Chapter 5-2

Large and Fast: Exploiting Memory Hierarchy

Interconnecting Components

- Need interconnections between
  - CPU, memory, I/O controllers
- Bus: shared communication channel
  - Parallel set of wires for data and synchronization of data transfer
  - Can become a bottleneck
- Performance limited by physical factors
  - Wire length, number of connections
- More recent alternative: high-speed serial connections with switches
  - Like networks

Bus Types

- Processor-Memory buses
  - Short, high speed
  - Design is matched to memory organization
- I/O buses
  - Longer, allowing multiple connections
  - Specified by standards for interoperability
  - Connect to processor-memory bus through a bridge

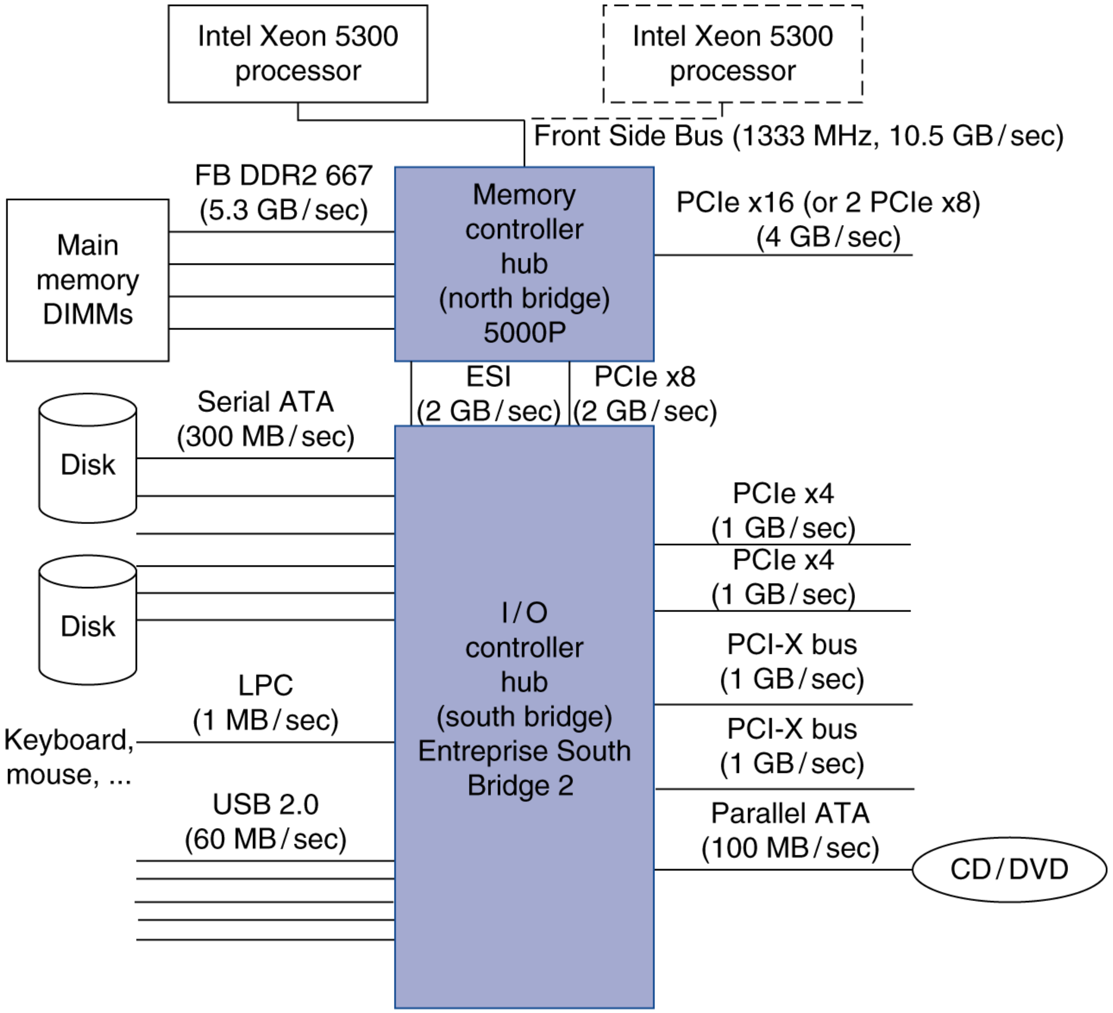
Bus Signals and Synchronization

- Data lines
  - Carry address and data
  - Multiplexed or separate
- Control lines
  - Indicate data type, synchronize transactions
- Synchronous
  - Uses a bus clock
- Asynchronous
  - Uses request/acknowledge control lines for handshaking

I/O Bus Examples

	Firewire	USB 2.0	PCI Express	Serial ATA	Serial Attached SCSI
Intended use	External	External	Internal	Internal	External
Devices per channel	63	127	1	1	4
Data width	4	2	2/lane	4	4
Peak bandwidth	50MB/s or 100MB/s	0.2MB/s, 1.5MB/s, or 60MB/s	250MB/s/lane 1×, 2×, 4×, 8×, 16×, 32×	300MB/s	300MB/s
Hot pluggable	Yes	Yes	Depends	Yes	Yes
Max length	4.5m	5m	0.5m	1m	8m
Standard	IEEE 1394	USB Implementers Forum	PCI-SIG	SATA-IO	INCITS TC T10

Typical x86 PC I/O System



Intel Xeon 5300 processor
Intel Xeon 5300 processor
Front Side Bus (1333 MHz, 10.5 GB/sec)
FB DDR2 667 (5.3 GB/sec)
Main memory DIMMs
Memory controller hub (north bridge) 5000P
PCIe x16 (or 2 PCIe x8) (4 GB/sec)
ESI (2 GB/sec)
PCIe x8 (2 GB/sec)
Serial ATA (300 MB/sec)
Disk
Disk
PCIe x4 (1 GB/sec)
PCIe x4 (1 GB/sec)
PCI-X bus (1 GB/sec)
PCI-X bus (1 GB/sec)
Parallel ATA (100 MB/sec)
CD/DVD
I/O controller hub (south bridge) Entreprise South Bridge 2
LPC (1 MB/sec)
Keyboard, mouse, ...
USB 2.0 (60 MB/sec)

RAID

- RAID: Redundant Array of Inexpensive (Independent) Disks (Section 5.11)
  - Use multiple smaller disks (c.f. one large disk)
  - Parallelism improves performance
  - Plus extra disk(s) for redundant data storage
- Provides fault tolerant storage system
  - Especially if failed disks can be "hot swapped"
- RAID 0
  - No redundancy ("AID"?)
    - Just stripe data over multiple disks
  - But it does improve performance

RAID 1 & 2

- RAID 1: Mirroring
  - N + N disks, replicate data
    - Write data to both data disk and mirror disk
    - On disk failure, read from mirror
- RAID 2: Error correcting code (ECC)
  - N + E disks (e.g., 10 + 4)
  - Split data at bit level across N disks
  - Generate E-bit ECC
  - Too complex, not used in practice

