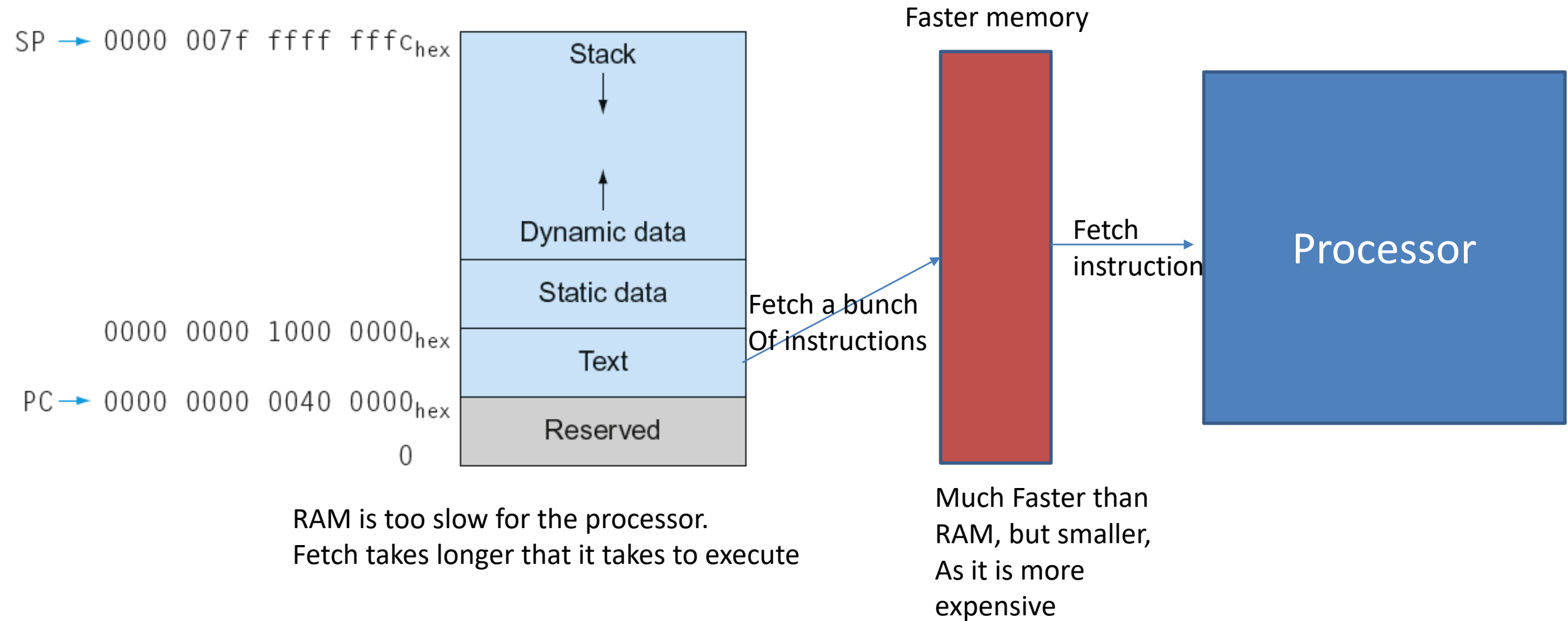


Computer Organization and Architecture

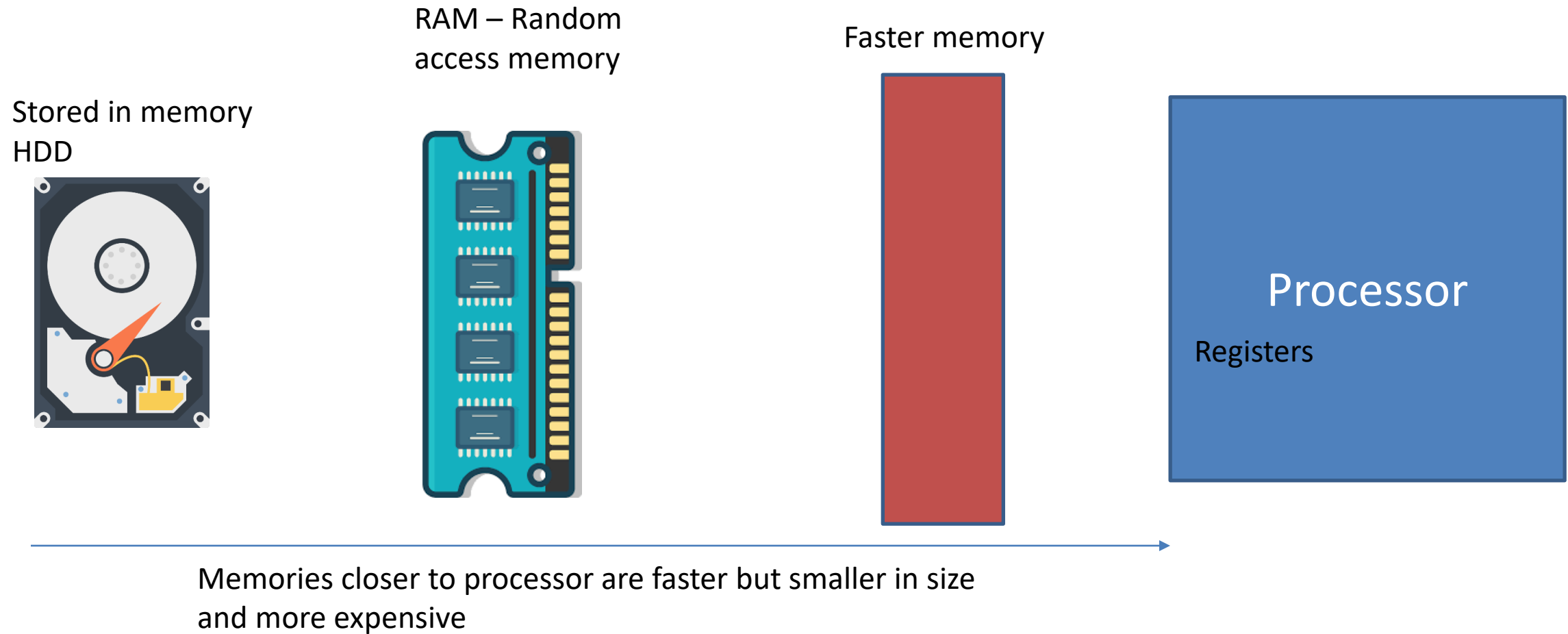
Lecture – 24

Nov 9th , 2022

Executing instructions



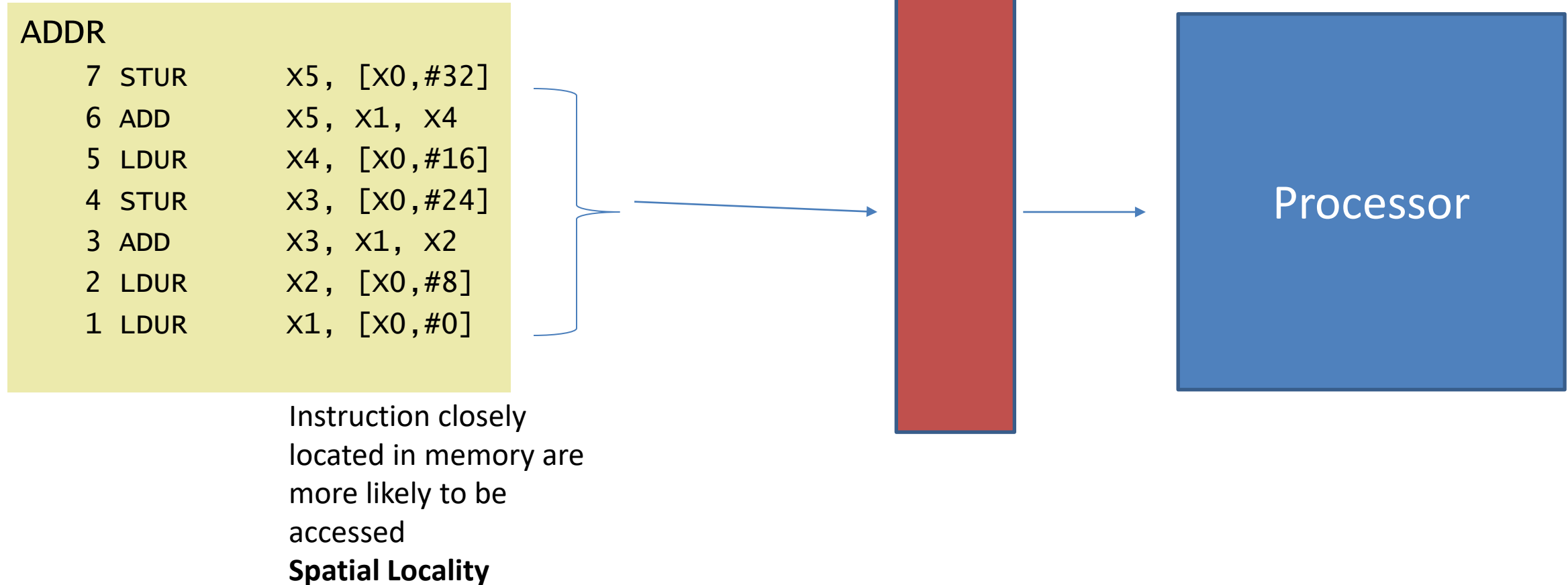
Hierarchy of memories



Principle of Locality

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


Example



Example - 2

ADDR

5 *CBNZ X2, 2*
4 *SUBI X2, X0, #100*
3 *ADDI X0, X0, #1*
2 *ADDI X1, X1, #1*
1 *ADD X0, XZR, XZR*

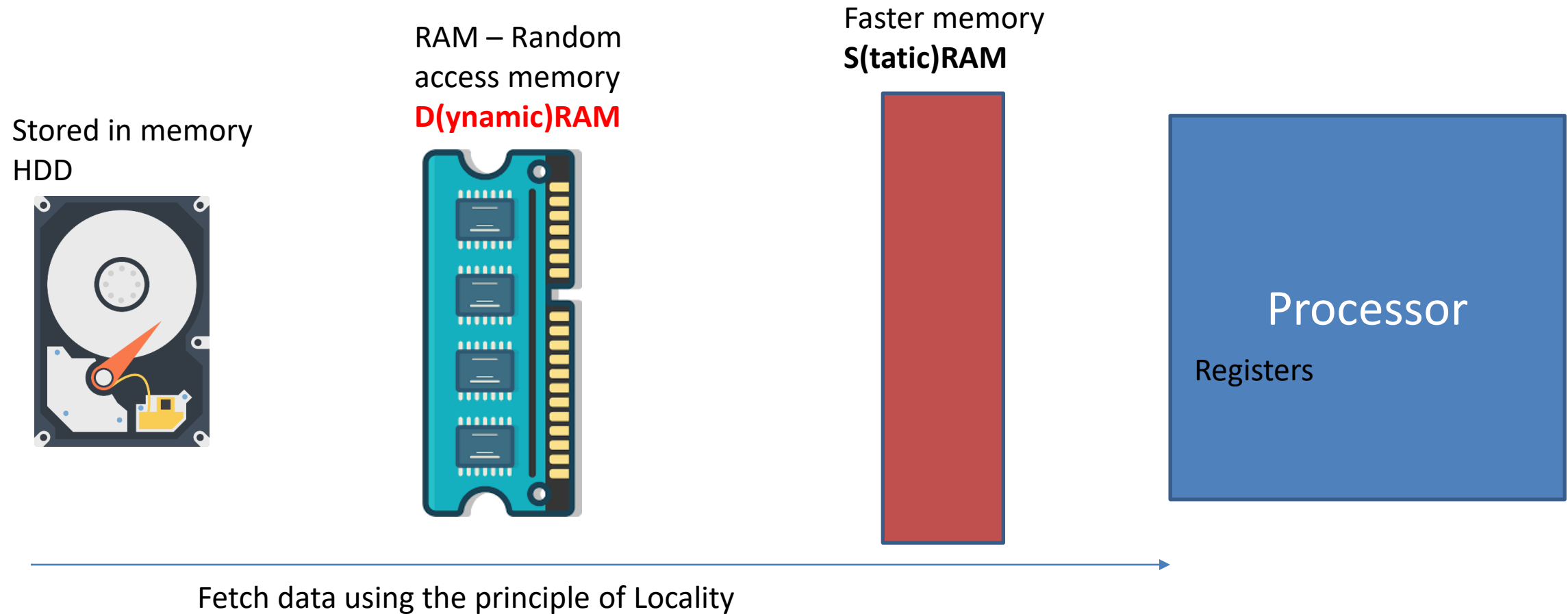


Processor

Instructions 2, 3, 4, and 5 are accessed a 100 times repeatedly.
Instructions accessed once, tend to be accessed again

Temporal Locality

Hierarchy of memories



Taking Advantage of Locality

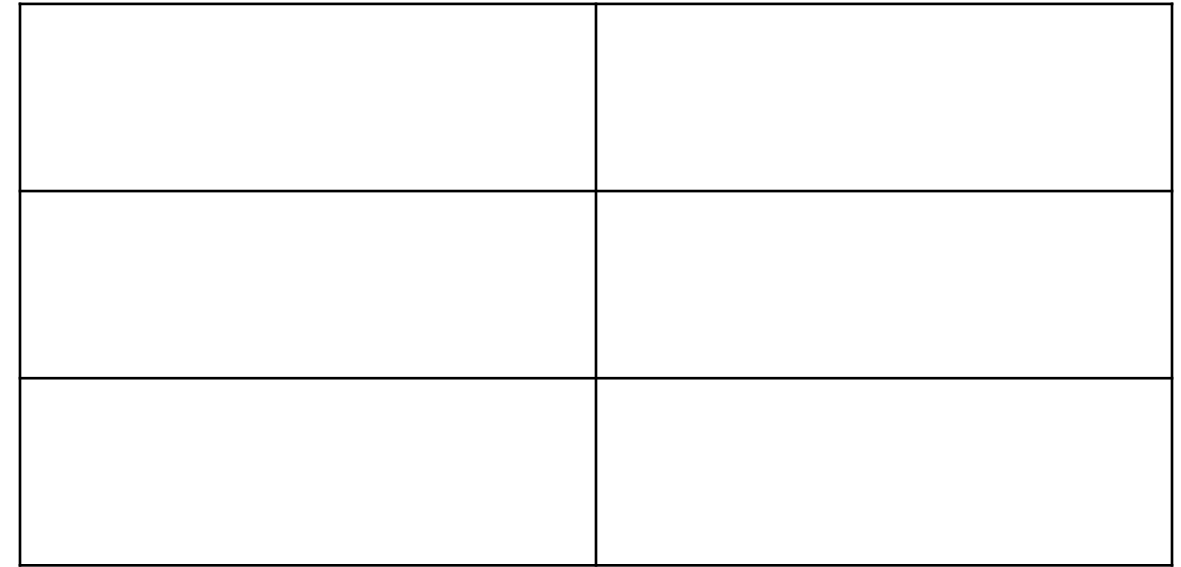
- Memory hierarchy
- Store everything on disk
- Copy recently accessed (and nearby) items from disk to smaller DRAM memory
 - Main memory
- Copy more recently accessed (and nearby) items from DRAM to smaller SRAM memory
 - Cache memory attached to CPU

Memory Hierarchy Levels

ADDR

```
7 STUR    x5, [x0,#32]
6 ADD     x5, x1, x4
5 LDUR    x4, [x0,#16]
4 STUR    x3, [x0,#24]
3 ADD     x3, x1, x2
2 LDUR    x2, [x0,#8]
1 LDUR    x1, [x0,#0]
```

