CMSC 611: Advanced Computer Architecture

Cache

Some material adapted from Mohamed Younis, UMBC CMSC 611 Spr 2003 course slides

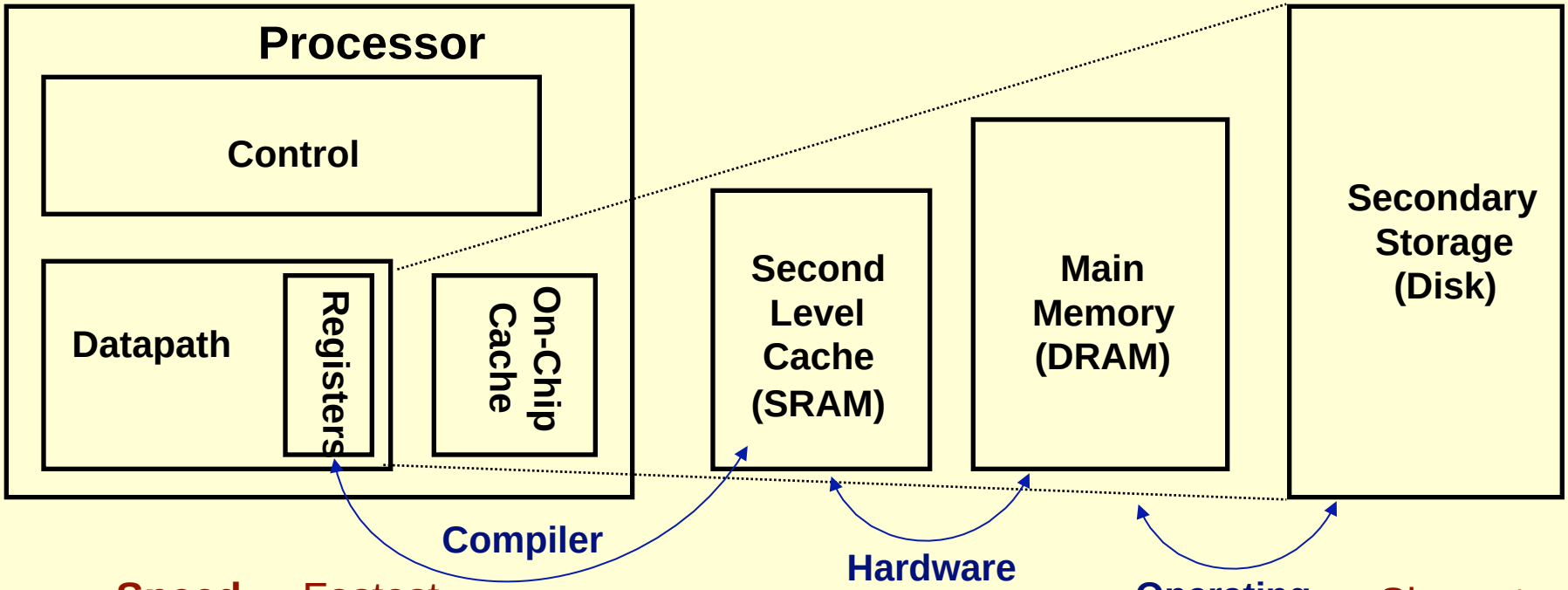
Memory Hierarchy

- Temporal Locality (Locality in Time):

  ⇒ Keep most recently accessed data items closer to the processor

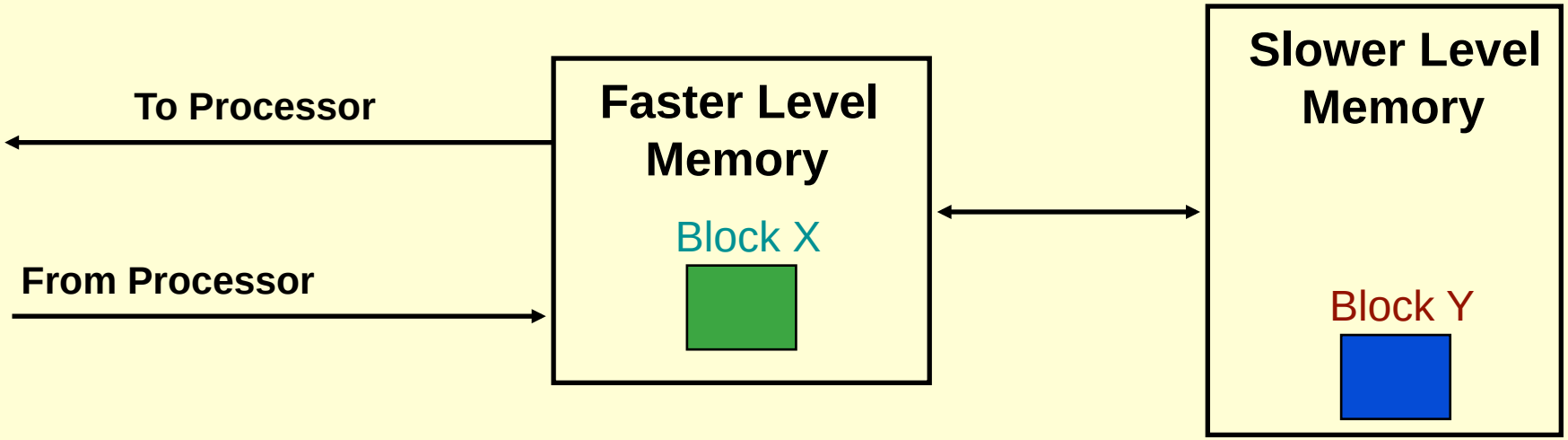
- Spatial Locality (Locality in Space):

