Most of the slides in this lecture are either from or adapted from slides provided by the authors of the textbook “Computer Systems: A Programmer’s Perspective,” 2nd Edition and are provided from the website of Carnegie-Mellon University, course 15-213, taught by Randy Bryant and David O’Hallaron in Fall 2010. These slides are indicated “Supplied by CMU” in the notes section of the slides.
Logical Disk Blocks

- Modern disks present a simpler abstract view of the complex sector geometry:
  - the set of available sectors is modeled as a sequence of b-sized logical blocks (0, 1, 2, ...)
- Mapping between logical blocks and actual (physical) sectors
  - maintained by hardware/firmware device called disk controller
  - converts requests for logical blocks into (surface, track, sector) triples
- Allows controller to set aside spare cylinders for each zone
  - accounts for the difference in “formatted capacity” and “maximum capacity”
Supplied by CMU.
CPU initiates a disk read by writing a command, logical block number, and destination memory address to a port (address) associated with disk controller.
Supplied by CMU.
Reading a Disk Sector (3)

When the DMA transfer completes, the disk controller notifies the CPU with an interrupt (i.e., asserts a special "interrupt" pin on the CPU).

Supplied by CMU.
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

Supplied by CMU.
### SSD Performance Characteristics

<table>
<thead>
<tr>
<th>Operation</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential read</td>
<td>250 MB/s</td>
</tr>
<tr>
<td>Random read</td>
<td>140 MB/s</td>
</tr>
<tr>
<td>Random read access</td>
<td>30 us</td>
</tr>
<tr>
<td>Sequential write</td>
<td>170 MB/s</td>
</tr>
<tr>
<td>Random write</td>
<td>14 MB/s</td>
</tr>
<tr>
<td>Random write access</td>
<td>300 us</td>
</tr>
</tbody>
</table>

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

Supplied by CMU.
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

Supplied by CMU.
Reading a File on a Rotating Disk

- Suppose the data of a file is stored on consecutive disk sectors on one track
  - this is the best possible scenario for reading data quickly
    - single seek required
    - single rotational delay
    - all sectors read in a single scan
Quiz 1

We have two files on the same (rotating) disk. The first file’s data resides in consecutive sectors on one track, the second in consecutive sectors on another track. It takes a total of $t$ seconds to read all of the first file then all of the second file.

Now suppose the files are read concurrently, perhaps a sector of the first, then a sector of the second, then the first, then the second, etc. Compared to reading them sequentially, this will take

a) less time
b) about the same amount of time
c) more time
Quiz 2

You’ve replaced the rotating disk on your computer with a solid-state disk. The data of the two files are again in consecutive locations. Suppose it still takes a total of \( t \) seconds to read the first file then the second. Suppose it took \( u \) seconds to read the two files concurrently on the rotating disk. It takes \( v \) seconds to read them concurrently on the SSD.

\[
\begin{align*}
a) & \; v < u \text{ (faster on the SSD)} \\
b) & \; v \approx u \text{ (about the same)} \\
c) & \; v > u \text{ (slower on the SSD)}
\end{align*}
\]
## 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 (access (ns))</td>
<td>19,200</td>
<td>2,900</td>
<td>320</td>
<td>256</td>
<td>100</td>
<td>75</td>
<td>60</td>
<td>320</td>
</tr>
<tr>
<td></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 (access (ns))</td>
<td>8,000</td>
<td>880</td>
<td>100</td>
<td>30</td>
<td>1</td>
<td>0.1</td>
<td>0.06</td>
<td>130,000</td>
</tr>
<tr>
<td></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>2,000</td>
<td>8,000</td>
<td>125,000</td>
</tr>
</tbody>
</table>

### Disk

<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 (access (ms))</td>
<td>500</td>
<td>100</td>
<td>8</td>
<td>0.30</td>
<td>0.01</td>
<td>0.005</td>
<td>0.005</td>
<td>1,600,000</td>
</tr>
<tr>
<td></td>
<td>87</td>
<td>75</td>
<td>28</td>
<td>10</td>
<td>8</td>
<td>4</td>
<td>3</td>
<td>29</td>
</tr>
<tr>
<td>typical size (MB)</td>
<td>1</td>
<td>10</td>
<td>160</td>
<td>1,000</td>
<td>20,000</td>
<td>160,000</td>
<td>1,500,000</td>
<td>1,500,000</td>
</tr>
</tbody>
</table>