DRAM

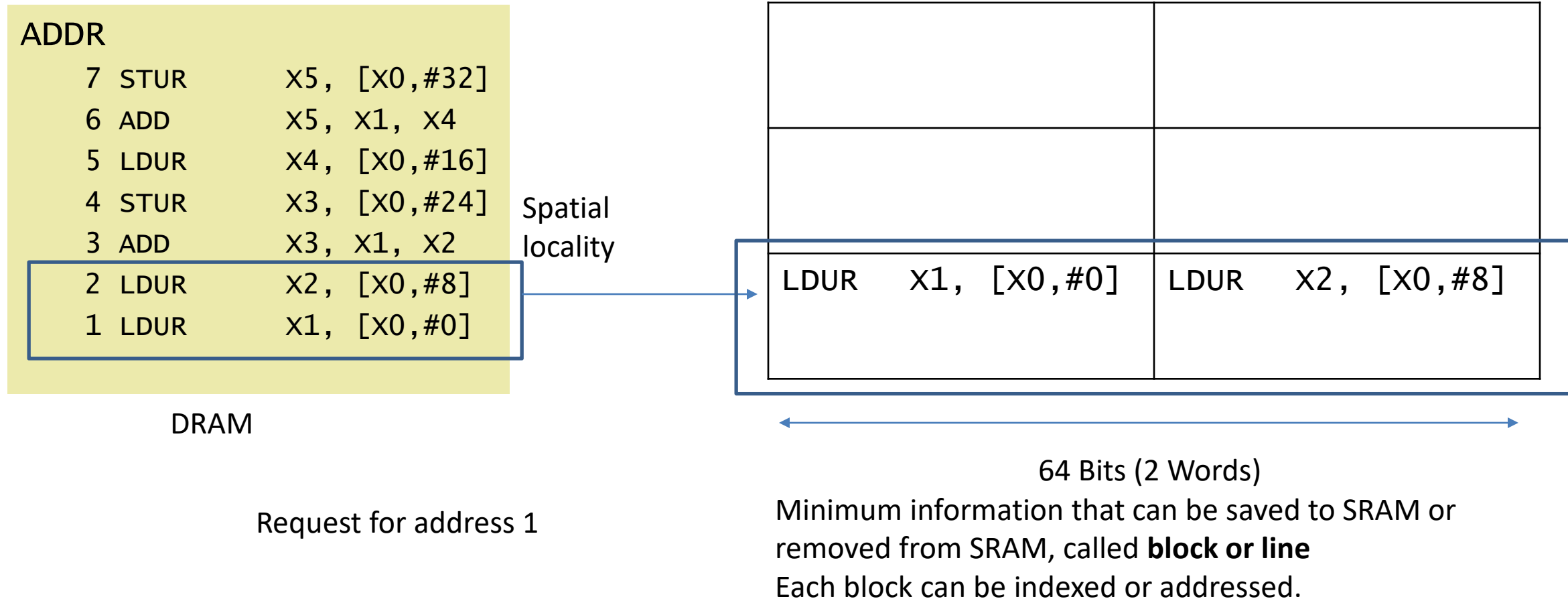


Request for address 1

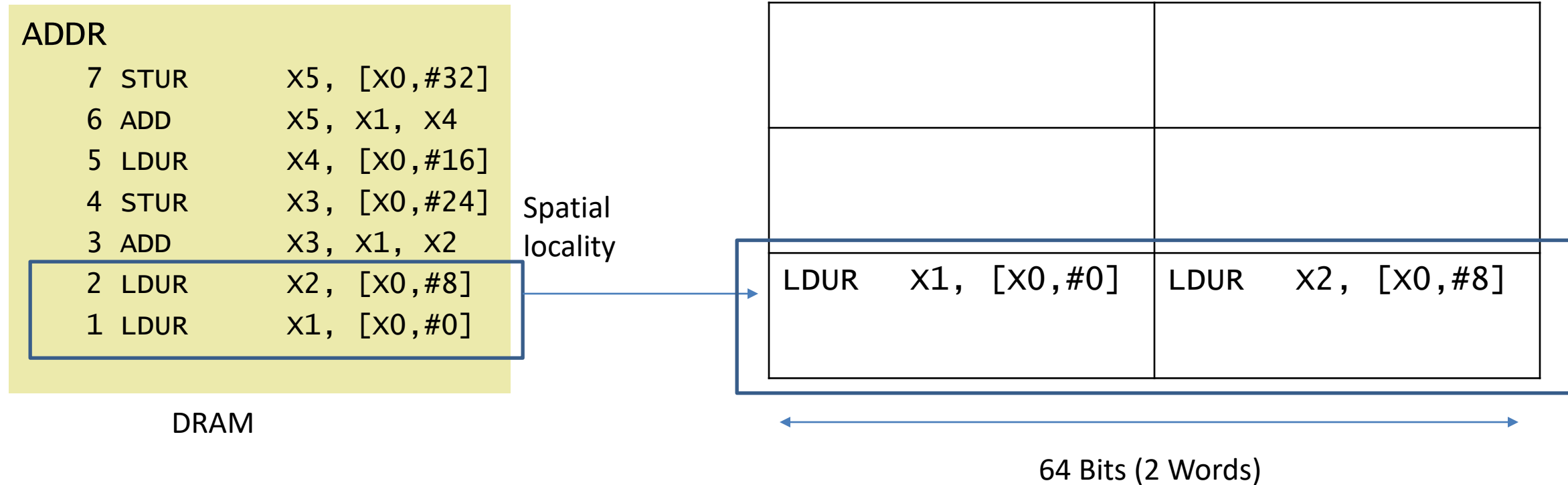
Address 1 is not available in SRAM.

This request is called a **Miss**

Memory Hierarchy Levels

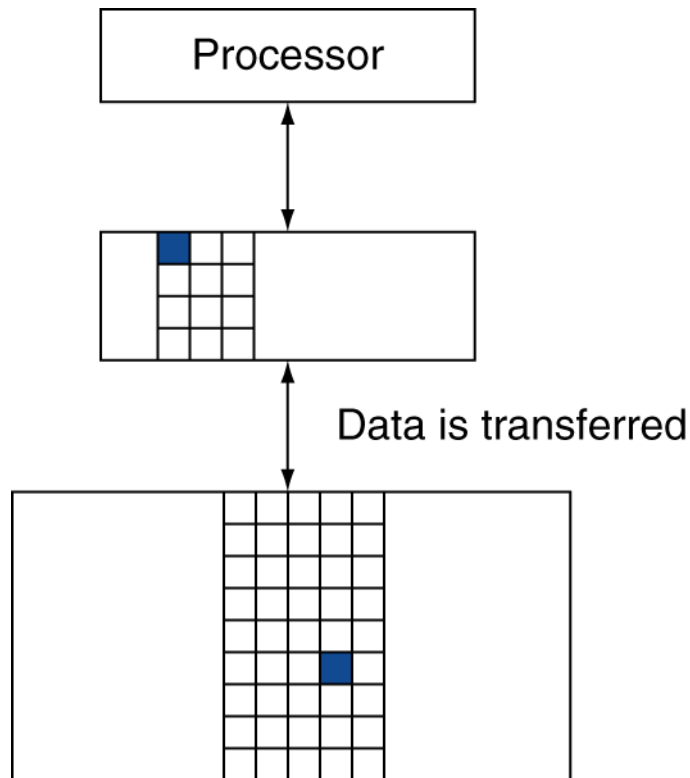


Memory Hierarchy Levels



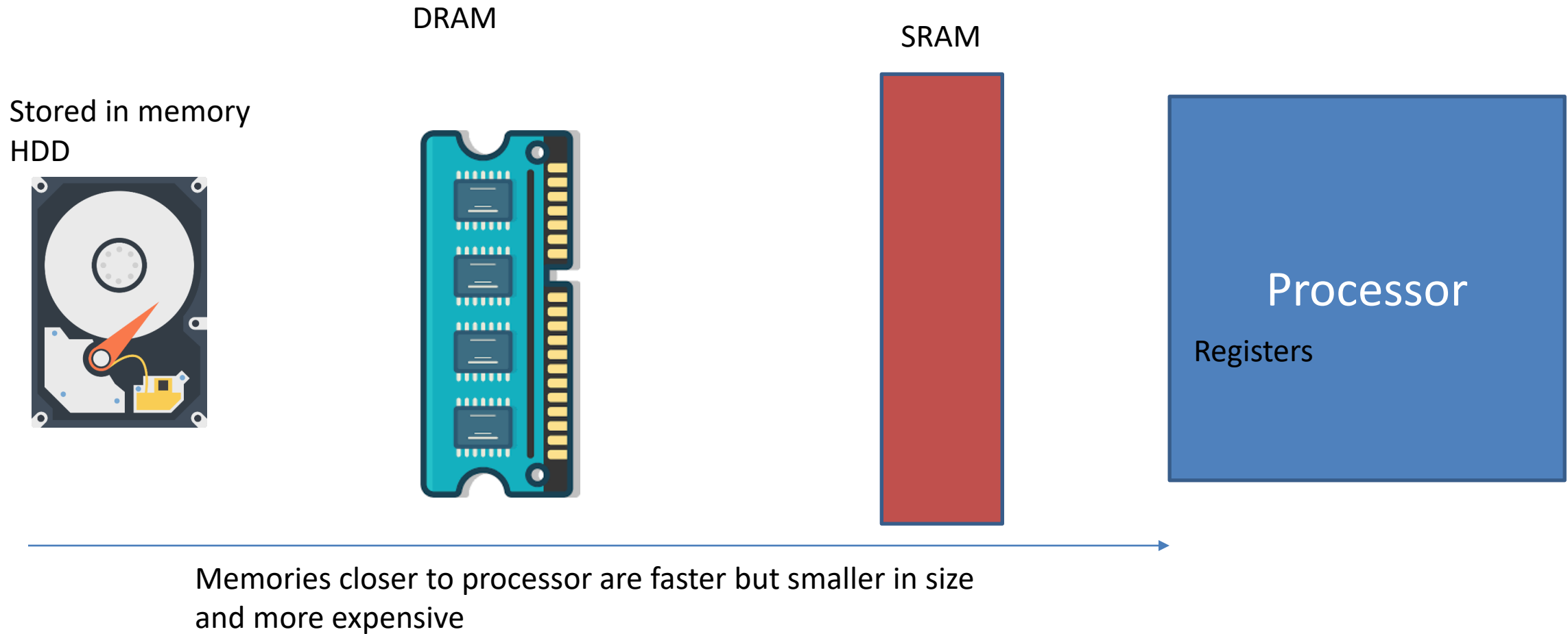
Request for address 2
Already available in SRAM
This request is called a **HIT**

Memory Hierarchy Levels



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

Memory Technologies

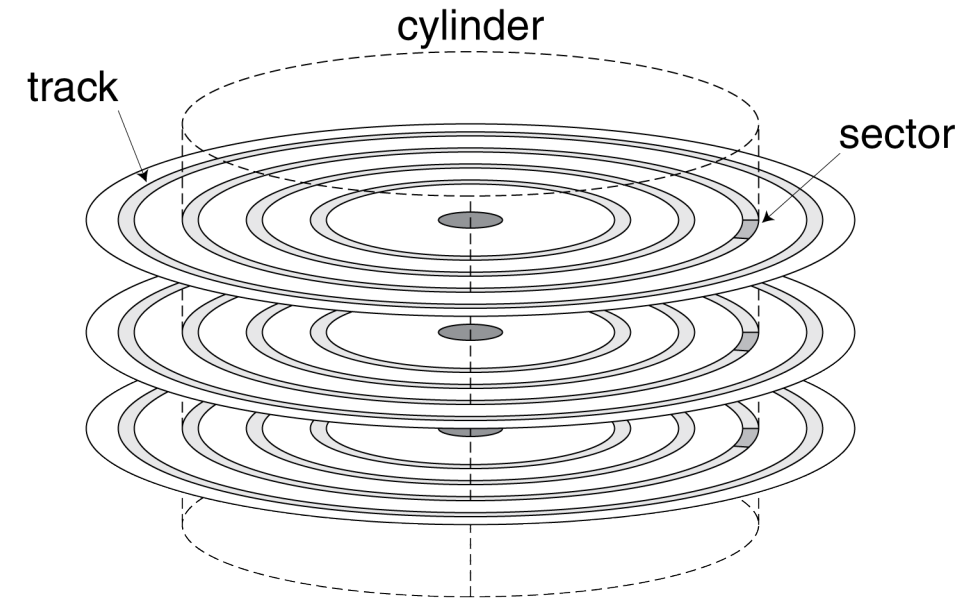


Memory Technology

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

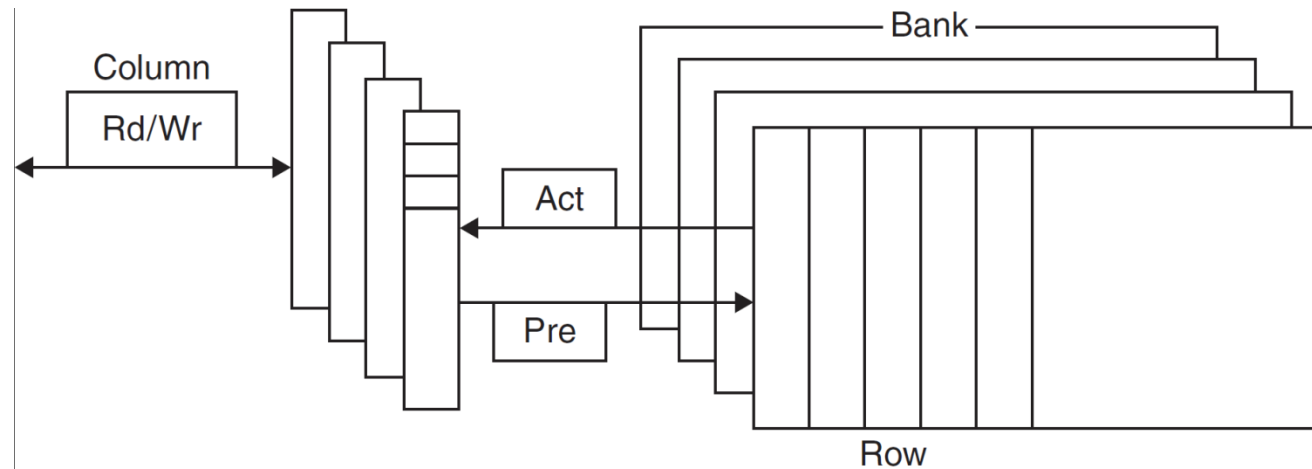
Disk Storage

- Nonvolatile, rotating magnetic storage



D(ynamic)RAM Technology

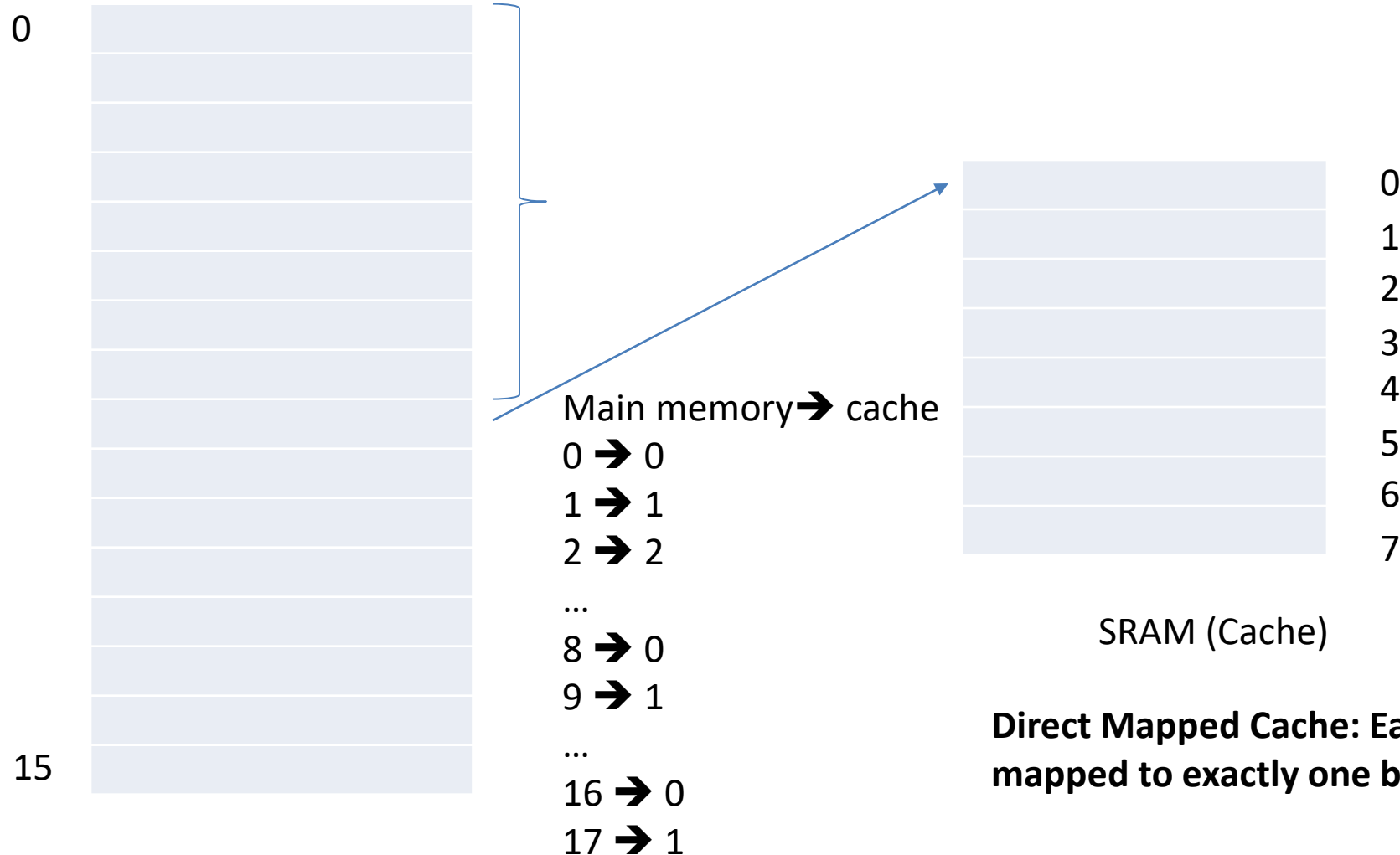
- Data stored as a charge in a capacitor
 - Single transistor used to access the charge
 - Must periodically be refreshed
 - Read contents and write back
 - Performed on a DRAM “row”



S(tatic)RAM Technology

- *Static Random Access Memory (SRAM)*
 - Requires low power to retain bit
 - Fixed access time to data
 - No need to refresh (unlike DRAM)
 - Requires 6 transistors/bit
 - Expensive

Example



Fetch 9:

