Roadmap

- Cache basics
  - Memory hierarchy
  - Locality
  - Direct-mapped cache
    - Index, tag, offset
  - Cache and processor
- Cache performance

Cache & CPU

- 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

Write On a Hit

- Write request and the address is in cache
- Write back: just update the block in cache
  - Cache and memory would be inconsistent
  - When to update memory?
    - Need to mark "dirty" blocks and update memory at replacement
- Write through: update both cache and memory
  - Performance?
  - Write buffer: hold data waiting to be written to memory; CPU continues

Write On a Miss

- What happen on a write miss?
  - Write request but the address is not in cache
- Write allocate
  - Fetch the block and allocate an entry in cache on a write miss
  - Write-back caches usually use this approach
- Write around (no write allocate):
  - Don't fetch the block, just update at the lower level
  - Example: OS zeros a whole page of memory

Write Approaches

	Hit	Miss
Update cache only (until replacement)	Write back	Write allocate
Update memory	Write through	Write around

- Either write miss policy could be used with write-through or write-back
- Usual pairing is as above

Roadmap

- Cache basics
  - Direct-mapped cache
  - Interaction between cache and processor
- Cache performance
  - Evaluation
  - Improvement

Measuring Cache Performance

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

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

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

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
- What is the actual CPI?
  - $CPI_{Actual} = CPI_{Base}$ + average memory stall per instruction

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
- Stall cycles per instruction caused by misses
  - I-cache: 2% * 100 = 2
  - D-cache: 4% * 100 * 36% = 1.44
- Actual CPI = 2 + 2 + 1.44 = 5.44
  - CPU with a perfect memory 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, cache miss rate = 5%
  - AMAT = ?

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, cache miss rate = 5%
  - AMAT = 1ns + 5%*20ns = 2ns

Performance Notes

- When CPU performance improved
  - 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

Types of Cache Misses

- "Three Cs" model of misses
- Compulsory misses
  - Occur when a program is first started
    - Cache does not contain any of that program's data yet, so misses are bound to occur
  - Can't be avoided easily
- Capacity misses
  - Miss that occurs because the cache has a limited size
  - Miss that would not occur if we increase the size of the cache
  - Many compiler techniques to reduce misses of this type by transforming programs

Types of Cache Misses

- Conflict misses
  - Miss that occurs because two distinct memory addresses map to the same cache location
  - Two blocks (which happen to map to the same location) can keep overwriting each other
  - Big problem in direct-mapped caches
  - How do we lessen the effect of these?
- Dealing with conflict misses
  - Solution 1: make the cache size bigger
    - Fails at some point
  - Solution 2: more flexible placement of blocks?

