VMware SIG
Deep Dive into Kubernetes Scheduling

Performance and high availability options for vSphere

Steve Wong, Michael Gasch

KubeCon North America

December 13, 2018

Presenter

Bios

Steve Wong

Open Source Community Relations Engineer
VMware

Active in Kubernetes storage community since 2015. Chair of Kubernetes VMware SIG.

GitHub: @cantbewong

Michael Gasch

Application Platform Architect
VMware

Supports enterprises with architectural guidance, and works closely with VMware R&D, and member of CTO Ambassador staff.

GitHub: @embano1

Abstract

Kubernetes allows using topology labels to affect the scheduler’s placement of pods. This is used to spread pods across availability zones, while still respecting resource access and availability concerns. When Kubernetes runs on vSphere, the hypervisor platform also supports an underlying tier of high availability and automated placement options, for both control plane and worker nodes. 2 levels of scheduling and resource management are active.

Currently no automatic scheduling integration occurs, that is, Kubernetes is not aware of the underlying vSphere topology (sites, affinity groups, NUMA, etc.).

This session will explain the options to gain better performance, resource optimization and availability through tuning of vSphere, and Kubernetes configuration and labeling. This is applicable to any K8s distribution running on the vSphere stack.

Agenda

Kubernetes default scheduling

How it works

Utilizing Zones to improve scheduling

Using vSphere tags to define regions and zones – add cloud provider

What is NUMA?

How to solve potential issues with CPU and memory intensive workloads

Kubernetes default resource management

How it works

Extending the functionality of Kubernetes

Using vSphere DRS with Kubernetes

High Availability options

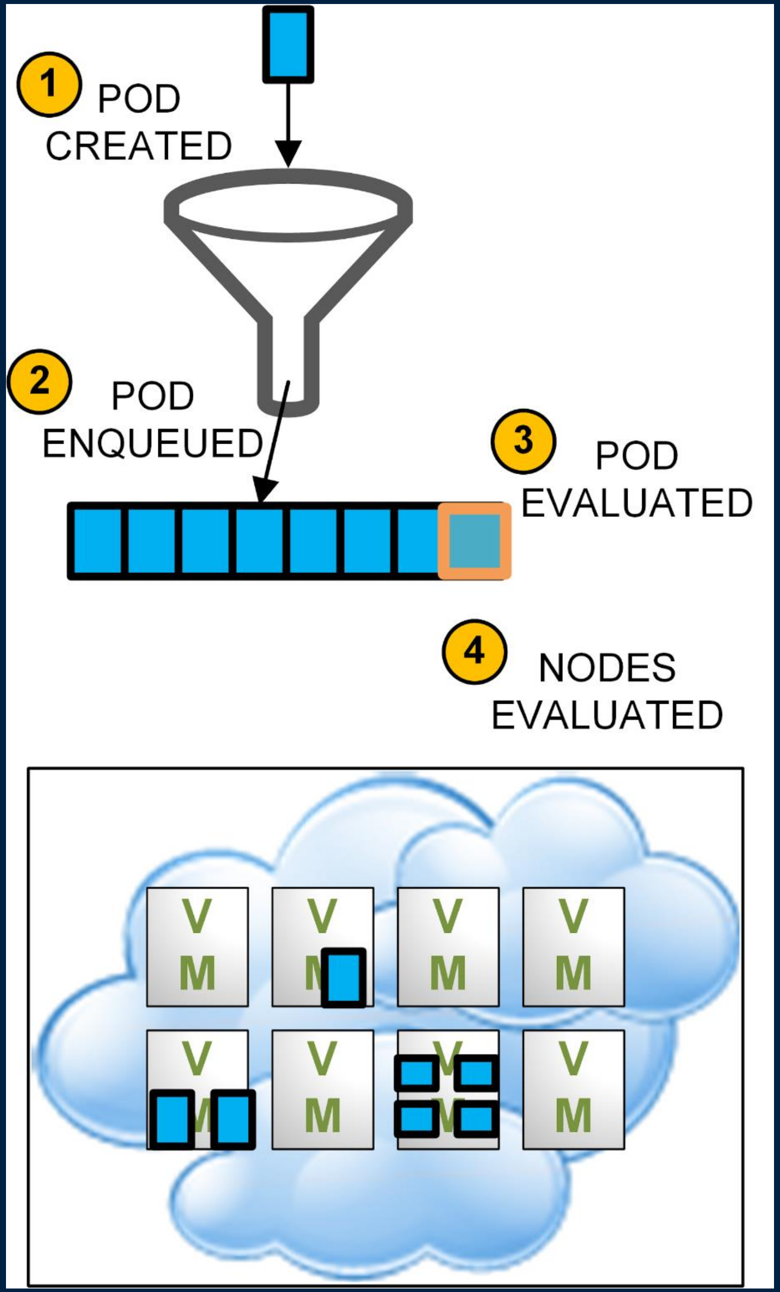
Using vSphere HA with Kubernetes

Kubenetes scheduling

What does the scheduler do:

As pod are created, they are place in a queue. (priority available in Beta)

The scheduler continuously pull pods off the queue, evaluates the pod's requirements, and assigns it to a worker node.



