github.com/coreos/rkt

Jonathan Boulle
github.com/jonboulle
@baronboulle

CoreOS

Alban Crequy
github.com/alban

kinvolk.io
Core. Linux. Engineering.

Why rkt?

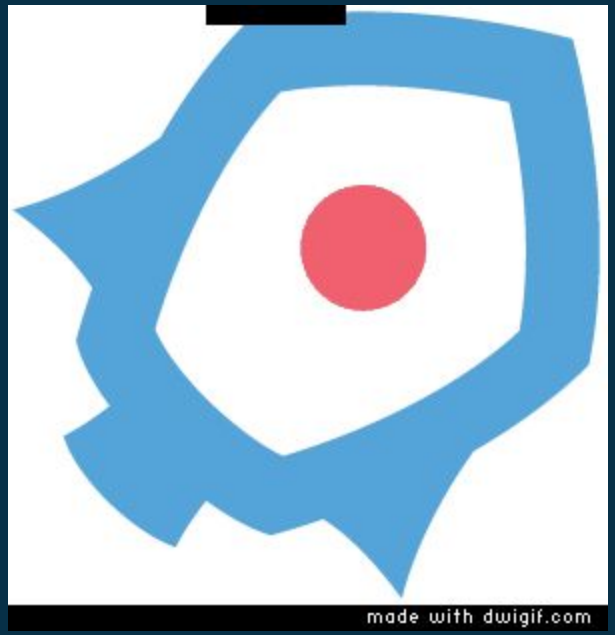
Open standards.
Composability.
See: last few talks…


CONTAINERS...
HISTORY.COM
imgflip.com

Why rkt?

Why this talk?

See: last few talks…


made with dwigif.com

rkt

a modern, secure container runtime

rkt

simple CLI tool

rkt

an implementation of appc

quick digression: appc

appc

App Container (appc)

github.com/appc

appc-dev@googlegroups.com

appc

github.com/appc/spec
github.com/appc/acbuild
github.com/appc/docker2aci
github.com/appc/cni
github.com/appc/...

appc

github.com/appc/spec

github.com/appc/acbuild

github.com/appc/docker2aci

github.com/appc/cni

github.com/appc/...

appc spec in a nutshell

- Image Format (ACI)
  - what does an application consist of?
- Image Discovery
  - how can an image be located?
- Pods
  - how can applications be grouped and run?
- Executor (runtime)
  - what does the execution environment look like?

rkt

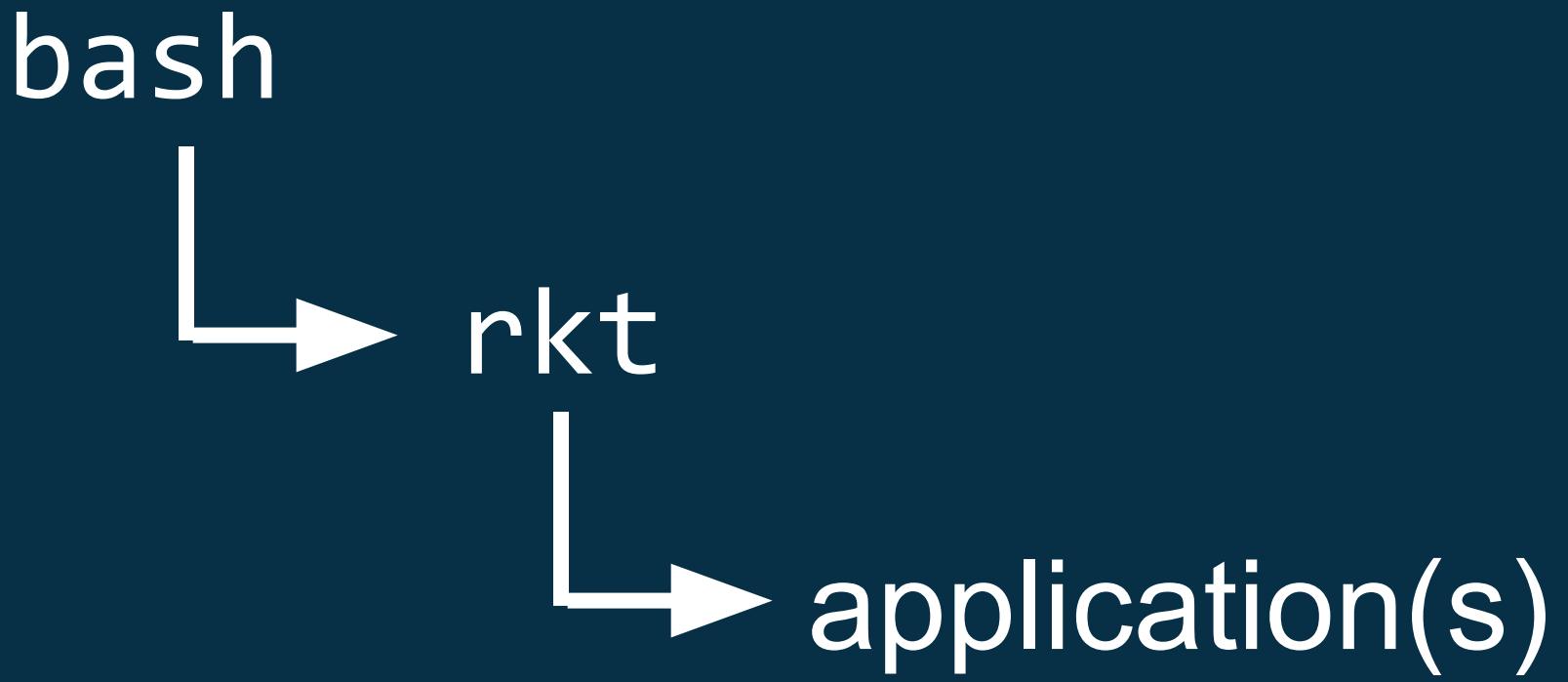
a modern, secure container runtime
simple CLI tool
an implementation of appc

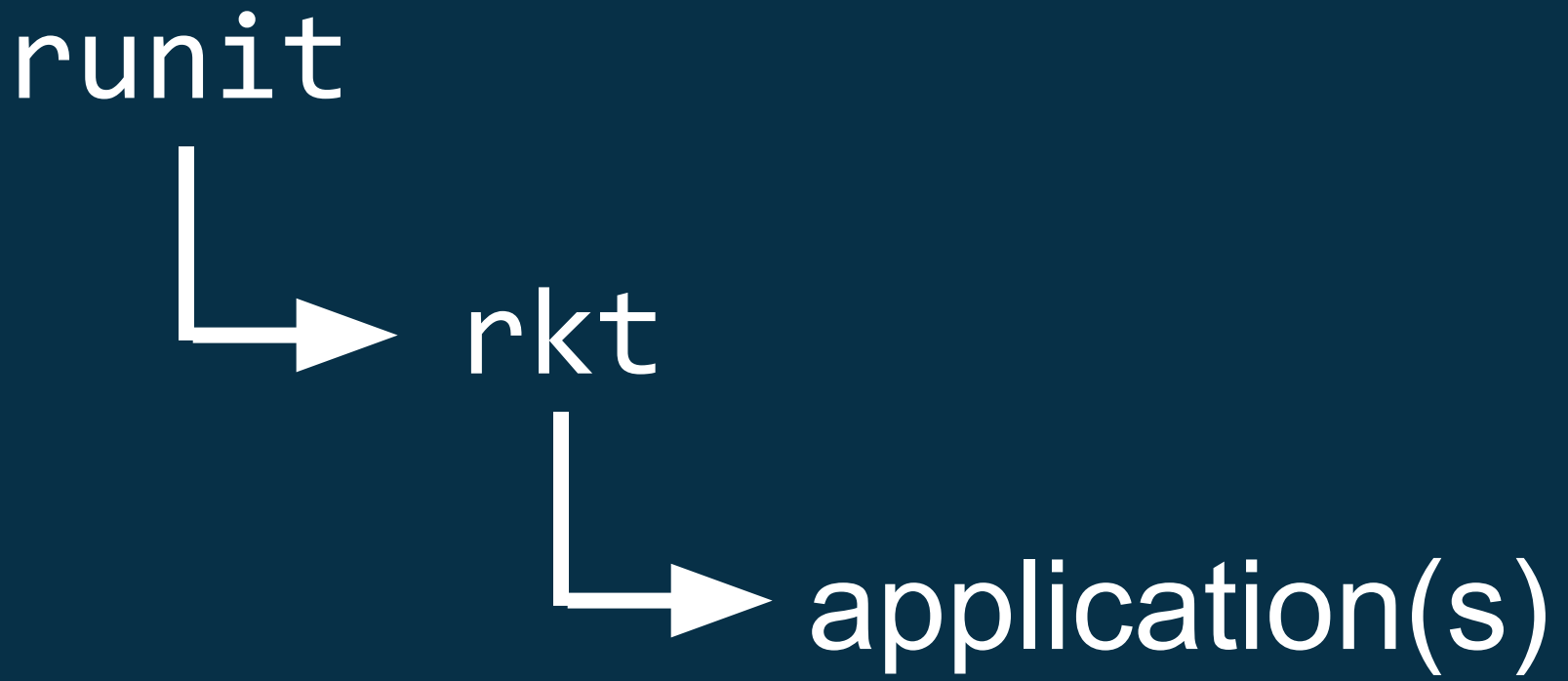
simple CLI tool

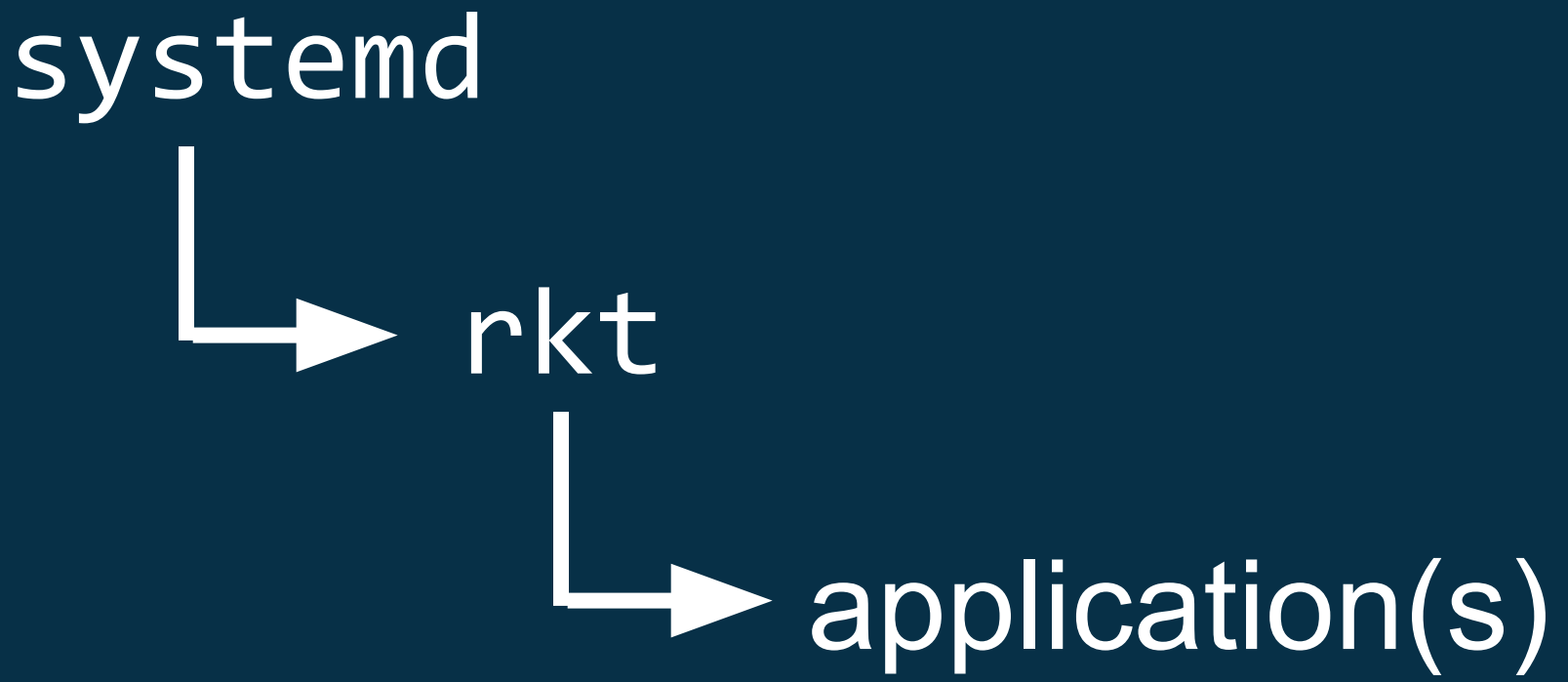
golang + Linux
self-contained
init system/distro agnostic

