浙江大学
ZHEJIANG UNIVERSITY

计算机组成与设计

Computer Organization & Design

The Hardware/Software Interface

Chapter 5

Large and Fast:

Exploiting Memory Hierarchy

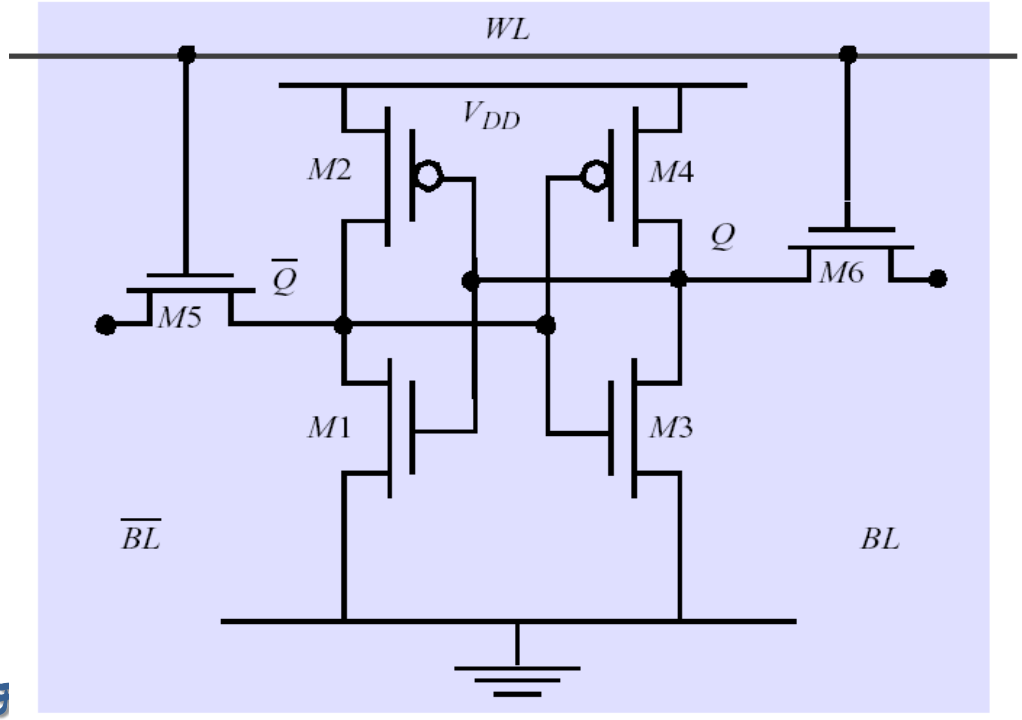
马德
College of Computer Science and Technology
Zhejiang University
made@zju.edu.cn

Contents of Chapter 5

5.1 Memory Technologies

Memories: Review

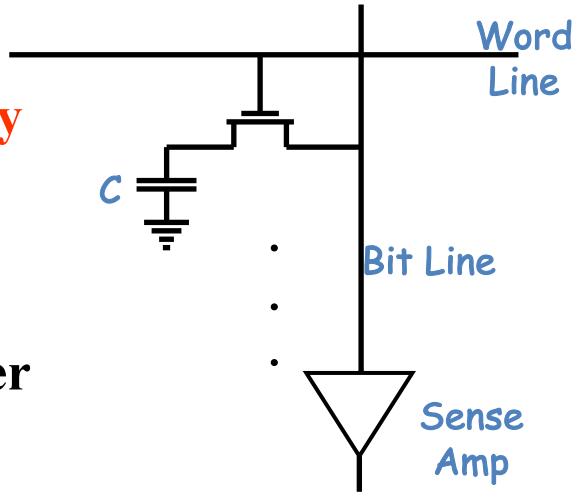
- SRAM
  - value is stored on a pair of inverting gates
  - very fast but takes up more space than DRAM (4 to 6 transistors)



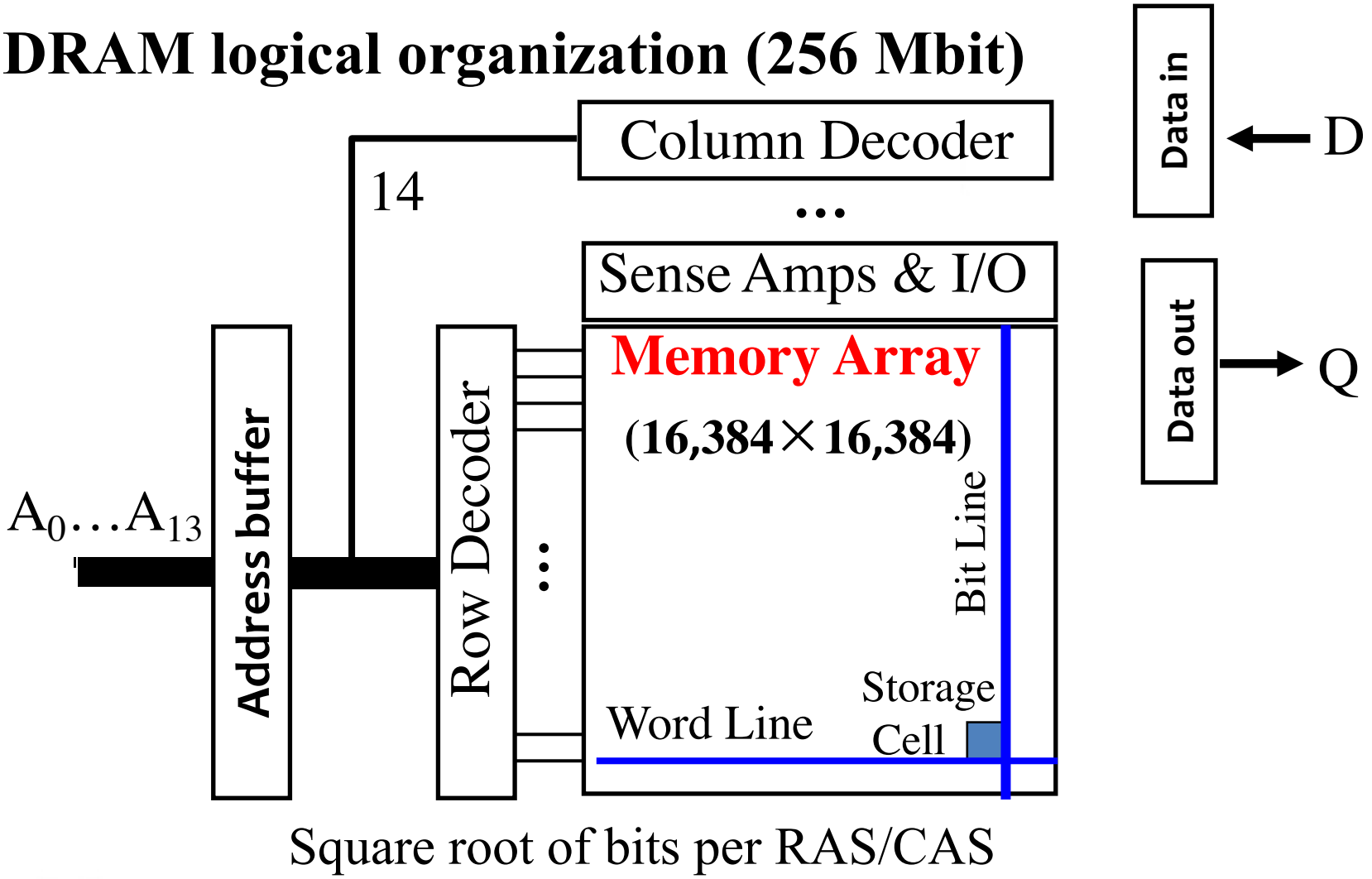
Memories: Review

DRAM:

- value is stored as a charge on capacitor (must be refreshed)
- very small but slower than SRAM (factor of 5 to 10)
- Write
  - **Charge bitline HIGH or LOW and set wordline HIGH**
- Read
  - **Bit line is precharged to a voltage halfway between HIGH and LOW, and then the word line is set HIGH.**
  - **Depending on the charge in the cap, the precharged bitline is pulled slightly higher or lower.**
  - **Sense Amp Detects change**



DRAM logical organization (256 Mbit)



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

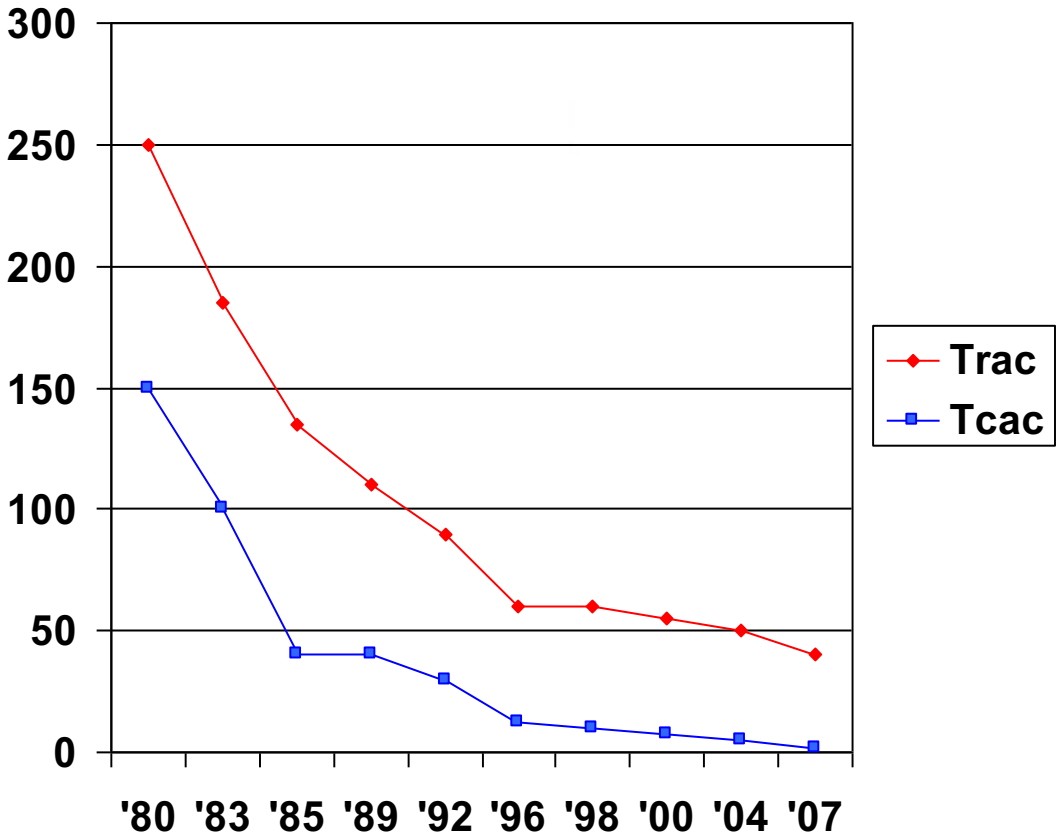
DRAM developed

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
2004	1024Mbit	$0.10	45ns	3ns
2012	4G bit	$0.05	35ns	0.8ns

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

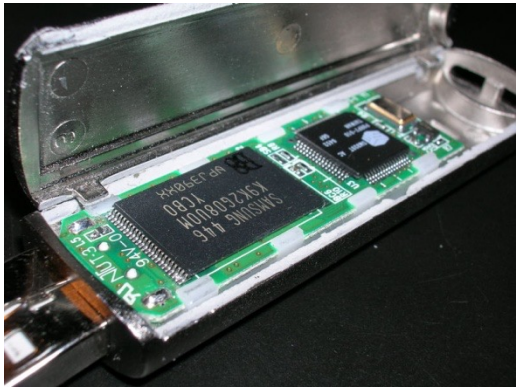
DRAM Generations

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



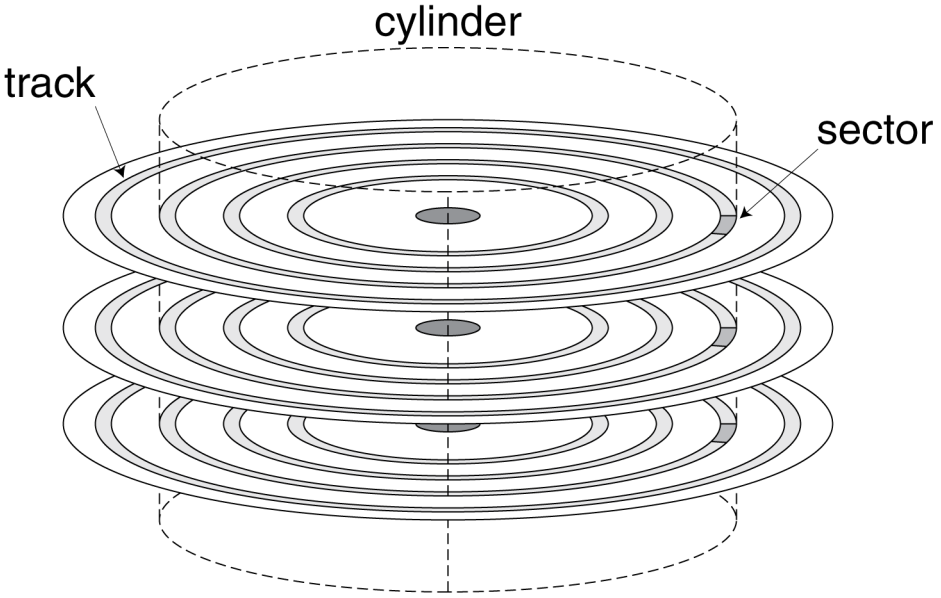
Flash Storage

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



Disk Storage

- Nonvolatile, rotating magnetic storage



Problems in memory designing

- In fact

Memory technology	Typical access time	Cost per GByte (2012)
SRAM	0.5-2.5ns	$2000-$5,000
DRAM	50-70ns	$20-$75
Magnetic disk	5,000,000-20,000,000ns	$0.2-$2

- Users want large and fast memories!

