



University at Buffalo

Department of Computer Science  
and Engineering

School of Engineering and Applied Sciences

# *Caching*

Karthik Dantu

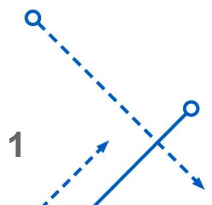
Ethan Blanton

Computer Science and Engineering

University at Buffalo

`kdantu@buffalo.edu`

Karthik Dantu



# Writing & Reading Memory

- Write

- Transfer data from CPU to memory

`movq 8(%rsp), %rax`

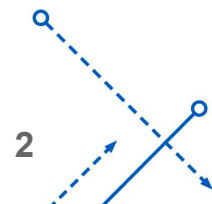
- “Store” operation

- Read

- Transfer data from memory to CPU

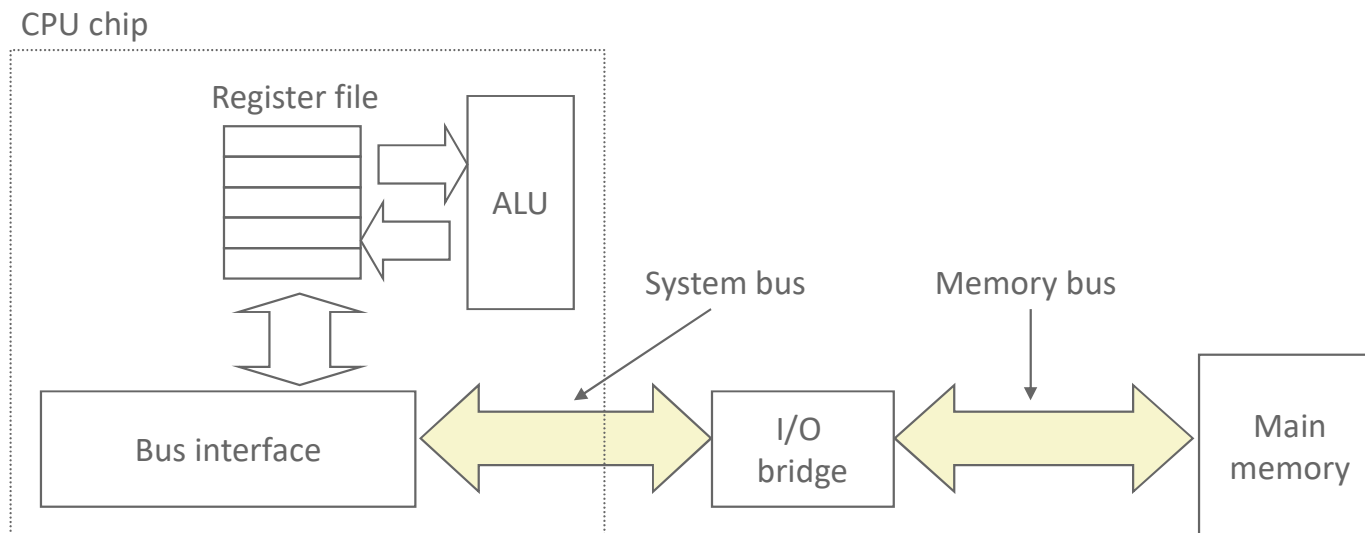
`movq %rax, 8(%rsp)`

- “Load” operation

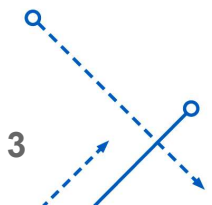


# Traditional Bus Structure Connecting CPU and Memory

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

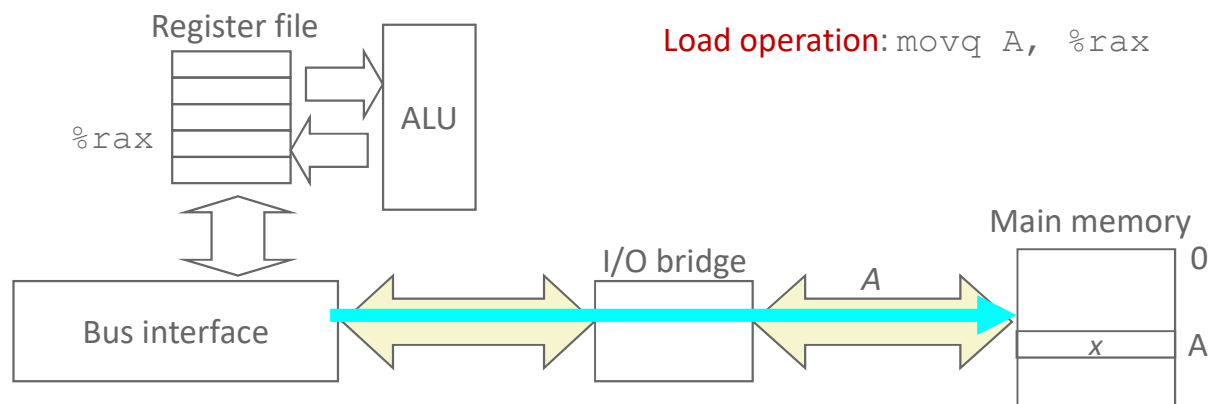


Karthik Dantu



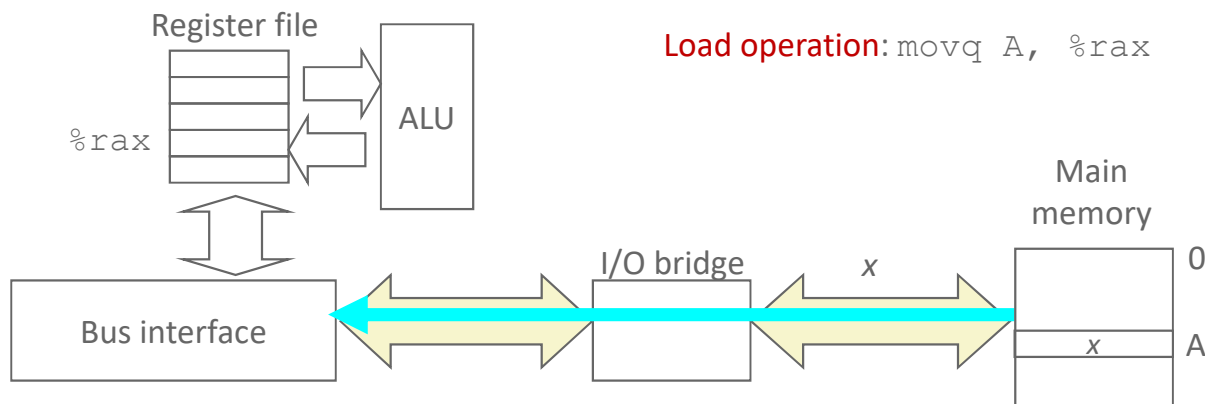
# Memory Read Transaction (1)

- CPU places address  $A$  on the memory bus.



