Memory hierarchy: caching

Jinyang Li

Based on Patterson and Hennessy's slides

What we’ve learnt so far

- Single cycle RISC-V CPU design
- 5 stage pipelined RISC-V CPU
  - Pipelining challenges: hazards
    - Must stall (bubble) to ensure correctness
  - 3 types of hazards:
    - Structure (To mitigate, add resources)
    - Data (To mitigate, do forwarding/bypassing)
    - Control (Predict/speculate)

Today’s lesson plan

- Memory hierarchy
- Caching
    - Performance
    - Design

Programmers want fast and unlimited memory, but…

Processor	Flip-flop		< 0.5ns	
Memory	SRAM	Integrated on chip	0.5-2.5ns	\$500-\$1000/GB
Memory	DRAM		50-70ns	\$7-\$15/GB
Memory	Flash		5-50μs	\$0.75-\$1/GB
Memory	Magnetic disk		5-20ms	\$0.05-\$0.1/GB

speed: fastest ↓ slowest

size: biggest ↑ smallest

How to give programmers the illusion of fast and vast memory? Caching!

- Copy recently accessed (and nearby) items from disk to smaller DRAM memory

Referred to as main memory

- Copy more recently accessed (and nearby) items from DRAM to smaller SRAM memory

Referred to as cache memory (integrated on CPU chip)

Done at the granularity of a block or line

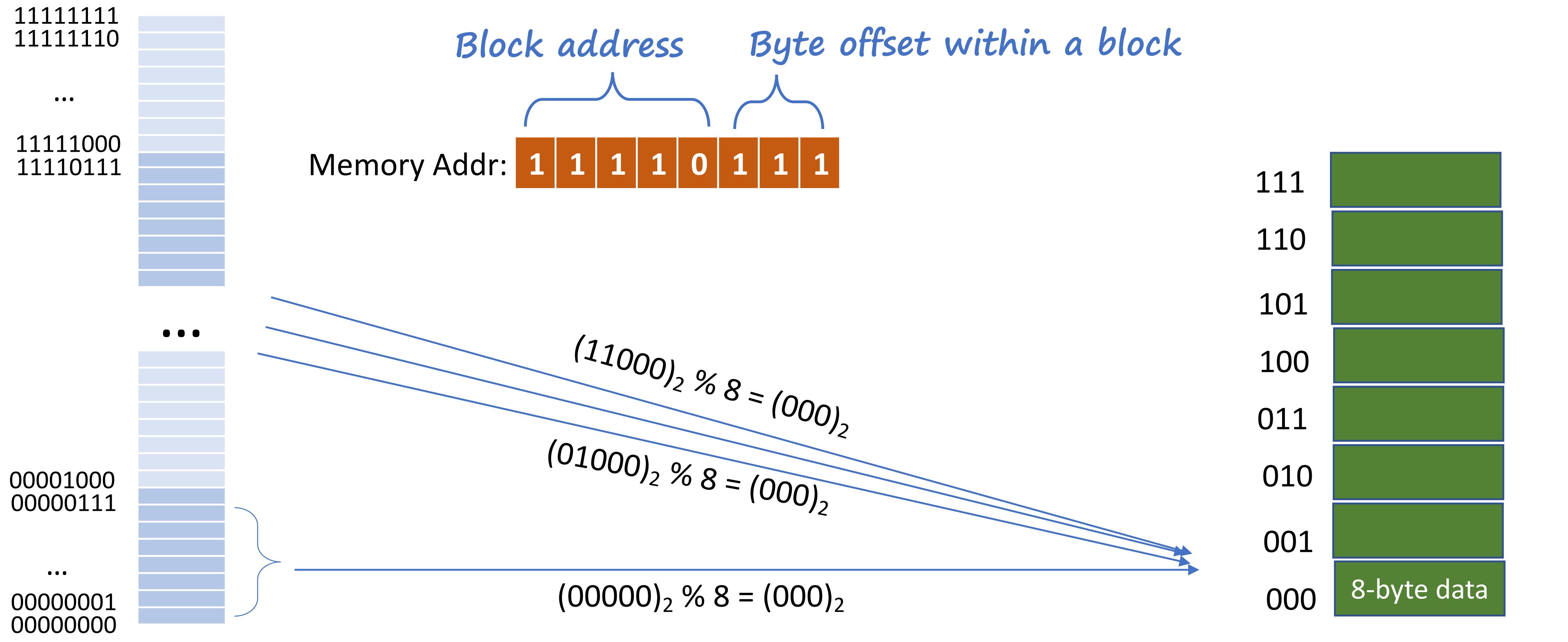
This lecture: focus on cache memory only

Why caching works? 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

Direct Mapped Cache

- Direct mapped: each mem block can only be cached at one location
  - (Block address) modulo (#Blocks in cache)

