182.690 RECHNERSTRUKTUREN – MEMORY HIERARCHY

Thomas Polzer
tpolzer@ecs.tuwien.ac.at
Institut für Technische Informatik

The Problem

- Software developers expect fast, unbounded memory
- Large and fast memories are unaffordable

Memory type	Access time	Price/GB
SRAM	0.5 .. 2.5 ns	$2000 - $5000
DRAM	50 .. 70 ns	$20 - $75
HDD	10 .. 20 million ns	$0.2 - $2

SSD (Flash technology)

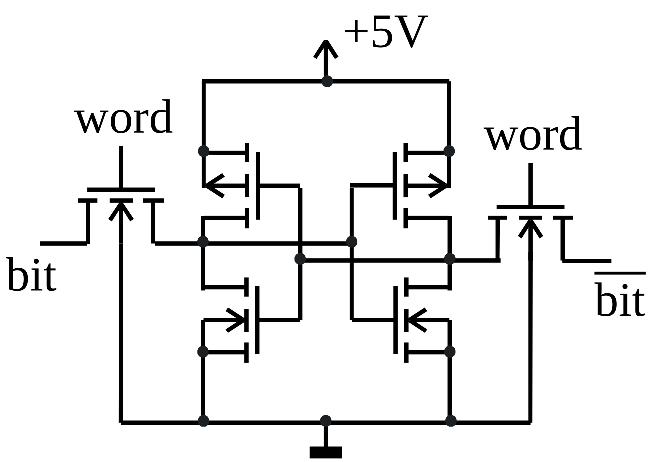
100-1000x faster than HDD

- Ideal: Access time of SRAM, price of HDD

Memory Components

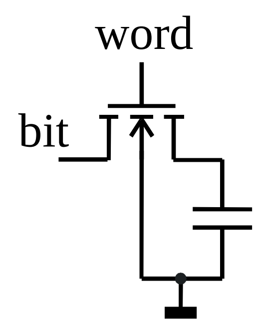
SRAM (static RAM)

- More complex
- Less susceptible to faults
- No refresh required
- Faster

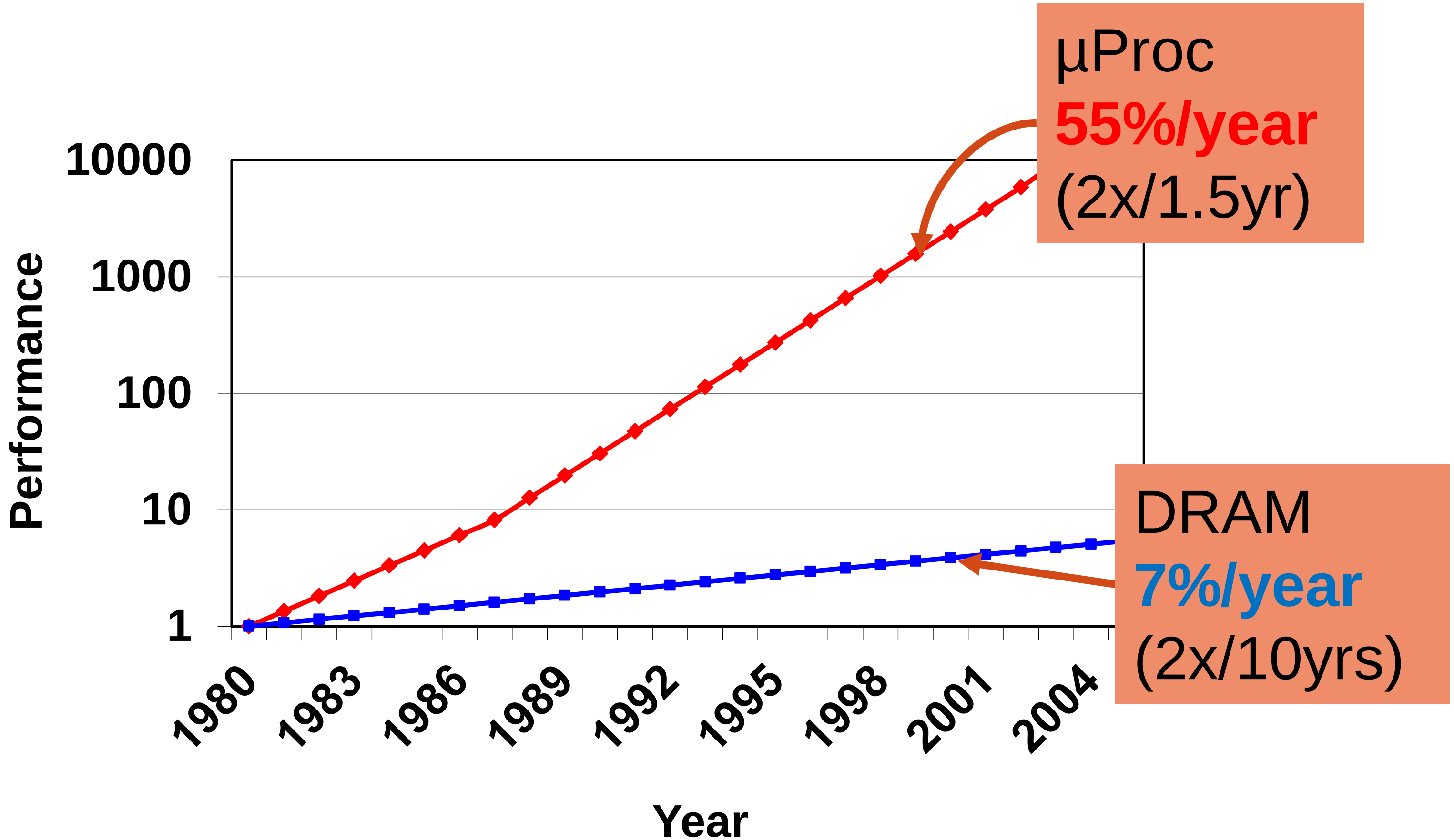


DRAM (dynamic RAM)

- Higher density
- More prone to faults
- Refresh required
- Slower (more complex access mechanism)



Processor-Memory Performance Gap



Processor-memory performance gap
(grows 50%/year)

“Ray of Hope”: Locality

- Items accessed recently are likely to be accessed again soon (e.g., instructions in a loop, induction variables)

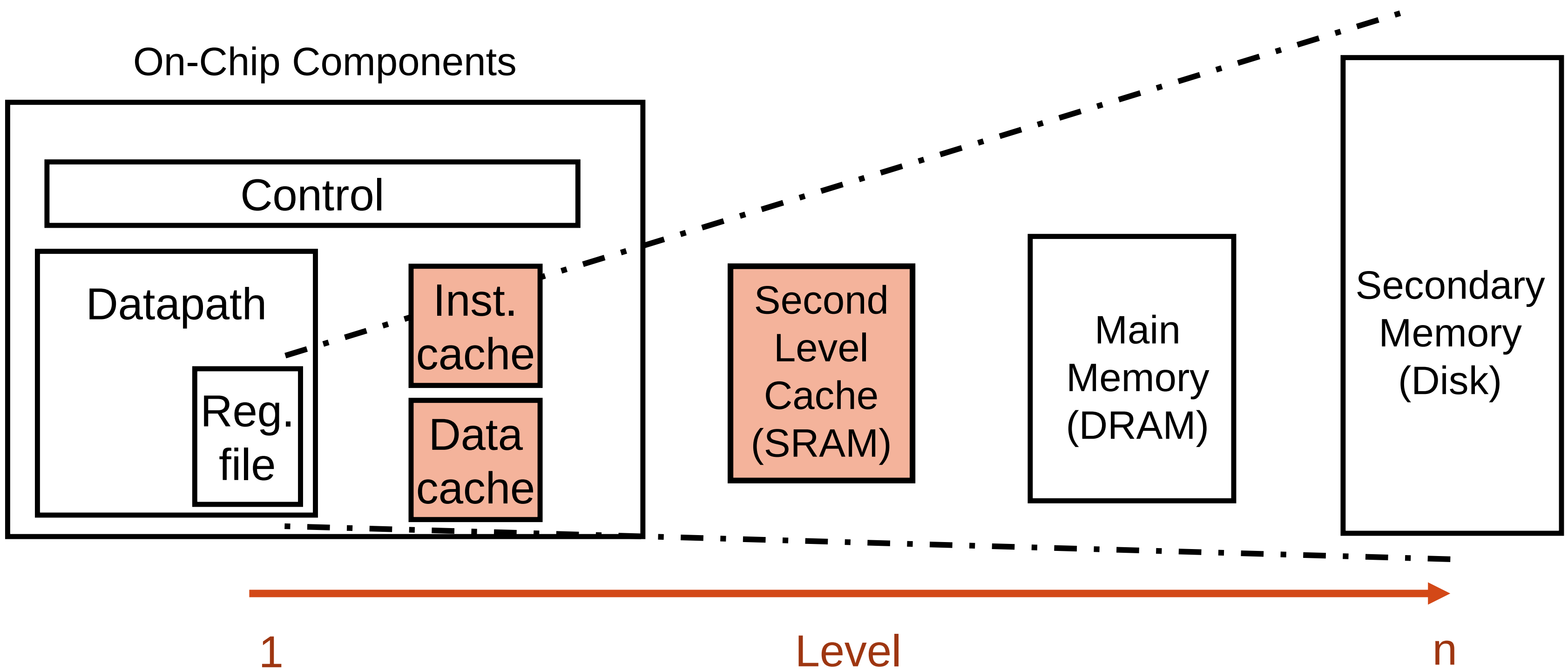
→Temporal Locality

- Items near those accessed recently are likely to be accessed soon (e.g., sequential instructions, array data)

