It is ‘impossible’ to have memory that is both
- Unlimited (large in capacity)
- And fast

We create an illusion for the programmer
- Before that, let us that the way programs access memory

Principle of Locality
- Programs access a small proportion of their address space at any time
  - Temporal locality
    - Items accessed recently are likely to be accessed again soon
    - E.g., instructions in a loop
  - Spatial locality
    - Items near those accessed recently are likely to be accessed soon
    - E.g., sequential instruction access, array data

To Take Advantage of Locality
- Employ memory hierarchy
  - Use multiple levels of memories
  - ‘Larger’ distance from processor =>
    - Larger size
    - Larger access time
Memory Technology

- Static RAM (SRAM)
  - 0.5ns – 2.5ns, $2000 – $5000 per GB
- Dynamic RAM (DRAM)
  - 50ns – 70ns, $20 – $75 per GB
- Magnetic disk
  - 5ms – 20ms, $0.20 – $2 per GB
- Ideal memory
  - Access time of SRAM
  - Capacity and cost/GB of disk

Taking Advantage of Locality

- Memory hierarchy
- Store everything on disk
- Copy recently accessed (and nearby) items from disk to smaller DRAM memory
  - Main memory
- Copy more recently accessed (and nearby) items from DRAM to smaller SRAM memory
  - Cache memory attached to CPU

Memory Hierarchy Levels

- Block (aka line): unit of copying
  - May be multiple words
- If accessed data is present in upper level
  - Hit: access satisfied by upper level
    - Hit ratio: hits/accesses
- If accessed data is absent
  - Miss: block copied from lower level
    - Time taken: miss penalty
    - Miss ratio: misses/accesses
    - Miss ratio = 1 – hit ratio
  - Then accessed data supplied from upper level
This structure, with the appropriate operating mechanisms, allows the processor to have an access time that is determined primarily by level 1 of the hierarchy and yet have a memory as large as level n. Although the local disk is normally the bottom of the hierarchy, some systems use tape or a file server over a local area network as the next levels of the hierarchy.

### Cache Memory

- **Cache memory**
  - The level of the memory hierarchy closest to the CPU

- **Given accesses** \(X_1, \ldots, X_{n-1}, X_n\)
  - How do we know if the data is present?
  - Where do we look?

### Direct Mapped Cache

- Location determined by address
- Direct mapped: only one choice
  - \((\text{Block address}) \mod (\#\text{Blocks in cache})\)
Direct Mapped Cache

- If \#Blocks is a power of 2
  - Use low-order address bits to compute address

Tag Bits

- Each cache location can store the contents of more than one memory location
- How do we know which particular block is stored in a cache location?
  - Add a set of tag bits to the cache
  - Tag needs only need the high-order bits of the address

Valid Bits

- What if there is no data in a location?
  - Valid bit: 1 = present, 0 = not present
  - Initially 0 because when the processor starts up, the cache does not have any valid data

Cache Example

- 8-blocks, 1 word/block, direct mapped
- Initial state

<table>
<thead>
<tr>
<th>Index</th>
<th>V</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>001</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>010</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>011</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>110</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>111</td>
<td>N</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Cache Example

<table>
<thead>
<tr>
<th>Word addr</th>
<th>Binary addr</th>
<th>Hit/miss</th>
<th>Cache block</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>10 110</td>
<td>Miss</td>
<td>110</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Index</th>
<th>V</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>001</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>010</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>011</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>110</td>
<td>Y</td>
<td>10</td>
<td>Mem[10110]</td>
</tr>
<tr>
<td>111</td>
<td>N</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Cache Example

<table>
<thead>
<tr>
<th>Word addr</th>
<th>Binary addr</th>
<th>Hit/miss</th>
<th>Cache block</th>
</tr>
</thead>
<tbody>
<tr>
<td>26</td>
<td>11 010</td>
<td>Miss</td>
<td>010</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Index</th>
<th>V</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>001</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>010</td>
<td>Y</td>
<td>11</td>
<td>Mem[11010]</td>
</tr>
<tr>
<td>011</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>110</td>
<td>Y</td>
<td>10</td>
<td>Mem[10110]</td>
</tr>
<tr>
<td>111</td>
<td>N</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Cache Example

<table>
<thead>
<tr>
<th>Word addr</th>
<th>Binary addr</th>
<th>Hit/miss</th>
<th>Cache block</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>10 000</td>
<td>Hit</td>
<td>000</td>
</tr>
<tr>
<td>3</td>
<td>00 011</td>
<td>Miss</td>
<td>011</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Index</th>
<th>V</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>Y</td>
<td>10</td>
<td>Mem[10000]</td>
</tr>
<tr>
<td>001</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>010</td>
<td>Y</td>
<td>11</td>
<td>Mem[11010]</td>
</tr>
<tr>
<td>011</td>
<td>Y</td>
<td>00</td>
<td>Mem[00011]</td>
</tr>
<tr>
<td>100</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>110</td>
<td>Y</td>
<td>10</td>
<td>Mem[10110]</td>
</tr>
<tr>
<td>111</td>
<td>N</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Cache Example