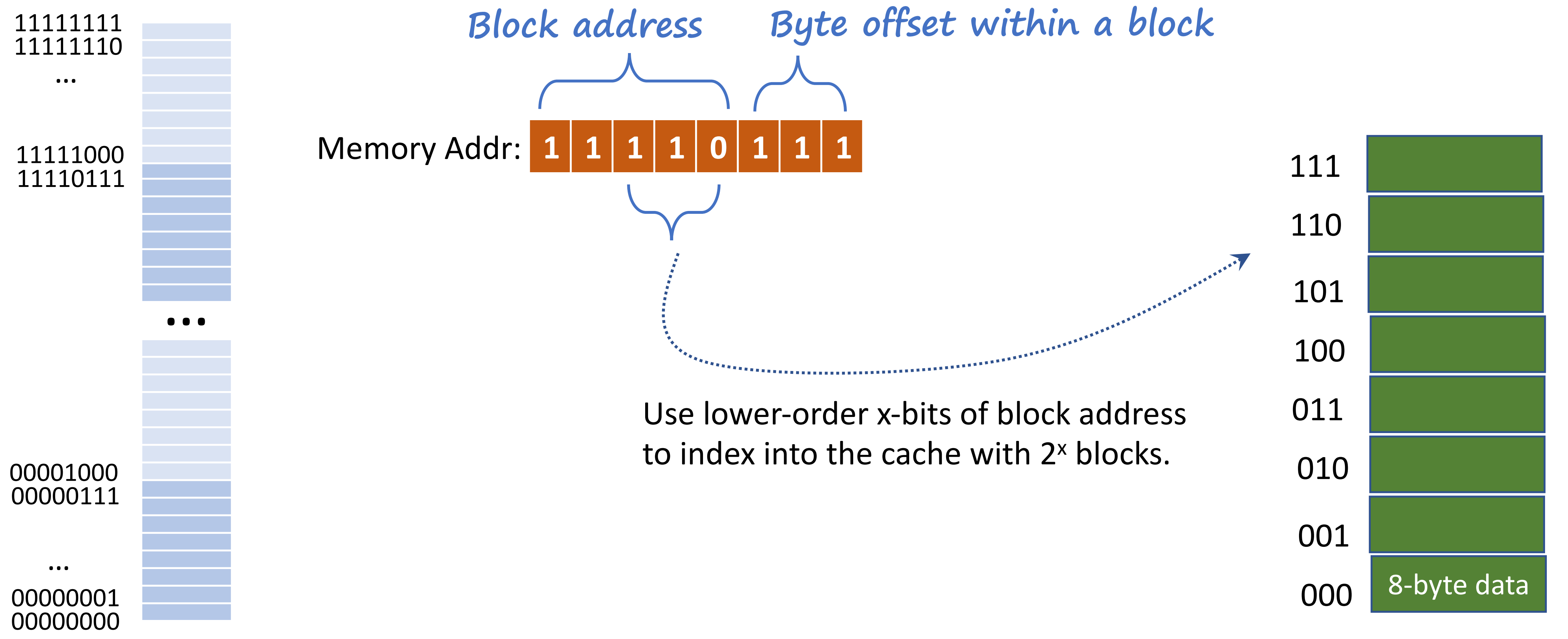


Suppose cache has 8 blocks

#Blocks =TotalMem/BlockSize =$2^8/2^3$

Direct Mapped Cache

- Direct mapped: each mem block can only be cached at one location
  - (Block address) modulo (#Blocks in cache)



