



@nimmerrichterm  
@certitude\_lab





# Kubernetes Security Challenge and Opportunity

# \$whoami

- > Marc Nimmerrichter
  - > Consultant for IT security for the last 10 years
  - > Mainly doing pentesting, appsec, security concepts
  - > Kubernetes pentests and audits, trainings, concepts, etc.
  - >  @nimmerrichterm
- 
- > Certitude is an IT security consulting firm based in Vienna
  - > We do pentesting, security trainings, application security, designing secure systems and infrastructure, etc.
  - >  @certitude\_lab



# \$whoami

- > Marc Nimmerrichter
  - > Consultant for IT security for the last 10 years
  - > Mainly doing pentesting, appsec, security concepts
  - > Kubernetes pentests and audits, trainings, concepts, etc.
  - >  @nimmerrichterm
- 
- > Certitude is an IT security consulting firm based in Vienna
  - > We do pentesting, security trainings, application security, designing secure systems and infrastructure, etc.
  - >  @certitude\_lab



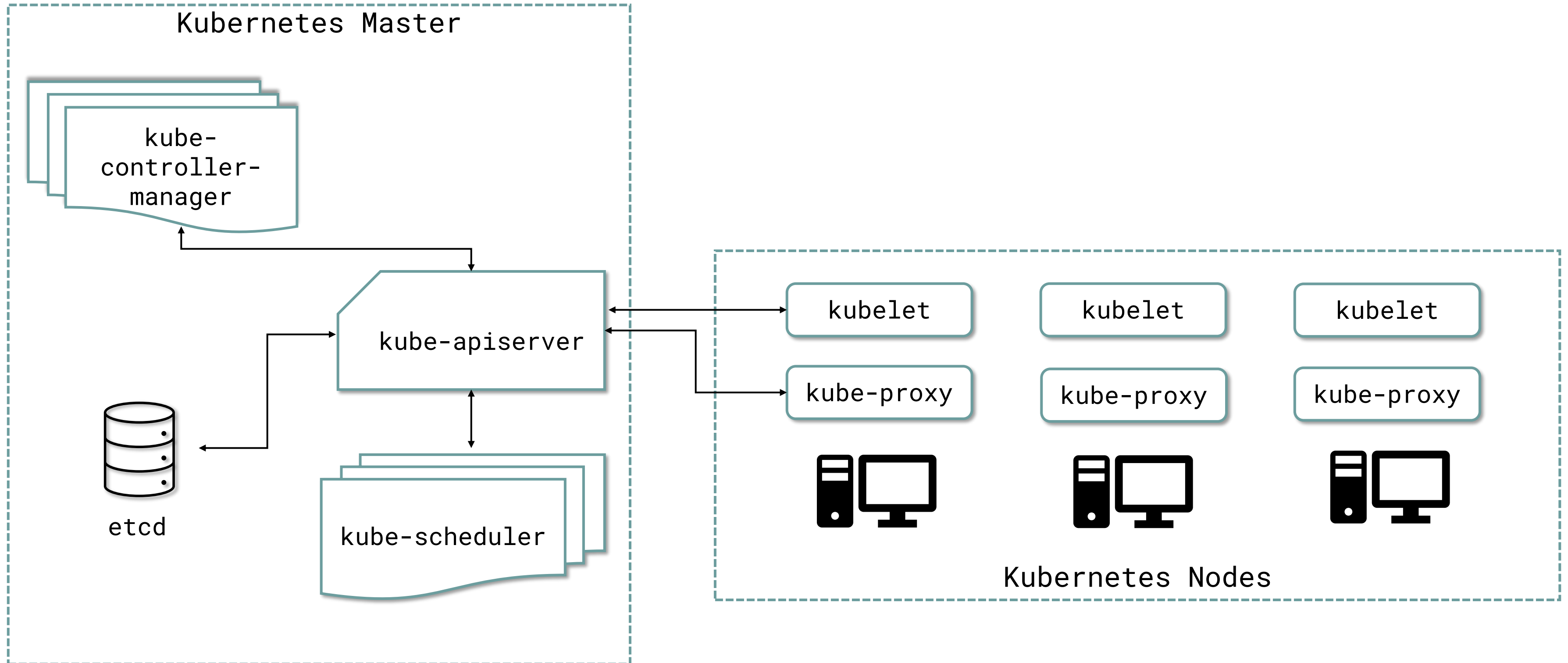
We are hiring!

# Agenda

- > Intro to Kubernetes
- > Container isolation
- > Kubernetes attacks vectors
- > Common vulnerabilities
- > Multi tenancy
- > Opportunities for security

# Kubernetes Overview

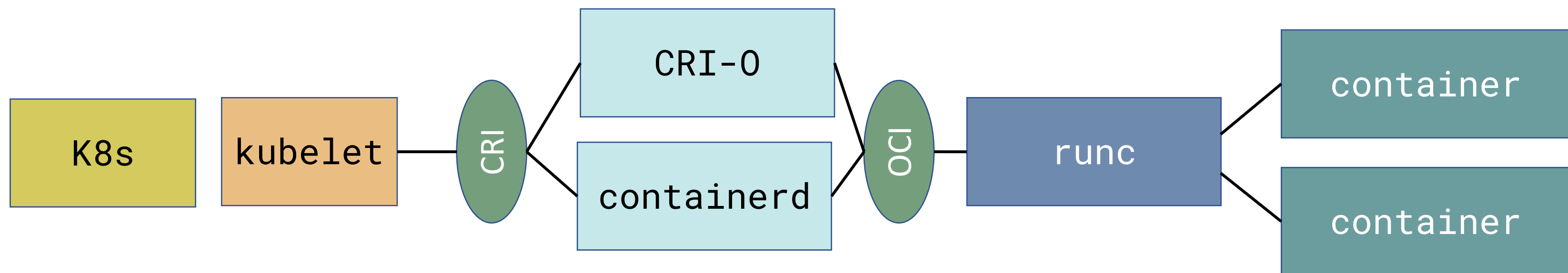
## Components



# Kubernetes Overview

## Kubelet

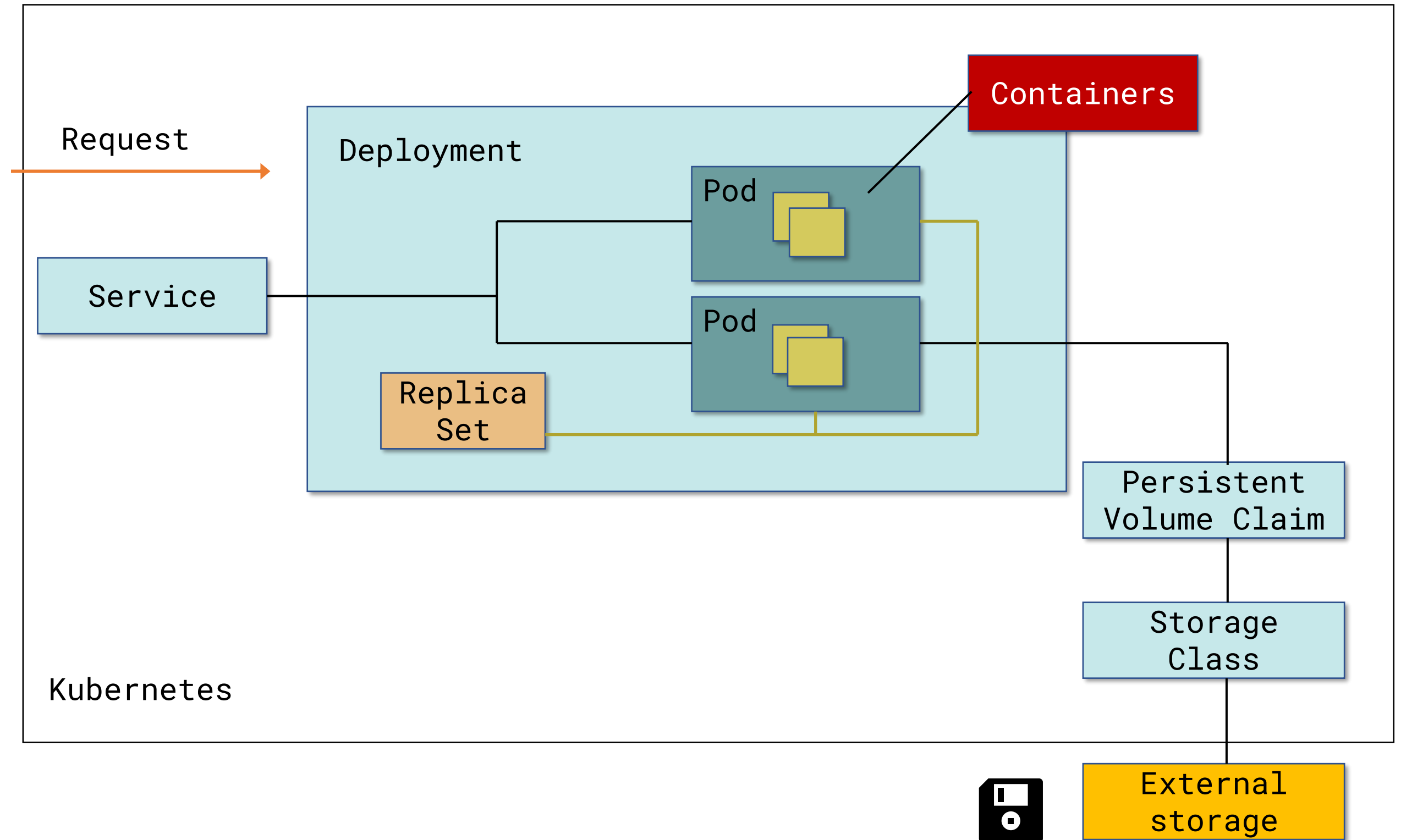
- > Kubelet manages the node and runtime
- > Checks API server for workloads scheduled on the node
- > Manages lifecycle of workloads through API
- > Kubernetes removed dockershim with 1.24



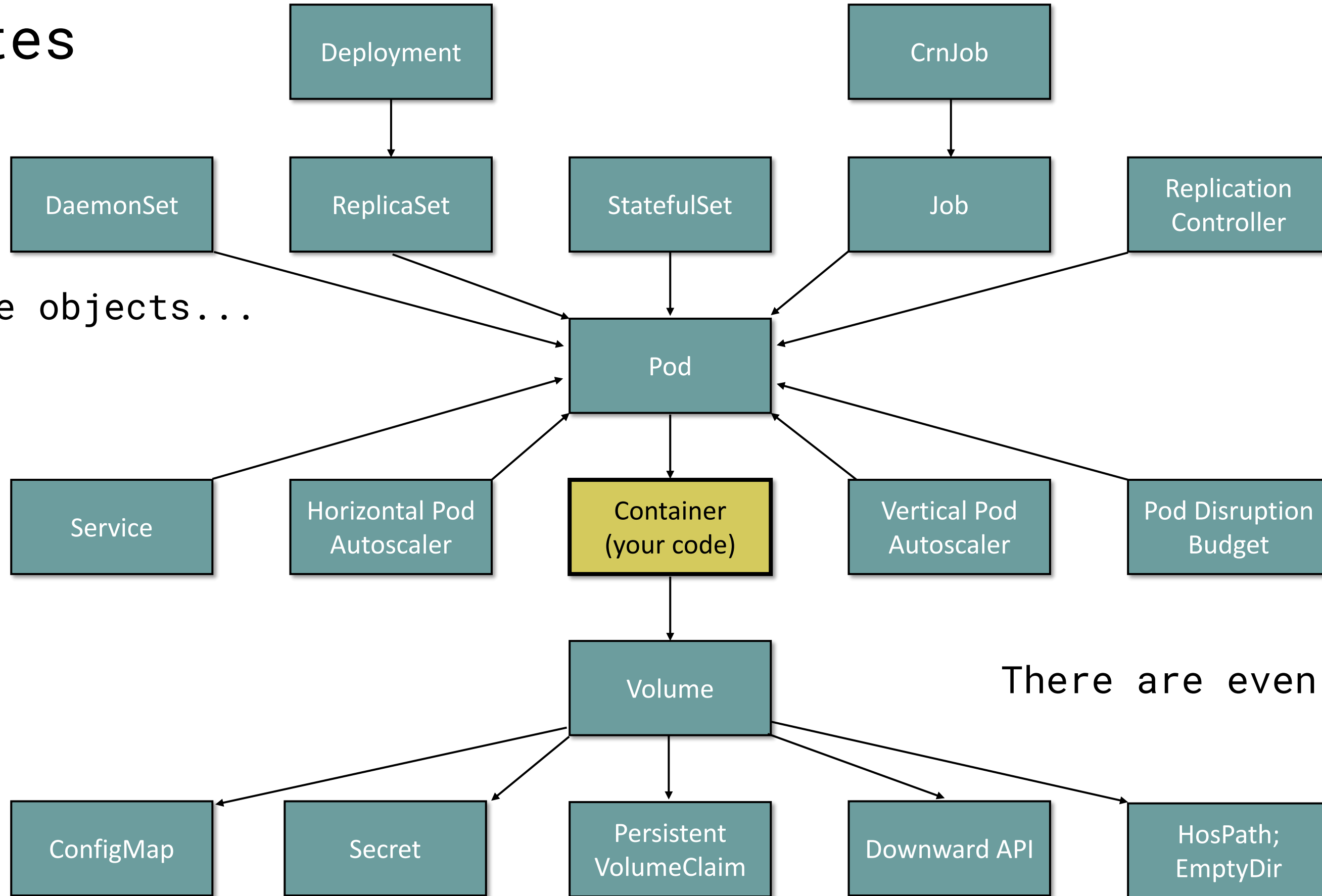
# Kubernetes Overview

## Objects

- > Objects are Kubernetes configurations
- > Service objects routes traffic to pods
- > Deployment object allows to update pods, replicaset etc.
- > Replicaset object scales pods
- > Careful:
  - ◇ Traffic does not actually go through a service or deployment -> it's just a configuration object



# Kubernetes Objects



> A few more objects...

There are even more..



