CS 423
Operating System Design: OS Support for Containers

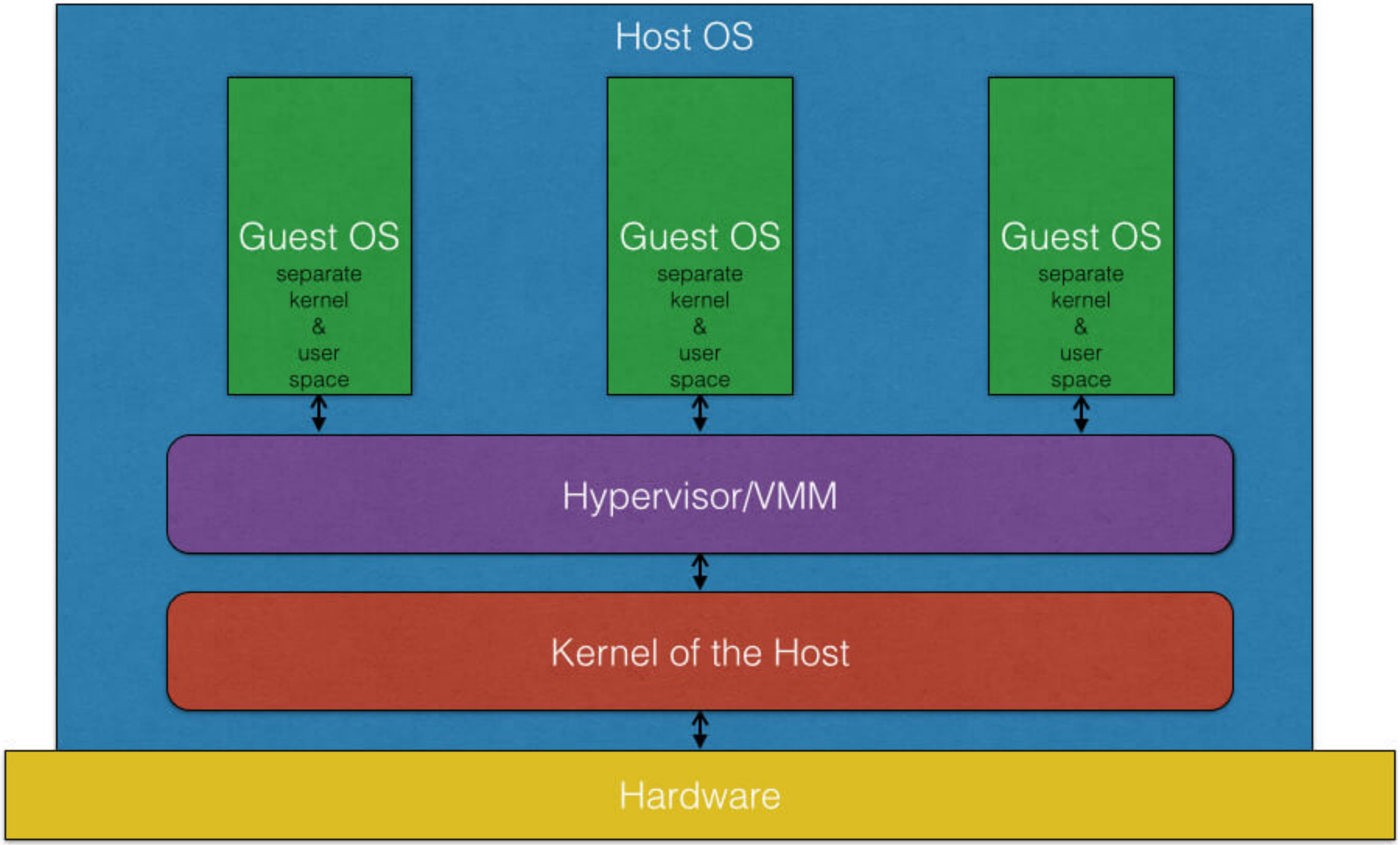
Tianyin Xu

Thanks for Adam Bates and Julie Evans

Cloud Computing (Gen 1)

- Dominated by Infrastructure-as-a-Service clouds (and storage services)
- Big winner was Amazon EC2
- Hypervisors that virtualized the hardware-software interface
- Customers were responsible for provisioning the software stack from the kernel up

Hypervisors

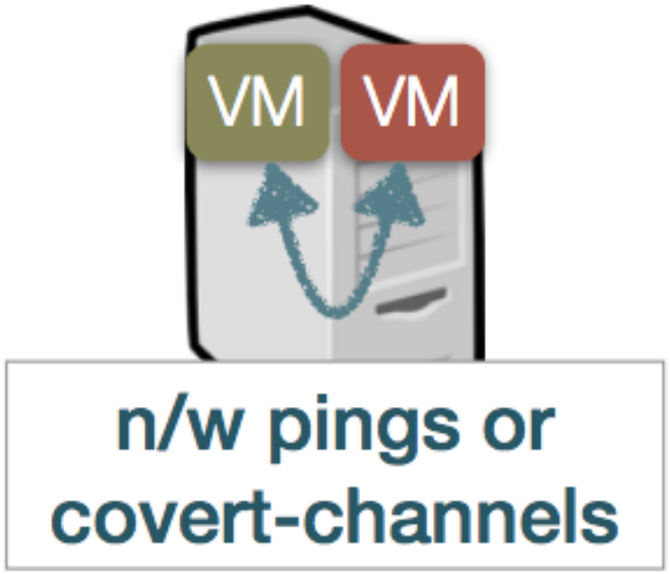


Hypervisors

- Strong isolation between different customer's virtual machines
- VMM is 'small' compared to the kernel… less LoC means less bugs means (~)more security.

Hypervisors

- ‘Practical’ attacks on IaaS clouds relied on side channels to detect co-location between attacker and victim VM
- E.g., we could correlate the performance of a shared resource
  - network RTT’s, cache performance
- After co-resident, make inferences about victim’s activities

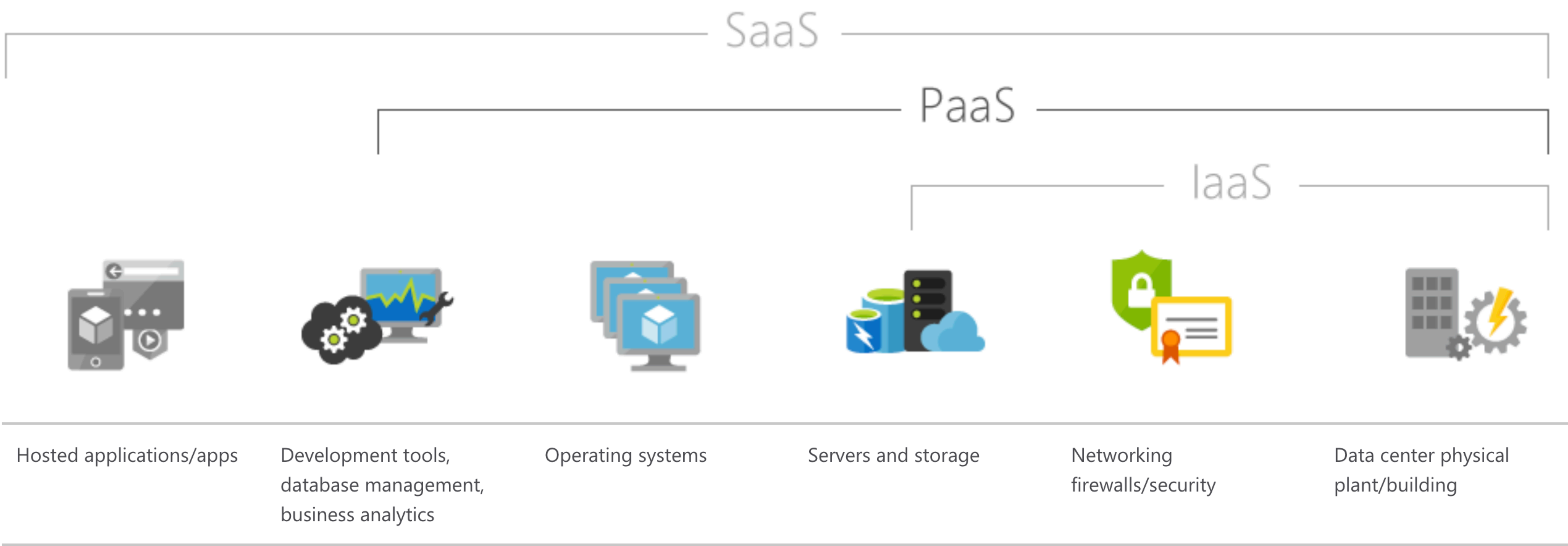


Hypervisors

- Strong isolation between different customer’s virtual machines
- VMM is ‘small’ compared to the kernel… less LoC means less bugs means (~)more security.
- High degree of flexibility… but did most customers really need it?

