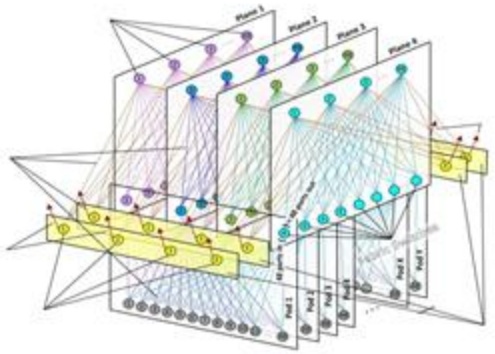
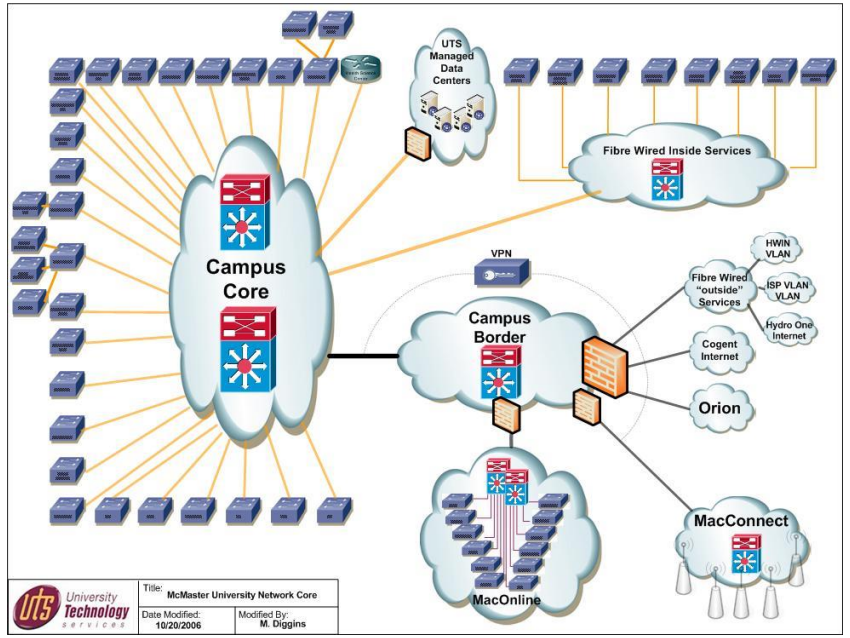


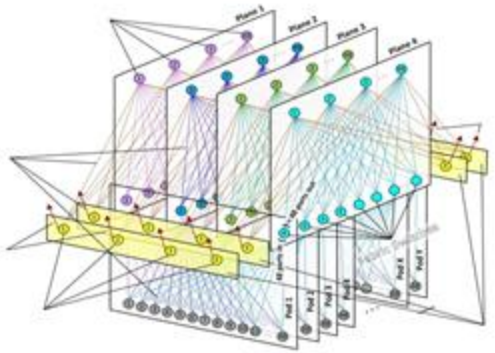
Large Systems:

Design + Implementation + Administration

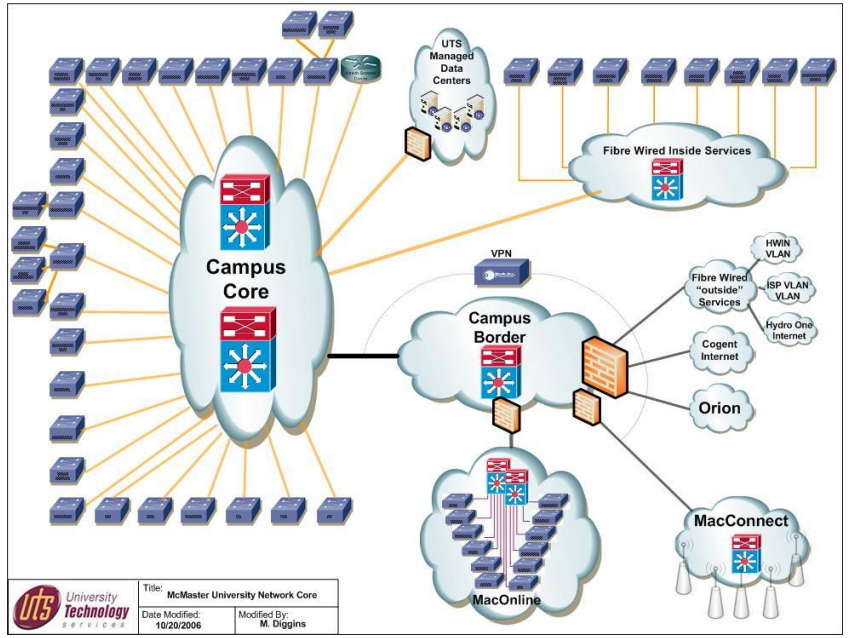


Large Systems:

Design + Implementation



2024-2025



- Week2-L4: Virtualization- Part 4

Shashikant Ilager
shashikantilager.com

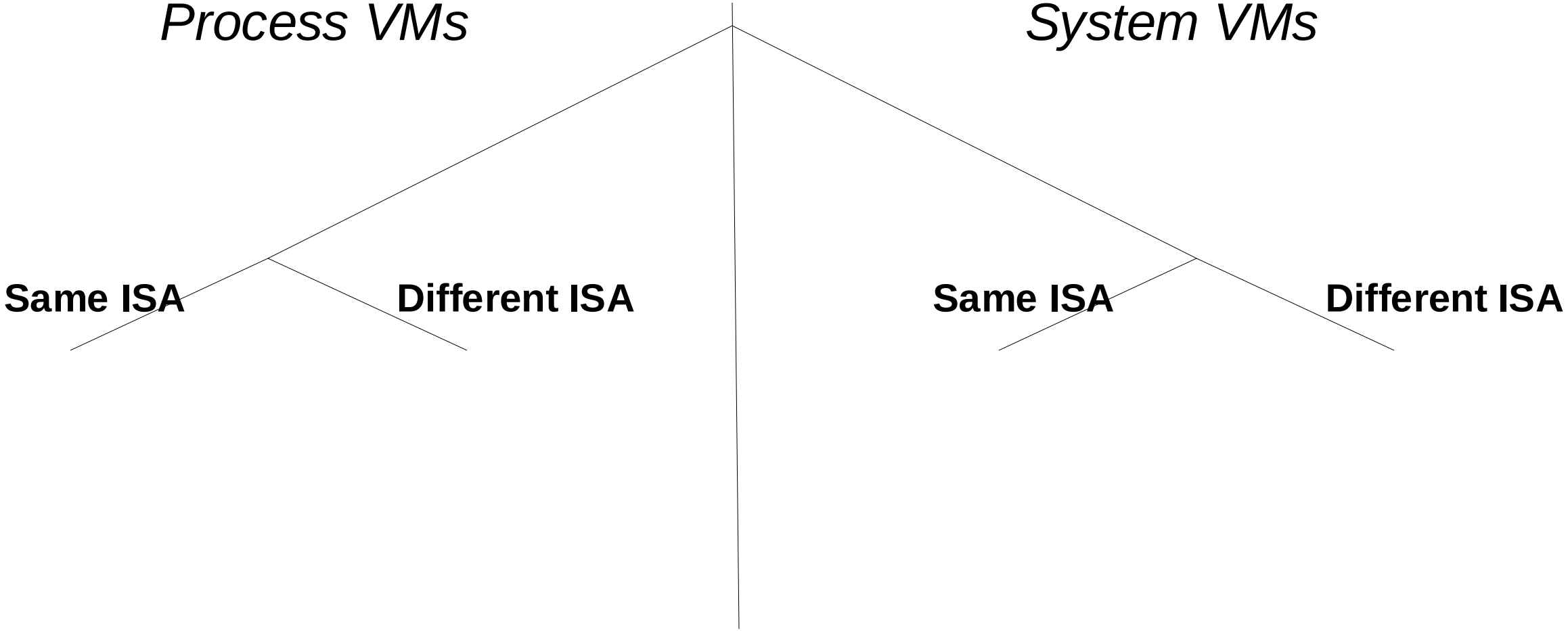
7 november 2024

Admin tasks....

