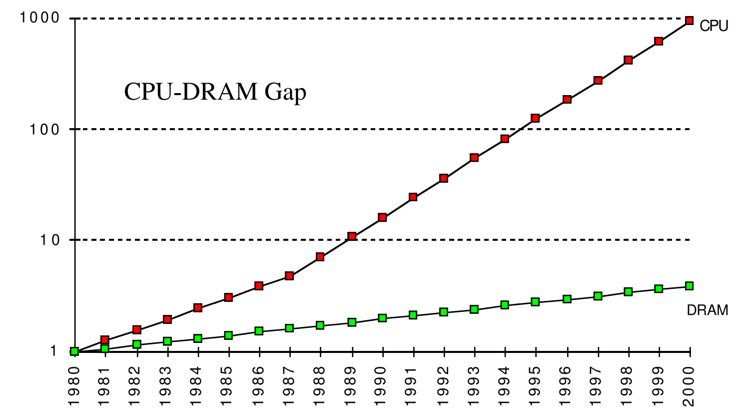
Memory Subsystem Design

or

Nothing Beats Cold, Hard Cache

Who Cares about Memory Hierarchy?

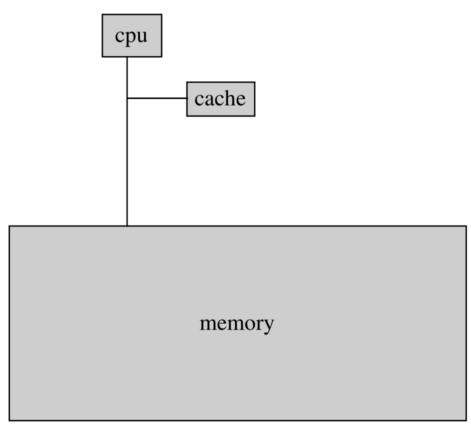
- Processor Only Thus Far in Course



1980: no cache in µproc;

1995 2-level cache, 60% trans. on Alpha 21164 µproc

Memory Cache



Memory Locality

- Memory hierarchies take advantage of *memory locality*.
- *Memory locality* is the principle that future memory accesses are *near* past accesses.
- Memories take advantage of two types of locality
  - ***Temporal locality*** -- near in time => we will often access the same data again very soon
  - ***Spatial locality*** -- near in space/distance => our next access is often very close to our last access (or recent accesses).

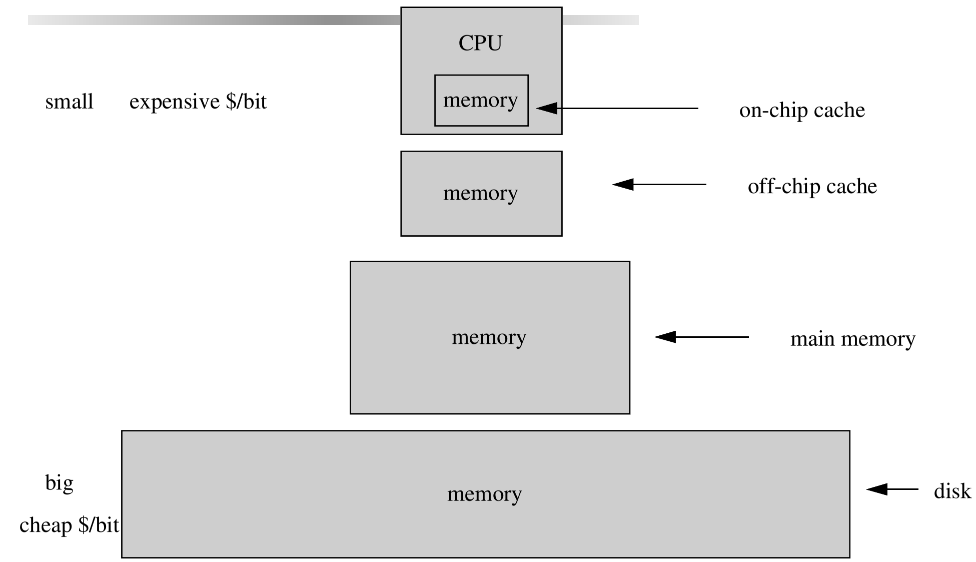
1,2,3,1,2,3,8,8,47,9,10,8,8...

