CMPE110 Lecture 17
Caches II

Heiner Litz

https://canvas.ucsc.edu/courses/19290

Announcements

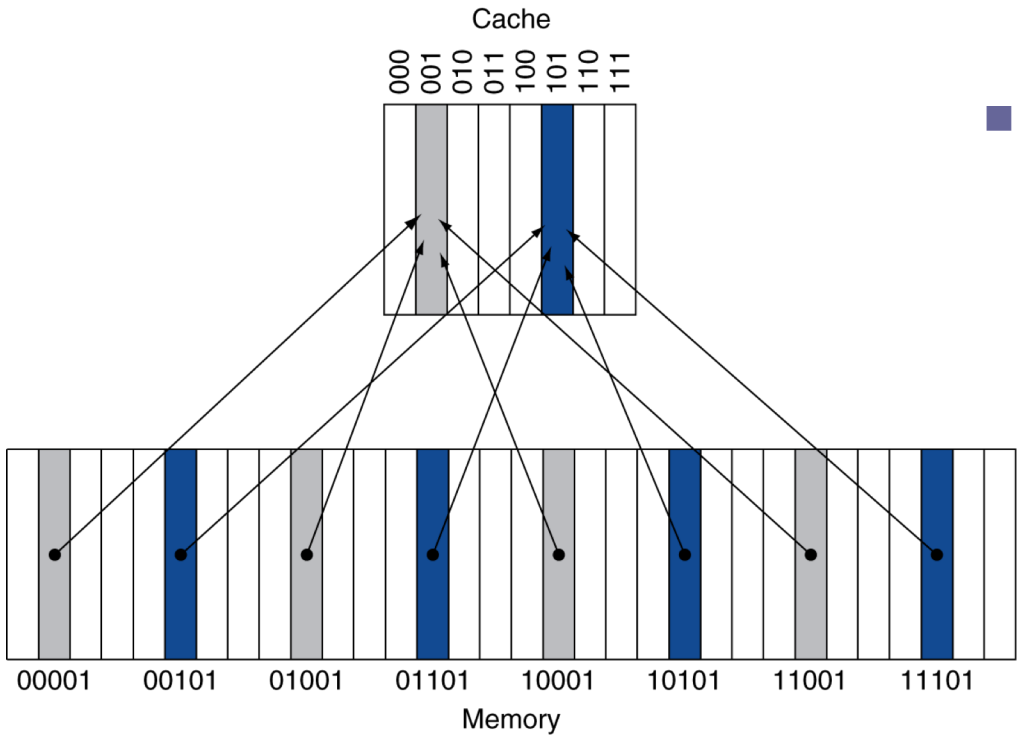
Review

Locality

- Principle of locality
  - Programs work on a relatively small portion of data at any time
  - Can predict data accessed in near future by looking at recent accesses
- Temporal locality
  - If an item has been referenced recently, it will probably be accessed again soon
- Spatial locality
  - If an item has been accessed recently, nearby items will tend to be referenced soon
- Examples?

