Advanced Programming in the UNIX Environment

Week 13, Segment 6:

Capabilities, Control Groups, Containers

Department of Computer Science
Stevens Institute of Technology

Jan Schaumann
jschauma@stevens.edu
https://stevens.netmeister.org/631/

POSIX *Capabilities*

With so many things to try to restrict, one approach to more fine grained control are so-called *Capabilities*:

- `CAP_CHOWN` - the ability to chown files
- `CAP_SETUID` - allow setuid
- `CAP_LINUX_IMMUTABLE` - allow append-only or immutable flags
- `CAP_NET_BIND_SERVICE` - allow network sockets <1024
- `CAP_NET_ADMIN` - allow interface configuration, routing table manipulation, …
- `CAP_NET_RAW` - raw packets
- `CAP_SYS_ADMIN` - broad sysadmin privs (mounting file systems, setting hostname, handling swap, …)...

Note the difference in implementation (again); *e.g.,* POSIX, FreeBSD `capsicum(4)`, NetBSD/macOS `kauth(9)`, Linux `capabilities(7)`.

Linux Namespaces

Inspired by Bell Labs’ Plan 9 Operating System, Linux Namespaces partition kernel resources to expose them with granular visibility to processes and process groups:

- mnt — mount points
- pid — process ID visibility
- net — virtualized network stack
- ipc — System V IPC visibility
- uts — Unix Time Sharing (different host- and domain names)
- user — user-IDs and privileges
- time — system time
- cgroup — control groups

Linux Control Groups

Originally termed *process containers*, `cgroups` allow for:

- resource limiting (*e.g.*, memory limit)
- prioritization (*e.g.*, CPU utilization, disk I/O throughput)
- accounting
- control (*e.g.*, freezing, checkpointing, and restarting)

Linux Control Groups

cgroups provide the following controls:

- cpu - ability to schedule tasks
- cpuset - CPUs and memory nodes
- freezer - activity of control groups
- hugetlb - large page support (HugeTLB) usage
- io - block device I/O
- memory - memory, kernel memory, swap memory
- perf_event - ability to monitor threads
- pids - number of processes
- rdma - remote direct memory access

Linux Control Groups

cgroups are implemented as a virtual file system, often under /sys/fs/cgroup:

```
# create a new memory cgroup:
mkdir /sys/fs/cgroup/memory/group0
# move the current shell into the memory controller group:
echo $$ > /sys/fs/cgroup/memory/group0/tasks
# limit the shell’s memory usage:
echo 40M > /sys/fs/cgroup/memory/group0/memory.limit_in_bytes
```

See cgroups(7) for more details.

