CS152
Computer Architecture and Engineering
Lecture 19

Caches and Virtual Memory

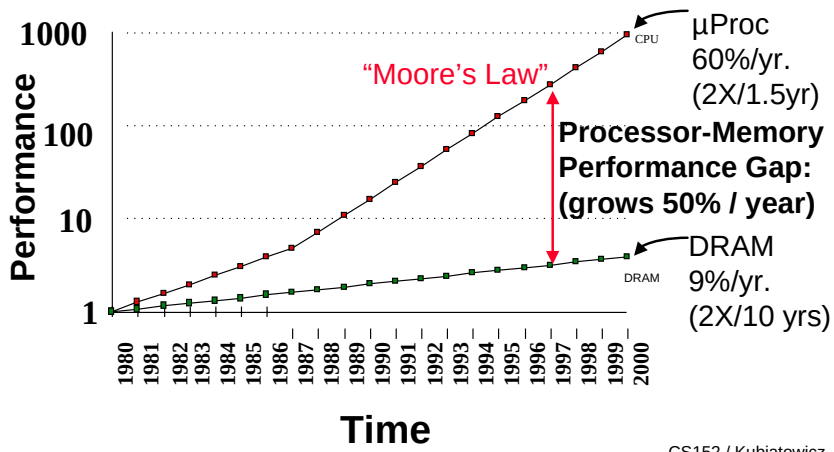
April 12, 1999

John Kubiatowicz (http.cs.berkeley.edu/~kubitron)

lecture slides: http://www-inst.eecs.berkeley.edu/~cs152/

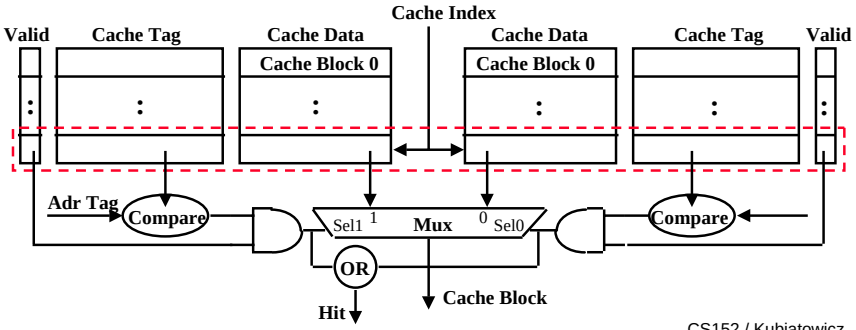
Recap: Who Cares About the Memory Hierarchy?

Processor-DRAM Memory Gap (latency)



Recap: Set Associative Cache

- **N-way set associative**: N entries for each Cache Index
  - N direct mapped caches operates in parallel
- Example: Two-way set associative cache
  - Cache Index selects a "set" from the cache
  - The two tags in the set are compared to the input in parallel
  - Data is selected based on the tag result



Recap: Sources of cache misses – the three Cs (+1)

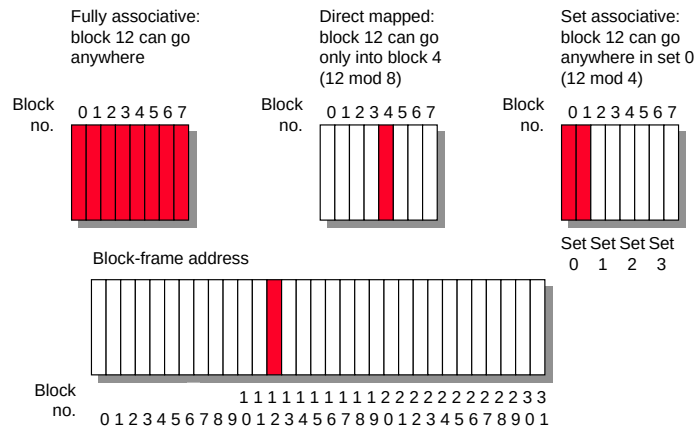
- **Compulsory** (cold start or process migration, first reference): first access to a block
  - "Cold" fact of life: not a whole lot you can do about it
  - Note: If you are going to run "billions" of instruction, Compulsory Misses are insignificant
- **Capacity**:
  - Cache cannot contain all blocks access by the program
  - Solution: increase cache size
- **Conflict** (collision):
  - Multiple memory locations mapped to the same cache location
  - Solution 1: increase cache size
  - Solution 2: increase associativity
- **Coherence** (Invalidation): other process (e.g., I/O) updates memory

Recap: Four Questions for Caches and Memory Hierarchy

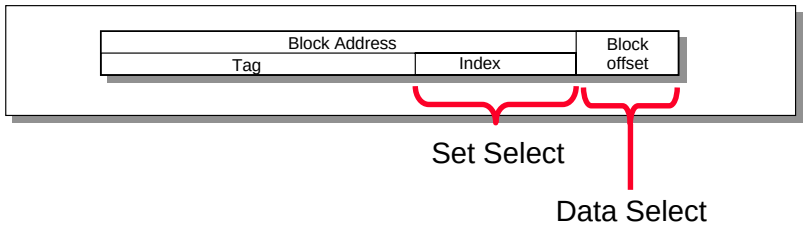
- Q1: Where can a block be placed in the upper level? *(Block placement)*
- Q2: How is a block found if it is in the upper level? *(Block identification)*
- Q3: Which block should be replaced on a miss? *(Block replacement)*
- Q4: What happens on a write? *(Write strategy)*

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



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



- Direct indexing (using index and block offset), tag compares, or combination
- Increasing associativity shrinks index, expands tag

Q3: Which block should be replaced on a miss?

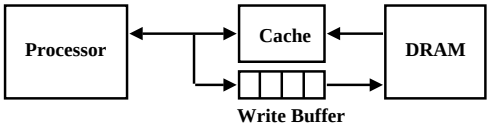
- Easy for Direct Mapped
- Set Associative or Fully Associative:
  - Random
  - LRU (Least Recently Used)

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?