  ⇒ Move blocks consists of contiguous words to the faster levels



Memory Hierarchy Terminology

- Hit: data appears in some block in the faster level (example: Block X)
  - Hit Rate: the fraction of memory access found in the faster level
  - Hit Time: Time to access the faster level which consists of
    - Memory access time + Time to determine hit/miss
- Miss: data needs to be retrieve from a block in the slower level (Block 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



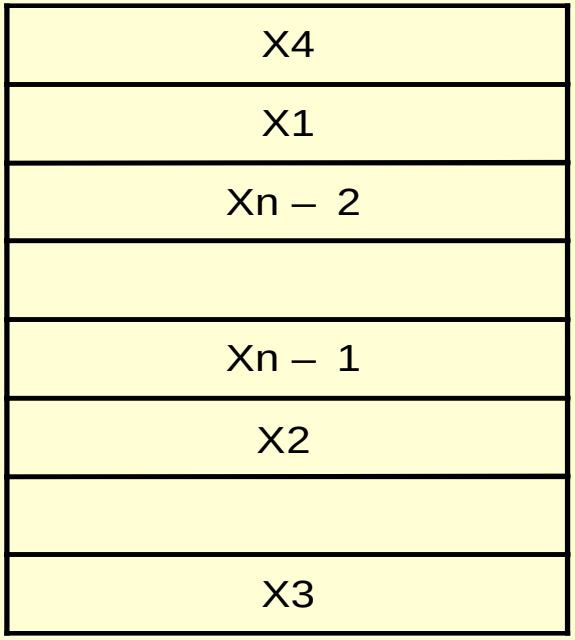
Memory Hierarchy Design Issues

- Block identification
  - How is a block found if it is in the upper (faster) level?
    - Tag/Block
- Block placement
  - Where can a block be placed in the upper (faster) level?
    - Fully Associative, Set Associative, Direct Mapped
- Block replacement
  - Which block should be replaced on a miss?
    - Random, LRU
- Write strategy
  - What happens on a write?
    - Write Back or Write Through (with Write Buffer)

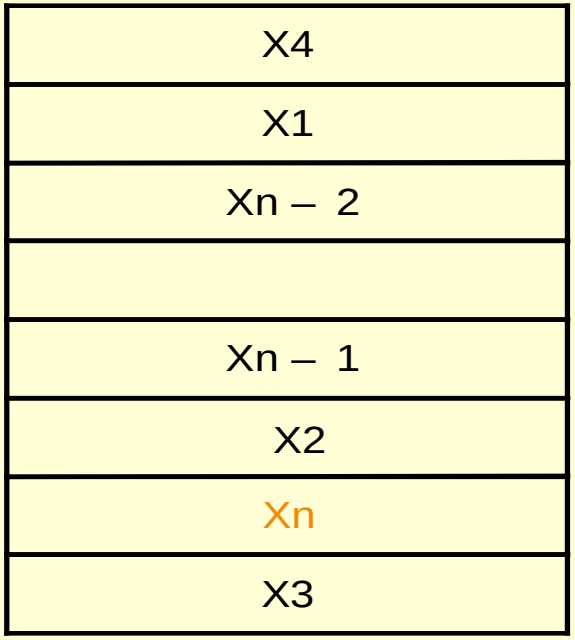
The Basics of Cache

- Cache: level of hierarchy closest to processor
- Caches first appeared in research machines in early 1960s
- Virtually every general-purpose computer produced today includes cache

Requesting X_n generates a miss and the word X_n will be brought from main memory to cache



