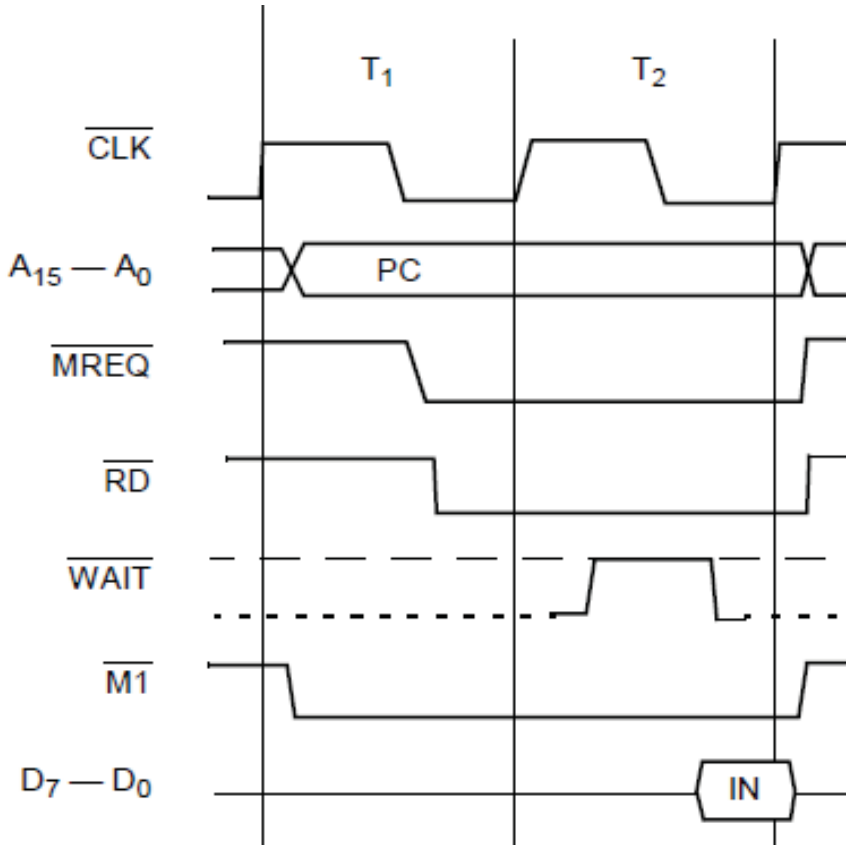
Memory Hierarchy

The Old Days

- Processor slower than (D)RAM
- Example from the 80s
  - My first computer
  - Zilog Z80 processor
    - 4 MHz – 375 ns
  - 1 KB SRAM
    - 2x F2114 (1024x4 bit)
    - Read cycle time 200 ns



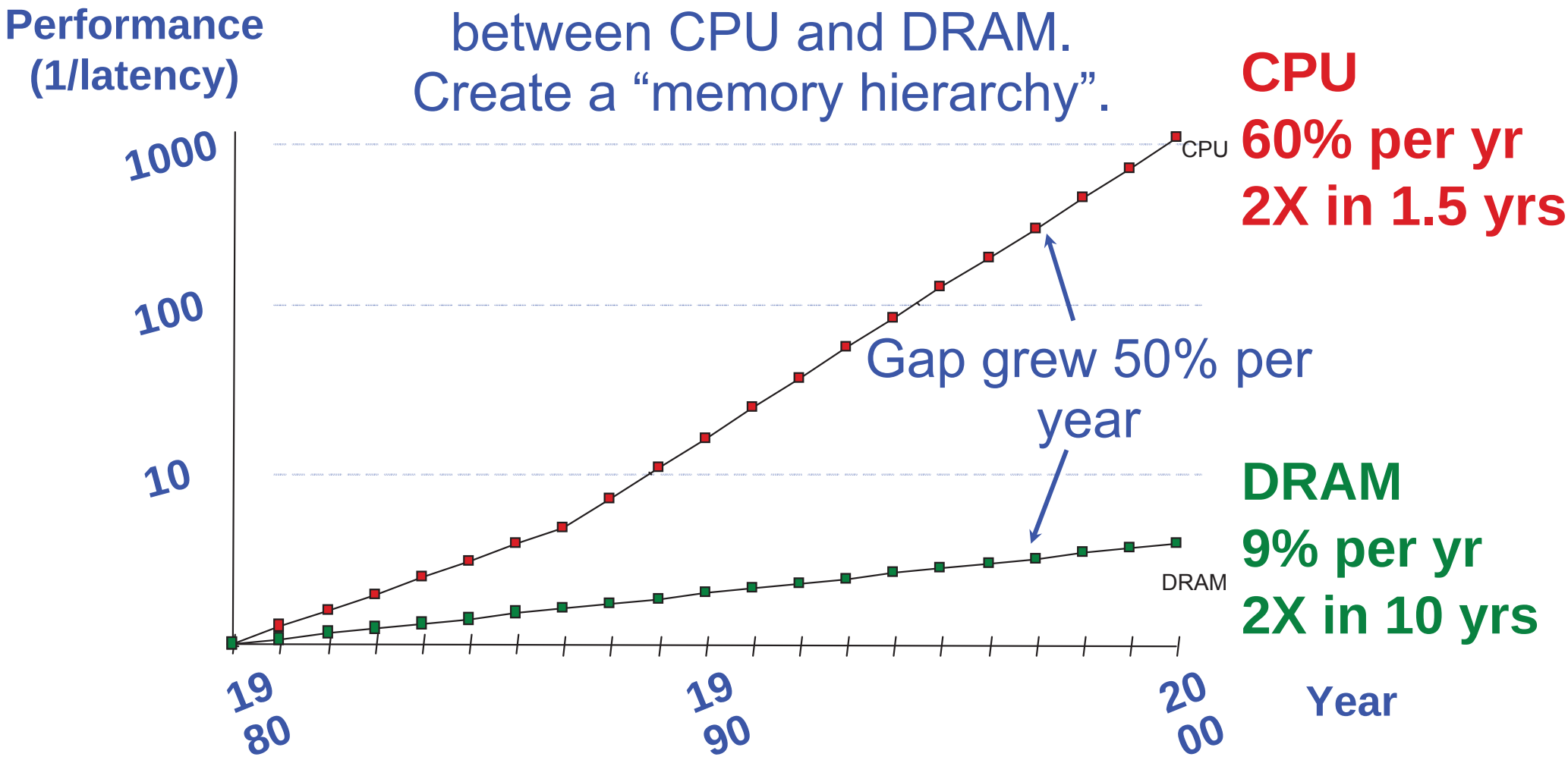
Now

- Standard processor
  - f_{cpu} x GHz, several instructions per clock
  - SDRAM x00 MHz, several cycles latency
  - CPU outperforms memory by >100
- Embedded system (FPGA example)
  - CPU 100 MHz (slow)
  - SRAM 15 ns (fast)
    - 2 cycle memory access
  - 2 words per clock needed (instruction + data)
  - CPU outperforms memory by 4

