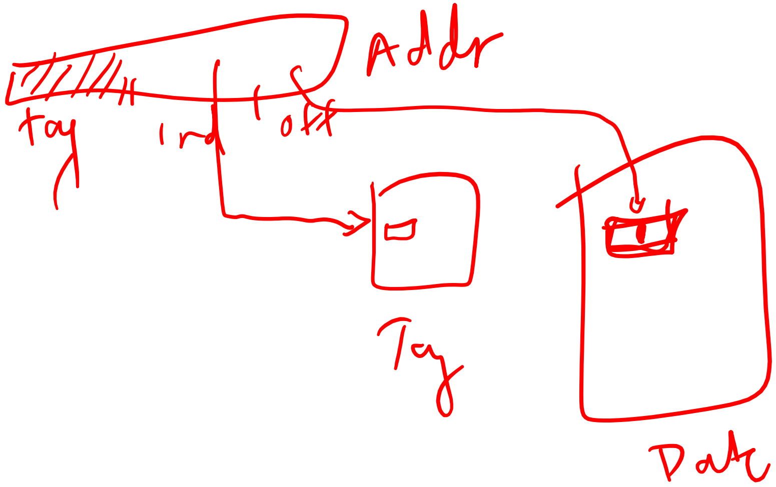
Lecture 23: Cache, Memory

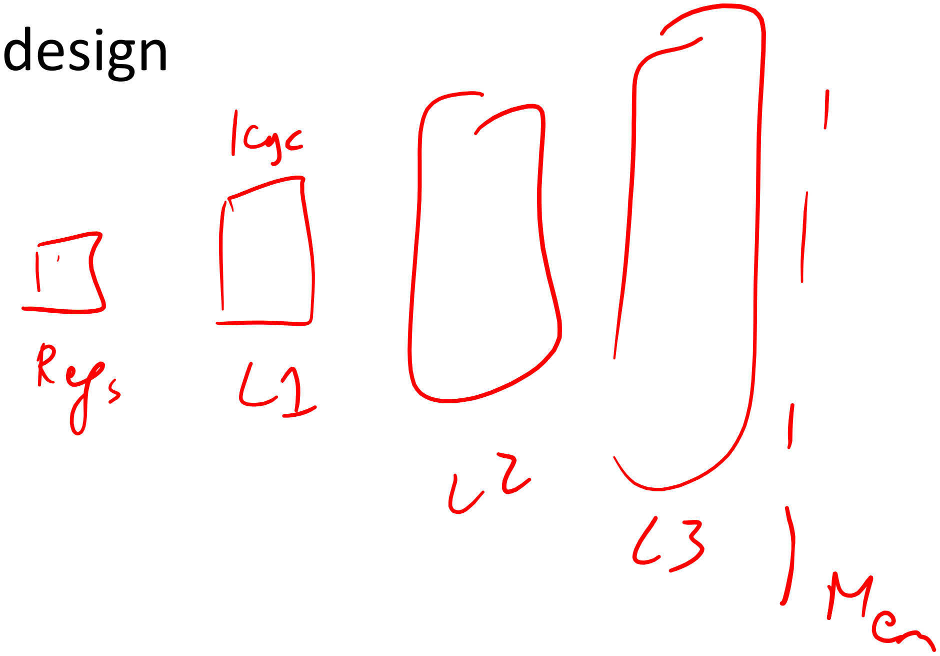
40% of instr end up in L2
20% of in end up in L3

- Today's topics:

