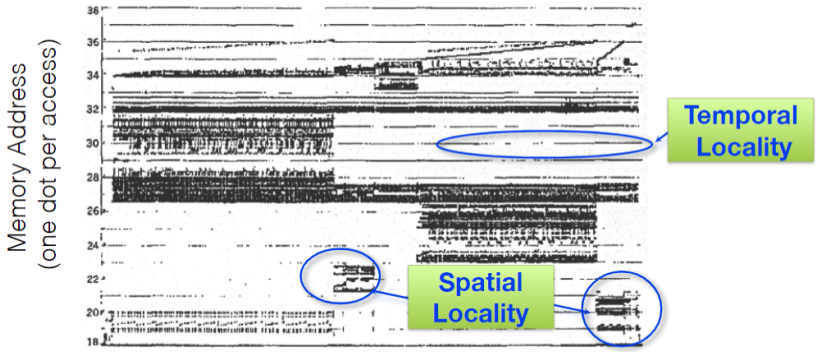
CSM CS61C Note #7: Caches

Anthony Han

1 Intro to Caches

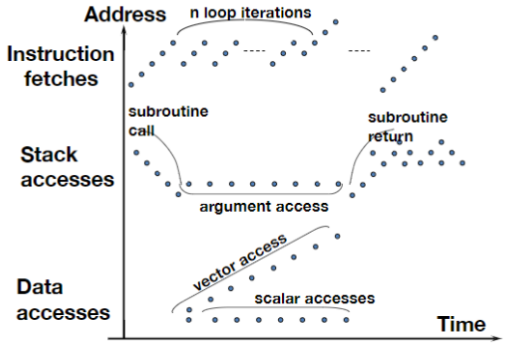
Memory Reference Patterns: Principle of Locality

- **Principle of Locality** - Programs access small portion of address space at any instant of time (spatial locality) and repeatedly access that portion (temporal locality).

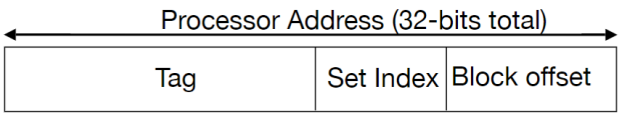


Donald J. Hatfield, Jeanette Gerald: Program Restructuring for Virtual Memory. IBM Systems Journal 10(3): 168-192 (1971)

- Some program structures lead to temporal and spatial locality in instruction and data accesses.



Processor Address Fields Used by Cache Controller



- **Block Offset** – byte address within a multi-byte block.

$$\text{Size of Offset} = \log_2(\#\text{ of bytes/block}).$$

- **Set Index** – selects which set.

$$\text{Size of Index} = \log_2(\#\text{ of sets}).$$

- **Tag** – remaining portion of processor address.

$$\text{Size of Tag} = \text{Address size} - \text{Size of Index} - \text{Size of Offset}.$$

Valid Bit

We need an indicator whether a tag entry is valid for a program; so we add a **"valid bit"** to the cache tag entry, initiated to 0.

- If valid bit is 0, then cache miss no matter what.
- If valid bit is 1, then cache hit if and only if processor address matches the tag entry in the specified set.

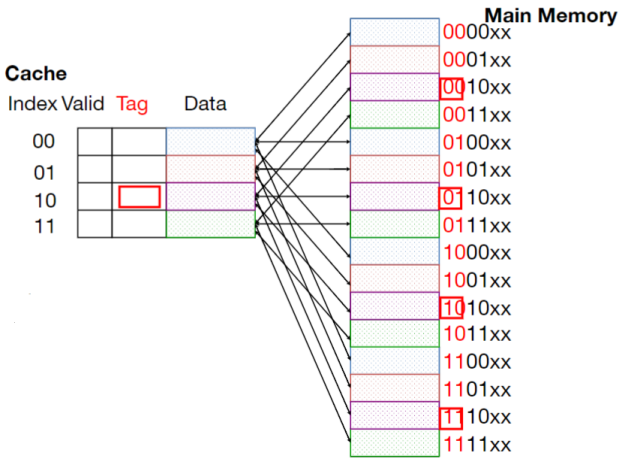
In case the cache is full, evict a block based on its replacement policy (more on that later).