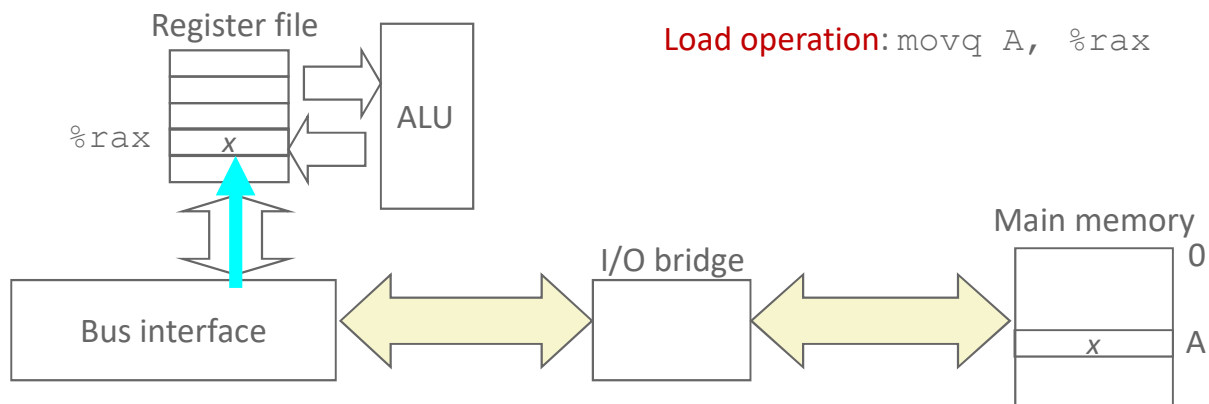
## Memory Read Transaction (2)

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



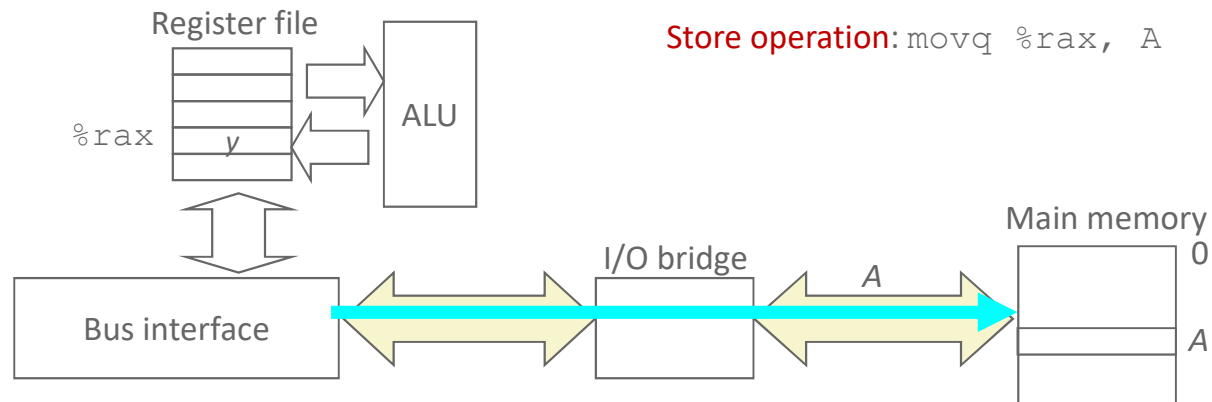
## Memory Read Transaction (3)

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



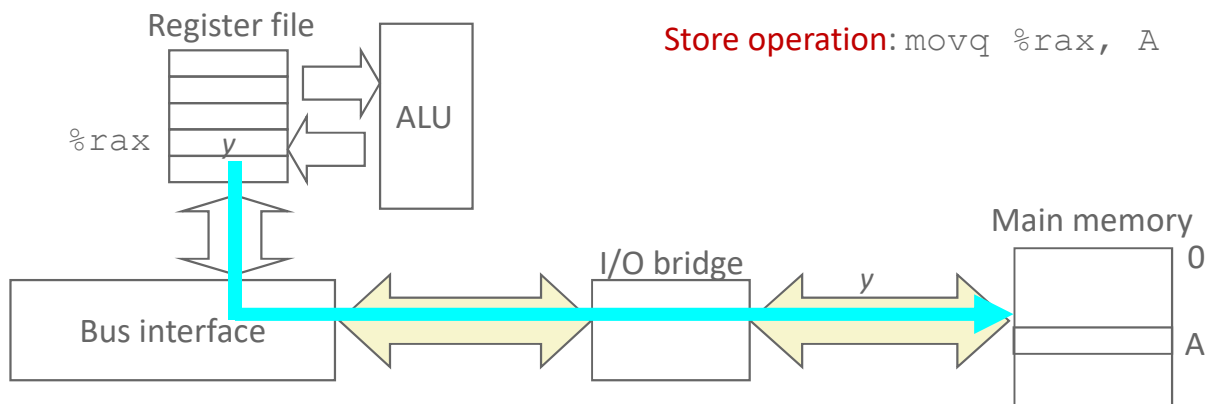
## Memory Write Transaction (1)