Locality and cacheing

- Memory hierarchies exploit locality by *cacheing* (keeping close to the processor) data likely to be used again.
- This is done because we can build large, slow memories and small, fast memories, but we can't build large, fast memories.
- If it works, we get the illusion of SRAM access time with disk capacity

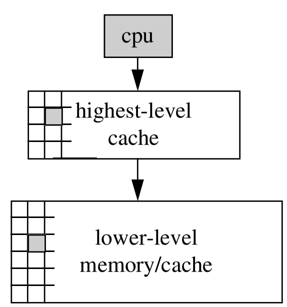
SRAM (static RAM) -- 5-20 ns access time, very expensive
DRAM (dynamic RAM) -- 60-100 ns, cheaper
disk -- access time measured in milliseconds, very cheap

A typical memory hierarchy



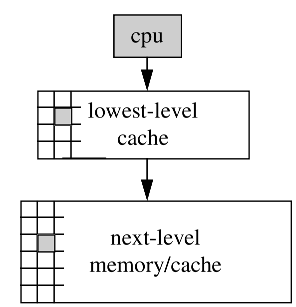
•so then where is my program and data??

Cache Fundamentals



- *cache hit* -- an access where the data is found in the cache.
- *cache miss* -- an access which isn't
- *hit time* -- time to access the higher cache
- *miss penalty* -- time to move data from lower level to upper, then to cpu
- *hit ratio* -- percentage of time the data is found in the higher cache
- *miss ratio* -- (1 - hit ratio)

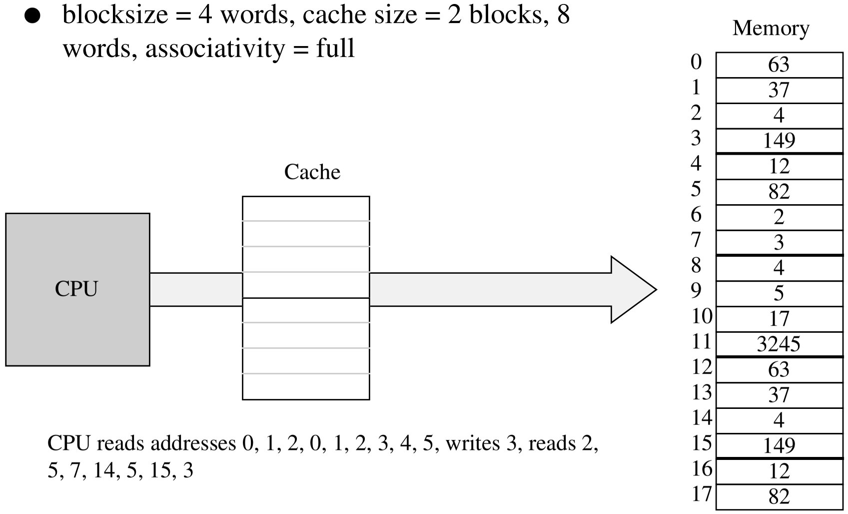
Cache Fundamentals, cont.



- *cache block size* or *cache line size*-- the amount of data that gets transferred on a cache miss.
- *instruction cache* -- cache that only holds instructions.
- *data cache* -- cache that only caches data.
- *unified cache* -- cache that holds both.

A Cache

- blocksize = 4 words, cache size = 2 blocks, 8 words, associativity = full



CPU reads addresses 0, 1, 2, 0, 1, 2, 3, 4, 5, writes 3, reads 2, 5, 7, 14, 5, 15, 3

Cache Characteristics

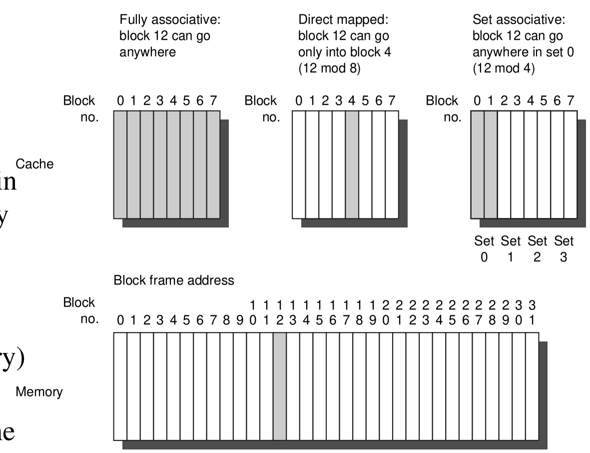
- Cache Organization
- Cache Access
- Cache Replacement
- Write Policy

Cache Organization: Where can a block be placed in the cache?

