NTNU

Norwegian University of Science and Technology

Memory Hierarchy: Lecture 6

Basics

Who am I?

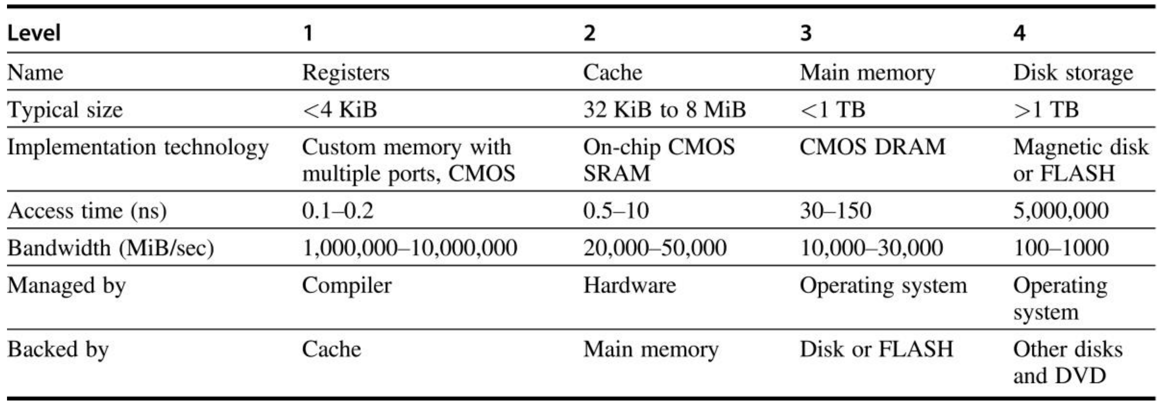
- Ole Henrik Jahren (ole.jahren@arm.com)
- Work for Arm (http://www.arm.com)
- Part of Mali GPU team
- Design and architecture of memory system
  - Multilevel coherent caches
  - Virtual memory MMU/TLB
  - On chip network

Agenda

- Memory hierarchy
  - Motivation
  - What makes memory hierarchy work
- Caches
  - Motivation
  - Organization
  - Design

Memory Hierarchy – Why

- Want large amount of fast memory
- Fast memory is physically big and financially expensive
- Large memory is slow, but financially cheaper



Level	1	2	3	4
Name	Registers	Cache	Main memory	Disk storage
Typical size	<4 KiB	32 KiB to 8 MiB	<1 TB	>1 TB
Implementation technology	Custom memory with multiple ports, CMOS	On-chip CMOS SRAM	CMOS DRAM	Magnetic disk or FLASH
Access time (ns)	0.1–0.2	0.5–10	30–150	5,000,000
Bandwidth (MiB/sec)	1,000,000–10,000,000	20,000–50,000	10,000–30,000	100–1000
Managed by	Compiler	Hardware	Operating system	Operating system
Backed by	Cache	Main memory	Disk or FLASH	Other disks and DVD

