Memory hierarchy: caching

Jinyang Li

Based on Patterson and Hennessy's slides

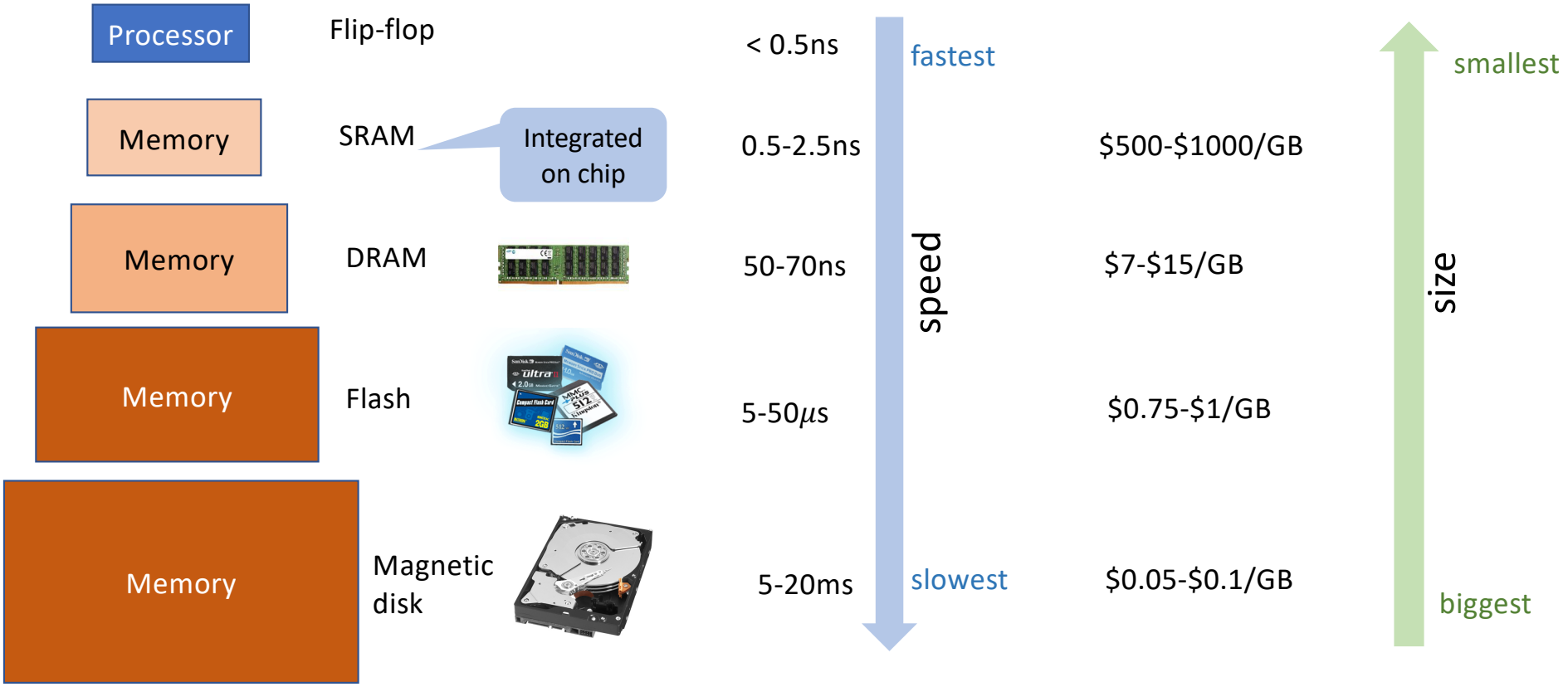
What we've learnt so far

- 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

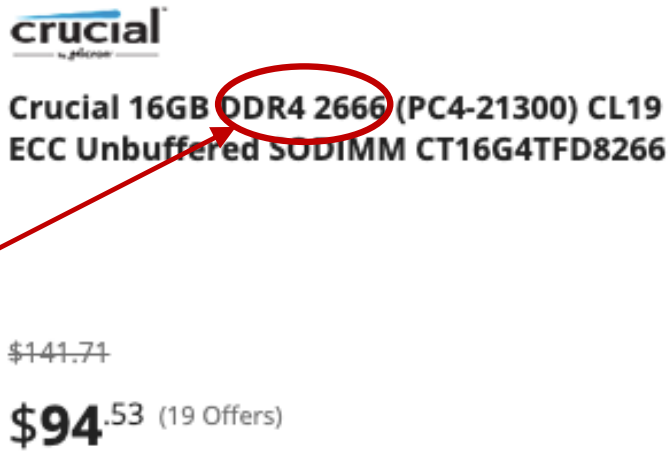
- Memory hierarchy
- Caching
  - Performance
  - Design

