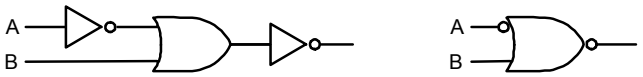
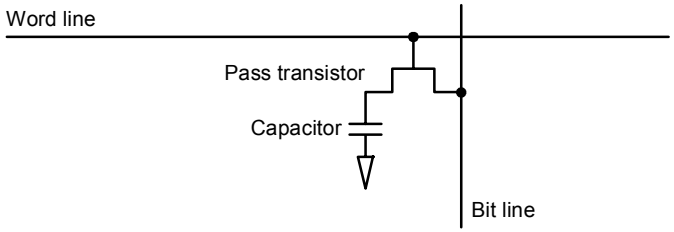
Computer memories

- **SRAM:**
  - **value is stored on a pair of inverting gates**
  - **very fast but takes up more space than DRAM (4 to 6 transistors)**

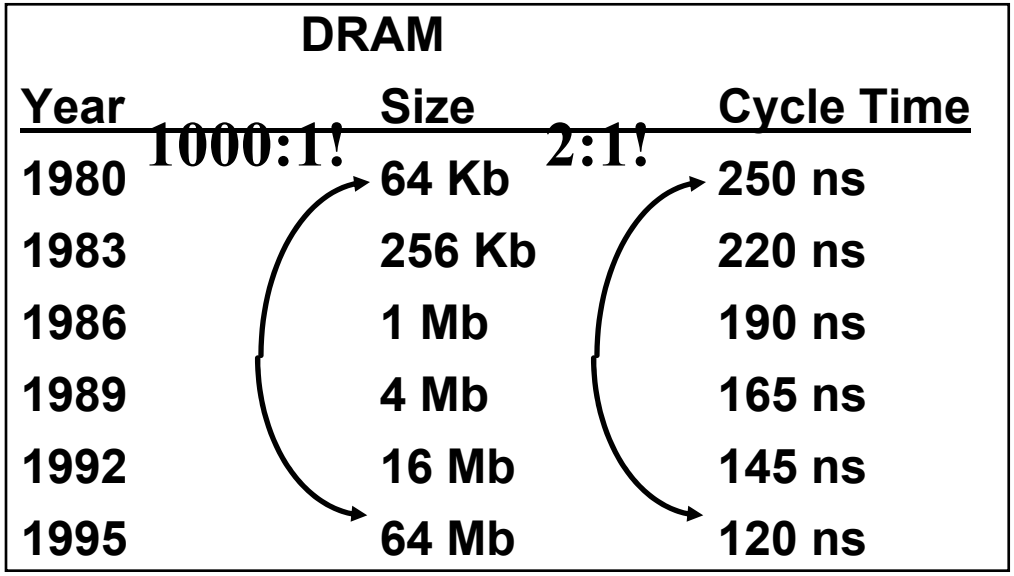
- **DRAM:**
  - **value is stored as a charge on capacitor (must be refreshed)**
  - **very small but slower than SRAM (factor of 5 to 10)**



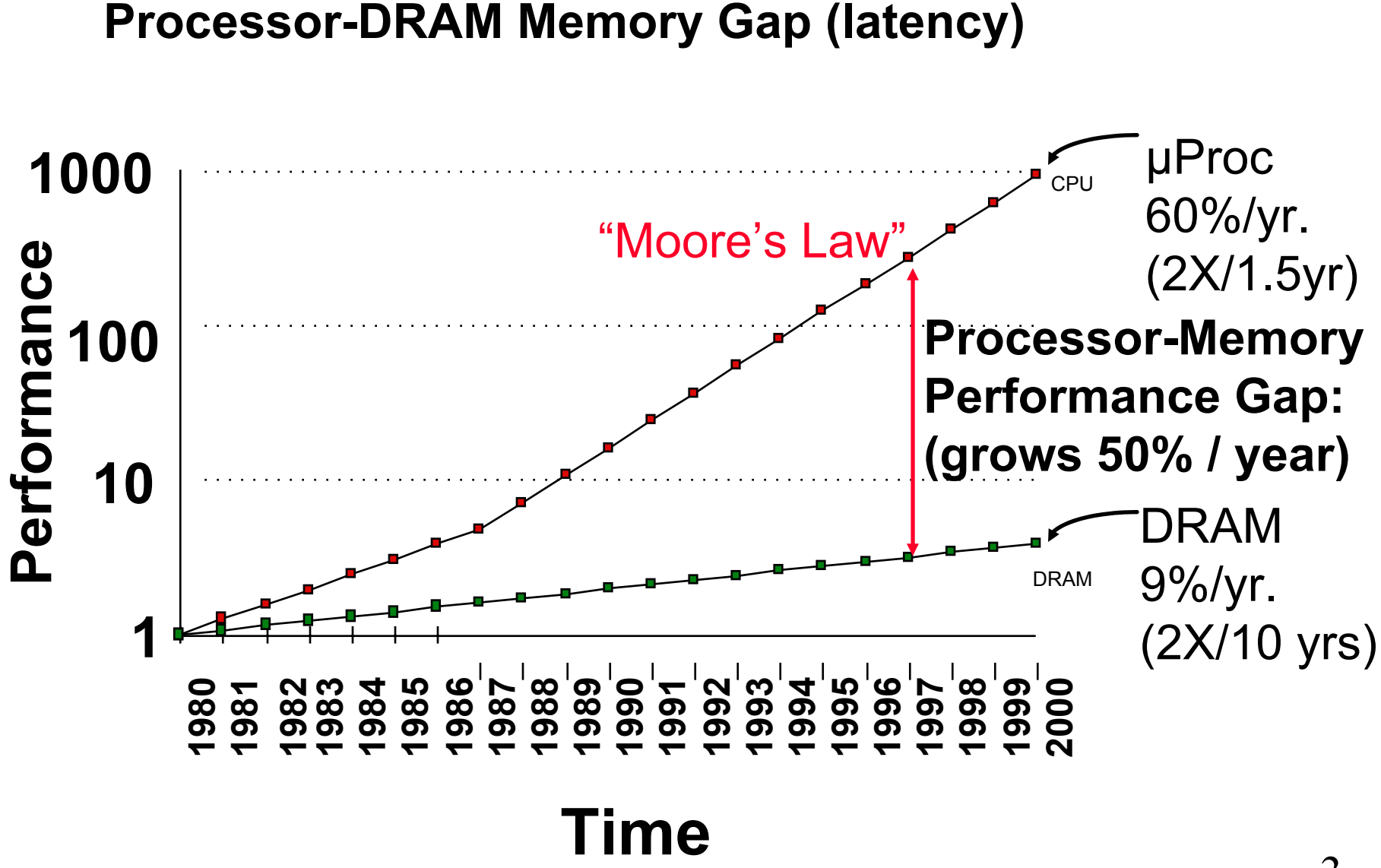


Technology Trends

	Capacity	Speed (latency)
SRAM:	2x in 3 years	2x in 3 years
DRAM:	4x in 3 years	2x in 10 years
Disk:	4x in 3 years	2x in 10 years



Who Cares About the Memory Hierarchy?