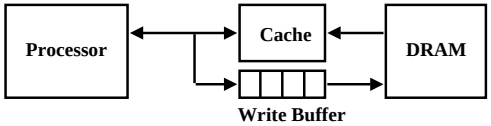
- ***Write through***—The information is written to both the block in the cache and to the block in the lower-level memory.
- ***Write back***—The information is written only to the block in the cache. The modified cache block is written to main memory only when it is replaced.
  - is block clean or dirty?
- 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

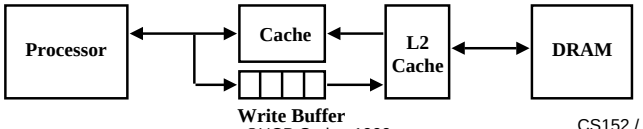


- A Write Buffer is needed between the Cache and Memory
  - Processor: writes data into the cache and the write buffer
  - Memory controller: write contents of the buffer to memory
- Write buffer is just a FIFO:
  - Typical number of entries: 4
  - Works fine if: Store frequency (w.r.t. time) << 1 / DRAM write cycle
- Memory system designer's nightmare:
  - Store frequency (w.r.t. time) > 1 / DRAM write cycle
  - Write buffer saturation

Write Buffer Saturation

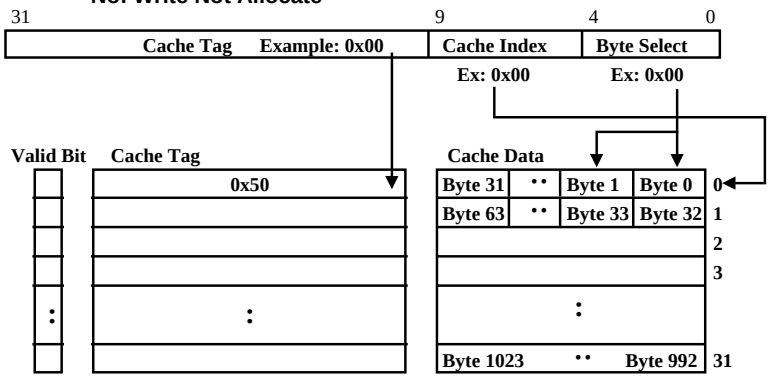


- Store frequency (w.r.t. time) > 1 / DRAM write cycle
  - If this condition exist for a long period of time (CPU cycle time too quick and/or too many store instructions in a row):
    - Store buffer will overflow no matter how big you make it
    - The CPU Cycle Time <= DRAM Write Cycle Time
- Solution for write buffer saturation:
  - Use a write back cache
  - Install a second level (L2) cache: (does this always work?)



Write-miss Policy: Write Allocate versus Not Allocate

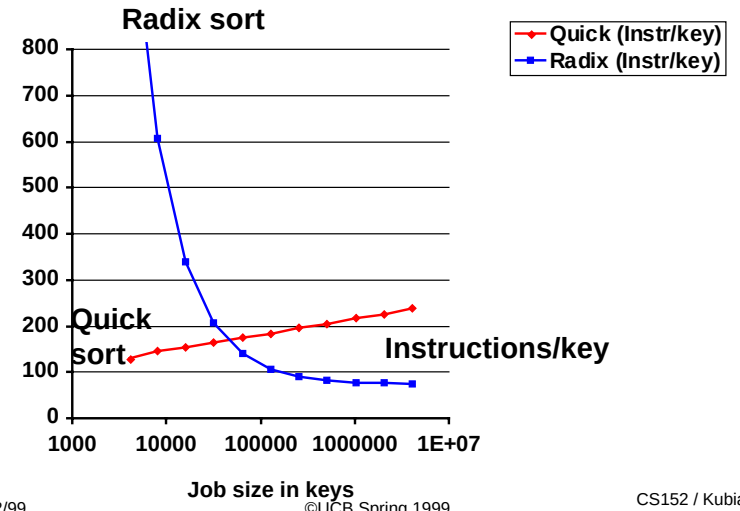
- Assume: a 16-bit write to memory location 0x0 and causes a miss
  - Do we read in the block?
    - Yes: Write Allocate
    - No: Write Not Allocate



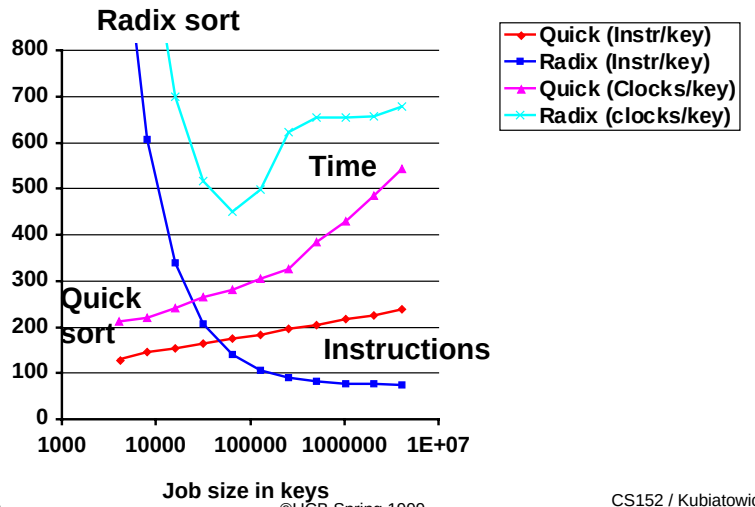
Impact of Memory Hierarchy on Algorithms

- Today CPU time is a function of (ops, cache misses) vs. just f(ops): What does this mean to Compilers, Data structures, Algorithms?
- "The Influence of Caches on the Performance of Sorting" by A. LaMarca and R.E. Ladner. *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, January, 1997, 370-379.
- Quicksort: fastest comparison based sorting algorithm when all keys fit in memory
- Radix sort: also called "linear time" sort because for keys of fixed length and fixed radix a constant number of passes over the data is sufficient independent of the number of keys
- For Alphastation 250, 32 byte blocks, direct mapped L2 2MB cache, 8 byte keys, from 4000 to 4000000

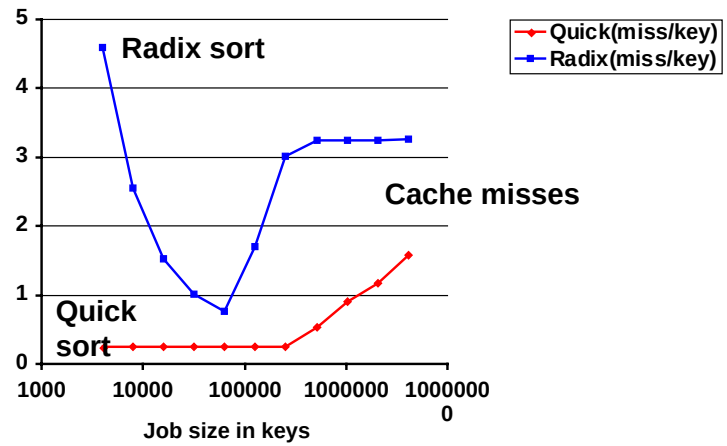
Quicksort vs. Radix as vary number keys: Instructions