Since 1980, CPU has outpaced DRAM

Q. How do architects address this gap?

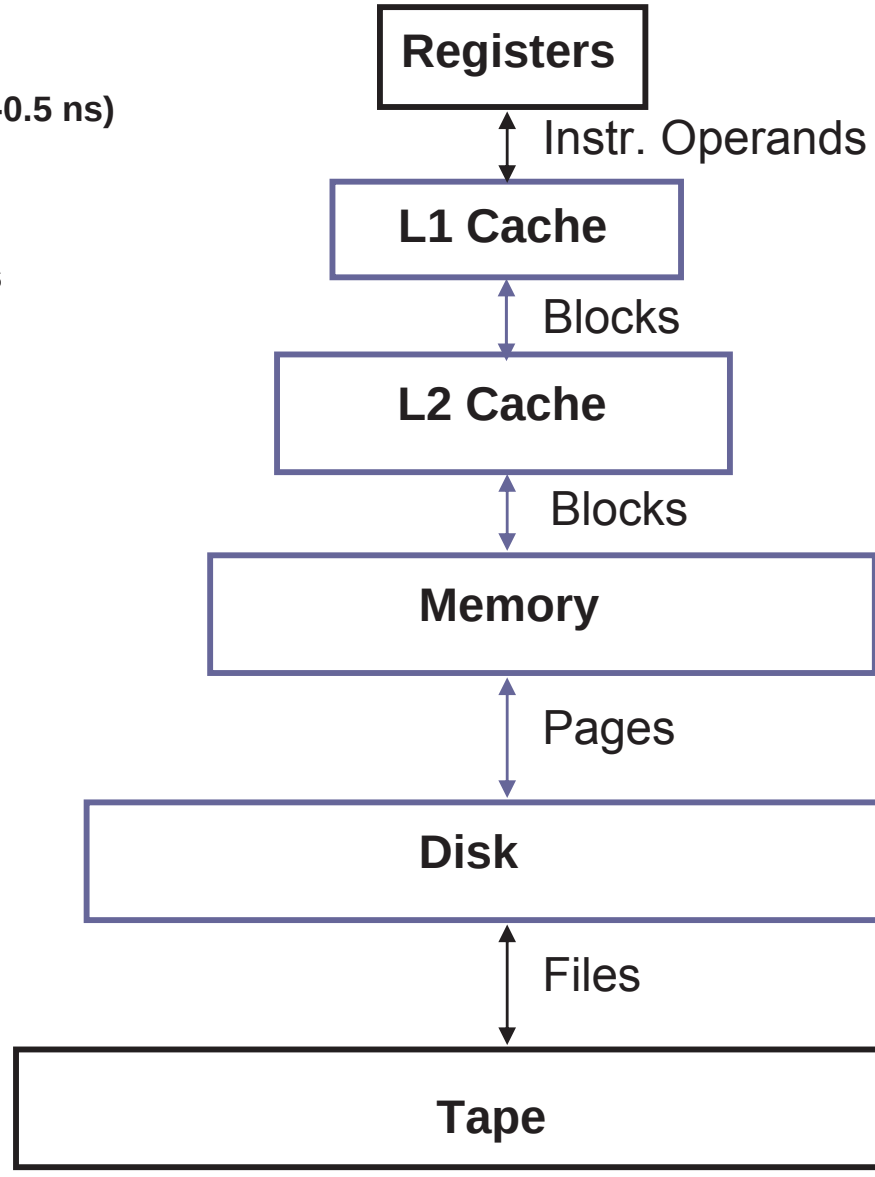
A. Put smaller, faster “cache” memories between CPU and DRAM. Create a “memory hierarchy”.



Levels of Memory Hierarchy

Capacity
Access Time
Cost

Staging
Transfer Unit



CPU Registers
100s Bytes
300 – 500 ps (0.3-0.5 ns)

prog./compiler
1-8 bytes

Upper Level

L1 and L2 Cache
10s-100s K Bytes
~1 ns - ~10 ns
$1000s/ GByte

cache cntl
32-64 bytes

cache cntl
64-128 bytes

Main Memory
G Bytes
80ns- 200ns
~ $100/ GByte

OS
4K-8K bytes

Disk
10s T Bytes, 10 ms
(10,000,000 ns)
~ $1 / GByte

user/operator
Mbytes

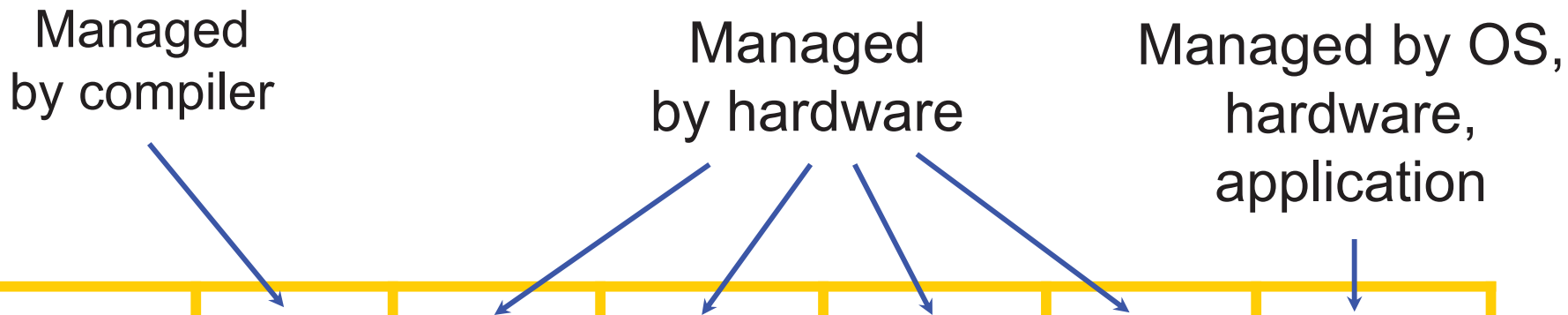
Tape
infinite
sec-min
~$1 / GByte

faster

Larger

Lower Level

Memory Hierarchy: Apple iMac G5

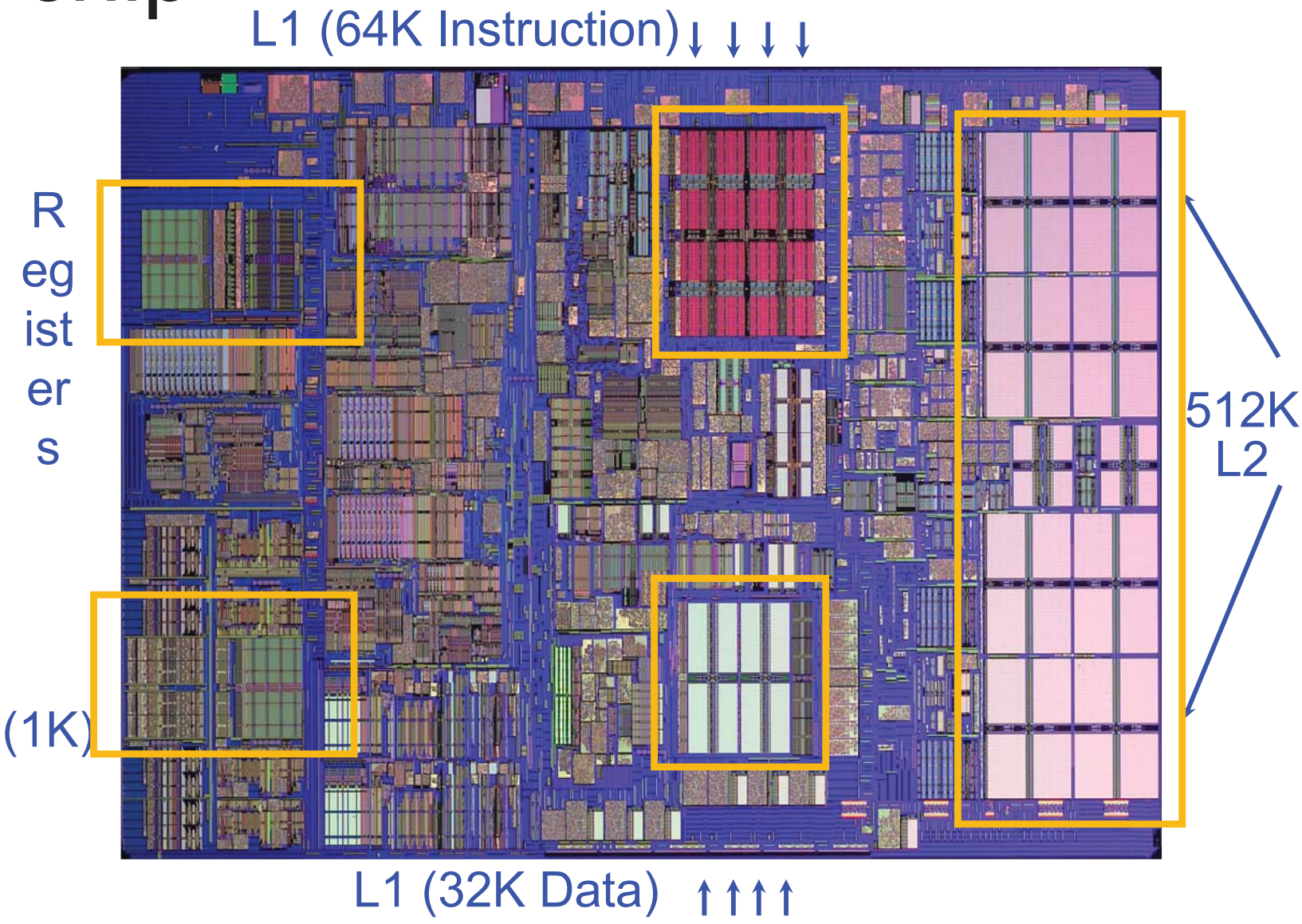


07	Reg	L1 Inst	L1 Data	L2	DRAM	Disk
Size	1K	64K	32K	512K	256M	80G
Latency Cycles, Time	1, 0.6 ns	3, 1.9 ns	3, 1.9 ns	11, 6.9 ns	88, 55 ns	10^7, 12 ms

