The Memory Hierarchy
## Memory Technologies

- Technologies have vastly different tradeoffs between capacity, latency, bandwidth, energy, and cost.

<table>
<thead>
<tr>
<th>Memory Technology</th>
<th>Capacity</th>
<th>Latency</th>
<th>Cost/GB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register</td>
<td>100s of bits</td>
<td>20 ps</td>
<td>$$$$</td>
</tr>
<tr>
<td>SRAM</td>
<td>~10 KB-10 MB</td>
<td>1-10 ns</td>
<td>~$1000</td>
</tr>
<tr>
<td>DRAM</td>
<td>~10 GB</td>
<td>80 ns</td>
<td>~$10</td>
</tr>
<tr>
<td>Flash*</td>
<td>~100 GB</td>
<td>100 us</td>
<td>~$1</td>
</tr>
<tr>
<td>Hard disk*</td>
<td>~1 TB</td>
<td>10 ms</td>
<td>~$0.1</td>
</tr>
</tbody>
</table>

*non-volatile (retains contents when powered off)*
### Memory Technologies

- Technologies have vastly different tradeoffs between capacity, latency, bandwidth, energy, and cost
  - ... and thus different applications

<table>
<thead>
<tr>
<th></th>
<th>Capacity</th>
<th>Latency</th>
<th>Cost/GB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register</td>
<td>100s of bits</td>
<td>20 ps</td>
<td>$$$$$$$</td>
</tr>
<tr>
<td>SRAM</td>
<td>~10 KB-10 MB</td>
<td>1-10 ns</td>
<td>~$1000</td>
</tr>
<tr>
<td>DRAM</td>
<td>~10 GB</td>
<td>80 ns</td>
<td>~$10</td>
</tr>
<tr>
<td>Flash*</td>
<td>~100 GB</td>
<td>100 us</td>
<td>~$1</td>
</tr>
<tr>
<td>Hard disk*</td>
<td>~1 TB</td>
<td>10 ms</td>
<td>~$0.1</td>
</tr>
</tbody>
</table>

* non-volatile (retains contents when powered off)
Memory Technologies

- Technologies have vastly different tradeoffs between capacity, latency, bandwidth, energy, and cost
  - ... and thus different applications

<table>
<thead>
<tr>
<th>Memory Technology</th>
<th>Capacity</th>
<th>Latency</th>
<th>Cost/GB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register</td>
<td>100s of bits</td>
<td>20 ps</td>
<td>$$$$$</td>
</tr>
<tr>
<td>SRAM</td>
<td>~10 KB-10 MB</td>
<td>1-10 ns</td>
<td>~$1000</td>
</tr>
<tr>
<td>DRAM</td>
<td>~10 GB</td>
<td>80 ns</td>
<td>~$10</td>
</tr>
<tr>
<td>Flash*</td>
<td>~100 GB</td>
<td>100 us</td>
<td>~$1</td>
</tr>
<tr>
<td>Hard disk*</td>
<td>~1 TB</td>
<td>10 ms</td>
<td>~$0.1</td>
</tr>
</tbody>
</table>

* non-volatile (retains contents when powered off)
Memory Technologies

- Technologies have vastly different tradeoffs between capacity, latency, bandwidth, energy, and cost
  - ... and thus different applications

<table>
<thead>
<tr>
<th></th>
<th>Capacity</th>
<th>Latency</th>
<th>Cost/GB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register</td>
<td>100s of bits</td>
<td>20 ps</td>
<td>$$$$$</td>
</tr>
<tr>
<td>SRAM</td>
<td>~10 KB-10 MB</td>
<td>1-10 ns</td>
<td>~$1000</td>
</tr>
<tr>
<td>DRAM</td>
<td>~10 GB</td>
<td>80 ns</td>
<td>~$10</td>
</tr>
<tr>
<td>Flash*</td>
<td>~100 GB</td>
<td>100 us</td>
<td>~$1</td>
</tr>
<tr>
<td>Hard disk*</td>
<td>~1 TB</td>
<td>10 ms</td>
<td>~$0.1</td>
</tr>
</tbody>
</table>

* non-volatile (retains contents when powered off)
Memory Technologies

- Technologies have vastly different tradeoffs between capacity, latency, bandwidth, energy, and cost
  - ... and thus different applications

