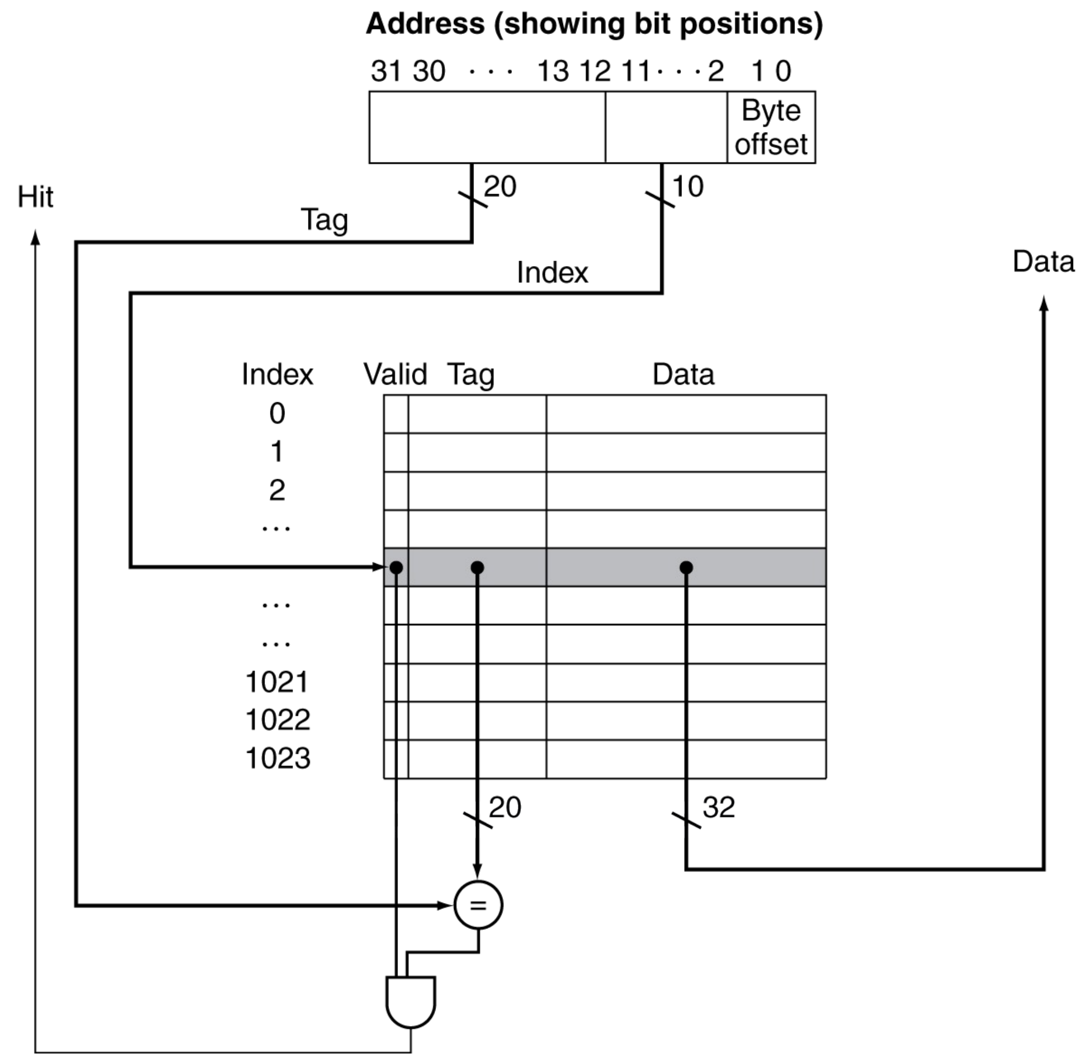
CENG311 Computer Architecture

Memory System

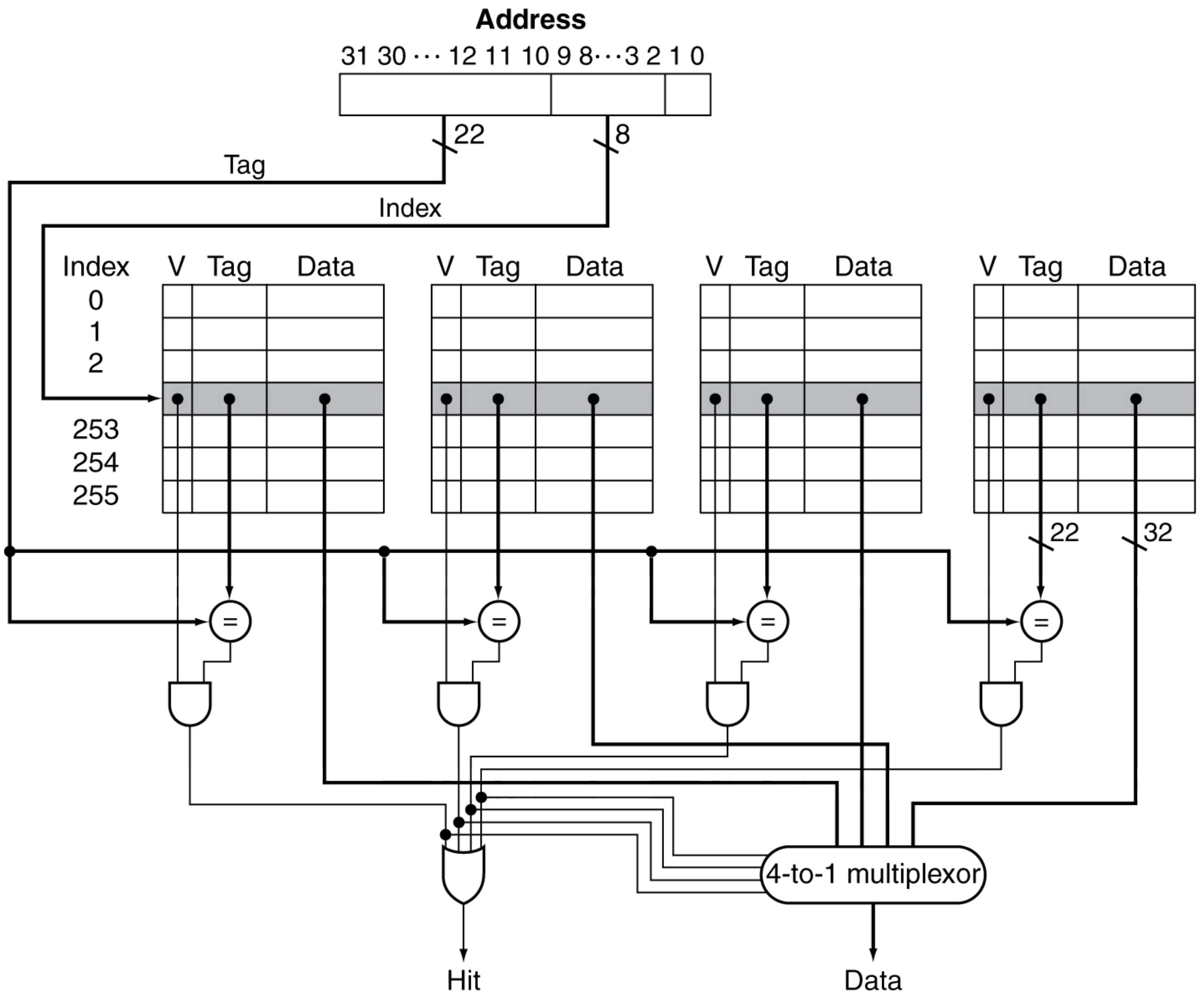
IZTECH, Fall 2023

28 December 2023

Direct-Mapped Cache - 4KB



