ViryaOS RFC:
Secure Containers for Embedded and IoT

A proposal for a new Xen Project sub-project

Stefano Stabellini @stabellinist

The problem

Package applications for the target

- Contain all dependencies
- Easy to update, Independent lifecycle

Run applications on the target

- Run in isolation
- No interference between applications

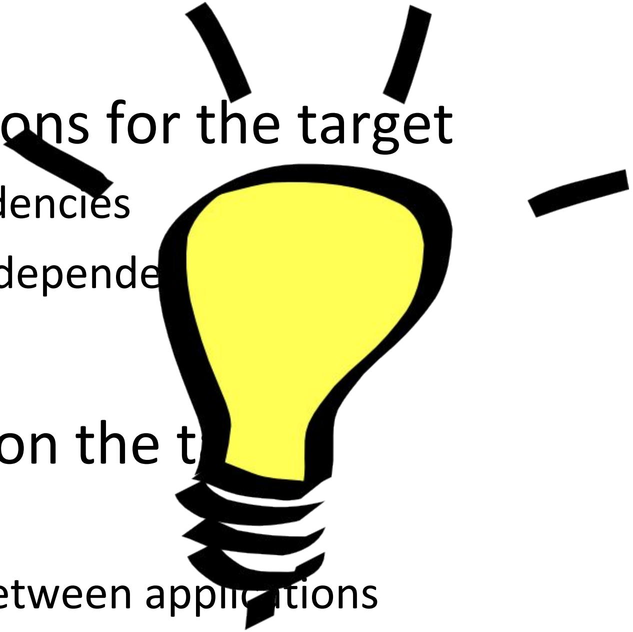
The problem

Package applications for the target

- Contain all dependencies