<table>
<thead>
<tr>
<th></th>
<th>Capacity</th>
<th>Latency</th>
<th>Cost/GB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register</td>
<td>100s of bits</td>
<td>20 ps</td>
<td>$$$$$</td>
</tr>
<tr>
<td>SRAM</td>
<td>~10 KB-10 MB</td>
<td>1-10 ns</td>
<td>~$1000</td>
</tr>
<tr>
<td>DRAM</td>
<td>~10 GB</td>
<td>80 ns</td>
<td>~$10</td>
</tr>
<tr>
<td>Flash*</td>
<td>~100 GB</td>
<td>100 us</td>
<td>~$1</td>
</tr>
<tr>
<td>Hard disk*</td>
<td>~1 TB</td>
<td>10 ms</td>
<td>~$0.1</td>
</tr>
</tbody>
</table>

* non-volatile (retains contents when powered off)
Memory Technologies: SRAM, DRAM, Flash, Hard Disk

NOTE: Demystification, will not be on the quiz
Static RAM (SRAM)

Address decoder

8x6 SRAM array

Drivers

Sense amplifiers
Static RAM (SRAM)

Drivers

Address decoder

Address

3

8x6 SRAM array

Sense amplifiers
Static RAM (SRAM)

Drivers

Wordlines (horizontal)

Sense amplifiers

Address decoder

8x6 SRAM array
Static RAM (SRAM)

Drivers

Sense amplifiers

SRAM cell

Address decoder

Address 3

8x6 SRAM array
Static RAM (SRAM)

8x6 SRAM array

Drivers
SRAM cell
Wordlines (horizontal)
Bitlines (vertical)
Sense amplifiers

Address decoder

Address 3

October 29, 2019
MIT 6.004 Fall 2019
Static RAM (SRAM)

Drivers

SRAM cell

Wordlines (horizontal)

Bitlines (vertical)

Sense amplifiers

Address decoder

8x6 SRAM array

Address 3

Data out 6
Static RAM (SRAM)

Drivers

SRAM cell

Wordlines (horizontal)

Bitlines (vertical)

Sense amplifiers

8x6 SRAM array

Address decoder

Data in

6

3

Address

Data out

6

October 29, 2019

MIT 6.004 Fall 2019
SRAM Cell

6-transistor (6T) cell:
SRAM Cell

6-transistor (6T) cell:
- Two CMOS inverters (4 FETs) forming a bistable element

Bistable element (two stable states) stores a single bit
- Vdd “1”
- GND “0”
SRAM Cell

6-transistor (6T) cell:
- Two CMOS inverters (4 FETs) forming a bistable element
- Two access transistors

Bistable element (two stable states) stores a single bit

Vdd
GND "1"

GND
Vdd "0"
SRAM Read

6T SRAM Cell

access FETs

wordline

bitline

October 29, 2019
1. Drivers precharge all bitlines to Vdd (1), and leave them floating.
1. Drivers precharge all bitlines to Vdd (1), and leave them floating.

2. Address decoder activates one wordline.
1. Drivers precharge all bitlines to Vdd (1), and leave them floating.
2. Address decoder activates one wordline.
3. Each cell in the activated word slowly pulls down one of the bitlines to GND (0).
SRAM Read

1. Drivers precharge all bitlines to Vdd (1), and leave them floating.

2. Address decoder activates one wordline.

3. Each cell in the activated word slowly pulls down one of the bitlines to GND (0).

4. Sense amplifiers sense change in bitline voltages, produce output data.
SRAM Write
1. Drivers set and hold bitlines to desired values (Vdd and GND for 1, GND and Vdd for 0)
SRAM Write

1. Drivers set and hold bitlines to desired values (Vdd and GND for 1, GND and Vdd for 0)

2. Address decoder activates one wordline
SRAM Write

1. Drivers set and hold bitlines to desired values (Vdd and GND for 1, GND and Vdd for 0)
2. Address decoder activates one wordline
3. Each cell in word is overpowered by the drivers, stores value
1. Drivers set and hold bitlines to desired values (Vdd and GND for 1, GND and Vdd for 0)

2. Address decoder activates one wordline

3. Each cell in word is overpowered by the drivers, stores value

Cell transistors are carefully sized so that bitline GND overpowers cell Vdd, but bitline Vdd does not overpower cell GND
Multiported SRAMs

- SRAM so far can do either one read or one write/cycle
- We can do multiple reads and writes with multiple ports by adding one set of wordlines and bitlines per port
Multiported SRAMs