Quicksort vs. Radix as vary number keys: Instrs & Time



Quicksort vs. Radix as vary number keys: Cache misses

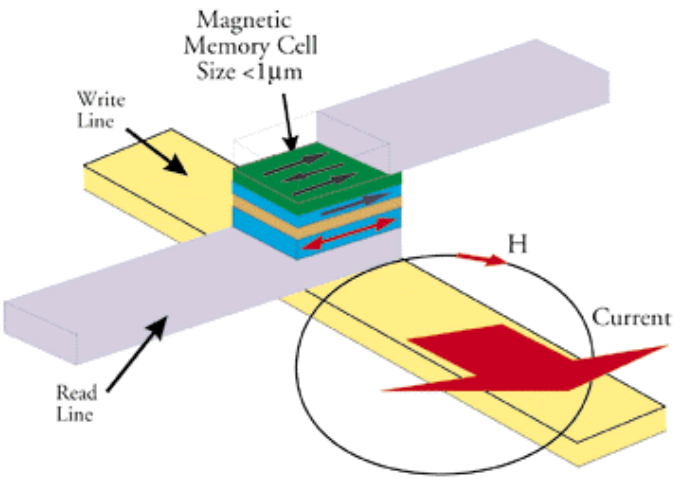


What is proper approach to fast algorithms?

Administrivia

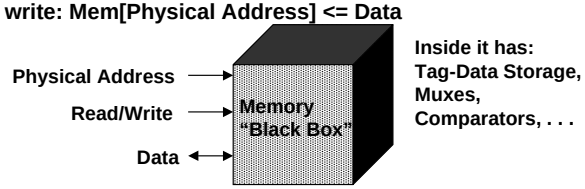
- Tomorrow's sections: in Evans
  - Will discuss critical path with respect to branchs
- Second midterm coming up:
  - Wed, April 21 in 277 Cory
  - Review session Sunday, 4/18 at 7:00 in 306 Soda
- Computers in the news: IBM breakthrough!
  Tunneling Magnetic Junction RAM (TMJ-RAM)
  - Speed of SRAM, density of DRAM, non-volatile (no refresh)
  - New field called "Spintronics": combination of quantum spin and electronics
  - Same technology used in high-density disk-drives

Structure of Tunneling Magnetic Junction

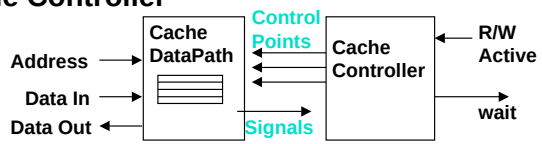


How Do you Design a Cache?

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

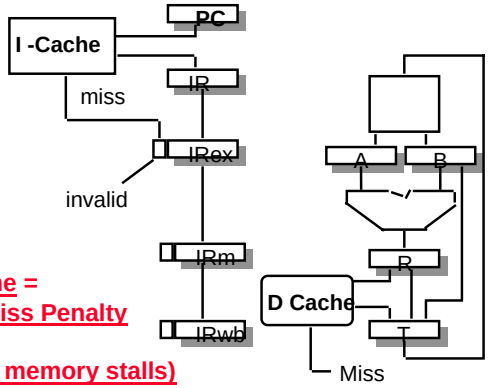


- Determine the internal register transfers
- Design the Datapath
- Design the Cache Controller



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)

What happens on a Cache miss?

° For in-order pipeline, 2 options:

- Freeze pipeline in Mem stage (popular early on: Sparc, R4000)

```
IF  ID  EX  Mem stall stall stall … stall Mem   Wr
    IF  ID  EX  stall stall stall … stall stall Ex Wr
```

- Use Full/Empty bits in registers + MSHR queue
  - MSHR = "Miss Status/Handler Registers" (Kroft)
    Each entry in this queue keeps track of status of outstanding memory requests to one complete memory line.
    - Per cache-line: keep info about memory address.
    - For each word: register (if any) that is waiting for result.
    - Used to "merge" multiple requests to one memory line
  - New load creates MSHR entry and sets destination register to "Empty". Load is "released" from pipeline.
  - Attempt to use register before result returns causes instruction to block in decode stage.
  - Limited "out-of-order" execution with respect to loads.
    Popular with in-order superscalar architectures.

° Out-of-order pipelines already have this functionality built in… (load queues, etc).

Improving Cache Performance: 3 general options

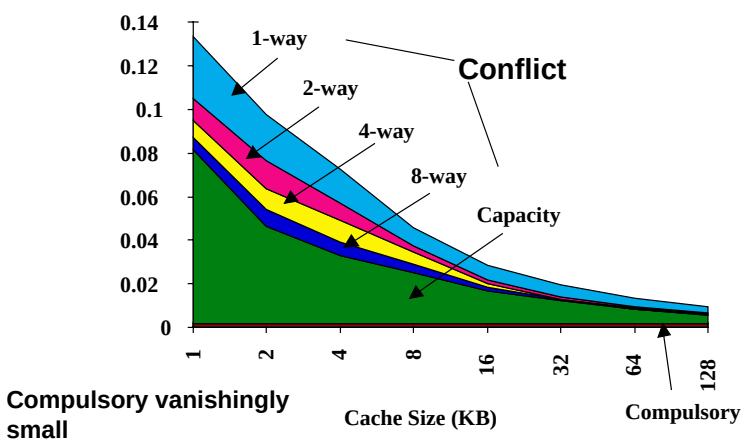
Time = IC x CT x (ideal CPI + memory stalls)

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

(Hit Rate x Hit Time) + (Miss Rate x Miss Time)

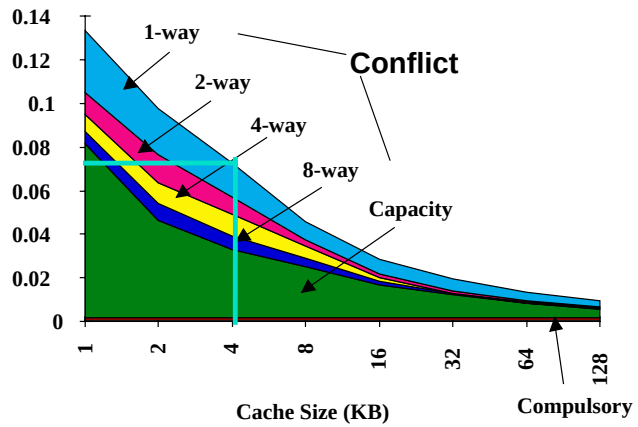
1. Reduce the miss rate,
2. Reduce the miss penalty, or
3. Reduce the time to hit in the cache.

