The Memory Hierarchy

15-213/18-213/15-513: Introduction to Computer Systems
10th Lecture, June 16, 2021

Today

- **The memory abstraction**
- RAM : main memory building block
- Locality of reference
- The memory hierarchy
- Storage technologies and trends

Writing & Reading Memory

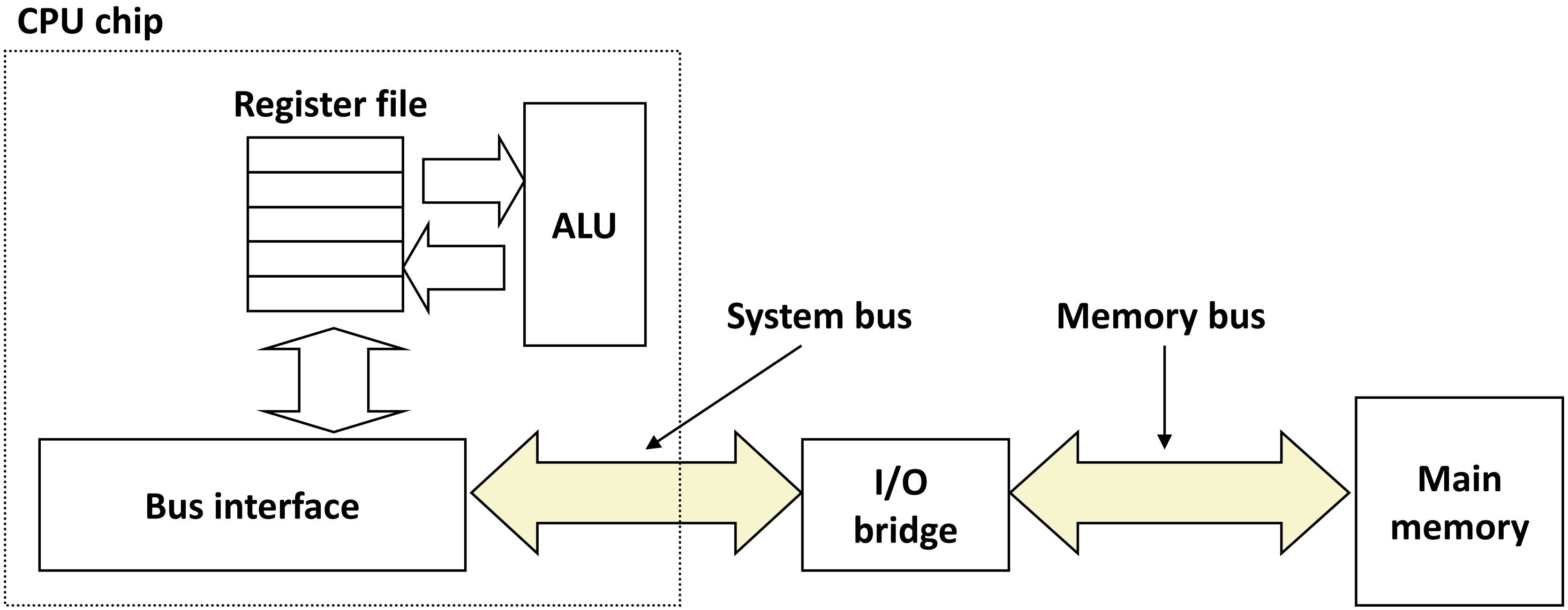
- **Write**
  - Transfer data from memory to CPU
    ```
    movq %rax, 8(%rsp)
    ```
  - “Store” operation

- **Read**
  - Transfer data from CPU to memory
    ```
    movq 8(%rsp), %rax
    ```
  - “Load” operation

From 5th lecture

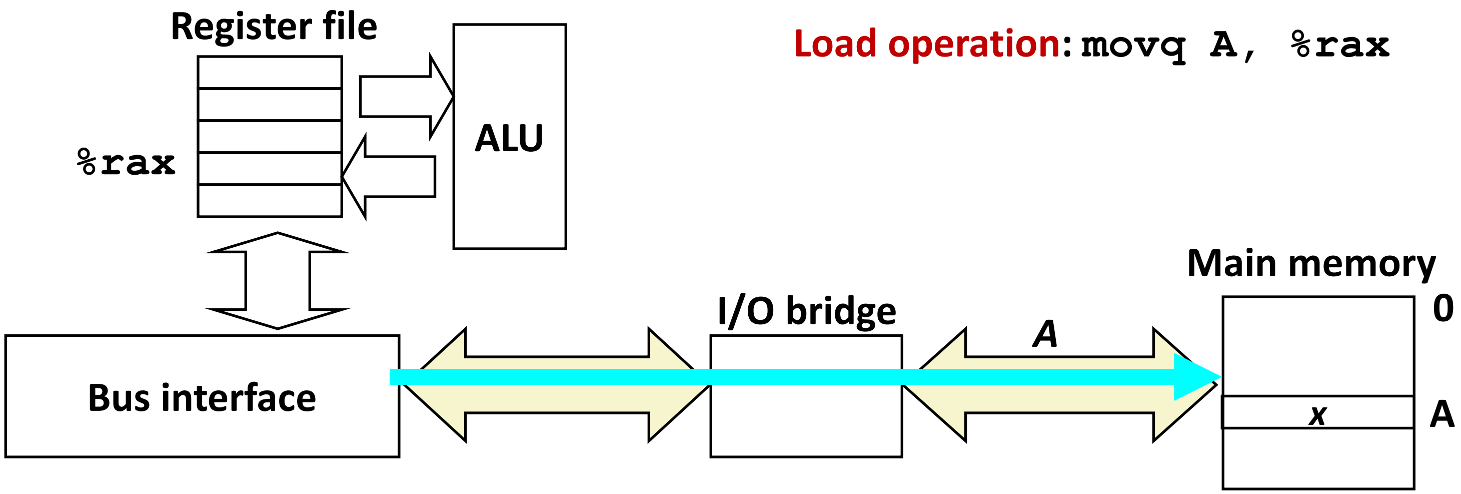
Traditional Bus Structure Connecting CPU and Memory

