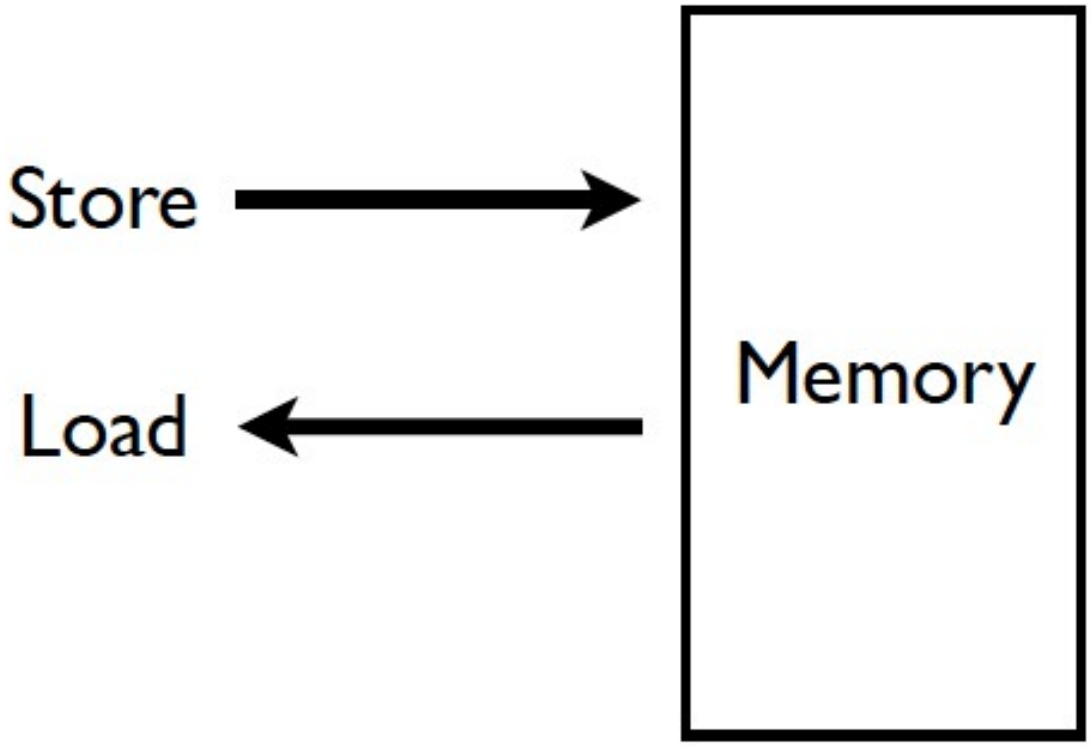
CENG311 Computer Architecture

Memory System

IZTECH, Fall 2023

21 December 2023

Memory (Programmer's View)



Memory

Static RAM

- Relatively fast, only one data word at a time
- Expensive (one bit costs 6 transistors)

Dynamic RAM

- Slower, one data word at a time, reading destroys content (refresh), needs special process for manufacturing
- Cheap (one bit costs only one transistor plus one capacitor)

Other storage technology (flash memory, hard disk, tape)

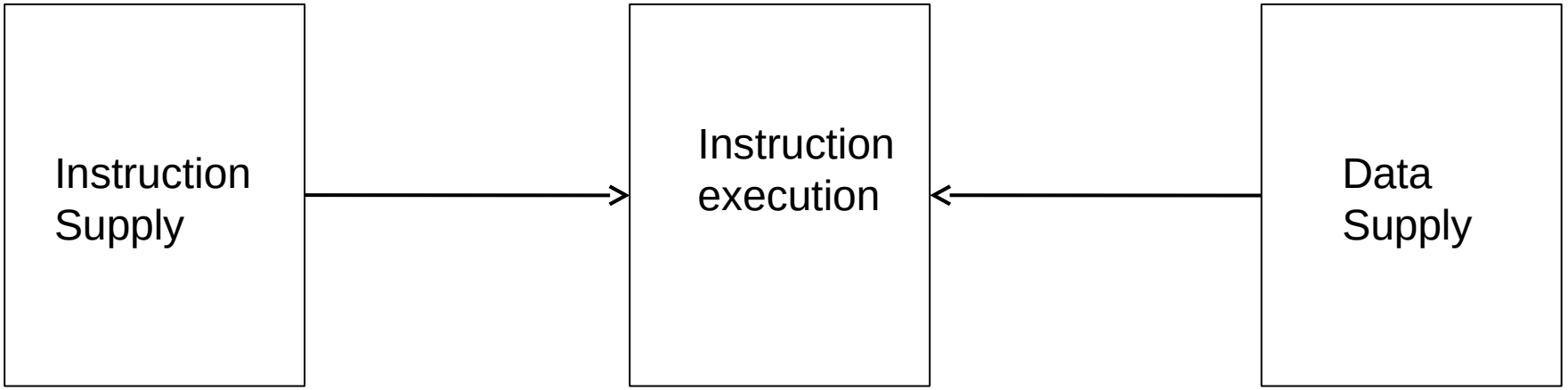
- Much slower, access takes a long time, non-volatile
- Very cheap (no transistors directly involved)

Memory System Performance

Memory system performance is largely captured by two parameters, latency and bandwidth

- Latency: the time from the issue of a memory request to the time the data is available at the processor
- Bandwidth: the rate at which data can be pumped to the processor by the memory system
- Latency vs bandwidth: Fire-hose analogy
  - If the water comes out of the hose 2 seconds after the hydrant is turned on, the **latency** of the system is 2 seconds (put out a fire immediately)
  - If the hydrant delivers water at the rate of 1 gallon/second, the **bandwidth** of the system is 1 gallon/second (large fires)

