CS 110
Computer Architecture
Lecture 15:
Caches Part II

Instructors:
Sören Schwertfeger & Chundong Wang

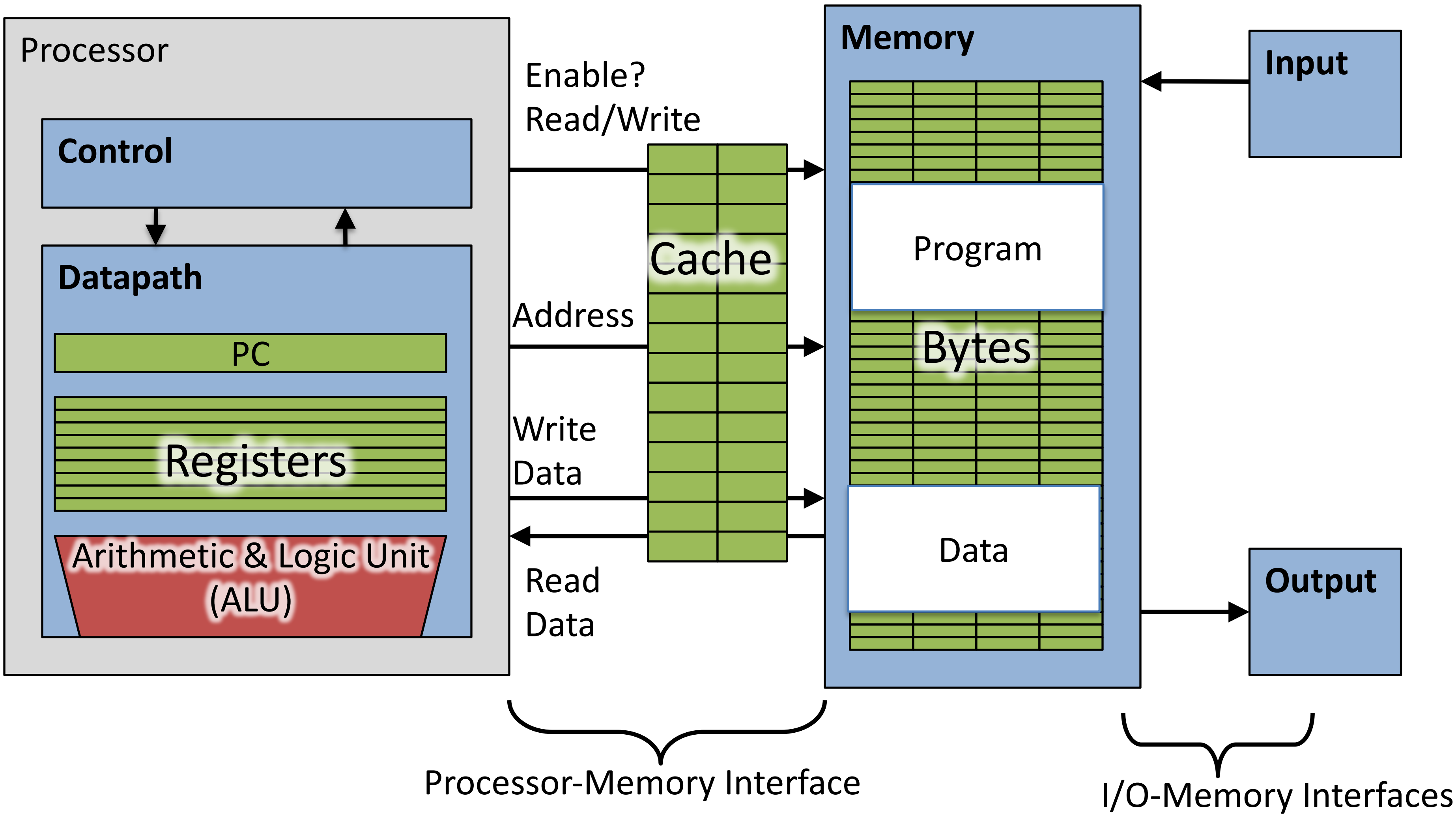
https://robotics.shanghaitech.edu.cn/courses/ca/20s/

School of Information Science and Technology SIST

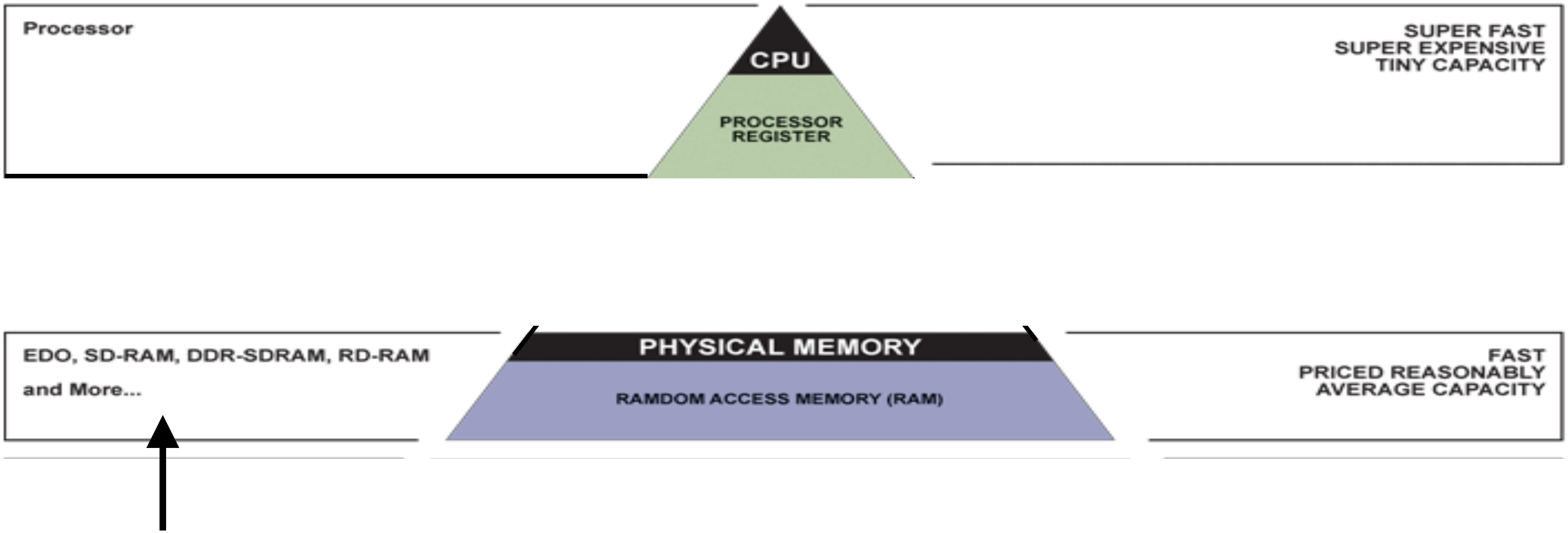
ShanghaiTech University

Slides based on UC Berkley's CS61C

Adding Cache to Computer



Great Idea #3: Principle of Locality / Memory Hierarchy



Note: These names are a bit dated

Big Idea: Locality

- *Temporal Locality* (locality in time)
  - If a memory location is referenced, then it will tend to be referenced again soon

