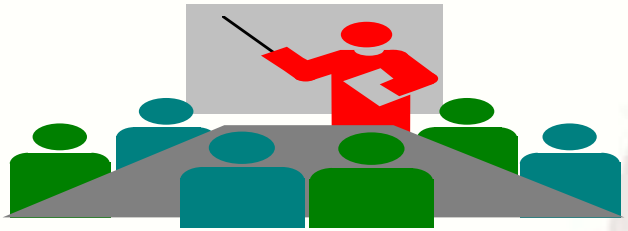


计算机组成与设计

Computer Organization & Design

The Hardware/Software Interface

Chapter 5

Large and Fast:

Exploiting Memory Hierarchy

Haifeng Liu
College of Computer Science and Technology, Zhejiang University
haifengliu@zju.edu.cn

Contents of Chapter 5

Locality -- two important concepts

1. temporal locality (locality in time):

If an item is referenced, it will tend to be referenced again soon.

2. spatial locality (locality in space):

If an item is referenced, items whose addresses are close by will tend to be referenced soon.

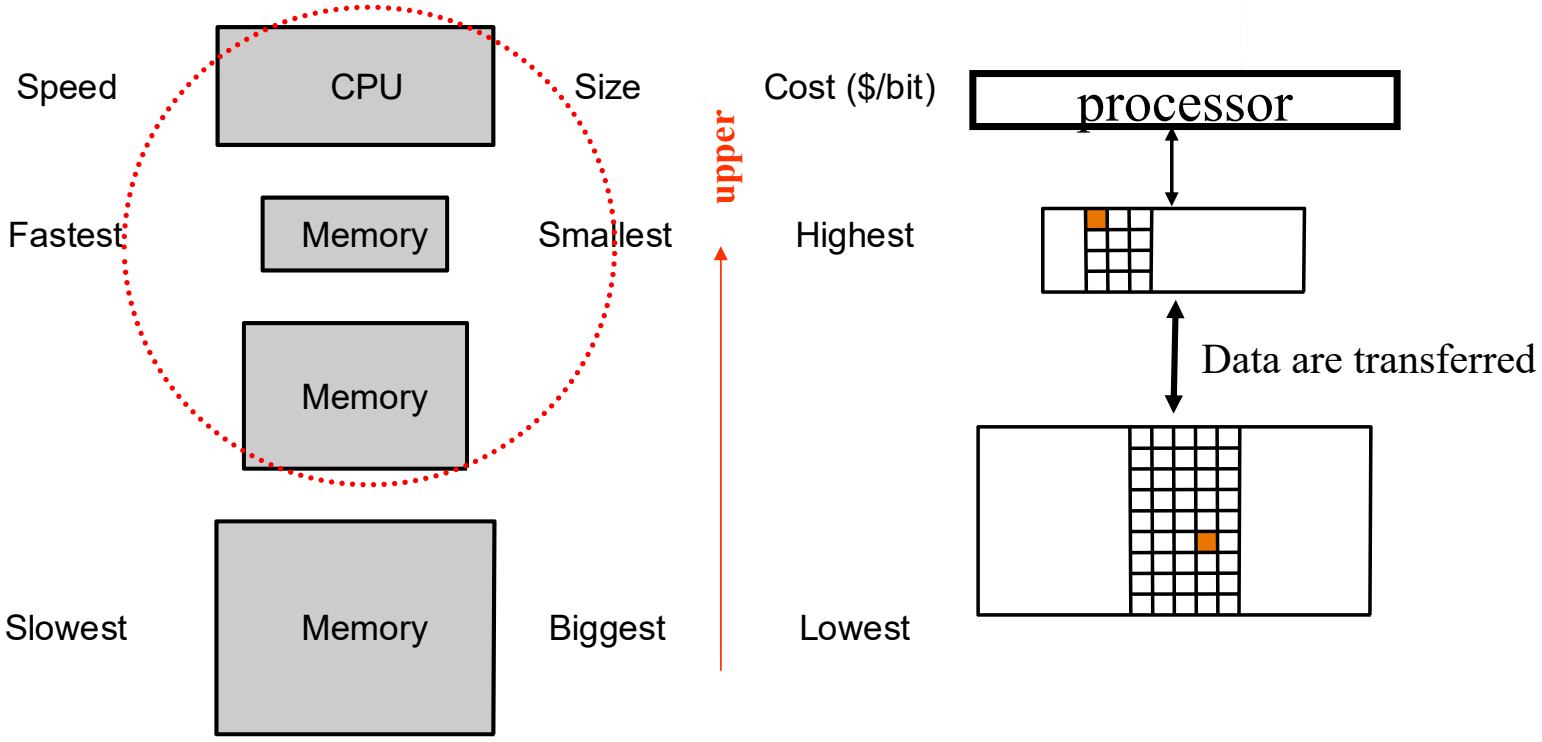
- **As we know, these two principles actually exists in most programs.**
  - *Why does code have locality?*
- **Our initial focus: two levels (upper, lower)**
  - **block:** minimum unit of data (block) for transfers
  - **hit:** data requested is in the **upper** level
  - **miss:** data requested is not in the **upper** level

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

Solutions

- **Build a memory hierarchy**



Some important items

hit: The CPU accesses the upper level and succeeds.

Miss: The CPU accesses the upper level and fails.

Hit time:

The time to access the upper level of the memory hierarchy, which includes the time needed to determine whether the access is a hit or a miss.

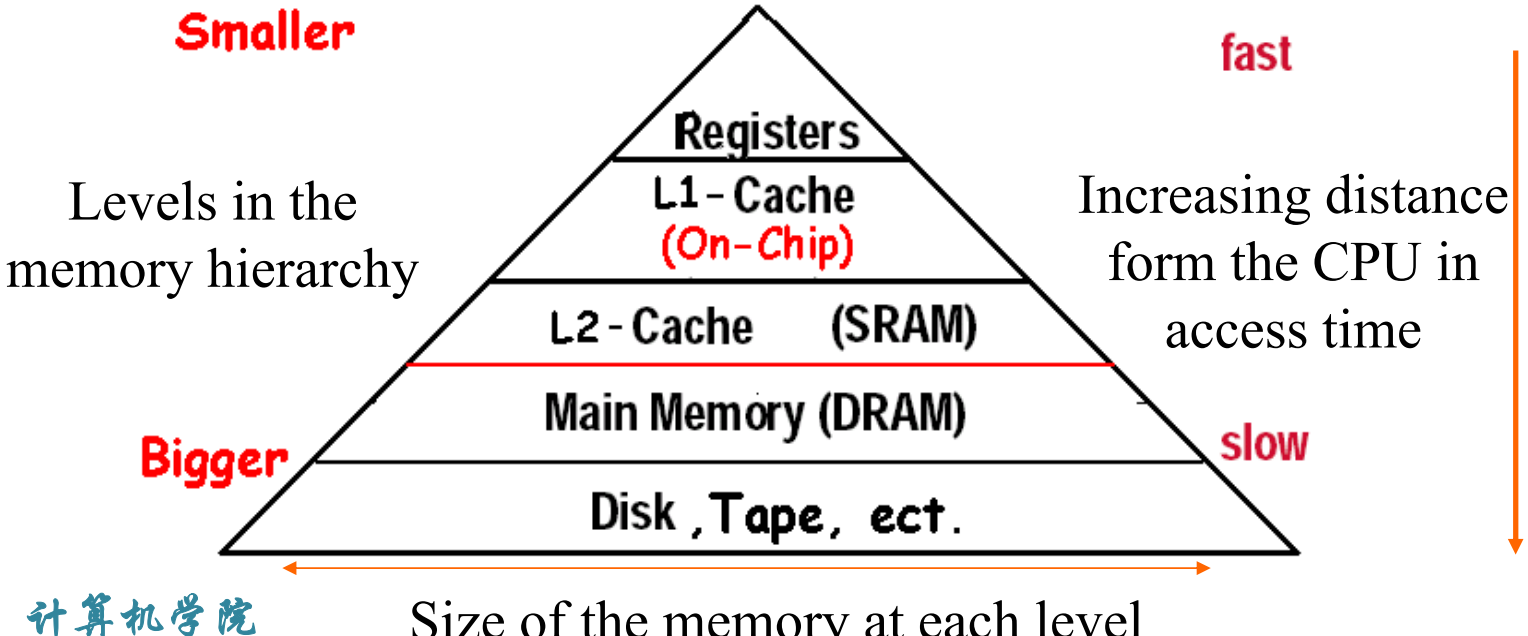
miss penalty:

The time to replace a block in the upper level with the corresponding block from the lower level, plus the time to deliver this block to the processor.

Exploiting Memory Hierarchy

The method

- Hierarchies bases on memories of different speeds and size
- The more closely CPU the level is,the faster the one is.
- The more closely CPU the level is,the smaller the one is.
- The more closely CPU the level is,the more expensive



There has been exploited Memory Hierarchy

1. The basics of Cache: SRAM and DRAM (main memory)

The solution is in speed

2. Virtual Memory: DRAM and DISK

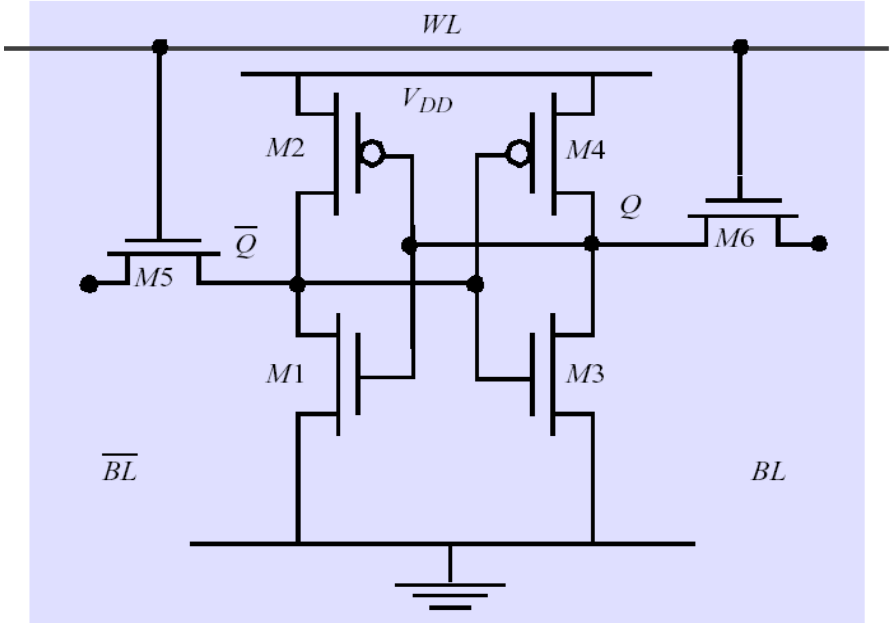
The solution is in size

5.2 Memory Technology

- **Static RAM (SRAM)**
  - 0.5ns – 2.5ns, $2000 – $5000 per GB
- **Dynamic RAM (DRAM)**
  - 50ns – 70ns, $20 – $75 per GB
- **Magnetic disk**
  - 5ms – 20ms, $0.20 – $2 per GB
- **Ideal memory**
  - Access time of SRAM
  - Capacity and cost/GB of disk

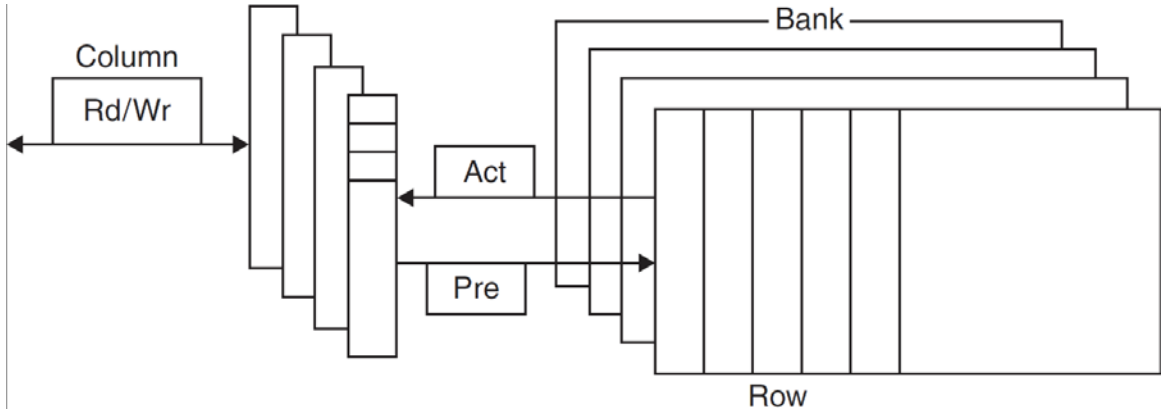
SRAM Technology

- SRAM
  - value is stored on a pair of inverting gates
  - very fast but takes up more space than DRAM (4 to 6 transistors)



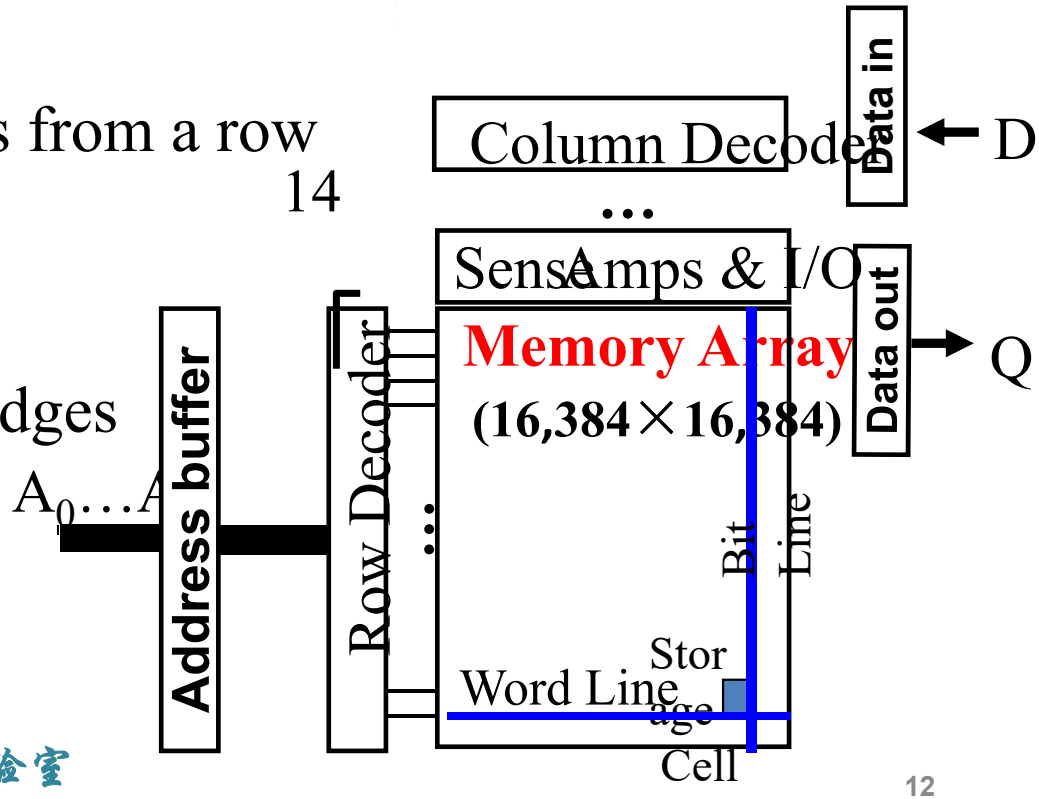
DRAM Technology

- **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”