Kubenetes scheduling

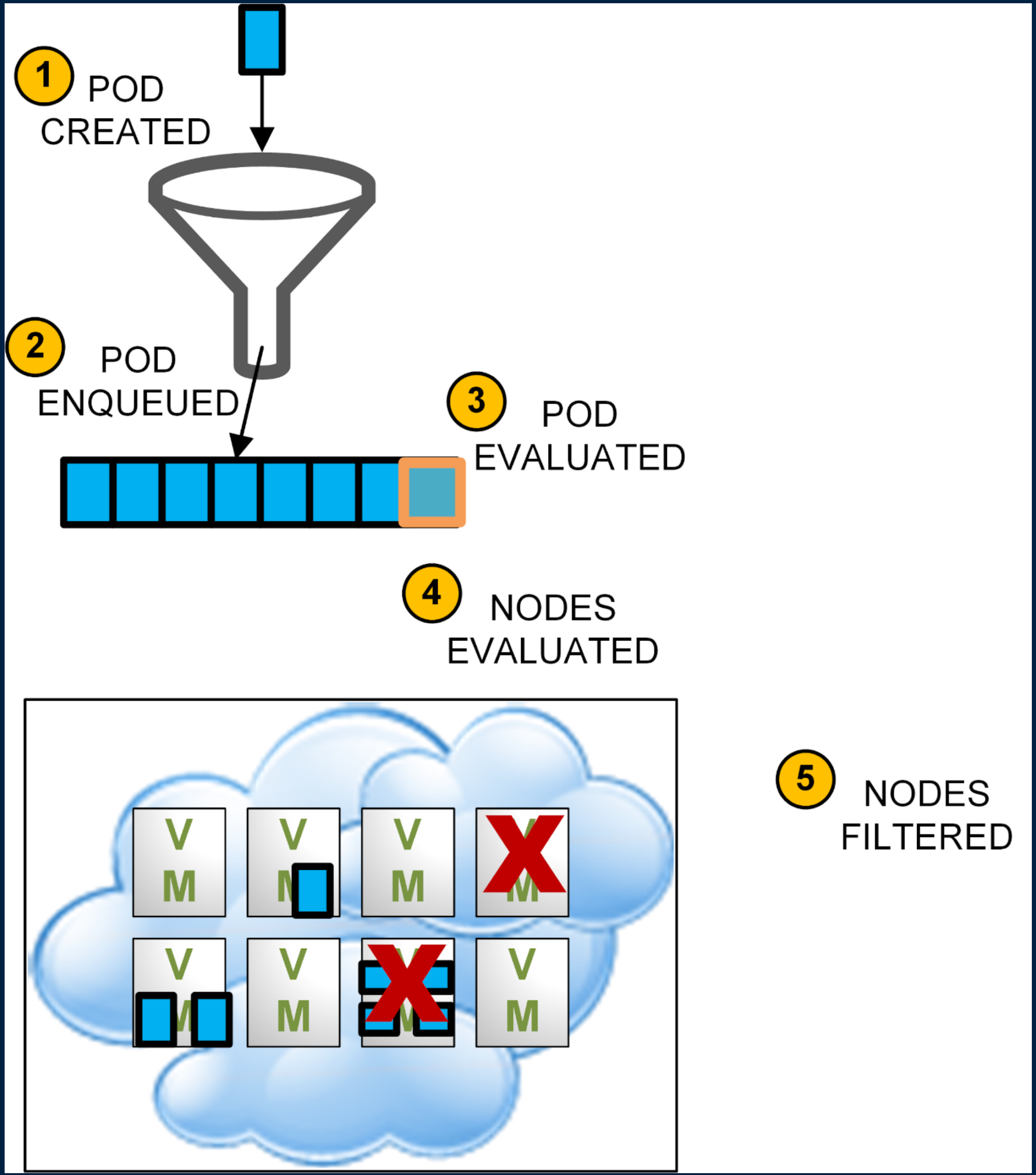
What does the scheduler do:

As pod are created, they are place in a queue. (priority available in Beta)

The scheduler continuously pull pods off the queue, evaluates the pod's requirements, and assigns it to a worker node.

Placement Decision Stages:

1. Filter out impossible worker nodes
   a. Filters are called *predicates* - extensible in code with a default list



Kubenetes scheduling

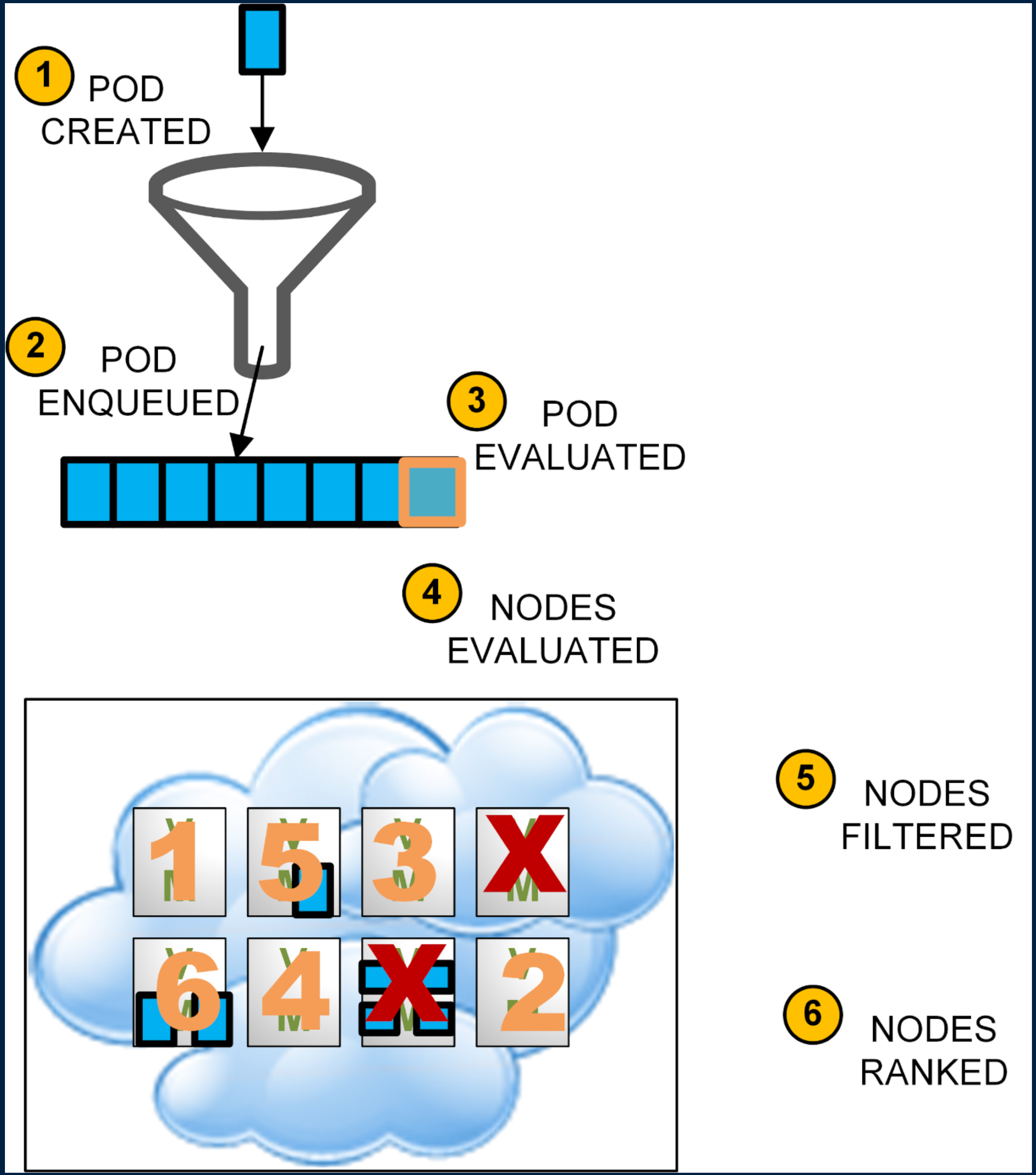
What does the scheduler do:

As pod are created, they are place in a queue. (priority available in Beta)

The scheduler continuously pull pods off the queue, evaluates the pod's requirements, and assigns it to a worker node.

Placement Decision Stages:

1. Filter out impossible worker nodes
   a. Filters are called *predicates* - extensible in code with a *default list*
2. Rank remaining nodes
   a. ranking is driven by priorities - this is extensible and configurable with a default list (e.g. zones)



Kubenetes scheduling

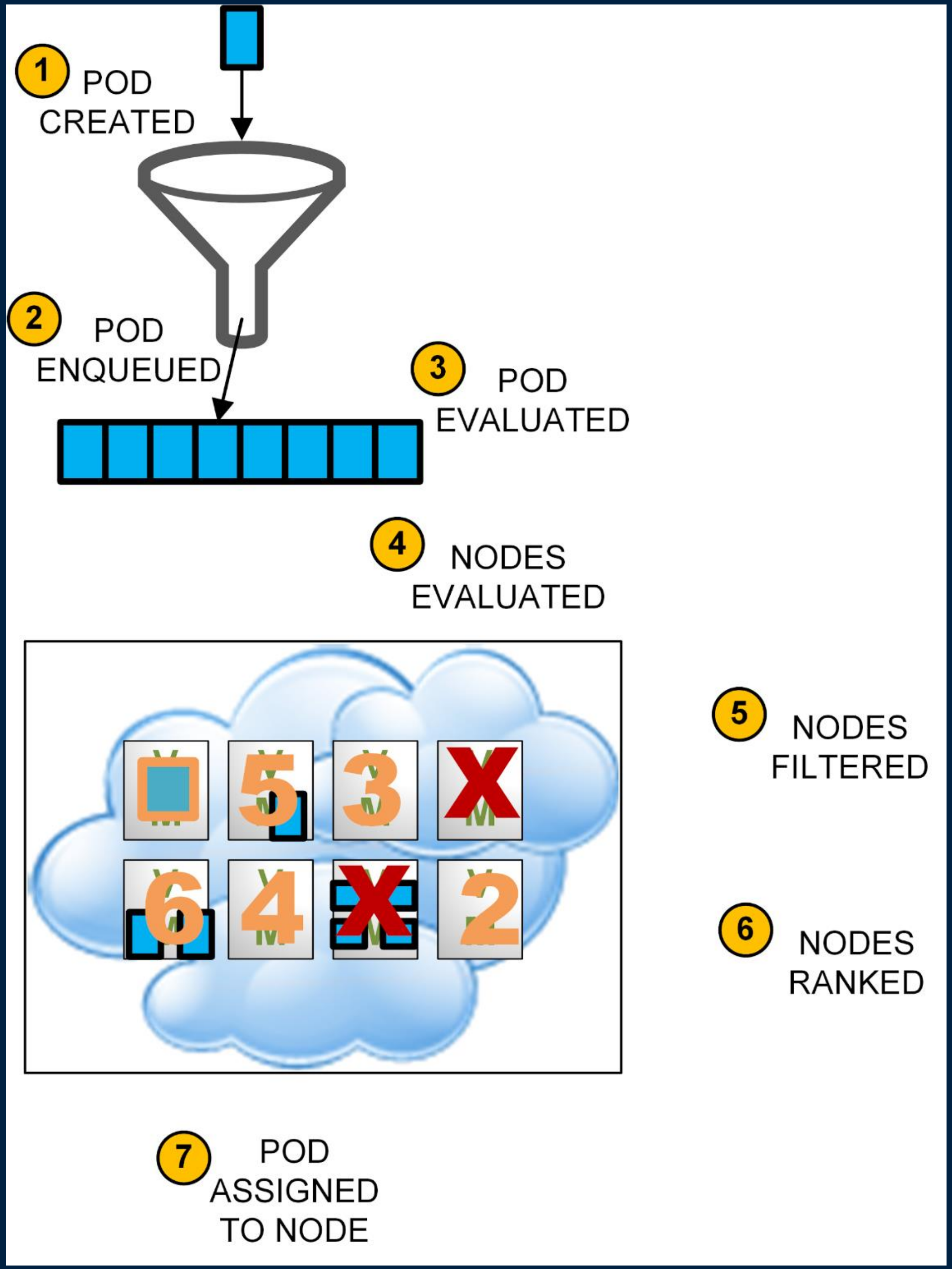
What does the scheduler do:

As pod are created, they are place in a queue. (priority available in Beta)

The scheduler continuously pull pods off the queue, evaluates the pod’s requirements, and assigns it to a worker node.

