Computer System Architecture

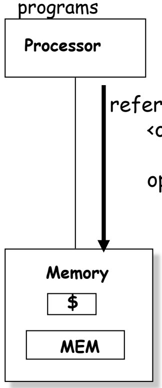
Memory Part II

Chalermek Intanagonwiwat

Slides courtesy of David Patterson and John Hennessy

The Art of Memory System Design

Workload or
Benchmark
programs



reference stream
<op,addr>, <op,addr>,<op,addr>,<op,addr>, . . .

op: i-fetch, read, write

Optimize the memory system organization to minimize the average memory access time for typical workloads

Cache

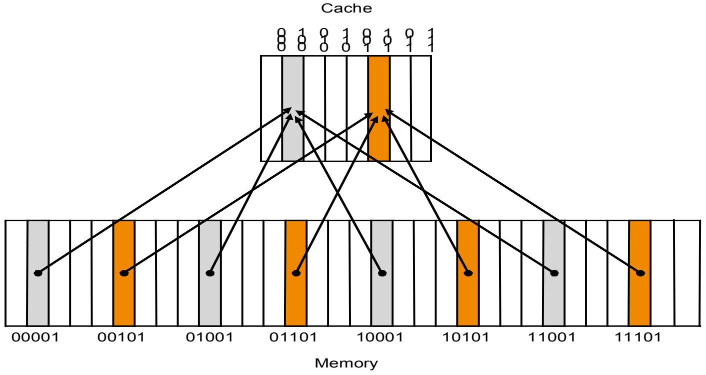
- Two issues:
  - How do we know if a data item is in the cache?
  - If it is, how do we find it?
- Our first example:
  - block size is one word of data
  - "direct mapped"

For each item of data at the lower level, there is exactly one location in the cache where it might be.

e.g., lots of items at the lower level share locations in the upper level

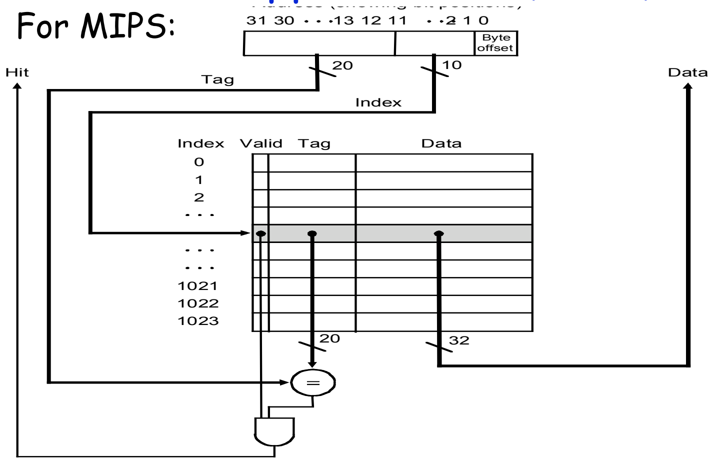
Direct Mapped Cache

- Mapping: address is modulo the number of blocks in the cache



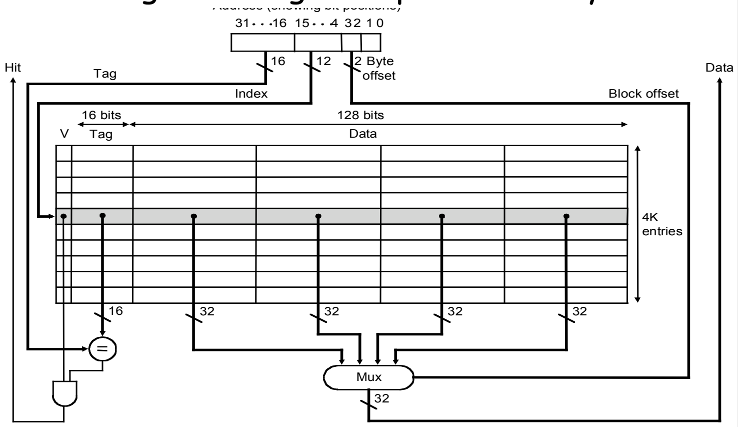
Direct Mapped Cache (cont.)

- For MIPS:

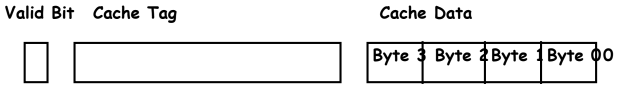


Direct Mapped Cache (cont.)

