Container mechanics

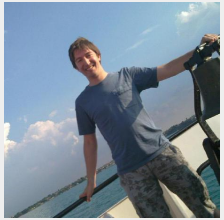
in rkt and Linux

Alban Crequy

LinuxCon Europe 2015 Dublin

Alban Crequy

- Working on rkt
- One of the maintainer of rkt
- Previously worked on D-Bus and AF_BUS

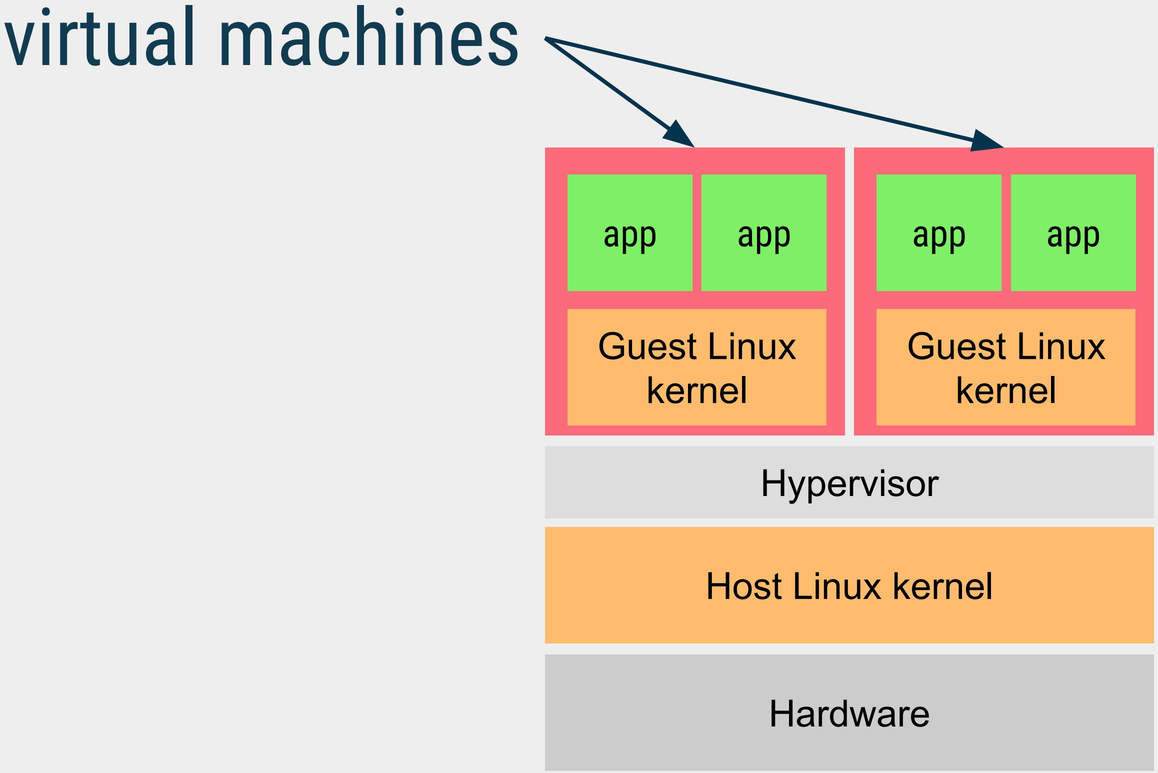


https://github.com/alban

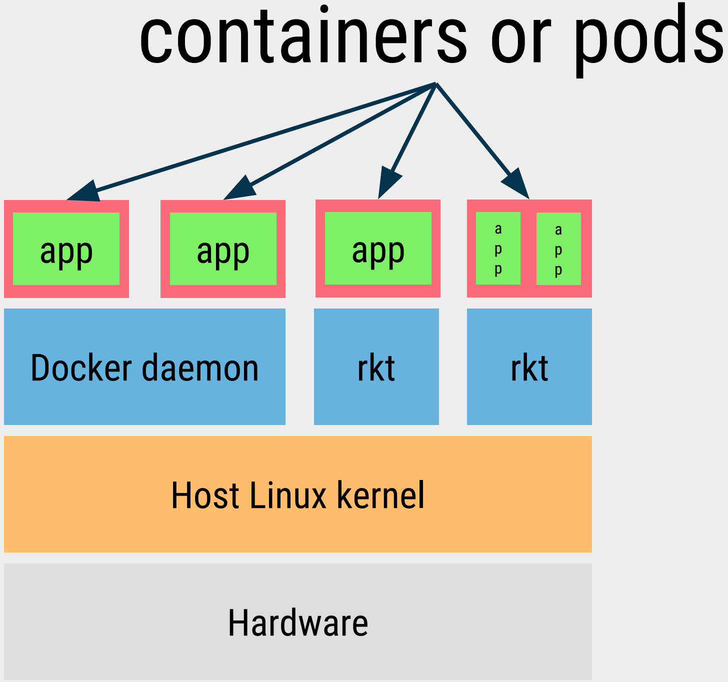
Container mechanics in rkt and Linux

- Containers
- Linux namespaces
- Cgroups
- How rkt use them

Containers vs virtual machines



Containers


containers or pods
app
app
app
a
p
p
a
p
p
Docker daemon
rkt
rkt
Host Linux kernel
Hardware

rkt architecture

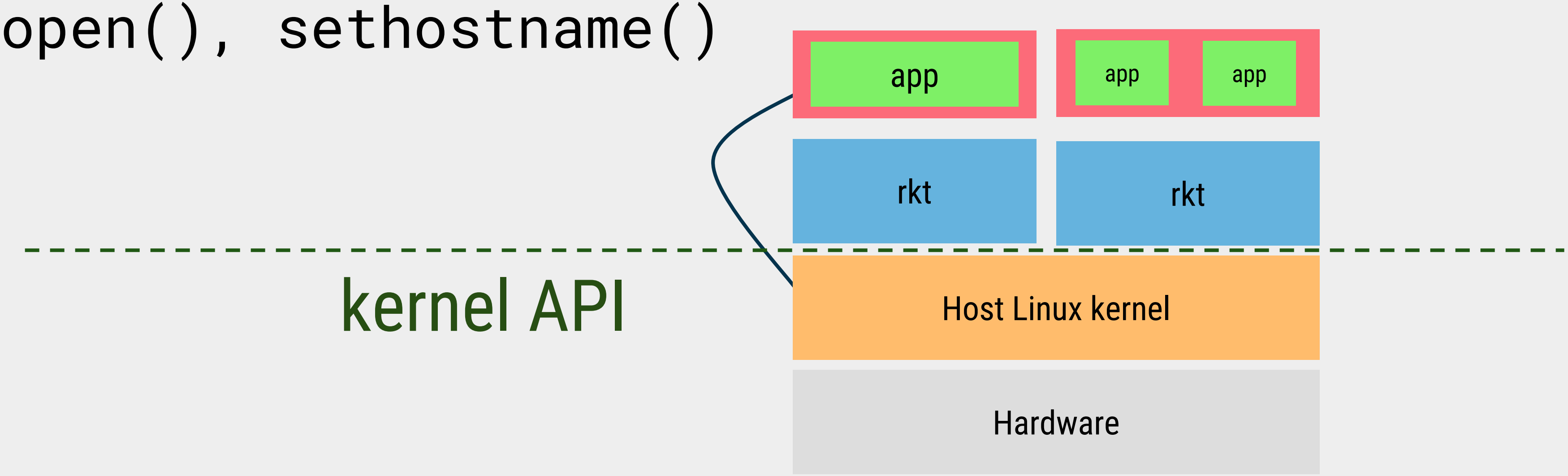
stage 2	apps	apps
stage 1	systemd-nspawn	lkvm
stage 0	rkt	rkt

Containers: no guest kernel

system calls:

`open(), sethostname()`

kernel API



Containers with an example

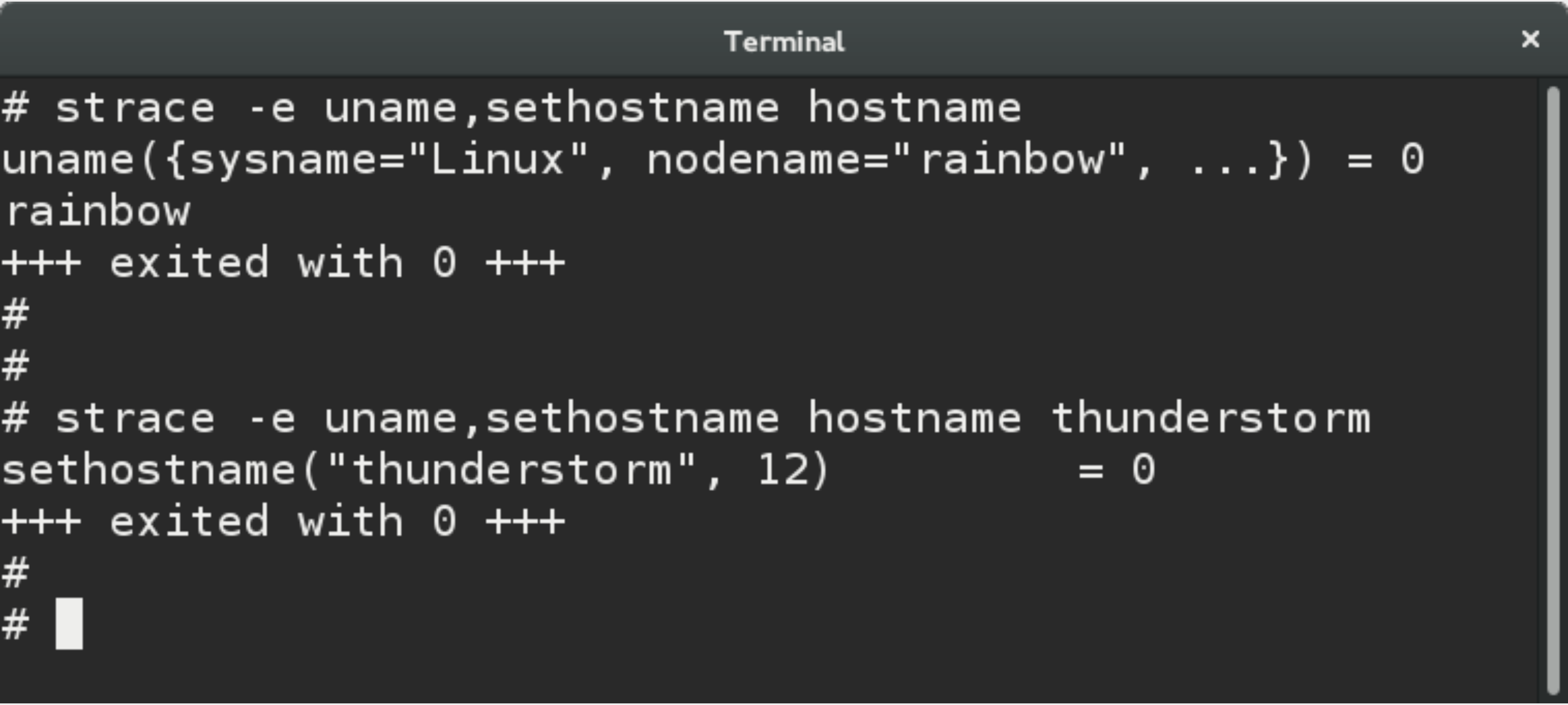
Getting and setting the hostname:

- **The system calls for getting and setting the hostname are older than containers**

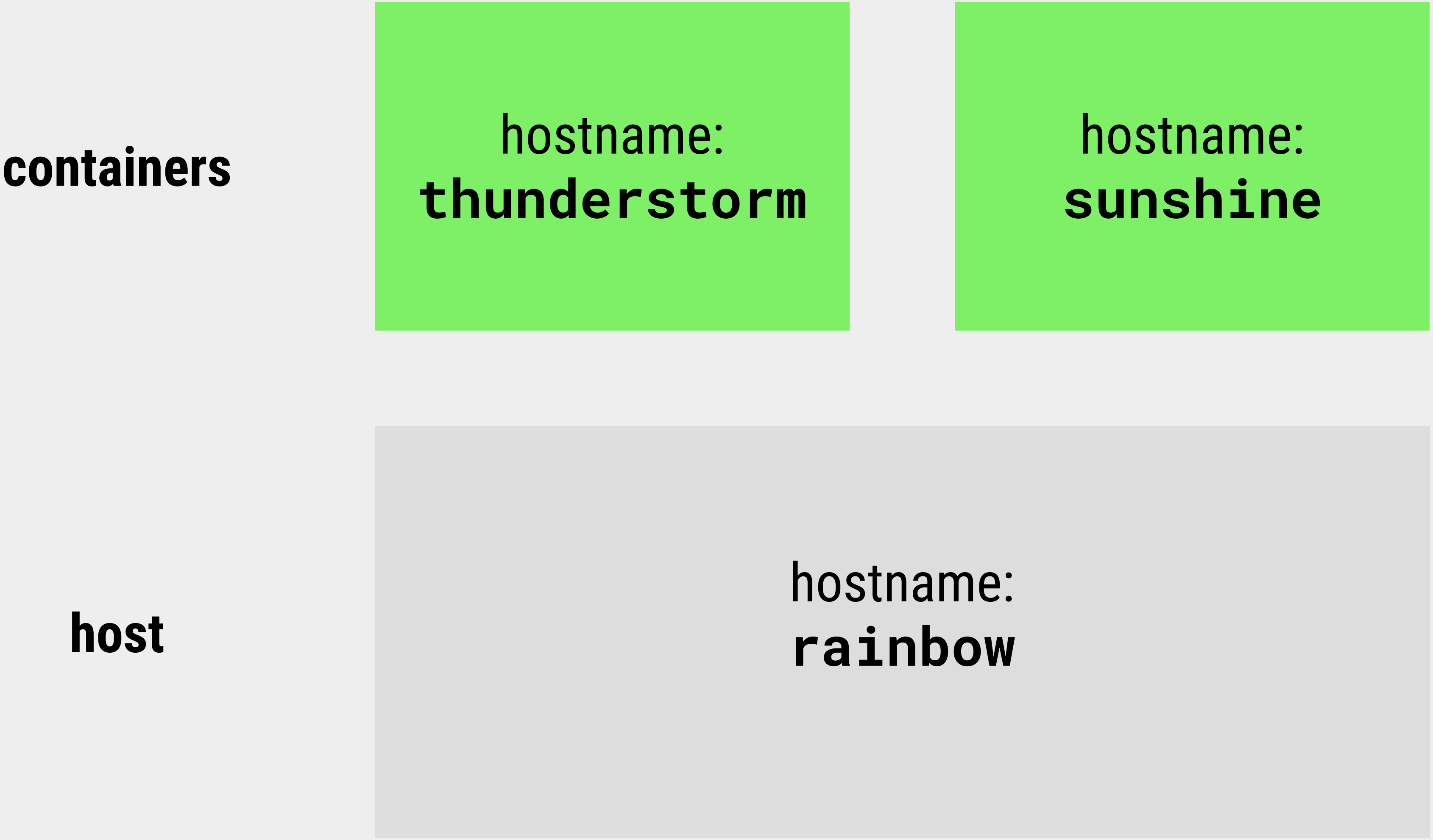
```
int uname(struct utsname *buf);

int gethostname(char *name, size_t len);

int sethostname(const char *name, size_t len);
```



```
# strace -e uname,sethostname hostname
uname({sysname="Linux", nodename="rainbow", ...}) = 0
rainbow
+++ exited with 0 +++
#
#
# strace -e uname,sethostname hostname thunderstorm
sethostname("thunderstorm", 12)         = 0
+++ exited with 0 +++
#
# 
```