Ideal Memory System



- Zero latency access
- Infinite capacity
- Infinite bandwidth
- Zero cost

Problem about Ideal Memory

Ideal memory's requirements oppose each other

Bigger is slower

Takes longer to determine the location

Faster is more expensive

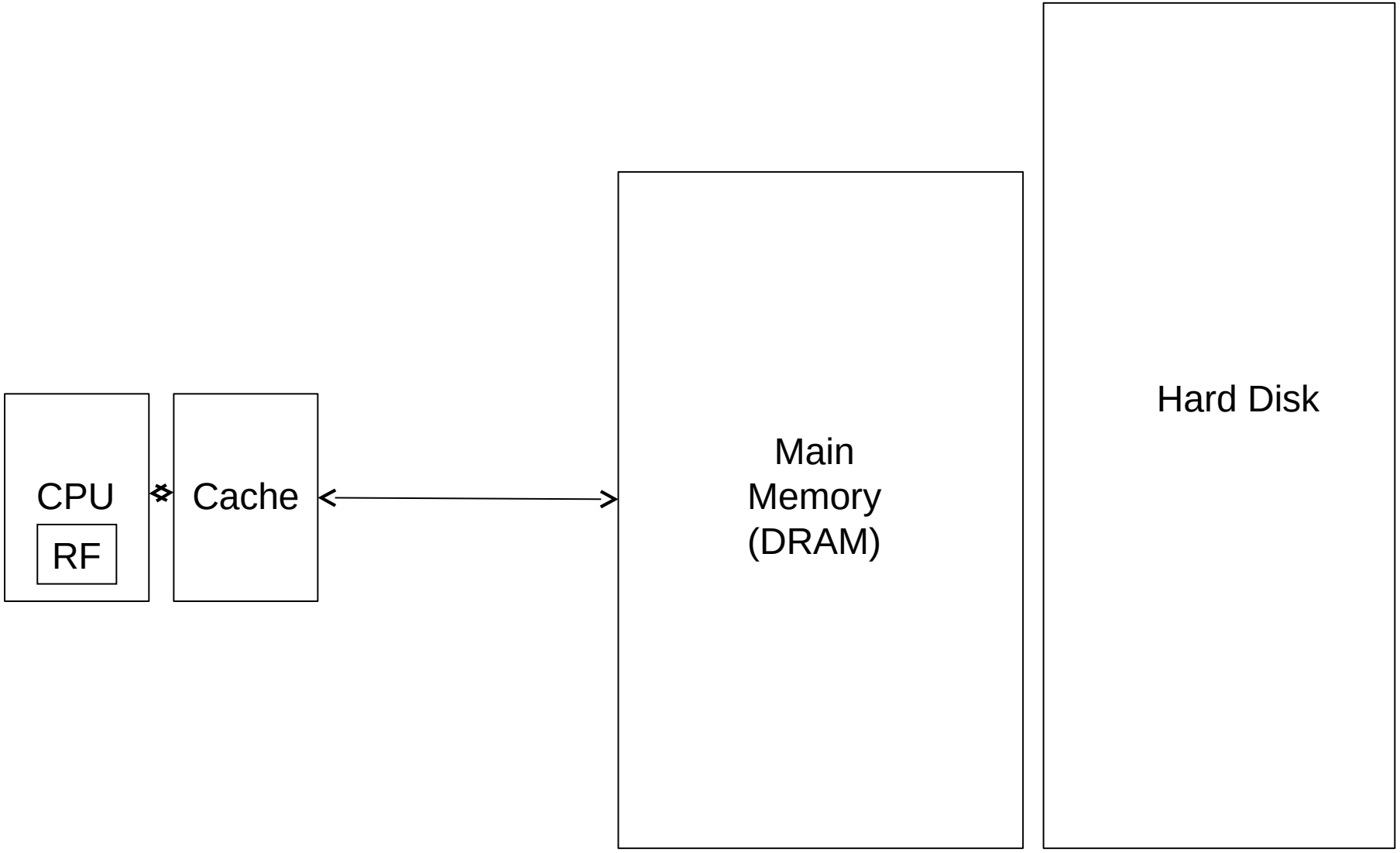
Memory technology: SRAM vs. DRAM vs. Disk vs. Tape

Higher bandwidth is more expensive

Need more banks, more ports, higher frequency, or faster technology

Multiple levels of hierarchy to ensure most of the data the processor needs in faster levels

Memory Hierarchy



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

Spatial locality

- Items near those accessed recently are likely to be accessed soon
- e.g., sequential instruction access, array data

Locality Example

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

Data

- Reference array elements in succession
- Reference sum each iteration

Instructions

- Reference instructions in sequence
- Cycle through loop repeatedly

Locality Example

Does this function have good locality?

```
int sumarrayrows(int a[M][N])
{
    int i, j, sum = 0;

    for (i = 0; i < M; i++)
        for (j = 0; j < N; j++)
            sum += a[i][j];
    return sum;
}
```

