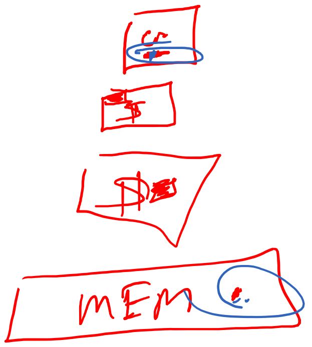
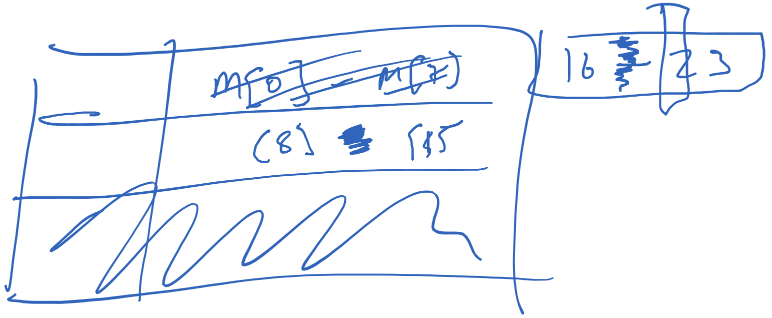
Dealing with Stores



- Stores must be handled differently than loads, because…
  - they don't necessarily require the CPU to stall.
  - they change the content of cache/memory (creating memory *consistency* issues)
- Q: Can you think of a situation when you might need to load from memory before you can execute a store?
  - Can you think of another one?

Store [12]
[18]
load [20]

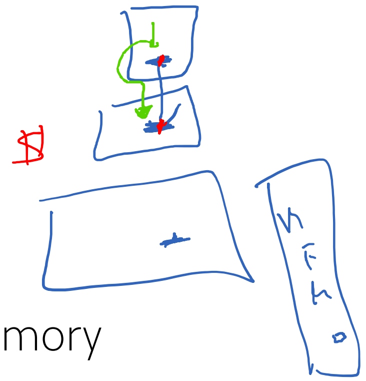


Policy decisions for stores

What do we do when the memory block is in the cache

- Keep memory and cache identical?
  - write - through => all writes go to both cache and main memory
  - write - back => writes go only to cache. Modified cache lines are written back to memory when the line is replaced. ↳ what happens on evict?
- Make room in cache for store miss?
  - *write-allocate* => on a store miss, bring written line into the cache
  - *write-around* => on a store miss, ignore cache

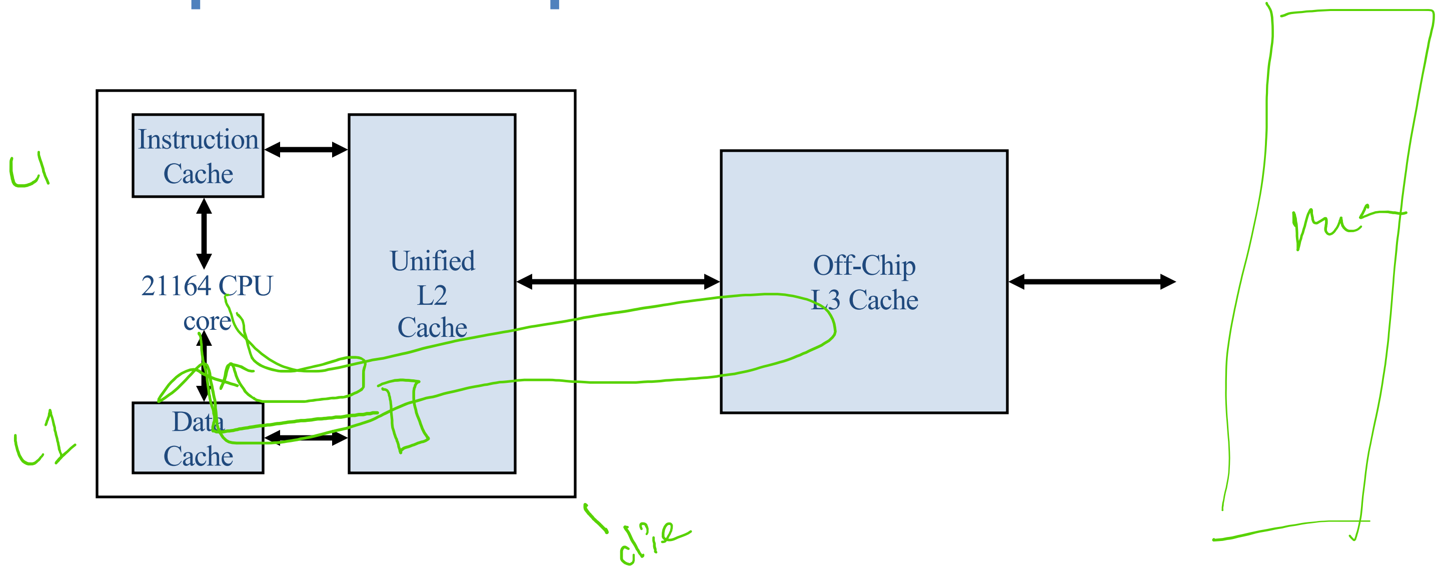
What do we do when the block is not in the cache already