Goal: Illusion of large, fast, cheap memory

Let programs address a memory space that scales to the disk size, at a speed that is usually as fast as register access

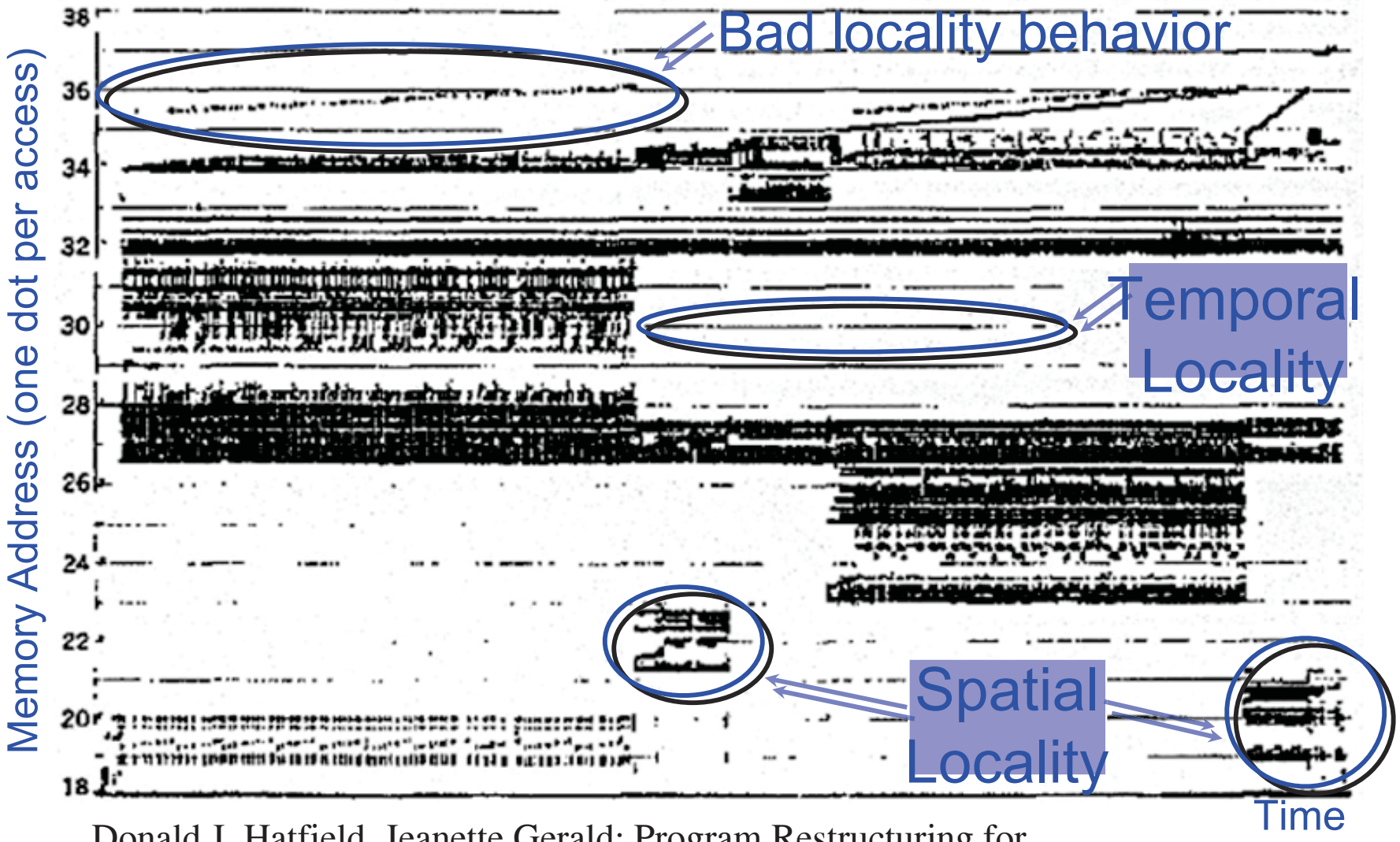
iMac's PowerPC 970: All caches on-chip



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., straight-line code, array access)
- Memory Hierarchy - Caches

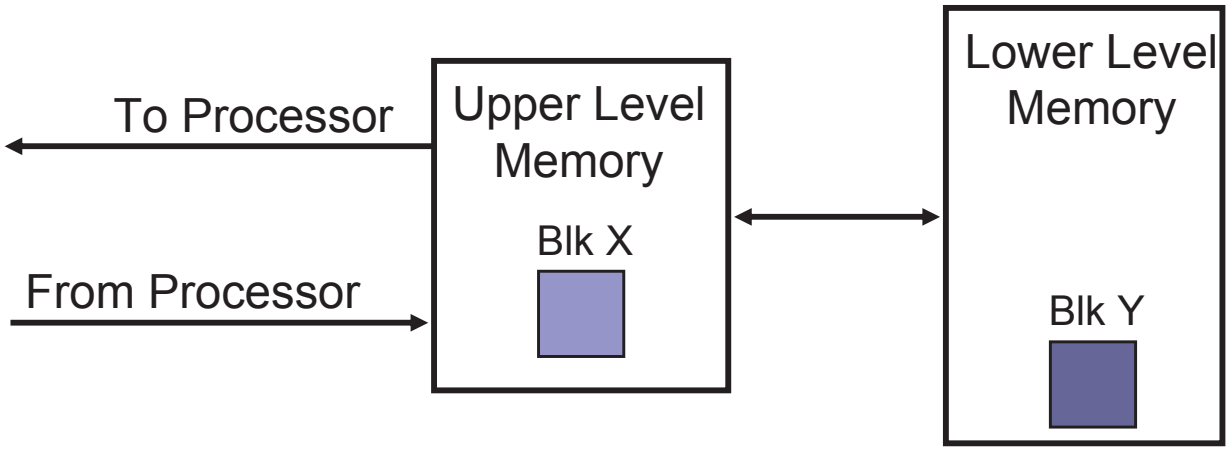
Programs with locality cache well



Donald J. Hatfield, Jeanette Gerald: Program Restructuring for Virtual Memory. IBM Systems Journal 10(3): 168-192 (1971)

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!)



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 x 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)

Performance Equation

$$
\begin{aligned}
CPU\ time &= (CPU_{clock\ cycles} + Memory_{stall\ cycles}) \times t_{clk} \\
&= (IC \times CPI + Number\ of\ misses \times Miss\ penalty) \times t_{clk} \\
&= IC \times (CPI + \frac{Misses}{Instruction} \times Miss\ penalty) \times t_{clk} \\
&= IC \times (CPI + \frac{Memory\ access}{Instruction} \times Miss\ rate \times Miss\ penalty) \times t_{clk}
\end{aligned}
$$

Example

- CPU CPI is 1.0
- Load, store 30%
- Miss penalty 50 cycles
- Miss rate 2%
- Effective CPI?

Example

$$CPI = CPI_{CPU} + CPI_{Cache}$$

$$= 1 + \frac{Memory\ access}{Instruction} \times Miss\ rate \times Miss\ penalty$$