- SRAM so far can do either one read or one write/cycle
- We can do multiple reads and writes with multiple ports by adding one set of wordlines and bitlines per port
Multiported SRAMs

- SRAM so far can do either one read or one write/cycle
- We can do multiple reads and writes with multiple ports by adding one set of wordlines and bitlines per port
Multiported SRAMs

- SRAM so far can do either one read or one write/cycle
- We can do multiple reads and writes with multiple ports by adding one set of wordlines and bitlines per port
Multiported SRAMs

- SRAM so far can do either one read or one write/cycle
- We can do multiple reads and writes with multiple ports by adding one set of wordlines and bitlines per port
Multiported SRAMs

- SRAM so far can do either one read or one write/cycle
- We can do multiple reads and writes with multiple ports by adding one set of wordlines and bitlines per port

- Cost/bit for $N$ ports?
  - Wordlines?
  - Bitlines?
  - Access FETs?
Multiported SRAMs

- SRAM so far can do either one read or one write/cycle
- We can do multiple reads and writes with multiple ports by adding one set of wordlines and bitlines per port

- **Cost/bit for N ports?**
  - Wordlines? $N$
  - Bitlines?
  - Access FETs?
Multiported SRAMs

- SRAM so far can do either one read or one write/cycle
- We can do multiple reads and writes with multiple ports by adding one set of wordlines and bitlines per port

- Cost/bit for N ports?
  - Wordlines? $N$
  - Bitlines? $2N$
  - Access FETs?
Multiported SRAMs

- SRAM so far can do either one read or one write/cycle
- We can do multiple reads and writes with multiple ports by adding one set of wordlines and bitlines per port

**Cost/bit for N ports?**
- **Wordlines?** $N$
- **Bitlines?** $2*N$
- **Access FETs?** $2*N$
Multiported SRAMs

- SRAM so far can do either one read or one write/cycle
- We can do multiple reads and writes with multiple ports by adding one set of wordlines and bitlines per port

- **Cost/bit for N ports?**
  - Wordlines? \( N \)
  - Bitlines? \( 2*N \)
  - Access FETs? \( 2*N \)

- Wires dominate area \( \rightarrow O(N^2) \) area!
Summary: SRAMs

- Array of $k \times b$ cells ($k$ words, $b$ cells per word)
- Cell is a bistable element + access transistors
  - Analog circuit with carefully sized transistors
- Read: Precharge bitlines, activate wordline, sense
- Write: Drive bitlines, activate wordline, overpower cells
Summary: SRAMs

- Array of $k \times b$ cells ($k$ words, $b$ cells per word)
- Cell is a bistable element + access transistors
  - Analog circuit with carefully sized transistors
- Read: Precharge bitlines, activate wordline, sense
- Write: Drive bitlines, activate wordline, overpower cells

- 6 FETs/cell... can we do better?
1T Dynamic RAM (DRAM) Cell

- Wordline
- Bitline
- Access FET
1T Dynamic RAM (DRAM) Cell

- Storage capacitor
- 1T DRAM Cell
- Wordline
- Access FET
- Bitline
1T Dynamic RAM (DRAM) Cell

- Storage capacitor
- 1T DRAM Cell
  - Wordline
  - Access FET
  - Bitline

Trench capacitors take little area

Cyferz (CC BY 2.5)
1T Dynamic RAM (DRAM) Cell

Storage capacitor

1T DRAM Cell

V_{REF}

wordline

access FET

bitline

Trench capacitors take little area

✓ ~20x smaller area than SRAM cell → Denser and cheaper!
1T Dynamic RAM (DRAM) Cell

✓ ~20x smaller area than SRAM cell → Denser and cheaper!
✗ Problem: Capacitor leaks charge, must be refreshed periodically (~milliseconds)
DRAM Writes and Reads

- **Writes**: Drive bitline to Vdd or GND, activate wordline, charge or discharge capacitor
DRAM Writes and Reads

- Writes: Drive bitline to Vdd or GND, activate wordline, charge or discharge capacitor

- Reads:

![Diagram of 1T DRAM Cell]
DRAM Writes and Reads

- **Writes:** Drive bitline to Vdd or GND, activate wordline, charge or discharge capacitor

- **Reads:**
  1. Precharge bitline to Vdd/2
  2. Activate wordline
### DRAM Writes and Reads

- **Writes:** Drive bitline to Vdd or GND, activate wordline, charge or discharge capacitor