simple CLI tool

no daemon
no API*
apps run directly under spawning process

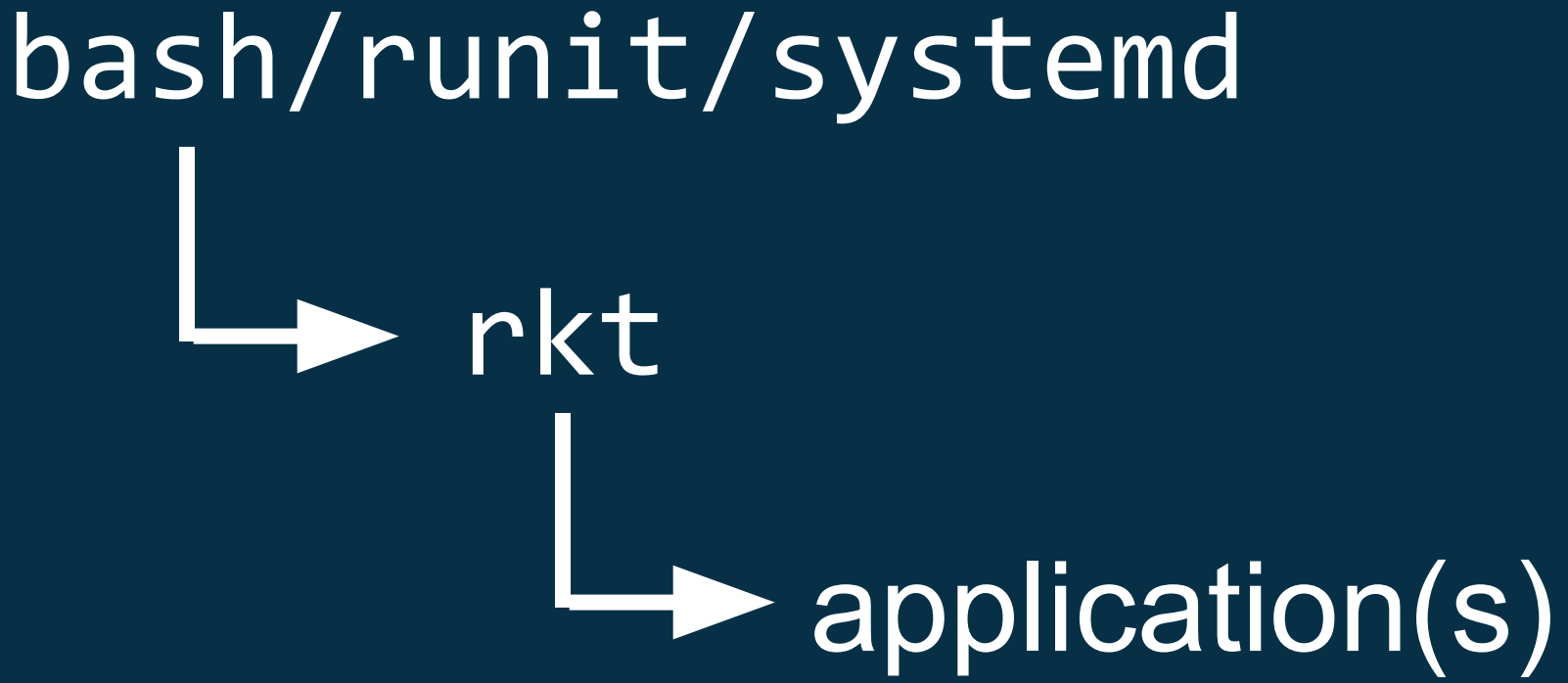

bash
rkt
application(s)

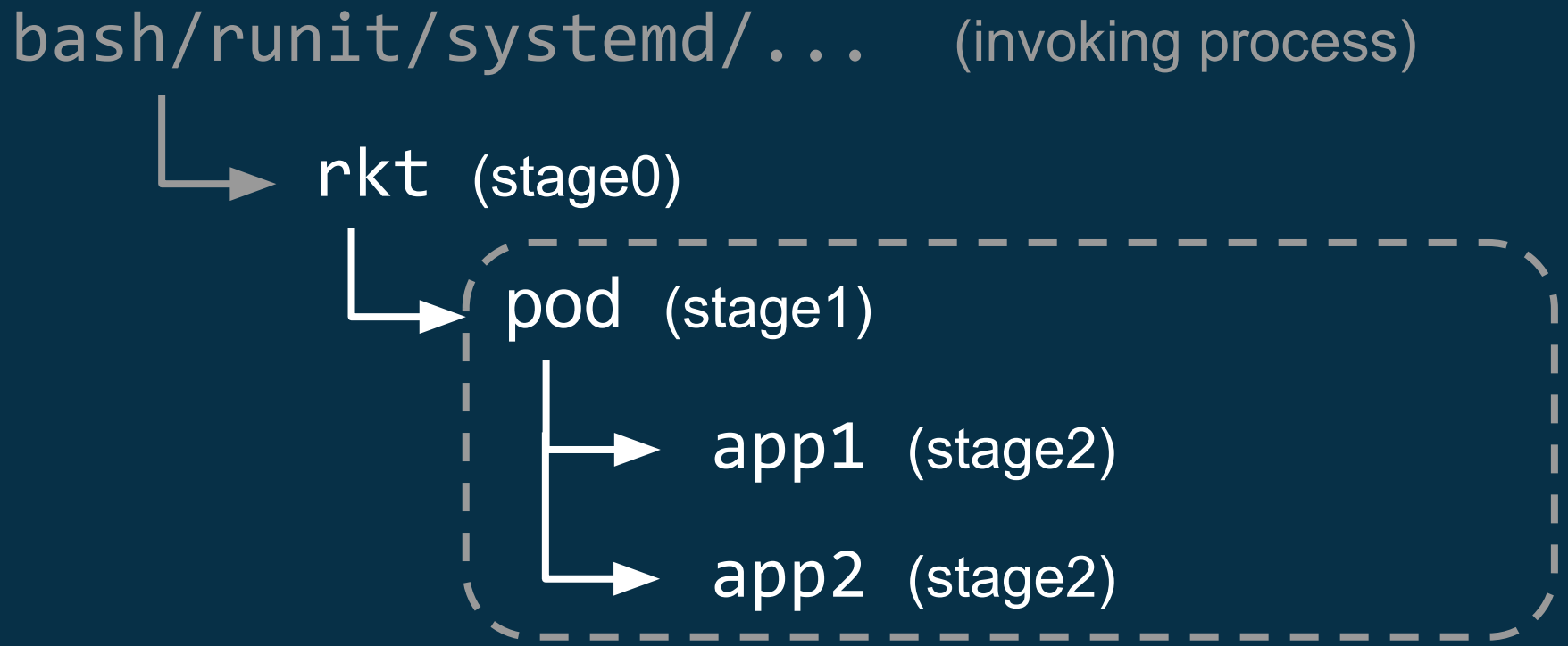

runit
rkt
application(s)

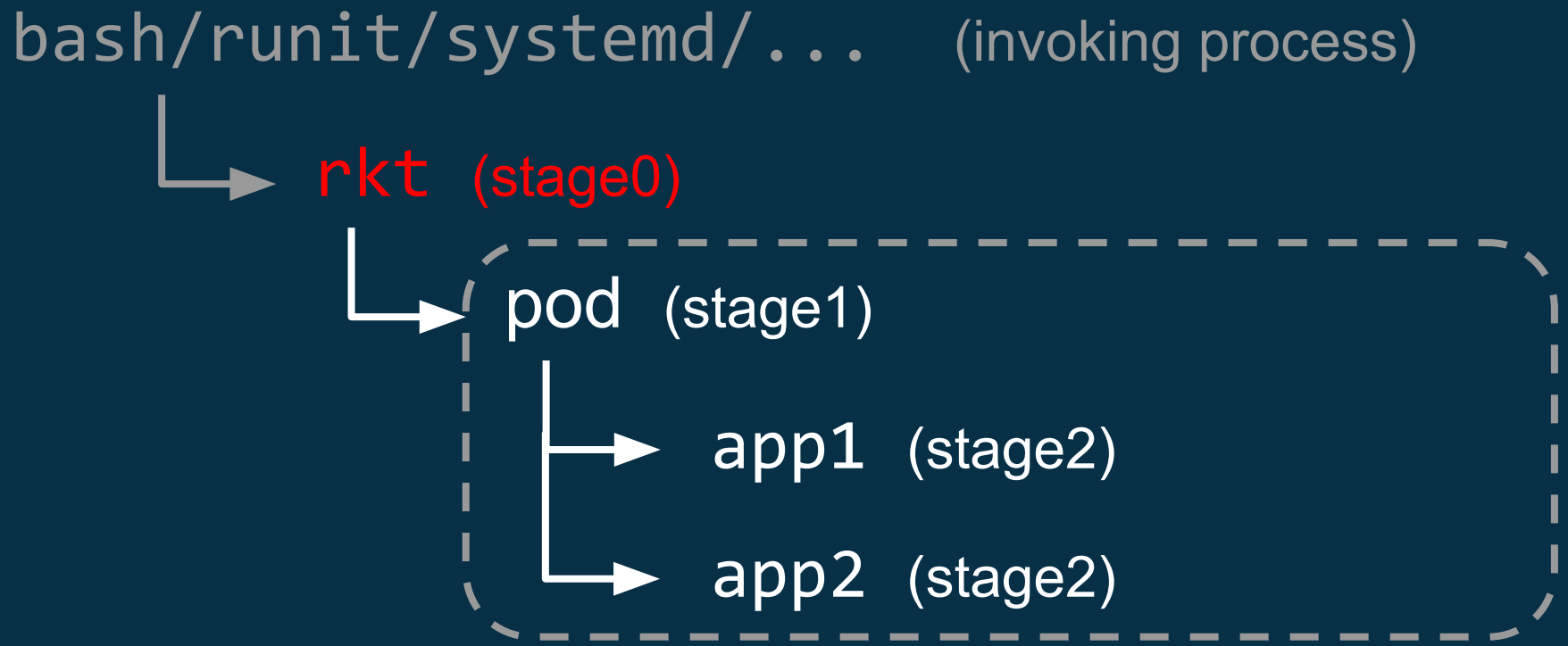

systemd
rkt
application(s)

rkt internals

modular architecture
execution divided into *stages*
stage0 → stage1 → stage2


bash/runit/systemd
rkt
application(s)


bash/runit/systemd/...
(invoking process)
rkt
(stage0)
pod
(stage1)
app1
(stage2)
app2
(stage2)


bash/runit/systemd/...
(invoking process)
rkt
(stage0)
pod
(stage1)
app1
(stage2)
app2
(stage2)

stage0 (`rkt` binary)

discover, fetch, manage application images
set up pod filesystems
commands to manage pod lifecycle

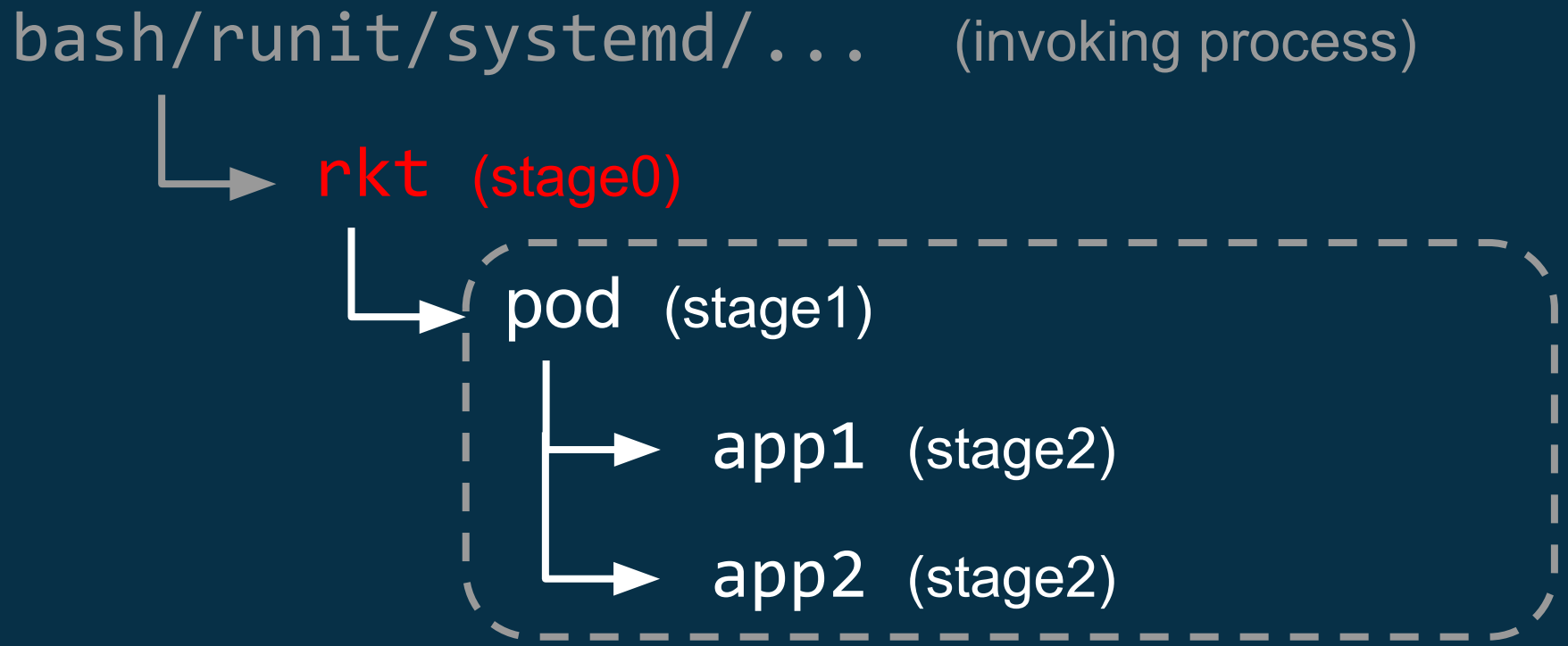
stage0 (rkt binary)

