

Advanced Cloud Computing

Container Virtualization

Wei Wang
CSE@HKUST
Spring 2025



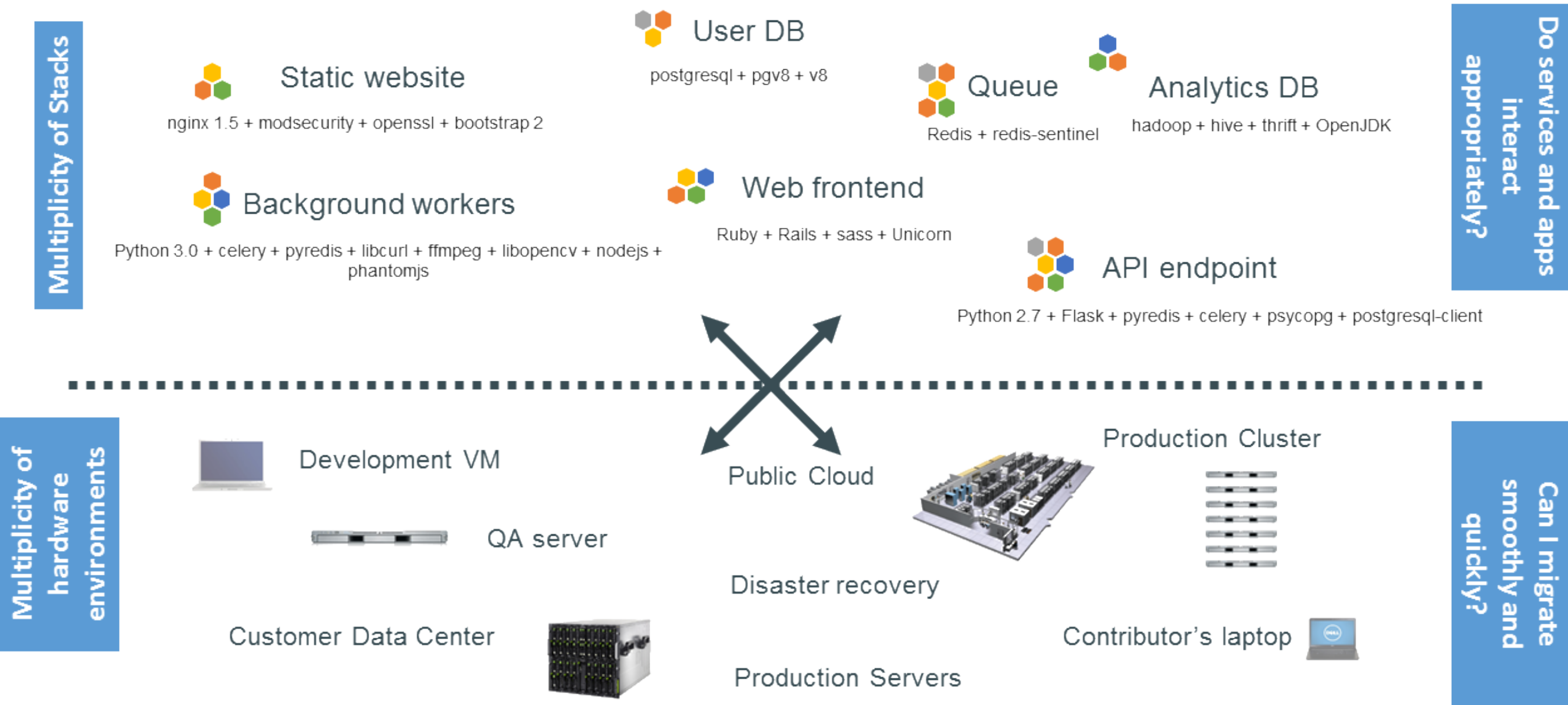
THE DEPARTMENT OF

COMPUTER SCIENCE & ENGINEERING

計算機科學及工程學系

Why container?

The challenge

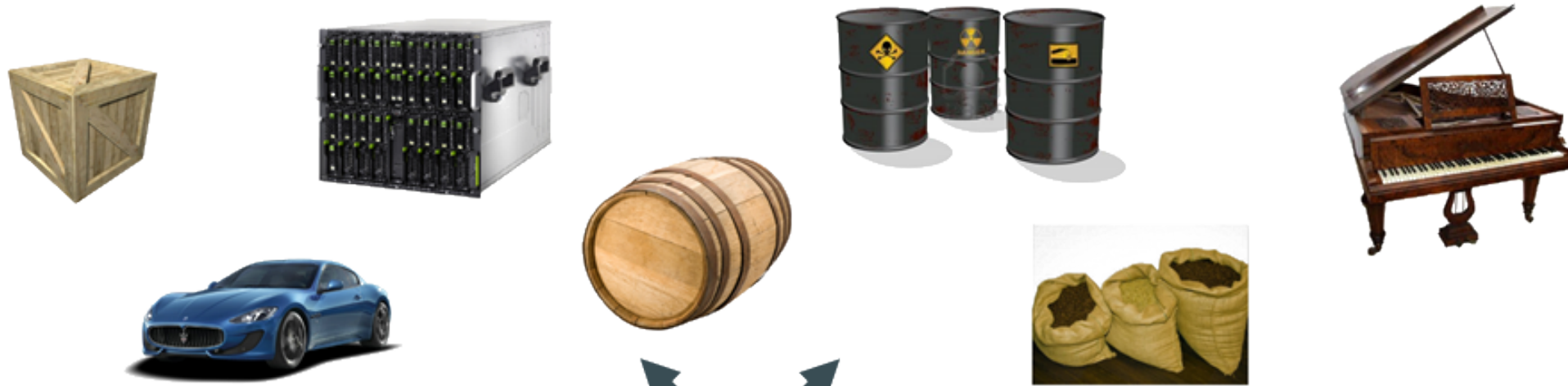


The Matrix from hell

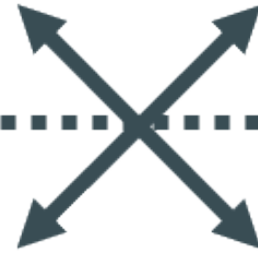
	Static website	?	?	?	?	?	?	?
	Web frontend	?	?	?	?	?	?	?
	Background workers	?	?	?	?	?	?	?
	User DB	?	?	?	?	?	?	?
	Analytics DB	?	?	?	?	?	?	?
	Queue	?	?	?	?	?	?	?
		Development VM	QA Server	Single Prod Server	Onsite Cluster	Public Cloud	Contributor's laptop	Customer Servers
								