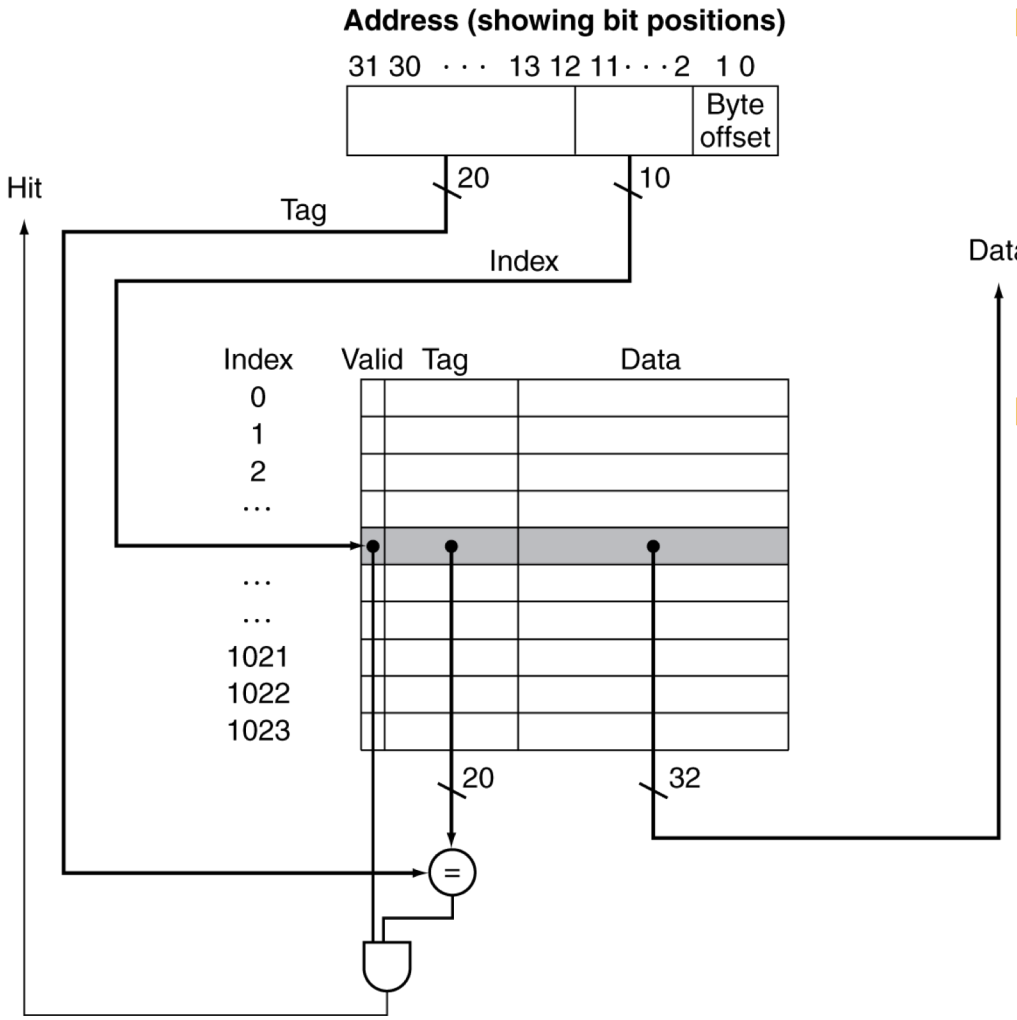
Direct Mapped Cache

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



- Simplification
  - If # blocks in cache is power of 2
  - Modulo is just using the low-order bits

Cache Organization & Access



- Assumptions
  - 32-bit address
  - 4 Kbyte cache
  - 1024 blocks, 32 bit/block
- Steps
  1. Use index to read V, tag from cache
  2. Compare read tag with tag from address
  3. If match, return data & hit signal
  4. Otherwise, return miss

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

Compulsory/Cold Miss

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

Compulsory/Cold Miss

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

Hit

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

Replacement

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		

Terminology

- *Block* – Minimum unit of data that is present at any level of the hierarchy
- *Hit* – Data found in the cache
- *Hit rate* – Percent of accesses that hit
- *Hit time* – Time to access on a hit
- *Miss* – Data not found in the cache
- *Miss rate* – Percent of misses (1 - Hit rate)
- *Miss penalty* – Overhead in getting data from a higher numbered level
  - Miss penalty = higher level access time + Time to deliver to lower level + Cache replacement / forward to processor time
  - Miss penalty is usually much larger than the hit time
  - This is in addition to the hit time
- These apply to each level of a multi-level cache
  - e.g., we may miss in the L1 cache and then hit in the L2

Average Memory Access Times

Access time = hit time + miss rate × miss penalty

- Average Memory Access Time (AMAT)
  - Formula can be applied to any level of the hierarchy
    - Access time for that level
  - Can be generalized for the entire hierarchy
    - Average access time that the processor sees for a reference