- *Spatial Locality* (locality in space)
  - If a memory location is referenced, the locations with nearby addresses will tend to be referenced soon

Cache Names for Each Organization

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

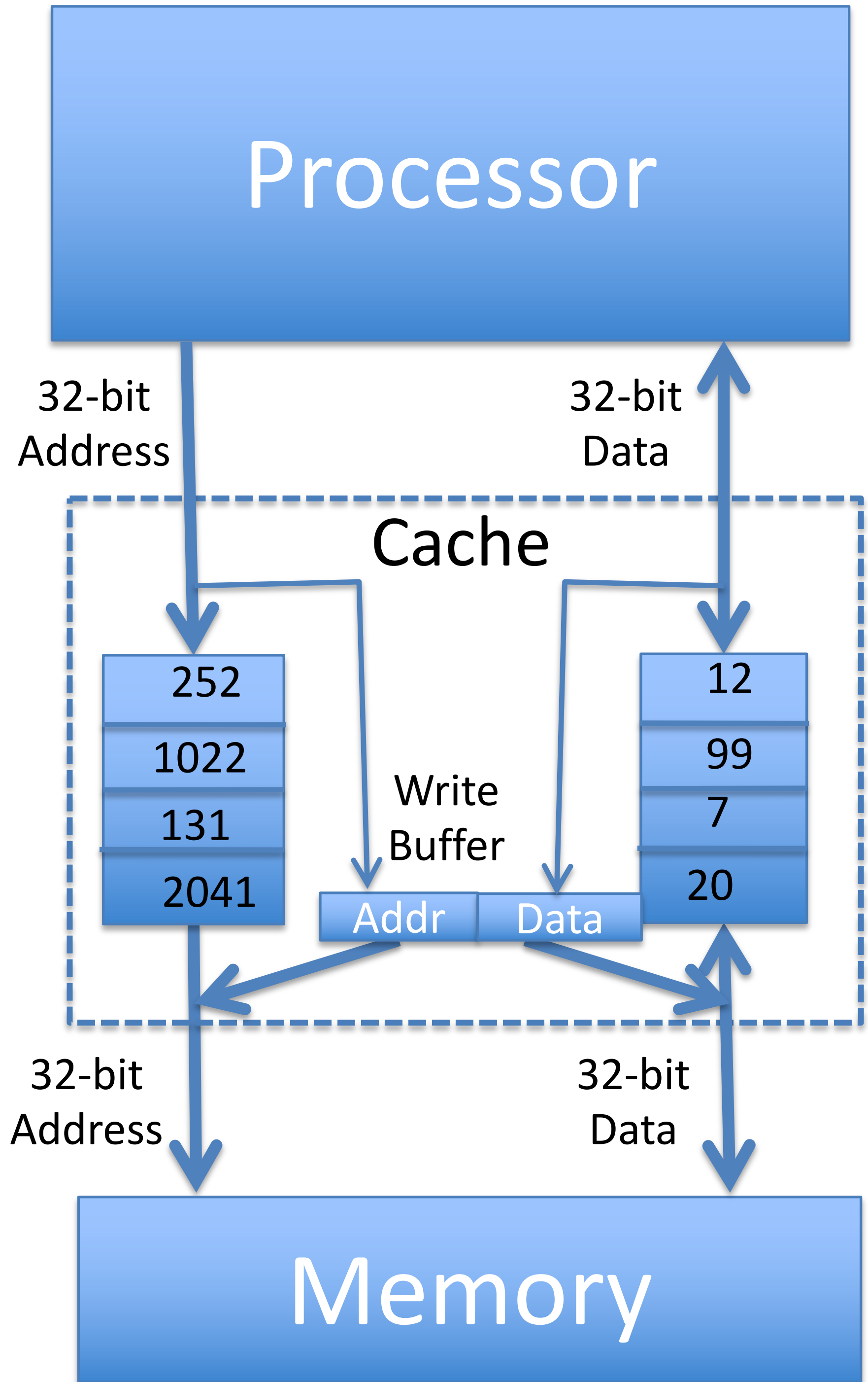
Handling Stores with Write-Through

- Store instructions write to memory, changing values
- Need to make sure cache and memory have same values on writes: 2 policies

1) Write-Through Policy: write cache and write *through* the cache to memory
   - Every write eventually gets to memory
   - Too slow, so include Write Buffer to allow processor to continue once data in Buffer
   - Buffer updates memory in parallel to processor

Write-Through Cache

- Write both values in cache and in memory
- Write buffer stops CPU from stalling if memory cannot keep up
- Write buffer may have multiple entries to absorb bursts of writes
- What if store misses in cache?



