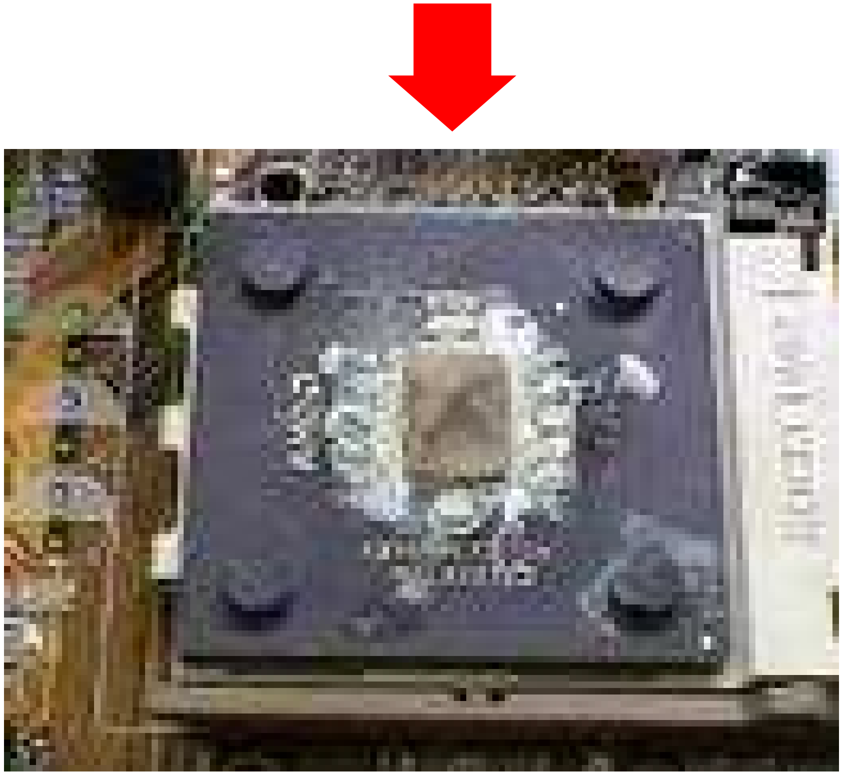
Lecture 5:
Memory Hierarchy

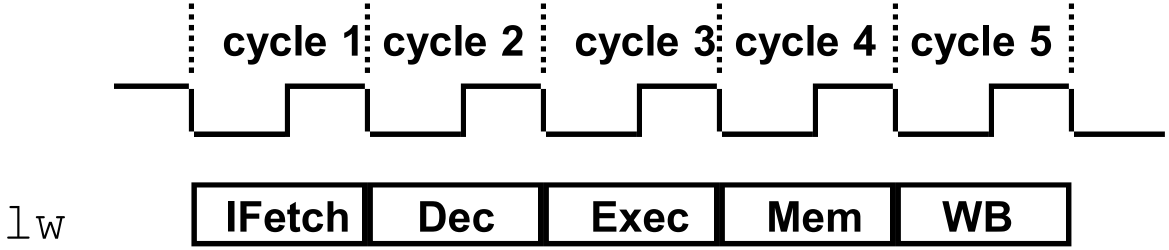
Some slides adapted from Mary Jane Irwin at Penn State University for *Computer Organization and Design*, Patterson & Hennessy, © 2005

Watch this

Click the chip



Review: Instruction Type vs # of Required Cycles



cycle 1 cycle 2 cycle 3 cycle 4 cycle 5

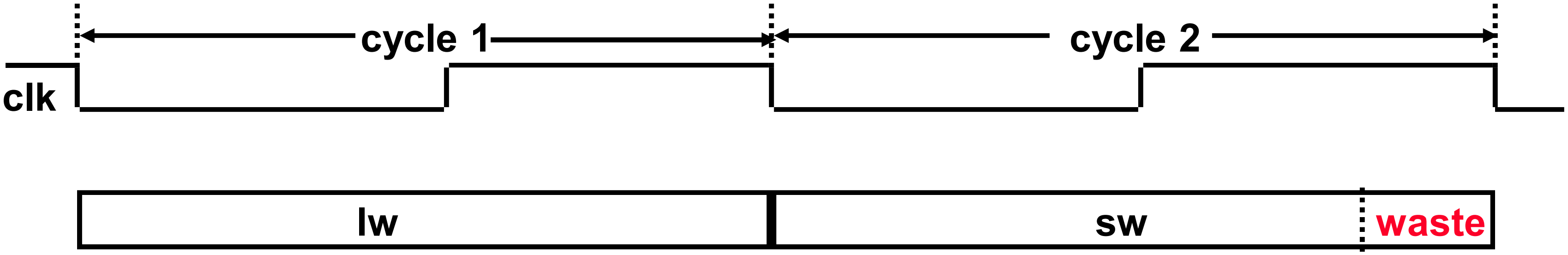
`lw` IFetch Dec Exec Mem WB

- IFetch: Instruction Fetch and Update PC
- Dec: Instruction Decode, Register Read, Sign Extend Offset
- Exec: Execute R-type; Calculate Memory Address; Branch Comparison; Branch and Jump Completion
- Mem: Memory Read; Memory Write Completion; R-type Completion (RegFile write)
- WB: Memory Read Completion (RegFile write)

INSTRUCTIONS TAKE FROM 3 - 5 CYCLES!

Review: Single Cycle Pros & Cons

- uses the clock cycle inefficiently – the clock cycle must be timed to accommodate the slowest instruction
  - especially problematic for more complex instructions like floating point multiply



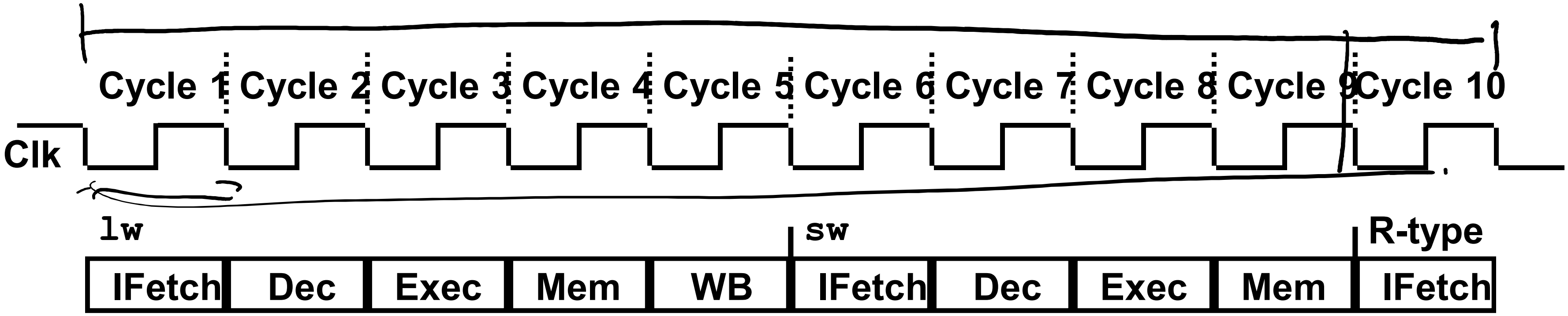
- may be wasteful of area since some functional units (e.g., adders) must be duplicated since they can not be shared during a clock cycle

but

- is simple and easy to understand

Review: Multicycle Pros & Cons

- uses the clock cycle efficiently – the clock cycle is timed to accommodate the slowest instruction step