- Block 12 placed in 8-block cache:
  - Fully associative, direct mapped, n-way set associative
  - index = pointer to the set in the cache where a memory location might be cached

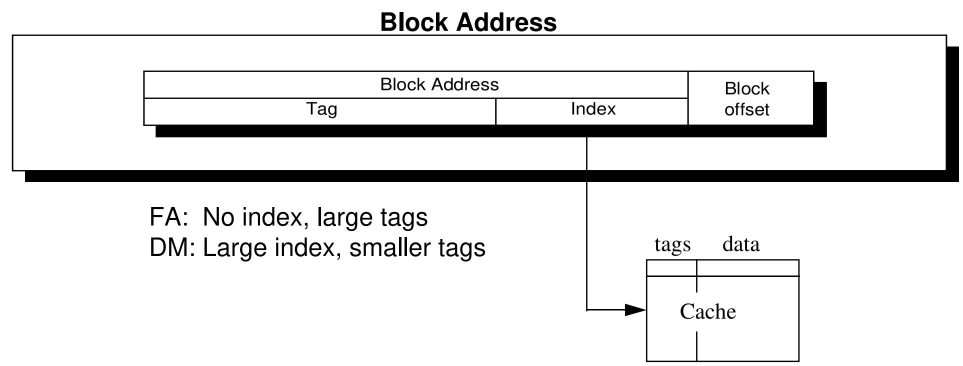
  (associativity = degree of freedom in placing a particular block of memory)

  (set = a collection of cache blocks with the same cache index)



Cache Access: How Is a Block Found In the Cache?

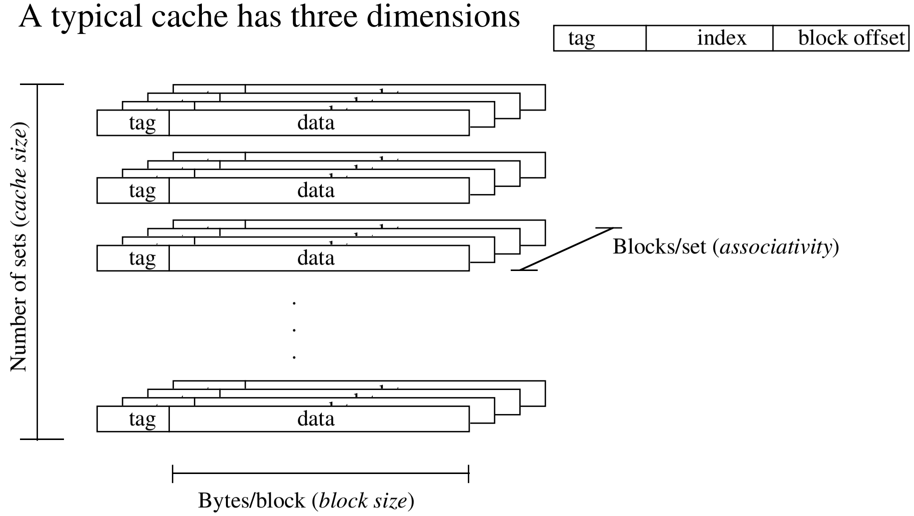
- Tag on each block
  - No need to check index or block offset
- Increasing associativity shrinks index, expands tag



FA: No index, large tags
DM: Large index, smaller tags

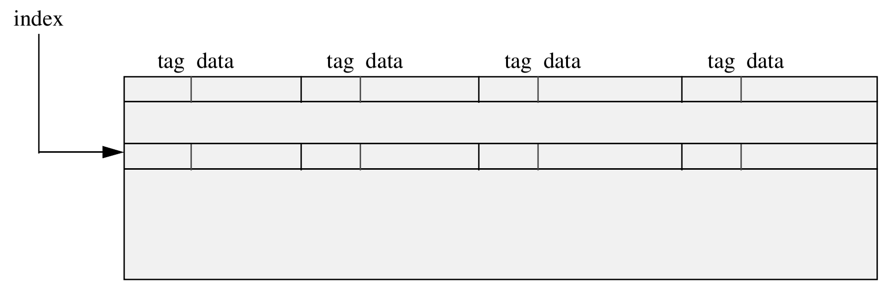
Cache Organization -- Overview

- A typical cache has three dimensions



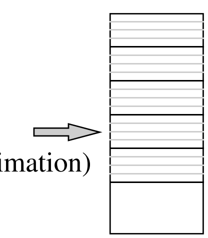
Cache Access

- 16 KB, 4-way set-associative cache, 32-bit address, byte-addressable memory, 32-byte cache blocks/lines
- how many tag bits?
- Where would you find the word at addres 0x200356A4?



Which Block Should be Replaced on a Miss?

- Direct Mapped is Easy
- Set associative or fully associative:
  - longest till next use (ideal, impossible)
  - least recently used (best practical approximation)
  - pseudo-LRU (e.g., NMRU, NRU)
  - random (easy)
  - how many bits for LRU?



Associativity:	2-way		4-way		8-way	
Size	LRU	Random	LRU	Random	LRU	Random
16 KB	5.18%	5.69%	4.67%	5.29%	4.39%	4.96%
64 KB	1.88%	2.01%	1.54%	1.66%	1.39%	1.53%
256 KB	1.15%	1.17%	1.13%	1.13%	1.12%	1.12%

What Happens on a Write?

