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.

Static RAM (SRAM)
- Each cell stores a bit with a four or six-transistor circuit.
- Retains value indefinitely, as long as it is kept powered.
- Relatively insensitive to electrical noise (EMI), radiation, etc.
- Faster and more expensive than DRAM.

Dynamic RAM (DRAM)
- Each cell stores bit with a capacitor. One transistor is used for access
- Value must be refreshed every 10-100 ms.
- More sensitive to disturbances (EMI, radiation,...) than SRAM.
- Slower and cheaper than SRAM.
## SRAM vs DRAM Summary

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SRAM</td>
<td>4 or 6</td>
<td>1X</td>
<td>No</td>
<td>Maybe</td>
<td>100x</td>
<td>Cache memories</td>
</tr>
<tr>
<td>DRAM</td>
<td>1</td>
<td>10X</td>
<td>Yes</td>
<td>Yes</td>
<td>1X</td>
<td>Main memories, frame buffers</td>
</tr>
</tbody>
</table>

EDC, ECC = error detecting codes, error correcting codes
Conventional DRAM Organization

- **d x w DRAM:**
  - dw total bits organized as d supercells of size w bits
Reading DRAM Supercell (2,1)

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 column 1.
Step 2(b): Supercell (2,1) copied from buffer to data lines, and eventually back to the CPU.
Memory Modules

64 MB memory module consisting of eight 8Mx8 DRAMs

addr (row = i, col = j)

64-bit doubleword at main memory address A

Memory controller

64-bit doubleword

: supercell (i,j)
Enhanced DRAMs

- Basic DRAM cell has not changed since its invention in 1966.
  - Commercialized by Intel in 1970.
- DRAM cores with better interface logic and faster I/O:
  - Synchronous DRAM (SDRAM)
    - Uses a conventional clock signal instead of asynchronous control
    - Allows reuse of the row addresses (e.g., RAS, CAS, CAS, CAS)
  - Double data-rate synchronous DRAM (DDR SDRAM)
    - Double edge clocking sends two bits per cycle per pin
    - Different types distinguished by size of small prefetch buffer:
      - DDR (2 bits), DDR2 (4 bits), DDR3 (8 bits), DDR4 (16 bits)
    - By 2010, standard for most server and desktop systems
    - Intel Core cpus (i3, i5, i7) support only DDR3, DDR4 SDRAM
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
  - Erasable PROM (EPROM): can be bulk erased (UV, X-Ray)
  - Electrically erasable PROM (EEPROM): electronic erase capability
  - Flash memory: EEPROMs with partial (block) erase capability
    - Wears out, erase life short depending on flash technology.
      - NAND SLC, MLC, TLC, QLC, etc. progressively shorter life

- Uses for Nonvolatile Memories
  - Firmware programs stored in a ROM (BIOS, controllers for disks, network cards, graphics accelerators, security subsystems,...)
  - Solid state disks (replace rotating disks in thumb drives, smart phones, mp3 players, tablets, laptops,...)
  - Disk caches
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: `movl A, %eax`
Memory Read Transaction (2)

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

Load operation: \texttt{movl A, %eax}
Memory Read Transaction (3)

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

Load operation: `movl A, %eax`
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**: `movl %eax, A`

Diagram:
- CPU
- Register file
- ALU
- Bus interface
- I/O bridge
- Main memory
Memory Write Transaction (2)

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

Store operation: `movl %eax, A`
Memory Write Transaction (3)

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

 Store operation: `movl %eax, A`
What’s Inside A Disk Drive?

- Spindle
- Arm
- Actuator
- Platters
- Electronics (including a processor and memory!)
- SCSI connector

Image courtesy of Seagate Technology
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**.
Disk Geometry (Multiple-Platter View)

- Aligned tracks form a cylinder.

![Diagram showing disk geometry with multiple platters and surfaces aligned to form a cylinder.](image-url)
Disk Capacity

- **Capacity**: maximum number of bits that can be stored.
  - Vendors express capacity in units of gigabytes (GB), where 1 GB = $10^9$ Bytes (Lawsuit pending! Claims deceptive advertising).

- **Capacity is determined by these technology factors:**
  - **Recording density** (bits/in): number of bits that can be squeezed into a 1 inch segment of a track.
  - **Track density** (tracks/in): number of tracks that can be squeezed into a 1 inch radial segment.
  - **Areal density** (bits/in²): product of recording and 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.
A human hair is ~ 100 microns for comparison

A portion of a disk track. Two sectors are illustrated.
A disk with five zones. Each zone has many tracks.
Computing Disk Capacity

Capacity = (# bytes/sector) \times (\text{avg. # sectors/track}) \times \\
(\text{# tracks/surface}) \times (\text{# surfaces/platter}) \times \\
(\text{# platters/disk})

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

Capacity = 512 \times 300 \times 20000 \times 2 \times 5
\quad = 30,720,000,000
\quad = 30.72 \text{ GB}
Disk Operation (Single-Platter View)

The disk surface spins at a fixed rotational rate.

