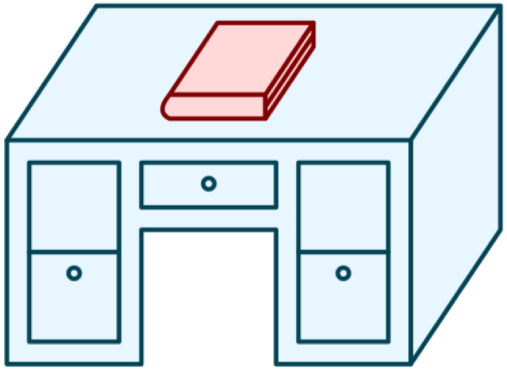
Lecture 12: Caches

CS 105

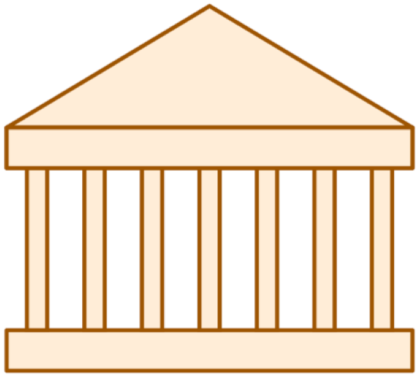
March 4, 2019

Life without caches

- Imagine that you have a midterm coming up in your systems class next week (this should be easy to imagine!) and you decide it's time to learn everything there is to know about computer systems.
- The library contains all the books you could possibly want, but you don't like to study in libraries, you prefer to study at home.
- You have the following constraints:

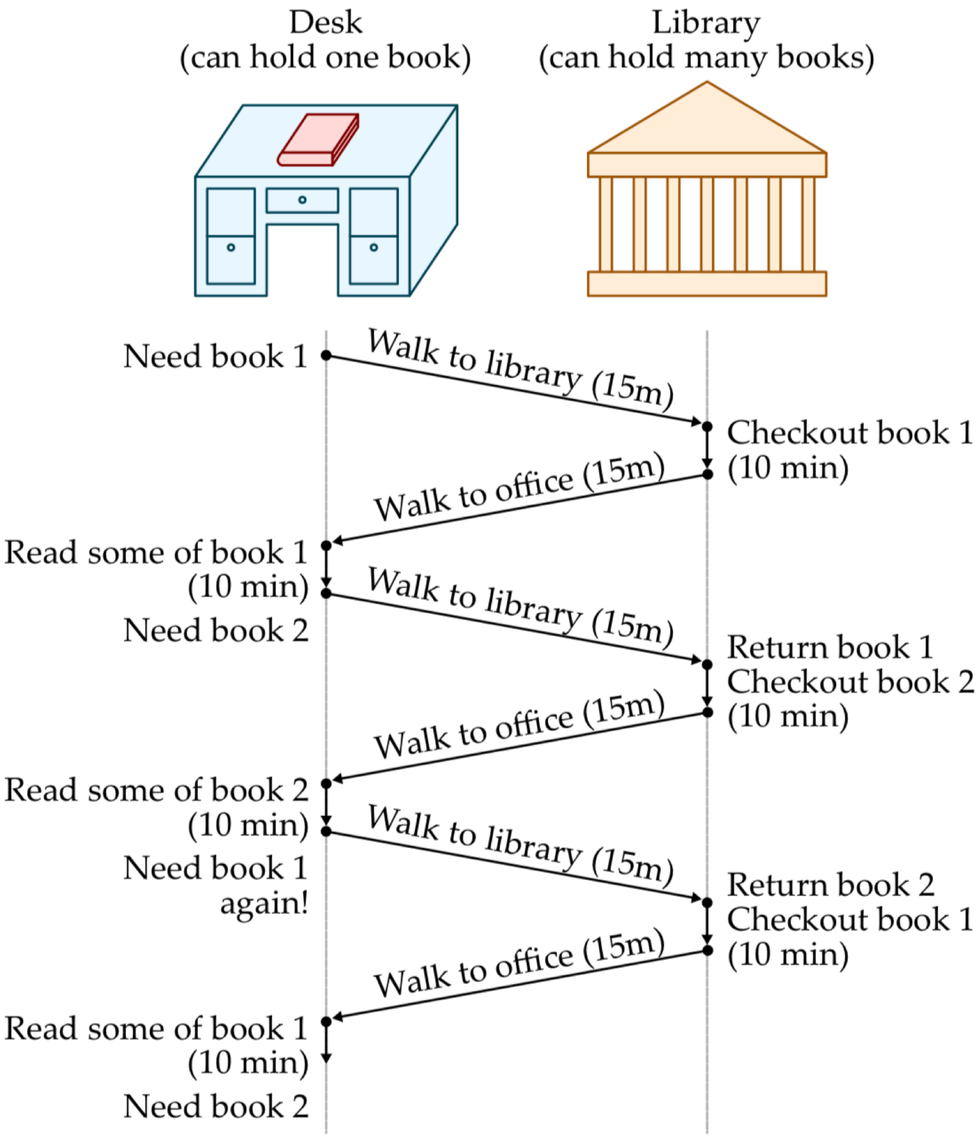


Desk
(can hold one book)