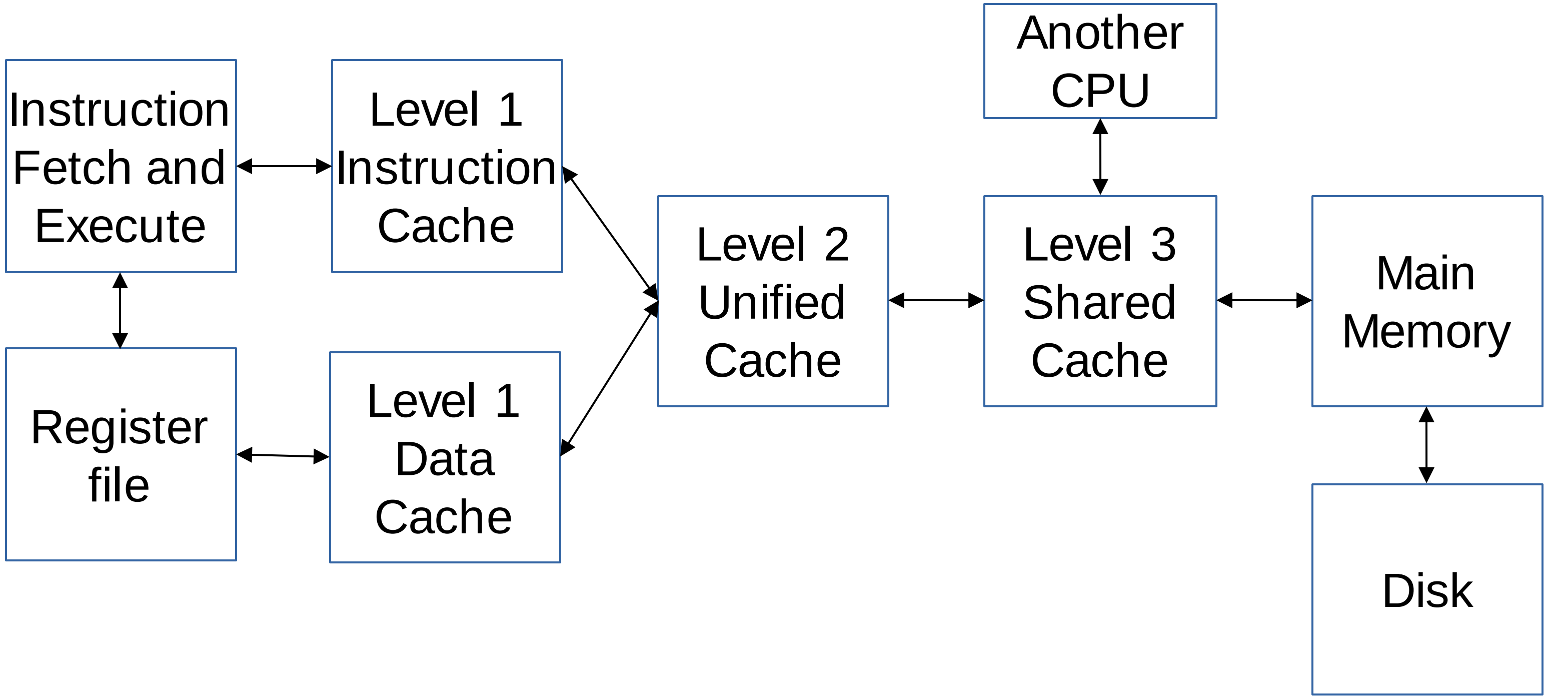
Memory Hierarchy – What

- Memory hierarchy is a level based approach
  - Levels closer to CPU provide faster, but smaller memories
  - Levels further from CPU provide slower, but larger memories
- Several of the levels can be caches

Memory Hierarchy – Locality

- Relies upon principle of locality
  - Temporal locality means recently used bytes are stored closer to CPU
  - Spatial locality means neighbouring bytes of recently used ones are stored closer to CPU

Memory Hierarchy – Anatomy



Cache – Why

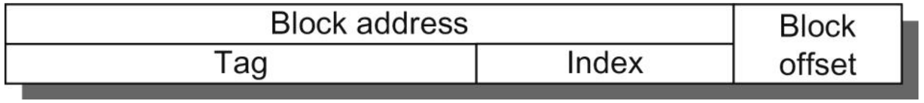
- Caches build upon the principle of locality
- Temporal locality
  - Operating on same data repeatedly (variables)
  - Executing same instruction repeatedly (loops)
- Spatial locality
  - Operating on neighbouring data (arrays)
  - Executing neighbouring instructions (sequential program)

Cache – What

- A cache consists of a number equally sized cache locations
- Each location holds a block size amount worth of data
- 3 common ways of organizing the locations inside a cache
  - Direct mapped
  - Set associative
  - Fully associative

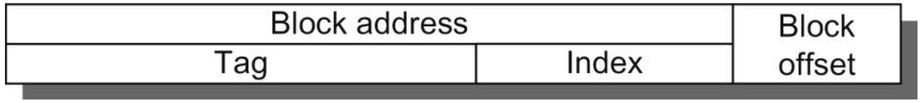
Cache – Block

- The memory address is made up on the block address and the block offset
  - Block address specifies which block in main memory
  - Block offset specifies which bytes inside the block we are referencing
- A cache block is the native unit the cache operates on



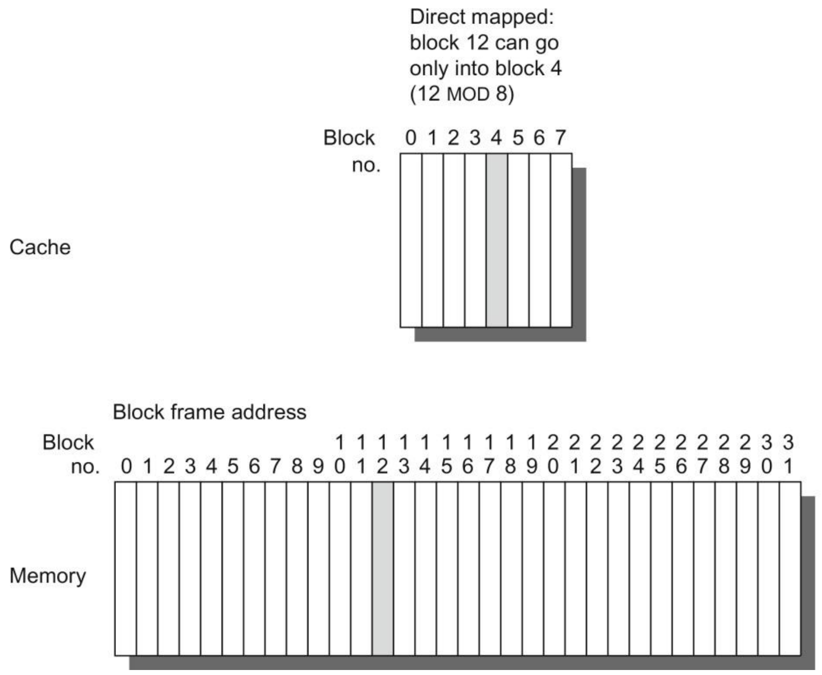
Cache – Direct Mapped

- Each block address is mapped to exactly one cache location
  - index = block_addr mod block_count
- When looking something up in the cache, we only know the index. To be able to reconstruct the full address, we need to store the tag part of the address with the data block.