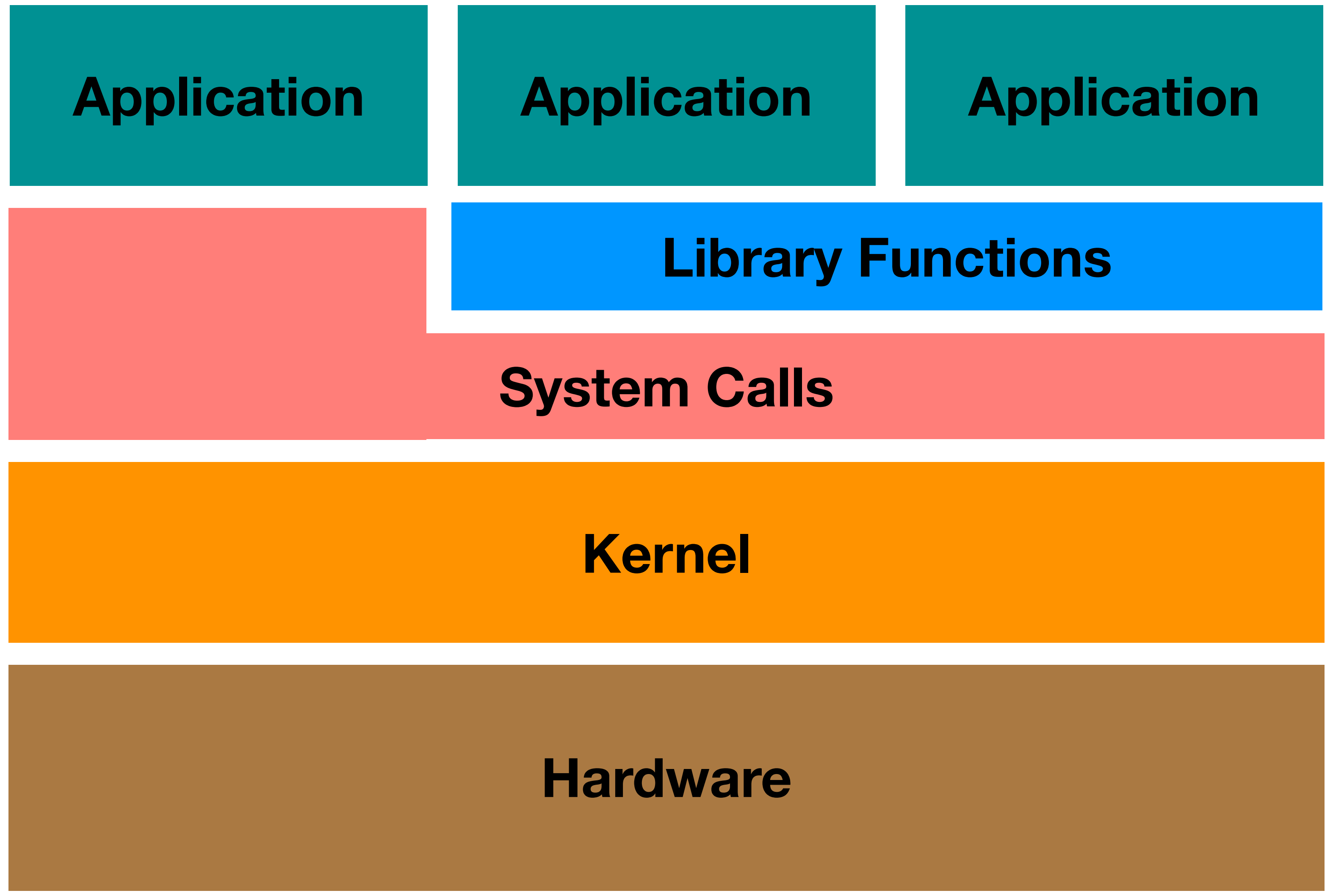
Containers

A *container* is an isolated execution environment providing a form of lightweight virtualization:

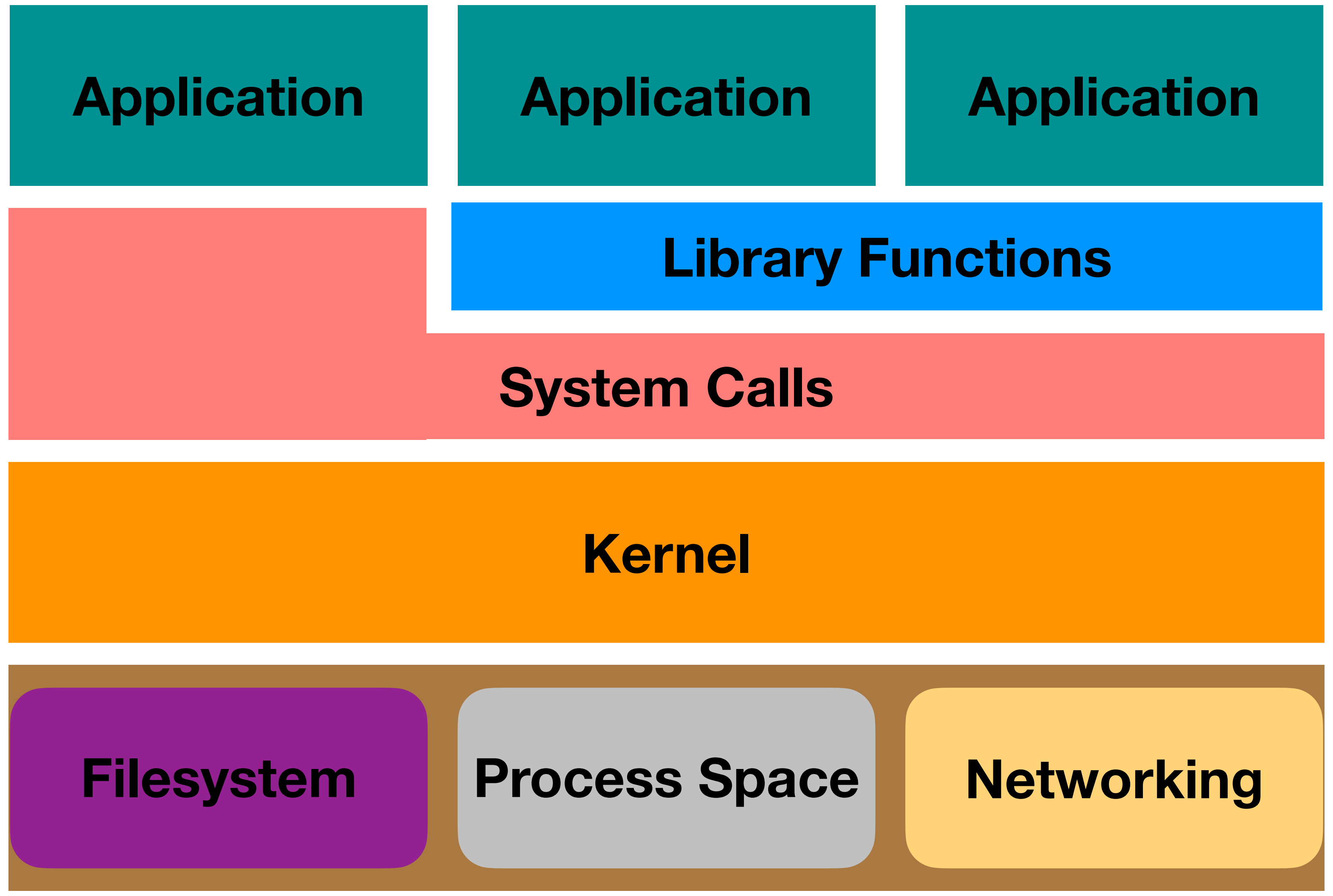
- use null and union mounts to provide the right environment
- restrict processes in their utilization
- restrict filesystem views
- restrict processes from what they can see
- restrict processes from what they can do

