

CS 33

Memory Hierarchy I

Random-Access Memory (RAM)

- **Key features**
 - **RAM** is traditionally packaged as a chip
 - basic storage unit is normally a **cell** (one bit per cell)
 - multiple RAM chips form a memory
- **Static RAM (SRAM)**
 - each cell stores a bit with a four- or six-transistor circuit
 - retains value indefinitely, as long as it is kept powered
 - relatively insensitive to electrical noise (EMI), radiation, etc.
 - faster and more expensive than DRAM
- **Dynamic RAM (DRAM)**
 - each cell stores bit with a capacitor; transistor is used for access
 - value must be refreshed every 10-100 ms
 - more sensitive to disturbances (EMI, radiation,...) than SRAM
 - slower and cheaper than SRAM

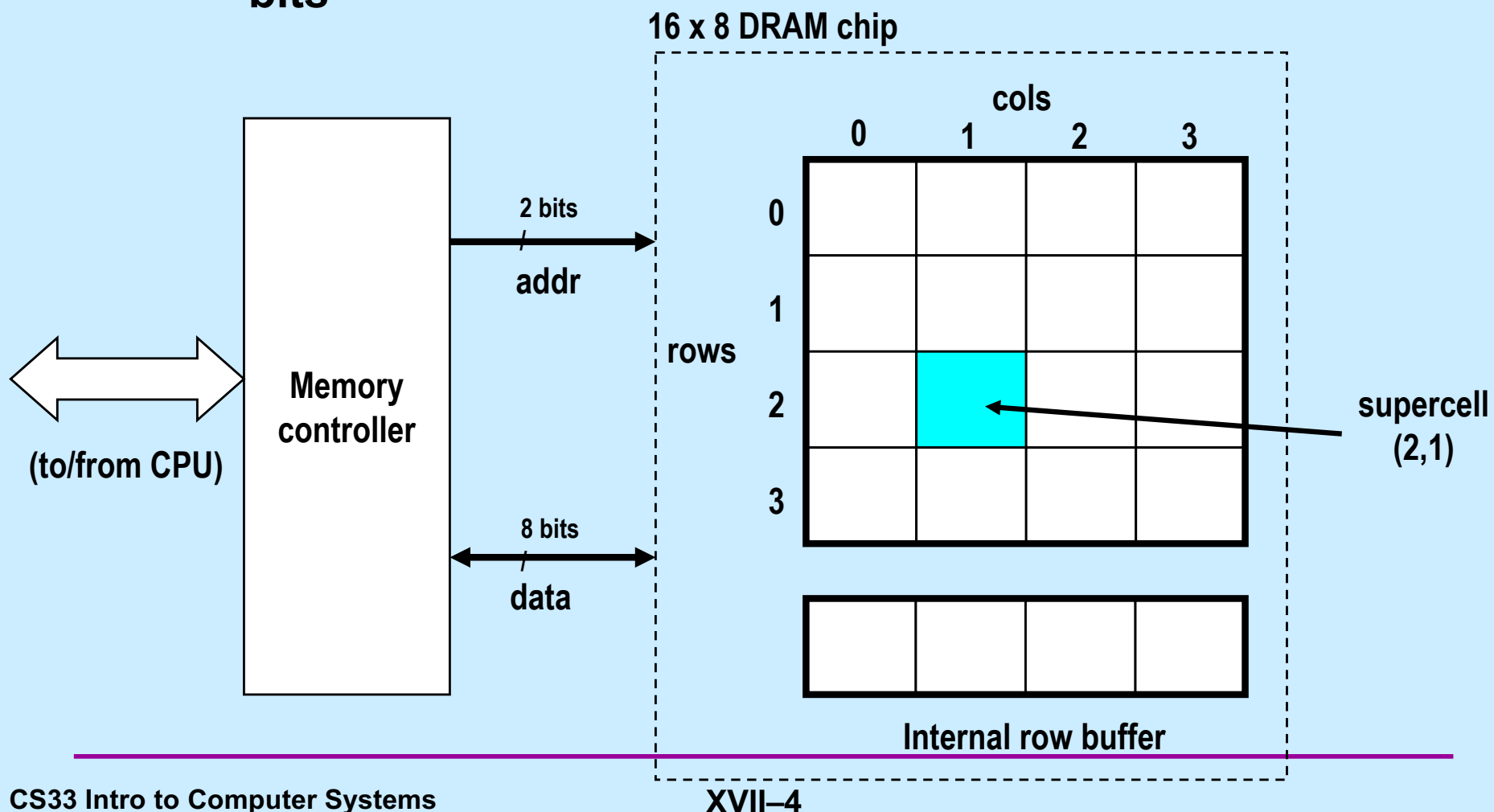
SRAM vs DRAM Summary

	Trans. per bit	Access time	Needs refresh?	Needs EDC?	Cost	Applications
SRAM	4 or 6	1X	No	Maybe	100x	Cache memories
DRAM	1	10X	Yes	Yes	1X	Main memories, frame buffers

- **EDC = error detection and correction**
 - to cope with noise, etc.

Conventional DRAM Organization

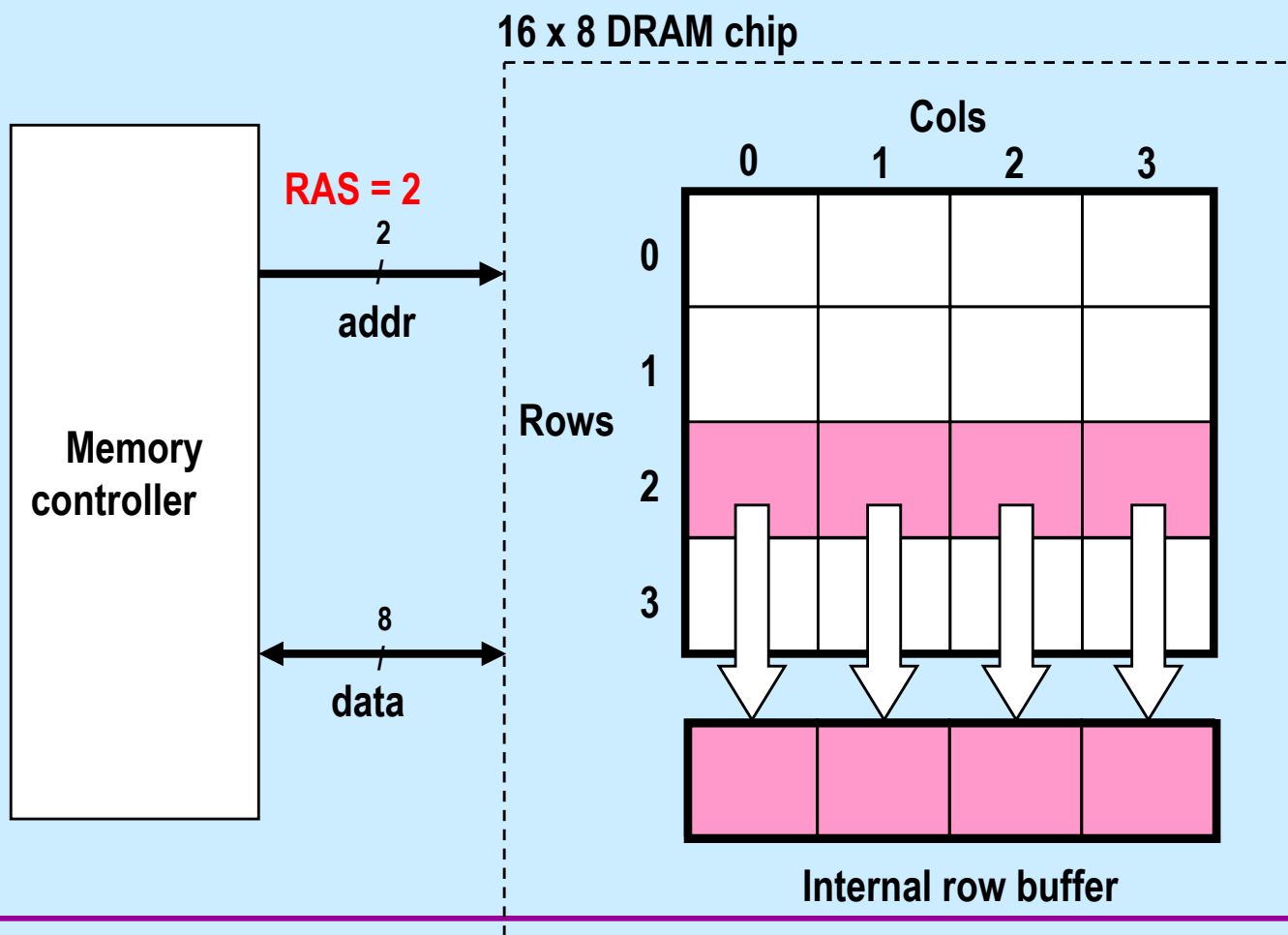
- $d \times w$ DRAM:
 - dw total bits organized as d **supercells** of size w bits



Reading DRAM Supercell (2,1)

Step 1(a): row access strobe (**RAS**) selects row 2

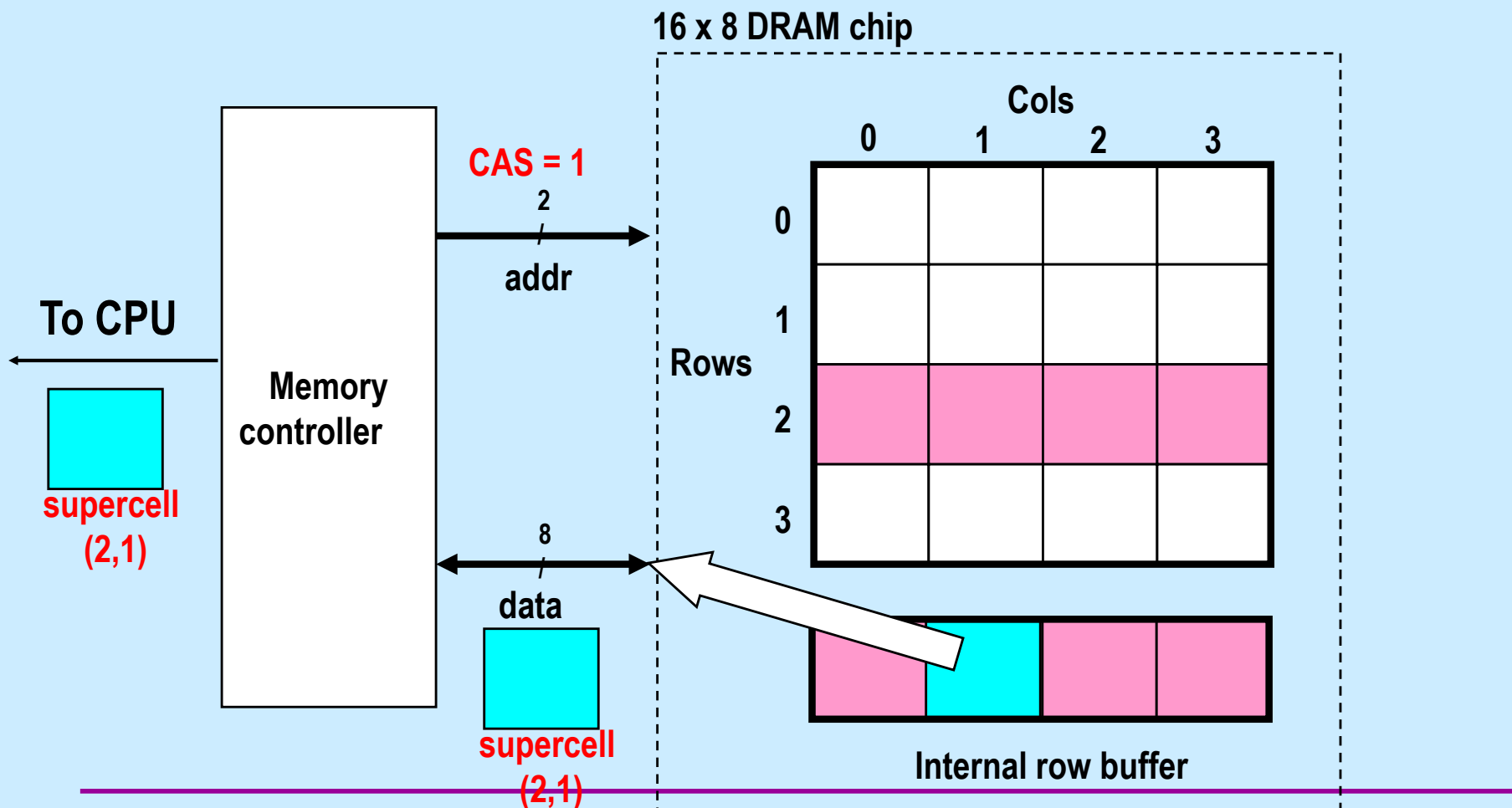
Step 1(b): row 2 copied from DRAM array to row buffer



