COMP3211 03S2 Lecture 11

Cache Design

Adapted from

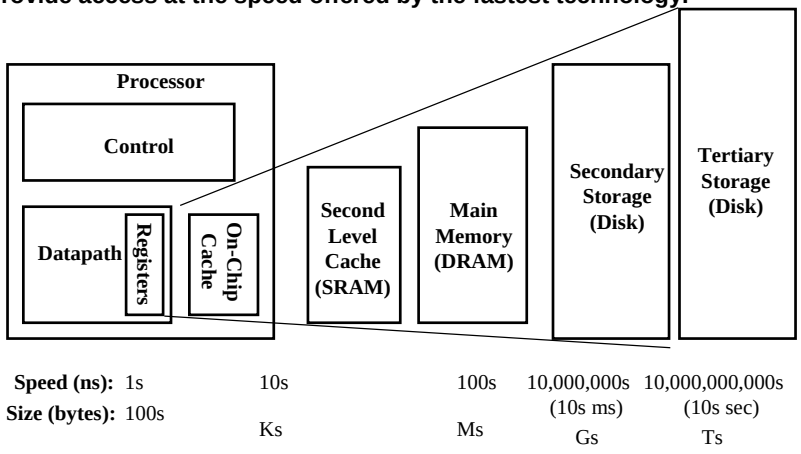
CS152: Computer Architecture and Engineering
Dave Patterson (www.cs.berkeley.edu/~pattrsn)

Before we begin: Some questions you should ask yourself

- What is the purpose of memory hierarchy in a computer system? (2 mins)
- Where is cache located in the hierarchy? (1 min)
- What is the purpose of cache in the hierarchy? (1 min)
- How does cache work? (3 mins)
- What is the effect of increasing cache size? (1 min)
- What is the effect of increasing cache associativity? (1 min)
- What is the best cache design? (2 mins)

Recap: Memory Hierarchy of a Modern Computer System

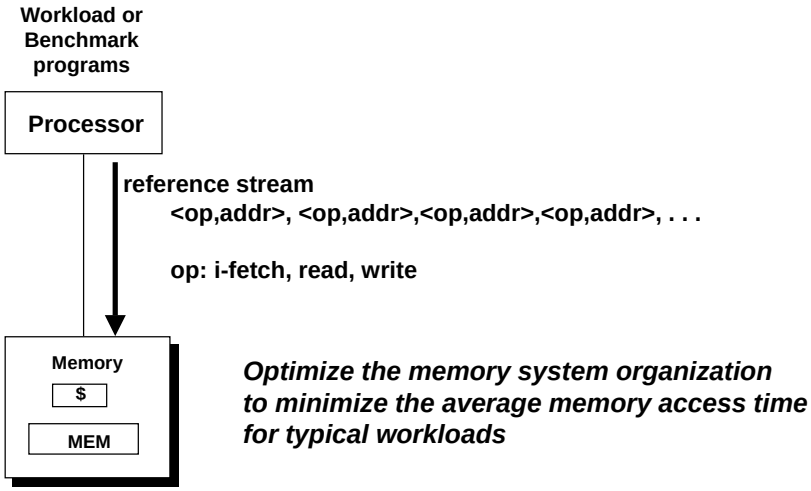
- By taking advantage of the principle of locality:
  - Present the user with as much memory as is available in the cheapest technology.
  - Provide access at the speed offered by the fastest technology.



Recap:

- Two Different Types of Locality:
  - Temporal Locality (Locality in Time): If an item is referenced, it will tend to be referenced again soon.
  - Spatial Locality (Locality in Space): If an item is referenced, items whose addresses are close by tend to be referenced soon.
- By taking advantage of the principle of locality:
  - Present the user with as much memory as is available in the cheapest technology.
  - Provide access at the speed offered by the fastest technology.
- DRAM is slow (50-100ns) but cheap and dense:
  - Good choice for presenting the user with a BIG memory system
- SRAM is fast (5-25ns) but expensive and not very dense:
  - Good choice for providing the user FAST access time.