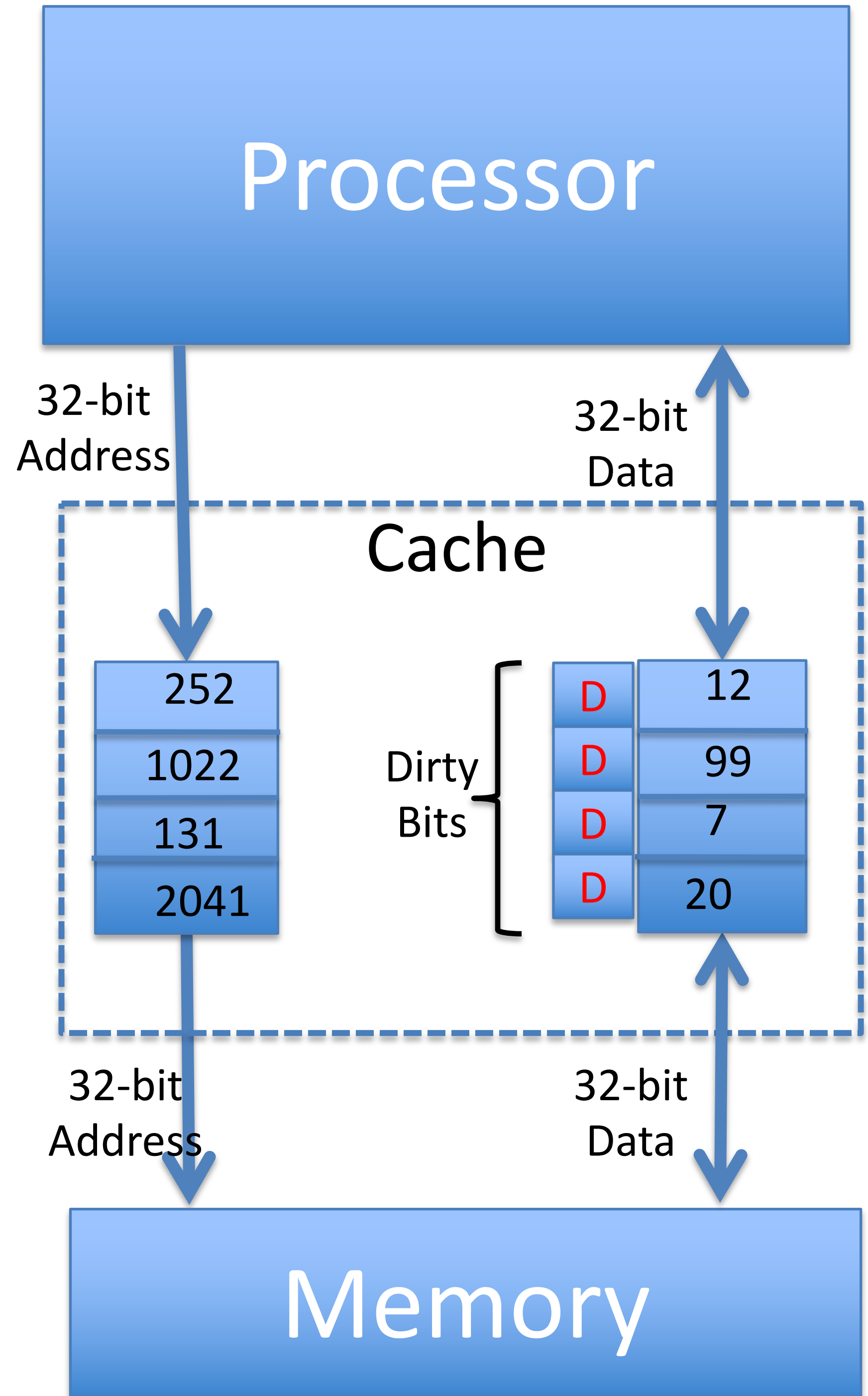
Handling Stores with Write-Back

2) Write-Back Policy: write only to cache and then write cache block *back* to memory when evict block from cache

- Writes collected in cache, only single write to memory per block
- Include bit to see if wrote to block or not, and then only write back if bit is set
  - Called "Dirty" bit (writing makes it "dirty")

Write-Back Cache

- Store/cache hit, write data in cache *only* & set dirty bit
  - Memory has stale value
- Store/cache miss, read data from memory, then update and set dirty bit
  - “Write-allocate” policy
- Load/cache hit, use value from cache
- On any miss, write back evicted block, only if dirty. Update cache with new block and clear dirty bit.



Write-Through vs. Write-Back

- Write-Through:
  - Simpler control logic
  - More predictable timing simplifies processor control logic
  - Easier to make reliable, since memory always has copy of data (big idea: Redundancy!)
- Write-Back
  - More complex control logic
  - More variable timing (0,1,2 memory accesses per cache access)
  - Usually reduces write traffic
  - Harder to make reliable, sometimes cache has only copy of data

Cache (*Performance*) Terms

- Hit rate: fraction of accesses that hit in the cache
- Miss rate: 1 – Hit rate
- Miss penalty: time to replace a line/ block from lower level in memory hierarchy to cache
- Hit time: time to access cache memory (including tag comparison)

- Abbreviation: “$” = cache (cash …)

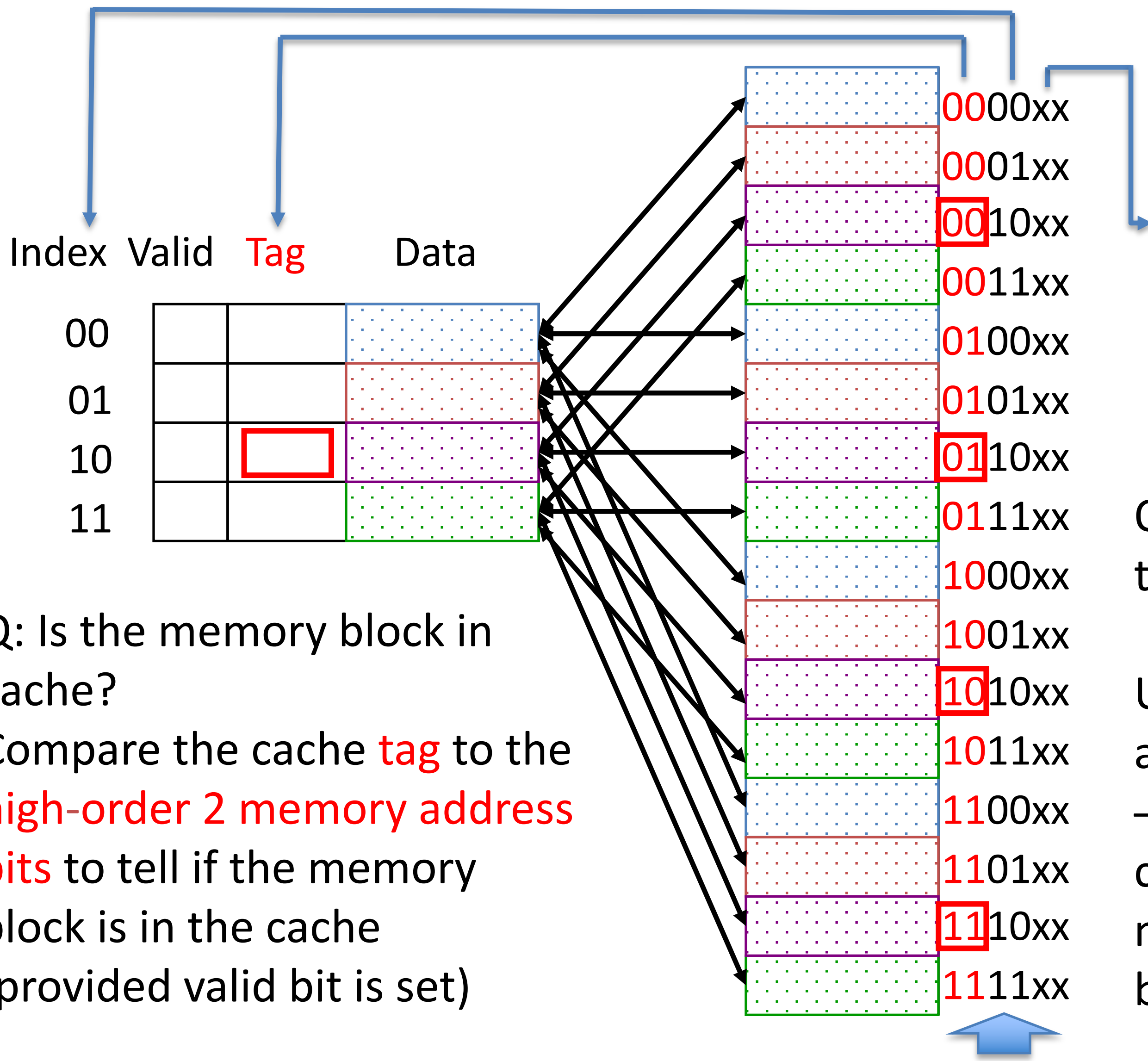
Average Memory Access Time (AMAT)

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

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

Direct Mapped Cache Example

Cache **Main Memory**



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?