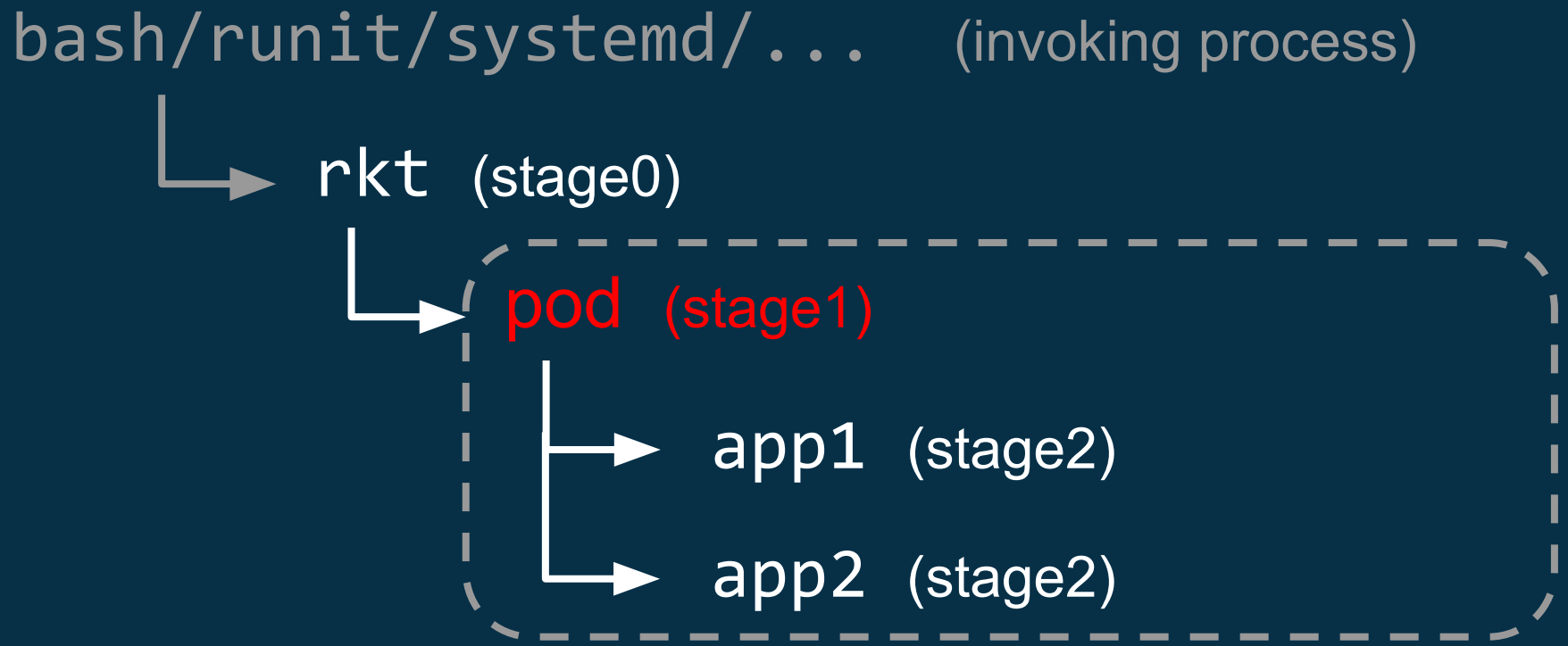
- rkt run
- rkt prepare
- rkt run-prepared
- rkt list
- rkt status
- ...

- rkt fetch
- rkt trust
- rkt image list
- rkt image export
- rkt image gc
- ...

stage0 (`rkt` binary)

file-based locking for concurrent operation
(e.g. `rkt gc`, `rkt list` for pods)
database + reference counting for images


bash/runit/systemd/...
(invoking process)
rkt
(stage0)
pod
(stage1)
app1
(stage2)
app2
(stage2)


bash/runit/systemd/...
(invoking process)
rkt
(stage0)
pod
(stage1)
app1
(stage2)
app2
(stage2)

stage1

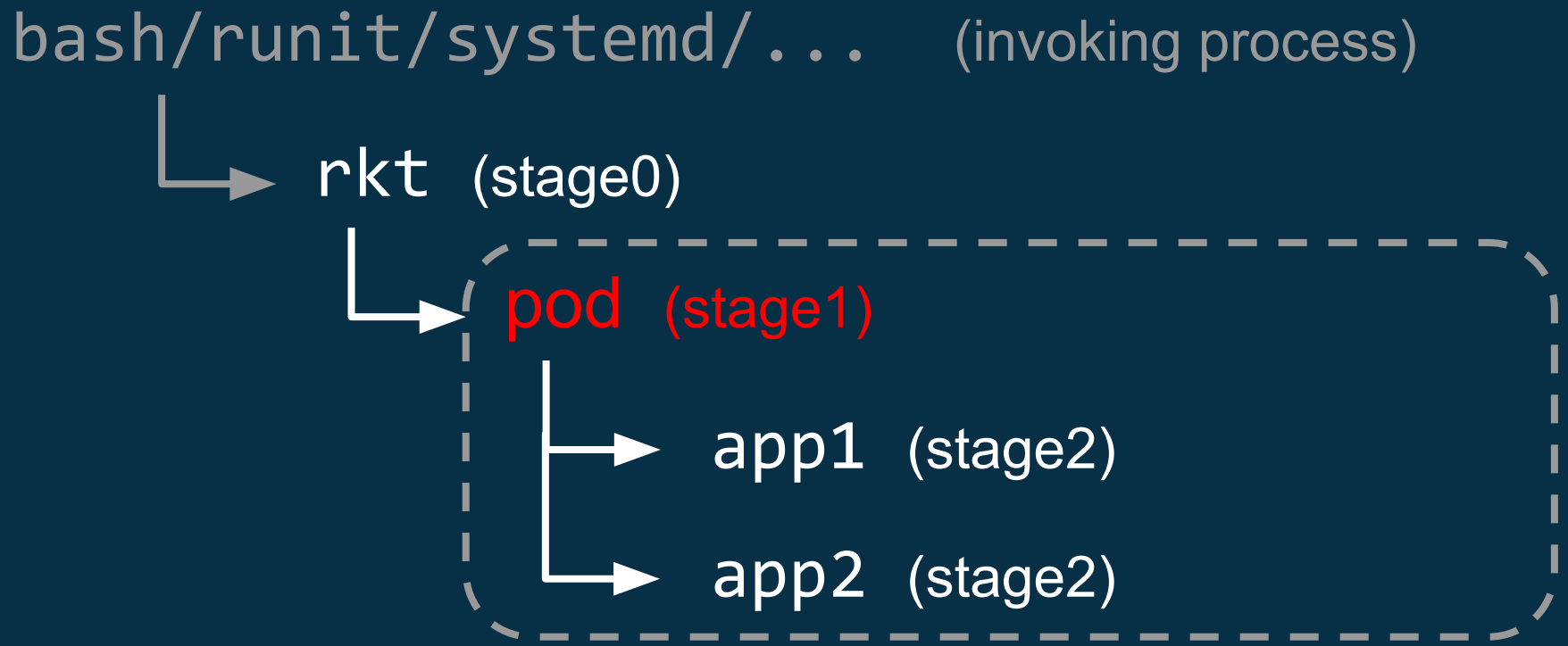
execution environment for pods
app process lifecycle management
isolators

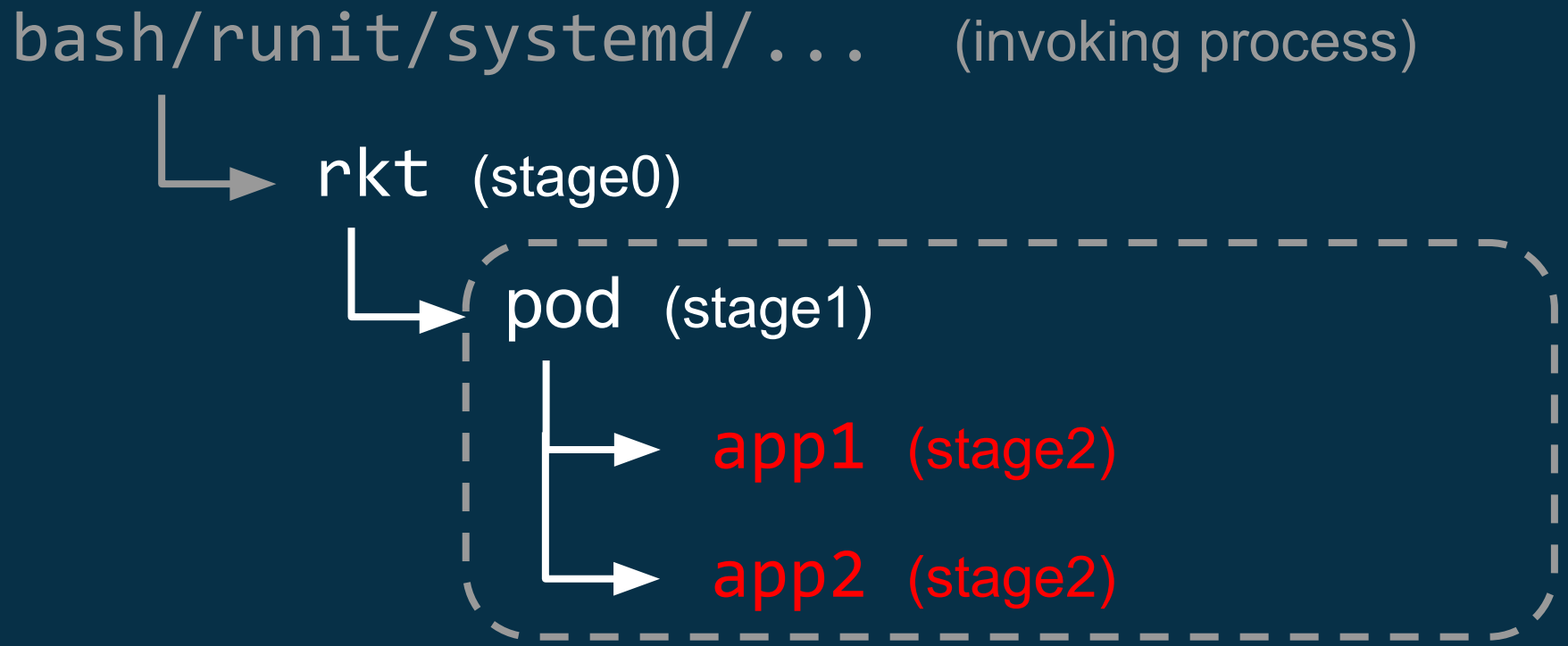
stage1 (swappable)

binary ABI with stage0
stage0 calls an `execve(stage1)`

stage1 (swappable)

- default implementation
  - based on systemd-nspawn+systemd
  - Linux namespaces + cgroups for isolation
- kvm implementation
  - based on lkvm+systemd
  - hardware virtualisation for isolation
- others?


bash/runit/systemd/...
(invoking process)
rkt
(stage0)
pod
(stage1)
app1
(stage2)
app2
(stage2)


bash/runit/systemd/...
(invoking process)
rkt (stage0)
pod (stage1)
app1 (stage2)
app2 (stage2)

stage2

actual app execution
independent filesystems (`chroot`)
shared namespaces, volumes, IPC, ...