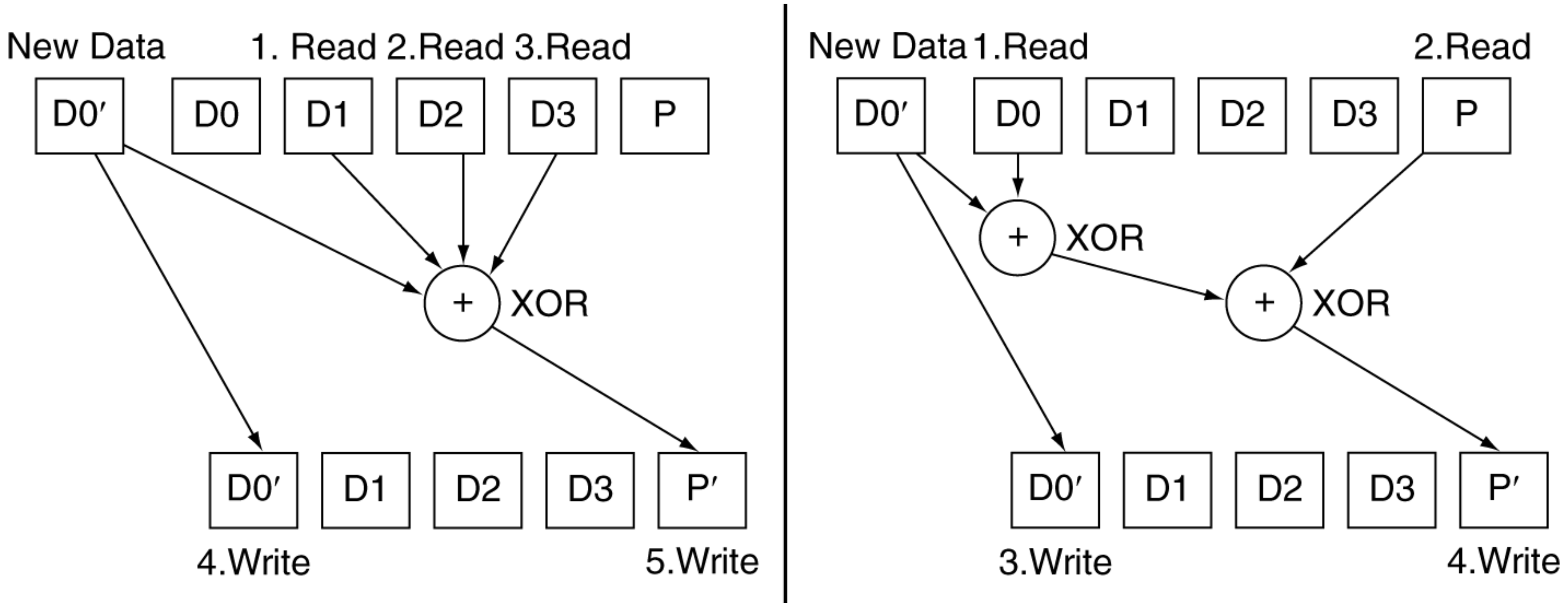
RAID 3: Bit-Interleaved Parity

- N + 1 disks
  - Data striped across N disks at byte level
  - Redundant disk stores parity
  - Read access
    - Read all disks
  - Write access
    - Generate new parity and update all disks
  - On failure
    - Use parity to reconstruct missing data
- Not widely used

RAID 4: Block-Interleaved Parity

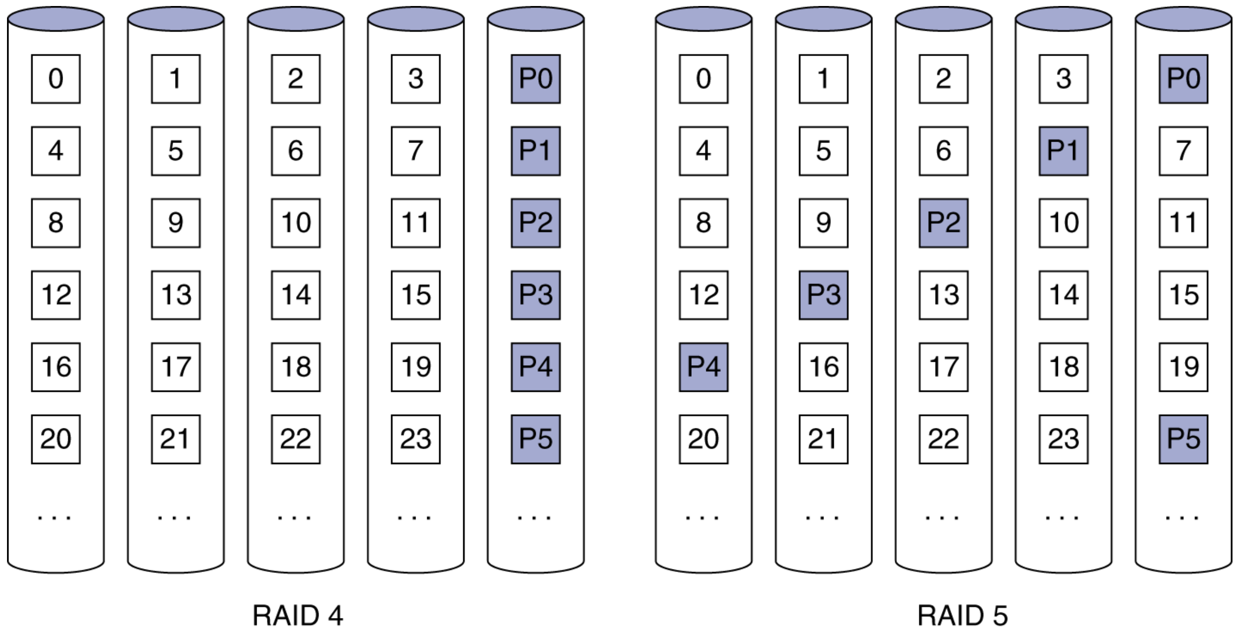
- N + 1 disks
  - Data striped across N disks at block level
  - Redundant disk stores parity for a group of blocks
  - Read access
    - Read only the disk holding the required block
  - Write access
    - Just read disk containing modified block, and parity disk
    - Calculate new parity, update data disk and parity disk
  - On failure
    - Use parity to reconstruct missing data
- Not widely used

RAID 3 vs RAID 4



RAID 5: Distributed Parity

- N + 1 disks
  - Like RAID 4, but parity blocks distributed across disks
    - Avoids parity disk being a bottleneck
- Widely used



RAID 6: P + Q Redundancy

- N + 2 disks
  - Like RAID 5, but two lots of parity
  - Greater fault tolerance through more redundancy
- Multiple RAID
  - More advanced systems give similar fault tolerance with better performance

RAID Summary

- RAID can improve performance and availability
  - High availability requires hot swapping
- Assumes independent disk failures
  - Too bad if the building burns down!
- See “Hard Disk Performance, Quality and Reliability”
  - http://www.pcguide.com/ref/hdd/perf/index.htm

I/O System Design

- Satisfying latency requirements
  - For time-critical operations
  - If system is unloaded
    - Add up latency of components
- Maximizing throughput
  - Find “weakest link” (lowest-bandwidth component)
  - Configure to operate at its maximum bandwidth
  - Balance remaining components in the system
- If system is loaded, simple analysis is insufficient
  - Need to use queuing models or simulation

Virtual Machines

- Virtual Machine (VM): Host computer emulates guest operating system and machine resources
  - Improved isolation of multiple guests
  - Avoids security and reliability problems
  - Aids sharing of resources
- Virtualization has some performance impact
  - Feasible with modern high-performance comptuers
- Examples
  - IBM VM/370 (1970s technology!)
  - VMWare
  - Microsoft Virtual PC

Virtual Machine Monitor (VMM)

- VMM maps virtual resources to physical resources
  - Memory, I/O devices, CPUs
- Guest code runs on native machine in user mode
  - Traps to VMM on privileged instructions and access to protected resources
- Guest OS may be different from host OS
- VMM handles real I/O devices
  - Emulates generic virtual I/O devices for guest

Example: Timer Virtualization

- In native machine, on timer interrupt
  - OS suspends current process, handles interrupt, selects and resumes next process
- With Virtual Machine Monitor
  - VMM suspends current VM, handles interrupt, selects and resumes next VM
- If a VM requires timer interrupts
  - VMM emulates a virtual timer
  - Emulates interrupt for VM when physical timer interrupt occurs

Instruction Set Support

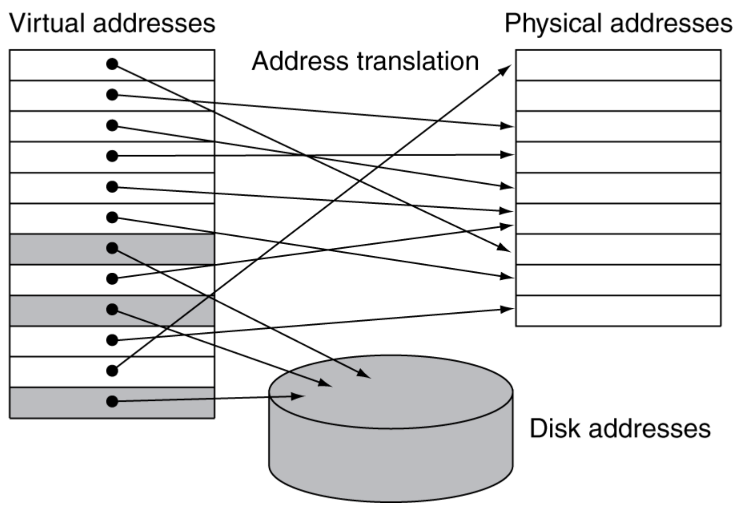
- User and System modes
- Privileged instructions only available in system mode
  - Trap to system if executed in user mode
- All physical resources only accessible using privileged instructions
  - Including page tables, interrupt controls, I/O registers
- Renaissance of virtualization support
  - Current ISAs (*e.g.*, x86) adapting