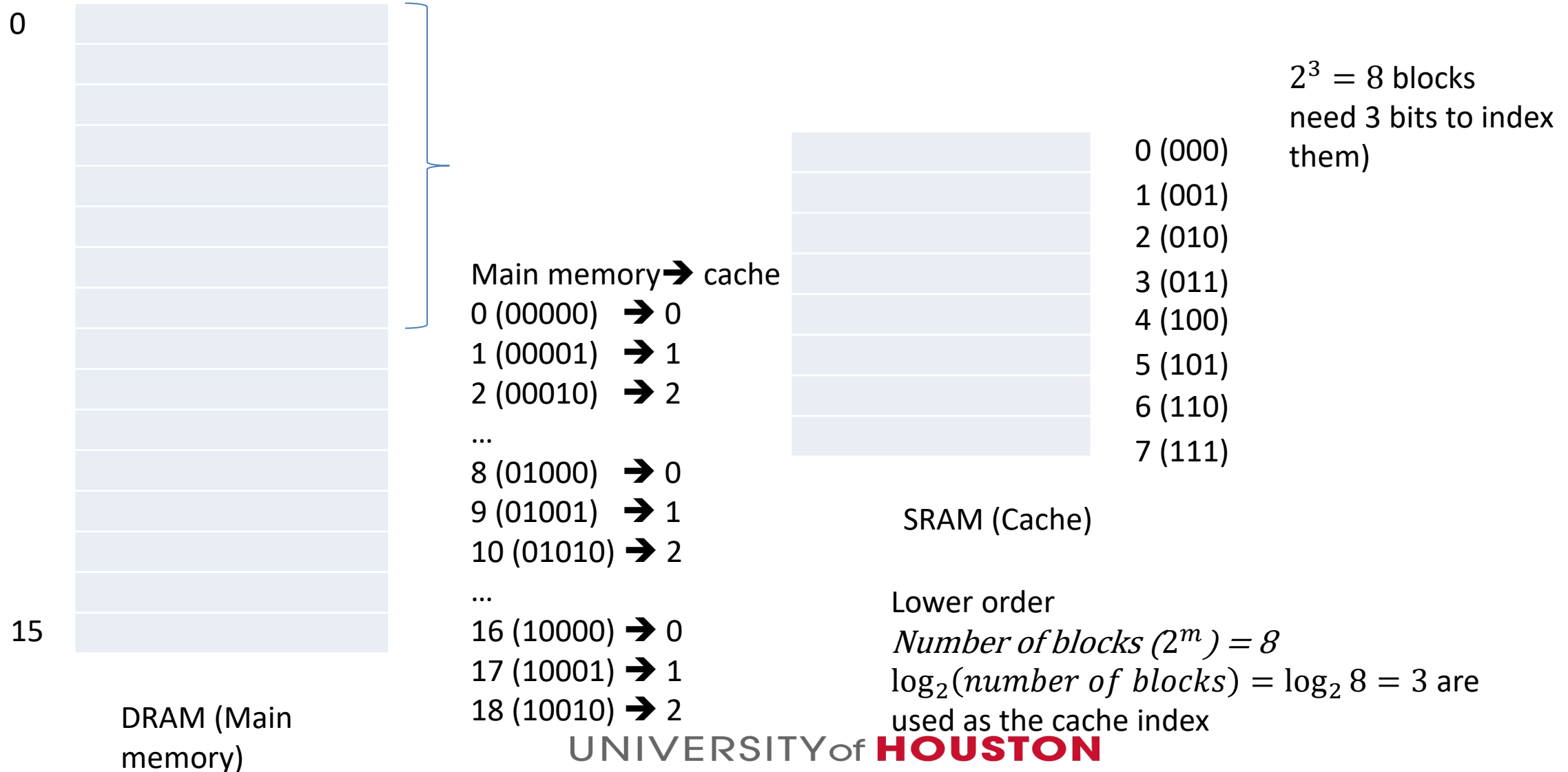
Not available in cache, so fetch from main memory

Temporal locality:

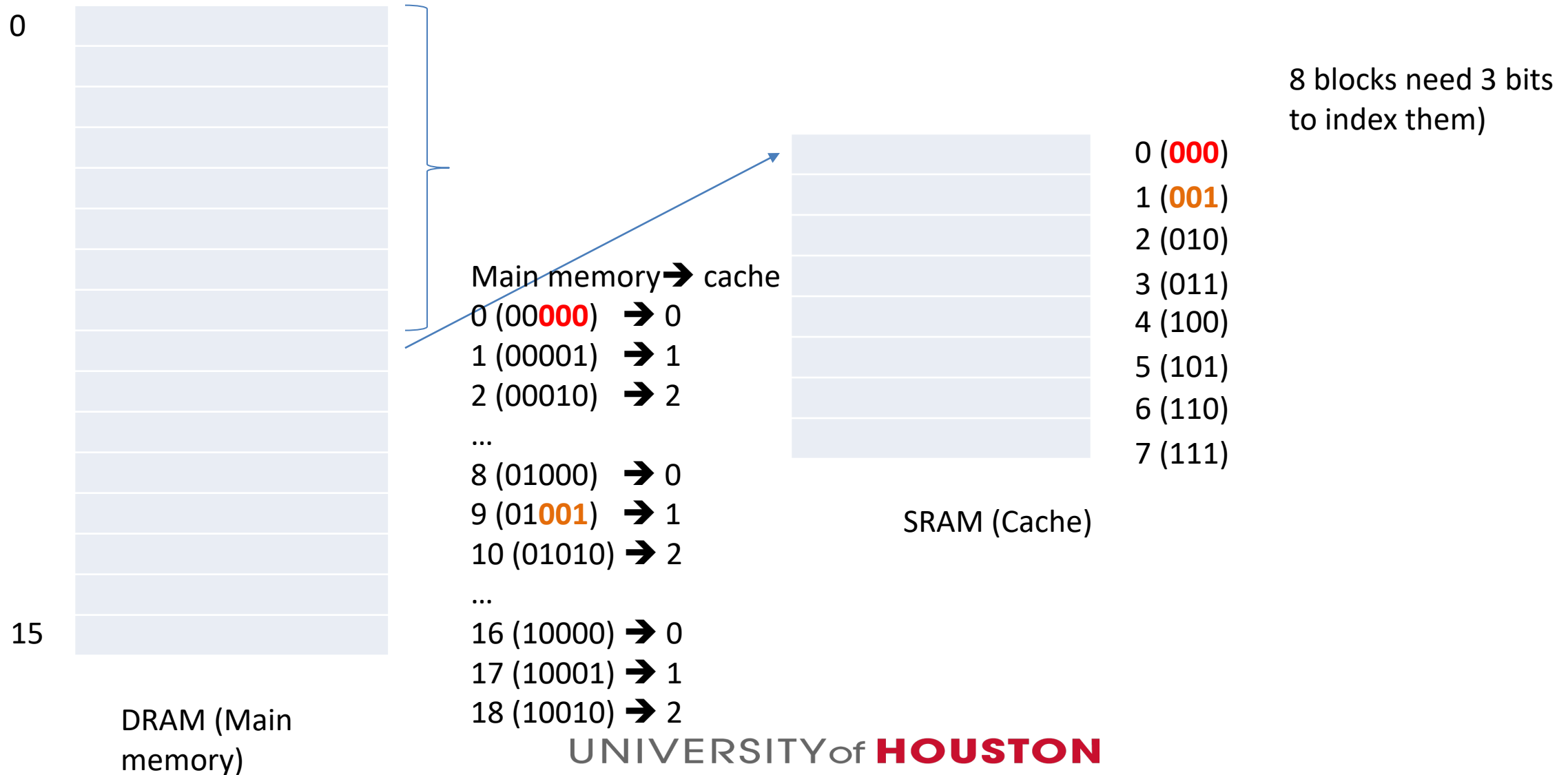
Retain the latest accessed ones and replace the oldest one

Direct Mapped Cache: Each address in main memory is mapped to exactly one block

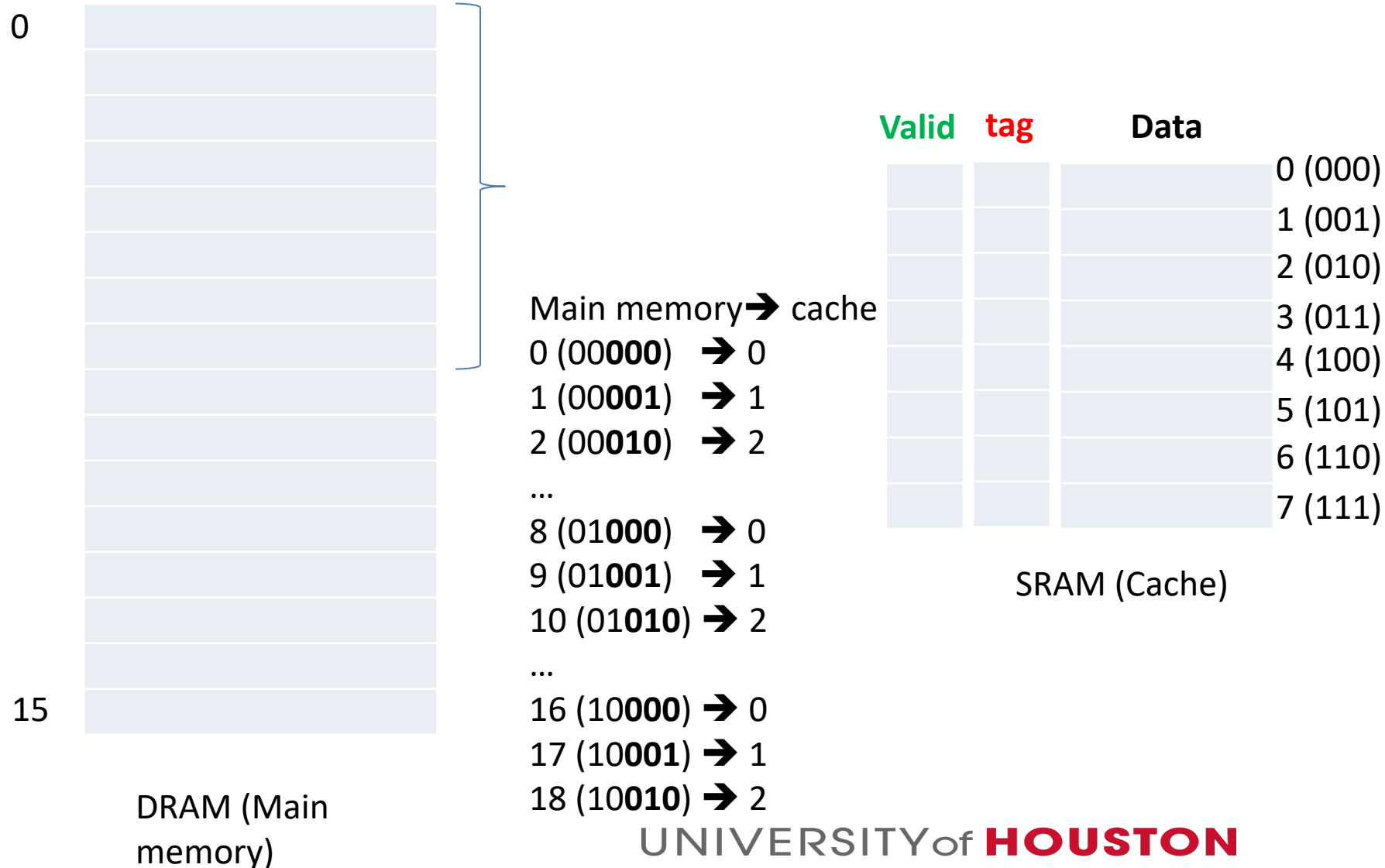
Example



Example



Example



Fetch 19:

19 in binary 10011

Block 3

19 modulo 8 = 3

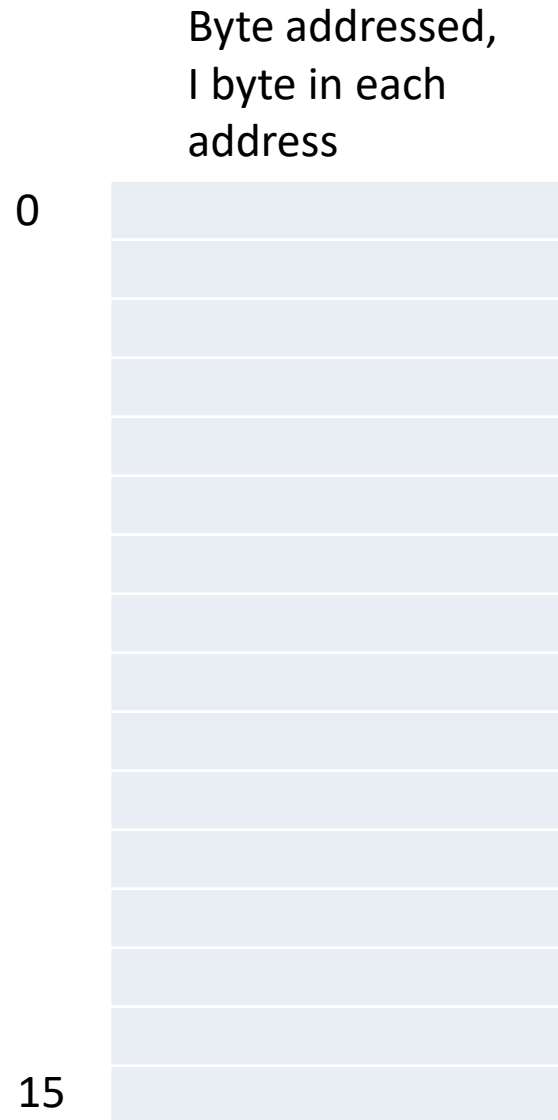
But the data in 011 could have come from address 3, 11 or from 19

Make sure that the data in block is valid.

For example when processor starts, the cache does not have good values

Additional 1 bit (**valid bit**) to indicate validity for each block.

SRAM Data

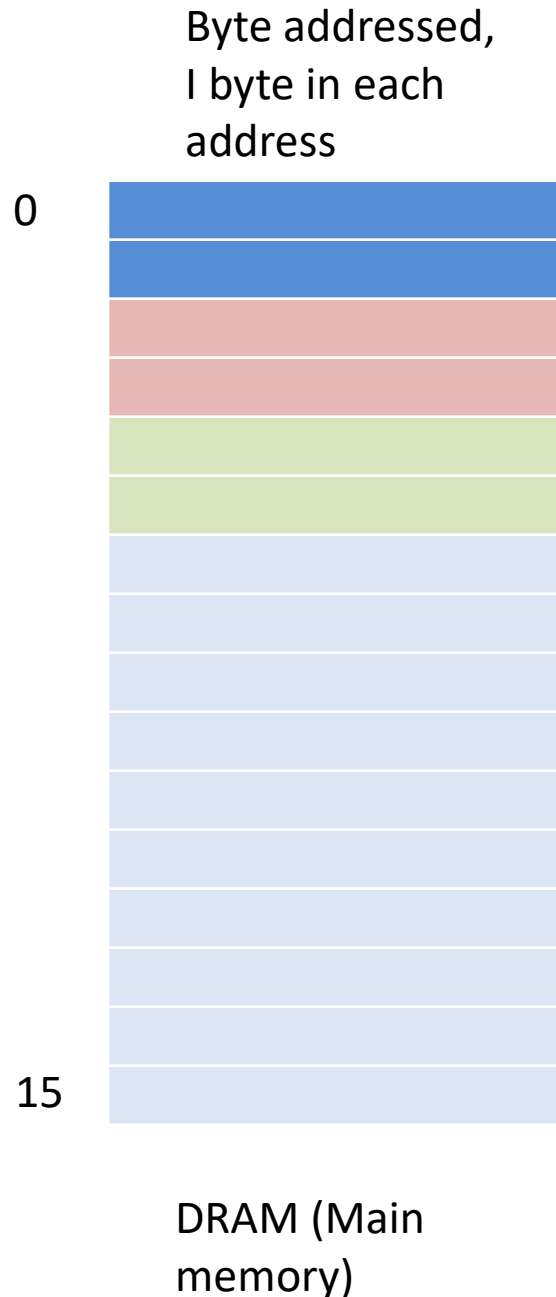


DRAM (Main
memory)

		0 (000)
		1 (001)
		2 (010)
		3 (011)
		4 (100)
		5 (101)
		6 (110)
		7 (111)

SRAM (Cache)

SRAM Data



Number of blocks (2^m) = 8
Mapped to the index matching
 $\log_2(8) = 3$

Fetch

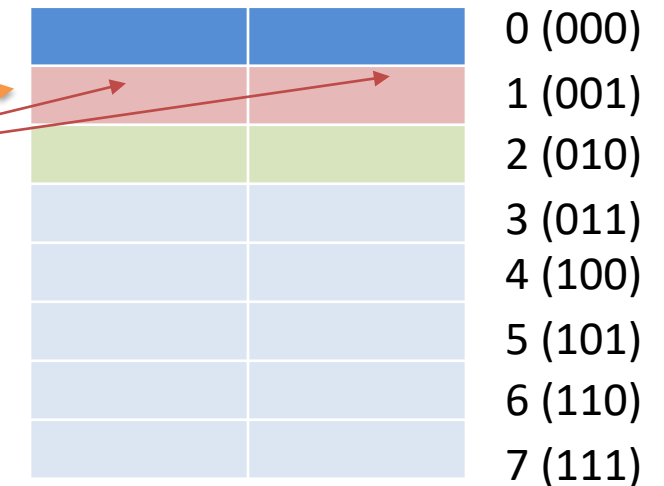
0 (0000**0**) → miss (load)