Roadmap

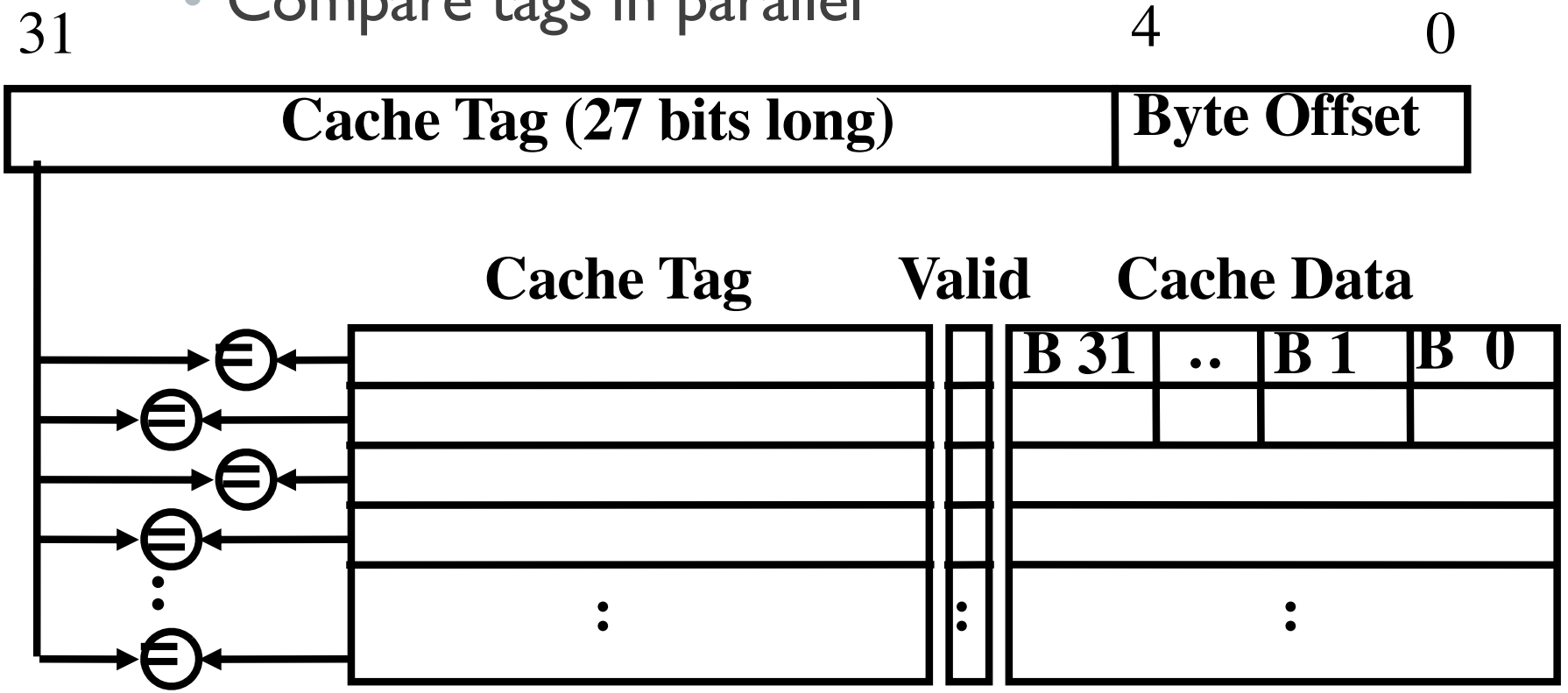
- Cache basics
- Cache performance
  - Evaluation
    - Hit/miss rate, memory stall cycles (CPI)
    - AMAT = Hit time + Miss rate × Miss penalty
  - Improvement
    - Three C's: compulsory misses, capacity misses, conflict misses
    - Placement policy
      - Direct mapped, fully associative, set associative
    - Block replacement policy
    - Multilevel caches
- Virtual memory

Fully Associative Cache

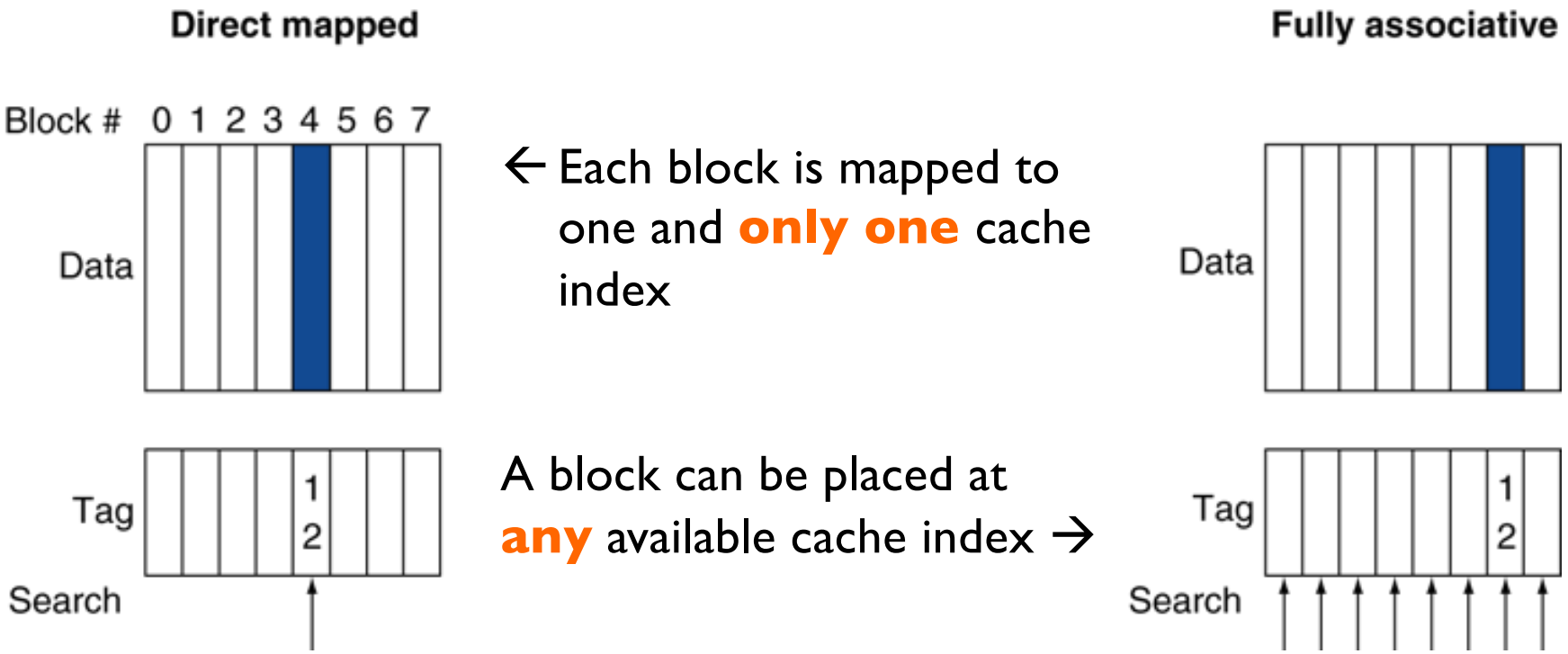
- Any block can go anywhere in the cache
- Memory address fields:
  - Offset: same as before
  - Index: nonexistent
  - Tag: same as before
- Must compare with all tags in entire cache to see if data is there