HW 9 due Tue/Wed

  - Example problems in cache design
  - Caching policies
  - Main memory system

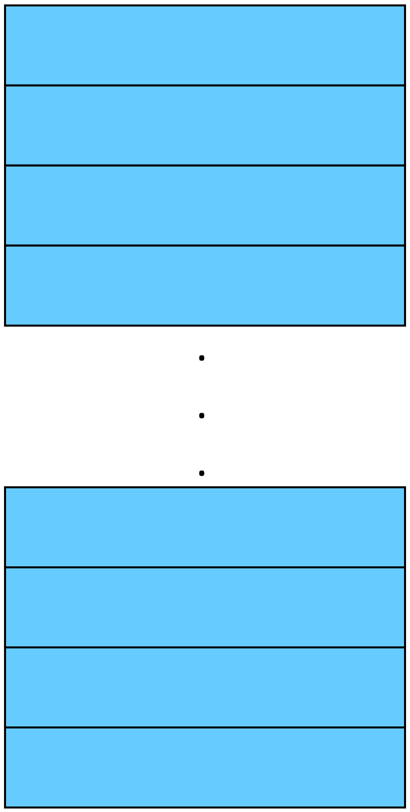




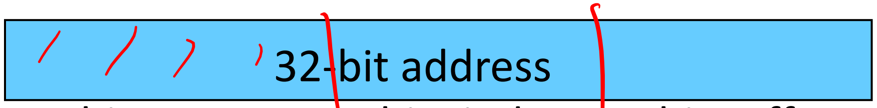
Example 2

Offset = address % blksize
Index = (address / blksize) % sets
Tag = address / (blksize x sets)

Show how the following addresses map to the cache and yield hits or misses.
The cache is direct-mapped, has 16 sets, and a 64-byte block size.
Addresses: 8, 96, 32, 480, 976, 1040, 1096



Offset = address % 64 (address modulo 64, extract last 6)
Index = address/64 % 16 (shift right by 6, extract last 4)
Tag = address/1024 (shift address right by 10)



	22 bits tag	4 bits index	6 bits offset	
8:	0	0	8	M
96:	0	1	32	M
32:	0	0	32	H
480:	0	7	32	M
976:	0	15	16	M
1040:	1	0	16	M
1096:	1	1	8	M

Example 3

(HW 9, Q1) – yet different

(Also see 1st example 3/29 lecture)

- A pipeline has CPI 1 if all loads/stores are L1 cache hits
  40% of all instructions are loads/stores
  85% of all loads/stores hit in 1-cycle L1
  50% of all (10-cycle) L2 accesses are misses
  Memory access takes 100 cycles
  What is the CPI?

1000 instrs

→ 1000 cycles (ignore pipeline warm-up)

Exec time

= 1000 + 600
+ 3000
= 4600 cyc

$\text{CPI} = \frac{4600}{1000} = 4.6$

400 are ld/st

85% of 400 = $\underbrace{\text{340 are L1 hits}}_{\text{not stalls}}$

15% (60 instrs) are L1 misses

60 x 10 cyc = 600 cyc accessing L2. ←

30 are L2 misses

30 X 100 cyc = 3000 cyc accessing memory. ←

Example 3

- A pipeline has CPI 1 if all loads/stores are L1 cache hits
  40% of all instructions are loads/stores
  85% of all loads/stores hit in 1-cycle L1
  50% of all (10-cycle) L2 accesses are misses
  Memory access takes 100 cycles
  What is the CPI?

Start with 1000 instructions
1000 cycles (includes all 400 L1 accesses)
+ 400 (ld/st) x 15% x 10 cycles (the L2 accesses)
+ 400 x 15% x 50% x 100 cycles (the mem accesses)
= 4,600 cycles
CPI = 4.6

