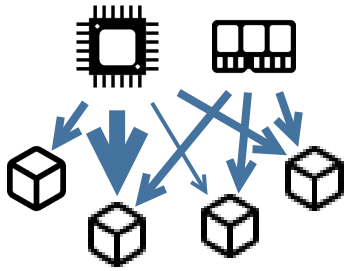
Containerization (e.g. docker)

Ressource Sharing

- Allocate resources to containers proportionally
- Limit resource access



➡ Control Groups (Cgroups)

```
/sys/fs/cgroup
├── group (z.B. docker)
│   ├── cpu.max
│   ├── memory.max
│   ├── procs
│   └── <subgroups>
└── ...
```

Filesystem Isolation

- Assign “own” file system to container
- Prevent exiting from subfile system

➡ `chroot`

```
/
├── bin
├── dev
├── etc
├── ...
├── tmp
│   └── container-fs
│       ├── bin
│       ├── dev
│       ├── etc
│       ├── home
│       └── ...
└── var
```

OS-level Isolation

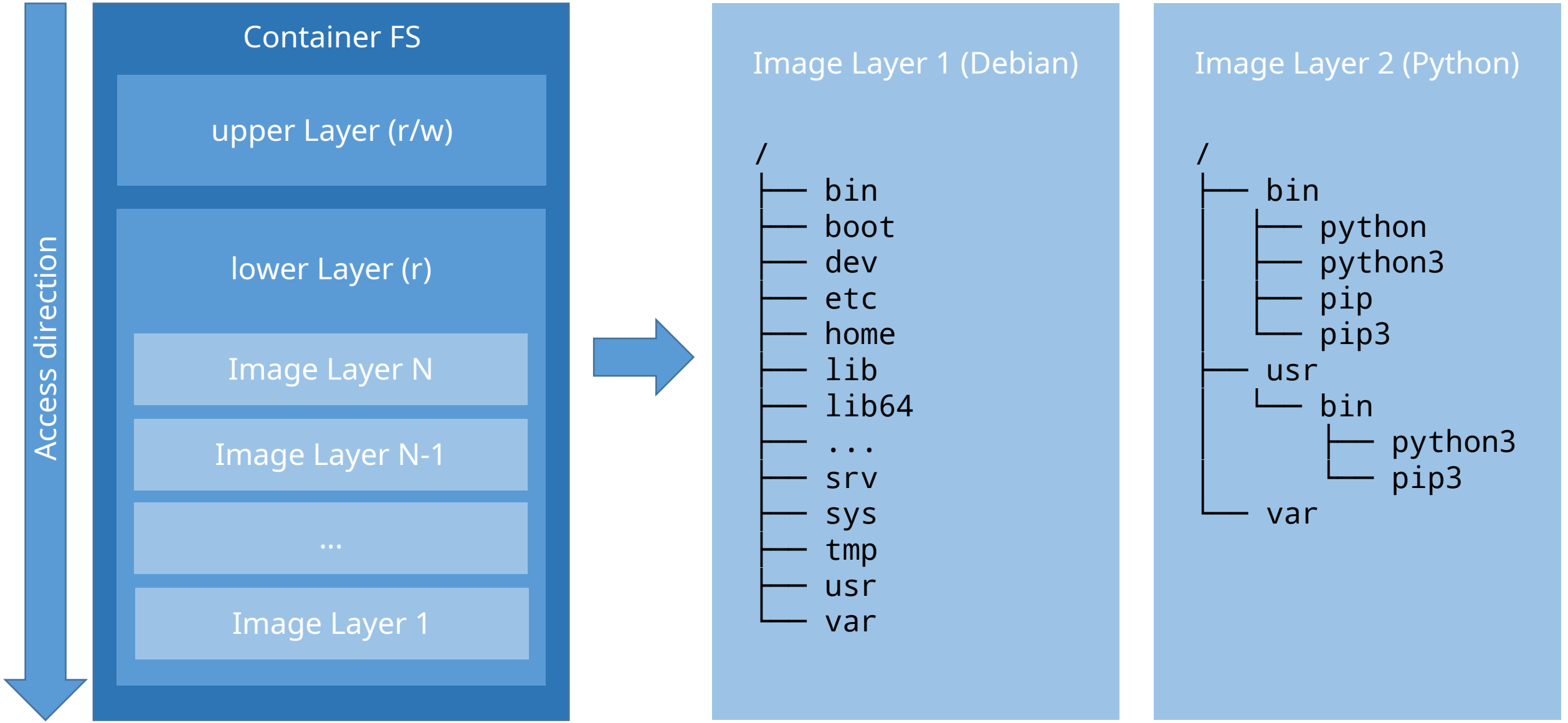
- Isolate Users, Processes, Devices, IPC, ...
- Containers have their own isolated view of the OS

➡ Namespaces

```
PID: 4                 PID: 4
Procs: 1, 3            Procs: 2, 3
Host: MyBox            Host: MyBox
Mounts:                Mounts:
  /mnt/data              /mnt/data
                         /dev/loopX

Host: MyServer
Procs: 1, 7, 8, 92, 104, ...
Mounts: ...
```

OverlayFS


Access direction
Container FS
upper Layer (r/w)
lower Layer (r)
Image Layer N
Image Layer N-1
...
Image Layer 1
Image Layer 1 (Debian)
/
bin
boot
dev
etc
home
lib
lib64
...
srv
sys
tmp
usr
var
Image Layer 2 (Python)
/
bin
python
python3
pip
pip3
usr
bin
python3
pip3
var

Step-by-Step Process

Parent (`main`)

1) setup Cgroups
2) specify limits for Cgroup
3) clone(*with flags*)

Child (`container_function`)

4) move self into Cgroup
5) set hostname
6) mount special paths (e.g. proc, sysfs)
7) chroot
8) chdir
9) exec actual program

Container

10) wait for child

utilize the given `CgroupManager` („`cgroup`“) and the function `enable_subtree_controllers`

Cgroup v2 Sample

```
/sys/fs/cgroup/socker
├── cgroup.subtree_control
├── ...
├── container-a/
│   ├── cgroup.procs
│   ├── memory.max
│   └── ...
├── container-b/
│   ├── cgroup.procs
│   └── ...
```