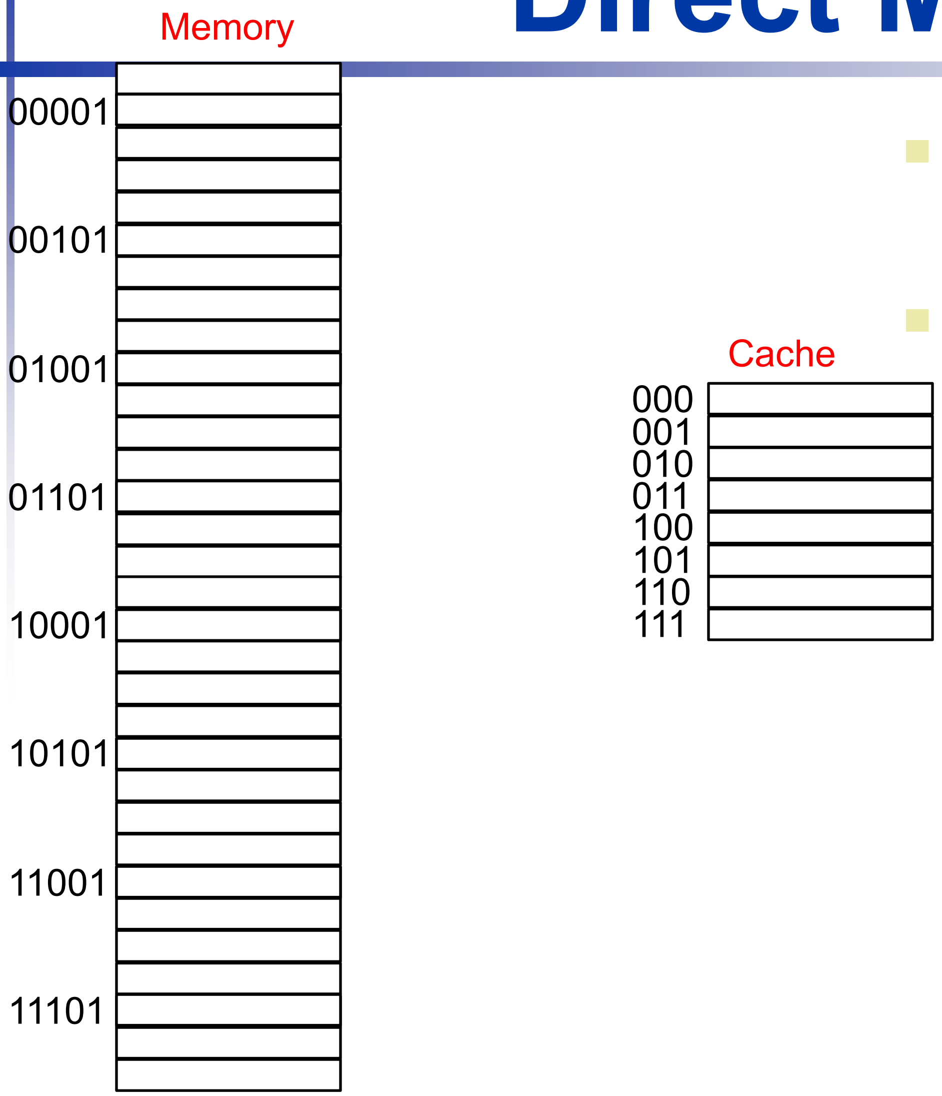
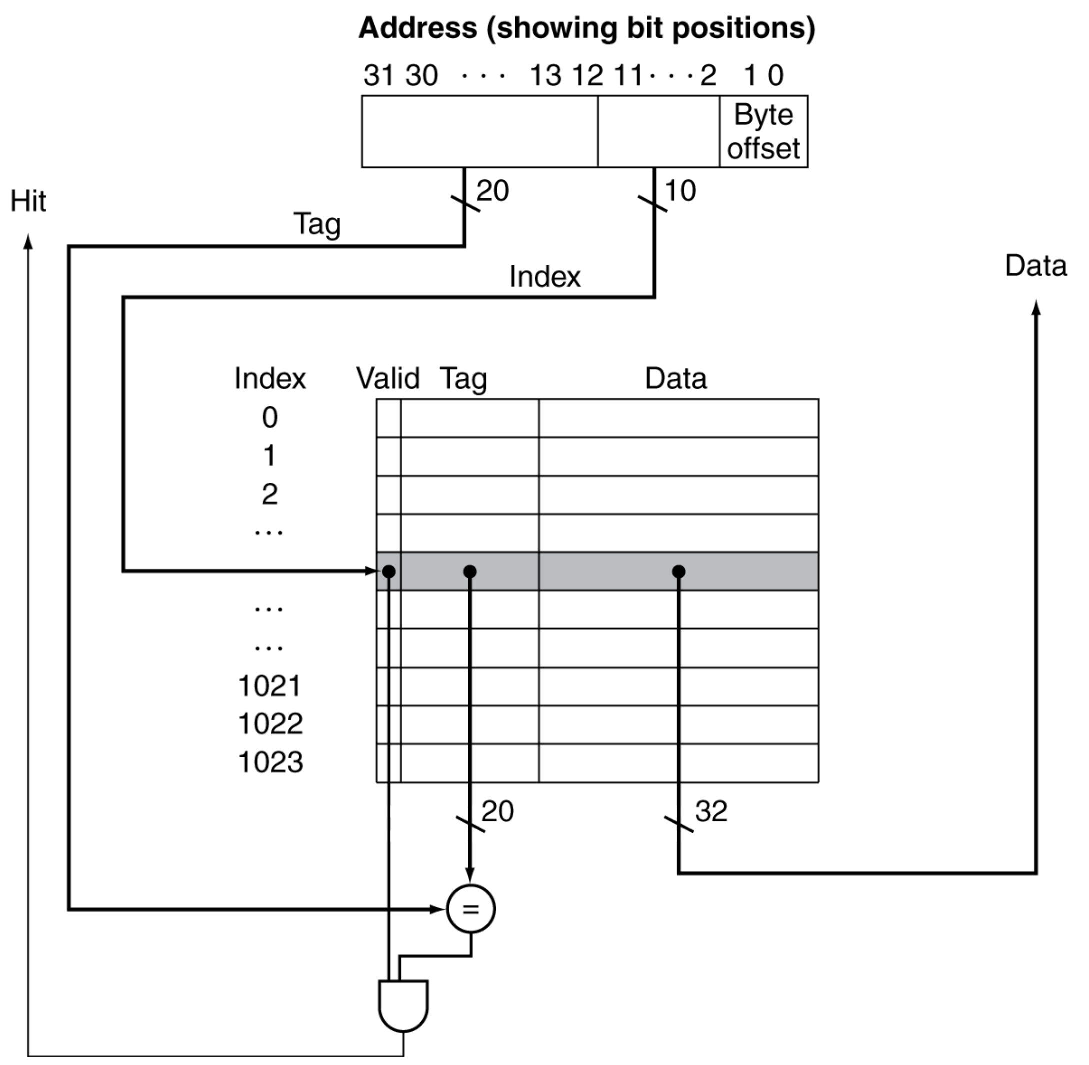
Memory Management

