# 21. Cache Organization: Wrapup & Practice Problems

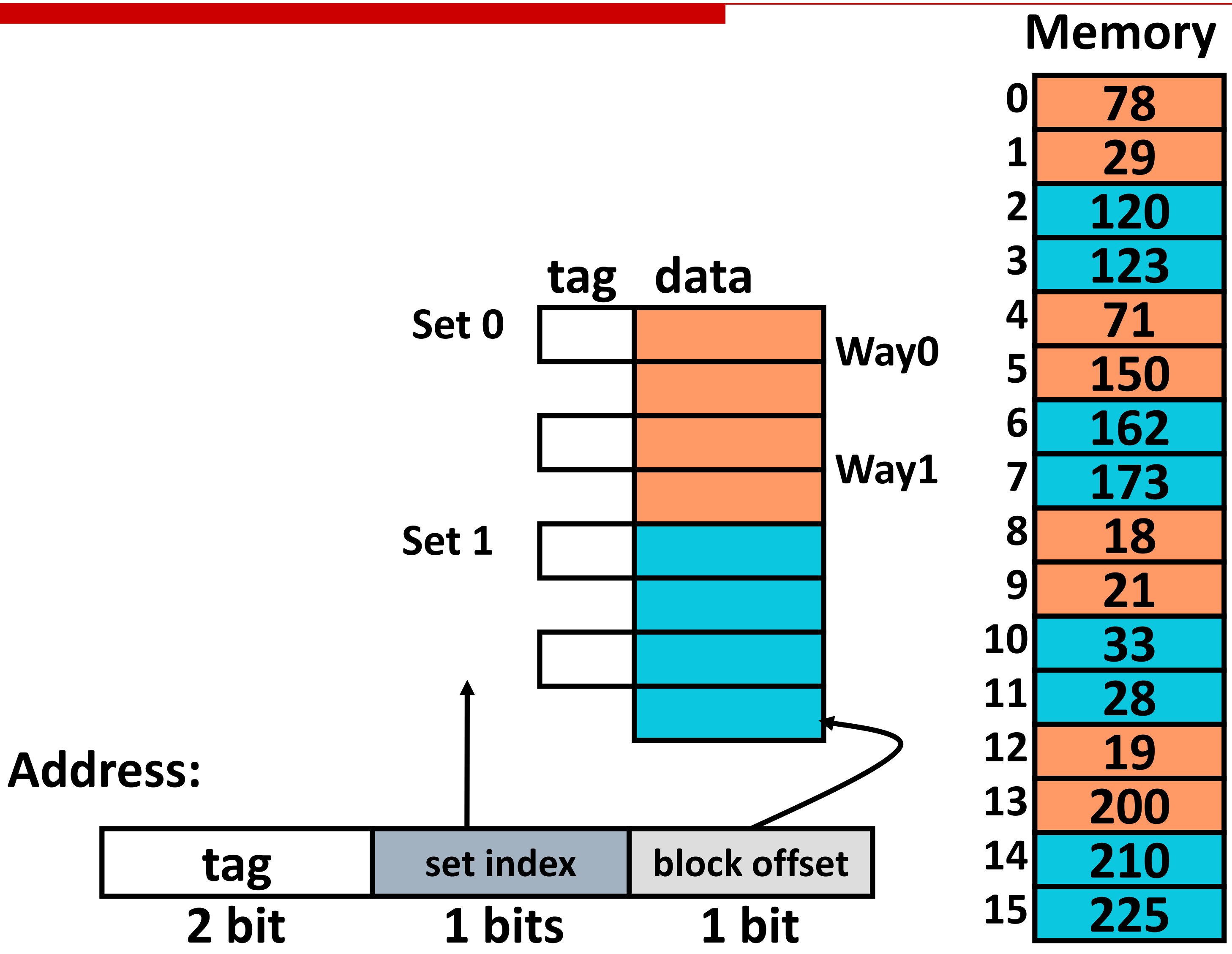
EECS 370 – Introduction to Computer Organization – Winter 2023

EECS Department
University of Michigan in Ann Arbor, USA

## Assignments

- HW5 due Monday April 3rd
- Project 4 due Thursday April 13th

# Review: Set-associative cache



## Review: The 3C's of Cache Misses

- First reference to an address
  - **Compulsory** miss
    - Also sometimes called a **"cold start"** miss
    - First reference to any block will always miss
- Cache is too small to hold all the data
  - **Capacity** miss
    - Would have had a hit with a large enough cache
- Replaced it from a busy set
  - **Conflict** miss
    - Would have had a hit with a fully associative cache

