KubeCon

CloudNativeCon

North America 2019

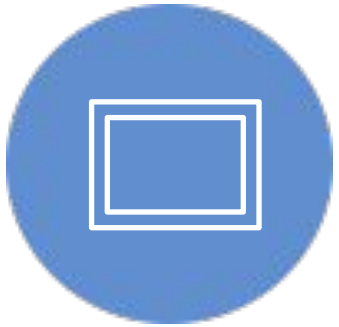
Staying in Tune: Optimize Kubernetes for Stability and Utilization

Randy Johnson, Field Engineer <jrandy@vmware.com>
Koushik Radhakrishnan, Field Engineer <radhakrishnk@vmware.com>

Why Kubernetes?
Kubernetes has potential

Declarative API
Software Interfacing with software

Self Healing
Autonomously converging on building blocks



Bin Packing
Approximation with performance guarantee

How it actually feels...

Goals

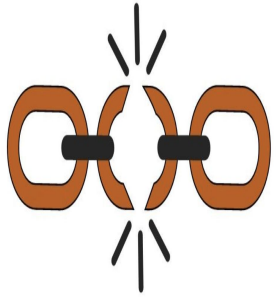
Stability

Utilization

Challenges

Kubernetes isn't configured for our app

Configuration is especially important as utilization increases



Disruptions occur, how can we maintain stability?

The "right" solution is a moving target

What can we do about it?

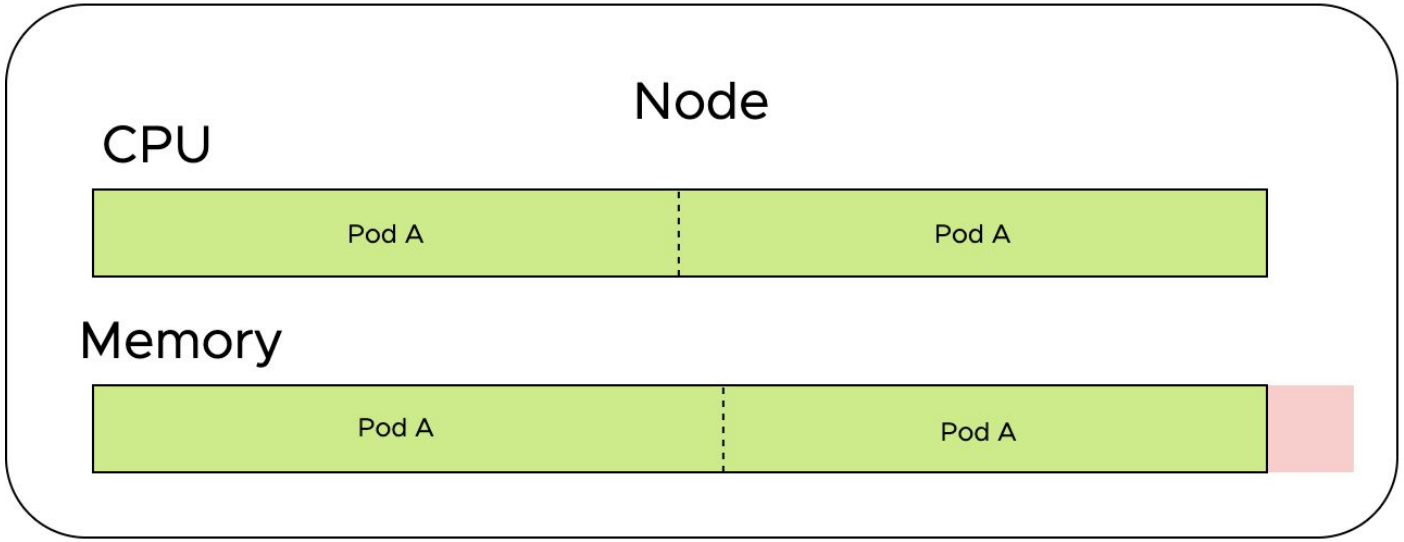
Limits and Requests

- Scheduling
- Overcommit
- Eviction

Allocatable Capacity

- Defaults
- Eviction Threshold
- Kube & System Reserved

Unbounded Resource Consumption




```
resources:
  limits:
    cpu:
    memory:
  requests:
    cpu:
    memory:
```

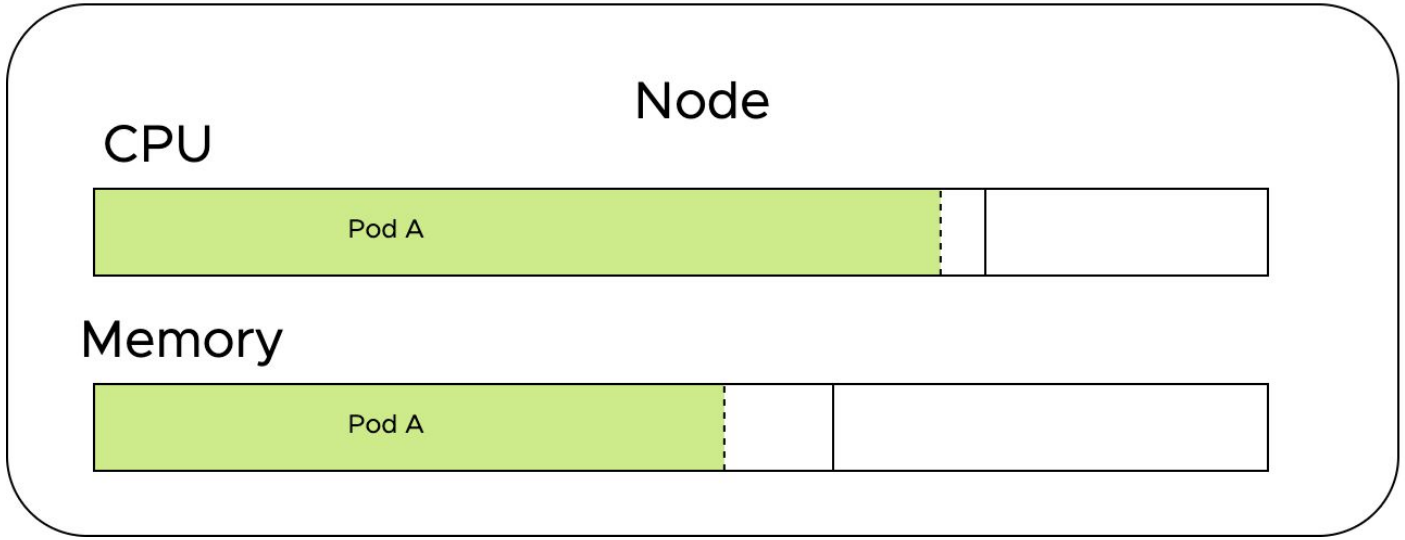
Pod A

```
resources:
  limits:
    cpu:
    memory:
  requests:
    cpu:
    memory:
```

