

# Memory & Caches

Dr. Arjan Durrresi  
Louisiana State University  
Baton Rouge, LA 70810  
Durrresi@Csc.LSU.Edu

These slides are available at:  
[http://www.csc.lsu.edu/~durrresi/CSC7080\\_06/](http://www.csc.lsu.edu/~durrresi/CSC7080_06/)



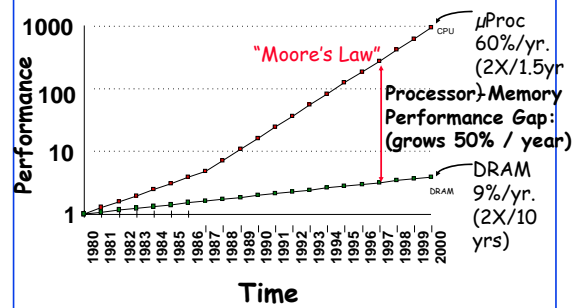
- Basics of Caches
- Measuring and Improving Cache Performance
- Framework for Memory Hierarchies

# Memory

- "You can never be too rich, too thin, or have too much memory"
- Create the illusion of unlimited fast memory
- Analogy with the use of library books
  - Get the books you need once
  - Or go to library each time you need new data
- Temporal Locality: if an item is referenced, it will tend to be referenced again soon.
- Spatial Locality: If an item is referenced, items whose addresses are close by will tend to be referenced soon
- Locality comes from natural program structure such as loops, arrays etc.
- Memory hierarchy: multiple levels of memory with different speeds and sizes

# Recap: Who Cares About the Memory Hierarchy?

## Processor-DRAM Memory Gap (latency)

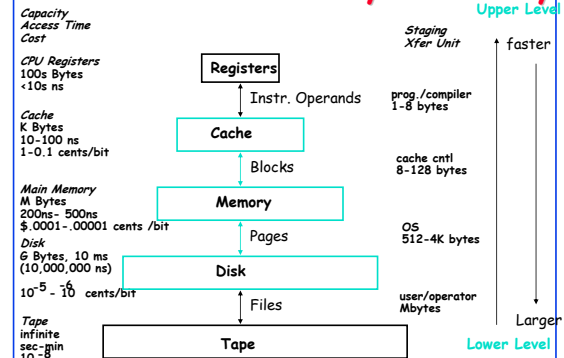


# Memories: Review

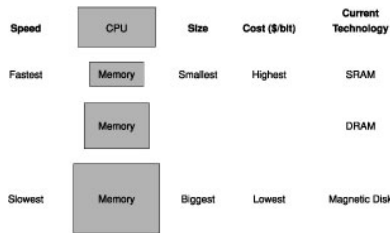
- SRAM:
  - value is stored on a pair of inverting gates
  - very fast but takes up more space than DRAM (4 to 6 transistors)
- DRAM:
  - value is stored as a charge on capacitor (must be refreshed)
  - very small but slower than SRAM (factor of 5 to 10)



# Levels of the Memory Hierarchy



## Basic Structure of Memory Hierarchy



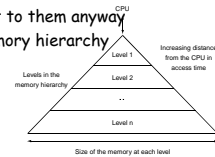
The user has the illusion of a memory as large as the largest level, But can be accessed as if it were all built from the fastest one

## Exploiting Memory Hierarchy

- Users want large and fast memories!

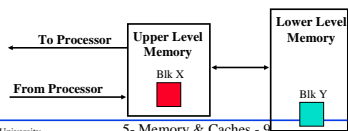
SRAM access times are .5 - 5ns at cost of \$4000 to \$10,000 per GB.  
 DRAM access times are 50-70ns at cost of \$100 to \$200 per GB.  
 Disk access times are 5 to 20 million ns at cost of \$.50 to \$2 per GB.

- Try and give it to them anyway
  - build a memory hierarchy



## 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!)



## 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 locality for speed

## Locality

- A principle that makes having a memory hierarchy a good idea
- If an item is referenced,
  - temporal locality: it will tend to be referenced again soon
  - spatial locality: nearby items will tend to be referenced soon.

Why does code have locality?