rkt + systemd

The different ways rkt integrates with systemd

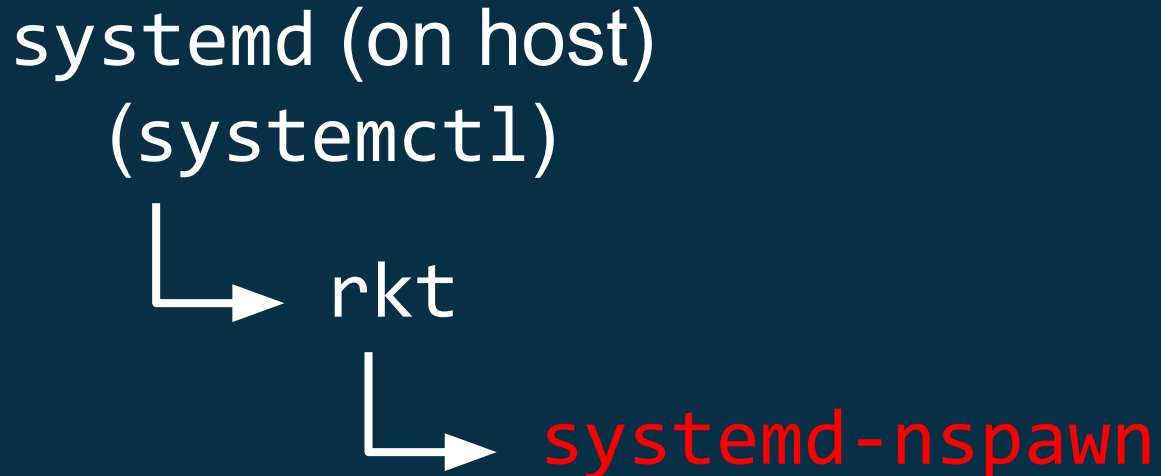
rkt

systemd (on host)
(systemctl)

rkt

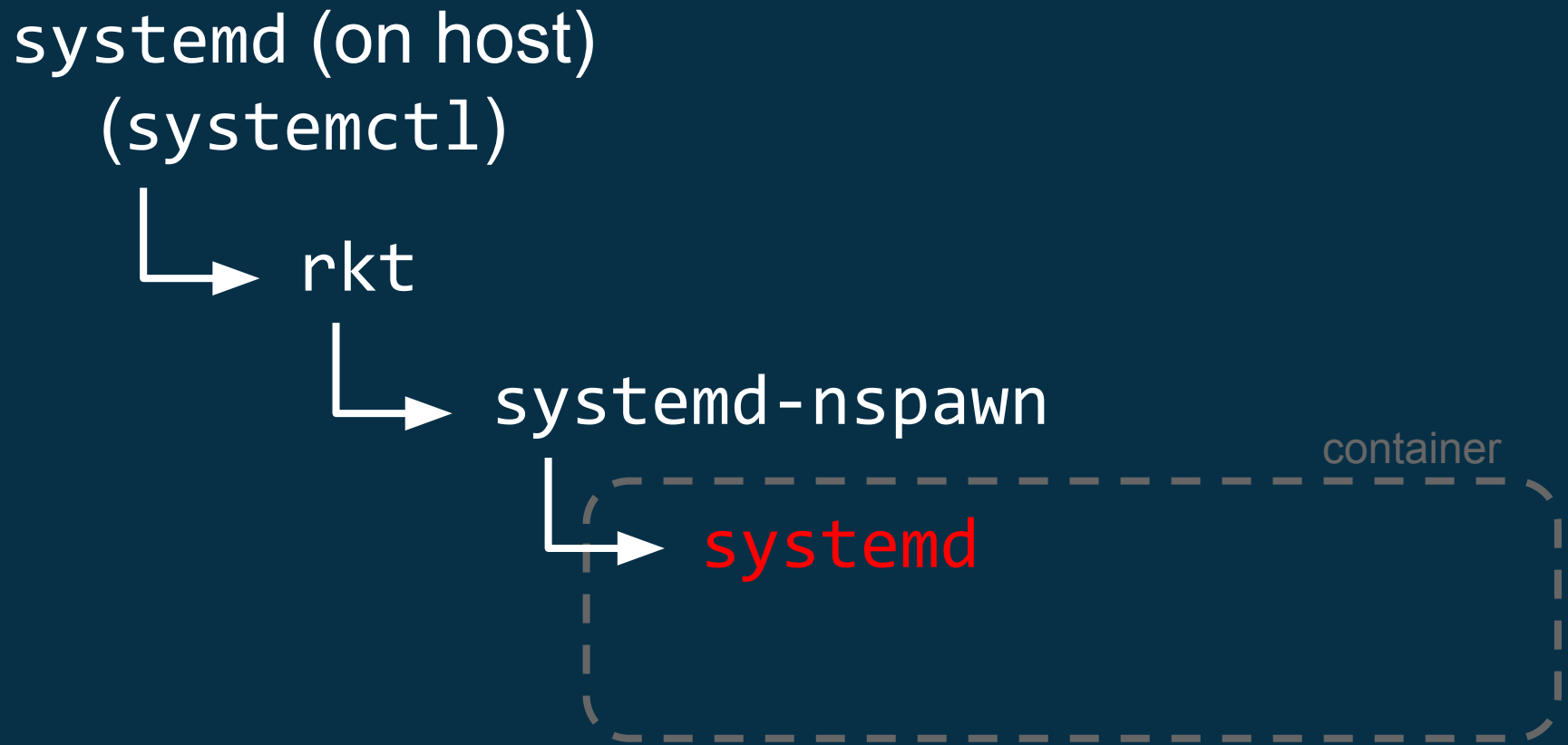
systemd (on host)

optional
"systemctl stop" just works
socket activation
pod-level isolators: CPUShares, MemoryLimit


