UNM SCHOOL of ENGINEERING
Department of Computer Science

Lecture #18

Ch. 6 The Memory Hierarchy (2)

Prof. Soraya Abad-Mota, PhD

From the index cards

- The only storage inside the CPU is for the registers, there is no RAM inside the CPU
- Primary storage is main memory (and caches which are faster than main memory)
- Make sure you follow the slides about the different types of volatile memory.
  - The fastest and most expensive in dollars ($) is SRAM (used in cache memories)
  - the next in speed is DRAM and it is 1000 times less expensive

From the index cards

- Secondary storage (farther away from the CPU)
- Rotating (magnetic) disks and SSD are two technologies widely used for secondary storage.
- How can a file be scattered on the disk?
  - A file is not fixed, it can get more data added to it. The system does not reserve all the space it will ever need (except for External hash maybe, and even that one may grow). Therefore, when you are adding more data, there might not be enough space for it next to the previous data and you need to find spece elsewhere and “link” it to the current space.

Sequence of “formulas” for disks

- Disk capacity (raw capacity without format)
  - Input: platters, tracks/cylinders, sectors per track, bytes in each sector
  - Each platter has 2 surfaces

  Cap = 2 * # platters x #tracks x #sectors x #bytes/sector
- For simplicity, one sector is the same as one block of disk, i.e. the same in size.
- Files that we place on the disk are comprised of “blocks” of disk.

Sequence of “formulas” for disks (2)

- Access to a file
  - Input: size of the file
  - Placement on disk:
    - Contiguous (one block next to the other)
    - Scattered (some blocks are contiguous others are separate)
- For rotating disks:
  - Seek time: 3-9 ms (usually given like that)
  - Rotational delay: given RPM can calculate time for a full rotation and time to rotate over one sector. Can you build that formula?
  - Transfer time

Locality (more examples)

- Recall that we are after accessing data elements that are close to each other.
- We saw one example of accessing a one-dimensional array.
- Two more examples here example.

Qualitative Estimates of Locality

- Claim: Being able to look at code and get a qualitative sense of its locality is a key skill for a professional programmer.
- Question: Does this function have good locality with respect to array `a`? (Figure 6.18)

```
int sum_array_rows(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

- Question: Does this function have good locality with respect to array `a`? (Figure 6.19)

```
int sum_array_cols(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;
}
```

Locality Example

- Question: Can you permute the loops so that the function scans the 3-d array `a` with a stride-1 reference pattern (and thus has good spatial locality)?

```
int sum_array_3d(int a[M][N][N])
{
    int i, j, k, sum = 0;