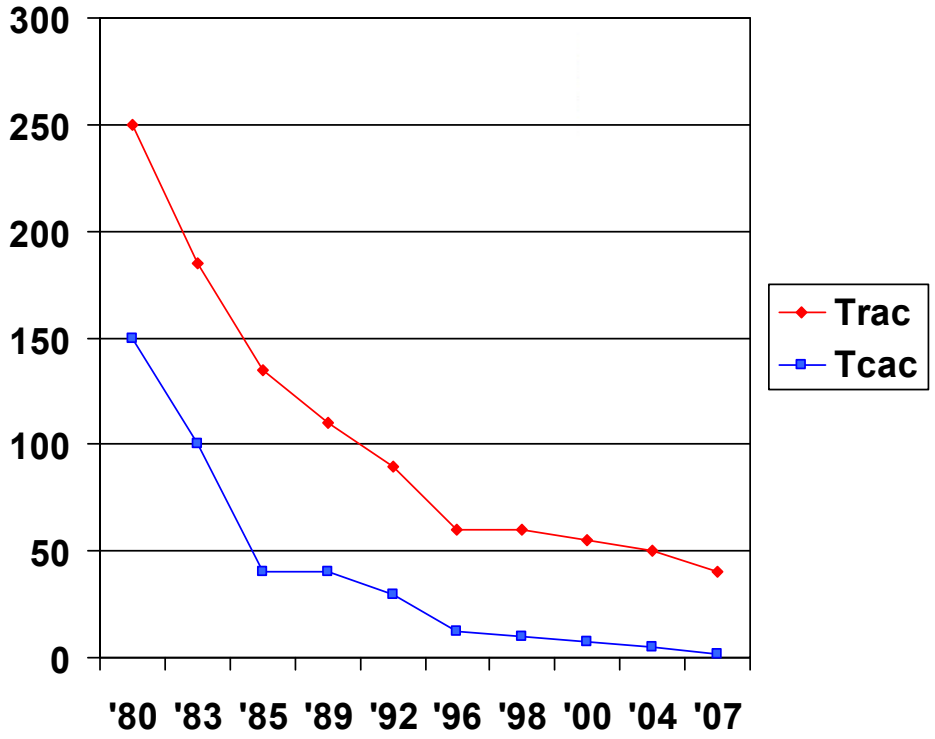
Advanced DRAM Organization

- **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



DRAM Generations

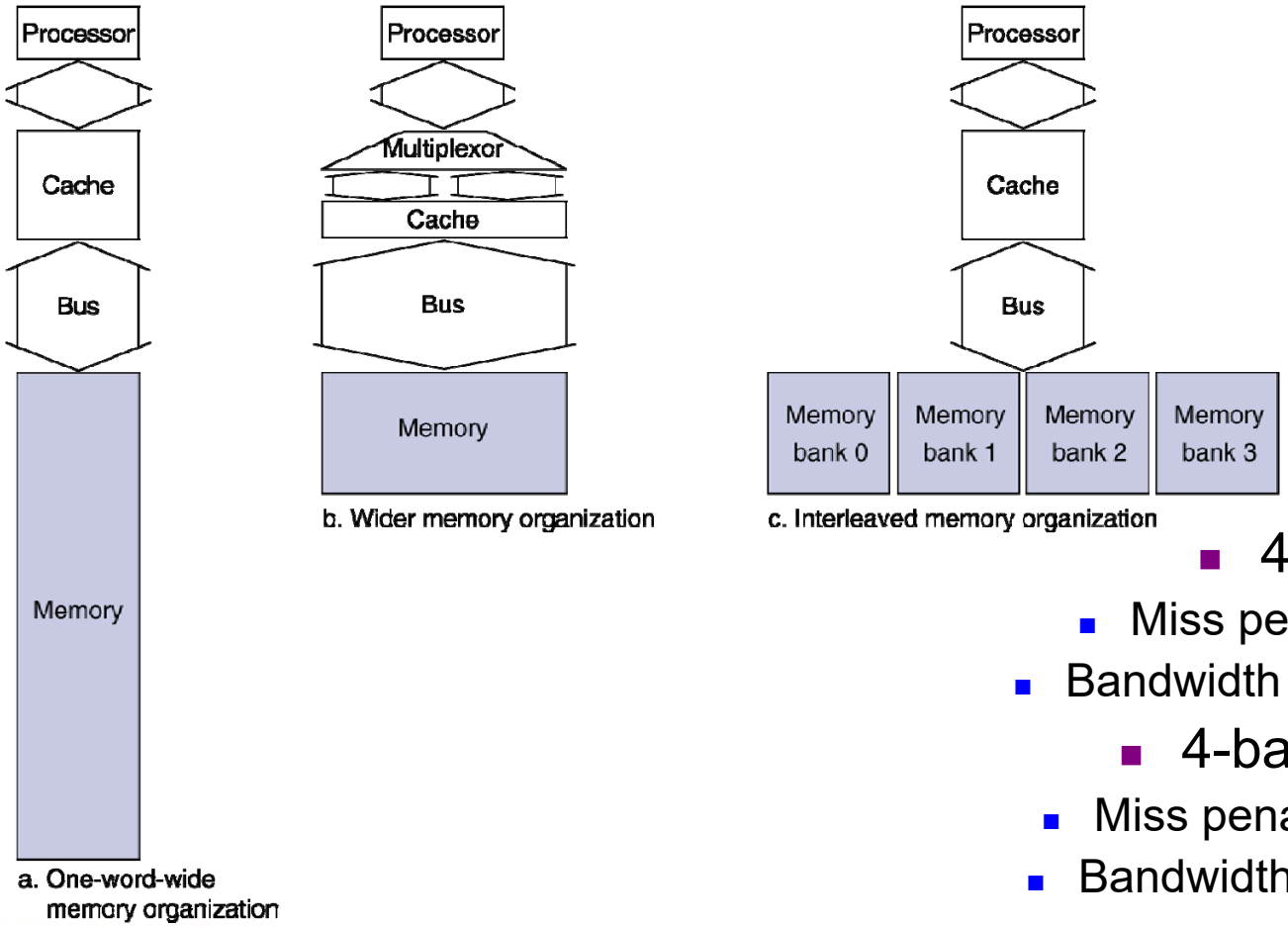
Year	Capacity	$/GB
1980	64Kbit	$1500000
1983	256Kbit	$500000
1985	1Mbit	$200000
1989	4Mbit	$50000
1992	16Mbit	$15000
1996	64Mbit	$10000
1998	128Mbit	$4000
2000	256Mbit	$1000
2004	512Mbit	$250
2007	1Gbit	$50



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

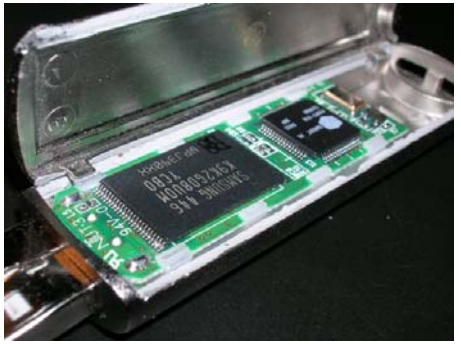
Increasing Memory Bandwidth



- 4-word wide memory
  - Miss penalty = 1 + 15 + 1 = 17 bus cycles
  - Bandwidth = 16 bytes / 17 cycles = 0.94 B/cycle
- 4-bank interleaved memory
  - Miss penalty = 1 + 15 + 4×1 = 20 bus cycles
  - Bandwidth = 16 bytes / 20 cycles = 0.8 B/cycle

Flash Storage

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

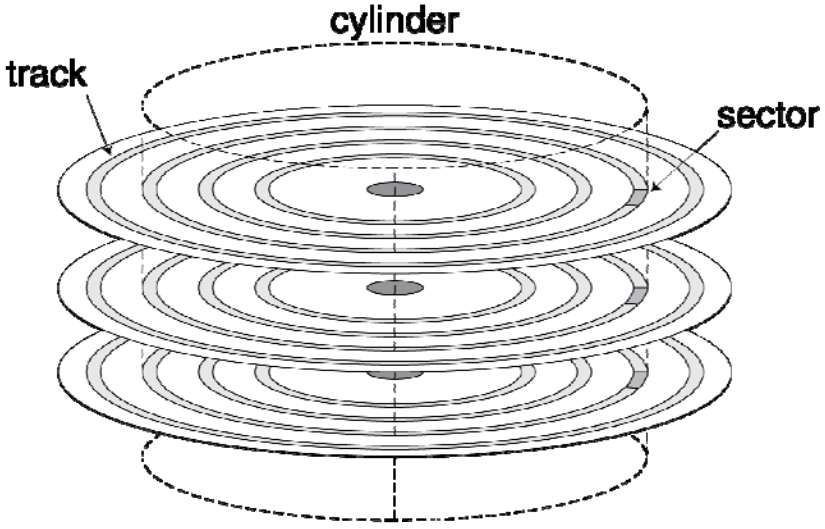


Flash Types

- 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 1000's of accesses
  - Not suitable for direct RAM or disk replacement
  - Wear leveling: remap data to less used blocks

Disk Storage

- Nonvolatile, rotating magnetic storage



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
  - Synchronization fields and gaps
- **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

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

Problems in memory designing

- In fact

Memory technology	Typical access time	Cost per GByte (2004)
SRAM	0.5-5ns	$4000-$10,000
DRAM	50-70ns	$100-$200
Magnetic disk	5,000,000-20,000,000ns	

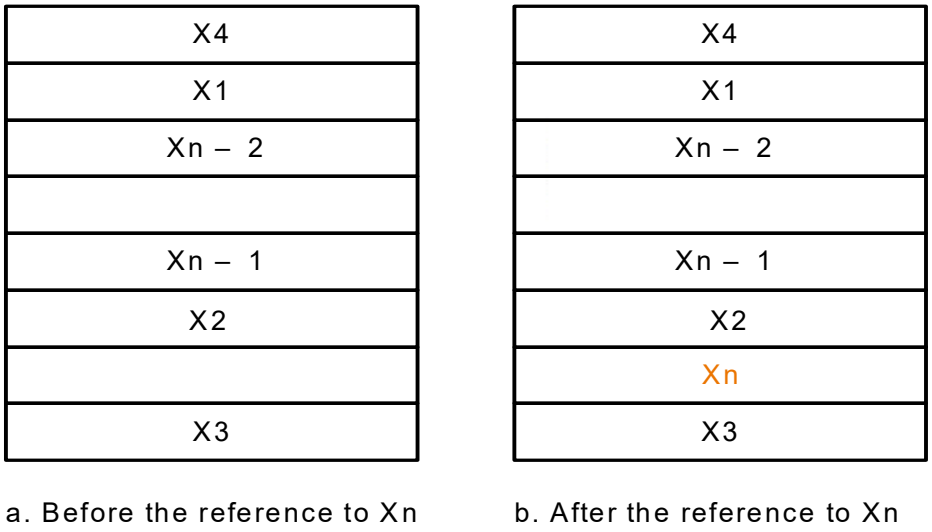
- Users want large and fast memories!

5.3 The basics of Cache

Simple implementations

- For each item of data at the lower level, there is exactly one location in the cache where it might be.

e.g., lots of items at the lower level share locations in the upper level



a. Before the reference to Xn

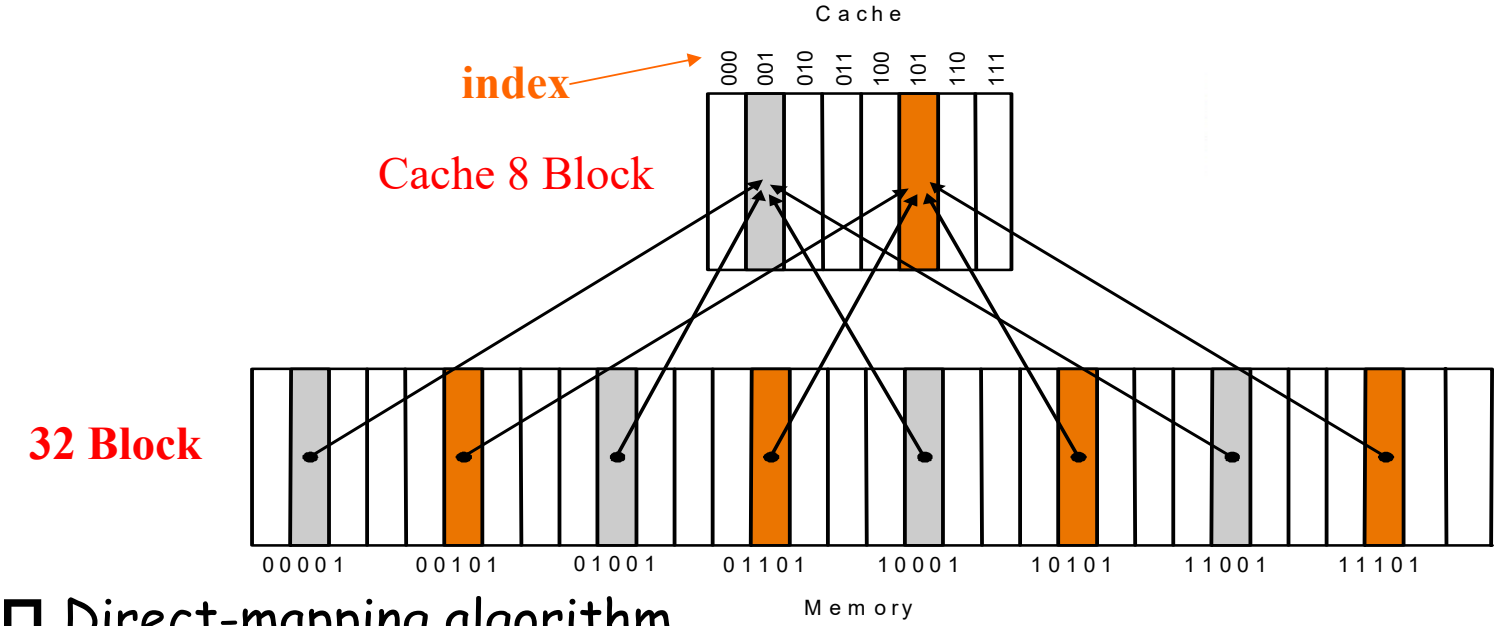
b. After the reference to Xn

- **Two issues:**
  - How do we know if a data item is in the cache?
  - If it is, how do we find it?

- **Our first example: "direct mapped"**
  - block size is one word of data

Direct Mapped Cache

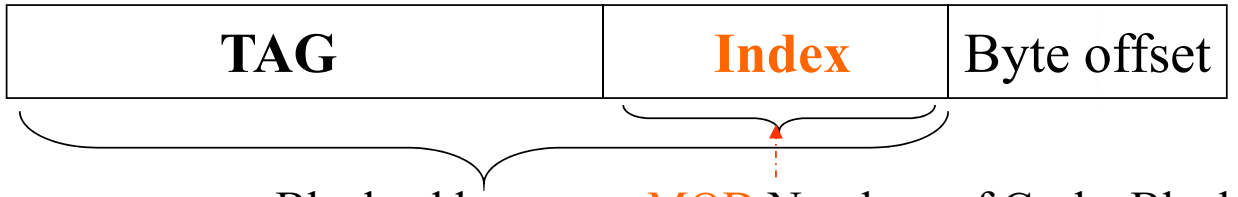
- Where can a block be placed in the upper level?



- Direct-mapping algorithm.
  memory address is modulo the number of blocks in the cache
- Fortunately, while the cache has 2^n blocks, the corresponding index is equal to the lowest n bits of memory block address. Here n=3. Let's check

Accessing a cache---how do we find it?

- **Memory block address is larger than cache block address**



Valid bit

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

a. The initial state of the cache after power-on

Access sequence

- 10110,11010,10110,11010,10000,00011,10000,10010

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	Y	$(10)_2$	Memory(10110)
111	N		

b. After handling a miss of address(10110)

Index	V	Tag	Data
000	N		
001	N		
010	Y	$(11)_2$	Memory(11010)
011	N		
100	N		
101	N		
110	Y	$(10)_2$	Memory(10110)
111	N		

c. After handling a miss of address(11010)

Index	V	Tag	Data
000	N		
001	N		
010	Y	$(11)_2$	Memory(11010)
011	N		
100	N		
101	N		
110	Y	$(10)_2$	Memory(10110)
111	N		

d. After handling a hit of address(10110)

Index	V	Tag	Data
000	N		
001	N		
010	Y	$(11)_2$	Memory(11010)
011	N		
100	N		
101	N		
110	Y	$(10)_2$	Memory(10110)
111	N		

e. After handling a hit of address(11010)

Access sequence-2

- 10110,11010,10110,11010,10000,00011,10000,10010

Index	V	Tag	Data
000	Y	$(10)_2$	Memory(10000)
001	N		
010	Y	$(11)_2$	Memory(11010)
011	N		
100	N		
101	N		
110	Y	$(10)_2$	Memory(10110)
111	N		

f. After handling a miss of address(10000)

Index	V	Tag	Data
000	Y	$(10)_2$	Memory(10000)
001	N		
010	Y	$(11)_2$	Memory(11010)
011	Y	$(00)_2$	Memory(00011)
100	N		
101	N		
110	Y	$(10)_2$	Memory(10110)
111	N		

g. After handling a miss of address(00011)

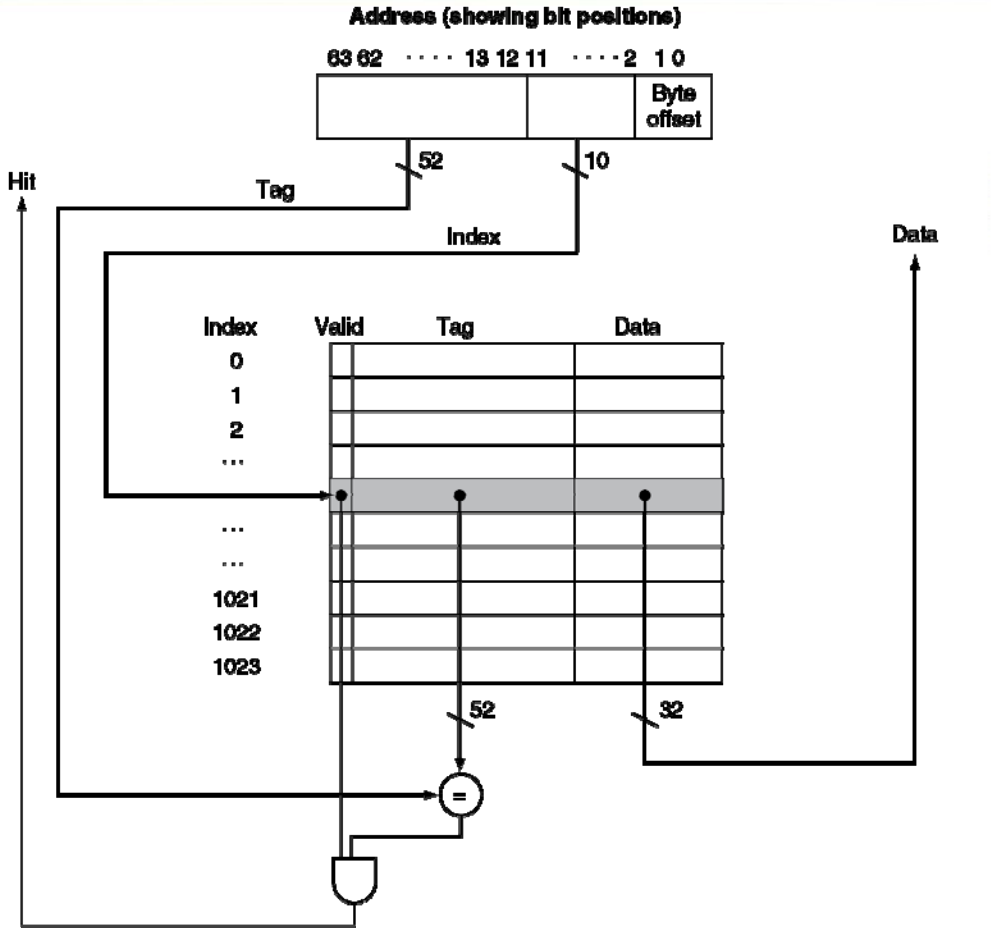
Index	V	Tag	Data
000	Y	$(10)_2$	Memory(10000)
001	N		
010	Y	$(11)_2$	Memory(11010)
011	Y	$(00)_2$	Memory(00011)
100	N		
101	N		
110	Y	$(10)_2$	Memory(10110)
111	N		

h. After handling a hit of address(10000)

Index	V	Tag	Data
000	Y	$(10)_2$	Memory(10000)
001	N		
010	Y	$(11) \rightarrow (10)_2$	Memory(10010)
011	Y	$(00)_2$	Memory(00011)
100	N		
101	N		
110	Y	$(10)_2$	Memory(10110)
111	N		

i. After handling a miss of address(10010)

Address Subdivision



Bits in Cache

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

Mapping an Address to Multiword Cache Block

Example

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

Answer

(Block address) modulo (Number of cache blocks)

