CS152: Computer Architecture and Engineering
Caches and Virtual Memory

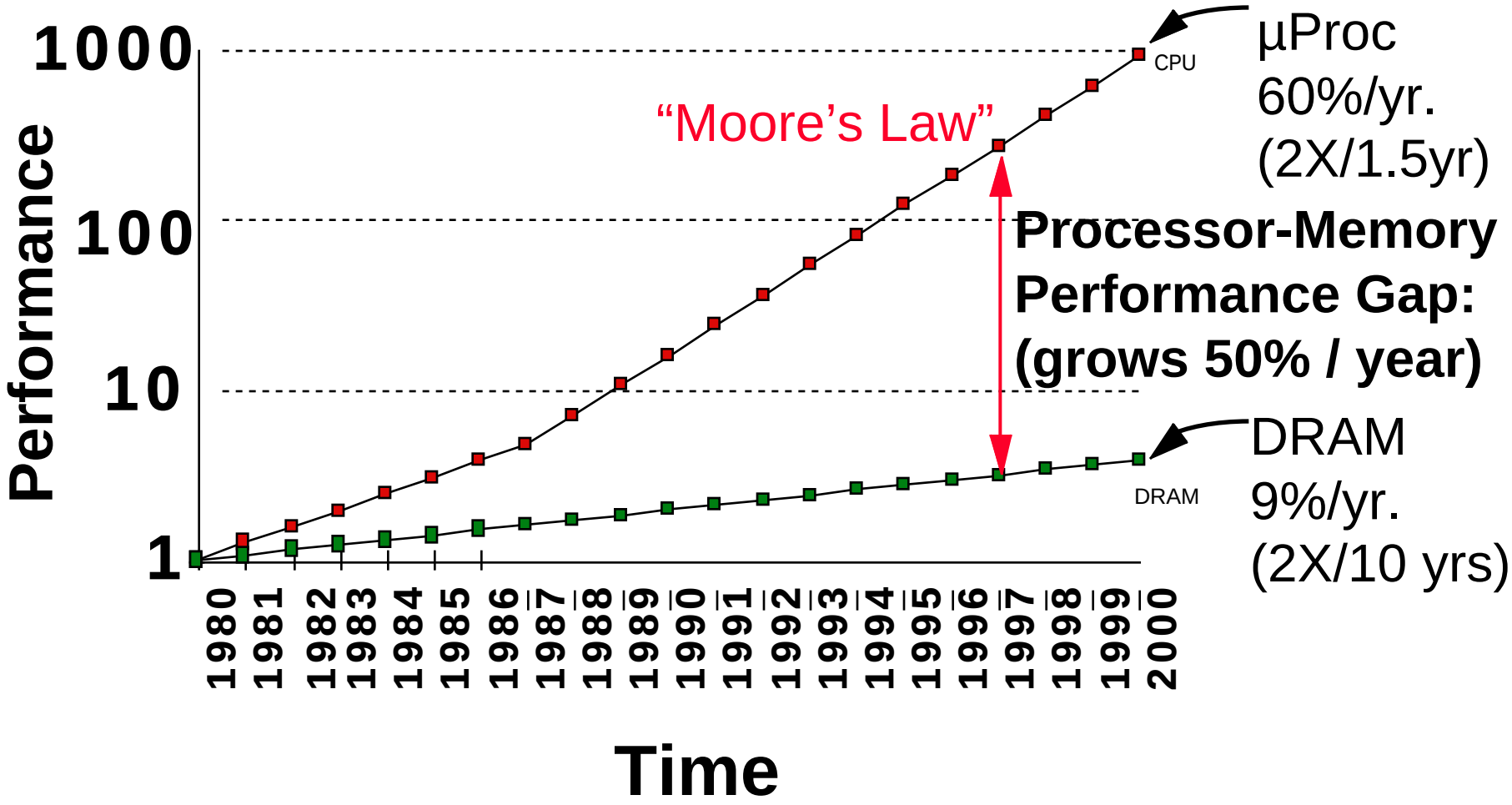
October 31, 1997

Dave Patterson (http.cs.berkeley.edu/~patterson)

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

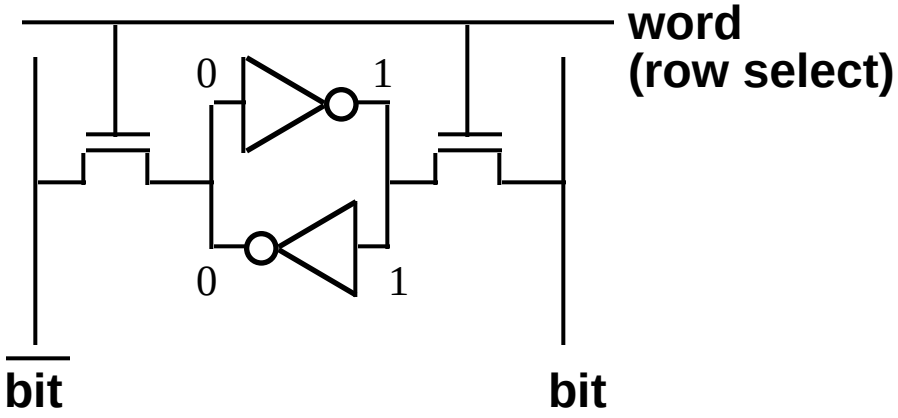
Recap: Who Cares About the Memory Hierarchy?

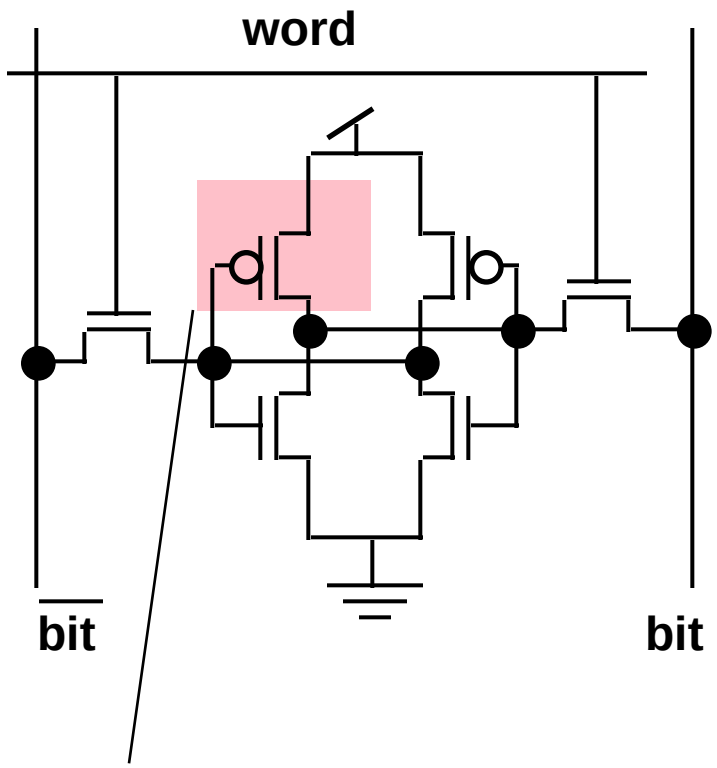
Processor-DRAM Memory Gap (latency)



Recap: Static RAM Cell

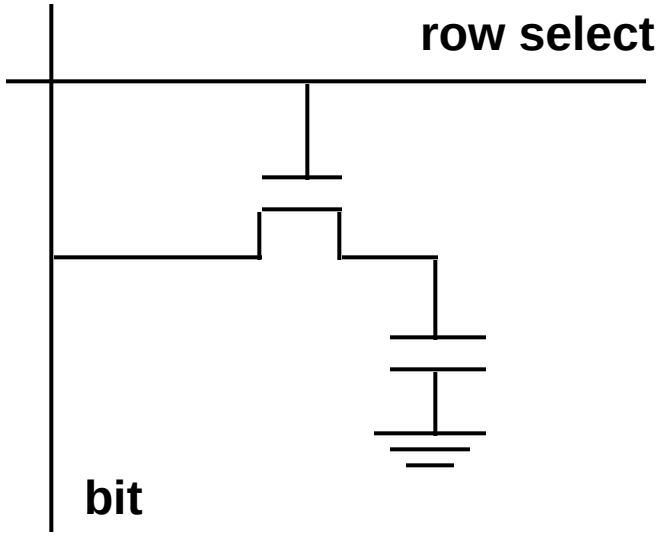
6-Transistor SRAM Cell





- **Write:**
  1. Drive bit lines (bit=1, $\overline{\text{bit}}$=0)
  2.. Select row
- **Read:**
  1. Precharge bit and bit to Vdd
  2.. Select row
  3. Cell pulls one line low
  4. Sense amp on column detects difference between bit and $\overline{\text{bit}}$

Recap: 1-Transistor Memory Cell (DRAM)



- **Write:**
  - 1. Drive bit line
  - 2.. Select row
- **Read:**
  - 1. Precharge bit line to Vdd
  - 2.. Select row
  - 3. Cell and bit line share charges
    - Very small voltage changes on the bit line
  - 4. Sense (fancy sense amp)
    - Can detect changes of ~1 million electrons
  - 5. Write: restore the value
- **Refresh**
  - 1. Just do a dummy read to every cell.

DRAMs over Time

	DRAM Generation					
1st Gen. Sample	‘84	‘87	‘90	‘93	‘96	‘99
Memory Size	1 Mb	4 Mb	16 Mb	64 Mb	256 Mb	1 Gb
Die Size (mm^2)	55	85	130	200	300	450
Memory Area (mm^2)	30	47	72	110	165	250
Memory Cell Area (μm^2)	28.84	11.1	4.26	1.64	0.61	0.23

(from Kazuhiro Sakashita, Mitsubishi)

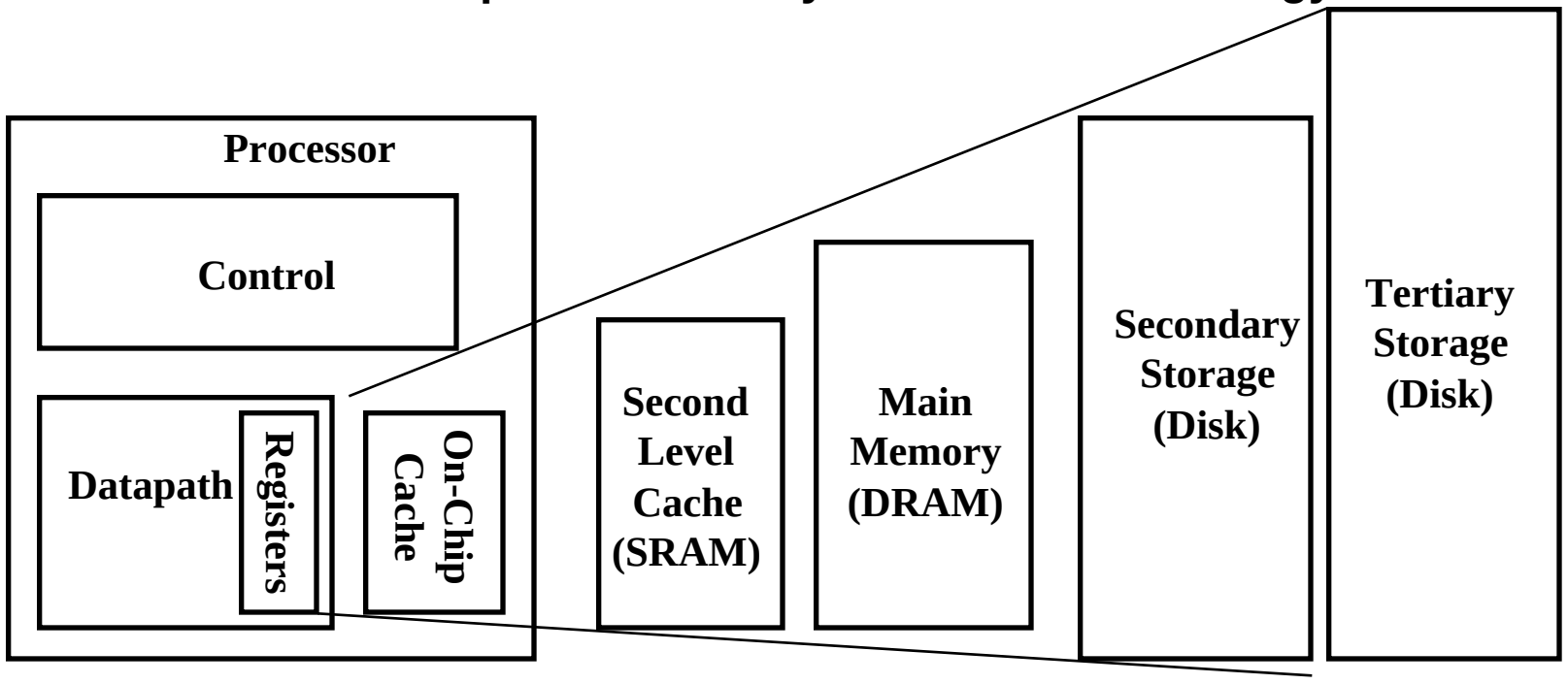
DRAM v. Desktop Microprocessors Cultures