The Art of Memory System Design

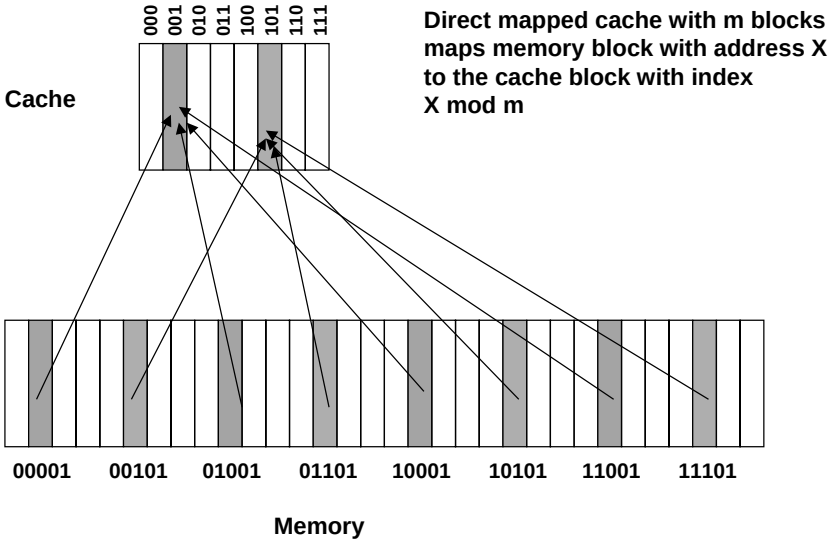


Optimize the memory system organization to minimize the average memory access time for typical workloads

We are going to examine

- How cache reduces memory access overheads
- How cache works
- What effect varying cache design parameters has
- How cache misses are classified
- How cache should be designed

Cache Principles



Direct mapped cache with m blocks maps memory block with address X to the cache block with index X mod m

Cache references

Initial state after power on

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

The (V)alid bit indicates whether data is cached at a particular index

Cache references

After handling a miss on reference to address 10110

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache references

After handling a miss on reference to address 11010

Index	V	Tag	Data
000	N		
001	N		
010	Y	11	Mem[11010]
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache references

After handling a miss on reference to address 10000

Index	V	Tag	Data
000	Y	10	Mem[10000]
001	N		
010	Y	11	Mem[11010]
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache references

After handling a miss on reference to address 00010
Contents of cache entry for index 010 replaced

Index	V	Tag	Data
000	Y	10	Mem[10000]
001	N		
010	Y	00	Mem[00010]
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

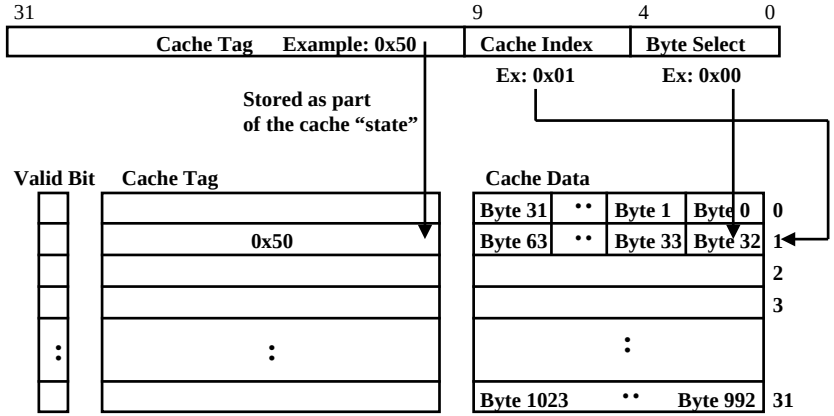
Reducing replacement of cache contents

We don't usually just store a single byte at each cache entry, rather we store a block of data to facilitate the need to store spatially related data

Example: 1 KB Direct Mapped Cache with 32 B Blocks

° **For a 2 ** N byte cache:**

- **The uppermost (32 - N) bits are always the Cache Tag**
- **The lowest M bits are the Byte Select (Block Size = 2 ** M)**