- 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)

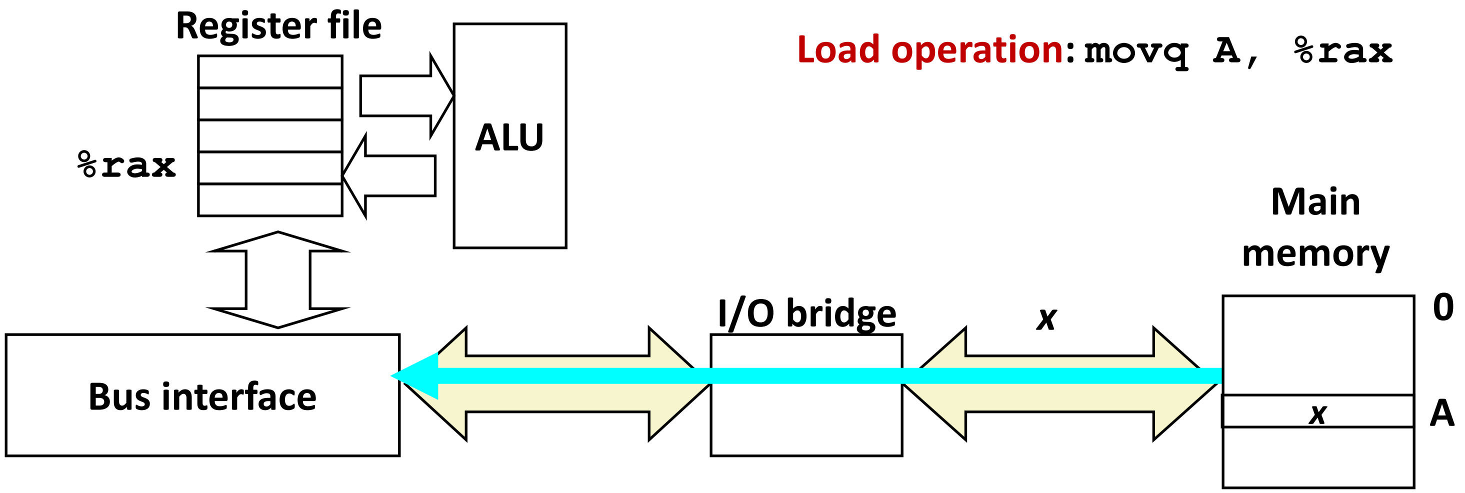
- CPU places address A on the memory bus.



Load operation: `movq A, %rax`

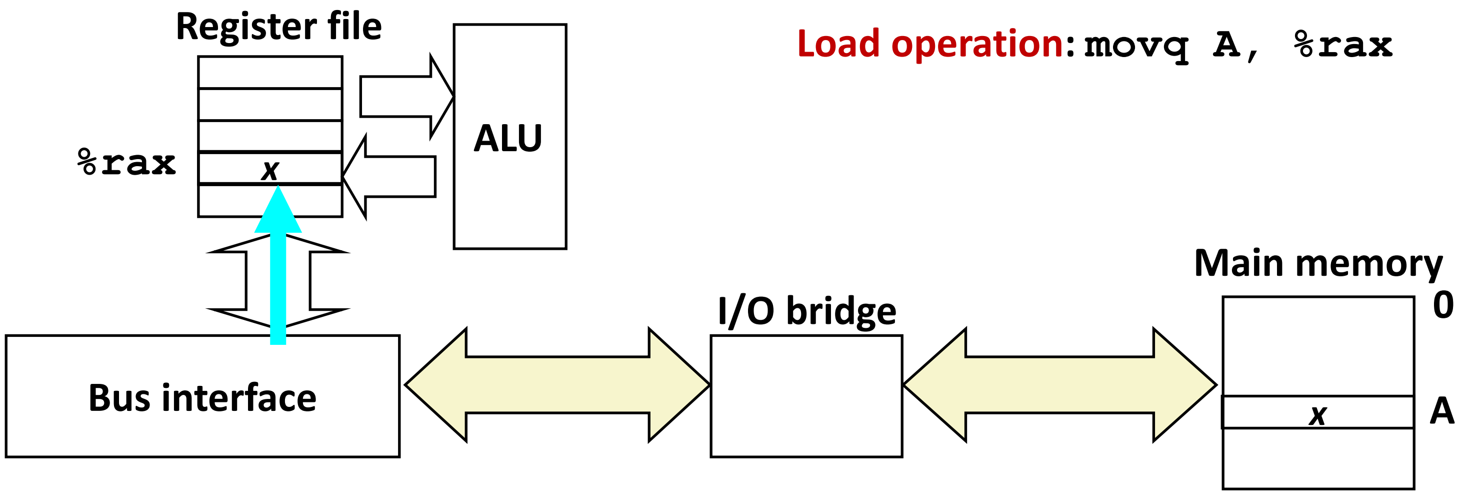
Memory Read Transaction (2)

