

CSC2510 Computer Organization Caches and Virtual Memory

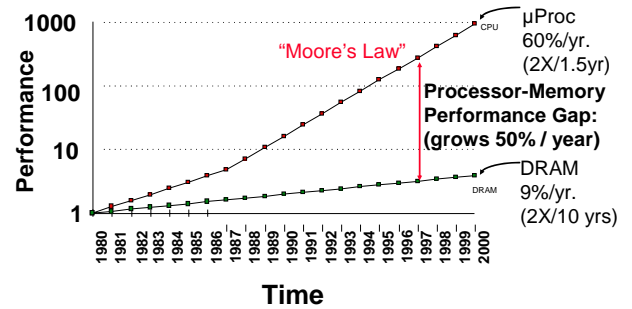
(Slides from "Computer Organization and Design",
Patterson and Hennessy)

1

DAP Fall97, © U.C.B.

Who Cares About the Memory Hierarchy?

Processor-DRAM Memory Gap (latency)



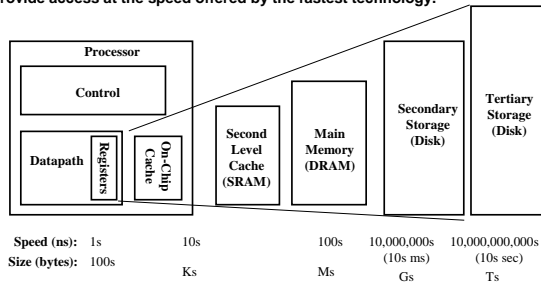
2

DAP Fall97, © U.C.B.

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.



3

DAP Fall97, © U.C.B.

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.

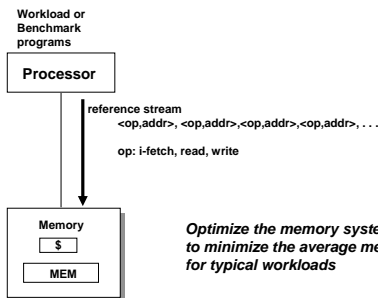
By taking advantage of the principle of locality:

- 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 but cheap and dense:
 - Good choice for presenting the user with a BIG memory system
- SRAM is fast but expensive and not very dense:
 - Good choice for providing the user FAST access time.

4

DAP Fall97, © U.C.B.

The Art of Memory System Design



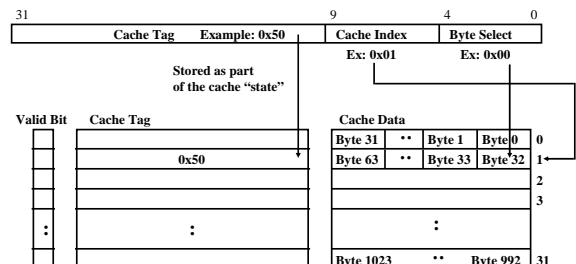
5

DAP Fall97, © U.C.B.

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 N bits are the Byte Select (Block Size = 2^N M)



6

DAP Fall97, © U.C.B.

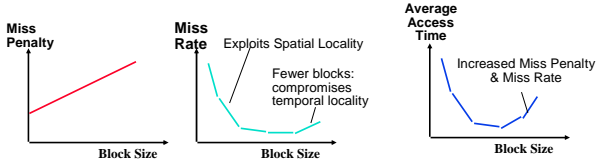
Block Size Tradeoff

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

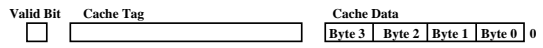
- Larger block size means larger miss penalty:
 - Takes longer time to fill up the block
- If block size is too big relative to cache size, miss rate will go up
 - Too few cache blocks

◦ In general, Average Access Time:

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



Extreme Example: single big line



◦ Cache Size = 4 bytes

Block Size = 4 bytes

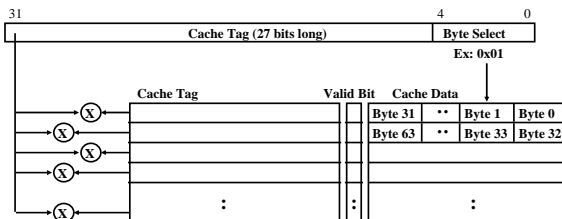
- Only ONE entry in the cache
- If an item is accessed, likely that it will be accessed again soon
 - But it is unlikely that it will be accessed again immediately!!!
 - The next access will likely to be a miss again
 - Continually loading data into the cache but discard (force out) them before they 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 = 2 B blocks, we need N 27-bit comparators

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



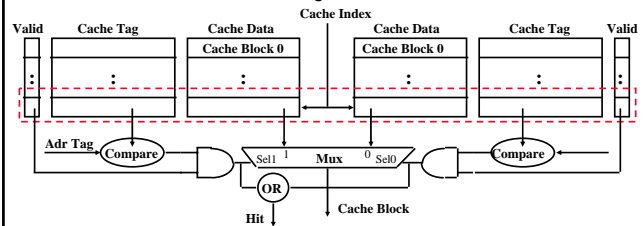
A Two-way Set Associative Cache

◦ 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 in parallel
- Data is selected based on the tag result



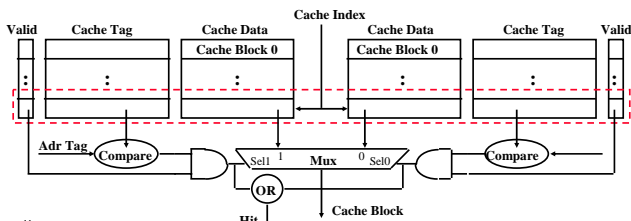
Disadvantage of Set Associative Cache

◦ 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

- "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

◦ 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 access by the program
- Solution: increase cache size

