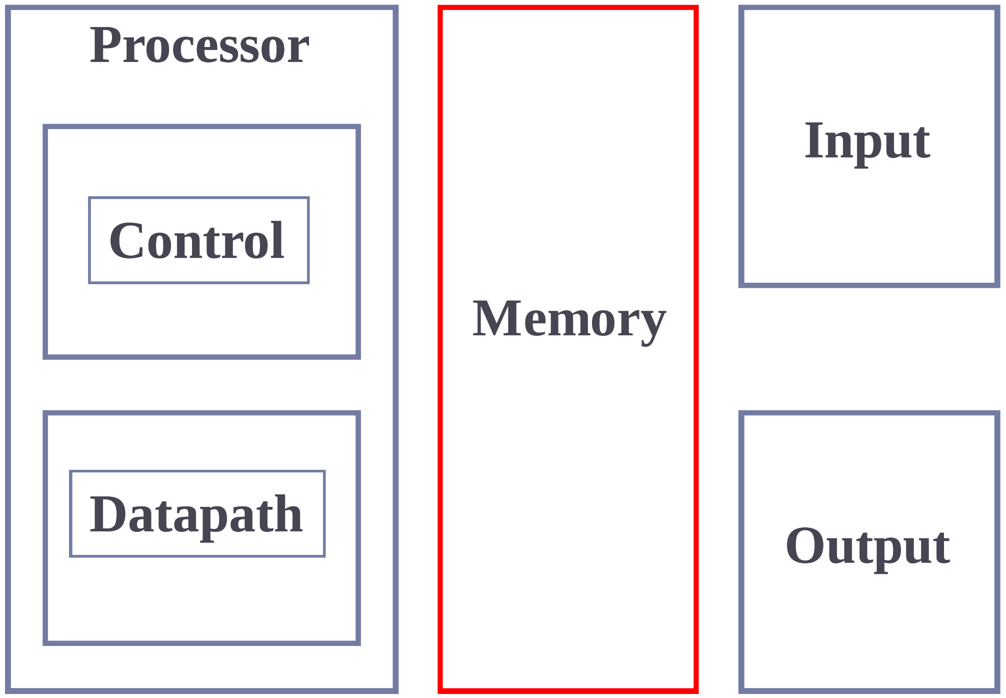
Computer Architecture

Memory Organization

Components



Reality!

- Memory is the required for storing
  - Program
  - Data
- Programmers want unlimited amounts of memory with low latency
- Different memory types
  - Static RAMs
  - Dynamic RAMs
  - Magnetic disk
- How to make memory Bigger, Faster, and Cheaper?

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

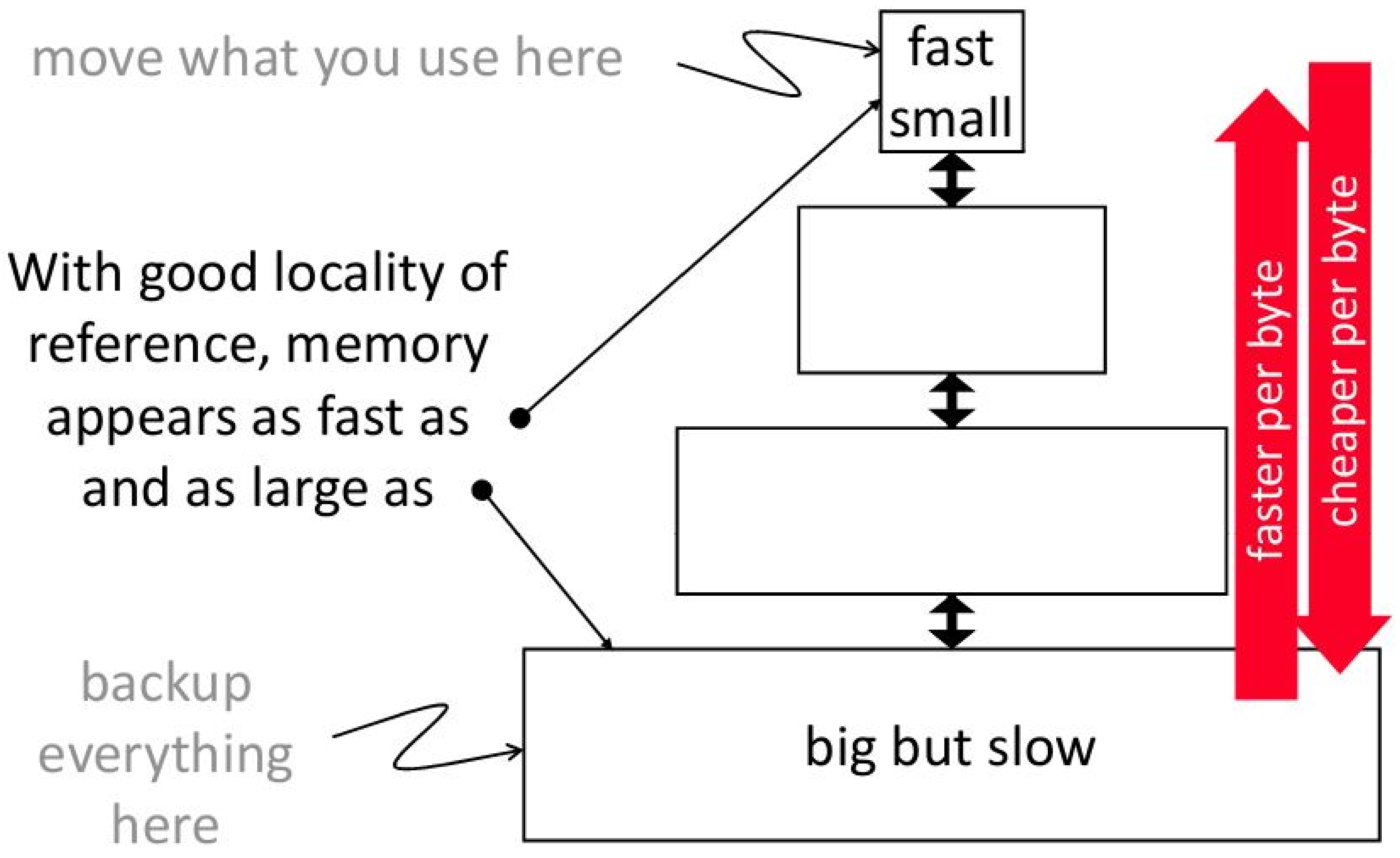
How does hardware exploit principle of locality?