Direct Mapped Cache

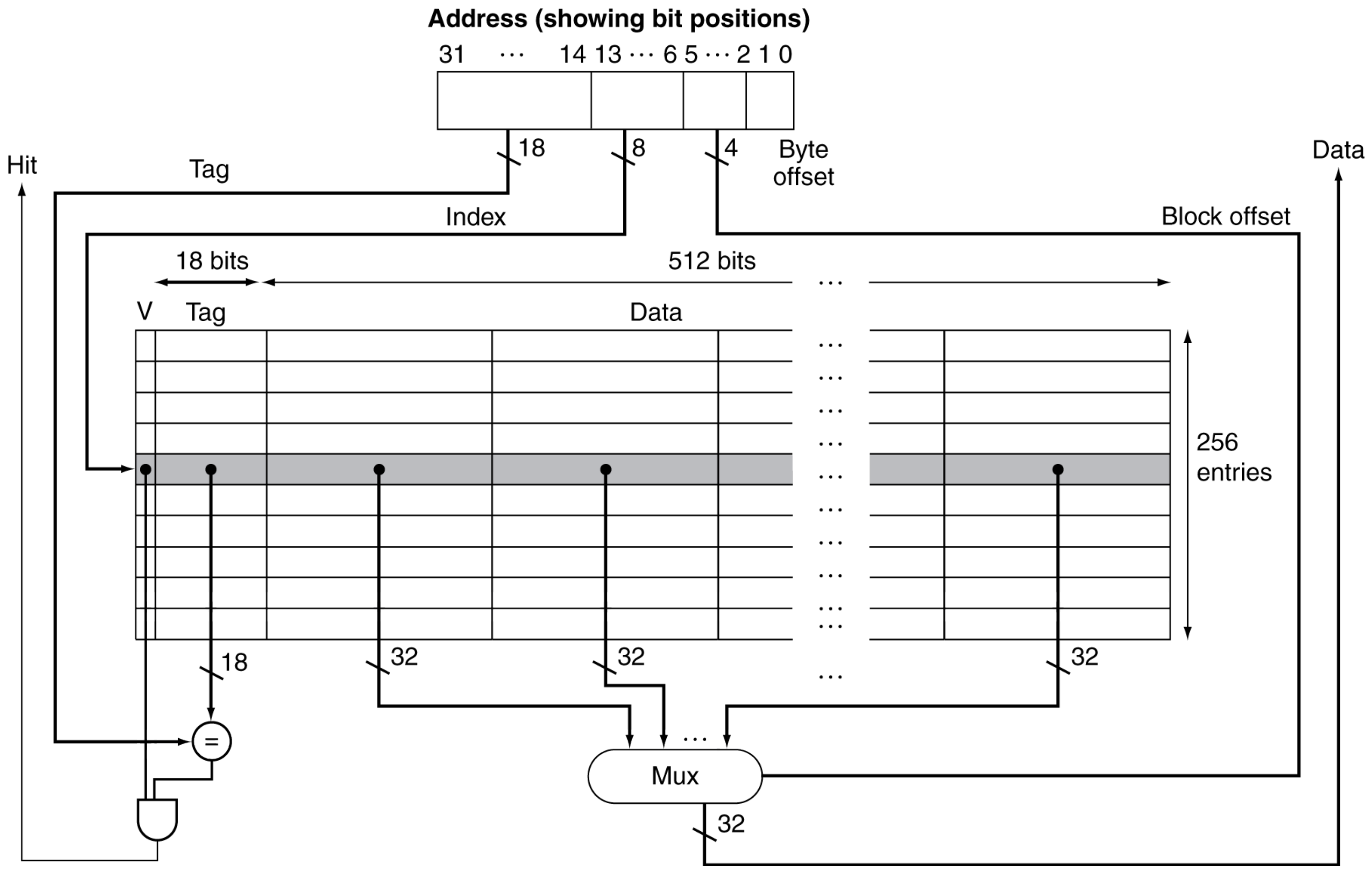


- Location determined by address
- Direct mapped: only one choice
  - (Block address in memory) modulo (#Blocks in cache)
- Use low-order address bits