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

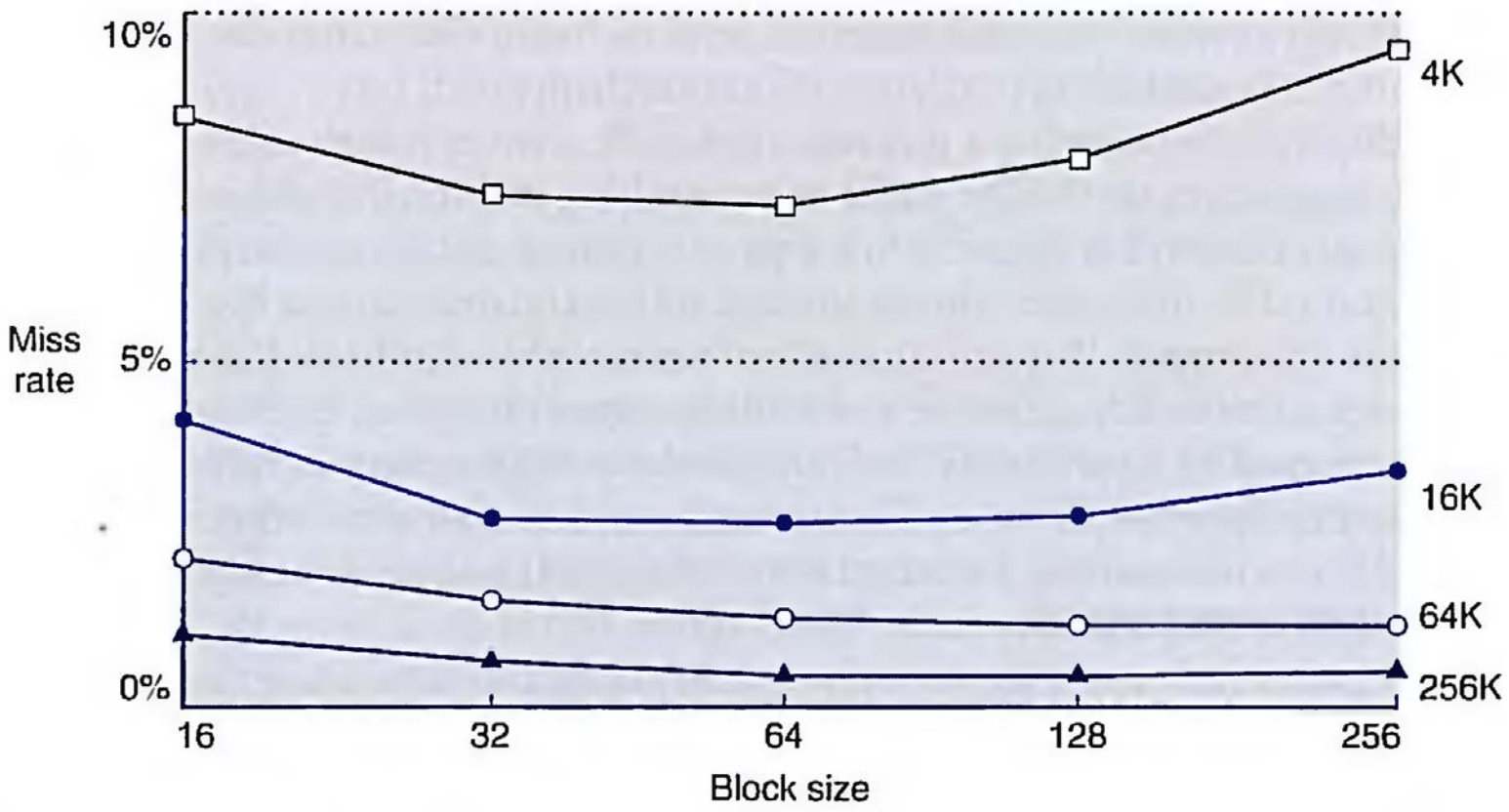
$$\left\lfloor \frac{\text{Byte address}}{\text{Bytes per block}} \right\rfloor \times \text{Byte per block} \longleftrightarrow \left\lfloor \frac{\text{Byte address}}{\text{Bytes per block}} \right\rfloor \times \text{Byte per block} + (\text{Byte per block} - 1)$$

Here: 1200 ⟷ 1215

Block Size Considerations

- **Larger blocks should reduce miss rate**
  - Due to spatial locality
- **But in a fixed-sized cache**
  - Larger blocks ⇒ fewer of them
    - More competition ⇒ increased miss rate
  - Larger blocks ⇒ pollution
- **Larger miss penalty**
  - Can override benefit of reduced miss rate
  - Early restart and critical-word-first can help

Miss rate versus block size.



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**
  - Allocate on miss: fetch the block
  - Write around: 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

Handling Cache reads hit and Misses

- **Read hits**
  - this is what we want!
- **Read misses—two kinds of misses**
  - instruction cache miss
  - data cache miss
- **let's see main steps taken on an instruction cache miss**
  - **Stall the CPU**, fetch block from memory, deliver to cache, restart CPU read

1. Send the original PC value (current PC-4) to the memory.

2. Instruct main memory to perform a read and wait for the memory to complete its access.

3. Write the cache entry, putting the data from memory in the data portion of the entry, writing the upper bits of the address (from the ALU) into the tag field, and turning the valid bit on.

4. Restart the instruction execution at the first step, which will refetch the instruction again, this time finding it in the cache.

Handling Cache Writes hit and Misses

- **Write hits:** Difference Strategy
  - write-back: Cause Inconsistent
    - Wrote the data into only the data cache
    - Strategy ---- write back data from the cache to memory later *Fast!*
  - write-through: Ensuring Consistent
    - Write the data into both the memory the cache
    - Strategy ---- writes always update both the cache and the memory
    - Slower!----write buffer
- **Write misses:**
  - read the entire block into the cache, then write the word----**write allocate**
  - Write around the data into the memory (the lower level memory) ----**no write allocate** (also called **write around**).

Example

- Assume a fully associative write-back cache with many cache entries that starts empty. Below is a sequence of five memory operations(the address is in square brackets):

  1 write Mem[100];
  2 write Mem[100];
  3 Read Mem[200];
  4 write Mem[200];
  5 write Mem[100];

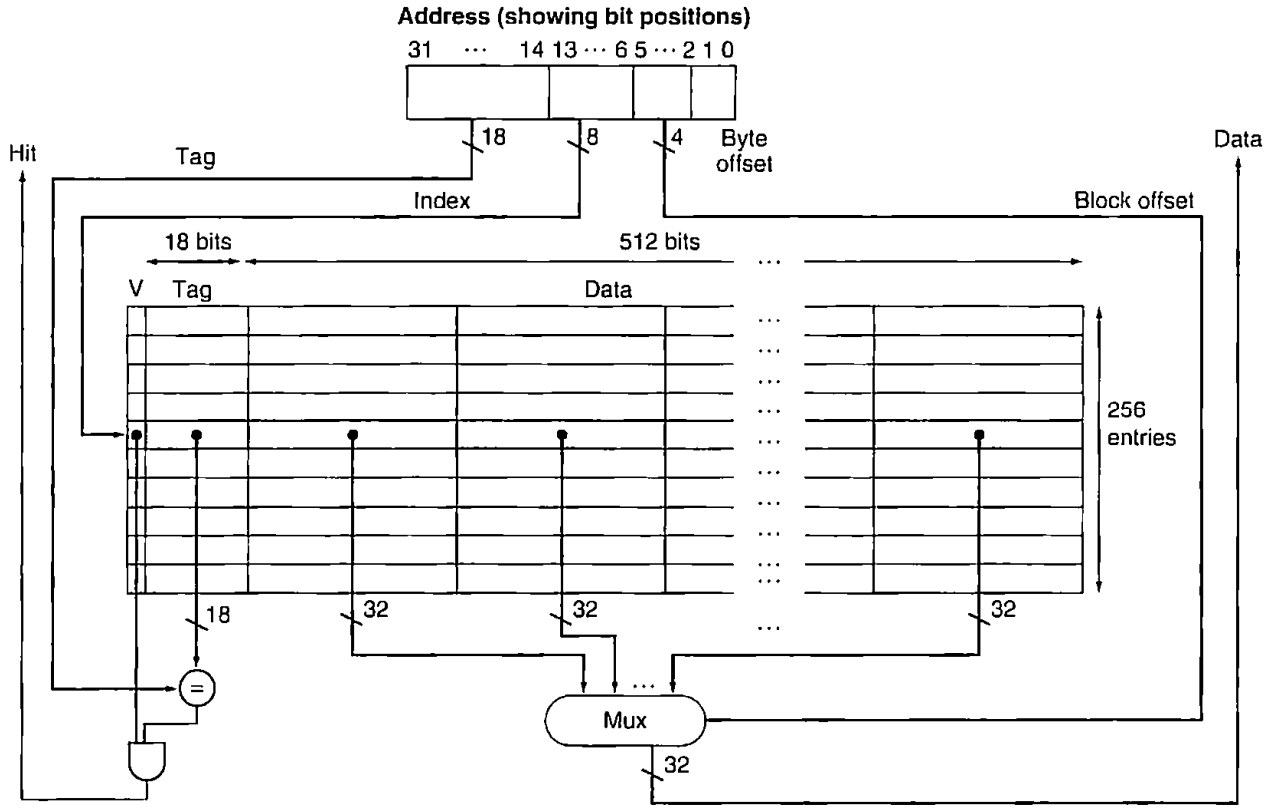
Answer :

for no-write allocate	misses:	1,2,3,5
	hit :	4
for write allocate	misses:	1,3
	hit :	2,4,5

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



- The 16 KB caches in the Intrinsity FastMATH each contain 256 blocks with 16 words per block.

read request

1. Send the address to the appropriate cache. The address comes either from the PC (for an instruction) or from the ALU (for data).

2. If the cache signals hit, the requested word is available on the data lines. Since there are 16 words in the desired block, we need to select the right one. A block index field is used to control the multiplexor (shown at the bottom of the figure), which selects the requested word from the 16 words in the indexed block.

3. If the cache signals miss, we send the address to the main memory. When the memory returns with the data, we write it into the cache and then read it to fulfill the request.

writes

- both write-through and write-back, leaving it up to the operating system to decide which strategy to use for an application.

- It has a one-entry write buffer.

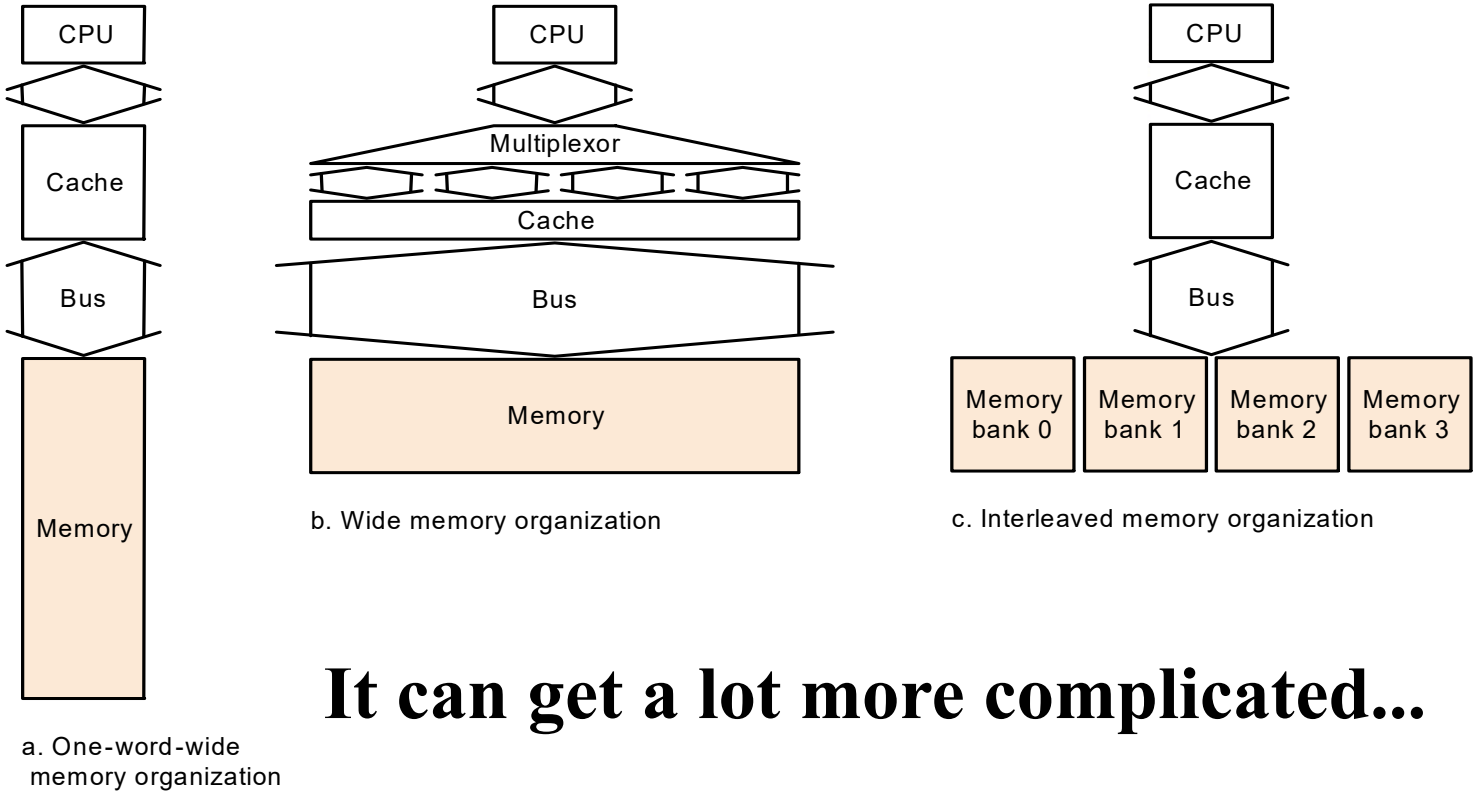
- Approximate instruction and data miss rates for the Intrinsity FastMATH processor for SPEC2000 benchmarks.

Instruction miss rate	Data miss rate	Effective combined miss rate
0.4%	11.4%	3.2%

- combined cache vs split caches

Designing the Memory system to Support Cache

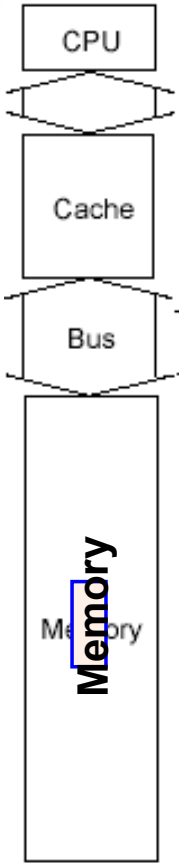
- Make reading multiple words easier by using banks of memory



-

It can get a lot more complicated...

Performance basic memory organization



Assume

1 clock cycles to send the address

15 memory bus clock cycles for each DRAM access initiated

1 bus clock cycles to send a word of data

Block size is 4 words

Every word is 4 bytes

The time to transfer one word is 1+15+1=17

The miss penalty (The time to transfer one block is):

$$1+4\times(1+15)=65 \text{ CLKs}$$

Bandwidth : $\frac{4\times4}{65} \approx \frac{1}{4}$

Only one word is useful, and three other words may be useless. So, for caches using four-word blocks, this memory system is not viable.

Performance in Wider Main Memory

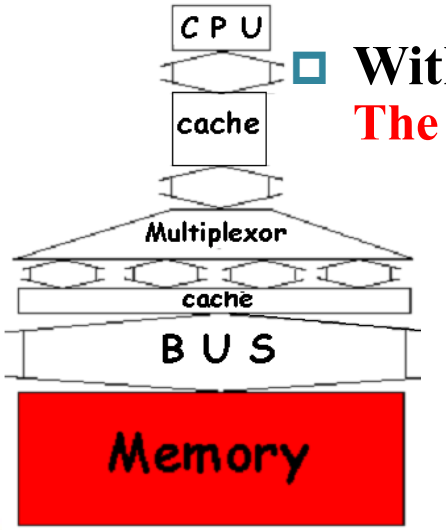
- With a main memory width of 2 words(64bits)

  The miss penalty: 4words/Block

$$1+2\times(15+1)=33 \text{ CLKs}$$

only two times that needed to transfer one word.

Bandwidth : $\frac{4\times4}{33}=\frac{16}{33}\approx0.48$



- With a main memory width of 4 words(128bits)

  The miss penalty: 4words/Block

$$1+1\times(15+1)=17 \text{ CLKs}$$

Bandwidth : $\frac{4\times4}{17}=\frac{16}{17}\approx0.98$

Equal to time to transfer one word.

Performance in Four-way interleaved memory

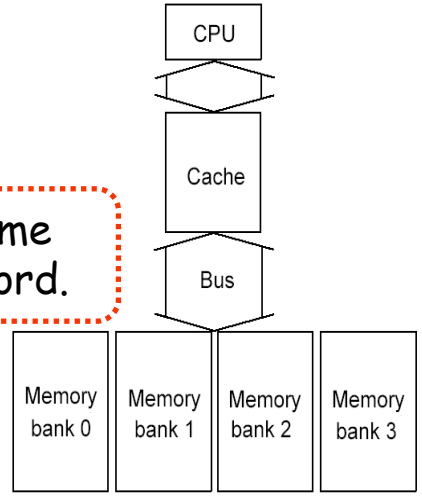
- With 4 banks Interleaved Memory

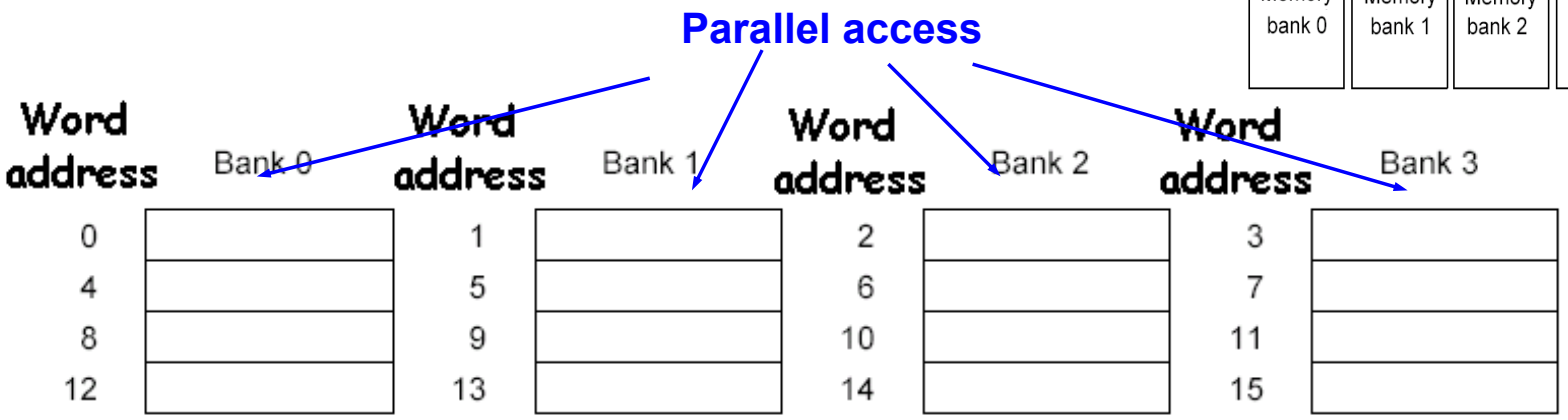
The miss penalty: 4words/Block

$$1+15+(4\times 1)=20$$

Almost equal to time to transfer one word.

Bandwidth : $\frac{4\times 4}{20}=0.8$





Optimizes sequential address access patterns

DRAM developed

Year introduced	Chip size	$ per MB	Total access time to a new row/column	Columm access time to existing row
1980	64Kbit	$1500	250ns	150ns
1983	128Kbit	$500	185ns	100ns
1985	1Mbit	$200	135ns	40ns
1989	4Mbit	$50	110ns	40ns
1992	16Mbit	$15	90ns	30ns
1996	64Mbit	$10	60ns	12ns
1998	128Mbit	$4	60ns	10ns
2000	256Mbit	$1	55ns	7ns
2002	512Mbit	$0.25	50ns	5ns
2004	1024Mbit	$0.10	45ns	3ns

DRAM size increased by multiples of four approximately once every three year until 1996,and thereafter doubling approximately every two years.