# Kubernetes Overview

## Pod & service objects

**apiVersion:** v1

**kind:** Pod

**metadata:**

**name:** nginx-pod

**labels:**

**app:** nginx

**spec:**

**containers:**

- **name:** nginx-cont

**image:** nginx:1.14.2

**ports:**

- **containerPort:** 9376

**apiVersion:** v1

**kind:** Service

**metadata:**

**name:** nginx-service

**spec:**

**selector:**

**app:** nginx

**ports:**

- **protocol:** TCP

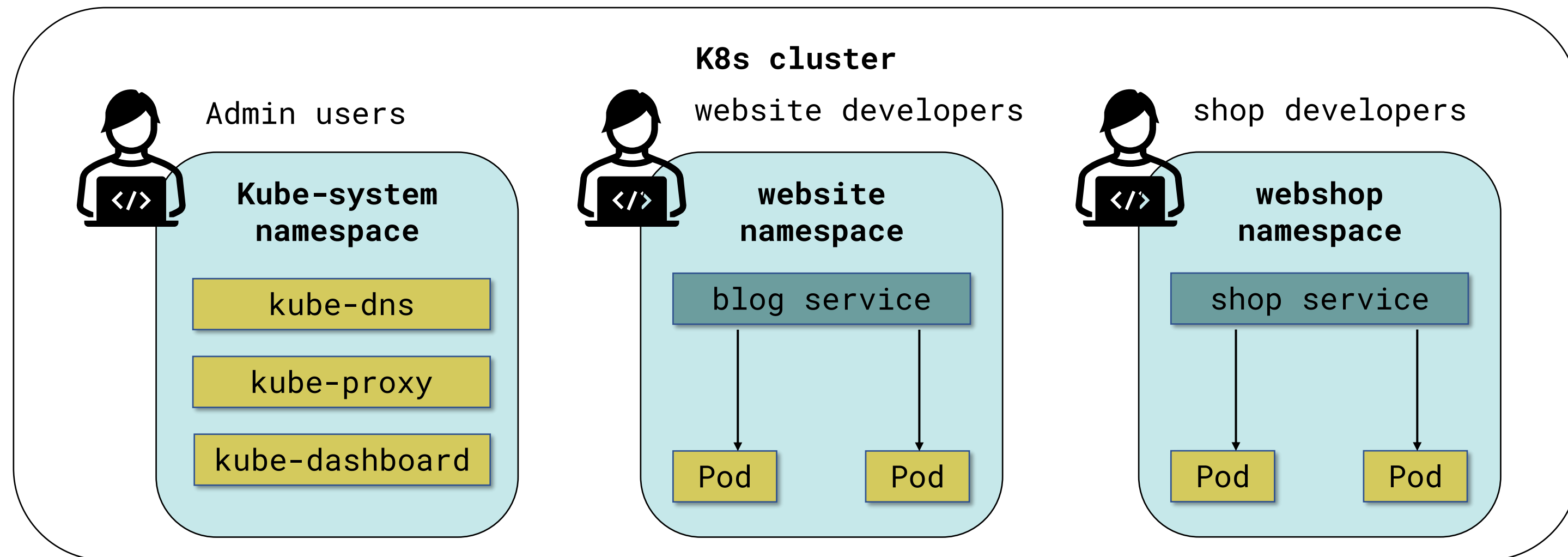
**port:** 80

**targetPort:** 9376

# Kubernetes Overview

## Namespace objects

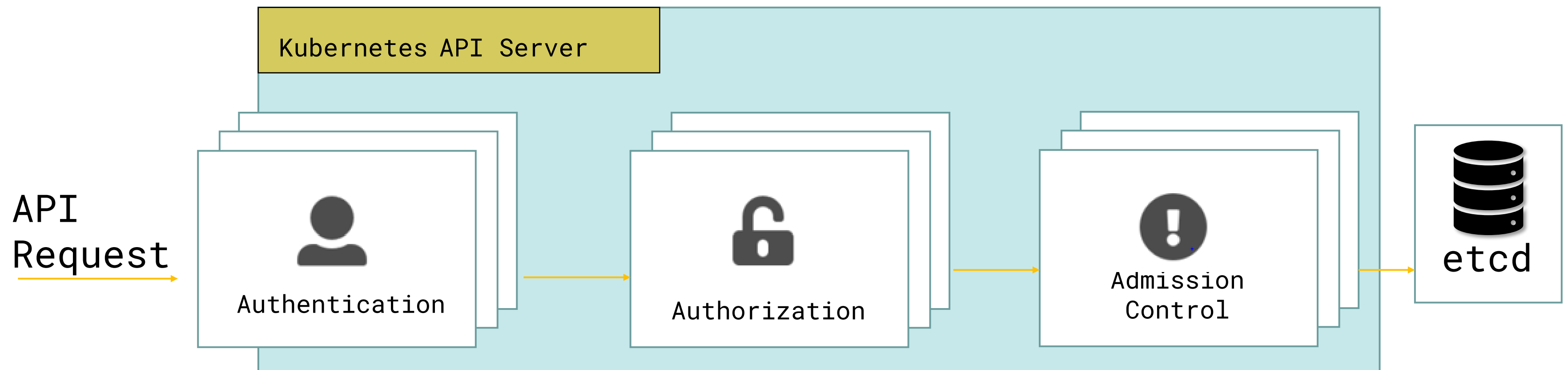
- > K8s objects are organized in namespaces (which is an object too)
- > The kube-system namespace contains the control plane
- > Some k8s objects are “namespaced” (belong to one namespace), others are not (belong to whole cluster)



# Kubernetes Overview

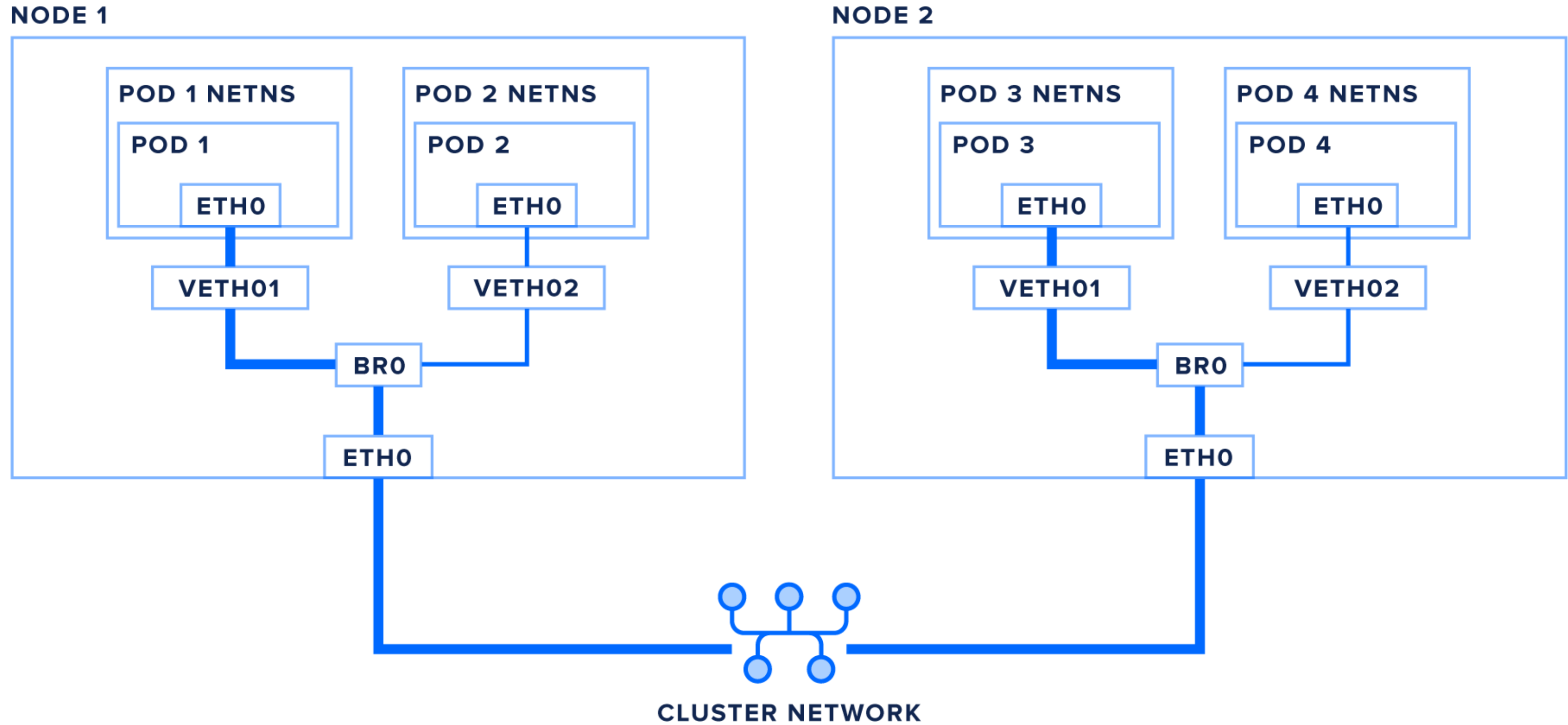
## Authentication, Authorization, Admission Control

- > The authentication module checks who the user is
- > The RBAC module checks which CRUD operations are allowed on which objects
- > Admission control allows to inspect object configurations
  - ◇ There are some default admission controllers, but you can also do custom ones



# Kubernetes Overview

## Networking



# Kubernetes Overview

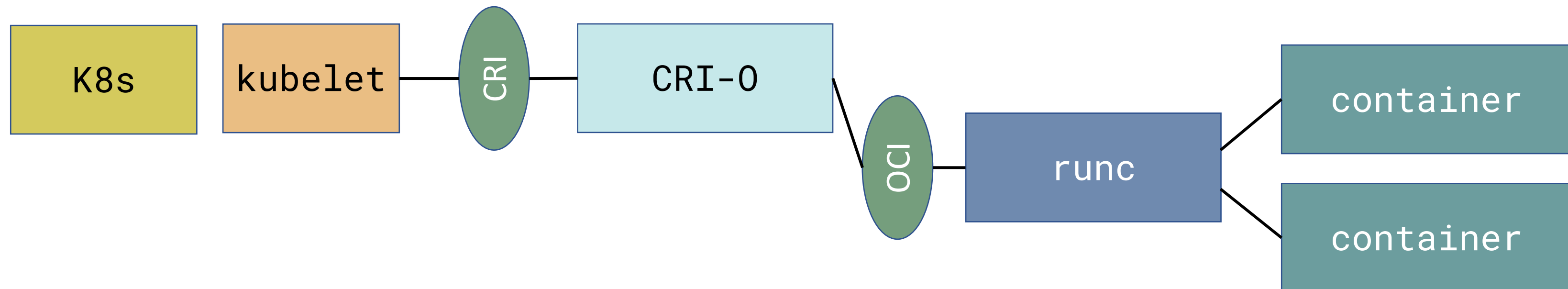
## CNI

- > CNI standardizes how orchestrators connect containers to a network
- > Enables pod-to-pod communication
- > CNI plug-in assigns interfaces and IPs to pods and does IP address management
- > To not depend on a particular container, network interface is configured for dummy „pause“ aka „sandbox“ container
- > Container runtime (e.g. containerd) calls CNI plug-in executable (e.g. Calico) to add an interface to container's networking namespace (sandbox/pause container)

# Kubernetes Overview

## CNI

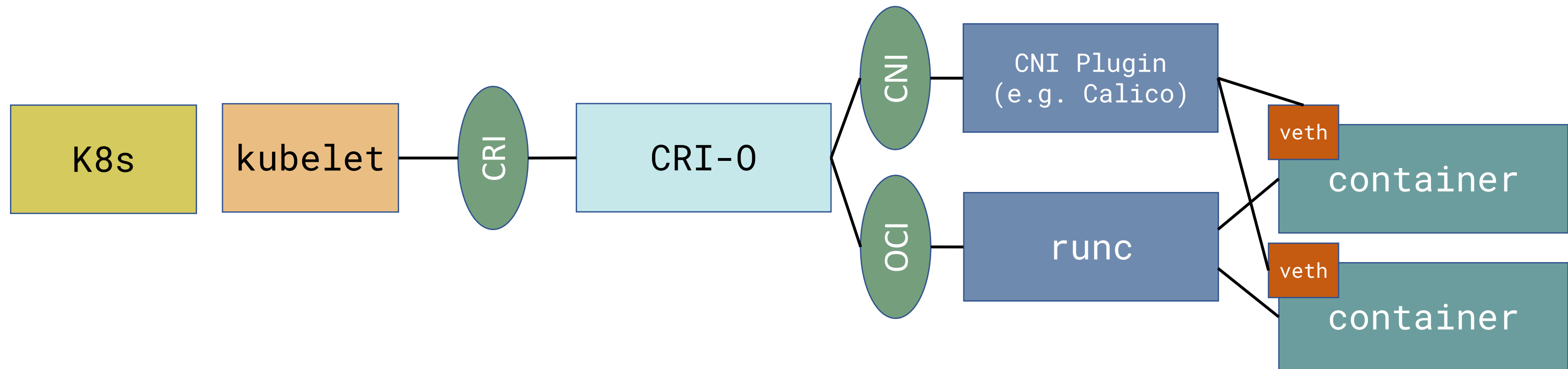
- > CNI standardizes how orchestrators connect containers to a network
- > Enables pod-to-pod communication
- > CNI plug-in assigns interfaces and IPs to pods and does IP address management
- > To not depend on a particular container, network interface is configured for dummy „pause“ aka „sandbox“ container
- > Container runtime (e.g. containerd) calls CNI plug-in executable (e.g. Calico) to add an interface to container's networking namespace (sandbox/pause container)



# Kubernetes Overview

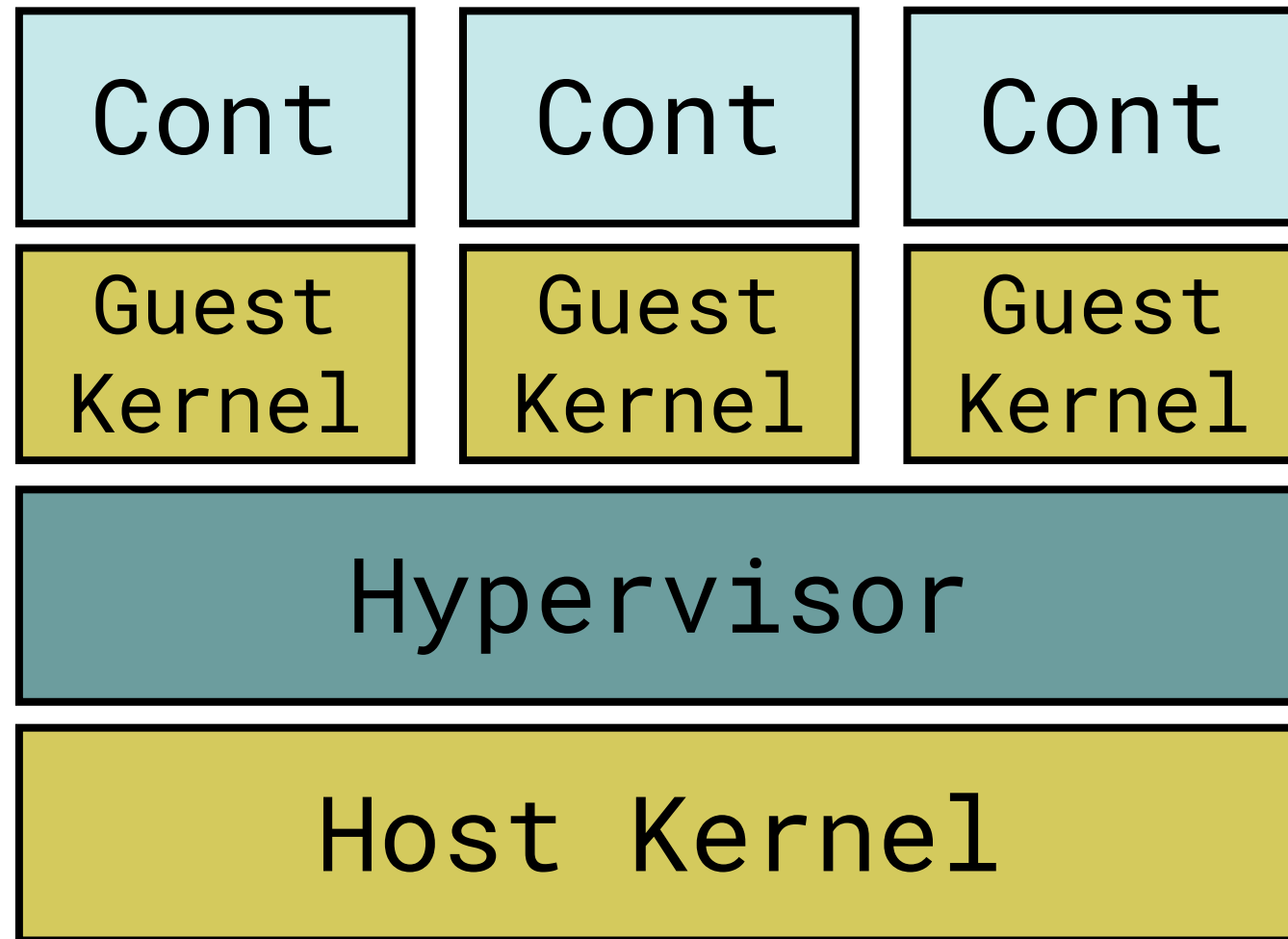
## CNI

- > CNI standardizes how orchestrators connect containers to a network
- > Enables pod-to-pod communication
- > CNI plug-in assigns interfaces and IPs to pods and does IP address management
- > To not depend on a particular container, network interface is configured for dummy „pause“ aka „sandbox“ container
- > Container runtime (e.g. containerd) calls CNI plug-in executable (e.g. Calico) to add an interface to container's networking namespace (sandbox/pause container)