° **For a given address we compare the cache tag of the indexed entry to check it contains the requested data**

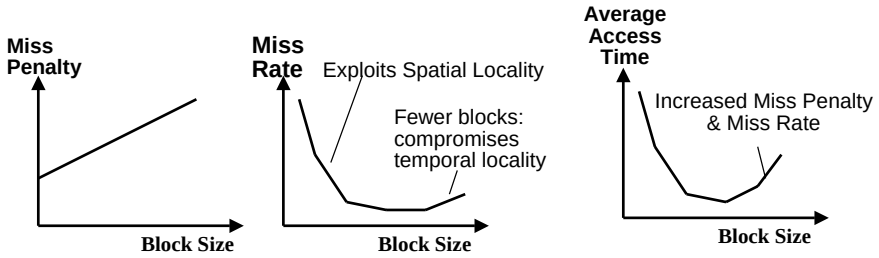
Block Size Tradeoff

° **In general, larger block size take advantage of spatial locality BUT:**

- **Larger block size means larger miss penalty:**
  - **Takes longer time to fill up the block**
- **If block size is too big relative to cache size, miss rate will go up**
  - **Too few cache blocks**

° **In gerneral, Average Access Time:**

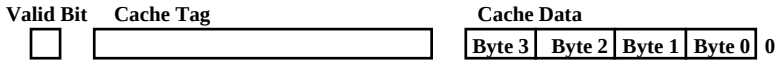
- **= Hit Time x (1 - Miss Rate) + Miss Penalty x Miss Rate**



Avoiding conflict misses

° **We'll look at two extreme cases next to see how to avoid conflict misses, in which the same memory locations are mapped to the one cache line**

Extreme Example: single big line



° Cache Size = 4 bytes Block Size = 4 bytes
- Only ONE entry in the cache

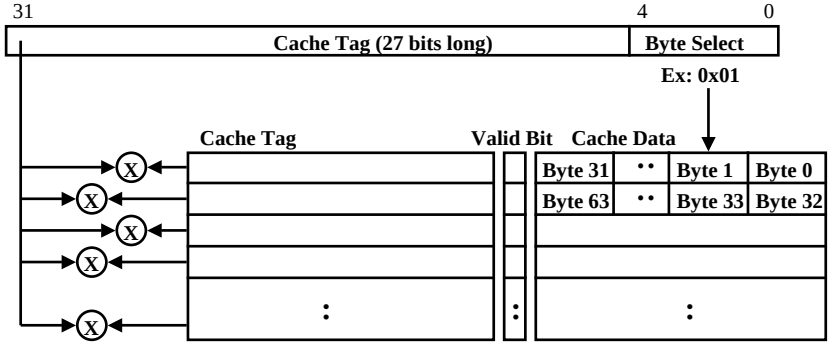
° If an item is accessed, likely that it will be accessed again soon
- But it is unlikely that it will be accessed again immediately!!!
- The next access will likely to be a miss again
  - Continually loading data into the cache but discard (force out) them before they are used again
  - Worst nightmare of a cache designer: Ping Pong Effect

° Conflict Misses are misses caused by:
- Different memory locations mapped to the same cache index
  - Solution 1: make the cache size bigger
  - Solution 2: Multiple entries for the same Cache Index

Another Extreme Example: Fully Associative

° Fully Associative Cache
- Forget about the Cache Index
- Compare the Cache Tags of all cache entries in parallel
- Example: Block Size = 32B blocks, we need N 27-bit comparators → higher comparator cost

° By definition: Conflict Miss = 0 for a fully associative cache



Question:

So how do we gain the benefits of reduced conflict misses, as is possible with full associativity, while not sacrificing the lower cost of direct mapping?