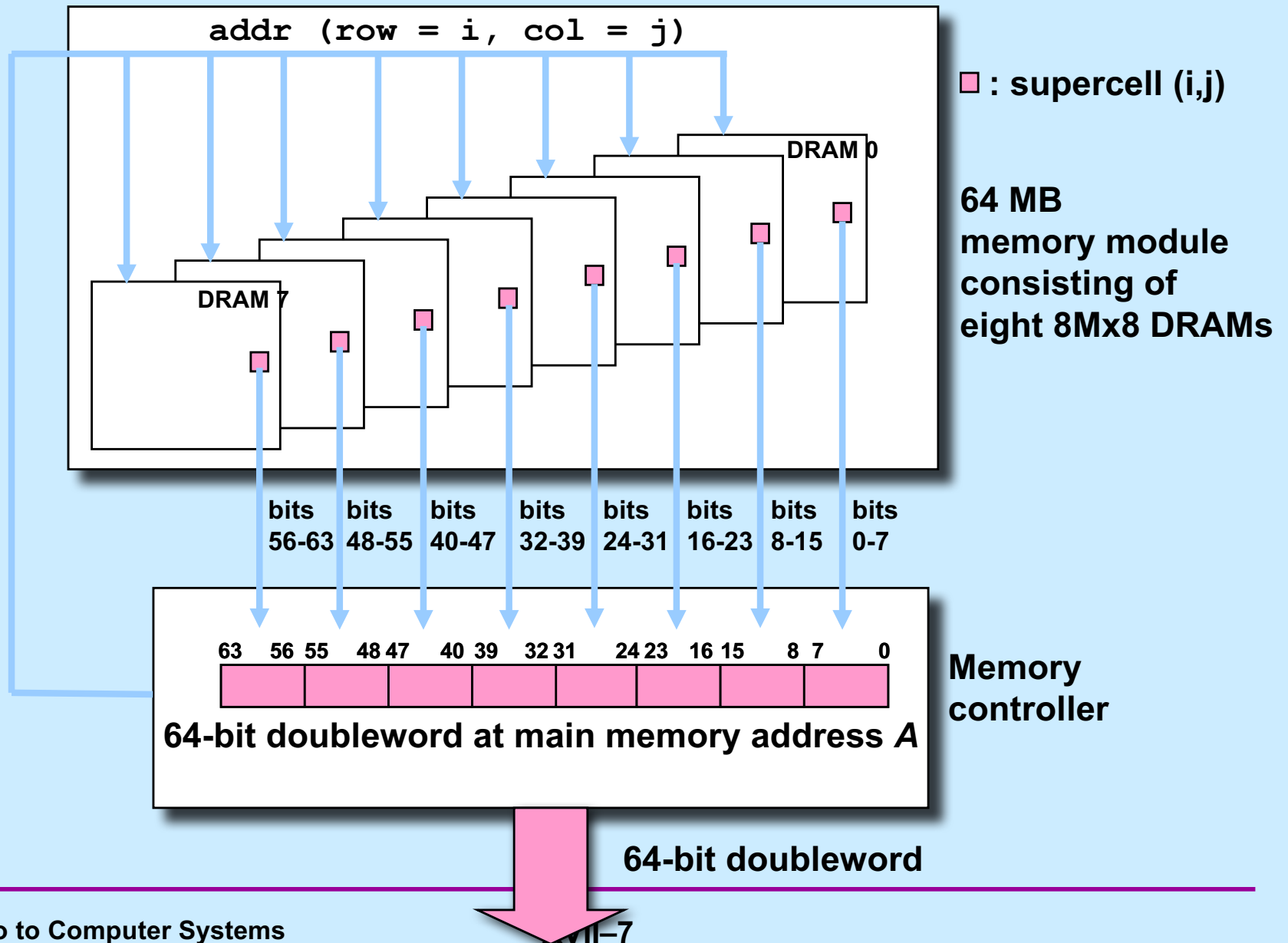
Reading DRAM Supercell (2,1)

Step 2(a): column access strobe (**CAS**) selects column 1

Step 2(b): supercell (2,1) copied from buffer to data lines, and eventually back to the CPU



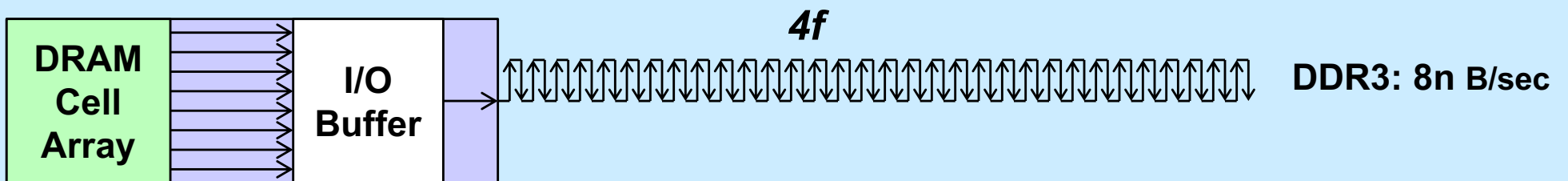
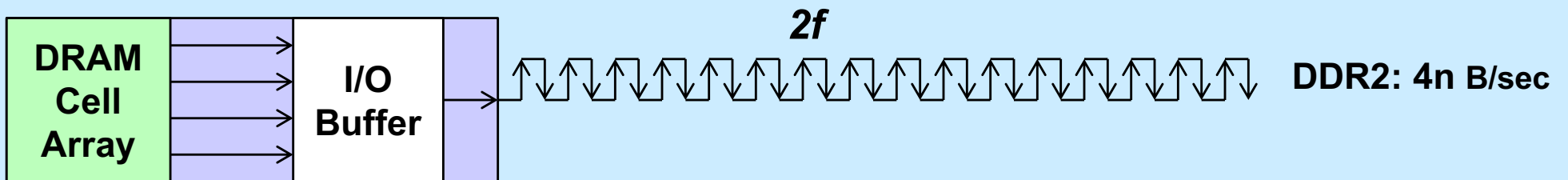
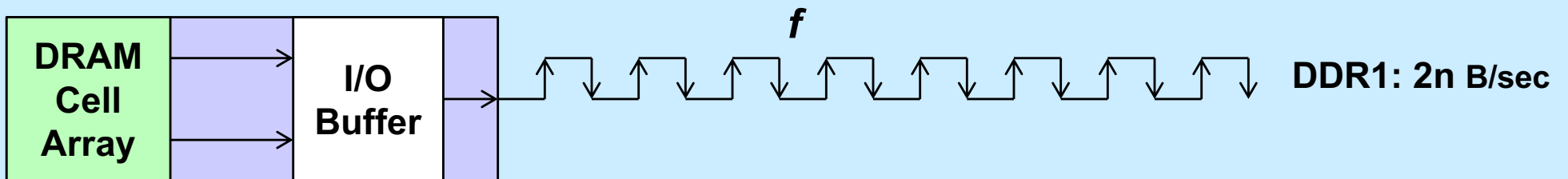
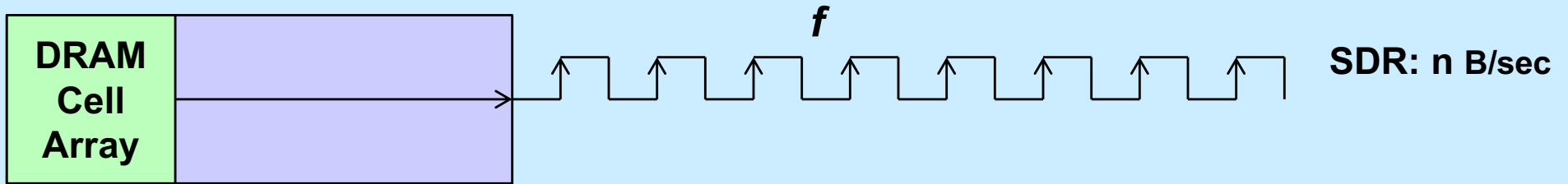
Memory Modules



Enhanced DRAMs

- **Basic DRAM cell has not changed since its invention in 1966**
 - commercialized by Intel in 1970
- **DRAMs with better interface logic and faster I/O:**
 - **synchronous DRAM (SDRAM)**
 - » uses a conventional clock signal instead of asynchronous control
 - » allows reuse of the row addresses (e.g., RAS, CAS, CAS, CAS)
 - **double data-rate synchronous DRAM (DDR SDRAM)**
 - » **DDR1**
 - twice as fast
 - » **DDR2**
 - four times as fast
 - » **DDR3**
 - eight times as fast

Enhanced DRAMs



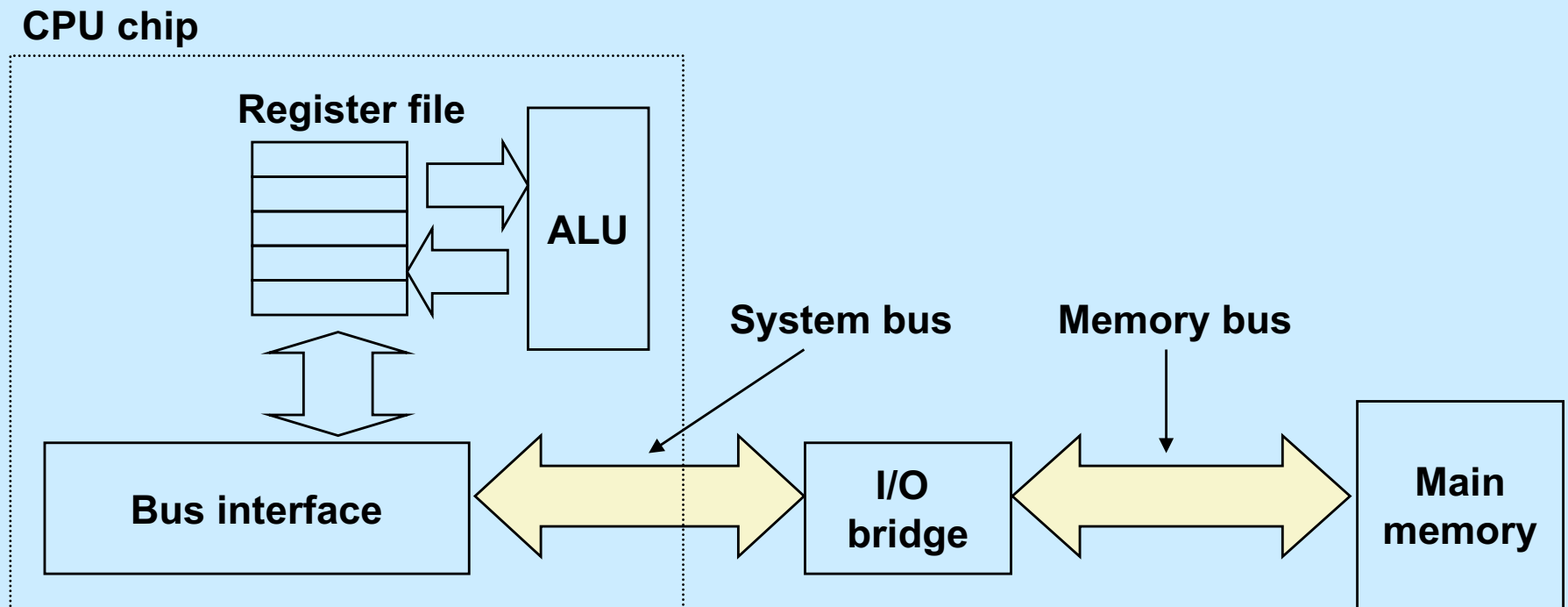
Quiz 1

A program is loading randomly selected bytes from memory. These bytes will be delivered to the processor on a DDR3 system n times faster than on an SDR system, where n is:

