CSE 31
Computer Organization

Lecture 19 – Cache (1)

Announcement

- Lab #9
  - Due in 1 week
- Project #2
  - Start working on it during lab this week
  - Due Monday (4/29)
- HW #6 in CatCourses
  - Due Monday (4/22) at 11:59pm
- Reading assignment
  - Chapter 6.4-6.7 of zyBooks
    - Make sure to do the Participation Activities
    - Due Friday (4/19) at 11:59pm
  - Chapter 5.1-5.6of zyBooks
    - Make sure to do the Participation Activities
    - Due Friday (4/26) at 11:59pm

Announcement

- Midterm Exam 2
  - 4/24 (Wednesday, in lecture) Not 4/17 as scheduled
  - Lectures #8 - #18
  - HW #2 - #6
  - Practice exam in CatCourses
  - Closed book
  - 1 sheet of note (8.5” x 11”)
  - MIPS reference sheet will be provided

Library Analogy

- Writing a report based on books on reserve
  - E.g., *History of most boring video games*
- Go to library to get reserved book and place on desk in your dorm room
- If need more, check them out and keep on desk
  - But don't return earlier books since might need them later
- You hope this collection of ~10 books on desk enough to write report, despite 10 being only 0.00001% of books in the library
- We can do the same with memory use

Big Idea: Locality

- *Temporal Locality* (locality in time)
  - Go back to same book on desk multiple times
  - If a memory location is referenced then it will tend to be referenced again soon
- *Spatial Locality* (locality in space)
  - When go to book shelf, pick up multiple books on *History of most boring video games* since library stores related books together
  - If a memory location is referenced, the locations with nearby addresses will tend to be referenced soon

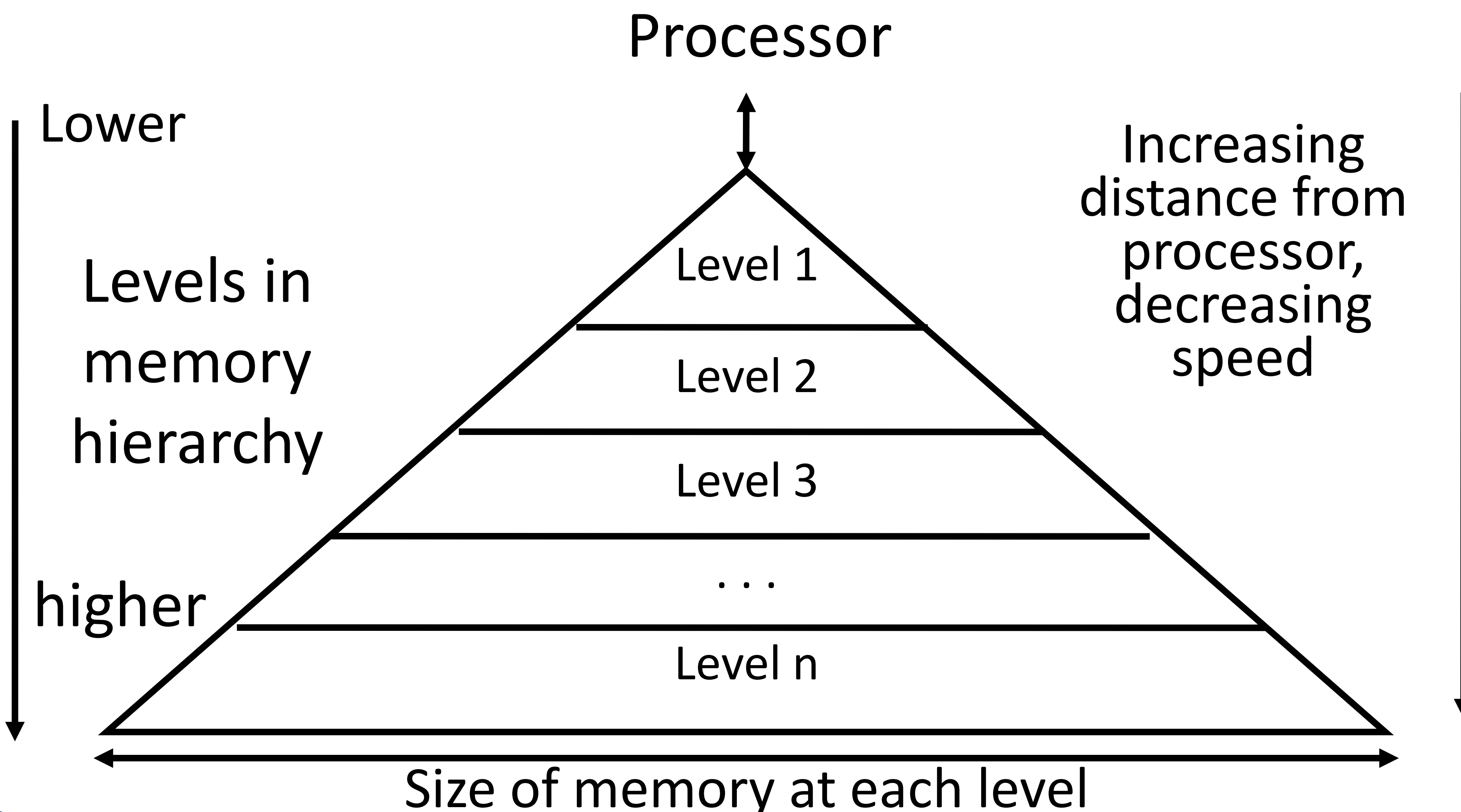
Principle of Locality

- *Principle of Locality*: Programs access small portion of address space at any instant of time
- What **program** structures lead to temporal and spatial locality in code?
- What about in **data**?