a. Before the reference to Xn

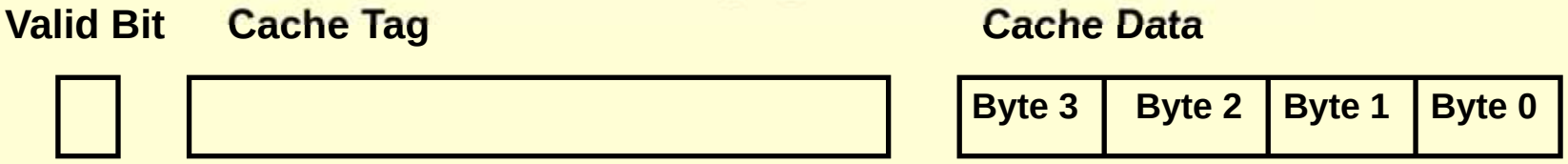


b. After the reference to Xn

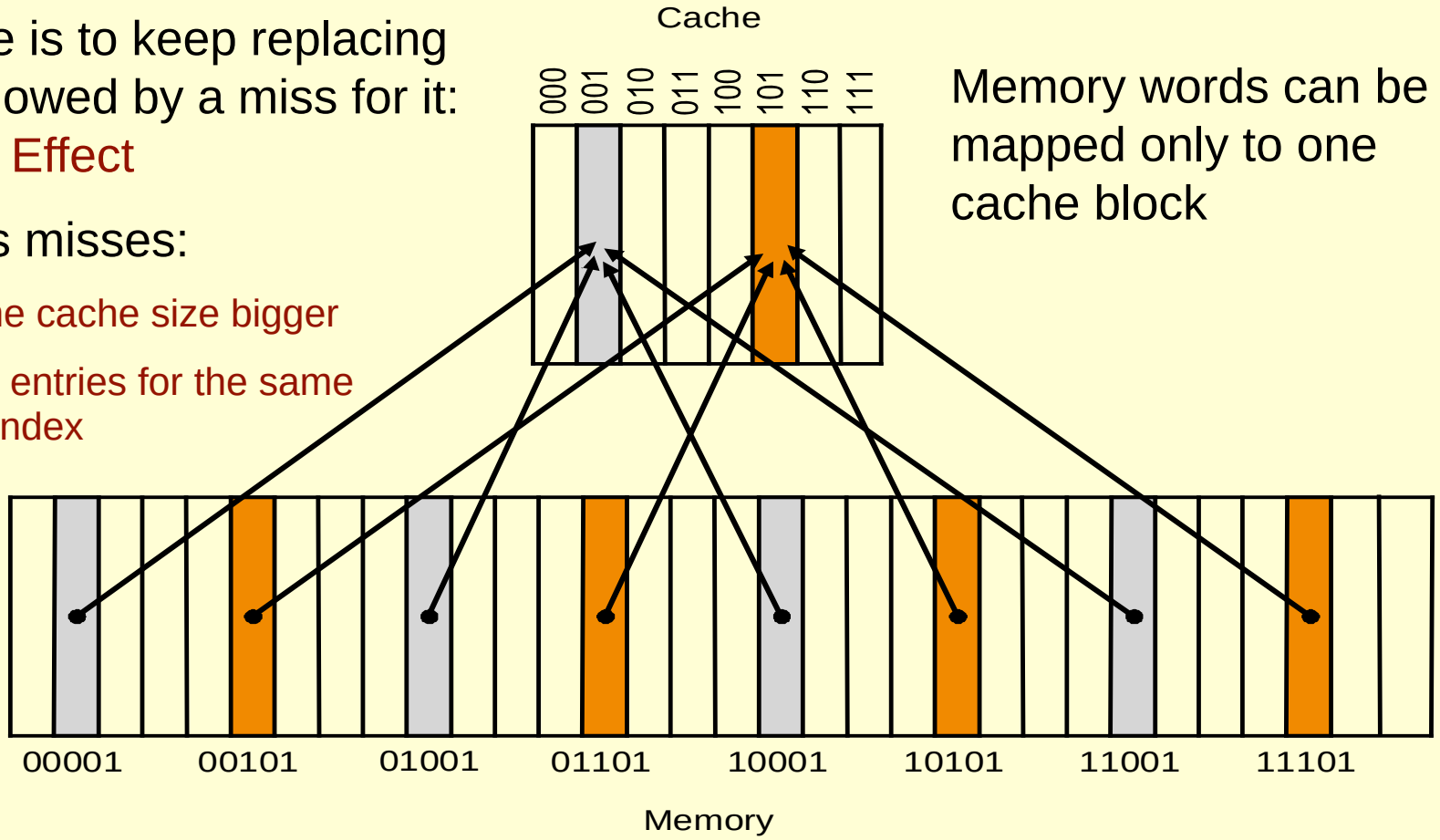
Issues:

- How do we know that a data item is in cache?
- If so, How to find it?

Direct-Mapped Cache



- Worst case is to keep replacing a block followed by a miss for it: Ping Pong Effect
- To reduces misses:
  - make the cache size bigger
  - multiple entries for the same Cache Index



Cache block address = (Block address) modulo (Number of cache blocks)

Accessing Cache

- Cache Size depends on:
  - # cache blocks
  - # address bits
  - Word size
- Example:
  - For n-bit address, 4-byte word & 1024 cache blocks:
  - cache size =

