+

Memory Subsystem

+ 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.

- **RAM comes in two varieties:**
  - SRAM (Static RAM)
  - DRAM (Dynamic RAM)

+ 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

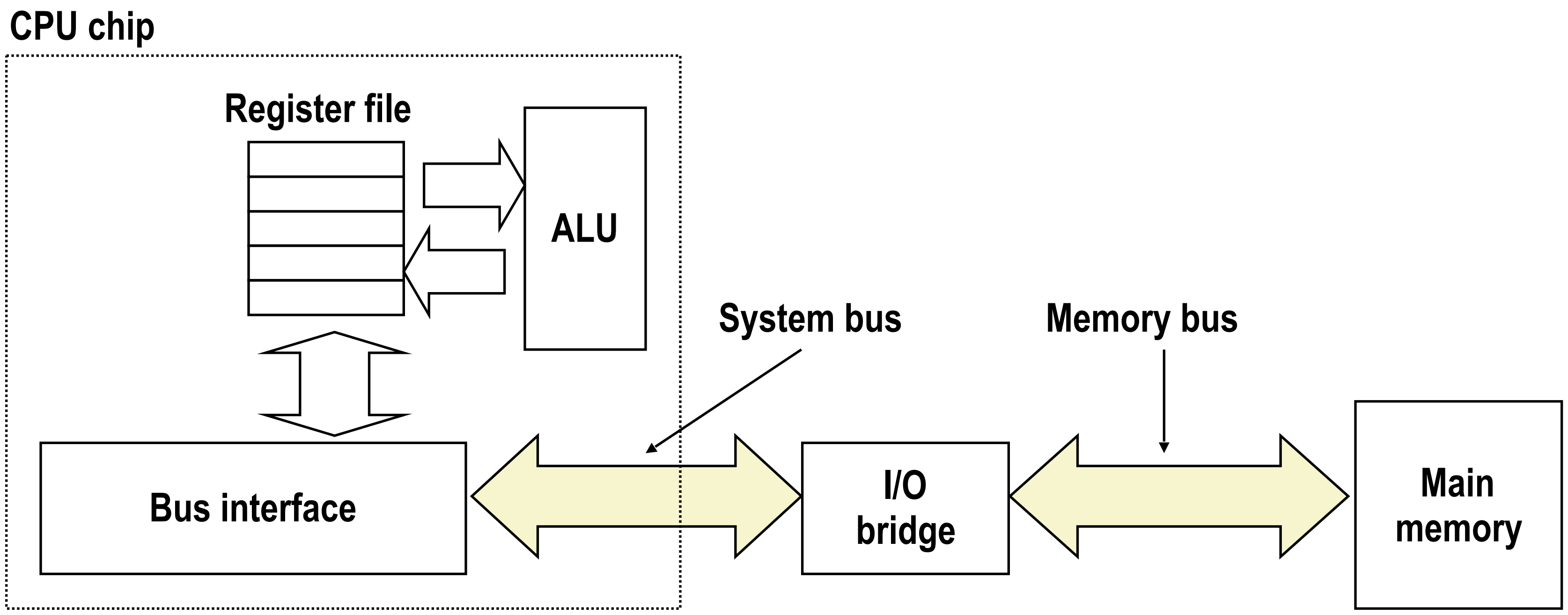
Nonvolatile Memories

- **DRAM and SRAM are volatile memories**
  - Lose information if powered off.
- **Nonvolatile memories retain value even if powered off**
  - Read-only memory (**ROM**): programmed during production
  - Programmable ROM (**PROM**): can be programmed once
  - Eraseable PROM (**EPROM**): can be bulk erased (UV, X-Ray)
  - Electrically eraseable PROM (**EEPROM**): electronic erase capability
  - Flash memory: EEPROMs with *partial* erase capability

- **Uses for Nonvolatile Memories**
  - Firmware programs stored in a ROM (**BIOS**, controllers for disks, network cards, graphics accelerators, security subsystems,…)
  - Solid state disks aka **SSD** (replace rotating disks in thumb drives, smart phones, mp3 players, tablets, laptops,…)