- Easy to update, Independe

Run applications on the t

- Run in isolation
- No interference between appli tions



The problem

Package applications for the target

Contain all dependencies

Easy to update, In

Run applications

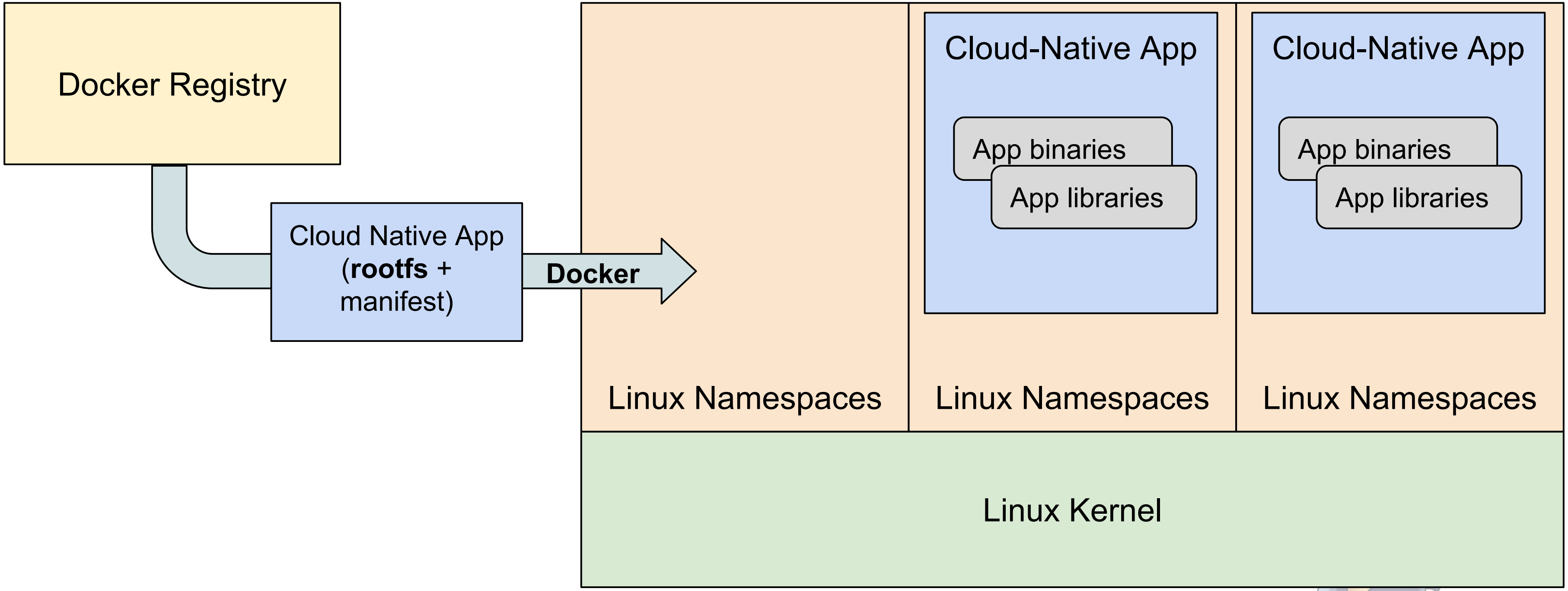
Run in isolation

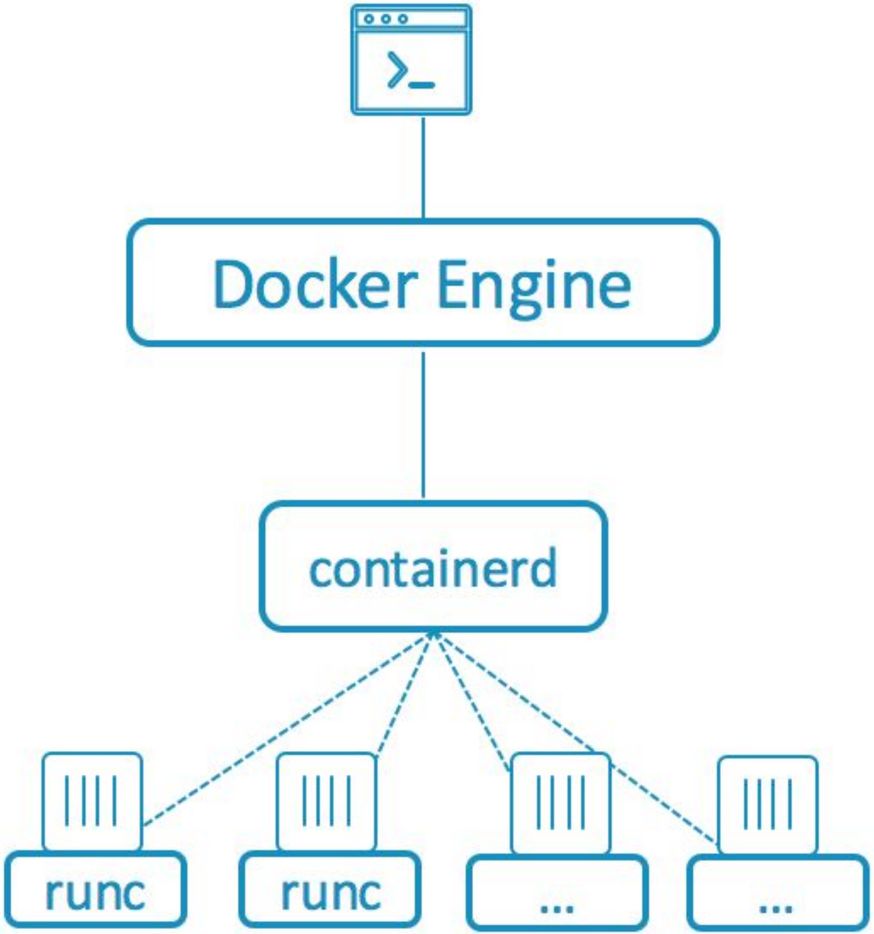
No interference between applications

Packaging vs. Runtime

OCI Image Spec vs. OCI Runtime Spec

Containers != Linux Namespaces



