
KubeCon
CloudNativeCon
Europe 2023
TiKV

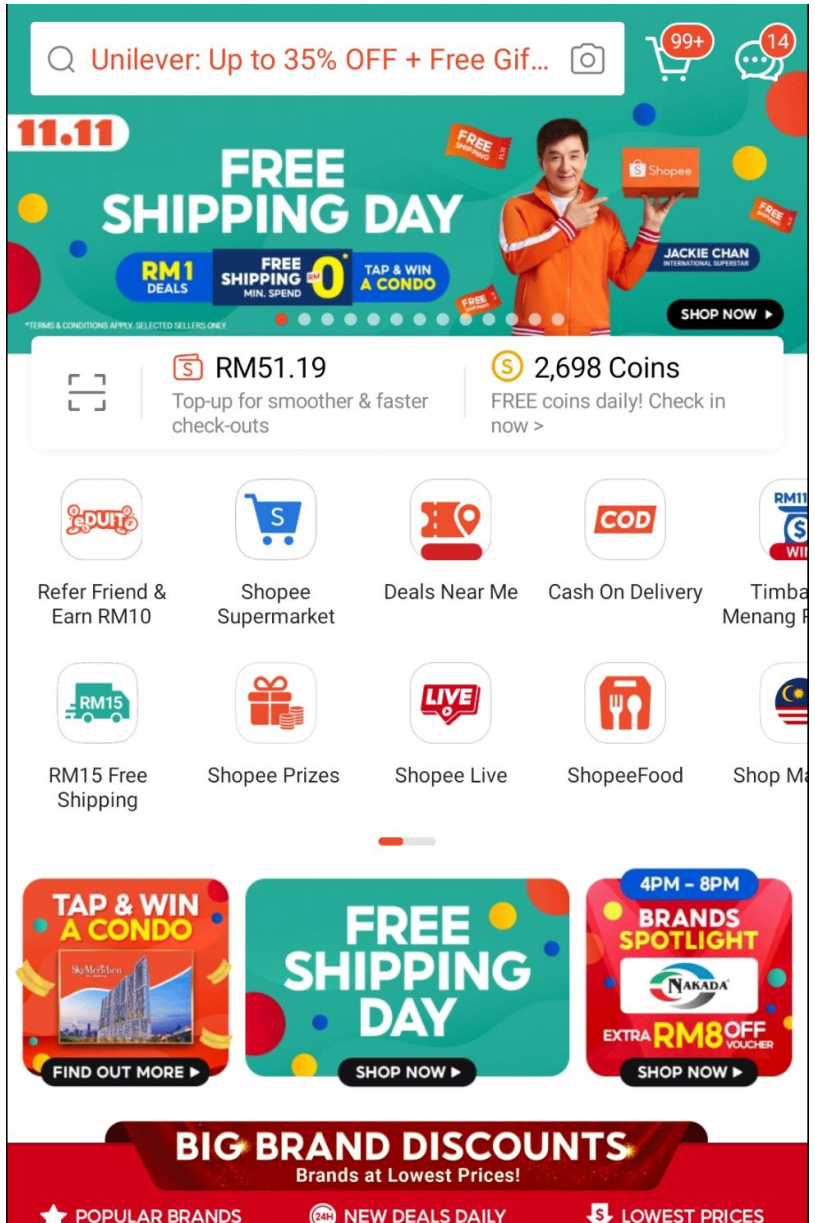
Colocating Hadoop YARN with Kubernetes to Save Massive Costs on Big Data