- a) 1**
- b) 2**
- c) 4**
- d) 8**

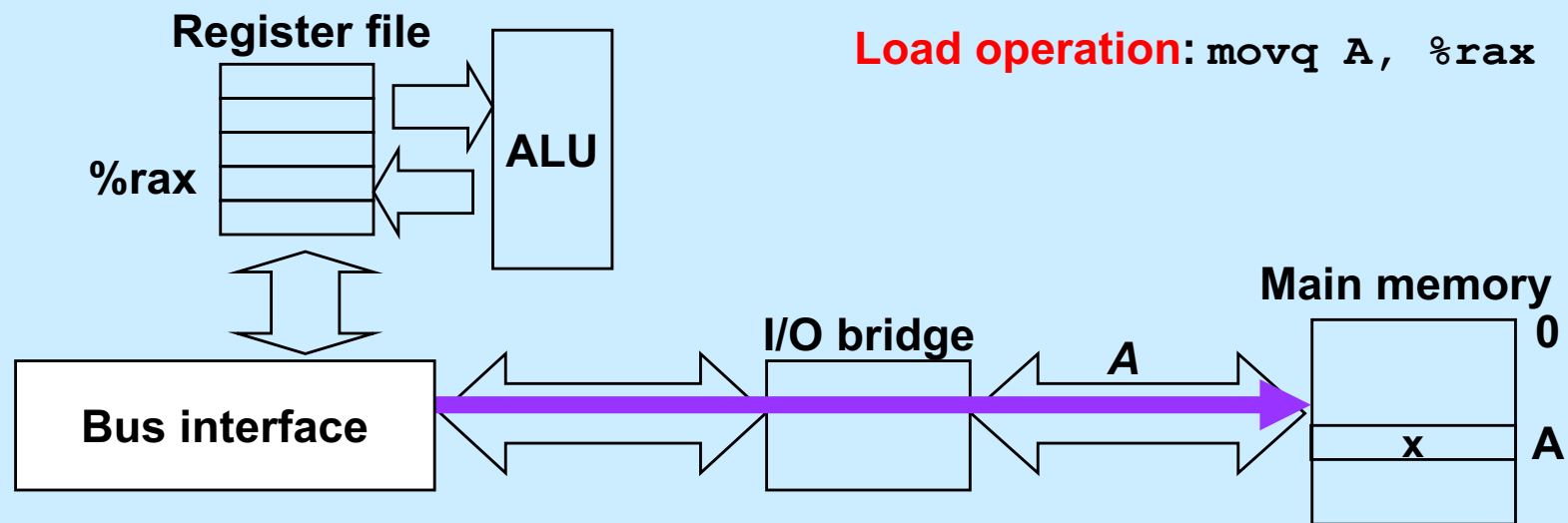
Traditional Bus Structure Connecting CPU and Memory

- A **bus** is a collection of parallel wires that carry address, data, and control signals
- Buses are typically shared by multiple devices



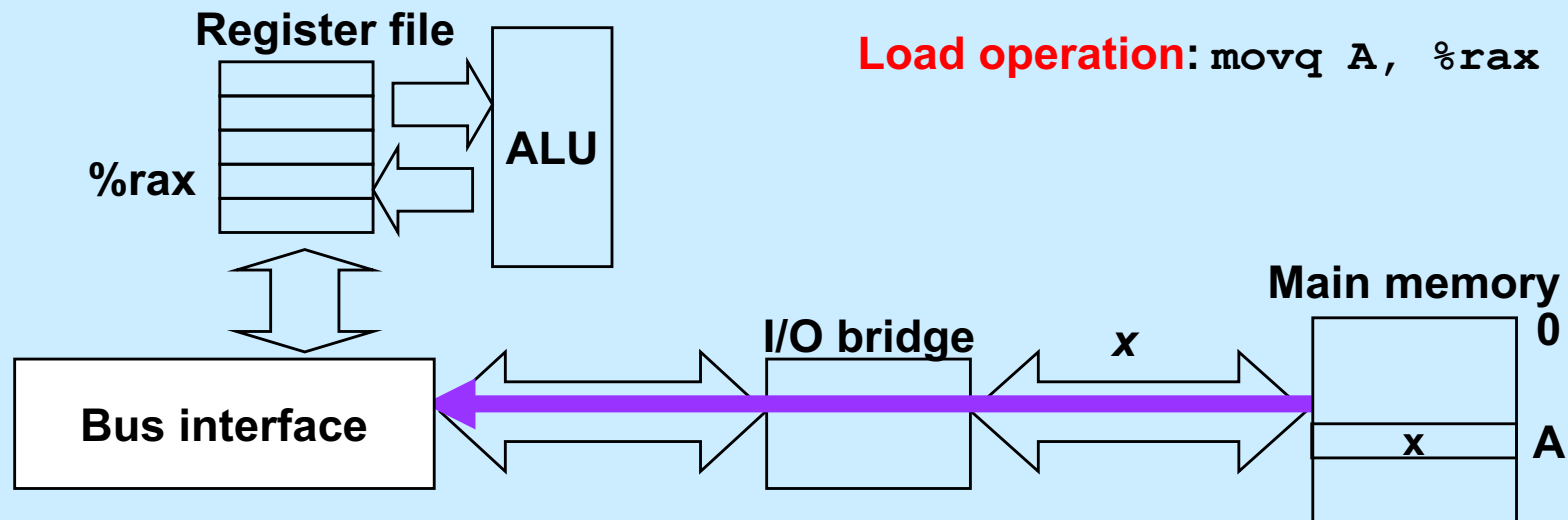
Memory Read Transaction (1)

- CPU places address *A* on the memory bus



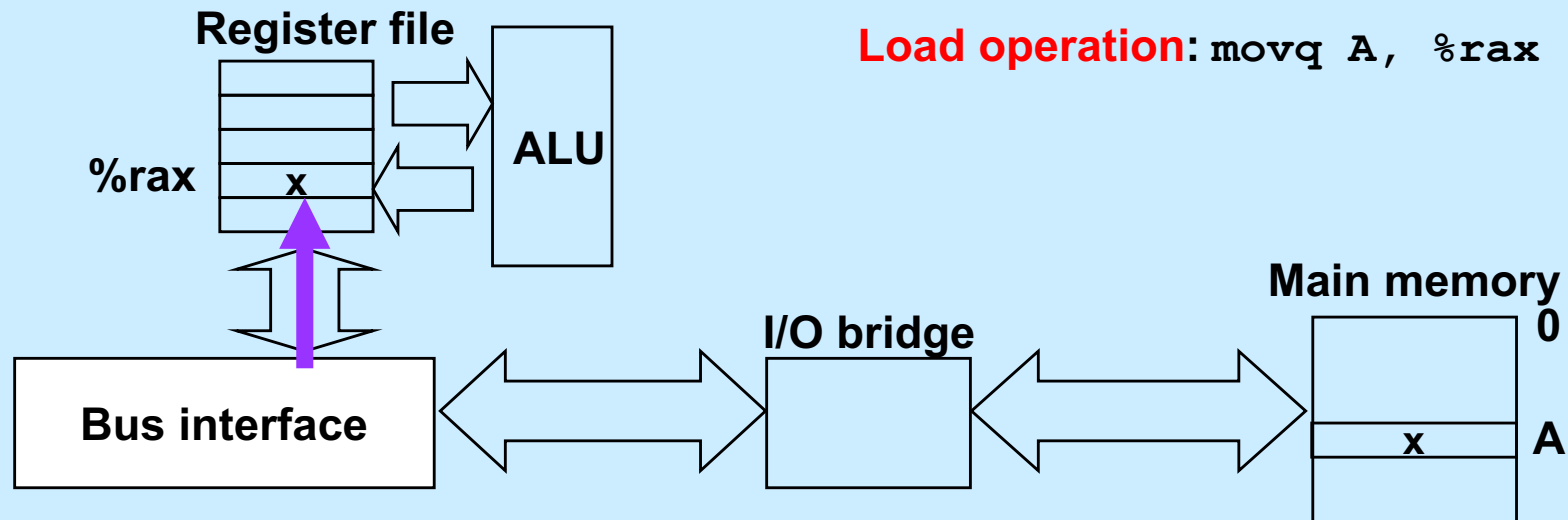
Memory Read Transaction (2)

- Main memory reads *A* from the memory bus, retrieves word *x*, and places it on the bus



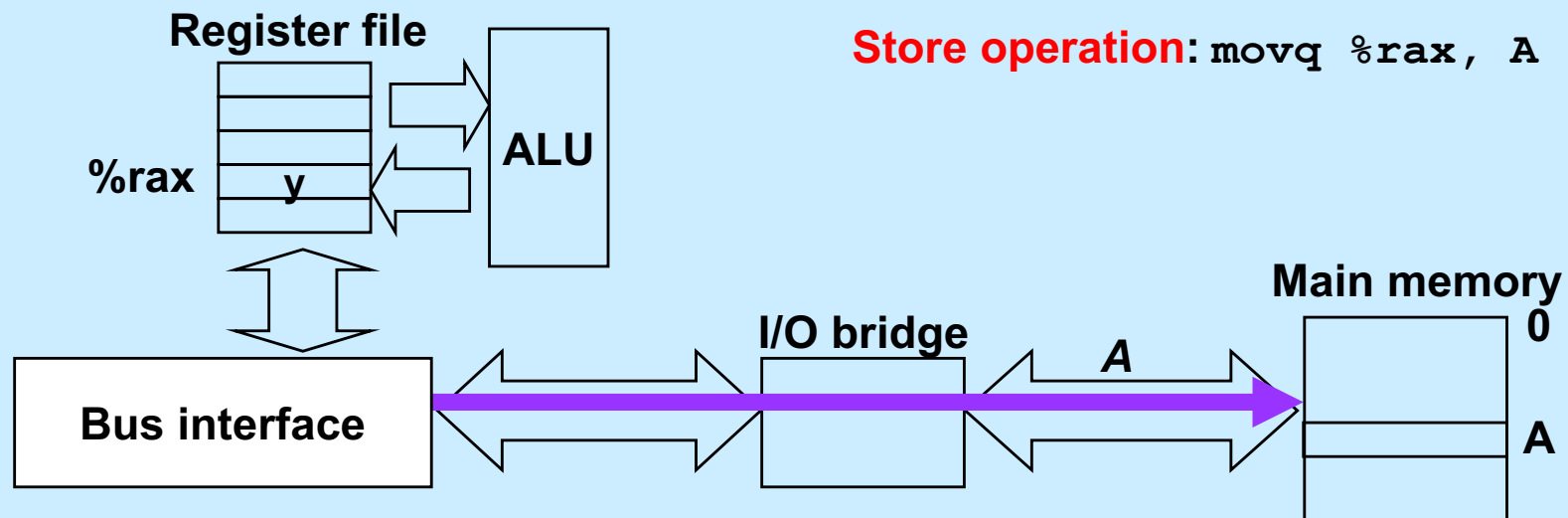
Memory Read Transaction (3)