1 (0000**1**) → hit

2 (000**1**0) → miss (load)

3 (000**1**1) → hit

4 (00**1**00) → miss (load)



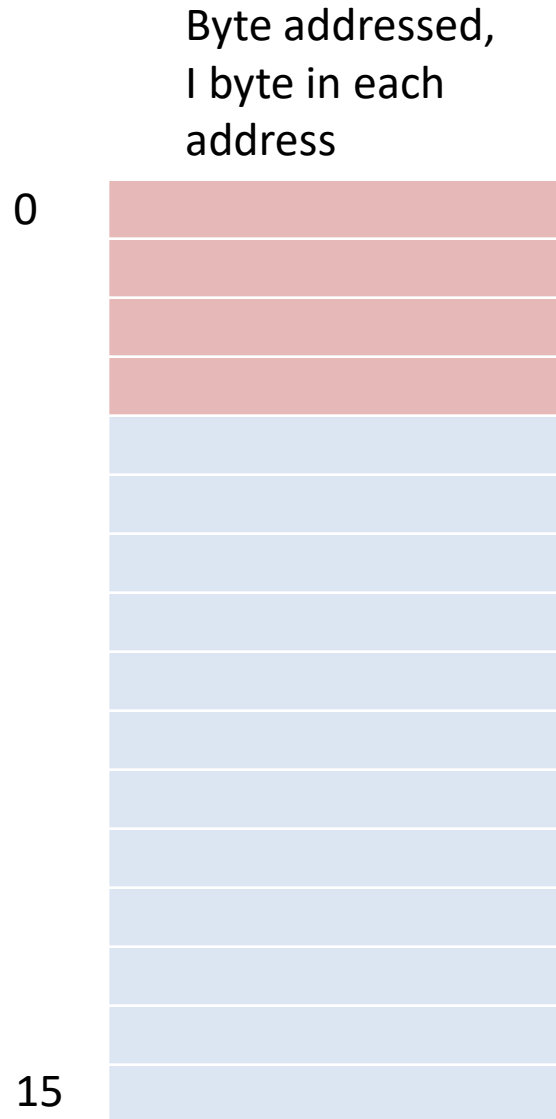
SRAM (Cache)

Memory Alignment:

Make sure that each block in a cache
stores 2^n bytes (power of 2).

Last n bits used as offset to locate bytes
in block

SRAM Data



DRAM (Main
memory)

Number of blocks (2^m) = 8
Mapped to the index matching
 $\log_2(8) = 3$

Fetch

0 (**000**00) → miss (load)

1 (**000**01) → hit

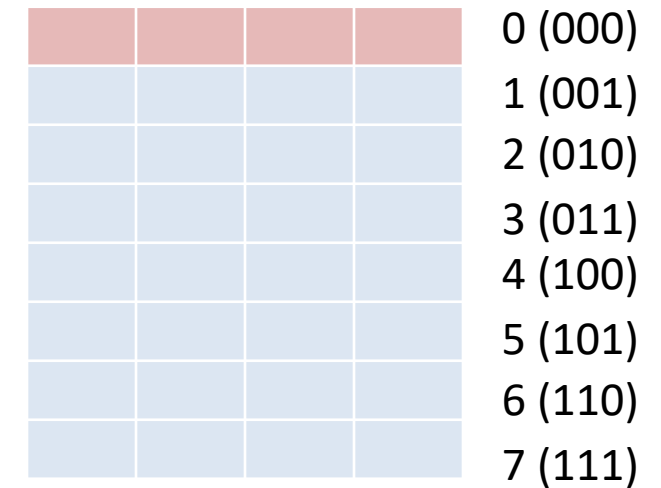
2 (**000**10) → hit

3 (**000**11) → hit

4 (00100) →

n=1 → n=2
More hits

Is larger block size better?



SRAM (Cache)

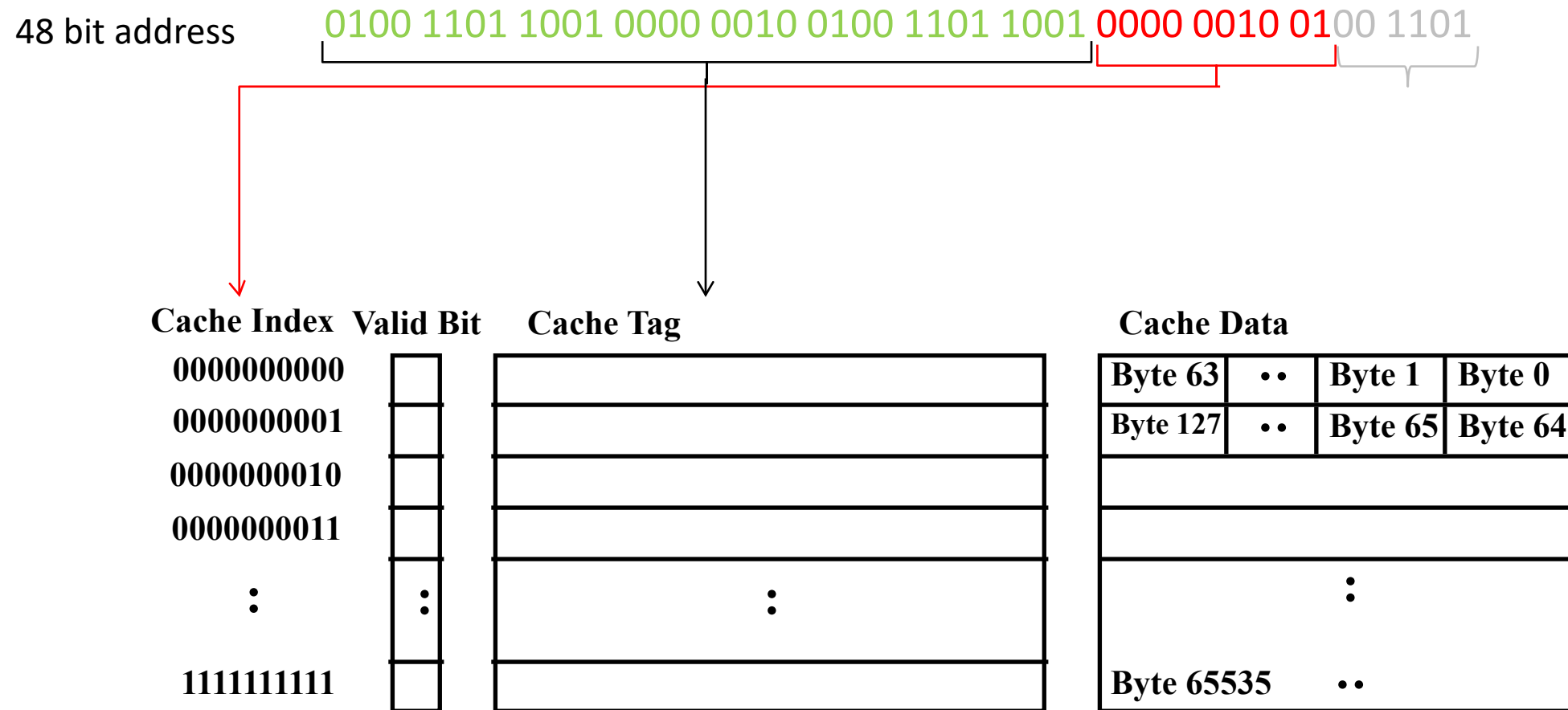
Block Size Considerations

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

A more realistic example

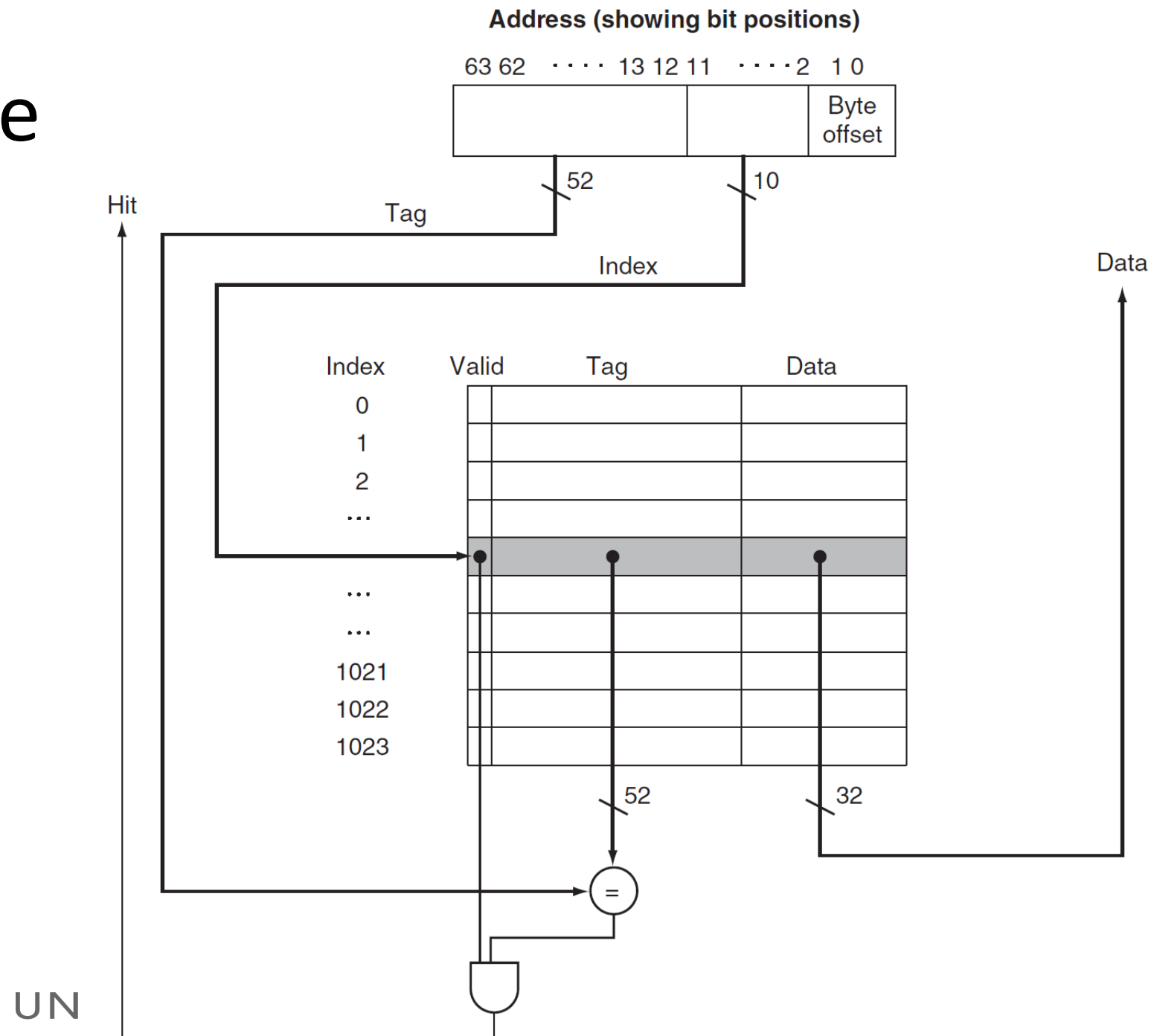
- 48 bit addresses
- 64 KB ($64 * 1024$ Bytes) of direct access cache
- Cache block size of 64 Bytes

