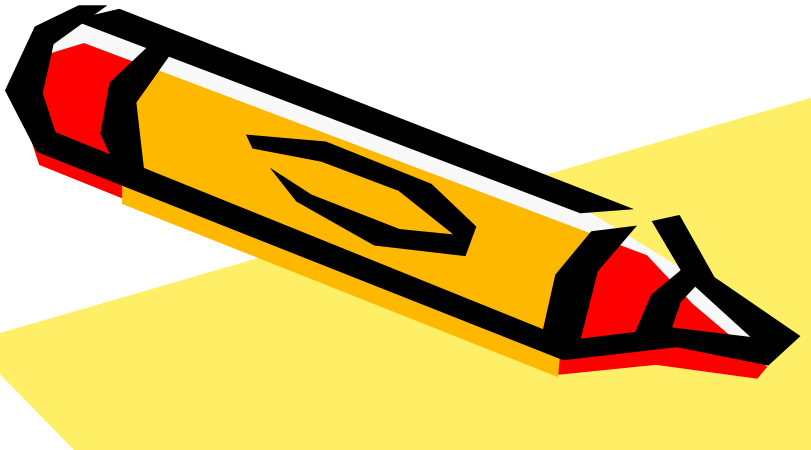


Computer Architecture

Hossein Asadi
Department of Computer Engineering
Sharif University of Technology
asadi@sharif.edu

Today's Topics

- Memory & Memory Organization
- Memory Hierarchy
- Principle of Locality
- Cache Memory
  - Directed-mapped
  - Set-associative
  - Fully-associative
  - Cache configuration

Copyright Notice

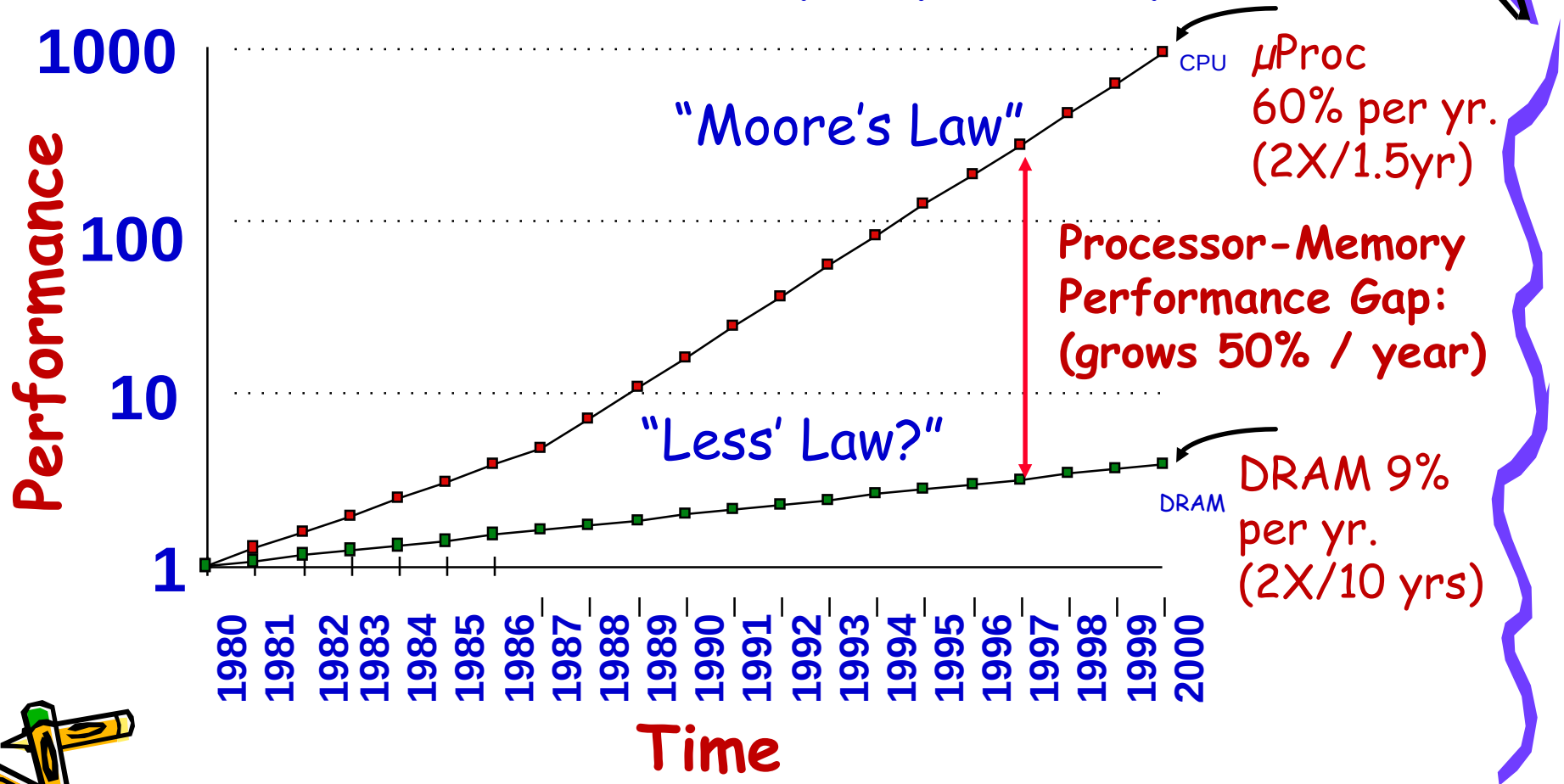
- Parts (text & figures) of this lecture adopted from:
  - Computer Organization & Design, The Hardware/Software Interface, 3rd Edition, by D. Patterson and J. Hennessey, MK publishing, 2005.
  - "Intro to Computer Architecture" handouts, by Prof. Hoe, CMU, Spring 2009.
  - "Computer Architecture & Engineering" handouts, by Prof. Kubiatowicz, UC Berkeley, Spring 2004.
  - "Intro to Computer Architecture" handouts, by Prof. Hoe, UWisc, Spring 2019.
  - "Computer Arch I" handouts, by Prof. Garzarán, UIUC, Spring 2009.
  - "Intro to Computer Organization" handouts, by Prof. Mahlke & Prof. Narayanasamy, Winter 2008.

Ideal Memory

- Processors
  - Would run one instruction per cycle if:
    - Every memory access takes one cycle
    - Every request to memory is successful
- Our Ideal Memory?
  - Very large
  - Can be accessed in one clock cycle
- Reality
  - Any GB-size memory running at Ghz?

Why Memory a Big Deal?

Processor-DRAM Memory Gap (latency)



The Law of Storage

- Bigger is Slower
  - FFs, 512 Bytes, sub-nanosec
  - SRAM, KByte~MByte, ~nanosec
  - DRAM, Gigabyte, ~50 nanosec
  - Hard Disk, Terabyte, ~10 millisec
- Faster is More Expensive ($ and chip area)
  - SRAM < 10$ per Megabyte
  - DRAM, < 1$ per Megabyte
  - Hard Disk < 1$ per Gigabyte
- *Note* these sample values scale with time

Question is:

- How to Make Memory?
  - Bigger,
  - Faster, &
  - Cheaper?

Principle of Locality

- Locality
  - One's recent past is a very good predictor of his/her near future
- Temporal Locality:
  - If you just did something, it is very likely that you will do same thing again soon
- Spatial Locality:
  - If you just did something there, it is very likely you will do something similar/related around again

Locality in Memory

- Locality in Memory
  - A "typical" program has a lot of locality in memory references
  - Programs are sequential and composed of "loops"
- Temporal:
  - A program tends to reference same memory location many times and all within a small window of time
- Spatial:
  - A program tends to reference a cluster of memory locations at a time

Locality in Memory (cont.)

- Example 1:
  - This sequence of addresses has both types of locality

1, 2, 3, 1, 2, 3, 8, 8, 47, 9, 10, 8, 8 …

spatial **temporal** **non-local**

Locality in Memory (cont.)

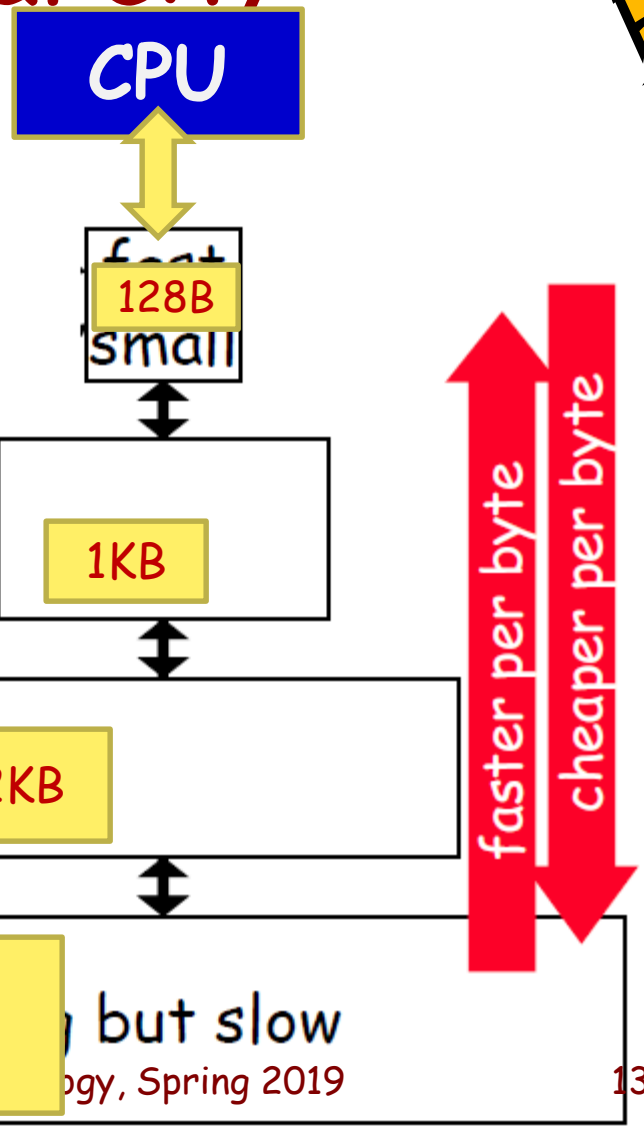
- Example 2:
  - Data
    - Reference array elements in succession (spatial)
  - Instructions
    - Reference instructions in sequence (spatial)
    - Cycle through loop repeatedly (temporal)

```
sum = 0;
for (i = 0; i < n; i++)
    sum += a[i];
*v = sum;
```

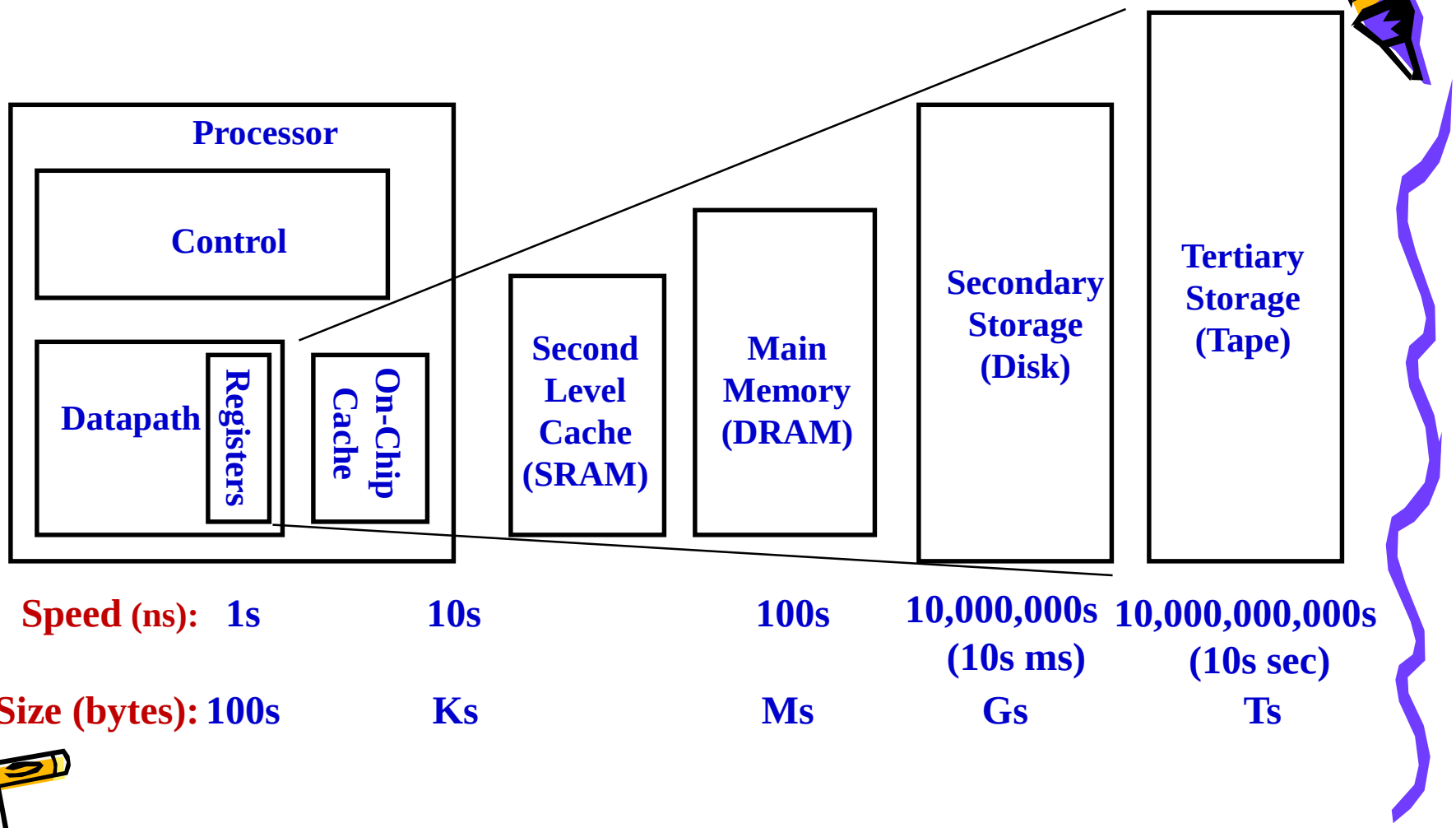
Probability of Reference

Memory Hierarchy

- Faster & Small
- Bigger & Slower



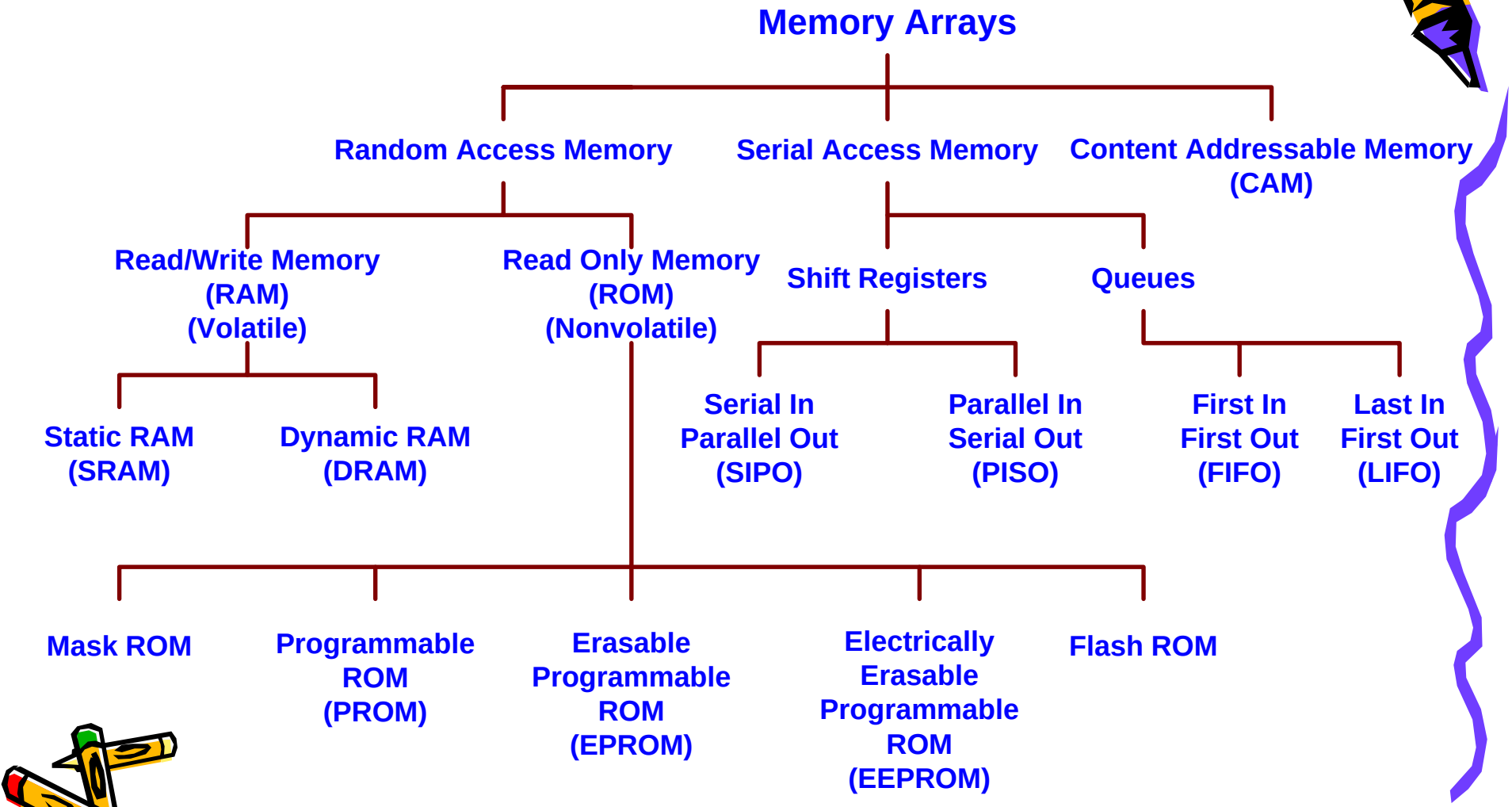
Memory Hierarchy (cont.)



