Chapt 4-1: Memory hierarchy

- Memory hierarchy
- The basic of cache
- Organization of main memory

Chapter C & 5: Memory Hierarchy

- Memory Hierarchy ABC
- How to improve Cache performance
- Memory Organization
- Virtual Memory

4.1 Introduction

- Why do designers need to know about memory technology?
  - Processor performance is usually limited by memory bandwidth
  - As IC densities increase, lots of memory will fit on processor chip
- Application requirements:
  - Unlimited amounts of memory
  - Faster memory, higher bandwidth
  - Lower price per byte
  - If for embedded systems: lower power comsumption
- These requirements are contradictory.
  - The bigger, more difficult to make it fast
  - The faster, more expensive
  - The faster will consume much more power.

Memory Technologies

- Random Access Memories
  - **DRAM**: *Dynamic Random Access Memory*
    - High density, low power, cheap, slow
    - Dynamic: needs to be "refreshed" regularly
  - **SRAM**: *Static Random Access Memory*
    - Low density, high power, expensive, fast
    - Static: content will last "forever"(until lose power)
- What gets used where?
  - Main memory is **DRAM**: you need it big, so you need it cheap
  - CPU cache memory is **SRAM**: you need it fast, so it's more expensive, so it's smaller than you would usually want due to resource limitations
- Relative performance
  - Size: DRAM/SRAM: 4-8x bigger for DRAM
  - Cost/Cycle time: SRAM/DRAM: 8-16x faster, more $$$ for SRAM

Memory Hierarchy: a natural Solution

- How can we provide a memory with small access time, big capacity and lower price ?
- The first principle: make the common case fast !
  - What is the common case ?
- Recall: the principle of locality of reference !
  - Program access a relatively small portion of the address space at any instant of time.
  - Ok, we should make these accesses more quickly.
  - We can hold the recently accessed items in a fast memory.
- Yeah: Smaller memories will be faster !
  - We can use more expensive and smaller memories to hold the most recently used items.
  - The cost and power impact is lessoned for small size.

What is Memory Hierarchy ?

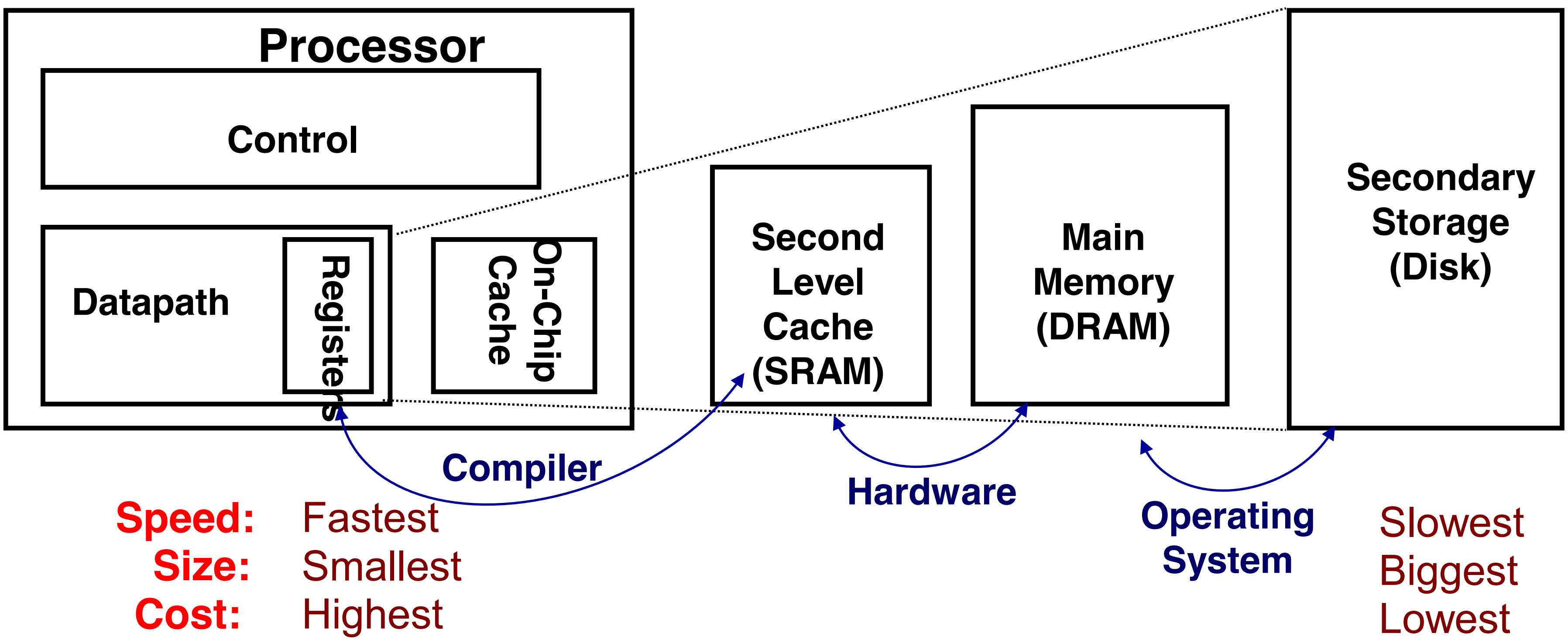
- Memory hierarchy is organized into several levels:
  - Each smaller, faster, and more expensive per byte than the next lower level.
  - **Temporal Locality** (Locality in Time):

    $\Rightarrow$ Keep most recently accessed data items closer to the processor
  - **Spatial Locality** (Locality in Space):

    $\Rightarrow$ Move blocks consists of contiguous words to the faster levels

Memory Hierarchy

- The goal: To provide a memory system with cost most almost as low as the cheapest level of memory and speed almost as fast as the fastest level.