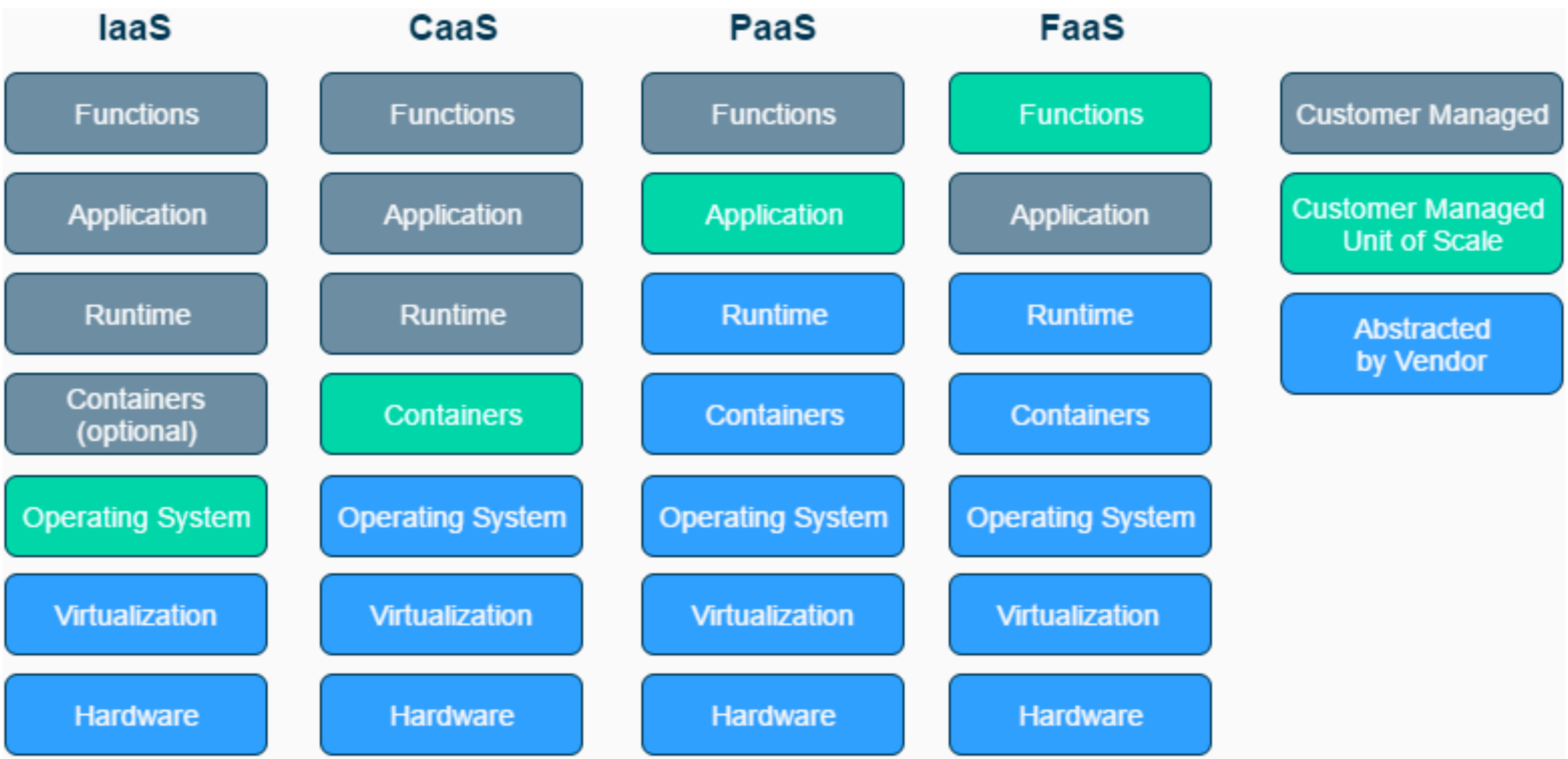
Cloud Computing (Gen 2)

