

CSE 31

Computer Organization

Lecture 20 – Caches (2)

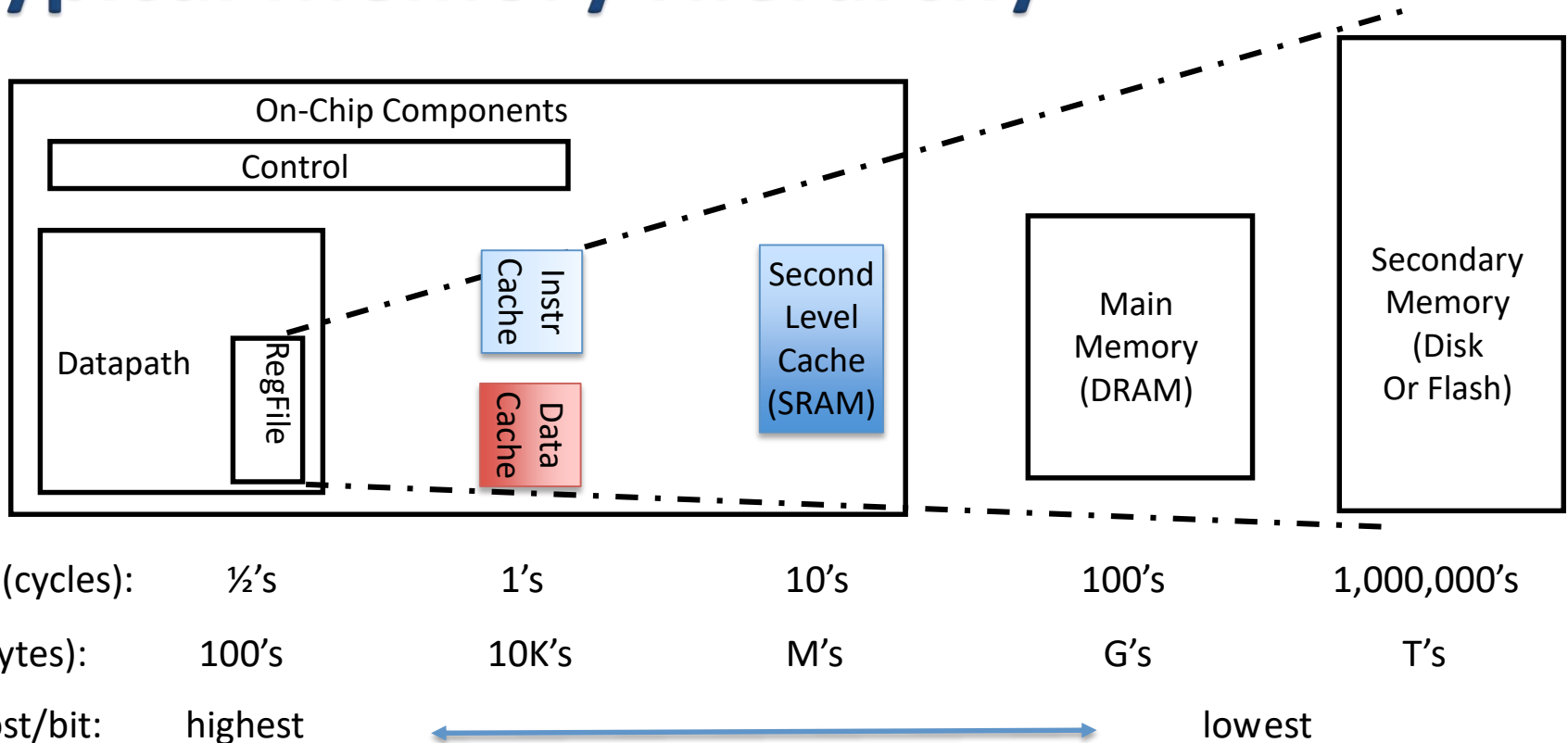
Announcement

- ▶ Lab #9 this week
 - Due in one week
- ▶ Project #2 out Friday
 - Due Monday (12/3)
 - Don't start late, you won't have time!
- ▶ Reading assignment
 - Chapter 5.1 – 5.6 of zyBooks (Reading Assignment #6)
 - Make sure to do the Participation Activities
 - Due Monday (11/26)