Standards	pinout, package, refresh rate, capacity, ...	binary compatibility, IEEE 754, I/O bus
Sources	Multiple	Single
Figures of Merit	1) capacity, 1a) $/bit 2) BW, 3) latency	1) SPEC speed 2) cost
Improve Rate/year	1) 60%, 1a) 25%, 2) 20%, 3) 7%	1) 60%, 2) little change

Recap: 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.**



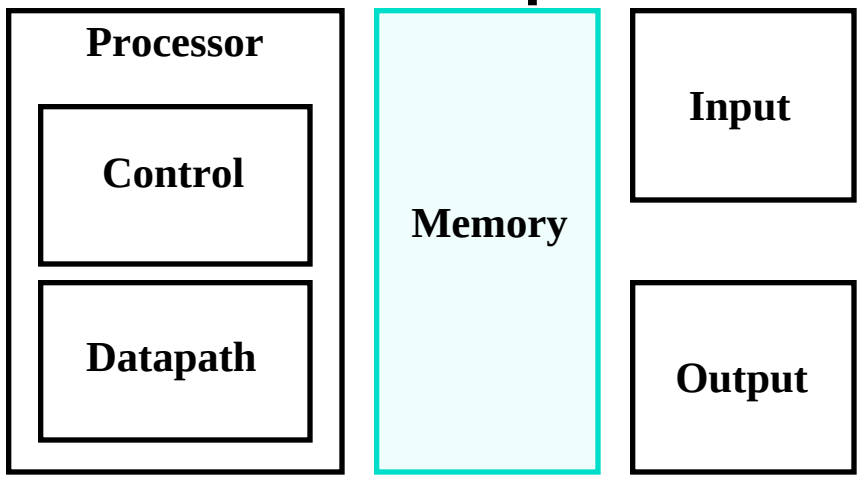
Speed (ns):	1s	10s		100s	10,000,000s (10s ms)	10,000,000,000s (10s sec)
Size (bytes):	100s	Ks		Ms	Gs	Ts

Recap:

- **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.**

The Big Picture: Where are We Now?

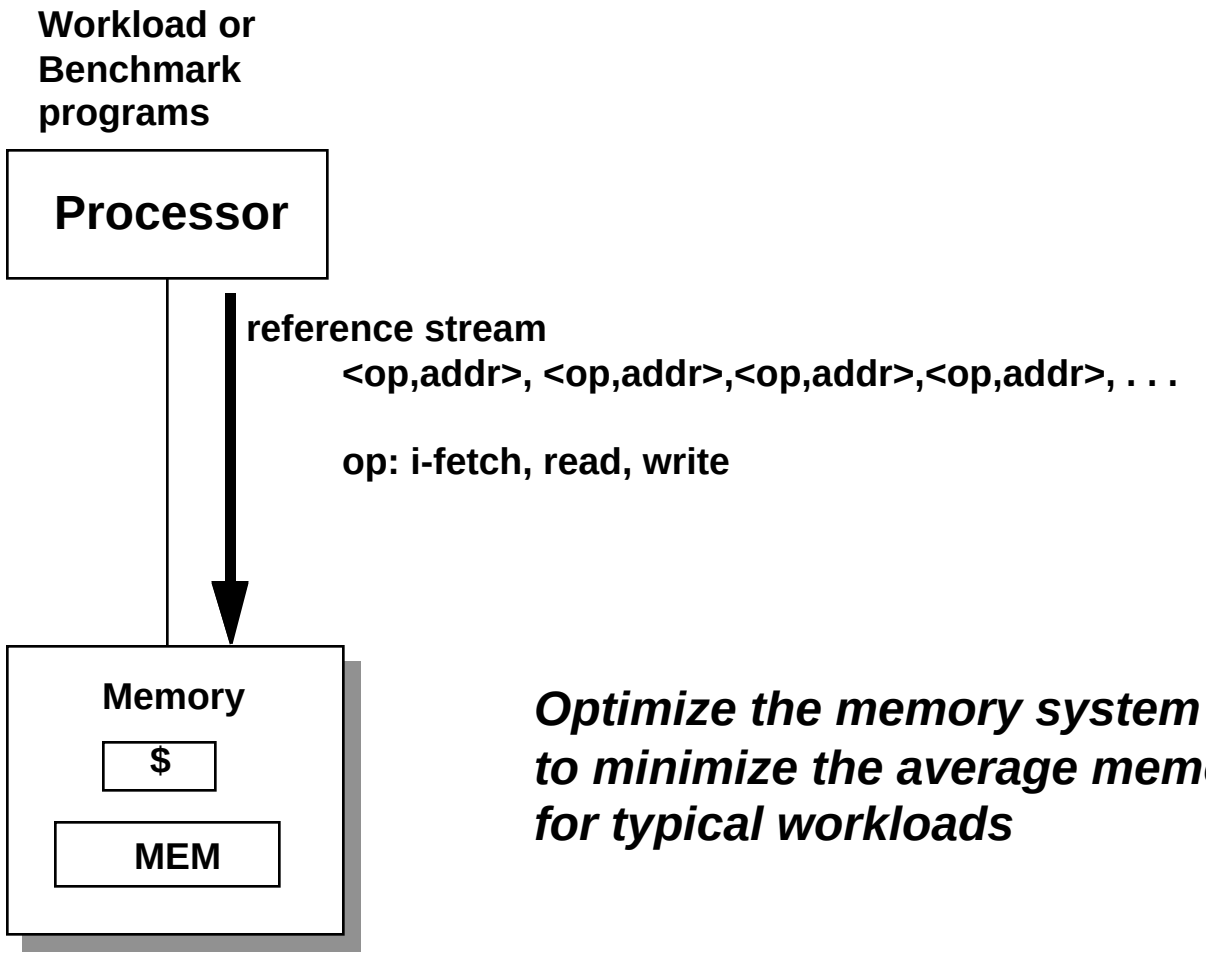
° The Five Classic Components of a Computer



° Today's Topics:

- Recap last lecture
- Cache Review
- Administrivia
- Advanced Cache
- Virtual Memory
- Protection
- TLB

The Art of Memory System Design

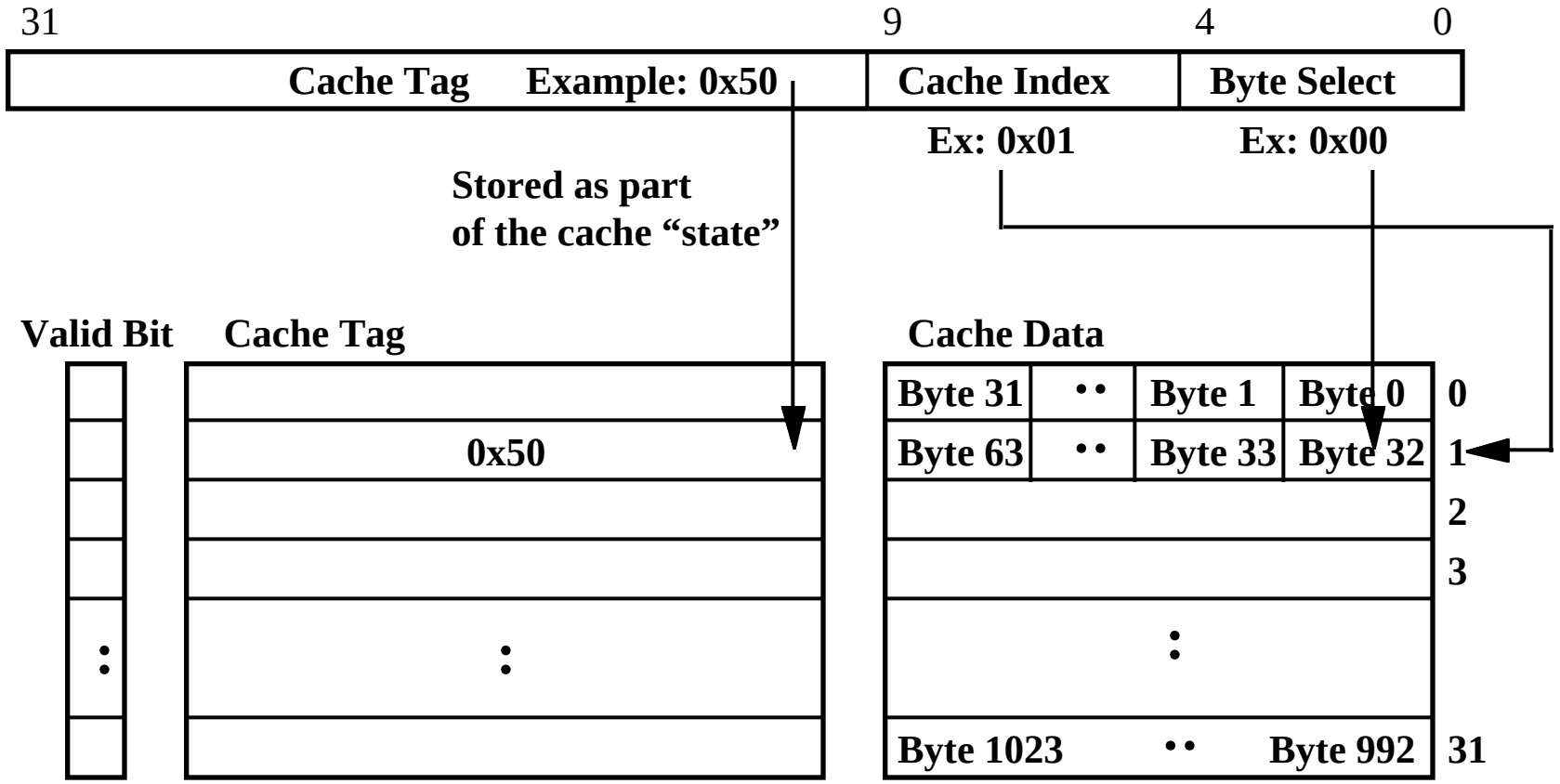


Optimize the memory system organization to minimize the average memory access time for typical workloads

Example: 1 KB Direct Mapped Cache with 32 B Blocks

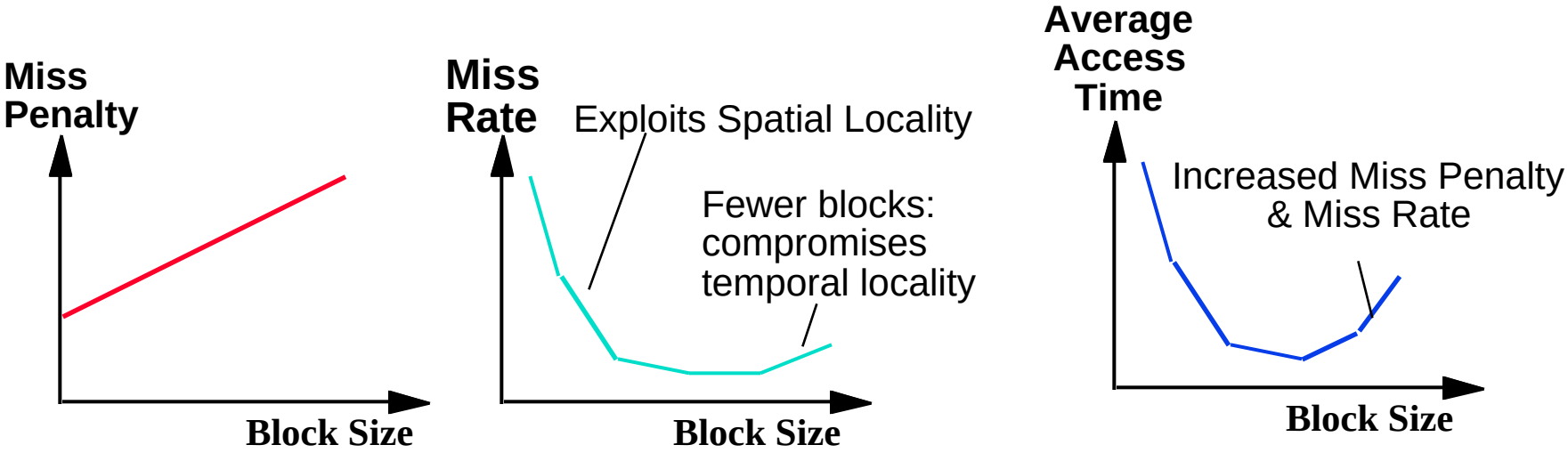
° For a 2 ** N byte cache:

- The uppermost (32 - N) bits are always the Cache Tag
- The lowest M bits are the Byte Select (Block Size = 2 ** M)

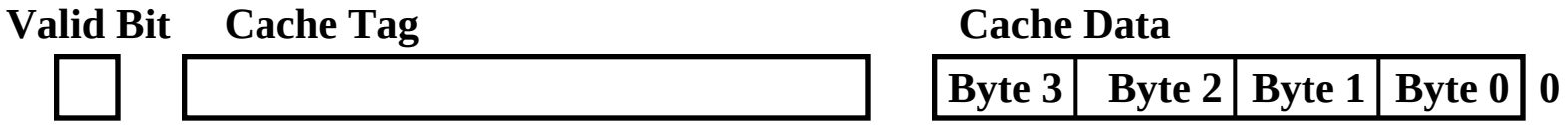


Block Size Tradeoff

- 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 gerneral, Average Access Time:
  - = Hit Time x (1 - Miss Rate) + Miss Penalty x Miss Rate



Extreme Example: single big line



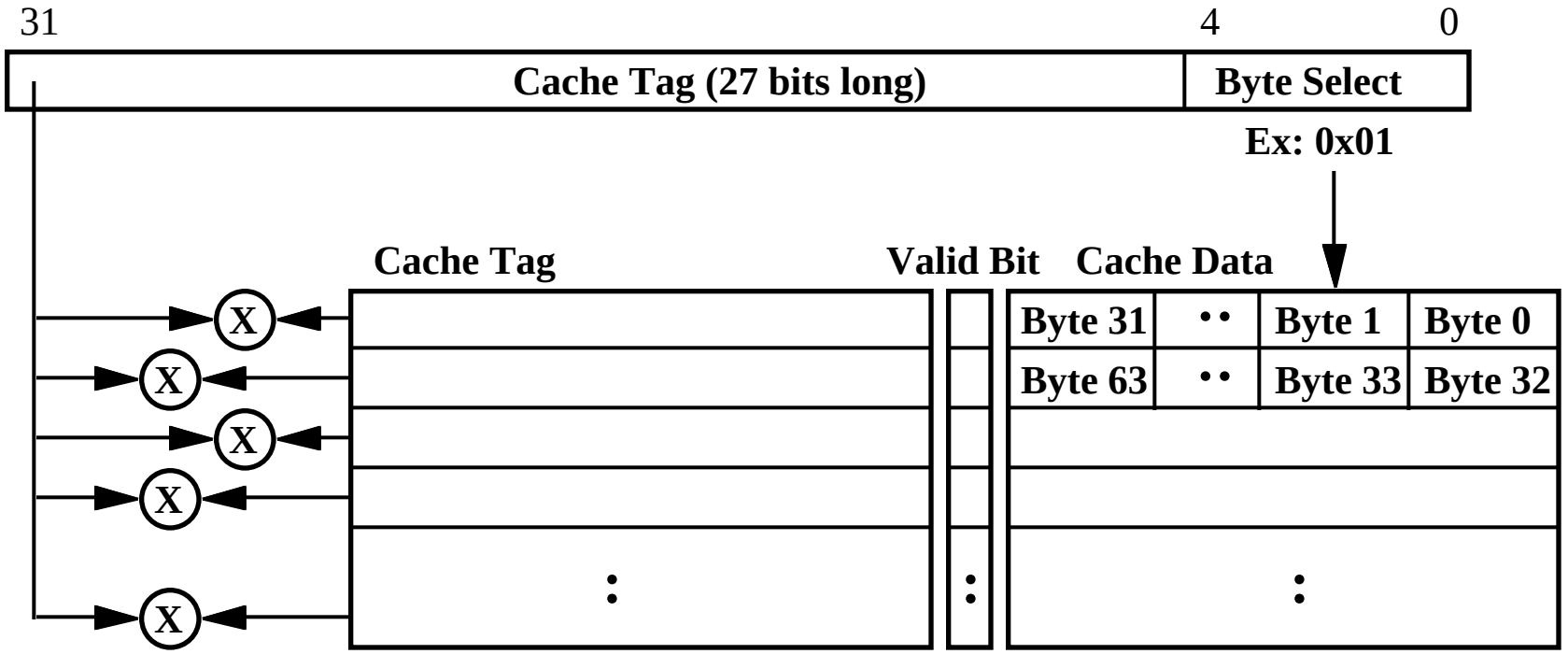
- Cache Size = 4 bytes Block Size = 4 bytes
  - Only ONE entry in the cache
- If an item is accessed, likely that it will be accessed again soon
  - But it is unlikely that it will be accessed again immediately!!!
  - The next access will likely to be a miss again
    - Continually loading data into 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**

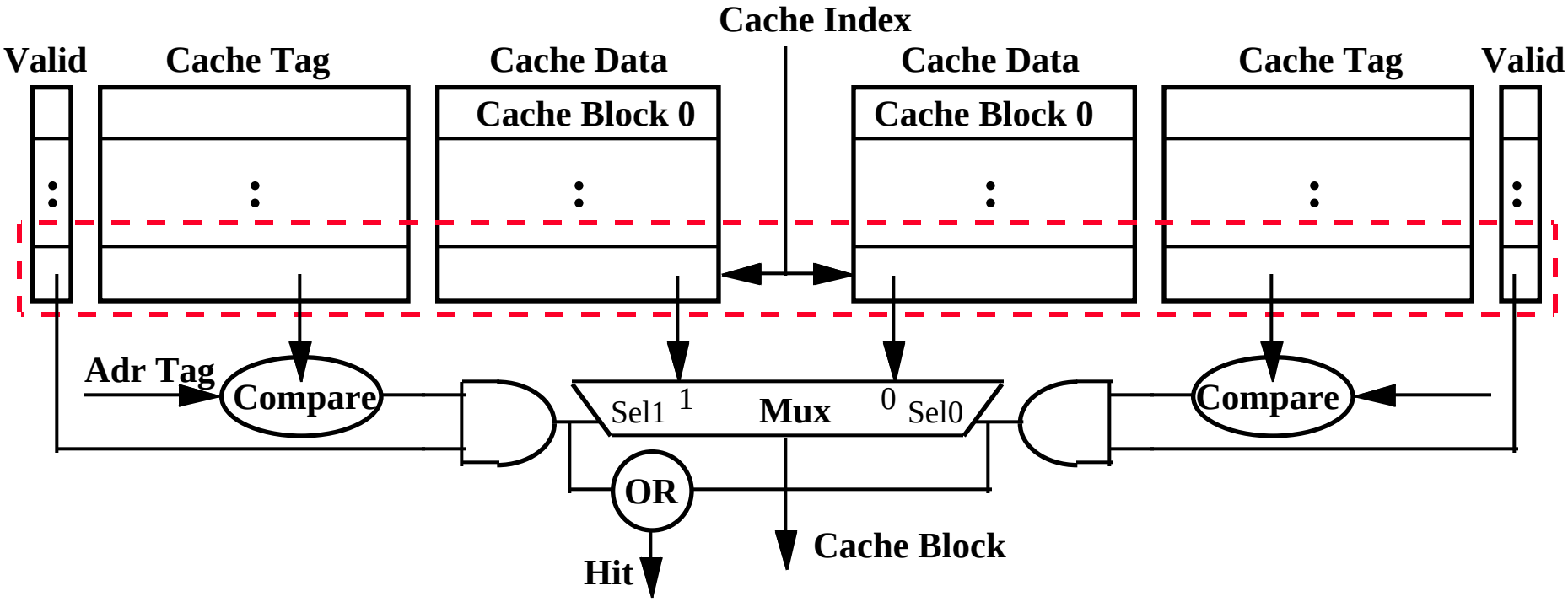
- **Forget about the Cache Index**
- **Compare the Cache Tags of all cache entries in parallel**
- **Example: Block Size = 2 B blocks, we need N 27-bit comparators**

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



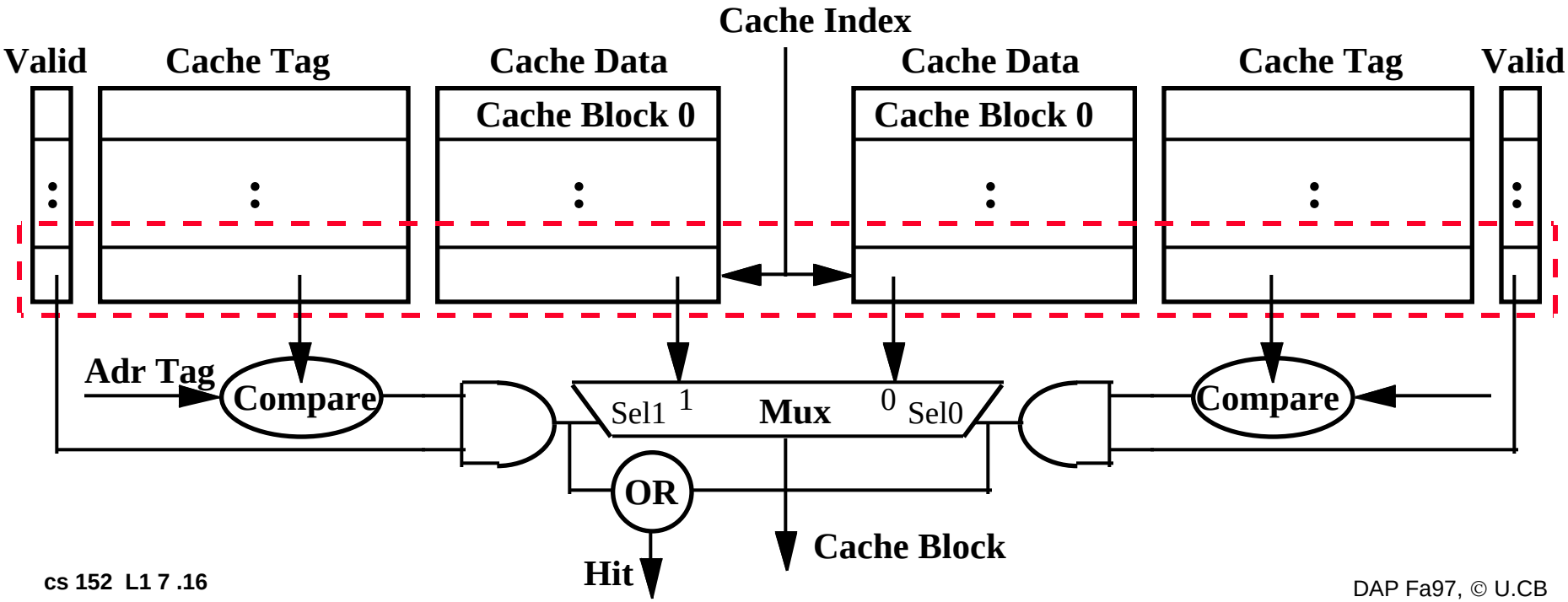
A Two-way Set Associative Cache

- **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

Administrative Issues

° New Office Hours:

- Gebis: Tue, 3:30-4:30, Kirby: Wed 1-2, Kozyrakis: Mon 1pm-2pm, Th 11am-noon ,Patterson: Wed 1-2 and Wed 3:30-4:30

° Reflector site for handouts and lecture notes (backup):

- http://HTTP.CS.Berkeley.EDU/~patterson/152F97/index_handouts.html
- http://HTTP.CS.Berkeley.EDU/~patterson/152F97/index_lectures.html

° Read: Chapter 7 of COD 2/e; how many taken CS162?

° Upcoming events in CS152:

• Wed	11/5	Intro to I/O Systems	Brian Wong, Sun
• Fri	11/7	Advanced I/O Systems	Brian Wong, Sun
• Wed	11/12	Intro Digital Signal Processor (DSP)	Prof. Brodersen
• Fri	11/14	Advanced DSP	Jeff Bier, BDTI
• Sun	11/16	Miterm Review 1-3PM 306 Soda	TAs
• Wed	11/19	Midterm II 5:30-8:30 306 Soda; >8:30 - pizza@La Val's	
• Fri	11/21	Field Trip to Intel (leave 9AM, Return 5PM)	

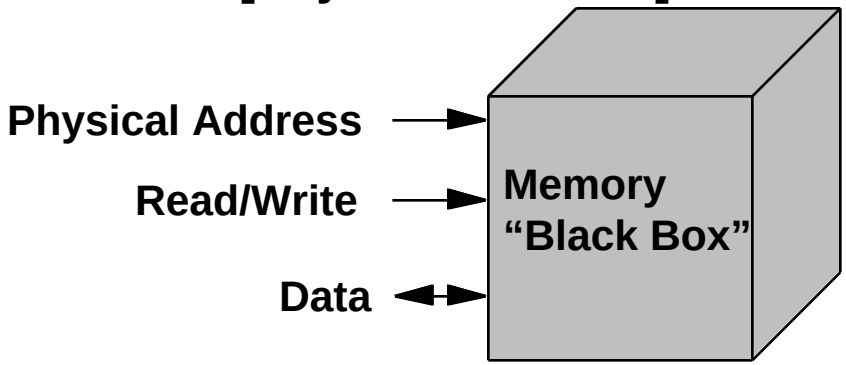
Sources of Cache Misses Answer

	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.