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



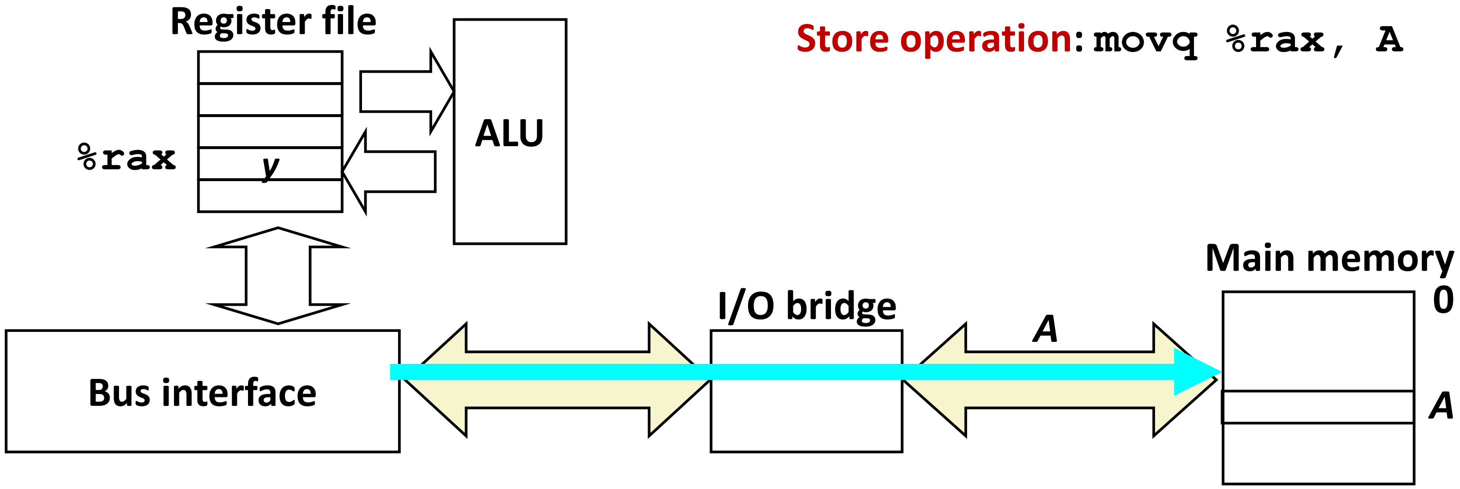
Memory Read Transaction (3)

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



Memory Write Transaction (1)

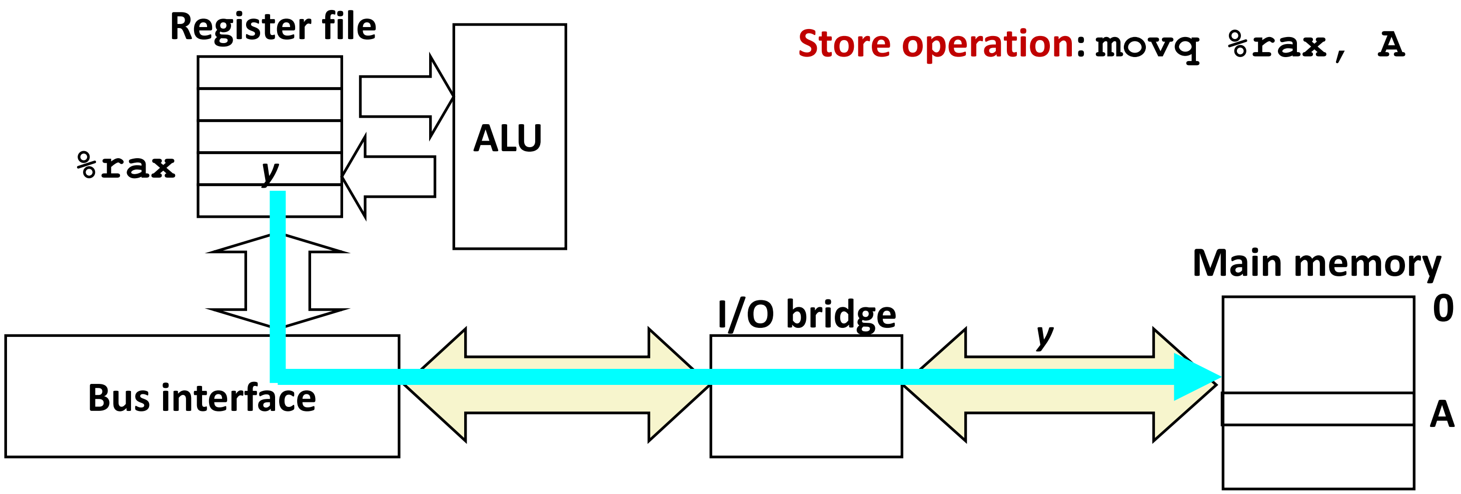
- CPU places address A on bus. Main memory reads it and waits for the corresponding data word to arrive.