→Spatial Locality

→Programs access a small proportion of their address space at any given time

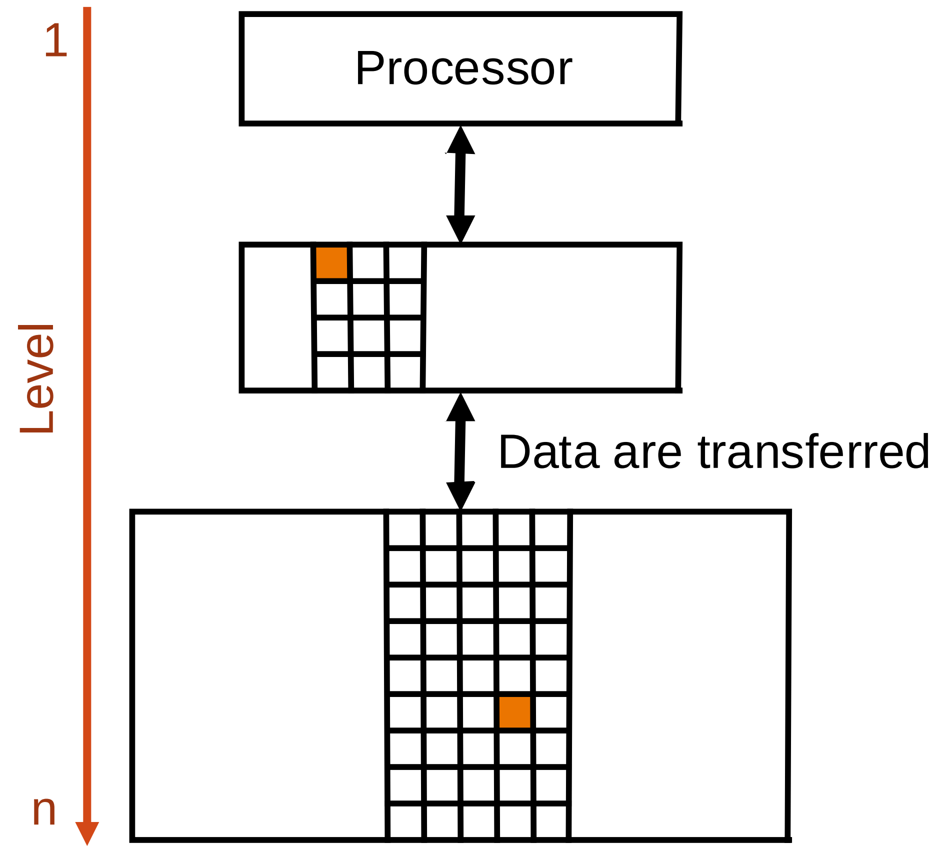
Solution: Memory Hierarchy



- Goal: Access time of level 1 for all data of level n
- Takes advantage of the principle of 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 rate**: hits/accesses)
- If accessed data is absent
  - **Miss**: block copied from lower level
    - Time taken: **miss penalty**
    - **Miss rate**: misses/accesses
  - Then accessed data supplied to upper level



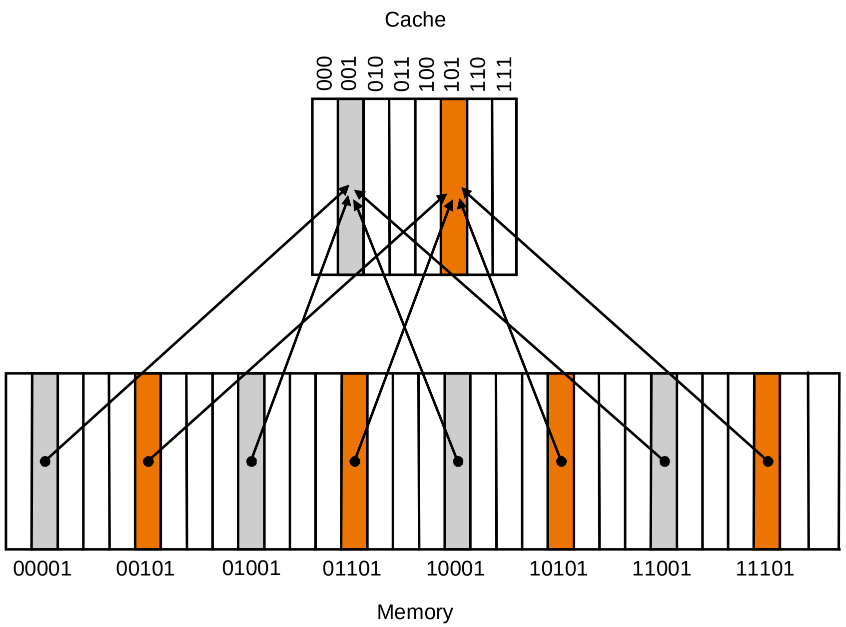
Cache Principle

- SRAM enables fast CPU access to instruction and data
  - Assumption when designing pipelining was 1 cycles
- If cache miss: information is loaded from main memory
- CPU never accesses main memory directly!

- Problems:
  - How to determine if required data is already in cache?
  - How to find data in the cache?

Direct Map Cache

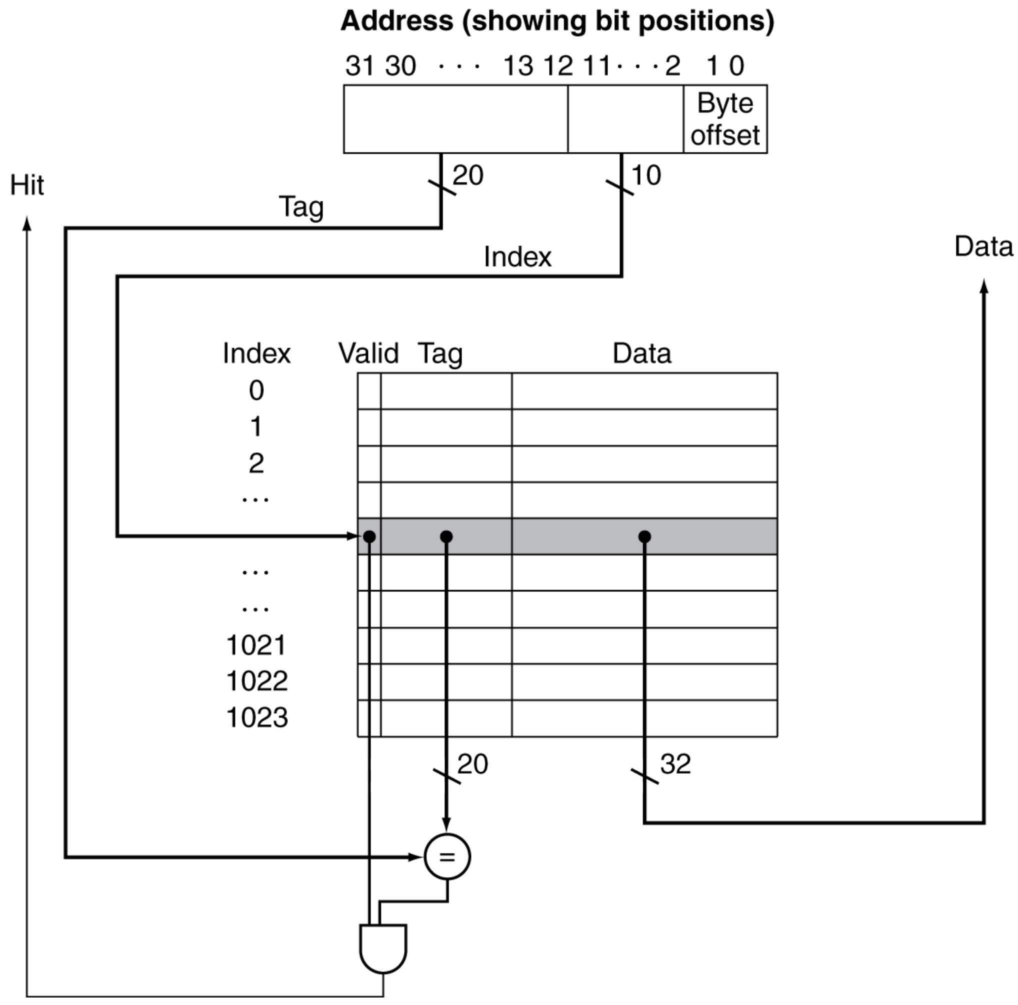
- Location determined by part of address (cache index)
- Direct mapped: only one choice possible
  - (Block address) modulo (#Blocks in cache)
- #Blocks is a power of 2
- Use low-order address bits



Tags and Valid Bits

- How do we know which particular block is stored in a cache location?
  - Store block address as well as the data
  - Actually, only need the high-order bits
  - Called the **tag**
- What if there is no data in a location?
  - Valid bit: 1 = present, 0 = not present
  - Initially 0