◦ Invalidation: other process (e.g., I/O) updates memory

Source of Cache Misses Quiz

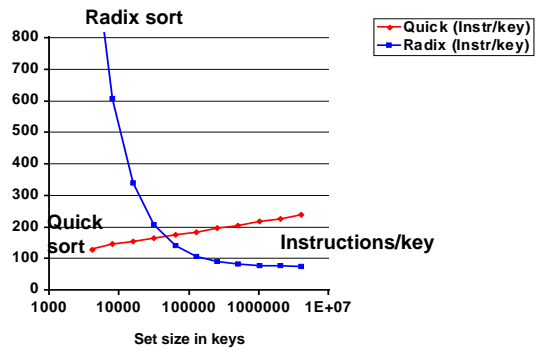
	Direct Mapped	N-way Set Associative	Fully Associative
Cache Size: Small, Medium, Big?			
Compulsory Miss:			
Conflict Miss			
Capacity Miss			
Invalidation Miss			

Choices: Zero, Low, Medium, High, Same

13

DAP Fa97, © U.C.B

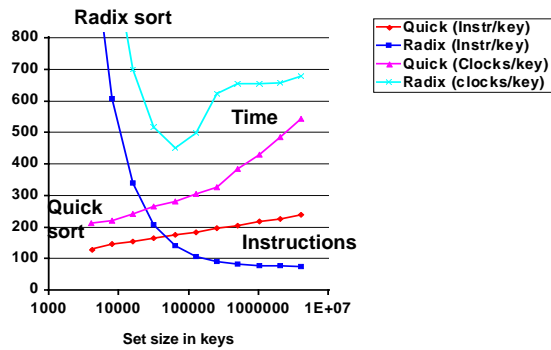
Quicksort vs. Radix as vary number keys: Instructions



15

DAP Fa97, © U.C.B

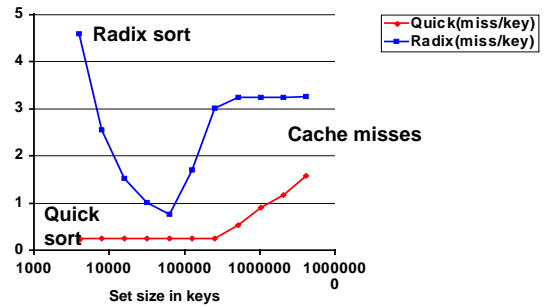
Quicksort vs. Radix as vary number keys: Instrs & Time



16

DAP Fa97, © U.C.B

Quicksort vs. Radix as vary number keys: Cache misses

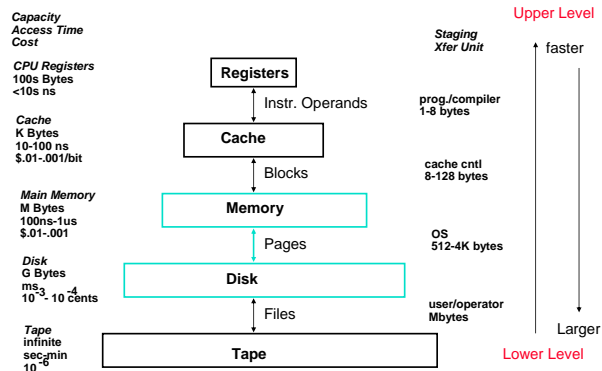


What is proper approach to fast algorithms?

17

DAP Fa97, © U.C.B

Recall: Levels of the Memory Hierarchy

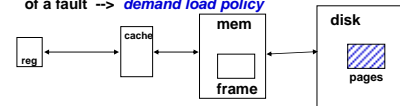


18

DAP Fa97, © U.C.B

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*

19

DAP Fa97, © U.C.B

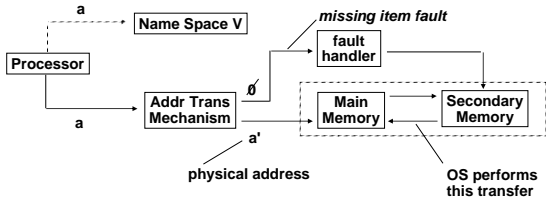
Address Map

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

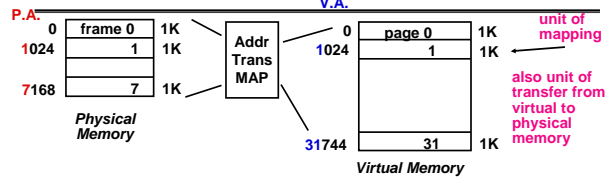
MAP: $V \rightarrow M \cup \{\emptyset\}$ address mapping function

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

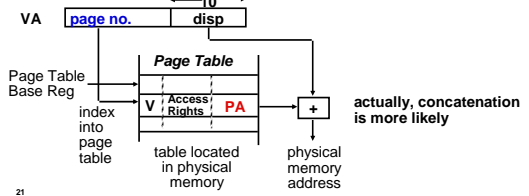
= \emptyset if data at virtual address a is not present in M



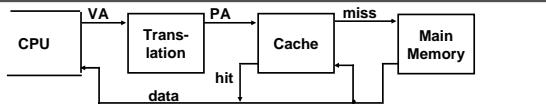
Paging Organization



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

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)

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.