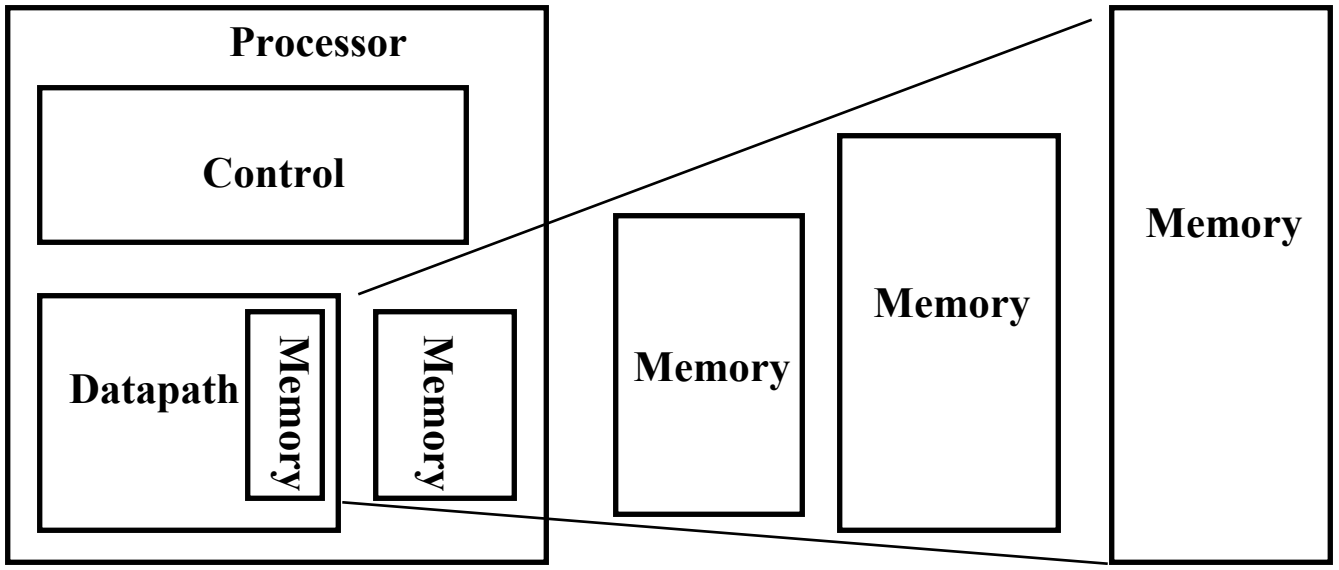


The Goal: illusion of large, fast, cheap memory

- **Fact: Large memories are slow, fast memories are small**
- **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.

- **How do we create a memory that is large, cheap and fast (most of the time)?**
  - **Hierarchy**
  - **Parallelism**

An Expanded View of the Memory System



Speed: Fastest
Size: Smallest
Cost: Highest

Slowest
Biggest
Lowest

Why hierarchy works

- **The Principle of Locality:**
  - **Program access a relatively small portion of the address space at any instant of time.**

Locality

- **A principle that makes having a memory hierarchy a good idea**

- **If an item is referenced,**

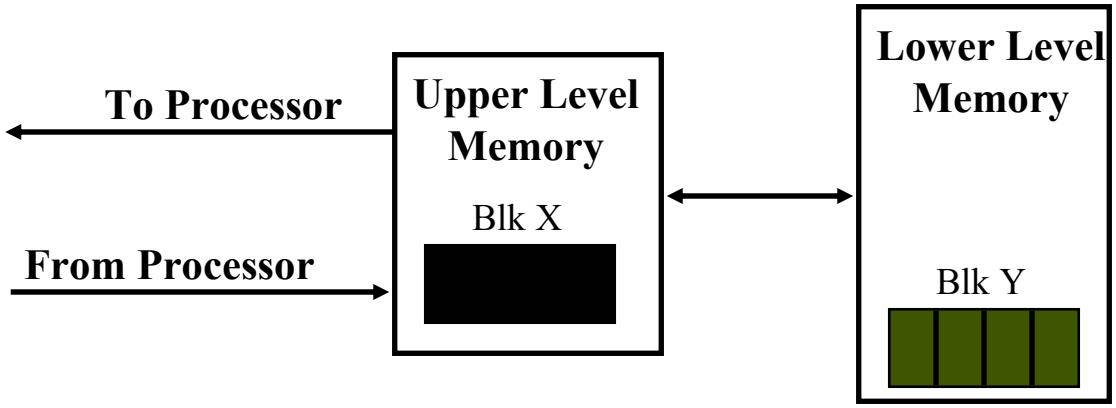
  temporal locality: it will tend to be referenced again soon
  spatial locality: nearby items will tend to be referenced soon.

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: How Does it Work?

