

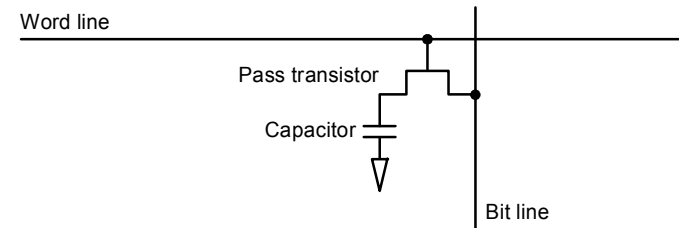
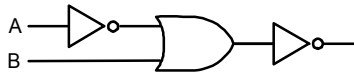
---

# Chapter Seven

# 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)



# Exploiting Memory Hierarchy

- **Users want large and fast memories!**

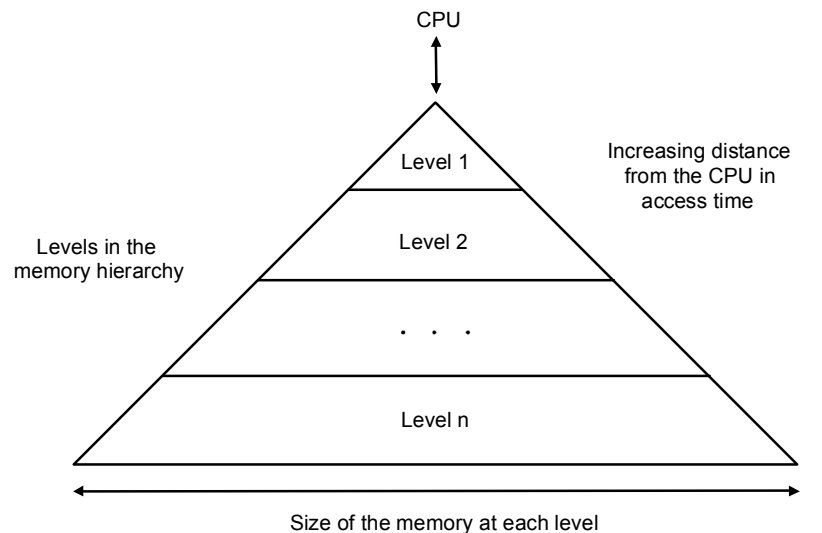
**SRAM access times are 2 - 25ns at cost of \$100 to \$250 per Mbyte.**