$$= 1 + (1 + 0.3) \times 0.02 \times 50$$

$$= 1 + 1.3 \times 1$$

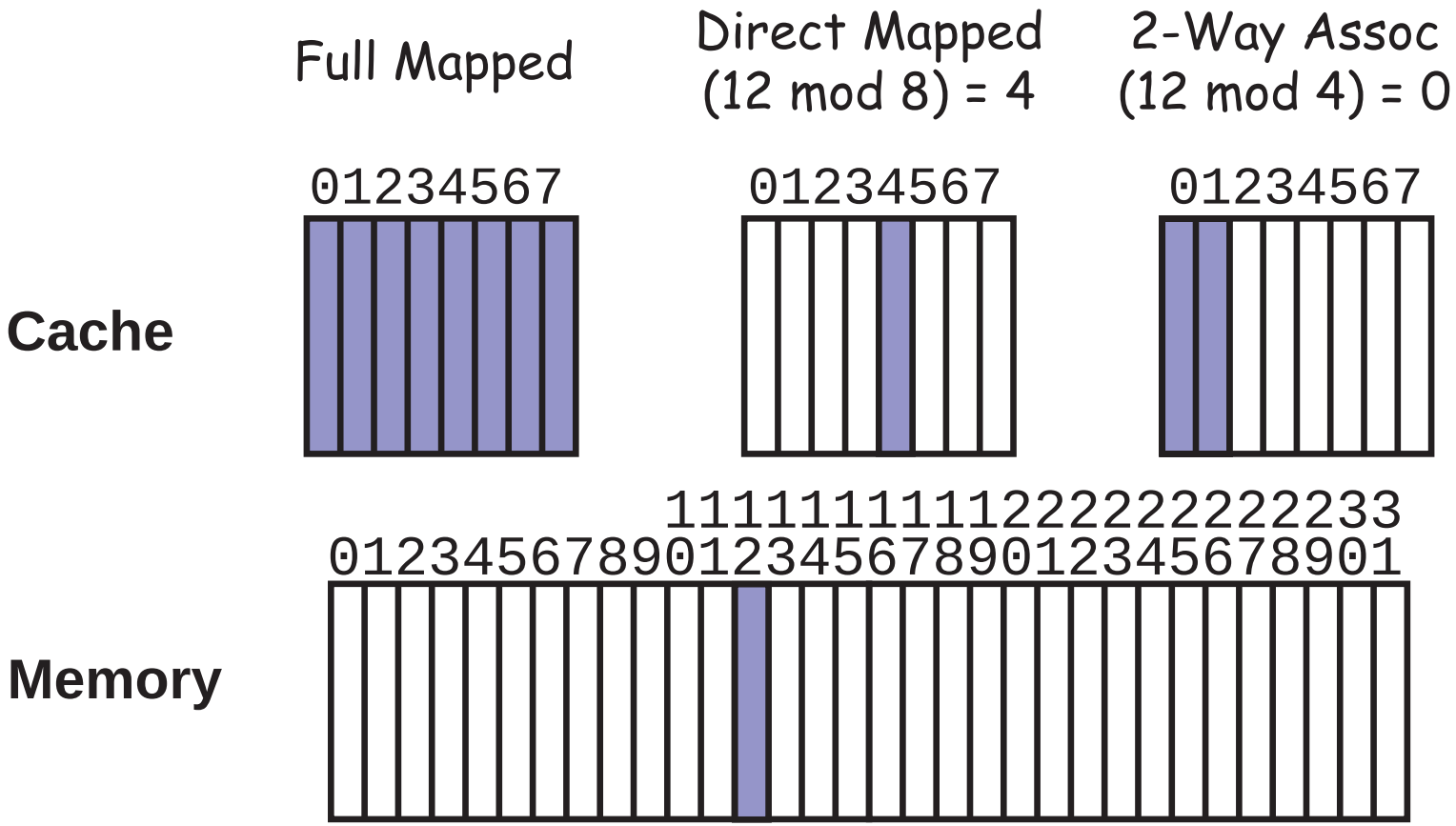
$$= 2.3$$

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

Tag Memory

- Tag check and cache read in parallel
  - On a miss discard read value
- Tag check sequential on a write
- Valid bit

Q3: Which Block Should be Replaced on a Miss?

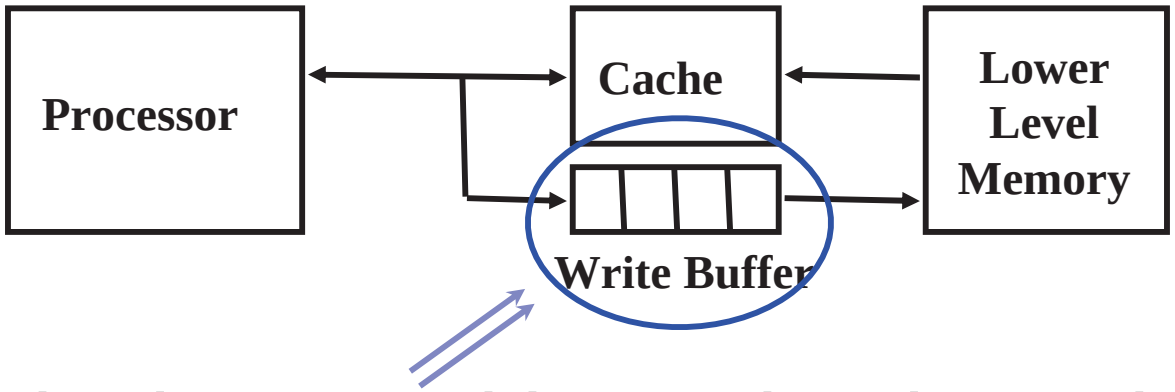
- Easy for Direct Mapped
- Set Associative or Fully Associative:
  - LRU (Least Recently Used)
  - Random
  - FIFO

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	Write-Back
Policy	Data written to cache block also written to lower-level memory	Write data only to the cache Update lower level when a block falls out of the cache
Debug	Easy	Hard
Do read misses produce writes?	No	Yes
Do repeated writes make it to lower level?	Yes	No

Write Buffers



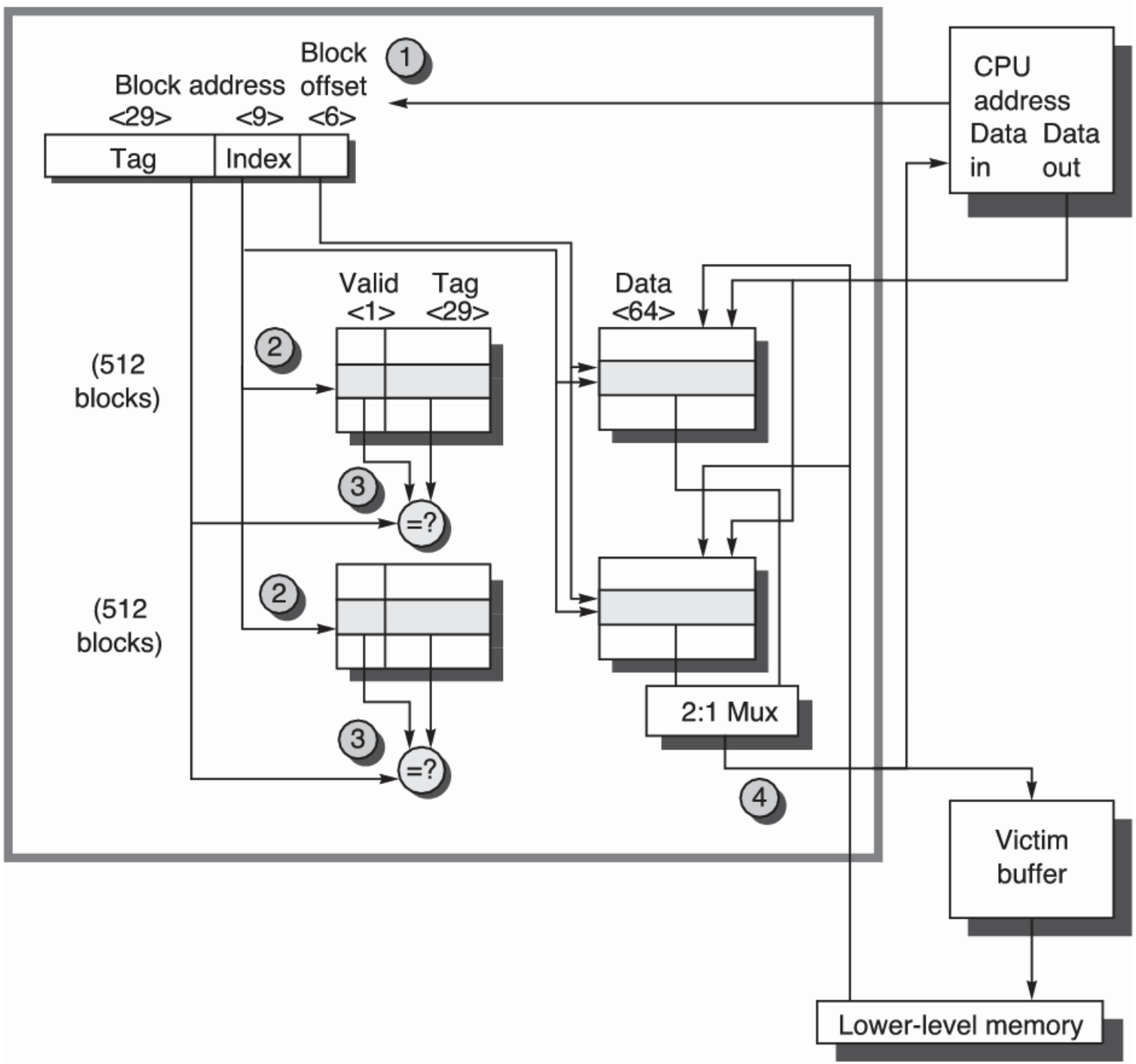
Holds data awaiting write-through to lower level memory