How does hardware exploit principle of locality?

- Offer a hierarchy of memories where
  - closest to processor is fastest
    (and most expensive per bit so smallest)
    - Books on your desk
  - furthest from processor is largest
    (and least expensive per bit so slowest)
    - Books in the library
- Goal is to create **illusion** of memory almost **as fast** as fastest memory and almost **as large** as biggest memory of the hierarchy

Big Idea: Memory Hierarchy



As we move to deeper levels the latency goes up and price per bit goes down. Why?

Cache Concept

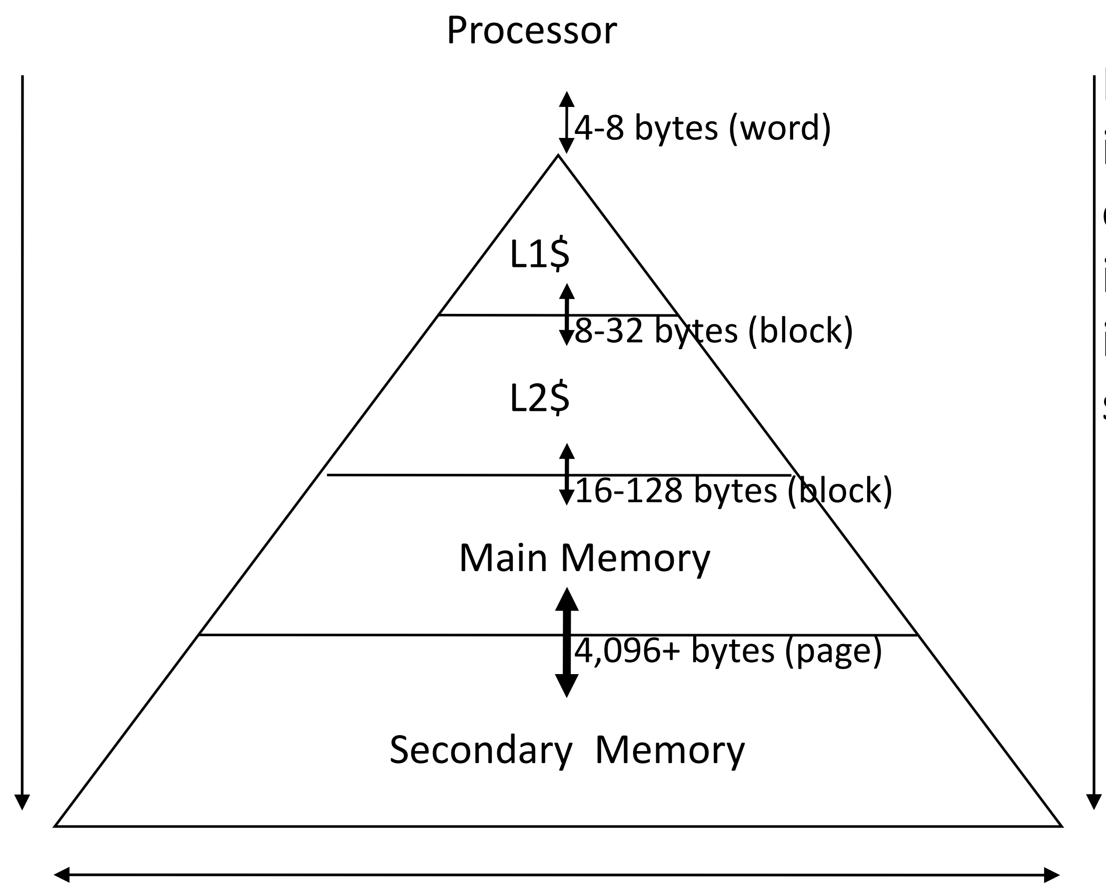
- Processor and memory speed mismatch leads us to add a new level: a memory *cache*
- Implemented with same integrated circuit processing technology as processor, integrated on-chip: faster but more expensive than DRAM memory
- *Cache is a copy of a subset of main memory*
- Modern processors have separate caches for instructions and data, as well as several levels of caches implemented in different sizes
- As a pun, often use $ (“cash”) to abbreviate cache, e.g. D$ = Data Cache, I$ = Instruction Cache

Memory Hierarchy Technologies

- Caches use SRAM (Static RAM) for speed and technology compatibility
  - Fast (typical access times of 0.5 to 2.5 ns)
  - Low density (6 transistor cells), higher power, expensive ($1000 to $2000 per GB in 2018)
  - Static: content will last as long as power is on
- Main memory uses DRAM (Dynamic RAM) for size (density)
  - Slower (typical access times of 50 to 70 ns)
  - High density (1 transistor cells), lower power, cheaper (~$10 per GB in 2017)
  - Dynamic: needs to be “refreshed” regularly (~ every 8 ms)
    - Consumes 1% to 2% of the active cycles of the DRAM

