A Common Framework for Memory Hierarchies

Caching, paged virtual memory and TLBs all use the same underlying concepts

Feature	Cache	Paged Mem	TLB
Size,Blocks	1K-100K	2K-250K	32-4000
Size, Bytes	8KB-8MB	8MB-8GB	128B-8000B
Blk Size, B	4-256	4KB-64KB	4-32
Miss Penalty	10-100clk	1M-10Mclk	10-100clk
Miss Rate	0.1%-10%	10^-4-10^-5%	0.01%-2%

Four Questions for Classification

- Where can a block be placed? *Block placement*
  - direct mapped, set associative, fully associative
- How is a block found? *Block identifcation*
  - indexing, set search, separate lookup table
- What block is replaced on a miss? *Block replacement*
  - LRU, Random, FIFO, MRU
- How are writes handled? *Write strategy*
  - write through or write back

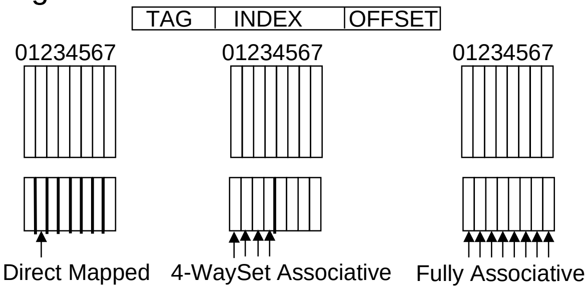
Summary and Review

Block Placement

The extremes of cache mapping -- direct mapped and fully associative are end points on a spectrum

Blocks are assigned to a cache by directly indexing any of its *n* sets and matching any of the *m* entries of the set associatively by the tag

Indexing is "block number modulo number of sets"

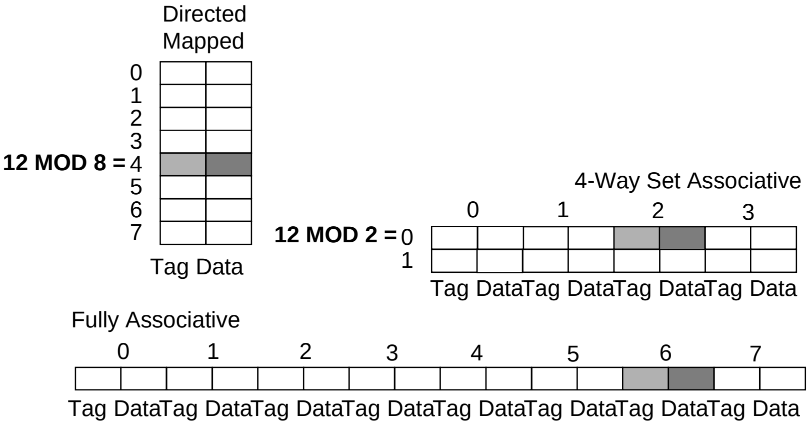

TAG | INDEX | OFFSET

01234567 01234567 01234567

Direct Mapped 4-WaySet Associative Fully Associative

Block Identification

Placement of a block whose address is 12 varies for direct, set associative, and fully associative