- Presentation papers selection
  - Make sure you have 2 member group
  - Please send your papers by next Tuesday (Nov 12th)
  - The email should contain details of two members and at least two papers selected
  - Once approved, a link will be open to upload your paper on Canvas

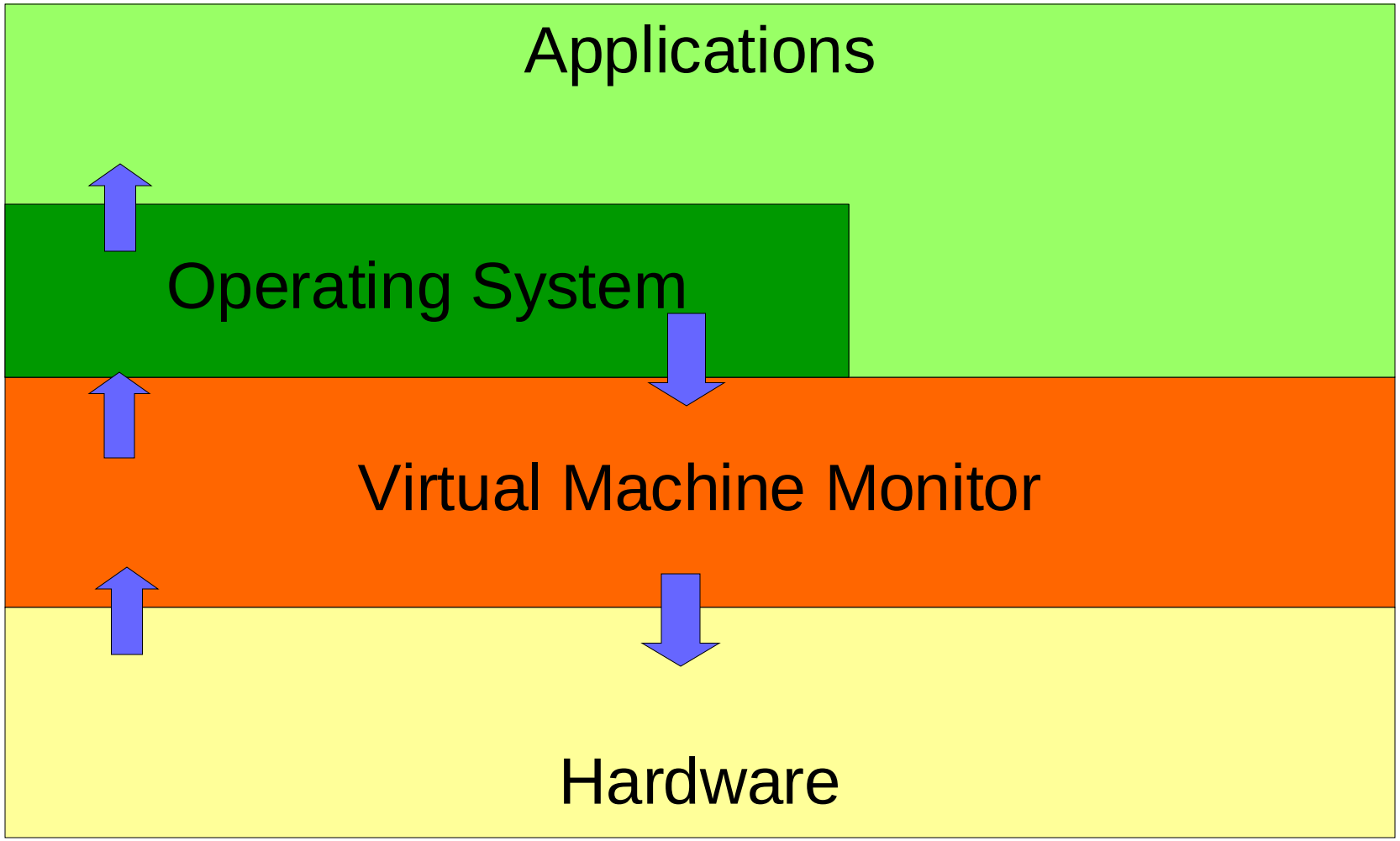
Email id: s.s.ilager@uva.nl

Recap: Taxonomy

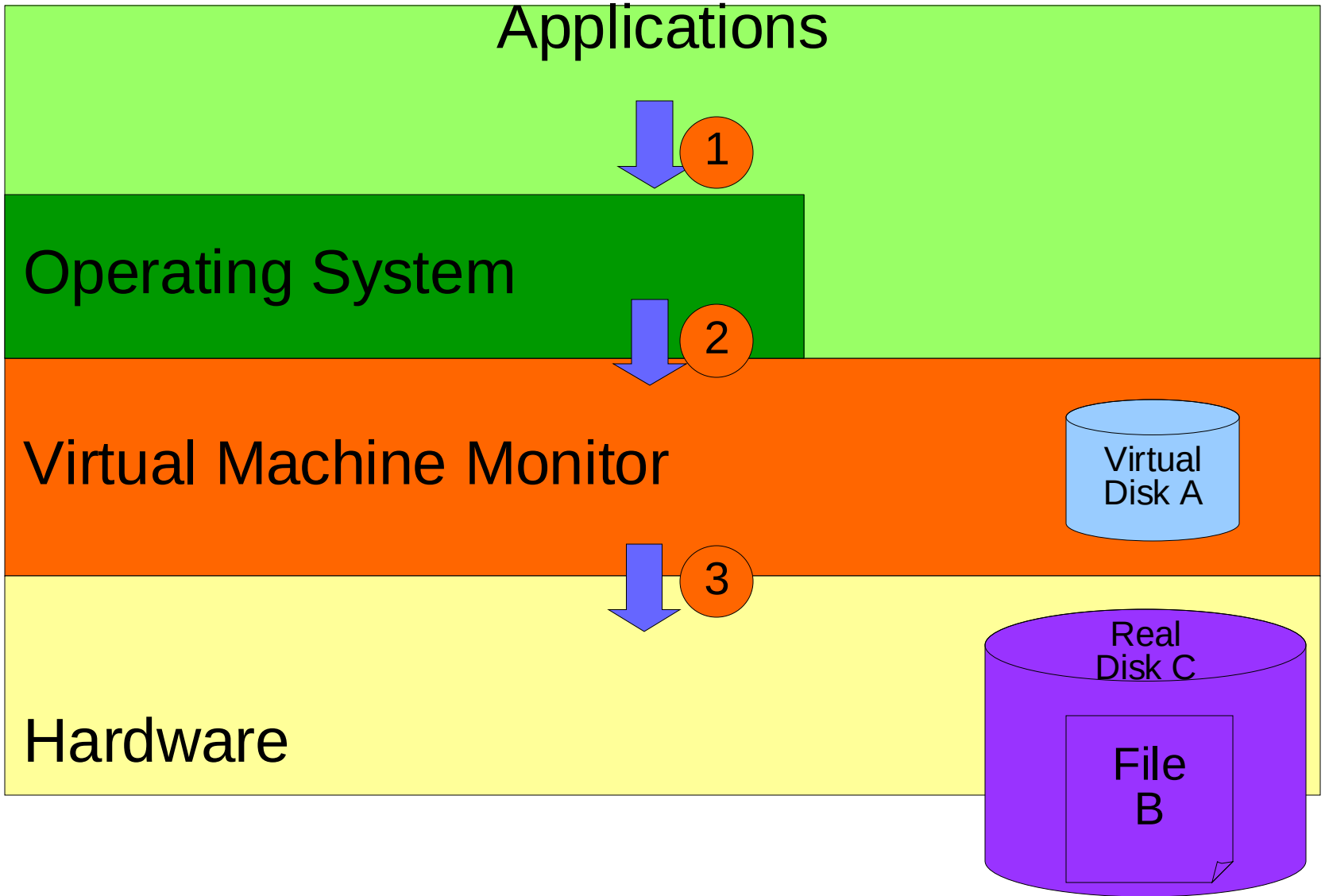


Java, Xen and Android emulation

Recap: App Scheduling



Recap: Example- Reading from disk



Recap: KVM

Poor Performance
Scope for Improvement
Optimal Performance

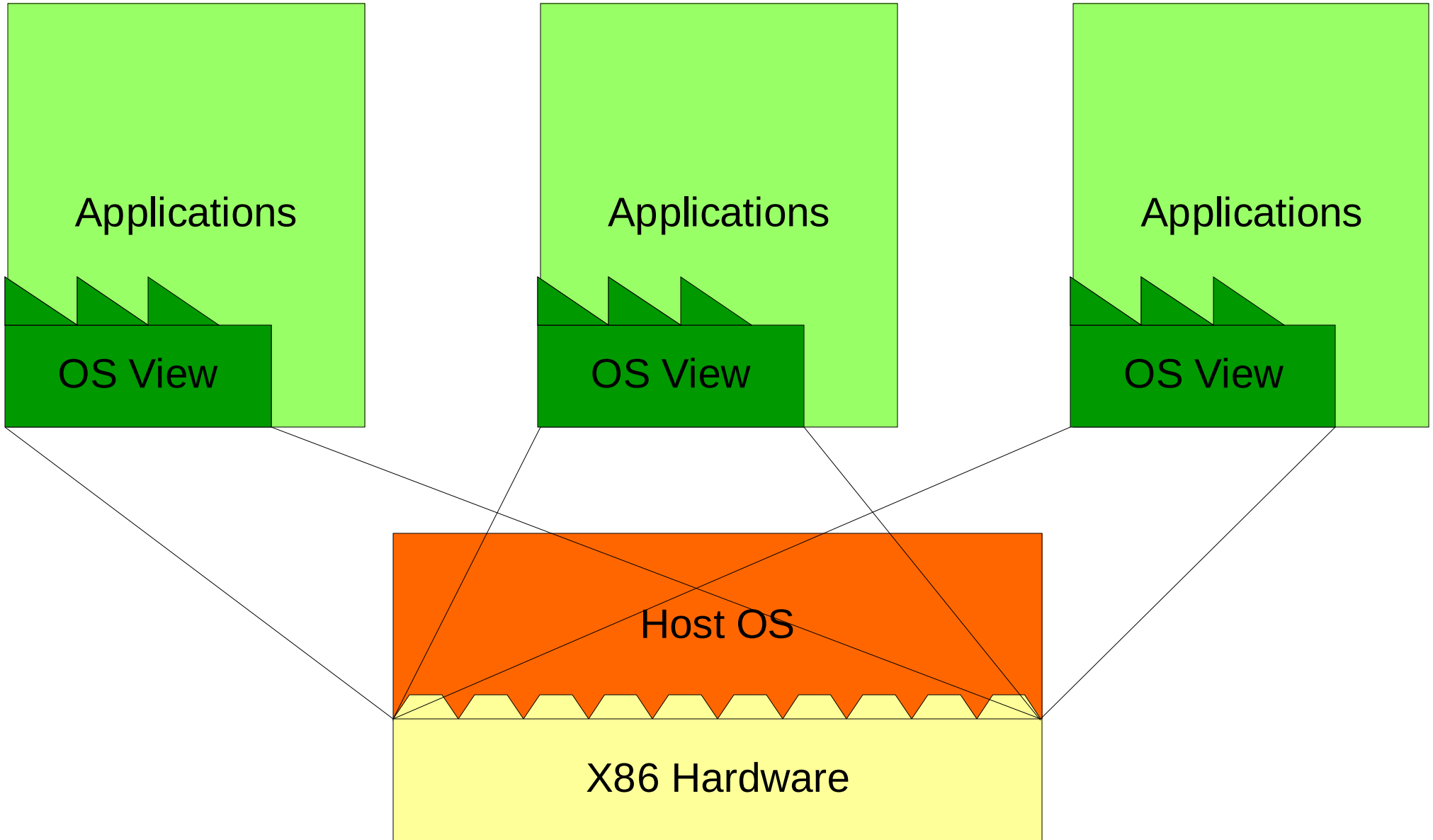
P = Paravirtualized
VS = Software Virtualized (QEMU)
VH = Hardware Virtualized

