CS 61C: Great Ideas in Computer Architecture (Machine Structures) Caches Part 1

Instructors:

Bernhard Boser & Randy H. Katz

http://inst.eecs.berkeley.edu/~cs61c/

New-School Machine Structures

Software *Hardware*

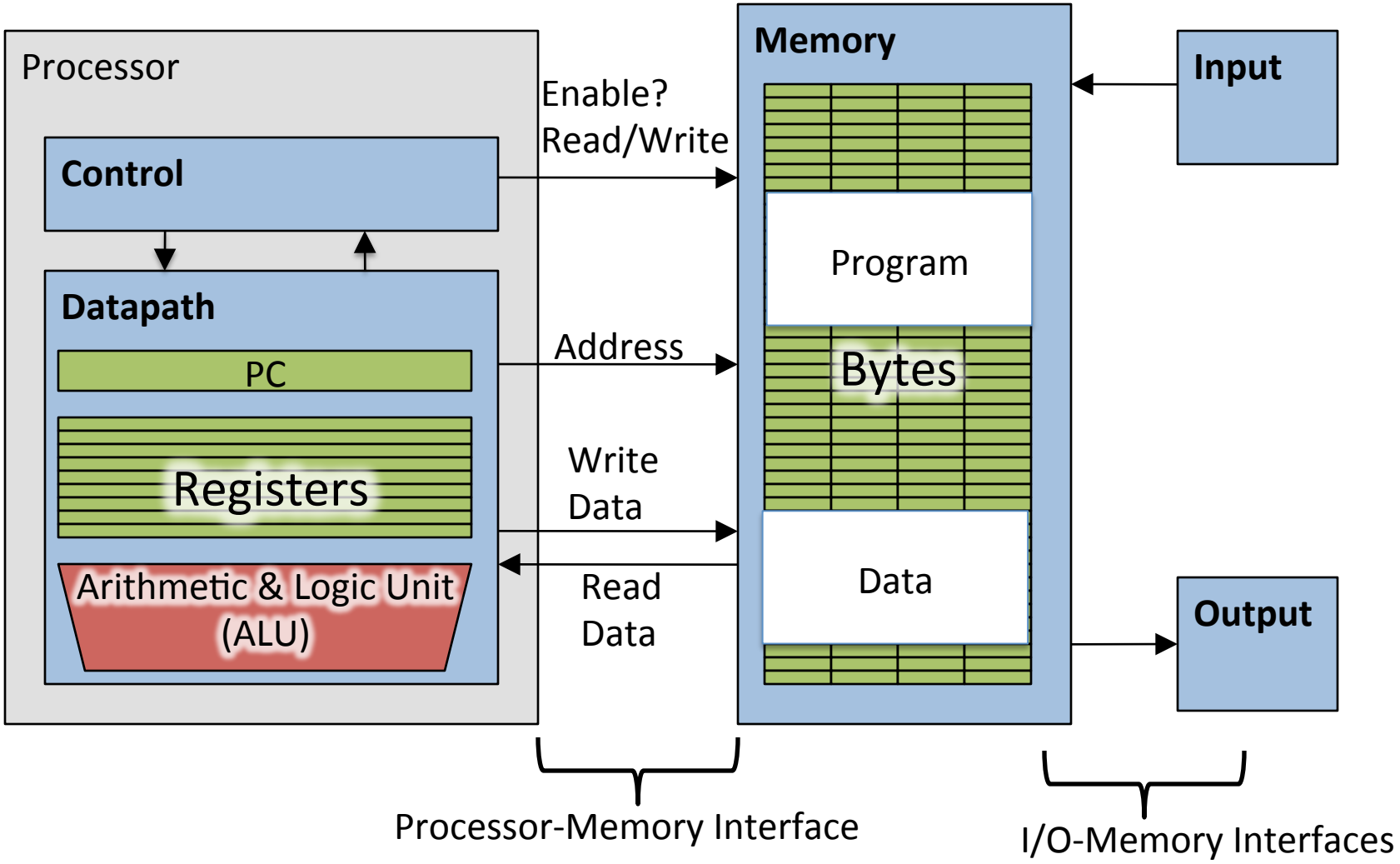
- Parallel Requests
  Assigned to computer
  e.g., Search "Katz"
- Parallel Threads
  Assigned to core
  e.g., Lookup, Ads
- Parallel Instructions
  >1 instruction @ one time
  e.g., 5 pipelined instructions
- Parallel Data
  >1 data item @ one time
  e.g., Add of 4 pairs of words
- Hardware descriptions
  All gates @ one time
- Programming Languages

Harness Parallelism & Achieve High Performance

Warehouse Scale Computer

Smart Phone

Components of a Computer



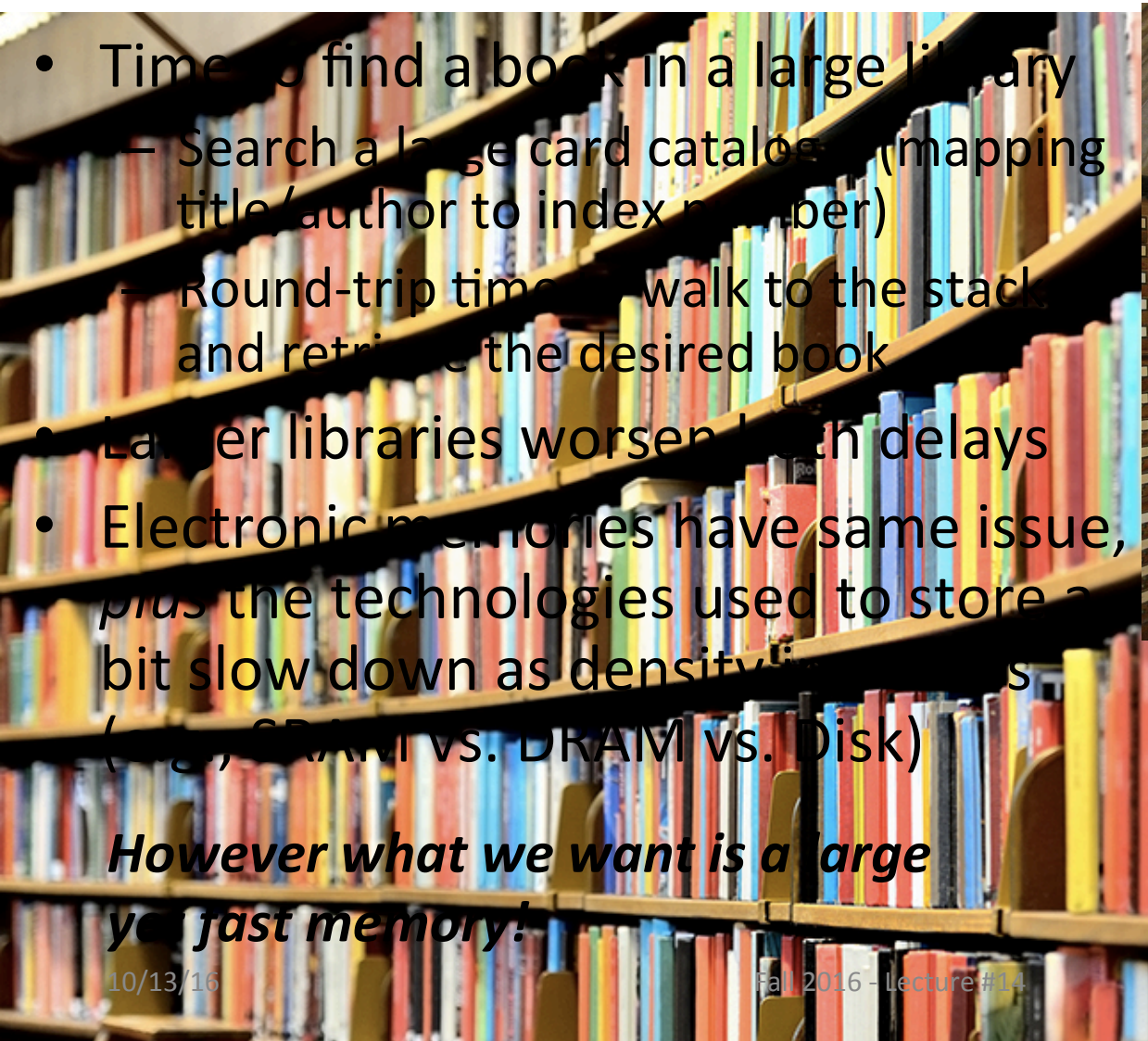
Outline

- Memory Hierarchy and Latency
- Caches Principles
- Basic Cache Organization
- Different Kinds of Caches
- Write Back vs. Write Through
- And in Conclusion …

Outline

- Memory Hierarchy and Latency
- Caches Principles
- Basic Cache Organization
- Different Kinds of Caches
- Write Back vs. Write Through
- And in Conclusion …