4-way Set Associative Cache

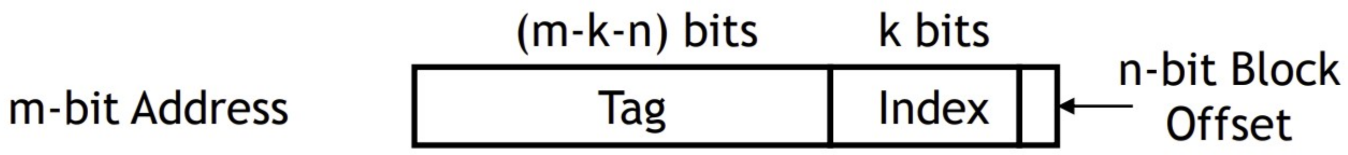


Cache Bits in Direct Mapped Cache

A cache with 2^k blocks, each containing 2^n bytes

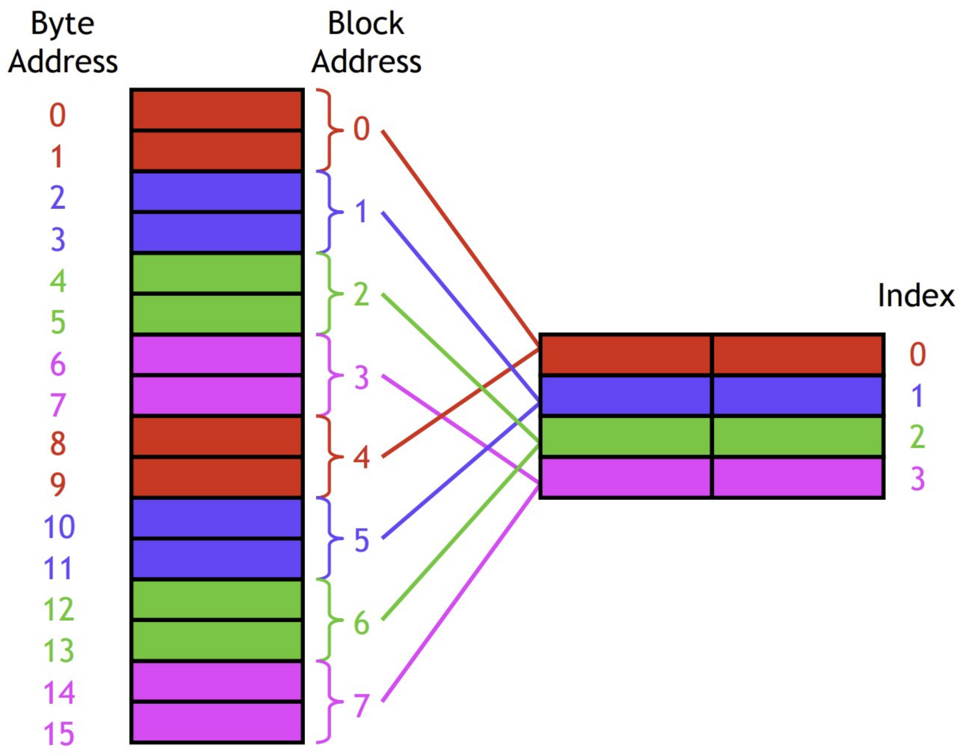
k bits of the address will select one of the 2^k cache blocks

The lowest n bits are now a block offset that decides which of the 2^n bytes in the cache block will store the data