5.2 Memory Hierarchy Introduction

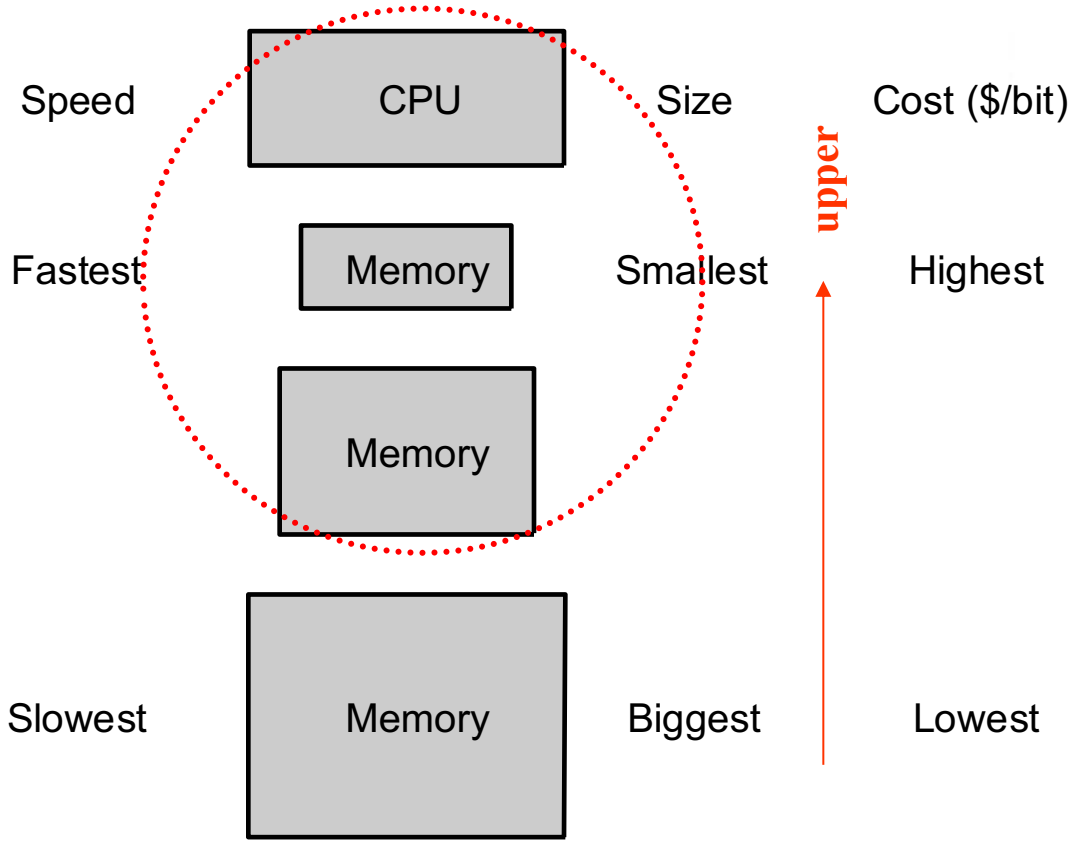
- **Programs access a small proportion of their address space at any time**
- **Temporal locality**
  - Items accessed recently are likely to be accessed again soon
  - e.g., instructions in a loop, induction variables
- **Spatial locality**
  - Items near those accessed recently are likely to be accessed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

Taking Advantage of Locality

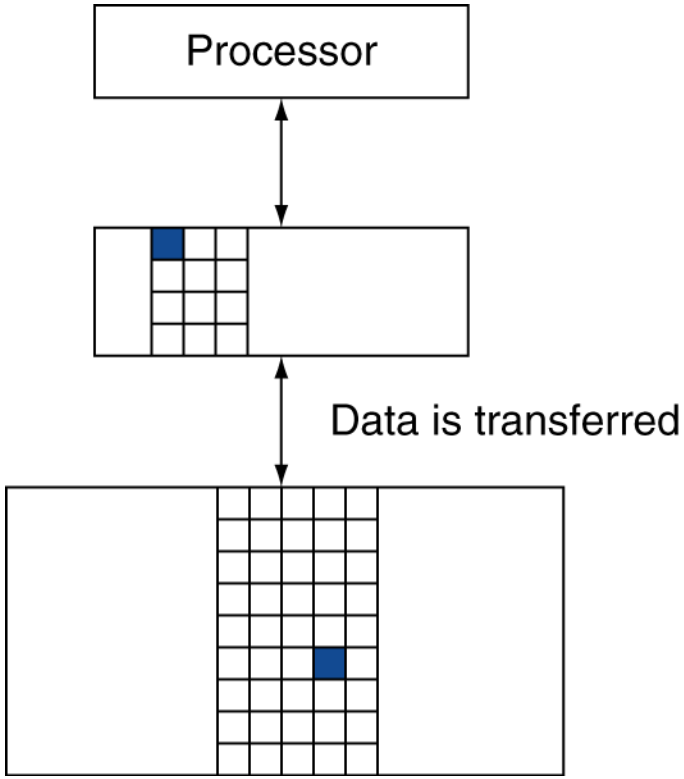
- Memory hierarchy
- Store everything on disk
- Copy recently accessed (and nearby) items from disk to smaller DRAM memory
  - Main memory
- Copy more recently accessed (and nearby) items from DRAM to smaller SRAM memory
  - Cache memory attached to CPU

Memory Hierarchy Levels

- **Build a memory hierarchy**



Some important items(1)



- **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

Some important items(2)

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.

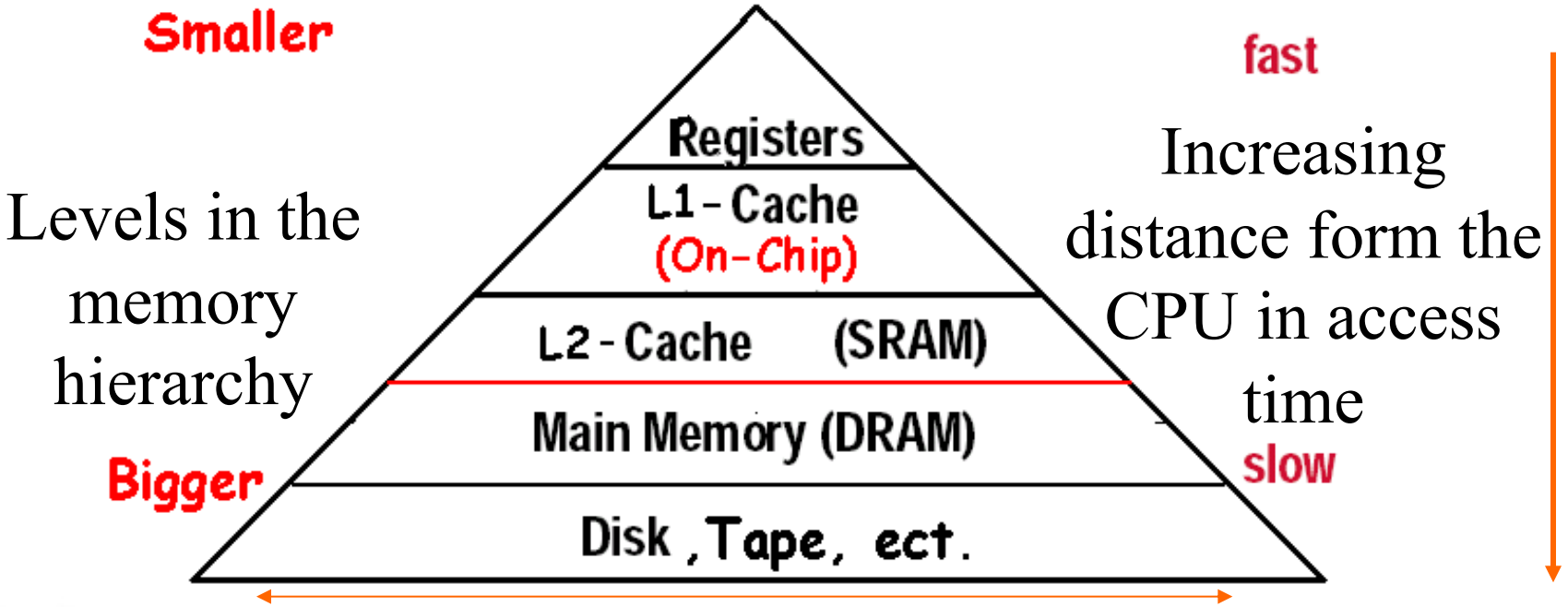
miss penalty:

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

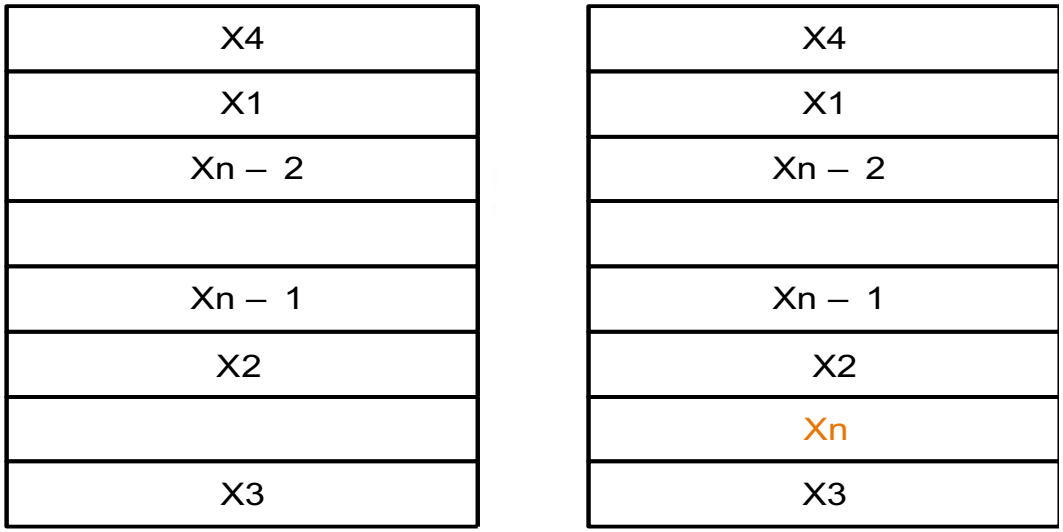
The solution is in size

5.3 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

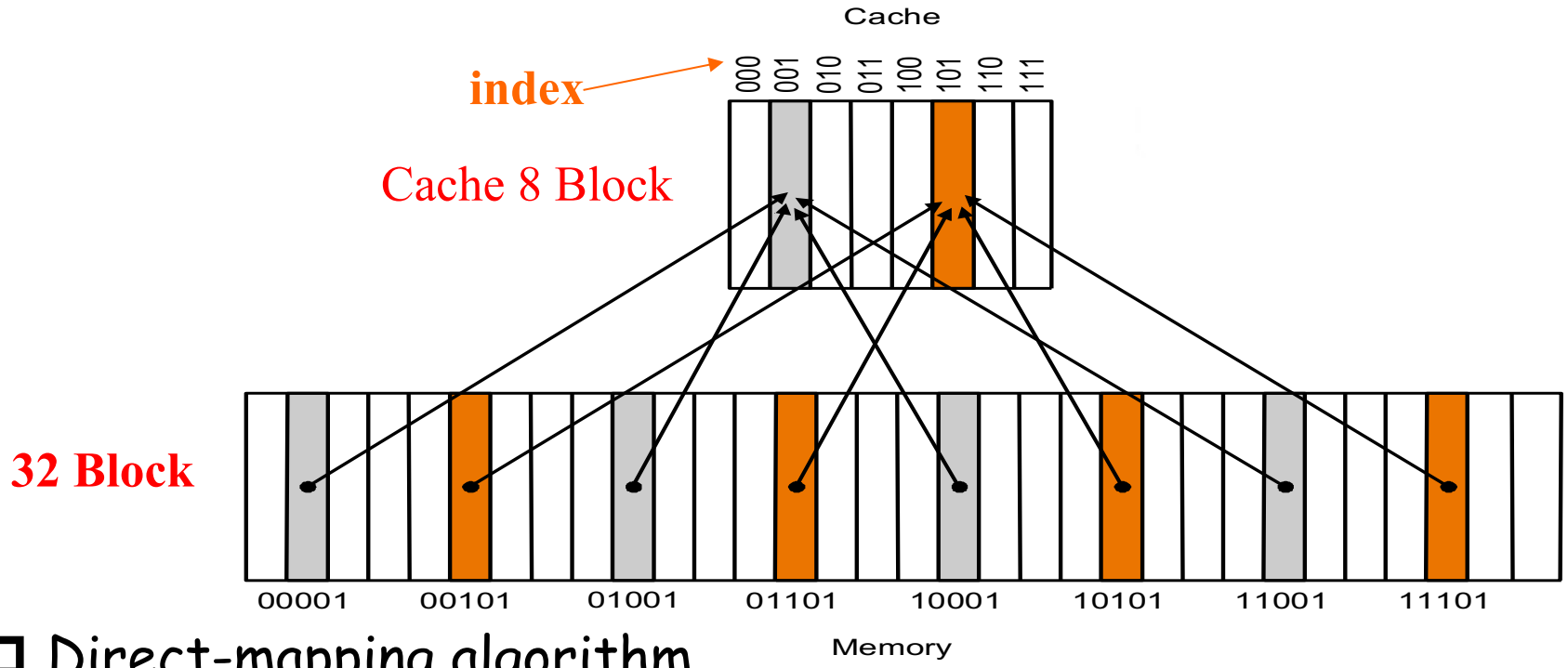
b. After the reference to Xn

- **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?



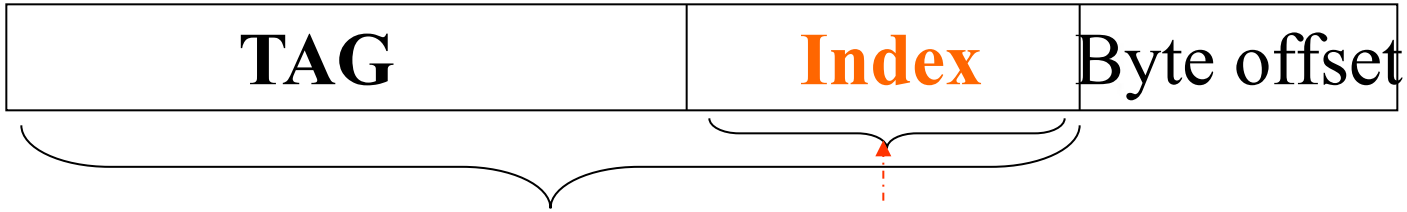
- Direct-mapping algorithm.
  memory address is modulo the number of blocks in the cache
- Fortunately, while the cache has 2^n blocks, the corresponding index is equal to the lowest n bits of memory block address. Here n=3. Let's check

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

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

Cache Example

- 8-blocks, 1 word/block, direct mapped
- Access Sequence:
  10110,11010,10110,11010,10000,00011,10000,10010
- Initial state

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

Cache Example

After Accessing 10110

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

After Accessing 11010

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

After Accessing 10110, 11010

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		

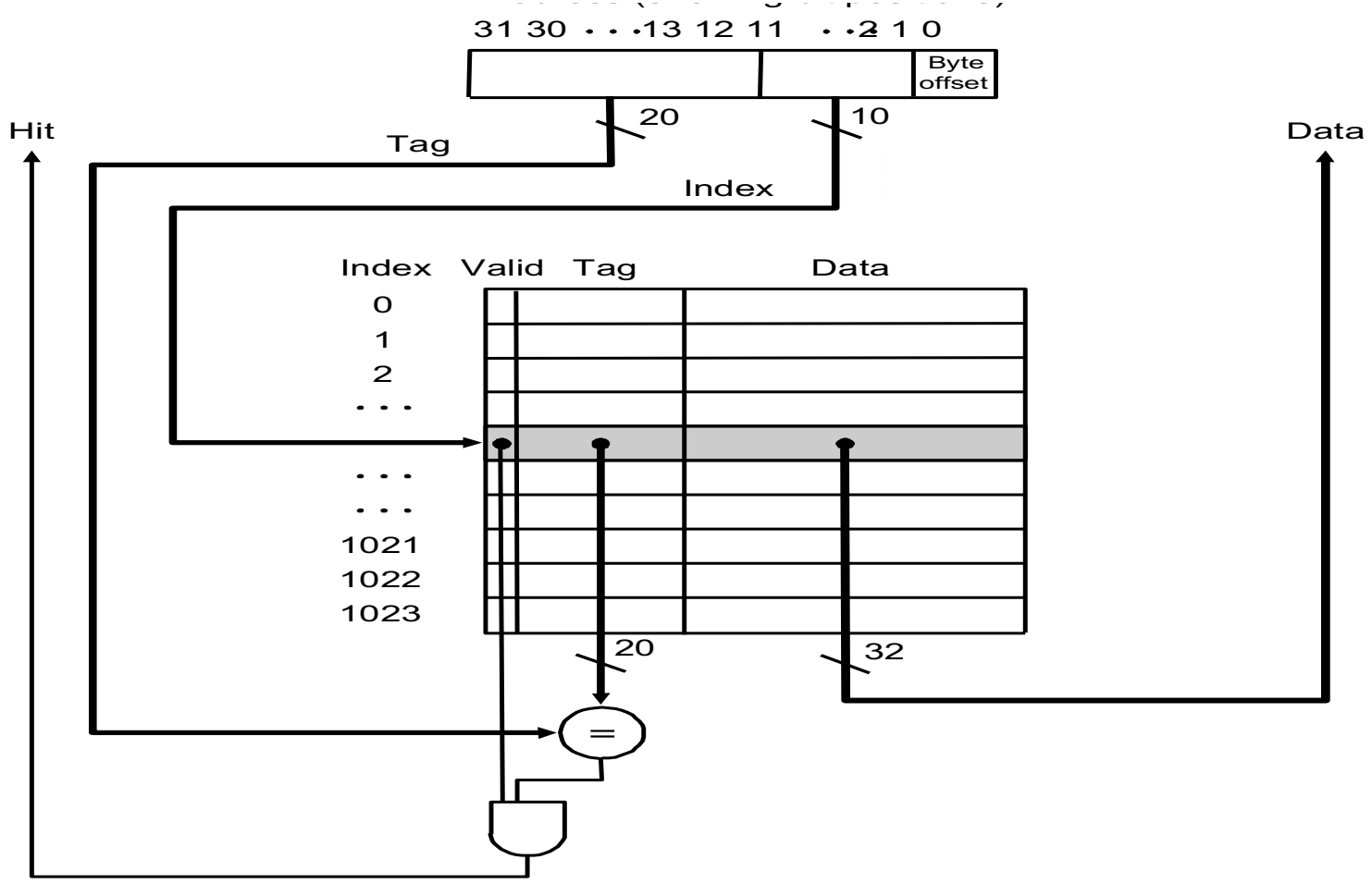
Cache Example

After accessing 10010

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		

Direct Mapped Cache construction



What kind of locality are we taking advantage of?

temporal

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?

