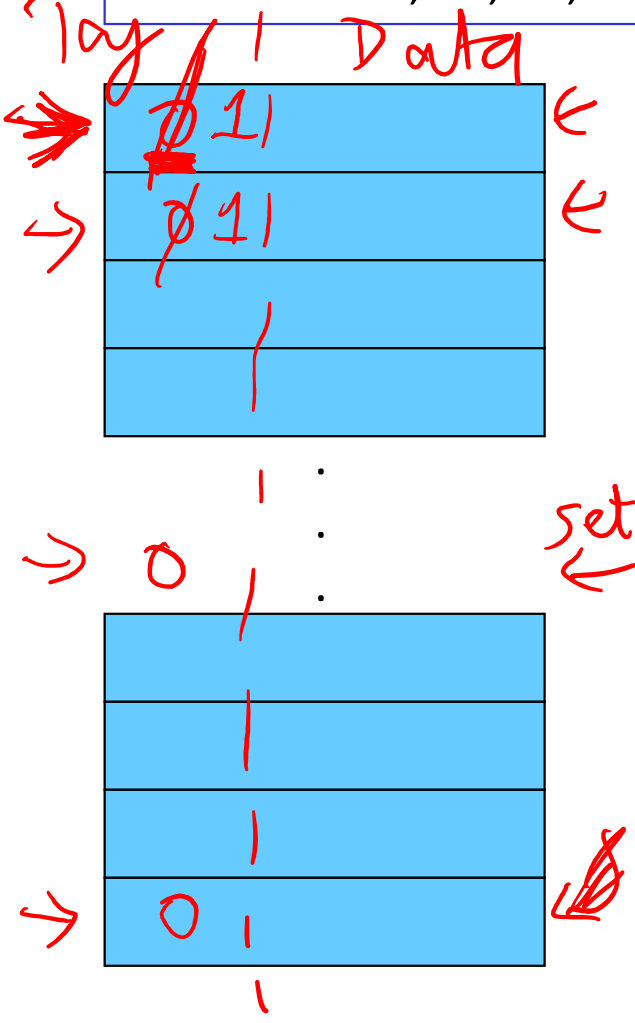
Lecture 24: Cache Wrap-Up, Memory, Security

- Today's topics:
  - Cache examples, policies
  - Main memory system
  - Hardware security intro

Example 2

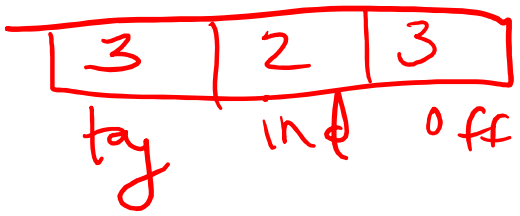
Show how the following addresses map to the cache and yield hits or misses.
The cache is direct-mapped, has 16 sets, and a 64-byte block size.
Addresses: 8, 96, 32, 480, 976, 1040, 1096



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)