containers

hostname: **thunderstorm**

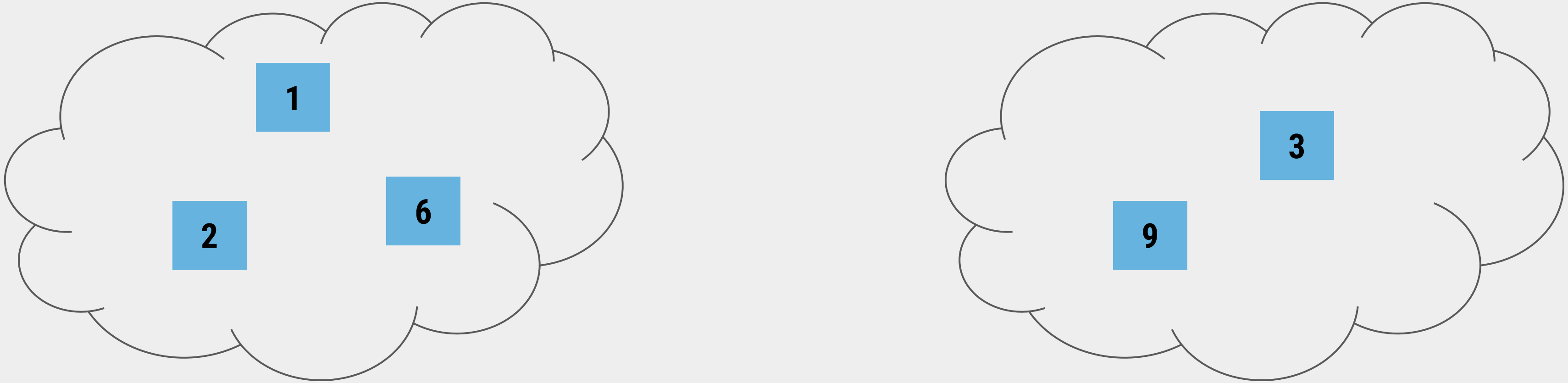
hostname: **sunshine**

host

hostname: **rainbow**

Linux namespaces

Processes in namespaces



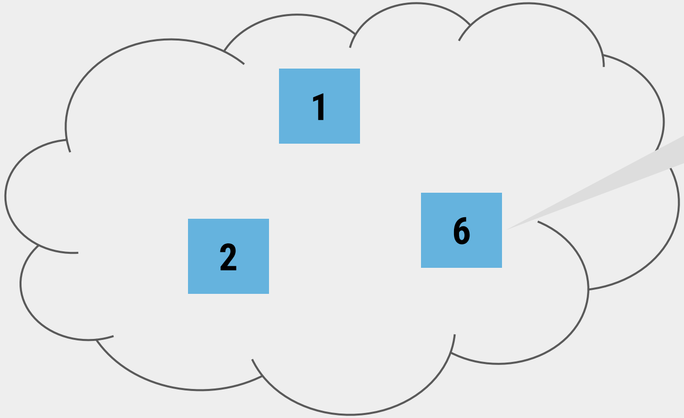
```
gethostname() -> “rainbow”        gethostname() -> “thunderstorm”
```

Linux Namespaces

Several independent namespaces

- **uts** (Unix Timesharing System) **namespace**
- **mount namespace**
- **pid namespace**
- **network namespace**
- **user namespace**

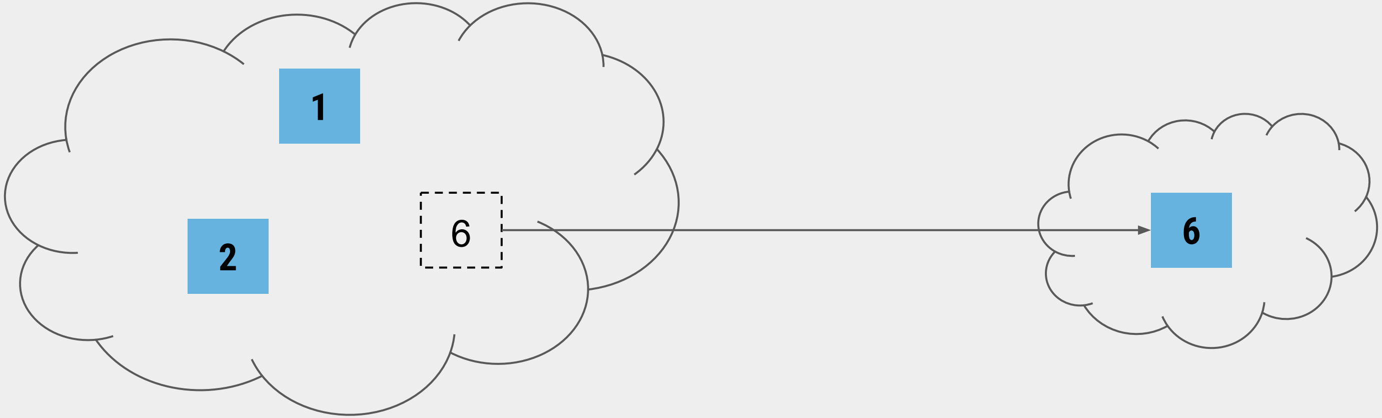
Creating new namespaces



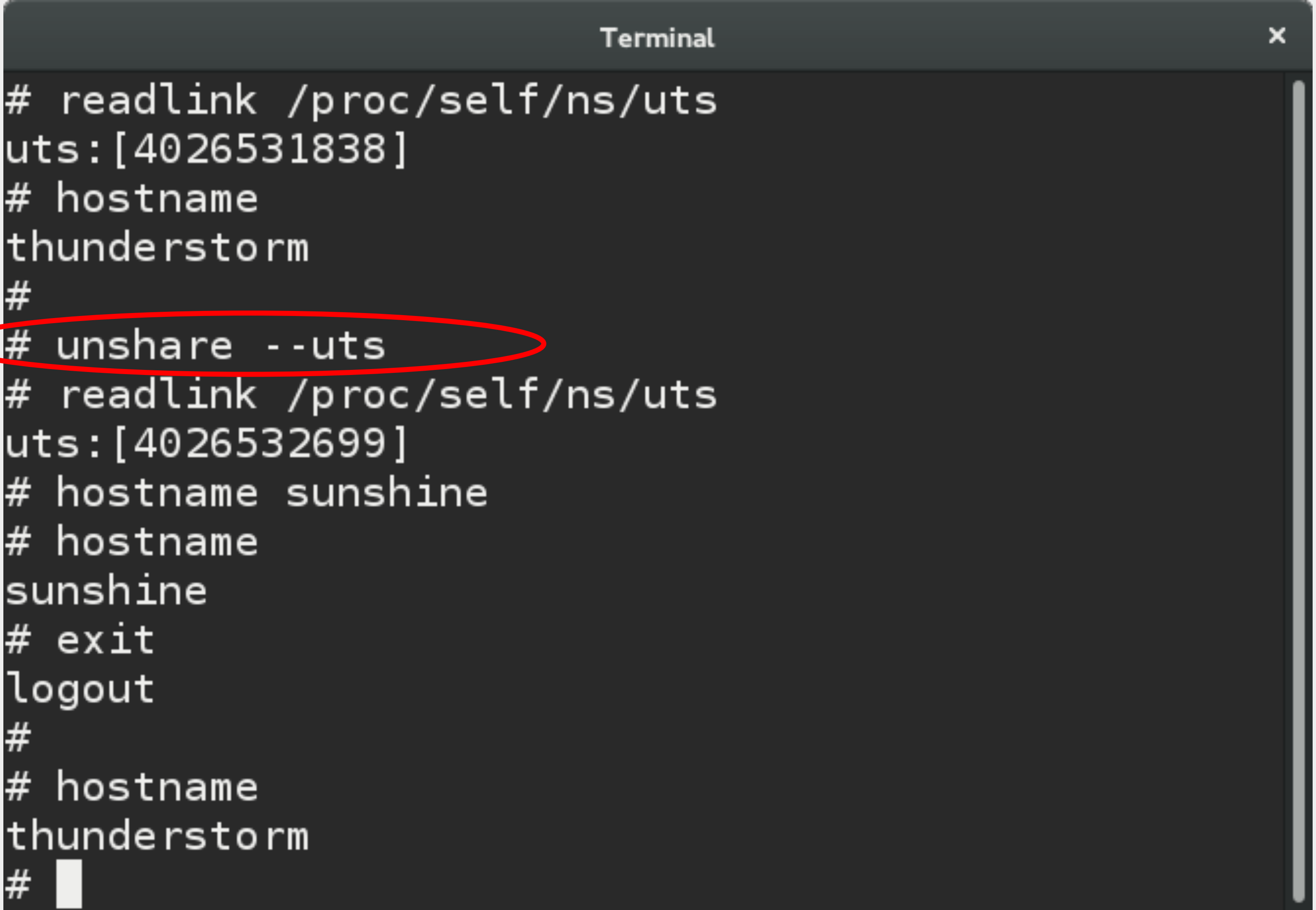
```
unshare(CLONE_NEWUTS);
```

“rainbow”

Creating new namespaces



“rainbow” “rainbow”



```
# readlink /proc/self/ns/uts
uts:[4026531838]
# hostname
thunderstorm
#
# unshare --uts
# readlink /proc/self/ns/uts
uts:[4026532699]
# hostname sunshine
# hostname
sunshine
# exit
logout
#
# hostname
thunderstorm
# 
```

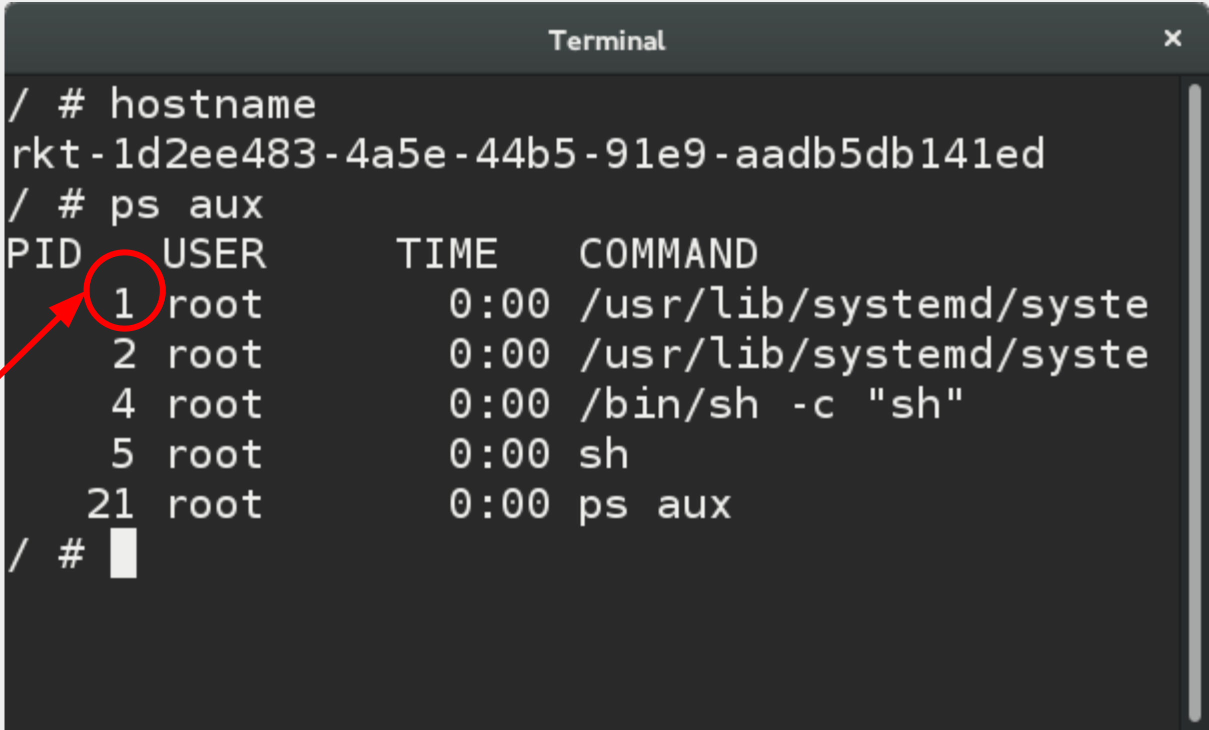
PID namespace

Hiding processes and PID translation

- the host sees all processes
- the container only its own processes

Hiding processes and PID translation

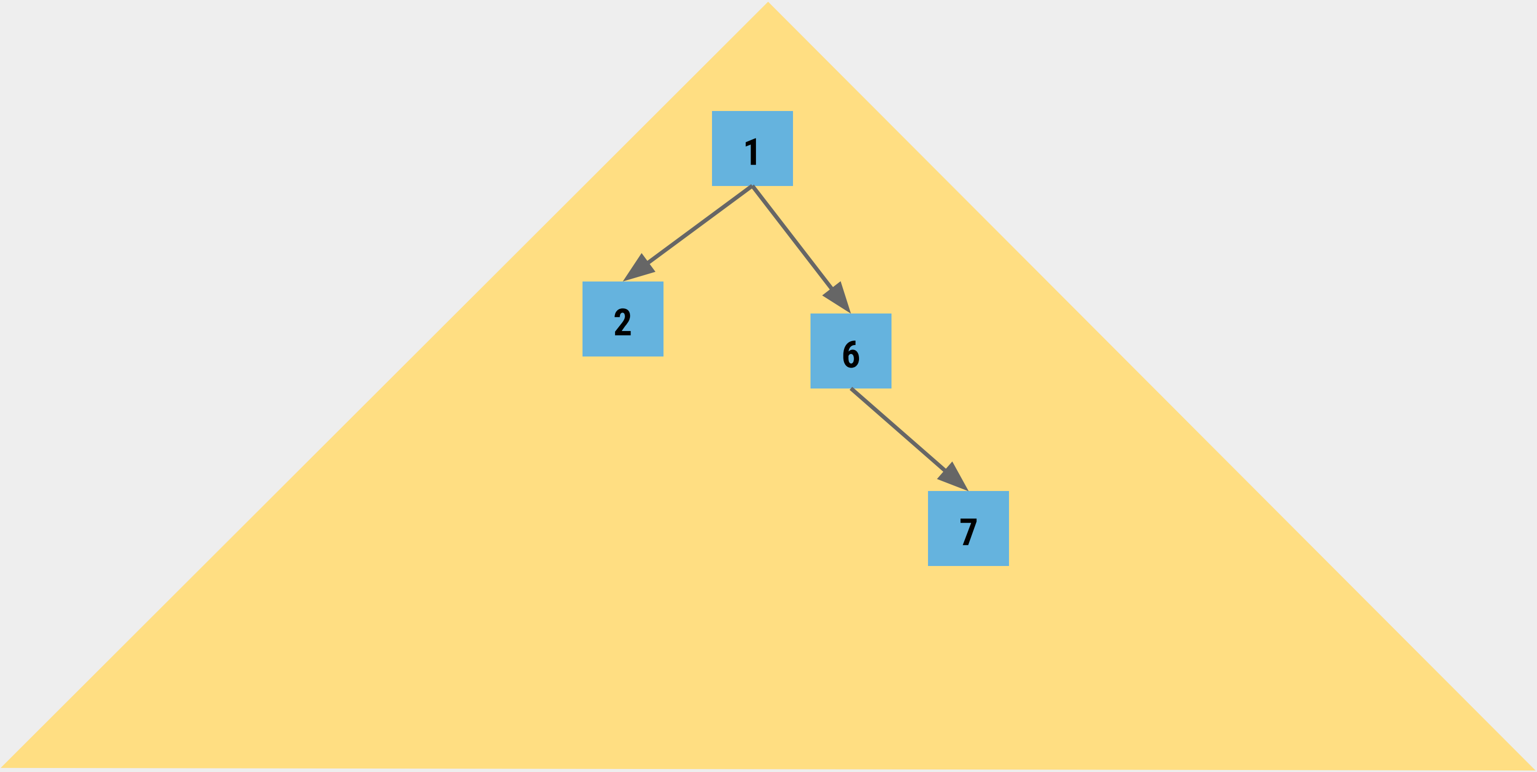
- the host sees all processes
- the container only its own processes