systemd (on host)
(systemctl)
rkt
systemd-nspawn

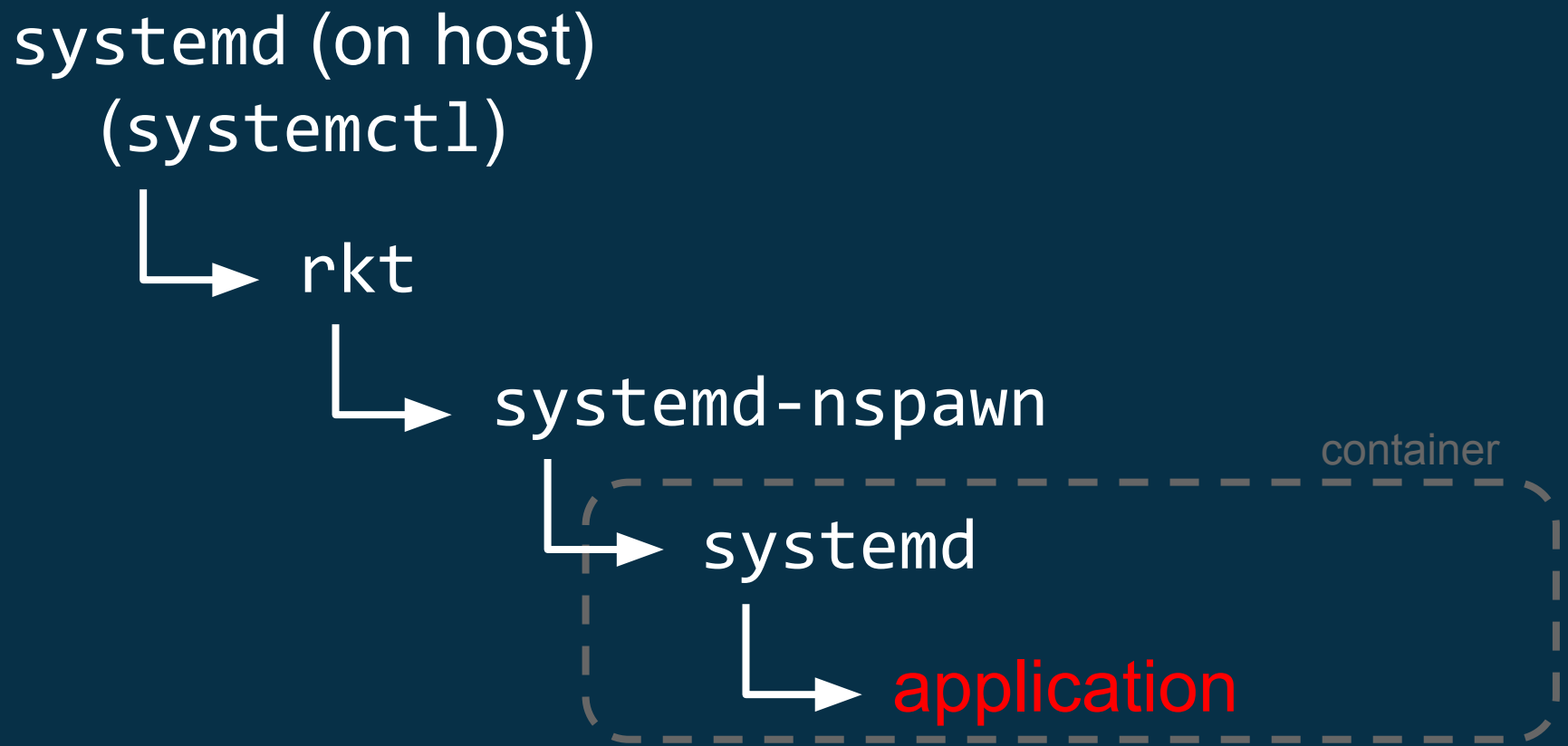
`systemd-nspawn`

default stage1, besides lkvm
taking care of most of the low-level things


systemd (on host)
(systemctl)
rkt
systemd-nspawn
container
systemd

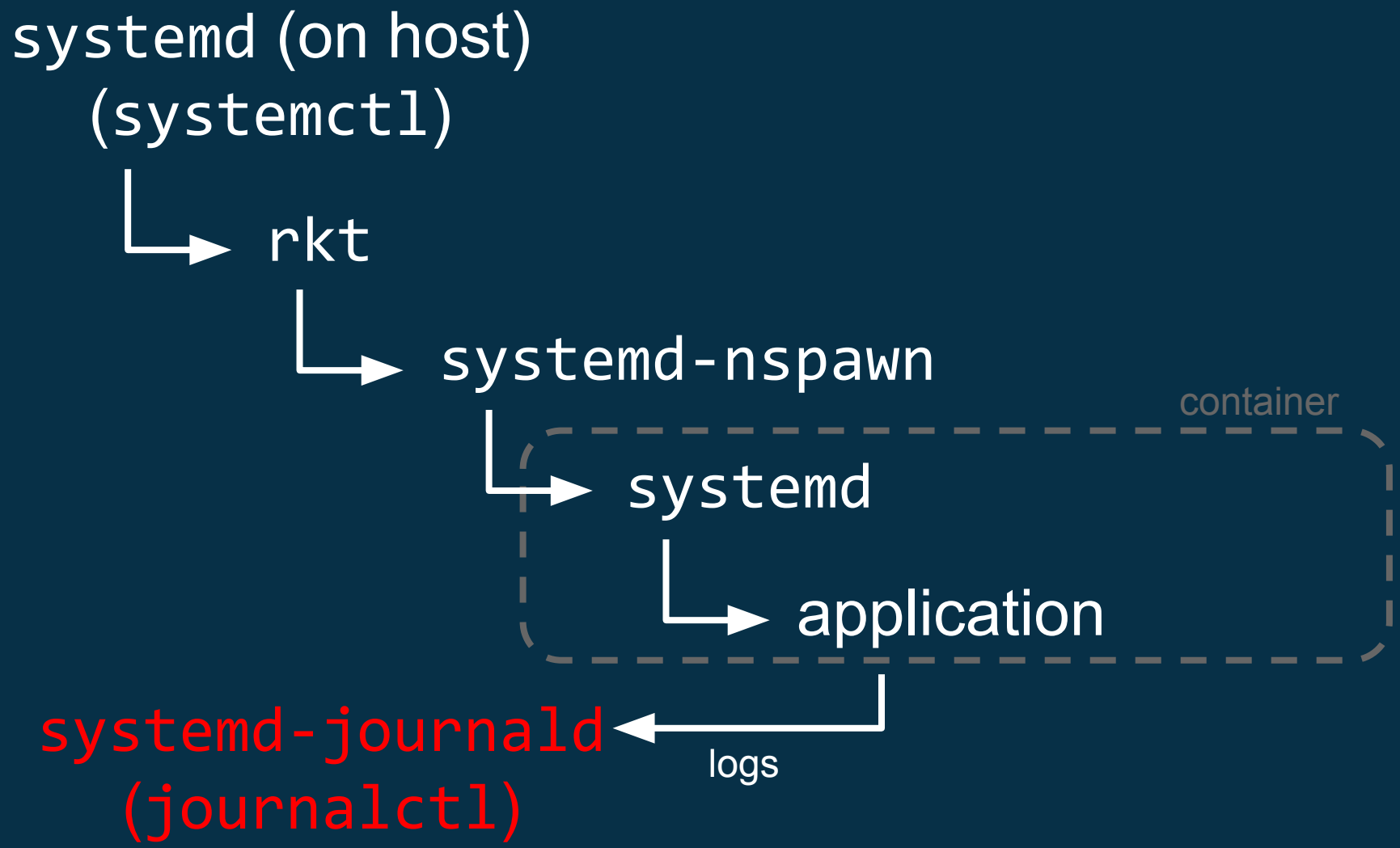
systemd

pid1
service files
socket activation


systemd (on host)
(systemctl)
rkt
systemd-nspawn
container
systemd
application

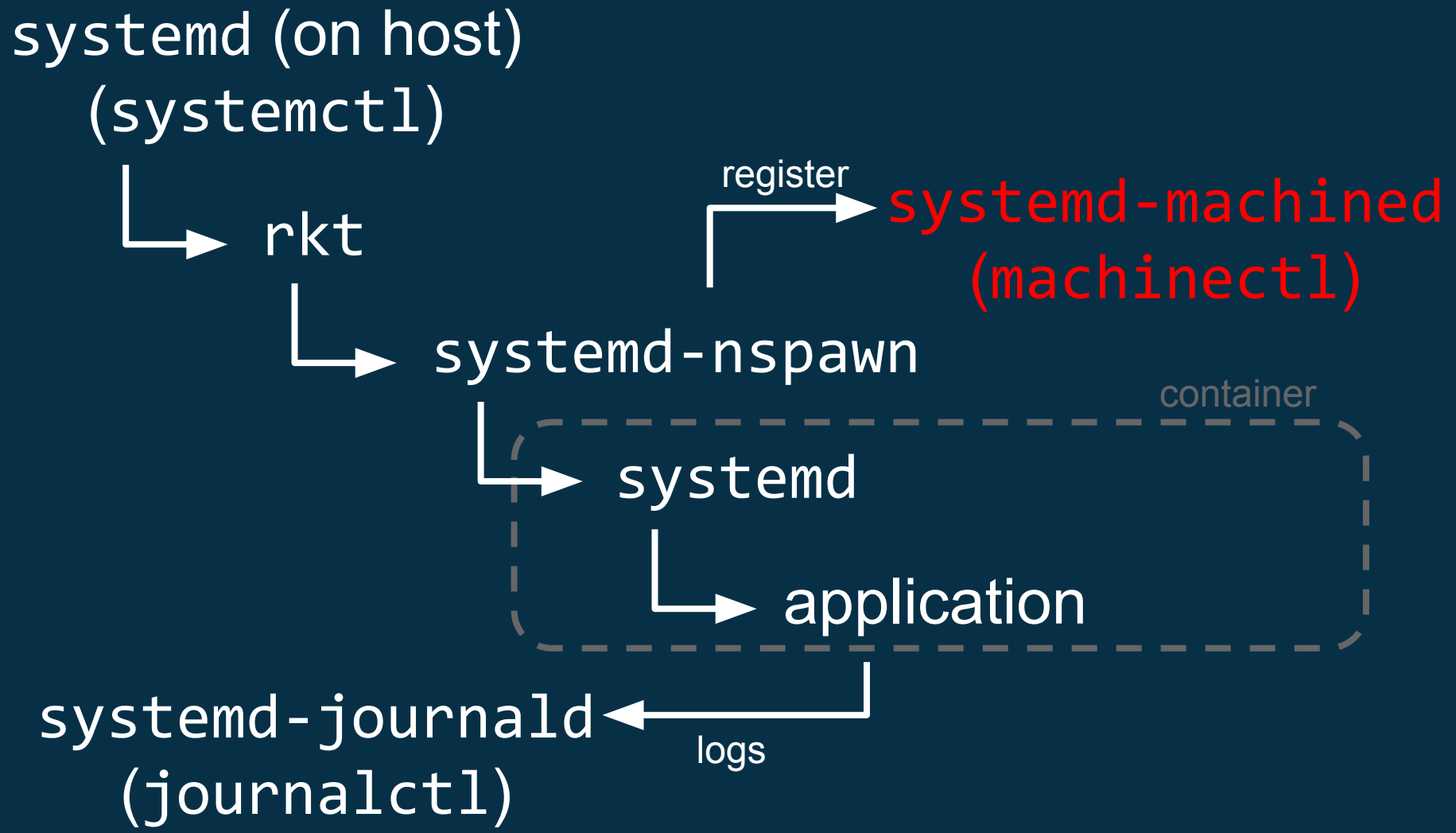
application

app-level isolators: CPUShares, MemoryLimit
chrooted


systemd (on host)
(systemctl)
rkt
systemd-nspawn
container
systemd
application
systemd-journald
(journalctl)
logs

systemd-journald

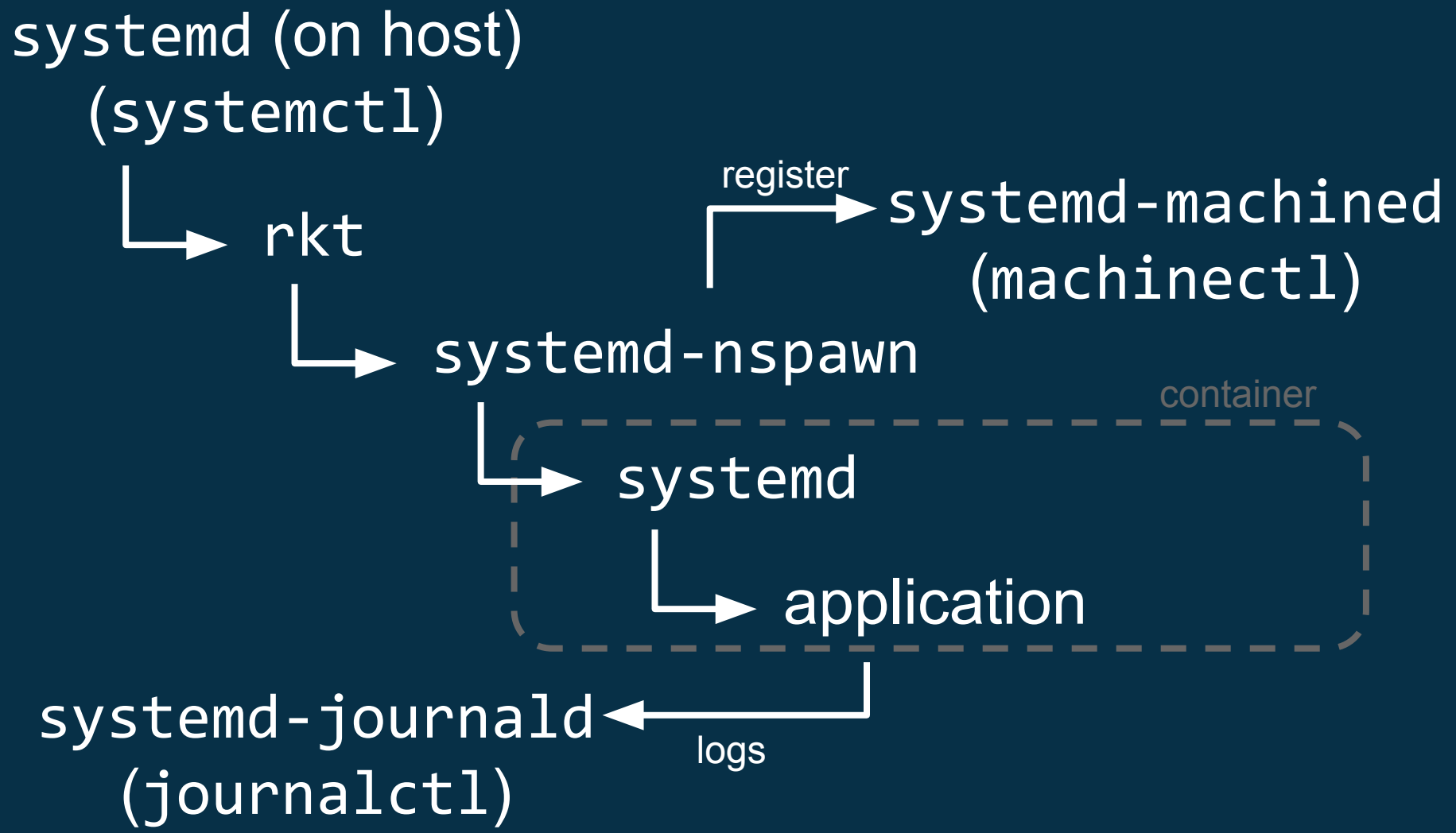
no changes in apps required
logs in the container
available from the host with journalctl -m / -M


systemd (on host)
(systemctl)
rkt
register
systemd-machined
(machinectl)
systemd-nspawn
container
systemd
application
systemd-journald
(journalctl)
logs

systemd-machined

register on distros using systemd

`machinectl {show,status,poweroff...}`

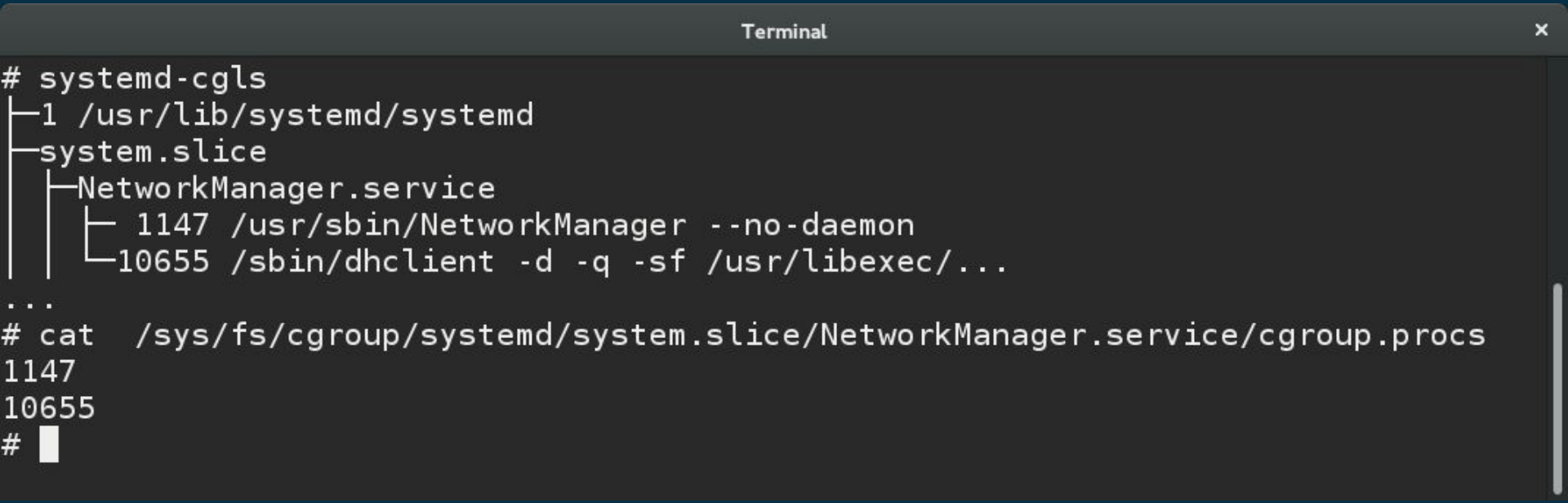

systemd (on host)
(systemctl)
rkt
register
systemd-machined
(machinectl)
systemd-nspawn
container
systemd
application
systemd-journald
(journalctl)
logs

cgroups

What’s a control group? (cgroup)

- group processes together
- organised in trees
- applying limits to them as a group

cgroups

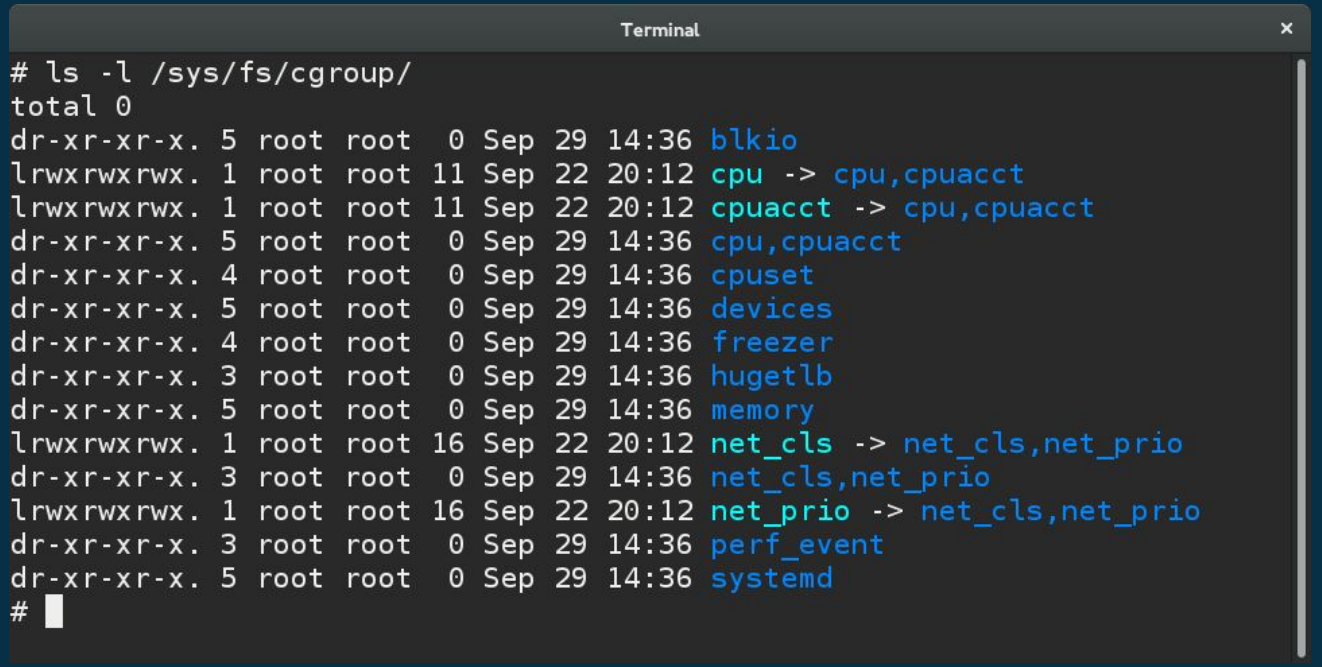

Terminal
systemd-cgls
├─1 /usr/lib/systemd/systemd
├─system.slice
│ ├─NetworkManager.service
│ │ ├─ 1147 /usr/sbin/NetworkManager --no-daemon
│ │ └─10655 /sbin/dhclient -d -q -sf /usr/libexec/...
...
cat /sys/fs/cgroup/systemd/system.slice/NetworkManager.service/cgroup.procs
1147
10655
#