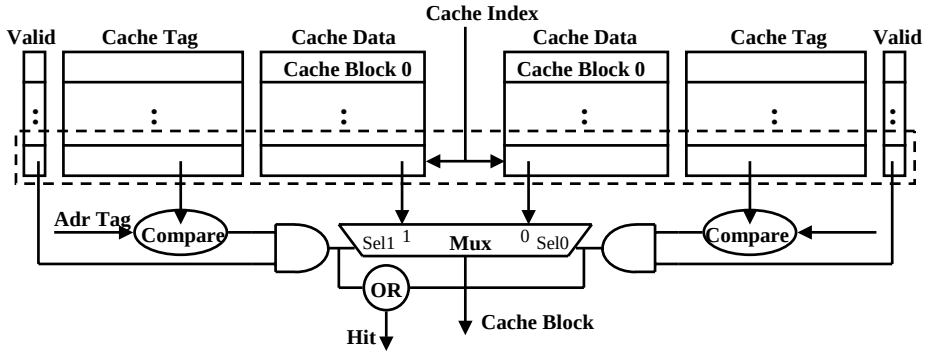
Answer:

We should consider a design compromise

A Two-way Set Associative Cache

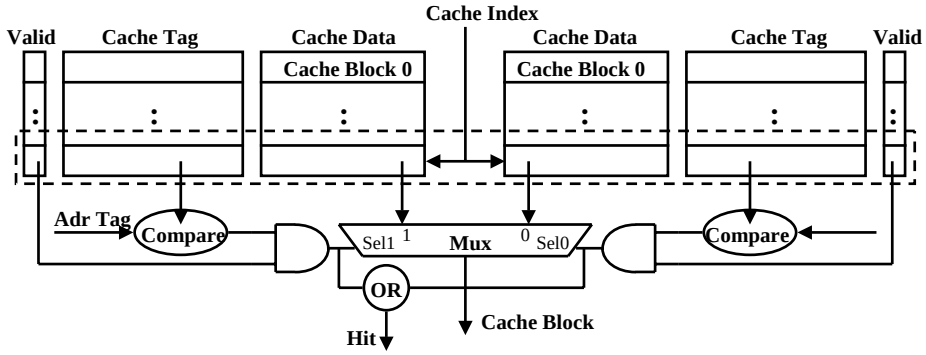
° N-way set associative: N entries for each Cache Index
- N direct mapped caches operates in parallel
- Compromise between direct-mapped & fully associative cache

° Example: Two-way set associative cache
- Cache Index selects a "set" (marked _____) from the cache
- The two tags in the set are compared in parallel
- Data is selected based on the tag result
- Reduced conflict misses and comparator cost



Disadvantage of Set Associative Cache

- N-way Set Associative Cache versus Direct Mapped Cache:
  - N comparators vs. 1
  - Extra MUX delay for the data
  - Data comes AFTER Hit/Miss decision and set selection
- In a direct mapped cache, Cache Block is available BEFORE Hit/Miss:
  - Possible to assume a hit and continue. Recover later if miss.



A Summary on Sources of Cache Misses

- Compulsory (cold start or process migration, first reference): first access to a block
  - When all entries are marked "invalid"
  - "Cold" fact of life: not a whole lot you can do about it
  - Note: If you are going to run "billions" of instructions, Compulsory Misses are insignificant
- Conflict (collision):
  - Multiple memory locations mapped to the same cache location
  - Solution 1: increase cache size
  - Solution 2: increase associativity
- Capacity:
  - Cache cannot contain all blocks accessed by the program
  - Solution: increase cache size – but there is additional indexing or comparator cost
- Invalidation: other process (e.g., I/O) updates memory
  - Question: which sensible strategies alleviate these?

Source of Cache Misses Quiz

	Direct Mapped	N-way Set Associative	Fully Associative
Cache Size: Small, Medium, Big?			
Compulsory Miss:			
Conflict Miss			
Capacity Miss			
Invalidation Miss			

Choices: Zero, Low, Medium, High, Same

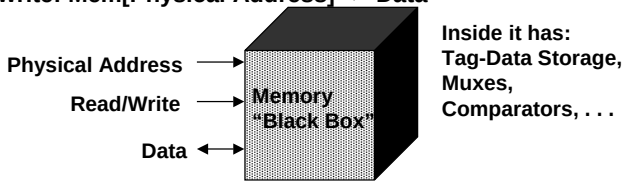
Sources of Cache Misses Answer

	Direct Mapped	N-way Set Associative	Fully Associative
Cache Size	Big	Medium	Small
Compulsory Miss	Same	Same	Same
Conflict Miss	High	Medium	Zero
Capacity Miss	Low	Medium	High
Invalidation Miss	Same	Same	Same

Note:
If you are going to run "billions" of instruction, Compulsory Misses are insignificant.

