EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

FACULTY OF
SCIENCE
Communication Networks

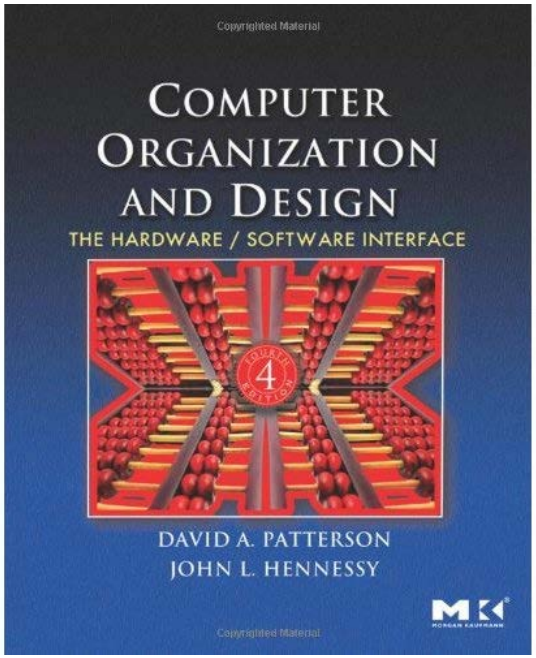
Informatik der Systeme – Chapter 7: Large and Fast: Exploiting Memory Hierarchy

Prof. Dr. Michael Menth

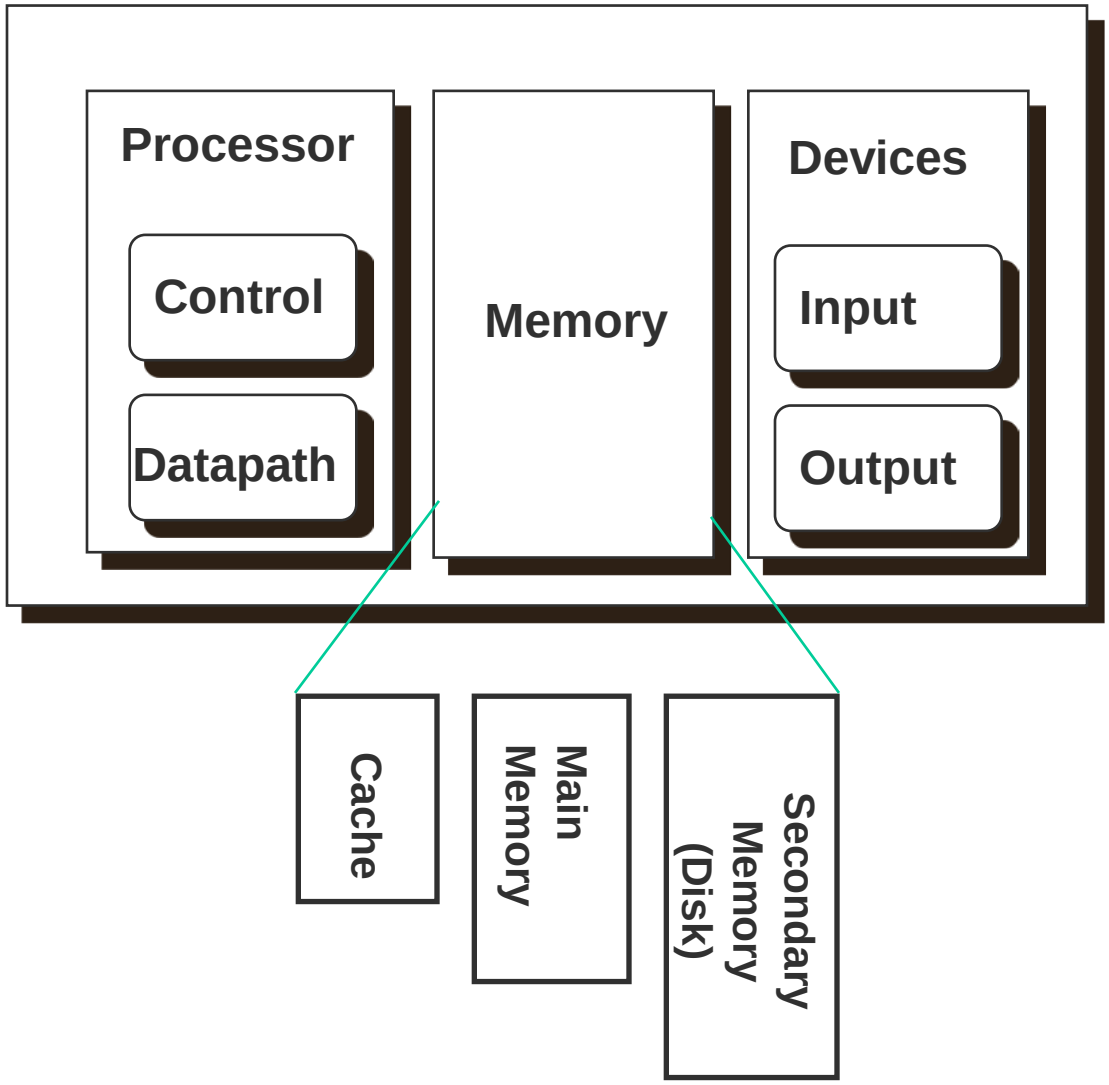
http://kn.inf.uni-tuebingen.de

Acknowledgements

- This set of slides is an adaptation of Prof. Mary Jane Irwin‘s lecture notes, http://www.cse.psu.edu/research/mdl/mji/
- Adapted from Patterson & Hennessy: “Computer Organization and Design”, 4th Edition, © 2008, MK
- German translation: Patterson & Hennessy: “Rechnerorganisation und Rechnerentwurf”



Review: Major Components of a Computer



Memory Technology

- RAM: random-access memory
- Static RAM (SRAM)
  - Access times: 0.5 ns – 2.5 ns, cost: $2000 – $5000 per GB
  - Static: content will last “forever” (as long as power is left on)
- Dynamic RAM (DRAM)
  - Access times: 50 ns – 70 ns, cost: $20 – $75 per GB
  - Dynamic: needs to be “refreshed” regularly (~ every 8 ms)
- Magnetic disk
  - Access times: 5 ms – 20 ms, cost: $0.20 – $2 per GB
- Ideal memory
  - Access time of SRAM
  - Capacity and cost/GB of disk

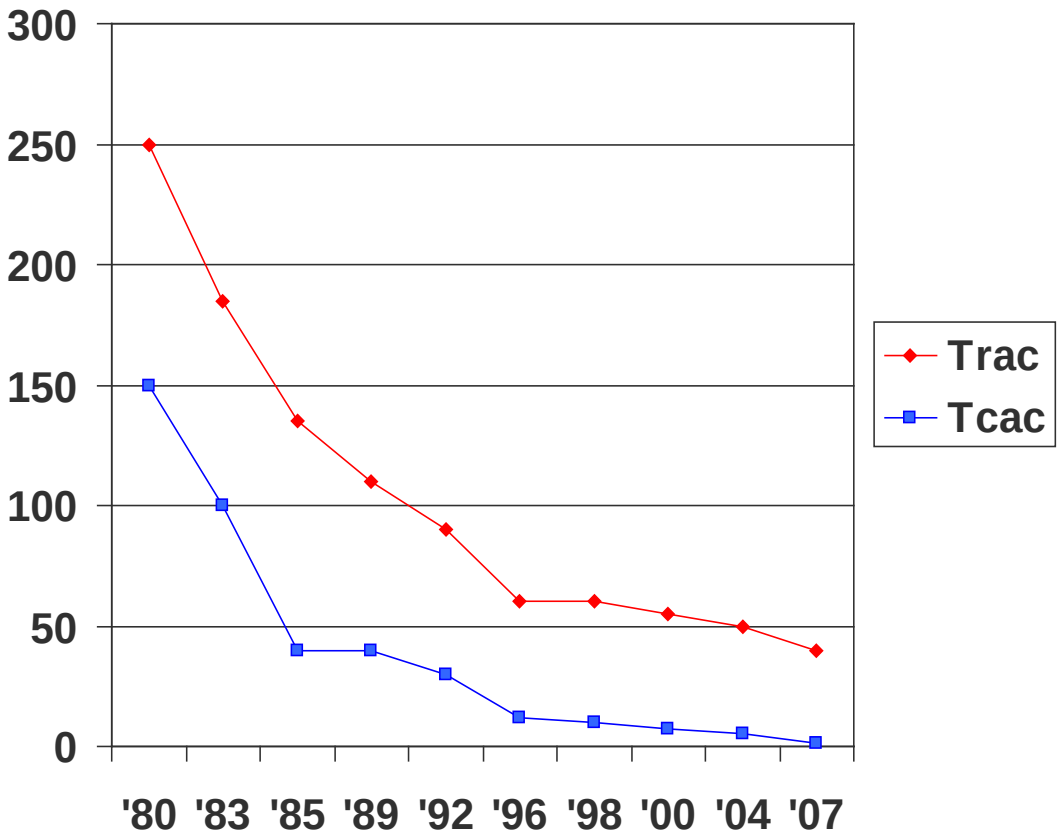
Advanced DRAM Organization

- Bits in a DRAM are organized as a rectangular array
  - RAS (Row Access Strobe): signal triggering the row decoder
    - to strobe: abtasten
  - CAS (Column Access Strobe): signal triggering the column selector
  - DRAM accesses an entire row
  - Burst mode: supply successive words from a row with reduced latency
- Double data rate (DDR) DRAM
  - Transfer on rising and falling clock edges

DRAM Generations

- t_{RAC}: time for random access of words
- t_{CAC}: time for column access of words (more efficient)

Year	Capacity	$/GB
1980	64Kbit	$1500000
1983	256Kbit	$500000
1985	1Mbit	$200000
1989	4Mbit	$50000
1992	16Mbit	$15000
1996	64Mbit	$10000
1998	128Mbit	$4000
2000	256Mbit	$1000
2004	512Mbit	$250
2007	1Gbit	$50



Principle of Locality

- Programs access a small proportion of their address space at any time
- Temporal locality
  - Items accessed recently are likely to be accessed again soon
  - E.g., instructions in a loop, induction variables
- Spatial locality
  - Items near those accessed recently are likely to be accessed soon
  - E.g., sequential instruction access, array data

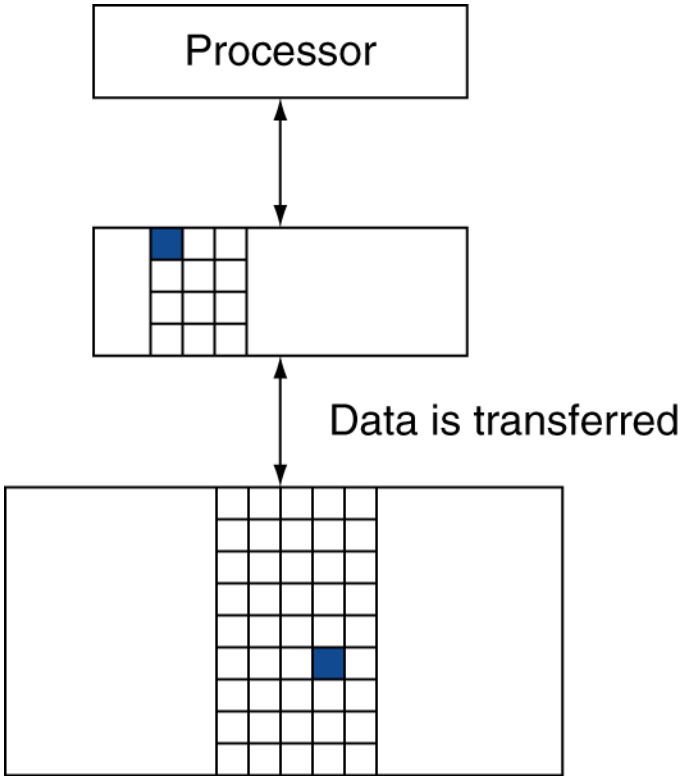
Memory Hierarchy

- Memory hierarchy
  - Disk
    - Lowest level
  - Main memory
    - DRAM
  - Cache memory
    - SRAM
    - Attached to CPU
    - Highest level

- Strategy
  - Store everything on disk
  - Copy recently accessed (and nearby) items from disk to main memory
  - Copy more recently accessed (and nearby) items from main memory to cache
  - Efficient due to temporal and spatial locality

Memory Hierarchy Levels

- Block (aka line): unit of copying
  - May be multiple words
- If accessed data is present in upper level
  - Hit: access satisfied by upper level
  - Hit ratio: hits/accesses
- If accessed data is absent
  - Miss: block copied from lower level
  - Miss ratio: misses/accesses
    = 1 – hit ratio
  - Then accessed data supplied from upper level
  - Miss penalty: time needed to load data from lower level