## Review: Classifying Cache Misses

- Can you classify all cache misses?
  - **Compulsory miss?**
  - **Capacity miss?**
  - **Conflict miss?**
- Yes! (with a cache simulator)
  - Simulate with a cache of unlimited size (cache size = memory size) – Any misses must be **compulsory misses**
  - Simulate again with a fully associative cache of the intended size - Any new misses must be **capacity misses**
  - Simulate a third time, with the actual intended cache - Any new misses must be **conflict misses**

## Review: Fixing cache misses

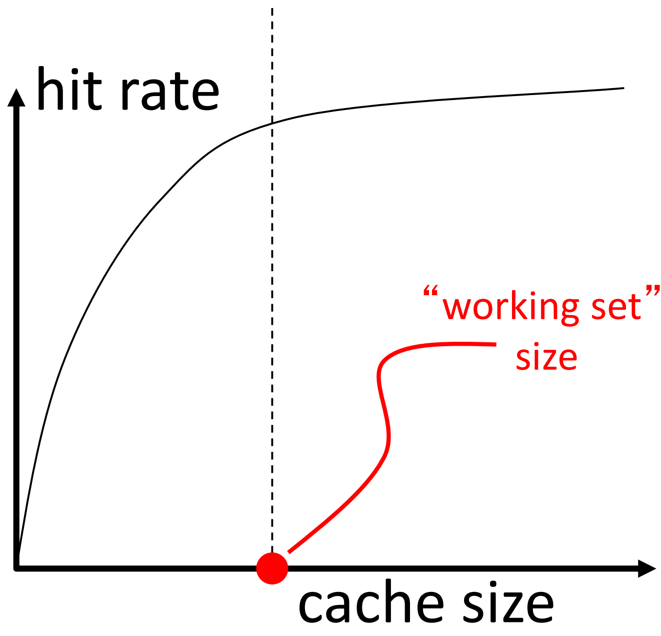
- **Compulsory** misses
  - First reference to a address
  - No way to completely avoid these
  - Reduce by **increasing block size**
  - This reduces the total number of blocks
- **Capacity** misses
  - Would have a hit with a large enough cache
  - Reduce by **building a bigger cache**
- **Conflict** misses
  - Would have had a hit with a fully associative cache
  - Cache does not have enough associativity
  - Reduce by **increasing associativity**

## Cache Parameters vs. Miss Rate

- Cache Size
- Block Size
- Associativity
- Replacement policy

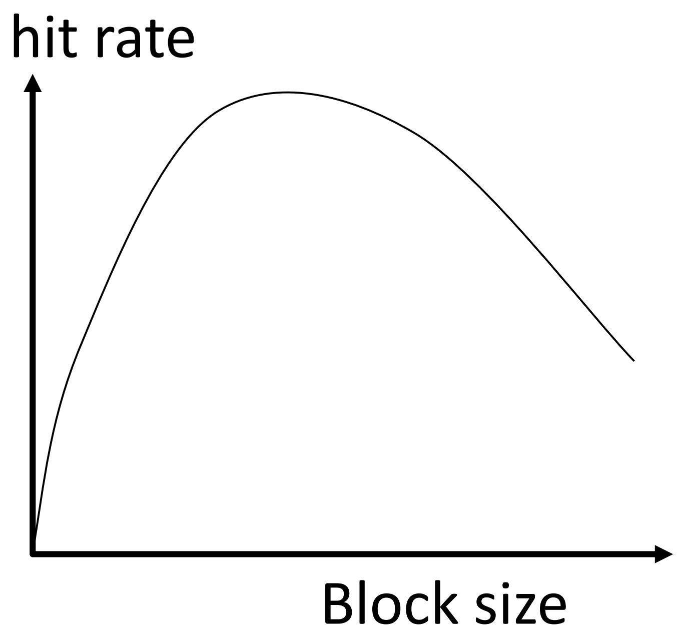
## Cache Size

- Cache size in the total data (not including tag) capacity
  - bigger can exploit temporal locality better
  - not ALWAYS better
- Too large a cache adversely affects hit & miss latency
  - smaller is faster => bigger is slower
  - access time may degrade critical path
- Too small a cache
  - doesn't exploit temporal locality well
  - useful data replaced often
- Working set: the whole set of data executing application references
  - **Within a time interval**