CS33 Intro to Computer Systems  
XVII–13

Supplied by CMU.
<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>CPU</td>
<td>8080</td>
<td>386</td>
<td>Pentium</td>
<td>P-III</td>
<td>P-4</td>
<td>Core 2</td>
<td>Core i7</td>
<td>---</td>
</tr>
<tr>
<td>Clock rate (MHz)</td>
<td>1</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>50</td>
<td>6</td>
<td>1.6</td>
<td>0.3</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>2</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Effective cycle time (ns)</td>
<td>1000</td>
<td>50</td>
<td>6</td>
<td>1.6</td>
<td>0.3</td>
<td>0.25</td>
<td>0.1</td>
<td>10,000</td>
</tr>
</tbody>
</table>
The CPU-Memory Gap

The gap widens between DRAM, disk, and CPU speeds

Supplied by CMU.
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

```c
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**

Supplied by CMU.
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 \( a \)?

```c
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;
}
```

Supplied by CMU.
Quiz 3

Does this function have good locality with respect to array a?
  a) yes
  b) no

```c
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;
}
```

Supplied by CMU.
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

Supplied by CMU.
An Example Memory Hierarchy

- **L0:** CPU registers hold words retrieved from L1 cache
- **L1:** L1 cache holds cache lines retrieved from L2 cache
- **L2:** L2 cache holds cache lines retrieved from main memory
- **L3:** Main memory holds disk blocks retrieved from local disks
- **L4:** Local disks hold files retrieved from disks on remote network servers
- **L5:** Remote secondary storage (distributed file systems, cloud storage)

Supplied by CMU.
Putting Things Into Perspective ...

- Reading from:
  - ... the L1 cache is like grabbing a piece of paper from your desk (3 seconds)
  - ... the L2 cache is picking up a book from a nearby shelf (14 seconds)
  - ... main system memory is taking a 4-minute walk down the hall to talk to a friend
  - ... a hard drive is like leaving the building to roam the earth for one year and three months

This analogy is from http://duartes.org/gustavo/blog/post/what-your-computer-does-while-you-wait (definitely worth reading!).
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**: 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.

Supplied by CMU.
Supplied by CMU.
General Cache Concepts: Hit

<table>
<thead>
<tr>
<th>Cache</th>
<th>8</th>
<th>9</th>
<th>14</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
</tbody>
</table>

Request: 14

Data in block b is needed

Block b is in cache:
Hit!

Supplied by CMU.
General Cache Concepts: Miss

- Request: 12
  - Cache
    - Block b is not in cache: Miss!
    - Block b is fetched from memory
    - Data in block b is needed
  - Memory
    - Block b is stored in cache
      - placement policy: determines where b goes
      - replacement policy: determines which block gets evicted (victim)

Supplied by CMU.
General Caching Concepts:
Types of Cache Misses

- **Cold (compulsory) 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 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

- **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 is Cached?</th>
<th>Where is Cached?</th>
<th>Latency (cycles)</th>
<th>Managed By</th>
</tr>
</thead>
<tbody>
<tr>
<td>Registers</td>
<td>4-8 bytes words</td>
<td>CPU core</td>
<td>0</td>
<td>Compiler</td>
</tr>
<tr>
<td>TLB</td>
<td>Address translations</td>
<td>On-Chip TLB</td>
<td>0</td>
<td>Hardware</td>
</tr>
<tr>
<td>L1 cache</td>
<td>64-bytes block</td>
<td>On-Chip L1</td>
<td>1</td>
<td>Hardware</td>
</tr>
<tr>
<td>L2 cache</td>
<td>64-bytes block</td>
<td>On/Off-Chip L2</td>
<td>10</td>
<td>Hardware</td>
</tr>
<tr>
<td>Virtual Memory</td>
<td>4-KB 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>Disk cache</td>
<td>Disk sectors</td>
<td>Disk controller</td>
<td>100,000</td>
<td>Disk firmware</td>
</tr>
<tr>
<td>Network buffer cache</td>
<td>Parts of files</td>
<td>Local disk</td>
<td>10,000,000</td>
<td>CIFS/NFS client</td>
</tr>
<tr>
<td>Browser cache</td>
<td>Web pages</td>
<td>Local disk</td>
<td>10,000,000</td>
<td>Web browser</td>
</tr>
<tr>
<td>Web cache</td>
<td>Web pages</td>
<td>Remote server disks</td>
<td>1,000,000,000</td>
<td>Web proxy server</td>
</tr>
</tbody>
</table>

Supplied by CMU.
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