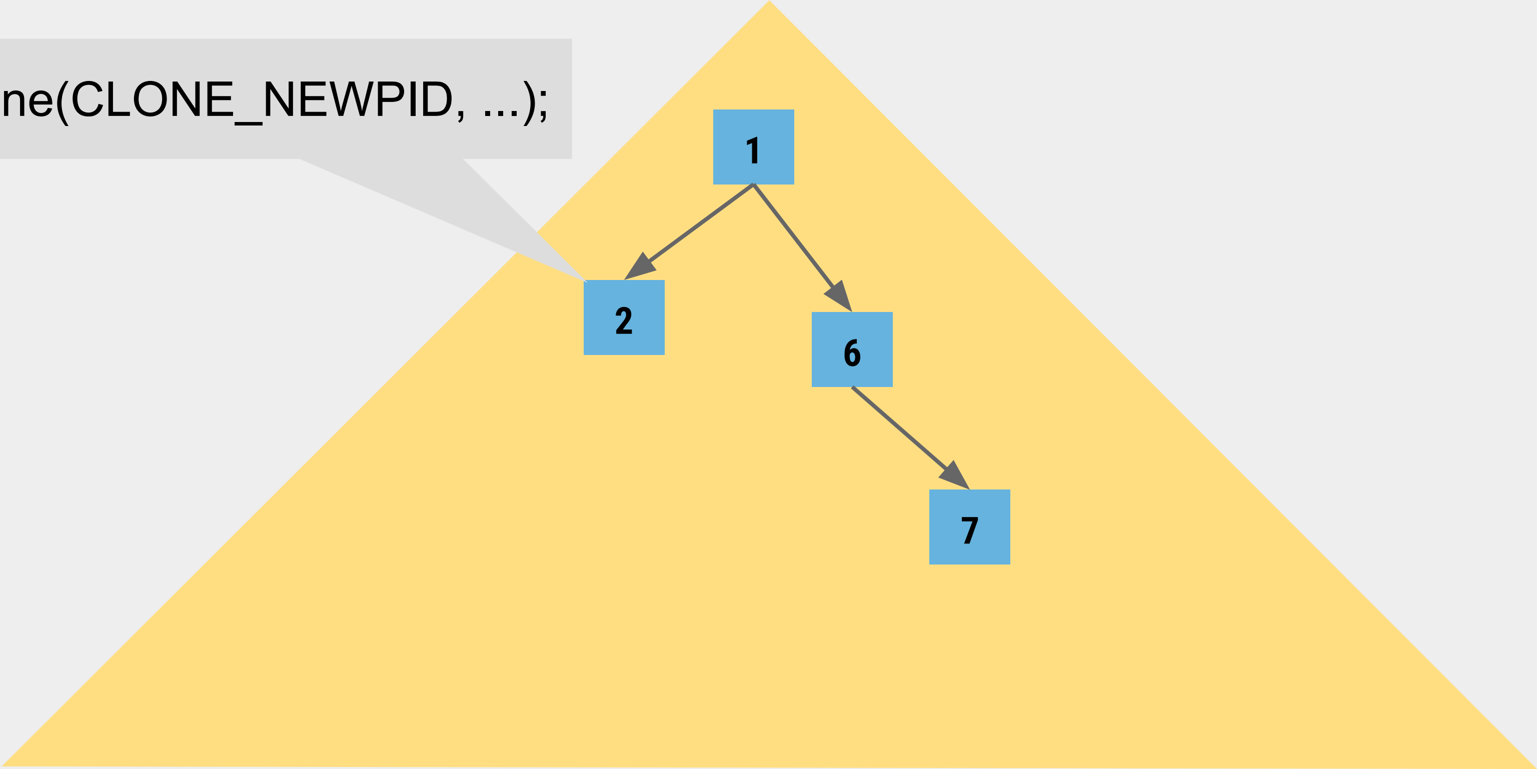
```
/ # hostname
rkt-1d2ee483-4a5e-44b5-91e9-aadb5db141ed
/ # ps aux
PID   USER     TIME   COMMAND
    1 root       0:00 /usr/lib/systemd/syste
    2 root       0:00 /usr/lib/systemd/syste
    4 root       0:00 /bin/sh -c "sh"
    5 root       0:00 sh
   21 root       0:00 ps aux
/ # 
```

Actually pid 30920

Initial PID namespace

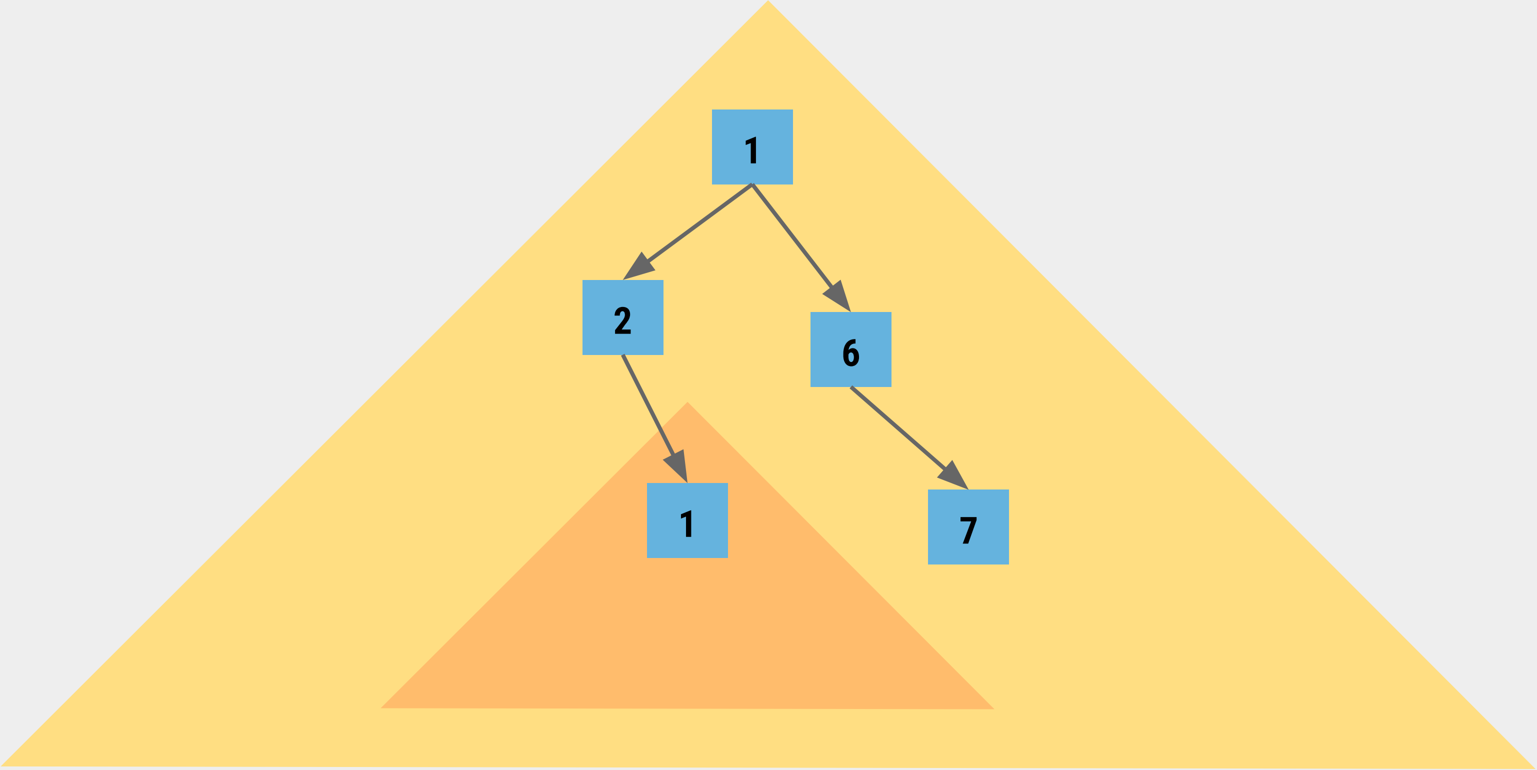


Creating a new namespace



```
clone(CLONE_NEWPID, ...);
```

Creating a new namespace



rkt

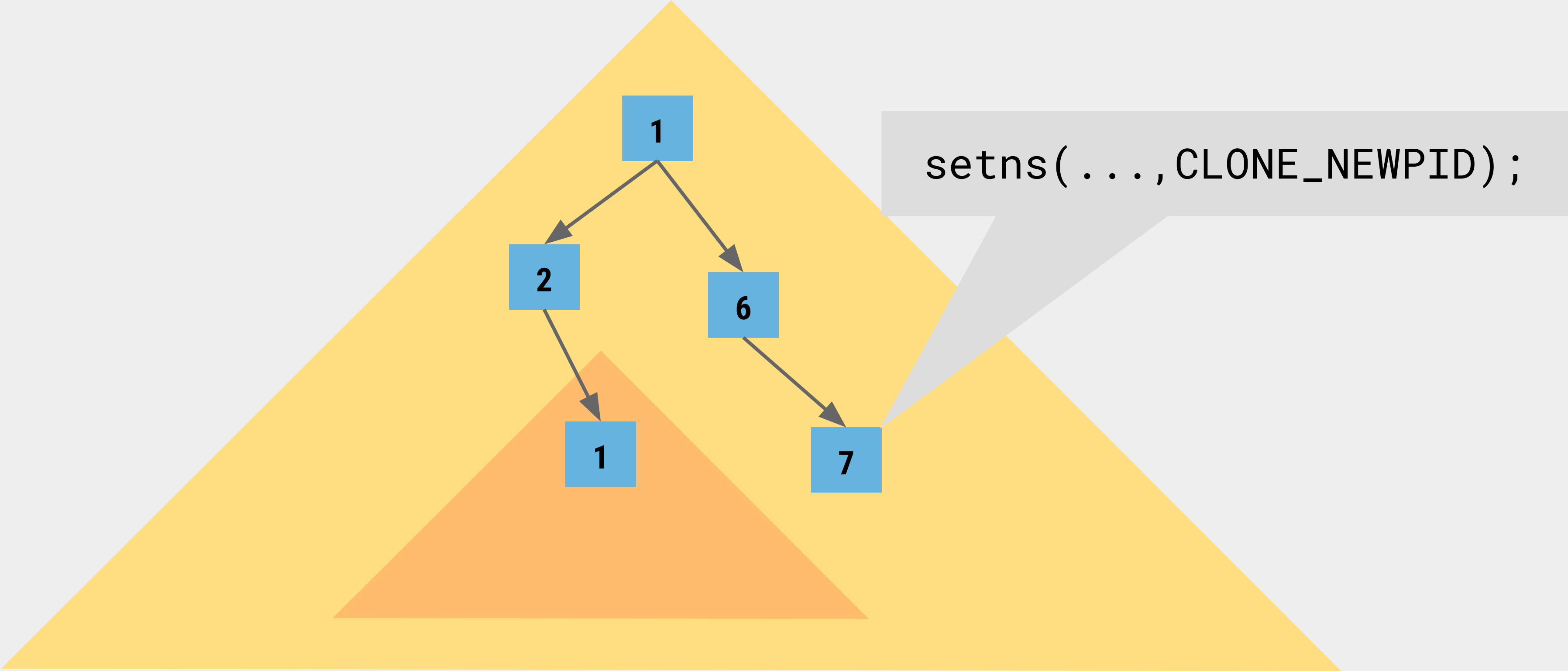
```
rkt run …
```

- uses unshare() to create a new network namespace
- uses clone() to start the first process in the container with a new pid namespace

```
rkt enter …
```

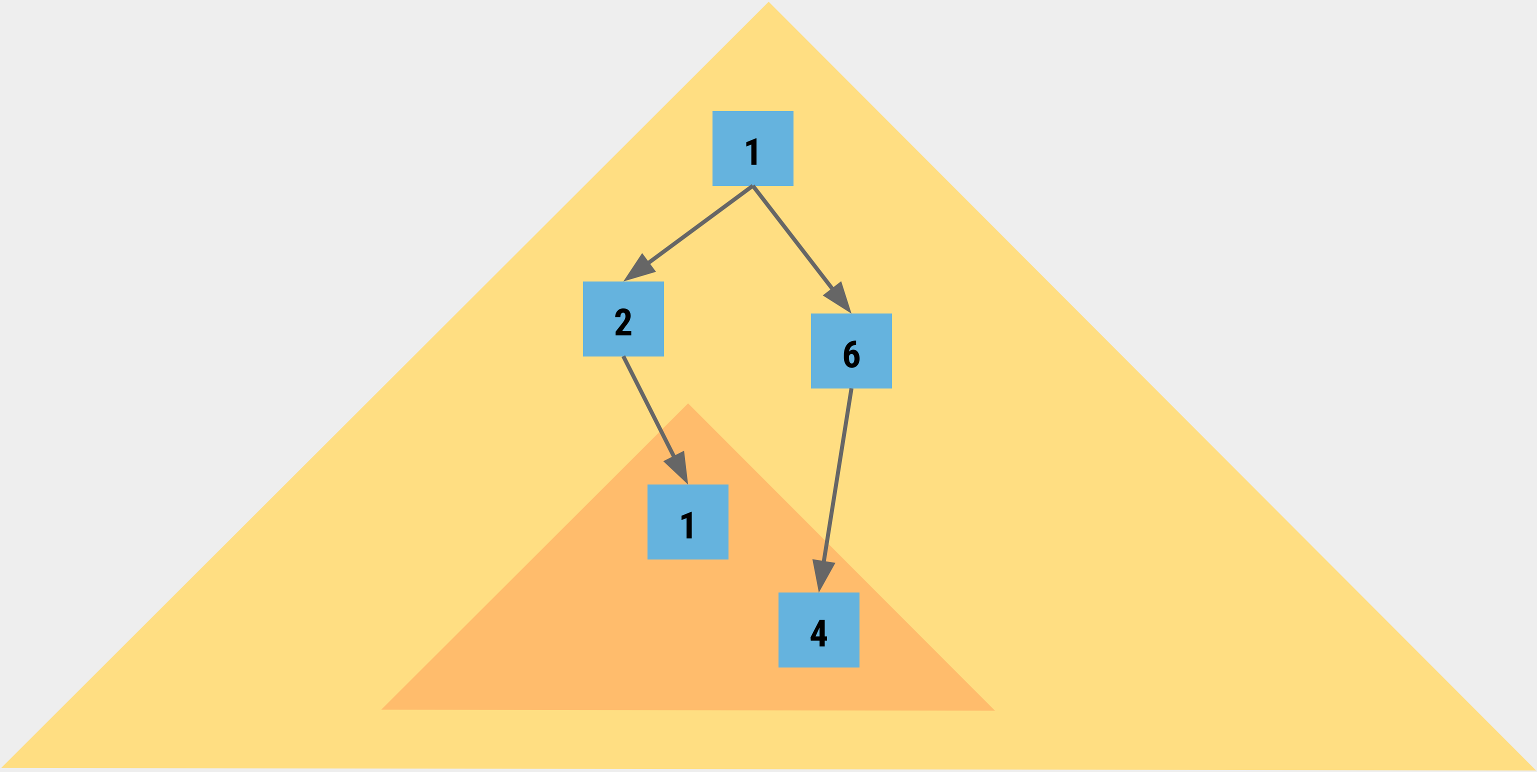
- uses setns() to enter an existing namespace

Joining an existing namespace



```
setns(...,CLONE_NEWPID);
```

Joining an existing namespace



When does PID translation happen?

