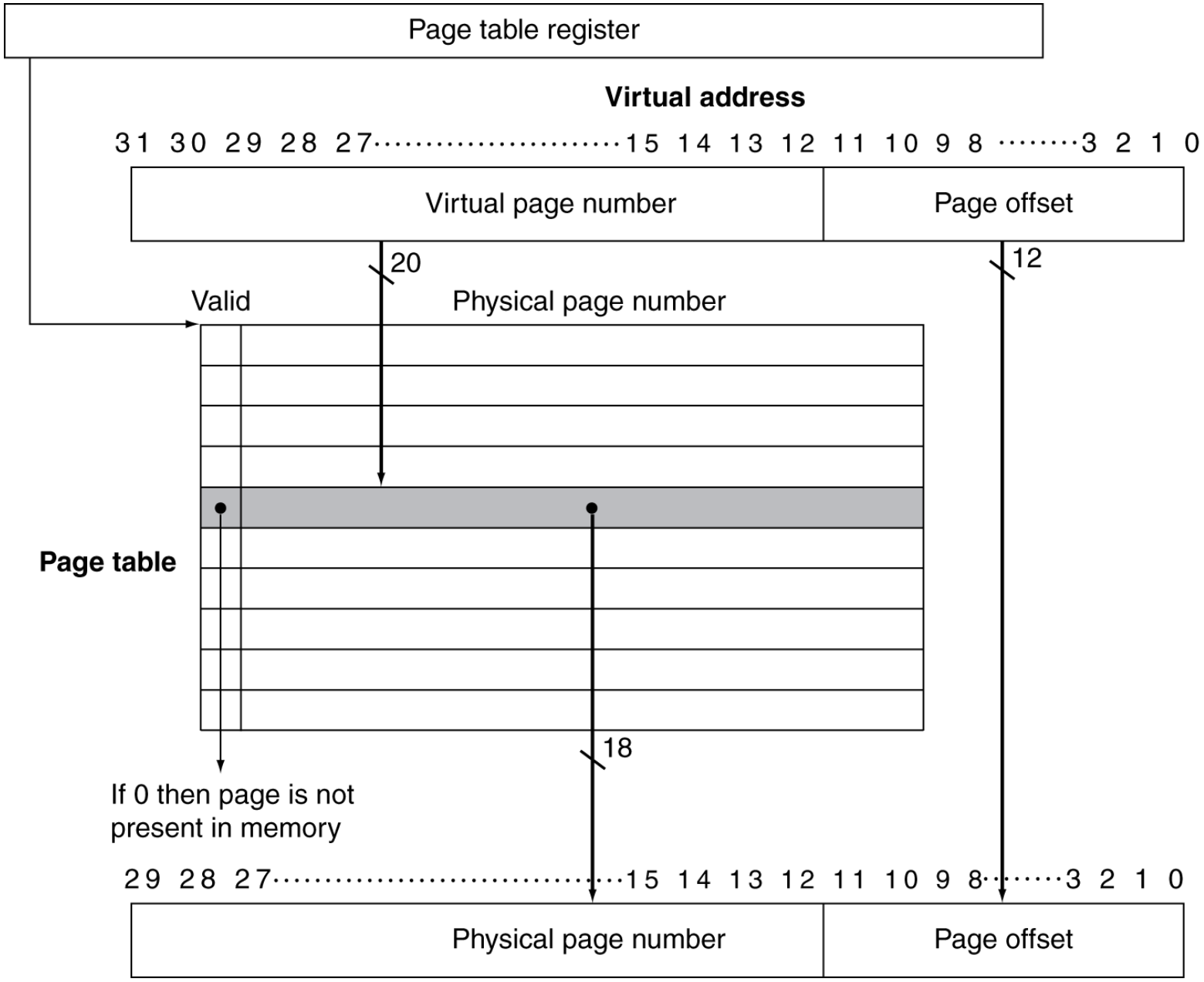


Page Fault Penalty

- On page fault, the page must be fetched from disk
  - Takes millions of clock cycles
  - Handled by OS code
- Try to minimize page fault rate
  - Fully associative placement
  - Smart replacement algorithms

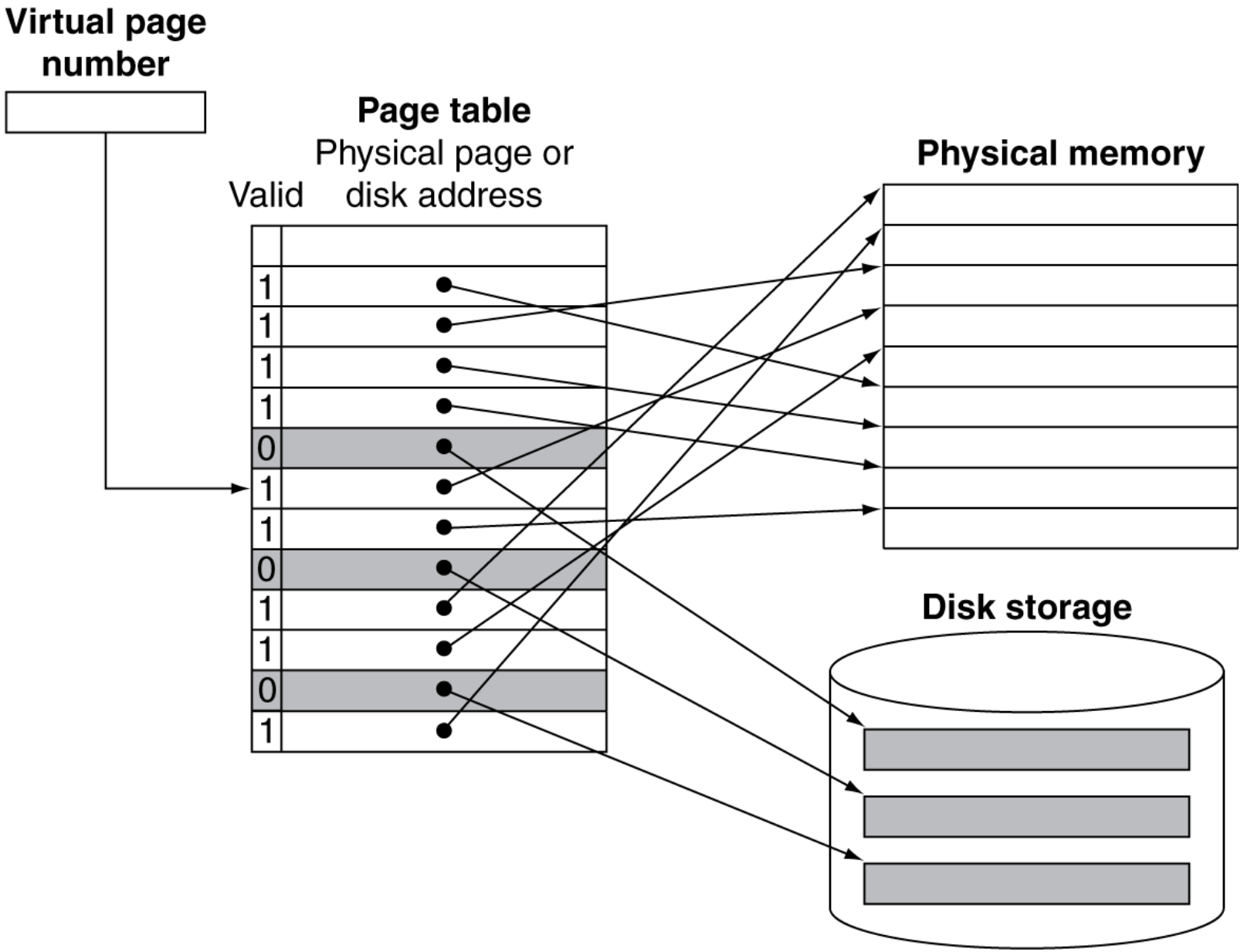
Page Tables

- Stores placement information
  - Array of page table entries, indexed by virtual page number
  - Page table register in CPU points to page table in physical memory
- If page is present in memory
  - Page translation entry (PTE) stores the physical page number
  - Plus other status bits (referenced, dirty, …)
- If page is not present
  - PTE can refer to location in swap space on disk



Translation Using a Page Table

Mapping Pages to Storage



Replacement and Writes

- To reduce page fault rate, prefer least-recently used (LRU) replacement
  - Reference bit (aka use bit) in PTE set to 1 on access to page
  - Periodically cleared to 0 by OS
  - A page with reference bit = 0 has not been used recently
- Disk writes take millions of cycles
  - Block at once, not individual locations
  - Write through is impractical
  - Use write-back
  - Dirty bit in PTE set when page is written

Size of Page Table

- With a 32-bit virtual address, 4 KiB pages, and 4 bytes per page table entry, we can compute the total page table size: 2^{20} entries * 4 bytes = 4MiB
- What if there are hundreds of processes running, each with their own page table?