Shortcut	Mode	With	Disk and Network	Interrupts & Timers	Emulated Motherboard, Legacy Boot	Privileged Instructions, Page Tables
HVM / Fully Virtualized	HVM		VS	VS	VS	VH
HVM + PV drivers	HVM	PV Drivers	P	VS	VS	VH
KVM	HVM		P	VS	VS P	VH
PVHVM	HVM	PVHVM Drivers	P	P	VS	VH
PVH	PV	pvh=1	P	P	P	VH
PV	PV		P	P	P	P

Operating-System Level Virtualization

- In between System VM and Process VM
- Not System VM:
  - Cannot choose OS
- Not Process VM:
  - Multiple processes, not isolated
- As if multiple instances of the same OS are running on the same machine
  - Example: Linux Containers
  - cf. Docker

Linux Containers

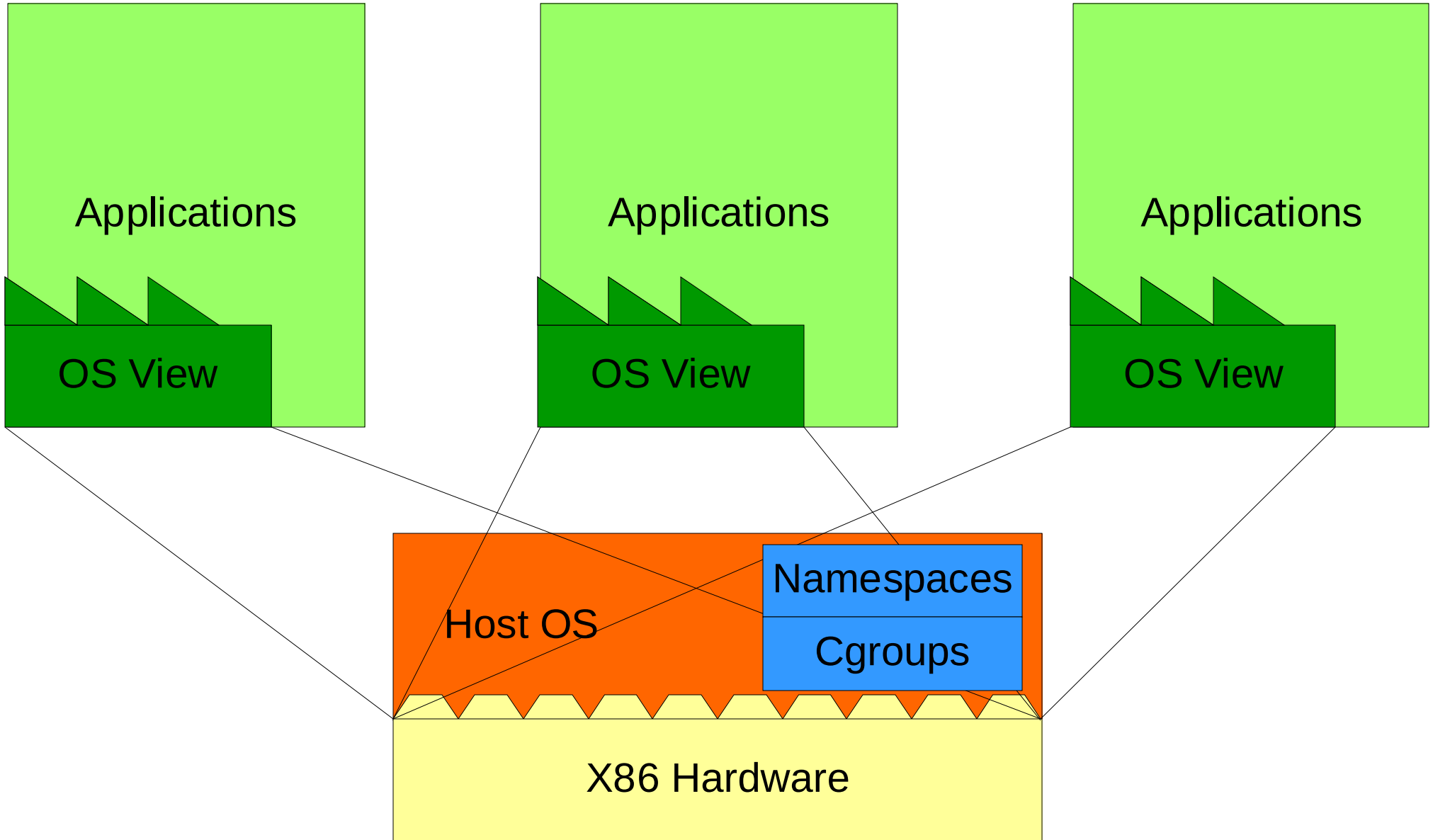


Containers

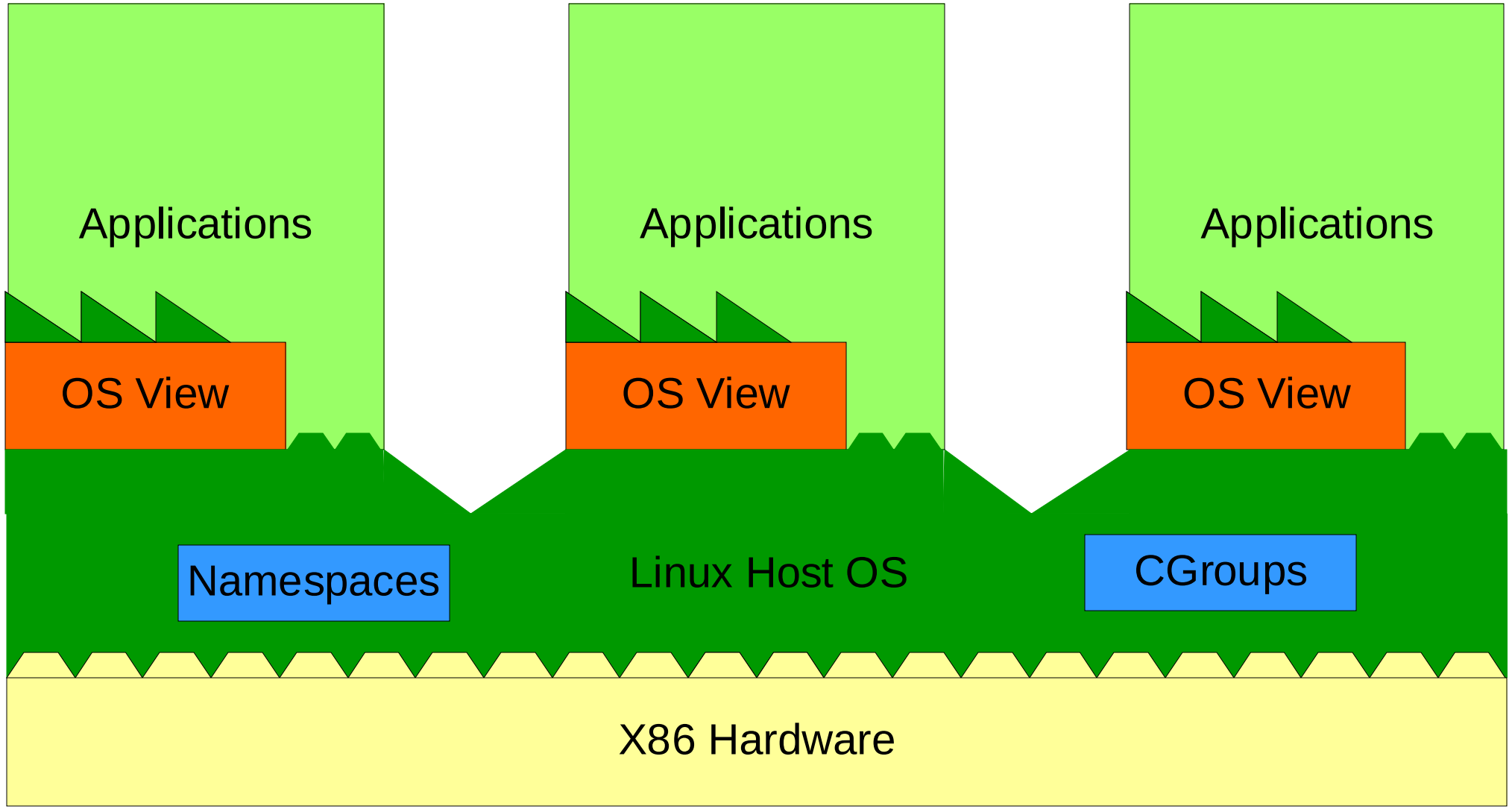
- Only one kernel is installed, and the hardware is not virtualized.
- Rather, the operating system is virtualized, providing processes within a container with the impression that they are the only processes on the system.
- One or more containers can be created, and each can have its own applications, network addresses, user accounts, and so on.