- CPU reads word x from the bus and copies it into register `%rax`



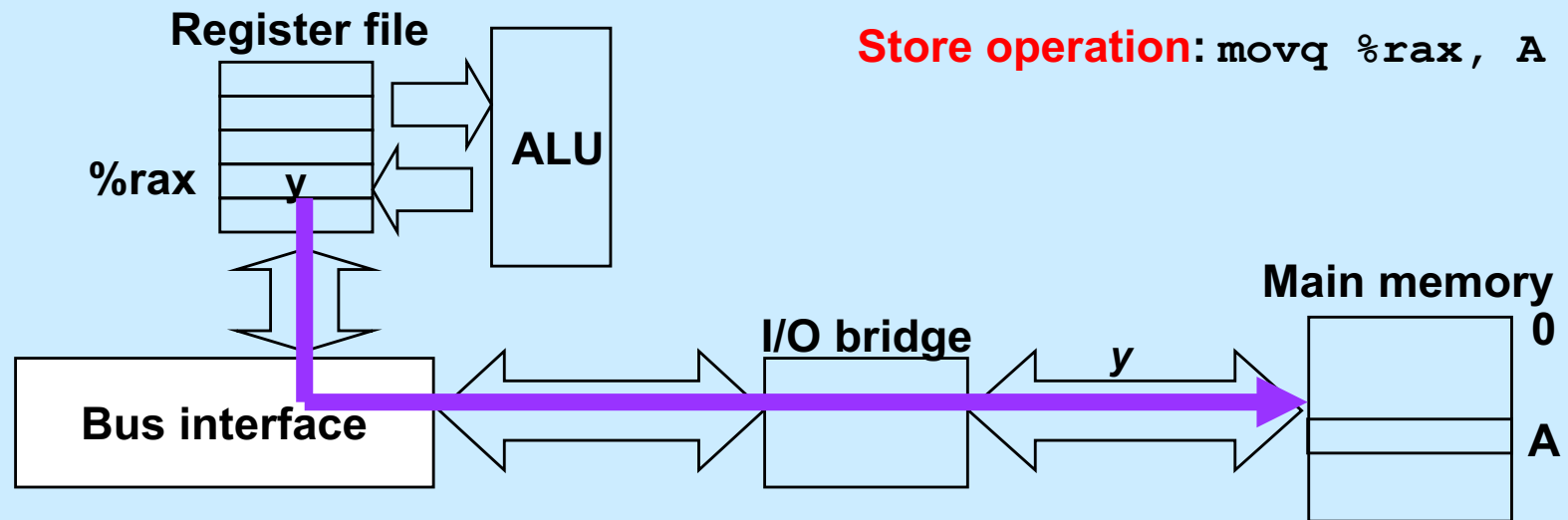
Memory Write Transaction (1)

- CPU places address *A* on bus. Main memory reads it and waits for the corresponding data word to arrive



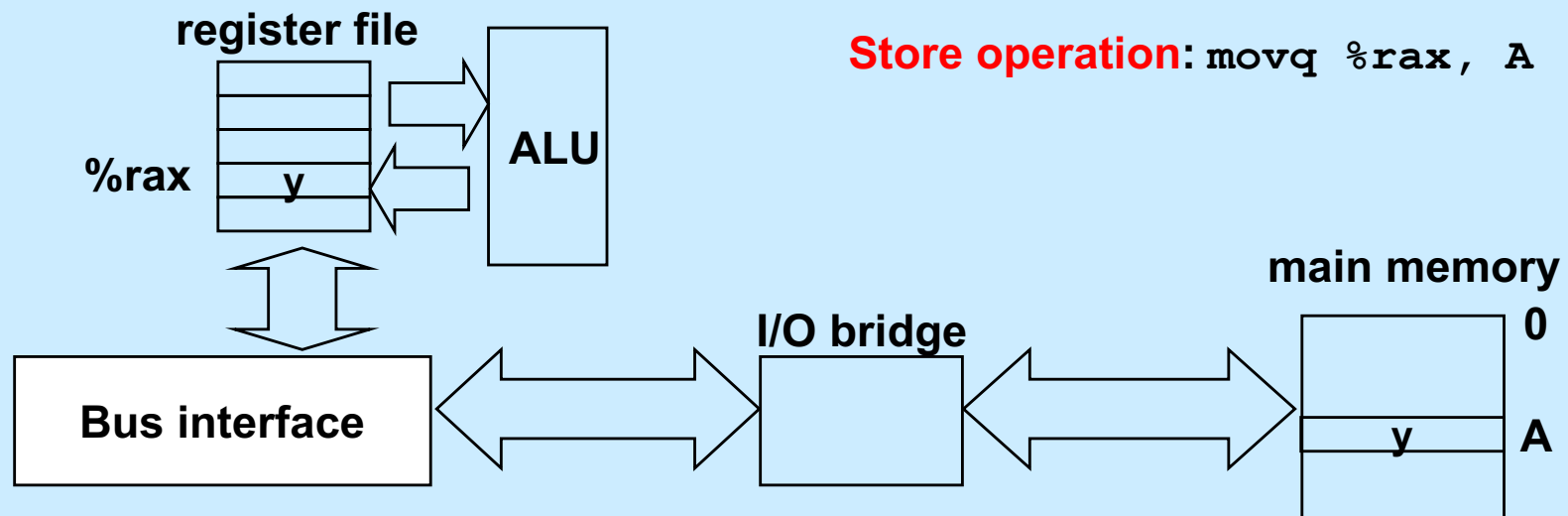
Memory Write Transaction (2)

- CPU places data word y on the bus



Memory Write Transaction (3)

- Main memory reads data word y from the bus and stores it at address A



A Mismatch

- **A processor clock cycle is ~0.3 nsecs**
 - SunLab machines (Intel Core i5-4690) run at 3.5 GHz
- **Basic operations take 1 – 10 clock cycles**
 - .3 – 3 nsecs
- **Accessing memory takes 70-100 nsecs**
- **How is this made to work?**