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

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)

6bit Memory Address

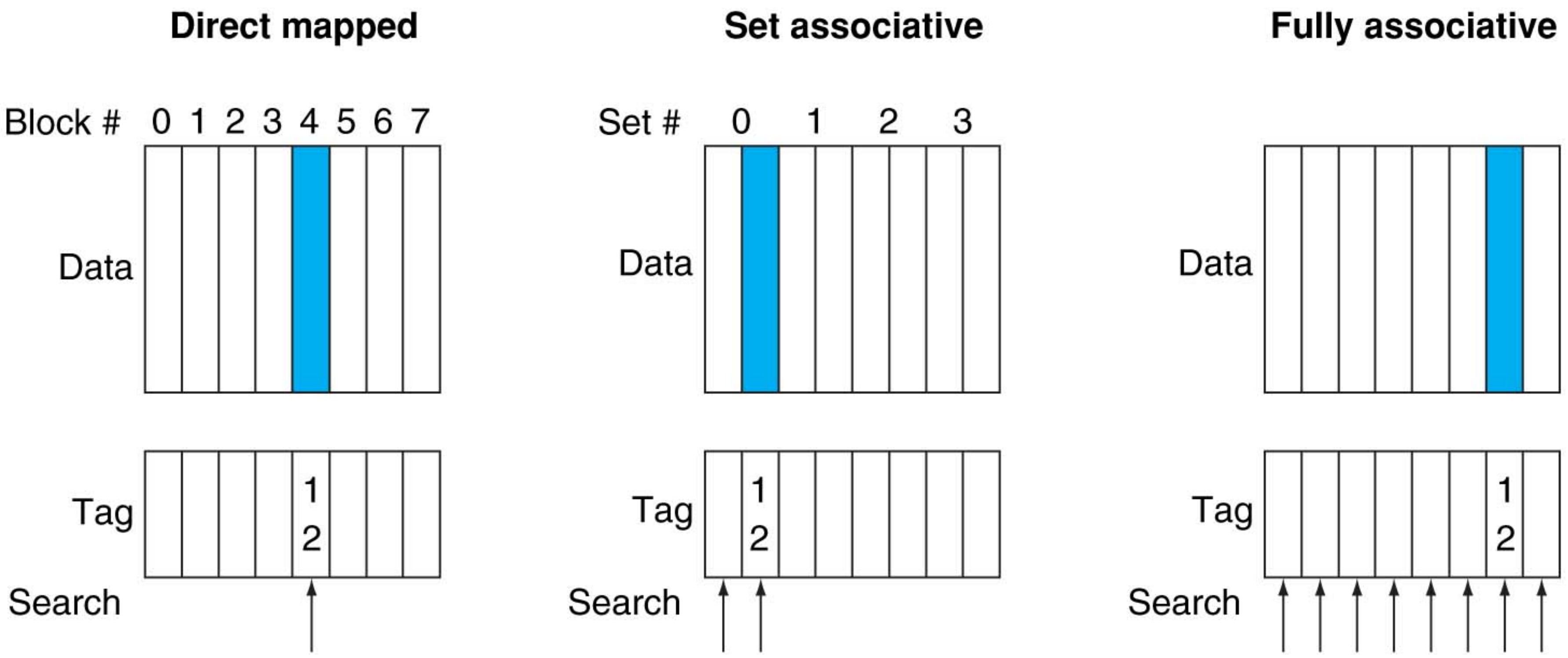
Example: Direct-Mapped Cache with 4 Single-Word Blocks, Worst-Case Reference String

- Consider the main memory address (words) reference string of word numbers: 0 4 0 4 0 4 0 4

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

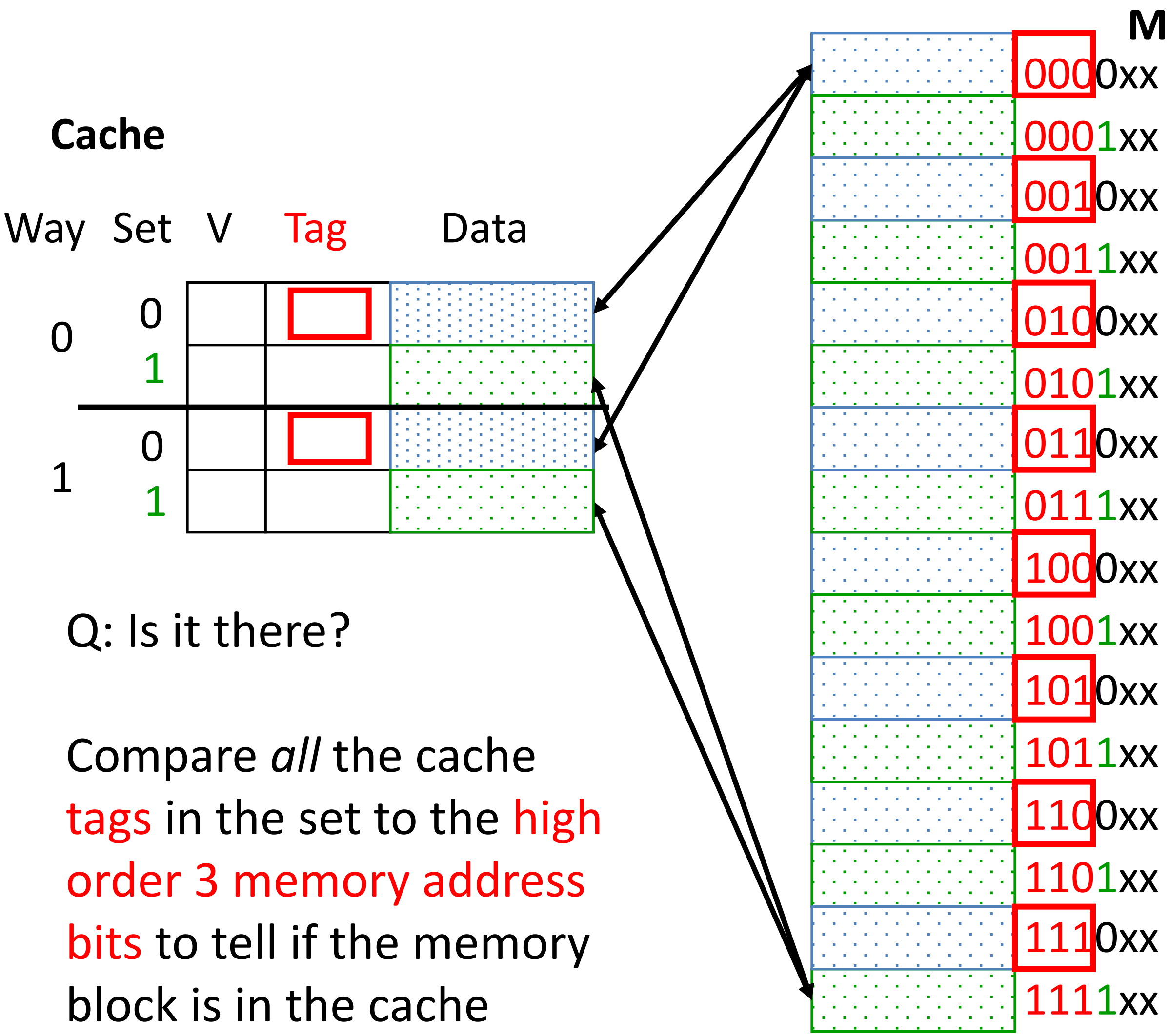
  - 8 requests, 8 misses
- Ping-pong effect due to conflict misses - two memory locations that map into the same cache block

Alternative Block Placement Schemes



- DM placement: mem block 12 in 8 block cache: only one cache block where mem block 12 can be found—(12 modulo 8) = 4
- SA placement: four sets x 2-ways (8 cache blocks), memory block 12 in set (12 mod 4) = 0; either element of the set
- FA placement: mem block 12 can appear in any cache blocks

Example: 2-Way Set Associative $
(4 words = 2 sets x 2 ways per set)



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

Q: Is it there?