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.
Disk Operation (Multi-Platter View)

Read/write heads move in unison from cylinder to cylinder

Spindle

Arm
Disk Structure - top view of single platter

Surface organized into tracks

Tracks divided into sectors
Disk Access

Head in position above a track
Disk Access

Rotation is counter-clockwise
Disk Access – Read

About to read blue sector
Disk Access – Read

After BLUE read

After reading blue sector
Disk Access – Read

After BLUE read

Red request scheduled next
Disk Access – Seek

After BLUE read

Seek for RED

Seek to red’s track
Disk Access – Rotational Latency

After BLUE read

Seek for RED

Rotational latency

Wait for red sector to rotate around
Disk Access – Read

- After BLUE read
- Seek for RED
- Rotational latency
- After RED read

Complete read of red
Disk Access – Service Time Components

After BLUE read

Seek for RED

Rotational latency

After RED read

Data transfer

Seek

Rotational latency

Data transfer
Disk Access Time

- **Average time to access some target sector approximated by:**
  - $T_{access} = T_{avg \ seek} + T_{avg \ rotation} + T_{avg \ transfer}$

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

- **Rotational latency** ($T_{avg \ rotation}$)
  - Time waiting for first bit of target sector to pass under r/w head.
  - $T_{avg \ rotation} = 1/2 \times 1/RPMs \times 60 \ sec/1 \ min$
  - Typical $T_{avg \ rotation} = 7200 \ RPMs$

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

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

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

- **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 then DRAM.
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 \((0, 1, 2, \ldots)\)

- 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”.
I/O Bus

Expansion slots for other devices such as network adapters.
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

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential read tput</td>
<td>250 MB/s</td>
<td>Sequential write tput</td>
<td>170 MB/s</td>
</tr>
<tr>
<td>Random read tput</td>
<td>140 MB/s</td>
<td>Random write tput</td>
<td>14 MB/s</td>
</tr>
<tr>
<td>Rand read access</td>
<td>30 us</td>
<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)
  - 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
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 (October 2013 about 75 times more expensive $69.99 vs $5,288.36 MLC -1TB)
  ▪ Today in late 2015 about 7 times more ($49.99 vs $369.99)

■ Applications
  ▪ MP3 players, smart phones, laptops, desktops
  ▪ Beginning to take over in servers and storage systems
### 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>2,900</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>
<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>

#### 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>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>access (ms)</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</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 (ms)</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>
## CPU Clock Rates

<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>

Inflection point in computer history when designers hit the “Power Wall”
The CPU-Memory Gap

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

The graph illustrates the trend in access times for different storage and processing technologies from 1980 to 2010. The y-axis represents time in nanoseconds (ns), and the x-axis represents years from 1980 to 2010.

- **Disk** seek time
- **Flash SSD** access time
- **DRAM** access time
- **SRAM** access time
- **CPU** cycle time
- **Effective CPU cycle time**

As the years progress, the access times for disk and flash SSD decrease, reflecting technological advancements. The graph highlights the widening gap in access times between these technologies and the CPU, emphasizing the challenge of the CPU-memory gap.
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.

- **Instruction references**
  - Reference instructions in sequence.
  - Cycle through loop repeatedly.
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;
}
```
Question: Does this function have good locality with respect to array $a[][]$?
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)?

```c
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 complement each other beautifully.

- They suggest an approach for organizing memory and storage systems known as a memory hierarchy.
An Example Memory Hierarchy (modified)

- **L0**: Registers
  - CPU registers hold words retrieved from L1 cache

- **L1**: L1 cache (SRAM)
  - L1 cache holds cache lines retrieved from L2 cache
  - L1 cache holds cache lines retrieved from main memory

- **L2**: L2 cache (SRAM)
  - L2 cache holds cache lines retrieved from main memory
  - Main memory holds disk blocks retrieved from local disks

- **L3**: Remote secondary storage (local disks)
  - Flash Cache extends DRAM and is much faster than local disk
  - Local disks hold files retrieved from disks on remote network servers

- **L3A**: Remote secondary storage (tapes, distributed file systems, Web 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**: 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.

Larger, slower, cheaper memory viewed as partitioned into “blocks”.

Cache

Memory

Data is copied in block-sized transfer units.
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: 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 it 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 +</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>AFS/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>
## Intel i7 Cache Implementations

### Translation Lookaside Buffers

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Instruction TLB</th>
<th>Data DLB</th>
<th>Second-level TLB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>128</td>
<td>64</td>
<td>512</td>
</tr>
<tr>
<td>Associativity</td>
<td>4-way</td>
<td>4-way</td>
<td>4-way</td>
</tr>
<tr>
<td>Replacement</td>
<td>Pseudo-LRU</td>
<td>Pseudo-LRU</td>
<td>Pseudo-LRU</td>
</tr>
<tr>
<td>Access latency</td>
<td>1 cycle</td>
<td>1 cycle</td>
<td>6 cycles</td>
</tr>
<tr>
<td>Miss</td>
<td>7 cycles</td>
<td>7 cycles</td>
<td>Hundreds of cycles to access page table</td>
</tr>
</tbody>
</table>

### Instruction – Data Caches

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>L1</th>
<th>L2</th>
<th>L3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>32 KB I/32 KB D</td>
<td>256 KB</td>
<td>2 MB per core</td>
</tr>
<tr>
<td>Associativity</td>
<td>4-way I/8-way D</td>
<td>8-way</td>
<td>16-way</td>
</tr>
<tr>
<td>Access latency</td>
<td>4 cycles, pipelined</td>
<td>10 cycles</td>
<td>35 cycles</td>
</tr>
<tr>
<td>Replacement scheme</td>
<td>Pseudo-LRU</td>
<td>Pseudo-LRU</td>
<td>Pseudo-LRU but with an ordered selection algorithm</td>
</tr>
</tbody>
</table>

---
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.