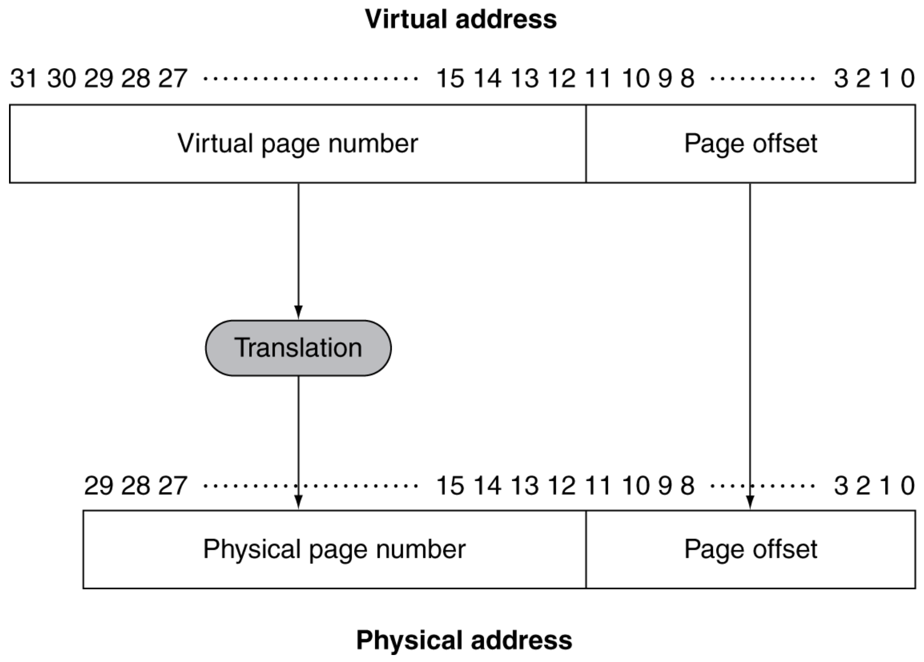
Virtual Memory

- Virtual Memory uses 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)