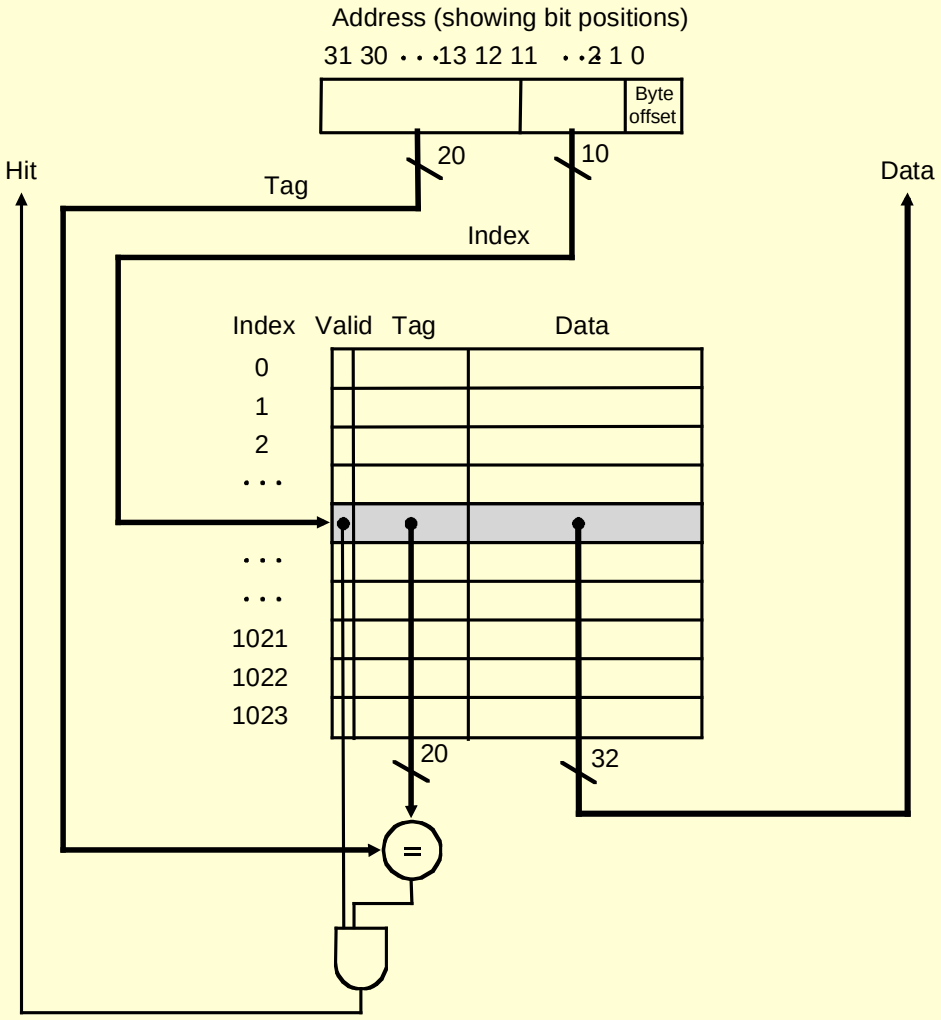
1024 [(n-10 -2) + 1 + 32] bit

 # cache blocks

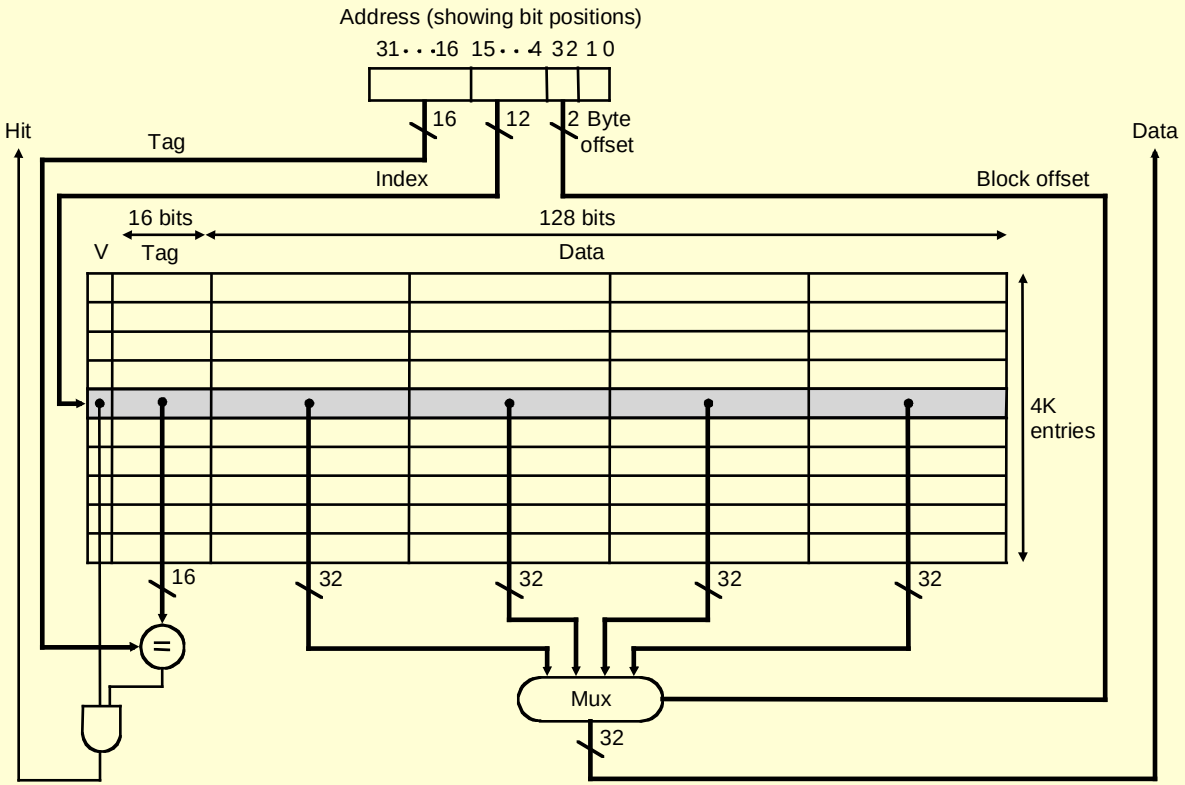
Tag

Valid bit

Word size



Cache with Multi-Word/Block

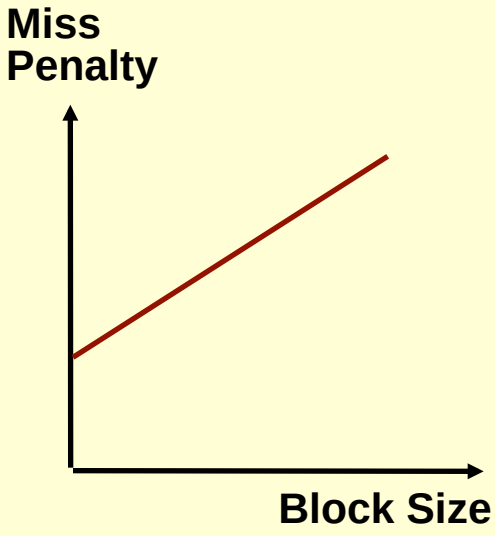


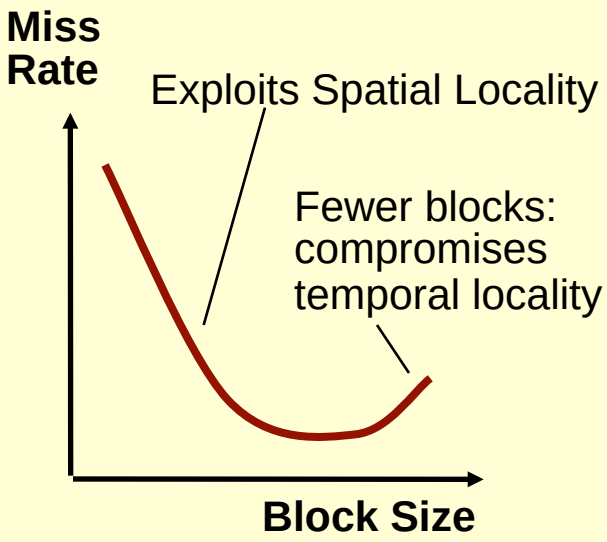
- Takes advantage of spatial locality to improve performance
- Cache block address = (Block address) modulo (Number of cache blocks)
- Block address = (byte address) / (bytes per block)

