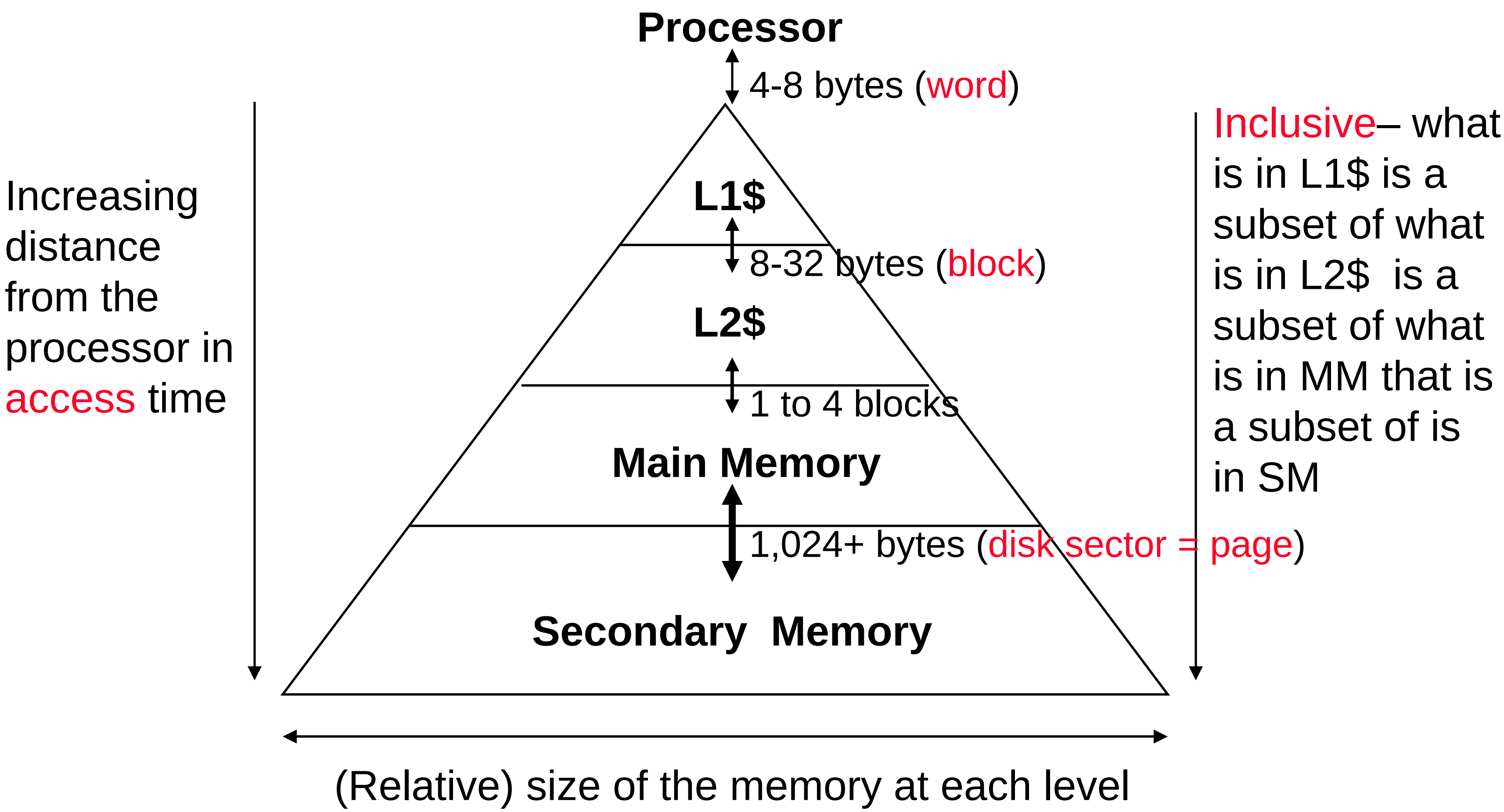
ECE7995
(5) Caching in Processor Cache

[Adapted from Mary Jane Irwin's slides (PSU)]