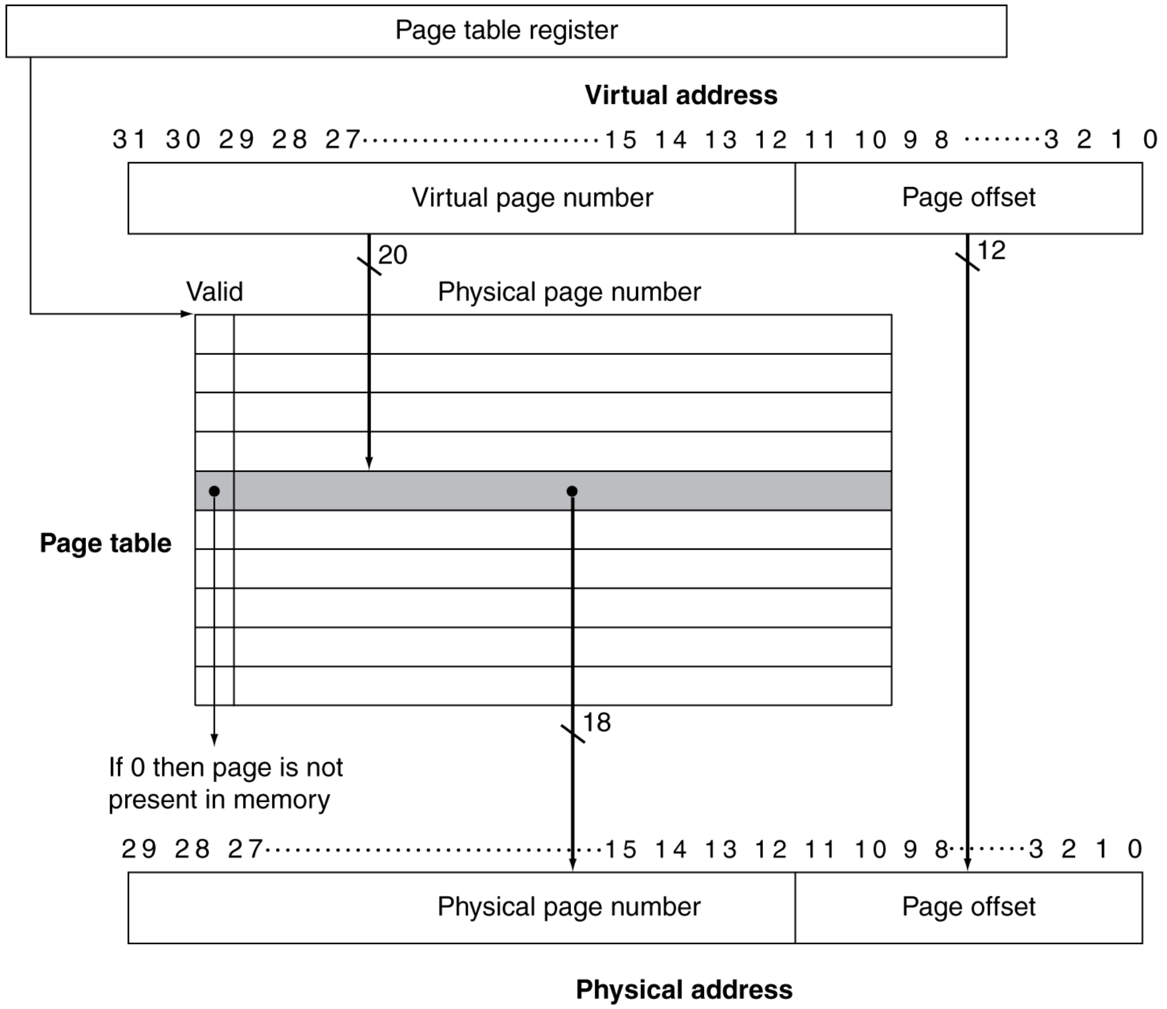
Page Fault Penalty

- 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

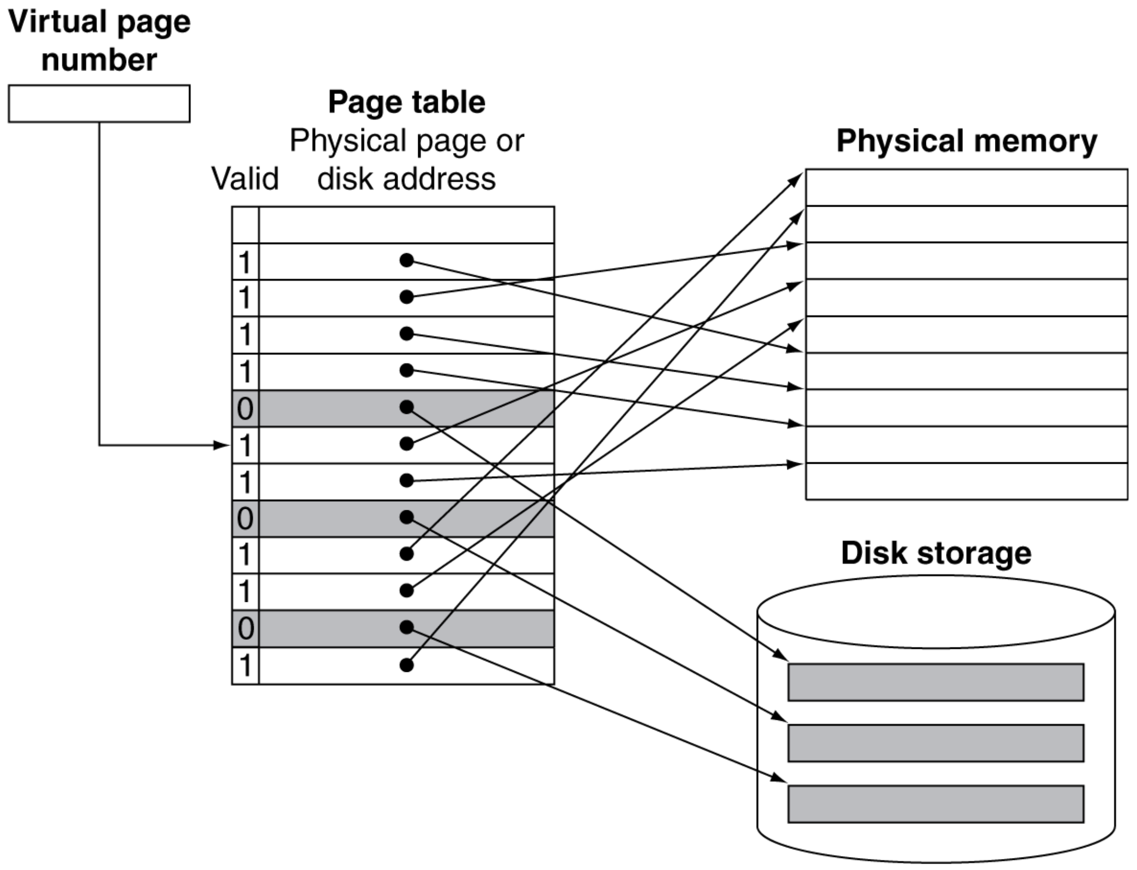
Page Tables

- Stores placement information
  - Array of page table entries (PTE), indexed by virtual page number
  - Page table register in CPU points to page table in physical memory
- If page is present in memory (valid=1)
  - PTE stores the physical page number
  - Plus other status bits (referenced, dirty, …)
- If page is not present (valid=0)
  - PTE can refer to location in swap space on disk

Translation Using a Page Table



Mapping Pages to Storage



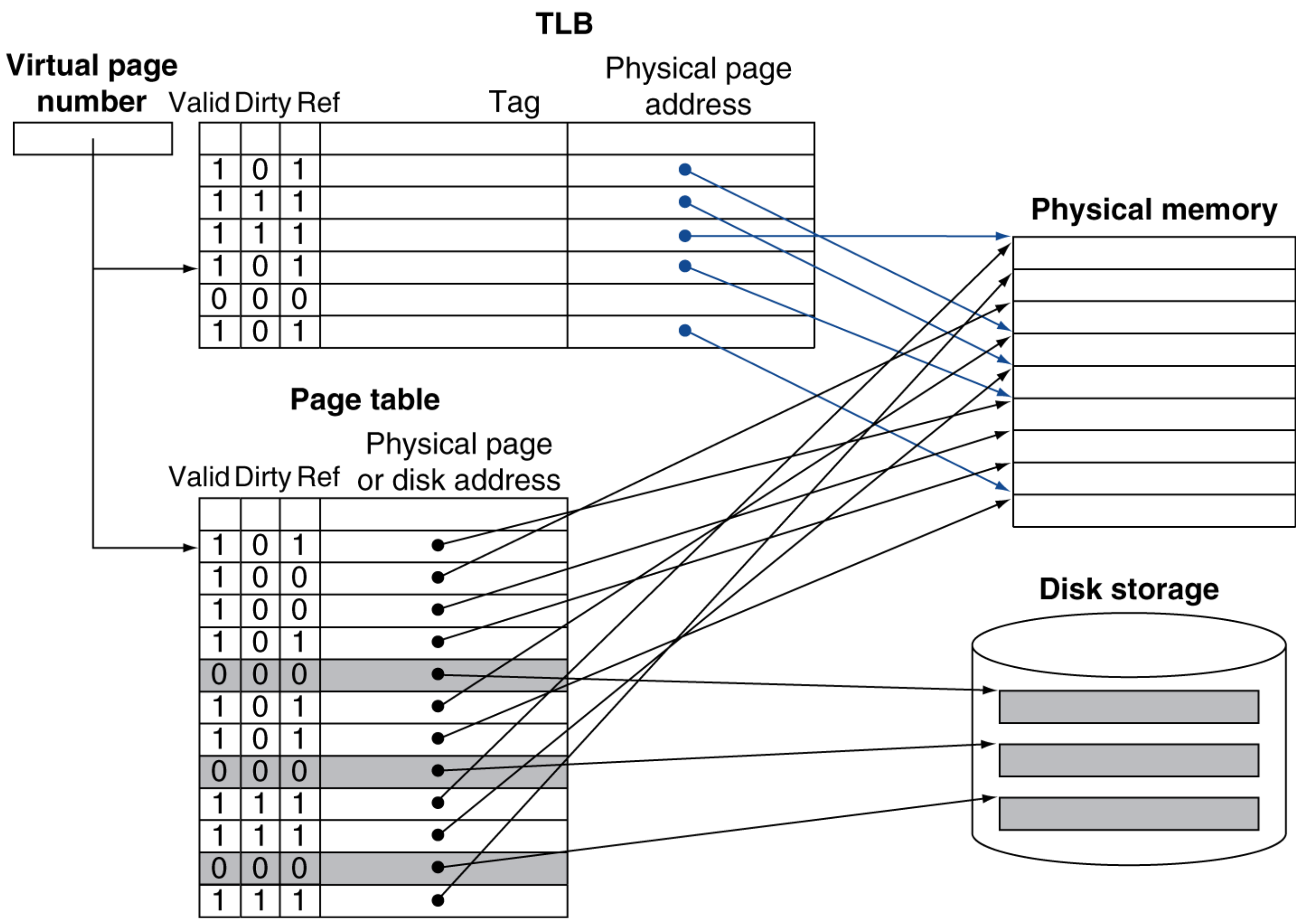
Replacement and Writes

- To reduce page fault rate, prefer least-recently used (LRU) replacement
  - Reference bit (*aka* use bit) in page table entry (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