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

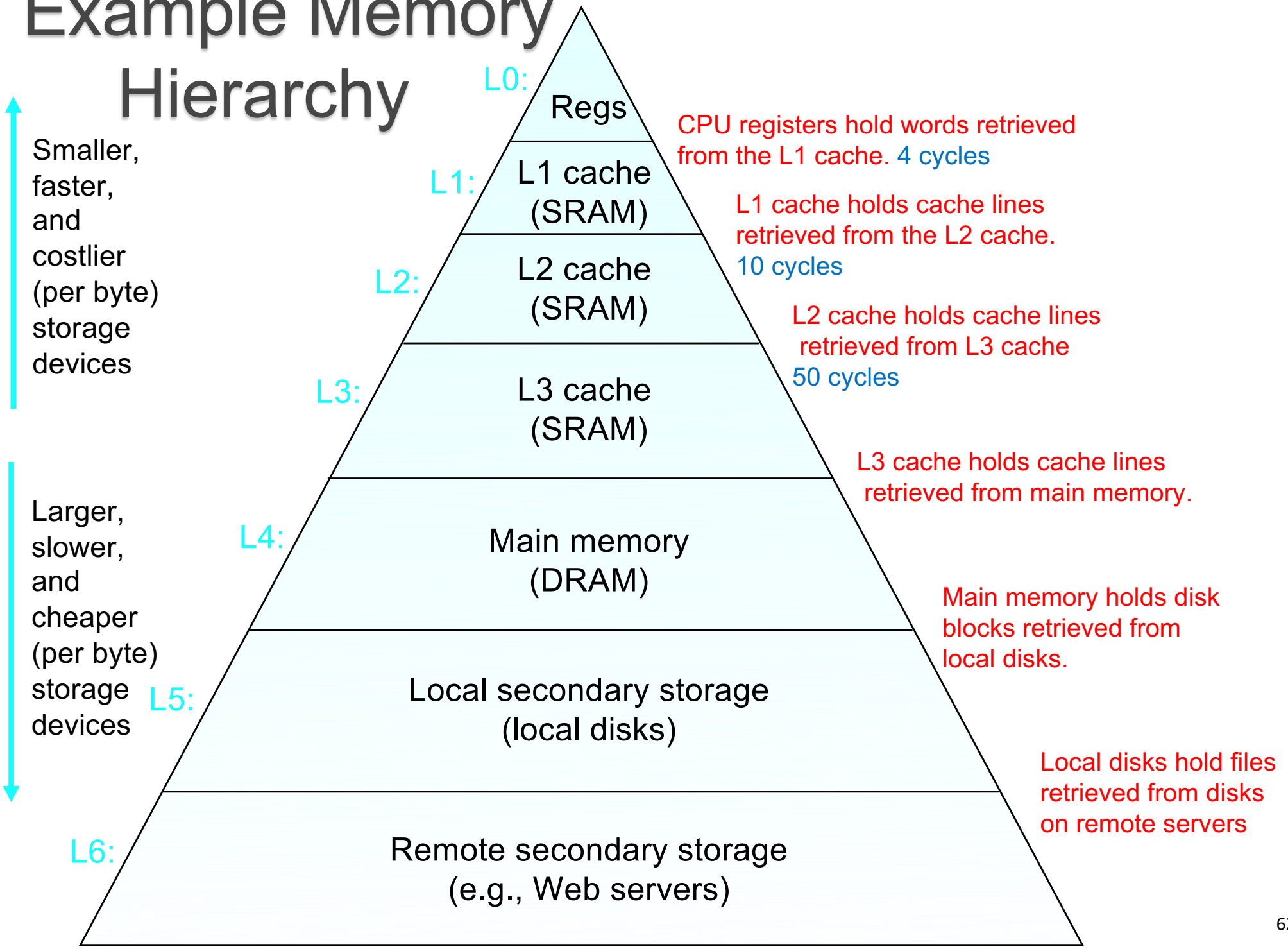
Topics

- Chapter 6: The Memory Hierarchy (2)
  - Detailed calculations of problem 6.4
  - Locality of reference
  - Caching in the memory hierarchy (Sec. 6.3 and 6.4)
    - general cache concepts: hit and miss
    - general cache organization and operation
    - Writes
    - Cache metrics
  - Performance impact of caches (Sec. 6.5)
  - The memory Mountain (Sec. 6.6)

Memory Hierarchies

- Some fundamental and enduring properties of hardware and software:
  - Fast storage technologies cost more per byte, have less capacity, and require more power (heat!).
  - The gap between CPU and main memory speed is widening.
  - Well-written programs tend to exhibit good locality.
- These fundamental properties complement each other beautifully.
- They suggest an approach for organizing memory and storage systems known as a memory hierarchy.

Example Memory Hierarchy



Caches

- *Cache:* A smaller, faster storage device that acts as a staging area for a subset of the data in a larger, slower device.
- Fundamental idea of a memory hierarchy:
  - For each k, the faster, smaller device at level k serves as a cache for the larger, slower device at level k+1.
- Why do memory hierarchies work?
  - Because of locality, programs tend to access the data at level k more often than they access the data at level k+1.
  - Thus, the storage at level k+1 can be slower, and thus larger and cheaper per bit.