That is, the basis of many container technologies, such as CoreOS, LXC, or Docker, are `cgroups`, *namespaces*, and the application of all the various concepts discussed in this series.

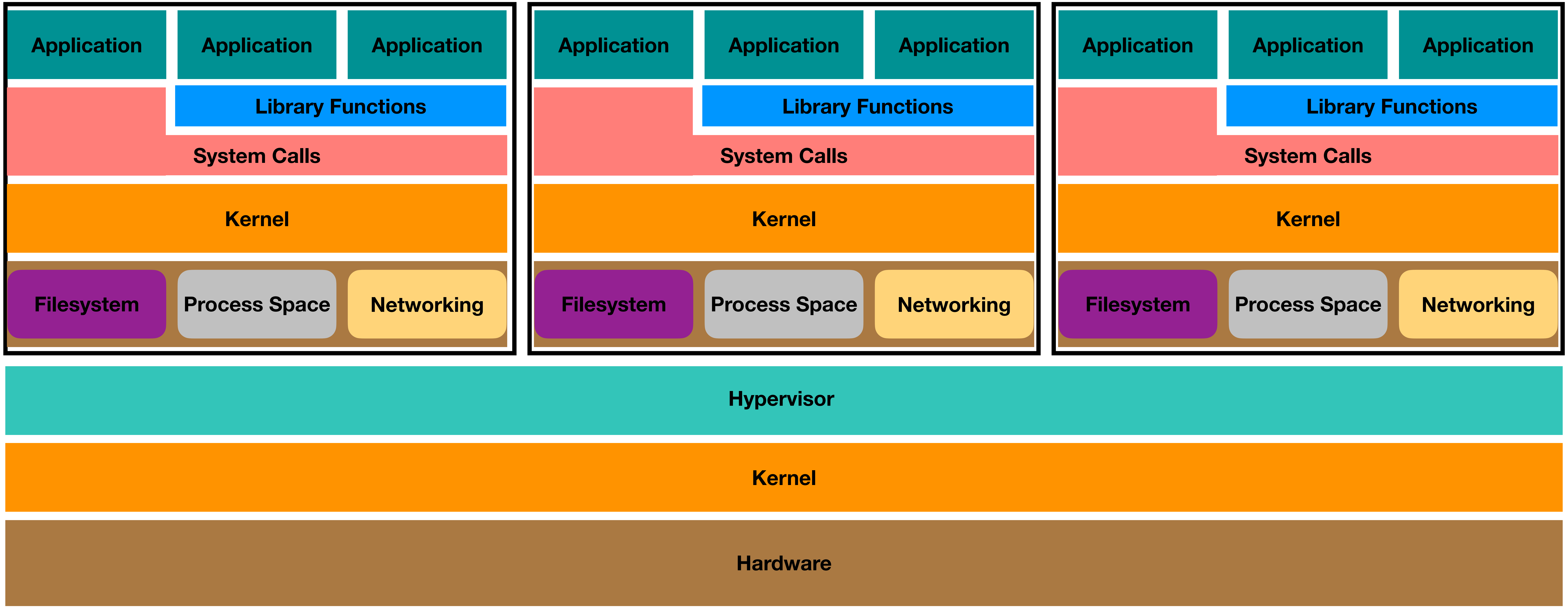
Basic OS


Application
Application
Application
Library Functions
System Calls
Kernel
Hardware