cgroup API

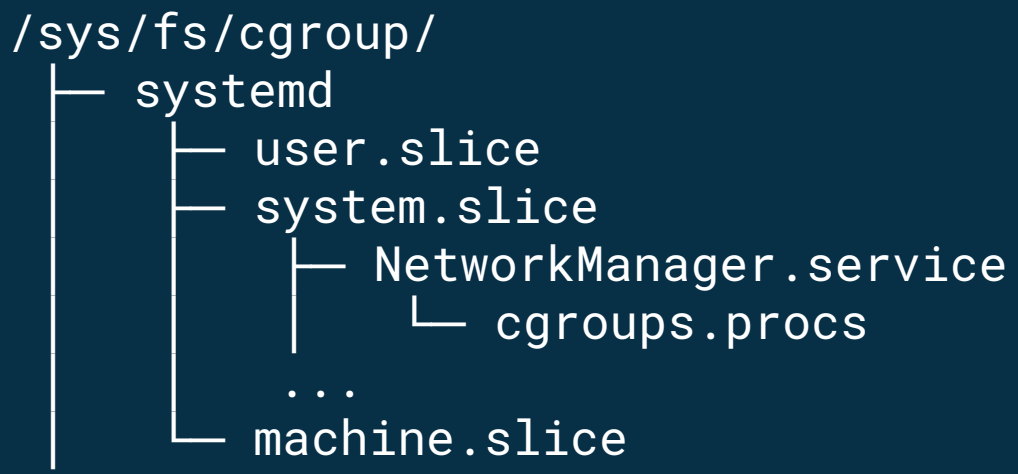
```
/sys/fs/cgroup/*/
/proc/cgroups
/proc/$PID/cgroup
```

List of cgroup controllers

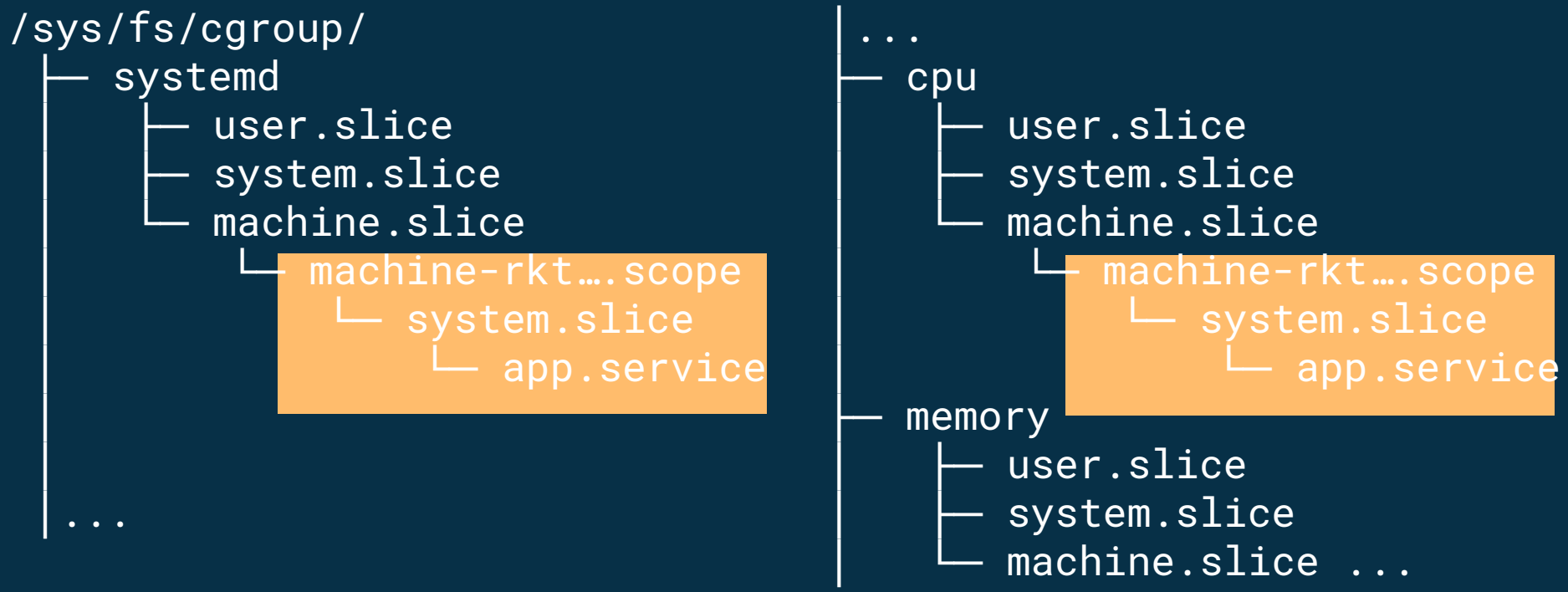
```
/sys/fs/cgroup/
├── cpu
├── devices
├── freezer
├── memory
├── ...
└── systemd
```



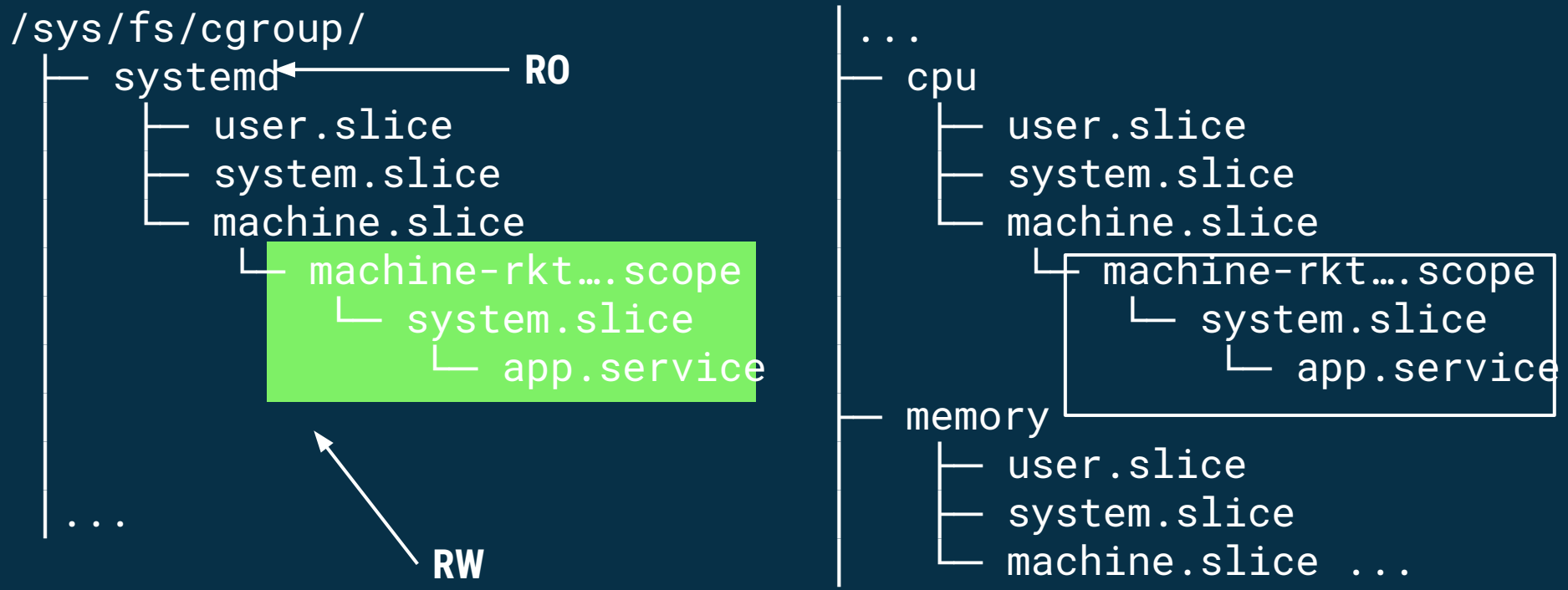
How systemd units use cgroups



How systemd units use cgroups w/ containers



cgroups mounted in the container



Example: memory isolator

Example: CPU isolator

Application Image Manifest

systemd service file

systemd action

Unified cgroup hierarchy

- Multiple hierarchies:
  - one cgroup mount point for each controller (memory, cpu, etc.)
  - flexible but complex
  - cannot remount with a different set of controllers
  - difficult to give to containers in a safe way
- Unified hierarchy:
  - cgroup filesystem mounted only one time
  - still in development in Linux: mount with option "`__DEVEL__sane_behavior`"
  - initial implementation in systemd-v226 (September 2015)
  - no support in rkt yet

How rkt helps systemd

Regression bug fixes
nspawn bug fixes (~journald and cgroups)
PID1 fixes (e.g. RootDirectory)
nspawn to exit with a return code

rkt: a few other things

- rkt and security
- rkt API service (new!)
- rkt networking
- rkt and user namespaces
- rkt and production

rkt and security

"secure by default"

rkt security

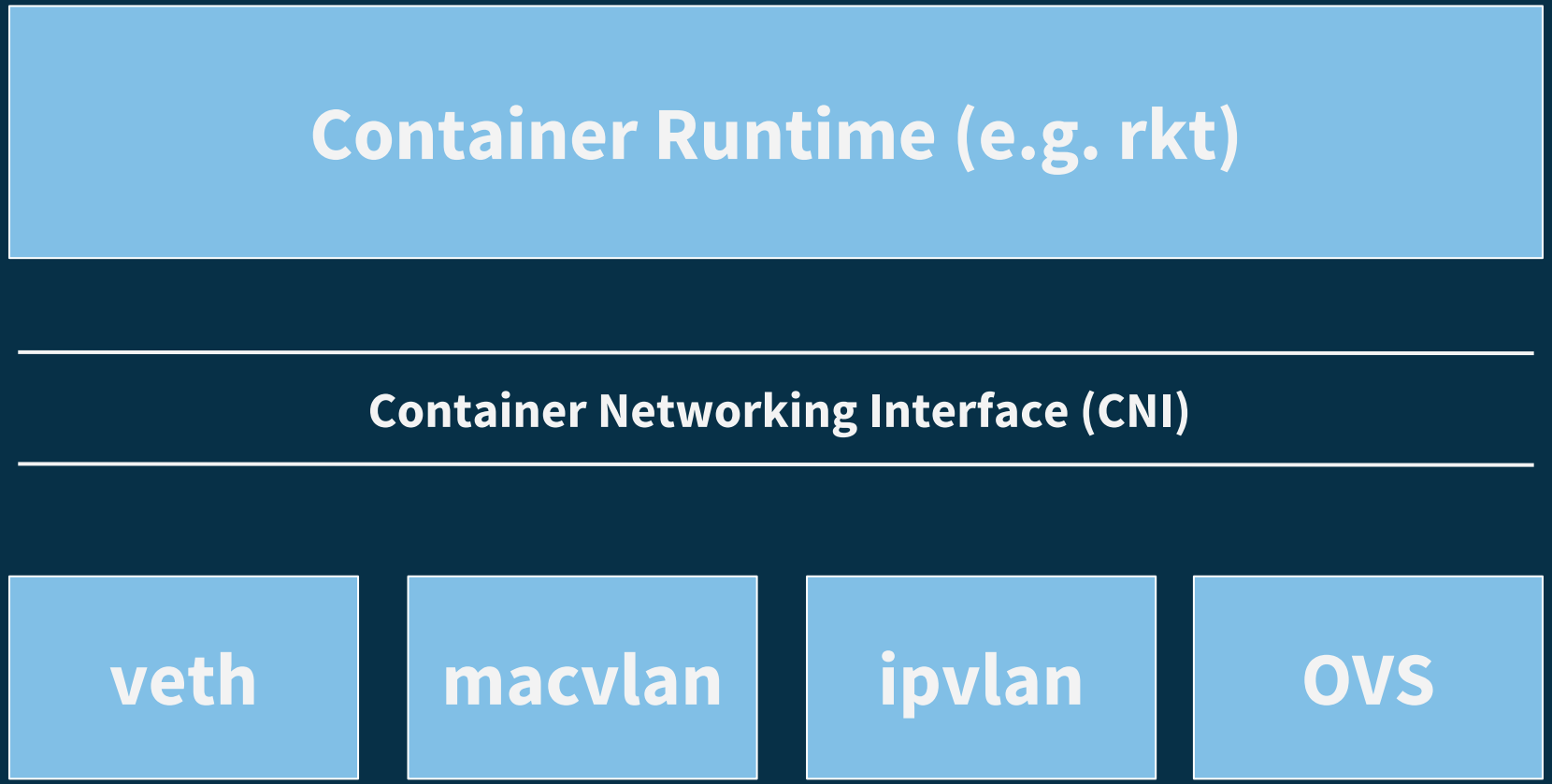
- image signature verification
- privilege separation
    - e.g. fetch images as non-root user
- SELinux integration
- kernel keyring integration (soon)
- lkvm stage1 for true hardware isolation

rkt API service (new!)