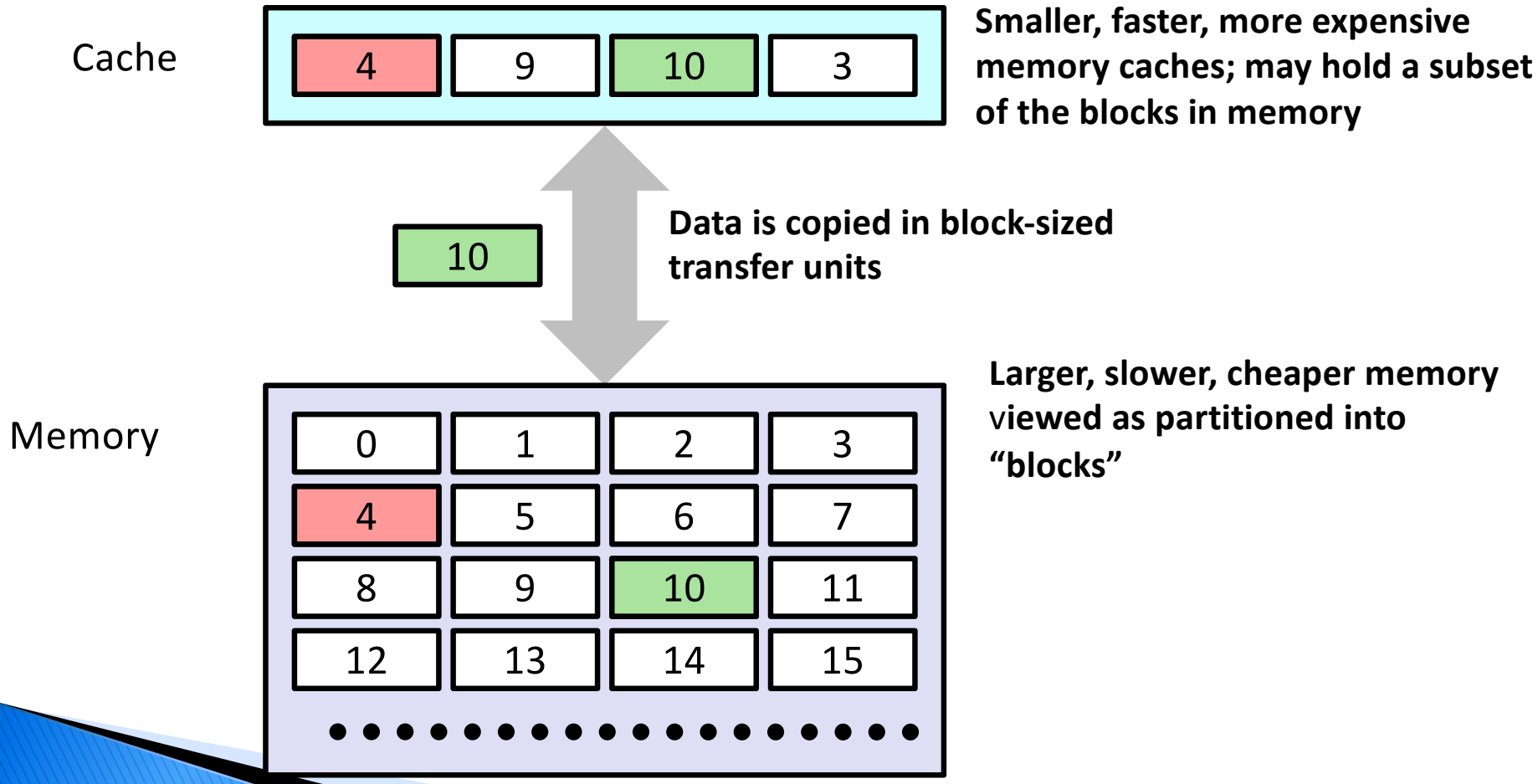
Goal of using caches

- *Big Idea*

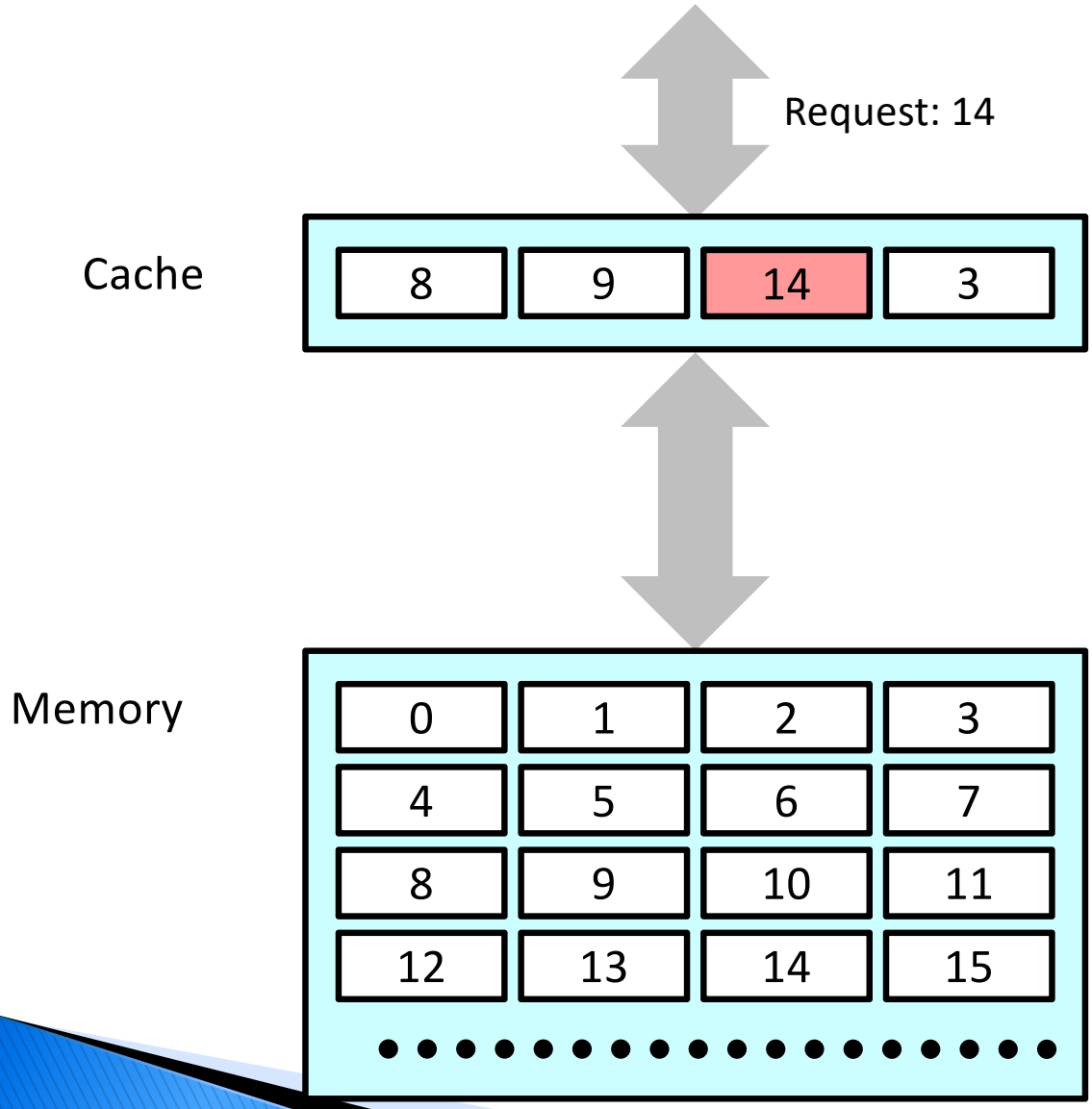
The memory hierarchy creates a large pool of storage that costs as much as the cheap storage near the bottom, but that provides data to programs at the rate of the fast storage near the top.