- **Temporal Locality (Locality in Time):**
  => Keep most recently accessed data items closer to the processor
- **Spatial Locality (Locality in Space):**
  => Move blocks consists of contiguous words to the upper levels

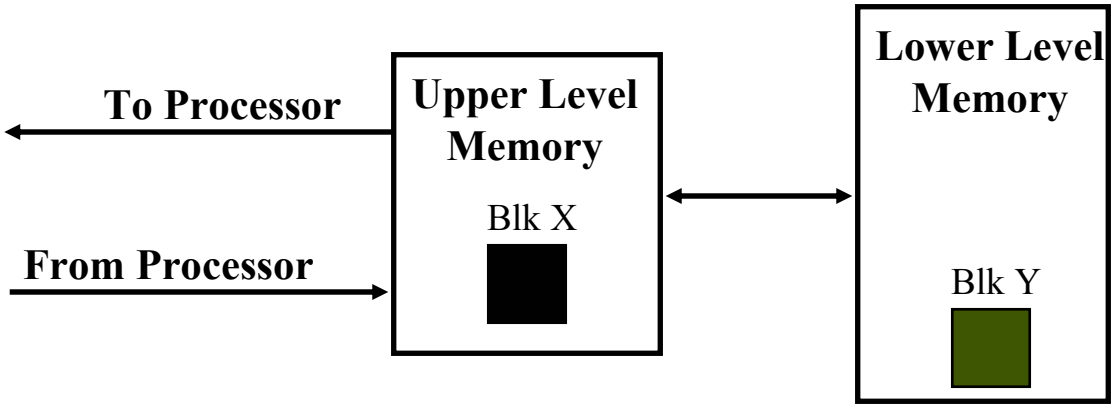


Memory Hierarchy: Terminology

- **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**

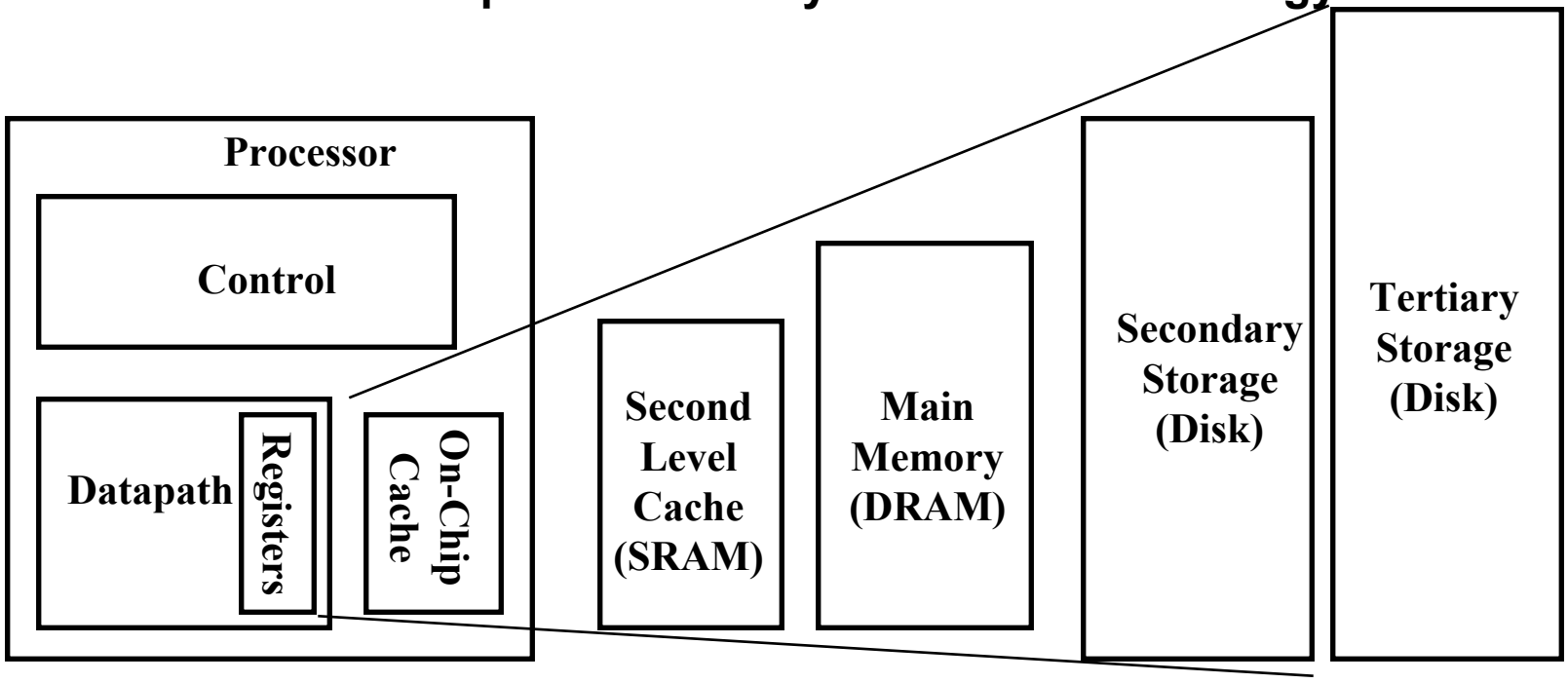
    RAM access time + Time to determine hit/miss
- **Miss: data needs to be retrieve from a block in the lower level (Block Y)**
  - **Miss Rate = 1 - (Hit Rate)**
  - **Miss Penalty: Time to replace a block in the upper level +**

    Time to deliver the block the processor
- **Hit Time << Miss Penalty**



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

How is the hierarchy managed?

- Registers <-> Memory
  - by compiler (programmer?)
- cache <-> memory
  - by the hardware
- memory <-> disks
  - by the hardware and operating system (virtual memory)
  - by the programmer (files)

Cache

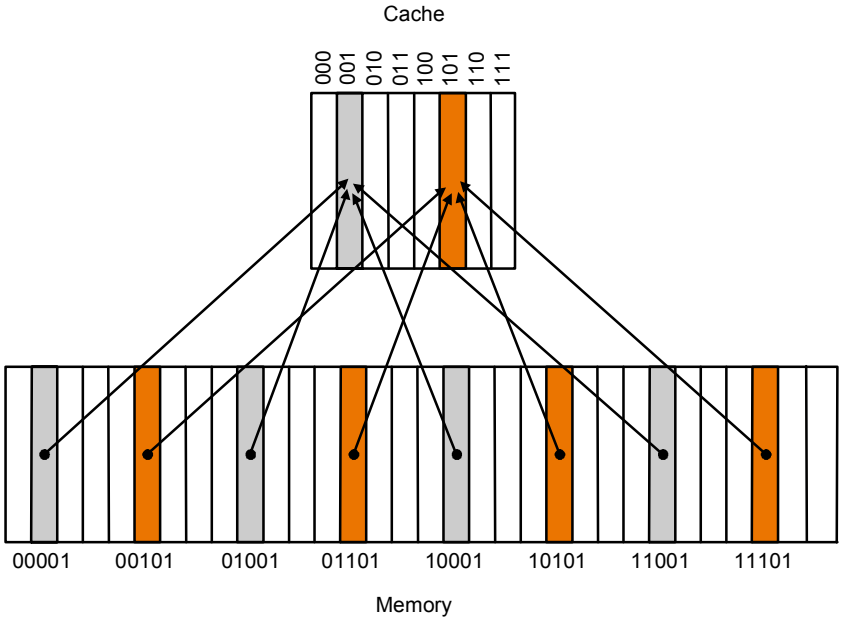
- Two issues:
  - How do we know if a data item is in the cache?
  - If it is, how do we find it?
- Our first example:
  - block size is one word of data
  - "direct mapped"