**DRAM access times are 60-120ns at cost of \$5 to \$10 per Mbyte.**

**Disk access times are 10 to 20 million ns at cost of \$.10 to \$.20 per Mbyte.**

1997

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



# 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

# 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"

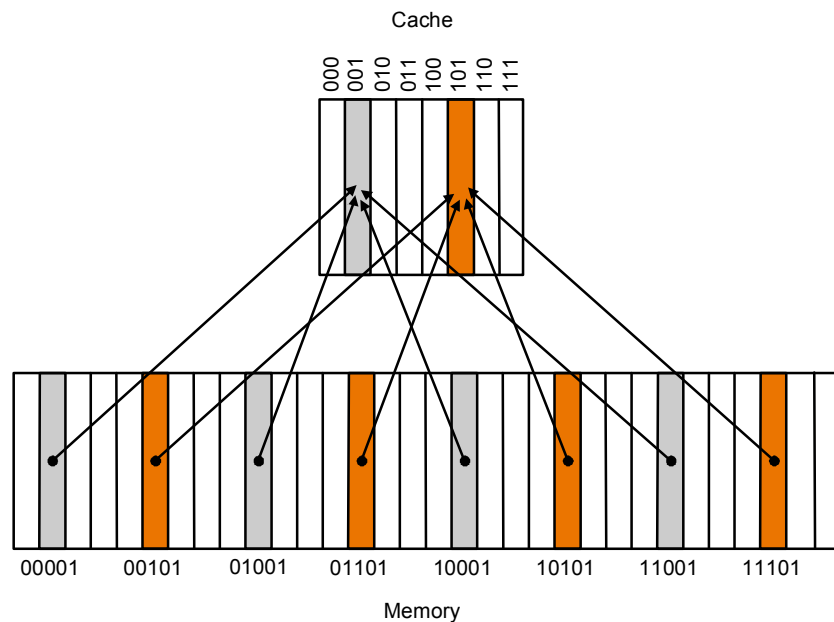


**For each item of data at the lower level,  
there is exactly one location in the cache where it might be.**

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

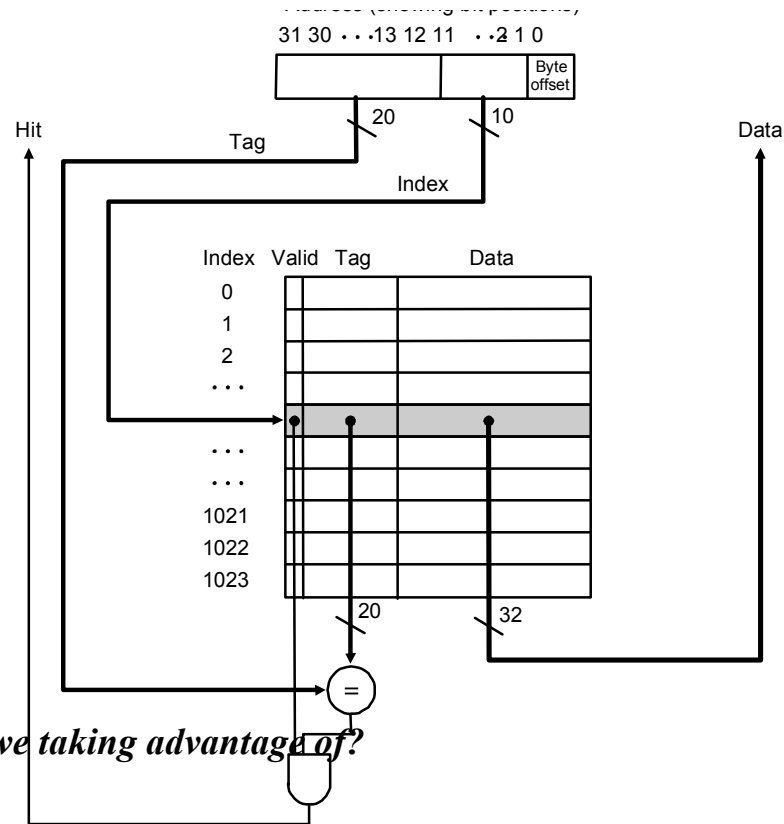
# Direct Mapped Cache

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



# Direct Mapped Cache

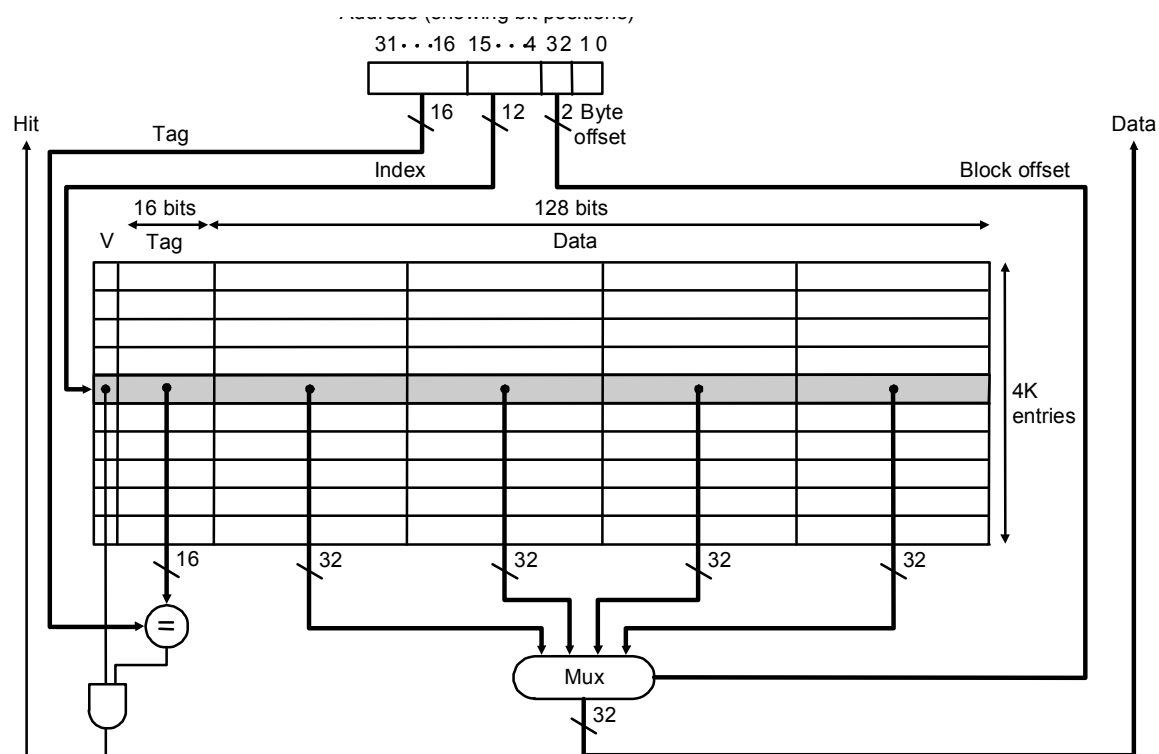
- For MIPS:



*What kind of locality are we taking advantage of?*

# Direct Mapped Cache

- Taking advantage of spatial locality:





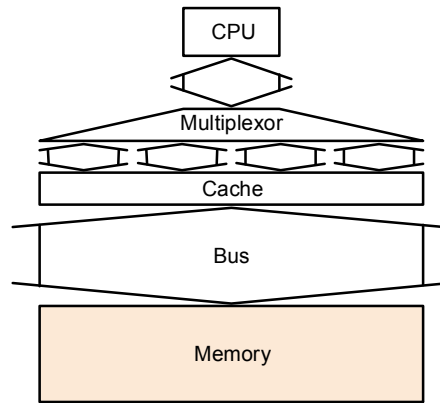
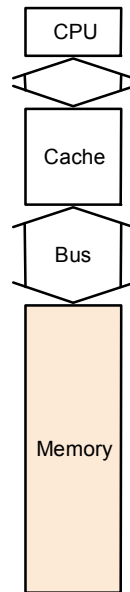
# Hits vs. Misses

---

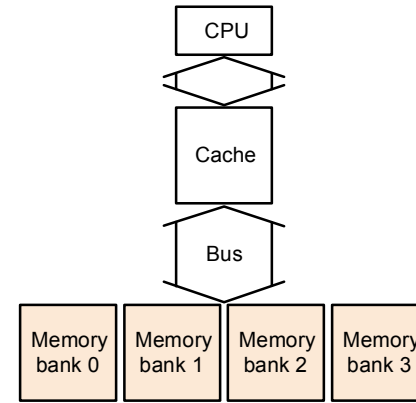
- **Read hits**
  - this is what we want!
- **Read misses**
  - stall the CPU, fetch block from memory, deliver to cache, restart
- **Write hits:**
  - can replace data in cache and memory (write-through)
  - write the data only into the cache (write-back the cache later)
- **Write misses:**
  - read the entire block into the cache, then write the word

# Hardware Issues

- Make reading multiple words easier by using banks of memory



b. Wide memory organization



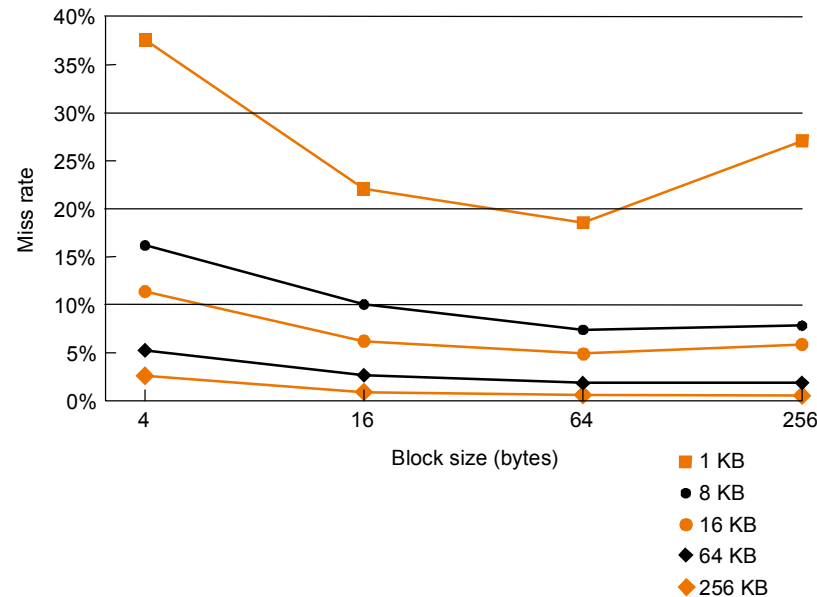
c. Interleaved memory organization

- It can get a lot more complicated...