Placement Decision Stages:

1. Filter out impossible worker nodes
   a. Filters are called *predicates* - extensible in code with a *default list*
2. Rank remaining nodes
   a. ranking is driven by priorities - this is extensible and configurable with a default list (e.g. zones)



Scheduling modifiers

Node selector

constrain which nodes your pod is eligible to be scheduled on

based on <key: value> labels on the **node**

- **Some labels are automatically created, but you can add more**

specified as NodeSelector <key: value> in the Pod spec

Affinity

Pod can define rules based on node labels, or based on placement of other pods

Elements that influence pod placements

Zones – label nodes with failure zone/regions

Taints / Tolerations – mark nodes with arbitrary labels which could correspond to resource or whatever you like

Admission Controller – a wide variety are available, in validating and mutating classes

Why use Zones?

Kubernetes will automatically spread the pods in replication controllers or services across zones - to reduce the impact of zone failures

How it works:

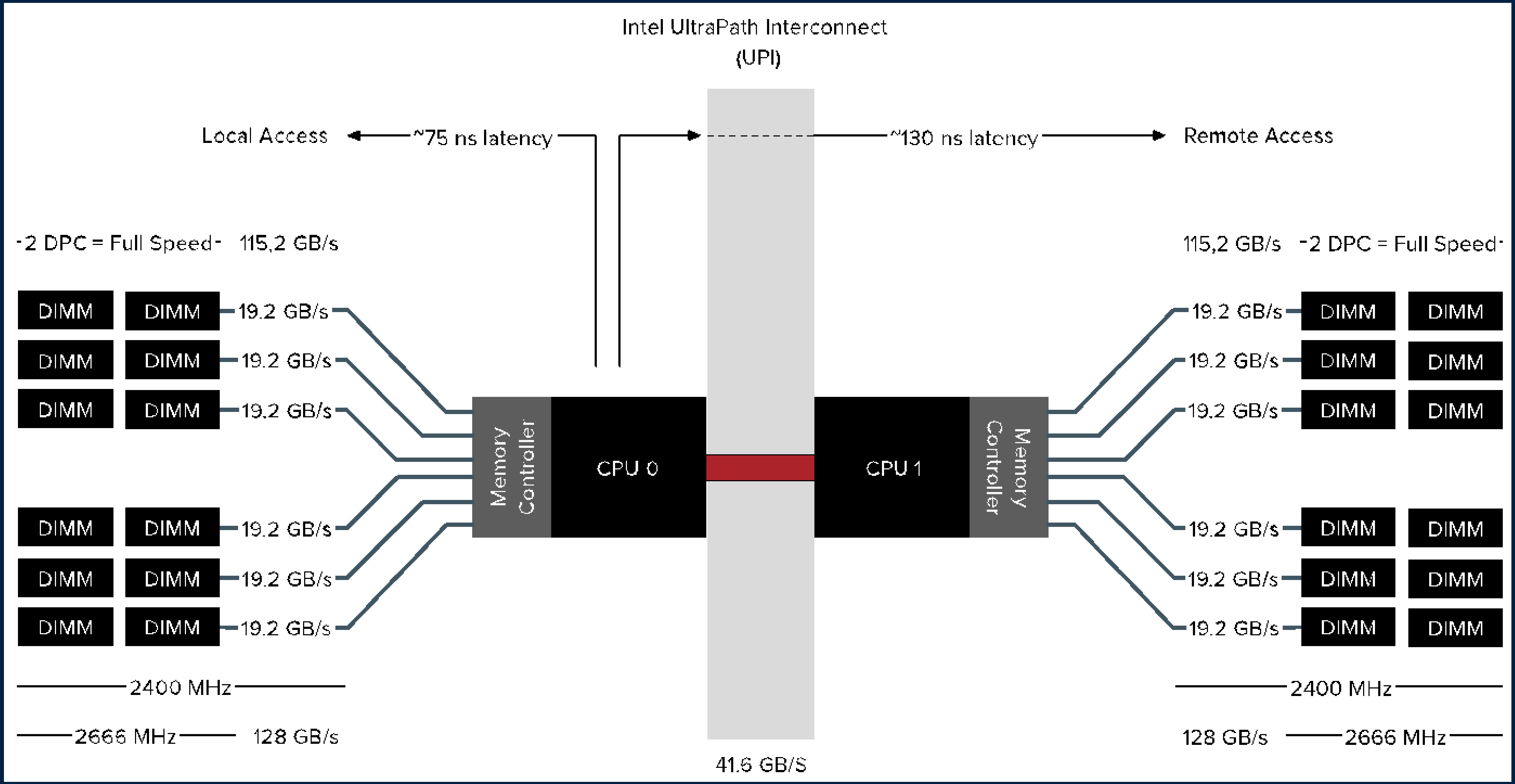
- Kubernetes supports running a single cluster in multiple failure zones.
- When nodes are started, labels are automatically added with zone information, based on tags pre-applied by a vSphere administrator.
- A developer can use these labels to place (selectors and required (anti-) affinity, i.e. *predicate*) and distribute (*priority*) pods
- Since Kubernetes v1.12 volume creation is coordinated with the scheduler (*VolumeZonePredicate* and topology-aware scheduling)

Limitations

- Zone spreading is a *priority* function not a *predicate*, i.e. it is a best-effort placement. If the zones in your cluster have uneven available resources due to node variations or unevenly distributed pre-existing workloads, this might prevent perfectly even spreading of your pods across zones (same applies to downscaling a deployment).
- The Kubernetes Zones feature is designed to intelligently place Pods on worker nodes. It does not place the nodes themselves within vSphere failure domains.

What is NUMA?