Dealing with stores

- On a store hit, write the new data to cache.
  - In a *write-through* cache, write the data immediately to memory.
  - In a *write-back* cache, mark the line as dirty.
- On a store miss, initiate a cache block load from memory for a write-allocate cache.
  - Write directly to memory for a write-around cache.
- On any kind of cache miss in a write-back cache, if the line to be replaced in the cache is dirty, write it back to memory.

Example -- DEC Alpha 21164 Caches

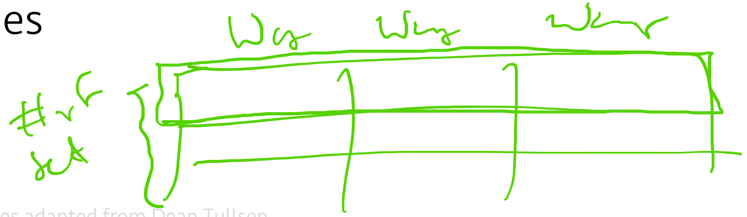


- ICache and DCache -- 8 KB, DM, 32-byte lines
  - D$ is also dual-read-ported, single-write ported, write-through, read-allocate...
- L2 cache -- 96 KB, ?-way SA, 32-byte lines
- L3 cache -- 1 MB, DM, 32-byte lines

$\frac{96KB}{32} = 3072$ entries $\frac{}{3}$ $= 1024$ sets

3

10 idx bits



Cache Performance

1.0 for scalar

- CPI = BCPI + MCPI
  - BCPI = base CPI, which means the CPI assuming perfect memory
  - MCPI = the memory CPI, the number of cycles (per instruction) the processor is stalled waiting for memory.

Cache Performance

CPI = BCPI + MCPI

– BCPI = base CPI, which means the CPI assuming perfect memory

– MCPI = the memory CPI, the number of cycles (per instruction) the processor is stalled waiting for memory.

MCPI = accesses/instruction * miss rate * miss penalty

– this assumes we stall the pipeline on both read and write misses, that the miss penalty is the same for both, that cache hits require no stalls.

– If the miss penalty or rate is different for Inst cache and data cache (common case), then

$$\text{MCPI} = \underbrace{\underbrace{\text{I\$ accesses/inst}}_{1.0} * \underbrace{\text{I\$MR}}_{10\%} * \underbrace{\text{I\$MP}}_{20\text{cyc}}}_{+2} + \underbrace{\text{D\$ acc/inst}}_{\%\text{age} \,*\, \text{ld/st} \, 50\%} * \underbrace{\text{D\$MR}}_{5\%} * \underbrace{\text{D\$MP}}_{10\text{ cyc}}$$

2.5

1.0 + 2 + 2.5

In fact…

- Can generalize this formula further for other stalls:

- CPI = $\boxed{\text{BCPI}}$ + DHSPI + BHSPI + MCPI
  - ~~DHSPI~~ = data hazard stalls per instruction
  - ~~BHSPI~~ = branch hazard stalls per instruction.

Cache Performance

Instruction cache miss rate of 4%

Data cache miss rate of 10%

BCPI = 1.0 (no data or control hazards)

20% of instructions are loads and stores

Miss penalty = 12 cycles

CPI = ~~???~~

$$1.0 + \underbrace{}_{I} + \underbrace{}_{D}$$

$$\underbrace{1.0 * 0.04 * 12} \quad \underbrace{.2 * .10 * 12}$$

$$1.0 + 0.48 + 0.24 = 1.72$$

Selection	CPI (rounded if necessary)
A	1.24
B	1.34
C	1.48
D	1.72
E	None of the above

Cache Performance

- Unified cache
- 25% of instructions are loads and stores
- BCPI = 1.2, miss penalty of 10 cycles

- If we improve the miss rate from 10% to 4% (e.g. with a larger cache), how much do we improve performance?