How Do you Design a Cache?

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

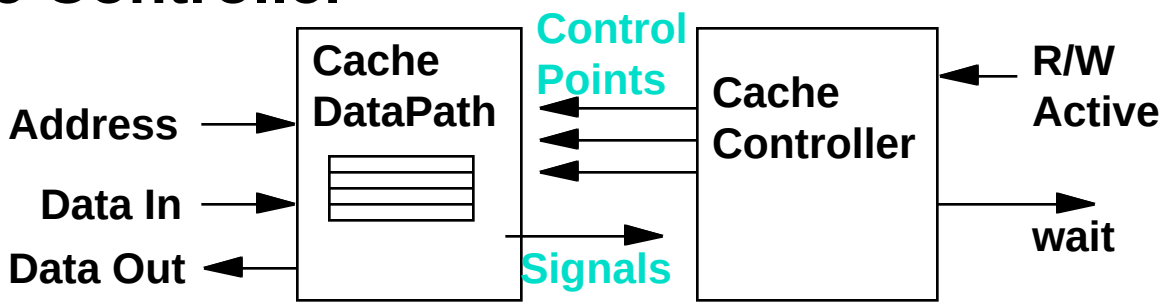


Inside it has:
Tag-Data Storage,
Muxes,
Comparators, . . .

° Deterimine the internal register transfers

° Design the Datapath

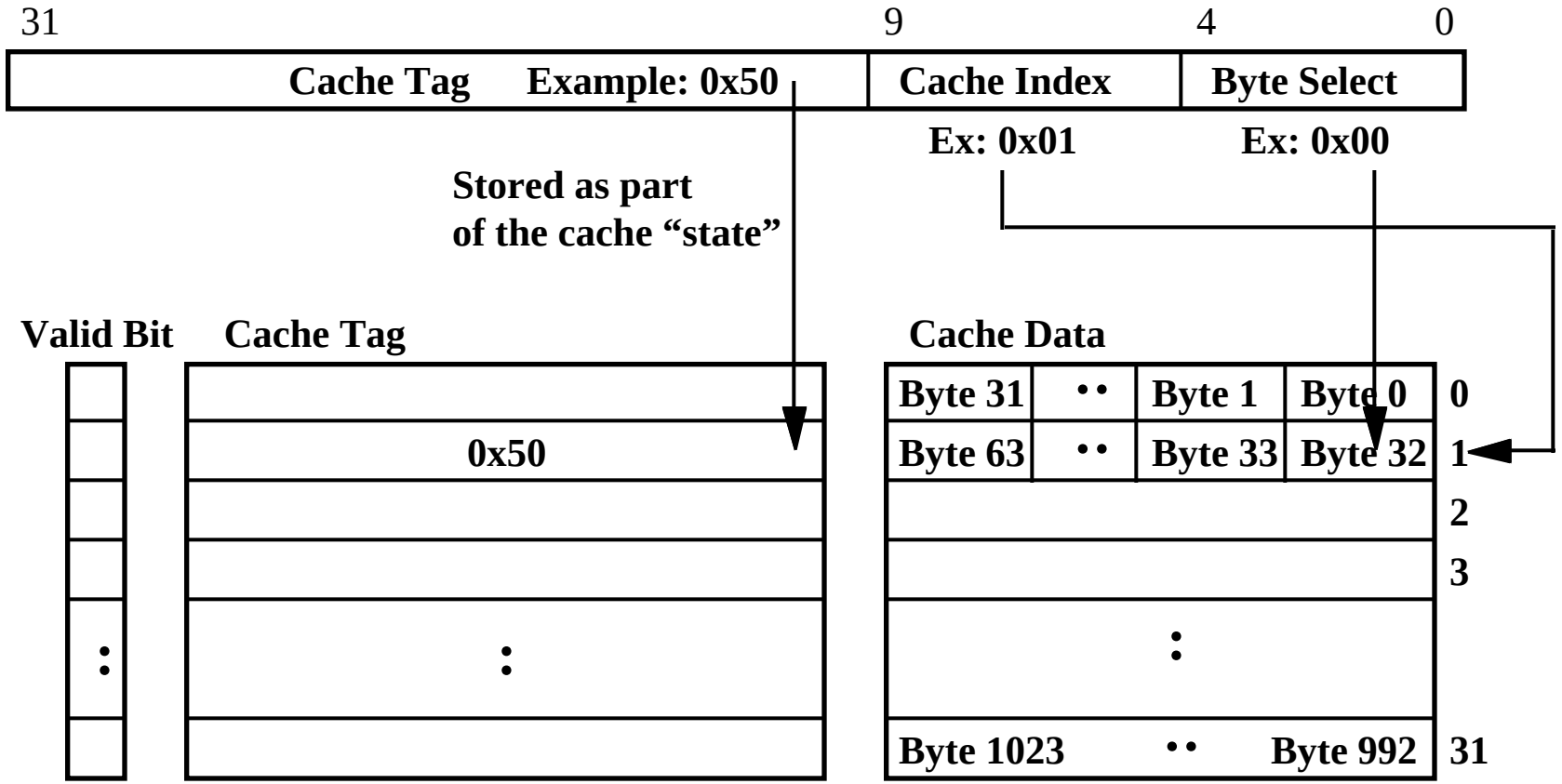
° Design the Cache Controller



1 KB Direct Mapped Cache, 32B blocks

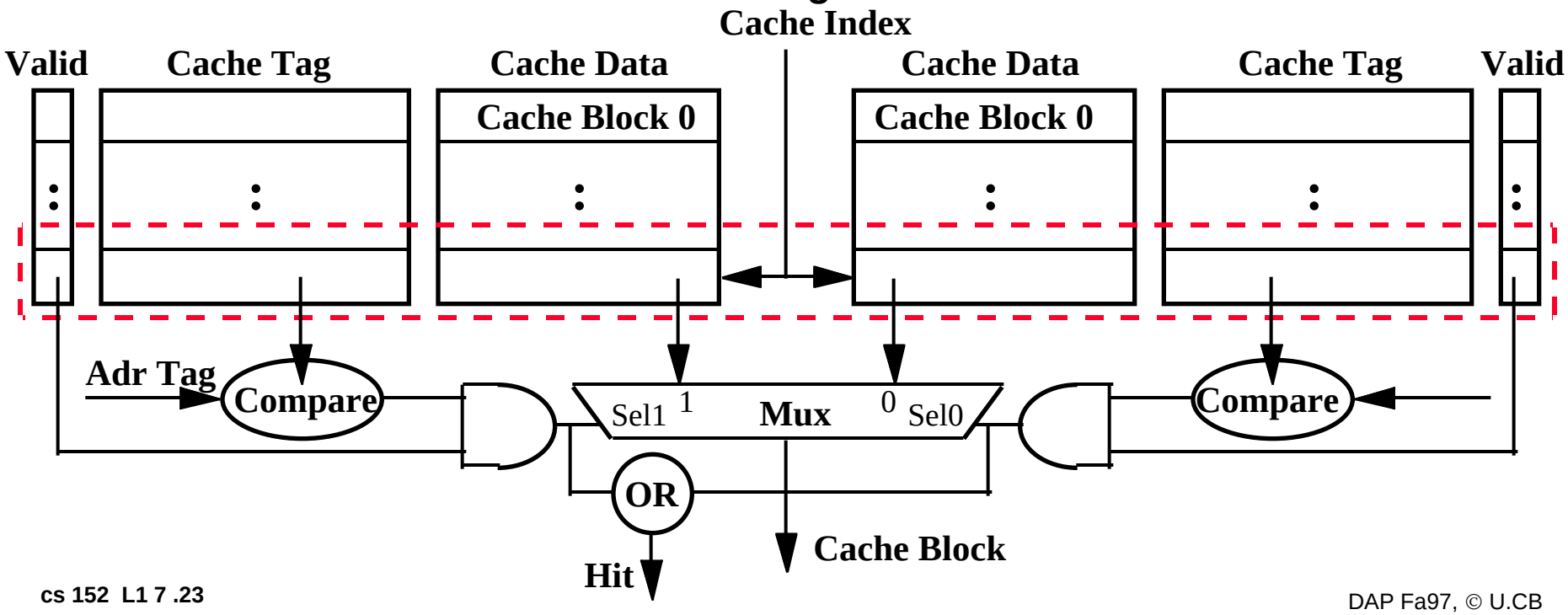
° For a 2 ** N byte cache:

- The uppermost (32 - N) bits are always the Cache Tag
- The lowest M bits are the Byte Select (Block Size = 2 ** M)



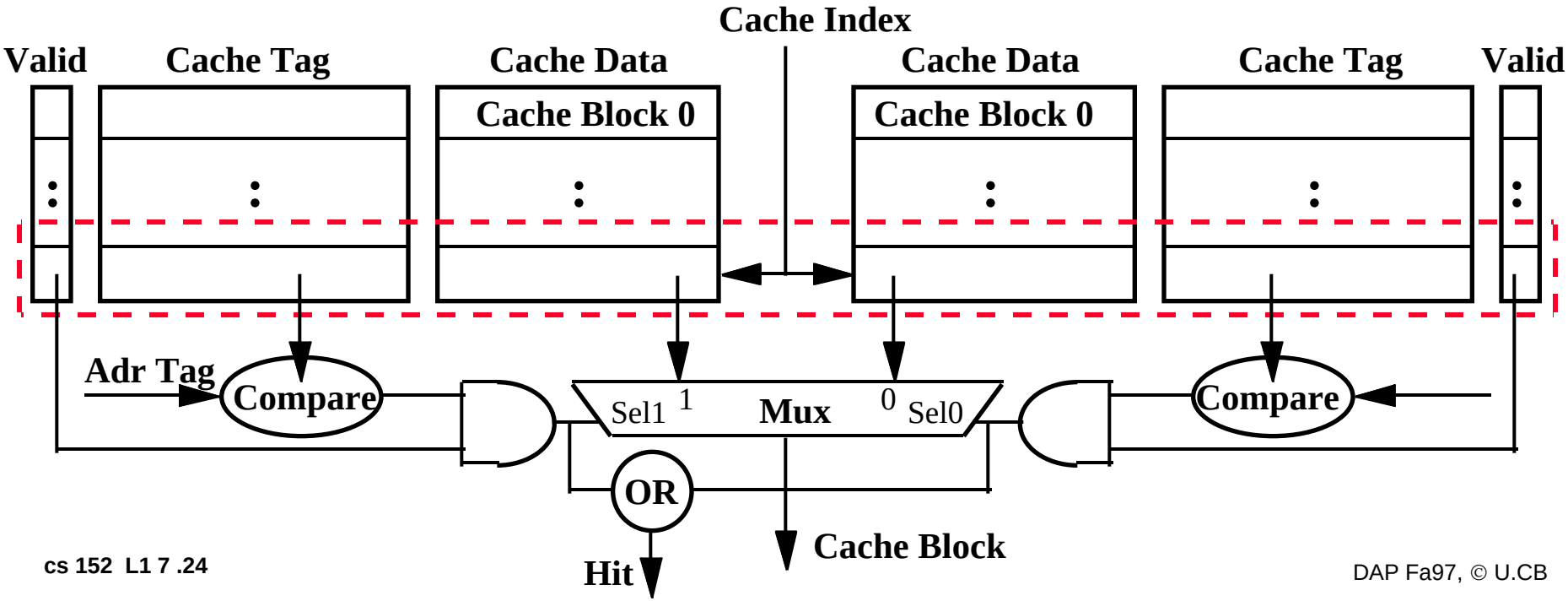
Two-way Set Associative Cache

- N-way set associative: N entries for each Cache Index
  - N direct mapped caches operates in parallel
- 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