Store operation: `movq %rax, A`

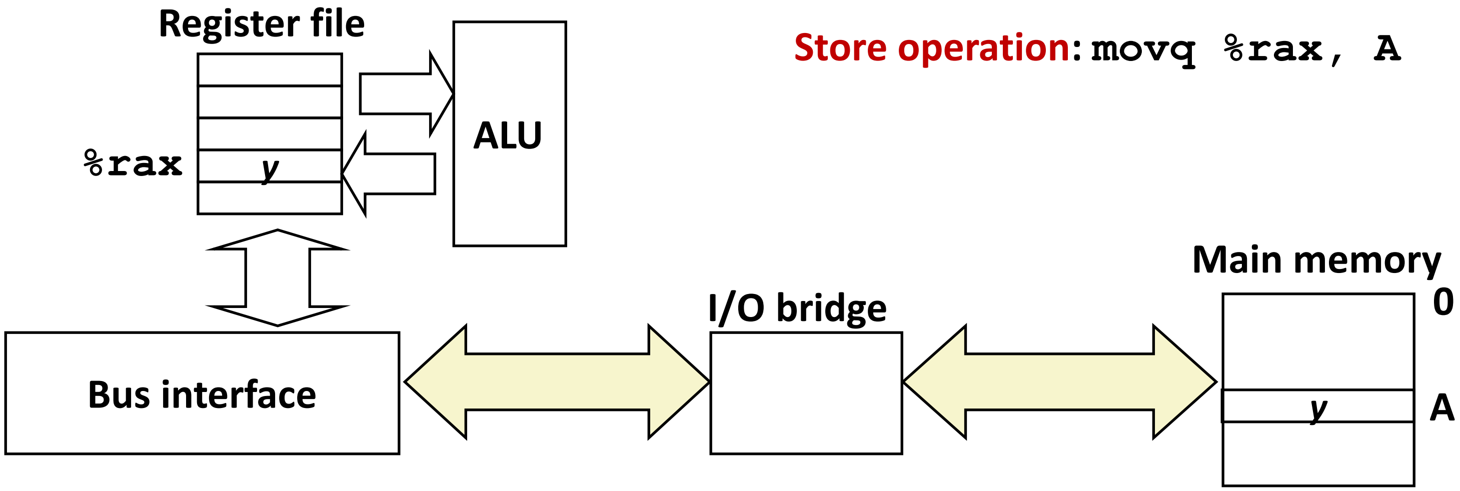
Memory Write Transaction (2)

- 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.