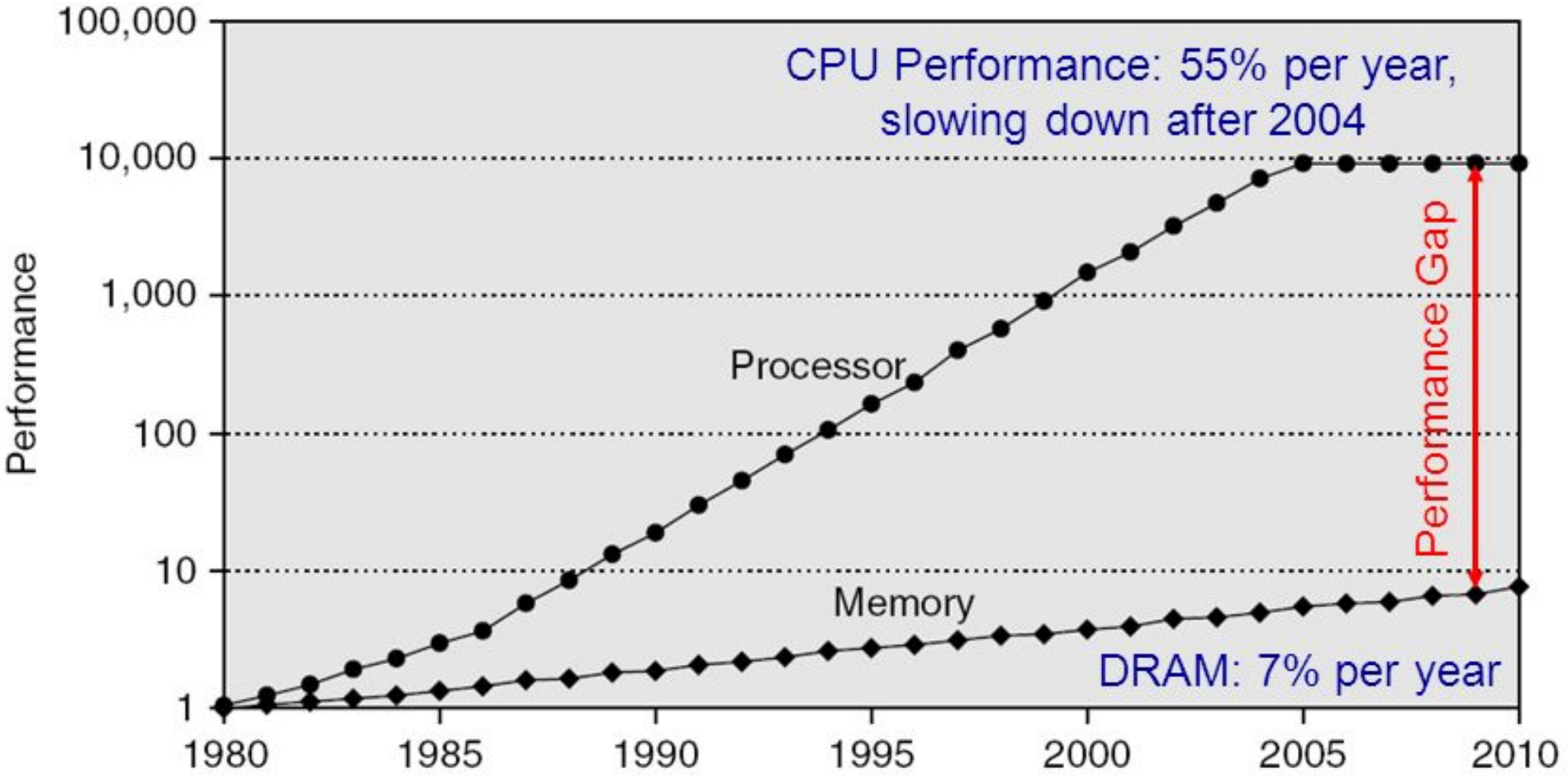
Why are Large Memories Slow? Library Analogy



- Time to find a book in a large library
  - Search a large card catalog (mapping title/author to index number)
  - Round-trip time to walk to the stacks and retrieve the desired book
- Larger libraries worsen both delays
- Electronic memories have same issue, *plus* the technologies used to store a bit slow down as density increases (e.g., SRAM vs. DRAM vs. Disk)

However what we want is a large yet fast memory!

Processor-DRAM Gap (Latency)



1980 microprocessor executes **~one instruction** in same time as DRAM access

2016 microprocessor executes **~1000 instructions** in same time as DRAM access

Slow DRAM access has disastrous impact on CPU performance!

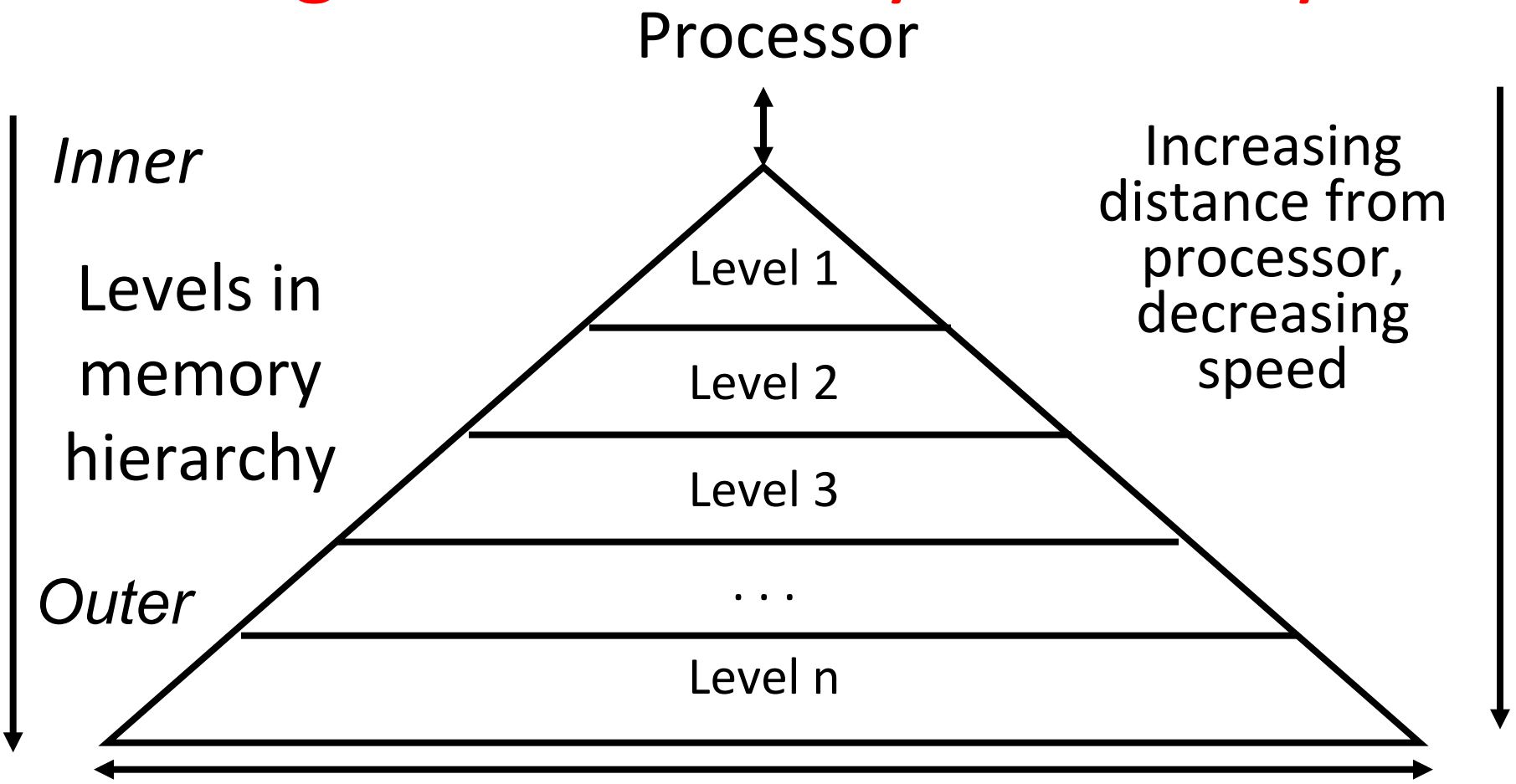
What To Do: Library Analogy

- Write a report using library books
  - E.g., works of J.D. Salinger
- Go to library, look up relevant books, fetch from stacks, and place on desk in library
- If need more, check them out and keep on desk
  - But don't return earlier books since might need them
- You hope this collection of ~10 books on desk enough to write report, despite 10 being only 0.00001% of books in UC Berkeley libraries

Outline

- Memory Hierarchy and Latency
- Caches Principles
- Basic Cache Organization
- Different Kinds of Caches
- Write Back vs. Write Through
- And in Conclusion …

