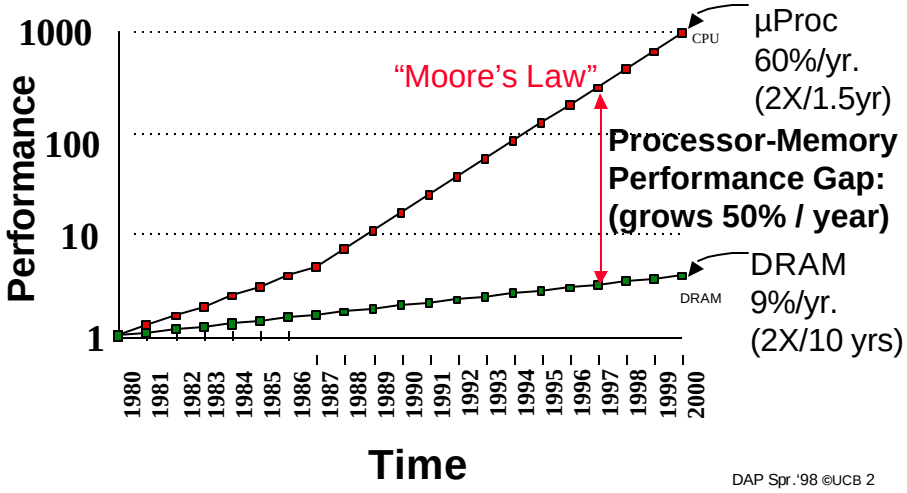
Caches

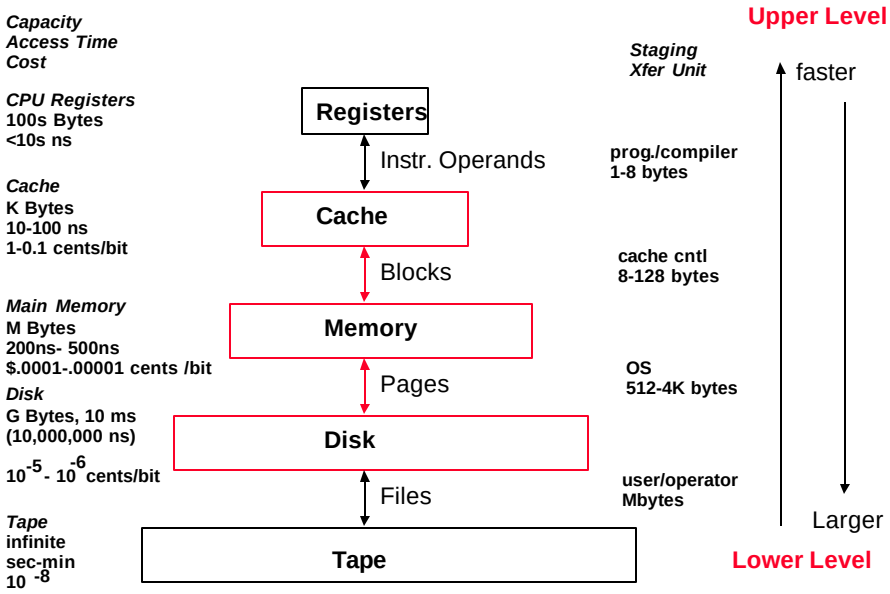
Prof. David A. Patterson
Computer Science 252
Spring 1998

Recap: Who Cares About the Memory Hierarchy?