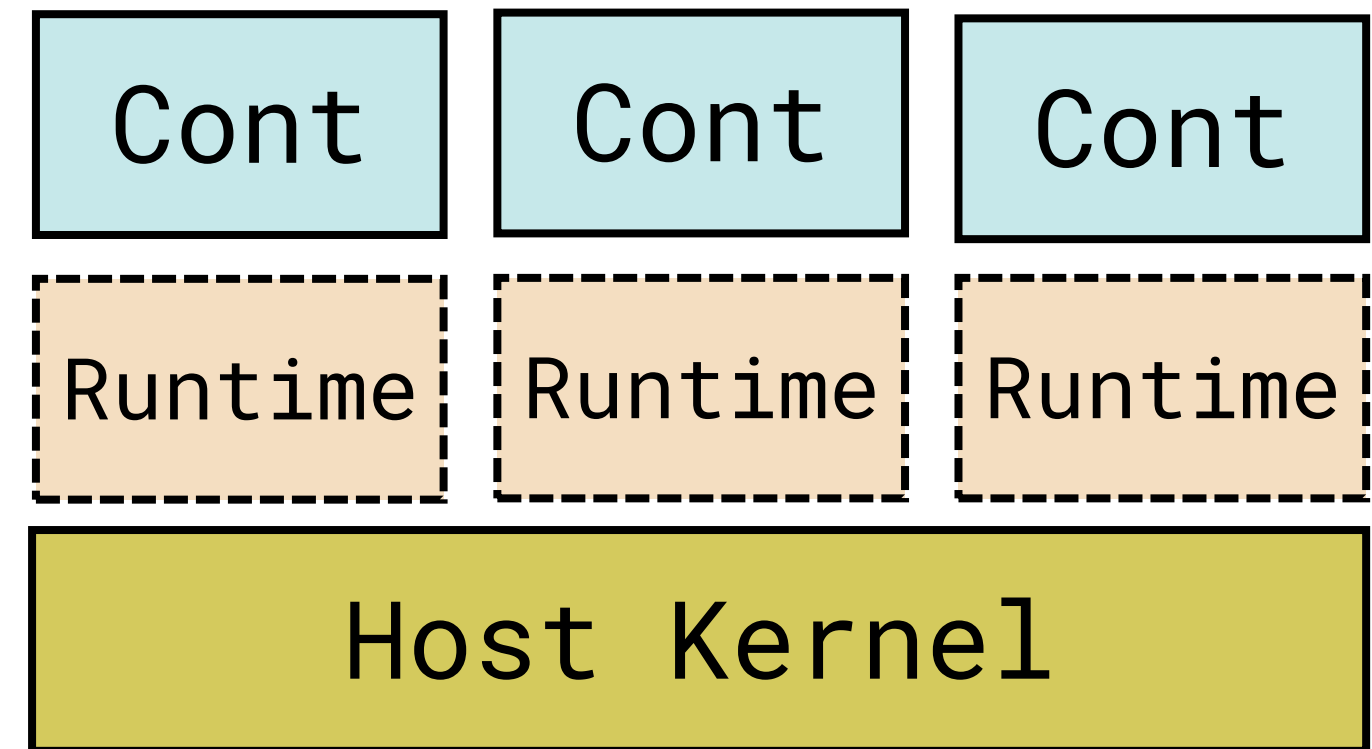


# Container isolation

## Overview



VMs



Containers



# Container isolation

## Overview

- > Containers are just processes
  - ◇ Kernel does not know containers
- > Configuration of isolation depend on runtime (and Kubernetes)!!
  - ◇ Different runtimes, different levels of security
  - ◇ Runtimes are configured by Kubernetes
- > Kernel namespaces and groups
  - ◇ Cgroups are object quotas and restrict how much CPU/memory/disk the process gets
  - ◇ Namespaces restrict what a process can see (see next slide)

# Container isolation

## Kernel namespaces

- > Mount namespaces provides separate view on mounts
- > Network namespaces provides new virtual network stack
- > Pid namespace provides separate view on running processes
- > UTS namespace provides a separate hostname for this process
- > Cgroup namespace hides actual cgroup the process is running under
- > IPC namespace to prevent shared memory other IPC between processes

# Container isolation

## Kernel namespaces

- > Mount namespaces provides separate view on mounts
- > Network namespaces provides new virtual network stack
- > Pid namespace provides separate view on running processes
- > UTS namespace provides a separate hostname for this process
- > Cgroup namespace hides actual cgroup the process is running under
- > IPC namespace to prevent shared memory other IPC between processes
- > User namespaces provides a separate user inside the container different from the one the process is running as on the host
  - ◇ User namespaces are NOT turned on by default in docker and most other runtimes!
  - ◇ But can be configured

# Container isolation

## Further measures

### > Capabilities

- ◇ Capabilities deny privileged operations, e.g. no CAP\_SYS\_ADMIN, no CAP\_SYS\_MODULE

### > Seccomp and MAC (SELinux / AppArmor) increase level of isolation

- ◇ Often these mechanism exist, but are not enabled by default in Kubernetes / on node OS ... Enable them for better isolation!

- ◇ Seccomp filters dangerous system calls, e.g. clock\_adjtime, delete\_module, reboot

- ◇ MAC denies access to file system paths (e.g. /proc or /sys) and system calls

# Container isolation

## Seccomp

### > Seccomp

- ◇ Seccomp profile are a kernel feature
  - Linux kernel has a few hundred system calls
  - Only a few of them are needed by a given program
  - The rest is an unnecessary attack surface
  - Check out your `/proc/PID/status` to see which processes use seccomp
    - 0 for disabled, 1 for strict, and 2 for filter
- ◇ K8s disables seccomp by default (Unconfined)
  - Since v1.22 you can globally change default at least (alpha)
  - If SeccompDefault feature gate is enabled for kubelet and `--seccomp-default` command line argument set, pods use the RuntimeDefault seccomp profile whenever no other seccomp profile is specified
- ◇ RuntimeDefault or custom seccomp profile can be configured in pod configuration
- ◇ Consider RuntimeDefault (if using uncommon runtime, check seccomp defaults first!)
  - Or audit system calls and use a custom list for even better isolation

# Container isolation

## Seccomp

SecurityContext

**seccompProfile:**

**type:** Localhost

**localhostProfile:** profiles/audit.json

- > Now violations are logged
- > Grep executable to find used systemcalls
- > Use that to build your custom profile

```
audit.json:
{
  "defaultAction": "SCMP_ACT_LOG",
  "architectures": [
    "SCMP_ARCH_X86_64",
    "SCMP_ARCH_X86",
    "SCMP_ARCH_X32"
  ],
  "syscalls": [
    {
      "names": [
        "accept4",
        "epoll_wait",
        "pselect6",
        ...
      ],
      "action": "SCMP_ACT_ALLOW"
    }
  ]
}
```

# Container isolation

## Seccomp

SecurityContext

**seccompProfile:**

**type:** Localhost

**localhostProfile:** profiles/block.json

- > When profile is complete
  - ◇ Change defaultAction to error mode
- > Pod is terminated upon violation

```
block.json:
{
  "defaultAction": "SCMP_ACT_ERRNO",
  "architectures": [
    "SCMP_ARCH_X86_64",
    "SCMP_ARCH_X86",
    "SCMP_ARCH_X32"
  ],
  "syscalls": [
    {
      "names": [
        "accept4",
        "epoll_wait",
        "pselect6",
        ...
      ],
      "action": "SCMP_ACT_ALLOW"
    }
  ]
}
```

# Container isolation

## SELinux

- > In some node OSes (e.g. CoreOS), SELinux is configured, but not enabled
- > There are some incompatibilities with certain non-local volumes

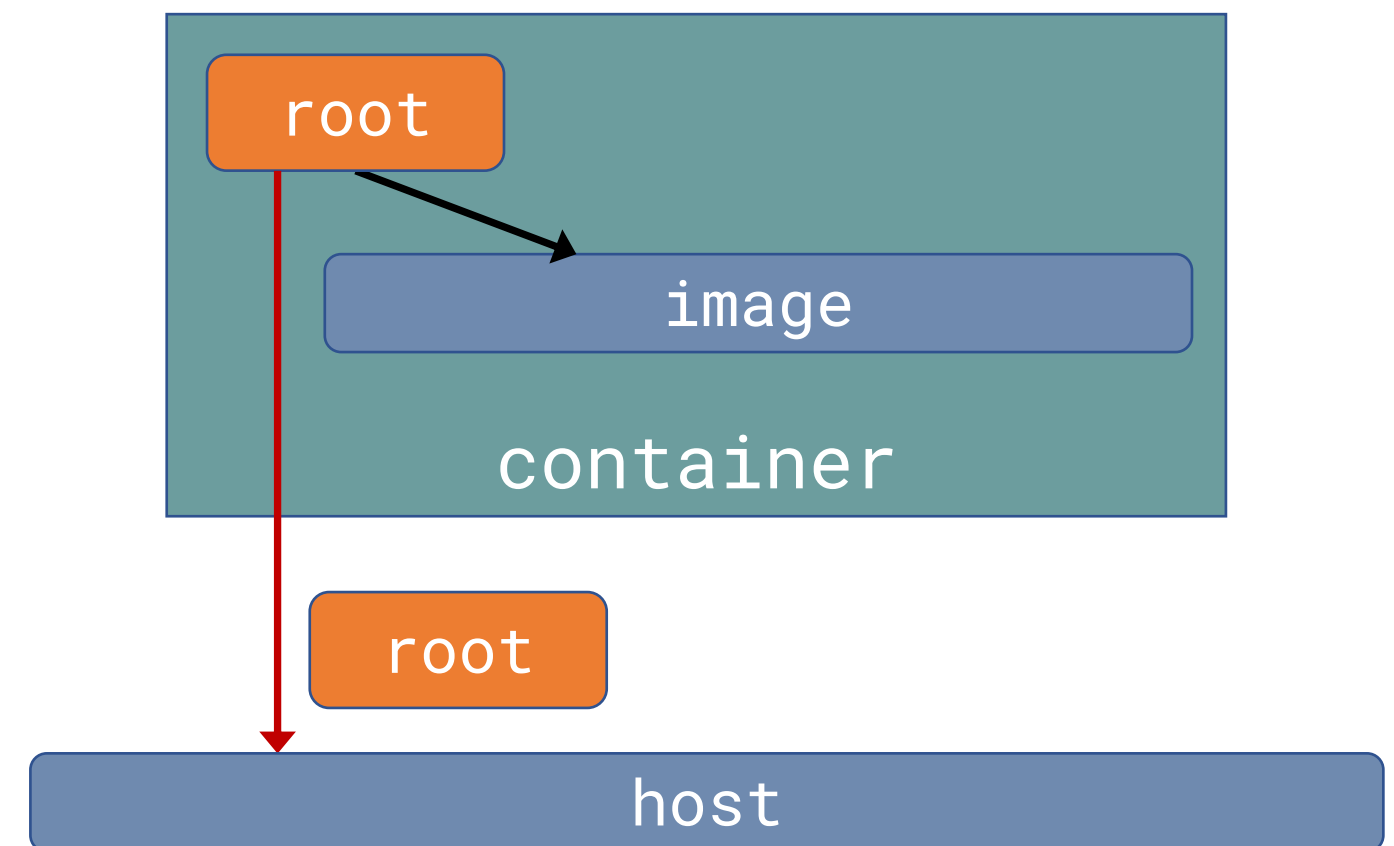


# Container isolation

## Users

- > The container runs under a specific user
- > In most runtimes, the user namespace is not enabled by default
  - ◇ User on the host = user in the container
- > Therefore, a non-root user (uid/gid>0) should be selected to run the container
  
- > User namespaces could be enabled
  - ◇ Are incompatible with some non-local volumes
- > In case there is a problem with container isolation, damage might be reduced

Without user namespaces:

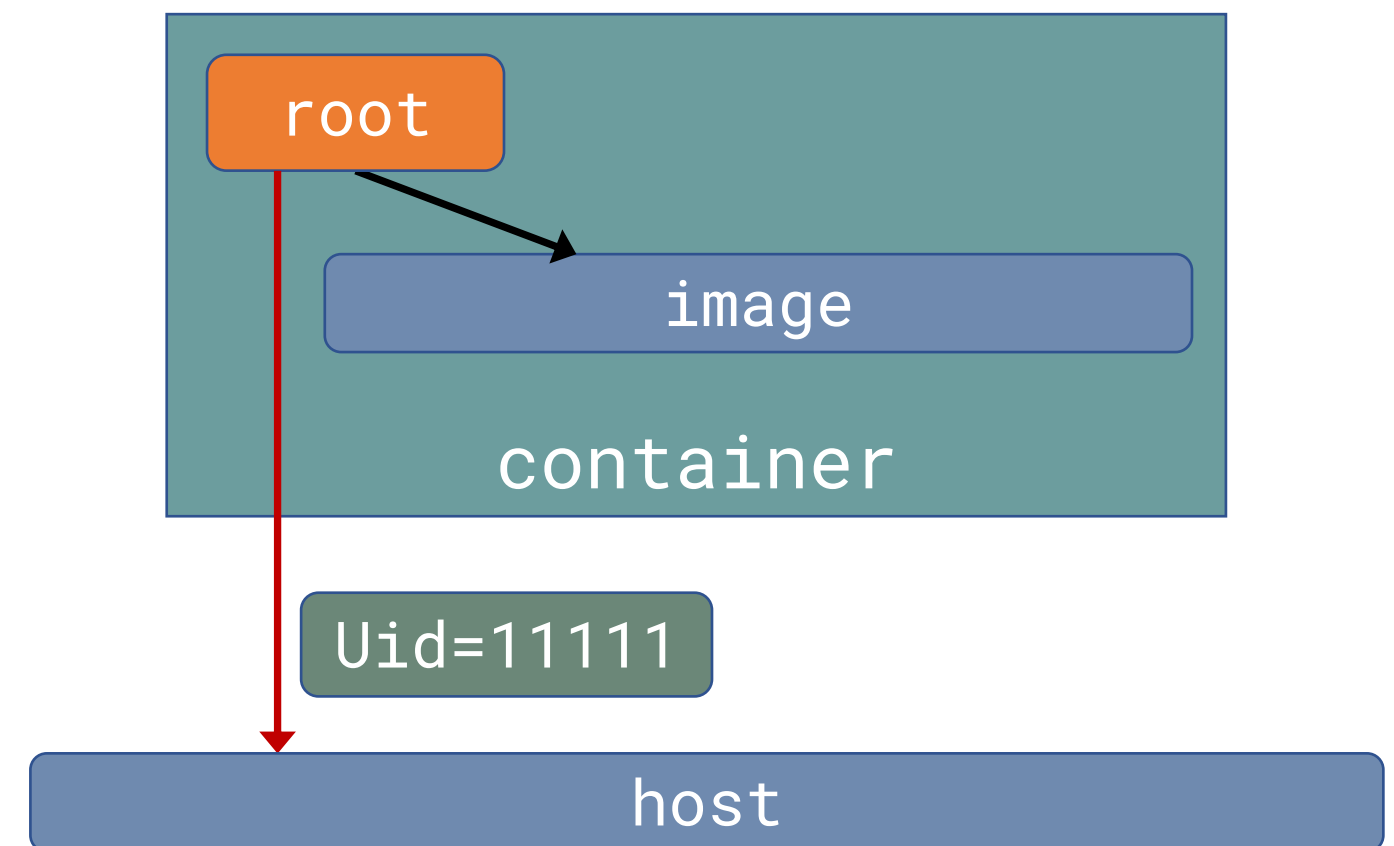


# Container isolation

## Users

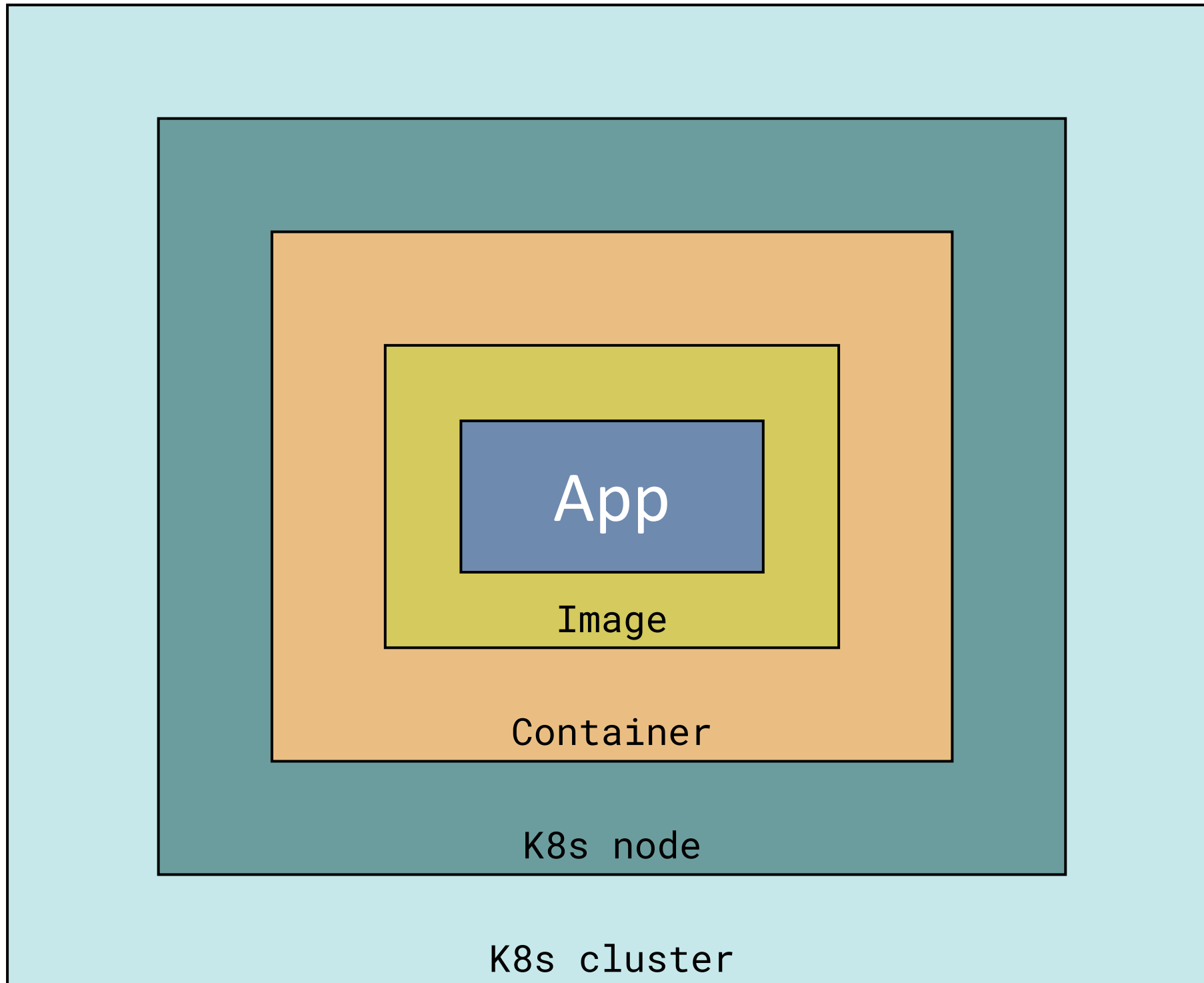
- > The container runs under a specific user
- > In most runtimes, the user namespace is not enabled by default
  - ◇ User on the host = user in the container
- > Therefore, a non-root user (uid/gid>0) should be selected to run the container
  
- > User namespaces could be enabled
  - ◇ Are incompatible with some non-local volumes
- > In case there is a problem with container isolation, damage might be reduced

With user namespaces:



# Kubernetes attacks vectors

## Attack chain example



### Example vulnerability

Application vuln

Image not hardened

Vulnerable kernel

Node not hardened

Cluster vulnerability

### Attack chain

RCE in application

Get Shell

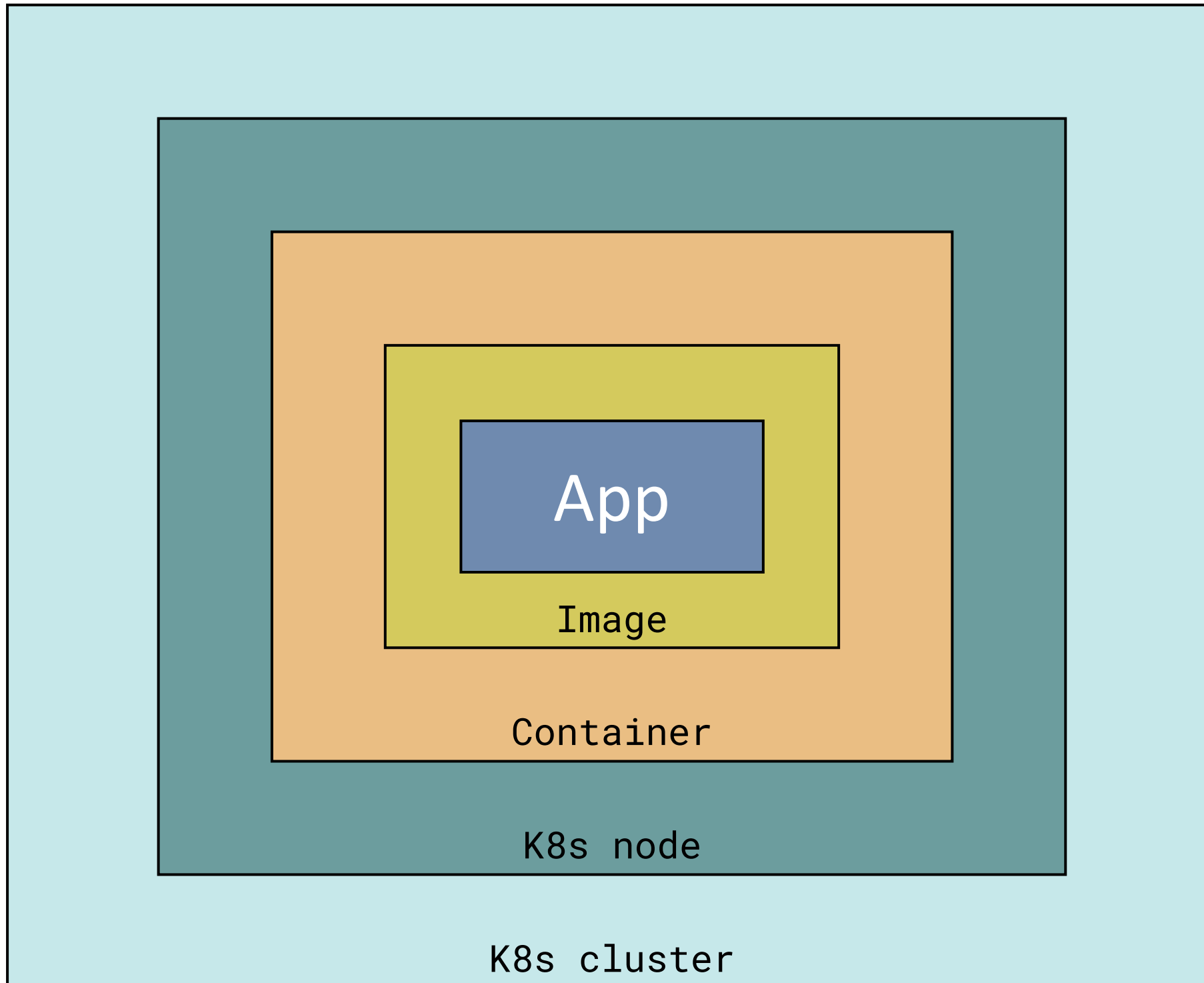
Container escape

Compromise workloads

Compromise cluster

# Kubernetes attacks vectors

## Attack chain example



> With access to cluster (e.g. multi-tenancy), you are already here

### Example vulnerability

Application vuln

Image not hardened

Vulnerable kernel

Node not hardened

Cluster vulnerability

### Attack chain

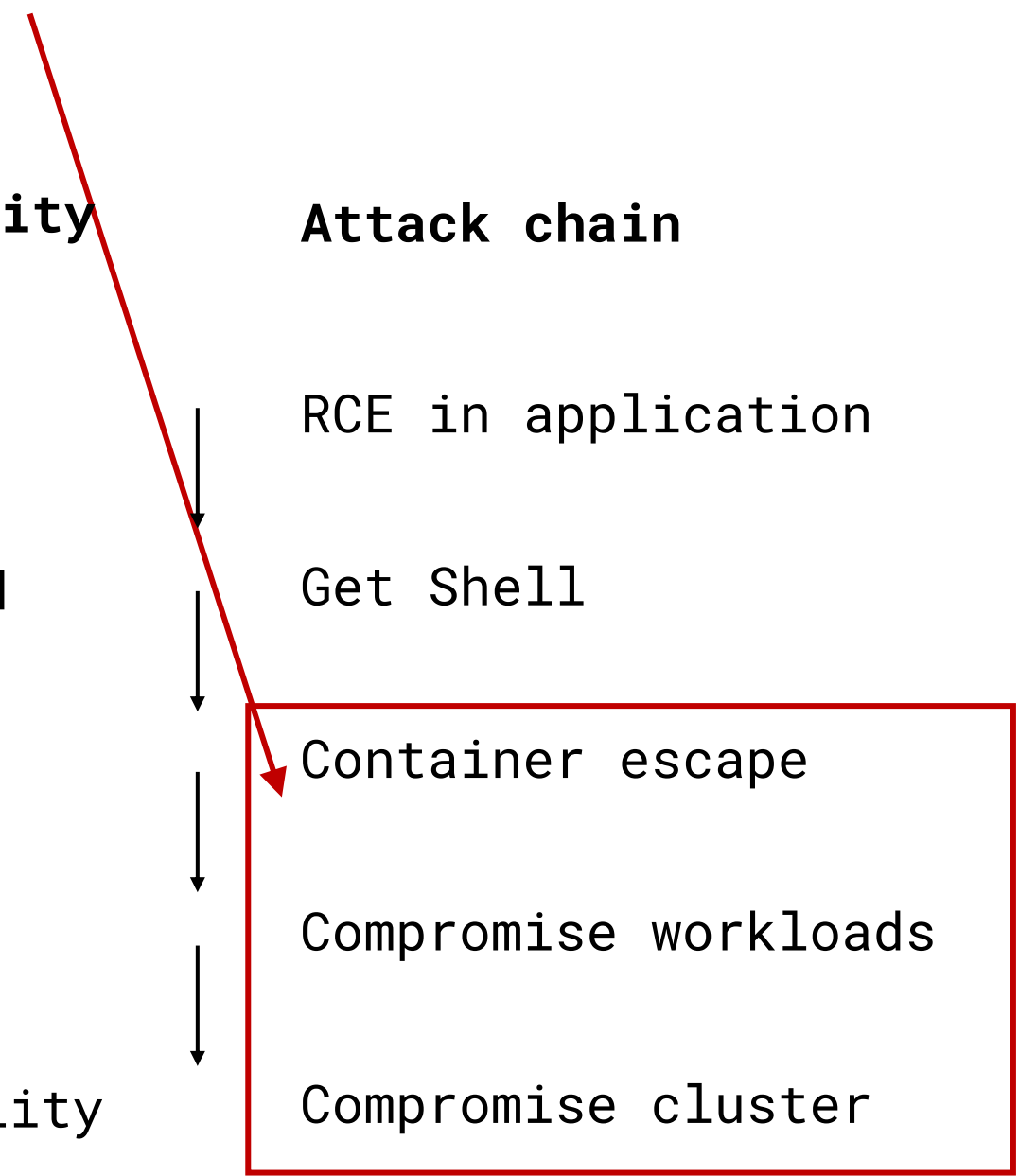
RCE in application

Get Shell

Container escape

Compromise workloads

Compromise cluster



# Kubernetes attacks vectors

## Threat matrix

> Microsoft released a threat matrix for k8s

Initial Access	Execution	Persistence	Privilege Escalation	Defense Evasion	Credential Access	Discovery	Lateral Movement	Collection	Impact
Using Cloud credentials	Exec into container	Backdoor container	Privileged container	Clear container logs	List K8S secrets	Access the K8S API server	Access cloud resources	Images from a private registry	Data Destruction
Compromised images in registry	bash/cmd inside container	Writable hostPath mount	Cluster-admin binding	Delete K8S events	Mount service principal	Access Kubelet API	Container service account		Resource Hijacking
Kubeconfig file	New container	Kubernetes CronJob	hostPath mount	Pod / container name similarity	Access container service account	Network mapping	Cluster internal networking		Denial of service
Application vulnerability	Application exploit (RCE)	Malicious admission controller	Access cloud resources	Connect from Proxy server	Applications credentials in configuration files	Instance Metadata API	Applications credentials in configuration files		
Exposed sensitive interfaces	SSH server running inside container				Access managed identity credential		Writable volume mounts on the host		
	Sidecar injection				Malicious admission controller		CoreDNS poisoning		
							ARP poisoning and IP spoofing		