- the kernel always show the
- getpid(), getppid()
- /proc
- /sys/fs/cgroup/<subsys>/.../cgroup.procs
- credentials passed in Unix sockets (SCM_CREDS)
- pid = fork()

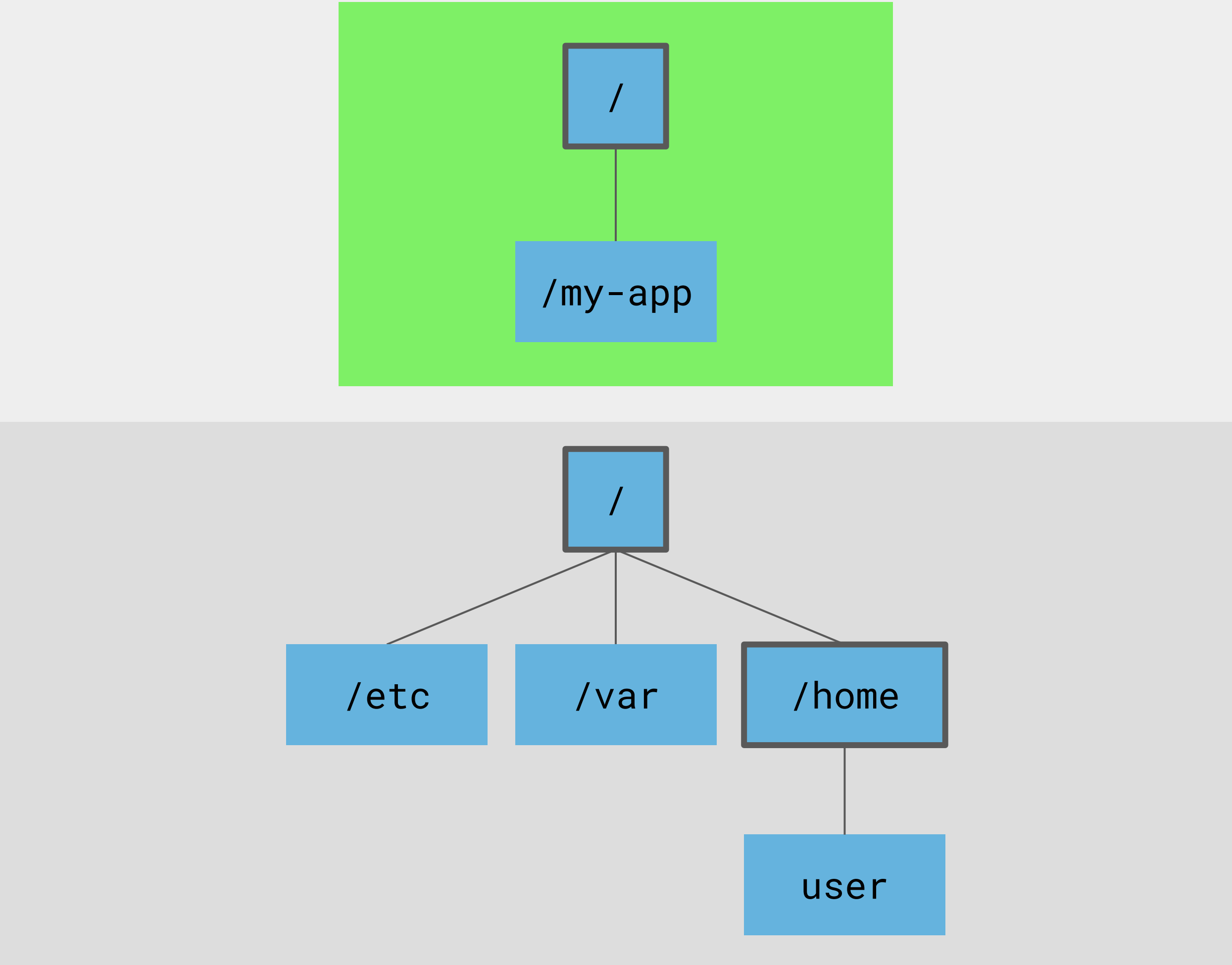
Future:

- Possibly: getvpid() patch being discussed

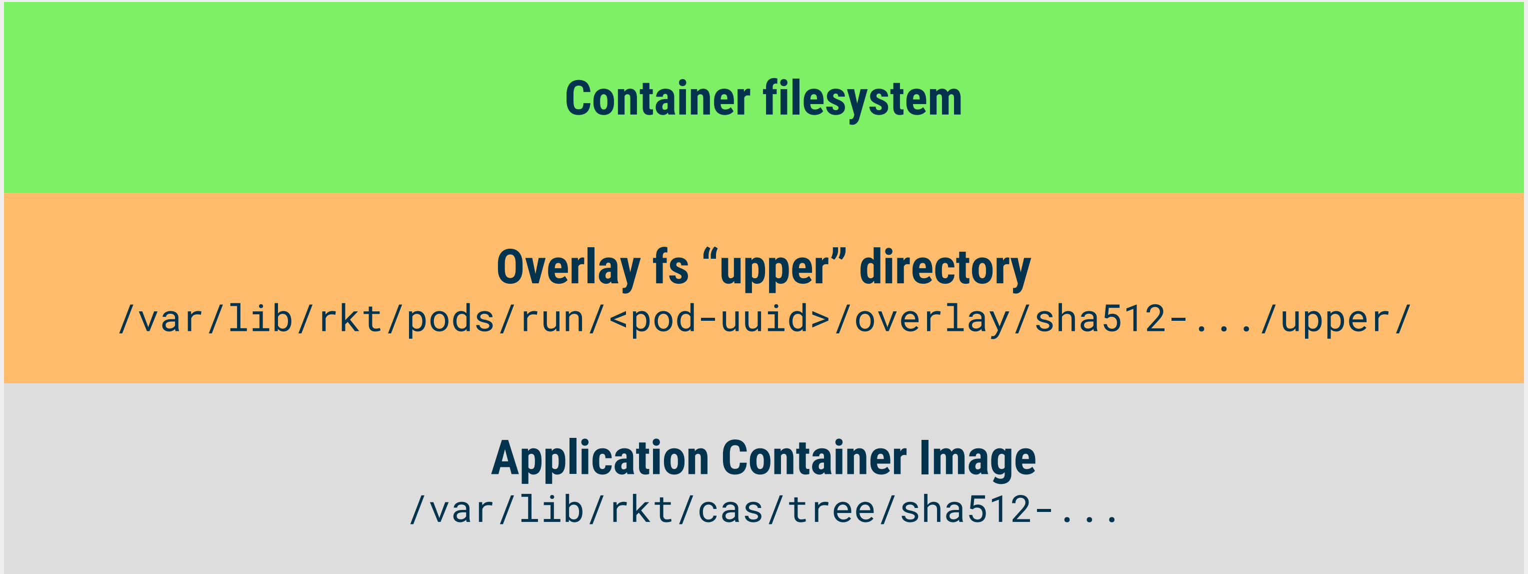
Mount namespaces

container

host



Storing the container data (Copy-on-write)



Container filesystem

Overlay fs “upper” directory

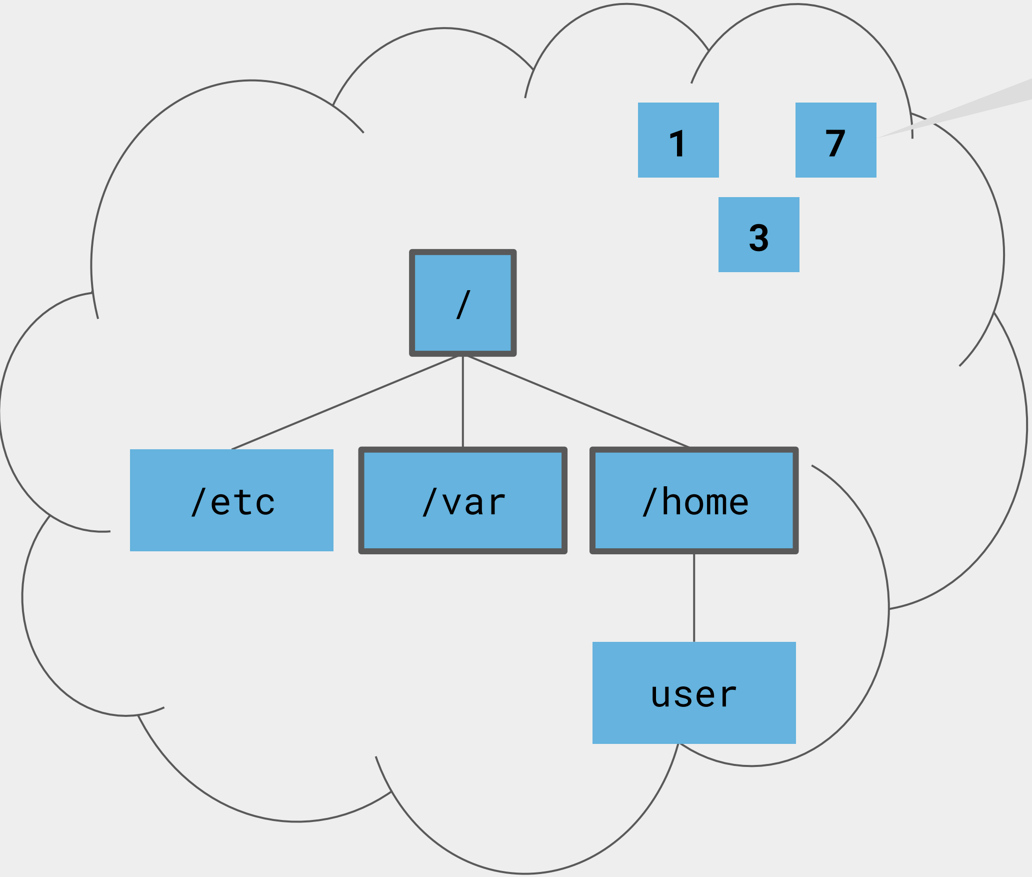
`/var/lib/rkt/pods/run/<pod-uuid>/overlay/sha512-.../upper/`

Application Container Image

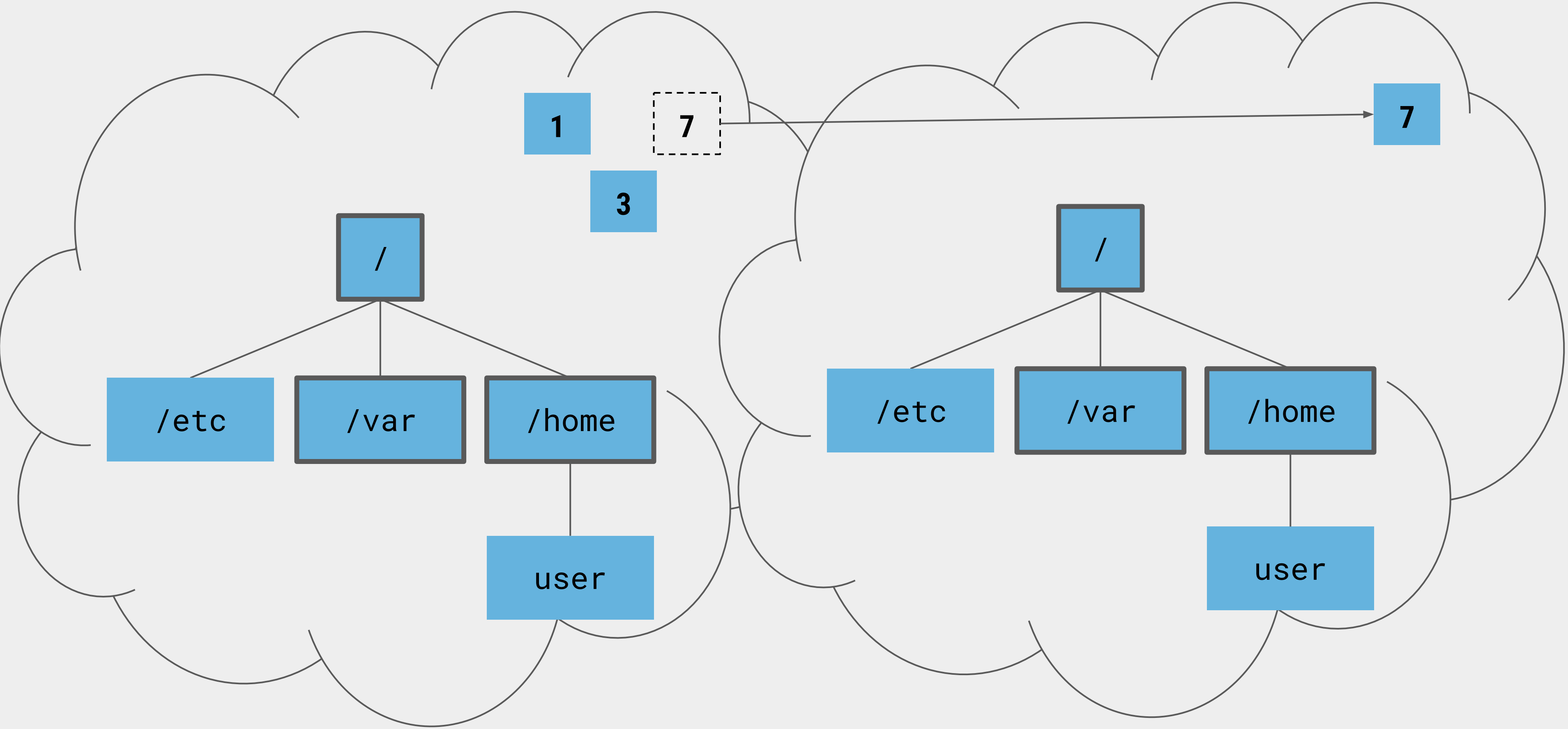
`/var/lib/rkt/cas/tree/sha512-...`

rkt directories

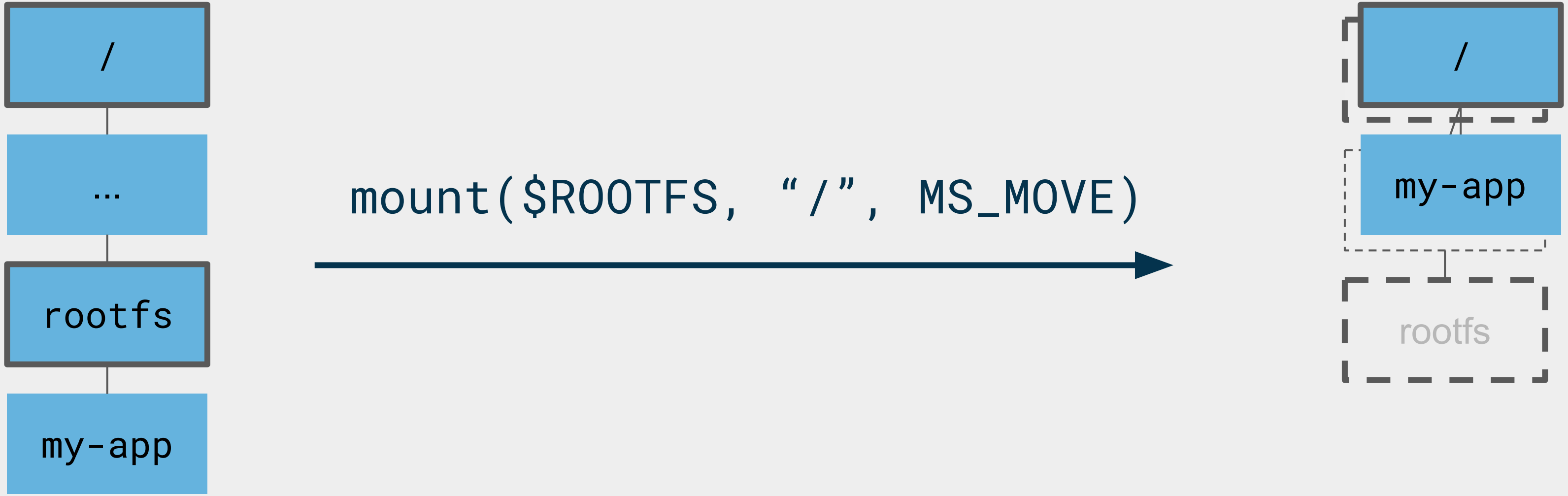
```
/var/lib/rkt
 ├─ cas
 │   └─ tree
 │       ├─ deps-sha512-19bf...
 │       └─ deps-sha512-a5c2...
 └─ pods
    └─ run
        └─ e0ccc8d8
            └─ overlay/sha512-19bf.../upper
            └─ stage1/rootfs/
```



```
unshare(..., CLONE_NEWNS);
```


