

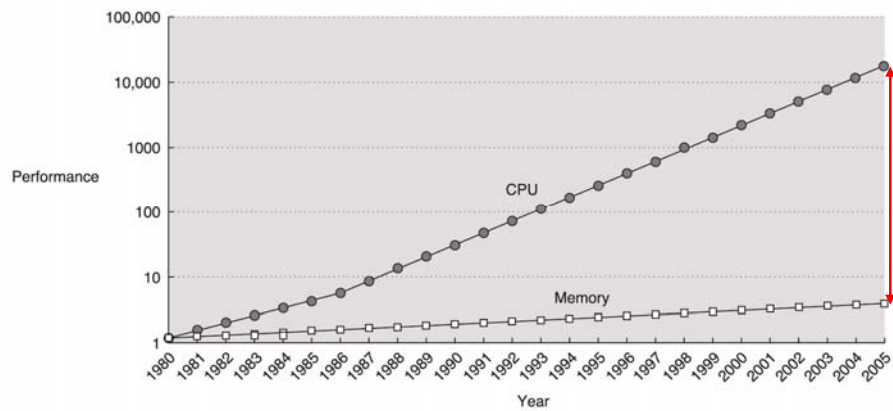
CS2410: Computer Architecture

L1 cache design

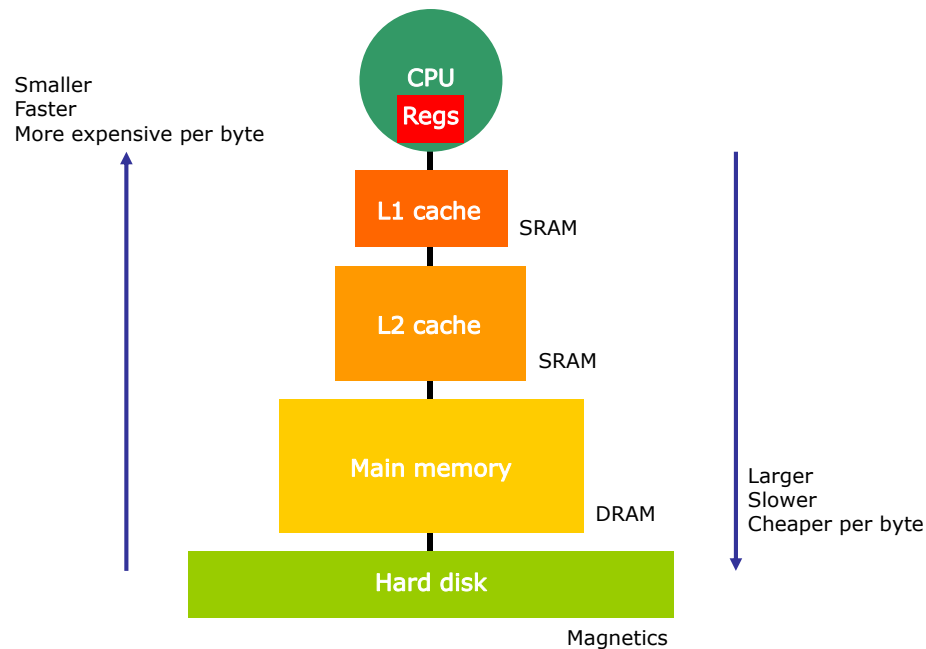
Sangyeun Cho

Computer Science Department
University of Pittsburgh

Why memory hierarchy?



Memory hierarchy



CS2410: Computer Architecture

University of Pittsburgh

Memory hierarchy goals

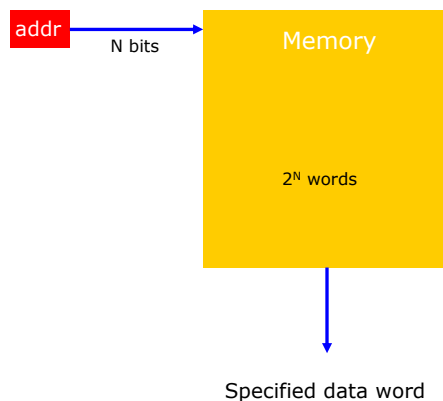
- To provide CPU with necessary data (and instructions) as quickly as possible
 - To achieve this goal, a cache should keep frequently used data
 - "Cache hit" when CPU finds a requested data in cache
 - Hit rate = # of cache hits/# of cache accesses
 - Average memory access latency (AMAL) = cache hit time + (1 – cache hit rate) × miss penalty
 - To decrease AMAL, reduce hit time, increase hit rate, and reduce miss penalty
- To reduce traffic on memory bus
 - Cache becomes a "filter"
 - Reduces the bandwidth requirements from the main memory
 - Typically, max. L1 bandwidth (to CPU) > max. L2 bandwidth (to L1) > max. memory bandwidth