- Taking advantage of spatial locality:



Extreme Example: single big line



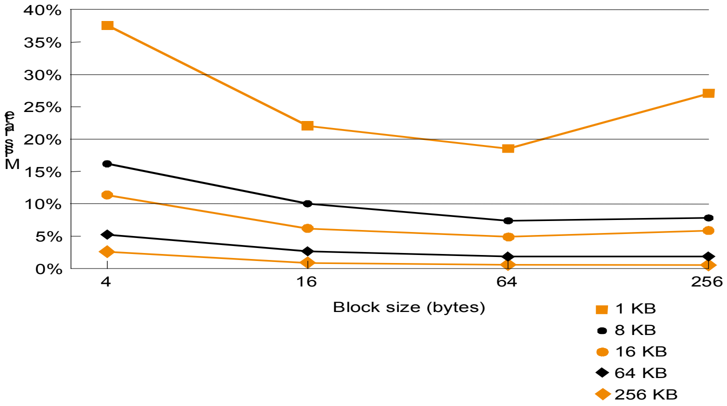
- Cache Size = 4 bytes
  Block Size = 4 bytes
  - Only ONE entry in the cache

Extreme Example: single big line (cont.)

- 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

Performance

- Increasing the block size tends to decrease miss rate:



Performance (cont.)

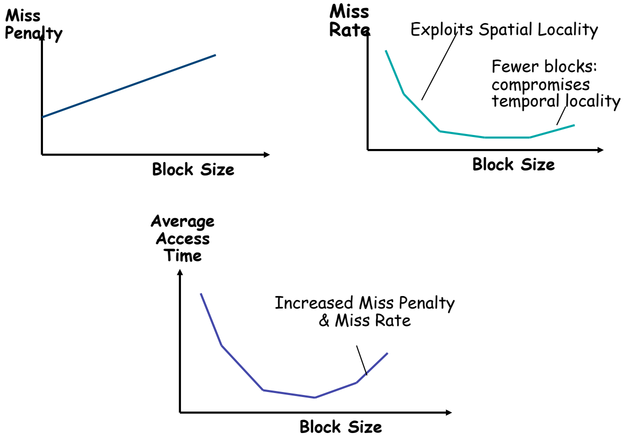
- Use split caches because there is more spatial locality in code:

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%

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

Block Size Tradeoff (cont.)


Miss Penalty
Block Size
Miss Rate
Exploits Spatial Locality
Fewer blocks: compromises temporal locality
Block Size
Average Access Time
Increased Miss Penalty & Miss Rate
Block Size

Hits vs. Misses

- Read hits
  - this is what we want!
- Read misses
  - stall the CPU, fetch block from memory, deliver to cache, restart

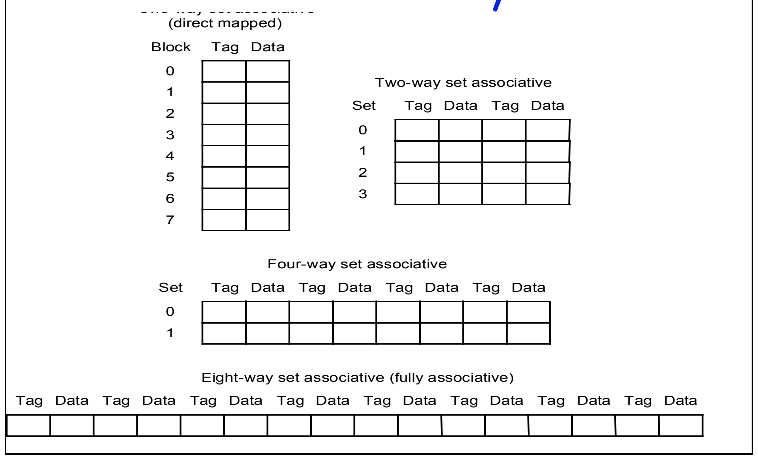
Hits vs. Misses (cont.)