The 3Cs of Cache Misses

- Why does a reference miss in the cache?
- Compulsory – this is the first time you referenced this item
- Capacity – not enough room in the cache to hold items
  - i.e., this miss would disappear if the cache were big enough
- Conflict – item was replaced because of a conflict in its set
  - i.e., this miss would disappear with more associativity

How Processor Handles a Miss

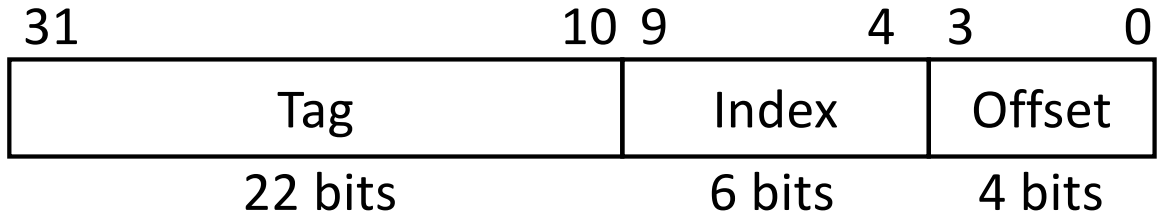
- Assume that cache access occurs in 1 cycle
  - Hit is great, and basic pipeline is fine

    CPI penalty = miss rate × miss penalty

- For our processor, a miss stalls the pipeline (for a instruction or data miss)
  - Stall the pipeline (you don't have the data it needs)
  - Send the address that missed to the memory
  - Instruct main memory to perform a read and wait
  - When access completes, return the data to the processor
  - Resume the instruction

Larger Block Size

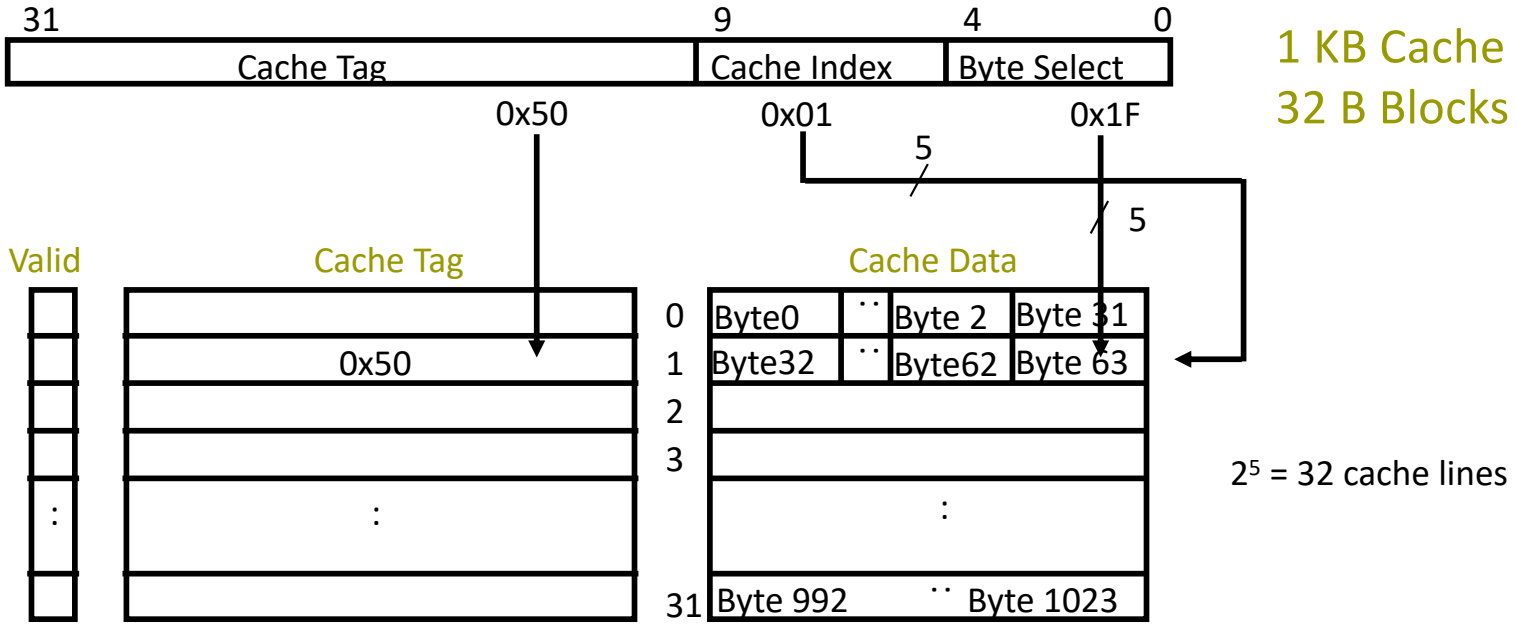
- Motivation: exploit spatial locality & amortize overheads
- This example: 64 blocks, 16 bytes/block
  - To what block number does address 1200 map?
  - Block address = 1200/16 = 75
  - Block index = 75 modulo 64 = 11