CS2410: Computer Architecture

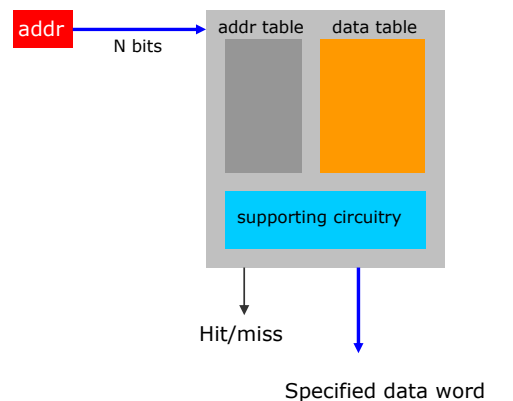
University of Pittsburgh

Cache organization

- Caches use “blocks” or “lines” (block > byte) as their granule of management
- Memory > cache: we can only keep a subset of memory blocks
- Cache is in essence a fixed-width hash table; the memory blocks kept in a cache are thus associated with their addresses (or “tagged”)



CS2410: Computer Architecture



University of Pittsburgh

L1 cache vs. L2 cache

- Their basic parameters are similar
 - Associativity, block size, and cache size (the capacity of data array)
- Address used to index
 - L1: typically virtual address (to quickly index first)
 - Using a virtual address causes some complexities
 - L2: typically physical address
 - Physical address is available by then
- System visibility
 - L1: not visible
 - L2: *page coloring* can affect hit rate
- Hardware organization (esp. in multicores)
 - L1: private
 - L2: often shared among cores

CS2410: Computer Architecture

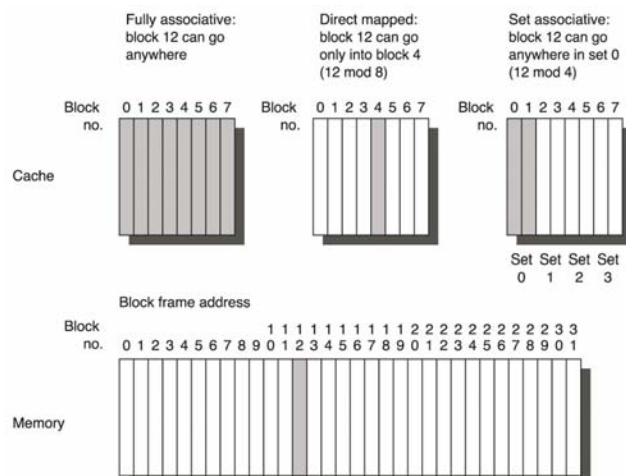
University of Pittsburgh

Key questions

- Where to place a block?
- How to find a block?
- Which block to replace for a new block?
- How to handle a write?
 - Writes make a cache design much more complex!

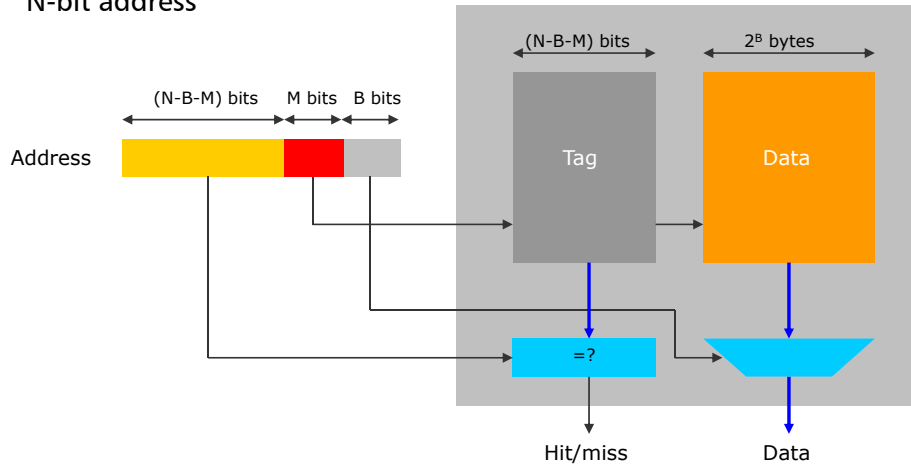
Where to place a block?

