Plan today

• Caches
Memory (ideal)

- Random access
- Fast
- Cheap
- Large
- Unified

4294967292
4294967293
4294967294
4294967295
Options for memory storage

• **SRAM** (transistors/registers)
  – Expensive, power-consuming, but fast is small units
  – The larger the memory is, the slower it gets

• **DRAM** (capacitors)
  – Cheap, large, low power to maintain data, but expensive to access

• **Permanent memory** (magnetic or flash storage)
  – Even cheaper, no power needed to maintain data
  – Very long time to access
Memory Hierarchy

- We can’t build one memory that does everything we want, so we build a system of multiple memories that work together.
- Some parts of memory (the parts currently in use) will be held in small, fast memories near the processor, called **caches**.
- Memory addresses get checked against what is already in the cache:
  - If the address is present or cached, we have a **cache hit**.
  - If the address is not present in the cache, we **miss** and go the larger memory.
Cache in hardware

- **Lines**
  - A collection of addresses stored in one cache entry
- **Tags**
  - Address of a line stored in the cache
- **Access procedure:**
  - Determine where in the cache the address should be
  - Check the tag for that cache line
    - Match, cache hit, perform the access
    - Match, cache miss, load new data from memory for that line and update the tag
## Example

<table>
<thead>
<tr>
<th>Address</th>
<th>Tag</th>
<th>Line</th>
<th>Word</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>4</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

### Cache Line

<table>
<thead>
<tr>
<th>Cache Line</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Access:
- 0x12085091
- 0x0129384C
- 0x12085094
- 0x0129384A
- 0x08503920
Locality

- Caches work because programs do not access memory randomly
- *Spatial Locality*: when a program accesses an address, it often accesses nearby addresses too
- *Temporal Locality*: when a program accesses an address once, it often accesses it again soon

So, what if the program has bad locality?
All accesses miss cache and have to go to slower memories
Parallel Caches

Instruction Cache

Data Cache

Credit: Wikimedia Commons
Cache Hierarchy

- Most processors have multiple levels of caches
  - Caches closest to the pipeline need to be fast enough to be accessed in a few cycles at most
  - Caches that fast are hard to make bigger than 32KB
  - Caches closest to main memory store as much data as possible on chip to avoid the need to access memory
  - Main memory takes ~1000 clock cycles to access on cache miss