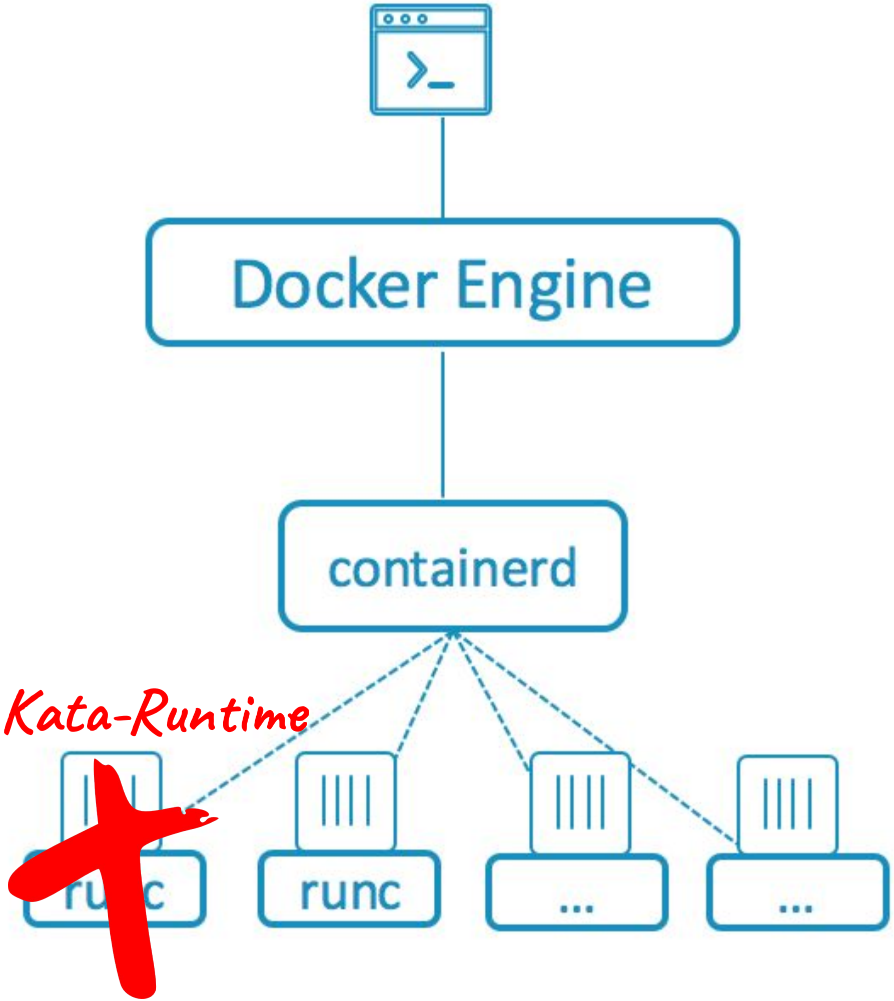


Same Docker UI and commands

User interacts with the Docker Engine

Engine communicates with containerd

containerd spins up runc or other OCI compliant runtime to run containers



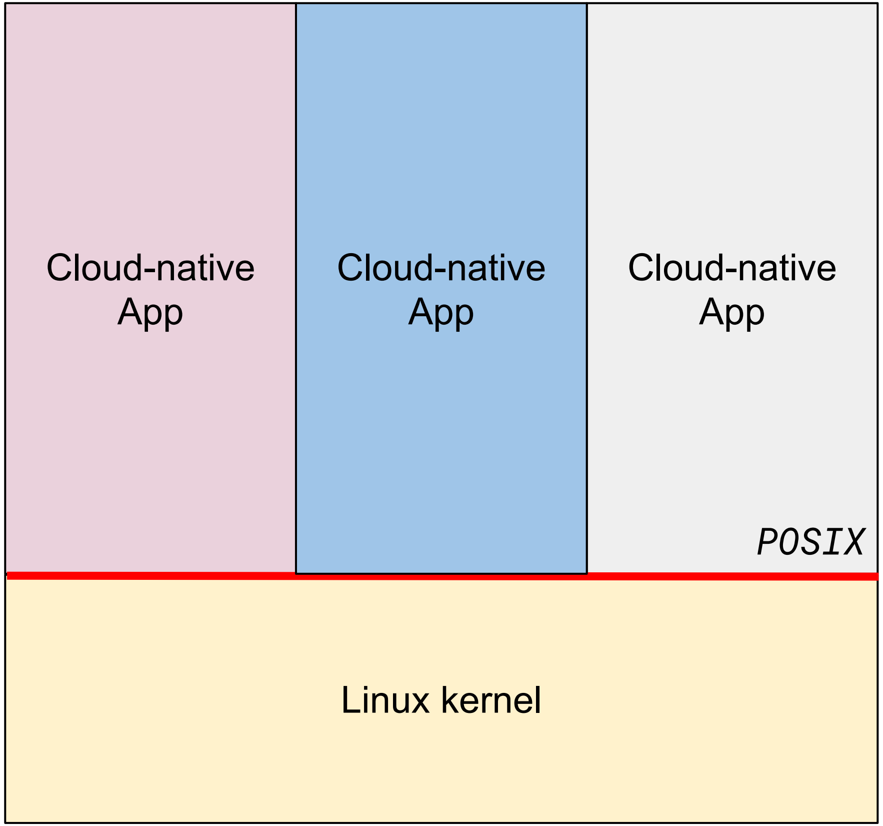
Same Docker UI and commands

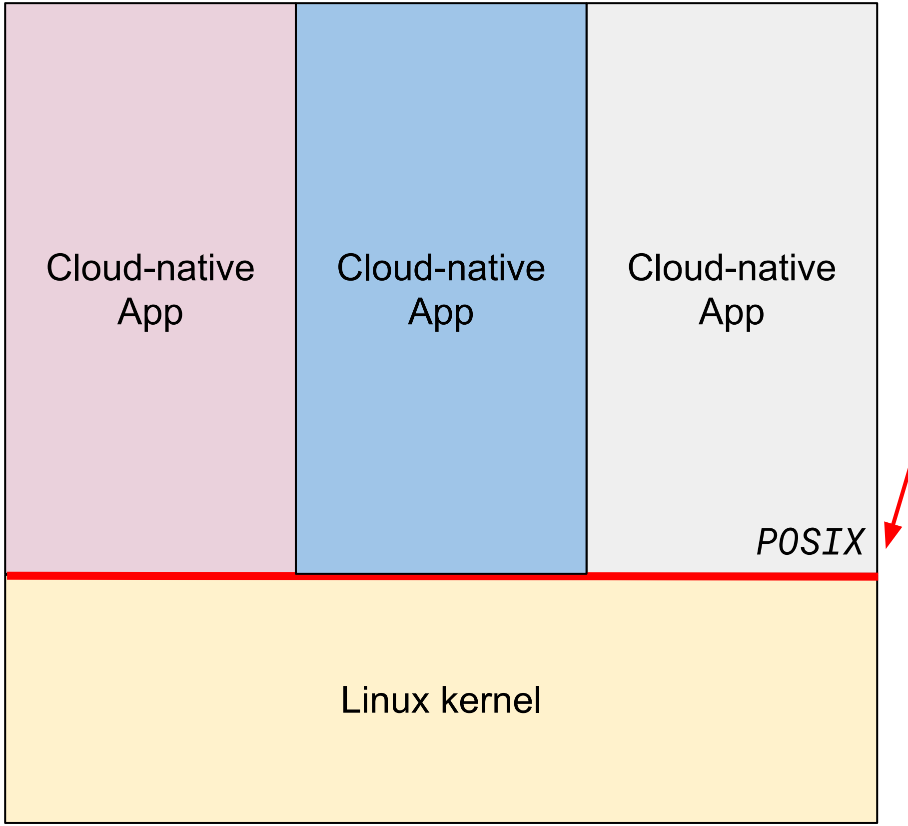
User interacts with the Docker Engine

Engine communicates with containerd

containerd spins up runc or other OCI compliant runtime to run containers