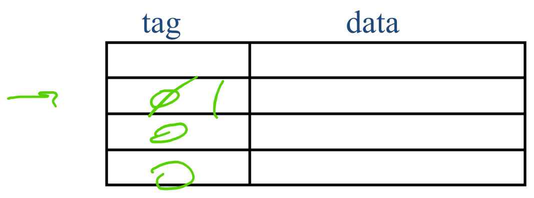
Cache Performance

- BCPI = 1
- Miss rate of 8% overall, 20% loads, miss penalty 20 cycles, never stalls on stores.

- What is the speedup from doubling the CPU clock rate?

Three types of cache misses

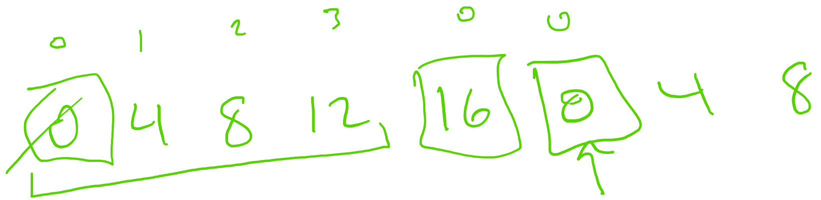
- Compulsory (or cold-start) misses
  - first access to the data.
- Capacity misses — would also happen in FA, LRU cache
  - we missed only because the cache isn't big enough.
- Conflict misses
  - we missed because the data maps to the same line as other data that forced it out of the cache.



DM cache

address string:

4	00000100	M	Comp
8	00001000	M	Comp
12	00001100	M	Comp
4	00000100	H	
8	00001000	H	
$\boxed{20}$	00010100	M	Comp
4	00000100	M	Conflict
8	00001000		
20	00010100		
24	00011000		
12	00001100		
8	00001000		
4	00000100		



Q: Categorizing Misses

- Suppose you experience a cache miss on a block (let's call it block A).
- You have accessed block A in the past. There have been precisely 1027 different blocks accessed between your last access to block A and your current miss.
- Your block size is 32-bytes and you have a 64KB cache. What kind of miss was this?

Selection	Cache Miss
A	~~Compulsory~~
B	~~Capacity~~
[✓ C]	[✓ Conflict]
D	Both Capacity and Conflict
E	None of the above