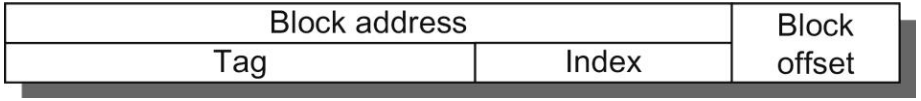
Cache – Direct Mapped

- Each block address in memory, map only to a single location in the cache
  - index = 12 mod 8 = 4
  - tag = floor(12 / 8) = 1
- Index and tag for block address 27?



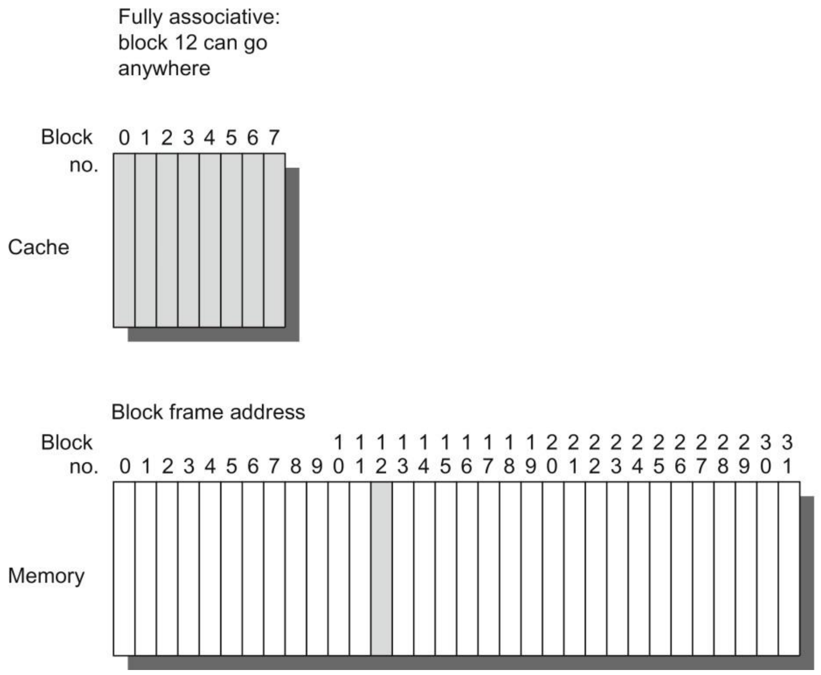
Cache – Fully Associative

- Each block address is mapped to exactly block_count cache locations
  - No index in fully associative caches, as each cache location must be checked
  - Full block address must be stored as tag
- No location conflicts possible, as any block can be in any location



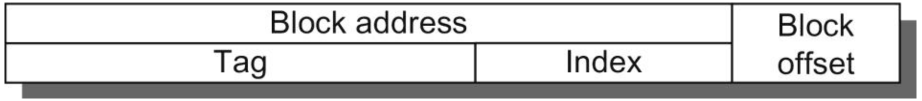
Cache – Fully Associative

- Each block address in memory, map to any location in the cache
  - No index
  - tag = 12
- Index and tag for block address 27?