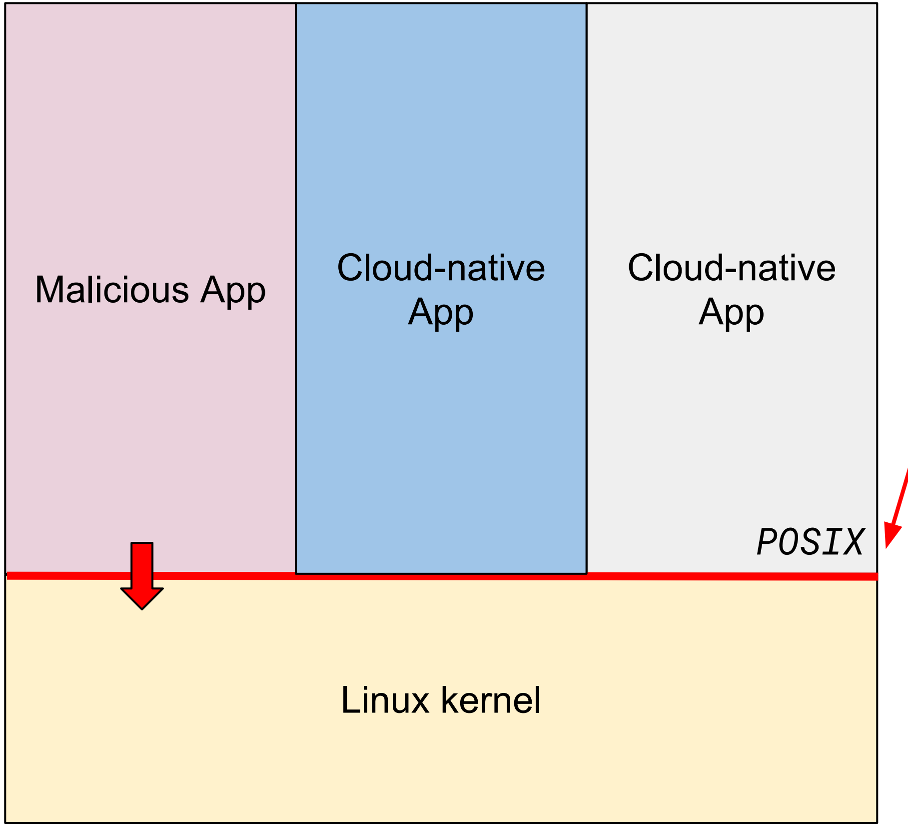
The problem with Linux namespaces


Cloud-native App
Cloud-native App
Cloud-native App
POSIX
Linux kernel



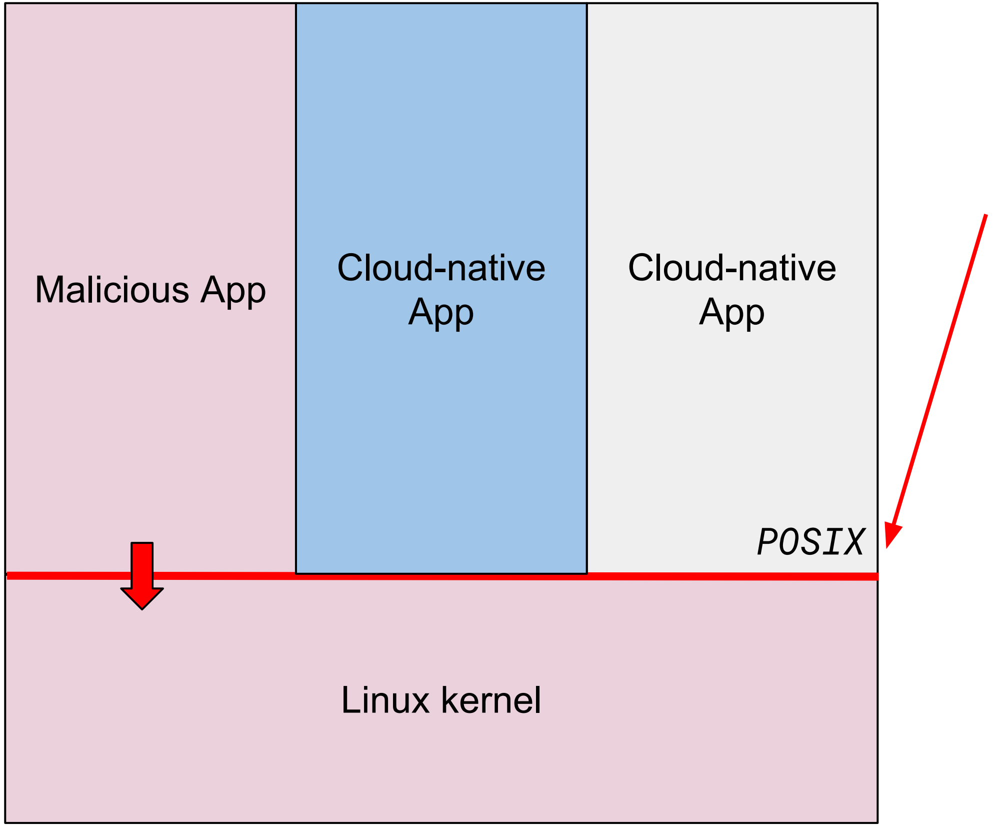
Large surface of attack

On average, 3 privilege escalation vulnerabilities per Linux release!



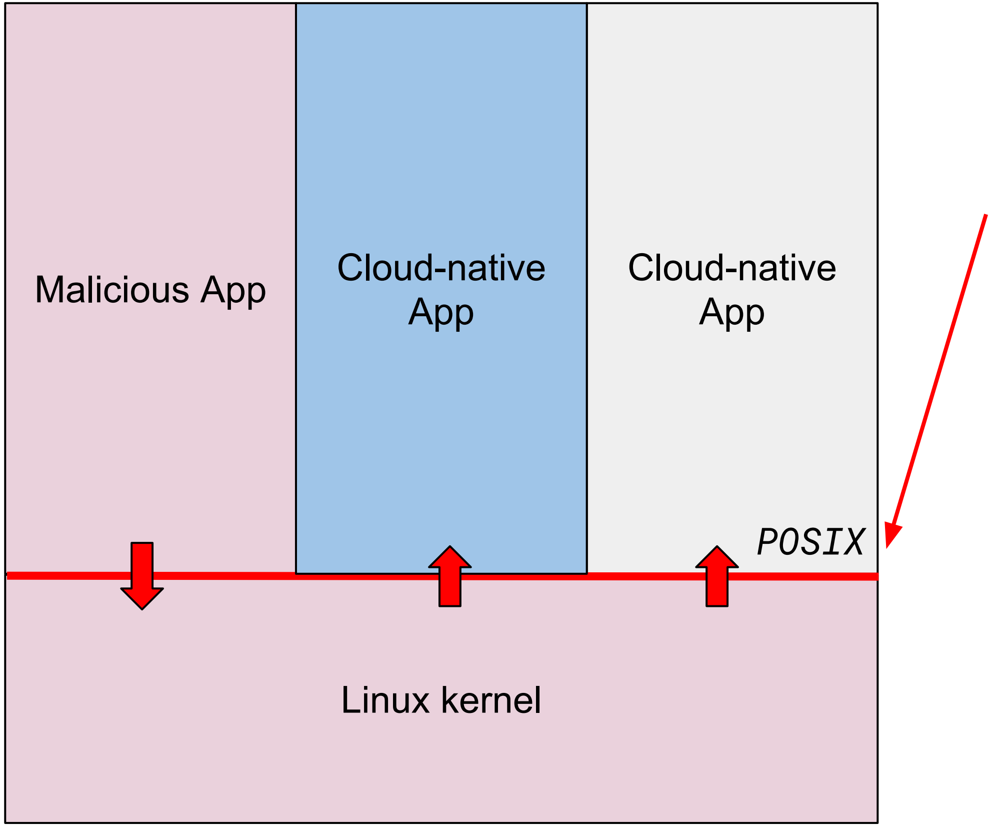
Large surface of attack

On average, 3 privilege escalation vulnerabilities per Linux release!