Characteristics of the Memory Hierarchy

Increasing distance from the processor in access time



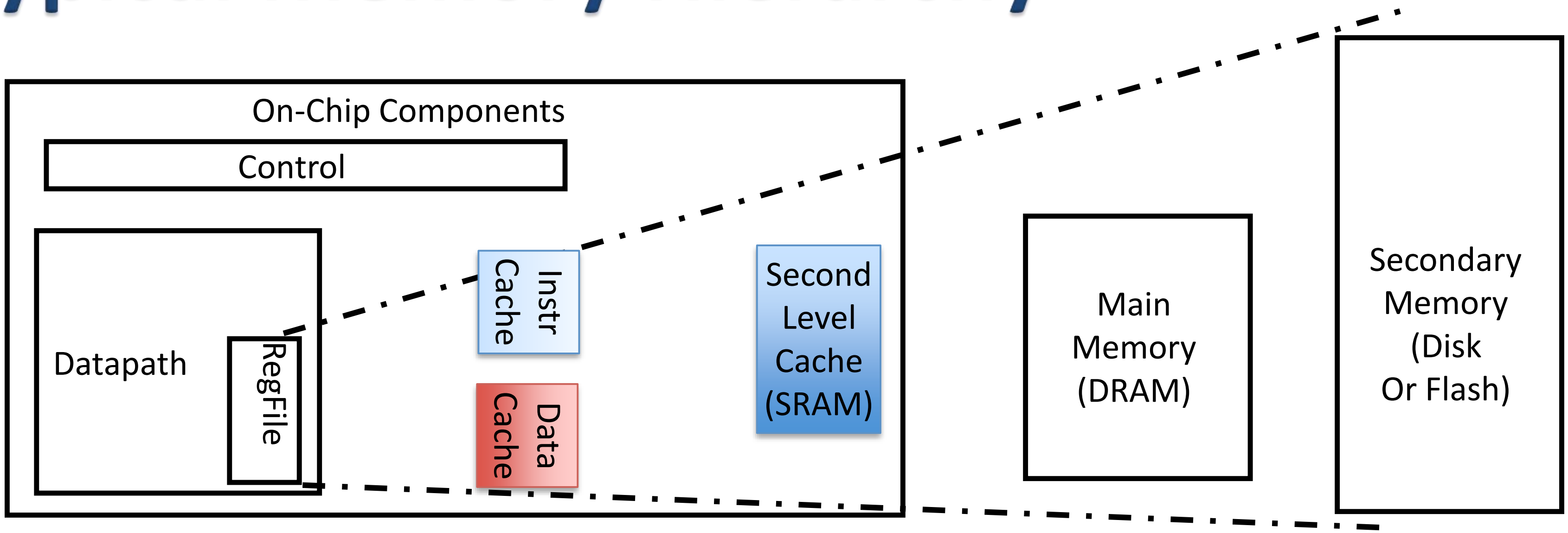
Inclusive– what is in L1$ is a subset of what is in L2$ is a subset of what is in MM that is a subset of is in SM

(Relative) size of the memory at each level

How is the Hierarchy Managed?

- registers $\leftrightarrow$ memory
  - By compiler (or assembly level programmer)
- cache $\leftrightarrow$ main memory
  - By the cache controller hardware
- main memory $\leftrightarrow$ disks (secondary storage)
  - By the operating system (virtual memory)
  - Virtual to physical address mapping assisted by the hardware (TLB)
  - By the programmer (files)
  - (Talk about it later in CSE140)

Typical Memory Hierarchy



Speed (cycles):	½'s	1's	10's	100's	1,000,000's
Size (bytes):	100's	10K's	M's	G's	T's
Cost/bit:	highest				lowest

- 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

So far ...

- Wanted: effect of a large, cheap, fast memory
- Approach: Memory Hierarchy
  - Successively lower levels contain "most used" data from next higher level
  - Exploits *temporal & spatial locality*
- Memory hierarchy follows 2 Design Principles: Smaller is faster and Do common case fast