- **Reads:**
  1. Precharge bitline to Vdd/2
  2. Activate wordline
  3. Capacitor and bitline share charge
     - If capacitor was discharged, bitline voltage decreases slightly
     - If capacitor was charged, bitline voltage increases slightly
  4. Sense bitline to determine if 0 or 1
DRAM Writes and Reads

- **Writes:** Drive bitline to Vdd or GND, activate wordline, charge or discharge capacitor

- **Reads:**
  1. Precharge bitline to Vdd/2
  2. Activate wordline
  3. Capacitor and bitline share charge
     - If capacitor was discharged, bitline voltage decreases slightly
     - If capacitor was charged, bitline voltage increases slightly
  4. Sense bitline to determine if 0 or 1

- **Issue:** Reads are **destructive!** (charge is gone!)
DRAM Writes and Reads

- **Writes:** Drive bitline to Vdd or GND, activate wordline, charge or discharge capacitor

- **Reads:**
  1. Precharge bitline to Vdd/2
  2. Activate wordline
  3. Capacitor and bitline share charge
     - If capacitor was discharged, bitline voltage decreases slightly
     - If capacitor was charged, bitline voltage increases slightly
  4. Sense bitline to determine if 0 or 1

- **Issue:** **Reads are destructive!** (charge is gone!)
  - Data must be rewritten to cells at end of read
Non-Volatile Storage: Flash

Flash Memory: Use “floating gate” transistors to store charge (floating gate is a well insulated conductor)

Electrons here diminish strength of field from control gate ⇒ no inversion ⇒ NFET stays off even when word line is high.

Cyferz (CC BY 2.5)
Non-Volatile Storage: Flash

Flash Memory: Use “floating gate” transistors to store charge (floating gate is a well insulated conductor)
- **Very dense:** Multiple bits/transistor, read and written in blocks

Electrons here diminish strength of field from control gate ⇒ no inversion ⇒ NFET stays off even when word line is high.

Cyferz (CC BY 2.5)
Non-Volatile Storage: Flash

Flash Memory: Use “floating gate” transistors to store charge (floating gate is a well insulated conductor)

- Very dense: Multiple bits/transistor, read and written in blocks
- Slow (especially on writes), 10-100 us

Electrons here diminish strength of field from control gate ⇒ no inversion ⇒ NFET stays off even when word line is high.

Cyferz (CC BY 2.5)
Non-Volatile Storage: Flash

Flash Memory: Use “floating gate” transistors to store charge (floating gate is a well insulated conductor)

- **Very dense**: Multiple bits/transistor, read and written in blocks
- **Slow** (especially on writes), 10-100 us
- **Limited number of writes**: charging/discharging the floating gate (writes) requires large voltages that damage transistor

Electrons here diminish strength of field from control gate ⇒ no inversion ⇒ NFET stays off even when word line is high.

Cyferz (CC BY 2.5)
Non-Volatile Storage: Hard Disk

Hard Disk: Rotating magnetic platters + read/write head

Surachit (CC BY 2.5)
Non-Volatile Storage: Hard Disk

Hard Disk: Rotating magnetic platters + read/write head
- Extremely slow (~10ms): Mechanically move head to position, wait for data to pass underneath head
Hard Disk: Rotating magnetic platters + read/write head

- **Extremely slow** (~10ms): Mechanically move head to position, wait for data to pass underneath head
- ~100MB/s for sequential read/writes
Non-Volatile Storage: Hard Disk

Hard Disk: Rotating magnetic platters + read/write head

- **Extremely slow** (~10ms): Mechanically move head to position, wait for data to pass underneath head
- ~100MB/s for sequential read/writes
- ~100KB/s for random read/writes
Non-Volatile Storage: Hard Disk

Hard Disk: Rotating magnetic platters + read/write head
- Extremely slow (~10ms): Mechanically move head to position, wait for data to pass underneath head
- ~100MB/s for sequential read/writes
- ~100KB/s for random read/writes
- Cheap
The Memory Hierarchy
### Summary: Memory Technologies

<table>
<thead>
<tr>
<th>Technology</th>
<th>Capacity</th>
<th>Latency</th>
<th>Cost/GB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register</td>
<td>100s of bits</td>
<td>20 ps</td>
<td>$$$$$</td>
</tr>
<tr>
<td>SRAM</td>
<td>~10 KB-10 MB</td>
<td>1-10 ns</td>
<td>~$1000</td>
</tr>
<tr>
<td>DRAM</td>
<td>~10 GB</td>
<td>80 ns</td>
<td>~$10</td>
</tr>
<tr>
<td>Flash</td>
<td>~100 GB</td>
<td>100 us</td>
<td>~$1</td>
</tr>
<tr>
<td>Hard disk</td>
<td>~1 TB</td>
<td>10 ms</td>
<td>~$0.1</td>
</tr>
</tbody>
</table>

- Different technologies have vastly different tradeoffs
Different technologies have vastly different tradeoffs.