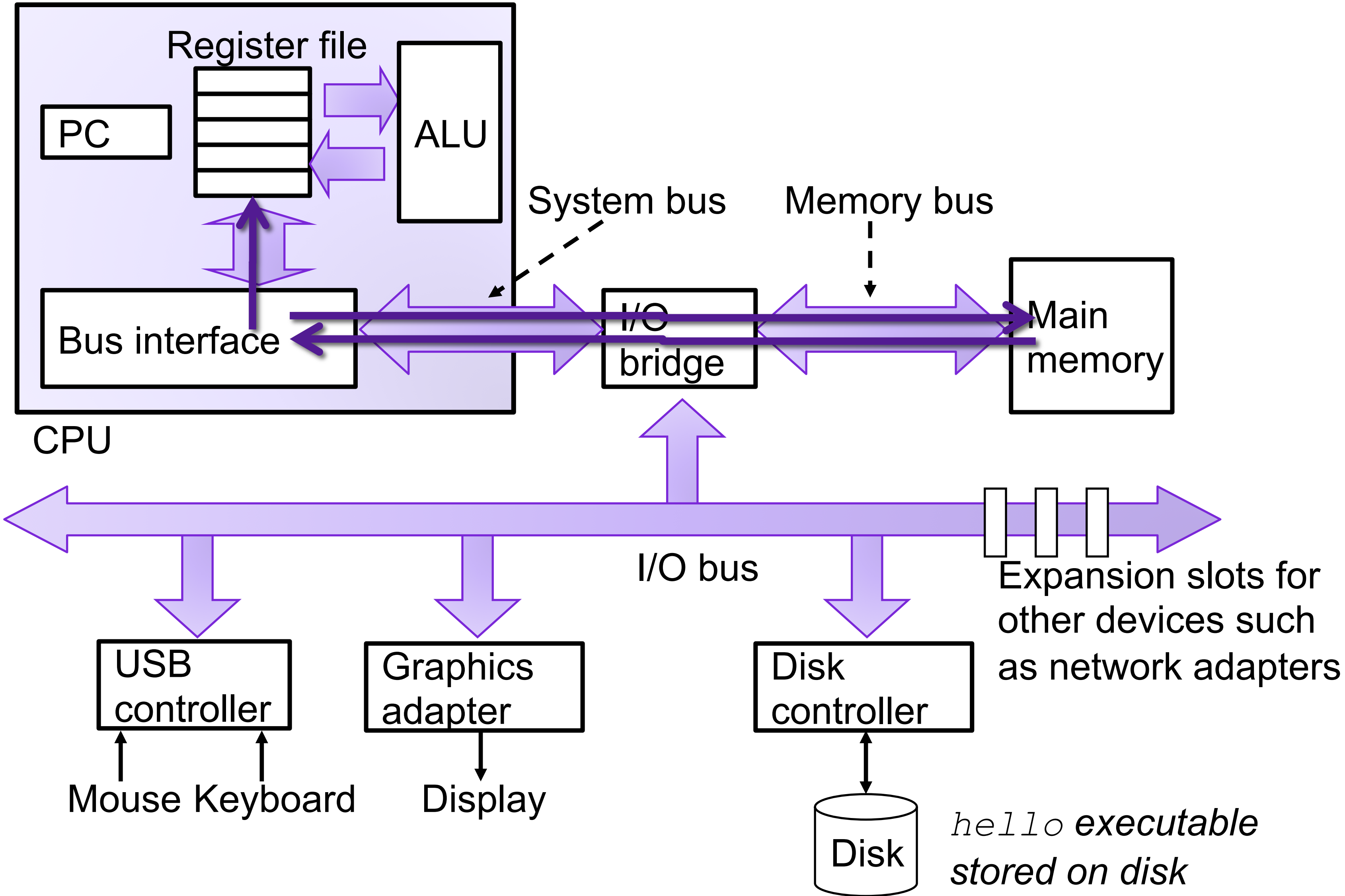
Library
(can hold many books)