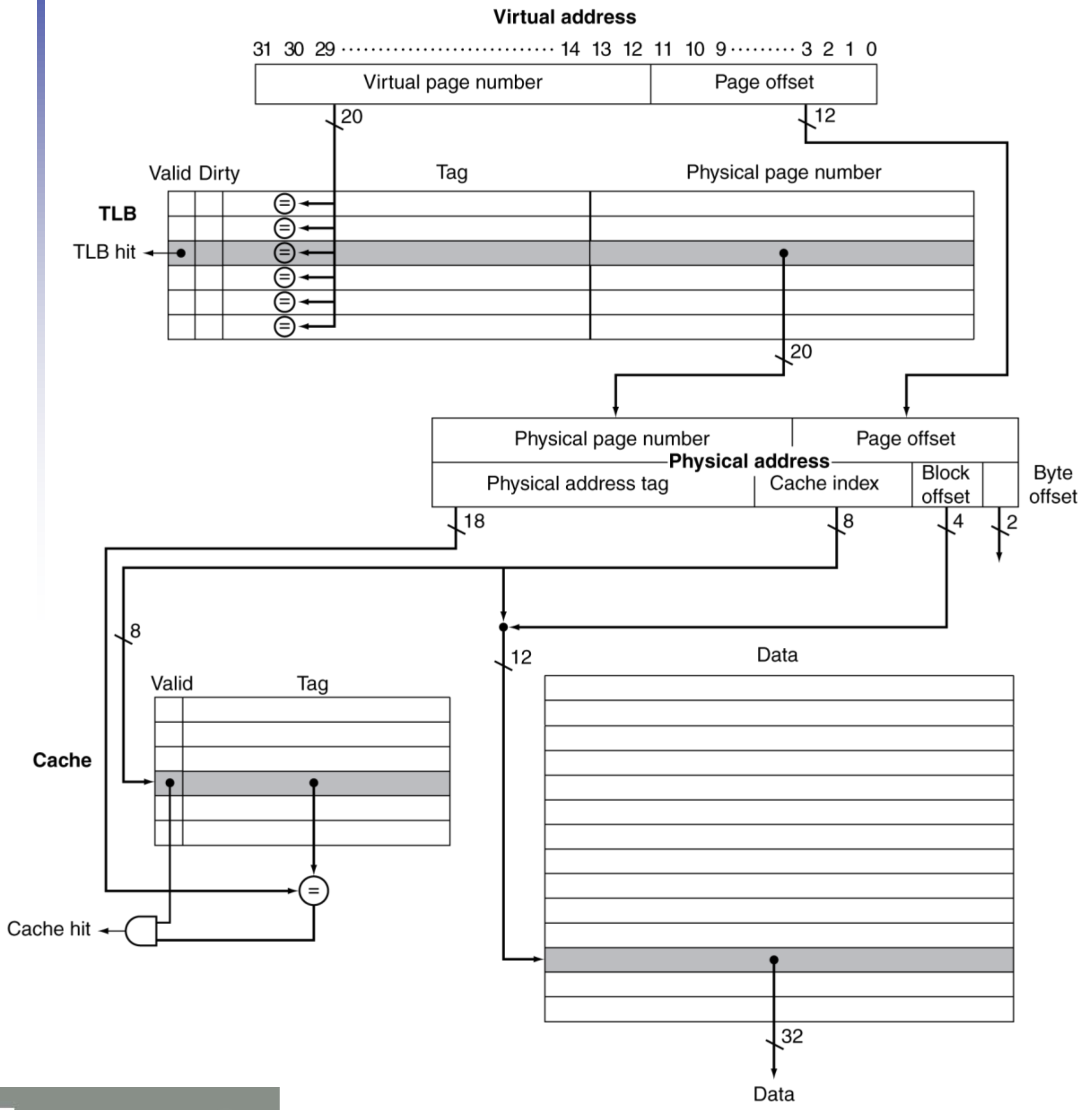
TLB Miss Handler

- TLB miss indicates
  - Page present, but PTE not in TLB
  - Or Page not present
- 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

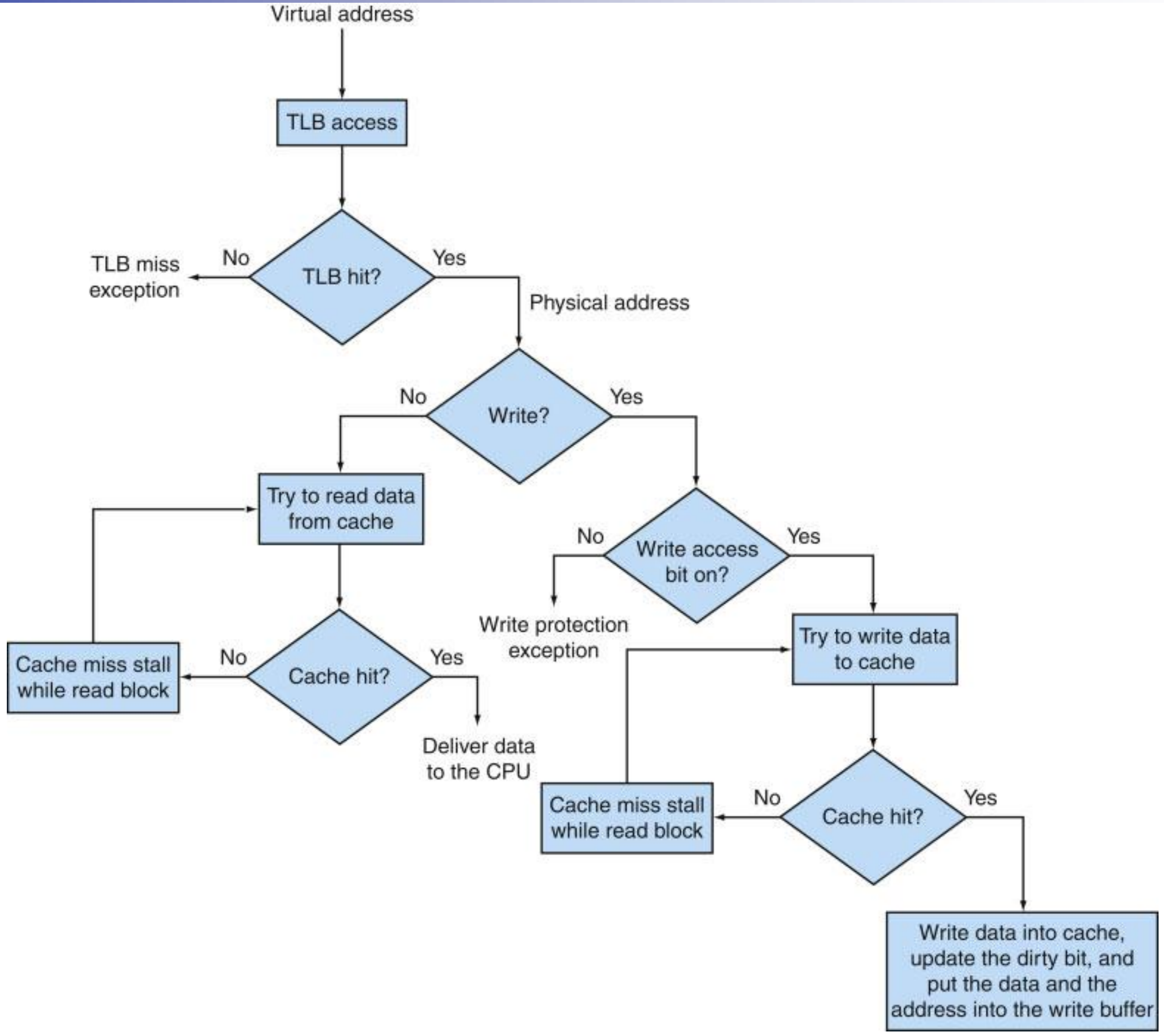
- Use faulting virtual address to find PTE
- Locate page on disk
- Choose page to replace
  - If dirty, write to disk first
- Read page into memory and update page table
- Make process runnable again
  - Restart from faulting instruction

TLB and Cache in FastMath



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

Read or Write-through in TLB and Cache


Virtual address
TLB access
TLB hit?
No
Yes
TLB miss exception
Physical address
Write?
No
Yes
Try to read data from cache
Write access bit on?
No
Yes
Write protection exception
Try to write data to cache
Cache miss stall while read block
Cache hit?
No
Yes
Deliver data to the CPU
Cache miss stall while read block
Cache hit?
No
Yes
Write data into cache, update the dirty bit, and put the data and the address into the write buffer

Memory Protection

- Different tasks can share parts of their virtual address spaces
  - But need to protect against errant access
  - Requires OS assistance
- Hardware support for OS protection
  - Privileged supervisor mode (*aka* kernel mode)
  - Privileged instructions
  - Page tables and other state information only accessible in supervisor mode
  - System call exception (*e.g.*, syscall in MIPS)

The Memory Hierarchy

The BIG Picture

- Common principles apply at all levels of the memory hierarchy
  - Based on notions of caching
- At each level in the hierarchy
  - Block placement
  - Finding a block
  - Replacement on a miss
  - Write policy

Block Placement

- Determined by associativity
  - Direct mapped (1-way associative)
    - One choice for placement
  - n-way set associative
    - n choices within a set
  - Fully associative
    - Any location
- Higher associativity reduces miss rate
  - Increases complexity, cost, and access time

Finding a Block

Associativity	Location method	Tag comparisons
Direct mapped	Index	1
n-way set associative	Set index, then search entries within the set	n
Fully associative	Search all entries	#entries
	Full lookup table	0

- Hardware caches
  - Reduce comparisons to reduce cost
- Virtual memory
  - Full (page-)table lookup makes full associativity feasible
  - Benefit in reduced miss rate

Replacement

- Choice of entry to replace on a miss
  - Least recently used (LRU)
    - Complex and costly hardware for high associativity
  - Random
    - Close to LRU, easier to implement
- Virtual memory
  - LRU approximation with hardware support

Write Policy

- Write-through
  - Update both upper and lower levels
  - Simplifies replacement, but may require write buffer
- Write-back
  - Update upper level only
  - Update lower level when block is replaced
  - Need to keep more state
- Virtual memory
  - Only write-back is feasible, given disk write latency