Source: Silberschatz et al. 9th US Edition.

Linux Containers



Linux Containers



Linux Containers: Namespaces

- Linux kernel has a configuration + state
- Controlled via many files and outside input
- Idea: allow a configuration per process (group)
- E.g. for process A the hostname is X, for process B the hostname is Y
- cf. old chroot system call: A sees different filesystem than B
- Now: configuration is set of 6 namespaces

- *Source: Rami Rosen, "Linux Kernel Networking", APress.*
- *http://www.haifux.org/lectures/299/netLec7.pdf*
- *Michael Kerrisk, https://lwn.net/Articles/531114/*

6 Namespaces of Linux

- uts (hostname)
- mnt (mount points, filesystems)
- pid (processes)
- user (UIDs)
- ipc (inter process communication IDs)
- net (network stack)

- (plans to add more)

UTS Namespace (1/3)

- “UNIX time sharing” ?!
- Contains 6 strings:
  - sysname Operating system name (e.g., "Linux")
  - nodename Name within "some implementation-defined network”
  - release OS release (e.g., "2.6.28")
  - version OS version
  - machine Hardware identifier
  - domainname NIS or YP domain name
- i.e., control the names of the container

Source: uname(2)

UTS Namespaces (2/3)

The **old** implementation of ***gethostname()***:

```
asmlinkage long sys_gethostname(char __user *name, int len)
{
    ...
    if (copy_to_user(name, system_utsname.nodename, I))
            errno = -EFAULT;
    ...
}
```

system_utsname is a global variable

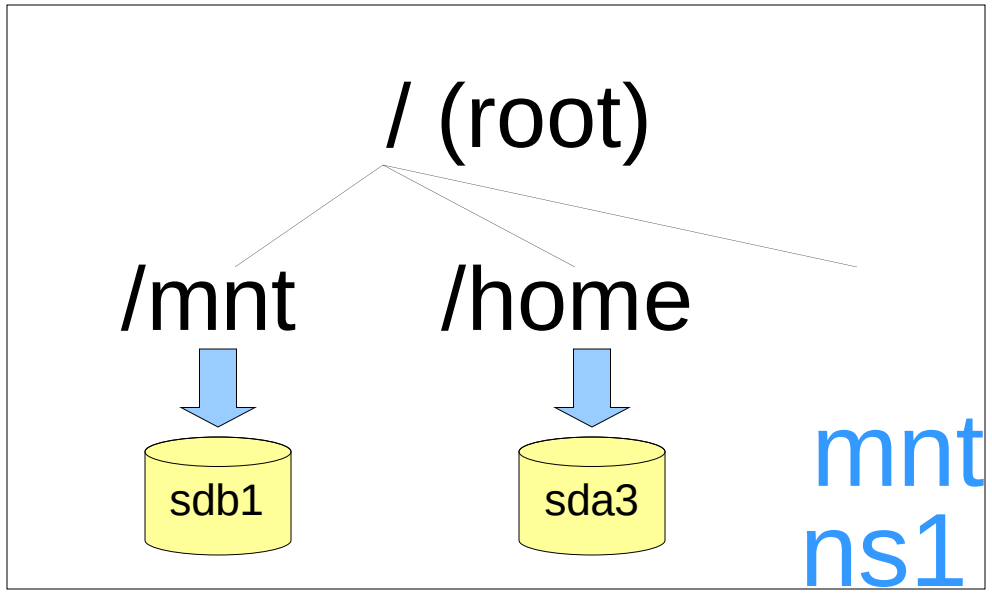
UTS Namespaces (3/3)

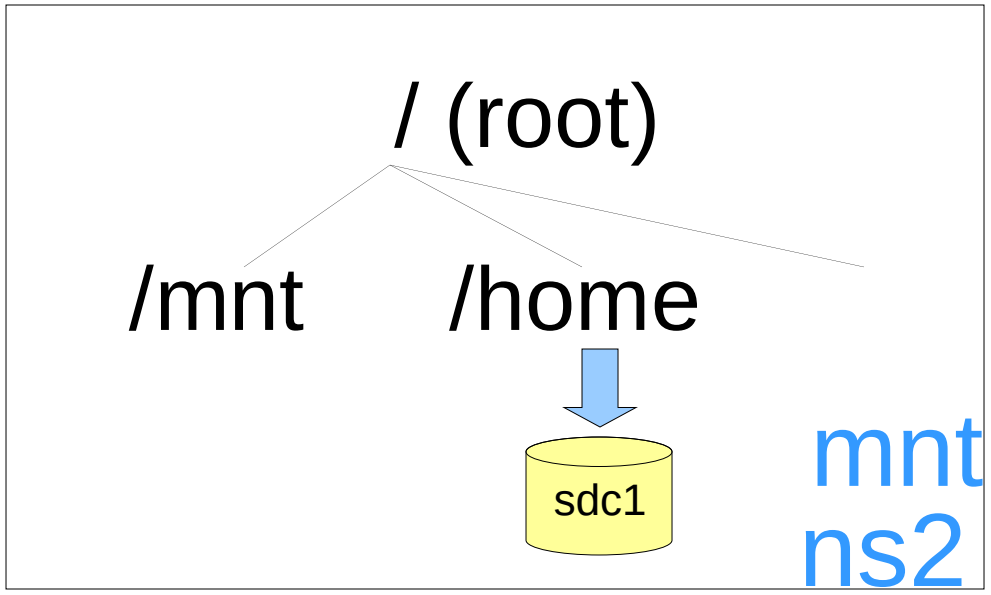
The **new** implementation of ***gethostname()***:

```
static inline struct new_utsname *utsname(void) {
        return &current->nsproxy->uts_ns->name;
}
SYSCALL_DEFINE2(gethostname, char __user *, name, int, len) {
        struct new_utsname *u;
        ...
        u = utsname();
        if (copy_to_user(name, u->nodename, i))
            errno = -EFAULT;
```

MNT Namespace

View of which filesystems are mounted





User namespace

- New namespace = new set of User IDs and Group IDs
  - E.g. user “ls24” has user ID 1023, the group “students2024”
  - Existing UIDs are mapped into new space
  - E.g. UID 1000 becomes UID 0 in new space
- First process in the new space has “root”
  - Only for namespaces inside the new space!
  - Outside: permissions of parent UID
- Gives the impression there are no other users

UID Namespace Example

Global UIDs

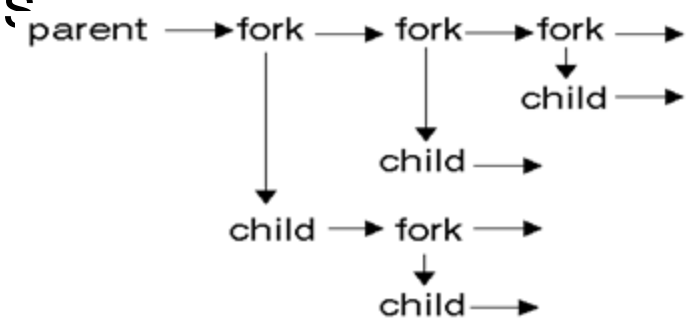
- 0 root
- 1 daemon
- 2 bin
- 1000 max
- 2000 root2
- 2001 daemon2
- 2002 bin2
- 2003 harry

UIDs inside Harry's container

- 0 root
- 1 daemon
- 2 bin
- 3 harry

PID Namespace

- Processes in different PID namespaces can have the same process ID.
- When creating the first process in a new namespace, its PID is 1.
- Can have hierarchy of PID namespace
    - parent can see inside child namespaces
- Gives the impression there are not other processes



PID Namespace Example

Global PIDs

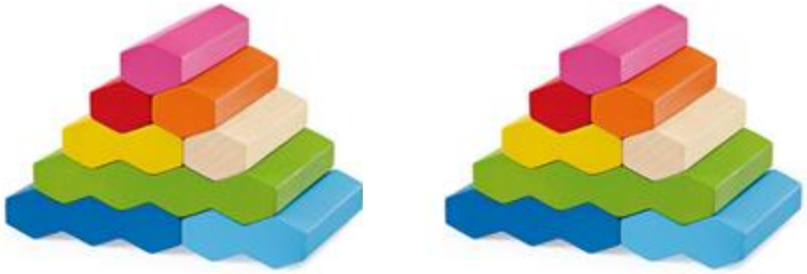
- 1 init
- 2 cron
- 300 httpd
- 1000 bash
- 2001 init
- 2002 cron
- 2003 bind
- 2004 42sh

PIDs inside Harry's container

- 1 init
- 2 cron
- 3 bind
- 4 42sh

NET Namespace

- A network namespace is logically another copy of the network stack:
  - own routes,
  - own firewall rules,
  - own network devices.
- A network device belongs to exactly 1 network namespace
- A socket belongs to exactly 1 network namespace
- Gives the impression of own network stack



NET Namespace (2/3)

- The initial network namespace includes:
  - loopback device
  - all physical devices,
  - networking tables, etc.
- New network namespace includes only the loopback device
  - Real devices can be moved into NS
  - Virtual devices can be added
  - ...