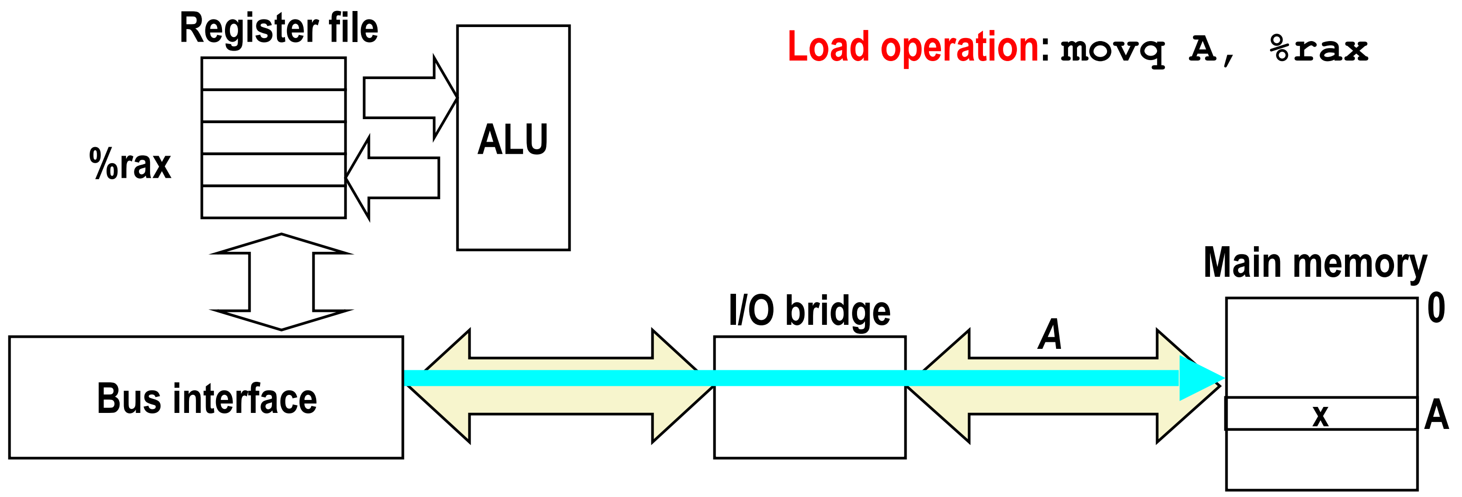
+ Traditional Bus Structure

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



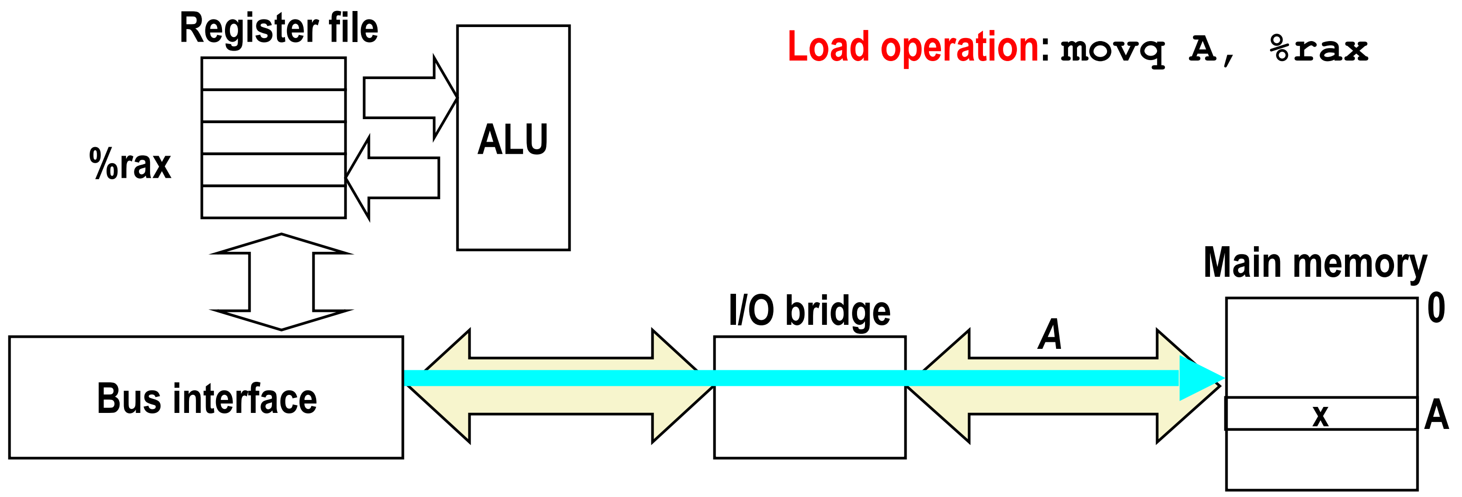
Memory Read Transaction (1)

- CPU places address A on the memory bus.