Levels of the Memory Hierarchy

Capacity
Access Time
Cost

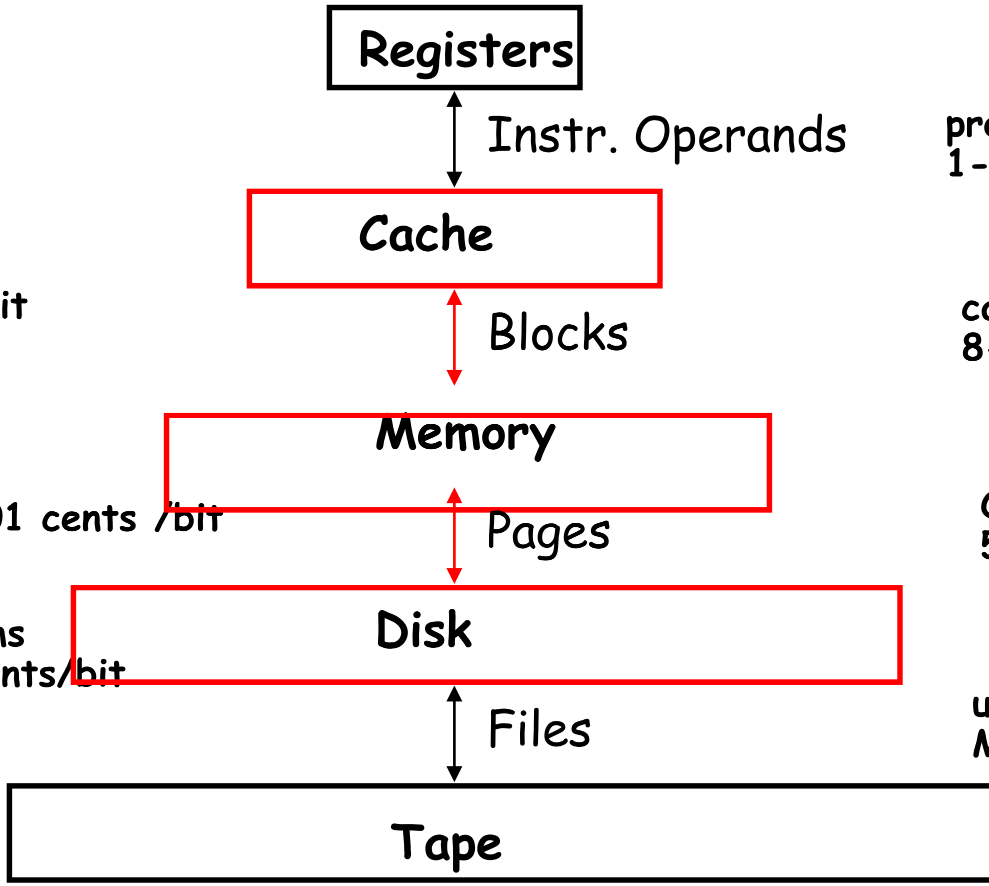
CPU Registers
500 Bytes
<250 ps

Cache
K Bytes
1-10ns
1-0.1 cents/bit

Main Memory
1G Bytes
100ns- 150ns
$.0001-.00001 cents /bit

Disk
T Bytes, 10 ms
(10^{-5}-10^{-6} cents/bit

Tape
infinite
sec-min
10^{-8}



Upper Level

Staging
Xfer Unit

prog./compiler
1-8 bytes

cache cntl
8-128 bytes

OS
512-4K bytes

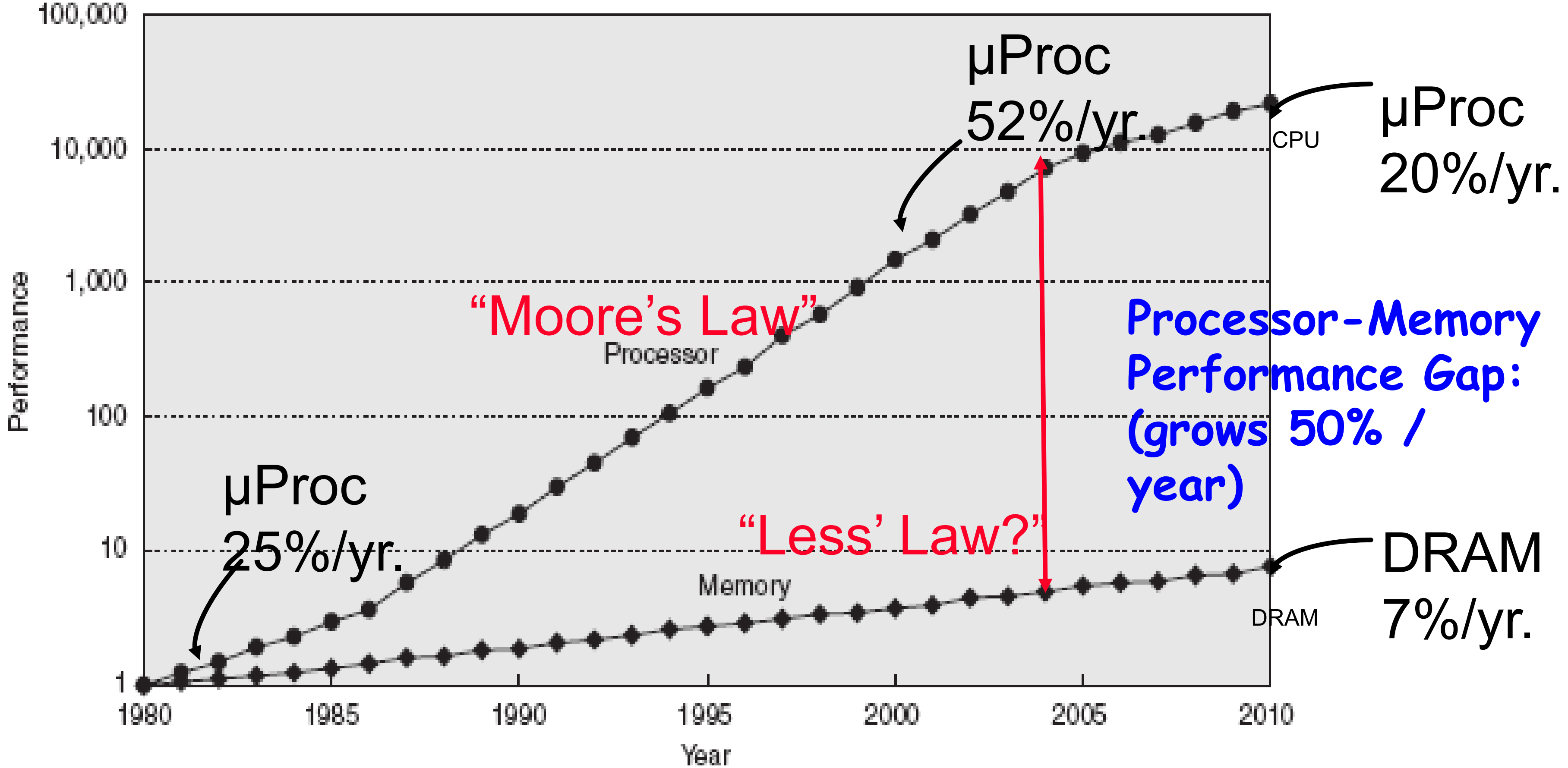
user/operator
Mbytes

faster

Larger

Lower Level

Who Cares About the Memory Hierarchy?



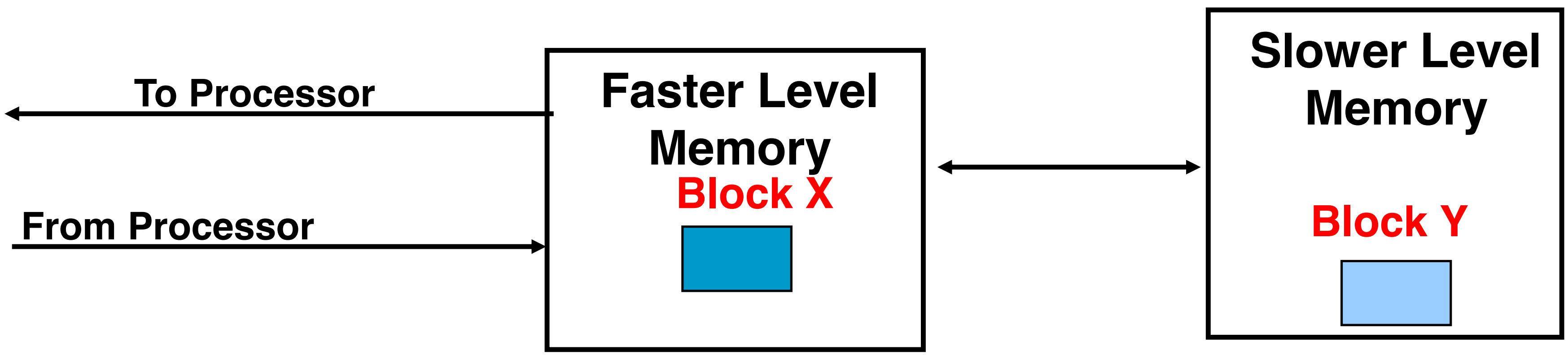
- 1980: no cache in µproc; 2001: 2-level cache on chip (1989 first Intel µproc with a cache on chip)

Different concerns for desktops, servers, and embedded computers

- Desktop computers:
  - primarily running one application for single user
  - concerned more with **average latency** from the memory hierarchy.
- Servers computers:
  - May have hundreds of users running potentially
  dozens of applications simultaneously.
  - concerned about memory **bandwidth**.
- Embedded computers:
  - Used for real-time applications, so **worst-case performance** is a focus
  - Power and battery life, may **NOT** choose **hardware optimizations**
  - Running only one application using very simple OS, so **protection** role of memory **is often diminished**.

Memory Hierarchy Terminology

- **Hit**: data appears in some block in the faster level (Block X)
  - Hit Rate the fraction of memory access found in the faster level
  - Hit Time: Time to access the faster level which consists of
    Memory access time + Time to determine hit/miss
- **Miss**: data needs to be retrieve from a block in the slower level (Block Y)
  - Miss Rate = 1 - (Hit Rate)
  - Miss Penalty: Time to replace a block in the upper level +
    Time to deliver the block to the processor
- Hit Time << Miss Penalty



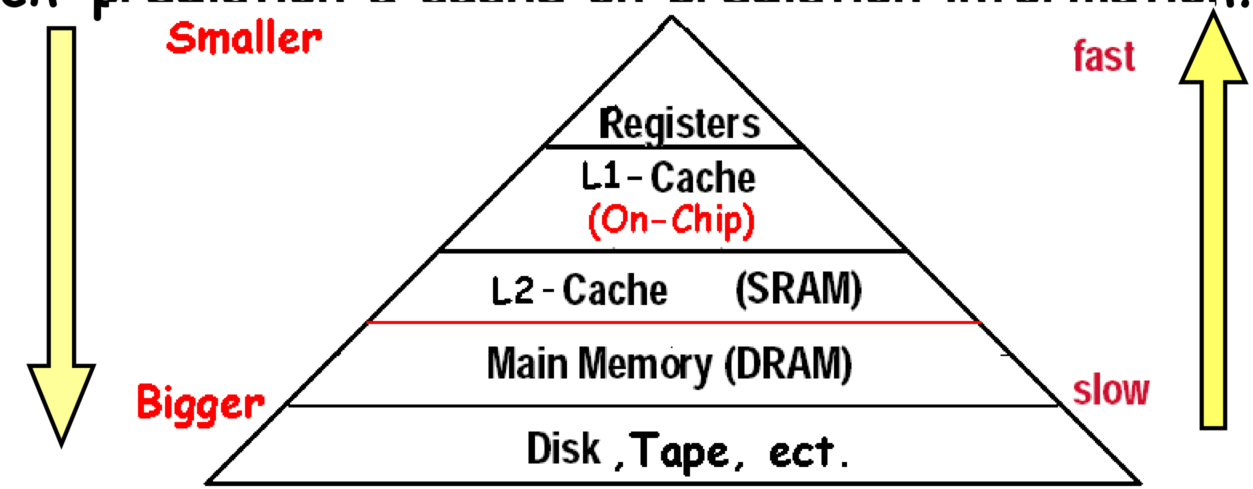
Review of the ABCs of Caches

36 terms of Cache

Cache	Virtual memory	
data cache	Instruction cache	unified cache
block	page	tag field
Block address	index field	block offset
full associative	set associative	**direct mapped**
n-way set associative	set	address trace
misses per instruction	Memory stall cycles	**miss penalty**
Valid bit	**dirty bit**	l**ocality**
cache hit	hit time	
cache miss	miss rate	page fault
Write through	write back	**write allocate**
random replacememt	least-recently used	**no-write allocate**
Average memory access time	write buffer	write stall

What is a cache?