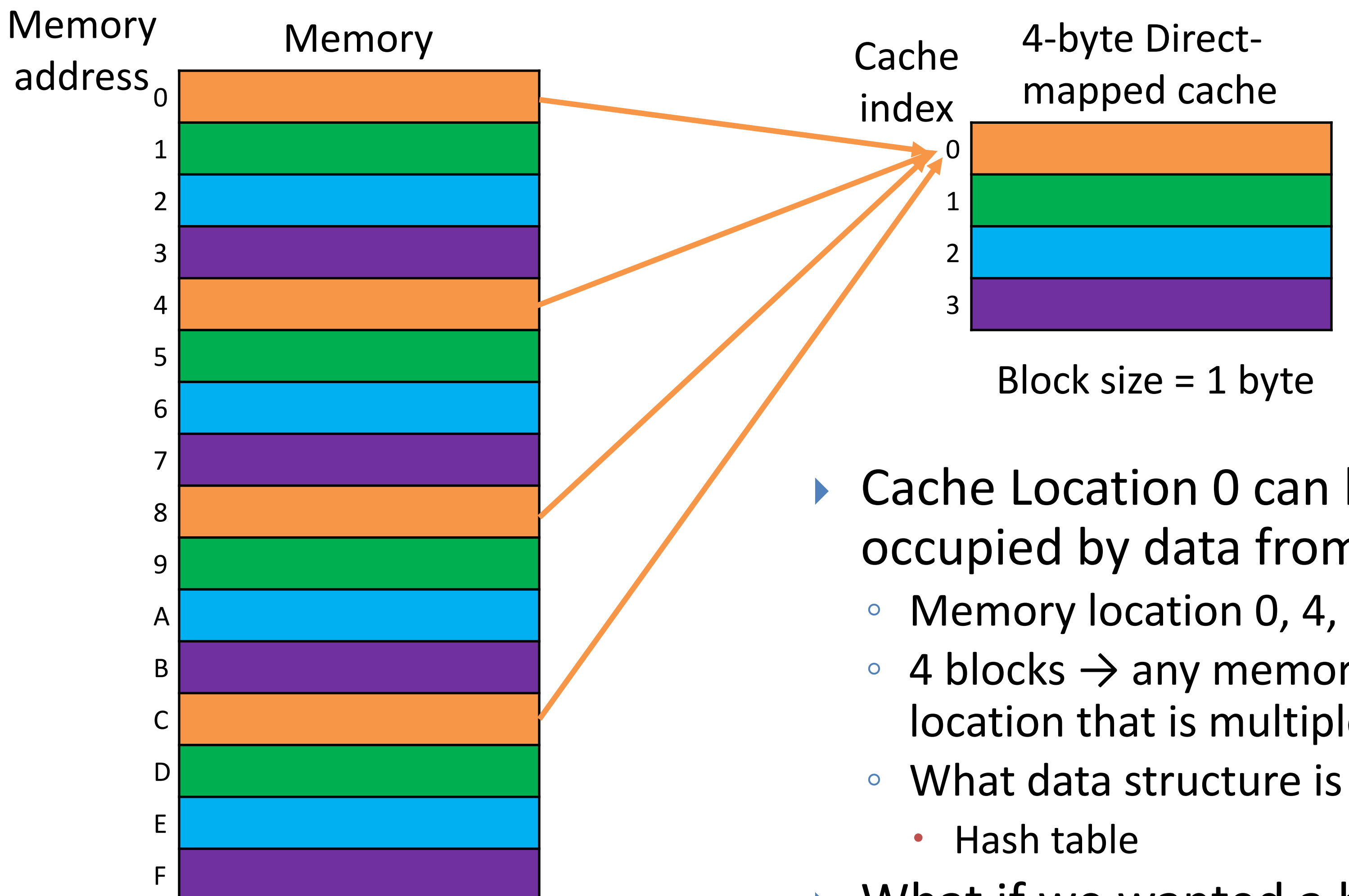
Cache Design Questions

- How best to organize the memory blocks of the cache?
- To which block of the cache does a given main memory address map?
  - Since the cache is a subset of memory, multiple memory addresses can be mapped to the same cache location
- How do we know which blocks of main memory currently have a copy in cache?
- How do we find these copies quickly?

Direct-Mapped Cache (1/4)

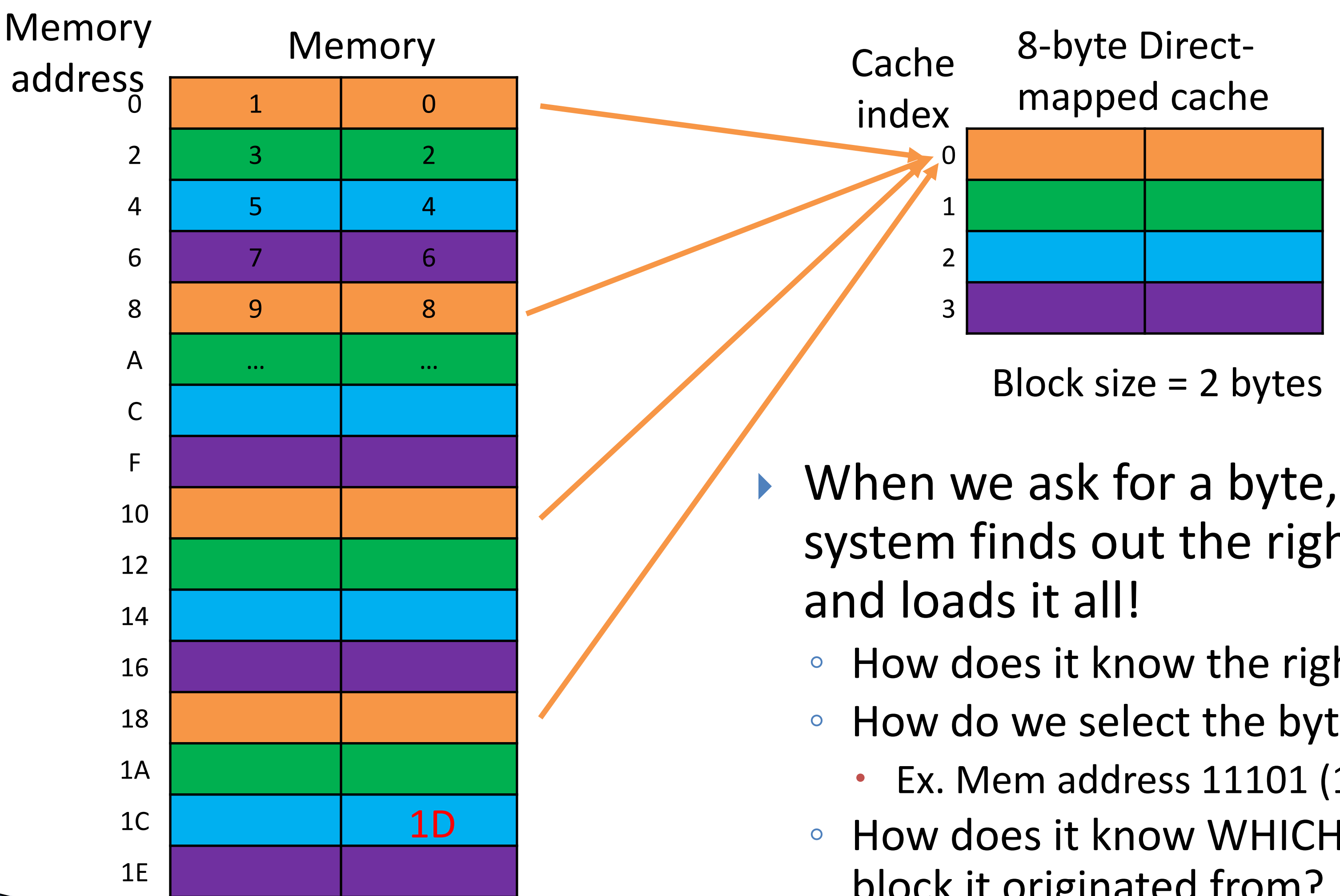
- In a direct-mapped cache, each memory address is associated with one possible block within the cache
  - Therefore, we only need to look in a single location in the cache for the data if it exists in the cache
  - Block is the unit of transfer between cache and memory