Write Buffers

- Why a write buffer?
- So CPU does not stall
- Why a buffer, why not just one register?
- Bursts of writes are common
- Are Read After Write (RAW) hazards an issue for write buffer?
- Yes! Drain buffer before next read, or send read 1st after check write buffers

Example

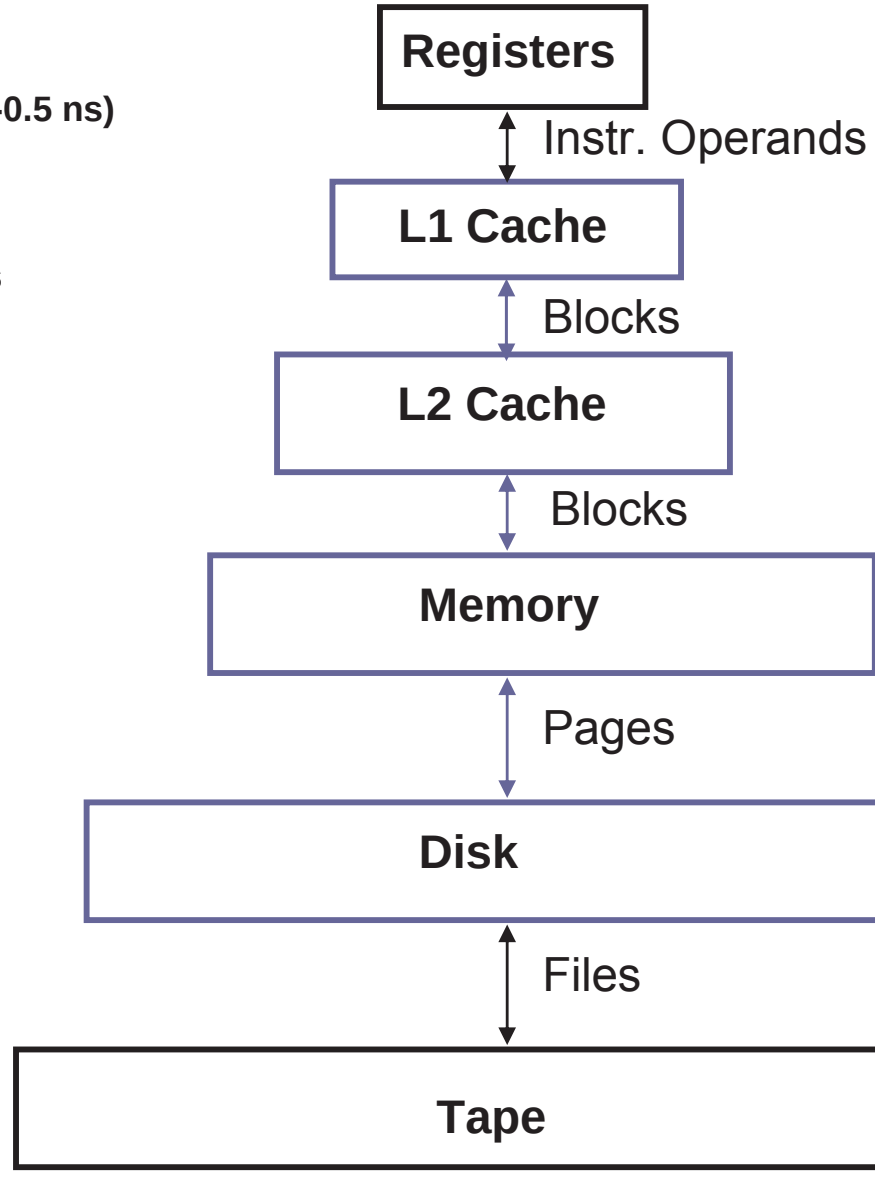
- Alpha 21264 data cache
- 64 KB, 64-byte blocks
- two-way set associative
  - One cache element contains 512 blocks
- Victim buffer = write buffer


Block address
<29>
Block offset
<9>
<6>
Tag
Index
1
CPU address
Data in
Data out
Valid
<1>
Tag
<29>
Data
<64>
(512 blocks)
2
3
=?
(512 blocks)
2
3
=?
2:1 Mux
4
Victim buffer
Lower-level memory

Levels of Memory Hierarchy

Capacity
Access Time
Cost

Staging
Transfer Unit



CPU Registers
100s Bytes
300 – 500 ps (0.3-0.5 ns)

prog./compiler
1-8 bytes

Upper Level

L1 and L2 Cache
10s-100s K Bytes
~1 ns - ~10 ns
$1000s/ GByte

cache cntl
32-64 bytes

cache cntl
64-128 bytes

Main Memory
G Bytes
80ns- 200ns
~ $100/ GByte

OS
4K-8K bytes

Disk
10s T Bytes, 10 ms
(10,000,000 ns)
~ $1 / GByte

user/operator
Mbytes

Tape
infinite
sec-min
~$1 / GByte

faster

Larger

Lower Level

Multilevel Caches

- Primary cache attached to CPU
  - Small, but fast
- Level-2 cache services misses from primary cache
  - Larger, slower, but still faster than main memory
- Main memory services L-2 cache misses
- Some high-end systems include L-3 cache

Multilevel Cache Example

- Given
  - CPU base CPI = 1, clock rate = 4GHz
  - Miss rate/instruction = 2%
  - Main memory access time = 100ns
- With just primary cache
  - Miss penalty = 100ns/0.25ns = 400 cycles
  - Effective CPI = 1 + 0.02 × 400 = 9

Example (cont.)

- Now add L-2 cache
  - Access time = 5ns
  - Global miss rate to main memory = 0.5%
- Primary miss with L-2 hit
  - Penalty = 5ns/0.25ns = 20 cycles
- Primary miss with L-2 miss
  - Extra penalty = 500 cycles
- CPI = 1 + 0.02 × 20 + 0.005 × 400 = 3.4
- Performance ratio = 9/3.4 = 2.6

Multilevel Cache Considerations

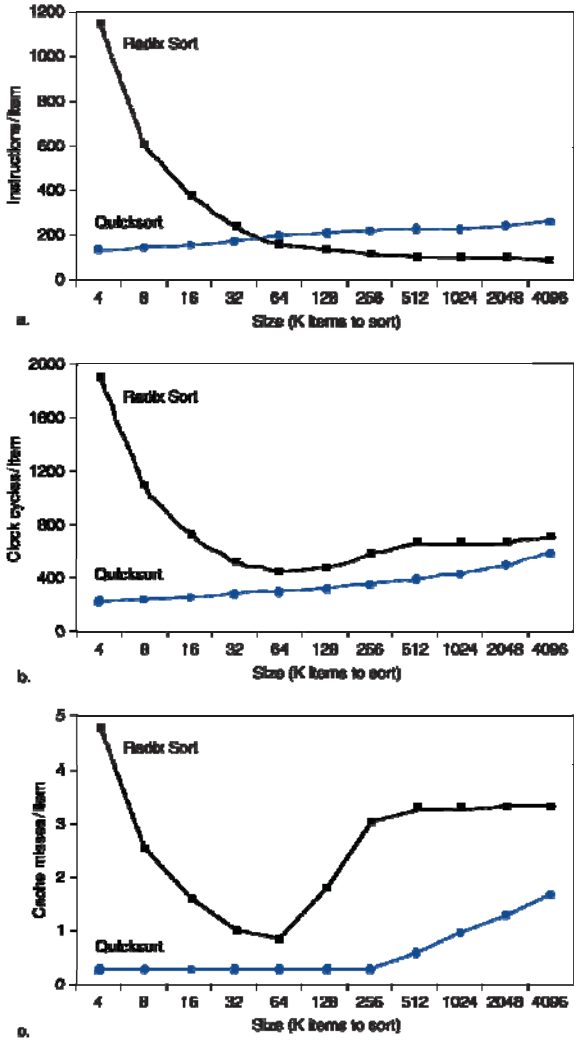
- Primary cache
  - Focus on minimal hit time
- L-2 cache
  - Focus on low miss rate to avoid main memory access
  - Hit time has less overall impact
- Results
  - L-1 cache usually smaller than a single cache
  - L-1 block size smaller than L-2 block size

Interactions with Advanced CPUs

- Out-of-order CPUs can execute instructions during cache miss
  - Pending store stays in load/store unit
  - Dependent instructions wait in reservation stations
    - Independent instructions continue
- Effect of miss depends on program data flow
  - Much harder to analyse
  - Use system simulation