Directed Mapped

0 1 2 3 4 5 6 7

12 MOD 8 = 4

Tag Data

4-Way Set Associative

0 1 2 3

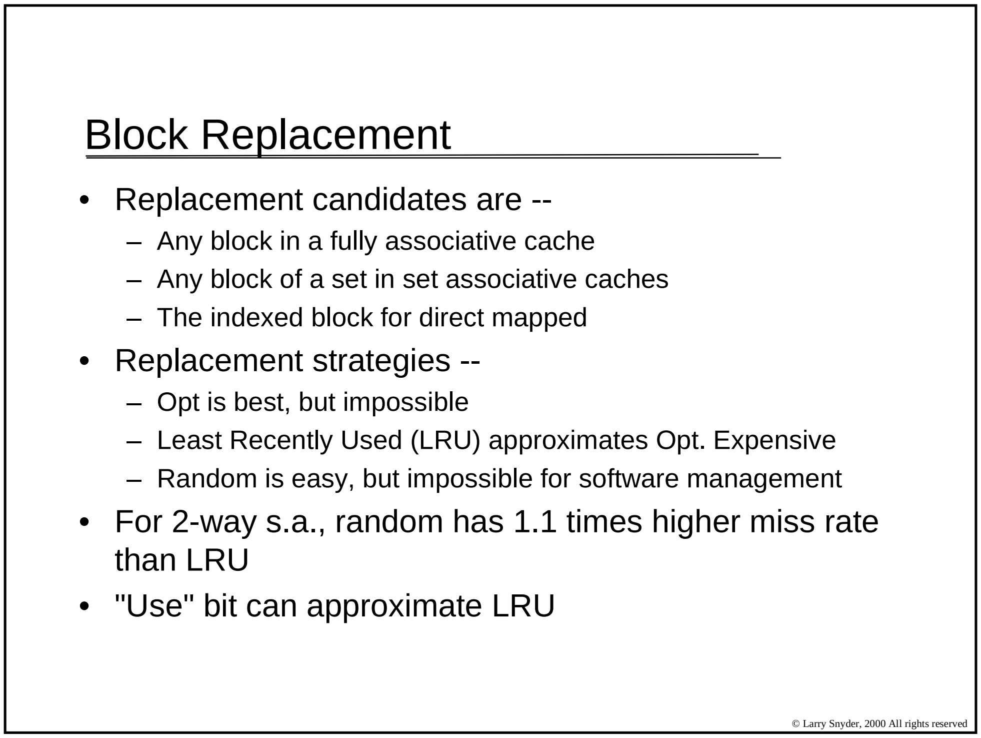
12 MOD 2 = 0 1

Tag DataTag DataTag DataTag Data

Fully Associative

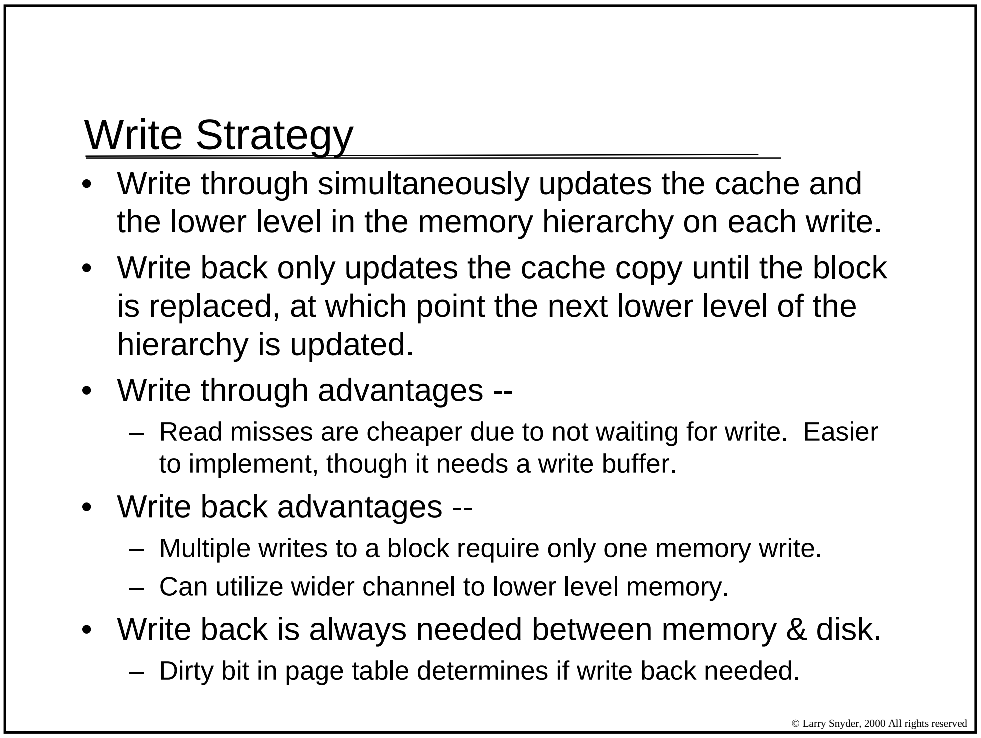
0 1 2 3 4 5 6 7

Tag DataTag DataTag DataTag DataTag DataTag DataTag DataTag Data



Block Replacement

- Replacement candidates are --
  - Any block in a fully associative cache
  - Any block of a set in set associative caches
  - The indexed block for direct mapped
- Replacement strategies --
  - Opt is best, but impossible
  - Least Recently Used (LRU) approximates Opt. Expensive
  - Random is easy, but impossible for software management
- For 2-way s.a., random has 1.1 times higher miss rate than LRU
- "Use" bit can approximate LRU



Write Strategy

- Write through simultaneously updates the cache and the lower level in the memory hierarchy on each write.
- Write back only updates the cache copy until the block is replaced, at which point the next lower level of the hierarchy is updated.
- Write through advantages --
  - Read misses are cheaper due to not waiting for write. Easier to implement, though it needs a write buffer.
- Write back advantages --
  - Multiple writes to a block require only one memory write.
  - Can utilize wider channel to lower level memory.
- Write back is always needed between memory & disk.
  - Dirty bit in page table determines if write back needed.