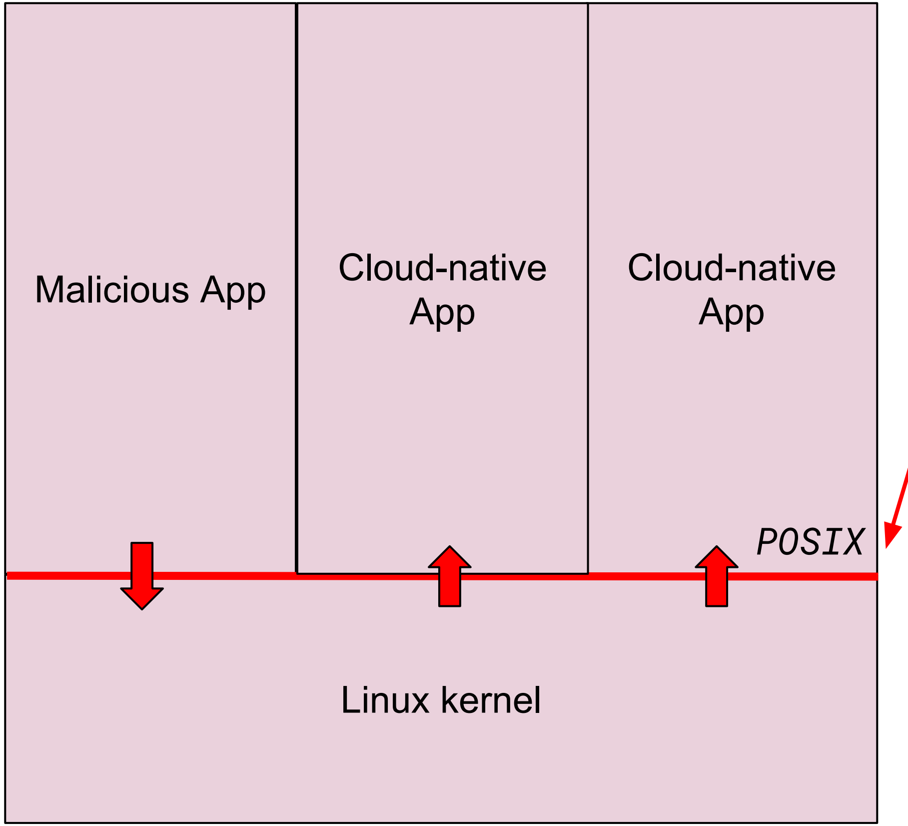
Large surface of attack

On average, 3 privilege escalation vulnerabilities per Linux release!



Large surface of attack

On average, 3 privilege escalation vulnerabilities per Linux release!



Large surface of attack

On average, 3 privilege escalation vulnerabilities per Linux release!

Security hardening techniques

From “Understanding and Hardening Linux Containers” by NCC Group:

- Run unprivileged containers (user namespaces, root capability, dropping)
- Apply a Mandatory Access Control system, such as SELinux
- Build a custom kernel binary with as few modules as possible
- Apply sysctl hardening
- Apply disk and storage limits
- Control device access and limit resource usage with cgroups
- Drop any capabilities which are not required for the application within the container

[...]

Security hardening techniques

[...]

- Use custom mount options to increase defense in depth
- Apply GRSecurity and PAX patches to Linux
- Reduce Linux attack surface with Seccomp-bpf
- Isolate containers based on trust and exposure
- Logging, auditing and monitoring is important for container deployment
- **Use hardware virtualization along application trust zones**

Security hardening techniques

- Securing Linux namespaces is ***possible*** but ***very difficult***
- It requires specific knowledge of the cloud-native app
- Auditing and monitoring should be performed everywhere
- Using ***virtualization*** for isolation is still ***recommended***

Linux Namespaces: very embedded problems

Mixed-criticality workloads are not supported

Limits on resources utilization hard to enforce

Real-time support is difficult

Certifications are very difficult

An Example: Singularity

Containers are a great packaging format

Linux Namespaces are not suitable for all use-cases

Singularity: bringing Docker containers to HPC

https://goo.gl/5Cw9uh

The Solution for Embedded and IoT: Xen As Containers Runtime

- Security, Isolation and Partitioning
- Multi-tenancy
- Mixed-criticality workloads
- Hardware access to applications
- Real-time support

ViryaOS: a ready-to-use runtime environment for VMs and Secure Containers

The problem #2

Cross-building multiple VMs is difficult

Assembling the output in a single runnable image is a manual process

Embedded and IoT use patterns

Typically users know all the VMs they need beforehand

They still need to:

- build them all, plus Xen and Dom0
- install all images on target
- partition the hardware using device assignment
  - edit the Dom0 device tree
  - generate appropriate device trees for DomUs with device nodes
- plan for image upgrades and security fixes

It’s a lot of work!


You think this is bad enough...
...then you try disaggregation

Current status

Everybody has their own scripts and handcrafted solutions

- They are limited
- Only target one use-case
- Limited support for driver domains and service domains
- Only support one hardware platform