- What is the impact of larger blocks on tag/index size?
- What is the impact of larger blocks on the cache overhead?
  - Overhead = tags & valid bits

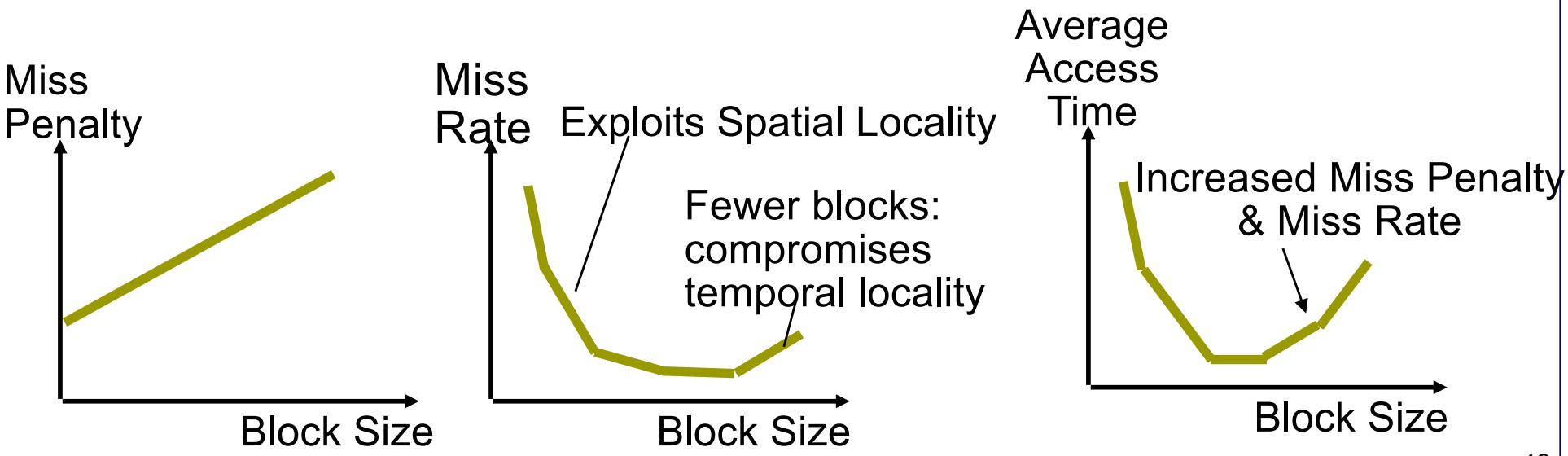
Cache Block Example

- Assume a 2^n byte direct mapped cache with 2^m byte blocks
  - Byte select – The lower m bits
  - Cache index - The lower (n-m) bits of the memory address
  - Cache tag - The upper (32-n) bits of the memory address