General Cache Concepts

process of using a cache is called *caching*



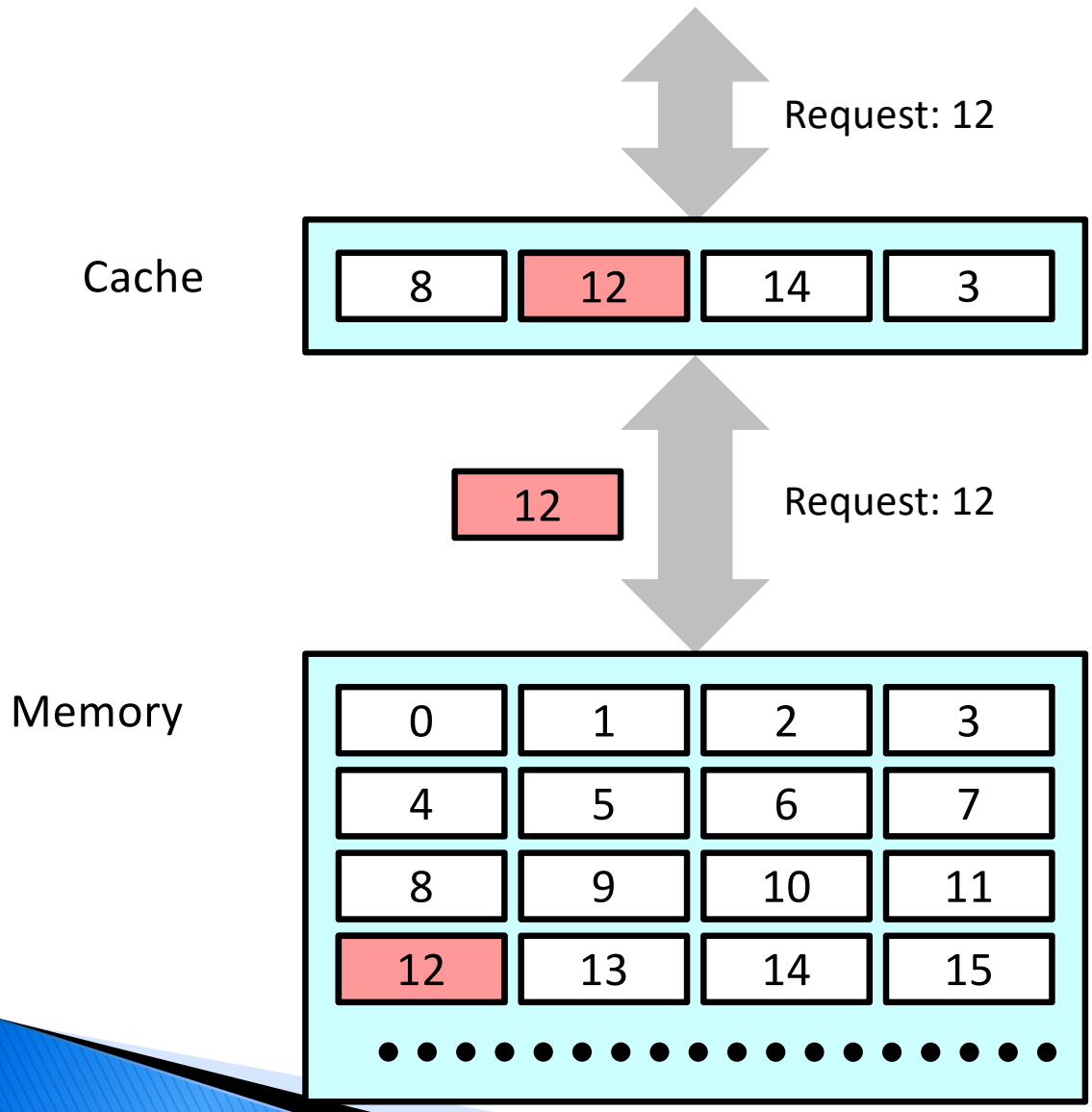
General Cache Concepts: Hit



Data in block b is needed

Block b is in cache:
cache Hit!

General Cache Concepts: Miss



Data in block b is needed

Block b is not in cache: cache Miss!

Block b is fetched from *memory*

Block b is stored in cache

- Replacement (eviction) policy: determines which block gets evicted (victim: random or LRU)
- Placement policy: determines where b goes

General Caching Concepts: Types of Cache Misses

- Cold (compulsory) miss
  - Cold misses occur because the cache is empty.
- Conflict miss
  - Most caches limit blocks at level k+1 to a small subset (sometimes a singleton) of the block positions at level k.
    - E.g. Block i at level k+1 must be placed in block (i mod 4) at level k.
  - Conflict misses occur when the level k cache is large enough, but multiple data objects all map to the same level k block.
    - E.g. Referencing blocks 0, 8, 0, 8, 0, 8, ... would miss every time.
- Capacity miss
  - program running as a sequence of phases (loops) access a (somewhat) constant set of cache blocks (*working set*).
  - When the size of the *working set* is larger than the size of the cache you get this kind of miss.

Examples of Caching in the Memory Hierarchy

Fig. 6.23 (to show several instances in which cache is used)

Cache Type	What is Cached?	Where is it Cached?	Latency (cycles)	Managed By
Registers	4-8 bytes words	CPU core	0	Compiler
TLB (translation lookaside buffer)	Address translations Chap. 9: Virtual Mem.	On-Chip TLB	0	Hardware MMU
L1 cache	64-byte blocks	On-Chip L1	4	Hardware
L2 cache	64-byte blocks	On-Chip L2	10	Hardware
Virtual Memory	4-KB pages	Main memory (DRAM)	100	Hardware + OS
Buffer cache	Parts of files	Main memory	100	OS
Disk cache	Disk sectors	Disk controller	100,000	Disk firmware
Network buffer cache	Parts of files	Local disk	10,000,000	NFS client
Browser cache	Web pages	Local disk	10,000,000	Web browser
Web cache	Web pages	Remote server disks	1,000,000,000	Web proxy server