Pod A

Incompressible, like water

Limits

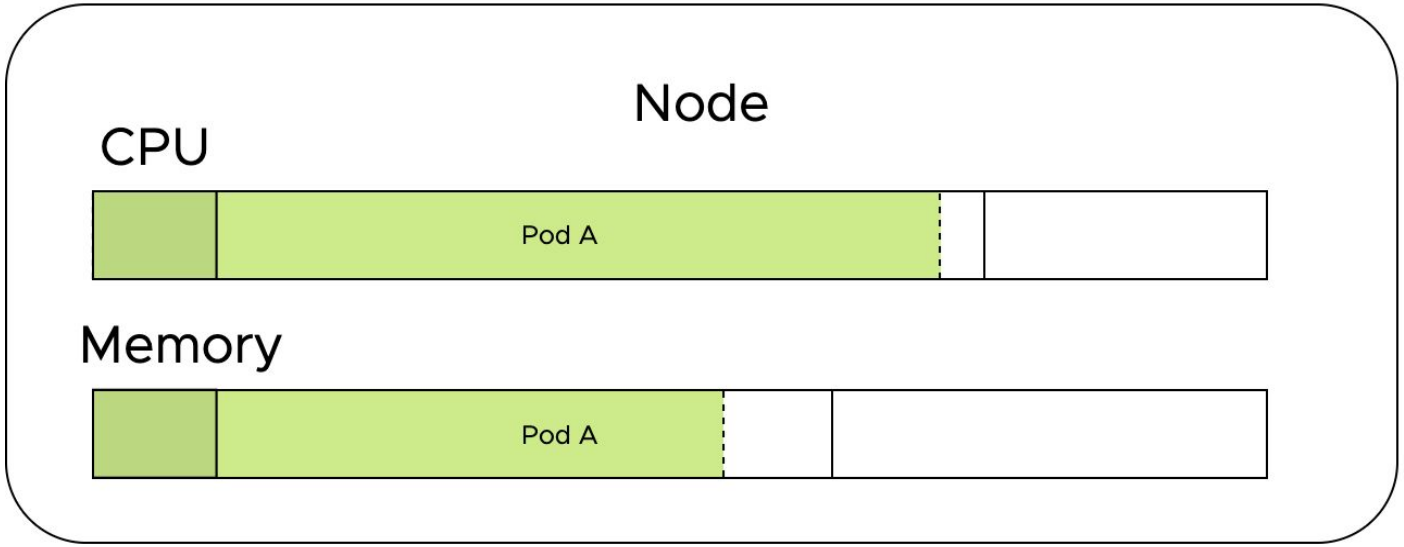


Pod A

```
resources:
  limits:
    cpu: "700m"
    memory: "600Mi"
  requests:
    cpu:
    memory:
```

https://kubernetes.io/docs/concepts/configuration/manage-compute-resources-container/