- Organize memory system into a hierarchy
  - Multiple levels of memory with different speeds and sizes
  - Entire addressable memory space available in largest, slowest memory
  - Incrementally smaller and faster memories, each containing a subset of the memory below it, proceed in steps up toward the processor

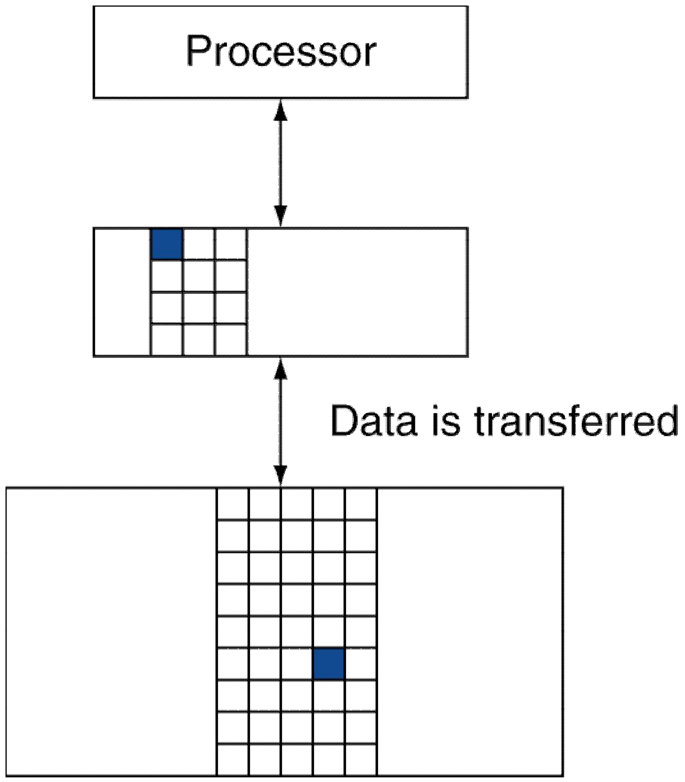
How does hardware exploit principle of locality

- Store everything on disk
- Copy recently accessed (and nearby) items from disk to smaller DRAM memory
  - Main memory
- Copy more recently accessed (and nearby) items from DRAM to smaller SRAM memory
  - Cache memory

Memory Hierarchy



Memory Hierarchy: Terminology

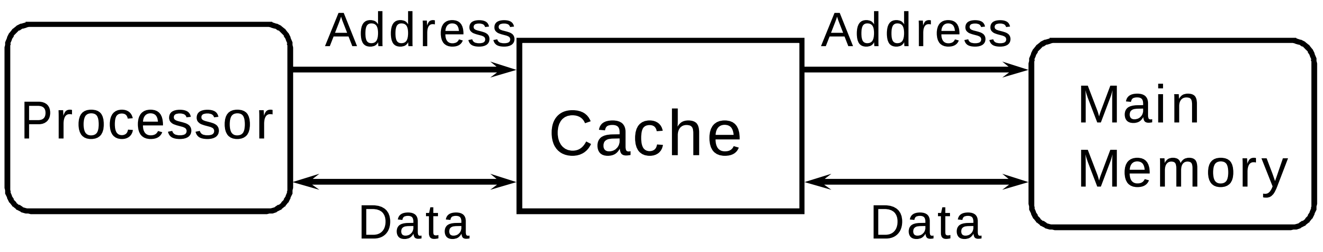


- Block (or line)
  - The minimum unit of information that can be either present or not present in the memory
- If accessed data is present in upper level
  - Hit: access satisfied by upper level
- If accessed data is absent
  - Miss: block copied from lower level

Memory Hierarchy: Terminology

- Hit: data appears in some block in the upper level
  - Hit Rate/Hit Ratio: hits/accesses, i.e., the fraction of memory access found in the upper level
  - Hit Time: Time to access the upper level which consists of Access time + Time to determine hit/miss
- Miss: data needs to retrieve from a block in the lower level
  - Miss Rate/ Miss Ratio = misses/accesses = 1 - (Hit Rate)
  - Miss Penalty: Time to replace a block in the upper level + Time to deliver the block to the processor
- Hit Time << Miss Penalty

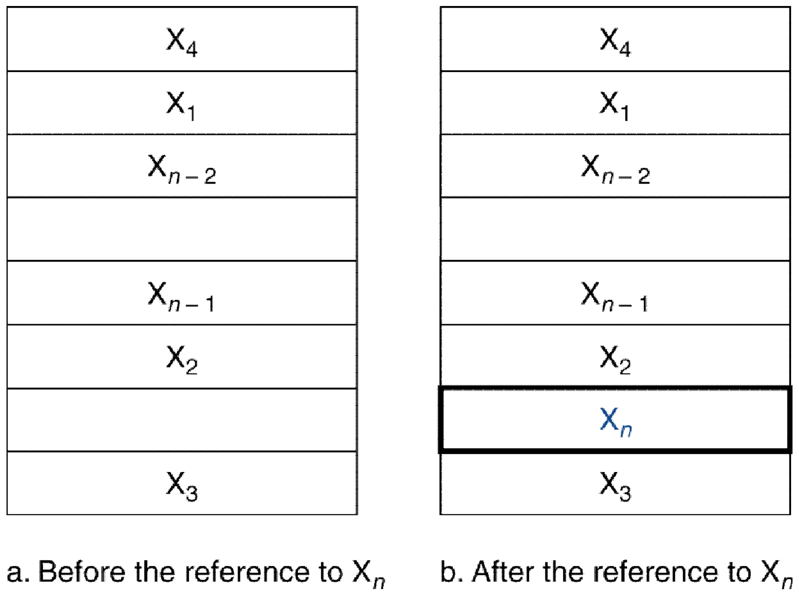
Cache



- Cache memory
  - The level of the memory hierarchy closest to the CPU based on SRAM
  - Memorize the most frequently accessed DRAM memory locations to avoid repeatedly paying for the DRAM access latency

Cache

- Which information goes into the cache?
  - Access to word X_n
  - Create a miss
  - Load X_n into the cache