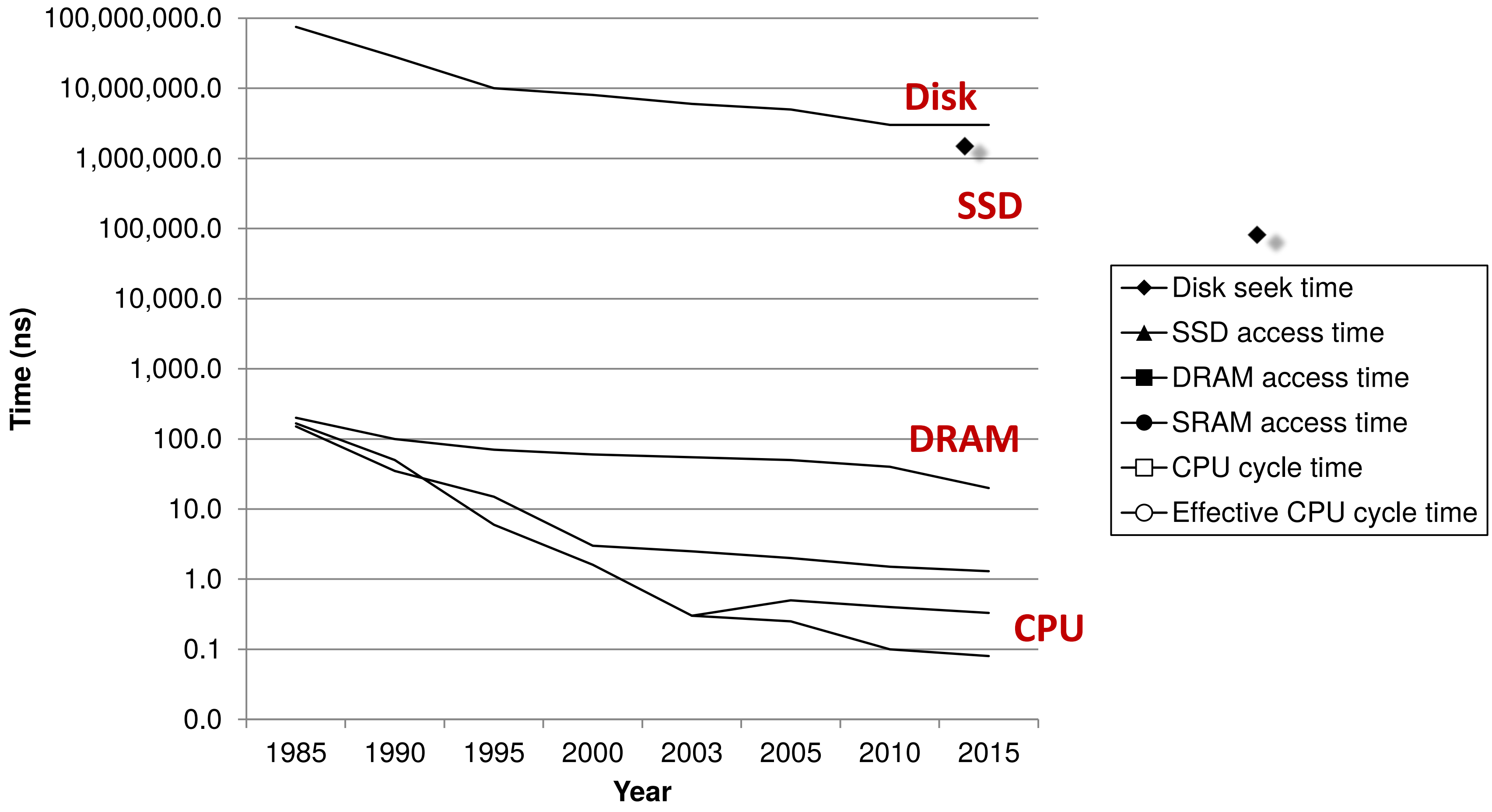
Store operation: `movq %rax, A`

Today

- The memory Abstraction
- DRAM : main memory building block
- **Locality of reference**
- The memory hierarchy
- Storage technologies and trends

The CPU-Memory Gap

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

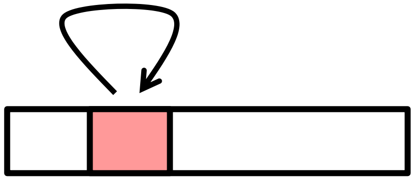


Locality to the Rescue!

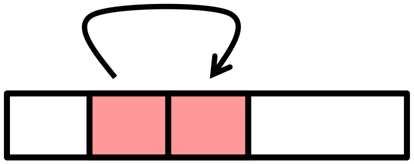
The key to bridging this CPU-Memory gap is a fundamental property of computer programs known as locality.

Locality

- **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

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

	Spatial or Temporal Locality?
Data references	
Reference array elements in succession (stride-1 reference pattern).	**spatial**
Reference variable **`sum`** each iteration.	**temporal**
Instruction references	
Reference instructions in sequence.	**spatial**
Cycle through loop repeatedly.	**temporal**

Qualitative Estimates of Locality

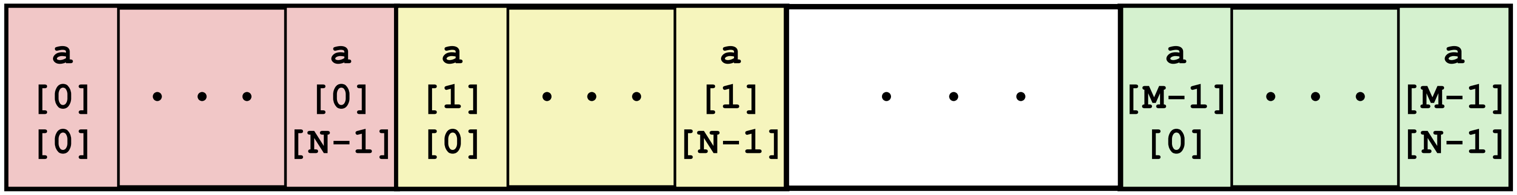
- **Claim:** Being able to look at code and get a qualitative sense of its locality is a key skill for a professional programmer.
- **Question:** Does this function have good locality with respect to array `a`?

```
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;
}
```

Hint: array layout is row-major order

Answer: yes



Locality Example

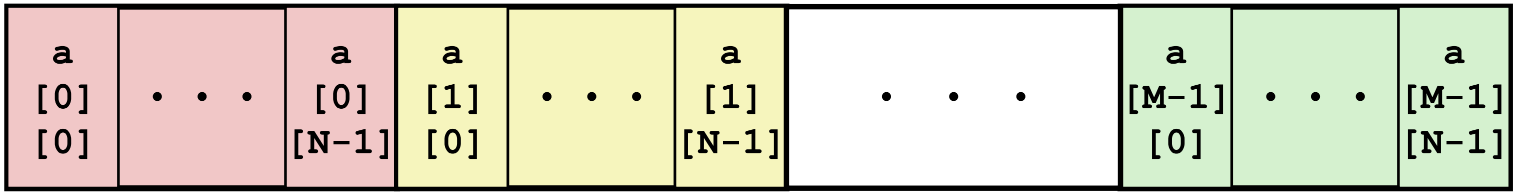
- **Question:** Does this function have good locality with respect to array a?

```
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;
}
```