3Cs Absolute Miss Rate (SPEC92)

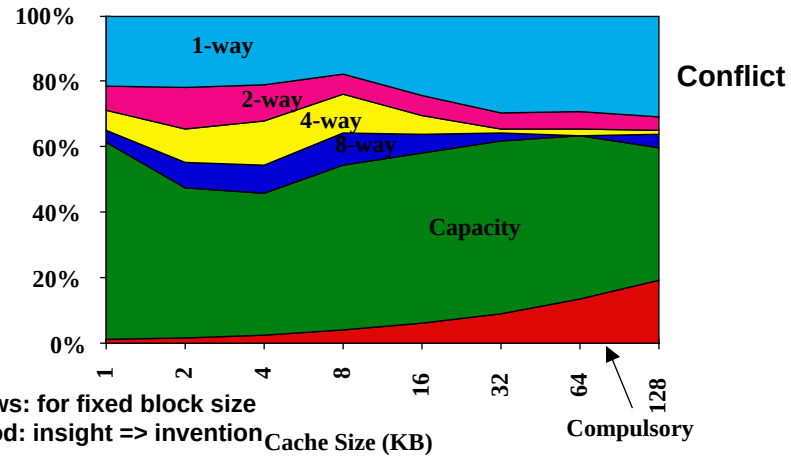


2:1 Cache Rule

miss rate 1-way associative cache size X
= miss rate 2-way associative cache size X/2

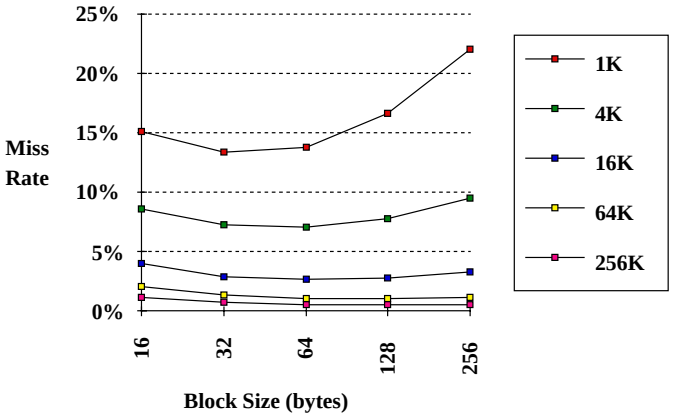


3Cs Relative Miss Rate



Flaws: for fixed block size
Good: insight => invention

1. Reduce Misses via Larger Block Size



2. Reduce Misses via Higher Associativity

- 2:1 Cache Rule:
  - Miss Rate DM cache size N - Miss Rate 2-way cache size N/2
- Beware: Execution time is only final measure!
  - Will Clock Cycle time increase?
  - Hill [1988] suggested hit time for 2-way vs. 1-way external cache +10%, internal + 2%

Example: Avg. Memory Access Time vs. Miss Rate

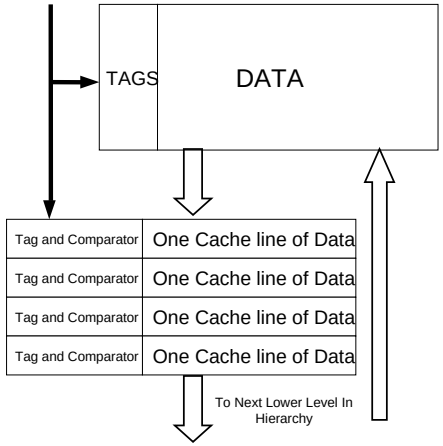
- Example: assume CCT = 1.10 for 2-way, 1.12 for 4-way, 1.14 for 8-way vs. CCT direct mapped

Cache Size	Associativity			
(KB)	1-way	2-way	4-way	8-way
1	2.33	2.15	2.07	2.01
2	1.98	1.86	1.76	1.68
4	1.72	1.67	1.61	1.53
8	1.46	1.48	1.47	1.43
16	1.29	1.32	1.32	1.32
32	1.20	1.24	1.25	1.27
64	1.14	1.20	1.21	1.23
128	1.10	1.17	1.18	1.20

(Red means A.M.A.T. not improved by more associativity)

3. Reducing Misses via a "Victim Cache"

- How to combine fast hit time of direct mapped yet still avoid conflict misses?
- Add buffer to place data discarded from cache
- Jouppi [1990]: 4-entry victim cache removed 20% to 95% of conflicts for a 4 KB direct mapped data cache
- Used in Alpha, HP machines



4. Reducing Misses by Hardware Prefetching

- E.g., Instruction Prefetching
  - Alpha 21064 fetches 2 blocks on a miss
  - Extra block placed in "stream buffer"
  - On miss check stream buffer
- Works with data blocks too:
  - Jouppi [1990] 1 data stream buffer got 25% misses from 4KB cache; 4 streams got 43%
  - Palacharla & Kessler [1994] for scientific programs for 8 streams got 50% to 70% of misses from 2 64KB, 4-way set associative caches
- Prefetching relies on having extra memory bandwidth that can be used without penalty

5. Reducing Misses by Software Prefetching Data

- Data Prefetch
  - Load data into register (HP PA-RISC loads)
  - Cache Prefetch: load into cache (MIPS IV, PowerPC, SPARC v. 9)
  - Special prefetching instructions cannot cause faults; a form of speculative execution
- Issuing Prefetch Instructions takes time
  - Is cost of prefetch issues < savings in reduced misses?
  - Higher superscalar reduces difficulty of issue bandwidth

6. Reducing Misses by Compiler Optimizations

- McFarling [1989] reduced caches misses by 75% on 8KB direct mapped cache, 4 byte blocks in software
- Instructions
  - Reorder procedures in memory so as to reduce conflict misses
  - Profiling to look at conflicts(using tools they developed)
- Data
  - *Merging Arrays*: improve spatial locality by single array of compound elements vs. 2 arrays
  - *Loop Interchange*: change nesting of loops to access data in order stored in memory
  - *Loop Fusion*: Combine 2 independent loops that have same looping and some variables overlap
  - *Blocking*: Improve temporal locality by accessing "blocks" of data repeatedly vs. going down whole columns or rows