Life without caches

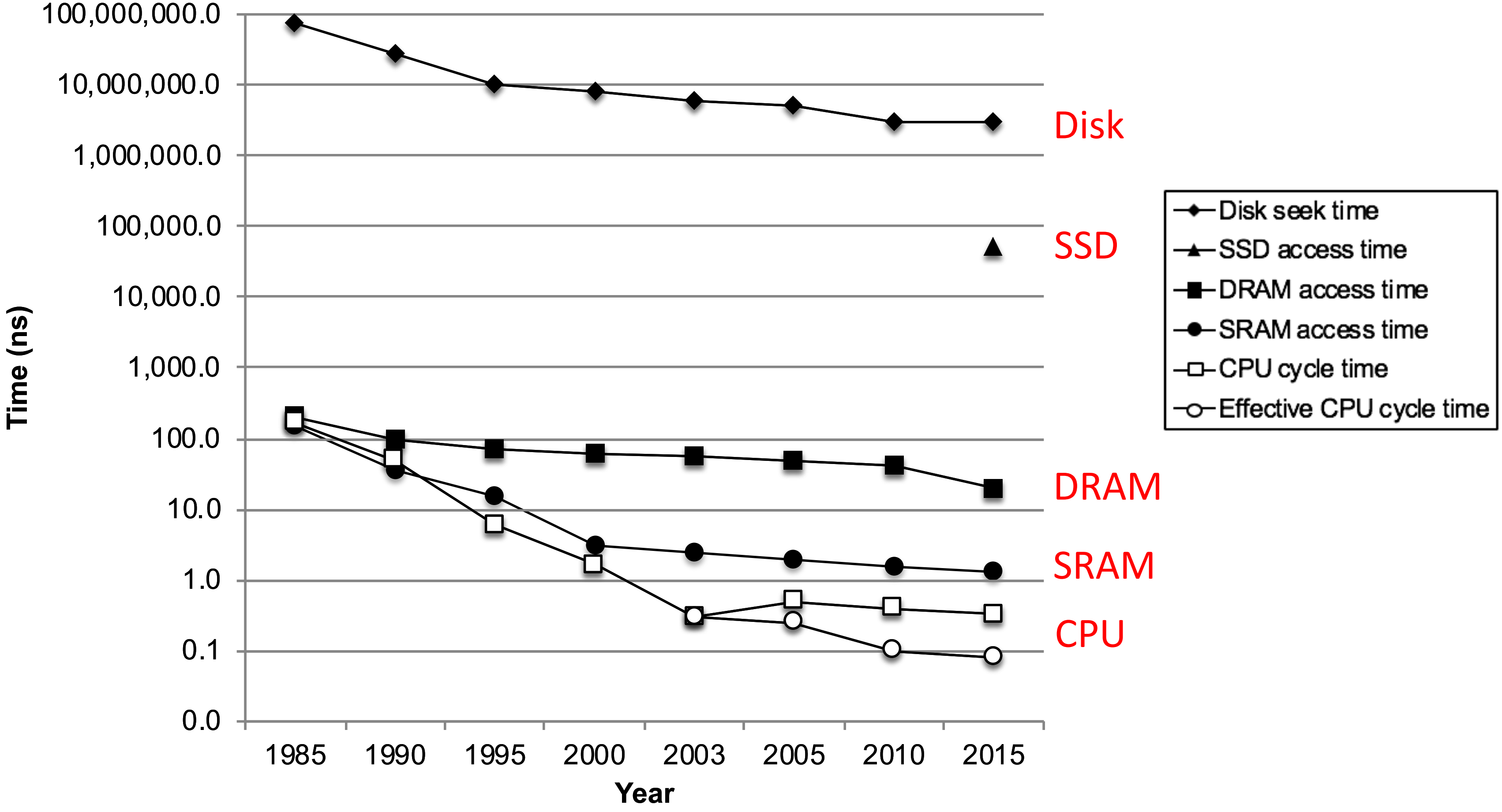


- Average latency to access a book: 40mins
- Average throughput (incl. reading time): 1.2 books/hr