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



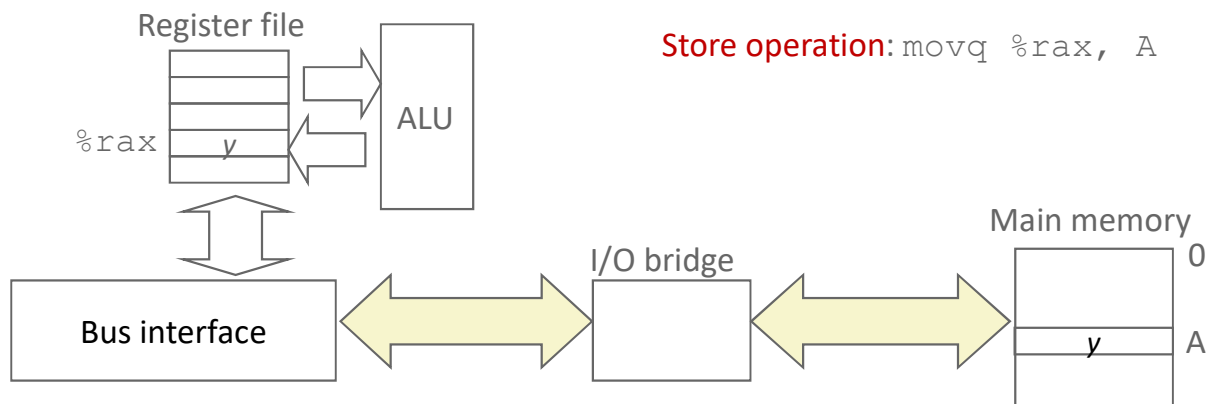
## Memory Write Transaction (2)

- CPU places data word  $y$  on the bus.



## Memory Write Transaction (3)

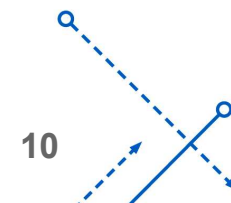
- Main memory reads data word  $y$  from the bus and stores it at address  $A$ .





# Today

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

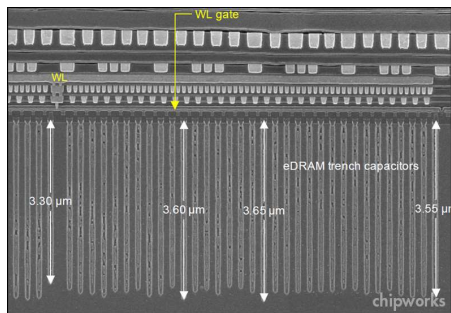


# Random-Access Memory (RAM)

- Key features
  - **RAM** is traditionally packaged as a chip.
    - or embedded as part of processor 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)

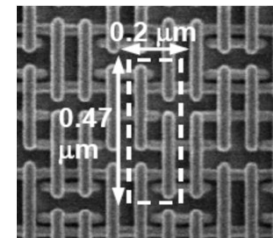
# RAM Technologies

- DRAM

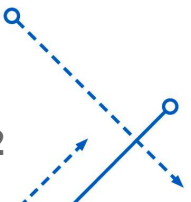


- 1 Transistor + 1 capacitor / bit
  - Capacitor oriented vertically
- Must refresh state periodically

- SRAM



- 6 transistors / bit
- Holds state indefinitely

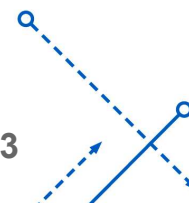


# SRAM vs DRAM Summary

	Trans. per bit	Access time	Needs refresh?	Needs EDC?	Cost	Applications
SRAM	6 or 8	1x	No	Maybe	100x	Cache memories
DRAM	1	10x	Yes	Yes	1x	Main memories, frame buffers

EDC: Error detection and correction

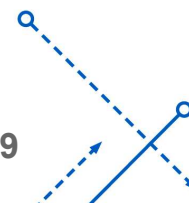
- Trends
  - SRAM scales with semiconductor technology
    - Reaching its limits
  - DRAM scaling limited by need for minimum capacitance
    - Aspect ratio limits how deep can make capacitor
    - Also reaching its limits