<https://www.microsoft.com/security/blog/2021/07/21/the-evolution-of-a-matrix-how-attck-for-containers-was-built/>

# Kubernetes attacks vectors

## Threat matrix

> Microsoft released a threat matrix for k8s

Initial Access	Execution	Persistence	Privilege Escalation	Defense Evasion	Credential Access	Discovery	Lateral Movement	Collection	Impact
Using Cloud credentials	Exec into container	Backdoor container	Privileged container	Clear container logs	List K8S secrets	Access the K8S API server	Access cloud resources	Images from a private registry	Data Destruction
Compromised images in registry	bash/cmd inside container	Writable hostPath mount	Cluster-admin binding	Delete K8S events	Mount service principal	Access Kubelet API	Container service account		Resource Hijacking
Kubeconfig file	New container	Kubernetes CronJob	hostPath mount	Pod / container name similarity	Access container service account	Network mapping	Cluster internal networking		Denial of service
Application vulnerability	Application exploit (RCE)	Malicious admission controller	Access cloud resources	Connect from Proxy server	Applications credentials in configuration files	Instance Metadata API	Applications credentials in configuration files		
Exposed sensitive interfaces	SSH server running inside container		Runtime vuln		Access managed identity credential		Writable volume mounts on the host		
	Sidecar injection		Kernel vuln		Malicious admission controller		CoreDNS poisoning		
								ARP poisoning and IP spoofing	

<https://www.microsoft.com/security/blog/2021/07/21/the-evolution-of-a-matrix-how-attck-for-containers-was-built/>



# Common vulnerabilities

## Privileged containers / host-namespaces used

Good overview:

<https://bishopfox.com/blog/kubernetes-pod-privilege-escalation>

Priv flag	Hostpid	Hostpath	Hostnetw	Hostipc	Example privilege escalation techniques
Y	Y	Y	Y	Y	<ul style="list-style-type: none"> <li>chroot to host fs and read secrets mounted in any pod on node</li> <li>If you can schedule pod to control plane node, read secrets from etcd</li> </ul>
Y	Y				<ul style="list-style-type: none"> <li>Nsenter into namespace of process 1 (init) and execute shell on node</li> </ul>
Y					<ul style="list-style-type: none"> <li>Mount host filesystem and go from there</li> <li>Exploit cgroup user mode helper programs (metasploit module)</li> <li>Install a bad kernel module</li> <li>Cgroup's release_agent feature (Felix Wilhelm)</li> </ul>
		Y			<ul style="list-style-type: none"> <li>Kubeconfig files with secrets</li> <li>chroot to host fs and read secrets mounted in any pod on node</li> <li>If you have SSH network access, add an authorized SSH key</li> </ul>
	Y				<ul style="list-style-type: none"> <li>Kill processes</li> <li>Look for passwords/tokens, e.g. in command line args, env. vars</li> </ul>
			Y		<ul style="list-style-type: none"> <li>Sniff traffic</li> <li>Access services bound to localhost</li> <li>Bypass network policy</li> </ul>
				Y	<ul style="list-style-type: none"> <li>Check available IPC mechanisms and read/write from/to them</li> </ul>

# Common vulnerabilities

## Outdated software

### > Outdated Kubernetes

- ◇ Many companies or providers often multiple major versions behind
  - Might lead to missing security features
- ◇ At least patches should be applied regularly and a maintained major version used!
- ◇ Examples
  - Symlink exchange allows host filesystem access (CVE-2021-25741)
    - Affects kubelet and applies to multi-tenant clusters
  - Secrets exfiltration (CVE-2021-25742)
    - Affects nginx ingress controller and applies to multi-tenant cluster



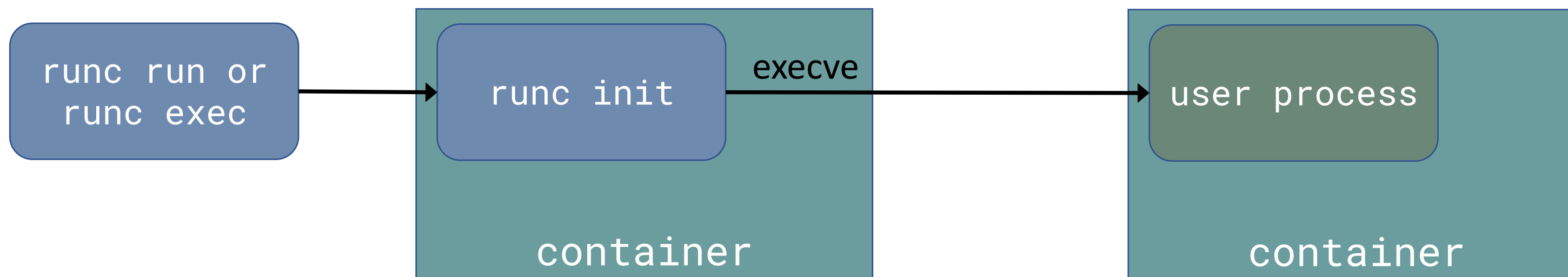
# Common vulnerabilities

## Outdated runtime

### > Outdated runtime

#### ◇ Example

- Runc allowed escape by overwriting `/proc/[runc-pid]/exe` (CVE-2019-5736)
  - By executing `/proc/self/exe` within container (custom entrypoint for new container or `exec` in existing container), `/proc/[runc-pid]/exe` would point to runc on the host; attacker can overwrite it (if container runs a root), which results in a modified runc on the host; next call of runtime results in root



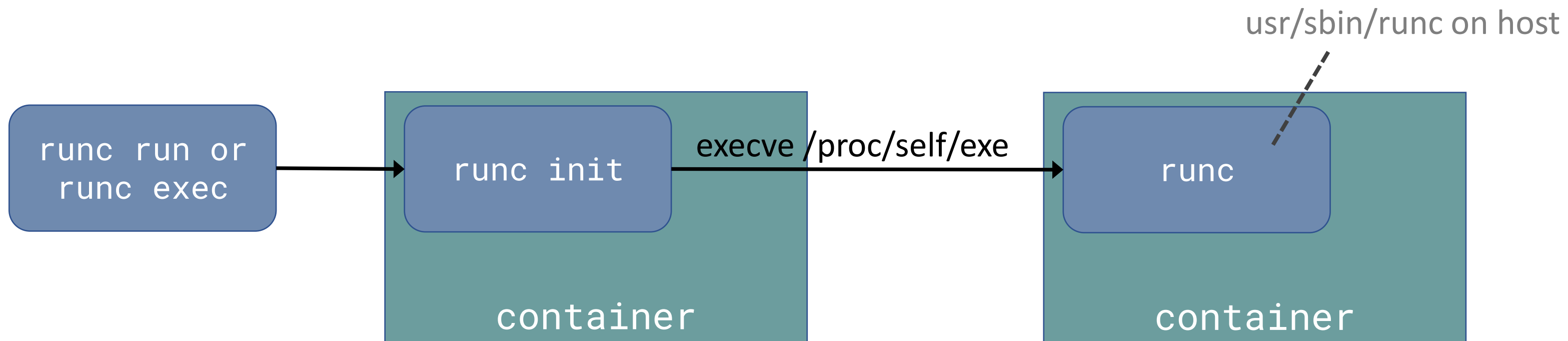
# Common vulnerabilities

## Outdated runtime

### > Outdated runtime

#### ◇ Example

- Runc allowed escape by overwriting `/proc/[runc-pid]/exe` (CVE-2019-5736)
  - By executing `/proc/self/exe` within container (custom entrypoint for new container or `exec` in existing container), `/proc/[runc-pid]/exe` would point to `runc` on the host; attacker can overwrite it (if container runs as root), which results in a modified `runc` on the host; next call of runtime results in root



# Common vulnerabilities

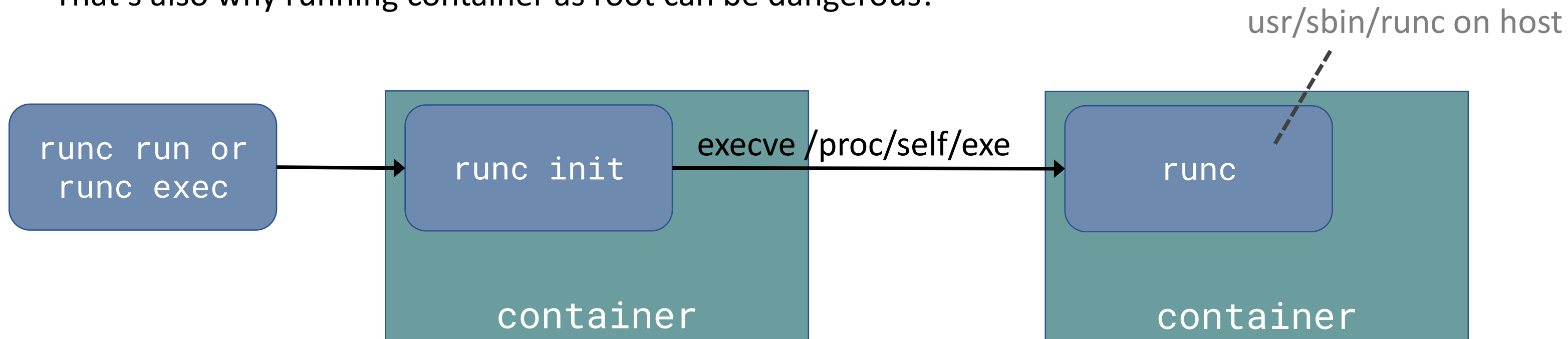
## Outdated runtime

### > Outdated runtime

#### ◇ Example

- Runc allowed escape by overwriting `/proc/[runc-pid]/exe` (CVE-2019-5736)
  - By executing `/proc/self/exe` within container (custom entrypoint for new container or `exec` in existing container), `/proc/[runc-pid]/exe` would point to `runc` on the host; attacker can overwrite it (if container runs as root), which results in a modified `runc` on the host; next call of runtime results in root

That's also why running container as root can be dangerous!



# Common vulnerabilities

## Outdated runtime

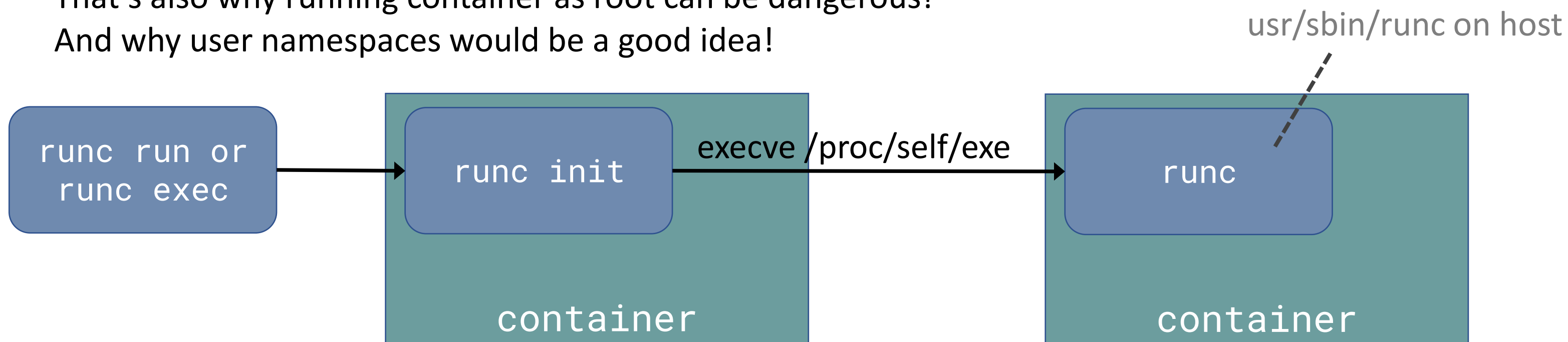
### > Outdated runtime

#### ◇ Example

- Runc allowed escape by overwriting `/proc/[runc-pid]/exe` (CVE-2019-5736)
  - By executing `/proc/self/exe` within container (custom entrypoint for new container or `exec` in existing container), `/proc/[runc-pid]/exe` would point to `runc` on the host; attacker can overwrite it (if container runs as root), which results in a modified `runc` on the host; next call of runtime results in root

That's also why running container as root can be dangerous!

And why user namespaces would be a good idea!



# Common vulnerabilities

## Outdated kernel

- > Kernel is a large codebase with lots of vulnerabilities not yet discovered
- > Kernel vulnerabilities are published regularly
  - ◇ E.g. CVE-2022-0185 from January
    - Exploit would need CAP\_SYS\_ADMIN
    - Using unshare, one can easily escalate to CAP\_SYS\_ADMIN on default Kubernetes
      - Because no seccomp by default that blocks unshare

# Common vulnerabilities

## Outdated kernel

- > Dirty COW container escape
  - ◇ DEMO

# Common vulnerabilities

## Outdated kernel

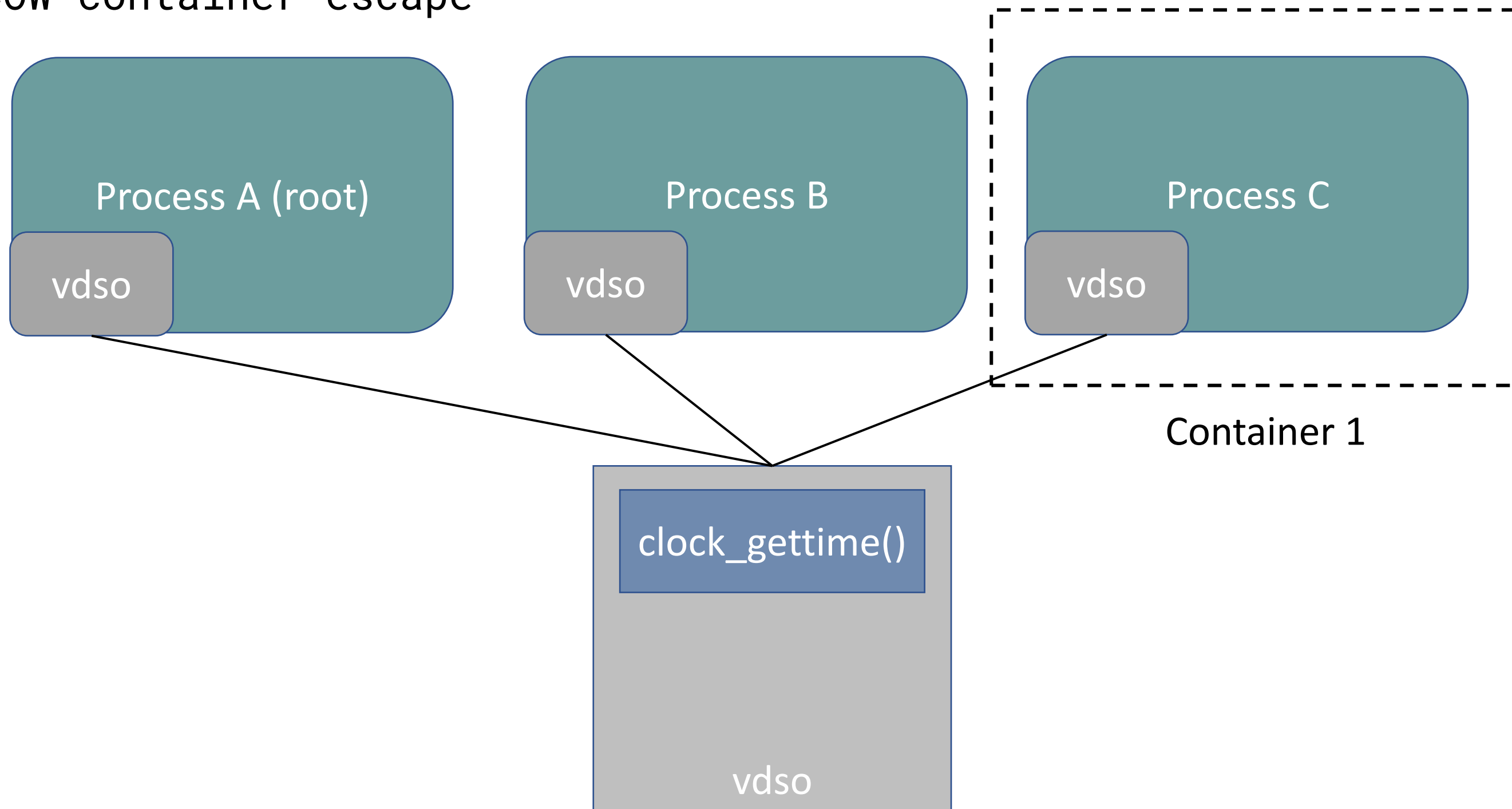
### > Dirty COW container escape

- ◇ Race condition in the way Linux handles read-only private mappings upon copy-on-write
- ◇ Under certain circumstances, someone can overwrite the original rather than the copy
- ◇ scumjr's dirtycow-vdso exploit
  - vDSO is a memory area mapped into every process' address space to optimize performance for certain system calls
  - Exploit modifies `clock_gettime()` in vDSO to run shellcode
  - Shellcode waits for a root process to call it and invokes reverse shell to an IP:port