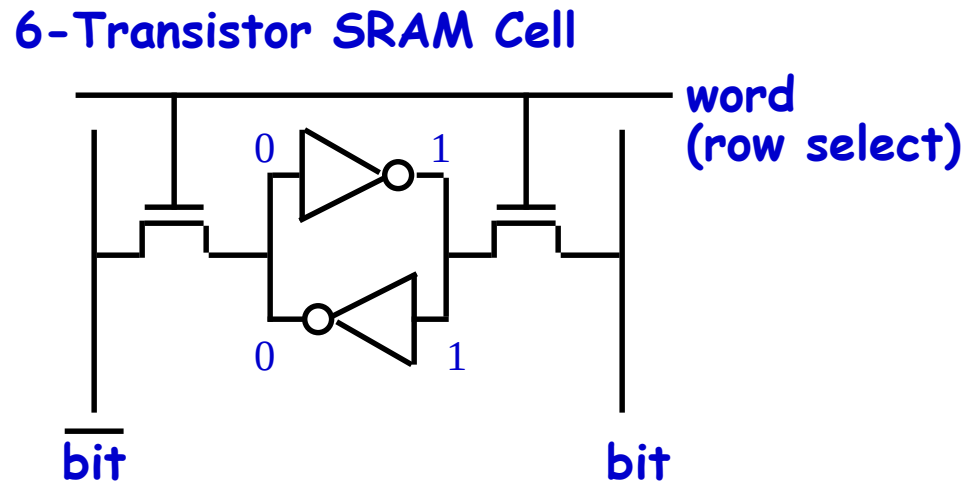
Technology Used in Main Memory & Cache Memory

- SRAM (Static Random Access Memory)
  - No refresh (6 transistors/bit vs. 1 transistor)
  - # of transistors per bit: DRAM < SRAM
  - Cycle time: DRAM > SRAM
- DRAM (Dynamic Random Access Memory)
  - Dynamic since needs to be refreshed periodically
  - Addresses divided into 2 halves
    - Memory as a 2D matrix
    - RAS or Row Address Strobe
    - CAS or Column Address Strobe

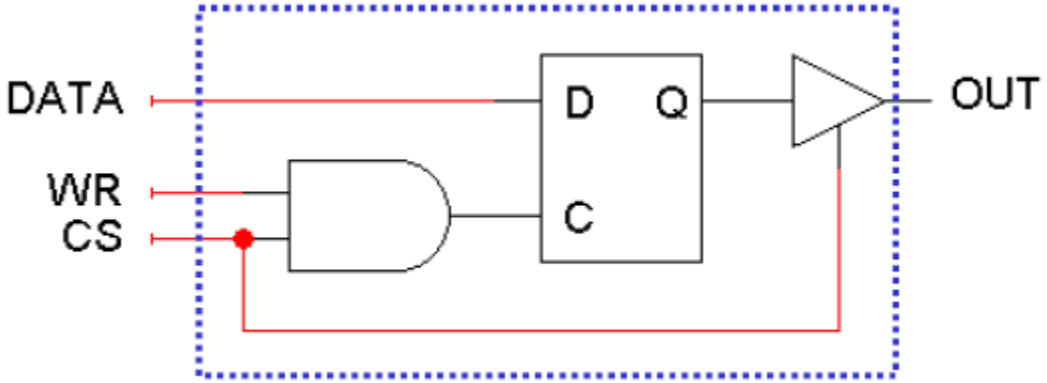
Memory Arrays

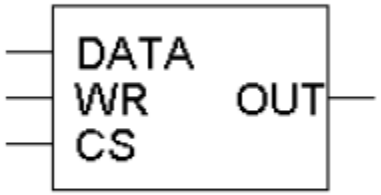


Static RAM Cell



How to Build Big Memory?

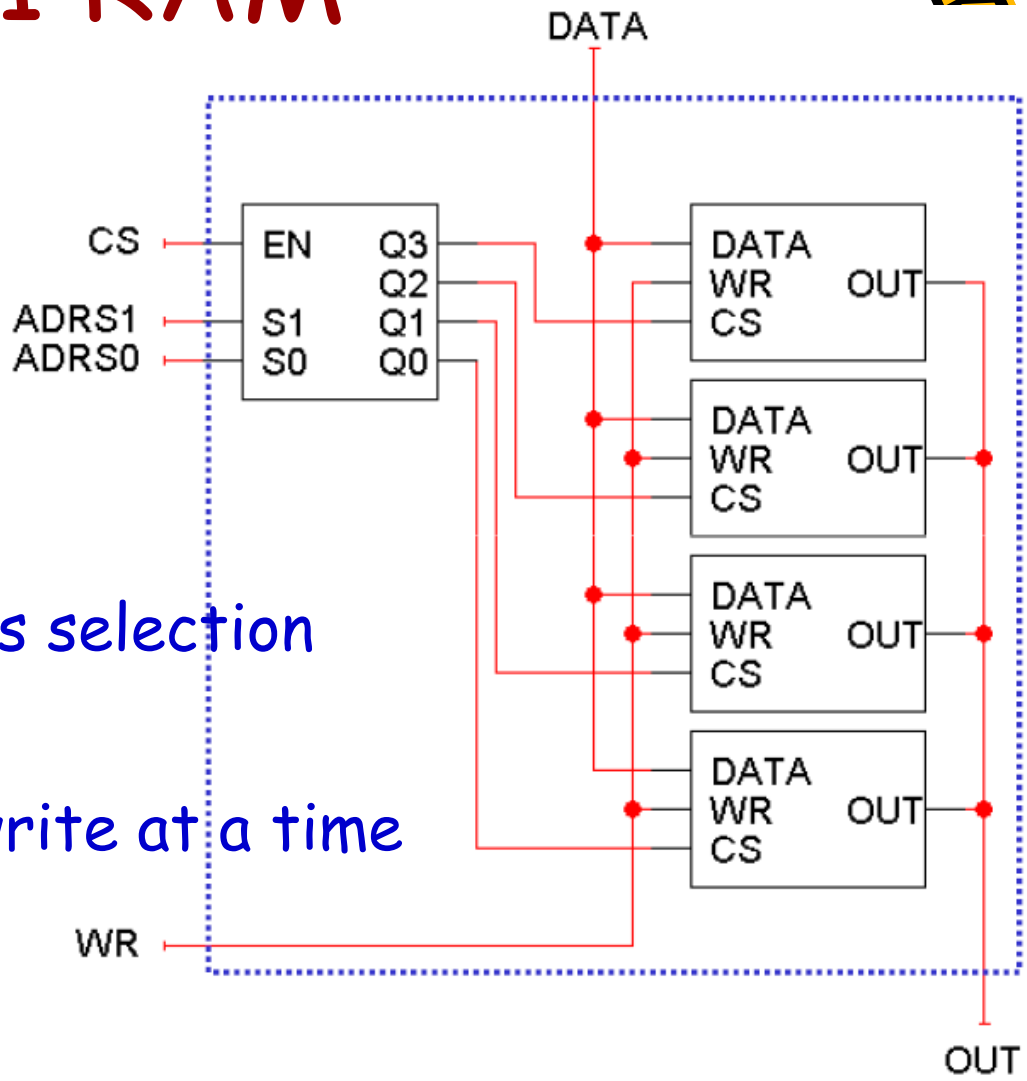




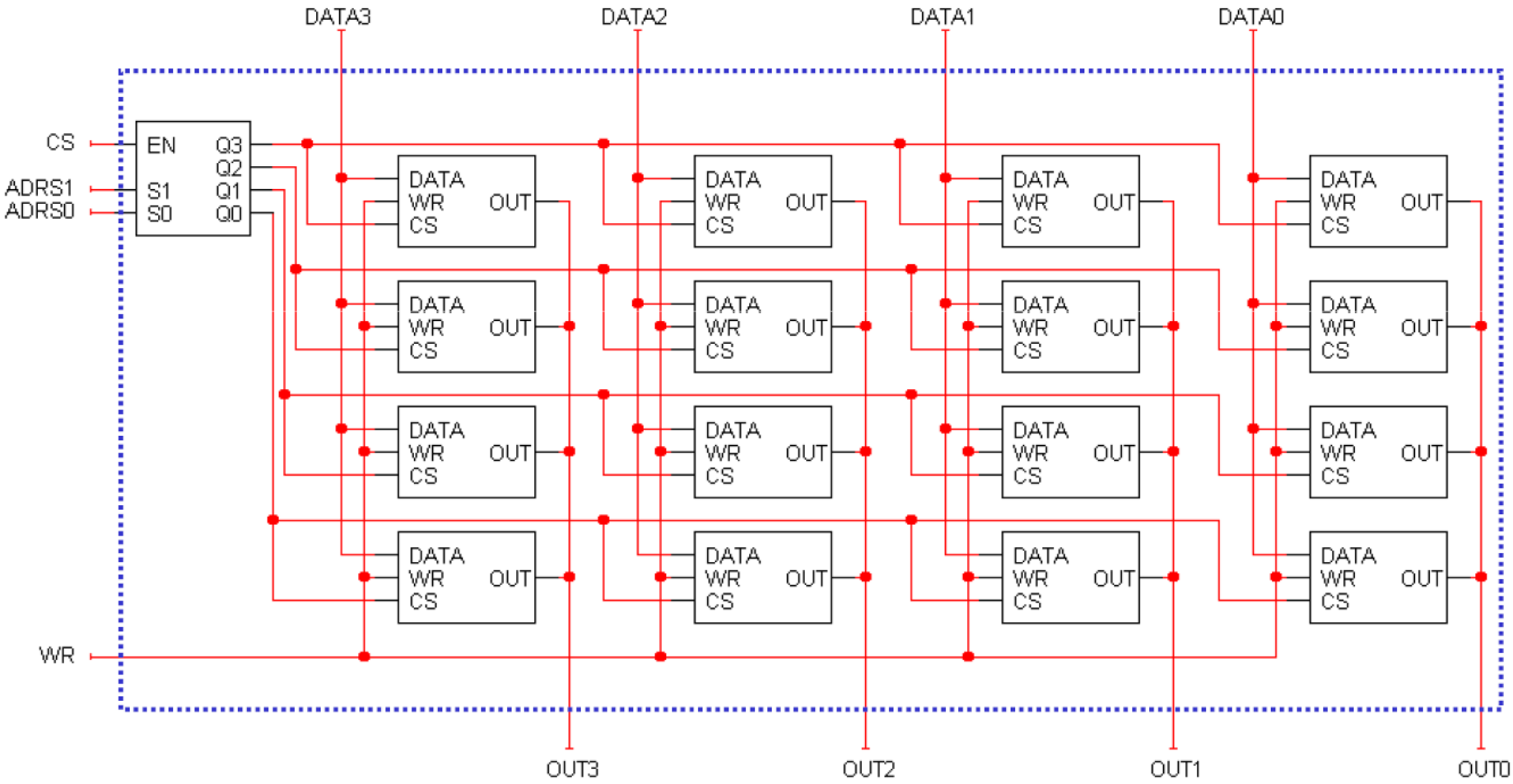
- CS / OE
  - Chip select or Output Enable
- WR
  - Write enable
- Address not required (one-bit)

4x1 RAM

- Address Lines
  - Two bits
- Data Line
  - One bit
- Decoder
  - Used for address selection
- Only One Bit
  - Can be read or write at a time