Big Idea: Memory Hierarchy

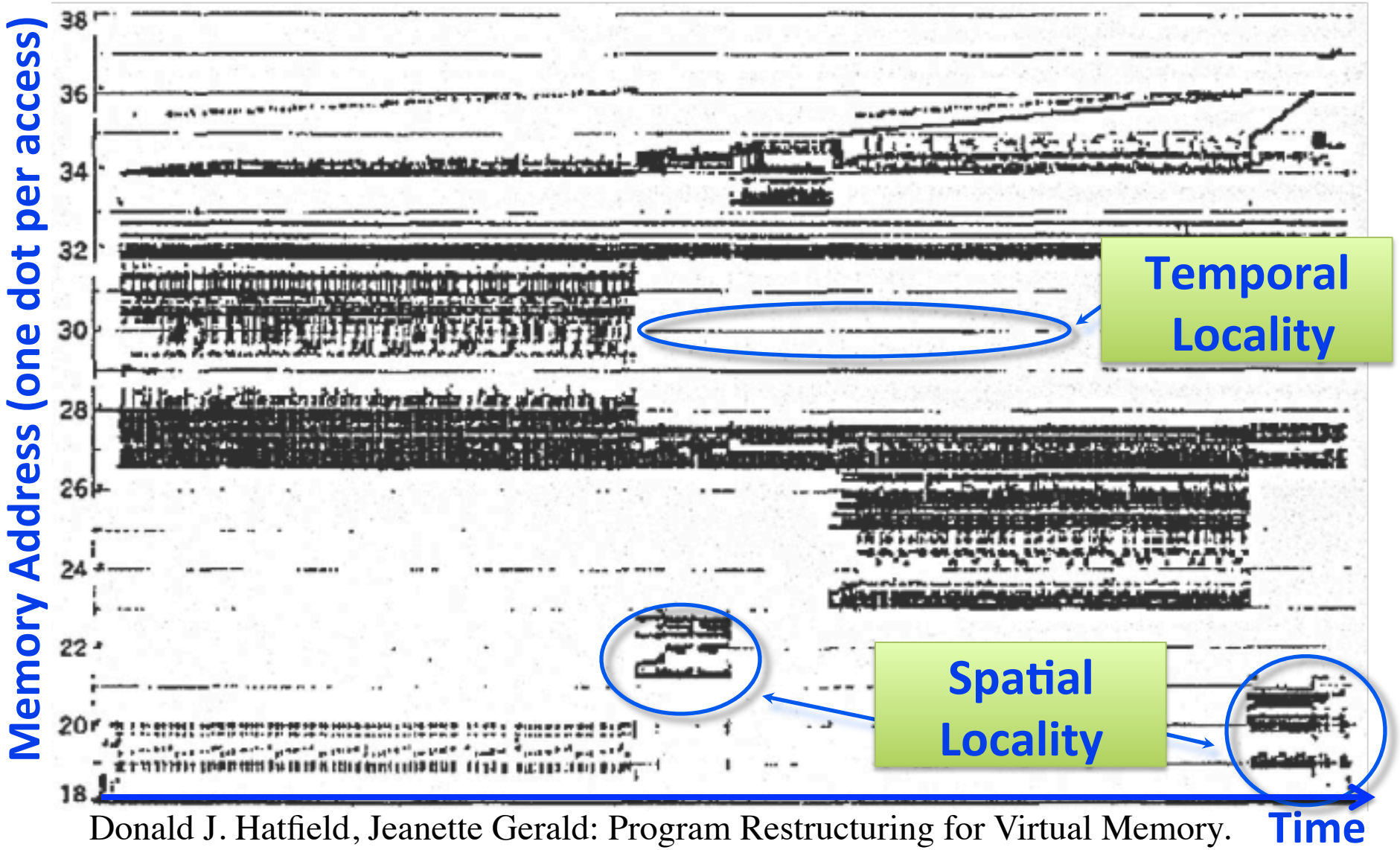


As we move to outer levels the latency goes up and price per bit goes down. Why?

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 J.D. Salinger since library stores related books together
  - If a memory location is referenced, the locations with nearby addresses will tend to be referenced soon

Memory Reference Patterns

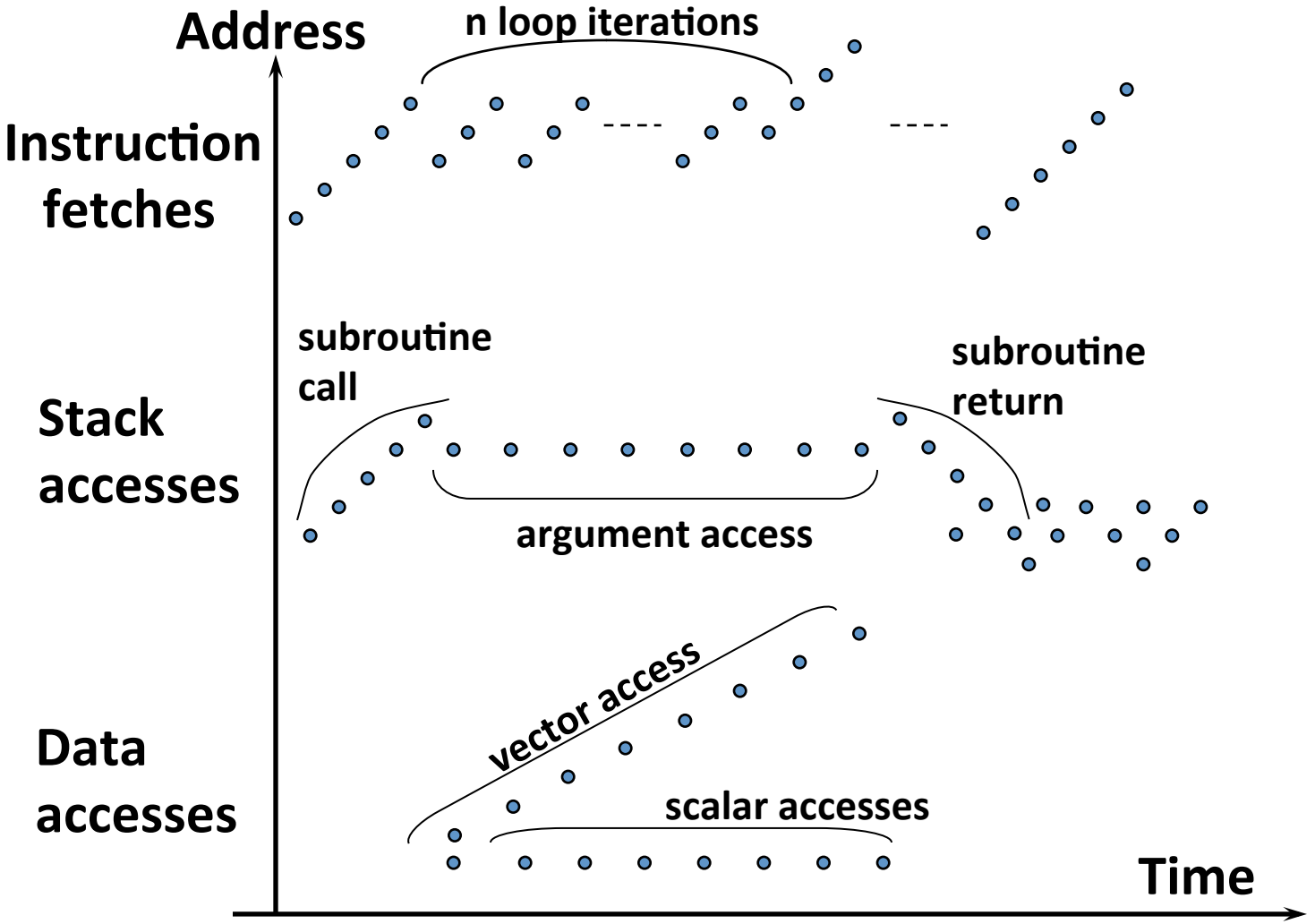


Donald J. Hatfield, Jeanette Gerald: Program Restructuring for Virtual Memory. IBM Systems Journal 10(3): 168-192 (1971)

Principle of Locality

- *Principle of Locality*: Programs access small portion of address space at any instant of time (spatial locality) and repeatedly access that portion (temporal locality)
- What program structures lead to temporal and spatial locality in instruction accesses?
- In data accesses?

Memory Reference Patterns



cache

/kaSH/

noun

noun: **cache**; plural noun: **caches**

1. a collection of items of the same type stored in a hidden or inaccessible place.
   "an arms cache"
   synonyms: hoard, store, stockpile, stock, supply, reserve; More
   - a hidden or inaccessible storage place for valuables, provisions, or ammunition.
     synonyms: hoard, store, stockpile, stock, supply, reserve; More
   - COMPUTING
     an auxiliary memory from which high-speed retrieval is possible.
     noun: **cache memory**; plural noun: **cache memories**

verb

verb: **cache**; 3rd person present: **caches**; past tense: **cached**; past participle: **cached**; gerund or present participle: **cacheing**; gerund or present participle: **caching**

1. store away in hiding or for future use.
   - COMPUTING
     store (data) in a cache memory.
   - COMPUTING
     provide (hardware) with a cache memory.

Origin

FRENCH

cacher → cache

to hide — *late 18th century*

late 18th century: from French, from *cacher* 'to hide.'

