Memory Hierarchy and Intro to Caching

CSCI 237: Computer Organization
22nd Lecture, Apr 14, 2025

Jeannie Albrecht

Administrative Details

- Lab 4 – Bitmap
  - Due Tue/Wed
- Lab 5 – Cache simulator
  - Read Ch 6.4 before lab! (FYI: I need to proofread it)
  - Submit partner form
- Glow HW
  - Due Friday at noon
  - Practice problems mostly related to caching (and one pipeline)
- TA eval form and apps due this week

Last time

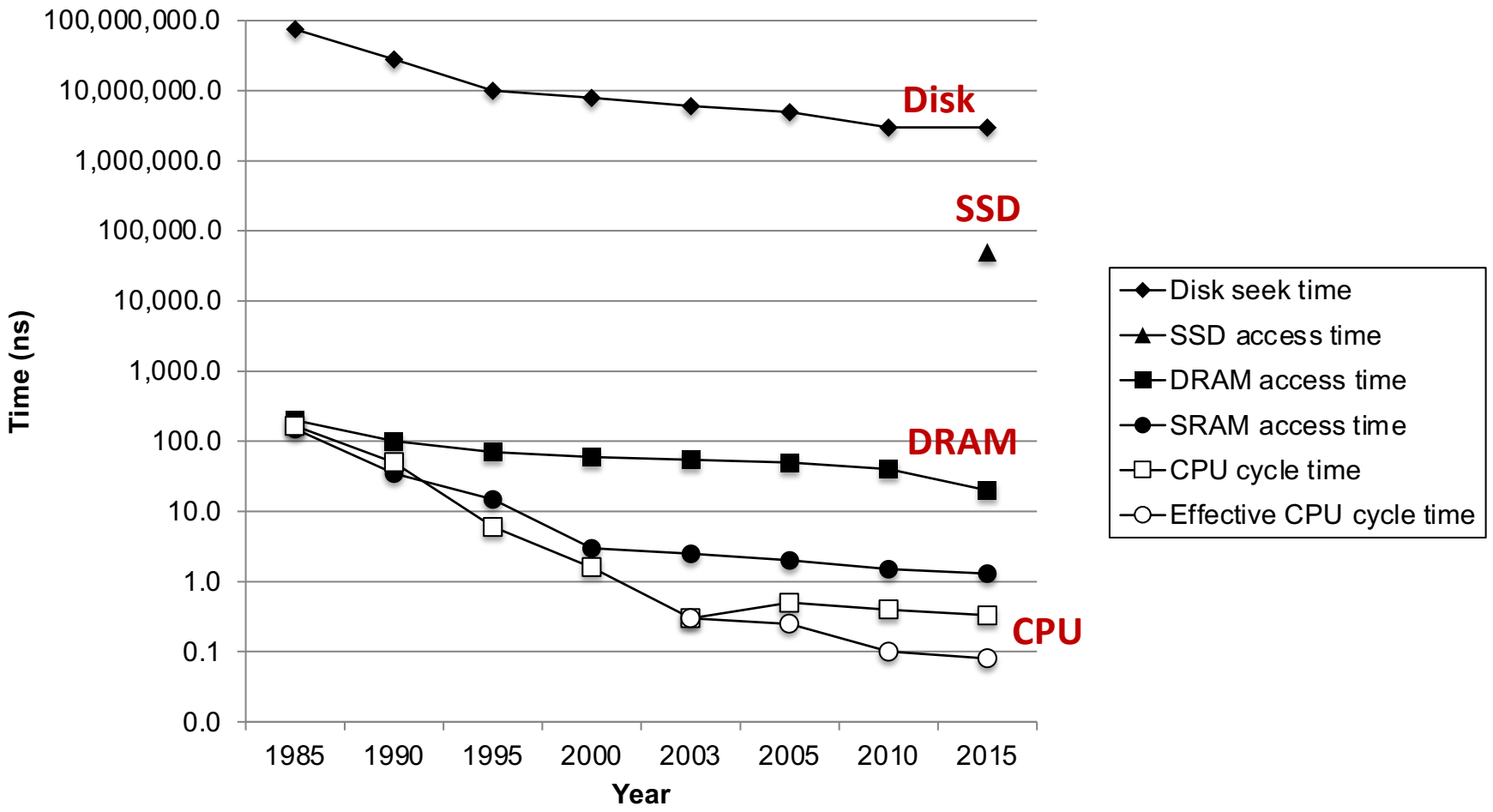
- Storage technologies and trends (Ch 6.1)
  - Memory technologies
  - Disk storage
  - Solid state disks

Today

- Locality of reference (Ch 6.2)
- The memory hierarchy (Ch 6.3)
- Cache memory organization and operation (Ch 6.4)

Problem: The CPU-Memory Gap

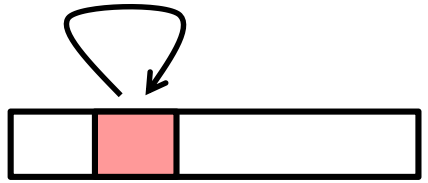
The gap *widens* between DRAM, disk, and CPU speeds.



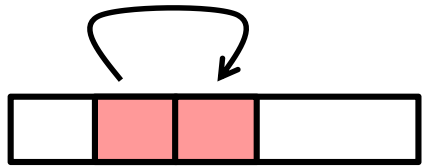
Locality to the Rescue!

- Principle of Locality: Programs tend to use data and instructions with addresses near or equal to those they have used recently

- Temporal locality:
  - Recently referenced items are likely to be referenced again in the near future



- Spatial locality:
  - Items with nearby addresses tend to be referenced close together in time



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`? Yes!

```
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`? No!

