
Virtualization: one interface

Key takeaways

1. Virtualization keeps one interface and swaps what stands behind it. \$PATH, a venv, a container, and a VM do it with a variable, a directory, a kernel, and hardware.
2. What a boundary is made of decides what it isolates. A venv is a PATH prefix any program can ignore; a container shares the host's kernel; a VM shares only the hardware.
3. Virtualize the whole machine when the interface you must keep is the hardware's: it stays stable while operating systems and libraries drift.

Common misunderstandings of virtualization

Which claim would you defend right now? Choose before the evidence.

“A virtual machine is a slower computer; virtualization is the overhead you accept in exchange for sharing hardware.”

“A container is a small virtual machine.”

“Virtualization is what hypervisors do; \$PATH, venvs, and conda are something else.”

Keep your choice. Look for the mechanism or counterexample that would make it fail.

You already virtualize: \$PATH

The same command on two hosts reaches two different files

on odachi

mpirun

the name you type

looked up in

search \$PATH

PATH=/usr/local/bin:/usr/bin

first match

/usr/bin/mpirun

the file that runs

on epee

mpirun

the name you type

looked up in

search \$PATH

PATH=/opt/amazon/openmpi/bin:/usr/bin

first match

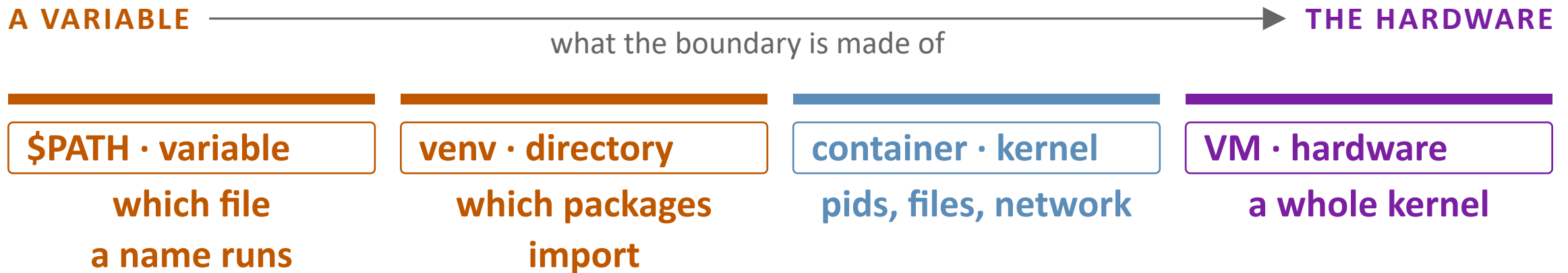
/opt/amazon/openmpi/bin/mpirun

the file that runs

The name is the interface; the file behind it is the implementation; \$PATH picks one.

Four boundaries, one move

Under each rung: what it isolates, then what still crosses it



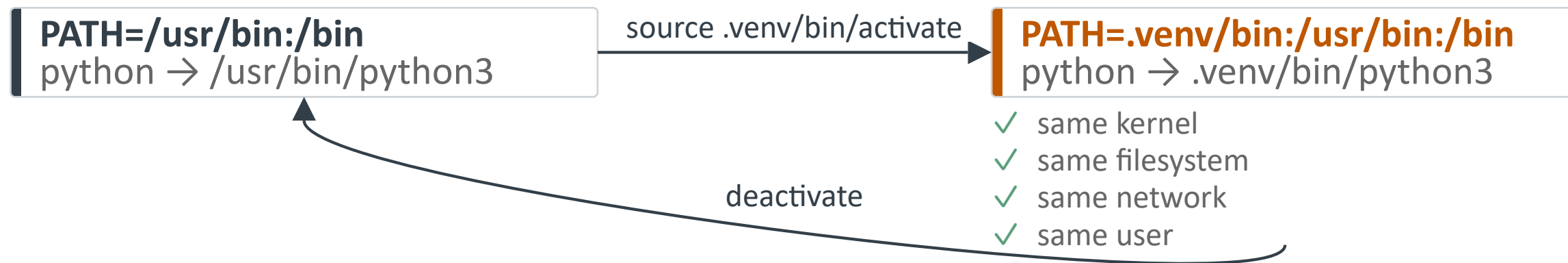
WHAT STILL CROSSES EACH BOUNDARY



A boundary isolates only what its material can enforce; heavier material leaks less and costs more to start.