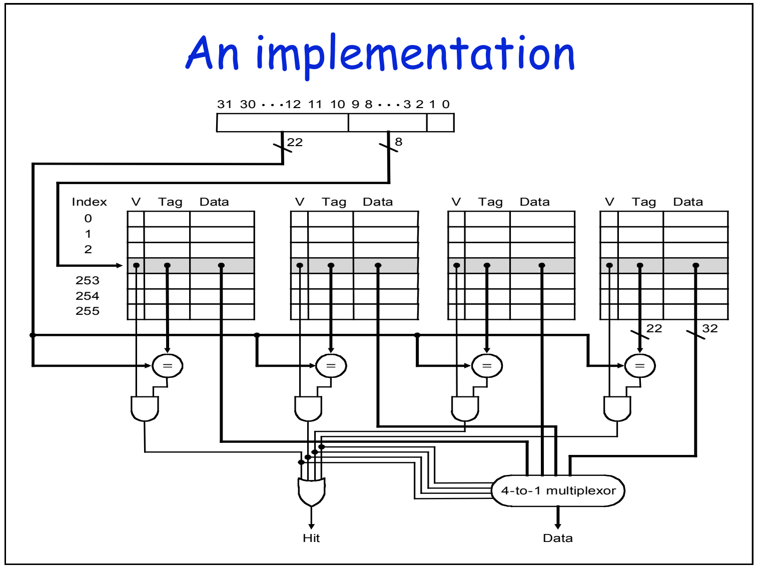
- Write hits:
  - can replace data in cache and memory (write-through)
  - write the data only into the cache (write-back the cache later)
- Write misses:
  - read the entire block into the cache, then write the word

Direct Mapped Cache and Conflict Misses

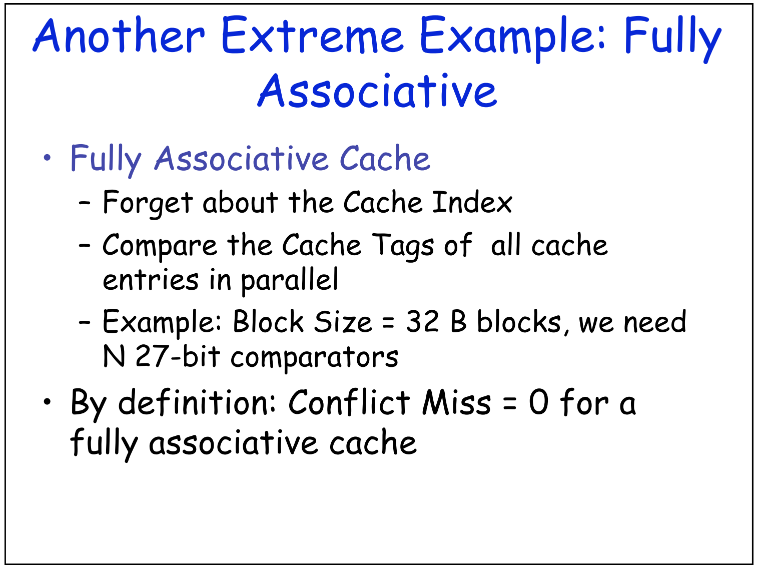
- Conflict Misses are misses caused by:
  - Different memory locations mapped to the same cache index
    - Solution 1: make the cache size bigger
    - Solution 2: Multiple entries for the same Cache Index

