How Container Runtimes matter in Kubernetes?

Kunal Kushwaha
NTT OSS Center

About me

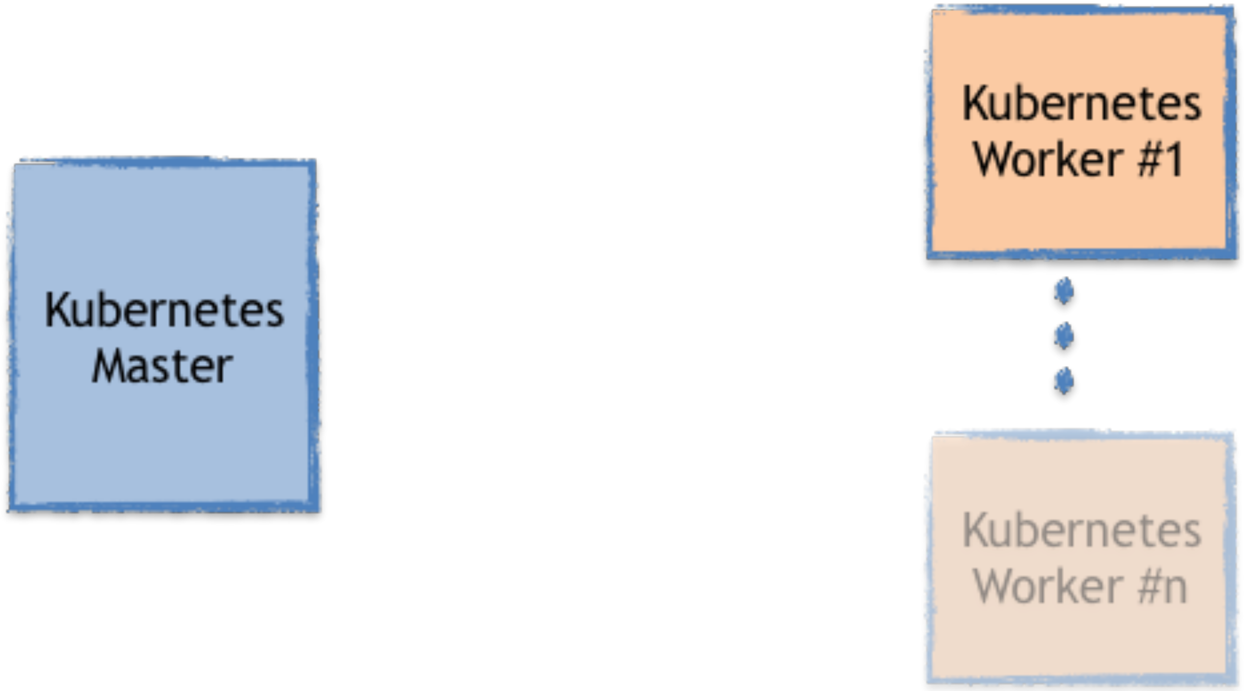
- Works @ NTT Open Source Software Center
- Contributes to containerd and other related projects.
- Docker community leader, Tokyo

@kunalkushwaha

Agenda

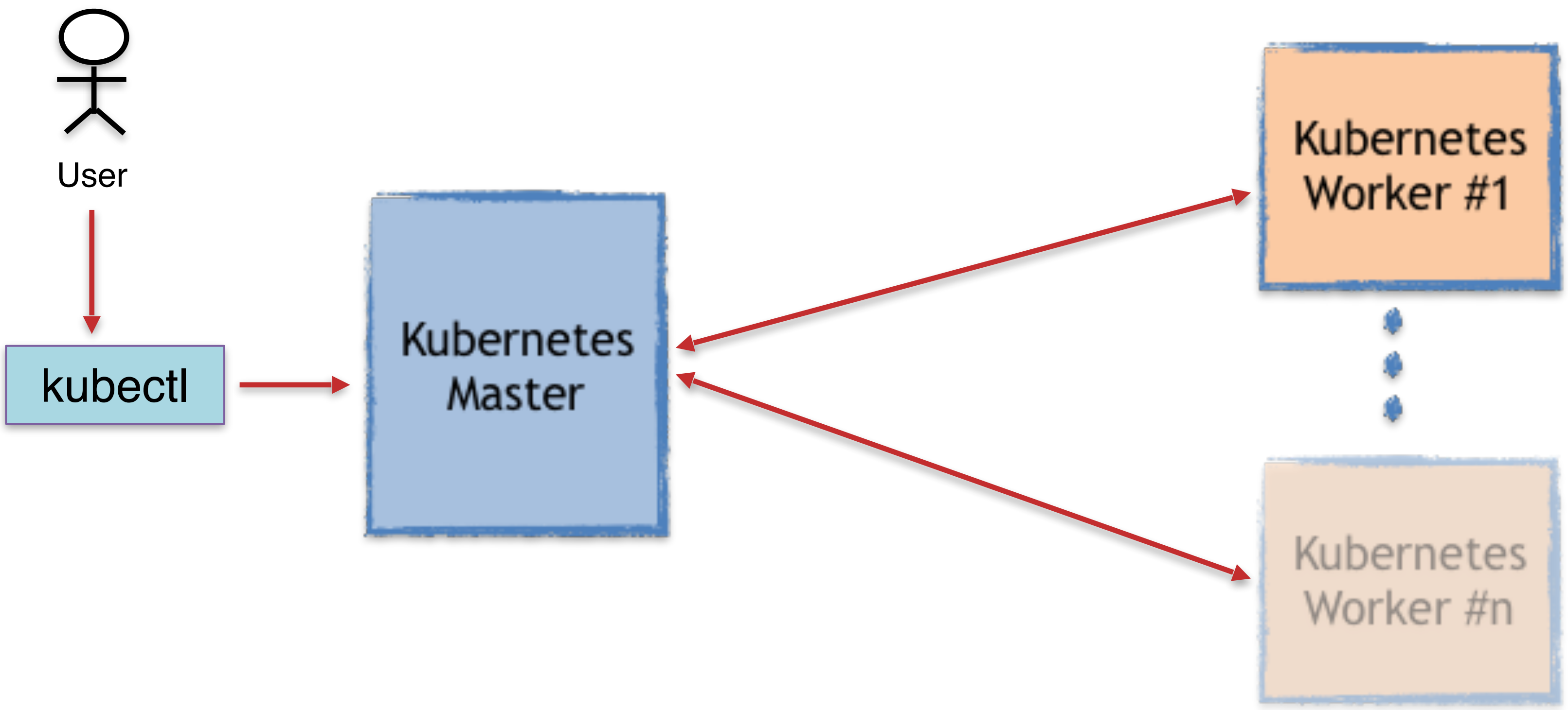
- Kubernetes Architecture.
- What is CRI (Container Runtime Interface)
- What is OCI (Open Container Initiative)
- CRI & OCI Implementations
- Why runtimes affect Kubernetes.
- Runtime Benchmarking results
- Analyzing for various workloads
- Summary

Kubernetes Architecture



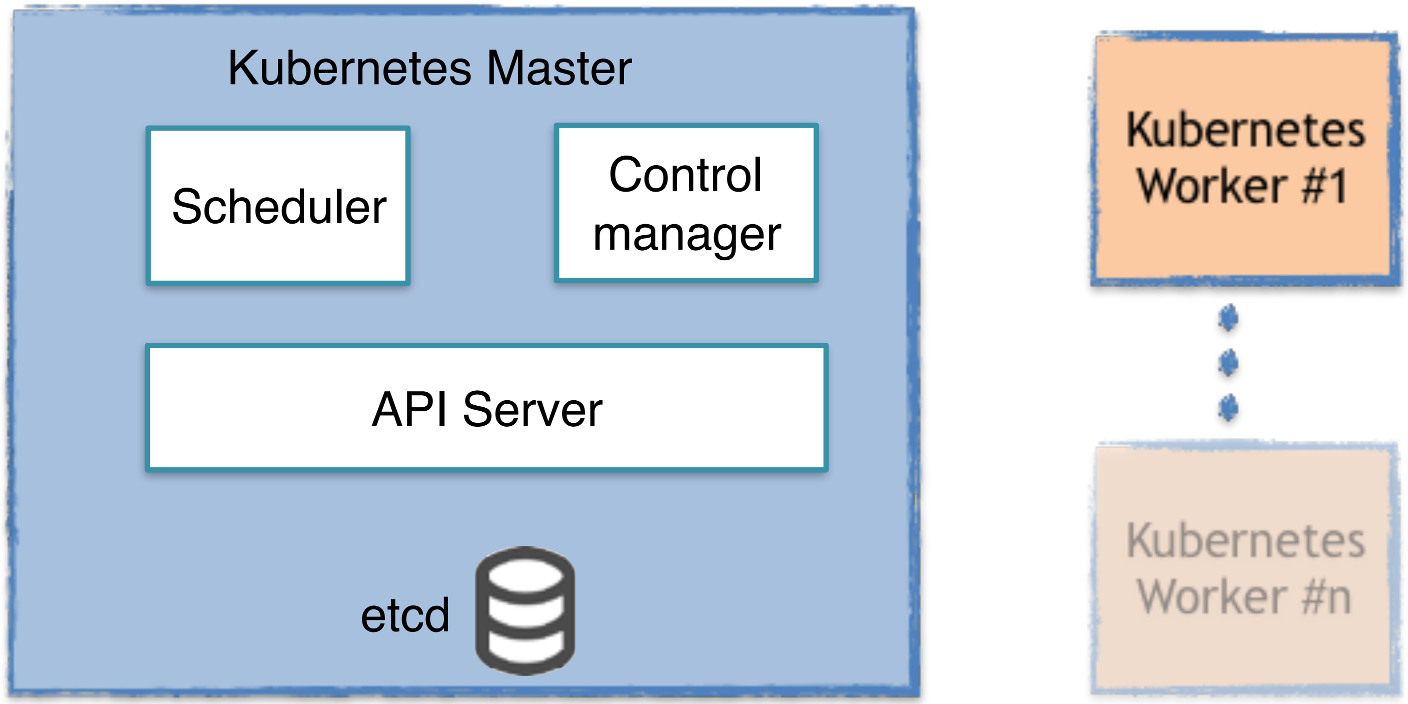
A typical Kubernetes cluster

Kubernetes Cluster Overview



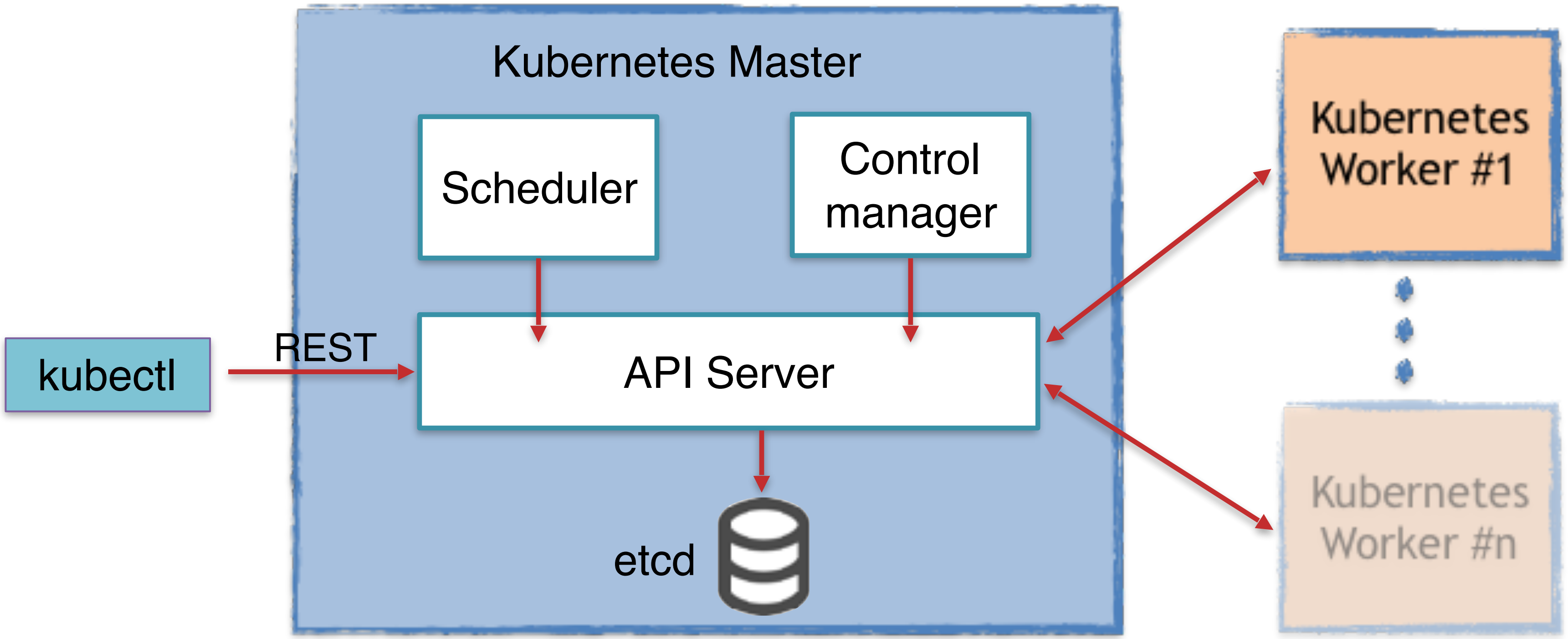
- `kubectl` is tool for user to interact with k8s cluster.
- Master node interpret the command and if required interact with worker nodes.