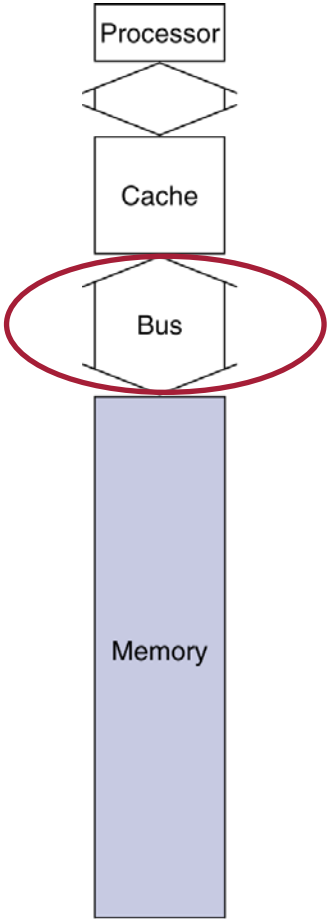


Actions on Cache Hit and Cache Miss

- On cache hit, CPU proceeds normally
- On cache miss
  - Stall the CPU pipeline
  - Fetch block from next level of hierarchy
  - Instruction cache miss
    - Restart instruction fetch
  - Data cache miss
    - Complete data access

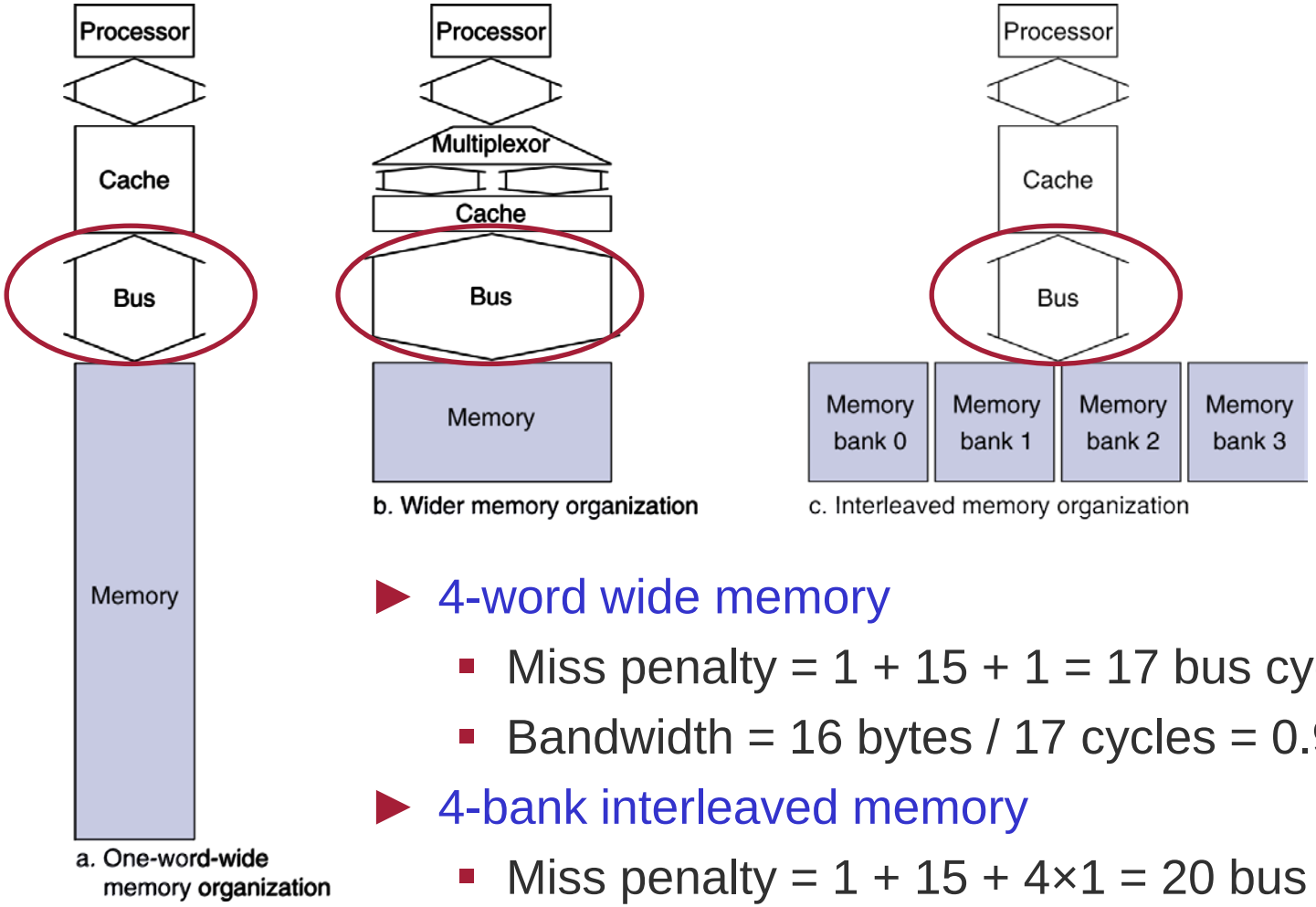
Data Transfer: Main Memory → Cache

- Use DRAMs for main memory
  - Fixed width (e.g., 1 word)
  - Connected by fixed-width clocked bus
    - Bus clock is typically slower than CPU clock
- Example cache block read
  - 1 bus cycle for address transfer
  - 15 bus cycles per DRAM access
  - 1 bus cycle per data transfer
- For 4-word block, 1-word-wide DRAM
  - Miss penalty = 1 + 4×15 + 4×1 = 65 bus cycles
  - Bandwidth = 16 bytes / 65 cycles ≈ 0.25 bytes/cycle



a. One-word-wide memory organization

Increasing Memory Bandwidth



- 4-word wide memory
  - Miss penalty = 1 + 15 + 1 = 17 bus cycles
  - Bandwidth = 16 bytes / 17 cycles = 0.94 bytes/cycle
- 4-bank interleaved memory
  - Miss penalty = 1 + 15 + 4×1 = 20 bus cycles
  - Bandwidth = 16 bytes / 20 cycles = 0.8 bytes/cycle

Handling Write-Hits

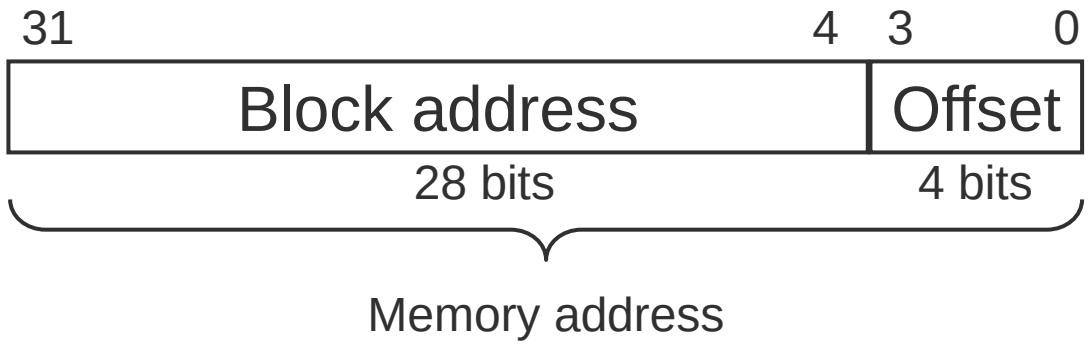
- On data-write hit, could just update the block in cache
  - But then cache and memory would be inconsistent
- Write-through strategy
  - Updates block in cache and memory
  - Takes longer, example:
    - Base CPI = 1
    - 10% of instructions are stores
    - Write to memory takes 100 cycles
    - Effective CPI = 1 + 0.1×100 = 11
  - Solution: write buffer
    - Holds data waiting to be written to memory
    - CPU continues immediately
    - Only stalls on write if write buffer is already full
- Write-back strategy
  - Updates only block in cache
    - Keeps track of whether blocks are „dirty“
    - Writes dirty block back to memory when it is replaced by another block
    - Can use a write buffer to allow replacing block to be read first
  - More state needed
  - More complex on replacement

Handling Write-Misses

- Strategies in combination with write-through
  - Write-allocate: fetch the block and write into cache
  - No write-allocate: don't fetch the block
    - Write only into memory
    - Useful when programs write a whole block before reading it (e.g., initialization)
  - Strategy may be specific to memory page
- Strategies in combination with write-back
  - Usually only write-allocate
  - Takes longer than with write-through
    - Write-back needed for replaced block before overwriting cache memory

Memory Blocks

- Block
  - 2^m bytes
  - Offset: indicates position of a specific byte within block
- Block address
  - Memory subdivided into 2^{32-m} blocks
  - Block address indicates position of block in memory



- Cache stores recently referenced memory blocks

Cache Structure

- Cache size
  - 2^n blocks
- Index
  - Block number in cache
  - $0 \leq \text{index} < 2^n$
- Valid bit
  - 0: data in cache not valid
  - 1: data in cache valid
  - Initially 0
- Data
  - Block cached from main memory

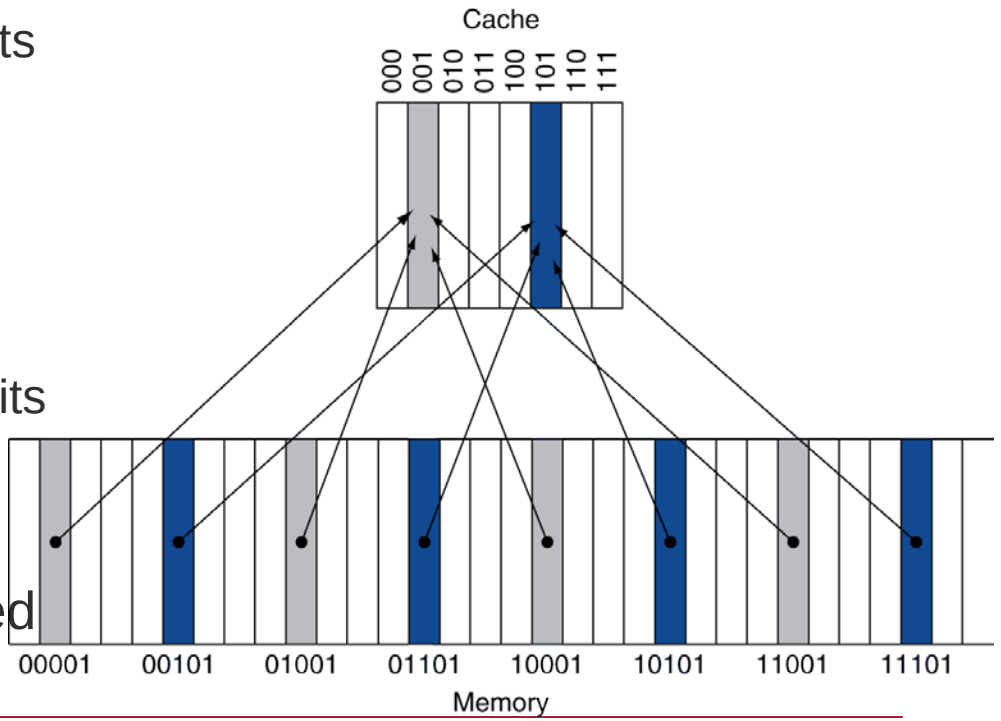
Index	Valid bit	Data
000		
…		
111		

- Challenge: mapping between
  - Block number in cache
  - Block address in memory

Direct-Mapped Cache

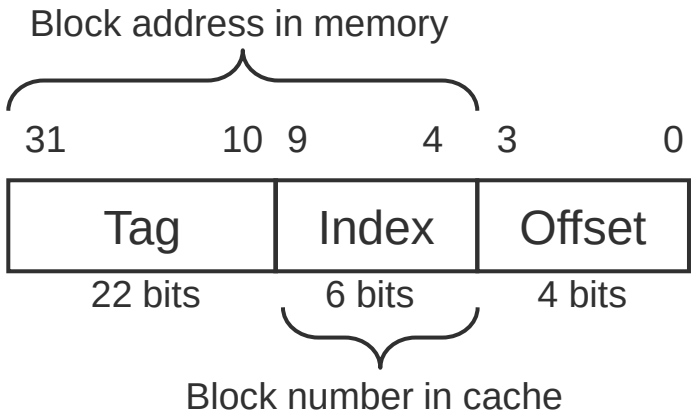
- How to determine position of memory block in cache?
  - Block with block address A
  - Cache size $S=2^n$
  - Block stored at index
    i = A modulo S
    - Low-order block address bits
- How to find out memory block address of block at index i?
  - Store tag in cache
    - Tag = $\lfloor A / S \rfloor$
    - High-order block address bits
  - Memory block address
    A = tag:index
- Multiple memory blocks mapped to same position in cache

Index	Valid bit	Tag	Data
000			
…			
111			



Direct-Mapped Cache: A Concrete Example

- Cache structure
  - Address width 32 bits
  - Cache with $1024=2^{10}$ bytes
    - Subdivided into blocks of $16=2^4$ bytes
      - Offset is 4 bits
    - $\Rightarrow$ cache has $1024/16=64=2^6$ blocks
      - Index is 6 bits
    - $\Rightarrow$ Tag is 32-4-6=22 bits