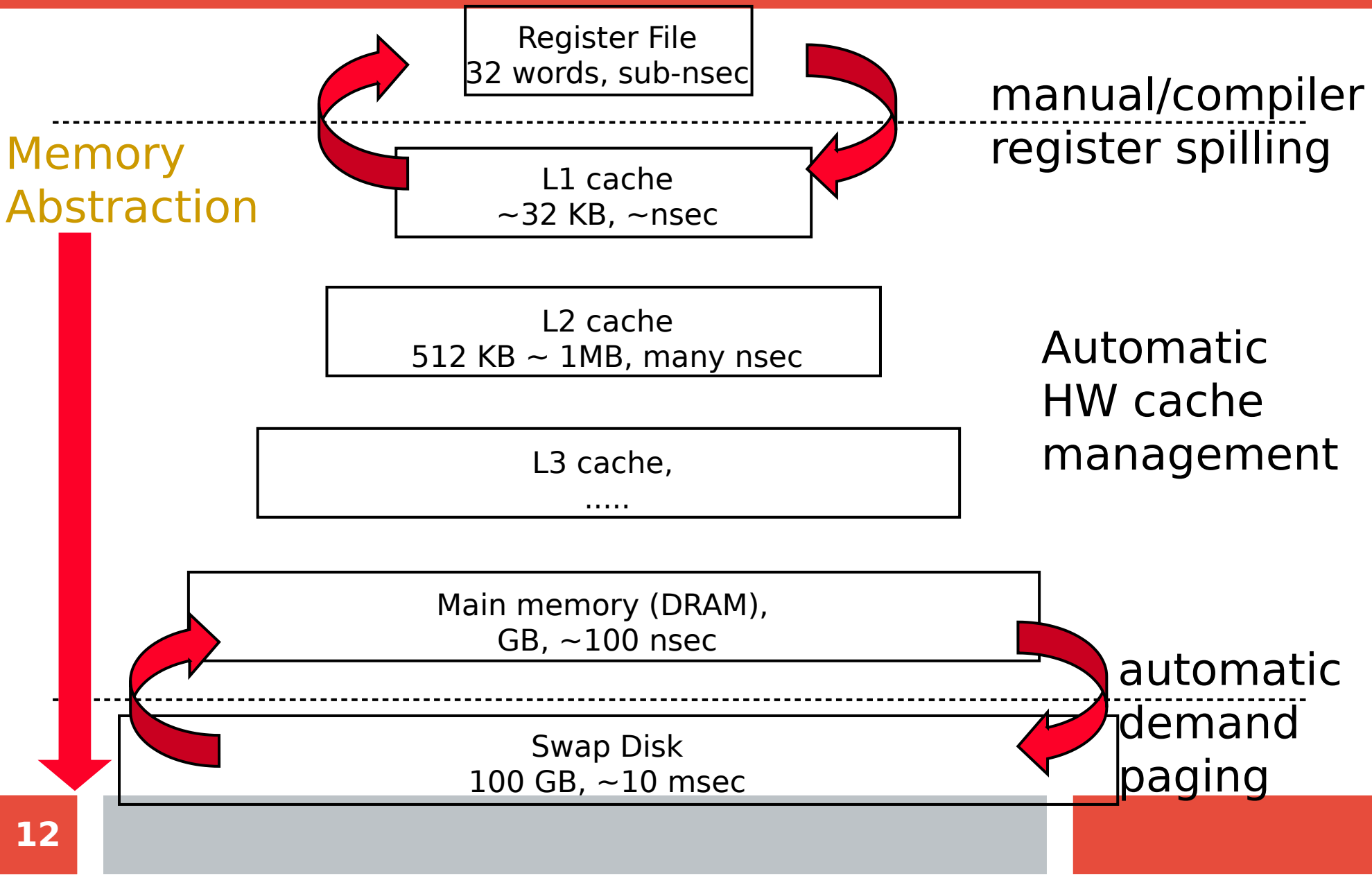
Locality Example

Does this function have good locality?

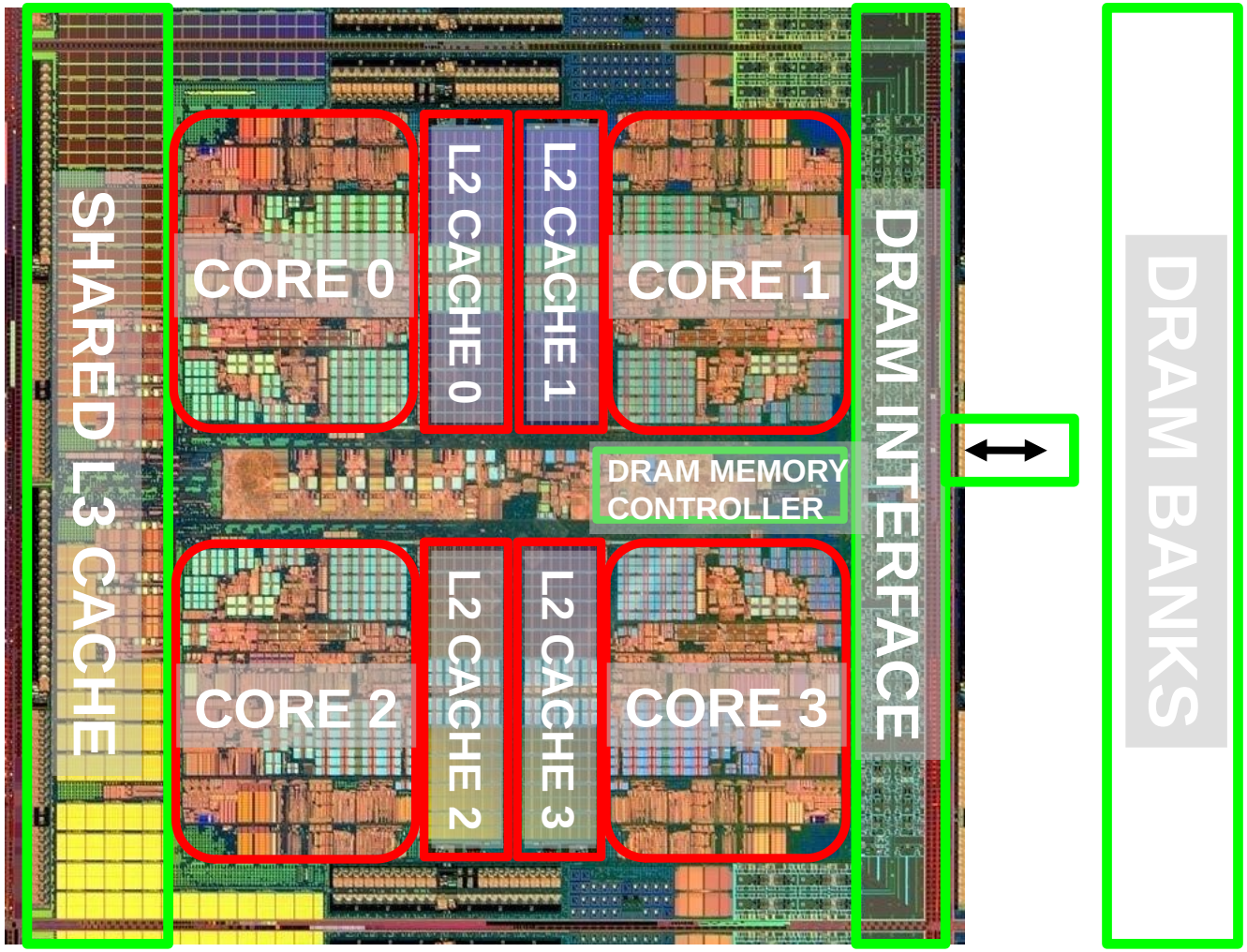
```
int sumarraycols(int a[M][N])
{
    int i, j, sum = 0;

    for (j = 0; j < N; j++)
        for (i = 0; i < M; i++)
            sum += a[i][j];
    return sum;
}
```

Modern Memory Hierarchy



Memory in a Modern System



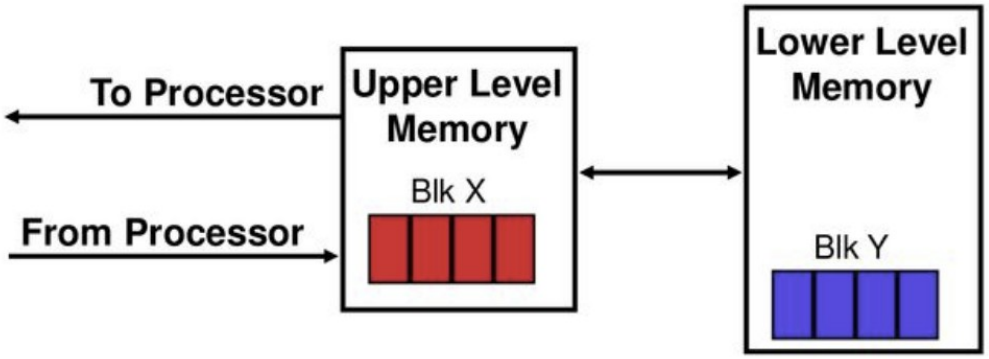
How Memory Hierarchy Works?