optional, gRPC-based API daemon
exposes information on pods and images
runs as unprivileged user
easier integration with other projects

rkt networking

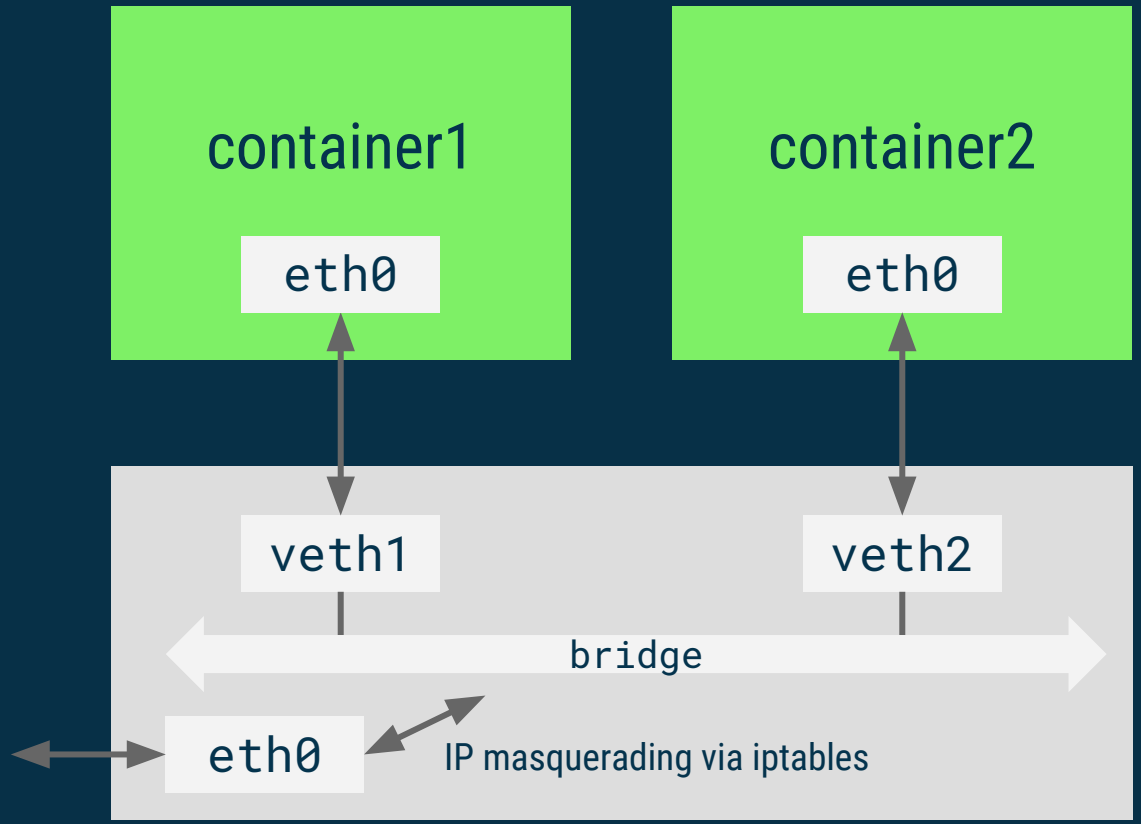
plugin-based
Container Networking Interface (CNI)


Container Runtime (e.g. rkt)
Container Networking Interface (CNI)
veth
macvlan
ipvlan
OVS

Networking, the rkt way

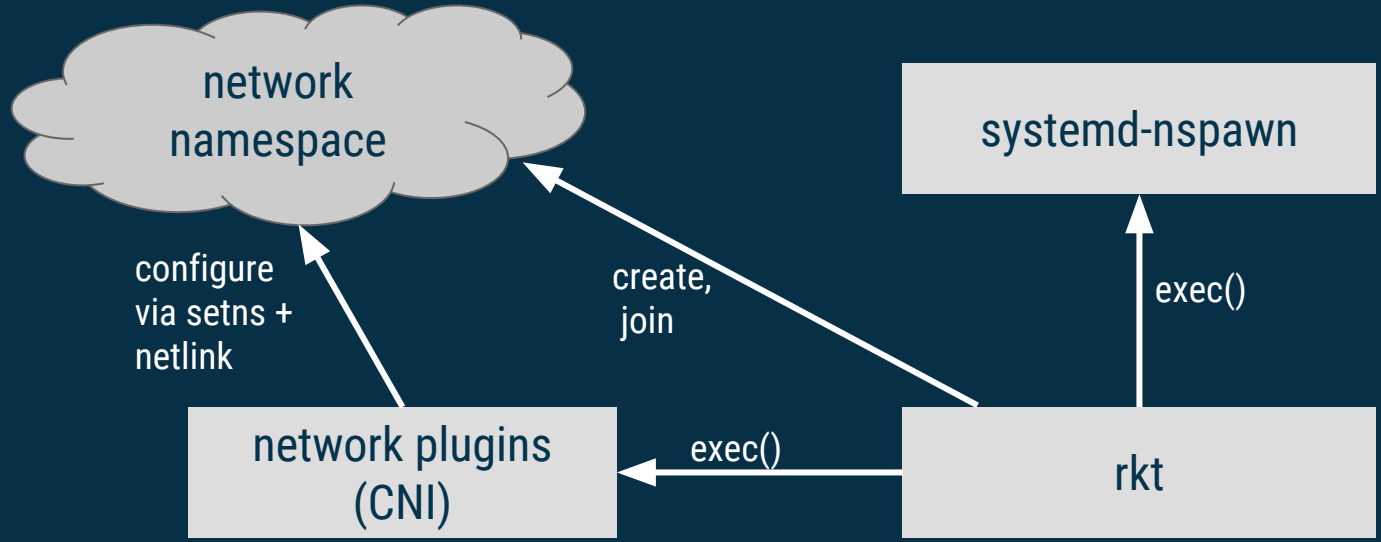
Network tooling

- Linux can create pairs of virtual net interfaces
- Can be linked in a bridge



How does rkt do it?

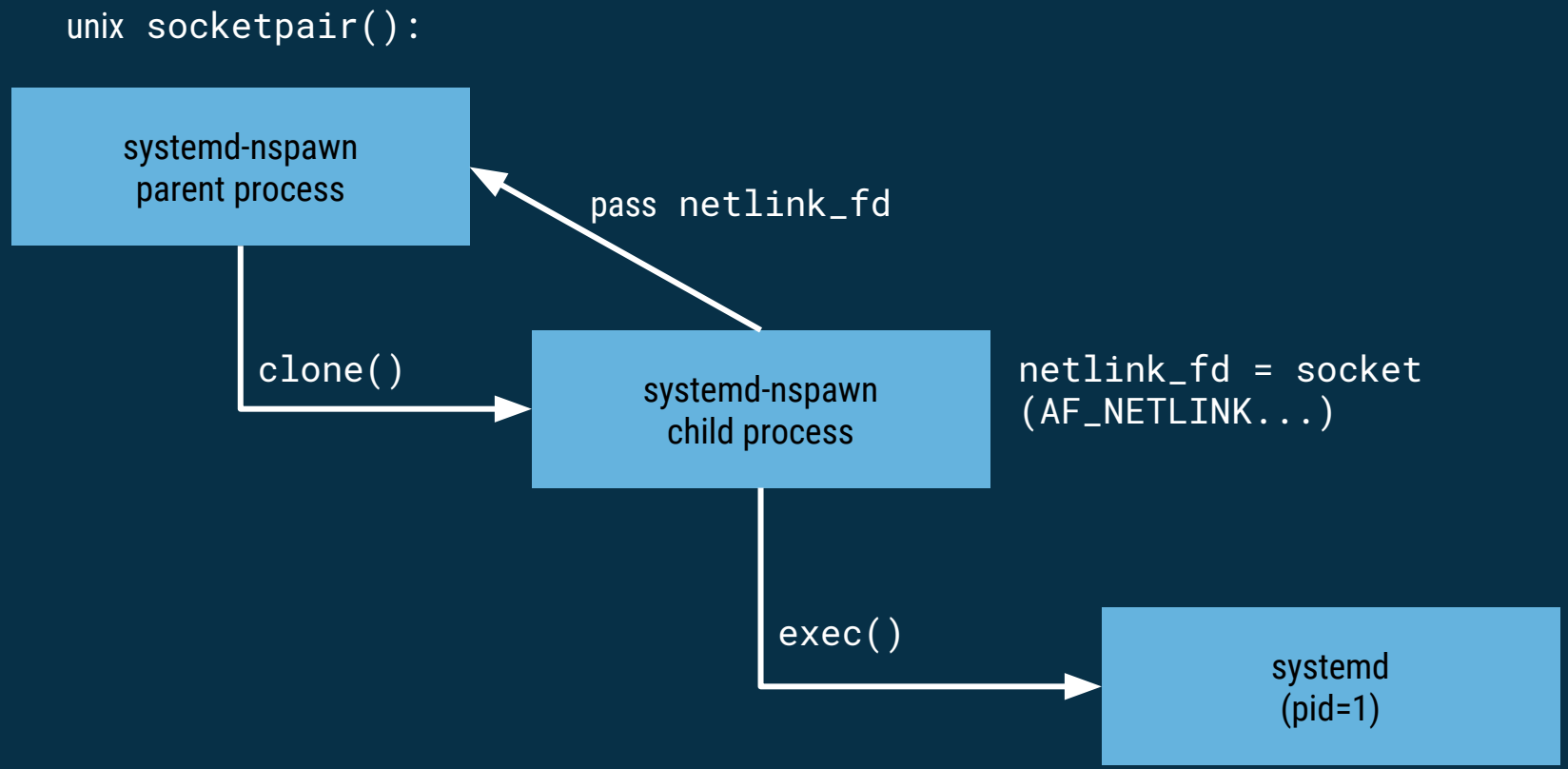
- rkt uses the Container Network Interface (CNI)