A Computer System



The CPU-Memory Gap



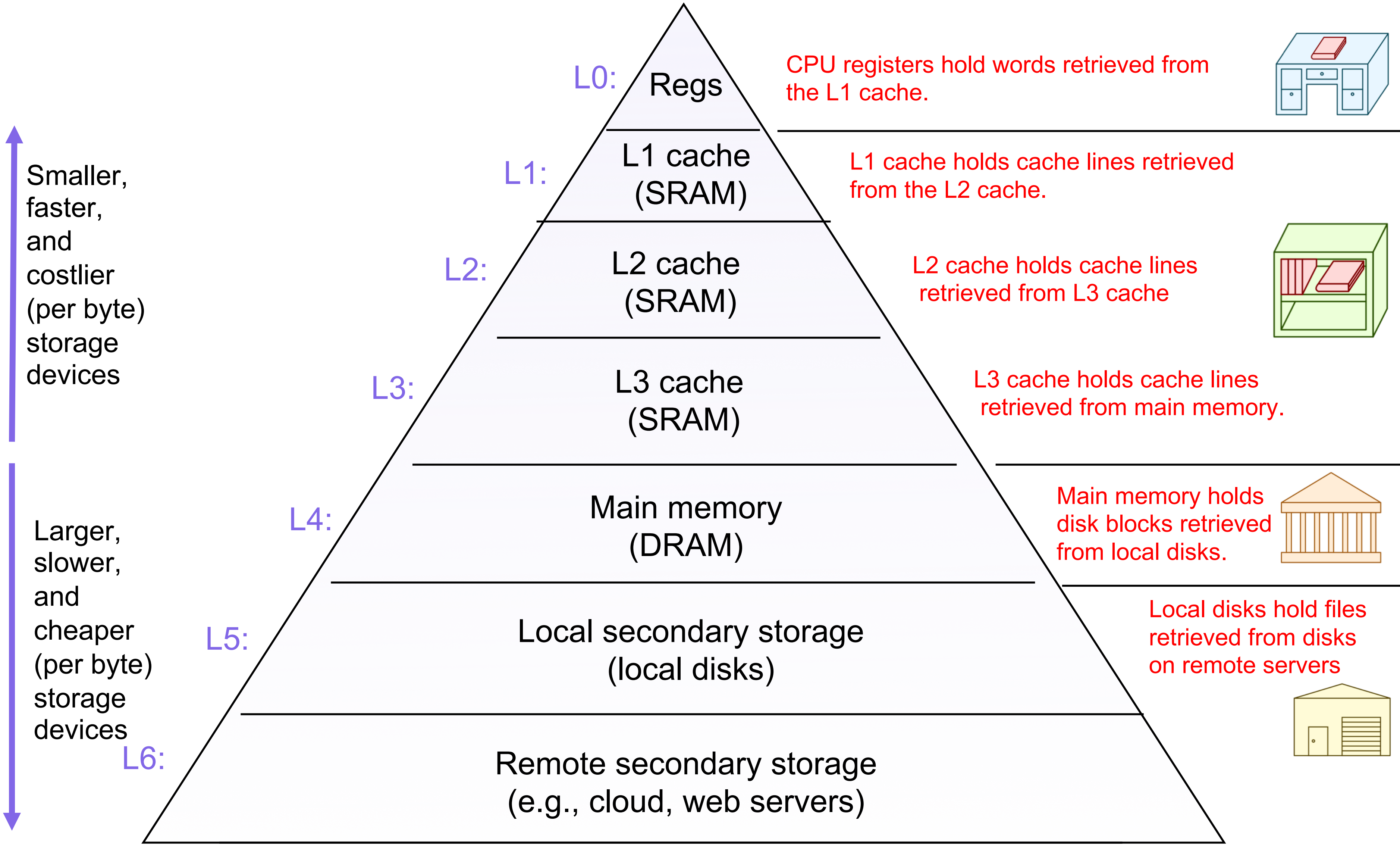
Caching—The Very Idea

- Keep “local” (spatially and temporally) memory values nearby in fast memory

- Modern systems have 3 or even 4 levels of caches

- Cache idea is widely used:
    - Disk controllers
    - Web
    - (Virtual memory: main memory is a “cache” for the disk)

Memory Hierarchy



Smaller, faster, and costlier (per byte) storage devices

Larger, slower, and cheaper (per byte) storage devices

L0: Regs

CPU registers hold words retrieved from the L1 cache.

L1: L1 cache (SRAM)

L1 cache holds cache lines retrieved from the L2 cache.

L2: L2 cache (SRAM)

L2 cache holds cache lines retrieved from L3 cache

L3: L3 cache (SRAM)

L3 cache holds cache lines retrieved from main memory.

L4: Main memory (DRAM)

Main memory holds disk blocks retrieved from local disks.

L5: Local secondary storage (local disks)