- Block placement is a matter of *mapping*
- If you have a simple rule to place data, you can find them later using the same rule



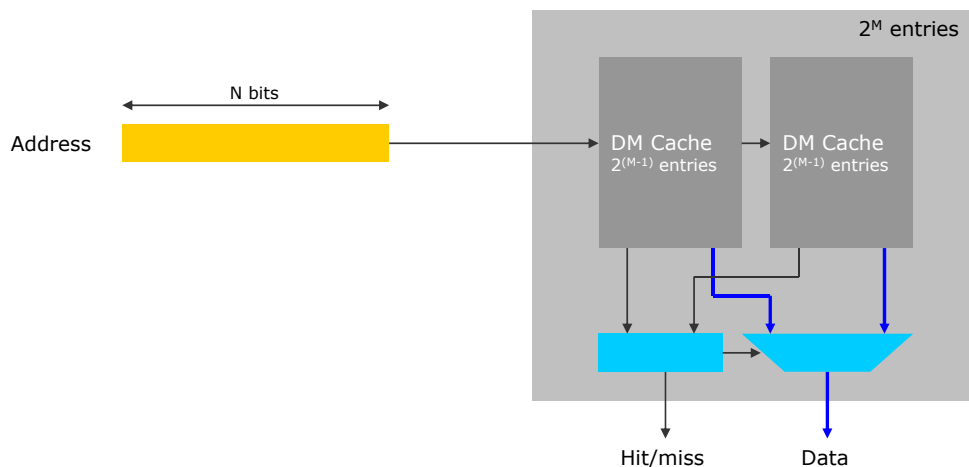
Direct-mapped cache

- 2^B byte block
- 2^M entries
- N-bit address



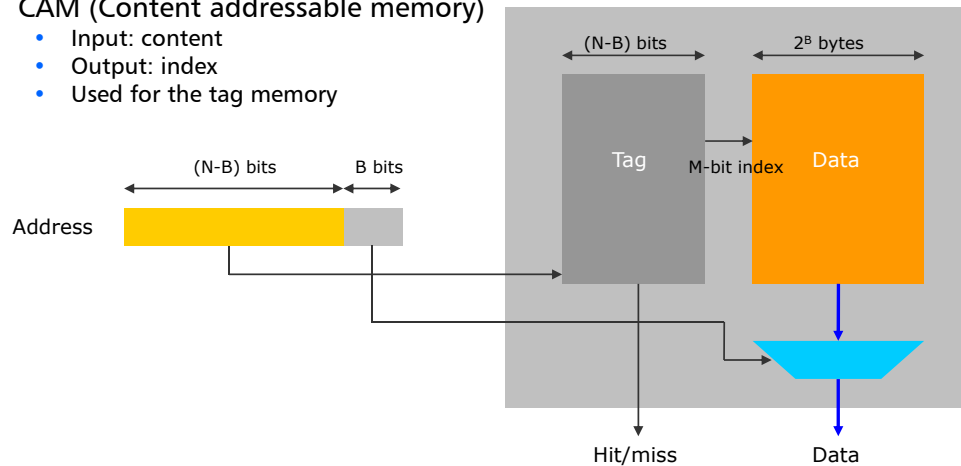
2-way set-associative cache

- 2^B byte block
- 2^M entries
- N-bit address



Fully associative cache

- 2^B byte block
- 2^M entries
- N-bit address
- CAM (Content addressable memory)
 - Input: content
 - Output: index
 - Used for the tag memory



CS2410: Computer Architecture

University of Pittsburgh

Why caches work (or do not work)

- Principle of locality
 - Temporal locality
 - If the location A is accessed now, it'll be accessed again soon
 - Spatial locality
 - If the location A is accessed now, the location nearby (e.g., A+1) will be accessed soon
- Can you explain how locality is manifested in your program (at the source code level)?
 - Data
 - Instructions
- Can you write the same program twice, having
 - A high degree of locality
 - Badly low locality

CS2410: Computer Architecture

University of Pittsburgh

Which block to replace?