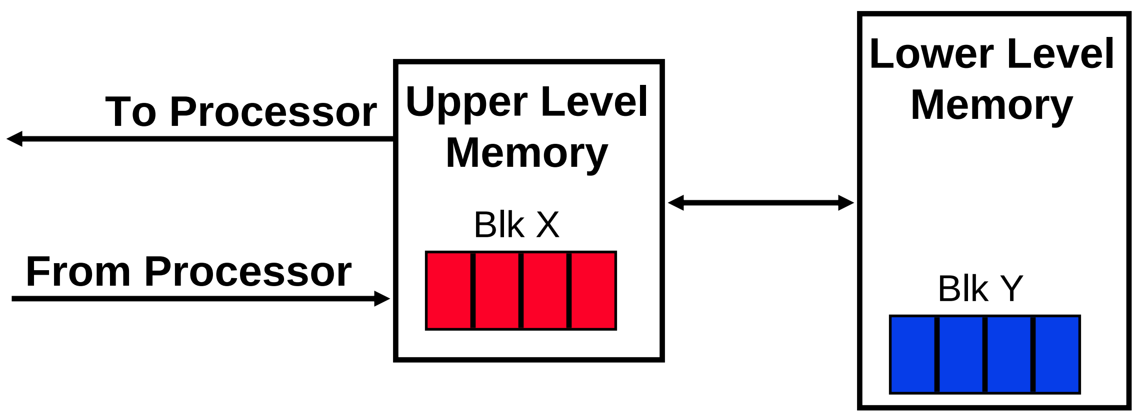
Review: The Memory Hierarchy

- Take advantage of the principle of locality to present the user with as much memory as is available in the cheapest technology at the speed offered by the fastest technology



The Memory Hierarchy: Why Does it Work?

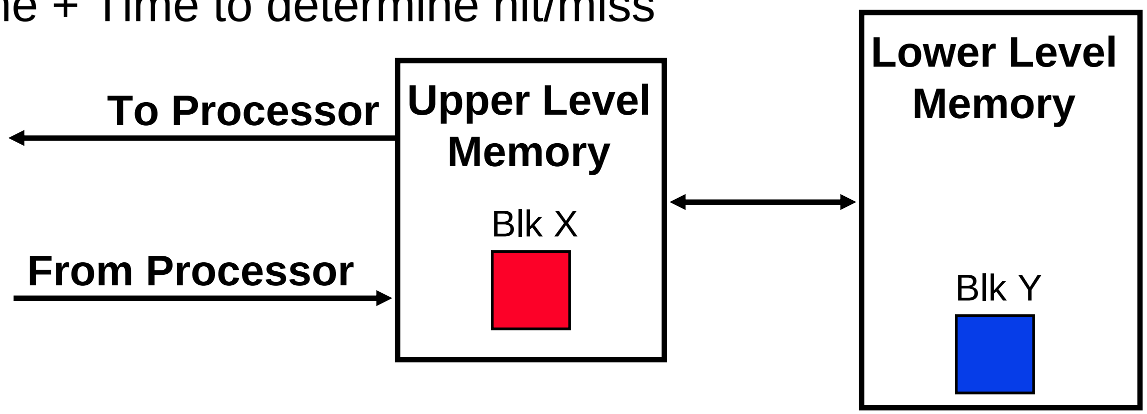
- Temporal Locality (Locality in Time):
  - ⇒ Keep most recently accessed data items closer to the processor
- Spatial Locality (Locality in Space):
  - ⇒ Move blocks consisting of contiguous words to the upper levels



The Memory Hierarchy: Terminology

- Hit: data is in some block in the upper level (Blk X)
  - Hit Rate: the fraction of memory accesses found in the upper level
  - Hit Time: Time to access the upper level which consists of

    RAM access time + Time to determine hit/miss



- Miss: data is not in the upper level so needs to be retrieve from a block in the lower level (Blk Y)
  - Miss Rate = 1 - (Hit Rate)
  - Miss Penalty: Time to replace a block in the upper level
    + Time to deliver the block the processor
  - Hit Time << Miss Penalty

How is the Hierarchy Managed?