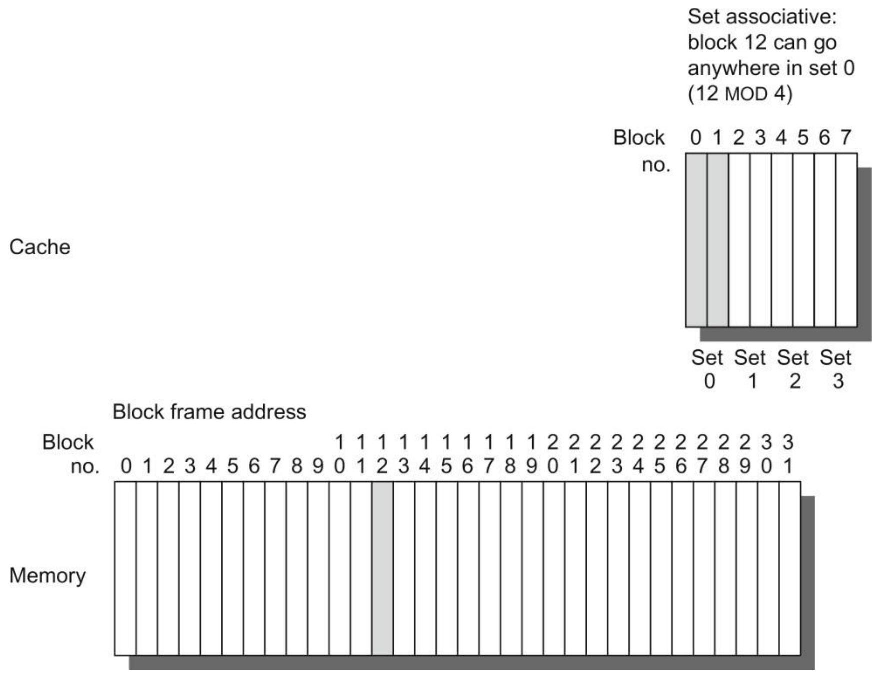
Cache – n-Way Set Associative

- Each block address is mapped to exactly *n* cache locations
  - set_count = block_count / n
  - index = block_addr mod set_count
- Less prone to location conflicts compared to direct mapped caches



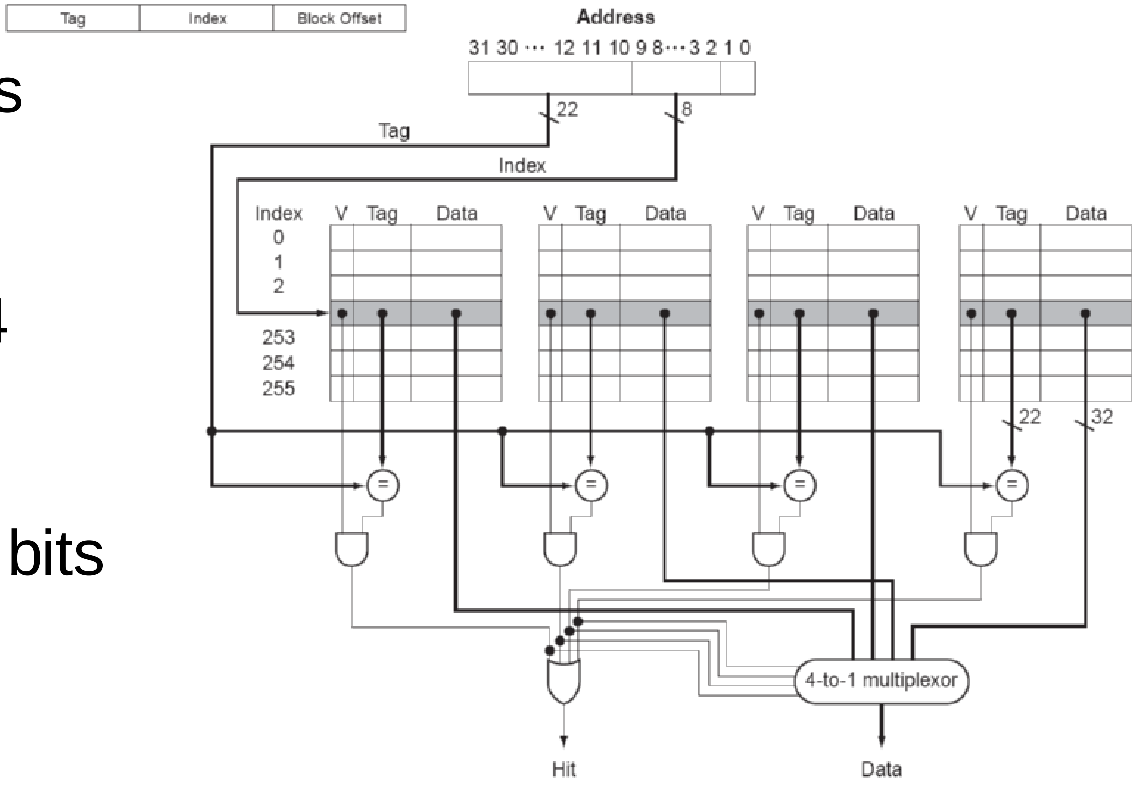
Cache – n-Way Set Associative

- Each block address in memory, map only to n locations in the cache
  - index = 12 mod 4 = 0
  - tag = floor(12 / 4) = 3
- Index and tag for block address 27?



Cache – 4-Way Set Associative

- Block size: 4 bytes
- Set count: 256
- Way count: 4
- Block count: 1024
- Cache size: 4KiB
- Tag size: 22 bits
- Bits per block: 45 bits



Cache – Associativity Summary

- Direct Mapped and Fully Associative are special cases of Set Associative!