# Common vulnerabilities

Outdated kernel

- > Dirty COW container escape

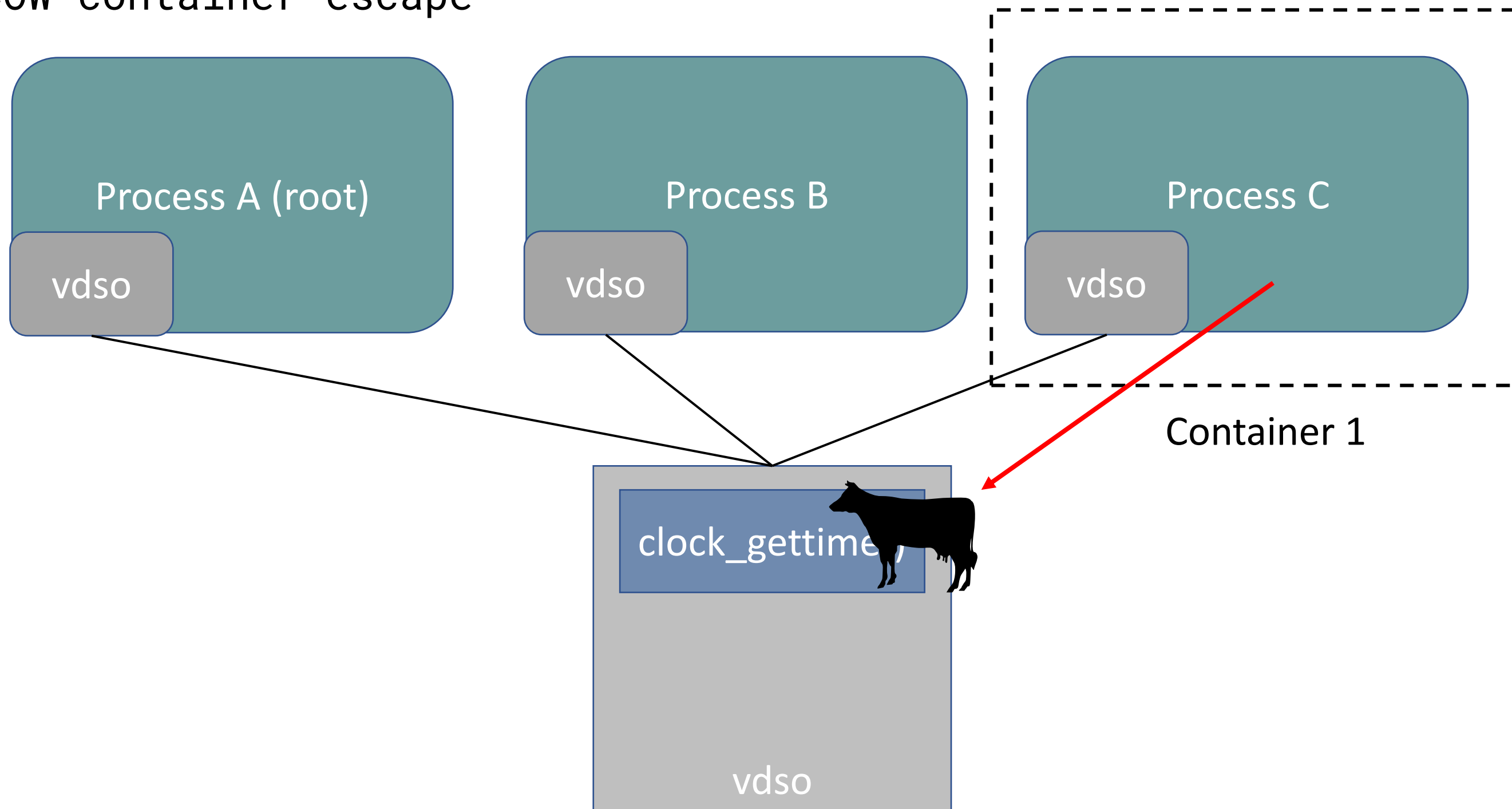




# Container isolation

## Outdated kernel

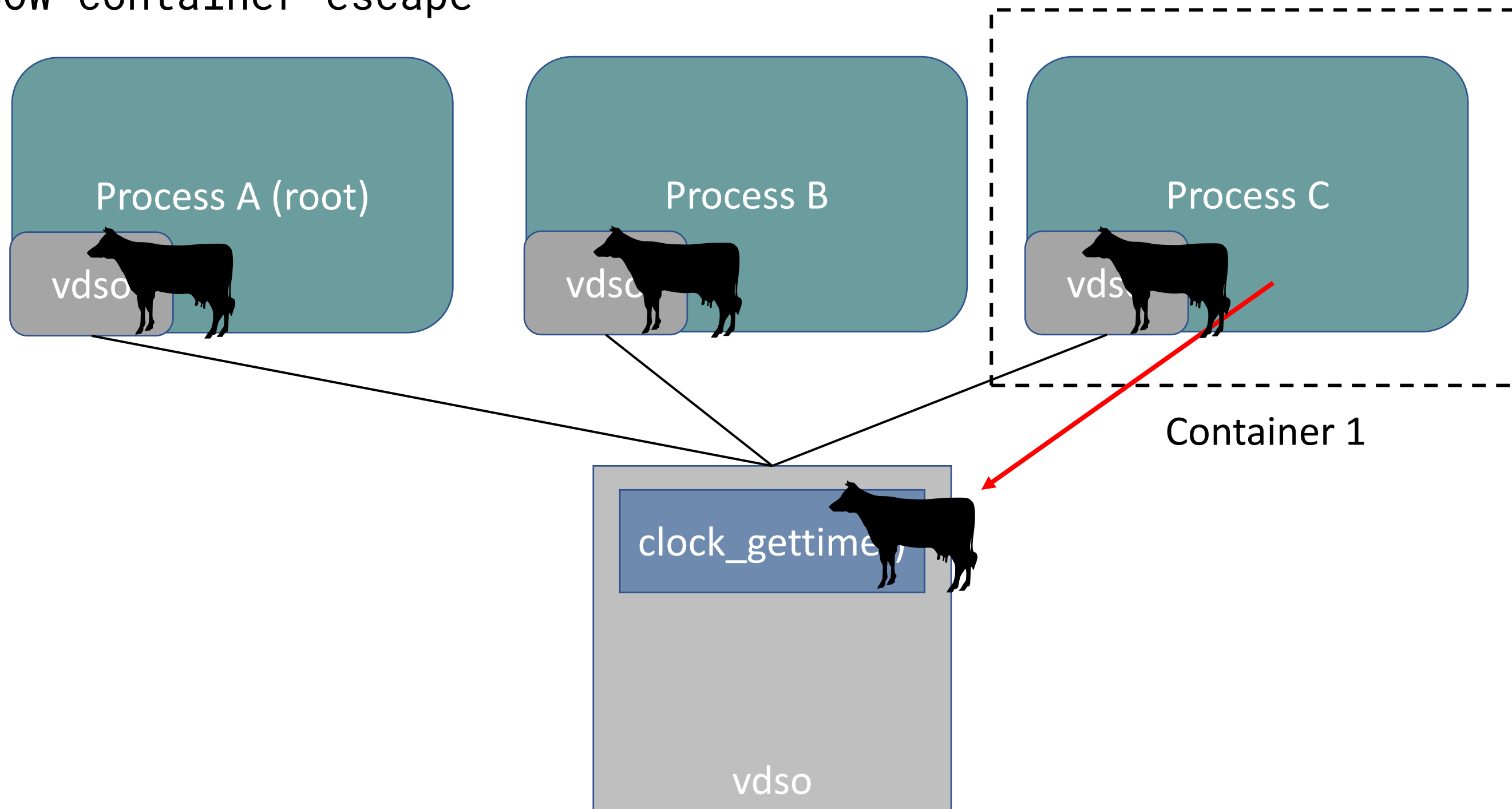
### > Dirty COW container escape



# Container isolation

## Outdated kernel

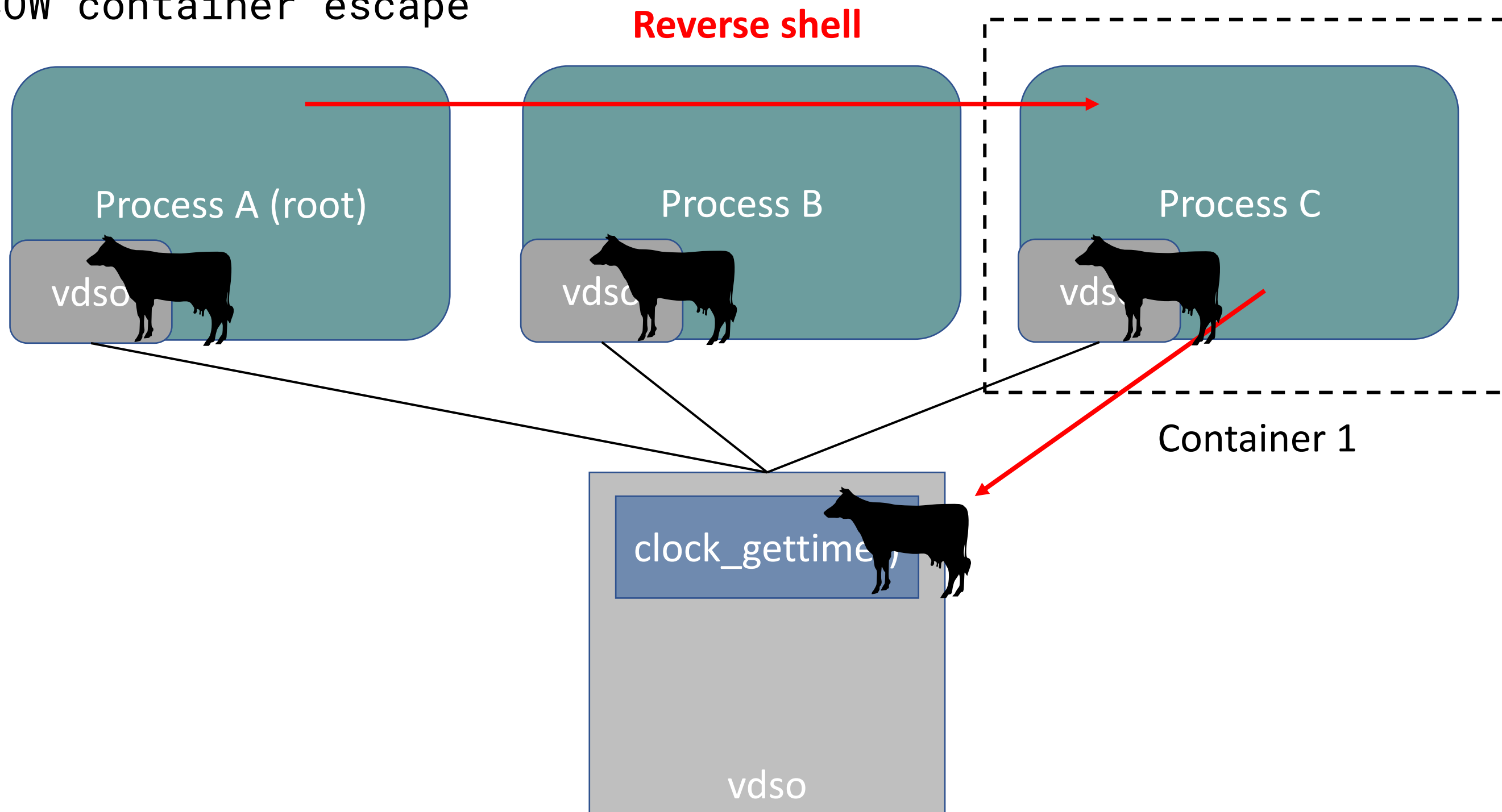
### > Dirty COW container escape



# Container isolation

Outdated kernel

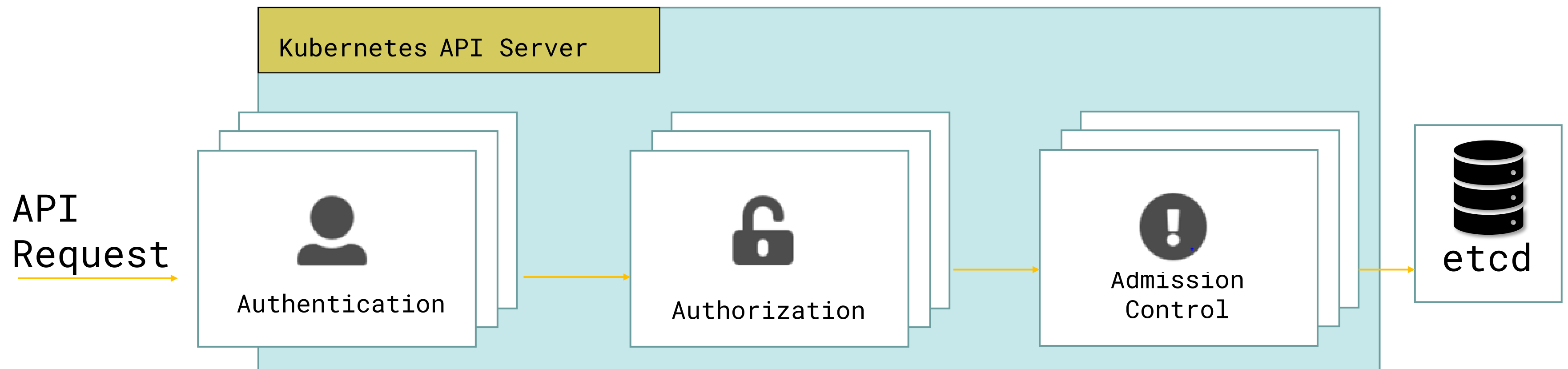
> Dirty COW container escape



# Common vulnerabilities

## RBAC issues

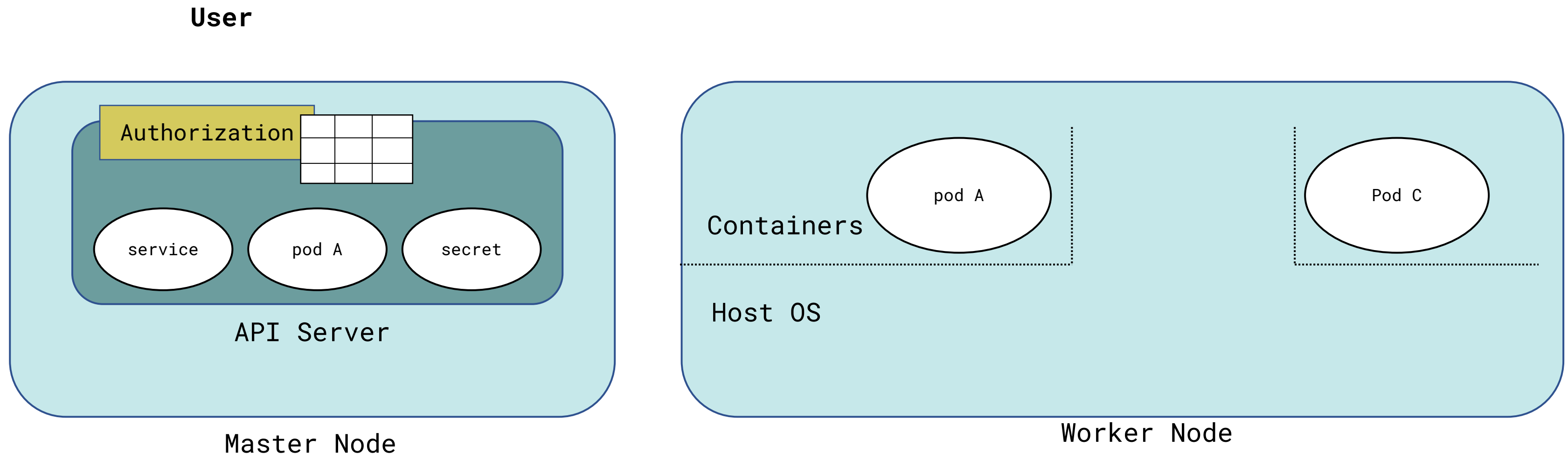
- > RBAC not configured correctly
  - ◇ Many users, objects, etc. and therefore misconfigurations
  - ◇ Limited to CRUD (see next slide)



# Common vulnerabilities

## RBAC issues

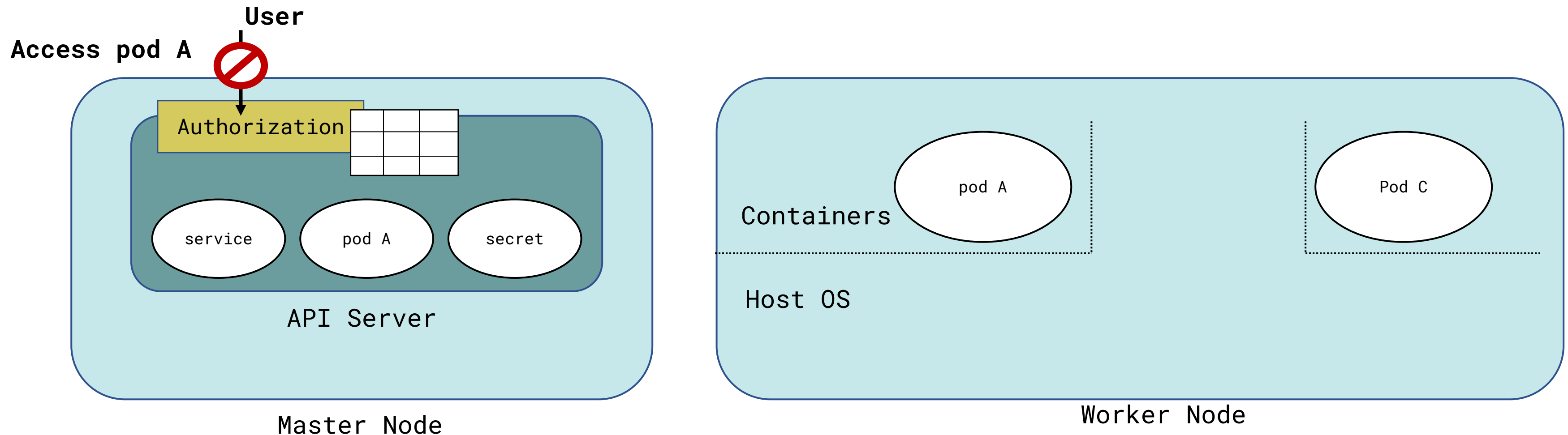
- > RBAC only applies to k8s objects accessed via the API
  - ◇ Only CRUD
  - ◇ Does not inspect created objects
- > It can be bypassed with the pod create privilege
- > Admission controller needs to be used



# Common vulnerabilities

## RBAC issues

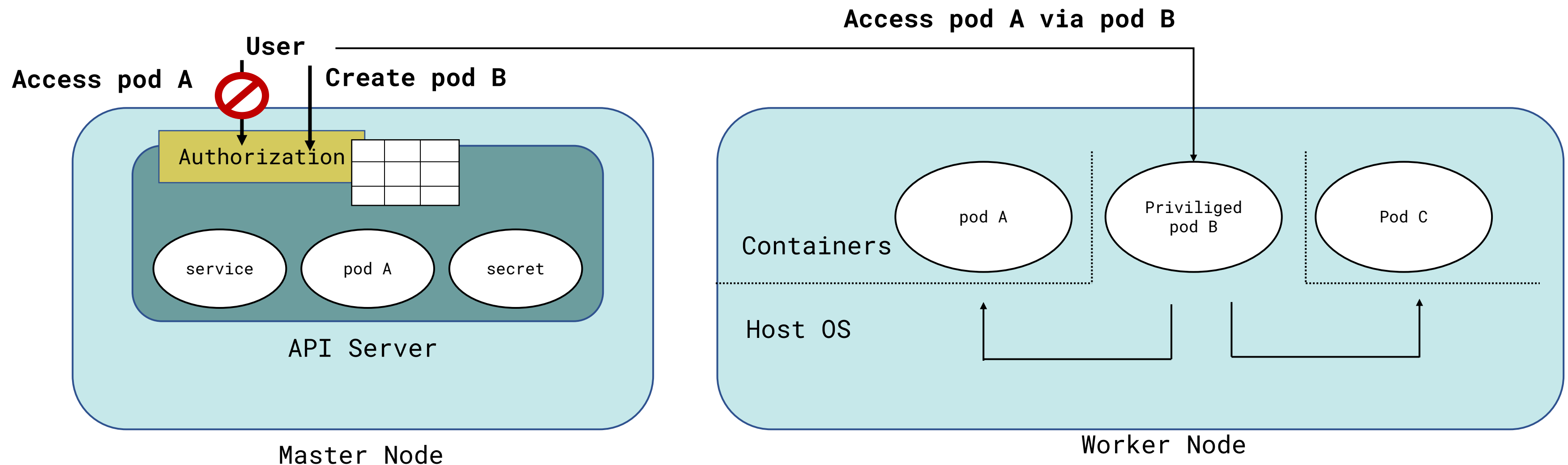
- > RBAC only applies to k8s objects accessed via the API
  - ◇ Only CRUD
  - ◇ Does not inspect created objects
- > It can be bypassed with the pod create privilege
- > Admission controller needs to be used