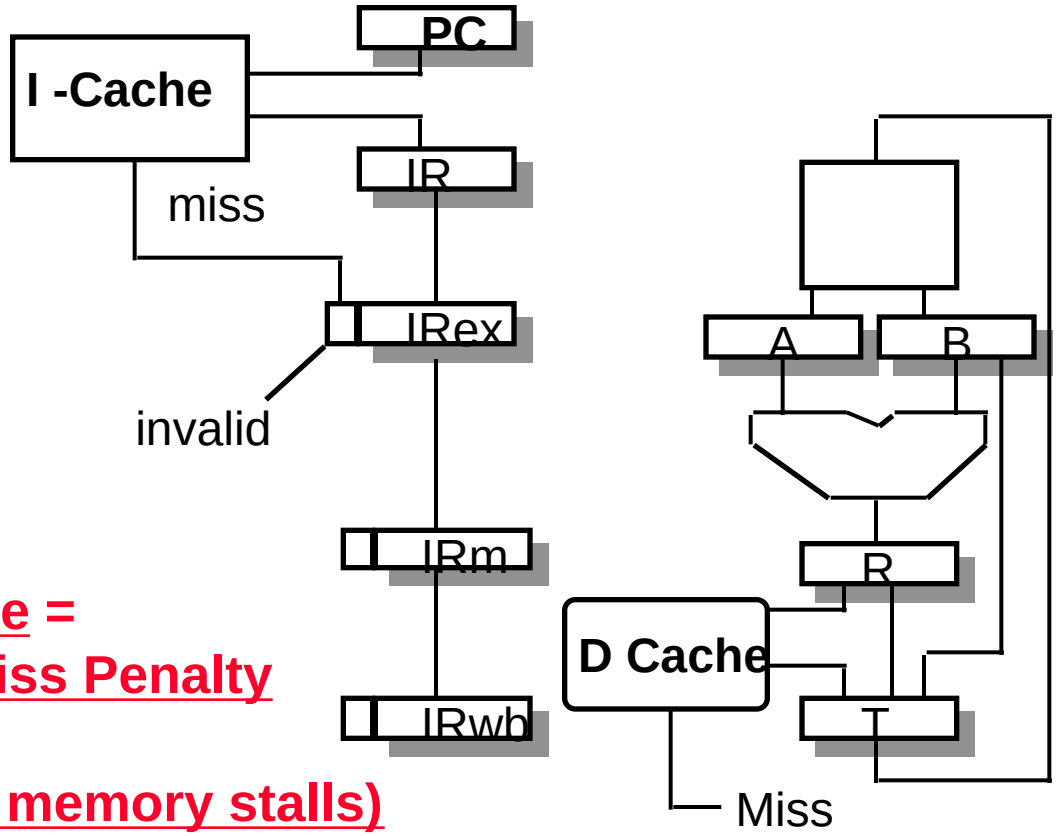
- N-way Set Associative Cache v. Direct Mapped Cache:
  - N comparators vs. 1
  - Extra MUX delay for the data
  - Data comes AFTER Hit/Miss
- In a direct mapped cache, Cache Block is available BEFORE Hit/Miss:
  - Possible to assume a hit and continue. Recover later if miss.



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)

Example: direct map allows miss signal after data

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.

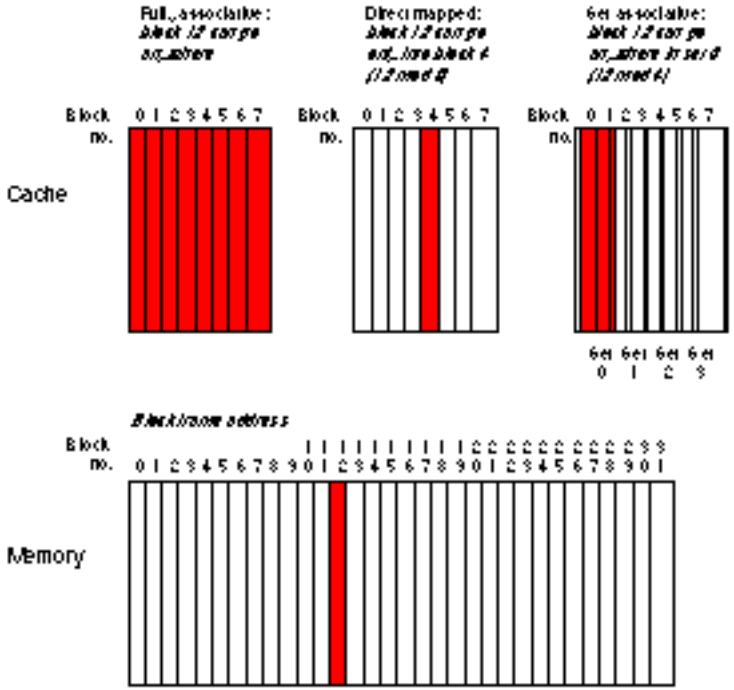
4 Questions for Memory Hierarchy

- **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?

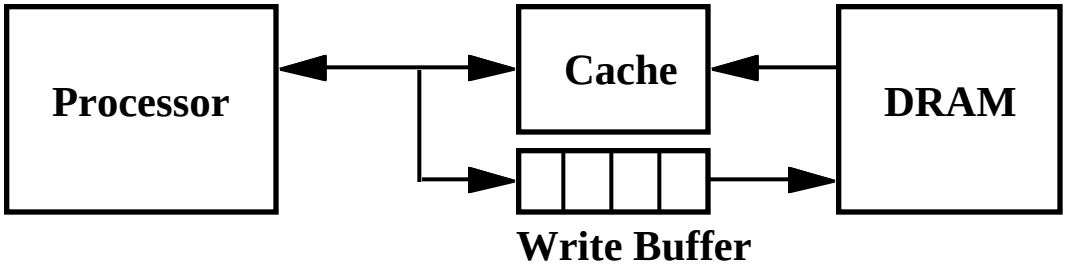
- Easy for Direct Mapped
- Set Associative or Fully Associative:
  - Random
  - LRU (Least Recently Used)

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?

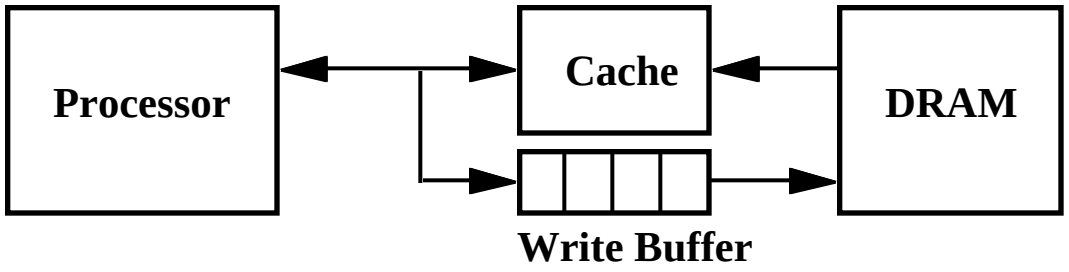
- ***Write through*—The information is written to both the block in the cache and to the block in the lower-level memory.**
- ***Write back*—The information is written only to the block in the cache. The modified cache block is written to main memory only when it is replaced.**
  - **is block clean or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

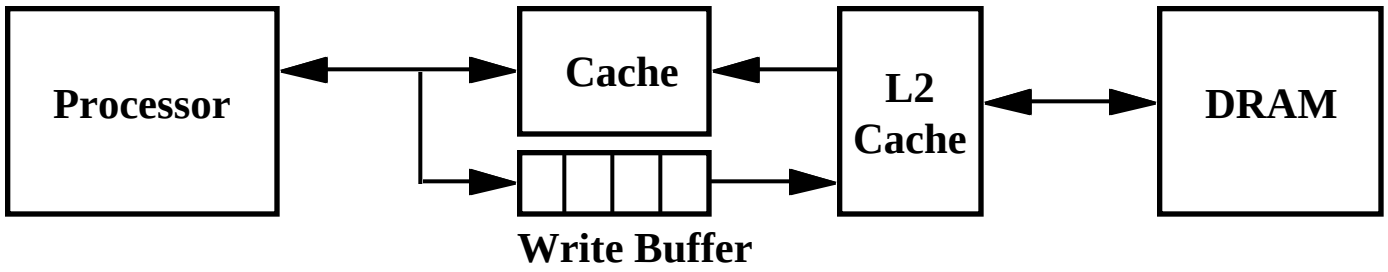


- A Write Buffer is needed between the Cache and Memory
  - Processor: writes data into the cache and the write buffer
  - Memory controller: write contents of the buffer to memory
- Write buffer is just a FIFO:
  - Typical number of entries: 4
  - Works fine if: Store frequency (w.r.t. time) << 1 /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

Write Buffer Saturation



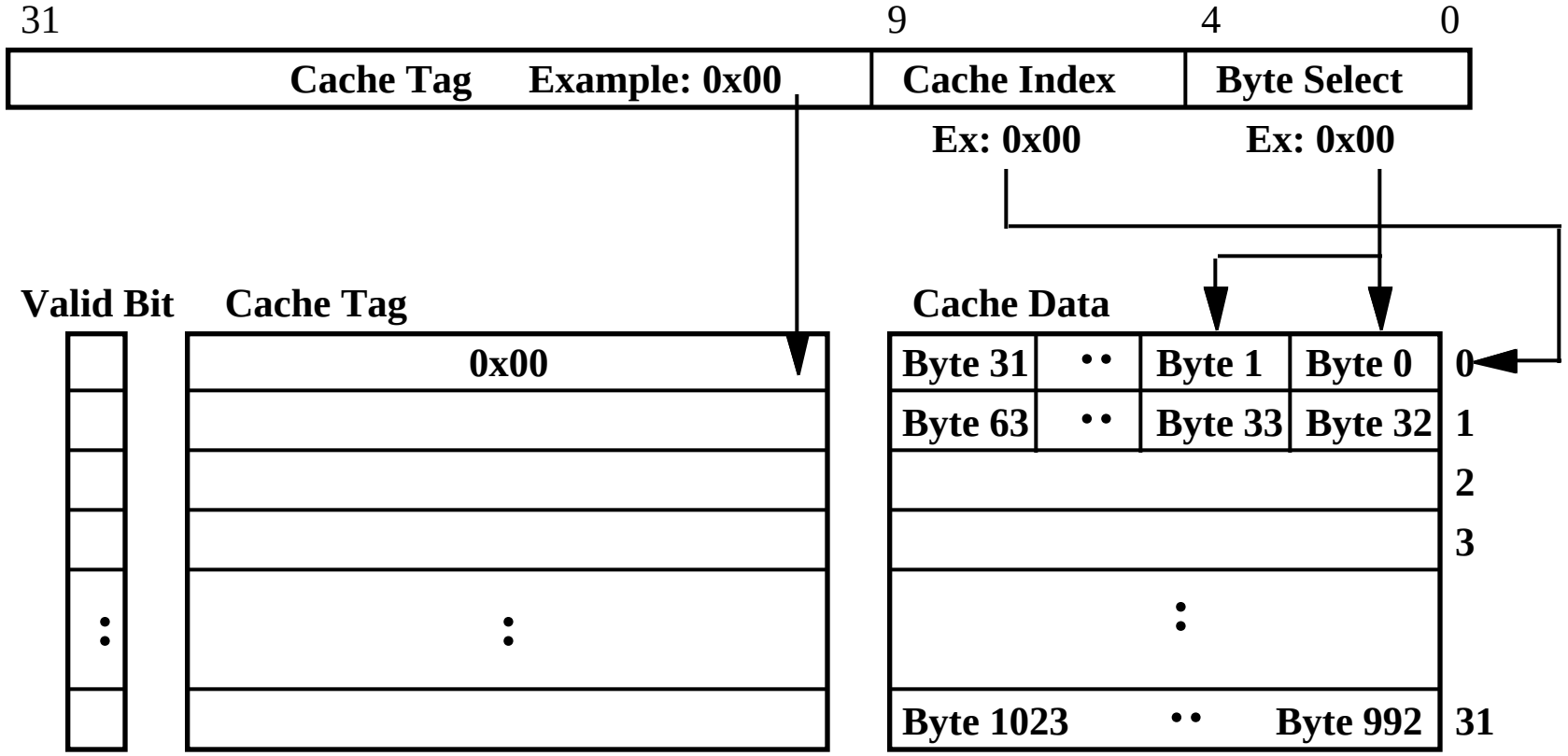
- **Store frequency (w.r.t. time) -> 1 / DRAM write cycle**
  - **If this condition exist for a long period of time (CPU cycle time too quick and/or too many store instructions in a row):**
    - **Store buffer will overflow no matter how big you make it**
    - **The CPU Cycle Time <= DRAM Write Cycle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