Announcement

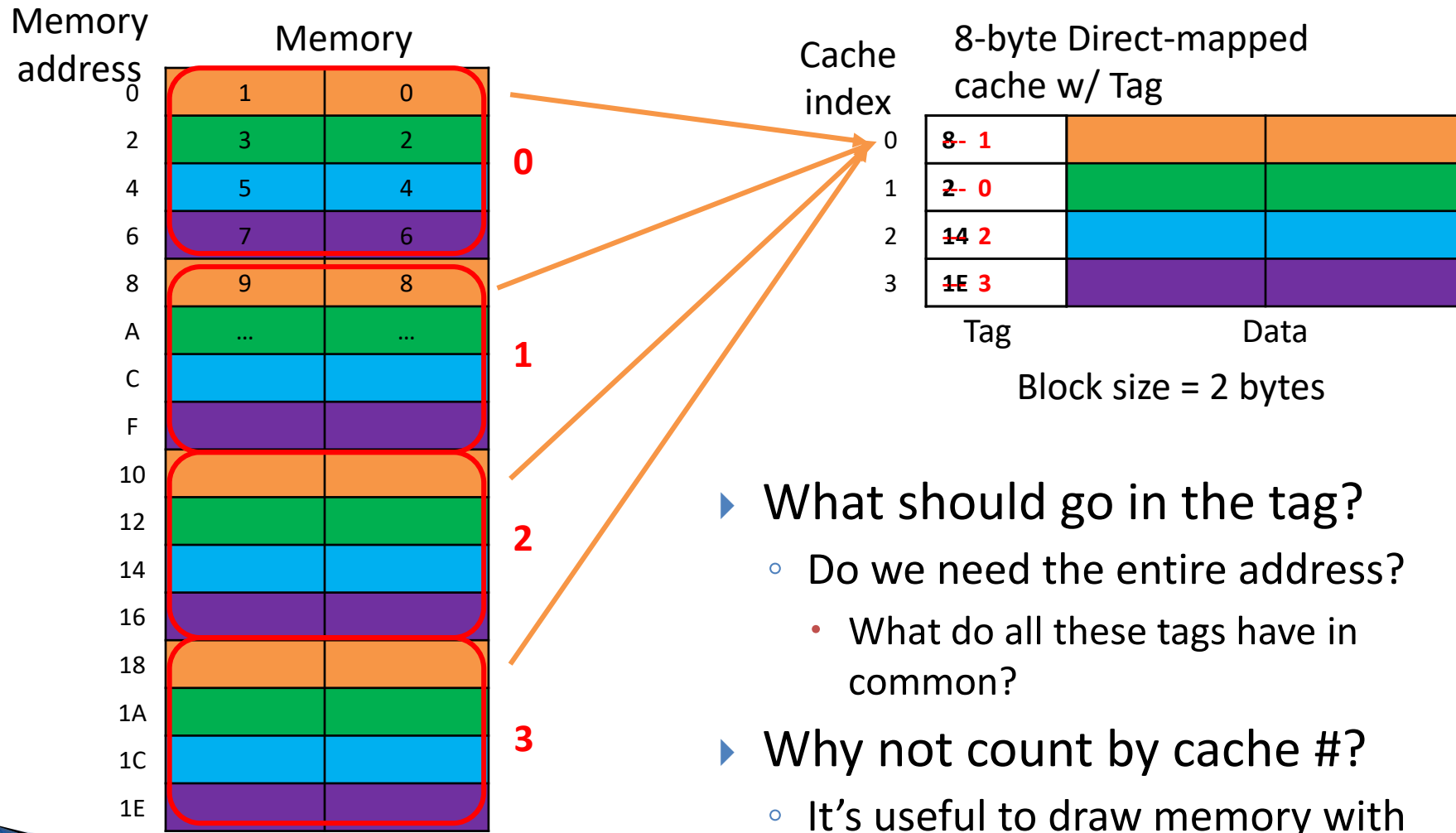
- ▶ Midterm Exam 2
 - 11/28 (Wednesday, in lecture) Not 11/7 as scheduled
 - Lectures #8 - #18
 - HW #3 - #5
 - Practice exam in CatCourses
 - Closed book
 - 1 sheet of note (8.5" x 11")
 - MIPS reference sheet will be provided

Typical Memory Hierarchy



- ▶ **Principle of locality + memory hierarchy** presents programmer with as much memory as is available in the *cheapest* technology at the speed offered by the *fastest* technology

Direct-Mapped Cache (4/4)



- ▶ What should go in the tag?
 - Do we need the entire address?
 - What do all these tags have in common?
- ▶ Why not count by cache #?
 - It's useful to draw memory with the same width as the block size

Issues with Direct-Mapped

- ▶ Since multiple memory addresses map to same cache index, how do we tell which one is in there?
- ▶ What if we have a block size > 1 byte?
 - Answer: divide memory address into three fields

t t t t t t t t t t t t t t t t t t	i i i i i i i i i i	o o o o
-------------------------------------	---------------------	---------

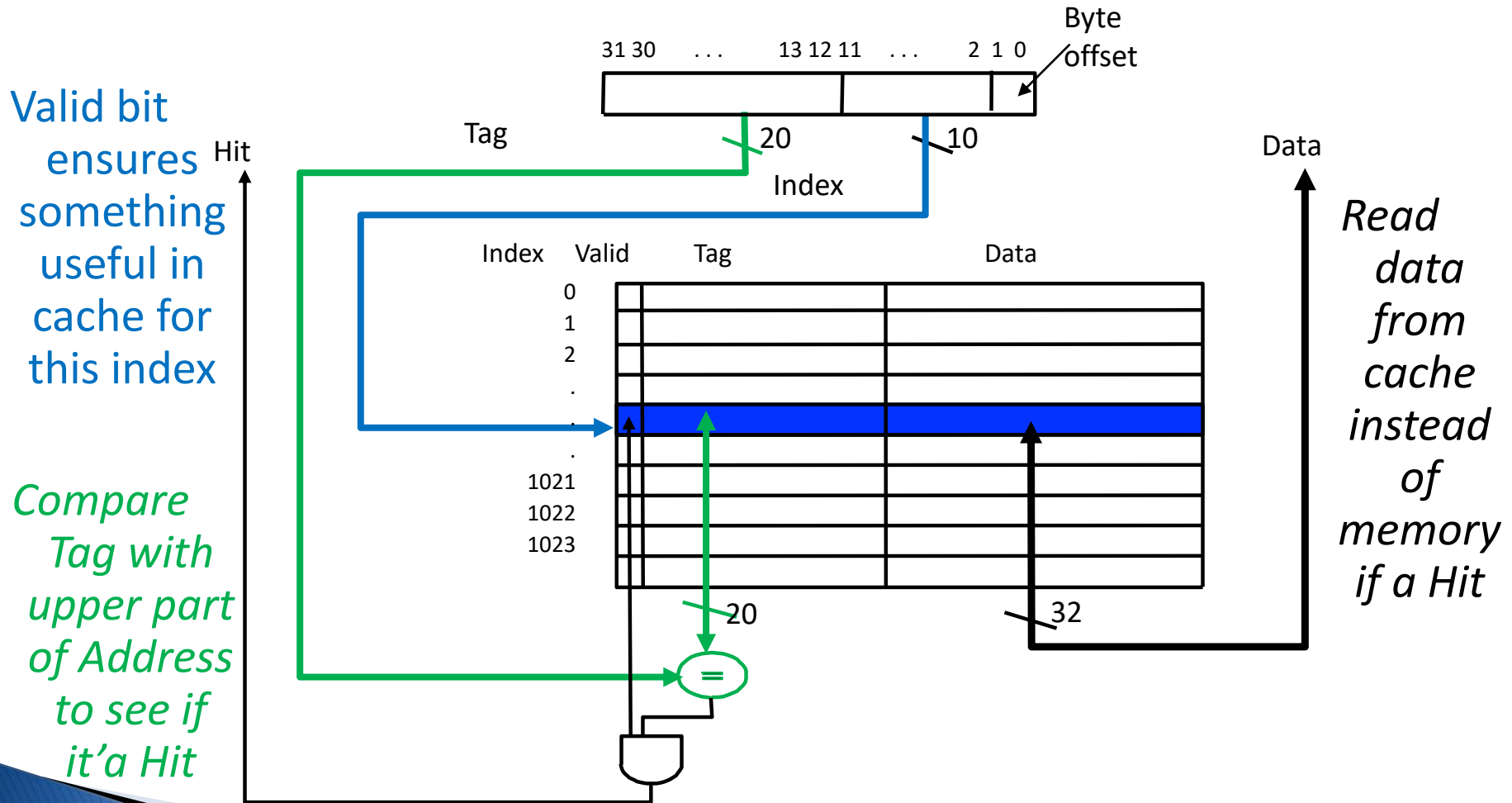
Tag to check if it has the correct block