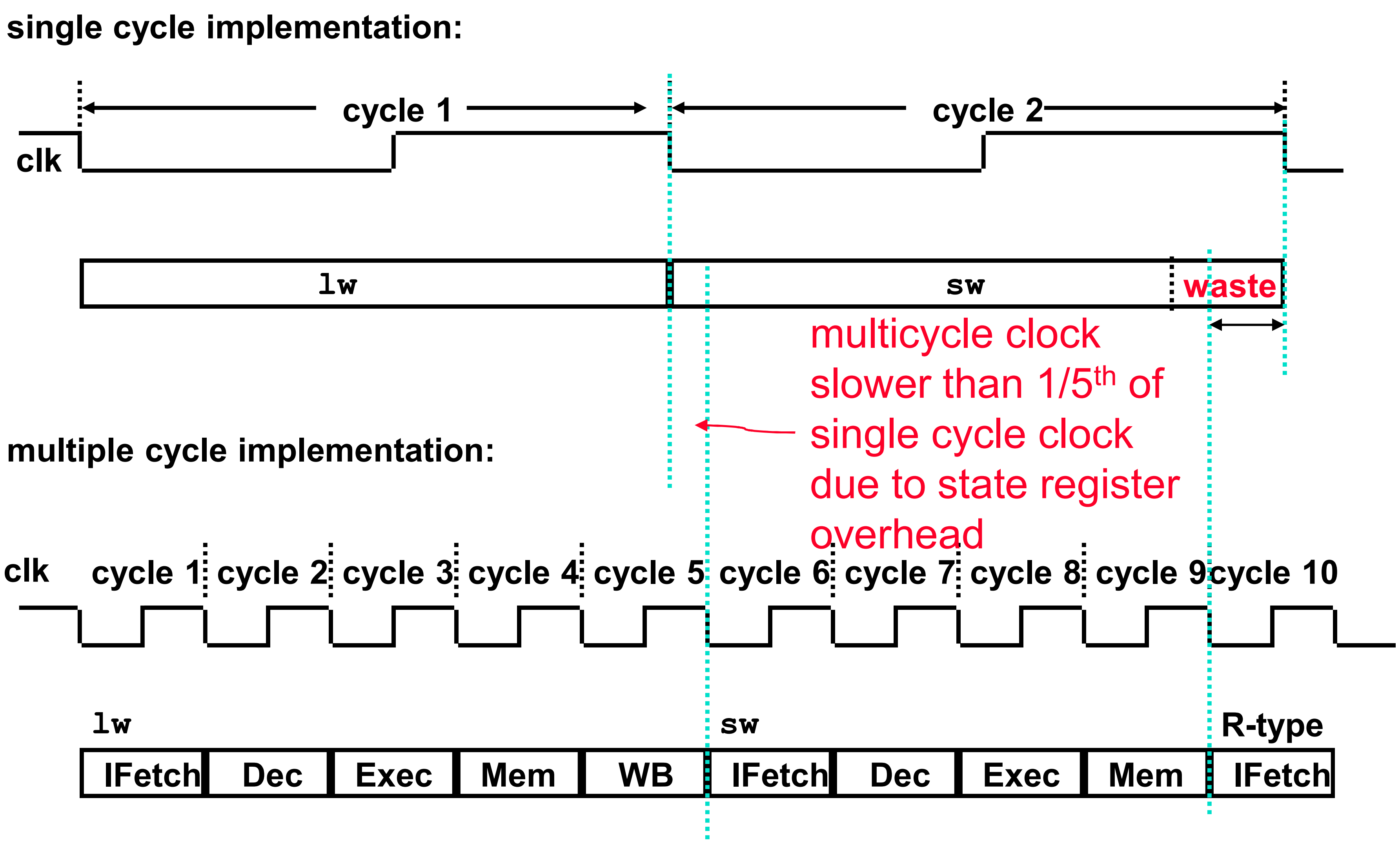


- multicycle implementations allow functional units to be used more than once per instruction as long as they are used on different clock cycles

but

- requires additional internal state registers, more muxes, and more complicated (FSM) control

Review: Single Cycle vs. Multiple Cycle Timing



multicycle clock slower than 1/5th of single cycle clock due to state register overhead

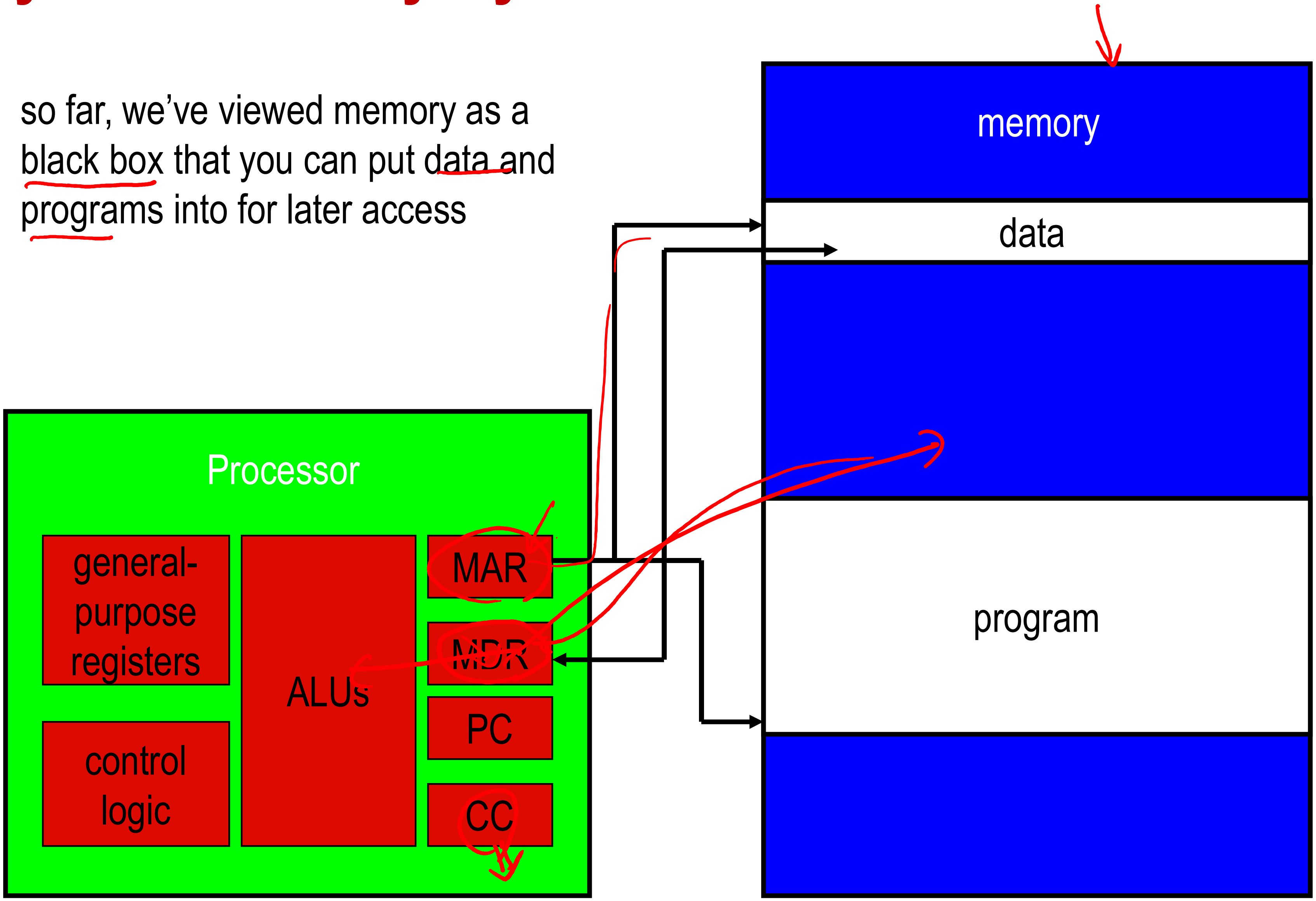
Review: Will multicycle design be faster?

- ❑ let’s assume t_{setup} + t_{cq} time for registers = 0.1 ns
  - single cycle design:
    - clock cycle time = 4.7 + 0.1 = 4.8 ns
    - time/inst = 1 cycle/inst × 4.8 ns/cycle = 4.8 ns/inst
  - multicycle design:
    - clock cycle time = 1.0 + 0.1 = 1.1
    - time/inst = CPI × 1.1 ns/cycle (depends on the types or mixture of instructions!)