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?

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

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 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

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

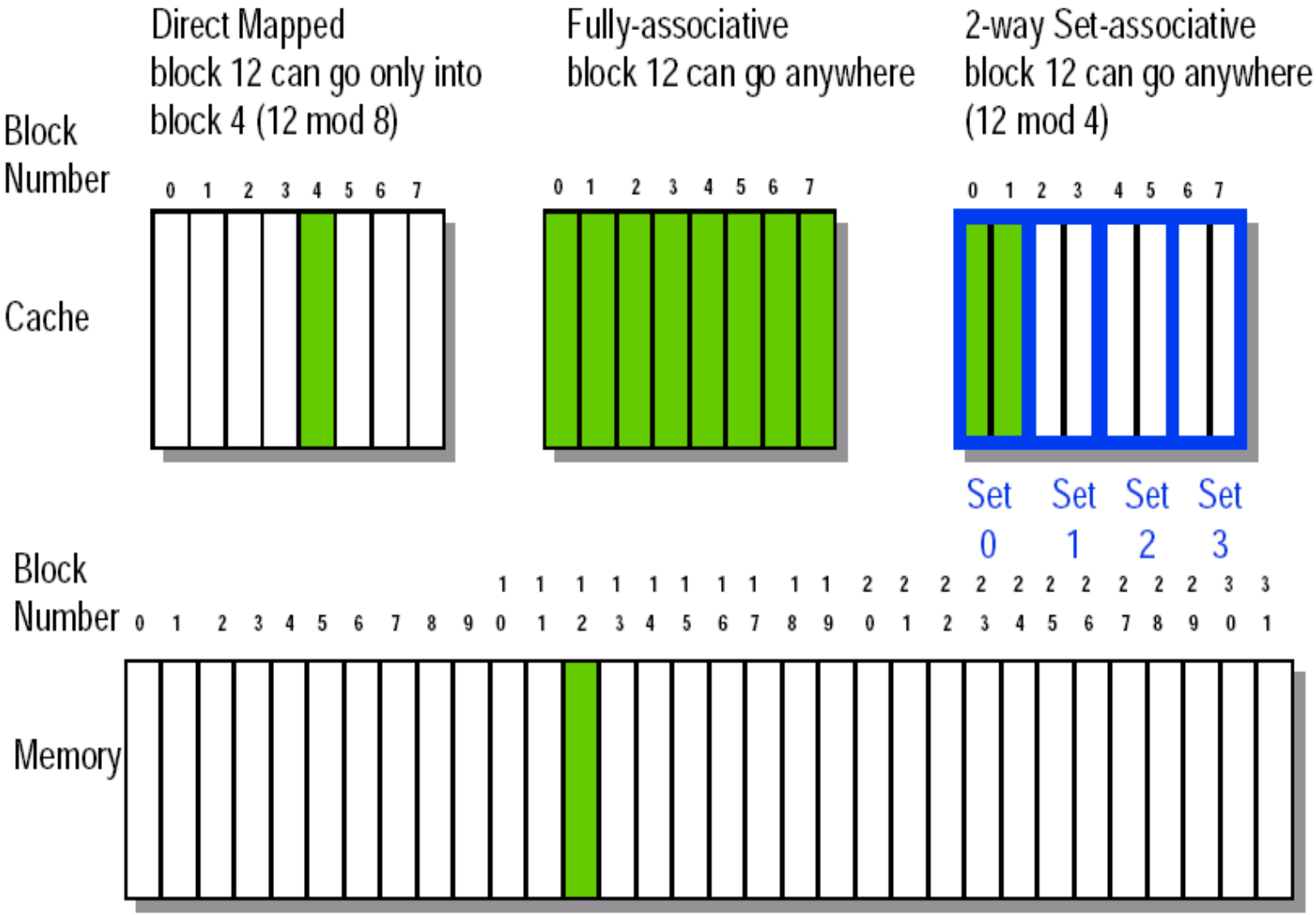
    Usually address MOD Number of blocks in cache
- **Fully associative**

  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 sets 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

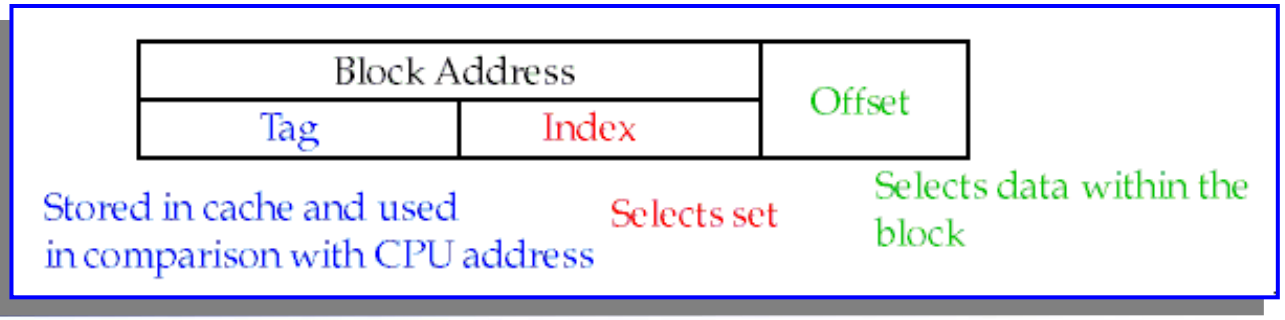
Valid bit

- 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

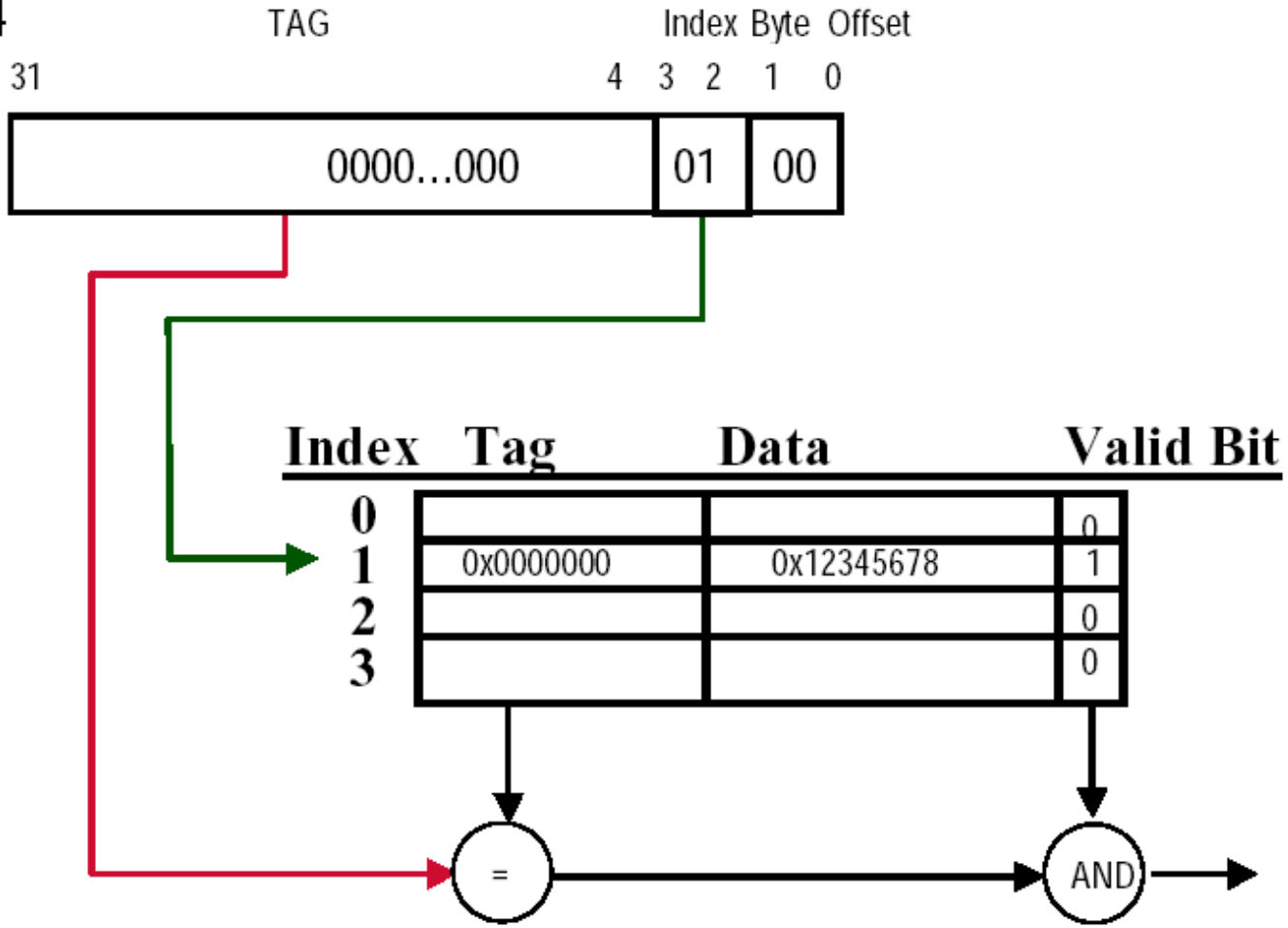
- The Index field selects
  - The set, in case of a set-associative cache
  - The block, in case of a direct-mapped cache
  - Has as many bits as log2(#sets) for set-associative caches, or log2(#blocks) for direct-mapped caches
- The Byte Offset field selects
  - The byte within the block
  - Has as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