Programmers want fast and unlimited memory, but…



DRAM Technology

- Data stored as a charge in a capacitor
  - Single transistor used to access the charge
  - Must periodically be refreshed
    - Read contents and write back
- Double data rate (DDR) DRAM
  - Transfer on rising and falling clock edges



2666 *10^6 transfers/sec
8 bytes per transfer

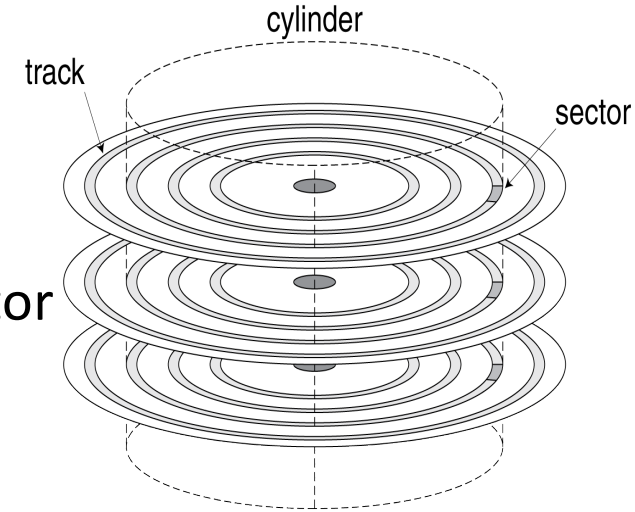
Flash Storage

- Nonvolatile semiconductor storage
  - Compared to DRAM
    - 100x – 1000x slower
  - Compared to magnetic disk
    - 100× – 1000× faster than magnetic disk
    - Smaller, lower power, more robust
- Flash bits wears out after 1000's of accesses
  - Wear leveling: remap data to less used blocks

Magnetic disk

- Nonvolatile, rotating magnetic storage
- Data is stored in sectors
  - 512 bytes in size
- Data access time includes:
  - Seek time: move the heads to the right track
    - Typically, 4-5ms
  - Rotational time: wait till head is over the right sector
    - = ½ /rotation frequency, e.g. ½/(15000/60)=2ms
  - Data transfer time
    - E.g. 512B/ 100MB/s = 0.005ms

e.g. Transfer 512Bytes



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

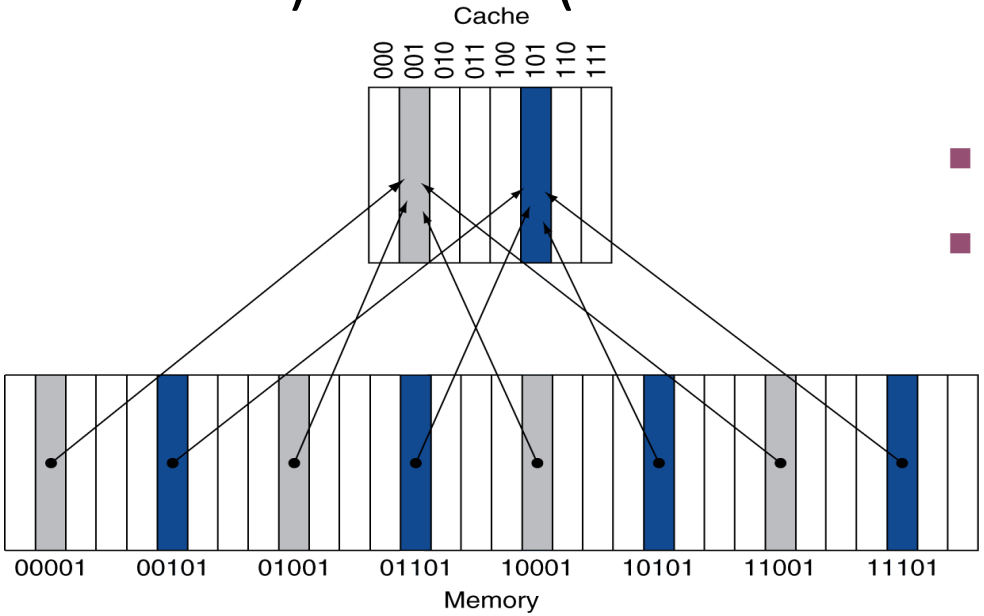
We will focus on cache memory only

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

- How to check if an access has previously been cached?
  - Location in cache determined by address
- Direct mapped: only one choice
  - (Block address) modulo (#Blocks in cache)