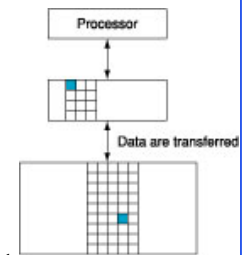
- Our initial focus: two levels (upper, lower)
  - block: minimum unit of data
  - hit: data requested is in the upper level
  - miss: data requested is not in the upper level

## Memory Hierarchy

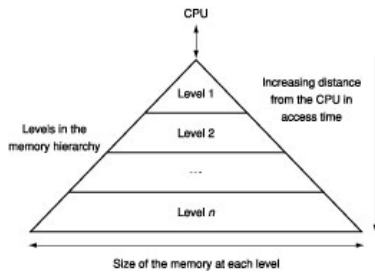
Miss rate: the fraction of memory Access not found in a level of the Memory hierarchy

Hit time: the time required to Access a level of the hierarchy, Including the time needed to determine The access is a hit or a miss

Miss penalty: the time required to fetch a block into a level of the memory Hierarchy from the lower level, including the time to access the block, transmit it, and insert in the needed level



## Memory Hierarchy



## Cache

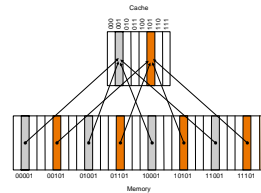
- Two issues:
    - How do we know if a data item is in the cache?
    - If it is, how do we find it?
  - Our first example:
    - block size is one word of data
    - "direct mapped"
- For each item of data at the lower level, there is exactly one location in the cache where it might be.
- e.g., lots of items at the lower level share locations in the upper level

## 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 × 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)

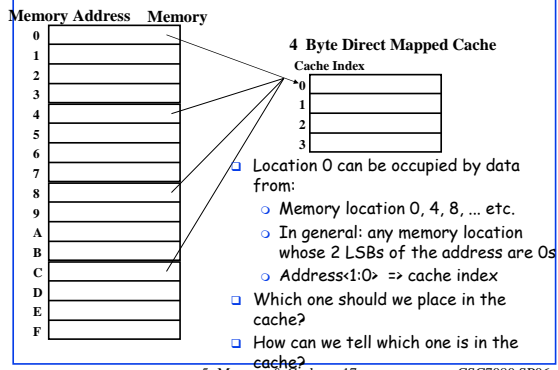
## Direct Mapped Cache

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



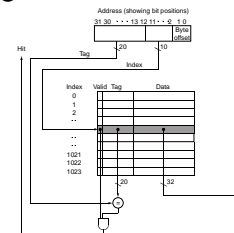
(Block Address) modulo (Number of cache blocks in the cache)

## Simplest Cache: Direct Mapped



## Direct Mapped Cache

- For MIPS:



- Index** - used to select the word
- Tag** - used to compare with the value of tag field of the cache
- Valid bit** - indicate whether an entry contains a valid address

## Cache

- How many bits are required for a direct-mapped cache with 16 KB of data and 4-word blocks, assuming 32-bit address?
- 16 KB  $\rightarrow$  4Kwords,  $2^{12}$  words, with a block size of 4 words  $\rightarrow 2^{10}$  blocks
- Each block has  $4 \times 32 = 128$  bits data plus a tag
- If number of blocks =  $2^n$ , word =  $2^m$ , the tag field =  $32 - n - m - 2$ 
  - Tag =  $32 - 10 - 2 - 2 = 18$
- Total cache =  $2^{10} \times (128 + (32 - 10 - 2 - 2) \times 1) = 2^{10} \times 147 = 147$  Kbits = 18.4 KB
- So 18.4 KB for a 16 KB cache, total number of bits is 1.15 times as needed just for the storage of data

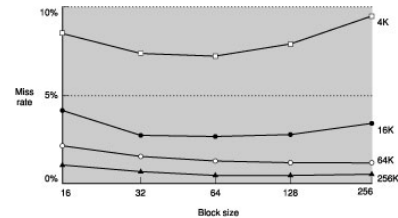
## Mapping to Cache

- A cache with 64 blocks and a block size of 16 bytes.
- What block number does byte address 1200 map to?
- (Block Address) modulo (Number of cache blocks in the cache)
- The address of the block =  $\frac{\text{Byte address}}{\text{Bytes per block}} = 75$
- That maps to cache block number (75 modulo 64) = 11
- This block maps all addresses between 1200 and 1215
- Larger blocks exploit spatial locality to lower miss rate
- But when block too big relatively to cache - more competition for those blocks - the block will be out of cache before many of its words are used