Sources of Misses

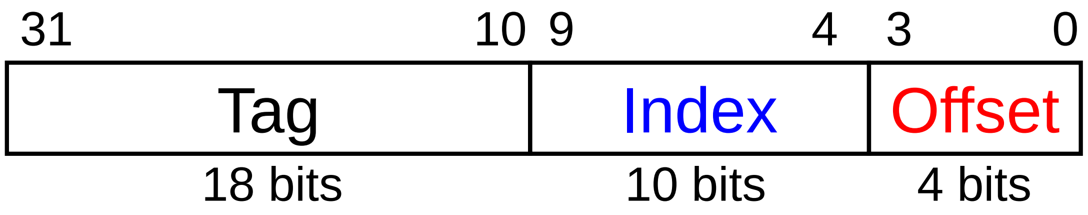
- Compulsory misses (*aka* cold start misses)
  - First access to a block
- Capacity misses
  - Due to finite cache size
  - A replaced block is later accessed again
- Conflict misses (*aka* collision misses)
  - In a non-fully associative cache
  - Due to competition for entries in a set
  - Would not occur in a fully associative cache of the same total size

Cache Design Trade-offs

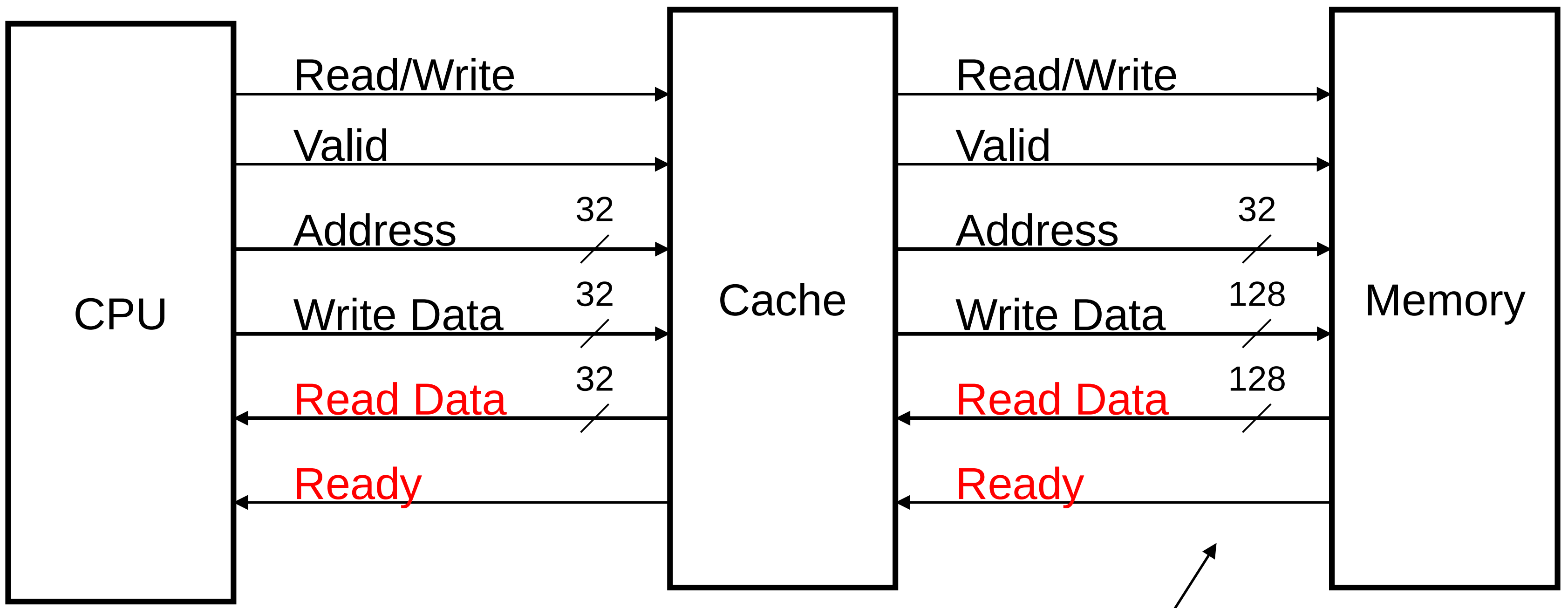
Design change	Effect on miss rate	Negative performance effect
Increase cache size	Decrease capacity misses	May increase access time
Increase associativity	Decrease conflict misses	May increase access time
Increase block size	Decrease compulsory misses	Increases miss penalty. For very large block size, may increase miss rate due to pollution.

Cache Control

- Example cache characteristics
  - **Direct-mapped, write-back, write allocate**
  - Block size: 4 words (16 bytes ⇒ 4-bit Offset)
  - Cache size: 16 KB (1024 blocks ⇒10-bit Index)
  - 32-bit byte addresses
  - Valid bit and dirty bit per block
  - Blocking cache
    - CPU waits until access is complete