<table>
<thead>
<tr>
<th>Word addr</th>
<th>Binary addr</th>
<th>Hit/miss</th>
<th>Cache block</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>10 000</td>
<td>Hit</td>
<td>000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Index</th>
<th>V</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>Y</td>
<td>10</td>
<td>Mem[10000]</td>
</tr>
<tr>
<td>001</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>010</td>
<td>Y</td>
<td>11</td>
<td>Mem[11010]</td>
</tr>
<tr>
<td>011</td>
<td>Y</td>
<td>00</td>
<td>Mem[00011]</td>
</tr>
<tr>
<td>100</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>110</td>
<td>Y</td>
<td>10</td>
<td>Mem[10110]</td>
</tr>
<tr>
<td>111</td>
<td>N</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Cache Example

<table>
<thead>
<tr>
<th>Word addr</th>
<th>Binary addr</th>
<th>Hit/miss</th>
<th>Cache block</th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>10 010</td>
<td>Miss</td>
<td>010</td>
</tr>
</tbody>
</table>

Index | V | Tag | Data               |
------|---|-----|--------------------|
000   | Y | 10  | Mem[10000]         |
001   | N |     |                    |
010   | Y | 10  | Mem[10010]         |
011   | Y | 00  | Mem[00011]         |
100   | N |     |                    |
101   | N |     |                    |
110   | Y | 10  | Mem[10110]         |
111   | N |     |                    |

Cache Misses

- On cache hit, CPU proceeds normally
- On cache miss
  - Stall the CPU pipeline
  - Fetch block from next level of hierarchy
  - Instruction cache miss
    - Restart instruction fetch
  - Data cache miss
    - Complete data access

Write-Through

- On each data-write hit, could just update the block in cache
  - But then cache and memory would be inconsistent
- Write through: also update memory
  - But makes writes take longer

Write-Buffer

- Solution: write buffer
  - Holds data waiting to be written to memory
  - CPU continues immediately after writing to write-buffer
    - Write-buffer is freed later when memory write is completed
  - But CPU stalls on write if write buffer is already full
### Write-Buffer

- Write-buffer can become full and the processor will stall if
  - Rate of memory completing the write operation is less than the rate at which write instructions are generated
  - Or if there is a burst of writes

### Write-Back

- Alternative: On data-write hit, just update the block in cache
  - Keep track of whether each block is dirty
- When a dirty block is replaced
  - Write it back to memory

### Measuring Cache Performance

- Components of CPU time
  - Program execution cycles
    - Includes cache hit time
  - Memory stall cycles
    - Mainly from cache misses

### Measuring Cache Performance

- Components of CPU time
  - Program execution cycles
    - Includes cache hit time
  - Memory stall cycles
    - Mainly from cache misses
- With simplifying assumptions:

\[
\text{Memory stall cycles} = \frac{\text{Memory accesses}}{\text{Program}} \times \text{Miss rate} \times \text{Miss penalty}
\]

\[
= \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Misses}}{\text{Instruction}} \times \text{Miss penalty}
\]
**Cache Performance Example**

- **Given**
  - I-cache miss rate = 2%
  - D-cache miss rate = 4%
  - Miss penalty = 100 cycles
  - Base CPI (ideal cache) = 2
  - Load & stores are 36% of instructions
- **How much faster would a processor with a perfect cache that never misses?**

**Virtual Memory**

- VM is the technique of using main memory as a “cache” for secondary (disk) storage
  - Managed jointly by CPU hardware and the operating system (OS)
  - Same underlying concept as in cache but different in terminologies

**VM Terms**

- Virtual address is the address produced by the program
- Physical address is an address in the main memory
- CPU and OS translate virtual addresses to physical addresses
  - VM “block” is called a page
  - VM translation “miss” is called a page fault

**Cache Performance Example**

- Miss cycles for all instructions if \( I \) is the instruction count
  - I-cache: \( I \times 0.02 \times 100 = 2I \)
  - D-cache: \( I \times 0.36 \times 0.04 \times 100 = 1.44I \)
  - Miss cycle per instruction is \( 2 + 1.44 \)
  - Total cycles per instruction is \( 2 + 2 + 1.44 \)
  - So, actual CPI = \( 2 + 2 + 1.44 = 5.44 \)
  - Ideal CPU is \( 5.44/2 = 2.72 \) times faster
Page Fault Penalty

- On page fault, the page must be fetched from disk
  - Takes millions of clock cycles. Main memory latency is around 100,000 times better than the disk latency
- Try to minimize page fault rate
  - Smart replacement algorithms implemented in software in the OS
    - Reading from disk is slow enough and software overhead is negligible

Motivation I

- Multiple programs share main memory and they can change dynamically
- To avoid writing into each other’s data, we would like separate address space for each program
- With VM, each gets a private virtual address space holding its frequently used code and data
- VM translates the virtual address into physical address allowing protection from other programs

Motivation II

- A large program cannot fit into the main memory
  - VM automatically maps addresses into disk space if the main memory is not sufficient

Address Translation

- Address translation: the process by which the virtual address is mapped to a physical address
Address Translation

Two components of virtual address:

- Page offset does not change and the number of bits determines the size of the page.
- The number of pages addressable with virtual address might be larger than the number of pages addressable with the physical address which gives the illusion of unbounded amount of virtual memory.