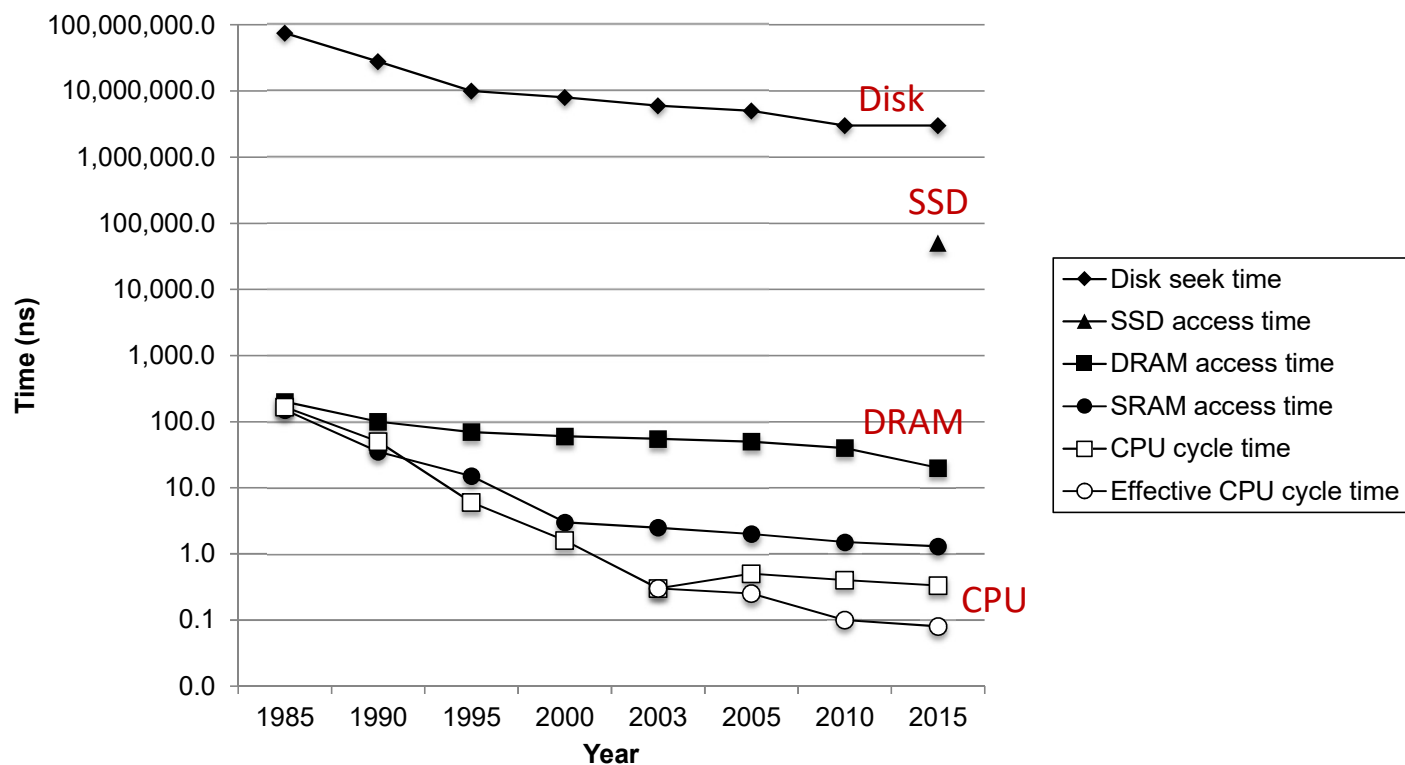
# Today

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



# The CPU-Memory Gap

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

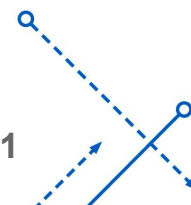


Karthik Dantu



# Locality to the Rescue!

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

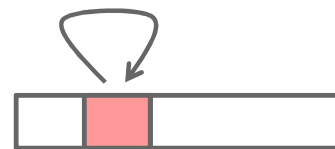


# Locality

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

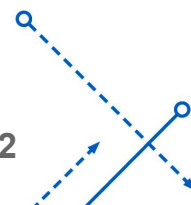
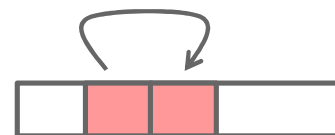
- **Temporal locality:**

- 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;
```

- Data references

- Reference array elements in succession (stride-1 reference pattern).
- Reference variable **sum** each iteration.

Spatial or Temporal  
Locality?

spatial

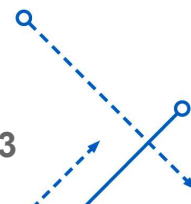
temporal

- Instruction references

- Reference instructions in sequence.
- Cycle through loop repeatedly.

spatial

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  $a$ ?

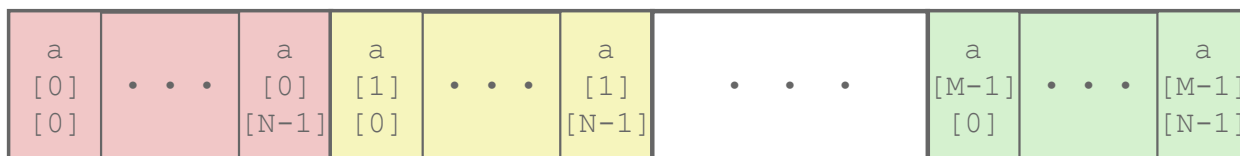
Hint: array layout is row-major order

Answer: yes

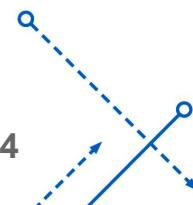
```
int sum_array_rows(int a[M][N])
{
    int i, j, sum = 0;

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

    return sum;
}
```



Karthik Dantu



# Locality Example

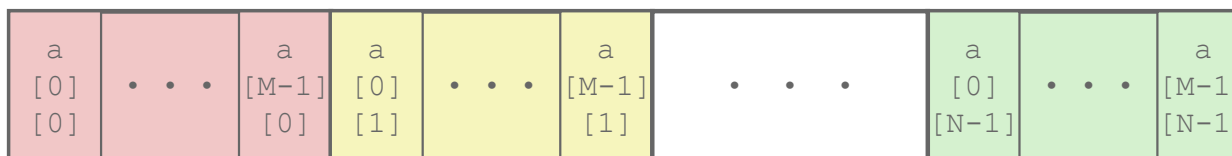
- Question:** Does this function have good locality with respect to array  $a$ ?

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

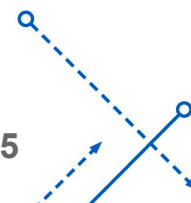
    for (j = 0; j < N; j++)
        for (i = 0; i < M; i++)
            sum += a[i][j];
    return sum;
}
```

Answer: no, unless...