Fully Associative Cache

- Fully associative cache (with 32-byte blocks)
  - Compare tags in parallel



Cache Block Placement



← Each block is mapped to one and **only one** cache index

A block can be placed at **any** available cache index →

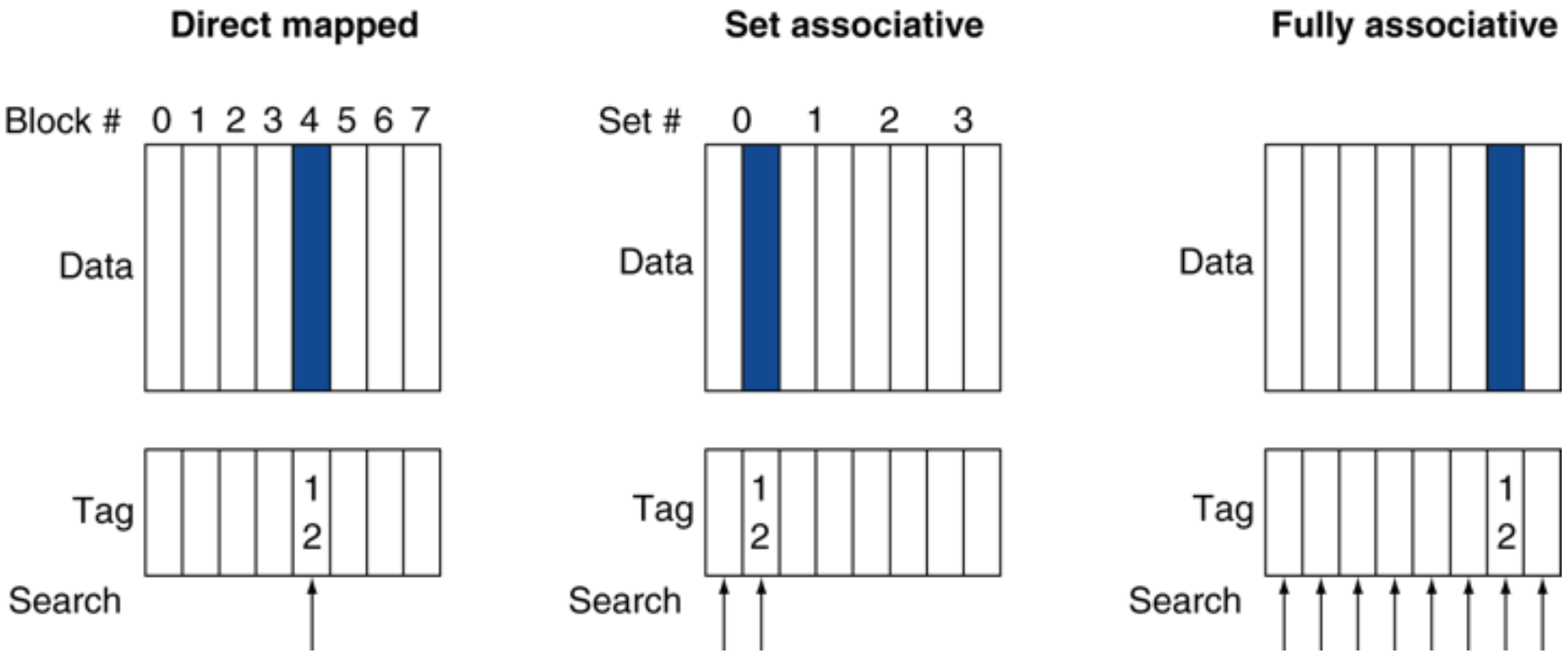
Fully Associative Cache

- Benefits
  - No conflict misses (since data can go anywhere)
  - Mainly capacity misses
- Drawbacks
  - Need hardware comparator for every single entry: if we have a 64KB of data in cache with 4B entries, we need 16K comparators: infeasible