Temporal locality:

Keep most recently accessed data items closer to the processor

Spatial locality:

Move blocks (a set of data items) to the upper levels



Memory Hierarchy Terminology

Cache hit: data is in some block in the cache

Hit rate: The fraction of memory accesses found in the cache

Cache miss: data is not in the cache so needs to be retrieved from a block in the lower level

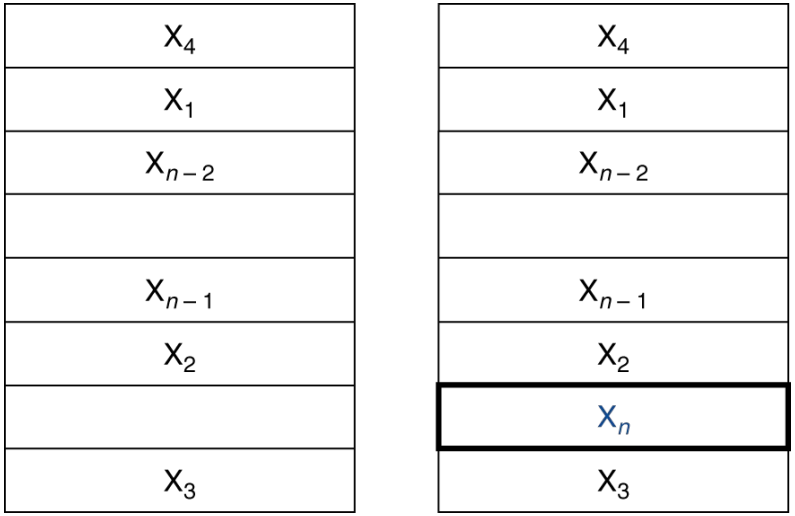
Miss rate: 1- (Hit rate)

Cache Memory

Given accesses X_1, …, X_{n-1}, X_n

How do we know if the data is present?

Where do we look?



a. Before the reference to X_n b. After the reference to X_n

Cache Block (Cache Line)

In order to exploit the principle of spatial locality, the system uses an effectively wider interconnect to access data

A memory access operates on blocks of data instead of individual data items: cache block or cache line

If a cache block stores 16 floats,
sum+=z[0] operation reads data z[0] – z[15] from memory to cache,
next 15 additions use elements that are already in cache

```
float z[1000];
. . .
sum = 0.0;
for (i = 0; i < 1000; i++)
   sum += z[i];
```

Cache Mappings

Direct mapped

Each cache line has a unique location in the cache to which it will be assigned

N-way set associative

Each cache line can be placed in one of n different locations in the cache

Fully associative

A new line can be placed at any location in the cache

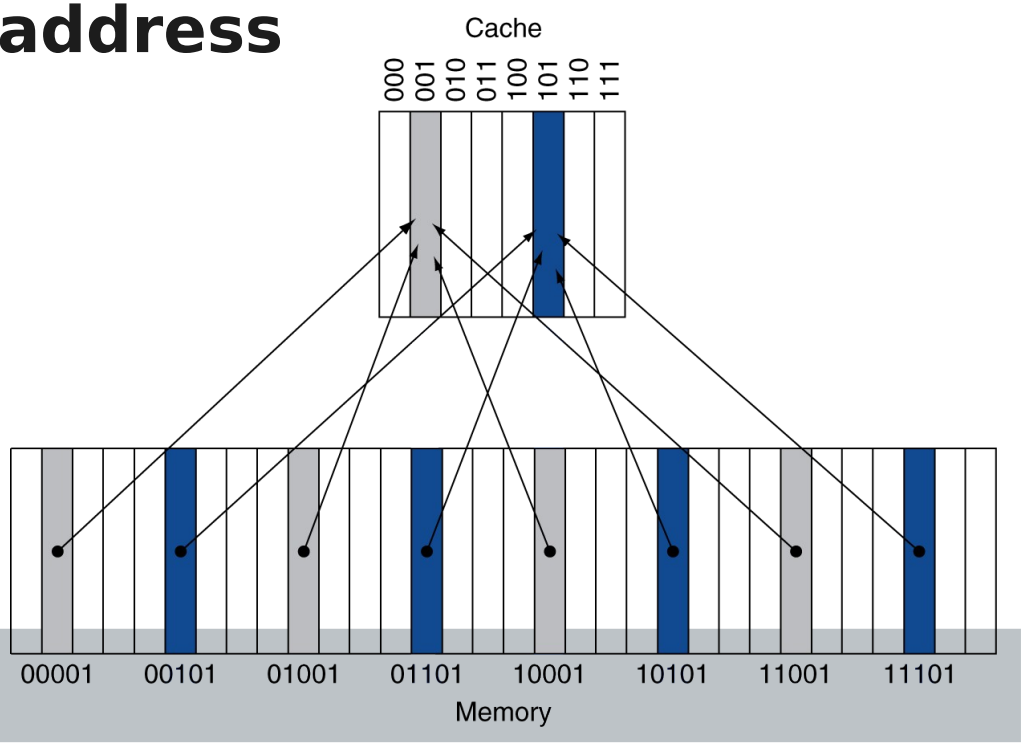
Direct-Mapped Cache

To find a block in a direct-mapped cache:

(Block address) mod (Number of blocks in the cache)

Low-order $\log_2$ (number of blocks) bits of the address

If 8 blocks in the cache, block number= 3 lowest bits of the block address



Tags and Valid Bits

Each cache location can contain the contents of a number of different memory locations

How do we know whether the requested data is in the cache or not?

Tag: contains upper portion of the address (not used as an index)

What if there is no data in a location?