Write-miss Policy: Write Allocate versus Not Allocate

° **Assume: a 16-bit write to memory location 0x0 and causes a miss**

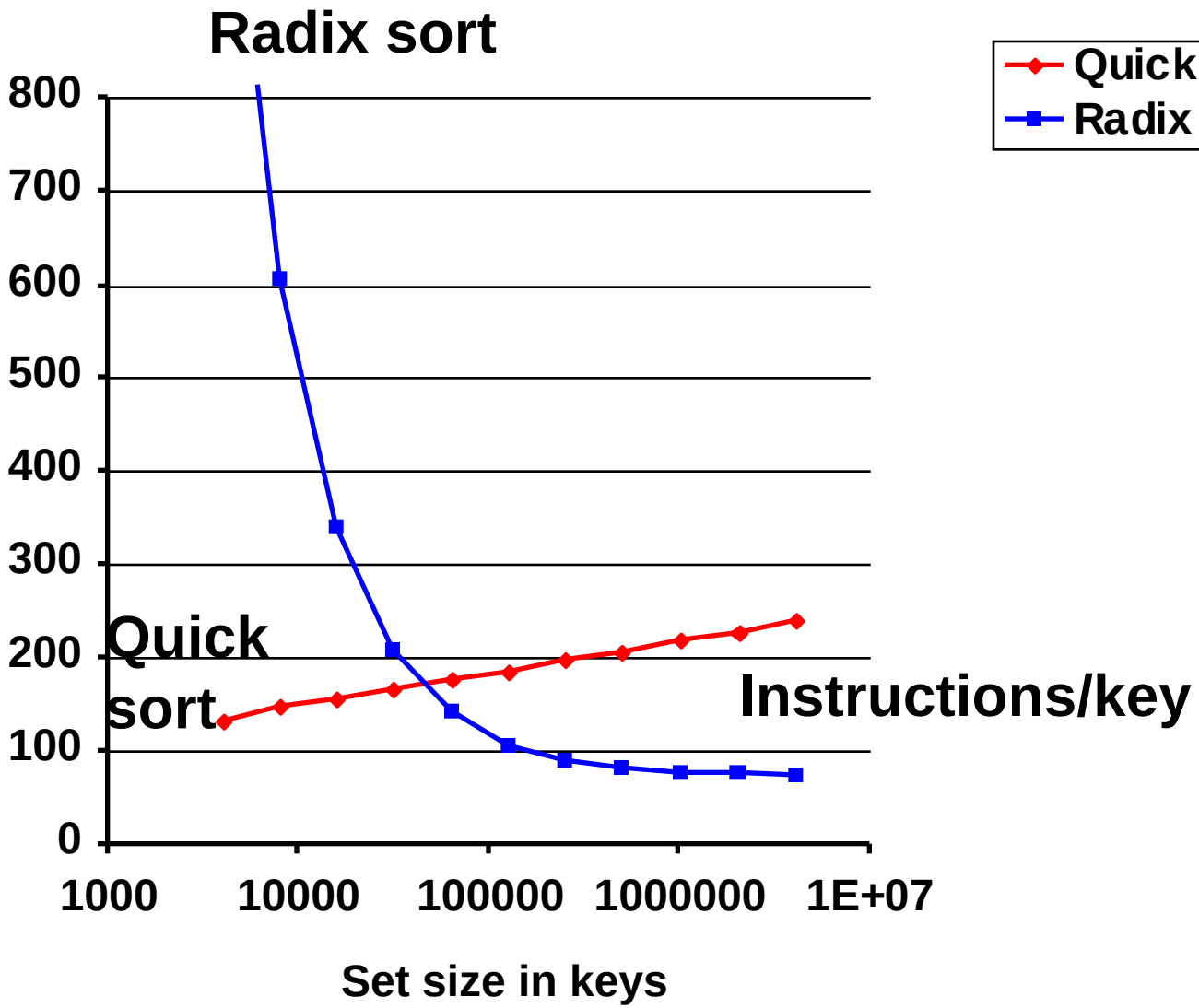
- **Do we read in the block?**
  - **Yes: Write Allocate**
  - **No: Write Not Allocate**



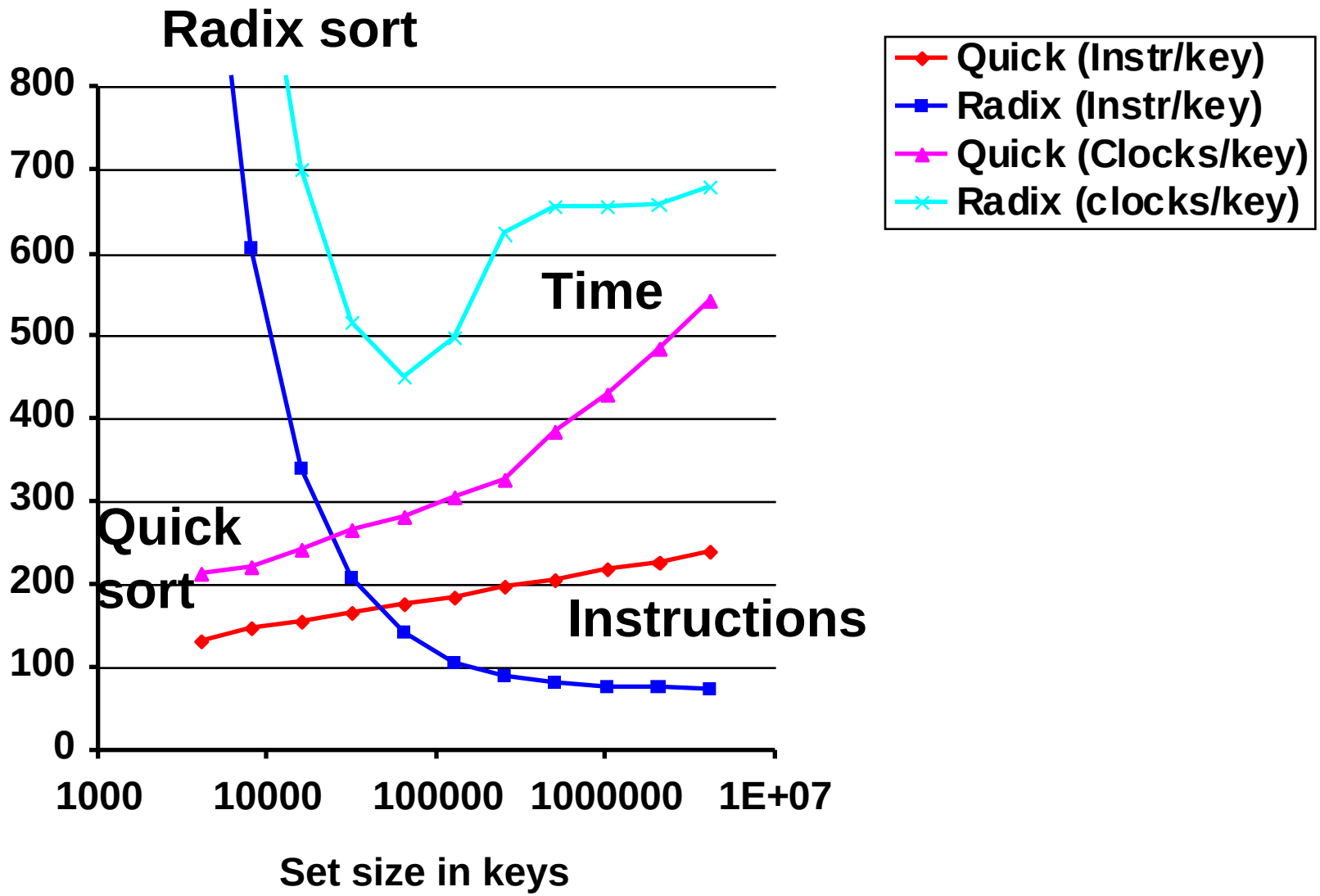
Impact of Memory Hierarchy on Algorithms

- 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

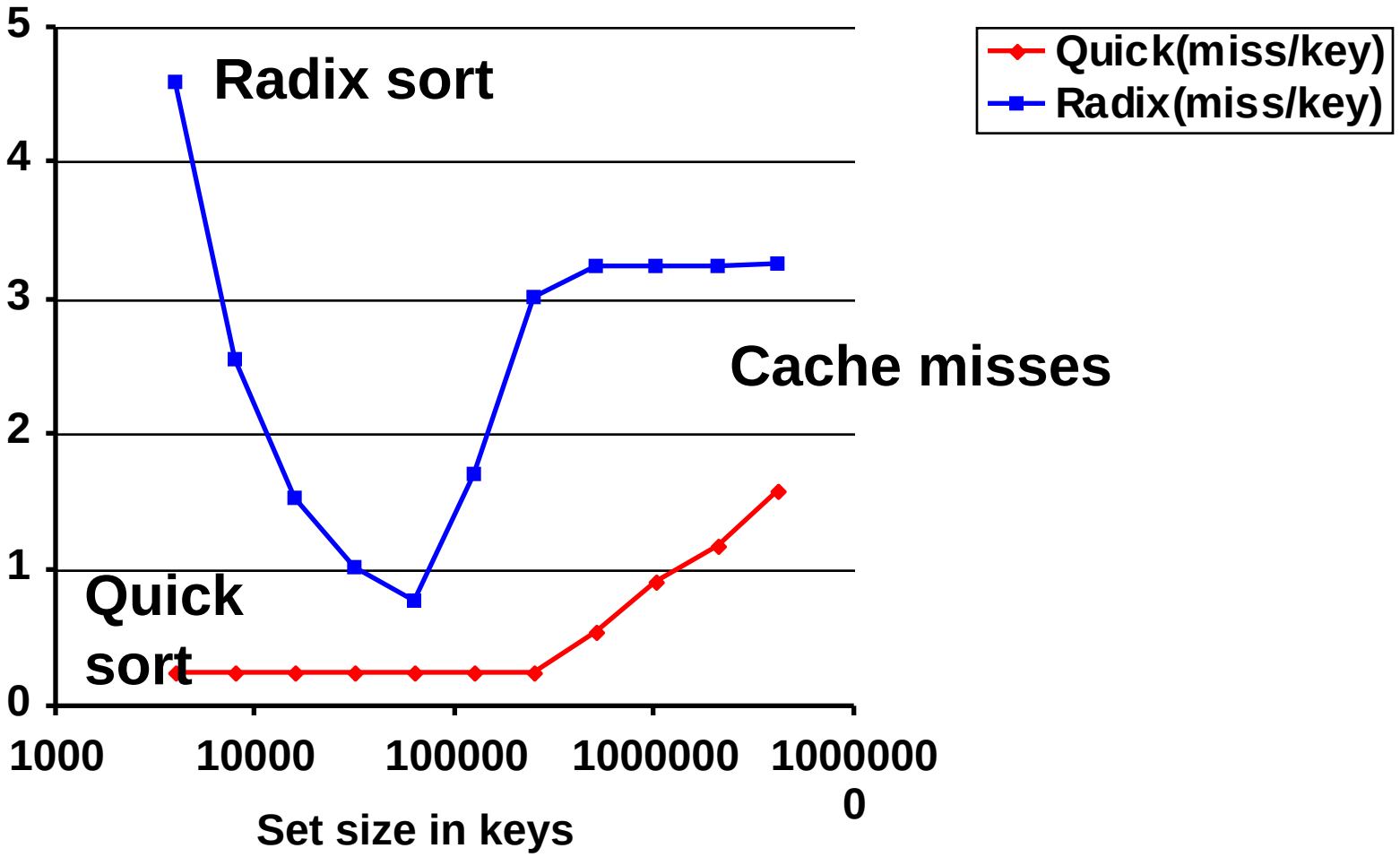
Quicksort vs. Radix as vary number keys: Instructions