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

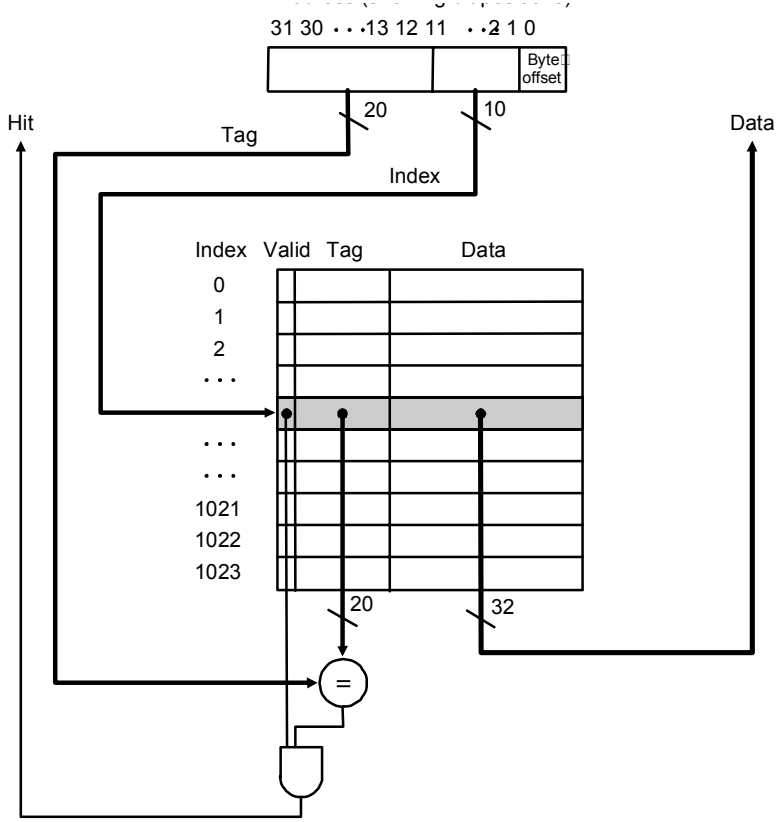
Direct Mapped Cache

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



Direct Mapped Cache

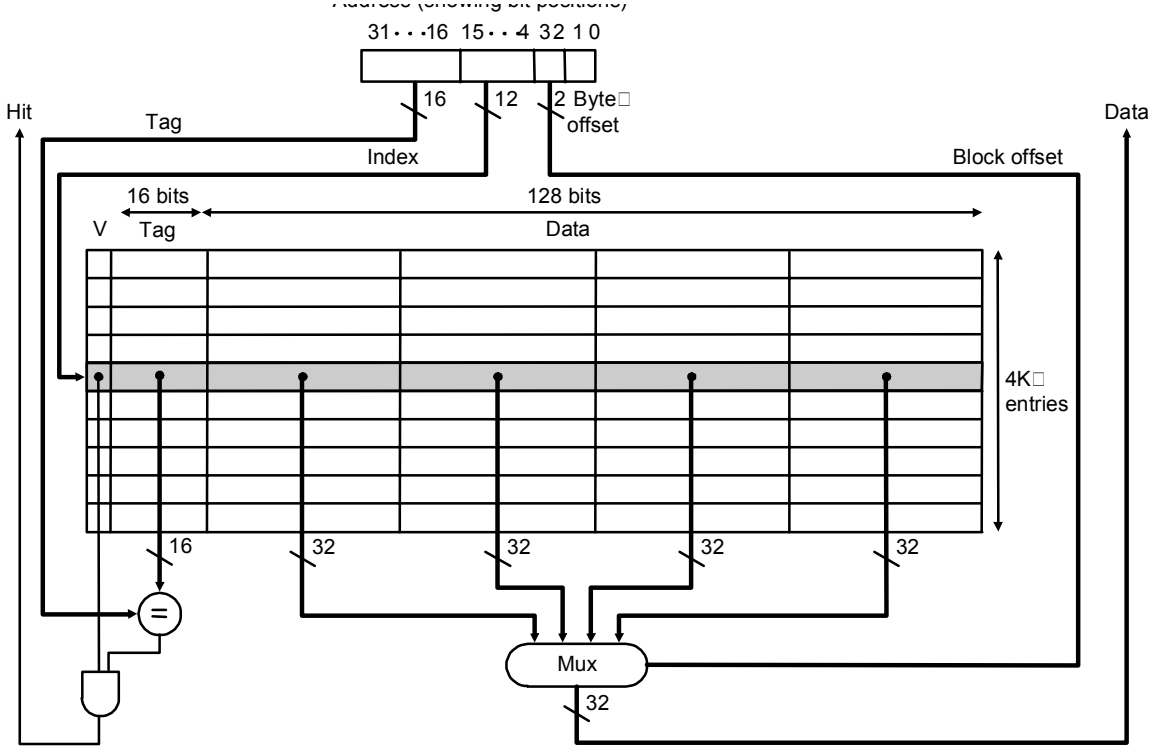
- **For MIPS:**



What kind of locality are we taking advantage of?