- Data retrieval from cache
  - From what block number (index) and at which offset can memory address 1200 be retrieved?
  - Block address = $\lfloor 1200/16 \rfloor = 75$
  - Index = 75 modulo 64 = 11
  - Offset = 1200 modulo 16 = 0

Example: Direct-Mapped Cache

- Block size: 1 word (4 bytes)
- Cache size: 8 blocks
- Initial state

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

Example: Direct-Mapped Cache

Word addr	Binary addr	Hit/miss	Cache block
22	10 110	Miss	110

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	**Y**	**10**	**Mem[10110]**
111	N		

Example: Direct-Mapped Cache

Word addr	Binary addr	Hit/miss	Cache block
26	11 010	Miss	010

Index	V	Tag	Data
000	N		
001	N		
010	**Y**	**11**	**Mem[11010]**
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Example: Direct-Mapped Cache

Word addr	Binary addr	Hit/miss	Cache block
22	10 110	Hit	110
26	11 010	Hit	010

Index	V	Tag	Data
000	N		
001	N		
010	Y	11	Mem[11010]
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Example: Direct-Mapped Cache

Word addr	Binary addr	Hit/miss	Cache block
16	10 000	Miss	000
3	00 011	Miss	011
16	10 000	Hit	000

Index	V	Tag	Data
000	**Y**	**10**	**Mem[10000]**
001	N		
010	Y	11	Mem[11010]
011	**Y**	**00**	**Mem[00011]**
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

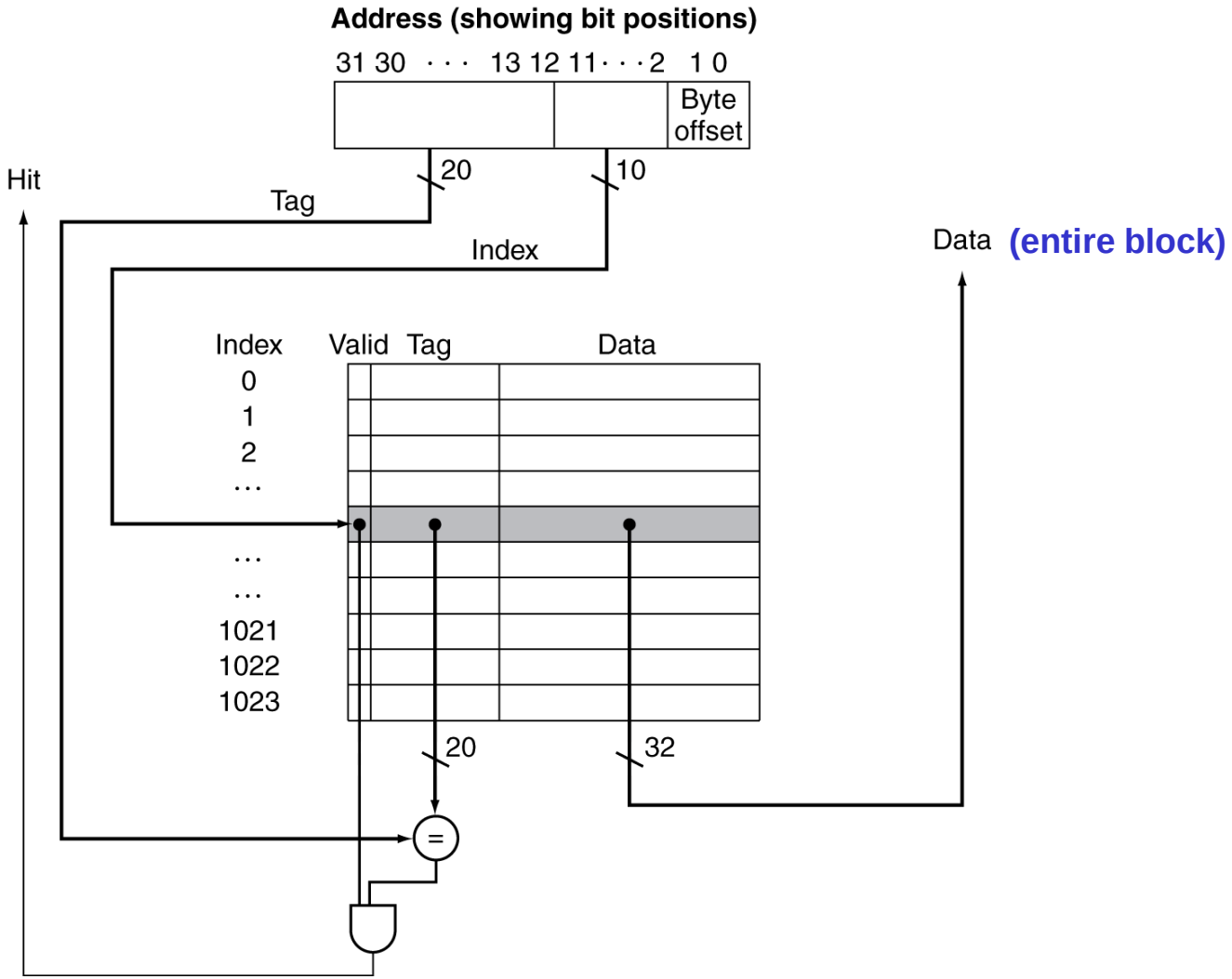
Example: Direct-Mapped Cache

Word addr	Binary addr	Hit/miss	Cache block
18	10 010	Miss	010

Existing entry (MEM[11010]) is replaced

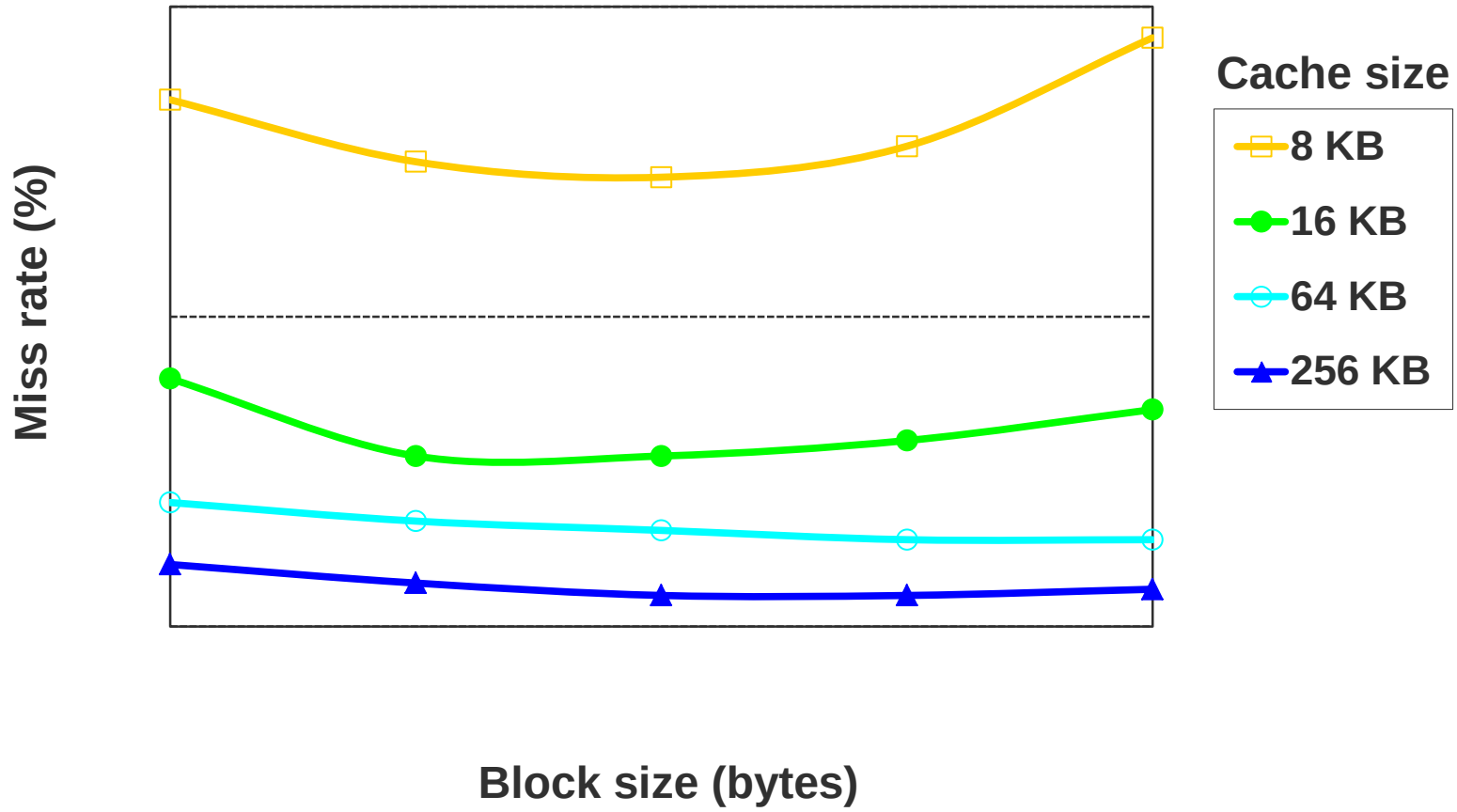
Index	V	Tag	Data
000	Y	10	Mem[10000]
001	N		
010	**Y**	**10**	**Mem[10010]**
011	Y	00	Mem[00011]
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Direct-Mapped Cache Organization



Direct-Mapped Cache – Performance Tradeoffs

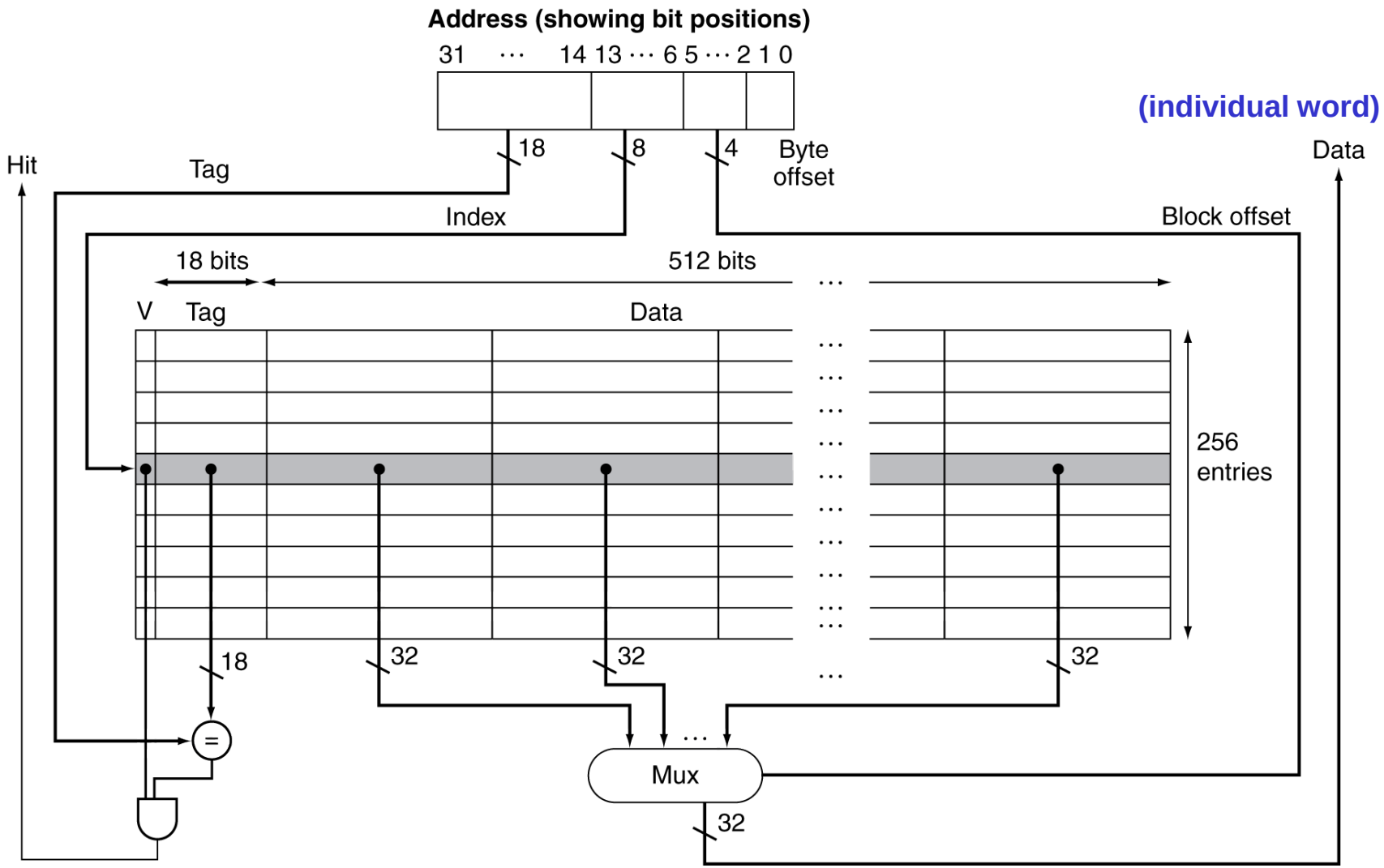
- Larger cache ⇒ lower miss rate
- Optimum block size exists