NET Namespace (3/3)

- Control via `ip netns` command
  - And e.g. `/etc/netns/<nsname>/hosts`

Containers via namespaces

Create a container:

1. Create a user namespace
2. Create a PID and UTS namespace inside
3. Create a MNT namespace to get your own filesystem
4. Mount container disk image
5. Create NET namespace, add virtual devices
6. Connect virtual devices to real network via e.g. virtual bridges
7. [Apply SELinux/AppArmor for extra security]

Docker

- Subset of Linux/Windows Containers
- Only 1 application per container
- Container is read-only
- Docker company and community offer disk images via repos
  - “Docker container”

CGroups

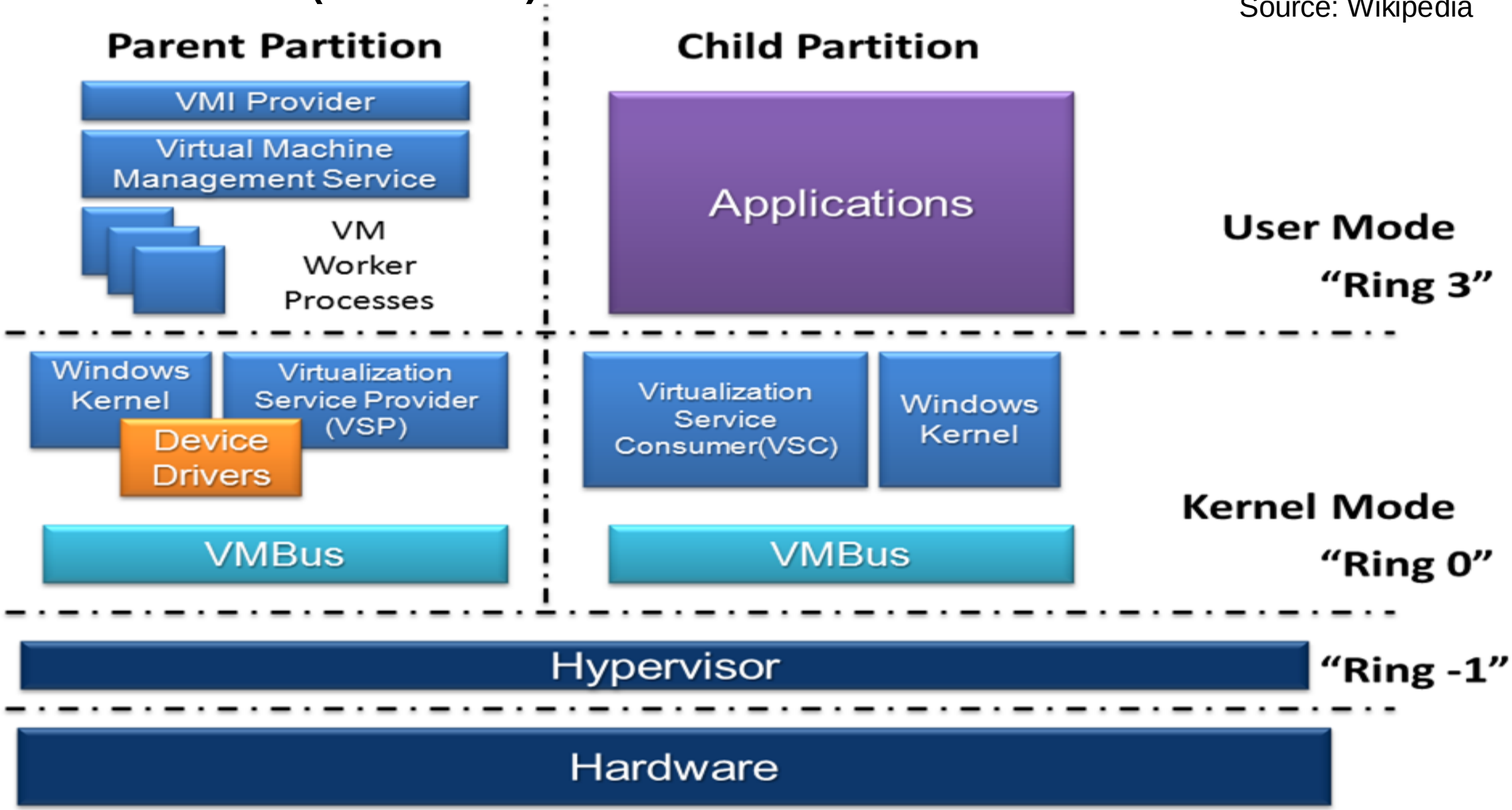
- Namespaces can give groups of processes:
  - Same view of the OS
  - Illusion there are no other groups
- Control Groups is a mechanism for resource management for groups of processes:
  - Set limits, e.g. on memory usage (main + FS cache)
  - Set priorities (CPU or disk bandwidth)
  - Accounting
  - Checkpointing
- E.g. stop set of processes for resuming later

Virtualization on Windows

- Hypervisor:
    - Hyper V
- Containers:
    - Windows Server Containers
    - Hyper V Containers
    - Docker integration

Windows (cont'd)

Source: Wikipedia

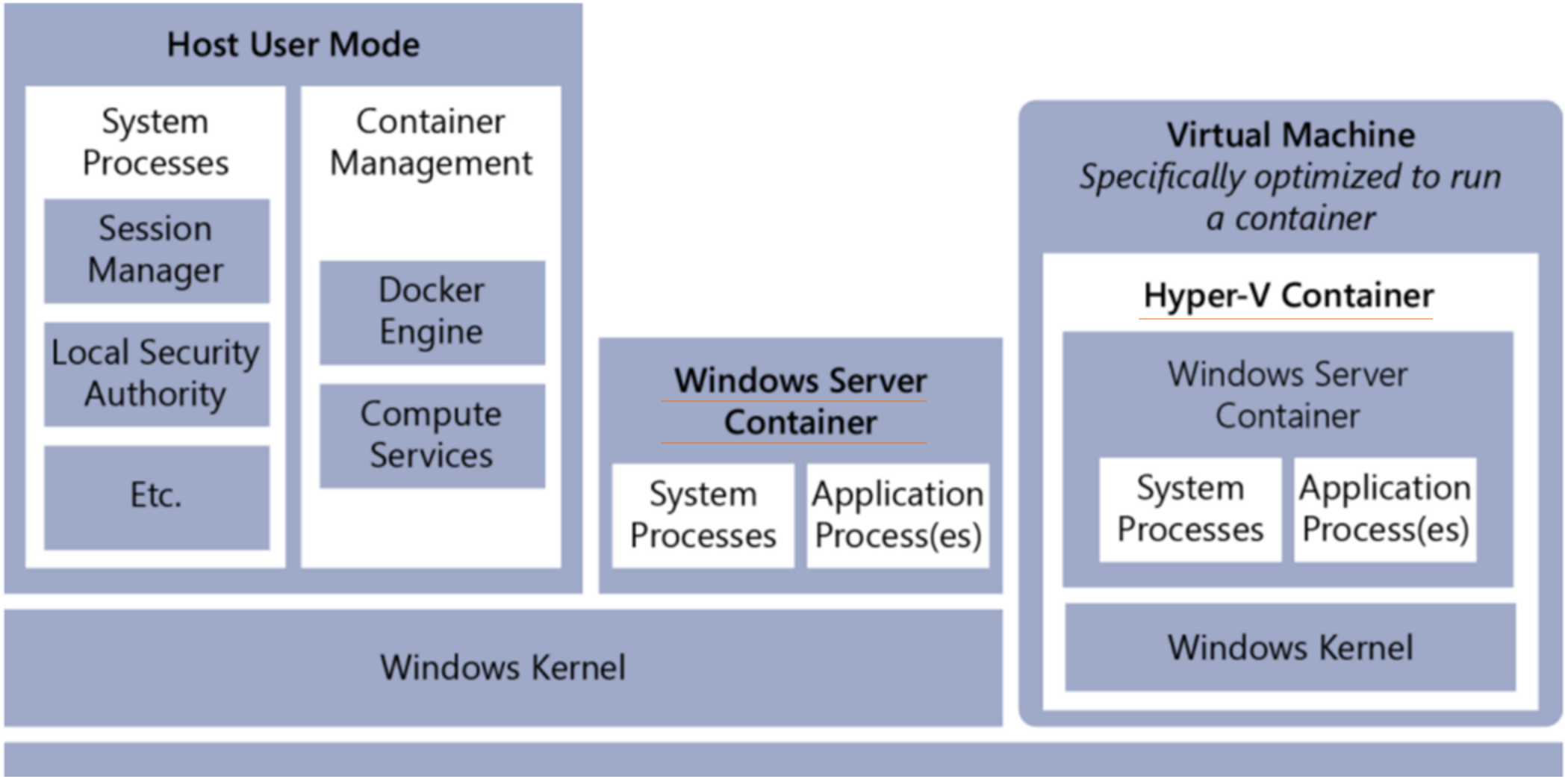


Windows (cont')

- Windows Server Containers:
  - provide application isolation through process and namespace isolation technology.
  - shares a kernel with the container host and all containers running on the host.
- Hyper-V Containers: like Server Containers but
  - expand on the isolation by running each container in a highly optimized virtual machine.
  - kernel of the container host is not shared with the Hyper-V Containers.