Irvin Lim
Xiang Hailin

Engineering Infrastructure, Shopee

Who We Are

Shopee



Who We Are

Leading e-commerce platform in Southeast Asia, Taiwan and Brazil

#1 Shopping App in Southeast Asia and Taiwan

By average Monthly Active Users and total time spent in-app

#1 Shopping App in Brazil

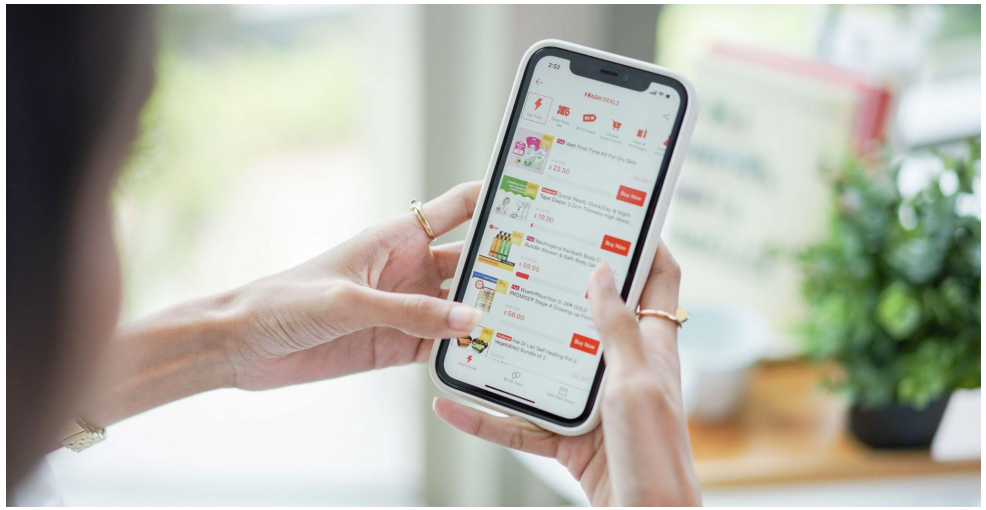
By average Monthly Active Users and total time spent in-app

Singapore Malaysia Thailand Taiwan

Indonesia Vietnam Philippines Brazil

Who We Are

We continue to grow and scale, building on our strong brand recognition across the region



#1 Shopping App Globally

By total time spent in-app on Google Play

#2 Shopping App Globally

By average Monthly Active Users on Google Play

#5 Best Brand Globally

According to YouGov 2022 Global Best Brand Rankings

Rank	Brand name
1	Samsung
2	Google
3	YouTube
4	Netflix
5	Shopee
6	WhatsApp
7	Toyota
8	Colgate
9	Mercedes-Benz
10	Lidl

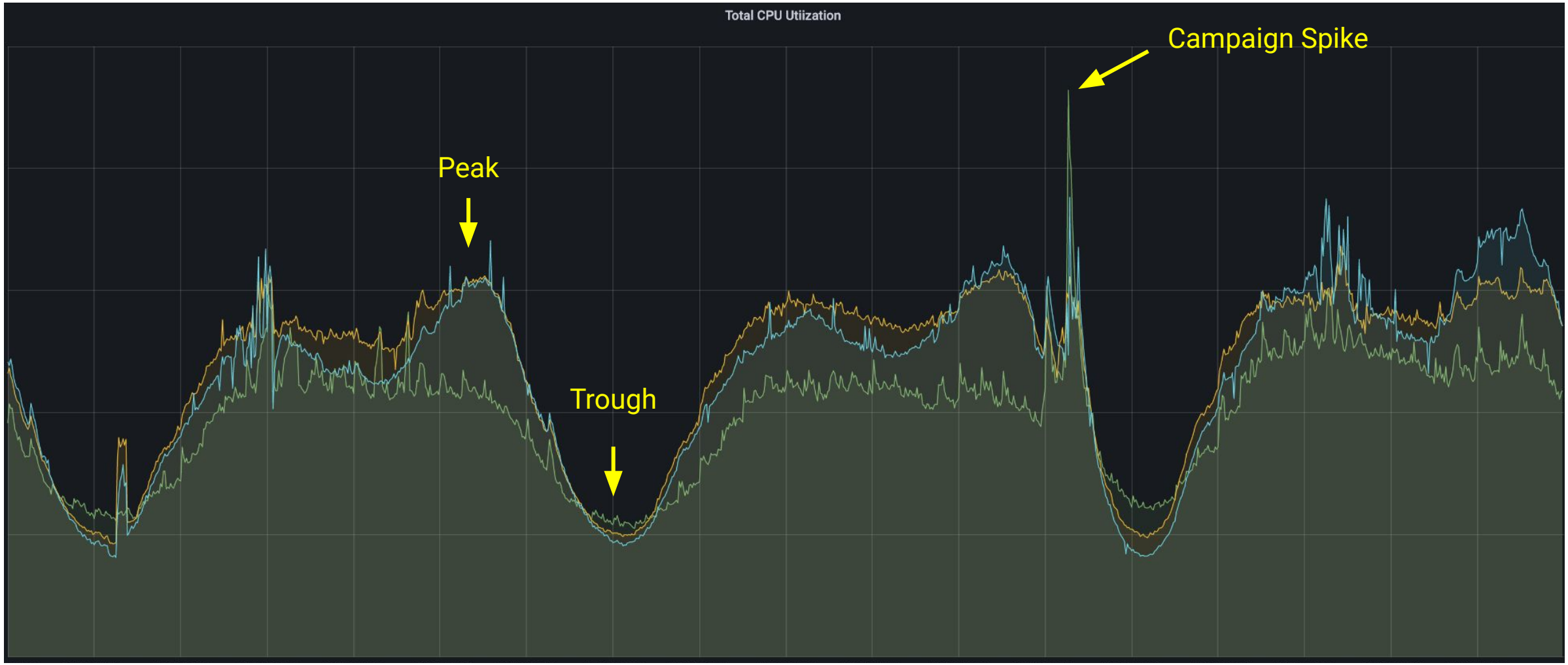
Source: data.ai Q2 2022 Data | 2022 YouGov Global Best Brand Rankings Report