1
7
3
/
/etc
/var
/home
user
7
/
/etc
/var
/home
user

Changing root with MS_MOVE



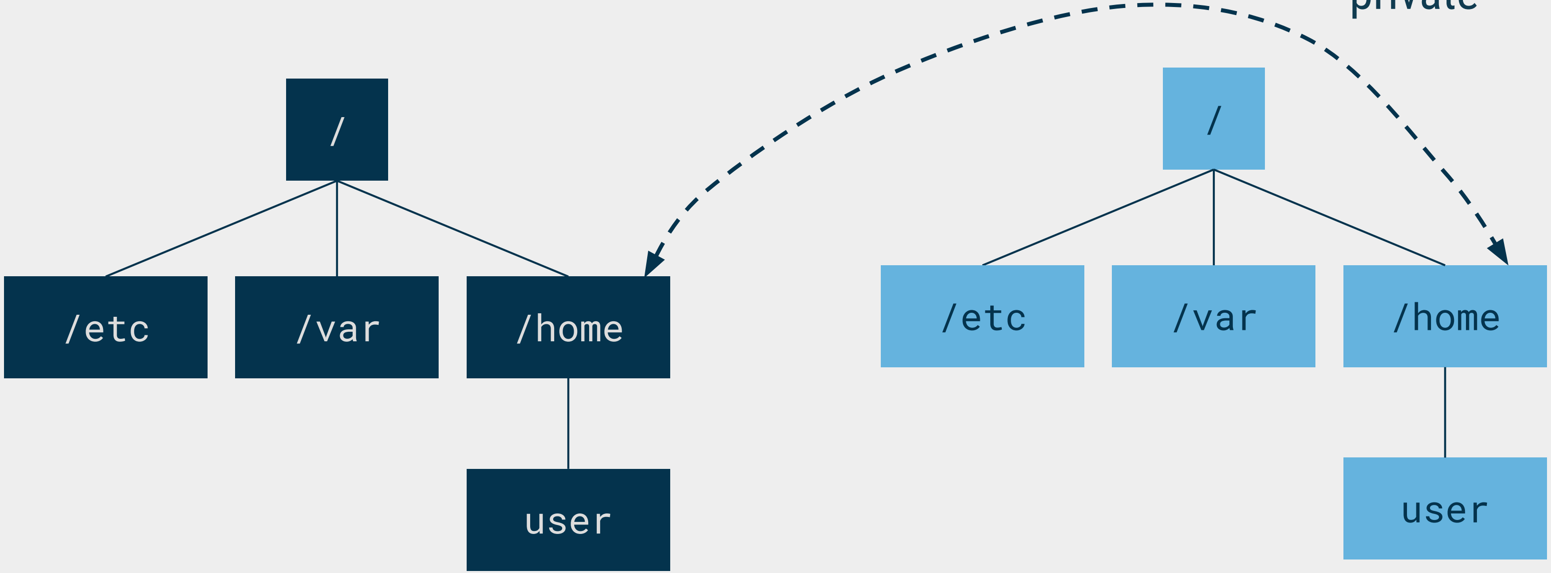
```
mount($ROOTFS, “/”, MS_MOVE)
```

```
$ROOTFS = /var/lib/rkt/pods/run/e0ccc8d8.../stage1/rootfs
```

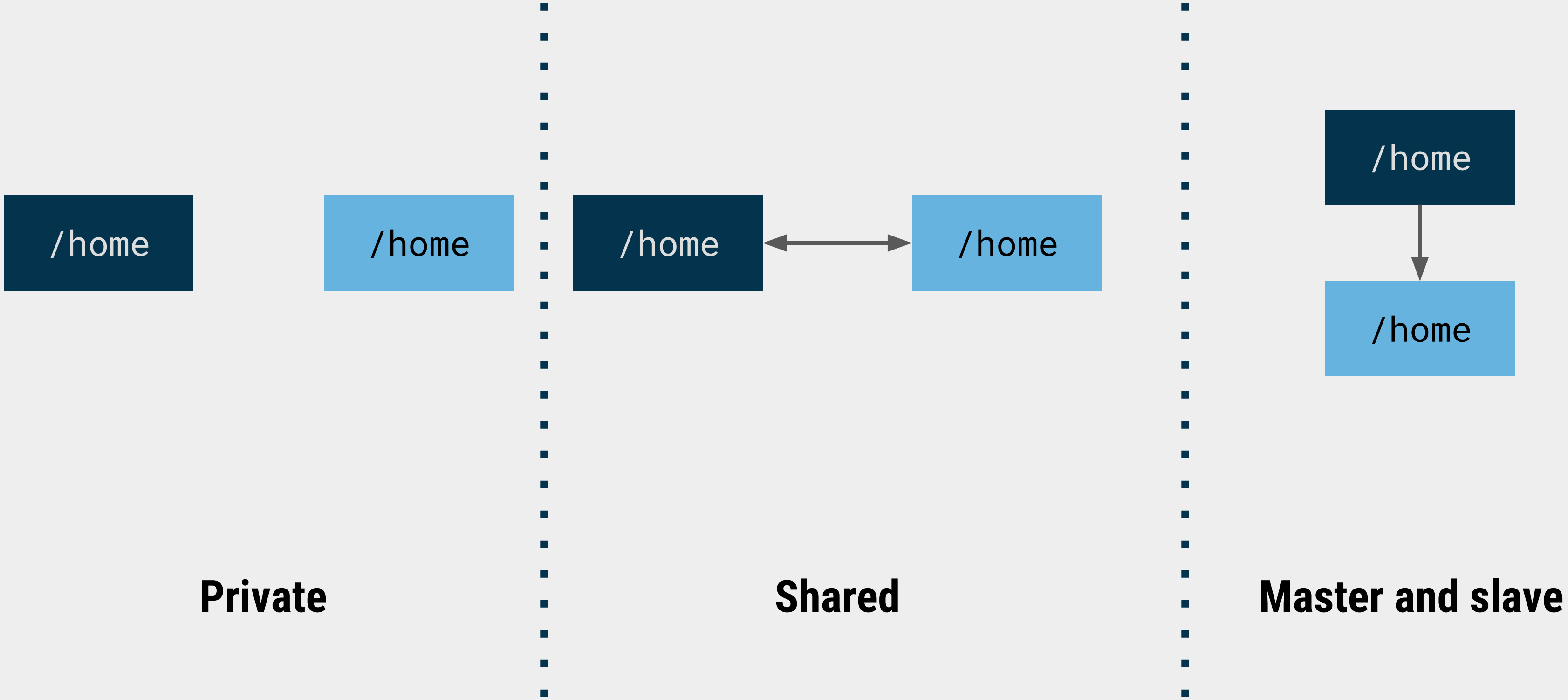
Mount propagation events

Relationship between the two mounts:

- shared
- master / slave
- private



Mount propagation events



/home

/home

Private

/home

/home

Shared

/home

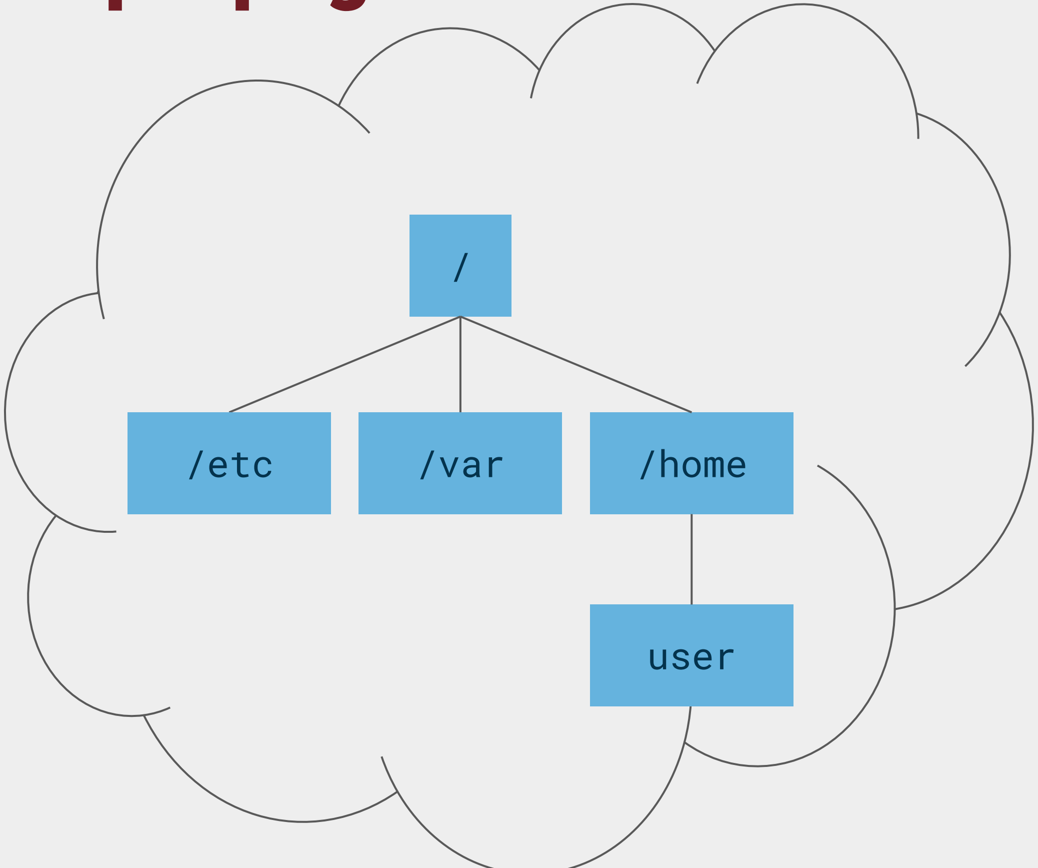
/home

Master and slave

How rkt uses mount propagation events

- **/ in the container namespace is recursively set as slave:**

```
mount(NULL, "/", NULL,
    MS_SLAVE|MS_REC, NULL)
```

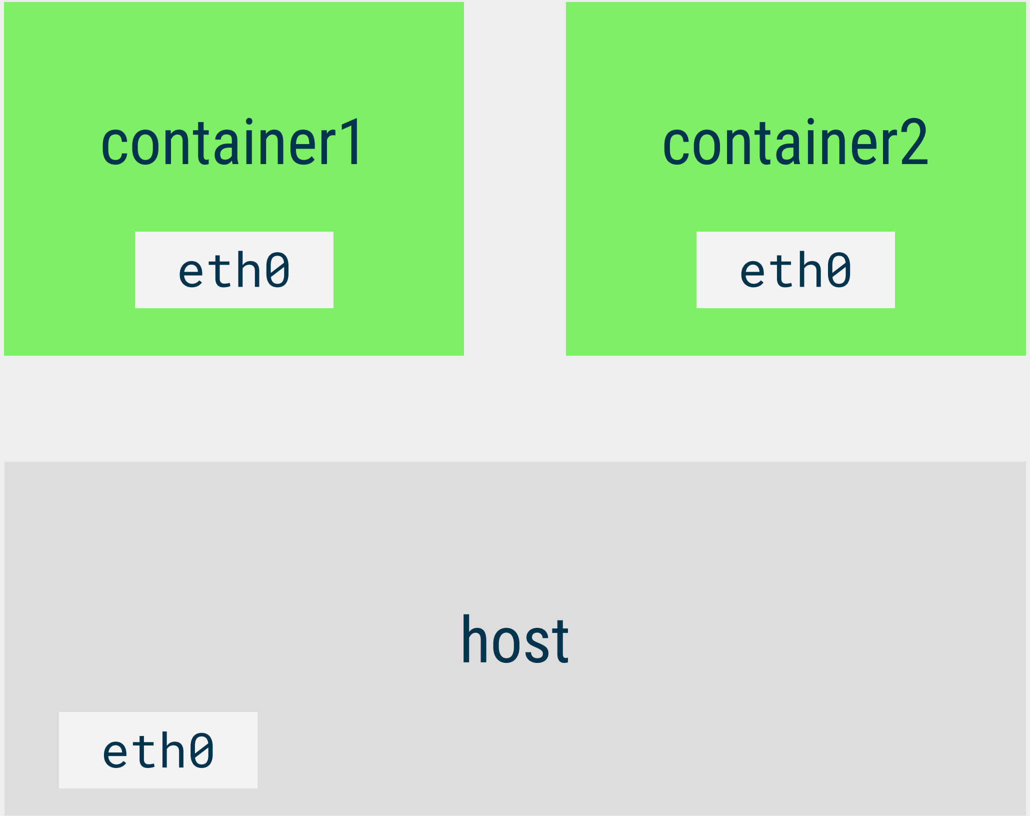


Network namespace

Network isolation

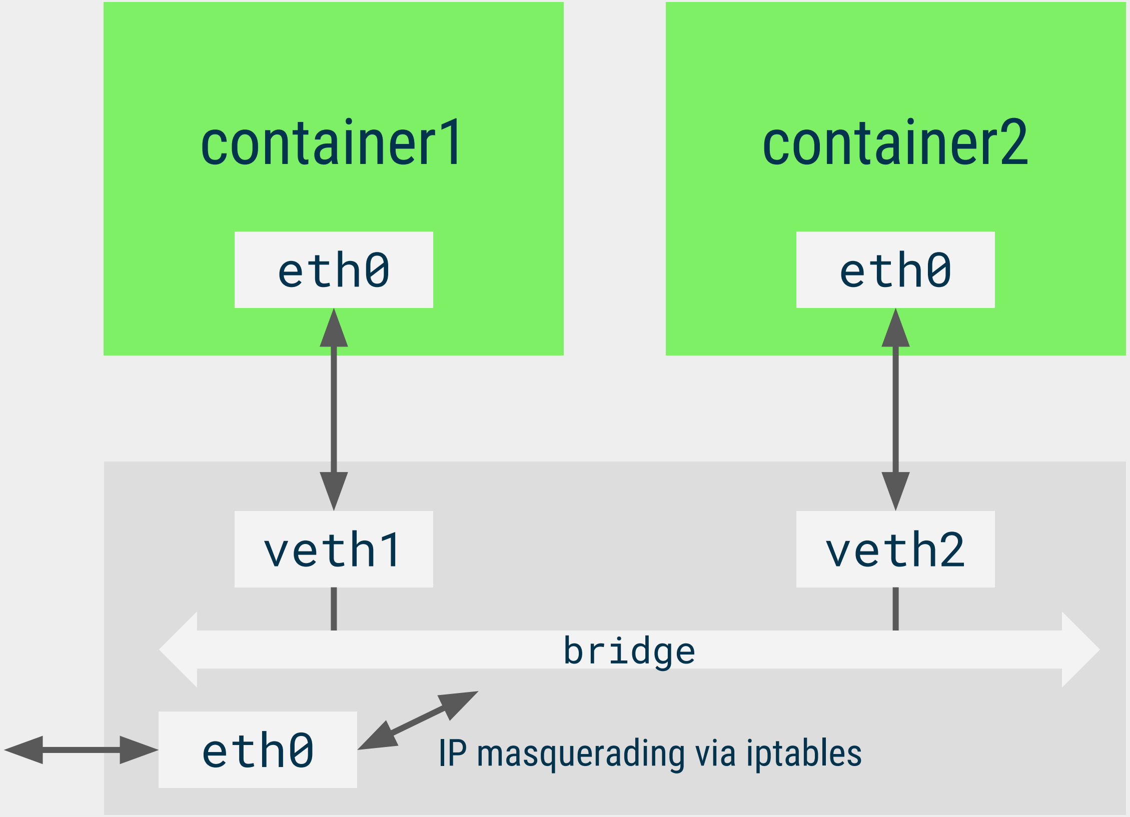
Goal:

- each container has their own network interfaces
- Cannot see the network traffic outside the container (e.g. tcpdump)



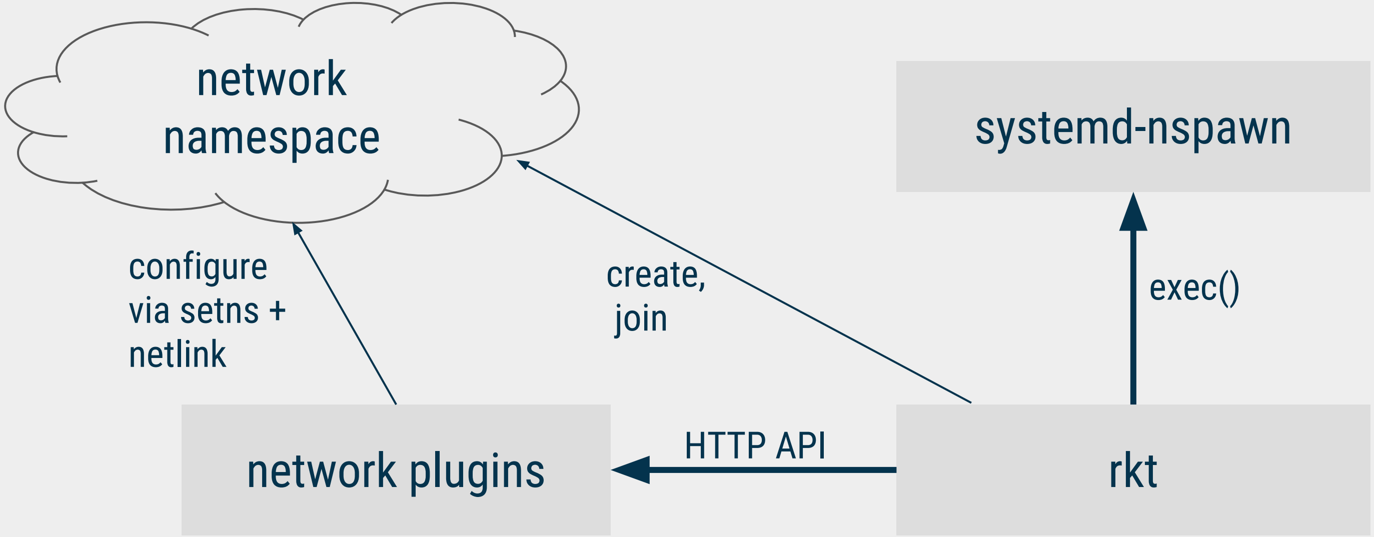
Network tooling

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



How does rkt do it?

- rkt uses the network plugins implemented by the Container Network Interface (CNI, https://github.com/appc/cni)



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

User namespaces

History of Linux namespaces

✓ 1991: Linux

✓ 2002: namespaces in Linux 2.4.19

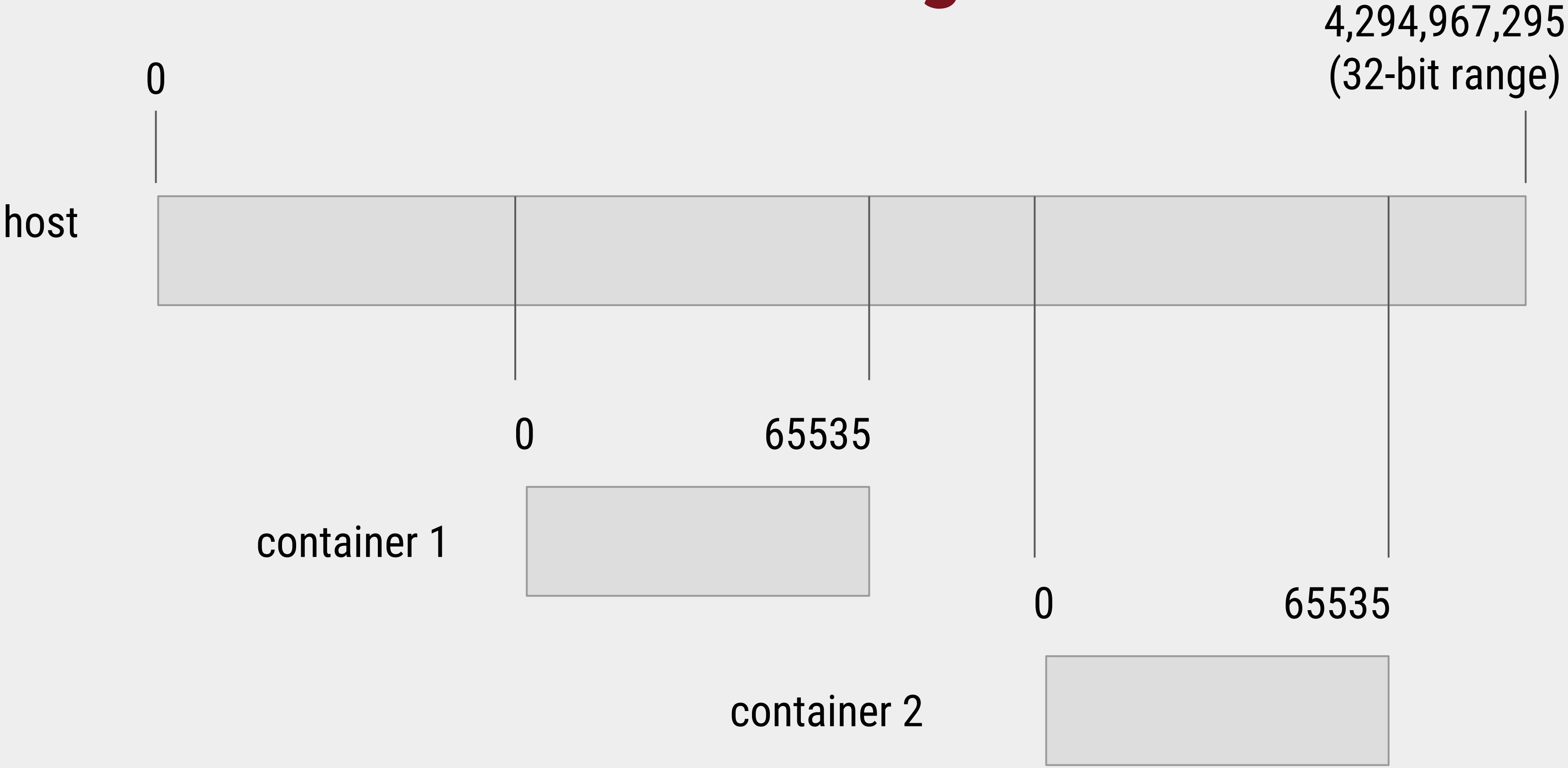
✓ 2008: LXC
✓ 2011: systemd-nspawn
✓ 2013: **user namespaces** in Linux 3.8
✓ 2013: Docker
✓ 2014: rkt

… development still active

Why user namespaces?

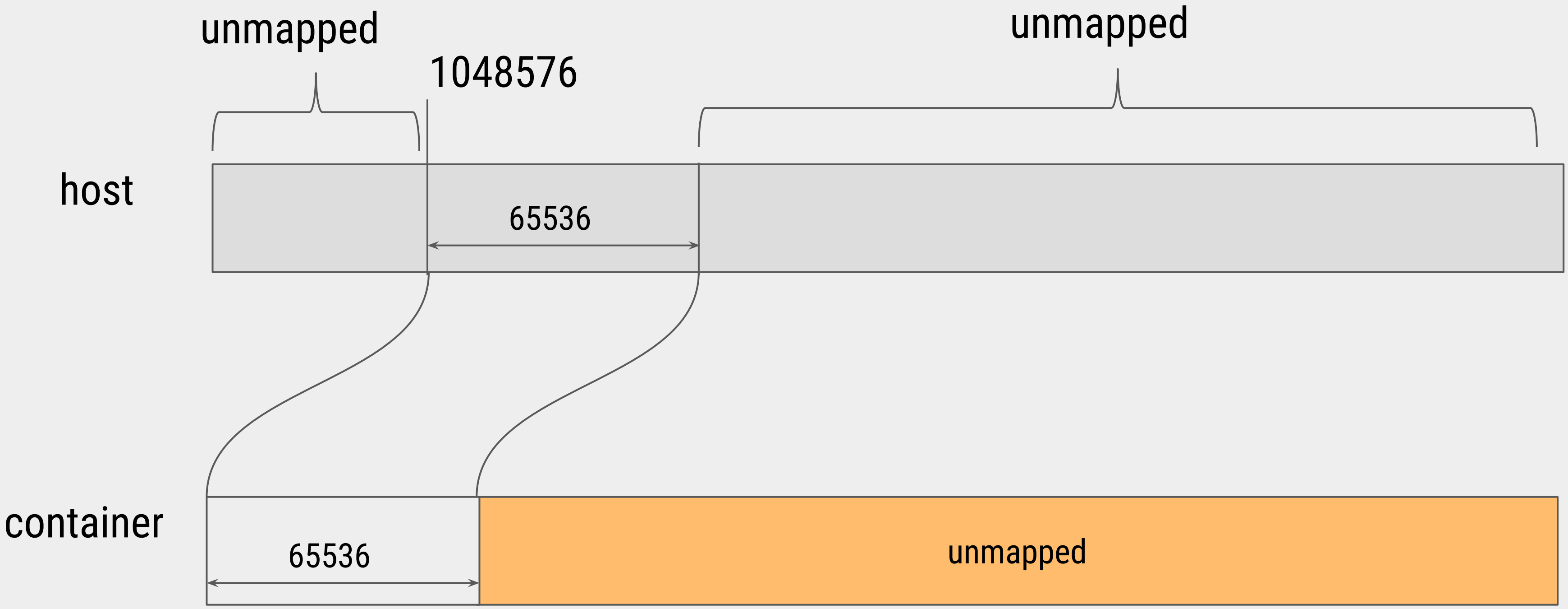
- Better isolation
- Run applications which would need more capabilities
- Per user limits
- Future:
  - Unprivileged containers: possibility to have container without root

User ID ranges

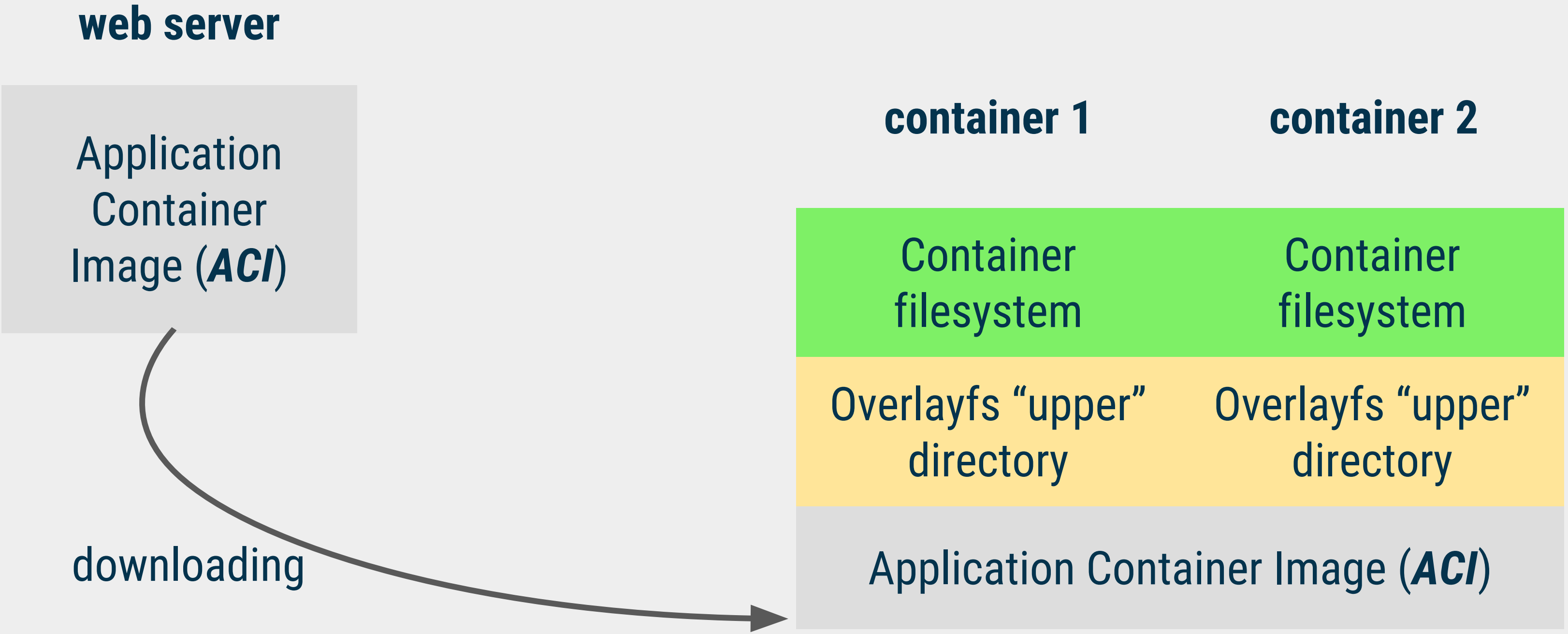


User ID mapping

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



Problems with container images



web server

Application Container Image (***ACI***)

downloading

container 1

container 2

Container filesystem	Container filesystem
Overlayfs “upper” directory	Overlayfs “upper” directory