Address Subdivision



Example: Intrinsity FastMATH

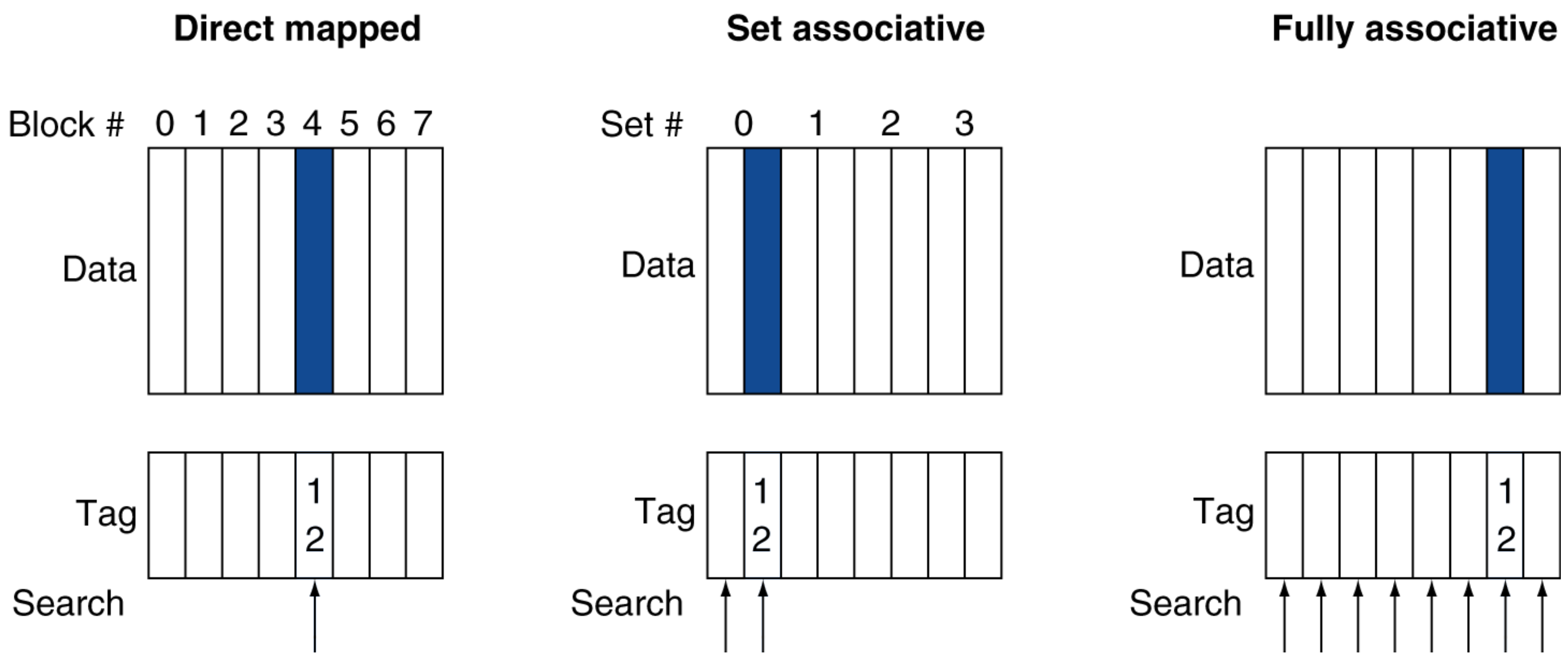


Block Size Tradeoffs

- Larger block sizes
  - Take advantage of spatial locality
  - Incur larger miss penalty since it takes longer to transfer the block into the cache
  - Can increase the average hit time and miss rate