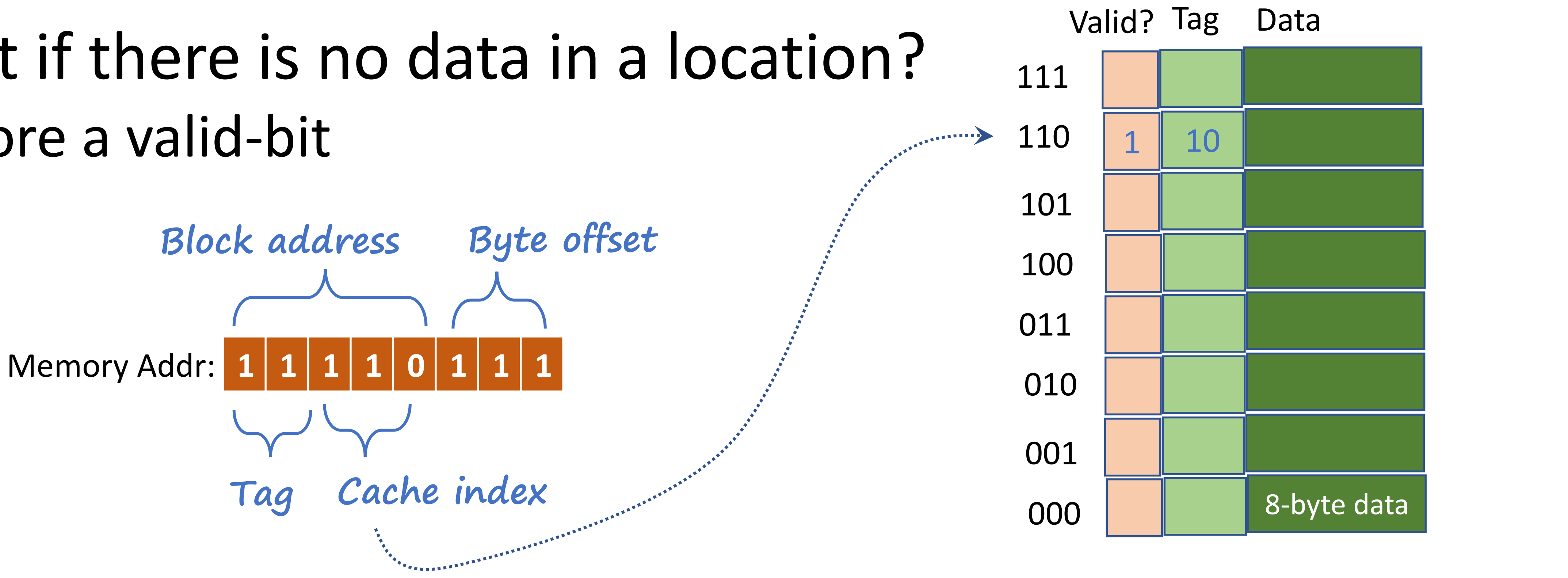
Use lower-order x-bits of block address to index into the cache with 2^x blocks.

#Blocks =TotalMem/BlockSize $=2^8/2^3$

Suppose cache has 8 blocks or 8 “lines”

Tags and Valid Bits

- How to track which mem block is stored in a cache location?
  - Store tag
- What if there is no data in a location?
  - Store a valid-bit



Suppose cache has 8 blocks or 8 "lines"

Cache Example

- 8-blocks, 8-bytes/block, direct mapped
- Initial state

	Valid?	Tag	Data
111	N		
110	N		
101	N		
100	N		
011	N		
010	N		
001	N		
000	N		

Cache Example

Memory addr accessed: 10110xxx

	Valid?	Tag	Data
111	N		
110	N		
101	N		
100	N		
011	N		
010	N		
001	N		
000	N		

Cache Example

Memory addr accessed: 11010xxx

	Valid?	Tag	Data
111	N		
110	Y	10	8-bytes starting at mem[10110]
101	N		
100	N		
011	N		
010	N		
001	N		
000	N		

Cache Example

Memory addr accessed: 10110xxx

	Valid?	Tag	Data
111	N		
110	Y	10	8-bytes starting at mem[10110]
101	N		
100	N		
011	N		
010	Y	11	8-bytes starting at mem[11010]
001	N		
000	N		

Cache Example

Memory addr accessed: 00010xxx

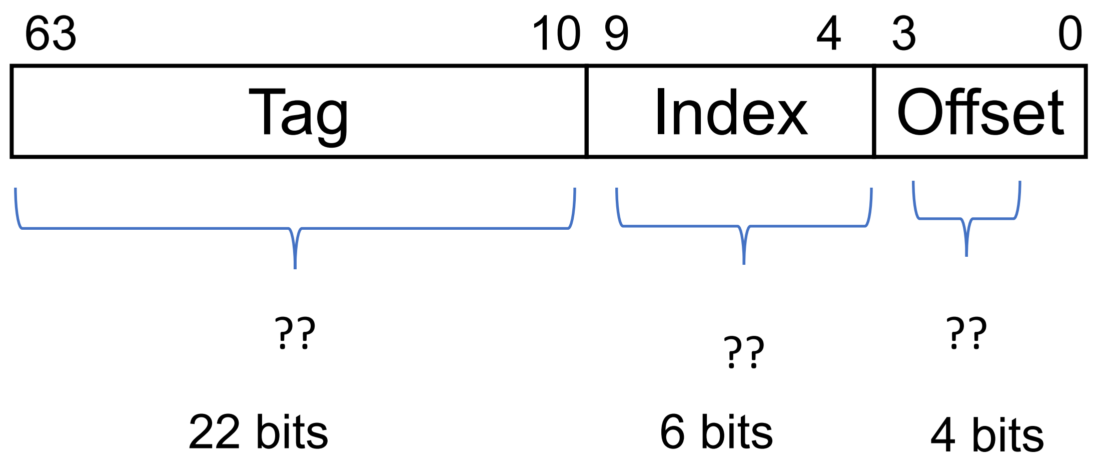
	Valid?	Tag	Data
111	N		
110	Y	10	8-bytes starting at mem[10110]
101	N		
100	N		
011	N		
010	Y	11	8-bytes starting at mem[11010]
001	N		
000	N		