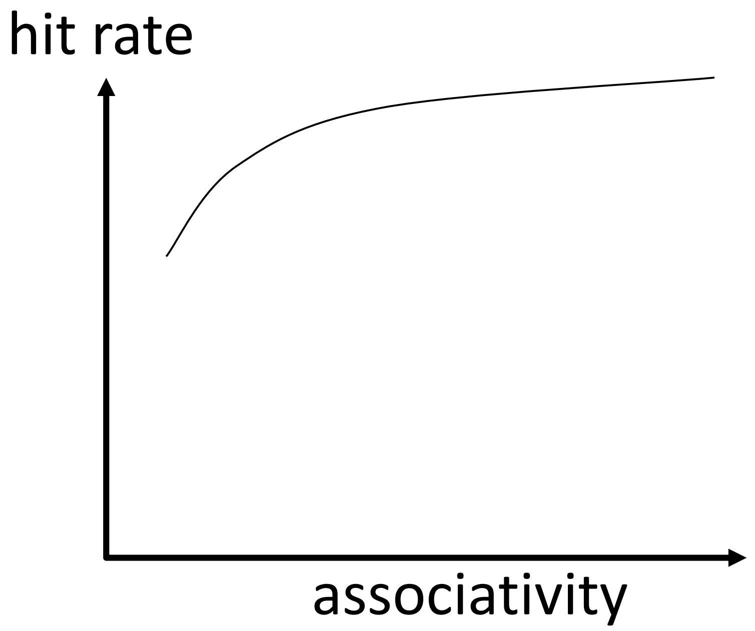
# Block size (also called Line size)

- Block size is the data that is associated with an address tag
  - Sub-blocking: A block divided into multiple pieces (each with V bit)
    - Can improve "write" performance
- Too small blocks
  - don't exploit spatial locality well
  - have larger tag overhead
- Too large blocks
  - too few total # of blocks
    - likely-useless data transferred
    - Extra bandwidth/energy consumed



## Associativity

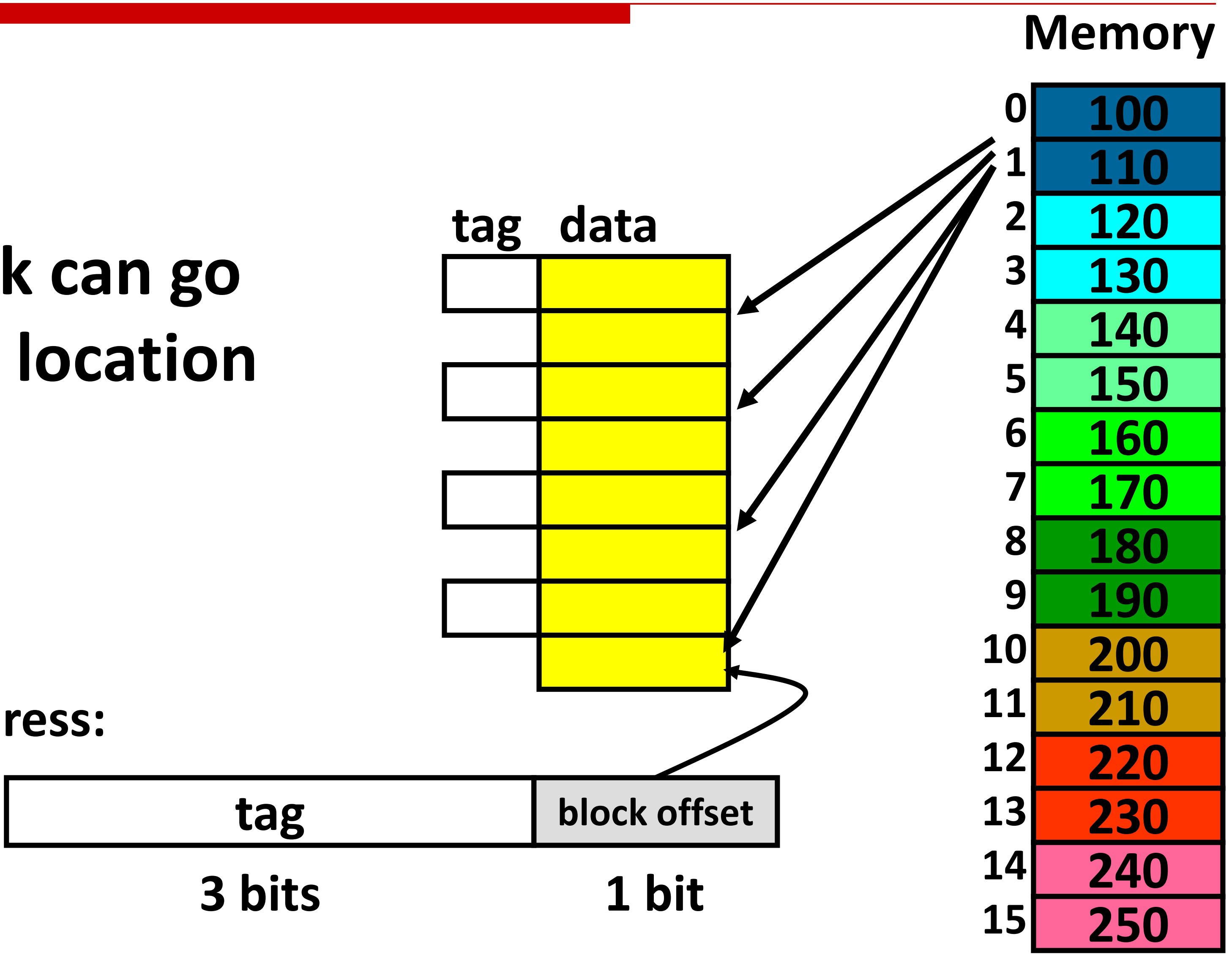
- How many blocks can map to the same index (or set)?
- Larger associativity
  - lower miss rate, less variation among programs
  - diminishing returns