→ We would all benefit from a unifying effort

ViryaOS

A proposal for a new Xen Project sub-project

ViryaOS

A **Secure** Xen based runtime

Containers supported natively

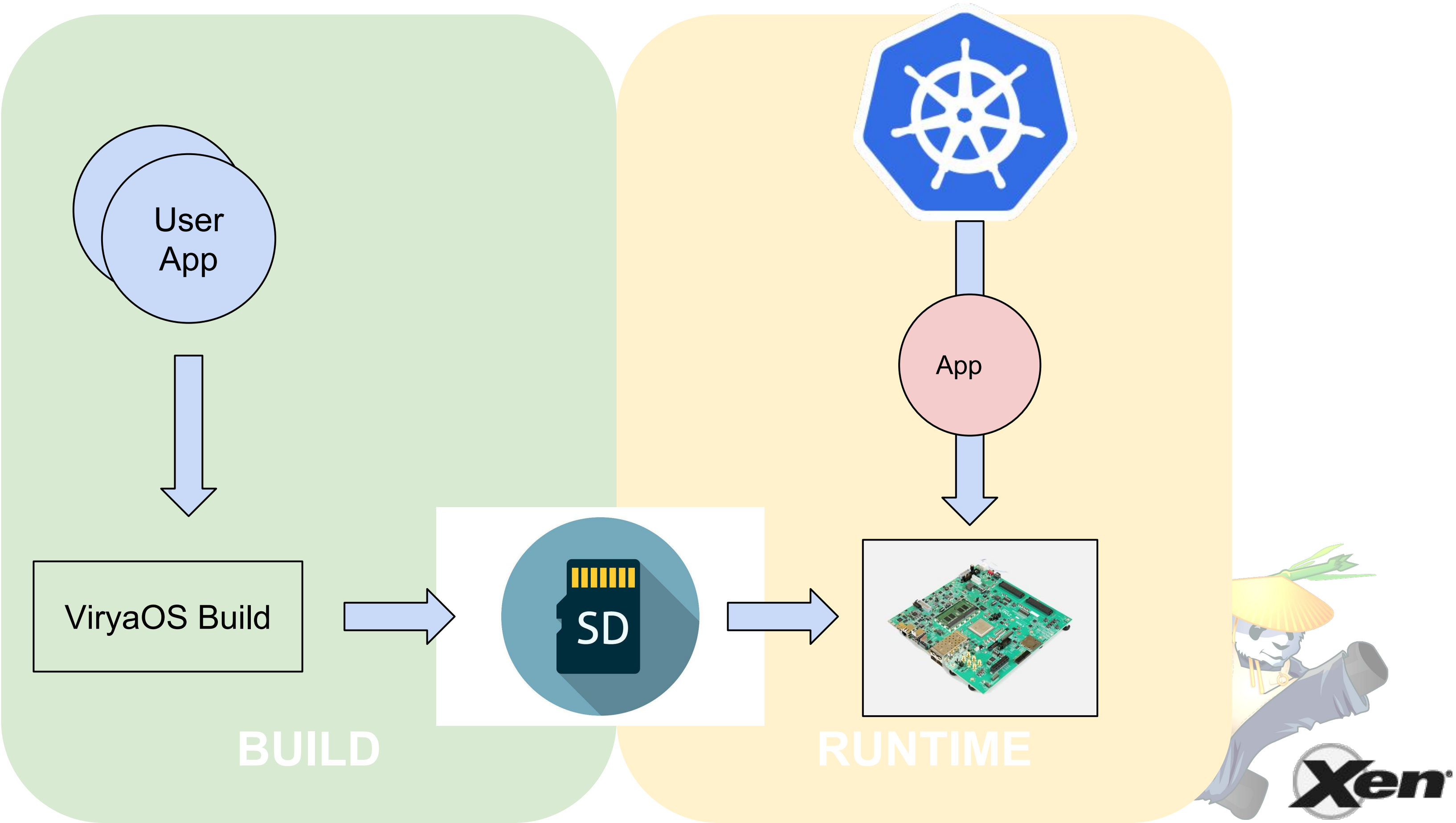
A turnkey solution

A **Flexible** build system

Supports aarch64 and x86_64

Targeted at Embedded and IoT

ViryaOS



ViryaOS Runtime

Dynamically deploy VMs and Secure Containers

Containers are run securely, transparently as Xen **VMs**

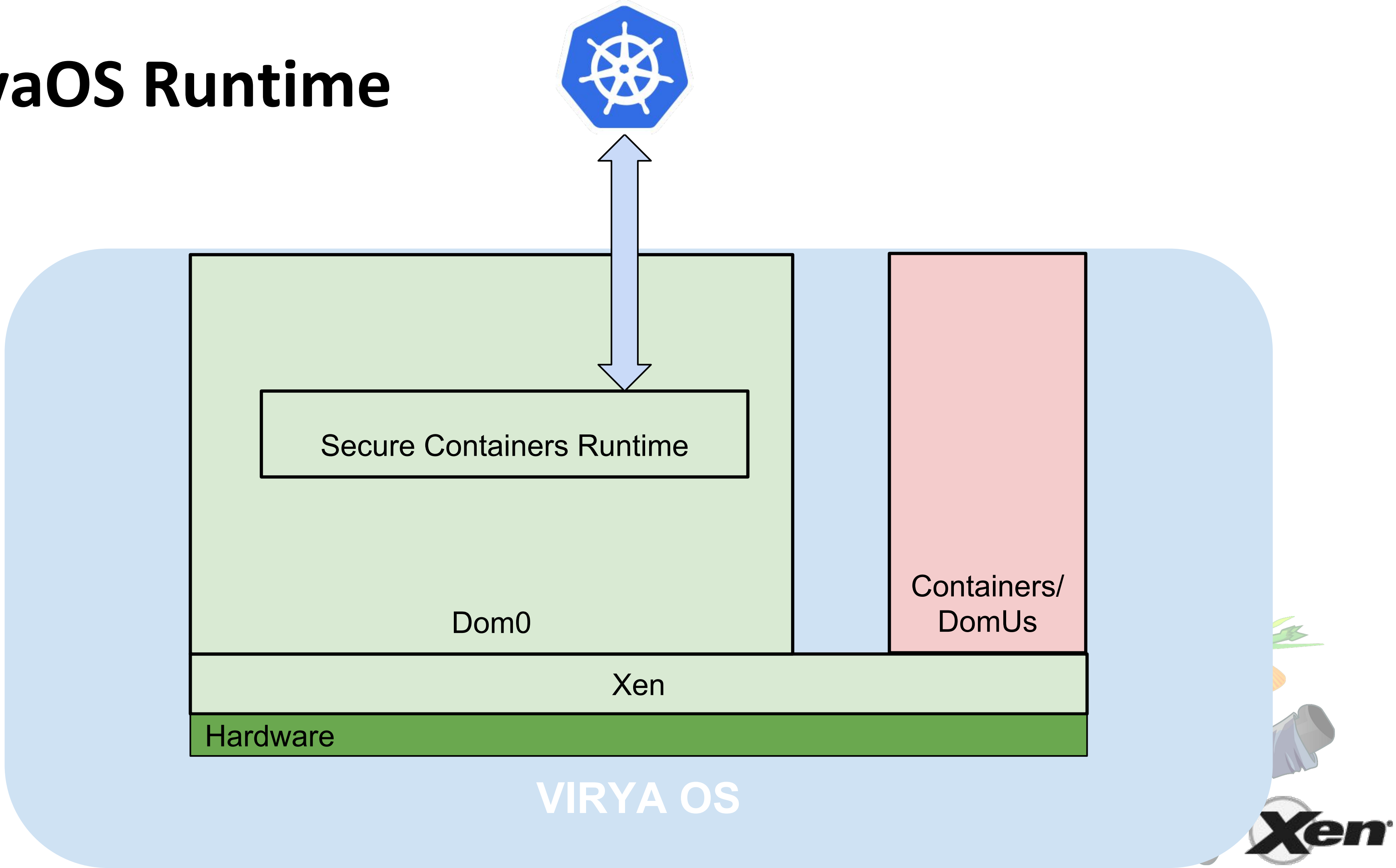
- 1 Kubernetes Pod per VM
- *See <u>KataContainers</u> and stage1-xen*