Direct Mapped Cache

- **Taking advantage of spatial locality:**

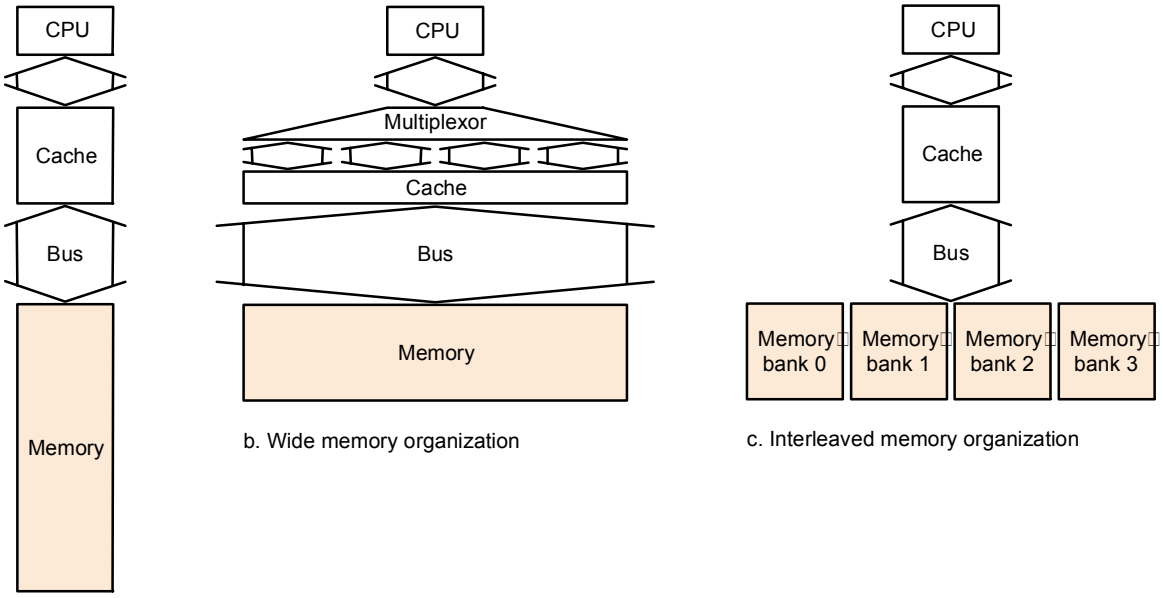


Hits vs. Misses

- **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**



a. One-word-wide memory organization

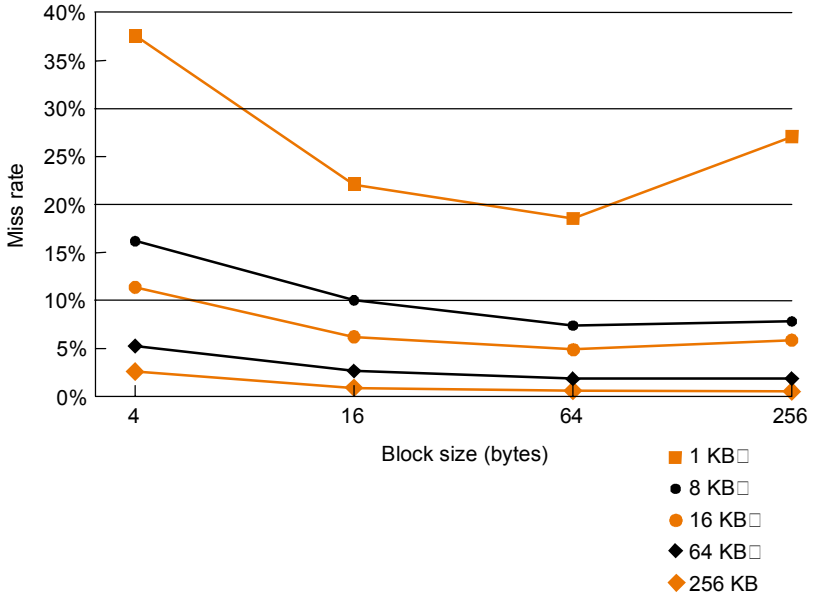
b. Wide memory organization

c. Interleaved memory organization

- **It can get a lot more complicated...**

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:**

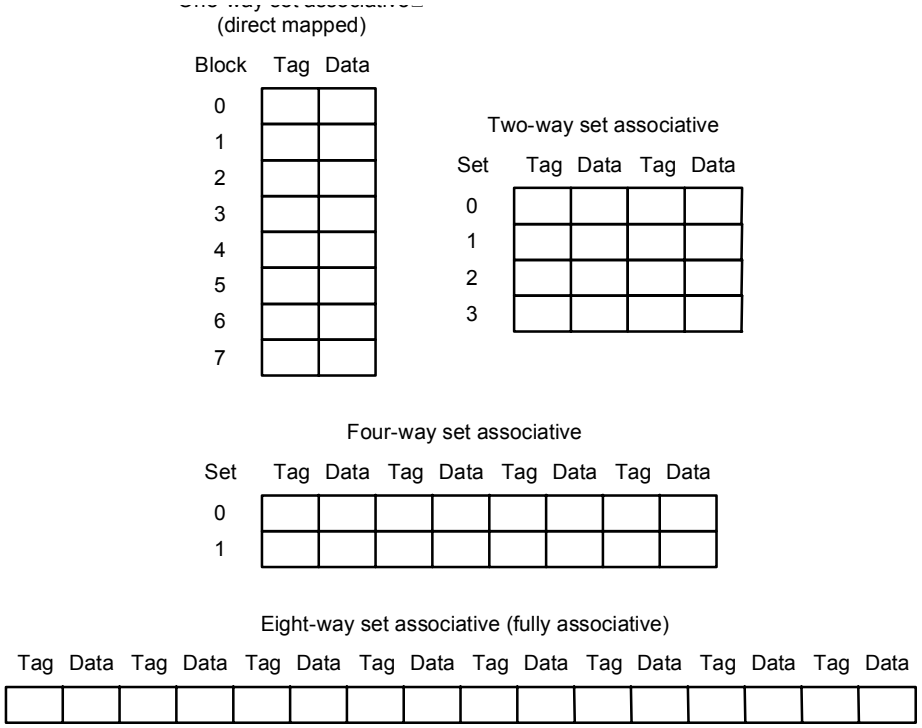
  execution time = (execution cycles + stall cycles) × cycle time