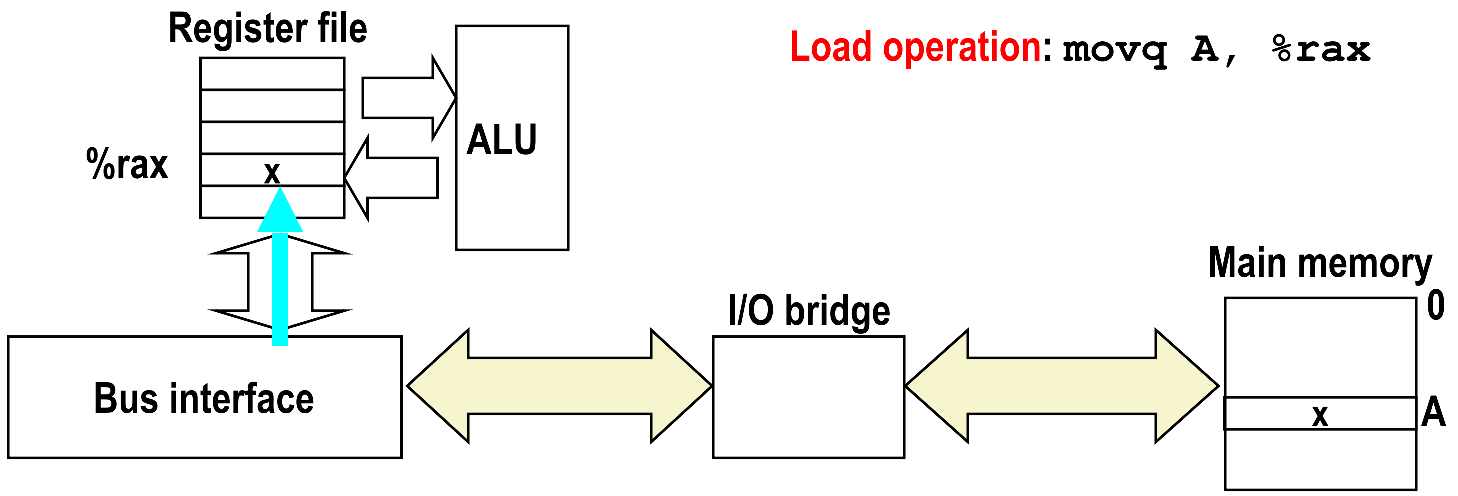
Memory Read Transaction (2)

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



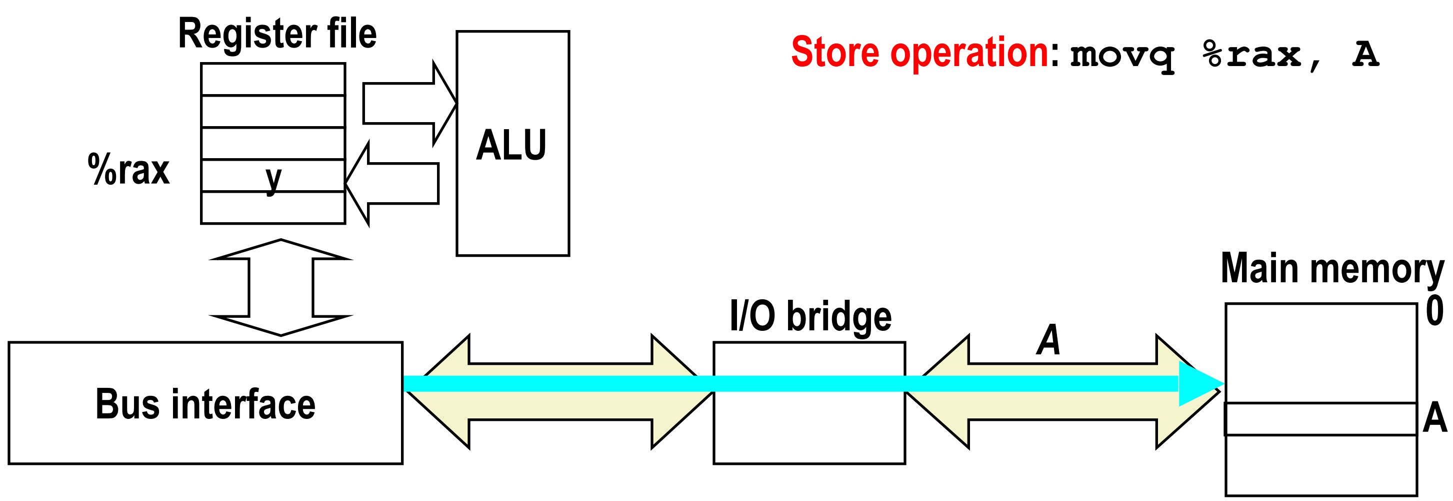
Memory Read Transaction (3)