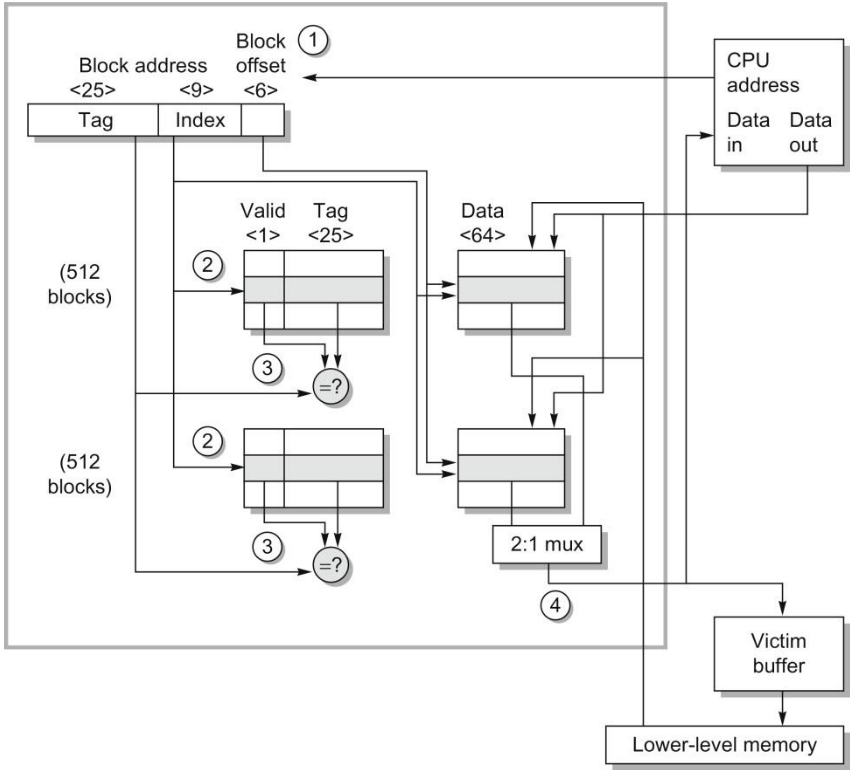
	Direct Mapped	Fully Associative	n-Way Set Associative
Block size	b	b	b
Cache size	s	s	s
Way count	1	s / b	n
Set count	s / b	1	s / (b * n)
Block count	s / b	s / b	s / b

Cache – Misses

- Each cache location contains valid bit, tag and data
- When a lookup in the cache can not find a valid tag match, it counts as a cache miss
- A cache miss can be caused by any of the 3 C’s
  - Compulsory
  - Capacity
  - Conflict

Cache – Miss Example

- Operation from CPU
- Look up tags
- If valid and tag matches, way hit
- If no way hit, cache miss



Cache – Miss Reasons

- Compulsory misses are the misses that still would occur even if the cache was infinitely large
- Capacity misses are the misses that still would occur even if the cache was fully associative
- Conflict misses are the misses that occur because the cache is not fully associative
- Reducing miss rate means reducing how often one of the reasons leads to a miss

Cache – Design Choices

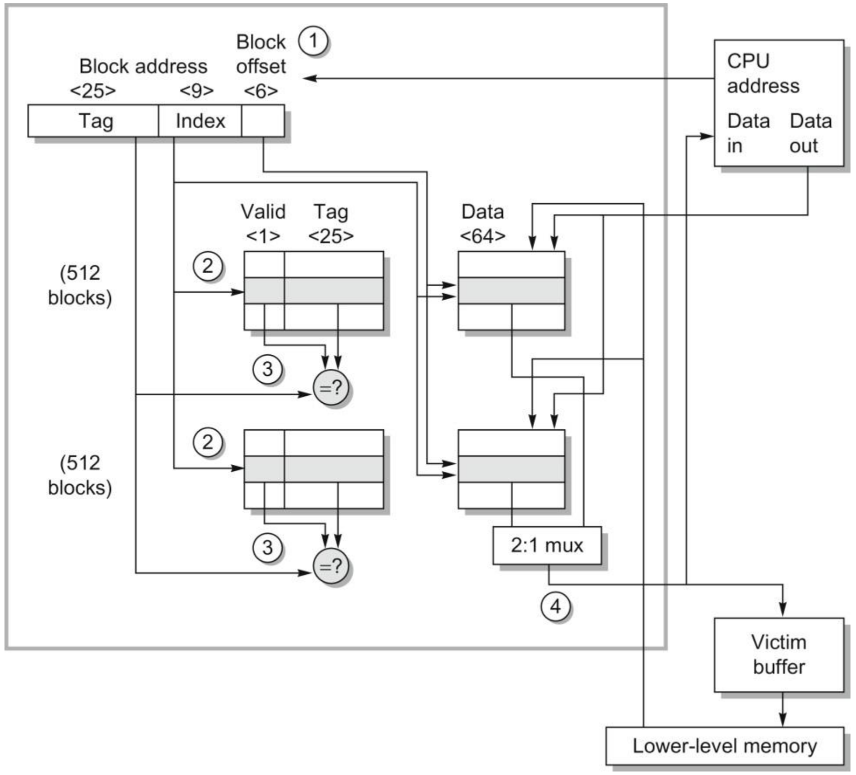
- The design space for caches is huge
- Will go through some of the common choices face by cache designers

Cache – Write Handling

- Write-back
  - Write will only write to cache location
- Write-through
  - Write will write to cache location and to next level of memory hierarchy

Cache – Write Example

- Write-back will write to cache location data only
- Write-through will write to cache location and next-level in memory hierarchy