Compare *all* the cache tags in the set to the high order 3 memory address bits to tell if the memory block is in the cache

Q: How do we find it?

Use next 1 low order memory address bit to determine which cache set (i.e., modulo the number of sets in the cache)

Example: 4-Word 2-Way SA $ Same Reference String

- Consider the main memory address (word) reference string

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

0 4 0 4 0 4 0 4

0 miss

000	Mem(0)

4 miss

000	Mem(0)
010	Mem(4)

0 hit

000	Mem(0)
010	Mem(4)

4 hit

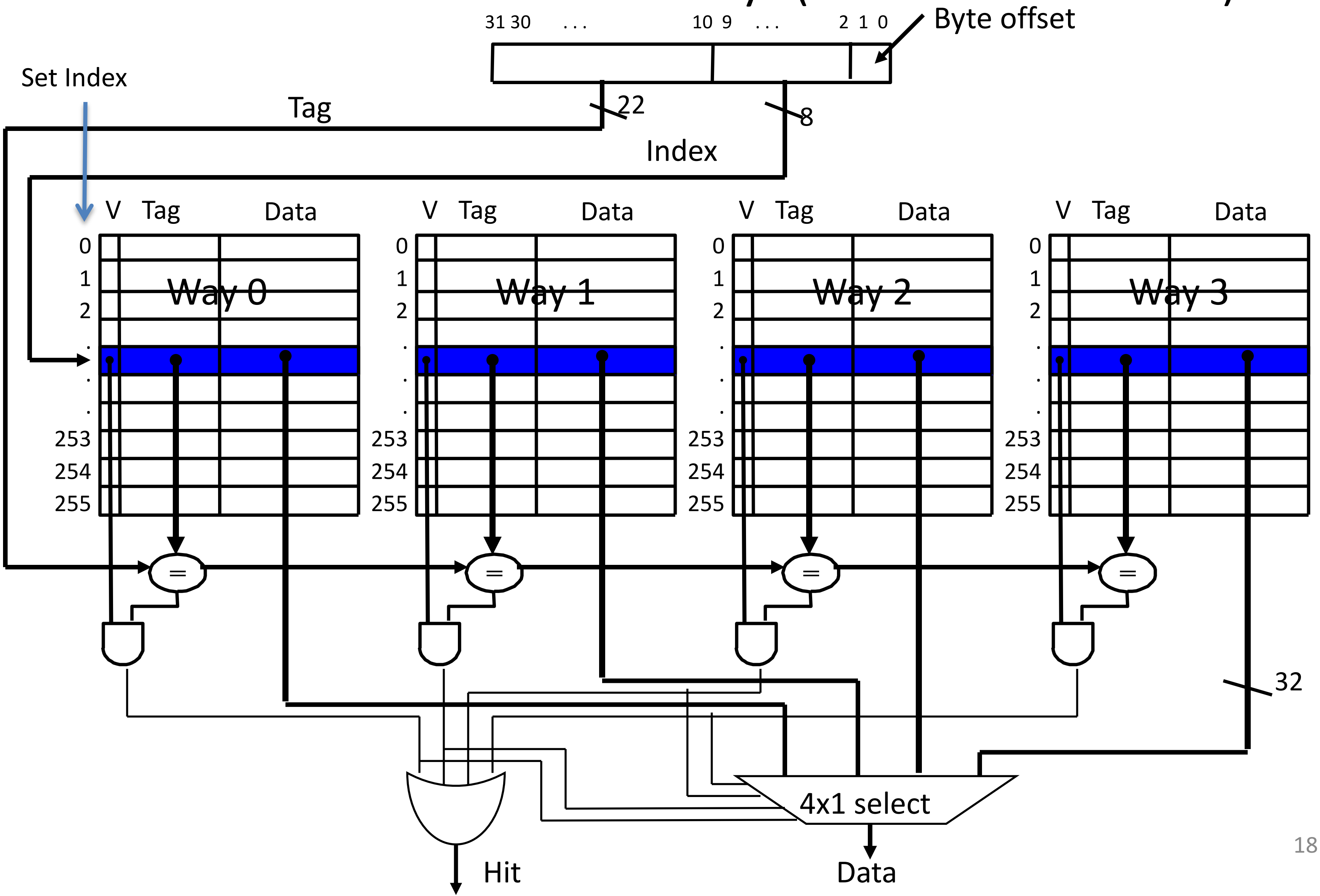
000	Mem(0)
010	Mem(4)

  - 8 requests, 2 misses

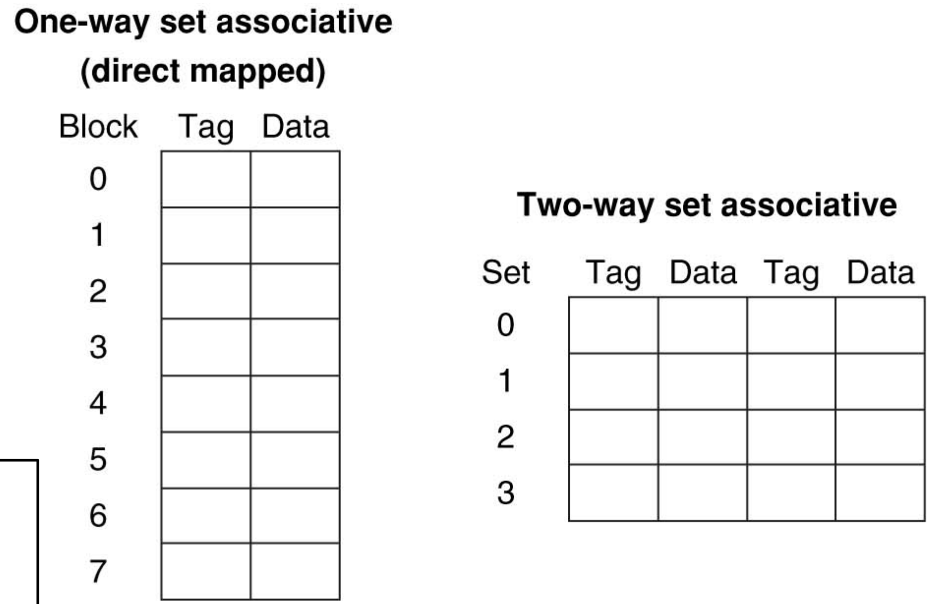
- Solves the ping-pong effect in a direct-mapped cache due to conflict misses since now two memory locations that map into the same cache set can co-exist!