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

- **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

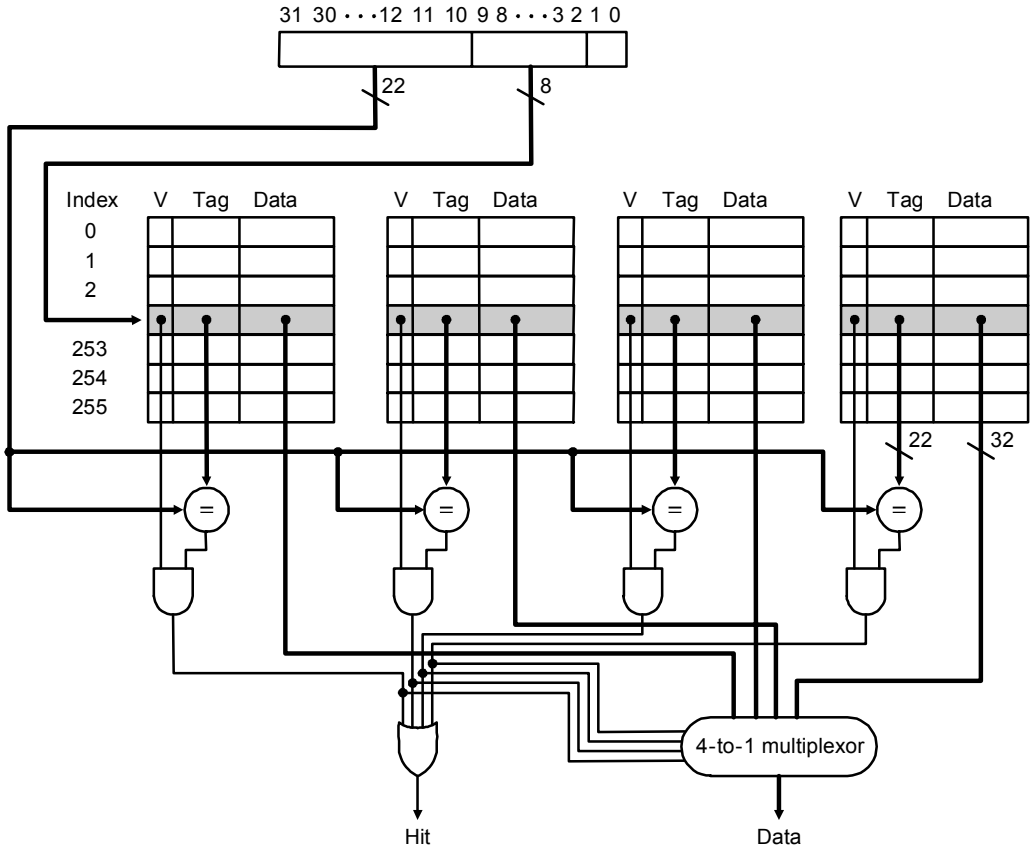


Compared to direct mapped, give a series of references that:

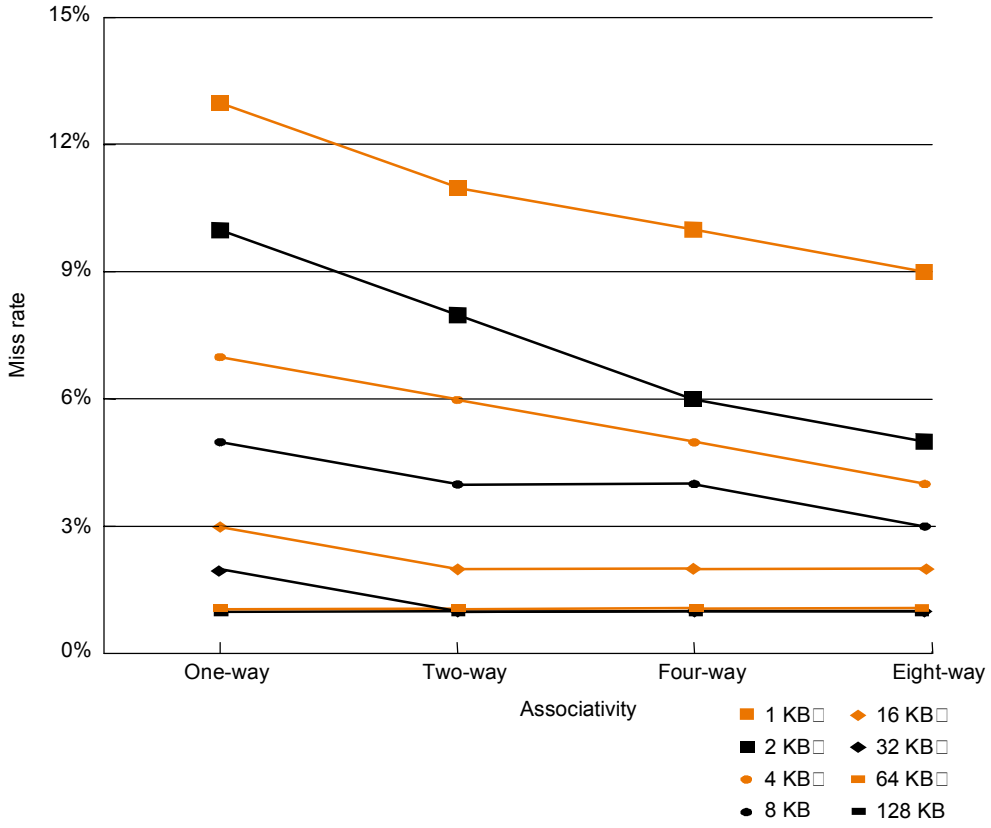
- ***results in a lower miss ratio using a 2-way set associative cache***
- ***results in a higher miss ratio using a 2-way set associative cache***

assuming we use the "least recently used" replacement strategy

An implementation

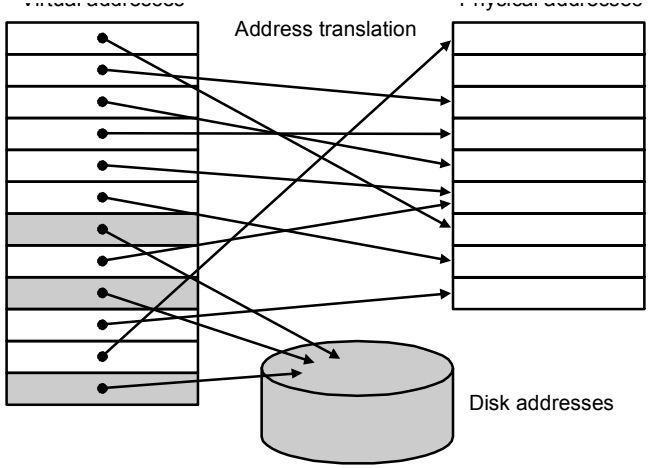


Performance



Virtual Memory

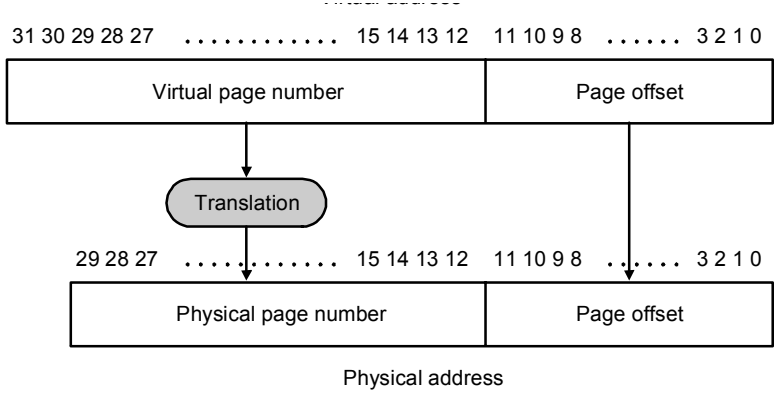
- **Main memory can act as a cache for the secondary storage (disk)**