Mapping Choices in Hierarchy

- Tradeoff cost of miss *vs* cost of associativity
- VM uses fully associative mapping
  - Reduces miss rate, because miss penalty is high
  - Mapping done in software
  - Large page size means page table size overhead is small
  - Note that page table is indexed, but full map provides fully associative placement
- Small caches (TLB) often use set associative placement
- Large caches never use fully associative placement
  - High cost and hit time penalties
  - Small performance advantage to set associative

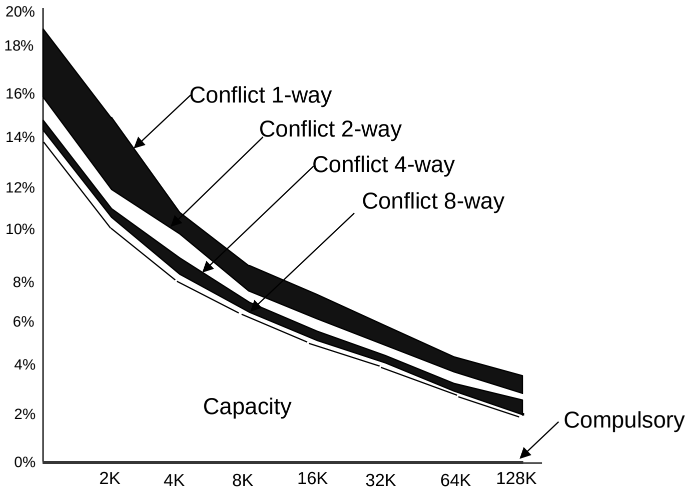
The Three Cs

Missing in the cache can be caused by three different circumstances:

- Compulsory misses -- miss on first access
- Capacity misses -- miss due to cache not having enough blocks
- Conflict misses -- miss due to cache organization

In cache design, larger is always better ... but there are always trade-offs

Miss Rates



The Problem with Miss-rate

It doesn’t tell the whole story:

Consider increasing direct-mapped cache from 32K to 64K

Miss Rate drops from 5% to 4%. If the larger cache implies a cycle time of 18ns and the smaller cache implies a cycle time of 15ns, the smaller cache machine has better performance

Postulate: CPI w/o stalls is unchanged
Miss penalty 180ns
Memory references per instruction = 1.5

CPU Time = (CPU execution clock cycles + Memory-stall clock cycles) × Clock cycle time

Cache Analysis, Continued

$$\text{Memory-stall clock cycles} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Misses}}{\text{Instruction}} \times \text{Miss penalty}$$

$$\text{Misses} = \text{Instruction miss rate} + \text{Data miss rate} \times \frac{\text{Data references}}{\text{Instructions}}$$

Let IC be *instructions per program*

<u>Smaller Cache</u>

Memory stall clock cycles =

$$IC \times (0.05 + 0.05 \times 0.5) \times \frac{\text{Absolute miss penalty}}{\text{Clockcycle time}}$$

$$= IC \times 0.075 \times 180/15 = .9IC$$

<u>Larger Cache</u>

Memory stall clock cycles =

$$IC \times (0.04 + 0.04 \times 0.5) \times \frac{\text{Absolute miss penalty}}{\text{Clockcycle time}}$$

$$= IC \times 0.06 \times 180/18 = .6IC$$

Cache Analysis, Continued

Memory-stall clock cycles = 0.9IC (Small) and 0.6IC (Large cache).

Substituting into the CPU time equation, letting CPI w/o stalls be C:

CPU Time = (CPU execution clock cycles + Memory-stalls clock cycles) × Clock cycle time

Small Cache

CPU time

$$=((C \times IC) + (0.9 \times IC) \times 15ns$$
$$= 15 \times C \times IC + 13.5 \times IC$$
$$= (15C + 13.5)IC$$

Large Cache

CPU time

$$=((C \times IC) + (0.6 \times IC) \times 18ns$$
$$= 18 \times C \times IC + 10.8 \times IC$$
$$= (18C + 10.8)IC$$

For C ≥ 1 the smaller cache is better