Associative Cache Example

- Cache index for (memory) block address: 12

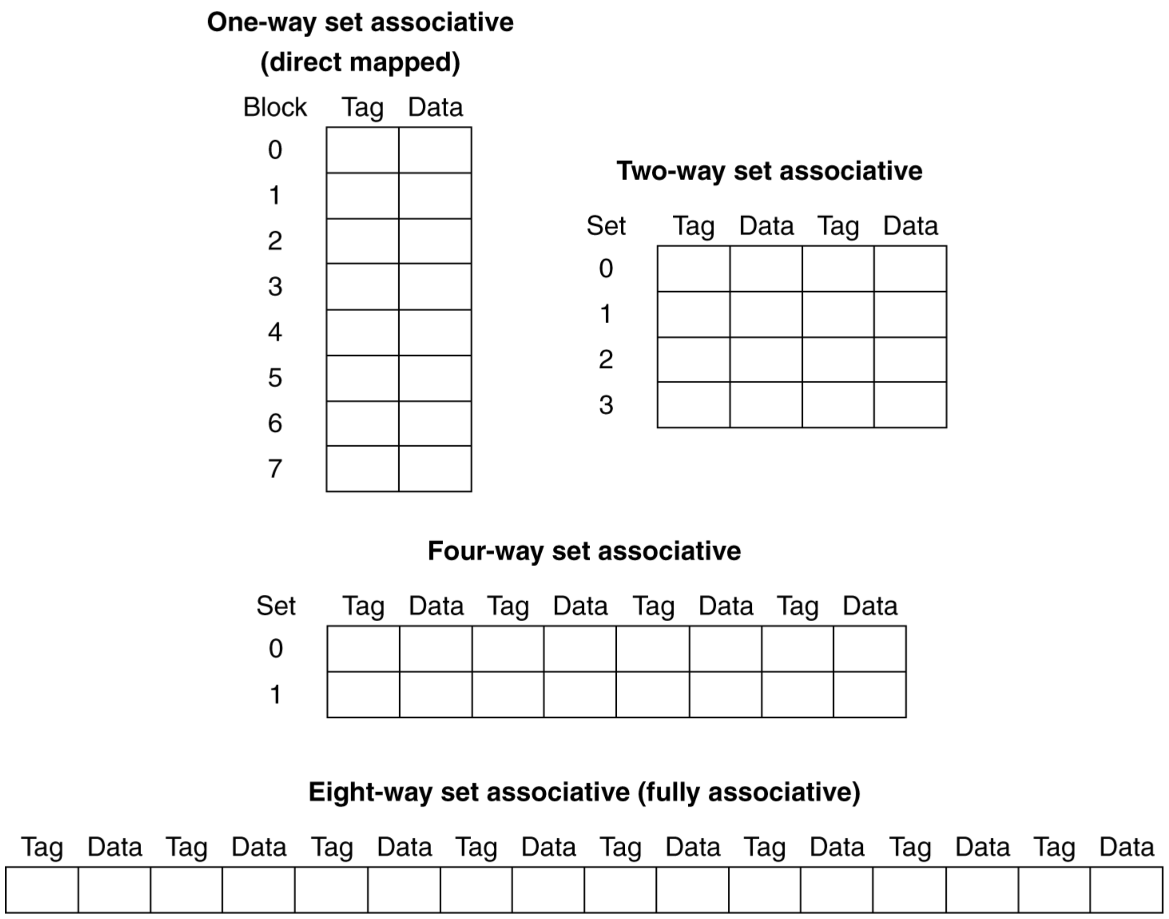


Reducing Cache Miss: #1 Associative Caches

- Direct mapped cache
  - A memory block maps to exactly one cache block
- 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 address determines which set
    - (Block address) modulo (#Sets in cache)
  - Search all entries in a given set at once
  - *n* comparators (less expensive)

Spectrum of Associativity

- For a cache with 8 entries



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						
8						
0						
6						
8						

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					
8	0					
0	0					
6	2					
8	0					

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					
6	2					
8	0					

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					
8	0					

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					

Associativity Example

- Compare 4-block caches
  - Direct mapped, 2-way set associative, fully associative
  - Block address 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

- 2-way set associative

Block address	Cache index	Hit/miss	Cache content after access			
			Set 0		Set 1	
0						
8						
0						
6						
8						

Associativity Example

- 2-way set associative

Block address	Cache index	Hit/miss	Cache content after access			
			Set 0		Set 1	
0	0					
8	0					
0	0					
6	0					
8	0					

Associativity Example

- 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					
0	0					
6	0					
8	0					

Associativity Example

- 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					
6	0					
8	0					

Associativity Example

- 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					
8	0					

Associativity Example

- 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

- Fully associative

Block address		Hit/miss	Cache content after access			
0						
8						
0						
6						
8						