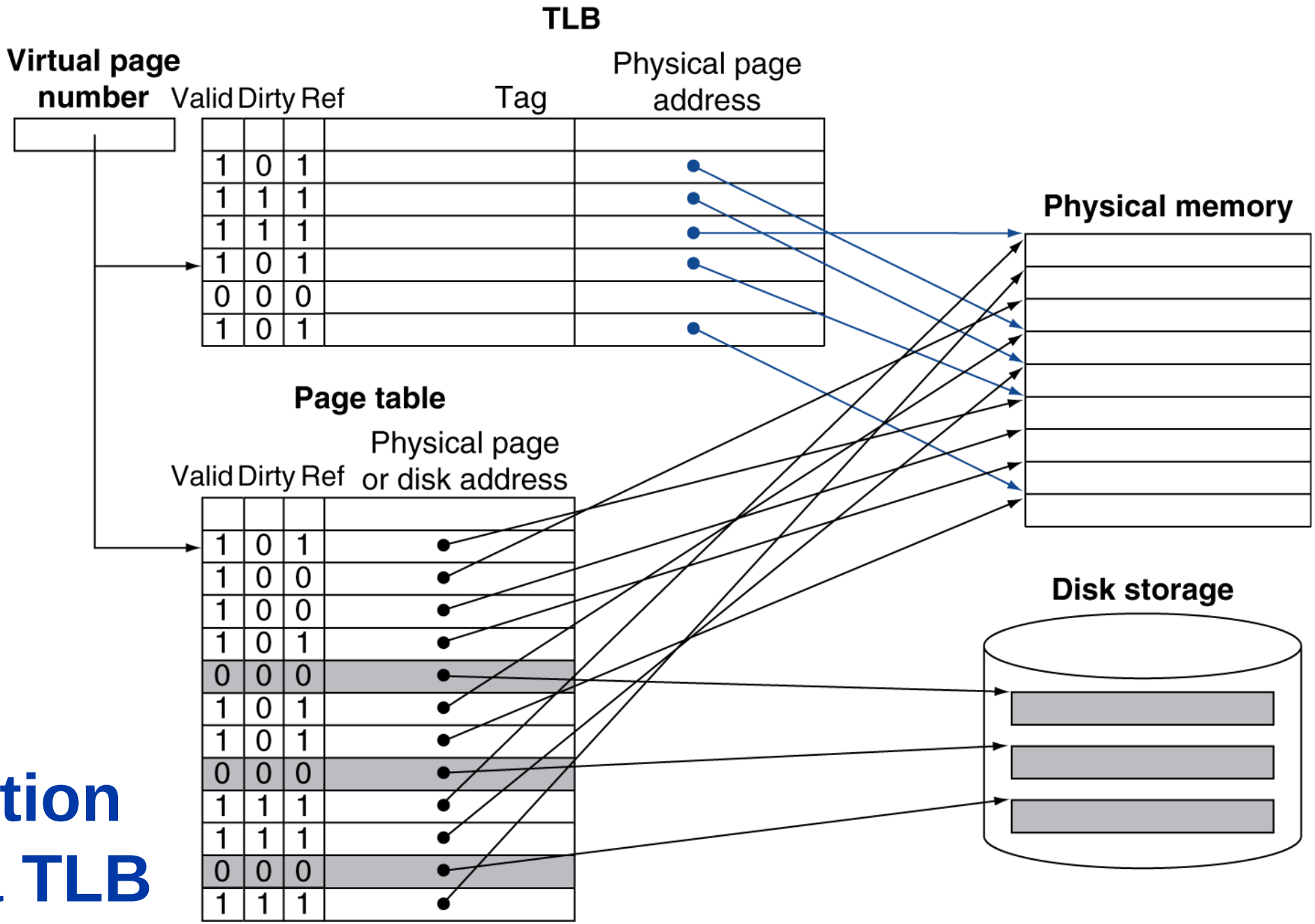
Reduce the Memory for Page Tables

- A limit register that restricts the size of the page table for a given process.
- Two page tables and two limit registers. Divide the page table and let it grow from the highest address down, as well as from the lowest address up.
- inverted page table. Apply a hashing function to the virtual address so that the page table need be only the size of the number of physical pages in main memory.
- Multiple levels of page tables
- allow the page tables to be paged

Fast Translation Using a TLB

- Address translation would appear to require extra memory references
  - One to access the PTE
  - Then the actual memory access
- But access to page tables has good locality
  - So use a fast cache of PTEs within the CPU
  - Called a Translation Look-aside Buffer (TLB)
  - Typical: 16–512 PTEs, 0.5–1 cycle for hit, 10–100 cycles for miss, 0.01%–1% miss rate
  - Misses could be handled by hardware or software

Fast Translation Using a TLB



TLB Misses

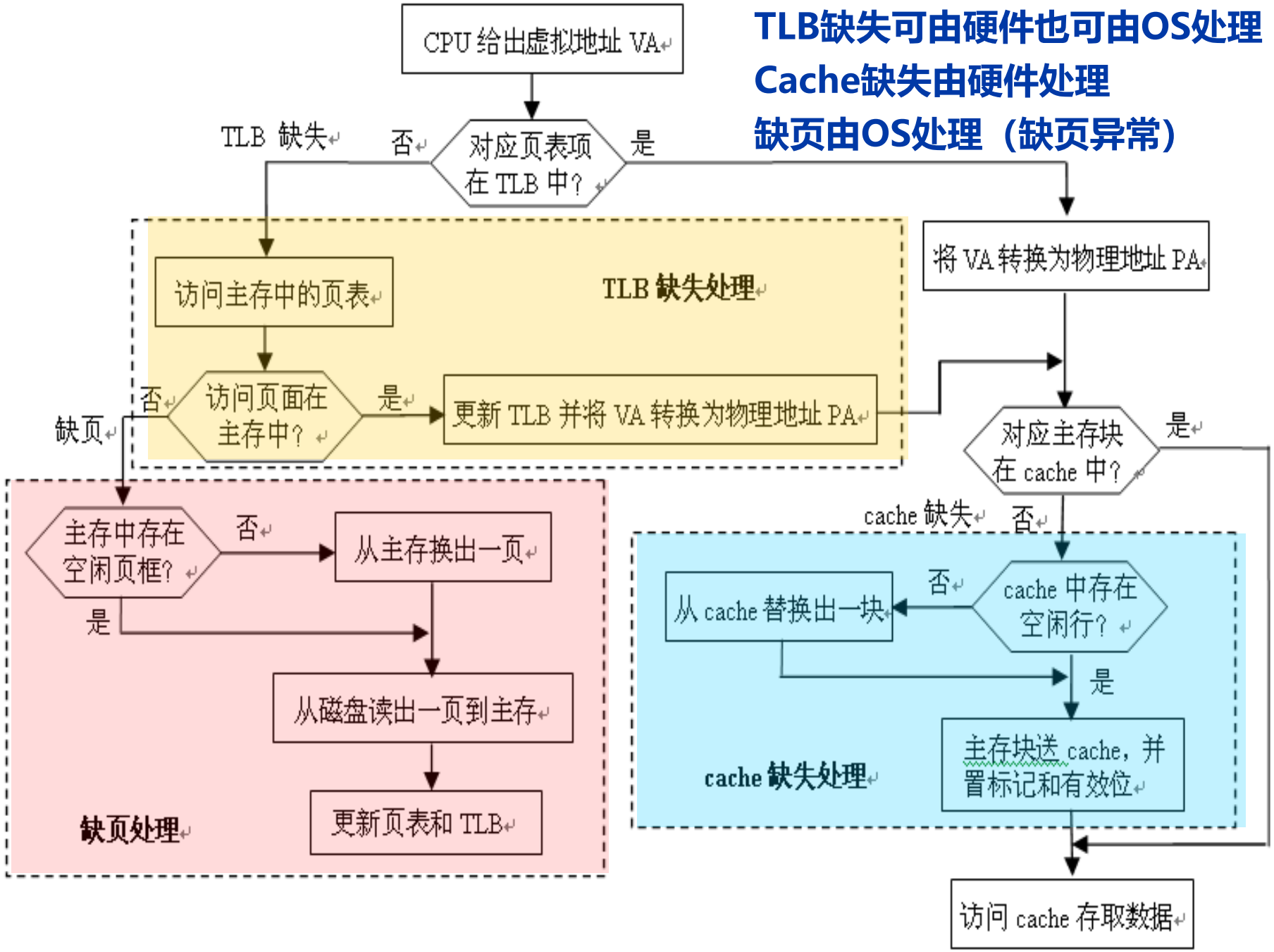
- If page is in memory
  - Load the PTE from memory and retry
  - Could be handled in hardware
    - Can get complex for more complicated page table structures
  - Or in software
    - Raise a special exception, with optimized handler
- If page is not in memory (page fault)
  - OS handles fetching the page and updating the page table
  - Then restart the faulting instruction