Non Uniform Memory Architecture



Why should you care about NUMA?

Memory intensive workloads

Nearly all database servers (e.g. Oracle, MongoDB), present a workload which will attempt to detect and consume as much of the system's memory as possible.

Where does this lead?

2 CPU Nodes – NUMA host

Node 0 32GB	Node 1 21GB

Unpredictable performance?

Swapping?

When Linux initially allocates a threads, it is assigned a preferred node, by default memory allocations come from this node the thread runs on, but can potentially come from other nodes with broad performance implications.

This basically comes down to a choice of whether you would rather have a fast cache, or a slower cache that is larger.

Many popular application runtimes (e.g. Java jre) have similar NUMA related issues.

How can NUMA issues be avoided?

Application can be modified / reconfigured?

- The application can be "wrapped" with a numactl command to interleave memory, or engage other options
  - potentially broad performance effects. (e.g interleaving get predictable albeit reduced performance)
- A cgroup aware version (e.g. Java jre v10) can be deployed
  - This is often not available – many were developed in a pre-container era

Active discussions regarding Kubernetes enhancements going on now in Resource Management Working Group – please join in

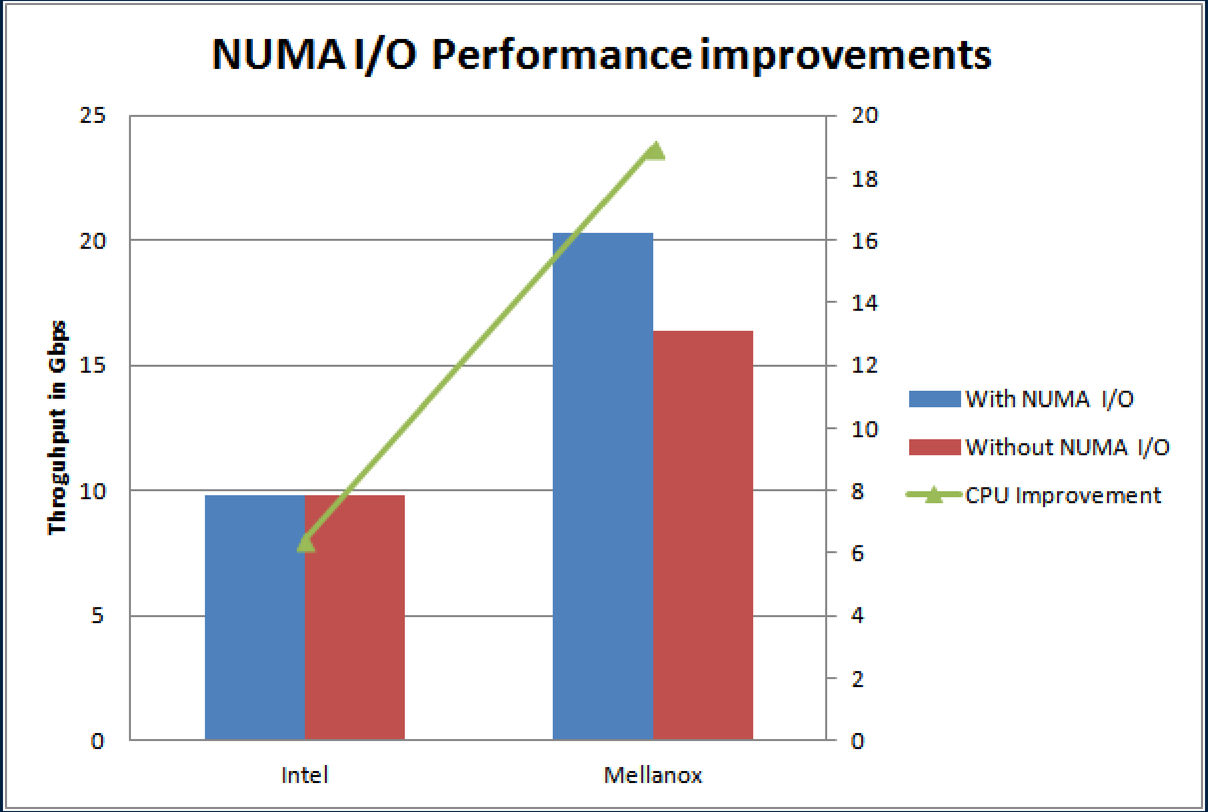
- See Issue #49964

Using a NUMA aware hypervisor to solve issues now

VM composition guidelines

- Assuming you workload fits with the footprint of a single node, compose worker node VMs as “walled gardens” corresponding to node size
    - Specify multiple cores per socket, not multiple sockets
- If you can’t fit in a single node because of core or memory requirements:
    - Minimize socket count to what is needed to meet requirements
    - Don’t assign an odd number of vCPUs
    - Never compose a VM larger than the number of physical cores

A NUMA aware hypervisor can have IO benefits too



For the vSphere hypervisor, there are advanced vNUMA settings, they rarely need to be changed from defaults. link

Kubernetes Resource Management

How it works

- Specified and “metered” on a per container basis
  - Requests
    - What a container is guaranteed to get – won’t be scheduled if not available
  - Limits
    - *Restrictions* are engaged when this is exceeded
  - Unmanaged by default
    - Mechanisms exist to allow a cloud provider or admin to supply a default and over-ride container specification outside an allowed range
- Supplemental “Metering” at the namespace level
  - Resource Quotas can be applied by an administrator at a namespace level
    - Requests
    - Limits
    - Numeric count of allowed instances of objects

Kubernetes Resource Management

What Resource are managed?