	I Fetch	Decode, R-Read	ALU	PC update	D Memory	R-Write	Total (ns)
Add	1	1	.9	-	-	.8	3.7
Load	1	1	.9	-	1	.8	4.7
Store	1	1	.9	-	1	-	3.9
beq	1	1	.9	.1	-	-	3.0

Physical Memory Systems

so far, we've viewed memory as a black box that you can put data and programs into for later access



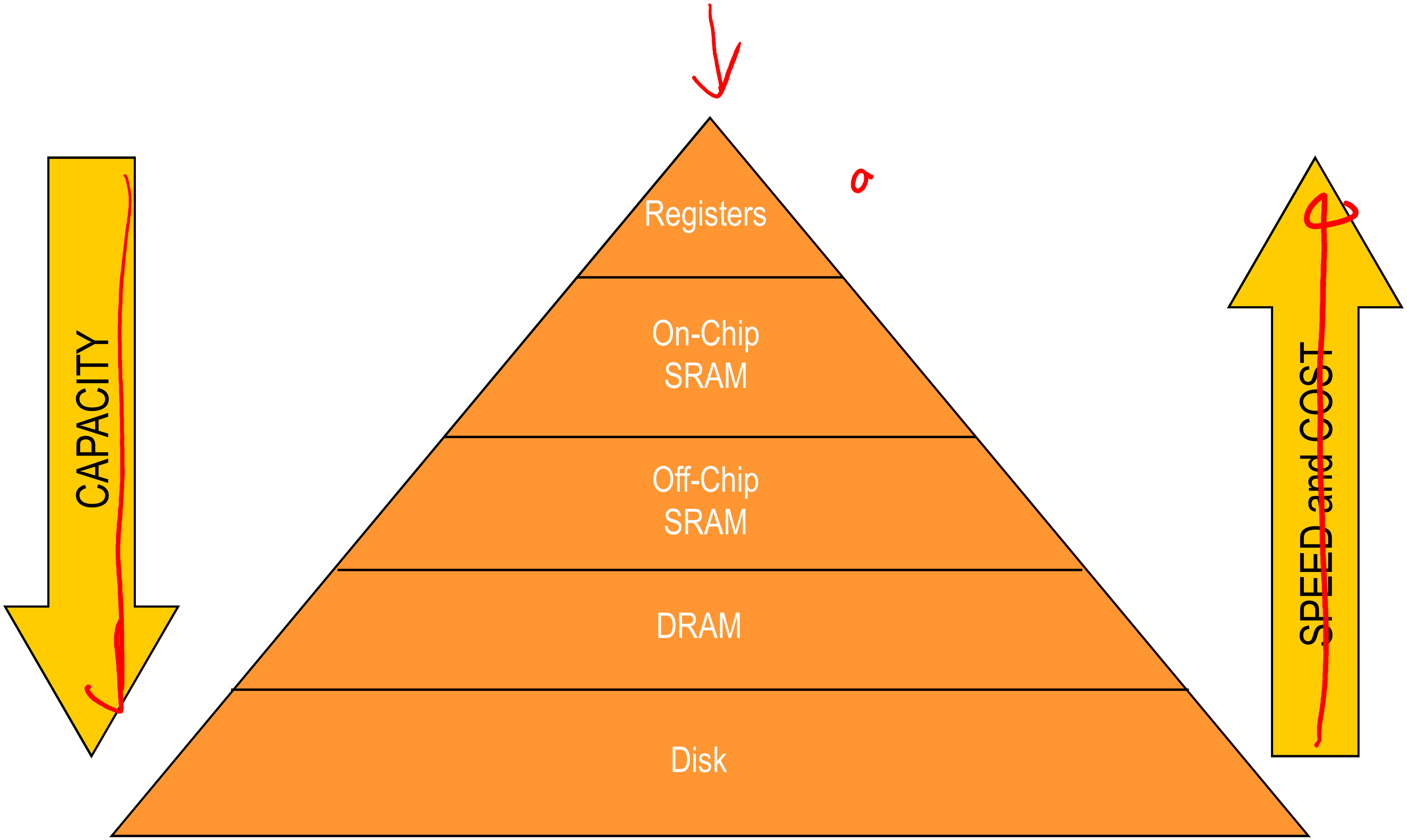
Types of Memories

- SSD: $0.75 /GB, HD: $0.1-0.2/GB
- DRAM: $20-25/GB
- more on bandwidth later

Need understand the trade-off among size, speed, cos

Type	Size	Latency	Cost/bit
Register	< 1KB 1ns	< 1ns	$$$$
On-chip SRAM	8KB-6MB 2ns	< 2ns	$$$
Off-chip SRAM	1Mb – 16Mb 10ns	< 10ns	$$
DRAM	64MB – 1TB 100ns	< 100ns	$
Disk (SSD, HD)	40GB – 1PB 20m	< 20ms	< $1/GB

Memory Hierarchy



Why Does a Hierarchy Work?

- locality of reference
  - ✓ temporal locality
    - ○ reference same memory location many times (close together, in time)
  - ✓ spatial locality
    - ○ reference near neighbors around the same time

Example of Locality

```
int A[100], B[100], C[100], D;
for (i=0; i<100; i++) {
        C[i] = A[i] * B[i] + D;
}
```

▨	▨	▨	D	C[99]	C[98]	C[97]	C[96]
. . . .							
C[7]	C[6]	C[5]	C[4]	C[3]	C[2]	C[1]	C[0]
. . . .							
B[11]	~~B[10]~~	~~B[9]~~	~~B[8]~~	~~B[7]~~	~~B[6]~~	B[5]	B[4]
B[3]	~~B[2]~~	~~B[1]~~	~~B[0]~~	A[99]	A[98]	A[97]	A[96]
. . . .							
A[7]	~~A[6]~~	~~A[5]~~	~~A[4]~~	~~A[3]~~	~~A[2]~~	~~A[1]~~	A[0]

8words

a cache line (one fetch)

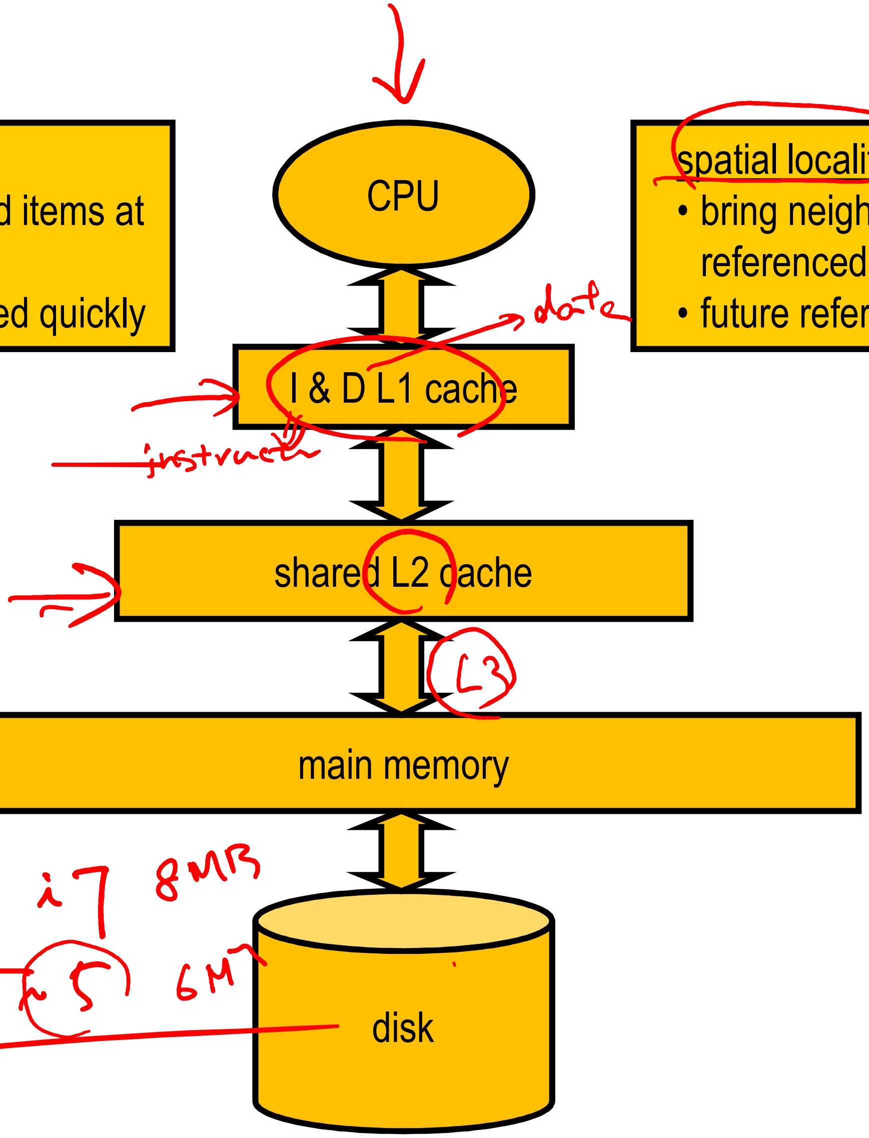
Memory Hierarchy

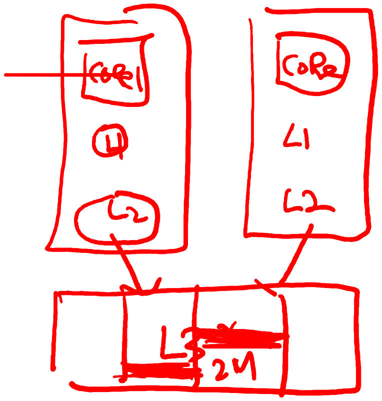
temporal locality

- keep recently referenced items at higher levels
- future references satisfied quickly

spatial locality

- bring neighbors of recently referenced to higher levels
- future references satisfied quickly

