The Memory Hierarchy

Today
- Storage technologies and trends
- Locality of reference
- Caching in the memory hierarchy

Next time
- Cache memory
Random-Access Memory (RAM)

- Computer technology success due in big part to progress in storage technology
- Random-Access Memory
  - Packaged as a chip
  - Basic storage unit is a cell (one bit per cell).
  - Multiple RAM chips form a memory
  - Comes in two flavors static and dynamic

IBM PC 5150
August 1981
16KB Main memory
36kbit DRAM chips
Static and Dynamic RAMs

- Static RAM (SRAM)
  - Static – no need for periodic refreshed; keeps data while powered
  - Each cell stores bit in a bistable cell, a six-transistor circuit
  - Relatively insensitive to disturbances such as electrical noise
  - *Faster and more expensive* than DRAM

- Dynamic RAM (DRAM)
  - Each cell stores bit as a charge in a capacitor, cell is a capacitor and a single access transistor
  - Value must be refreshed every 10-100 ms
  - Sensitive to disturbances
  - *Slower and cheaper* than SRAM

<table>
<thead>
<tr>
<th></th>
<th>Transistors/bit</th>
<th>Acc time</th>
<th>Persist?</th>
<th>Sensitive?</th>
<th>Cost</th>
<th>Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRAM</td>
<td>6</td>
<td>1X</td>
<td>Yes</td>
<td>No</td>
<td>100X</td>
<td>Cache mem.</td>
</tr>
<tr>
<td>DRAM</td>
<td>1</td>
<td>10X</td>
<td>No</td>
<td>Yes</td>
<td>1X</td>
<td>Main mem., frame buffers</td>
</tr>
</tbody>
</table>
Conventional DRAM organization

- Cell in a DRAM chip partitioned into supercells
  - Supercells organized as arrays of $r$ rows and $c$ cols ($r\times c = d$)
  - Each of $w$ DRAM cells; a $(d \times w)$ DRAM stores $(d \times w)$ bits

![Diagram of DRAM organization](image)
Reading DRAM supercell (2,1)

- Access done in two steps
  - Step 1(a): Row access strobe (RAS) selects row 2.
  - Step 1(b): Row 2 copied from DRAM array to row buffer.
Reading DRAM supercell (2,1)

... 

- Step 2(a): Column access strobe (CAS) selects col 1.
- Step 2(b): Supercell (2,1) copy row buffer to data lines, and eventually back to the CPU.
Memory modules

- DRAM chips are packaged in memory modules
- Mem. modules plug into expansion slots on main system board
  - Examples
    - 168-pin Dual Inline Memory Module (DIMM) – transfer data in 64-bit chunks
    - 72-pin Single Inline Memory Module (SIMM) – transfer data in 32-bit chunks
Memory module basic idea

- Memory module consisting of eight 8Mx8 DRAMs
- Super cell (i, j)
- 64 MB memory module
- 64-bit doubleword at main memory address A
- Memory controller
Enhanced DRAMs

- Basic cell has not changed since 1966
- Enhanced DRAMs are built around conventional core
  - Synchronous DRAM (SDRAM)
    - Driven with rising clock edge instead of asynchronous control signals.
    - Allows reuse of the row address [RAS, CAS, CAS, CAS, CAS]
  - Double data-rate synchronous DRAM (DDR SDRAM)
    - Enhancement of SDRAM that uses both clock edges as control signals.
    - Different types with different sizes for small prefetch buffer
      - DDR (2b), DDR2 (4b), DDR4 (8b)
    - Standard for most servers and desktops (2010)
Nonvolatile memories

- DRAM and SRAM are volatile memories
  - Lose information if powered off
- Nonvolatile memories retain value even if powered off
  - Generic name is read-only memory (ROM)
  - Misleading because some ROMs can be read and modified.

Types of ROMs
- Programmable ROM (PROM)
- Erasable Programmable ROM (EPROM)
- Electrically Erasable PROM (EEPROM)
- Flash memory (based on EEPROM)

Firmware
- Program stored in a ROM
  - Boot time code, BIOS (basic input/output system)
  - graphics cards, disk controllers.
Accessing main memory

- Data flows between main memory and CPU over buses
  - A collection of parallel wires that carry address, data, and control signals
  - Typically shared by multiple devices

Example configuration
Memory read transaction (1)

- Load content of address A into a register
- CPU places address A on the system bus, I/O bridge passes it on to the memory bus

Load operation: `movl A, %eax`
Memory read transaction (2)

- Main memory reads A from the memory bus, retrieves word x, and places it on the bus; I/O bridge passes it along to the system bus

Load operation: `movl A, %eax`

![Diagram showing memory read transaction]
Memory read transaction (3)

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

Load operation: \texttt{movl A, %eax}
Memory write transaction (1)