A more realistic example

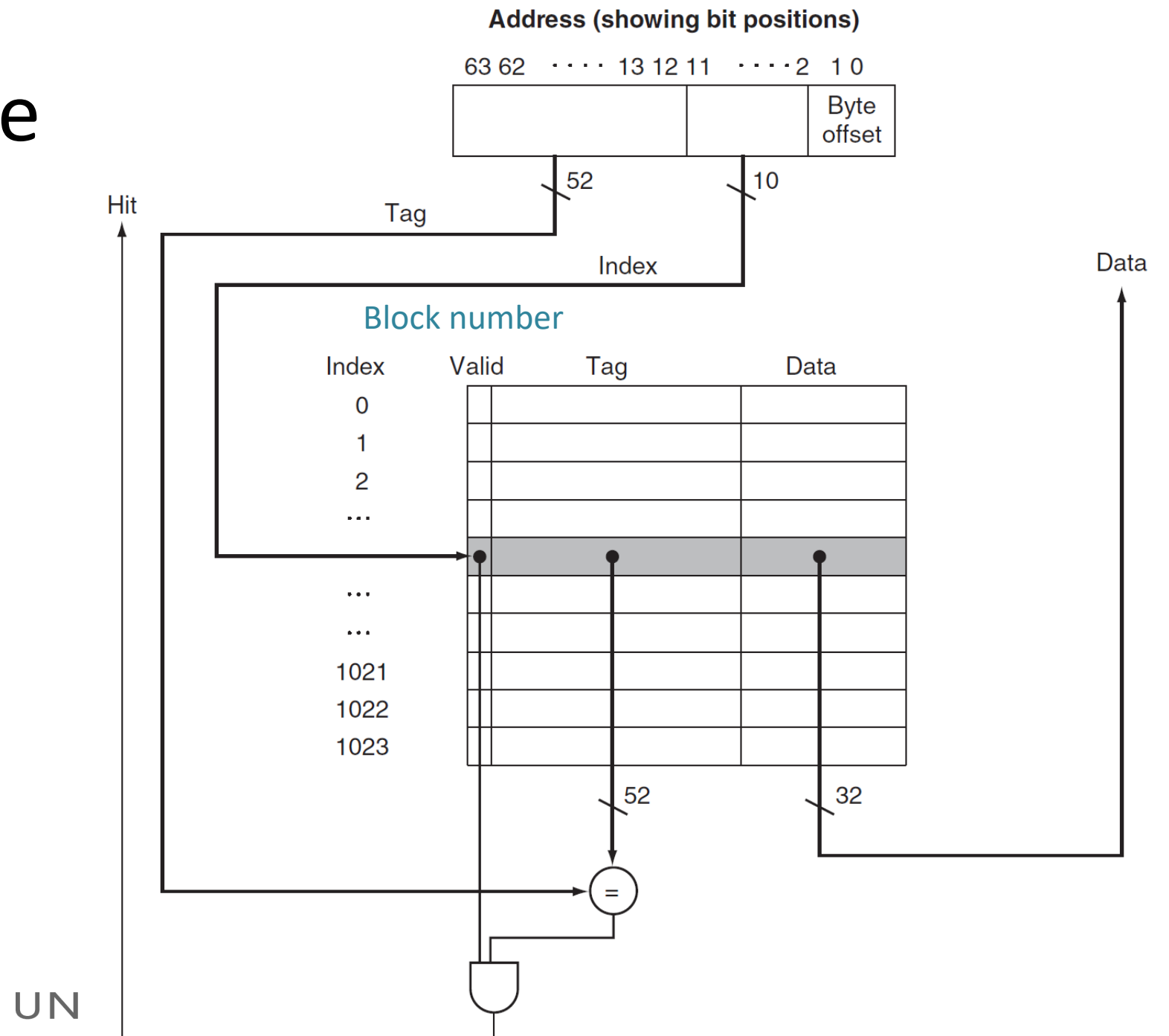


- Handling Cache Hits
- Handling Cache Misses
- Handling Cache Writes

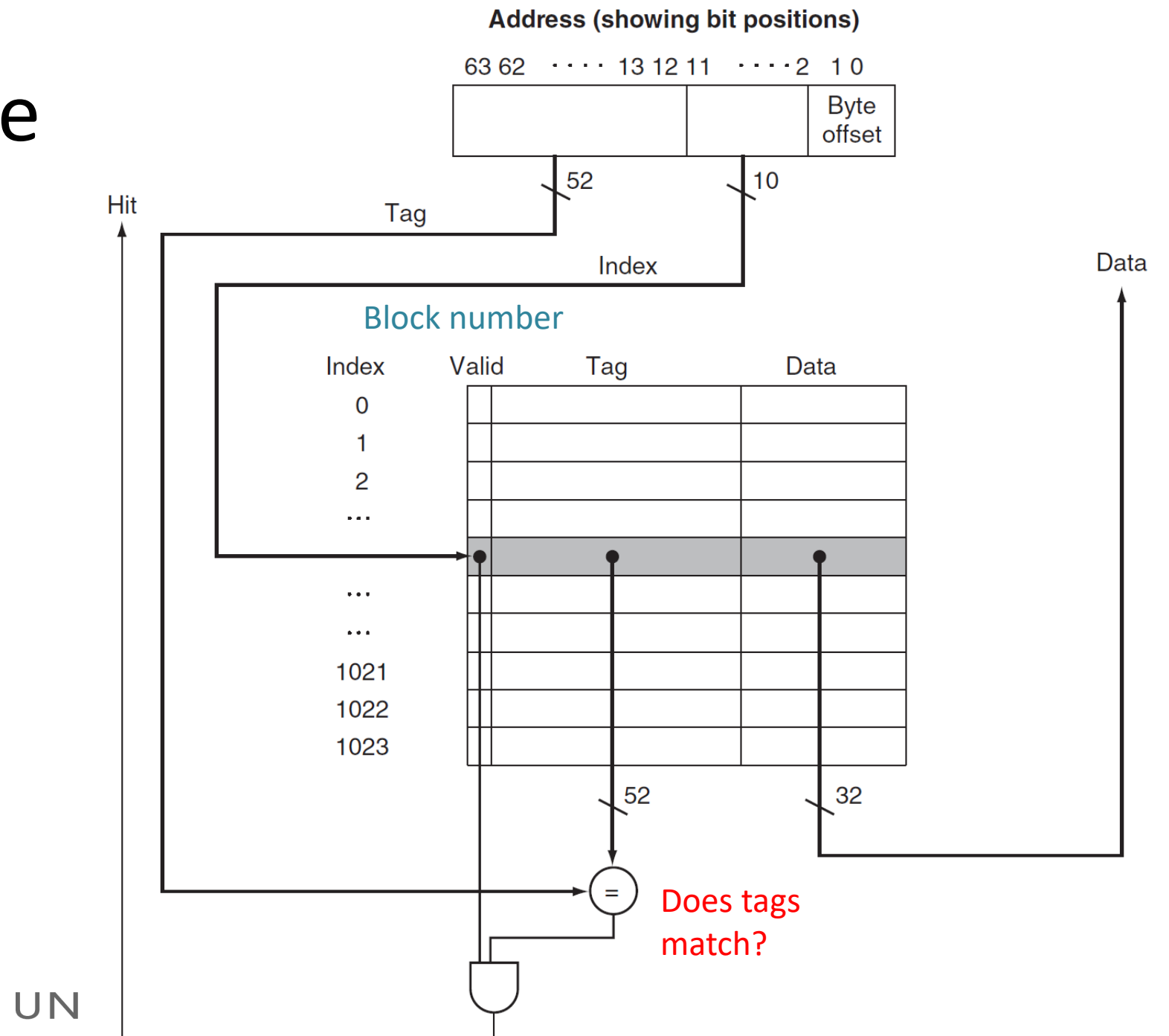
Cache Hardware



Cache Hardware

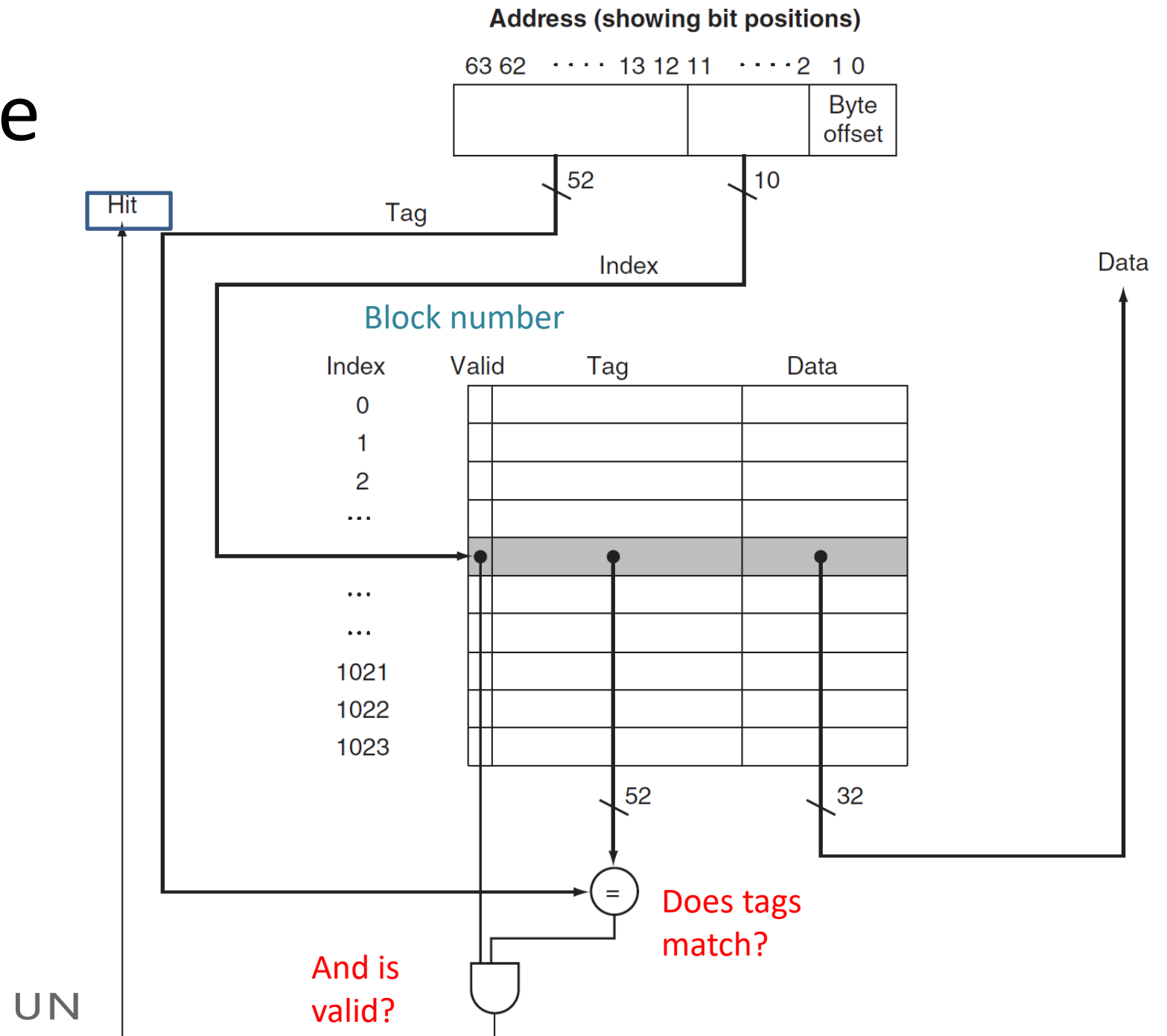


Cache Hardware



Cache Hardware

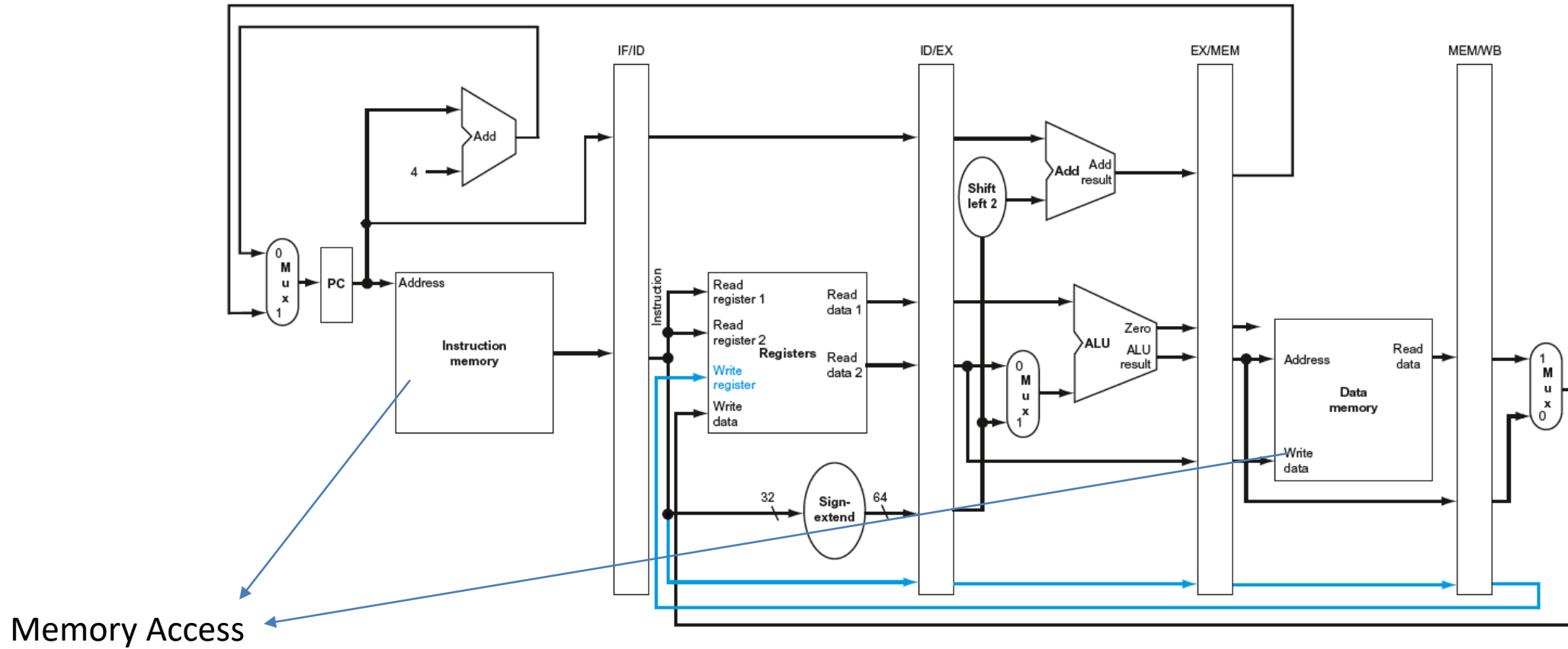
Easy to handle cache hits.
CPU proceeds normally



Handling Cache Misses

- On cache miss
 - Stall the CPU pipeline
 - Freeze the contents of register, wait for data and then proceed
 - Fetch block from next level of hierarchy

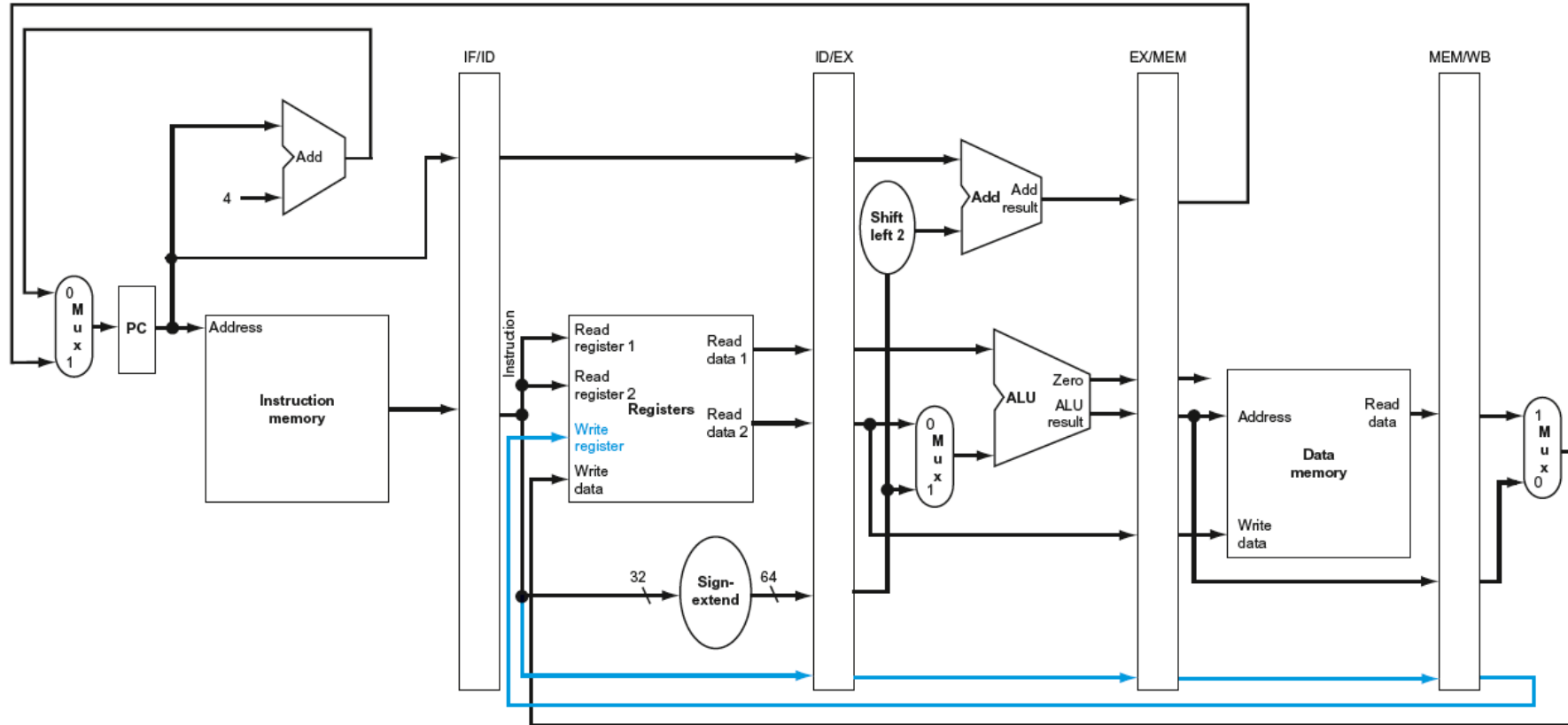
Datapath



Handling Cache Misses

- On cache miss
 - Stall the CPU pipeline
 - Fetch block from next level of hierarchy
- Cache miss can occur in instructions and data access
 - Instruction cache miss
 - Restart instruction fetch
 - Data cache miss
 - Complete data access

Handling Instruction Misses



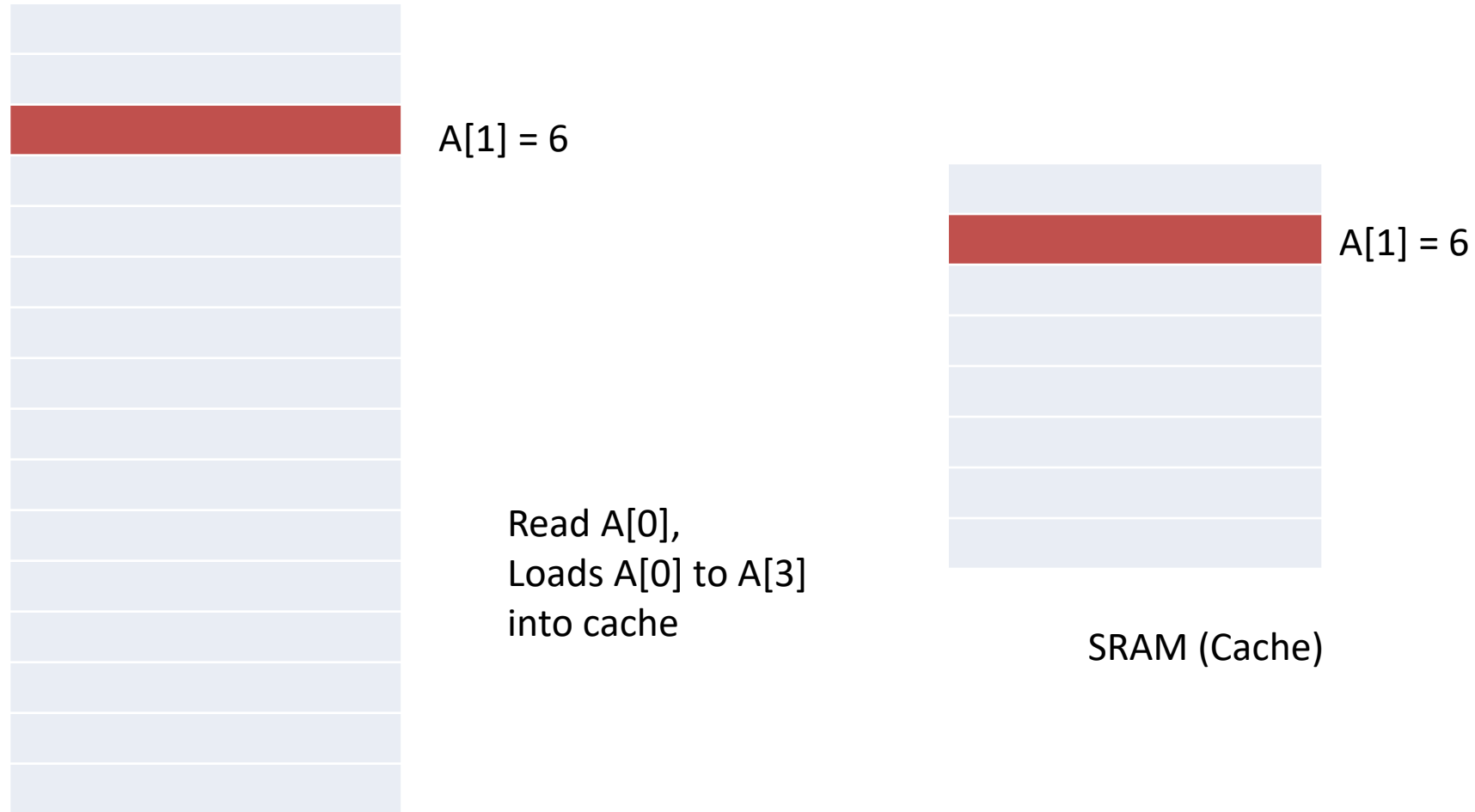
- Stage 1:
 - PC incremented to PC+4

Handling Instruction Misses

- Address of the instruction resulting in cache miss is $PC - 4$.
- Steps in handling cache miss
 1. Send the original PC value to the memory
 2. Instruct main memory to perform a read and wait for the memory
 3. Write the cache entry (valid bit, tag, and data)
 4. Restart the instruction execution at the first step