Cache Philosophy

- Programmer-invisible hardware mechanism gives illusion of speed of fastest memory with size of largest memory
  - Works even if you have no idea what a cache is
  - Performance-oriented programmers sometimes "reverse engineer" cache organization to design data structures and access patterns optimized for a specific cache design
  - You are going to do that in Project #4!

Outline

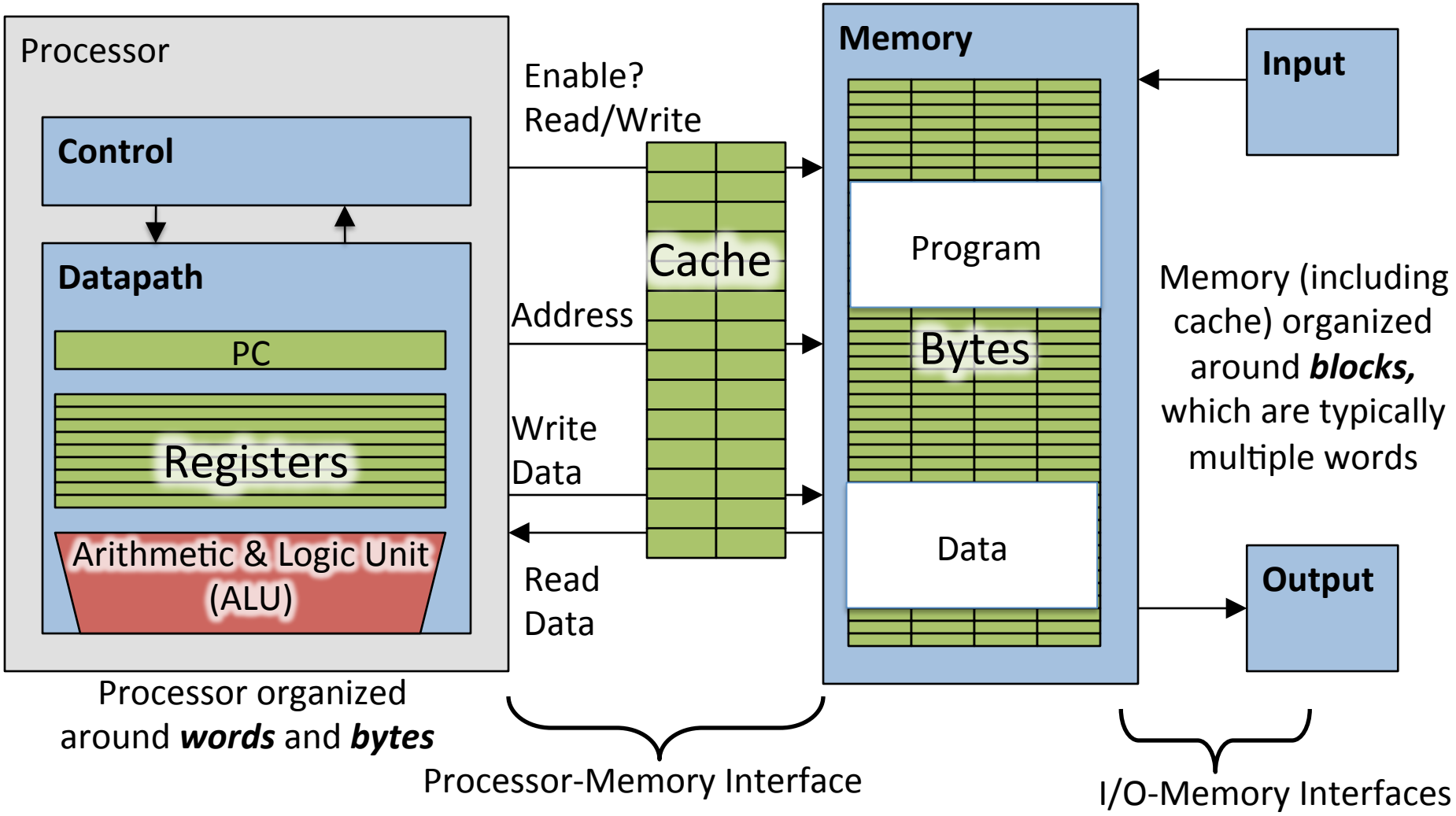
- Memory Hierarchy and Latency
- Caches Principles
- Basic Cache Organization
- Different Kinds of Caches
- Write Back vs. Write Through
- And in Conclusion …

Memory Access without Cache

- Load word instruction: `lw $t0,0($t1)`
- $t1 contains $1022_{ten,}$ Memory[1022] = 99

1. Processor issues address 1022_{ten} to Memory
2. Memory reads word at address 1022_{ten} (99)
3. Memory sends 99 to Processor
4. Processor loads 99 into register $t0

Adding Cache to Computer



Memory Access with Cache

- Load word instruction: `lw $t0,0($t1)`
- $t1 contains $1022_{ten,}$ Memory[1022] = 99
- With cache: Processor issues address 1022_{ten} to Cache
  1. Cache checks to see if has copy of data at address 1022_{ten}
     2a. If finds a match (**Hit**): cache reads 99, sends to processor
     2b. No match (**Miss**): cache sends address 1022 to Memory
        I. Memory reads 99 at address 1022_{ten}
        II. Memory sends 99 to Cache
        III. Cache replaces word with new 99
        IV. Cache sends 99 to processor
  2. Processor loads 99 into register $t0

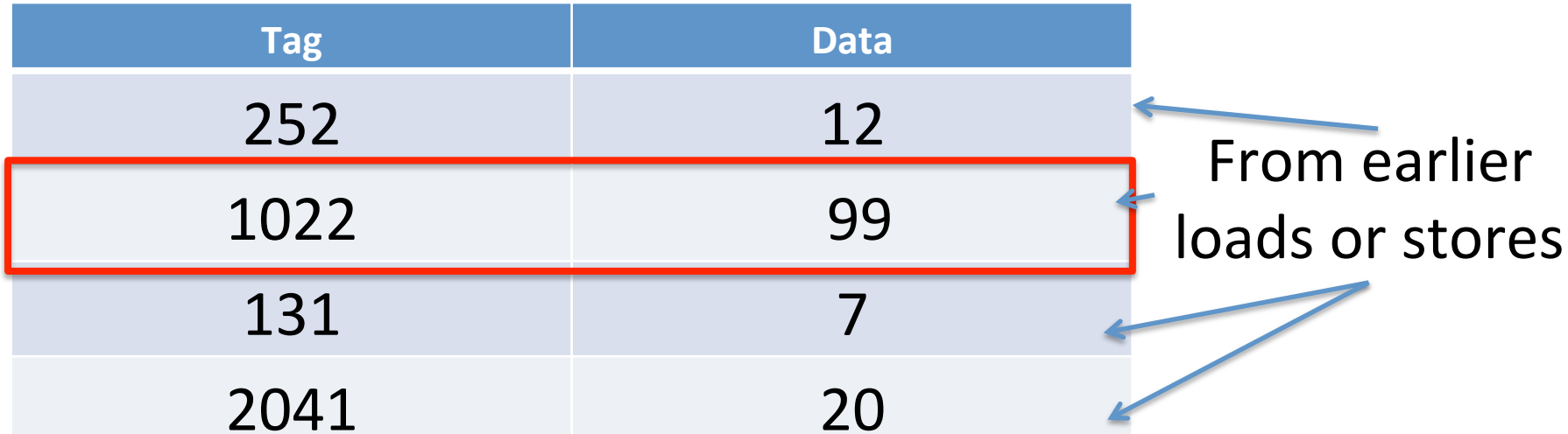
Administrivia

- Project 3-1 Released tonight!
- Midterm #2 2.5 weeks away! November 1!
  - In class! 3:40-5 PM
  - Focus on Pipelines and Caches
  - ONE Double sided Crib sheet
  - Review Session, Sunday, 10/30, 1-3 PM, 10 Evans

Cache "Tags"