Index to select block

Byte
offset
within
block

Direct Mapped Cache Example

One word blocks, cache size = 1K words (or 4KB)

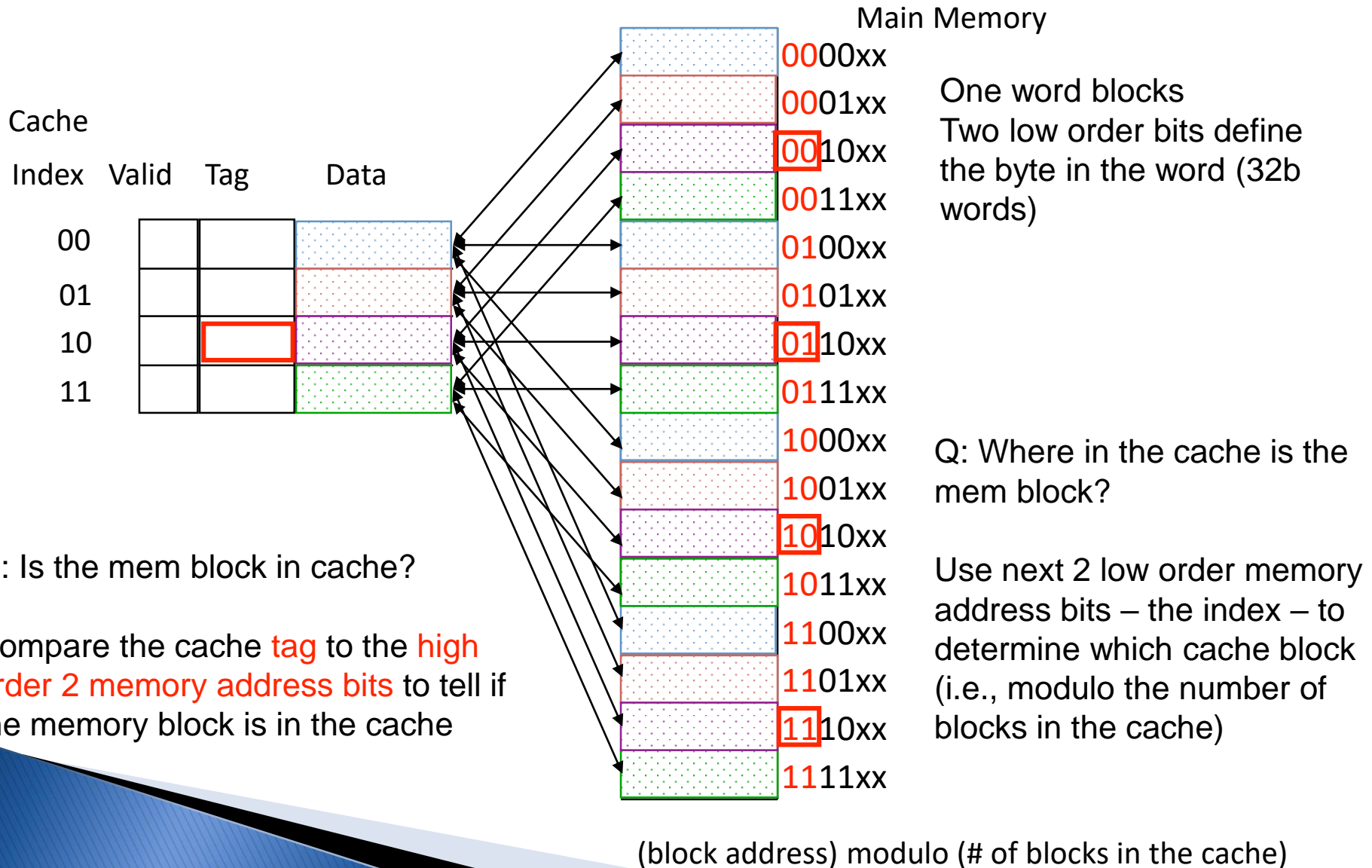


What kind of locality are we taking advantage of?

Caching Terminology

- ▶ When reading memory, 3 things can happen:
 - cache hit:
 - cache block is valid and contains proper address, so read desired word
 - cache miss:
 - nothing in cache at appropriate block, so fetch from memory
 - cache miss, block replacement:
 - wrong data is in cache at appropriate block, so discard it and fetch desired data from memory (cache always copy)