Cargo transport pre-1960

Multiplicity of Goods



Do I worry about
how goods interact
(e.g. coffee beans
next to spices)















Multiplicity of
methods for
transporting/storing



Can I transport quickly
and smoothly
(e.g. from boat to train
to truck)

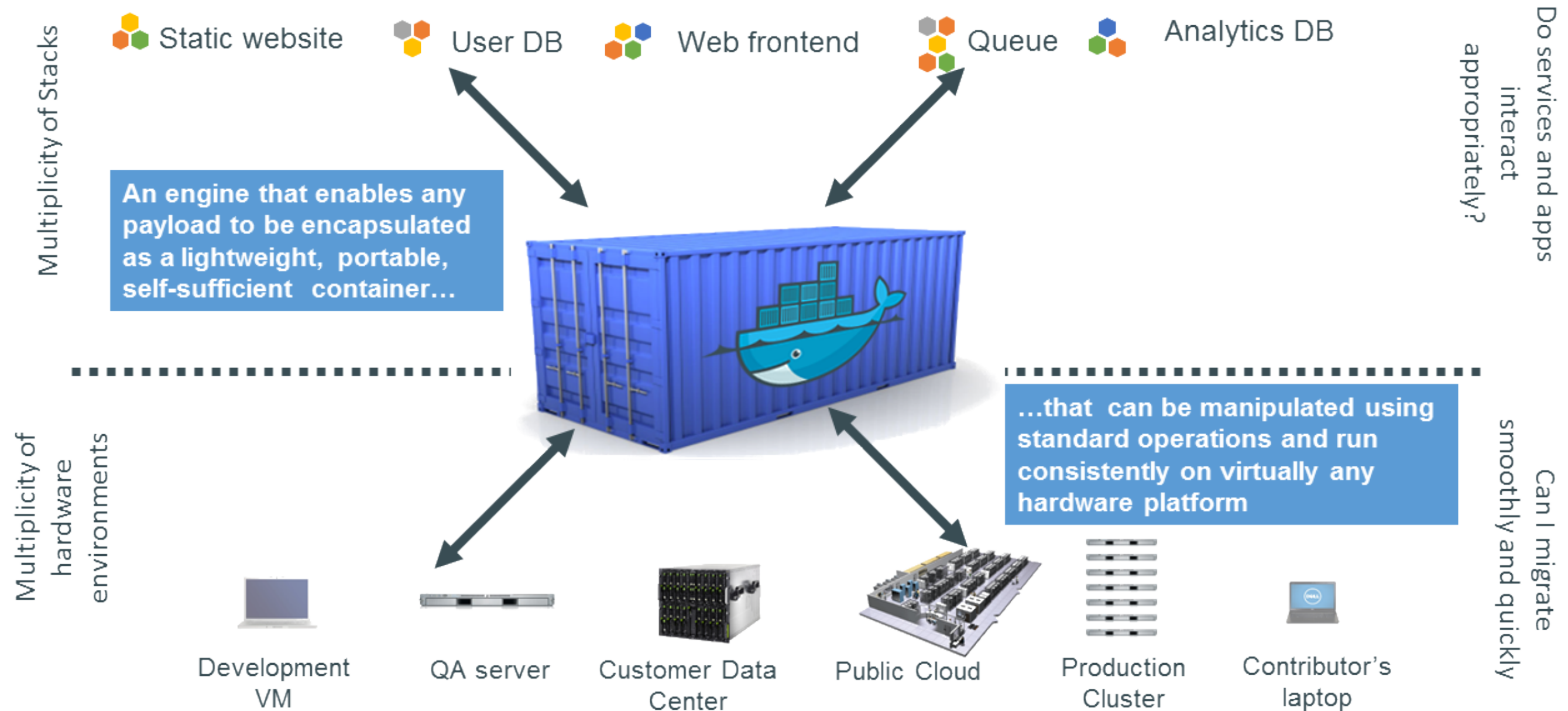
Also a matrix from hell

	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
							