Caching to the Rescue

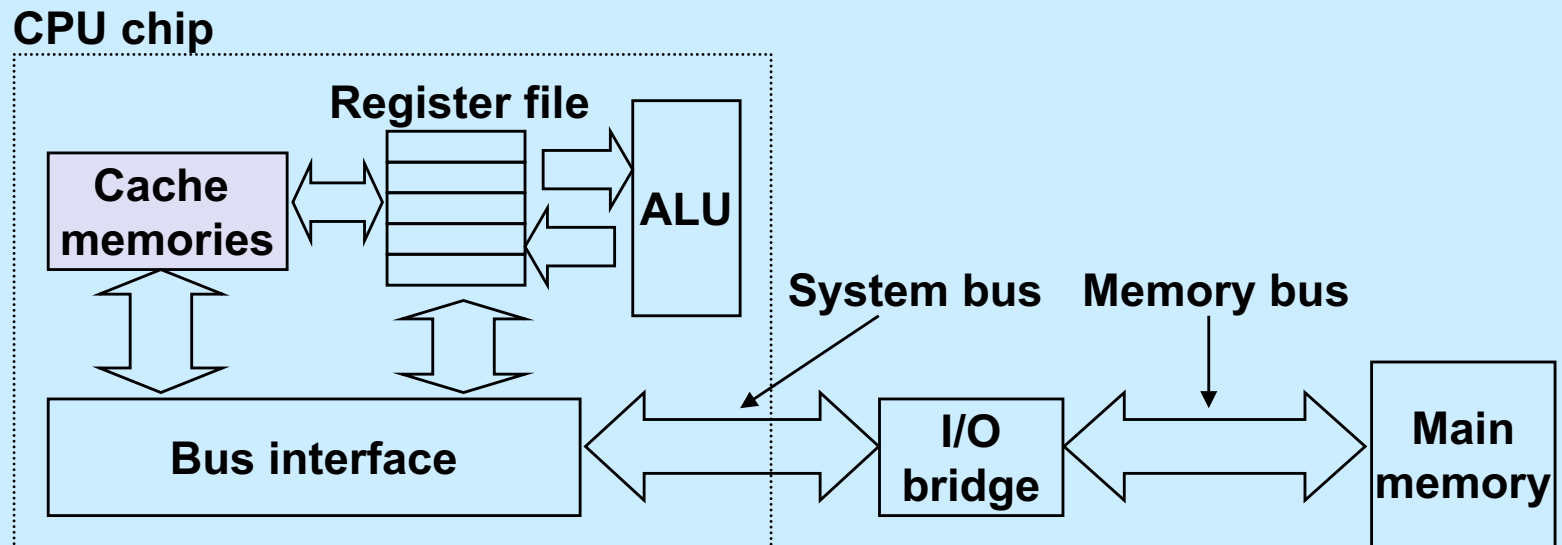
CPU

Cache

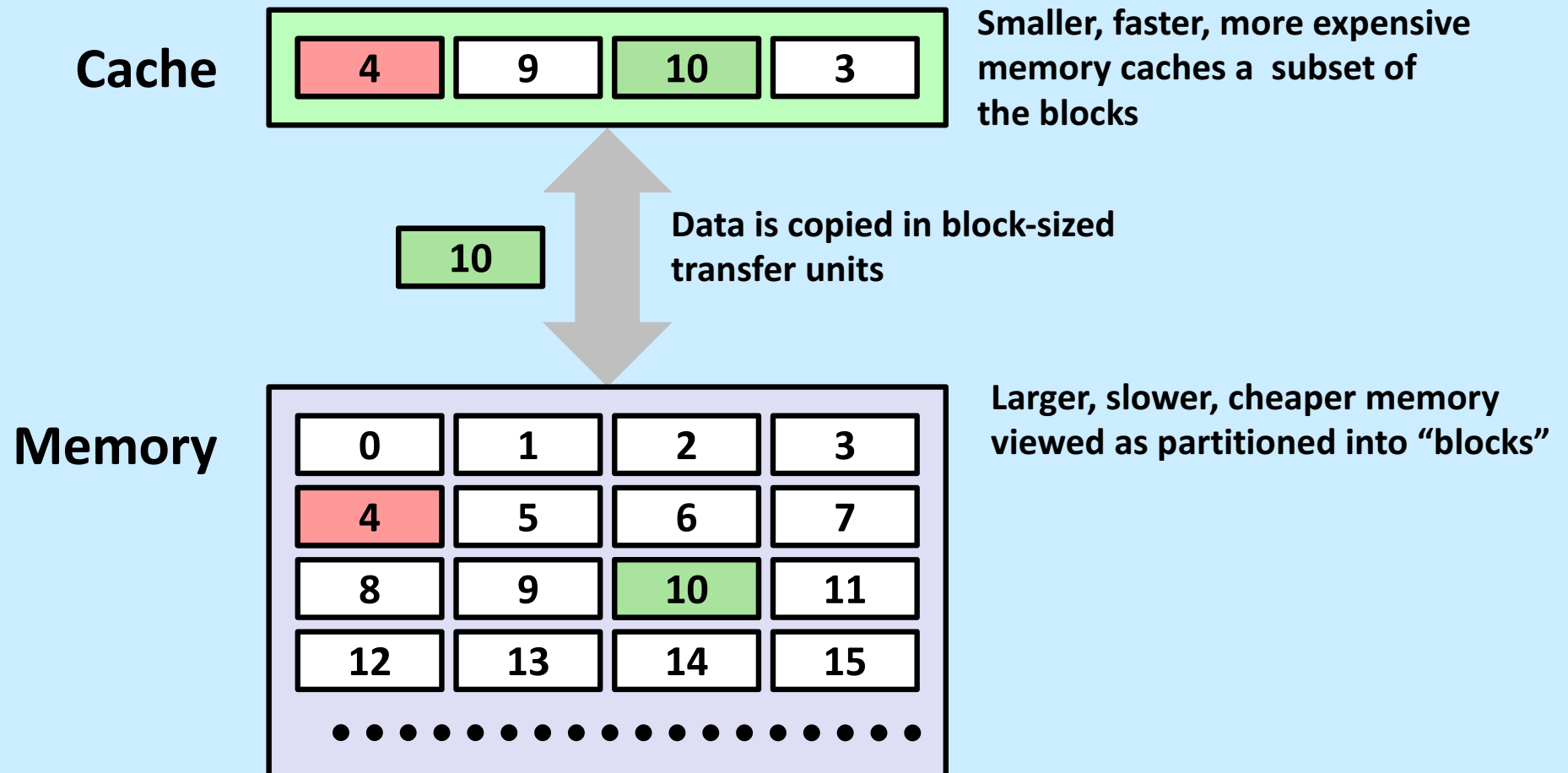


Cache Memories

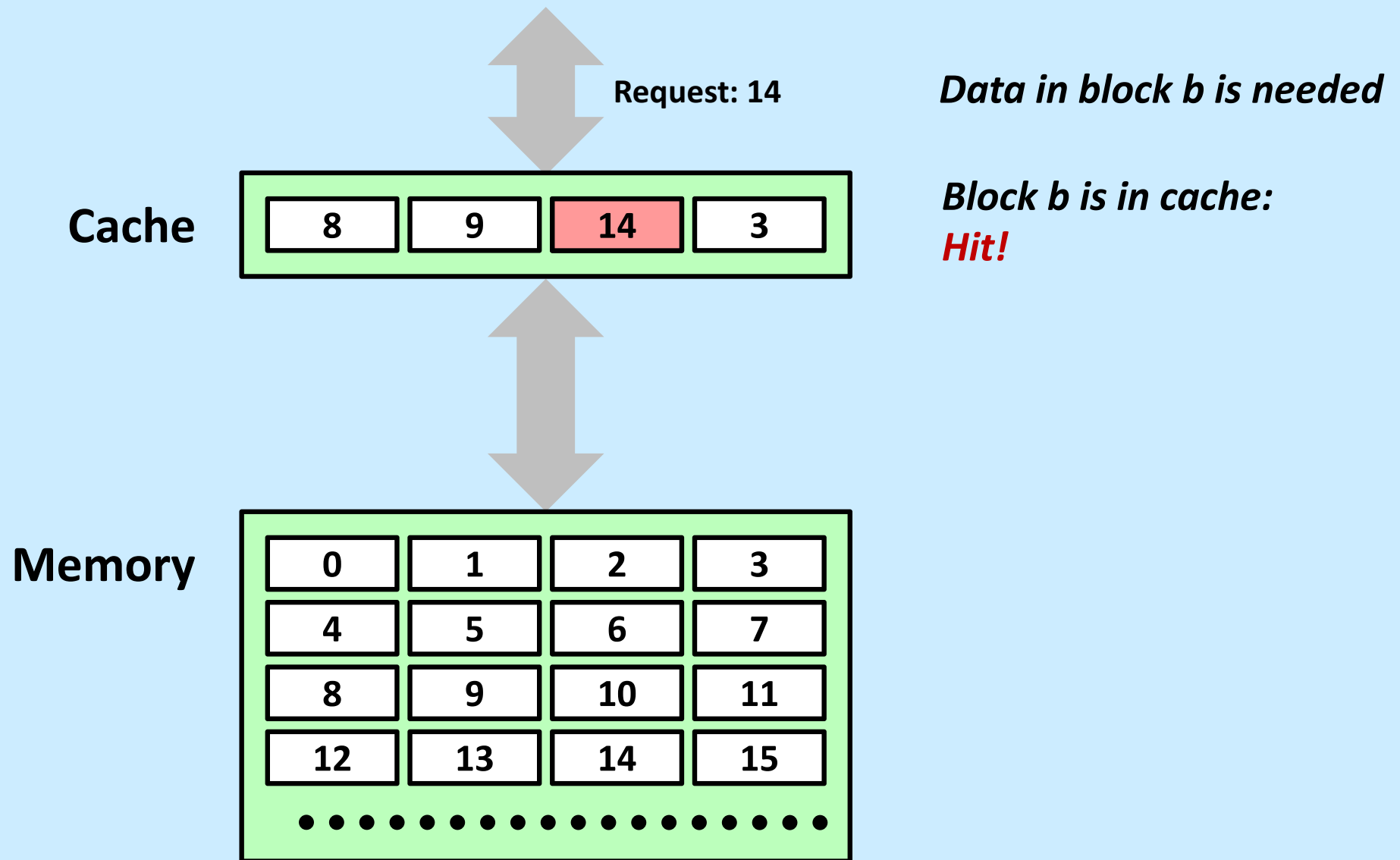
- **Cache memories** are small, fast SRAM-based memories managed automatically in hardware
 - hold frequently accessed blocks of main memory
- CPU looks first for data in caches (e.g., L1, L2, and L3), then in main memory
- Typical system structure:



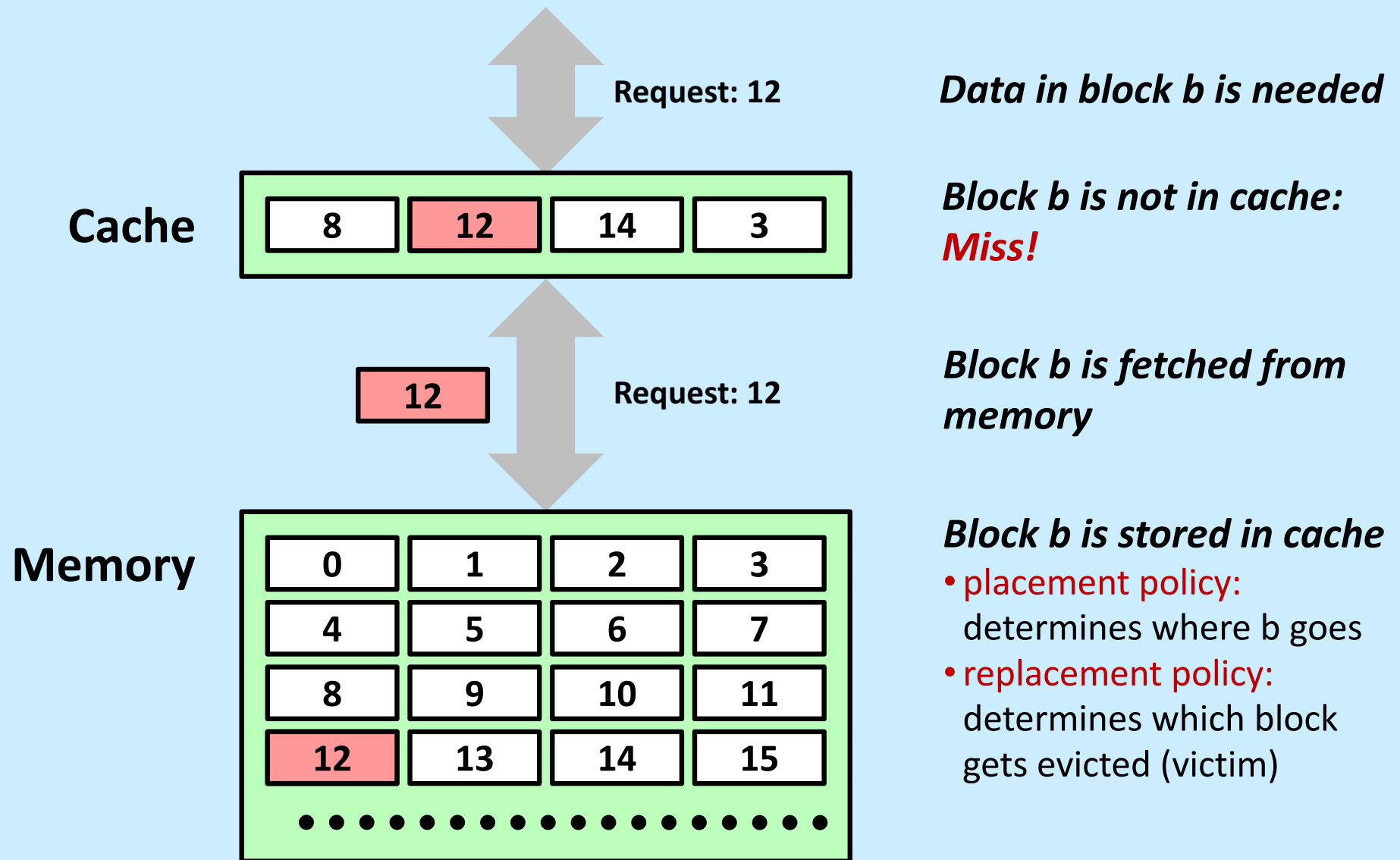
General Cache Concepts



General Cache Concepts: Hit



General Cache Concepts: Miss

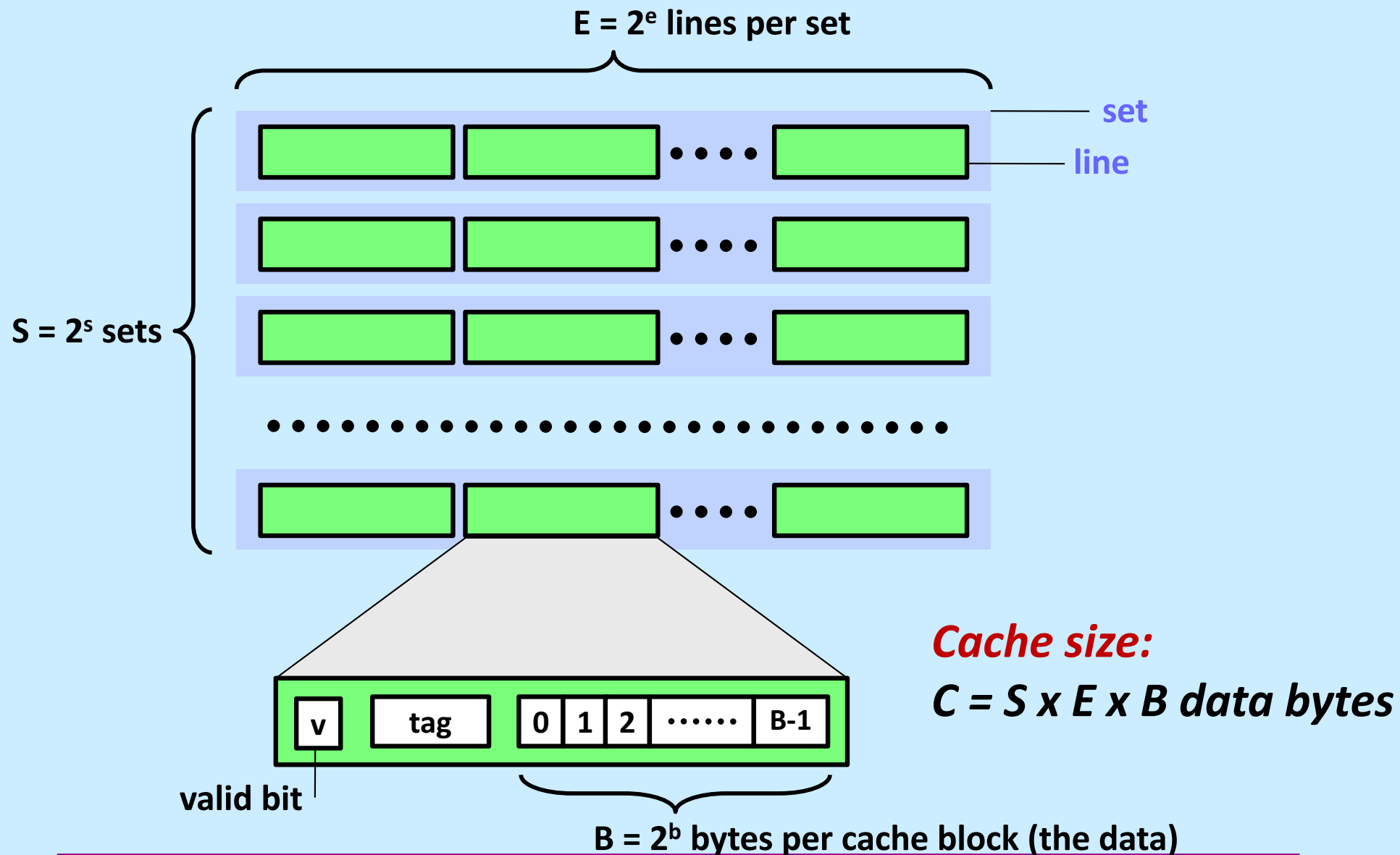


General Caching Concepts:

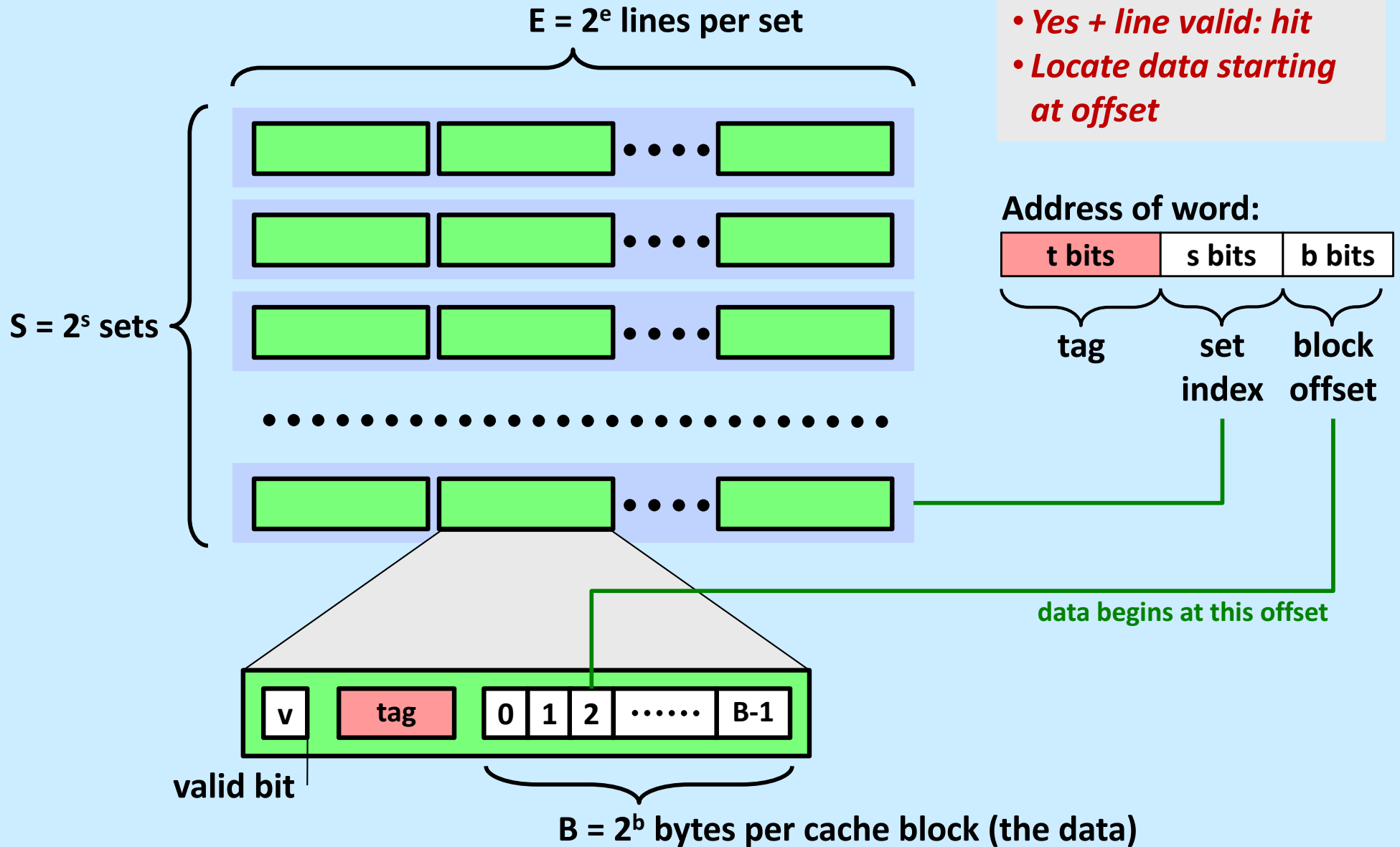
Types of Cache Misses

- **Cold (compulsory) miss**
 - cold misses occur because the cache is empty
- **Conflict miss**
 - most caches limit blocks to a small subset (sometimes a singleton) of the block positions in RAM
 - » e.g., block i in RAM must be placed in block $(i \bmod 4)$ in the cache
 - conflict misses occur when the cache is large enough, but multiple data objects all map to the same cache block
 - » e.g., referencing blocks 0, 8, 0, 8, 0, 8, ... would miss every time
- **Capacity miss**
 - occurs when the set of active cache blocks (**working set**) is larger than the cache

General Cache Organization (S, E, B)

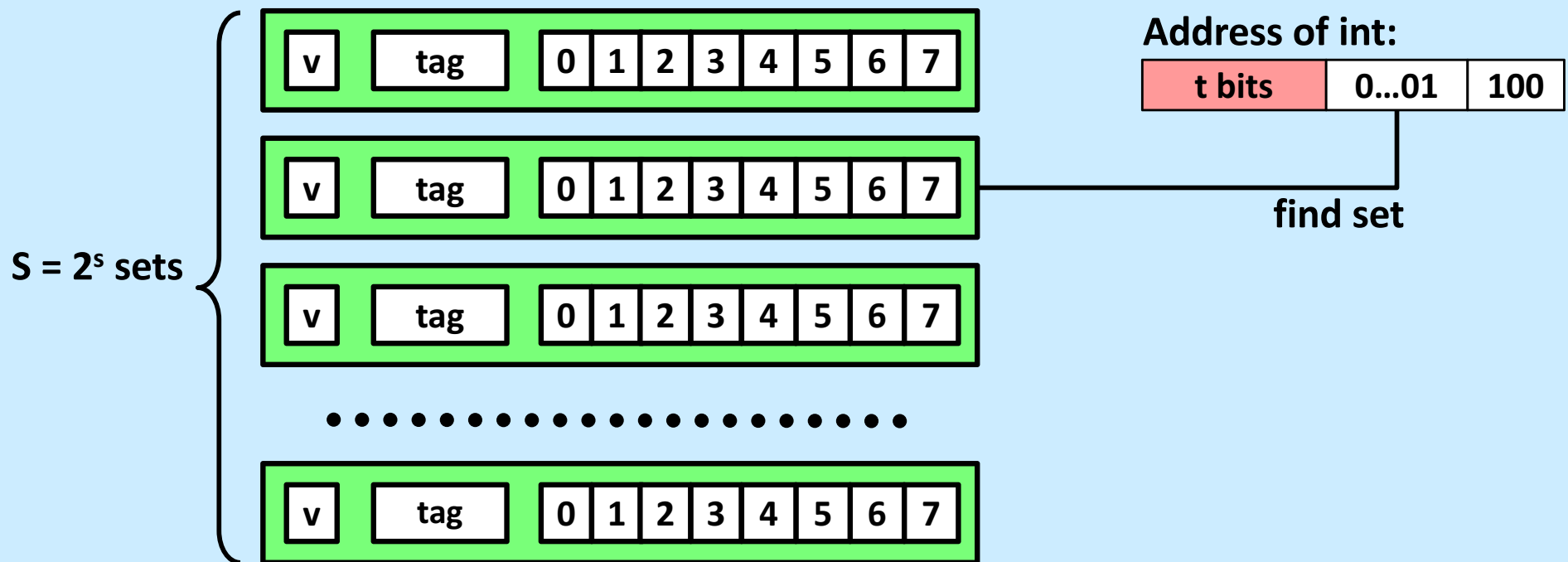


Cache Read



Example: Direct Mapped Cache (E = 1)

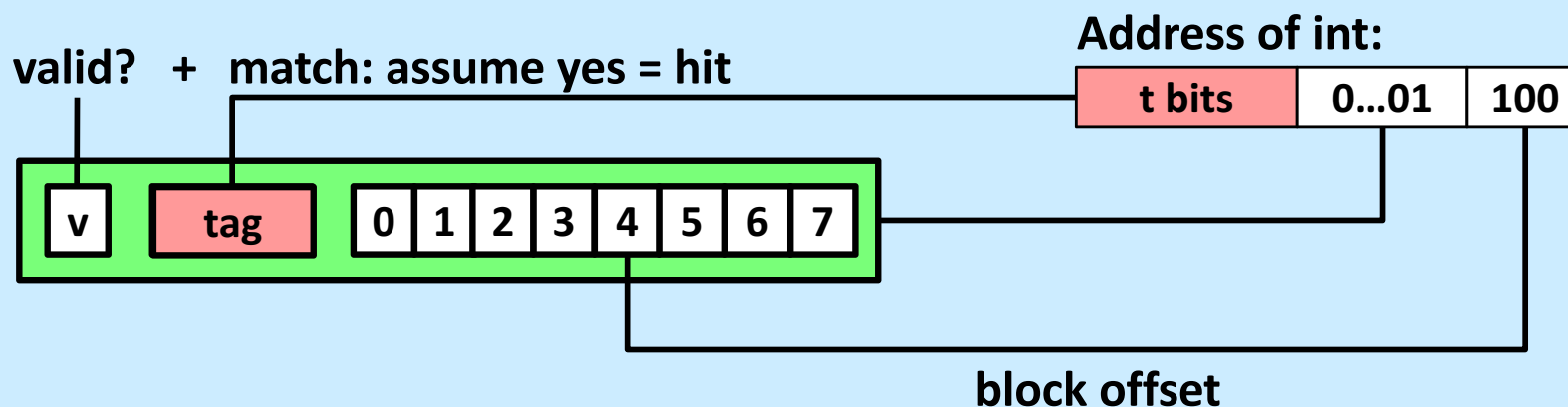
Direct mapped: one line per set
Assume: cache block size 8 bytes



Example: Direct Mapped Cache (E = 1)

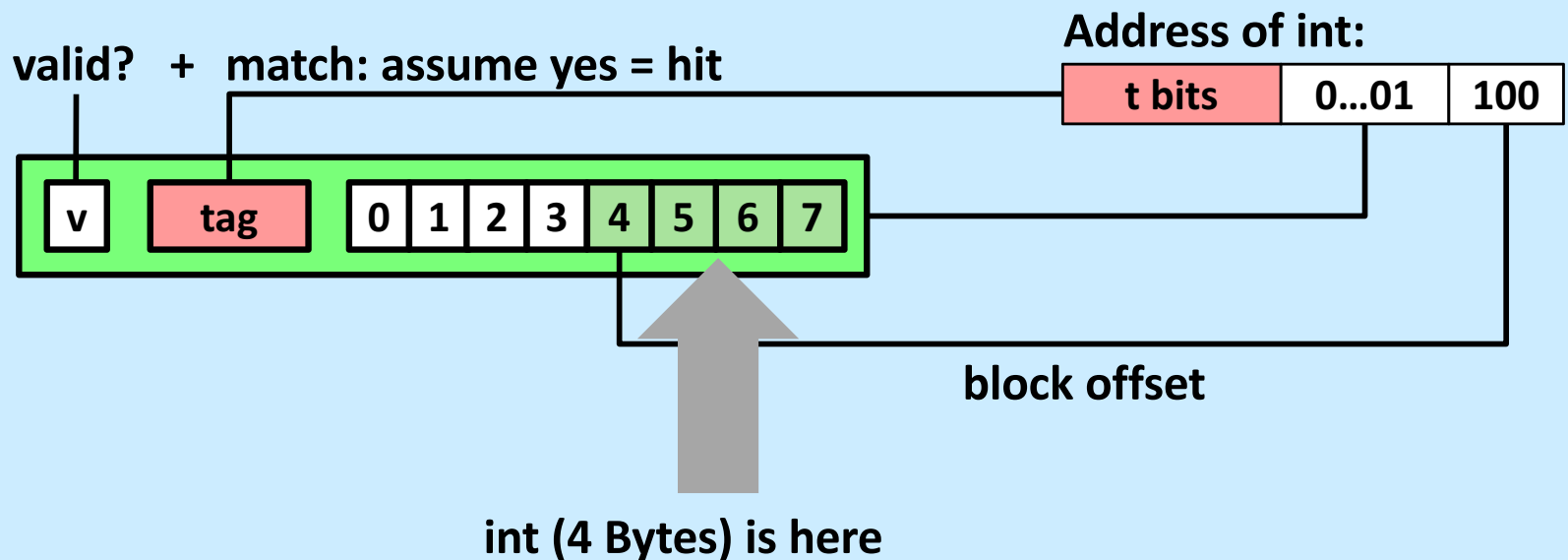
Direct mapped: one line per set

Assume: cache block size 8 bytes



Example: Direct Mapped Cache (E = 1)

Direct mapped: one line per set
Assume: cache block size 8 bytes



No match: old line is evicted and replaced

Direct-Mapped Cache Simulation

t=1	s=2	b=1
x	xx	x

M=16 byte addresses, B=2 bytes/block,
S=4 sets, E=1 Blocks/set

Address trace (reads, one byte per read):