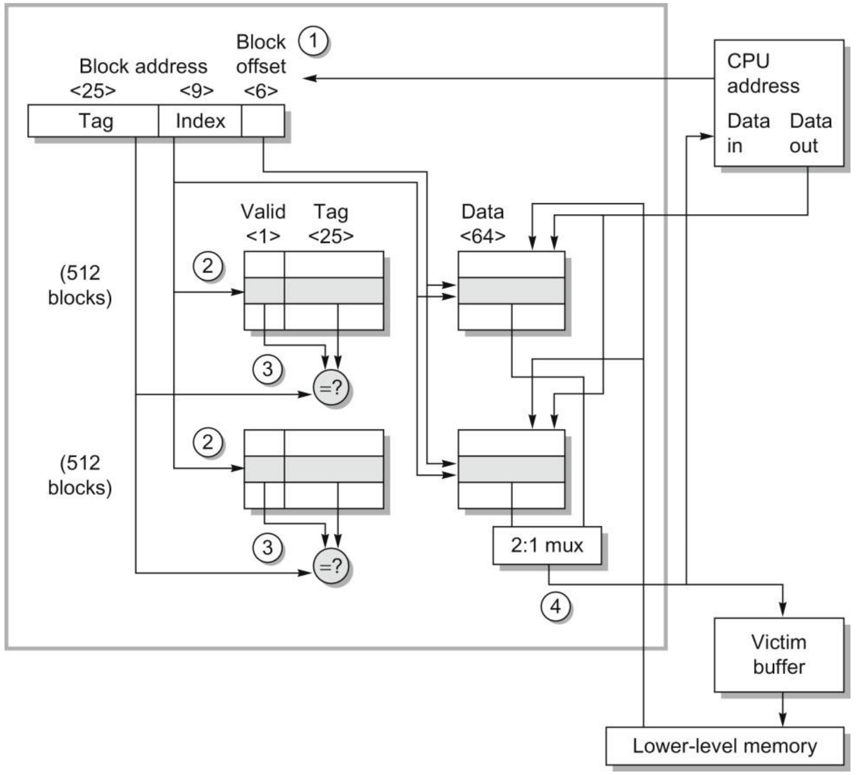


Cache – Allocation Policy

- Write allocate
  - Writes always allocate in cache
  - Writes will write to cache location regardless if hit or miss
- No-write allocate
  - Writes never allocate in cache
  - Writes will write to cache location if hit
  - Writes will write to next level in memory hierarchy if miss

Cache – Allocation Example

- Write allocate always writes to cache location data
- No-write allocation will only write to cache location data if hit, else write is done to next level in memory hierarchy



Cache – Write-back Allocation

- Write-back write allocate
  - Writes will write to cache location regardless if hit or miss
- Write-back no-write allocate
  - Writes will write to cache location if hit, to next level of memory hierarchy if miss

Cache – Write-through Allocation

- Write-through write allocate
  - Write will write to cache location and next level of memory hierarchy regardless if hit or miss
- Write-through no-write allocate
  - Writes will write to cache location and next level of memory hierarchy if hit, to next level of memory hierarchy if miss

Cache – Block size

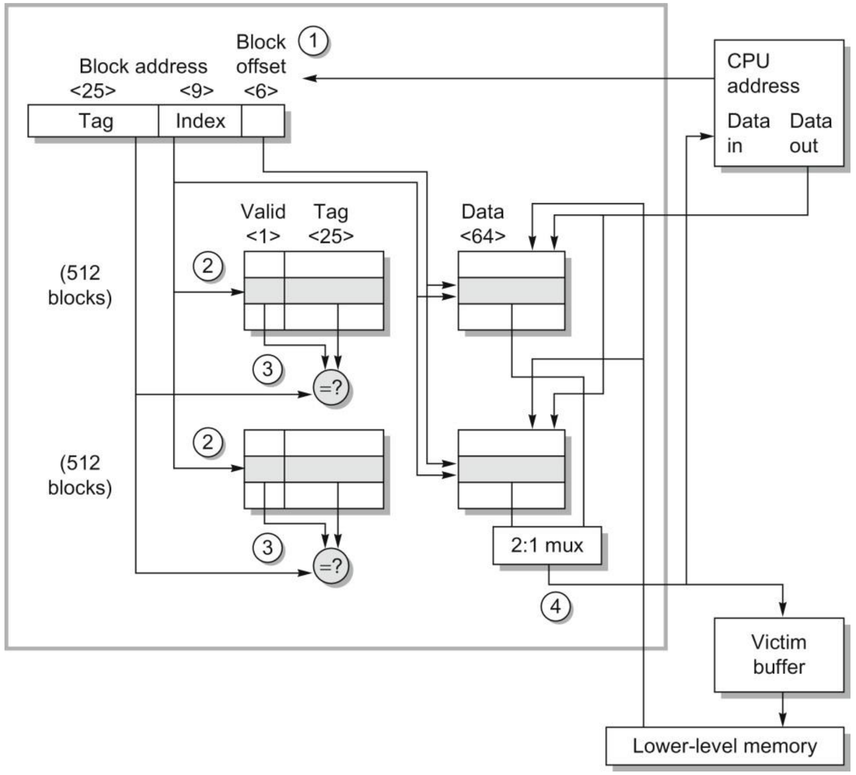
- Large block gives greater opportunity for spatial locality
- Too large blocks means large portion of block is unreferenced before evicted
- Large blocks take longer time to read block into cache, which means increased miss penalty