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

\[
\text{Store operation:} \text{movl } \%eax, A
\]
Memory write transaction (2)

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

\[ \text{Store operation: } \text{movl } \%eax, A \]
Memory write transaction (3)

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

Store operation: \texttt{movl \%eax, A}
Disk storage

- Workhorse storage devices
  - 100-1,000x GB
  - Milliseconds to read (100,000x longer than from DRAM, 1,000,000x longer than from SRAM)

IBM 350 Disk Storage Unit
Announced Sep. 4, 1956
5MB

Seagate Barracuda XT
2TB, 2010
Disk geometry

- Disks consist of **platters**, each with two **surfaces**
- Each surface consists of concentric rings called **tracks**
- Each track consists of **sectors** separated by **gaps**
- Sectors contain equal # of data bits (typically 512B)
Disk geometry (Multiple-platter view)

- Aligned tracks form a cylinder
Disk capacity

- **Capacity**: maximum number of bits that can be stored.
  - Vendors express capacity in units of gigabytes (GB), where 1 GB = 10^9

- Capacity is determined by these technology factors:
  - *Recording density* (bits/in): # bits that fit into a 1 inch of a track
  - *Track density* (tracks/in): # of tracks that fit into a 1 inch radial segment
  - *Areal density* (bits/in^2): recording density * track density

- Modern disks partition tracks into disjoint subsets called recording zones
  - Each track in a zone has the same number of sectors, determined by the circumference of innermost track
  - Each zone has a different number of sectors/track
Computing disk capacity

- Capacity =
  
  (# bytes/sector)  
  x (avg. # sectors/track)  
  x (# tracks/surface)  
  x (# surfaces/platter)  
  x (# platters/disk)

- Example:
  - 512 bytes/sector
  - 300 sectors/track (on average)
  - 20,000 tracks/surface
  - 2 surfaces/platter
  - 5 platters/disk

- Capacity = 512 x 300 x 20000 x 2 x 5 = 30,720,000,000 = 30.72 GB
Disk operation (Single-platter view)

The disk surface spins at a fixed rotational rate (5400-15000 RPM)

The read/write head is attached to the end of the arm and flies over the disk surface on a thin cushion of air.

By moving radially, the arm can position the read/write head over any track.

read/write heads move in unison from cylinder to cylinder
Disk access time

- **Avg time to access 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 \( T_{\text{avg seek}} = 9 \) ms

- **Rotational latency** (\( T_{\text{avg rotation}} \))
  - Time waiting for first bit of sector to pass under r/w head.
  - \( T_{\text{avg rotation}} = T_{\text{max rotation}} / 2 = 1/\text{RPMs} \times 60 \text{ sec} / 1 \text{ min} \times 1/2 \)

- **Transfer time** (\( T_{\text{avg transfer}} \))
  - Time to read the bits in the target sector.
  - \( T_{\text{avg transfer}} = 1/\text{RPM} \times 1/(\text{avg # sectors/track}) \times 60 \text{ secs} / 1 \text{ min.} \)
Disk access time example

- **Given:**
  - Rotational rate = 7,200 RPM
  - Average seek time = 9 ms.
  - Avg # sectors/track = 400.

- **Derived:**
  - \( T_{\text{avg rotation}} = \frac{1}{2} \times \left( \frac{60 \text{ secs}}{7200 \text{ RPM}} \right) \times 1000 \text{ ms/sec} = 4 \text{ ms.} \)
  - \( T_{\text{avg transfer}} = \frac{60}{7200} \text{ RPM} \times \frac{1}{400} \text{ sec/track} \times 1000 \text{ ms/sec} = 0.02 \text{ ms} \)
  - \( T_{\text{access}} = T_{\text{avg seek}} + T_{\text{avg rotation}} + T_{\text{avg transfer}} = 9 \text{ ms} + 4 \text{ ms} + 0.02 \text{ ms} \)

- **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 \( \sim 4 \text{ ns/doubleword} \), DRAM \( \sim 60 \text{ ns} \)
    - Disk is about 40,000 times slower than SRAM,
    - 2,500 times slower than DRAM
Logical disk blocks

- Disks present a simpler abstract view of the complex sector geometry
  - Set of available sectors modeled as a sequence of b-sized logical blocks (0, 1, 2, ...)

- Mapping between logical blocks and physical sectors
  - Maintained by hw/firmware device called disk controller.
  - Converts requests for logical blocks into (surface, track, sector) triples

- Allows controller to set aside spare cylinders per zone
  - Accounts for the difference in “formatted capacity” and “maximum capacity”
Connecting I/O devices – I/O Bus

Monitor, mice, keywords, disks, … connected to the CPU and main memory using an I/O bus.

Expansion slots for other devices such as network adapters.