0	[<u>0000</u> ₂],	miss
1	[<u>0001</u> ₂],	hit
7	[<u>0111</u> ₂],	miss
8	[<u>1000</u> ₂],	miss
0	[<u>0000</u> ₂]	miss

	v	Tag	Block
Set 0	1	0	M[0-1]
Set 1			
Set 2			
Set 3	1	0	M[6-7]

A Higher-Level Example

```
int sum_array_rows(double a[16][16])
{
    int i, j;
    double sum = 0;

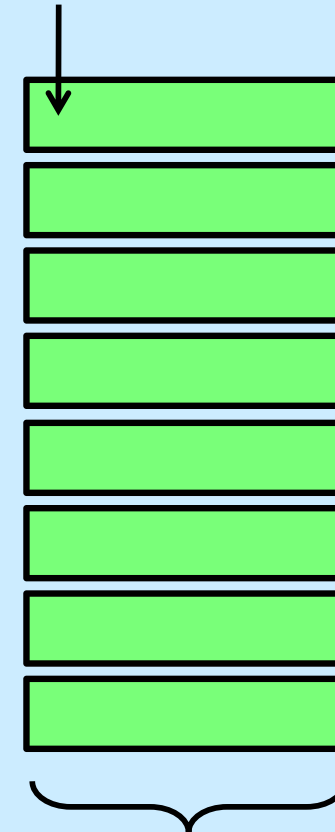
    for (i = 0; i < 16; i++)
        for (j = 0; j < 16; j++)
            sum += a[i][j];
    return sum;
}
```

```
int sum_array_cols(double a[16][16])
{
    int i, j;
    double sum = 0;

    for (j = 0; j < 16; j++)
        for (i = 0; i < 16; i++)
            sum += a[i][j];
    return sum;
}
```

Ignore the variables sum, i, j

assume: cold (empty) cache,
a[0][0] goes here



32 B = 4 doubles

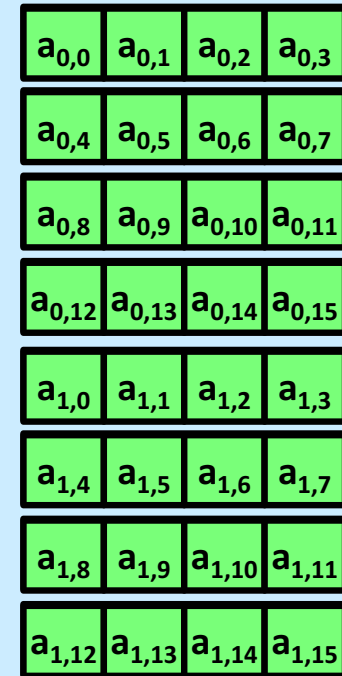
A Higher-Level Example

```
int sum_array_rows(double a[16][16])
{
    int i, j;
    double sum = 0;

    for (i = 0; i < 16; i++)
        for (j = 0; j < 16; j++)
            sum += a[i][j];
    return sum;
}
```

```
int sum_array_cols(double a[16][16])
{
    int i, j;
    double sum = 0;

    for (j = 0; j < 16; j++)
        for (i = 0; i < 16; i++)
            sum += a[i][j];
    return sum;
}
```



32 B = 4 doubles

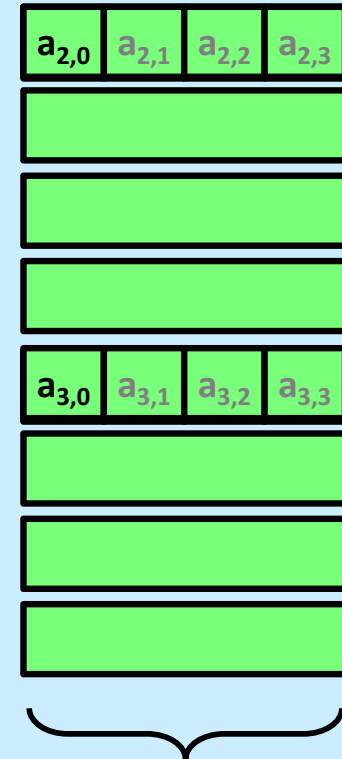
A Higher-Level Example

```
int sum_array_rows(double a[16][16])
{
    int i, j;
    double sum = 0;

    for (i = 0; i < 16; i++)
        for (j = 0; j < 16; j++)
            sum += a[i][j];
    return sum;
}
```

```
int sum_array_cols(double a[16][16])
{
    int i, j;
    double sum = 0;

    for (j = 0; j < 16; j++)
        for (i = 0; i < 16; i++)
            sum += a[i][j];
    return sum;
}
```



32 B = 4 doubles

Conflict Misses: Aligned

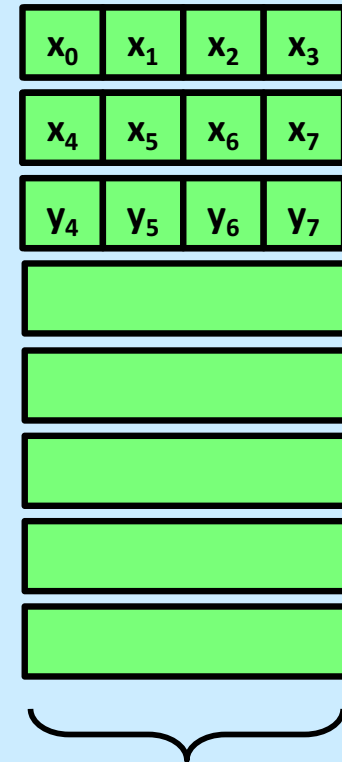
```
double dotprod(double x[8], double y[8]) {  
    double sum = 0.0;  
    int i;  
  
    for (i=0; i<8; i++)  
        sum += x[i] * y[i];  
  
    return sum;  
}
```



32 B = 4 doubles

Different Alignments

```
double dotprod(double x[8], double y[8]) {  
    double sum = 0.0;  
    int i;  
  
    for (i=0; i<8; i++)  
        sum += x[i] * y[i];  
  
    return sum;  
}
```

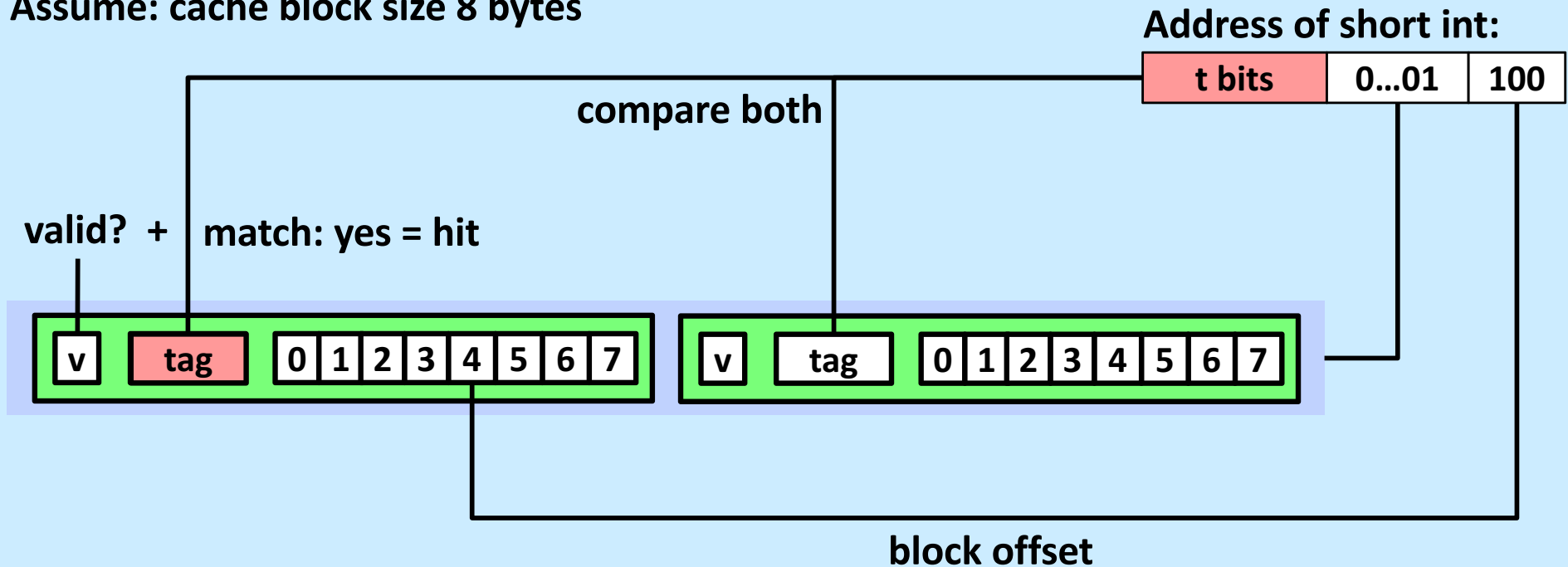


32 B = 4 doubles

E-way Set-Associative Cache (Here: E = 2)

E = 2: two lines per set

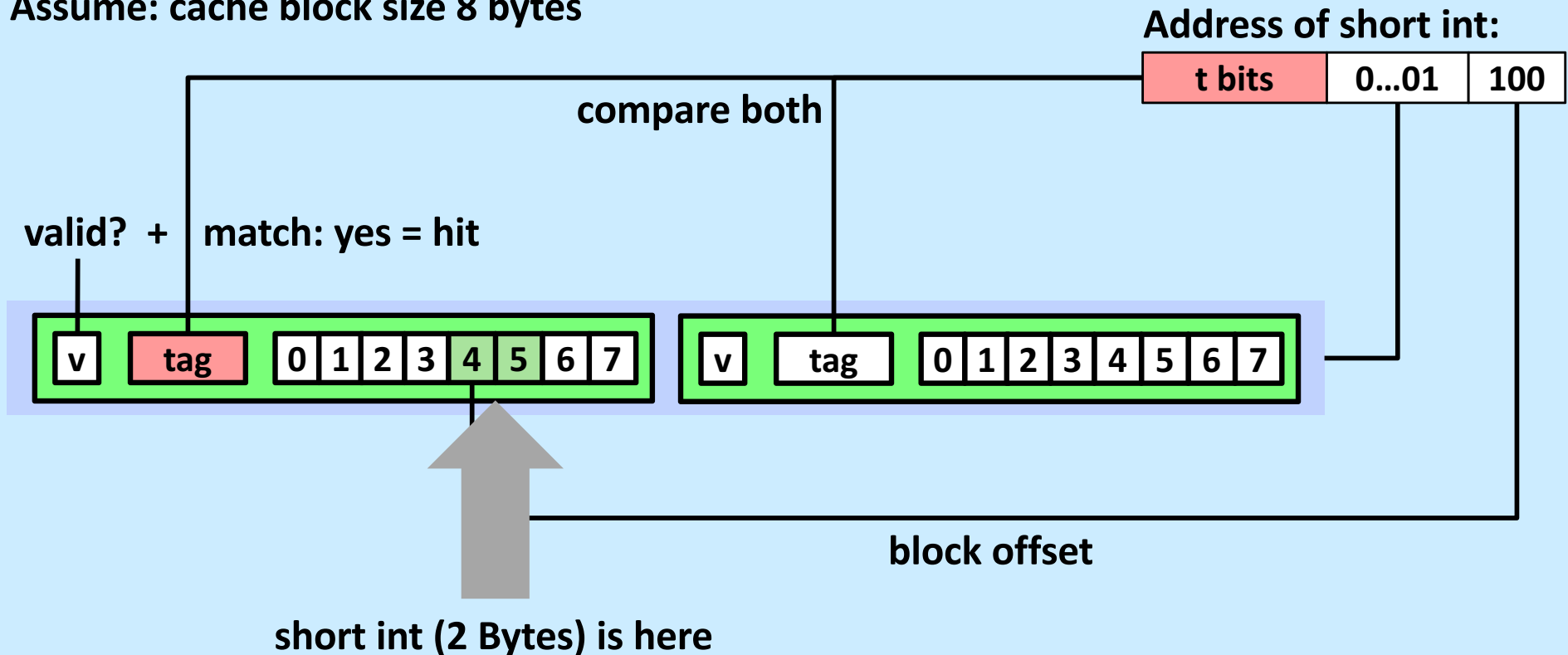
Assume: cache block size 8 bytes



E-way Set-Associative Cache (Here: E = 2)

E = 2: two lines per set

Assume: cache block size 8 bytes



No match:

- One line in set is selected for eviction and replacement
- Replacement policies: random, least recently used (LRU), ...