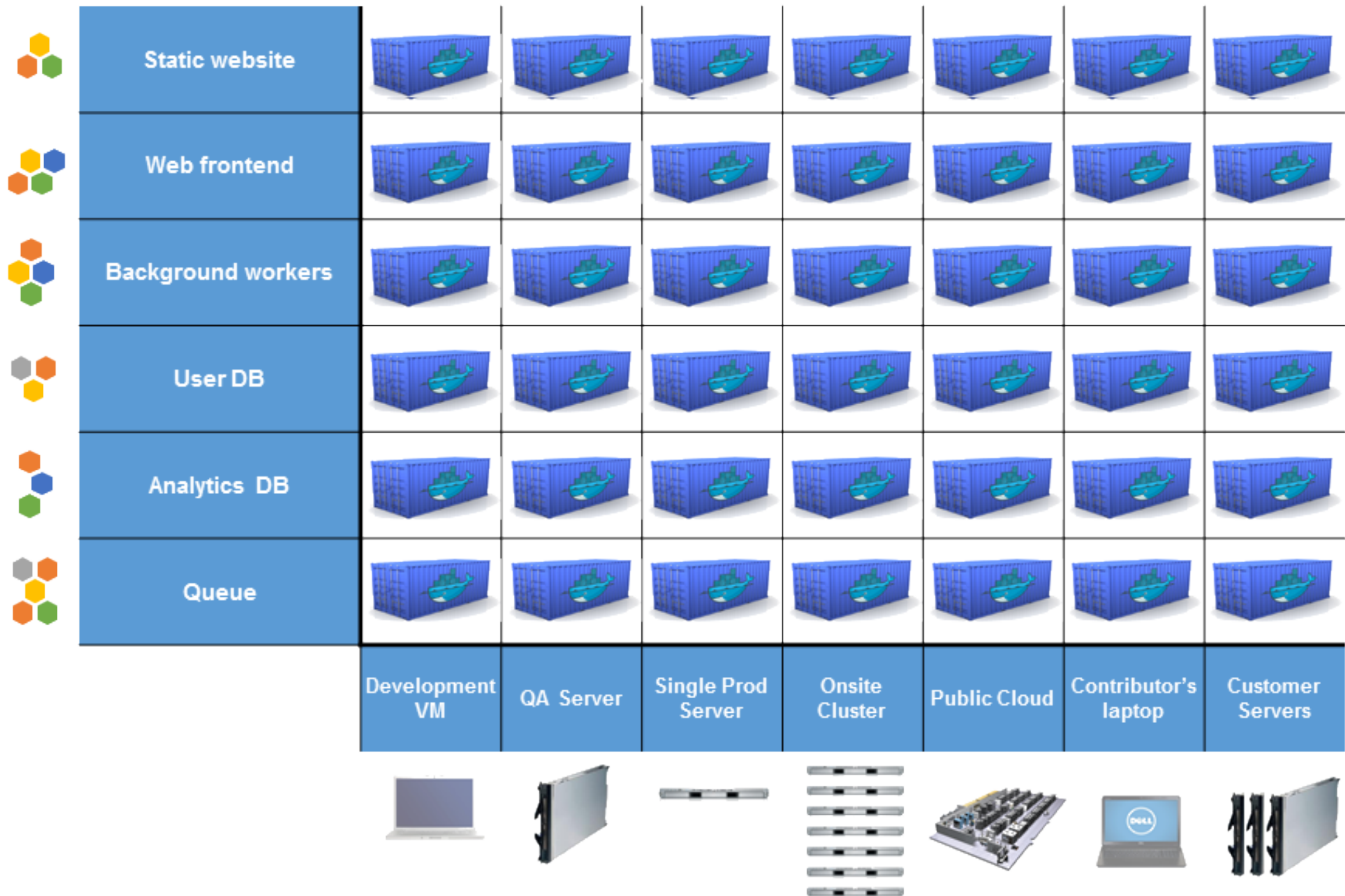
Intermodal shipping container



A container system for code



Eliminates the matrix from hell

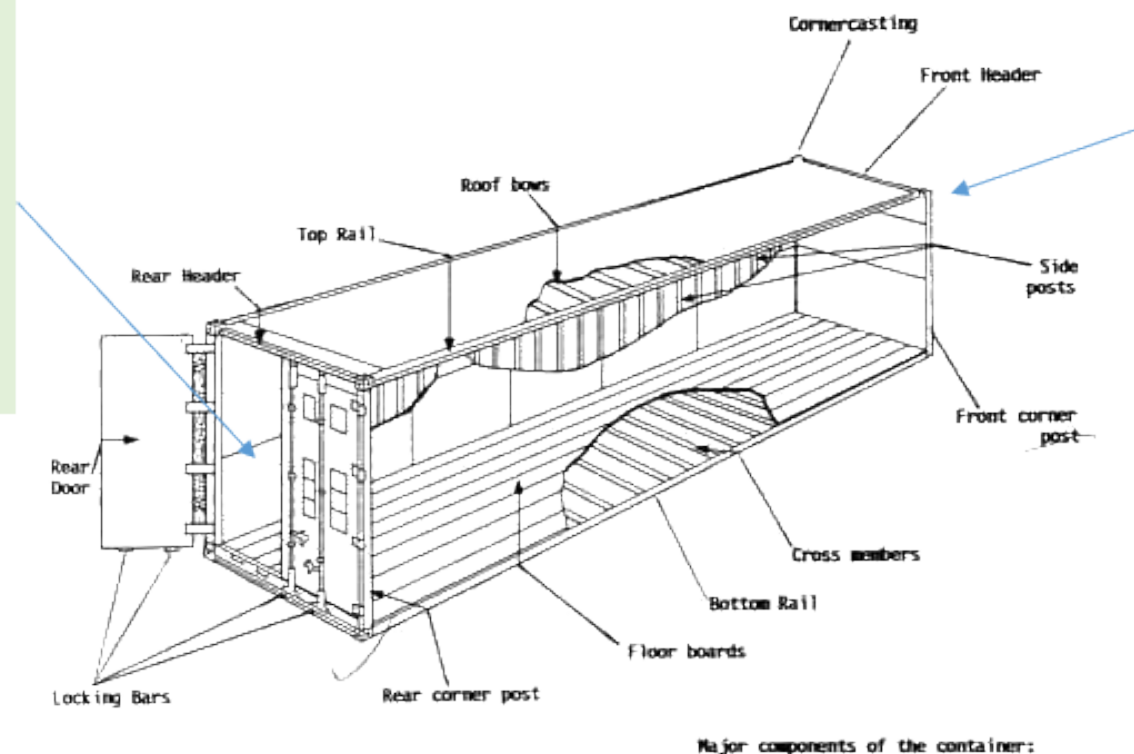


Configure once, run anything

Separation of concerns

- **Dan the Developer**

- Worries about what's "inside" the container
 - His code
 - His Libraries
 - His Package Manager
 - His Apps
 - His Data
- All Linux servers look the same