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



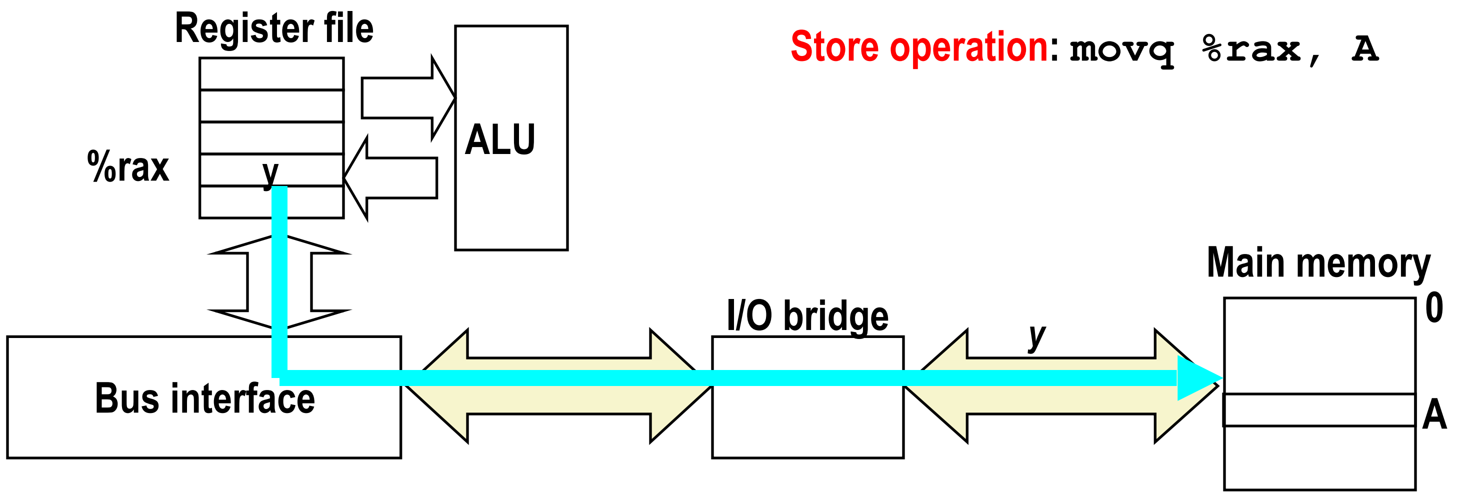
+ Memory Write Transaction (1)

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



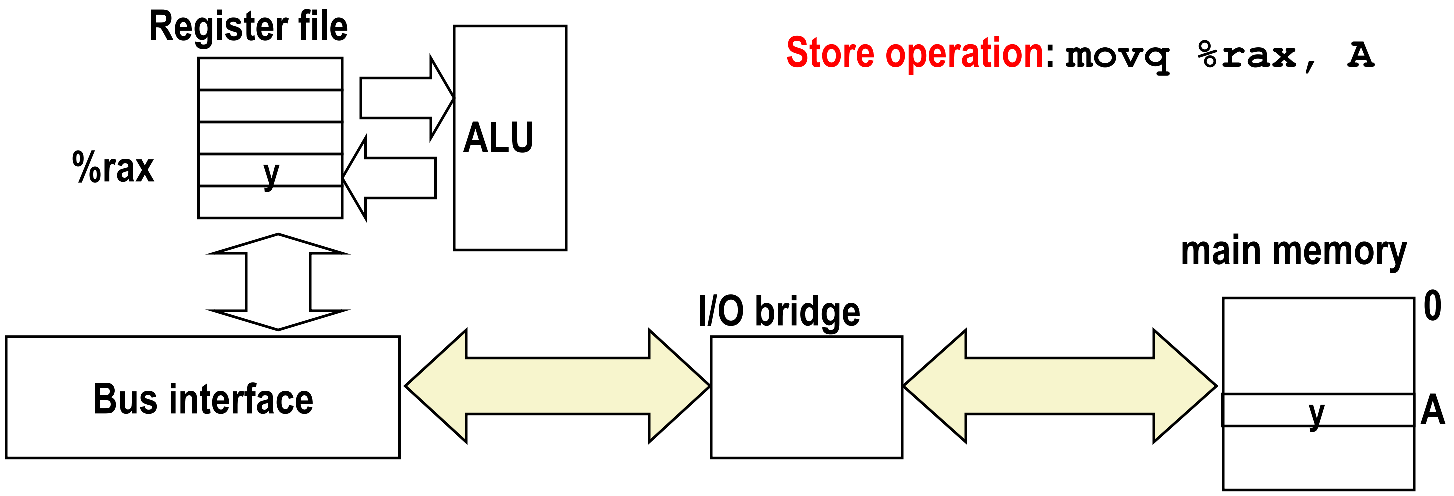
Memory Write Transaction (2)

- CPU places data word y on the bus.