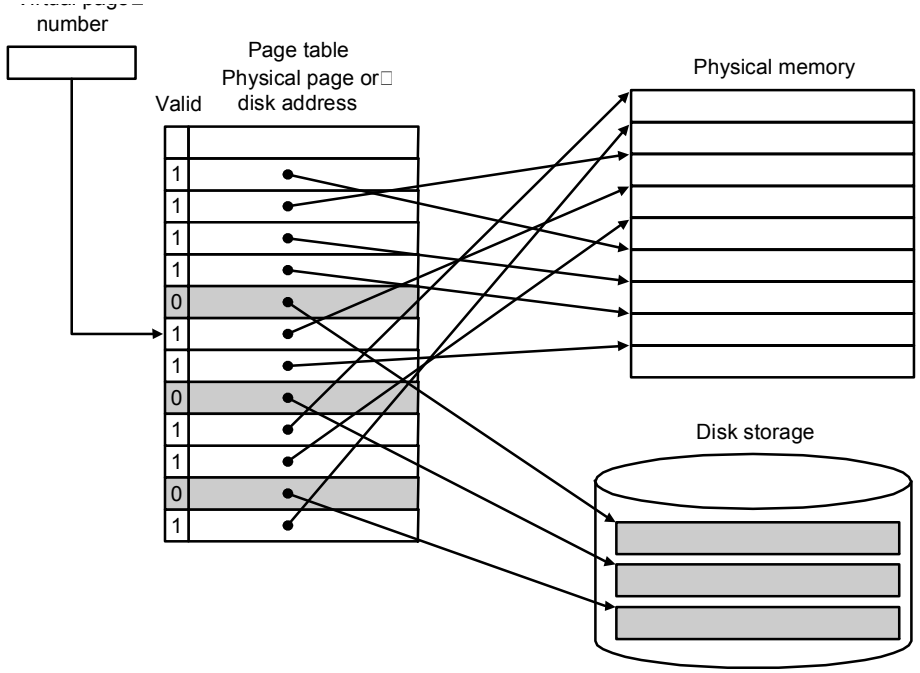
- **Advantages:**
  - **illusion of having more physical memory**
  - **program relocation**
  - **protection**

Pages: virtual memory blocks

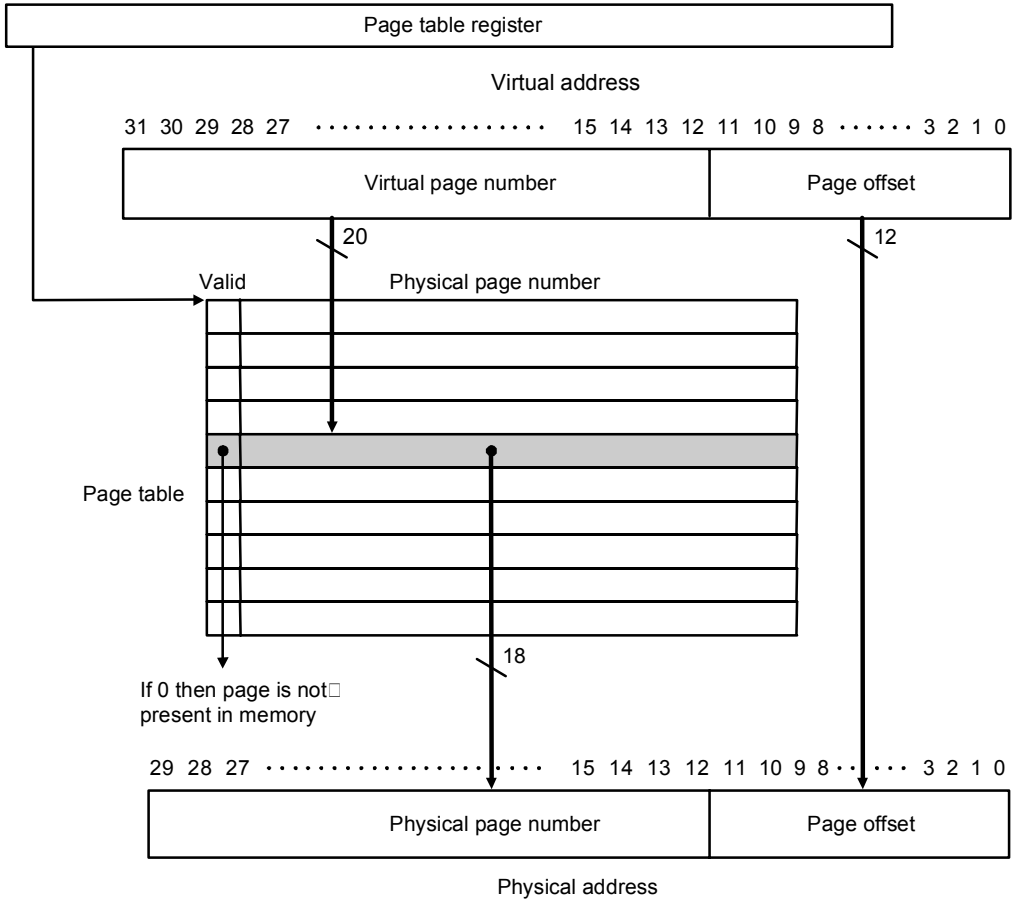
- **Page faults: the data is not in memory, retrieve it from disk**
  - **huge miss penalty, thus pages should be fairly large (e.g., 4KB)**
  - **reducing page faults is important (LRU is worth the price)**
  - **can handle the faults in software instead of hardware**
  - **using write-through is too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



