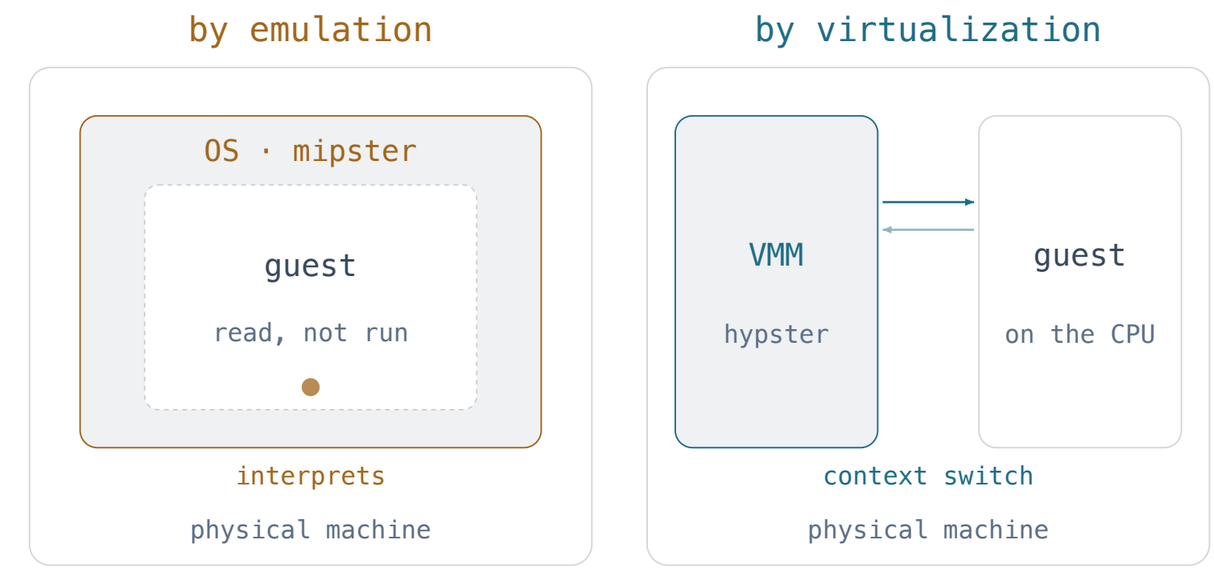
SYSTEMS ENGINEERING · WEEK 6 · STATION III

Self-Hosting

Hypster. Emulation equals virtualization. The bootstrap problem is Gödel's second theorem, and the trusted computing base is its axiom.

VIRTUALIZATION

Run the guest's instructions on the host's processor. Trap only what crosses the boundary.



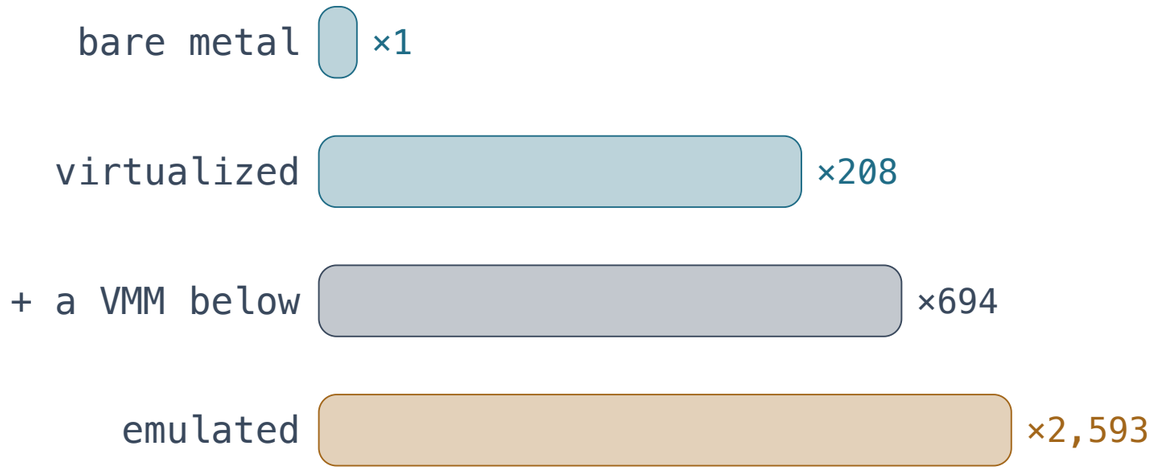
an OS by emulation reads the guest; an OS by virtualization runs it

Hypster does what mipster does with one change: instead of interpreting the guest's instructions, it maps the guest's context onto the processor and lets the processor execute them. Adds, loads, branches: native speed. System calls, page faults, the timer: exceptions that come back to hypster, which handles them exactly as mipster would.

Same signature as mipster, same contexts, same handlers. The guest cannot tell which one hosts it: same output, same 86,380 instructions. Emulation and virtualization are semantically equivalent, and the emulator is the executable specification.