TLB Miss Handler

- TLB miss indicates
  - Page present, but PTE not in TLB
  - Page not preset
- Must recognize TLB miss before destination register overwritten
  - Raise exception
- Handler copies PTE from memory to TLB
  - Then restarts instruction
  - If page not present, page fault will occur

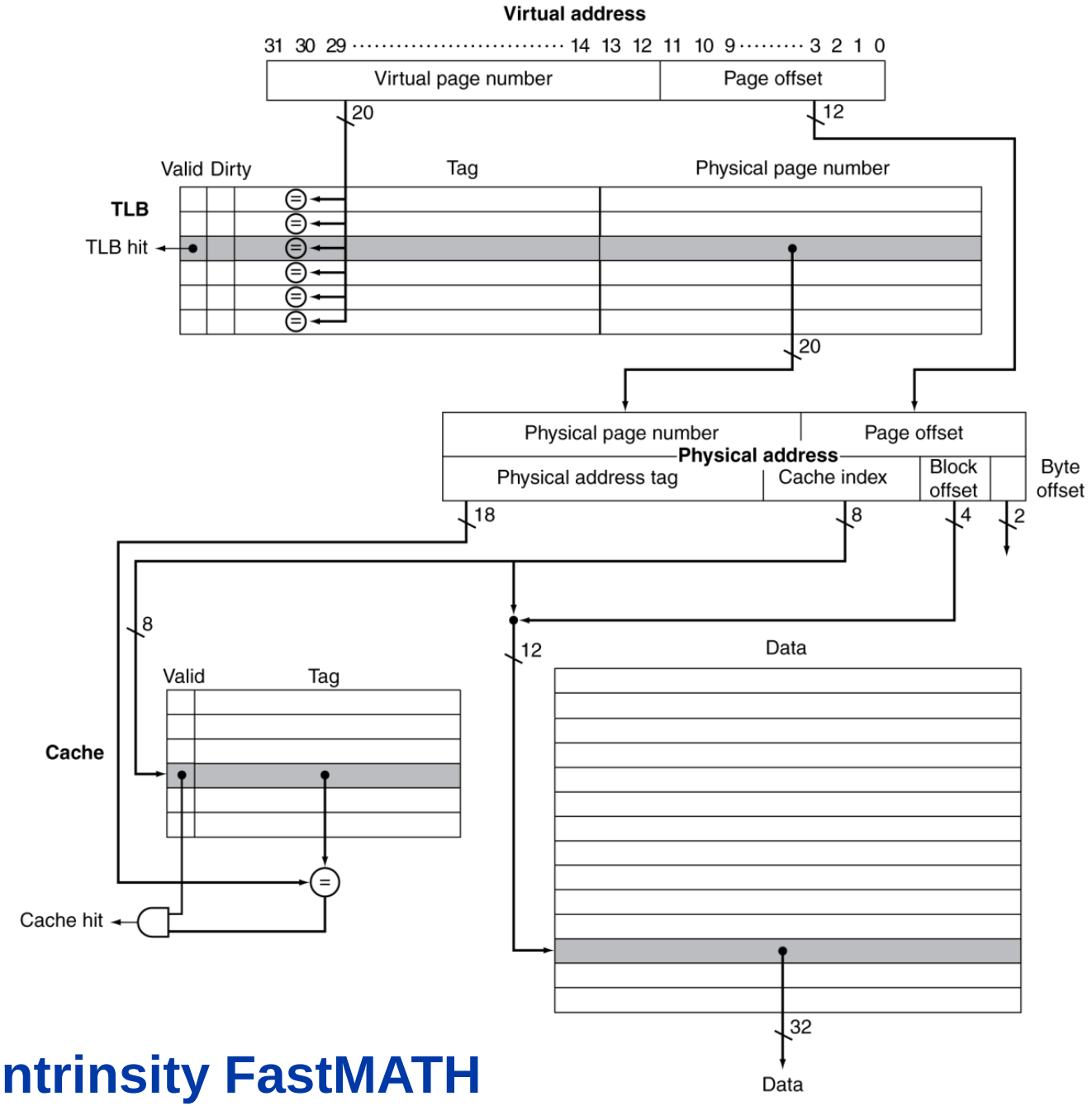
Page Fault Handler

- Use faulting virtual address to find PTE
- Locate page on disk
- Choose page to replace
  - If dirty, write to disk first
- Read page into memory and update page table
- Make process runnable again
  - Restart from faulting instruction