Handling Writes

Write Example

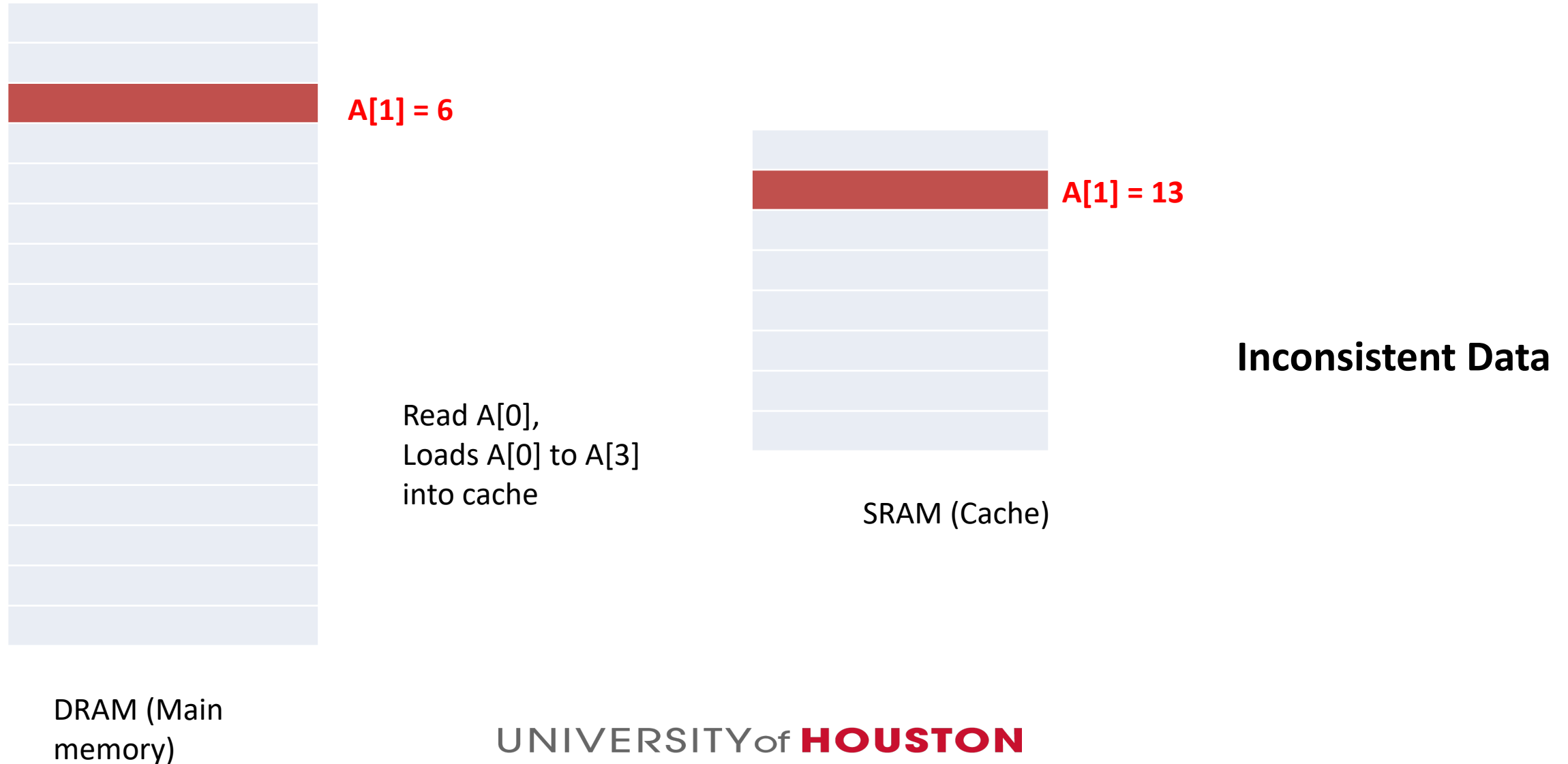


DRAM (Main
memory)

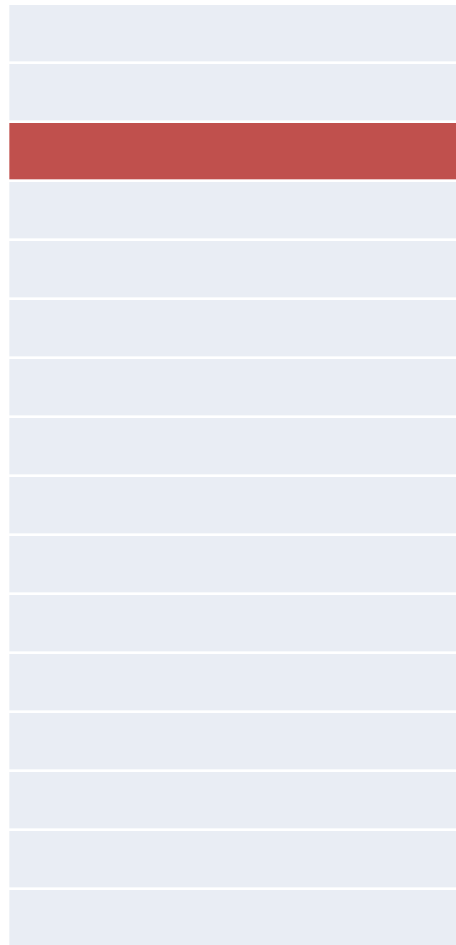
Handling Writes

- Lets Say $X0 = 13$, and the base address for A is in register X1
- Let's consider a store instruction
STUR X0, [X1, #8]

Write Example

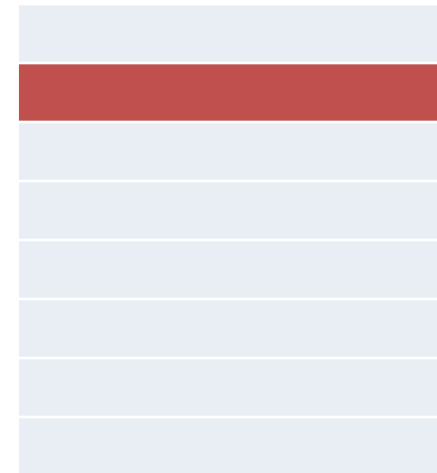


Write Example



A[1] = 13

Read A[0],
Loads A[0] to A[3]
into cache



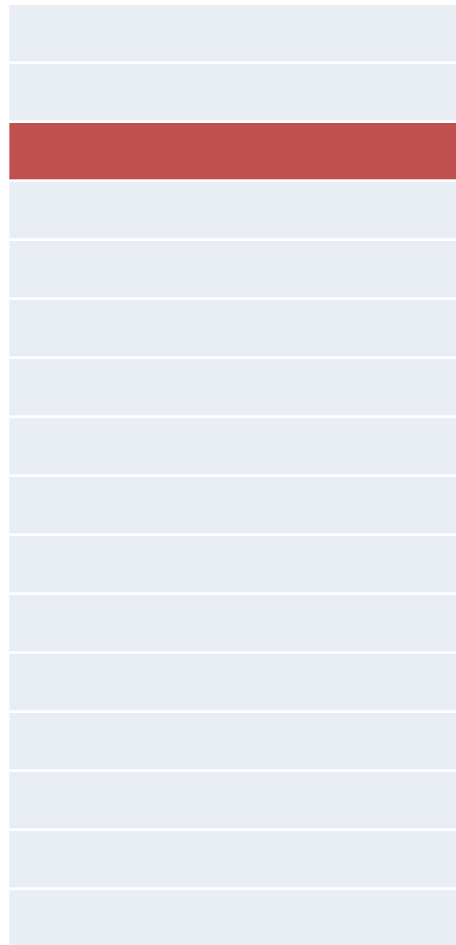
A[1] = 13

SRAM (Cache)

Inconsistent Data
Solution 1: Write to both Cache
and Memory
Write-Through strategy

DRAM (Main
memory)

Write Example



A[1] = 13

Read A[0],
Loads A[0] to A[3]
into cache



A[1] = 13

SRAM (Cache)

Inconsistent Data

Solution 1: Write to both Cache
and Memory

Write-Through strategy

Does not provide good
performance.

Every write is a write to memory
Each write can take up to 100
cycles

DRAM (Main
memory)

Example

- CPI of processor = 1
- 10% instructions are store
- Main Memory access takes 100 cycles
- What is the CPI if we use write-through strategy?

Example

- CPI of processor = 1
- 10% instructions are store
- Main Memory access takes 100 cycles
- What is the CPI if we use write-through strategy?

$$CPI = 1 + (10\% \text{ of } 100) = 11$$

Write Buffer

- Temporary memory to hold data that needs to be written.
- Holds data waiting to be written to memory.
- CPU continues immediately
 - Only stalls on write if write buffer is already full
- When write to main memory completes, the buffer is freed.

Write Buffer

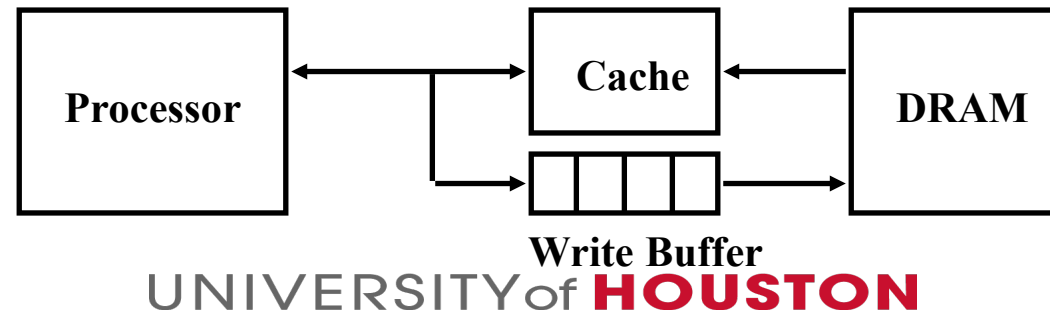
- Temporary memory to hold data that needs to be written.
- Holds data waiting to be written to memory.
- CPU continues immediately
 - Only stalls on write if write buffer is already full
- When write to main memory completes, the buffer is freed.
- Write buffer is full
 - Rate of Memory write < rate of writes generated

Write Buffer

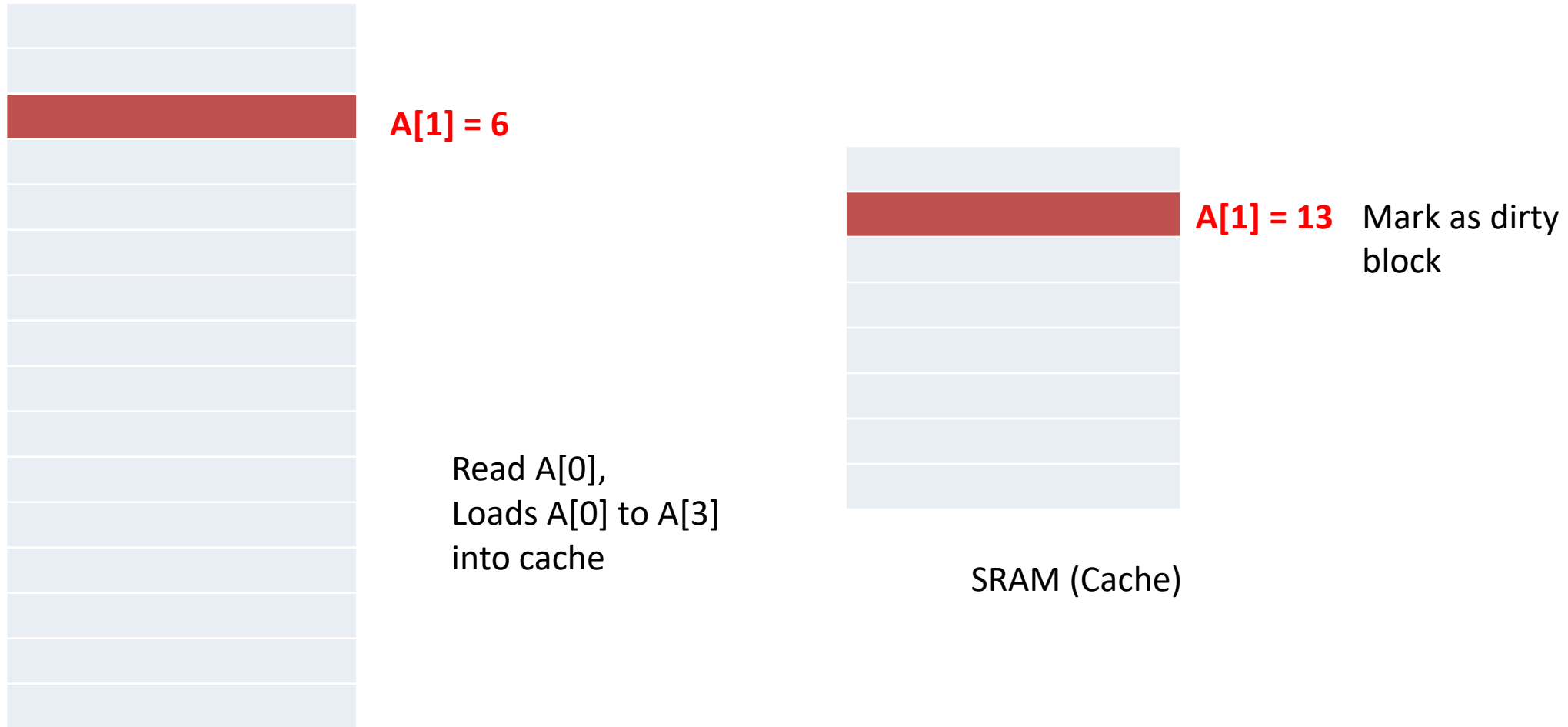
- Temporary memory to hold data that needs to be written.
- Holds data waiting to be written to memory.
- CPU continues immediately
 - Only stalls on write if write buffer is already full
- When write to main memory completes, the buffer is freed.
- Write buffer is full
 - Rate of Memory write < rate of write generated
- Use Deeper memory

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., write to memory takes 100 cycles
- Solution: write buffer
 - Holds data waiting to be written to memory
 - CPU continues immediately
 - Only stalls on write if write buffer is already full

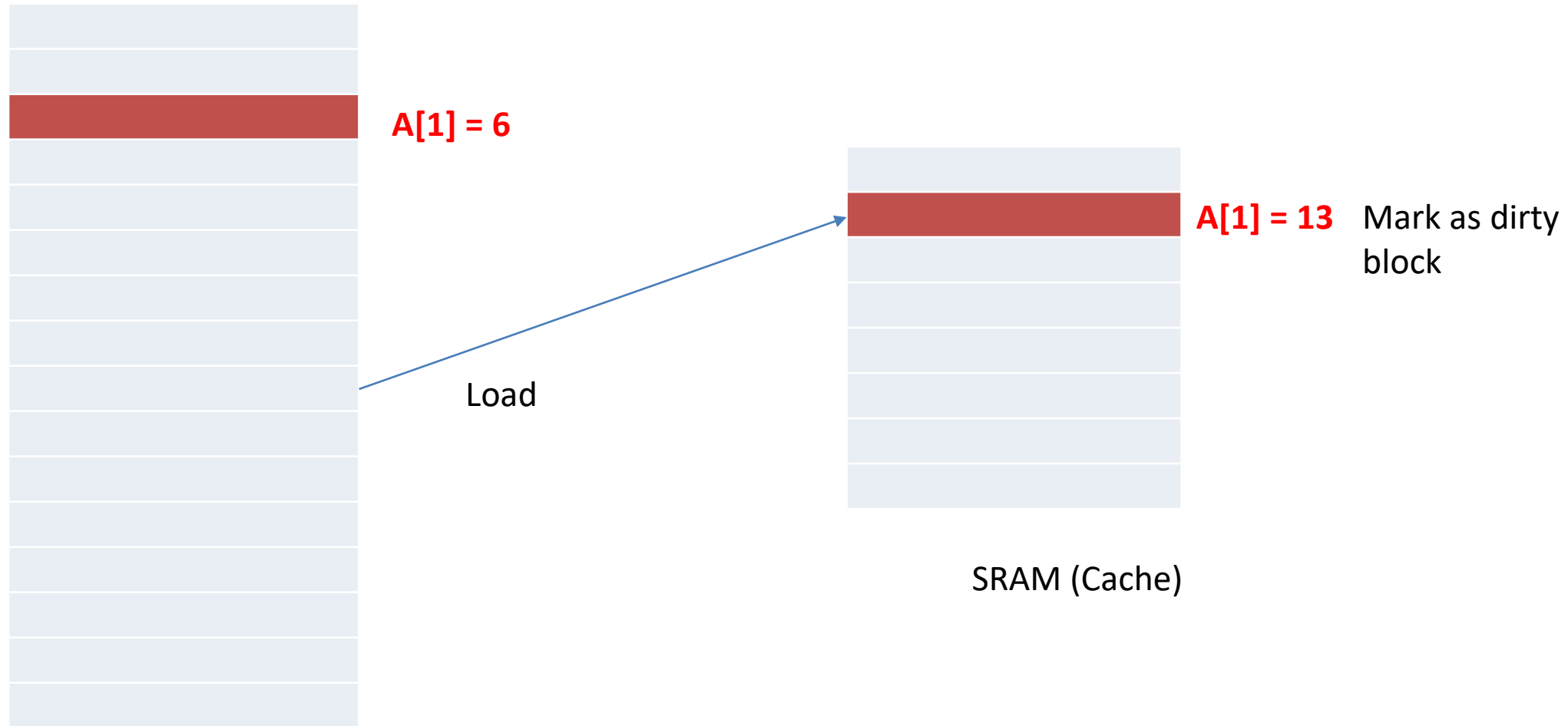


Another Write Strategy



DRAM (Main
memory)