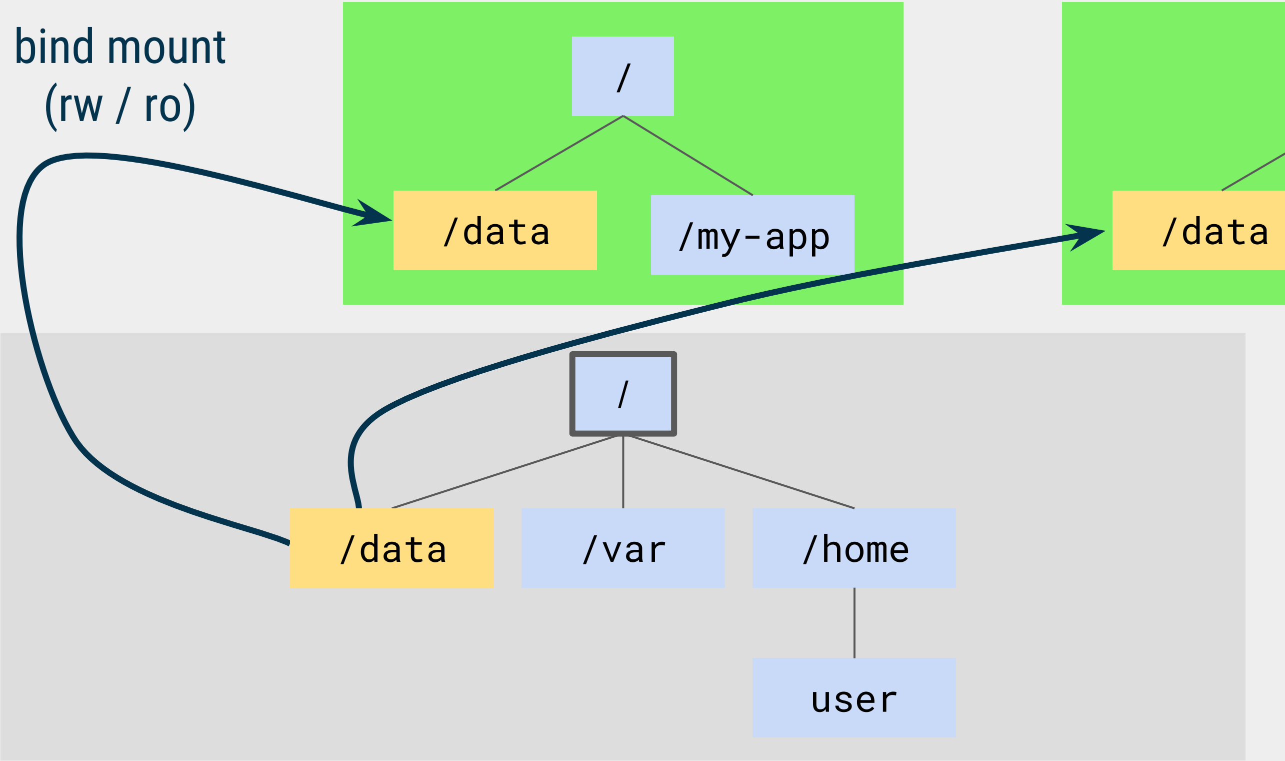
Application Container Image (***ACI***)

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
- Dynamic UID maps

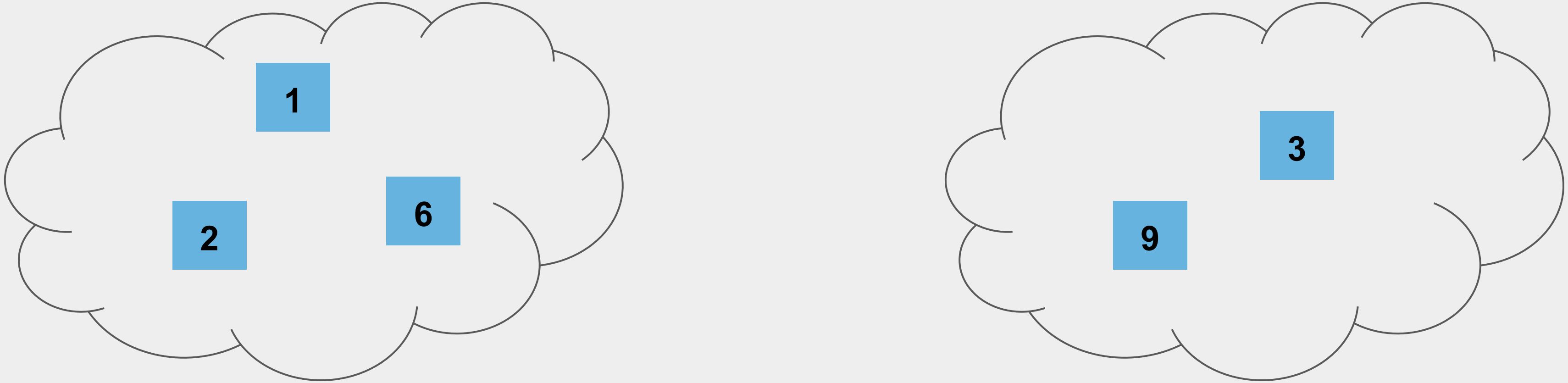


User namespace and filesystem problem

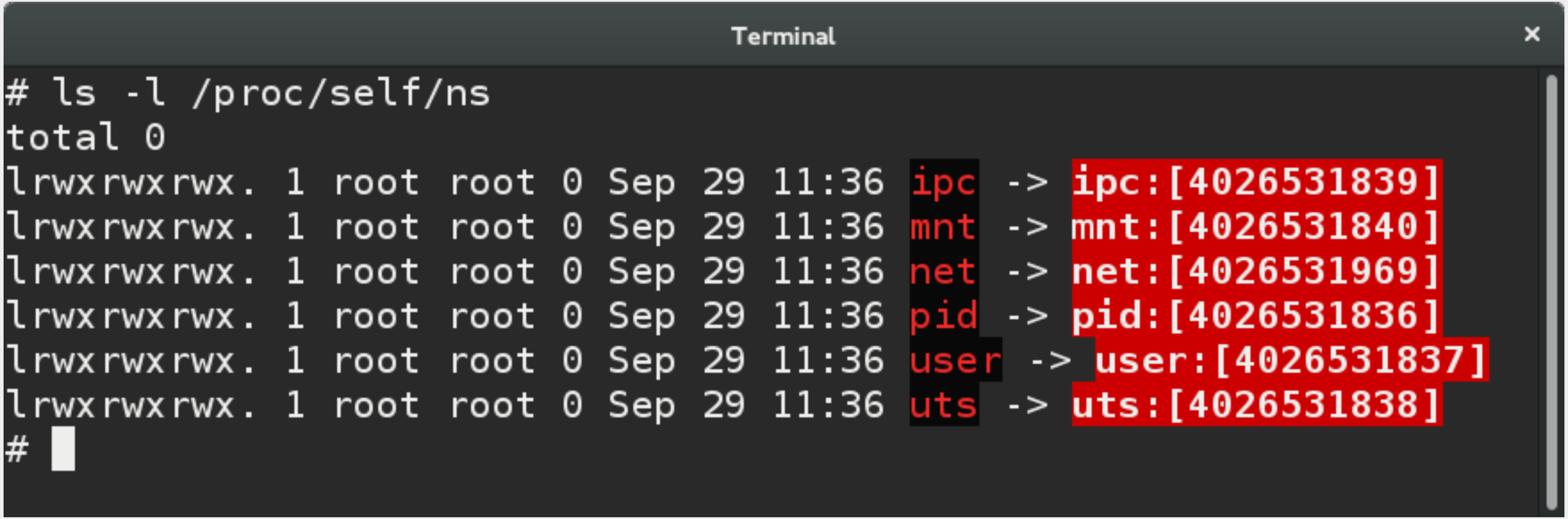
- Possible solution: add options to mount() to apply a UID mapping
- rkt would use it when mounting:
  - the overlay rootfs
  - volumes
- Idea suggested on kernel mailing lists

Namespace lifecycle

Namespace references



Namespace file descriptor



```
# ls -l /proc/self/ns
total 0
lrwxrwxrwx. 1 root root 0 Sep 29 11:36 ipc -> ipc:[4026531839]
lrwxrwxrwx. 1 root root 0 Sep 29 11:36 mnt -> mnt:[4026531840]
lrwxrwxrwx. 1 root root 0 Sep 29 11:36 net -> net:[4026531969]
lrwxrwxrwx. 1 root root 0 Sep 29 11:36 pid -> pid:[4026531836]
lrwxrwxrwx. 1 root root 0 Sep 29 11:36 user -> user:[4026531837]
lrwxrwxrwx. 1 root root 0 Sep 29 11:36 uts -> uts:[4026531838]
# 
```

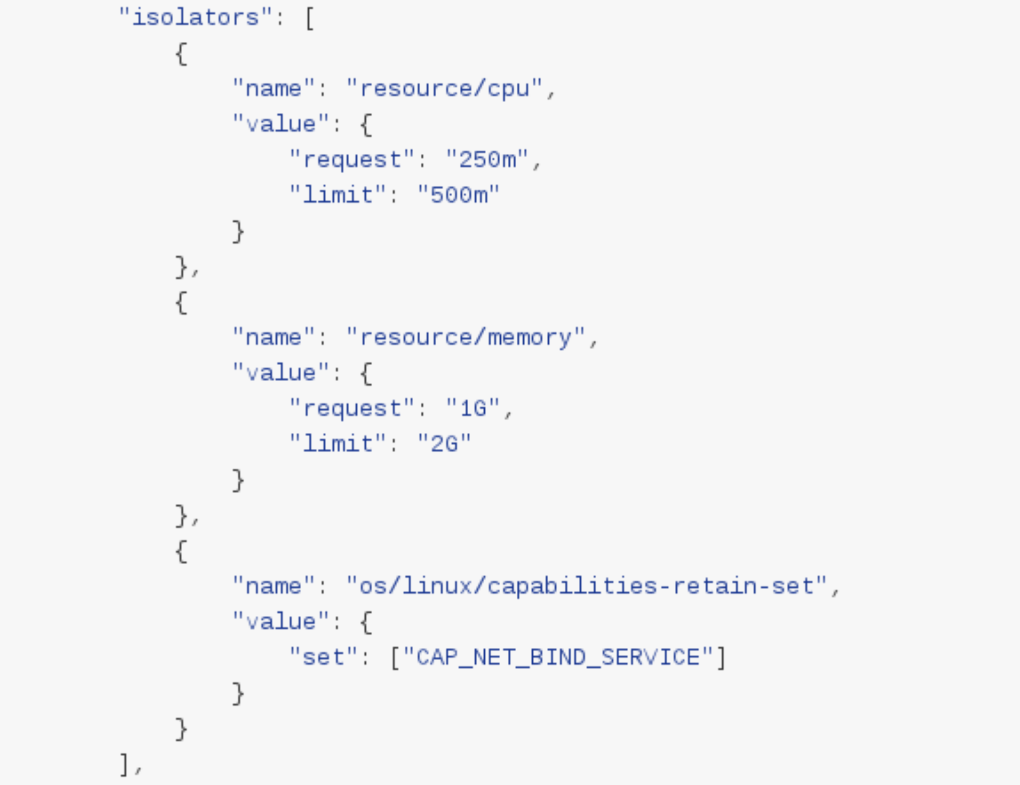
These files can be opened, bind mounted, fd-passed (SCM_RIGHTS)

Isolators

Isolators in rkt

- specified
  in an image manifest

- limiting capabilities
  or resources



```
"isolators": [
    {
        "name": "resource/cpu",
        "value": {
            "request": "250m",
            "limit": "500m"
        }
    },
    {
        "name": "resource/memory",
        "value": {
            "request": "1G",
            "limit": "2G"
        }
    },
    {
        "name": "os/linux/capabilities-retain-set",
        "value": {
            "set": ["CAP_NET_BIND_SERVICE"]
        }
    }
],
```

Isolators in rkt

Currently implemented

- capabilities
- cpu
- memory

Possible additions

- block-bandwidth
- block-iops
- network-bandwidth
- disk-space

Capabilities (1/3)

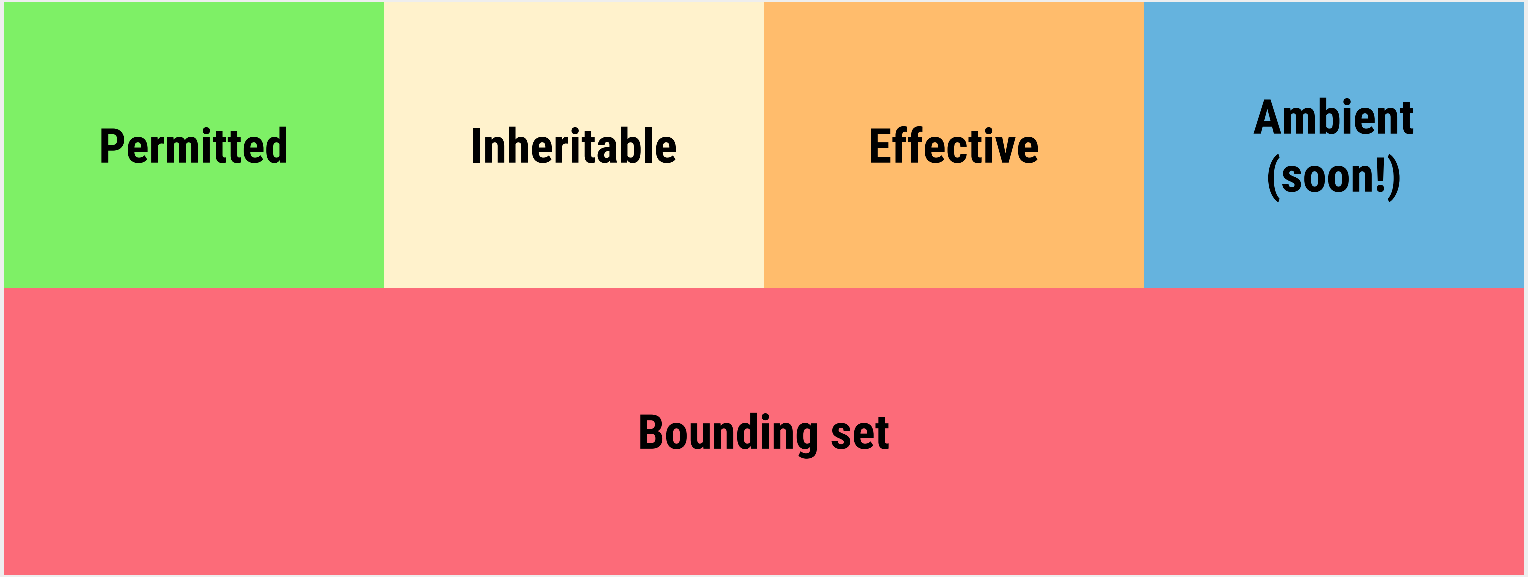
- **Old model (before Linux 2.2):**
  - **User *root* (user id = 0) can do everything**
  - **Regular users are limited**

- **Now: processes have capabilities**

Configuring the network	`CAP_NET_ADMIN`
Mounting a filesystem	`CAP_SYS_ADMIN`
Creating a block device	`CAP_MKNOD`
etc.	37 different capabilities today

Capabilities (2/3)

- Each process has several capability sets:



Permitted

Inheritable

Effective

Ambient (soon!)