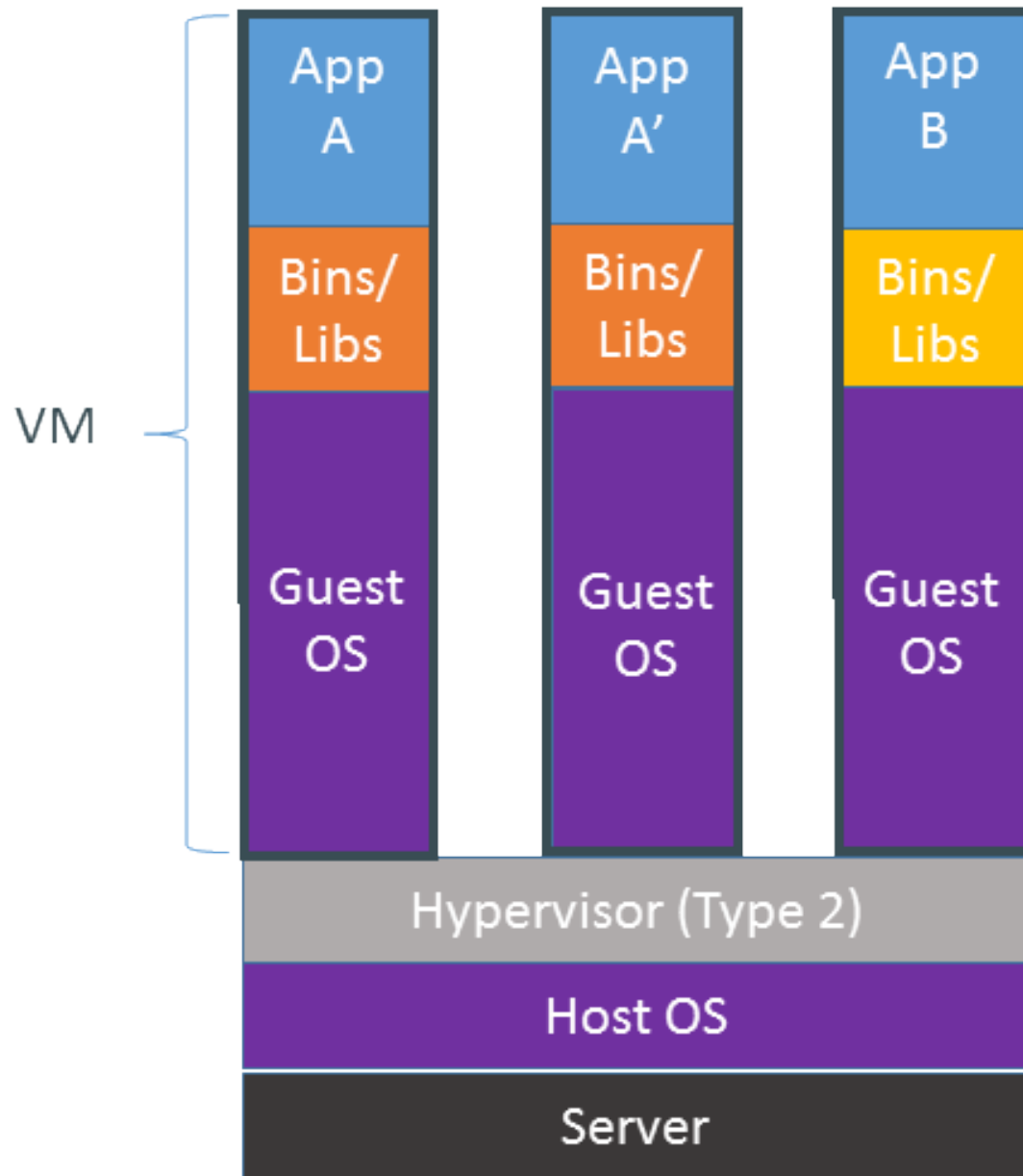


- **Oscar the Ops Guy**

- Worries about what's "outside" the container
 - Logging
 - Remote access
 - Monitoring
 - Network config
- All containers start, stop, copy, attach, migrate, etc. the same way

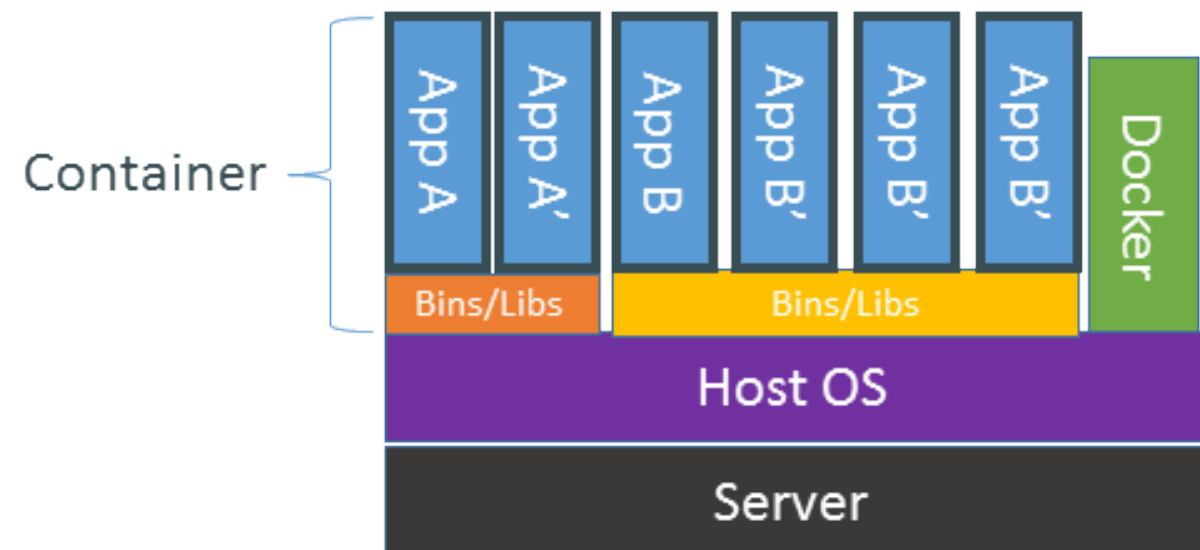
Configure once, run anything anywhere

VM vs. Containers



Containers are isolated, but share OS and, where appropriate, bins/libraries

...result is significantly faster deployment, much less overhead, easier migration, faster restart



VM vs. Container

VM system call path

- ▶ application inside the VM makes a system call
- ▶ trap to the hypervisor (or host OS)
- ▶ hand trap back to the guest OS

Container virtualization system call path

- ▶ application inside the container makes a system call
- ▶ trap to the OS
- ▶ OS returns the results to application

No binary translation,
no emulation

More technical details

High level: a lightweight “VM”

- ▶ own process space
- ▶ own network interface
- ▶ can run stuffs as root
- ▶ can have its own /sbin/init (different from host)
- ▶ <<machine container>>

Low level: chroot on steroids

- ▶ Container = isolated process: <<application container>>

Container implementation

Leveraging Linux kernel mechanisms

- ▶ **namespaces**: per process resource isolation
- ▶ **cgroups**: manage resources for groups of processes
- ▶ seccomp: limit available system calls
- ▶ capabilities: limit available privileges
- ▶ CRIU: checkpoint/restore (w/ kernel support)

What names must be virtualized?