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



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

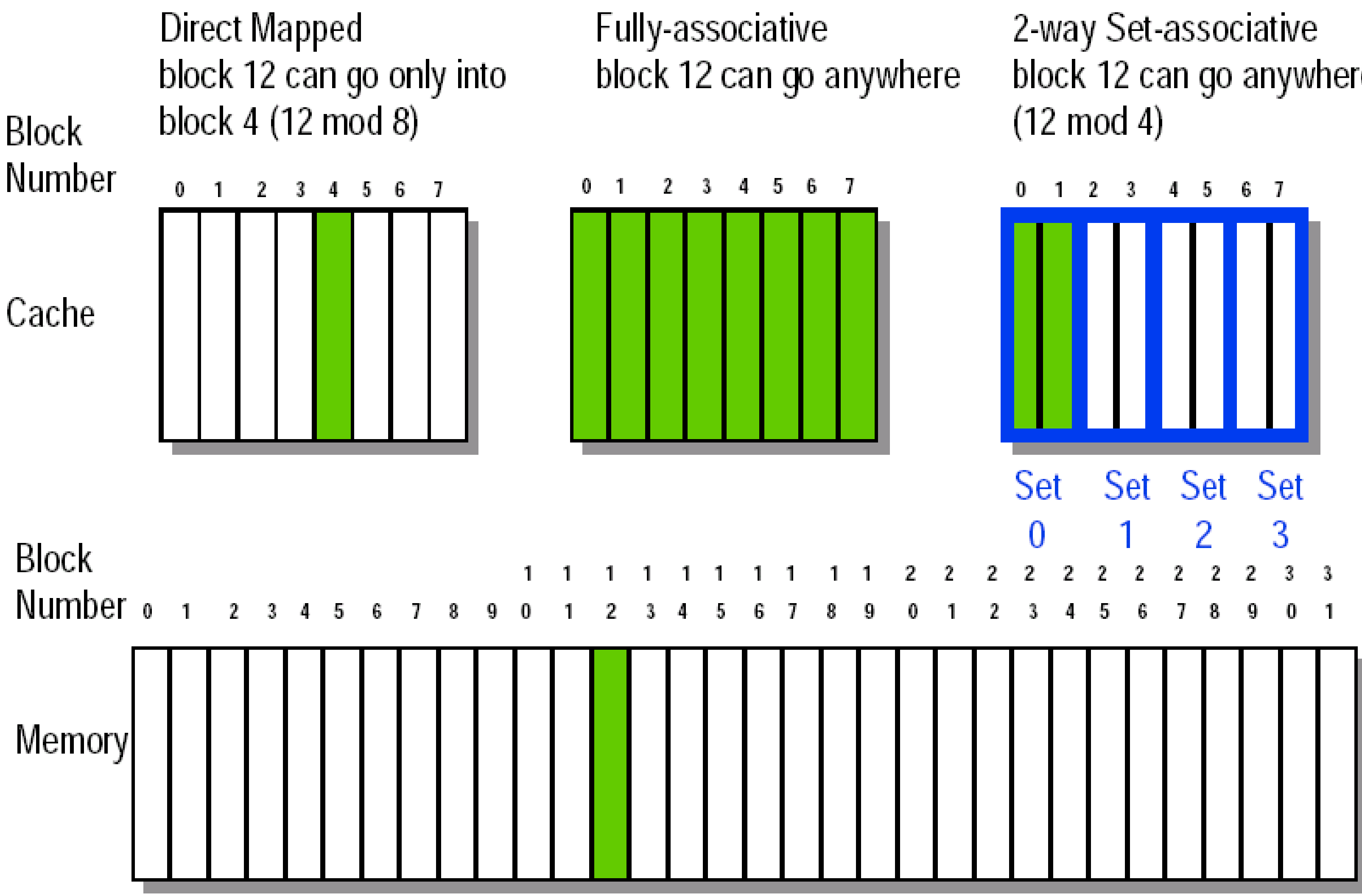
- Direct mapped
  - Block can only go in one place in the cache

    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).**

8-32 Block Placement



Q2: Block Identification

- 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
- 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

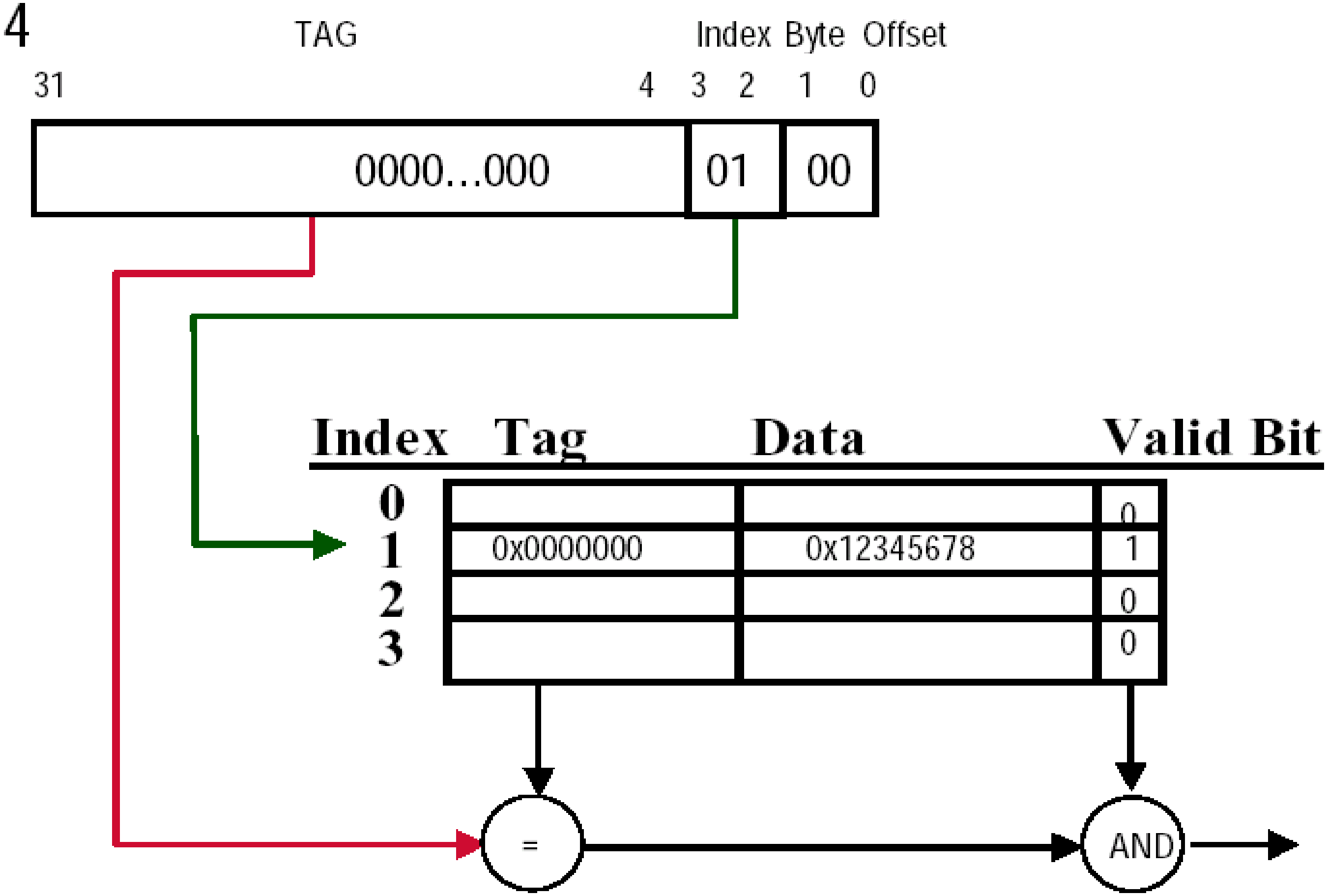
Block Address		Offset
Tag	Index	
Stored in cache and used in comparison with CPU address	Selects set	Selects data within the block