Cache Example

	Valid?	Tag	Data
111	N		
110	Y	10	8-bytes starting at mem[10110]
101	N		
100	N		
011	N		
010	Y	00	8-bytes starting at mem[00010]
001	N		
000	N		

Another example: Larger Block Size

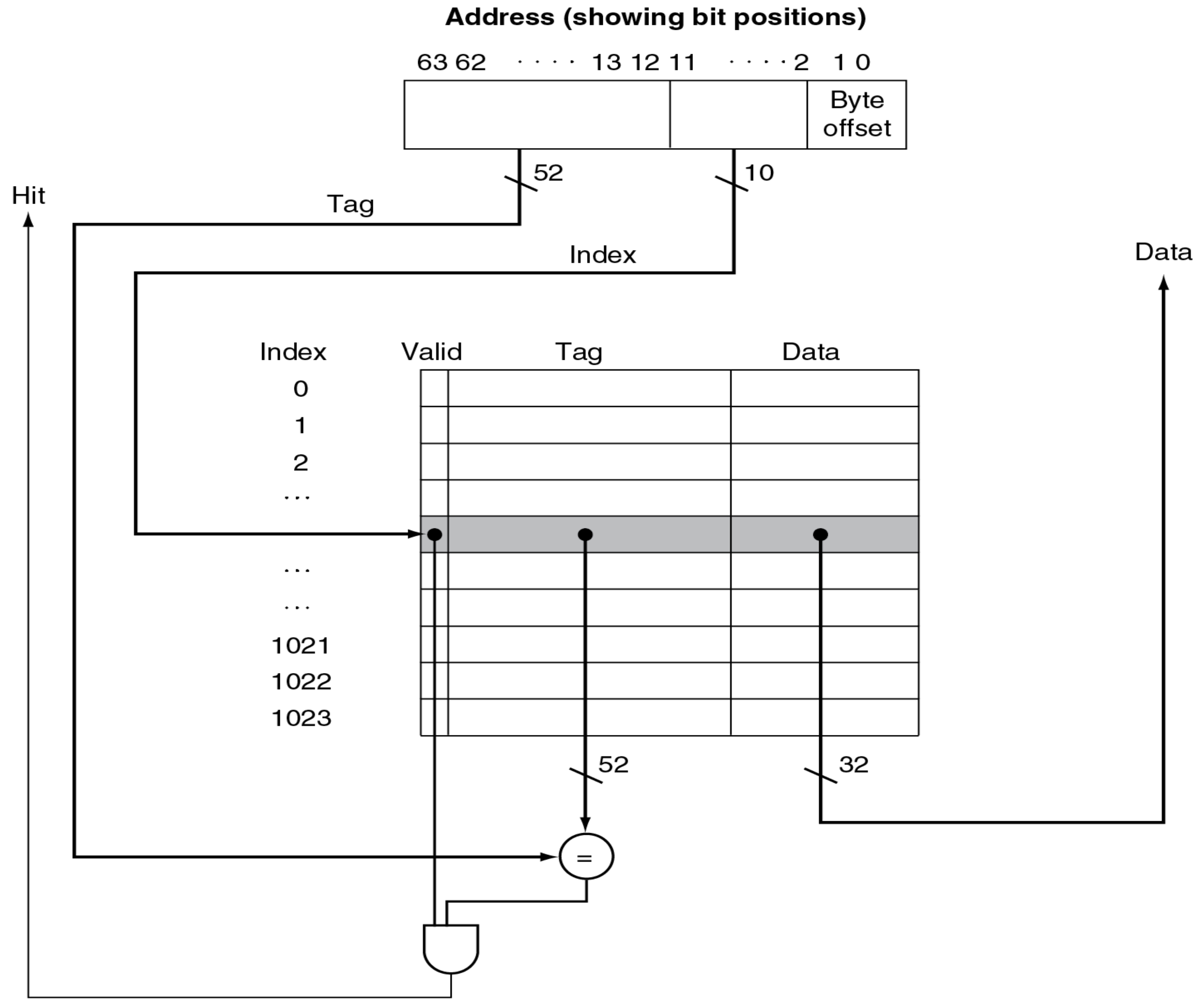
- 64-bit memory address
- 64 cache blocks, 16 bytes/block



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 $\Rightarrow$ pollution
- Larger miss penalty
  - Can override benefit of reduced miss rate