Requests

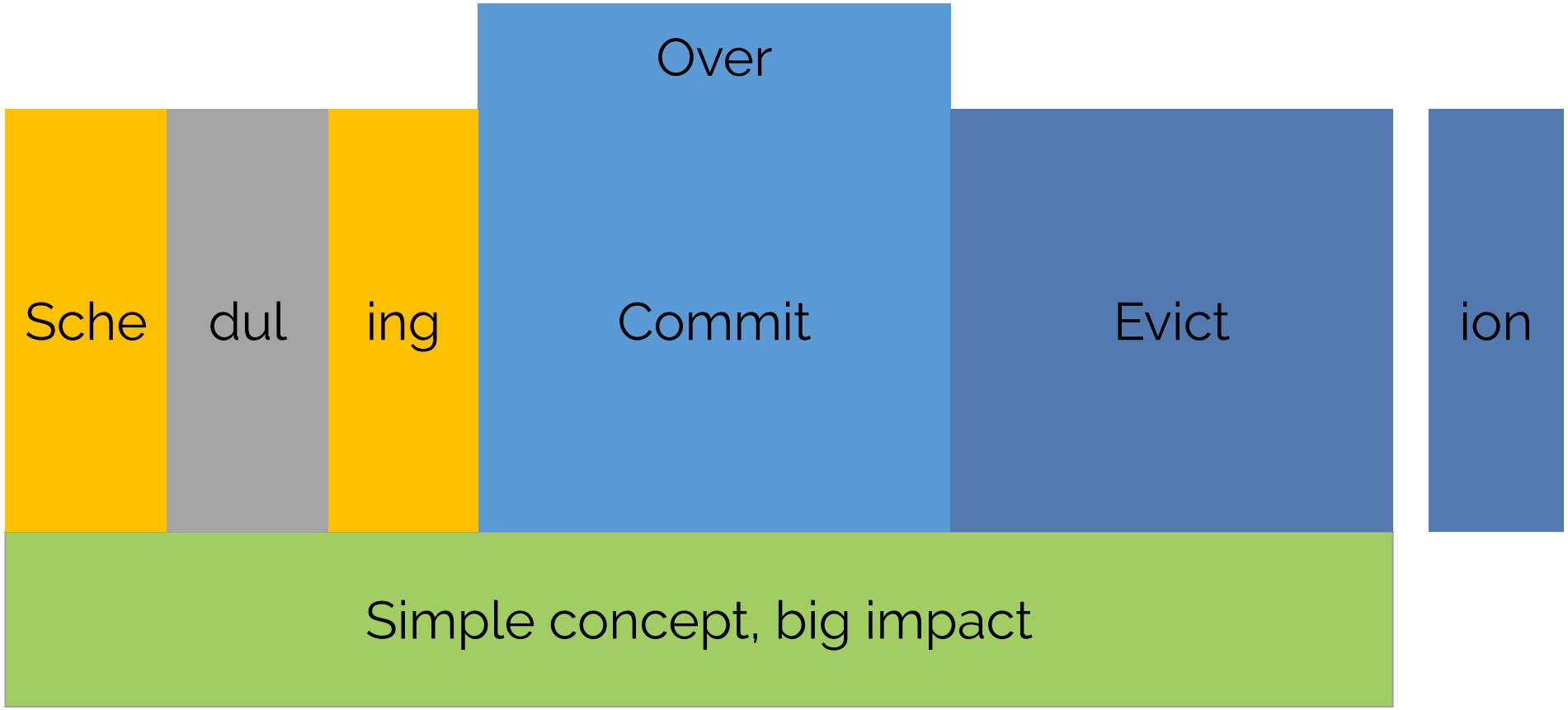


Pod A

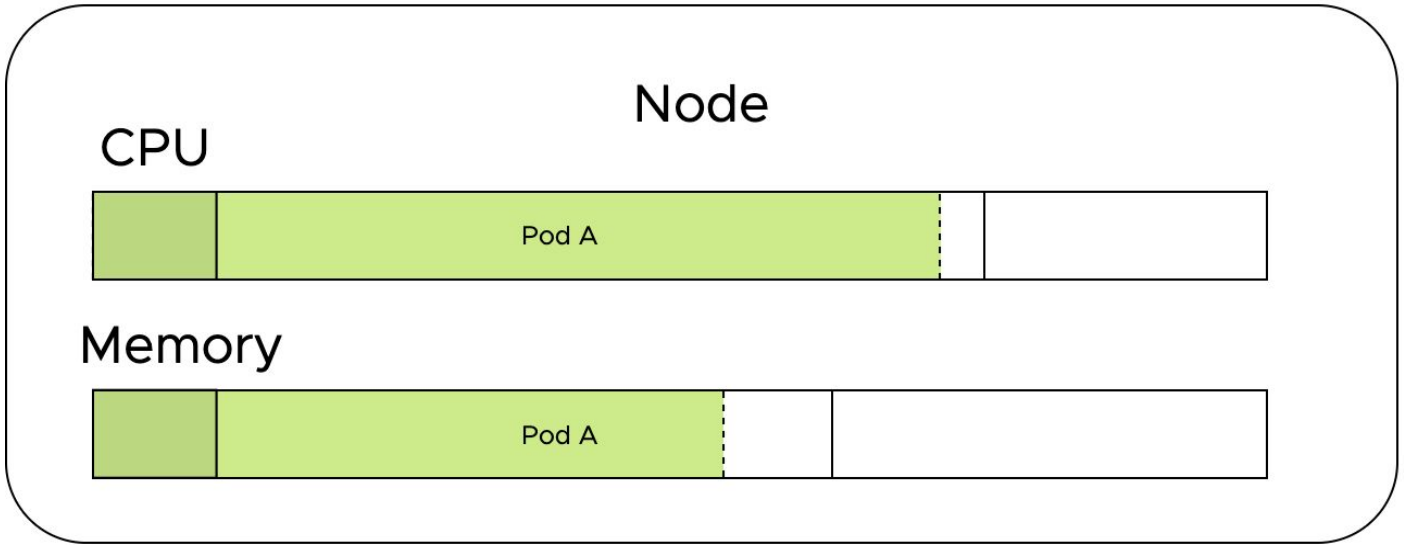
```
resources:
  limits:
    cpu: "700m"
    memory: "600Mi"
  requests:
    cpu: "100m"
    memory: "100Mi"
```

https://kubernetes.io/docs/concepts/configuration/manage-compute-resources-container/

Limits and Requests



Scheduling

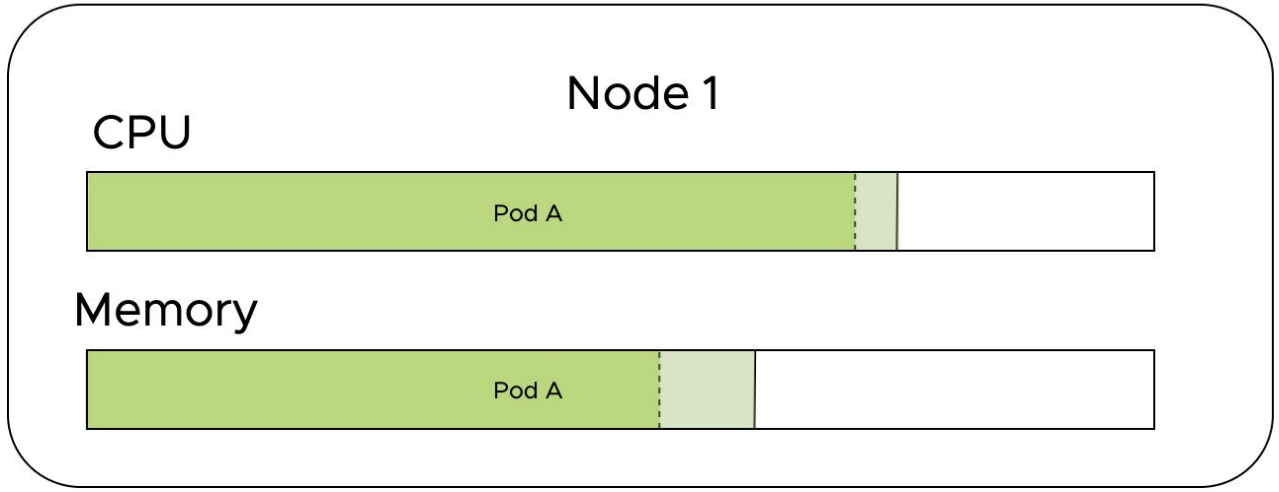


Pod A

```
resources:
  limits:
    cpu: "700m"
    memory: "600Mi"
  requests:
    cpu: "100m"
    memory: "100Mi"
```