Caching: A Simple First Example

Consider a Direct Mapped Cache in a 6-bit address space, where

- Individual data is 1B.
- Cache/Memory block size is 4B.
- Four cache blocks.

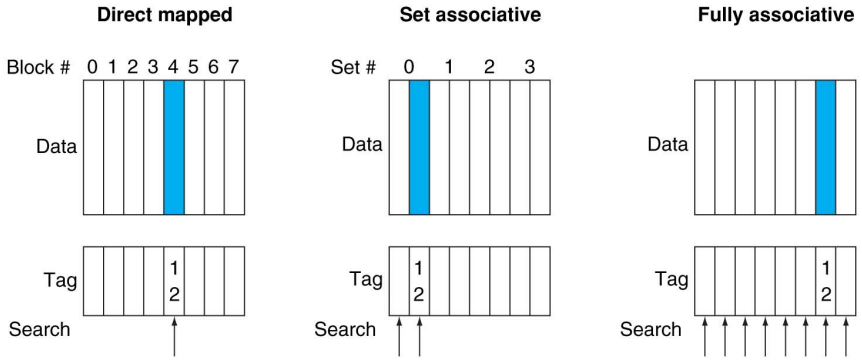
Since there are 16 memory blocks, each cache block is mapped to by 4 memory blocks.



Cache Block Placement Schemes

- **"Fully Associative"** - all blocks in one set.
  - No index field.
  - 1 comparator per block required.

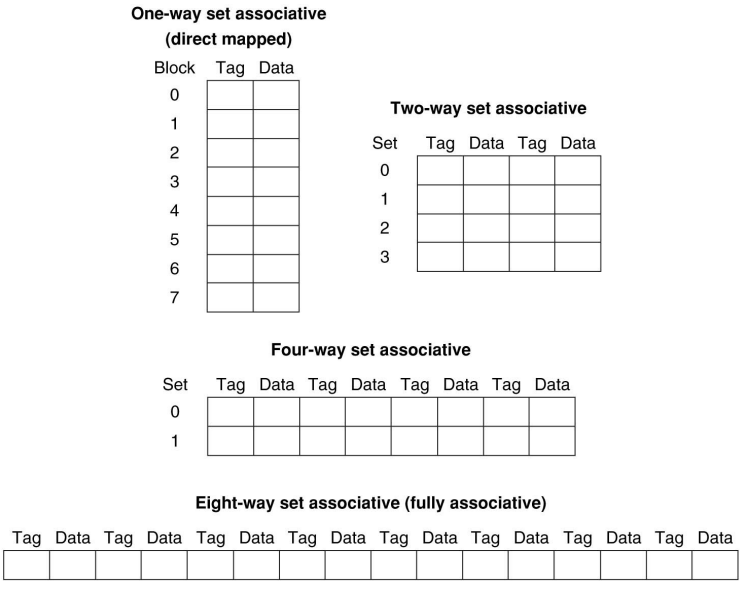
- **"Direct Mapped"** - 1 block in one set.
  - Only 1 comparator required.
  - # of sets = # of blocks.
- **"N-way Set Associative"** - N blocks in one set. N is called the **associativity** of the cache.
  - # of sets = # of blocks / N.
  - N comparators required.
  - ***Fully Associative: N = # of blocks.***
  - ***Direct Mapped: N = 1.***



In this case,

- DM placement – mem block 12 in only one cache block 4.
- SA placement – mem block 12 in set 0, either element of the set.
- FA placement – mem block 12 can appear in any cache block.