Pod + Namespace Level:

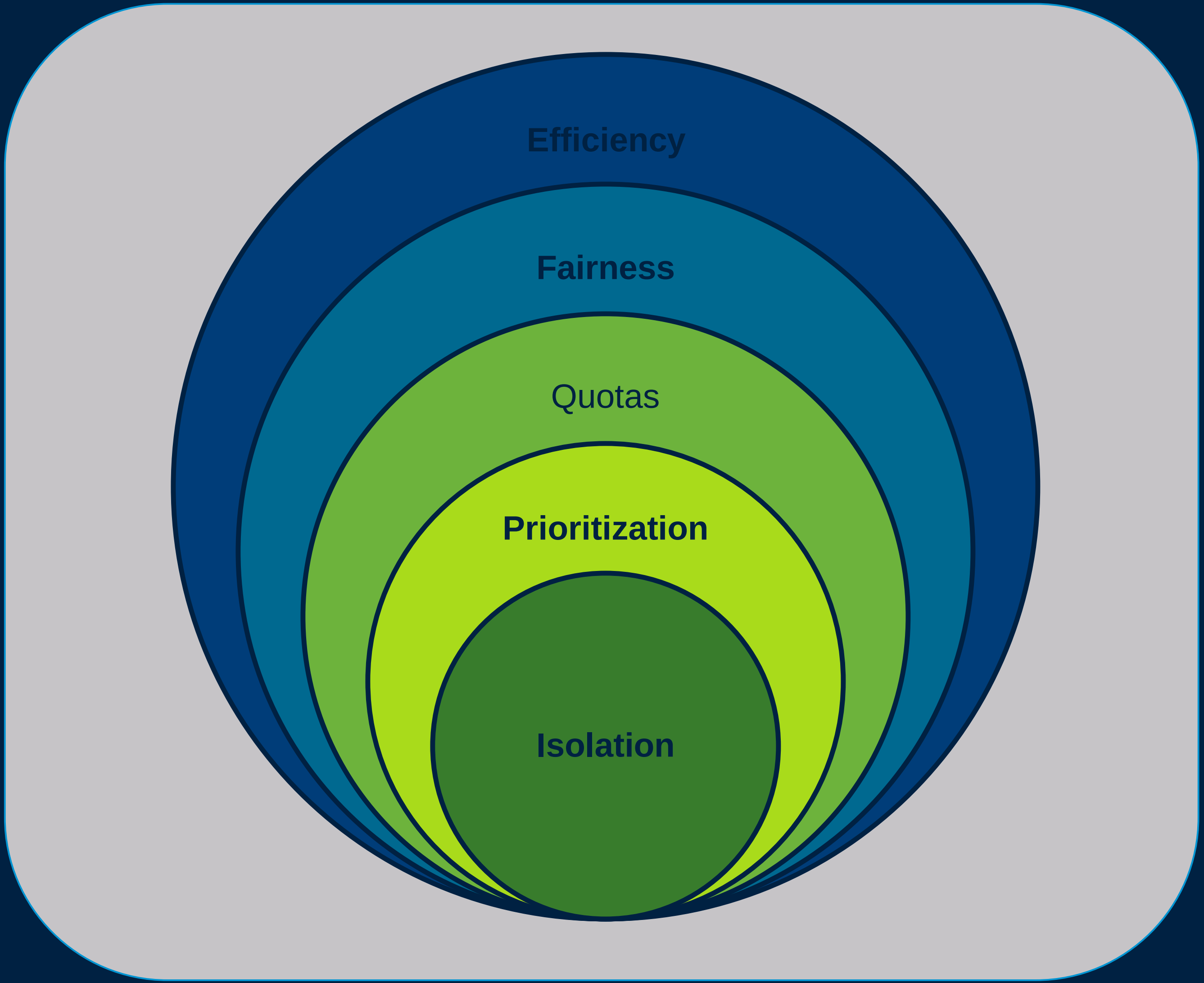
- CPU
  - Units are millicores, 2000m = 2 cores
- Memory
  - Mibibytes, 1000Mi = 1,048,576 bytes

Supplemental “Metering” at the namespace level

- Memory
- CPU
- Object counts
  - configmaps
  - persistentvolumeclaims
  - replicationcontrollers
  - secrets
  - services
  - loadbalancers

Kubernetes Default Resource Management

Goals



Kubernetes built-in resource management

Enforcement

Run time enforcement at worker node level

- CPU
  - “Compressible” = violation results in throttling
- Memory
  - “Uncompressible” = violation triggers “death penalty” of Pod hosting container

Scheduling time enforcement

- ResourceQuota admission controller will refuse to schedule a Pod that would violate limit
- After scheduling, running Pods are not affected by quota

Limitations

CPU measurement is in arbitrary units, not uniform across hosts and is a share not a guarantee