Associativity Example

- Fully associative

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

Associativity Example

- Fully associative

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

Associativity Example

- 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						
8						

Associativity Example

- 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						

Associativity Example

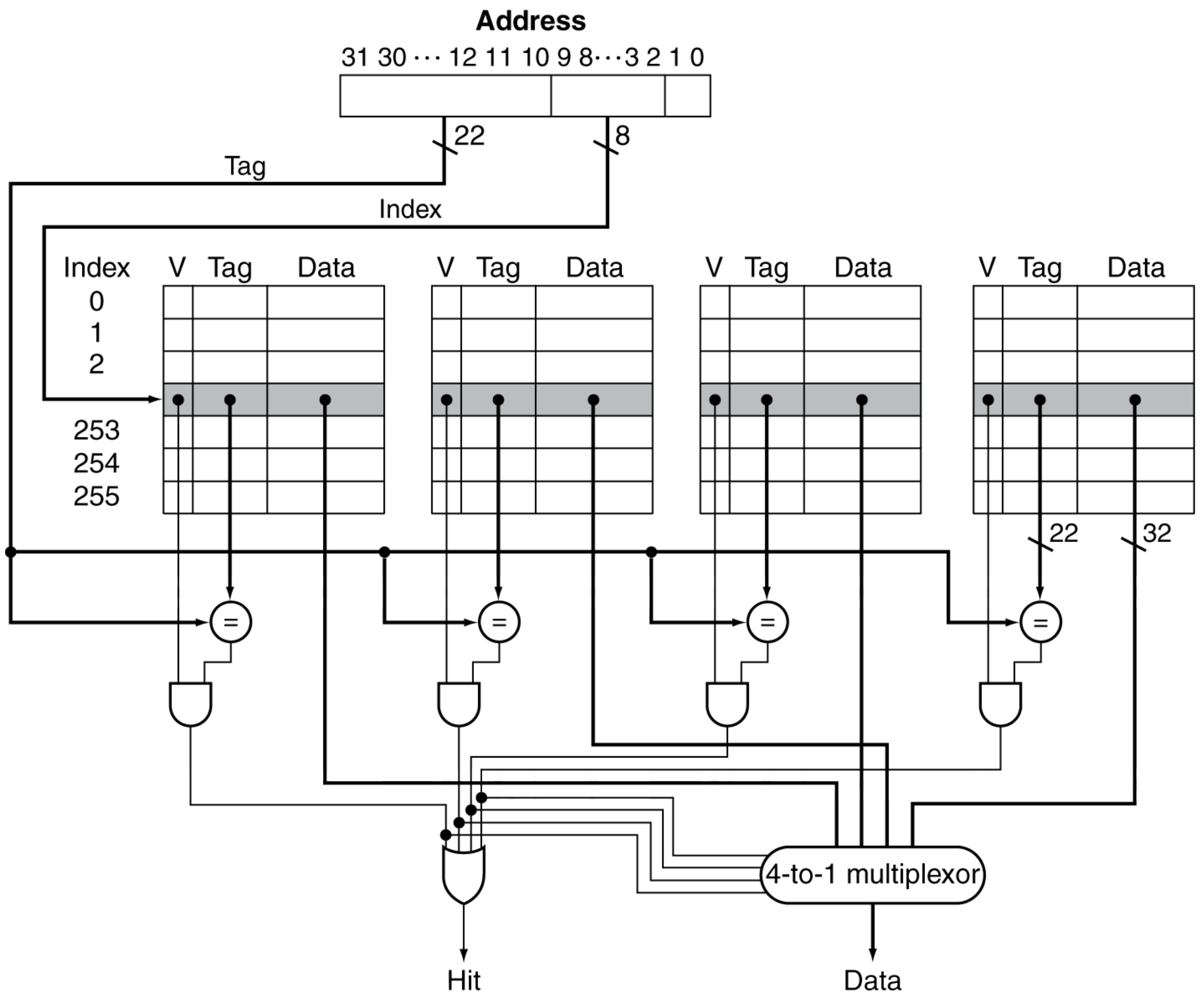
- Fully associative

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

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

Set Associative Cache Organization



Associativity Implies Choices

- Direct-mapped
  - Compare addr. with only one tag
  - Location A can be stored in exactly one cache line
- N-way set-associative
  - Compare addr. with N tags simultaneously
  - Location A can be stored in exactly one set, but in any of the N cache lines belonging to that set
- Fully associative
  - Compare addr. With each tag simultaneously
  - Location A can be stored in any cache line

Replacement Policy

- Direct mapped: no choice

- Set associative
  - Prefer non-valid entry, if there is one
  - Otherwise, choose among entries in the set

- Optimal policy: Replace the block that is accessed furthest in the future
  - Requires knowing the future

- Predict the future from looking at the past
  - If a block has not been used recently, it’s often less likely to be accessed in the near future (a locality argument)

- Least-recently used (LRU)
  - Choose the one unused for the longest time

Write Policy

- Write-through:
  - CPU writes are cached, but also written to main memory immediately
  - Stalling the CPU until write is completed
  - Simple, slow
- Write-back:
  - CPU writes are cached, but not written to main memory until we replace the block
  - Commonly implemented in current systems
  - Fast, more complex

```
sum = 0;
for (i = 0; i < n; i++)
    sum += a[i];
return sum;
```

Write-Back with ‘Dirty’ Bits

- Add 1 bit per block to record whether block has been written to.
- Only write back dirty blocks

Virtual Memory

- Software caches:
- Same objective: fake large, fast, and cheap memory
- Conceptually similar
- Different implementations

Virtual Memory

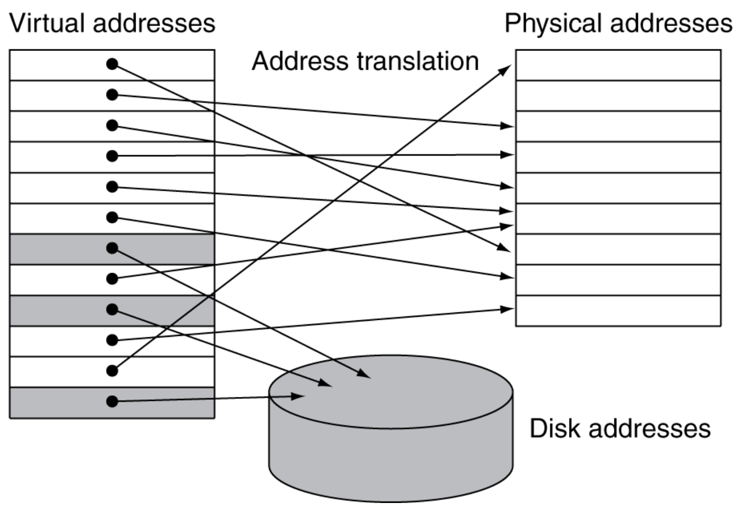
- Use main memory as a "cache" for secondary (disk) storage
  - Managed jointly by CPU hardware and the operating system (OS)
- Programs share main memory
  - Each gets a private virtual address space holding its frequently used code and data
  - Protected from other programs
- CPU and OS translate virtual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