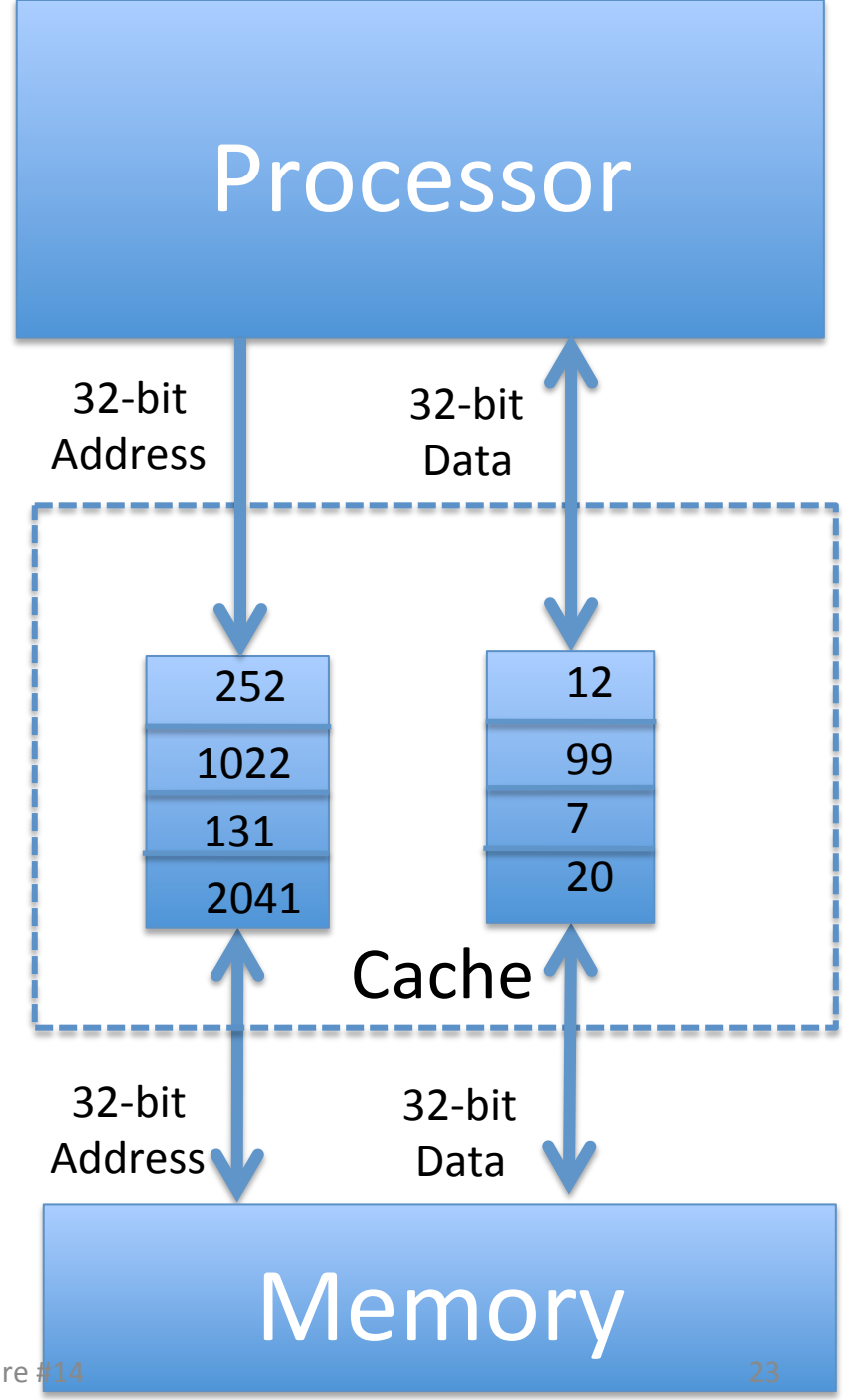
- Need way to tell if have copy of location in memory so that can decide on hit or miss
- On cache miss, put memory address of block in "tag address" of cache block
  - 1022 placed in tag next to data from memory (99)

Tag	Data
252	12
1022	99
131	7
2041	20



Anatomy of a 16 Byte Cache, with 4 Byte Blocks

- Operations:
  1. Cache Hit
  2. Cache Miss
  3. Refill cache from memory
- Cache needs Address Tags to decide if Processor Address is a Cache Hit or Cache Miss
  - Compares all four tags



Cache Replacement

- Suppose processor now requests location 511, which contains 11?
- Doesn't match any cache block, so must "evict" a resident block to make room
  - Which block to evict?
- Replace "victim" with new memory block at address 511

Tag	Data
252	12
1022	99
131	7
2041	20

Cache Replacement

- Suppose processor now requests location 511, which contains 11?
- Doesn't match any cache block, so must "evict" a resident block to make room
  - Which block to evict?
- Replace "victim" with new memory block at address 511

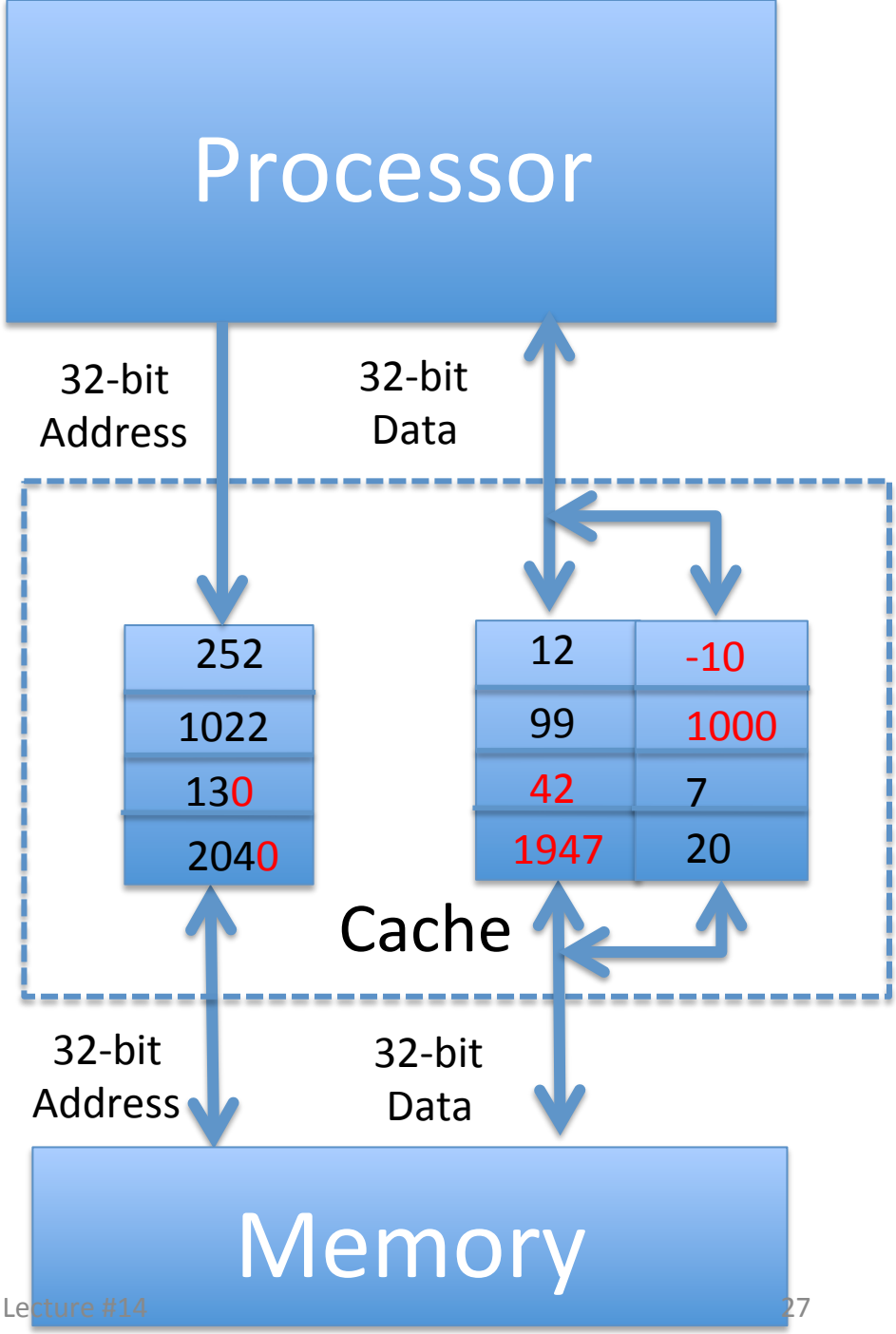
Tag	Data
252	12
1022	99
511	11
2041	20

Block Must be Aligned in Memory

- Word blocks are aligned, so binary address of all words in cache always ends in 00_{two}
- How to take advantage of this to save hardware and energy?
- Don't need to compare last 2 bits of 32-bit byte address (comparator can be narrower)
  - Don't need to store last 2 bits of 32-bit byte address in Cache Tag (Tag can be narrower)