Decreasing miss ratio with associativity



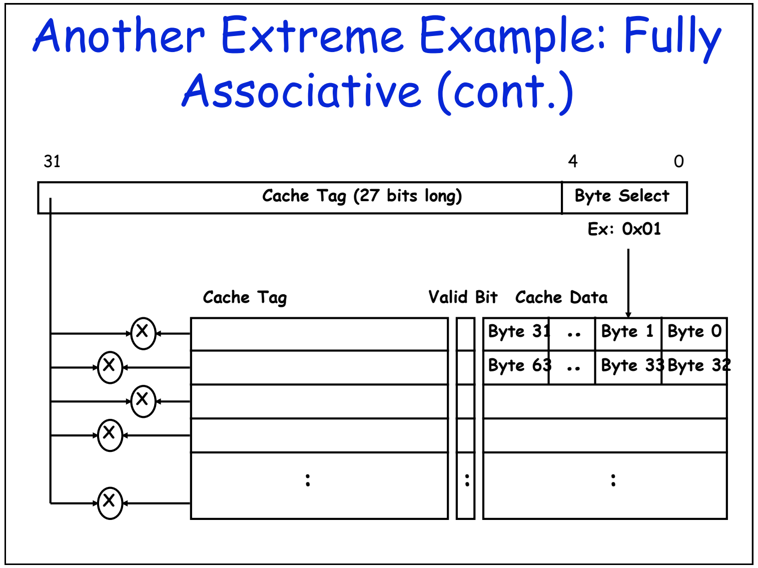


An implementation

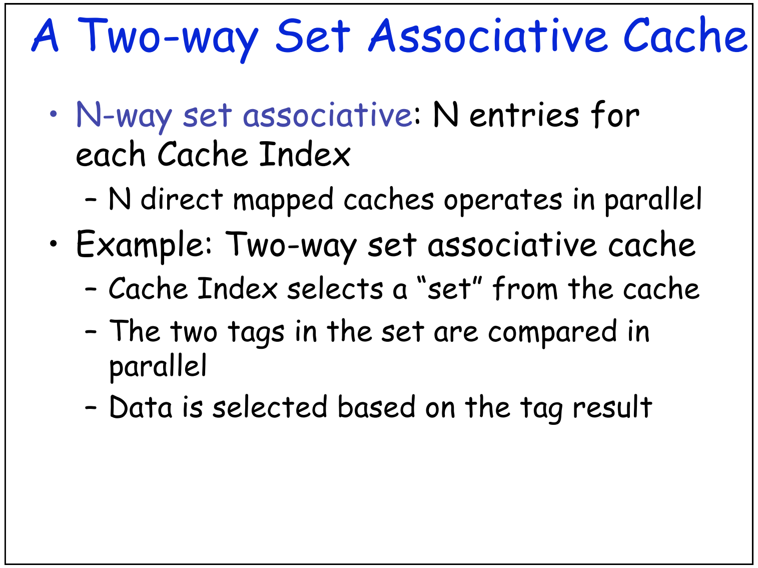


Another Extreme Example: Fully Associative

- Fully Associative Cache
  - Forget about the Cache Index
  - Compare the Cache Tags of all cache entries in parallel
  - Example: Block Size = 32 B blocks, we need N 27-bit comparators
- By definition: Conflict Miss = 0 for a fully associative cache



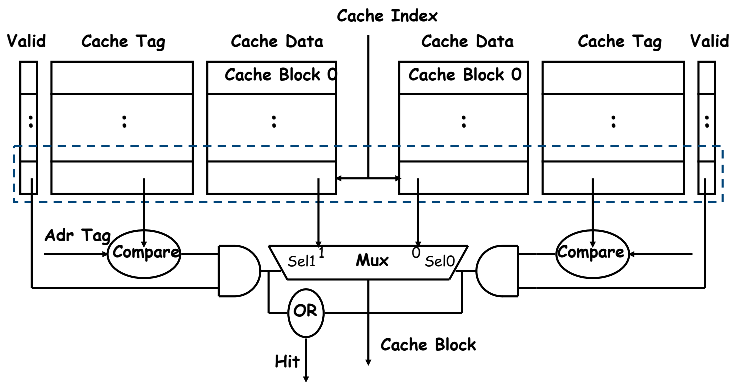
Another Extreme Example: Fully Associative (cont.)



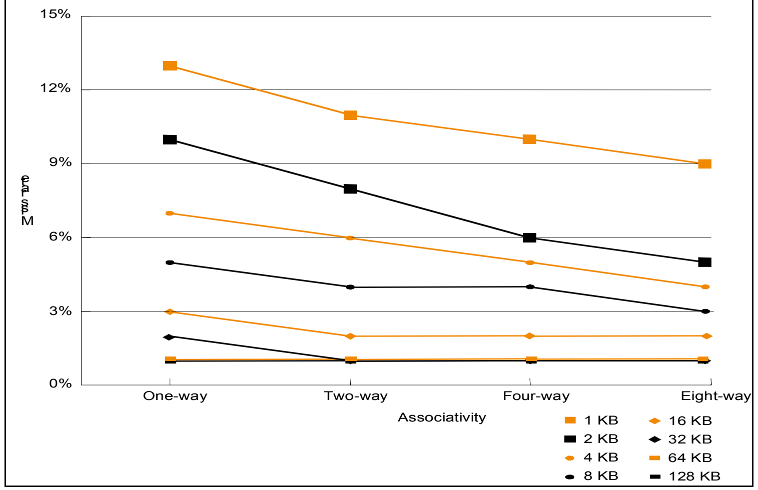
A Two-way Set Associative Cache

- 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

A Two-way Set Associative Cache (cont.)



Performance



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.

Decreasing miss penalty with multilevel caches

- Add a second level cache:
  - often primary cache is on the same chip as the processor
  - use SRAMs to add another cache above primary memory (DRAM)
  - miss penalty goes down if data is in 2nd level cache

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

A Summary on Sources of Cache Misses (cont.)

