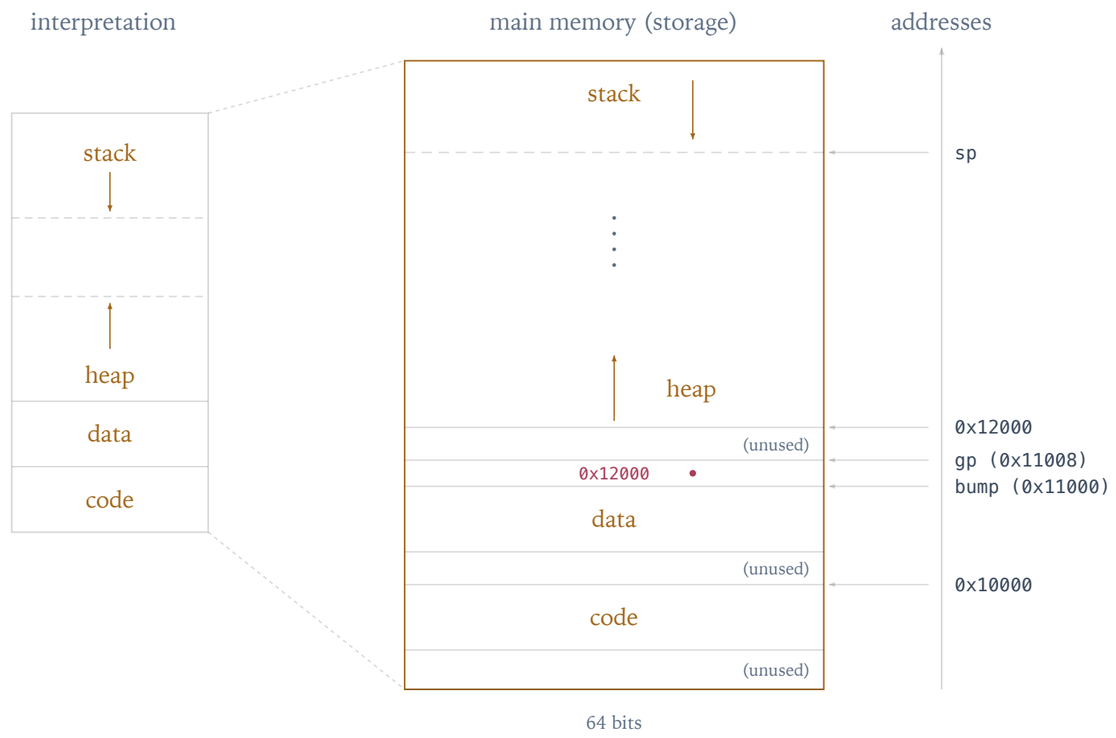
SYSTEMS ENGINEERING · WEEK 4 · STATION III

Virtual Memory

Paging: the virtual address is notation, the physical address is its meaning.

THE PROBLEM

Two programs, one memory, both compiled to start at address `0x10000`.



a program's memory layout: code, data, heap, stack

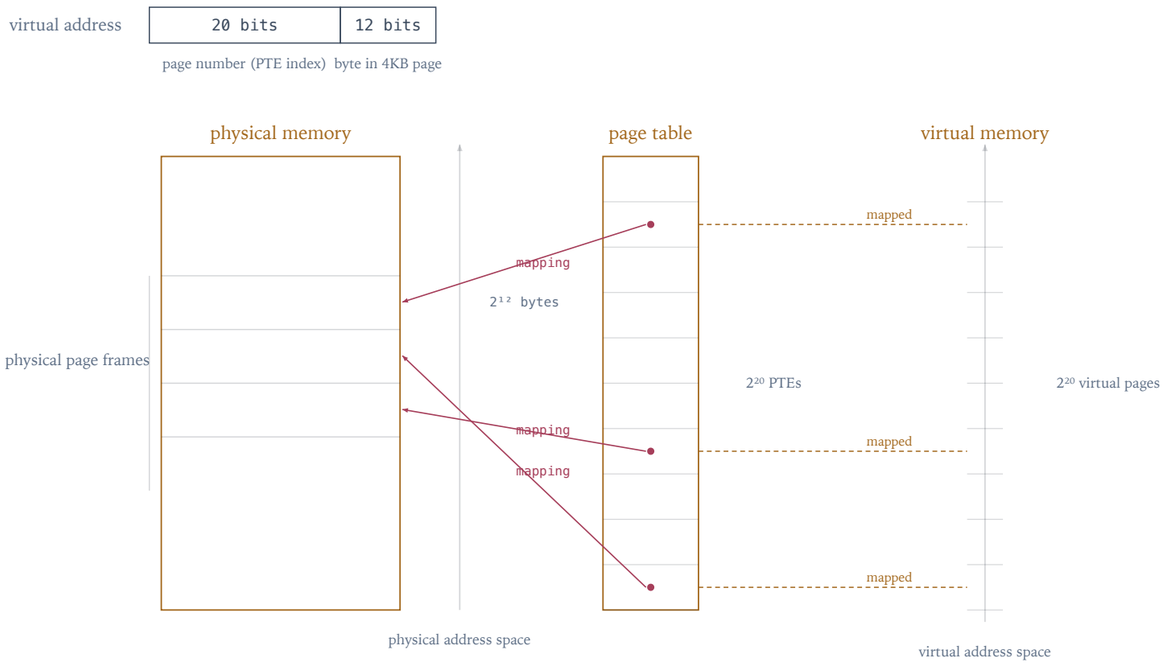
The compiler lays out every program the same way, and the machine has one memory. So either the second program is relocated, which means rewriting its addresses, or every address a program uses is a name that the kernel resolves at run time.

