Memory Hierarchy

- **L0:** CPU registers hold words retrieved from the L1 cache.
- **L1:** L1 cache holds cache lines retrieved from the L2 cache.
- **L2:** L2 cache holds cache lines retrieved from L3 cache.
- **L3:** L3 cache holds cache lines retrieved from main memory.
- **L4:** Main memory holds disk blocks retrieved from local disks.
- **L5:** Local secondary storage (local disks)
- **L6:** Remote secondary storage (e.g., cloud, web servers)

- Smaller, faster, and costlier (per byte) storage devices
- Larger, slower, and cheaper (per byte) storage devices

Storage devices:
- Local disks hold files retrieved from disks on remote servers
- Main memory holds disk blocks retrieved from local disks
- L3 cache holds cache lines retrieved from main memory
- L2 cache holds cache lines retrieved from L3 cache
- L1 cache holds cache lines retrieved from the L2 cache
- CPU registers hold words retrieved from the L1 cache
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
How well does this take advantage of spacial locality?
How well does this take advantage of temporal locality?
2-way Set Associative Cache

E = 2: Two lines per set
Assume: cache block size 8 bytes

Address of data: tag index offset

- log(# sets) bits
- log(block size) bits
- the rest of the bits
Exercise: 2-way Set Associative Cache

Memory

```
0x14 | 18
0x10 | 17
0x0c | 16
0x08 | 15
0x04 | 14
0x00 | 13
```

Access | tag | idx | off | h/m
--- | --- | --- | --- | ---
rd 0x00 |   |   |   |   
rd 0x04 |   |   |   |   
rd 0x14 |   |   |   |   
rd 0x00 |   |   |   |   
rd 0x04 |   |   |   |   
rd 0x14 |   |   |   |   
rd 0x20 |   |   |   |   

Assume 8 byte data blocks

<table>
<thead>
<tr>
<th>Set 0</th>
<th>Set 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Line 0</td>
<td>Line 1</td>
</tr>
<tr>
<td>0 0 47 48</td>
<td>0 1 47 48</td>
</tr>
</tbody>
</table>
Eviction from the Cache

On a cache miss, a new block is loaded into the cache

- Direct-mapped cache: A valid block at the same location must be evicted—no choice

- Associative cache: If all blocks in the set are valid, one must be evicted
  - Random policy
  - FIFO
  - LIFO
  - Least-recently used; requires extra data in each set
  - Most-recently used; requires extra data in each set
  - Most-frequently used; requires extra data in each set
Caching Organization Summarized

- A cache consists of lines

- A line contains
  - A block of bytes, the data values from memory
  - A tag, indicating where in memory the values are from
  - A valid bit, indicating if the data are valid

- Lines are organized into sets
  - Direct-mapped cache: one line per set
  - k-way associative cache: k lines per set
  - Fully associative cache: all lines in one set
Categorizing Misses

- **Compulsory**: first-reference to a block
- **Capacity**: cache is too small to hold all of the data
- **Conflict**: collisions in a specific set

**Average access time**: $\text{hit-time} + \text{miss-rate} \times \text{miss-penalty}$
Typical Intel Core i7 Hierarchy

Processor package

Core 0
- Regs
- L1 d-cache
- L1 i-cache
- L2 unified cache
- L3 unified cache (shared by all cores)

Core 3
- Regs
- L1 d-cache
- L1 i-cache
- L2 unified cache

L1 d-cache and i-cache: 32 KB, 8-way, Access: 4 cycles

L2 unified cache: 256 KB, 8-way, Access: 10 cycles

L3 unified cache: 8 MB, 16-way, Access: 40-75 cycles

Block size: 64 bytes for all caches.

Main memory
Caching and Writes

• What to do on a write-hit?
  • **Write-through:** write immediately to memory
  • **Write-back:** defer write to memory until replacement of line
    • Need a dirty bit (line different from memory or not)

• What to do on a write-miss?
  • **Write-allocate:** load into cache, update line in cache
    • Good if more writes to the location follow
  • **No-write-allocate:** writes straight to memory, does not load into cache

• Typical
  • Write-through + No-write-allocate
  • **Write-back + Write-allocate**
Exercise: Write-through + No-write-allocate

Memory

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x24</td>
<td>22</td>
</tr>
<tr>
<td>0x20</td>
<td>21</td>
</tr>
<tr>
<td>0x1c</td>
<td>20</td>
</tr>
<tr>
<td>0x18</td>
<td>19</td>
</tr>
<tr>
<td>0x14</td>
<td>18</td>
</tr>
<tr>
<td>0x10</td>
<td>17</td>
</tr>
</tbody>
</table>

Cache

<table>
<thead>
<tr>
<th>Line</th>
<th>Valid Tag</th>
<th>Data Block</th>
</tr>
</thead>
<tbody>
<tr>
<td>Line 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Line 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Line 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Line 3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Assume 4 byte data blocks

Access | tag | idx | off | h/m | rd 0x10 | wr 0x10 | wr 0x24 | rd 0x24 | rd 0x20 |
<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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Line 0</th>
<th>Line 1</th>
<th>Line 2</th>
<th>Line 3</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>47</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>47</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>47</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>47</td>
<td>0</td>
<td>47</td>
</tr>
</tbody>
</table>
Exercise: Write-back + Write-allocate

Assume 4 byte data blocks

<table>
<thead>
<tr>
<th>Access</th>
<th>tag</th>
<th>idx</th>
<th>off</th>
<th>h/m</th>
</tr>
</thead>
<tbody>
<tr>
<td>rd 0x10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>wr 0x10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>wr 0x24</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rd 0x24</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rd 0x20</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>