TLB缺失可由硬件也可由OS处理
Cache缺失由硬件处理
缺页由OS处理（缺页异常）

CPU访存过程

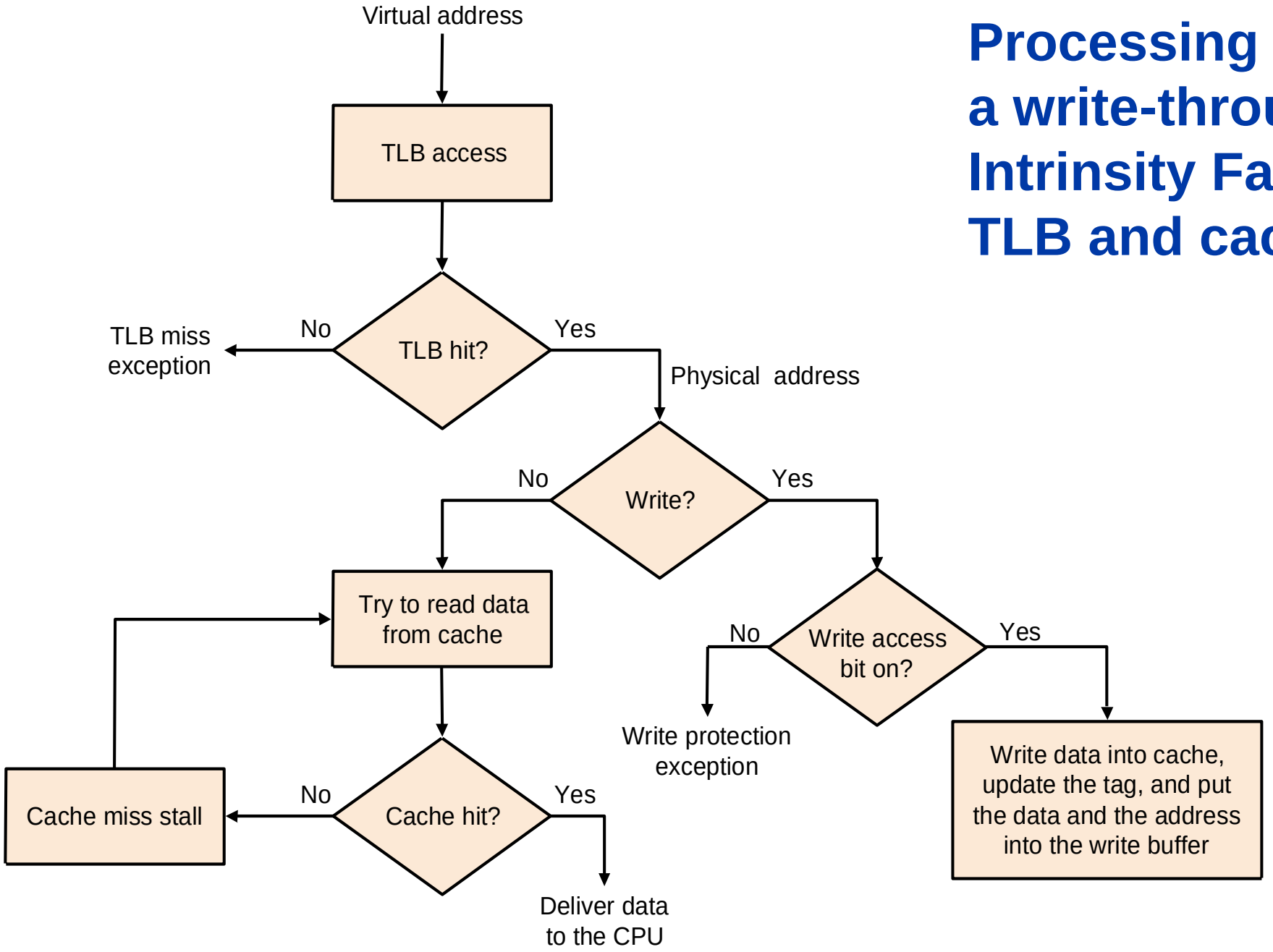


TLB and Cache Interaction

- If cache tag uses physical address
  - Need to translate before cache lookup
- Alternative: use virtual address tag
  - Complications due to aliasing
  - Different virtual addresses for shared physical address

Intrinsity FastMATH

Processing a read or a write-through in the Intrinsity FastMATH TLB and cache



三种不同缺失的组合

- 最好的情况：hit、hit、hit，不需要访问主存！
- 在下表组合中，最好的情况：hit、hit、miss和miss、hit、hit，访存1次
- 其次是：miss、hit、miss，不需访问磁盘、但访存至少2次
- 最坏的情况是：miss、miss、miss，需访问磁盘1次、访存至少2次

TLB	Page table	Cache	Possible? If so, under what circumstance?
hit	hit	miss	可能，TLB命中则页表一定命中，但实际上不会查页表
miss	hit	hit	可能，TLB缺失但页表命中，信息在主存，就可能在Cache
miss	hit	miss	可能，TLB缺失但页表命中，信息在主存，但可能不在Cache
miss	miss	miss	可能，TLB缺失页表缺失，信息不在主存，一定也不在Cache
hit	miss	miss	不可能，页表缺失，信息不在主存，TLB中一定没有该页表项
hit	miss	hit	同上
miss	miss	hit	不可能，页表缺失，信息不在主存，Cache中一定也无该信息

Memory Protection

- Different tasks can share parts of their virtual address spaces
  - But need to protect against errant access
  - Requires OS assistance
- Hardware support for OS protection
  - Privileged supervisor mode (aka kernel mode)
  - Privileged instructions
  - Page tables and other state information only accessible in supervisor mode
  - System call exception (e.g., syscall in MIPS)

Understanding Program Performance

- a program will run very slowly if it continuously swaps pages between memory and disk, called thrashing
- Reexamine your algorithm and data structures to see if you can change the locality and thereby reduce the number of pages that your program uses simultaneously. This set of popular pages is informally called the working set.

Understanding Program Performance

- A more common problem is TLB misses.
  - Since a TLB might handle only 32–64 page entries at a time
  - A program could easily see a high TLB miss rate, as the processor may access less than a quarter mebibyte directly: 64 × 4KiB = 0.25MiB.
- Most computer architectures now support variable page sizes. For example, in addition to the standard 4KiB page, MIPS hardware supports 16KiB, 64KiB, 256KiB, 1MiB, 4MiB, 16MiB, 64MiB, and 256MiB pages.
- If a program uses large page sizes, it can access more memory directly without TLB misses.

The Memory Hierarchy

- Common principles apply at all levels of the memory hierarchy
  - Based on notions of caching
- At each level in the hierarchy
  - Block placement
  - Finding a block
  - Replacement on a miss
  - Write policy

Block Placement

- Determined by associativity
  - Direct mapped (1-way associative)
    - One choice for placement
  - n-way set associative
    - n choices within a set
  - Fully associative
    - Any location
- Higher associativity reduces miss rate
  - Increases complexity, cost, and access time

Finding a Block

- Hardware caches
  - Reduce comparisons to reduce cost
- Virtual memory
  - Full table lookup makes full associativity feasible
  - Benefit in reduced miss rate

Associativity	Location method	Tag comparisons
Direct mapped	Index	1
n-way set associative	Set index, then search entries within the set	n
Fully associative	Search all entries	#entries
	Full lookup table	0

Replacement

- Choice of entry to replace on a miss
  - Least recently used (LRU)
    - Complex and costly hardware for high associativity
  - Random
    - Close to LRU, easier to implement
- Virtual memory
  - LRU approximation with hardware support

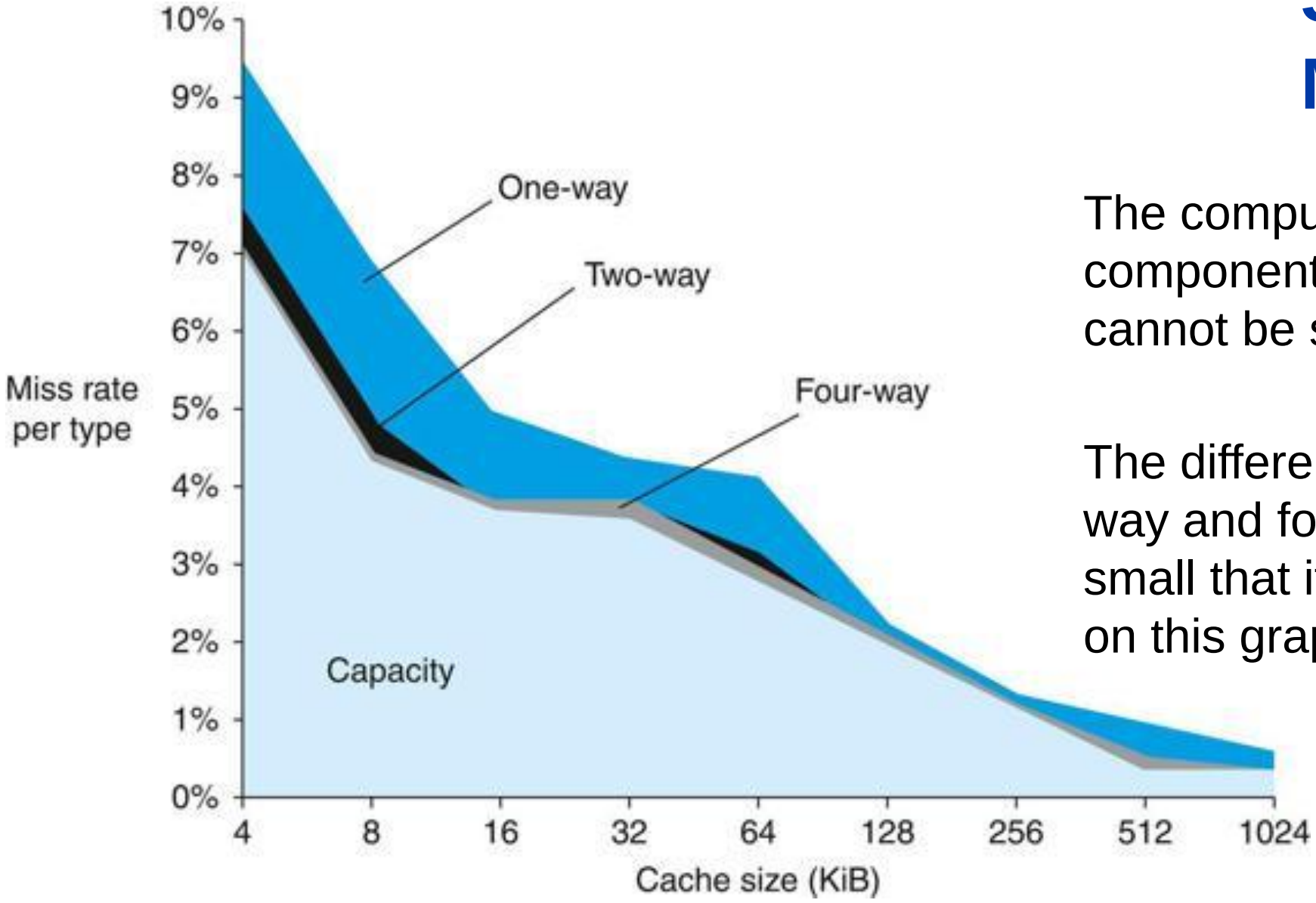
Write Policy

- Write-through
  - Update both upper and lower levels
  - Simplifies replacement, but may require write buffer
- Write-back
  - Update upper level only
  - Update lower level when block is replaced
  - Need to keep more state
- Virtual memory
  - Only write-back is feasible, given disk write latency