Caching: A Simple First Example



Direct Mapped Cache

Consider the main memory word reference string

Start with an empty cache - all blocks
initially marked as not valid

0 1 2 3 4 3 4 15
0000 0001 0010 0011 0100 0011 0100 1111

Address 0

Direct Mapped Cache

Consider the main memory word reference string

Start with an empty 4-word cache - all blocks initially marked as not valid

0 1 2 3 4 3 4 15
0000 0001 0010 0011 0100 0011 0100 1111

0 miss

00	Mem(0)

- 1 requests, 1 miss

Direct Mapped Cache

Consider the main memory word reference string

Start with an empty cache - all blocks initially marked as not valid

0 1 2 3 4 3 4 15

0000 0001 0010 0011 0100 0011 0100 1111

0 miss

00	Mem(0)

1 miss

00	Mem(0)
00	Mem(1)

2 miss

00	Mem(0)
00	Mem(1)
00	Mem(2)

3 miss

00	Mem(0)
00	Mem(1)
00	Mem(2)
00	Mem(3)

4 miss

00	Mem(0)
00	Mem(1)
00	Mem(2)
00	Mem(3)

3 hit

01	Mem(4)
00	Mem(1)
00	Mem(2)
00	Mem(3)

4 hit

01	Mem(4)
00	Mem(1)
00	Mem(2)
00	Mem(3)

15 miss

01	Mem(4)
00	Mem(1)
00	Mem(2)
00	Mem(3)

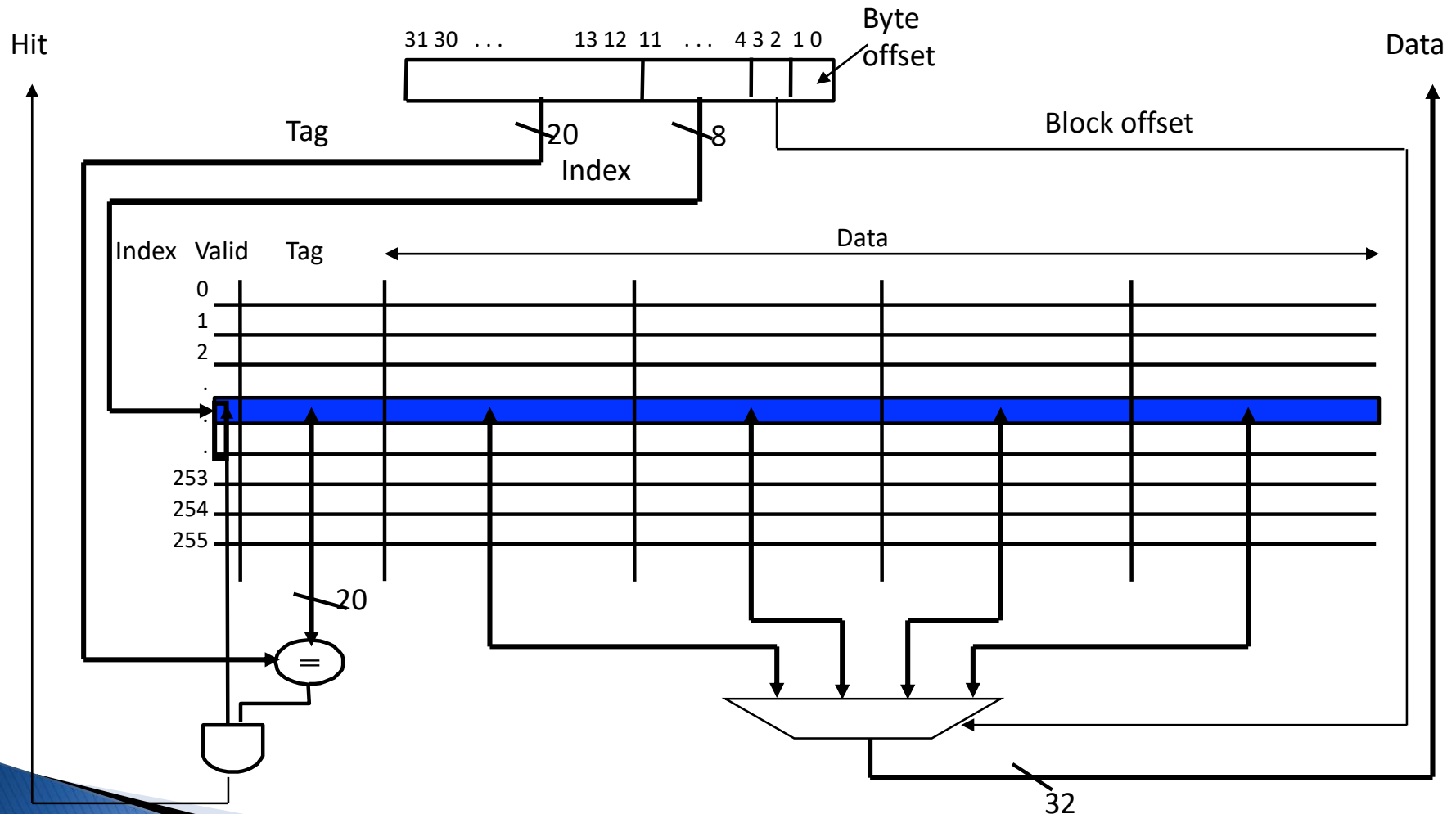
11

15

- 8 requests, 6 misses

Multiword Block Direct Mapped Cache

Four words/block, cache size = 1K words



What kind of locality are we taking advantage of?