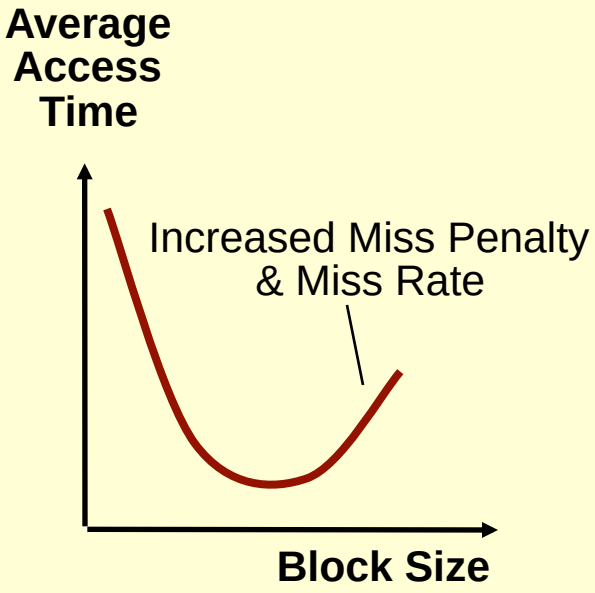
Determining Block Size

- Larger block size take advantage of spatial locality BUT:
  - Larger block size means larger miss penalty:
    - Takes longer time to fill up the block
  - If block size is too big relative to cache size, miss rate will go up
    - Too few cache blocks
- Average Access Time =

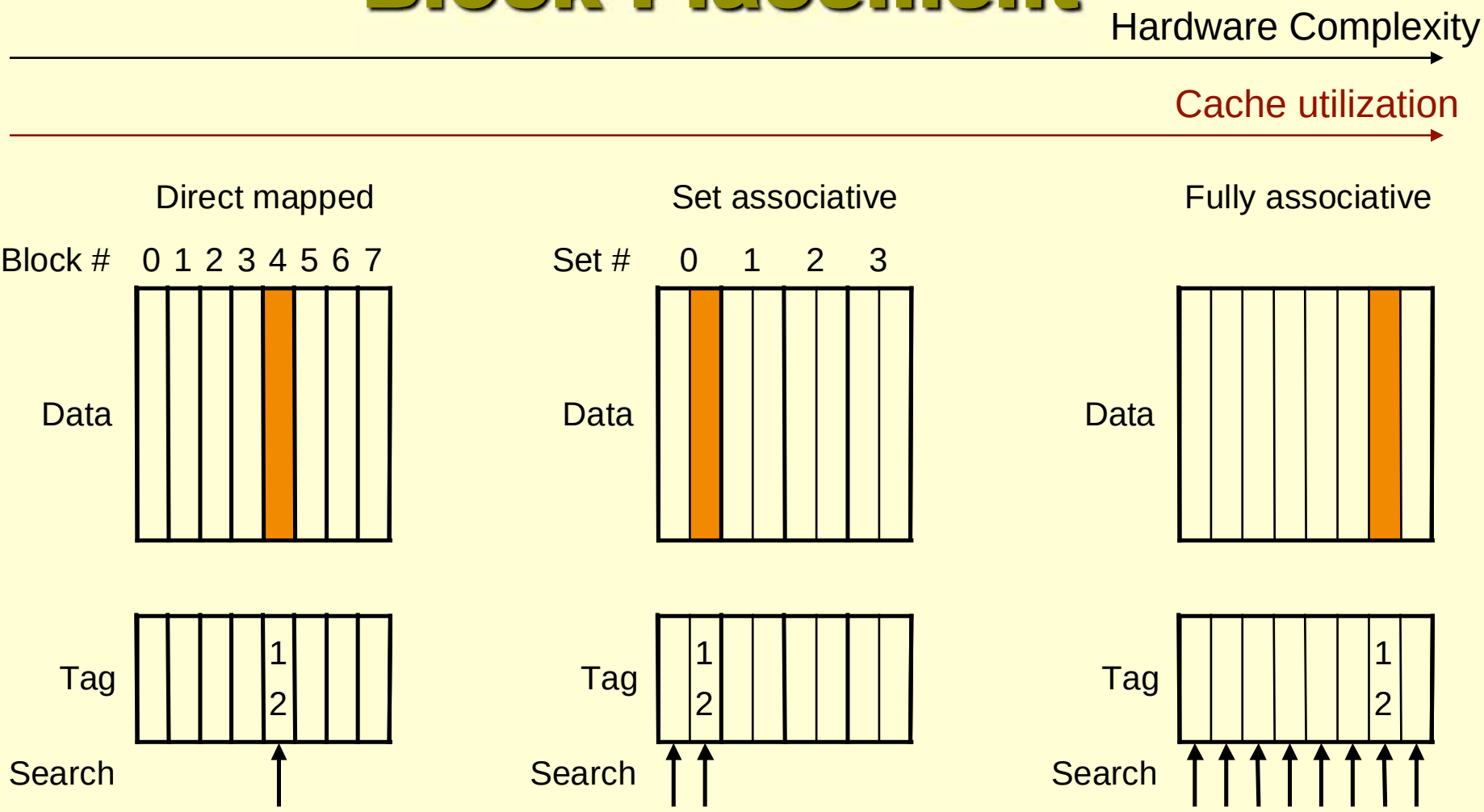
  Hit Time * (1 - Miss Rate) + Miss Penalty * Miss Rate