- 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

Sources of Cache Misses

	Direct Mapped	N-way Set Associative	Fully Associative
Cache Size	Big	Medium	Small
Compulsory Miss	Same	Same	Same
Conflict Miss	High	Medium	Zero
Capacity Miss	Low	Medium	High
Invalidation Miss	Same	Same	Same

Note:
If you are going to run "billions" of instruction, Compulsory Misses are insignificant.

Improving Cache Performance: 3 general options

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

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

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%

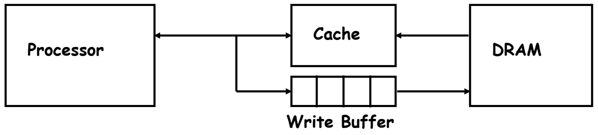
Q4: What happens on a write?

- **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?

Q4: What happens on a write? (cont.)

- 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

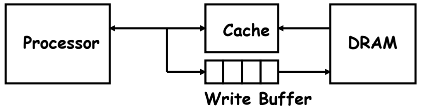


- A Write Buffer is needed between the Cache and Memory
  - Processor: writes data into the cache and the write buffer
  - Memory controller: write contents of the buffer to memory

Write Buffer for Write Through (cont.)

- 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

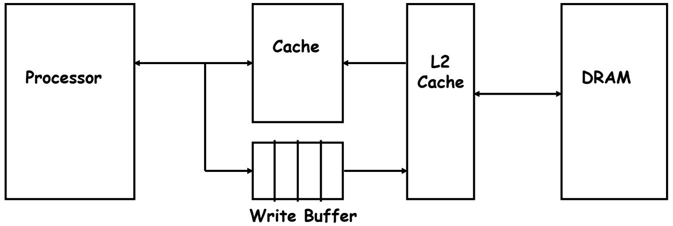
Write Buffer Saturation



- 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

Write Buffer Saturation (cont.)

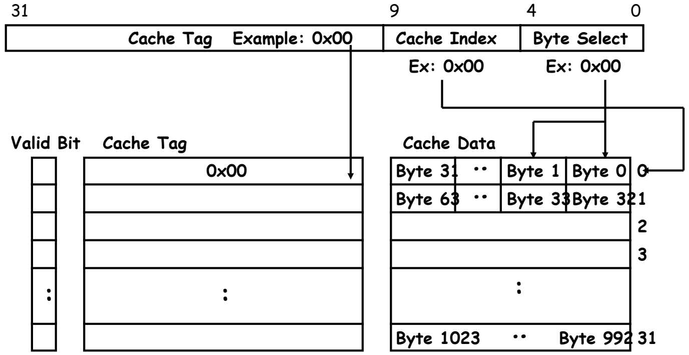
- Solution for write buffer saturation:
  - Use a write back cache
  - Install a second level (L2) cache:



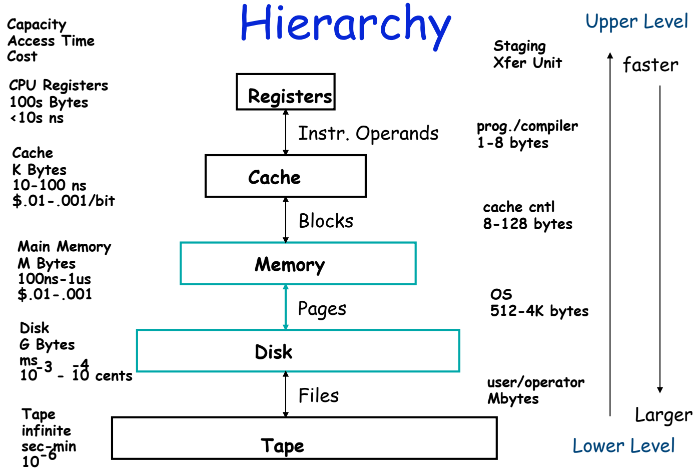
Write-miss Policy: Write Allocate versus Not Allocate

- 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

Write-miss Policy: Write Allocate versus Not Allocate (cont.)