Sources of Misses

- Compulsory misses (aka cold start misses)
  - First access to a block
- Capacity misses
  - Due to finite cache size
  - A replaced block is later accessed again
- Conflict misses (aka collision misses)
  - In a non-fully associative cache
  - Due to competition for entries in a set
  - Would not occur in a fully associative cache of the same total size

Sources of Misses



The compulsory miss component is 0.006% and cannot be seen in this graph

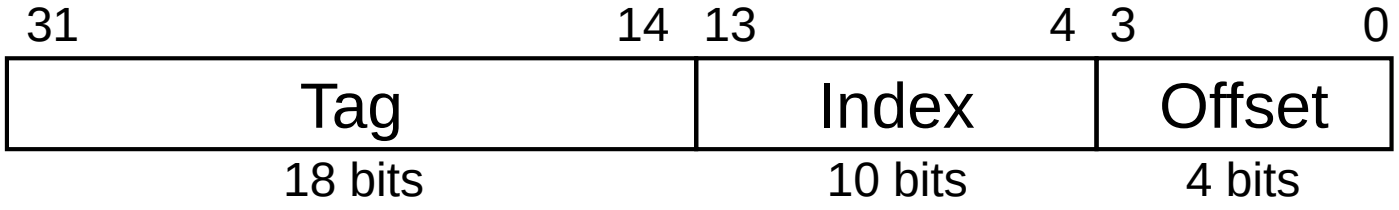
The difference betwee eight way and four-way is so small that it is difficult to see on this graph

Cache Design Trade-offs

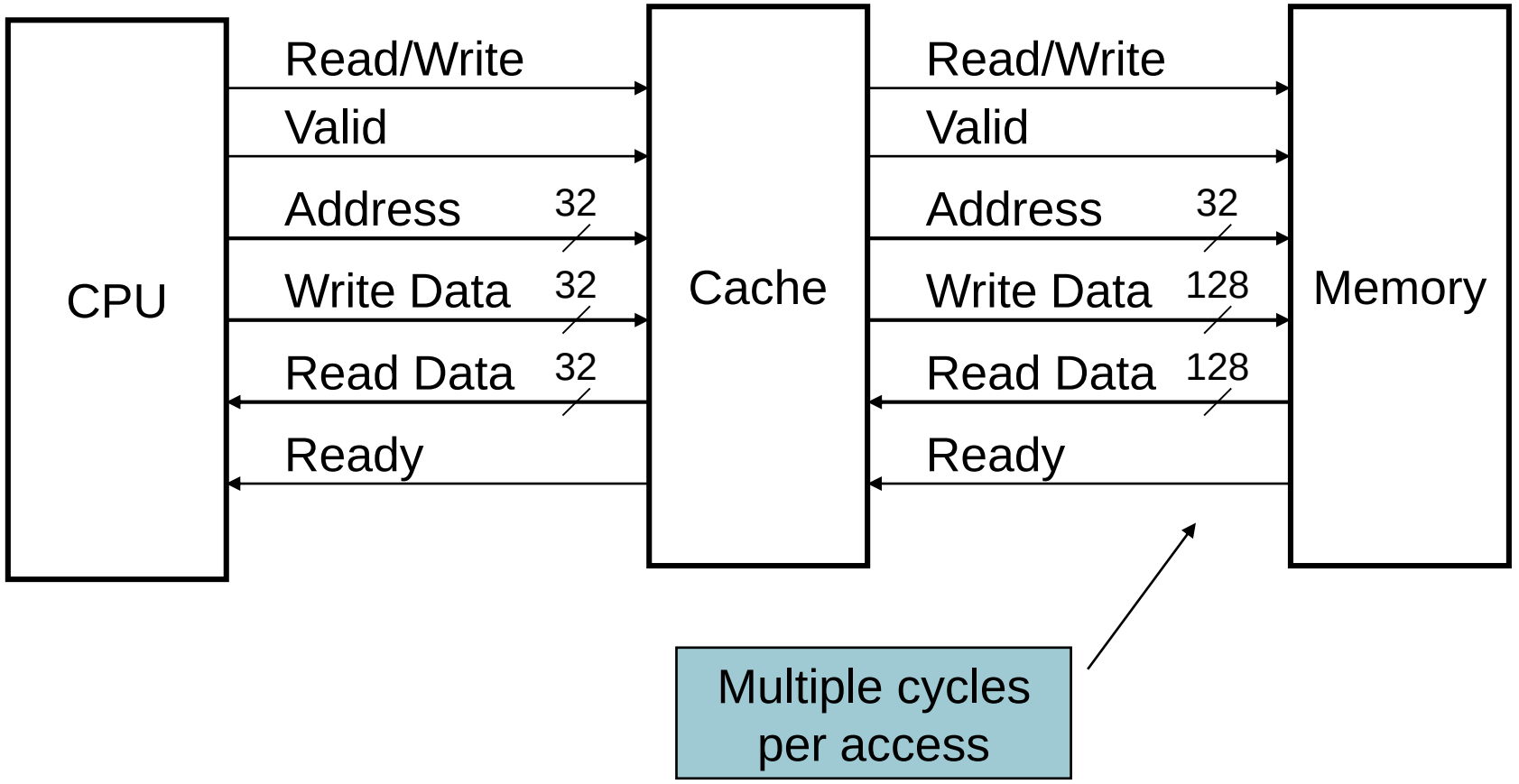
Design change	Effect on miss rate	Negative performance effect
Increase cache size	Decrease capacity misses	May increase access time
Increase associativity	Decrease conflict misses	May increase access time
Increase block size	Decrease compulsory misses	Increases miss penalty. For very large block size, may increase miss rate due to pollution.

Cache Control

- Example cache characteristics
  - Direct-mapped, write-back, write allocate
  - Block size: 4 words (16 bytes)
  - Cache size: 16 KB (1024 blocks)
  - 32-bit byte addresses
  - Valid bit and dirty bit per block
  - Blocking cache
    - CPU waits until access is complete