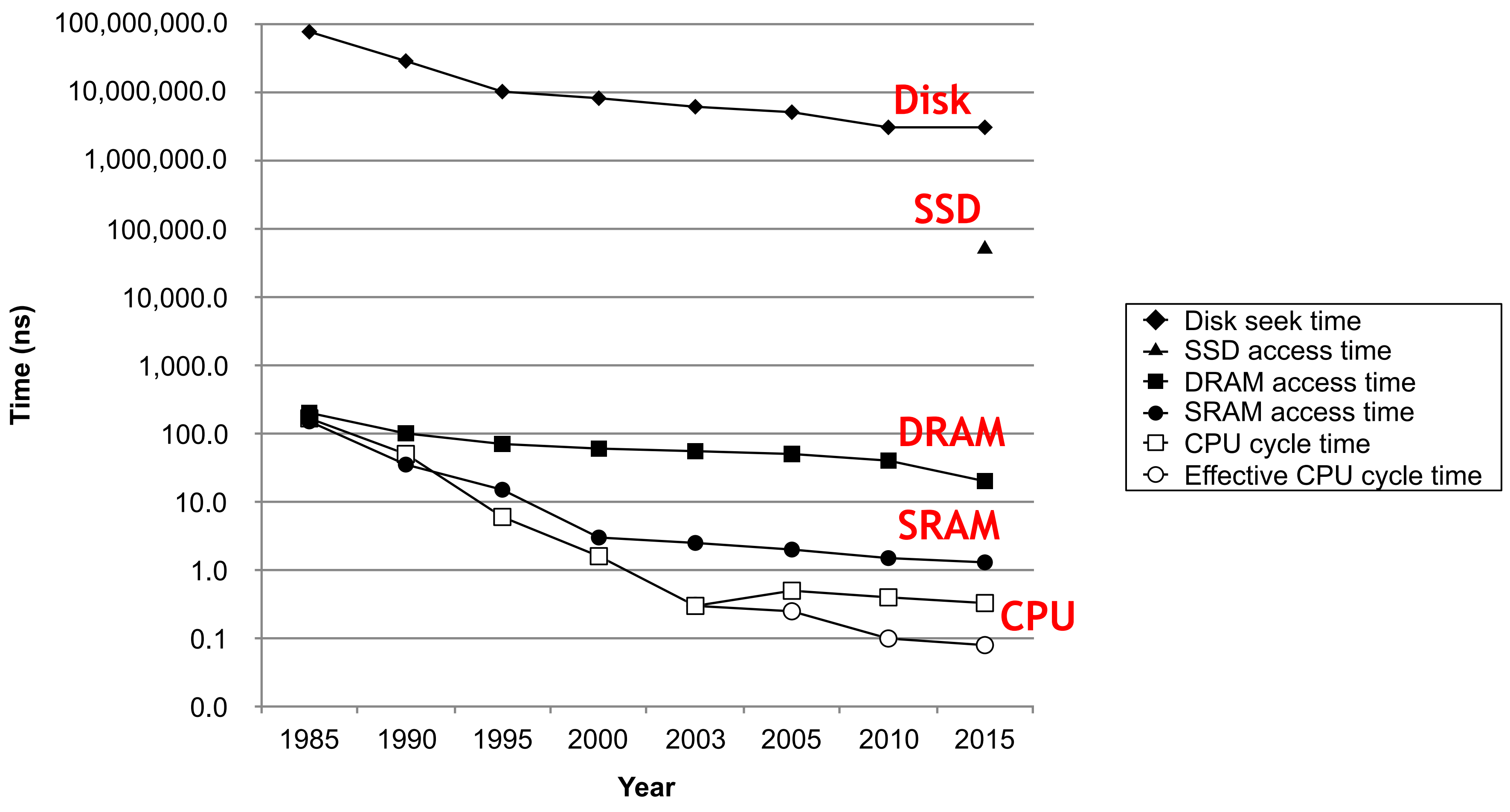
+ Memory Write Transaction (3)

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



+ The CPU-Memory Gap

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



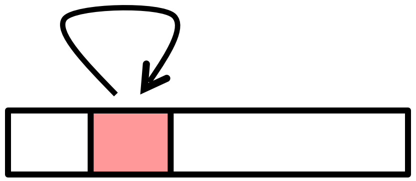
+

Locality

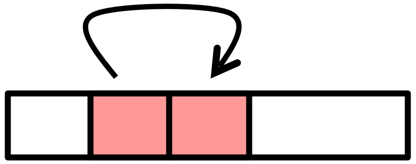
+ Locality

- **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;
```

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

+ Locality Example

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

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

Spatial locality

+ Locality Example

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

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

Locality Example

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

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

+ Locality Example

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

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

+ Locality Example #1

- **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;
}
```

+ Locality Example #2

- **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;
}
```

Locality Example #3

- **Question:** Can you permute the loop *indices* 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 < M; i++)
        for (j = 0; j < N; j++)
            for (k = 0; k < N; k++)
                sum += a[k][i][j];

    return sum;
}
```