Deep concept in Cache

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?**
  (Block placement)
  - Fully Associative, Set Associative, Direct Mapped
- **Q2: How is a block found if it is in the upper level?**
  (Block identification)
  - Tag/Block
- **Q3: Which block should be replaced on a miss?**
  (Block replacement)
  - Random, LRU,FIFO
- **Q4: What happens on a write?**
  (Write strategy)
  - Write Back or Write Through (with Write Buffer)

Q1: Block Placement

- **Direct mapped**
  - Block can only go in one place in the cache

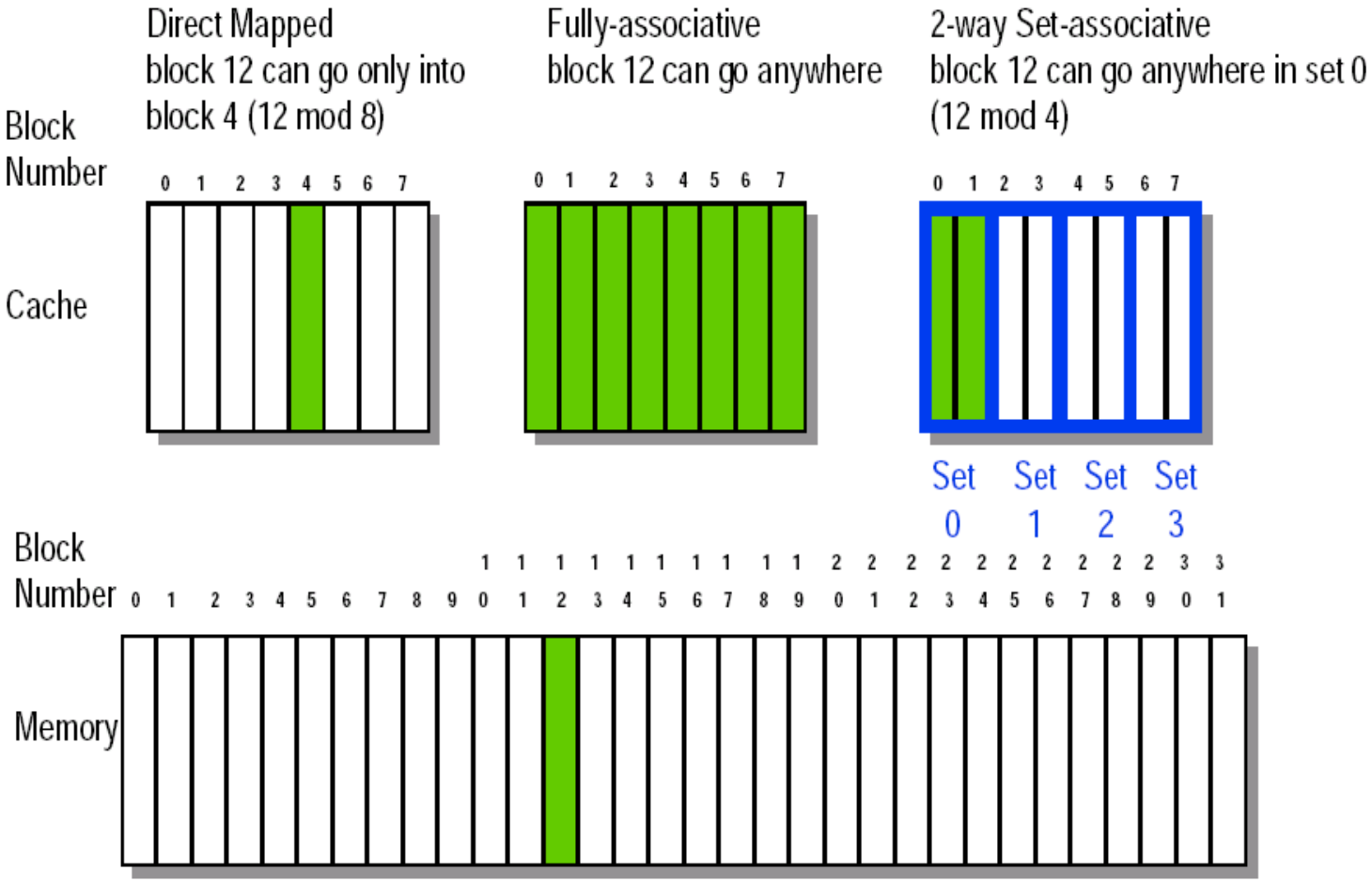
    Usually address MOD Number of blocks in cache
- **Fully associative**

  Block can go anywhere in cache.
- **Set associative**
  - Block can go in one of a set of places in the cache.
  - A set is a group of blocks in the cache.

    Block address MOD Number of *sets* in the cache
  - If a set have n blocks, the cache is said to be n-way set associative.

•Note that direct mapped is the same as 1-way set associative, and fully associative is m-way set-associative (for a cache with m blocks).

Figure 8-32 Block Placement



Q2: Block Identification

Tag

- Every block has an **address tag** that stores the main memory address of the data stored in the block.
- When checking the cache, the processor will **compare** the requested **memory address to the cache tag** -- if the two are equal, then there is a cache hit and the data is present in the cache

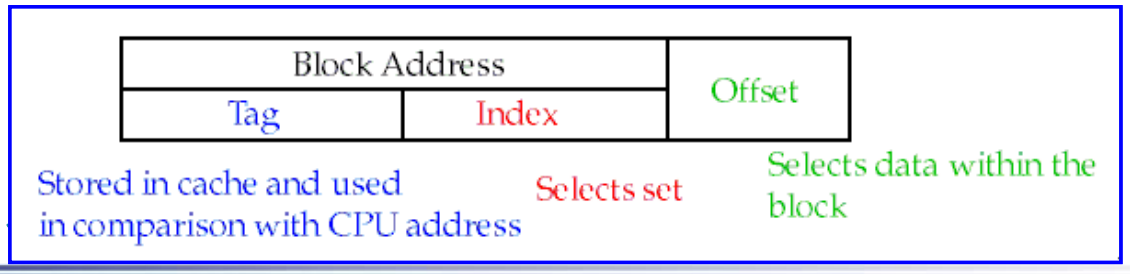
Valid bit

- Often, each cache block also has a **valid bit** that tells if the contents of the cache block are valid

The Format of the Physical Address

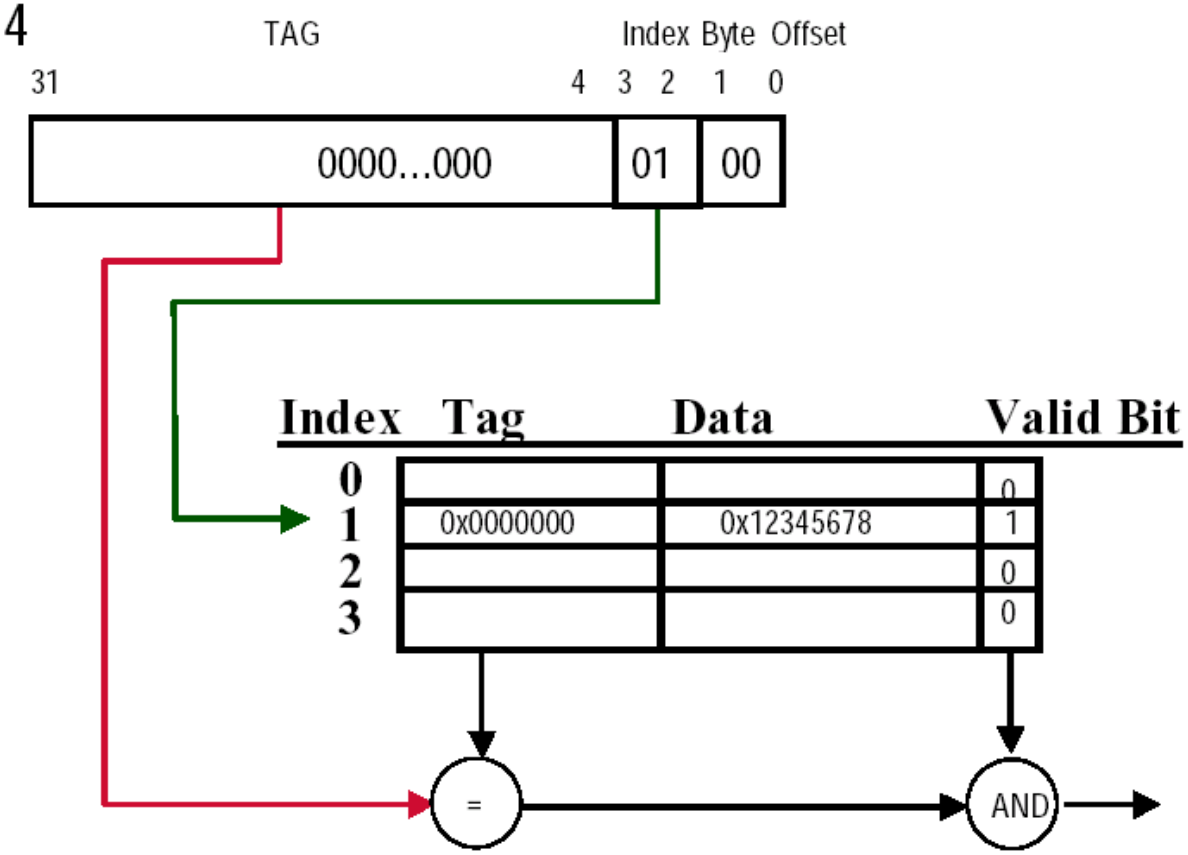
- The **Index** field selects
  - The set, in case of a set-associative cache
  - The block, in case of a direct-mapped cache
  - Has as many bits as log2(#sets) for set-associative caches, or log2(#blocks) for direct-mapped caches
- The **Byte Offset** field selects
  - The byte within the block
  - Has as many bits as log2(size of block)
- The **Tag** is used to find the matching block within a set or in the cache
  - Has as many bits as

Address_size - Index_size - Byte_Offset_Size



Direct-mapped Cache Example (1-word Blocks)

LOAD R1, 0x04

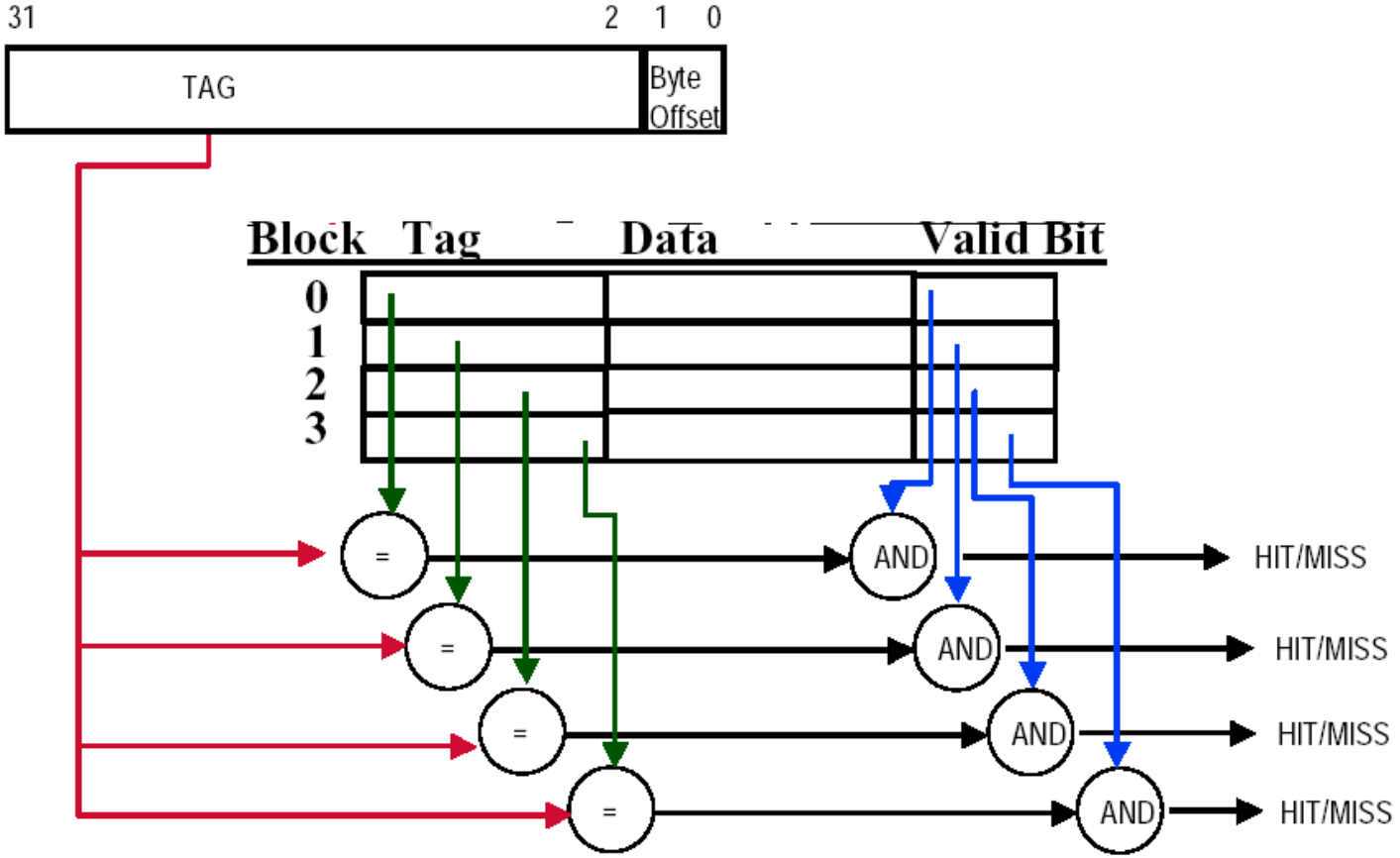


MEMORY

Address	Data
0x00	0x00000000
0x04	0x12345678
0x08	0x87654321
0x0C	0x11111111
0x10	0x22222222
0x14	0x33333333
0x18	0x44444444
0x1C	0x55555555
0x20	0x10101010

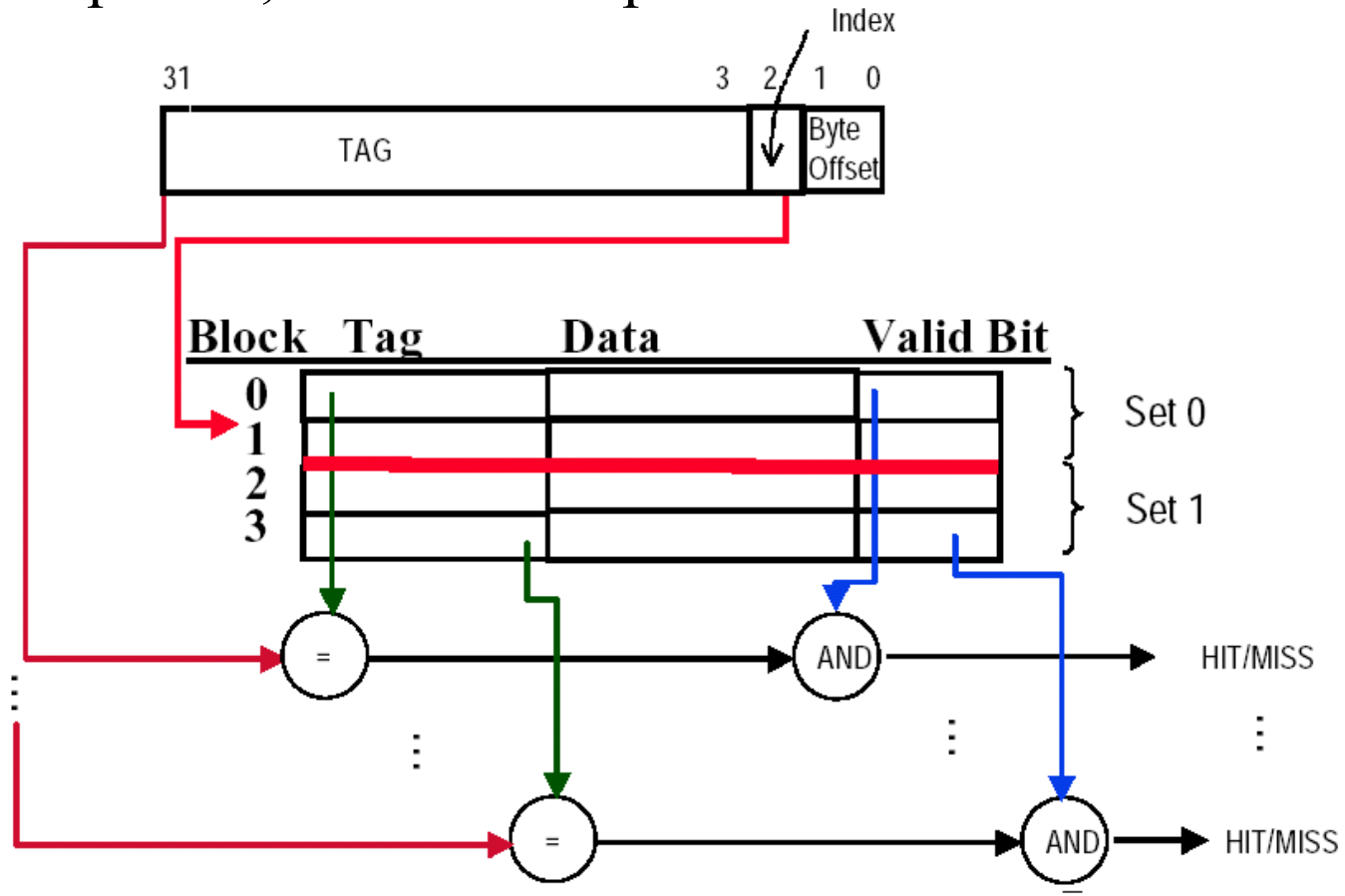
Fully-Associative Cache example (1-word Blocks)