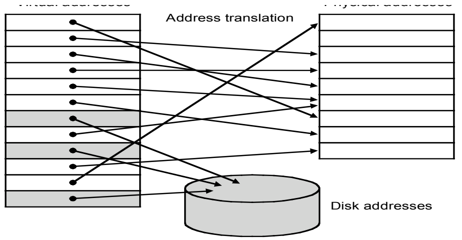


Recall: Levels of the Memory Hierarchy



Virtual Memory

- Main memory can act as a cache for the secondary storage (disk)



- Advantages:
  - illusion of having more physical memory
  - program relocation
  - protection

Basic Issues in Virtual Memory System Design

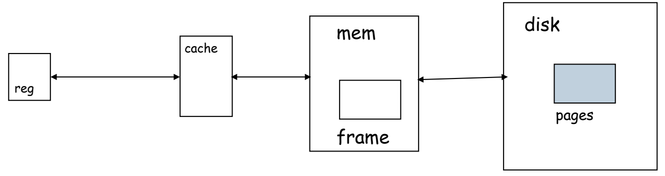
- Size of information blocks that are transferred from secondary to main storage (M)

- 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**

Basic Issues in Virtual Memory System Design (cont.)



Paging Organization

virtual and physical address space partitioned into blocks of equal size

page frames

pages

Pages: virtual memory blocks

- Page faults: the data is not in memory, retrieve it from disk
  - huge miss penalty, thus pages should be fairly large (e.g., 4KB)
  - reducing page faults is important (LRU is worth the price)
  - can handle the faults in software instead of hardware
  - using write-through is too expensive so we use writeback

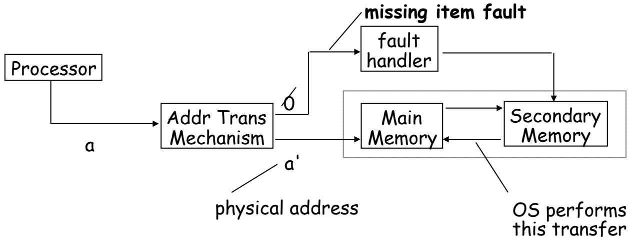
Address Map

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

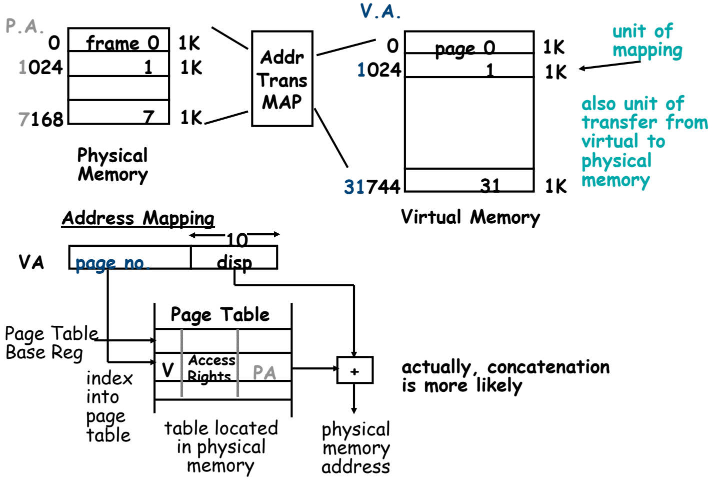
MAP: V --> M U {Ø} address mapping function

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

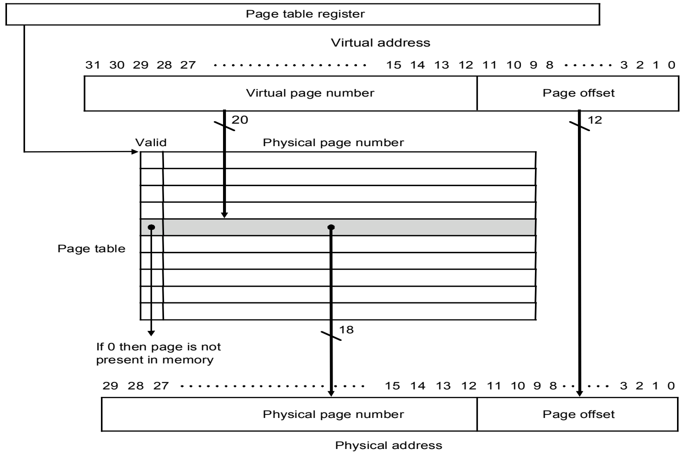
= Ø if data at virtual address a is not present in M