Master Node Overview



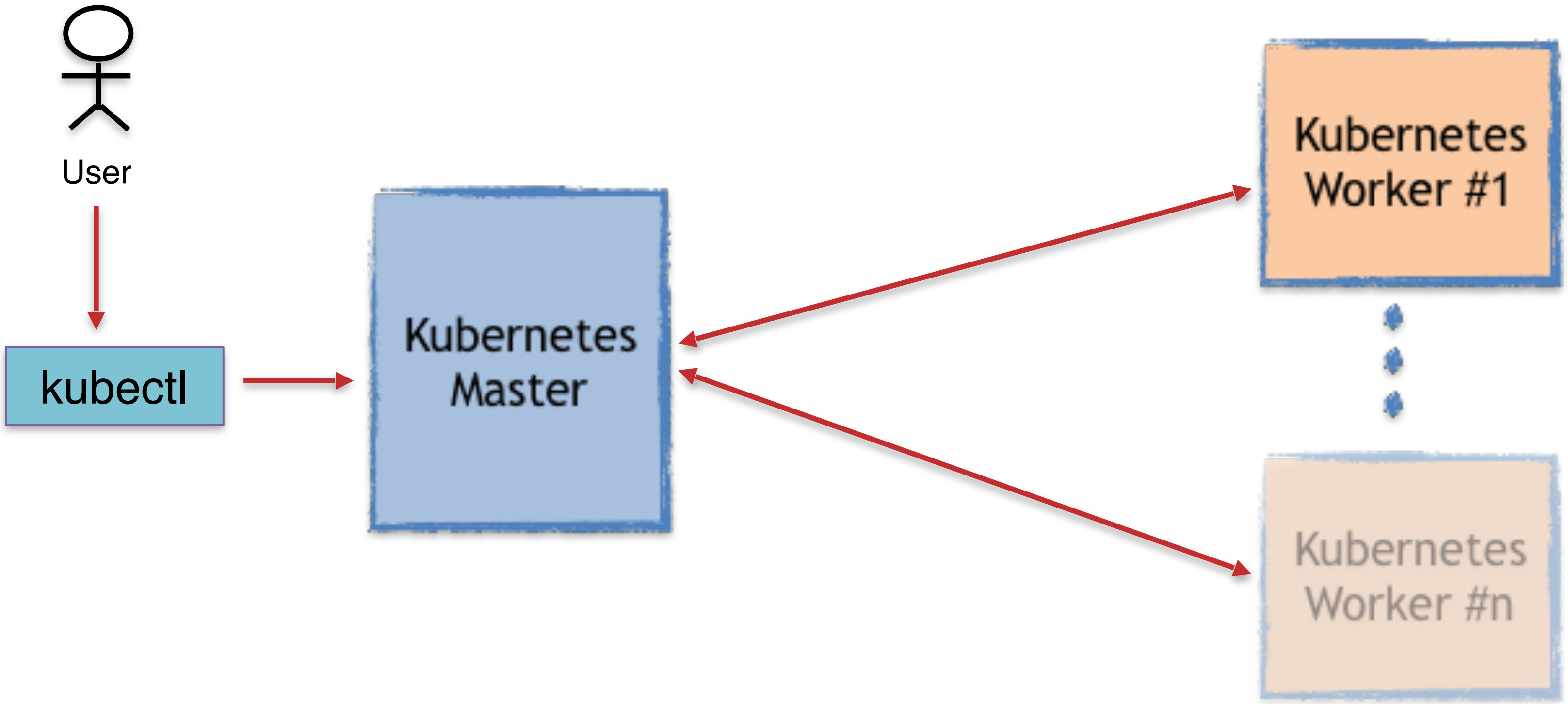
Important components of Kubernetes Master Node

Master Node Control Flow

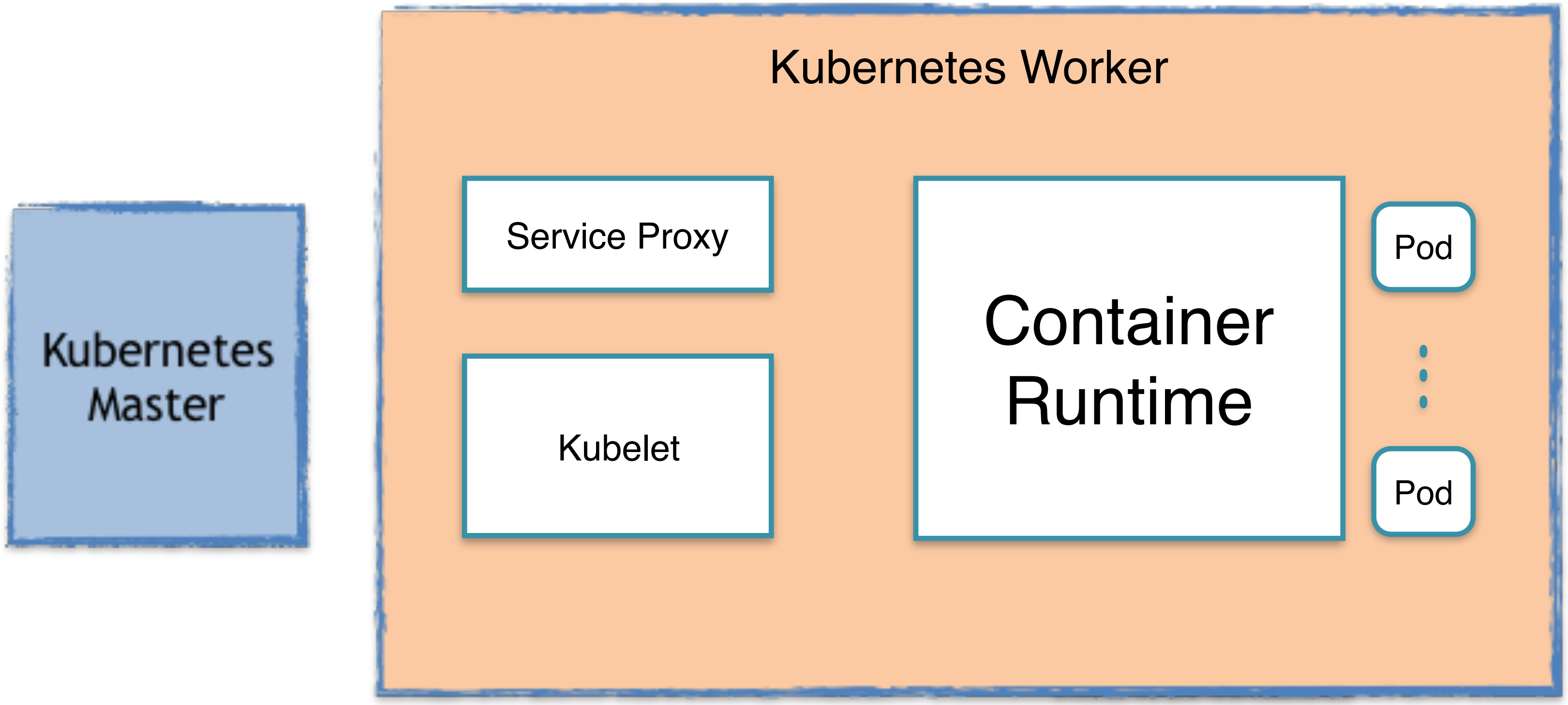


- API Server plays a central part for cluster communication
- etcd store all definition of kubernetes resources
- Scheduler and Control Manager push commands for workers via API Server

Kubernetes Architecture

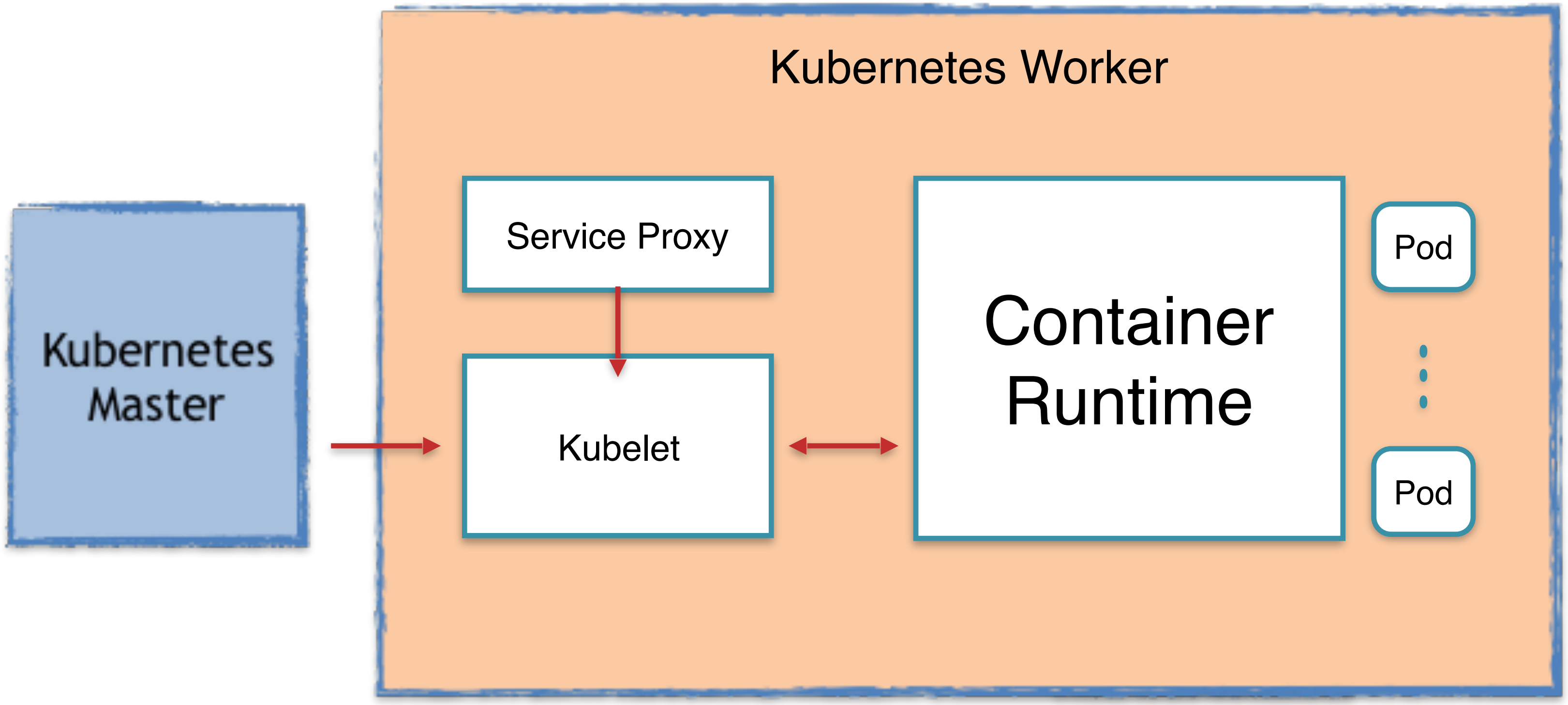


Kubernetes Worker Overview



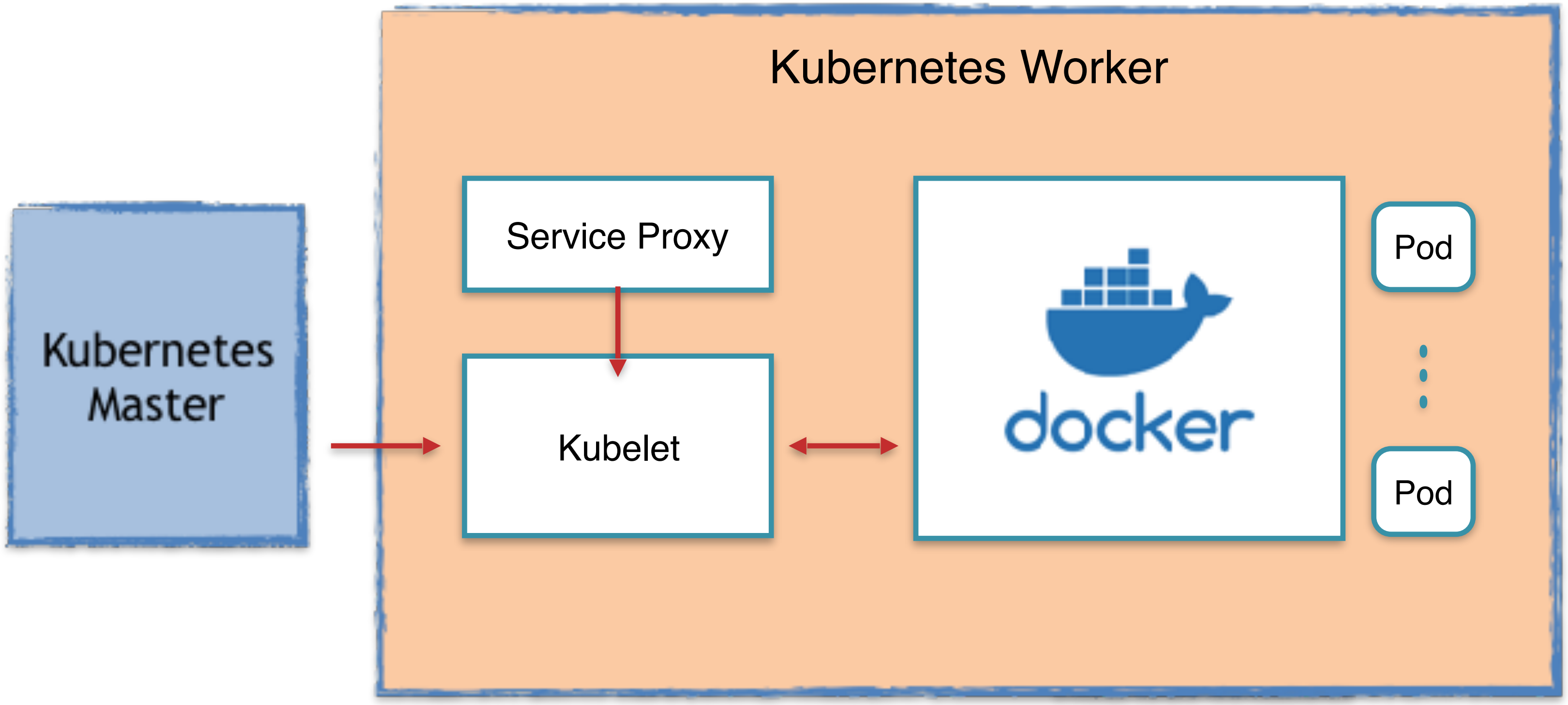
Important components of Kubernetes Worker Node