Answer: no, unless…

M is very small



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;
}
```

Answer: make j the inner loop

Today

- The memory abstraction
- DRAM : main memory building block
- Storage technologies and trends
- Locality of reference
- **The memory hierarchy**

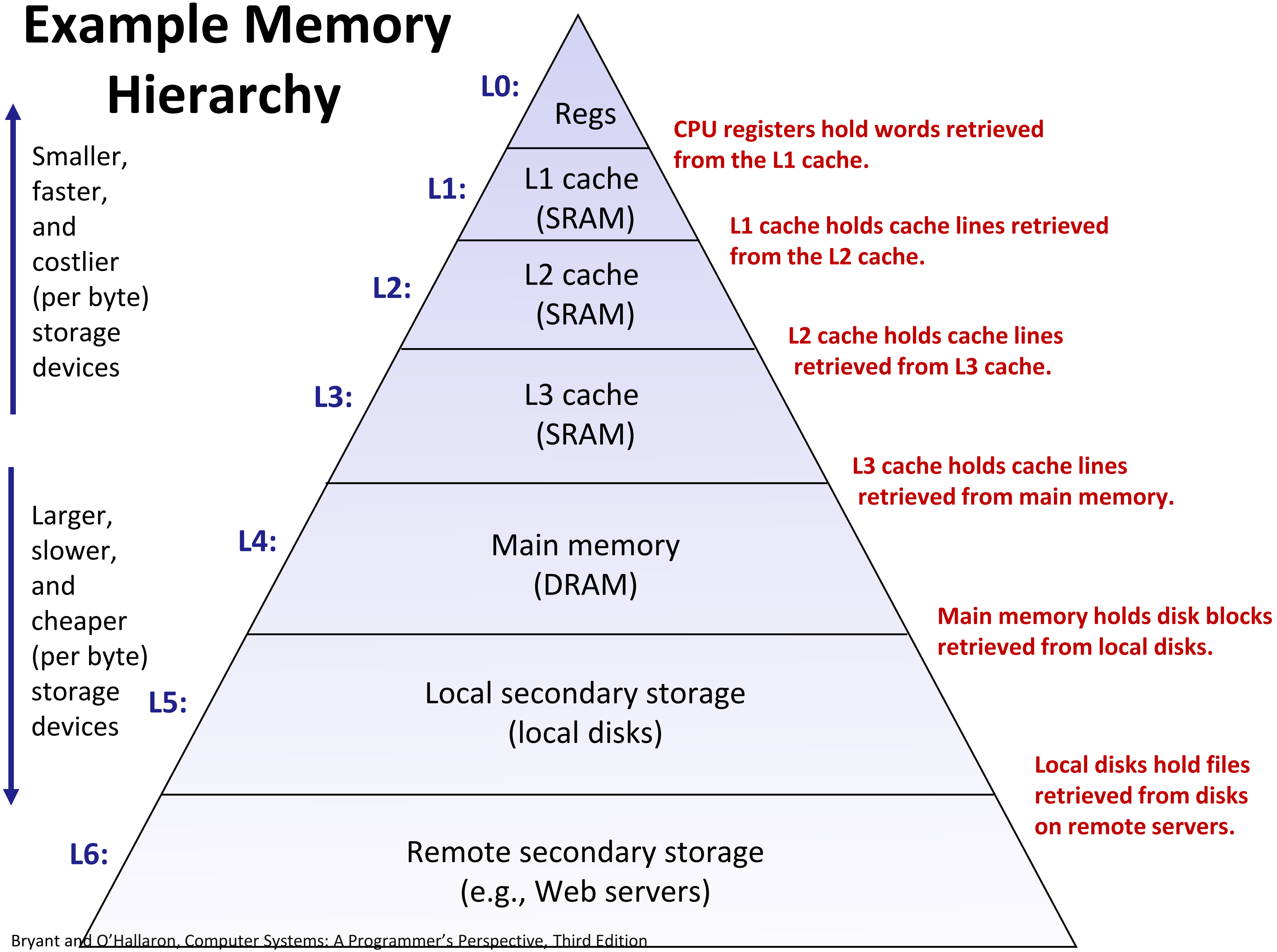
Memory Hierarchies

- **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



L0: Regs

L1: L1 cache (SRAM)

L2: L2 cache (SRAM)

L3: L3 cache (SRAM)

L4: Main memory (DRAM)

L5: Local secondary storage (local disks)

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

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

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

CPU registers hold words retrieved from the L1 cache.