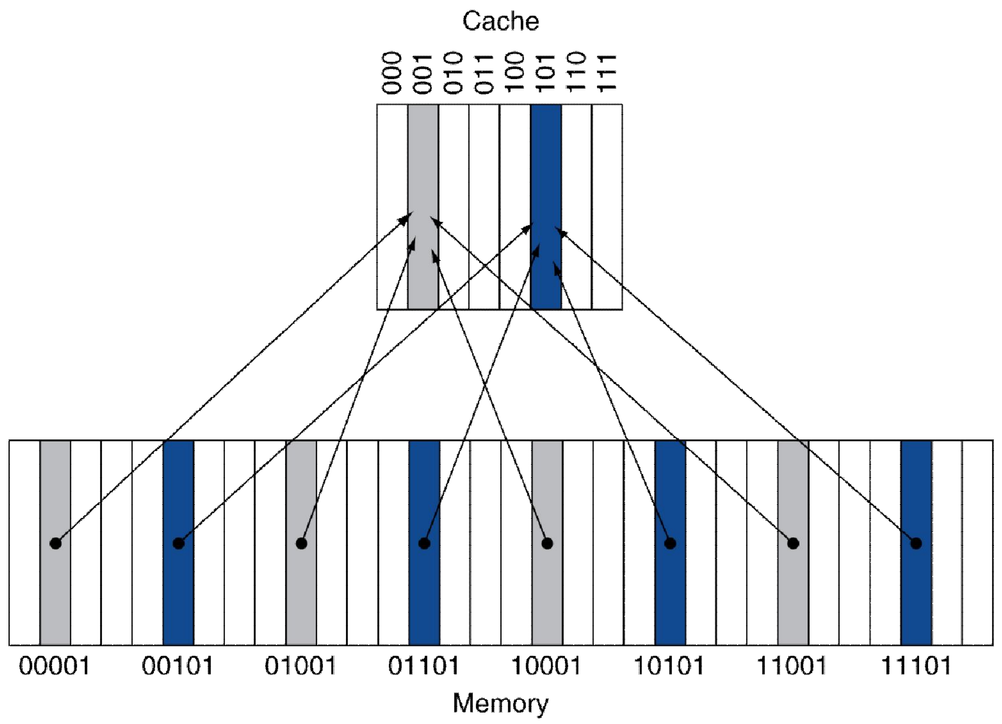


Cache

- How do we know if the data is present?
- Where do we look?
- Simple approach: Direct mapping
  - Every memory location can be mapped only to exactly one cache position
- Mapping:
  - (Block address) modulo (number of cache blocks in the cache)

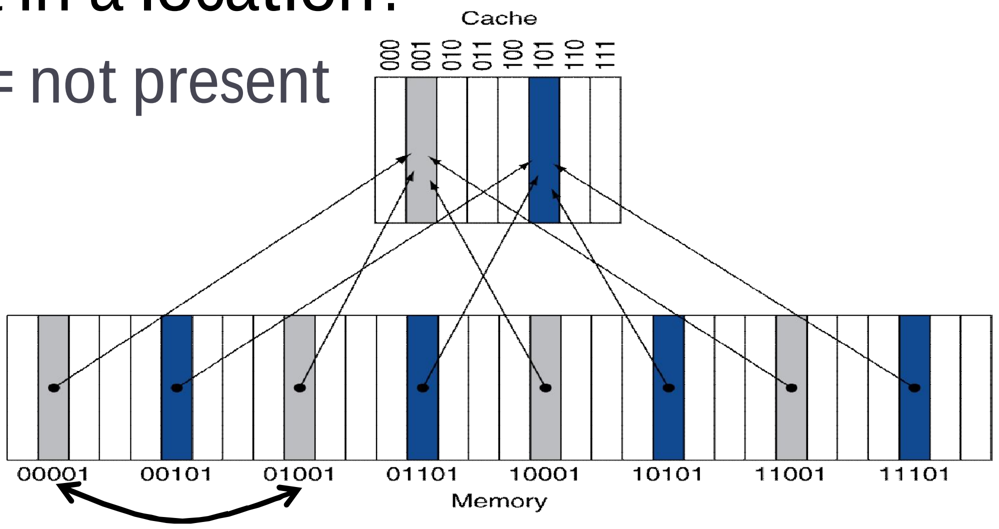
Direct Mapped Cache

- Location determined by address
- Direct mapped: only one choice
  - (Block address) modulo (#Blocks in cache)



Tags and Valid Bits

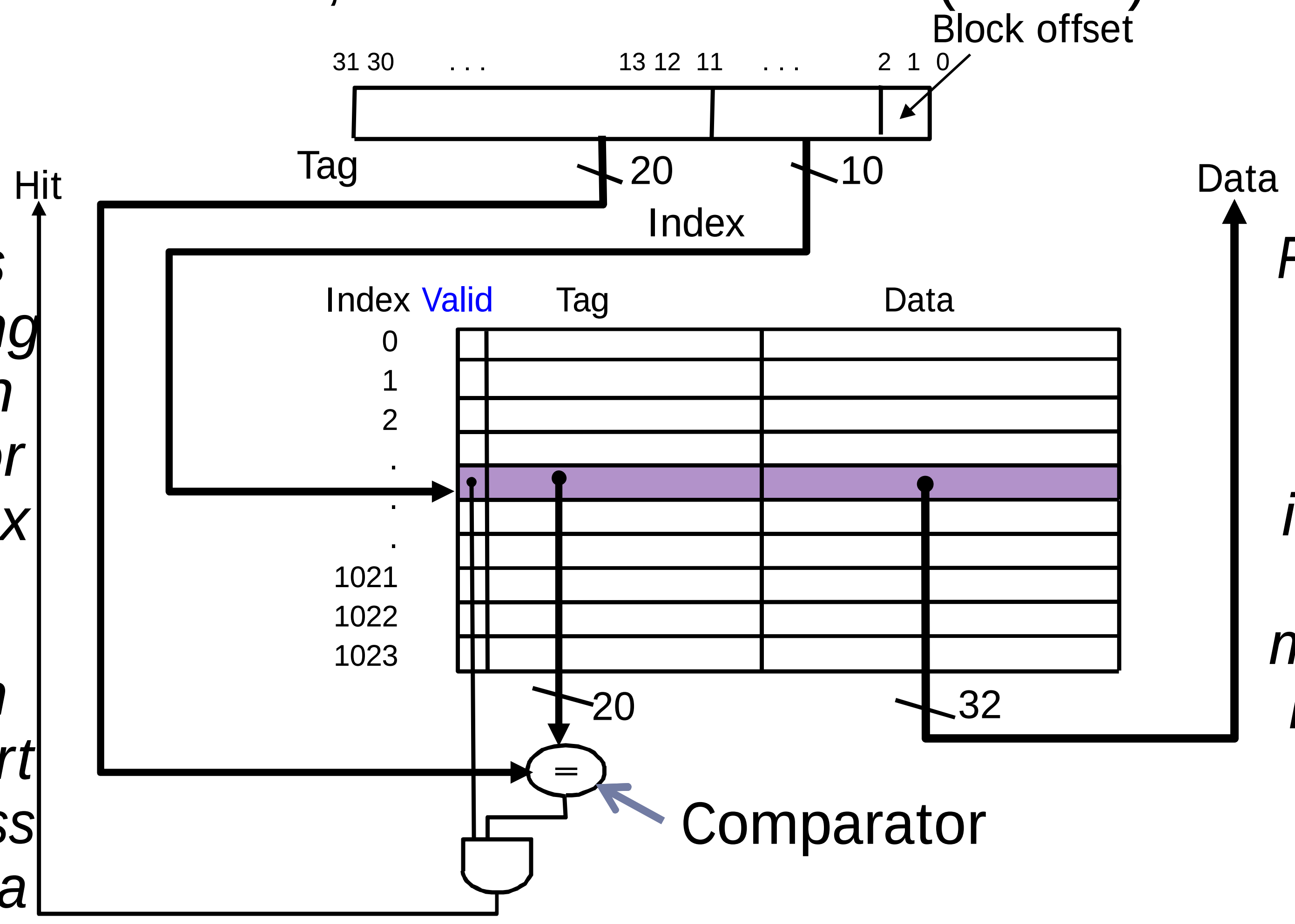
- How do we know which particular block is stored in a cache location?
  - Store block address as well as the data
  - Actually, only need the high-order bits
  - Called the tag
- What if there is no data in a location?
  - Valid bit: 1 = present, 0 = not present
  - Initially 0



Difference?