Review: Improving Cache Performance

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

0. Reducing Penalty: Faster DRAM / Interface

- New DRAM Technologies
  - RAMBUS - same initial latency, but much higher bandwidth
  - Synchronous DRAM
  - TMJ-RAM from IBM??
  - Merged DRAM/Logic - IRAM project here at berkeley
- Better BUS interfaces
- CRAY Technique: only use SRAM

1. Reducing Penalty: Read Priority over Write on Miss

- Write through with write buffers offer RAW conflicts with main memory reads on cache misses
- If simply wait for write buffer to empty, might increase read miss penalty (old MIPS 1000 by 50%)
- Check write buffer contents before read; if no conflicts, let the memory access continue
- Write Back?
  - Read miss replacing dirty block
  - Normal: Write dirty block to memory, and then do the read
  - Instead copy the dirty block to a write buffer, then do the read, and then do the write
  - CPU stall less since restarts as soon as do read

2. Reduce Penalty: Early Restart and Critical Word First

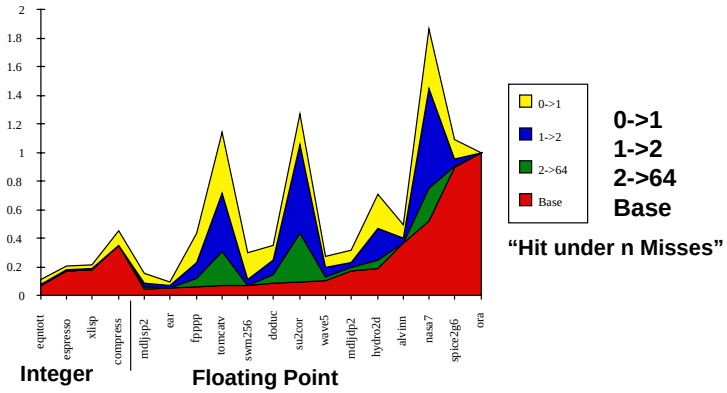
- Don't wait for full block to be loaded before restarting CPU
  - *Early restart*—As soon as the requested word of the block ar rives, send it to the CPU and let the CPU continue execution
  - *Critical Word First*—Request the missed word first from memory and send it to the CPU as soon as it arrives; let the CPU continue execution while filling the rest of the words in the block. Also called *wrapped fetch* and *requested word first*
- Generally useful only in large blocks,
- Spatial locality a problem; tend to want next sequential word, so not clear if benefit by early restart

block

3. Reduce Penalty: Non-blocking Caches

- *Non-blocking cache* or *lockup-free cache* allow data cache to continue to supply cache hits during a miss
  - requires F/E bits on registers or out-of-order execution
  - requires multi-bank memories
- "*hit under miss*" reduces the effective miss penalty by working during miss vs. ignoring CPU requests
- "*hit under multiple miss*" or "*miss under miss*" may further lower the effective miss penalty by overlapping multiple misses
  - Significantly increases the complexity of the cache controller as there can be multiple outstanding memory accesses
  - Requires muliple memory banks (otherwise cannot support)
  - Penium Pro allows 4 outstanding memory misses

Value of Hit Under Miss for SPEC



- FP programs on average: AMAT= 0.68 -> 0.52 -> 0.34 -> 0.26
- Int programs on average: AMAT= 0.24 -> 0.20 -> 0.19 -> 0.19
- 8 KB Data Cache, Direct Mapped, 32B block, 16 cycle miss

4. Reduce Penalty: Second-Level Cache

- **L2 Equations**

$$\text{AMAT} = \text{Hit Time}_{L1} + \text{Miss Rate}_{L1} \times \text{Miss Penalty}_{L1}$$

$$\text{Miss Penalty}_{L1} = \text{Hit Time}_{L2} + \text{Miss Rate}_{L2} \times \text{Miss Penalty}_{L2}$$

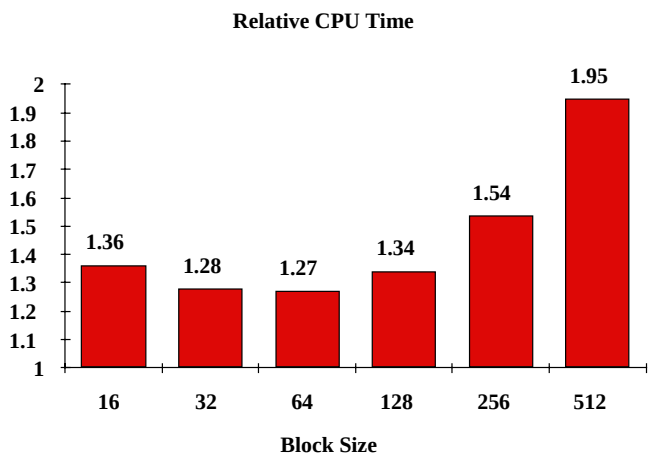
$$\text{AMAT} = \text{Hit Time}_{L1} + \text{Miss Rate}_{L1} \times (\text{Hit Time}_{L2} + \text{Miss Rate}_{L2} \times \text{Miss Penalty}_{L2})$$

- **Definitions:**
  - *Local miss rate*— misses in this cache divided by the total number of memory accesses *to this cache* (Miss rate_{L2})
  - *Global miss rate*—misses in this cache divided by the total number of memory accesses *generated by the CPU* ($\text{Miss Rate}_{L1} \times \text{Miss Rate}_{L2}$)
  - Global Miss Rate is what matters

Reducing Misses: which apply to L2 Cache?

- **Reducing Miss Rate**
  1. Reduce Misses via Larger Block Size
  2. Reduce Conflict Misses via Higher Associativity
  3. Reducing Conflict Misses via Victim Cache
  4. Reducing Misses by HW Prefetching Instr, Data
  5. Reducing Misses by SW Prefetching Data
  6. Reducing Capacity/Conf. Misses by Compiler Optimizations

L2 cache block size & A.M.A.T.