- Valid bit: 1 = present, 0 = not present
- Initially 0

Blocks and Addressing the Cache

Memory is logically divided into fixed-size blocks

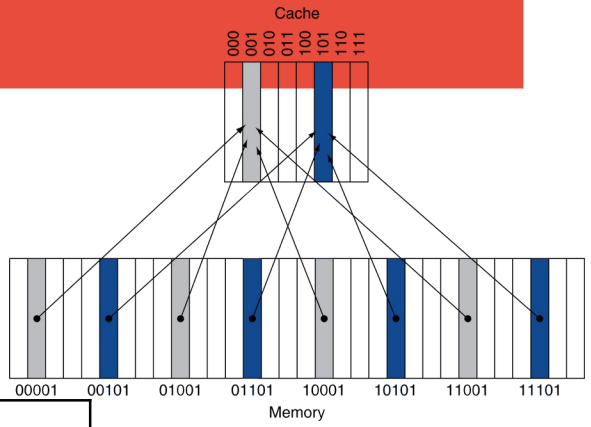
Each block maps to a location in the cache, determined by the index bits in the address

Cache access:

1) index into the tag and data stores with index bits in address

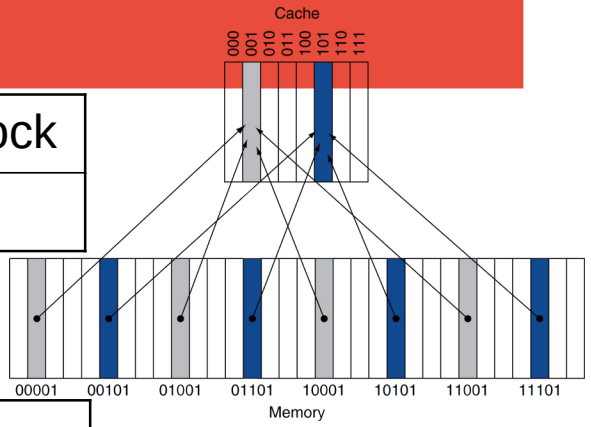
2) check valid bit in tag store

3) compare tag bits in address with the stored tag in tag store



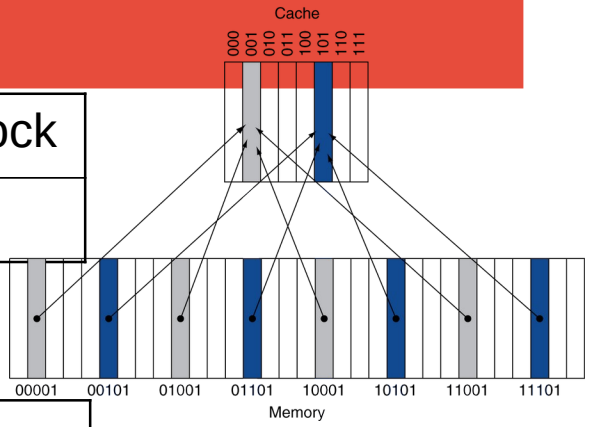
Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

Word addr	Binary addr	Hit/miss	Cache block
22	10 110	Miss	110



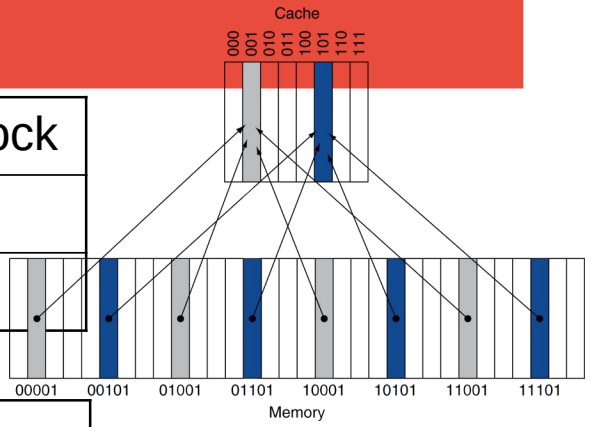
Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	**Y**	**10**	**Mem[10110]**
111	N		

Word addr	Binary addr	Hit/miss	Cache block
26	11 010	Miss	010



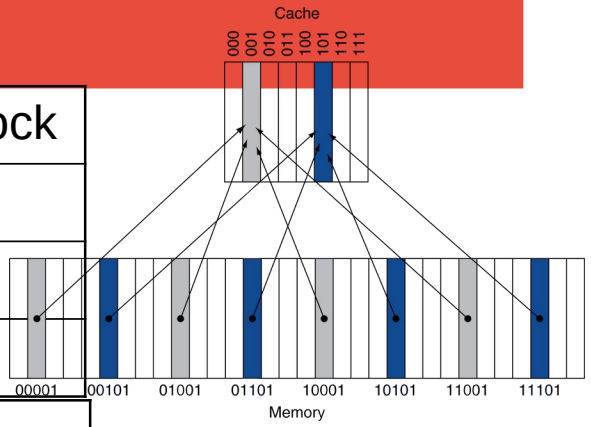
Index	V	Tag	Data
000	N		
001	N		
010	**Y**	**11**	**Mem[11010]**
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Word addr	Binary addr	Hit/miss	Cache block
22	10 110	Hit	110
26	11 010	Hit	010



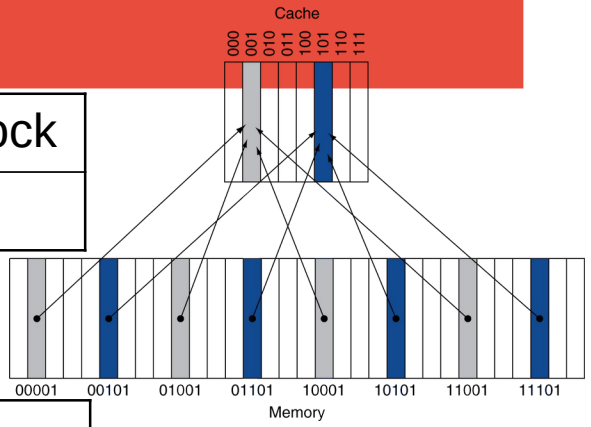
Index	V	Tag	Data
000	N		
001	N		
010	Y	11	Mem[11010]
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