Direct Mapped Cache Example

- One word blocks, cache size = 1K words (or 4KB)

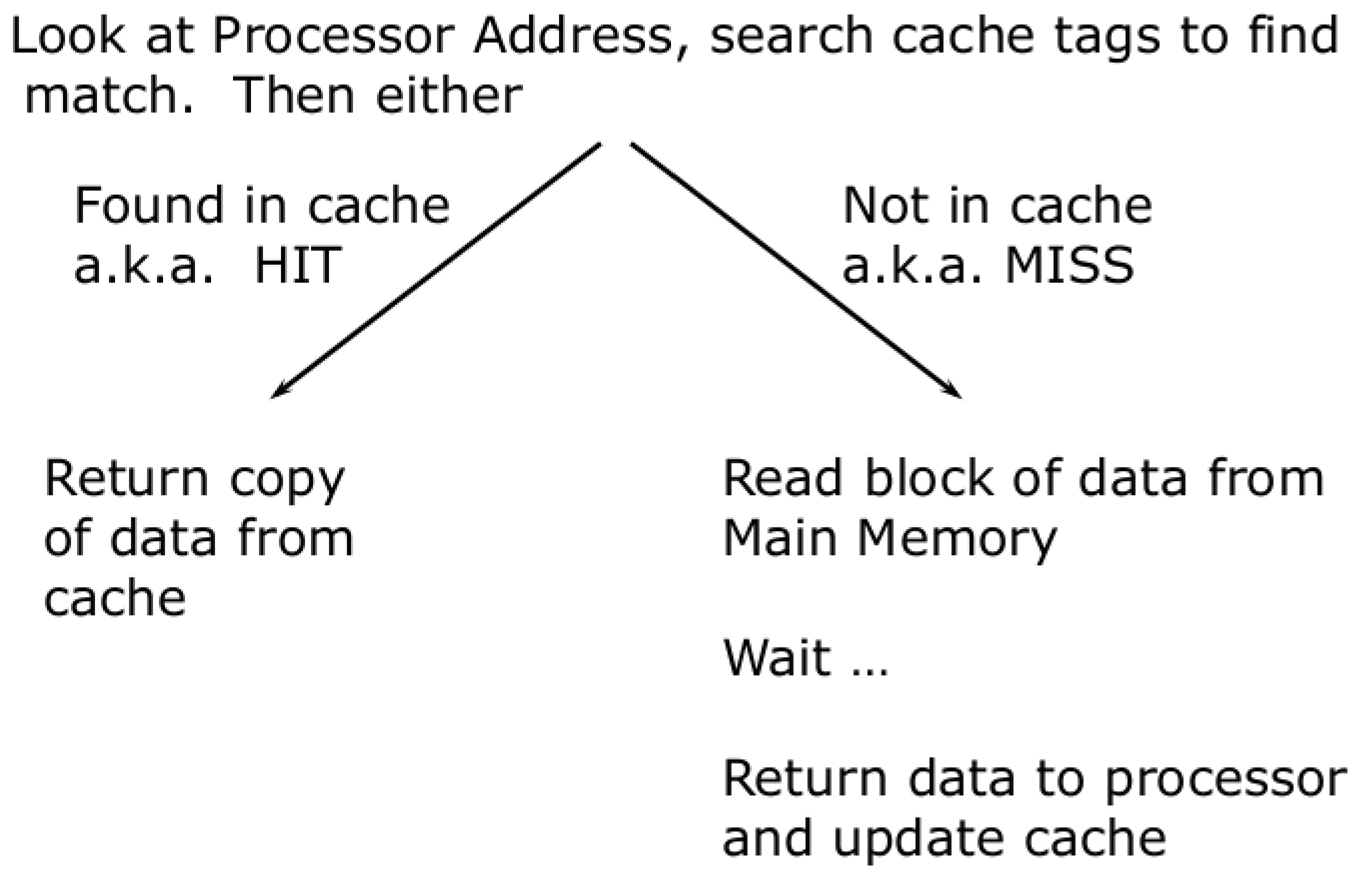


Valid bit ensures something useful in cache for this index

Compare Tag with upper part of Address to see if a Hit

Read data from cache instead of memory if a Hit

Cache: Read Algorithm


Look at Processor Address, search cache tags to find match. Then either

Found in cache
a.k.a. HIT

Return copy
of data from
cache

Not in cache
a.k.a. MISS

Read block of data from
Main Memory

Wait …

Return data to processor
and update cache

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	?	010

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		

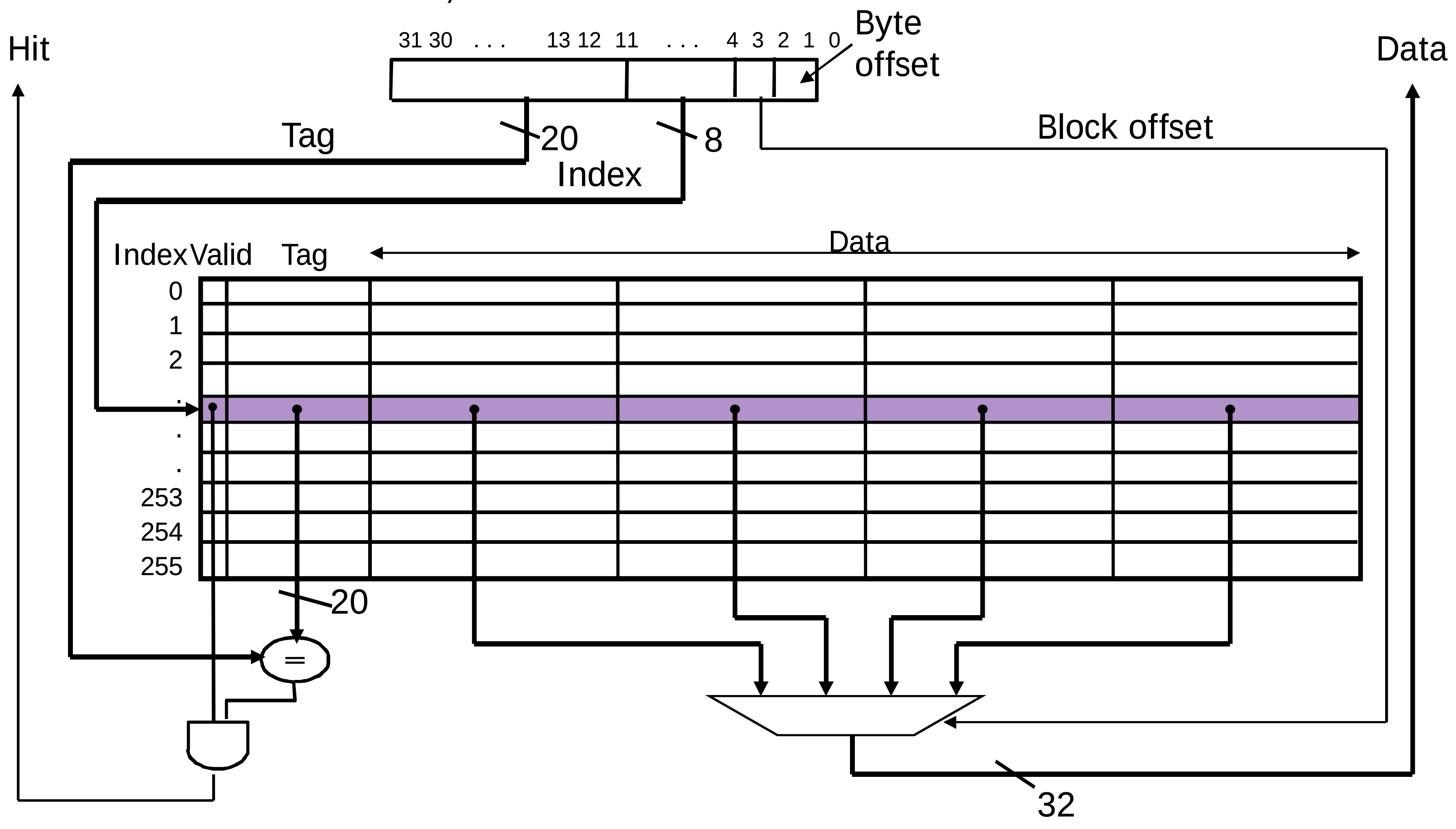
Cache Example

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		

Multiword Block Direct Mapped Cache

- Four words/block, cache size = 1K words



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 miss penalty
  - Can override benefit of reduced miss rate

Cache Size

- 32-bit byte address
- A direct-mapped cache
- The cache size is 2^n blocks
- The block size is 2^m words (2^{m+2} bytes)

- What is the size of the tag field?
  - $32-(n+m+2)$

n	select cache block
m	select words within block
2	select byte within word

Cache Size

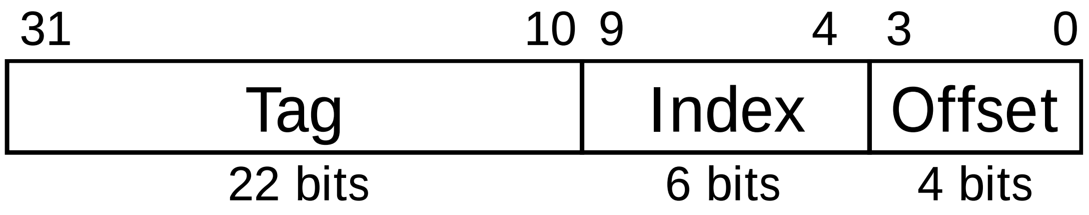
- 32-bit byte address
- A direct-mapped cache
- The cache size is 2^n blocks
- The block size is 2^m words (2^{m+2} bytes)

- What is the size of the cache (unit: no. of bit)?
  - 2^n X $(2^m$ X $32 + 32-(n+m+2) +1)$

Mapping an Address to a Multiword Cache Block

- 64 blocks, 16 bytes/block
- To what block number does address 1200 map?

Block address = $\lfloor 1200/16 \rfloor$ = 75
Block number = 75 modulo 64 = 11



Cache Example

- 32-bit byte address
- A direct-mapped cache

Tag	Index	Offset
31-10	9-4	3-0

- What is the cache line size (in words)?
- **4**
- How many entries does the cache have?
- **64**

Cache Example

- 32-bit byte address
- A direct-mapped cache

Address	0	4	16	132	232	160	1024	30	140	3100	180	2180
Line ID	0	0	1	8	14	10	0	1	9	1	11	8
Hit/miss	M	H	M	M	M	M	M	H	H	M	M	M

- What is the hit ratio?
- **0.25**

Handling Cache Misses