- Smaller associativity
  - lower cost
  - faster hit time
    - Especially important for L1 caches
- Power of 2 associativity?

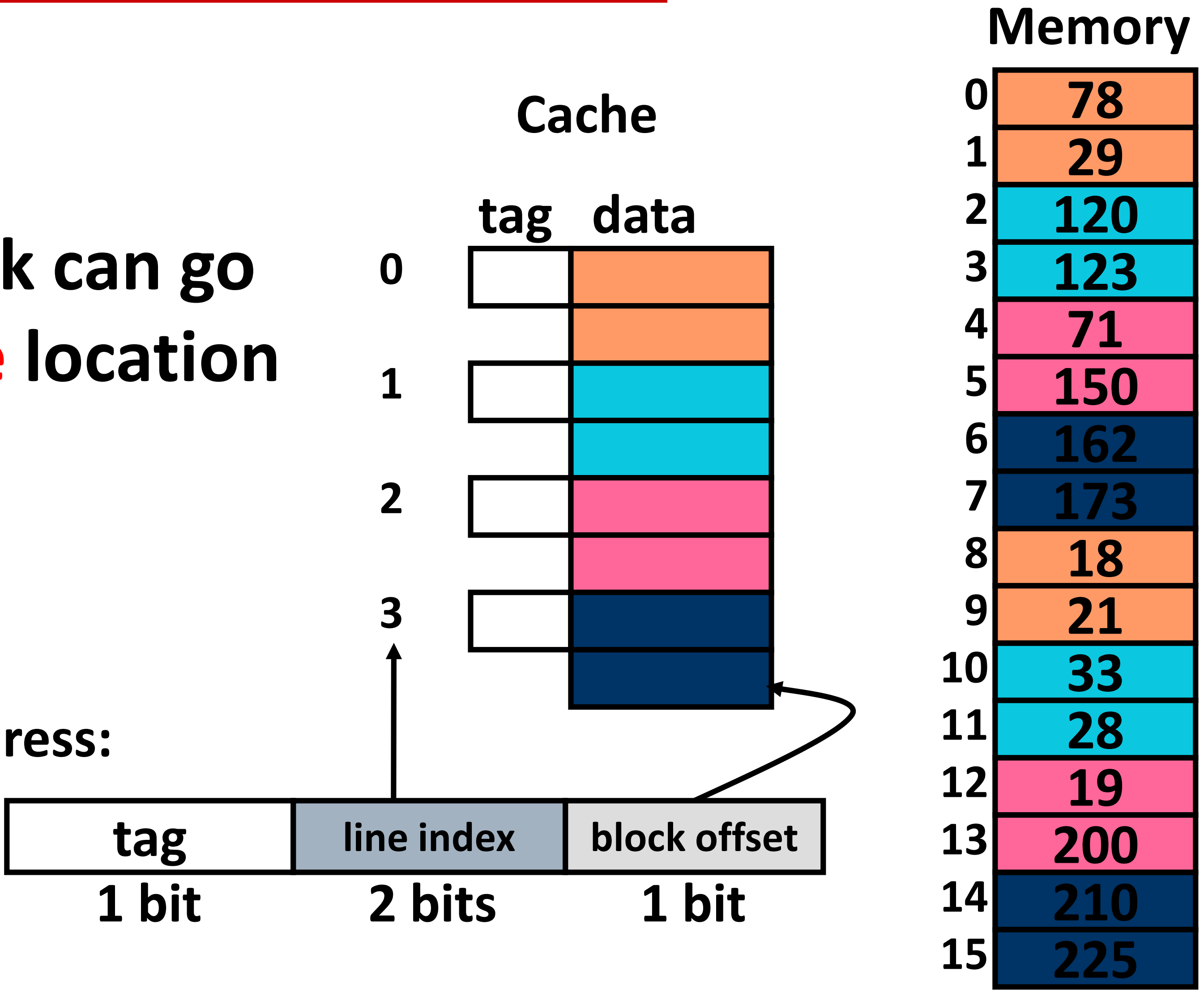
# Review: Fully-associative caches

A block can go to any location

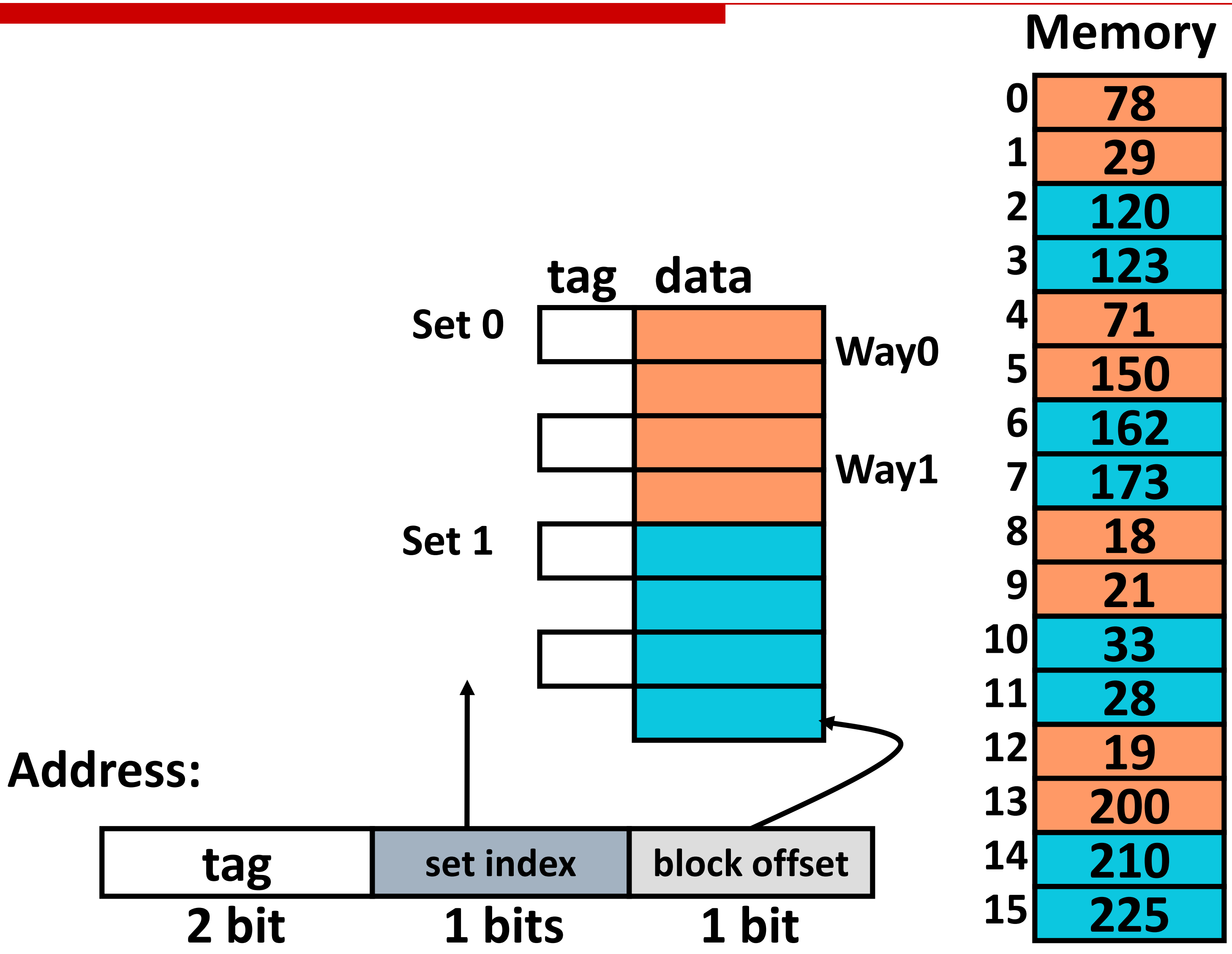


## Review: Direct-mapped caches

A block can go to **one** location



# Review: Set-associative cache



## Practice Problem 1: CPI with caches

The *grinder* application run on the LC2k with full data forwarding and all branches predicted not-taken has the following instruction frequencies:

45% R-type 20% Branches 15% Loads 20% Stores

In *grinder*, 40% of branches are taken and 50% of LWs are followed by an immediate use.

The I-cache has a miss rate of 3% and the D-cache has a miss rate of 6% (no overlapping of misses). On a miss, the main memory is accessed and has a latency of 100 ns. The clock frequency is 500 MHz.

What is the CPI of *grinder* on the LC2k?

# Practice Problem 1: Solution

The *grinder* application run on the LC2k with full data forwarding and all branches predicted not-taken has the following instruction frequencies:

45% R-type   20% Branches   15% Loads   20% Stores

In *grinder*, 40% of branches are taken and 50% of LWs are followed by an immediate use.

The I-cache has a miss rate of 3% and the D-cache has a miss rate of 6% (no overlapping of misses). On a miss, the main memory is accessed and has a latency of 100 ns. The clock frequency is 500 MHz.

What is the CPI of *grinder* on the LC2k?

Stalls per cache miss = 100 ns / 2ns = 50 cycles (500 Mhz → 2ns cycle time)

CPI = 1 + data hazard stalls + control hazard stalls + icache stalls + dcache stalls

CPI = 1 + 0.15*0.50*1 + 0.20*0.40*3 + 1*0.03*50 + 0.35*0.06*50

## Practice Problem 2: Memory Usage

- Say you have the following:
  - A program that generates 2 Billion loads and 1 Billion stores, each 4 bytes in size.
  - A cache with a 32-byte block which gets a 95% hit rate on that program.

- How many bytes of memory would be read and written if:
  - We had a write-though cache with a no-write allocate policy?
  - We had a write-back cache with a write-allocate policy? (Assume 25% of all misses result in a dirty eviction)
  - We had no cache?

## Practice Problem 2: Solution (1/3)

- Let’s start with the no-cache case.
  - All stores go to memory and are 4 bytes each
    - Writes: 1 billion stores* 4 bytes = 4 billion bytes
  - All loads go to memory and are 4 bytes each.
    - Reads: 2 billion loads* 4 bytes = 8 billion bytes
- Write-though, no allocate.
  - All stores still go to memory and are still 4 bytes each.
    - Writes: 1 billion stores* 4 bytes = 4 billion bytes
  - Only loads that miss in the cache go to memory. But they read the full cache block.
    - Reads: 2 billion loads* 0.05* 32 bytes = 3.2 billion bytes

## Practice Problem 2: Solution (2/3)

- Write-back, write-allocate (data reads)
  - *Store* misses result in a cache block being read.
    - Reads: 1 billion stores* 0.05* 32 bytes = 1.6 billion bytes
  - *Load* misses result in a cache block being read.
    - Reads: 2 billion loads* 0.05* 32 bytes = 3.2 billion bytes
  - So that is 4.8 billion bytes of data read.

## Practice Problem 2: Solution (3/3)

- When do we write to memory?
  - Only on dirty evictions. Can be done by both loads and stores.
    - Recall we are assuming 25% of all evictions are of dirty data.