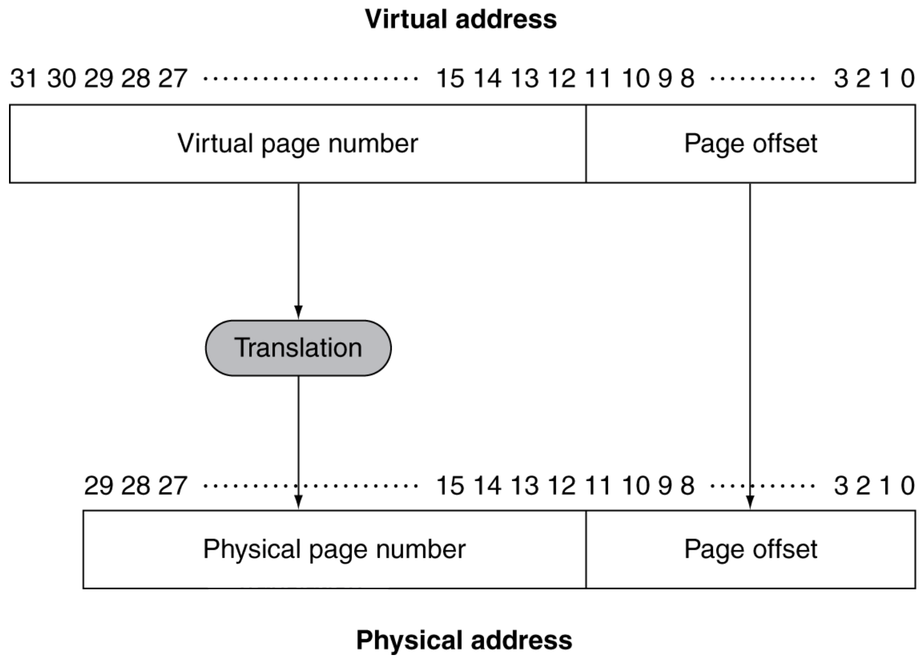
Virtual Memory

- Two kinds of addresses:
  - CPU (also programs) uses virtual addresses
  - Main memory uses physical addresses
- Hardware translates virtual addresses to physical addresses via an operating system(OS)-managed table, the page map.

Address Translation

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





Virtual Memory: Paging

- Divide physical memory in fixed-sized blocks, called pages
  - Typical page size: 4~16KB
  - Virtual address: Virtual page number + offset bits
  - Physical address: Physical page number + offset bits
  - Why use lower bits as offset?
- MMU maps virtual address to physical address, or cause a page fault (a miss) if no translation
  - Use page map to perform translation