Interleaved memory organization

# 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%
spice	1	1.2%	1.3%	1.2%
	4	0.3%	0.6%	0.4%

# Performance

---

- Simplified model:

**execution time = (execution cycles + stall cycles)  $\times$  cycle time**

**stall cycles = # of instructions  $\times$  miss ratio  $\times$  miss penalty**

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

*What happens if we increase block size?*

\_\_\_\_\_

Set	Tag	Data	Tag	Data
0				
1				
2				
3				

### Associative Cache

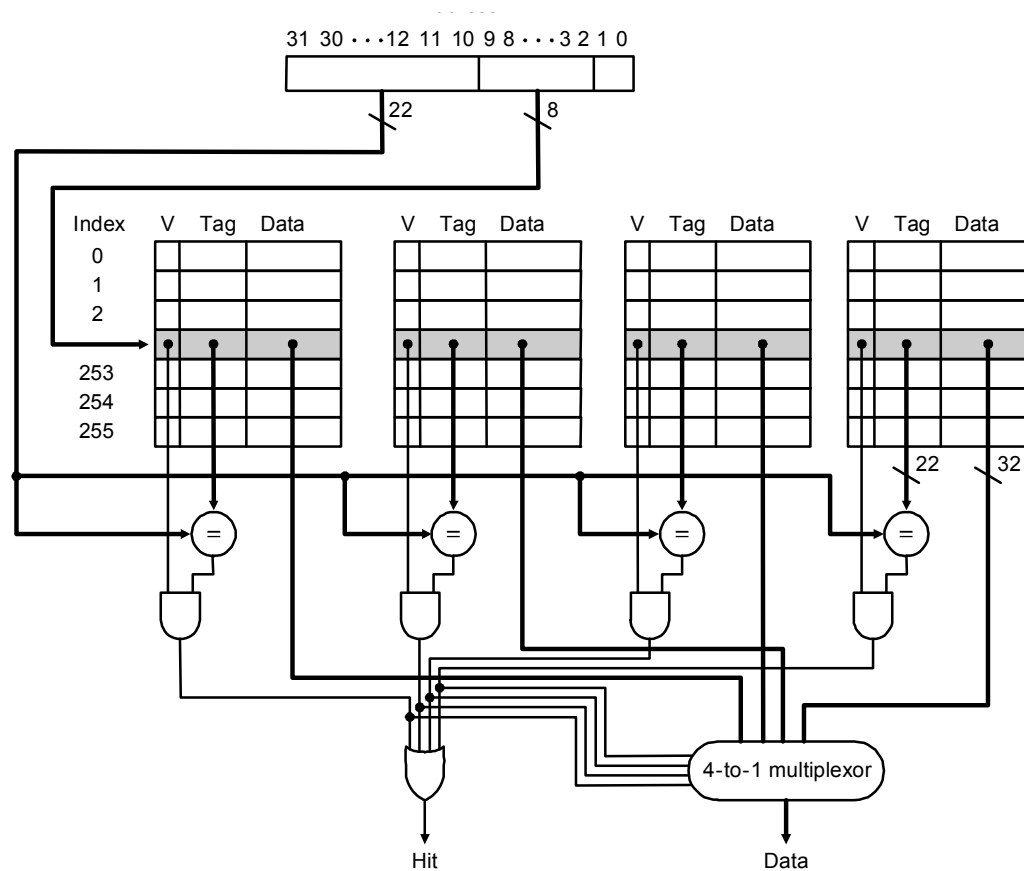
Eight-way set associative (fully associative)