Taking Advantage of Spatial Locality

Let cache block hold more than one word

Start with an empty cache - all blocks initially marked as not valid

0 1 2 3 4 3 4 15

0 miss

00	Mem(1)	Mem(0)

1 hit

00	Mem(1)	Mem(0)

2 miss

00	Mem(1)	Mem(0)
00	Mem(3)	Mem(2)

3 hit

00	Mem(1)	Mem(0)
00	Mem(3)	Mem(2)

01 5 4 miss 4

00	Mem(1)	Mem(0)
00	Mem(3)	Mem(2)

3 hit

01	Mem(5)	Mem(4)
00	Mem(3)	Mem(2)

4 hit

01	Mem(5)	Mem(4)
00	Mem(3)	Mem(2)

15 miss

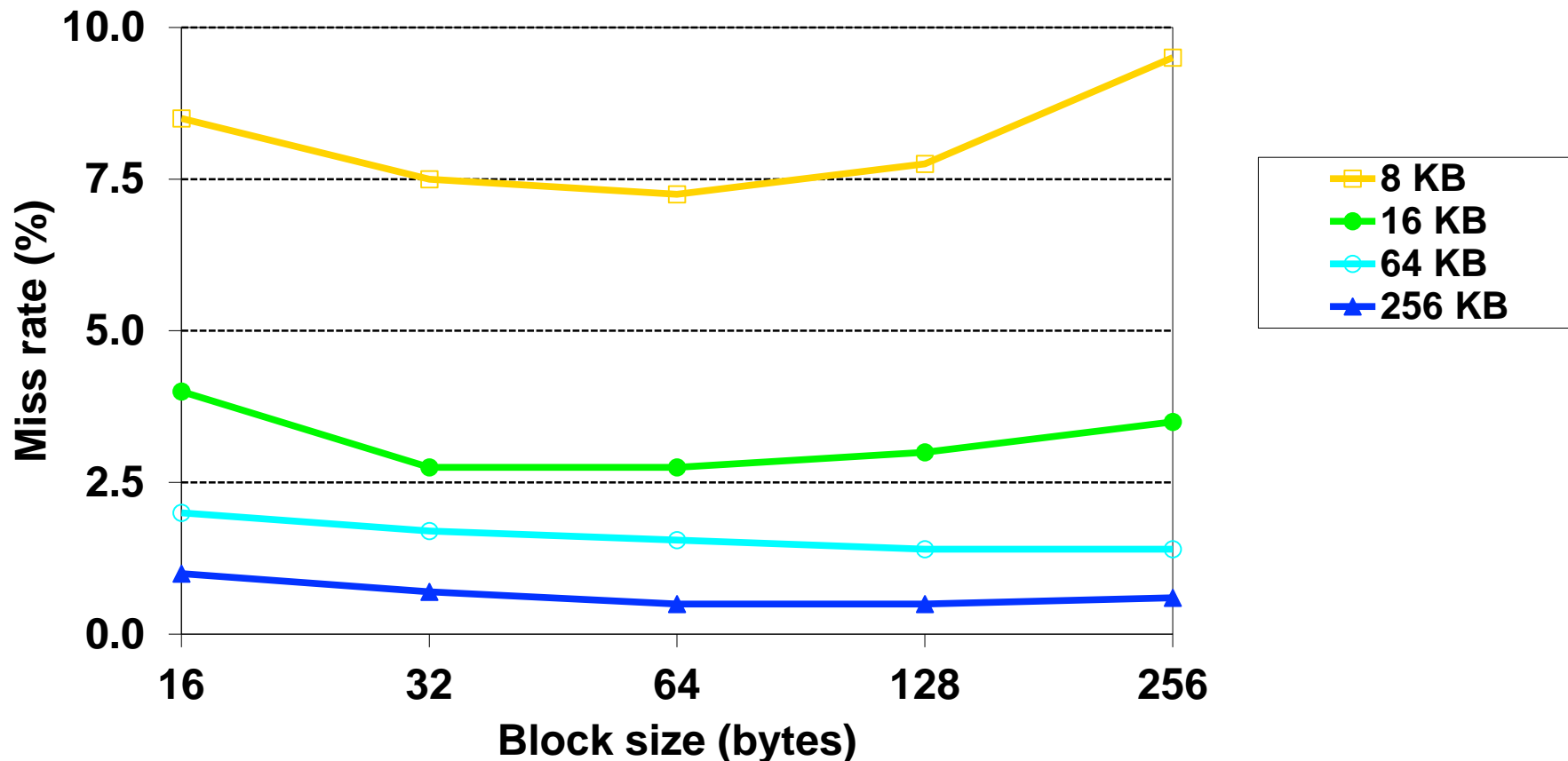
11

01	Mem(5)	Mem(4)
00	Mem(3)	Mem(2)

15 14

- 8 requests, 4 misses

Miss Rate vs Block Size vs Cache Size



Miss rate goes up if the block size becomes a significant fraction of the cache size because the number of blocks that can be held in the same size cache is smaller (increasing capacity misses)

Average Memory Access Time (AMAT)

- ▶ Average Memory Access Time (AMAT) is the average to access memory considering both hits and misses

$$\text{AMAT} = \text{Time for a hit} + \text{Miss rate} \times \text{Miss penalty}$$

- ▶ What is the AMAT for a processor with a 200 psec clock, a miss penalty of 50 clock cycles, a miss rate of 0.02 misses per instruction and a cache access time of 1 clock cycle?

$$1 + 0.02 \times 50 = 2 \text{ clock cycles}$$

$$\text{Or } 2 \times 200 = 400 \text{ psecs}$$

- ▶ Potential impact of much larger cache on AMAT?

- 1) Lower Miss rate
- 2) Longer Access time (Hit time): smaller is faster

At some point, increase in hit time for a larger cache may overcome the improvement in hit rate, yielding a decrease in performance

Block Size Tradeoff (1/3)

- ▶ Benefits of Larger Block Size
 - **Spatial Locality:** if we access a given word, we're likely to access other nearby words soon
 - Very applicable with Stored-Program Concept: if we execute a given instruction, it's likely that we'll execute the next few as well
 - Works nicely in sequential array accesses too

Block Size Tradeoff (2/3)

▶ Drawbacks of Larger Block Size

- Larger block size means **larger miss penalty**
 - on a miss, takes longer time to load a new block from next level
- If block size is too big relative to cache size, then there are too few blocks
 - Result: miss rate goes up

▶ In general, minimize

Average Memory Access Time (AMAT)

= Hit Time + Miss Penalty x Miss Rate

Block Size Tradeoff (3/3)

- ▶ Hit Time

- time to find and retrieve data from current level cache

- ▶ Miss Penalty

- average time to retrieve data on a current level miss (includes the possibility of misses on successive levels of memory hierarchy)