Processor-DRAM Memory Gap (latency)



Levels of the Memory Hierarchy

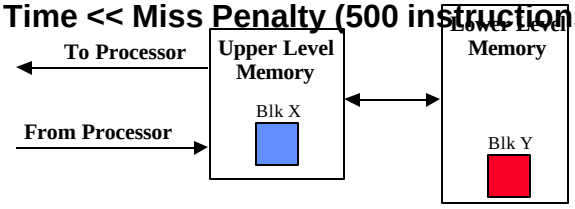


The Principle of Locality

- **The Principle of Locality:**
  - **Program access a relatively small portion of the address space at any instant of time.**
- **Two Different Types of Locality:**
  - **Temporal Locality (Locality in Time): If an item is referenced, it will tend to be referenced again soon (e.g., loops, reuse)**
  - **Spatial Locality (Locality in Space): If an item is referenced, items whose addresses are close by tend to be referenced soon (e.g., straightline code, array access)**
- **Last 15 years, HW relied on localilty for speed**

Memory Hierarchy: Terminology

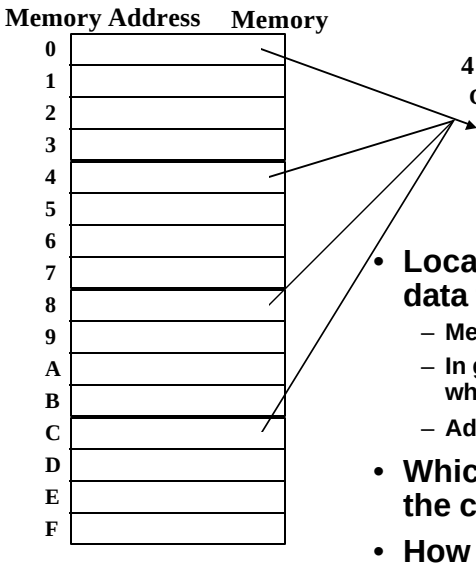
- Hit: data appears in some block in the upper level (example: Block X)
  - Hit Rate: the fraction of memory access found in the upper level
  - Hit Time: Time to access the upper level which consists of
    RAM access time + Time to determine hit/miss
- Miss: data needs to be retrieve from a block in the lower level (Block Y)
  - Miss Rate = 1 - (Hit Rate)
  - Miss Penalty: Time to replace a block in the upper level +
    Time to deliver the block the processor
- Hit Time << Miss Penalty (500 instructions on 21264!)



Cache Measures

- *Hit rate*: fraction found in that level
  - So high that usually talk about *Miss rate*
  - Miss rate fallacy: as MIPS to CPU performance, miss rate to average memory access time in memory
- Average memory-access time
  = Hit time + Miss rate x Miss penalty
  (ns or clocks)
- *Miss penalty*: time to replace a block from lower level, including time to replace in CPU
  - *access time*: time to lower level
    = f(latency to lower level)
  - *transfer time*: time to transfer block
    =f(BW between upper & lower levels)

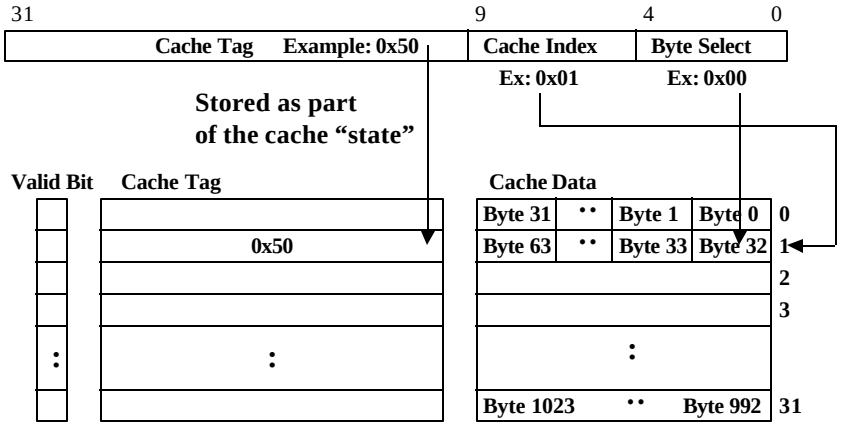
Simplest Cache: Direct Mapped



- Location 0 can be occupied by data from:
  - Memory location 0, 4, 8, ... etc.
  - In general: any memory location whose 2 LSBs of the address are 0s
  - Address<1:0> => cache index
- Which one should we place in the cache?
- How can we tell which one is in the cache?