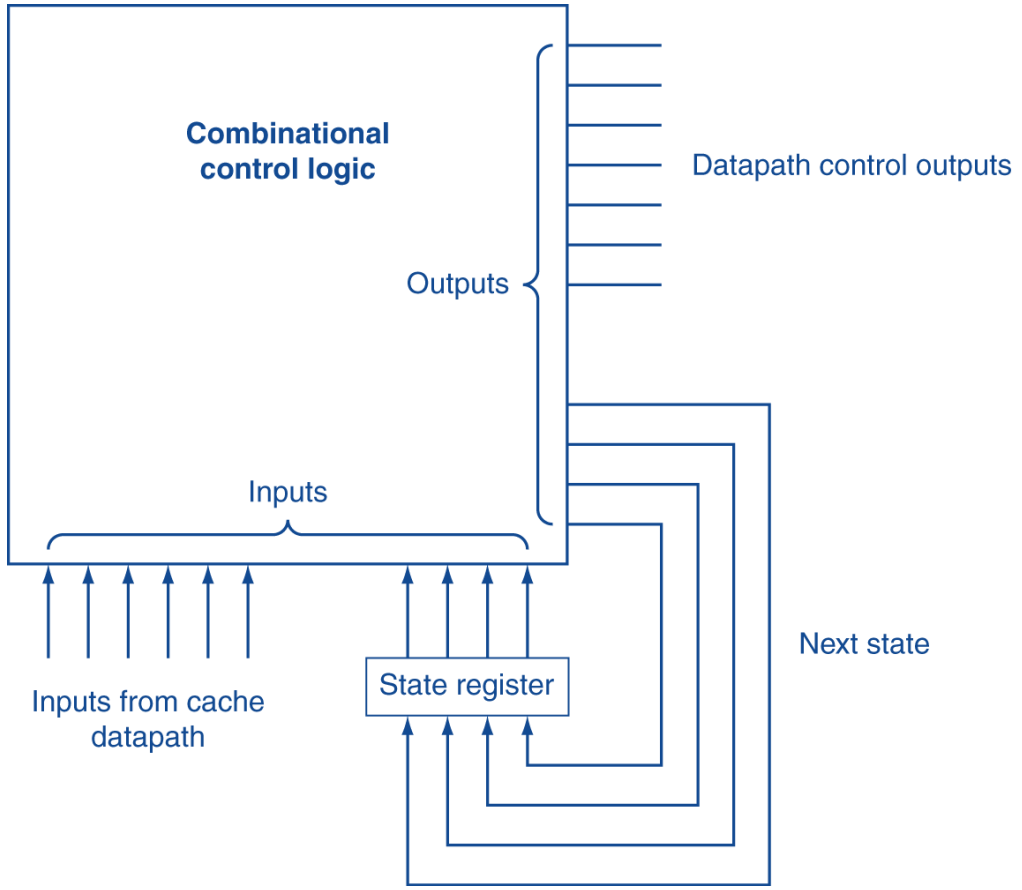


Interface Signals

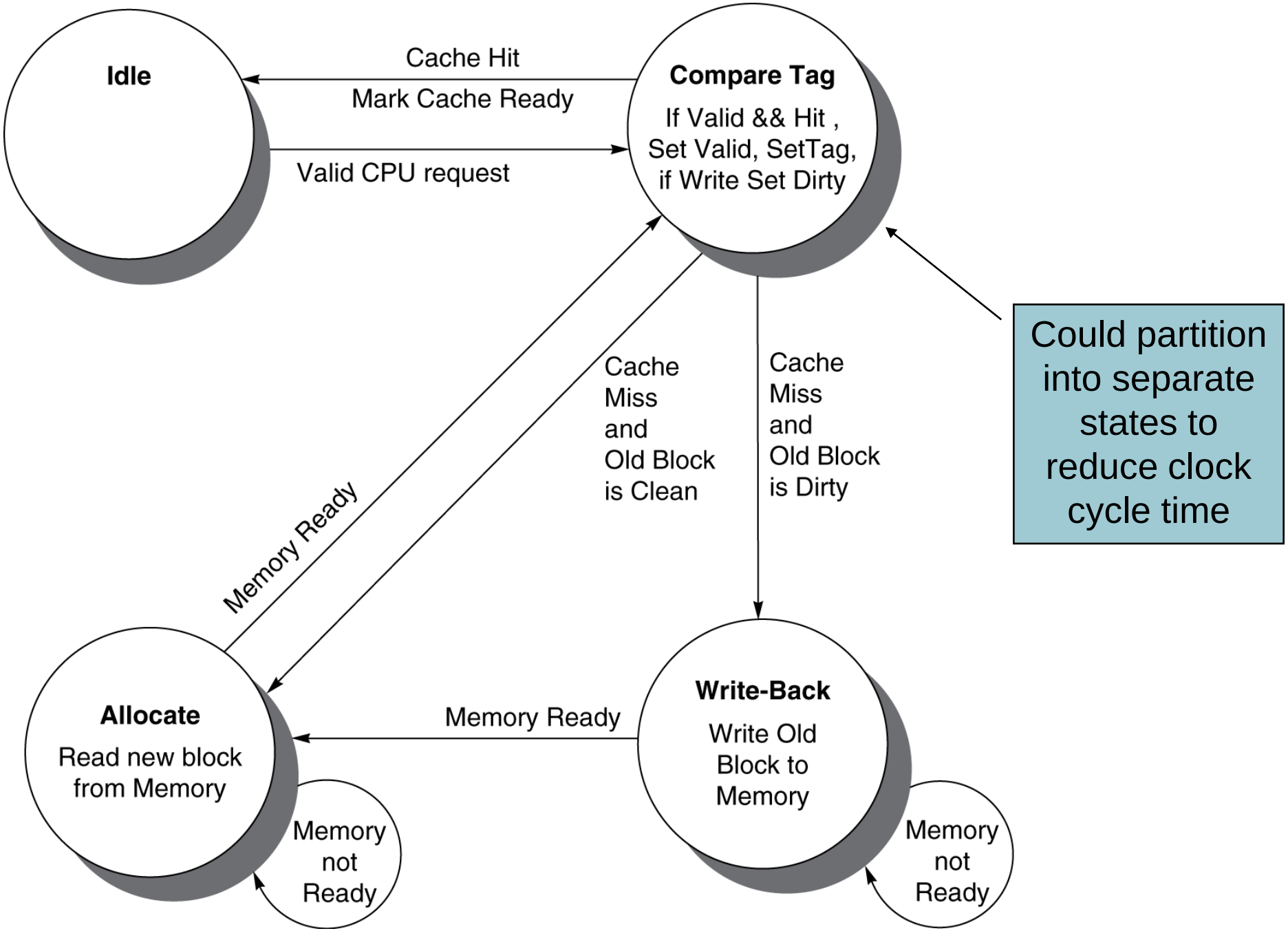


Finite State Machines

- Use an FSM to sequence control steps
- Set of states, transition on each clock edge
  - State values are binary encoded
  - Current state stored in a register
  - Next state
    = f_n (current state,
    current inputs)
- Control output signals
  = f_o (current state)



Cache Controller FSM



Could partition into separate states to reduce clock cycle time

Cache Coherence Problem

- Suppose two CPU cores share a physical address space
  - Write-through caches

Time step	Event	CPU A's cache	CPU B's cache	Memory
0				0
1	CPU A reads X	0		0
2	CPU B reads X	0	0	0
3	CPU A writes 1 to X	1	0	1

Coherence Defined

- Informally: Reads return most recently written value
- Formally:
  - P writes X; P reads X (no intervening writes)
    $\Rightarrow$ read returns written value
  - P_1 writes X; P_2 reads X (sufficiently later)
    $\Rightarrow$ read returns written value
    - c.f. CPU B reading X after step 3 in example
  - P_1 writes X, P_2 writes X
    $\Rightarrow$ all processors see writes in the same order
    - End up with the same final value for X

Cache Coherence Protocols

- Operations performed by caches in multiprocessors to ensure coherence
  - Migration of data to local caches
    - Reduces bandwidth for shared memory
  - Replication of read-shared data
    - Reduces contention for access
- Snooping protocols
  - Each cache monitors bus reads/writes
- Directory-based protocols
  - Caches and memory record sharing status of blocks in a directory

Invalidating Snooping Protocols

- Cache gets exclusive access to a block when it is to be written
  - Broadcasts an invalidate message on the bus
  - Subsequent read in another cache misses
    - Owning cache supplies updated value

CPU activity	Bus activity	CPU A's cache	CPU B's cache	Memory
				0
CPU A reads X	Cache miss for X	0		0
CPU B reads X	Cache miss for X	0	0	0
CPU A writes 1 to X	Invalidate for X	1		0
CPU B read X	Cache miss for X	1	1	1