$M$  is very small



Karthik Dantu



## Locality Example

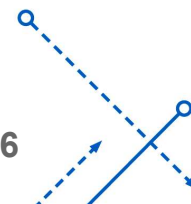
- **Question:** Can you permute the loops so that the function scans the 3-d array *a* with a stride-1 reference pattern (and thus has good spatial locality)?

```
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

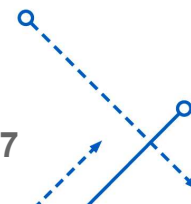
Karthik Dantu





# Today

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



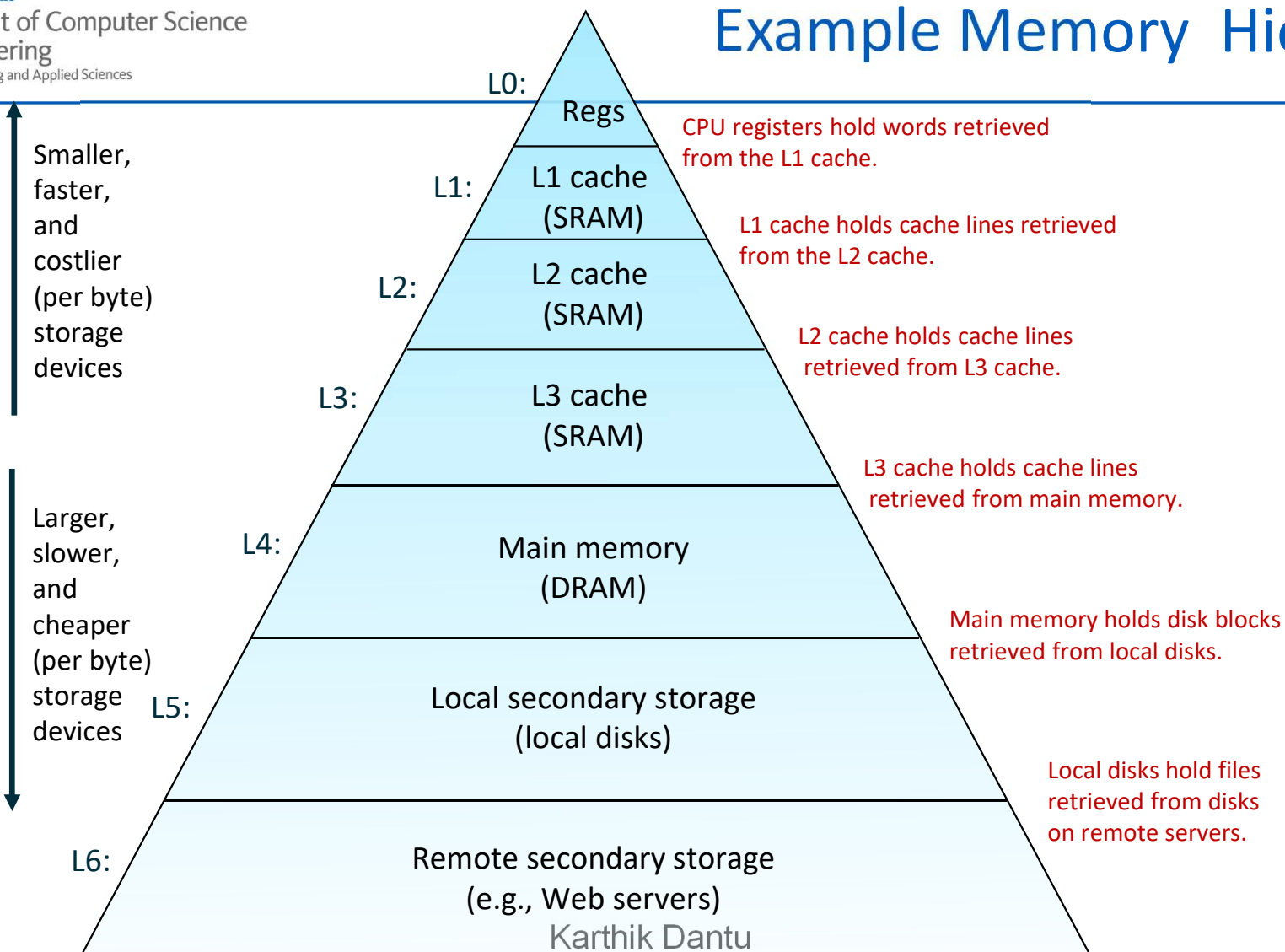
# Memory Hierarchies

- 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**.



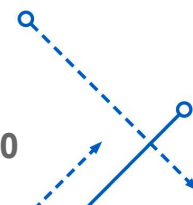


# Example Memory Hierarchy



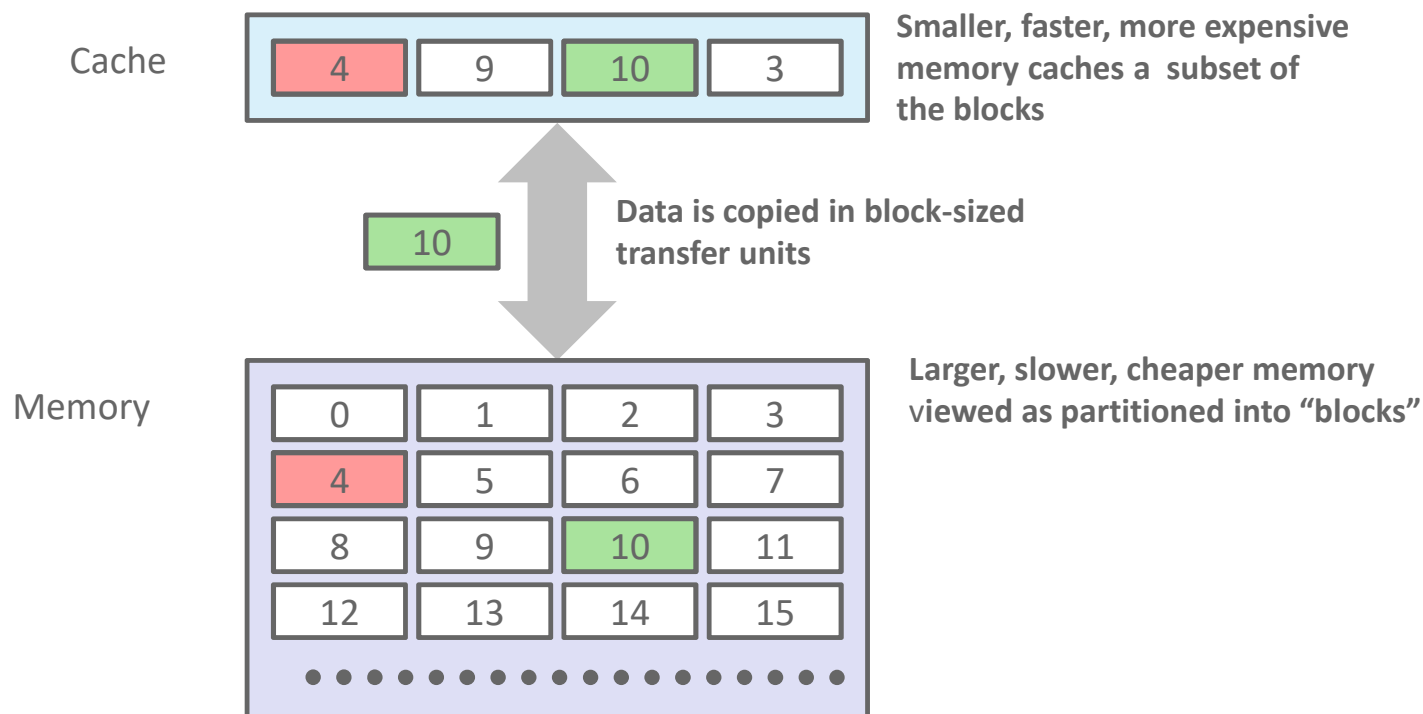
# Caches

- **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.
- **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.

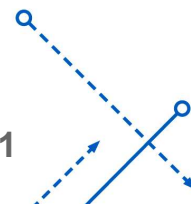




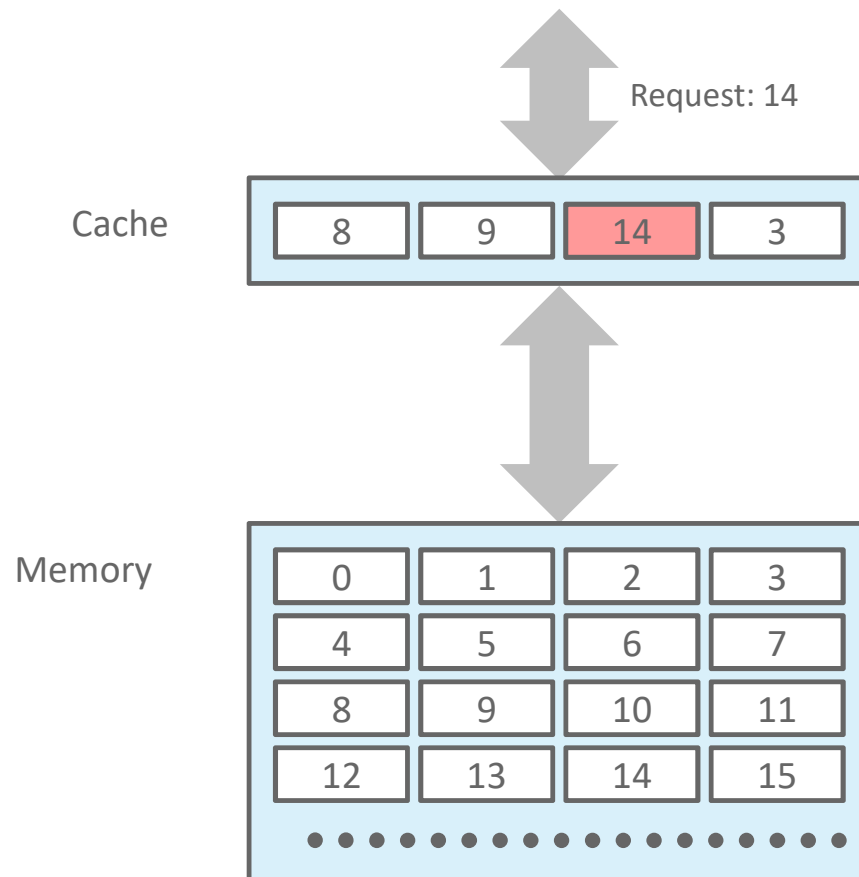
# General Cache Concepts



Karthik Dantu



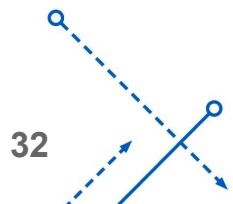
# General Cache Concepts: Hit



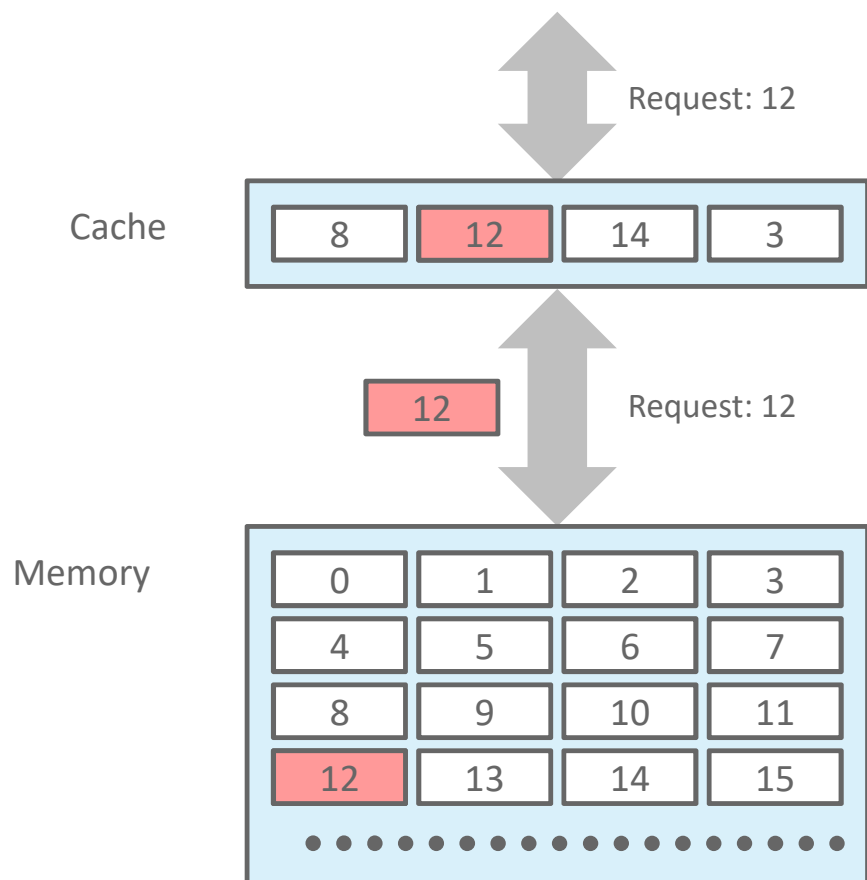
*Data in block b is needed*

*Block b is in cache:*  
*Hit!*

Karthik Dantu



# General Cache Concepts: Miss



*Data in block b is needed*

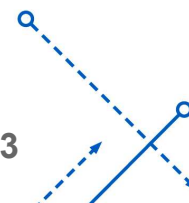
*Block b is not in cache:*  
**Miss!**

*Block b is fetched from memory*

*Block b is stored in cache*

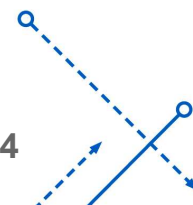
- **Placement policy:**  
determines where b goes
- **Replacement policy:**  
determines which block gets evicted (victim)

Karthik Dantu



# 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  $(i \bmod 4)$  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.

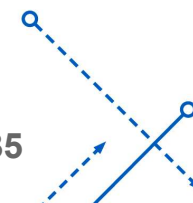