# Common vulnerabilities

## RBAC issues

- > RBAC only applies to k8s objects accessed via the API
  - ◇ Only CRUD
  - ◇ Does not inspect created objects
- > It can be bypassed with the pod create privilege
- > Admission controllers (or different node pools) need to be used



# Common vulnerabilities

## Missing hardening measures

- > Missing container hardening (pod's securityContext)
  - ◇ seccomp not enabled
  - ◇ "allowPrivilegeEscalation: false" not configured
  - ◇ readOnlyRootFilesystem not configured
- > Missing cluster hardening
  - ◇ Recommended admission controllers not configured
  - ◇ Seccomp not enabled by default
- > Nodes not on an immutable container OS
- > Protecting from dangerous workloads
  - ◇ Pod Security Admission
    - 3 levels: privileged, baseline, restricted



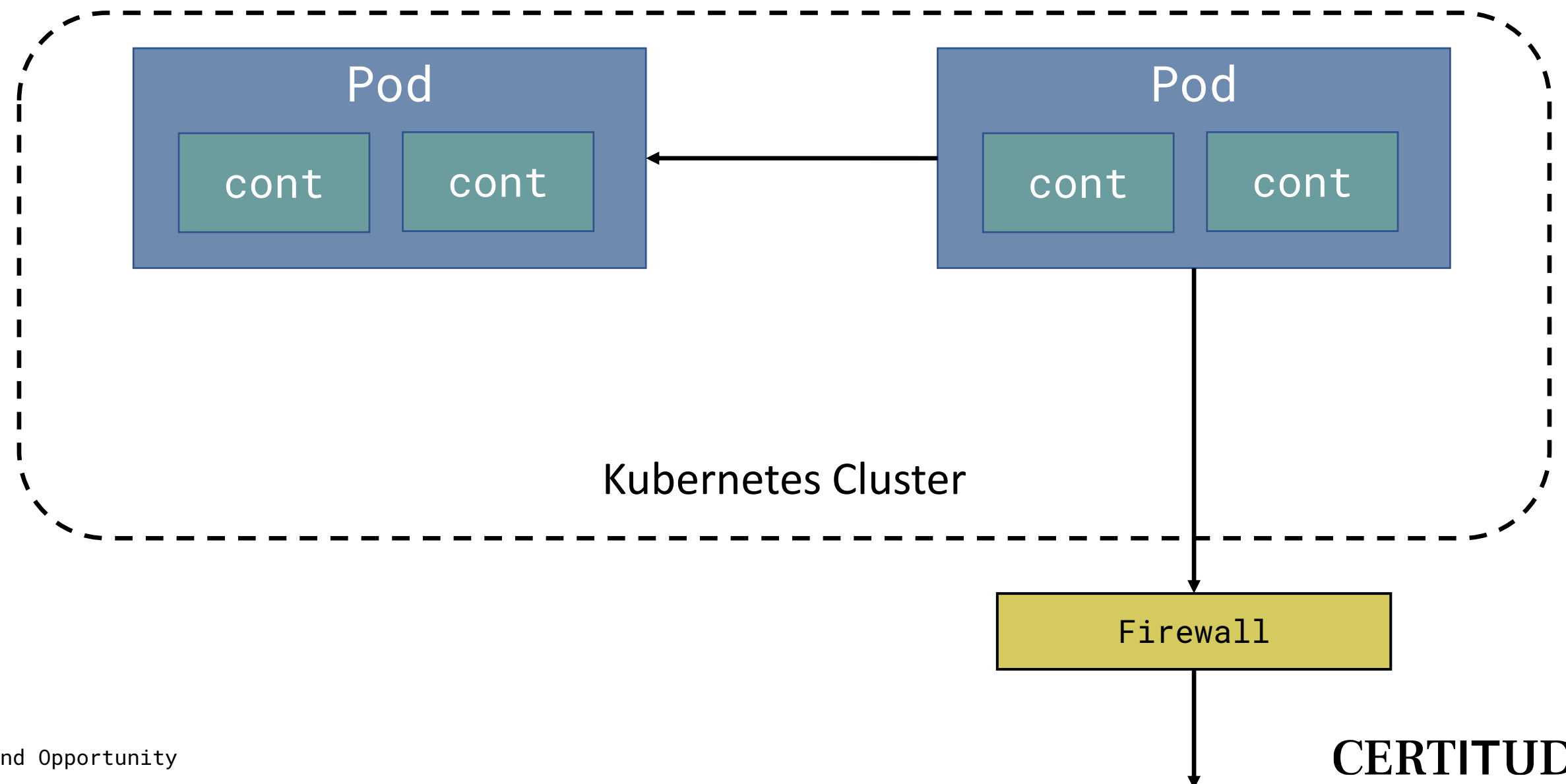
# Common vulnerabilities

## Missing network policies

> By default all communication is allowed

> Other devices on network, e.g.

- ◇ Cloud metadata service
- ◇ Vulnerable applications



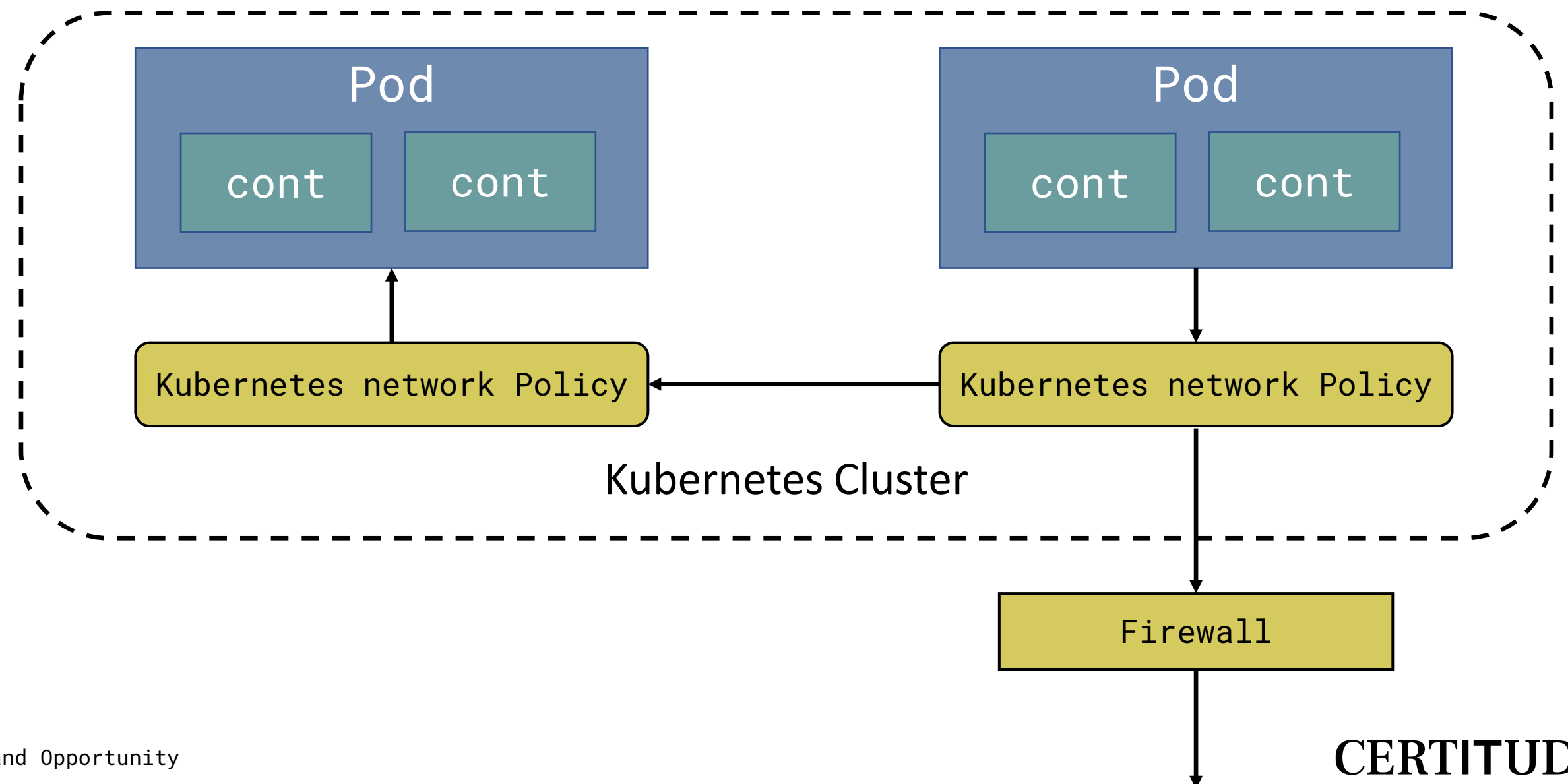
# Common vulnerabilities

## Missing network policies

- > By default all communication is allowed
- > If there is an ingress policy for pod, ingress traffic is denied by default
- > If there is an egress policy for pod, egress traffic is denied by default

- > Other devices on network, e.g.