- PaaS: Platform as a Service
- SaaS: Software as a Service



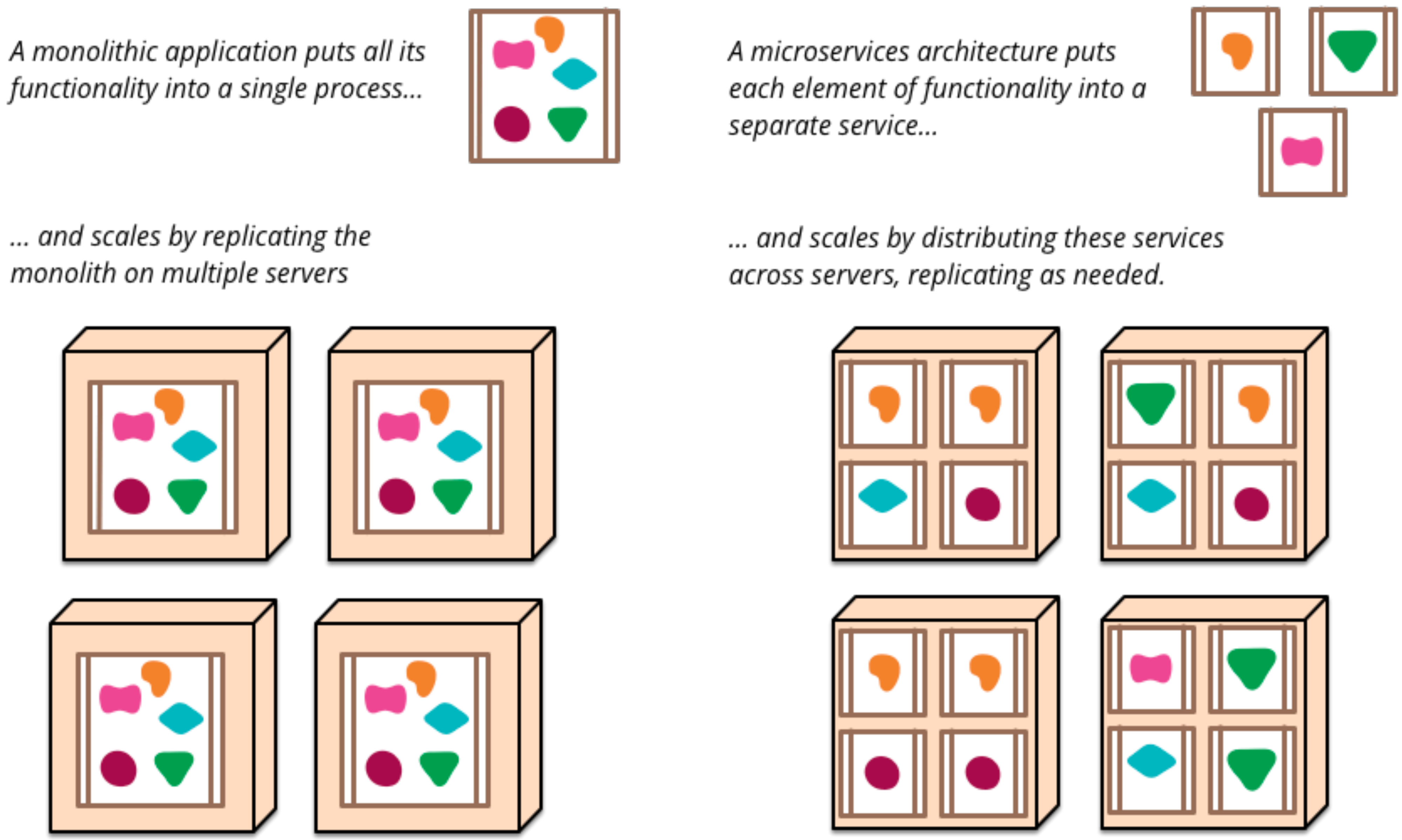
New Gen of Cloud Computing

- FaaS: Function as a Service



New Gen of Cloud Computing

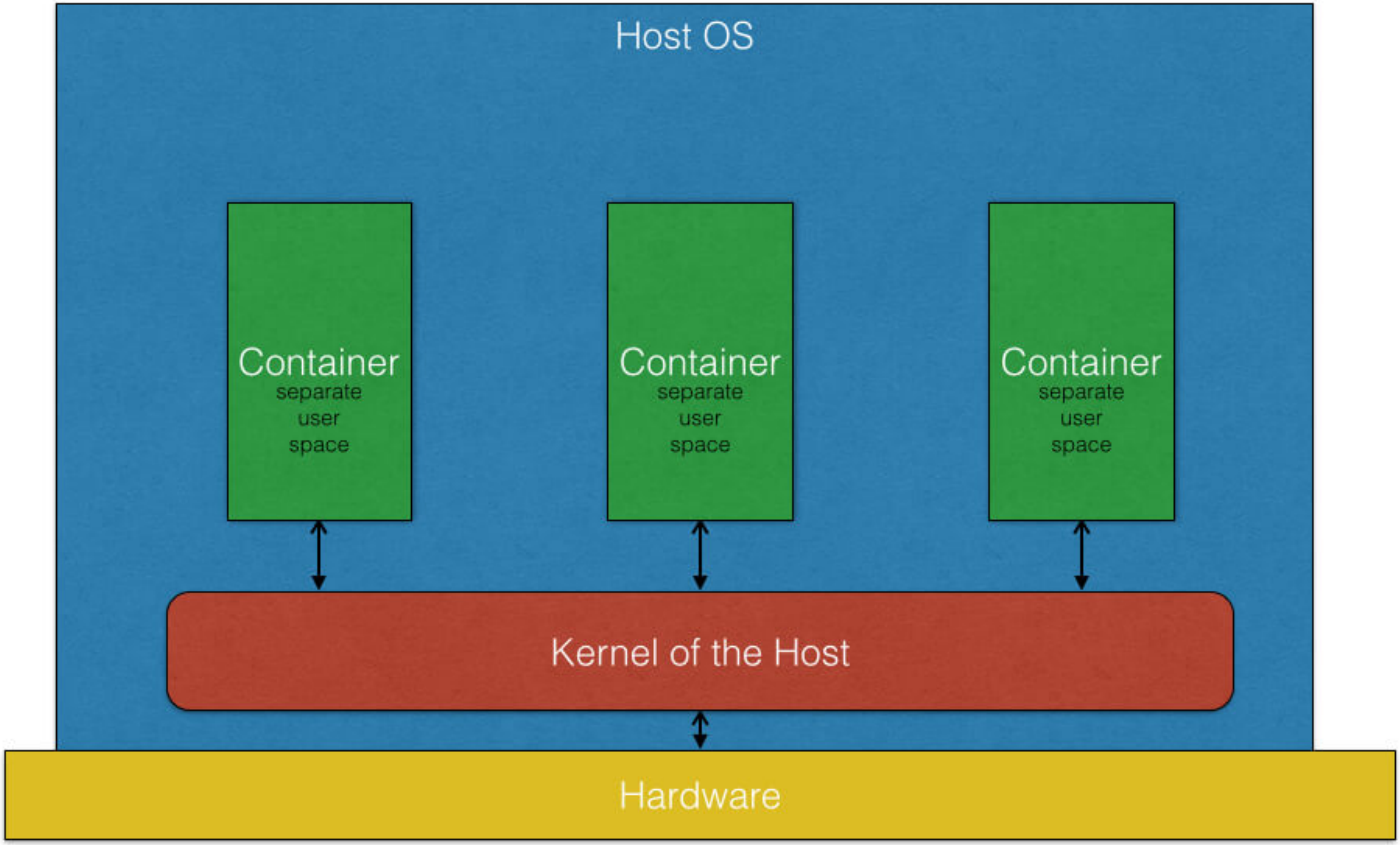
- Microservices



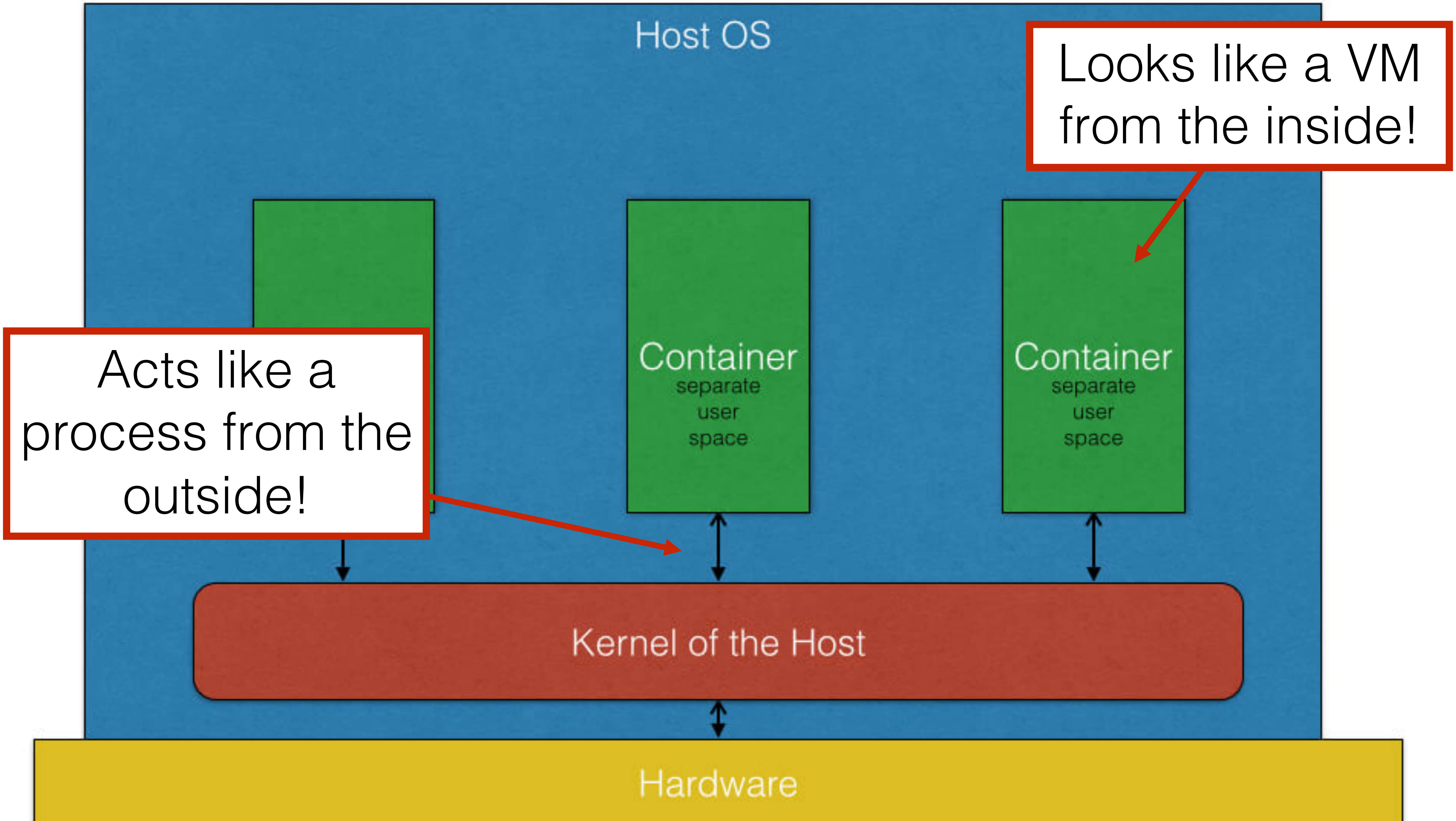
Enter Containers

- Rather than virtualize both user space and kernel space… why not just ‘virtualize’ user space?
- Meets the needs of most customers, who don’t require significant customization of the OS.
- Sometimes called ‘OS virtualization,’ which is highly misleading given our existing taxonomy of virtualization techniques
- Running natively on host, containers enjoy bare metal performance without reliance on advanced virtualization support from hardware.