- Assume cache has 4 blocks



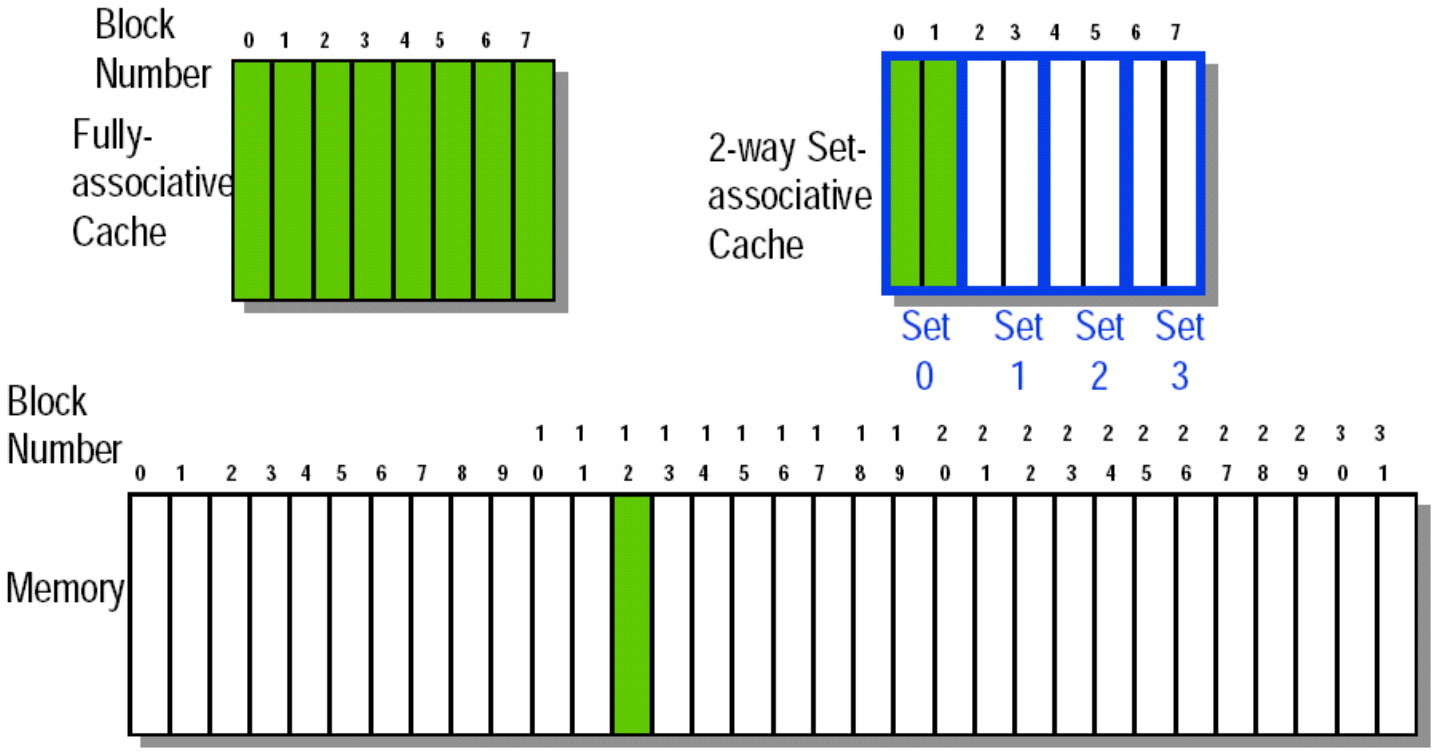
2-Way Set-Associative Cache

- Assume cache has 4 blocks and each block is 1 word
- 2 blocks per set, hence 2 sets per cache



Q3: Block Replacement

- In a direct-mapped cache, there is **only one block** that can be replaced
- In set-associative and fully-associative caches, there are N blocks (where N is the degree of associativity



Strategy of block Replacement

- Several different replacement policies can be used
  - **Random replacement** - *randomly pick any block*
    - Easy to implement in hardware, just requires a random number generator
    - Spreads allocation uniformly across cache
    - May evict a block that is about to be accessed
  - **Least-recently used (LRU)** - *pick the block in the set which was least recently accessed*
    - Assumed more recently accessed blocks more likely to be referenced again
    - This requires extra bits in the cache to keep track of accesses.
  - **First in,first out(FIFO)**-*Choose a block from the set which was first came into the cache*

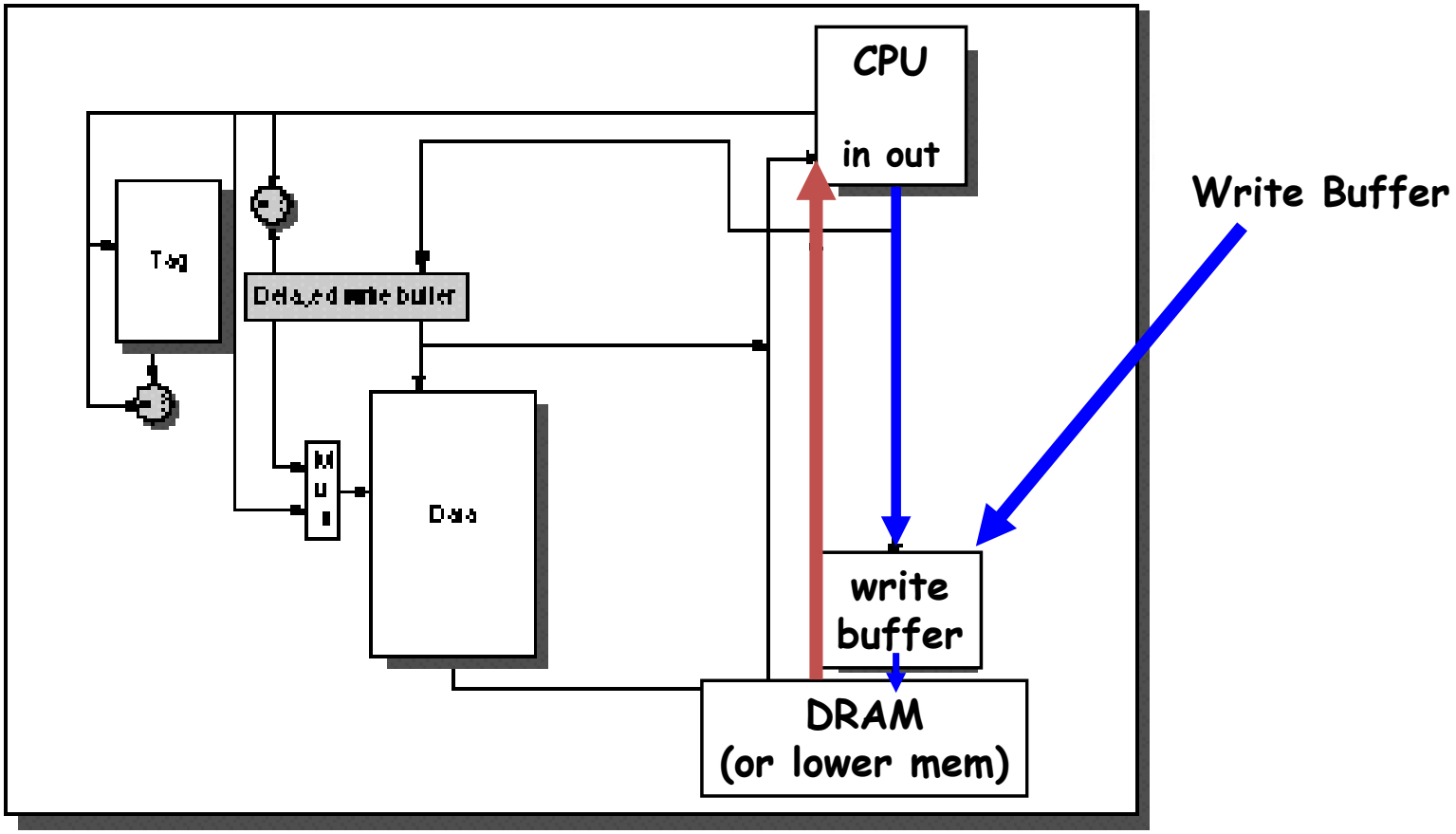
Q4: Write Strategy

- 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.

Write stall

- **Write stall --**When the CPU must wait for writes to complete during write through
- **Write buffers**
  - A small cache that can hold a few values waiting to go to main memory.
  - *To avoid stalling on writes, many CPUs use a write buffer.*
  - This buffer helps when writes are clustered.
  - It does not entirely eliminate stalls since it is possible for the buffer to be full if the burst is larger than the buffer.

Write buffers



Write misses

Write misses

- ***If a miss occurs on a write (the block is not present), there are two options.***
- *Write allocate*
  - ***The block is loaded into the cache on a miss before anything else occurs.***
- *Write around (no write allocate)*
  - ***The block is only written to main memory***
  - ***It is not stored in the cache.***

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

7.3 Measuring and improving cache performance

- In this section, we will discuss two questions:
  1. How to measure cache performance?
  2. How to improve performance?

- The main contents are the following:
  1. Measuring cache performance
  2. Reducing cache misses by more flexible placement of blocks
  3. Reducing the miss penalty using multilevel caches

Average Memory Access time = hit time + miss time

= hit rate × Cache time + miss rate ×memory time

= 99% × 5 + (1-99%) × 45 =5.5ns

Measuring cache performance

- We use CPU time to measure cache performance.

CPU time= $CPU_{time} = I \times CPI \times Clock\ \ cycle\ time$

(CPU execution clock cycles + Memory-stall clock cycles) × Clock cycle time

Memory-stall clock cycles = # of instructions × miss ratio × miss penalty

= Read-stall cycles + Write-stall cycles

For Read-stall

$$\text{Read-stall cycles} = \frac{\text{Read}}{\text{Program}} \times \text{Read miss rate} \times \text{Read miss penalty}$$

- For a write-through plus write buffer scheme:

$$\text{Write-stall cycles} = \left(\frac{\text{write}}{\text{Program}} \times \text{Write miss rate} \times \text{Write miss penalty} \right) + \text{Write buffer stalls}$$

- *If the write buffer stalls are small, we can safely ignore them .*

Combine the reads and writes

- In most write-through cache organizations, the read and write miss penalties are the same
  - the time to fetch the block from memory.
- If we neglect the write buffer stalls, we get the following equation:

$$\text{Memory-stall clock cycles} = \frac{\text{Memory accesses}}{\text{Program}} \times \text{Miss rate} \times \text{Miss penalty}$$

We can also write this as:

$$\text{Memory-stall clock cycles} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Misses}}{\text{Instructions}} \times \text{Miss penalty}$$

Calculating cache performance

- **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 much 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

How faster a processor for ideal

$$\frac{\text{CPU time with stalls}}{\text{CPU time with perfect cache}} = \frac{I \times CPI_{stall} \times \text{Clock cycle}}{I \times CPI_{perfect} \times \text{Clock cycle}}$$

$$= \frac{CPI_{stal}}{CPI^{I}_{perfect}} = \frac{5.44}{2} = 2.72$$

- **What happens if the processor is made faster?**

Assume CPI reduces from 2 to 1

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

=(1+3.44)I = 4.44I

$$\frac{\text{CPU time with stalls}}{\text{CPU time with perfect cache}} = \frac{CPI_{stal}}{CPI^{I}_{perfect}} = \frac{4.44}{1} = 4.44$$

Ratio time for Memory stalls

from $\frac{3.44}{5.44} = 63\%$ to $\frac{3.44}{4.44} = 77\%$

Calculating cache performance with Increased Clock Rate

- *Suppose we increase the performance of the computer in the previous example by doubling its clock rate for same memory system.*
- *Question : How much faster will the computer be with the faster clock?*
- *Answer*

Total miss cycles per instruction = (2%×200) + 36%×(4%×200)=6.88

CPI with cache misses = 2 + 6.88 =8.88

$$\frac{\text{Performance with fast clock}}{\text{Performance with slow clock}} = \frac{\text{Execution time with slow clock}}{\text{Execution time with fast clock}}$$

$$= \frac{IC \times CPI_{\text{slow clock}} \times \text{Clock cycle}}{IC \times CPI_{\text{fast clock}} \times \text{Clock cycle/2}} = \frac{5.44}{8.88 \times 1/2} = 1.23$$

This, the computer with the faster clock is about 1.2 times faster rather than 2 time faster.

Solution 1

Reducing cache misses by more flexible placement of blocks

(1) The disadvantage of a direct-mapped cache

(2) The basics of a set-associative cache

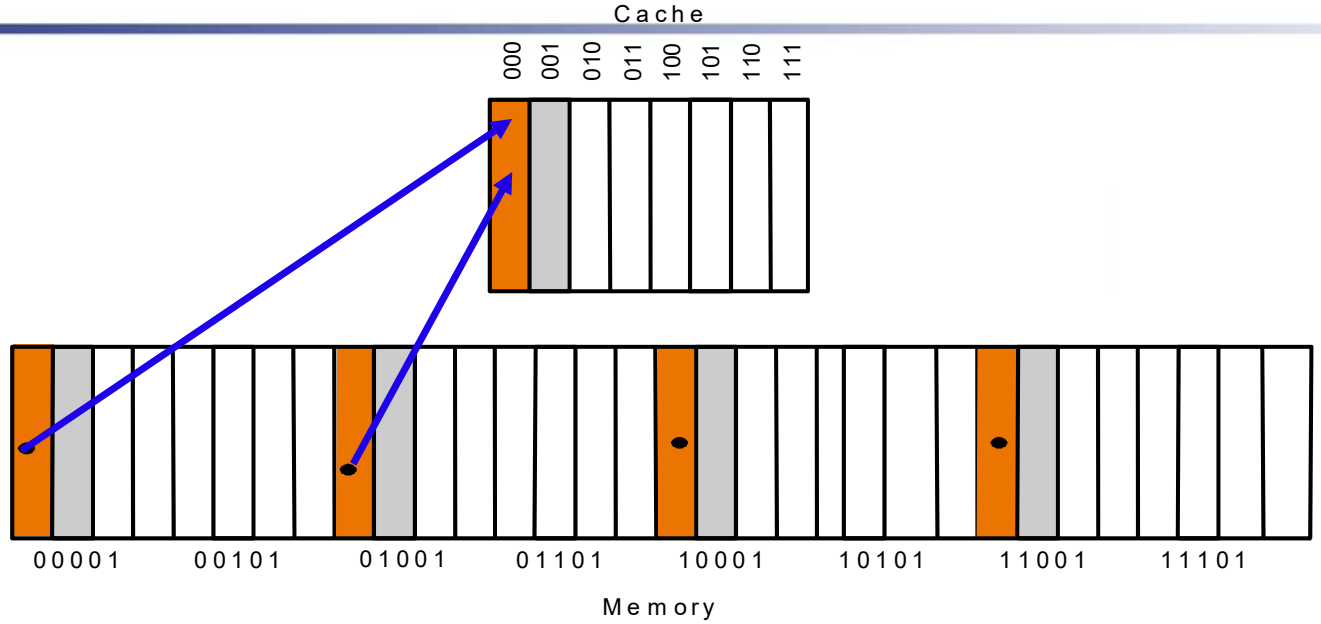
(3) Miss rate versus set-associative

(4) Locating a block in the set-associative cache

(5) Size of tags versus set associative

(6) Choosing which block to replace

The disadvantage of a direct-mapped cache



- If the CPU requires the following memory units sequentially: word 0,word 8 and word 0. Word 0 and word 8 both are mapped to cache block 0, so the third access will be a miss.
- But obviously, if one memory block can be placed in any cache block , the miss can be avoided. So, there is possibility that the miss rate can be improved.

The basics of a set-associative cache

Decreasing miss ratio with associativity

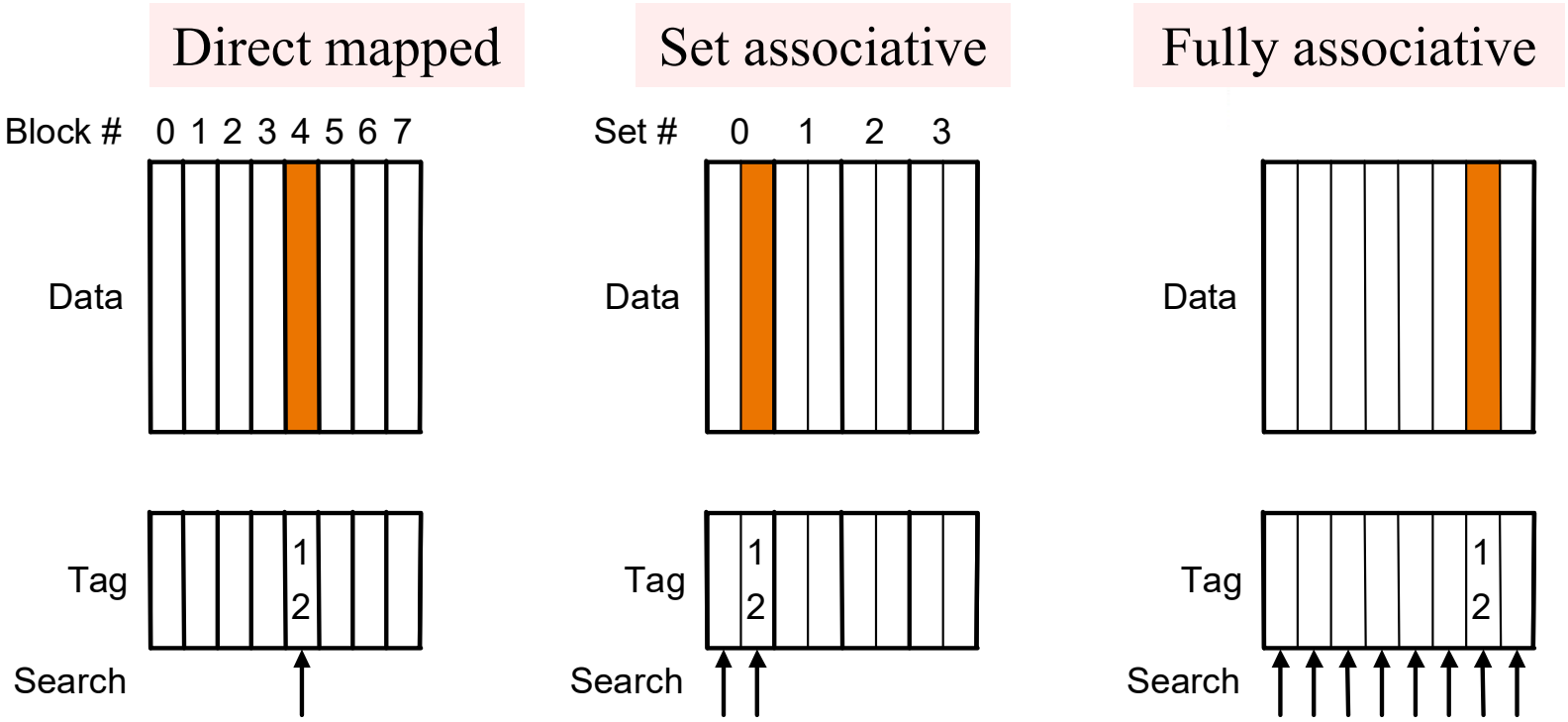
- **A set-associative cache**
  - is divided into some sets. A set contains several blocks.
- **A memory block is mapped to a set in the cache**
  - Through a mapping algorithm.
  - The memory block can be placed in any block in the corresponding set.
- **The mapping algorithm is:** (set with direct-mapped)