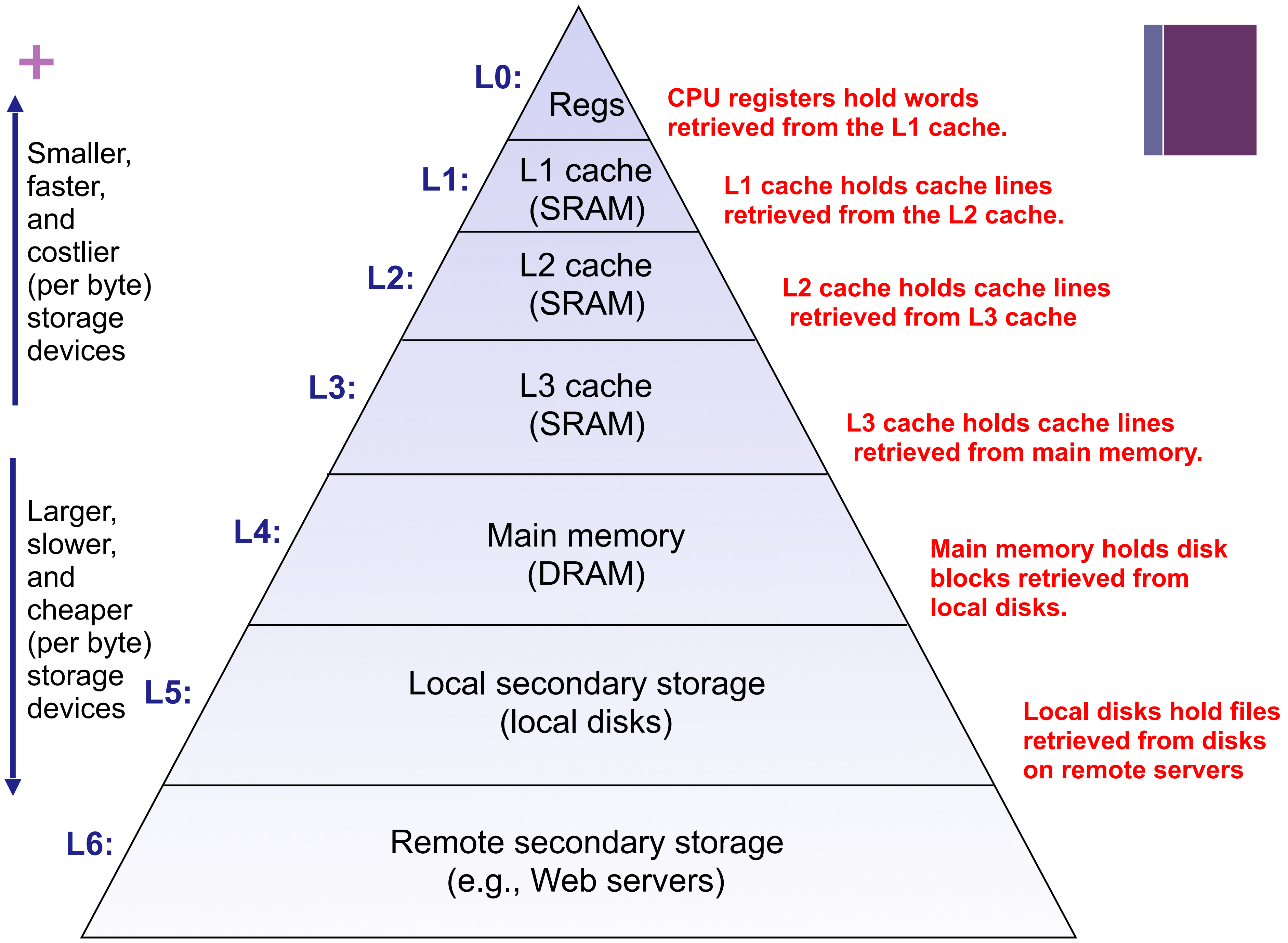
+ Memory Hierarchies

- **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 lead to an approach for organizing memory and storage systems known as a memory hierarchy.**

+

Memory Hierarchy & Caches



Smaller, faster, and costlier (per byte) storage devices
Larger, slower, and cheaper (per byte) storage devices
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)
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

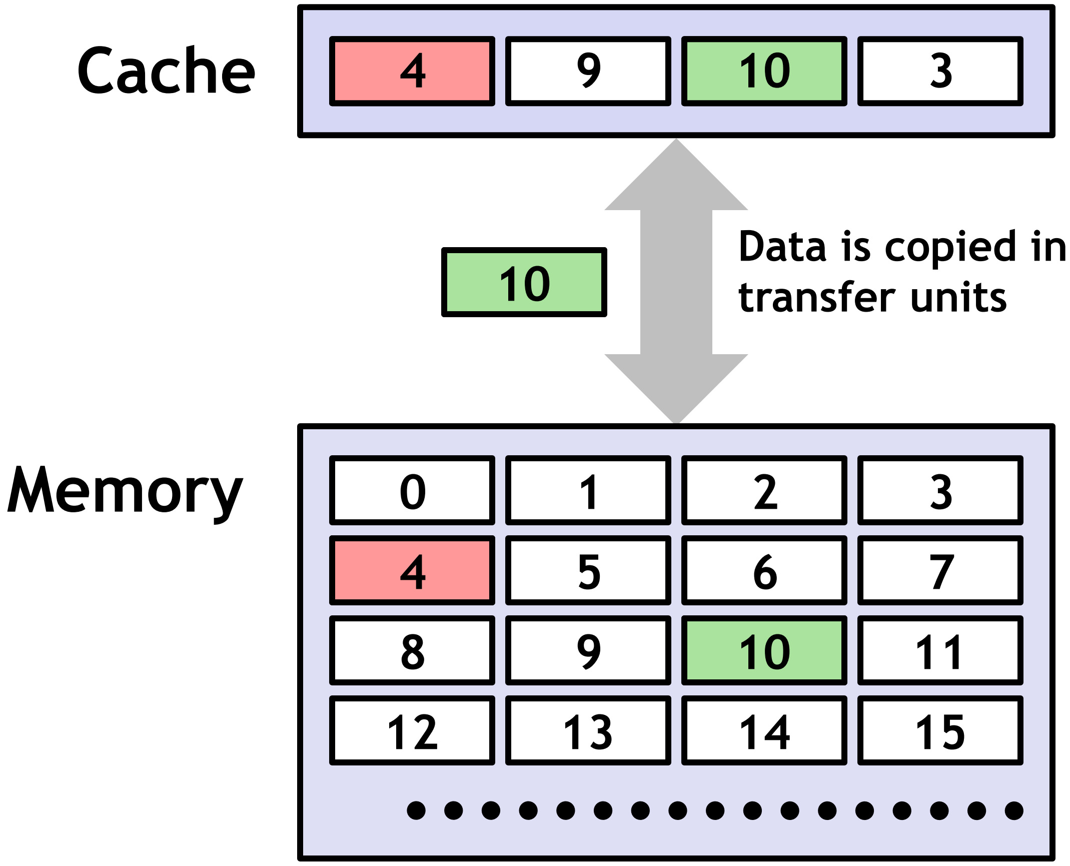
- **Cache:** A smaller, faster storage device that acts as a staging area for a subset of the data in a larger, slower device.