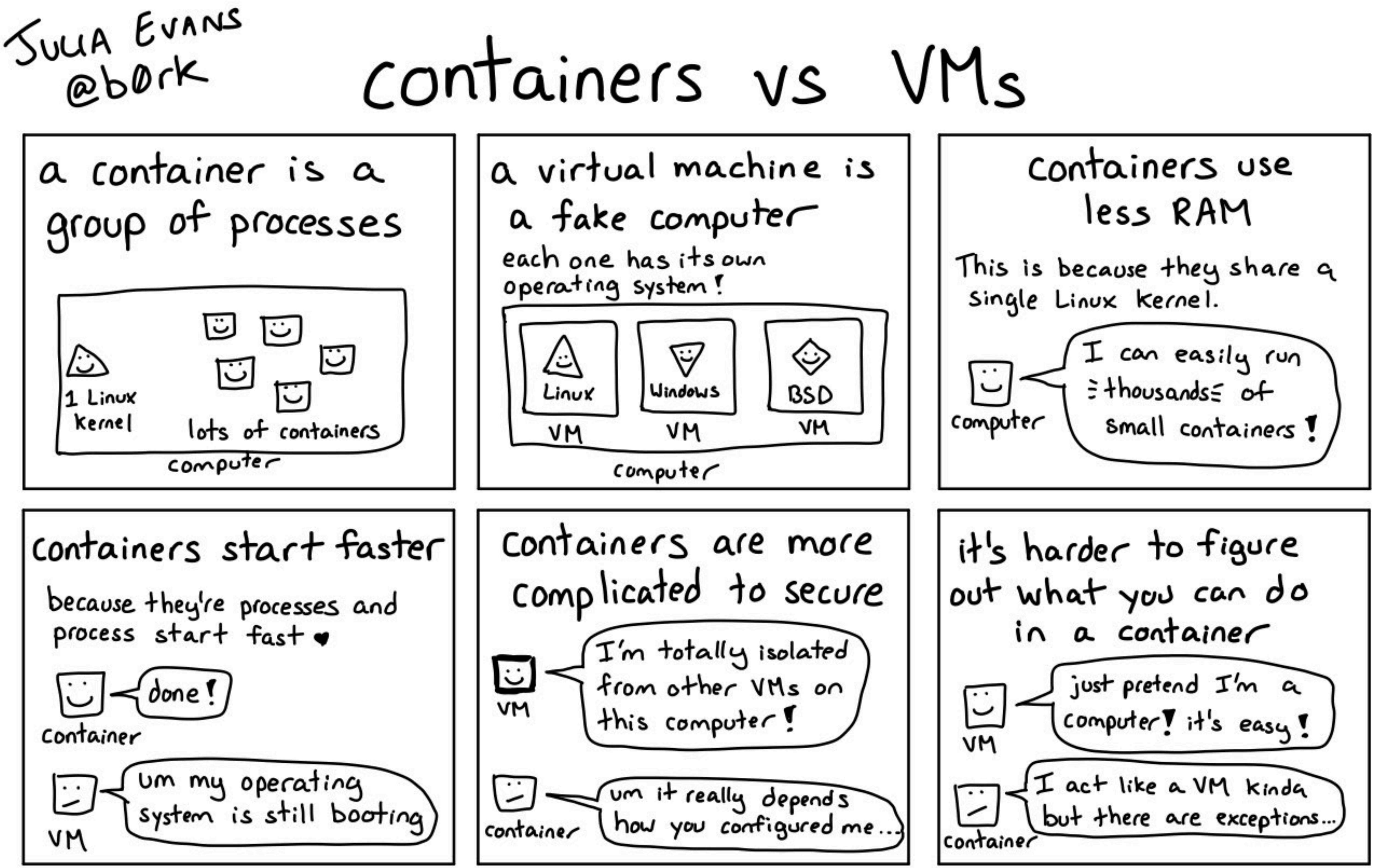
Enter Containers



Enter Containers



Containers are processes

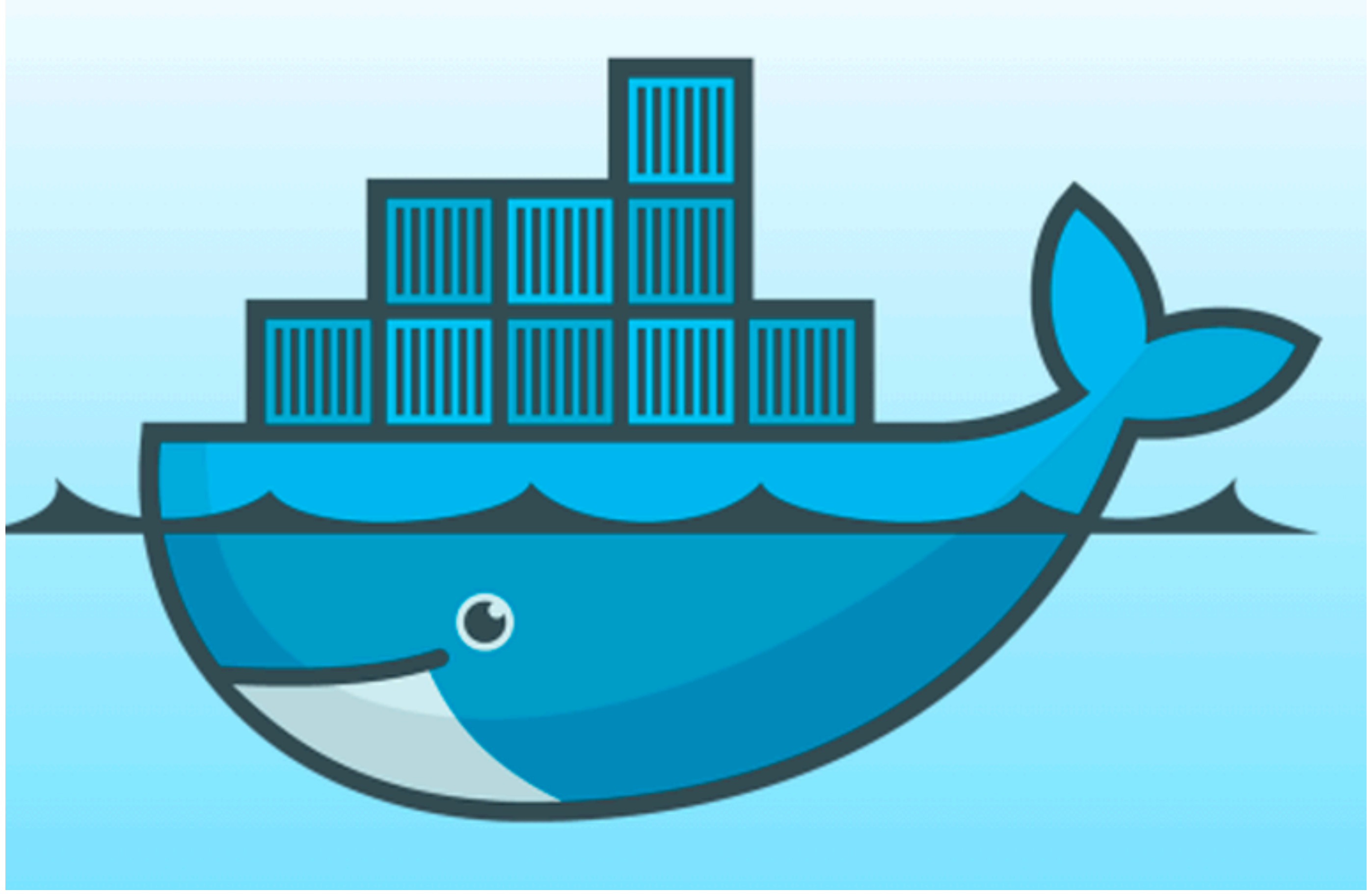


''Container is an old idea''