Set number (Index) =

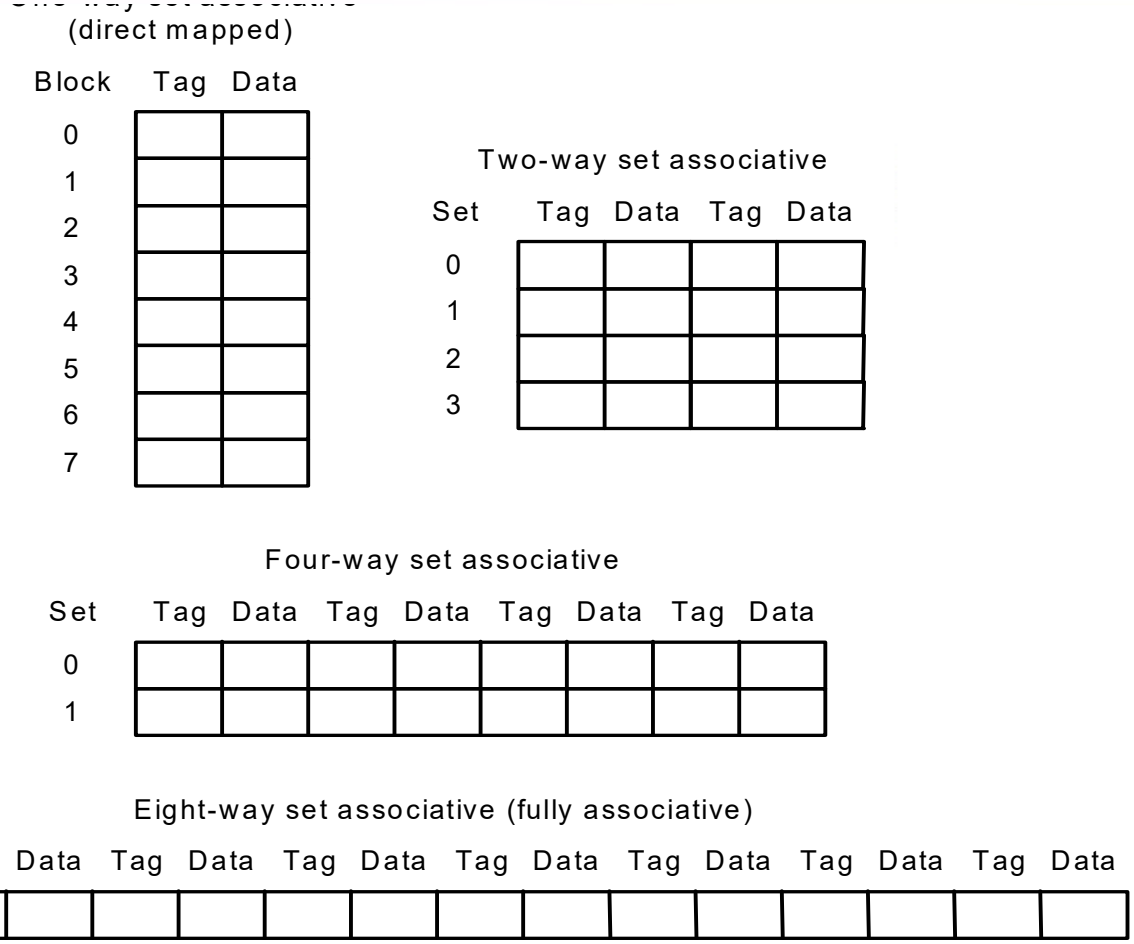
(Memory block number) modulo (Number of sets in the cache)

- If a set has only **one block**, this set-associative cache is actually a **direct-mapped** cache.
- If a set-associative cache has only **one set**, this set-associative cache is called a **fully-associative** cache.

Memory block whose address is 12 in a cache with 8 blocks for different mapped



An eight-block cache configured as variety-way



Miss rate versus set-associativity — 8 Blocks

Assume: there are three small caches, each consisting of four one-word blocks. One cache is direct-mapped, the second is two-way set associative, and the third is fully associative.

Question: Given the following sequence of block addresses:

0, 8, 0, 6, 8, find the number of misses for each cache organization.

Answer: for direct-mapped 5 misses

Memory block	Hit or miss	Contents after each reference			
		Set 0	Set 1	Set 2	Set 3
		Block 0	Block 1	Block 2	Block 3
0	Miss	M[0]			
8	Miss	M[8]			
0	Miss	M[0]			
6	Miss	M[0]		M[6]	
8	Miss	M[8]		M[6]	

Second, for the two-way set associative cache. 4 misses

Memory block	Hit or miss	Contents after each reference			
		Set 0		Set 1	
		Block 0	Block 1	Block 2	Block 3
0	Miss	M[0]			
8	Miss	M[0]	M[8]		
0	Hit	M[0]	M[8]		
6	Miss	M[0]	M[6]		
8	Miss	M[8]	M[6]		

Finally, for the fully associative cache. 3 misses

Memory block	Hit or miss	Contents after each reference			
		Only one set			
		Block 0	Block 1	Block 2	Block 3
0	Miss	M[0]			
8	Miss	M[0]	M[8]		
0	Hit	M[0]	M[8]		
6	Miss	M[0]	M[8]	M[6]	
8	Hit	M[0]	M[8]	M[6]	

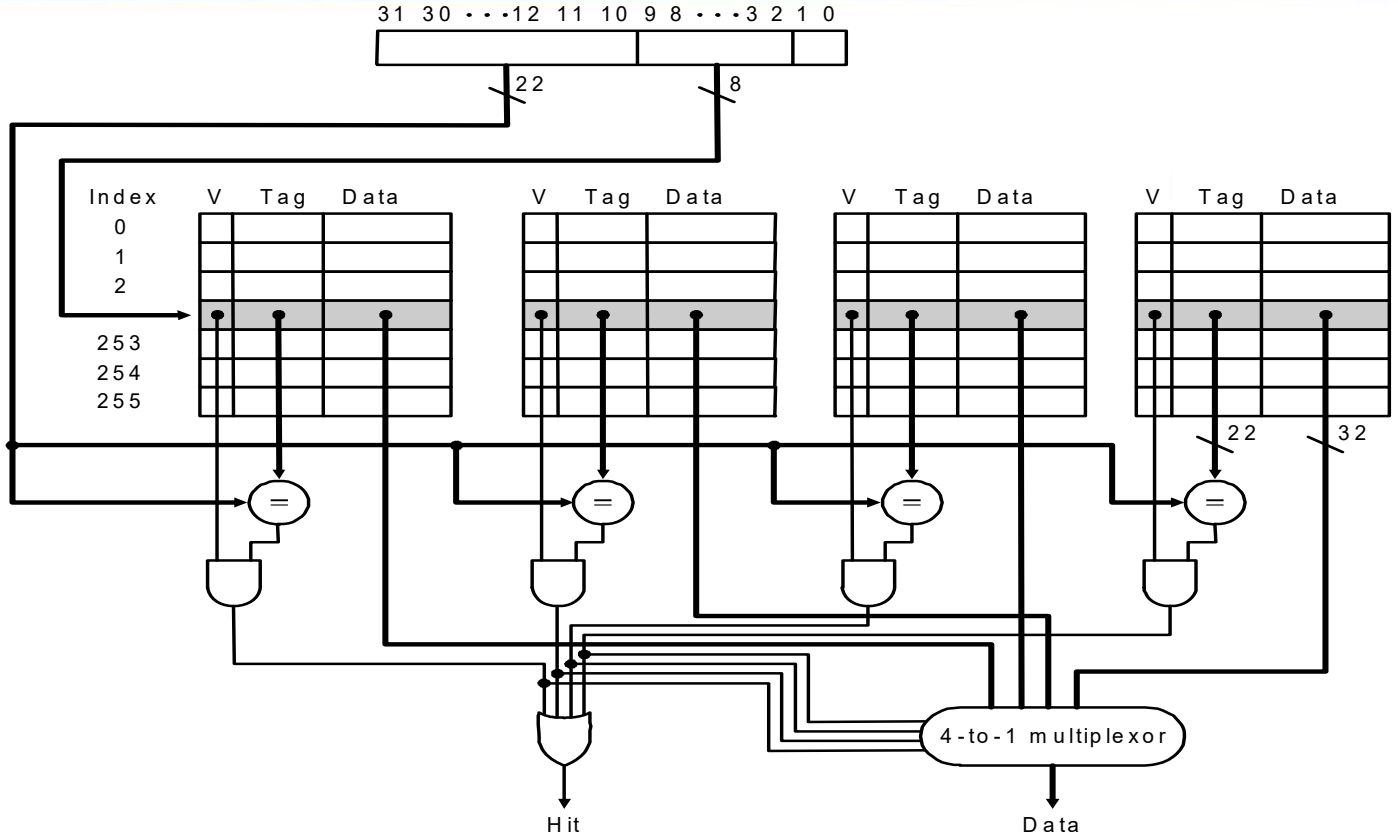
How much of a reduction in the miss rate is achieved by associativity?

Associativity	Data miss rate
1	10.3%
2	8.6%
4	8.3%
8	8.1%

The data cache miss rates for organization like the Intrinsity FastMATH processor for SPEC2000 benchmarks with associativity varying form one-way to eight-way .

- Data cache organization is 64KB data cache and 16-word block

Locating a block in the set-associative cache



- The implementation of a four-way set-associative cache requires four comparators and a 4-to-1 multiplexor.

Size of tags versus set associativity

Assume

Cache has 4K Blocks

Block size is 4 words

Physical address is 32bits

Question

Find the total number of set and total number of tag bits for variety associativity

Answer

Offset size (Byte) = 16= 2^4 — 4 bits for address

Number of memory block = $2^{32} \div 2^4 = 2^{28}$ — 28 bits for Block address

Number of cache block = 2^{12} — 12 bits for Block address

For direct-mapped

Bits of index = 12 bits

bits of Tag = (28-12) ×4K=16×4K=64 Kbits

For two-way associative

Number of cache set = $2^{12} \div 2 = 2^{11}$

Bits of index = 12-1=11 bits

Bits of Tag = (28-11) ×2×2K=17×2×2K=68 Kbits

For four-way associative

Number of cache set = $2^{12} \div 4 = 2^{10}$

Bits of index = 12-2=10 bits

Bits of Tag = (28-10) ×4×1K=18×4×1K=72 Kbits

For full associative

Number of cache set = $2^{12} \div 2^{12} = 2^{0}$

Bits of index = 12-12=0 bits

Bits of Tag = (28-0) ×4K×1=112 Kbits

	Direct	2-way	4-way	Fully
Index(bit)	12	11	10	0
Tag(bit)	16	17	18	28

Choosing which block to replace

- In an associative cache, we must decide which block to replace when a miss happens and the corresponding set is full.
- The most commonly used scheme is **least recently** used (LRU), which we used in the previous example. In an LRU scheme, the block replaced is the one that has been unused for the longest time.
- For a two-way set associative cache, the LRU can be implemented easily. We could keep a single bit in each set. We set the bit whenever a specific block in the set is referenced, and reset the bit whenever another block is referenced.
- As associativity increases, implementing LRU gets harder.

Decreasing miss penalty with multilevel caches

- **Add a second level cache:**
  - often primary cache is on the same chip as the processor
  - use SRAMs to add another cache above primary memory (DRAM)
  - miss penalty goes down if data is in 2nd level cache
- **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:** $\frac{100\text{ns}}{0.2} = 500$ 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

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

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

Total CPI = 1.0 + Primary stalls per instruction + Secondary stalls per instruction

= 1 + 2% ×25 + 0.5% × 500

= 1.0 + 0. 5 +2.5 = 4.0

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

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

- **Using multilevel caches:**
  - **try and optimize the hit time on the 1st level cache**
  - **try and optimize the miss rate on the 2nd level cache**

Multilevel Cache Considerations

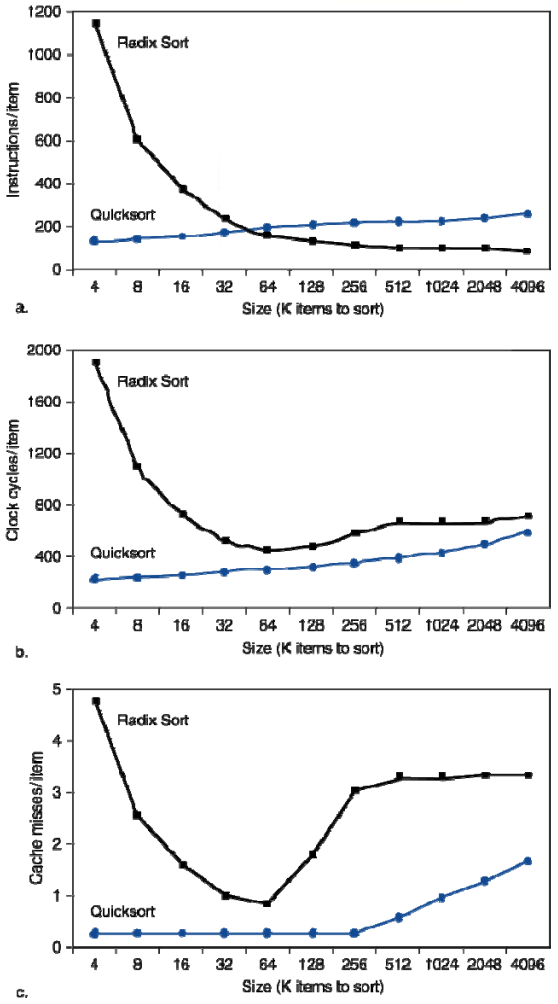
- **Primary cache**
  - Focus on minimal hit time
- **L-2 cache**
  - Focus on low miss rate to avoid main memory access
  - Hit time has less overall impact
- **Results**
  - L-1 cache usually smaller than a single cache
  - L-1 block size smaller than L-2 block size

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



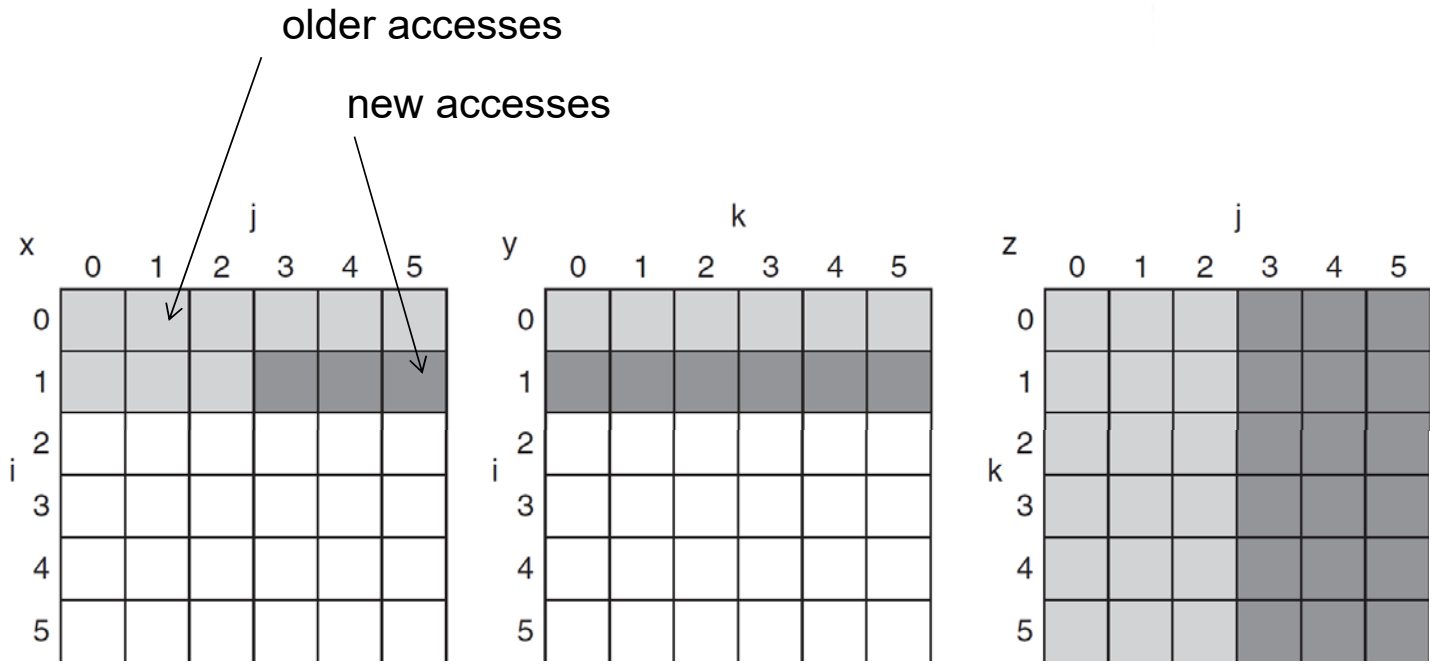
Software Optimization via Blocking

- 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

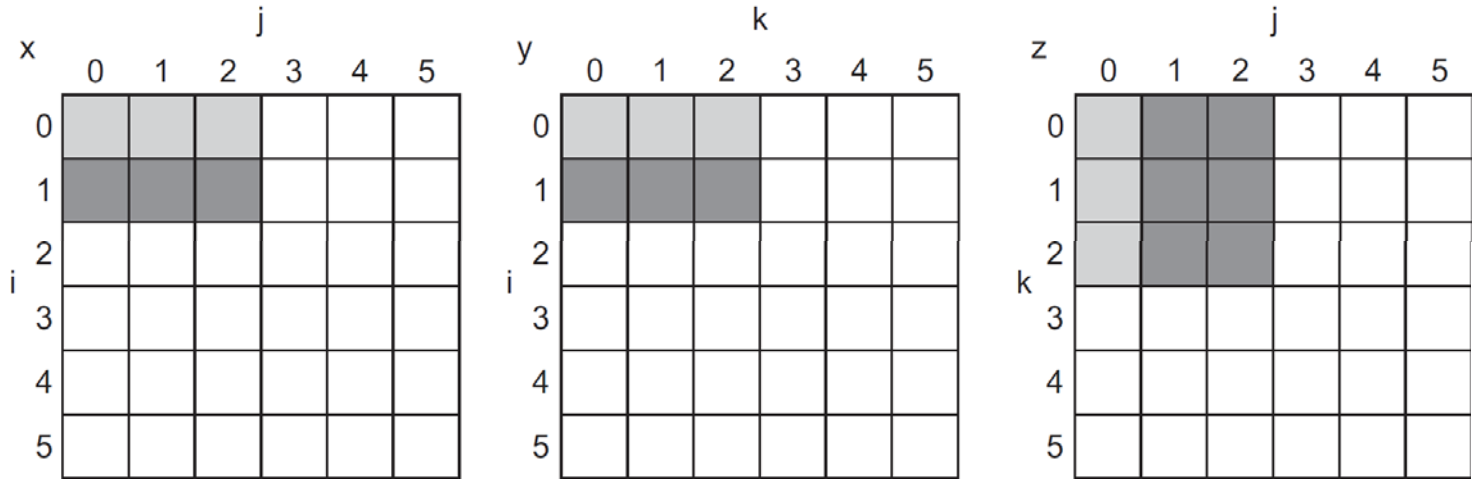
- C, A, and B arrays

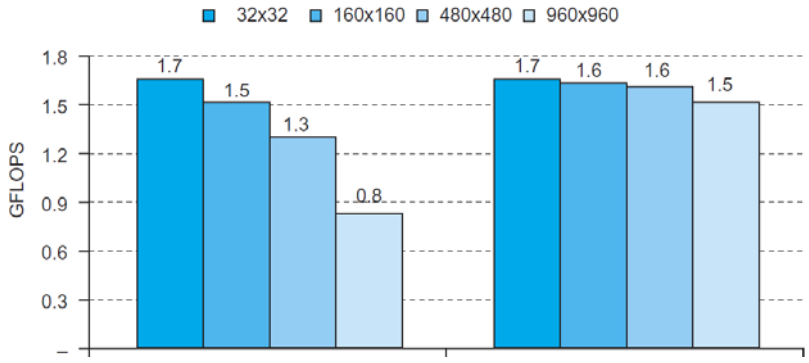


Cache Blocked DGEMM

```
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 }
```