Example: Intrinsity FastMATH

- Embedded MIPS processor
  - 12-stage pipeline
  - Instruction and data access on each cycle
- Split cache: separate I-cache and D-cache
  - Each 16KB: 256 blocks × 16 words/block
  - D-cache: write-through or write-back
- SPEC2000 miss rates
  - I-cache: 0.4%
  - D-cache: 11.4%
  - Weighted average: 3.2%

Example: Intrinsity FastMATH

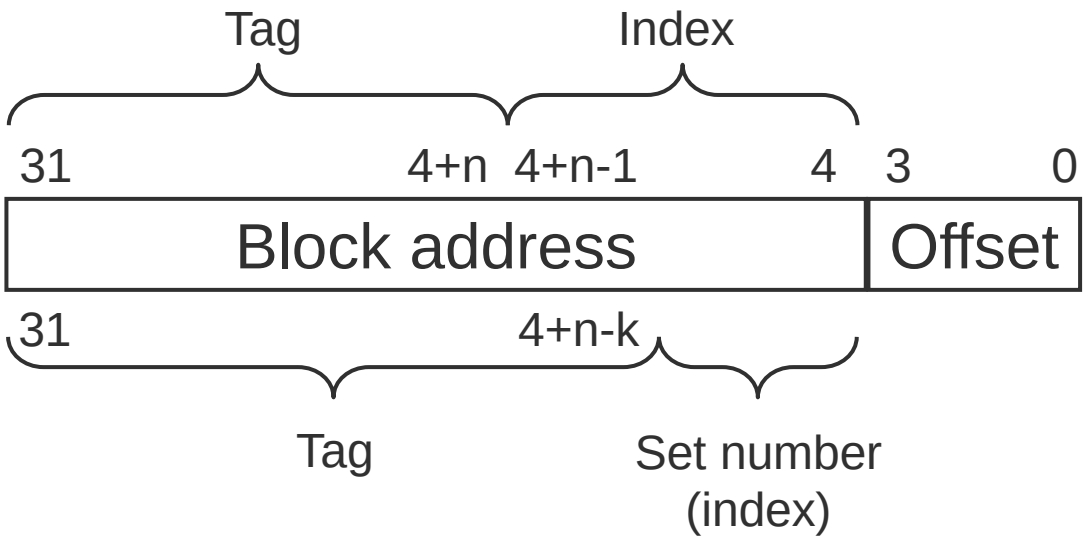


Set Associative Caches

- Fully associative cache (of size $S=2^n$ blocks à 2^m bytes)
  - Allows a given block to go in any cache entry
  - Retrieval: search all entries of the cache at once
    - Requires 2^n comparators (one per entry)
    - Expensive hardware
- 2^k-way set associative cache
  - Cache of size $S=2^n$ partitioned into 2^{n-k} sets with 2^k entries each
  - Block address A mapped to set number (index) i = A modulo 2^{n-k}
  - Block can be stored in that set
  - Retrieval: search all entries of set i at once
    - Requires 2^k comparators
    - Hardware less expensive than for fully associative cache ($2^k < 2^n$)
- Higher associativity reduces miss rate
  - Increases complexity, cost, and access time

Tags in Set Associative Caches

- Direct-mapped cache of size $S=2^n$



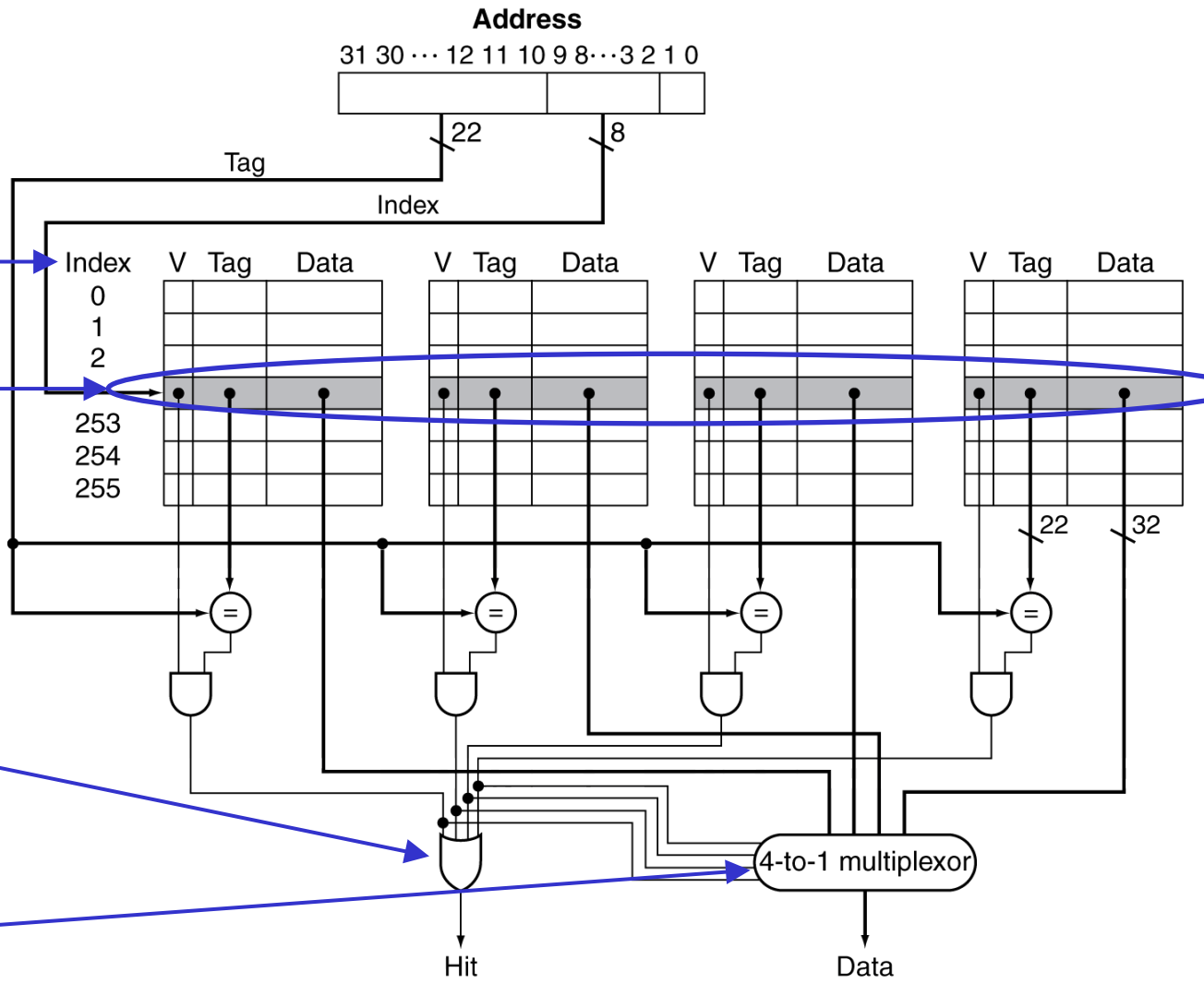
- 2^k-way set associative cache
  - Size $S=2^n$
  - Subdivided into 2^{n-k} sets
  - Requires larger tags than direct-mapped cache of same size

Organization of Set Associative Caches

Set number

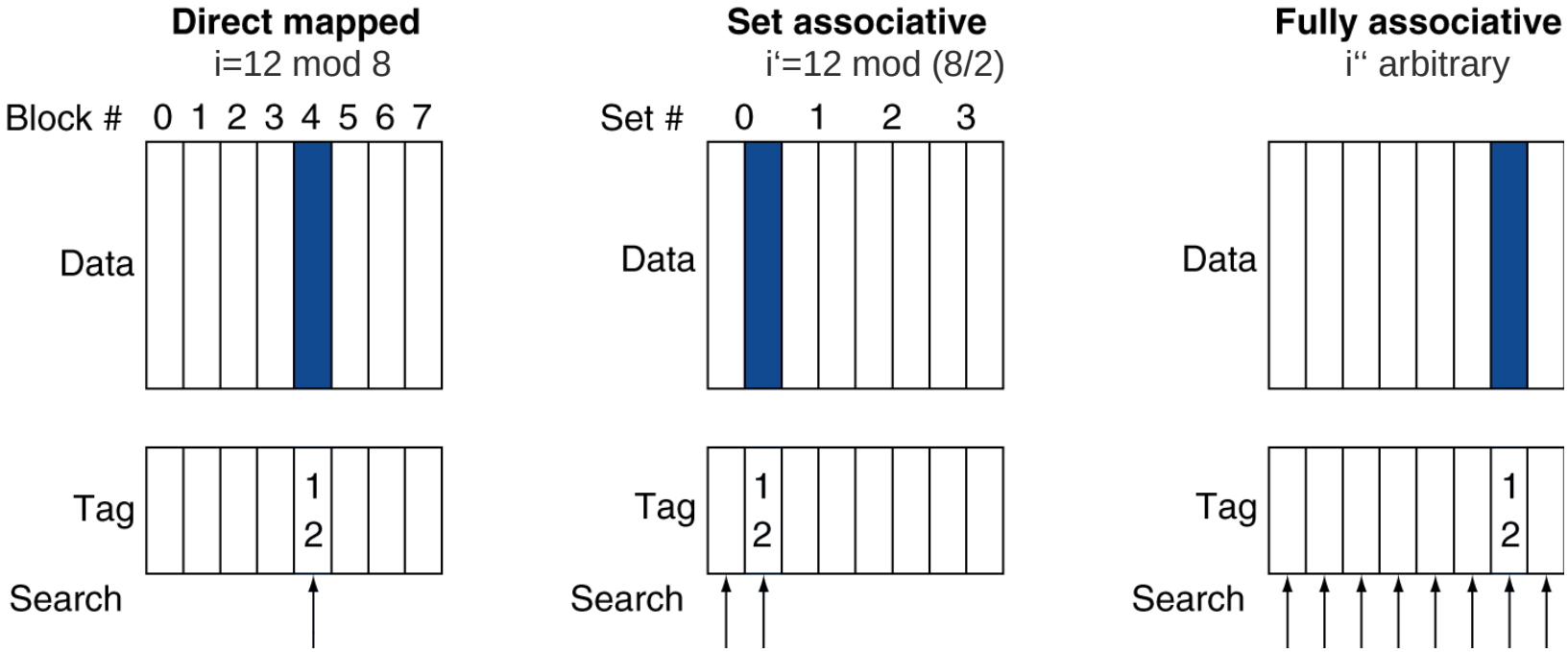
All entries of a set

- 2^k parallel direct-mapped caches of size 2^{n-k}
  - OR to calculate hit
  - Multiplexer to retrieve result



Comparison of Cache Variants (1)

- Data with address 12 stored in cache of size 8 (byte=block size)
  - Direct-mapped
  - 2-way set associative
  - Fully associative
- Different search overhead needed for retrieval



Comparison of Cache Variants (2)

- For a cache with 8 entries
- 1-, 2-, 4, and 8-way set associative caches provide a spectrum of caches

One-way set associative (direct mapped)

Block	Tag	Data
0		
1		
2		
3		
4		
5		
6		
7		

Two-way set associative

Set	Tag	Data	Tag	Data
0				
1				
2				
3				

Four-way set associative

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

Eight-way set associative (fully associative)

Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data

Comparison of Cache Variants (3)

- Compare different variants of caches containing 4 blocks
  - Direct-mapped
  - 2-way set associative
  - Fully associative
- Block access sequence: 0, 8, 0, 6, 8

Block address	Cache block
0	(0 % 4) = 0
6	(6 % 4) = 2
8	(8 % 4) = 0

- Direct-mapped cache

Time

Block address	Cache index	Hit/miss	Cache content after access			
			0	1	2	3
0	0	miss	**Mem[0]**			
8	0	miss	**Mem[8]**			
0	0	miss	**Mem[0]**			
6	2	miss	Mem[0]		**Mem[6]**	
8	0	miss	**Mem[8]**		Mem[6]	

  - Five misses for five accesses

Comparison of Cache Variants (4)

- 2-way set associative cache

Time

Block address	Cache index	Hit/miss	Cache content after access			
			Set 0		Set 1	
0	0	miss	**Mem[0]**			
8	0	miss	Mem[0]	**Mem[8]**		
0	0	hit	**Mem[0]**	Mem[8]		
6	0	miss	Mem[0]	**Mem[6]**		
8	0	miss	**Mem[8]**	Mem[6]		

Block address	Cache block
0	(0 % 2) = 0
6	(6 % 2) = 0
8	(8 % 2) = 0

  - Four misses for five accesses

- Fully associative cache

Time

Block address		Hit/miss	Cache content after access			
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[0]	**Mem[8]**	Mem[6]	