Address Subdivision



Cache Example

- 8-blocks, 1 word/block, direct mapped
- Initial state

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

Cache Example

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		

Cache Example

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		

Cache Example

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		

Cache Example

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		

Cache Example

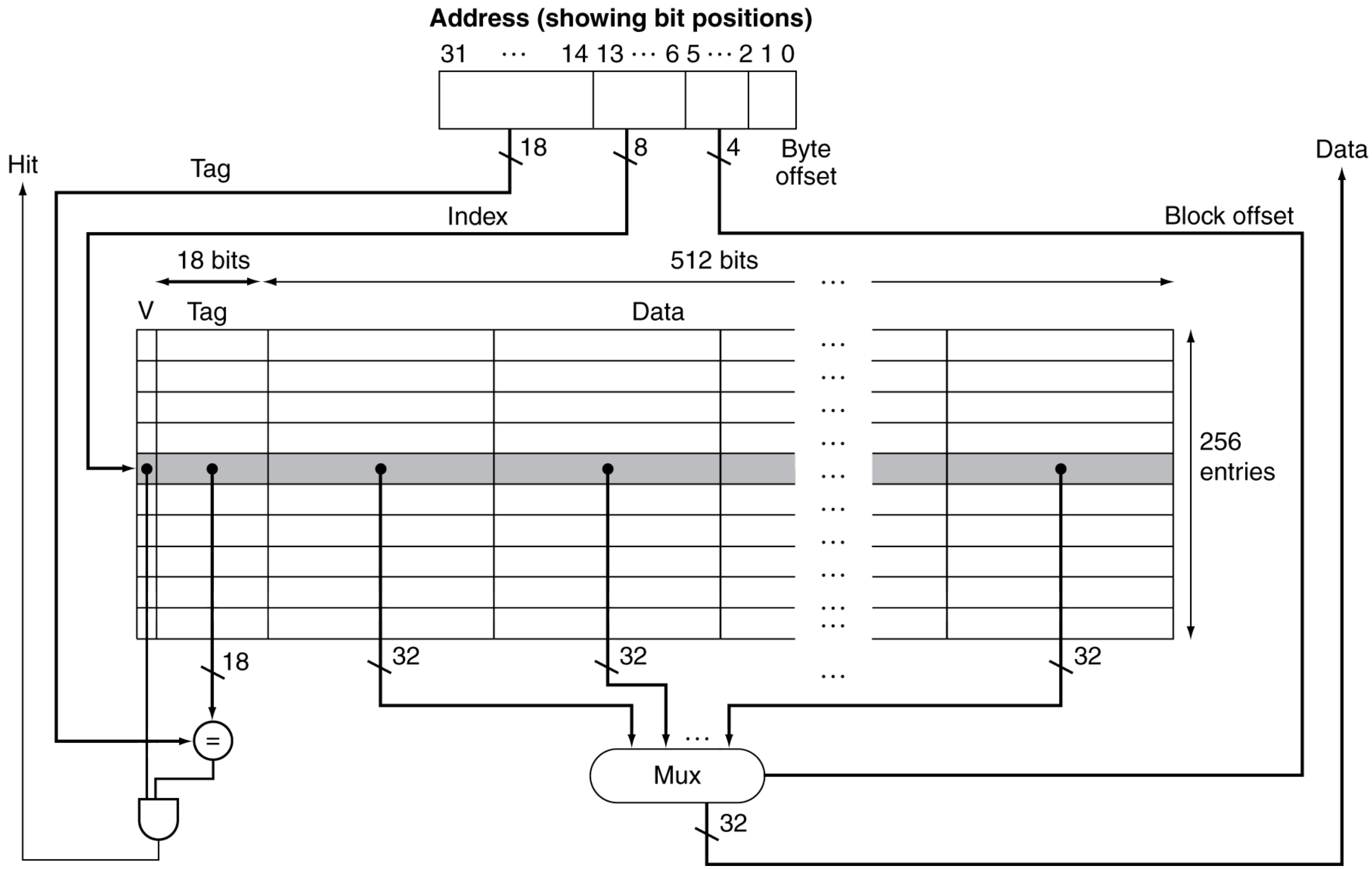
Word addr	Binary addr	Hit/miss	Cache block
18	10 010	Miss	010

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		

Example: Multiword Cache

- 256 blocks, 16 word/block
  - To which block number does address 24000 map?
- Block address = $\lfloor 6000/16 \rfloor$ = 375
- Block number = 375 modulo 256 = 119

Multiword Cache

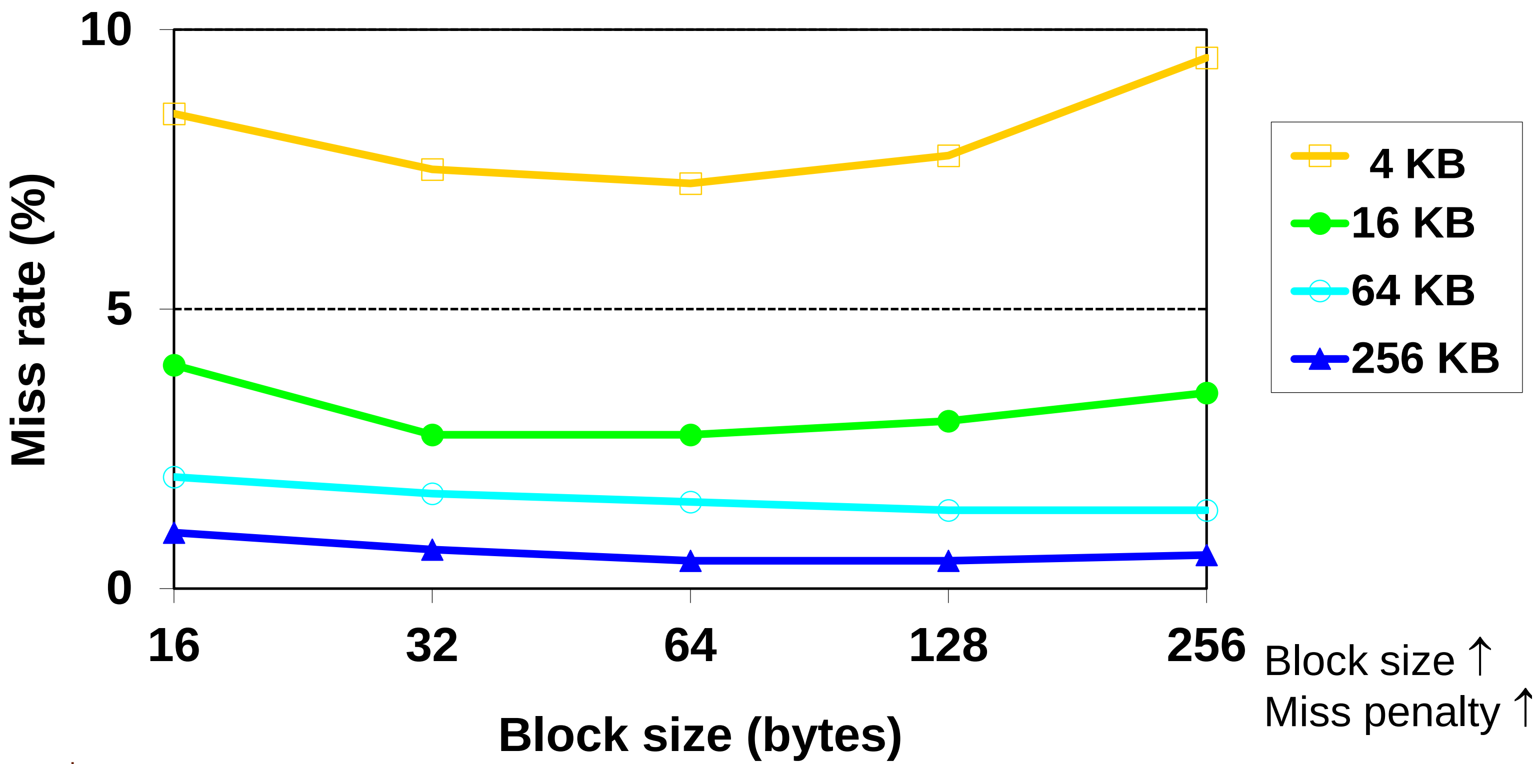


Block Size Considerations

- Larger blocks should reduce miss rate
  - Due to spatial locality
- But in a fixed-sized cache
  - Larger blocks $\Rightarrow$ fewer of them
  - More competition $\Rightarrow$ increased miss rate
  - Larger blocks $\Rightarrow$ spatial locality may decrease
- Larger miss penalty
  - Can override benefit of reduced miss rate
  - Early restart and critical-word-first can help

Miss Rate vs Block Size vs Cache Size

- Miss rate goes up if the block size becomes a significant fraction of the cache size because the number of blocks that can be held in the same size cache is smaller



Block size ↑
Miss penalty ↑

Cache Misses

- On cache hit:
  - CPU proceeds normally
  - Goal accomplished ☺
- On cache miss
  - Stall the CPU pipeline
  - Fetch block from next level of hierarchy
  - Continue with the execution

Hits & Misses When Writing Data

Hit

Write-Through

Copy-Back

Write-Around

Miss

Fetch-on-Write

Write Hit: Write-Through

- Update the cache and the main memory immediately

- Data consistency guaranteed
- Frequent accesses to the main memory
- Performance penalty

Write Buffer

- Example:

  13% of all instructions execute a write,

  CPI without miss = 1.2

  Length of a write: 10 cycles

  ➔ CPI with writes = 1.2 + 10 * 0.13 = 2.5
- Improvement: write buffer
  - Holds data waiting to be written to memory
  - CPU continues immediately except if buffer is full

Write Hit: Write(Copy)-Back

- Updates the cache and marks the cache block as dirty
- When a dirty block is replaced
  - Write it back to memory
  - Can use a write buffer to allow replacing block to be read first

- No consistency between main memory and cache
- Writes are faster
- Read may be slower (due to copy operation)
- Fewer accesses to the main memory as in case of write through policy

Write Miss: Write Around

- Ignore the cache and write directly to main memory

Write Miss: Fetch on Write

- Replace the current content of the cache line and update its tag
- If block size greater than one word:
  - Fetch the other words of the line from main memory