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		

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

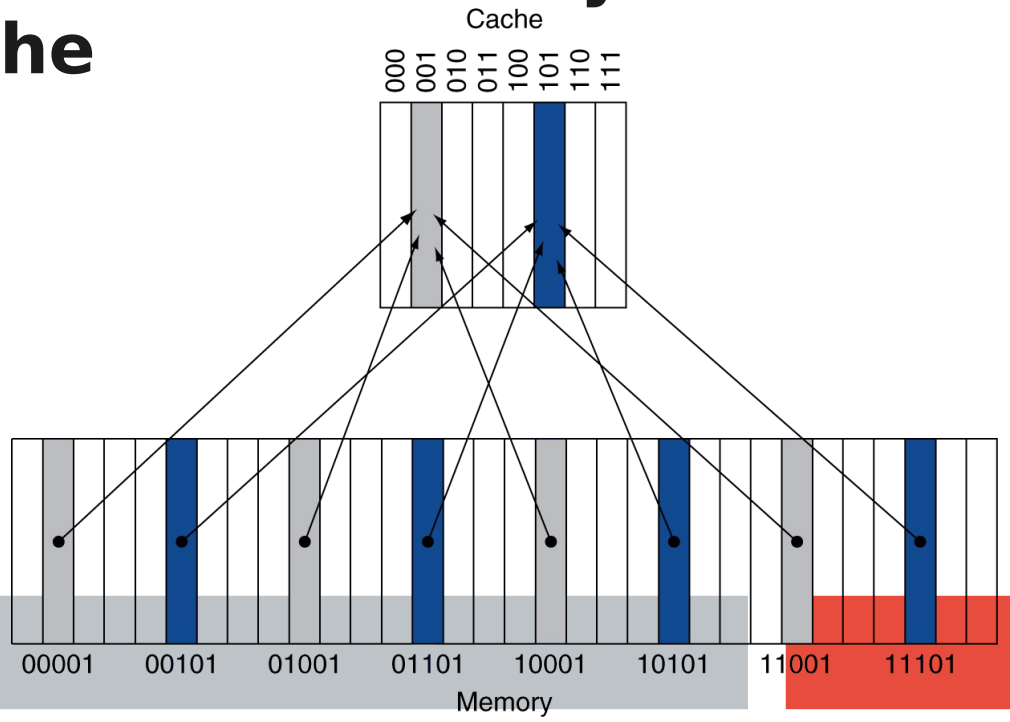


Index	V	Tag	Data
000	Y	10	Mem[10000]
001	N		
010	**Y**	**10**	**Mem[10010]**
011	Y	00	Mem[00011]
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Direct-Mapped Caches

Addresses with the same index contend for the same location, two blocks in memory that map to the same index in the cache cannot be present in the cache at the same time

Example: Addresses 0 and 8 always conflict in direct-mapped cache



Associative Caches

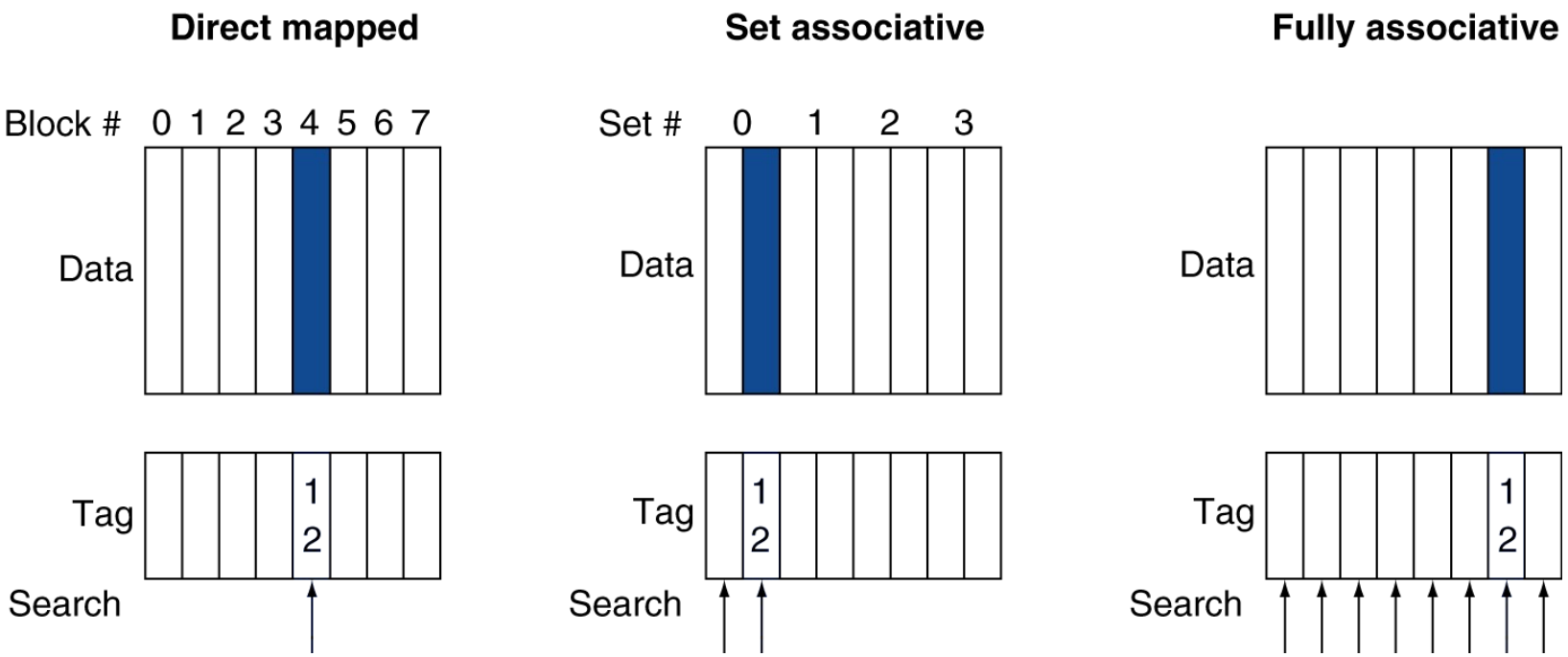
Fully associative

- 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

- Each set 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 with 8 Entries

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 caches with 4 one-word blocks