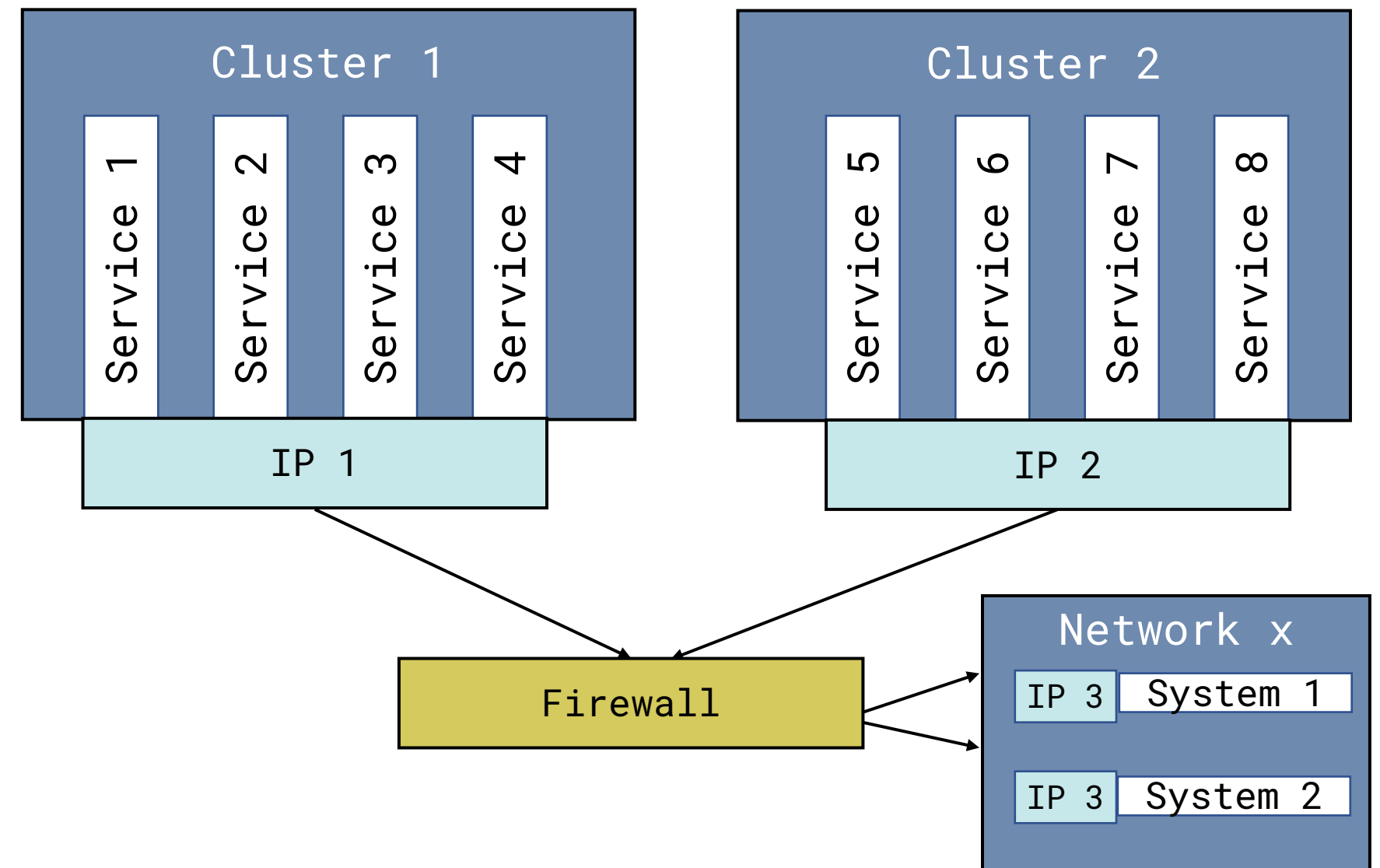
- ◇ Cloud metadata service
- ◇ Vulnerable applications



# Common vulnerabilities

## Firewalling outside cluster

- > All service „hide“ between one or the same set of Ips
  - ◇ Ingress
    - „By default“, a cluster has a single external IP (ingress controller)
    - Node IPs could also be used
      - But hosts could be scaled up or down and IPs might change
  - ◇ Egress
    - Traffic leaves with node-IP
      - Since pod can run on any host (by default), it could be any node IP



# Common vulnerabilities

## Other issues

- > Changes in modules might render certain controls ineffective
  - ◇ Network plugins have different levels of support for certain object types (e.g. network policies)
- > Debugging artefacts
  - ◇ Nodes used for debugging, which might introduce vulnerabilities
  - ◇ Anonymous\_auth enabled for api-server or kubelet
  - ◇ Old/vulnerable pods
- > Secrets in images
  - ◇ E.g. Microsoft had the DP API encryption key in images (same for all images)
    - <https://certitude.consulting/blog/en/windows-docker-dp-api-vulnerability-cve-2021-1645/>

# Multi-tenancy

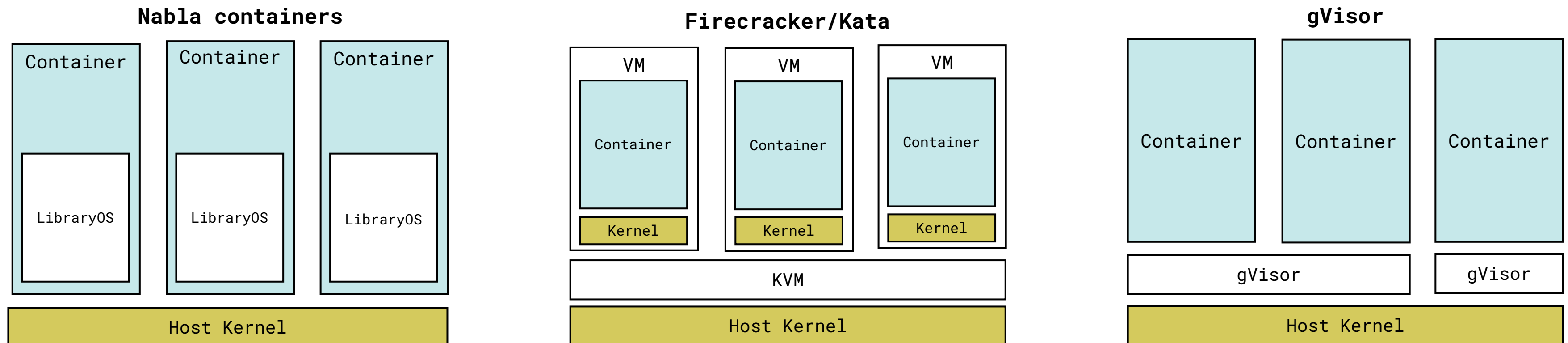
Does kubernetes allow multi-tenancy?

- > Generally, avoid multi-tenant clusters
  - ◇ Except you really know what you are doing
- > Normal container runtimes do not provide enough isolation
  - ◇ There are runtimes built for better security
    - Kubernetes provides RuntimeClass to use different runtimes in the same cluster
- > Cluster configuration mistakes are a large attack vector
- > Software-vulnerabilities in cluster components are a large attack vector

# Multi-tenancy

How do cloud providers build multi-tenant clusters?

- > How do cloud providers build multi-tenant clusters?
  - ◇ They don't: In GKE, AKS, EKS each customer gets own cluster and virtual hardware
- > What about their Container-as-a-Service offerings?
  - ◇ They use container runtimes designed for security
    - Based on virtualization, lightweight VMs (Kata containers, Firecracker)
    - User-mode kernels (Nabla Containers, gVisor)
  - ◇ AND they have a secure architecture covering network separation, access control, etc.



# Multi-tenancy

What would need to be considered for a multi-tenant cluster?

- > What would need to be done for a multi-tenant cluster?
  - ◇ Secure containers or separate resource pools (nodes) in separate network segments (otherwise container escape)
  - ◇ Separate overlay networks or mandatory network policies (otherwise other tenants can be reached)
  - ◇ Admission controllers to restrict dangerous objects being created (otherwise privileged containers/disabled namespaces)
  - ◇ Make sure one tenant cannot use another tenants private images cached on node (AlwaysPullImages admission controller)
  - ◇ Separate namespaces and strict RBAC configuration (otherwise access to objects from other tenants)
  - ◇ Always up-to-date cluster, runtime, etc. (otherwise escape vulnerabilities)
  - ◇ All exposed components must support multi-tenancy (e.g. monitoring tools)
  - ◇ Many other things to consider
  - ◇ → No room for mistakes...

# Opportunities

Not all is bad!

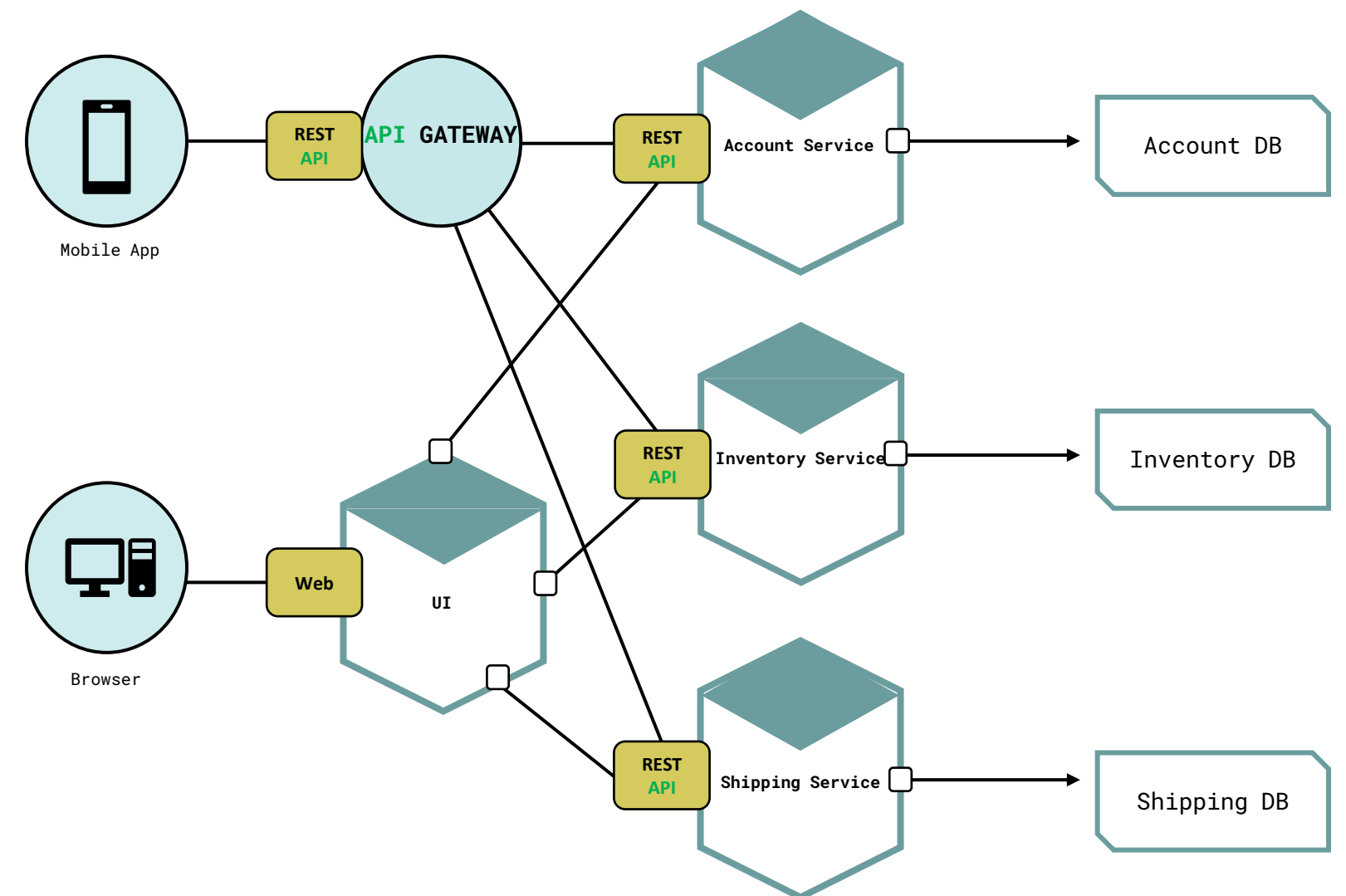
- > Many (free or commercial) tools to help you spot configuration issues
- > Free tools are e.g.
  - ◇ StackRox kube-linter: Checks YAML files for issues
  - ◇ AquaSec kube-bench: Runs pod to check cluster against CIS-benchmarks
  - ◇ AquaSec trivy: IaC scanner for K8s, docker images, dockerfiles, terraform
  - ◇ CyberArk kubiscan: Identify risks in RBAC configuration
  - ◇ Quay clair: Image vulnerability scanner
  - ◇ K8s krew plugin "access-matrix": Useful to check RBAC
  - ◇ Corneliusweig rakkess: Useful to check RBAC
  - ◇ Quarkslab kdigger: Check context/find vulnerabilities from inside a pod



# Opportunities

Not all is bad!

- > Images can be hardened
  - ◇ E.g., no shell, no package manager, no tools (curl/wget, etc.)
- > Cluster can be hardened
  - ◇ Often not default!
- > Small services easier to understand
  - ◇ Easier to create security profiles
- > Granular network policies
- > Granular containerization
- > So, with the right architecture, we might be able to improve security



# Thanks for your attention!

Questions?

> Follow us on social media

>  @nimmerrichter

>  @cert\_it\_ude / @certitude\_lab

>  Certitude Consulting GmbH