THE BILL

×2,593 versus ×208. Performance is the only reason.



host instructions per guest instruction, emulated and virtualized

```
$ make os-emu
> selfie.m: summary: 86380 executed instructions in total
./selfie: summary: 17860937 executed instructions in total
```

Hypster on mipster: the guest still 86,380, the host 18 million instead of 222. The factor that remains is the cost of the exceptions and the context switches, not of the instructions. On real hardware it is a few percent.

Nothing else changed. Not the isolation the guest gets, not the output, not the semantics. Virtualization is an optimisation of emulation, and every optimisation in this book must preserve meaning. So what does it cost, if not meaning?

SELF-REFERENCE

The kernel now runs on the machine it isolates. It needs the isolation it provides.

virtualization is emulation plus self-reference

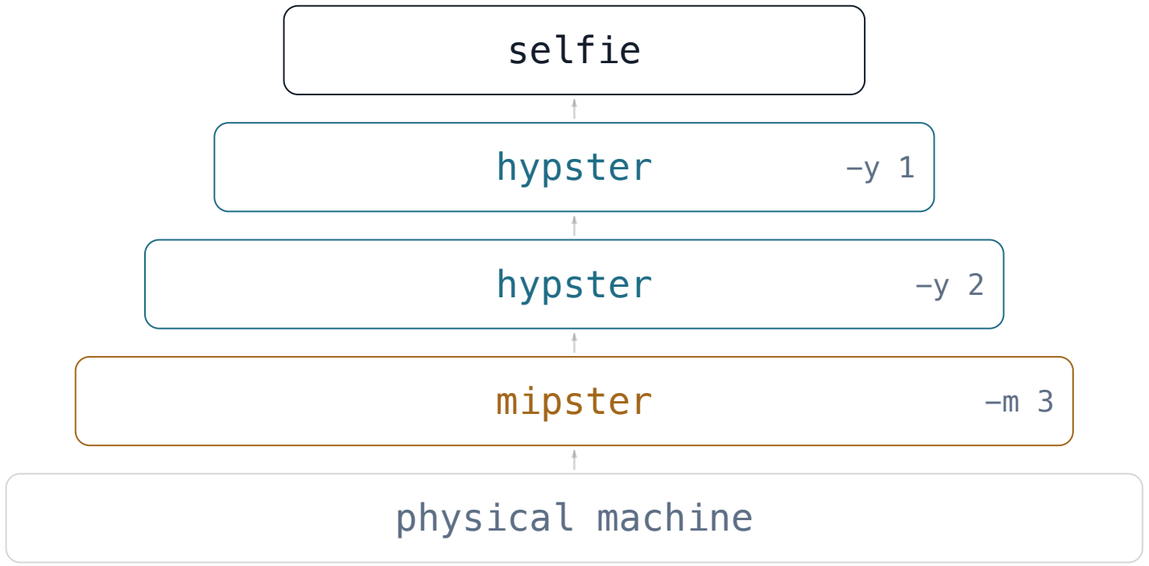
In mipster the guest is data; the host's processor and memory are never at stake. In hypster the guest runs on the host's processor, in the host's memory, translated by the host's page tables. Who protects the page tables? The page tables. Who stops a guest that ignores the timer? The timer, if the guest could not switch it off.

So the kernel must be isolated from the very things whose isolation it provides: its memory mapped by a table only it can write, its code in a mode only it can enter, its timer set by an instruction only it can execute. The privilege levels of week 2 exist for this and nothing else.

That is the price: self-reference. And self-reference, this book has said three times, is where every limit lives.

BOOTSTRAP

Hypervisor on hypervisor on emulator. Somebody has to be first.



self-hosting: each level virtualizes the one above, and the bottom is emulated

```
$ ./selfie -c selfie.c -o selfie.m -m 3 -l selfie.m -y 2 -
```

Hypster hosting hypster hosting selfie: each level asks the level below to create a context and run it; requests fall through to the bottom, where mipster, an emulator, actually executes. The ladder stands on something that does not stand on itself.

Gödel's second theorem in a kernel: a system cannot establish its own consistency from inside. A kernel cannot establish its own isolation from inside. Both need a ground outside the system. For the kernel it is hardware privilege, or an emulator, or a smaller kernel underneath.

TRUSTED COMPUTING BASE

What you cannot prove, you name. The TCB is the axiom set of a system.

The trusted computing base is everything a system's isolation depends on but cannot itself check: the hardware's privilege mechanism, the bottom of the ladder, the code that switches contexts. Engineering does not eliminate it; it makes it small, and names it.

Selfie's TCB is mipster: a few hundred lines that read the guest. seL4, the verified microkernel, has a TCB of about ten thousand lines and a proof of everything above it — a proof that assumes the hardware. There is always an assumed hardware. Axioms, not proven; chosen.

ASSIGNMENT · FORK–WAIT

Add `fork` and `wait` as system calls: a process that creates a copy of itself, and a parent that blocks until a child exits. Design this week; week 7 is about what a process is, and finishes it with `exit`. `./grader/self.py fork–wait` when it runs.