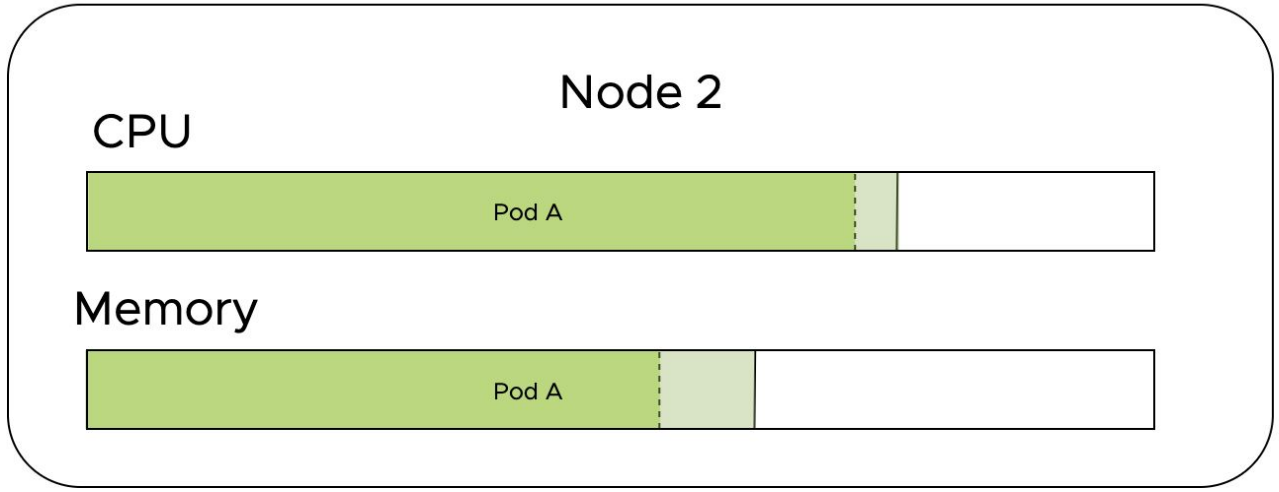
https://kubernetes.io/docs/concepts/scheduling/kube-scheduler/

Scheduling



Pod A

```
resources:
  limits:
    cpu: "700m"
    memory: "600Mi"
  requests:
    cpu: "700m"
    memory: "600Mi"
```

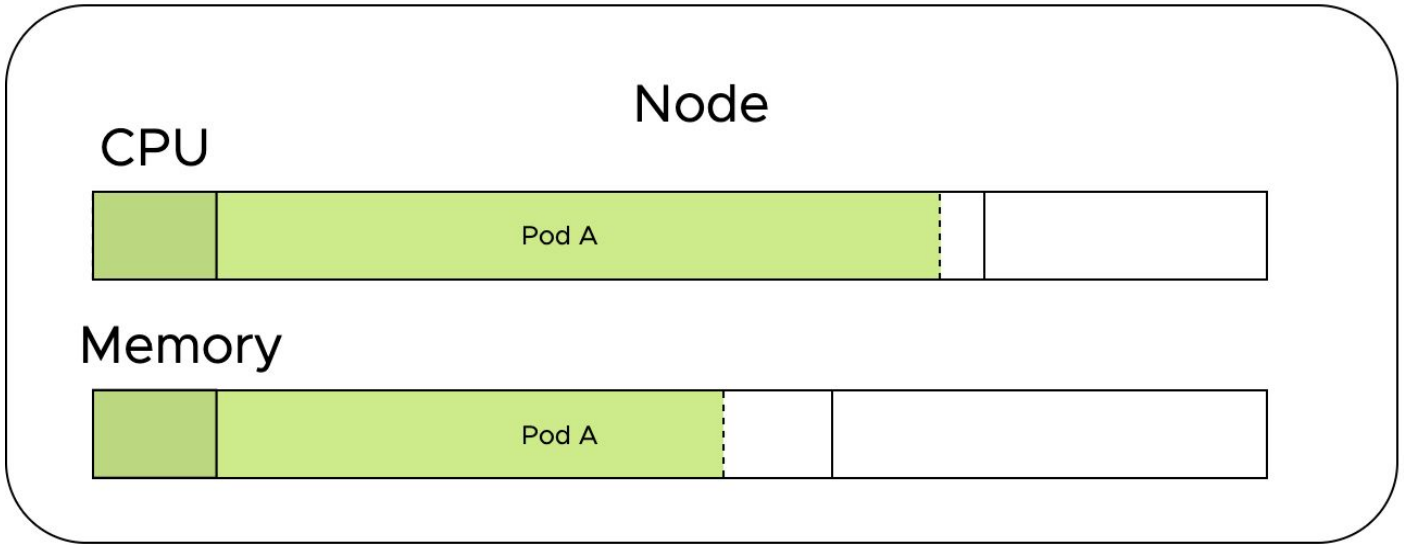


Pod A

```
resources:
  limits:
    cpu: "700m"
    memory: "600Mi"
  requests:
    cpu: "700m"
    memory: "600Mi"
```