- *results in a lower miss ratio using a 2-way set associative cache.*

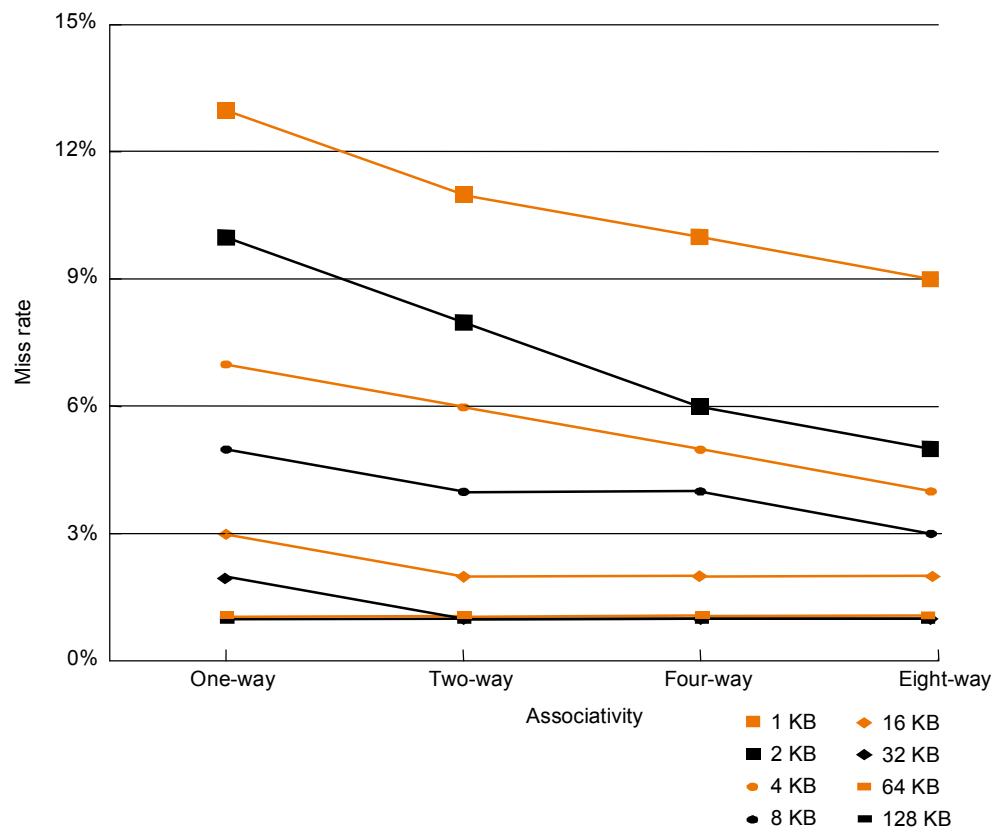
- *results in a higher miss ratio using a 2-way set associative cache*

[illegible]

# An implementation



# Performance



# Decreasing miss penalty with multilevel caches

---

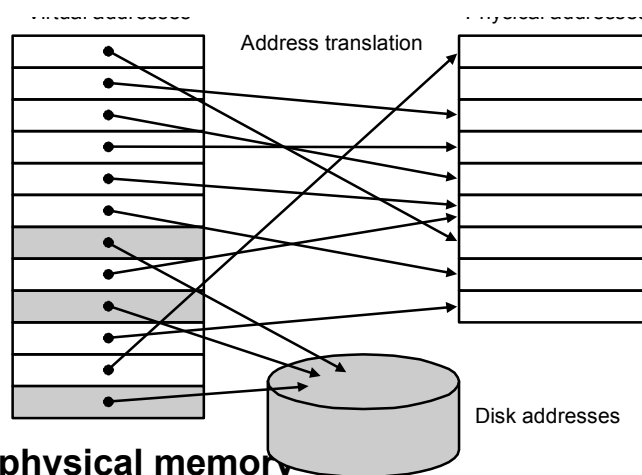
- **Add a second level cache:**
  - often primary cache is on the same chip as the processor
  - use SRAMs to add another cache above primary memory (DRAM)
  - miss penalty goes down if data is in 2nd level cache
- **Example:**
  - CPI of 1.0 on a 500Mhz machine with a 5% miss rate, 200ns DRAM access
  - Adding 2nd level cache with 20ns access time decreases miss rate to 2%
- **Using multilevel caches:**
  - try and optimize the hit time on the 1st level cache
  - try and optimize the miss rate on the 2nd level cache



# Virtual Memory

---

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

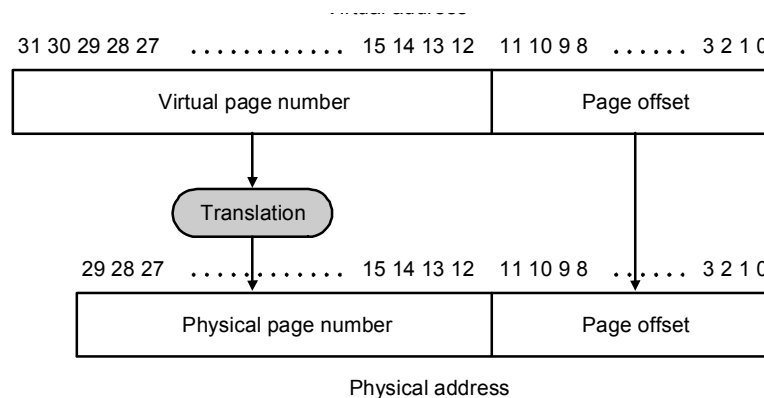


- **Advantages:**
  - illusion of having more physical memory
  - program relocation
  - protection

