Topic 16: Memory Caching

COS / ELE 375

Computer Architecture and Organization

Princeton University
Fall 2015

Prof. David August

These the same?

Problem size is defined by SIZE_X and SIZE_Y

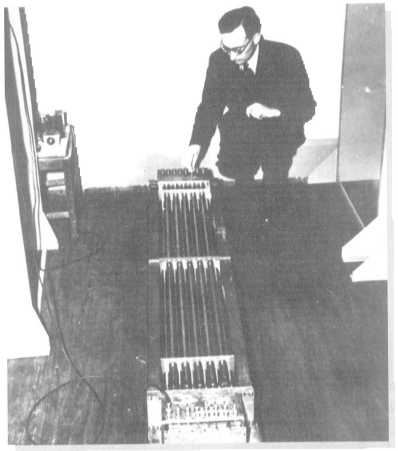
Code 1:

```
for(x = 0; x < SIZE_X; x++)
    for(y = 0; y < SIZE_Y; y++)
      sum += Array[x][y];
```

Code 2:

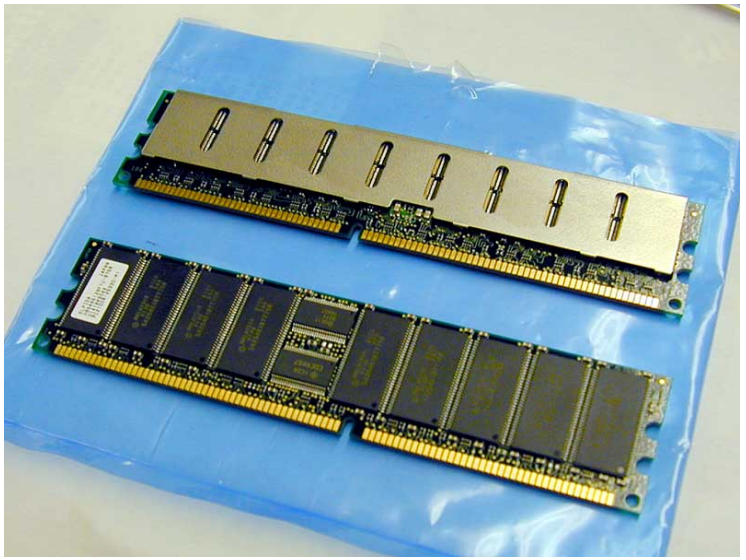
```
for(y = 0; y < SIZE_Y; y++)
    for(x = 0; x < SIZE_X; x++)
      sum += Array[x][y];
```


2 Bytes of Memory (circa 1947)



- Maurice Wilkes, in 1947, with first mercury tank memories built for EDSAC.

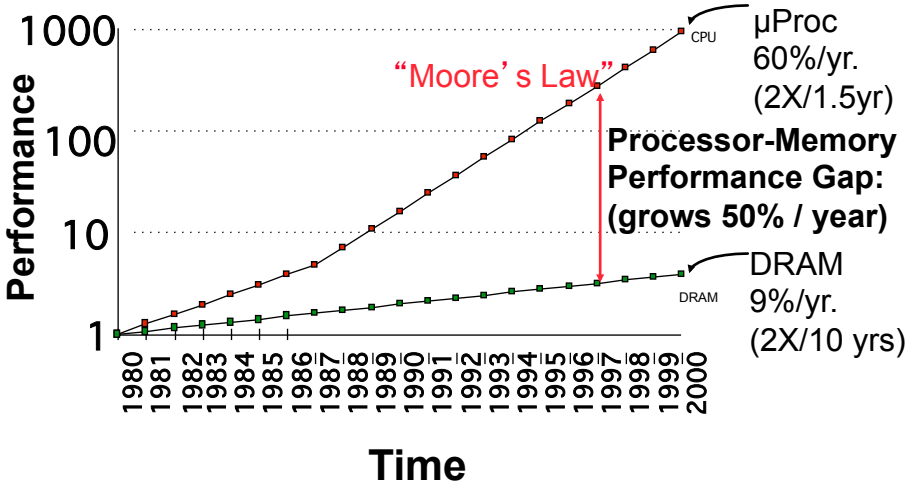
Memory (circa 2004)