Direct-Mapped Cache (2/4)



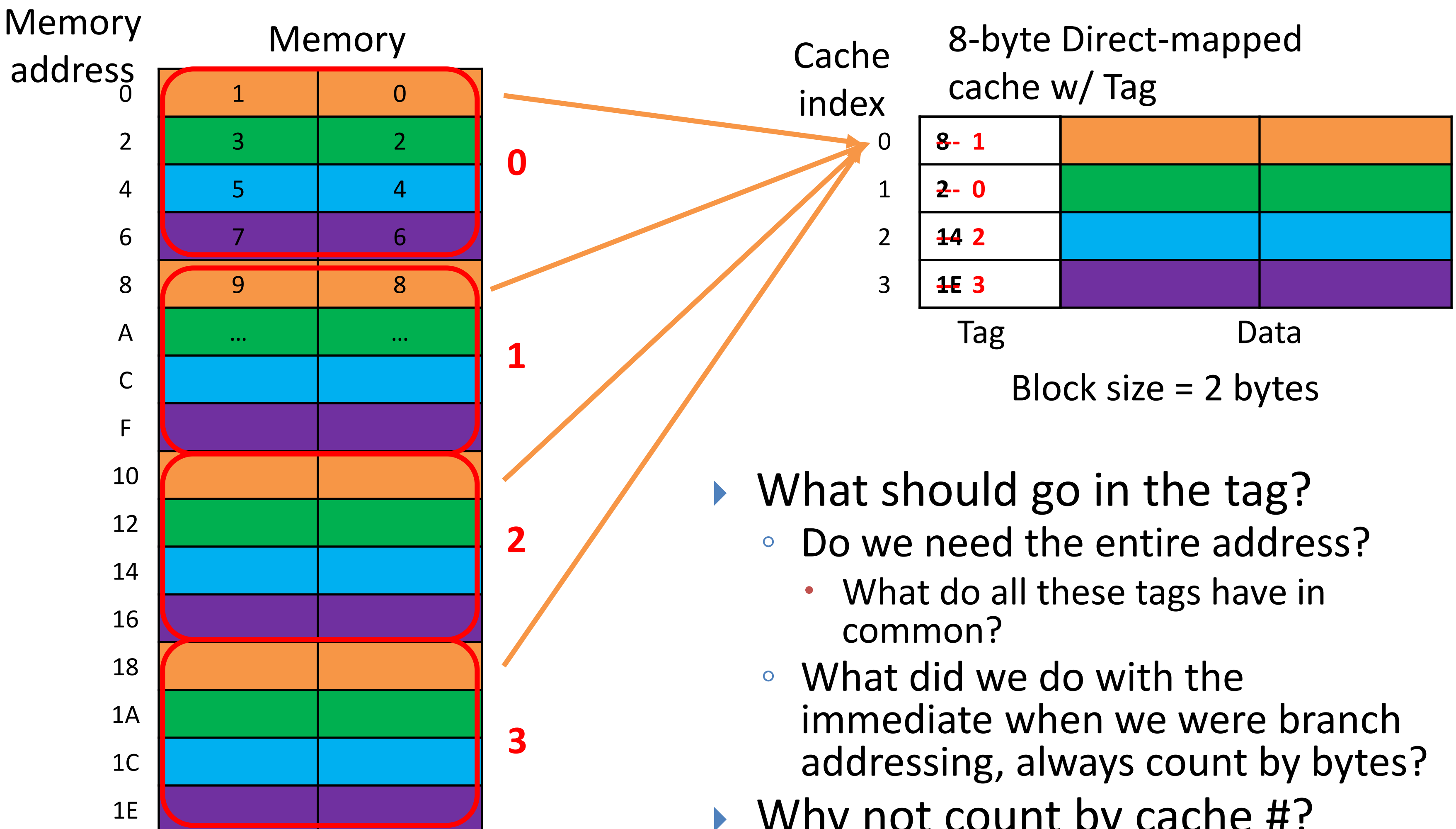
- Cache Location 0 can be occupied by data from:
  - Memory location 0, 4, 8, ...
  - 4 blocks → any memory location that is multiple of 4
  - What data structure is it?
    - Hash table
- What if we wanted a block to be bigger than one byte?

Direct-Mapped Cache (3/4)



- When we ask for a byte, the system finds out the right block and loads it all!
  - How does it know the right block?
  - How do we select the byte?
    - Ex. Mem address 11101 (1D)?
  - How does it know WHICH colored block it originated from?
  - What do you do at baggage claim?

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?
  - What did we do with the immediate when we were branch addressing, always count by bytes?
- 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	oooo
Tag to check if it has the correct block	Index to select block	Byte offset within block