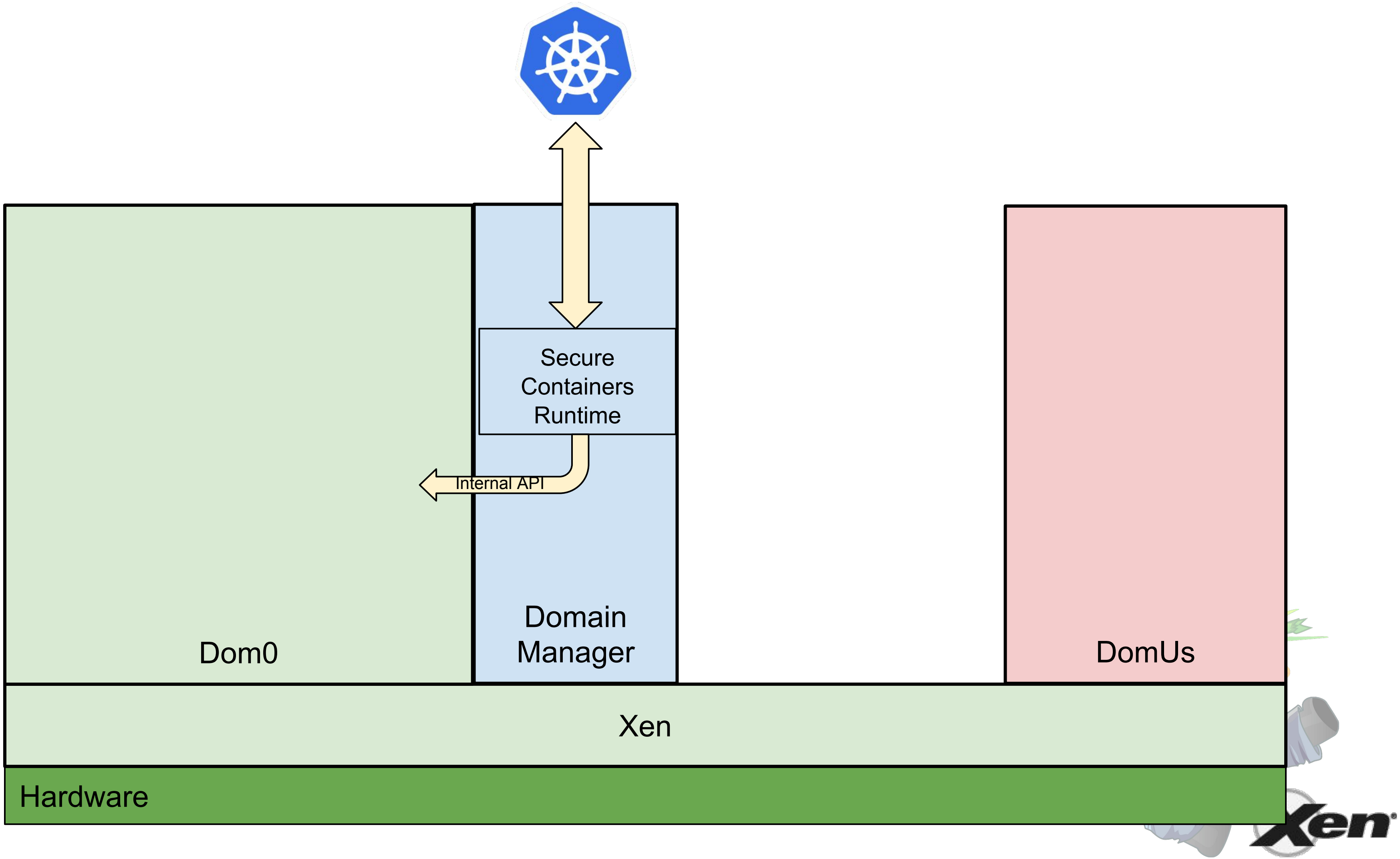
Measured Boot

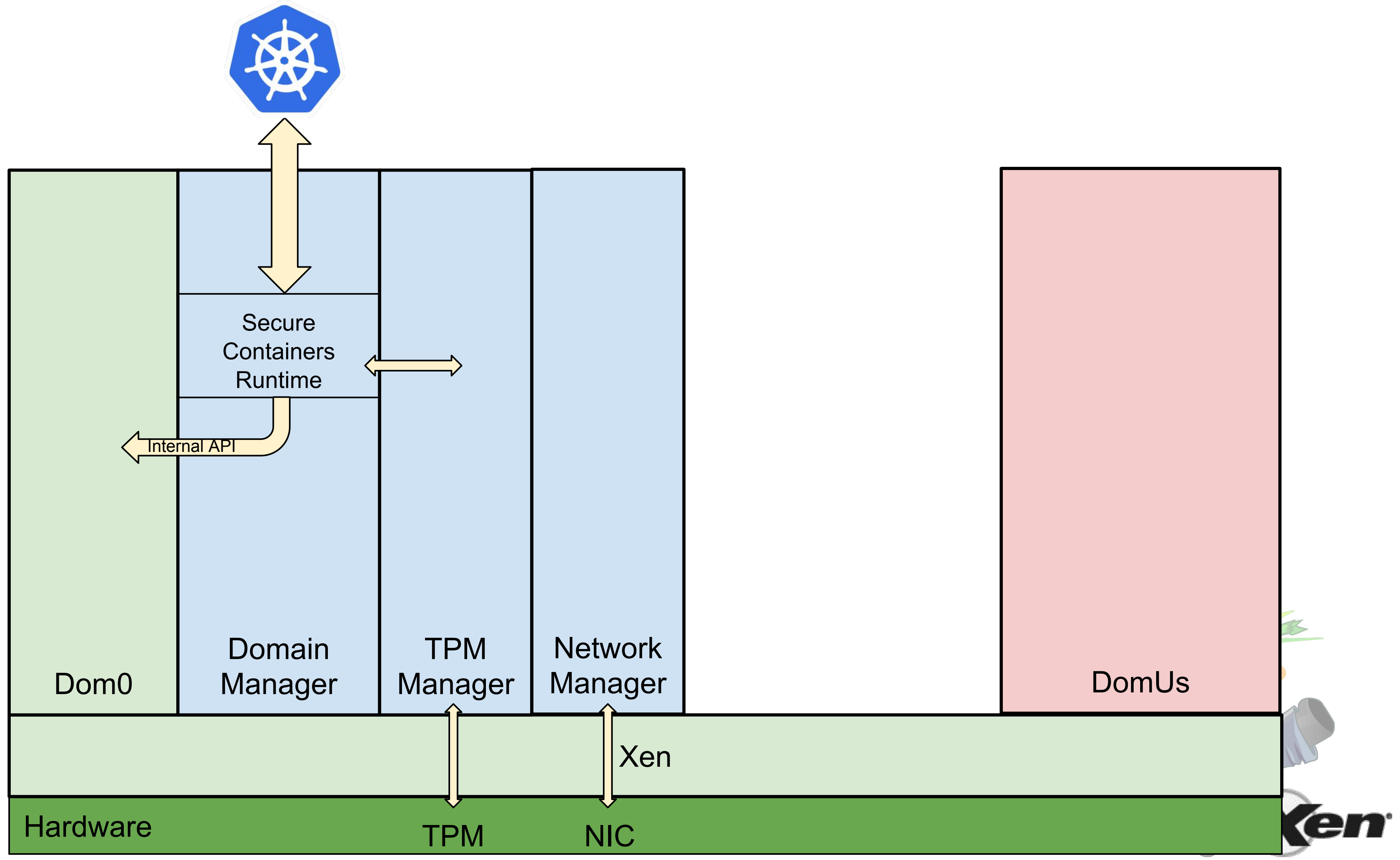
System Software Updates, and Containers updates