Quiz 1

Address of int:

100	01	100
-----	----	-----

0	v	tag=0	0	0	0	0	1	1	1	1
1	v	tag=0	4	4	4	4	5	5	5	5
2	v	tag=2	8	8	8	8	9	9	9	9
3	v	tag=4	c	c	c	c	d	d	d	d
	v	tag=2	2	2	2	2	3	3	3	3
	v	tag=4	6	6	6	6	7	7	7	7
	v	tag=3	a	a	a	a	b	b	b	b
	v	tag=a	e	e	e	e	f	f	f	f

Given the address above and the cache contents as shown, what is the value of the *int* at the given address?

- a) 1111
- b) 3333
- c) 4444
- d) 7777

2-Way Set-Associative Cache Simulation

t=2	s=1	b=1
xx	x	x

M=16 byte addresses, B=2 bytes/block,
S=2 sets, E=2 blocks/set

Address trace (reads, one byte per read):

0	[0000 ₂],	miss
1	[0001 ₂],	hit
7	[0111 ₂],	miss
8	[1000 ₂],	miss
0	[0000 ₂]	hit

	v	Tag	Block
Set 0	1	00	M[0-1]
	1	10	M[8-9]
Set 1	1	01	M[6-7]
	0		

A Higher-Level Example

```
int sum_array_rows(double a[16][16])
{
    int i, j;
    double sum = 0;

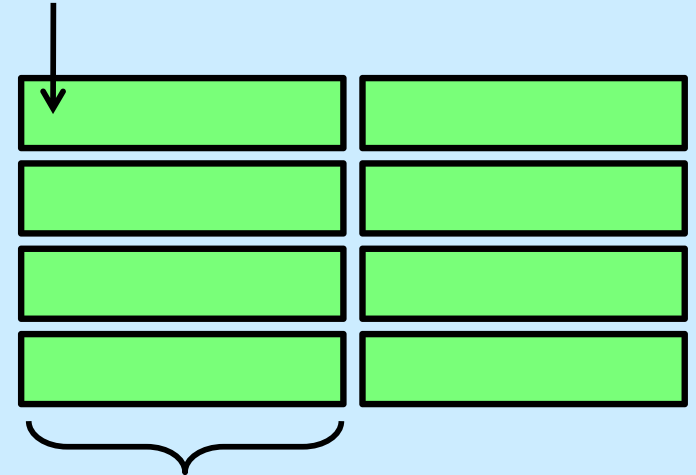
    for (i = 0; i < 16; i++)
        for (j = 0; j < 16; j++)
            sum += a[i][j];
    return sum;
}
```

```
int sum_array_rows(double a[16][16])
{
    int i, j;
    double sum = 0;

    for (j = 0; j < 16; j++)
        for (i = 0; i < 16; i++)
            sum += a[i][j];
    return sum;
}
```

Ignore the variables sum, i, j

assume: cold (empty) cache,
a[0][0] goes here



A Higher-Level Example

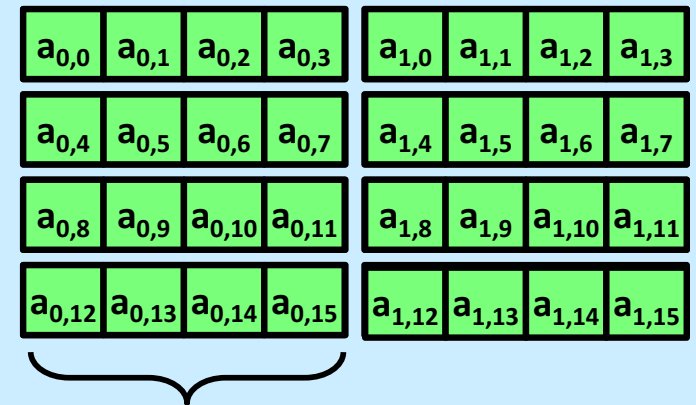
Ignore the variables `sum`, `i`, `j`

```
int sum_array_rows(double a[16][16])
{
    int i, j;
    double sum = 0;

    for (i = 0; i < 16; i++)
        for (j = 0; j < 16; j++)
            sum += a[i][j];
    return sum;
}
```

```
int sum_array_cols(double a[16][16])
{
    int i, j;
    double sum = 0;

    for (j = 0; j < 16; j++)
        for (i = 0; i < 16; i++)
            sum += a[i][j];
    return sum;
}
```



32 B = 4 doubles

A Higher-Level Example

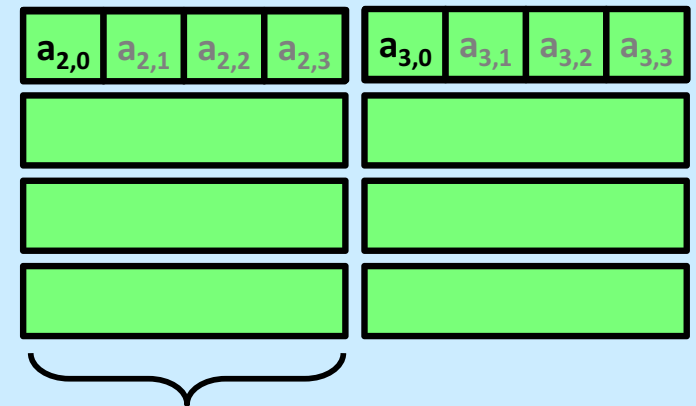
Ignore the variables sum , i , j

```
int sum_array_rows(double a[16][16])
{
    int i, j;
    double sum = 0;

    for (i = 0; i < 16; i++)
        for (j = 0; j < 16; j++)
            sum += a[i][j];
    return sum;
}
```

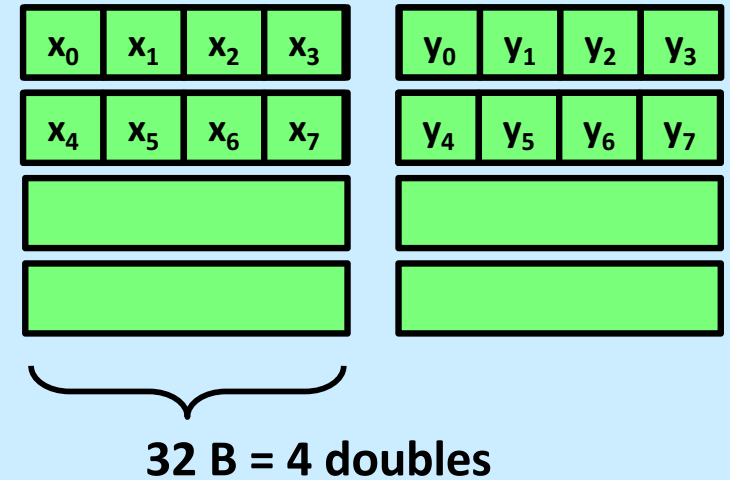
```
int sum_array_cols(double a[16][16])
{
    int i, j;
    double sum = 0;

    for (j = 0; j < 16; j++)
        for (i = 0; i < 16; i++)
            sum += a[i][j];
    return sum;
}
```



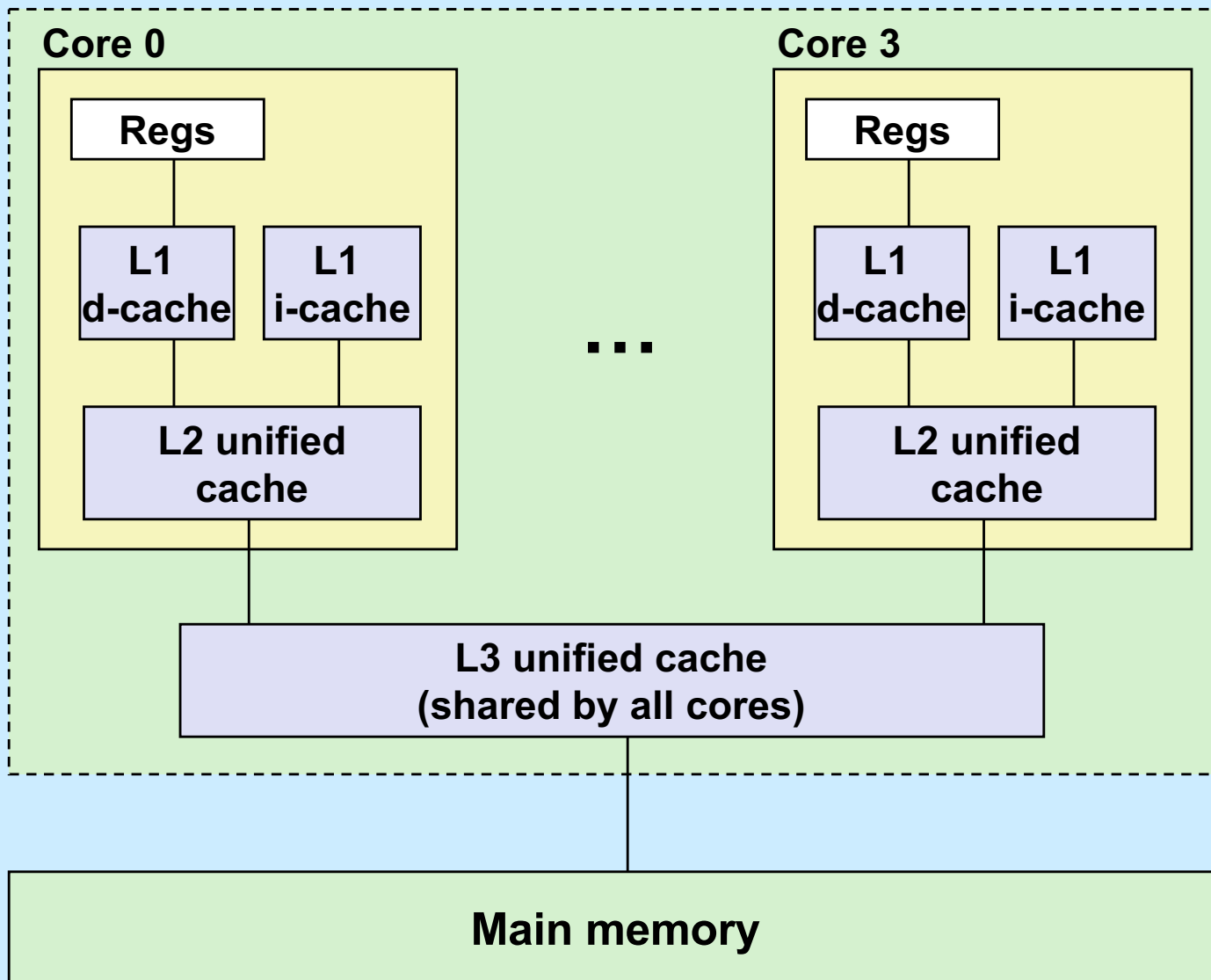
Conflict Misses

```
double dotprod(double x[8], double y[8]) {  
    double sum = 0.0;  
    int i;  
  
    for (i=0; i<8; i++)  
        sum += x[i] * y[i];  
  
    return sum;  
}
```



Intel Core i5 and i7 Cache Hierarchy

Processor package



L1 i-cache and d-cache:

32 KB, 8-way,
Access: 4 cycles

L2 unified cache:

256 KB, 8-way,
Access: 11 cycles

L3 unified cache:

8 MB, 16-way,
Access: 30-40 cycles

Block size: 64 bytes for
all caches

What About Writes?

- Multiple copies of data exist:
 - L1, L2, main memory, disk
 - What to do on a write-hit?
 - **write-through** (write immediately to memory)
 - **write-back** (defer write to memory until replacement of line)
 - » need a dirty bit (line different from memory or not)
 - What to do on a write-miss?
 - **write-allocate** (load into cache, update line in cache)
 - » good if more writes to the location follow
 - **no-write-allocate** (writes immediately to memory)
 - Typical
 - write-through + no-write-allocate
 - write-back + write-allocate
-