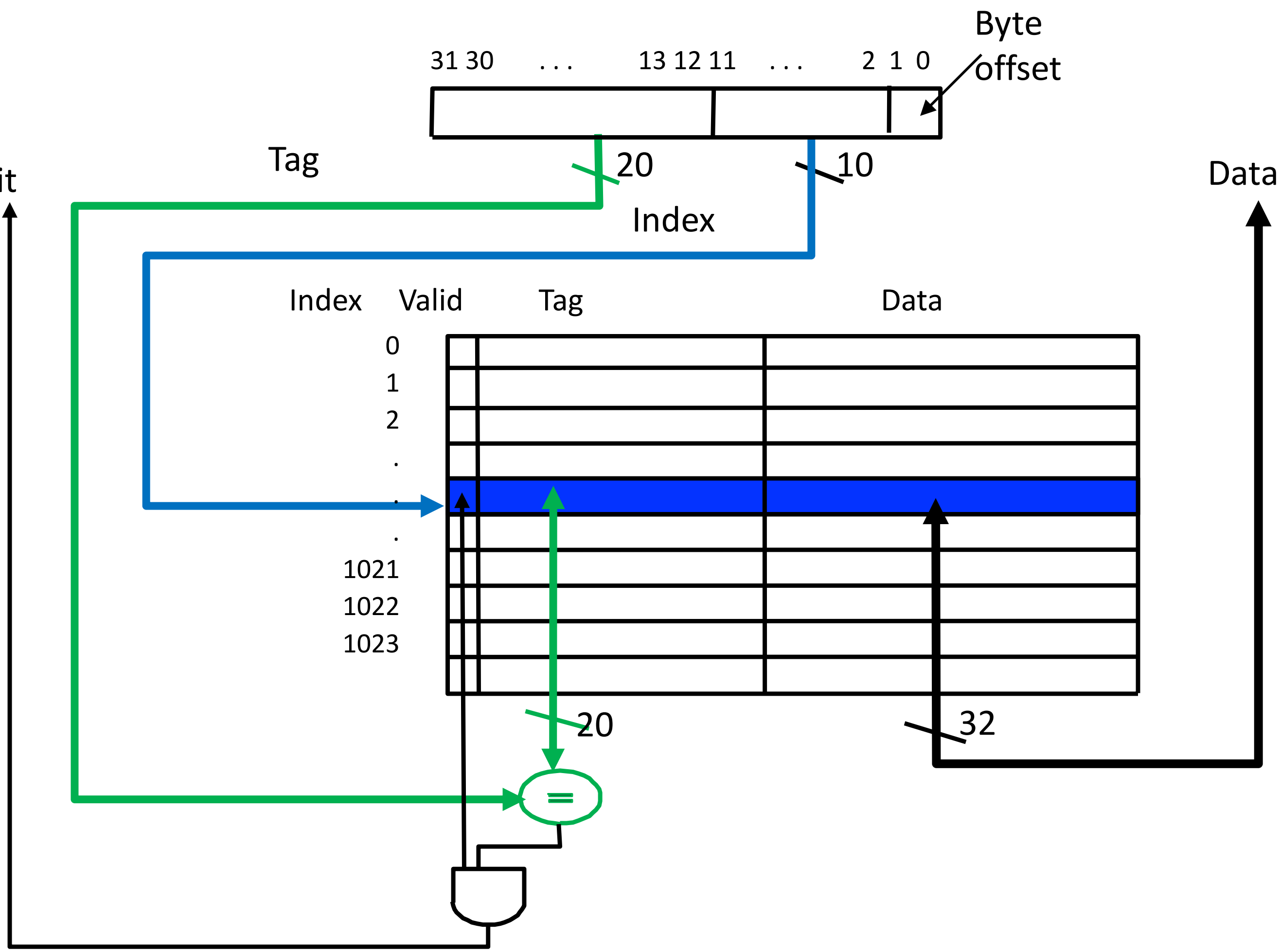
Direct-Mapped Cache Terminology

- All fields are read as unsigned integers
- Index
  - specifies the cache index (which “row”/block of the cache we should look in)
- Offset
  - once we’ve found correct block, specifies which byte within the block we want
- Tag
  - the remaining bits after offset and index are determined; these are used to distinguish between all the memory addresses that map to the same location