How Do you Design a Cache?

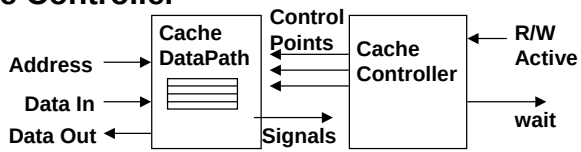
° Set of Operations that must be supported
- read: data <= Mem[Physical Address]
- write: Mem[Physical Address] <= Data



° Determine the internal register transfers

° Design the Datapath

° Design the Cache Controller

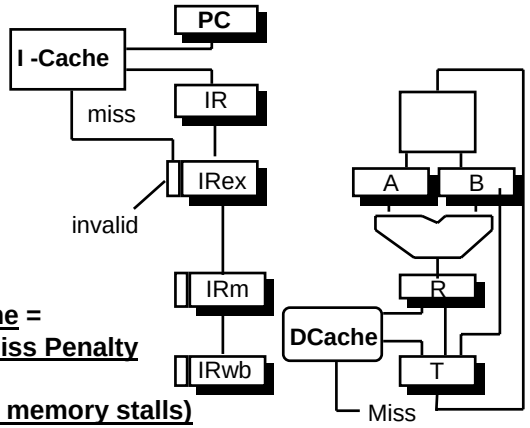


Improving Cache Performance: 3 general options

1. Reduce the miss rate,
2. Reduce the miss penalty, or
3. Reduce the time to hit in the cache.

Impact on Cycle Time

Cache Hit Time:
directly tied to clock rate
increases with cache size
increases with associativity



Average Memory Access time =
Hit Time + Miss Rate x Miss Penalty

Time = IC x CT x (ideal CPI + memory stalls)

Example: direct map allows miss signal after data

4 Questions for the design of Memory Hierarchy

° Q1: Where can a block be placed in the upper level? *(Block placement)*

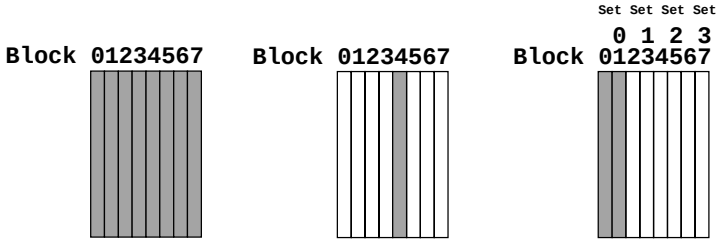
° Q2: How is a block found if it is in the upper level? *(Block identification)*

° Q3: Which block should be replaced on a miss? *(Block replacement)*

° Q4: What happens on a write? *(Write strategy)*

Q1: Where can a block be placed in the upper level?

° Block 12 placed in 8 block cache:
- Fully associative, direct mapped, 2-way set associative
- S.A. Mapping = Block Number Modulo Number Sets



Fully associative: Block 12 can go anywhere

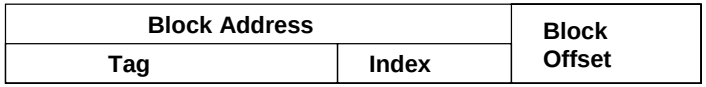
Direct mapped: Block 12 can only enter block 4 (12 mod 8)

Set associative: Block 12 can go anywhere in set 0 (12 mod 4)

Q2: How is a block found if it is in the upper level?

° Tag on each block
- No need to check index or block offset

° Increasing associativity shrinks index, expands tag



Q3: Which block should be replaced on a miss?