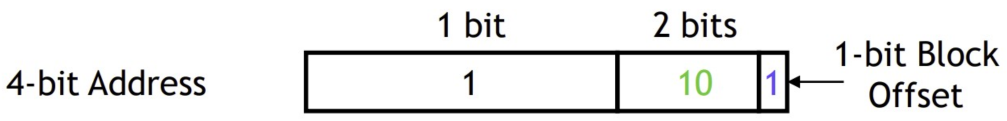
Example Direct-Mapped Cache

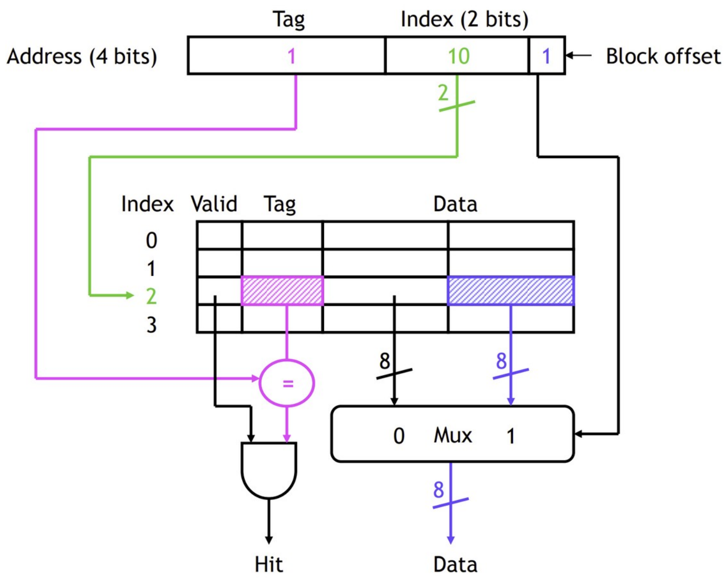
2^2 block cache with 2^1 bytes per block



Example Memory Address

Memory address 13 (1101)





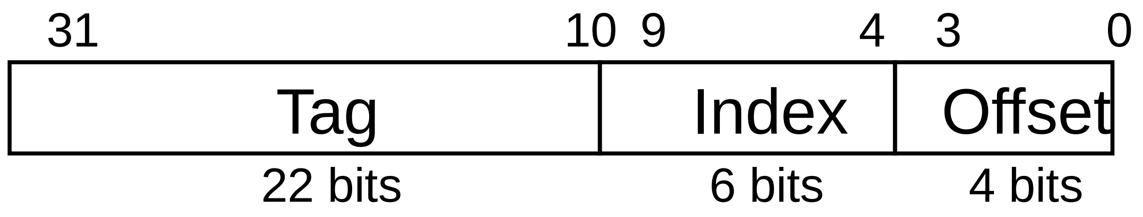
Multiword Cache Block

64 blocks, 16 bytes/block

To what block number does address 1200 map?

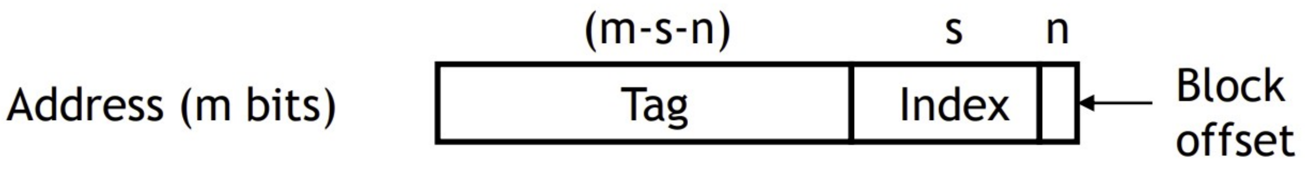
Block address = 1200/16 = 75 1 001011 0000

Block number = 75 modulo 64 = 11



Cache Bits in Associative Caches

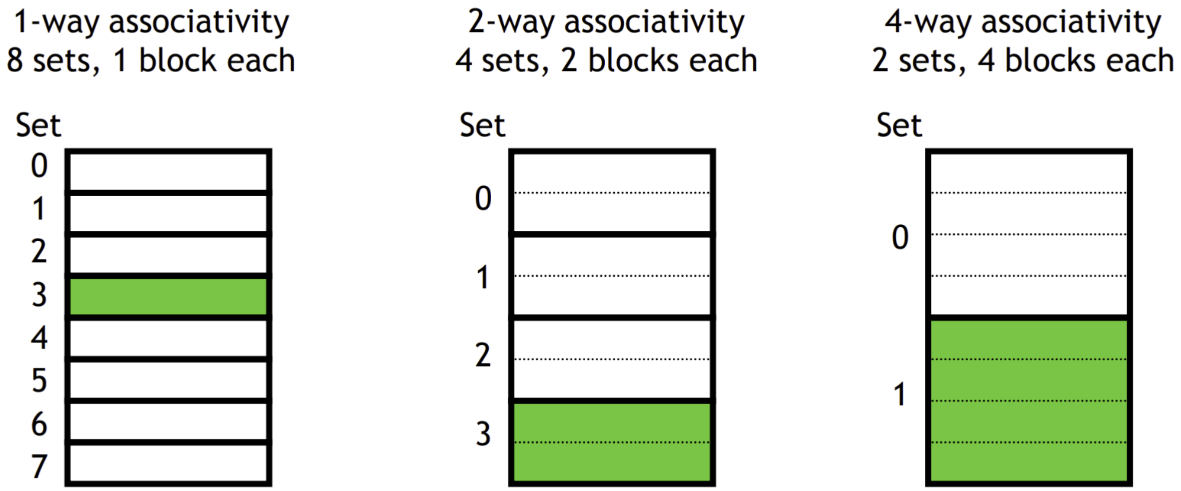
A cache with 2^s sets and each block has 2^n bytes