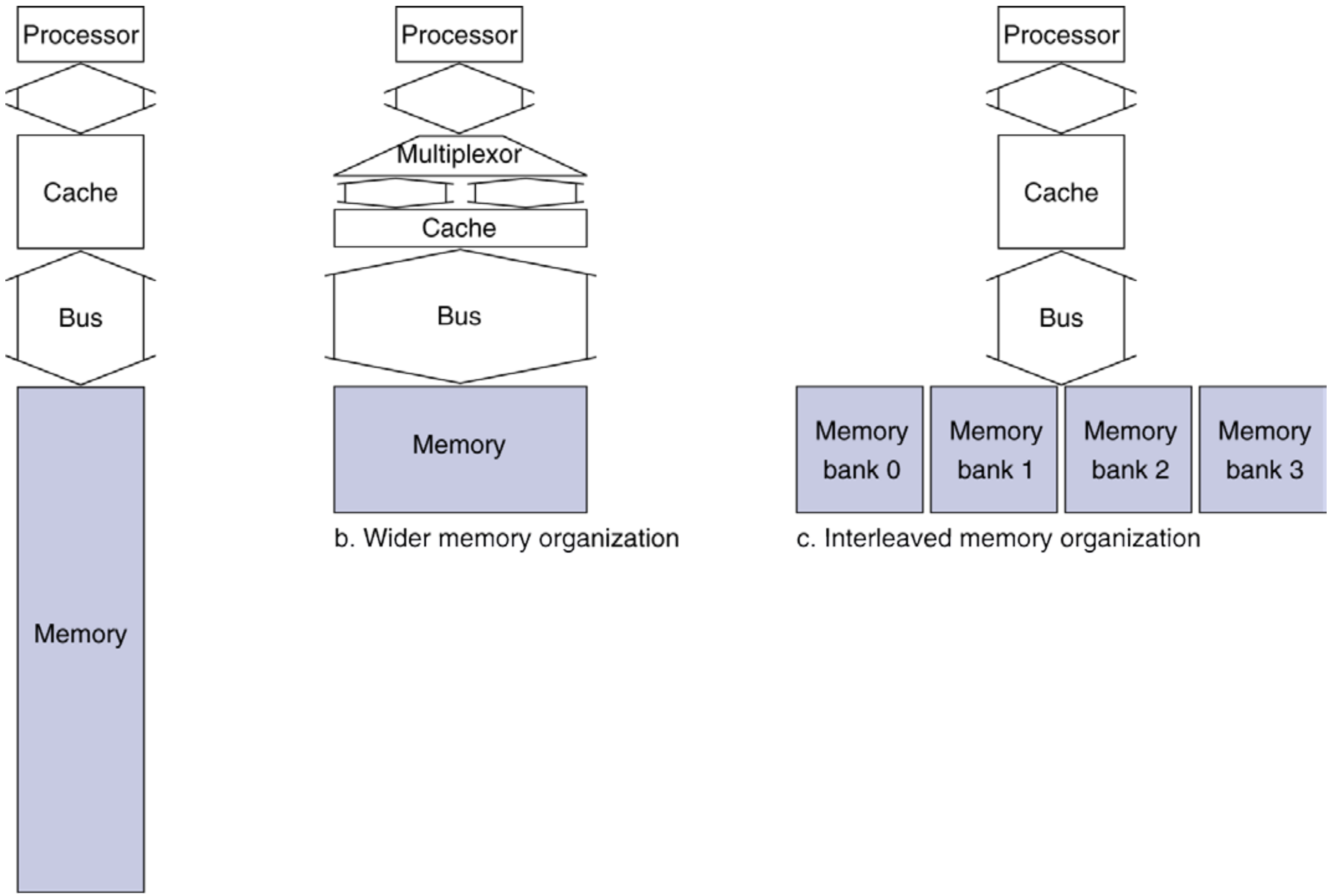
Write Allocation

- Miss-alternatives for write-through
  - Fetch on write
  - Write around
- For write-back
  - Usually fetch the block

Main Memory Supporting Caches

- Use DRAMs for main memory
  - Fixed width (e.g., 1 word)
  - Connected by fixed-width clocked bus
- 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 = $1 + 4 \times 15 + 4 \times 1 = 65$ bus cycles
  - Bandwidth = 16 bytes / 65 cycles = 0.25 B/cycle

Increasing Memory Bandwidth



a. One-word-wide memory organization

b. Wider memory organization

c. Interleaved memory organization

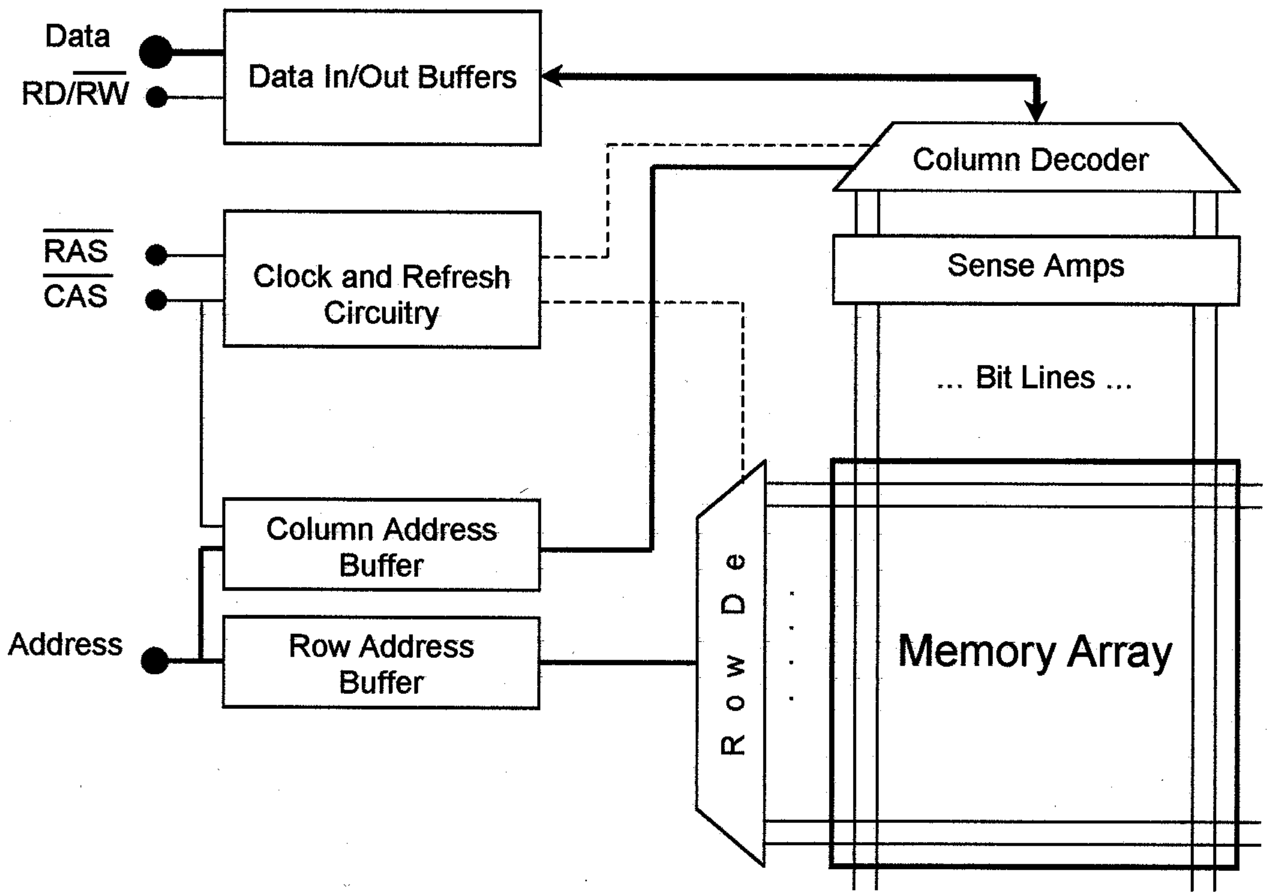
Advanced DRAM Organization

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

Advanced DRAM Organization

- Bits in a DRAM are organized as a rectangular array
  - 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
- Quad data rate (QDR) DRAM
  - Separate DDR inputs and outputs

Internal DRAM Structure



Measuring Cache Performance

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

$$\text{Memory stall cycles} \qquad \text{(simplified read and write combined)}$$

$$= \frac{\text{Memory accesses}}{\text{Program}} \times \text{Miss rate} \times \text{Miss penalty}$$

$$= \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Misses}}{\text{Instruction}} \times \text{Miss penalty}$$

Cache Performance Example

- Given
  - 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: $0.02 \times 100 = 2$
  - D-cache: $0.36 \times 0.04 \times 100 = 1.44$
- Actual CPI = $2 + 2 + 1.44 = 5.44$
  - Ideal CPU is $5.44/2 = 2.72$ times faster

Average Access Time

- Hit time is also important for performance
- Average memory access time (AMAT)
  - AMAT = Hit time + Miss rate × Miss penalty
- Example
  - CPU with 1ns clock, hit time = 1 cycle, miss penalty = 20 cycles, I-cache miss rate = 5%
  - AMAT = 1 + 0.05 × 20 = 2ns
    - 2 cycles per instruction