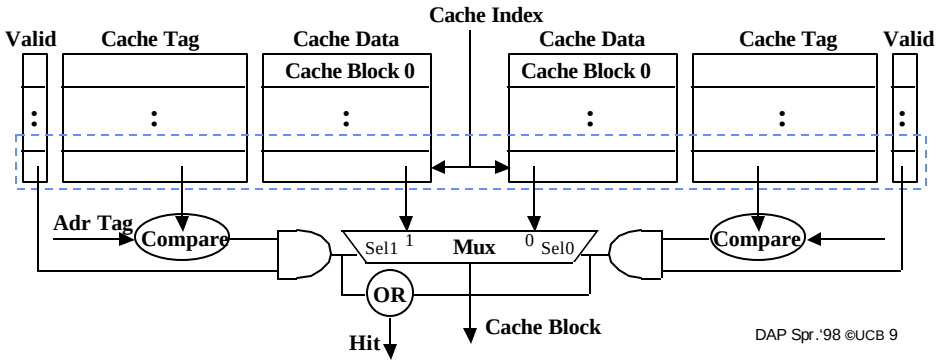
1 KB Direct Mapped Cache, 32B blocks

- 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)



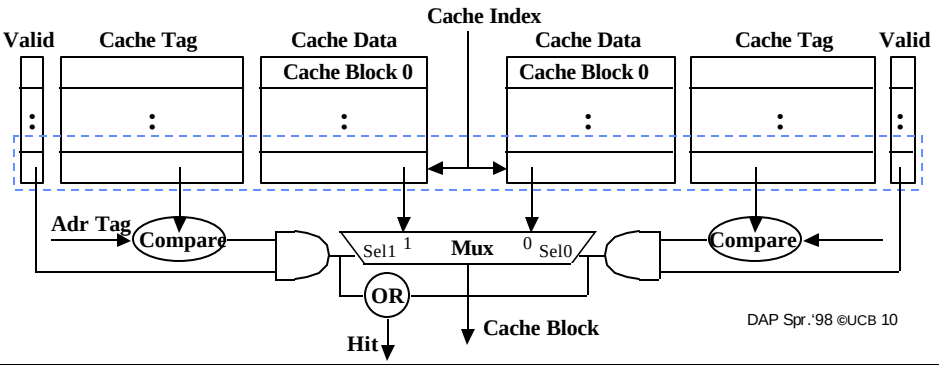
Two-way Set Associative Cache

- N-way set associative: N entries for each Cache Index
  - N direct mapped caches operates in parallel (N typically 2 to 4)
- Example: Two-way set associative cache
  - Cache Index selects a “set” from the cache
  - The two tags in the set are compared in parallel
  - Data is selected based on the tag result



Disadvantage of Set Associative Cache

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

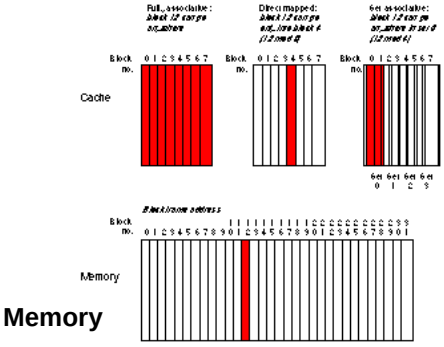


4 Questions for Memory Hierarchy

- **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?