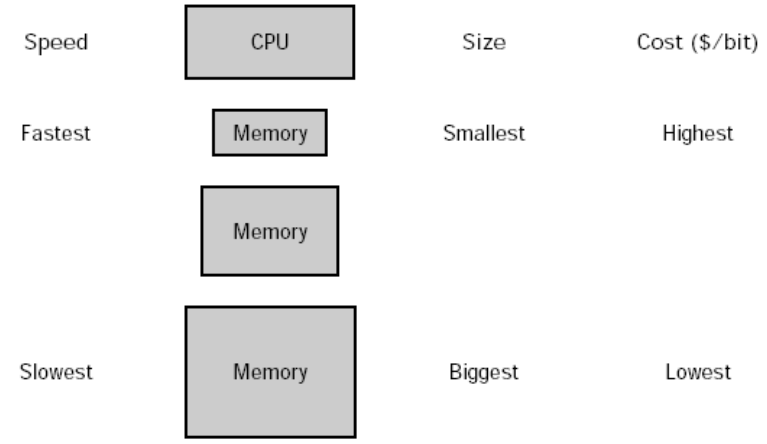
ProSeries.mov

We have a problem!

Processor-DRAM Performance Gap (latency)



Memory Speed and Cost



The Principle of Locality

- Program access a relatively small portion of the address space at any instant of time.
- The "90-10" rule…

Temporal Locality

- If an item is referenced, it will tend to be referenced again soon

Spatial Locality

- If an item is referenced, nearby items will tend to be referenced soon

Temporal and Spatial Locality

```
for (i=0; i<1000; i++) {
   for (j=0; j<1000; j++) {
        A[i,j] = B[i,j] + C[i,j];
   }
}
if (errorcond) {
   …
}
for (i=0; i<100; i++) {
   for (j=0; j<100; j++) {
        E[i,j] = D[i,j] * A[i,j];
   }
}
```

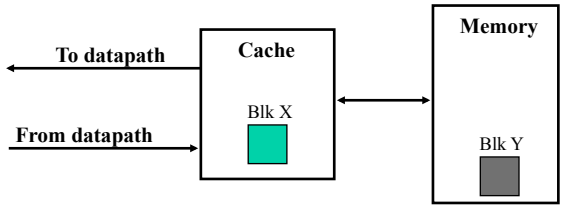
Data reference stream locality?

Instruction stream locality?

Working Sets:

- Working set refers to portion of the address space accessed
- Different phases of execution may localize on different pieces of data/code (phased behavior)

One Solution: Caching



- Hit: data appears in cache (example: Block X)
  - Hit Rate: Fraction of memory access found in cache
  - Hit Time: Time to access cache (deliver data and determine hit/miss)
- Miss: data not in cache (Block Y)
  - Miss Rate = 1 - (Hit Rate)
  - Miss Penalty: Time to replace a block in cache + deliver data
- Hit Time << Miss Penalty

Cache Management?

- Compiler/Programmer, Static
- Compiler/Programmer, Dynamic
  - Memory/Disk
  - Operating system with HW support (virtual memory)
  - Demand Fetched
- Hardware, Dynamic
  - CPU/Memory
  - Demand Fetched

Invisible to the program, except for performance

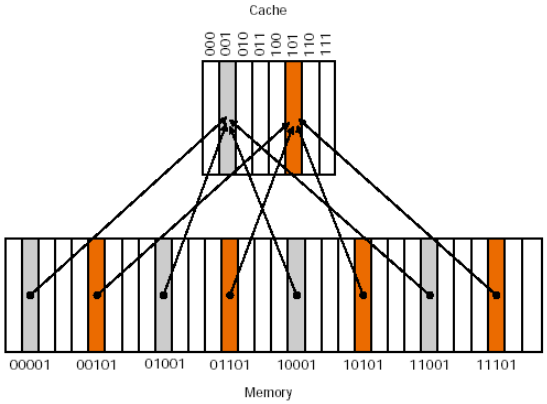
Improving Cache Performance: 3 Paths

Memory Latency = hit time + P(miss) * miss penalty

- Reduce the miss rate
- Reduce the miss penalty
- Reduce the time to hit in the cache.

Look at the cache design strategies that impact these…

Direct Mapped Cache



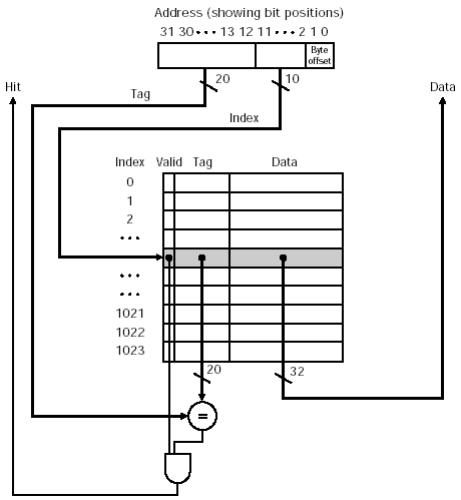
Mapping: address is modulo the number of blocks

When can this behave badly? Pros?