Interactions with Software

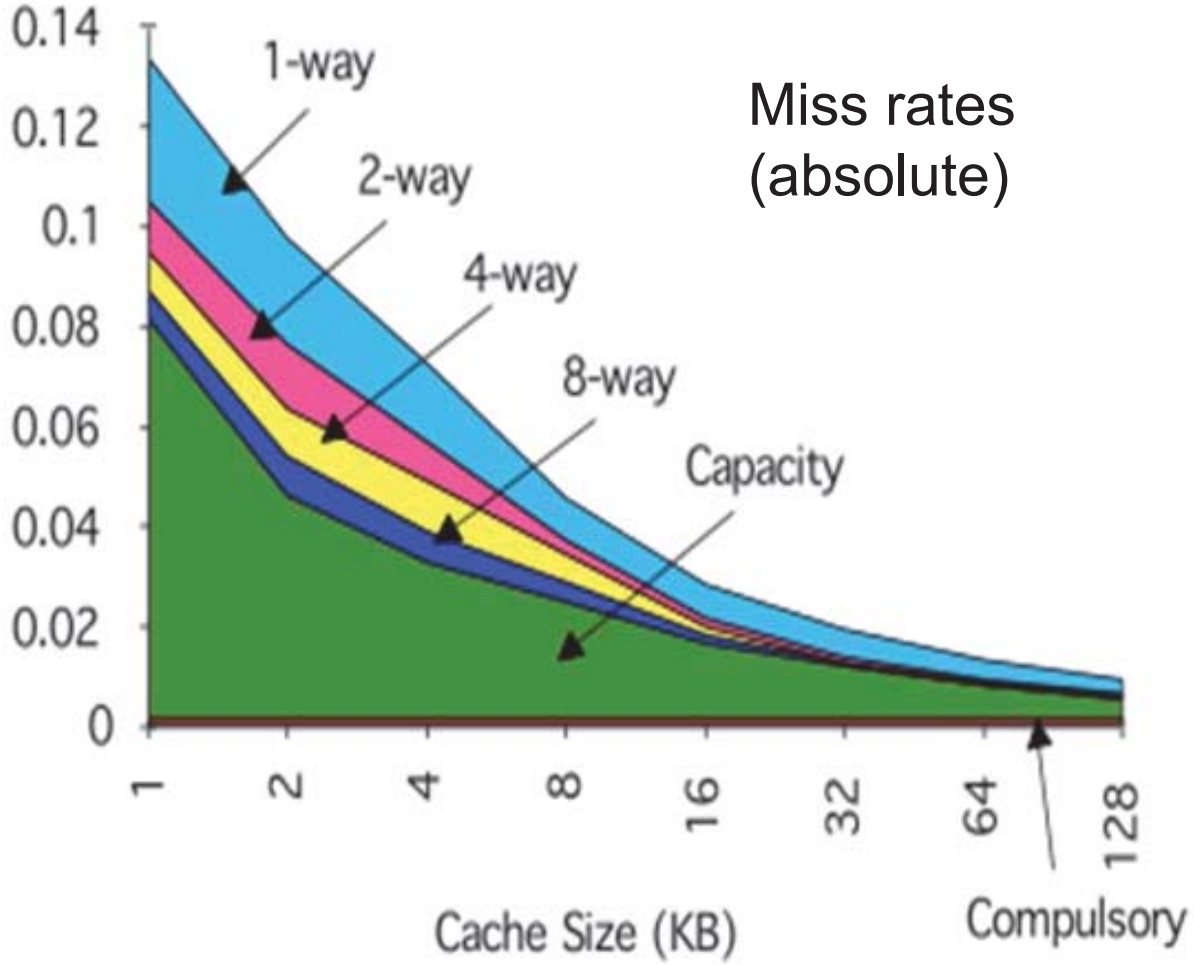
- Misses depend on memory access patterns
  - Algorithm behavior
  - Compiler optimization for memory access



Reducing Miss Rate

- Larger cache size
  - Capacity misses
- Larger block size
  - Compulsory misses – very first access to a block *cannot* be in the cache
- Higher associativity
  - Conflict misses

Capacity, Conflict, and Compulsory



Reducing Hit Time

- Small and simple caches
  - Tag comparison is critical
- Pipelined cache access
  - Pentium Pro – Pentium III: 2 clocks
  - Pentium 4: 4 clocks
  - Increases bandwidth and latency!
- Trace cache
  - Dynamic sequence of instructions
  - Including taken branches – branch prediction is part of the cache

Summary

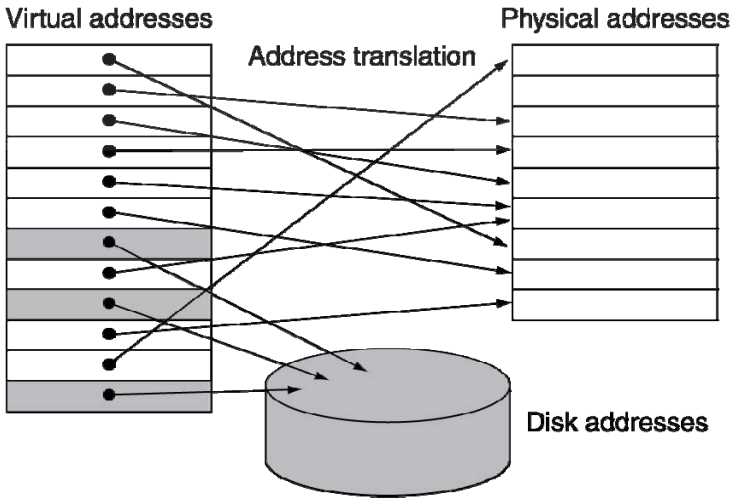
- Growing CPU/memory gap
- Caching to bridge the gap
  - Temporal and spatial locality
  - Illusion of large, fast memory
- Large part of the transistor budged used for on-chip caches
- Multilevel caches
  - L1 usually instruction and data cache
  - L2… unified

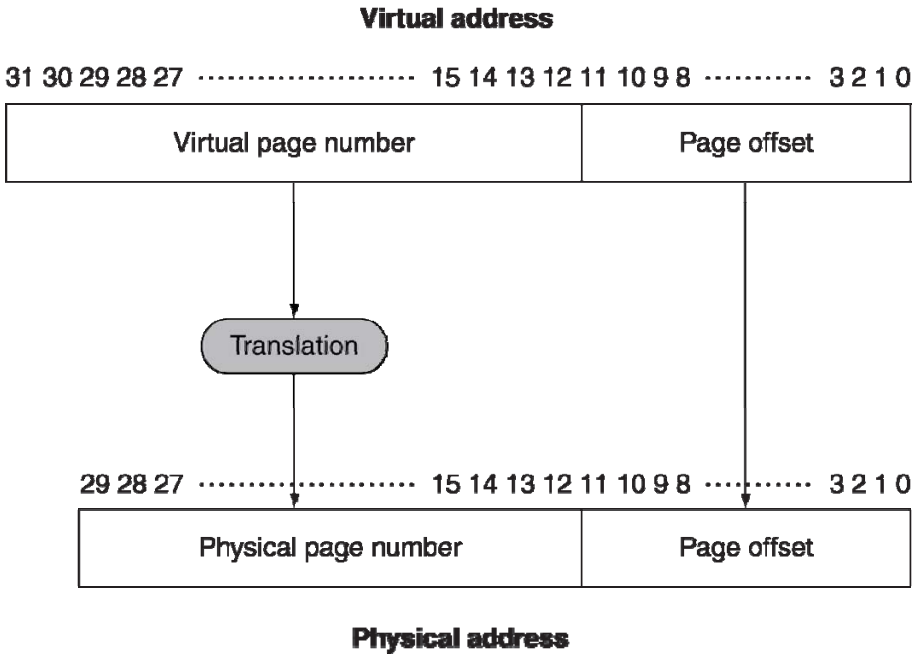
Virtual Memory

- Use main memory as a “cache” for secondary (disk) storage
  - Managed jointly by CPU hardware and the operating system (OS)
- Programs share main memory
  - Each gets a private virtual address space holding its frequently used code and data
  - Protected from other programs
- CPU and OS translate virtual addresses to physical addresses
  - VM “block” is called a page
  - VM translation “miss” is called a page fault

Address Translation

- Fixed-size pages (e.g., 4K)