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

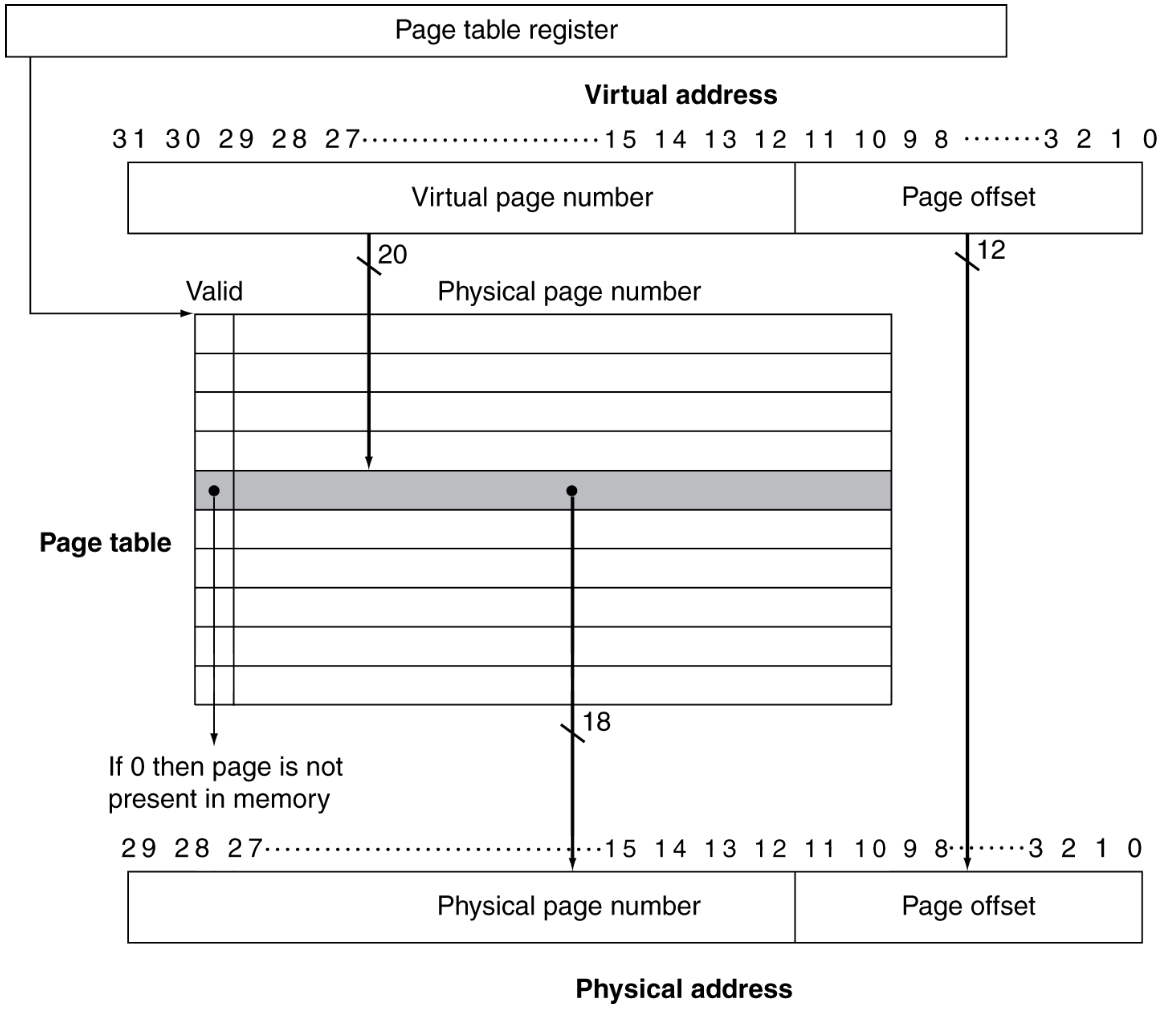
Page Tables

- 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

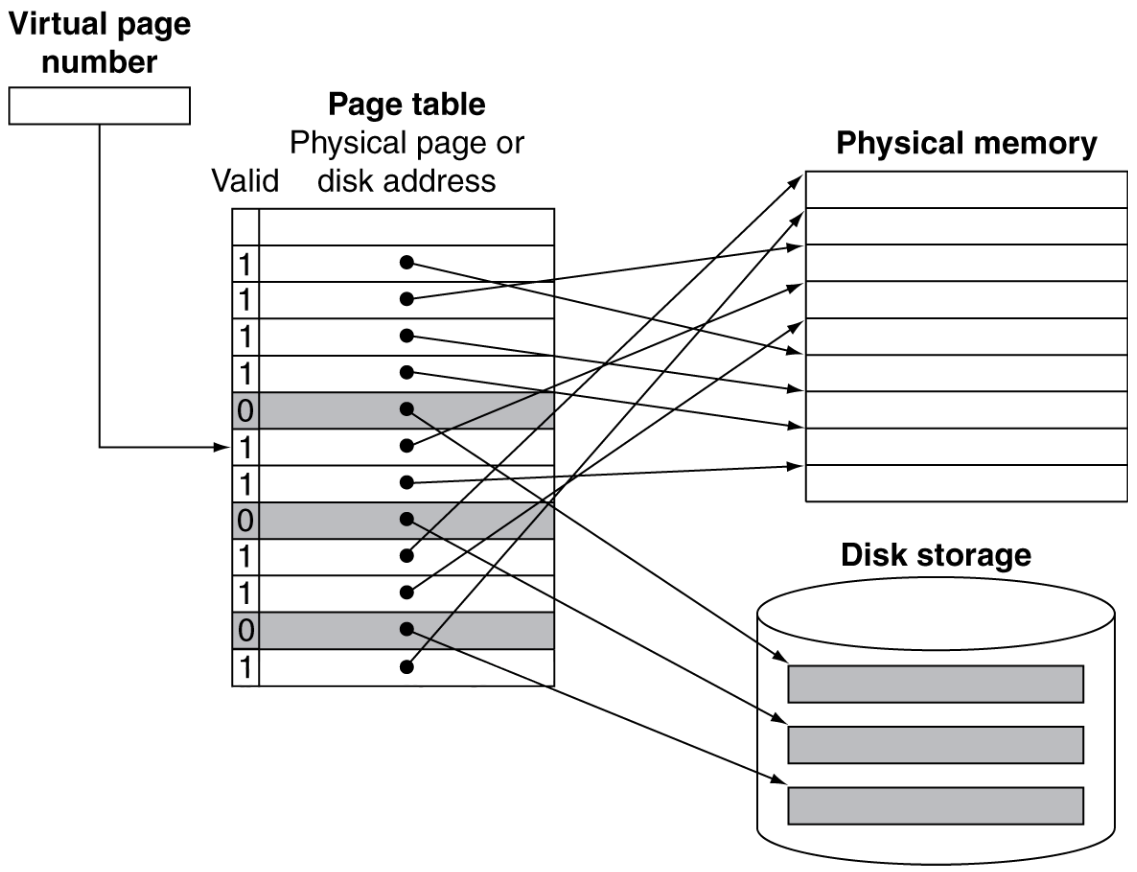
- If page is present in memory
  - PTE stores the physical page number
  - Plus other status bits (referenced, dirty, …)