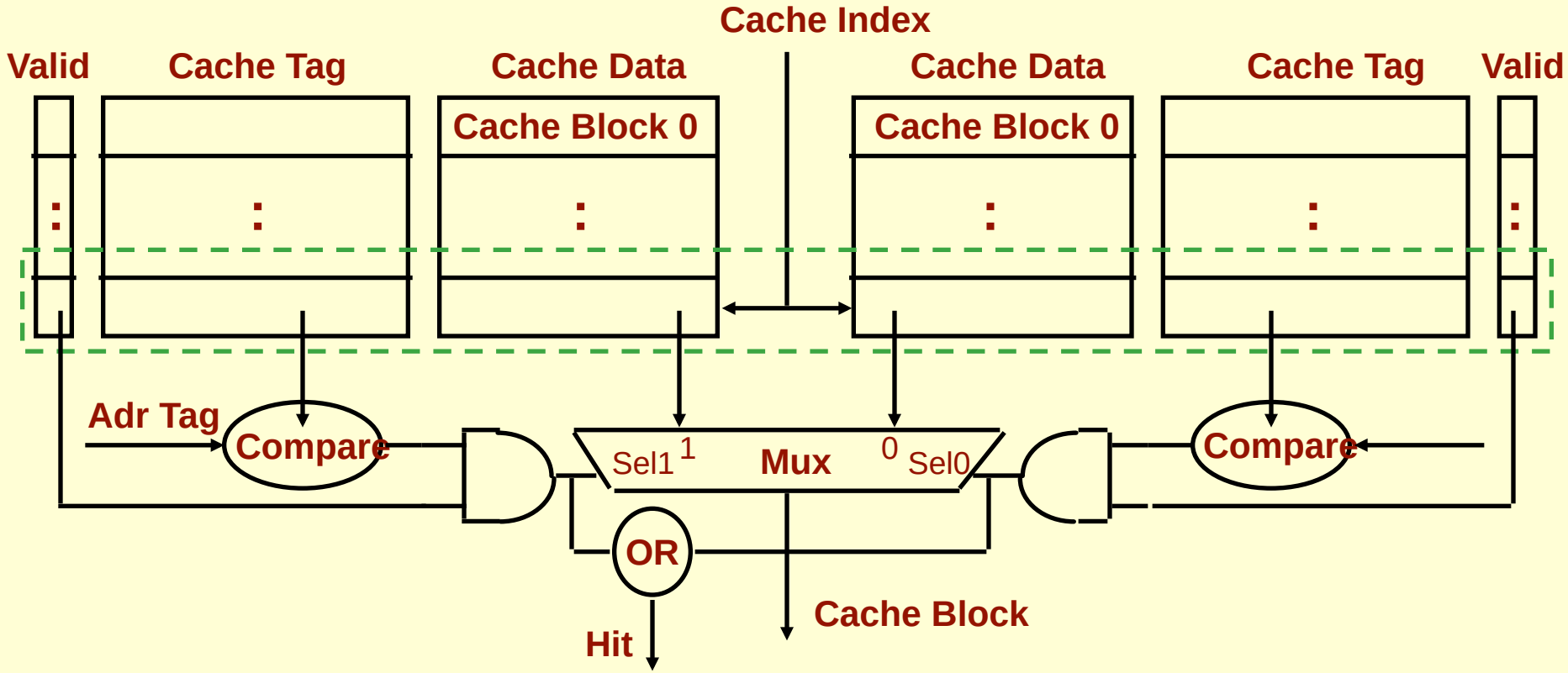
Block Placement



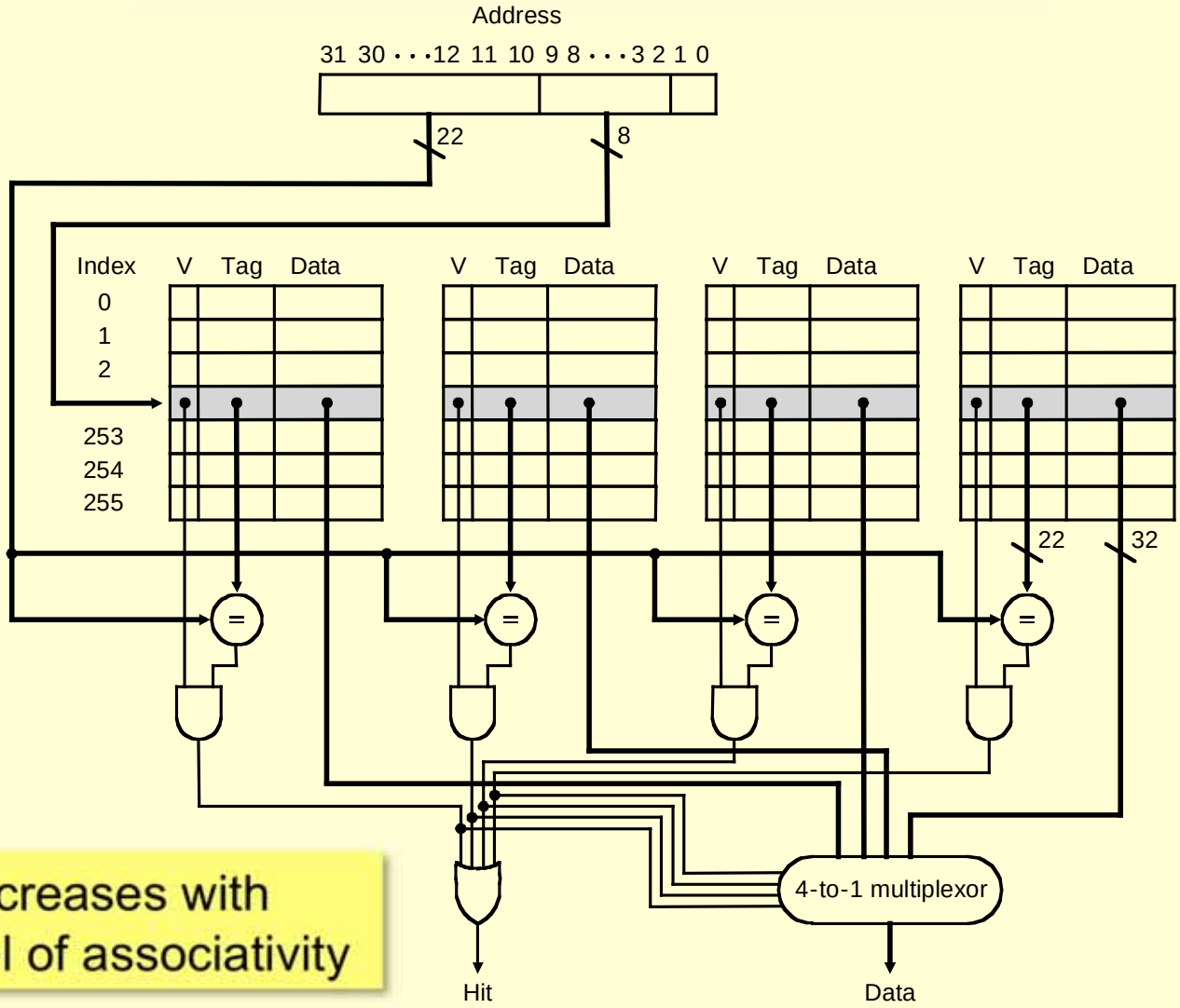
- Set number = (Block number) modulo (Number of sets in the cache)
- Increased flexibility of block placement reduces probability of cache misses

N-way Set Associative Cache

- N entries for each Cache Index
- Example: Two-way set associative cache
  - Cache Index selects a “set” from the cache
  - The two tags in the set are compared in parallel
  - Data is selected based on the tag result