Block Placement
Direct Mapped Cache

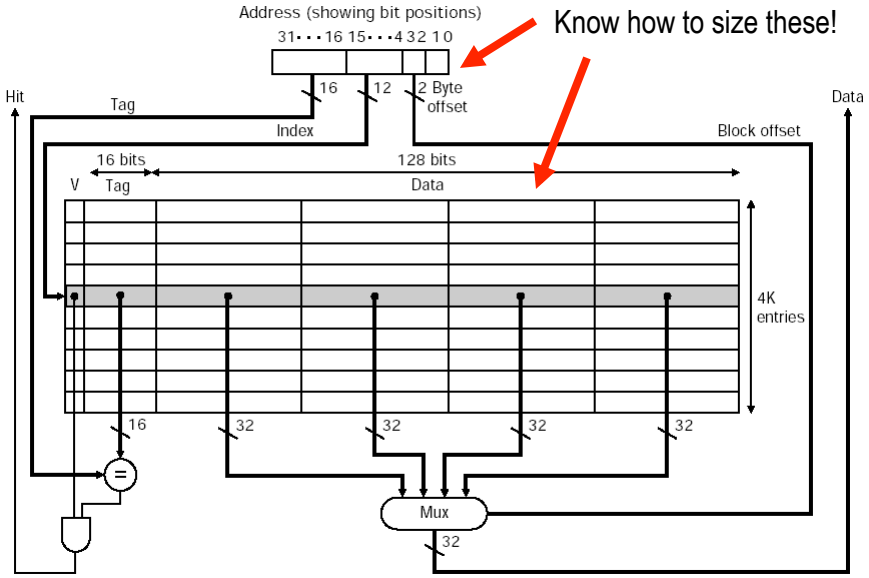
- How many hits, misses in direct mapped cache (mod 256)?
  - Read location 0: Miss
  - Read location 16: Miss
  - Read location 32: Miss
  - Read location 0: Hit
  - Read location 16: Hit
  - Read location 32: Hit
  - Read location 256: Miss
  - Read location 256: Hit
  - Read location 0: Miss
- Miss rate = 5/9 = 55%
- Note "types" of misses:
  - Cold misses
  - Conflict misses
  - Also a third type (not here): capacity misses
  - The three "C"s of cache misses

Direct Mapped Cache: Hardware



- Implementation of Mod
- Tags
- Valid
- Data
- How much state?

Direct Mapped Cache: Hardware
Capture Spatial Locality



4 Questions for Caching
Answers for Direct Mapped Caching?

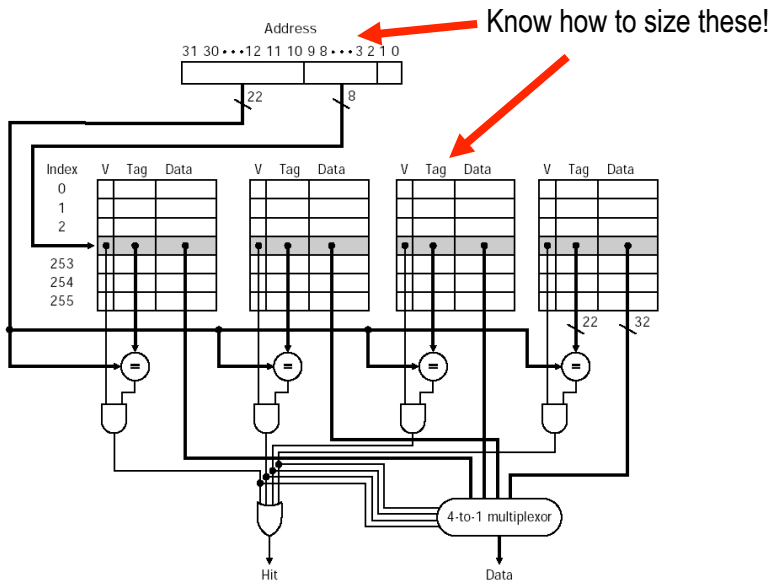
- Q1: Where can a block be placed in cache?
  (Block placement)
- Q2: How is a block found if it is in cache?
  (Block identification)
- Q3: Which block should be replaced on a miss?
  (Block replacement)
- Q4: What happens on a write?
  (Write strategy)