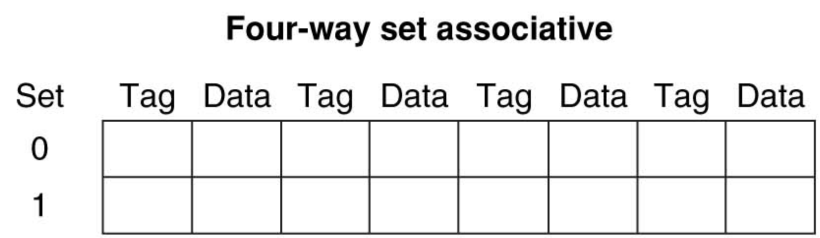
Four-Way Set-Associative Cache

- 2^8 = 256 sets each with four ways (each with one block)

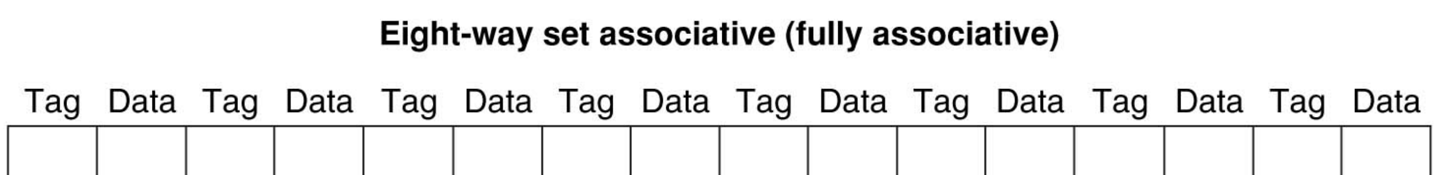


Different Organizations of an Eight-Block Cache



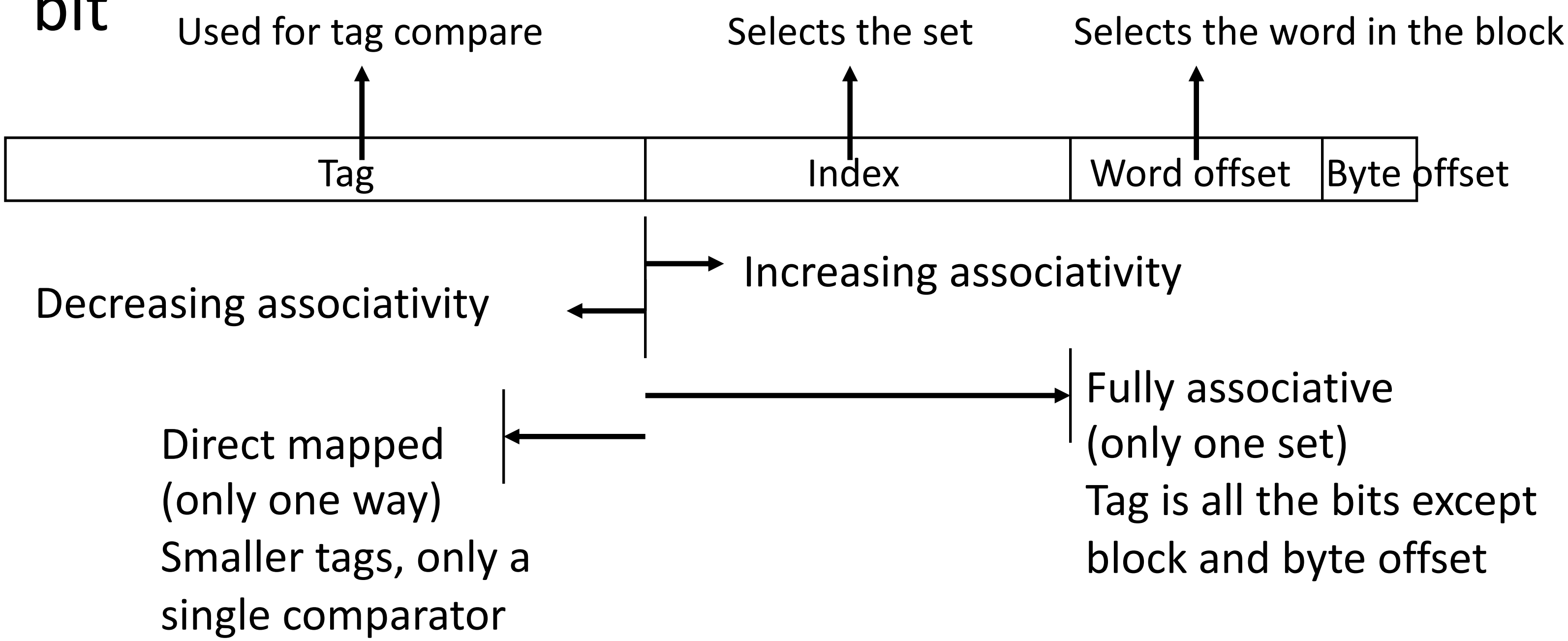


Total size of $ in blocks is equal to *number of sets* × *associativity*. For fixed $ size and fixed block size, increasing associativity decreases number of sets while increasing number of elements per set. With eight blocks, an 8-way set-associative $ is same as a fully associative $.



Range of Set-Associative Caches

- For a *fixed-size* cache and fixed block size, each increase by a factor of two in associativity doubles the number of blocks per set (i.e., the number or 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

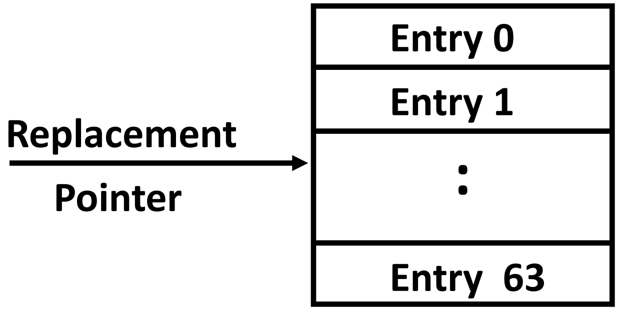


Costs of Set-Associative Caches

- N-way set-associative cache costs
  - N comparators (delay and area)
  - MUX delay (set selection) before data is available
  - Data available after set selection (and Hit/Miss decision). DM $: block is available before the Hit/Miss decision
    - In Set-Associative, not possible to just assume a hit and continue and recover later if it was a miss
- When miss occurs, which way's block selected for replacement?
  - Least Recently Used (LRU): one that has been unused the longest (principle of temporal locality)
    - Must track when each way's block was used relative to other blocks in the set
    - For 2-way SA $, one bit per set → set to 1 when a block is referenced; reset the other way's bit (i.e., "last used")