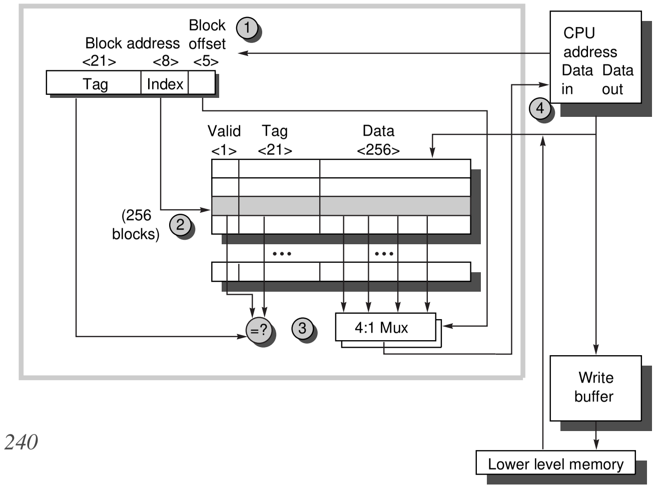
- *Write through*: The information is written to both the block in the cache and to the block in the lower-level memory.
- *Write back*: The information is written only to the block in the cache. The modified cache block is written to main memory only when it is replaced.
  - is block clean or dirty?
- Pros and Cons of each:
  - WT: read misses cannot result in writes (because of replacements)
  - WB: no writes of repeated writes
- WT always combined with *write buffers* so that don't wait for lower level memory

What happens on a write miss?

- *write-allocate* -- make room for the cache line in the cache, fetch rest of line from memory.
- *no-write-allocate* (write-around) -- write to lower levels of memory hierarchy, ignoring this cache.
- Tradeoffs?
- Which makes most sense for write-back?
- Which makes most sense for write-through?

21064 L1 Cache

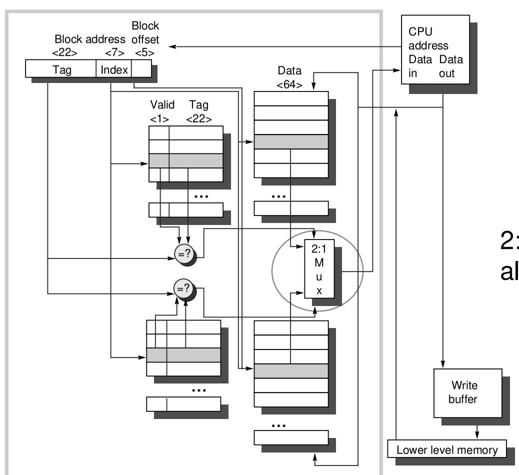
- 8 KB, 32-byte blocks, direct-mapped (SA = 1), ? blocks, ? sets
- write-through, write-around



2-way Set Associative Cache

Two sets of Address tags and data memory

Use address bits to select correct line



2:1 Mux along the way

Cache Organization: Separate Instruction and Data Caches?

Size	Instruction Cache	Data Cache	Unified Cache
1 KB	3.06%	24.61%	13.34%
2 KB	2.26%	20.57%	9.78%
4 KB	1.78%	15.94%	7.24%
8 KB	1.10%	10.19%	4.57%
16 KB	0.64%	6.47%	2.87%
32 KB	0.39%	4.82%	1.99%
64 KB	0.15%	3.77%	1.35%
128 KB	0.02%	2.88%	0.95%

if 75% of accesses are instructions?

Other reasons to separate?

Cache Performance

- CPU time = (CPU execution clock cycles + Memory stall clock cycles) x clock cycle time
- Memory stall clock cycles = (Reads x Read miss rate x Read miss penalty + Writes x Write miss rate x Write miss penalty)
- Memory stall clock cycles = Memory accesses x Miss rate x Miss penalty

Cache Performance

CPUtime = IC x ($CPI_{execution}$ + Memory stalls per instruction) x Clock cycle time

CPUtime = IC x ($CPI_{execution}$ + Mem accesses per instruction x Miss rate x Miss penalty) x Clock cycle time
(includes hit time as part of CPI)

Average memory-access time = Hit time + Miss rate x Miss penalty (ns or clocks)

Improving Cache Performance

Average memory-access time = Hit time + Miss rate x Miss penalty (ns or clocks)

How are we going to improve cache performance??

1.

2.

3.

Cache Overview Key Points

- CPU-Memory gap is a major performance obstacle
- Caches take advantage of program behavior: locality
- Designer has lots of choices -> cache size, block size, associativity, replacement policy, write policy, ...
- Time of program still only reliable performance measure