Size is a **fundamental limit**, even setting cost aside:
- Small + low latency, high bandwidth, low energy, or
- Large + high-latency, low bandwidth, high energy

<table>
<thead>
<tr>
<th></th>
<th>Capacity</th>
<th>Latency</th>
<th>Cost/GB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register</td>
<td>100s of bits</td>
<td>20 ps</td>
<td>$$$$$</td>
</tr>
<tr>
<td>SRAM</td>
<td>~10 KB-10 MB</td>
<td>1-10 ns</td>
<td>~$1000</td>
</tr>
<tr>
<td>DRAM</td>
<td>~10 GB</td>
<td>80 ns</td>
<td>~$10</td>
</tr>
<tr>
<td>Flash</td>
<td>~100 GB</td>
<td>100 us</td>
<td>~$1</td>
</tr>
<tr>
<td>Hard disk</td>
<td>~1 TB</td>
<td>10 ms</td>
<td>~$0.1</td>
</tr>
</tbody>
</table>
Summary: Memory Technologies

Different technologies have vastly different tradeoffs

Size is a fundamental limit, even setting cost aside:
  - Small + low latency, high bandwidth, low energy, or
  - Large + high-latency, low bandwidth, high energy

Can we get best of both worlds? (large, fast, cheap)

<table>
<thead>
<tr>
<th></th>
<th>Capacity</th>
<th>Latency</th>
<th>Cost/GB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register</td>
<td>100s of bits</td>
<td>20 ps</td>
<td>$$$</td>
</tr>
<tr>
<td>SRAM</td>
<td>~10 KB-10 MB</td>
<td>1-10 ns</td>
<td>~$1000</td>
</tr>
<tr>
<td>DRAM</td>
<td>~10 GB</td>
<td>80 ns</td>
<td>~$10</td>
</tr>
<tr>
<td>Flash</td>
<td>~100 GB</td>
<td>100 us</td>
<td>~$1</td>
</tr>
<tr>
<td>Hard disk</td>
<td>~1 TB</td>
<td>10 ms</td>
<td>~$0.1</td>
</tr>
</tbody>
</table>
The Memory Hierarchy

Want large, fast, and cheap memory, but...
- Large memories are slow (even if built with fast components)
- Fast memories are expensive

Solution: Use a hierarchy of memories with different tradeoffs to fake a large, fast, cheap memory

- CPU
- Mem
- Mem
- Mem

≈

<table>
<thead>
<tr>
<th>Speed:</th>
<th>Fastest</th>
<th>Slowest</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capacity:</td>
<td>Smallest</td>
<td>Largest</td>
</tr>
<tr>
<td>Cost:</td>
<td>Highest</td>
<td>Lowest</td>
</tr>
</tbody>
</table>

Fast
Large
Cheap
Memory Hierarchy Interface

Approach 1: Expose Hierarchy

- Registers, SRAM, DRAM, Flash, Hard Disk each available as storage alternatives
- Tell programmers: “Use them cleverly”
Approach 1: Expose Hierarchy
- Registers, SRAM, DRAM, Flash, Hard Disk each available as storage alternatives
- Tell programmers: “Use them cleverly”

Approach 2: Hide Hierarchy
- Programming model: Single memory, single address space
- Machine transparently stores data in fast or slow memory, depending on usage patterns
Memory Hierarchy Interface

Approach 1: Expose Hierarchy

- Registers, SRAM, DRAM, Flash, Hard Disk each available as storage alternatives
- Tell programmers: “Use them cleverly”

Approach 2: Hide Hierarchy

- Programming model: Single memory, single address space
- Machine transparently stores data in fast or slow memory, depending on usage patterns
Memory Hierarchy Interface

Approach 1: Expose Hierarchy
- Registers, SRAM, DRAM, Flash, Hard Disk each available as storage alternatives
- Tell programmers: “Use them cleverly”