## Block Size

- Increasing the block size - cost of a miss increases
- Miss penalty - time required to fetch the block from the next level of hierarchy and load it into the cache.
- Time to fetch
  - Latency to the first word
  - Transfer time for the rest of the block
- When the block size increases
  - The improvement in the miss rate starts to decrease
  - The increase in miss penalty  $\gt$  improvement in miss rate
  - The cache performance decreases

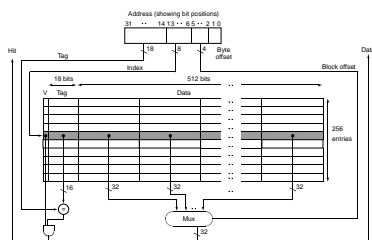
## Miss Rate vs. Block Size



Each line represents a cache of different size  
The miss rate goes up if the block size is too large relative to cache size.

## Direct Mapped Cache

- Taking advantage of spatial locality:



For SPEC200 : Instruction miss rate = 0.4%, Data miss rate = 11.4%,  
Effective combined miss rate = 3.2%

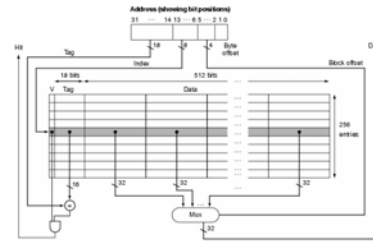
## Hits vs. Misses

- Read hits
  - this is what we want!
- Read misses
  - stall the CPU, fetch block from memory, deliver to cache, restart
  - Similar to pipeline stalls. What is the difference among them?
- Steps for cache (instruction) miss :
  - Send the original PC value (current PC -4) to the memory
  - Instruct main memory to perform a read and wait for the memory to complete access.
  - Write the cache entry, putting the data from memory in the data portion of the entry, writing the upper bits of the address from ALU into the tag field, and turning the valid bit on.
  - Restart the instruction execution at the first step, which will refetch the instruction. this time in the cache.

## Writes

- Write hits:
  - Avoid inconsistency between cache and memory
  - can replace data in cache and memory (write-through)
    - Writes very slow - 100 processor clock
    - SPEC2000 integer - 10% of instructions are stores
    - If CPI without writes is 1.0, spending 100 extra cycles on every writes:  $1.0 + 100 \times 10\% = 11$
  - Use write in cache and in buffer - a write buffer stores the data while it is waiting to be written into memory.
    - If buffer is full - processor will stall
    - To avoid stalling - increase the depth of the buffer
  - write the data only into the cache (write-back the cache later). It written into memory when replaced. More complex mechanism.
- Write misses:
  - read the entire block into the cache, then write the word
  - Write the word into memory
  - Write-back
    - Uses a store buffer. Two cycles - one to write in the buffer, second to write from the buffer to the cache.
    - Write-through - can be done in one cycle

## FastMATH Example



FastMATH – a fast embedded microprocessor – MIPS architecture  
It has separate instruction and data cache. Each cache is 16KB, or 4K words, with 16-word blocks.

## Read

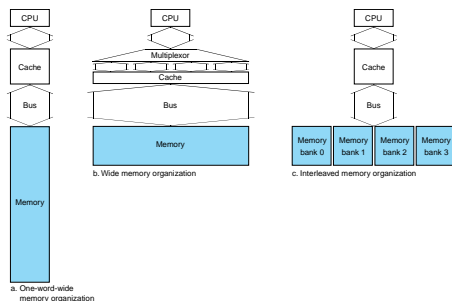
- Send address to the appropriate cache. The address comes from the PC (for an instruction) or from ALU (for data)
- If it is a hit, the requested word is available on the data lines. Since there are 16 words in the desired block, it is selected using a multiplexor.
- If it is a miss, the address is sent to the main memory. When the memory returns the data, it is written into the cache and then read.

## Writes

- FastMATH offers both write-through and write-back
- Operating systems decide which strategy to use for a given application
- It has a one-entry write buffer
- Miss rate for SPEC2000 benchmarks:
  - Instruction miss rate = 0.4%
  - Data miss rate = 11.4%
  - Effective combined miss rate = 3.2% . Depends on the frequency of data and instructions rate.