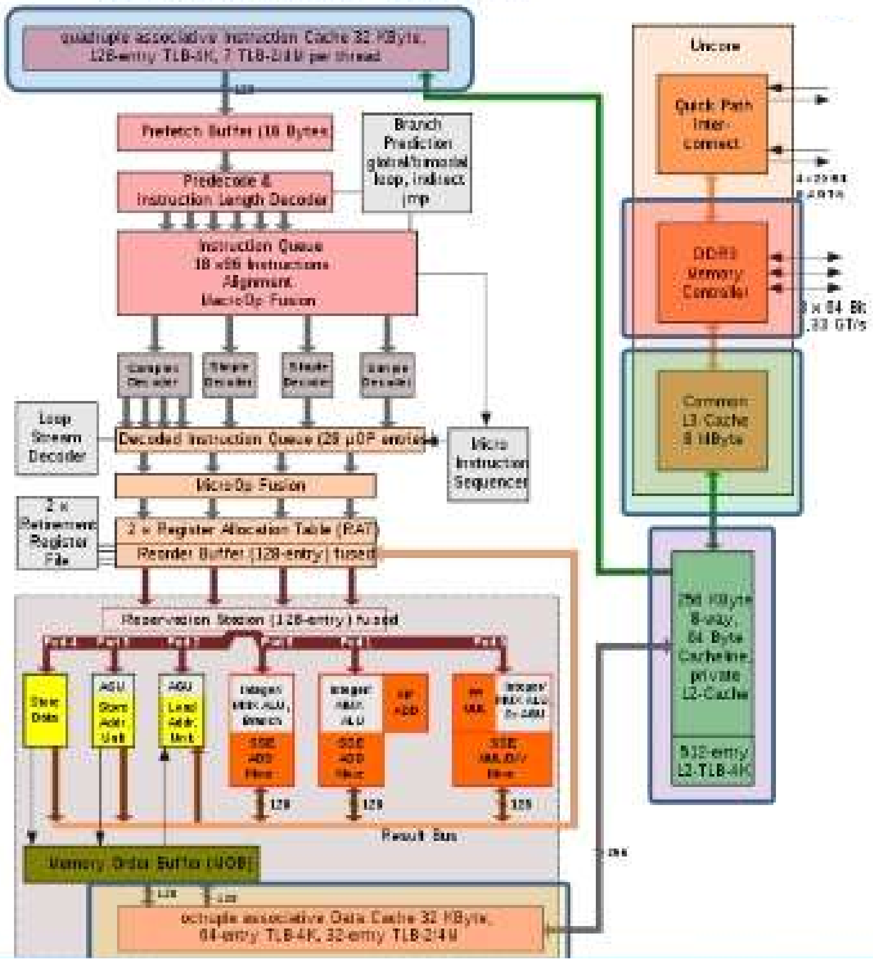




4-core intel i7 8MB

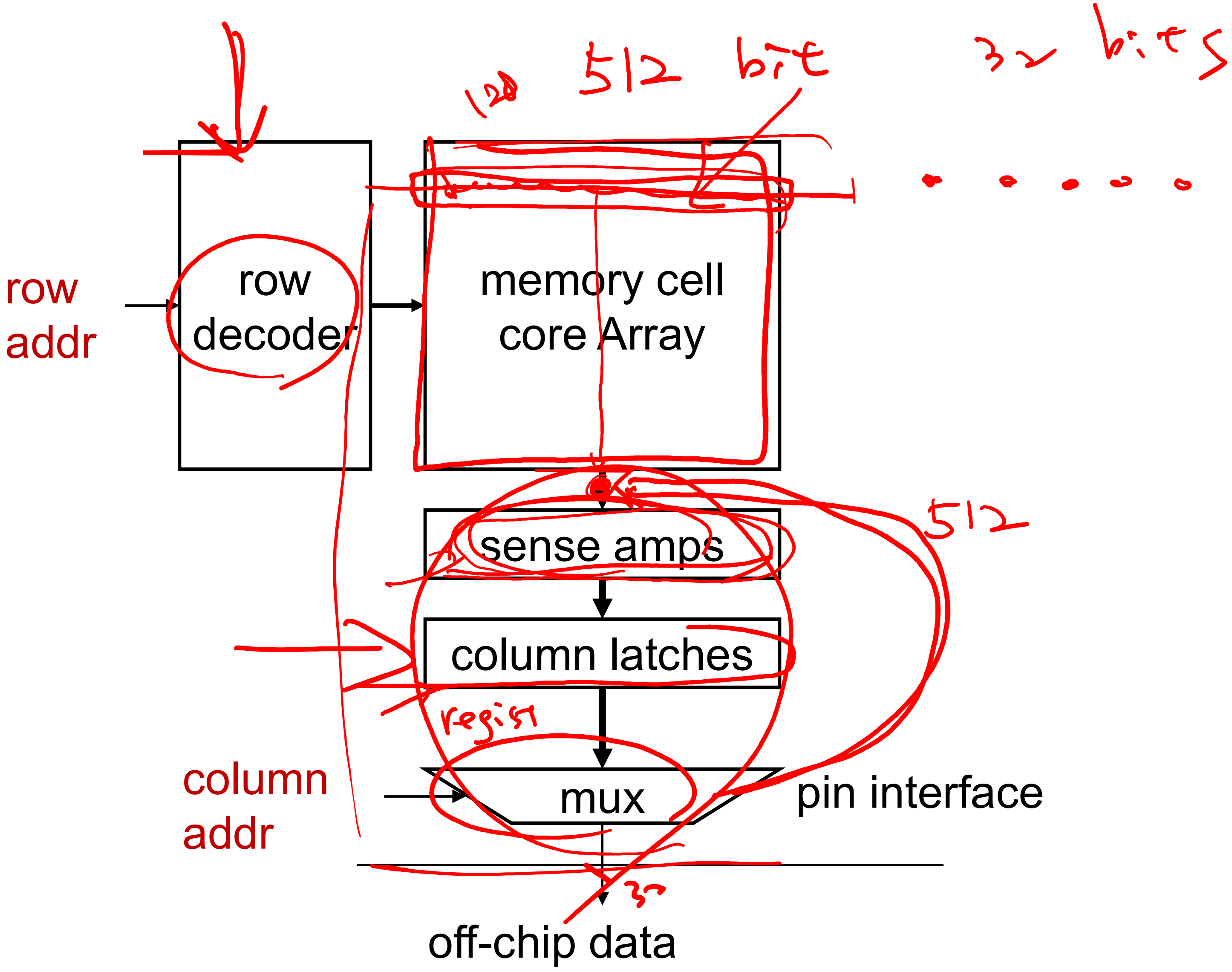
" " i5 6M

Example: Intel Nehalem Memory Hierarchy

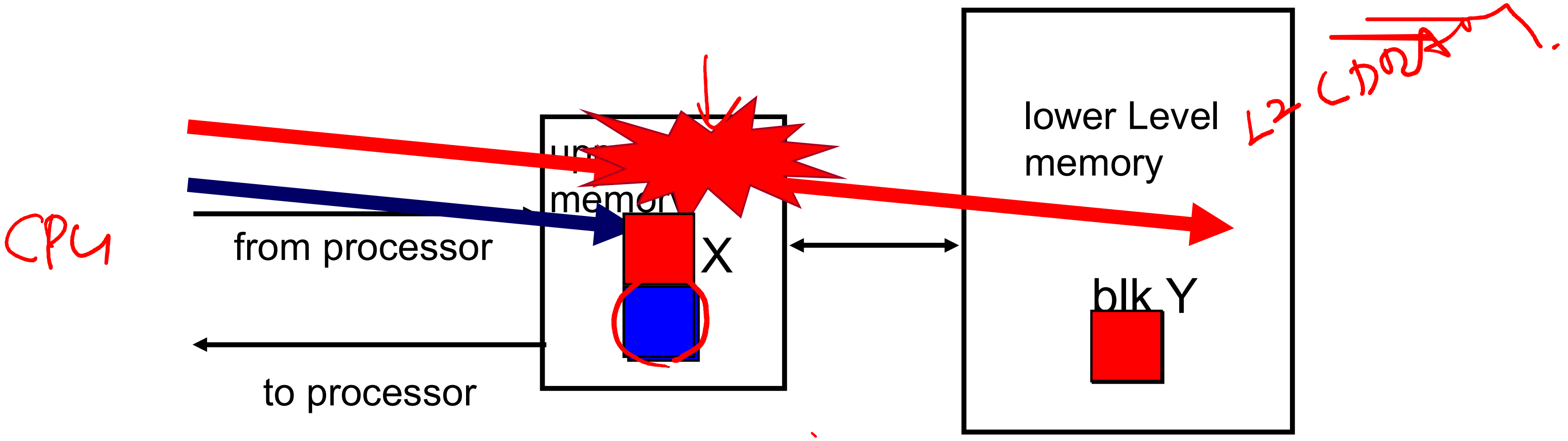

quadruple associative Instruction Cache 32 KByte, 128-entry TLB-4K, 7 TLB-2/4M per thread
128
Prefetch Buffer (16 Bytes)
Branch Prediction global/bimodal, loop, indirect jmp
Predecode & Instruction Length Decoder
Instruction Queue 18 x86 Instructions Alignment MacroOp Fusion
Complex Decoder
Simple Decoder
Simple Decoder
Simple Decoder
Loop Stream Decoder
Decoded Instruction Queue (28 µOP entries)
Micro Instruction Sequencer
MicroOp Fusion
2 x Retirement Register File
2 x Register Allocation Table (RAT)
Reorder Buffer (128-entry) fused
Reservation Station (128-entry) fused
Port 4
Port 3
Port 2
Port 0
Port 1
Port 5
Store Data
AGU Store Addr. Unit
AGU Load Addr. Unit
Integer/ MMX ALU, Branch
SSE ADD Move
Integer/ MMX ALU
SSE ADD Move
FP ADD
FP MUL
Integer/ MMX ALU, 2x AGU
SSE MUL/DIV Move
128
128
128
Result Bus
Memory Order Buffer (MOB)
128
128
octruple associative Data Cache 32 KByte, 64-entry TLB-4K, 32-entry TLB-2/4M
256
Uncore
Quick Path Inter-connect
4 x 20 Bit 6.4 GT/s
DDR3 Memory Controller
3 x 64 Bit 1.33 GT/s
Common L3 Cache 8 MByte
256 KByte 8-way, 64 Byte Cacheline, private L2-Cache
512-entry L2-TLB-4K

1. 4-way set associative instruction cache
2. 8-way set associative L1 data cache (32 KB)
3. 8-way set associative L2 data cache (256 KB)
4. 16-way shared L3 cache (8 MB)