Process IDs

- ▶ **top** inside the container shows only processes running inside it
- ▶ **top** outside the container may show processes inside the container, but with different process IDs

File names

- ▶ processes inside the container may have a limited different view of the mounted file system
- ▶ File names may resolve to different names - and some file names outside the container may be removed

What names must be virtualized?

User names

- ▶ containers may have different users w/ different roles
- ▶ **root** inside the container should not be the same as **root** outside it

Host name and IP addresses

- ▶ processes inside the container may use a different host name and IP addresses when performing network operations

namespaces

Limit the scope of kernel-side **names** and **data structures** at process granularity

mnt	(mount points, filesystems)	<i>CLONE_NEWNS</i>
pid	(processes)	<i>CLONE_NEWPID</i>
net	(network stack)	<i>CLONE_NEWNET</i>
ipc	(System V IPC)	<i>CLONE_NEWIPC</i>
uts	(unix timesharing - domain name, etc)	<i>CLONE_NEWUTS</i>
user	(UIDs)	<i>CLONE_NEWUSER</i>

Three system calls for management

- clone()** new process, new namespace, attach process to ns
- unshare()** new namespace, attach current process to it
- setns(int fd, int nstype)** join an existing namespace

Resource control

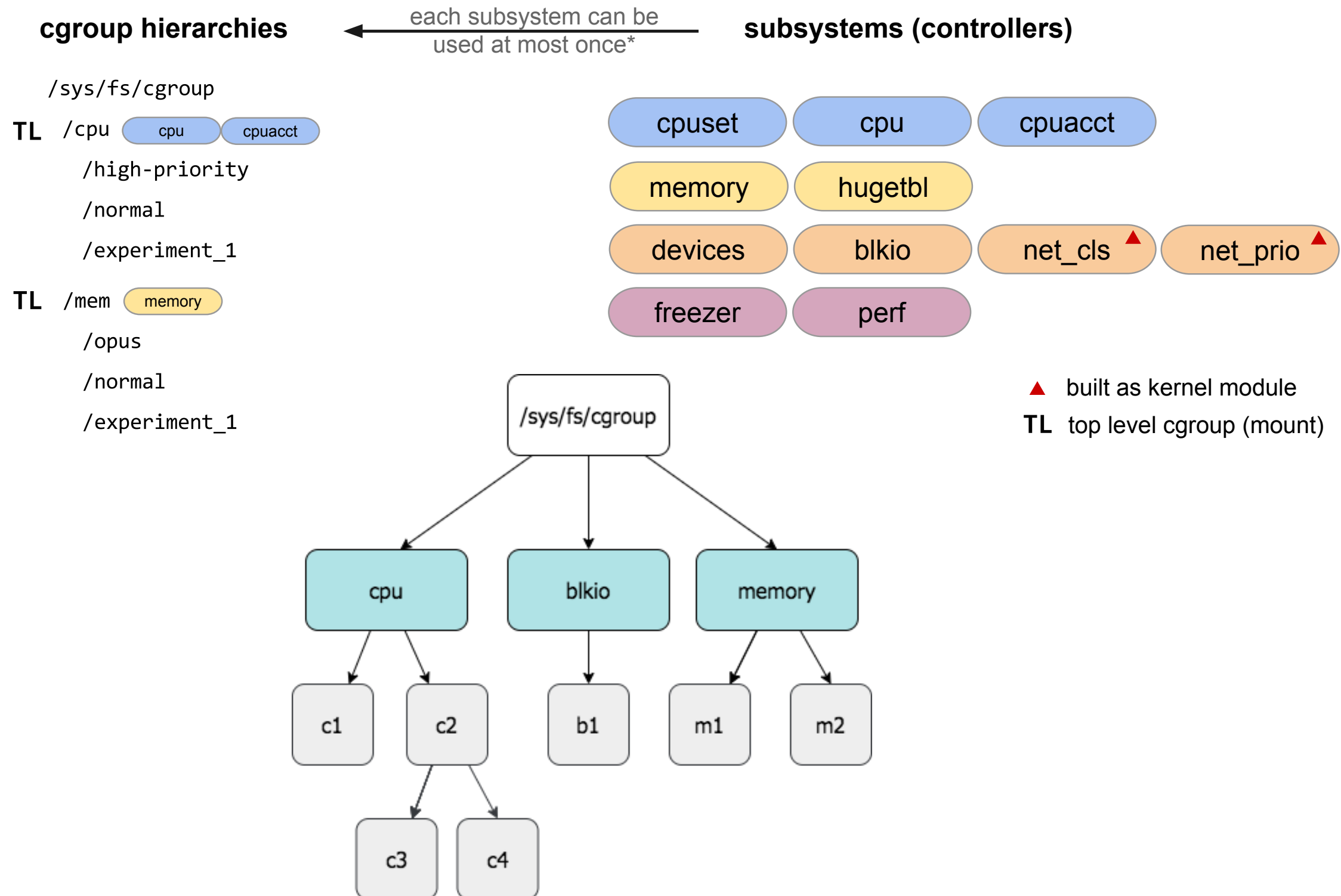
The OS may want to ensure that the entire container — or everything that runs inside it — cannot consume more than a certain amount of