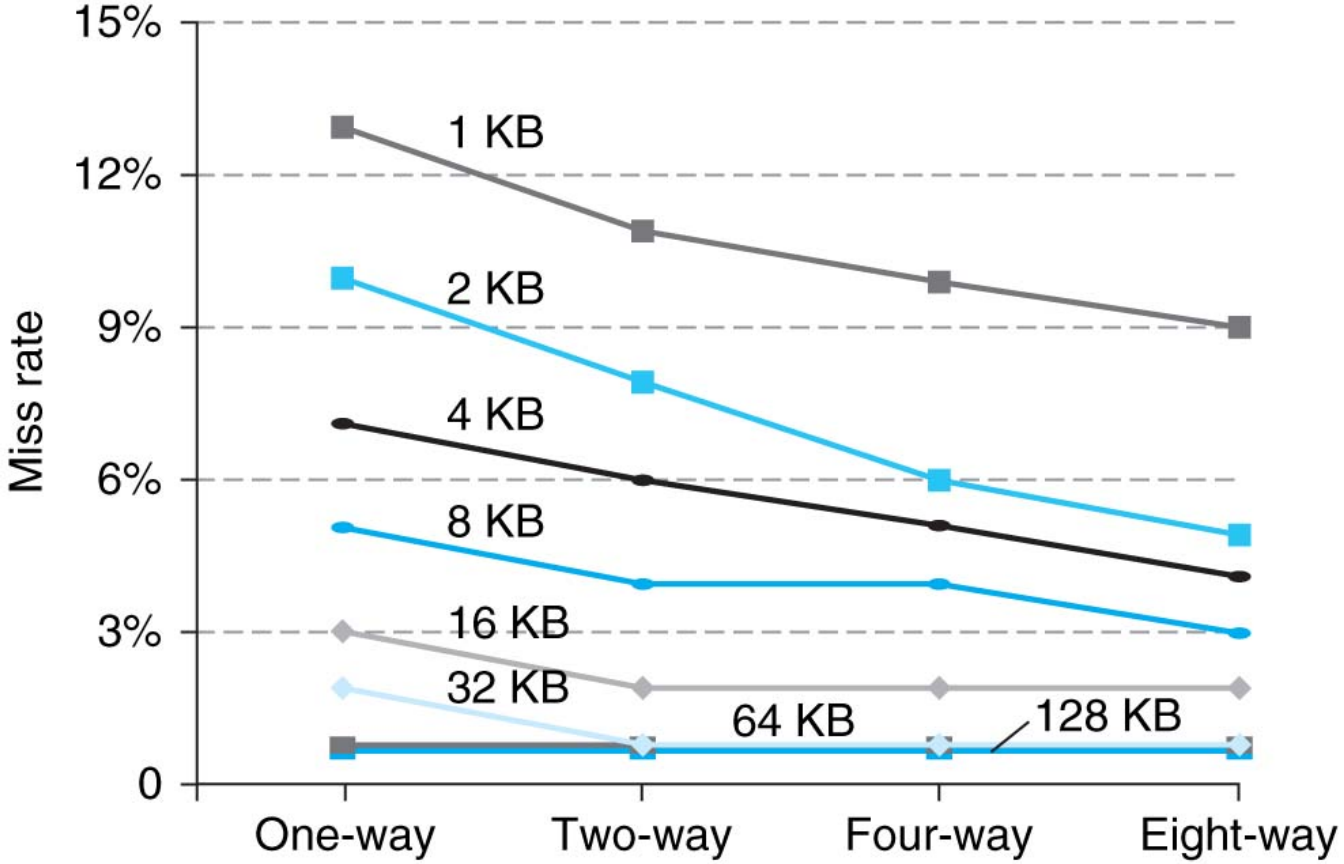
Cache Replacement Policies

- Random Replacement
  - Hardware randomly selects a cache evict
- Least-Recently Used
  - Hardware keeps track of access history
  - Replace the entry that has not been used for the longest time
  - For 2-way set-associative cache, need one bit for LRU replacement
- Example of a Simple "Pseudo" LRU Implementation
  - Assume 64 Fully Associative entries
  - Hardware replacement pointer points to one cache entry
  - Whenever access is made to the entry the pointer points to:
    - Move the pointer to the next entry
  - Otherwise: do not move the pointer
  - (example of "not-most-recently used" replacement policy)



Benefits of Set-Associative Caches

- Choice of DM $ versus SA $ depends on the cost of a miss versus the cost of implementation



- Largest gains are in going from direct mapped to 2-way (20%+ reduction in miss rate)

TA Discussion

Peifan Li

Video

Q & A

Quiz

Quiz

Piazza: "Online Lecture 15 $ Poll"

AMAT = Time for a hit + Miss rate x Miss penalty

Given a 200 psec clock, a miss penalty of 50 clock cycles, a miss rate of 0.02 misses per instruction and a cache hit time of 1 clock cycle, what is AMAT?

- ☐ A: ≤200 psec
- ☐ B: 400 psec
- ☐ C: 600 psec
- ☐ D: ≥ 800 psec

CS 110
Computer Architecture
Lecture 15:
Caches Part II
Video 2: 3Cs

Instructors:

Sören Schwertfeger & Chundong Wang

https://robotics.shanghaitech.edu.cn/courses/ca/20s/

School of Information Science and Technology SIST

ShanghaiTech University

Slides based on UC Berkley's CS61C

Understanding Cache Misses: The 3Cs

- Compulsory (cold start or process migration, 1st reference):
  - First access to block impossible to avoid; small effect for long running programs
  - Solution: increase block size (increases miss penalty; very large blocks could increase miss rate)
- Capacity:
  - Cache cannot contain all blocks accessed by the program
  - Solution: increase cache size (may increase access time)
- *Conflict (collision):*
  - *Multiple memory locations mapped to the same cache location*
  - *Solution 1: increase cache size*
  - *Solution 2: increase associativity (may increase access time)*