Memory Consistency

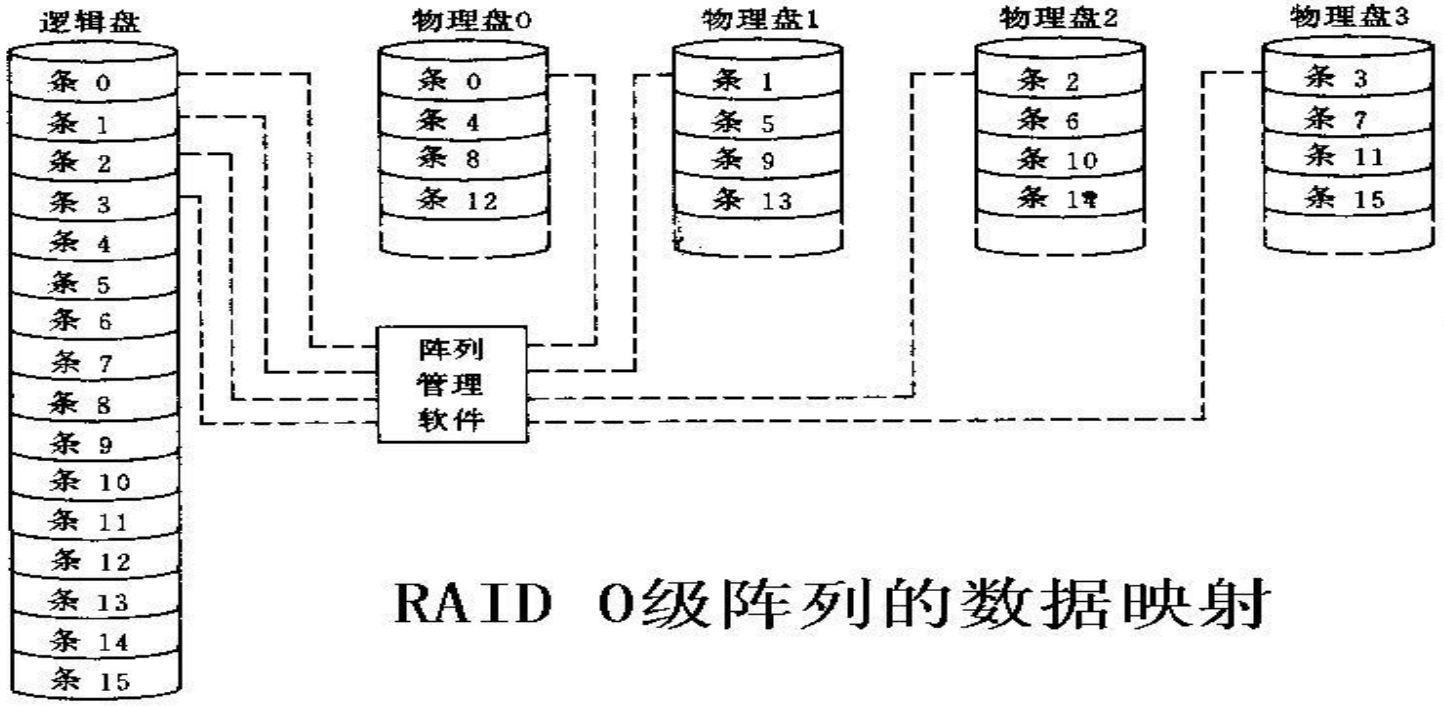
- When are writes seen by other processors
  - “Seen” means a read returns the written value
  - Can’t be instantaneously
- Assumptions
  - A write completes only when all processors have seen it
  - A processor does not reorder writes with other accesses
- Consequence
  - P writes X then writes Y
    $\Rightarrow$ all processors that see new Y also see new X
  - Processors can reorder reads, but not writes

冗余磁盘阵列(RAID)

- RAID (Redundant Arrays of Inexpensive Disk)
  - 将多个独立操作的磁盘按某种方式组织成磁盘阵列，以增加容量
  - 利用类似于主存中的多体交叉技术，将数据存储在多个磁盘上，通过使这些磁盘并行工作来提高数据传输速度
  - 用冗余磁盘技术来进行错误检测和恢复以提高系统可靠性
- RAID级别分为8级（0-7级），以及RAID10（结合0和1级）、RAID30 （结合0和3级）和 RAID50 （结合0和5级）

RAID 0

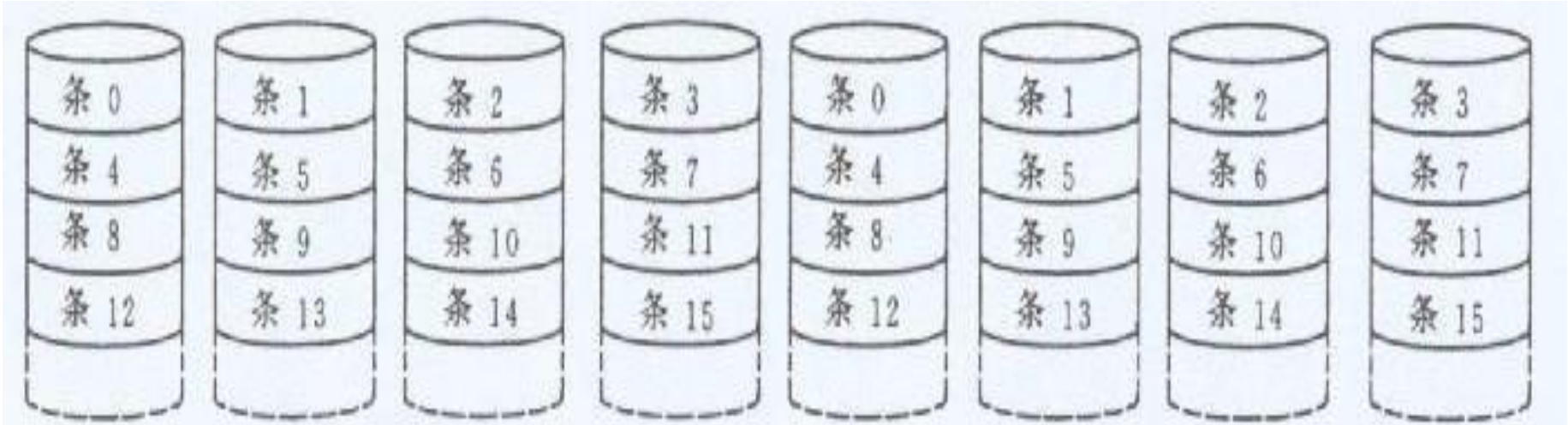
- 无冗余，适用于容量和速度要求高的非关键数据存储的场合
- 与单个大容量磁盘相比，具有较快的I/O响应能力
  - 数据连续分布时，如果两个I/O请求访问不同盘上的数据，则可并行发送，减少了I/O排队时间
  - 交叉分布时，同一个I/O请求有可能并行传送其不同的数据块，因而可达较高的数据传输率。例如，可以用在视频编辑和播放系统中



RAID 0级阵列的数据映射

RAID 1

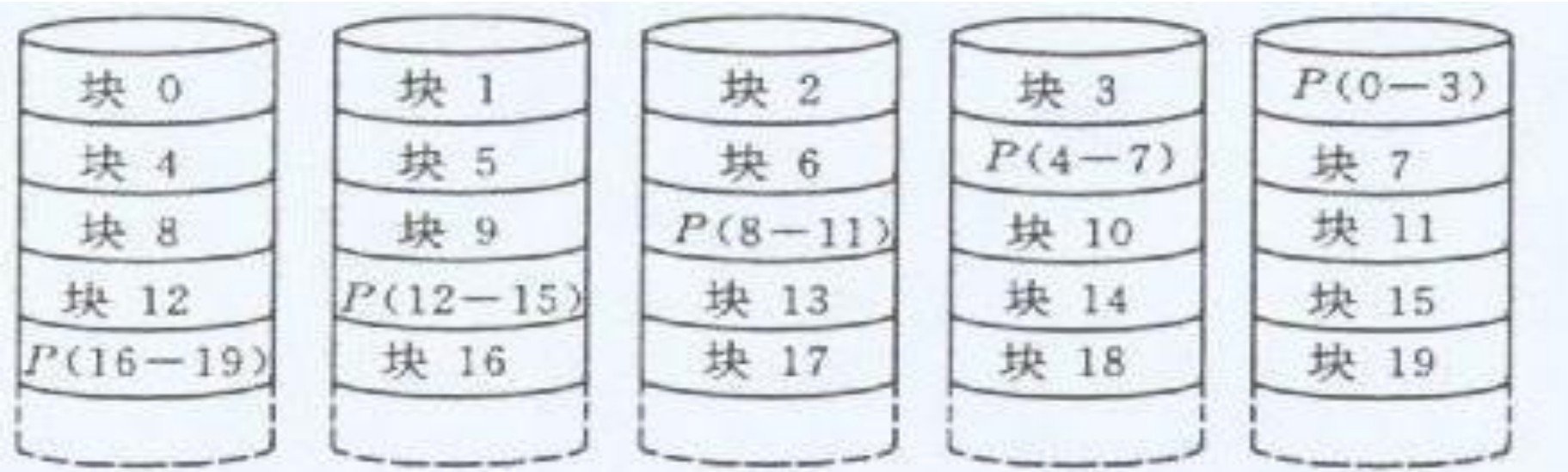
- 镜像盘实现1对1冗余
  - 读：一个读请求可由其中一个定位时间更少的磁盘提供数据。
  - 写：一个写请求对对应的两个磁盘并行更新。故写性能由较慢的一次写来决定，即定位时间更长的那一次。
  - 检错：数据恢复简单。当一个磁盘损坏时，数据仍能从另一个磁盘读取。
- 特点；可靠性高，但价格昂贵
  - 常用于可靠性要求很高的场合，如系统软件的存储，金融、证券等系统。