Source: https://msdn.microsoft.com/en-us/virtualization/windowscontainers/about/about_overview
https://azure.microsoft.com/en-us/blog/containers-docker-windows-and-trends/

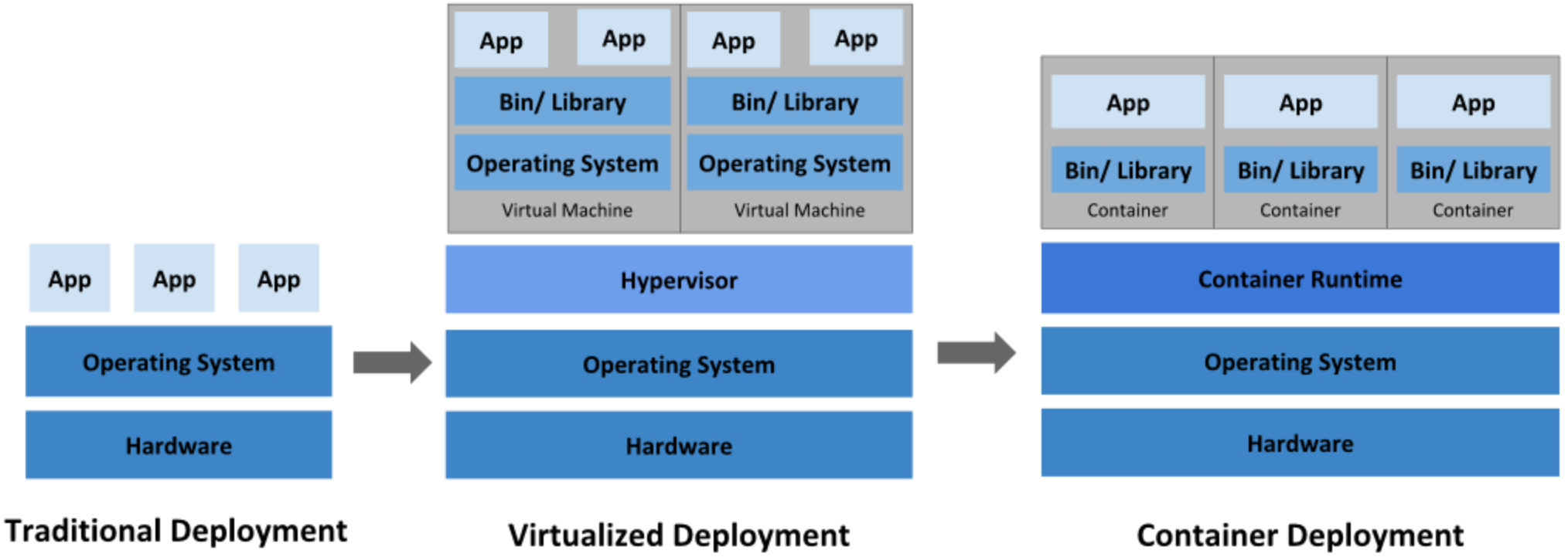
Windows Container Types



Source: McCabe, Friis - "Introduction to Windows Containers"

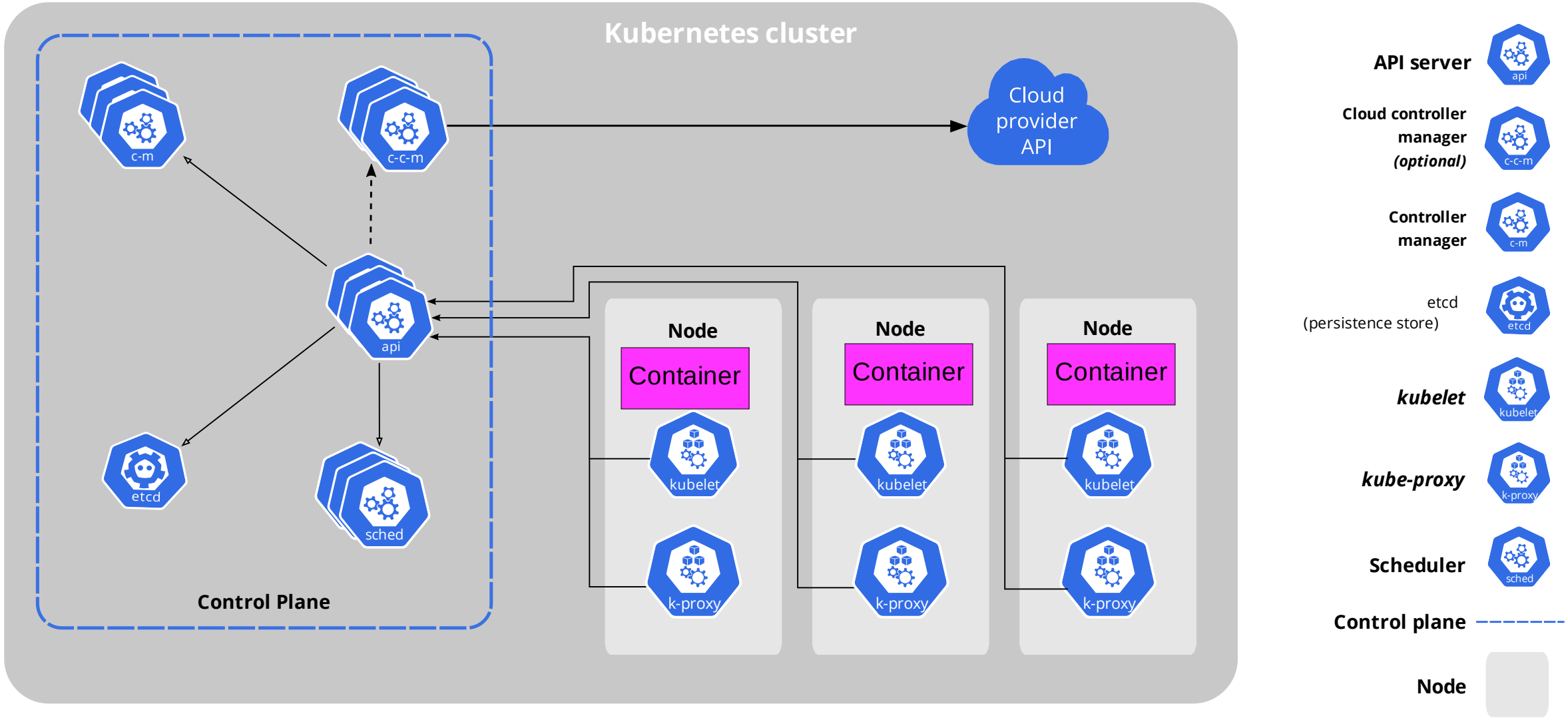
Kubernetes (k8s)

- "open-source system for automating deployment, scaling, and management of containerized applications."



Source: https://kubernetes.io/docs/home/

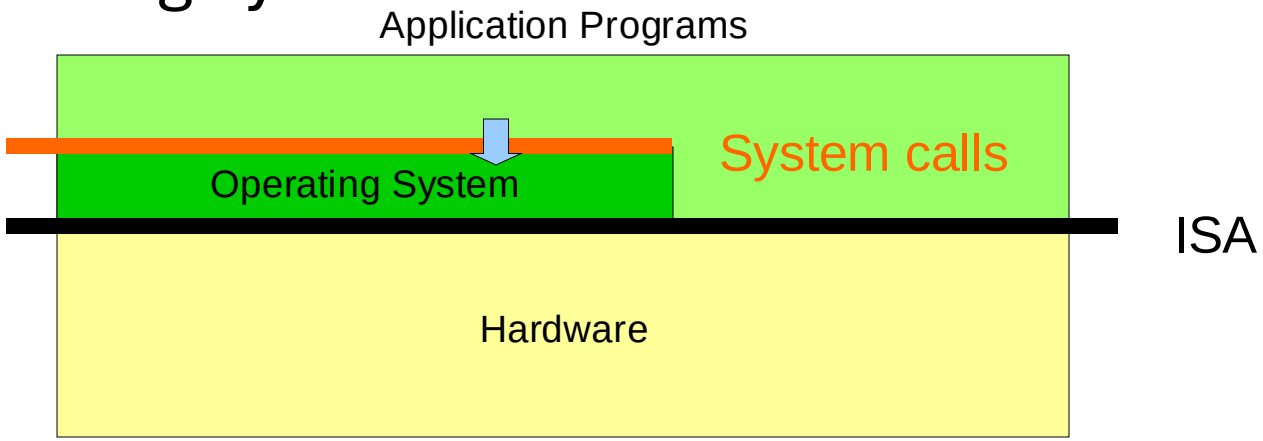
k8s Components



Source: https://kubernetes.io/docs/concepts/overview/components/

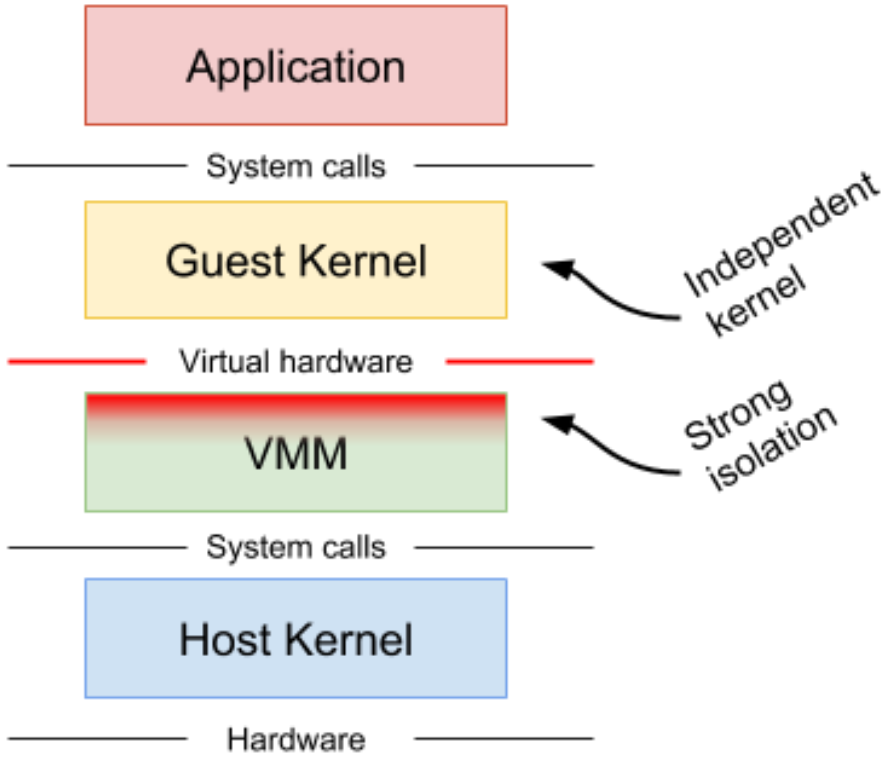
gVisor

- “gVisor is an application kernel, written in Go,
- implements a substantial portion of the Linux system call interface.
- It provides an additional layer of isolation between running applications and the host operating system.”