5. 3 DDR3 memory connections

Typical Memory Organization



Basic Cache Operation



Cache Terminology

- hit: data appears in some block
  - ✓ hit rate: the fraction of accesses found in the level → 90%
  - ✓ hit time: time to access the level (consists of RAM access time + time to determine hit)
- miss: data needs to be retrieved from a block in the lower level (e.g., 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

"1"   10 – 20 ~ any hundred ~~cycle~~.

Average Memory Access Time

- average memory-access time
  = hit time + miss rate x miss penalty

hit time(1 - miss rate) + missrate)
miss rat x miss penalty,

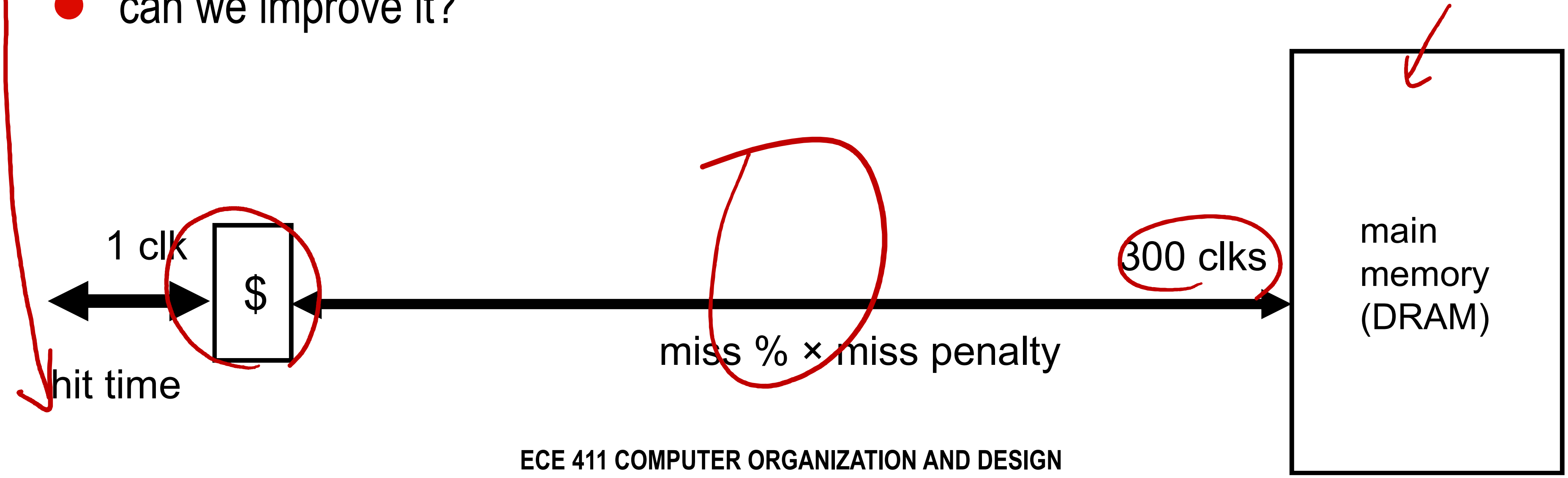
- miss penalty: time to fetch a block from lower memory level
  - ✓ access time: function of latency
  - ✓ transfer time: function of bandwidth b/w levels
    - ○ transfer one "cache line/block" at a time
    - ○ transfer at the size of the memory-bus width

Memory Hierarchy Performance

- Average Memory Access Time (AMAT)
  - ✓ = hit time + miss rate × miss penalty
  - ✓ = T_{hit}(L1) + miss%(L1) × T(memory)
- example:
  - ✓ cache hit = 1 cycle
  - ✓ miss rate = 10% = 0.1
  - ✓ miss penalty = 300 cycles
  - ✓ AMAT = 1 + 0.1 × 300 = 31 cycles

1 +

- can we improve it?



Reducing Penalty: Multi-Level Cache

- Average Memory Access Time (AMAT)

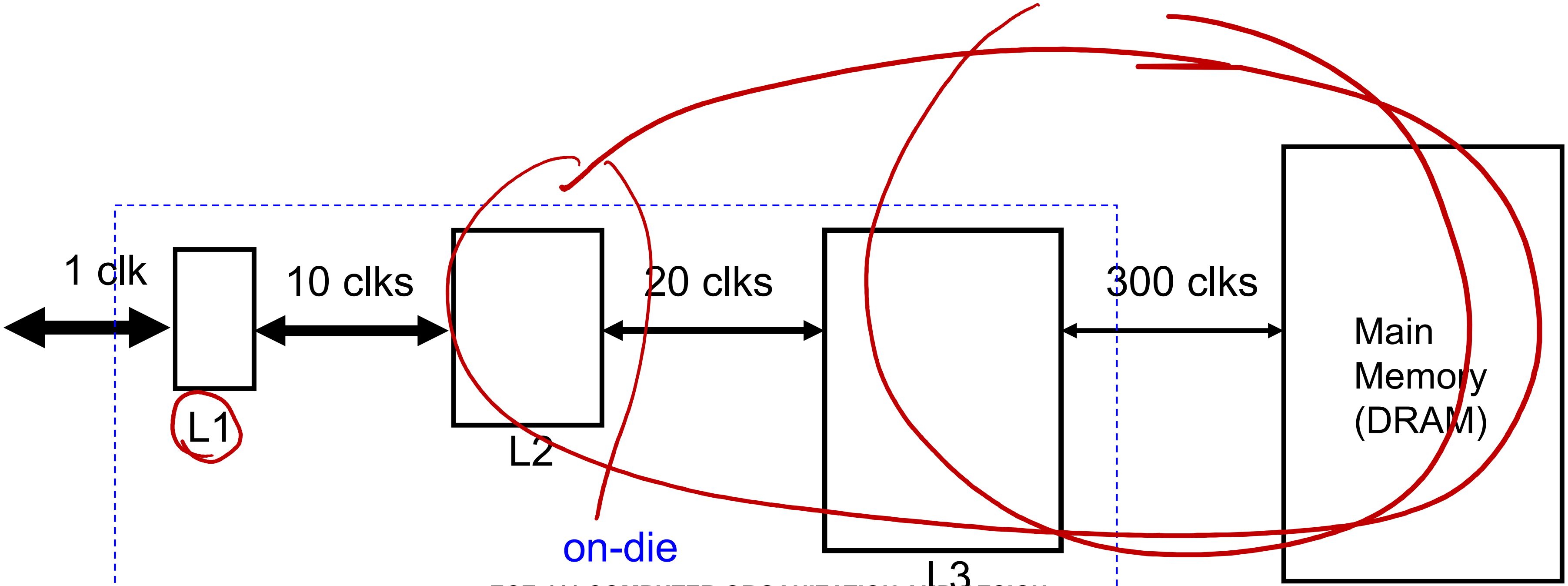
$= T_{hit}(L1) + miss\%(L1)\times(\underline{T_{hit}(L2) + miss\%(L2)\times(T_{hit}(L3) + miss\%(L3)\times T(memory)}\,)\,)$