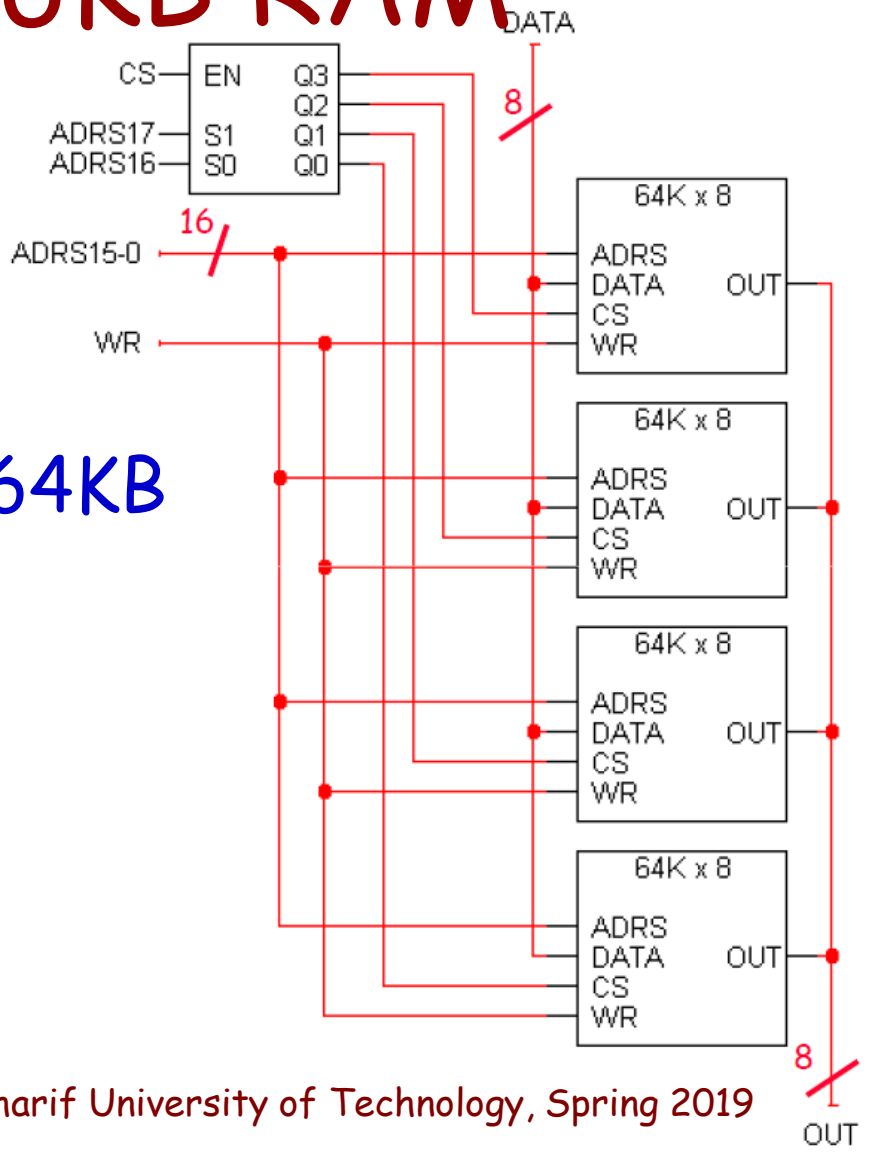


4x4 RAM

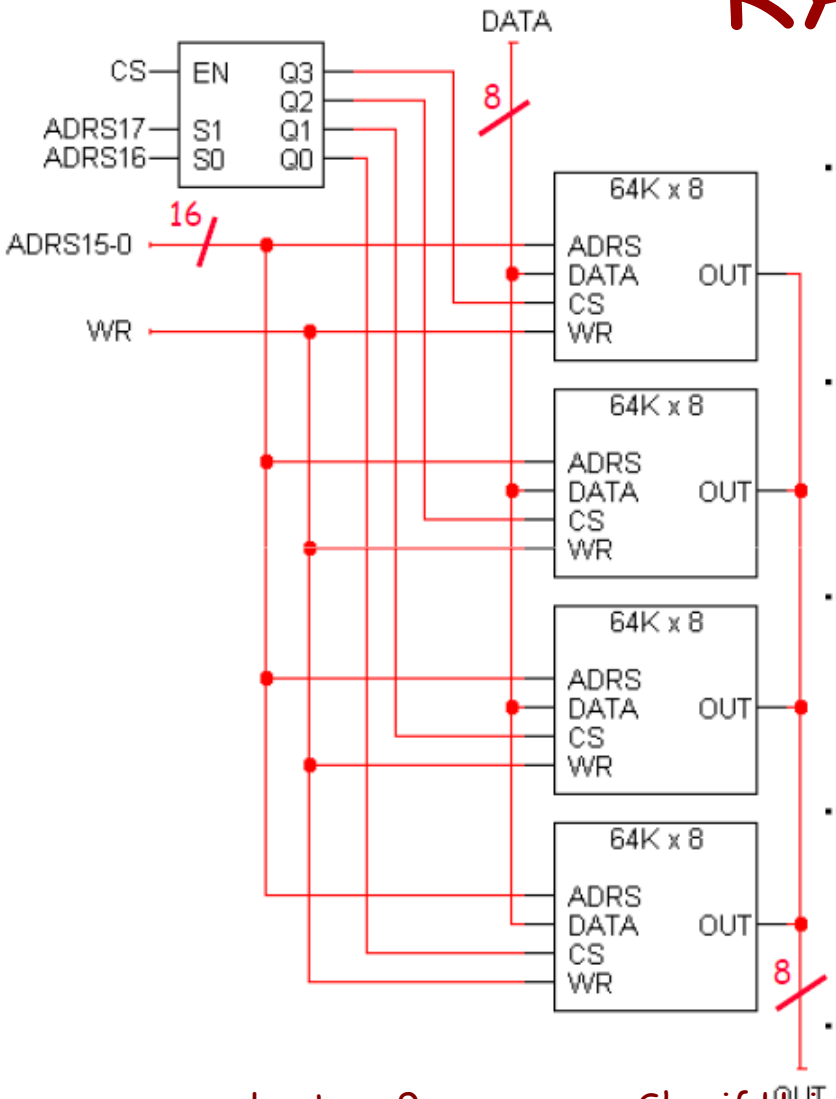

DATA3
DATA2
DATA1
DATA0
CS
ADRS1
ADRS0
EN
Q3
Q2
Q1
Q0
S1
S0
DATA
WR
CS
OUT
WR
OUT3
OUT2
OUT1
OUT0

256KB RAM

- Made of Four 64KB RAM chips

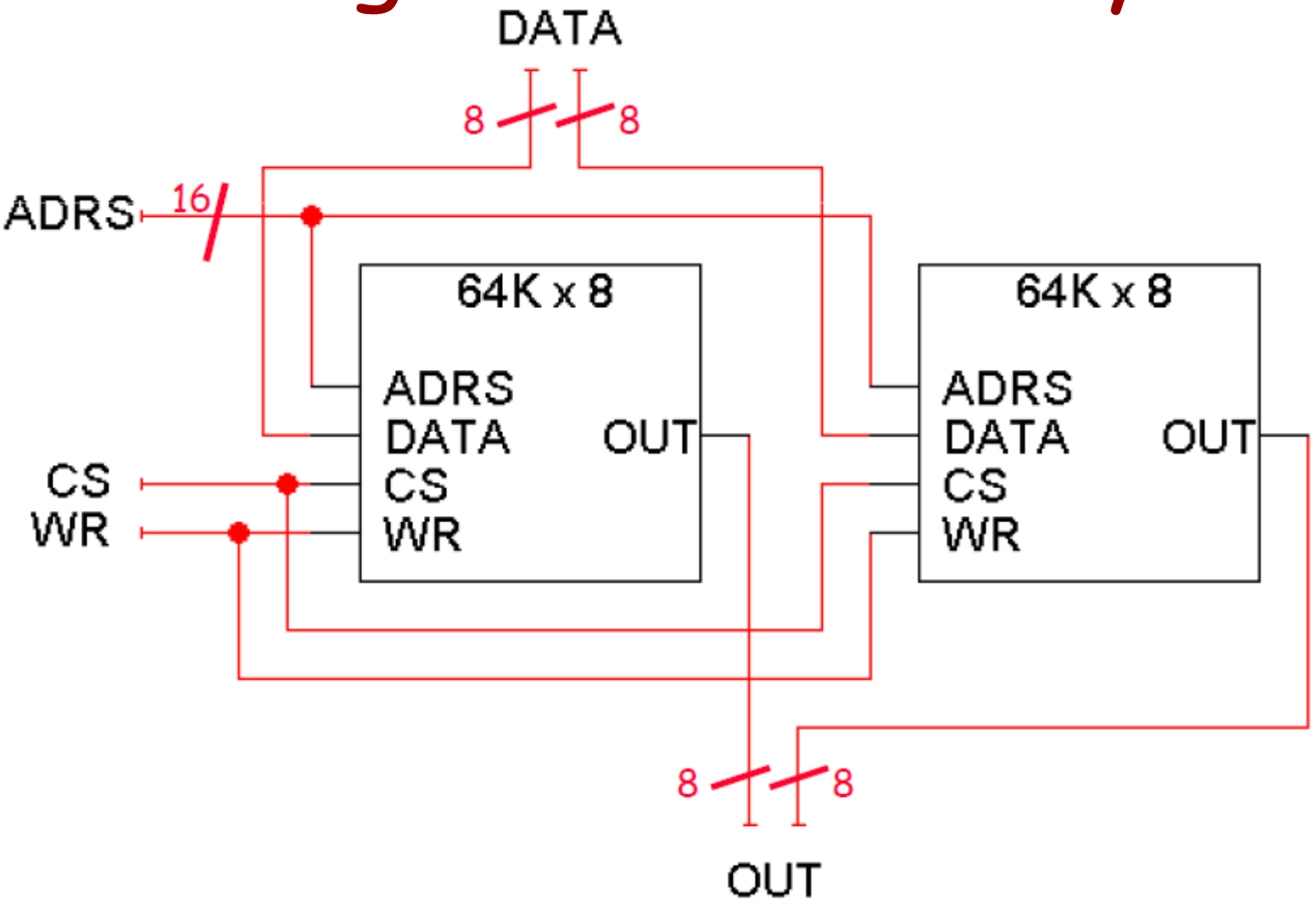


Address Ranges in 256KB RAM

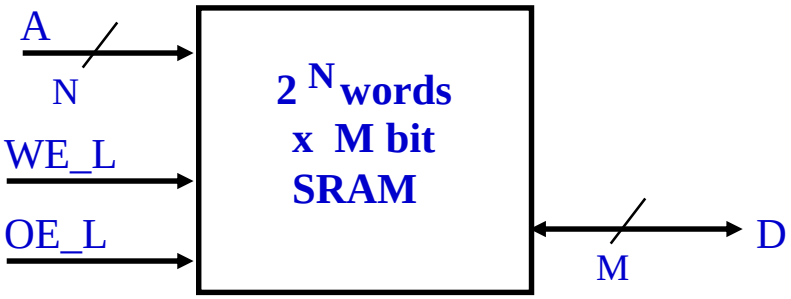


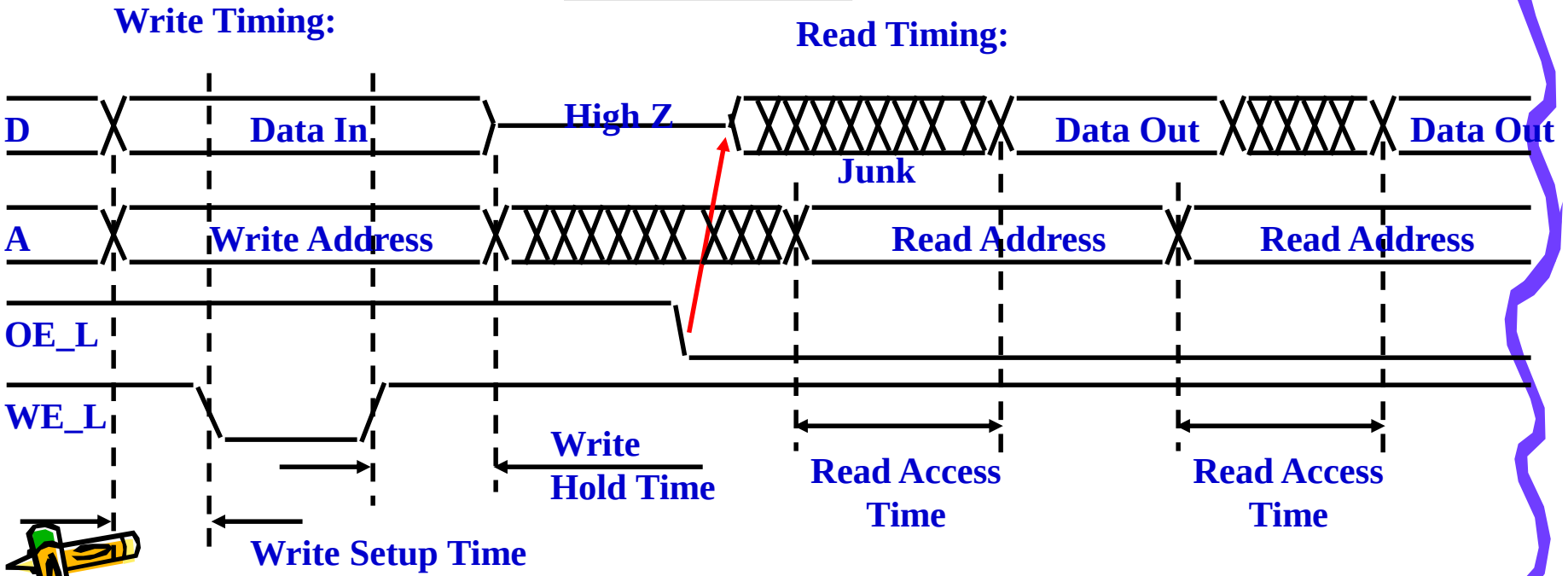
11 1111 1111 1111 1111	(0x3ffff)
to	
11 0000 0000 0000 0000	(0x30000)
10 1111 1111 1111 1111	(0x2ffff)
to	
10 0000 0000 0000 0000	(0x20000)
01 1111 1111 1111 1111	(0x1ffff)
to	
01 0000 0000 0000 0000	(0x10000)
00 1111 1111 1111 1111	(0x0ffff)
to	
00 0000 0000 0000 0000	(0x00000)

Making Wider Memory



Typical SRAM Timing





Slow Memory in Pipeline Datapath

- Freeze pipeline in *Mem* stage:

```
IF0   ID0  EX0  Mem0  Wr0    Noop   …  Noop   Noop
      IF1  ID1  EX1   Mem1   stall  …  stall  Mem1  Wr1
           IF2  ID2   EX2    stall  …  stall  Ex2   Mem2  Wr2
                IF3   ID3    stall  …  stall  ID3   Ex3   Mem3  Wr3
                      IF4    stall  …  stall  IF4   ID4   Ex4   Mem4  Wr4
                                              IF5   ID5   Ex5   Mem5
```

- Stall detected by end of Mem1 stage

CPU Performance

- CPU Time = (CPU execution clock cycles + memory stall clock cycles) x clock cycle time
- Memory Stall Clock Cycles = (reads x read miss rate x read miss penalty + writes x write miss rate x write miss penalty)
- Memory Stall Clock Cycles = Memory accesses x Miss rate x Miss penalty

CPU Performance (cont.)

- Different Measure:
  - Average Memory Access Time (AMAT)
- Expressed in Terms of:
  - Hit time
  - Miss rate
  - Miss penalty

CPU Performance (cont.)

- Hit Time
  - Time required to access a level of memory hierarchy including time required to determine whether access is hit or miss
- Hit Rate (Hit Ratio)
  - Fraction of memory accesses found in a cache
  - Miss Rate = 1 – Hit Rate

CPU Performance (cont.)

- Miss Penalty:
  - Time required to fetch a block into a level of memory hierarchy from lower level:
    - Time to access block +
    - Time to transmit it to higher level +
    - Time to insert it in appropriate block
- Block
  - Minimum unit of information transferred between two levels of memory hierarchy
  - Also called line

CPU Performance (cont.)

- AMAT = Hit Time +
  (Miss Rate x Miss Penalty)

CPU Performance (cont.)

- Example 1
  - A memory system consists of a cache and a main memory
  - Cache hit = 1 cycle
  - Cache miss = 100 cycles
  - What is average memory access time if hit rate in cache is 97%? 1+(3/100 * 100) = 4

CPU Performance (cont.)

- Example 2
  - A memory system has a cache, a main memory, and a virtual memory
  - Hit rate = 98%
  - Hit rate in main memory = 99%
  - 2 cycles to access cache
  - 150 cycles to fetch a line from main mem.
  - 100,000 cycles to access virtual memory
  - What is average memory access time?

Improving Cache Performance

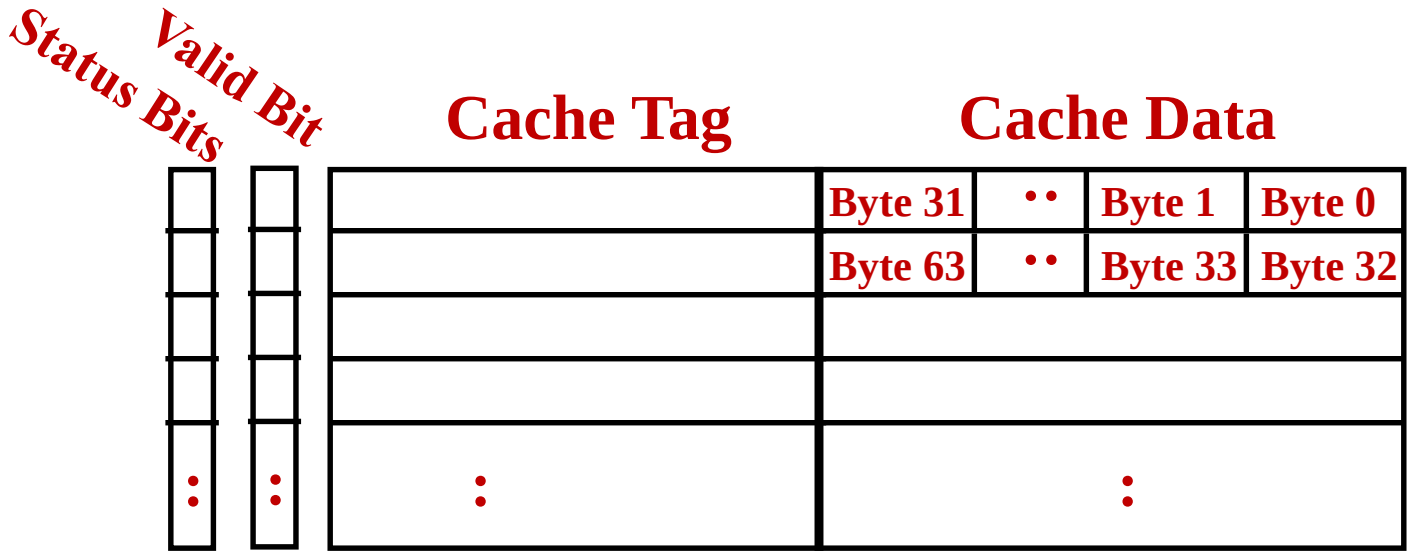
- AMAT =

  Hit Time + (Miss Rate x Miss Penalty)

- Options to Reduce AMAT
  - Reduce time to hit in cache
    - Use smaller cache size
  - Reduce miss rate
    - Increase cache size!
  - Reduce miss penalty
    - Use multi-level cache hierarchy

Cache Structure

- Data Bits
- Tag Bits
- Invalid Bit
- Status Bits (LRU bit, Dirty Bit, …)



Cache Configuration

- Q1:
  - Where can a block be placed in upper level?
  - Block placement
- Q2:
  - How is a block found if it is in upper level?
  - Block identification

Cache Configuration (cont.)

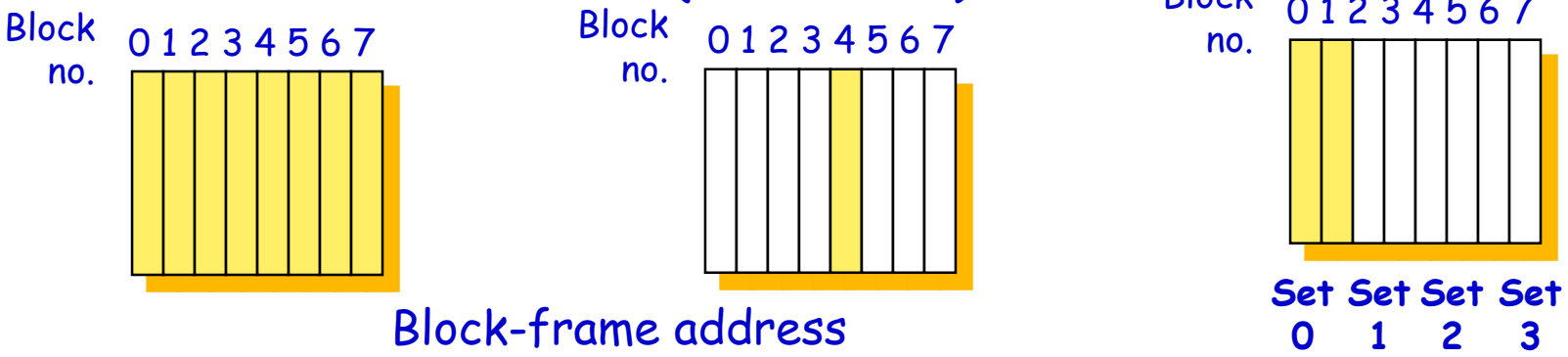
- Q3:
  - Which block should be replaced on a miss?
  - Block replacement
- Q4:
  - What happens on a write?
    - How to propagate changes?
  - Write strategy

Q1: Where can a block be placed in upper level?

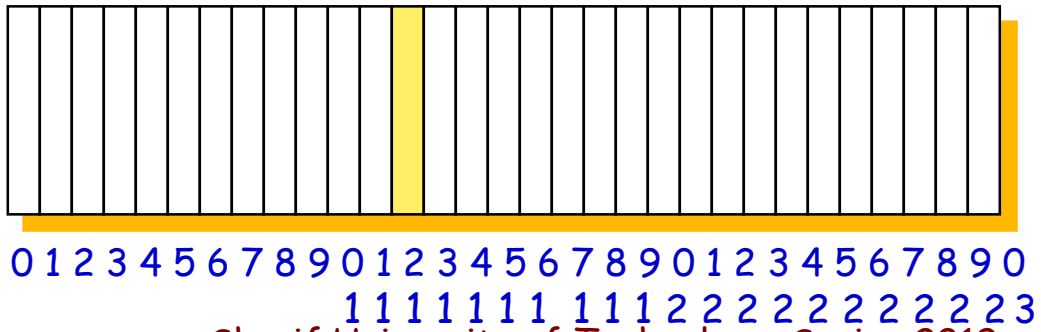
Fully associative: block 12 can go anywhere

Direct mapped: block 12 can go only into block 4 (12 mod 8)

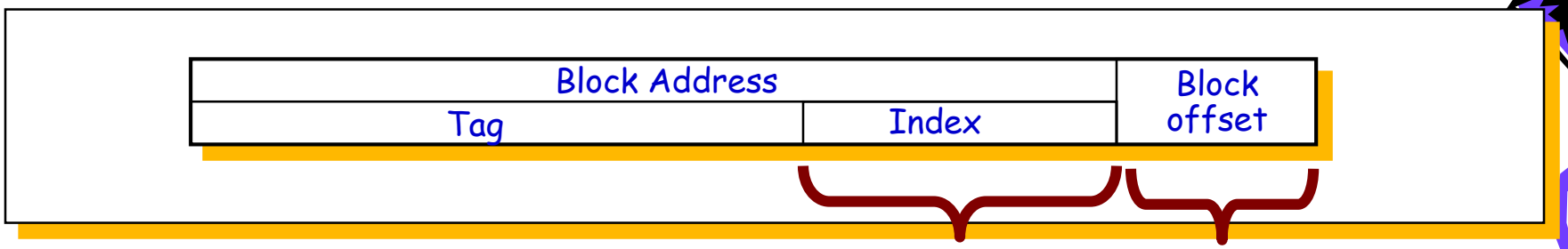
Set associative: block 12 can go anywhere in set0 (12 mod 4)



Block-frame address



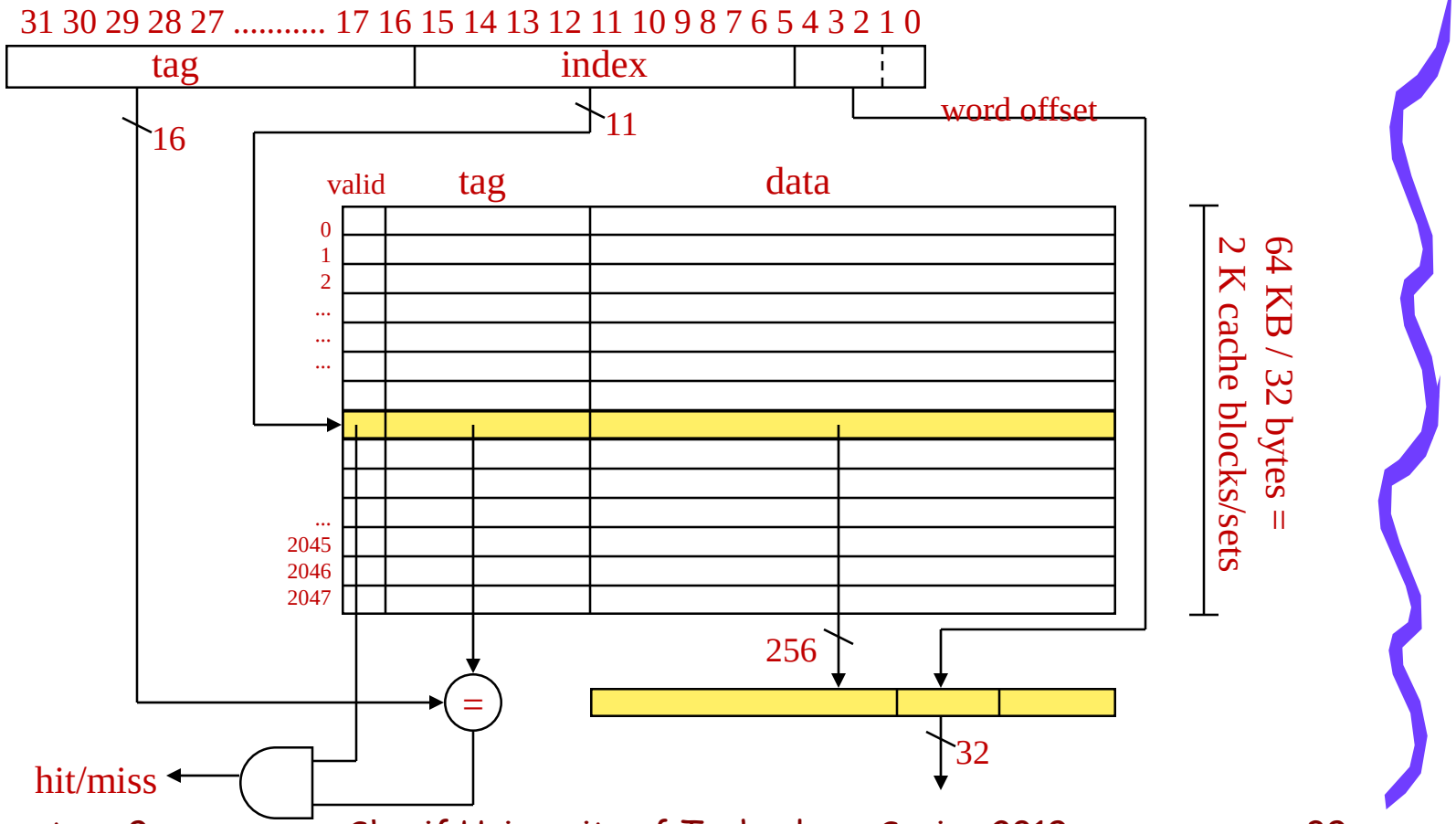
Q2: How is a block found if it is in upper level?