Modeled on Intel’s Peripheral Component Interconnect (PCI)
CPU initiates a disk read by writing a command, logical block number, and destination memory address to a *port* (address) associated with disk controller.
Disk controller reads the sector and performs a direct memory access (DMA) transfer into main memory.
When the DMA transfer completes, the disk controller notifies the CPU with an interrupt (i.e., asserts a special “interrupt” pin on the CPU)!
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 ~100,000 repeated writes
### SSD Performance Characteristics

**Why are random writes so slow?**
- Erasing a block is slow (around 1 ms)
- Write to a page triggers a copy of all useful pages in the block
  - Find an used block (new block) and erase it
  - Write the page into the new block
  - Copy other pages from old block to the new block

<table>
<thead>
<tr>
<th></th>
<th>Reads</th>
<th>Writes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential read tput</td>
<td>250MB/s</td>
<td>Sequential write tput</td>
</tr>
<tr>
<td>Random read tput</td>
<td>140MB/s</td>
<td>Random write tput</td>
</tr>
<tr>
<td>Random read access</td>
<td>30us</td>
<td>Random write access</td>
</tr>
</tbody>
</table>
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. Intel X25 guarantees 1 petabyte ($10^{15}$ bytes) of random writes before they wear out
  - In 2010, about 100 times more expensive per byte

- **Applications**
  - MP3 players, smart phones, laptops
  - Beginning to appear in desktops and servers
# Storage trends

## SRAM

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$/MB</td>
<td>19,200</td>
<td>2900</td>
<td>320</td>
<td>256</td>
<td>100</td>
<td>75</td>
<td>60</td>
<td>320</td>
</tr>
<tr>
<td>Access (ns)</td>
<td>300</td>
<td>150</td>
<td>35</td>
<td>15</td>
<td>3</td>
<td>2</td>
<td>1.5</td>
<td>200</td>
</tr>
</tbody>
</table>

## DRAM

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$/MB</td>
<td>800</td>
<td>880</td>
<td>100</td>
<td>30</td>
<td>1</td>
<td>.1</td>
<td>0.06</td>
<td>130,000</td>
</tr>
<tr>
<td>Access (ns)</td>
<td>375</td>
<td>200</td>
<td>100</td>
<td>70</td>
<td>60</td>
<td>50</td>
<td>40</td>
<td>9</td>
</tr>
<tr>
<td>Typical size (MB)</td>
<td>0.064</td>
<td>0.256</td>
<td>4</td>
<td>16</td>
<td>64</td>
<td>2000</td>
<td>8000</td>
<td>125,000</td>
</tr>
</tbody>
</table>

## Rotating disks

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$/MB</td>
<td>500</td>
<td>100</td>
<td>8</td>
<td>0.30</td>
<td>0.01</td>
<td>0.005</td>
<td>0.0003</td>
<td>1,600,000</td>
</tr>
<tr>
<td>Access (ns)</td>
<td>87</td>
<td>75</td>
<td>28</td>
<td>10</td>
<td>8</td>
<td>5</td>
<td>3</td>
<td>29</td>
</tr>
<tr>
<td>Typical size (MB)</td>
<td>1</td>
<td>10</td>
<td>16</td>
<td>1000</td>
<td>20,000</td>
<td>160,000</td>
<td>1,500,000</td>
<td>1,500,000</td>
</tr>
</tbody>
</table>

Cost decreased by a factor of ~200

Access times decreased by a factor of ~300

Different trends for DRAM and disks

*(Culled from back issues of Byte and PC Magazine)*
# CPU trends

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Intel CPU</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Clock rate (MHz)</td>
<td>1</td>
<td>6</td>
<td>20</td>
<td>150</td>
<td>600</td>
<td>3300</td>
<td>2000</td>
<td>2500</td>
<td>2500</td>
</tr>
<tr>
<td>Cycle time (ns)</td>
<td>1000</td>
<td>166</td>
<td>50</td>
<td>6</td>
<td>1.6</td>
<td>0.30</td>
<td>0.50</td>
<td>0.4</td>
<td>2500</td>
</tr>
<tr>
<td>Cores</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Eff. cycle times (ms)</td>
<td>1000</td>
<td>166</td>
<td>50</td>
<td>6</td>
<td>1.6</td>
<td>0.30</td>
<td>0.25</td>
<td>0.10</td>
<td>10,000</td>
</tr>
</tbody>
</table>

*Cycle time divided by number of cores*
The CPU-Memory gap

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

- **DRAM and disk and CPU performance widening**
- **SRAM roughly keeping up**
- **Split reflects the introduction of multicores**

<table>
<thead>
<tr>
<th>Year</th>
<th>Disk seek time</th>
<th>Flash SSD access time</th>
<th>DRAM access time</th>
<th>SRAM access time</th>
<th>CPU cycle time</th>
<th>Effective CPU cycle time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1980</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1985</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1990</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1995</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2003</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2005</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2010</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Locality