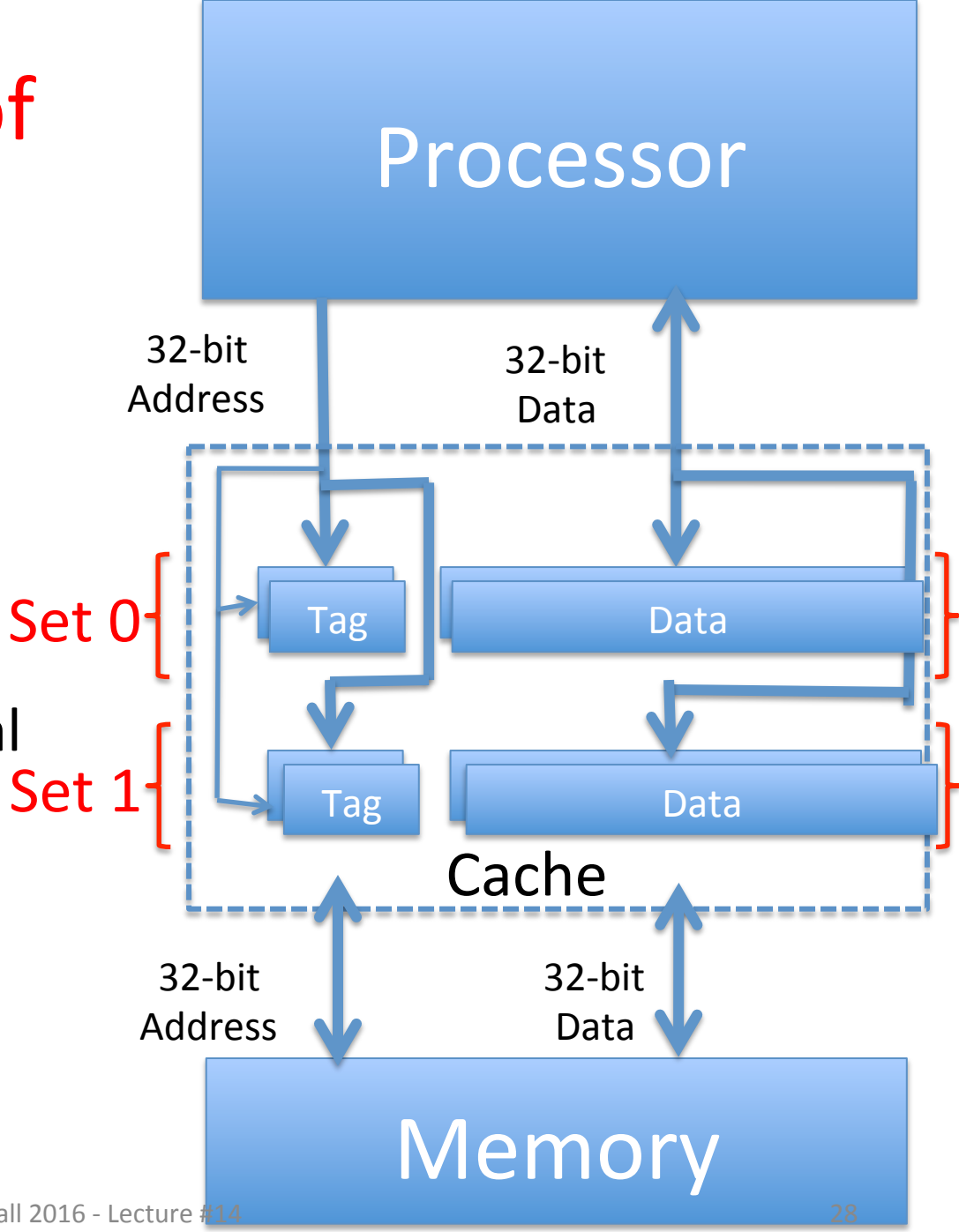
Anatomy of a 32B Cache, 8B Blocks

- Blocks must be aligned in pairs, otherwise could get same word twice in cache
  - Tags only have even-numbered words
  - Last 3 bits of address always 000_{two}
  - Tags, comparators can be narrower
- Can get hit for either word in block

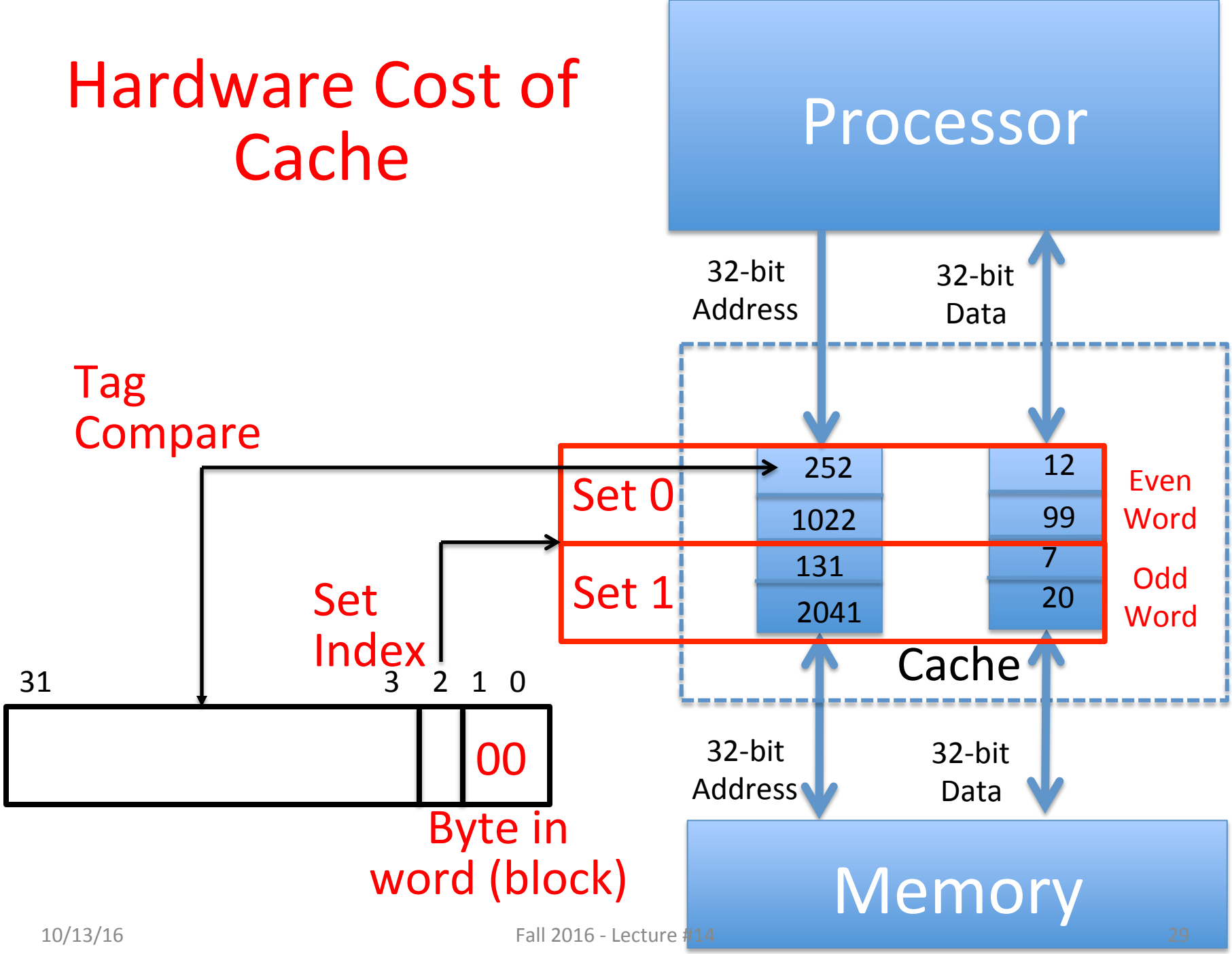


Hardware Cost of Cache

- Need to compare every tag to the Processor address
- Comparators are expensive
- Optimization: use two "sets" of data with a total of only 2 comparators
- Use one Address bit to select which set
- Compare only tags from selected set
- Generalize to more sets

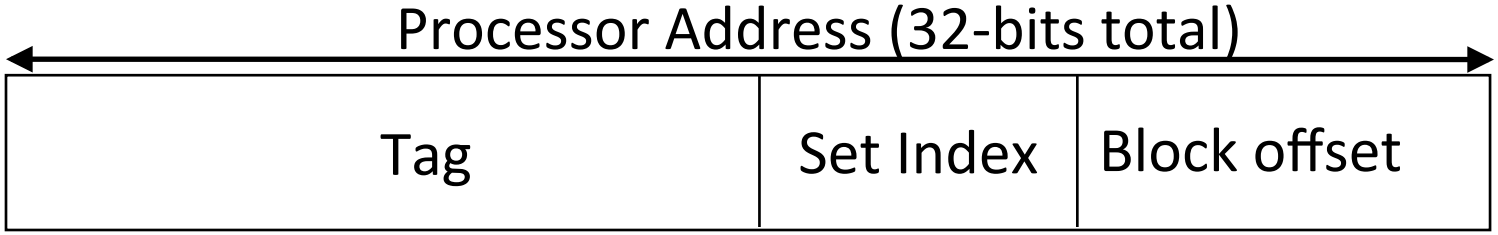


Hardware Cost of Cache



Processor Address Fields Used by Cache Controller

- Block Offset: Byte address within block
- Set Index: Selects which set
- Tag: Remaining portion of processor address

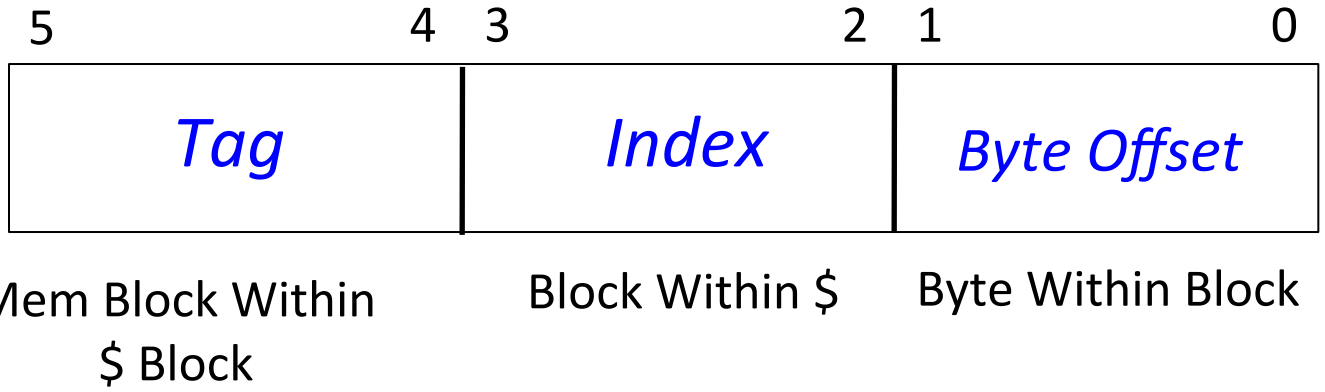


- Size of Index = $\log_2$(number of sets)
- Size of Tag = Address size – Size of Index – $\log_2$(number of bytes/block)

What Limits Number of Sets?