# Pages: virtual memory blocks

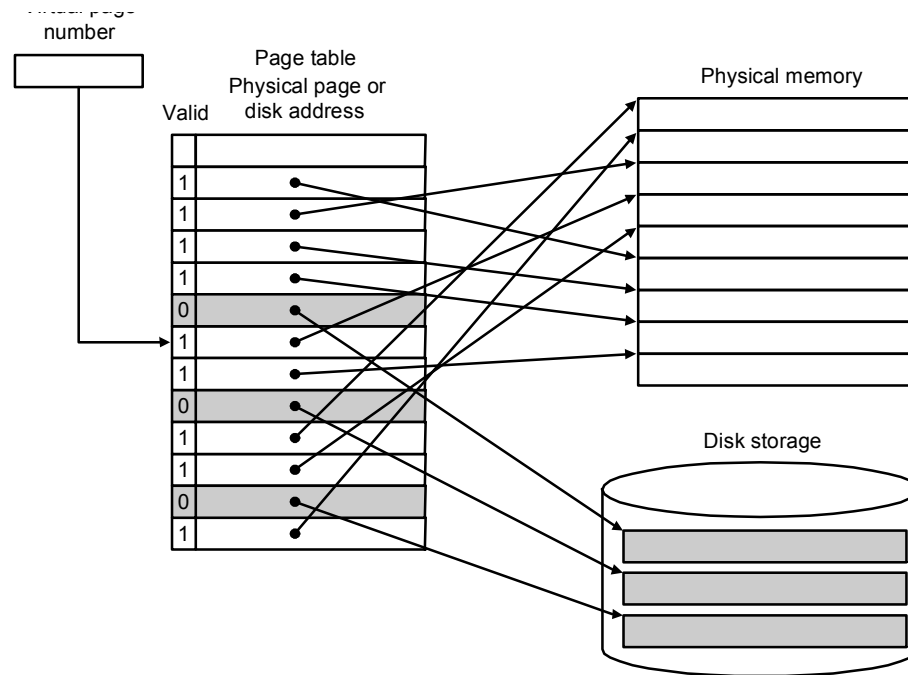
---

- **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

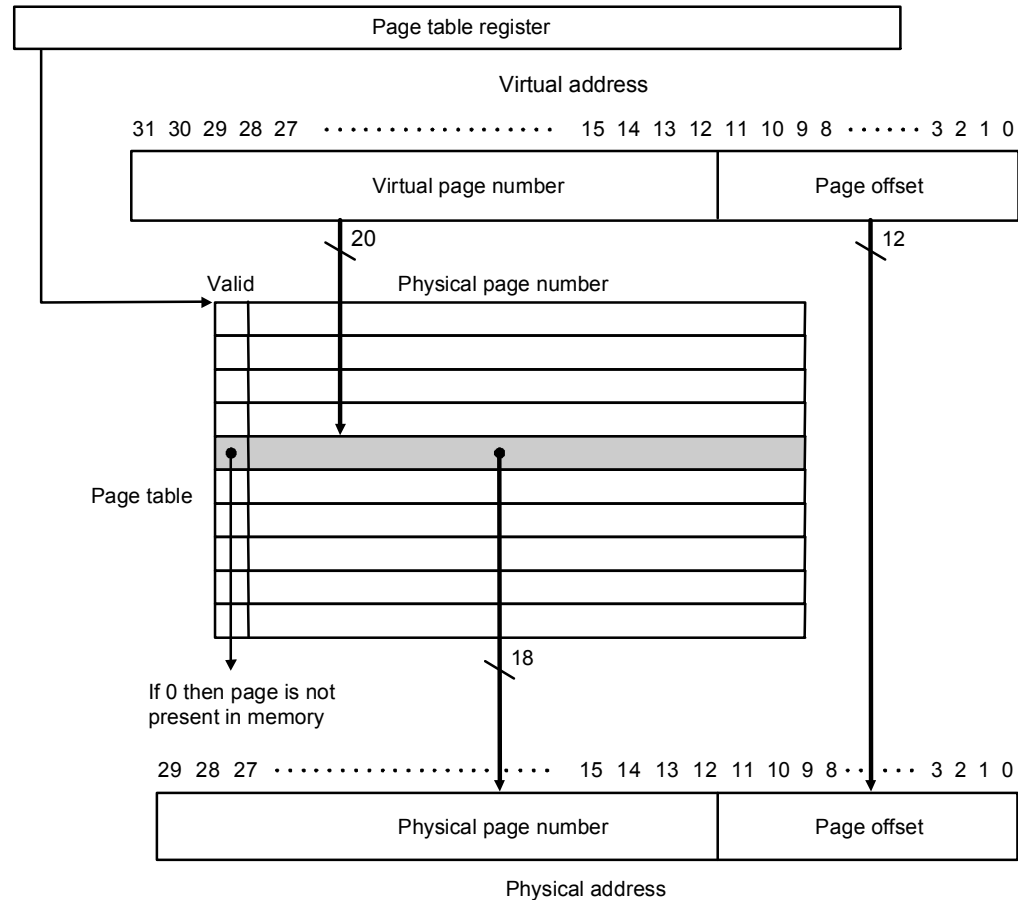


# Page Tables

---

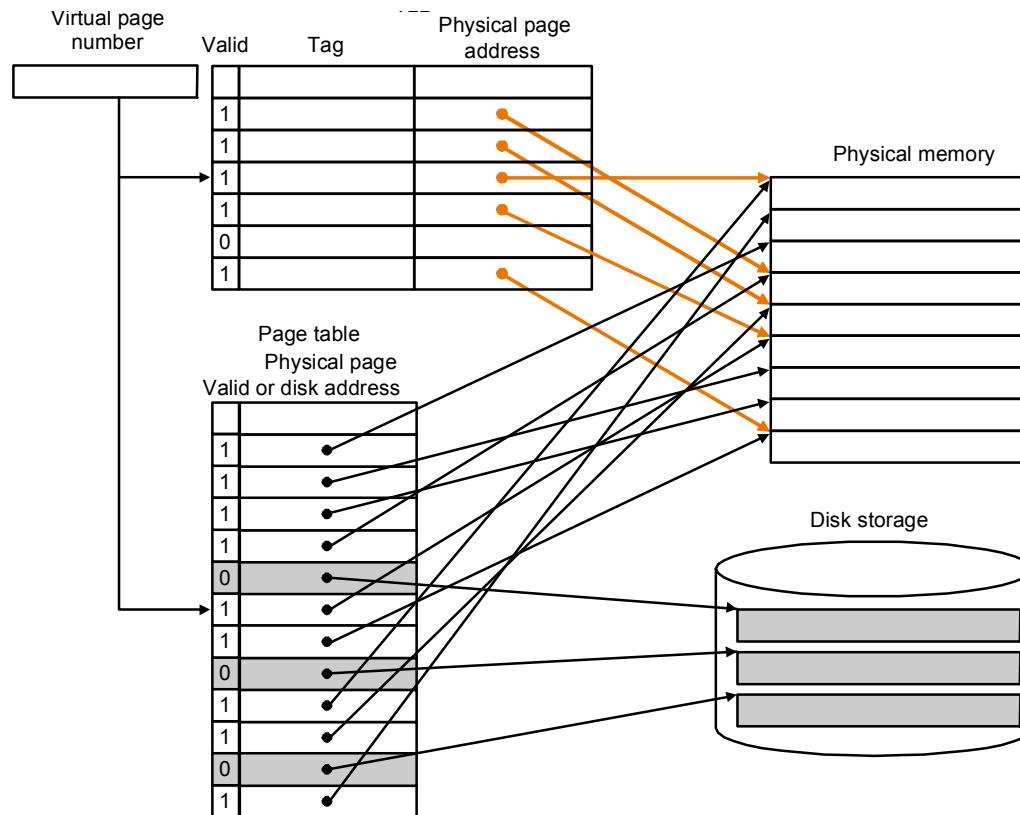


# Page Tables

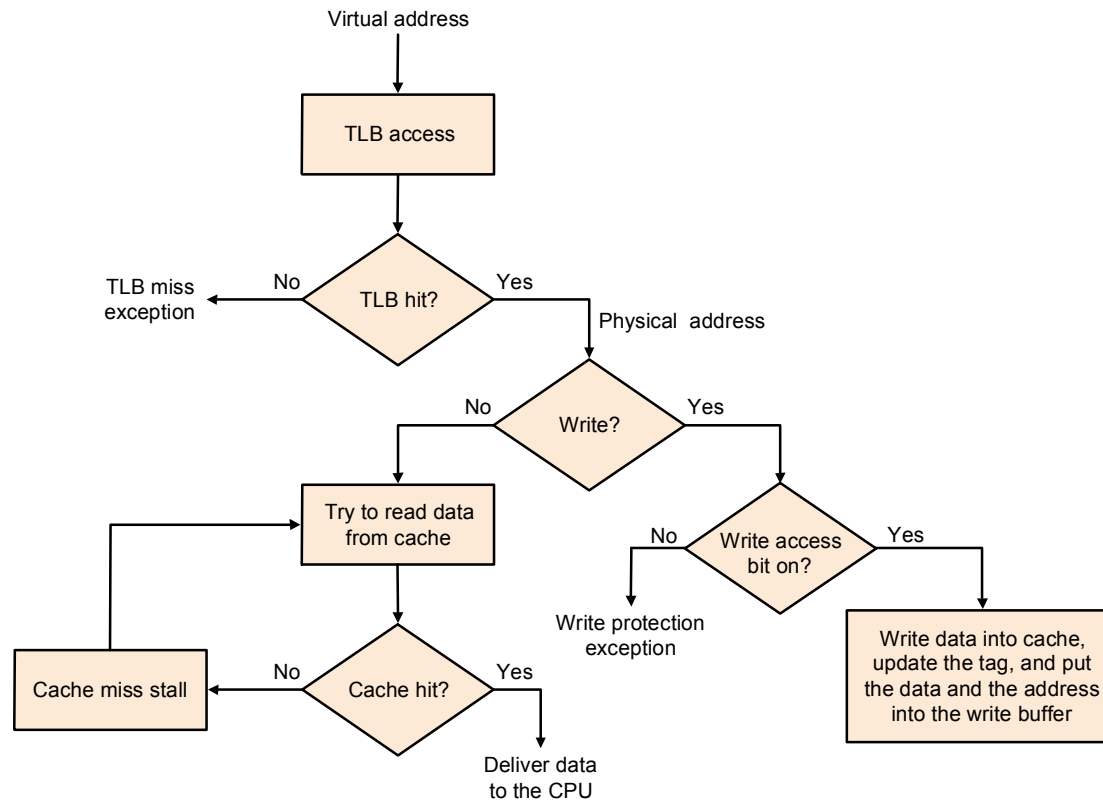


# Making Address Translation Fast

- A cache for address translations: translation lookaside buffer



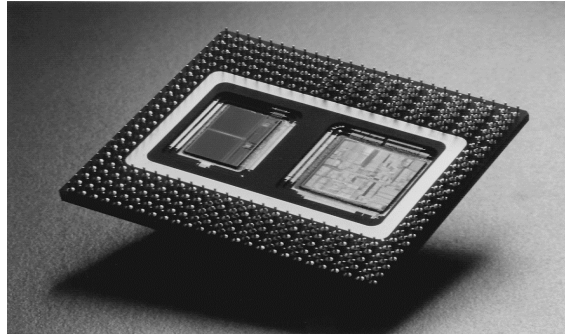
# TLBs and caches



# Modern Systems

- Very complicated memory systems:

Characteristic	Intel Pentium Pro	PowerPC 604
Virtual address	32 bits	52 bits
Physical address	32 bits	32 bits
Page size	4 KB, 4 MB	4 KB, selectable, and 256 MB
TLB organization	A TLB for instructions and a TLB for data Both four-way set associative Pseudo-LRU replacement Instruction TLB: 32 entries Data TLB: 64 entries TLB misses handled in hardware	A TLB for instructions and a TLB for data Both two-way set associative LRU replacement Instruction TLB: 128 entries Data TLB: 128 entries TLB misses handled in hardware



Characteristic	Intel Pentium Pro	PowerPC 604
Cache organization	Split instruction and data caches	Split instruction and data caches
Cache size	8 KB each for instructions/data	16 KB each for instructions/data
Cache associativity	Four-way set associative	Four-way set associative
Replacement	Approximated LRU replacement	LRU replacement
Block size	32 bytes	32 bytes
Write policy	Write-back	Write-back or write-through

# Some Issues

---

- **Processor speeds continue to increase very fast**
  - much faster than either DRAM or disk access times
- **Design challenge: dealing with this growing disparity**
- **Trends:**
  - synchronous SRAMs (provide a burst of data)
  - redesign DRAM chips to provide higher bandwidth or processing
  - restructure code to increase locality
  - use prefetching (make cache visible to ISA)