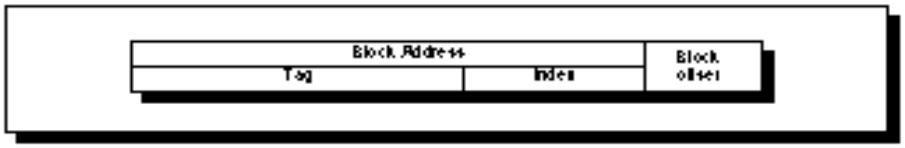
- **Block 12 placed in 8 block cache:**
  - **Fully associative, direct mapped, 2-way set associative**
  - **S.A. Mapping = Block Number Modulo Number Sets**



Memory

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?

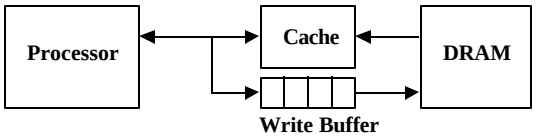
- Easy for Direct Mapped
- Set Associative or Fully Associative:
  - Random
  - LRU (Least Recently Used)

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?**
- **Pros and Cons of each?**
  - **WT: read misses cannot result in writes**
  - **WB: no repeated writes to same location**
- **WT always combined with write buffers so that don't wait for lower level memory**

Write Buffer for Write Through

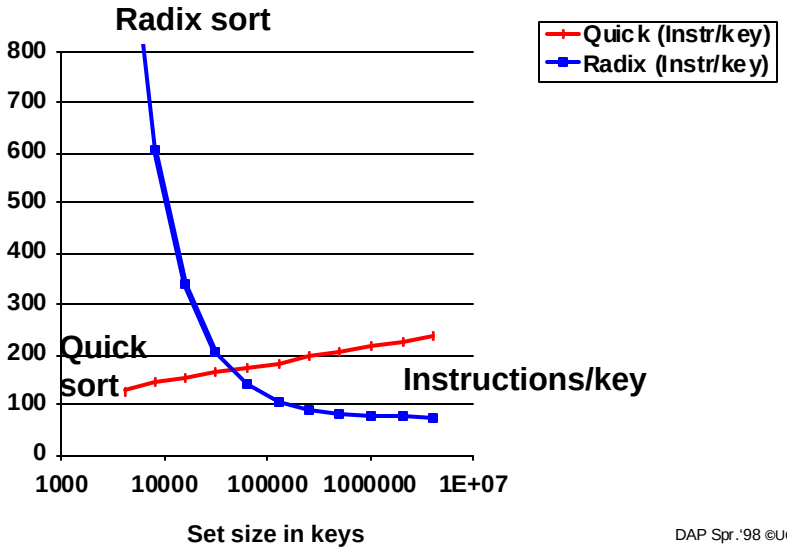


- **A Write Buffer is needed between the Cache and Memory**
  - **Processor: writes data into the cache and the write buffer**
  - **Memory controller: write contents of the buffer to memory**
- **Write buffer is just a FIFO:**
  - **Typical number of entries: 4**
  - **Works fine if: Store frequency (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**

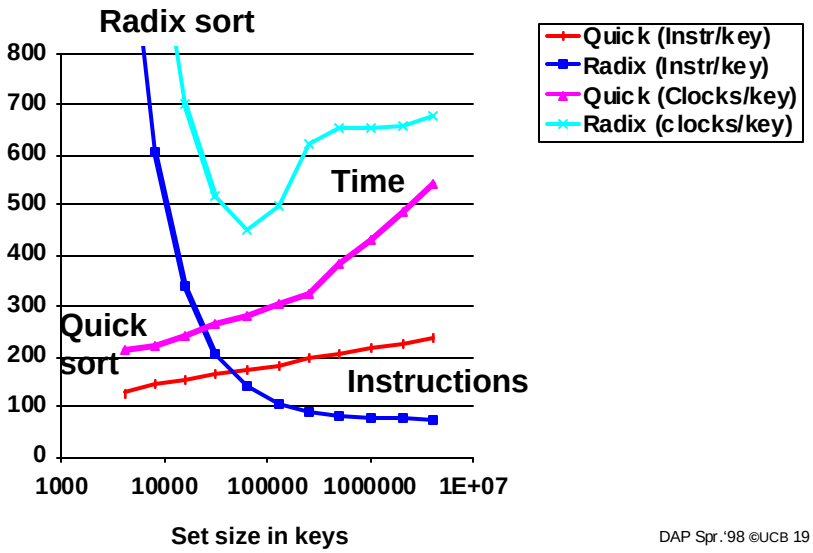
Impact of Memory Hierarchy on Algorithms