- For a given total number of blocks, we save comparators if have more than two sets
- Limit: As Many Sets as Cache Blocks => only one block per set – only needs one comparator!
- Called “Direct-Mapped” Design

Direct Mapped Cache Example: Mapping a 6-bit Memory Address



- In example, block size is 4 bytes/1 word
- Memory and cache blocks always the same size, unit of transfer between memory and cache
- # Memory blocks >> # Cache blocks
  - 16 Memory blocks = 16 words = 64 bytes => 6 bits to address all bytes
  - 4 Cache blocks, 4 bytes (1 word) per block
  - 4 Memory blocks map to each cache block
- Memory block to cache block, aka *index*: middle two bits
- Which memory block is in a given cache block, aka *tag*: top two bits

One More Detail: Valid Bit

- When start a new program, cache does not have valid information for this program
- Need an indicator whether this tag entry is valid for this program
- Add a "valid bit" to the cache tag entry

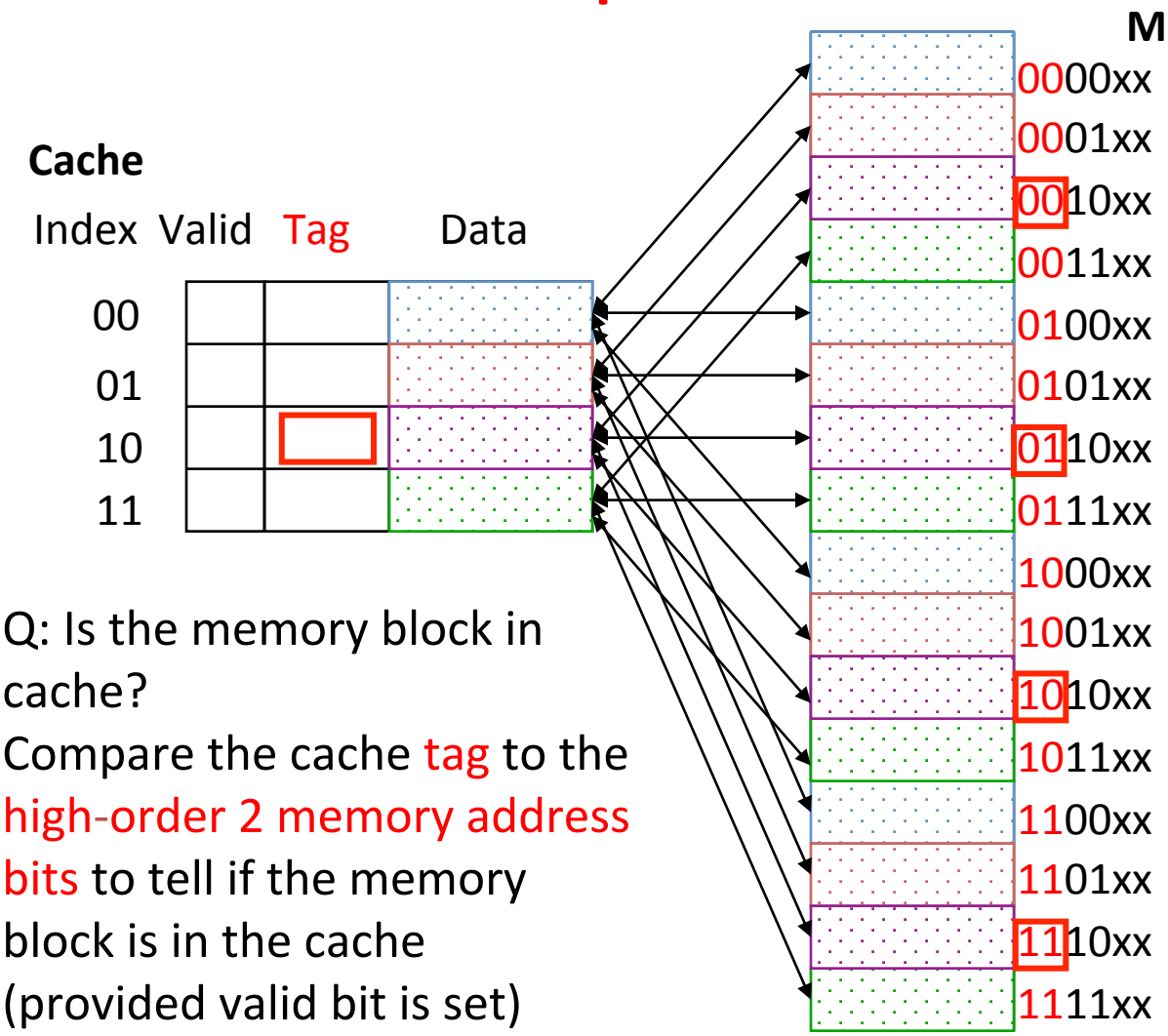
  0 => cache miss, even if by chance, address = tag

  1 => cache hit, if processor address = tag

Outline

- Memory Hierarchy and Latency
- Caches Principles
- Basic Cache Organization
- Different Kinds of Caches
- Write Back vs. Write Through
- And in Conclusion …

Cache Organization: Simple First Example



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

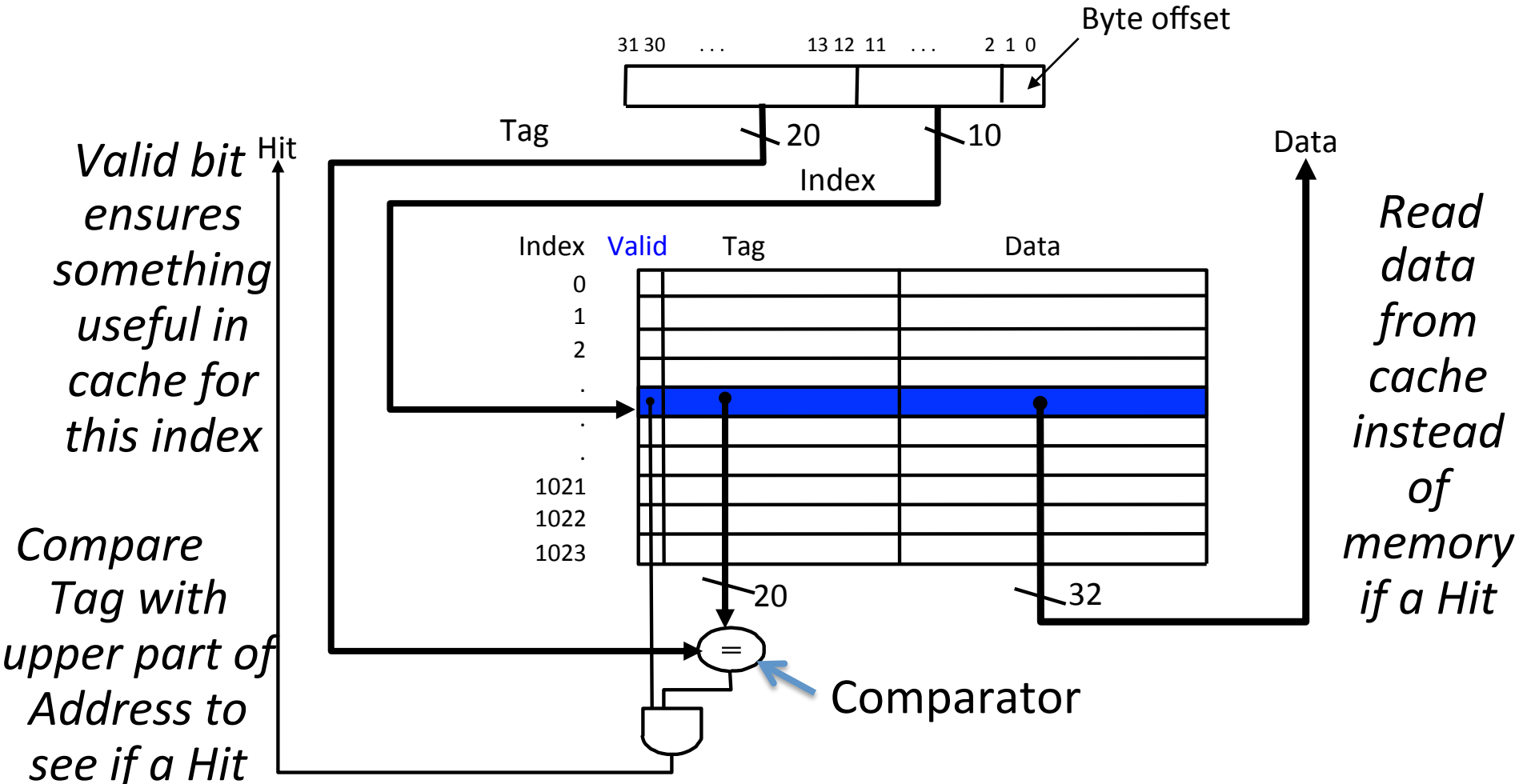
Q: Where in the cache is the mem block?