https://kubernetes.io/docs/concepts/scheduling/kube-scheduler/

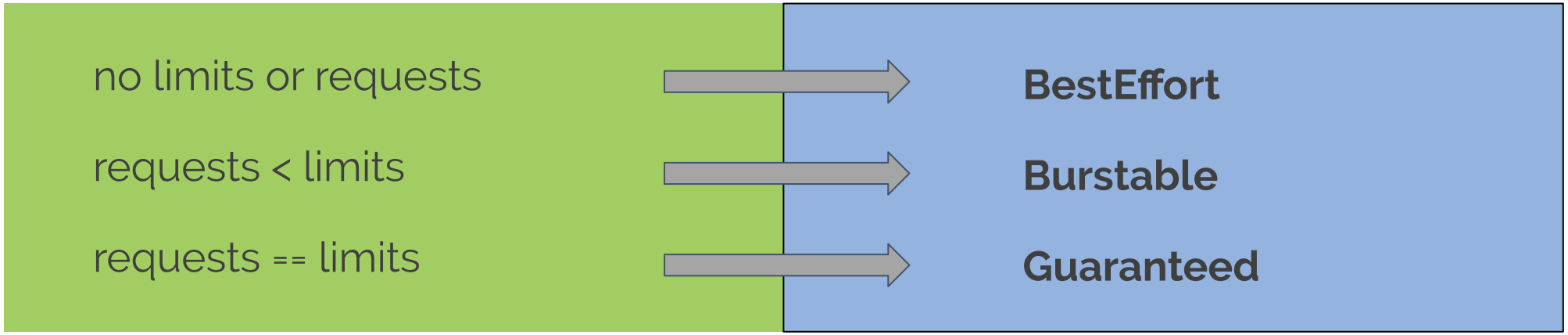
Overcommit



Pod A

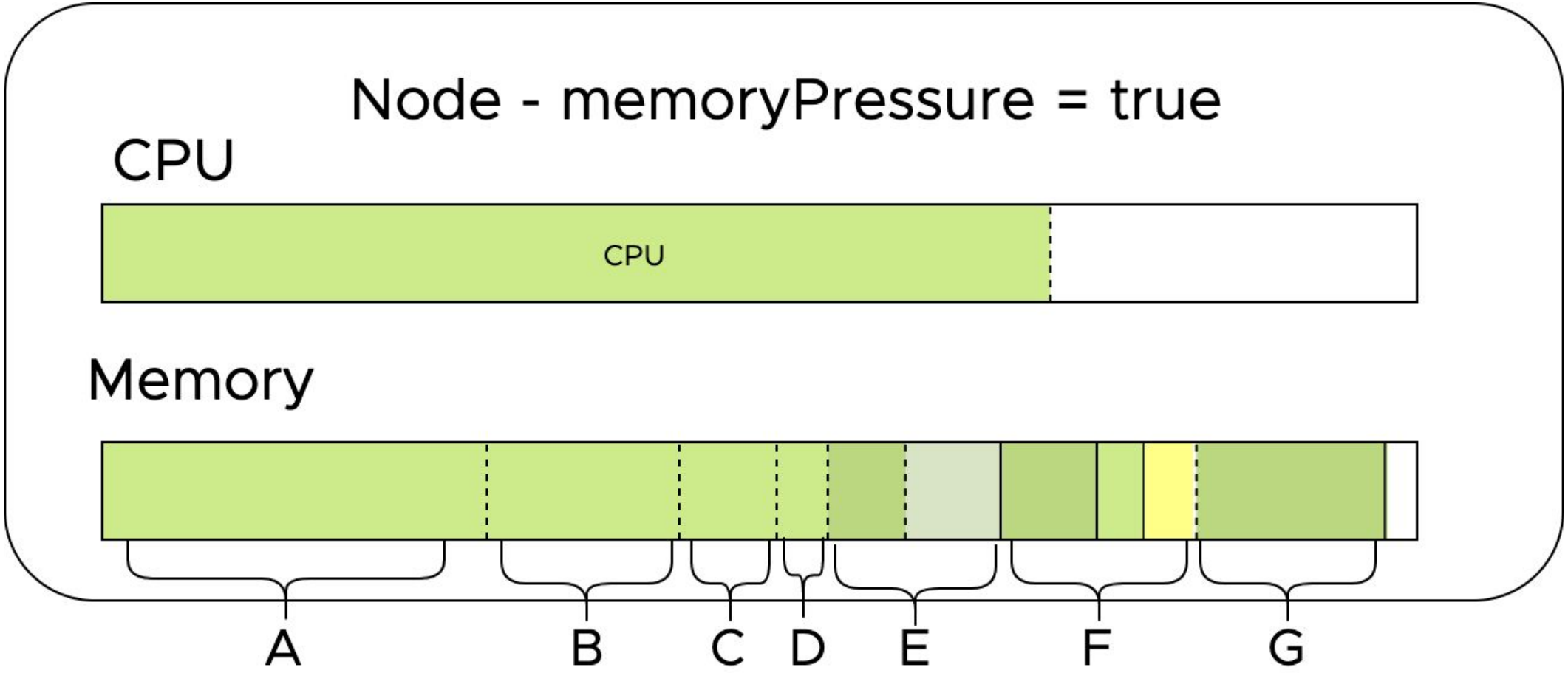
```
resources:
  limits:
    cpu: "700m"
    memory: "600Mi"
  requests:
    cpu: "100m"
    memory: "100Mi"
```

Quality of Service
Examples so far



https://kubernetes.io/docs/tasks/configure-pod-container/quality-service-pod/

Eviction



https://kubernetes.io/docs/tasks/administer-cluster/out-of-resource/#evicting-end-user-pods

Eviction

Eviction Order	QoS	Priority	Utilization	Utilization / Request	Pod Label
1	BestEffort	1	2%	N/A	D
2	BestEffort	2	5%	N/A	C
3	BestEffort	3	20%	N/A	A
4	BestEffort	3	10%	N/A	B
5	Burstable	2	N/A	2	F
6	Burstable	2	N/A	0.5	E
7	Guaranteed	1	N/A	1	G

https://kubernetes.io/docs/tasks/administer-cluster/out-of-resource/#evicting-end-user-pods