- If page is not present
  - PTE can refer to location on disk

Translation Using a Page Table



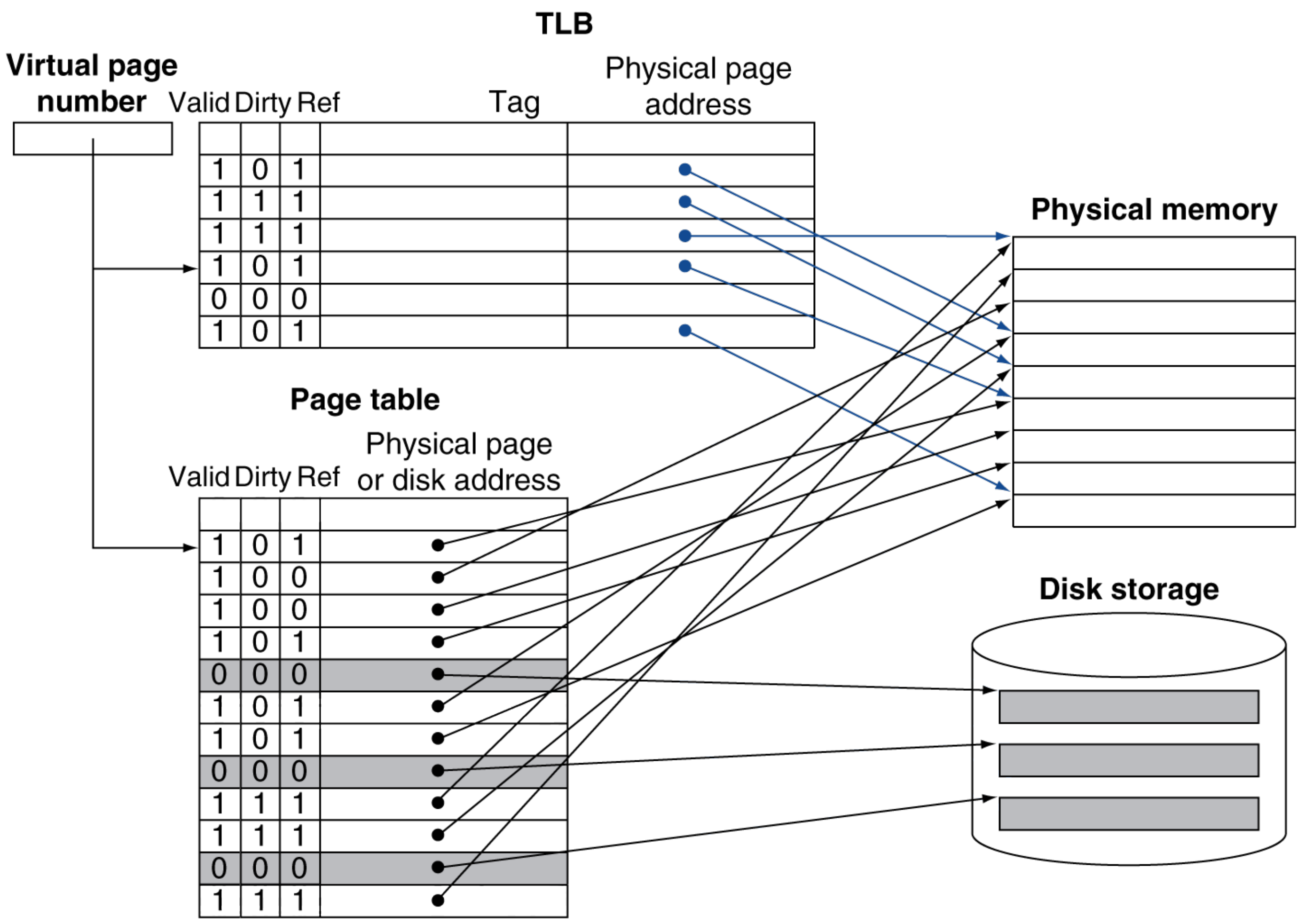
Mapping Pages to Storage



Replacement and Writes

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

Fast Translation Using a TLB



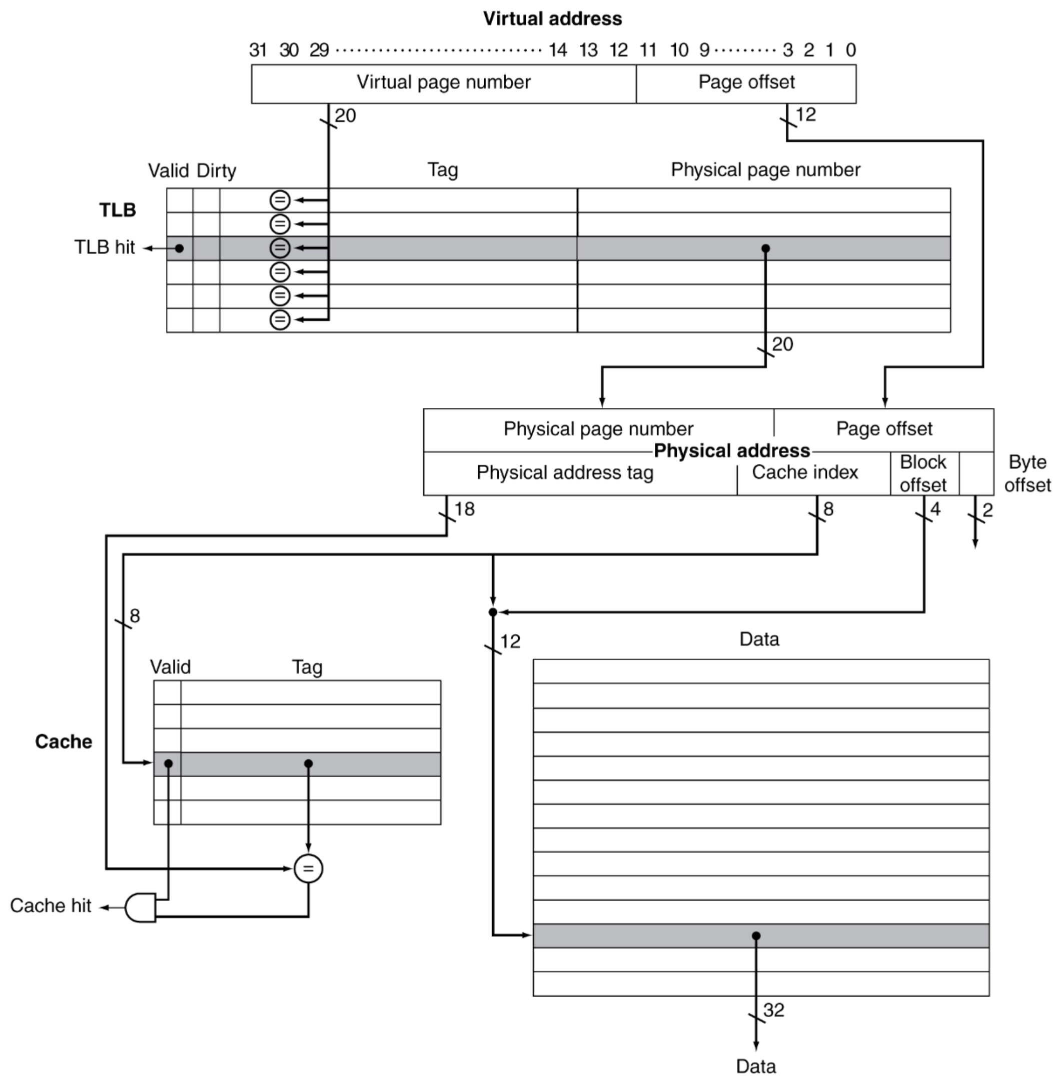
TLB Miss Handler

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

TLB and Cache Interaction



Virtual address
31 30 29 ··· 14 13 12 11 10 9 ··· 3 2 1 0
Virtual page number
Page offset
20
12
Valid Dirty
Tag
Physical page number
TLB
TLB hit
=
20
Physical page number
Page offset
Physical address
Physical address tag
Cache index
Block offset
Byte offset
18
8
4
2
8
12
Valid
Tag
Data
Cache
=
Cache hit
32
Data