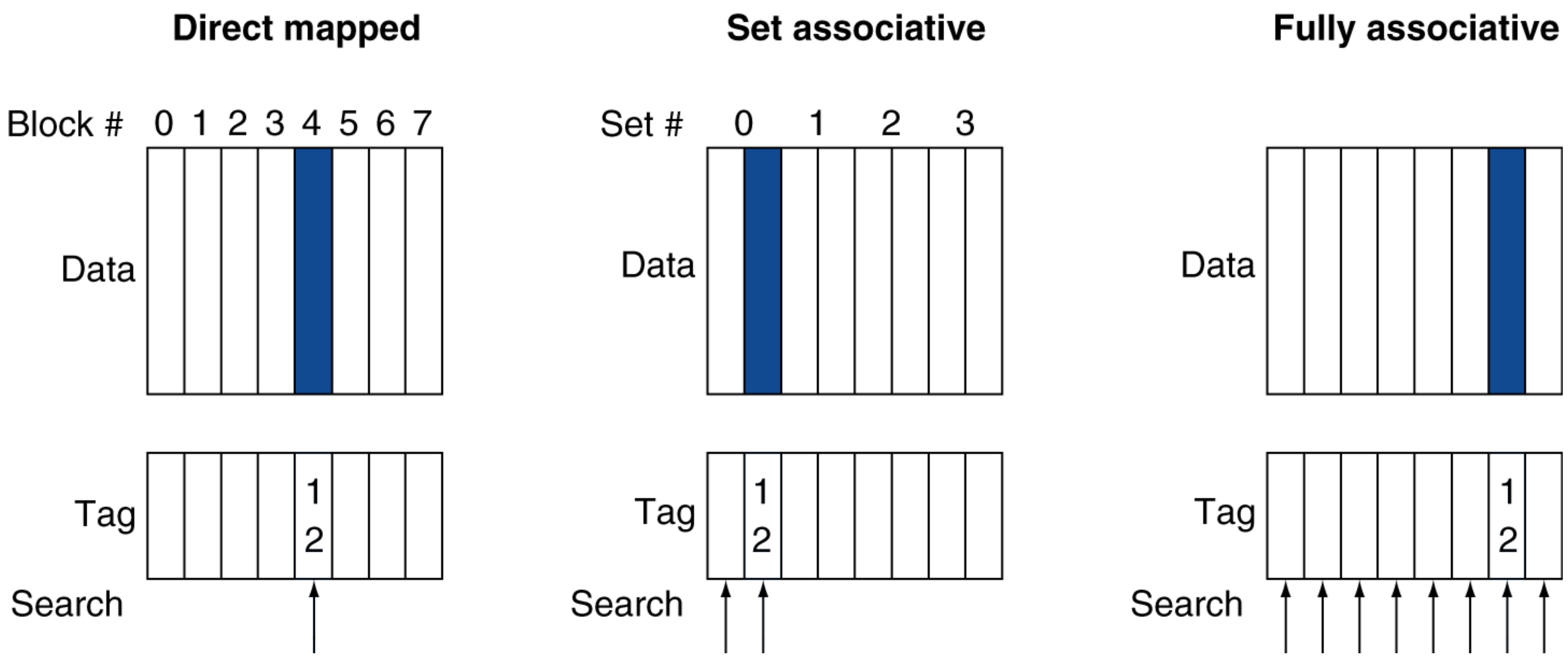
Performance Summary

- 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

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

Associativity of Caches



Direct Mapped Cache

- Each block is mapped to only one, unique position

- Simple cache organization
- No multiplexer in datapath
- Moderate hit rate

Fully Associative Cache

- Each block may be placed at any entry of the cache

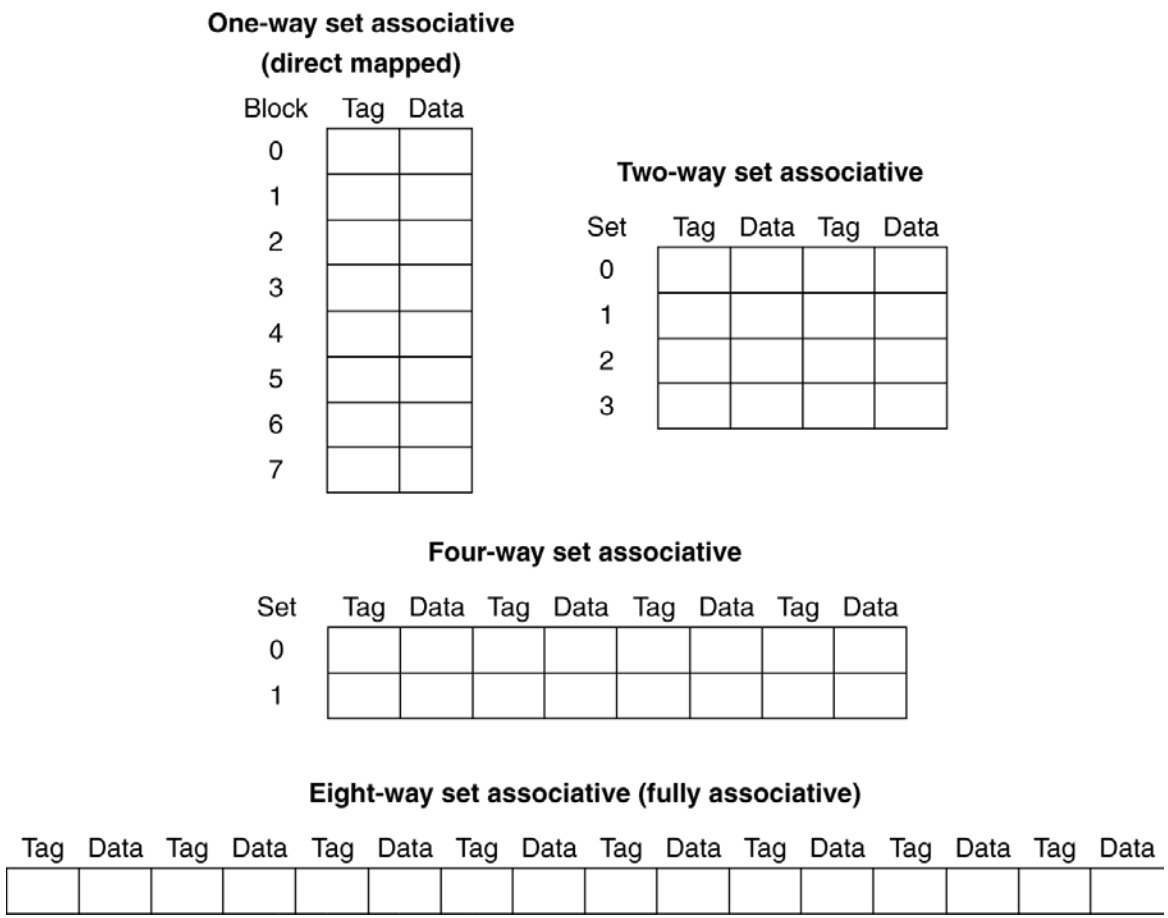
- Optimum hit rate
- Complex cache organization
  - Which entry should be replaced? (global optimum)
  - Where to look for a block? (Searching whole cache)

N-Way Set-Associative Cache

- Each block may be placed within N different cache entries
- Cache organizations still manageable
  - Searching only in constricted range (set)
  - Simple replacement rules (LRU)
- Good hit-rate

Spectrum of Associativity

- For a cache with 8 entries



Associativity Example

- Compare 4-block caches
  - Direct mapped, 2-way set associative, fully associative
  - Word access sequence: 0, 8, 0, 6, 8
- Direct mapped

5 misses

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]	

Associativity Example

- 2-way set associative (LRU replacement)

4 misses

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]		

- Fully associative

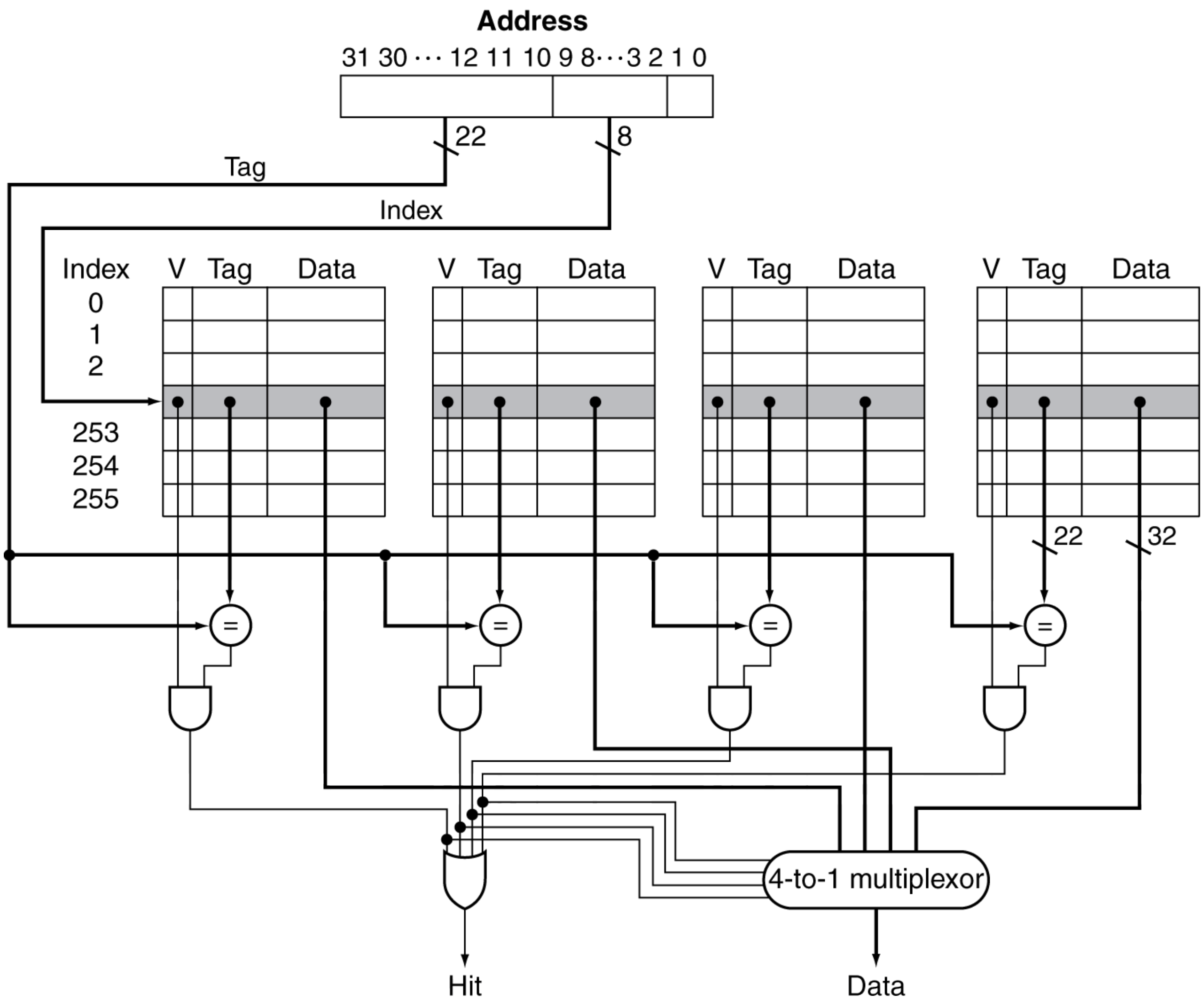
3 misses

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]	

How Much Associativity

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

Set Associative Cache Organization



Replacement Policy

- Direct mapped: no choice
- Set associative
  - Prefer non-valid entry, if there is one
  - Otherwise, choose among entries in the set
- Fully associative
  - Prefer non-valid entry, if there is one
  - Otherwise, choose among any entry in the cache

- Least-recently used (LRU)
  - Choose the one unused for the longest time (Simple for 2-way, manageable for 4-way, too hard beyond that)
- FIFO (first-in first-out)
- Random

Multilevel Caches

- Primary cache attached to CPU
  - Small, but fast
- Level-2 cache services misses from primary cache
  - Larger, slower, but still much 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 = 400 cycles
- CPI = $1 + 0.02 \times 20 + 0.005 \times 400 = 3.4$
- 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 analyze
  - Use system simulation

Pitfalls

- Byte vs. word addressing (in simulation)
  - 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