Cache – Replacement Policy

- When attempting to allocate a location for a block that is not currently in the cache, but all possible locations for the block address are occupied
  - Need to evict one of the blocks occupying the candidate locations
- How which location to evict is chosen is known as replacement policy
- Well known replacement policies include random, FIFO and LRU

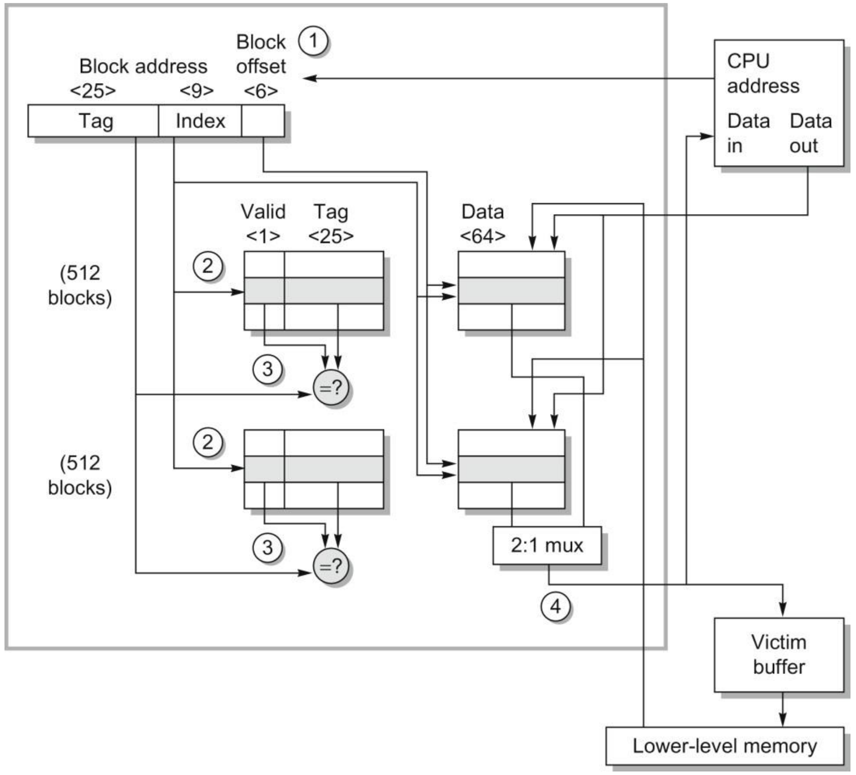
Cache – Replacement Example

- Miss in cache so need to allocate
- All ways are occupied so we need to evict something
- Replacement policy dictates which way is replaced (random, FIFO, LRU)



Cache – Replacement Example

- Random means we chose randomly which way to replace
- FIFO means we will replace the longest since allocated way
- LRU means we will replace the longest since last accessed way



Cache – Separate vs Unified

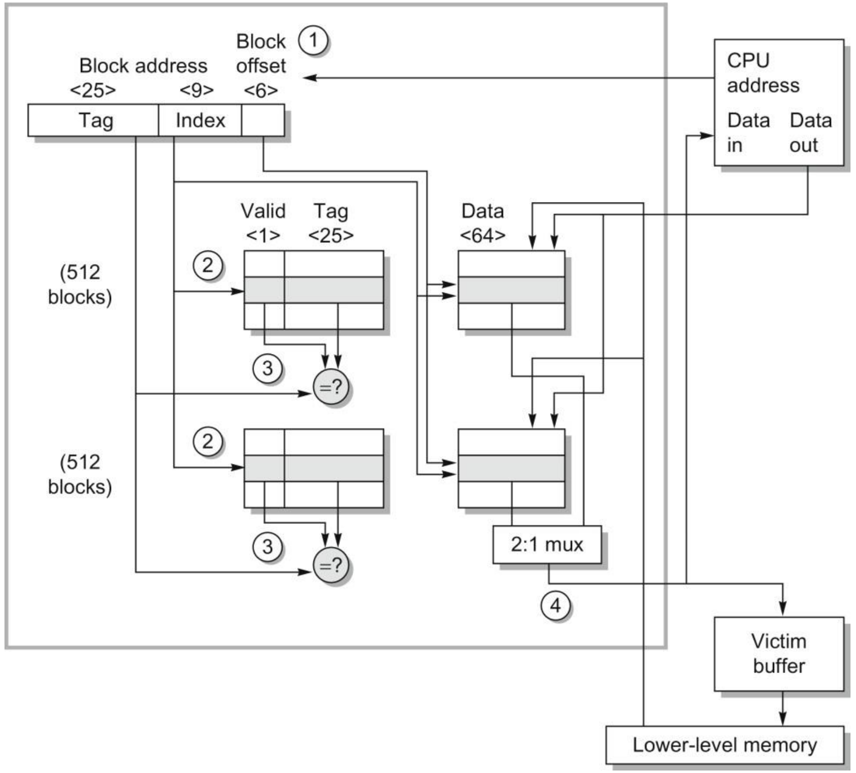
- Specialized caches are for separate specific function caches. E.g. Level 1 instruction cache.
  - Can exploit knowledge about what is stored in cache. E.g. instruction encoding.
- Only need functionality that is required for the specific function. E.g. instruction cache only needs reads
- Generic caches are for unified caches where very little can be assumed about blocks stored in cache. Need to be general. E.g. Unified Level 2 cache.