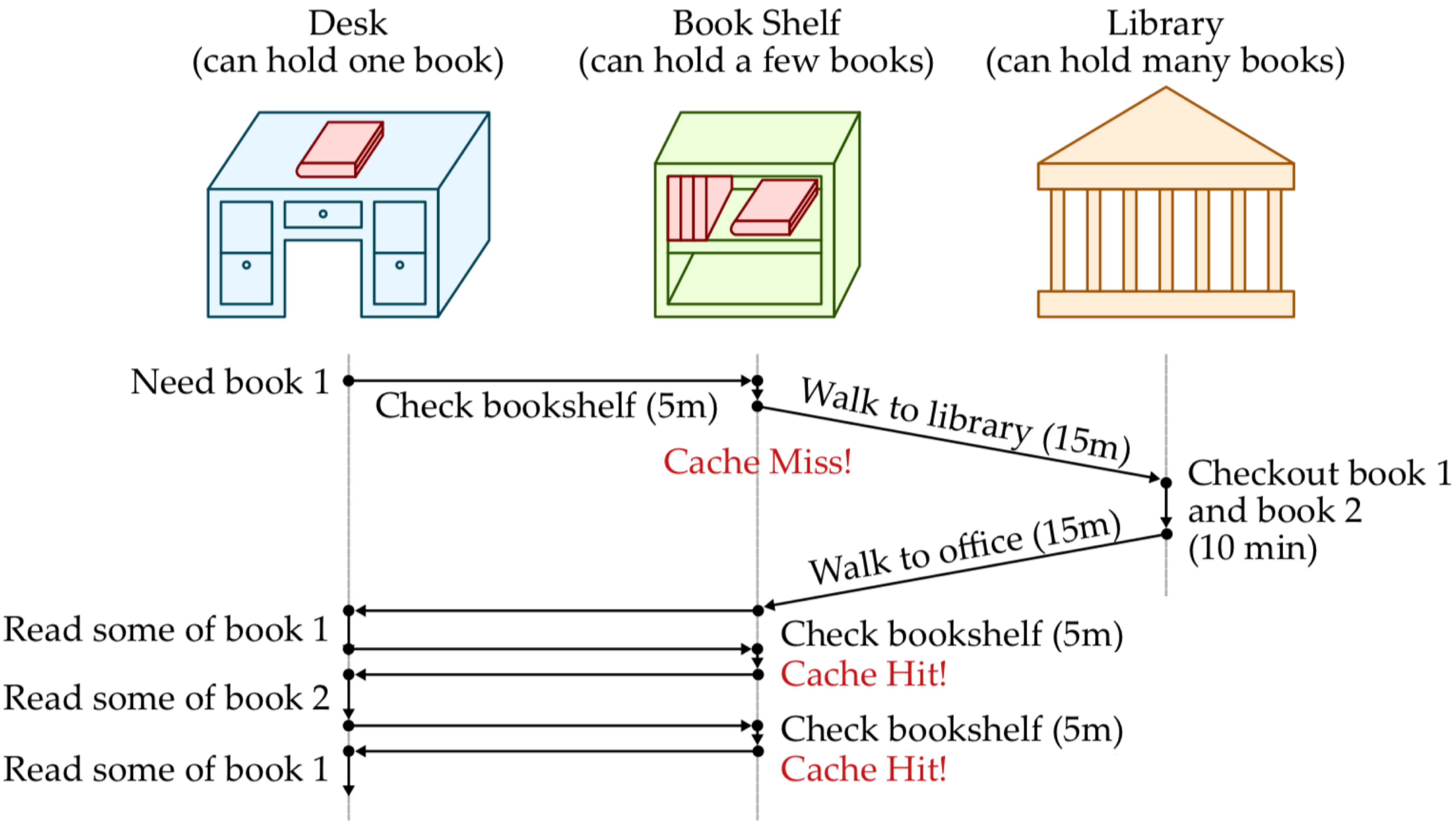
Local disks hold files retrieved from disks on remote servers

L6: Remote secondary storage (e.g., cloud, web servers)

Latency numbers every programmer should know (2019)

L1 cache reference	1 ns	
Branch mispredict	3 ns	
L2 cache reference	4 ns	
Main memory reference	100 ns	
memory 1MB sequential read	4,000 ns	4 μs
SSD random read	16,000 ns	16 μs
SSD 1MB sequential read	62,000 ns	62 μs
Disk random read	3,000,000 ns	3 ms
Disk 1MB sequential read	947,000 ns	< 1 ms
Round trip in Datacenter	500,000 ns	500 μs
Round trip CA<->Europe	150,000,000 ns	150 ms

Life with caching



- Average latency to access a book: <20mins
- Average throughput (incl. reading time): ~2 books/hr

Caching—The Vocabulary

- **Size:** the total number of bytes that can be stored in the cache

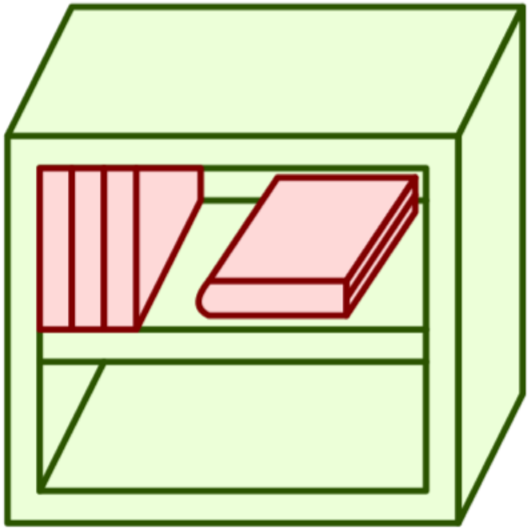
- **Cache Hit:** the desired value is in the cache and returned quickly
- **Cache Miss:** the desired value is not in the cache and must be fetched from a more distant cache (or ultimately from main memory)