- **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 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 Analogy: Hangry!

- **Option 1: Go to refrigerate**
  - Found -> Eat
  - Latency: 1 minute
- **Option 2: Go to deli**
  - Found -> Purchase, take home, eat.
  - Latency: 20 minutes
- **Option 3: Grow food**
  - Plant, wait…. harvest, eat
  - Latency: ~250k minutes (~6 months)

General Cache Concepts

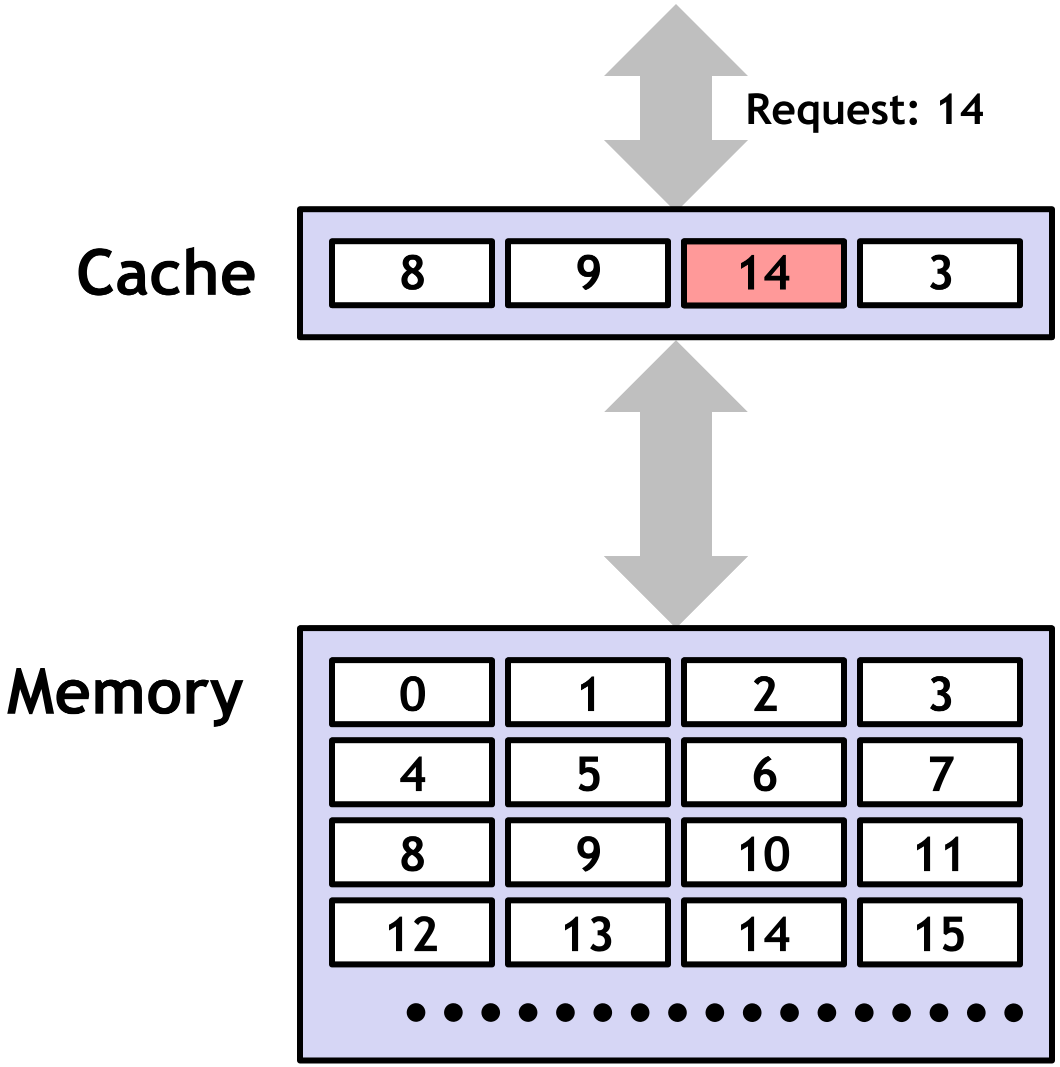


Smaller, faster, more expensive memory caches a subset of the blocks

Data is copied in block-sized transfer units

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

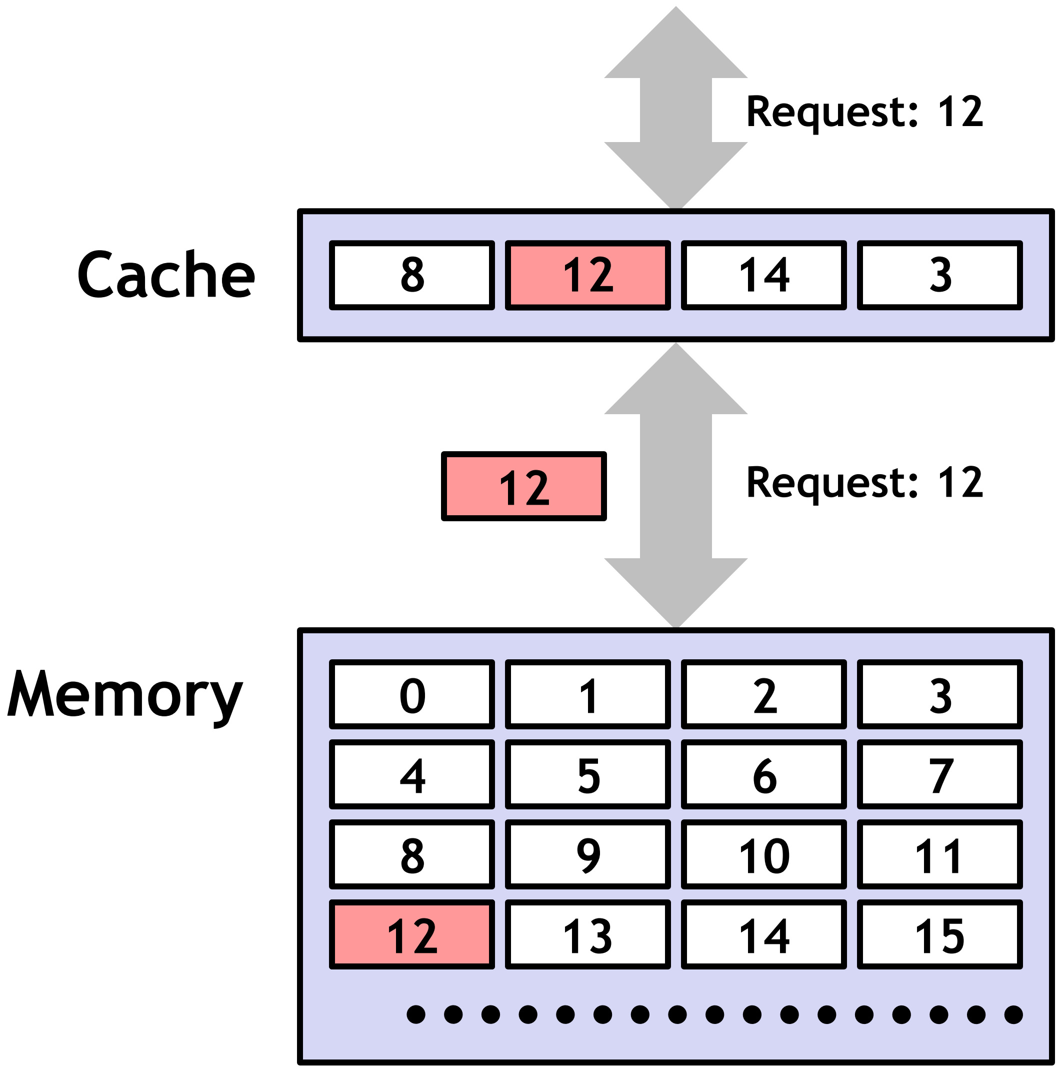
+ General Cache Concepts: Hit



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: Cache Miss Types

- **Cold (compulsory) miss**
  - Cold misses occur because the cache is empty.
- **Capacity miss**
  - 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 of the block positions at level k.
    - E.g. Given our example from previous slides, 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. Given our example from previous slides, referencing blocks 0, 4, 0, 4, 0, 4, ... 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	8 bytes words	CPU core	0	Compiler
L1 cache	64-byte blocks	On-Chip L1	4	Hardware
L2 cache	64-byte blocks	On-Chip L2	10	Hardware
Buffer cache	Parts of files	Main memory	100	OS
Disk cache	Disk sectors	Disk controller	100,000	Disk
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	1,000,000,000	Web proxy server

+ Summary

- 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.

- Caches are used everywhere in modern systems.