  - Three misses for five accesses

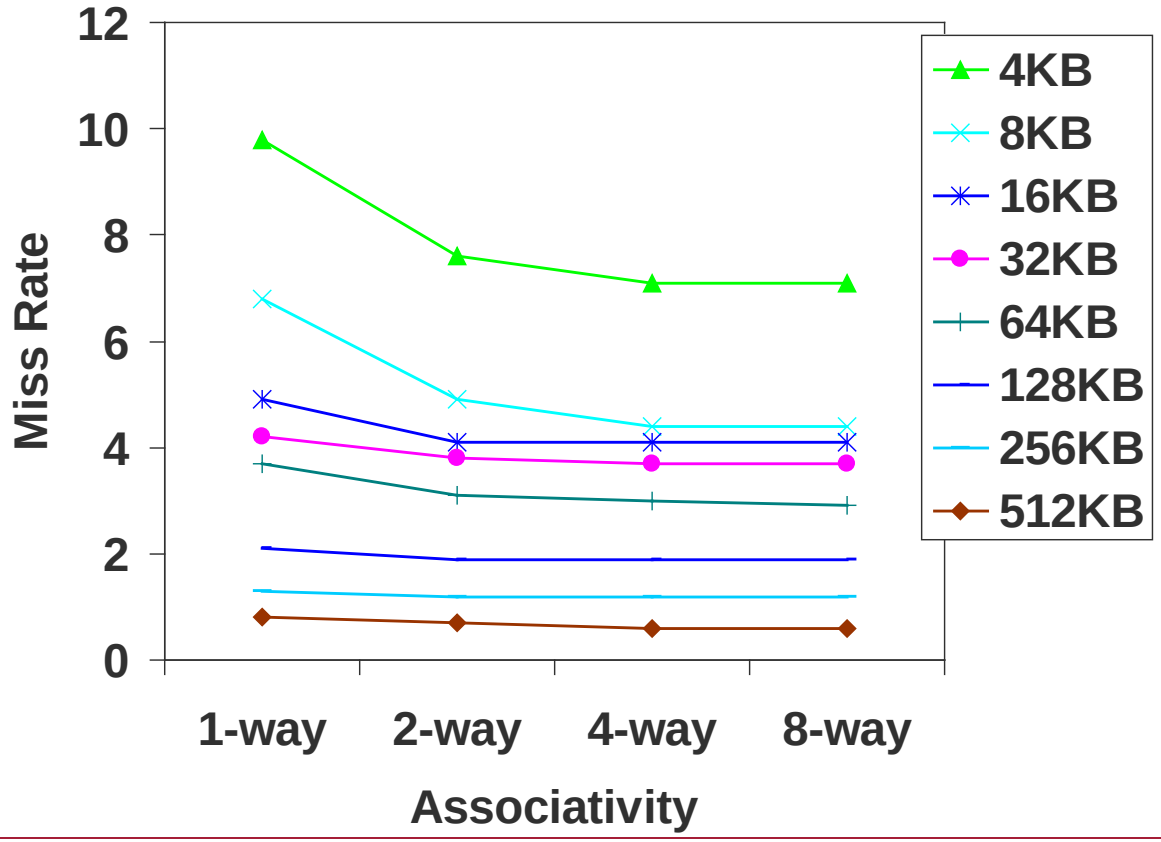
Gain of Associativity (1)

- Simulation of a system with 64 KB D-cache, 16-word blocks, SPEC2000
- Increased associativity decreases miss rate
  - 1-way: 10.3%
  - 2-way: 8.6%
  - 4-way: 8.3%
  - 8-way: 8.1%

  But with diminishing returns!

Gain of Associativity (2)

- Different setting, study also includes cache size
- Largest gains are
  - for small caches
  - in going from direct-mapped to 2-way



Data from Hennessy & Patterson, *Computer Architecture*, 2003

Replacement Policy for Caches

- Direct-mapped: no choice
- Set associative
  - Prefer non-valid entry if there is one
  - Otherwise, choose among entries in the set according to a replacement policy
- Replacement policies
  - Least-recently used (LRU)
    - Choose the one unused for the longest time
    - Simple for 2-way, manageable for 4-way, too hard beyond that
  - Least-frequently used (LFU)
    - Choose the least-frequently used
  - Random
    - Gives approximately the same performance as LRU for high associativity
  - First-in-first-out (FIFO)
    - Choose the one longest in cache
    - Not so appropriate for caches

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 Tradeoffs

Design change	Effect on miss rate	Negative performance effect
Increase cache size	Decreases capacity misses	May increase access time
Increase associativity	Decreases conflict misses	May increase access time
Increase block size	Decrease compulsory misses due to spatial locality; Increases capacity misses	Increases miss penalty due to higher block transfer overhead; For very large block size, may increase miss rate due to pollution (invalid data).

Average Memory Access Time (AMAT)

- AMAT = Hit time + Miss rate × Miss penalty
  - Hit time is also important for performance

- Example
  - CPU with 1 ns clock
  - Hit time = 1 cycle
  - Miss penalty = 20 cycles
  - I-cache miss rate = 5%
  - AMAT = 1 + 0.05 × 20 = 2 ns
    - 2 cycles per instruction

Impact of Cache Performance on Actual CPI

- Components of CPU time
  - Program execution cycles
    - Includes cache hit time
  - Memory stall cycles
    - Mainly from cache misses
    - With simplifying assumptions:

$$MemoryStallCycles = \frac{MemoryAccesses}{Program} \cdot MissRate \cdot MissPenalty$$

$$MemoryStallCycles = \frac{Instructions}{Program} \cdot \frac{Misses}{Instruction} \cdot MissPenalty$$

- Given (example)
  - I-cache miss rate = 2%
  - D-cache miss rate = 4%
  - Miss penalty = 100 cycles
  - Base CPI (ideal cache) = 2
  - Load & stores are 36% of instructions
- Miss cycles per instruction
  - I-cache: 0.02 × 100 = 2
  - D-cache: 0.36 × 0.04 × 100 = 1.44
- Actual CPI = 2 + 2 + 1.44 = 5.44
  - CPU with ideal cache is 5.44/2 =2.72 times faster

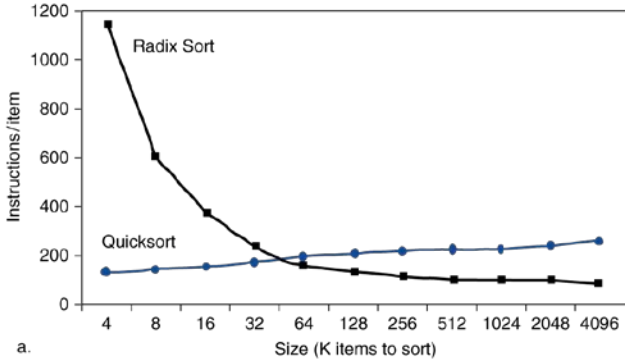
Performance Implications

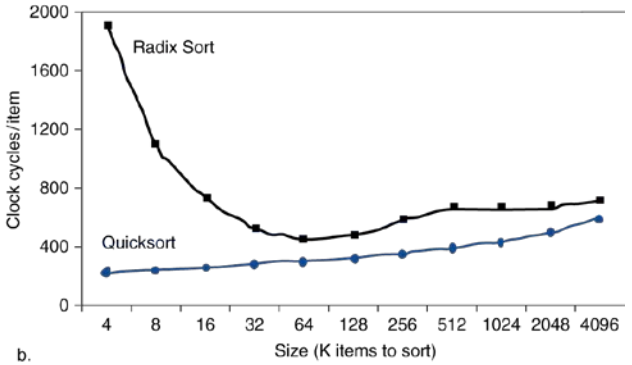
- When CPU performance increased
  - Miss penalty becomes more significant
- Decreasing base CPI
  - Greater proportion of time spent on memory stalls
- Increasing clock rate
  - Memory stalls account for more CPU cycles
- Can’t neglect cache behavior when evaluating system performance

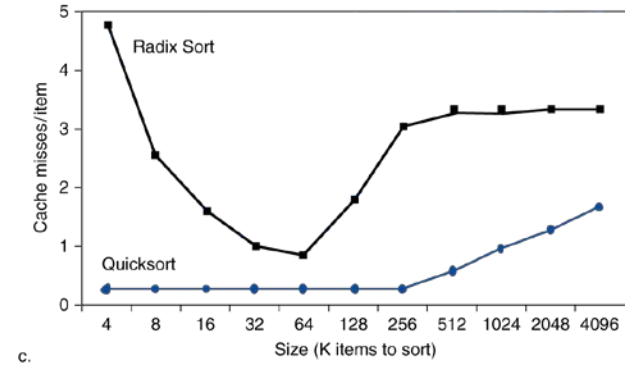
Interactions with Software

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

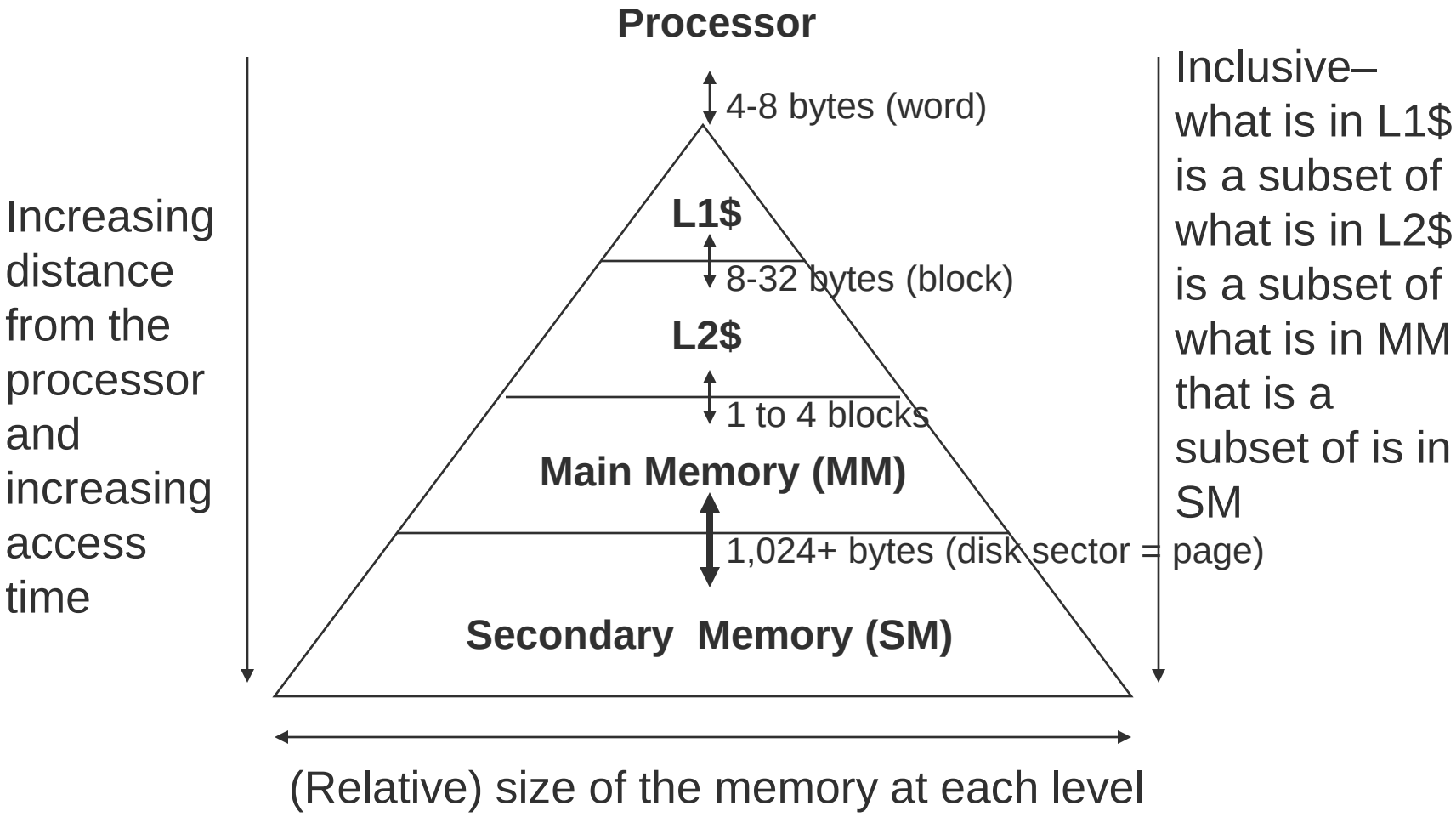
- Required clock cycles per item increase when number of cache misses per item increase