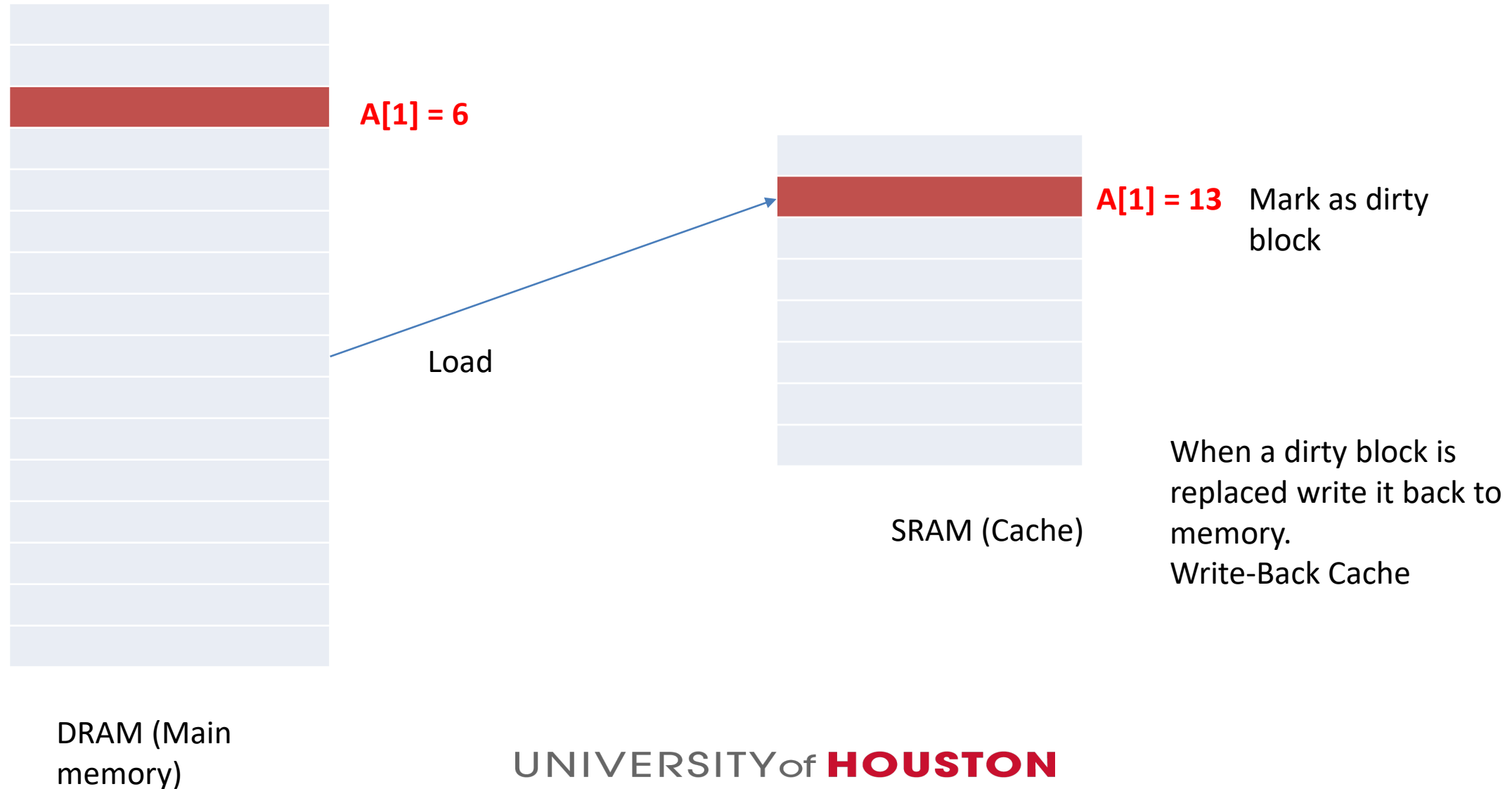
Another Write Strategy



DRAM (Main
memory)

SRAM (Cache)

Another Write Strategy



Write-Back

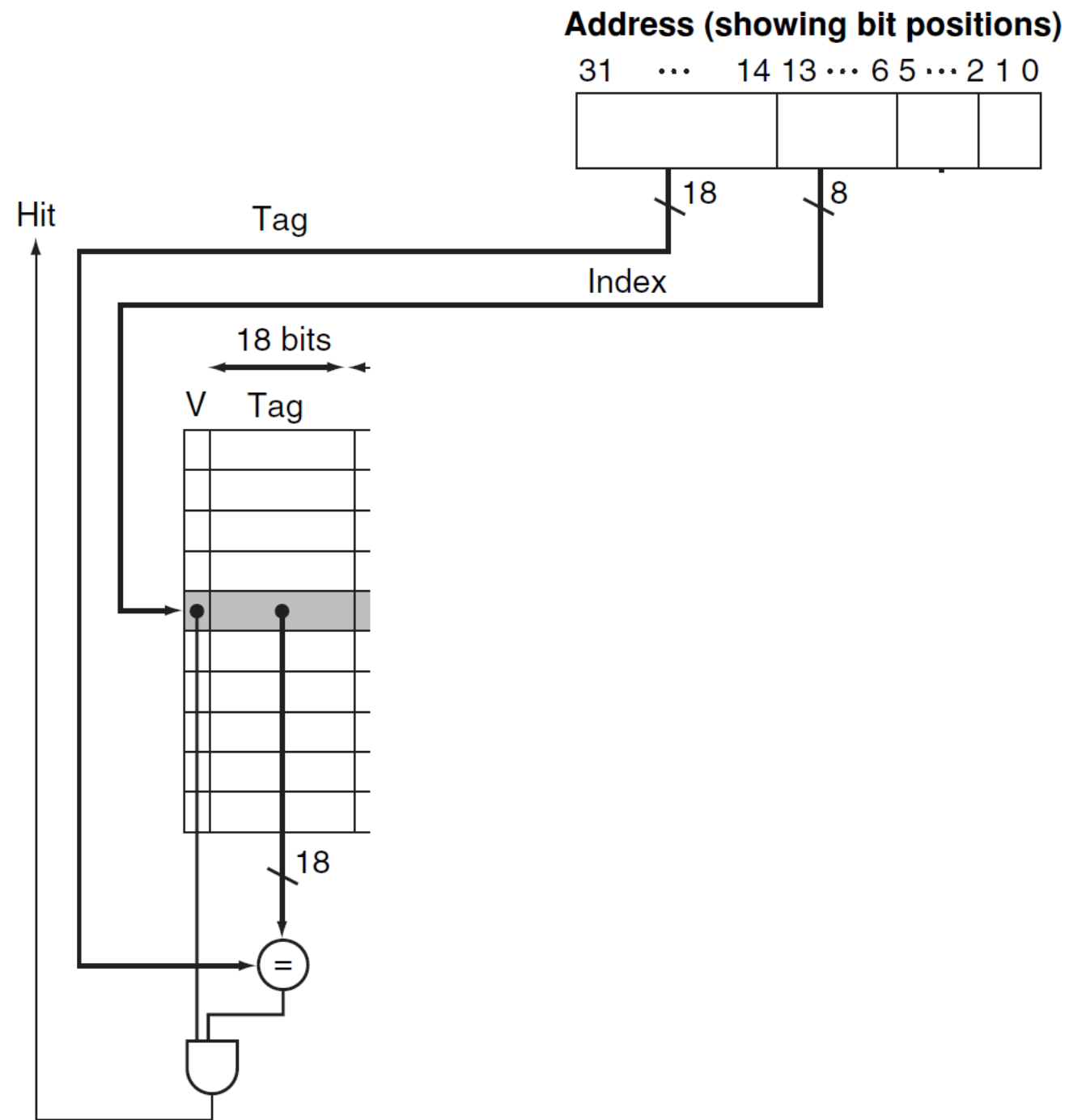
- Alternative: On data-write hit, just update the block in cache
 - Keep track of whether each block is dirty
- When a dirty block is replaced
 - Write it back to memory

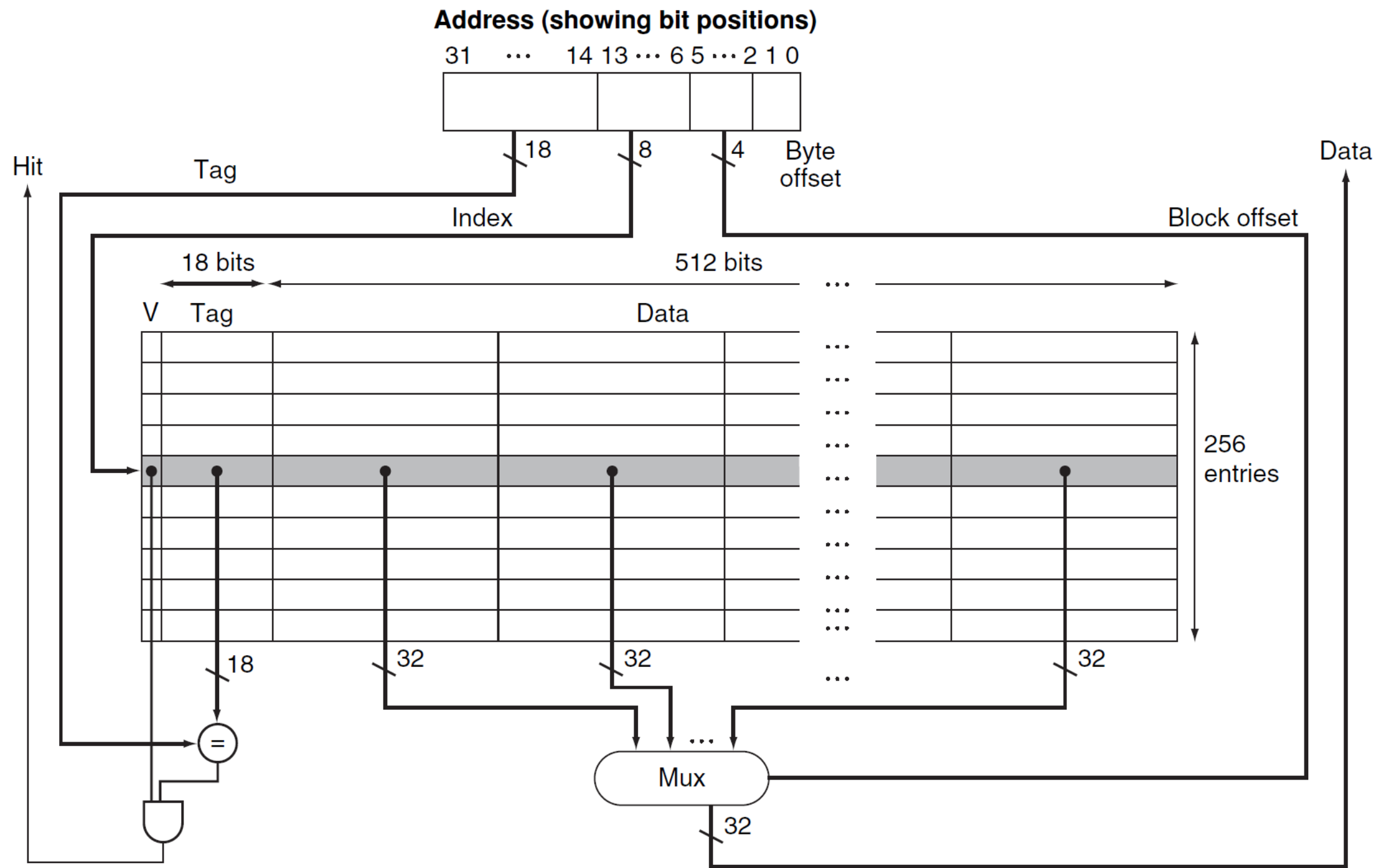
Handling Cache Writes

- Write through—The information is written to both the block in the cache and to the block in the lower-level 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.
- WT always combined with write buffers so that don't wait for lower level memory

Intrinsity FastMATH Processor

- Embedded microprocessor (MIPS architecture)
 - A simple cache implementation
 - 12 stage pipeline
- MIPS address size is 32-bits.
- Cache
 - 16 KiB ($16 * 1024\text{B}$ or 4096 Words)
 - 16 word per block





Cache Performance

- CPU time:
 - Time spent by CPU executing the program

Cache Performance

- CPU time:
 - Time spent by CPU executing the program
 - Time spent by CPU waiting (stalled) for memory

Cache Performance

- CPU time:
 - Time spent by CPU executing the program (include cache hits)
 - Time spent by CPU waiting (stalled) for memory (mainly cache misses)

Cache Performance

- CPU time:
 - Time spent by CPU executing the program (include cache hits)
 - Time spent by CPU waiting (stalled) for memory (mainly cache misses)

$$\text{CPU time} = \text{cycles} \times \text{cycle time}$$

Cache Performance

- CPU time:
 - Time spent by CPU executing the program (include cache hits)
 - Time spent by CPU waiting (stalled) for memory (mainly cache misses)

$$\text{CPU time} = \text{cycles} \times \text{cycle time}$$

$$\text{CPU time} = (\text{execution cycles} + \text{mem stall cycles}) \times \text{cycle time}$$

Cache Performance

- CPU time:
 - Time spent by CPU executing the program (include cache hits)
 - Time spent by CPU waiting (stalled) for memory (mainly cache misses)

$$\text{CPU time} = \text{cycles} \times \text{cycle time}$$

$$\text{CPU time} = (\text{execution cycles} + \text{mem stall cycles}) \times \text{cycle time}$$

- Memory stall cycles

- Either from read or write

$$\text{Mem stall cycles} = \text{Read stall cycles} + \text{write stall cycles}$$

- Miss penalty → cycles to fetch data from main memory into cache
- Miss rate → ratio of misses to total memory access (hits+misses)


$$\text{miss rate} = \frac{\text{no.of misses}}{\text{no.of mem access}}$$

Memory stall cycles= ?

Memory Stall Cycles

Memory stall cycles

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




Can be both read and write misses.

Memory Stall Cycles

Memory stall cycles

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

$$= \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Misses}}{\text{Instruction}} \times \text{Miss penalty}$$



Can be both read and write misses.