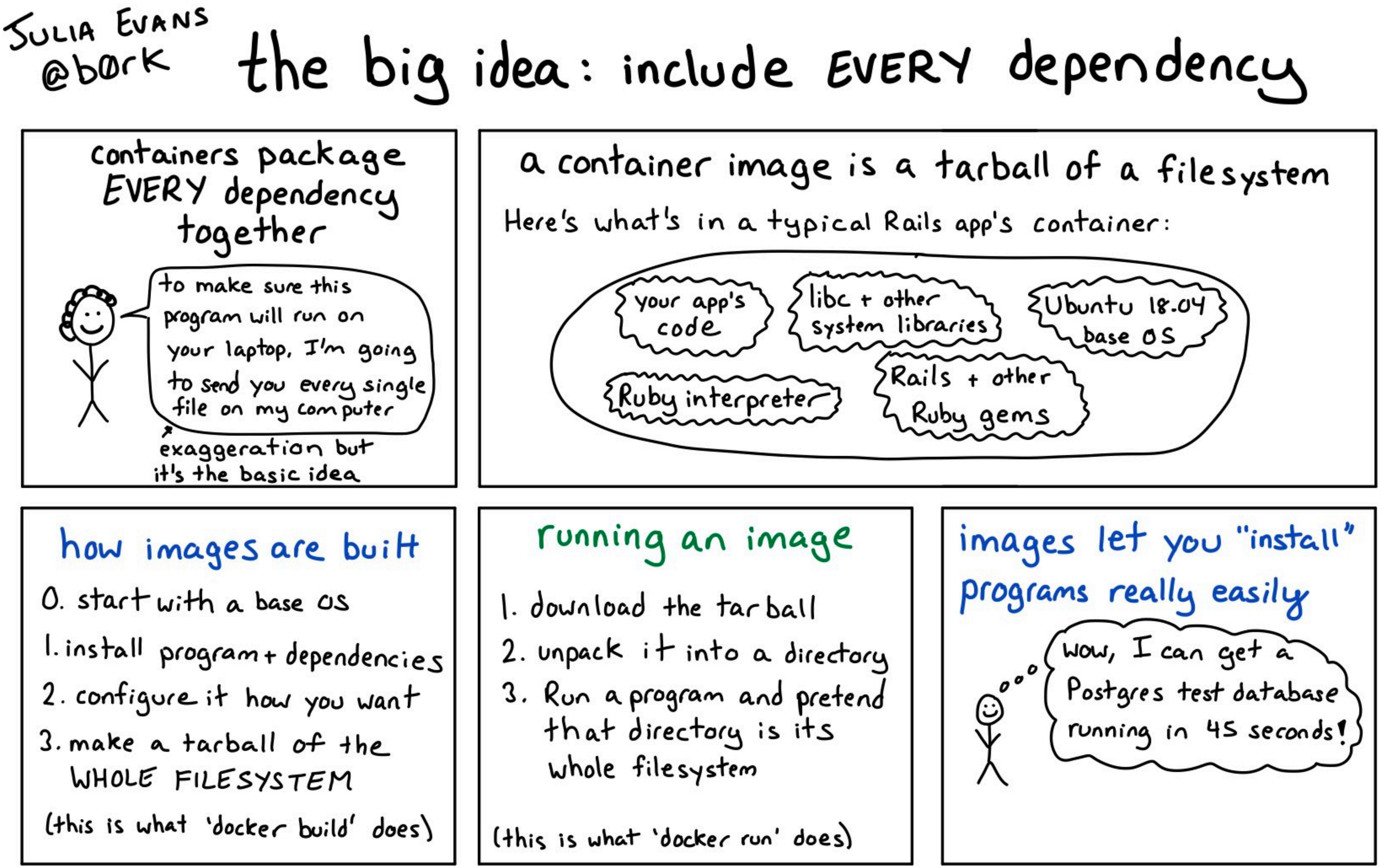
- You didn't heard of it cause it was not called "containers."
  - Linux containers
  - BSD Jails
  - Solaris Zones

Docker's Big Idea

- Build, Ship, and Run App, Anywhere
- Debug your app, not your environment -- Securely build and share any application, anywhere



Docker's Big Idea



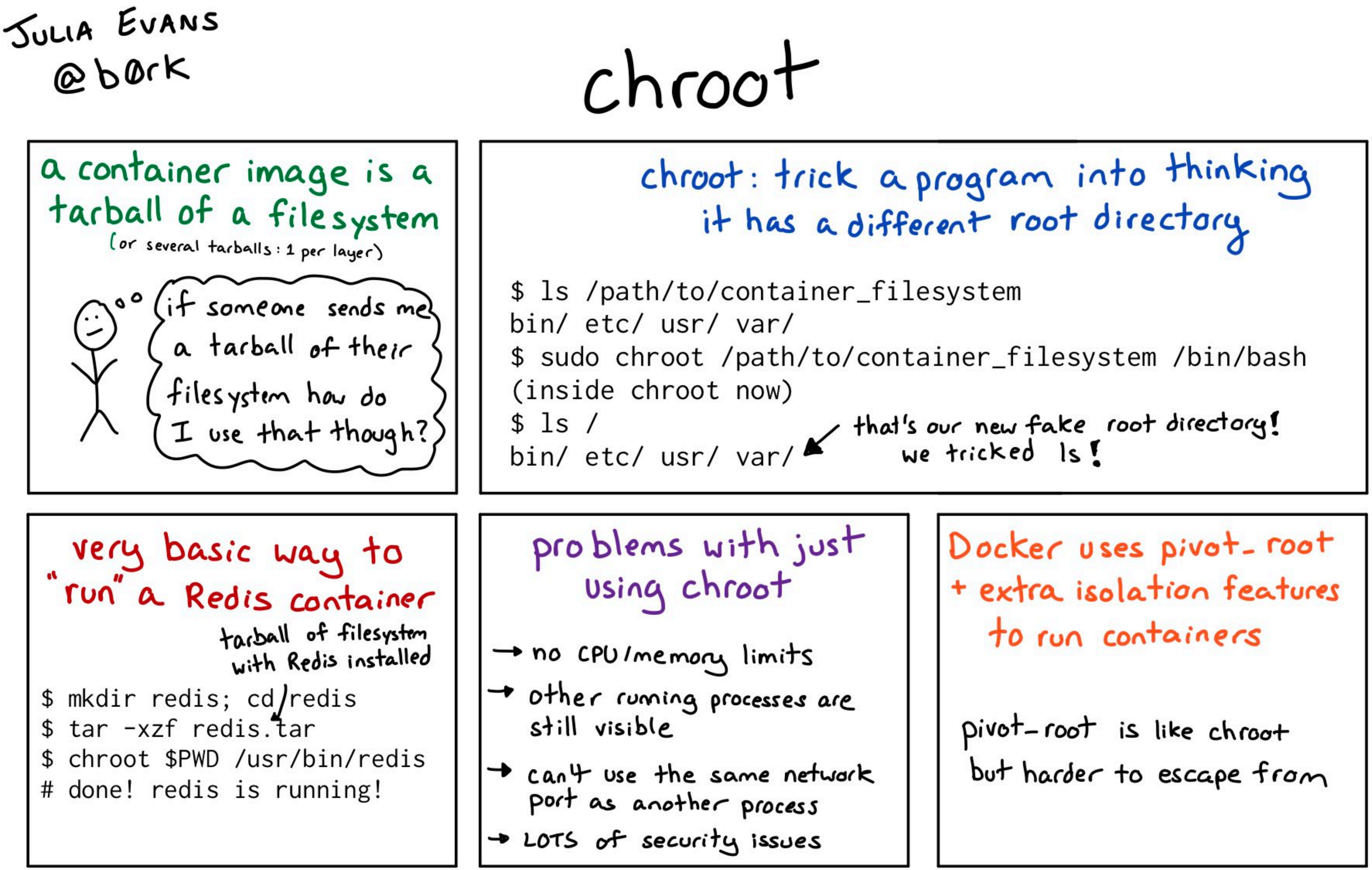
OS Support for Containers

- Linux Containers (LXC):
  - chroot
  - namespace
    - PID, Network, User, IPC, uts, mount
  - cgroups for HW isolation
  - Security profiles and policies
    - Apparmor, SELinux, Seccomp

containers = chroot on steroids

- `chroot` changes the apparent root directory for a given process and all of its children
- An old idea! POSIX call dating back to 1979
- Not intended to defend against privileged attackers… they still have root access and can do all sorts of things to break out (like `chroot`'ing again)
- Hiding the true root FS isolates a lot; in *nix, file abstraction used extensively
- Does not completely hide processes, network, etc., though!

Chroot



JULIA EVANS
@b0rk
chroot
a container image is a tarball of a filesystem
(or several tarballs: 1 per layer)
if someone sends me a tarball of their filesystem how do I use that though?
chroot: trick a program into thinking it has a different root directory
$ ls /path/to/container_filesystem
bin/ etc/ usr/ var/
$ sudo chroot /path/to/container_filesystem /bin/bash
(inside chroot now)
$ ls /
bin/ etc/ usr/ var/
that's our new fake root directory! we tricked ls!
very basic way to "run" a Redis container
tarball of filesystem with Redis installed
$ mkdir redis; cd redis
$ tar -xzf redis.tar
$ chroot $PWD /usr/bin/redis
# done! redis is running!
problems with just using chroot
→ no CPU/memory limits
→ other running processes are still visible
→ can't use the same network port as another process
→ LOTS of security issues
Docker uses pivot_root + extra isolation features to run containers
pivot_root is like chroot but harder to escape from

Namespaces

- The key feature enabling containerization!
- Partition practically all OS functionalities so that different process domains see different things
- <u>Mount (mnt)</u>: Controls mount points
- <u>Process ID (pid)</u>: Exposes a new set of process IDs distinct from other namespaces (i.e., the hosts)
- <u>Network (net)</u>: Dedicated network stack per container; each interface present in exactly one namespace at a time.
- ….