Cache – Write-buffer vs stalling

- When writing back evicted cache lines, or performing writes in a write-through cache, a write to the next level in the memory hierarchy is generated
- A common optimization is to avoid waiting for the write-back/through to complete by having a write-buffer
- This allows program execution to continue without stalling while waiting for completion of the write-back/through.

Cache – Write-buffer Example

- Miss in cache so need to allocate
- All ways are occupied so we need to evict something
- All ways are dirty so need to write something back
- Put write-back in write-buffer to avoid stalling



Cache – Inclusive vs Exclusive

- Caches being inclusive
  - If any block present in cache level n is also present in cache level n+1, cache level n+1 is said to be inclusive of cache level n
  - Inclusive caches "wastes" space, since same data is stored in multiple levels of cache

Cache – Inclusive vs Exclusive

- Caches being exclusive
  - If any block present in cache level n is never present in cache level n+1, cache level n+1 is said to be exclusive of cache level n
  - Exclusive caches does not waste space in the same manner inclusive caches do, but at the cost that caches on same level can not hit in next level cache because data is in neighbour

Cache – Inclusive vs Exclusive

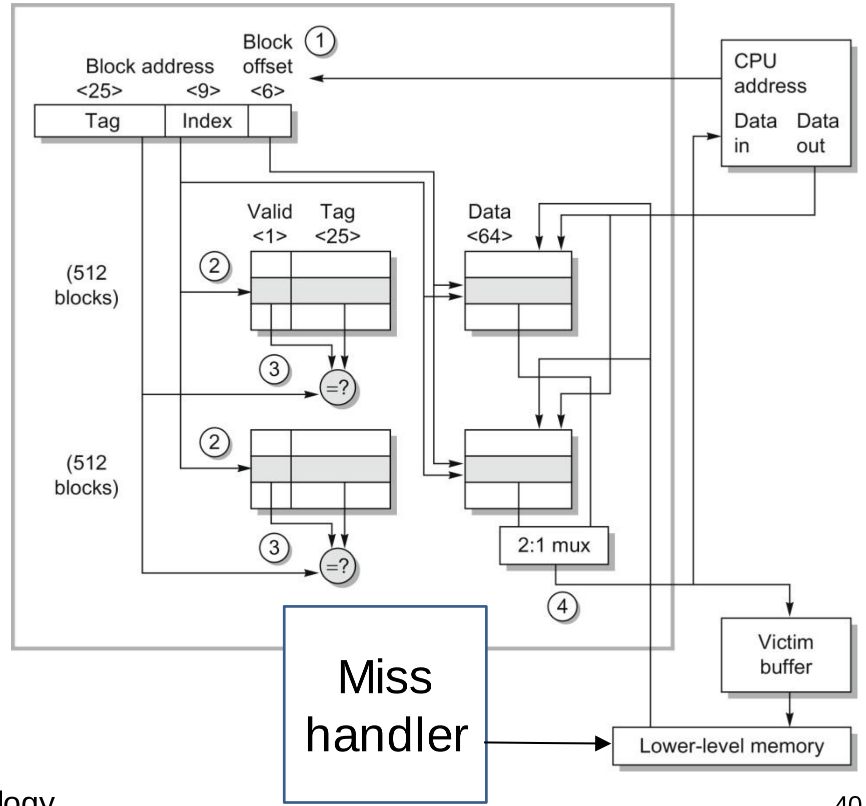
- Caches being non-inclusive non-exclusive
  - If neither the inclusive nor the exclusive property is satisfied

Cache – Miss Handling

- A cache that allows hitting on a cache block for address A while it handles a miss for address B, is said to support hit under miss.
- A cache that allows handling multiple misses concurrently, is said to support miss under miss.

Cache – Miss Handling Example

- Miss in cache so need to allocate
- Start reading block from next level of memory
- Put original operation in miss handler, while waiting for next level of memory
- Multi-slot miss handler can handle miss under miss



Cache – Miss Reduction

- Compulsory
  - E.g. larger cache blocks
- Capacity
  - E.g. different replacement policy, more blocks in cache
- Conflict
  - E.g. higher associativity

Draw a block diagram of a set associative cache

5 Minute Assignment