Direct Mapped Cache Example

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

Valid bit ensures something useful in cache for this index

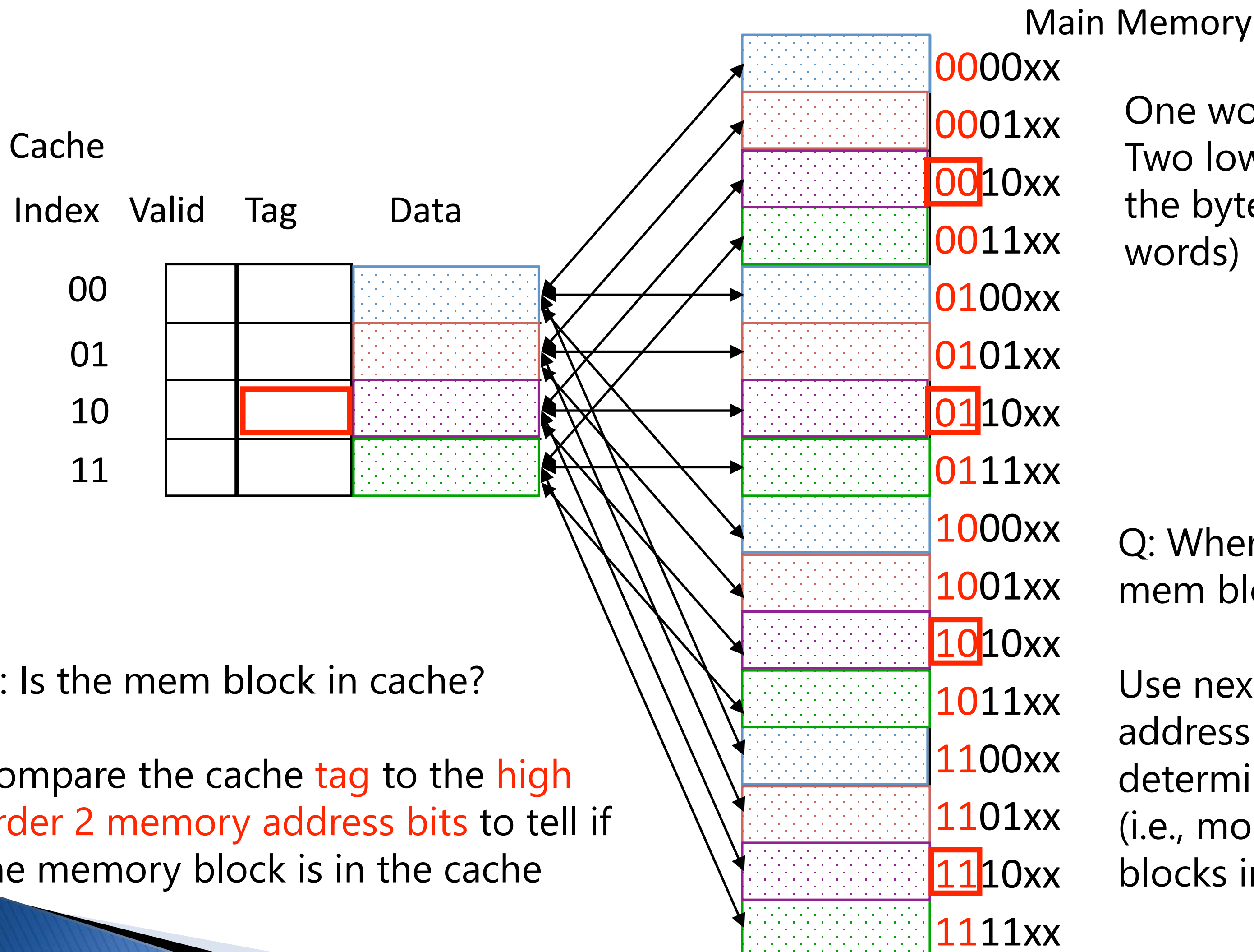


Compare Tag with upper part of Address to see if it'a Hit

Read data from cache instead of memory if a Hit

What kind of locality are we taking advantage of?

Caching: A Simple First Example



One word blocks
Two low order bits define the byte in the word (32b words)

Q: Where in the cache is the mem block?

Use next 2 low order memory address bits – the index – to determine which cache block (i.e., modulo the number of blocks in the cache)

Q: Is the mem block in cache?

Compare the cache tag to the high order 2 memory address bits to tell if the memory block is in the cache

(block address) modulo (# of blocks in the cache)

Direct Mapped Cache

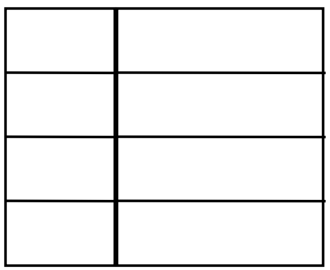
Consider the main memory word reference string

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

01 ~~00~~	Mem(~~0~~) 4
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

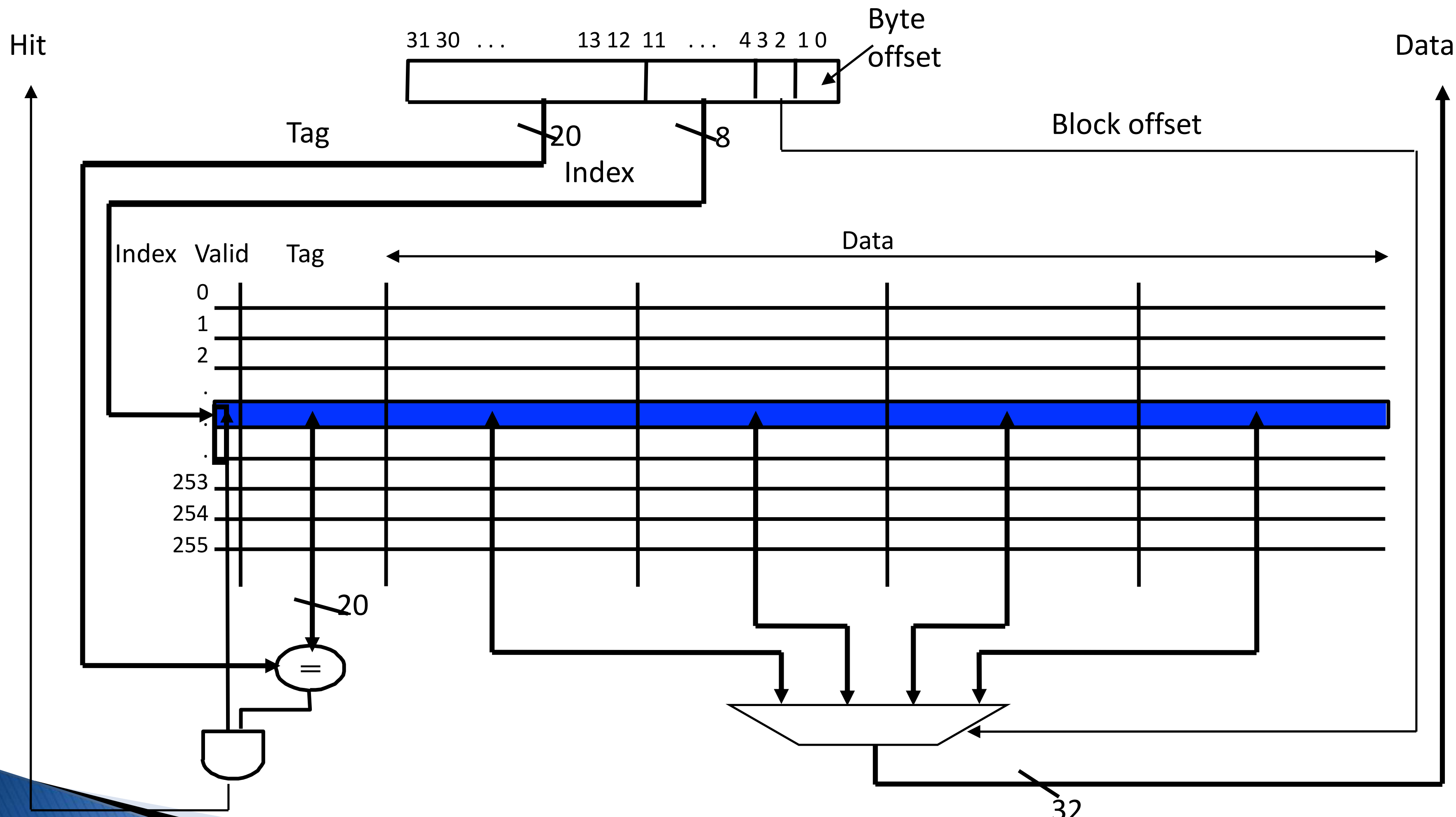
01	Mem(4)
00	Mem(1)
00	Mem(2)
~~00~~ 11	Mem(~~3~~) 15