- 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

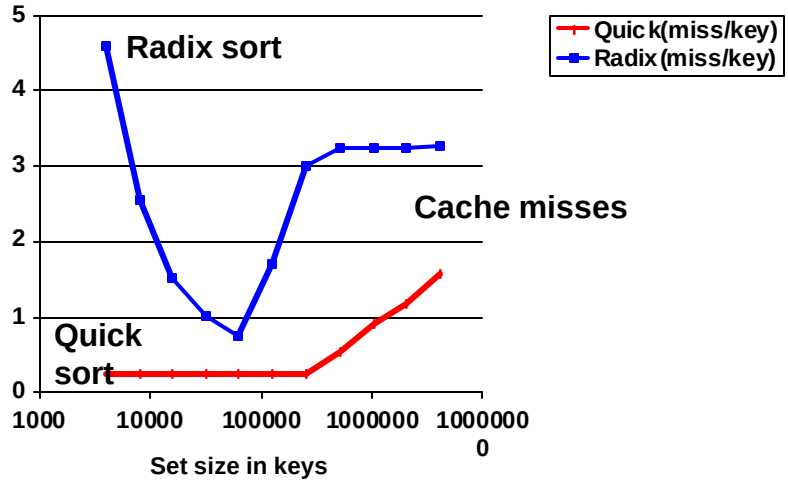
Quicksort vs. Radix as vary number keys: Instructions



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



Quicksort vs. Radix as vary number keys: Cache misses



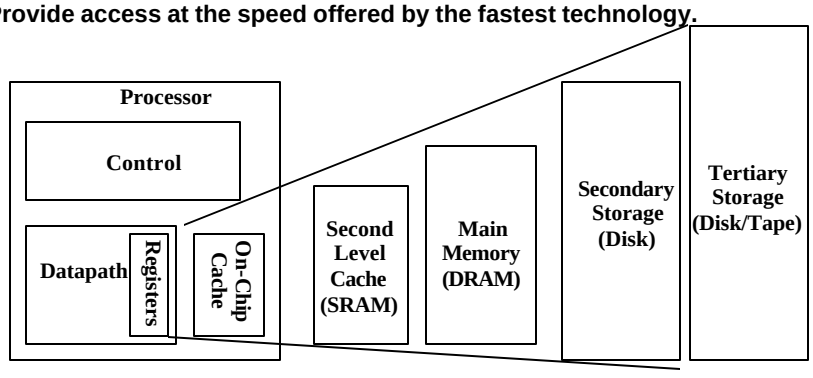
What is proper approach to fast algorithms?

5 minute Class Break

- 80 minutes straight is too long for me to lecture (12:40:00 – 2:00:00):
  - - 1 minute: review last time & motivate this lecture
  - - 20 minute lecture
  - - 3 minutes: discuss class manangement
  - - 25 minutes: lecture
  - 5 minutes: break
  - -25 minutes: lecture
  - -1 minute: summary of today's important topics

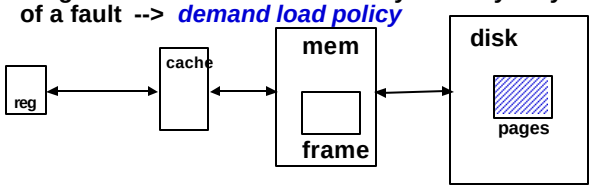
A Modern Memory Hierarchy

- 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.