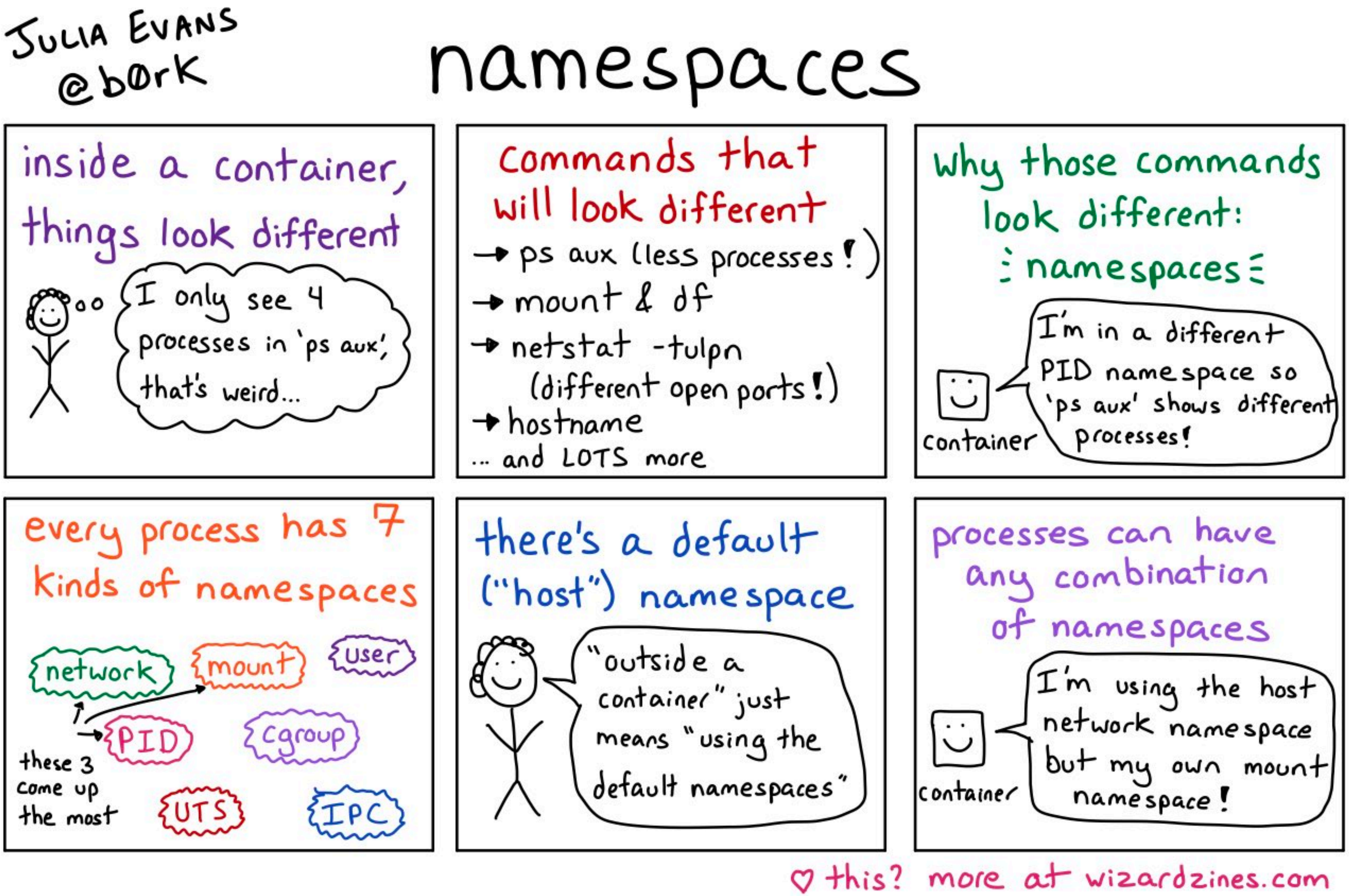
Namespaces

- The key feature enabling containerization!
- Partition practically all OS functionalities so that different process domains see different things
- <u>Interprocess Comm. (IPC):</u> Isolate processes from various methods of POSIX IPC.
    - e.g., no shared memory between containers!
- <u>UTS</u>: Allows the host to present different host/domain names to different containers.
- There's also a <u>User ID (user)</u> and <u>cgroup</u> namespace

User Namespace

- Like others, can provide a unique UID space to the container.
- More nuanced though — we can map UID 0 inside the container to UID 1000 outside; allows processes inside of container to think they're root.
- Enables containers to perform administration actions, e.g., adding more users, while remaining confined to their namespace.

Namespace

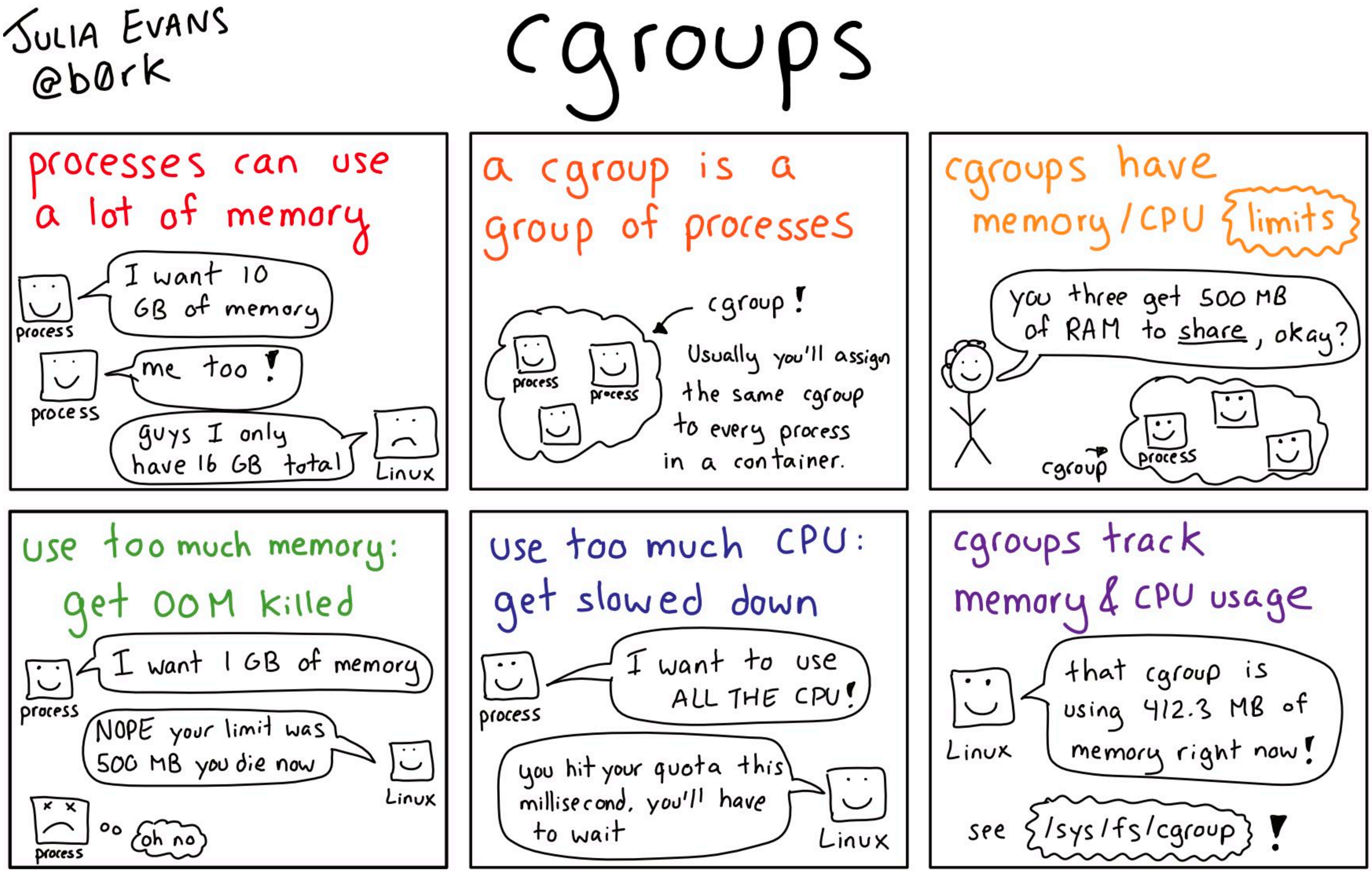


JULIA EVANS @b0rk
namespaces
inside a container, things look different
I only see 4 processes in 'ps aux', that's weird...
commands that will look different
→ ps aux (less processes!)
→ mount & df
→ netstat -tulpn (different open ports!)
→ hostname
... and LOTS more
why those commands look different:
namespaces
I'm in a different PID namespace so 'ps aux' shows different processes!
container
every process has 7 kinds of namespaces
network
mount
user
PID
cgroup
these 3 come up the most
UTS
IPC
there's a default ("host") namespace
"outside a container" just means "using the default namespaces"
processes can have any combination of namespaces
I'm using the host network namespace but my own mount namespace!
container
♡ this? more at wizardzines.com

cgroups

- Limit, track, and isolate utilization of hardware resources including CPU, memory, and disk.
- Important for ensuring QoS between customers! Protects against bad neighbors
- Features:
  - Resource limitation
  - Prioritization
  - Accounting (for billing customers!)
  - Control, e.g., freezing groups
- The cgroup namespace prevents containers from viewing or modifying their own group assignment