- registers ↔ memory
  - by compiler (programmer?)
- cache ↔ main memory
  - by the cache controller hardware
- main memory ↔ disks
  - by the operating system (virtual memory)
  - virtual to physical address mapping assisted by the hardware (TLB)
  - by the programmer (files)

Cache

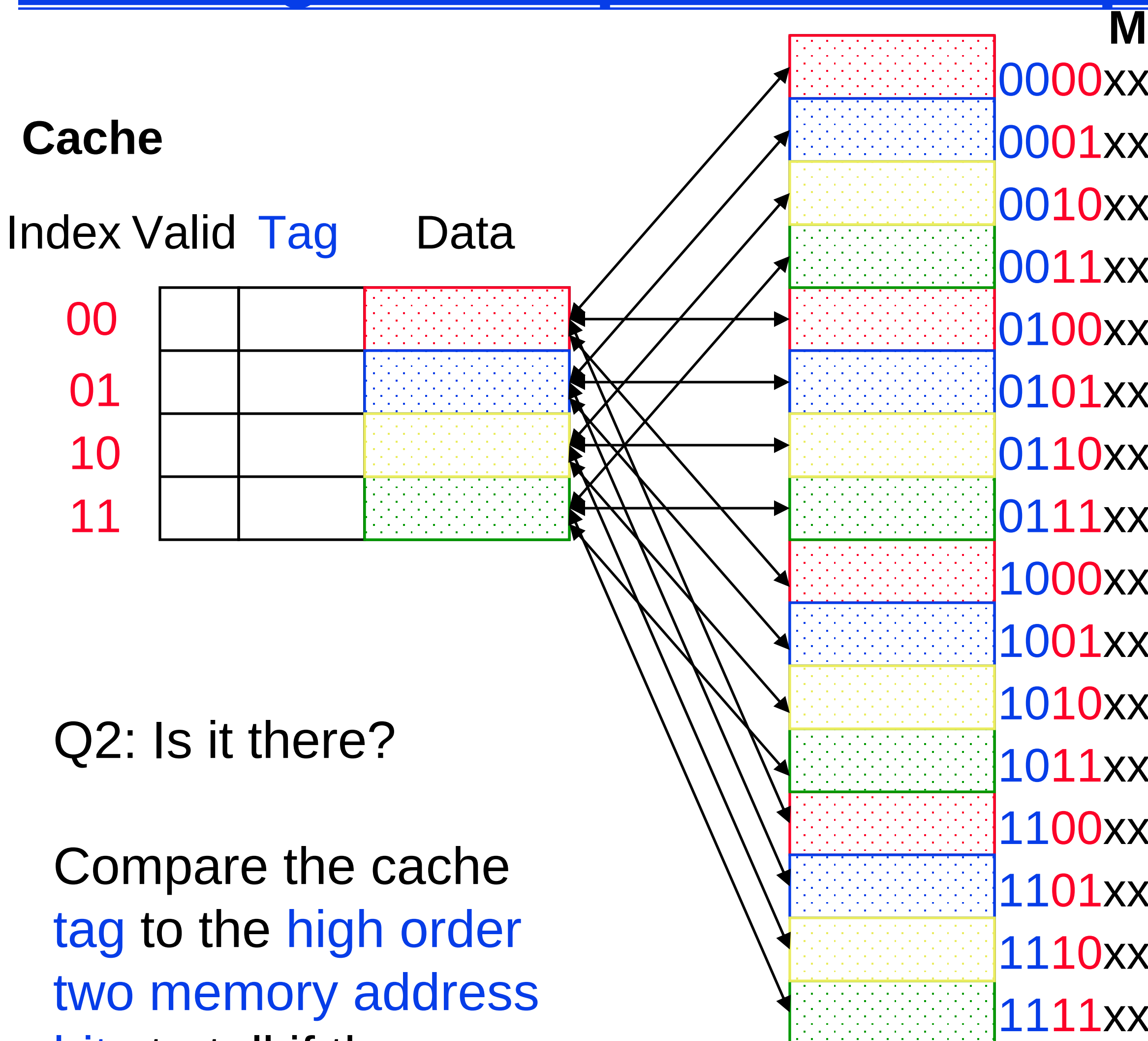
- Two questions to answer (in hardware):
  - Q1: How do we know if a data item is in the cache?
  - Q2: If it is, how do we find it?

- Direct mapped
  - For each item of data at the lower level, there is exactly one location in the cache where it might be - so lots of items at the lower level must share locations in the upper level
  - Address mapping:

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

  - First consider block sizes of one word

Caching: A Simple First Example



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

Q1: How do we find it?

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)

Q2: Is it there?