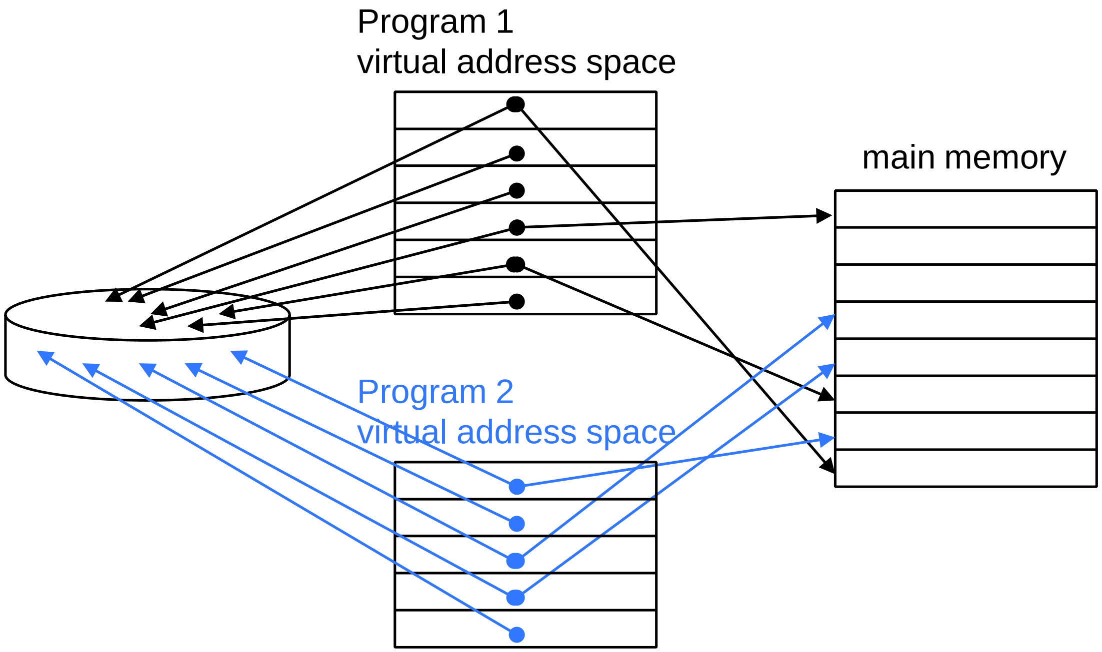
- In multiprocessor with shared L2 or L3 cache
  - Less associativity than cores results in conflict misses
  - More cores $\Rightarrow$ need to increase associativity
- Using average memory access time to evaluate performance of out-of-order processors
  - Evaluate performance by simulation

Virtual Memory

- Use DRAM as “cache” for secondary (disk) storage
- Emulates “infinite” amount of main memory
- Multiple programs can be executed independently
  - Each has private virtual address space
  - Memory access protection
- Address translation converts from virtual to physical address
  - Memory segmented into blocks
  - Blocks may be relocated (even to disk storage!)

Two Programs Sharing Physical Memory

- Programs memory space is divided into pages of fixed size
- Start address of each page stored in program's page table

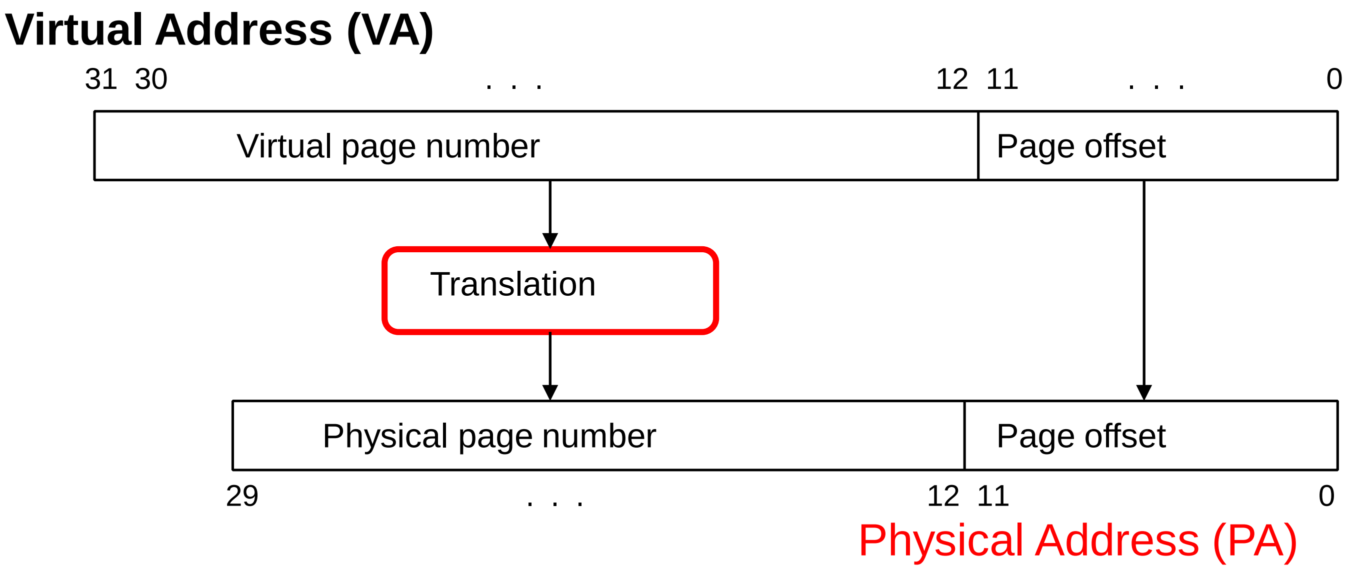


Nomenclature

Term	Description
page	memory block of fixed size, unit of memory management
page fault	page could not be found in DRAM, miss in virtual memory
virtual address	memory address as seen by the CPU
physical address	memory address as seen by the DRAM
address translation	Translation form virtual to physical addresses

Address Translation

- A virtual address is translated to a physical one by a combination of hard- and software



- Therefore each memory access first requires an address translation from virtual to physical memory

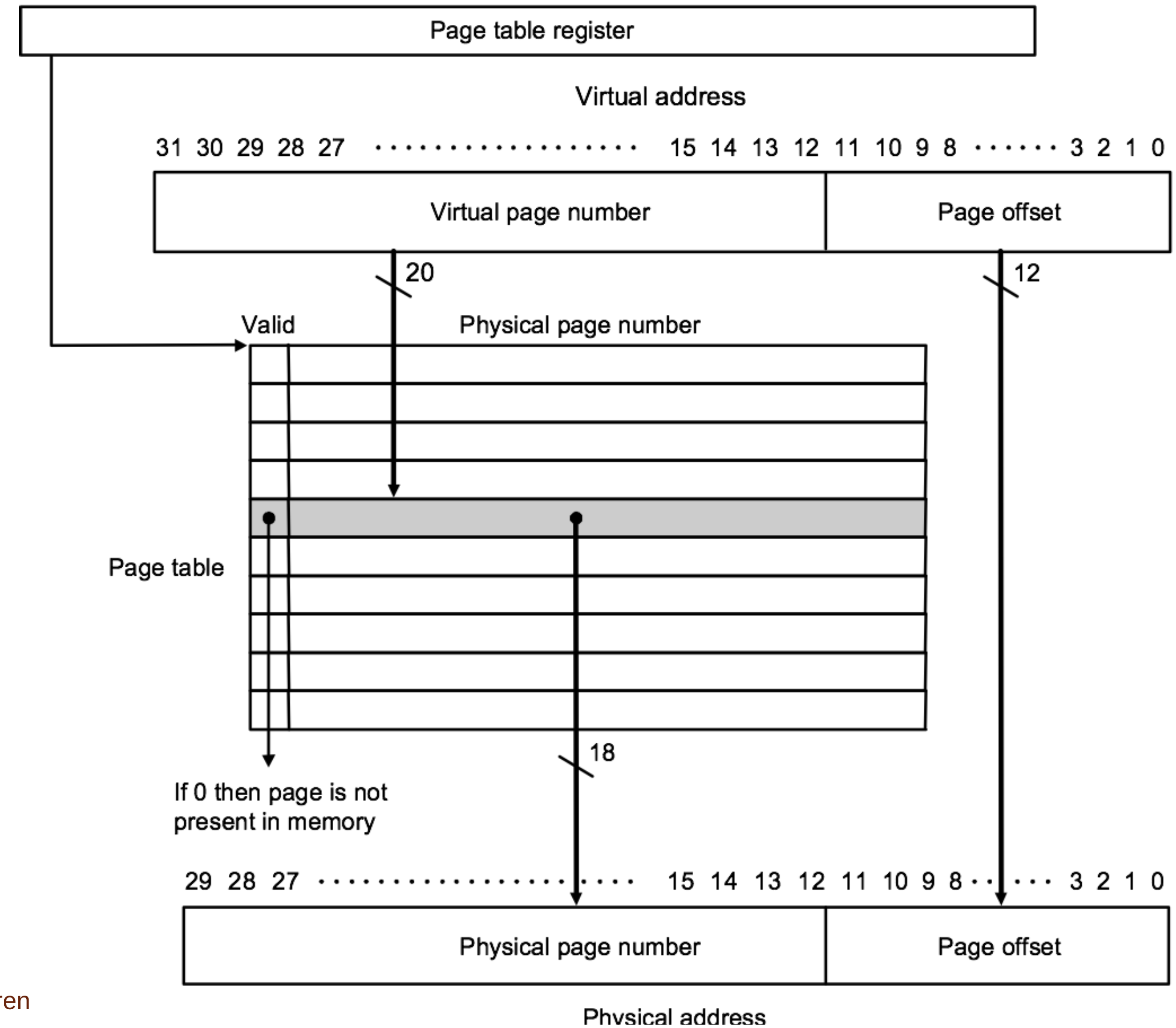
Page Fault Penalty

- On page fault: Fetch page from disk
  - Takes millions of clock cycles
  - Handles by OS code
- Try to minimize page fault rate
  - Fully associative placement in memory
  - Smart replacement algorithms