- **Miss rate:** the fraction of accesses that are misses

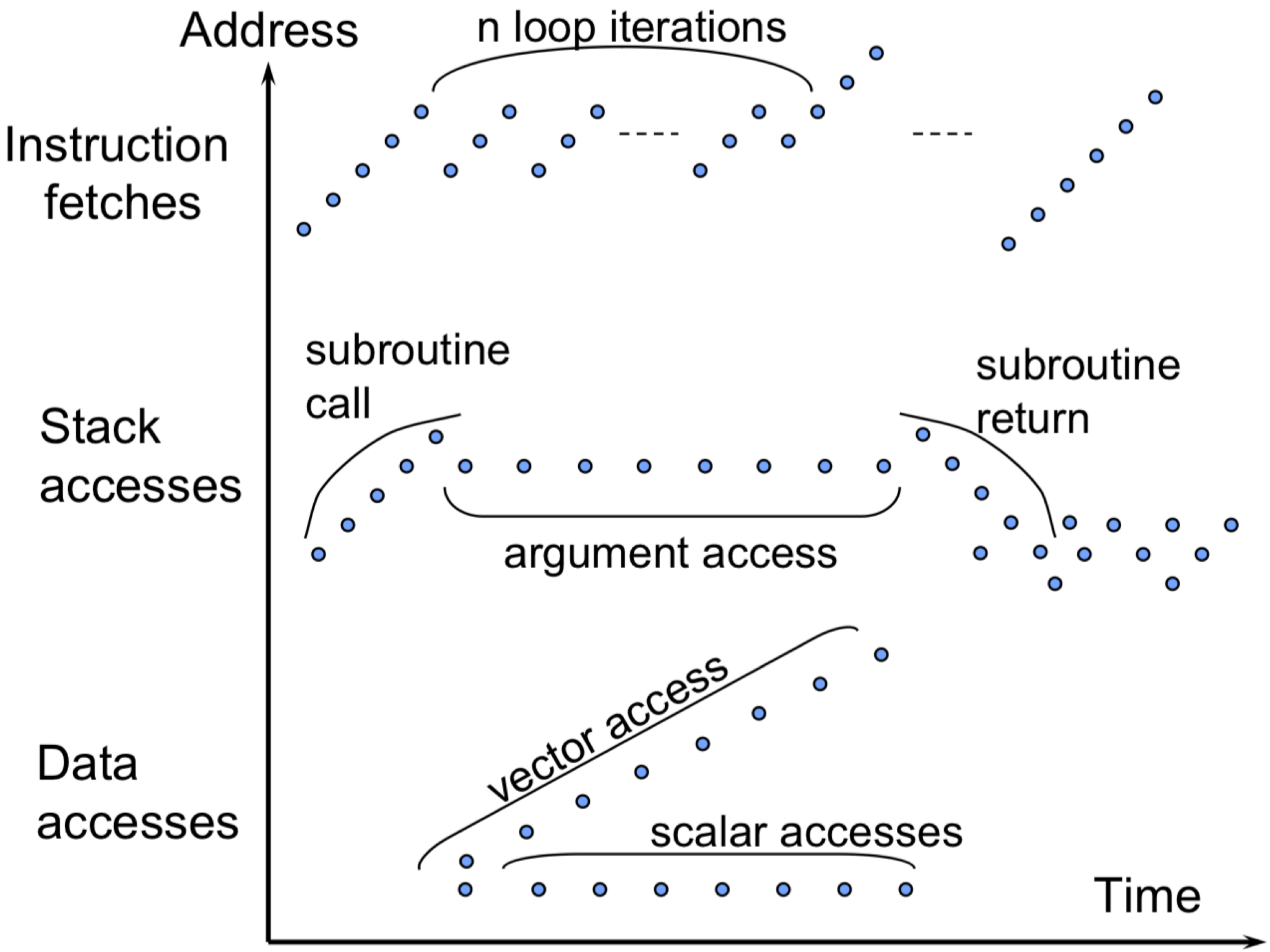
- **Hit time:** the time to process a hit
- **Miss penalty:** the *additional* time to process a miss

- **Average access time:** hit-time + miss-rate * miss-penalty

Question: how do we decide which books to put on the bookshelf?



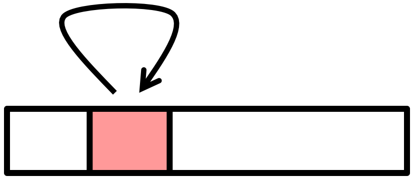
Example Access Patterns



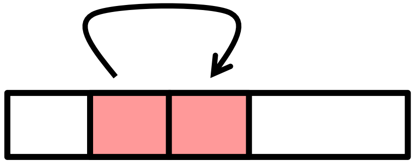
Principle of Locality

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



Locality Example

- Which of the following functions is better in terms of locality with respect to array src?

```
void copyij(int src[2048][2048],
            int dst[2048][2048])
{
  int i,j;
  for (i = 0; i < 2048; i++)
    for (j = 0; j < 2048; j++)
      dst[i][j] = src[i][j];
}
```

```
void copyji(int src[2048][2048],
            int dst[2048][2048])
{
  int i,j;
  for (j = 0; j < 2048; j++)
    for (i = 0; i < 2048; i++)
      dst[i][j] = src[i][j];
}
```