Set Associative Cache

- Design idea
  - Each block is mapped to a unique set
  - Each set contains N(N>=2) blocks: N locations where each block can be placed
    - Once we've found correct set, must compare with all tags in that set to find our data
- Memory address fields:
  - Tag: same as before
  - Offset: same as before
  - Index: points us to the correct group of rows (called a set in this case)
- Set-associative design is more general
  - Direct-mapped: 1-way set associative
  - Fully-associative: M-way set associative (M-block cache only has one set)

Cache of Different Associativity



Index: 3 bits

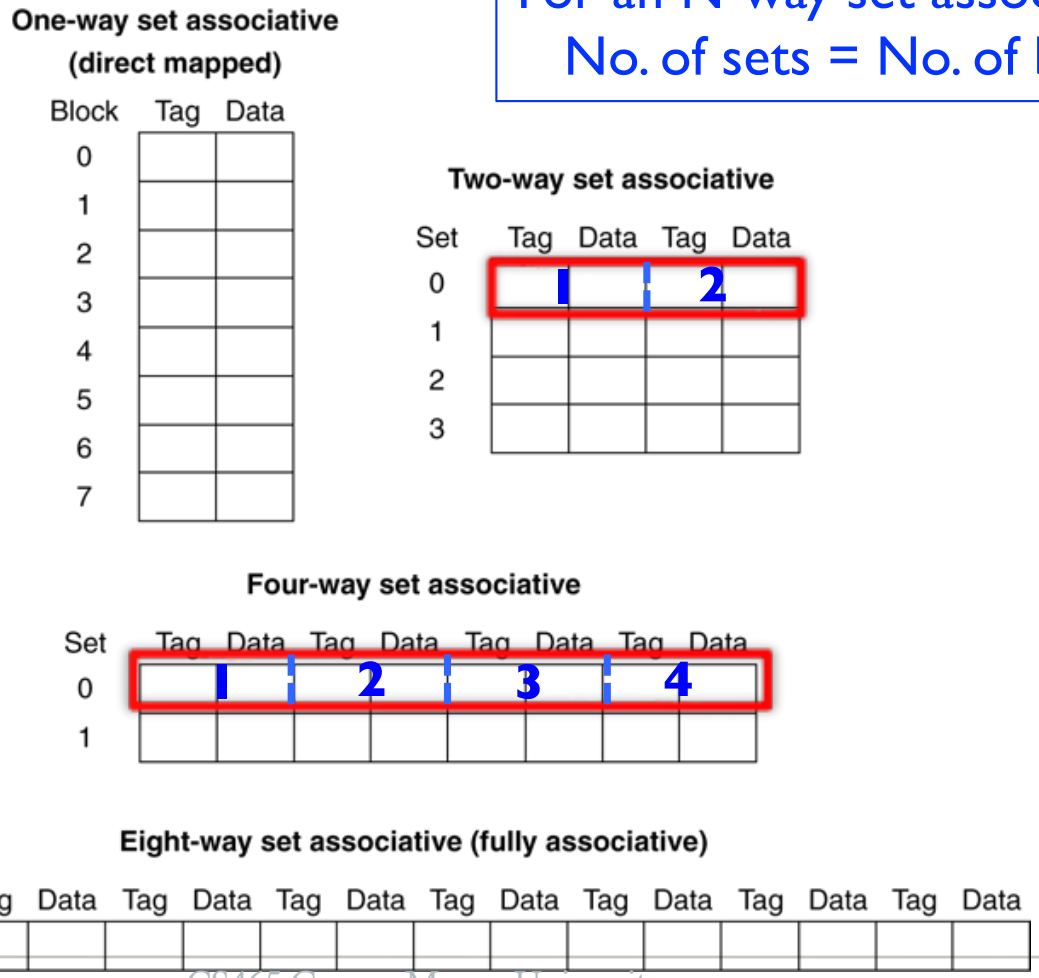
Index: 2 bits

Index: 0 bits

Spectrum of Associativity

- For a cache with 8 entries

For an N-way set associative cache:
No. of sets = No. of blocks / N



Set Associative Cache

- Given a memory address:
  - Find correct set using Index value.
  - Check valid bit and compare Tag with all Tag values in the determined set.
  - If a match occurs, hit! Otherwise a miss.
  - Finally, use the Offset field as usual to find the desired data within the block.