- #Blocks is a power of 2
- Use low-order address bits

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

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

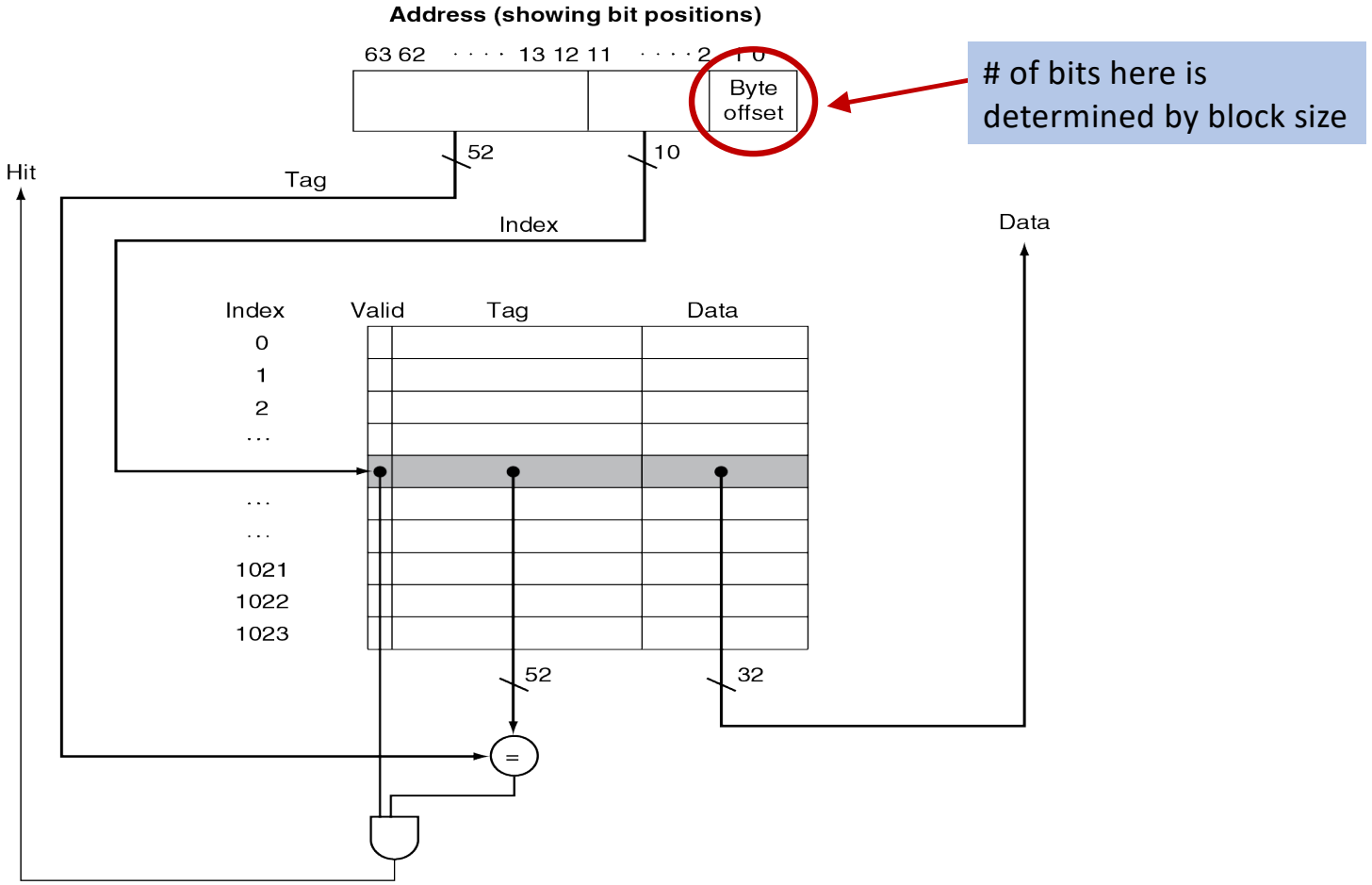
Index	V	Tag	Data
000	**Y**	**10**	**Mem[10000]**
001	N		
010	Y	11	Mem[11010]
011	**Y**	**00**	**Mem[00011]**
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
18	10 010	Miss	010