```
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 3D 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;
}
```

Locality Example

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

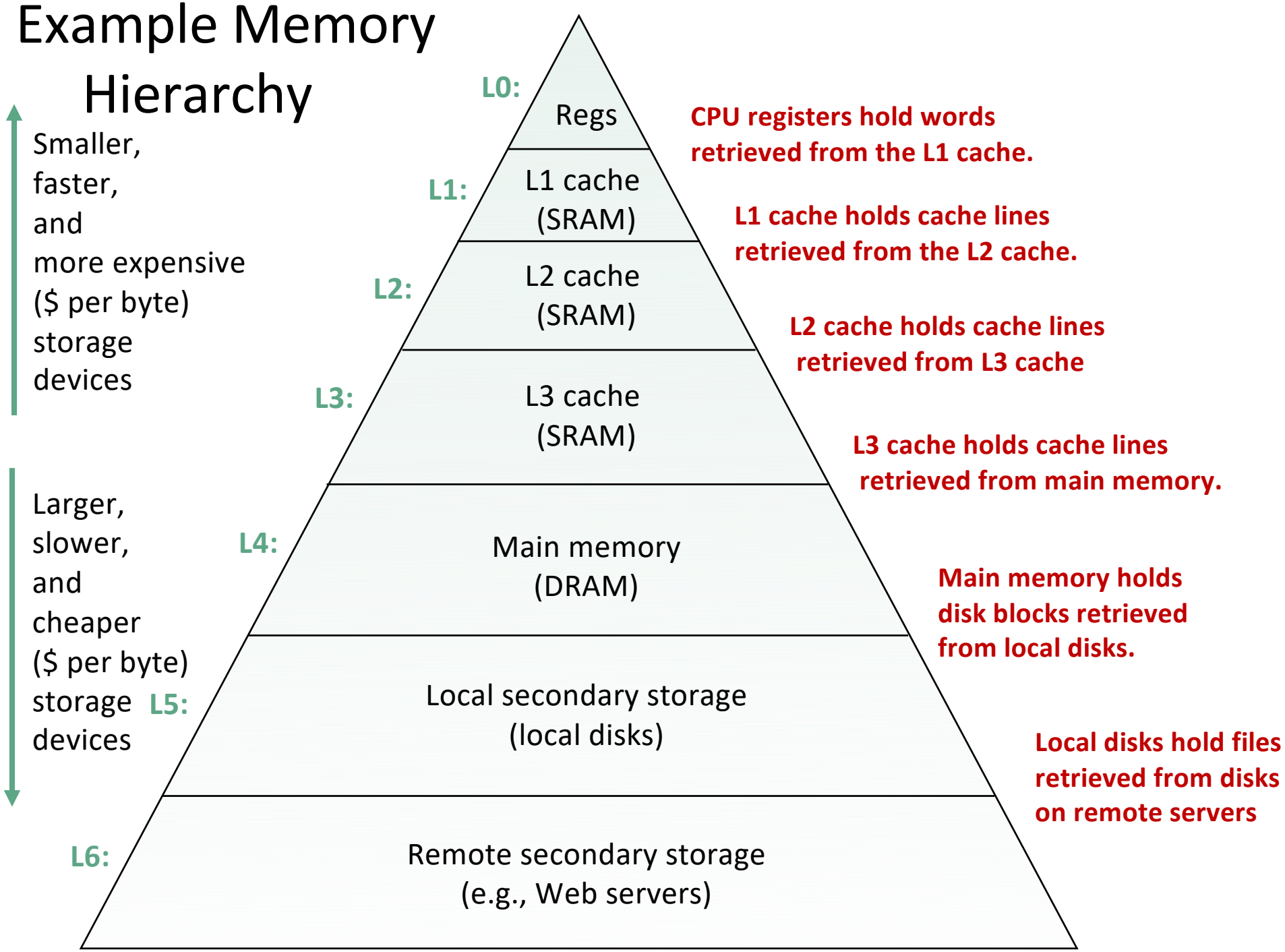
```
int sum_array_3d(int a[M][N][N])
{
    int i, j, k, sum = 0;

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

Rightmost indices should change "most rapidly"

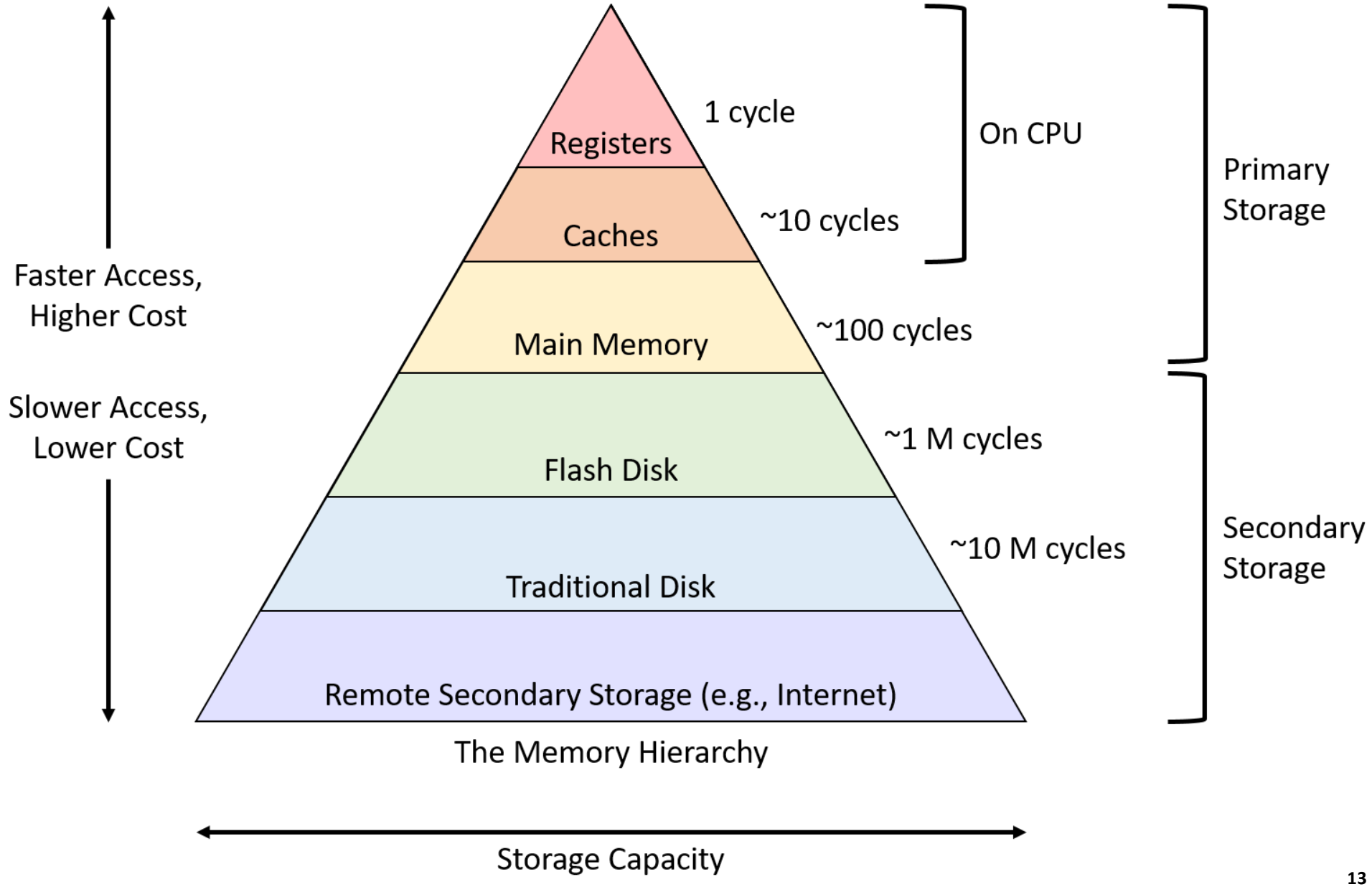
Memory Hierarchies

- 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.
- They suggest an approach for organizing memory and storage systems known as a memory hierarchy.