Approach 2: Hide Hierarchy
- Programming model: Single memory, single address space
- Machine transparently stores data in fast or slow memory, depending on usage patterns
Memory Hierarchy Interface

Approach 1: Expose Hierarchy
- Registers, SRAM, DRAM, Flash, Hard Disk each available as storage alternatives
- Tell programmers: “Use them cleverly”

Approach 2: Hide Hierarchy
- Programming model: Single memory, single address space
- Machine transparently stores data in fast or slow memory, depending on usage patterns
Memory Hierarchy Interface

Approach 1: Expose Hierarchy

- Registers, SRAM, DRAM, Flash, Hard Disk each available as storage alternatives
- Tell programmers: “Use them cleverly”

Approach 2: Hide Hierarchy

- Programming model: Single memory, single address space
- Machine transparently stores data in fast or slow memory, depending on usage patterns
Typical Memory Access Patterns

\[ \text{address} \]

\[ \text{time} \]
Typical Memory Access Patterns
Typical Memory Access Patterns

address

time

code

loop
Typical Memory Access Patterns

- Stack
- Code
- Loop

Address vs. Time
Typical Memory Access Patterns

address

stack

local variable accesses

code

loop

time
Typical Memory Access Patterns

- address
- stack
- code
- time

- local variable accesses
- procedure calls
- loop
Typical Memory Access Patterns

- address
- data
- stack
- code
- time

- local variable accesses
- procedure calls
- loop
Typical Memory Access Patterns

- **Address access**
- **Data access**
- **Stack access**
- **Code access**

**Time**

- Array accesses
- Local variable accesses
- Procedure calls
- Loop access
Common Predictable Patterns

- Two predictable properties of memory accesses:
  - **Temporal locality**: If a location has been accessed recently, it is likely to be accessed (reused) soon
  - **Spatial locality**: If a location has been accessed recently, it is likely that nearby locations will be accessed soon
Caches

- Cache: A small, interim storage component that transparently retains (caches) data from recently accessed locations
Caches

- Cache: A small, interim storage component that transparently retains (caches) data from recently accessed locations.

  ![Diagram showing CPU, Cache, and Main Memory with bidirectional arrows for Address and Data]

- Processor sends accesses to cache. Two options:
Caches

- Cache: A small, interim storage component that transparently retains (caches) data from recently accessed locations

- Processor sends accesses to cache. Two options:
  - **Cache hit**: Data for this address in cache, returned quickly
Caches

- Cache: A small, interim storage component that transparently retains (caches) data from recently accessed locations

- Processor sends accesses to cache. Two options:
  - Cache hit: Data for this address in cache, returned quickly
  - Cache miss: Data not in cache
    - Fetch data from memory, send it back to processor
    - Retain this data in the cache (replacing some other data)
Caches

- **Cache**: A small, interim storage component that transparently retains (caches) data from recently accessed locations

- **Processor sends accesses to cache. Two options:**
  - **Cache hit**: Data for this address in cache, returned quickly
  - **Cache miss**: Data not in cache
    - Fetch data from memory, send it back to processor
    - Retain this data in the cache (replacing some other data)
  - Processor must deal with variable memory access time
A Typical Memory Hierarchy

Computers use many levels of caches:

<table>
<thead>
<tr>
<th>On the CPU</th>
<th>Access time</th>
<th>Capacity</th>
<th>Managed By</th>
</tr>
</thead>
<tbody>
<tr>
<td>Registers</td>
<td>1 cycle</td>
<td>1 KB</td>
<td>Software/Compiler</td>
</tr>
<tr>
<td>Level 1 Cache</td>
<td>2-4 cycles</td>
<td>32 KB</td>
<td>Hardware</td>
</tr>
<tr>
<td>Level 2 Cache</td>
<td>10 cycles</td>
<td>256 KB</td>
<td>Hardware</td>
</tr>
<tr>
<td>Level 3 Cache</td>
<td>40 cycles</td>
<td>10 MB</td>
<td>Hardware</td>
</tr>
<tr>
<td>Main Memory</td>
<td>200 cycles</td>
<td>10 GB</td>
<td>Software/OS</td>
</tr>
<tr>
<td>Flash Drive</td>
<td>10-100us</td>
<td>100 GB</td>
<td>Software/OS</td>
</tr>
<tr>
<td>Hard Disk</td>
<td>10ms</td>
<td>1 TB</td>
<td>Software/OS</td>
</tr>
</tbody>
</table>
Cache Metrics