- Offset
  - Identifies a byte/word within a block
- Index
  - Identifies corresponding set
- Tag
  - Identifies whether associated block corresponds to a requested word or not

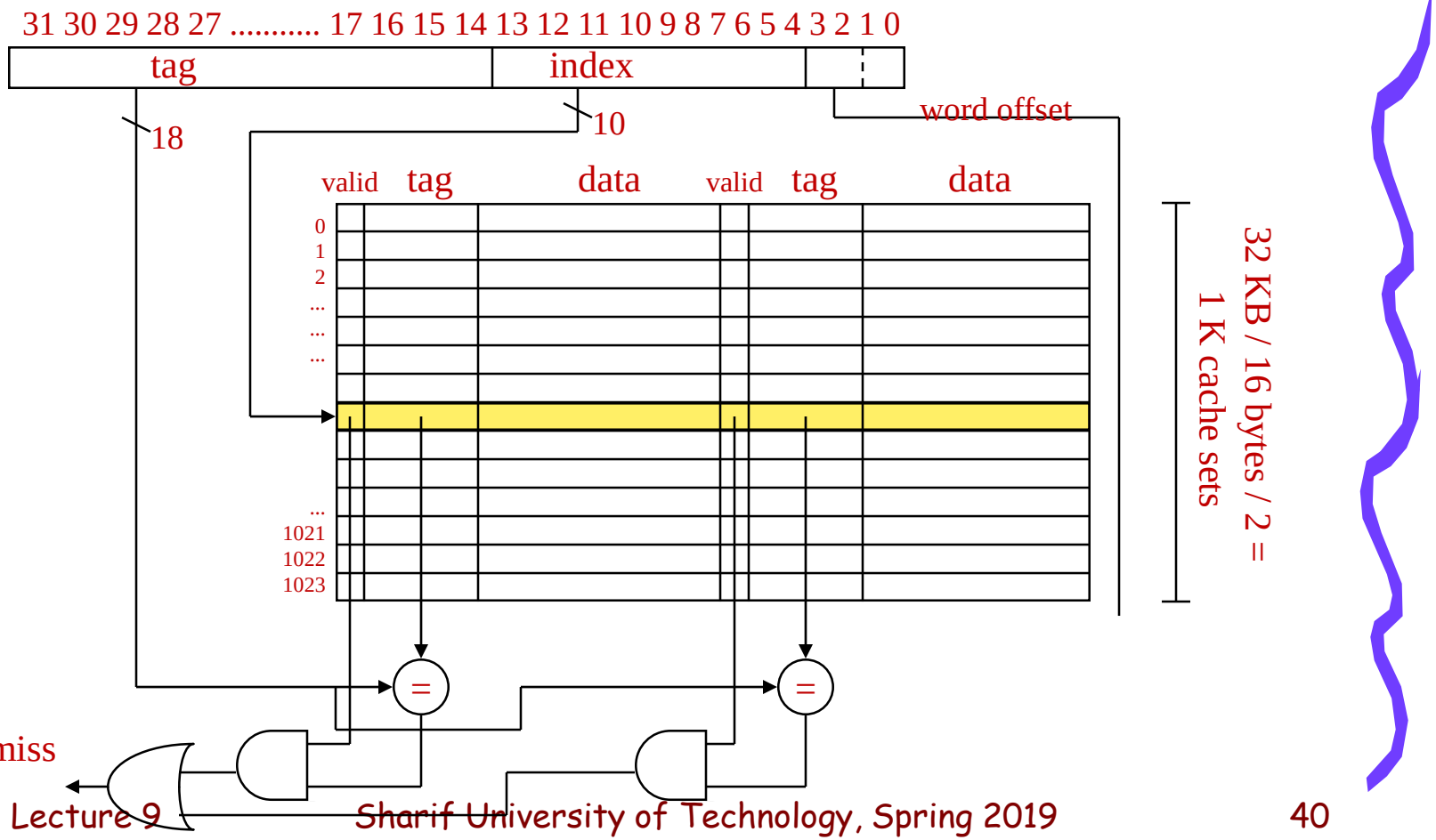
Direct-Mapped

64 KB cache, 32-byte cache block



Set-Associative

32 KB cache, 2-way,16-byte blocks



Cache Parameters

- Cache size =

  # of sets * block size * associativity

- Example 1
  - 128 blocks, 32-byte blocks, direct mapped, size = ?
- Example 2
  - 128 KB cache, 64-byte blocks, 512 sets, associativity = ?

Direct-Mapped vs. Fully-Associative

- Direct-Mapped
  - Require less area
    - Only one comparator
    - Fewer tag bits required
  - Fast hit time
    - Less bits to compare
  - Cache block is available BEFORE Hit/Miss
    - Return data to CPU in parallel with hit process
    - Possible to assume a hit and continue recover later if miss
  - Conflict misses reduce hit rate

Direct-Mapped vs. Fully-Associative (cont.)

- Fully-Associative
  - More area compared to direct-mapped
    - Need one comparator for each line in cache
  - Longer hit time
    - Too many comparison required
  - Less miss rate vs. direct-mapped
    - No conflict misses (conflict Miss = 0)
  - No cache index
    - Compare tag with all tags of all cache entries in parallel

Q3: Which block should be replaced on a miss?

- Easy for Direct Mapped; why?
- Set Associative or Fully Associative:
  - Random
  - LRU (Least Recently Used): in status bits
  - FIFO

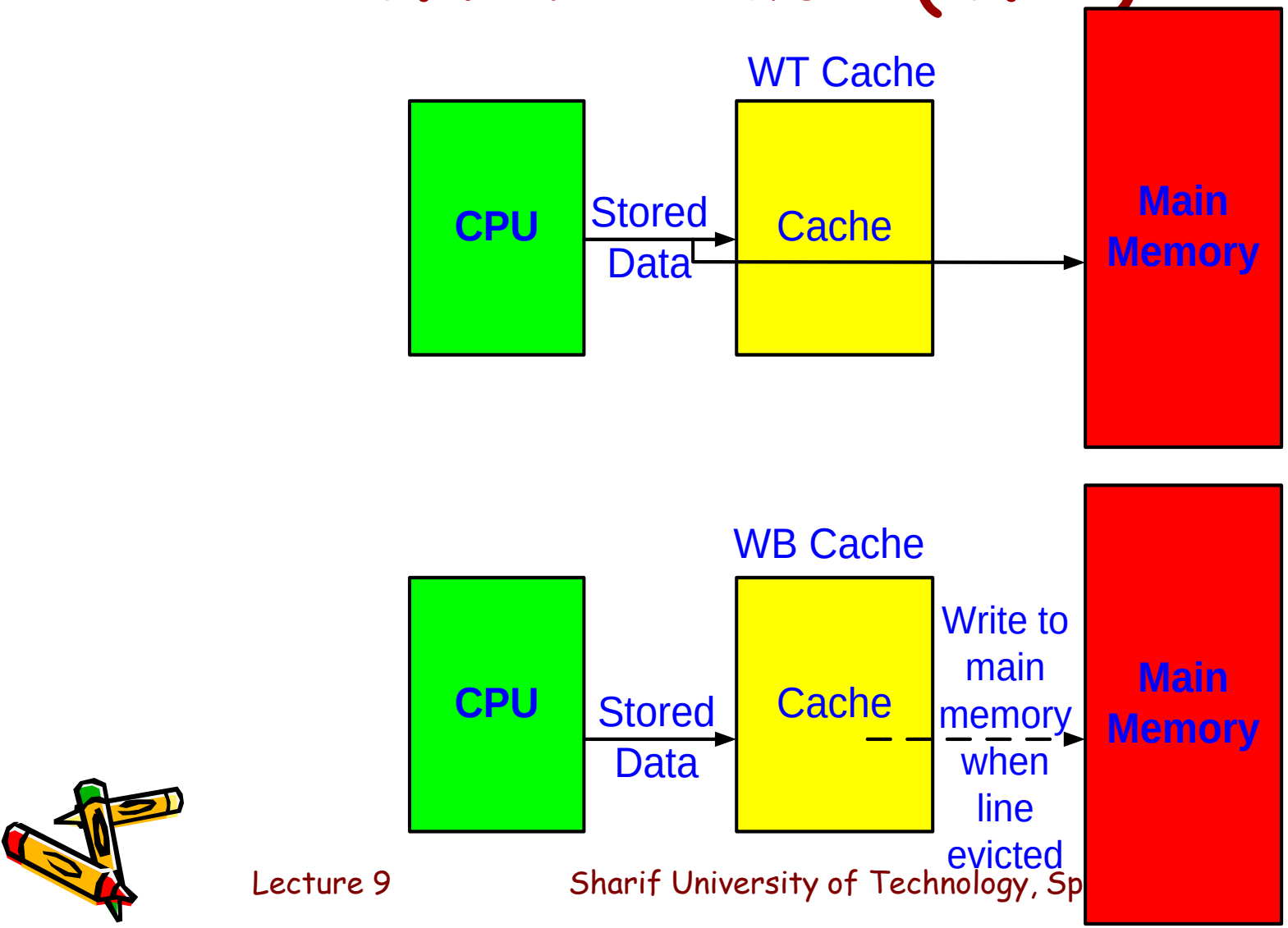
Associativity:

	2-way		4-way		8-way	
Size	LRU	Random	LRU	Random	LRU	Random
16 KB	5.2%	5.7%	4.7%	5.3%	4.4%	5.0%
64 KB	1.9%	2.0%	1.5%	1.7%	1.4%	1.5%
256 KB	1.15%	1.17%	1.13%	1.13%	1.12%	1.12%

Q4: What happens on a write?

- Write Through (Allocate/Non-Allocate)
  - Information written to both block in cache and to block in lower-level memory
- Write Back
  - Information written only to block in cache
  - Modified cache block written to main memory only when it is replaced
  - Inconsistent
    - Need cache coherency policy for multi-core chips
  - Is block clean or dirty? (status bits)

Write-Through (WT) vs. Write-Back (WB)



WT Cache

- Pros
  - Simpler to implement
  - Don't need dirty bit
  - No interface issues with I/O devices
    - Cache memory consistent with memory

WT Cache (cont.)

- Cons
  - Less performance vs. WB cache
  - Processor held up on writes unless writes buffered
- Write Buffer
  - Stores data while waiting to be written to memory

WB Cache

- Pros
  - Tends to have better performance
    - Repeated writes not sent to DRAM
    - Processor not held up on writes
    - Combines multiple writes into one line WB
  - Virtual memory systems use write-back
    - because of huge penalty for going out to disk

WB Cache (cont.)

- Cons
  - More complex
    - Read miss may require writeback of dirty data
  - Need to implement cache coherency
  - Typically requires two cycles on writes
    - Can't overwrite data and do tag comparison at same time as block may be dirty
    - Unless using store buffer

Write Policy in WT Caches

- Allocate-on-Write (Write Allocate)
  - Fetch line into cache
  - Then perform write in cache
  - Also called, fetch-on-miss, fetch-on-write
- No-Allocate-on-Write (No-Write Allocate)
  - Pass write through to main memory
  - Don't bring line into cache
  - Also called Write-Around or Read-Only

Write Policy in WT Caches

- Allocate-on-Write Pros
  - Better performance if data referenced again before it is evicted
- No-Allocate-on-Write Pros
  - Simpler write hardware
  - May be better for small caches if written data won't be read again soon

L1 Cache Configuration

- Split Cache
  - Two independent caches
    - Instruction cache (IL1)
    - Data cache (DL1) Text
- Unified

علتش این است که در پیکربندی قبلی برای این که مشخص کنیم الان hit داریم یا miss باید هر دو cache دستورات و داده را جستجو میکردیم
بنابراین hit rationکمتری داریم
ولی در این پیکربندی جدید فقط باید یک cache را جستجو کنیم

  - One unified L1 cache
  - Usually better hit ratio (same size); why?
- Question:

به این دلیل که دیگر structural hazard بین مراحل IF و MEM نخواهیم داشت
در صورتی که اگر یک cacheداشته باشم باعث بروز stallهنگام دستورات load و store می شود

  - Most processors use split caches; why?

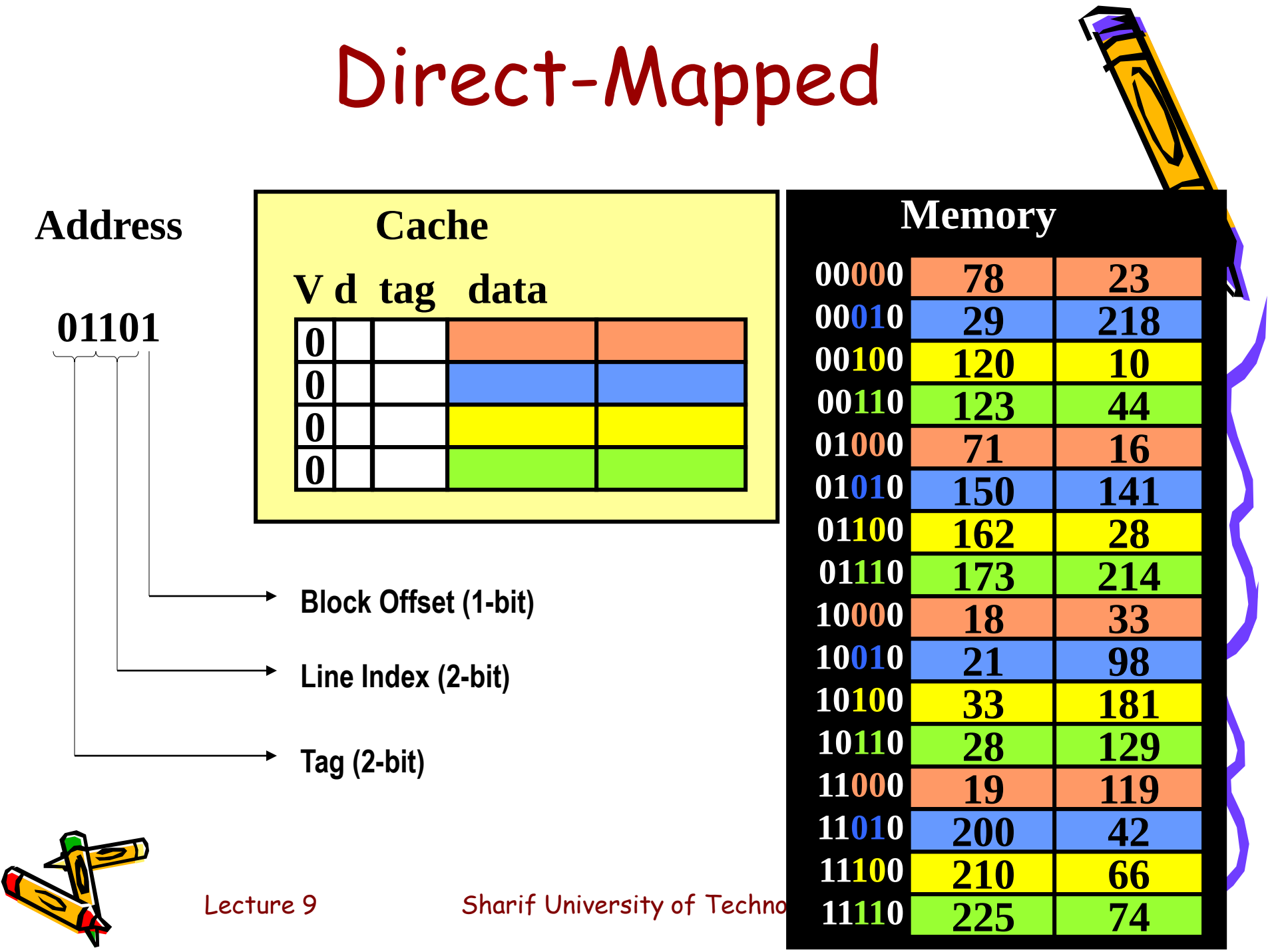
Practice

- Consider a 16KB Cache
  - 4-way, 32-bit address, byte-addressable memory, 32-byte cache blocks
- Q1:

offset. = 2 bit
index = 7 bit
tag = 32-9 = 23 bit

  - How many tag bits?
  - Total tag bits in cache?
- Q2:
  - Where to find word with address = 0x200356A4?