`/var/lib/rkt/pods/run/$POD_UUID/netns`

Network, the nspawn way

--private-network, --network-interface=
--network-macvlan=, --network-ipvlan=
--network-veth=, --network-bridge=


unix socketpair():
systemd-nspawn
parent process
pass netlink_fd
clone()
systemd-nspawn
child process
netlink_fd = socket
(AF_NETLINK...)
exec()
systemd
(pid=1)

rkt and user namespaces

History of Linux namespaces

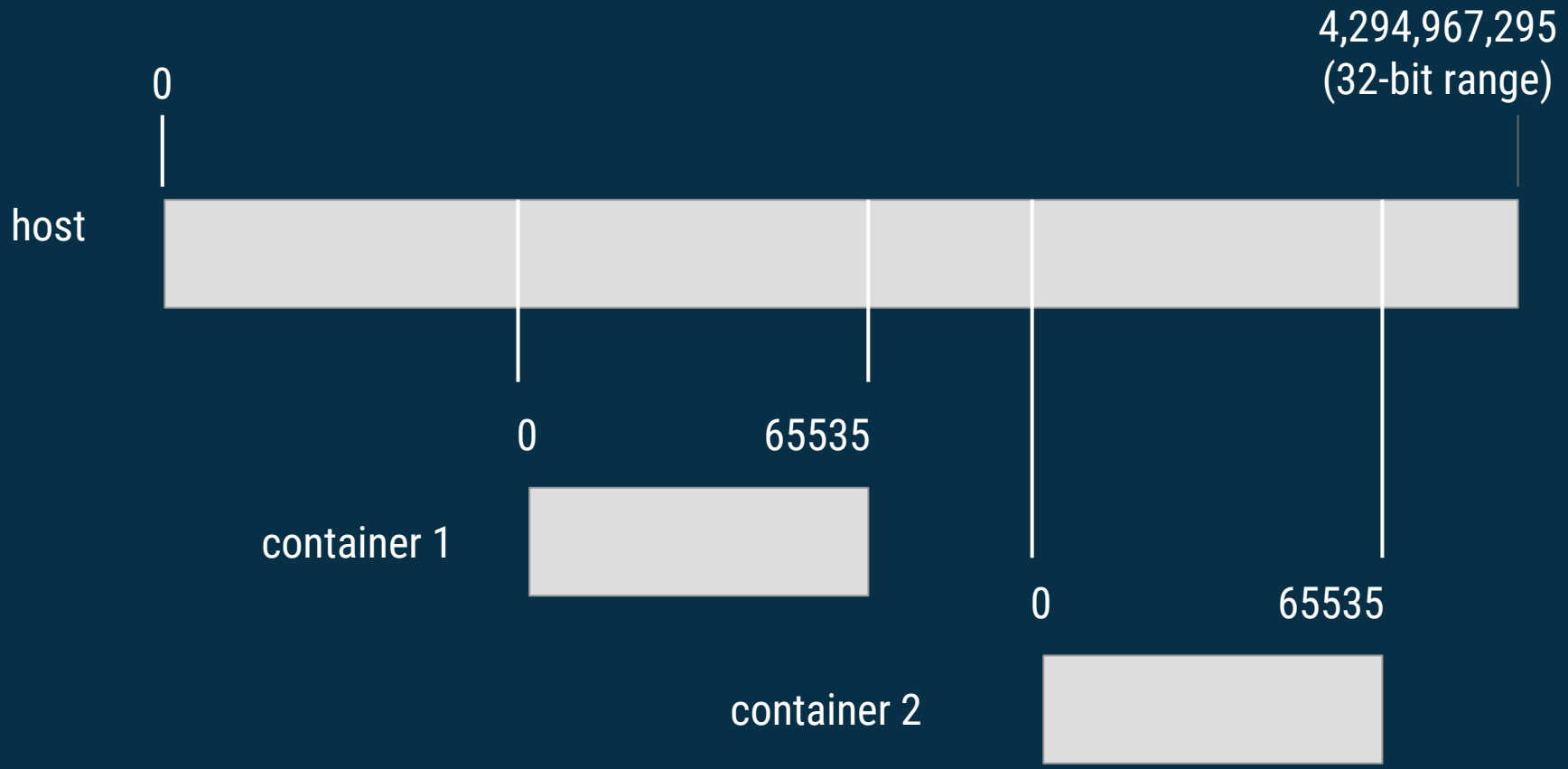
- ✓ 1991: Linux
- ✓ 2002: namespaces in Linux 2.4.19
- ✓ 2008: LXC
- ✓ 2011: systemd-nspawn
- ✓ 2013: **user namespaces** in Linux 3.8
- ✓ 2013: Docker
- ✓ 2014: rkt

… development still active

Why user namespaces?

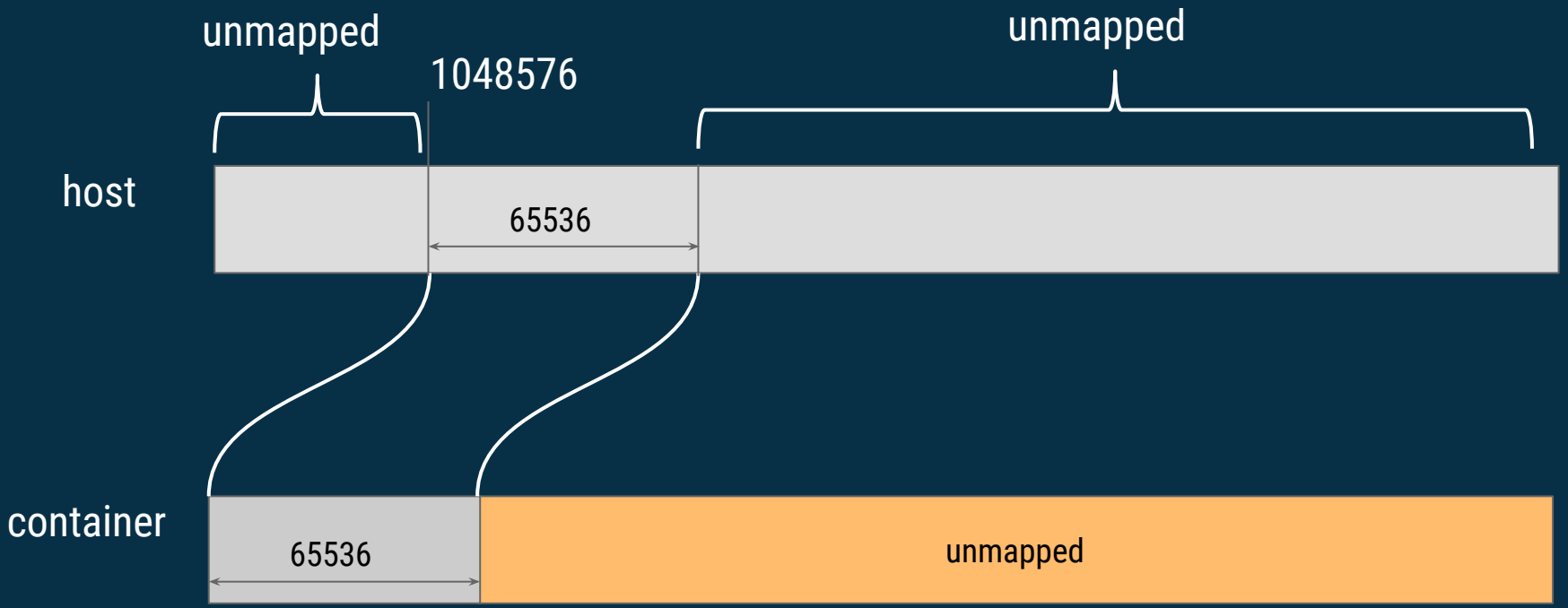
- Better isolation
- Run applications which would need more capabilities
- Per user limits
- Future?
  - Unprivileged containers: possibility to have container without root

User ID ranges

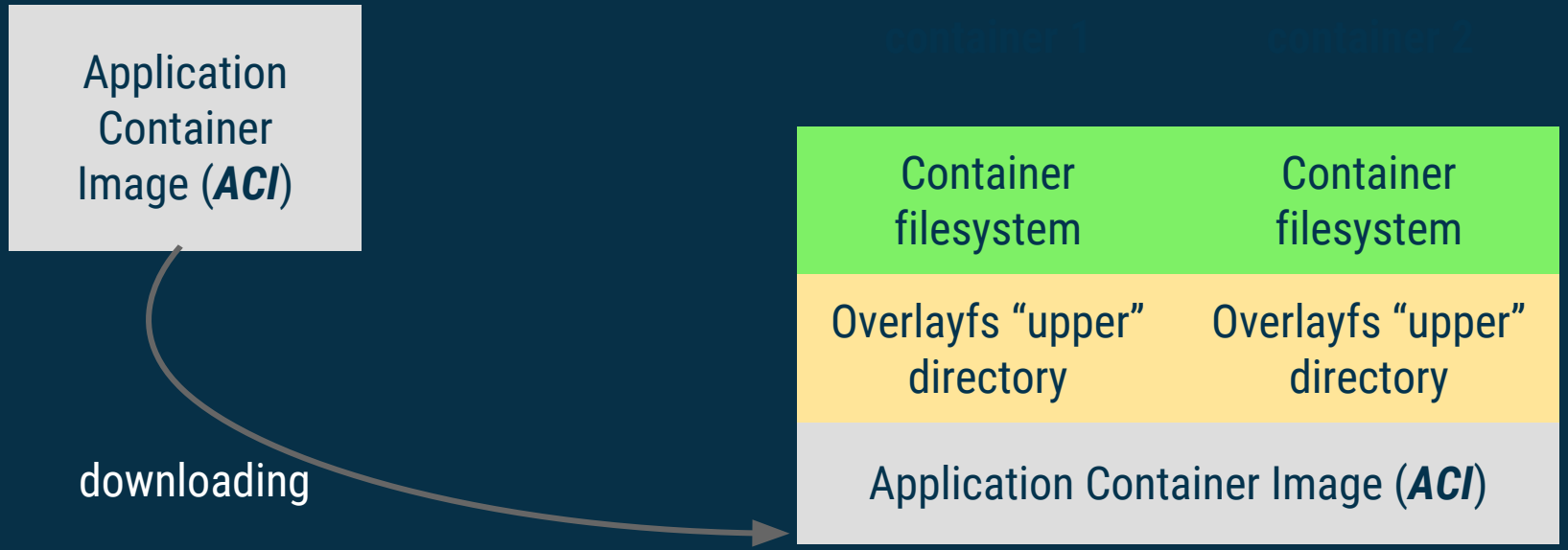


User ID mapping

`/proc/$PID/uid_map: “0  1048576  65536”`



Problems with container images

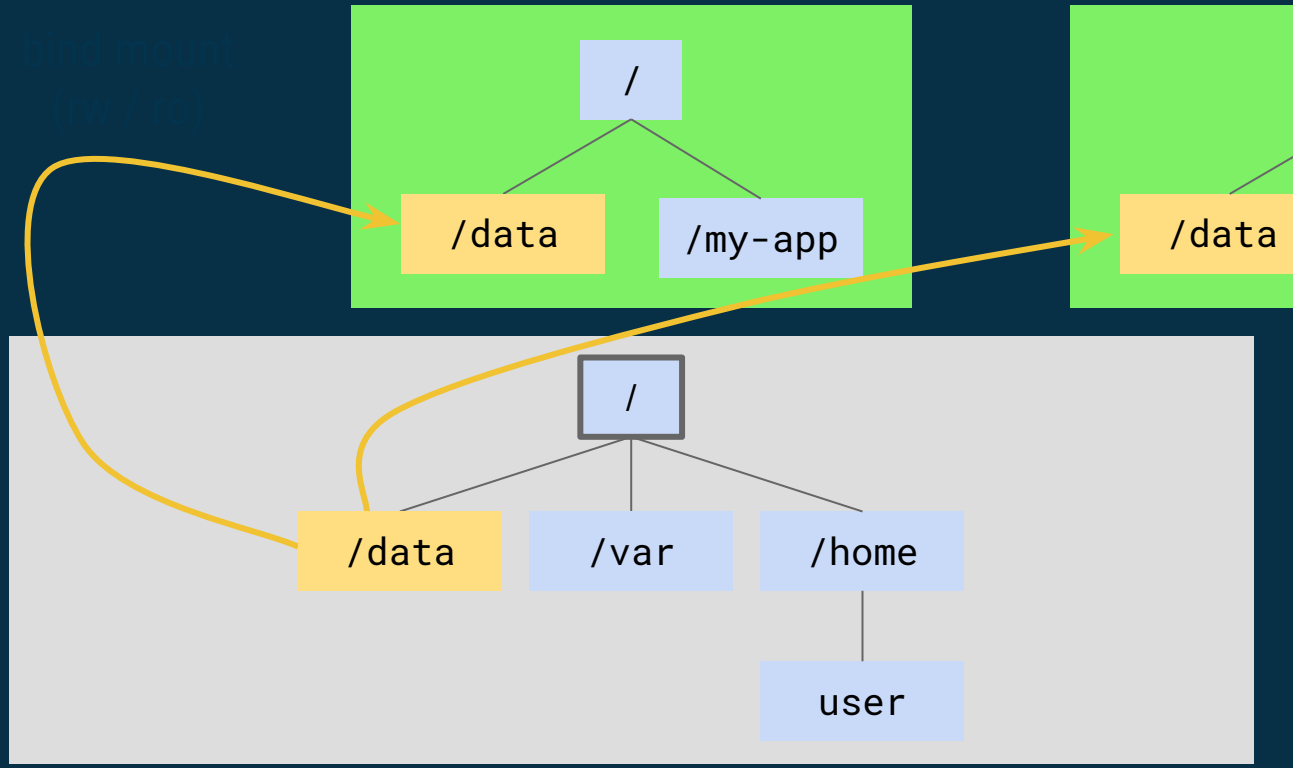


Problems with container images

- Files UID / GID
- rkt currently only supports user namespaces without overlayfs
  - Performance loss: no COW from overlayfs
  - “chown -R” for every file in each container

Problems with volumes

- mounted in several containers
- No UID translation



User namespace and filesystem problem

- Possible solution: add options to mount() to apply a UID mapping
- rkt would use it when mounting:
  - the overlay rootfs
  - volumes
- Idea suggested on kernel mailing lists
- Hackfest tomorrow!

rkt and production

- still pre-1.0
- unstable (but stabilising) CLI and API
- explicitly *not* recommended for production
    - although some early adopters

rkt v1.0.0

EOY (fingers crossed)
stable API
stable CLI
ready to use!

Questions?

Join us!

github.com/coreos/rkt

coreos.com/careers (soon in Berlin!)