4.3ms

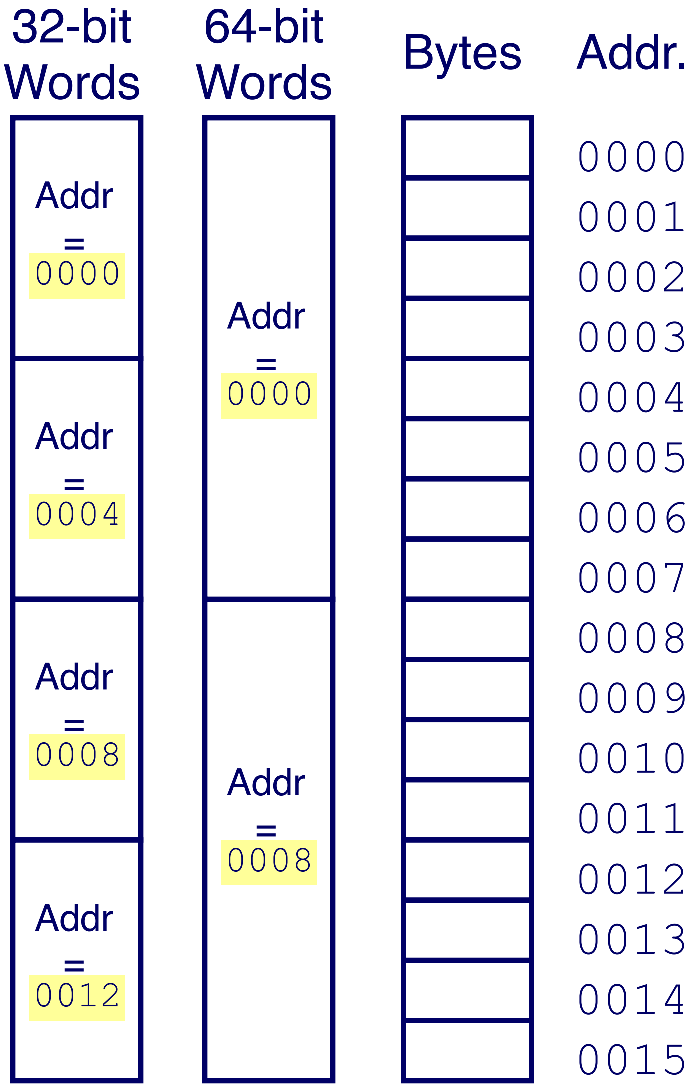
81.8ms

2.0 GHz Intel Core i7 Haswell

CACHE ORGANIZATION

Word-oriented Memory Organization

- Addresses Specify Byte Locations
  - Address of first byte in word
  - Addresses of successive words differ by m=4 (32-bit) or m=8 (64-bit)
- There are (up to) $M = 2^m$ unique addresses in memory



32-bit Words

Addr = 0000

Addr = 0004

Addr = 0008

Addr = 0012

64-bit Words

Addr = 0000

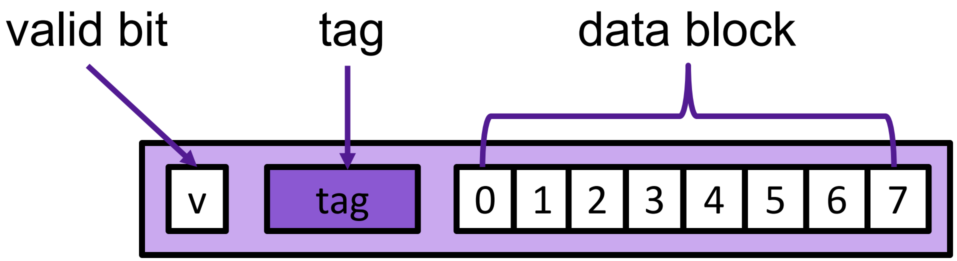
Addr = 0008

Bytes

Addr.