Direct-Mapped



2-Way Set Associative

Address

01101

Cache

V	d	tag	data	
0				
0				
0				
0				

Block Offset (unchanged)

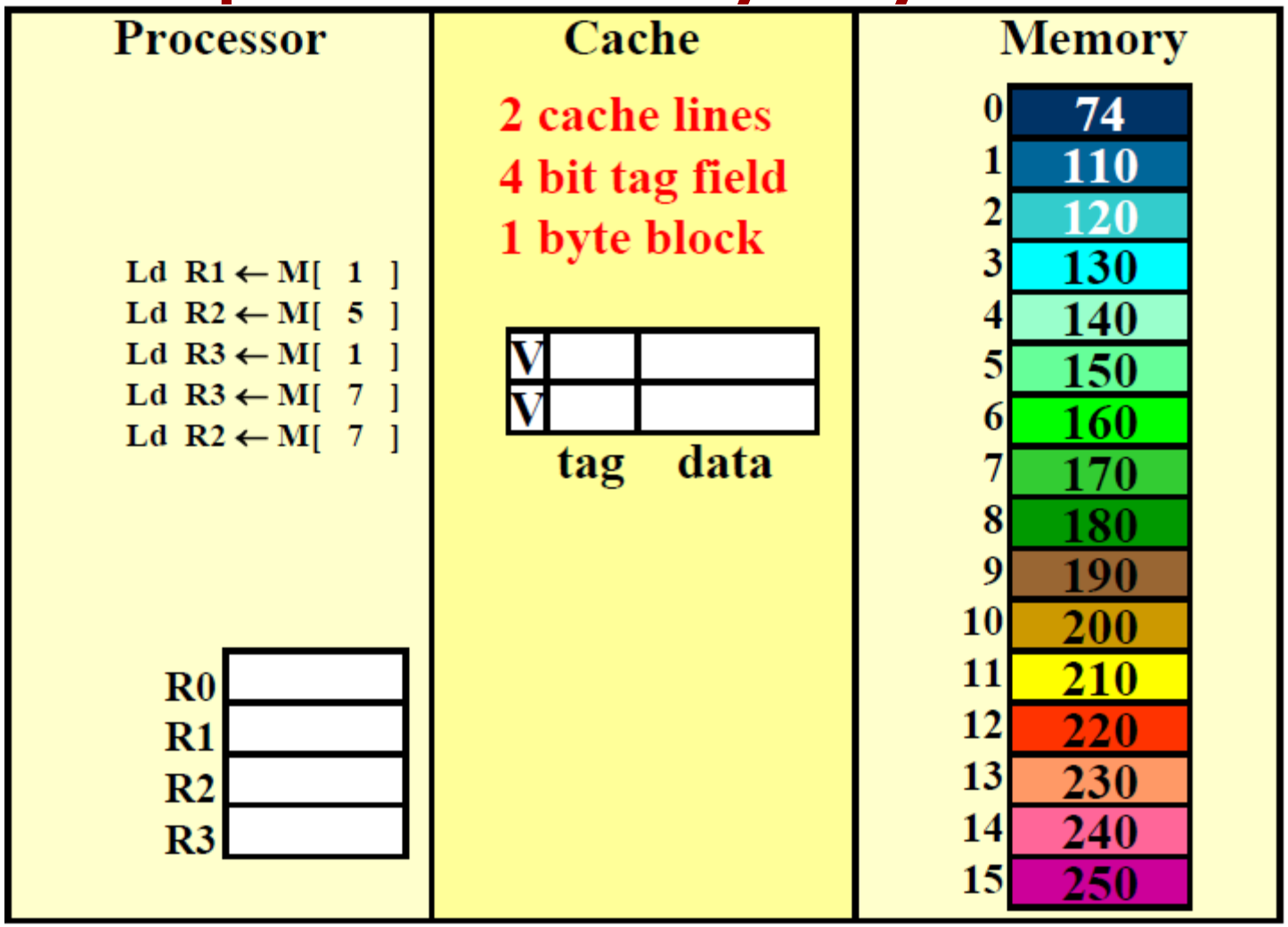
1-bit Set Index

Larger (3-bit) Tag

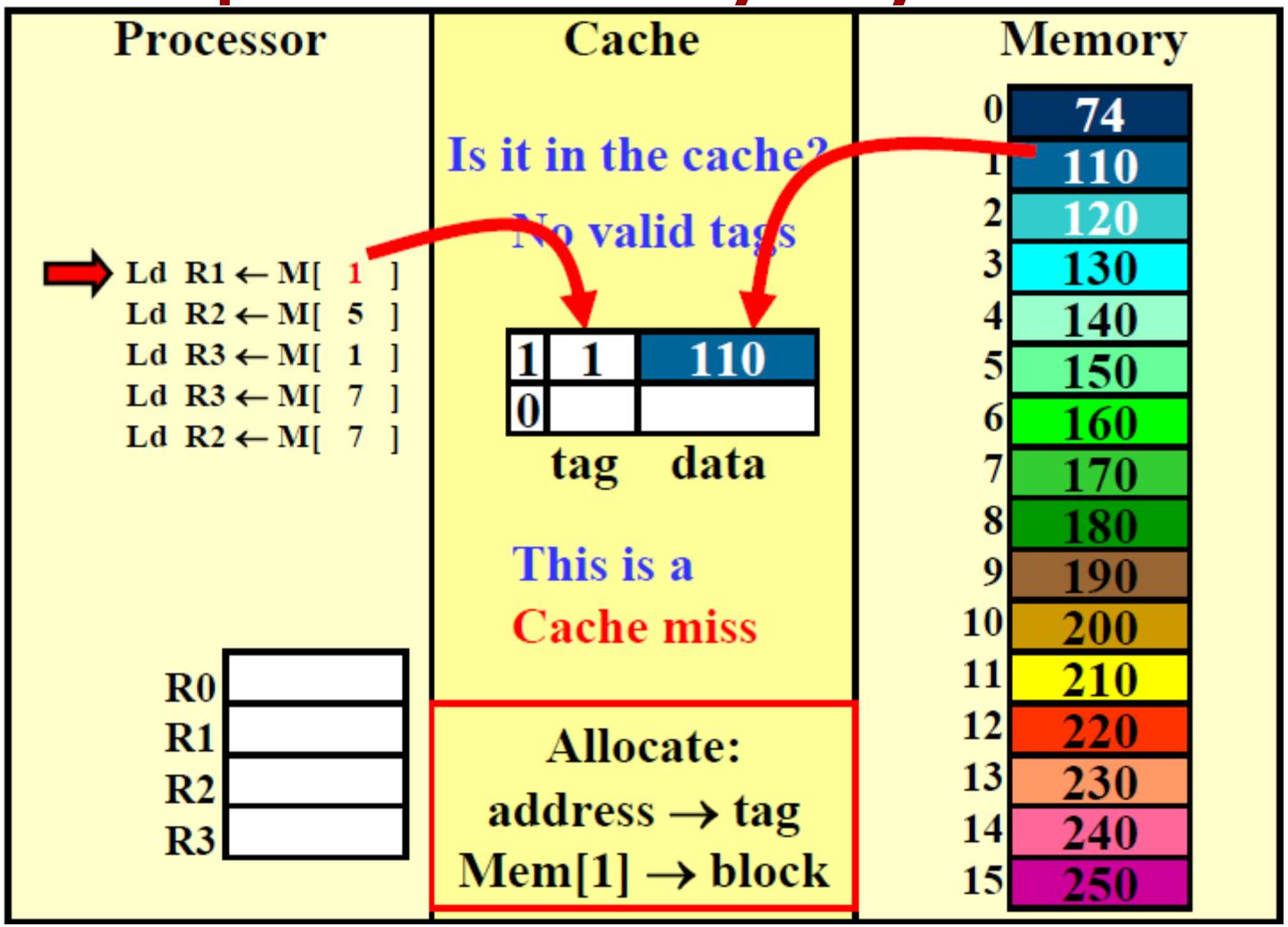
Memory

00000	78	23
00010	29	218
00100	120	10
00110	123	44
01000	71	16
01010	150	141
01100	162	28
01110	173	214
10000	18	33
10010	21	98
10100	33	181
10110	28	129
11000	19	119
11010	200	42
11100	210	66
11110	225	74