Kubernetes Worker Control Flow



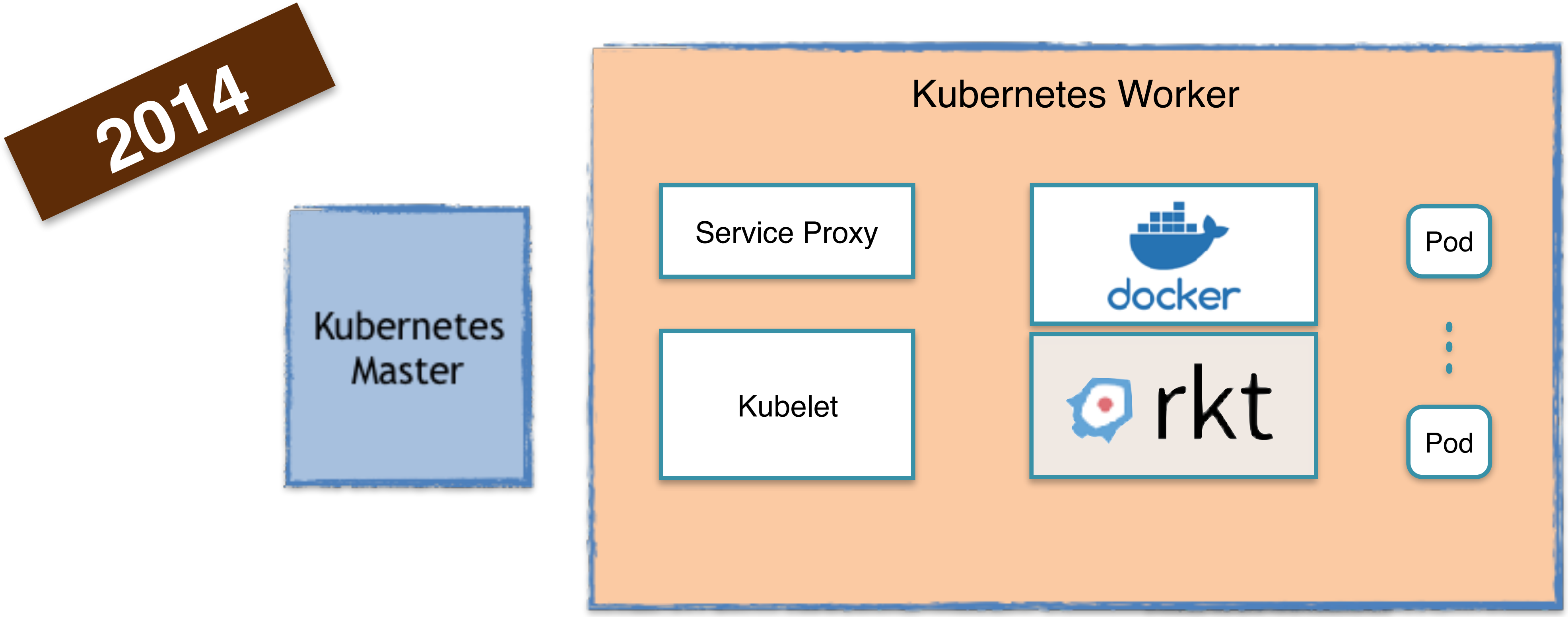
- Kubelet is the primary Node agent. API Server talks to Kubelet.
- Service Proxy enables user to access applications running on node.
- Docker running on node is used for creating Pods.

Kubernetes Worker Control Flow



- Kubelet is the primary Node agent. API Server talks to Kubelet.
- Service Proxy enables user to access applications running on node.
- Docker running on node is used for creating Pods.

Kubernetes Worker Overview



With alternative container runtimes, Kubelet code gets bloated to support each.

Container Runtime Interface

Introduced in Kubernetes 1.5 *. (2016)

Interfaces for gRPC service for Runtime & Image Management

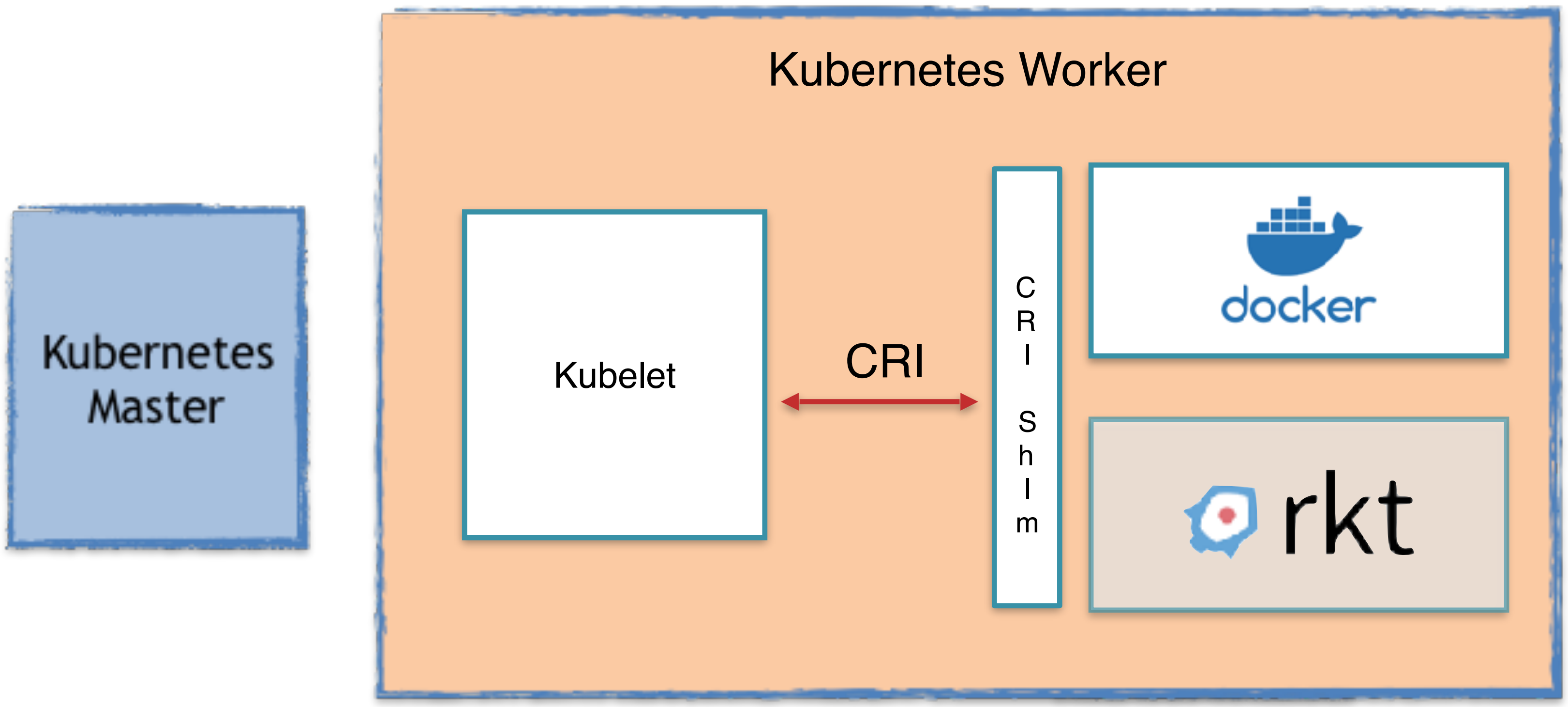
Container centric interfaces

Pod containers as Sandbox containers

Current status: v1alpha2

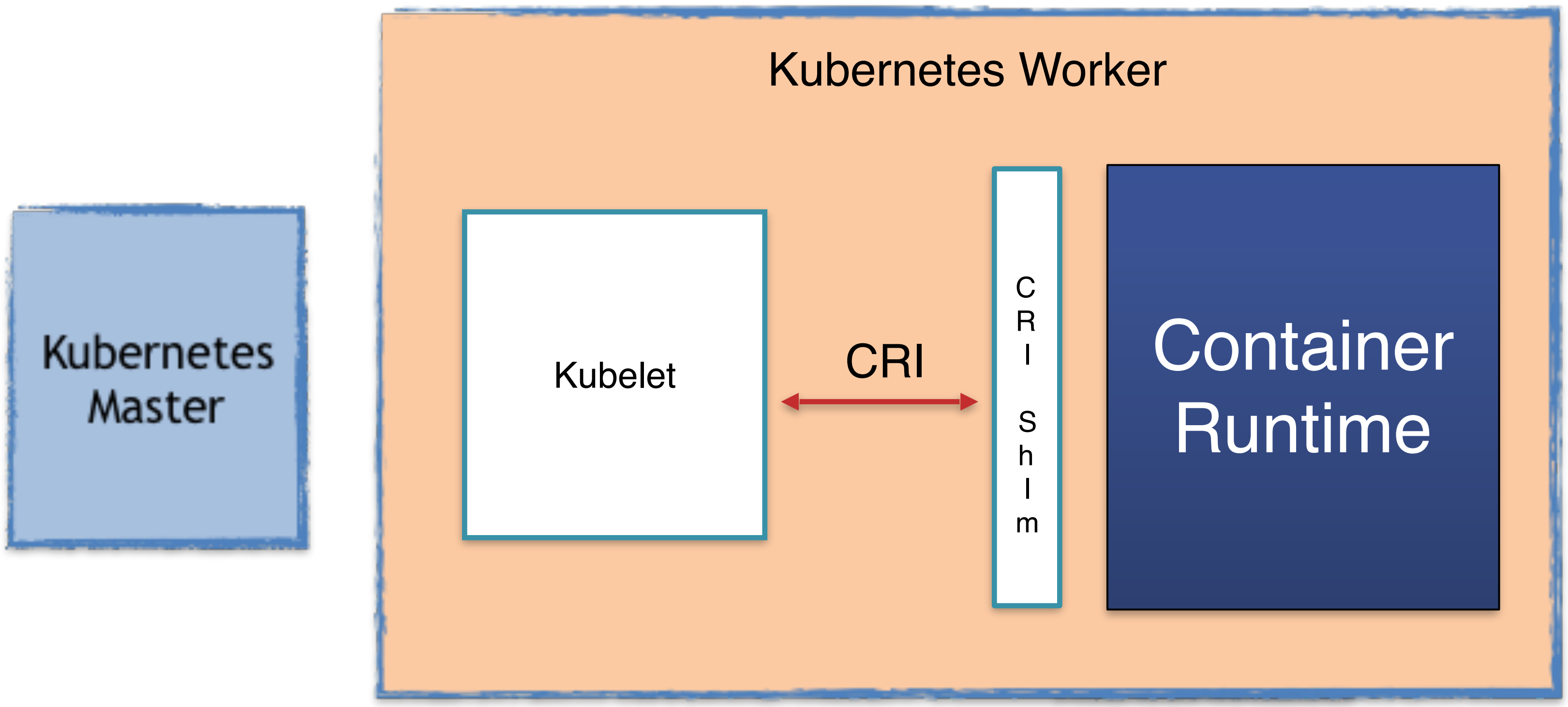
*https://github.com/kubernetes/kubernetes/blob/release-1.5/docs/proposals/container-runtime-interface-v1.md

Kubelet with CRI



CRI solves supporting various runtime alternatives with no change in Kubelet

Container Runtime



What is Container Runtime

Provides core primitives to manage containers on host

Container execution & supervision

Network Interfaces and management

Image management

Manage local storage

e.g. LXC, Docker, rkt

Open Container Initiative

Container runtime & Image specification

Runtime specs define input to create a container

Multiple platform supported (Linux, Windows, Solaris & VM)