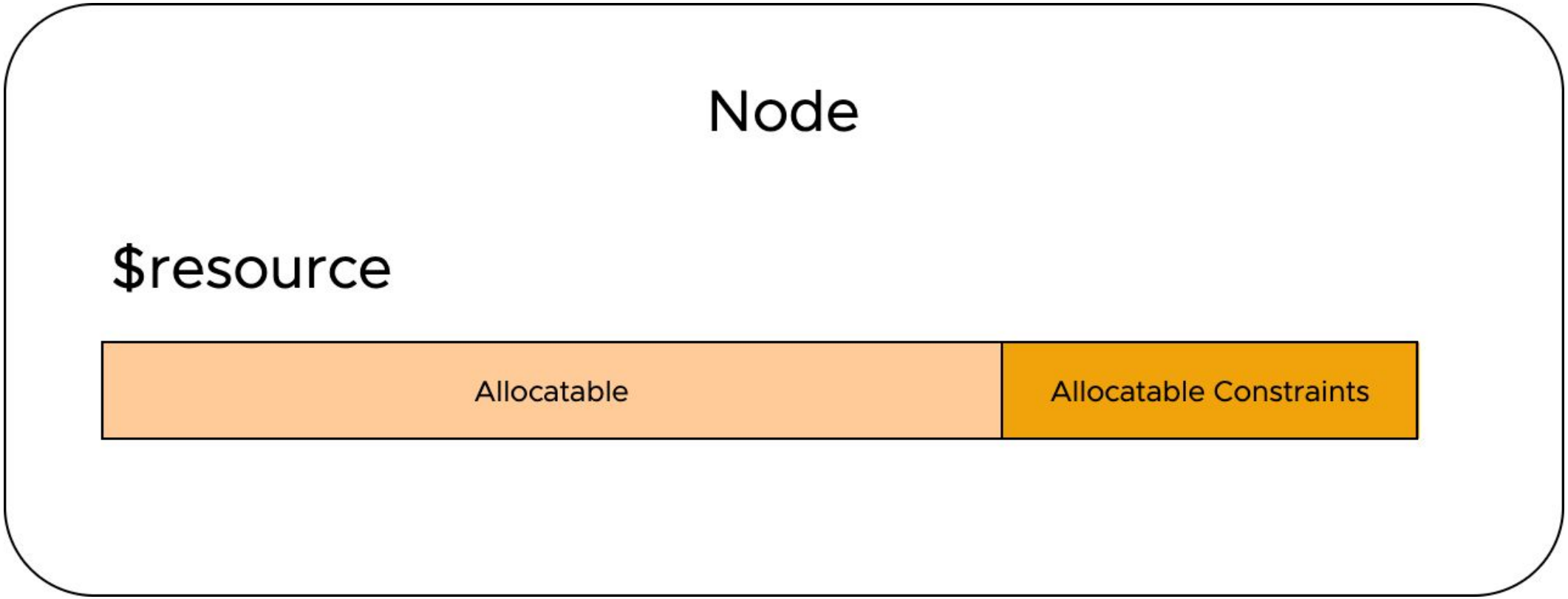
LimitRanges and Resource Quotas

- A *LimitRange* object enforces:
  - minimum
  - maximum
  - ratio
  - default
- A *ResourceQuota* enforces aggregate limits at the namespace level

https://kubernetes.io/docs/concepts/policy/limit-range/

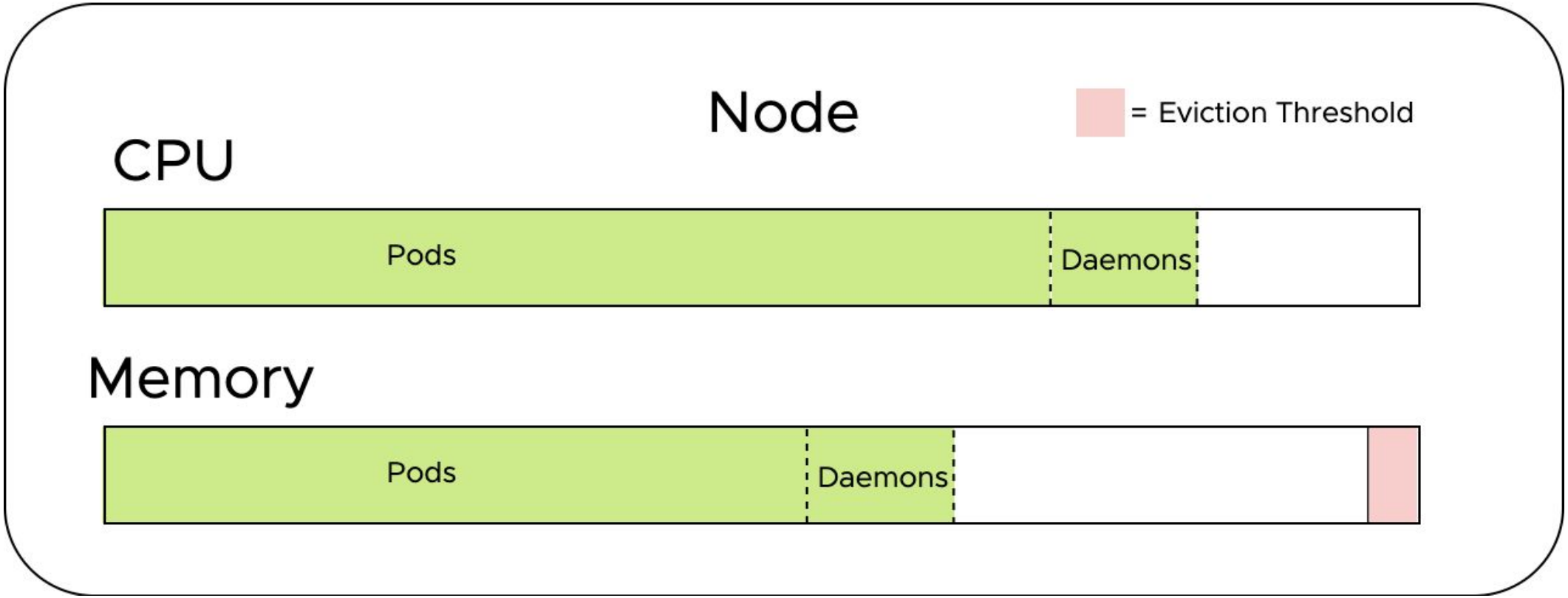
https://kubernetes.io/docs/concepts/policy/resource-quotas/