Block Sizes

- Larger block sizes take advantage of spatial locality
  - Also incurs larger miss penalty since it takes longer to transfer the block into the cache (can finesse this)
  - Large block can also increase the average time or the miss rate
- Tradeoff in selecting block size
- Average Access Time = Hit Time + Miss Penalty • MR



Direct Mapped Problems: Conflict misses

- Two blocks are used concurrently and map to same index
  - Only one can fit in the cache, regardless of cache size
  - No flexibility in placing 2nd block elsewhere
- Thrashing
  - If accesses alternate, one block will replace the other before reuse
    - in our previous example 18, 26, 18, 26, … - every reference will miss
  - No benefit from caching
- Conflicts & thrashing can happen quite often

This is a real problem!

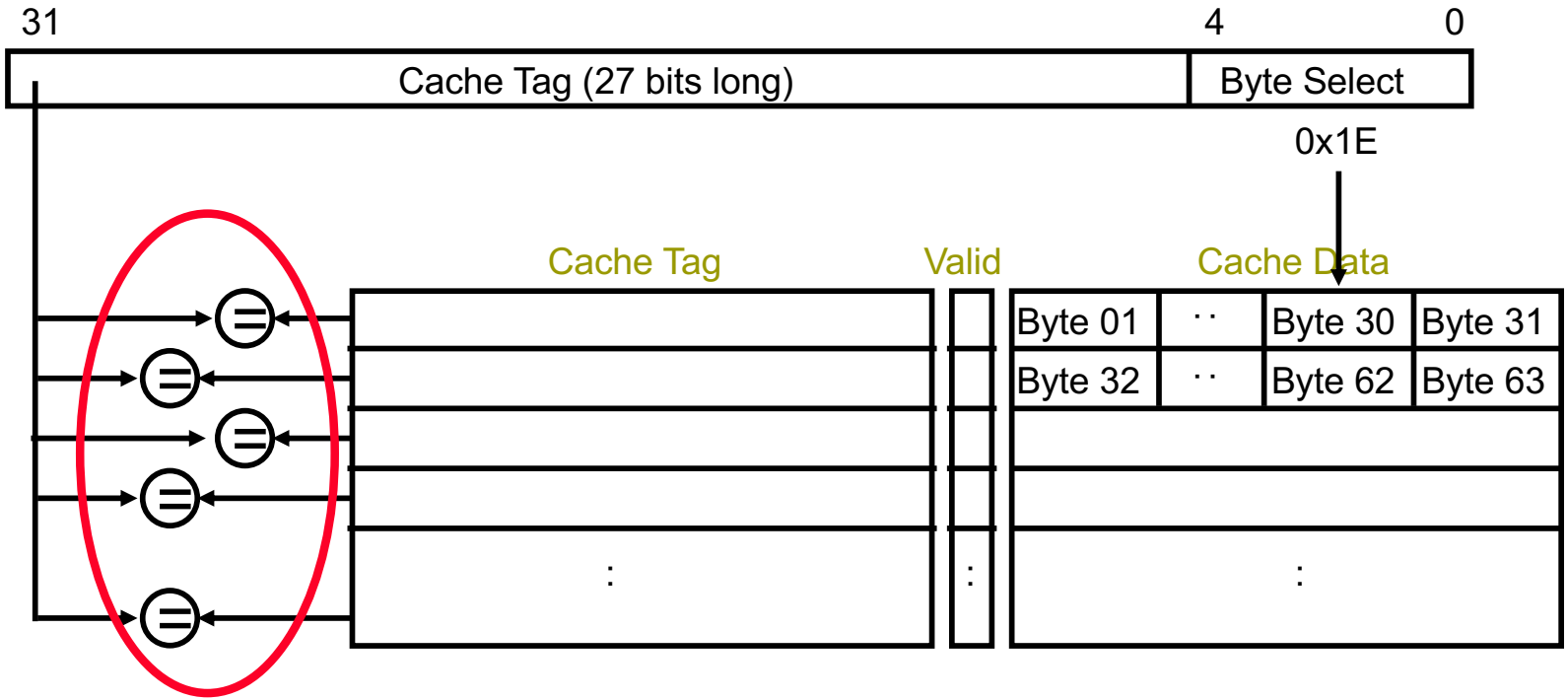
- Consider the following example code:

```
double a[8192], b[8192], c[8192];

void vector_sum()
{
    int i;

    for (i = 0; i < 8192; i++)
        c[i] = a[i] + b[i];
}
```