- Read misses
  - Stall execution, fetch the block from the next level in the memory hierarchy, install it in the cache, send requested word to processor, and then let execution resume
- Write misses
  - Write allocate: Stall execution, fetch the block from next level in the memory hierarchy, install it in cache, write the word from processor to cache, also update memory, then let execution resume

  or

  - No-write allocate: skip the cache write and just write the word to memory

Cache-Memory Consistency? (1/2)

- Need to make sure cache and memory have same values on writes: 2 policies

1) Write-Through Policy: write cache and write *through* the cache to memory
   - Every write eventually gets to memory
   - Too slow, so include Write Buffer to allow processor to continue once data in Buffer, Buffer updates memory in parallel to processor

Cache-Memory Consistency? (2/2)

- Need to make sure cache and memory have same values on writes: 2 policies

2) Write-Back Policy: write only to cache and then write cache block *back* to memory when evict block from cache

  - Writes collected in cache, only single write to memory per block
  - Include bit to see if wrote to block or not, and then only write back if bit is set
    - Called "Dirty" bit (writing makes it "dirty")

Write Through vs. Write Back

- Advantages
  - Write through
    - Misses are simpler and cheaper because they never require a block to be written back in the lower level
    - Easier to implement
  - Write Back
    - Individual words can be written by the processor at the rate that the cache, rather than the memory, can accept them
    - Multiple writes within a block require only one write to the lower level in the hierarchy
    - When blocks are written back, the system can make effective use of a high bandwidth transfer, since the entire block is written

Measuring Cache Performance

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

  - CPU time = (CPU execution clock cycles + Memory-stall clock cycles) X Clock cycle time

  - Memory-stall cycles = memory accesses/program × miss rate × 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