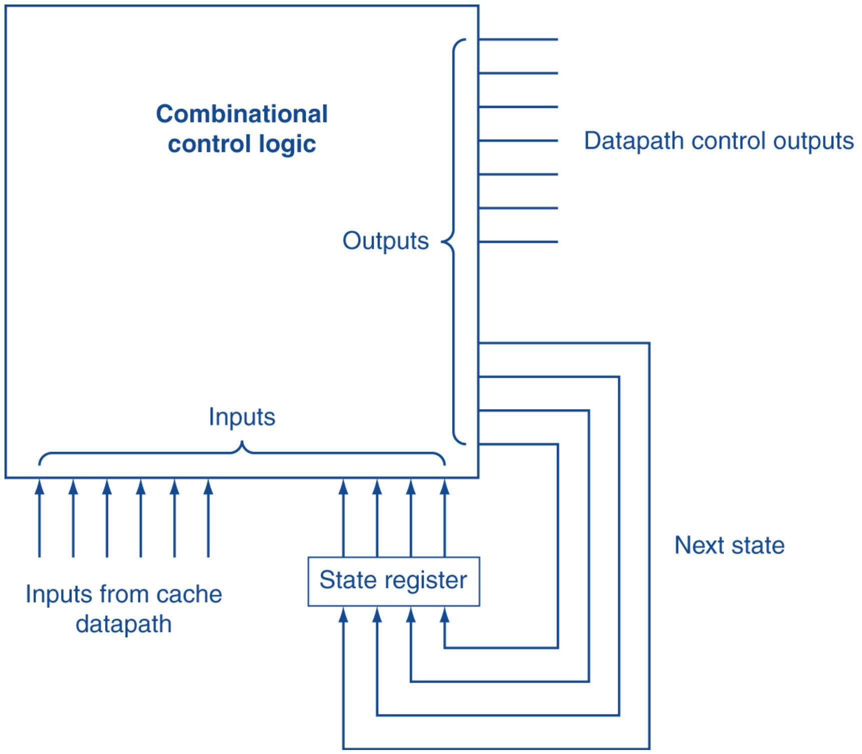
Interface Signals



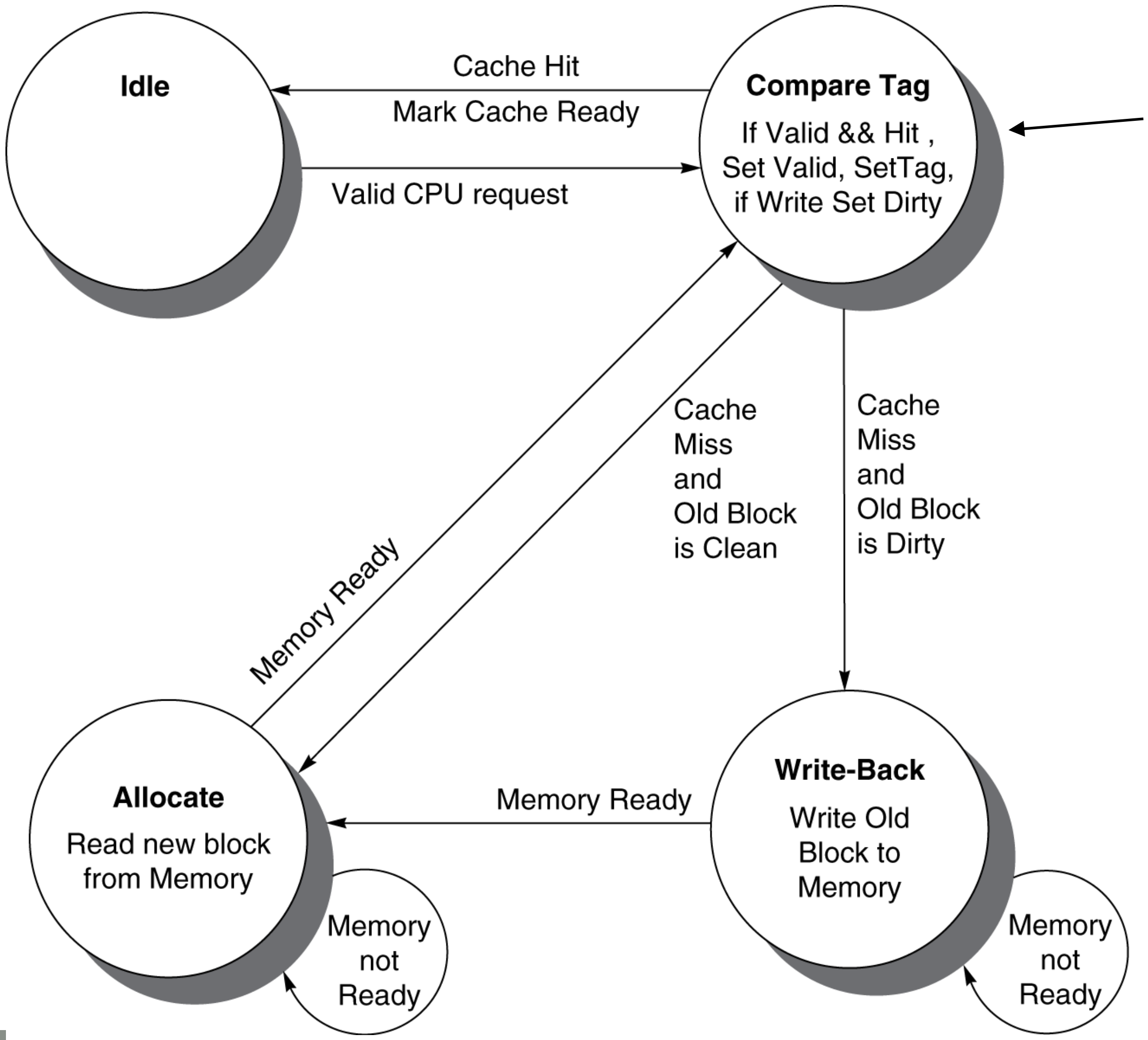
Multiple cycles per access

Finite State Machines

- Use an FSM to sequence control steps
- Set of states, transition on each clock edge
    - State values are binary encoded
    - Current state (cs) stored in a register
    - Next state (ns) ns = f_n (cs, current inputs)
- Control output signals
    - Output = f_o (cs)



Cache Controller FSM


Idle
Cache Hit
Mark Cache Ready
Valid CPU request
Compare Tag
If Valid && Hit , Set Valid, SetTag, if Write Set Dirty
Cache Miss and Old Block is Clean
Cache Miss and Old Block is Dirty
Memory Ready
Allocate
Read new block from Memory
Memory not Ready
Memory Ready
Write-Back
Write Old Block to Memory
Memory not Ready

Could partition into separate states to reduce clock cycle time

Cache Coherence Problem

- Suppose two CPU cores share a physical address space
  - Write-through caches

Time step	Event	CPU A's cache	CPU B's cache	Memory
0				0
1	CPU A reads X	0		0
2	CPU B reads X	0	0	0
3	CPU A writes 1 to X	1	0 ⇒1	1

Coherence Defined

- Informally: Return the most recently written value of the data requested by a Read
- Formally:
  - **P writes X; P reads X (no intervening writes)**
    $\Rightarrow$ read returns the value (written by **P)**
  - **P_1 writes X; P_2 reads X (sufficiently later)**
    $\Rightarrow$ read returns written value
    - *c.f.* CPU B reading X (=1) after Step 3
  - **P_1 writes X (=1); P_2 writes X (=2)**
    $\Rightarrow$ Need write serialization to guarantee that all processors see writes in the same order
    - End up with each processor sees its respective X value (must read **1** and **2** in the same order).

Cache Coherence Protocols

- Operations performed by caches in multiprocessors to ensure coherence
  - Migration of data to local caches
    - Reduces bandwidth for shared memory
  - Replication of read-shared data
    - Reduces contention for access
- Snooping protocols
  - Each cache monitors bus reads/writes
- Directory-based protocols
  - Caches and memory record sharing status of blocks in a directory

Invalidating Snooping Protocols

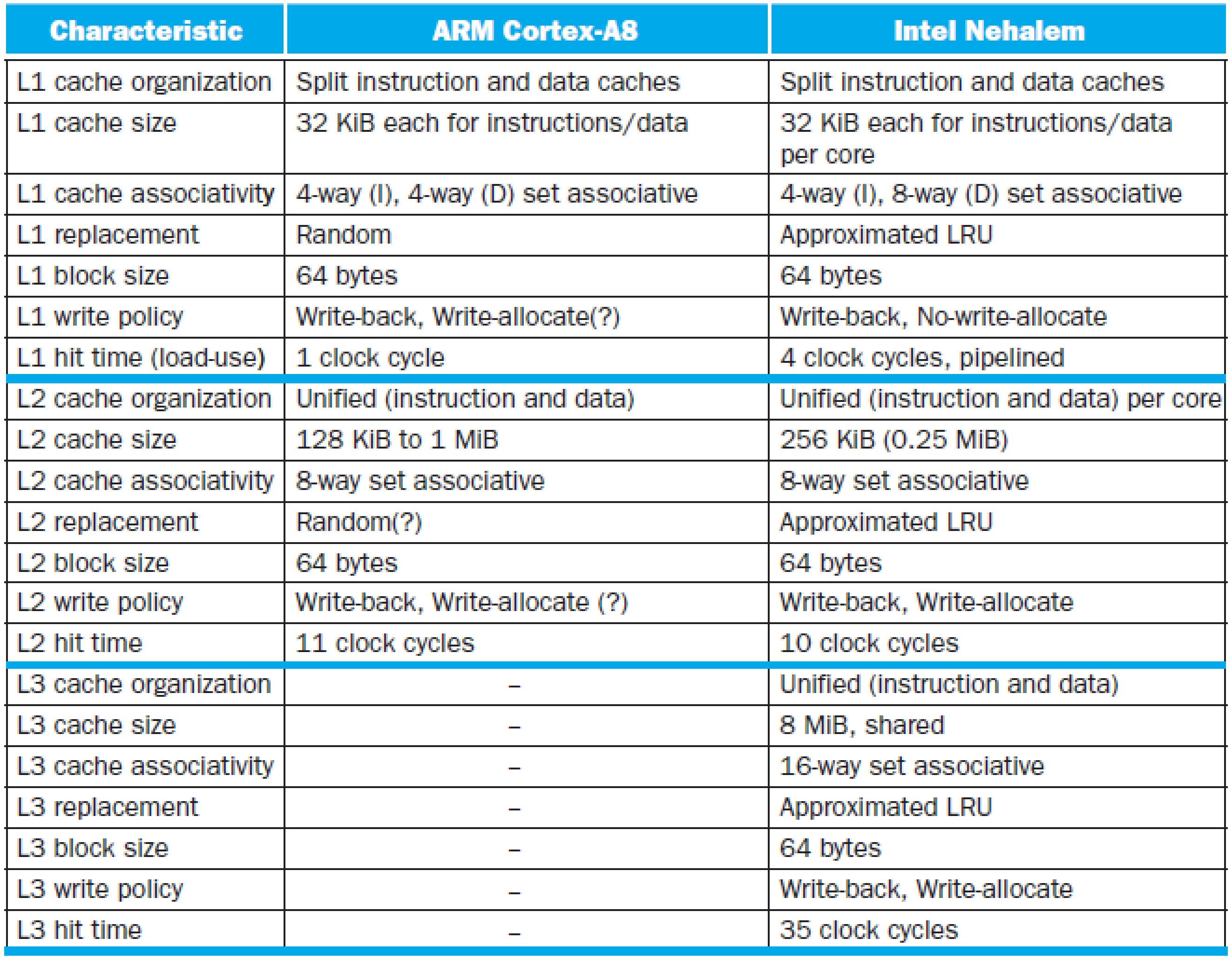
- Cache gets exclusive access to a block when it is to be written
  - Broadcasts an invalidate message on the bus
  - Subsequent read in another cache misses
    - Owning cache supplies updated value

CPU activity	Bus activity	CPU A's cache	CPU B's cache	Memory
				0
CPU A reads X	Cache miss for X	0		0
CPU B reads X	Cache miss for X	0	0	0
CPU A writes 1 to X	Invalidate for X	1		0
CPU B read X	Cache miss for X	1	1	1