Bounding set

Capabilities (3/3)

Other security mechanisms:

- Mandatory Access Control (MAC) with Linux Security Modules (LSMs):
  - SELinux
  - AppArmor…
- seccomp

Isolator: memory and cpu

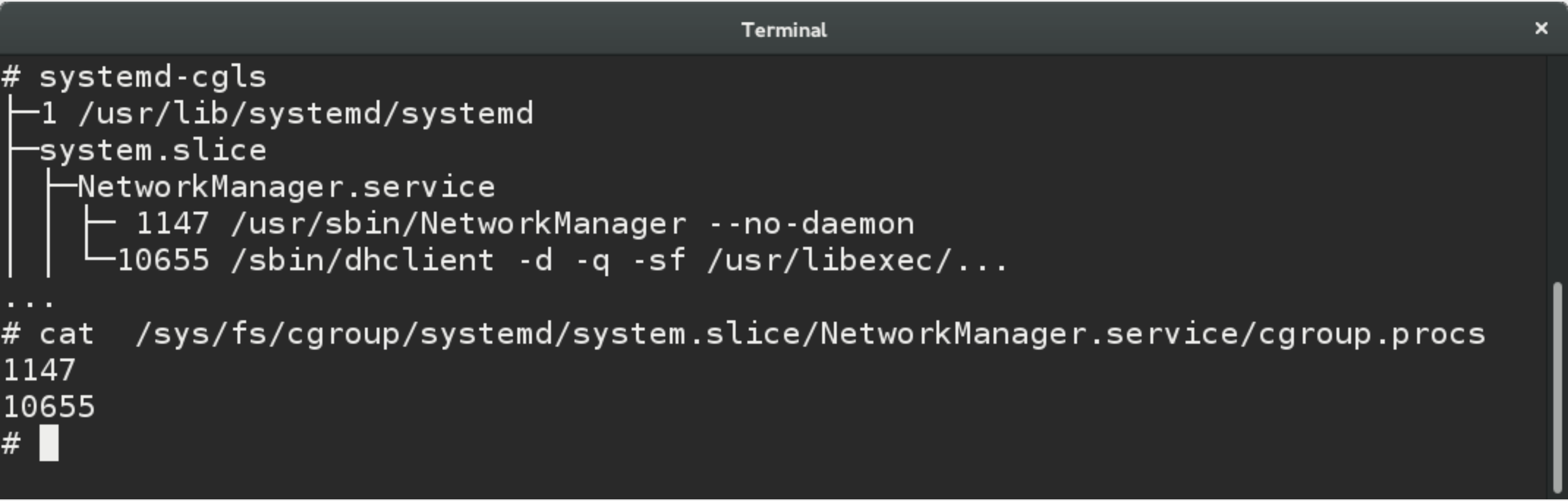
- based on cgroups

cgroups

What’s a control group (cgroup)

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

cgroups



```
# 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

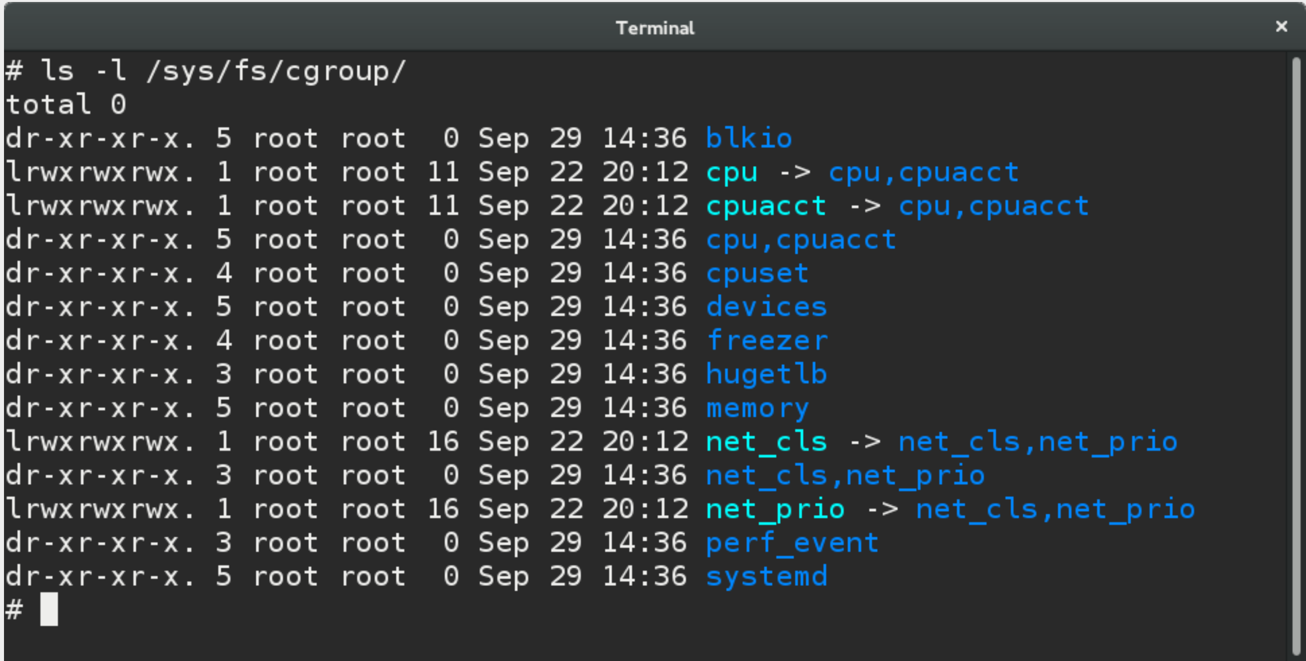
```
/sys/fs/cgroup/*/

/proc/cgroups

/proc/$PID/cgroup
```

List of cgroup controllers

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



```
# ls -l /sys/fs/cgroup/
total 0
dr-xr-xr-x. 5 root root  0 Sep 29 14:36 blkio
lrwxrwxrwx. 1 root root 11 Sep 22 20:12 cpu -> cpu,cpuacct
lrwxrwxrwx. 1 root root 11 Sep 22 20:12 cpuacct -> cpu,cpuacct
dr-xr-xr-x. 5 root root  0 Sep 29 14:36 cpu,cpuacct
dr-xr-xr-x. 4 root root  0 Sep 29 14:36 cpuset
dr-xr-xr-x. 5 root root  0 Sep 29 14:36 devices
dr-xr-xr-x. 4 root root  0 Sep 29 14:36 freezer
dr-xr-xr-x. 3 root root  0 Sep 29 14:36 hugetlb
dr-xr-xr-x. 5 root root  0 Sep 29 14:36 memory
lrwxrwxrwx. 1 root root 16 Sep 22 20:12 net_cls -> net_cls,net_prio
dr-xr-xr-x. 3 root root  0 Sep 29 14:36 net_cls,net_prio
lrwxrwxrwx. 1 root root 16 Sep 22 20:12 net_prio -> net_cls,net_prio
dr-xr-xr-x. 3 root root  0 Sep 29 14:36 perf_event
dr-xr-xr-x. 5 root root  0 Sep 29 14:36 systemd
# 
```

How systemd units use cgroups

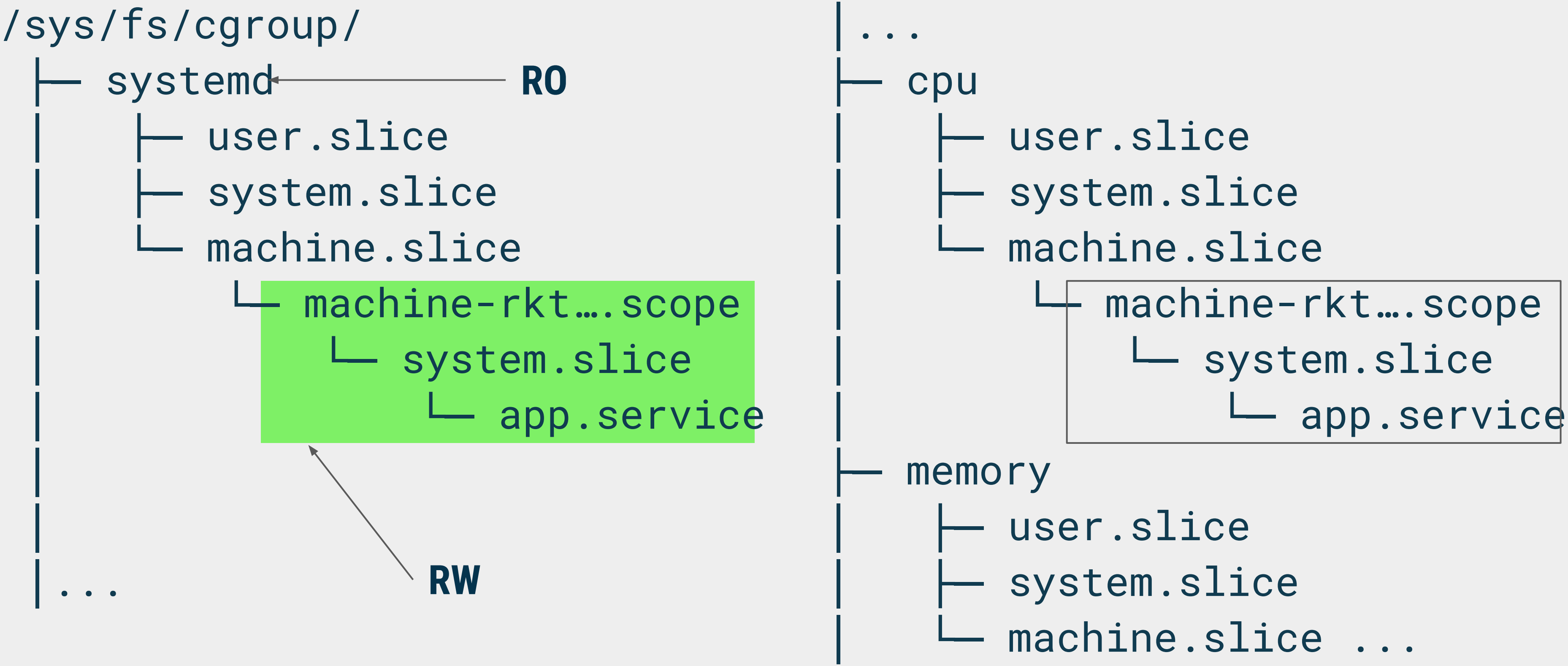
```
/sys/fs/cgroup/
 ├─ systemd
 │   ├─ user.slice
 │   ├─ system.slice
 │   │   ├─ NetworkManager.service
 │   │   │   └─ cgroups.procs
 │   │   ...
 │   └─ machine.slice
 │
```

How systemd units use cgroups w/ containers

```
/sys/fs/cgroup/
 ├─ systemd
 │    ├─ user.slice
 │    ├─ system.slice
 │    └─ machine.slice
 │         └─ machine-rkt….scope
 │              └─ system.slice
 │                   └─ app.service
 │
 │
 │...
```

```
│...
├─ cpu
│    ├─ user.slice
│    ├─ system.slice
│    └─ machine.slice
│         └─ machine-rkt….scope
│              └─ system.slice
│                   └─ app.service
├─ memory
│    ├─ user.slice
│    ├─ system.slice
│    └─ machine.slice ...
```

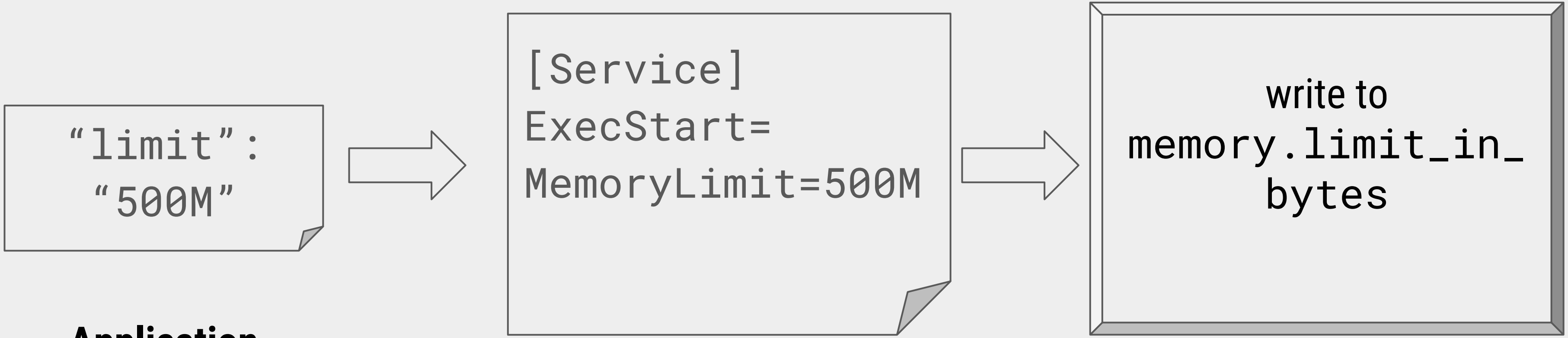
cgroups mounted in the container



```
/sys/fs/cgroup/
 ├─ systemd         RO
 |   ├─ user.slice
 |   ├─ system.slice
 |   └─ machine.slice
 |       └─ machine-rkt….scope
 |           └─ system.slice
 |               └─ app.service
 |
 |
 |...              RW
```

```
|...
├─ cpu
|   ├─ user.slice
|   ├─ system.slice
|   └─ machine.slice
|       └─ machine-rkt….scope
|           └─ system.slice
|               └─ app.service
├─ memory
|   ├─ user.slice
|   ├─ system.slice
|   └─ machine.slice ...
```

Memory isolator



```
"limit":
"500M"
```

Application
Image Manifest

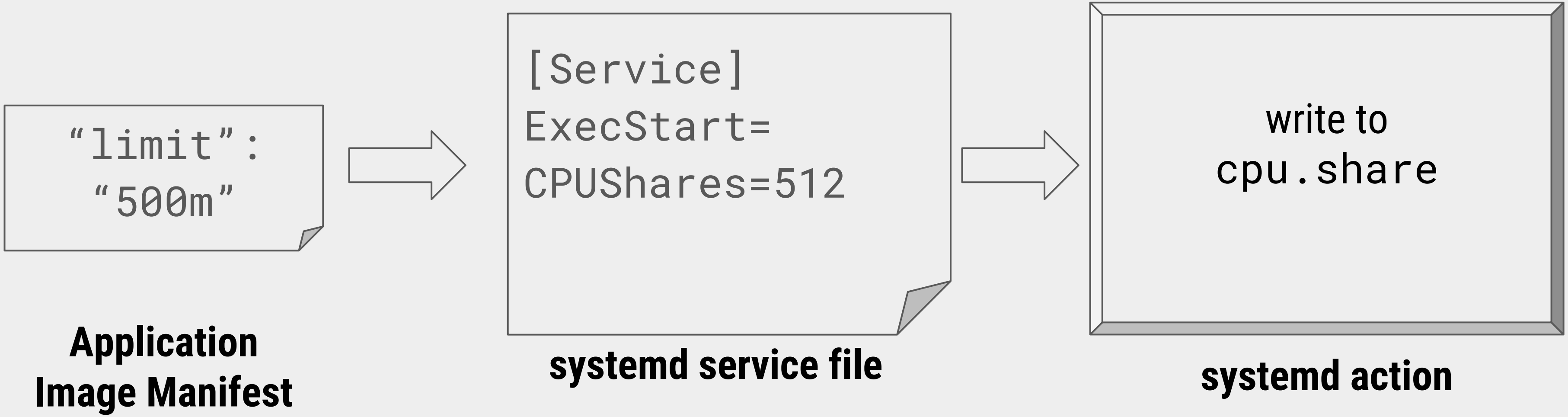
```
[Service]
ExecStart=
MemoryLimit=500M
```

systemd service file

write to `memory.limit_in_bytes`

systemd action

CPU isolator



“limit”: “500m”

Application Image Manifest

```
[Service]
ExecStart=
CPUShares=512
```

systemd service file

write to `cpu.share`

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

Isolator: network

- limit the network bandwidth
- cgroup controller “net_cls” to tag packets emitted by a process
- iptables / traffic control to apply on tagged packets
- open question: allocation of tags?
- not implemented in rkt yet

Isolator: disk quotas

Disk quotas

Not implemented in rkt

- loop device
- btrfs subvolumes
  - systemd-nspawn can use them
- per user and group quotas
  - not suitable for containers
- per project quotas: in xfs and soon in ext4
  - open question: allocation of project id?

Conclusion

We talked about:

- the isolation provided by rkt
- namespaces
- cgroups
- how rkt uses the namespace & cgroup API

Thanks

Thanks Chris for the theme!