L1 cache holds cache lines retrieved from the L2 cache.

L2 cache holds cache lines retrieved from L3 cache.

L3 cache holds cache lines retrieved from main memory.

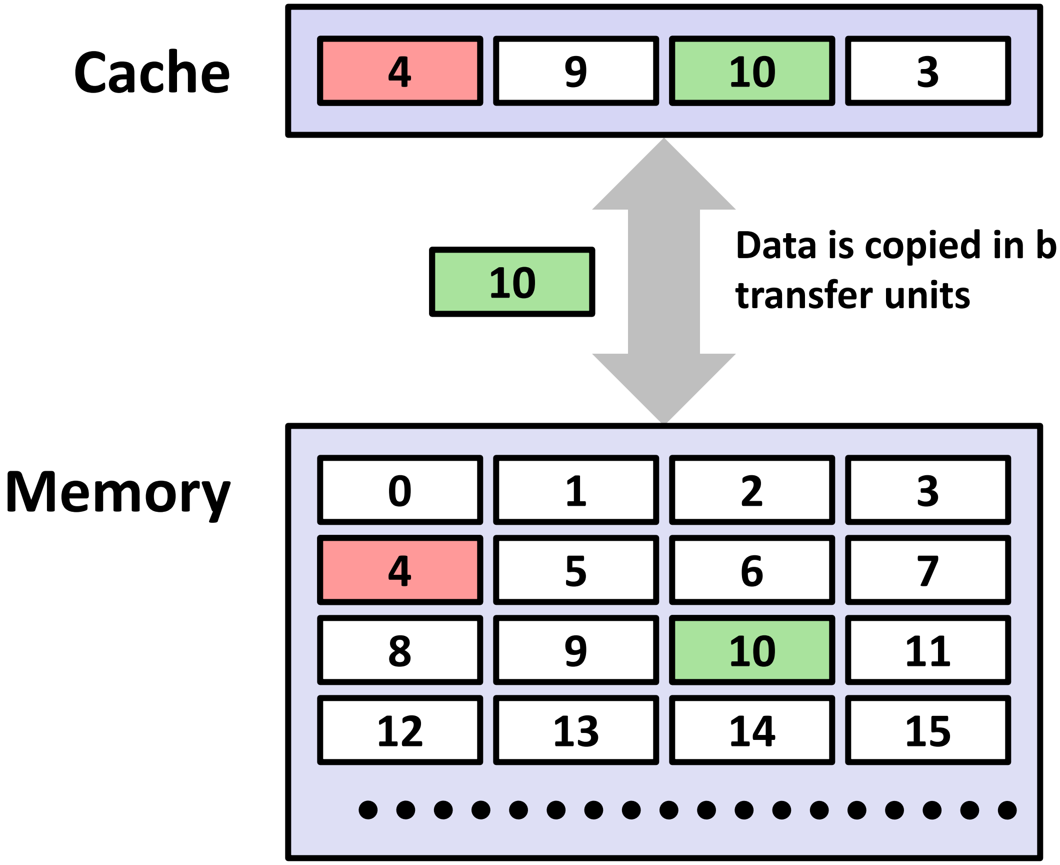
Main memory holds disk blocks retrieved from local disks.

Local disks hold files retrieved from disks on remote servers.

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.
- ***Big Idea (Ideal):*** **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.**

General Cache Concepts

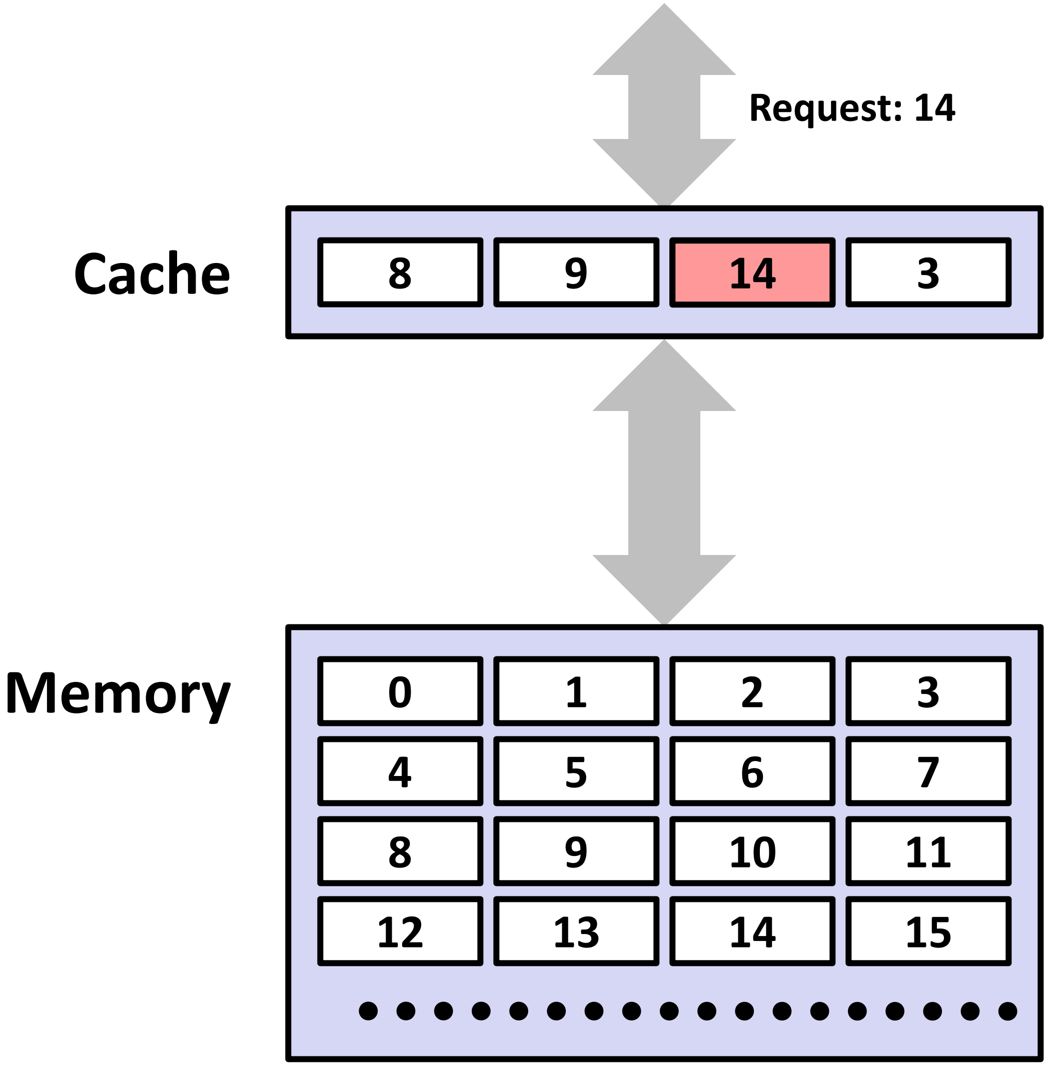


Smaller, faster, more expensive memory caches a subset of the blocks

Data is copied in block-sized transfer units

Larger, slower, cheaper memory viewed as partitioned into "blocks"

General Cache Concepts: Hit



8	9	14	3

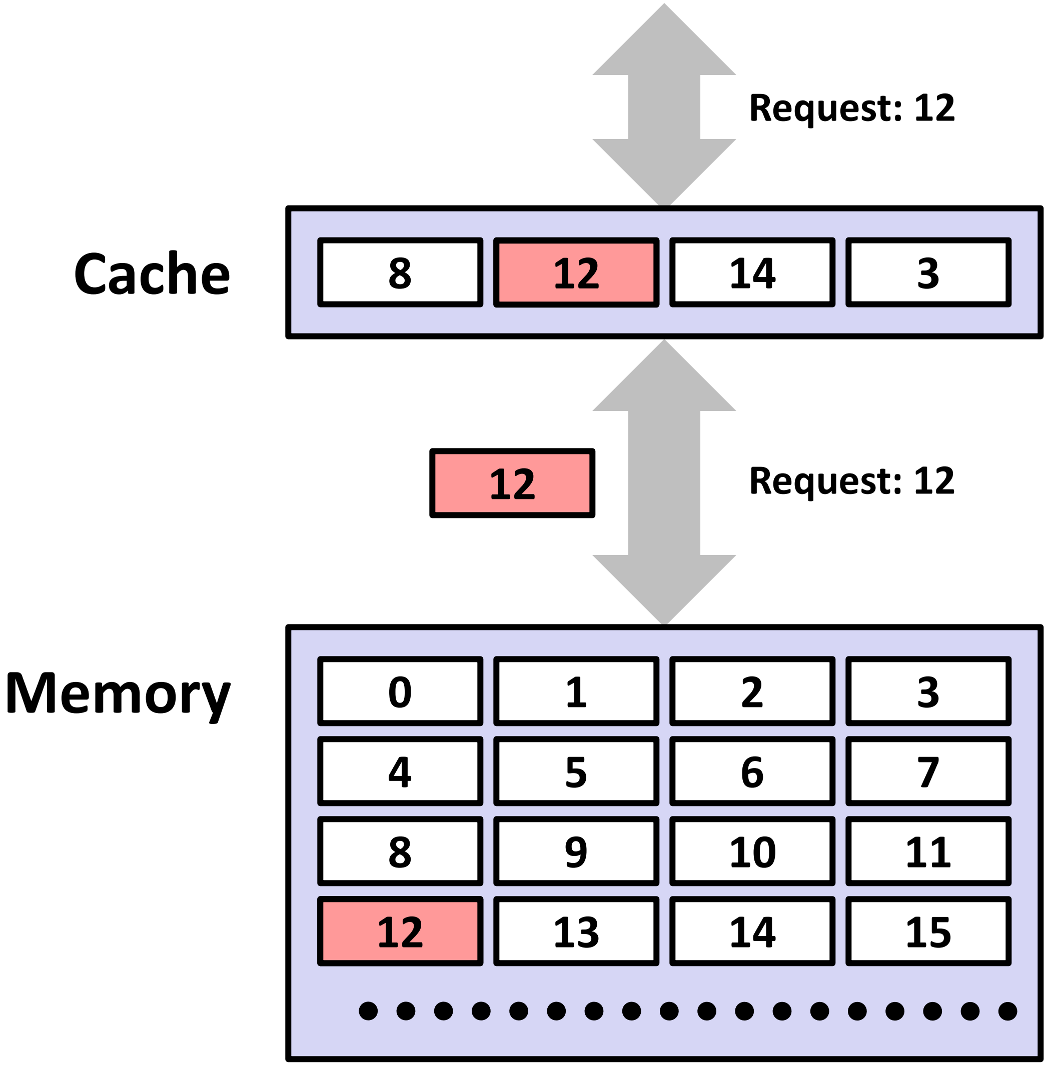
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

• • • • • • • • • • • • • • • • • •

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 (victim)

General Caching Concepts: 3 Types of Cache Misses

- **Cold (compulsory) miss**
  - Cold misses occur because the cache starts empty and this is the first reference to the block.
- **Capacity miss**
  - Occurs when the set of active cache blocks (working set) is larger than the cache.
- **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.

Examples of Caching in the Mem. Hierarchy

Cache Type	What is Cached?	Where is it Cached?	Latency (cycles)	Managed By
Registers	4-8 byte words	CPU core	0	Compiler
TLB	Address translations	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	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