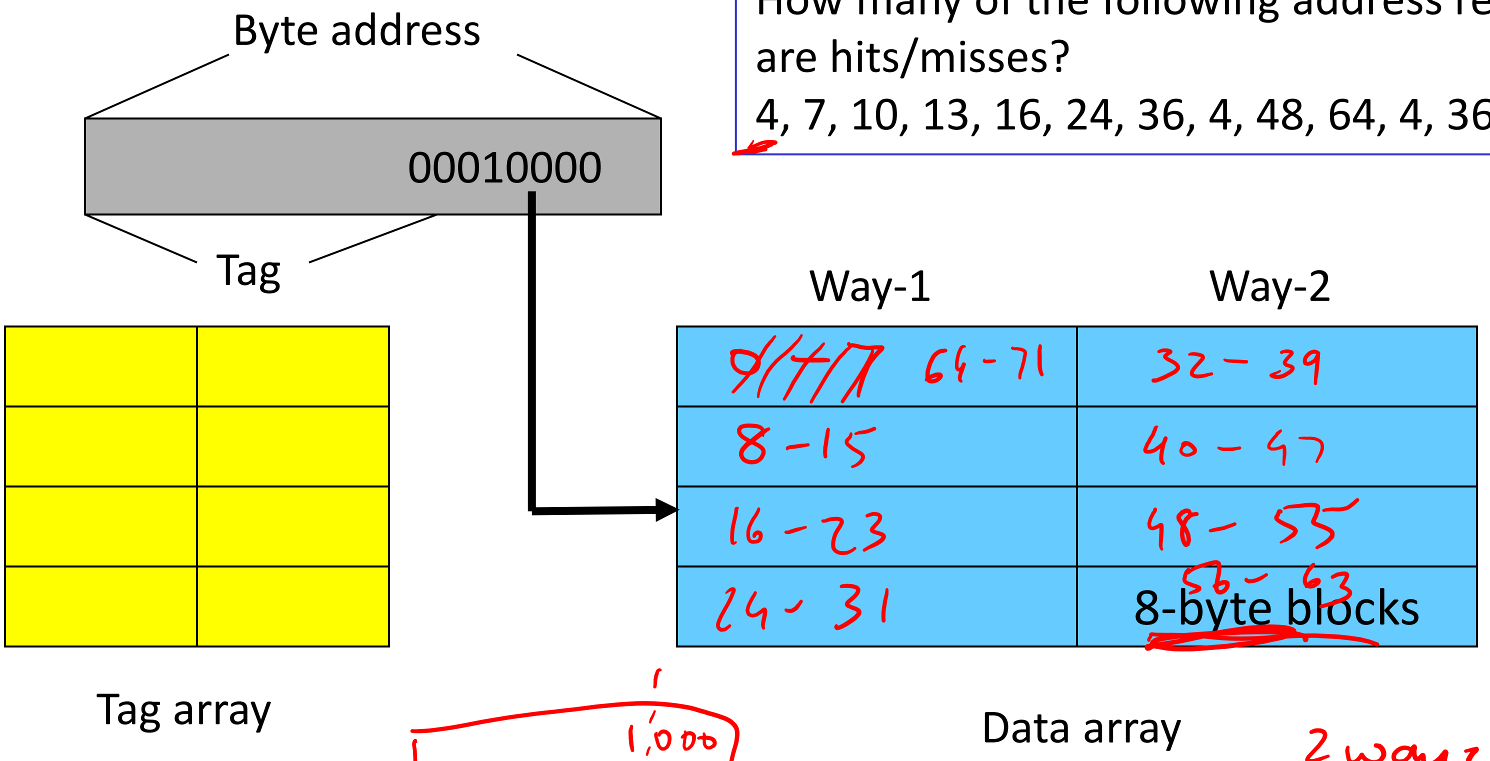
Example 4

0, 23 | 4 | 567

8B blocks

Assume that addresses are 8 bits long
How many of the following address requests are hits/misses?
4, 7, 10, 13, 16, 24, 36, 4, 48, 64, 4, 36, 64, 4