Example Memory Hierarchy



Another Example Memory Hierarchy



Processor

13-inch model

2.0GHz dual-core Intel Core i5, Turbo Boost up to 3.1GHz, with 4MB shared L3 cache

Configurable to 2.4GHz dual-core Intel Core i7, Turbo Boost up to 3.4GHz, with 4MB shared L3 cache

13-inch model with Touch Bar

2.9GHz dual-core Intel Core i5, Turbo Boost up to 3.3GHz, with 4MB shared L3 cache

Configurable to 3.1GHz dual-core Intel Core i5, Turbo Boost up to 3.5GHz, with 4MB shared L3 cache; or 3.3GHz dual-core Intel Core i7, Turbo Boost up to 3.6GHz, with 4MB shared L3 cache

Storage[1]

256GB

256GB PCIe-based onboard SSD

Configurable to 512GB or 1TB SSD

256GB

256GB PCIe-based onboard SSD

512GB

512GB PCIe-based onboard SSD

Configurable to 1TB SSD

Memory

8GB

8GB of 1866MHz LPDDR3 onboard memory

Configurable to 16GB of memory

8GB

8GB of 2133MHz LPDDR3 onboard memory

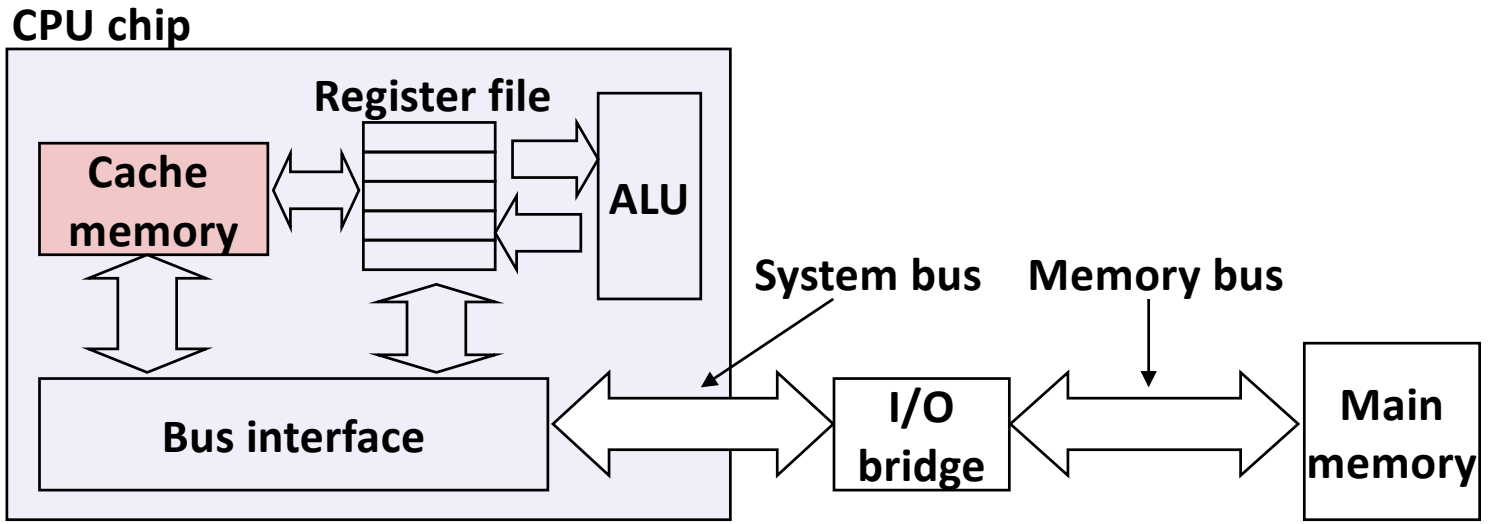
Configurable to 16GB of memory

Caches

- *Cache (pronounced "cash"):* 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 Lk serves as a cache for the larger, slower device at level Lk+1.
- Why do memory hierarchies work?
  - Because of locality, programs tend to access the data at level Lk more often than they access the data at level Lk+1.
  - The storage at level Lk+1 can be slower, and thus larger and cheaper per bit.
- *Big Idea:* The memory hierarchy creates a large pool of storage that costs as much as the cheap storage near the bottom, but that serves data to programs at the rate of the fast storage near the top.

Cache Memories

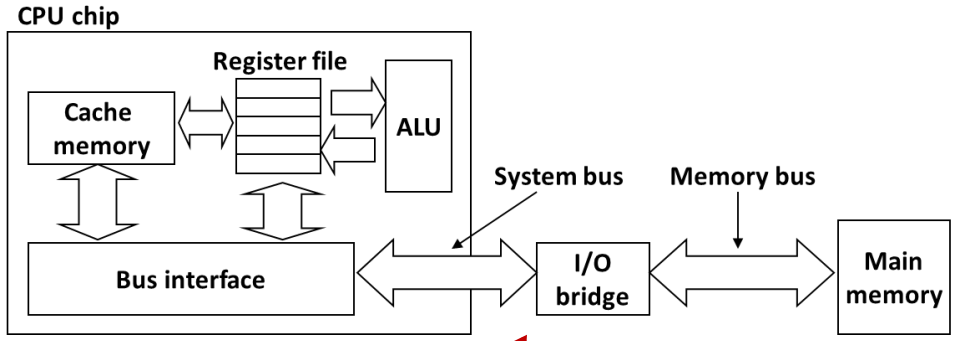
- Cache memories are small, fast SRAM-based memories managed automatically in hardware
  - Hold frequently accessed blocks of main memory
  - Part of CPU chip
- CPU looks first for data in cache
- Typical system structure:



What it Really Looks Like

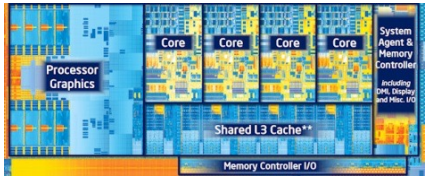
Desktop PC

Source: Dell



CPU (Intel Core i7)