Translation Using a Page Table

A page table, that resides on memory, is used for address translation.

- Each program has its own page table.
Translation Using a Page Table

Note the use of a Valid bit

---

Page Tables

- Stores placement information
  - Array of page table entries, indexed by virtual page number
  - Page table register in CPU points to page table in physical memory
- If page is present in memory
  - it stores the physical page number
  - Plus other status bits
- If page is not present
  - a page fault occurs and the OS is given control
  - Next few slides, we recap some OS concepts

---

Recap: Process

- A process (a program in execution), has a context defined by the values in its program counter, registers, and page table
- If another process preempts a running process, this context must be saved

---

The 5 State Process Model

New → Ready → Running → Blocked → Terminated

- New: Created
- Ready: Admit
- Running: Schedule new job
- Blocked: I/O completion
- Terminated: Exit

---
Role of the OS

- A process (a program in execution), has a context defined by the values in its program counter, registers, and page table.
- If another process preempts this process, this context must be saved:
  - Rather than save the entire page table, only the page table register is saved.
- To restart the process in the 'running' state, the operating system reloads the context.

The OS is responsible for allocating the physical memory and updating the page tables.

- It maintains that virtual address of different processes do not collide thus providing protection.
- Page fault is handled by OS.

Role of the OS

- The OS creates a space on the disk (not main memory) for all pages of a process when the process is created:
  - called swap space.
- OS also creates a record of where each virtual address of the process is located in the disk.

Mapping Pages to Storage
Page Fault

- Handled by OS
- If all pages in main memory are in use (it is full), the OS must choose a page to replace
- The replaced page must be written to the swap space on the disk
- To reduce page fault rate,
  - prefer least-recently used (LRU) replacement
  - Predict that the page that was NOT used recently will be NOT used in near future

Chapter 5 — Large and Fast: Exploiting Memory Hierarchy

Writes

- Disk writes take millions of cycles
  - Write through is impractical because it takes millions of cycles to write to the disk
    - Building a write buffer is impractical
  - VM uses write-back

Chapter 5 — Large and Fast: Exploiting Memory Hierarchy

Fast Translation Using a TLB

- Address translation would appear to require extra memory references
  - One to access the page table itself
  - Then the actual memory access
- But access to page tables has good locality
  - So use a fast cache of recently used translations
  - Called a Translation Look-aside Buffer (TLB)

Chapter 5 — Large and Fast: Exploiting Memory Hierarchy

Memory Protection

- VM allows different processes to share the same main memory
  - Need to protect against errant access
  - Requires OS assistance

Chapter 5 — Large and Fast: Exploiting Memory Hierarchy
Memory Protection

- Hardware support for OS protection
  - Support two modes
    - Privileged supervisor mode (aka kernel mode) meaning that the OS is running
    - User mode
  - Privileged instructions that only the OS can use
    - Allow it to write to the supervisor bit, and page table pointer
  - Allow mechanisms (e.g., special instructions) to switch between supervisor mode and the user mode (e.g., syscall in MIPS)

- These features allow the OS to change page tables while preventing a user process from changing them

Concluding Remarks

- Fast memories are small, large memories are slow
  - We really want fast, large memories 😊
  - Caching gives this illusion 😊
- Principle of locality
  - Programs use a small part of their memory space frequently
- Memory hierarchy
  - L1 cache ↔ L2 cache ↔ … ↔ DRAM memory ↔ disk

Exam

- Questions will be in English
- You may answer in Swedish or English
- Dictionary is allowed
  - Swedish to English
  - English to Swedish
  - But, electronic dictionary is NOT allowed

Exam Questions

- Most likely a mixture of problem oriented questions and subjective questions
- Problem oriented questions
  - Like questions in homework
- Subjective questions …
Subjective questions like ...

- What does the control unit do?
- What is RISC? CISC?
- Why are fix-length instructions good?
- What is principle of locality?

Exam Questions

- Questions from past year are in the course website
- You may look at them for ‘some’ inspiration
  - **Note:** Past year had different instructor, different textbook, and different home-works even if there are some common topics

From the Textbook

- *Parts from:* 5.1, 5.2, 5.3, 5.4