° 32KB L1, 8 byte path to memory

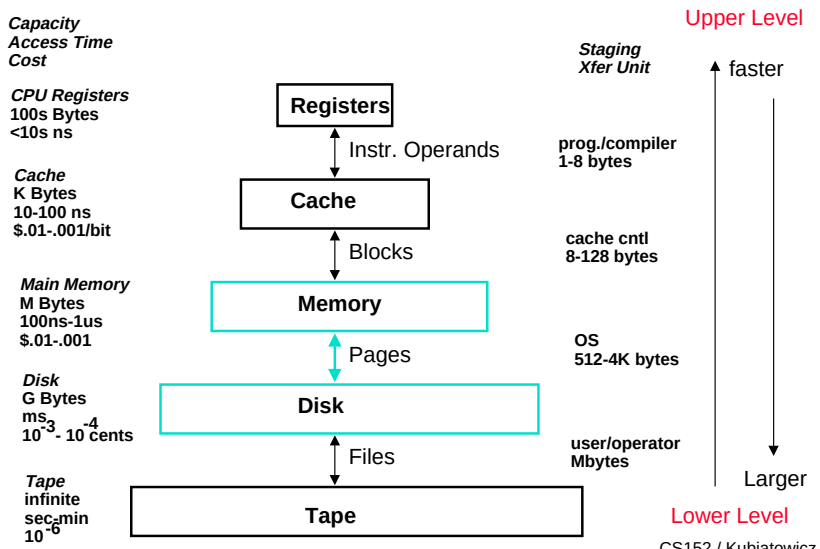
Reducing Miss Penalty Summary

$$CPUtime = IC \times \left(CPI_{Execution} + \frac{Memory\ \ accesses}{Instruction} \times \textbf{Miss rate} \times Miss\ \ penalty \right) \times Clock\ \ cycle\ \ time$$

° **Five techniques**
- Faster Main Memory
- Read priority over write on miss
- Early Restart and Critical Word First on miss
- Non-blocking Caches (Hit under Miss, Miss under Miss)
- Second Level Cache

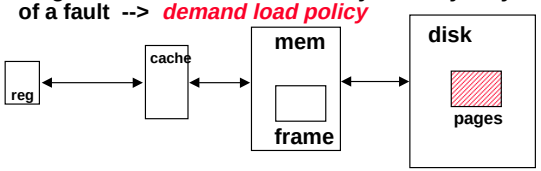
° **Can be applied recursively to Multilevel Caches**
- Danger is that time to DRAM will grow with multiple levels in between
- First attempts at L2 caches can make things worse, since increased worst case is worse

Recall: Levels of the Memory Hierarchy



Basic Issues in Virtual Memory System Design

- size of information blocks that are transferred from secondary to main storage (M)
- block of information brought into M, and M is full, then some region of M must be released to make room for the new block --> *replacement policy*
- which region of M is to hold the new block --> *placement policy*
- missing item fetched from secondary memory only on the occurrence of a fault --> *demand load policy*



Paging Organization

virtual and physical address space partitioned into blocks of equal size

page frames

pages

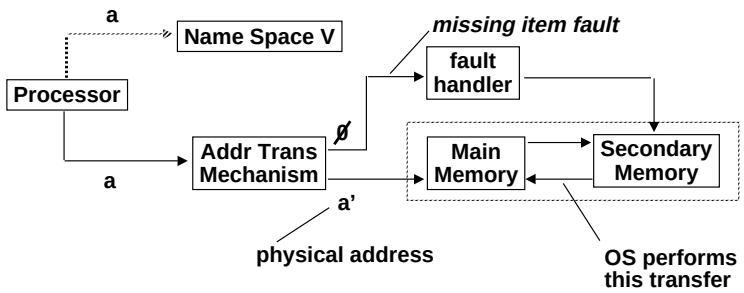
Address Map

V = {0, 1, . . . , n - 1} virtual address space
M = {0, 1, . . . , m - 1} physical address space n > m

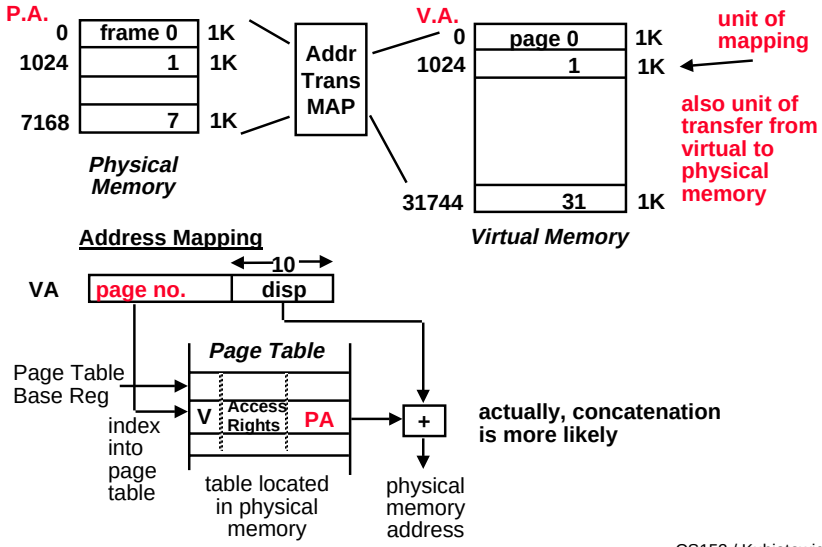
MAP: V --> M U {ø} address mapping function

MAP(a) = a' if data at virtual address a is present in physical address a' and a' in M

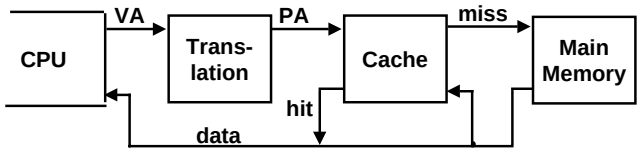
= ø if data at virtual address a is not present in M



Paging Organization



Virtual Address and a Cache



It takes an extra memory access to translate VA to PA

This makes cache access very expensive, and this is the "innermost loop" that you want to go as fast as possible

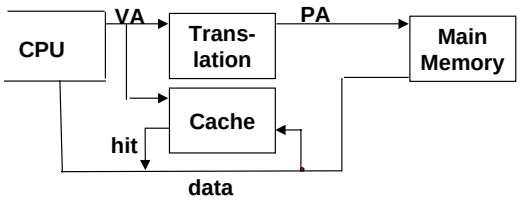
ASIDE: Why access cache with PA at all? VA caches have a problem!
synonym / alias problem: two different virtual addresses map to same physical address => two different cache entries holding data for the same physical address!