Associativity Example

- Compare 4-block caches
  - Direct mapped, 2-way set associative, fully associative
  - Block access sequence: 0, 8, 0, 6, 8
  - (Block number) modulo (number of sets)
- Direct mapped
  - Number of sets = 4

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 number) modulo (number of sets)
  - Number of sets = 4/2 = 2

Block address	Cache (set) 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]		

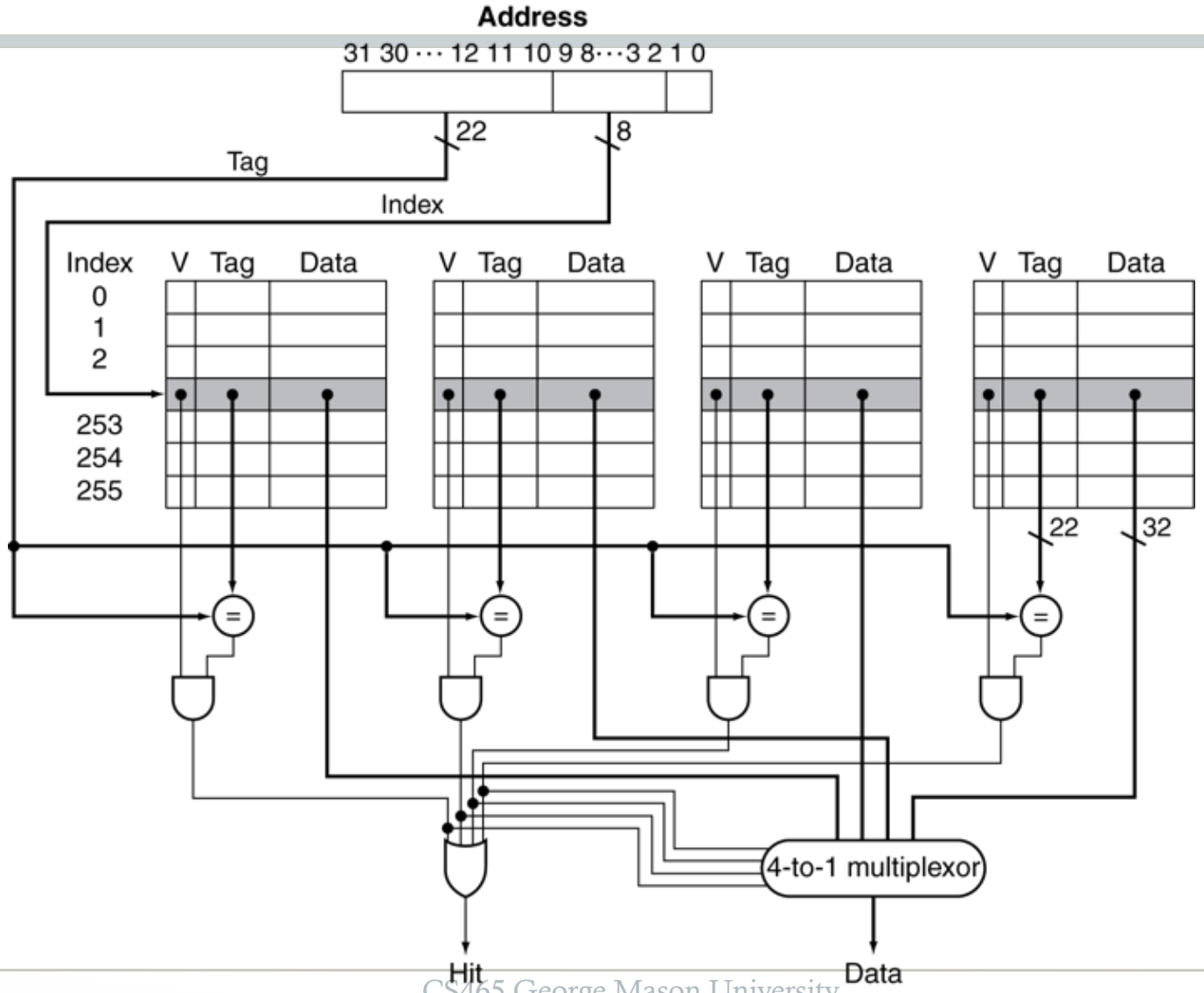
- Fully associative

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]	

Set Associative Cache

- What's so great about this?
  - Even a 2-way set assoc cache avoids a lot of conflict misses
  - Hardware cost isn't that bad: only need N comparators (N usually a small integer)

Set Associative Cache Organization



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%