Source: PC Magazine



Source: techreport.com

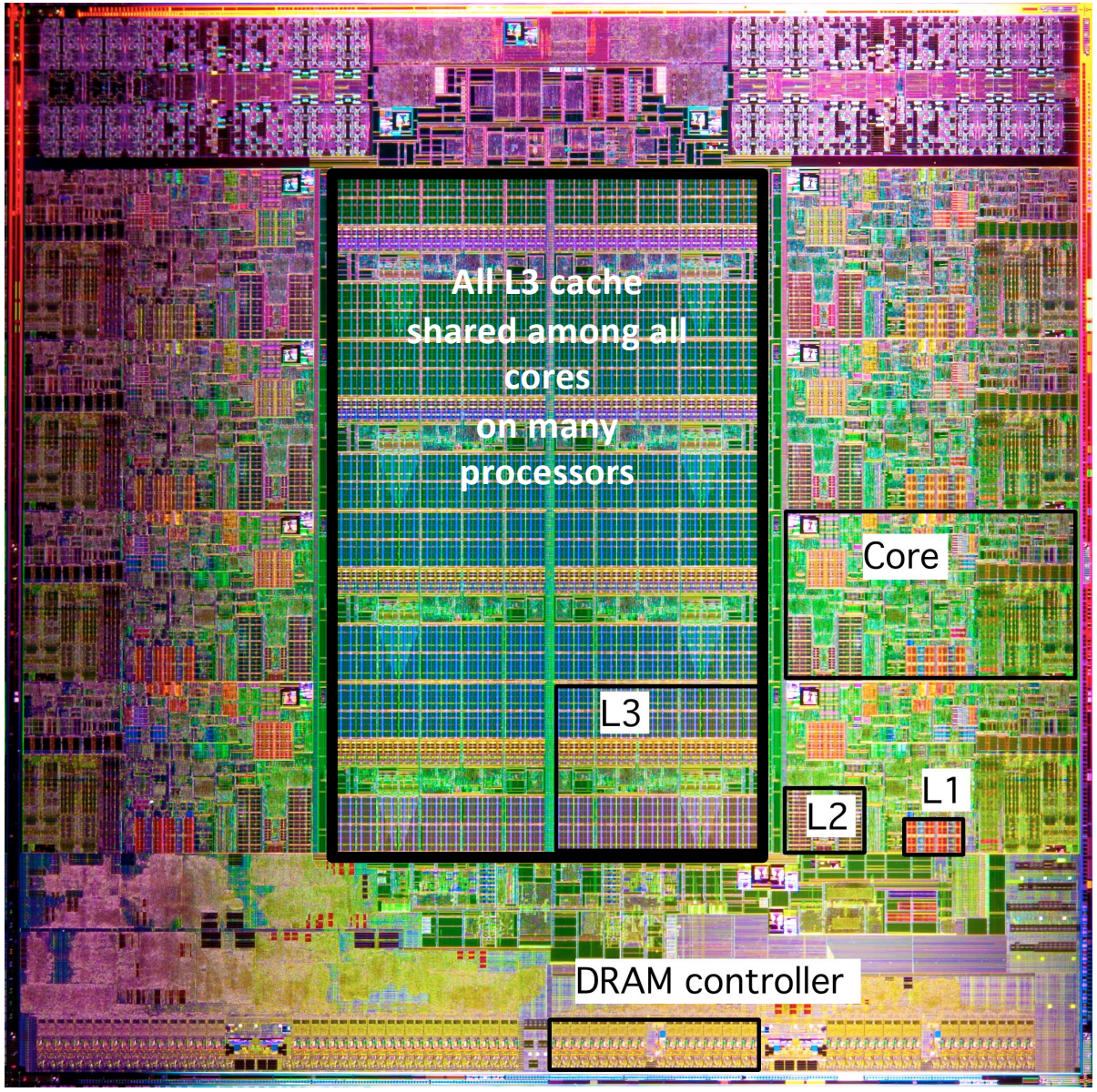
Main memory (DRAM)

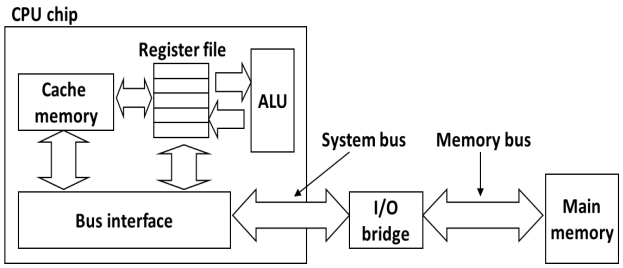
Source: Dell

Motherboard

Source: Dell

What it Really Looks Like (Cont.)