- Write-back, write-allocate (data writes)
  - *Store* misses result in dirty eviction 1/4 of the time.
    - Writes: 1 billion stores* 0.05* 32 bytes*(.25) = 0.4 billion bytes
  - *Load* misses result in a cache block being read.
    - Writes: 2 billion loads* 0.05* 32 bytes*(.25) = 0.8 billion bytes

- Note: could have just taken the read numbers and multiplied by 25%.

# Practice Problem 3: CPI w/ Caches 2

- Given a 200 MHz processor with 8KB instruction and data caches and a with memory access latency of 20 cycles. Both caches are 2-way associative. A program running on this processor has a 95% icache hit rate and a 90% dcache hit rate. On average, 30% of the instructions are loads or stores. The CPI of this system, if caches were ideal would be 1.
- Suppose you have 2 options for the next generation processor, which do you pick?
  - Option 1: Double the clock frequency—assume this will increase your memory latency to 40 cycles. Also assume a base CPI of 1 can still be achieved after this change.
  - Option 2: Double the size of your caches, this will increase the instruction cache hit rate to 98% and the data cache hit rate to 95%. Assume the hit latency is still 1 cycle.

## Practice Problem 3: Solution

Option 1: (double clock freq, base cycle time is 5 ns, so new cycle time is 2.5 ns)

CPI = baseCPI + IcacheStallCPI + DcacheStallCPI

CPI = 1.0 + 0.05*40 + 0.3*0.1*40 = 4.2

Execution time = 4.2 * Ninstrs * 2.5ns = 10.5ns * Ninstrs

Option 2 (icache/dcache miss rates lowered to 2% and 5%)

CPI = baseCPI + IcacheStallCPI + DcacheStallCPI

CPI = 1.0 + 0.02*20 + 0.3*0.05*20 = 1.7

Execution time = 1.7 * Ninstrs * 5ns = 8.5ns * Ninstrs

Therefore, Option 2 is the better choice

## Practice Problem 4: Guess that cache!

Here is the series of address references (in hex) to a cache of size 512 bytes. You are asked to **determine the configuration of the cache**. Assume 12-bit addresses

0x310 – Miss

0x30f – Miss

0x510 – Miss

0x31f – Hit

0x72d – Miss

0x72f – Hit

0x320 – Miss

0x520 – Miss

0x720 - Miss

**Block size: ?**

**Associativity: ?**

**Number of sets: ?**

# Practice Problem 4: Solution

Here is the series of address references (in hex) to a cache of size 512 bytes. You are asked to **determine the configuration of the cache**. Assume 12-bit addresses

0x310 – Miss

0x30f – Miss

0x510 – Miss

0x31f – Hit

0x72d – Miss

0x72f – Hit

0x320 – Miss

0x520 – Miss

0x720 - Miss

<u>Determine block size</u>

First hit must be brought in by another miss

Take closest address: 0x310, so know block size must be at least 16 bytes so 0x31f brought in when 0x310 miss occurs

Now, is the block size larger? Know that 0x30f was a miss, thus 0x310 and 0x30f not in the same block. Thus, block size must be <= 16 bytes

Thus Block Size = 16 bytes

# Practice Problem 4 : Solution (2)

Here is the series of address references (in hex) to a cache of size 512 bytes. You are asked to **determine the configuration of the cache**. Assume 12-bit addresses

0x310 – Miss

0x30f – Miss

0x510 – Miss

0x31f – Hit

0x72d – Miss

0x72f – Hit

0x320 – Miss

0x520 – Miss

0x720 - Miss

<u>Determine associativity</u>

Assume direct mapped: 3-bit tag, 5-bit index, 4-bit offset.
If direct mapped 0x310 and 0x510 would both map to index 17,
Thus 0x31f could not be a hit. So, not direct mapped.

Assume 2-way associative: 4-bit tag, 4-bit index, 4-bit offset
This fixes the green accesses, and allows 0x31f to be a hit.

What about > 2-way associative?
Now we also know that 0x720 is a miss even though 3 accesses earlier 0x72f was a hit, and thus it is in the cache. The intervening 2 accesses must kick it out, 0x320 and 0x520. Both go to set 2. If the associativity was > 2, then 0x720 would be a hit. So, must conclude that cache is 2-way associative.

Lastly, <u>number of sets = 512 / (2 * 16) = 16</u>