Page Tables

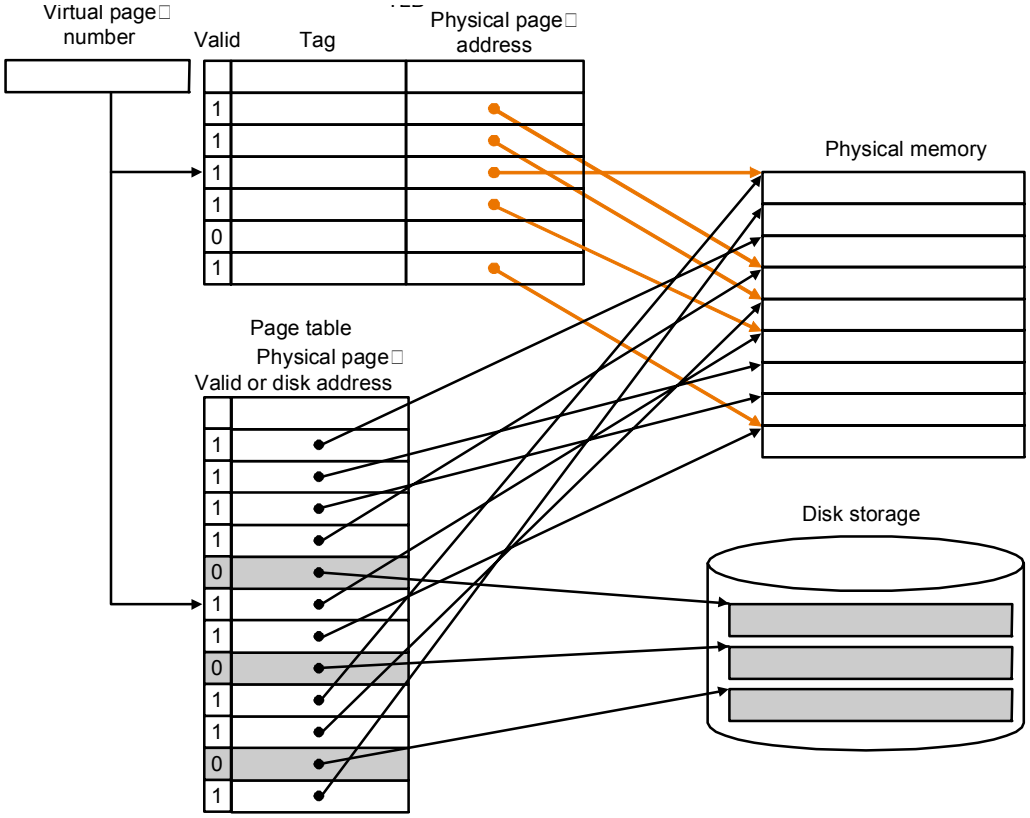


Page Tables

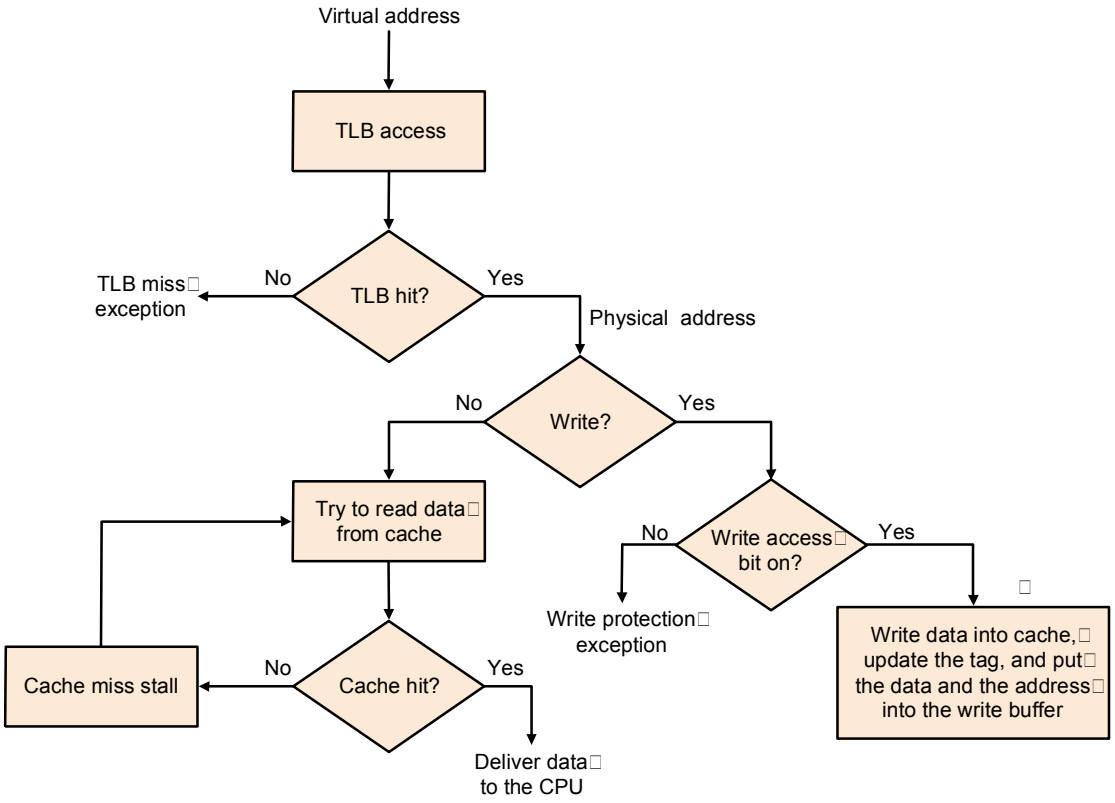


Making Address Translation Fast

- **A cache for address translations: translation lookaside buffer (TLB)**



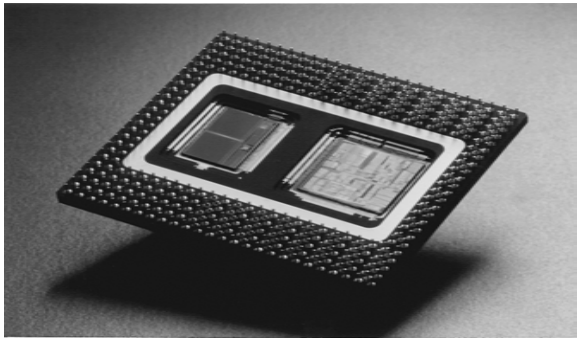
TLBs and caches



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 intruction 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