The second is virtual memory. The program's addresses are *virtual*: notation. The addresses the memory chip sees are *physical*: meaning. The mapping from one to the other is owned by the kernel, and a program that cannot name a physical address cannot touch another program's memory.

Isolation as a translation problem. The Meaning chapter's semantics, applied to addresses.

PAGING

Pages to frames: translate the top bits, keep the bottom twelve.



a virtual address splits into page number and offset; the page table maps pages to frames

Cut both address spaces into 4 KB blocks: *pages* on the virtual side, *frames* on the physical. A 32-bit address is a 20-bit page number and a 12-bit offset. Translate the page number through a table; copy the offset unchanged. That is the whole mechanism.

A million pages means a million entries: 4 or 8 MB per table if flat, which selfie accepts for simplicity. Real systems use a tree so that untouched regions cost nothing, and a cache, the TLB, so that the tree is rarely walked.

In selfie, `tlb` is the procedure that translates, and the emulator calls it on every load, store and fetch. Find it; it is a few lines.

PAGE FAULTS

Nothing is mapped until it is touched. The first touch is an exception.

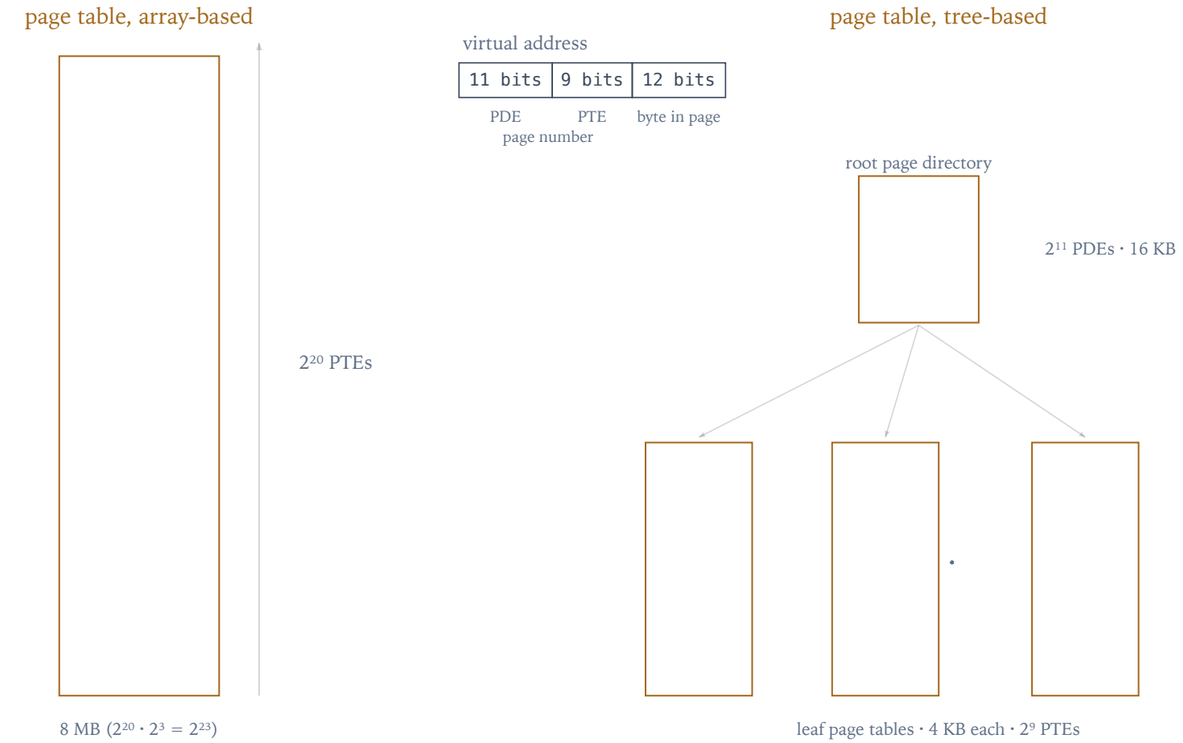
A new machine context has an empty page table. Its first fetch is a *page fault*: the translation fails, the machine stops, the kernel takes over, allocates a frame, writes the entry, and lets the machine retry. On-demand paging: creating a machine costs nothing until it runs.

The same exception distinguishes a legitimate first touch from an error. A fault in the heap below the program break, or in the stack, gets a frame. A fault anywhere else is a segmentation fault, and the kernel decides what to do with a program that did that: in selfie, it stops it.

```
$ ./selfie -c selfie.c -m 1
./selfie: selfie.c exiting with exit code 0
./selfie: summary: 85754 executed instructions in total
./selfie: 0.19MB mapped memory [19.92% of 1MB physical memo
./selfie: 14 syscalls, 1 page faults, 0 timer interrupts
```

TABLE OR TREE

The same map, two data structures, one meaning.



a flat page table and a two-level tree, mapping the same pages

Flat: one array, one lookup, a million entries whether used or not. Tree: a directory of tables, two lookups, only the touched regions allocated. Both denote the same function from pages to frames.

Same meaning, different cost: the pattern of the Cost chapter, seen for the first time in this class. The kernel chooses the notation; the program observes only the meaning. Which is exactly what makes the choice free.

WHERE THE LOOP STARTS

Who translates the kernel's own addresses?

In mipster, nobody has to: the emulator is a program with its own memory, the guest's memory is an array inside it, and translation is index arithmetic. No self-reference.

On real hardware, the kernel runs on the same processor with the same address translation switched on. So the page table that maps the kernel must itself be mapped, and the code that writes page tables runs through one. The bootstrap problem of week 6, in its spatial form.

THIS WEEK

Read `tlb`, `map_page` and the page fault handling in mipster. No new assignment: finish `self-assembler`. Next week time: the timer interrupt, and why it is the kernel's answer to the halting problem.