Reduce Conflict Misses

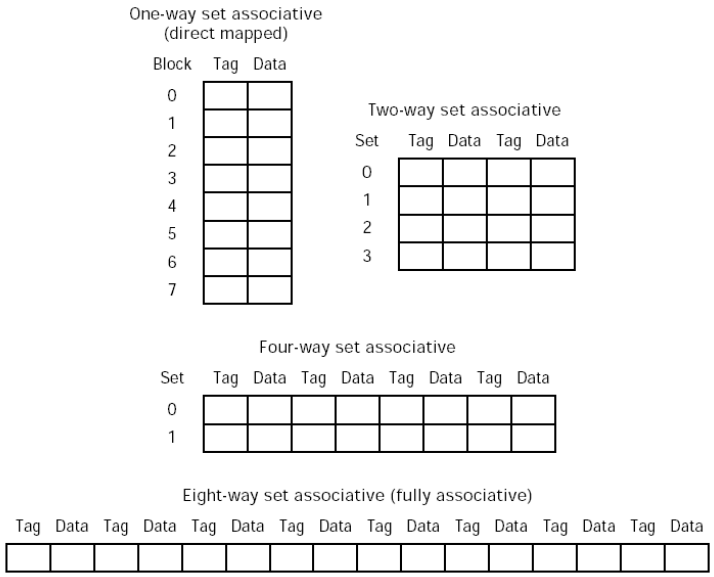
Memory time = Hit time + Prob(miss) * Miss penalty

- Previous example demonstrated conflict misses in direct-mapped cache
- Associativity: Allow blocks to go to several frames in cache
- Helps avoid pathological conflicts: 0,256,0,256,0,256…
- 2-way set associative: each block maps to either of 2 cache frames
- Fully associative: each block maps to any cache frame

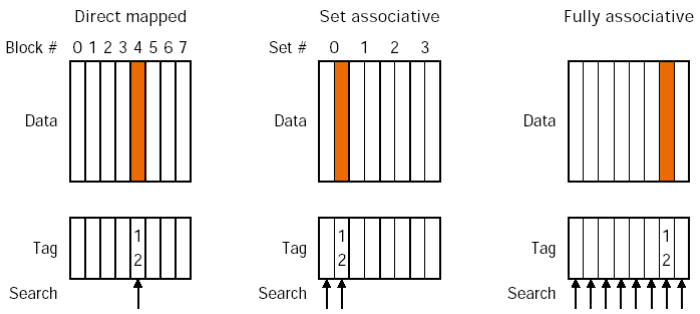
Four-Way Set Associative Cache



Direct → Fully Associative



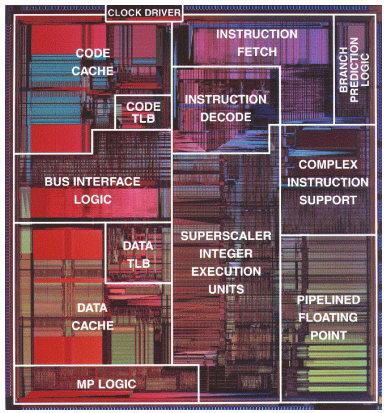
Associativity: Hardware Cost



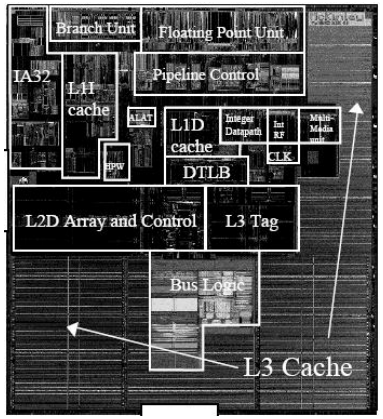
4 Questions for Caching
Set/Fully Associative Mapped Caching?

- Q1: Where can a block be placed in cache? (Block placement)
- Q2: How is a block found if it is in cache? (Block identification)
- Q3: Which block should be replaced on a miss? (Block replacement)
- Q4: What happens on a write? (Write strategy)

Caches take up 20-40+% of chip area!