Direct mapped, 2-way set associative, fully associative

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

Direct-Mapped

0, 8, 0, 6, 8

V	Tag	Data
N		
N		
N		
N		

2-way set associative

0, 8, 0, 6, 8

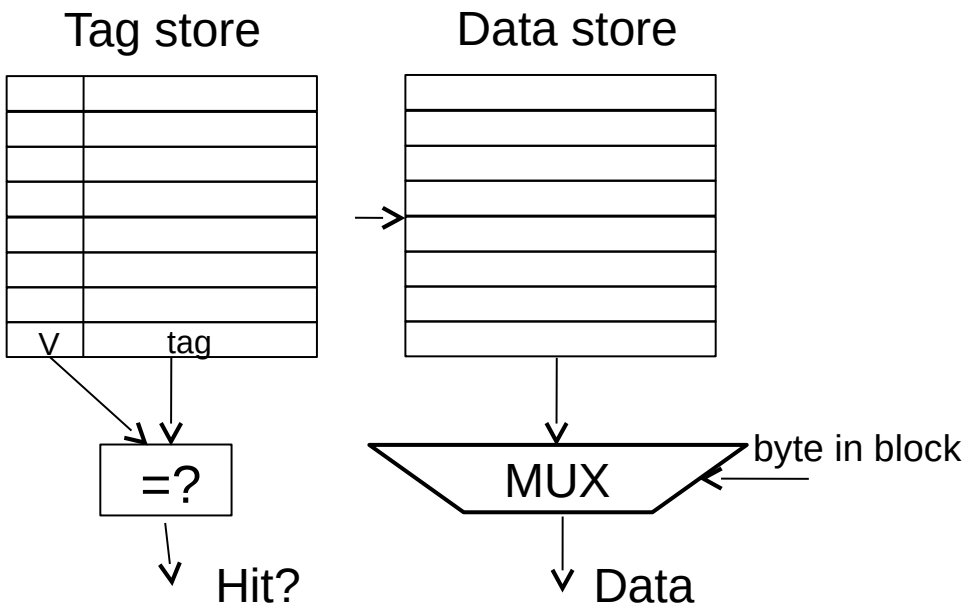
V	Tag	Data	V	Tag	Data
N			N		
N			N		

Fully associative

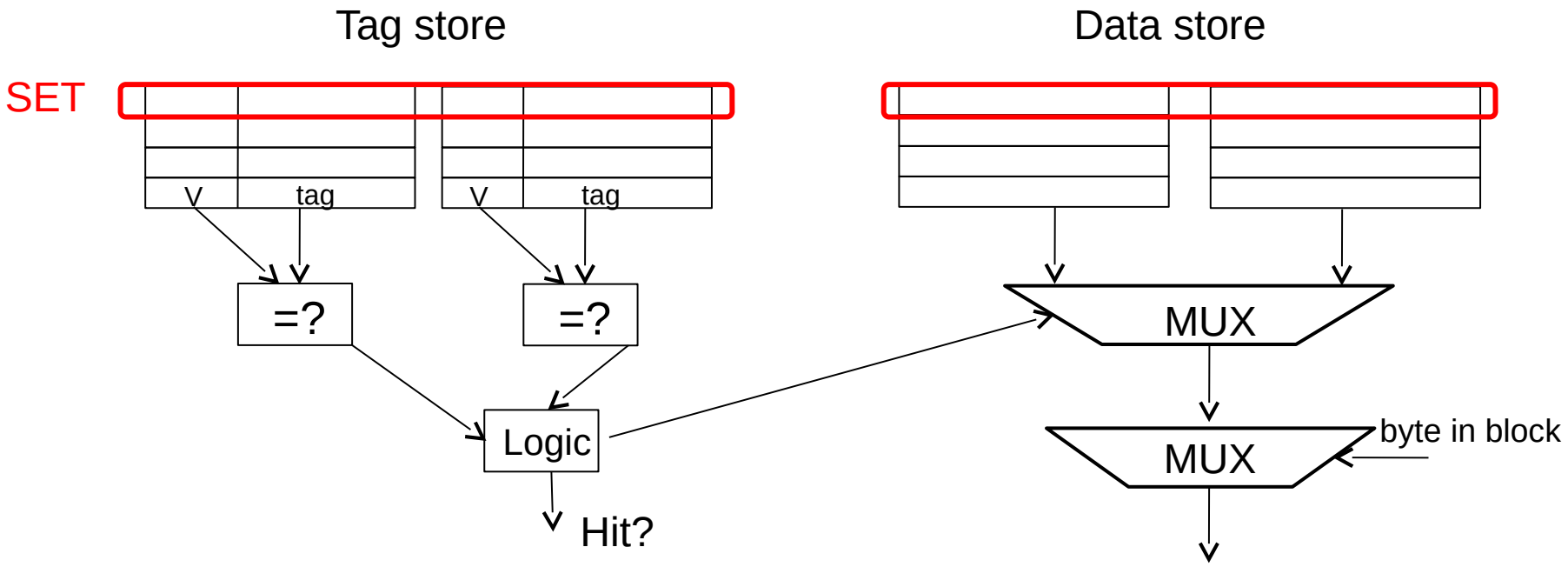
0, 8, 0, 6, 8