- The Index field selects
  - The set, in case of a set-associative cache
  - The block, in case of a direct-mapped cache
- The Byte Offset field selects
  - The byte within the block
  - Has as many bits as $\log_2(\text{size of block})$
- The Tag is used to find the matching block within a set or in the cache
  - Has as many bits as

    (AddressSize) – (IndexSize) – (ByteOffsetSize)

Direct-mapped Cache Example (1-word Blocks)

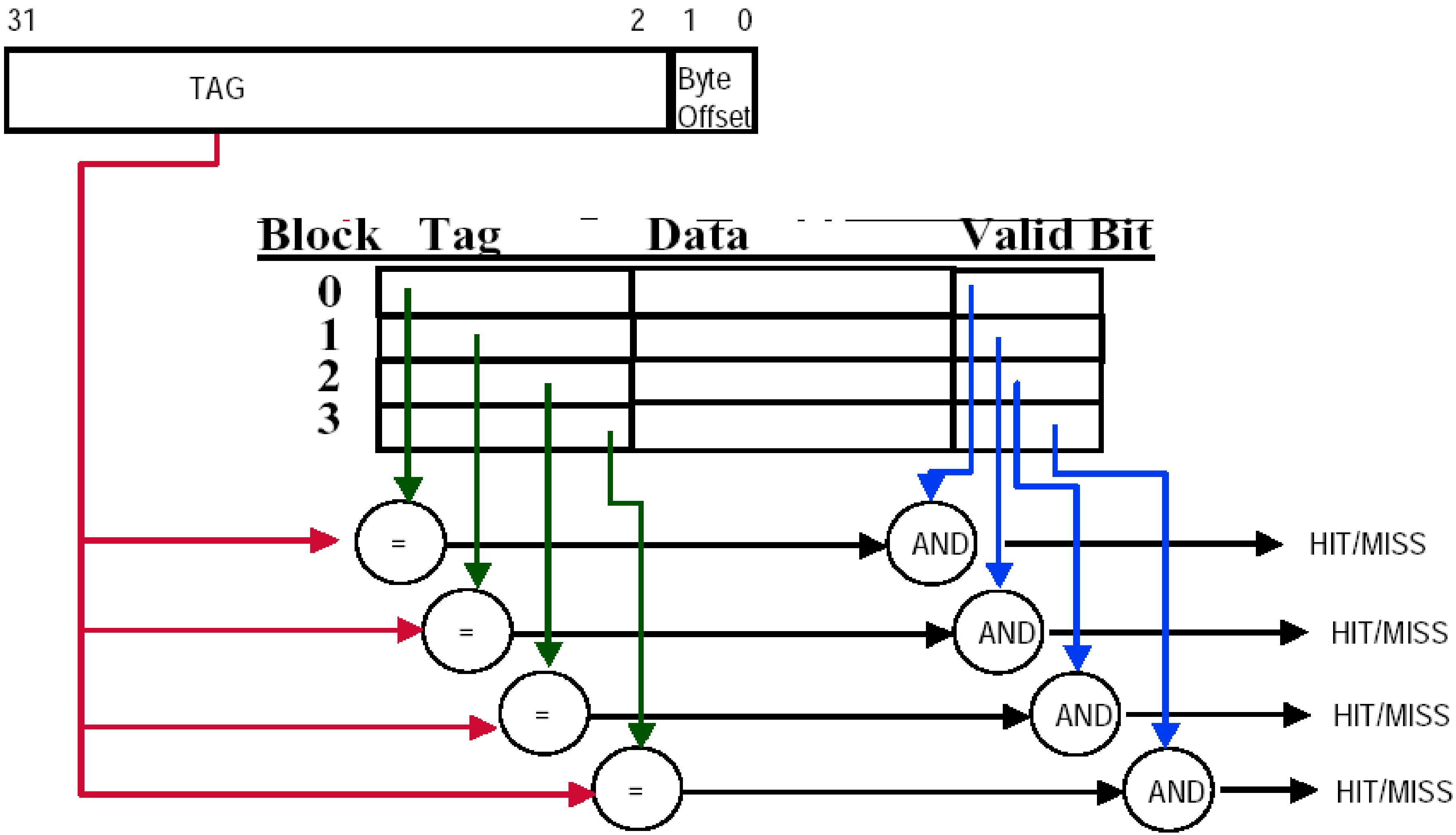
LOAD R1, 0x04



MEMORY

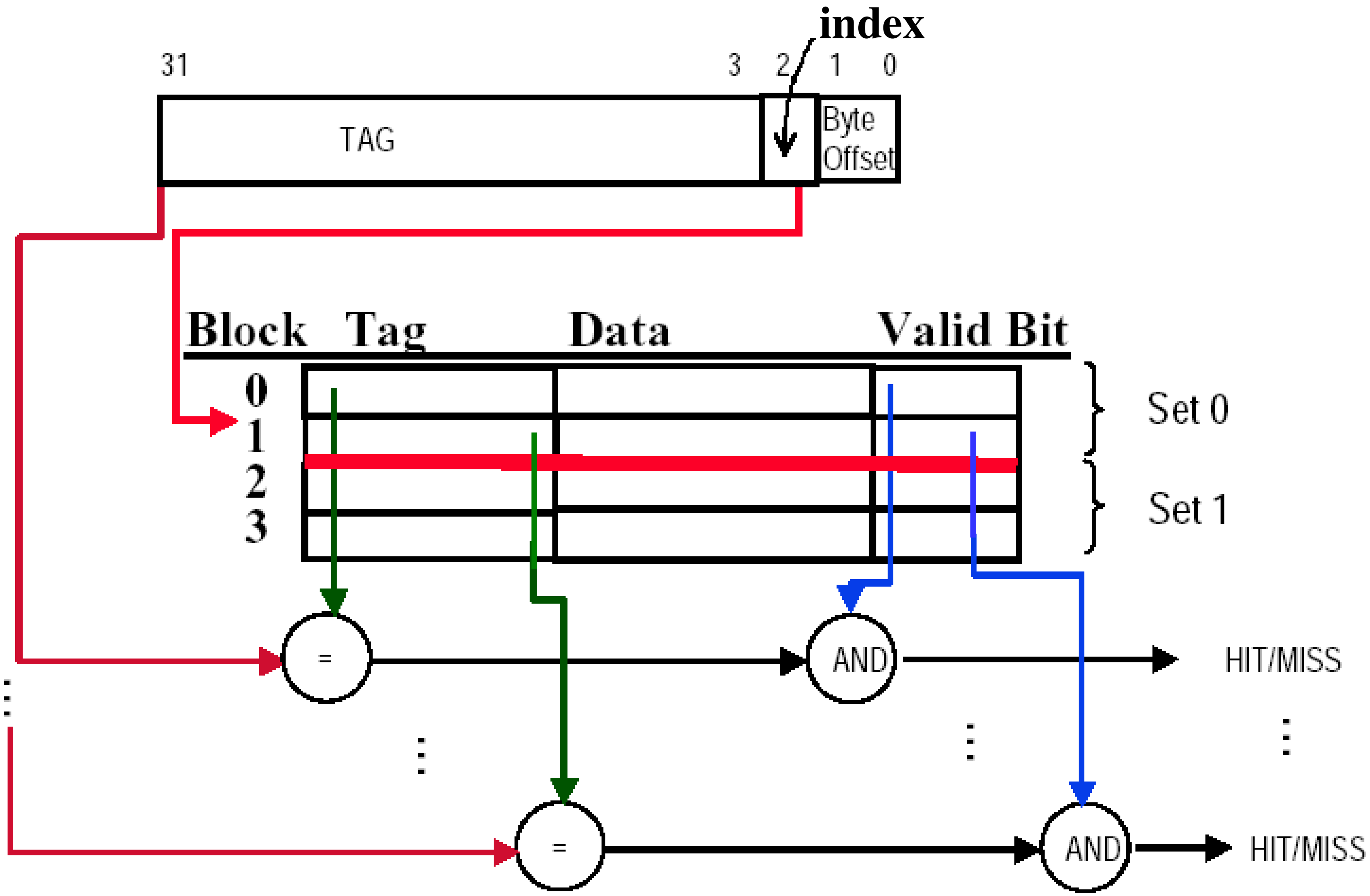
Address	Data
0x00	0x00000000
0x04	0x12345678
0x08	0x87654321
0x0C	0x11111111
0x10	0x22222222
0x14	0x33333333
0x18	0x44444444
0x1C	0x55555555
0x20	0x10101010

Fully-Associative Cache example (1-word Blocks)