Hardware of direct mapped cache



Cache Misses

- 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

Very expensive: usually 20~100 cycles

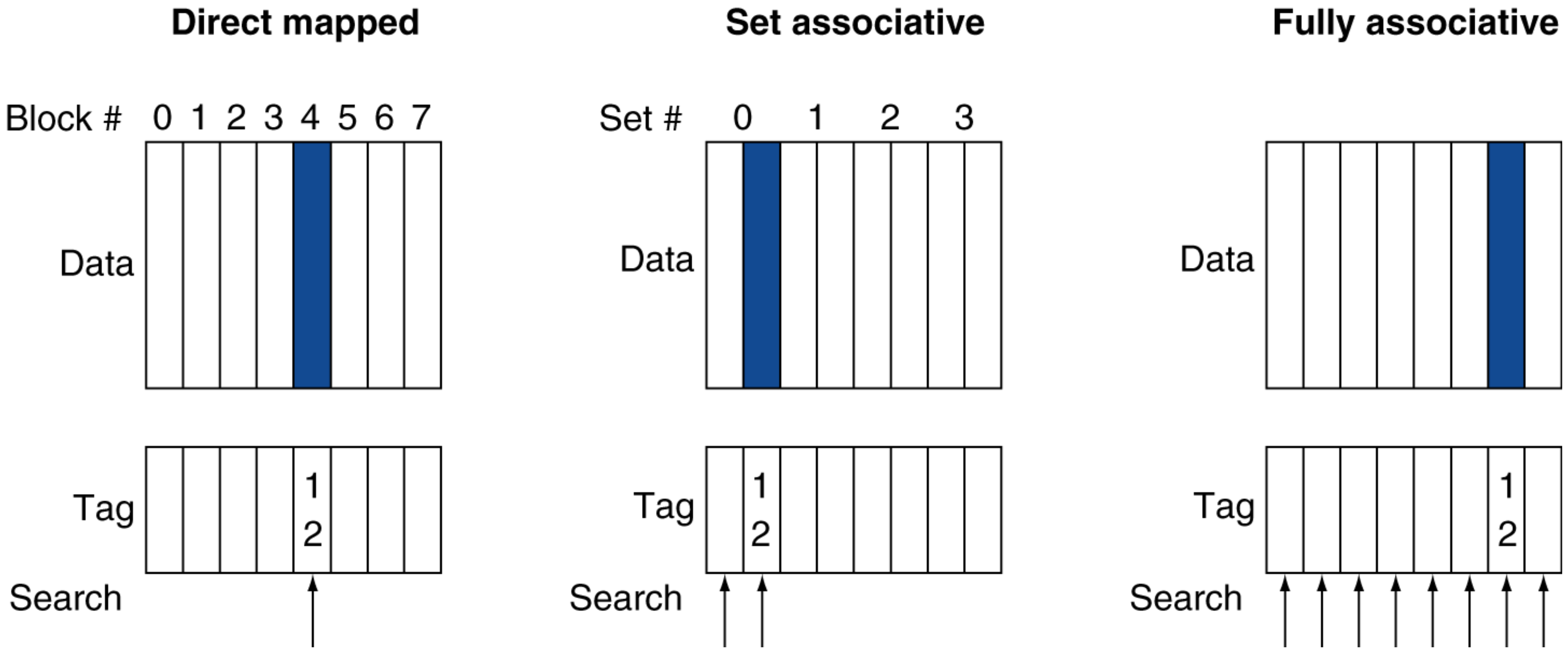
Write-Through vs write-back

- Write-through: On write hit, update memory upon write
- Write-back: On write hit, update cache only
  - Keep track of whether a block in cache is dirty
  - When a dirty block is replaced, write it back to memory

Reducing cache miss: Associative Cache

- Fully associative
    - Allow a given block to go in any cache entry
    - Require all entries to be searched at once
        - Need a comparator per cache entry (expensive)
- *n*-way set associative
    - Divide cache into sets each of which contains *n* entries
    - Block number determines which set
        - (Block number) modulo (#Sets in cache)
    - Fully-associative within the set: search n entries of a set at once
        - *n* comparators (less expensive)

Associative Cache Example



Cache locations of a memory block with block address 12

Associativity Example

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]	