$= T_{hit}(L1) + miss\%(L1)\times T_{miss}(L1)$

$= T_{hit}(L1) + miss\%(L1)\times\{T_{hit}(L2) + miss\%(L2)\times(T_{miss}(L2)\,\}$

$= T_{hit}(L1) + miss\%(L1)\times\{T_{hit}(L2) + miss\%(L2)\times(T_{miss}(L2)\,\}$

$= T_{hit}(L1) + miss\%(L1)\times\{T_{hit}(L2) + miss\%(L2)\times[\,T_{hit}(L3) + miss\%(L3)\times T(memory)\,]\,\}$



AMAT Example

$= T_{hit}(L1) + miss\%(L1)\times(T_{hit}(L2) + miss\%(L2)\times(T_{hit}(L3) + miss\%(L3) \times T(memory)))$

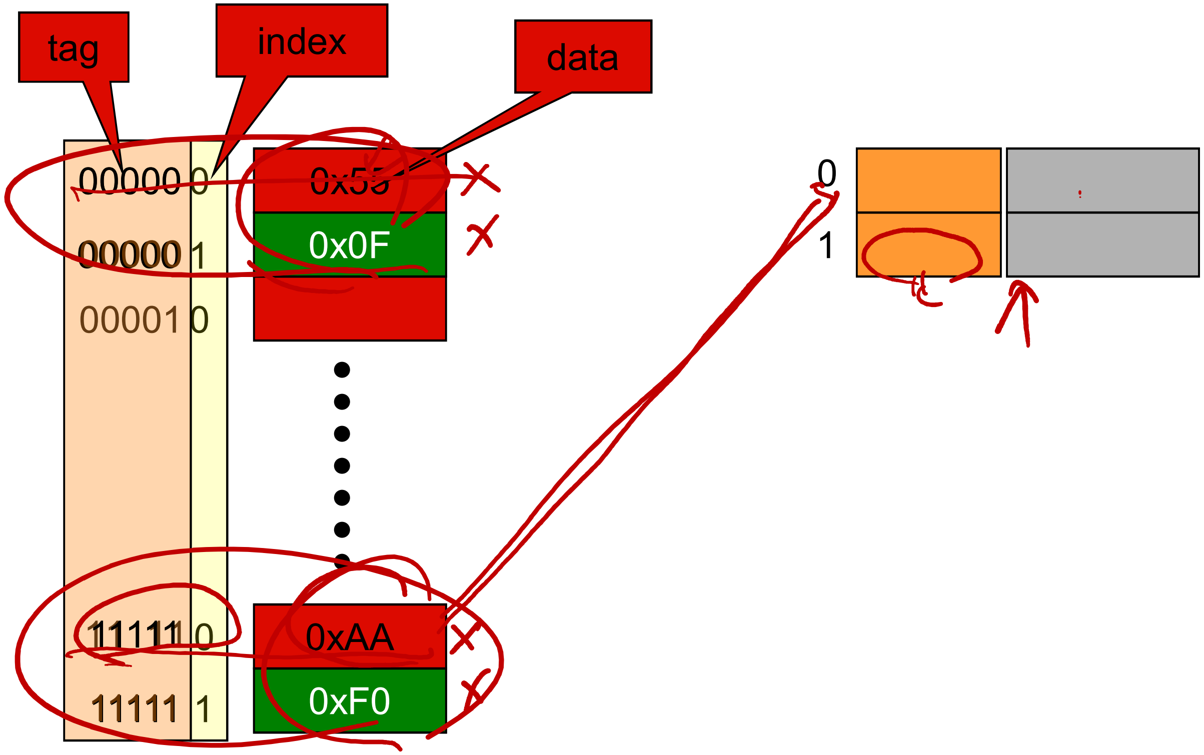
- Example:
  - ✓ miss rate L1=10%, $T_{hit}(L1)$ = 1 cycle
  - ✓ miss rate L2=5%, $T_{hit}(L2)$ = 10 cycles
  - ✓ miss rate L3=1%, $T_{hit}(L3)$ = 20 cycles
  - ✓ T(memory) = 300 cycles
- AMAT = ?
  - ✓ 2.115 (compare to 31 with no multi-levels)
  - ✓ 14.7× speed-up!

Types of Caches

type of cache	mapping of data from memory to cache	complexity of searching the cache
direct mapped (DM)	a memory value can be placed at **a single corresponding location** in the cache	fast indexing mechanism
set-associative (SA)	a memory value can be placed in **any of a set of locations** in the cache	slightly more involved search mechanism
fully-associative (FA)	a memory value can be placed in **any location** in the cache	extensive hardware resources required to search (CAM)

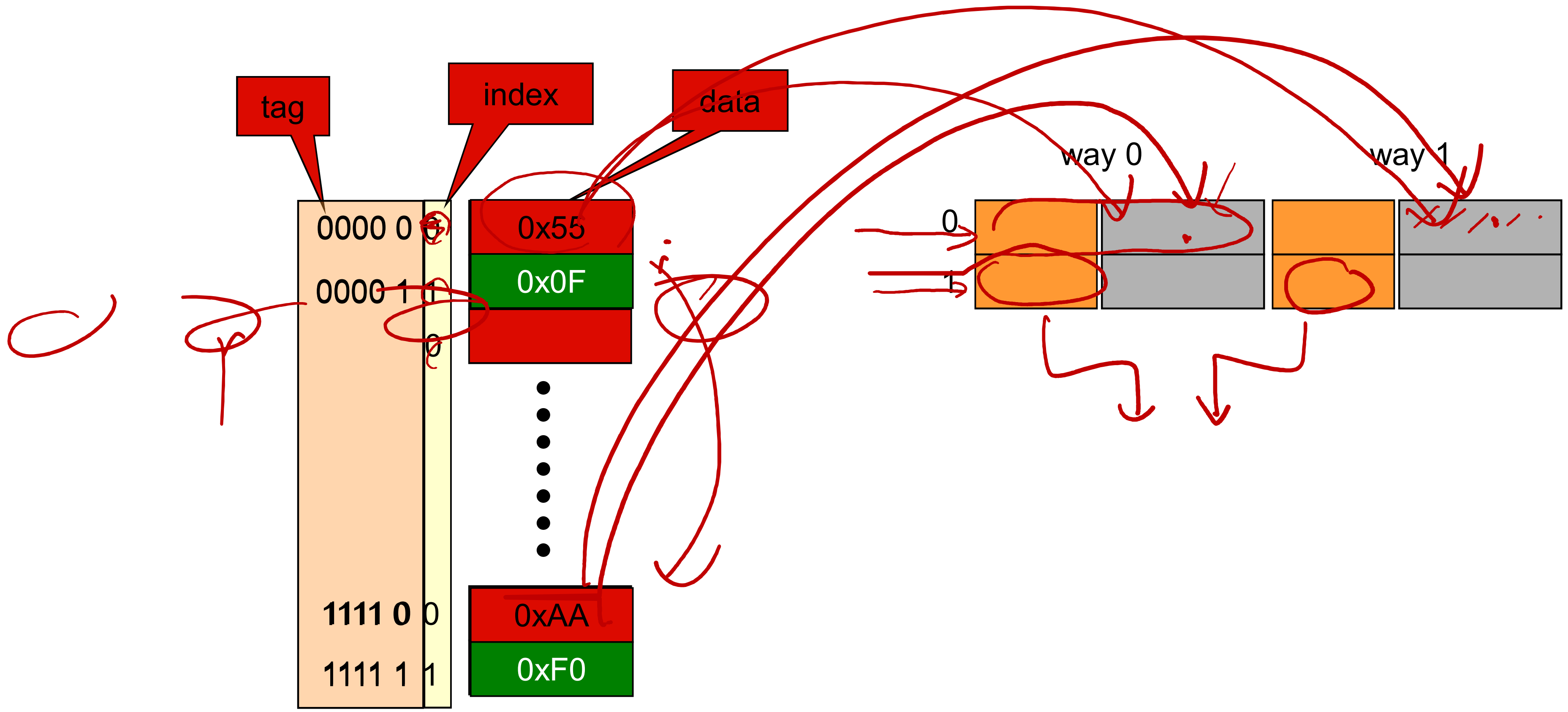
Direct Mapping

- direct mapping:
  - ✓ a memory value can only be placed at a single corresponding location in the cache