(b) RAID 1(镜像)

冗余磁盘阵列（RAID 5）

- 奇偶校验块分布在各个磁盘中，所有磁盘的地位等价，这样可提高容错性，并且避免了使用专门校验盘时潜在的I/O瓶颈。
- 采用独立的存取技术，有较高的I/O响应速度。
- 成本不高但效率高，所以被广泛使用



(f) RAID 5(块级分布式奇偶校验)

Multilevel On-Chip Caches

Characteristic	ARM Cortex-A53	Intel Core i7
L1 cache organization	Split instruction and data caches	Split instruction and data caches
L1 cache size	Configurable 16 to 64 KiB each for instructions/data	32 KiB each for instructions/data per core
L1 cache associativity	Two-way (I), four-way (D) set associative	Four-way (I), eight-way (D) set associative
L1 replacement	Random	Approximated LRU
L1 block size	64 bytes	64 bytes
L1 write policy	Write-back, variable allocation policies (default is Write-allocate)	Write-back, No-write-allocate
L1 hit time (load-use)	Two clock cycles	Four clock cycles, pipelined
L2 cache organization	Unified (instruction and data)	Unified (instruction and data) per core
L2 cache size	128 KiB to 2 MiB	256 KiB (0.25 MiB)
L2 cache associativity	16-way set associative	8-way set associative
L2 replacement	Approximated LRU	Approximated LRU
L2 block size	64 bytes	64 bytes
L2 write policy	Write-back, Write-allocate	Write-back, Write-allocate
L2 hit time	12 clock cycles	10 clock cycles
L3 cache organization	–	Unified (instruction and data)
L3 cache size	–	8 MiB, shared
L3 cache associativity	–	16-way set associative
L3 replacement	–	Approximated LRU
L3 block size	–	64 bytes
L3 write policy	–	Write-back, Write-allocate
L3 hit time	–	35 clock cycles

2-Level TLB Organization

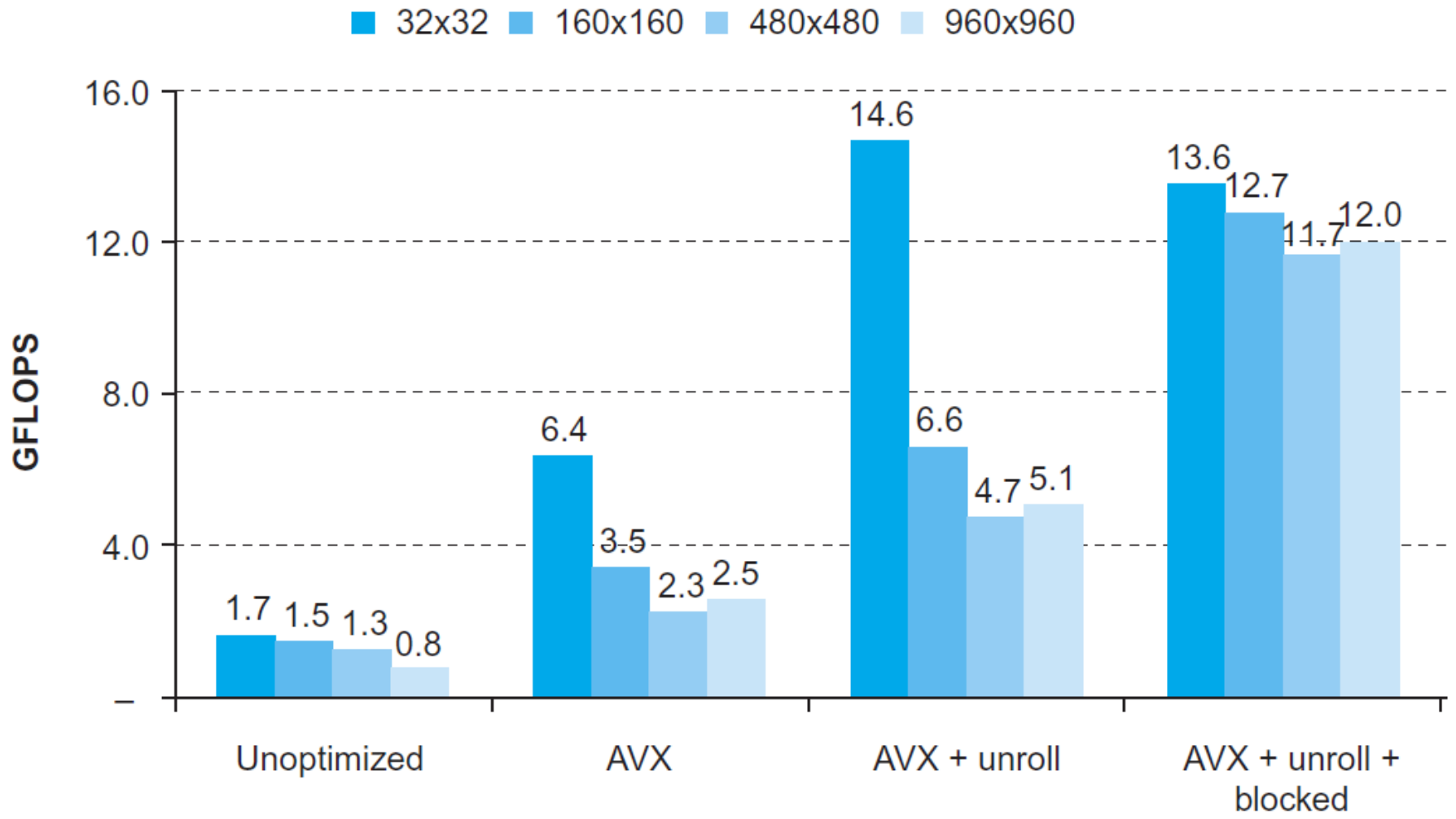
Characteristic	ARM Cortex-A53	Intel Core i7
Virtual address	48 bits	48 bits
Physical address	40 bits	44 bits
Page size	Variable: 4, 16, 64 KiB, 1, 2 MiB, 1 GiB	Variable: 4 KiB, 2/4 MiB
TLB organization	1 TLB for instructions and 1 TLB for data per core Both micro TLBs are fully associative, with 10 entries, round robin replacement 64-entry, four-way set-associative TLBs TLB misses handled in hardware	1 TLB for instructions and 1 TLB for data per core Both L1 TLBs are four-way set associative, LRU replacement L1 I-TLB has 128 entries for small pages, seven per thread for large pages L1 D-TLB has 64 entries for small pages, 32 for large pages The L2 TLB is four-way set associative, LRU replacement The L2 TLB has 512 entries TLB misses handled in hardware

Supporting Multiple Issue

- Both have multi-banked caches that allow multiple accesses per cycle assuming no bank conflicts
- Core i7 cache optimizations
  - Return requested word first
  - Non-blocking cache
    - Hit under miss
    - Miss under miss
  - Data prefetching

DGEMM

- Combine cache blocking and subword parallelism



Pitfalls

- Byte vs. word addressing
  - Example: 32-byte direct-mapped cache, 4-byte blocks
    - Byte 36 maps to block 1
    - Word 36 maps to block 4
- Ignoring memory system effects when writing or generating code
  - Example: iterating over rows vs. columns of arrays
  - Large strides result in poor locality

Pitfalls

- In multiprocessor with shared L2 or L3 cache
  - Less associativity than cores results in conflict misses
  - More cores ⇒ need to increase associativity
- Using AMAT (存储器平均访问时间) to evaluate performance of out-of-order processors
  - Ignores effect of non-blocked accesses
  - Instead, evaluate performance by simulation

Pitfalls

- Extending address range using segments
  - E.g., Intel 80286
  - But a segment is not always big enough
  - Makes address arithmetic complicated
- Implementing a VMM on an ISA not designed for virtualization
  - E.g., non-privileged instructions accessing hardware resources
  - Either extend ISA, or require guest OS not to use problematic instructions

Concluding Remarks

- Fast memories are small, large memories are slow
  - We really want fast, large memories ☹
  - Caching gives this illusion ☺
- Principle of locality
  - Programs use a small part of their memory space frequently
- Memory hierarchy
  - L1 cache ↔ L2 cache ↔ … ↔ DRAM memory ↔ disk
- Memory system design is critical for multiprocessors