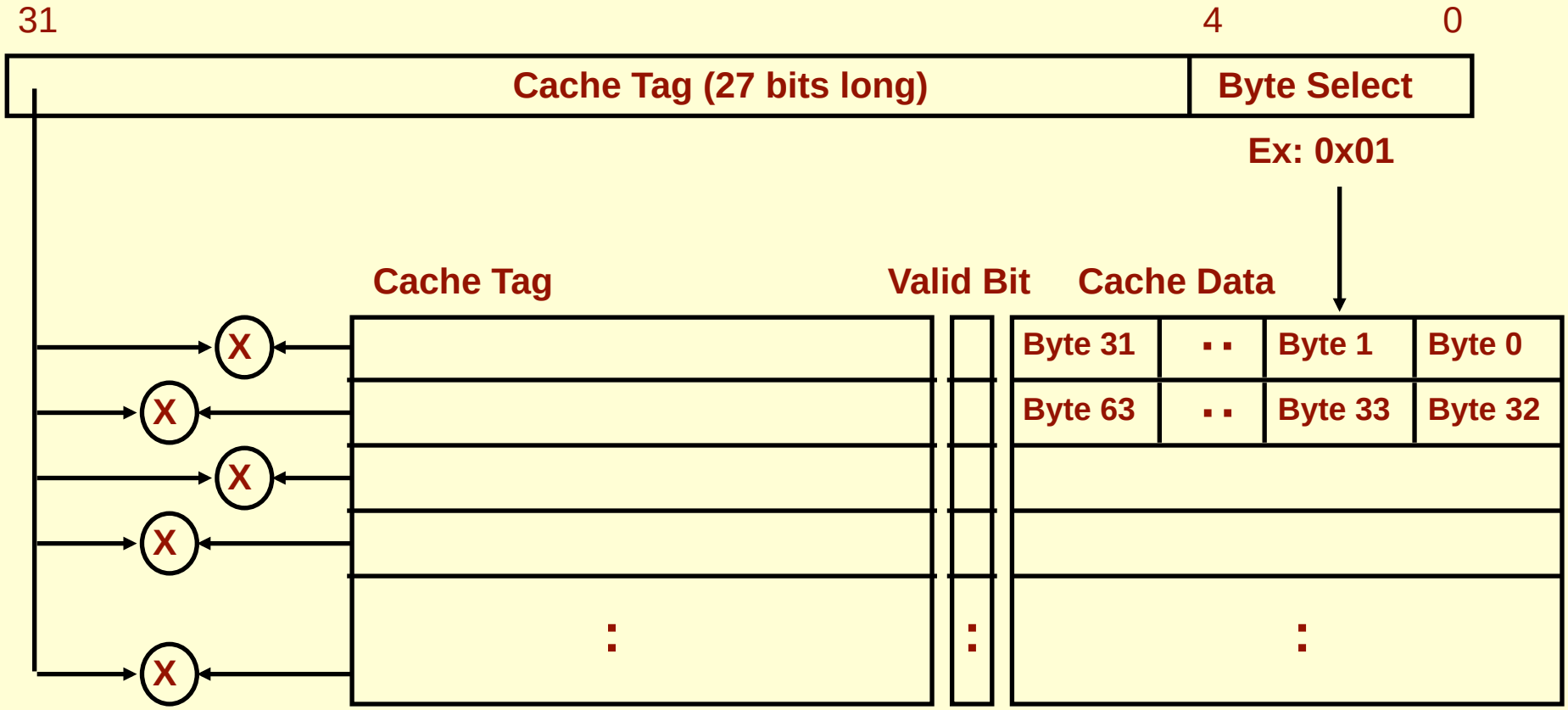
Locating a Block in Associative Cache



Tag size increases with higher level of associativity

Fully Associative Cache

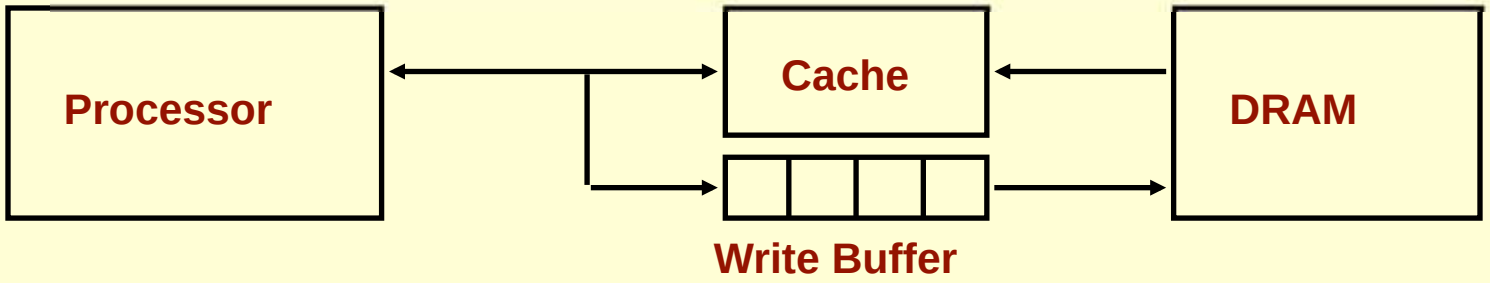
- Forget about the Cache Index
- Compare the Cache Tags of all cache entries in parallel
- Example: Block Size = 32 Byte blocks, we need N 27-bit comparators
- By definition: Conflict Miss = 0 for a fully associative cache



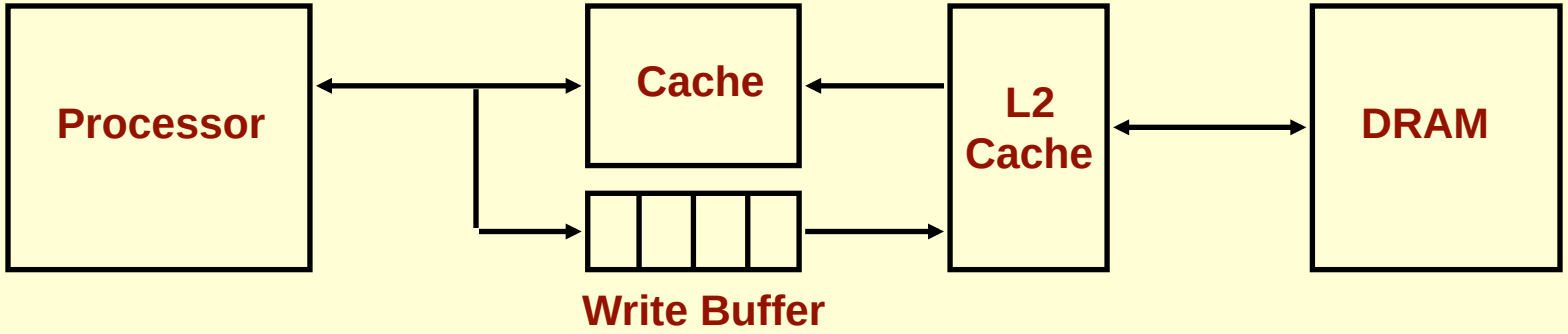
Handling Cache Misses

- Read misses bring blocks from memory
- Write access requires careful maintenance of consistency between cache and main memory
- Two write strategies:
  - Write through: write to both cache and memory
    - Read misses cannot result in writes
    - No allocation of a cache block is needed
    - Always combined with write buffers so that don't wait for slow memory
  - Write back: write cache only; write to memory when cache block is replaced
    - Is block clean or dirty?
    - No writes to slow memory for repeated write accesses
    - Requires allocation of a cache block

Write Through via Buffering



- Processor writes data into the cache and the write buffer
- Memory controller writes contents of the buffer to memory
- Increased write frequency can cause saturation of write buffer
- If CPU cycle time too fast and/or too many store instructions in a row:
  - Store buffer will overflow no matter how big you make it
  - The CPU Cycle Time get closer to DRAM Write Cycle Time
- Write buffer saturation can be handled by installing a second level (L2) cache



Block Replacement Strategy

- Straight forward for Direct Mapped since every block has only one location
- Set Associative or Fully Associative:
  - Random: pick any block
  - LRU (Least Recently Used)
    - requires tracking block reference
    - for two-way set associative cache, reference bit attached to every block
    - more complex hardware is needed for higher level of cache associativity

Associativity / Size	2-way		4-way		8-way	
	LRU	Random	LRU	Random	LRU	Random
16 KB	5.2%	5.7%	4.7%	5.3%	4.4%	5.0%
64 KB	1.9%	2.0%	1.5%	1.7%	1.4%	1.5%
256 KB	1.15%	1.17%	1.13%	1.13%	1.12%	1.12%

- Empirical results indicates less significance of replacement strategy with increased cache sizes