A venv is a directory and a PATH prefix

Activating it edits one variable in your shell; nothing else moves



WHERE IT BREAKS: NOTHING ENFORCES THE PREFIX

#!/usr/bin/python3
a script names its Python

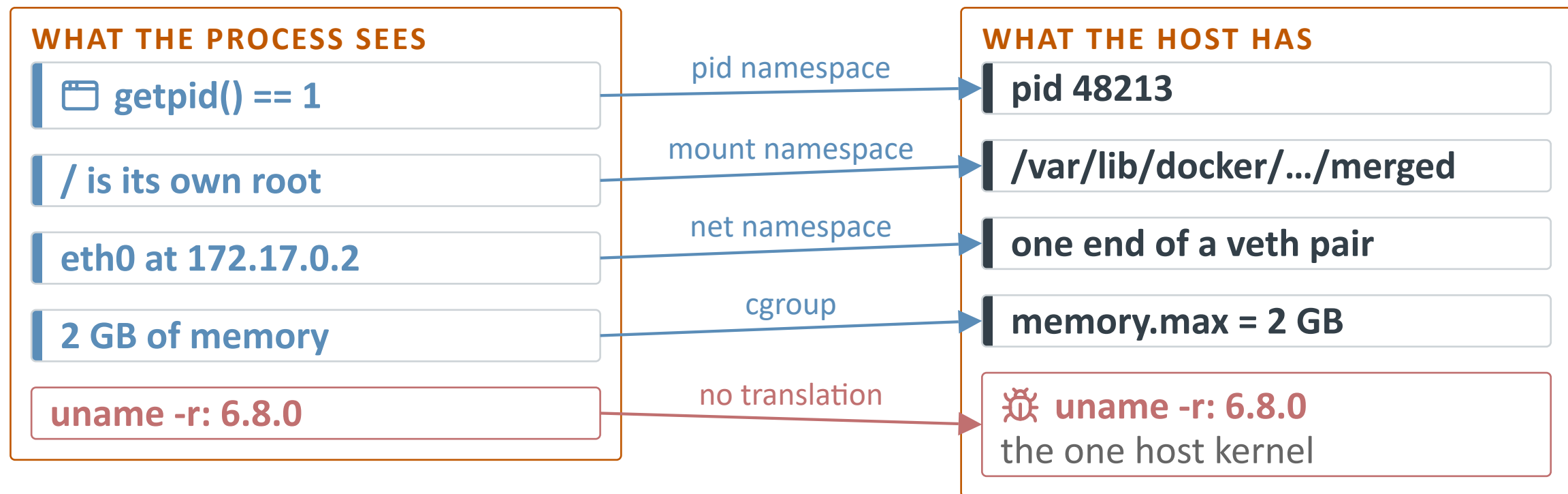
/usr/lib/libssl.so
C extensions link it

PYTHONPATH
still set by your shell

A venv works only for programs that ask `$PATH`; anything that does not walks past it.

A container is a translated process

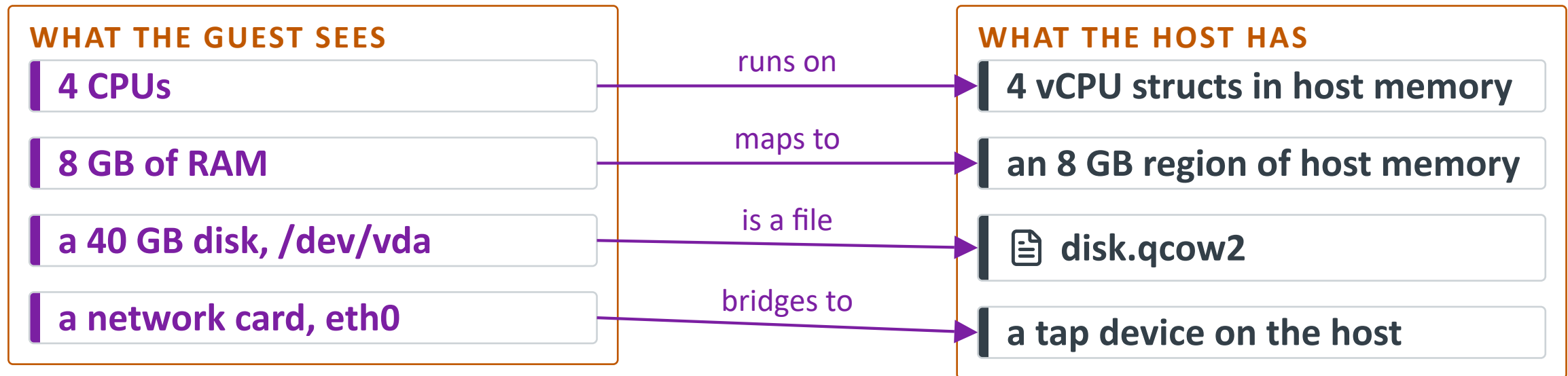
The kernel answers its system calls differently; nothing is copied (values illustrative)



Where it breaks: there is one kernel inside and out, so one kernel bug reaches every container.

A VM's hardware is a data structure

Every device the guest sees is an object the host owns (values illustrative)



Not a simulator: since 2005 the processor runs most guest instructions directly, so a VM is no longer slow.

Draw the boundary at a stable interface

Two ways to run a Windows program on another system

REIMPLEMENT THE OS'S INTERFACE Port the program

1993: Wine starts
rewriting the Windows API on Linux

2008: Wine 1.0
after 15 years of work

2026: Wine 11.0
still chasing each new Windows

REIMPLEMENT THE HARDWARE'S Virtualize the machine

Run the program's own OS
Windows itself, unmodified

CPU and RAM run directly
only disk and NIC are software

The hardware barely moves
decades-old x86 code still runs

Porting chases a moving OS for decades; the hardware interface barely moves, so virtualize it.