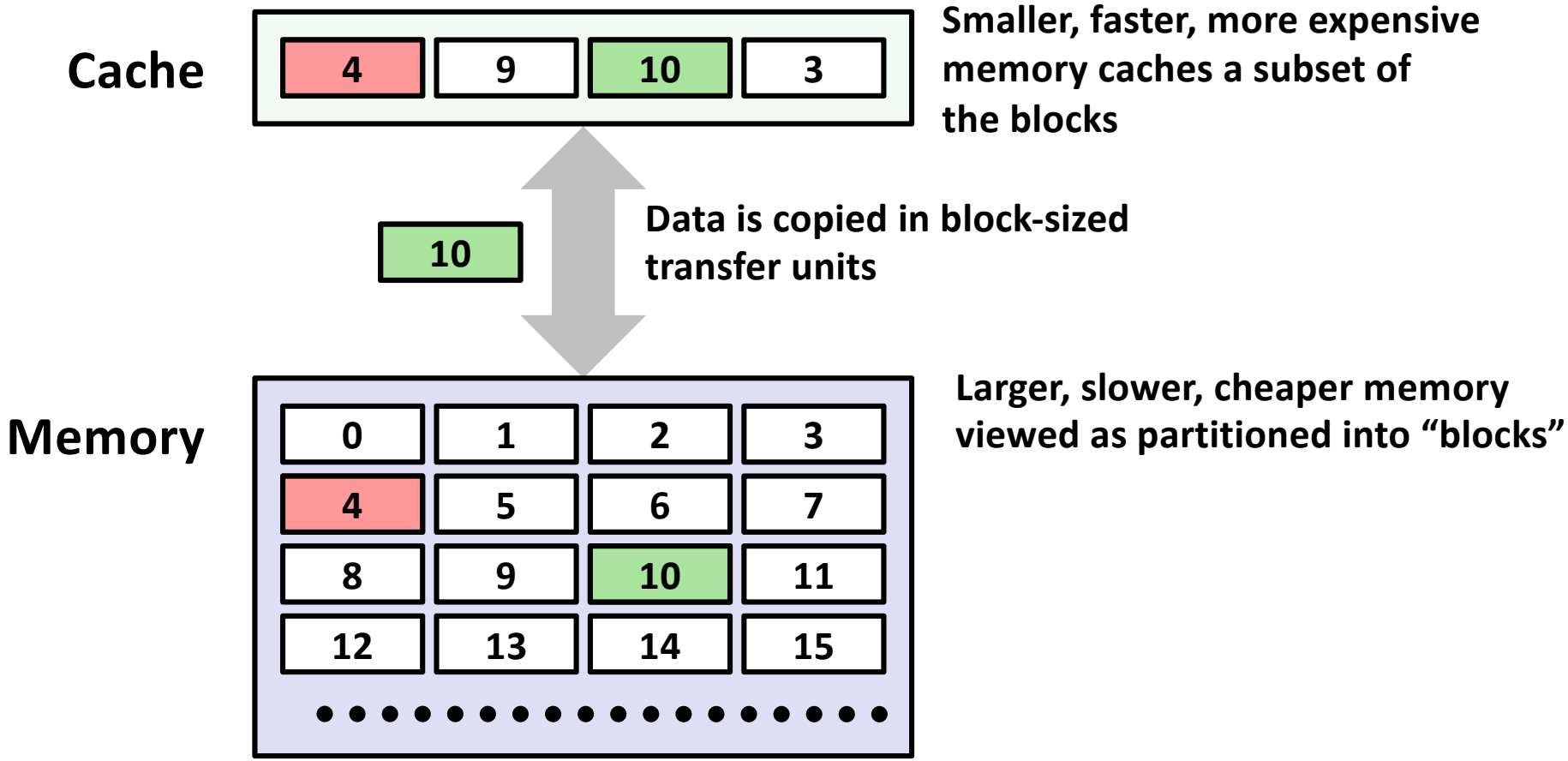




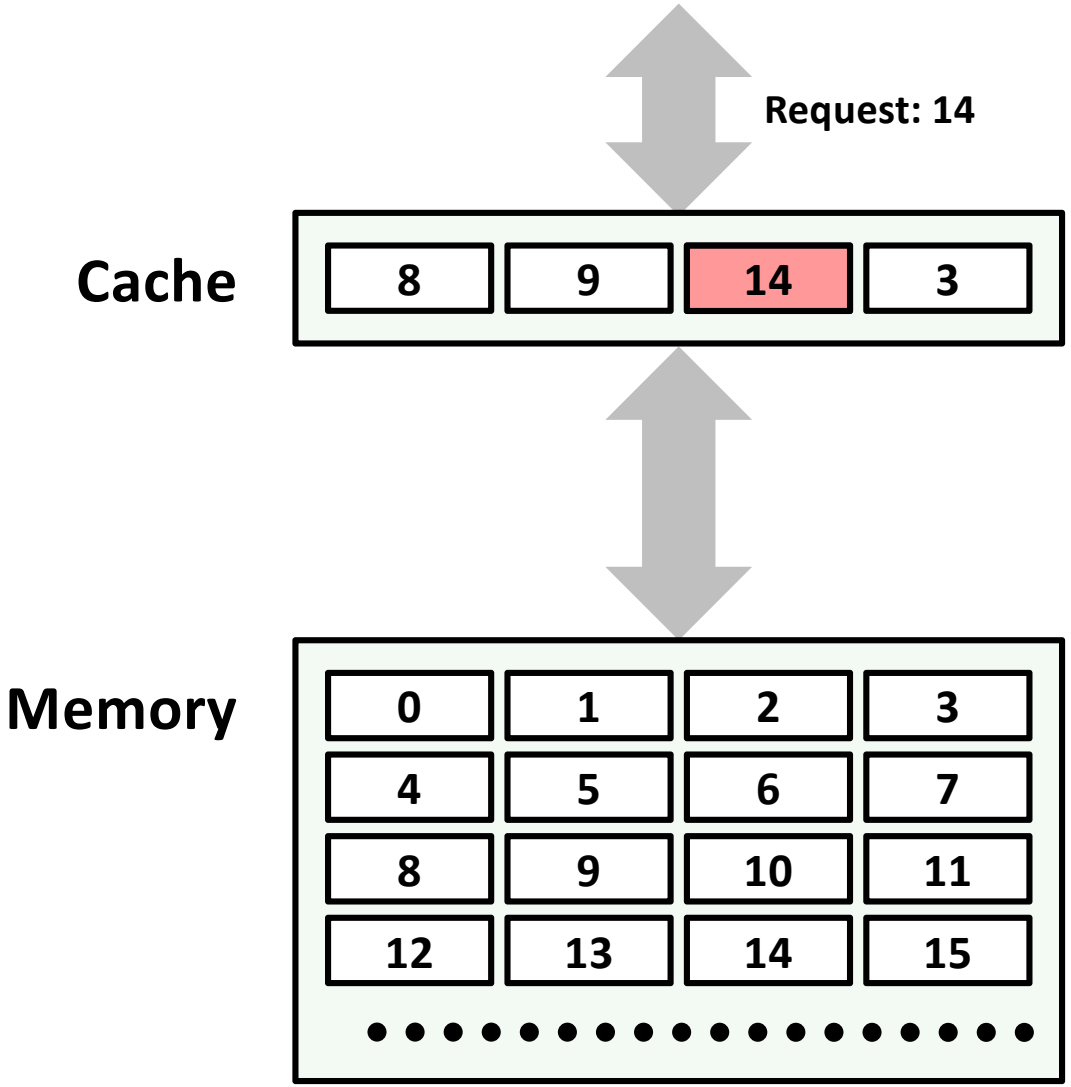
Intel Sandy Bridge (e.g., all Intel Core processors) Processor Die

L1: 32KB Instr + 32KB Data (per core)
L2: 256KB (per core)
L3: 3–20MB (MUCH bigger and shared)