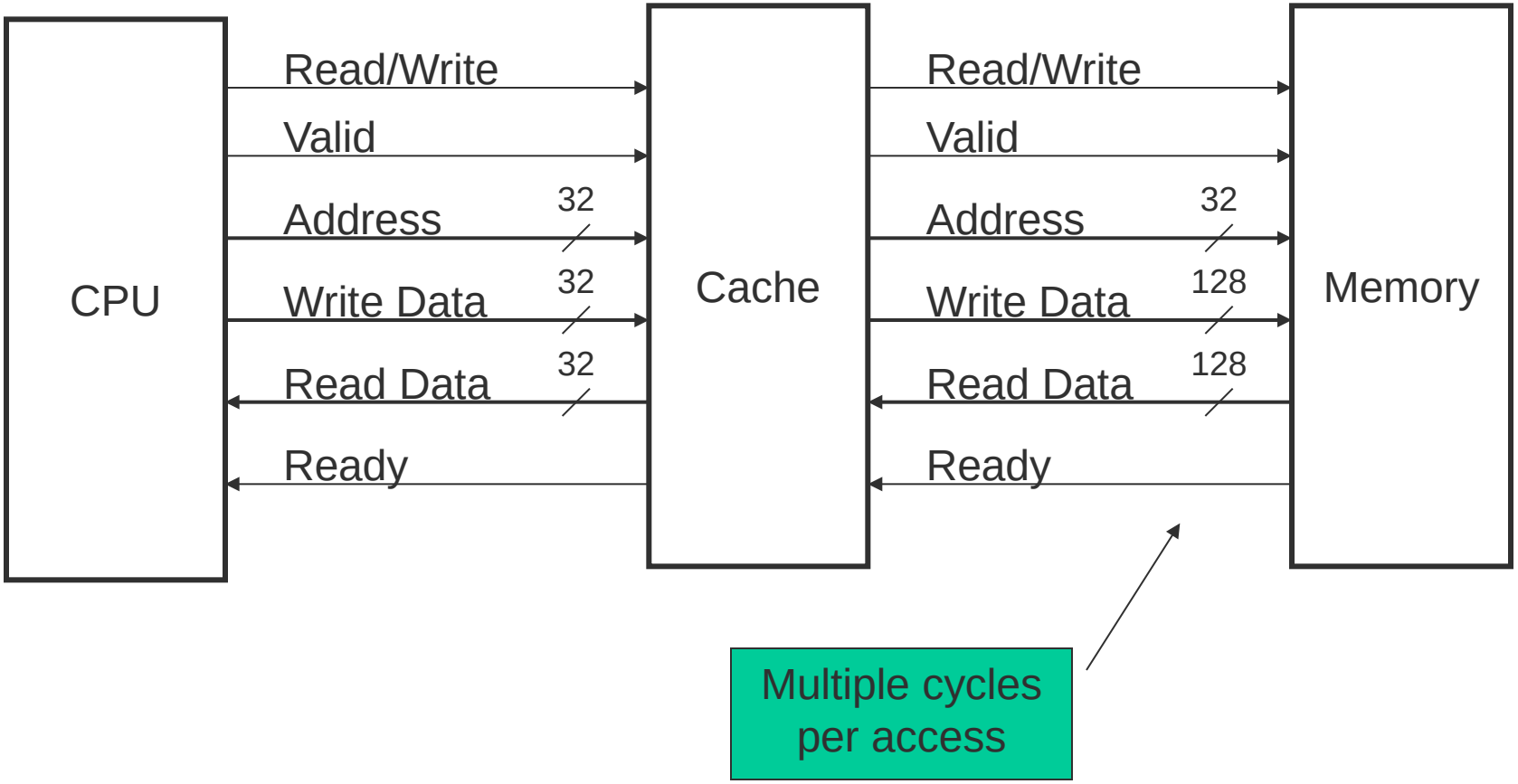




Characteristics of the Memory Hierarchy



Interface Signals



Multilevel Caches

- Primary (L-1) cache
  - Attached to CPU
  - Focus on minimal hit time
  - Small, but fast
  - L-1 cache in a multilevel cache hierarchy usually smaller than a single cache
- Level-2 (L-2) cache
  - Services misses from primary cache
  - Focus on low miss rate to avoid main memory access
  - Larger, slower, but still faster than main memory
  - Hit time has less overall impact
  - L-2 block size larger than L-1 block size
- 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 = 4 GHz
    - Clock cycle = 0.25 ns
  - Miss rate/instruction = 2%
  - Main memory access time = 100 ns
- With just primary cache
  - Miss penalty = 100 ns / (0.25 ns/cycle) = 400 cycles
- Effective CPI = 1 + 0.02 × 400 = 9

- Now add L-2 cache
- L1 characteristics
  - L2 access time = 5 ns
  - Miss penalty = 5 ns / (0.25 ns/cycle) = 20 cycles
  - Miss rate/instruction = 2% (from book)
- L2 characteristics
  - Main memory access time = 100 ns
  - Miss penalty = 100 ns / (0.25 ns/cycle) = 400 cycles
  - Miss rate/instruction = 0.5%
- Effective CPI = 1 + 0.02 × 20 + 0.005 × 400 = 3.4
- Performance ratio = 9/3.4 = 2.6
- With L1 miss rate/instruction = 4%
  - Effective CPI = 1 + 0.04x20+0.005x400=3.8
  - Performance ratio = 9/3.8=2.4

Process

- Definition
  - Instance of a computer program currently being executed
- Comprises
  - Virtual memory (see next section) containing the
    - Executable code
    - Process-specific data (input and output), a call stack, a heap
  - Operating system descriptors of resources
    - Allocated to the process, such as files or network connections
  - Security attributes
    - E.g., the process owner and the process' set of permissions
  - Processor context
    - E.g. the content of registers, physical memory addressing, etc.
  - Typically stored in memory or on disk when process not active
- States
  - Active, ready, waiting, new, terminated, … (Source: Wikipedia)

Mangagement of Processes

- Operating system (OS)
  - Controls processes
  - Controls resources and assigns them to processes

- Multitasking OS
  - Switches between processes (context switch) to give the appearance of many processes executing concurrently or simultaneously
  - Though in fact only one process can be executing at any one time on a single-core CPU

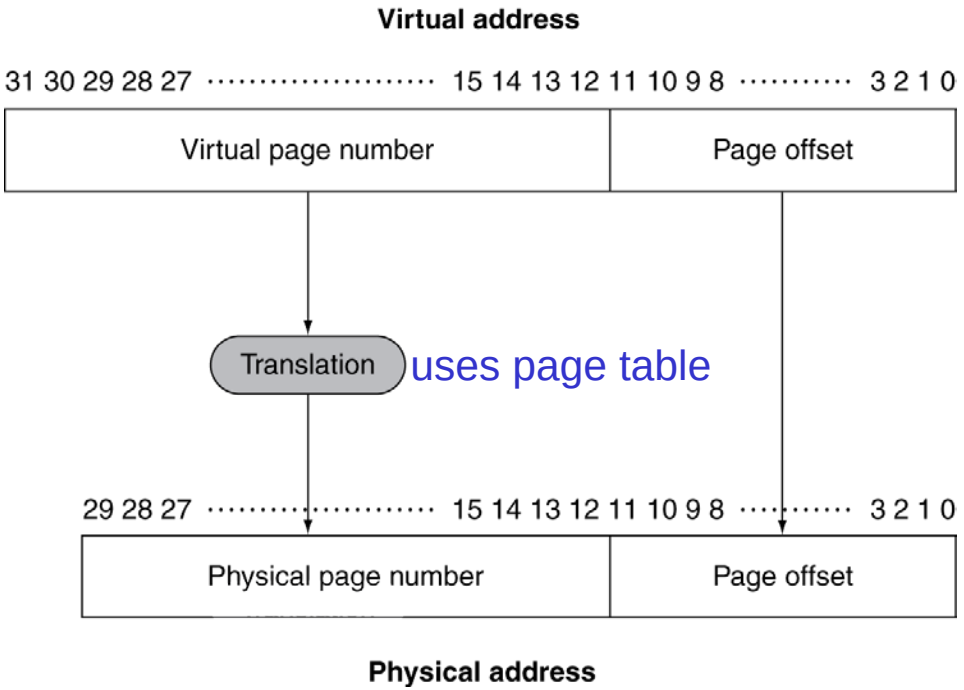
(Source: Wikipedia)

Virtual Memory

- Helps programs to share main memory
  - Each gets a private virtual address space holding its frequently used code and data
  - Protected from other programs
- Helps programs to address a memory that is larger than the main memory
  - Use main memory as a “cache” for secondary (disk) storage
  - Managed jointly by CPU hardware and the operating system (OS)
- CPU and OS translate virtual to physical addresses
  - Page
    - Virtual memory “block”
  - Page fault
    - Miss of a virtual memory page in main memory upon translation
  - 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

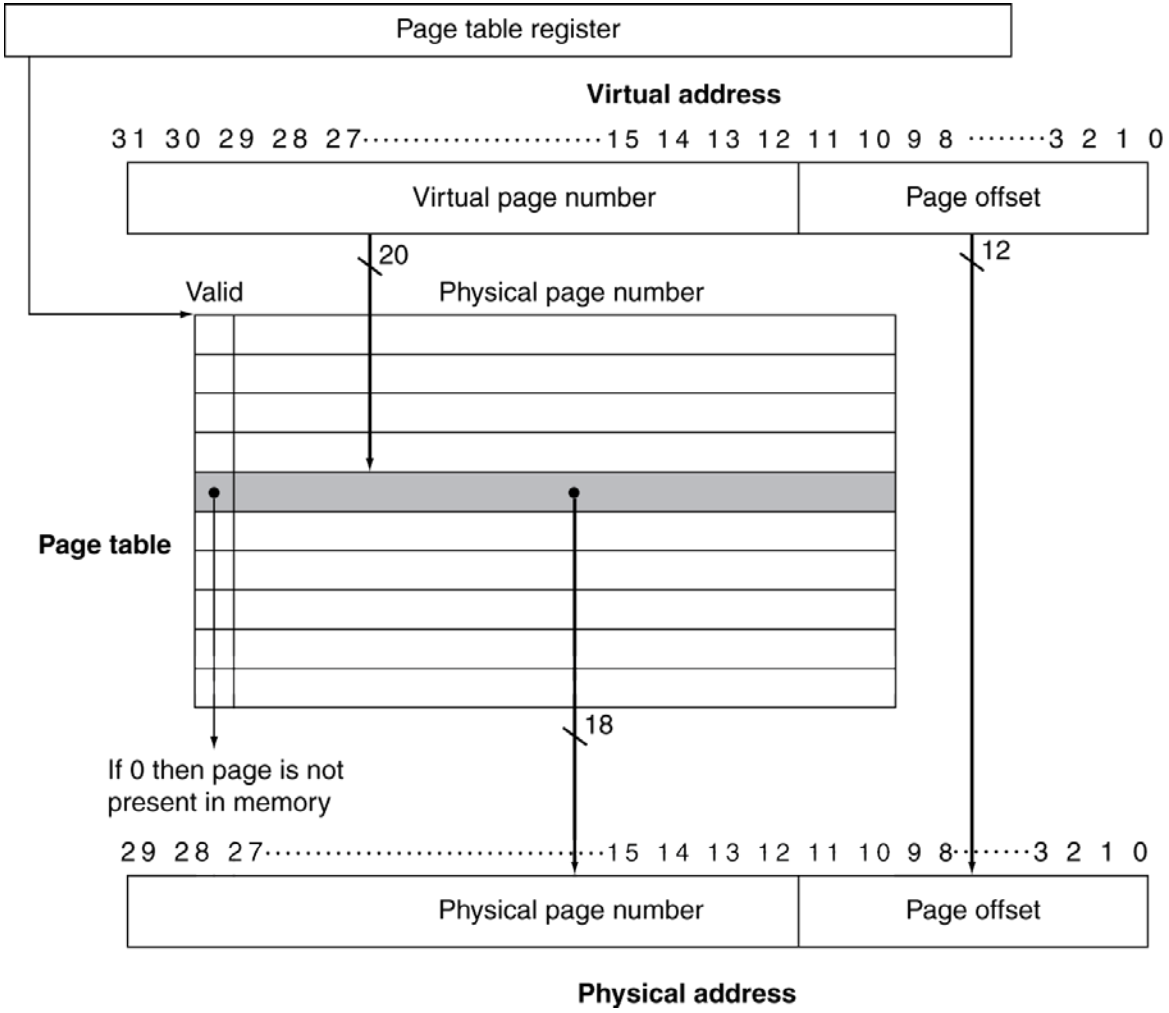
Virtual Address Translation (1)

- Virtual address consists of
  - Virtual page number
  - Page offset
    - Fixed-size pages (e.g., 4 KB)
- Virtual page number translated to physical page number



Virtual Address Translation (2)