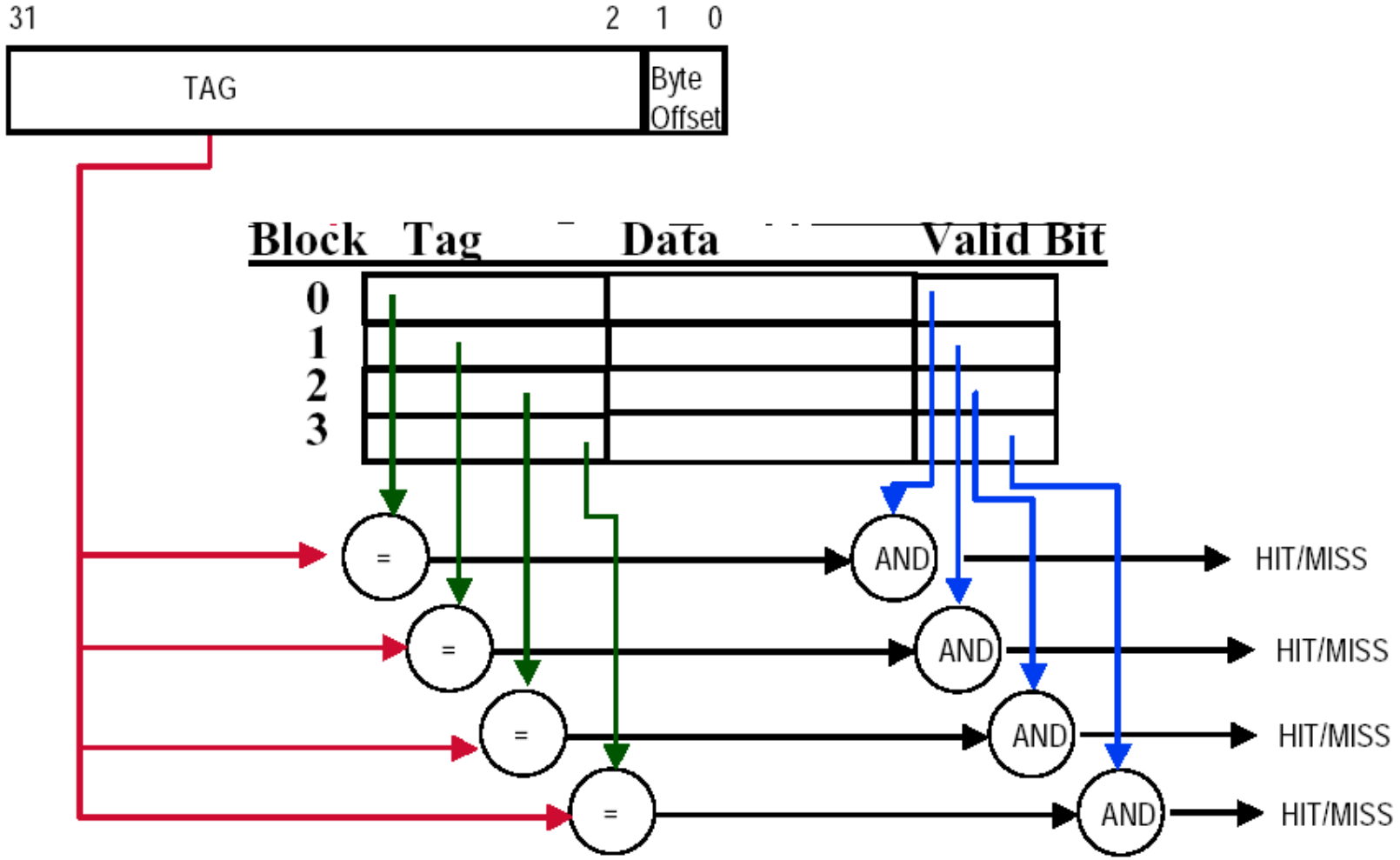


MEMORY

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

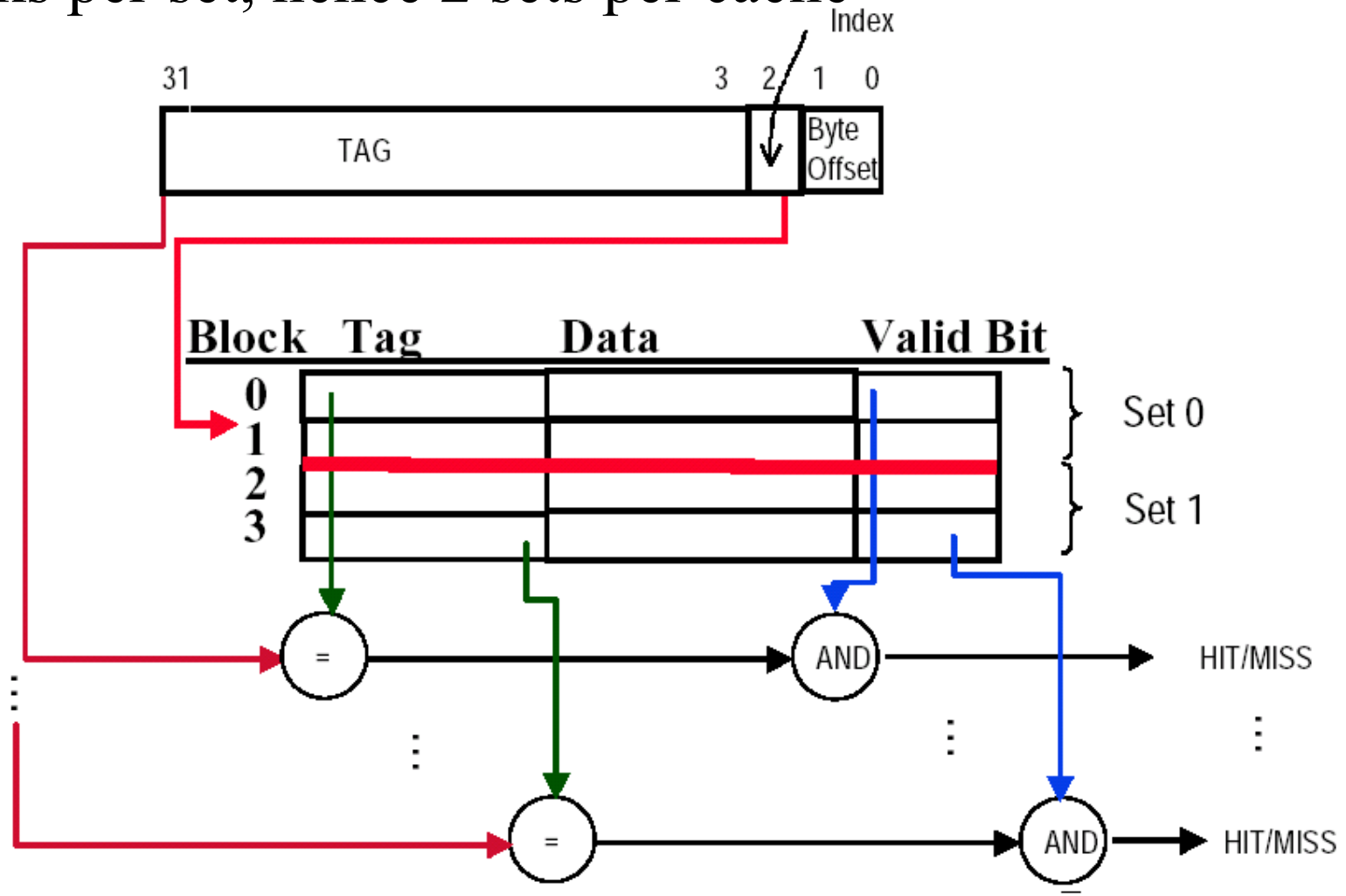
Fully-Associative Cache example (1-word Blocks)

- Assume cache has 4 blocks



2-Way Set-Associative Cache

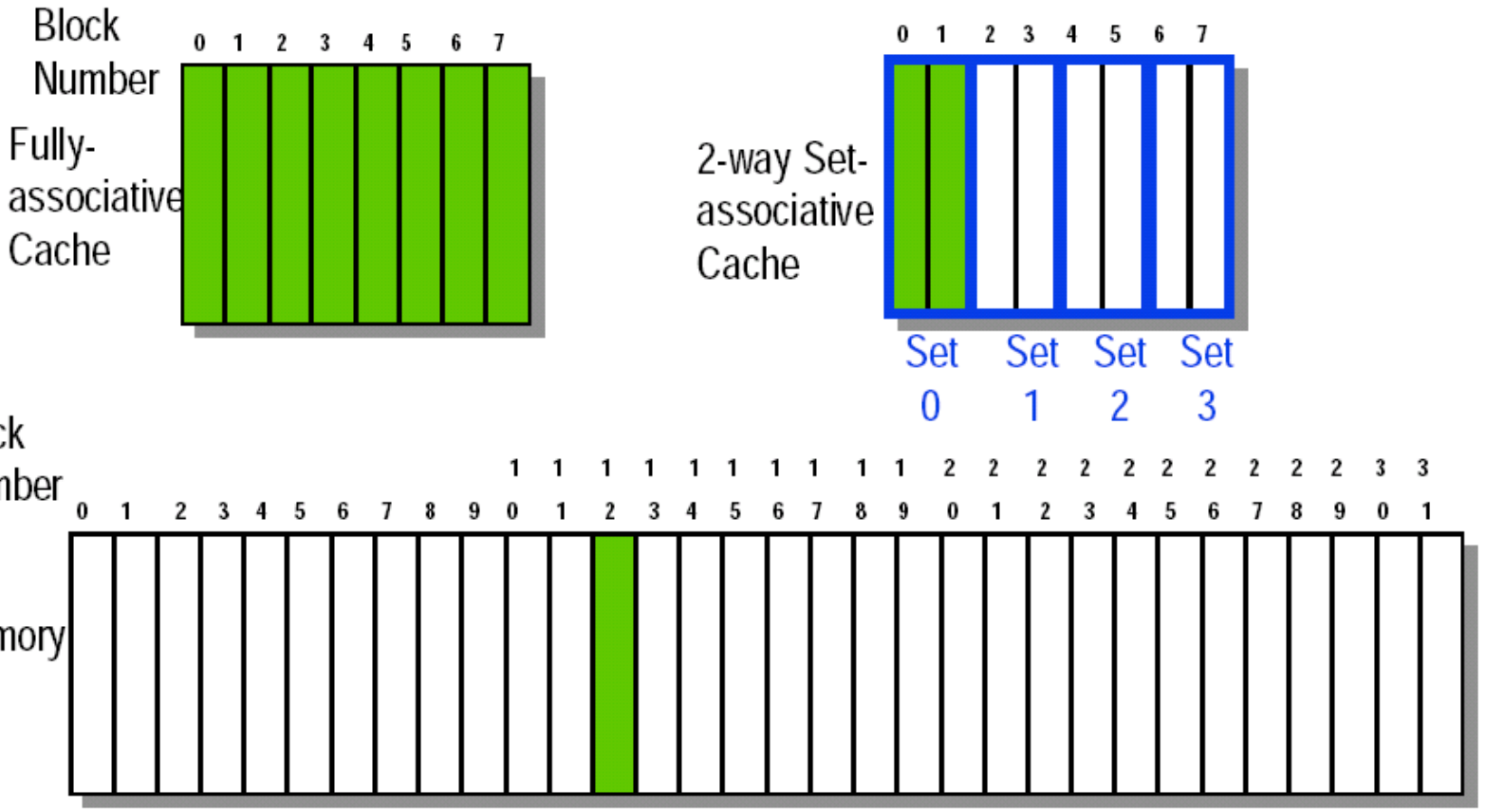
- Assume cache has 4 blocks and each block is 1 word
- 2 blocks per set, hence 2 sets per cache



Wrong ?

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*

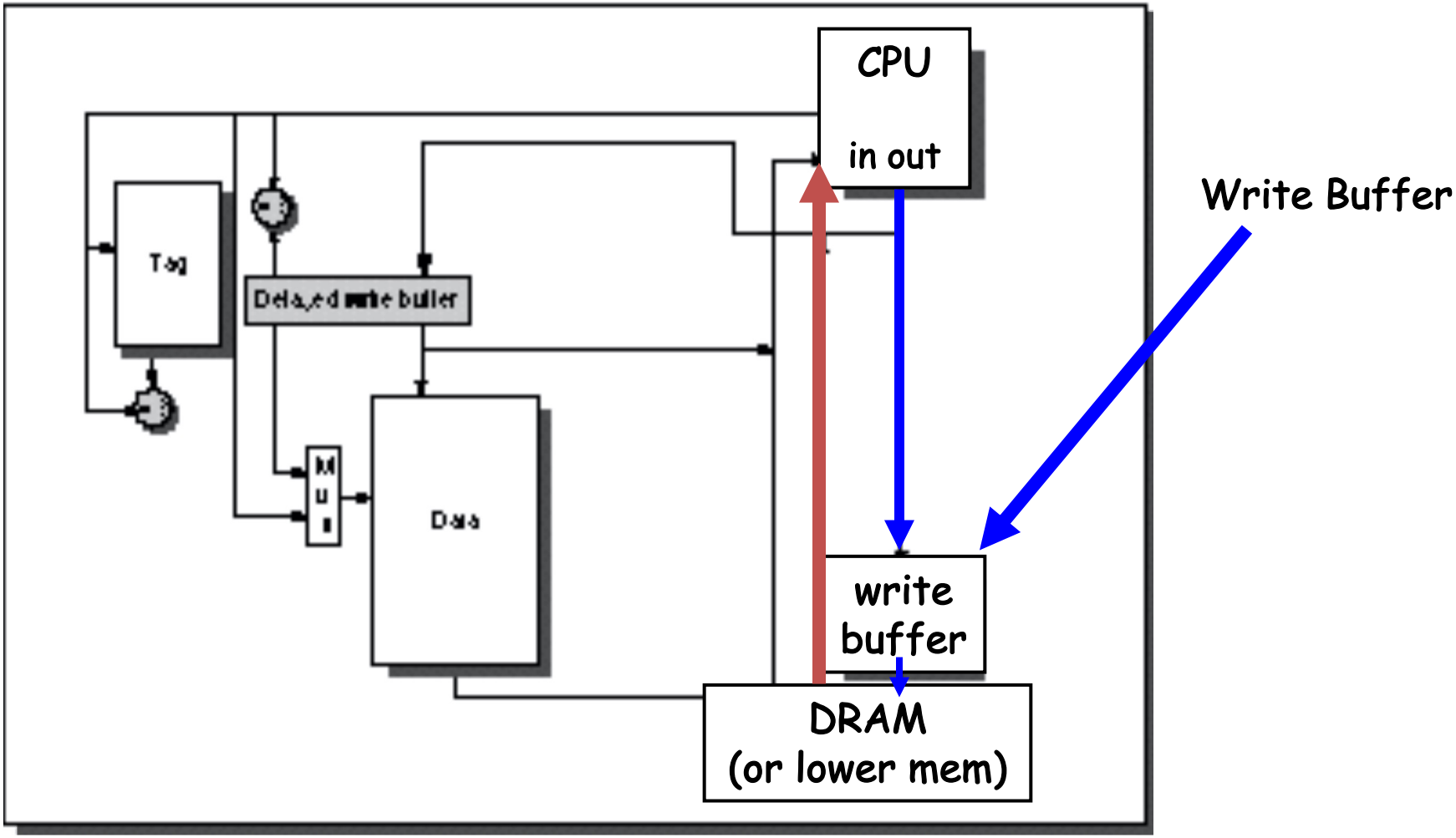
Q4: Write Strategy

- 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 fill if the burst is larger than the buffer.

Write buffers



Write misses

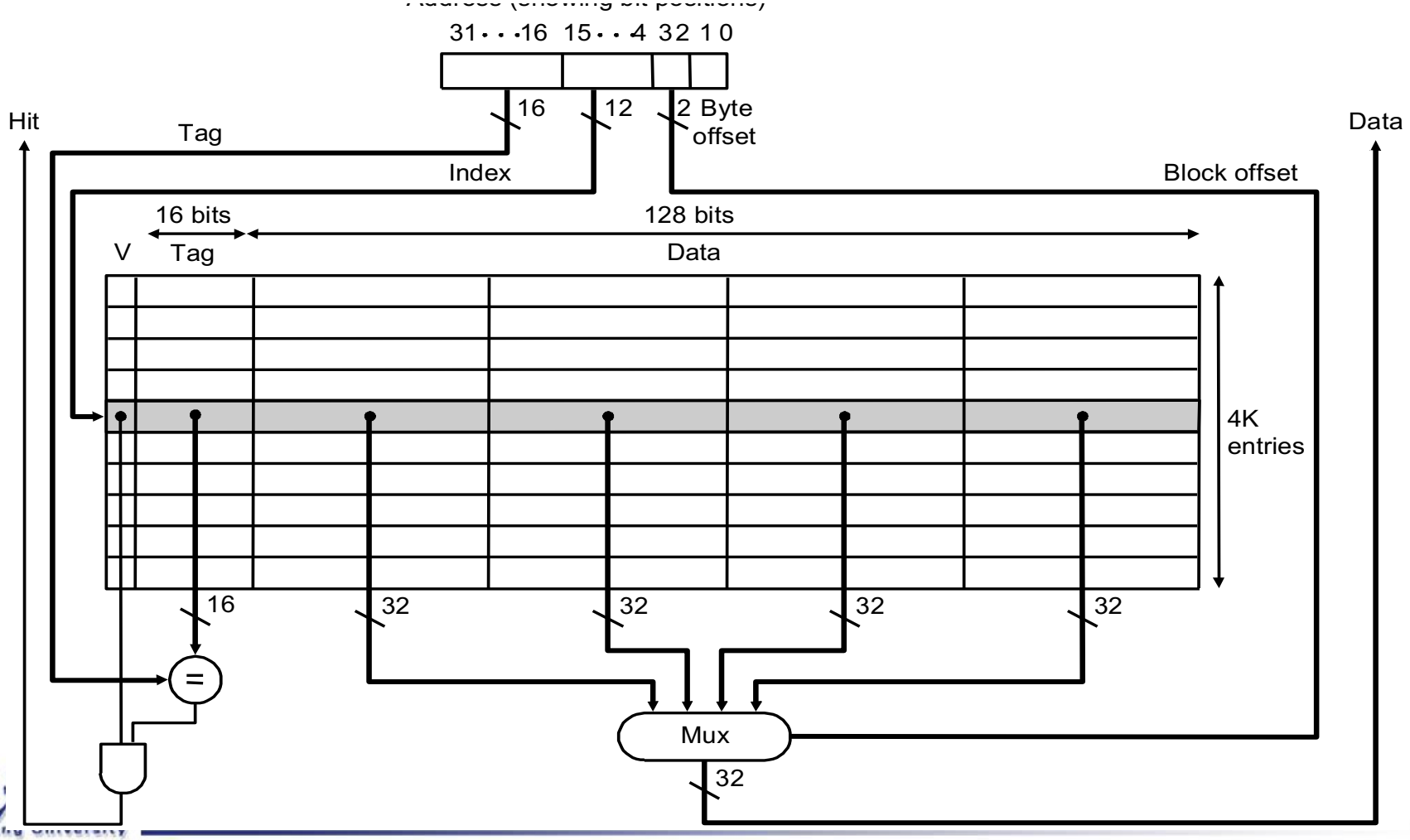
Write misses

- ***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 .***

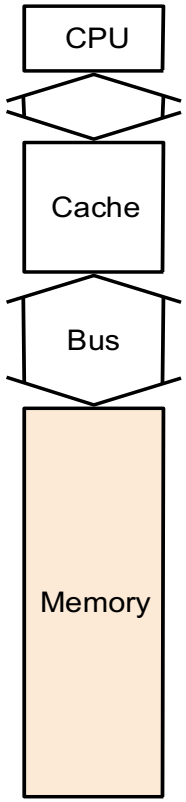
Larger blocks exploit spatial locality

- Taking advantage of spatial locality to lower miss rates with many word in the block:

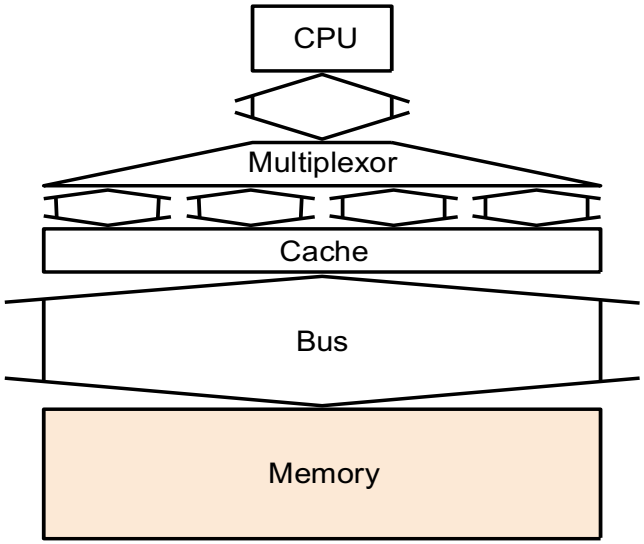


Designing the Memory system to Support Cache

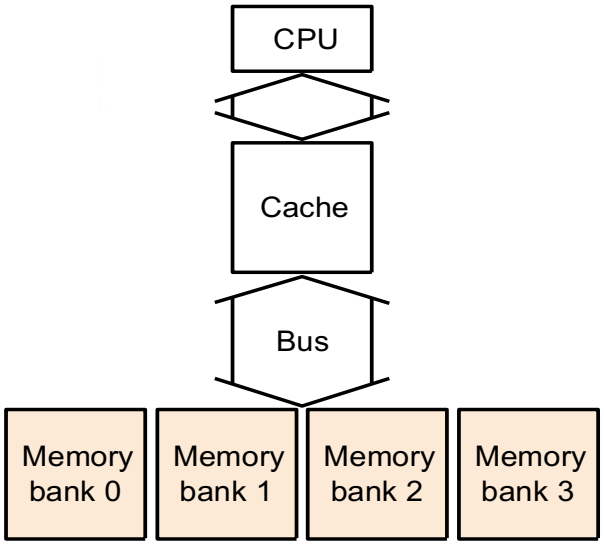
- Make reading multiple words easier by using banks of memory



a. One-word-wide memory organization



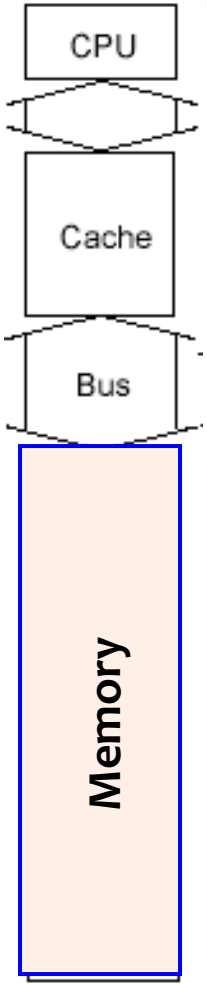
b. Wide memory organization



c. Interleaved memory organization

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

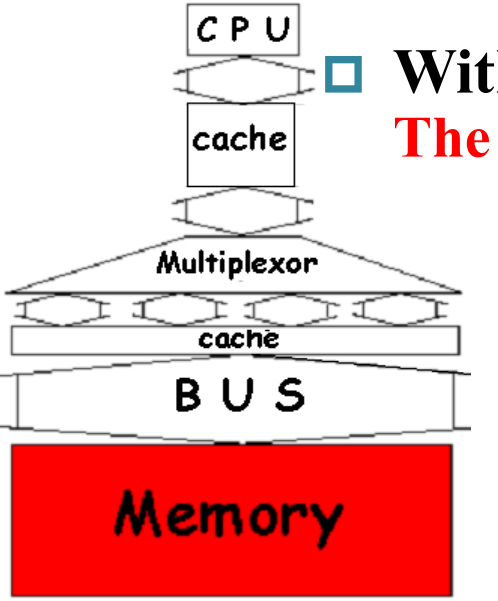
- With a main memory width of 2 words(64bits)

  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

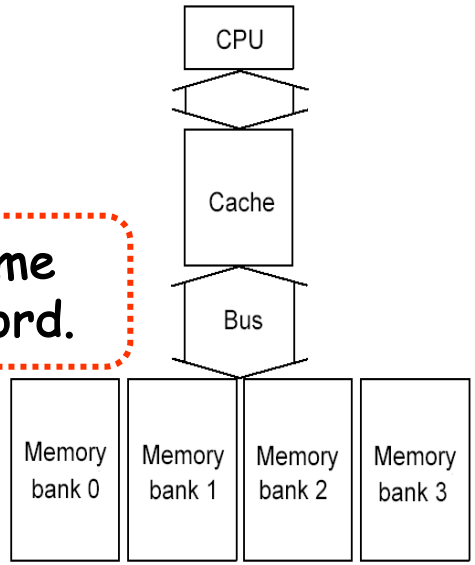
- With 4 banks Interleaved Memory

The miss penalty: 4words/Block

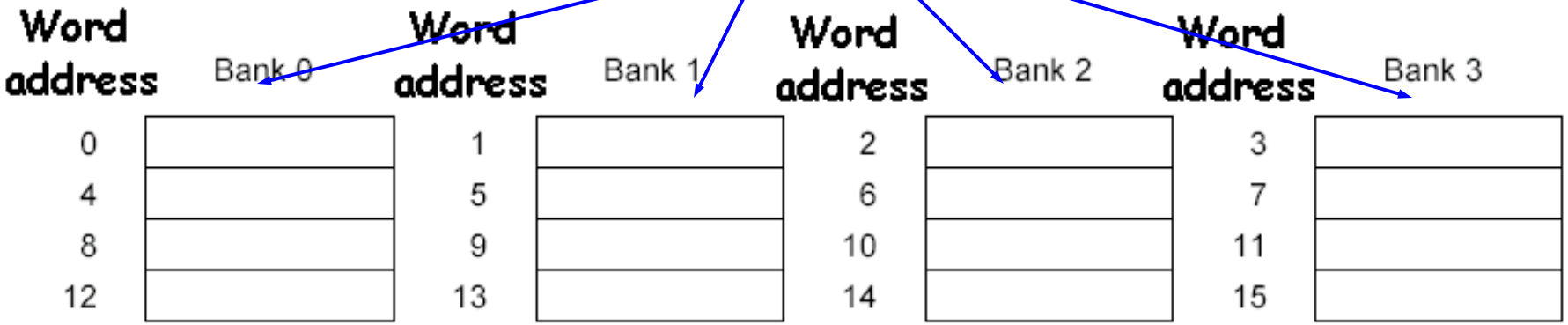
$$1+15+(4\times 1)=20$$

Almost equal to time to transfer one word.

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



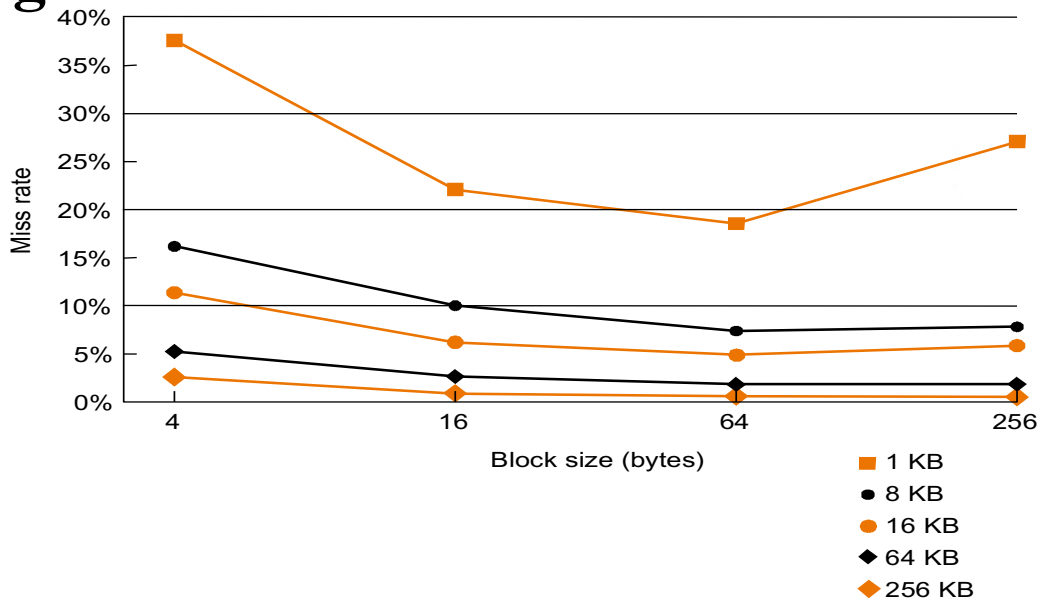
Parallel access



Optimizes sequential address access patterns

Performance in different block size

- Increasing the block size tends to decrease miss rate:



- Use split caches because there is more spatial locality in code:

Program	Block size in words	Instruction miss rate	Data miss rate	Effective combined miss rate
gcc	1	6.1%	2.1%	5.4%
	4	2.0%	1.7%	1.9%
spice	1	1.2%	1.3%	1.2%
	4	0.3%	0.6%	0.4%

5.4 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 Assess 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 .*
- *If the cache block size is one word, the write miss penalty is 0.*

Combine the reads and writes

- In most write-back 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:

  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%

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

- **Answer:**

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

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

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

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

  = (2 + 3.44)I = 5.44I

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 to slow 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

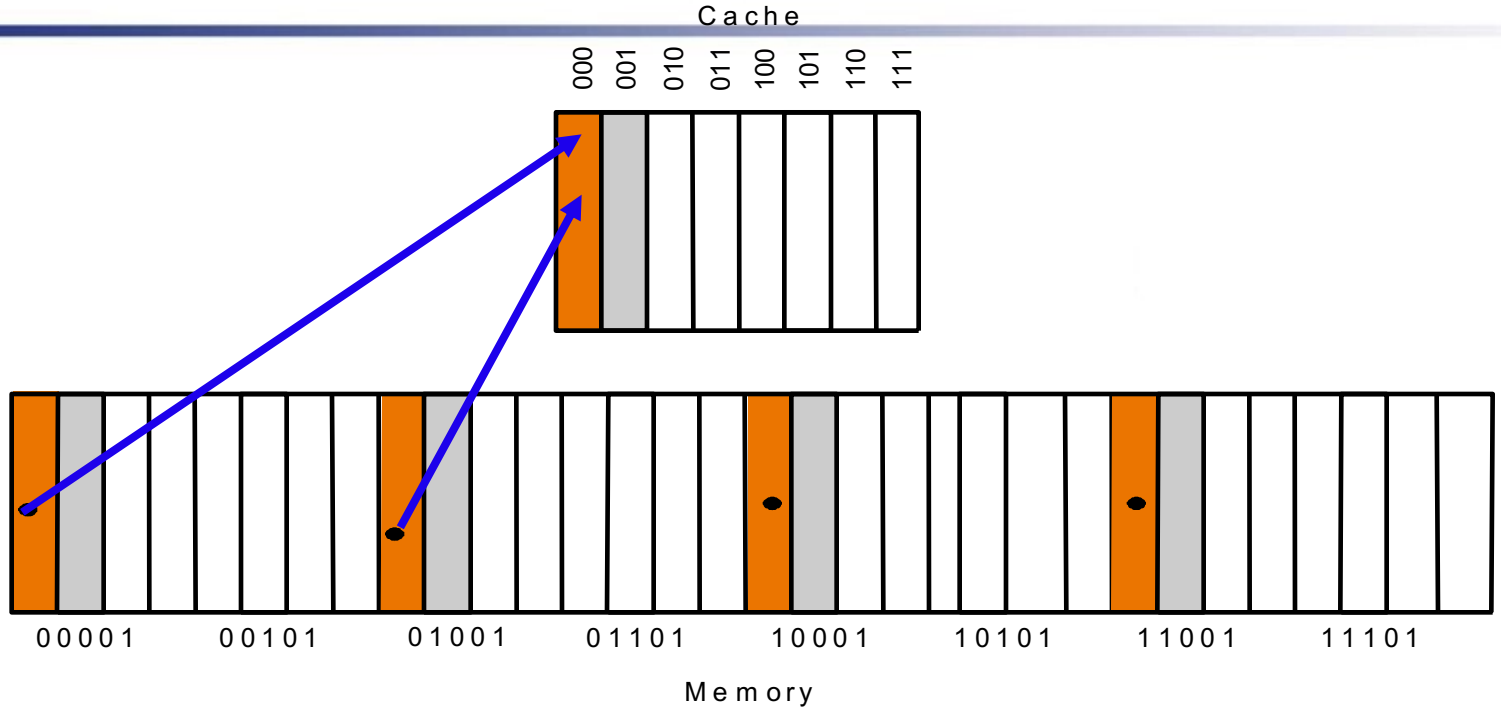
(3) Miss rate versus set-associative

(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

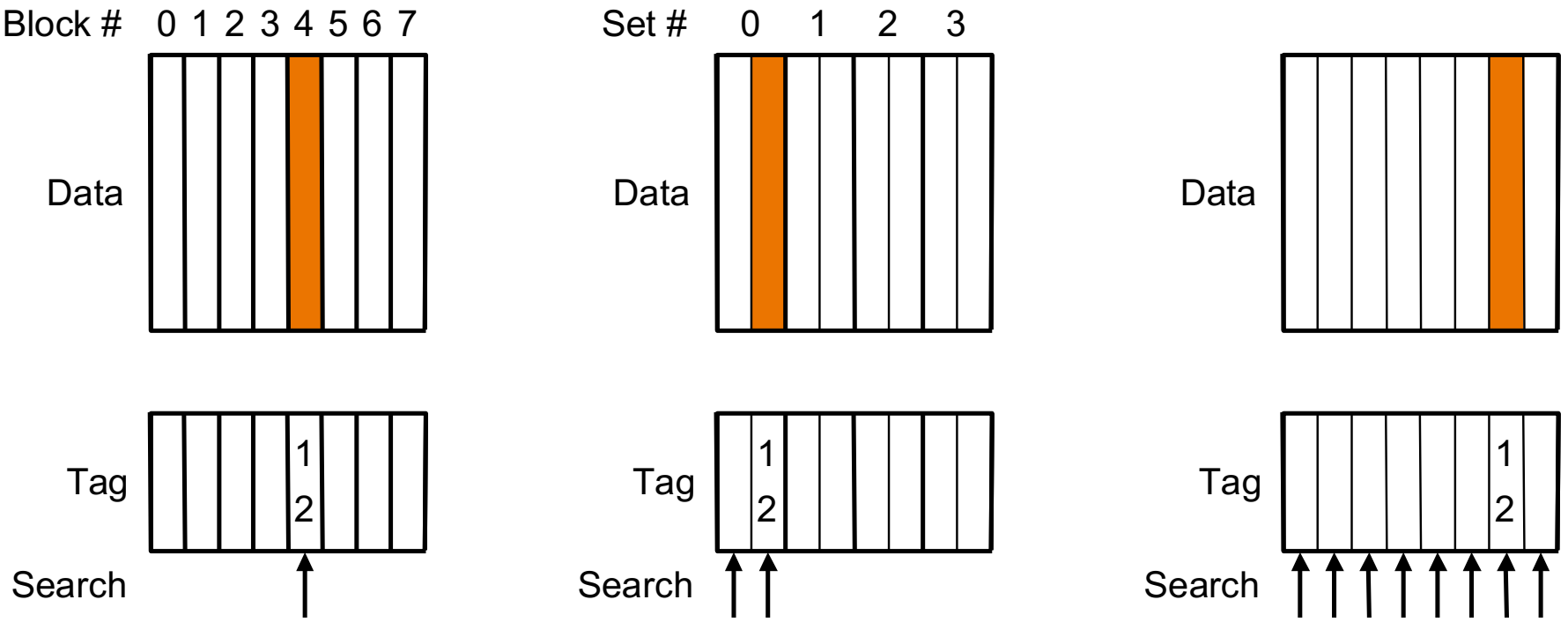
- **A set-associative cache**
  - is divided into some sets. A set contains several blocks.
- **A memory block is mapped to a set in the cache**
  - Through a mapping algorithm.
  - The memory block can be placed in any block in the corresponding set.
- **The mapping algorithm is:** (set with direct-mapped)