Compare the cache tag to the high order two 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

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~~ 01	~~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)

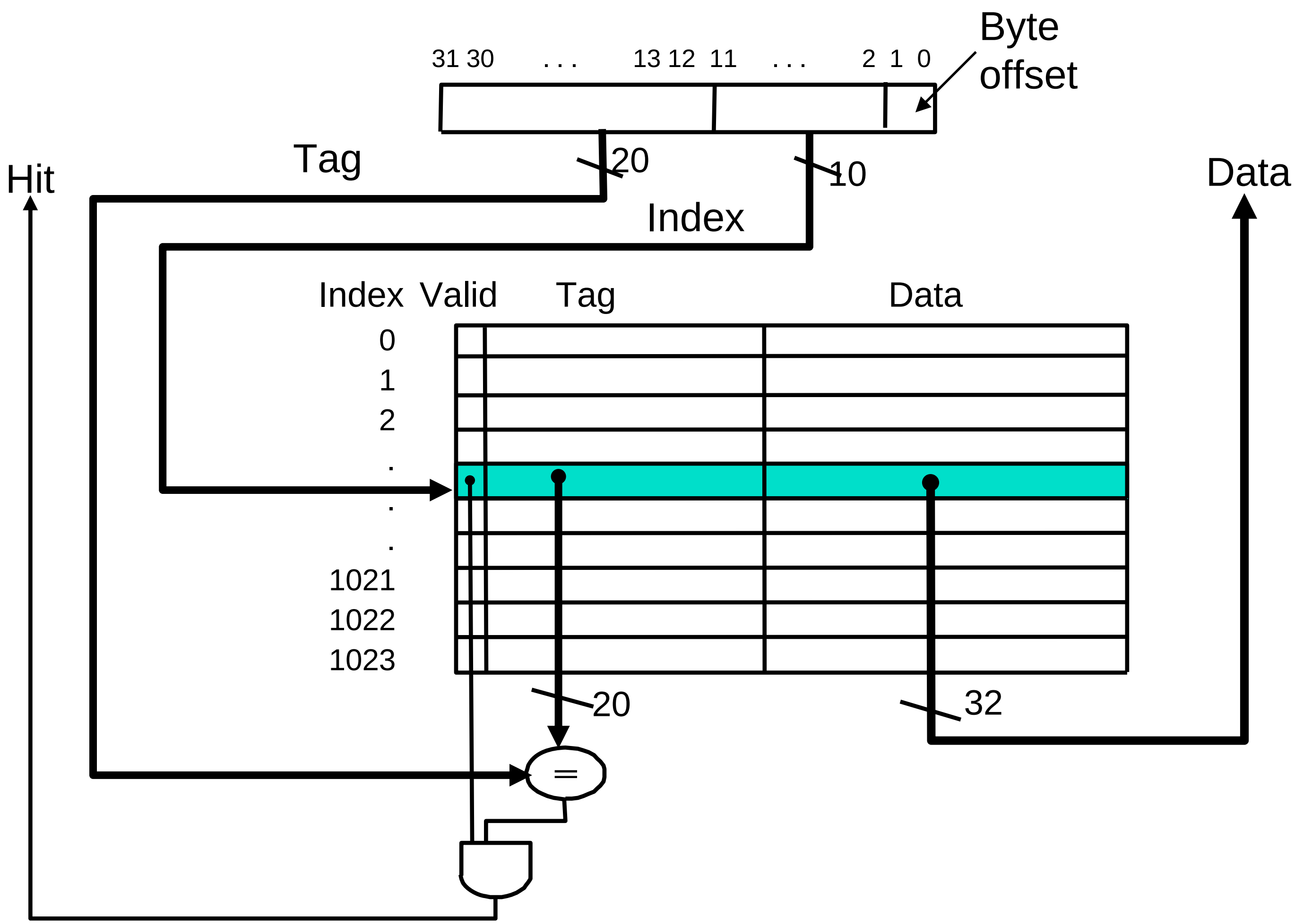
15 miss

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

- 8 requests, 6 misses

MIPS Direct Mapped Cache Example

- One word/block, cache size = 1K words



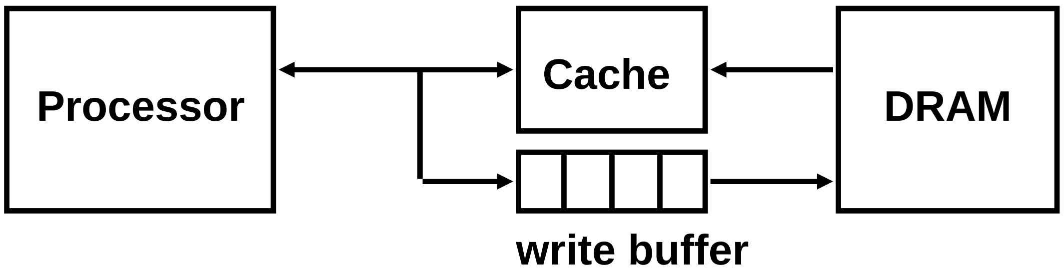
What kind of locality are we taking advantage of?

Handling Cache Hits

- Read hits (I$ and D$)
  - this is what we want!

- Write hits (D$ only)
  - allow cache and memory to be inconsistent
    - write the data only into the cache block (write-back the cache contents to the next level in the memory hierarchy when that cache block is "evicted")
    - need a dirty bit for each data cache block to tell if it needs to be written back to memory when it is evicted