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

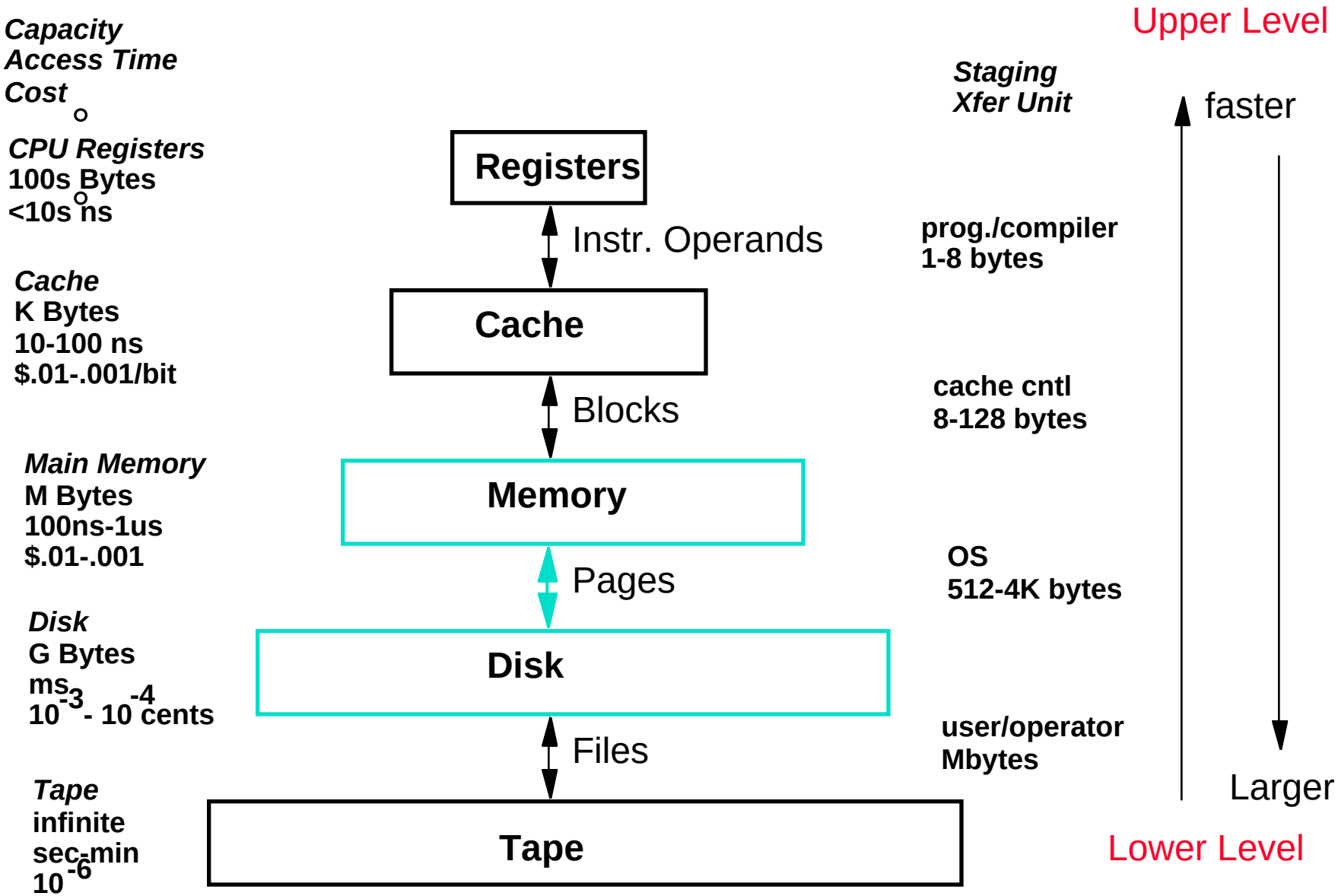


Quicksort vs. Radix as vary number keys: Cache misses



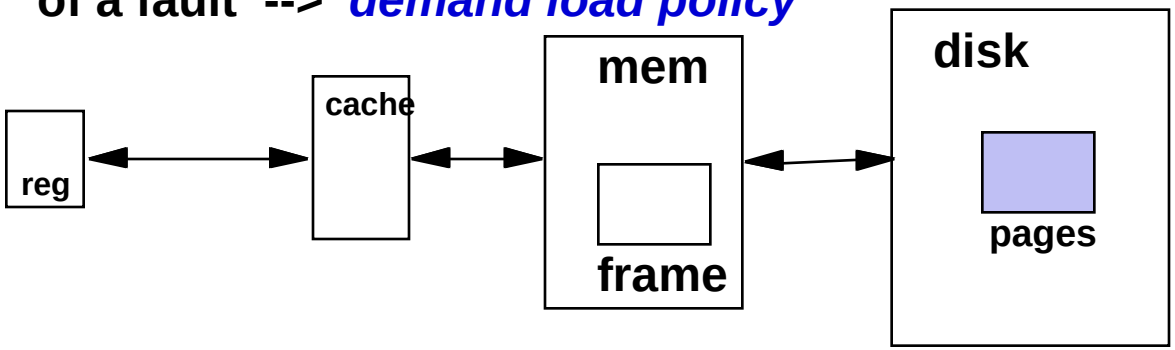
What is proper approach to fast algorithms?

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

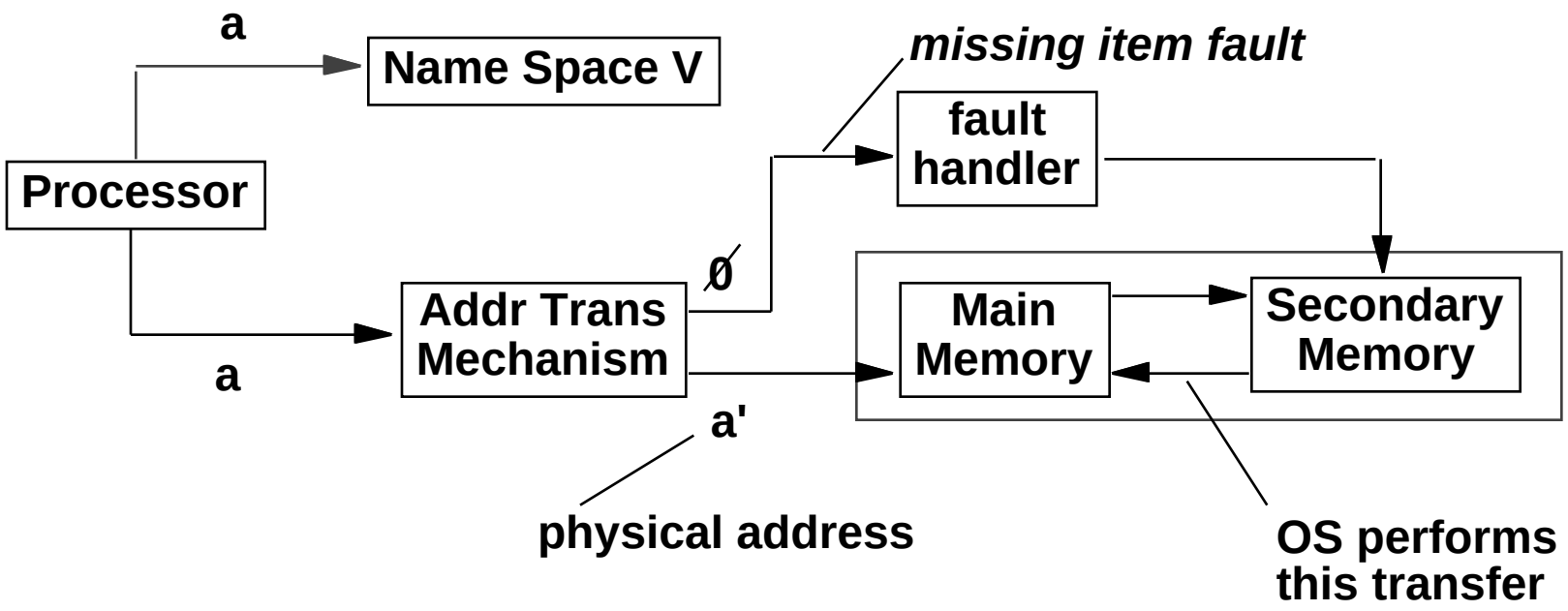
V = {0, 1, . . . , n - 1} virtual address space
M = {0, 1, . . . , m - 1} physical address space

n > m

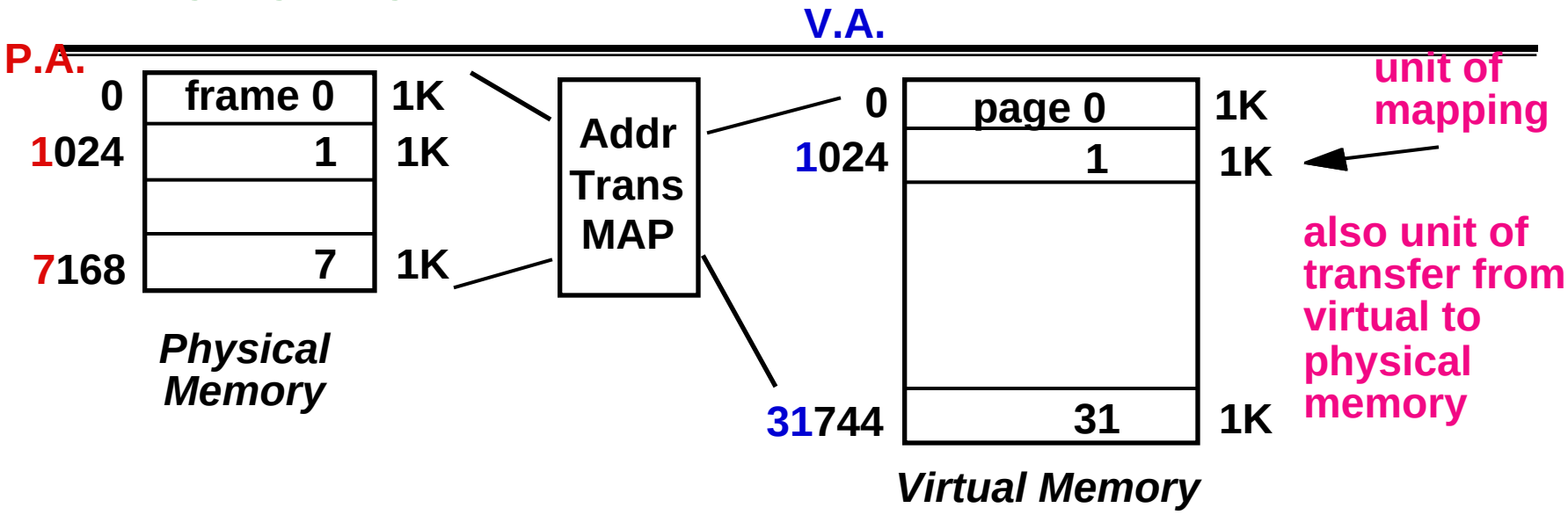
MAP: V --> M U {ø} address mapping function

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

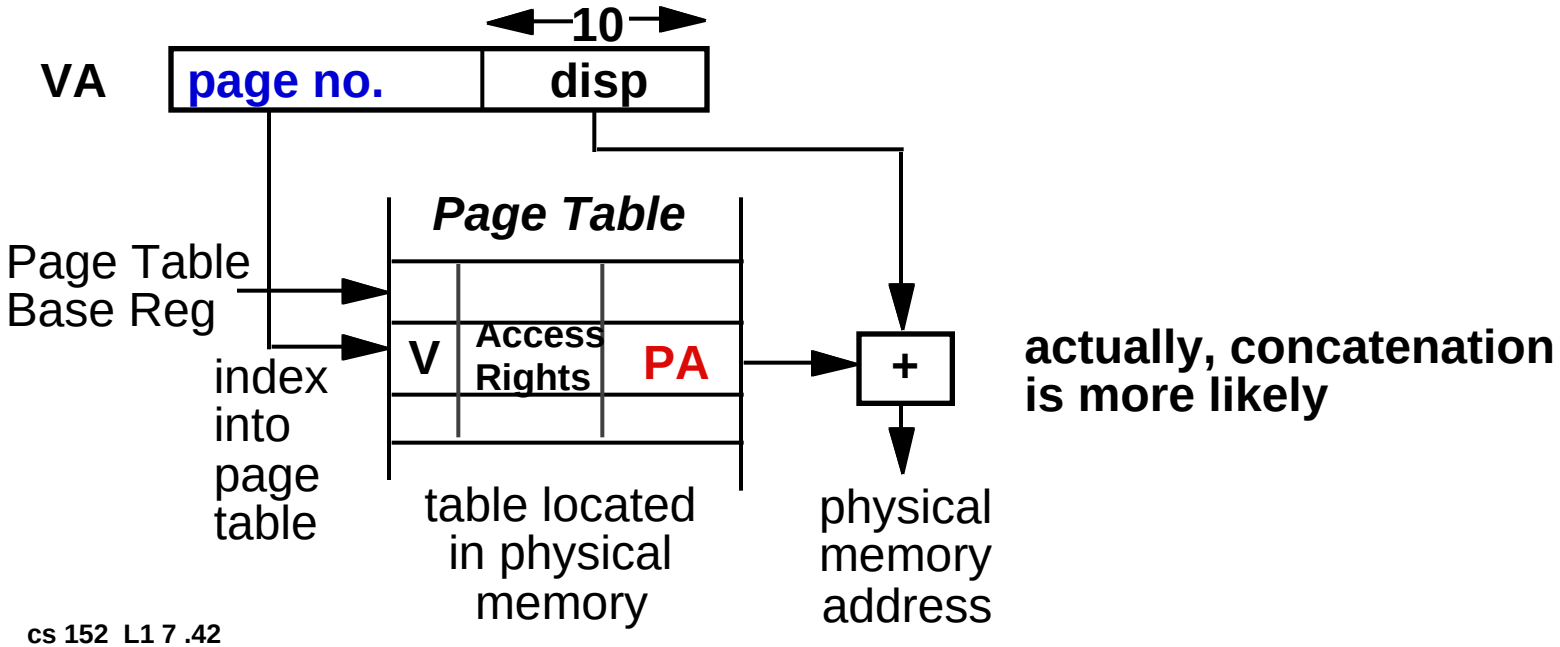
= ø if data at virtual address a is not present in M