0000
0001
0002
0003
0004
0005
0006
0007
0008
0009
0010
0011
0012
0013
0014
0015

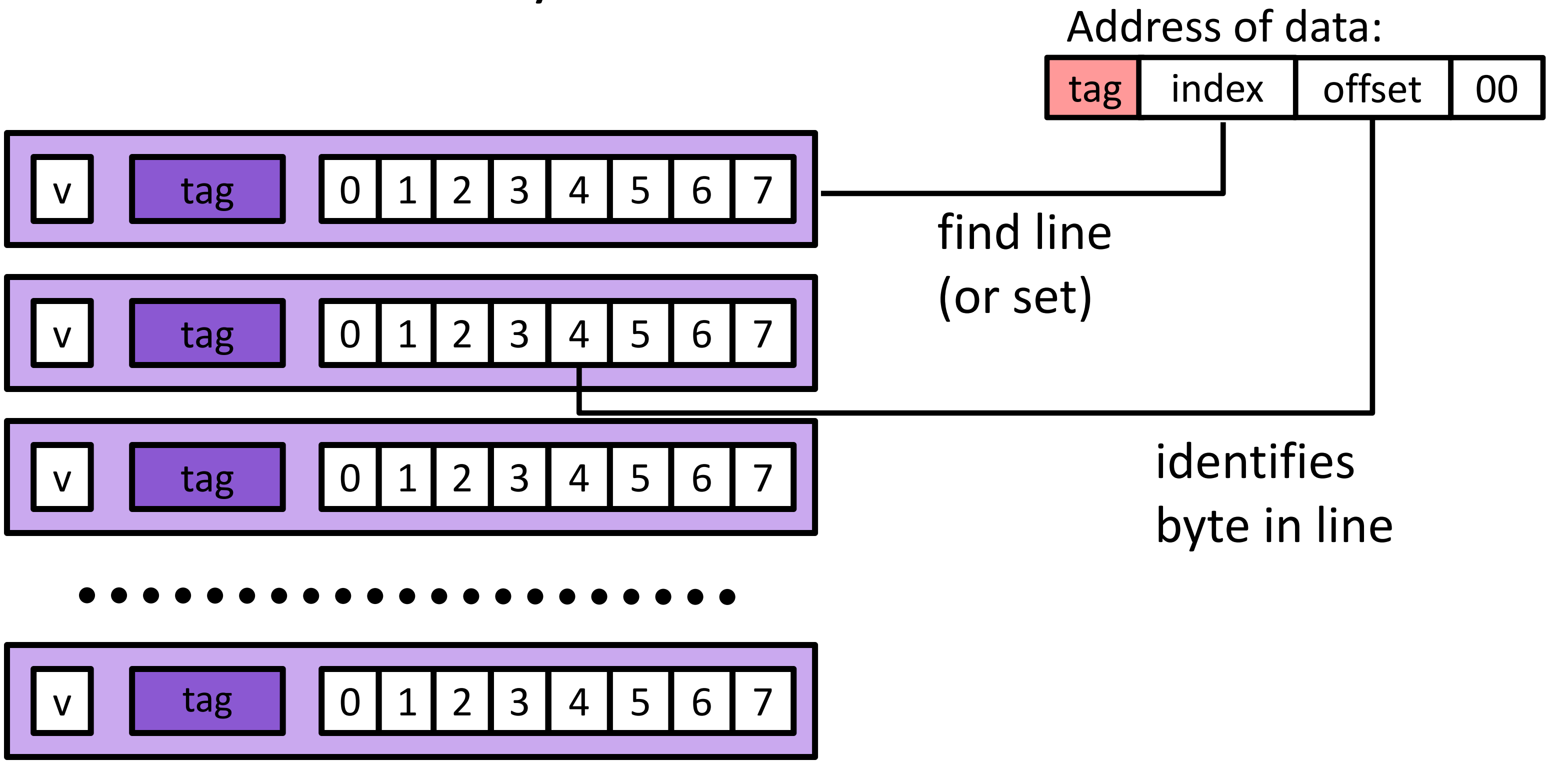
Cache Lines



- **data block:** cached data
- **tag:** uniquely identifies which data is stored in the cache line
- **valid bit:** indicates whether or not the line contains meaningful information

Direct-mapped Cache

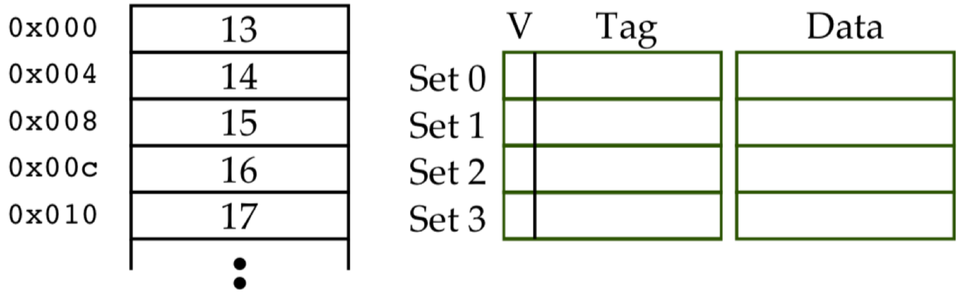
Assume: cache block size 8 bytes



Exercise: Direct-Mapped Cache

Dynamic Transaction Stream

```
rd 0x000
rd 0x004
rd 0x010
rd 0x000
rd 0x004
```



0x000	13
0x004	14
0x008	15
0x00c	16
0x010	17
	⋮

	V	Tag	Data
Set 0			
Set 1			
Set 2			
Set 3			

				Set			
	tag	**idx**	**h/m**	**0**	**1**	**2**	**3**
rd 0x000							
rd 0x004							
rd 0x010							
rd 0x000							
rd 0x004							
rd 0x020							

How well does this take advantage of spacial locality?
How well does this take advantage of temporal locality?