Allocatable



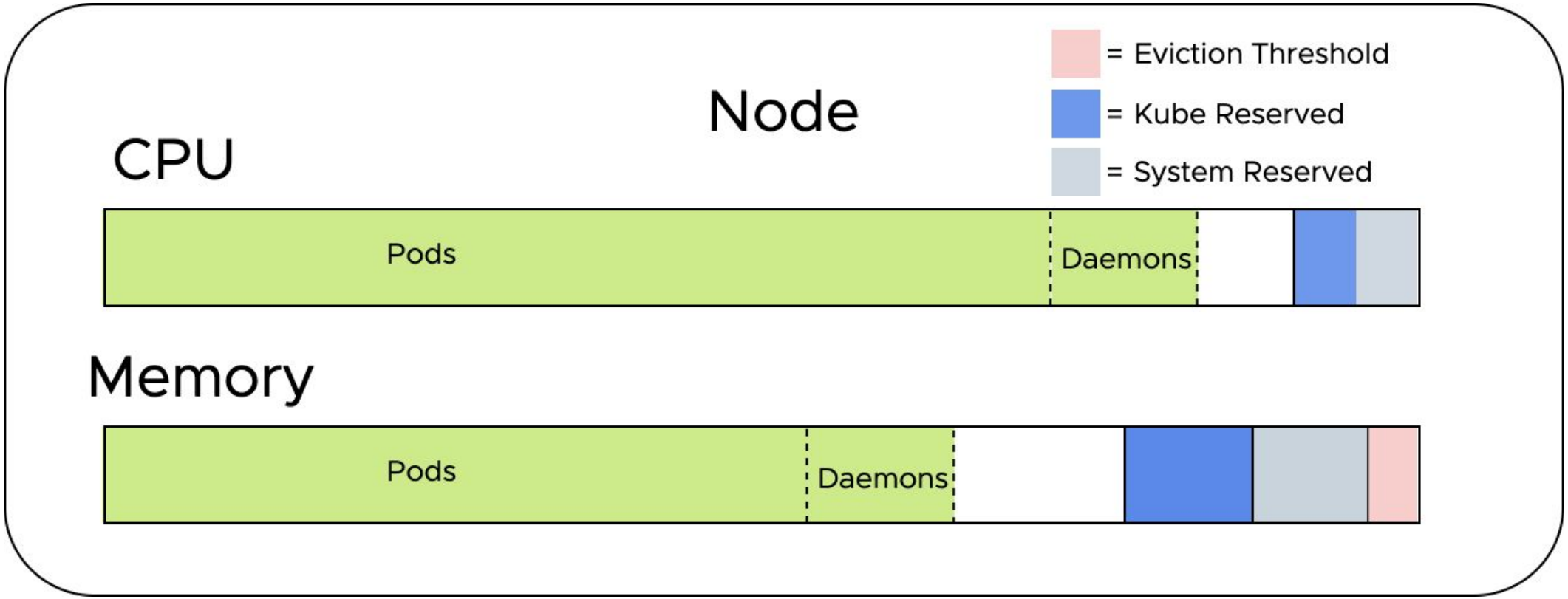
https://kubernetes.io/docs/tasks/administer-cluster/reserve-compute-resources/

Allocatable Constraints - Eviction Threshold



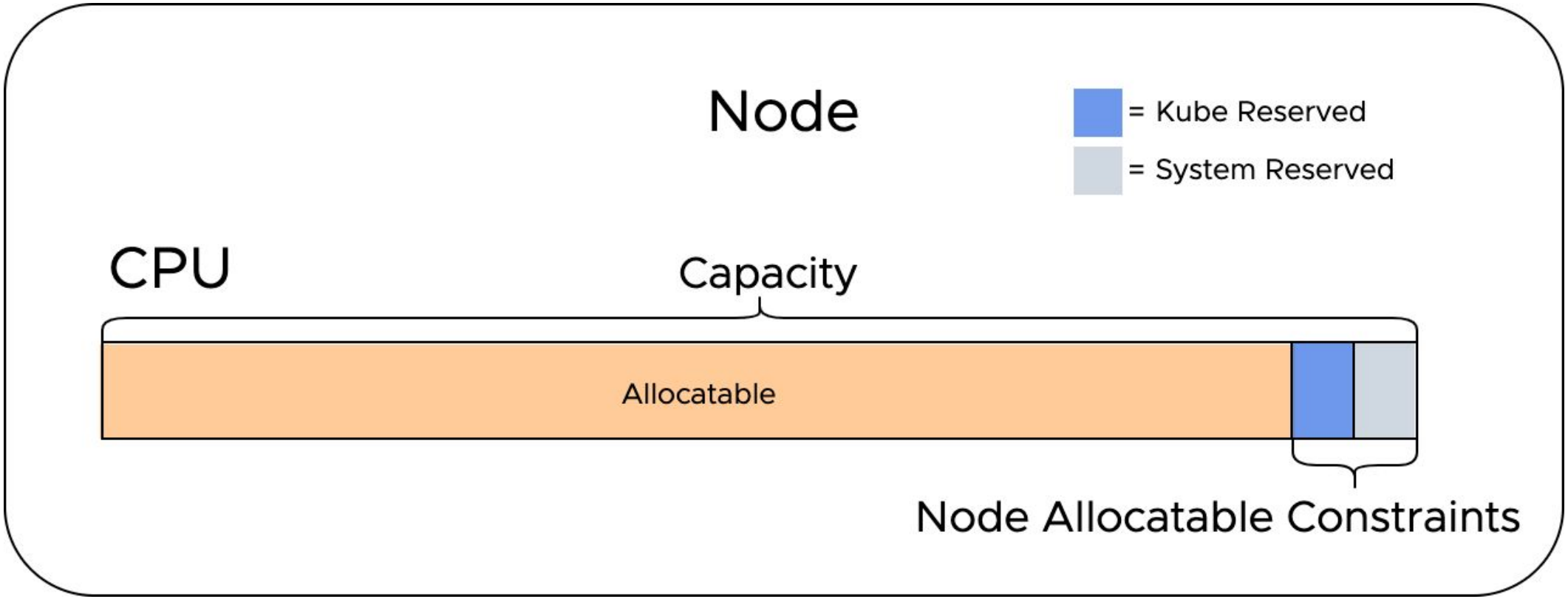
https://kubernetes.io/docs/tasks/administer-cluster/reserve-compute-resources/