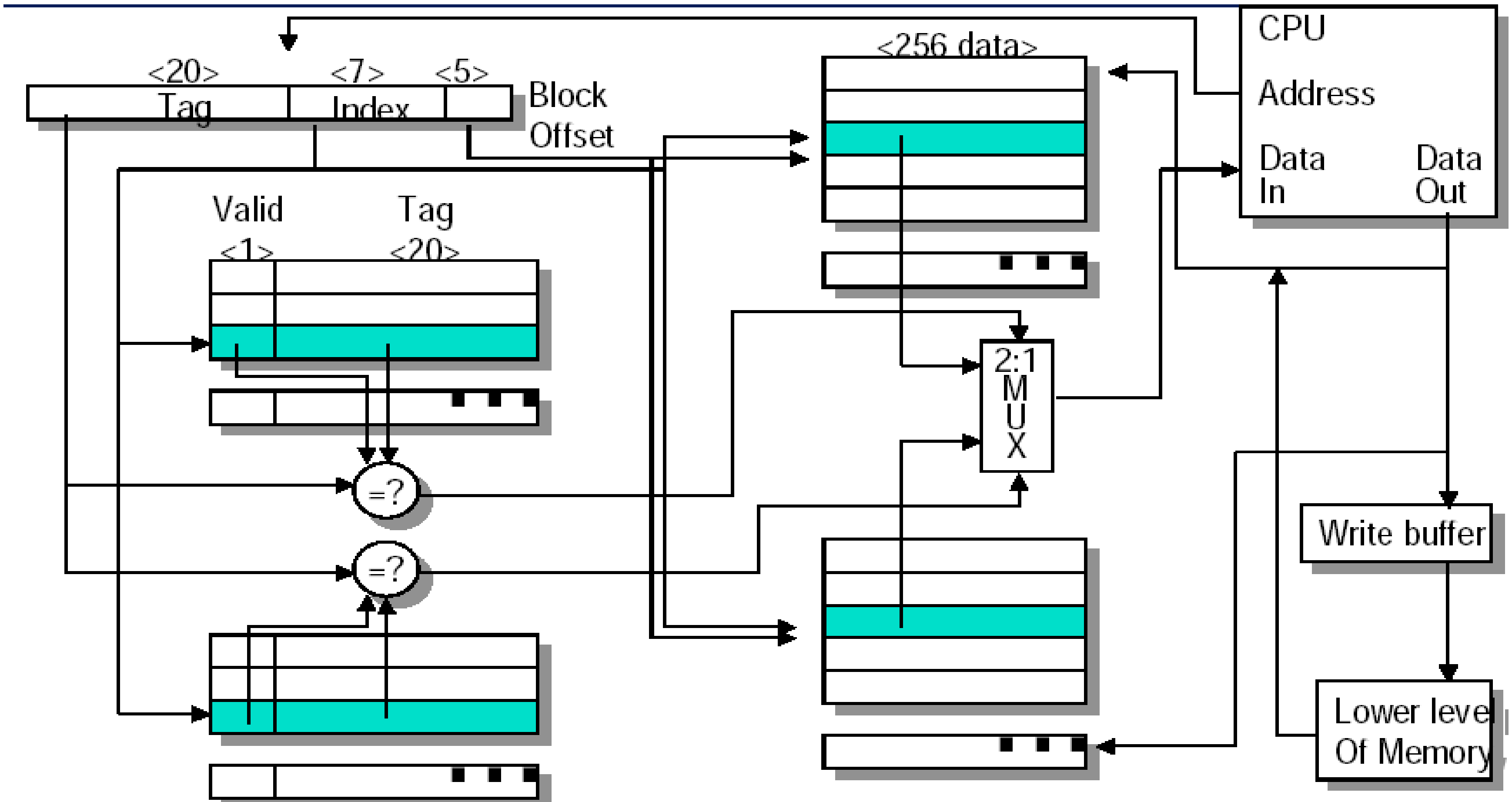
2-Way Set-Associative Cache

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



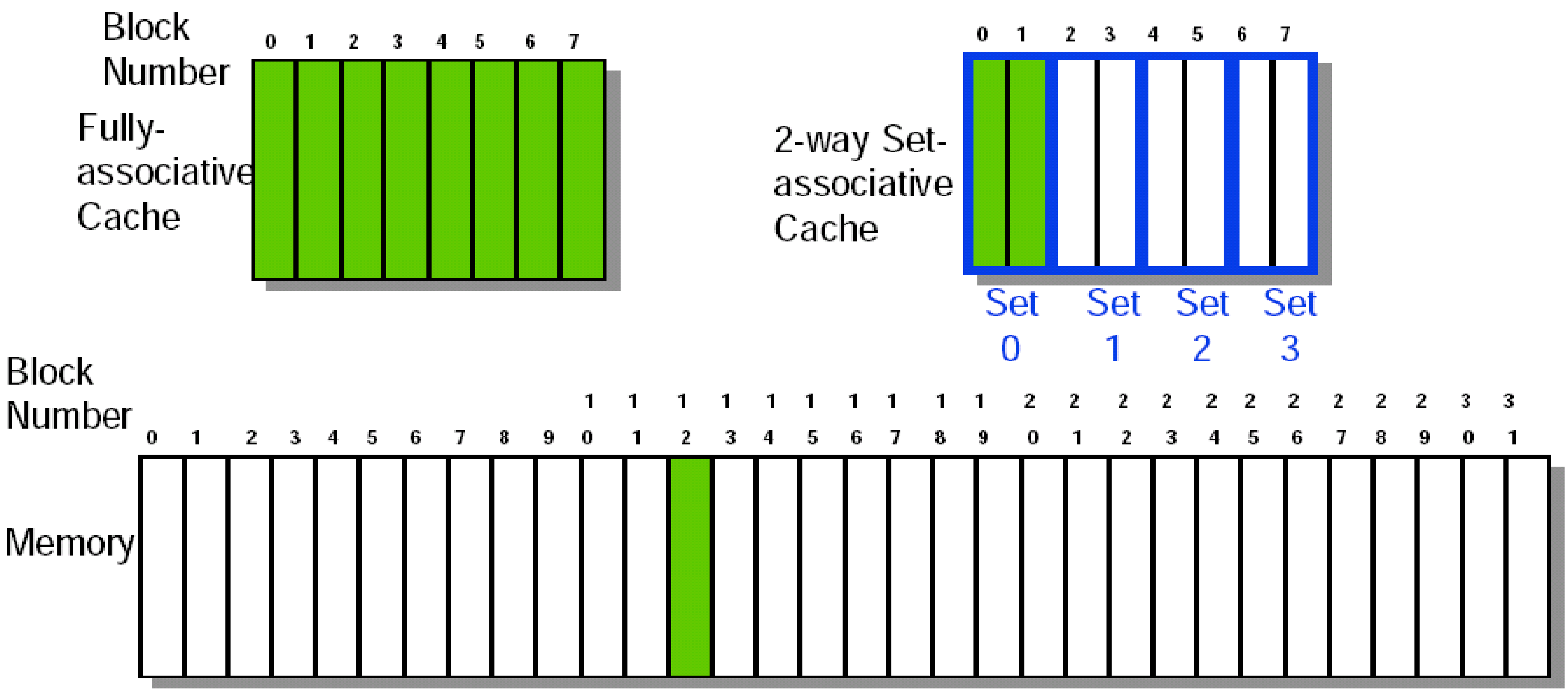
Example: set associate cache

- Memory size: 4G, Cache 8K, 2-way set associate



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

- **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*

Implementation of Replacement

- Psedo LRU
- Example:

	V	NV
A	1	0
B	0	1
C	0	0
D	0	0

- When Miss:

- Kick out the Victim,
- Make the NextVictim to be Victim,
- and select one from the left two blocks to be the NextVictim

Another psedo LRU

- 3 bit for a set (4-way)

- One bit for which is the LRU in AB
- One bit for which is the LRU in CD
- One bit for which is the LRU in AB / CD

Q4: Write Strategy

- When data is written into the cache (on a store), is the data also written to main memory?
- ***write-through*** **:** The information is written to both the block in the cache and to the block in the slower memory
  - Cache control bit: only a *valid* bit
  - memory (or other processors) always have latest data
  - Always combined with write buffers so that don't wait for slow memory
- ***write-back:*** The information is written only to the block in the cache. The modified cache block is written to main memory only when it is replaced
  - Cache control bits: both *valid* and *dirty* bits
  - much lower bandwidth, since No writes to slow memory for repeated write accesses

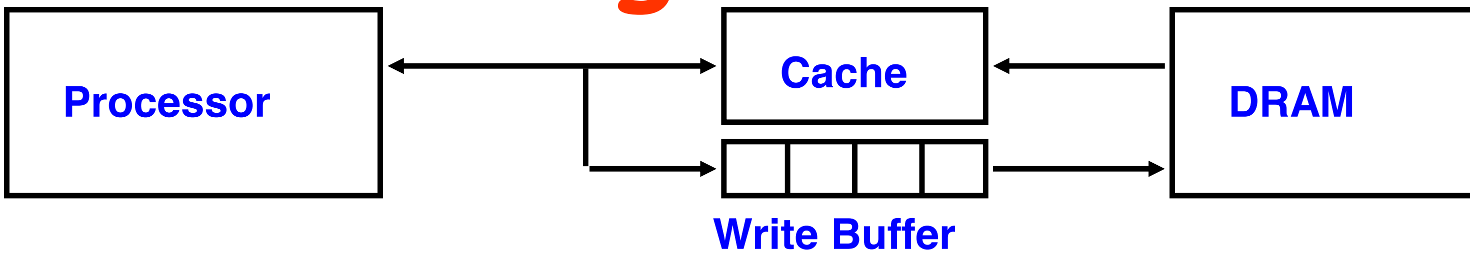
Pros and Cons for write strategy

- 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
  - main memory bandwidth is smaller when multiple writes occur to the same block.