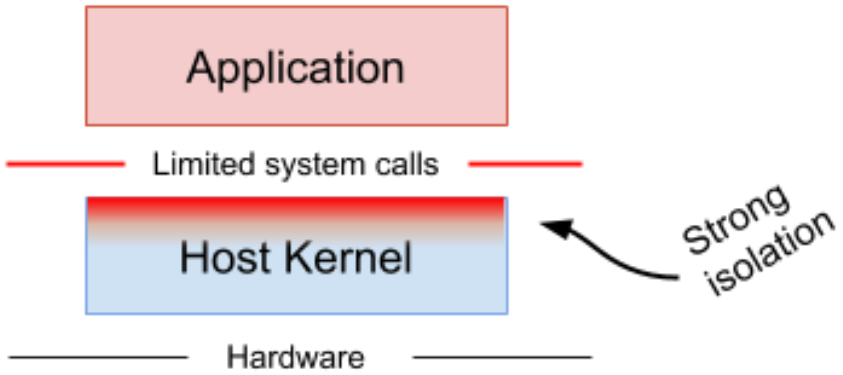


Source: https://gvisor.dev/docs/

gVisor



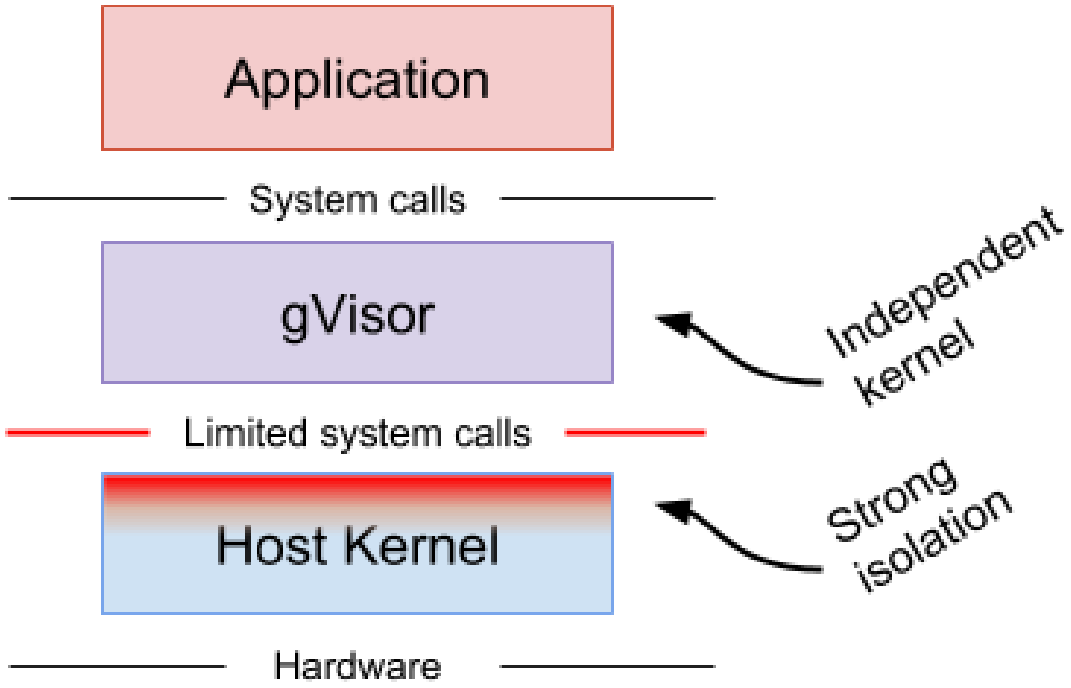
(a) Machine-level virtualization
e.g., KVM, Xen