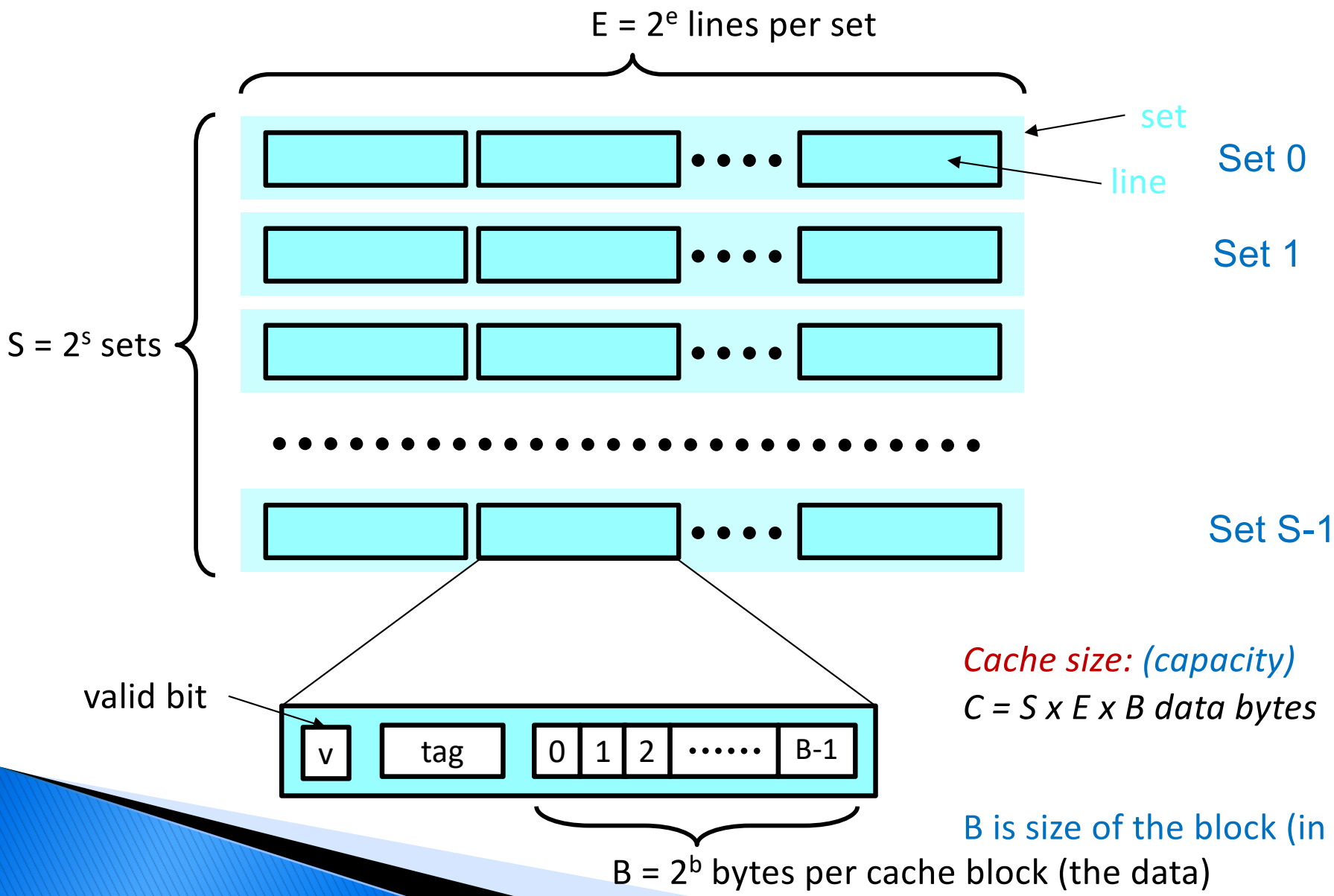
Summary (storage tech trends to locality)

- The speed gap between CPU, memory and mass storage continues to widen.
- Well-written programs exhibit a property called *locality*.
- Memory hierarchies based on *caching* close the gap by exploiting locality.

Topics

- Chapter 6: The Memory Hierarchy
  - Caching in the memory hierarchy (Sec. 6.3)
    - general cache concepts: hit and miss,
    - general cache organization and operation
    - Writes
    - Cache metrics
  - Performance impact of caches (Sec. 6.5)
  - The memory Mountain (Sec. 6.6)

General Cache Organization (S, E, B)