# Examples of Caching in the Mem. Hierarchy

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

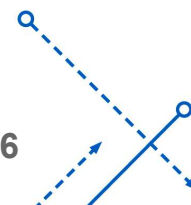
Karthik Dantu





# Today

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



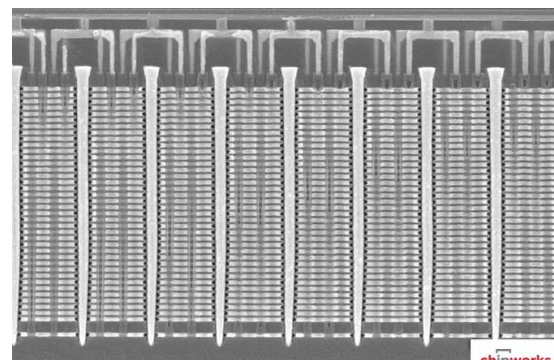
# Storage Technologies

- Magnetic Disks



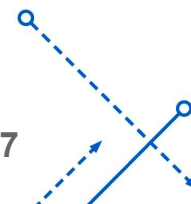
- Store on magnetic medium
- Electromechanical access

- Nonvolatile (Flash) Memory

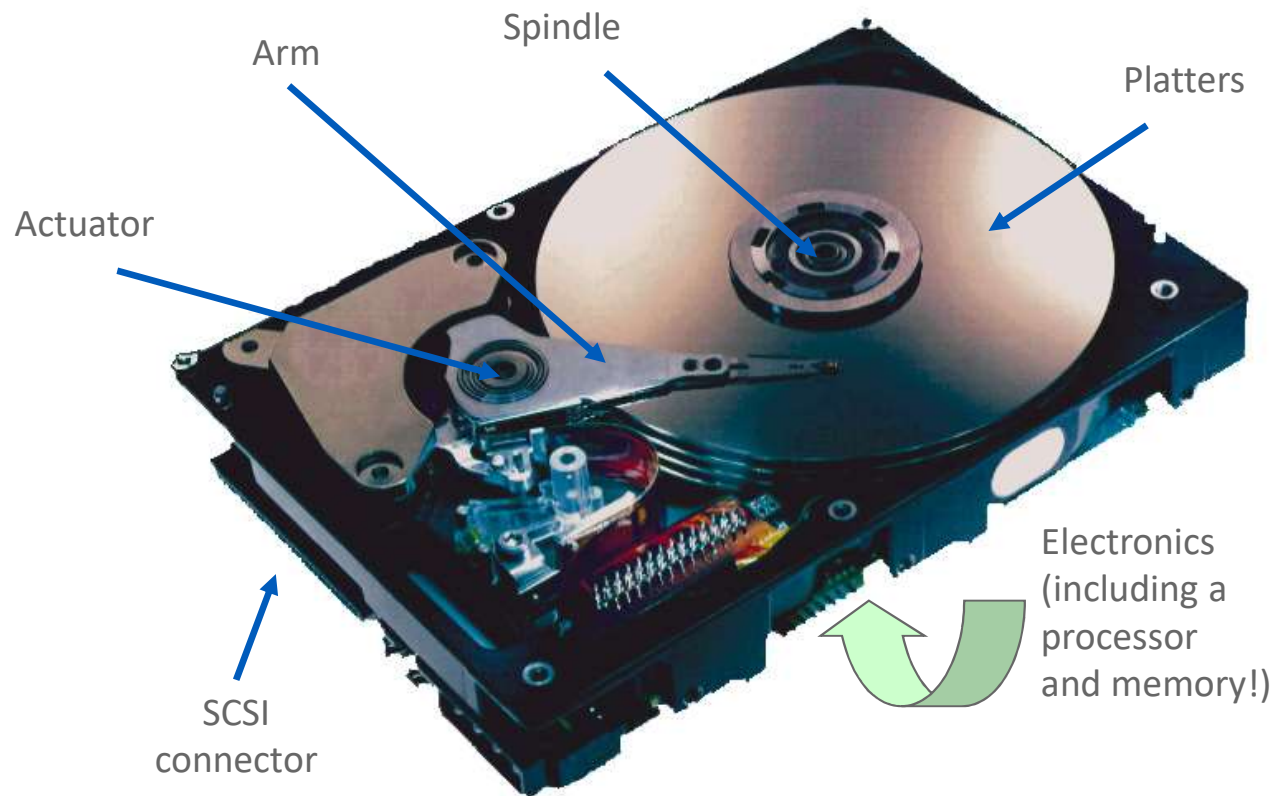


Close-up image of V-NAND flash array

- Store as persistent charge
- Implemented with 3-D structure
  - 100+ levels of cells
  - 3 bits data per cell

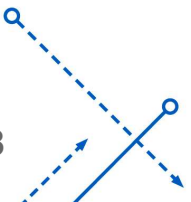


# What's Inside A Disk Drive?



*Image courtesy of Seagate Technology*

Karthik Dantu



# Disk Access Time

- Average time to access some target sector approximated by:

- $T_{\text{access}} = T_{\text{avg seek}} + T_{\text{avg rotation}} + T_{\text{avg transfer}}$

- **Seek time** ( $T_{\text{avg seek}}$ )

- Time to position heads over cylinder containing target sector.
  - Typical  $T_{\text{avg seek}}$  is 3—9 ms

- **Rotational latency** ( $T_{\text{avg rotation}}$ )

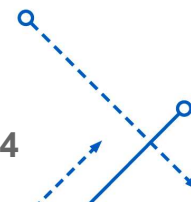
- Time waiting for first bit of target sector to pass under r/w head.
  - $T_{\text{avg rotation}} = 1/2 \times 1/\text{RPMs} \times 60 \text{ sec}/1 \text{ min}$
  - Typical rotational rate = 7,200 RPMs

- **Transfer time** ( $T_{\text{avg transfer}}$ )

- Time to read the bits in the target sector.
  - $T_{\text{avg transfer}} = \frac{1}{\text{RPM}} \times \frac{1}{(\text{avg \# sectors/track})} \times 60 \text{ secs}/1 \text{ min}$

time for one rotation (in minutes)      fraction of a rotation to be read