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

(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

Miss rate versus set-associativity—8Blocks

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]	

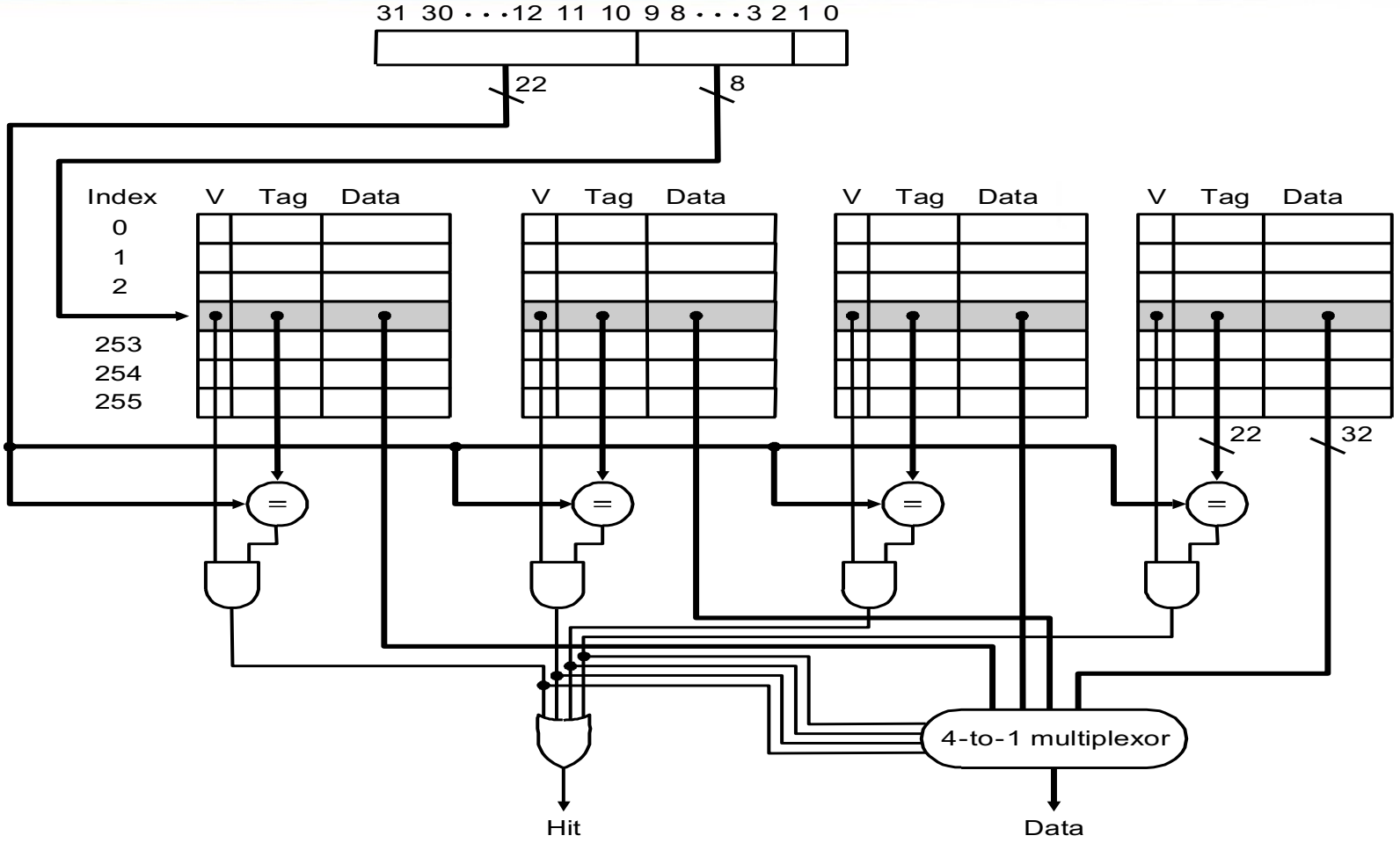
How much of a reduction in the miss rate is achieved by associativity?

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 Intrinsuty 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 size is 4K Block

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:**

$$\frac{100\text{ns}}{0.2} = 500 \text{ clock cycles}$$

- **The CPI with one level of caching**

Total CPI = 1.0 + 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

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

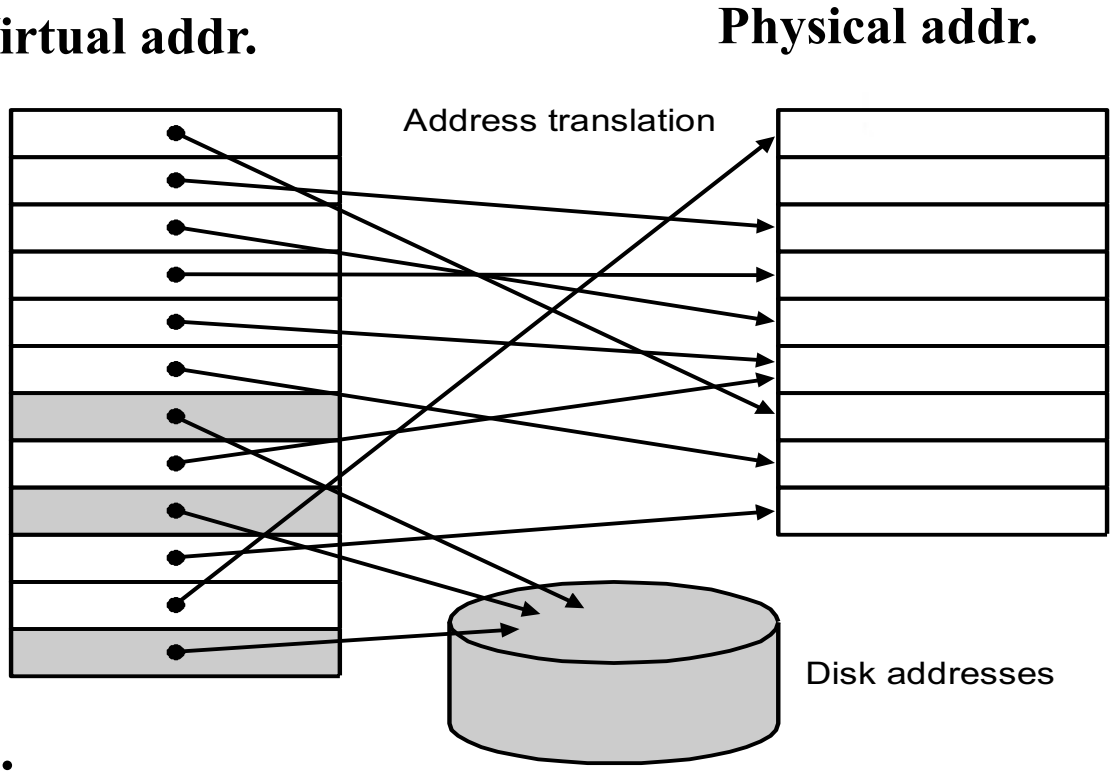
- Using multilevel caches:
  - try and optimize the hit time on the 1st level cache
  - try and optimize the miss rate on the 2nd level cache

7.4 Virtual Memory

- **Main Memory act as a "Cache" for the secondary storage.**
- **Motivation:**
  - Efficient and safe sharing of memory among multiple programs.
  - Remove the programming burdens of a small, limited amount of main memory.
- **Translation of a program's address space to physical address**

7.4 Virtual Memory

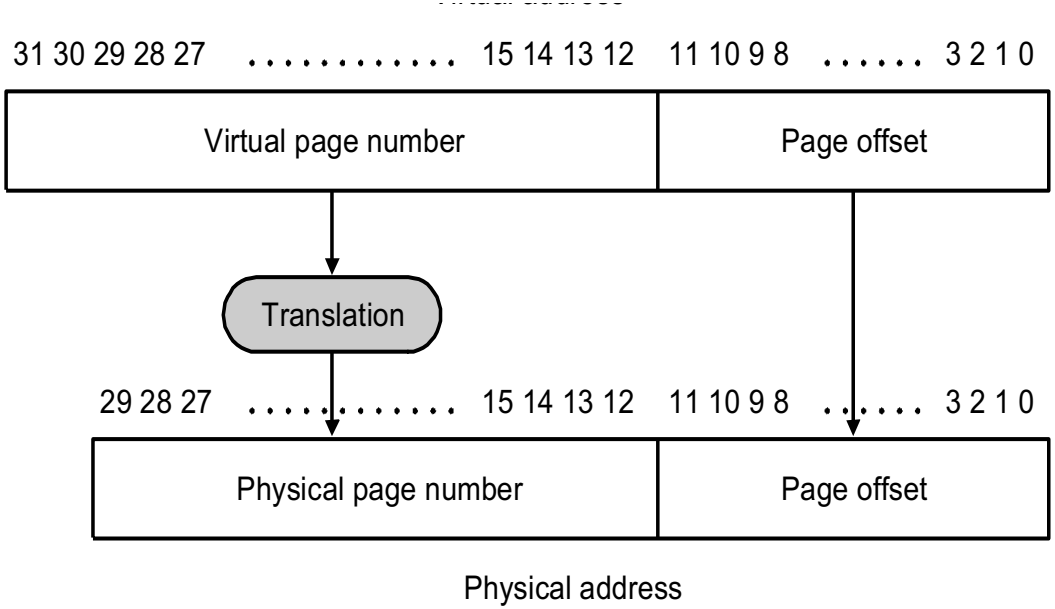
- Main memory can act as a cache for the secondary storage (disk)



- Advantages:
  - illusion of having more physical memory
  - program relocation
  - protection

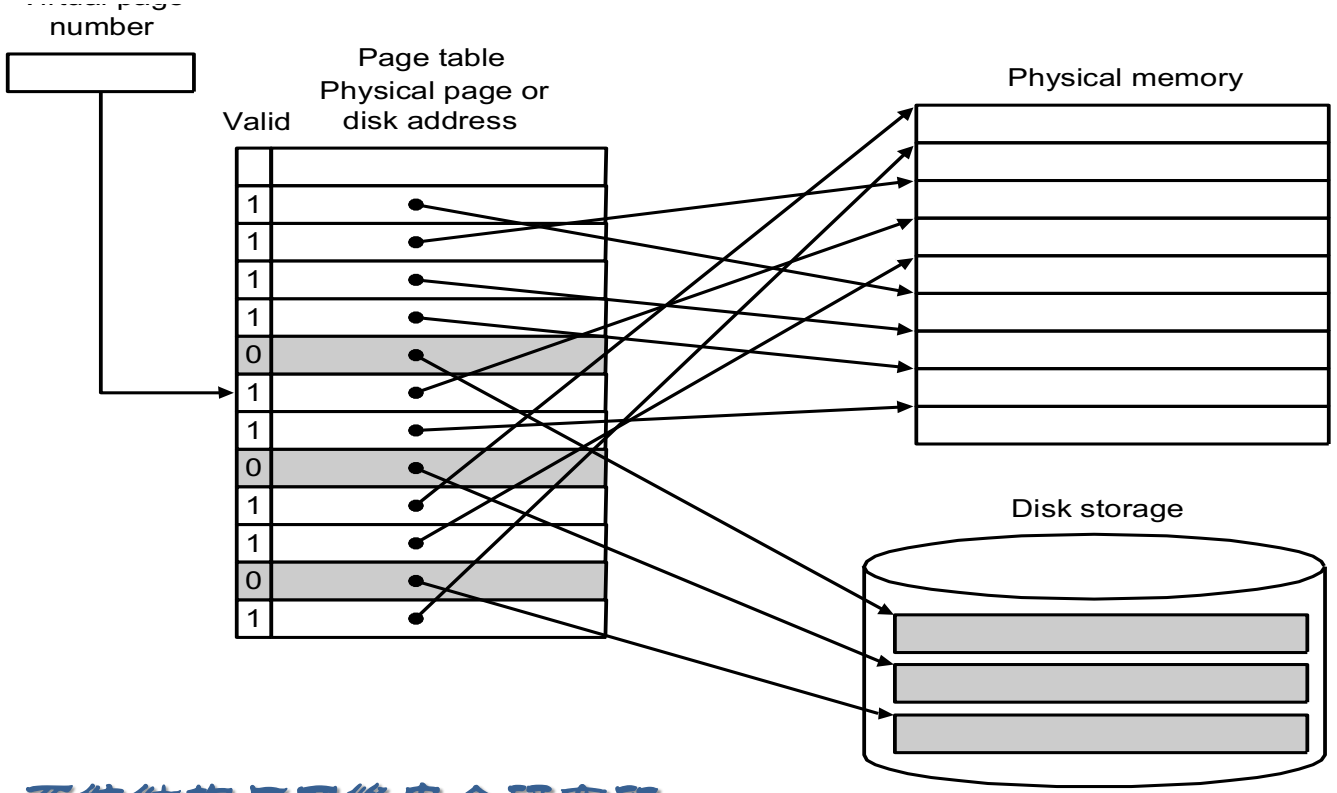
Pages: virtual memory blocks

- **Larger number of virtual pages than physical pages (Really now?)**
- **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