Basic Issues in VM 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

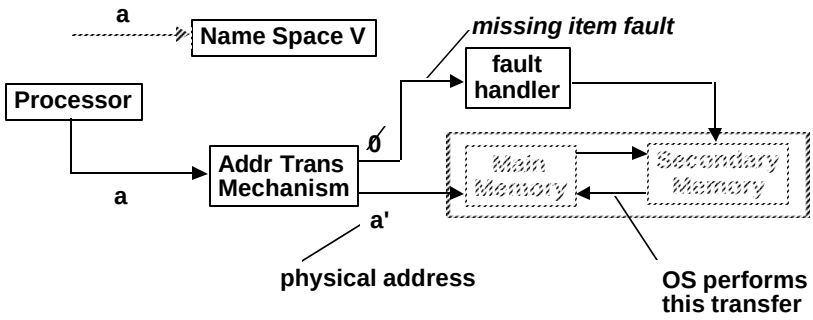
Address Map

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

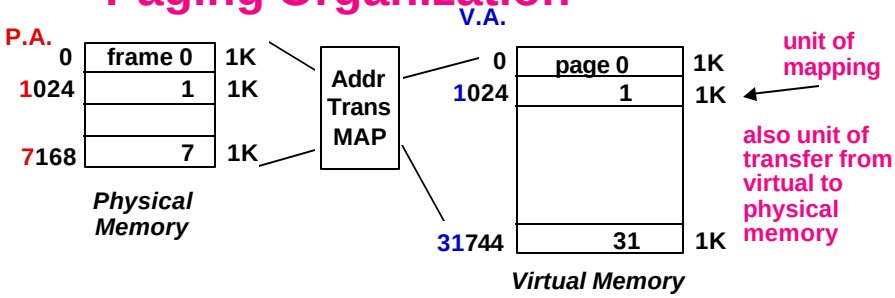
MAP: V --> M U {0} address mapping function

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

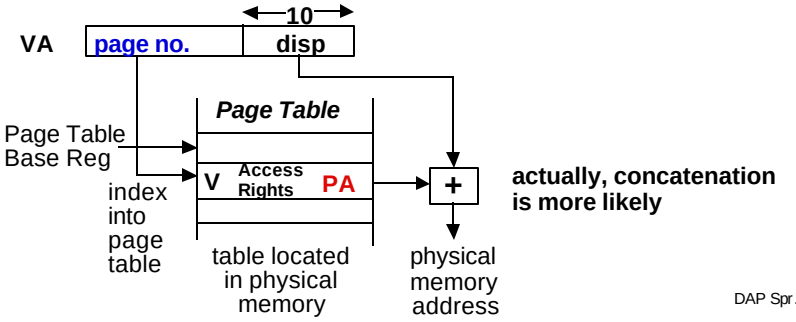
= 0 if data at virtual address a is not present in M



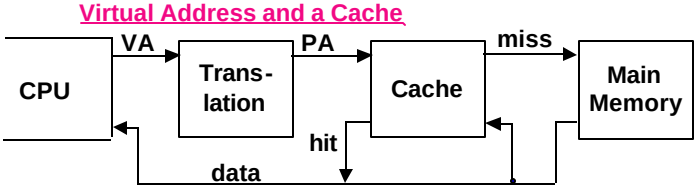
Paging Organization



Address Mapping



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

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

Really just a cache on the page table mappings

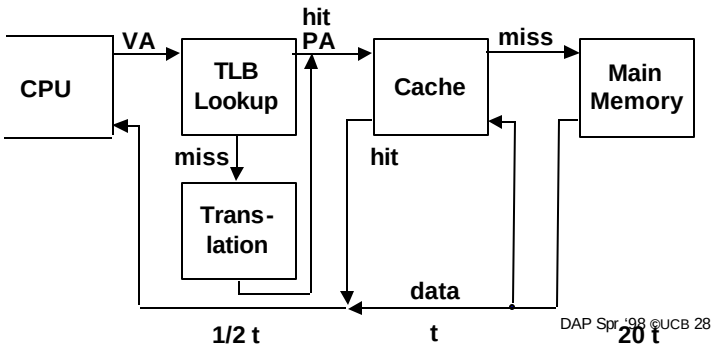
TLB access time comparable to cache access time (much less than main memory access time)