## Hardware Issues

- Make reading multiple words easier by using banks of memory

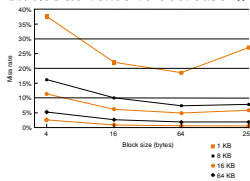


## Access Time

- 1 memory bus clock cycle to send the address
- 15 memory bus clock for each DRAM access initiated
- 1 memory bus clock cycle to send a word data
- If we have a cache of four words:
  - One-word-wide memory - Miss penalty =  $1 + 4 \times 15 + 4 \times 1 = 65$  memory bus clock cycles
  - Number of bytes transferred per bus clock for a single miss  $4 \times 4 / 65 = 2.5$
  - With 2 word width (Fig b, slide 18) miss penalty =  $1 + 2 \times 15 + 2 \times 1 = 33$
  - The bandwidth of a single miss is 0.48
  - With 4 word width (Fig b) miss penalty =  $1 + 15 + 1 = 17$
  - The bandwidth of a single miss is 0.94
- Try to have access initiated in parallel, use banks of memories (Fig c, slide 19)
  - Interleaving scheme:
    - Miss penalty =  $1 + 1 \times 15 + 4 \times 1 = 20$  memory bus clock cycles
    - This is an effective bandwidth per miss of 0.80 bytes per clock, or about

## Performance

- Increasing the block size tends to decrease miss rate:



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%
spicbe	1	1.2%	1.3%	1.2%
	4	0.3%	0.6%	0.4%

## Performance

- Simplified model:

execution time = (execution cycles + Memory-stall cycles) × cycle time

Memory-stall cycles = # of instructions × miss ratio × miss penalty

- Memory-stall = Read-stall cycles + Write-stall cycles
- Read-stall cycles = Reads/Program × Read miss rate × Read miss penalty
- Writes are more complex:
  - Write-through: write misses and buffer stalls (when buffer is full)
- Write-stall cycles = Write/Program × Write miss rate × Write miss penalty + Write buffer stalls
  - With a deep buffer (e.g. four or more words) and a memory of accepting writes at a rate higher than the average program write frequency - write buffer stall can be ignored

## Example of Cache Performance

- Assume an instruction miss rate for a program is 2% and a data cache miss rate is 4%
- If CPI is 2 without any memory stalls
- Miss penalty is 100 cycles for all misses
- How much faster it would run using SPECint2000 without miss?
- The number of memory miss cycles for instructions in terms of instructions I is :
  - Instruction miss cycle =  $I \times 2\% \times 100 = 2.00 \times I$
- The frequency of stores in SPECint2000 is 36%:
  - Data miss cycles =  $I \times 36\% \times 4\% \times 100 = 1.44 \times I$
- Total number of memory-stall cycles is  $2I + 1.44I = 3.44I$
- CPI =  $2 + 3.44 = 5.44$
- Performance improvement =  $5.44/2 = 2.72$ , amount of time in memory stalls =  $3.44/5.44 = 63\%$
- If we speed up the processor CPI = 1, Performance improvement =  $4.44$ , amount of time in memory stalls =  $3.44/4.44 = 77\%$

## Example (cont.)

- If we increase the processor clock rate by 2
- The miss penalty will be twice as many clocks - 200 clock cycles:
  - Total miss per instruction =  $2\% \times 200 + 36\% \times 4\% \times 200 = 6.88$
- The faster compute will have CPI =  $2 + 6.88 = 8.88$
- Let us compare the two computers:

Performance fast clock =  $\frac{\text{Execution time slow clock}}{\text{Execution time fast clock}}$

$$= \frac{IC \times CPI_{\text{slow}} \times \text{Clock cycles}}{IC \times CPI_{\text{fast}} \times \frac{\text{Clock cycles}}{2}} = \frac{5.44}{8.88/2} = 1.23$$

So the performance improvement is degraded from 2 to 1.23 because cache misses

## Example (cont.)

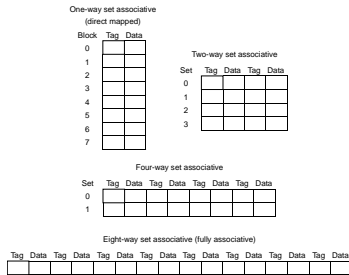
- If we improve both - lower CPI and higher clock
- The lower the CPI the more pronounced is the impact of stall cycles
- The higher the processor rate - larger miss penalty

## Improving Performance

- Two ways of improving performance:
  - decreasing the miss ratio
  - decreasing the miss penalty