- ▶ CPU time
- ▶ memory
- ▶ disk or network bandwidth

cgroups: Linux control groups

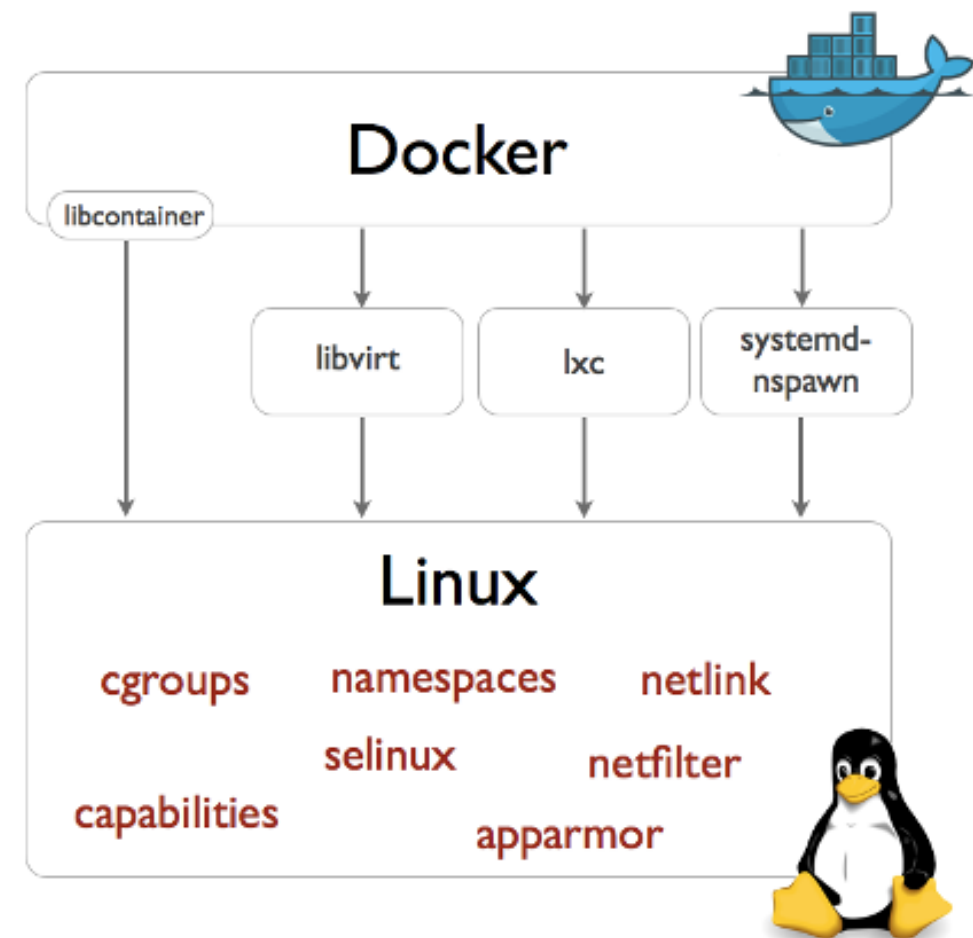
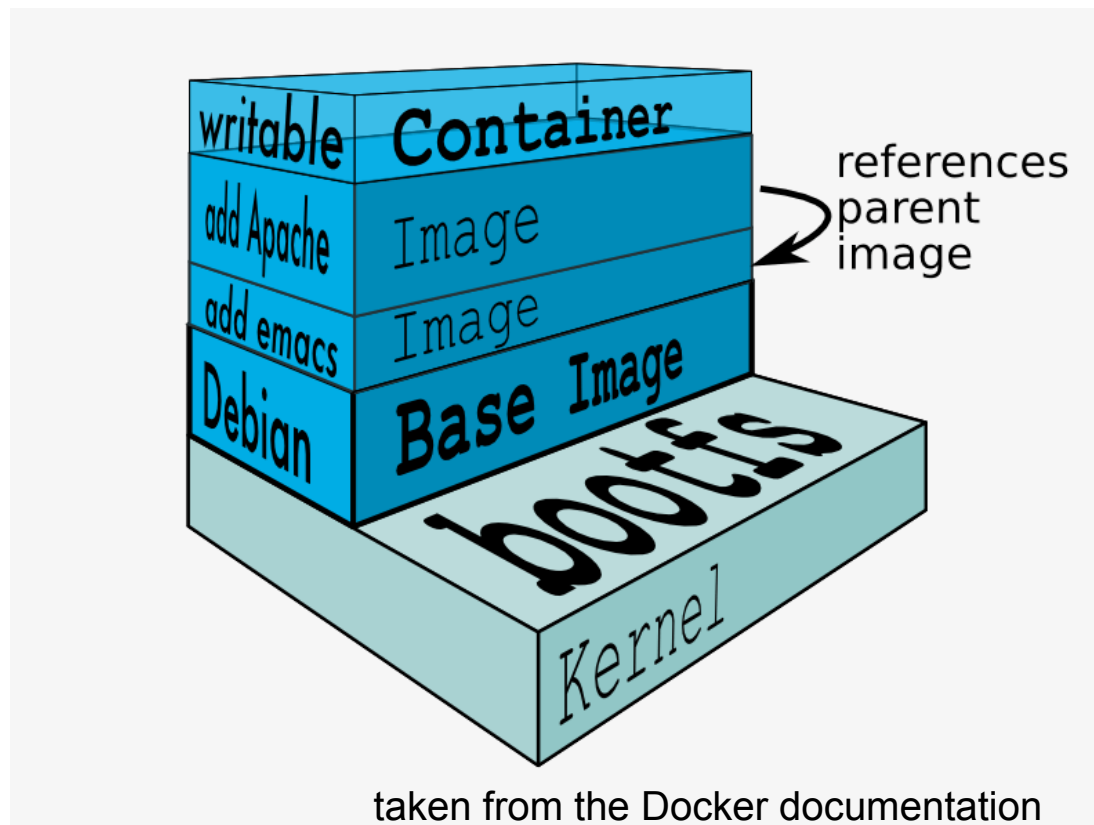
- ▶ control group subsystem offering a resource management solution for a group of processes
- ▶ Each subsystem has a *hierarchy* (a tree)
 - ▶ separate hierarchies for CPU, memory, block I/O
 - ▶ each process is in a node in each hierarchy
 - ▶ each node = a group of processes sharing the same resources