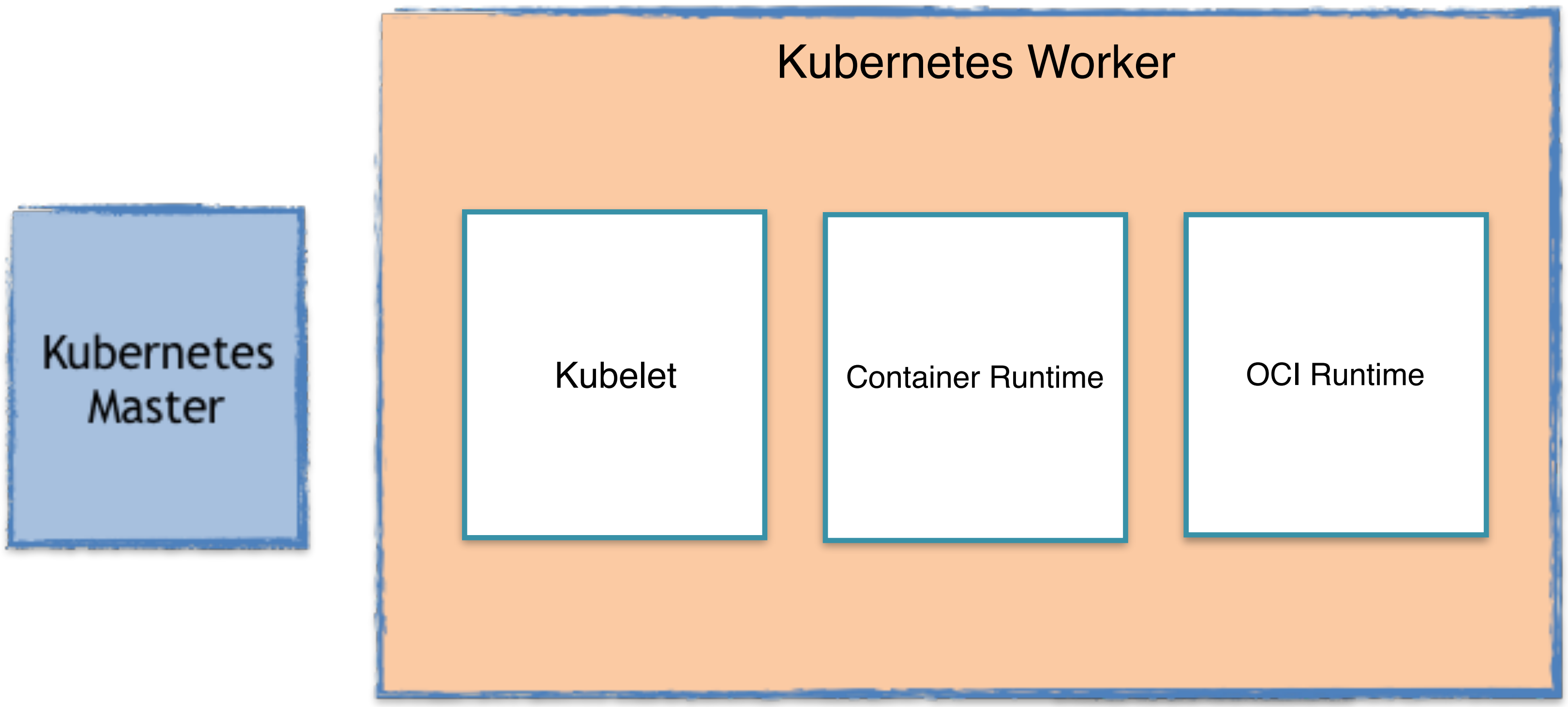
`runc` is default implementation of OCI Runtime Specs

Current Runtime Specs status : v1.0.1

Gap between Kubelet & OCI runtime

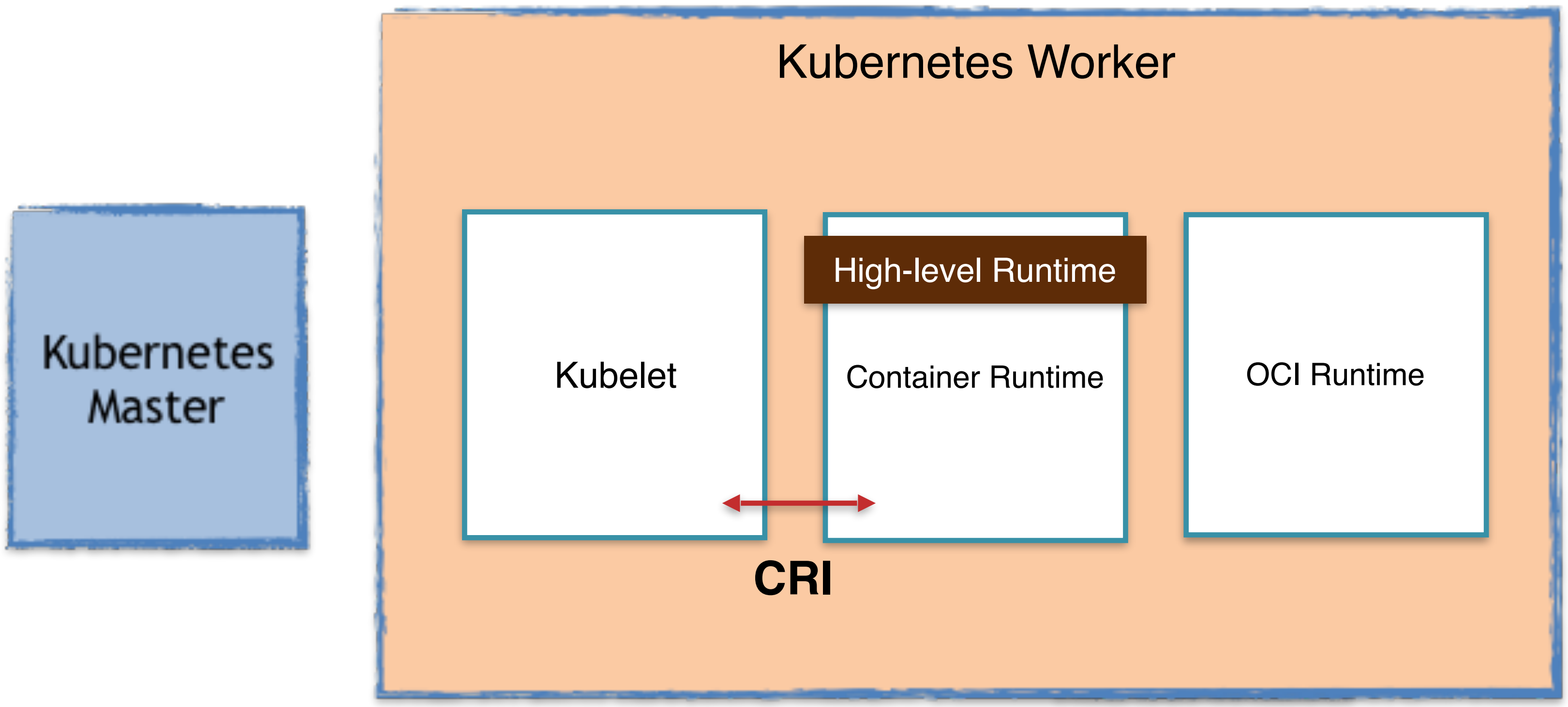
Kubelet Requirements for Runtime	OCI Runtime
Manage images (pull / push / rm ..)	Do not understand concept of image
Talks CRI / gRPC	Input is OCI specs (json and rootfs)
Prepare environment to successfully instantiate container.	Consume the rootfs and container config file (json)
Prepare network for pod	Attach network as pre-start hook.

Runtime in Kubernetes



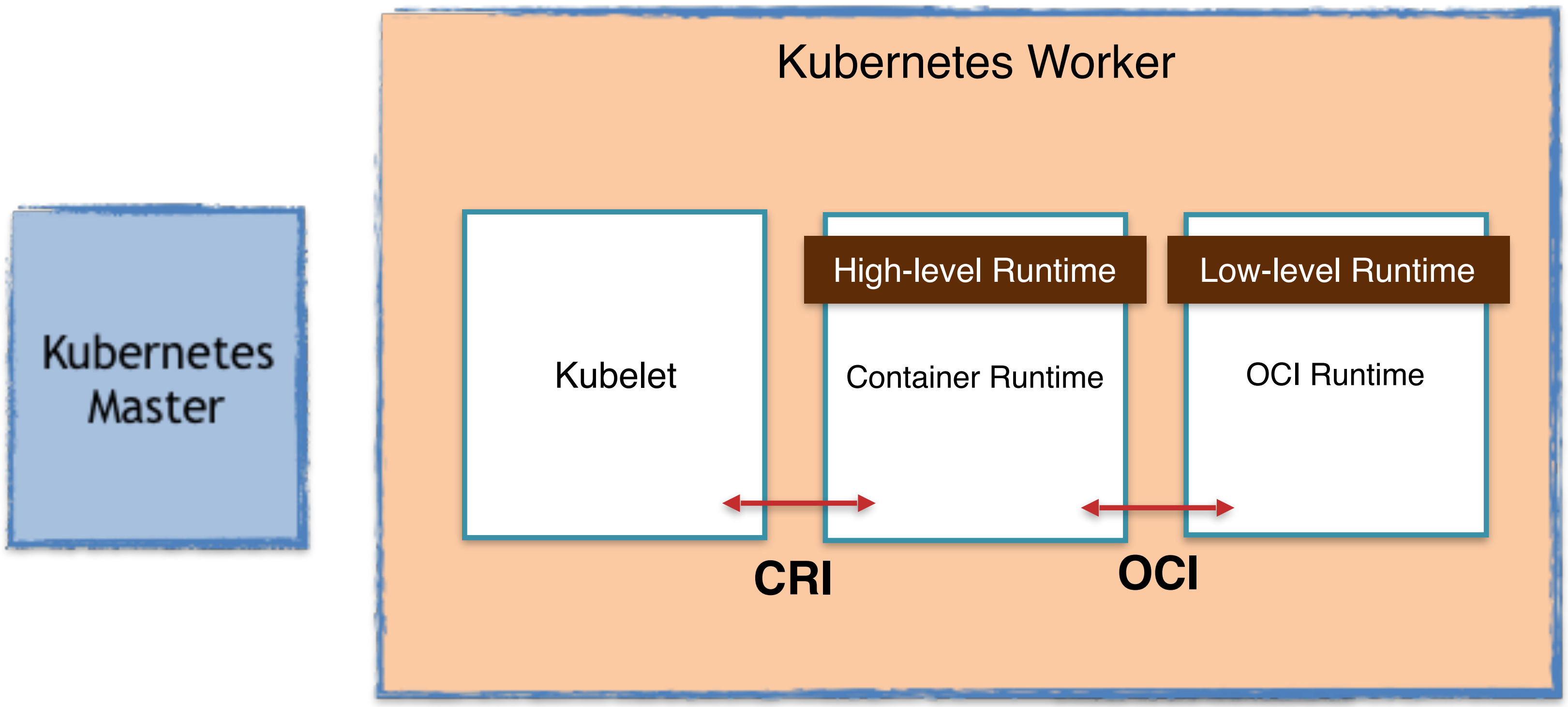
Apart from OCI, *another* **runtime component** is required

Runtime in Kubernetes



- High level runtime implement CRI gRPC services
- Take care of all prerequisite to successfully operate OCI runtimes

Runtime in Kubernetes

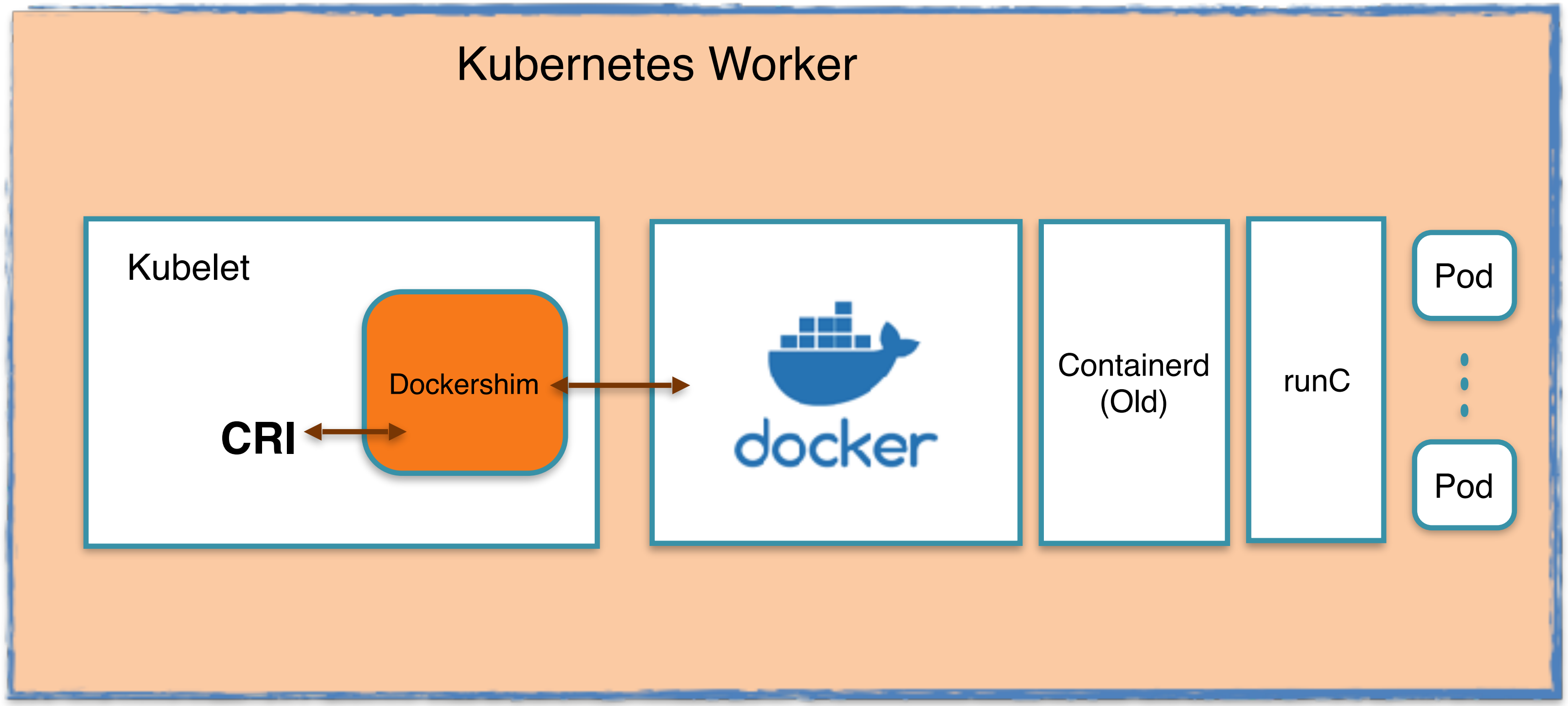


- OCI runtime works as low-level runtime
- High-level runtime provides inputs to OCI runtime as per OCI Specs