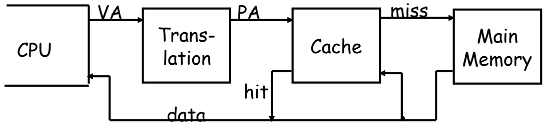
Paging Organization



Page Tables



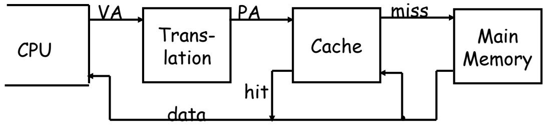
Virtual Address and a Cache



- 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 example, shared pages)

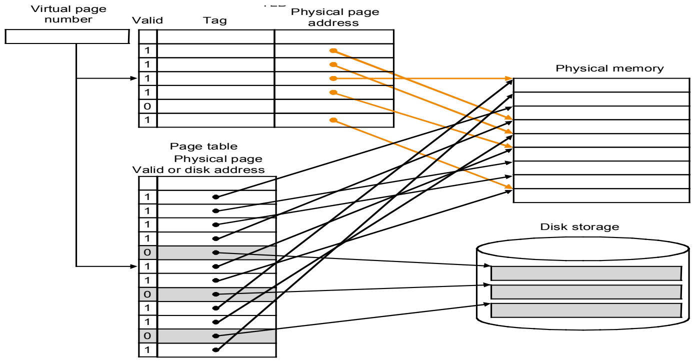
Virtual Address and a Cache (cont.)



- 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

Making Address Translation Fast

- A cache for address translations: translation lookaside buffer



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**

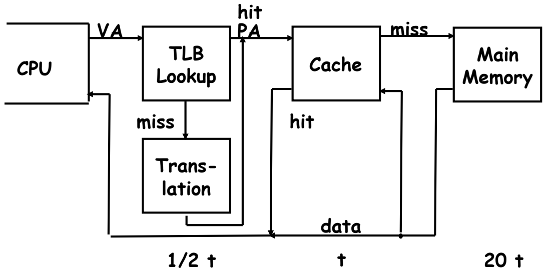
Virtual Address	Physical Address	Dirty	Ref	Valid	Access

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

Translation Look-Aside Buffers

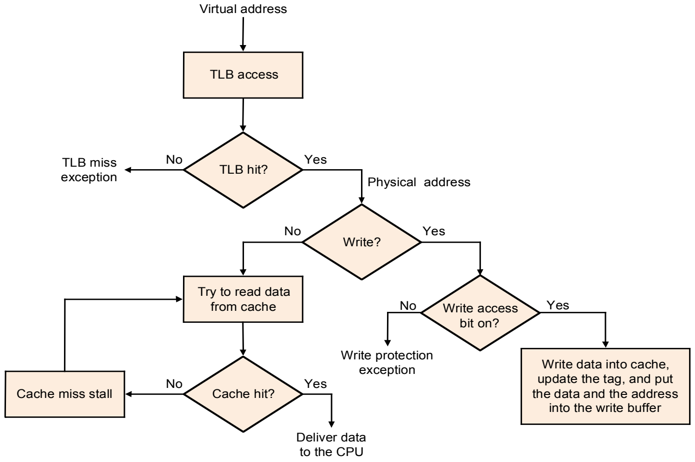
Just like any other cache, the TLB can be organized as fully associative, set associative, or direct mapped

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.



Translation with a TLB

TLBs and caches



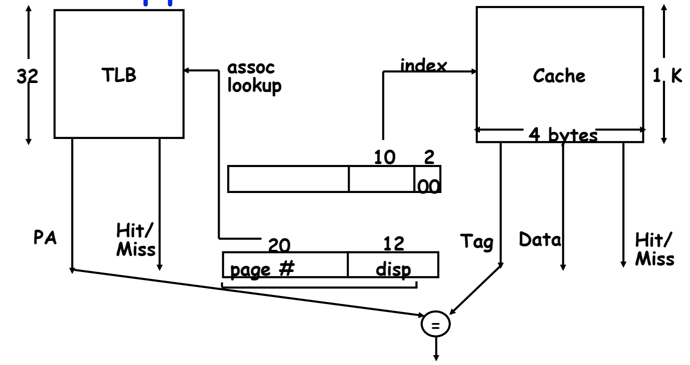
Reducing Translation Time

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

Overlapped Cache & TLB Access



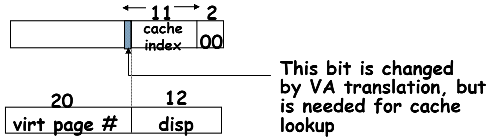
IF cache hit AND (cache tag = PA)
then deliver data to CPU
ELSE IF [cache miss OR (cache tag != PA)] and TLB hit
    THEN access memory with the PA from the TLB
    ELSE do standard VA translation

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

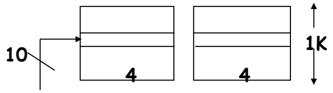
Example: suppose everything the same except that the cache is increased to 8 K bytes instead of 4 K:



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

Problems With Overlapped TLB Access (cont.)