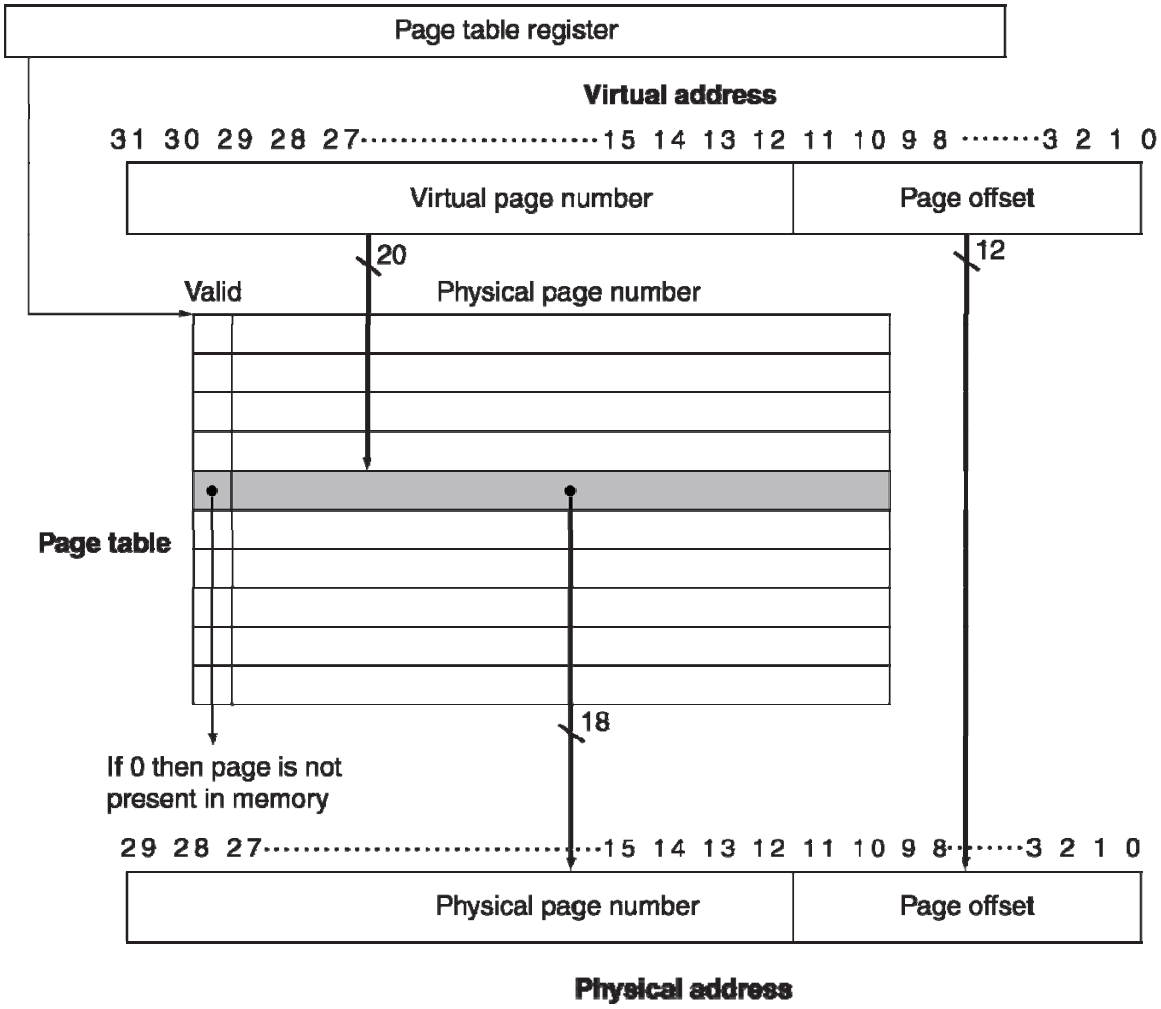
Page Fault Penalty

- On page fault, the page must be fetched from disk
  - Takes millions of clock cycles
  - Handled by OS code
- Try to minimize page fault rate
  - Fully associative placement
  - Smart replacement algorithms

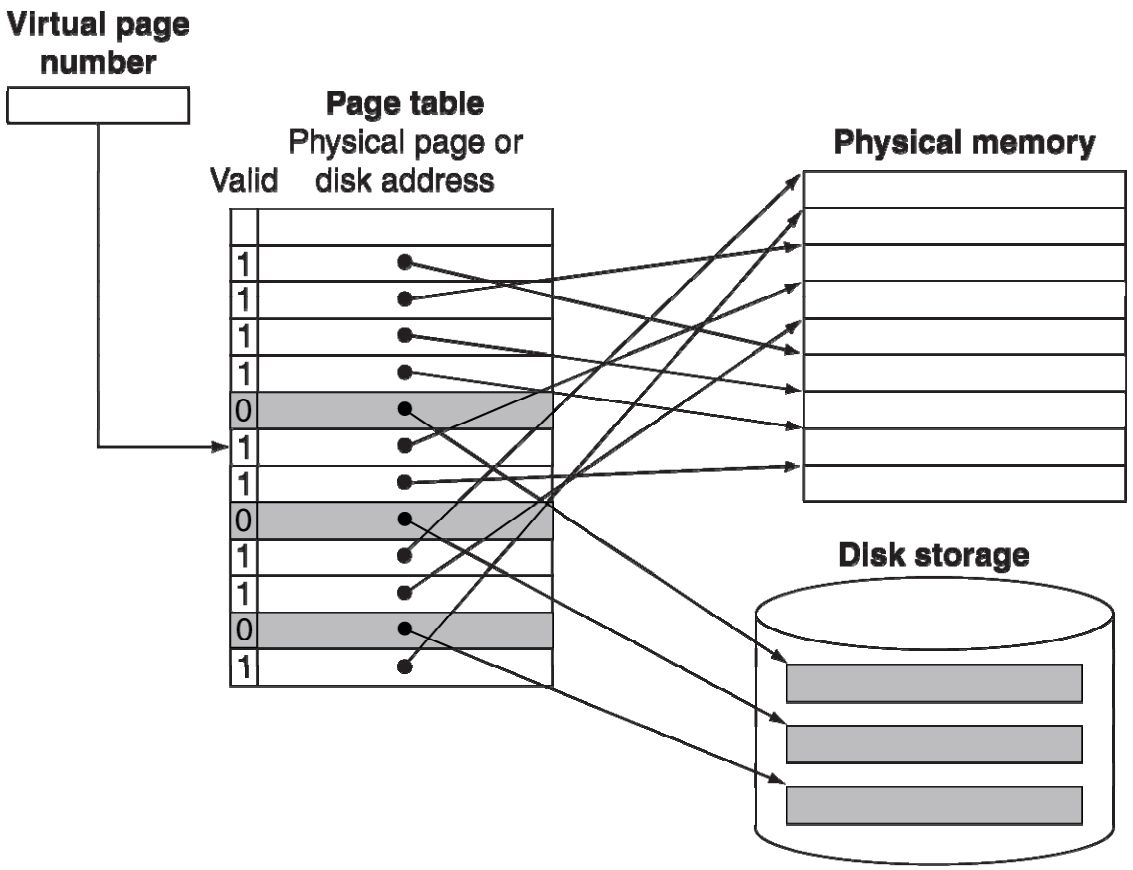
Page Tables

- Stores placement information
  - Array of page table entries, indexed by virtual page number
  - Page table register in CPU points to page table in physical memory
- If page is present in memory
  - PTE stores the physical page number
  - Plus other status bits (referenced, dirty, …)
- If page is not present
  - PTE can refer to location in swap space on disk