cgroups



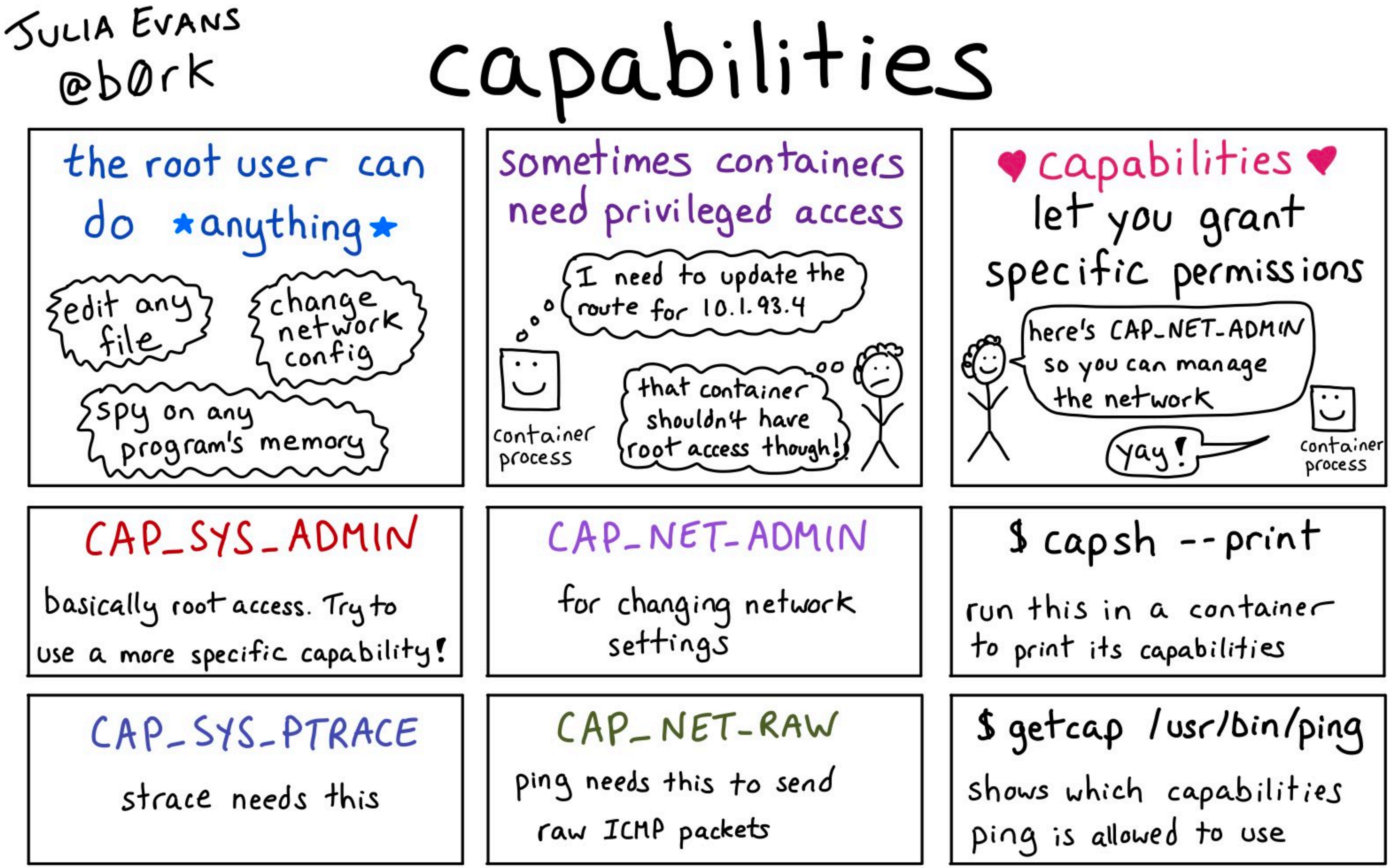
JULIA EVANS
@b0rk
cgroups
processes can use a lot of memory
I want 10 GB of memory
process
me too!
process
guys I only have 16 GB total
Linux
a cgroup is a group of processes
cgroup!
process
process
Usually you'll assign the same cgroup to every process in a container.
cgroups have memory/CPU limits
you three get 500 MB of RAM to share, okay?
cgroup
process
use too much memory: get OOM killed
I want 1 GB of memory
process
NOPE your limit was 500 MB you die now
Linux
process
oo
oh no
use too much CPU: get slowed down
I want to use ALL THE CPU!
process
you hit your quota this millisecond, you'll have to wait
Linux
cgroups track memory & CPU usage
that cgroup is using 412.3 MB of memory right now!
Linux
see /sys/fs/cgroup!

Container Security?

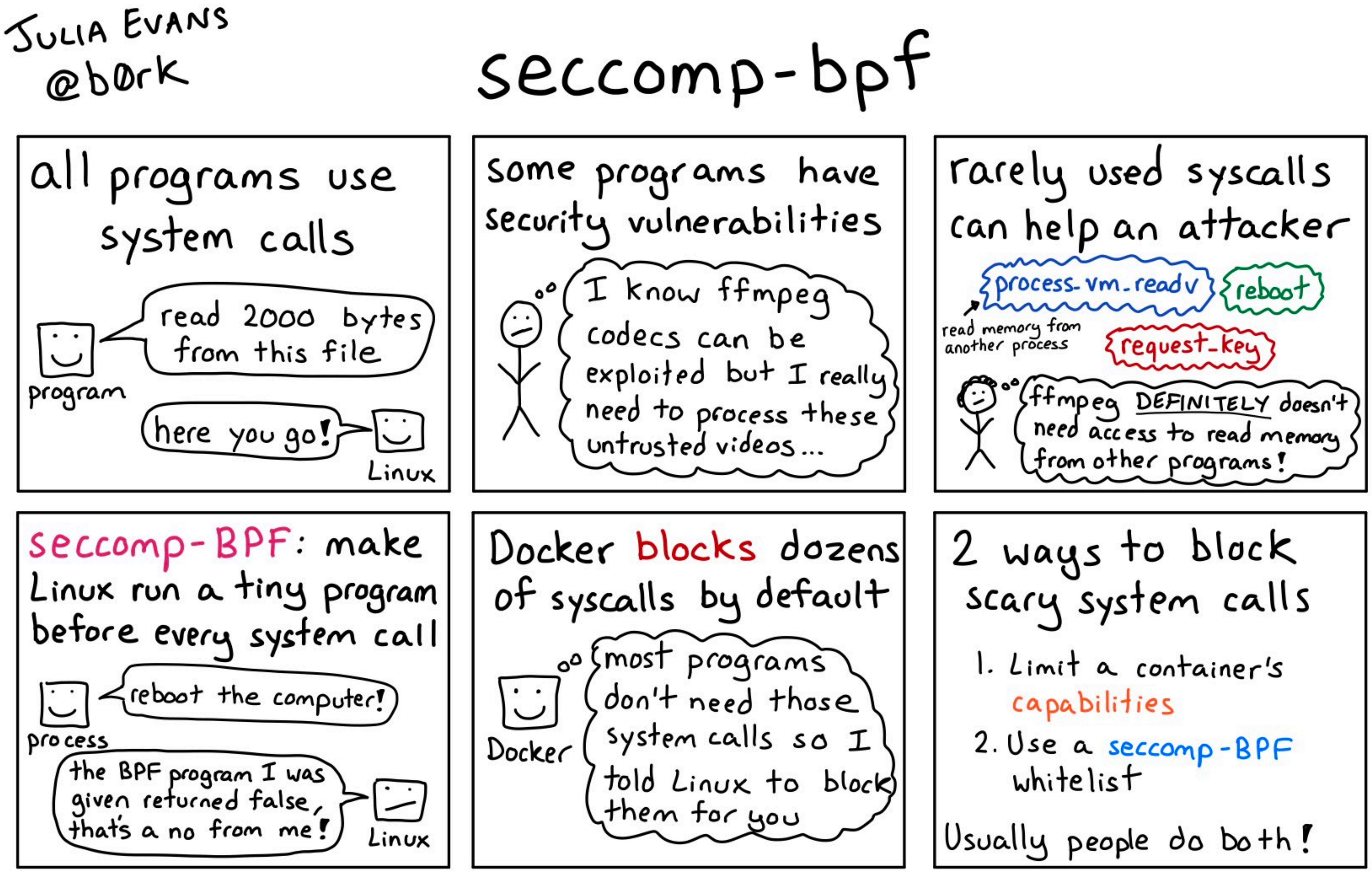
"Containers do not contain." - Dan Walsh (SELinux contributor)