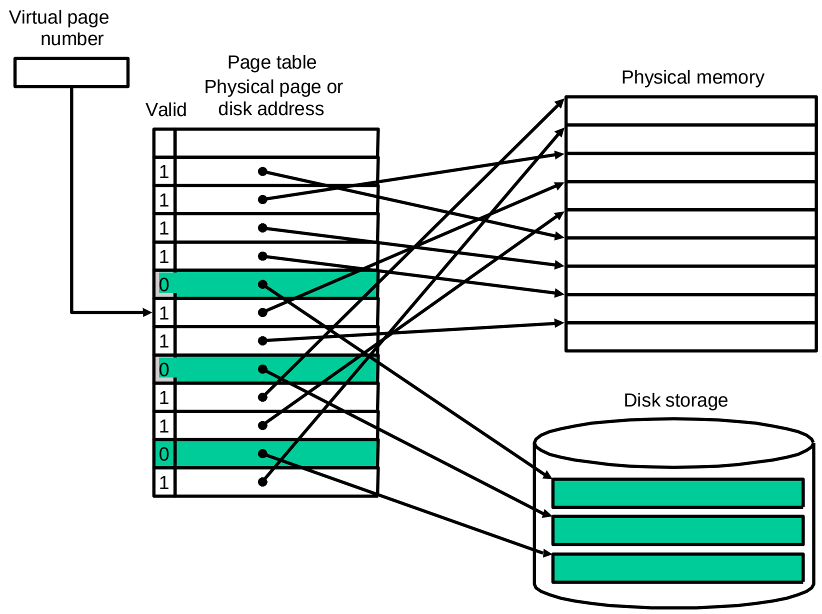
Page Tables

- Stores placement information
  - Array of entries indexed by the virtual page number
  - Page table register in CPU points to page table in memory
- If page is present in memory
  - Page table stores its physical address
  - Plus some status information
- If not present
  - Page table can refer to location on disk

Page Tables


Page table register
Virtual address
31 30 29 28 27 . . . . 15 14 13 12 11 10 9 8 . . . . . . 3 2 1 0
Virtual page number
Page offset
20
12
Valid
Physical page number
Page table
18
If 0 then page is not
present in memory
29 28 27 . . . . 15 14 13 12 11 10 9 8 . . . . 3 2 1 0
Physical page number
Page offset
Physical address

Page Fault



- If valid = 0 → page fault → exception → OS reloads page from disk

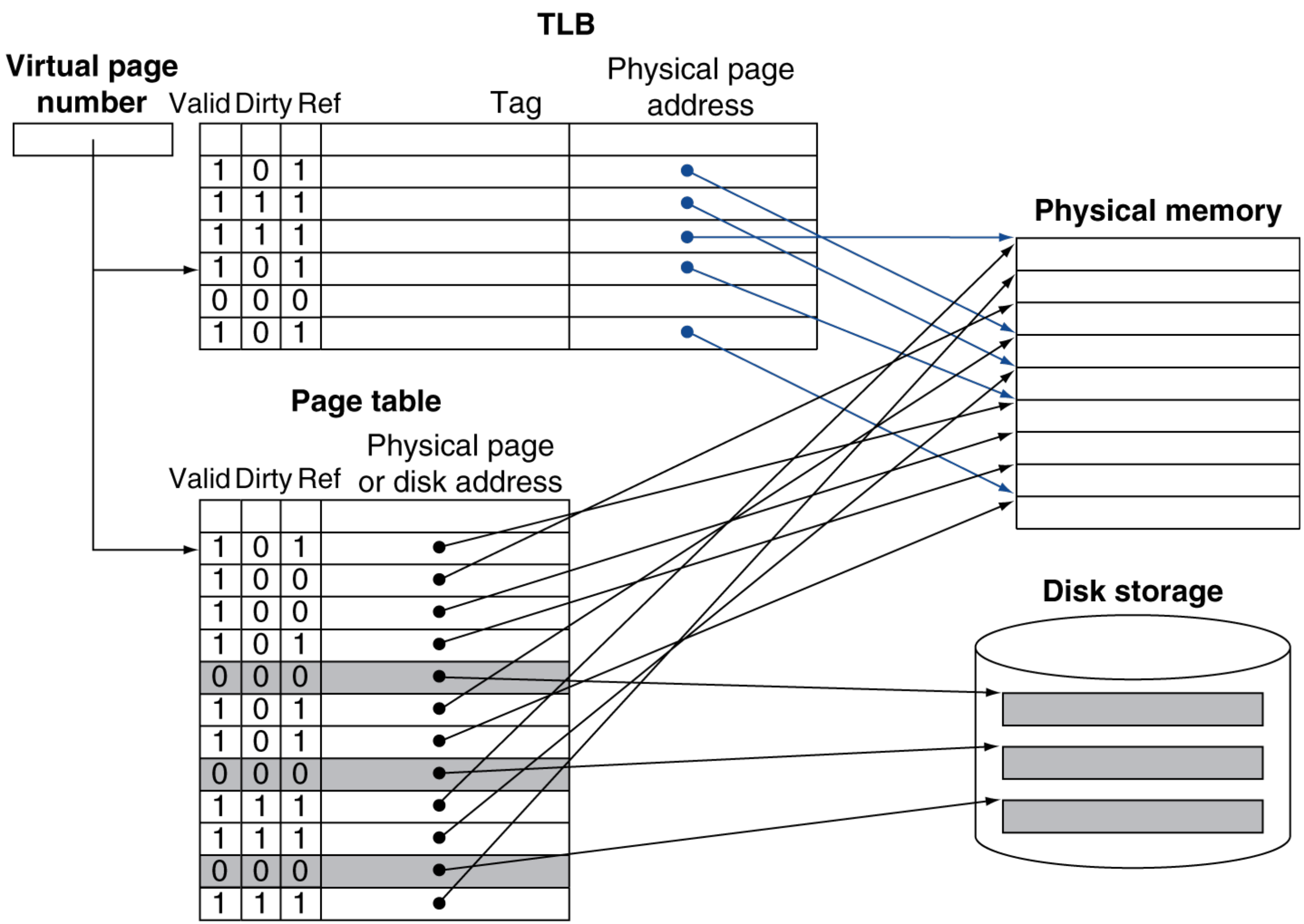
Replacements and Writes

- To reduce page fault rate, LRU is implemented
  - Referenced bit set on access of the page
  - Periodically cleared by OS
  - → If referenced bit = 0, page not used recently and can be swapped out
- Disk writes very expensive (millions of cycles!)
  - Write through impractical!
  - Use write back!
  - Dirty bit in page table keeps track of modified pages

Performance Improvement

- Address translation needs two memory accesses, one for the page table, one for the actual memory location
- But good locality for page accesses!
  - Try to cache recently used pages within the CPU → Translation Look-aside Buffer
  - Typical size: 16-512 page table entries
  - Typically 0.5 - 1 cycle on hit, 10-100 cycles on miss, miss rate: 0.01% - 1%

Translation Look-aside Buffer



TLB - Miss

- If page is in memory
  - Load page table entry from memory and retry
  - Could be handled directly in hardware or in software
- If page is not in memory (page fault)
  - OS handles fetching of page and updates page table

Memory Protection

- Different task but can reuse parts of the hardware
  - Needs to protect against errant access
  - Requires OS assistance
- Hardware support for OS assistance
  - Privileged supervisor mode
  - Privileged instructions
  - Page table and other state information only accessible in supervisor mode
  - System call exception

Conclusion (1)

- Processor speed increases faster as the speed of the memory
- Caches utilize the locality of programs to utilize the fast but expensive memory optimally
- Memory performance may impact the system performance drastically
- Cache performance mainly given by miss rate and miss penalty
- Larger blocks utilize the locality better but also increase the miss penalty

Conclusion (2)

- On write hits either a write through or a write back is performed
- On write misses a fetch on write is executed in most cases but other strategies exist
- For n-way associative caches each block can be placed on n different locations. Direct mapped cache (n=1) und fully associative cache (N=whole cache) are the extreme cases
- High associativity increase the hit rate but also the hardware overhead

Conclusion (3) - Cache Design Trade-off

Design change	Effect on miss rate	Negative performance effect
↑ cache size	↓ capacity misses	May ↑ access time
↑ associativity	↓ conflict misses	May ↑ access time
↑ block size	↓ compulsory misses	↑ miss penalty. For very large block size, may ↑miss rate.

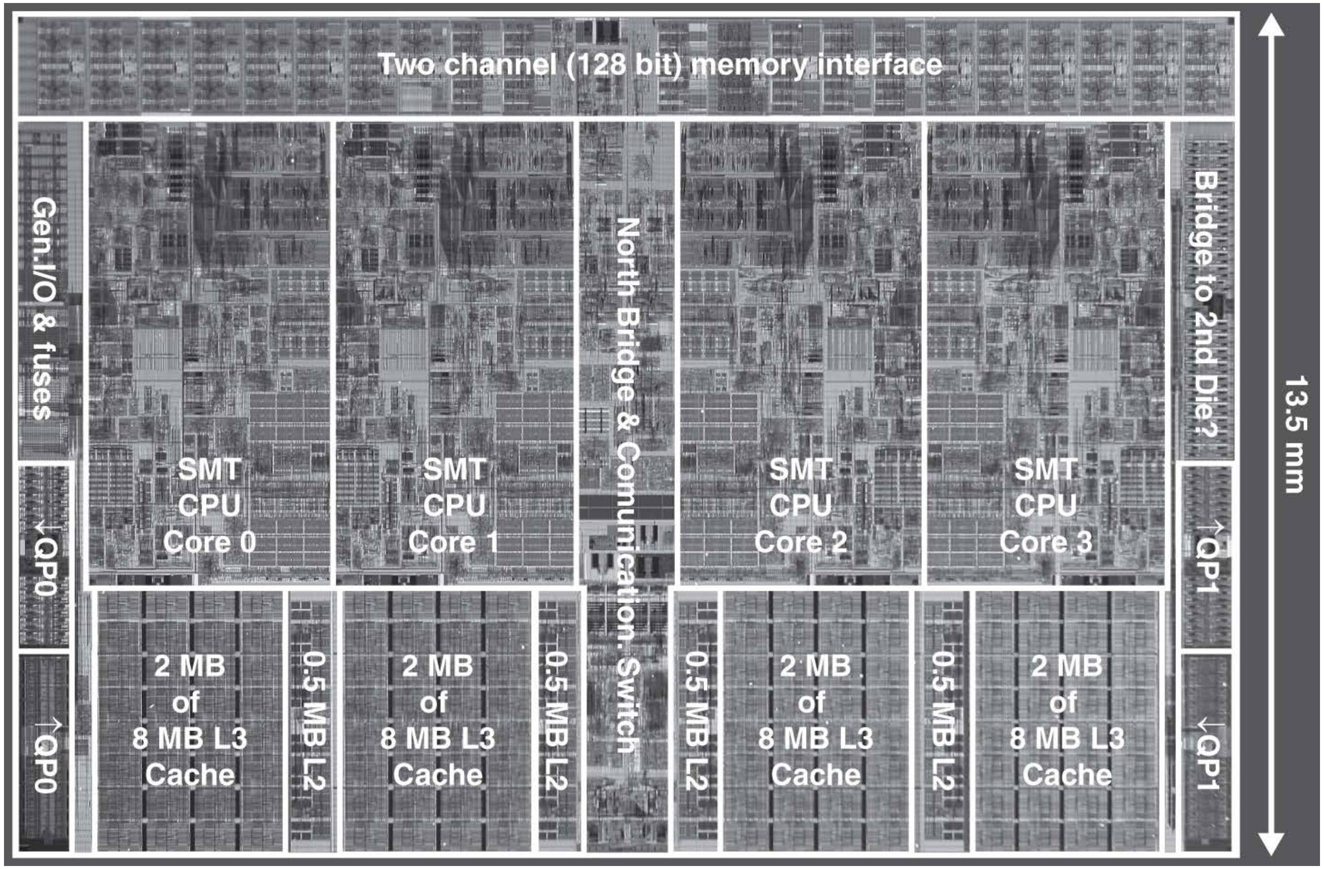
Conclusion (4)