Example Set Associative Cache

8-block cache with 16 bytes per block

Memory address 6195: 00...0110000 <u>011</u> <u>0011</u>



Cache Operations

When CPU needs to read data

- checks the level 1 cache, then the level 2, so on
- if data is available: cache **hit**
- otherwise: cache **miss**, accesses the main memory

When CPU writes data to cache

- the line is written to main memory when it is written to the cache: **write-through**
- the data isn't written, marked as dirty, when the cache line is replaced by a new cache line from memory, the dirty line is written to memory: **write-back**

Cache Operations

When CPU needs to read data

- checks the level 1 cache, then the level 2, so on
- if data is available: cache **hit**
- otherwise: cache **miss**, accesses the main memory

When CPU writes data to cache

- the data isn't written, marked as dirty, when the cache line is replaced by a new cache line from memory, the dirty line is written to memory: **write-back**
- the line is written to main memory when it is written to the cache: **write-through**

When do we write the modified data in a cache to the next level?

Write-back **: When the block is evicted

+ Can combine multiple writes to the same block before eviction

Potentially saves bandwidth between cache levels + saves energy

- Need a bit in the tag store indicating the block is "dirty/modified"

Write-through: At the time the write happens

+ Simpler

+ All levels are up to date (Consistency)

- More bandwidth intensive; no combining of writes

Eviction/Replacement Policy

Which block in the set to replace on a cache miss?

- Any invalid block first
- If all are valid: Random, FIFO, Least recently used, Not most recently used, Least frequently used, Least costly to re-fetch, Hybrid replacement policies
- Optimal replacement policy?

LRU

Evict the least recently accessed block

Need to keep track of access ordering of blocks

Instead of having true LRU, there are LRU approximations in modern processors

- Not MRU (not most recently used)

LRU vs Random

- Average hit rate of LRU and Random is similar

Manual vs Automatic Management

Manual: Programmer manages data movement across levels

- too painful for programmers on substantial programs

In some embedded processors (on-chip scratchpad SRAM) and GPUs (called "shared memory")