cgroup hierarchies



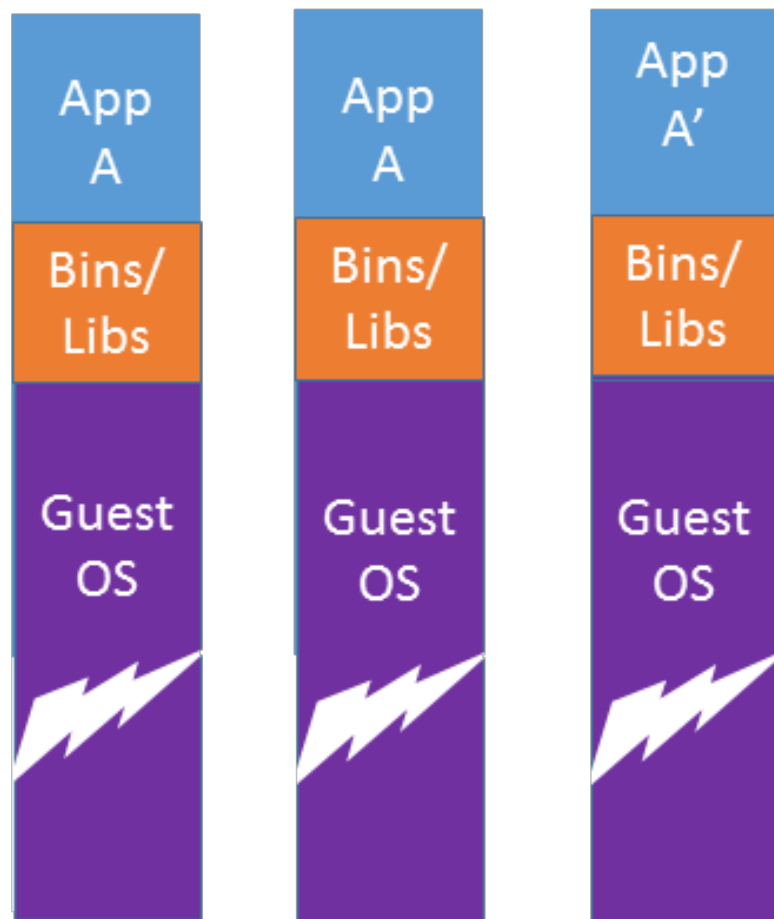
Containers

- ▶ A light form of resource virtualization based on kernel mechanisms like **cgroups** and **namespaces**
- ▶ Multiple containers **run on the same kernel** with the illusion that they are the only one using resources



Containers are lightweight

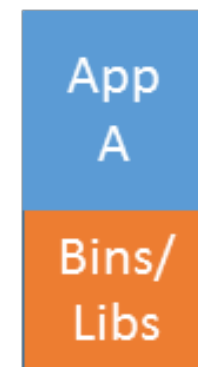
VMs



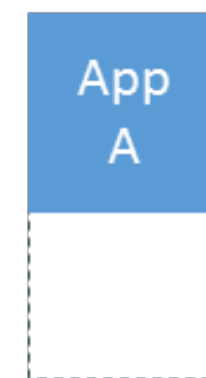
VMs

Every app, every copy of an app, and every slight modification of the app requires a new virtual server

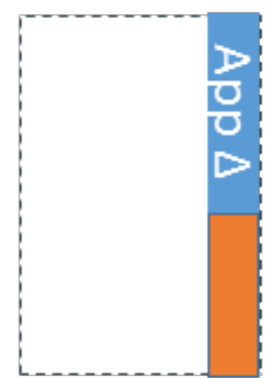
Containers



Original App
(No OS to take up space, resources, or require restart)