Shopee ❤ Kubernetes

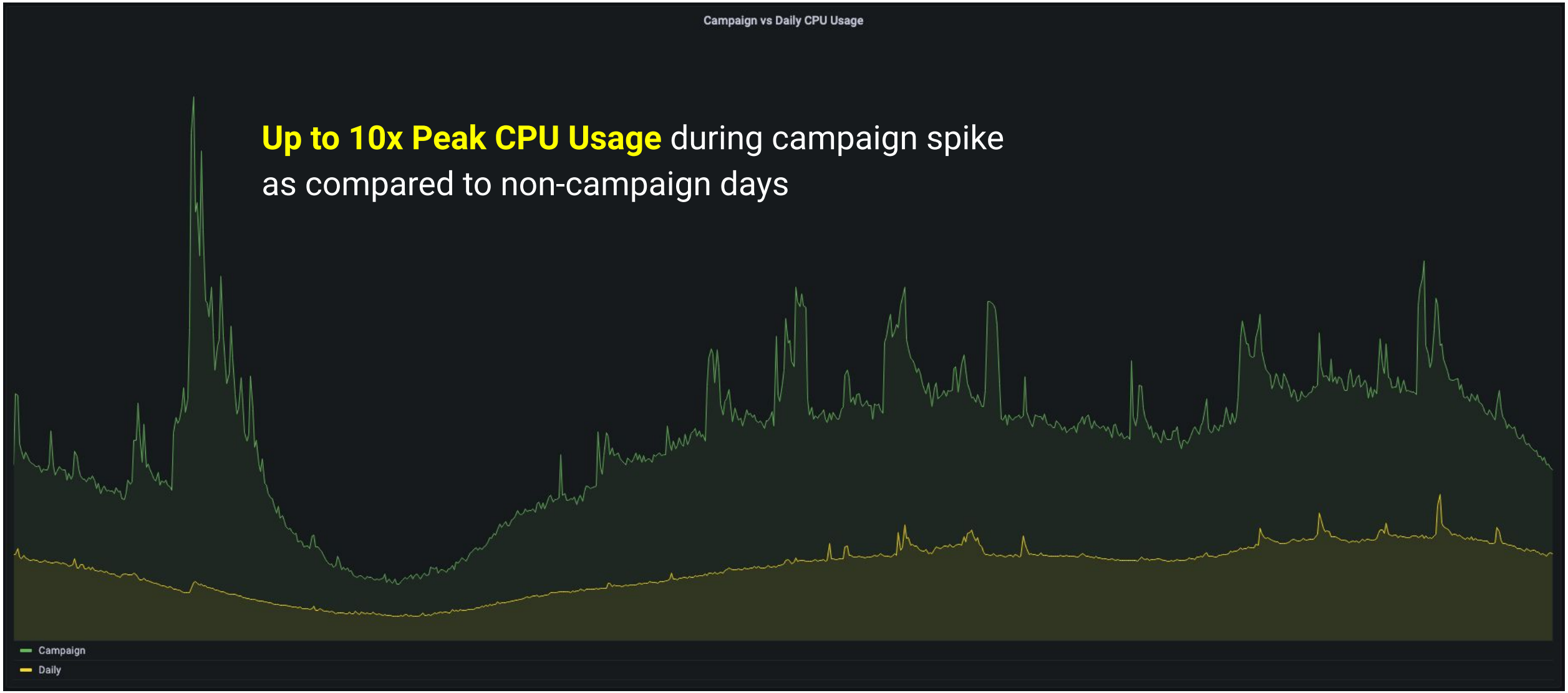
- 100,000+ pods
- 10,000+ nodes
- 100s of clusters
- 10s of data centers across the globe