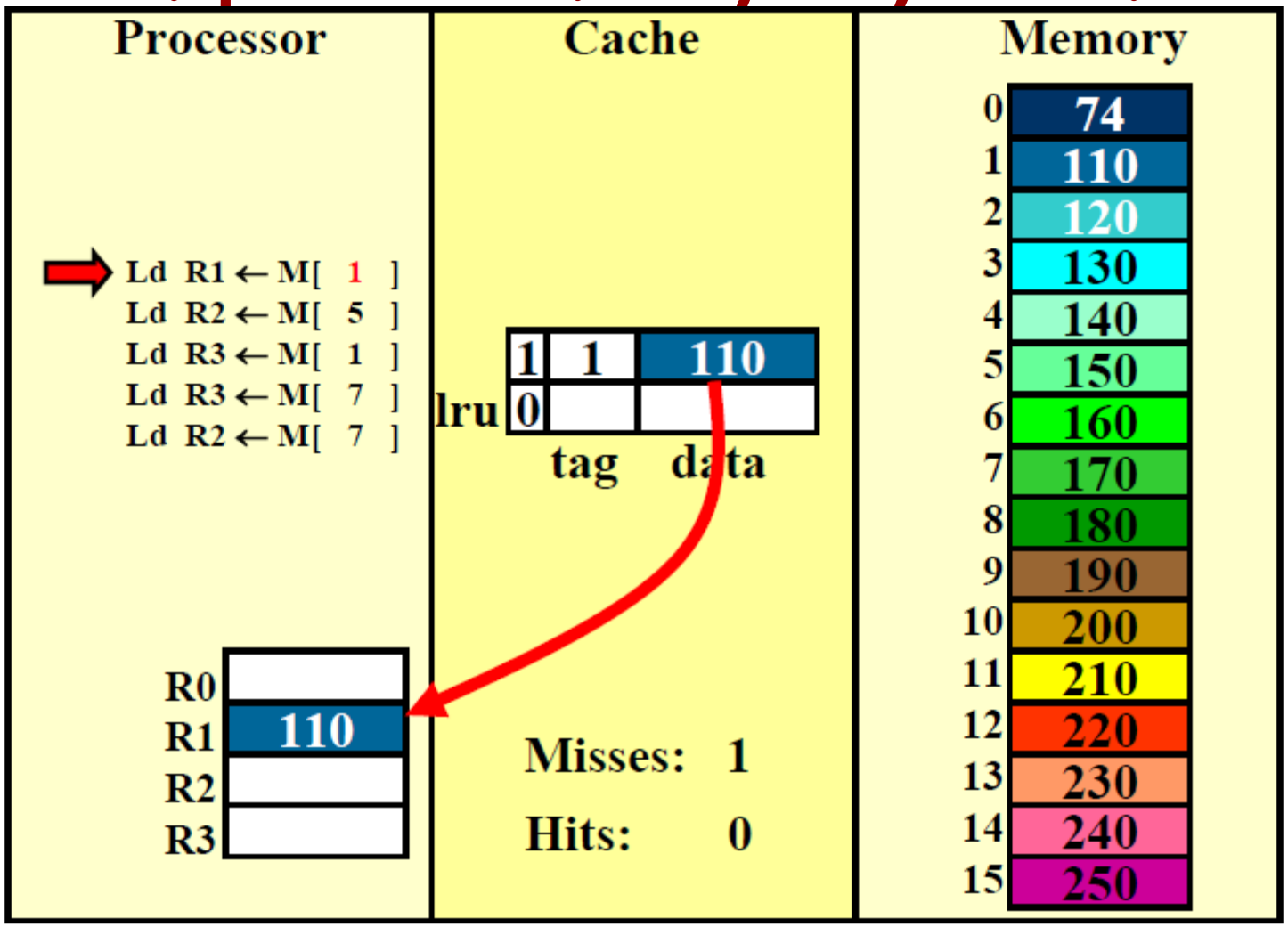
Simple Memory System



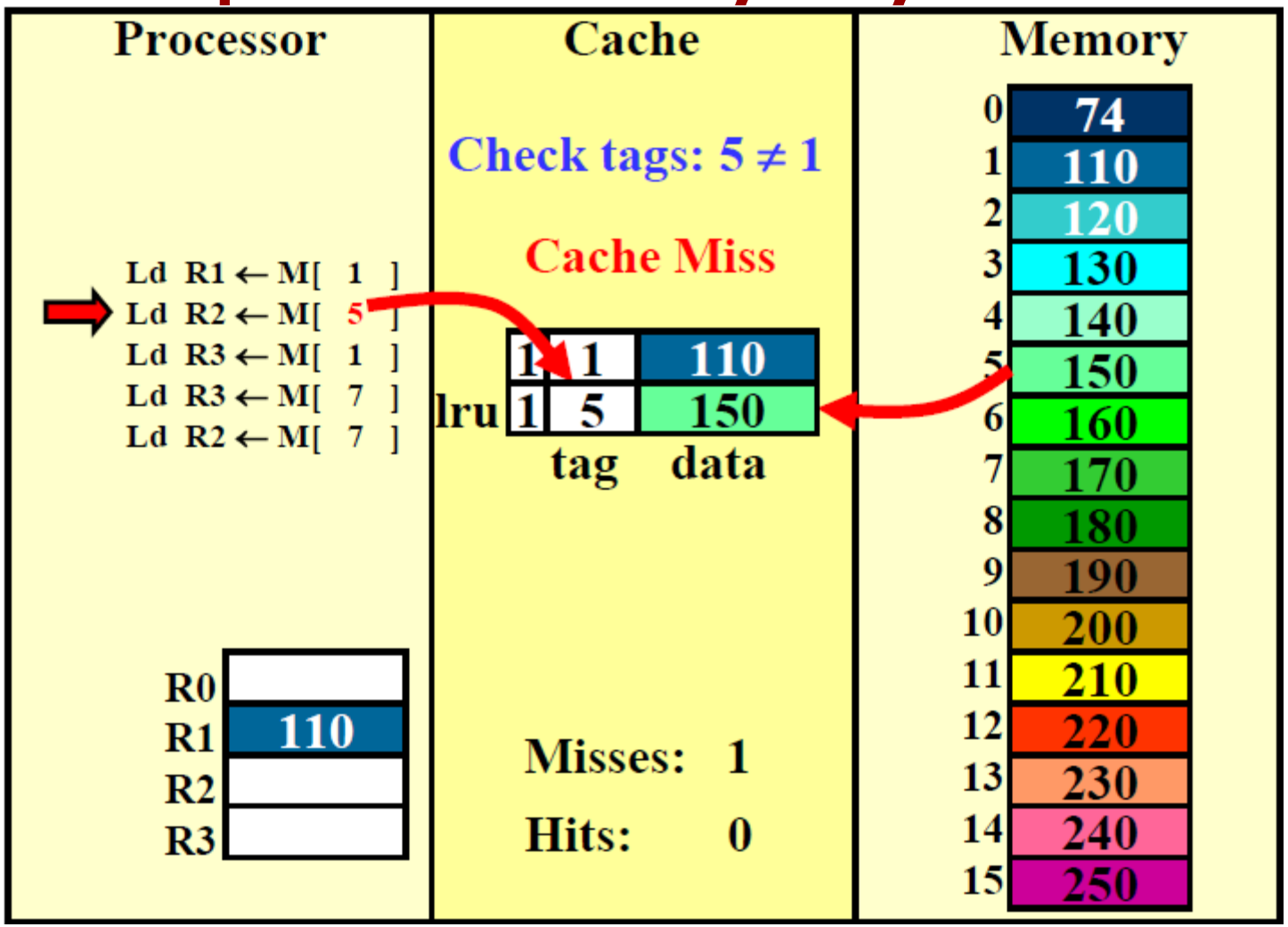
Simple Memory System



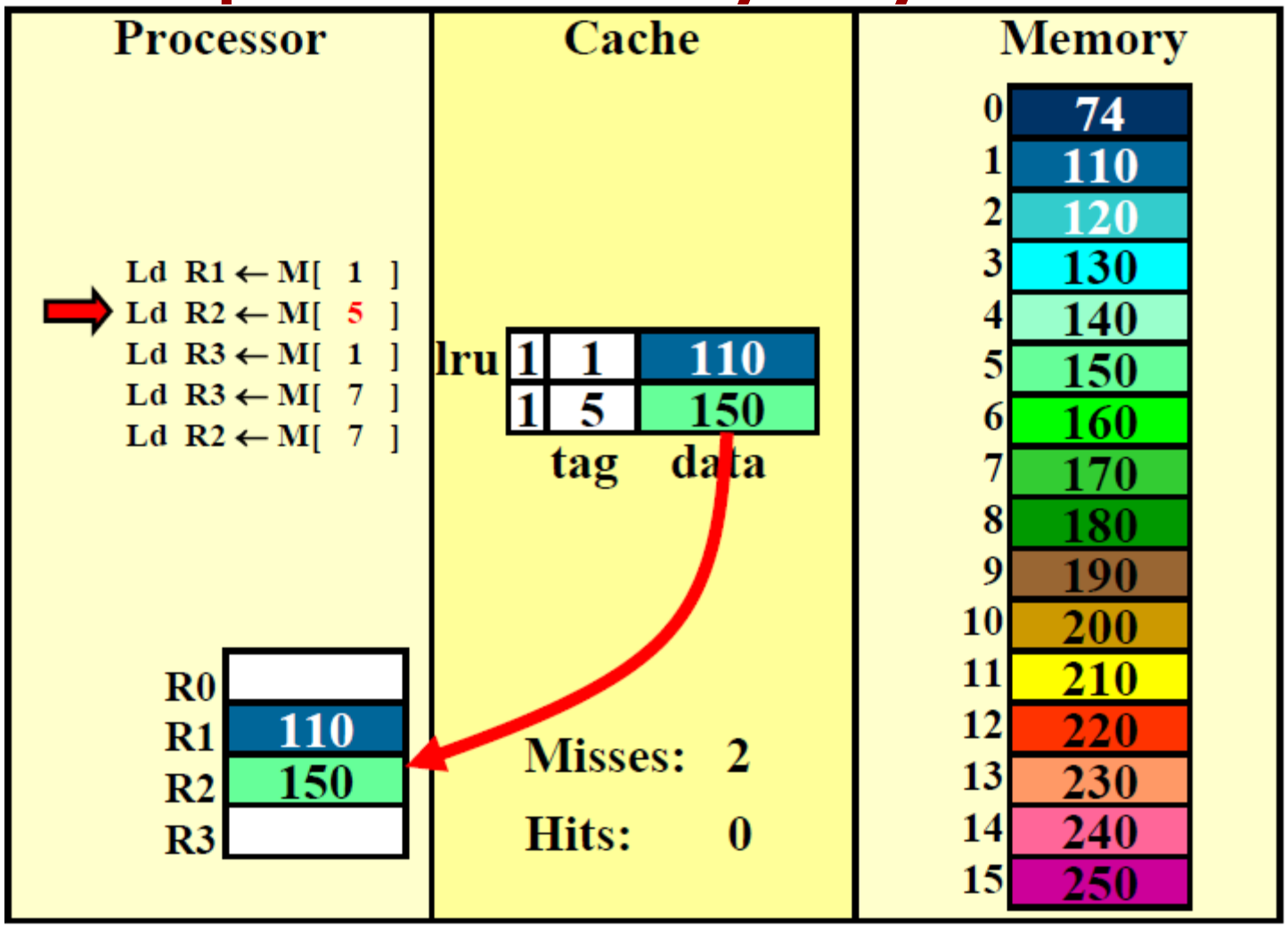
Simple Memory System



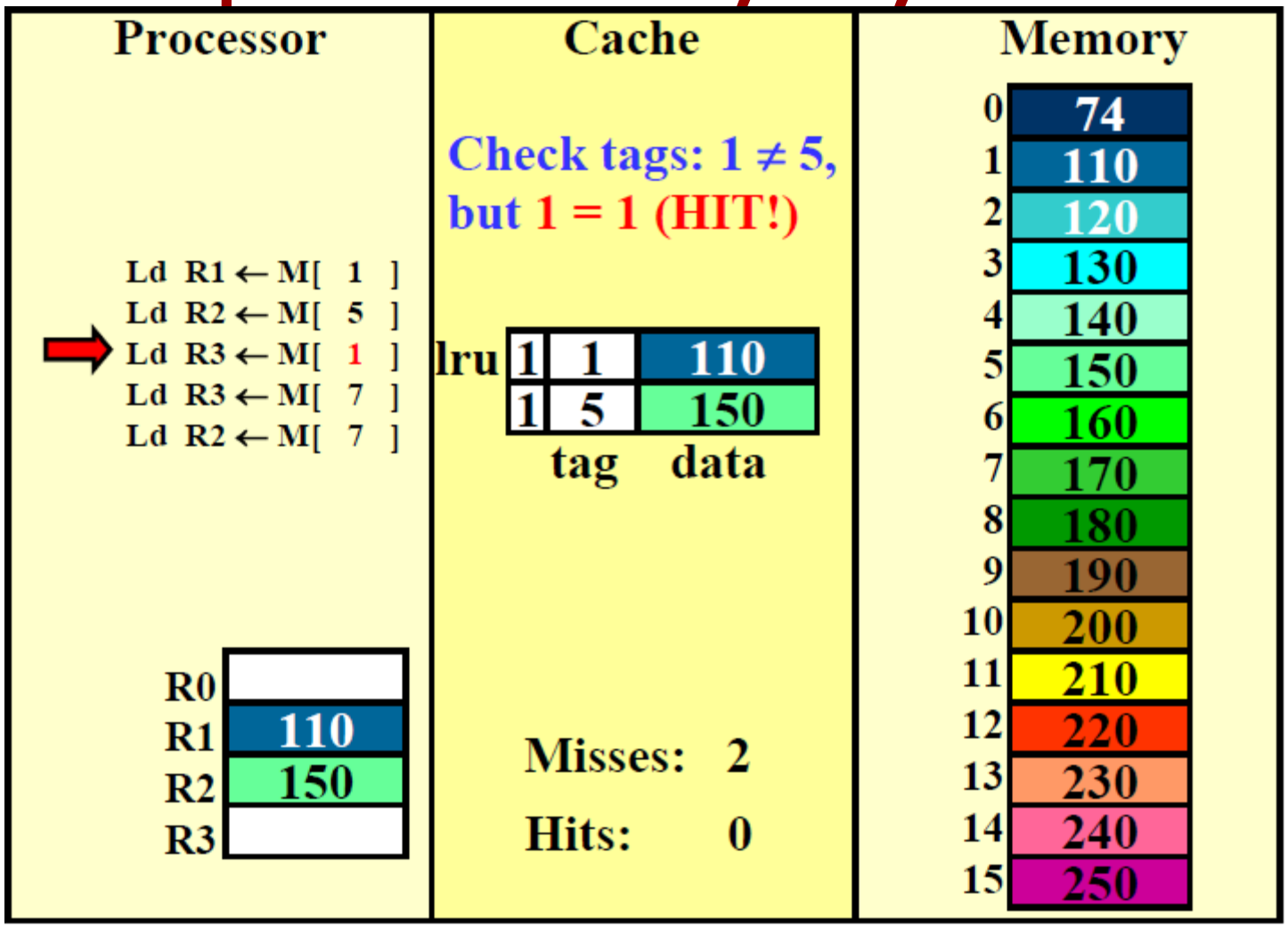
Simple Memory System



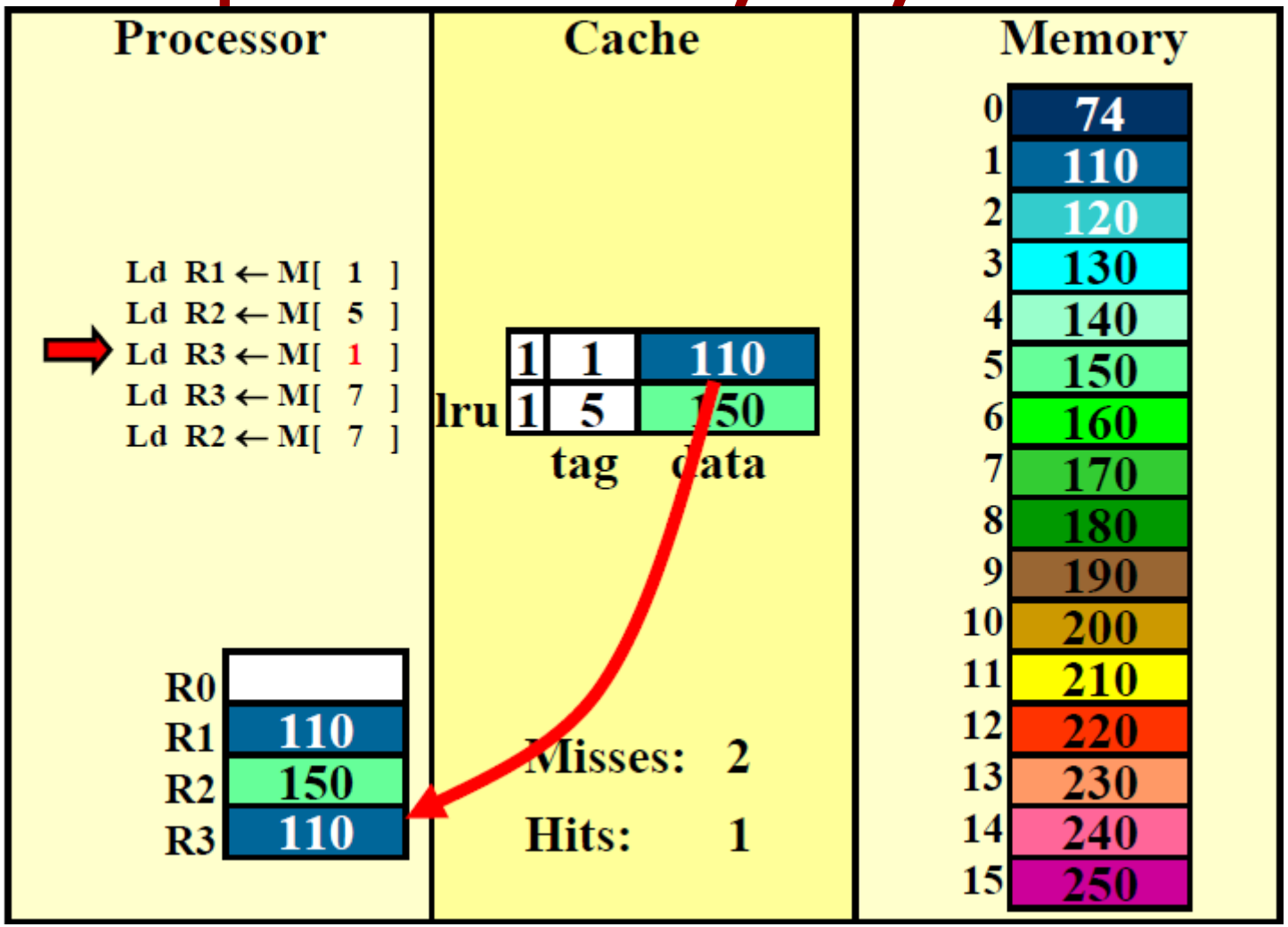
Simple Memory System



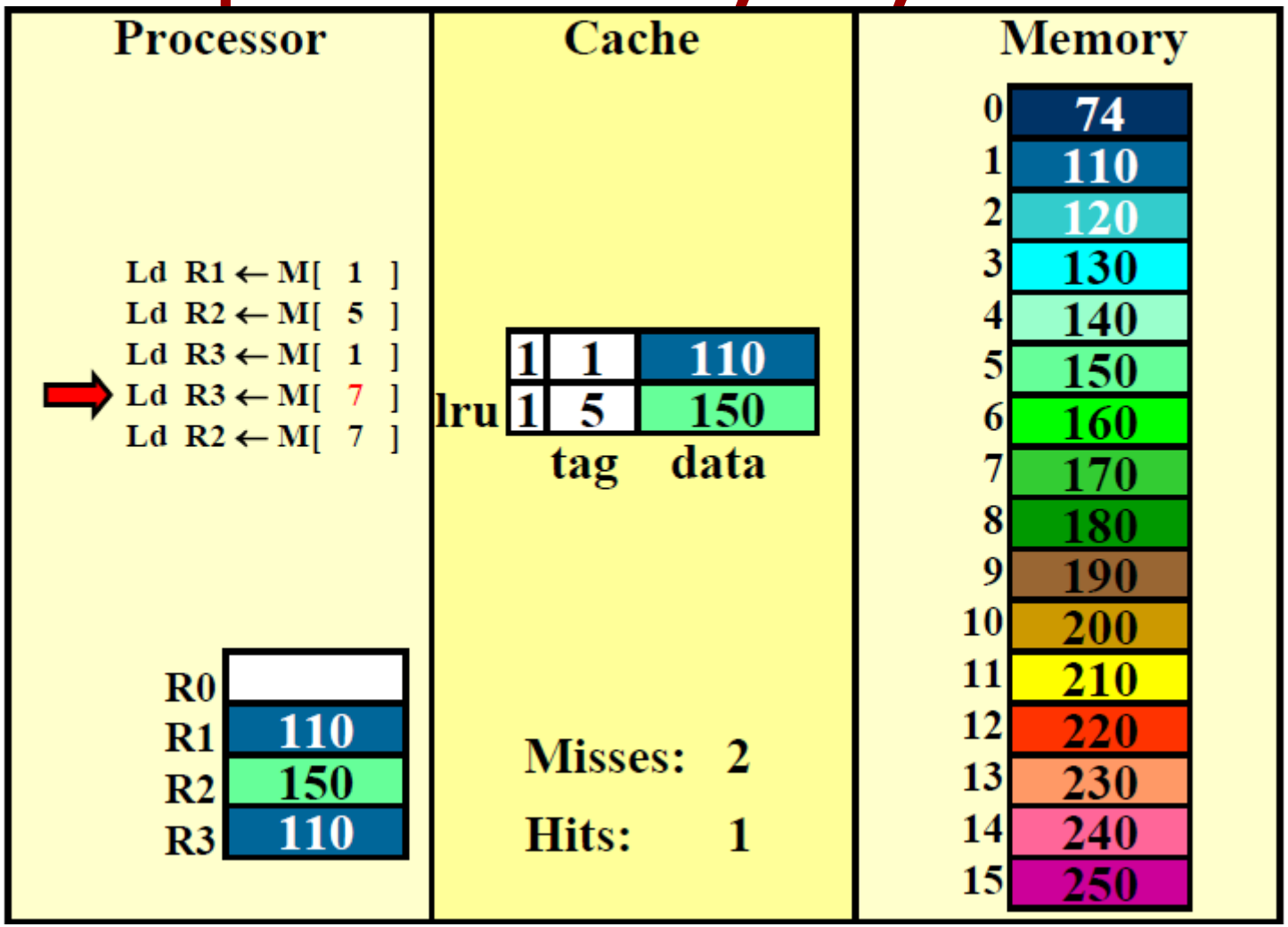
Simple Memory System



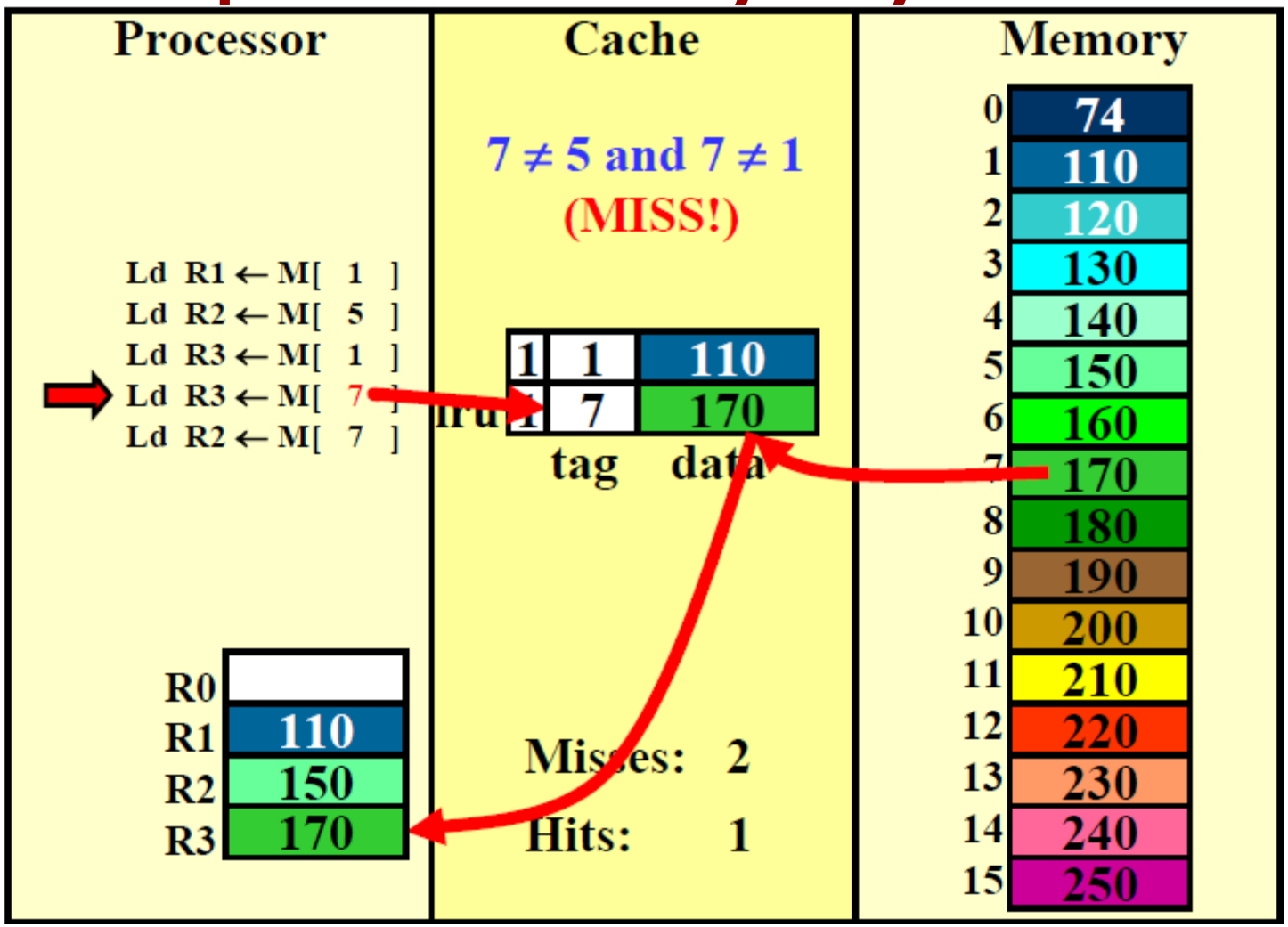
Simple Memory System



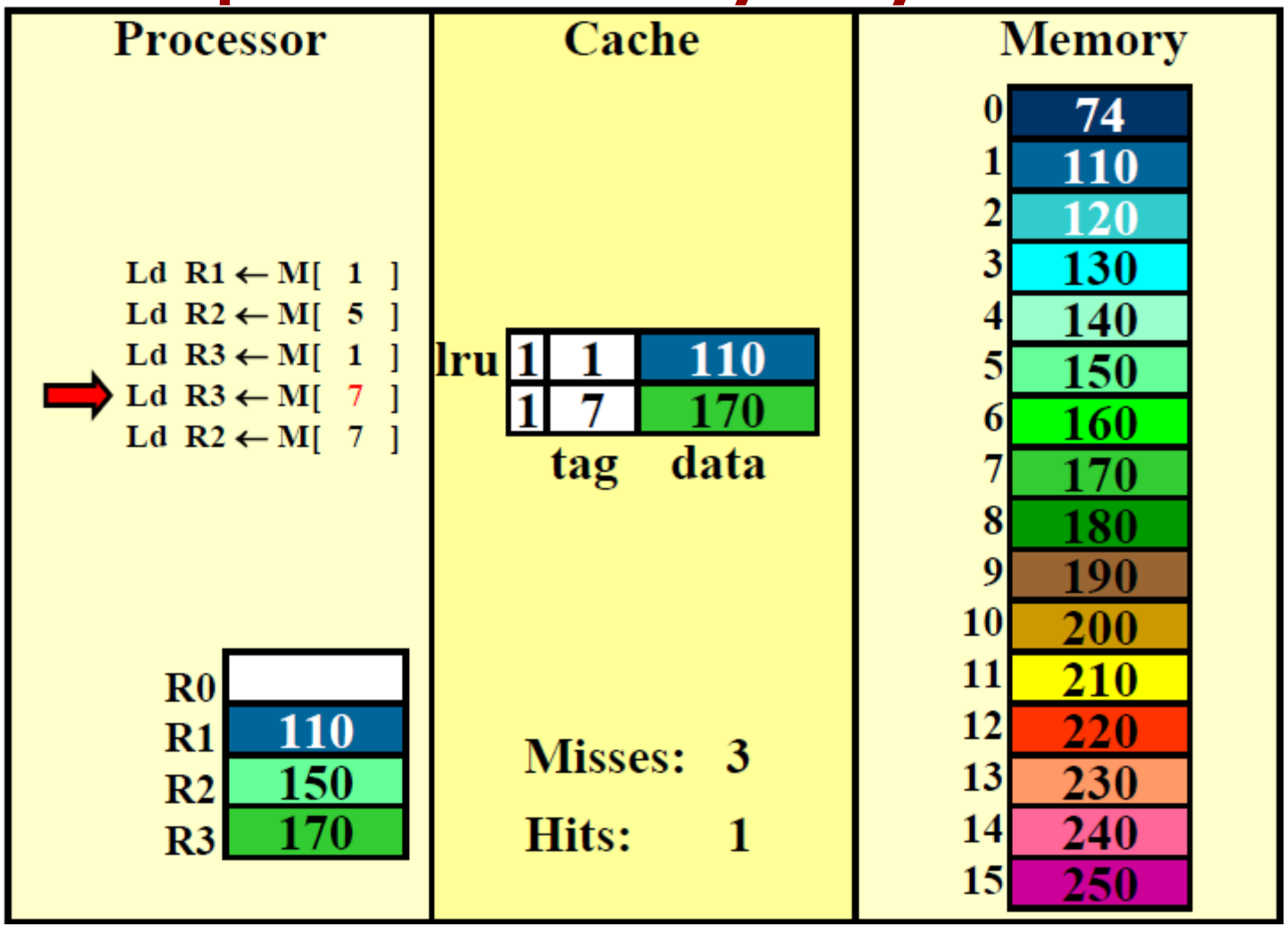
Simple Memory System



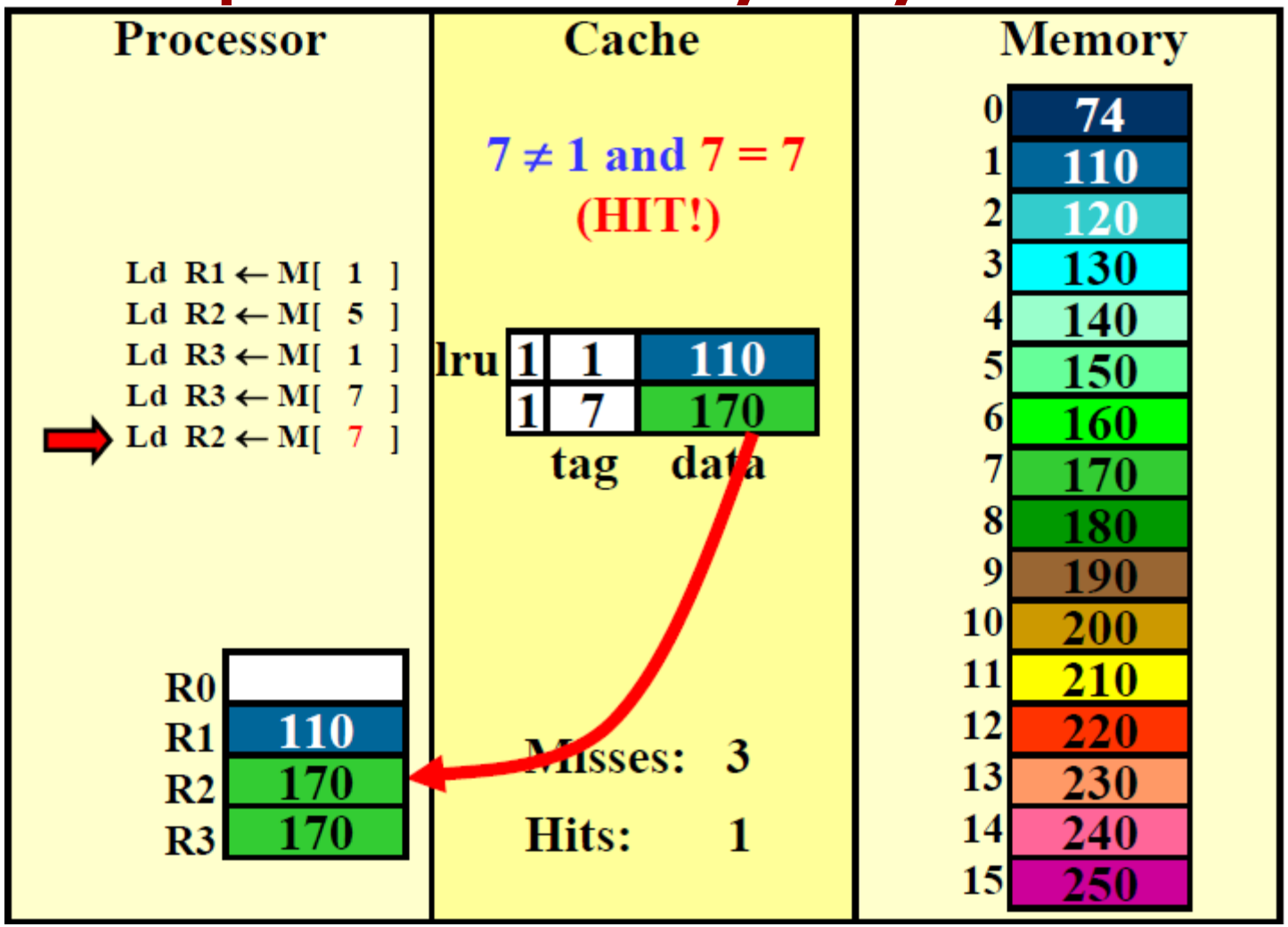
Simple Memory System



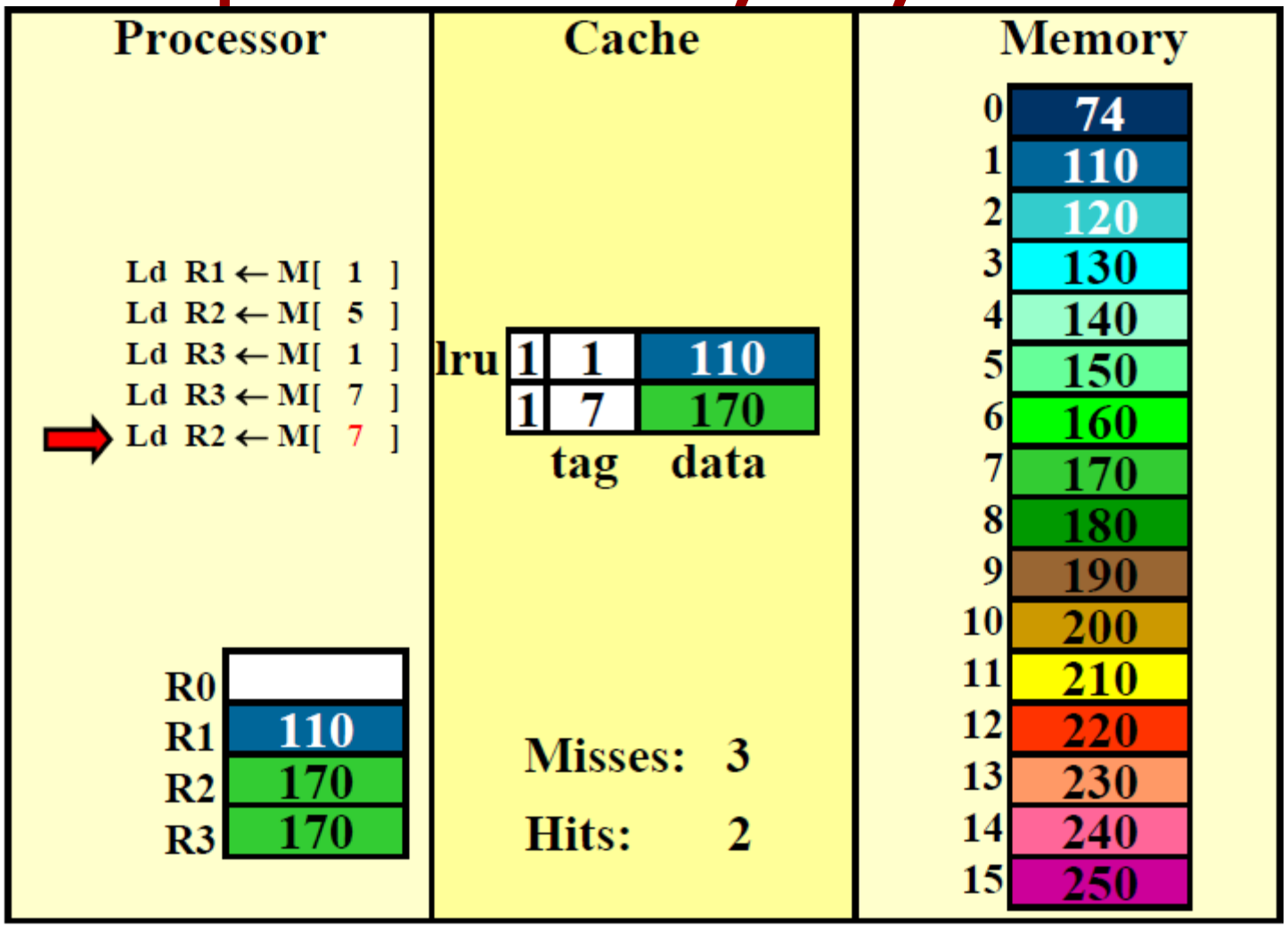
Simple Memory System



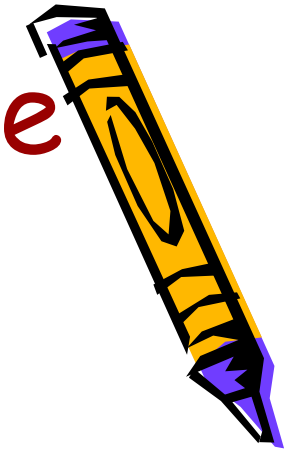
Simple Memory System



Simple Memory System



Reminder: Improving Cache Performance



- AMAT =

  Hit Time + (Miss Rate x Miss Penalty)

- Options to Reduce AMAT
  - Reduce time to hit in cache
    - Use smaller cache size
  - Reduce miss rate
    - Increase cache size