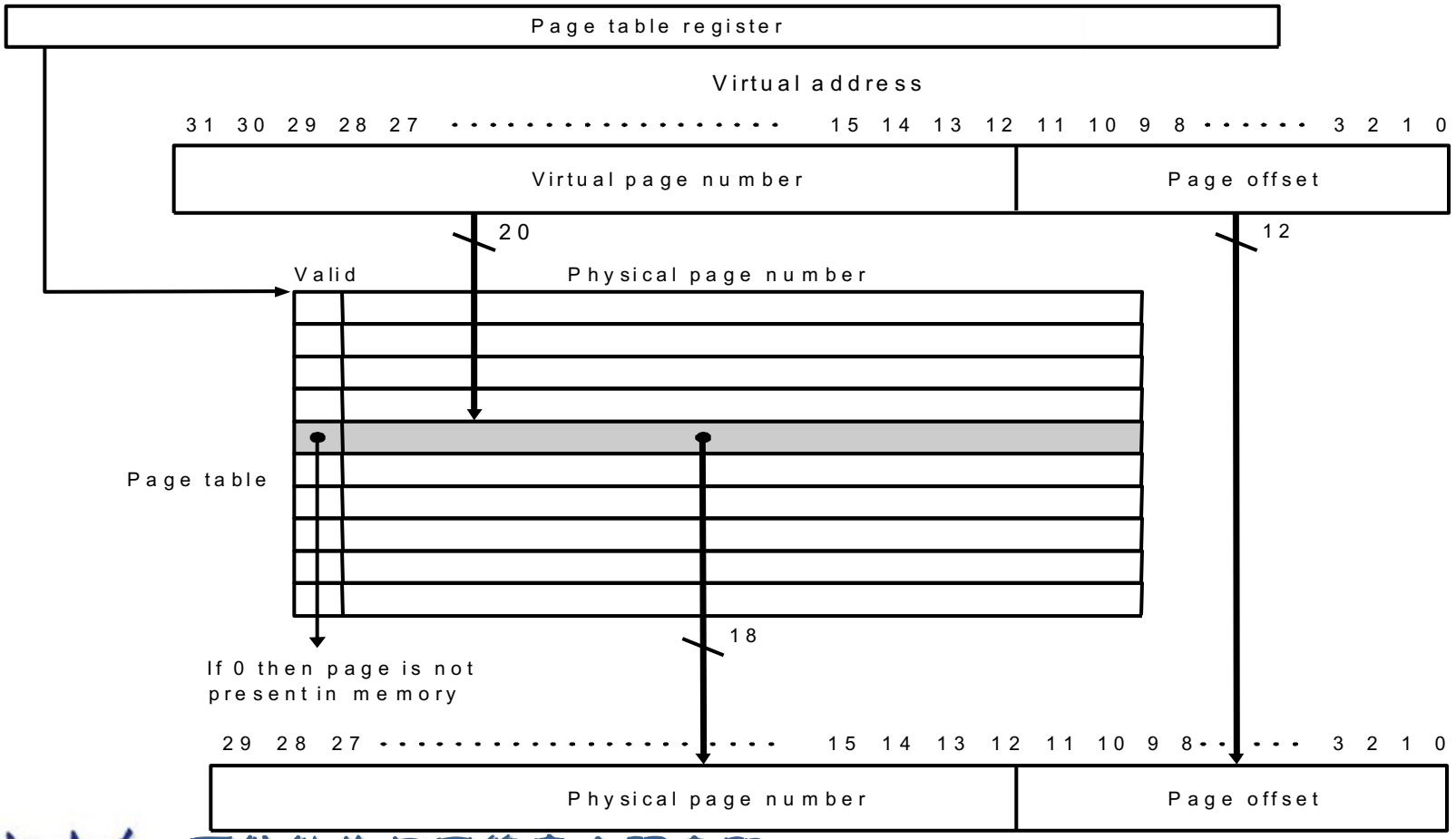
1.Page Table : Virtual to physical address
2. Stored into the memory, indexed by the virtual page number
3. Each Entry in the table contains the physical page number for that virtual pages if the page is current in memory
4. Page table, Program counter and the page table register, specifies the state of the program. Each process has one page table. (Process switch?)



Placing a page and finding it again --Page Tables

Each program has its own page table

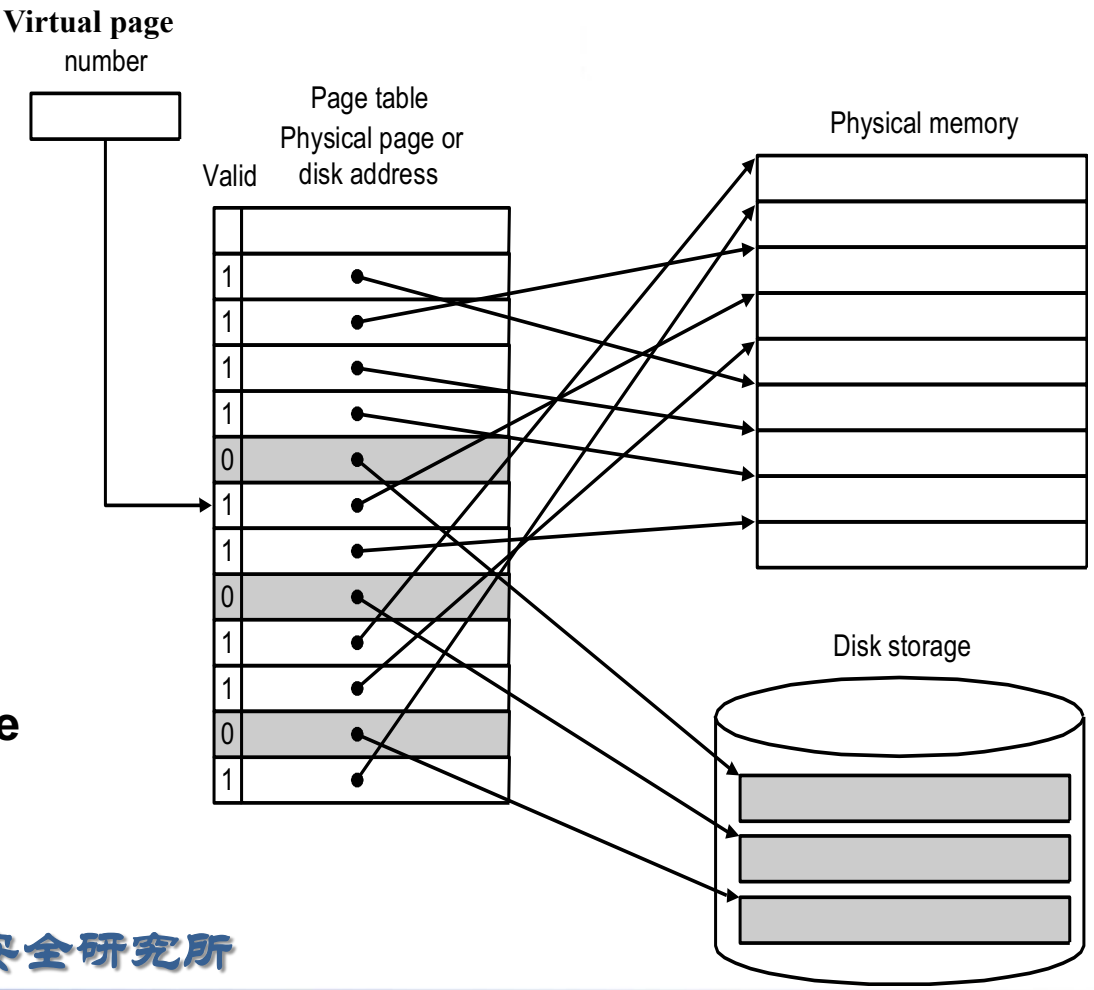
Virtual memory systems use fully associative mapping method



Page faults

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

- 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

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

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

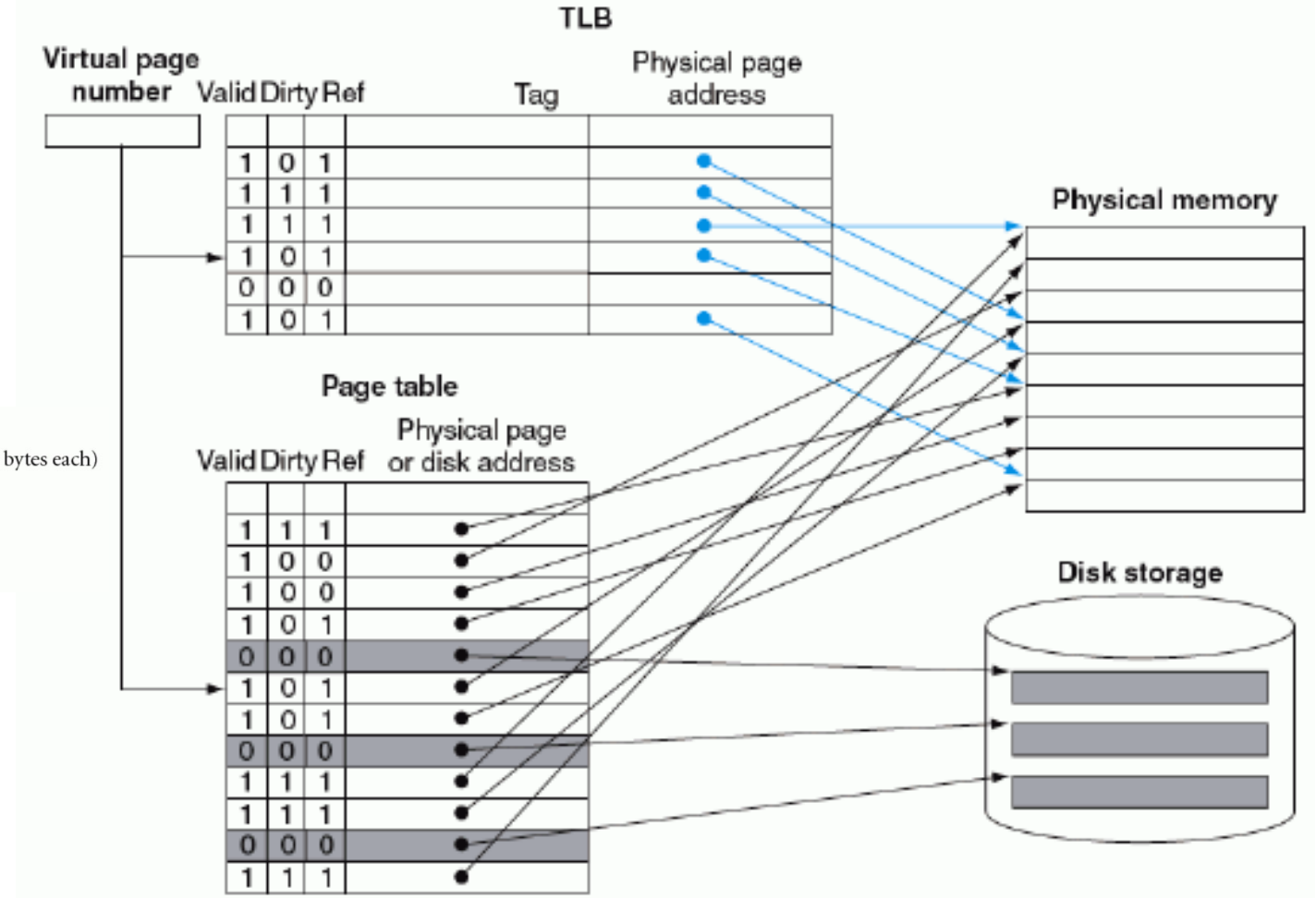
- **Five techniques is used to reduce page table size**

What about writes?

- Because disk accesses are too slow, virtual memory systems can not use write-through strategy.
- Instead, they must use write-back strategy. To do so, the machines need add a dirty bit to the entry of page table.
- The dirty bit is set when a page is first written. If the dirty bit of a page is set, the page must be written back to disk before being replaced.