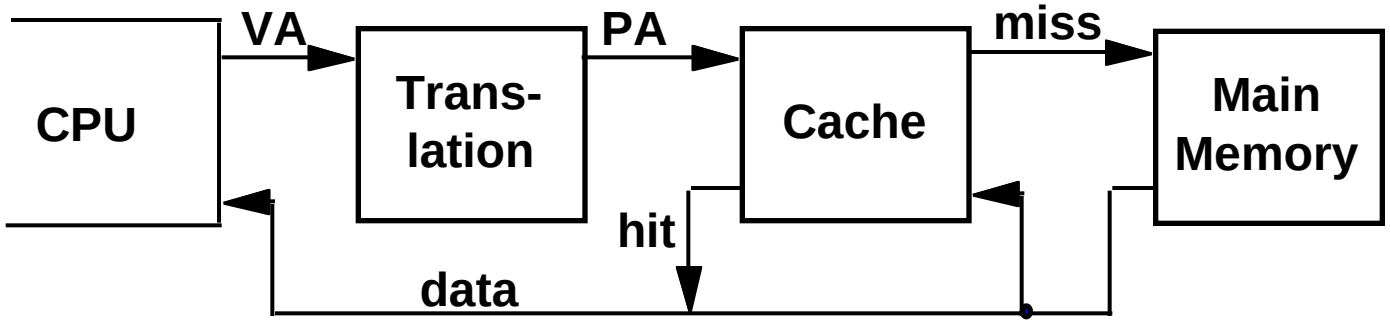
Paging Organization



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*

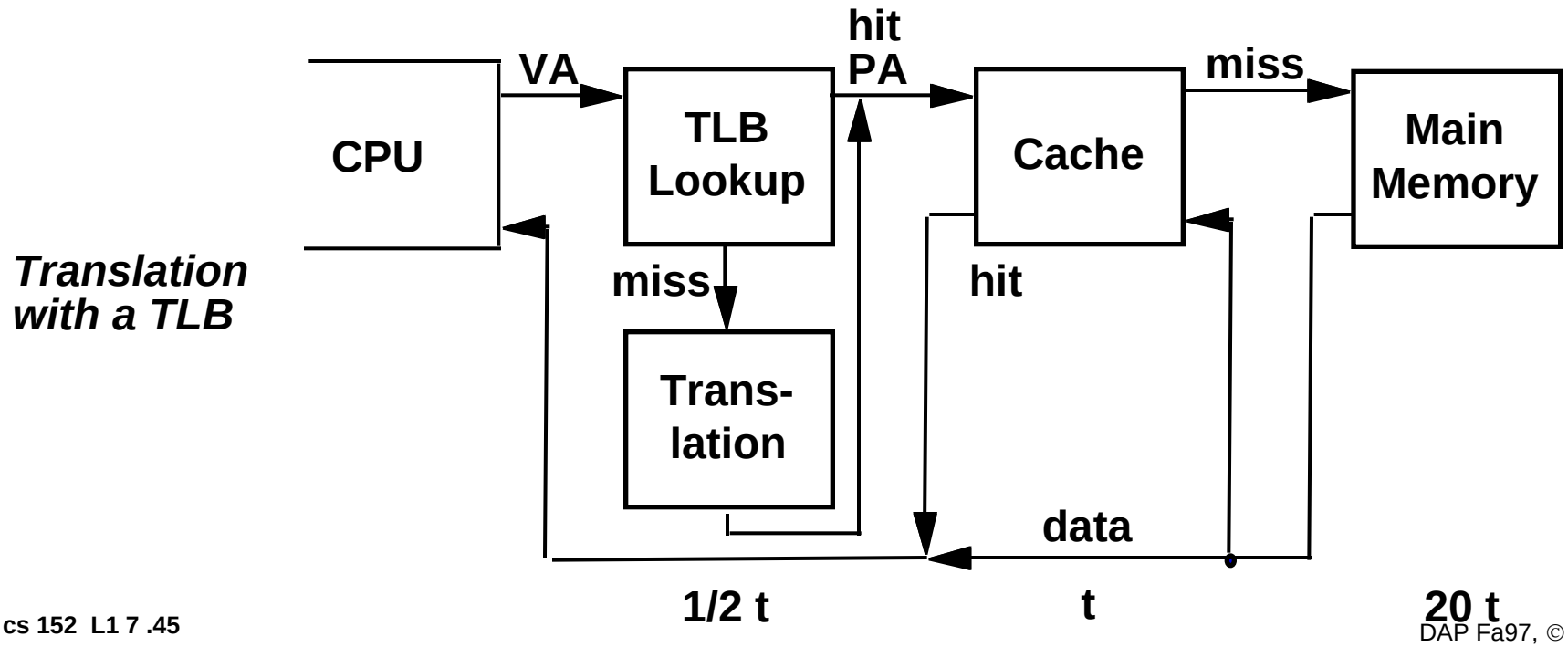
Virtual Address	Physical Address	Dirty	Ref	Valid	Access

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

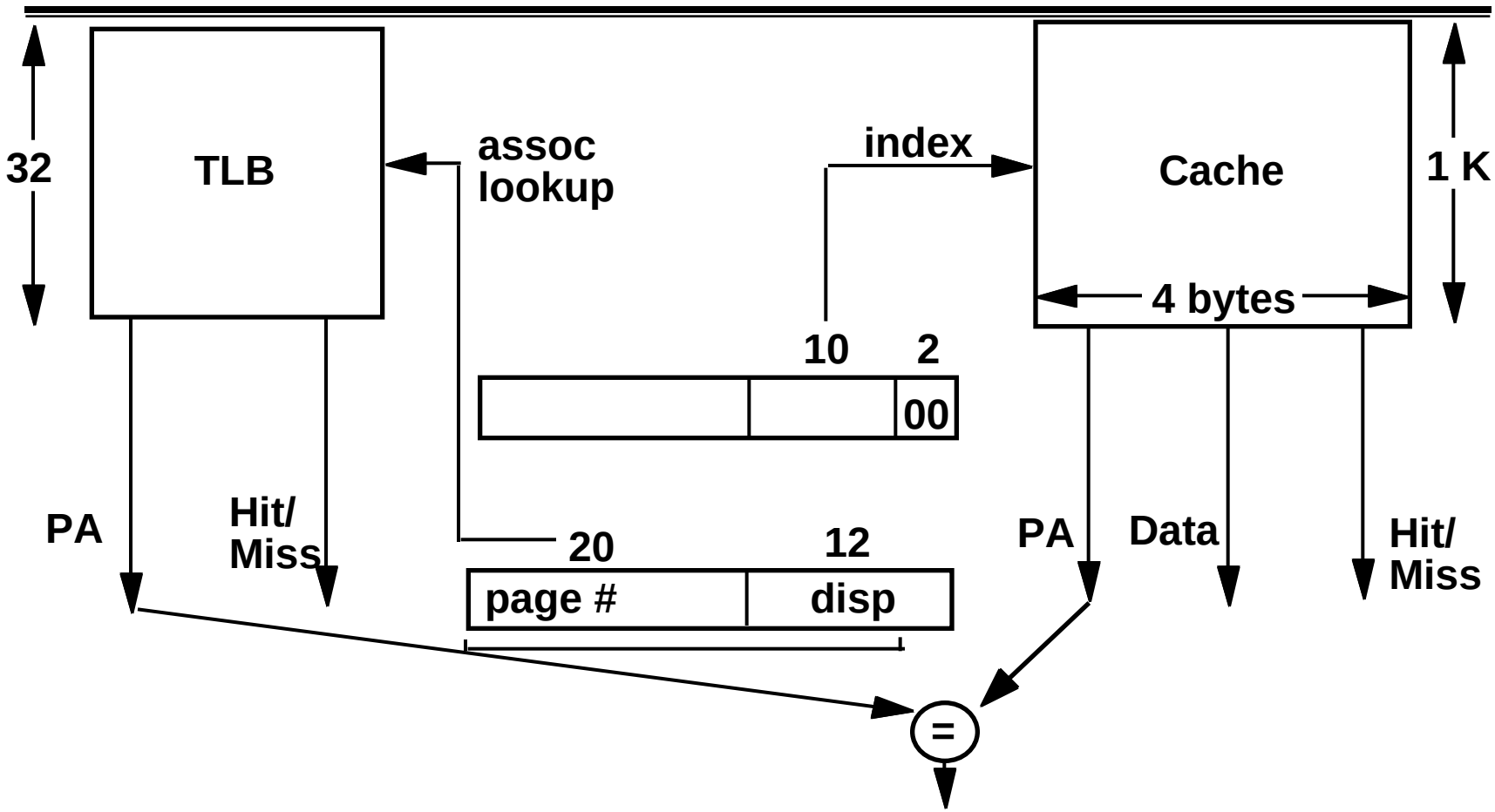
Reducing Translation Time

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

They overlap the cache access with the TLB access

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



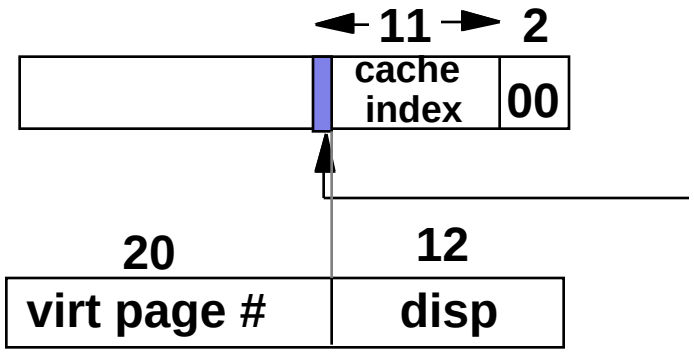
IF cache hit AND (cache tag = PA) then deliver data to CPU
ELSE IF [cache miss OR (cache tag = PA)] and TLB hit THEN
access memory with the PA from the TLB
ELSE do standard VA translation

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

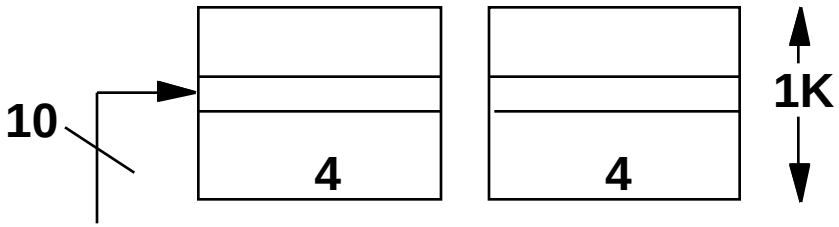
Example: suppose everything the same except that the cache is increased to 8 K bytes instead of 4 K:



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

Solutions:

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



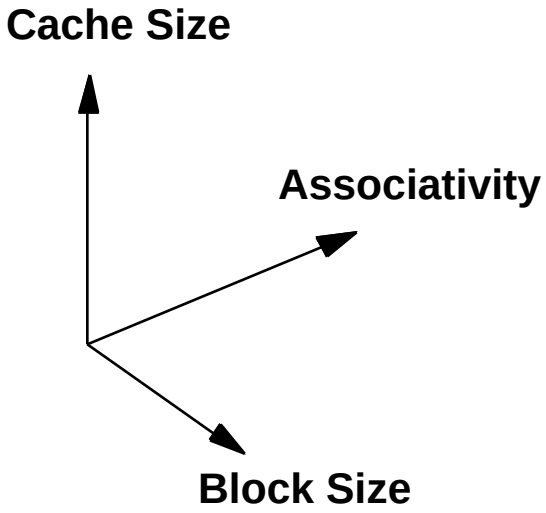
2 way set assoc cache

Summary #1/ 4:

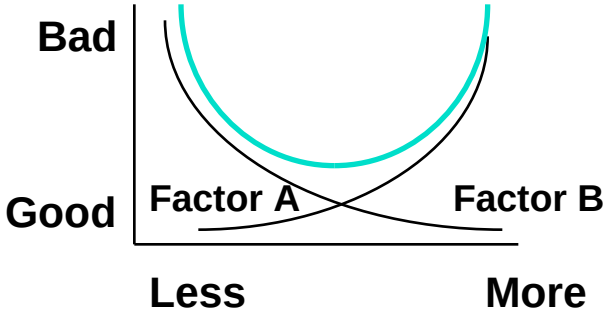
- The Principle of Locality:
  - Program likely to access a relatively small portion of the address space at any instant of time.
    - Temporal Locality: Locality in Time
    - Spatial Locality: Locality in Space
- Three Major Categories of Cache Misses:
  - Compulsory Misses: sad facts of life. Example: cold start misses.
  - Conflict Misses: increase cache size and/or associativity.
    Nightmare Scenario: ping pong effect!
  - Capacity Misses: increase cache size
- 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 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; 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?