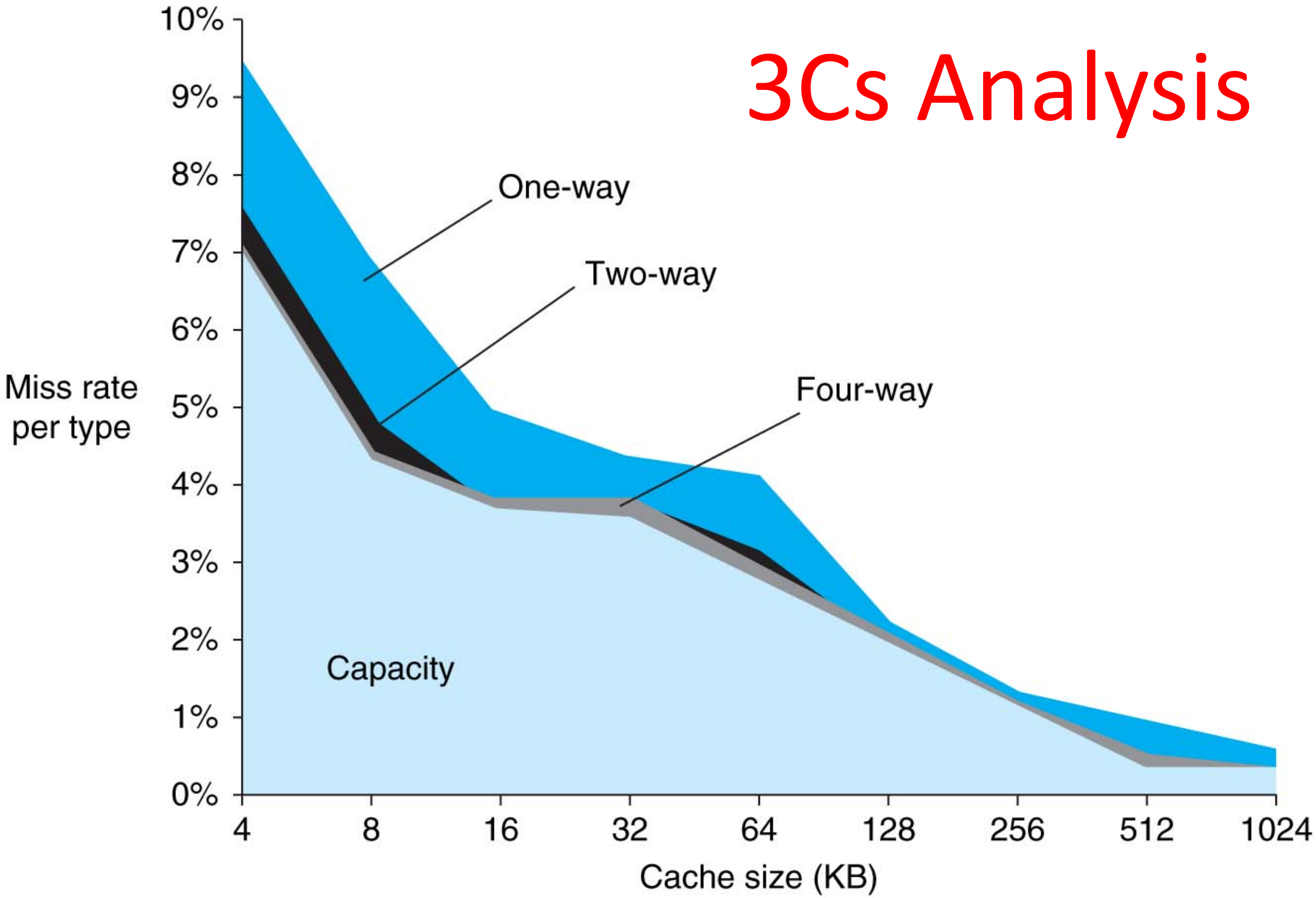
Prefetching...

- Programmer/Compiler: I know that, later on, I will need this data...
- Tell the computer to ***prefetch*** the data
  - Can be as an explicit prefetch instruction
  - Or an implicit instruction: **lw x0 0(t0)**
    - Won't stall the pipeline on a cache miss: The processor control logic recognizes this situation
- Allows you to hide the cost of compulsory misses
  - You still need to fetch the data however

How to Calculate 3C's using Cache Simulator

1. *Compulsory*: set cache size to infinity and fully associative, and count number of misses
2. *Capacity*: Change cache size from infinity, usually in powers of 2, and count misses for each reduction in size
   – 16 MB, 8 MB, 4 MB, … 128 KB, 64 KB, 16 KB
3. *Conflict*: Change from fully associative to n-way set associative while counting misses
   – Fully associative, 16-way, 8-way, 4-way, 2-way, 1-way

3Cs Analysis



- Three sources of misses (SPEC2000 integer and floating-point benchmarks)
  - Compulsory misses 0.006%; not visible
  - Capacity misses, function of cache size
  - Conflict portion depends on associativity and cache size

Improving Cache Performance

AMAT = Time for a hit + Miss rate x Miss penalty

- Note: miss penalty is **additional** time for cache miss
- Reduce the time to hit in the cache
  - E.g., Smaller cache
- Reduce the miss rate
  - E.g., Bigger cache
    Longer cache lines (somewhat: improves ability to exploit spatial locality at the cost of reducing the ability to exploit temporal locality)
  - E.g., Better programs!
- Reduce the miss penalty
  - E.g., Use multiple cache levels

Impact of Larger Cache on AMAT?

- 1) Reduces misses (what kind(s)?)
- 2) Longer Access time (Hit time): smaller is faster
  - Increase in hit time will likely add another stage to the pipeline
- At some point, increase in hit time for a larger cache may overcome the improvement in hit rate, yielding a decrease in performance
- Computer architects expend considerable effort optimizing organization of cache hierarchy – big impact on performance and power!

Questions: Impact of longer cache blocks on misses?

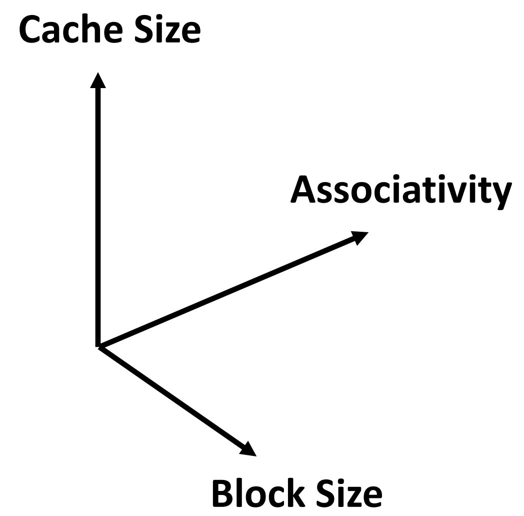
- For fixed total cache capacity and associativity, what is effect of longer blocks on each type of miss:
  - A: Decrease, B: Unchanged, C: Increase
- Compulsory?
- Capacity?
- Conflict?

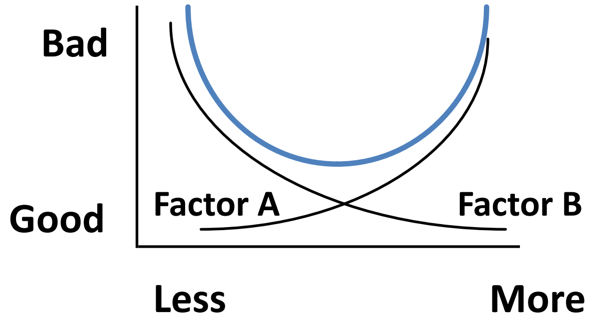
Questions: Impact of longer blocks on AMAT

- For fixed total cache capacity and associativity, what is effect of longer blocks on each component of AMAT:
  - A: Decrease, B: Unchanged, C: Increase
- Hit Time?
- Miss Rate?
- Miss Penalty?

Cache Design Space

- Several interacting dimensions
  - Cache size
  - Block size
  - Associativity
  - Replacement policy
  - Write-through vs. write-back
  - Write allocation
- Optimal choice is a compromise
  - Depends on access characteristics
    - Workload
    - Use (I-cache, D-cache)
  - Depends on technology / cost
- Simplicity often wins





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

Quiz

Piazza: “Video Lecture 15 $ Poll”

For fixed capacity and fixed block size, how does increasing associativity effect AMAT?

A: Increases hit time, decreases miss rate

B: Decreases hit time, decreases miss rate

C: Increases hit time, increases miss rate

D: Decreases hit time, increases miss rate