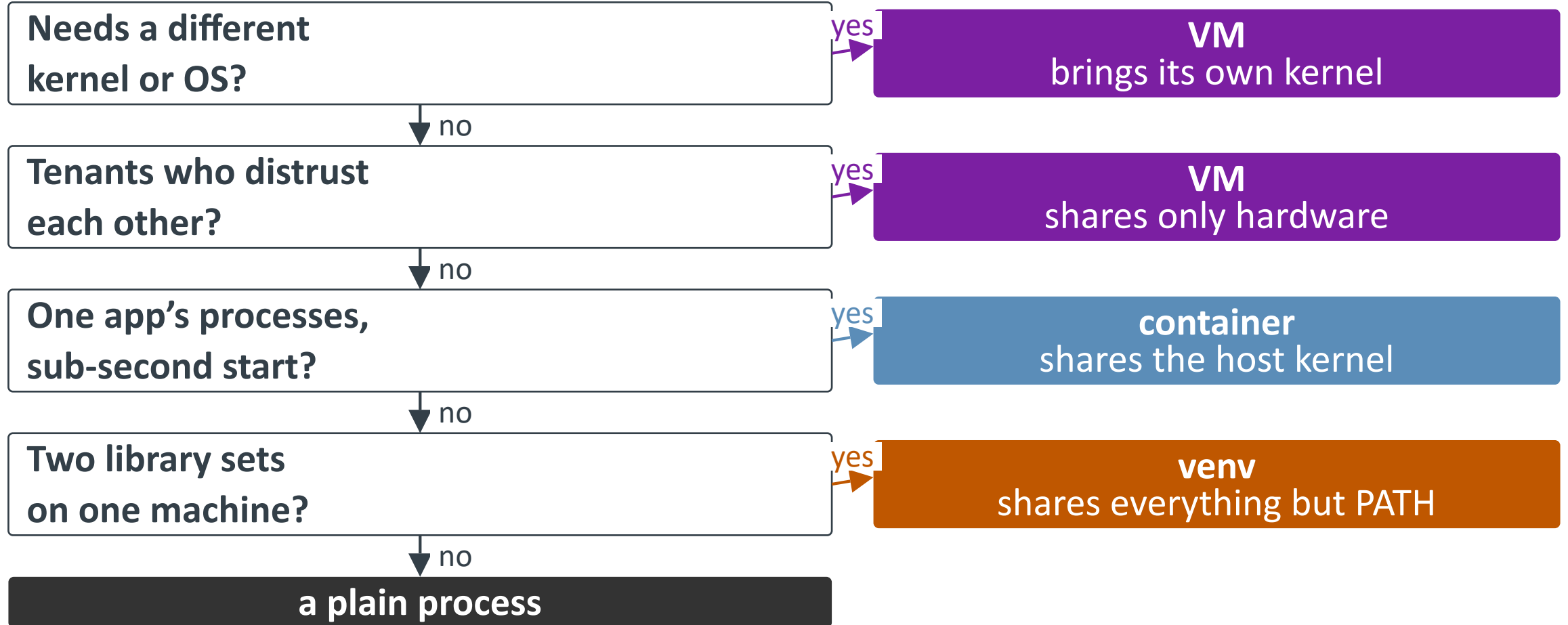
Popek & Goldberg is the yardstick

Three properties a virtual machine monitor must have, applied to every rung

	Equivalence same behavior	Resource control it holds the limits	Efficiency runs directly
venv	 partial if programs ask	 none nothing enforced	 yes runs natively
container	 yes host-kernel code	 partial cgroups cap it	 yes runs natively
VM	 yes any OS boots	 yes monitor holds all	 yes hardware-assisted

A container is not a small VM: it is faithful and fast but shares its kernel, so control is partial.

Pick the lightest boundary that holds



Ask in order: the first yes names the boundary; each rung lighter shares one more thing.

Key Takeaways

1. Virtualization keeps one interface and lets several implementations stand behind it. `$PATH`, a `venv`, a container, and a VM are the same move made with a variable, a directory, a kernel, and hardware; name the interface first.
2. What a boundary is made of decides what it isolates and what leaks. A `venv` is a `PATH` prefix any program can ignore; a container shares the host's kernel, so one kernel bug reaches every container; a VM shares only the hardware.
3. Virtualize the whole machine when the interface you need to keep is the hardware's. Porting chases a moving OS for decades; the hardware interface barely moves, so a VM runs the program's own OS unmodified at nearly native speed.

Exercise: the lightest boundary

10 min

- For each scenario name the lightest rung that works (process, venv, container, or VM) and the one thing that would leak one rung lighter.
- (a) Two projects on one laptop need Python 3.9 and Python 3.13.
- (b) A Windows-only chip-design tool must run on a Mac.
- (c) Run untrusted student submissions, each capped at 1 CPU and 2 GB of memory.
- (d) Rent one 96-core server to 100 paying customers who never meet.
- Produce a four-row table: scenario, rung, what leaks one rung lighter. Compare (b) and (c) with the group next to you.