- ▶ Hit Rate

- % of requests that are found in current level cache

- ▶ Miss Rate

- $1 - \text{Hit Rate}$

Extreme Example: One Big Block

Valid Bit

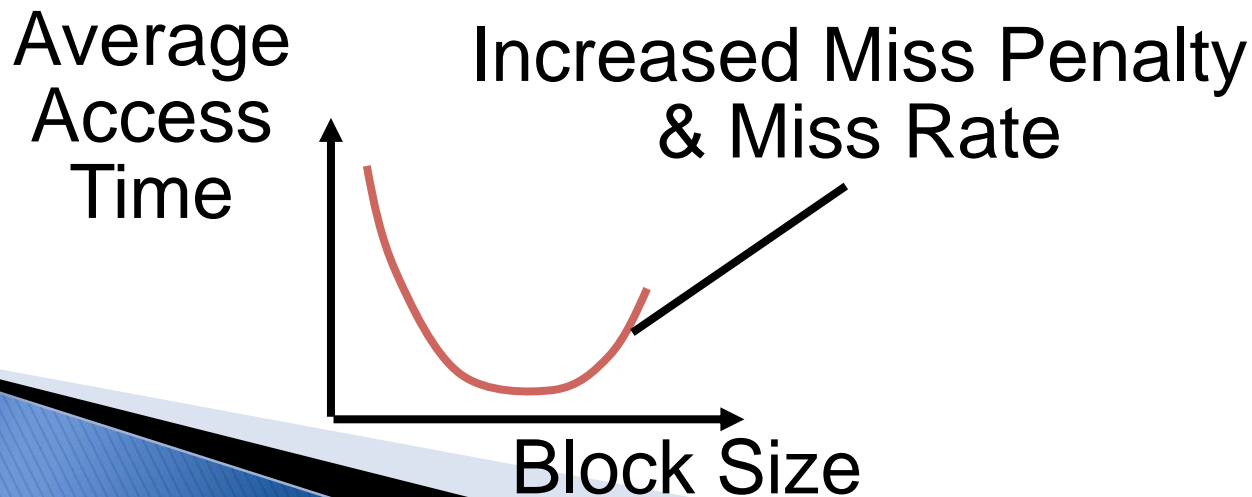
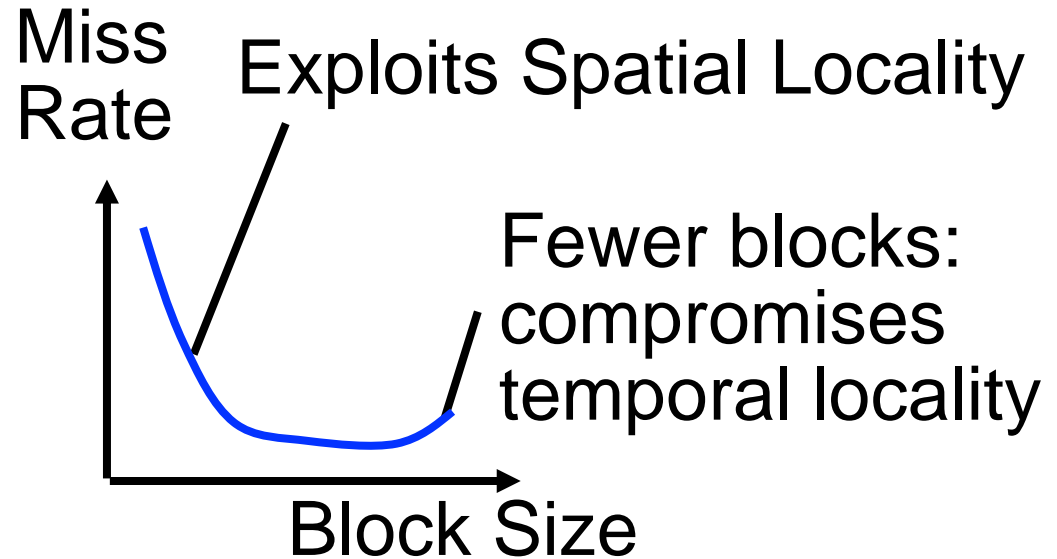
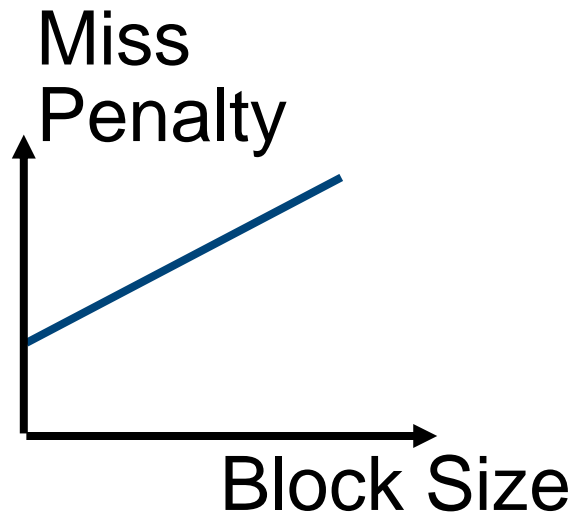
Tag

Cache Data



- ▶ Cache Size = 4 bytes Block Size = 4 bytes
 - Only **ONE** entry (row) in the cache!
- ▶ If item accessed, likely accessed again soon
 - But unlikely will be accessed again immediately!
- ▶ The next access will likely to be a miss again
 - Continually loading data into the cache but discard data (force out) before use it again
 - Nightmare for cache designer: **Ping Pong Effect**

Block Size Tradeoff Conclusions



What to do on a write hit?

▶ Write-through

- update the word in cache block and corresponding word in memory

▶ Write-back

- update word in cache block
- allow memory word to be “stale”
- add ‘dirty’ bit to each block indicating that memory needs to be updated when block is replaced
- OS flushes cache before I/O...

▶ Performance trade-offs?

Types of Cache Misses (1/2)

- ▶ “Three Cs” Model of Misses
- ▶ 1st C: **Compulsory Misses**
 - occur when a program is first started
 - cache does not contain any of that program’s data yet, so misses are bound to occur
 - can’t be avoided easily, so won’t focus on these in this course

Types of Cache Misses (2/2)

▶ 2nd C: Conflict Misses

- miss that occurs because two distinct memory addresses map to the same cache location
- two blocks (which happen to map to the same location) can keep overwriting each other
- big problem in direct-mapped caches
- how do we lessen the effect of these?