Q: Is the memory 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 (provided valid bit is set)

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)

Direct-Mapped Cache Example

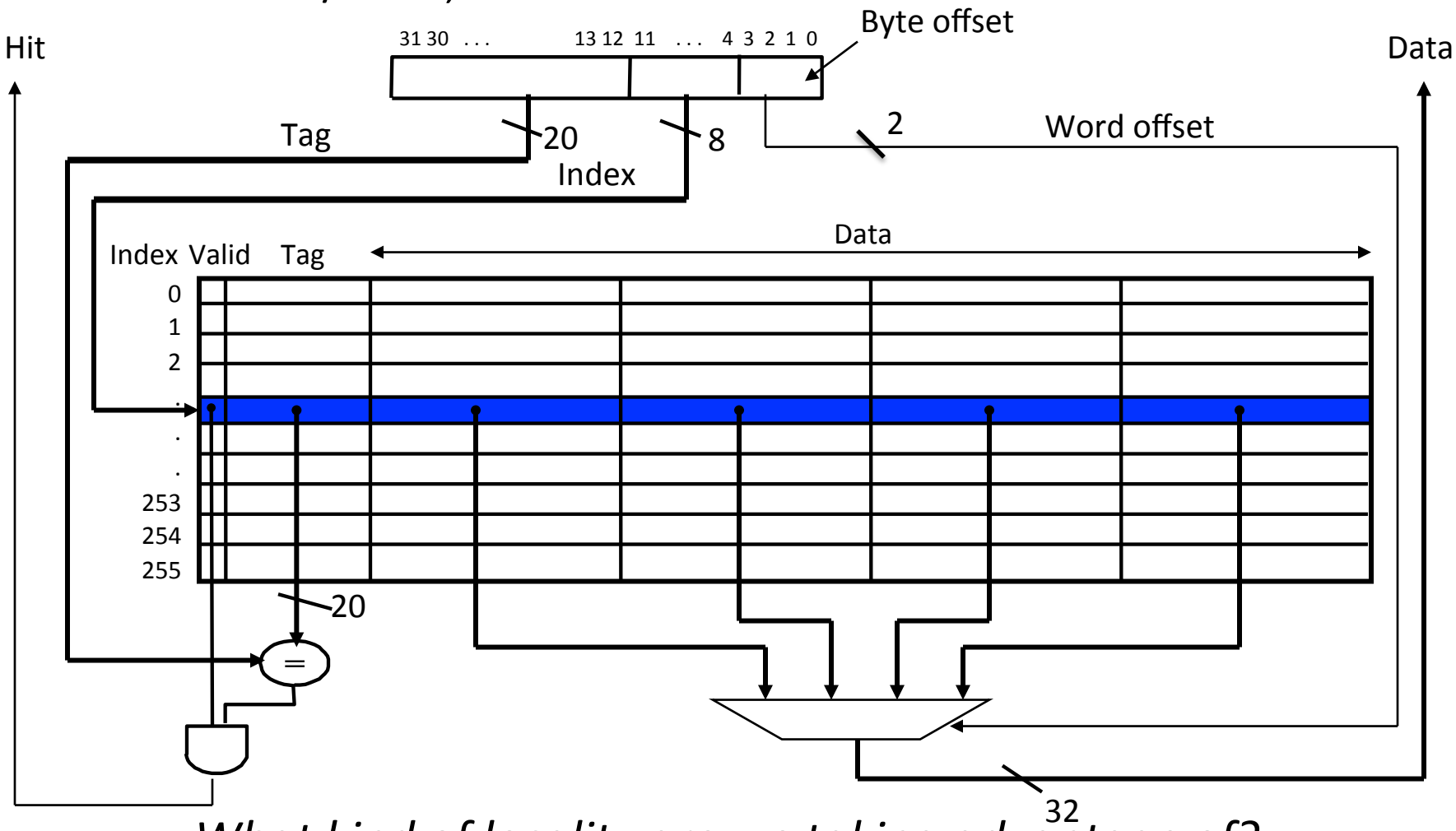
- One word blocks, cache size = 1K words (or 4KB)



What kind of locality are we taking advantage of?

Multiword-Block Direct-Mapped Cache

- Four words/block, cache size = 1K words



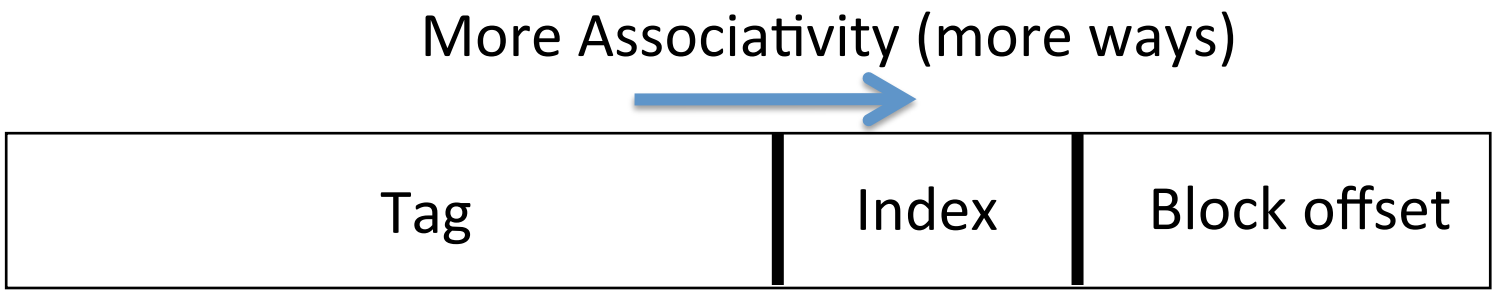
What kind of locality are we taking advantage of?

Alternative Cache Organizations

- "Fully Associative": Block can go anywhere
  - First design in lecture
  - Note: No Index field, but one comparator/block
- "Direct Mapped": Block goes one place
  - Note: Only 1 comparator
  - Number of sets = number blocks
- "N-way Set Associative": N places for a block
  - Number of sets = number of blocks / N
  - N comparators
  - ***Fully Associative: N = number of blocks***
  - ***Direct Mapped: N = 1***

Range of Set-Associative Caches

- For a fixed-size cache, and a given block size, each increase by a factor of two in associativity doubles the number of blocks per set (i.e., the number of "ways") and halves the number of sets –
  - Decreases the size of the index by 1 bit and increases the size of the tag by 1 bit



What if we can also change the block size?

Clickers/Peer Instruction

- For a cache with constant total capacity, if we increase the number of ways by a factor of two, which statement is false:

  A: The number of sets could be doubled

  B: The tag width could decrease

  C: The block size could stay the same

  D: The block size could be halved

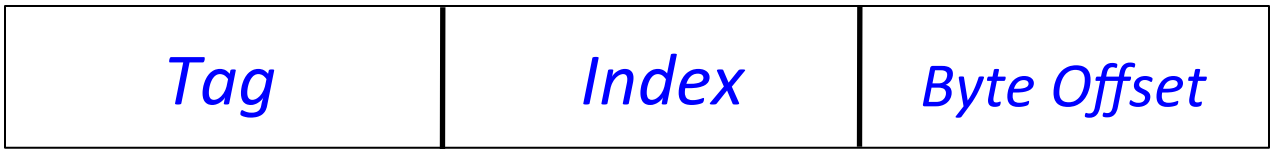
  E: Tag width must increase

Total Cache Capacity =

Associativity * # of sets * block_size

*Bytes = blocks/set * sets * Bytes/block*

*C = N * S * B*



address_size = tag_size + index_size + offset_size

= tag_size + $\log_2$(S) + $\log_2$(B)

Double the Associativity: Number of sets?

tag_size? index_size? # comparators?

Double the Sets: Associativity?

tag_size? index_size? # comparators?

Outline

- Memory Hierarchy and Latency
- Caches Principles
- Basic Cache Organization
- Different Kinds of Caches
- Write Back vs. Write Through
- And in Conclusion …

And In Conclusion, …

- Principle of Locality for Libraries /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
- Cache design choice:
  - Write-Through vs. Write-Back