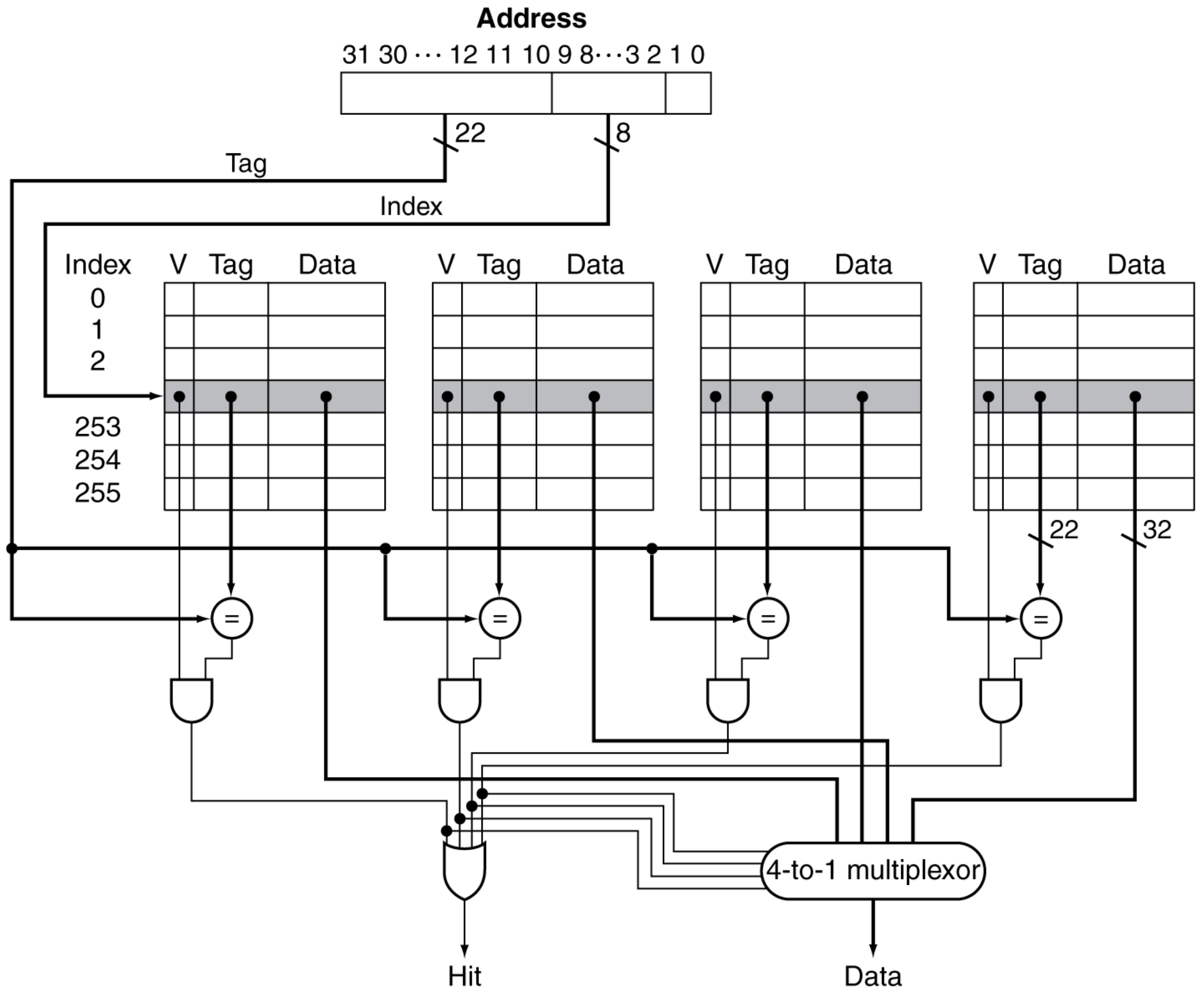
- 2-way set associative

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]		

How Much Associativity

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

4-way 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
- Least-recently used (LRU)
  - Choose the one unused for the longest time
    - Hardware implementation: simple for 2-way, manageable for 4-way, too hard beyond that

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
- High-end systems include L-3 cache