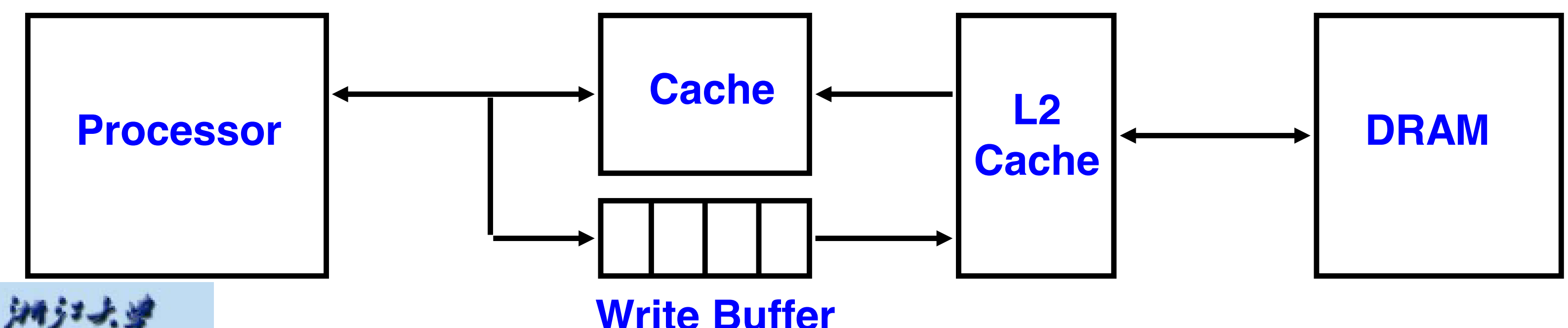
Write stall

- **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*
  - 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.

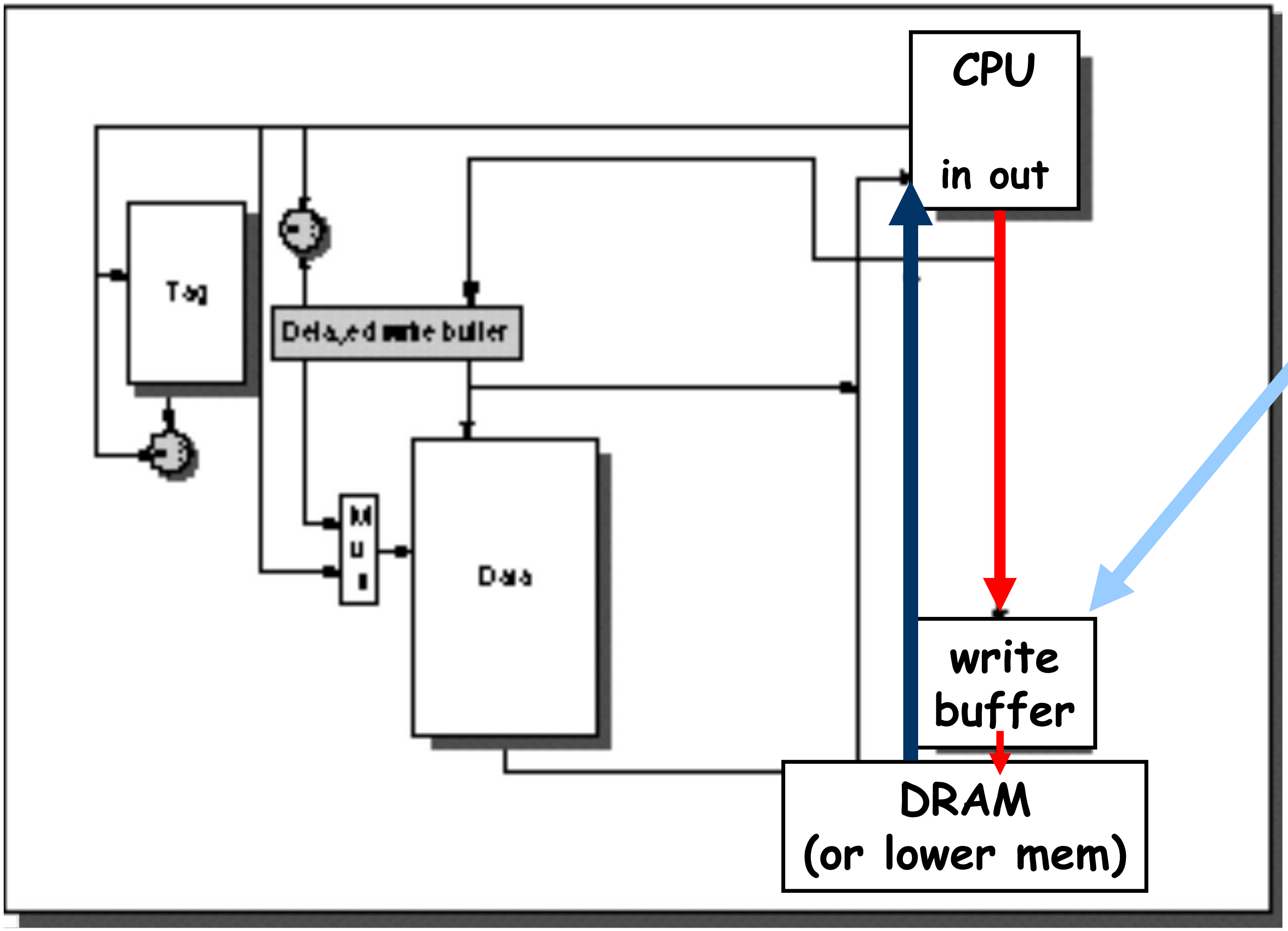
Write Through via Buffering



- Processor writes data into the cache and the write buffer
- Memory controller writes contents of the buffer to memory
- Increased write frequency can cause saturation of write buffer
- If CPU cycle time too fast and/or too many store instr. in a row:
  - Store buffer will overflow no matter how big you make it
  - The CPU Cycle Time get closer to DRAM Write Cycle Time
- Write buffer saturation can be handled by installing a second level (L2) cache



Write buffers



Write policy when 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 .

Example

- Assume a fully associative wtrie-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];

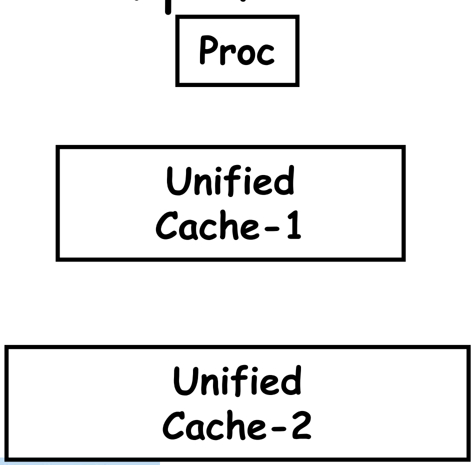
What are the number of hits and misses when using no-write allocate versus write allocate?

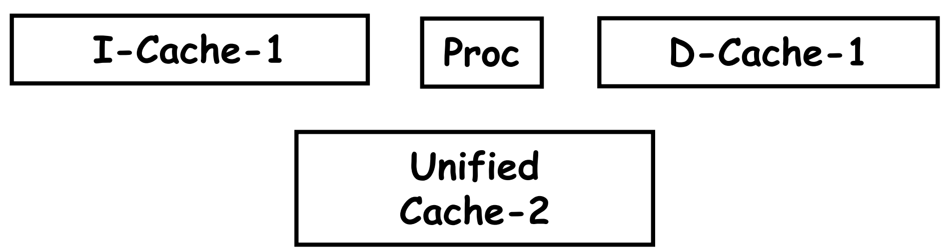
Answer ：

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

Split vs. unified caches

- **Unified cache**
  - All memory requests go through a single cache.
  - This requires less hardware, but also has lower hit rate
- **Split I & D cache**
  - A separate cache is used for instructions and data.
  - This uses additional hardware, though there are some simplifications (the I cache is read-only).