- Page table used for virtual address translation

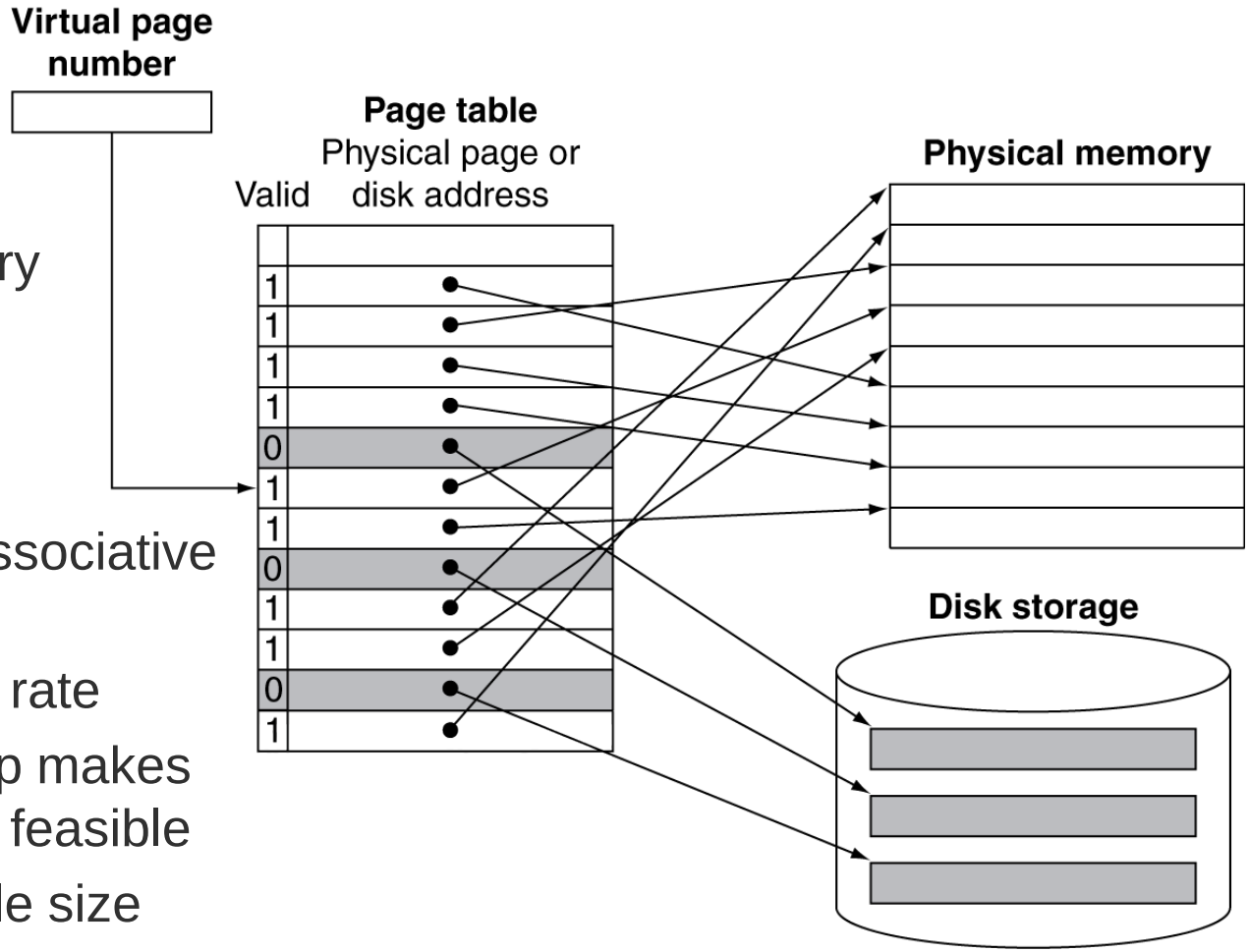


Page Table (1)

- Page table register
  - One page table per “process”
  - Page table register points to process-specific page table in OS memory
- Page table stores placement information of virtual pages
  - Objective: maps virtual page number to physical page number or disk address
  - Structure: array of page table entries (PTEs), indexed by virtual page number
  - Location: 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

Page Table (2)

- Page table maps virtual pages to pages in
  - Physical memory
  - And disk space
- Physical memory implements fully associative cache
  - Minimizes miss rate
  - Full table lookup makes implementation feasible
  - Large page table size



Comparison of Caches

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 lookup table makes full associativity feasible
  - Benefit in reduced miss rate

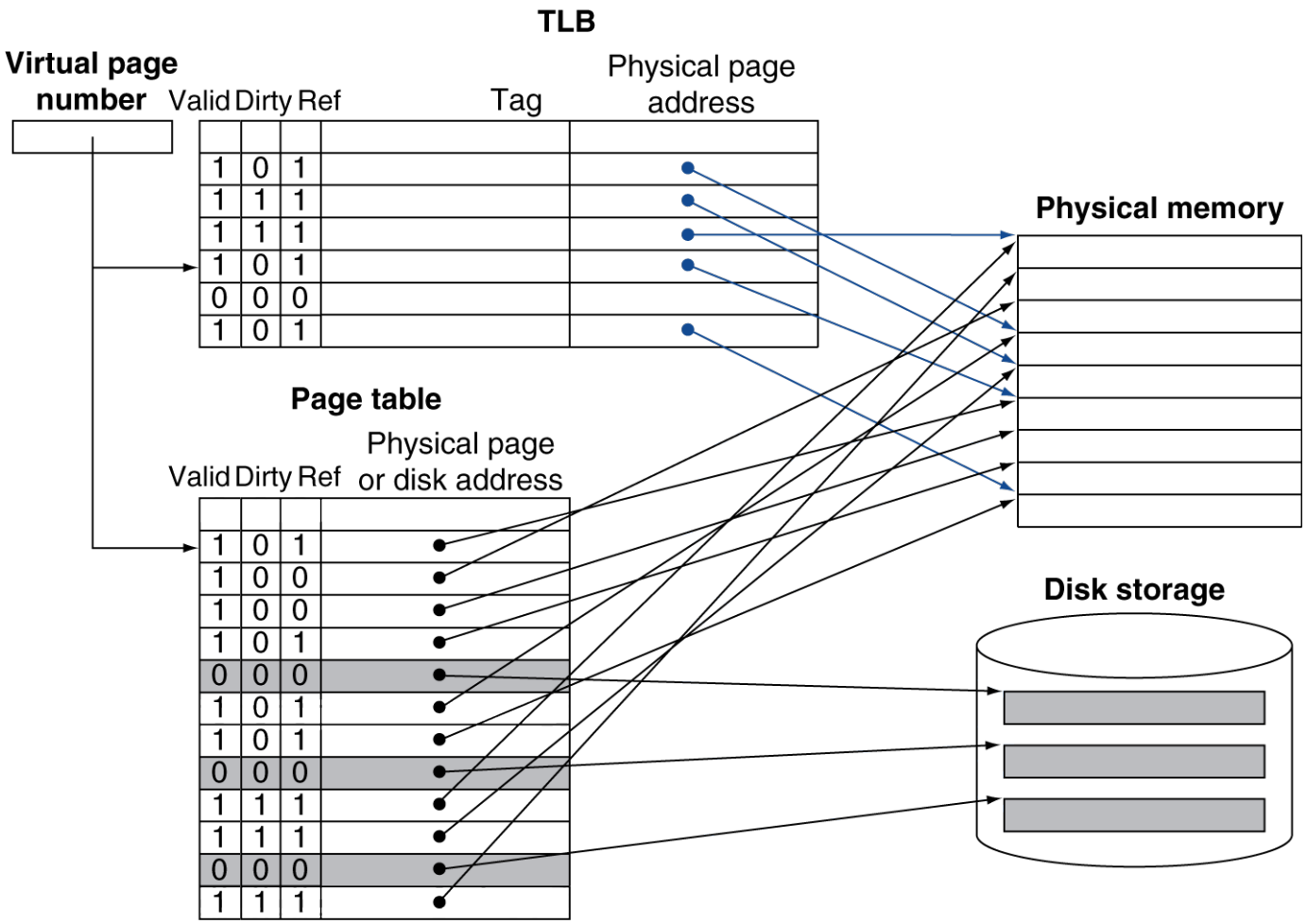
Replacement and Writes

- Replacement of pages in main memory
  - Least-recently used (LRU) replacement preferred to reduce page fault rate
  - LRU with hardware support
    - 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
- Handling writes
  - Use write-back strategy
    - Dirty bit in PTE set when page is written
    - Page updated on disk when replaced in memory
  - Write through is impractical
    - Disk writes take millions of cycles
    - Write entire block once, not individual words (too much penalty)

Translation Look-Aside Buffer (TLB)

- Increases translation speed
  - Virtual page number → physical page number
- Virtual addresses (without TLB) require extra memory reference
  - 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 example
    - 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

TLB Organization



Replacing a Page in Memory

- Operating system supports hierarchy by
  - Writing back data from cache to memory if dirty bit set
  - Flushing contents of page from cache
  - Writing back page content to disk if dirty bit set
  - Removing PTE from TLB
  - Modifying PTE in page table

Page Fault Handler

- Upon page fault
  - Referenced page not in memory
  - OS handles fetching the page and updating the page table
  - Then restart the faulting instruction

- Actions of the page fault handler
  - Use faulting virtual address to find PTE
  - Locate page on disk
  - Choose page to replace
    - If dirty, write to disk first (takes long)
  - Read page into memory (takes long) and update page table
  - Give control to OS so that other processes can run
  - Make process runnable again when page in memory
  - Restart from faulting instruction

TLB Miss Handler

- TLB miss
  - PTE not in TLB
  - TLB miss exception raised
  - TLB miss handler
    - Hardware or software
    - Copies PTE from memory to TLB
    - Then restarts instruction
    - If page not present, page fault will occur
- Two possibilities
  - Page present in memory
    - Valid bit in PTE on
    - Memory access can proceed
  - Page not present in memory
    - Valid bit in PTE off
    - Page fault, raise exception and fault page in

TLB Usage with Multiple Processes

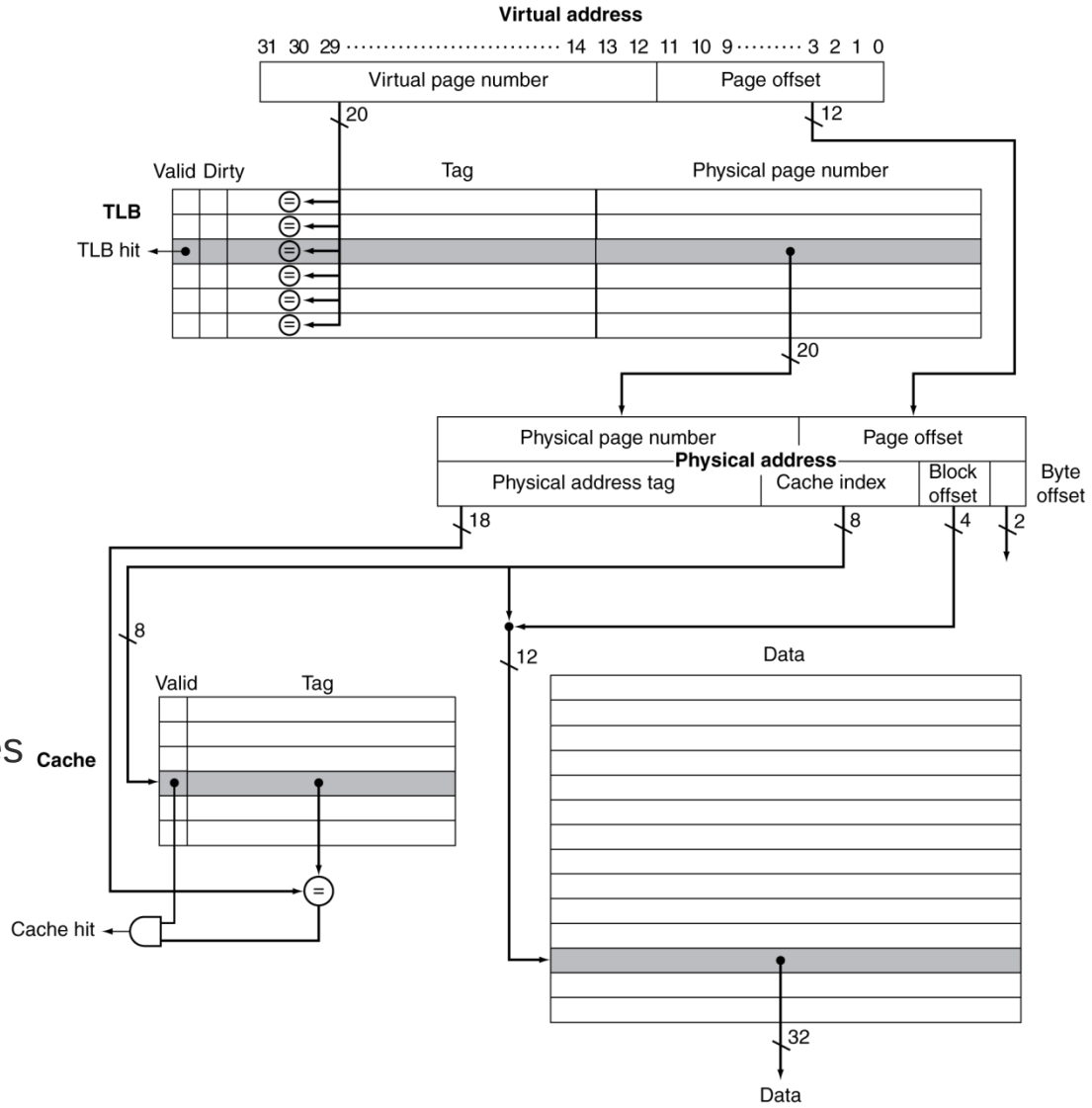
- Multiple processes have same virtual address space
  - Page table register points to page table of current process
  - Problem: single TLB – which PTE in TLB belongs to which process?