Where Resource Management enforcement takes place

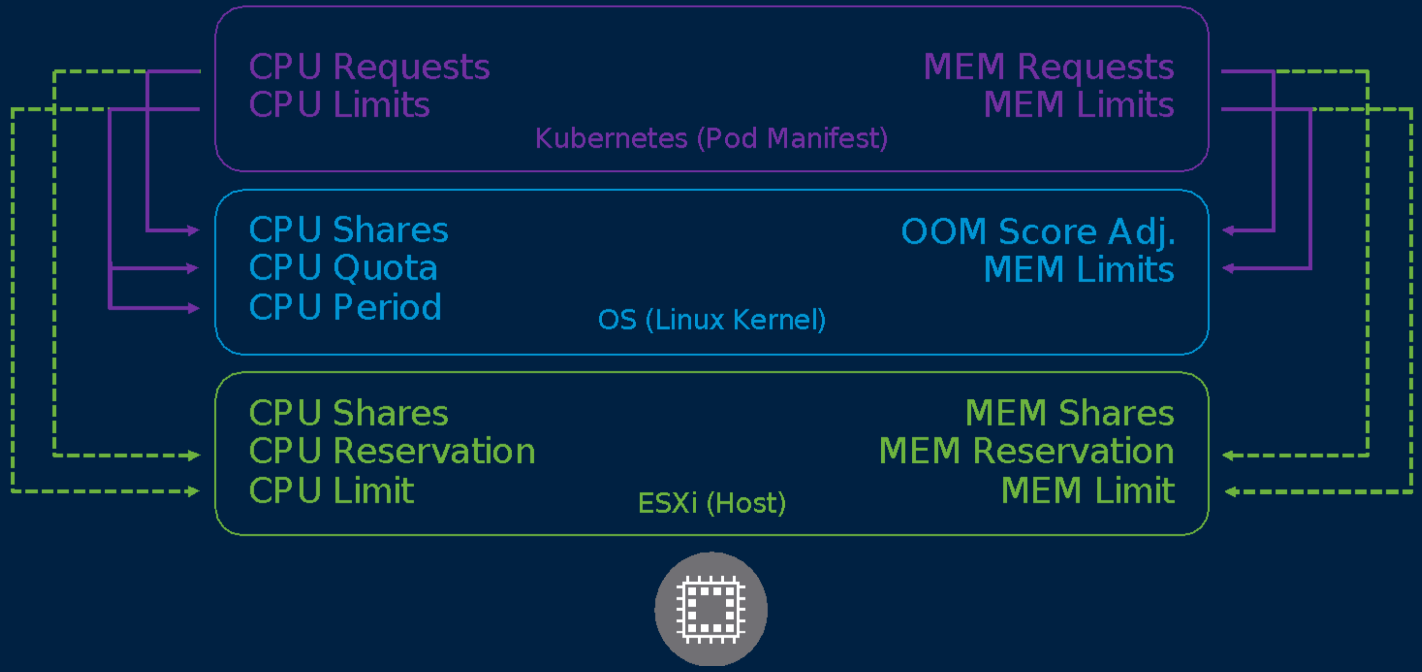
Kubernetes -> container runtime -> Linux -> hypervisor (optional)

Kubernetes control plane manages desired policy.

Enforcement passes Pod -> container runtime -> Linux OS

Cgroups are used to map Pod CPU and Memory Resources

- Note: Two Cgroups Drivers exist (cgroupfs [default], systemd)



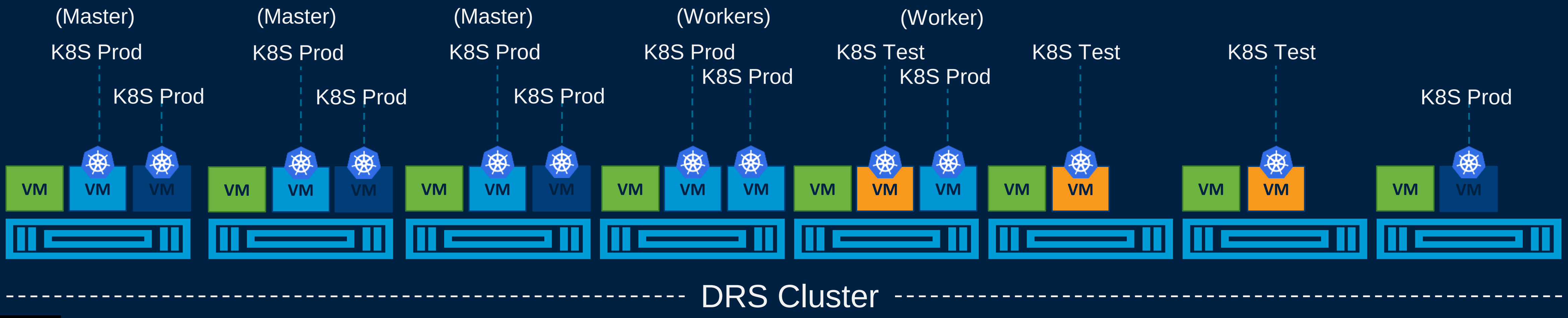
Supplement Kubernetes Resource Management with vSphere DRS

What is DRS?

The vSphere Distributed Resource Scheduler (DRS) is a load balancer for VMs deployed on a hypervisor cluster. It has advanced features that can provider actual guaranteed resource reservations, not just shares. It also incorporates health monitoring and IO awareness

Secure multi-tenant (multi-department) Kubernetes deployments

- with ability to have true guaranteed resource reservations (not just shares)
- with governed sharing of unutilized capacity for improved efficiency
- Allows maintenance with less service level disruption



Thank You

Questions?

remaining slides not presented to meet time constraints - included in published deck for reference

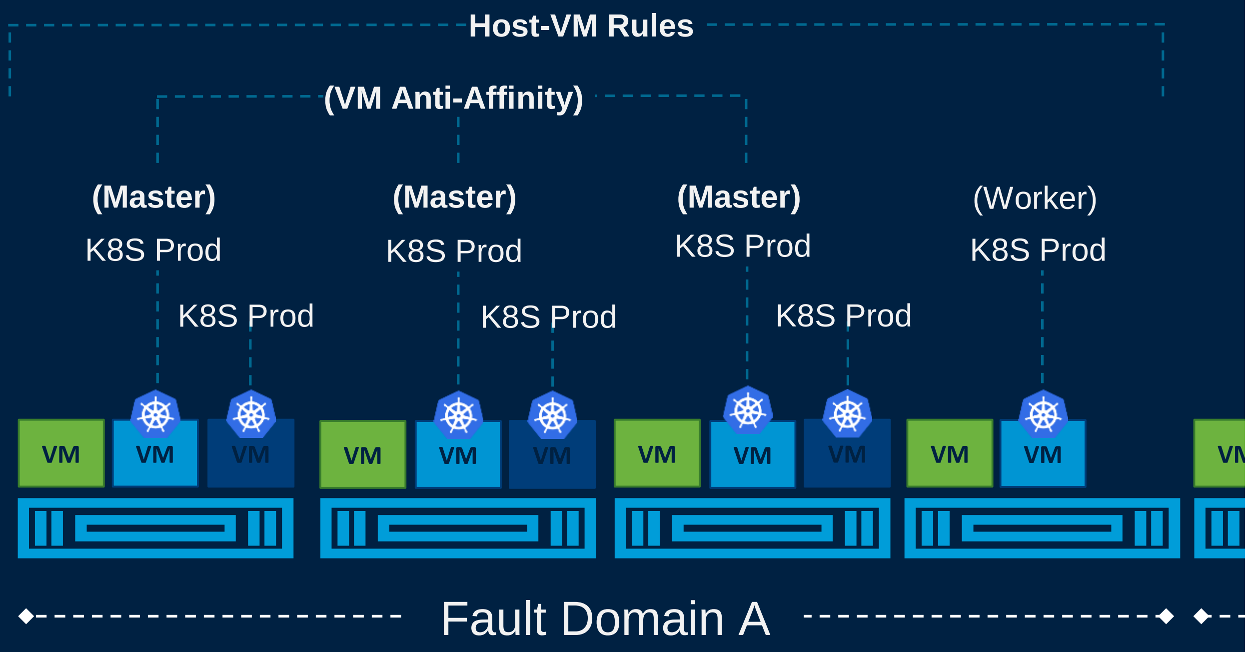
Open Issues (WIP)