  - Reduce miss penalty
    - Use multi-level cache hierarchy

Cache Hit Time

- Impact on Cycle Time
  - Directly tied to clock rate
  - Increases with cache size
  - Increases with associativity

Sources of Cache Misses

- 3Cs
  - Compulsory
  - Capacity
  - Conflict
- Another source of cache miss
  - Coherence

Sources of Cache Misses

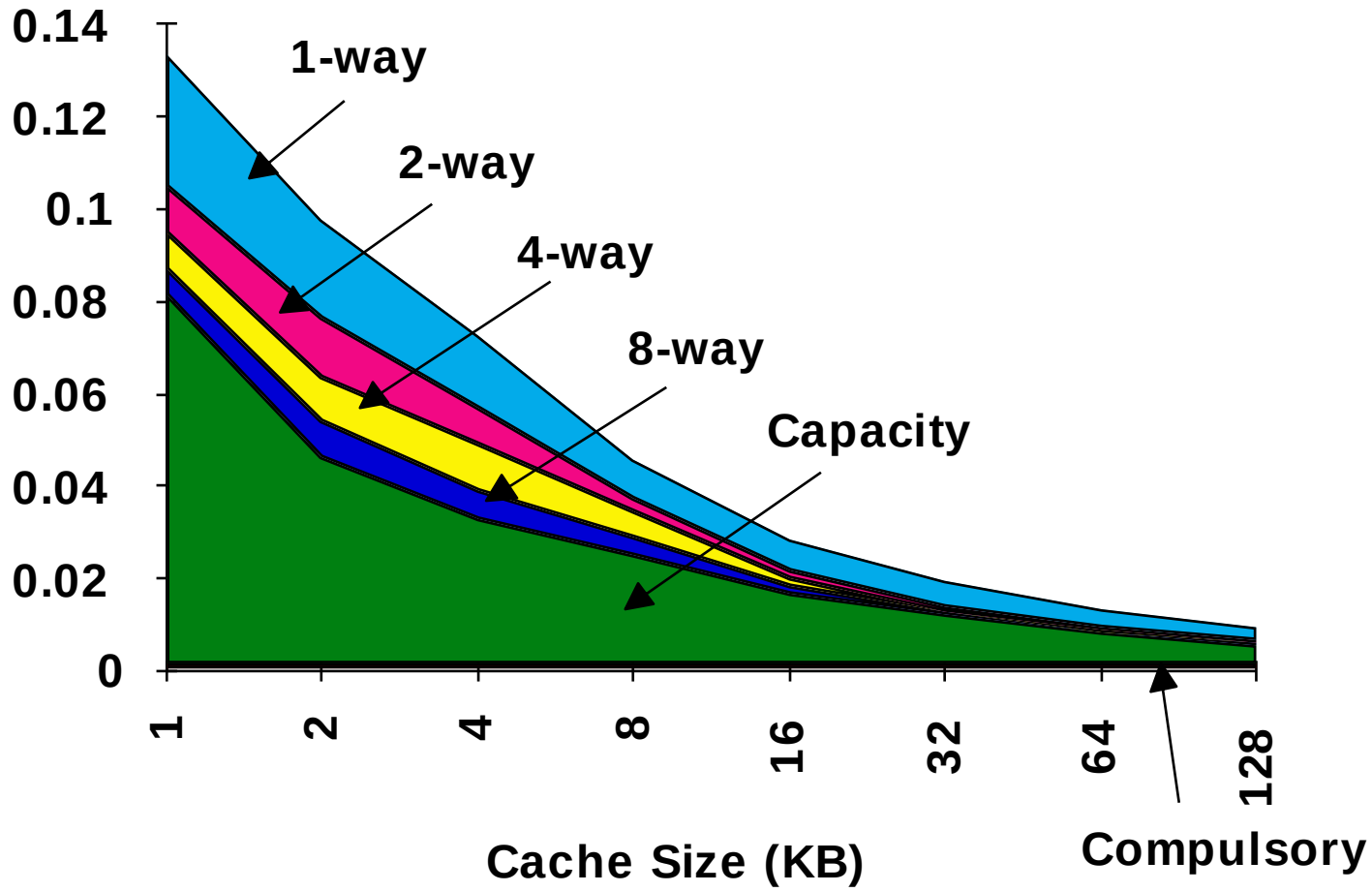
- Compulsory
  - Cold start or process migration
  - First access to a block
  - Compulsory misses are insignificant
    - When running "billions" of instruction
- Capacity
  - Cache cannot contain all blocks accessed by program
  - Solution: increase cache size

Sources of Cache Misses

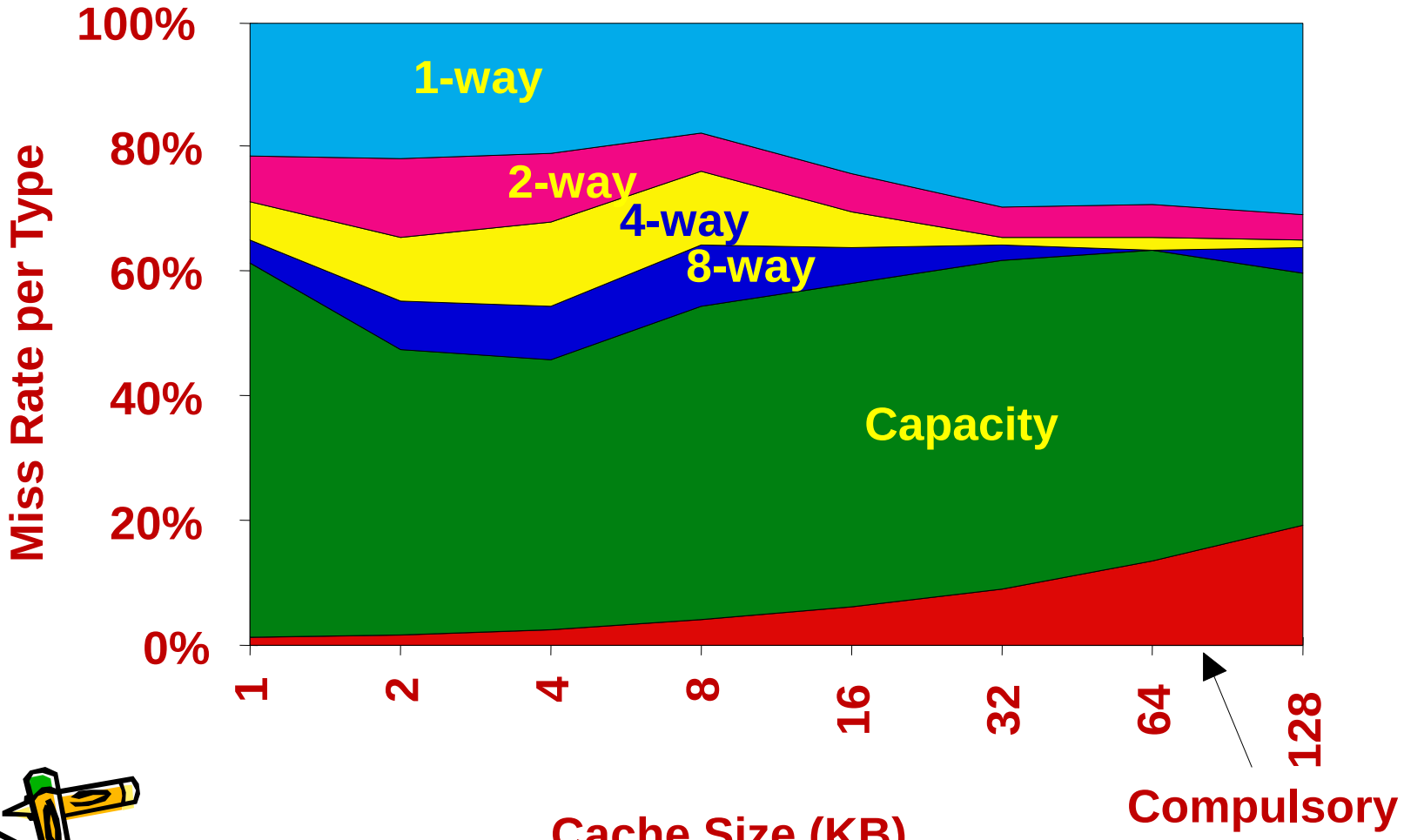
- Conflict (collision)
  - Multiple memory locations mapped to same cache location
  - Solution 1: increase cache size
  - Solution 2: increase associativity

- Coherence (Invalidation)
  - Other processes (e.g., I/O or a core in a CMP) updates memory