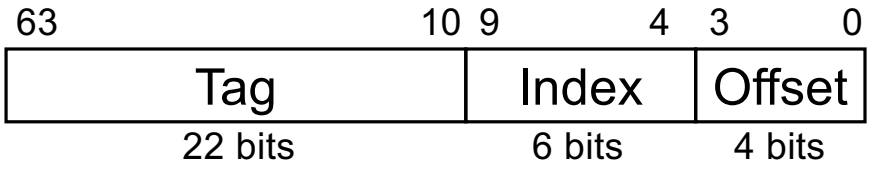
Index	V	Tag	Data
000	Y	10	Mem[10000]
001	N		
010	**Y**	**10**	**Mem[10010]**
011	Y	00	Mem[00011]
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Hardware of direct mapped cache



Example: Larger Block Size

- 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



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

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

Write-Through vs write-back

- Write-through: On write hit, update memory upon write
  - But, writes take longer
  - Solution: write buffer
    - Holds data to be written to memory so CPU continues immediately
    - Only stalls on write if write buffer is already full
- 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

Measuring Cache Performance

$$\text{Memory stall cycles}$$

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

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

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
- 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

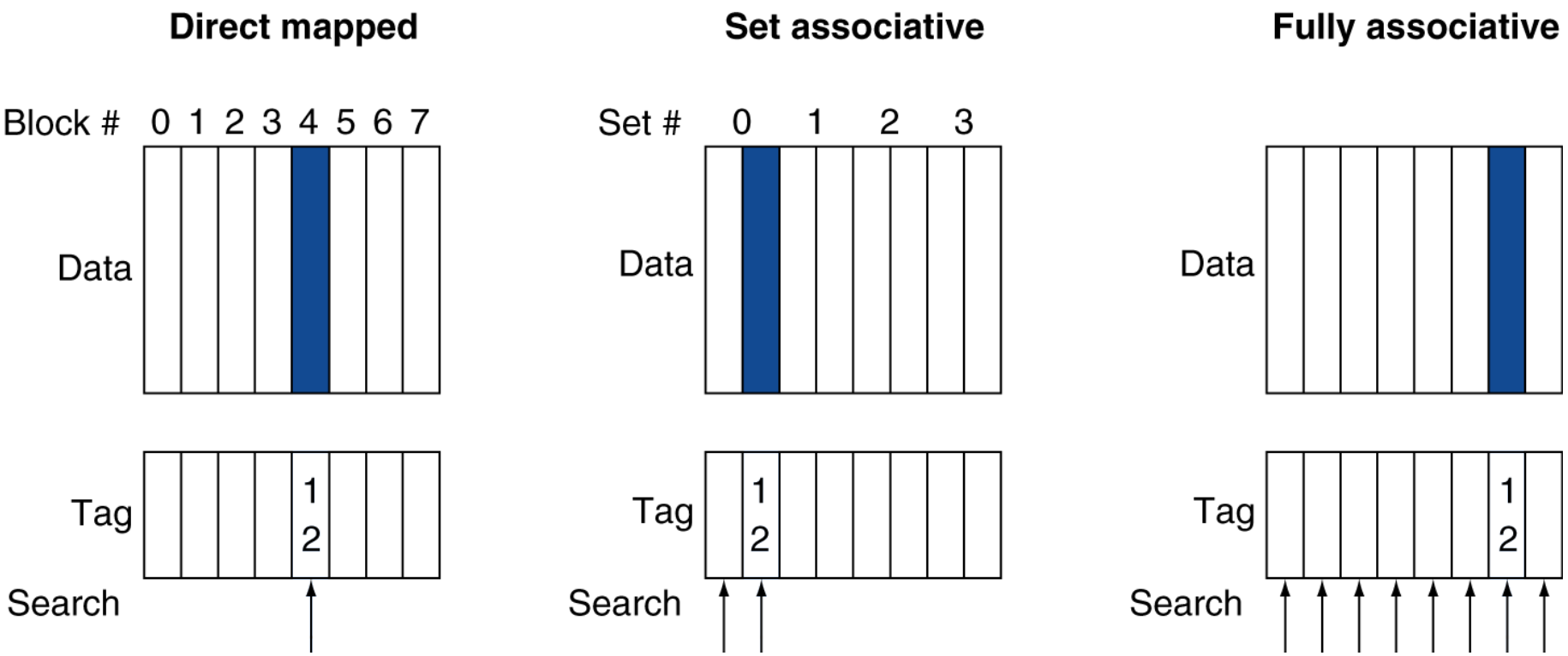
Average Access Time

- 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

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
  - Divide cache into sets each of which 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)

Associative Cache Example



Cache locations of a memory block with address 12

Associativity Example

- Direct mapped

Block access sequence: 0, 8, 0, 6, 8

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]	