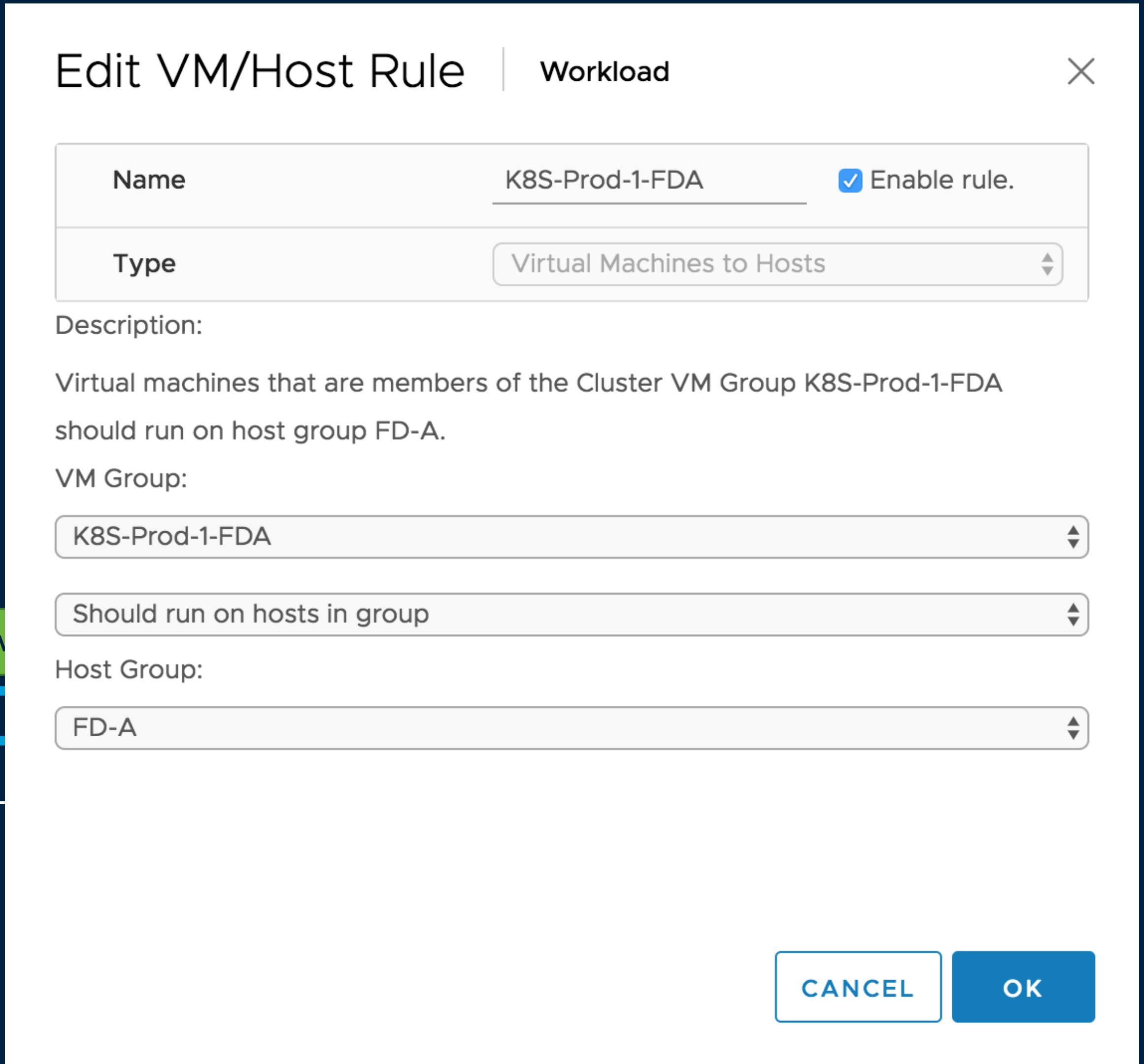
vSphere Cloud Provider should support implement Zones() interface #64021

vSphere Cloud Provider does not work when deployed across Zones with zone-local Storage #67703

Configuring VM affinity rules

Quorum dictates design





Extending Kubernetes with vSphere HA

What is HA?

Hosts in an HA cluster are health monitored and in the event of a failure, the virtual machines on a failed host are restarted on alternate hosts.

When running on hardware that supports health reporting, Pro-active failure avoidance can also be engaged. Example loss of a system cooling fan, degraded storage, or can trigger automated evacuation before host failure.

Deploying HA

A least 2 hypervisor hosts are required.

HA can be deployed independent of DRS, but the combination of the two in a cluster is recommended. This will enable load balancing and application of affinity/anti-affinity rules

Configuring HA restart priority

Ensure etcd, control plane starts first, and Prodsystems before others