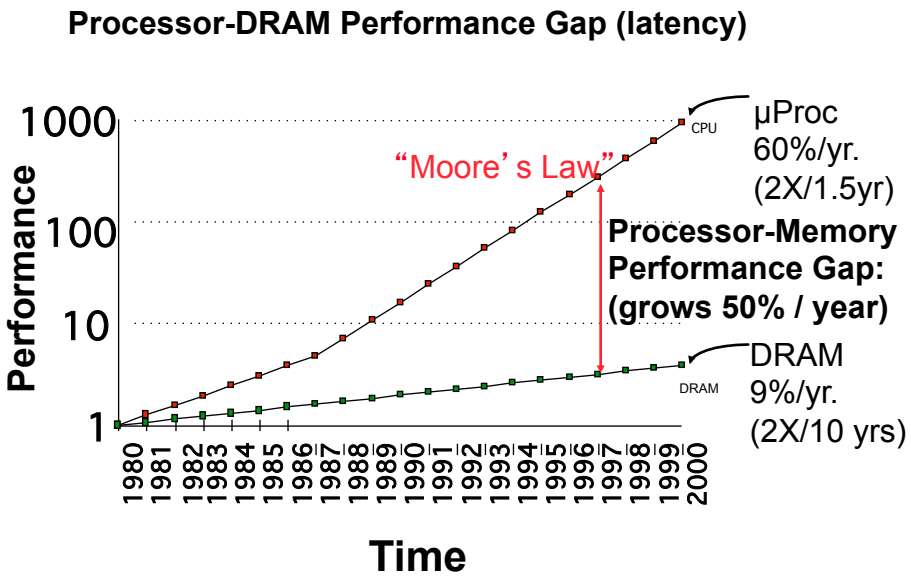


Pentium



Itanium 2 "McKinley"

We still have a problem!



Caching and The Principle of Locality

- Program access a relatively small portion of the address space at any instant of time. (90-10 rule)

Temporal Locality

- If an item is referenced, it will tend to be referenced again soon

Spatial Locality

- If an item is referenced, nearby items will tend to be referenced soon

USE CACHES!

Spatial Locality in Instruction & Data

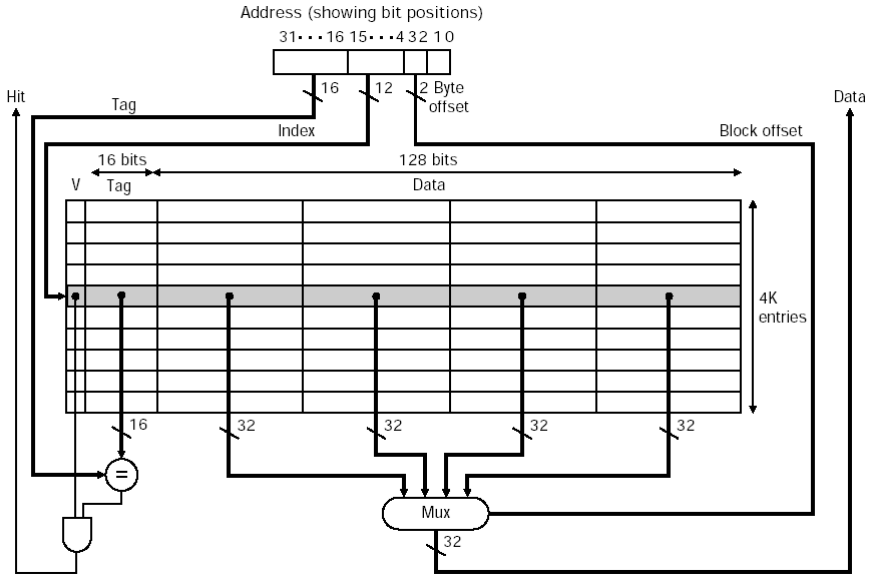
Instruction and Data References have distinct behavior:

Program	Block size in words	Instruction miss rate	Data miss rate	Effective combined miss rate
gcc	1	6.1%	2.1%	5.4%
	4	2.0%	1.7%	1.9%
spice	1	1.2%	1.3%	1.2%
	4	0.3%	0.6%	0.4%