° Easy for Direct Mapped

° Set Associative or Fully Associative:
- Random
- LRU (Least Recently Used)

Associativity:	2-way		4-way		8-way	
Size	LRU	Random	LRU	Random	LRU	Random
16 KB	5.2%	5.7%	4.7%	5.3%	4.4%	5.0%
64 KB	1.9%	2.0%	1.5%	1.7%	1.4%	1.5%
256 KB	1.15%	1.17%	1.13%	1.13%	1.12%	1.12%

Q4: What happens on a write?

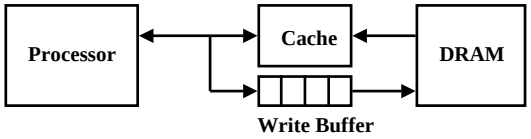
° ***Write through***—The information is written to both the block in the cache and to the block in the lower-level memory.

° ***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?
- need additional status bits

° Pros and Cons of each?
- WT: read misses cannot result in writes
- WB: no writes of repeated writes

° WT always combined with write buffers so that don't wait for lower level memory

Write Buffer for Write Through

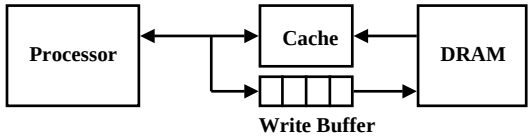


° A Write Buffer is needed between the Cache and Memory
- Processor: writes data into the cache and the write buffer
- Memory controller: write contents of the buffer to memory

° Write buffer is just a FIFO:
- Typical number of entries: 4
- Works fine if: Store frequency (w.r.t. time) << 1 / DRAM write cycle

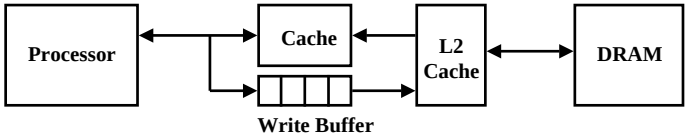
° Memory system designer's nightmare:
- Store frequency (w.r.t. time) → 1 / DRAM write cycle
- Write buffer saturation

Write Buffer Saturation



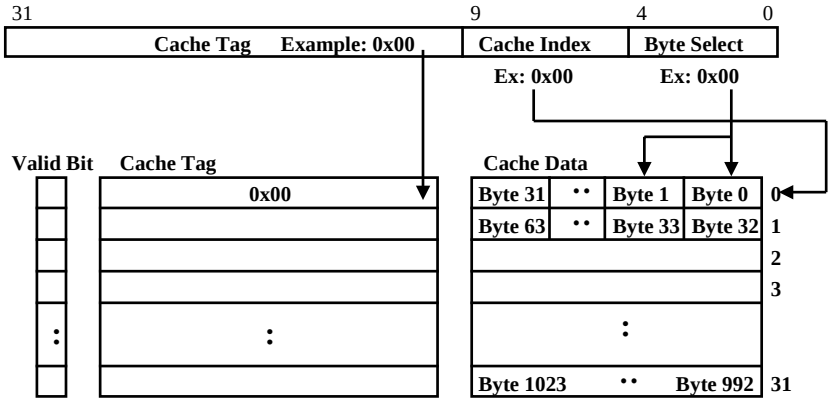
° Store frequency (w.r.t. time) → 1 / DRAM write cycle
- If this condition exists for a long period of time (CPU cycle time too quick and/or too many store instructions in a row):
  - Store buffer will overflow no matter how big you make it
  - Can happen when CPU Cycle Time <= DRAM Write Cycle Time

° Solution for write buffer saturation:
- Use a write back cache
- Install a second level (L2) cache: (expand the size of the write buffer)



Write-miss Policy: Write Allocate versus Not Allocate

° Assume: a 16-bit write to memory location 0x0 and causes a miss
- Do we read in the block?
  - Yes: Write Allocate
  - No: Write Not Allocate



Reflection

Please reflect upon what you have learnt today in the light of what you knew at the start of the lecture

° Was your understanding enhanced?

° How could you enhance your learning experience?

° Who is responsible for your learning?

° Is there something the lecturer could do to enhance your learning experience?