Translation Look-Aside Buffers

Just like any other cache, the TLB can be organized as fully associative, set associative, or direct mapped

TLBs are usually small, typically not more than 128 - 256 entries even on high end machines. This permits fully associative lookup on these machines. Most mid-range machines use small n-way set associative organizations.

Translation with a TLB



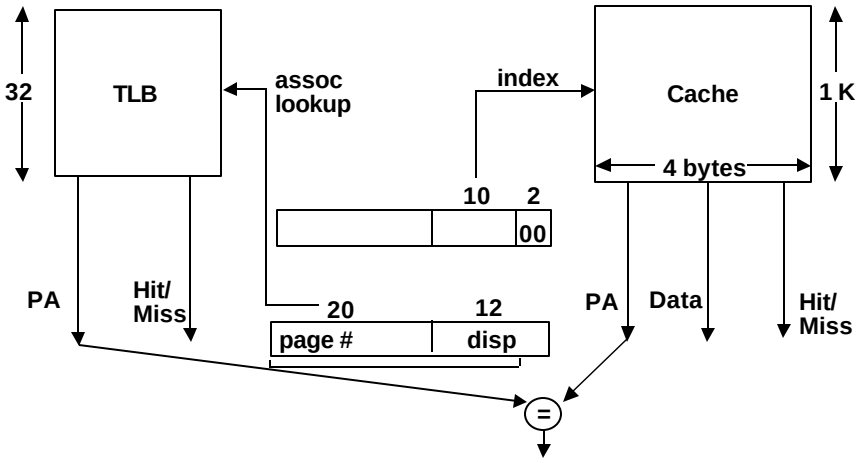
Reducing Translation Time

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

They overlap the cache access with the TLB access:

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

Overlapped Cache & TLB Access



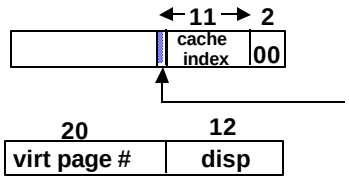
IF cache hit AND (cache tag = PA) then deliver data to CPU
ELSE IF [cache miss OR (cache tag = PA)] and TLB hit THEN
access memory with the PA from the TLB
ELSE do standard VA translation

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

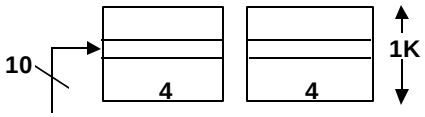
This usually limits things to small caches, large page sizes, or high n-way set associative caches if you want a large cache

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]



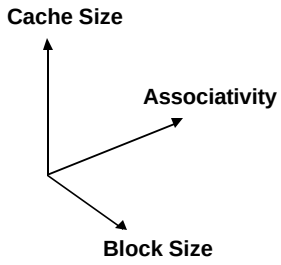
2 way set assoc cache

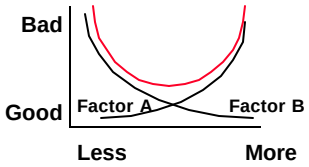
Summary #1/4:

- **The Principle of Locality:**
  - Program 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 Major Categories of Cache Misses:**
  - Compulsory Misses: sad facts of life. Example: cold start misses.
  - Capacity Misses: increase cache size
  - Conflict Misses: increase cache size and/or associativity. Nightmare Scenario: ping pong effect!
- **Write Policy:**
  - Write Through: needs a write buffer. Nightmare: WB saturation
  - Write Back: control can be complex

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 repalced 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; today 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?