The total # of blocks in the cache is equal to the # of sets times the associativity. Here are different organizations of an eight-block cache:



We note that

$$\boxed{\text{Total cache capacity} = \text{Associativity} \times \#\text{ of sets} \times \text{Block size},}$$

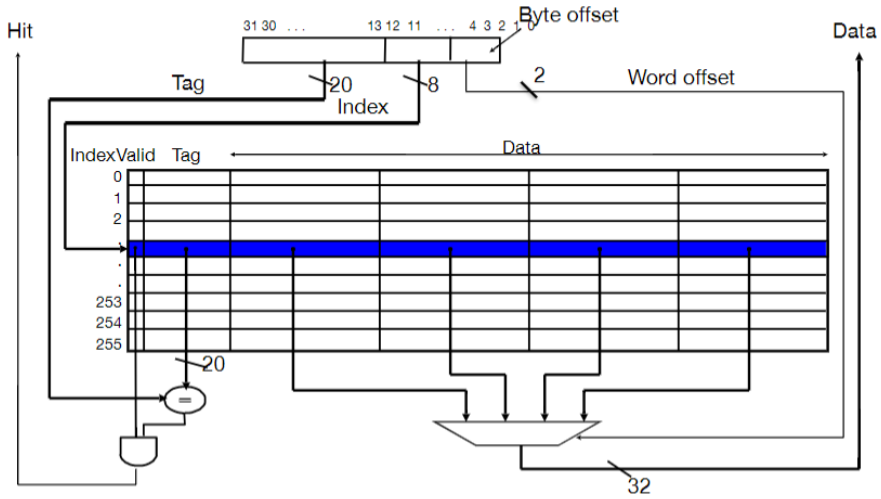
or,

$$\boxed{C = N \times S \times B.}$$

Note that

$$\text{Address width} = \text{Tag width} + \log_2 S + \log_2 B.$$

Cache Cost



An N-way set-associative cache costs

- MUX delay for set selection.
- N comparators.
- Hit/miss decision after set selection.

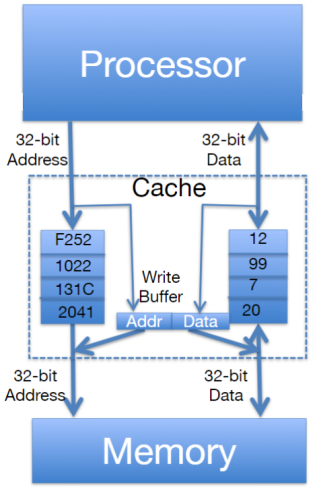
Cache Replacement Policies

- **Random Replacement** – hardware randomly selects a cache line to evict.
- **Least Recently Used (LRU)** – hardware replaces the entry that has not been used for the longest time to evict.
- Most Recently Used (MRU) – not particularly useful.

Handling Stores

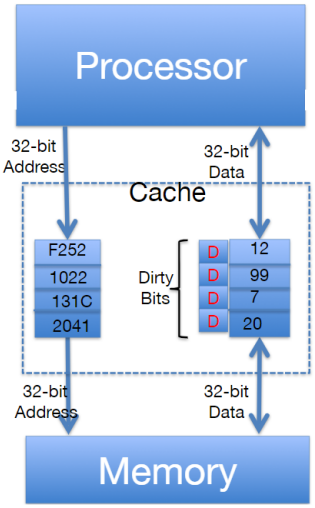
On writes, we need to make sure that the cache and memory have the same values. There are two ways to achieve this.

Write-Through Cache



- **When store hit in cache, write cache and write *through* the cache to memory.**
- Too slow, so include **write buffer**:
  - Allows processor to continue in parallel once data is in buffer so it doesn't stall.
  - May have multiple entries to absorb bursts of writes.
- When store misses in cache, typically **"no write allocate"** – only write to main memory.
- Advantages and disadvantages:
  - Simpler control logic.
  - More predictable timing.
  - Easier to make reliable, since memory always has copy of the data!

Write-Back Cache



- **When store hit in cache, write cache only and set *dirty bit*.**
  - Bit indicating if wrote to block or not.
  - Memory has stale value.
- When store misses in cache, typically **"write allocate"** (aka fetch on write) – read data from memory, then update and set dirty bit.
- On any miss, **write back evicted block only if it's dirty.** Update cache with new block and clear dirty bit.
- Advantages and disadvantages:
  - More complex control logic.
  - More variable timing (0, 1, 2 memory accesses per cache access).
  - Usually **reduces write traffic.**
  - Harder to make reliable, sometimes cache has the only copy of the data.