CRI Implementations

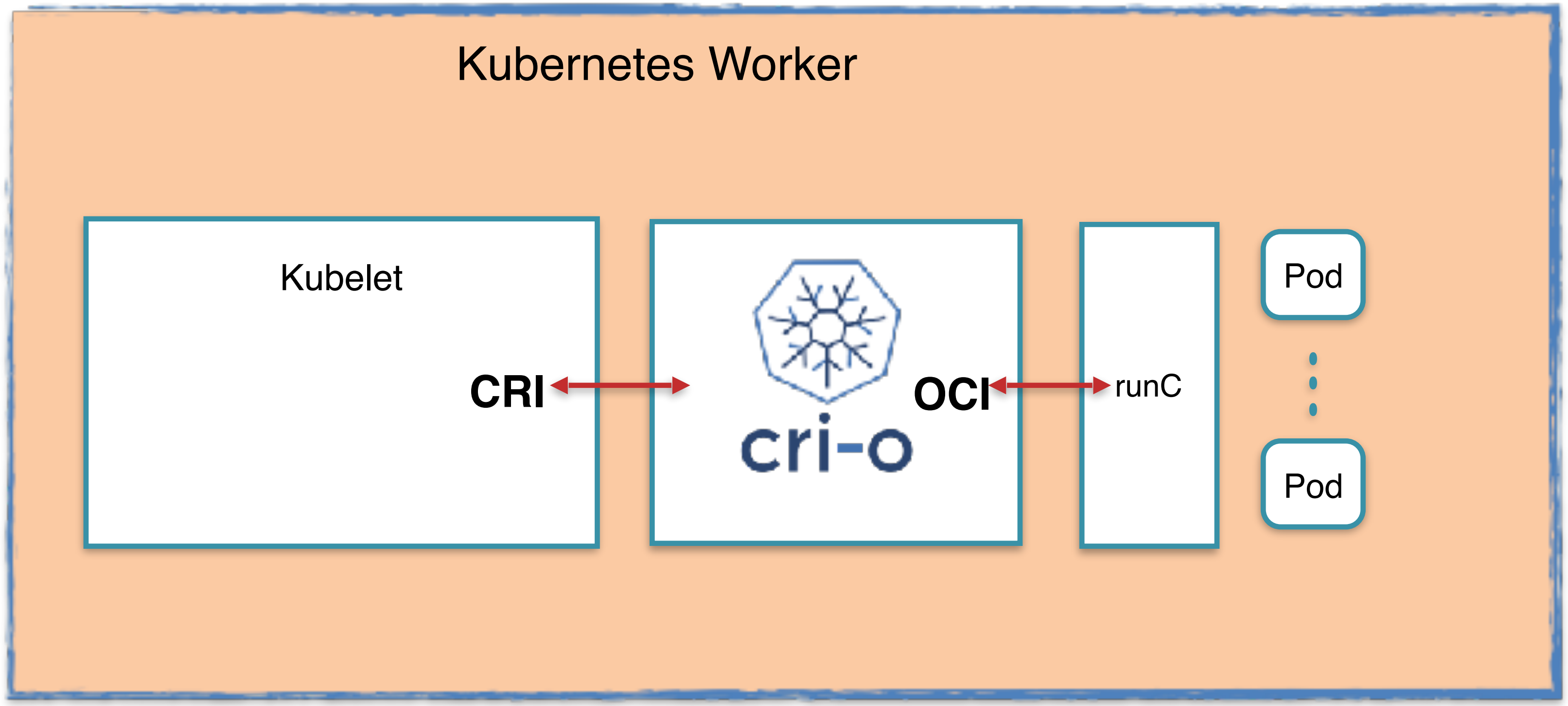
- Dockershim
- CRI-O
- Containerd
- Frakti
- rktlet

Dockershim



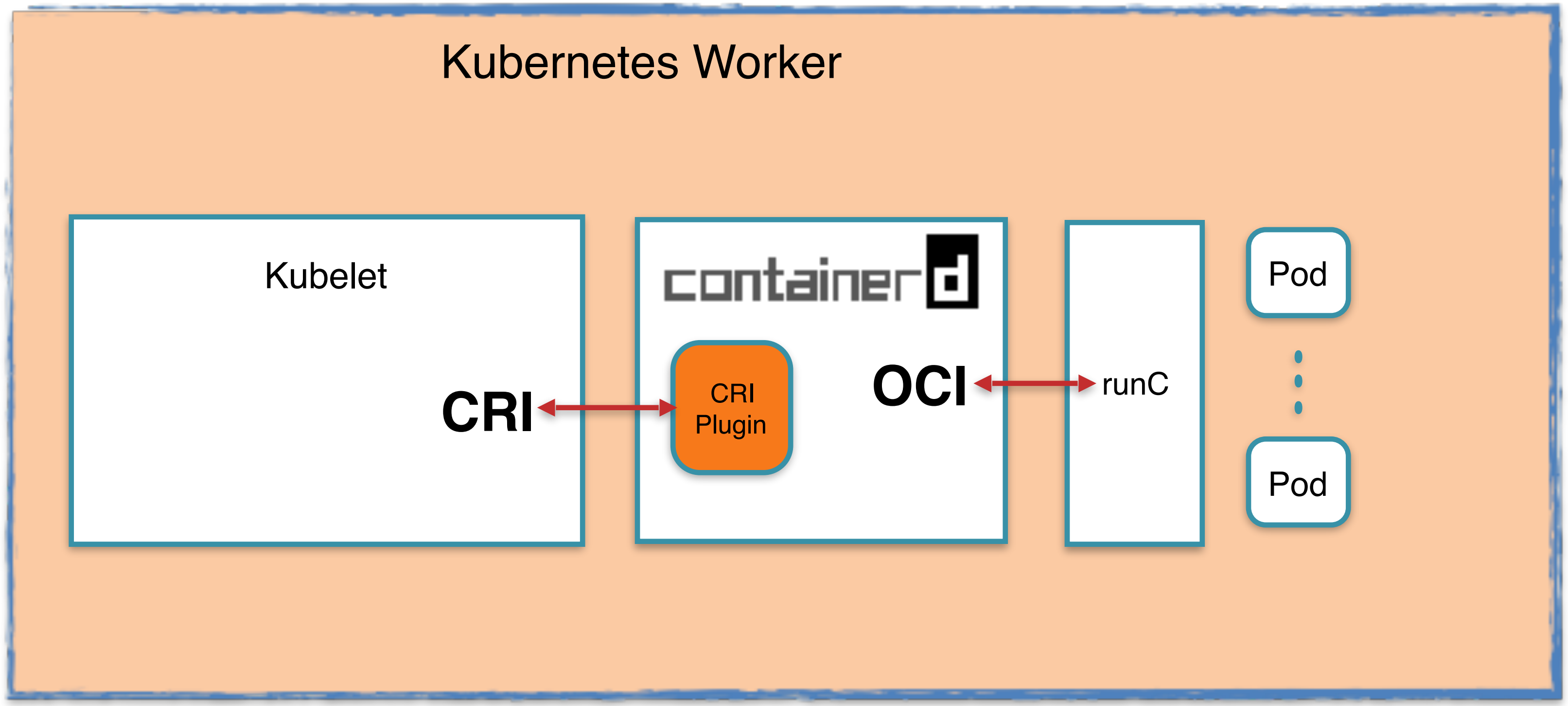
- Embedded into Kubelet.
- Dockershim talks to docker, which manage pods.
- Default CRI implementation & enjoy majority in current kubernetes deployments

CRI-O



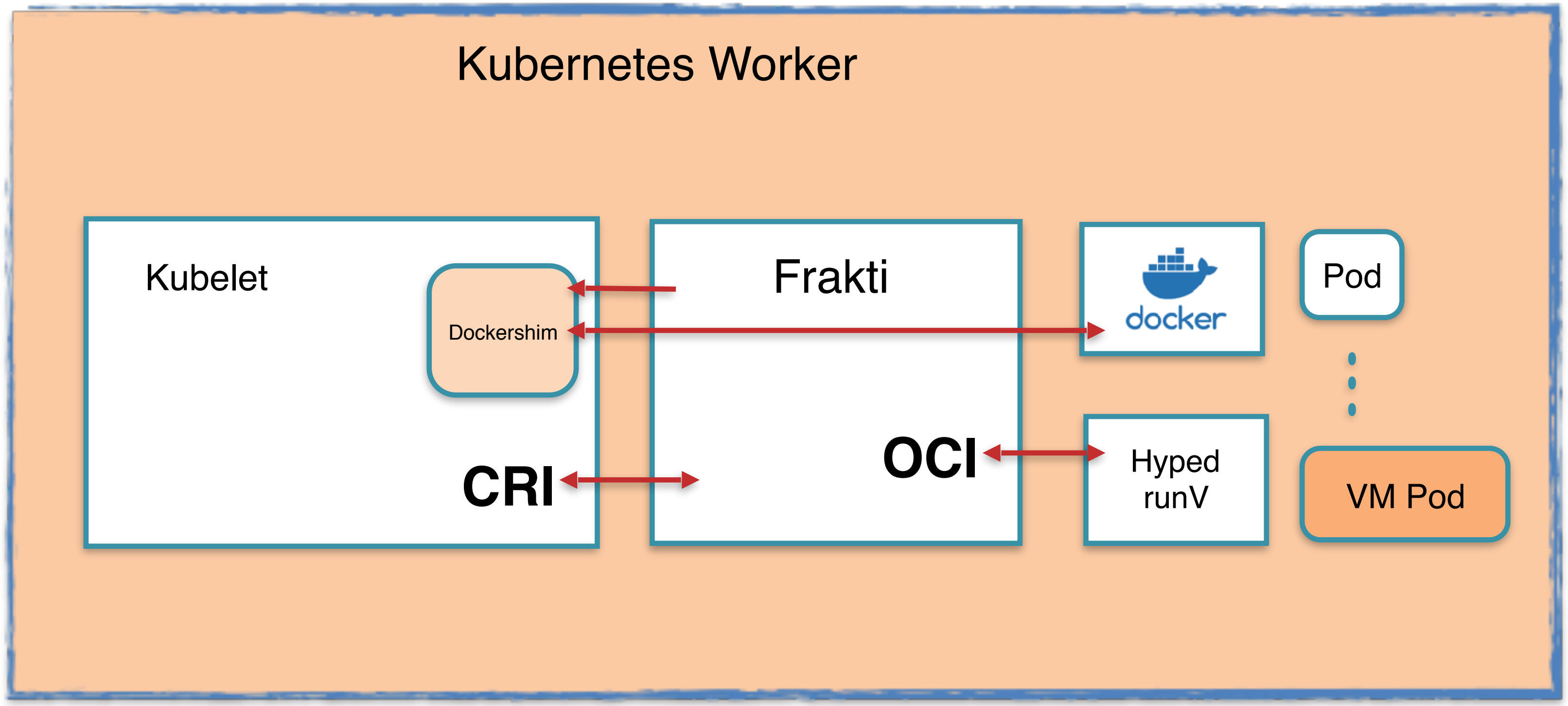
- CRI-O reduces the one extra hop from docker.
- CRI-O uses CNI for providing networking to pods.
- Monolithic design (understands CRI and outputs OCI compatible)
- Works with all OCI runtimes.