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

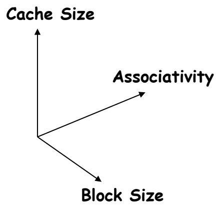
- The Principle of Locality:
  - Program likely to access a relatively small portion of the address space at any instant of time.
    - Temporal Locality: Locality in Time
    - Spatial Locality: Locality in Space

Summary (cont.)

- Three 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

Summary: The Cache Design Space

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



Summary: The Cache Design Space (cont.)

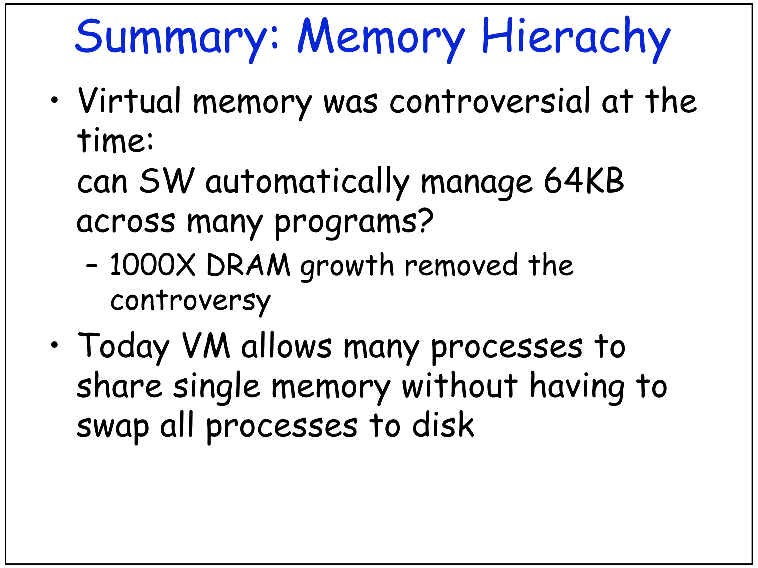
- The optimal choice is a compromise
  - depends on access characteristics
    - workload
    - use (I-cache, D-cache, TLB)
  - depends on technology / cost
- Simplicity often wins

Summary: 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

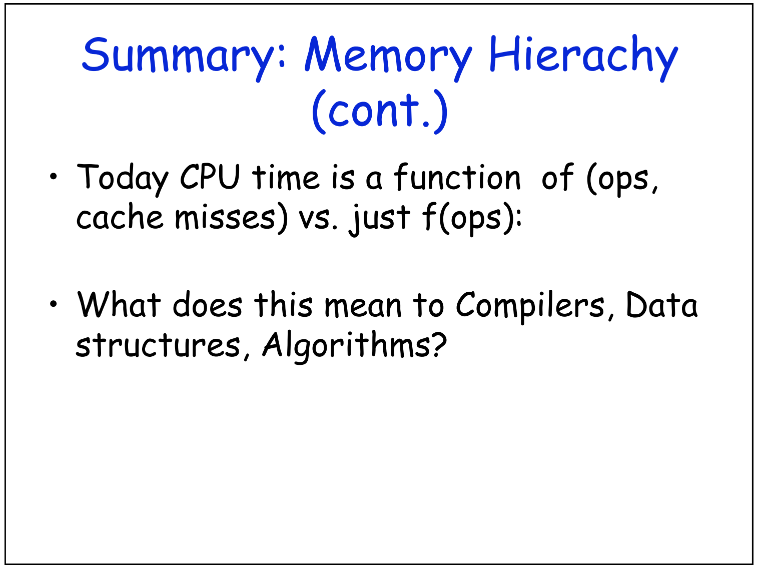
Summary: TLB, Virtual Memory (cont.)

- TLBs are important for fast translation
- TLB misses are significant in processor performance



Summary: 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



Summary: Memory Hierachy (cont.)

- Today CPU time is a function of (ops, cache misses) vs. just f(ops):
- What does this mean to Compilers, Data structures, Algorithms?