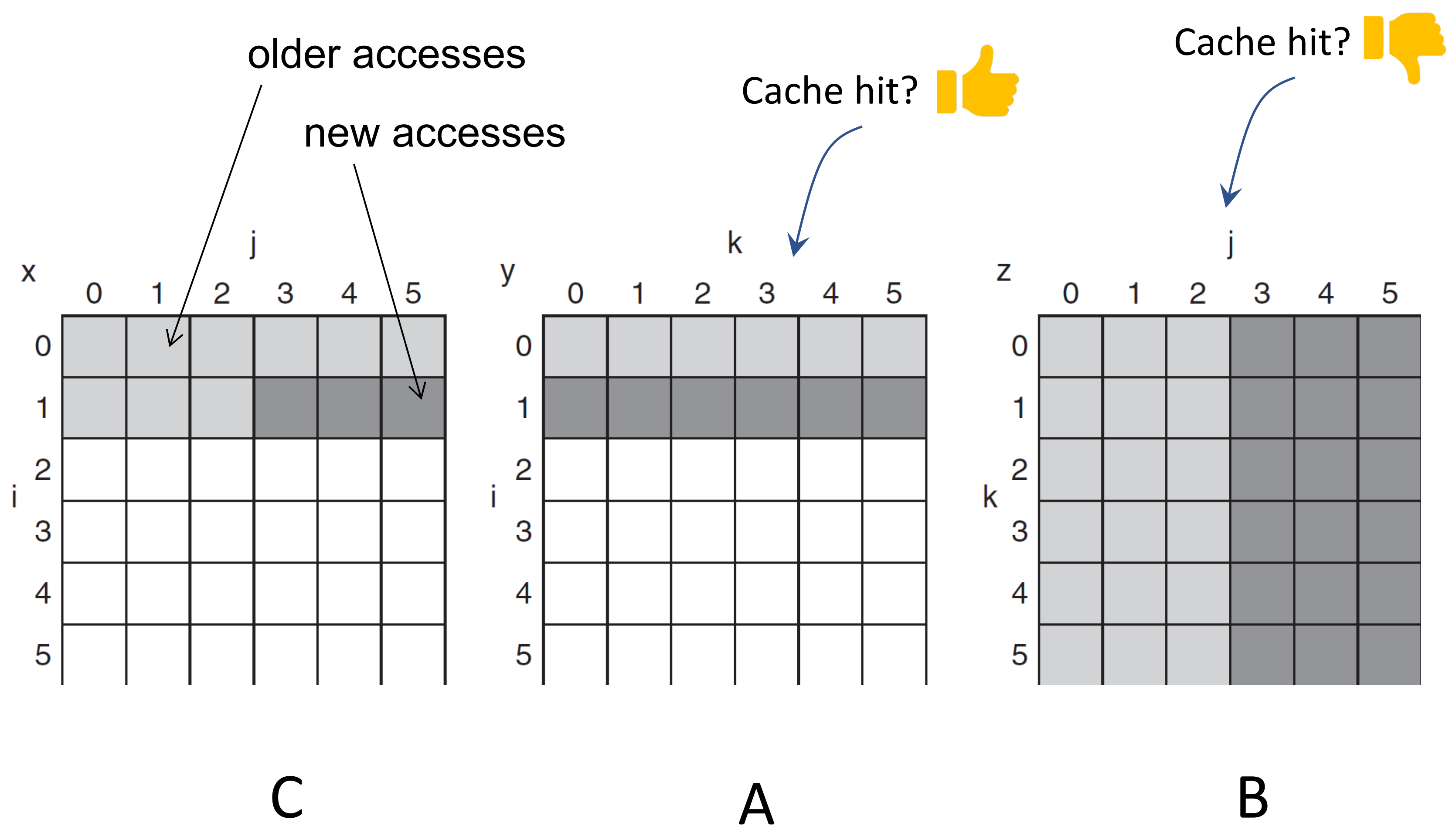
Software Optimization via Blocking

- Goal: maximize accesses to data before it is replaced
- Example: Naïve matrix multiplication (DGEMM)

$$C_{i,j} = \sum_{k=0}^{n-1} A_{i,k} * B_{k,j}$$

```
for (int i = 0; i < n; i++) {
  for (int j = 0; j < n; j++) {
      double cij = 0;
      for( int k = 0; k < n; k++ )
        cij += A[i+k*n] * B[k+j*n];
      C[i+j*n] = cij;
  }
}
```