- 8 requests, 6 misses

How can we reduce the number of 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)

4 miss

01 ~~00~~	5 ~~Mem(1)~~	4 ~~Mem(0)~~
00	Mem(3)	Mem(2)

3 hit

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

4 hit

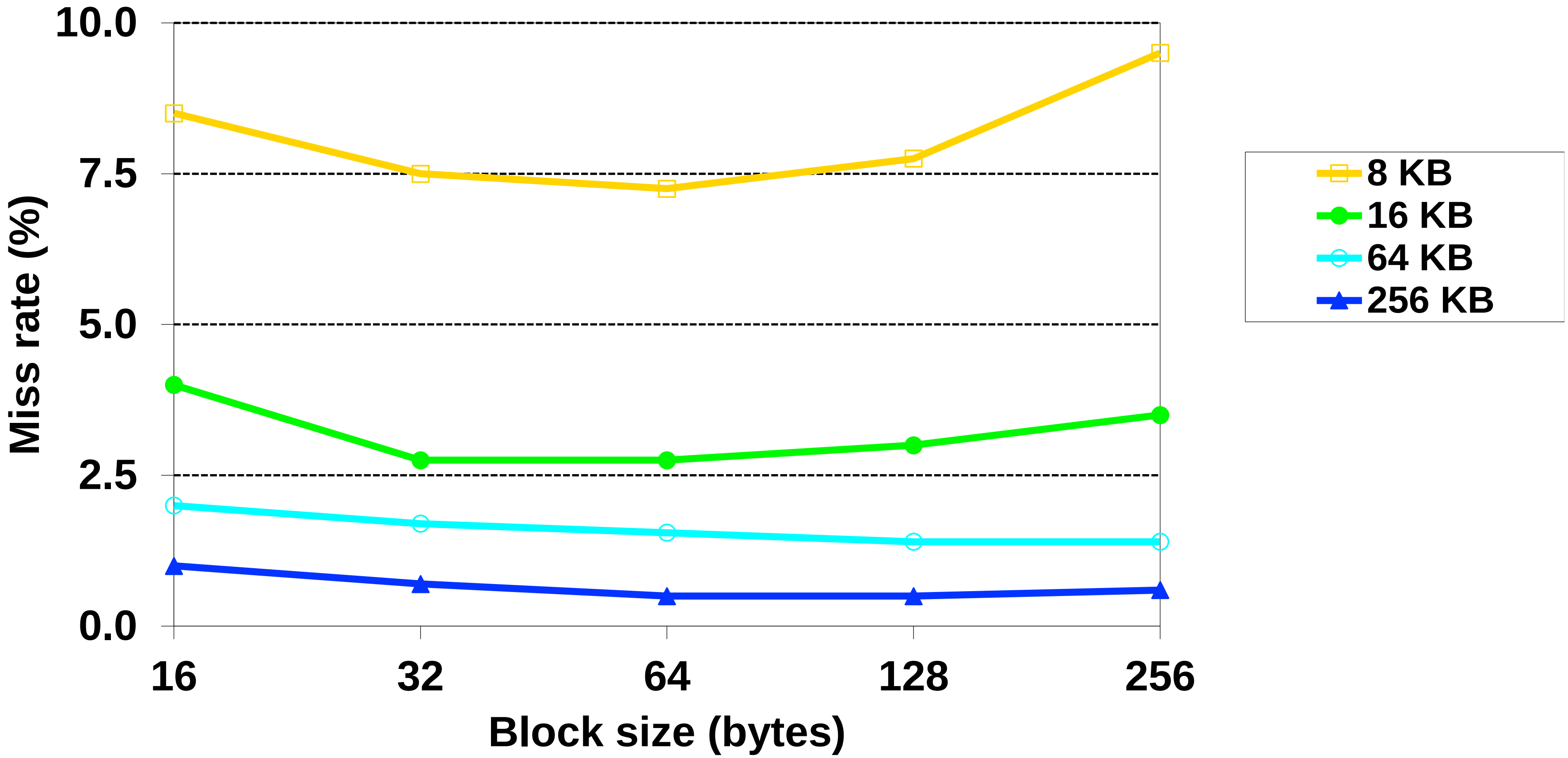
01	Mem(5)	Mem(4)
00	Mem(3)	Mem(2)

15 miss

01	Mem(5)	Mem(4)
11 ~~00~~	~~Mem(3)~~ 15	~~Mem(2)~~ 14

- 8 requests, 4 misses, with same size of cache!

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

  AMAT = Time for a hit + Miss rate x 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 x 50 = 2 clock cycles

  Or 2 x 200 = 400 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

Summary

- Principle of Locality for Computer Memory
- Hierarchy of Memories (speed/size/cost per bit) to Exploit Locality
- Cache – copy of data, a 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