containerD



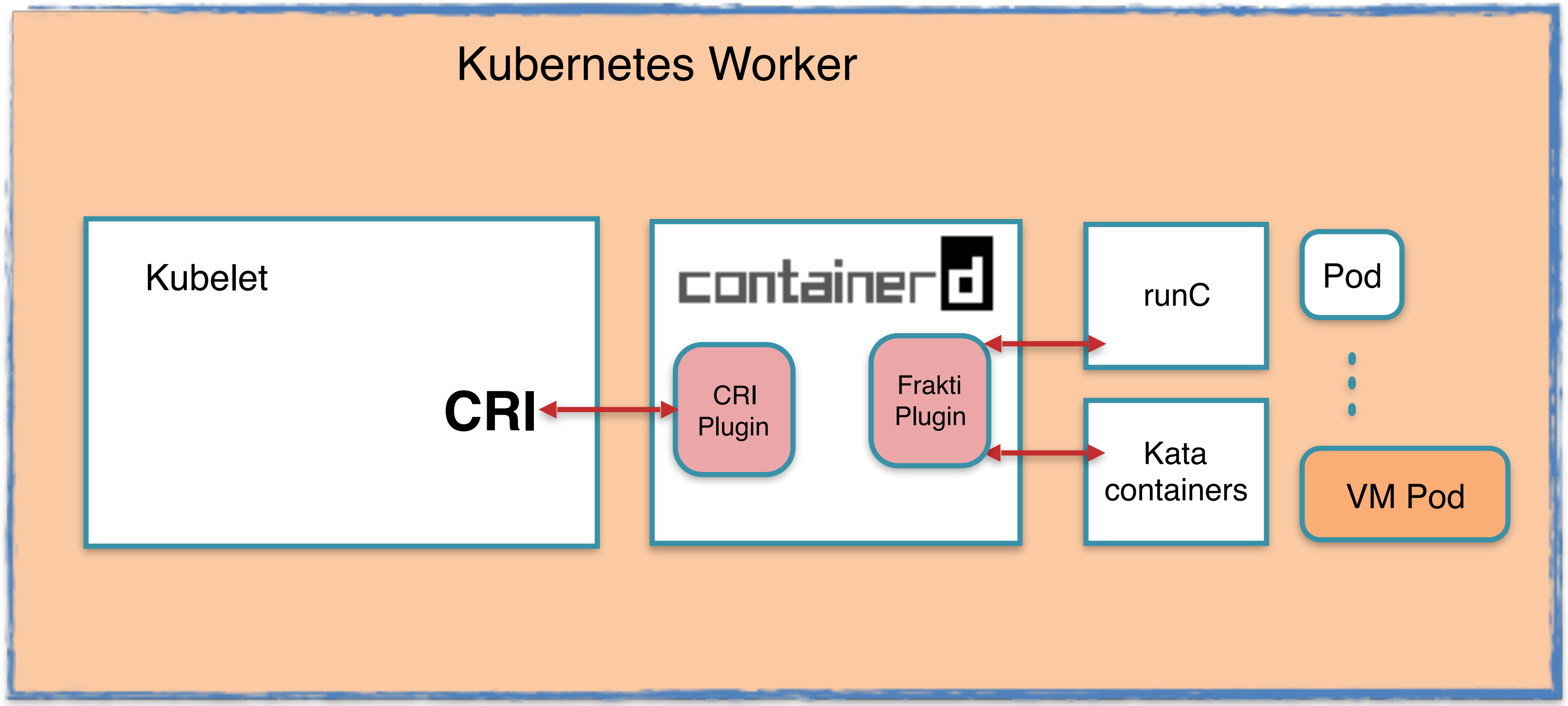
- containerD, with revised scope eliminates the extra hop required by docker.
- Redesigned storage drivers for simplicity and better performance.
- Extensible design, CRI service runs as plugin.
- Uses CNI for networking
- Works with all OCI runtimes.

Frakti



- Frakti runtime was designed to support VM based runtime to kubernetes.
- It supports mixed runtimes
  - Linux containers for privilege containers and runV containers for rest
  - Though uses dockershim to use linux containers, result into extra hops
- Also supports Unikernels

Frakti v2- Coming soon



- Frakti v2 will be implemented as runtime plugin for containerD.
- Reduce extra hops and implementation effort too.

OCI Runtimes

runC	- Default OCI specs implementation - Isolation based on Namespace, cgroups, secomp & MAC (AppArmor, SELinux)
runV	
Clear Containers	
kata-runtime	
gVisor	

OCI Runtimes

runC	- Default OCI specs implementation - Isolation based on Namespace, cgroups, secomp & MAC (AppArmor, SELinux)
runV	- OCI compliant VM based runtime - Uses optimized qemu & KVM. - A light weight guest kernel is used.
Clear Containers	
kata-runtime	
gVisor	

OCI Runtimes

runC	- Default OCI specs implementation - Isolation based on Namespace, cgroups, secomp & MAC (AppArmor, SELinux)
runV	- OCI compliant VM based runtime - Uses qemu & KVM. - A light weight guest kernel is used.
Clear Containers	- Hardware-virtualized containers using Intel’s VT-x - Utilize DAX “direct access” feature of 4.0 kernel
kata-runtime	
gVisor	

OCI Runtimes

runC	- Default OCI specs implementation - Isolation based on Namespace, cgroups, secomp & MAC (AppArmor, SELinux)
runV	- OCI compliant VM based runtime - Uses qemu & KVM. - A light weight guest kernel is used.
Clear Containers	- Hardware-virtualized containers using Intel’s VT-x - Utilize DAX “direct access” feature of 4.0 kernel
kata-runtime	- Best of runV & cc-containers - 1.0 Release (22nd May, 2018) - Under active development
gVisor	

OCI Runtimes

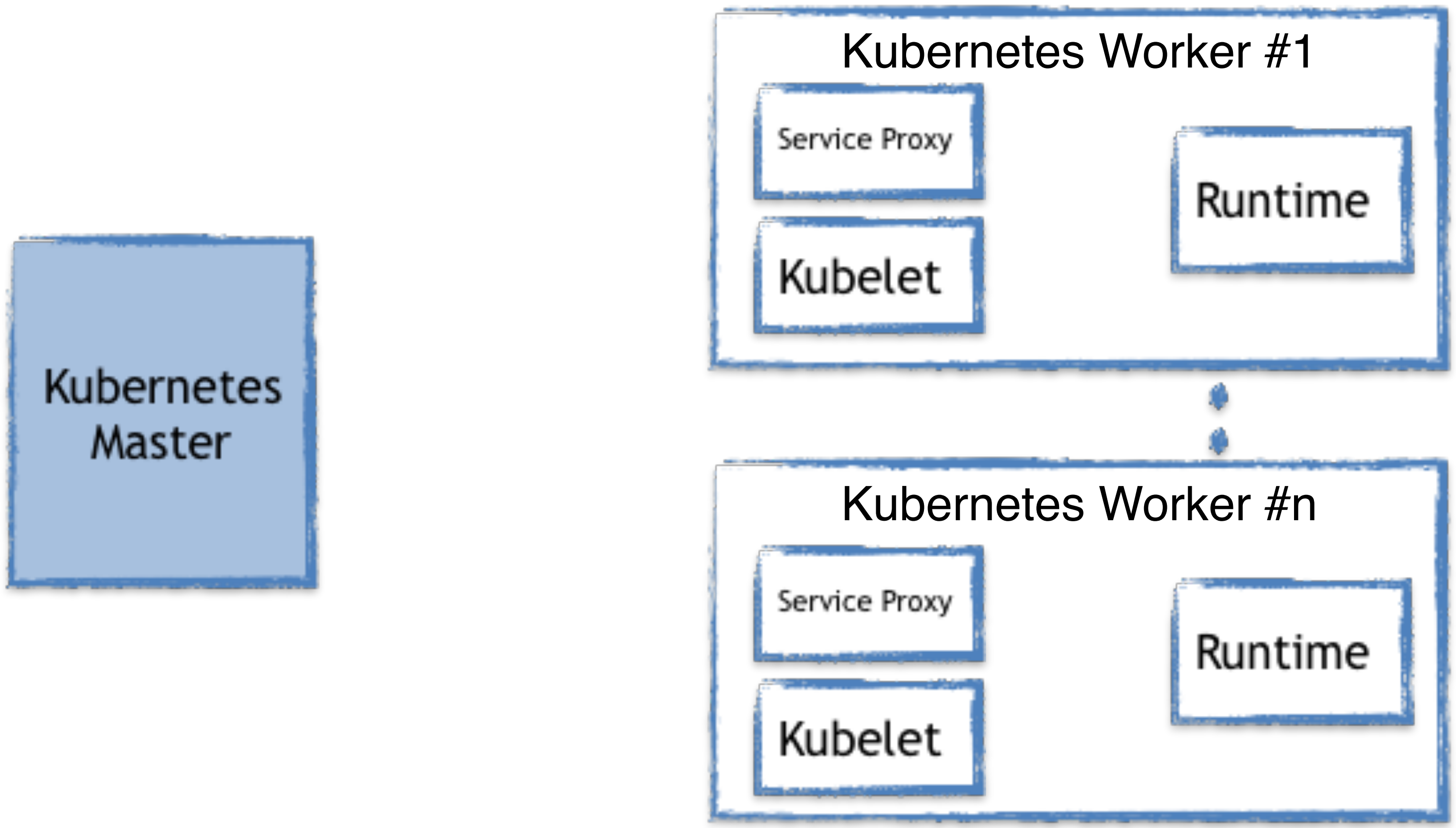
runC	- Default OCI specs implementation - Isolation based on Namespace, cgroups, secomp & MAC (AppArmor, SELinux)
runV	- OCI compliant VM based runtime - Uses qemu & KVM. - A light weight guest kernel is used.
Clear Containers	- Hardware-virtualized containers using Intel's VT-x - Utilize DAX "direct access" feature of 4.0 kernel
kata-runtime	- Best of runV & cc-containers - 1.0 Release (22nd May, 2018) - Under active development
gVisor	- Sandbox based containers - Intercepts application system call acts like kernel. - similar approach as User Mode Linux (UML) - Under active development

Final candidates for Evaluation

High-level Runtime	Low-level Runtime
Dockershim	runC
CRI-O	Kata containers (runV, clear containers)
containerD	

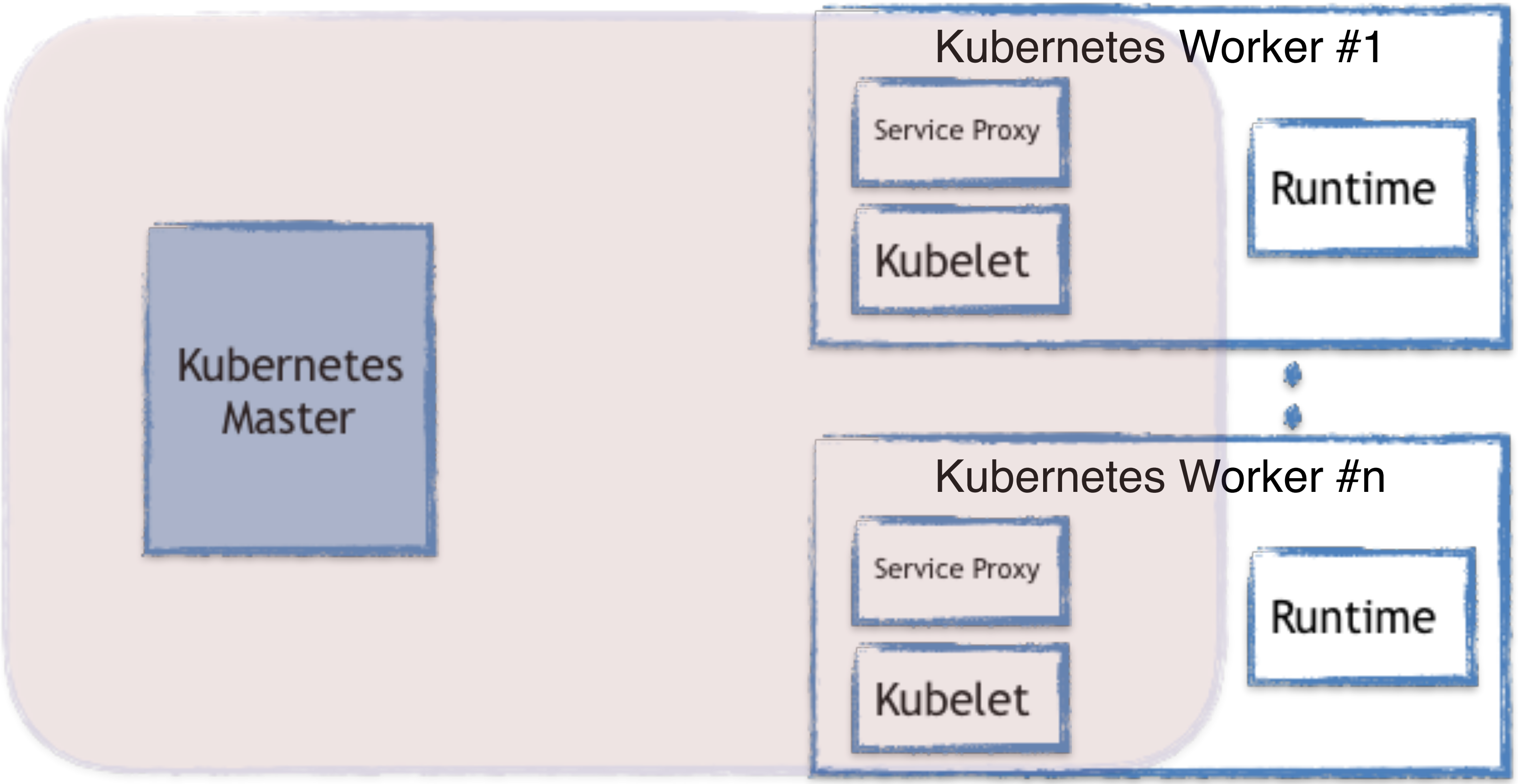
Why runtimes affect kubernetes

Kubernetes Architecture



- Kubernetes offers variety of choices to tune the system

Kubernetes Architecture



- Kubernetes offers variety of choices to tune the system
- Once rest of components finalized
  - for deployment and management runtime is only variable factor.
  - For application performance only low level runtime matters.

Performance benchmarking

Application deployment performance

- Container operations (Create, start, stop, remove)

Application Performance

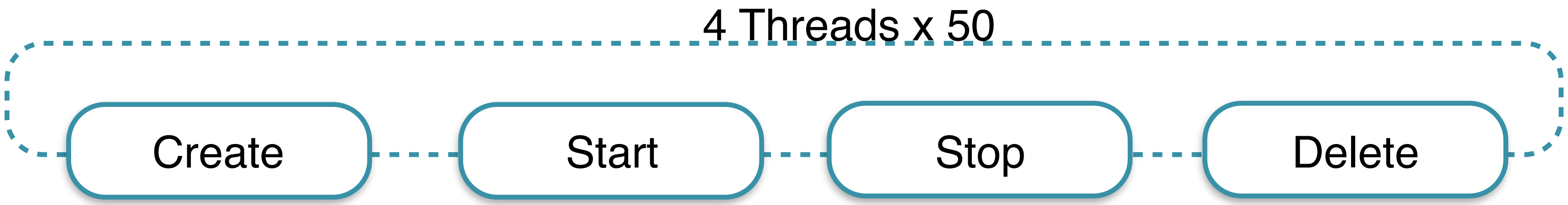
- Containerization / Virtualization overhead.