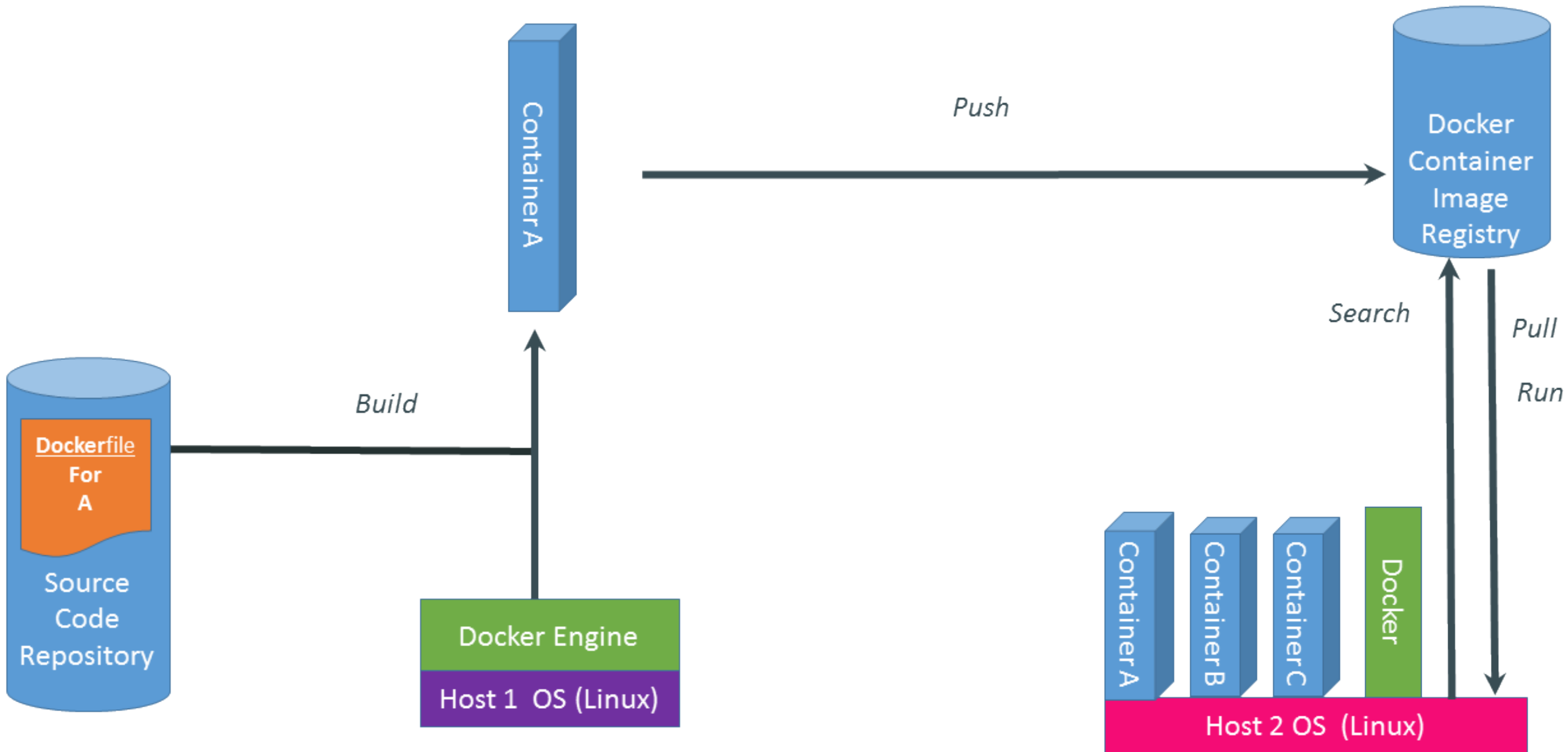
Copy of App
No OS. Can Share bins/libs



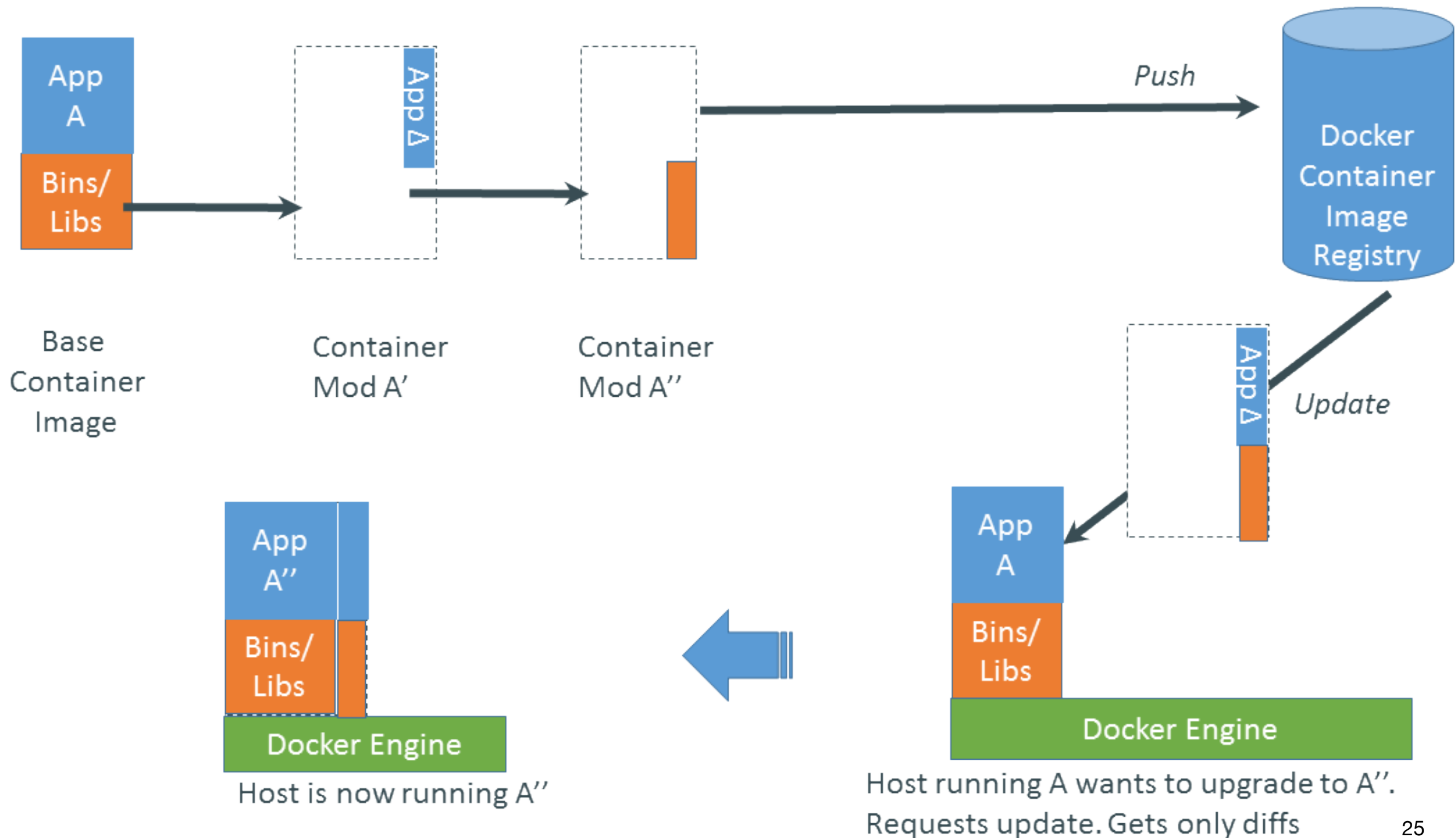
Modified App

Copy on write capabilities allow us to only save the diffs Between container A and container A'

Basics of a Docker system



Changes and updates



Credits

Slides are adapted from the community repository (CNCF) of presentations about Docker.