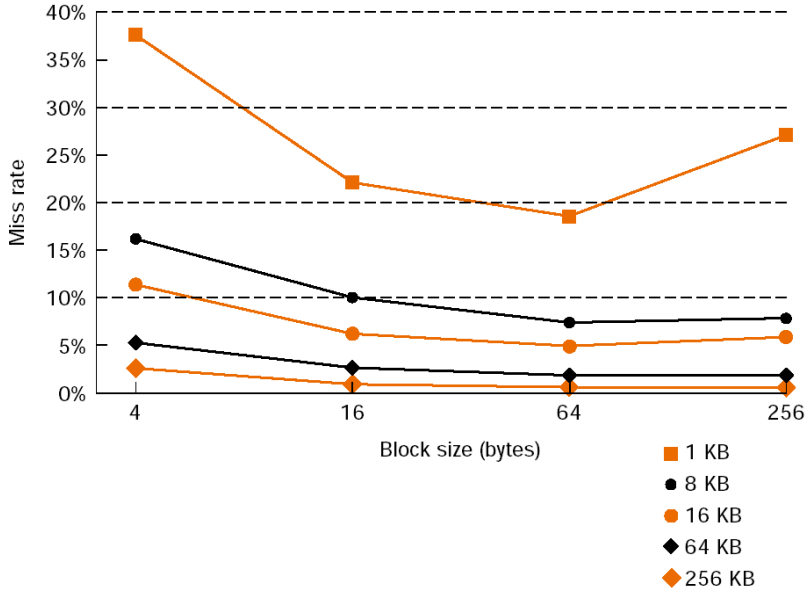
Split Instruction and Data Caches

- Optimize for behavior
- Smaller caches are faster
- Problem - when data is code or code is data

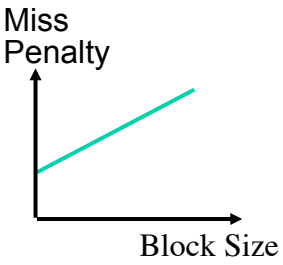
Direct Mapped Cache, Increased Block Size Capture Spatial Locality

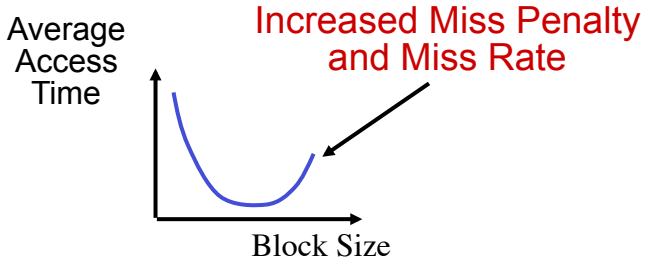


Block Size Increase: Miss Rate



Block Size Increase: Overall Performance





Block Size Increase: Fill Time

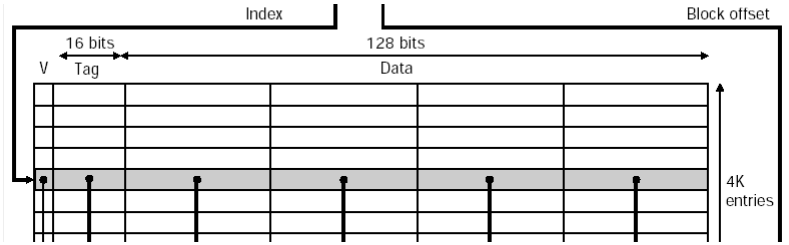
Larger Block Size → Must Wait for Block to Fill

Early Restart

- Deliver word to process/continue execution when word requested is delivered.

Critical Word First

- Early Restart and Fetch the requested word first.



The Three Types of Cache Misses
1. Conflict Misses

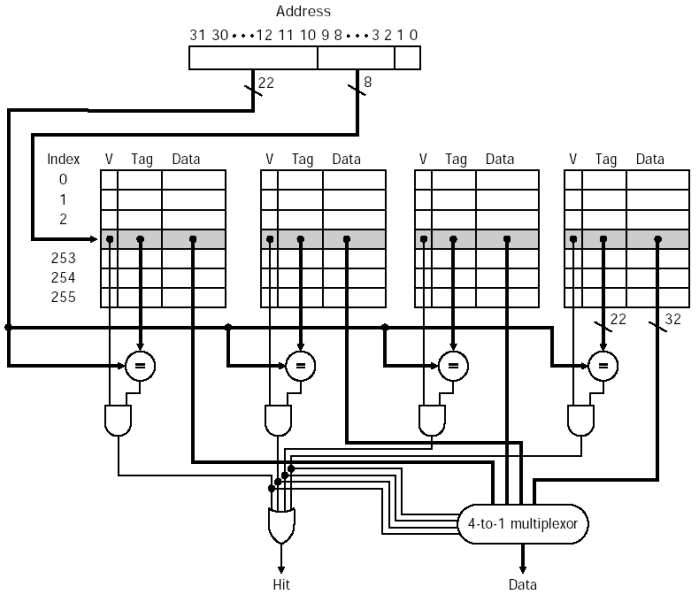
- Two distinct memory addresses map to the same cache location
- Big problem in direct-mapped caches

How do we reduce these?