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

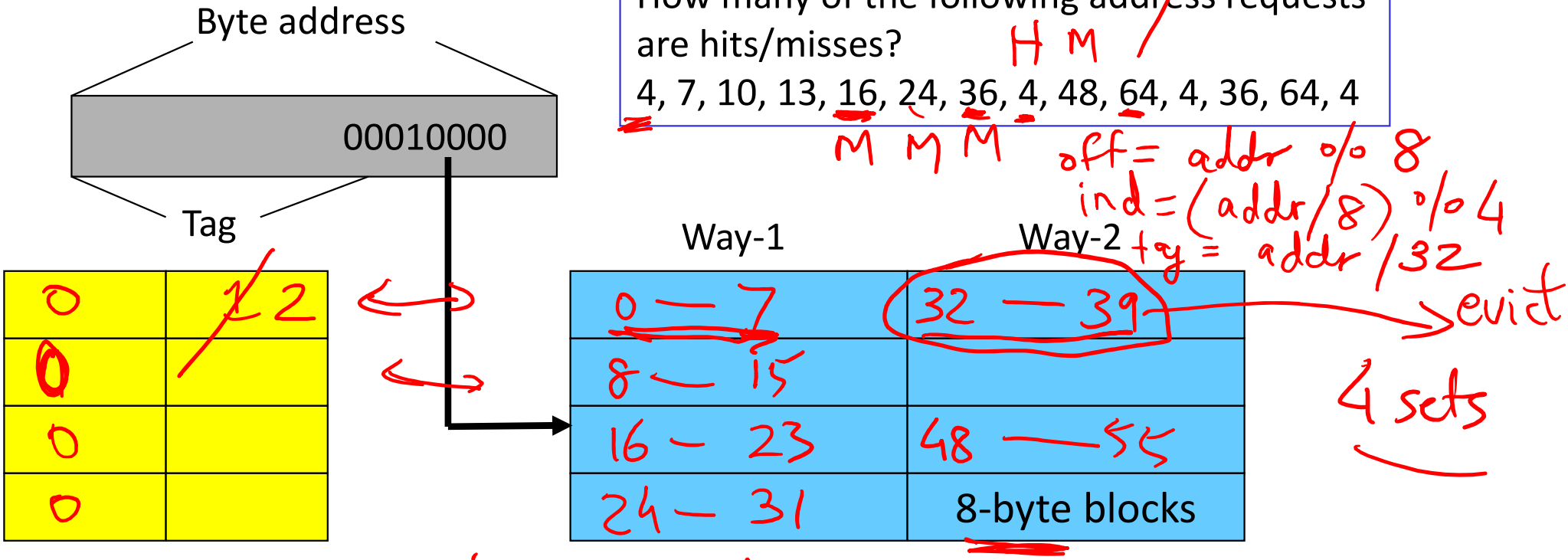
Example 4



index = 0
tag = 2

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

off = addr % 8
ind = (addr/8) % 4
tag = addr/32



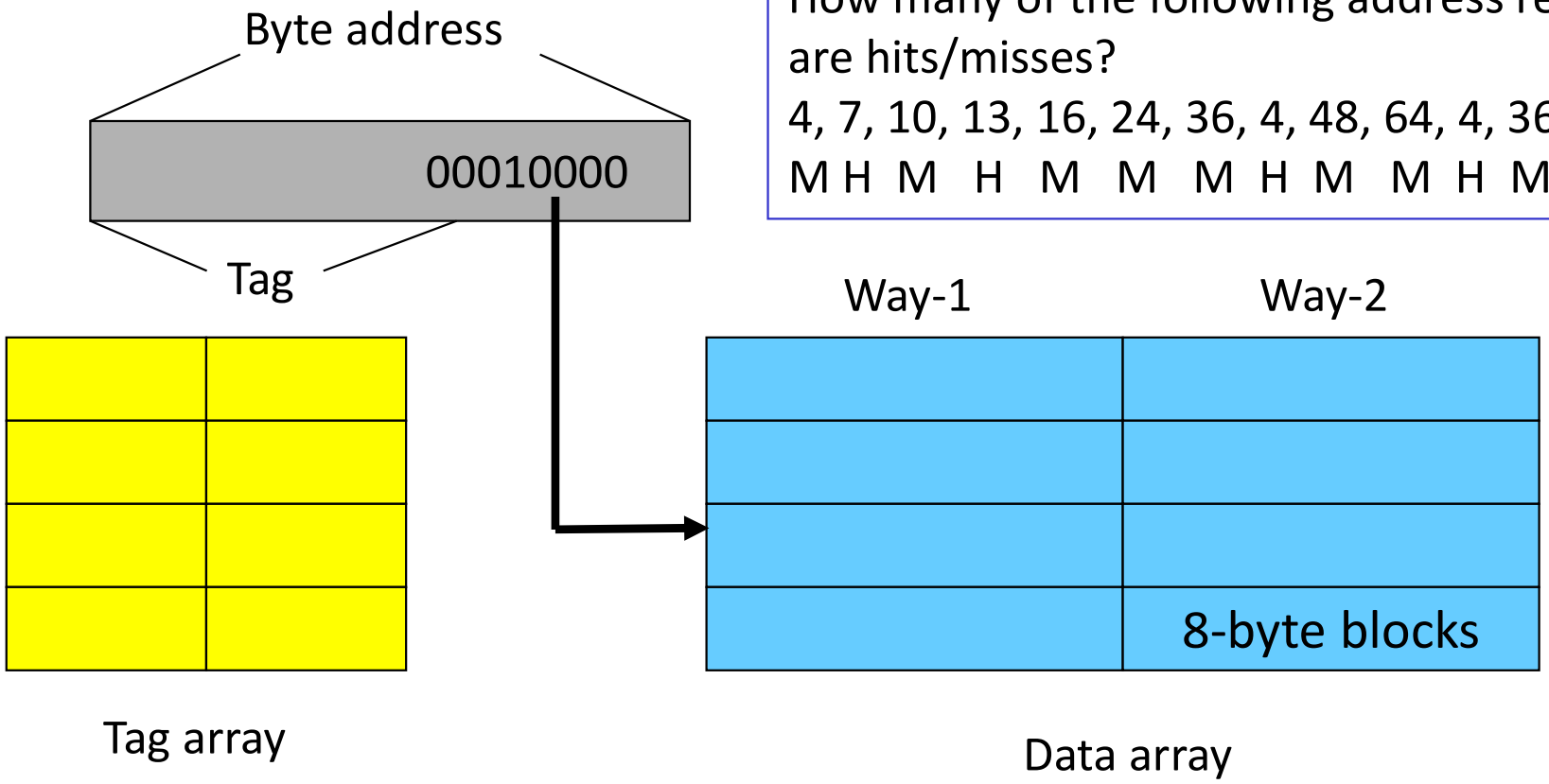