*What happens if we increase block size?*

## Decreasing miss ratio with associativity



## Misses and Associativity in Caches

- Assume three caches, each of four one-word blocks
- One cache is fully associative, a second is two-way set associative, and the third is direct mapped
- Find the number of misses for each cache given the following sequence of addresses: 0, 8, 0, 6, 8
- Direct mapped:
  - Cache block = Block address modul 4
  - 0=0module4, 2=6 module 4, 0=8module 4

Address of memory Block accessed	Hit or miss	Contents of a cache blocks after reference			
		0	1	2	3
0	miss	Mem[0]			
8	miss	Mem[8]			
0	miss	Mem[0]			
6	miss	Mem[0]		Mem[6]	
8	miss	Mem[8]		Mem[6]	

## Misses and Associativity in Caches

- The set-associative has two sets (with indices 0 and 1) with two elements per set.
- Set = address module 2
- 0=0module2, 0=6module2, 0=8module2
- Replace the least recently used block within a set

Address of memory Block accessed	Hit or miss	Contents of a cache blocks after reference			
		Set0	Set0	Set1	Set1
0	miss	Mem[0]			
8	miss	Mem[0]	Mem[8]		
0	hit	Mem[0]	Mem[8]		
6	miss	Mem[0]	Mem[6]		
8	miss	Mem[8]	Mem[6]		

## Misses and Associativity in Caches

- The fully associative has four blocks. Three misses
- If we had 8 blocks in cache, there would be no replacements - three misses
- If we had 16 blocks - the three caches would have the same performance
- Cache size and associativity are related in determining cache

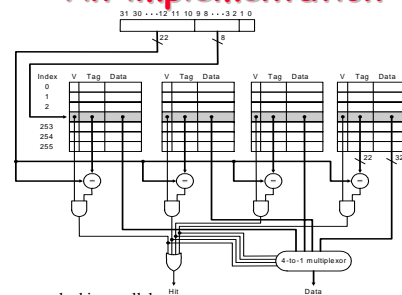
Address of memory Block accessed	Hit or miss	Contents of a cache blocks after reference			
		Block00	Block1	Block2	Block3
0	miss	Mem[0]			
8	miss	Mem[0]	Mem[8]		
0	hit	Mem[0]	Mem[8]		
6	miss	Mem[0]	Mem[8]	Mem[6]	
8	hit	Mem[8]	Mem[8]	Mem[6]	

## Performance

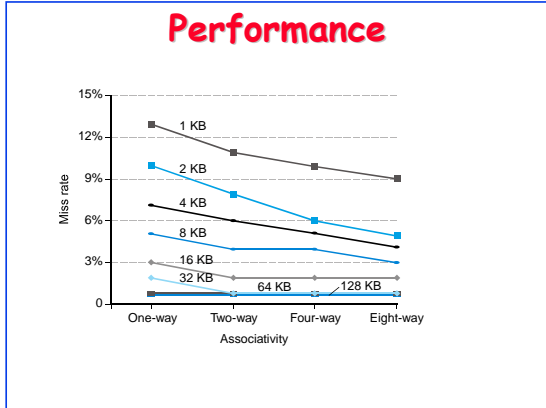
- SPEC2000 benchmarks for a 64KB data cache with a 16-word block. Associativity from one to eight way

Associativity	Data miss rate
1	10.3%
2	8.6%
4	8.3%
8	8.1%

## An implementation



Tags are searched in parallel  
 Each increase by two in associativity doubles the number of blocks per set and halves the number of sets – decreases the size of index by 1 bit and increase the size of tag by 1 bit

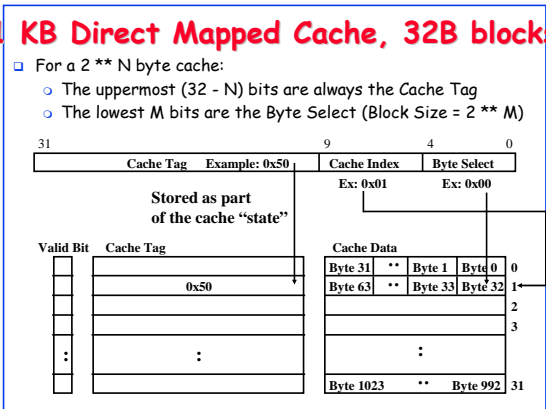


- ### Decreasing miss penalty with multilevel caches
- Add a second level cache:
    - often primary cache is on the same chip as the processor
    - use SRAMs to add another cache above primary memory (DRAM)
    - miss penalty goes down if data is in 2nd level cache
  - Example:
    - CPI of 1.0 on a 5 Ghz machine with a 2% miss rate, 100ns DRAM access
    - Adding 2nd level cache with 5ns access time decreases miss rate to .5%. How much faster will the processor be?