- **Hit Ratio:**
  \[ HR = \frac{\text{hits}}{\text{hits} + \text{misses}} = 1 - \text{MR} \]

- **Miss Ratio:**
  \[ MR = \frac{\text{misses}}{\text{hits} + \text{misses}} = 1 - \text{HR} \]
Cache Metrics

- **Hit Ratio:**
  \[ HR = \frac{\text{hits}}{\text{hits} + \text{misses}} = 1 - MR \]

- **Miss Ratio:**
  \[ MR = \frac{\text{misses}}{\text{hits} + \text{misses}} = 1 - HR \]

- **Average Memory Access Time (AMAT):**
  \[ AMAT = \text{HitTime} + \text{MissRatio} \times \text{MissPenalty} \]

- Goal of caching is to improve AMAT
Cache Metrics

- **Hit Ratio:** 
  \[ HR = \frac{\text{hits}}{\text{hits} + \text{misses}} = 1 - MR \]

- **Miss Ratio:** 
  \[ MR = \frac{\text{misses}}{\text{hits} + \text{misses}} = 1 - HR \]

- **Average Memory Access Time (AMAT):**
  \[ AMAT = \text{HitTime} + \text{MissRatio} \times \text{MissPenalty} \]

- Goal of caching is to improve AMAT
- Formula can be applied recursively in multi-level hierarchies:
  \[ AMAT = \text{HitTime}_{L_1} + \text{MissRatio}_{L_1} \times AMAT_{L_2} = \]
  \[ AMAT = \text{HitTime}_{L_1} + \text{MissRatio}_{L_1} \times (\text{HitTime}_{L_2} + \text{MissRatio}_{L_2} \times AMAT_{L_3}) = \ldots \]
Example: How High of a Hit Ratio?

What hit ratio do we need to break even? (Main memory only: AMAT = 100)
Example: How High of a Hit Ratio?

What hit ratio do we need to break even? (Main memory only: AMAT = 100)

\[ 100 = 4 + (1 - HR) \times 100 \Rightarrow HR = 4\% \]
Example: How High of a Hit Ratio?

What hit ratio do we need to break even?  
(Main memory only: AMAT = 100)

\[ 100 = 4 + (1 - HR) \times 100 \Rightarrow HR = 4\% \]

What hit ratio do we need to achieve AMAT = 5 cycles?
Example: How High of a Hit Ratio?

What hit ratio do we need to break even?
(Main memory only: AMAT = 100)

\[
100 = 4 + (1 - HR) \times 100 \implies HR = 4\%
\]

What hit ratio do we need to achieve AMAT = 5 cycles?

\[
5 = 4 + (1 - HR) \times 100 \implies HR = 99\%
\]
Basic Cache Algorithm (Reads)

On reference to $\text{Mem}[X]$, look for $X$ among cache tags

- CPU
- Tag: A, Mem[A]
- Tag: B, Mem[B]
- Main Memory
- (1-HR)
Basic Cache Algorithm (Reads)

On reference to Mem[X], look for X among cache tags

HIT: X = Tag(i) for some cache line i
Basic Cache Algorithm (Reads)

On reference to Mem[X], look for X among cache tags

HIT: X = Tag(i) for some cache line i

Return Data(i)
Basic Cache Algorithm (Reads)

On reference to Mem[X], look for X among cache tags

HIT: X = Tag(i) for some cache line i

Return Data(i)

MISS: X not found in Tag of any cache line
Basic Cache Algorithm (Reads)

On reference to Mem[X], look for X among cache tags

HIT: X = Tag(i) for some cache line i
Return Data(i)

MISS: X not found in Tag of any cache line
Read Mem[X]
Return Mem[X]
Select a line k to hold Mem[X]
Write Tag(k) = X,
Data(k) = Mem[X]
Basic Cache Algorithm (Reads)

On reference to Mem[X], look for X among cache tags

HIT: X = Tag(i) for some cache line i
Return Data(i)

MISS: X not found in Tag of any cache line
Read Mem[X]
Return Mem[X]
Select a line k to hold Mem[X]
Write Tag(k) = X,
Data(k) = Mem[X]

Q: How do we “search” the cache?
Thank you!

Next lecture: Cache Tradeoffs