Solution 1: Make cache bigger (limits)

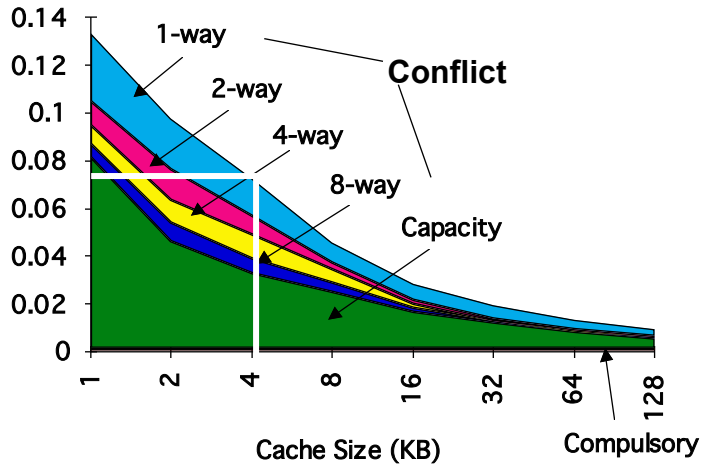
Solution 2: …

Four-Way Set Associative Cache
Avoid Conflicts



2:1 Cache Rule

Rule of Thumb: a direct-mapped cache of size N has about the same miss rate as a 2-way set associative cache of size N/2.



The Three Types of Cache Misses
2. Capacity Misses

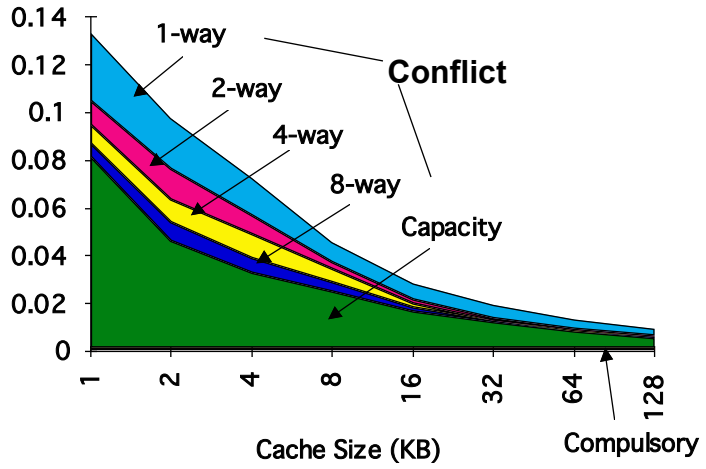
- Occurs because the cache has a limited size
- Increase the size of the cache, it goes away

- Sketchy definition, so just get the general idea
- Easy to understand in Fully Associative Caches.

How do we reduce these?

Capacity Misses

Fully Associative Cache yields no conflict misses.



The Three Types of Cache Misses
3. Compulsory Misses

- Occur when a program is first started
- Cache does not contain any of program's data yet

How do we reduce these?

Prefetching!
Reduces all types of misses, including "compulsory"!

Original Code:

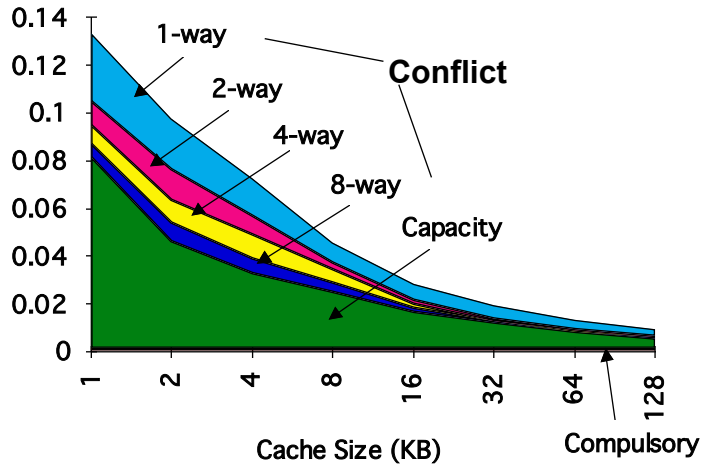
```
for(y = 0; y < SIZE_Y; y++)
    for(x = 0; x < SIZE_X; x++)
      sum += Array[x][y];
```

Code with Prefetching (ignoring boundary condition):

```
for(y = 0; y < SIZE_Y; y++)
    for(x = 0; x < SIZE_X; x++) {
      junk = Array[x+16][y];
      sum += Array[x][y];
}
```


Compulsory Misses

Fully Associative Cache yields no conflict misses.