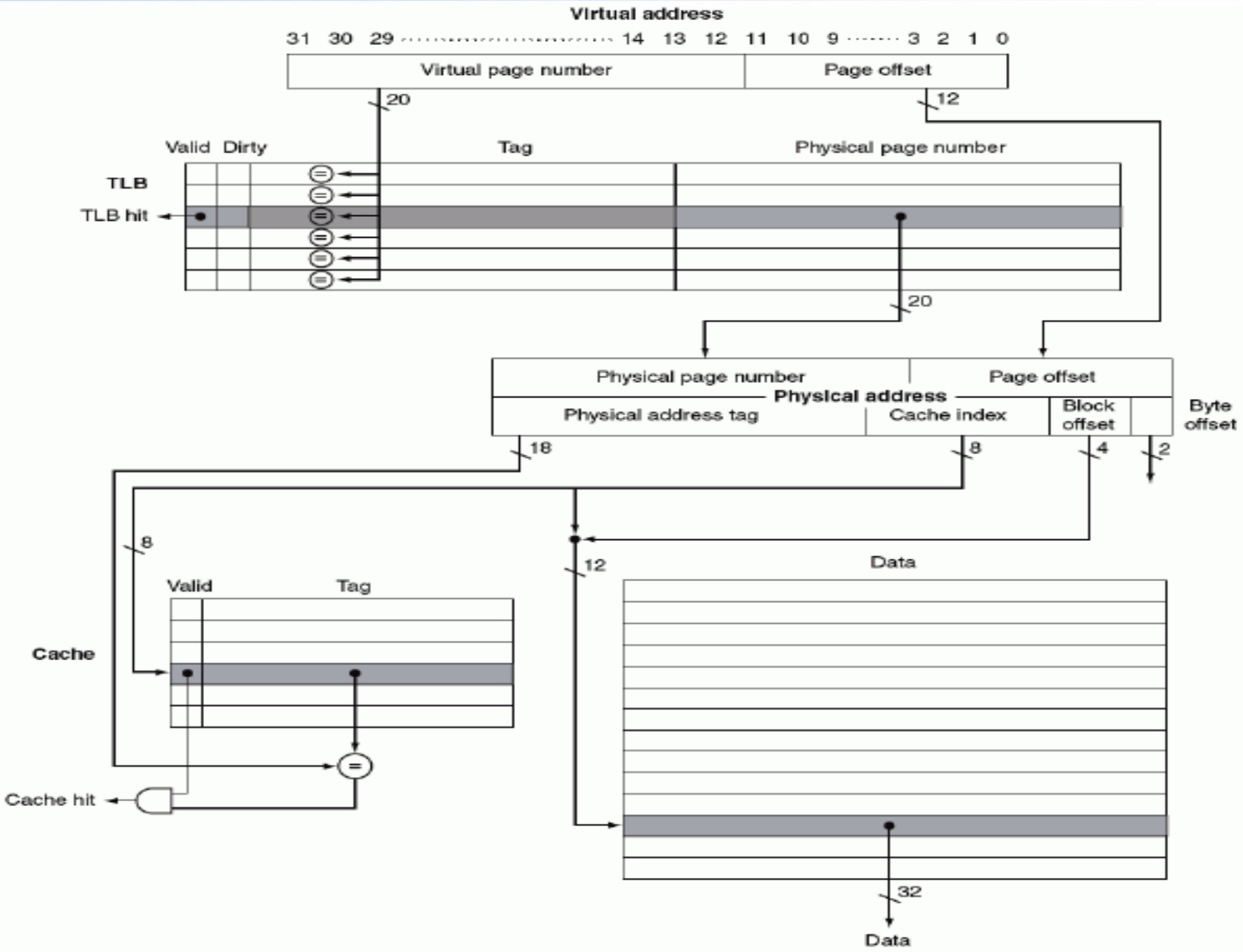
Making Address Translation Fast--TLB

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

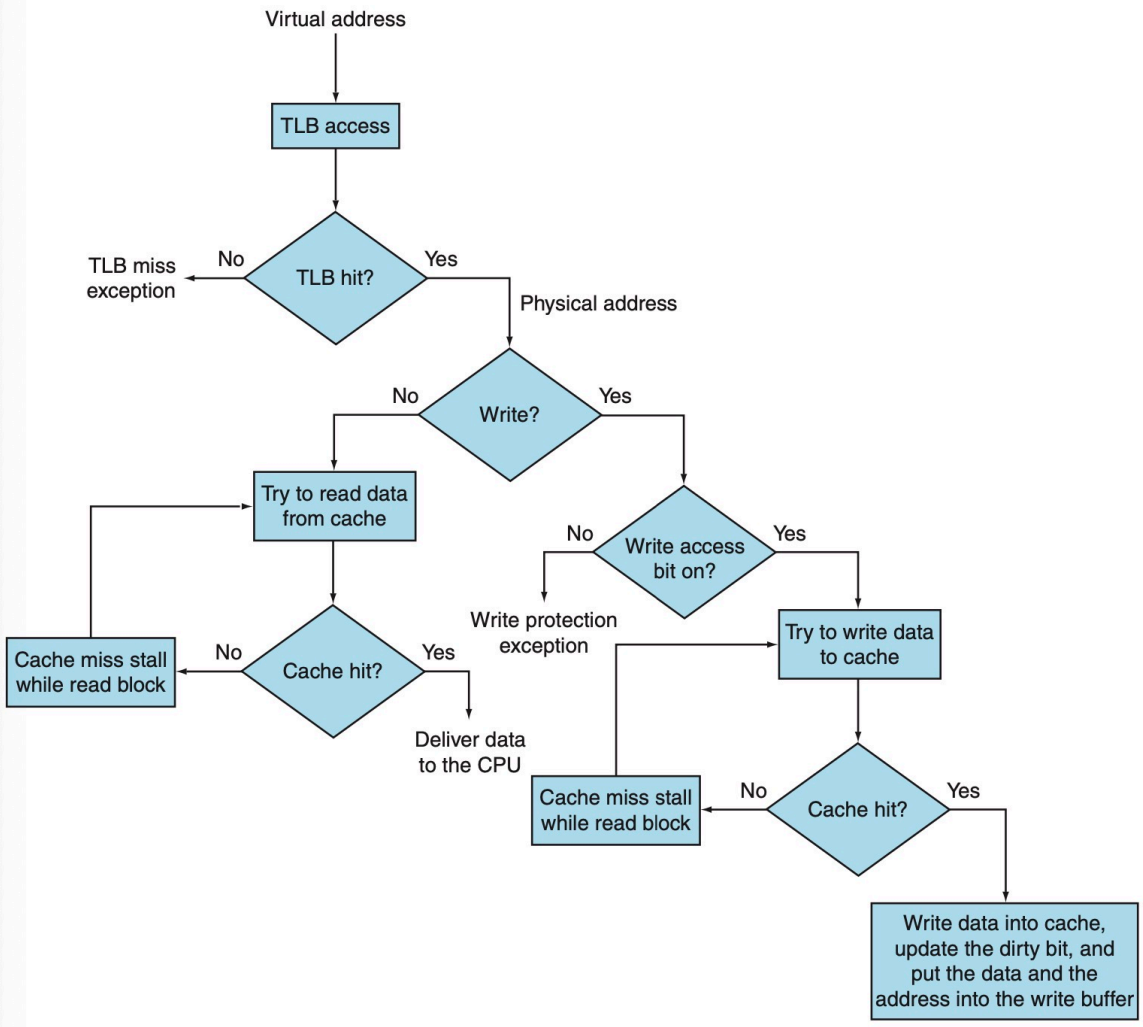


- TLB size: 16–512 entries
- Block size: 1–2 page table entries (typically 4–8 bytes each)
- Hit time: 0.5–1 clock cycle
- Miss penalty: 10–100 clock cycles
- Miss rate: 0.01%–1%

FastMATH Memory Hierarchy



TLBs and caches



Possible combinations of Event

- Three different types of misses: TLB miss, page Fault, cache miss

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

Protection in the virtual memory System

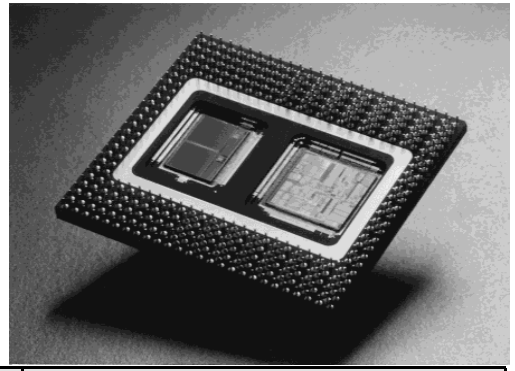
1. Support at least two modes that indicate whether the running process is a user process or an operating system process, variously called a **supervisor** process, a **kernel** process, or an *executive* process.
2. Provide a portion of the processor state that a user process can read but not write. This state includes the user/supervisor mode bit, which dictates whether the processor is in user or supervisor mode, the page table pointer, and the TLB. To write these elements, the operating system uses special instructions that are only available in supervisor mode.
3. Provide mechanisms whereby the processor can go from user mode to supervisor mode and vice versa. The first direction is typically accomplished by a **system call** exception, implemented as a special instruction (`ecall` in the RISC-V instruction set) that transfers control to a dedicated location in supervisor code space. As with any other exception, the program counter from the point of the system call is saved in the *supervisor exception program counter* (SEPC), and the processor is placed in supervisor mode. To return to user mode from the exception, use the *supervisor exception return* (`sret`) instruction, which resets to user mode and jumps to the address in SEPC.

By using these mechanisms and storing the page tables in the operating system's address space, the operating system can change the page tables while preventing a user process from changing them, ensuring that a user process can access only the storage provided to it by the operating system.

Modern Systems

- **Very complicated memory systems:**

Characteristic	Intel Pentium Pro	PowerPC 604
Virtual address	32 bits	52 bits
Physical address	32 bits	32 bits
Page size	4 KB, 4 MB	4 KB, selectable, and 256 MB
TLB organization	A TLB for instructions and a TLB for data Both four-way set associative Pseudo-LRU replacement Instruction TLB: 32 entries Data TLB: 64 entries TLB misses handled in hardware	A TLB for instructions and a TLB for data Both two-way set associative LRU replacement Instruction TLB: 128 entries Data TLB: 128 entries TLB misses handled in hardware



Characteristic	Intel Pentium Pro	PowerPC 604
Cache organization	Split instruction and data caches	Split intruction and data caches
Cache size	8 KB each for instructions/data	16 KB each for instructions/data
Cache associativity	Four-way set associative	Four-way set associative
Replacement	Approximated LRU replacement	LRU replacement
Block size	32 bytes	32 bytes
Write policy	Write-back	Write-back or write-through

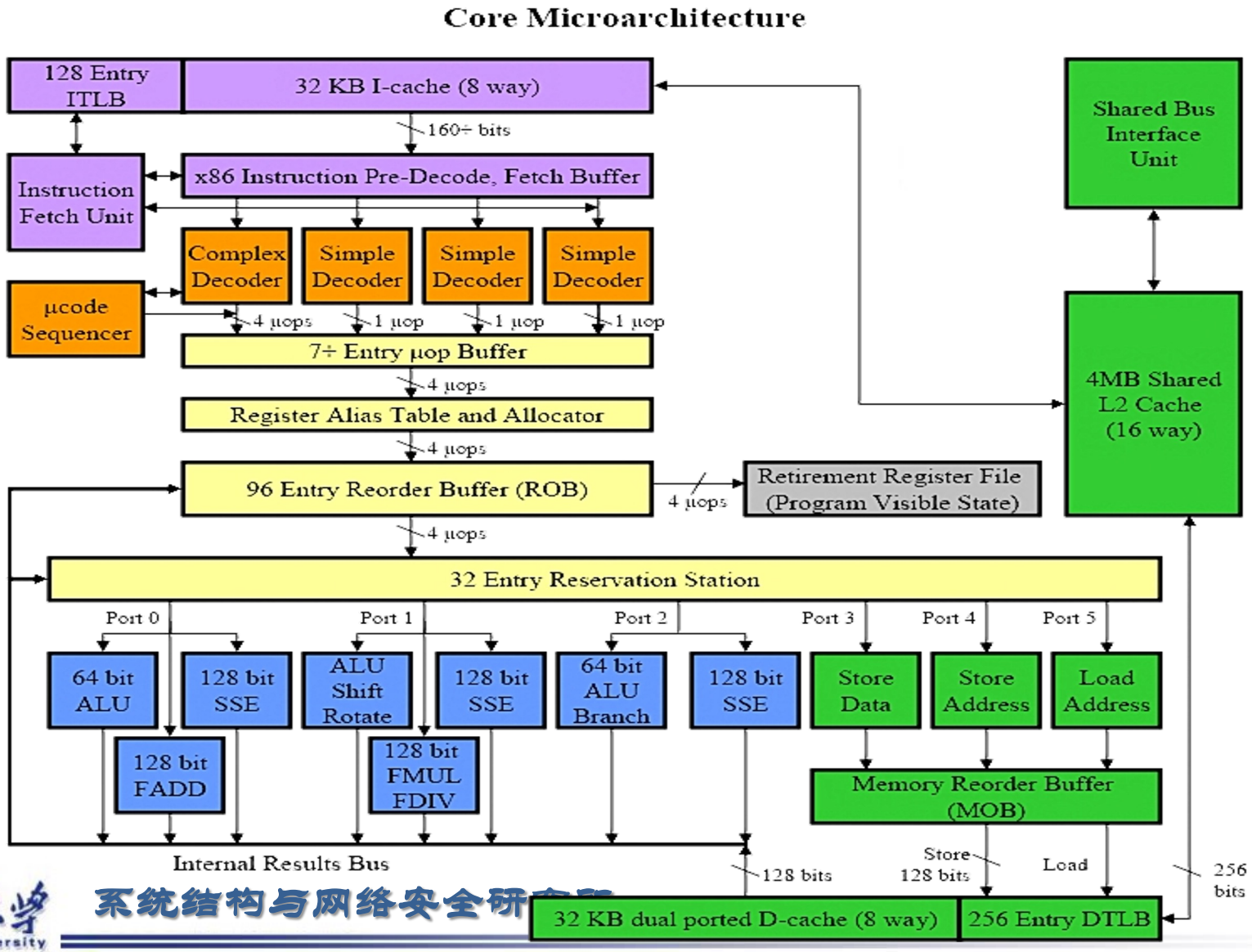
Some Issues

- **Processor speeds continue to increase very fast**
  — much faster than either DRAM or disk access times

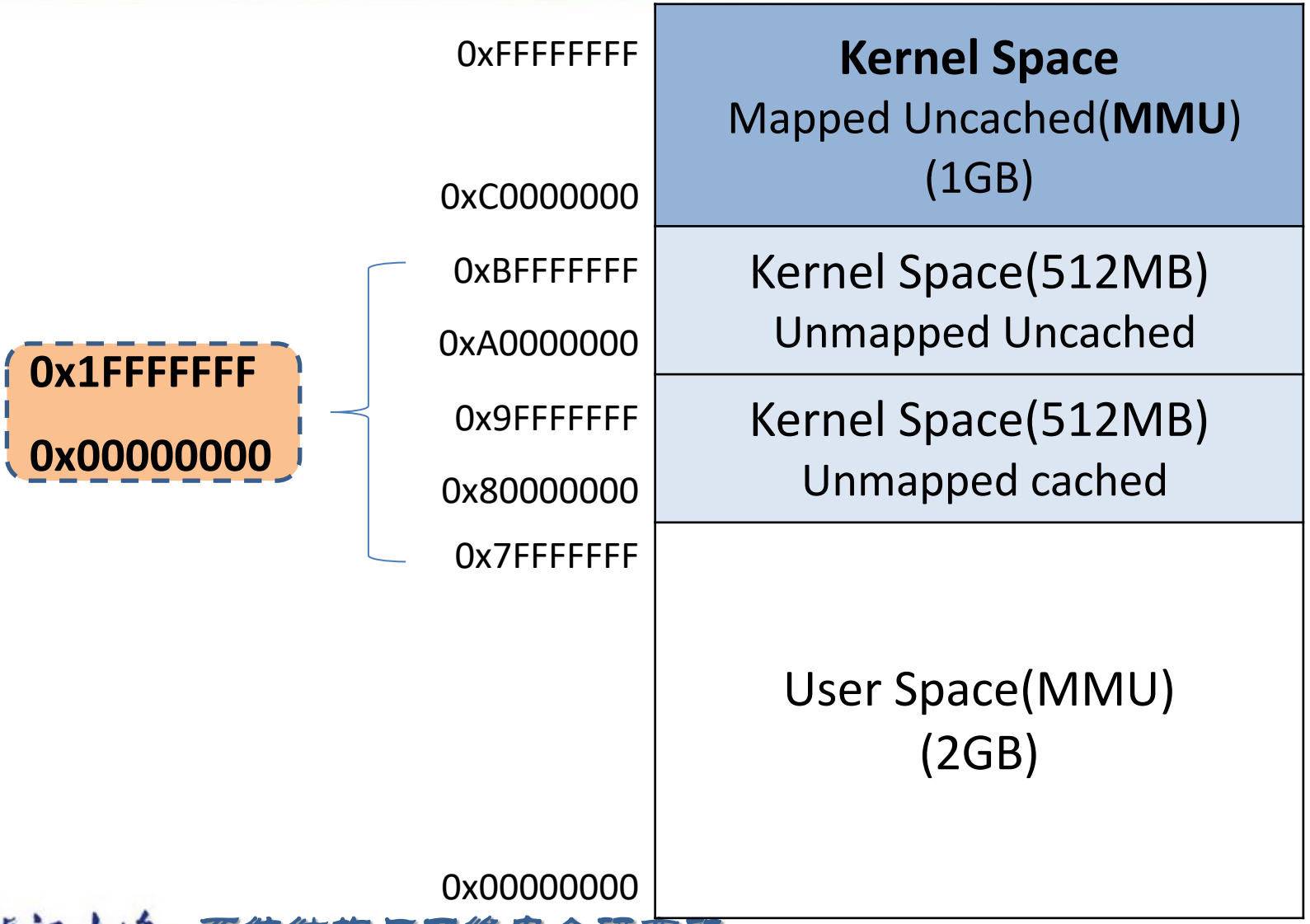
- **Design challenge: dealing with this growing disparity**

- **Trends:**
  - synchronous SDRAMs (provide a burst of data)
  - redesign DRAM chips to provide higher bandwidth or processing
  - restructure code to increase locality
  - use prefetching (make cache visible to ISA)

Intel Core微架构



MIPS MMU


0xFFFFFFFF
Kernel Space
Mapped Uncached(MMU)
(1GB)
0xC0000000
0xBFFFFFFF
Kernel Space(512MB)
Unmapped Uncached
0xA0000000
0x9FFFFFFF
Kernel Space(512MB)
Unmapped cached
0x80000000
0x7FFFFFFF
User Space(MMU)
(2GB)
0x00000000
0x1FFFFFFF
0x00000000

Homework

- 7.3， 7.4，7.5， 7.10，7.13， 7.32，7.39，7.41，7.45


END