- Principle of Locality:
  - Programs tend to reuse/use data items recently used or nearby those recently used
  - Temporal locality: Recently referenced items are likely to be referenced in the near future
  - Spatial locality: Items with nearby addresses tend to be referenced close together in time

Locality Example:

- Data
  - Reference array elements in succession (stride-1 reference pattern): **Spatial locality**
  - Reference sum each iteration: **Temporal locality**

- Instructions
  - Reference instructions in sequence: **Spatial locality**
  - Cycle through loop repeatedly: **Temporal locality**

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

- To look at code and get a qualitative sense of its locality is a key skill for a professional programmer.
- *Does this function have good locality?*

```c
int sumarrayrows(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
}
```

- Yes! Array is accessed in the same row-major order in which it is stored in memory.
Locality example

- Does this function have good locality?

```c
int sumarraycols(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
}
```

- No! Scans array column-wise instead of row-wise
Locality example

- 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)?

```c
int sumarray3d(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
}
```

Visiting every k-element of a contiguous vector is called a stride-k reference pattern.

```c
int sumarray3d(int a[M][N][N])
{
    int i, j, k, sum = 0;

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

    return sum
}
```
Some fundamental and enduring properties of hardware and software:
- Fast storage tech cost more per byte and have less capacity.
- 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.
An example memory hierarchy

L0: registers

L1: on-chip L1 cache (SRAM)

L2: off-chip L2 cache (SRAM)

L3: main memory (DRAM)

L4: local secondary storage (local disks)

L5: remote secondary storage (distributed file systems, Web servers)

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

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

CPU registers hold words retrieved from L1 cache.

L1 cache holds cache lines retrieved from the L2 cache memory.

L2 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 network 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?**
  - 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
  - **Net effect**: A large pool of memory 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
Caching in a memory hierarchy

Level k:

Smaller, faster, more expensive device at level k caches a subset of the blocks from level k+1

Data is copied between levels in block-sized transfer units

Level k+1:

Larger, slower, cheaper storage device at level k+1 is partitioned into blocks.
General caching concepts

- Program needs object d, which is stored in some block b
- Cache hit
  - Program finds b in the cache at level k. E.g., block 14
- Cache miss
  - b is not at level k, so level k cache must fetch it from level k+1. E.g., block 12
  - If level k cache is full, then some current block must be replaced (evicted). Which one is the “victim”? 
    - Placement policy: where can the new block go? E.g., b mod 4
    - Replacement policy: which block should be evicted? E.g., LRU
General caching concepts

• Types of cache misses:
  – Cold (compulsary) 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 go to block (i mod 4) at level k+1
    • Conflict misses occur when 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
# Examples of caching in the hierarchy

<table>
<thead>
<tr>
<th>Cache type</th>
<th>What cached</th>
<th>Where cached</th>
<th>Latency (cycles)</th>
<th>Managed by</th>
</tr>
</thead>
<tbody>
<tr>
<td>Registers</td>
<td>4B word</td>
<td>CPU registers</td>
<td>0</td>
<td>Compiler</td>
</tr>
<tr>
<td>TLB</td>
<td>Address translation</td>
<td>On-Chip TLB</td>
<td>0</td>
<td>Hardware</td>
</tr>
<tr>
<td>L1 cache</td>
<td>32B block</td>
<td>On-chip L1</td>
<td>1</td>
<td>Hardware</td>
</tr>
<tr>
<td>L2 cache</td>
<td>32B block</td>
<td>Off-chip L2</td>
<td>10</td>
<td>Hardware</td>
</tr>
<tr>
<td>Virtual memory</td>
<td>4KB page</td>
<td>Main memory</td>
<td>100</td>
<td>Hardware+OS</td>
</tr>
<tr>
<td>Buffer cache</td>
<td>Parts of files</td>
<td>Main memory</td>
<td>100</td>
<td>OS</td>
</tr>
<tr>
<td>Network buffer cache</td>
<td>Parts of files</td>
<td>Local disks</td>
<td>10,000,000</td>
<td>AFS/NFS client</td>
</tr>
<tr>
<td>Browser cache</td>
<td>Web pages</td>
<td>Local disks</td>
<td>10,000,000</td>
<td>Web browser</td>
</tr>
<tr>
<td>Web cache</td>
<td>Web pages</td>
<td>Remote server disk</td>
<td>1,000,000,000</td>
<td>Web proxy server</td>
</tr>
</tbody>
</table>
Summary

- Speed gap between CPU, memory and secondary storage keeps growing
- Well written code exhibits locality
- Memory hierarchy based on caching leverages locality to close the gap