  - require the cache and memory to be consistent
    - always write the data into both the cache block and the next level in the memory hierarchy (write-through) so don't need a dirty bit
    - writes run at the speed of the next level in the memory hierarchy – so slow! – or can use a write buffer, so only have to stall if the write buffer is full

Write Buffer for Write-Through Caching



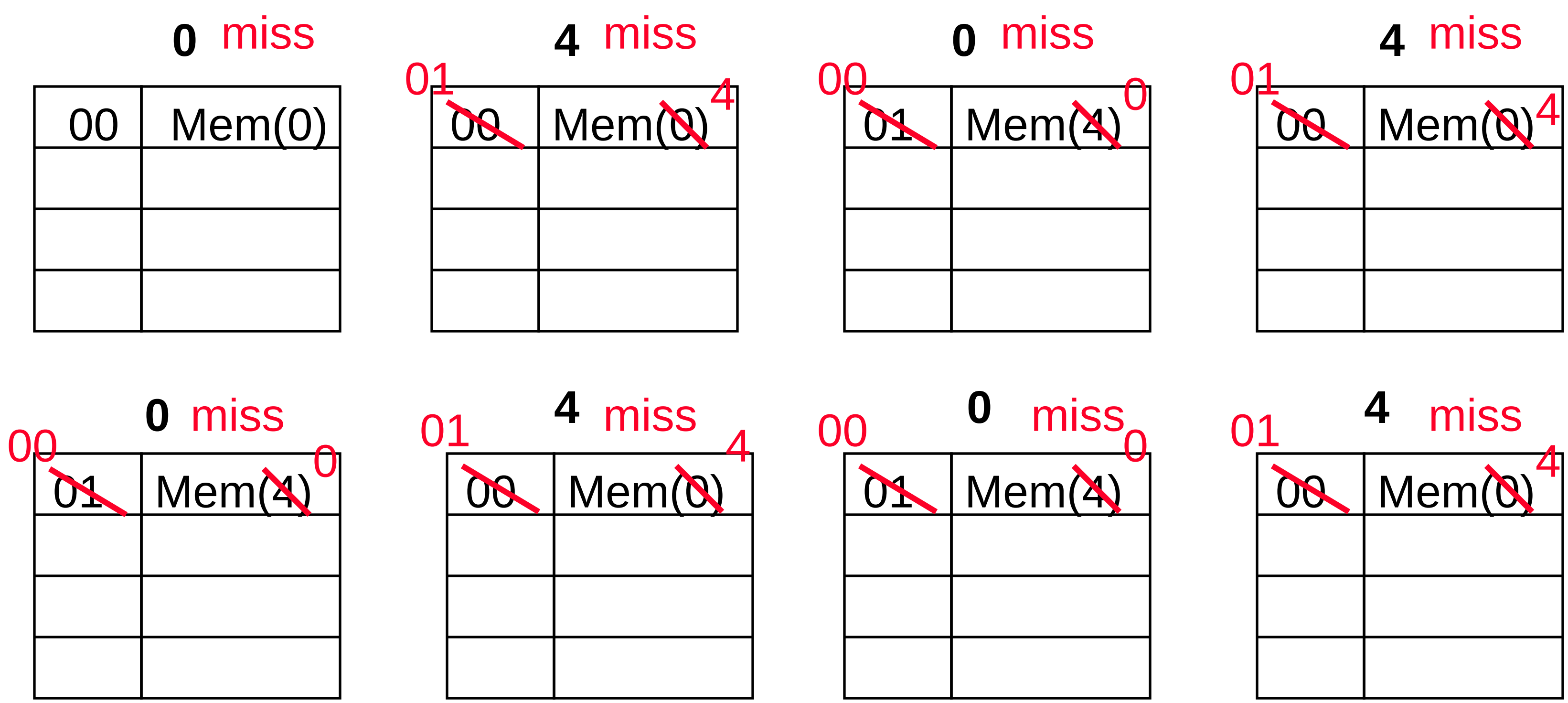
- Write buffer between the cache and main memory
  - Processor: writes data into the cache and the write buffer
  - Memory controller: writes contents of the write buffer to memory
- The write buffer is just a FIFO
  - Typical number of entries: 4
  - Works fine if store frequency (w.r.t. time) << 1 / DRAM write cycle
- Memory system designer's nightmare
  - When the store frequency (w.r.t. time) → 1 / DRAM write cycle leading to write buffer saturation
    - One solution is to use a write-back cache; another is to use an L2 cache