  - Arrays a, b, and c will tend to conflict in small caches
  - Code will get cache misses with *every* array access (3 per loop)
  - Spatial locality savings from blocks will be eliminated
- How can the severity of the conflicts be reduced?

Fully Associative Cache

- Opposite extreme in that it has no cache index to hash
  - Use any available entry to store memory elements
  - No conflict misses, only capacity misses
  - Must compare cache tags of *all* entries to find the desired one
    - Expensive or slow

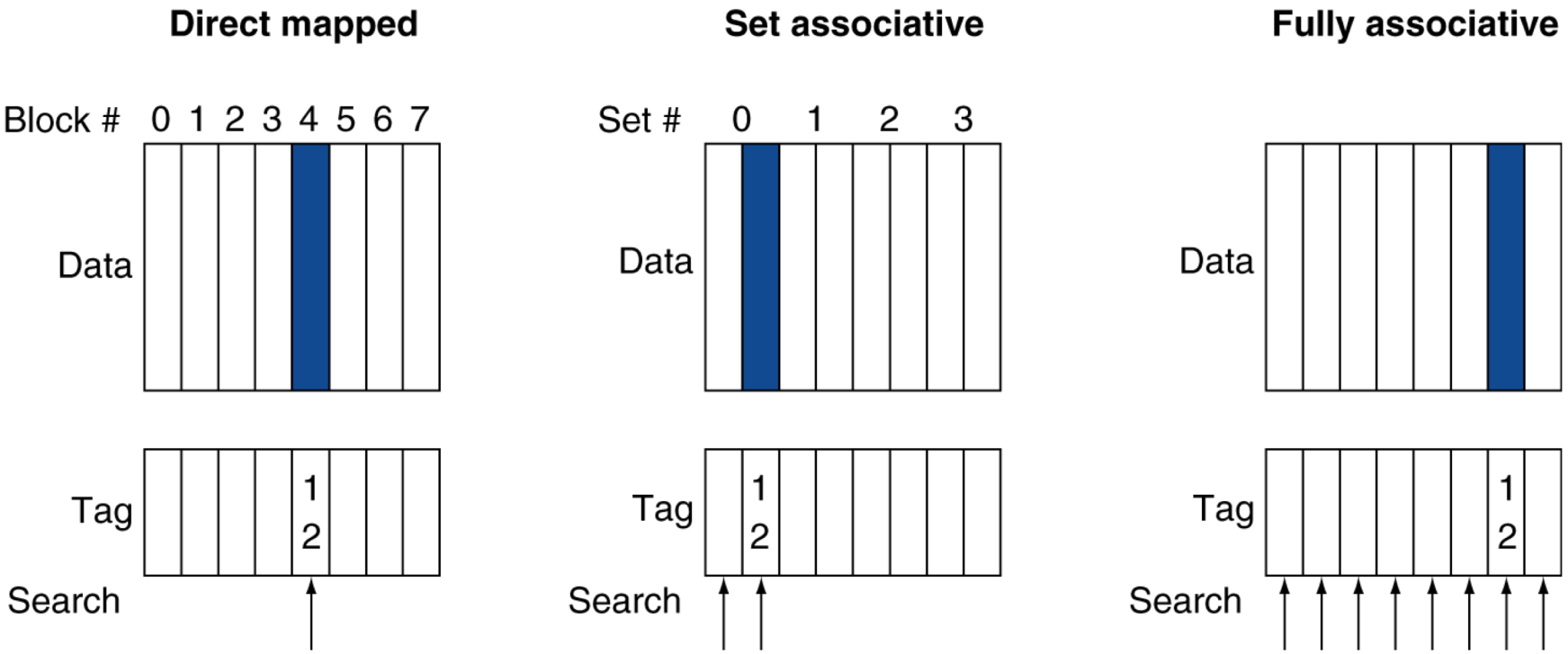


N-way Set Associative Cache

- Compromise between direct-mapped and fully associative
  - Each memory block can go to one of N entries in cache