V	Tag	Data	V	Tag	Data	V	Tag	Data	V	Tag	Data
N			N			N			N		

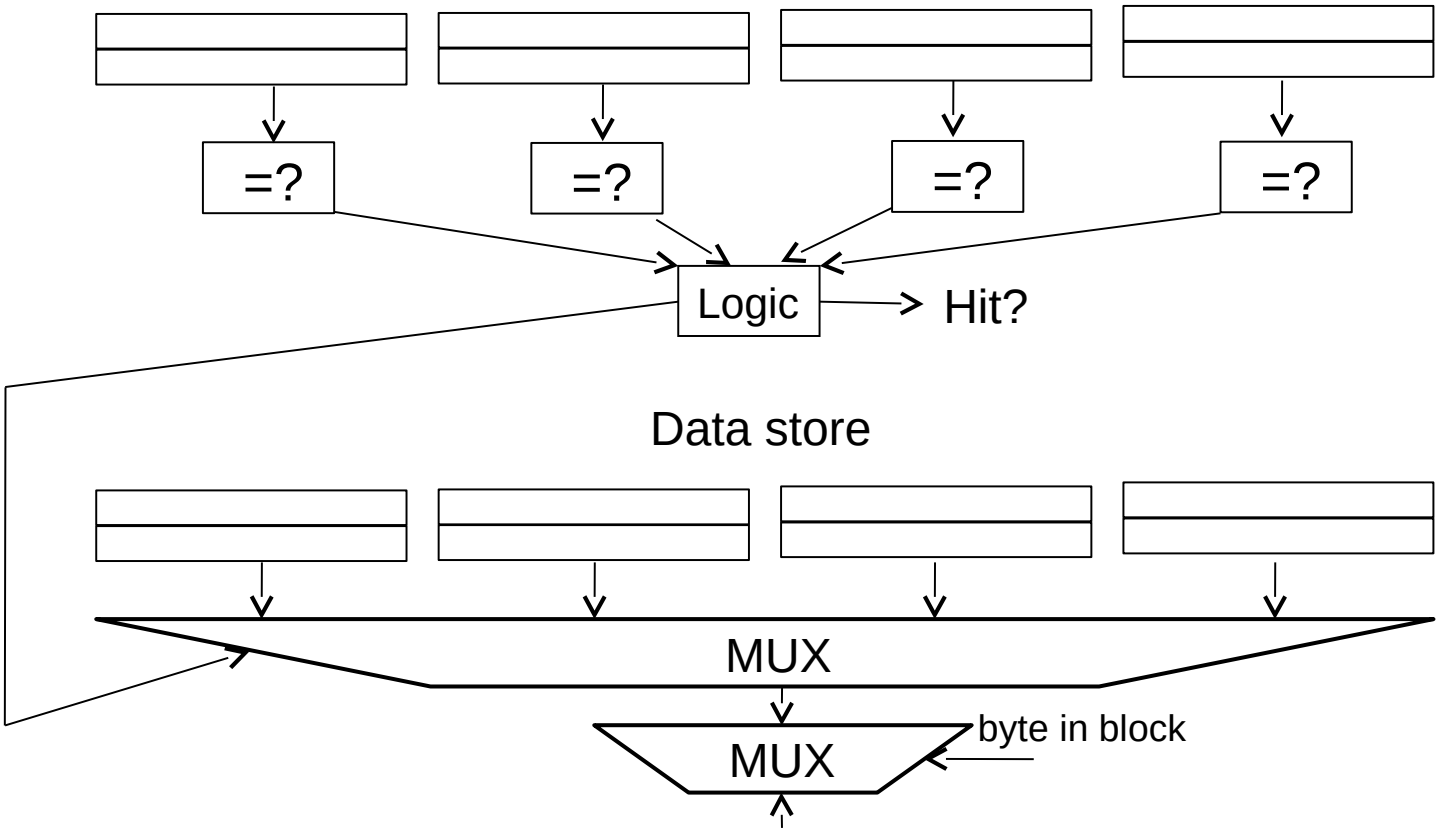
Direct-Mapped



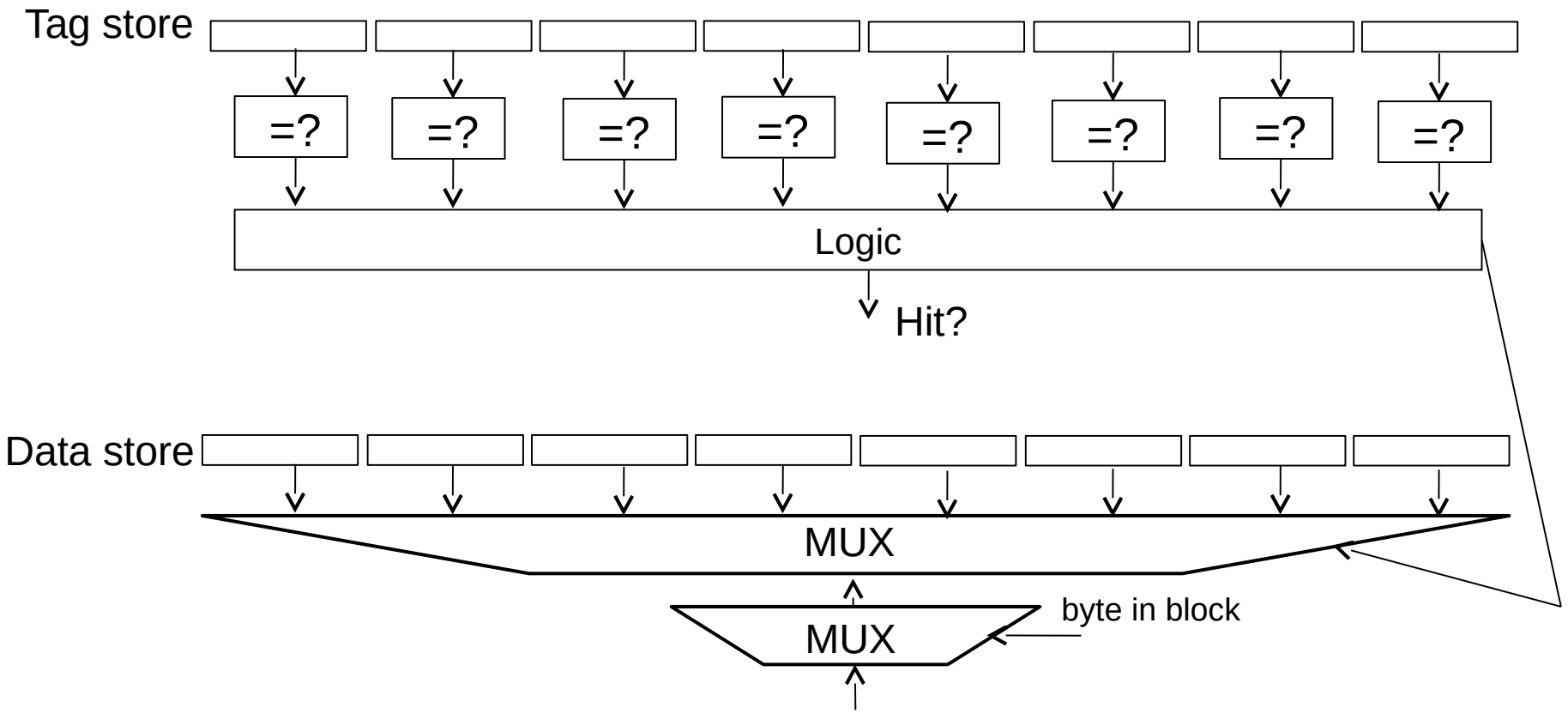
2-way Set Associative



4-way Set Associative



Fully Associative



References

Chapter 5.1 - 5.3

(Computer Organization and Design: The Hardware/Software Interface by Hennessy/Patterson, 5th edition)