LRU



Way-1	Way-2
~~0/4/7~~ 64-71	32-39
8-15	40-47
16-23	48-55
24-31	56-63

4 sets

2 ways

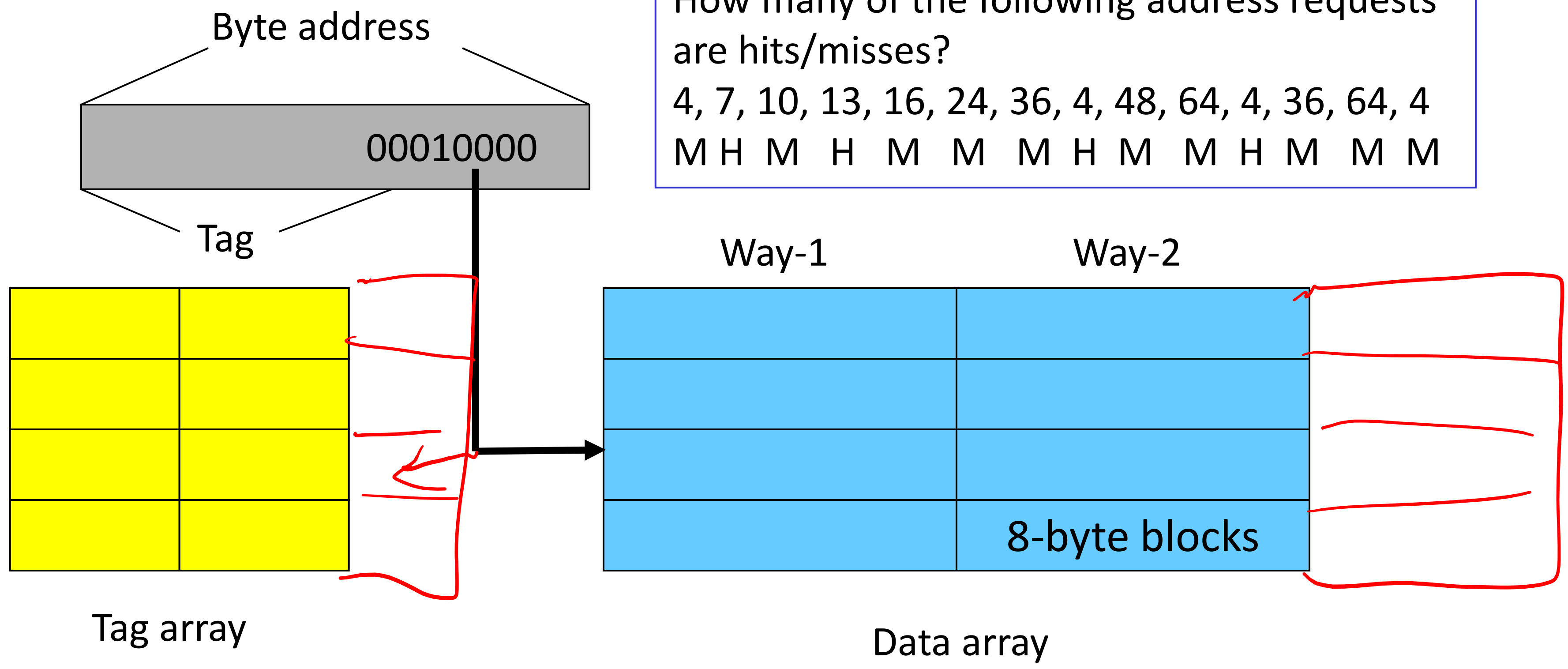
1,000

index 3 offset bits

010 000

Example 4

Assume that addresses are 8 bits long
How many of the following address requests are hits/misses?
4, 7, 10, 13, 16, 24, 36, 4, 48, 64, 4, 36, 64, 4
M H M H M M M H M M H M M M



Cache Misses

16-way 32MB L3

24-way 24MB L3

11-way 22MB L3

st

- On a write miss, you may either choose to bring the block into the cache (write-allocate) or not (write-no-allocate)
- On a read miss, you always bring the block in (spatial and temporal locality) – but which block do you replace?
  - no choice for a direct-mapped cache
  - randomly pick one of the ways to replace
  - replace the way that was least-recently used (LRU)
  - FIFO replacement (round-robin)

Pseudo-LRU

8-way cache
recency list for each set

MR: 3, 5, 1, 2, 6, 7 ... LR

MR: 1, 3, 5, 2, 6, 7 ...

Writes

- When you write into a block, do you also update the copy in L2?
  - write-through: every write to L1 → write to L2
  - write-back: mark the block as dirty, when the block gets replaced from L1, write it to L2

- Writeback coalesces multiple writes to an L1 block into one L2 write

- Writethrough simplifies coherency protocols in a multiprocessor system as the L2 always has a current copy of data

Types of Cache Misses

3 C's

prefetching → compiler

- Compulsory misses: happens the first time a memory word is accessed – the misses for an infinite cache

M H M M

A A A A

tiling → appl developer

- Capacity misses: happens because the program touched many other words before re-touching the same word – the misses for a fully-associative cache