DGEMM access pattern

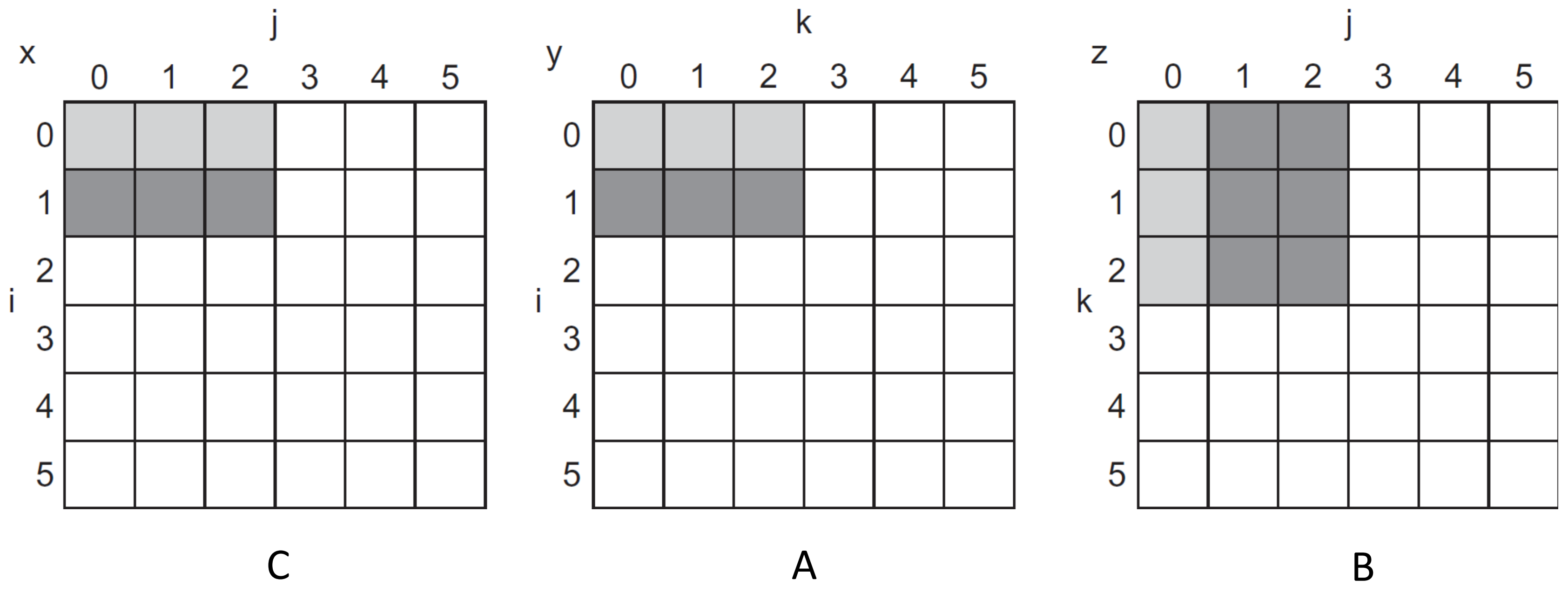


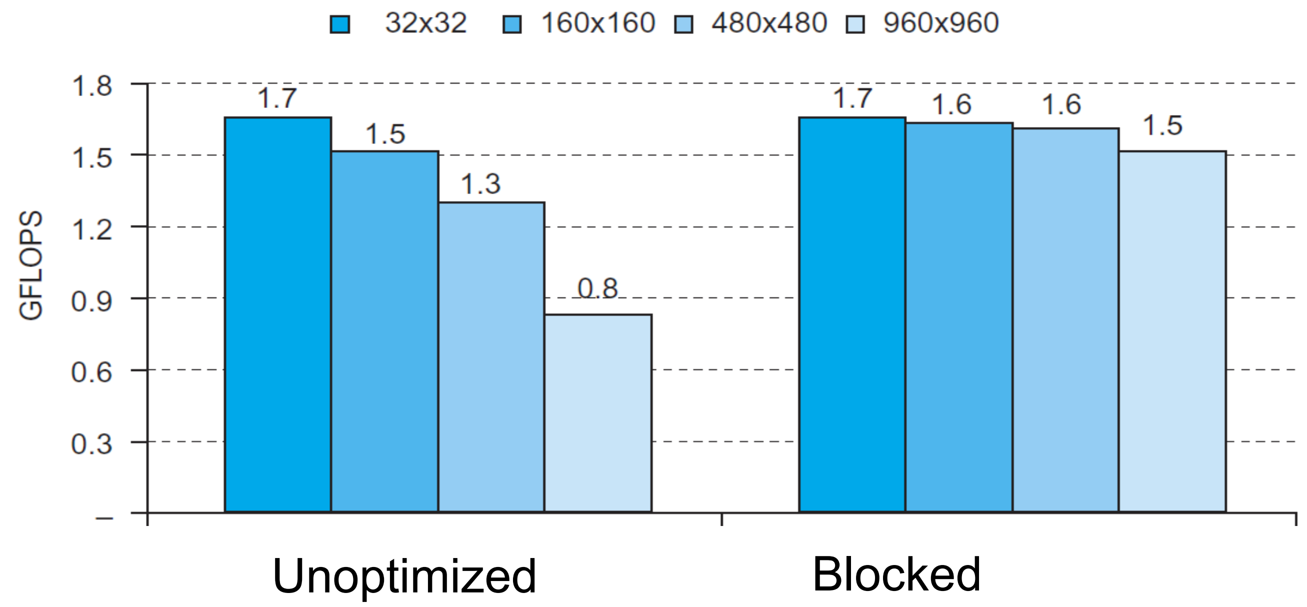
Blocked DGEMM

```
#define BLOCKSIZE 32
void do_block (int n, int si, int sj, int sk, double *A, double*B, double *C)
{
  for (int i = si; i < si+BLOCKSIZE; ++i)
    for (int j = sj; j < sj+BLOCKSIZE; ++j)
    {
     double s = 0;
     for( int k = sk; k < sk+BLOCKSIZE; k++ )
         s += A[i+k*n] * B[k+j*n];
     C[i+j*n] += s; //accumulate s to C[i][j] */
    }
}

void dgemm (int n, double* A, double* B, double* C)
{
 for ( int sj = 0; sj < n; sj += BLOCKSIZE )
    for ( int si = 0; si < n; si += BLOCKSIZE )
      for ( int sk = 0; sk < n; sk += BLOCKSIZE )
       do_block(n, si, sj, sk, A, B, C);
}
```

Blocked DGEMM Access Pattern





Summary

- Memory hierarchy
- Caching works due to principles of locality
- Direct mapped vs. set-associate cache
- Software optimization via blocking