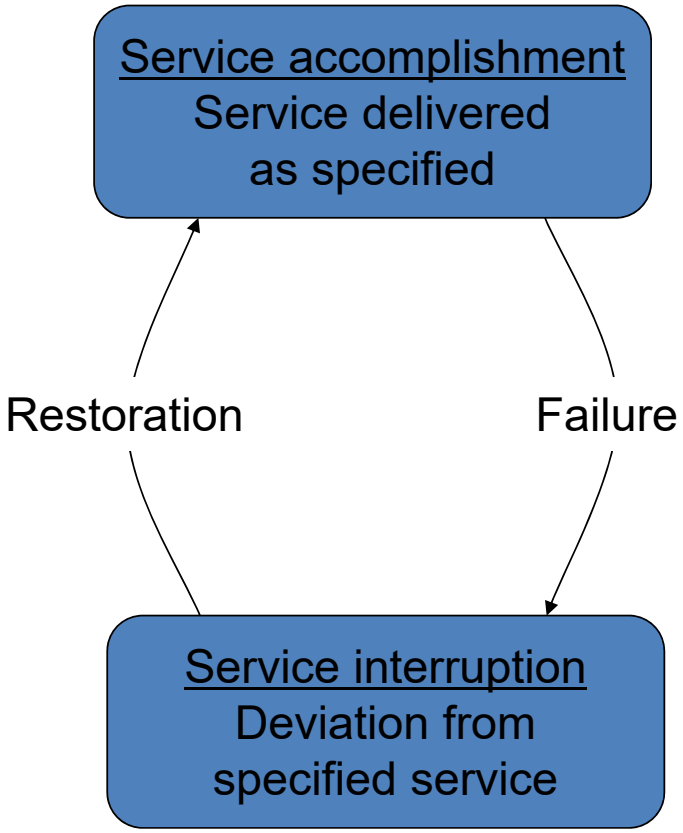
Blocked DGEMM Access Pattern





5.5 Dependable Memory Hierarchy

- **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
- **Availability = MTTF / (MTTF + MTTR)**
- **Improving Availability**
  - Increase MTTF: fault avoidance, fault tolerance, fault forecasting
  - Reduce MTTR: improved tools and processes for diagnosis and repair

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**

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

Decoding SEC

- **Value of parity bits indicates which bits are in error**
  - Use numbering from encoding procedure
  - E.g.
    - Parity bits = 0000 indicates no error
    - Parity bits = 1010 indicates bit 10 was flipped

SEC/DEC Code

- **Add an additional parity bit for the whole word (p_n)**
- **Make Hamming distance = 4**
- **Decoding:**
  - Let H = SEC parity bits
    - H even, p_n even, no error
    - H odd, p_n odd, correctable single bit error
    - H even, p_n odd, error in p_n bit
    - H odd, p_n even, double error occurred
- **Note: ECC DRAM uses SEC/DEC with 8 bits protecting each 64 bits**

5.6 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

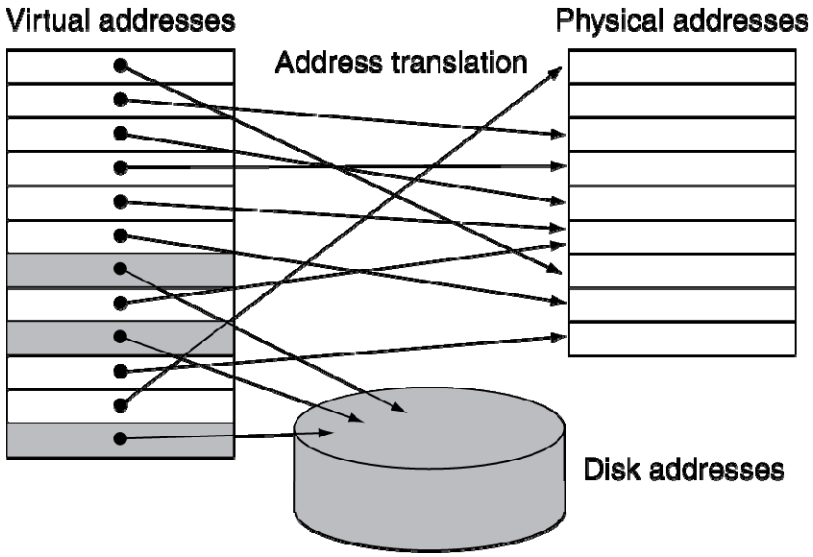
- **User and System modes**
- **Privileged instructions only available in system mode**
  - Trap to system if executed in user mode
- **All physical resources only accessible using privileged instructions**
  - Including page tables, interrupt controls, I/O registers
- **Renaissance of virtualization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

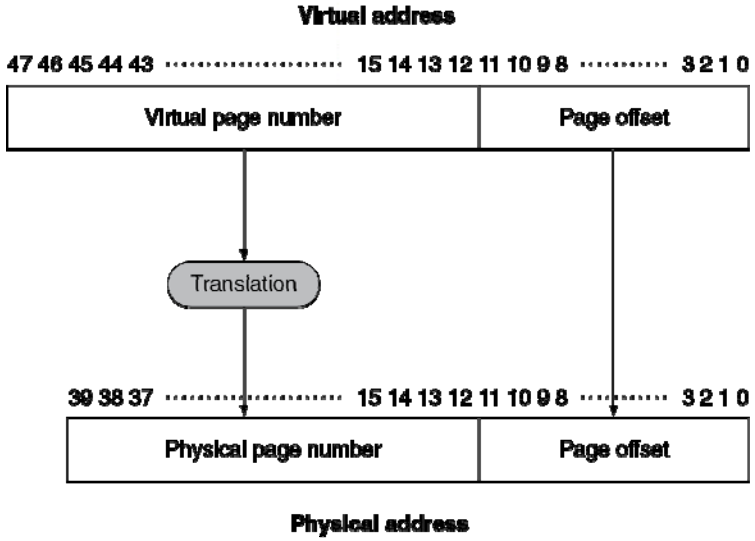
5.7 Virtual Memory

- **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

Address Translation

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





- Pages： virtual memory blocks

Pages Fault

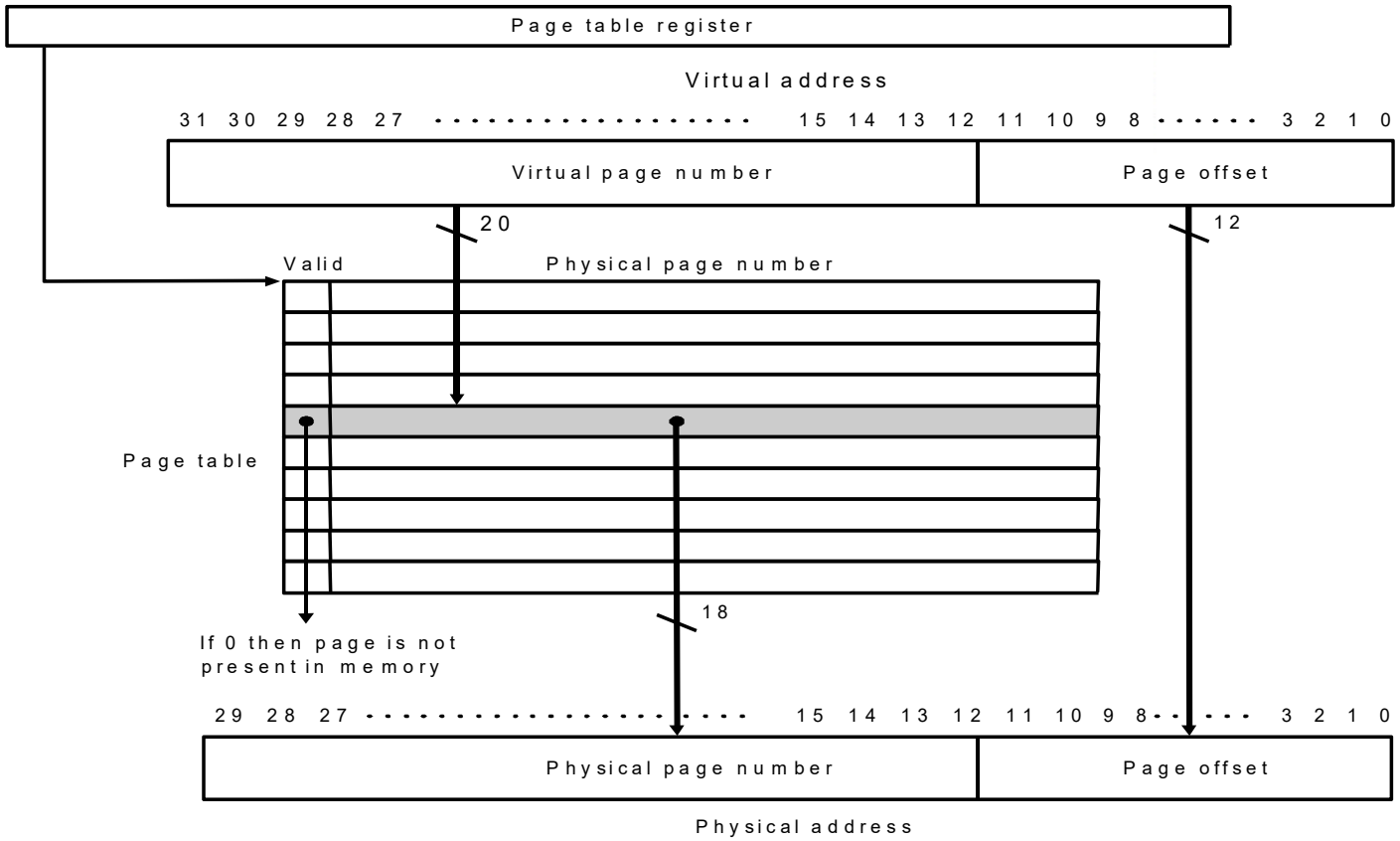
- **Page faults: the data is not in memory, retrieve it from disk**
  - huge miss penalty, thus pages should be fairly large (e.g., 4KB)
  - reducing page faults is important (LRU is worth the price)
  - can handle the faults in software instead of hardware
  - using write-through is too expensive so we use **write back**

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**
  - 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

Placing a page and finding it again --Page Tables

Virtual memory systems use fully associative mapping method



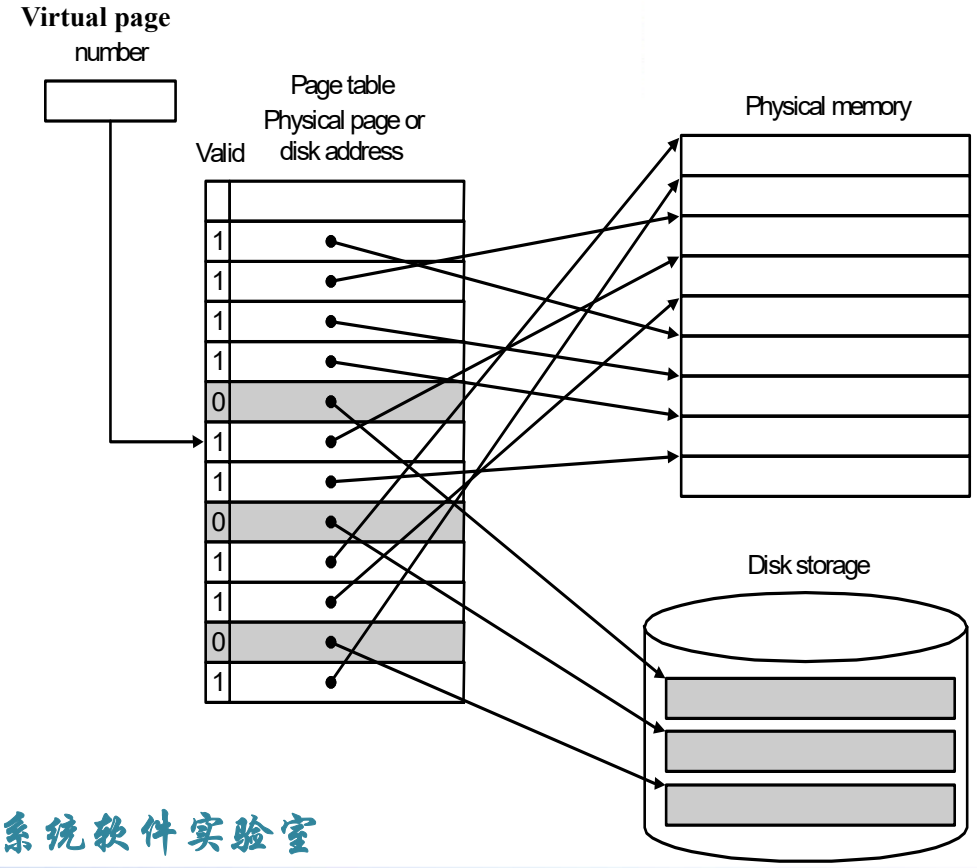
Mapping Pages to Storage

- **When the OS creates a process, it usually creates the space on disk for all the pages of a process.**

- **When a page fault occurs, the OS will be given control through exception mechanism.**

- **The OS will find the page in the disk by the page table.**

- **Next, the OS will bring the requested page into main memory. If all the pages in main memory are in use, the OS will use LRU strategy to choose a page to replace**



How larger page table?

Assume:

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

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

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

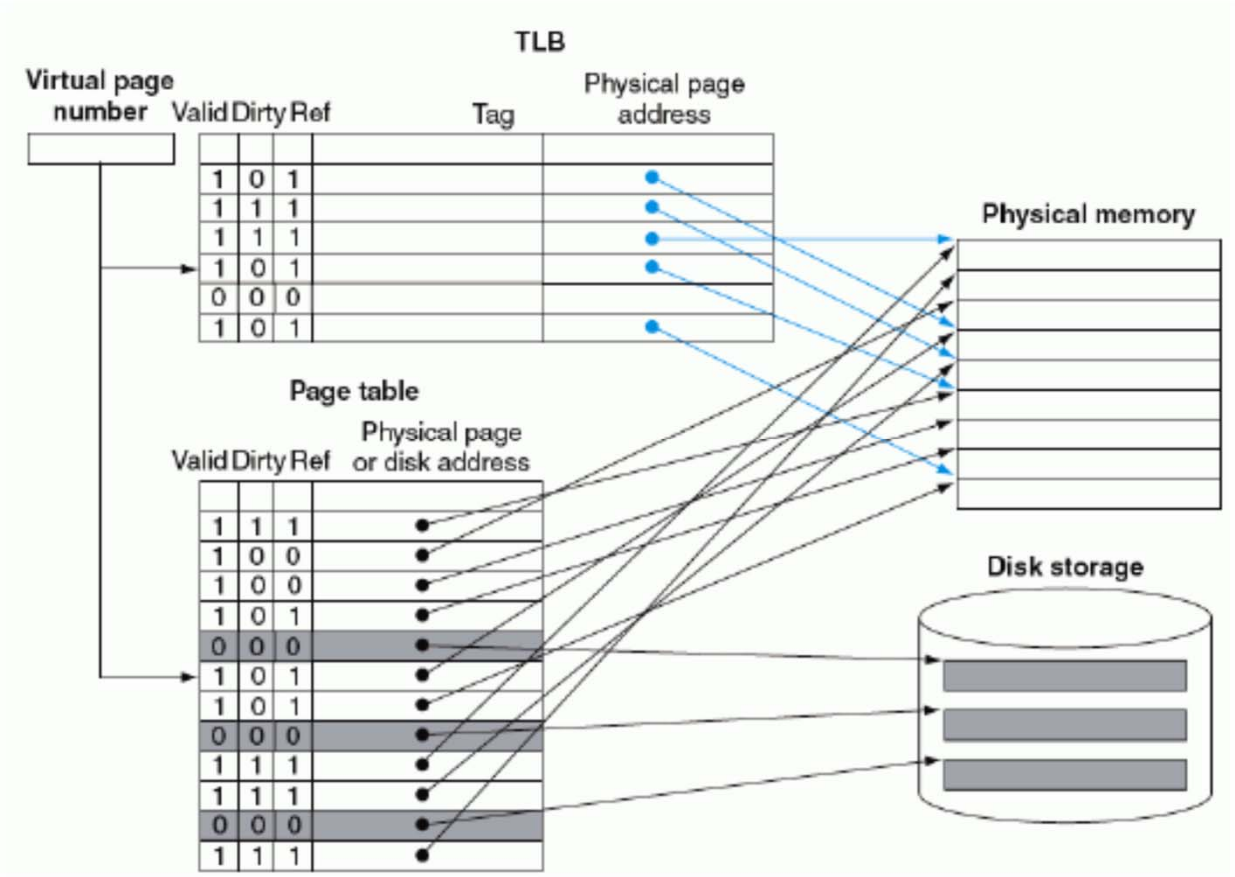
- **Five techniques is used to reduce page table size**
  - ***Section 5.7.3***