Allocatable Constraints - Kube & System Reserved



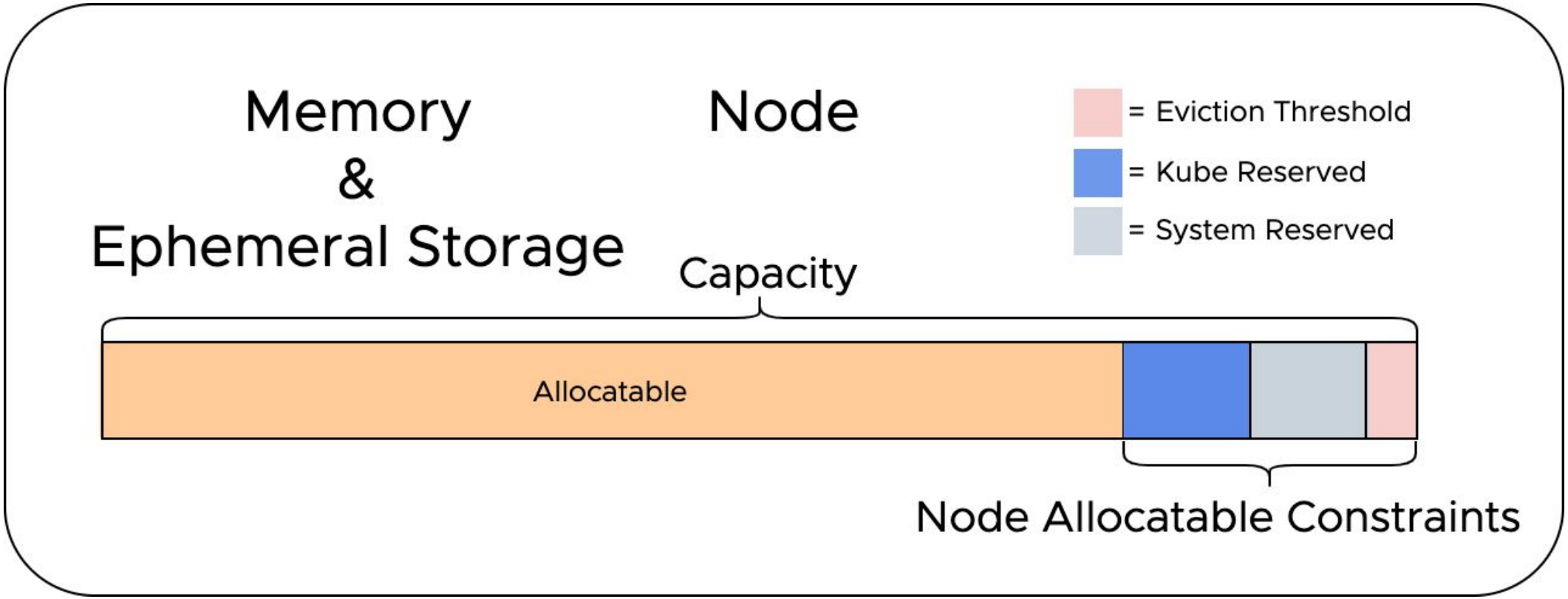
https://kubernetes.io/docs/tasks/administer-cluster/reserve-compute-resources/

Allocatable CPU



https://kubernetes.io/docs/tasks/administer-cluster/reserve-compute-resources/

Allocatable Memory & Ephemeral Storage



https://kubernetes.io/docs/tasks/administer-cluster/reserve-compute-resources/

Key Considerations

- Is the scheduler prepared to make informed decisions?
  - LimitRange
  - utilization / request
- Is overcommit configured properly?
  - limits too high causes eviction
  - limits too low causes container restarts or throttling
- How close are we to triggering eviction?
  - allocatable - pod_utilization > 0
  - capacity - eviction_threshold - total_utilization > 0
- What if we frequently encounter eviction caused by
  - utilizing allocatable?
    - Tune limits & requests to adjust overcommit
  - crossing an eviction threshold?
    - Tune kube-reserved & system-reserved

Max Node Utilization

$$maxUtilization(n,\ f, c) = \frac{n - f - c}{n}$$

$$for\ \ n \geq 2, f \geq 0, c \geq 0$$

Defaults

- Kubelet (as of v1.16.2) - https://godoc.org/k8s.io/kubelet/config/v1beta1
  - implied defaults:

```
--eviction-hard=memory.available<100Mi
--housekeeping-interval=10s
--eviction-pressure-transition-period=5m
--max-pods=110
```

  - flags to consider:

```
--kube-reserved
--system-reserved
--eviction-soft
--eviction-soft-grace-period
```

- Docker implied default - https://docs.docker.com/config/containers/live-restore/

```
{
  "live-restore": false
}
```

Summary

Limits and Requests	Allocatable Capacity
• Scheduling • Overcommit • Eviction	• Defaults • Eviction Threshold • Kube & System Reserved

Staying in Tune: Optimize Kubernetes for Stability and Utilization

Tell us your experience

Randy Johnson, Field Engineer
<jrandy@vmware.com>

Koushik Radhakrishnan, Field Engineer
<radhakrishnk@vmware.com>