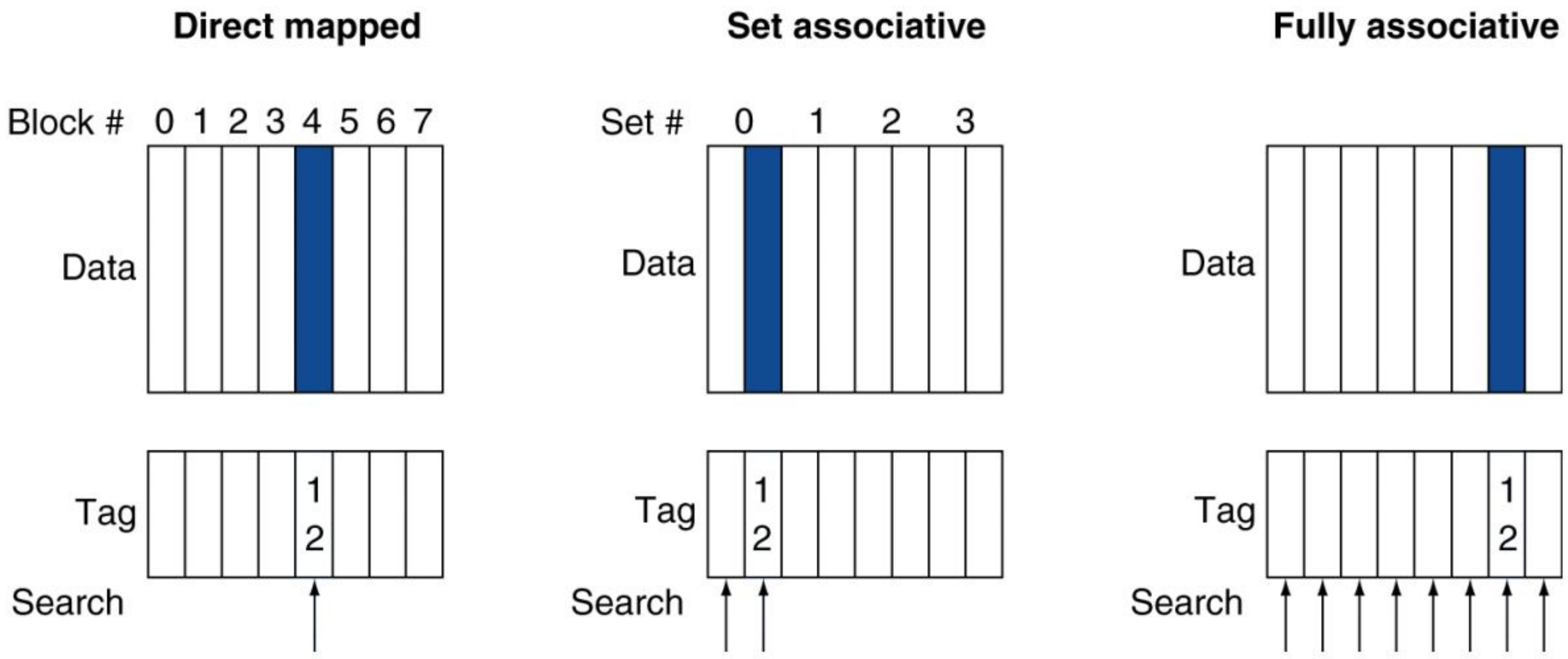
Improving Cache Performance

- Average memory access time (AMAT)
  - AMAT = Hit time + Miss rate × Miss penalty
- To improve performance:
  - reduce the hit time
  - reduce the miss rate (e.g., larger, better policy)
  - reduce the miss penalty (e.g., L2 cache)

Associative Caches

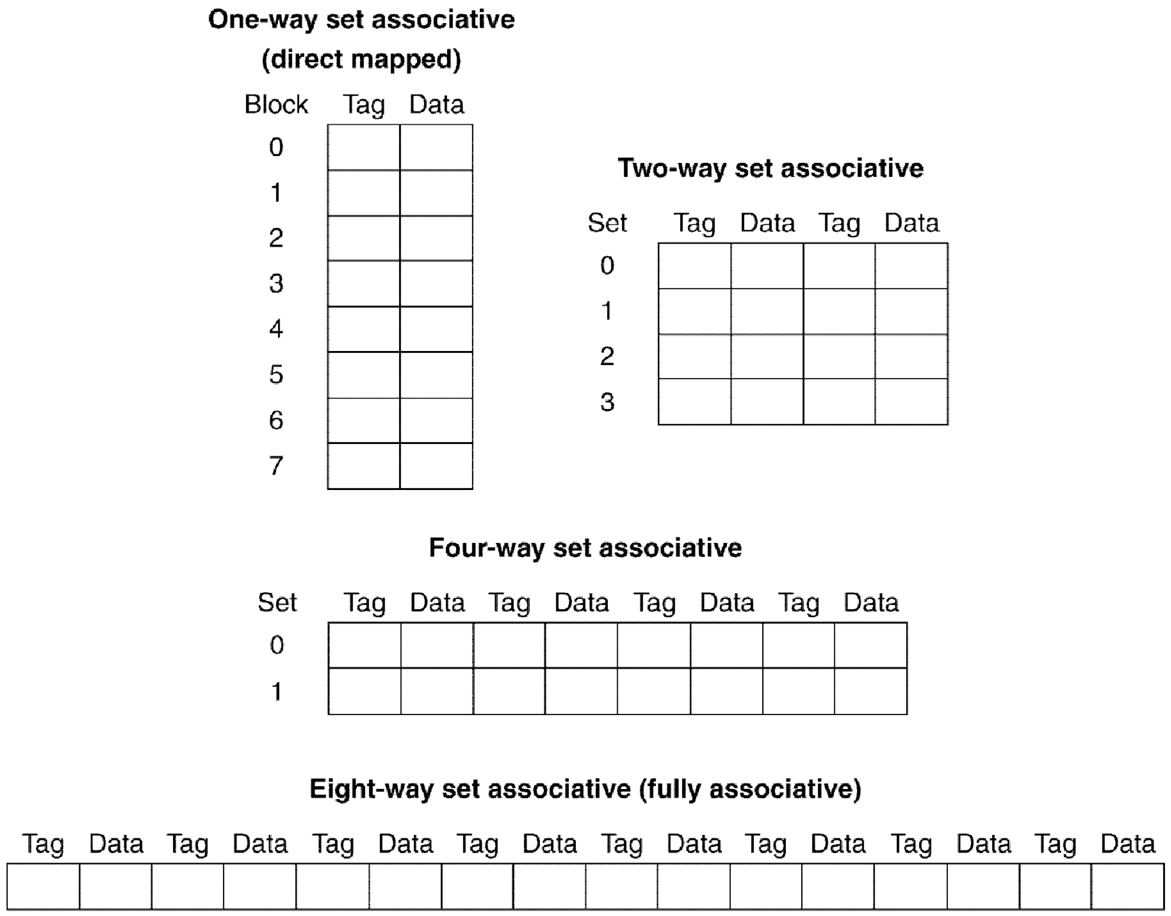
- Fully associative
  - Allow a given block to go in any cache entry
  - Requires all entries to be searched at once
  - Comparator per entry (expensive)
- *n*-way set associative
  - Each set contains *n* entries
  - Block number determines which set
    - (Block number) modulo (#Sets in cache)
  - Search all entries in a given set at once
  - *n* comparators (less expensive)