▶ Dealing with Conflict Misses

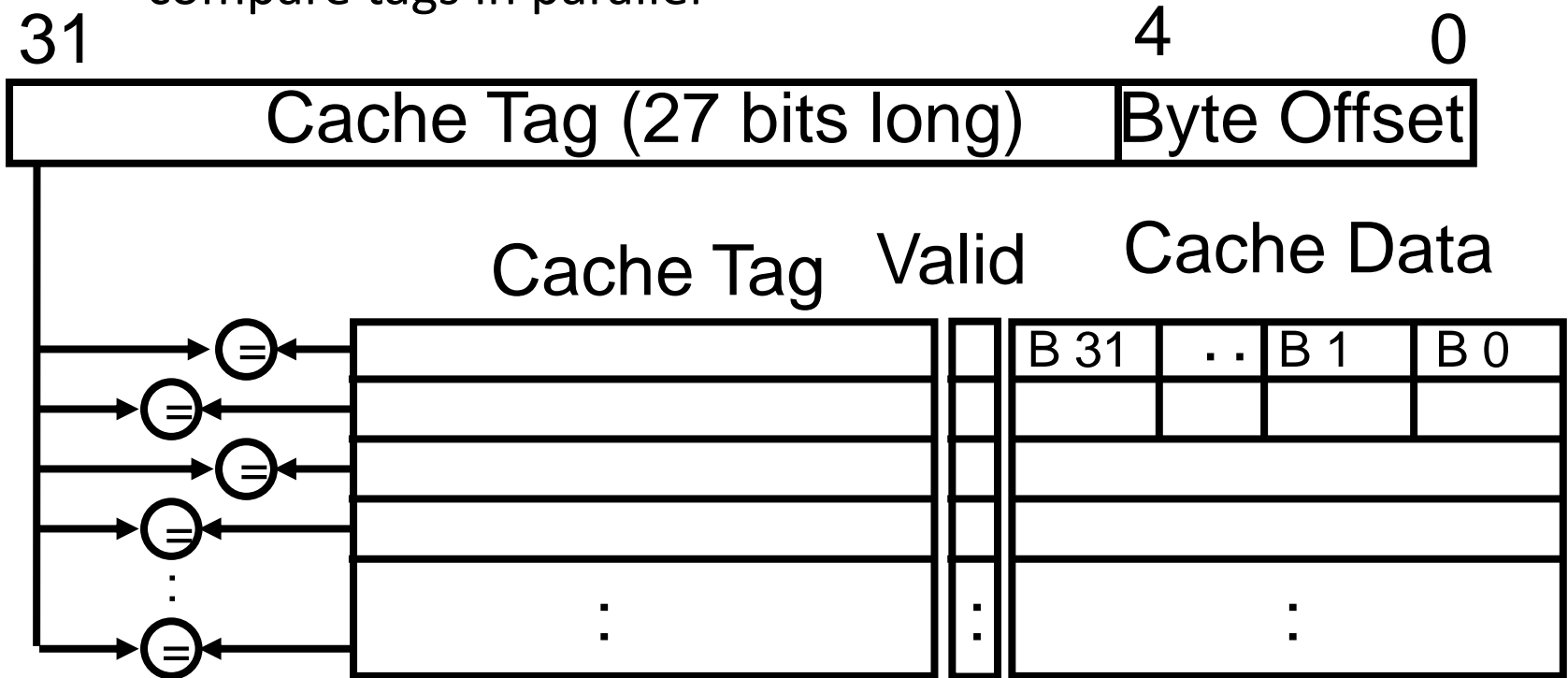
- Solution 1: Make the cache size bigger
 - Fails at some point
- Solution 2: Multiple distinct blocks can fit in the same cache Index
 - How???

Fully Associative Cache (1/3)

- ▶ Memory address fields:
 - Tag: same as before
 - Offset: same as before
 - Index: non-existent
- ▶ What does this mean?
 - no “rows”: any block can go anywhere in the cache
 - must compare with all tags in entire cache to see if data is there

Fully Associative Cache (2/3)

- ▶ Fully Associative Cache (e.g., 32 B block)
 - compare tags in parallel



Fully Associative Cache (3/3)

- ▶ Benefit of Fully Assoc. Cache
 - No Conflict Misses (since data can go anywhere)
- ▶ Drawbacks of Fully Assoc. Cache
 - Need hardware comparator for every single entry: if we have a 64KB of data in cache with 4B entries, we need 16K comparators: infeasibly high cost

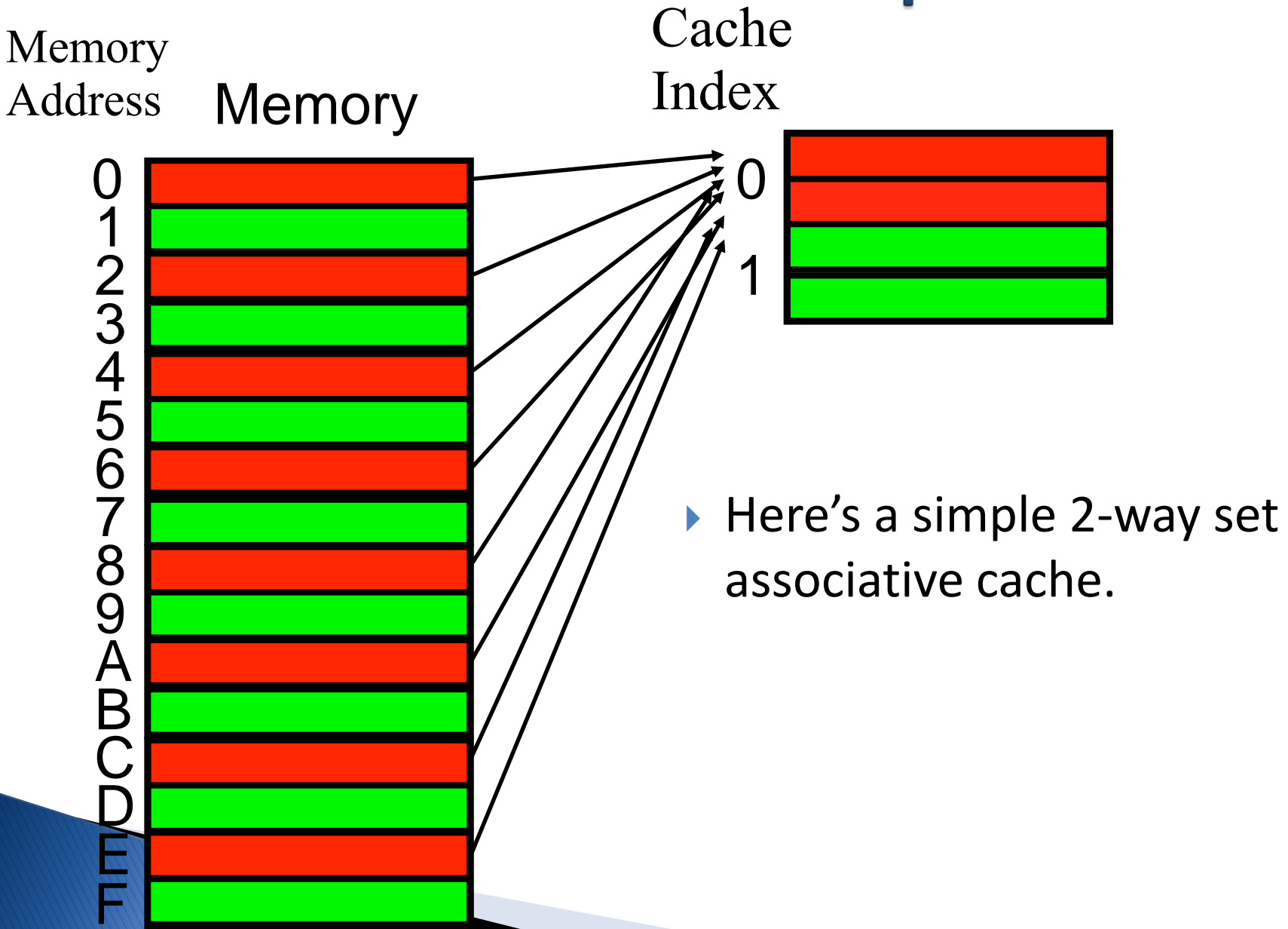
Final Type of Cache Miss

- ▶ 3rd C: Capacity Misses
 - miss that occurs because the cache has a limited size
 - miss that would not occur if we increase the size of the cache
 - sketchy definition, so just get the general idea
- ▶ This is the primary type of miss for Fully Associative caches.