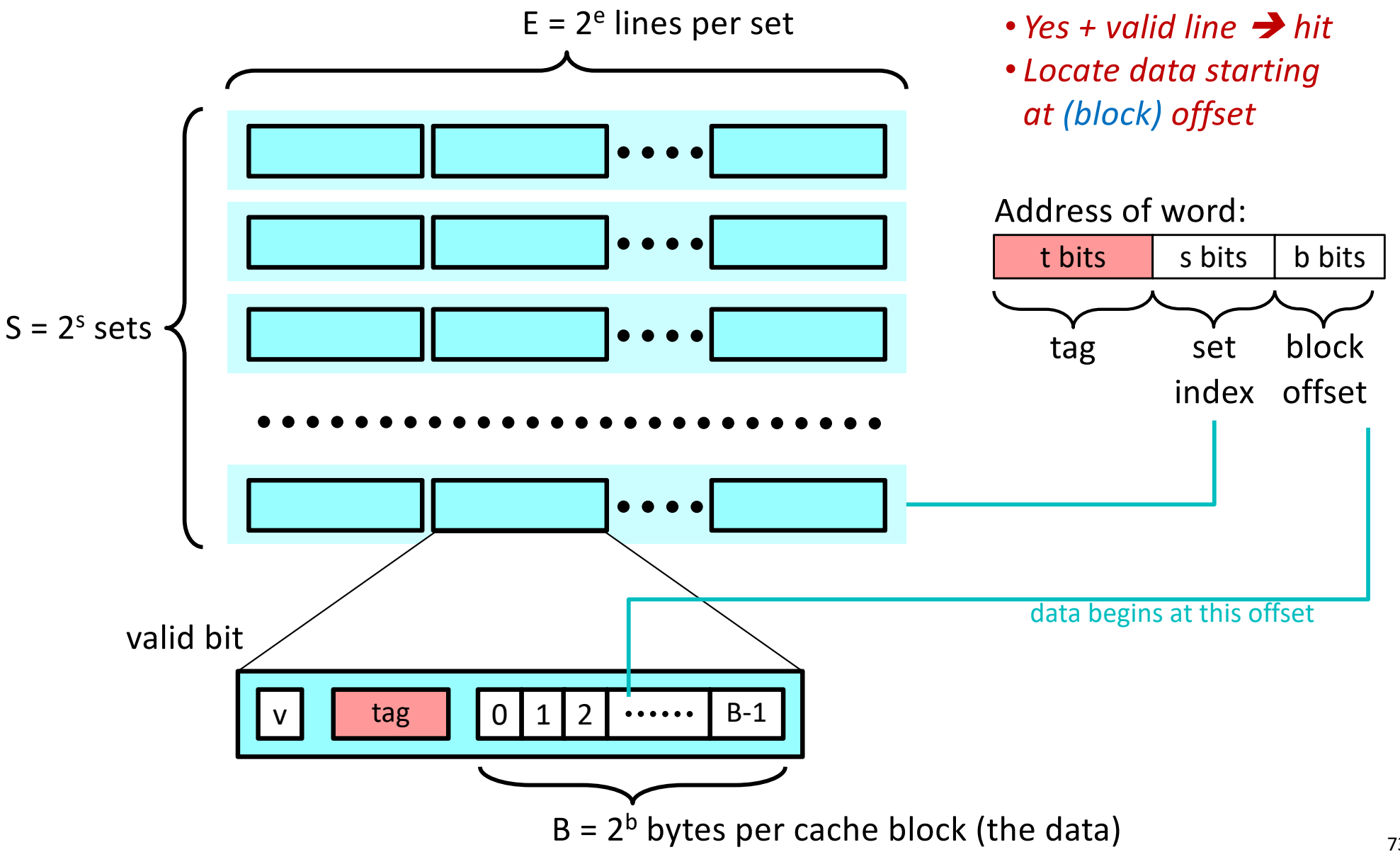
Cache size: (capacity)

C = S x E x B data bytes

B is size of the block (in cache)

Cache Read

- *Locate set (set index)*
- *Check if any line in set has matching tag*
- *Yes + valid line ➔ hit*
- *Locate data starting at (block) offset*



Practice Problem 6.9 (p. 616)

Cache	m	C	B	E	S	t	s	b
1	32	1,024	4	1				
2	32	1,024	8	4				
2	32	1,024	32	32				

$t = m - (b + s)$

$C = S \times E \times B$

To learn how to work with the parameters of cache

Practice Problem 6.9 (p. 616)

Cache	m	C	B	E	S	t	s	b
1	32	1,024	4	1				

S = number of sets

C = S x E x B ➔ S = C / B * E = 1024/4 = 256

Practice Problem 6.9 (p. 616)

Cache	m	C	B	E	S	t	s	b
1	32	1,024	4	1	256	22	8	2

S = number of sets

C = S x E x B ➔ S = C / B * E = 1024/4 = 256

$s = \log_2(256) = 8$

$b = \log_2(B) = \log_2(4) = 2$

m = t + s + b ➔ t = m – (b + s)

t = 32 – (2 + 8) = 22

Practice Problem 6.9 (p. 616)

Cache	m	C	B	E	S	t	s	b
1.	32	1,024	4	1	256	22	8	2
2.	32	1,024	8	4				
3.	32	1,024	32	32				

$t = m - (b + s)$

$C = S \times E \times B$

Fig. 6.26 (p.617) with summary of cache parameters: good to save for the cheat sheet.

Practice Problem 6.9 (p. 616)

Cache	m	C	B	E	S	t	s	b
1.	32	1,024	4	1	256	22	8	2
2.	32	1,024	8	4	32	24	5	3
3.	32	1,024	32	32	1	27	0	5

$t = m - (b + s)$

$C = S \times E \times B$

Fig. 6.26 (p.617) with summary of cache parameters: good to save for the cheat sheet.

Detailed content of section 6.4

- Cache memory organization and operation
  - Mapping of memory address to cache in detail
  - Simulation example of reads from memory, hit or miss?
  - E-way set associative cache
  - Writes
  - Cache metrics

Confirm

- what are S, E, and B?

- what are s, b, and t?