2 Cache Performance

Sources of Cache Misses (3C's)

- **Compulsory** - cold start, first reference

  - First access to a block, basically unavoidable.
  - Negligible if running billions of instructions.

- **Capacity**
  - Cache cannot contain all the blocks accessed by the program.
  - All lines of the cache are filled.
  - **Will not happen with infinite cache.**

- **Conflict** (Collision)
  - Multiple memory locations mapped to same cache set.
  - **Will not happen with ideal fully associative cache with LRU replacement policy of the same size.**

Cache Performance Terms

- **Hit rate** – fraction of accesses that hit in the cache.
- **Miss rate** – fraction of accesses that miss in the cache. Miss rate $= 1 -$ Hit rate.
- **Miss penalty** – time to replace a block from lower level in memory hierarchy to the cache.
- **Hit time** – time to access cache memory, including tag comparison. Property of cache.
- **Average Memory Access Time (AMAT)** – the average time to access memory considering both hits and misses in the cache.

$$\text{AMAT} = \text{Time for a hit} + \text{Miss rate} \times \text{Miss penalty}.$$

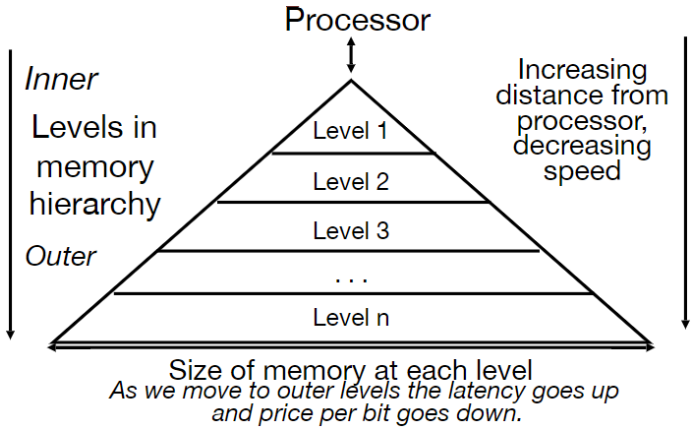
  Note that miss penalty is the **additional** time required for a cache miss.

Cache Performance Analysis

To improve cache performance, we could

- Reduce hit time in the cache – smaller cache.
- Reduce miss rate:
  - Bigger cache:
    - More blocks – more cash.
    - Longer cache lines – exploits spatial locality better but temporal locality worse.
  - Better programs.
- Reduce miss penalty – use multiple cache levels.

Reducing Miss Penalty: Multiple Cache Levels



1. If miss on the first-level cache L1, request the data from the next level, L2.
2. When we reach the last level of cache L_n and still misses, fetch the data from main memory into L_n.
3. If hits at any level (or the data is loaded from main memory), sends the data to the last level **and cache it.**
4. When we return to the first-level cache L1, send the data down the processor.

Local vs. Global Miss Rates

- **Local** miss rate - the fraction of references to one level of a cache that miss.

$$\text{Local miss rate L2\$} = \frac{\text{L2\$ misses}}{\text{L2\$ accesses}} = \frac{\text{L2\$ misses}}{\text{L1\$ misses}}.$$

- **Global** miss rate - the fraction of references that miss all levels of a multilevel cache. In a two-level cache,

$$\begin{aligned}\text{Global miss rate} &= \frac{\text{L2\$ misses}}{\text{Total accesses}} \\ &= \frac{\text{L2\$ misses}}{\text{L1\$ misses}} \cdot \frac{\text{L1\$ misses}}{\text{Total accesses}} \\ &= \text{L2\$ local miss rate} \times \text{L1\$ local miss rate}.\end{aligned}$$

  This implies L2\$ local miss rate >> Global miss rate.

- For a 2-level cache system,

  $\text{AMAT} = \text{L1\$ hit time} + \text{L1\$ miss rate} \times (\text{L2\$ hit time} + \text{L2\$ miss rate} \times \text{L2\$ miss penalty})$.