Translation Using a Page Table



Mapping Pages to Storage



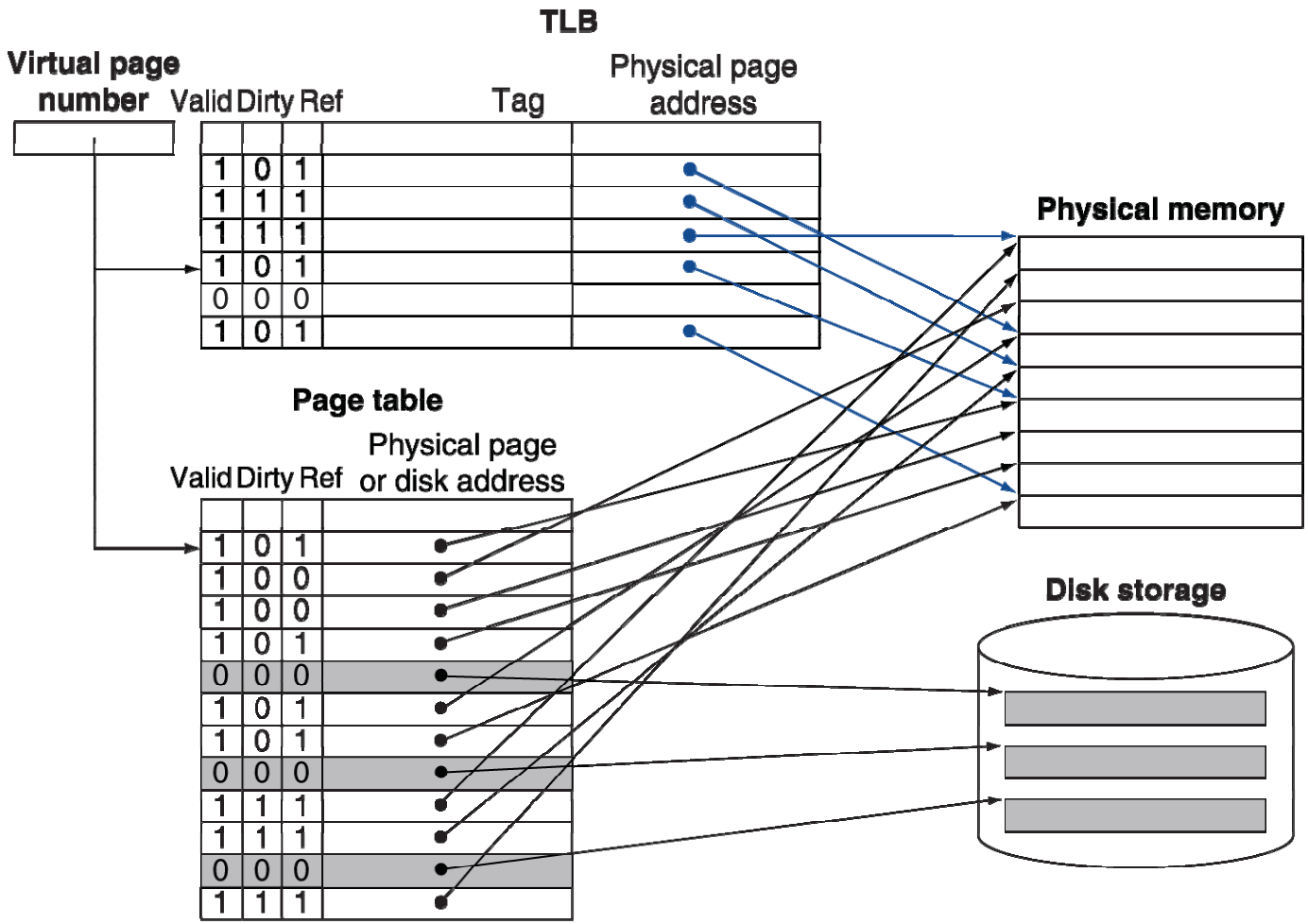
Replacement and Writes

- To reduce page fault rate, prefer least-recently used (LRU) replacement
  - Reference bit (aka use bit) in PTE set to 1 on access to page
  - Periodically cleared to 0 by OS
  - A page with reference bit = 0 has not been used recently
- Disk writes take millions of cycles
  - Block at once, not individual locations
  - Write through is impractical
  - Use write-back
  - Dirty bit in PTE set when page is written

Fast Translation Using a TLB

- Address translation would appear to require extra memory references
  - One to access the PTE
  - Then the actual memory access
- But access to page tables has good locality
  - So use a fast cache of PTEs within the CPU
  - Called a Translation Look-aside Buffer (TLB)
  - Typical: 16–512 PTEs, 0.5–1 cycle for hit, 10–100 cycles for miss, 0.01%–1% miss rate
  - Misses could be handled by hardware or software

Fast Translation Using a TLB



TLB Misses

- If page is in memory
  - Load the PTE from memory and retry
  - Could be handled in hardware
    - Can get complex for more complicated page table structures
  - Or in software
    - Raise a special exception, with optimized handler
- If page is not in memory (page fault)
  - OS handles fetching the page and updating the page table
  - Then restart the faulting instruction

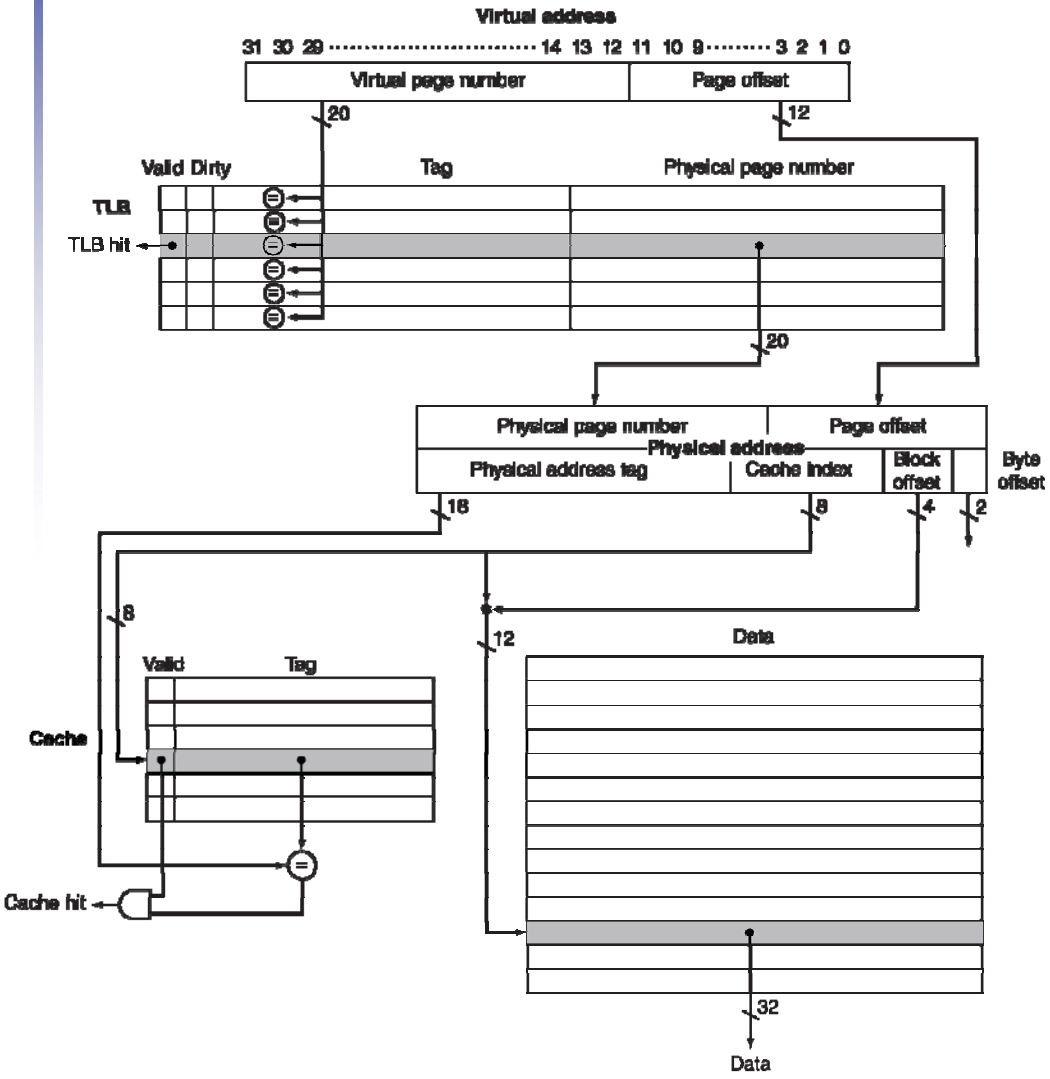
TLB Miss Handler

- TLB miss indicates
  - Page present, but PTE not in TLB
  - Page not preset
- Must recognize TLB miss before destination register overwritten
  - Raise exception
- Handler copies PTE from memory to TLB
  - Then restarts instruction
  - If page not present, page fault will occur

Page Fault Handler

- Use faulting virtual address to find PTE
- Locate page on disk
- Choose page to replace
  - If dirty, write to disk first
- Read page into memory and update page table
- Make process runnable again
  - Restart from faulting instruction

TLB and Cache Interaction



- If cache tag uses physical address
  - Need to translate before cache lookup
- Alternative: use virtual address tag
  - Complications due to aliasing
    - Different virtual addresses for shared physical address