Cache Example



Spectrum of Associativity

- For a cache with 8 entries


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

Associativity Example

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

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

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

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]**		

Associativity Example

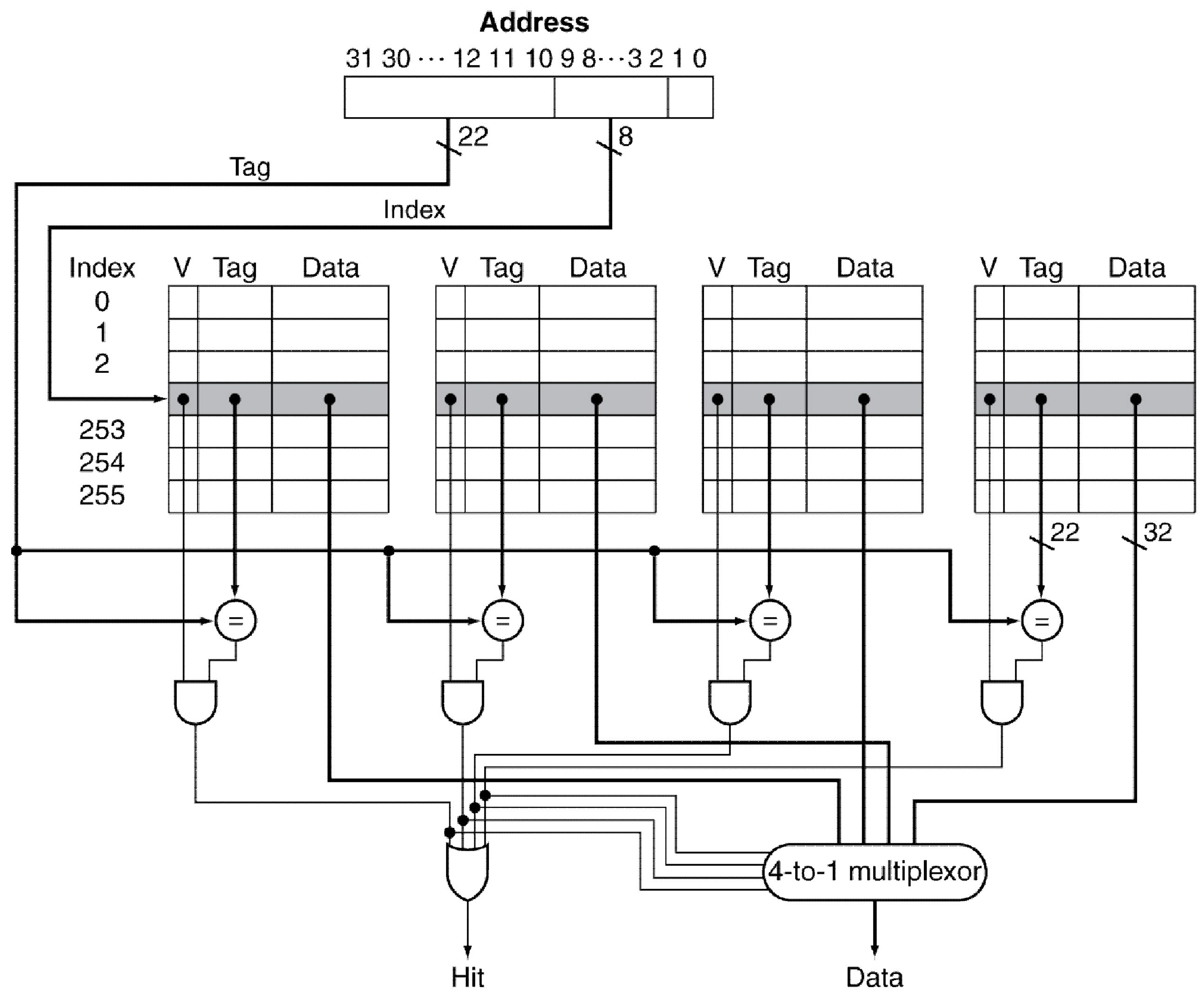
- Compare 4-block caches
  - Direct mapped, 2-way set associative, fully associative
  - Block access sequence: 0, 8, 0, 6, 8
- Fully associative

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
  - But with diminishing returns
- 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



Summary: Placement Policy

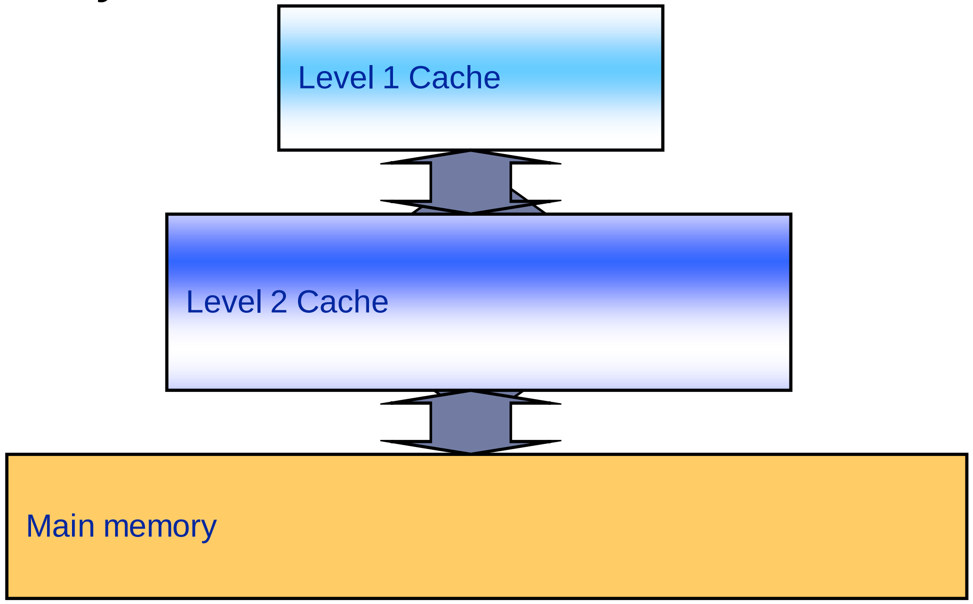
- Direct Mapped
  - No choice
- Set Associative
  - Any location in the set of lines
  - Replacement policy
- Fully Associative
  - Any line in the cache
  - Dictated by the replacement policy

Replacement Policy

- Direct mapped: no choice
- Set associative
  - Prefer non-valid entry, if there is one
  - Otherwise, choose among entries in the set
- Least-recently used (LRU)
  - Choose the one unused for the longest time
  - Simple for 2-way, manageable for 4-way, too hard beyond that
- Random
  - Any block is randomly selected for replacement providing uniform allocation
  - Gives approximately the same performance as LRU for high associativity

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/cycles) = 400 cycles
  - Effective CPI = $1 + 0.02 \times 400 = 9$

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/cycles) = 20 cycles
- Primary miss with L-2 miss
  - Extra penalty = 400 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

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.