General Cache Concepts



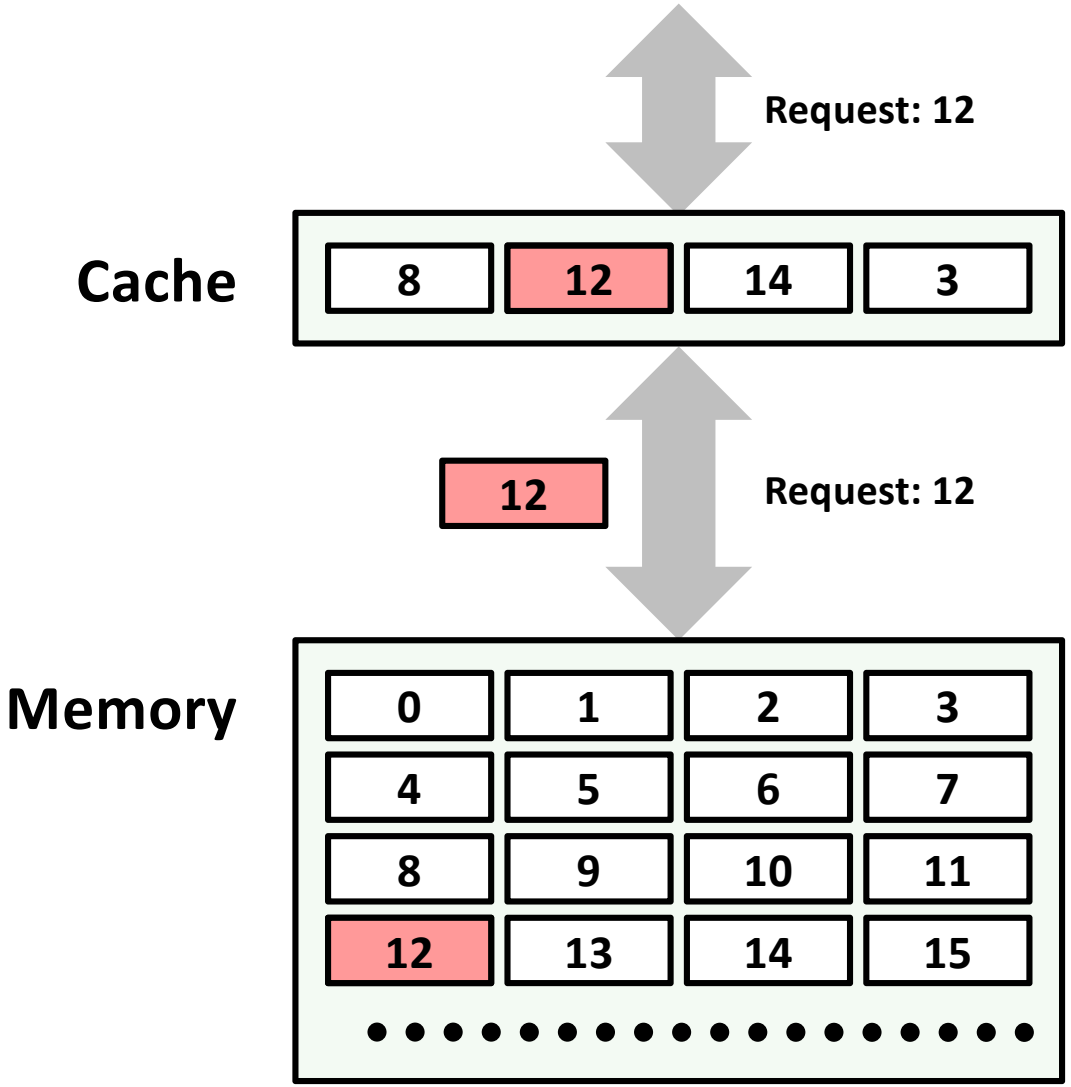
General Cache Concepts: Hit



Data in block b is needed

Block b is in cache:
Hit!

General Cache Concepts: Miss



Data in block b is needed

Block b is not in cache:
Miss!

Block b is fetched from memory

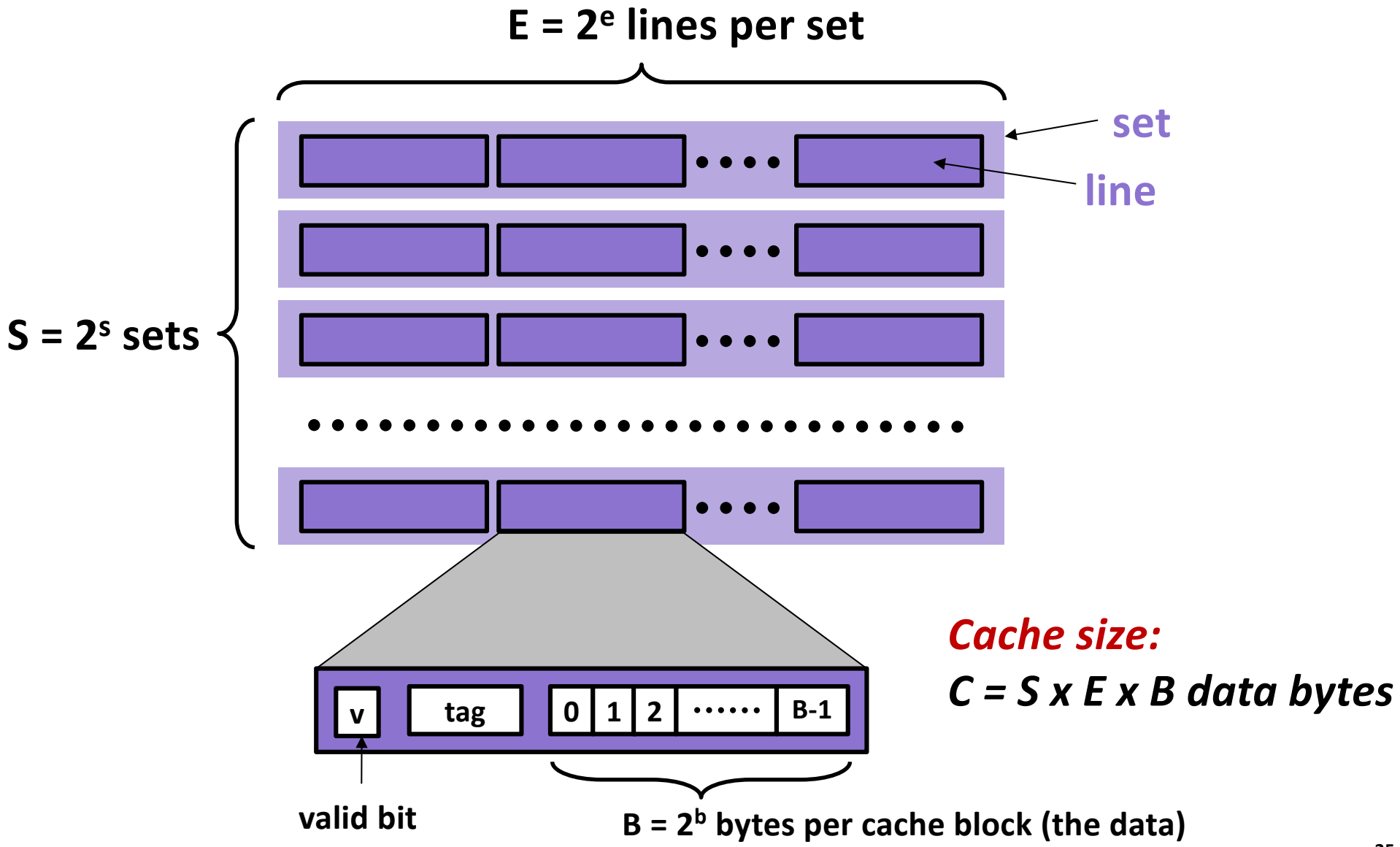
Block b is stored in cache

- Placement policy:
  determines where b goes
- Replacement policy:
  determines which block
  gets *evicted* (called the victim)