(b) Rule-based execution,
e.g. SELinux / AppArmor

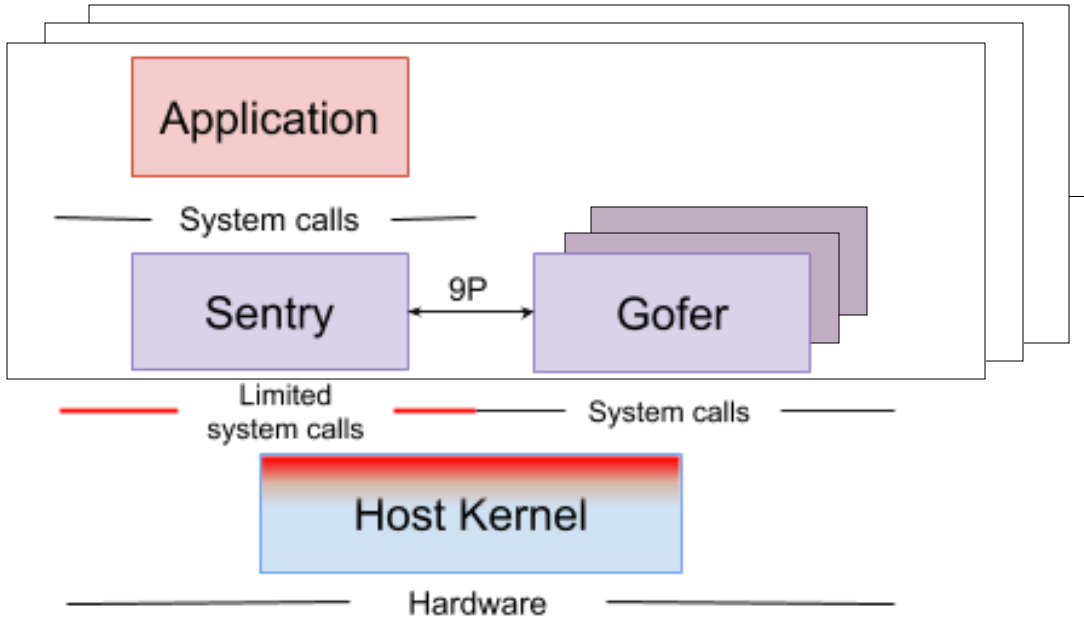
Source: https://gvisor.dev/docs/

gVisor



Source: https://gvisor.dev/docs/

gVisor Architecture



Sandbox containing one or more containers

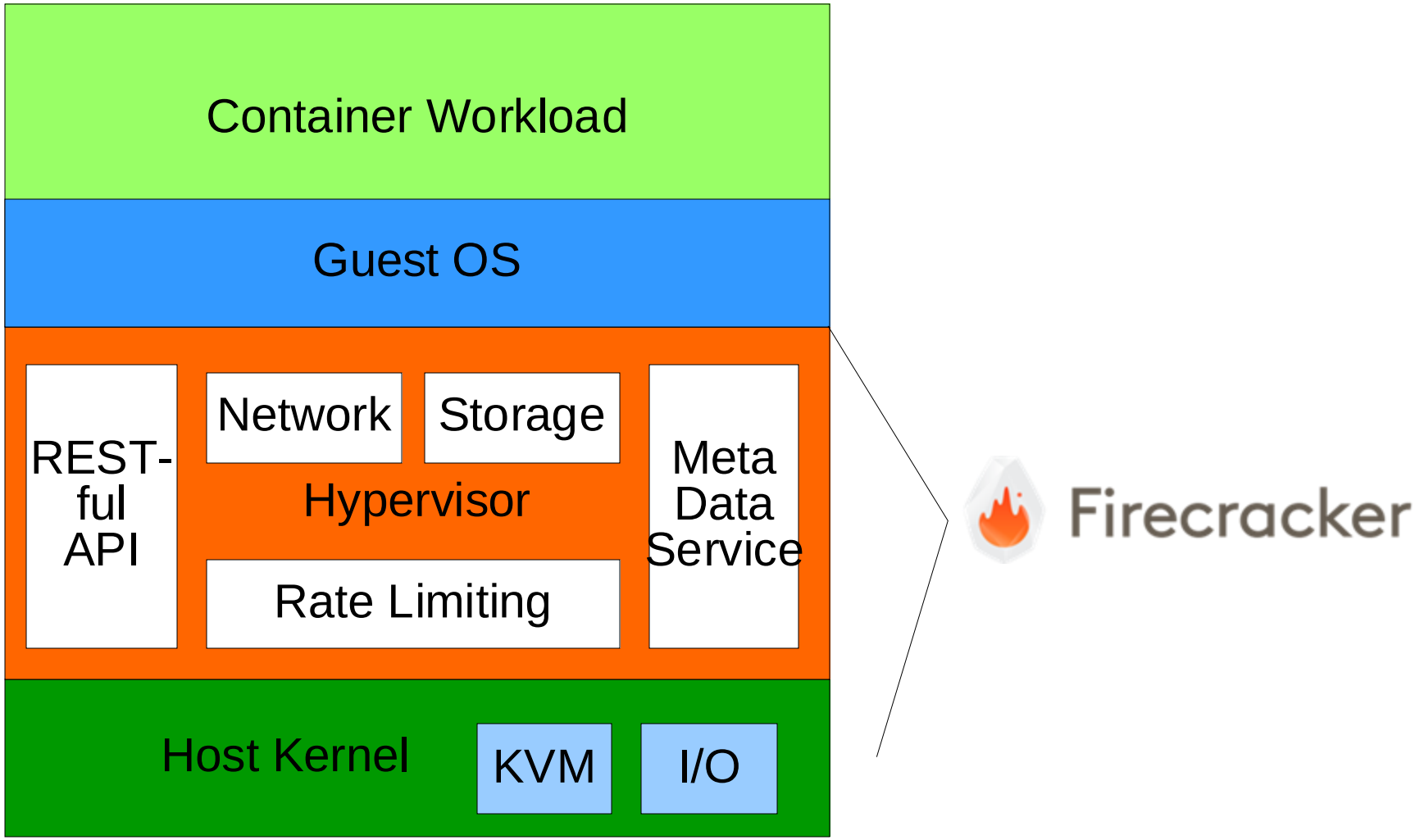
Source: https://gvisor.dev/docs/

Firecracker

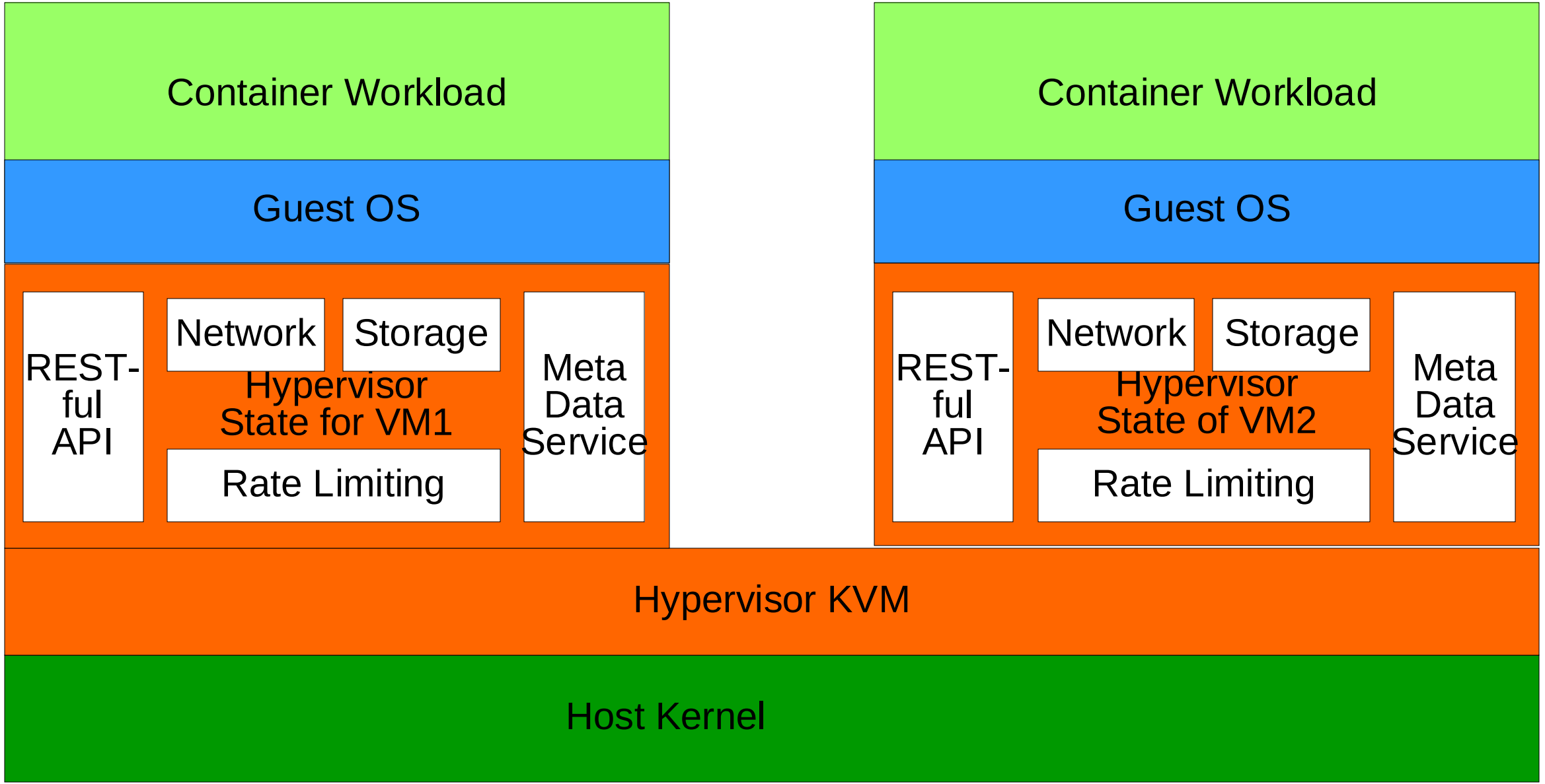
- “Firecracker is a Hypervisor that uses the Linux KVM to create and manage microVMs.”
- “excludes all non-essential functionality and reduces the attack surface area of the microVM.”
    - is an alternative to QEMU
- “accelerates kernel loading and provides a minimal guest kernel configuration. This enables fast startup times:
    - Starts application code “in as little as 125 ms”
    - Can create up-to “150 microVMs per second per host.”
- “microVM memory overhead of less than 5 MiB”
- “Firecracker can run Linux and OSv guests.”

Source: https://firecracker-microvm.github.io/

Firecracker Architecture



Firecracker Architecture



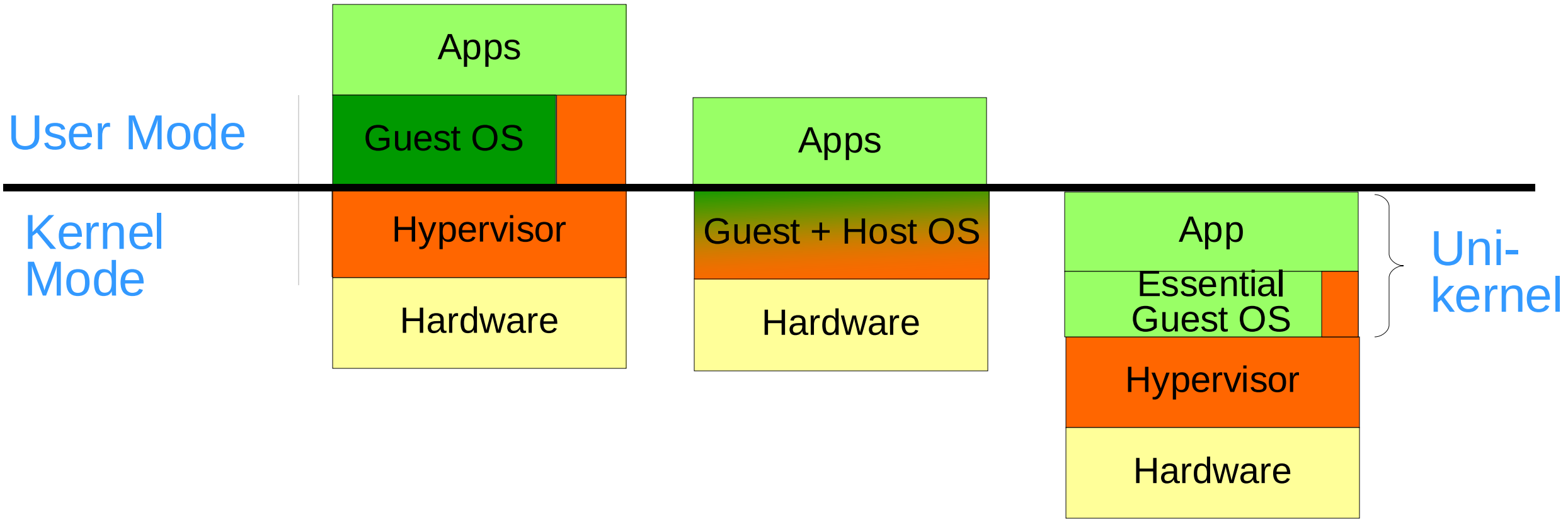
VMs without Operating Systems

Unikernels

- Optimize VM for one application:
- Strip unused parts of OS and libraries
- Link directly with application
- Advantages:
    - No context switching: No User-to-Kernel and vv.
    - Less memory usage
    - Apply application-specific OS optimizations
    - More secure?

More info see: http://book.rumpkernel.org/

Unikernels



(a) Paravirtual-ization (b) Containers (c) Unikernels

Video

- *"The Next Generation Cloud: Unleashing the Power of the Unikernel"*
- Russell Pavlicek, Xen Project Evangelist
- Large Installation System Administration Conference (LISA) 2015
- *https://www.usenix.org/conference/lisa15/conference-program/presentation/pavlicek*

- *(Thanks to Niels)*