Performance benchmarking process

- Prerequisite :
  - Pull Sandbox Image
  - Pull Container Image (ubuntu:latest)

Benchmark Environment

Architecture:	x86_64
CPU(s):	8
Core(s) per socket:	4
Model name:	i7-3630QM CPU @ 2.40GHz
Virtualization:	VT-x
Kernel :	linux 4.15
OS :	Ubuntu



Create
- Create & Run PodSandbox
- Create Application Container

Start
- Start Application Container

Stop
- Stop Application Container

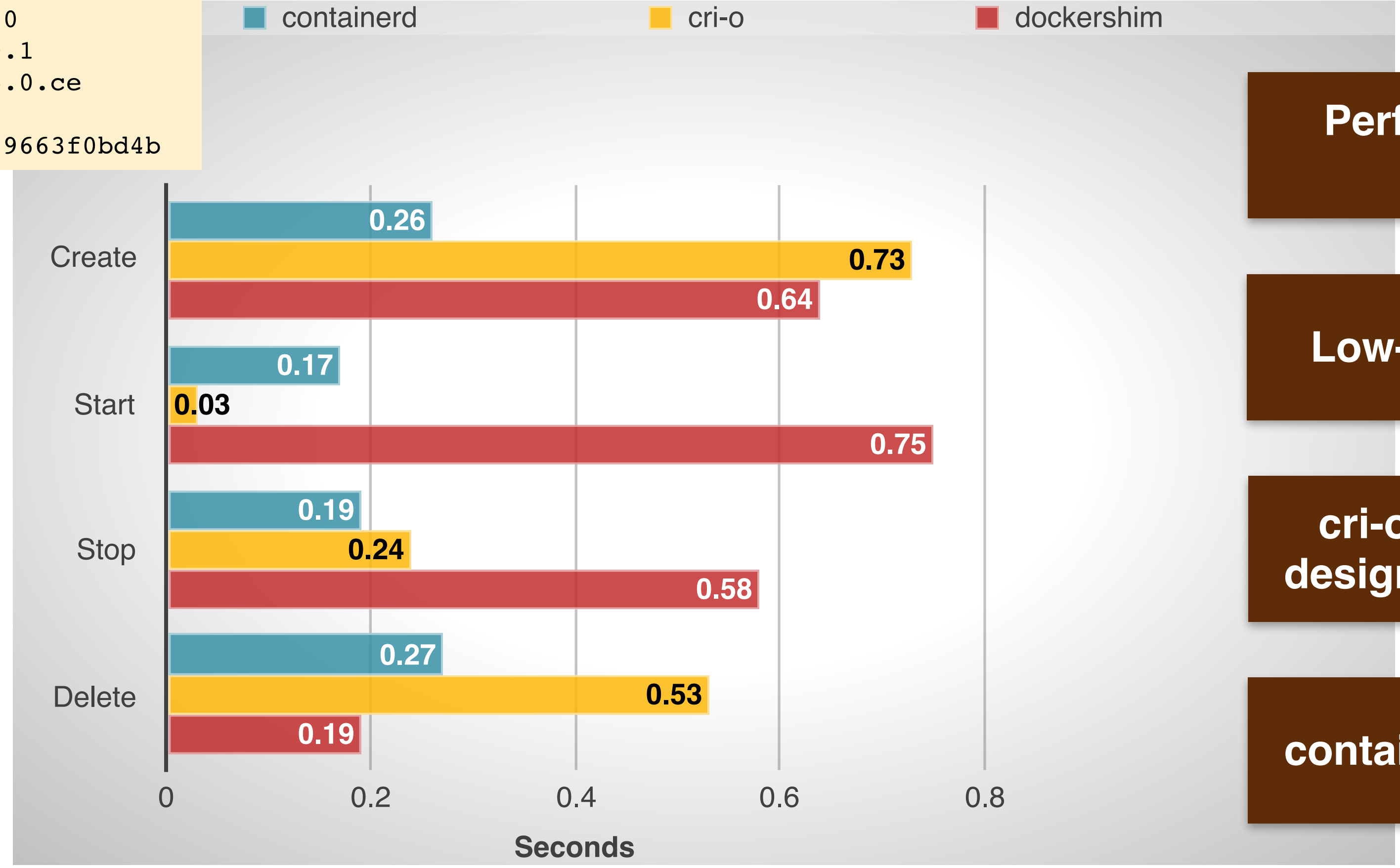
Delete
- Delete Application Container
- Stop PodSandbox
- Delete PodSandbox

- Rootfs prepared from Image
- Writable area for container
- CNI plugin invocation for Network

runC Performance

```
Software versions

Containerd : v1.1.0
cri-o      : v1.10.1
Docker     : 18.05.0.ce
Runc       : v1.0
            git #69663f0bd4b
```



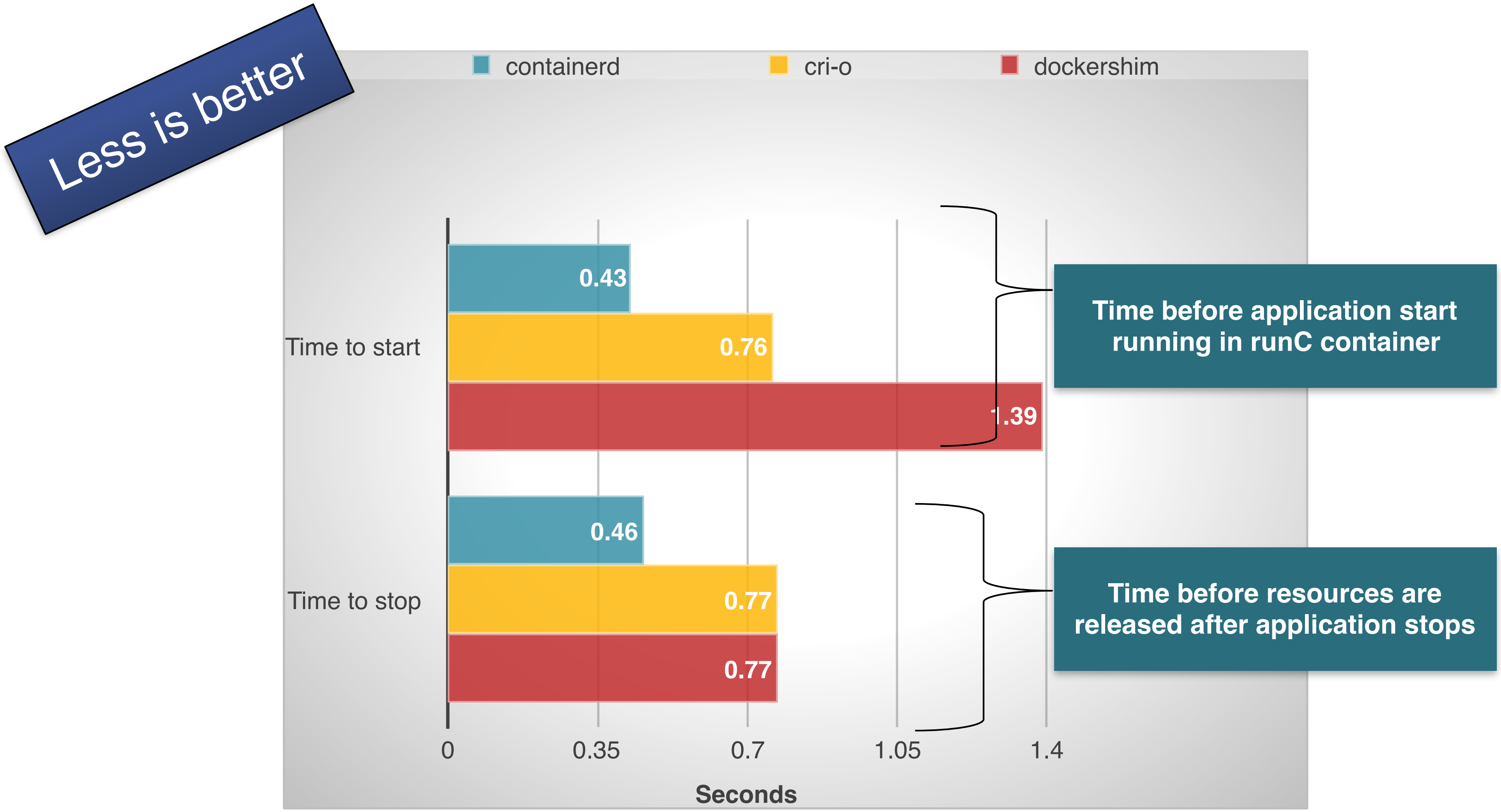
Performance difference due to high level runtime

Low-level runtime (runC) is constant in all

cri-o and docker share same graph driver design, could be reason for high create time.

containerD perform better in almost all case.

Latency with runC



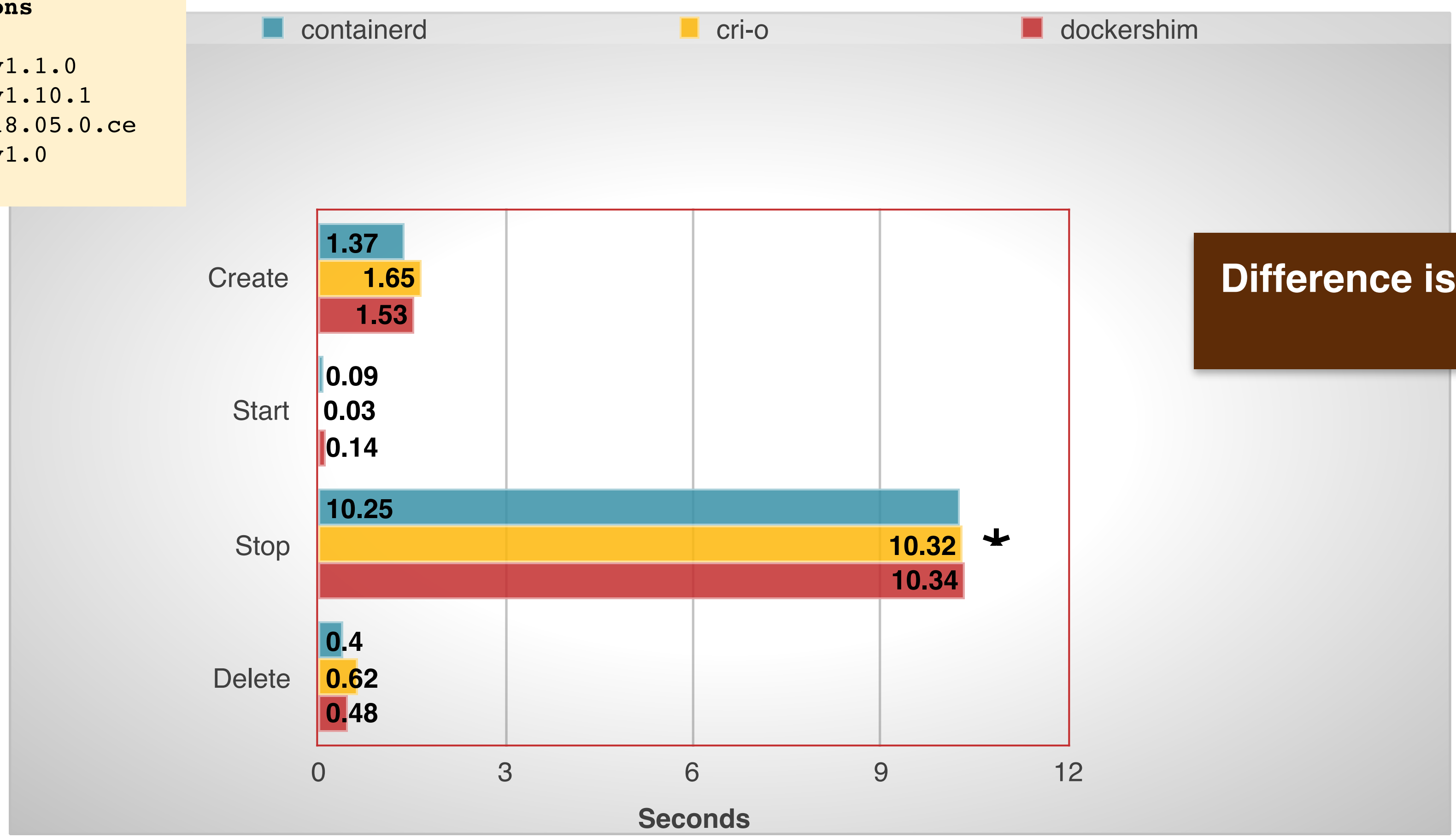
cri-o & containerD both perform better than docker

In performance, containerD performs better than cri-o

Kata-runtime Performance

```
Software versions

Containerd  : v1.1.0
cri-o       : v1.10.1
Docker      : 18.05.0.ce
kata-runtime: v1.0
```