3Cs Absolute Miss Rate



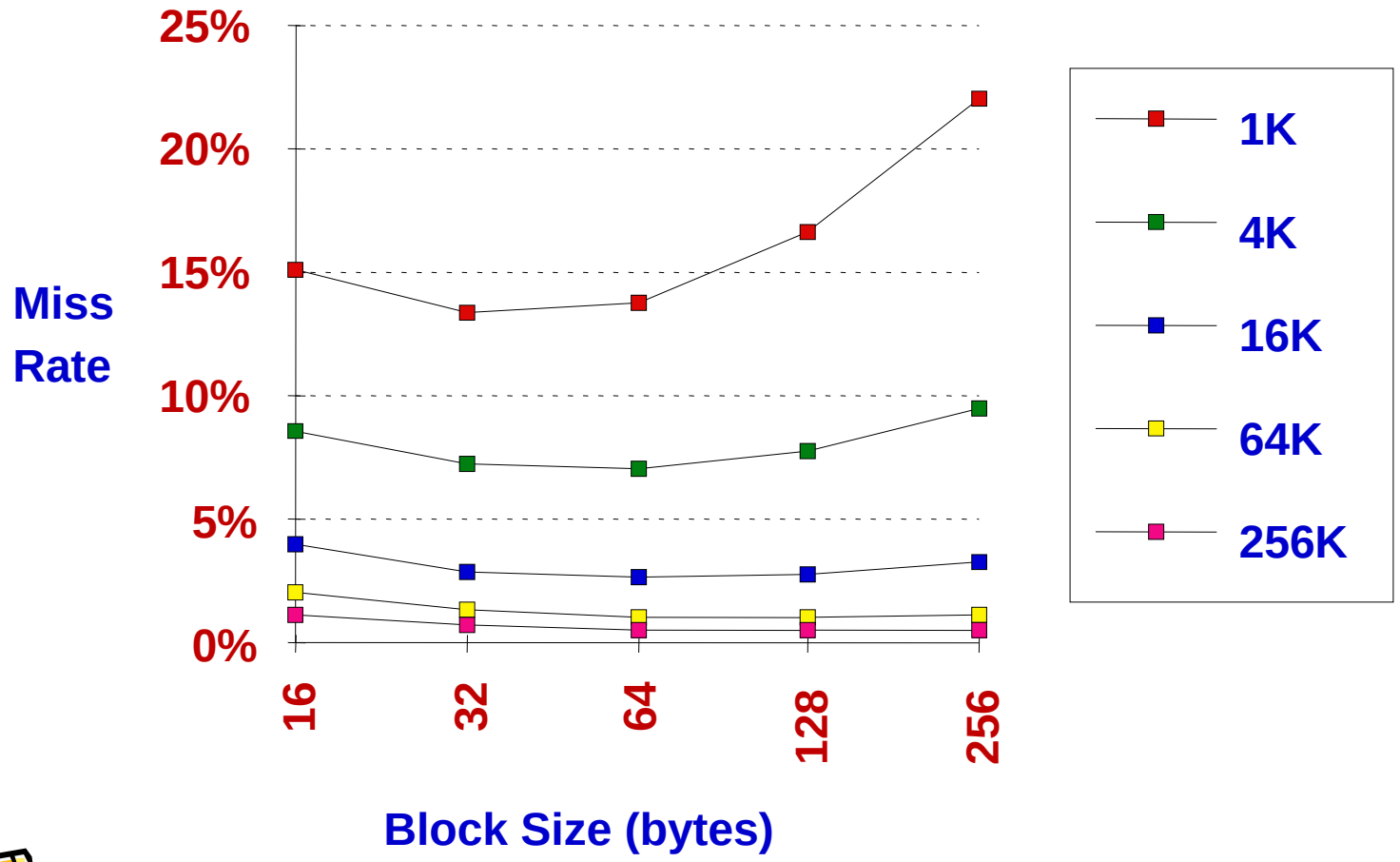
3Cs Relative Miss Rate



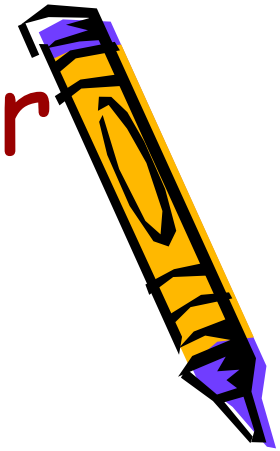
Reducing Miss Rate

- Larger Block Size
- Higher Associativity
- Prefetching
- Complier Optimization

Reducing Misses via Larger Block Size

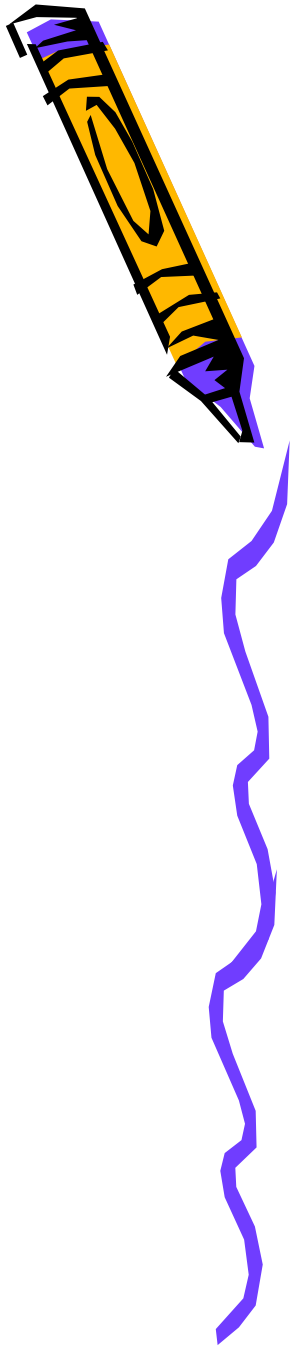


Reducing Misses via Higher Associativity



- 2:1 Cache Rule:
  - Miss Rate DM cache size N = Miss Rate 2-way cache size N/2
- Watch Out
  - Execution time is only final measure!
  - AMAT not always improved by more associativity!

Reducing Misses by Prefetching



- Instruction Prefetching
- Data Prefetching
- HW vs. SW Prefetching

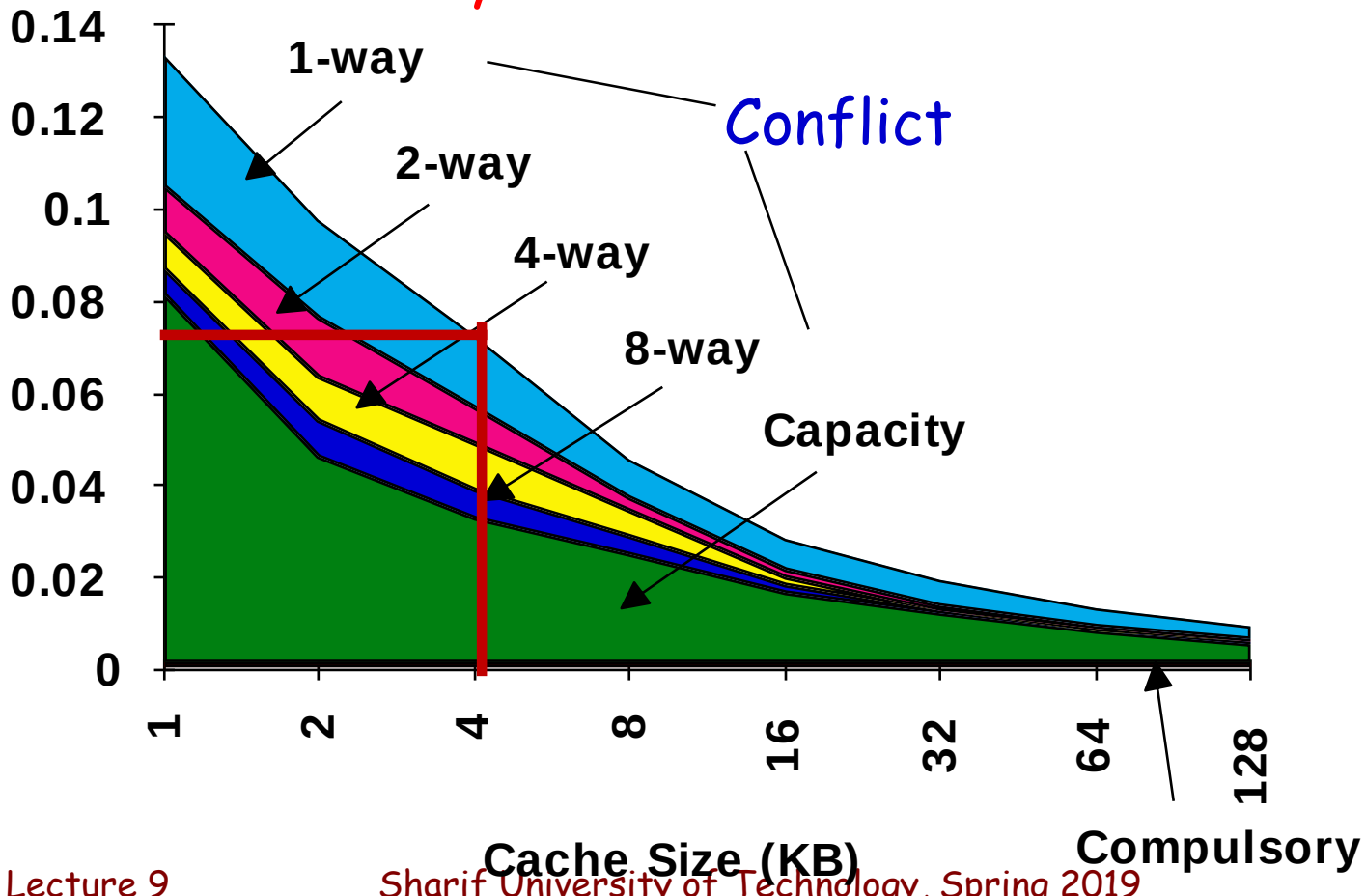
Reducing Miss Penalty

- Faster RAM Technologies
  - Use of faster SRAMs and DRAMs
- More Hierarchy Levels
  - 1-level ➔ 2-level ➔ 3-level
- Read Priority over Write on Miss
  - Reads on critical path

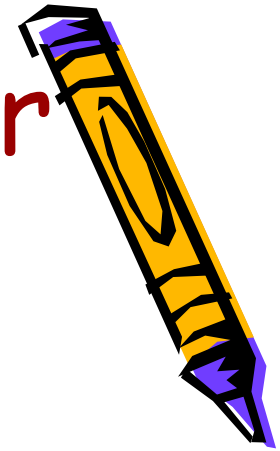
Backup

2:1 Cache Rule

miss rate 1-way associative cache size X
= miss rate 2-way associative cache size X/2



Reducing Misses via Higher Associativity



- 2:1 Cache Rule:
  - Miss Rate DM cache size N = Miss Rate 2-way cache size N/2
- Beware: Execution time is only final measure!
  - Will Clock Cycle time increase?
  - Hill [1988] suggested hit time for 2-way vs. 1-way
    external cache +10%,
    internal + 2%
- AMAT not always improved by more associativity

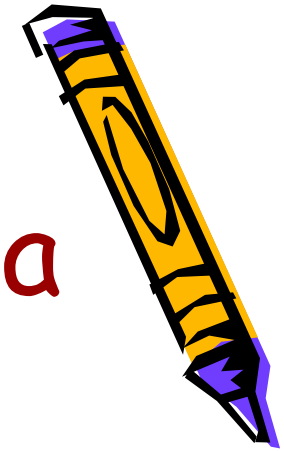
Reducing Misses via a "Victim Cache"

- Question:
  - How to combine fast hit time of direct mapped yet still avoid conflict misses?
- Add buffer to place data discarded from cache, called Victim Cache
- Used in Alpha, HP machines

Reducing Misses by Hardware Prefetching

- E.g., Instruction Prefetching
  - Alpha 21064 fetches 2 blocks on a miss
  - Extra block placed in "stream buffer"
  - On miss check stream buffer
- Prefetching relies on having extra memory bandwidth that can be used without penalty

Reducing Misses by Software Prefetching Data



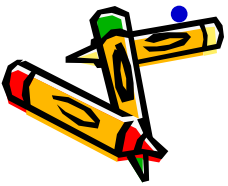
- Data Prefetch
  - Load data into register (HP PA-RISC loads)
  - Cache Prefetch: load into cache (MIPS IV, PowerPC, SPARC v. 9)
  - Special prefetching instructions cannot cause faults; a form of speculative execution

Reducing Misses by Compiler Optimizations

- Instructions
  - Reorder procedures in memory so as to reduce conflict misses
  - Profiling to look at conflicts(using tools they developed)
- Data
  - *Merging Arrays*: improve spatial locality by single array of compound elements vs. 2 arrays
  - *Loop Interchange*: change nesting of loops to access data in order stored in memory
  - *Loop Fusion*: Combine 2 independent loops that have same looping and some variables overlap
  - *Blocking*: Improve temporal locality by accessing "blocks" of data repeatedly vs. going down whole columns or rows

Reducing Penalty: Faster DRAM / Interface

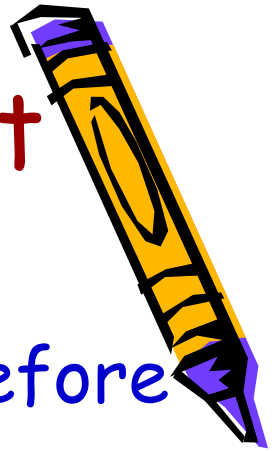
- New DRAM Technologies
  - RAMBUS - same initial latency, but much higher bandwidth
  - Synchronous DRAM
  - TMJ-RAM (Tunneling magnetic-junction RAM) from IBM??
  - Merged DRAM/Logic - IRAM project here at Berkeley
- Better BUS interfaces
- CRAY Technique: only use SRAM



Reducing Penalty

- Read Priority over Write on Miss
  - A Write Buffer Allows Reordering of Requests
    - Processor: writes data into cache and write buffer
    - Memory controller: write contents of buffer to memory
  - Writes go to DRAM only when idle
    - Allows so-called "Read Around Write" capability
    - Why important? Reads hold up the processor, writes do not

Reduce Penalty: Early Restart and Critical Word First

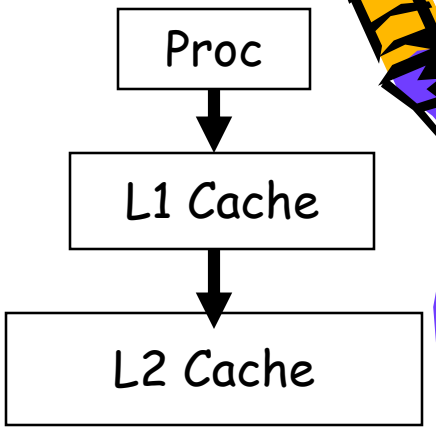


- Don't wait for full block to be loaded before restarting CPU
  - *Early restart*—As soon as the requested word of the block arrives, send it to the CPU and let the CPU continue execution
  - *Critical Word First*—Request the missed word first from memory and send it to the CPU as soon as it arrives; let the CPU continue execution while filling the rest of the words in the block. Also called *wrapped fetch* and *requested word first*
  - *DRAM FOR LAB 6 can do this in burst mode! (Check out sequential timing)*
- Generally useful only in large blocks,
- Spatial locality a problem; tend to want next sequential word, so not clear if benefit by early restart

Reduce Penalty: Non-blocking Caches

- *Non-blocking cache* or *lockup-free cache* allow data cache to continue to supply cache hits during a miss
  - requires F/E bits on registers or out-of-order execution
  - requires multi-bank memories
- "*hit under miss*" reduces the effective miss penalty by working during miss vs. ignoring CPU requests
- "*hit under multiple miss*" or "*miss under miss*" may further lower the effective miss penalty by overlapping multiple misses
  - Significantly increases the complexity of the cache controller as there can be multiple outstanding memory accesses
  - Requires muliple memory banks (otherwise cannot support)

Reduce Penalty: Second-Level Cache



- L2 Equations

$AMAT = Hit\ Time_{L1} + Miss\ Rate_{L1} \times Miss\ Penalty_{L1}$

$Miss\ Penalty_{L1} = Hit\ Time_{L2} + Miss\ Rate_{L2} \times Miss\ Penalty_{L2}$

$AMAT = Hit\ Time_{L1} +$
$\underline{Miss\ Rate_{L1}} \times (Hit\ Time_{L2} + \underline{Miss\ Rate_{L2}} \times Miss\ Penalty_{L2})$

- Definitions:
  - *Local miss rate*— misses in this cache divided by the total number of memory accesses *to this cache* (Miss $rate_{L2}$)
  - *Global miss rate*—misses in this cache divided by the total number of memory accesses *generated by the CPU*

Reducing Misses: which apply to L2 Cache?

- Reducing Miss Rate by:
  - Larger Block Size
  - Higher Associativity
  - Victim Cache
  - HW prefetching data/instruction
  - SW prefetching data
  - Compiler optimizations

Store Buffer

Locality

Virtual Memory

- Problems
  - Program might want more memory than we have
  - Two programs might want to use the same locations
- Solution
  - Break physical and virtual memory into chunks.
  - For each process provide a map of virtual chunks (called "pages") into physical chunks
  - If we don't have enough space in physical memory, allocate chunks on disk.

Virtual Memory: Issues

- If the data you need is on disk
  - Swap the page needed with an "old" page in memory. (***Page fault)***
- Processor generates virtual addresses, memory is addressed with physical addresses.
  - Need to translate (**TLB** or **OS**)
  - Need to figure out where to put the TLB.

Pages

- What we want is each of a processes virtual addresses to map to a physical address.
  - Could map each individual address to a physical one but that gets complex
  - Map chunks of ~4KB.
    - In that case the "offset" part of the address is 12 bits.
    - In a 32-bit address space that means we have 20 bit page numbers.
      - I'll assume virtual and physical space is the same size for now.

Page Table

- A page table simply has a mapping of each virtual page to a physical page.
  - This is kept in memory.
    - Scary eh?
  - Think of it as a black box.
    - You insert a virtual page number (20 bits in our example) and out pops a physical page number.
    - It is in software
      - Lots of potential complexity, don't worry about it.

Virtual Address Translation

- Say Virtual address is 0x12345678
  - Offset (if 4KB page) is 0x678
  - Virtual page number is 0x12345
- Ask the OS what the physical page number is
  - It may be that the data *isn't* in memory.
    - Need to get it from disk: *page fault*
  - Gives you a physical page number
    - *After* the page fault if there is one.

So for each memory access…

- Don't we need to do a translation for each memory access?
  - We don't want to have to get the OS involved
    - And even if we could do it quickly we'd need to look in the memory twice for each memory access.
  - Answer?
    - Cache it
      - TLB.

Other fun Virtual memory problems

- Self-modifying code.
  - Can use the I-TLB to detect that a store is going to a location we think is code.
    - This implies the I-TLB covers the whole of the Icache. This can be fun to get right.
  - If store hits the I-TLB assume we have self-modifying code
    - Nuke.

Backup