for update: must update all cache entries with same physical address or memory becomes inconsistent

determining this requires significant hardware, essentially an associative lookup on the physical address tags to see if you have multiple hits; or

software enforced alias boundary: same lsb of VA &PA > cache size

Virtually Addressed Cache



Only require address translation on cache miss!

synonym problem: two different virtual addresses map to same physical address => two different cache entries holding data for the same physical address!

nightmare for update: must update all cache entries with same physical address or memory becomes inconsistent

determining this requires significant hardware, essentially an associative lookup on the physical address tags to see if you have multiple hits.
(usually disallowed by fiat)

Reducing Translation Time

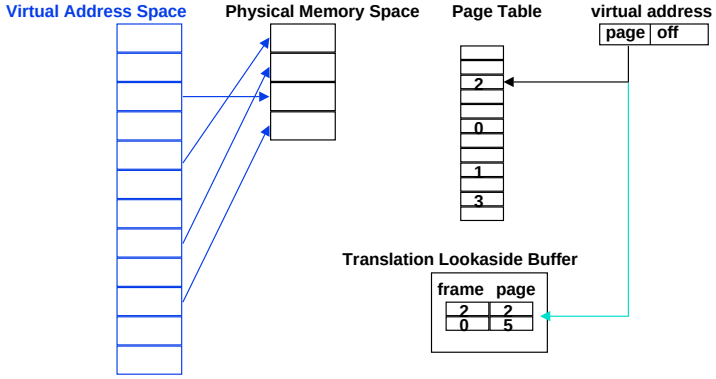
Machines with TLBs go one step further to reduce # cycles/cache access

They overlap the cache access with the TLB access

Works because high order bits of the VA are used to look in the TLB
while low order bits are used as index into cache

Making address translation practical: TLB

- Virtual memory => memory acts like a cache for the disk
- Page table maps virtual page numbers to physical frames
- Translation Look-aside Buffer (TLB) is a cache of recent translations



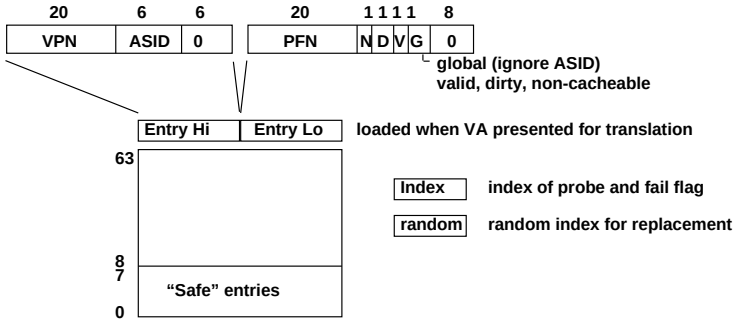
TLBs

A way to speed up translation is to use a special cache of recently used page table entries -- this has many names, but the most frequently used is *Translation Lookaside Buffer* or *TLB*

Virtual Address	Physical Address	Dirty	Ref	Valid	Access

TLB access time comparable to cache access time
(much less than main memory access time)

R3000 TLB & CP0 (MMU)



Optimal Page Size

° Mimimize wasted storage

— small page minimizes internal fragmentation

— small page increase size of page table

° Minmize transfer time

— large pages (multiple disk sectors) amortize access cost

— sometimes transfer unnecessary info

— sometimes prefetch useful data

— sometimes discards useless data early

General trend toward larger pages because

— big cheap RAM

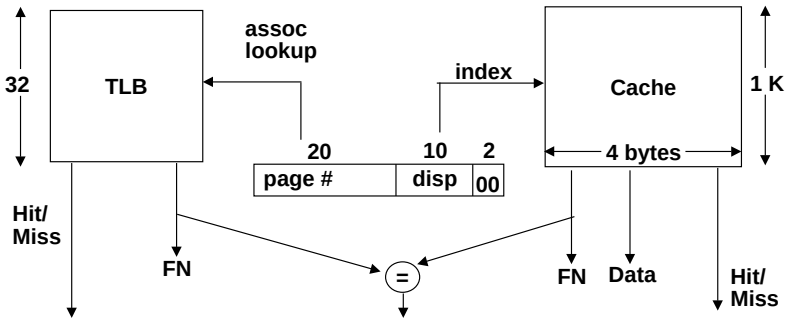
— increasing mem / disk performance gap

— larger address spaces

Overlapped TLB & Cache Access

° So far TLB access is serial with cache access

- can we do it in parallel?
- only if we are careful in the cache organization!



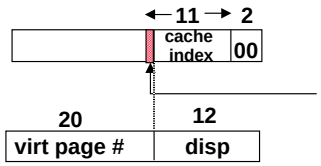
What if cache size is increased to 8KB?

Problems With Overlapped TLB Access

Overlapped access only works as long as the address bits used to index into the cache *do not change* as the result of VA translation

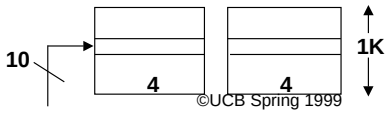
This usually limits things to small caches, large page sizes, or high n-way set associative caches if you want a large cache

Example: suppose everything the same except that the cache is increased to 8 K bytes instead of 4 K:



This bit is changed by VA translation, but is needed for cache lookup

Solutions:
go to 8K byte page sizes;
go to 2 way set associative cache; or
SW guarantee VA[13]=PA[13]



2 way set assoc cache

Page Fault: What happens when you miss?

° Not talking about TLB miss

- TLB is HWs attempt to make page table lookup fast (on average)

° Page fault means that page is not resident in memory

° Hardware must detect situation

° Hardware cannot remedy the situation

° Therefore, hardware must trap to the operating system so that it can remedy the situation

- pick a page to discard (possibly writing it to disk)
- load the page in from disk
- update the page table
- resume to program so HW will retry and succeed!

° What is in the page fault handler?

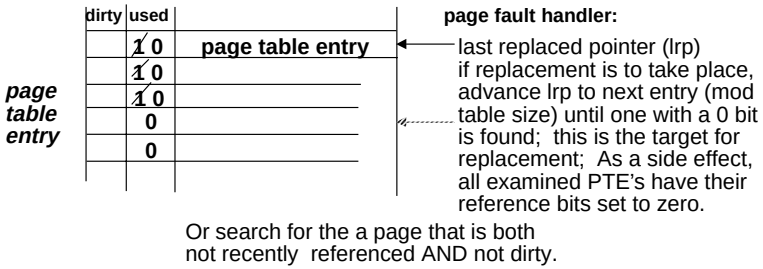
- see CS162

° What can HW do to help it do a good job?