Practice: Cache Performance

- Cache Miss Penalty = 50 cycles
- Instructions normally take 2 cycles, ignoring stalls
- Cache Miss rate is 2%
- Average of 0.33 Memory References per instruction
- Impact of cache vs. no cache?
  - CPU time = (CPU execution clock cycles + Memory-stall clock cycles) X Clock cycle time

- CPUTime(cache) = IC * (2 + 0.33 * 0.02 * 50) * Cycle Time

- CPUTime(nocache) = IC * (2 + 0.33 * 50) * Cycle Time

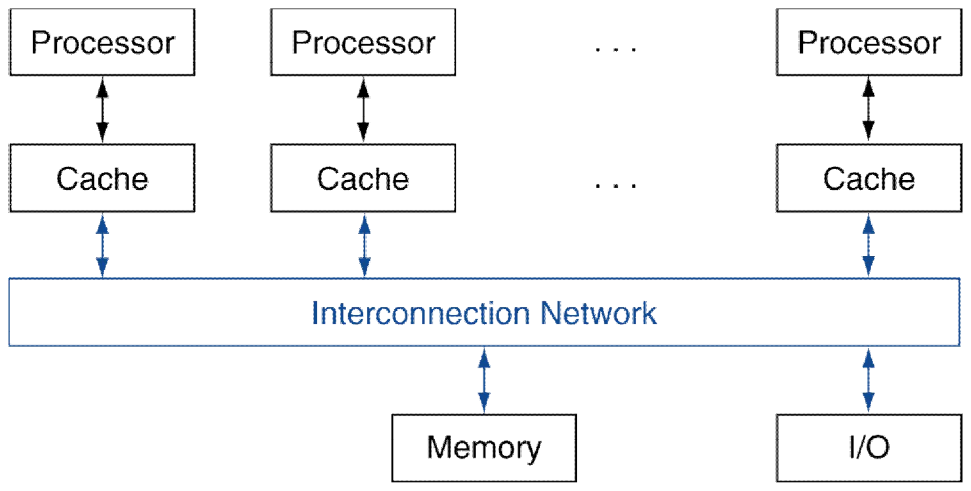
Practice: Cache Performance

- Base CPI 2.0
- Processor speed 3 GHZ
- Main memory access time 125 ns
- Cache miss rate peer instruction 5%
- What is the CPI of the processor?

- Memory miss cycles: 125 ns/(1/3) ns/clock cycle= 375 clock cycle
- Total CPI = 2.0 + 375 × 5% = 20.75

Cache Coherence

- A shared variable may exist in multiple caches
- Multiple copies to improve latency
- This is a really a synchronization problem

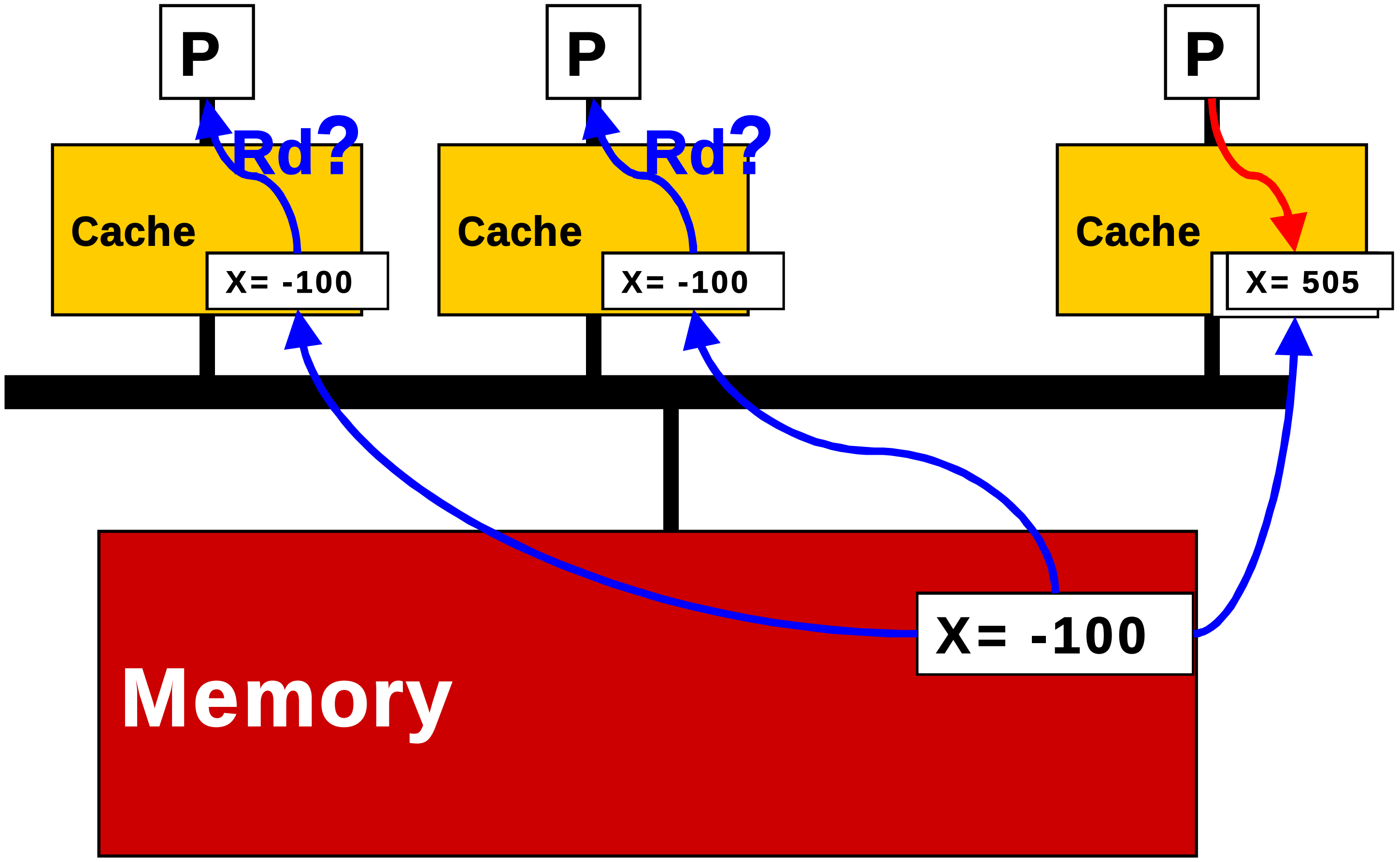


Cache Coherence Problem

- 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

Example (Writeback Cache)



Coherence Defined

- Informally: Reads return most recently written value
- Formally:
  - P writes X; P reads X (no intervening writes)
    $\Rightarrow$ read returns written value
  - P_1 writes X; P_2 reads X (sufficiently later)
    $\Rightarrow$ read returns written value
    - c.f. CPU B reading X after step 3 in example
  - P_1 writes X, P_2 writes X
    $\Rightarrow$ all processors see writes in the same order
    - End up with the same final value for X

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

Reading Materials

- Class Lectures
- Computer Organization and Design (ARM 4th Edition)
  - Sections 5.1 – 5.3, 5.4 (Pages 499-501 are excluded), 5.5, 5.8, 5.12
  - Related Exercise from 5.14

Acknowledgements

- These slides contain material developed and copyright by:
  - Krste Asanovic (UCB) (Course# 152, Spring 2012)
  - James Hoe (CMU) (Course# 18-447, Spring 2011)
  - Li-Shiuan Peh (MIT) (Course# 6.823, Fall 2012)
  - Sudhakar Yalamanchili (GATECH) (Course# ECE3056, Fall 2012)

The End