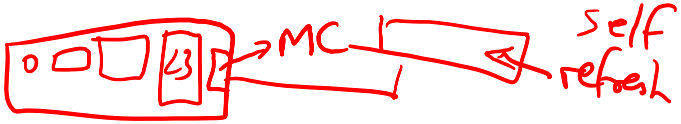
Comp

Capacity miss

confl miss

hw ← design a cache with many ways

- Conflict misses: happens because two words map to the same location in the cache – the misses generated while moving from a fully-associative to a direct-mapped cache



16 GB 64+GB

1TB

Off-Chip DRAM Main Memory

- Main memory is stored in DRAM cells that have much higher storage density

↳ Dynamic → many GBs of capacity
low cost
high density

Periodically refreshed

- DRAM cells lose their state over time – must be refreshed periodically, hence the name *Dynamic*

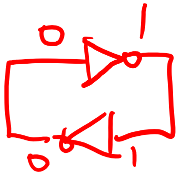
DRAM cell

8x higher density

- A number of DRAM chips are aggregated on a DIMM to provide high capacity – a DIMM is a module that plugs into a bus on the motherboard

A – every ms
B – a few times every sec ✓
every 64 ms

SRAM cell

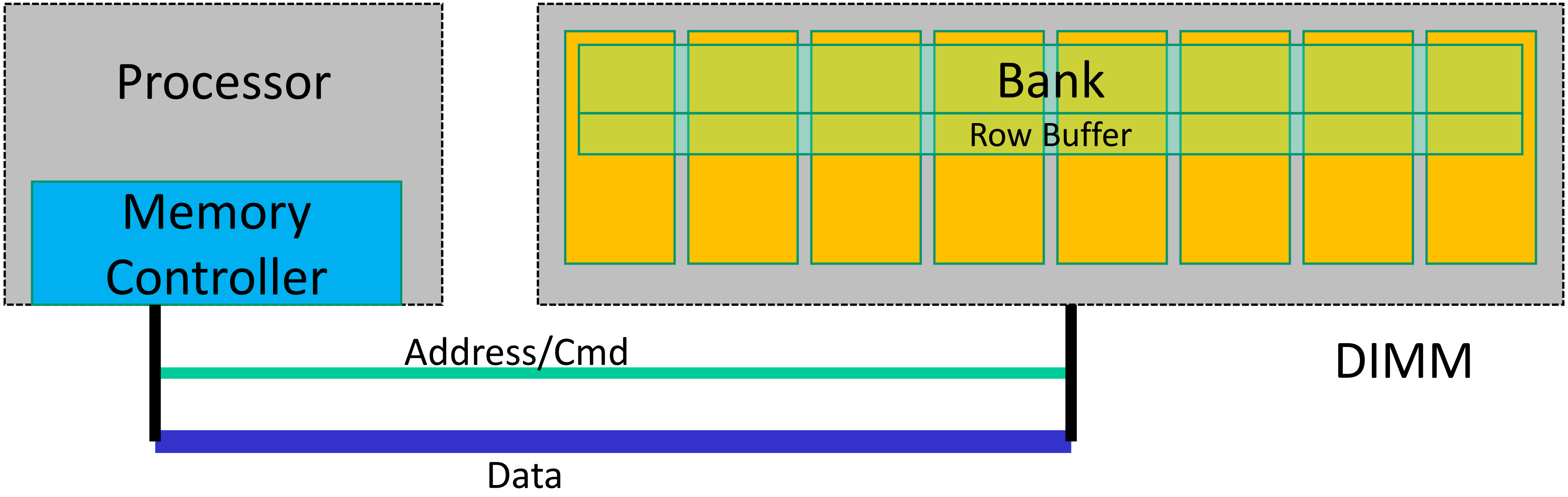


- DRAM access suffers from long access time and high energy overhead

Caches – SRAM cells → caches
static
speed
less focus on density

C – every min
D – every hour
E – Every day

Memory Architecture



- DIMM: a PCB with DRAM chips on the back and front
- The memory system is itself organized into ranks and banks; each bank can process a transaction in parallel
- Each bank has a row buffer that retains the last row touched in a bank (it's like a cache in the memory system that exploits spatial locality) (row buffer hits have a lower latency than a row buffer miss)