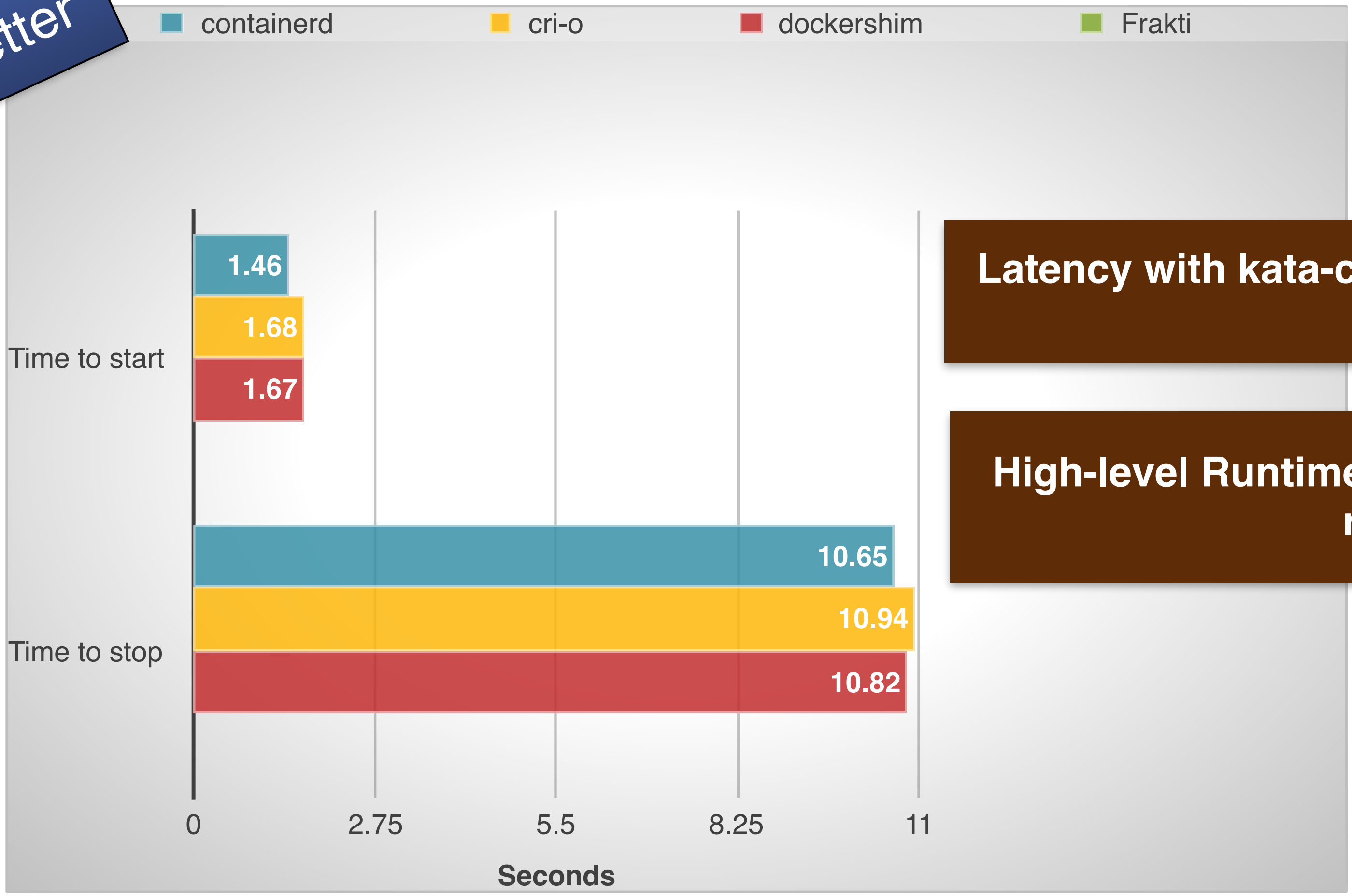
Difference is mainly due to high level runtime performance.

* - Bug in Stop logic, while invoked through CRI

- Takes < 2 seconds, if done directly through docker or containerD

Latency with Kata

Less is better



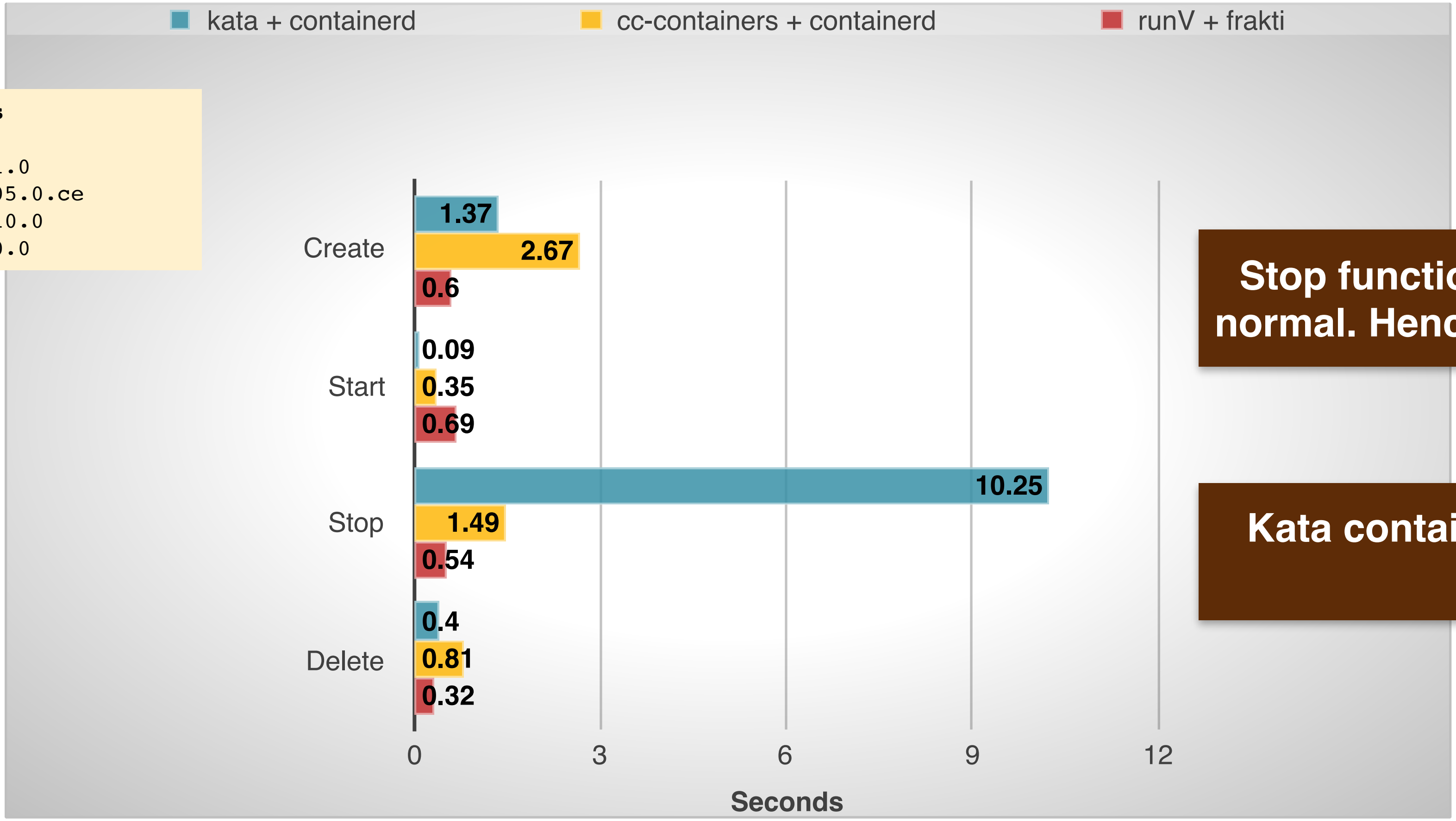
Latency with kata-container is comparable with all high-level runtimes.

High-level Runtime don't make much difference if low-level runtime consume most

kata vs runV vs clear-containers

Software versions

```
Containerd : v1.1.0
Docker     : 18.05.0.ce
Frakti     : v1.10.0
runV       : v1.0.0
```



Stop function of cc-containers & runV looks normal. Hence fix required for kata containers.

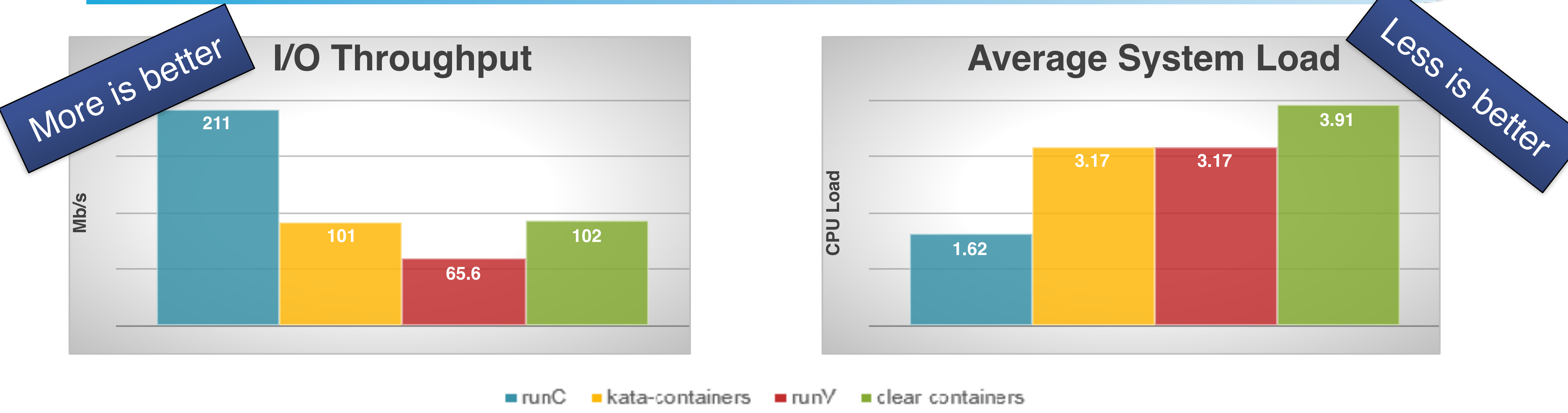
Kata containers performance is in-between runV and cc-runtime.

Latency with VM based runtimes

runV performs for container operations is best in VM containers.

Kata is still in active development

Performance Overhead – Low-level runtimes



Runtime performance overhead affect application running inside container.

runC perform best in both IO throughput and average CPU load.

kata-containers perform best among VM containers.

Workloads

Serverless

- **Host functions instead of applications?**
  - Functions as service
  - e.g. AWS Lambda
- **Ideal Platform**
  - Low latency
  - High parallelism i.e. high density.
  - Low on resources (CPU, Memory)

Serverless platform

	containerd + runC	cri-o + runC	Frakti + runV	Any + kata-containers
Latency	Best	Better	Good	Average
Cold start	Best	Better	Better	Average
Warm start	Better	Best	Average	Good
Density	Best	Good	Average	Average
Security	Good (namespace + seccomp + SELinux)	Good (namespace + seccomp + SELinux)	Best (VM based)	Best (VM Based)
Stability	Stable	Stable/Best with Openshift	Stable	Under Active development
Support Cycle	(defined support cycle for each release)	(Not defined)	(managed by hyper.sh) (not defined)	(Not defined)

Peak hour demand / Micro Services

- **Mostly applications are of type Micro services.**
- **Ideally immutable**
- **Quick scale up and scale down.**
- **Ideal Platform**
  - Low latency for start application and free resources.
  - Better utilize the host system.

Mean Time To Recover (MTTR) - DevOps

- Short Lived containers
- Frequent updates
- Fast recovery is important.
- Low on resources

Micro-services / MTTR

	containerd + runC	cri-o + runC	Frakti + runV	Any + kata-containers
Latency	Best	Better	Good	Average
Density	Best	Better	Average	Good
Security	Good (namespace + seccomp + SELinux)	Good (namespace + seccomp + SELinux)	Best (VM based)	Best (VM Based)
Stability **Support Cycle**	Stable (defined support cycle for each release)	Stable/Best with Openshift (Not defined)	Stable (managed by hyper.sh) (not defined)	Under Active development

Long running containers

- **Migrated application.**
- **Stateful containers.**
- **Hard to scale containers.**
- **Requirements**
  - Stability
  - Security
  - Performance
  - Migration

Long running containers

	containerd + runC	cri-o + runC	Frakti + runV	Any + kata-containers
Stability **Support Cycle**	Best (defined support cycle for each release)	Stable/Best with Openshift (Not defined)	Good (managed by hyper.sh) (not defined)	Under Active development (not defined)
Security	Good (namespace + seccomp + SELinux)	Good (namespace + seccomp + SELinux)	Best (VM based)	Best (VM Based)
Performance Overhead	Best	Best	Average	Better
Migration	Required	Required	Required	Required
Governance	CNCF + OCI ✔	Kubernetes Incubator + OCI	Kubernetes + hypersh	OpenStack Foundation ✔

Summary

- CRI and OCI enable more choices for container runtimes.
- For Cloud Native workloads, Linux containers based runtimes suite better.
- High level runtime performance do not matter much for long running containers, So low level runtime performance & capabilities become focus.
- VM based runtimes are promising, but still need some time to reach flexibility and usability as Linux containers runtime.
- Migration of monolithic applications / high security applications to modern platform like kubernetes will get boost with VM based runtimes.

Few more OCI runtimes

- **Runtime getting ready for OCI complaint**
  - rkt - container runtime from CoreOS
    - https://github.com/rkt/rkt
    - https://github.com/rkt/rkt/issues/3368
  - gVisor - Sandbox based containerization
    - https://github.com/google/gvisor
  - railcar – linux containers in implementation in rust
    - https://github.com/oracle/railcar
    - slow development
  - crun – linux containers in implementation in C
    - https://github.com/giuseppe/crun
    - Fully featured but lack clarity on maintenance and support.

Thank You