Questions (answers must be clear from today)

Cache organization

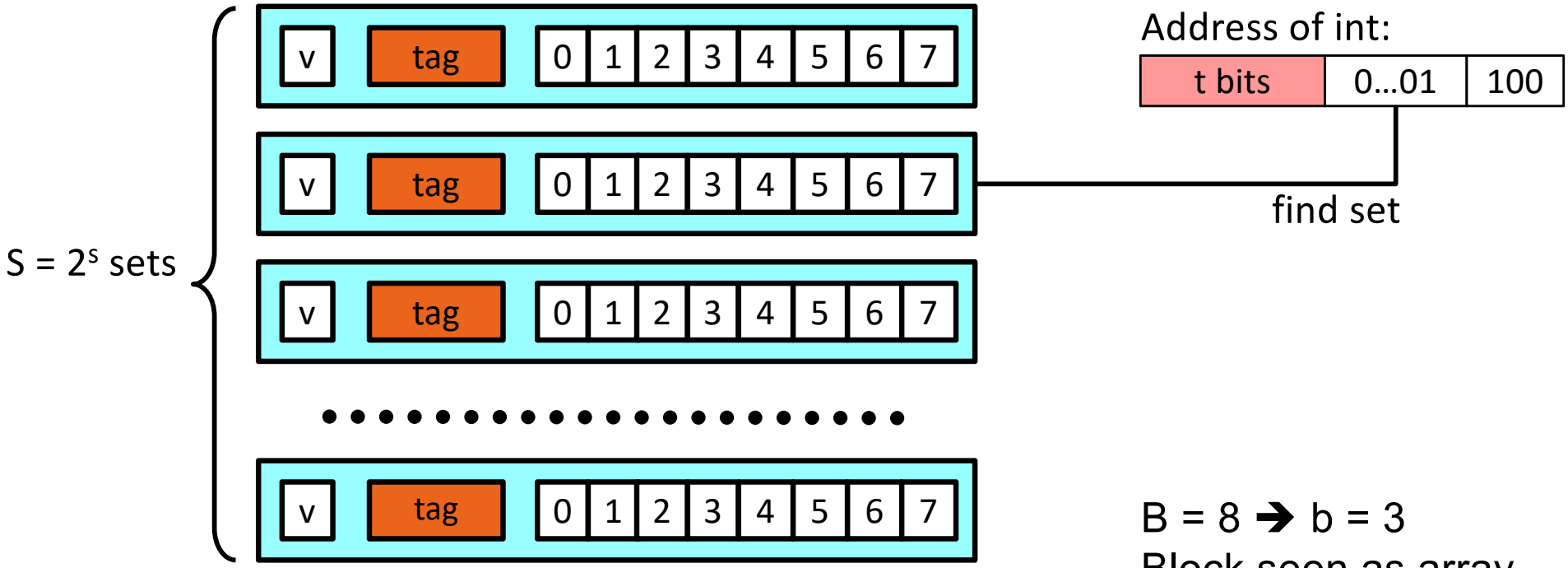
1. What is a line of cache?
2. What is the size of a line of cache? What is stored in a line?
3. How are memory addresses used when storing information in the cache?
4. What is `m`? how does it relate to the size of the cache?
5. List and describe the parameters that define a cache.

Example: Direct Mapped Cache (E = 1)

Direct mapped: one line per set

Assume: cache block size 8 bytes

Simplest to implement and understand



B = 8 ➔ b = 3

Block seen as array,

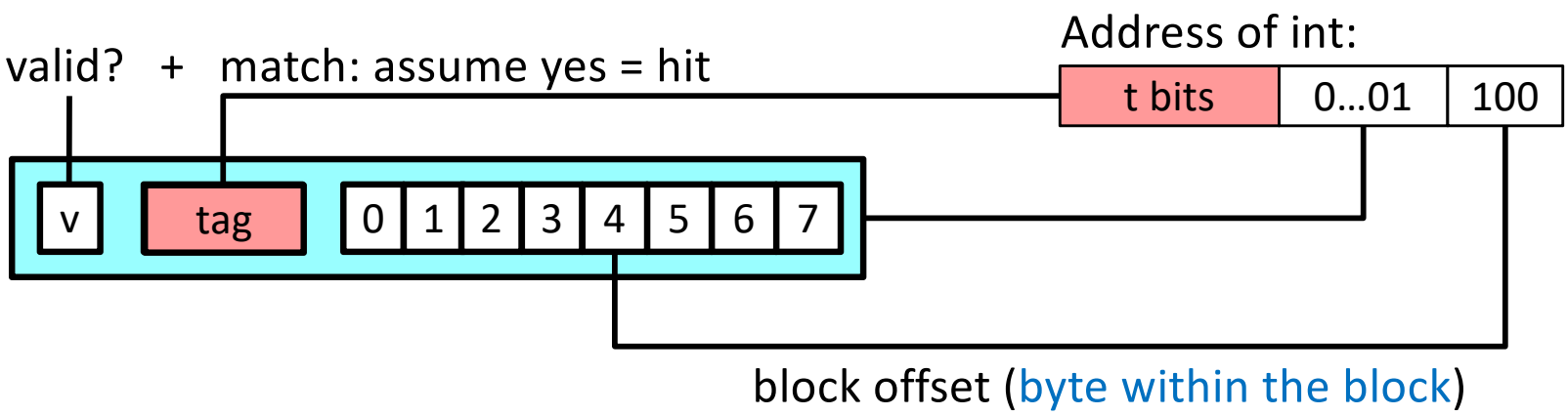
Indexed 0 to B-1

Example: Direct Mapped Cache (E = 1)