    - Each "set" can store N blocks; a cache contains some number of sets
  - For fast access, all blocks in a set are search in parallel

- How to think of a N-way associative cache with X sets
  - 1st view: N direct mapped caches each with X entries
    - Caches search in parallel
    - Need to coordinate on data output and signaling hit/miss
  - 2nd view: X fully associative caches each with N entries
    - One cache searched in each case

Associative Cache Example



Associative Cache Example

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
  - Index = (address % 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

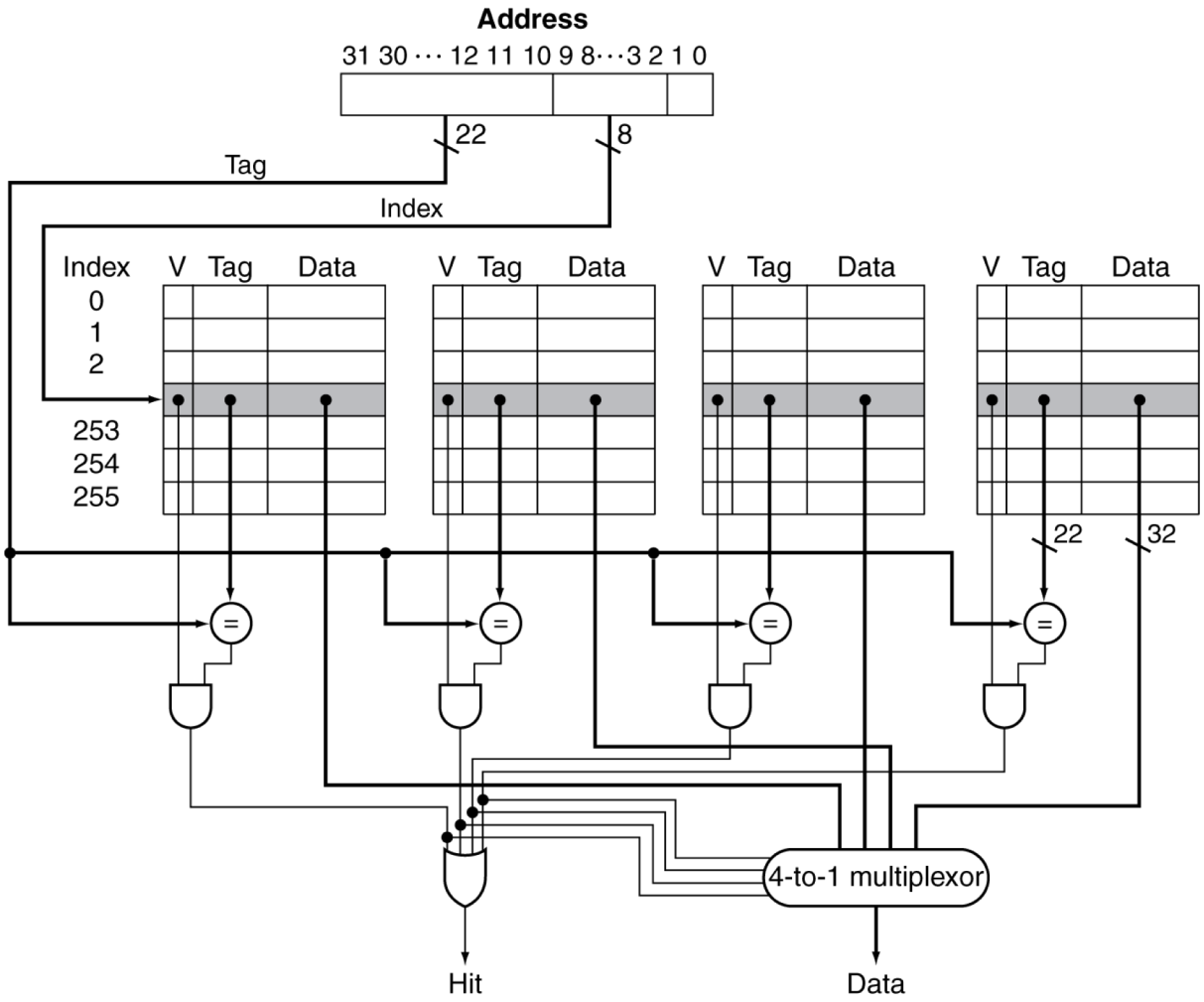
- 2-way set associative
  - Index = (address % 2)