- Solution 1
  - Flush TLB upon context switch
    - TLB misses occur after context switch ☹

- Solution 2
  - Add process or task identifier to virtual address
    - Intrinsity FastMATH has 8 bit address space ID (ASID)
    - OS makes ASID available in register
    - PTEs of multiple processes can be differentiated in TLB
    - TLB flushing not needed upon context switch

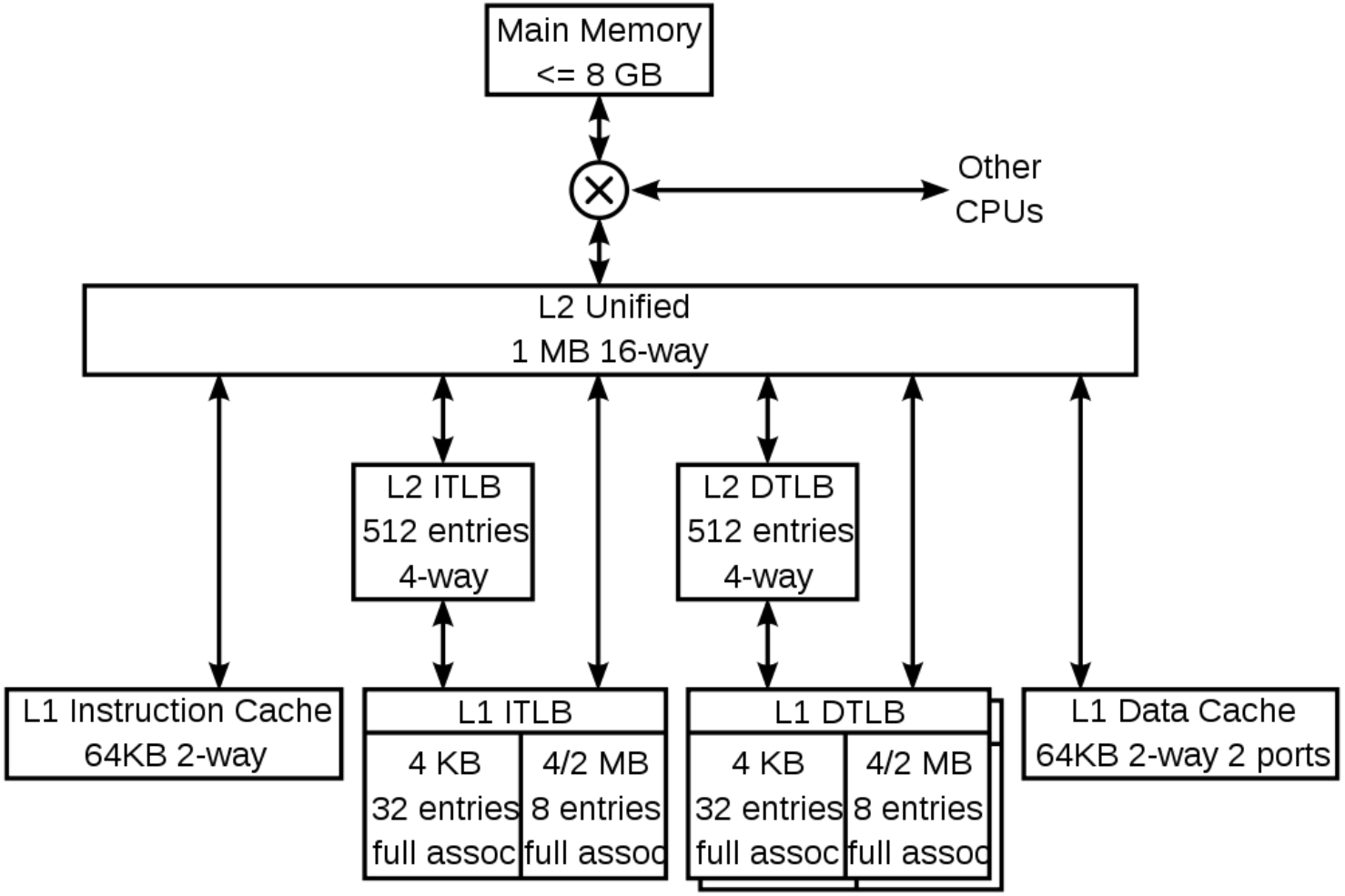
TLB and Cache Interaction

- Intrinsity FastMATH
  - Example: 4 KB pages
  - Memory access always involves
    - TLB lookup
    - Cache lookup
  - Cache holds physical addresses
    - Index and tag use physical address
- Alternative designs exist
  - Use virtual address for caches
  - Direct cache access possible
  - More difficult
    - Multiple processes
    - Shared data (has different virtual address, aliasing)



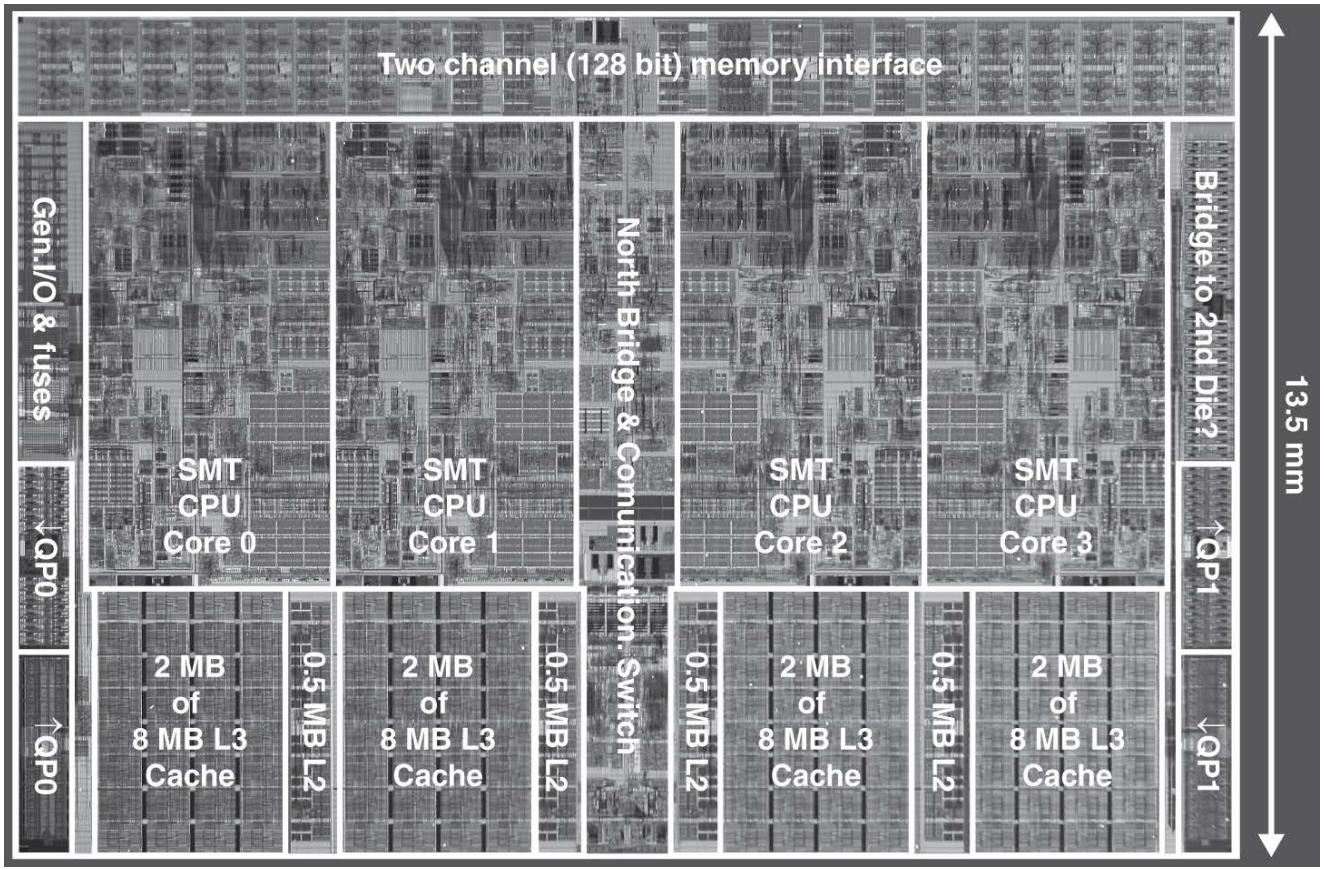
2-Level TLB Organization

- Cache hierarchy of the K8 core in the AMD Athlon 64 CPU



Multilevel On-Chip Caches

Intel Nehalem 4-core processor



Per core: 32KB L1 I-cache, 32KB L1 D-cache, 512KB L2 cache

2-Level TLB Organization

	Intel Nehalem	AMD Opteron X4
Virtual addr	48 bits	48 bits
Physical addr	44 bits	48 bits
Page size	4KB, 2/4MB	4KB, 2/4MB
L1 TLB (per core)	L1 I-TLB: 128 entries for small pages, 7 per thread (2×) for large pages L1 D-TLB: 64 entries for small pages, 32 for large pages Both 4-way, LRU replacement	L1 I-TLB: 48 entries L1 D-TLB: 48 entries Both fully associative, LRU replacement
L2 TLB (per core)	Single L2 TLB: 512 entries 4-way, LRU replacement	L2 I-TLB: 512 entries L2 D-TLB: 512 entries Both 4-way, round-robin LRU
TLB misses	Handled in hardware	Handled in hardware

3-Level Cache Organization

	Intel Nehalem	AMD Opteron X4
L1 caches (per core)	L1 I-cache: 32KB, 64-byte blocks, 4-way, approx LRU replacement, hit time n/a L1 D-cache: 32KB, 64-byte blocks, 8-way, approx LRU replacement, write-back/allocate, hit time n/a	L1 I-cache: 32KB, 64-byte blocks, 2-way, LRU replacement, hit time 3 cycles L1 D-cache: 32KB, 64-byte blocks, 2-way, LRU replacement, write-back/allocate, hit time 9 cycles
L2 unified cache (per core)	256KB, 64-byte blocks, 8-way, approx LRU replacement, write-back/allocate, hit time n/a	512KB, 64-byte blocks, 16-way, approx LRU replacement, write-back/allocate, hit time n/a
L3 unified cache (shared)	8MB, 64-byte blocks, 16-way, replacement n/a, write-back/allocate, hit time n/a	2MB, 64-byte blocks, 32-way, replace block shared by fewest cores, write-back/allocate, hit time 32 cycles

n/a: data not available

Memory Protection & OS Support

- Multiple processes share single memory
  - Avoid process can access other processes' memory (read or write)
  - Don't let processes modify the page tables
- Solution: operating system
  - Modifies a process' page table when needed
  - Also assists processes to access joint memory in a controlled way
    - Additional info needed in page table & TLB
- General OS concept
  - Privileged supervisor / executive / kernel mode
  - Parts of a process' state can be modified only in kernel mode
    - Page tables (located in OS address space), TLB, kernel mode bit, …
  - Privileged instructions
  - Change from user mode to kernel mode
    - System call exception
  - Change from kernel mode to user mode
    - "Return from exception" (ERET) instruction: processor continues with instruction in EPC

Cache Coherence Problem with Multiple Cores

- Suppose two CPU cores share a physical address space
  - Write-through caches
  - Coherence (Stimmigkeit) may be lost

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

Invalidating Snooping Protocols

- Cache gets exclusive access to a block when it is to be written
  - Broadcasts an invalidate message on the bus to other caches
  - Other caches flush entry
  - Subsequent read in another cache causes cache miss and reload

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		1
CPU B reads X	Cache miss for X	1	1	1

Concluding Remarks

- Fast memories are small, large memories are slow
  - We want fast and 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

Virtual Machine (VM)

- A software implementation of a machine (i.e. a computer) that executes programs like a physical machine
- Software running inside a VM is limited to the resources and abstractions provided by the VM – it cannot break out of its virtual environment

(Source: Wikipedia)

Process (or Application) VM

- Designed to run a single program, i.e., it supports a single process
- Provides a platform-independent programming environment
  - Abstracts away details of the underlying hardware or operating system
  - Allows a program to execute in the same way on any platform
- Examples
  - Java / Java VM
  - .NET / Common Language Runtime

(Source: Wikipedia)

System VM (1)

- Provides a complete system platform which supports the execution of a complete operating system (OS)

- Host computer emulates guest operating system and machine resources
  - Improved isolation of multiple guests
  - Avoids security and reliability problems
  - Aids sharing of resources

- Examples
  - IBM VM/370 (1970s technology!)
  - VMWare
  - Microsoft Virtual PC

System VM (2)

- Advantages
  - VMs run their own operating system (guest operating system) $\Rightarrow$ multiple OS environments can co-exist on the same computer
  - Ability to provide an ISA that is somewhat different from that of the real machine
  - Saves hardware and energy
    - Different services that normally run on individual machines to avoid interference can be instead run in separate VMs on the same physical machine (server consolidation)
- Disadvantages
  - Less efficient than a real machine when it accesses the hardware indirectly, but feasible with modern computers

(Source: Wikipedia)

Virtual Machine Monitor (VMM)

- Also called “hypervisor”
- 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