Another Reference String Mapping

- Consider the main memory word reference string

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

0 4 0 4 0 4 0 4



  - 8 requests, 8 misses

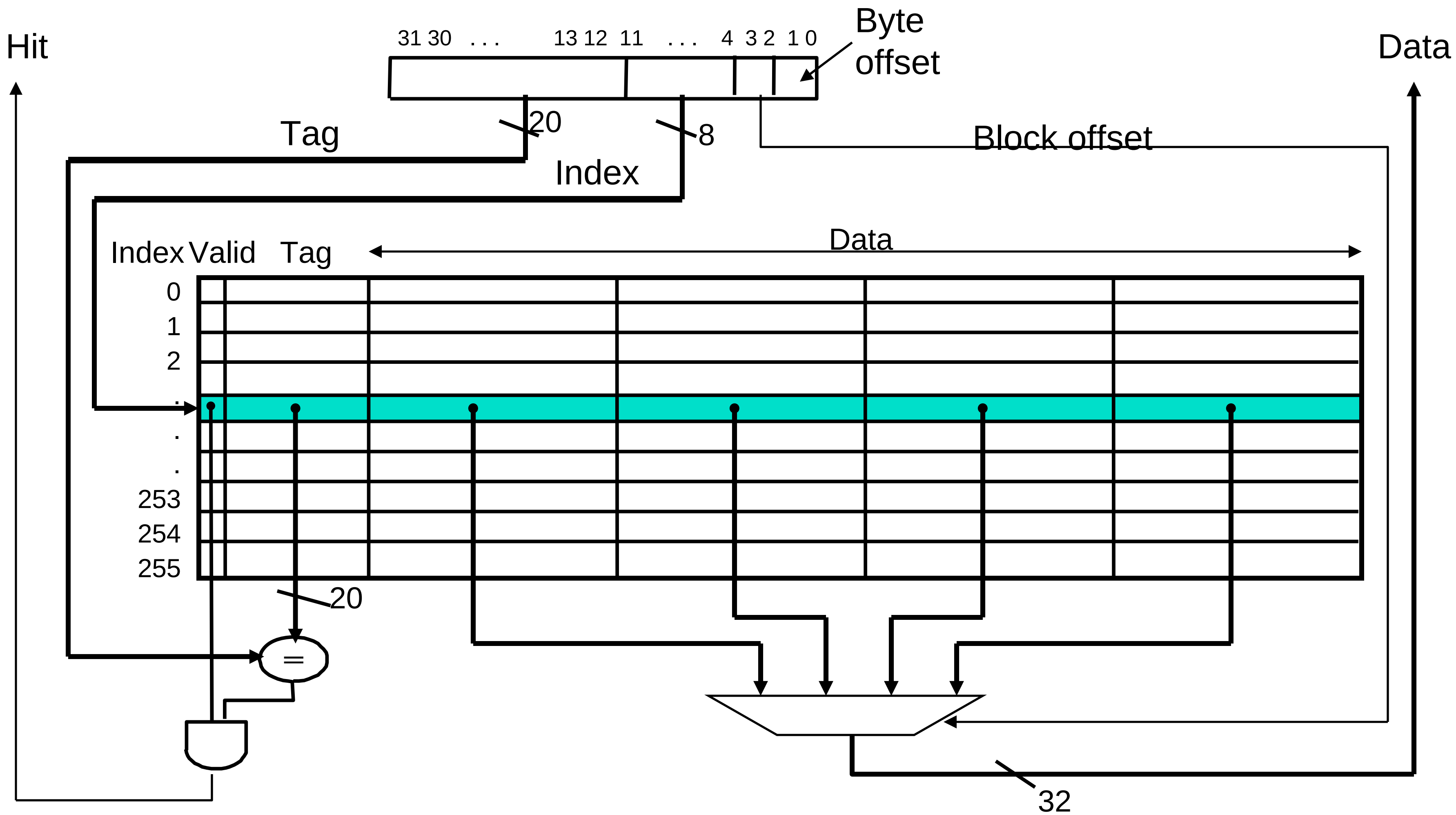
- Ping pong effect due to conflict misses - two memory locations that map into the same cache block

