Roadmap

C:

```c
void get_mpg(car* c)
{
    float mpg = (c->miles) / (c->gals);
    printf("%f
", mpg);
}
```

Java:

```java
public float getMPG()
{
    return miles / gals;
}
```

Assembly language:

```asm
; push %rbp
; mov %rsp, %rbp
    ...
; pop %rbp
; ret
```

OS:

```c
get_mpg:
    pushq %rbp
    movq %rsp, %rbp
    ...
    popq %rbp
    ret
```

Machine code:

```
011101100000111000
10001101010000010000000010
10001001110000101100000111111111101000011111
```

Computer system:

Again: Processes

- **Definition:** A process is an instance of a running program
  - One of the most important ideas in computer science
  - Not the same as "program" or "processor"

- **Process provides each program with two key abstractions:**
  - Logical control flow
    - Each process seems to have exclusive use of the CPU
  - Private virtual address space
    - Each process seems to have exclusive use of main memory

- **How are these illusions maintained?**
  - Process executions interleaved (multi-tasking) – done...
  - Address spaces managed by virtual memory system – now!

Virtual Memory (VM)

- Overview and motivation
- VM as tool for caching
- Address translation
- VM as tool for memory management
- VM as tool for memory protection

Memory as we know it so far... is virtual!

- Programs refer to virtual memory addresses
  - movl (%eax), %eax
    - Conceptually memory is just a very large array of bytes
    - Each byte has its own address
  - System provides address space private to particular "process"

- Allocation: Compiler and run-time system
  - Where different program objects should be stored
  - All allocation within single virtual address space

- But...
  - We probably don’t have exactly $2^n$ bytes of physical memory.
  - We *certainly* don’t have $2^n$ bytes of physical memory for every process.
  - We have multiple processes that usually should not interfere with each other, but sometimes should share code or data.
Problem 1: How Does Everything Fit?

64-bit addresses can address several exabytes (18,446,744,073,709,551,616 bytes)

Physical main memory offers a few gigabytes (e.g., 8,589,934,592 bytes)

(Actually, it’s smaller than that dot compared to virtual memory.)

1 virtual address space per process, with many processes...

Problem 2: Memory Management

Process 1
Process 2
Process 3
Process n

... X ... stack
heap .text
.data

What goes where?

Problem 3: How To Protect

Physical main memory

Process i

Process j

Problem 4: How To Share?

Physical main memory

Process i

Process j

How can we solve these problems?
Indirection

- "Any problem in computer science can be solved by adding another level of indirection." –David Wheeler, inventor of the subroutine (a.k.a. procedure)

- Without Indirection
  
  ![Diagram of Without Indirection](image)

- With Indirection
  
  ![Diagram of With Indirection](image)

  What if I want to move Thing?

Indirection in Virtual Memory

- Each process gets its own private virtual address space
- Solves the previous problems

Address Spaces

- **Virtual address space**: Set of $N = 2^n$ virtual addresses
  \{0, 1, 2, 3, ..., N-1\}

- **Physical address space**: Set of $M = 2^m$ physical addresses ($n \geq m$)
  \{0, 1, 2, 3, ..., M-1\}

- Every byte in main memory has:
  - one physical address
  - zero, one, or more virtual addresses
**Mapping**

A virtual address can be mapped to either physical memory or disk.

**A System Using Physical Addressing**

- Used in “simple” systems with (usually) just one process:
  - embedded microcontrollers in devices like cars, elevators, and digital picture frames

**A System Using Virtual Addressing**

- Physical addresses are completely invisible to programs.
- Used in all modern desktops, laptops, servers, smartphones...
- One of the great ideas in computer science

**VM and the Memory Hierarchy**

- Think of virtual memory as array of \( N = 2^n \) contiguous bytes.
- **Pages** of virtual memory are usually stored in physical memory, but sometimes spill to disk:
  - Pages are another unit of aligned memory (size is \( P = 2^n \) bytes)
  - Each virtual page can be stored in any physical page.
or: Virtual Memory as Cache

- Think of virtual memory as an array of $N = 2^n$ contiguous bytes stored on a disk.
- Then physical main memory is used as a cache for the virtual memory array
  - The cache blocks are called pages (size is $P = 2^p$ bytes)

![Virtual Memory Diagram]

Virtual Memory Design Consequences

- Large page size: typically 4-8 KB, sometimes up to 4 MB
- Fully associative
  - Any virtual page can be placed in any physical page
  - Requires a “large” mapping function – different from CPU caches
- Highly sophisticated, expensive replacement algorithms in OS
  - Too complicated and open-ended to be implemented in hardware
- Write-back rather than write-through

Memory Hierarchy: Core 2 Duo

- **SRAM**: Static Random Access Memory
  - L1 I-cache
  - L1 D-cache
  - L2 unified cache
  - Throughput: 16 B/cycle, 8 B/cycle, 2 B/cycle, 1 B/30 cycles
  - Latency: 3 cycles, 14 cycles, 100 cycles, millions

- **DRAM**: Dynamic Random Access Memory
  - ~4 MB
  - ~4 GB

- **Disk**: ~500 GB

Address Translation

**CPU Chip**

```
<table>
<thead>
<tr>
<th>CPU</th>
<th>Virtual address (VA)</th>
<th>MMU</th>
<th>Physical address (PA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4100</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

How do we perform the virtual -> physical address translation?
Address Translation: Page Tables

- A page table is an array of page table entries (PTEs) that maps virtual pages to physical pages.

![Page Table Diagram]

Page Hit

- Page hit: reference to VM byte that is in physical memory

Page Fault

- Page fault: reference to VM byte that is NOT in physical memory
Fault Example: Page Fault

- User writes to memory location
- That portion (page) of user’s memory is currently on disk

```c
int a[1000];
main()
{
    a[500] = 13;
}
```

80483b7:  c7 05 10 9d 04 08 0d movl  $0xd,0x8049d10

User Process  OS

movl  exception: page fault
returns
Create page and load into memory

- Page handler must load page into physical memory
- Returns to faulting instruction: mov is executed again!
- Successful on second try

Handling Page Fault

- Page miss causes page fault (an exception)
- Page fault handler selects a victim to be evicted (here VP 4)
Handling Page Fault

- Page miss causes page fault (an exception)
- Page fault handler selects a victim to be evicted (here VP 4)
- Offending instruction is restarted: page hit!

Why does VM work on RAM/disk? Locality.

- Virtual memory works well for avoiding disk accesses because of locality
  - Same reason that L1 / L2 / L3 caches work

- The set of virtual pages that a program is “actively” accessing at any point in time is called its working set
  - Programs with better temporal locality will have smaller working sets

- If (working set size of one process < main memory size):
  - Good performance for one process after compulsory misses

- But if SUM(working set sizes of all processes) > main memory size:
  - Thrashing: Performance meltdown where pages are swapped (copied) between memory and disk continuously. CPU always waiting or paging.
  - Full quote: "Every problem in computer science can be solved by adding another level of indirection, but that usually will create another problem."