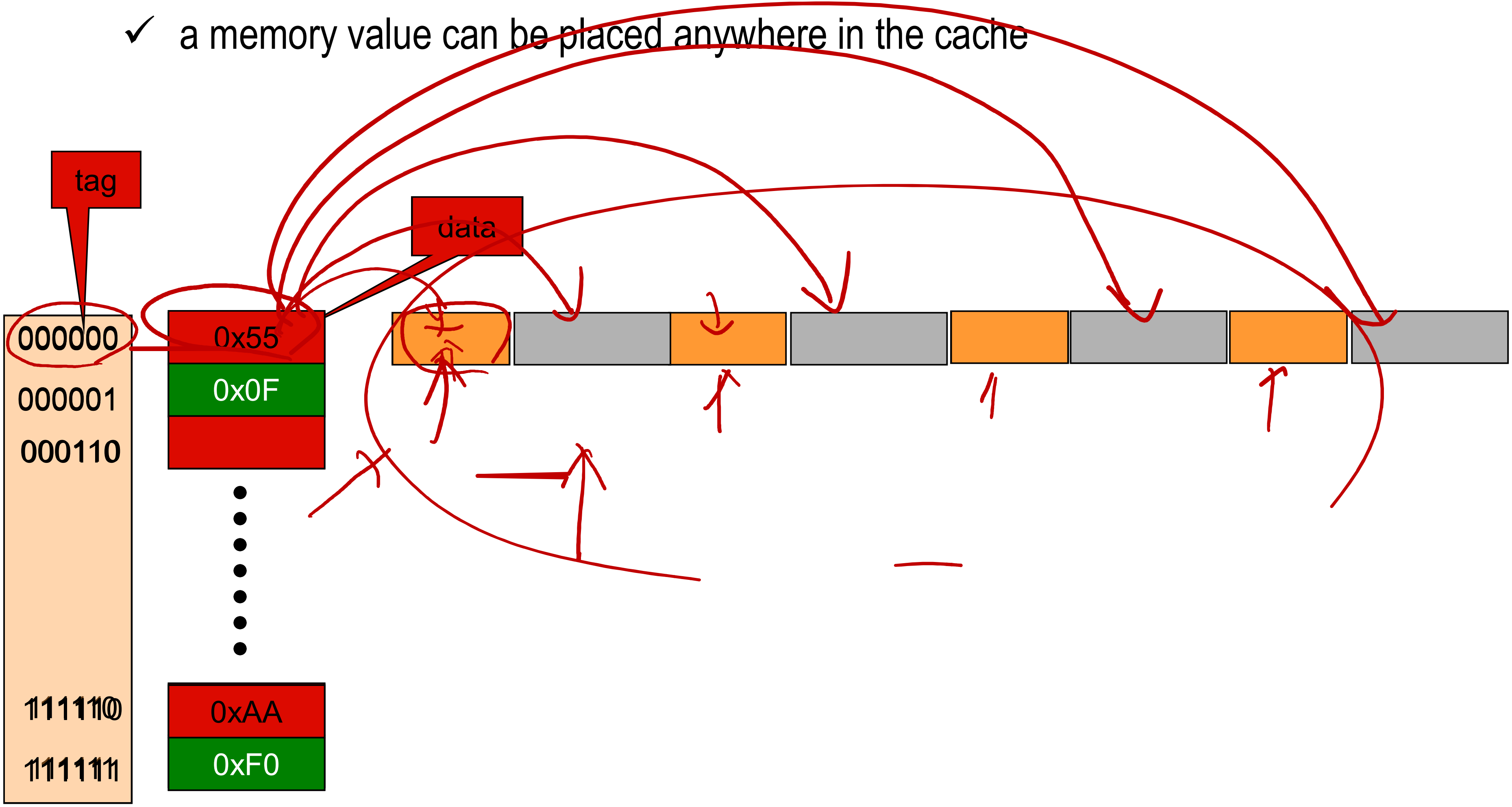
Set Associative Mapping (2-Way)

- set-associative mapping:
  - ✓ a memory value can be placed in any location of a set in the cache



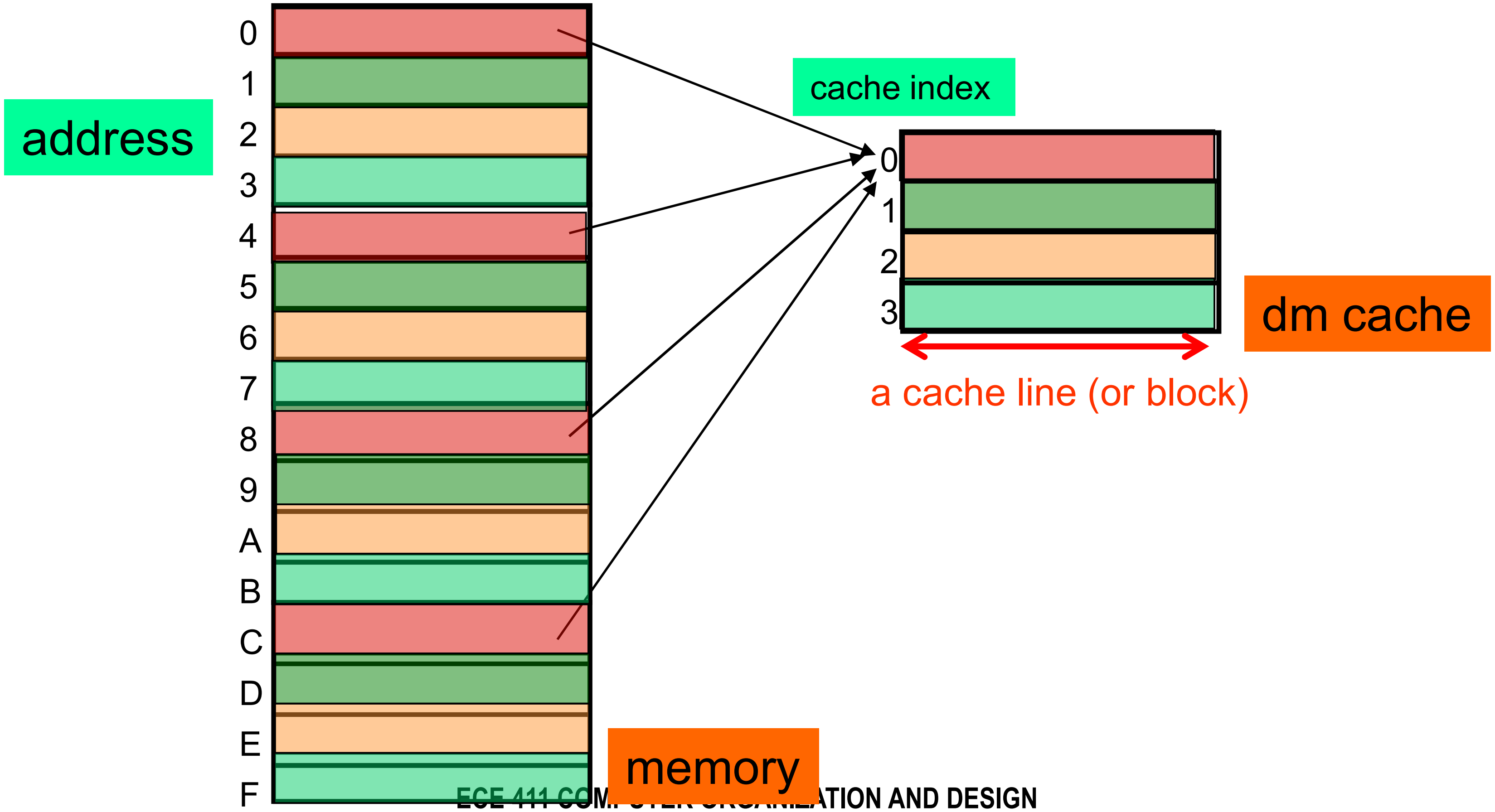
Fully Associative Mapping

- fully-associative mapping:
  - ✓ a memory value can be placed anywhere in the cache



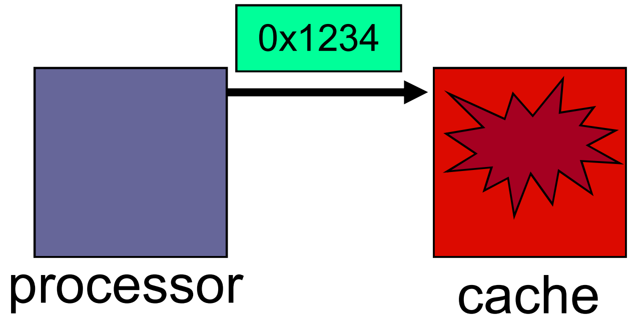
Direct Mapped Cache

- location 0 is occupied by data from (0, 4, 8, and C)
  - ✓ which one should we place in the cache?
  - ✓ how can we tell which one is in the cache?