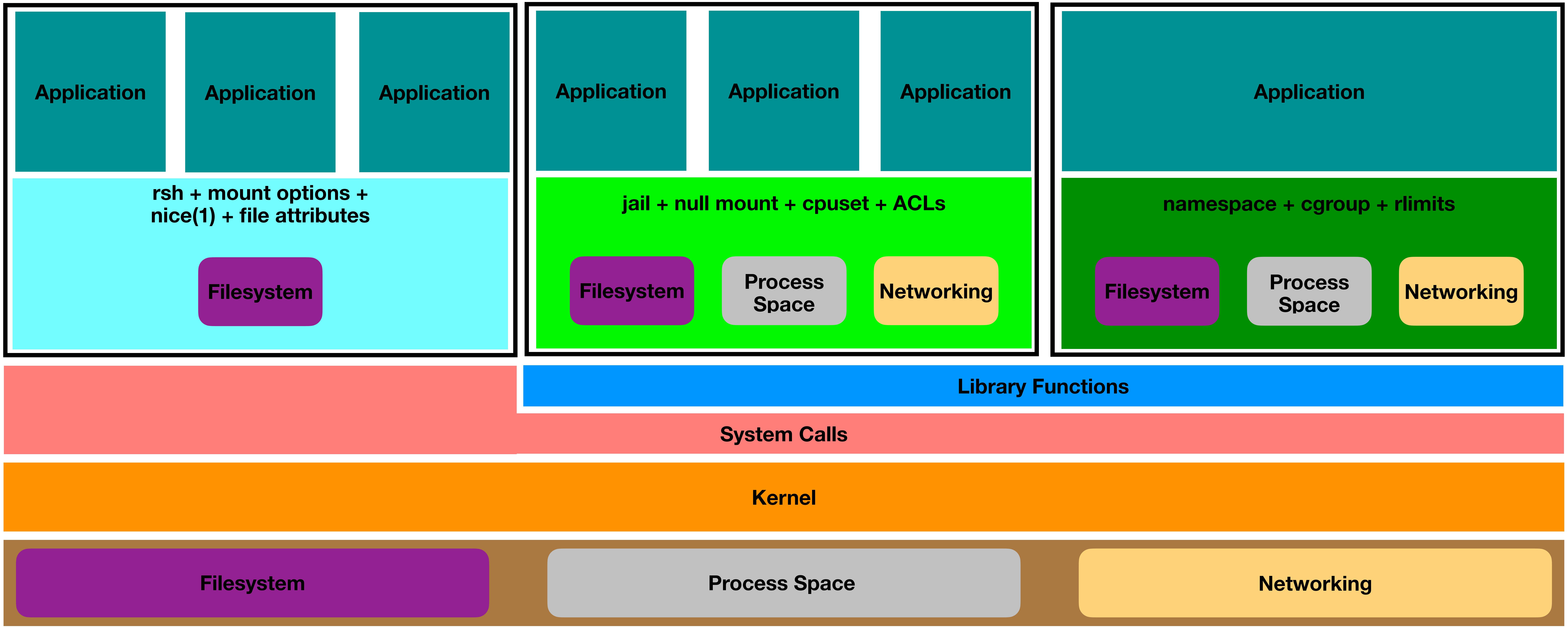
Basic OS


Application
Application
Application
Library Functions
System Calls
Kernel
Filesystem
Process Space
Networking

Virtualization



From restricted processes to containers



Additional Reading

- Capabilities: https://wiki.gentoo.org/wiki/Hardened/Overview_of_POSIX_capabilities
- NetBSD `kauth(9)`: https://man.netbsd.org/kauth.9
- macOS: https://developer.apple.com/library/archive/technotes/tn2127/_index.html
- Linux `capabilities(7)`: https://man7.org/linux/man-pages/man7/capabilities.7.html
- FreeBSD Capsicum: https://wiki.freebsd.org/Capsicum
- Linux Control Groups: https://www.kernel.org/doc/Documentation/cgroup-v2.txt
- Linux Namespaces: https://medium.com/@teddyking/linux-namespaces-850489d3ccf