- ### Performance
- The miss penalty to main memory is:
 
$$\frac{100 \text{ ns}}{0.2 \frac{\text{ns}}{\text{Clock cycle}}} = 500 \text{ clock cycles}$$
  - The effective CPI with one level of caching is given:
 
$$\text{Total CPI} = \text{CPI} + \text{Memory-stall cycles per instruction}$$
  - For one level of caching:
 
$$\text{Total CPI} = 1 + 2\% \times 500 = 11$$
  - For two level of caching a miss in primary cache leads to a secondary cache or main memory. The miss penalty for an access to the second-level cache:
 
$$\frac{5 \text{ ns}}{0.2} = 25 \text{ clock cycles}$$

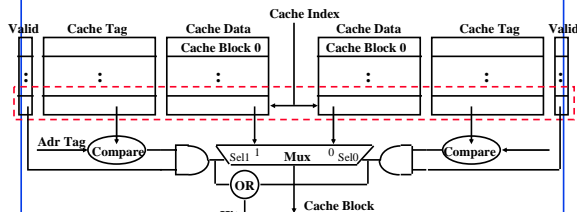
- ### Performance
- Total CPI = 1 Primary stalls per instr. + Secondary stalls per instr. =
 
$$= 1 + 2\% \times 25 + 0.5\% \times 500 = 4$$
- The processor with the secondary cache is faster by:
- $$11/4 = 2.8$$

- ### Improvements
- Using multilevel caches:
    - try and optimize the hit time on the 1st level cache
      - Small and fast
    - try and optimize the miss rate on the 2nd level cache
      - large



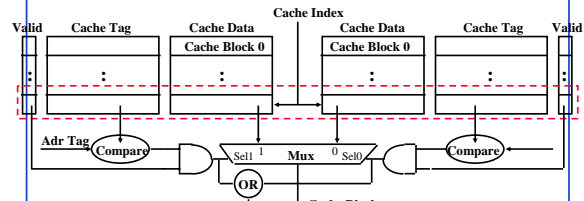
## Two-way Set Associative Cache

- 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.

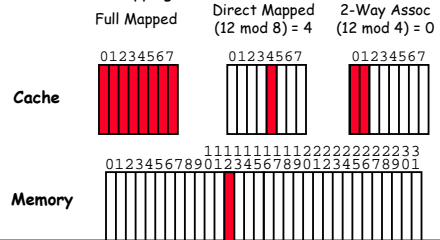


## 4 Questions for Memory Hierarchy

- 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



## 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

Block Address		Block Offset
Tag	Index	

## Q3: Which block should be replaced on a miss?

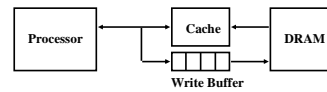
- Easy for Direct Mapped
- Set Associative or Fully Associative:
  - Random
  - LRU (Least Recently Used)

Assoc:	2-way	4-way	8-way			
Size	LRU	Ran	LRU	Ran	LRU	Ran
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



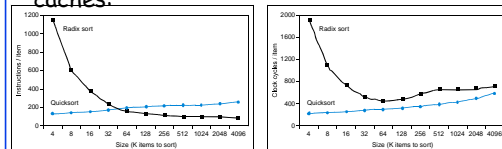
- A Write Buffer is needed between the Cache and Memory
  - Processor: writes data into the cache and the write buffer
  - Memory controller: write contents of the buffer to memory
- Write buffer is just a FIFO:
  - Typical number of entries: 4
  - Works fine if: Store frequency (w.r.t. time)  $\ll 1$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

## Impact of Memory Hierarchy on Algorithms

- Today CPU time is a function of (ops, cache misses) vs. just  $f(\text{ops})$ : 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

## Cache Complexities

- Not always easy to understand implications of caches:

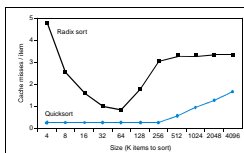


Theoretical behavior of Radix sort vs. Quicksort

Observed behavior of Radix sort vs. Quicksort

## Cache Complexities

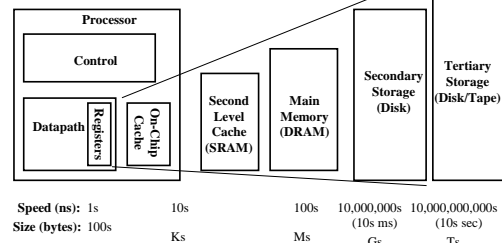
- Here is why:



- Memory system performance is often critical factor
- multilevel caches, pipelined processors, make it harder to predict outcomes
  - Compiler optimizations to increase locality sometimes hurt ILP

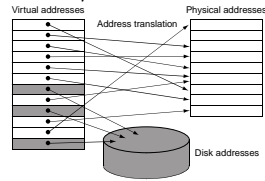
## A Modern Memory Hierarchy

- 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.



## Virtual Memory

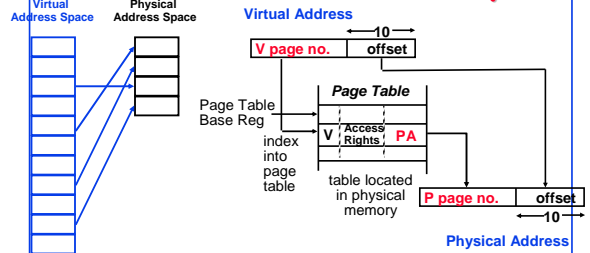
- Main memory can act as a cache for the secondary storage (disk)
- To allow efficient and safe sharing of memory among multiple programs
- Program to exceed the size of memory



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

Protection - by address translation

## What is virtual memory?



- Virtual memory => treat memory as a cache for the disk
- Terminology: blocks in this cache are called "Pages"
  - Typical size of a page: 1K – 8K
- Page table maps virtual page numbers to physical frames
  - "PTE" = Page Table Entry

## Three Advantages of Virtual Memory

- Translation:
  - Program can be given consistent view of memory, even though physical memory is scrambled
  - Makes multithreading reasonable (now used a lot!)
  - Only the most important part of program ("Working Set") must be in physical memory.
  - Contiguous structures (like stacks) use only as much physical memory as necessary yet still grow later.
- Protection:
  - Different threads (or processes) protected from each other.
  - Different pages can be given special behavior
    - (Read Only, Invisible to user programs, etc.)
  - Kernel data protected from User programs
  - Very important for protection from malicious programs => Far more "viruses" under Microsoft Windows
- Sharing:
  - Can map same physical page to multiple users ("Shared memory")

## Issues in Virtual Memory System Design

What is the size of information blocks that are transferred from secondary to main storage (M)? => **page size**  
(Contrast with physical block size on disk, i.e. **sector size**)

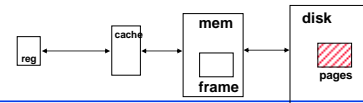
Which region of M is to hold the new block => **placement policy**

How do we find a page when we look for it? => **block identification**

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**

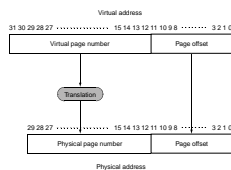
What do we do on a write? => **write policy**

Missing item fetched from secondary memory only on the occurrence of a fault => **demand load policy**

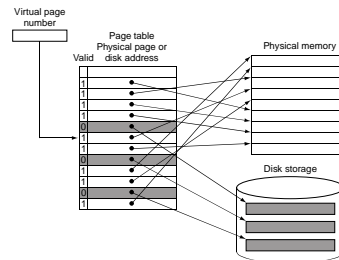


## Pages: virtual memory blocks

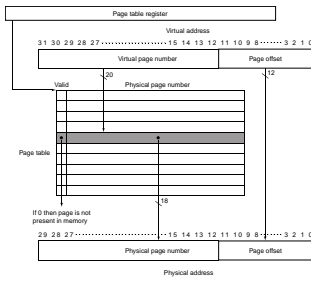
- Page faults: the data is not in memory, retrieve it from disk
  - huge miss penalty, thus pages should be fairly large (e.g., 4KB)
  - reducing page faults is important (LRU is worth the price)
  - can handle the faults in software instead of hardware
  - using write-through is too expensive so we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