Direct mapped: One line per set

Assume: cache block size 8 bytes

Mapping the memory address to cache:



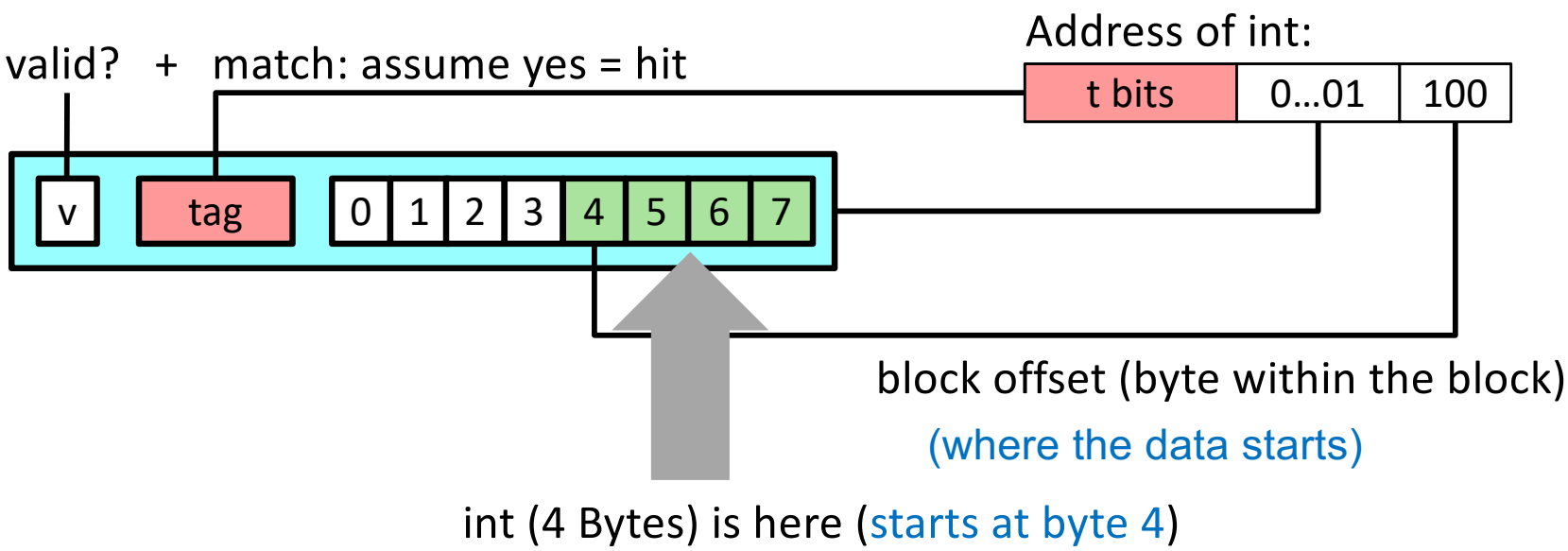
hit or miss & extract process: 1) set selection (using middle s bits)

2) line matching + validity checking

3) word extraction

Example: Direct Mapped Cache (E = 1)

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



If tag doesn't match: old line is evicted and replaced
(no alternative since there is only 1 line per set)

Tiny address space example

Direct-mapped cache
(S,E,B,m) = (4,1,2,4)

m is size of address ➔ 4-bit address

Address (in mem) 0-15

t = 1 s = 2 b = 1

Block number (in cache) 0-3

Mapping of addresses to cache set

Address	Set	Address	Set
0 (0000)	0	8 (1000)	0
1 (0001)	0	9 (1001)	0
2 (0010)	1	10 (1010)	1
3 (0011)	1	11 (1011)	1
4 (0100)	2	12 (1100)	2
5 (0101)	2	13 (1101)	2
6 (0110)	3	14 (1110)	3
7 (0111)	3	15 (1111)	3

(keep this mapping to follow the simulation)

4-bit addresses (tssb) 2 middle bits for set (to match simulation)
filled out this table with the 4-bit address space

Simulation of the cache

- Picture the cache:
  - 4 sets, 1 line in each, 2 bytes per line
  - set number is not in the cache, only valid bit, tag, and the actual content (data) of the 2 bytes (which are brought from memory)
- Cache is Initially empty all (4, why 4?) valid bits are 0

Direct-Mapped Cache Simulation

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

t = m – (s+b)= 4-3 = 1

M=16 bytes (4-bit addresses), B=2 bytes/block,
S=4 sets, E=1 line/set
(S,E,B,m) = (4,1,2,4) C = 8 bytes

Address trace (reads, one byte per read):

0	$[0\underline{00}0_2]$,	miss
1	$[0\underline{00}1_2]$,	hit
7	$[0\underline{11}1_2]$,	miss
8	$[1\underline{00}0_2]$,	miss
0	$[0\underline{00}0_2]$	miss

underlined set Index.
Order in which these addesses are required

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

Continue the simulation with this cache

- Status as shown in previous slide
- Read word at address 1 = 0001, which set?
  - hit or miss?

- Read word at address 13 = 1101
  - hit or miss?

Example is explained in textbook

- What we just showed is seen through the sequence of reads described on pages 620-621, make sure you understand what happens to the cache on each step.
- Questions about the process and when to know if it is a hit or miss?

The cachelab is coming up

- Last big project of the semester
- All you need for this project is to understand how the cache works.
- Statement will be published when we have covered all of the cache behavior.
- First a cache homework, to make sure you understand