Karthik Dantu



# Disk Access Time Example

- Given:
  - Rotational rate = 7,200 RPM
  - Average seek time = **9 ms**
  - Avg # sectors/track = 400
- Derived:
  - $T_{\text{avg rotation}} = 1/2 \times (60 \text{ secs}/7200 \text{ RPM}) \times 1000 \text{ ms/sec} = \mathbf{4 \text{ ms}}$
  - $T_{\text{avg transfer}} = 60/7200 \times 1/400 \times 1000 \text{ ms/sec} = \mathbf{0.02 \text{ ms}}$
  - $T_{\text{access}} = \mathbf{9 \text{ ms} + 4 \text{ ms} + 0.02 \text{ ms}}$
- Important points:
  - Access time dominated by seek time and rotational latency.
  - First bit in a sector is the most expensive, the rest are free.
  - ***SRAM access time is about 4 ns/doubleword, DRAM about 60 ns***
    - *Disk is about 40,000 times slower than SRAM,*
    - *2,500 times slower than DRAM.*

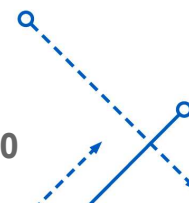


# Nonvolatile Memories

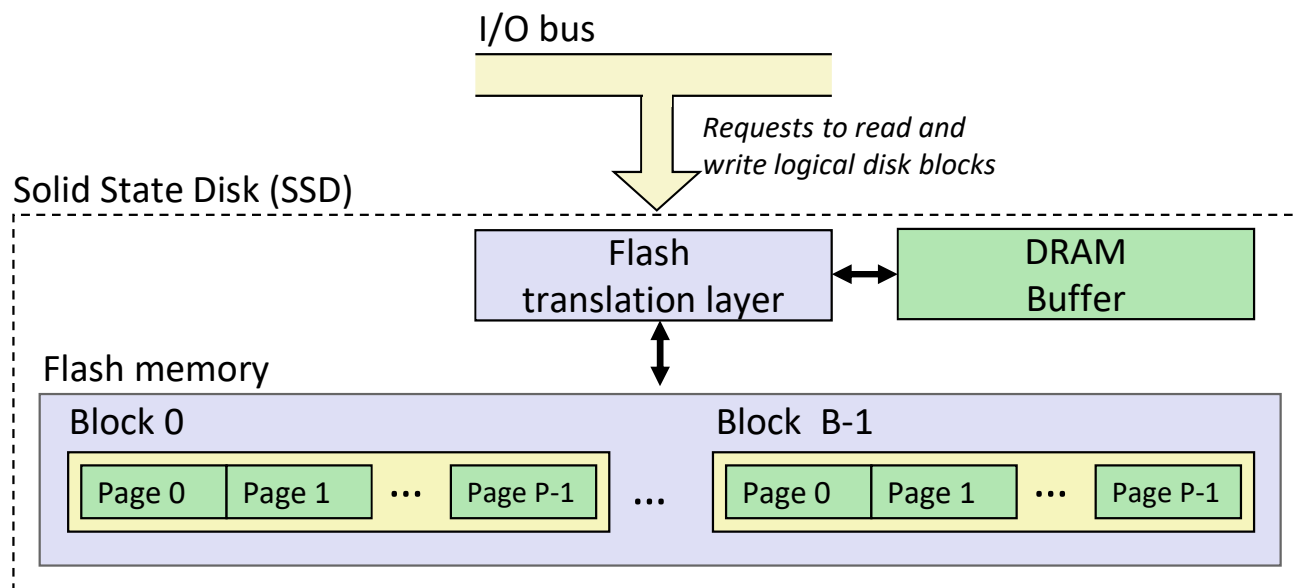
- DRAM and SRAM are volatile memories
  - Lose information if powered off.
- Nonvolatile memories retain value even if powered off
  - Read-only memory (**ROM**): programmed during production
  - Electrically erasable PROM (**EEPROM**): electronic erase capability
  - Flash memory: EEPROMs, with partial (block-level) erase capability
    - Wears out after about 100,000 erasings
  - 3D XPoint (Intel Optane) & emerging NVMs
    - New materials
- Uses for Nonvolatile Memories
  - Firmware programs stored in a ROM (BIOS, controllers for disks, network cards, graphics accelerators, security subsystems,...)
  - Solid state disks (replacing rotating disks)
  - Disk caches



Karthik Dantu

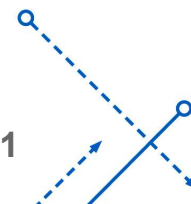


# Solid State Disks (SSDs)



- Pages: 512KB to 4KB, Blocks: 32 to 128 pages