Memory Consistency

- Consistency is related to when are writes seen by other processors
  - "Seen" means a read returns the written value
  - Can't be instantaneously
- Assumptions
  - A write completes only when all processors have seen it
  - A processor does not reorder writes with other accesses
- Consequence
  - P writes X then writes **Y** (see X first, then **Y**)
    $\Rightarrow$ all processors that see new **Y** also see new X
  - Processors can reorder reads, but not writes

Multilevel On-Chip Caches



Characteristic	ARM Cortex-A8	Intel Nehalem
L1 cache organization	Split instruction and data caches	Split instruction and data caches
L1 cache size	32 KiB each for instructions/data	32 KiB each for instructions/data per core
L1 cache associativity	4-way (I), 4-way (D) set associative	4-way (I), 8-way (D) set associative
L1 replacement	Random	Approximated LRU
L1 block size	64 bytes	64 bytes
L1 write policy	Write-back, Write-allocate(?)	Write-back, No-write-allocate
L1 hit time (load-use)	1 clock cycle	4 clock cycles, pipelined
L2 cache organization	Unified (instruction and data)	Unified (instruction and data) per core
L2 cache size	128 KiB to 1 MiB	256 KiB (0.25 MiB)
L2 cache associativity	8-way set associative	8-way set associative
L2 replacement	Random(?)	Approximated LRU
L2 block size	64 bytes	64 bytes
L2 write policy	Write-back, Write-allocate (?)	Write-back, Write-allocate
L2 hit time	11 clock cycles	10 clock cycles
L3 cache organization	–	Unified (instruction and data)
L3 cache size	–	8 MiB, shared
L3 cache associativity	–	16-way set associative
L3 replacement	–	Approximated LRU
L3 block size	–	64 bytes
L3 write policy	–	Write-back, Write-allocate
L3 hit time	–	35 clock cycles

2-Level TLB Organization

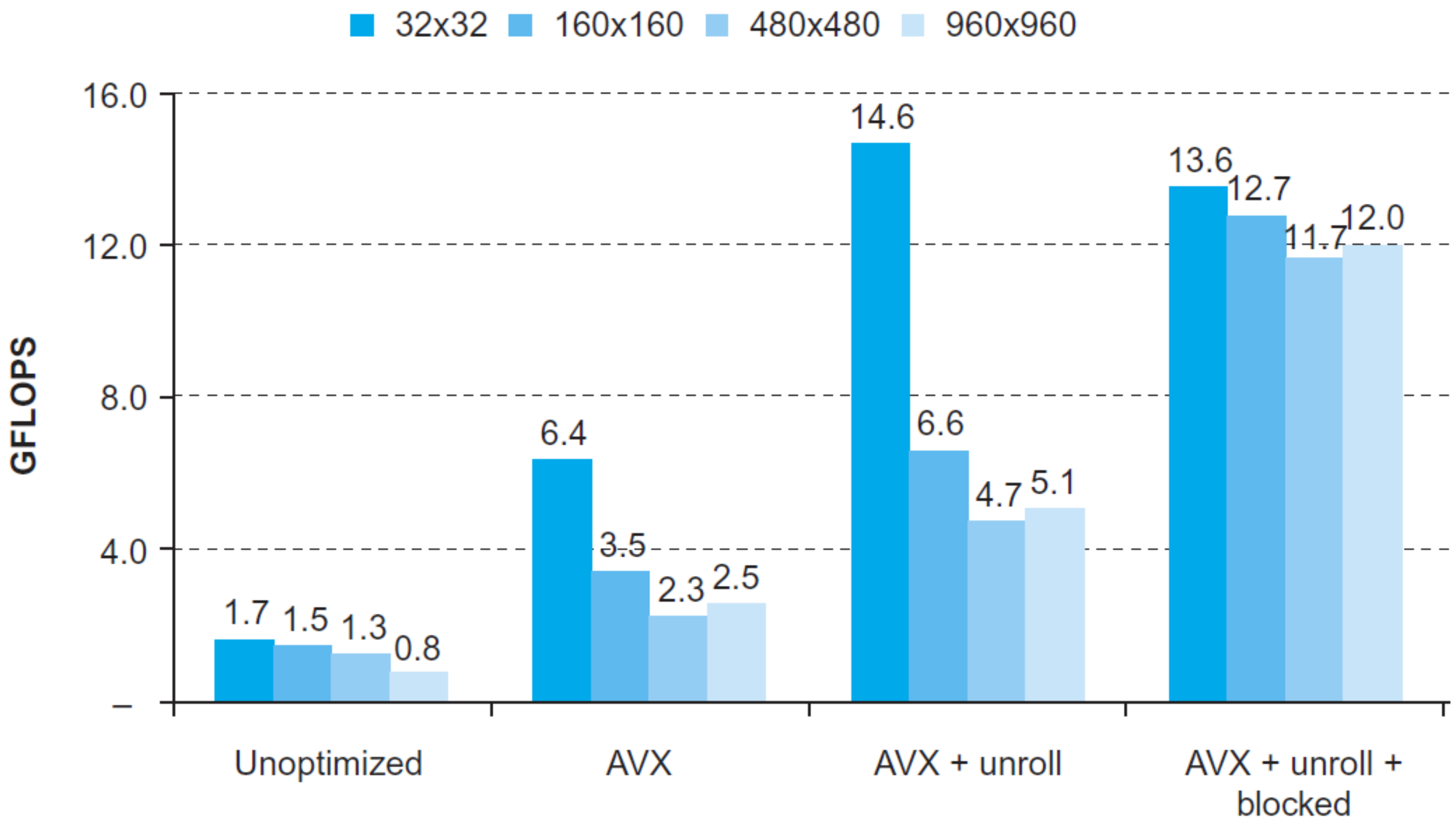
Characteristic	ARM Cortex-A8	Intel Core i7
Virtual address	32 bits	48 bits
Physical address	32 bits	44 bits
Page size	Variable: 4, 16, 64 KiB, 1, 16 MiB	Variable: 4 KiB, 2/4 MiB
TLB organization	1 TLB for instructions and 1 TLB for data Both TLBs are fully associative, with 32 entries, round robin replacement TLB misses handled in hardware	1 TLB for instructions and 1 TLB for data per core Both L1 TLBs are four-way set associative, LRU replacement L1 I-TLB has 128 entries for small pages, 7 per thread for large pages L1 D-TLB has 64 entries for small pages, 32 for large pages The L2 TLB is four-way set associative, LRU replacement The L2 TLB has 512 entries TLB misses handled in hardware

Supporting Multiple Issue

- Both have multi-banked caches that allow multiple accesses per cycle assuming no bank conflicts
- Core i7 cache optimizations
  - Return requested word first
  - Non-blocking cache
    - Hit under miss
    - Miss under miss
  - Data prefetching

DGEMM

- Combine cache blocking and subword parallelism



Pitfalls

- Byte vs. word addressing
  - Example: 32-byte direct-mapped cache, 4-byte blocks
    - Byte 36 maps to block 1
    - Word 36 maps to block 4
- Ignoring memory system effects when writing or generating code
  - Example: iterating over rows vs. columns of arrays
  - Large strides result in poor locality

Pitfalls

- In multiprocessor with shared L2 or L3 cache
  - Less associativity than cores results in conflict misses
  - More cores $\Rightarrow$ need to increase associativity
- Using AMAT to evaluate performance of out-of-order processors
  - Ignores effect of non-blocked accesses
  - Instead, evaluate performance by simulation

Pitfalls

- Extending address range using segments
  - *e.g.*, Intel 80286
  - But a segment is not always big enough
  - Makes address arithmetic complicated
- Implementing a VMM on an ISA not designed for virtualization
  - *e.g.*, non-privileged instructions accessing hardware resources
  - Either extend ISA, or require guest OS not to use problematic instructions

Concluding Remarks

- Fast memories are small, large memories are slow
  - We really want fast, large memories ☹
  - Caching gives this illusion ☺
- Principle of locality
  - Programs use a small part of their memory space frequently
- Memory hierarchy
  - L1 cache ↔ L2 cache ↔ … ↔ DRAM memory ↔ disk
- Memory system design is critical for multiprocessors