3C Summary

Compulsory misses (cold start)

- Cold fact of life
- First time data is referenced
- Run billions of instructions, become insignificant

Capacity misses

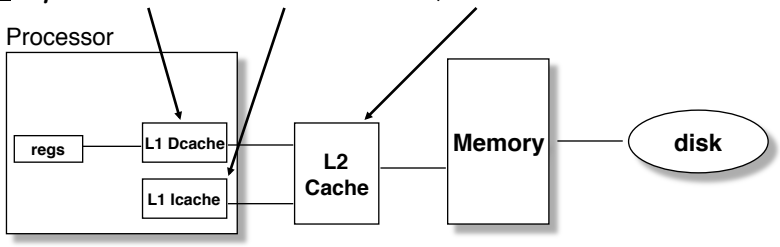
- Working set is larger than cache size
- Solution: increase cache size

Conflict misses

- Multiple memory locations mapped to the same location
- One set fills up, but space in other cache sets
- Solution 1: increase cache size
- Solution 2: increase associative indexes

Multi-Level Caches

Options: ***separate*** data and instruction caches, or a ***unified*** cache



Inclusive vs. Exclusive

Sample Sizes:

- L1: 32KB, 32 Byte Lines, 4-Way Set Associative
- L2: 256KB, 128 Byte Lines, 8-Way Set Associative
- L3: 4MB, 256 Byte Lines, Direct Mapped

Split Instruction and Data Caches

Self-Modifying Code!?!?

- Ignore problem, software must flush cache
- Permit duplicate lines: invalidate I-cache line on write
- Do not permit duplicate lines: data is exclusive to D- or I-Cache

- Page Faults - More next week

Handling Writes in Caches

First, Two Observations:

1. Writes change state → wait until exceptions are cleared
2. Stores aren't the source of a dependence - latency tolerant

Typical Implementation Decisions:

- Cache write policy?
  - Write-Through
  - Write-Back
  - Write-Around
- Include a Write buffer?
  - Small pseudo-FIFO buffer alongside cache

Write-Back vs. Write-Through Caches

Write back

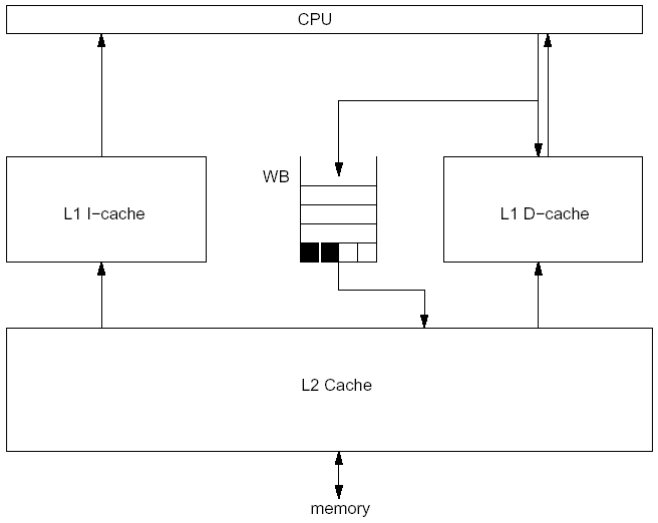
- Writes only go into top level of hierarchy
- Maintain a record of "dirty" lines
- Faster write speed (only has to go to top level to be considered complete)

Write through

- All writes go into L1 cache and then also write through into subsequent levels of hierarchy
- Better for "cache coherence" issues
- No dirty/clean bit records required
- Faster evictions

Write Around?

Write Buffer



Source: Skadron/Clark

Cache Summary

- Two types of locality: spatial and temporal
- Spatial locality: larger block sizes
- Cache contents include data, tags, and valid bits

- Miss penalty is increasing (processor vs. memory)
- Modern processors use set-associative caches worth the cost

- Multi-level caches used to reduce miss penalty

- Variations: Victim Caches, Trace Caches