Replacement and Writes

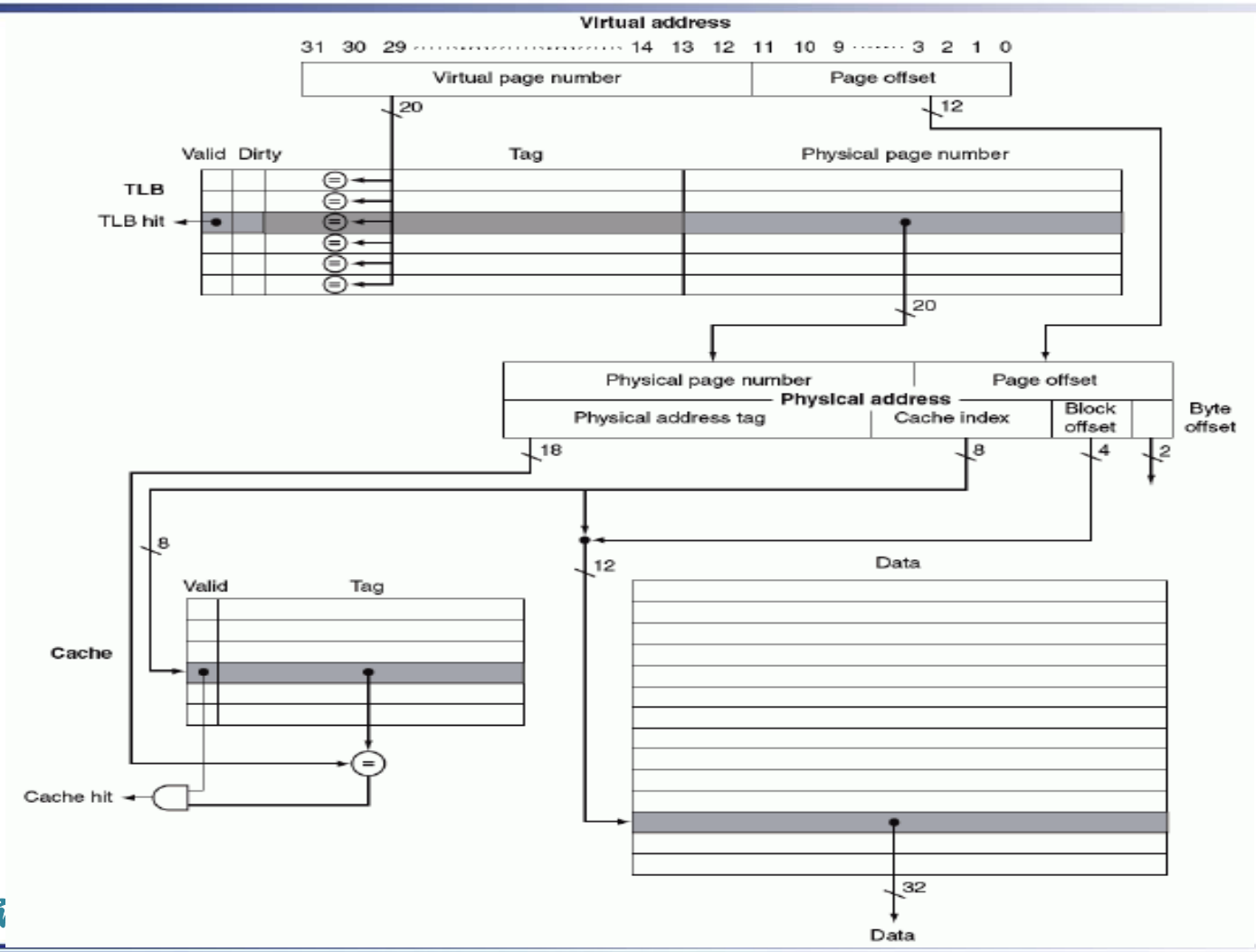
- **To reduce page fault rate, prefer least-recently used (LRU) replacement**
  - Reference bit (aka use bit) in PTE set to 1 on access to page
  - Periodically cleared to 0 by OS
  - A page with reference bit = 0 has not been used recently
- **Disk writes take millions of cycles**
  - Block at once,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

Making Address Translation Fast--TLB

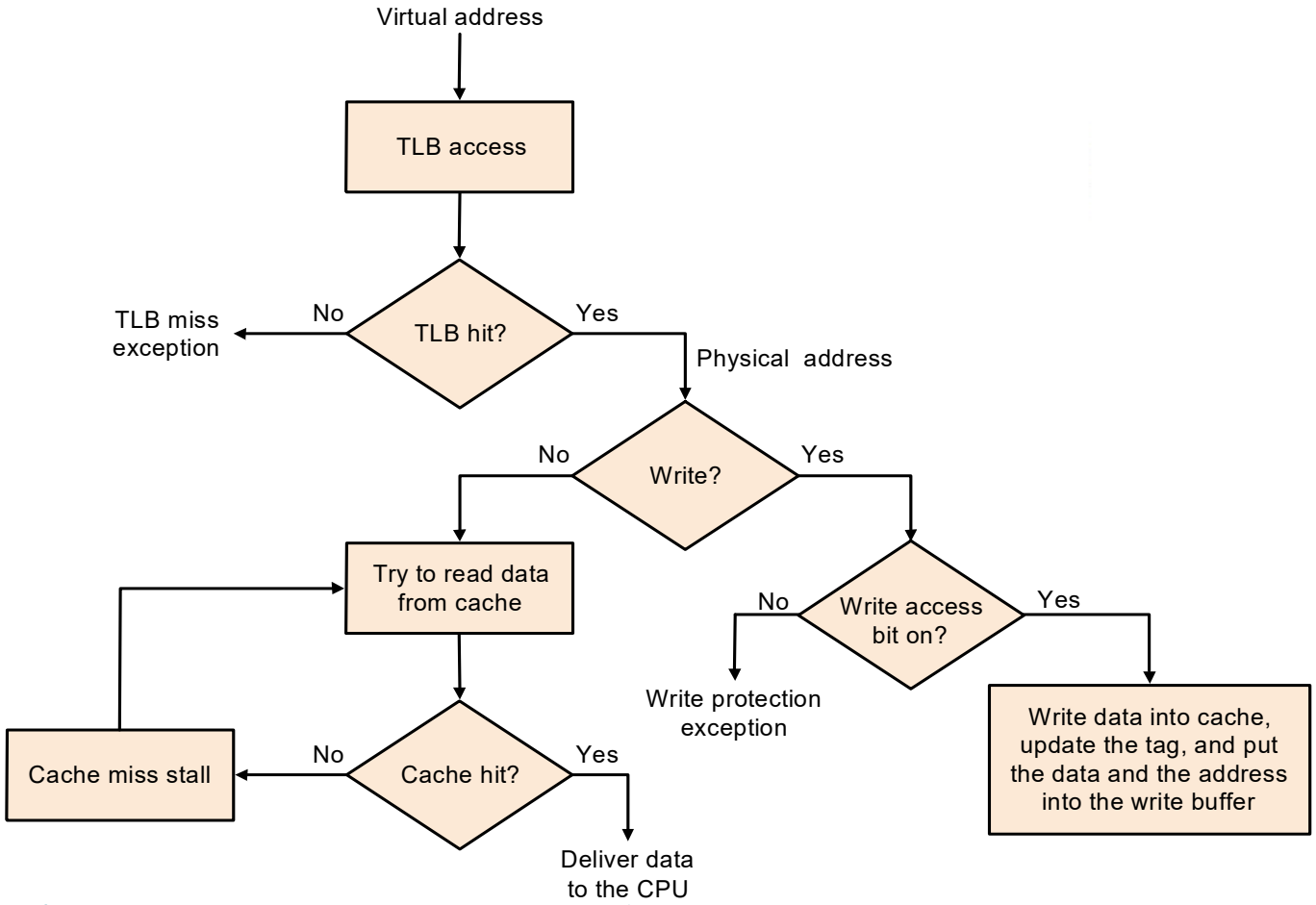
- **Address translation would appear to require extra memory references**
  - One to access the PTE
  - Then the actual memory access
- **But access to page tables has good locality**
  - So use a fast cache of PTEs within the CPU
  - Called a Translation Look-aside Buffer (TLB)
  - Typical: 16–512 PTEs,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



FastMATH Memory Hierarchy



TLBs and caches



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

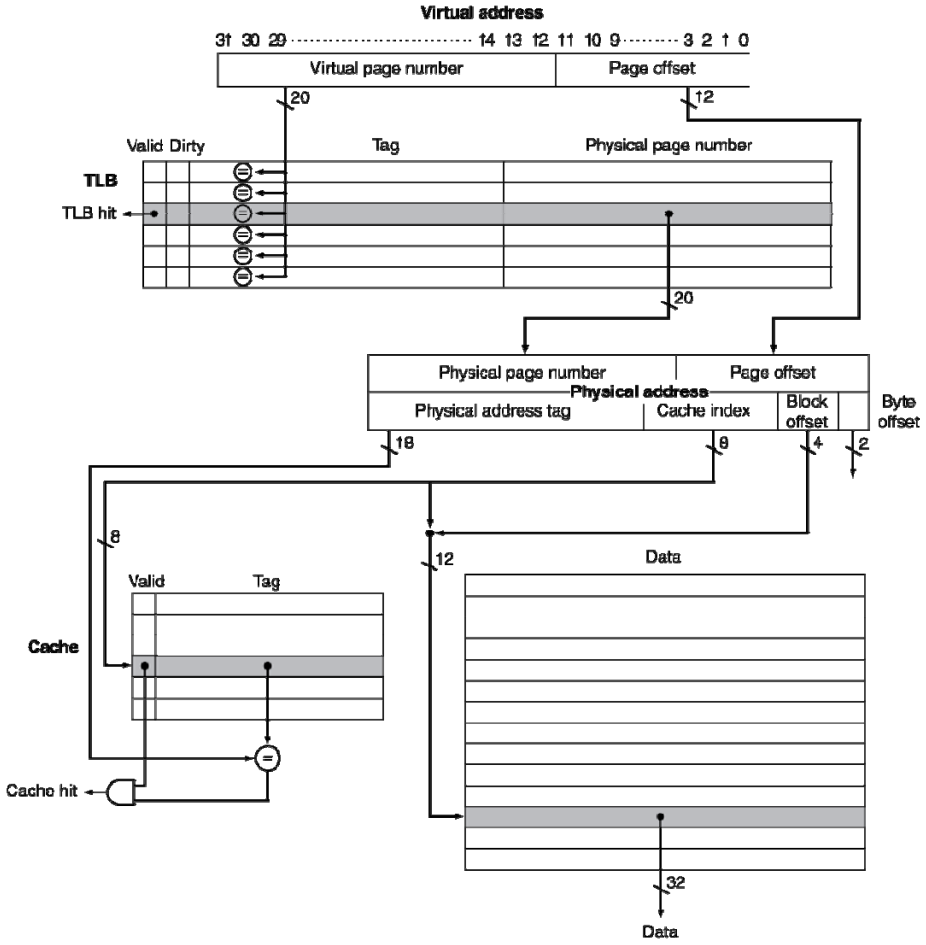
TLB Miss Handler

- **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 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

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., ecall in RISC-V)

5.8 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

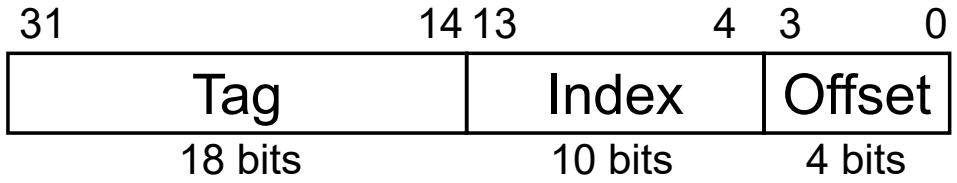
Cache Design Trade-offs

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.

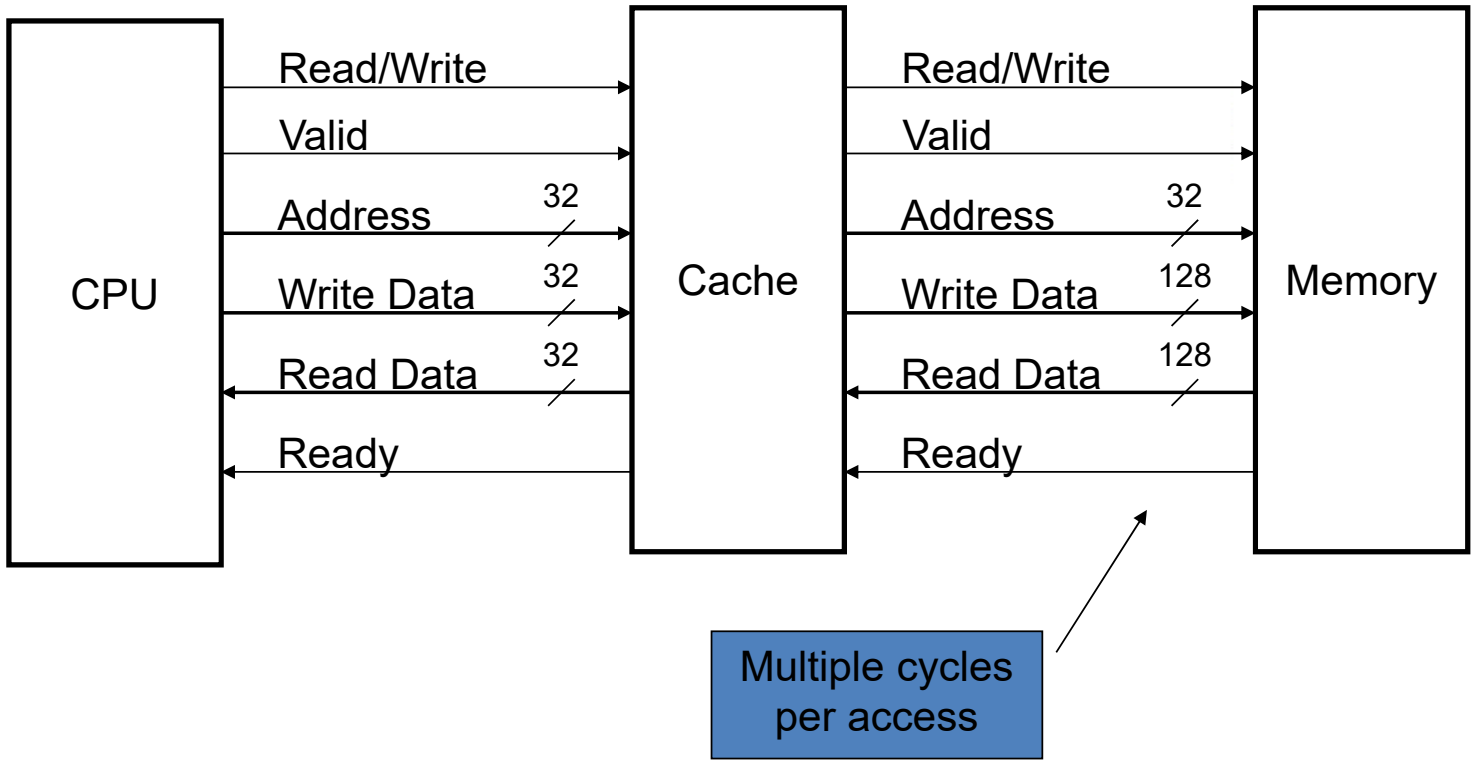
Using a Finite State Machine to Control A Simple Cache

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

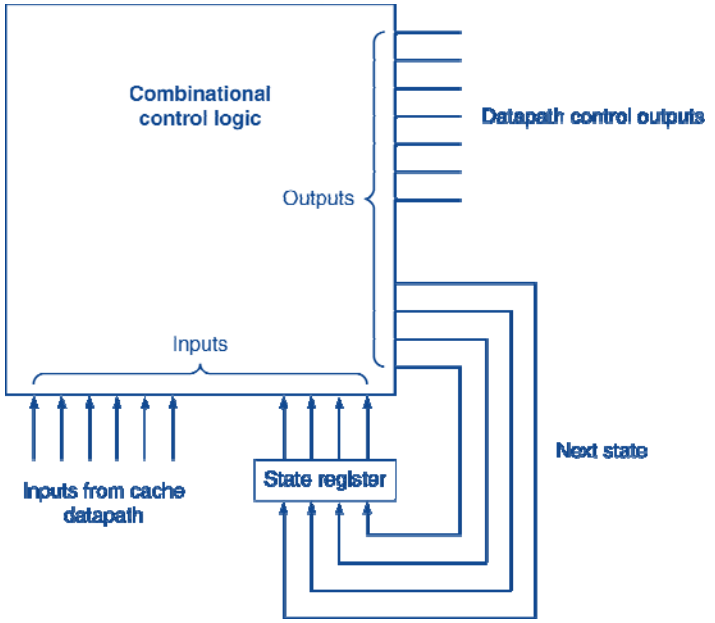


Interface Signals

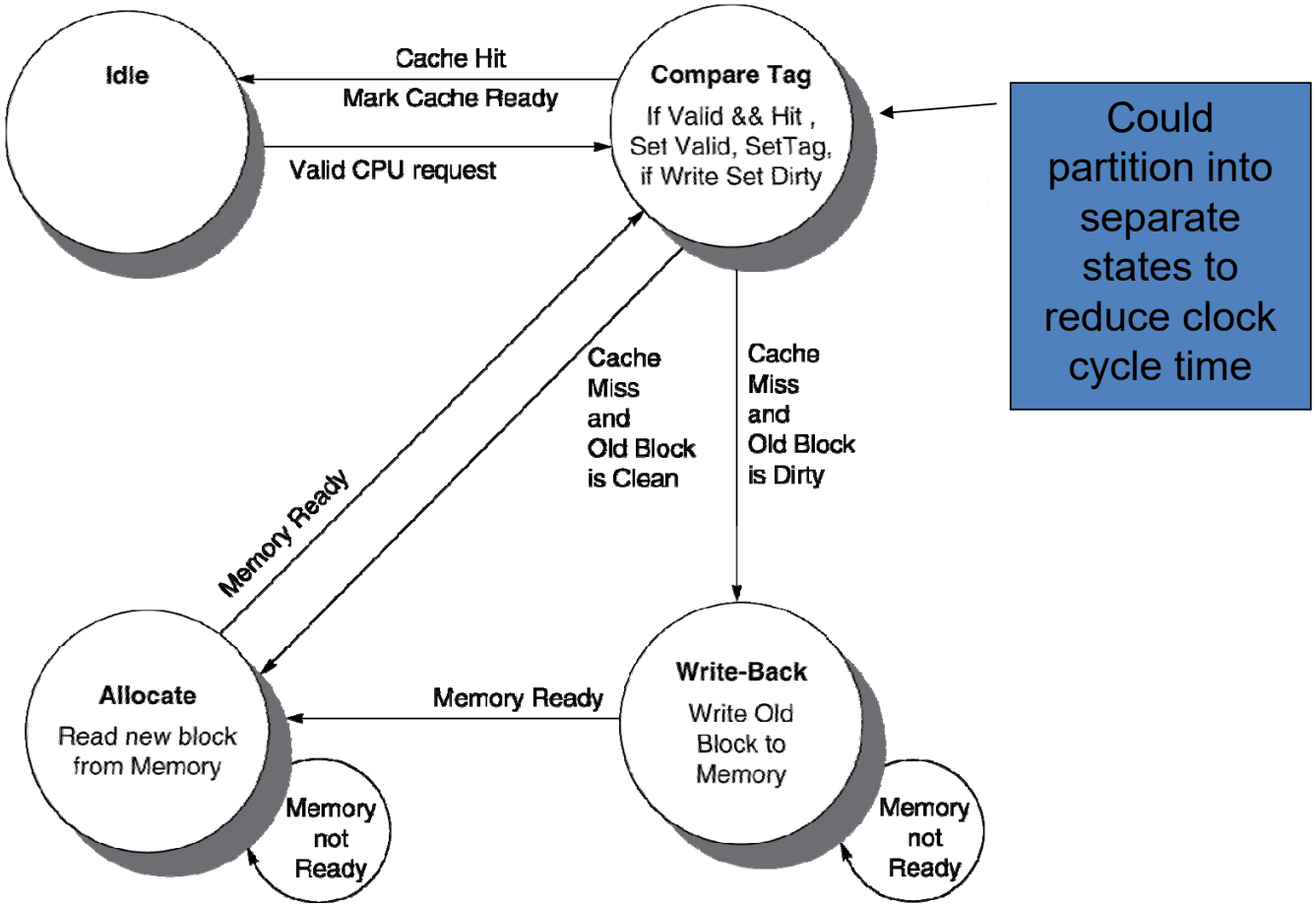


Finite State Machines

- **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



Assignment

Reading: chapter 5

Problems:

5.2.2, 5.3.1, 5.5, 5.10, 5.11.2, 5.13, 5.16.1, 5.17


END