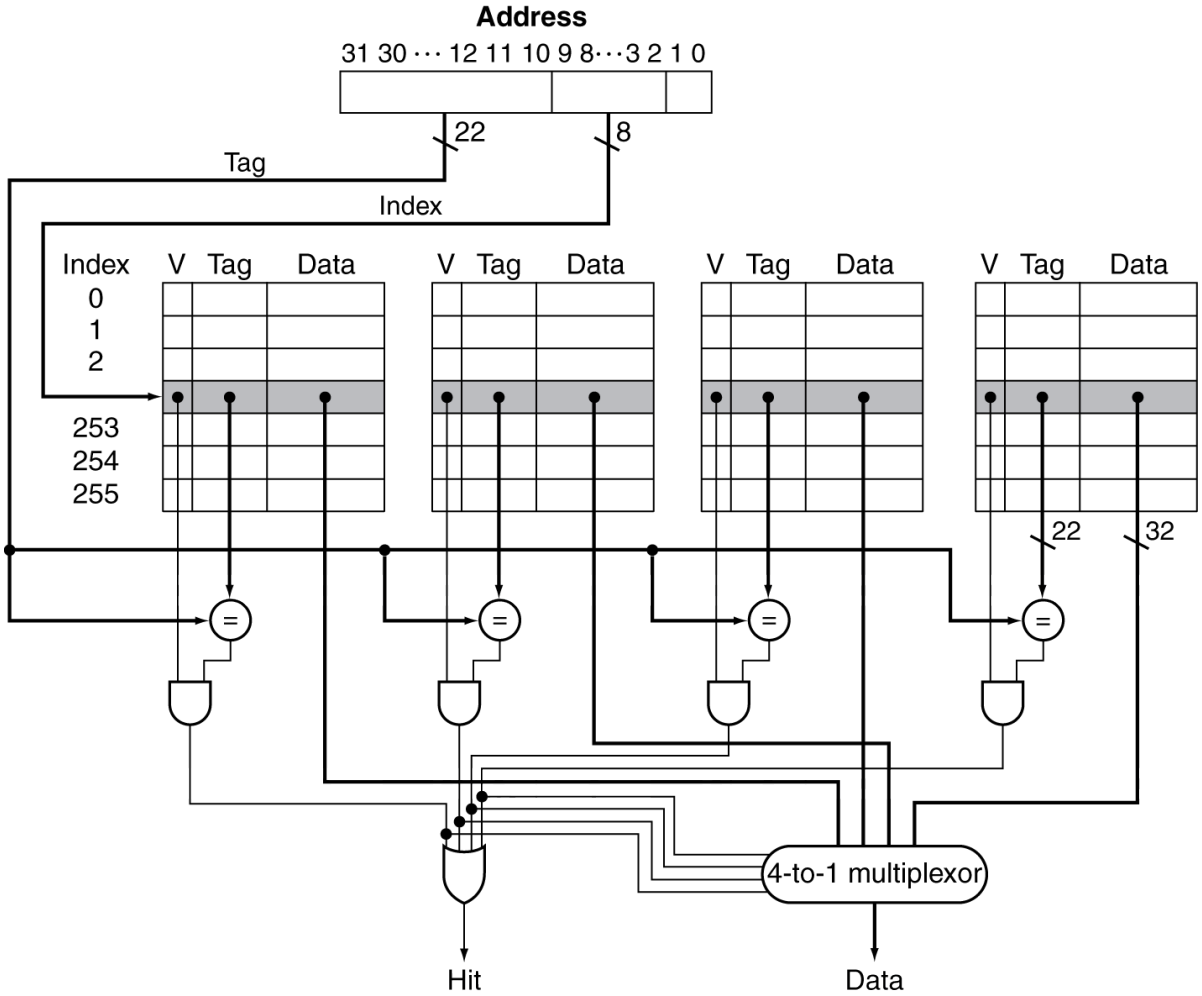
- 2-way set associative

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%

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

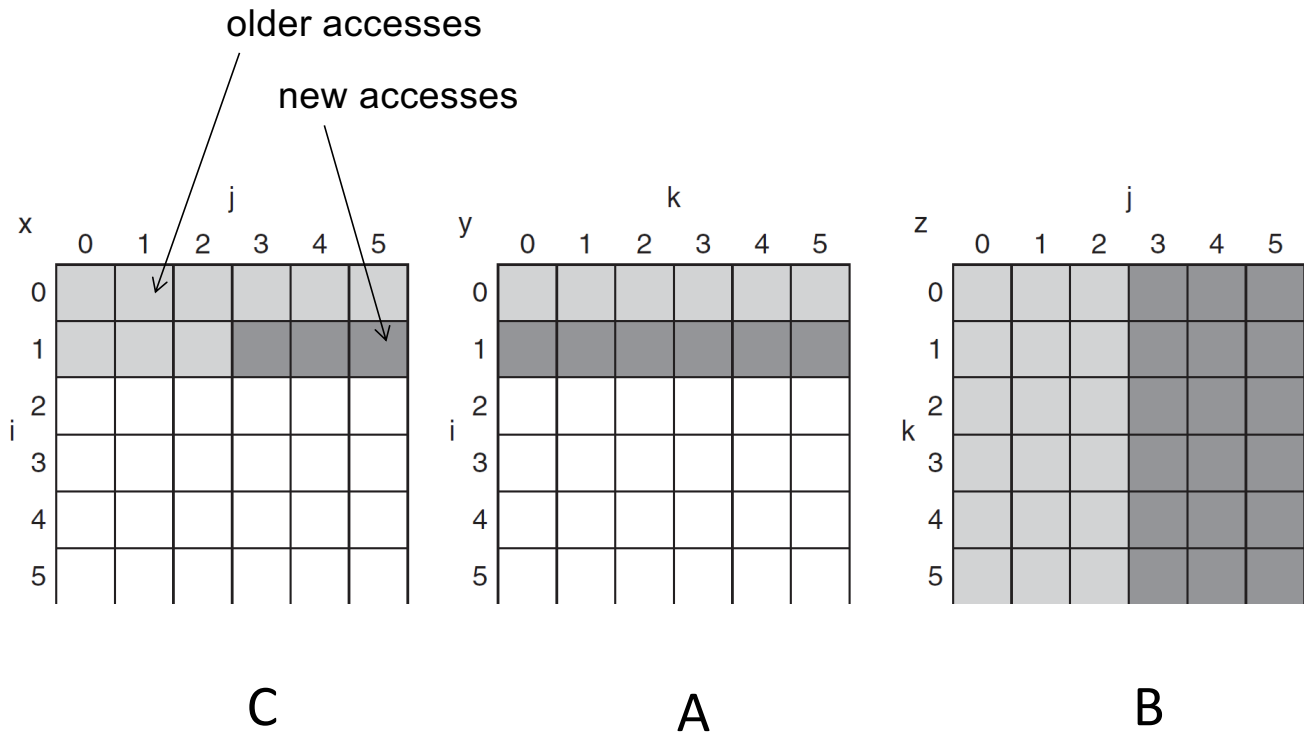
- Primary cache attached to CPU
  - Small, but fast
- Level-2 cache services misses from primary cache
  - Larger, slower, but still faster than main memory
- Main memory services L-2 cache misses
- High-end systems include L-3 cache

Software Optimization via Blocking

- Goal: maximize accesses to data before it is replaced
- Consider inner loops of DGEMM:

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

DGEMM Access Pattern

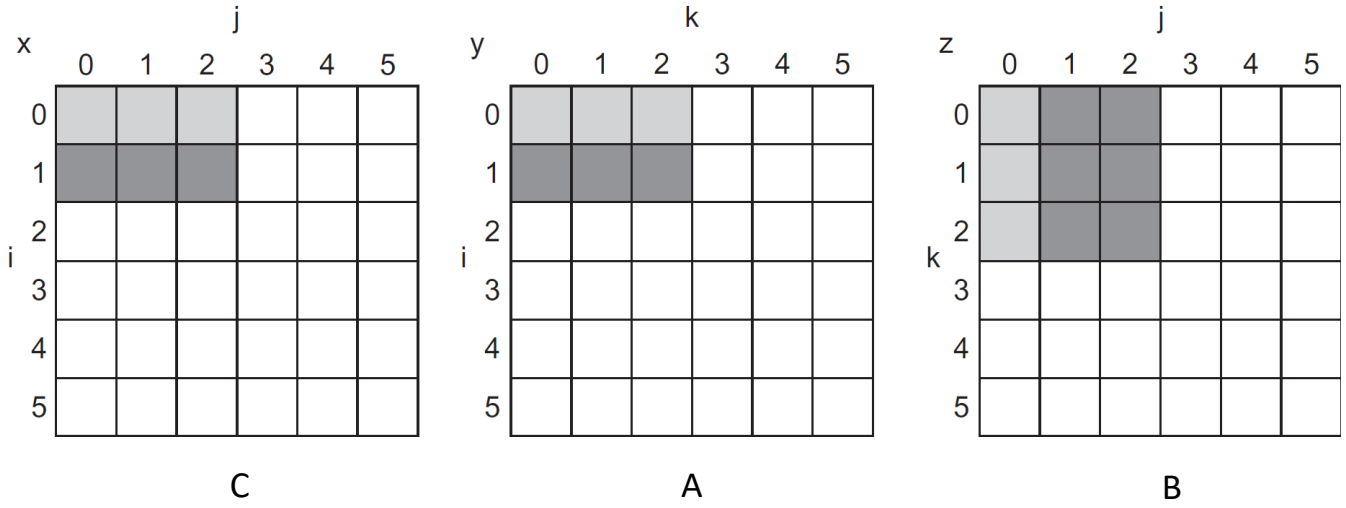


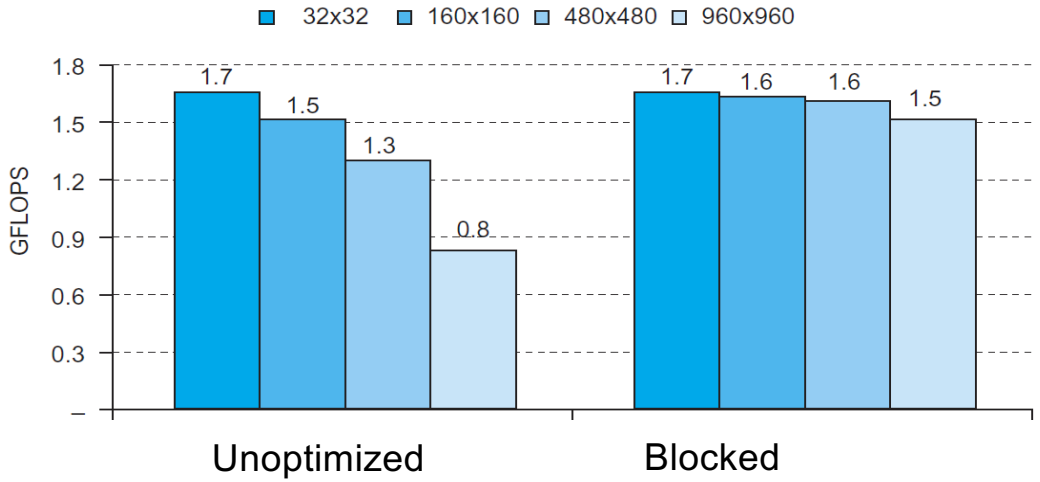
Blocked DGEMM

```
#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 cij = C[i+j*n]; /* cij = C[i][j] */
     for( int k = sk; k < sk+BLOCKSIZE; k++ )
         cij += A[i+k*n] * B[k+j*n];/* cij+=A[i][k]*B[k][j] */
     C[i+j*n] = cij;/* C[i][j] = cij */
    }
}

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