- When using virtual memory, the DRAM acts like a cache for the hard disk
- High miss penalty → large blocks, full associativity and write back only
- Page table translates between virtual and physical addresses
- Address translation enables relocation and memory protection
- Translation look-aside buffer is a cache for the page table

EXAMPLES

Multilevel On-Chip Caches

Intel Nehalem 4-core processor



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

Level-2 TLB Organization

	Intel Nehalem	AMD Opteron X4
Virtual addr	48 bits	48 bits
Physical addr	44 bits	48 bits (256 TB)
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

Level-3 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

Cache

You have 18 pcs of 32kx8 SRAMs and want to build an instruction cache for a processor with 32bit address space.

- What is the maximum possible size for a direct mapped cache with one word (32 bit) blocks (memory capacity in bytes)?
- Describe how you split the addresses to access the cache and describe how the SRAM chips are used.

Cache

What we know:

- Address width $A = 32$ Bit
- Word width $W = 32$ Bit
- Byte offset $O = \log_2(W/8) = 2$ Bit
- Block size $B = 32$ Bit
- Cache size $G = 18*32*1024*8 = 4718592$ Bit
- 1 valid bit / entry
- #Index bits $X = ???$
- #Tag bits $T = A – O – X$

Cache

- The cache size is dependent on the number of blocks which is derived from the index.
- Cache size G = data size + tag size + # valid bits
  - $G \geq 2^X * B + 2^X (A - O - X) + 2^X$
  - $G \geq 2^X * (W + A - O - X + 1)$

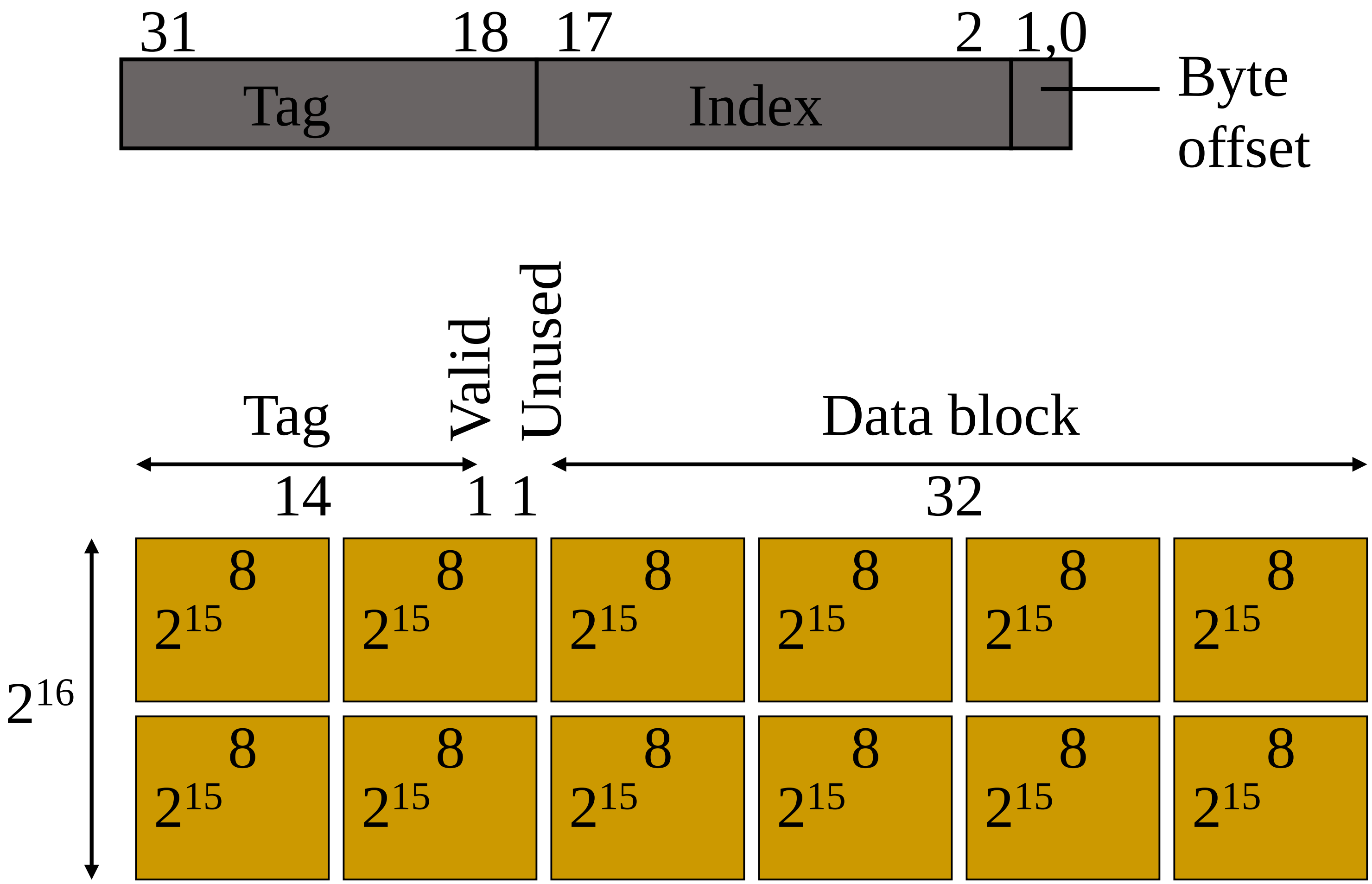
$\Rightarrow$ $G = 2^X * (63 - X) \leq 4718592$

$\Rightarrow$ $X = 16 \Rightarrow 65536$ entries (blocks) in the cache

$\Rightarrow$ $T = 32 - 2 - 16 = 14$

$\Rightarrow$ Data capacity = 256kB

Cache


31
18
17
2
1,0
Tag
Index
Byte
offset
Tag
Valid
Unused
Data block
14
1 1
32
8
2^{15}
2^{16}

Cache Size

- Assume that the address width of a computer is k bit and byte addressing is used. The cache size is S bytes, the block size B bytes and the cache is A-way set-associative. B is a power of two, therefore $B = 2^b$. Derive the following parameters in dependence of S, B, A, b and k:
- Number of sets in the cache
- Number of index bits
- Number of bits required to implement the cache

Cache Size

a) A set has A blocks, a block has B bytes

→ A set has A * B bytes

→ Number of possible sets in the cache: S / (A * B)

b) The index is used to specify a set

→ S / (A * B) addresses

→ #index bits = $\log_2(S / (A * B)) = \log_2(S / A) - b$

Cache Size

c) (Pure) data width: $S * 8$ bit

Tag width = Address width – block offset - #index bits

$= k - b - (\log_2(S / A) - b) = k - \log_2(S / A)$

Memory for tags = tag width * #sets * #blocks / set =

$= (k - \log_2(S / A)) * S / (A * B) * A =$

$= S / B * (k - \log_2(S / A))$

Overall cache memory =

$= S * 8 + S / B * (k - \log_2(S / A) + 1)$

Average Memory Access Time

Calculate the average memory access time for a machine with a two nano second clock cycle time, a miss penalty of 20 clock cycles, a miss rate of five percent and a hit time of one clock cycle (Assumption: Read and write miss penalties are the same, no other write stalls).

Solution:

AMAT = 2 ns + 20 * 2 ns * 0.05 = 4 ns

Average Memory Access Time

Assume you can lower the miss rate to three percent by doubling the cache. Unfortunately this increases the hit time by 20%. Calculate the AMAT to judge the viability of this improvement.

Solution: AMAT = 1.2 * 2 ns + 20 * 2 ns * 0.03 = 3.6 ns

The calculated result is, however, only hypothetical as the hit time would be increased by a whole clock cycle:

Solution: AMAT = 2 * 2 ns + 20 * 2 ns * 0.03 = 5.2 ns

Cache Access

Consider memory accesses with the following word addresses: 1, 4, 8, 5, 20, 17, 19, 56, 9, 11, 4, 43, 5, 6, 9 and 17.

Show the hits and misses for a two way set associative cache with a cache size of 16 words, a block size of one word and LRU replacement policy.

Cache Access

Referenz	Cache
1	
4	
8	
5	
20	
17	
19	
56	
9	
11	
4	
43	
5	
6	
9	
17	

Set#	Block Index 0	Block Index 1
0		
1		
2		
3		
4		
5		
6		
7		

Cache Access

Referenz	Cache
1	Miss
4	Miss
8	Miss
5	Miss
20	Miss
17	Miss
19	Miss
56	Miss
9	Miss
11	Miss
4	Hit
43	Miss
5	Hit
6	Miss
9	Hit
17	Hit

Set#	Block Index 0	Block Index 1	
0	56	8	
1	17	9	1
2			
3	43	11	19
4	4	20	
5	5		
6	6		
7			

182.690 RECHNERSTRUKTUREN – MEMORY HIERARCHY

Thomas Polzer
tpolzer@ecs.tuwien.ac.at
Institut für Technische Informatik