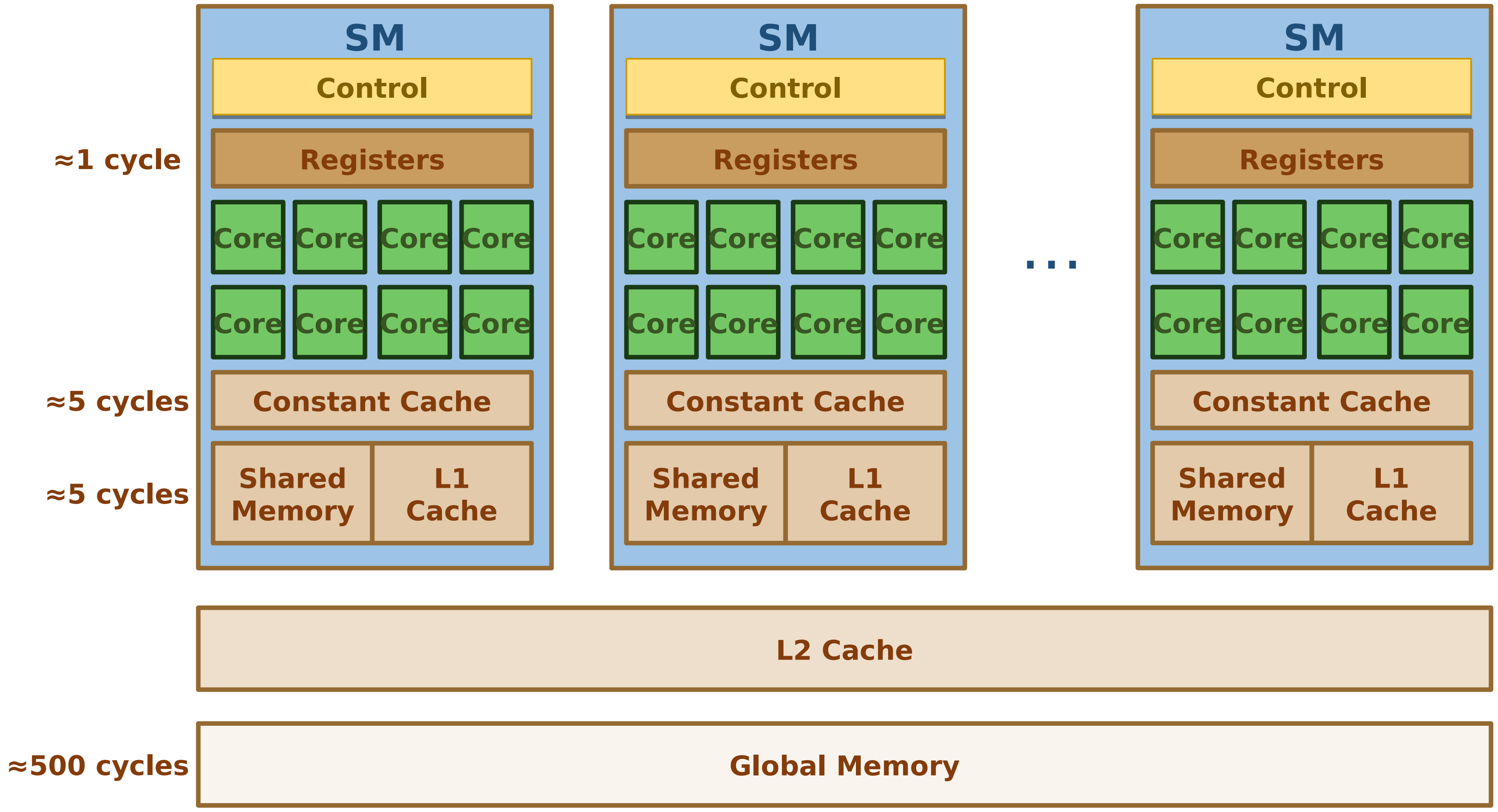
Automatic: Hardware manages data movement across levels, transparently to the programmer

+ programmer's life is easier

Average programmer doesn't need to know about it

You don't need to know how big the cache is and how it works to write a "correct" program! (What if you want a "fast" program?)

GPU Memory Structure



Instruction vs Data Caches

Unified cache

+ Dynamic sharing of cache space: no overprovisioning that might happen with static partitioning (i.e., separate I and D caches)

- Instructions and data can thrash each other (i.e., no guaranteed space for either)

- I and D are accessed in different places in the pipeline. Where do we place the unified cache for fast access?

First level caches are almost always split

Higher level caches are almost always unified

Cache Misses

Compulsory miss

First reference to an address (block) always results in a miss

Subsequent references should hit unless the cache block is replaced

Capacity miss

Cache is too small to hold everything needed

Defined as the misses that would occur even in a fully-associative cache (with optimal replacement) of the same capacity

Conflict miss

Defined as any miss that is neither a compulsory nor a capacity miss

How to Reduce Misses

Compulsory

Caching cannot help

Prefetching can: Anticipate which blocks will be needed soon

Capacity

Utilize cache space better: keep blocks that will be referenced

Software management: divide working set and computation such that each "computation phase" fits in cache

Conflict

More associativity

How to Improve Cache Performance

Reducing miss rate

- More associativity
- Better replacement/insertion policies

Reducing miss latency or miss cost

- Multi-level caches
- Better replacement/insertion policies
- Multiple accesses per cycle

Reducing hit latency or hit cost

References

Chapter 5.3 - 5.4

(Computer Organization and Design: The Hardware/Software Interface by Hennessy/Patterson, 5th edition)