## Page Tables

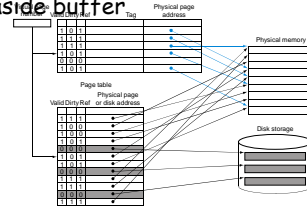


# Page Tables



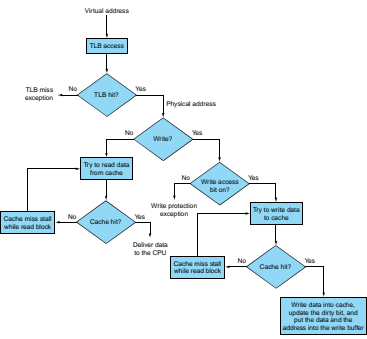
# Making Address Translation Fast

A cache for address translations: translation lookaside buffer

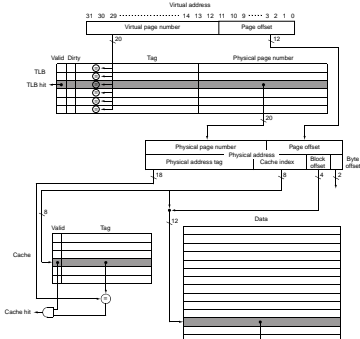


Typical values: 16-512 entries, miss-rate: .01% - 1% miss-penalty: 10 - 100 cycles

# TLBs and caches



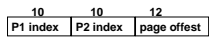
# TLBs and Caches



# Large Address Spaces

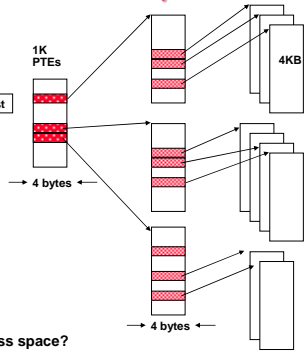
Two-level Page Tables

32-bit address:



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

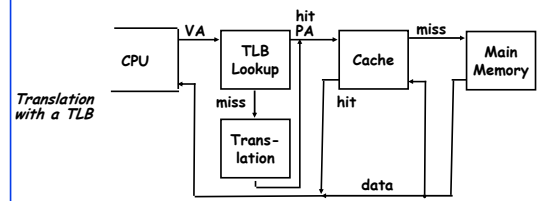
What about a 48-64 bit address space?



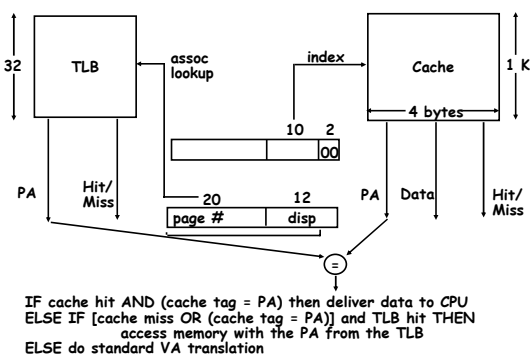
# Translation Look-Aside Buffers

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.



## Overlapped Cache & TLB Access

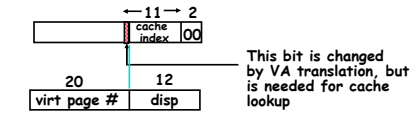


## Problems With Overlapped TLB Access

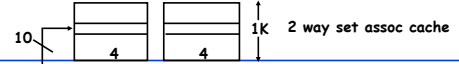
Overlapped access only works as long as the address bits used to index into the cache *do not change* as the result of VA translation

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:



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



## Summary #1/5: Control and Pipelining

- Control VIA State Machines and Microprogramming
- Just overlap tasks; easy if tasks are independent
- Speed Up  $\leq$  Pipeline Depth; if ideal CPI is 1, then:

$$\text{Speedup} = \frac{\text{Pipeline depth}}{1 + \text{Pipeline stall CPI}} \times \frac{\text{Cycle Time}_{\text{unpipelined}}}{\text{Cycle Time}_{\text{pipelined}}}$$

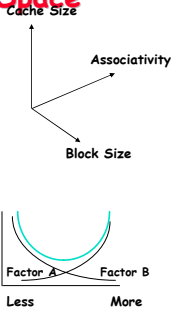
- Hazards limit performance on computers:
  - Structural: need more HW resources
  - Data (RAW, WAR, WAW): need forwarding, compiler scheduling
  - Control: delayed branch, prediction

## Summary #2/5: Caches

- 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 #3/5: 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 #4/5: 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!