Cache Performance Example

- Given
 - I-cache miss rate = 2%
 - D-cache miss rate = 4%
 - Miss penalty = 100 cycles
 - Base CPI (ideal cache) = 2
 - Load & stores are 36% of instructions
- Miss cycles per instruction

Cache Performance Example

- Given
 - I-cache miss rate = 2%
 - D-cache miss rate = 4%
 - Miss penalty = 100 cycles
 - Base CPI (ideal cache) = 2
 - Load & stores are 36% of instructions
- Miss cycles per instruction
 - I-cache: $0.02 \times 100 = 2$
 - D-cache: $0.36 \times 0.04 \times 100 = 1.44$

Cache Performance Example

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

Average Access Time

- Hit time is also important for performance
- Average memory access time (AMAT)
 - $AMAT = \text{Hit time} + \text{Miss rate} \times \text{Miss penalty}$

Average Access Time

- Hit time is also important for performance
- Average memory access time (AMAT)
 - $\text{AMAT} = \text{Hit time} + \text{Miss rate} \times \text{Miss penalty}$
- Example
 - CPU with 1ns clock, hit time = 1 cycle, miss penalty = 20 cycles, l-cache miss rate = 5%

Average Access Time

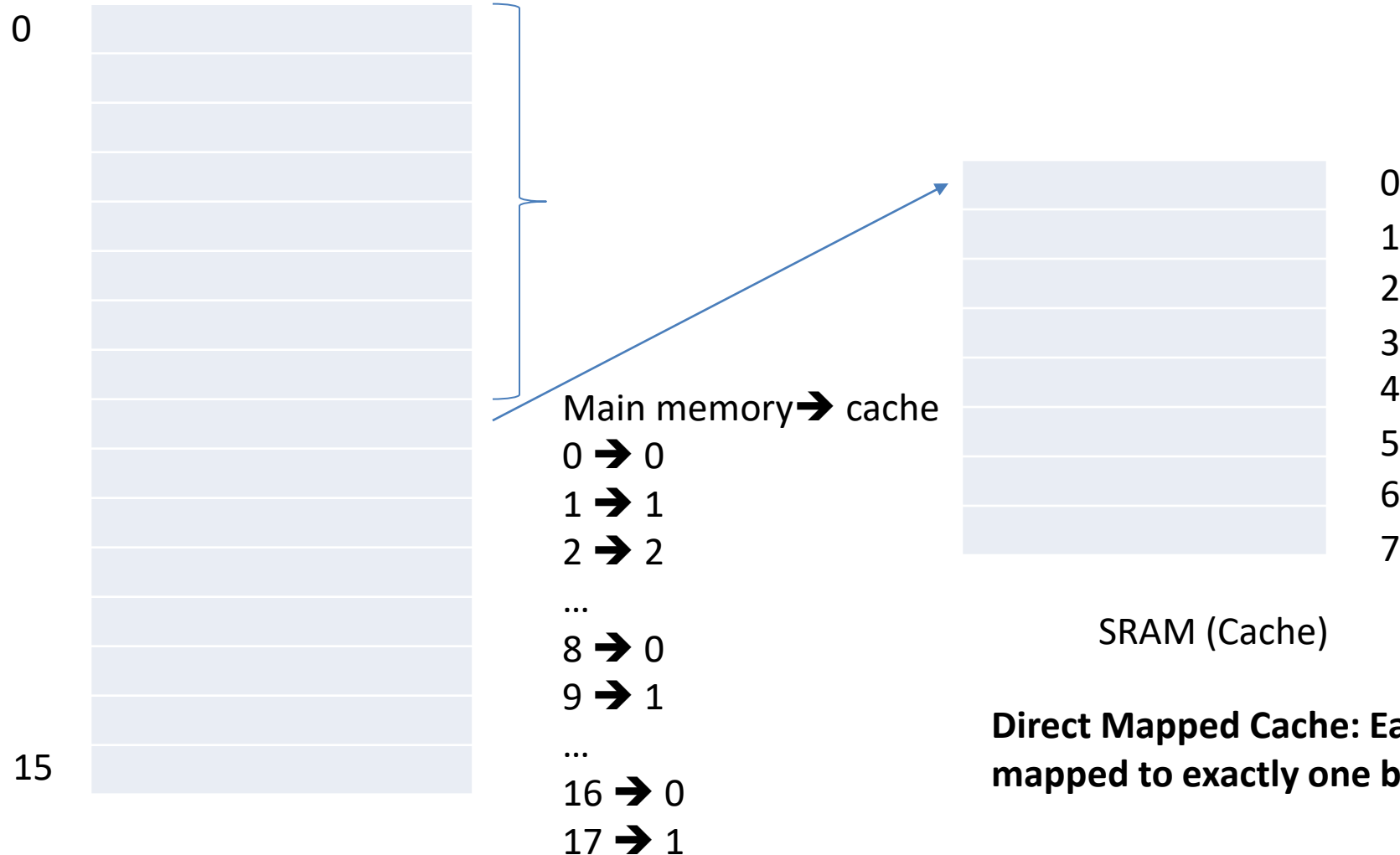
- Hit time is also important for performance
- Average memory access time (AMAT)
 - $\text{AMAT} = \text{Hit time} + \text{Miss rate} \times \text{Miss penalty}$
- Example
 - CPU with 1ns clock, hit time = 1 cycle, miss penalty = 20 cycles, l-cache miss rate = 5%
 - $\text{AMAT} = 1 + 0.05 \times 20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

Reducing Cache Misses

Types of cache misses

- Compulsory Misses: first access to a block cannot be in the cache (cold start misses)
- Capacity Misses: cache cannot contain all blocks required for the execution
- Conflict Misses: cache block has to be discarded because of block replacement

Example



Fetch 9:

Not available in cache, so fetch from main memory

Temporal locality:

Retain the latest accessed ones and replace the oldest one

Direct Mapped Cache: Each address in main memory is mapped to exactly one block

Associative Caches

- Direct Mapped Cache: Each address in main memory is mapped to exactly one block
- Each address in main memory can be mapped to
 - a set of cache blocks (Set associative)
 - any block (Full Associative)

Full Associative

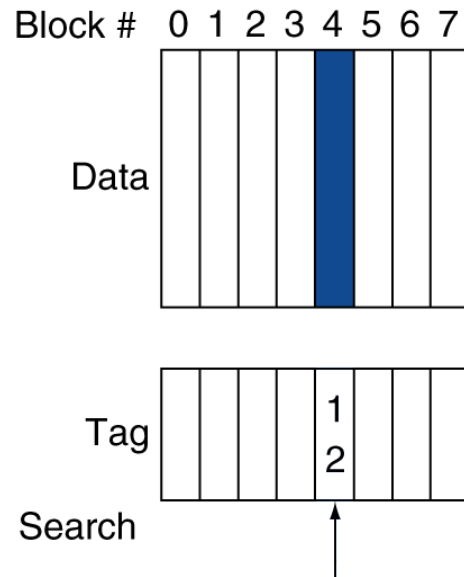
- Fully associative
 - Allow a given block to go in any cache entry

Full Associative

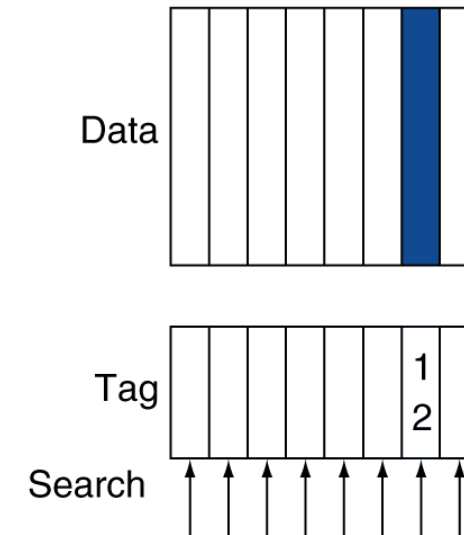
- Fully associative
 - Allow a given block to go in any cache entry
 - Requires all entries to be searched at once
 - Sequential search too slow
 - Parallely search all blocks (can be expensive)

Associative Cache Example

Direct mapped



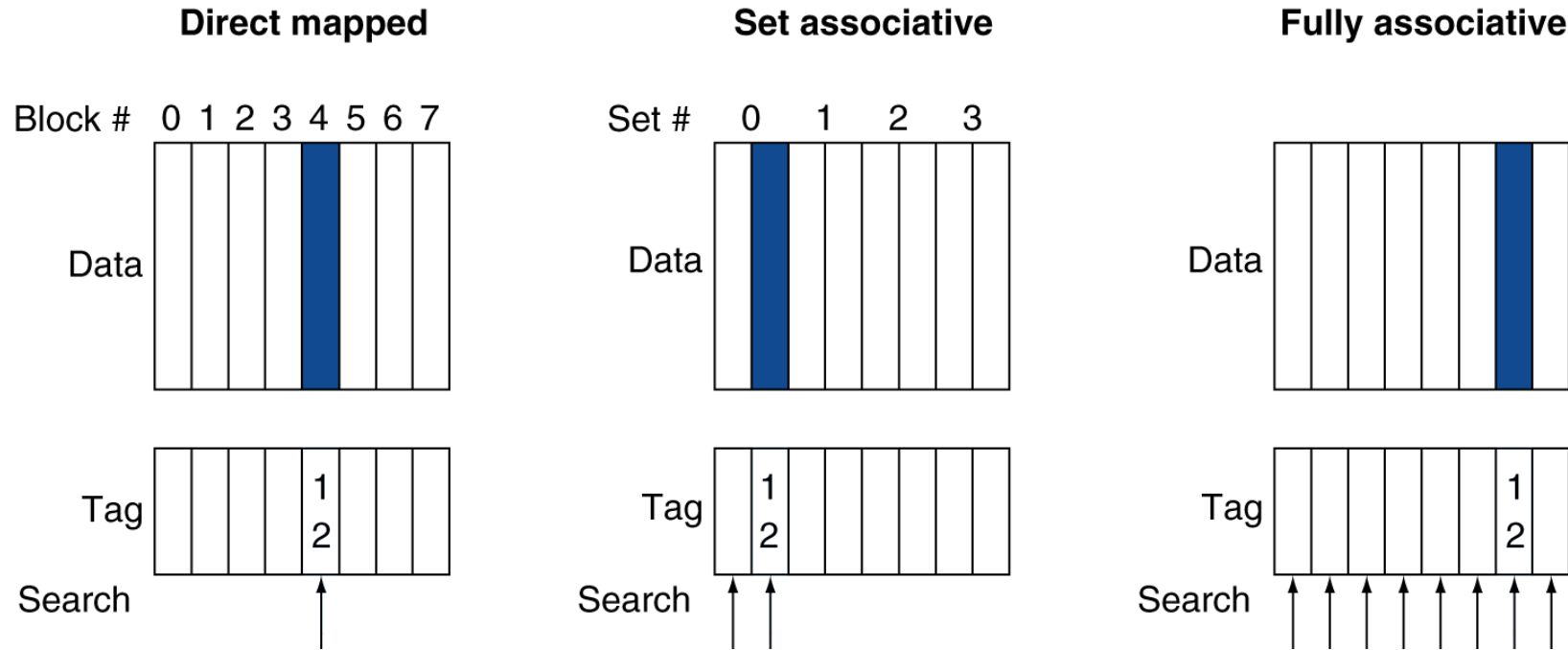
Fully associative



n -way set associative

- Each set contains n entries
- Address determines which set
 - (Address) modulo (#Sets in cache)
- Search all entries in a given set at once
- n comparators (less expensive)

Associative Cache Example



Associativity Example

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

Associativity Example

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

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

Associativity Example

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

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

Associativity Example

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

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

Associativity Example

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

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

Associativity Example

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

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

Associativity Example

- 2-way set associative

Block address	Cache index	Hit/miss	Cache content after access			
			Set 0		Set 1	
0						
8						
0						
6						
8						

Associativity Example

- 2-way set associative

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

Associativity Example

- 2-way set associative

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

Associativity Example

- 2-way set associative

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

Associativity Example

- 2-way set associative

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

Associativity Example

- 2-way set associative

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

■ Fully associative

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

Associativity Example

- 2-way set associative

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

■ Fully associative

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

Associativity Example

- 2-way set associative

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

■ Fully associative

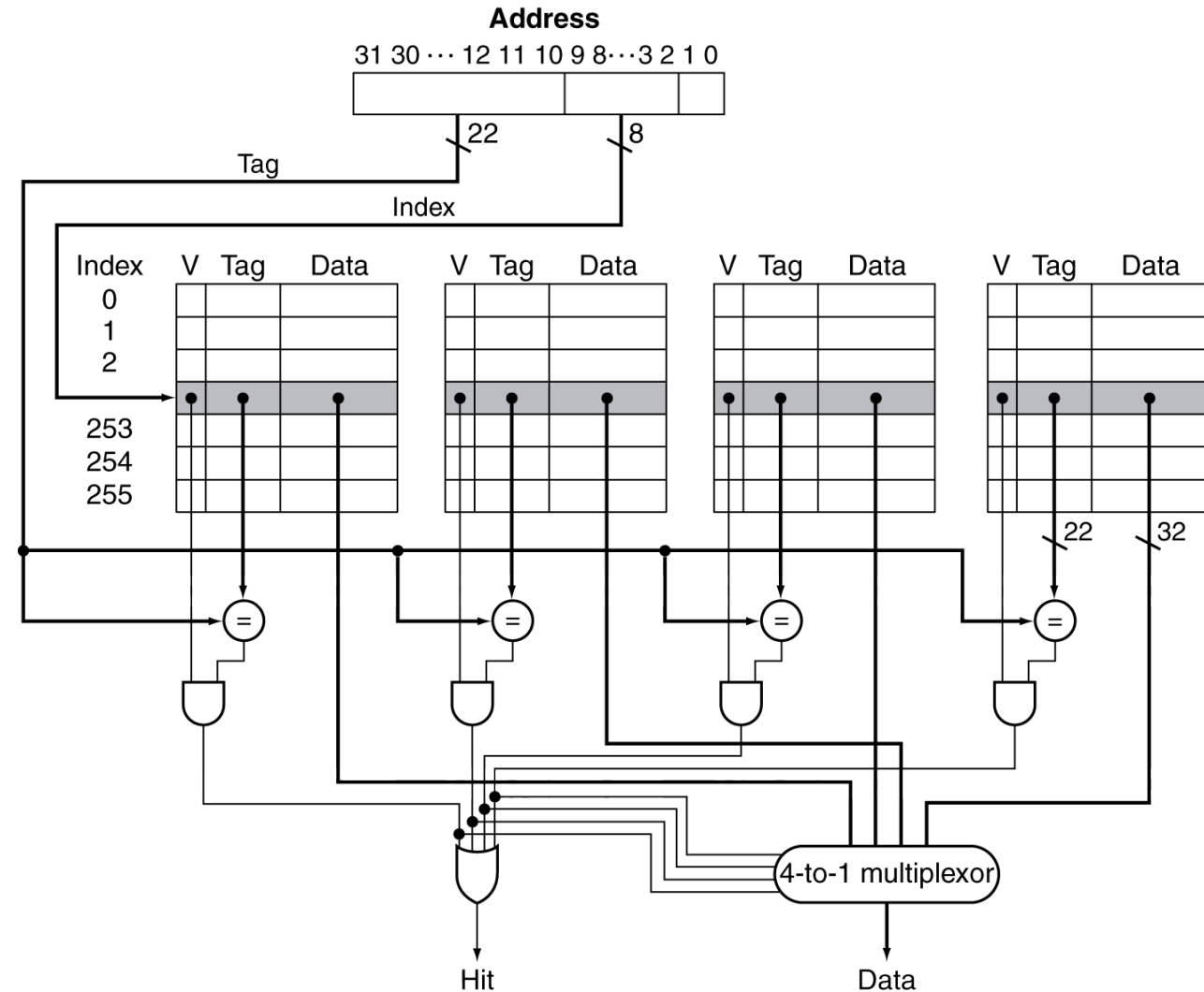
Block address		Hit/miss	Cache content after access			
0		miss	Mem[0]			
8		miss	Mem[0]	Mem[8]		
0		hit	Mem[0]	Mem[8]		
6		miss	Mem[0]	Mem[8]	Mem[6]	
8		hit	Mem[0]	Mem[8]	Mem[6]	

How Much Associativity

- Increased associativity decreases miss rate
 - But with diminishing returns
- Simulation of a system with 64KB D-cache, 16-word blocks, SPEC2000
 - 1-way: 10.3%
 - 2-way: 8.6%
 - 4-way: 8.3%
 - 8-way: 8.1%

Set Associative Cache Organization

Locating a block



Replacement Policy

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

4 Questions for Memory Hierarchy

- Q1: Where can a block be placed in the upper level?
(Block placement)
 - *Direct Mapped vs. Set-Associative vs. Fully Associative Caches*
- Q2: How is a block found if it is in the upper level?
(Block identification)
 - *cache index, cache tag, cache offset*
- Q3: Which block should be replaced on a miss?
(Block replacement)
 - *LRU vs. Random replacement*
- Q4: What happens on a write?
(Write strategy)

Multilevel Caches

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

Multilevel Cache Example

- Given
 - CPU base CPI = 1, clock rate = 4GHz
 - \Rightarrow clock cycle time = 0.25ns
 - Miss rate/instruction = 2%
 - Main memory access time = 100ns
- With just primary cache
 - Miss penalty = ? cycles

Multilevel Cache Example

- Given
 - CPU base CPI = 1, clock rate = 4GHz
 - \Rightarrow clock cycle time = 0.25ns
 - Miss rate/instruction = 2%
 - Main memory access time = 100ns
- With just primary cache
 - Miss penalty = $100\text{ns}/0.25\text{ns} = 400$ cycles
 - Effective CPI

Multilevel Cache Example

- Given
 - CPU base CPI = 1, clock rate = 4GHz
 - \Rightarrow clock cycle time = 0.25ns
 - Miss rate/instruction = 2%
 - Main memory access time = 100ns
- With just primary cache
 - Miss penalty = $100\text{ns} / 0.25\text{ns} = 400$ cycles
 - Effective CPI = $1 + 0.02 \times 400 = 9$

Example (cont.)

- Now add L-2 cache
 - Access time = 5ns
 - Global miss rate to main memory = 0.5%
- L1 miss with L-2 hit
 - Penalty = ? cycles

Example (cont.)

- Now add L-2 cache
 - Access time = 5ns
 - Global miss rate to main memory = 0.5%
- L1 miss with L-2 hit
 - Penalty = $5\text{ns}/0.25\text{ns} = 20$ cycles
- L1 miss with L-2 miss
 - Extra penalty = 400 cycles
- $\text{CPI} = 1 + 0.02 \times 20 + 0.005 \times 400 = 3.4$
- Performance ratio = $9/3.4 = 2.6$

Multilevel Cache Considerations

- Primary cache
 - Focus on minimal hit time
- L-2 cache
 - Focus on low miss rate to avoid main memory access
 - Hit time has less overall impact

Remarks

- Fast memories are small, large memories are slow
 - We really want fast, large memories
 - Caching gives this illusion
- Principle of locality
 - Programs use a small part of their memory space frequently
- Memory hierarchy
 - L1 cache \leftrightarrow L2 cache \leftrightarrow L3 cache \leftrightarrow DRAM memory \leftrightarrow disk
- Memory system design is critical for multiprocessors