- Data read/written in units of pages.
- Page can be written only after its block has been erased.
- A block wears out after about 100,000 repeated writes.

Karthik Dantu



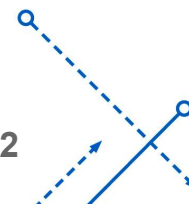
# SSD Performance Characteristics

- Benchmark of Samsung 940 EVO Plus

<https://ssd.userbenchmark.com/SpeedTest/711305/Samsung-SSD-970-EVO-Plus-250GB>

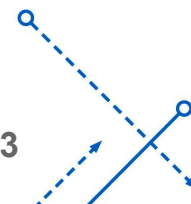
Sequential read throughput	2,126 MB/s	Sequential write tput	1,880 MB/s
Random read throughput	140 MB/s	Random write tput	59 MB/s

- Sequential access faster than random access
  - Common theme in the memory hierarchy
- Random writes are somewhat slower
  - Erasing a block takes a long time (~1 ms).
  - Modifying a block page requires all other pages to be copied to new block.
  - Flash translation layer allows accumulating series of small writes before doing block write.



# SSD Tradeoffs vs Rotating Disks

- Advantages
  - No moving parts → faster, less power, more rugged
- Disadvantages
  - Have the potential to wear out
    - Mitigated by “wear leveling logic” in flash translation layer
    - E.g. Samsung 940 EVO Plus guarantees 600 writes/byte of writes before they wear out
    - Controller migrates data to minimize wear level
  - In 2019, about 4 times more expensive per byte
    - And, relative cost will keep dropping
- Applications
  - MP3 players, smart phones, laptops
  - Increasingly common in desktops and servers



# 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.
- Flash memory progress outpacing all other memory and storage technologies (DRAM, SRAM, magnetic disk)
  - Able to stack cells in three dimensions