Uses **disaggregation**, service domains and driver domains

ViryaOS Runtime




Secure
Containers
Runtime
Internal API
Dom0
Domain
Manager
DomUs
Xen
Hardware


Secure
Containers
Runtime
Internal API
Dom0
Domain
Manager
TPM
Manager
Network
Manager
DomUs
Xen
Hardware
TPM
NIC

ViryaOS: Build

A **multi-domain** build system

Builds multiple domains in one go

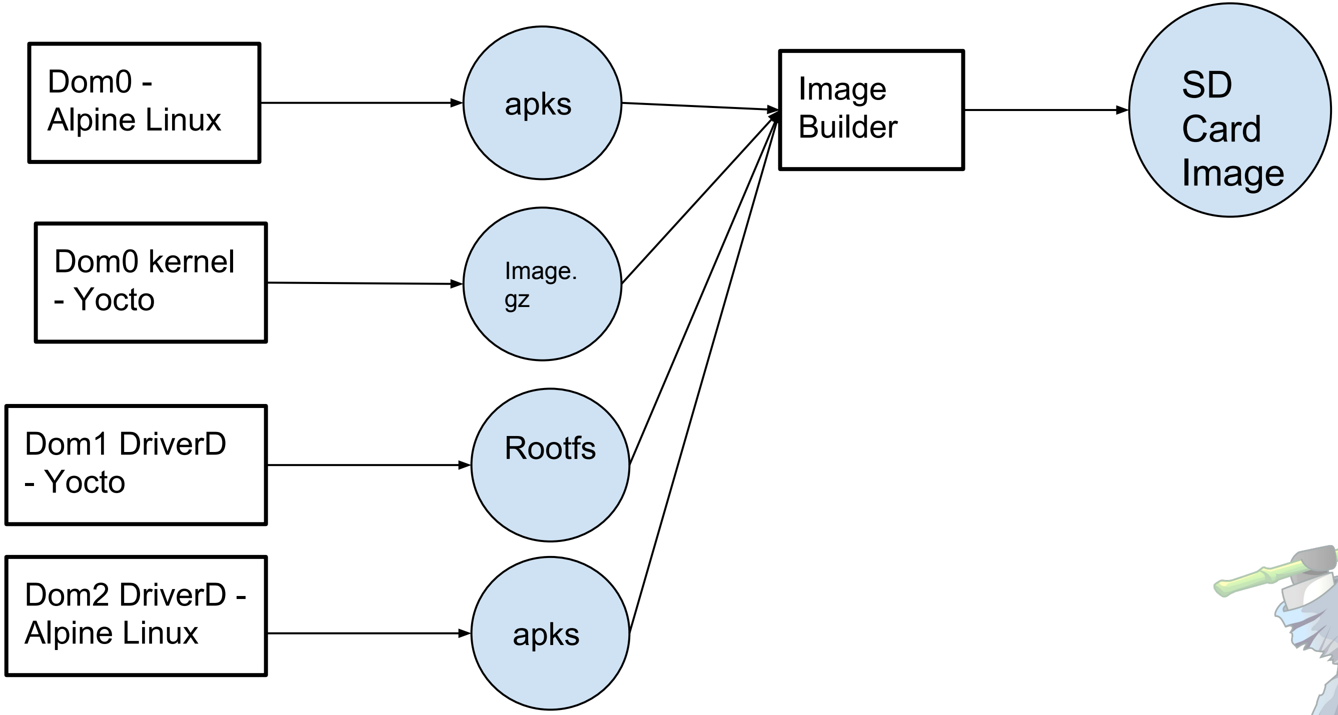
Creates a runnable SD Card image from multiple domain builds

Each domain build is independent and runs in a container

Pre-configured device passthrough to guests

Made for disaggregated architectures

ViryaOS Build



ViryaOS Build

Everything builds in a container

Supports cross-builds (ARM64 targets on x86) with qemu-user

Supports any build systems for domain builds

- enables mixed Alpine Linux / Yocto environments
- rootfs and kernel can be built independently

Supports multiple DomU build output formats

The DomU output itself is stored in a container

Anything can be pull to the Docker Hub to speed up the build

Status

Very early stages, experimental

Interest, but no company backers yet, community driver

Subscribe to the mailing list to learn more and participate!

Initial implementation available for:

- SDK
- Containers-driven build
- Yocto kernel build

Missing ImageBuilder

Questions?

To Be Continued....