	tag
4 :	0
7 :	0
10 :	0
13 :	0

ind	
0	M
0	H
1	M
1	H

LRU replacement policy

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



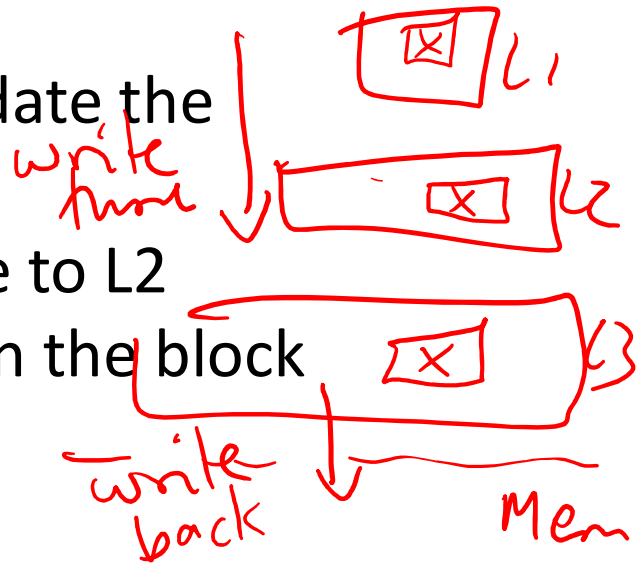
Cache Misses

- 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)