N-Way Set Associative Cache (1/3)

- ▶ Memory address fields:
 - **Tag**: same as before
 - **Offset**: same as before
 - **Index**: points us to the correct “row” (called a set in this case)
- ▶ So what’s the difference?
 - each set contains multiple blocks
 - once we’ve found correct set, must compare with all tags in that set to find our data
 - Hybrid of direct-mapped and fully associative

Associative Cache Example



N-Way Set Associative Cache (2/3)

▶ Basic Idea

- cache is direct-mapped w/respect to sets
- each set is fully associative with N blocks in it

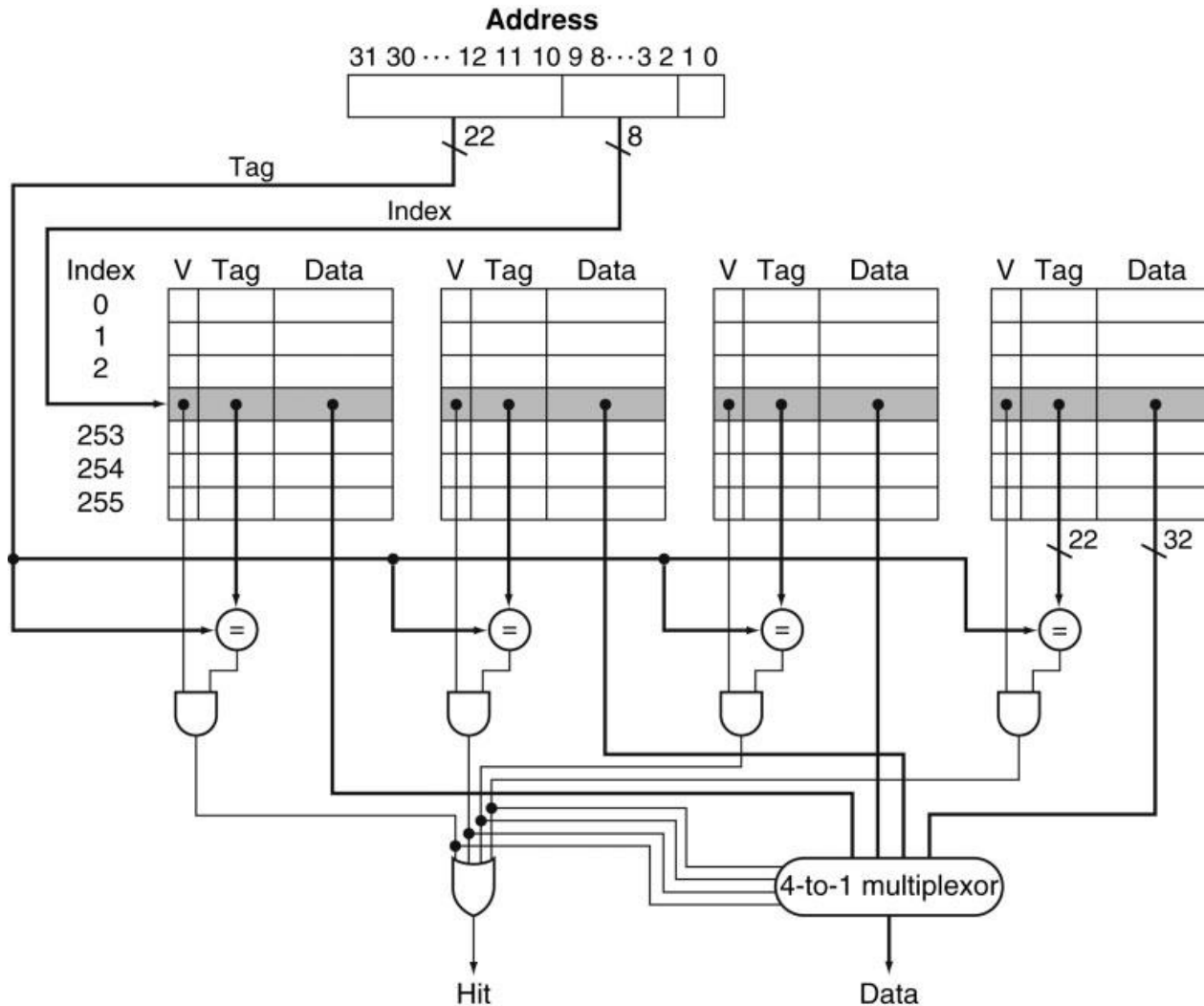
▶ Given memory address:

- Find correct set using Index value.
- Compare Tag with all Tag values in the determined set.
- If a match occurs, hit!, otherwise a miss.
- Finally, use the offset field as usual to find the desired data within the block.

N-Way Set Associative Cache (3/3)

- ▶ What's so great about this?
 - even a 2-way set assoc. cache avoids a lot of conflict misses
 - hardware cost isn't that bad: only need N comparators
- ▶ In fact, for a cache with M blocks,
 - it's **Direct-Mapped** if it's 1-way set assoc.
 - it's **Fully Assoc** if it's M-way set assoc.
 - so these two are just special cases of the more general set associative design

4-Way Set Associative Cache Circuit



Summary

- ▶ Principle of Locality for Computer Memory
 - ▶ Hierarchy of Memories (speed/size/cost per bit) to Exploit Locality
 - ▶ Cache – copy of data lower level in memory hierarchy
 - ▶ Direct Mapped to find block in cache using Tag field and Valid bit for Hit
 - ▶ Larger caches reduce Miss rate via Temporal and Spatial Locality, but can increase Hit time
 - ▶ Larger blocks to reduces Miss rate via Spatial Locality, but increase Miss penalty
 - ▶ AMAT helps balance Hit time, Miss rate, Miss penalty
- 