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
  - Occurs when the set of active cache blocks (working set) is larger than the cache.

General Cache Organization (S, E, B)

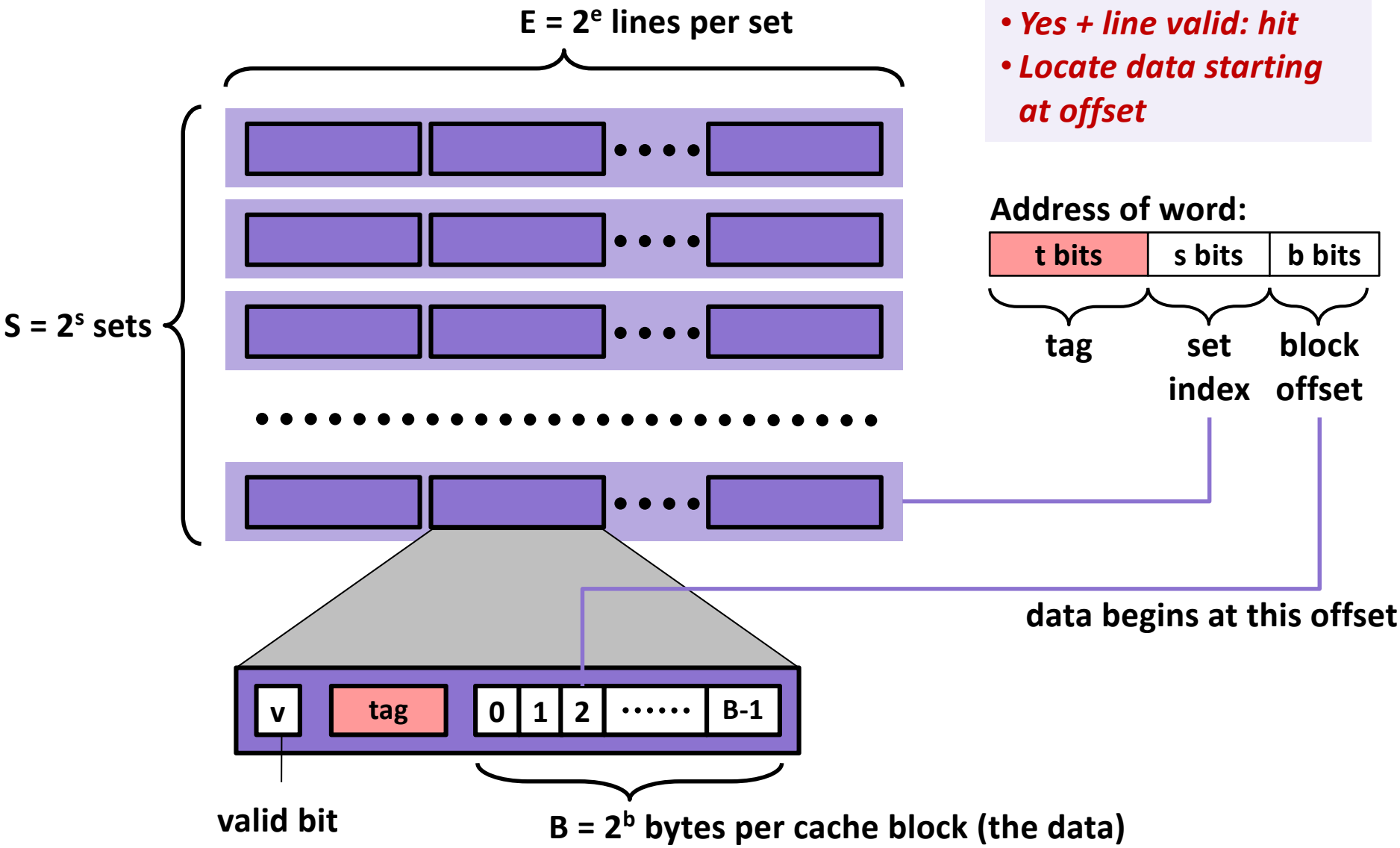


Cache size:

C = S x E x B data bytes

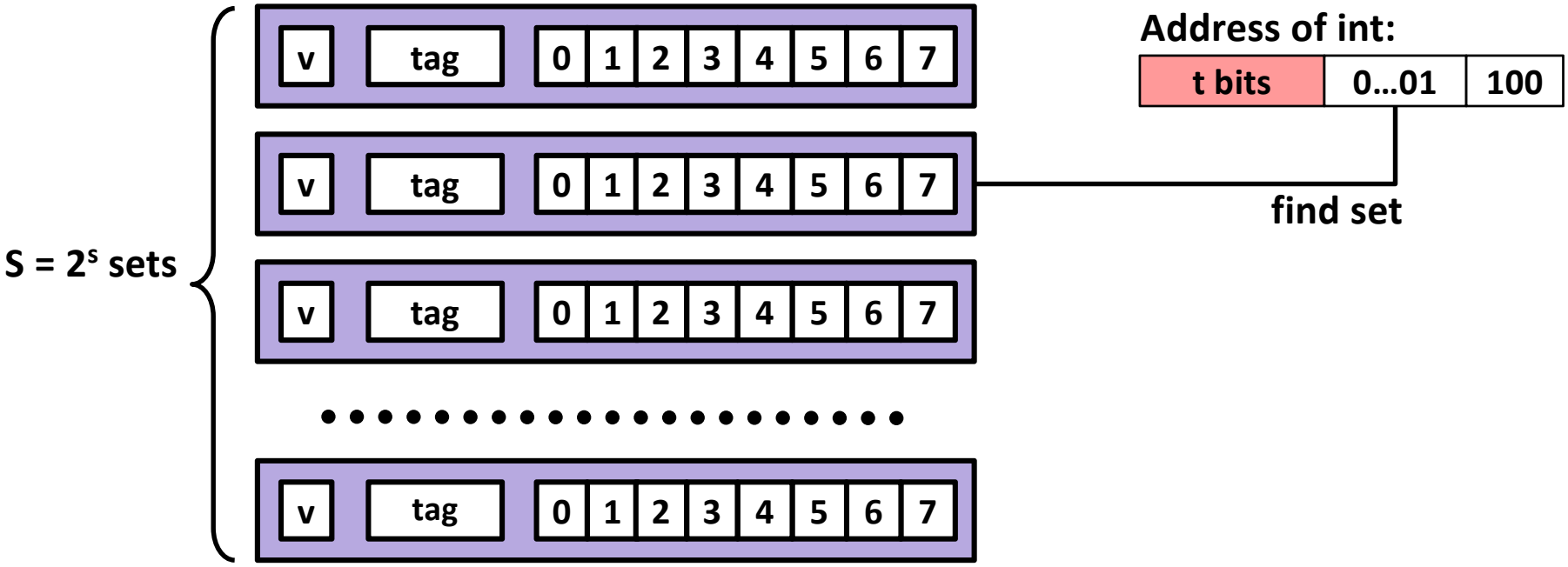
Cache Read

- *Locate set*
- *Check if any line in set has matching tag*
- *Yes + line valid: hit*
- *Locate data starting at offset*



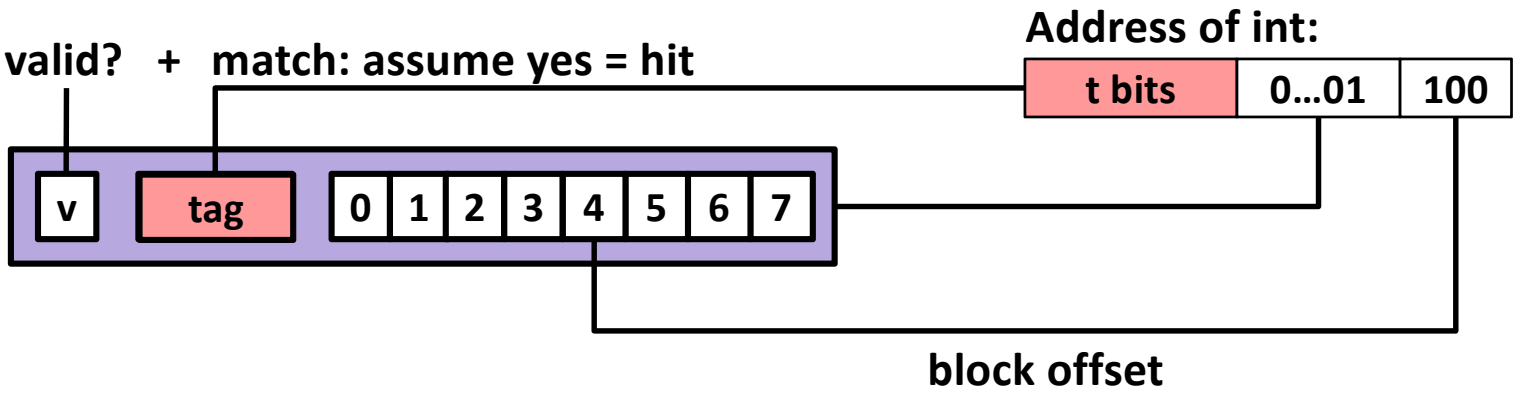
Example: Direct Mapped Cache (E = 1)

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



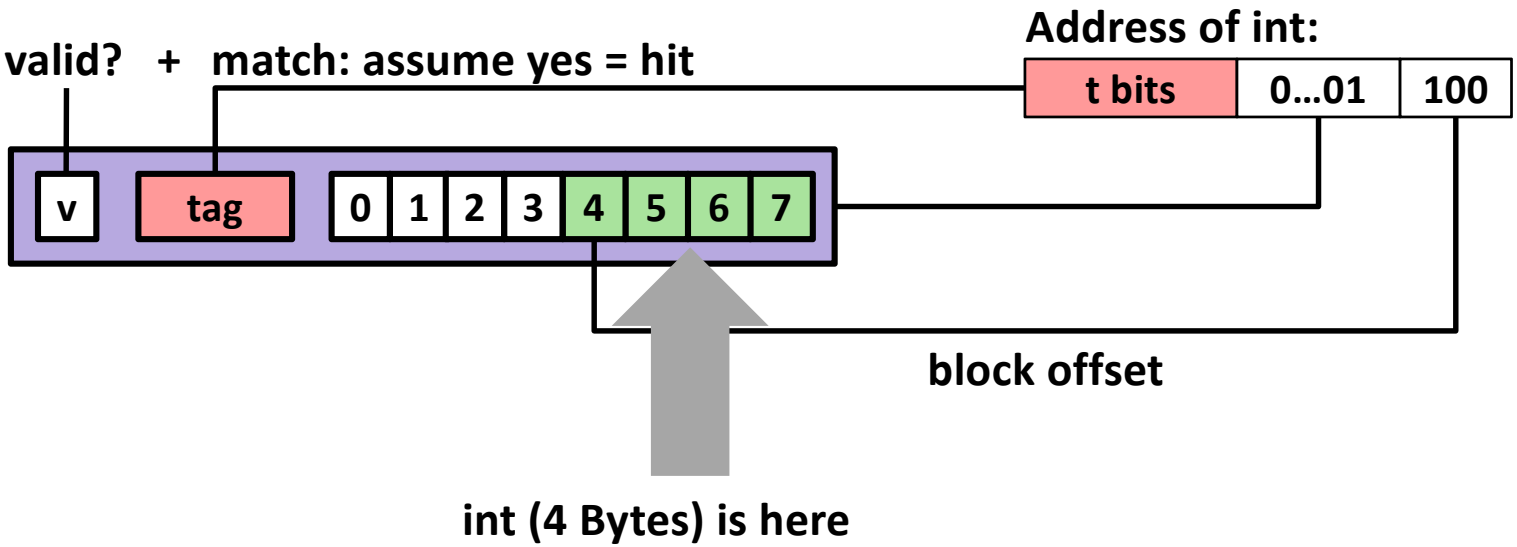
Example: Direct Mapped Cache (E = 1)

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



If tag doesn't match: old line is *evicted* and replaced

Direct-Mapped Cache Simulation

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

M=16 bytes (4-bit addresses), B=2 bytes/block,
S=4 sets, E=1 line/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]**	miss

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