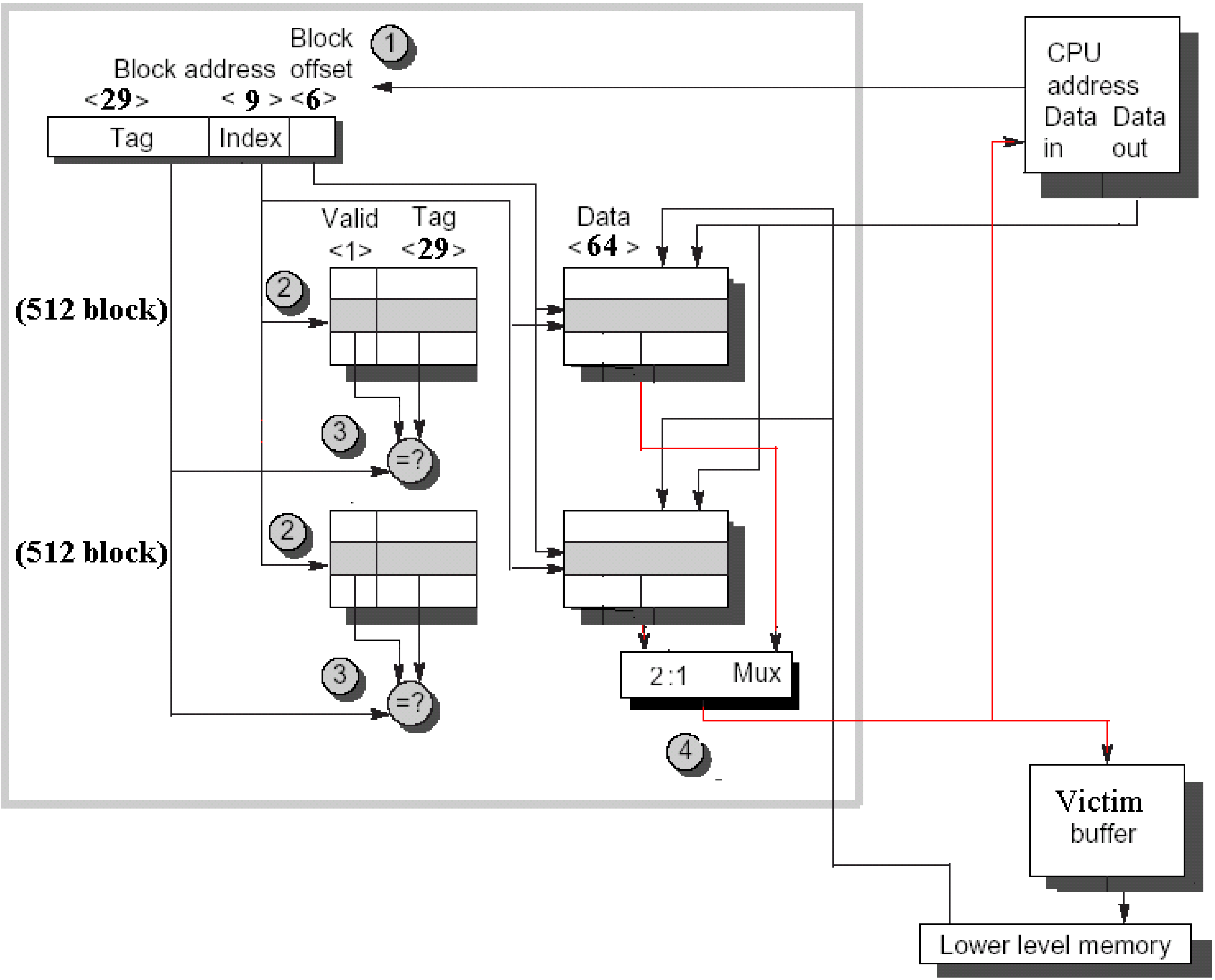
Split vs. mixed cache

- Miss per 1000 instructions for 2-way associate cache.

size	Instruction Cache	Data Cache	Unified Cache
8KB	8.16	44.0	63.0
16KB	3.82	40.9	51.0
32KB	1.36	38.4	43.3
64KB	0.61	36.9	39.4
128KB	0.30	35.3	36.2
256KB	0.02	32.6	32.9

- Average miss rate = Inst% $\times$ MRinst. + Data% $\times$MRdata
- Split： remove the misses due to conflicts between inst. blocks and data blocks , but has fixed cache space for both instructions and data.

Example:Alpha 21264 data cache



Superviser cache / User cache

- Instruction Cache
- Supervisor/ User Space Bit
  - 1: Supervisor access only
  - 0: Supervisor / User access

Procedure for Cache Accessing

Step1 Cache is divided into 2 fields: the 38 bit block address and the 6-bit block offset($64=2^6$ and $38+6=44$).

$$2^{\text{index}} = \frac{\text{Cache size}}{\text{Block size} \times \text{Set associativity}}$$

$$= \frac{65{,}536}{64 \times 2} = 512 = 2^9$$

Step3 the two tags are compared and the winner is selected. Tag contains the valid bit, otherwise the results of the comparion are ignored.

Step4 If one tag does match, CPU loads the proper data from the cache, otherwise from main memory.
The 21264 allows 3 clock cycles for these four steps, so the instructions in the following 2 clock cycles would wait if they tried to use the result of the load.

5.3 Cache performance

CPU Execution time=

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

$$\text{Memory stall cycles} = \text{IC} \times \text{Mem refs per instruction} \times \text{Miss rate} \times \text{Miss penalty}$$

$$CPUtime = IC \times \left(CPI_{Execution} + \frac{MemAccess}{Inst} \times MissRate \times MissPenalty \right) \times CycleTime$$

$$CPUtime = IC \times \left(CPI_{Execution} + \frac{MemMisses}{Inst} \times MissPenalty \right) \times CycleTime$$

$CPI_{Execution}$ includes ALU and Memory instructions

Average Memory Access Time

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

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

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

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

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

Cache performance metrics

- Miss rate
  - Independent of the speed of hardware.
- Average memory access time(AMAT)
  - Better than miss rate , but
  - Indirect measure of performance
- CPUtime

Ex1: Impact on Performance

- Suppose a processor executes at
  - Clock Rate = 200 MHz (5 ns per cycle), Ideal (no misses) CPI = 1.1
  - 50% arith/logic, 30% ld/st, 20% control
- Suppose that 10% of memory operations get 50 cycle miss penalty
- Suppose that 1% of instructions get same miss penalty
- Calculate the AMAT and real CPI.

- **Answer:** **CPI = ideal CPI + average stalls per instruction**

  = 1.1(cycles/ins) + [0.30 (DataMops/ins) × 0.10 (miss/DataMop) × 50 (cycle/miss)] + 1 (InstMop/ins) × 0.01 (miss/InstMop) × 50 (cycle/miss)]

  = (1.1 + 1.5 + .5) cycle/ins = 3.1
- **58% of the time the proc is stalled waiting for memory!**
- **AMAT=(1/1.3)x[1.1+0.01x50]+(0.3/1.3)x[1.1+0.1x50]**
- **=2.54**

Ex2: Impact on Performance

Assume : Ideal CPI=1 (no misses)

- L/S's structure . 50% of instructions are data accesses
- Miss penalty is 25 clock cycles
- Miss rate is 2%
- How faster would the computer be if all instructions were cache hits?

- **Answer: first compute the performance for always hits:**

$CPU_{execution\ time}$ =(CPU clock cycles+memory stall cycles)×clock cycle

=(IC ×CPI+0) ×Clock cycle

=IC ×1.0 ×clock cycle

Answer for example 2 (cont.)

Now for the computer with the real cache, first compute memory

$$Memory\ stall\ cycles = IC \times \frac{Memory\ accesses}{Instruction} \times Miss\ rate \times Miss\ penalty$$

$$= IC \times (1 + 0.5) \times 0.02 \times 25 = IC \times 0.75$$

The total performance is thus:

CPU execution time cache =(IC ×1.0+IC ×0.75) ×Clock cycle

=1.75 ×IC ×Clock cycle

The performance ratio is the inverse of the execution times

$$\frac{\text{CPU execution time}_{\text{cache}}}{\text{CPU execution time}} = \frac{1.75 \times IC \times \text{Clock cycle}}{1.0 \times IC \times \text{clock cycle}}$$

The computer with no cache misses is 1.75 time faster.

Ex3: Impact on Performance

Assume : unified caches: 32K unified cache

- Split cache: 16K D-cache and 16K I-cache
- 36% of the instructions are data transfer instructions
- A hit takes 1 colck cycle
- The miss penalty is 100 clock cycles
- A load/store take 1 extra clock cycle on a unified cache
- Write-through with a write-buffer
  and ignore stalls due to the write buffer
- What is the average memory access time in each case?