Page Replacement: Not Recently Used (1-bit LRU, Clock)

Associated with each page is a reference flag such that
ref flag = 1 if the page has been referenced in recent past
= 0 otherwise

-- if replacement is necessary, choose any page frame such that its reference bit is 0. This is a page that has not been referenced in the recent past



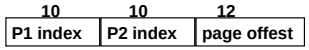
Or search for the a page that is both not recently referenced AND not dirty.

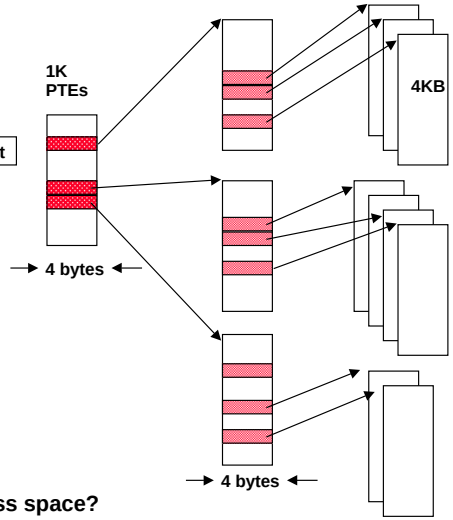
Architecture part: support dirty and used bits in the page table
=> may need to update PTE on any instruction fetch, load, store
How does TLB affect this design problem? Software TLB miss?

Large Address Spaces

Two-level Page Tables

32-bit address:





- 2 GB virtual address space
- 4 MB of PTE2
  - paged, holes
- 4 KB of PTE1

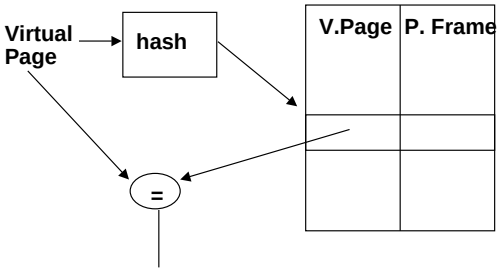
What about a 48-64 bit address space?

Inverted Page Tables

IBM System 38 (AS400) implements 64-bit addresses.

48 bits translated

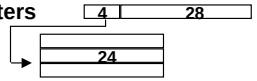
start of object contains a 12-bit tag



=> TLBs or virtually addressed caches are critical

Survey

- **R4000**
  - 32 bit virtual, 36 bit physical
  - variable page size (4KB to 16 MB)
  - 48 entries mapping page pairs (128 bit)
- **MPC601 (32 bit implementation of 64 bit PowerPC arch)**
  - 52 bit virtual, 32 bit physical, 16 segment registers
  - 4KB page, 256MB segment
  - 4 entry instruction TLB
  - 256 entry, 2-way TLB (and variable sized block xlate)
  - overlapped lookup into 8-way 32KB L1 cache
  - hardware table search through hashed page tables



- **Alpha 21064**
  - arch is 64 bit virtual, implementation subset: 43, 47,51,55 bit
  - 8,16,32, or 64KB pages (3 level page table)
  - 12 entry ITLB, 32 entry DTLB
  - 43 bit virtual, 28 bit physical octword address

Why virtual memory?

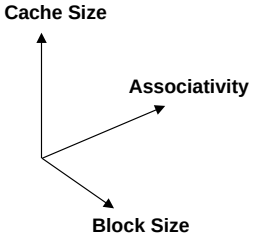
- *Generality*
  - ability to run programs larger than size of physical memory
- *Storage management*
  - allocation/deallocation of variable sized blocks is costly and leads to (external) fragmentation
- *Protection*
  - regions of the address space can be R/O, Ex, . . .
- *Flexibility*
  - portions of a program can be placed anywhere, without relocation
- *Storage efficiency*
  - retain only most important portions of the program in memory
- *Concurrent I/O*
  - execute other processes while loading/dumping page
- *Expandability*
  - can leave room in virtual address space for objects to grow.
- *Performance*
  - Observe: impact of multiprogramming, impact of higher level languages

Summary #1/ 4:

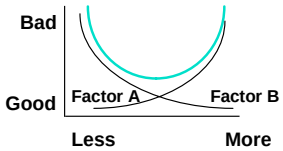
- The Principle of Locality:
  - Program likely to access a relatively small portion of the address space at any instant of time.
    - Temporal Locality: Locality in Time
    - Spatial Locality: Locality in Space
- Three (+1) Major Categories of Cache Misses:
  - Compulsory Misses: sad facts of life. Example: cold start misses.
  - Conflict Misses: increase cache size and/or associativity. Nightmare Scenario: ping pong effect!
  - Capacity Misses: increase cache size
  - Coherence Misses: Caused by external processors or I/O devices
- Cache Design Space
  - total size, block size, associativity
  - replacement policy
  - write-hit policy (write-through, write-back)
  - write-miss policy

Summary #2 / 4: The Cache Design Space

- Several interacting dimensions
  - cache size
  - block size
  - associativity
  - replacement policy
  - write-through vs write-back
  - write allocation



- The optimal choice is a compromise
  - depends on access characteristics
    - workload
    - use (I-cache, D-cache, TLB)
  - depends on technology / cost
- Simplicity often wins



Summary #3 / 4 : TLB, Virtual Memory

- Caches, TLBs, Virtual Memory all understood by examining how they deal with 4 questions: 1) Where can block be placed? 2) How is block found? 3) What block is replaced on miss? 4) How are writes handled?
- Page tables map virtual address to physical address
- TLBs are important for fast translation
- TLB misses are significant in processor performance: (funny times, as most systems can't access all of 2nd level cache without TLB misses!)

Summary #4 / 4: Memory Hierachy

- ° Virtual memory was controversial at the time: can SW automatically manage 64KB across many programs?
  - 1000X DRAM growth removed the controversy
- ° Today VM allows many processes to share single memory without having to swap all processes to disk; VM protection is more important than memory hierarchy
- ° Today CPU time is a function of (ops, cache misses) vs. just f(ops): What does this mean to Compilers, Data structures, Algorithms?