Block address	Cache index	Hit/miss	Cache content after access			
			Set 0 (even)		Set 1 (odd)	
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
  - Blocks can go anywhere

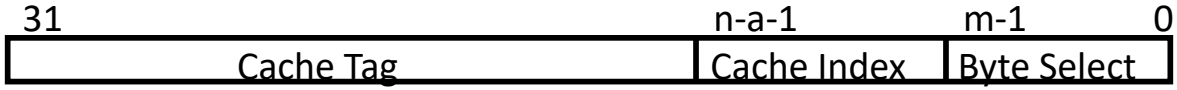
Block address		Hit/miss	Cache content after access			
0		miss	**Mem[0]**			
8		miss	Mem[0]	**Mem[8]**		
0		hit	**Mem[0]**	Mem[8]		
6		miss	Mem[0]	Mem[8]	**Mem[6]**	
8		hit	Mem[0]	**Mem[8]**	Mem[6]	

Set Associative Cache Design



Tag & Index with Set-Associative Caches

- Given a 2^n-byte cache with 2^m-byte blocks that is 2^a set-associative
  - Which bits of the address are the tag or the index?
  - m least significant bits are byte select within the block
- Basic idea
  - The cache contains $2^n/2^m=2^{n-m}$ blocks
  - Each cache way contains $2^{n-m}/2^a=2^{n-m-a}$ blocks
  - Cache index: (n-m-a) bits after the byte select
    - Same index used with all cache ways…



- Observation
  - For fixed size, length of tags increases with the associativity
  - Associative caches incur more overhead for tags

Associative Caches: Pros

- Increased associativity decreases miss rate
  - Eliminates conflicts
  - 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%
- Caveat: cache shared by multiple cores may have need higher associativity

Review: Cache Organization Options 256 bytes, 16 byte block, 16 blocks

Organization	# of sets	# blocks / set	12 bit Address
Direct mapped			tag: 4 | index: 4 | blk off: 4
2-way set associative			tag: 5 | index: 3 | blk off: 4
4-way set associative			tag: 6 | ind: 2 | blk off: 4
8-way set associative			tag: 7 | i: 1 | blk off: 4
16-way (fully) set associative			tag: 8 | blk off: 4

Associative Caches: Cons

- Area overhead
  - More storage needed for tags (compared to same sized DM)
  - N comparators
- Latency
  - Critical path = way access + comparator + logic to combine answers
    - Logic to OR hit signals and multiplex the data ouptuts
  - Cannot forward the data to processor immediately
    - Must first wait for selection and multiplexing
    - Direct mapped assumes a hit and recovers later if a miss
- Complexity: dealing with replacement