MR for Uni.cache & split cache

Miss per 1000 instructions for 2-way associate cache.

size	Instruction Cache	Data Cache	Unified Cache
8KB	8.16	44.0	63.0
16KB	3.82	40.9	51.0
32KB	1.36	38.4	43.3
64KB	0.61	36.9	39.4
128KB	0.30	35.3	36.2
256KB	0.02	32.6	32.9

Answer for example 3

Answer : first let's convert misses per 1000 instructions into miss rate.

$$\text{Miss rate}=\frac{\dfrac{\text{Misses}}{1000\ \text{Instructions}}\ /1000}{\dfrac{\text{Memory accesses}}{\text{Instructions}}}$$

Since every instruction access has exactly one memory access to fetch the instruction, according as Figure 5.8 the instruction miss rate is

$$\text{Miss rate}_{\text{16KB instruction}}=\frac{3.82/1000}{1.0}=0.0038$$

Since 36% of the instructions are data transfers, according as Figure 5.8 the data miss rate is

$$\text{Miss rate}_{\text{16KB data}}=\frac{40.9/1000}{0.36}=0.1136$$

Answer for example 3 (cont.)

Form Figure 5.8 The unified miss rate needs to account for instruction and data accesses:

$$\text{Miss rate}_{\text{32KB unified}} = \frac{43.3/1000}{1.00+0.36} = 0.0318$$

Basing on Figure 2.32 on page 138 there is 74% instruction references in split cache. The average miss rate for the split cache is:

$$(74\% \times 0.0038) + (26\% \times 0.1136) = 0.0323$$

Thus ,a 32KB unified cache has a slightly lower effective miss rate than two 16KB caches.

Answer for Example3 (cont.)

- The average memory access time can be divided into instruction and data accesses:

$$\begin{aligned}&\mathit{Average\,memory\,access\,time}\\&= \%instructions \times (HitTime_{Inst} + MissRate_{Inst} \times MissPenalty_{Inst})\\&\quad + \%\mathrm{data} \times (HitTime_{Data} + MissRate_{Data} \times MissPenalty_{Data})\end{aligned}$$

- **Therefore, the time for each organization is**

Average memory access time$_{split}$

=74%×(1+0.0038×100)+ 26%×(1+0.1136×100)

=(74%×1.38)+(26%×12.36)=1.021+3.214=4.25

Average memory access time$_{unified}$

=74%×(1+0.0318×100)+ 26%×(1+1+0.0318×100)

=(74%×4.18)+(26%×5.18)=3.093+1.347=4.40

Ex4: Impact on Performance

Assume: in-order execution computer, such as the Ultra SPARC III.

Miss penalty: 100 clock cycles
Miss rate : 2%
Memory references Per instruction: 1.5
Average cache misses per 1000 instructions: 30
CPI ＝1.0(ignoring memory stalls)

What is the impact on performance when behavior of the cache is included (Calclate the impact using both misses per instruction and miss rate.)?

Answer for example 4

Answer: The performance, including cache misses, is

$$CPU_{time} = IC \times \left(CPI_{exexution} + \frac{Memstallclockcycles}{Instruction} \right) \times Clockcycletime$$

CPU time $_{\text{with cache}}$ =

$= IC \times (1.0 + (30/1000 \times 100)) \times$ Clock cycle time

$= IC \times 4.00 \times$ Clock cycle time

Now caculating performance using miss rate:

$$CPU_{time} = IC \times \left(CPI_{exexution} + Missrate \times \frac{Mem\ accesses}{Instruction} \times Misspenalty \right) \times Clockcycletime$$

CPU time $_{\text{with cache}}$ =

$= IC \times (1.0 + (1.5 \times 2\% \times 100)) \times$ Clock cycle time

$= IC \times 4.00 \times$ Clock cycle time

Answer for example 4 (cont.)

- The clock cycles time and instruction count are the same, with or without a cache. Thus, CPU time increases fourfold, with CPI from 1.00 a "perfect cache" to 4.00 with a cache that can miss.
- Without any memory hierarchy at all the CPI would increase again to $1.0+100\times1.5$ or 151—factor of almost 40 time longer than a system with a cache.

Cache misses have a double-barreled impact on a CPU

- The lower the $CPI_{execution}$, the higher the relative impact of a fix number of cache miss clock cycle.
- When calculating CPI, the cache miss penalty is measured in CPU clock cycles for a miss. Therefore, even if memory hierarchies for two computers are identical, the CPU with the higher clock rate has a larger number of clock cycles per miss and hence a higher memory portion of CPI.

Ex5: Impact on Performance

Assume: CPI=2 (perfect cache) clock cycle time＝1.0 ns

- MPI(memory reference per instruction)＝1.5
- Size of both caches is 64K and size of bath block is 64 bytes
- One cache is direct mapped and other is two-way set associative. the former has miss rate of 1.4%, the latter has miss rate 1.0%
- The selection multiplexor forces CPU clock cycle time to be stretched 1.25 times
- Miss penalty is 75ns,and hit time is 1 clock cycle

- What is the impact of two diffect cache organizations on performance of CPU (first,calculate the average memory access time and then CPU performance.)?

Answer for example 5

Answer : Average memory access time is

Average memory access time = Hit time + Miss rate × miss penalty

Thus, the time for each organization is

$$\text{Average memory access time}_{1\text{-way}} = 1.0 + (0.014 \times 75) = 2.05 \text{ ns}$$

$$\text{Average memory access time}_{2\text{-way}} = 1.0 \times 1.25 + (0.01 \times 75) = 2.00 \text{ ns}$$

The average memory access time is better for the 2-way set-associative cache.

Answer for example 5 (cont.)

CPU performance is

$$CPUtime = IC \times \left(CPI_{execution} + \frac{Misses}{Instruction} \times Misspenalty \right) \times Clockcycletime$$

$$= IC \times \left[\left(CPI_{execution} \times Clockcycletime \right) + \left(Missrate \times \frac{Memory\,accesses}{Instruction} \times Misspenalty \times Clockcycletime \right) \right]$$

Substituting 75 ns for (miss penalty×Clock cycle time), the performance of each cache organization is

CPU $time_{1\text{-}way}$=IC×(2×1.0+(1.5 ×0.014 ×75))=3.58 ×IC

CPU $time_{2\text{-}way}$=IC×(2×1.0×1.25+(1.5 ×0.010 ×75))=3.63 ×IC

Answer for example 5 (cont.)

Relative performance is

$$\frac{CPU\ time_{2-way}}{CPU\ time_{1-way}} = \frac{3.63 \times Instruction\ count}{3.58 \times Instruction\ count} = \frac{3.63}{3.58} = 1.01$$

In contrast to the results of average memory access time, the direct-mapped lesds to slighly better average performance. Since CPU time is our bottom-line evaluation.

Miss penalty and Out-of-order Execution Processors

How do you define "miss penalty"?

- Is it the full latency of the miss to memory, or is it just the "exposed" or nonoverlapped latency when the processor must stall?
- To In-order processor, there is out of question, but here is out of the question.
- Refine memory stalls to lead to a new definition of miss penalty as nonoverlapped latency :

$$\frac{Memory stall cycles}{instruction} = \frac{Misses}{instruction} \times (Total miss latency - Overlapped miss latency)$$

Two definition

- **Length of memory latency** –What to consider as the start and the end of a memory operation in an out-of-order processor.
- **Length of latency overlapped**—What is the start of overlap with the processor(or equivalently, when do we say a memory operation is stalling the processor)

Ex6: Performance on out-of-order processor

Assume: 30% of the 75 ns (52.5) miss penalty can be overlapped; Another parameters are same with example 5 (above example)

What is the impact of performance for out-of-order (OOO) CPU in direct-mapped cache?

Answer: Average memory access time for the OOO computer is:

$$\text{Average memory access time}_{1\text{-way},OOO} = 1.0 * 1.25 + (0.014 \times 52.5) = 1.99 \text{ ns}$$

The performance of the OOO cache is:

$$\text{CPU time}_{1\text{-way},OOO} = IC \times (2 \times 1.0 * 1.25 + (1.5 \times 0.014 \times 52.5)) = 3.60 \times IC$$

Hence, despite a much slower clock cycle time and the higher miss rate of a direct-mapped cache, the OOO computer can be slightly faster if it can hide 30% of the miss penalty.

How to Improve Cache Performance?

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

1. Reduce the time to hit in the cache.--4
   ——small and simple caches, avoiding address translation, way prediction , and trace caches
2. Increase cache bandwidth .--3
   —— pipelined cache access, multibanked caches, non-blocking caches,
3. Reduce the miss penalty--4
   ——multilevel caches, critical word first, read miss prior to writes, merging write buffers, and victim caches
4. Reduce the miss rate--4
   ——larger block size, large cache size, higher associativity,and compiler optimizations
5. Reduce the miss penalty and miss rate via parallelism--2
   ——hardware prefetching,and compiler prefetching