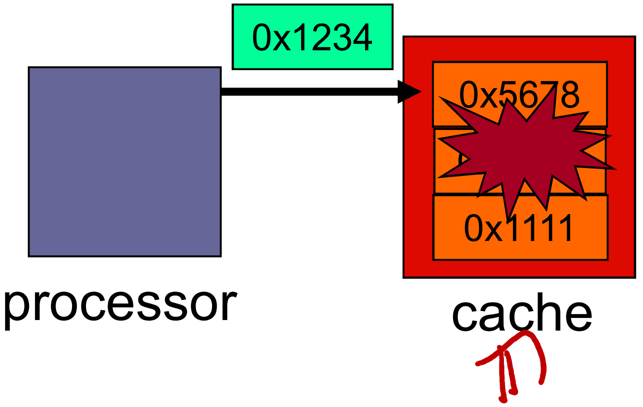
Three Cs (Cache Miss Terms)

- compulsory misses: conflict misses capacity ~~miss~~
  - ✓ cold start misses (caches do not have valid data at the start of the program)



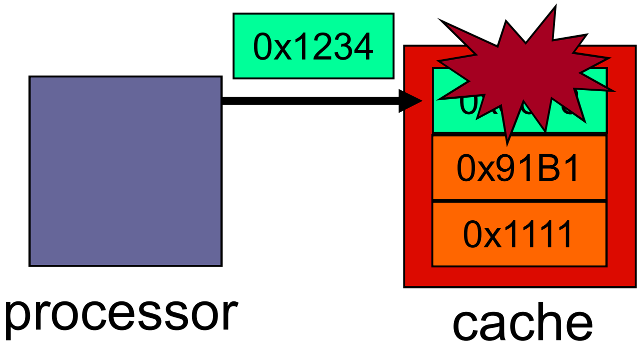
Three Cs (Cache Miss Terms)

- capacity misses:
  - ✓ increase cache size



Three Cs (Cache Miss Terms)

- conflict misses:
  - ✓ increase cache size and/or associativity.
  - ✓ associative caches reduce conflict misses



Four Central Questions in Designing a Cache

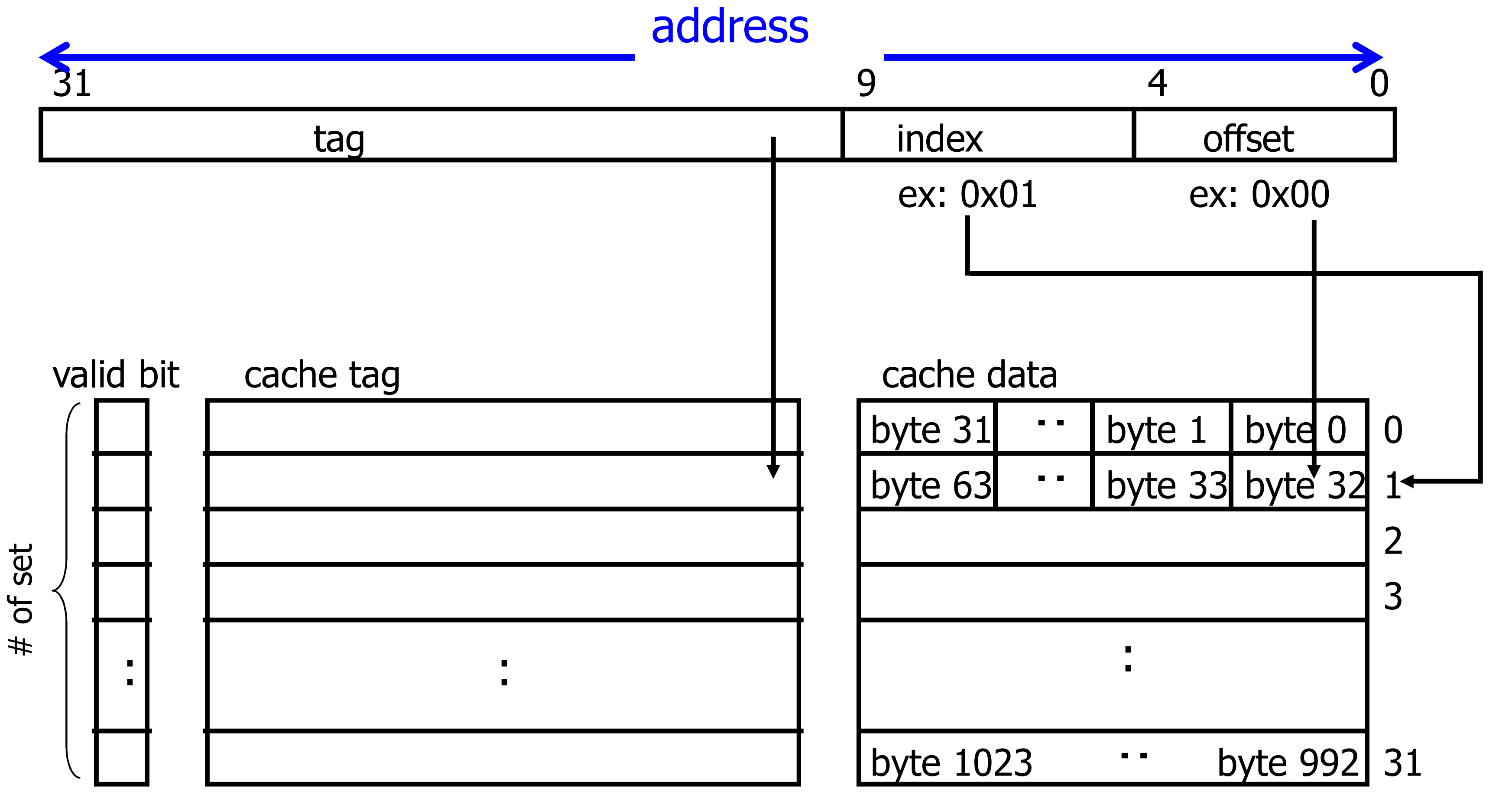
- P-I-R-W:
  - ✓ placement: where can a block of memory go?
  - ✓ identification: how do i find a block of memory?
  - ✓ replacement: how do i make space for new blocks?
  - ✓ write policy: how do i propagate changes?
- need to consider these for all levels of the memory hierarchy
  - ✓ L1/L2/L3 caches now
- main memory, disks have similar issues, addressed later

Describing Caches: 7 Parameters

- access time: T_{hit}
- capacity
  - ✓ total amount of data the cache can hold
    - ○ # of blocks × block size
- block (line) size
  - ✓ the amount of data that gets moved into or out of the cache as a chunk
    - ○ analogous to page size in virtual memory
- replacement policy
  - ✓ what data is replaced on a miss?
- associativity
  - ✓ how many locations in the cache is a given address eligible to be placed in?
- unified, instruction, data
  - ✓ what type of data is kept in the cache? We'll cover this in more detail later

Example: 1KB DM Cache, 32-byte Lines

- lowest M bits are offset (Line Size = 2M)
- index = log2 (# of sets)



Example of Caches

- given a 2MB, direct-mapped physical caches, line size=64bytes, and 52-bit physical address
  - ✓ tag size?
  - ✓ now change it to 16-way, tag size?
  - ✓ how about if it’s fully associative, tag size?

Announcement

- today's lecture: cache basics
  - ✓ Ch. 5.1 – 5.4 (HP1)
- next lecture: other cache topics (e.g., replacement policy)
  - ✓ Ch. 5.4 – 5.8 (HP1)
- MP assignment
  - ✓ MP1 due on 2/4 5pm