Sources of Cache Misses

- Compulsory (cold start or process migration, first reference):
  - First access to a block, “cold” fact of life, not a whole lot you can do about it
  - If you are going to run “millions” of instruction, compulsory misses are insignificant
- Conflict (collision):
  - Multiple memory locations mapped to the same cache location
  - Solution 1: increase cache size
  - Solution 2: increase associativity (next lecture)
- Capacity:
  - Cache cannot contain all blocks accessed by the program
  - Solution: increase cache size

Multiword Block Direct Mapped Cache (increase block size)

- 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

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

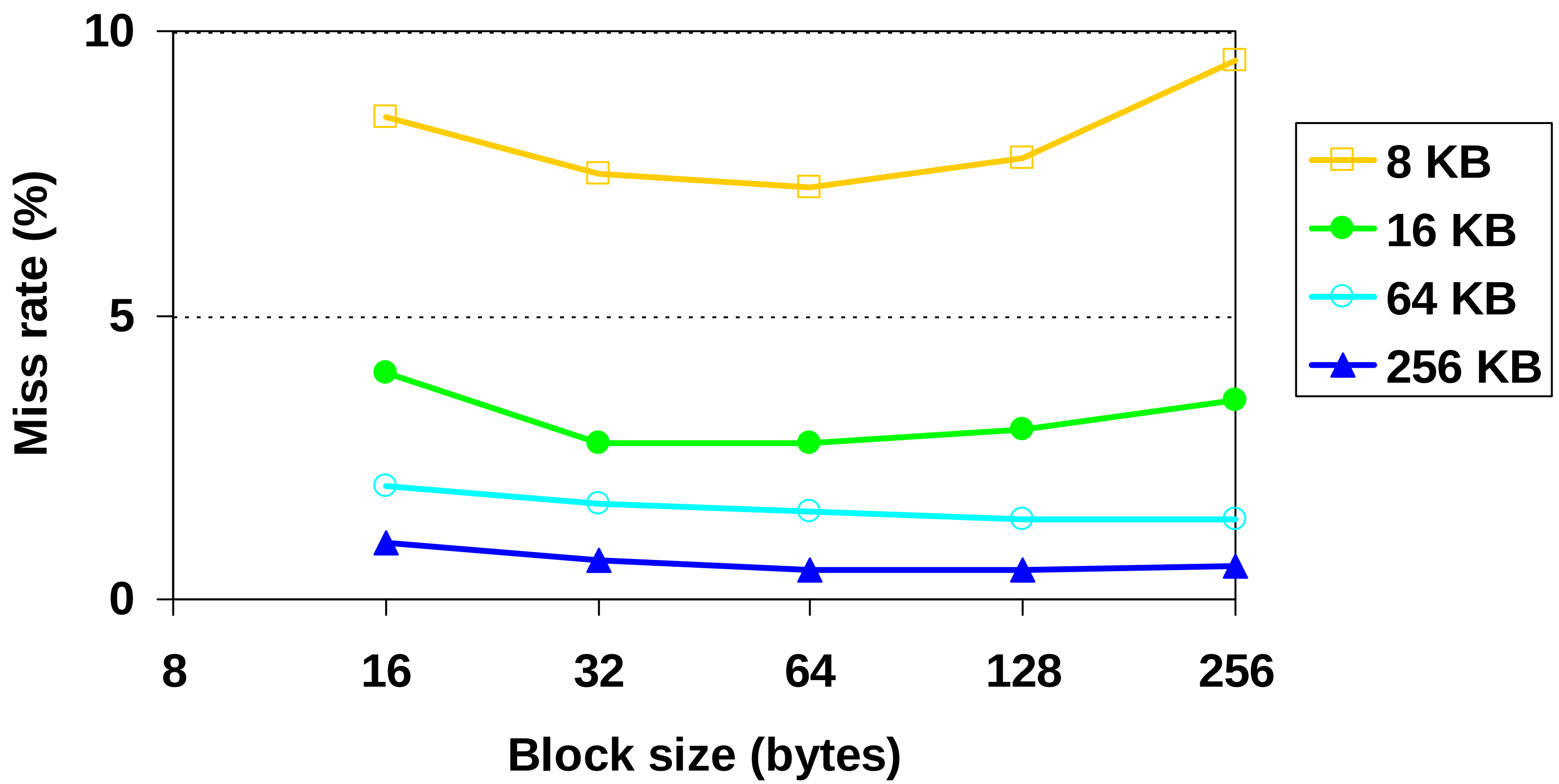
15 miss

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

- 8 requests, 4 misses

Exploitation of spatial locality has the benefit of prefetching

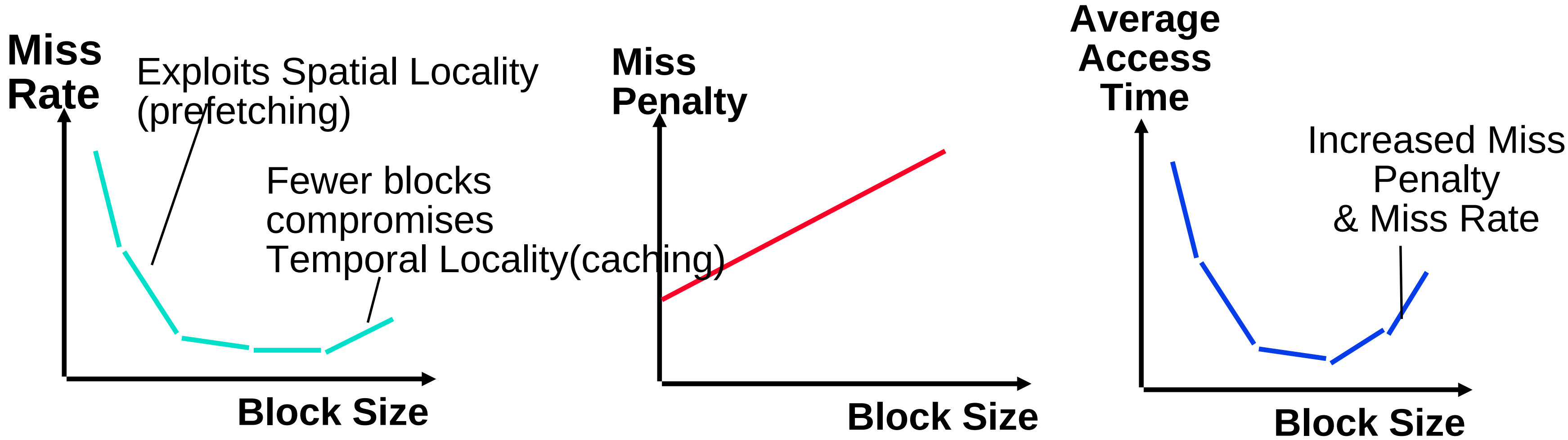
Miss Rate vs Block Size vs Cache Size



- 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)

Block Size Tradeoff

- Larger block sizes take advantage of spatial locality but
  - If the block size is too big relative to the cache size, the miss rate will go up
  - Larger block size means larger miss penalty
    - Latency to first word in block + transfer time for remaining words



- In general, Average Memory Access Time

  = Hit Time + Miss Penalty x Miss Rate