- Which block to replace, to make room for a new block on a miss?
- Goal: minimize the # of total misses
- Trivial in a direct-mapped cache
- N choices in N-way associative cache
- What is the optimal policy?
 - MRU (most remotely used) is considered optimal
 - This is an oracle scheme – we do not know the future
- Replacement approaches
 - LRU (least recently used) – look at the past to predict the future
 - FIFO (first in first out) – honor the new ones
 - Random – don't remember anything
 - Cost-based – what is the cost (e.g., latency) of bringing this block again?

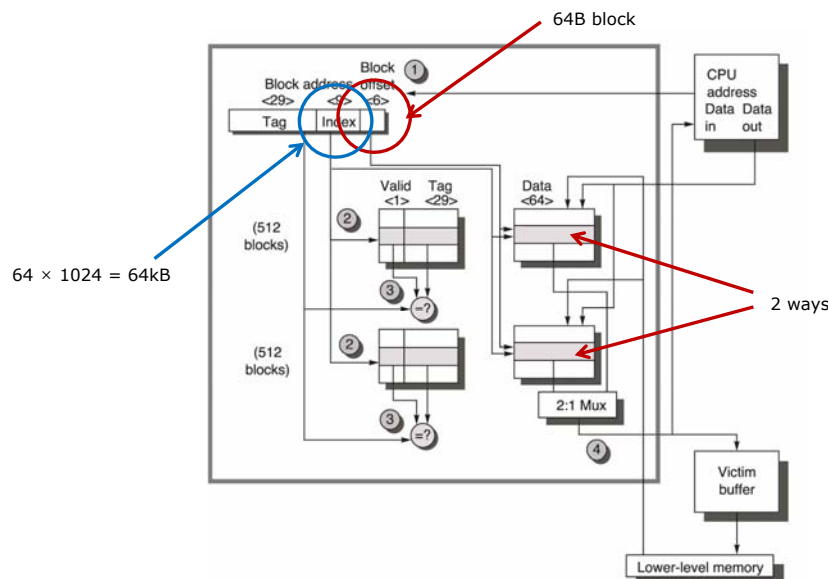
How to handle a write?

- Design considerations
 - Performance
 - Design complexity
- Allocation policy (on a miss)
 - Write-allocate
 - No-write-allocate
 - Write-validate
- Update policy
 - Write-through
 - Write-back
- Typical combinations
 - Write-back with write-allocate
 - Write-through with no-write-allocate

Write-through vs. write-back

- L1 cache: advantages of write-through + no-write-allocate
 - Simple control
 - No stalls for evicting dirty data on L1 miss with L2 hit
 - Avoids L1 cache pollution with results that are not read for a while
 - Avoids problems with coherence (L2 is consistent with L1)
 - Allows efficient transient error handling: parity protection in L1 and ECC in L2
 - What about high traffic between L1 and L2, esp. in a multicore processor?
- L2 cache: advantages of write-back + write-allocate
 - Typically reduces overall bus traffic by filtering all L1 write-through traffic
 - Better able to capture temporal locality of infrequently written memory locations
 - Provides a safety net for programs where write-allocate helps a lot
 - Garbage-collected heaps
 - Write-followed-by-read situations
 - Linking loaders (if unified cache, need not be flushed before execution)
- Some ISA/caches support explicitly installing cache blocks with empty contents or common values (e.g., zero)

Alpha 21264 example



More examples

- IBM Power5
 - L1I: 64kB 2-way 128B block LRU
 - L1D: 32kB 4-way 128B block write-through LRU
 - L2: 1.875MB (3 banks) 10-way 128B block pseudo LRU
- Intel Core Duo
 - L1I: 32kB 8-way 64B block LRU
 - L1D: 32kB 8-way 64B block LRU write-through
 - L2: 2MB 8-way 64B line LRU write-back
- Sun Niagara
 - L1I: 16kB 4-way 32B block random
 - L1D: 8kB 4-way 16B block random write-through write no-allocate
 - L2: 3MB 4 banks 64B block 12-way write-back

Impact of caches on performance

- Average memory access latency (AMAL) = cache hit time + $(1 - \text{cache hit rate}) \times \text{miss penalty}$
- Example 1
 - Hit time = 1 cycle
 - Miss penalty = 100 cycles
 - Miss rate = 2%
 - Average memory access latency?
- Example 2
 - 1GHz processor
 - Two configurations: 16kB direct-mapped, 16kB 2-way
 - Two miss rates: 3%, 2%
 - Hit time = 1 cycle, but clock cycle time is stretched by 1.1 in 2-way
 - Miss penalty = 100ns (how many cycles?)
 - Average memory access latency?