$$\frac{2^{16}}{2^{5}} = 2^{11} \quad 2K \text{ entries}$$

2048

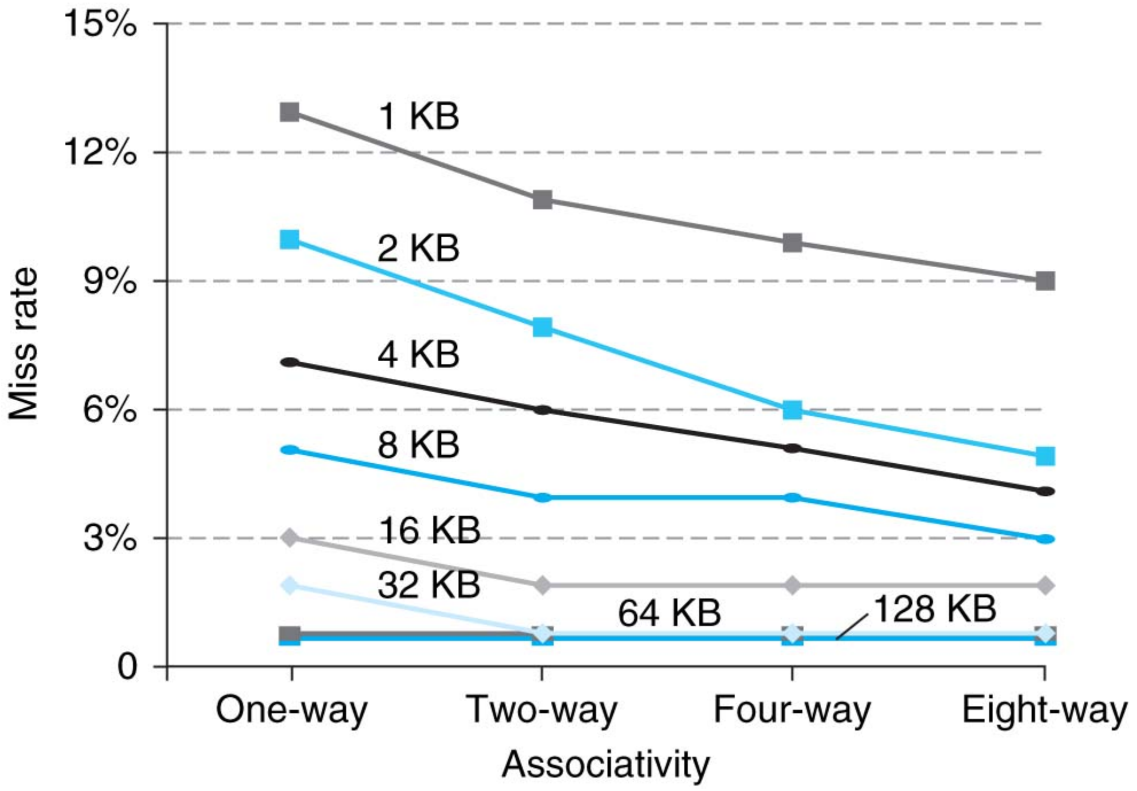
So, then, how do we decrease…

- Compulsory misses?

$\underbrace{0 \quad 4 \quad 8 \quad 12}$, . .

- Capacity misses?
- Conflict misses?

Cache Associativity



LRU replacement algorithms

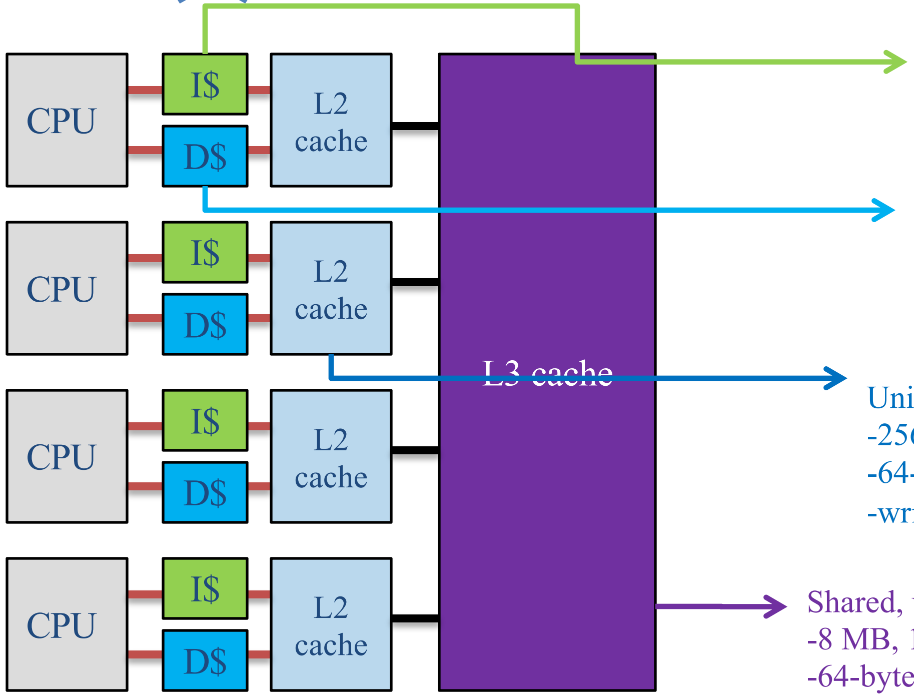
- only needed for associative caches
- requires one bit for 2-way set-associative, 8 bits (per set, 2/line) for 4-way, 24 bits for 8-way...
- can be emulated with log n bits (NMRU)
- can be emulated with use bits for highly associative caches (like page tables)
- However, for most caches (eg, associativity <= 8), LRU is calculated exactly.

Caches in Current Processors

- Not long ago, they were DM at lowest level (closest to CPU), associative further away. Today they are less associative near the processor (2-4+), and more associative farther away (4-16).
- split I and D close (L1) to the processor (for throughput rather than miss rate), unified further away (L2 and beyond).
- write-through and write-back both common, but never write-through all the way to memory.
- 64-byte cache lines common (but getting larger)

- Non-blocking
  - processor doesn't stall on a miss, but only on the use of a miss (if even then)
  - this means the cache must be able to keep track of multiple outstanding accesses, even multiple outstanding misses.

Intel Nehalem (i7)



Instruction Cache
-32 KB, 4-way
-64-byte line

Data Cache
-32 KB, 8-way
-64-byte line
-write-back, write-allocate

Unified L2 Cache
-256 KB, 8-way
-64-byte line
-write-back, write-allocate

Shared, unified L3 Cache
-8 MB, 16-way
-64-byte line
-write-back, write-allocate

Key Points

- Caches give illusion of a large, cheap memory with the access time of a fast, expensive memory.
- Caches take advantage of memory locality, specifically temporal locality and spatial locality.
- Cache design presents many options (block size, cache size, associativity, write policy) that an architect must combine to minimize miss rate and access time to maximize performance.

~~ADVANCED CACHE ARCHITECTURES~~