- In a nutshell, it's *real hard* to prove that every feature of the operating system is namespaced.
  - /sys? /proc? /dev? LKMs? kernel keyrings?
- Root access to any of these enables pwning the host
- Solution? Just don't forget about MAC; at this point SELinux pretty good support for namespace labeling.
- SELinux and Namespaces actually synergize nicely; *much* easier to express a correct isolation policy over a coarse-grained namespace than, say, individual processes

Capabilities



Seccomp-bpf



JULIA EVANS
@b0rk
seccomp-bpf
all programs use system calls
read 2000 bytes from this file
program
here you go!
Linux
some programs have security vulnerabilities
I know ffmpeg codecs can be exploited but I really need to process these untrusted videos...
rarely used syscalls can help an attacker
process_vm_readv
reboot
read memory from another process
request_key
ffmpeg DEFINITELY doesn't need access to read memory from other programs!
seccomp-BPF: make Linux run a tiny program before every system call
reboot the computer!
process
the BPF program I was given returned false, that's a no from me!
Linux
Docker blocks dozens of syscalls by default
most programs don't need those system calls so I told Linux to block them for you
Docker
2 ways to block scary system calls
1. Limit a container's capabilities
2. Use a seccomp-BPF whitelist
Usually people do both!

Linux Security Modules

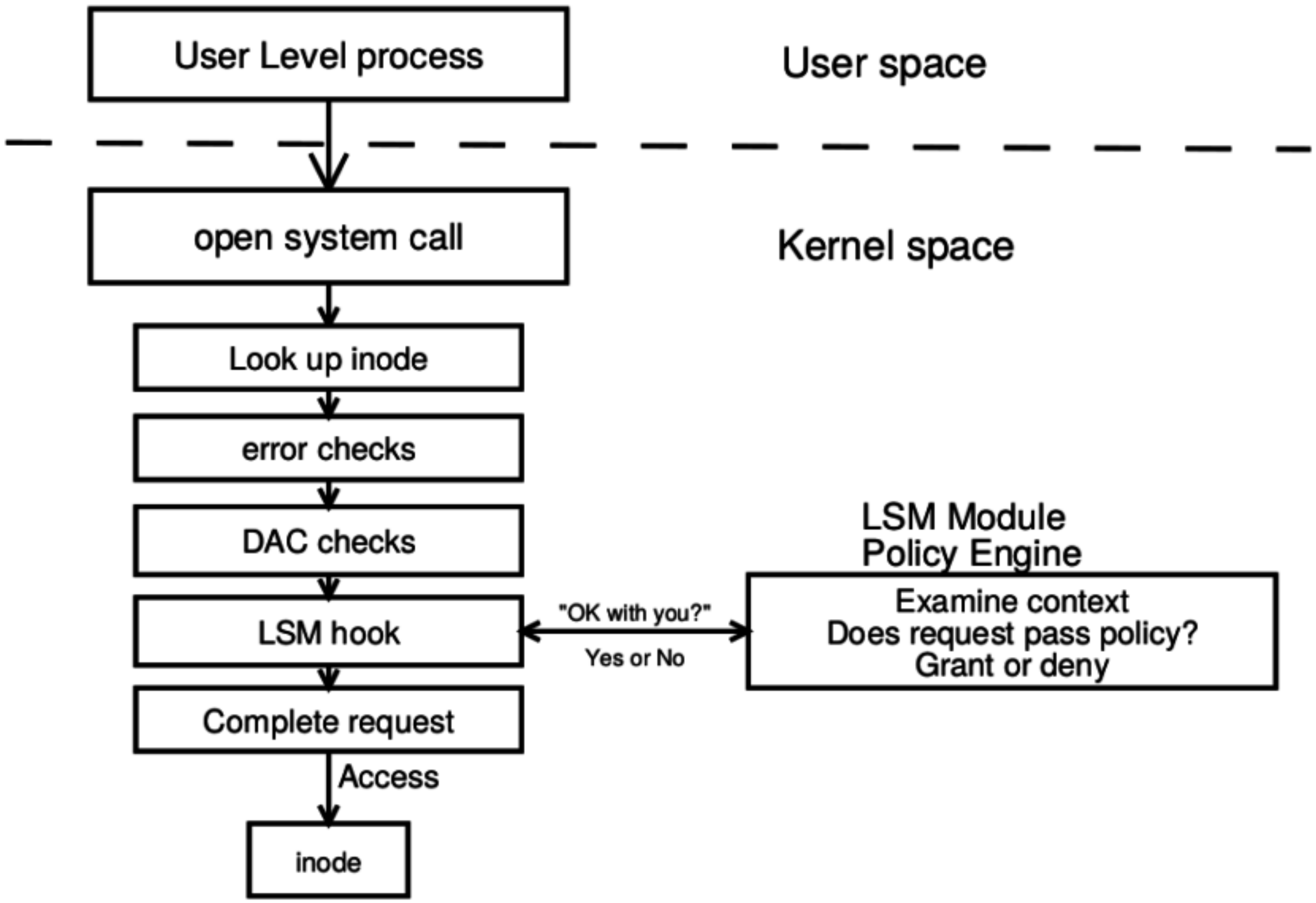
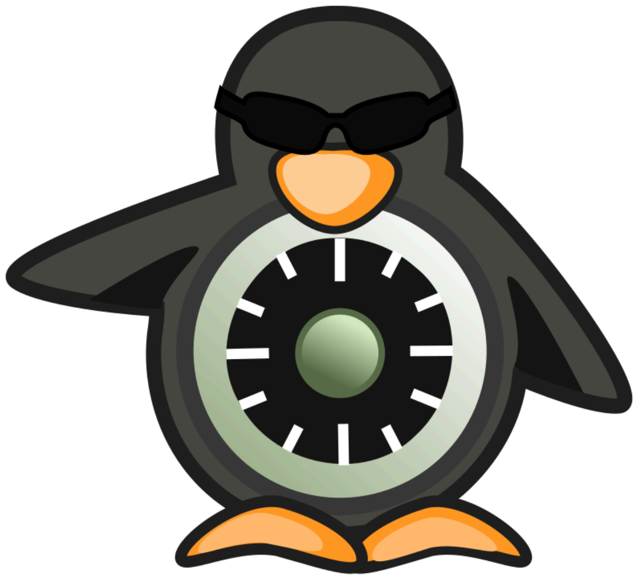


Figure 1: LSM Hook Architecture



DIY container

JULIA EVANS @b0rk

containers aren't magic

These 15 lines of bash will start a container running the fish shell. Try it!
(download this script at bit.ly/containers-arent-magic)

```
wget bit.ly/fish-container -O fish.tar                # 1. download the image
mkdir container-root; cd container-root               #
tar -xf ../fish.tar                                   # 2. unpack image into a directory
cgroup_id="cgroup_$(shuf -i 1000-2000 -n 1)"          # 3. generate random cgroup name
cgcreate -g "cpu,cpuacct,memory:$cgroup_id"           # 4. make a cgroup &
cgset -r cpu.shares=512 "$cgroup_id"                  #    set CPU/memory limits
cgset -r memory.limit_in_bytes=1000000000 \           #
      "$cgroup_id"                                    #
cgexec -g "cpu,cpuacct,memory:$cgroup_id" \           # 5. use the cgroup
    unshare -fmuipn --mount-proc \                    # 6. make + use some namespaces
    chroot "$PWD" \                                   # 7. change root directory
    /bin/sh -c "                                      #
        /bin/mount -t proc proc /proc &&              # 8. use the right /proc
        hostname container-fun-times &&               # 9. change the hostname
        /usr/bin/fish"                                # 10. finally, start fish!
```

Summary

Julia Evans
@b0rk

container kernel features

containers are implemented using these Linux kernel features

You can use any of these on their own. When we use them all we call it a "container".

♥ pivot_root ♥

set a process's root directory to a directory with the contents of the container image

★ cgroups ★

limit memory/CPU usage for a group of processes

Linux: only 500 MB of RAM for you!

♥ namespaces ♥

allow processes to have their own:

- network
- PIDs
- users
- hostname
- mounts
- + more

♥ seccomp-bpf ♥

security: prevent dangerous system calls

★ capabilities ★

security: avoid giving root access

★ overlay filesystems ★

optimization to reduce disk space used by containers which are using the same image

Takeaways

- Container support has existing in Linux for many years
- Foundations of containerization has been around for decades!
- Automating LXC for portability (i.e., Docker) has revolutionized cloud computing
- Lasting legacy of containers may be enabling the Function-as-a-Service revolution… cloud customers can now pay by the method invocation without any idle costs.