Writes

→ inclusive
exclusive

- 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

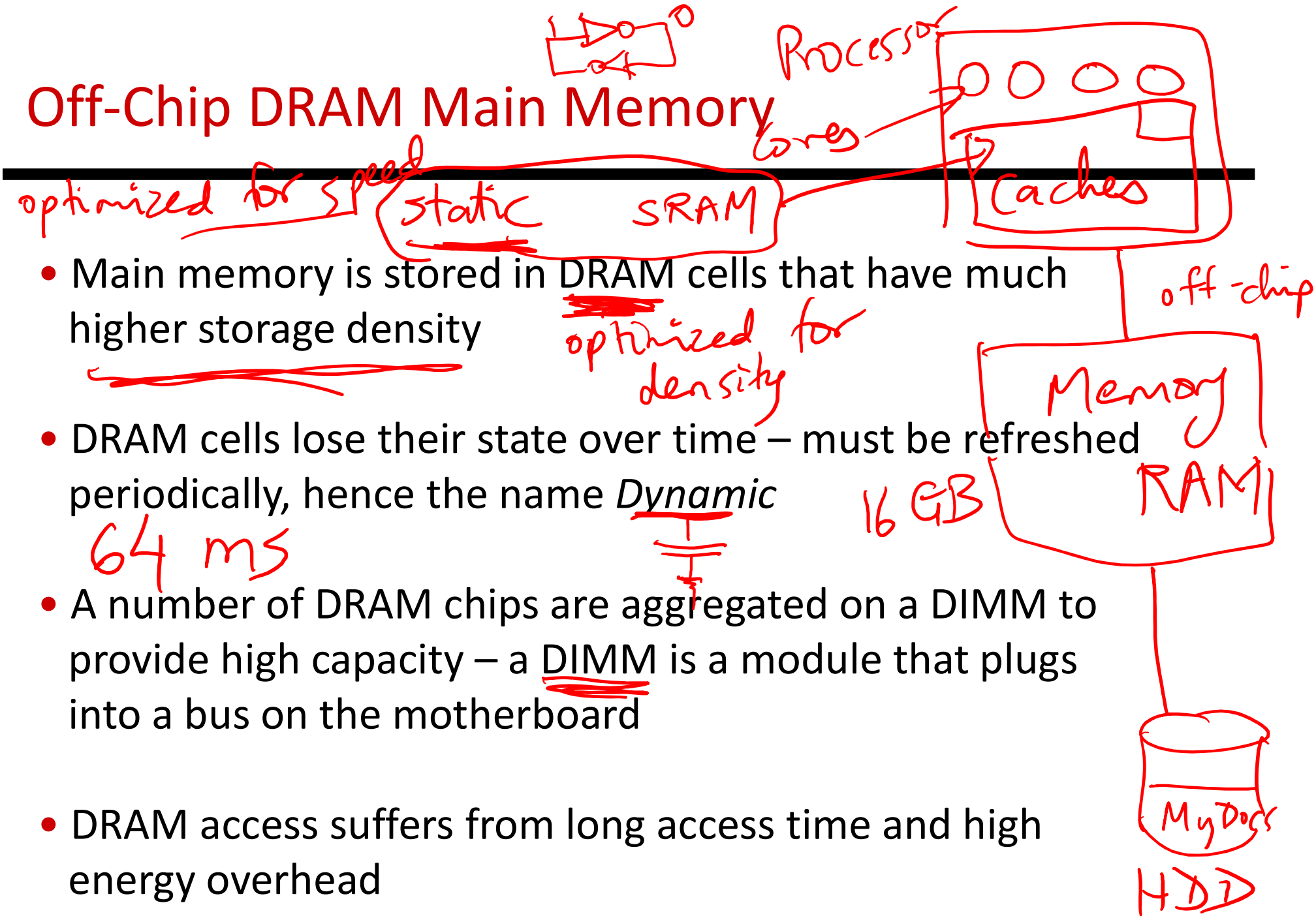
Types of Cache Misses

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

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

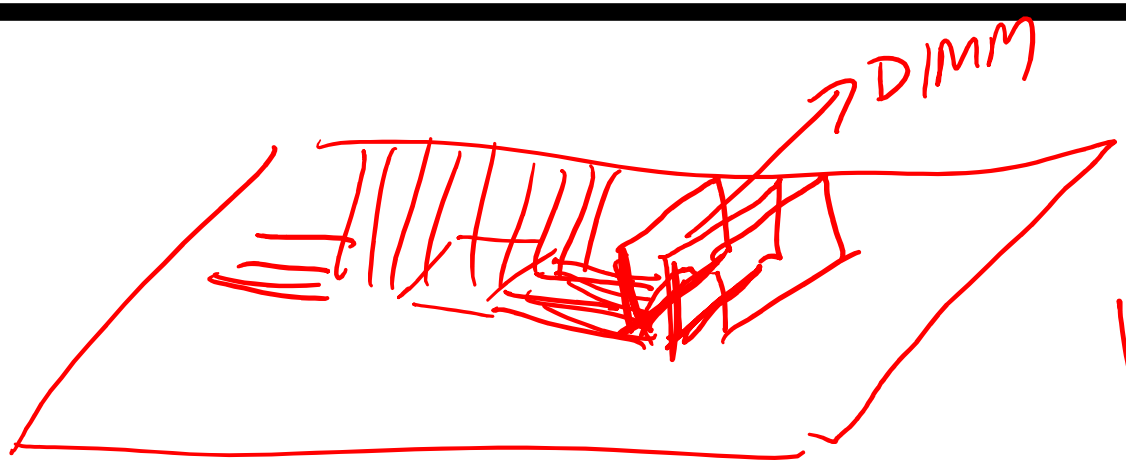
- Conflict misses: happens because two words map to the same location in the cache – the misses generated while moving from a fully-associative to a direct-mapped cache

Off-Chip DRAM Main Memory



- Main memory is stored in DRAM cells that have much higher storage density
- DRAM cells lose their state over time – must be refreshed periodically, hence the name *Dynamic*
- 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
- DRAM access suffers from long access time and high energy overhead

Memory Architecture



standardized

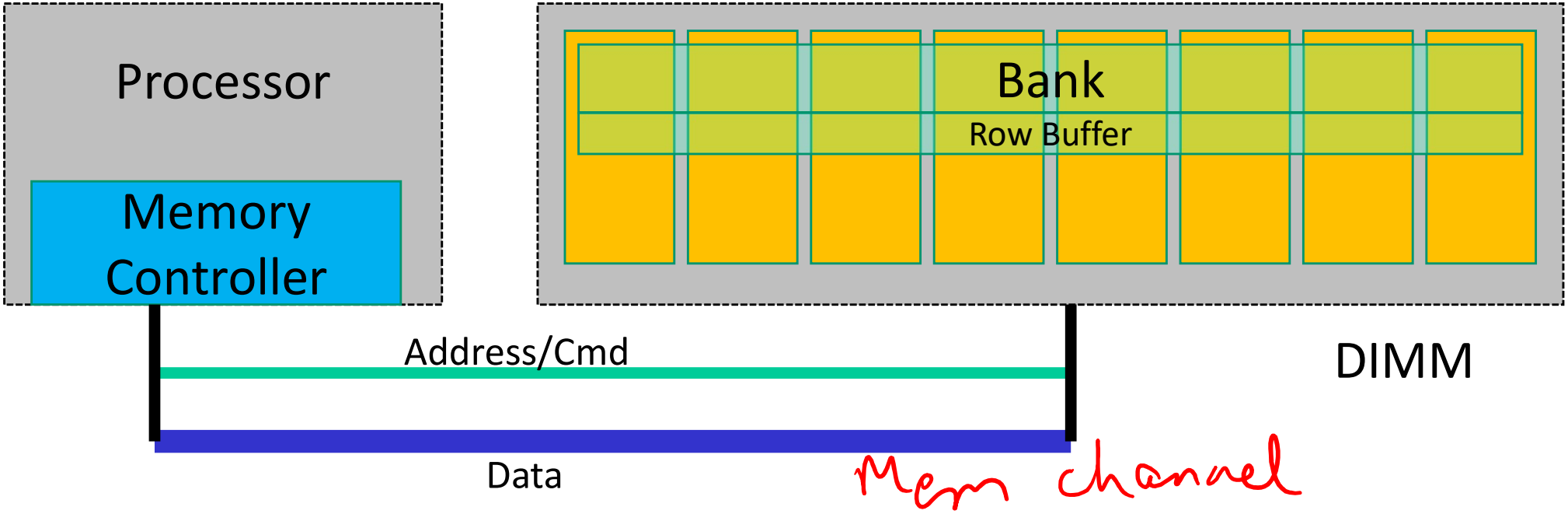
lw
↳ 1 bank
50ns
latency

64GB
2 channels
32GB ch1
32GB ch2
4 DIMMs
8GB DIMM
ranks
banks
chips

SO-DIMM
laptops

High parallelism
⇒ High Thruput

Memory Architecture



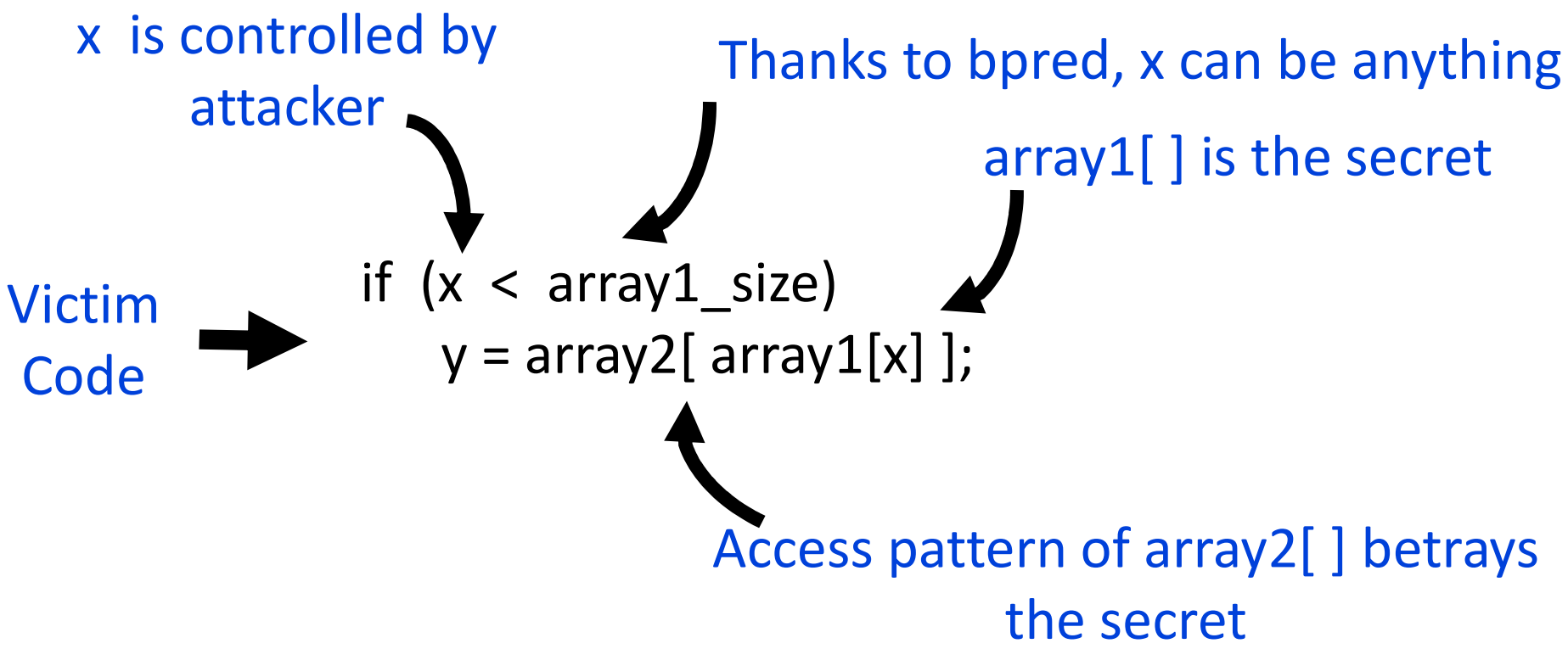
- DIMM: a PCB with DRAM chips on the back and front
- The memory system is itself organized into ranks and banks; each bank can process a transaction in parallel
- Each bank has a row buffer that retains the last row touched in a bank (it's like a cache in the memory system that exploits spatial locality)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

Hardware Security

- Software security: key management, buffer overflow, etc.
- Hardware security: hardware-enforced permission checks, authentication/encryption, etc.
- Information leakage, side channels, timing channels
- Meltdown, Spectre, SGX

Meltdown

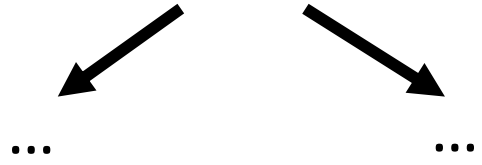
Spectre: Variant 1



Spectre: Variant 2

Victim code

R1 ← (from attacker)
R2 ← some secret
Label0: if (…)



Attacker code

Label0: if (1)

Label1: …

Victim code

Label1:
lw [R2]