Capacity Planning Challenges

Capacity Planning Challenges



Capacity Planning Challenges



Capacity Planning Challenges

Traffic is **extremely bursty**, due to timezone localities

Campaigns have **disproportionately larger peaks**

E-commerce users are **especially sensitive to latencies**

Capacity Planning Challenges

Due to business requirements…

Resources are generally **underutilized** most of the time

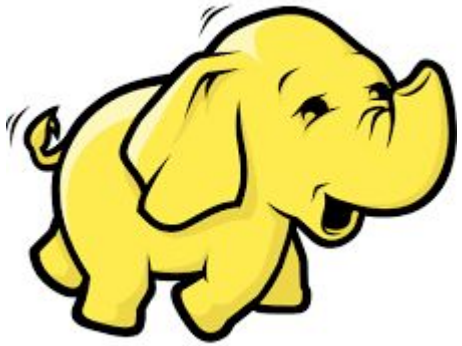
At the same time…

Big data’s resource demands are **increasing rapidly**

Capacity Planning Challenges

Can we run workloads with **weak latency requirements** and **predictable patterns** using these **underutilized Kubernetes resources**?

- Low priority batch jobs
- Big data analytics
- Cron jobs
- ML training
- etc.

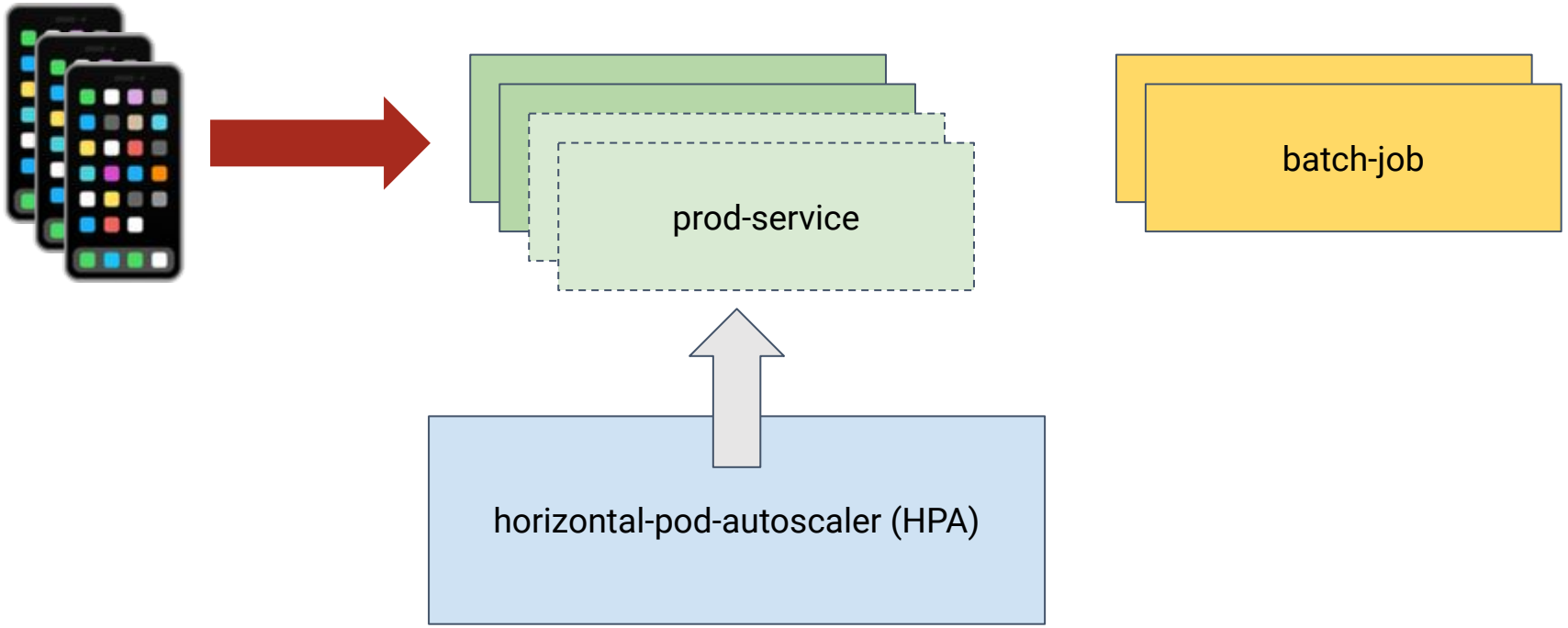


Capacity Planning Challenges

Can we do this?

Using Horizontal Pod Autoscaler (HPA) + PriorityClass:

- Automatically scale up Prod services during peaks

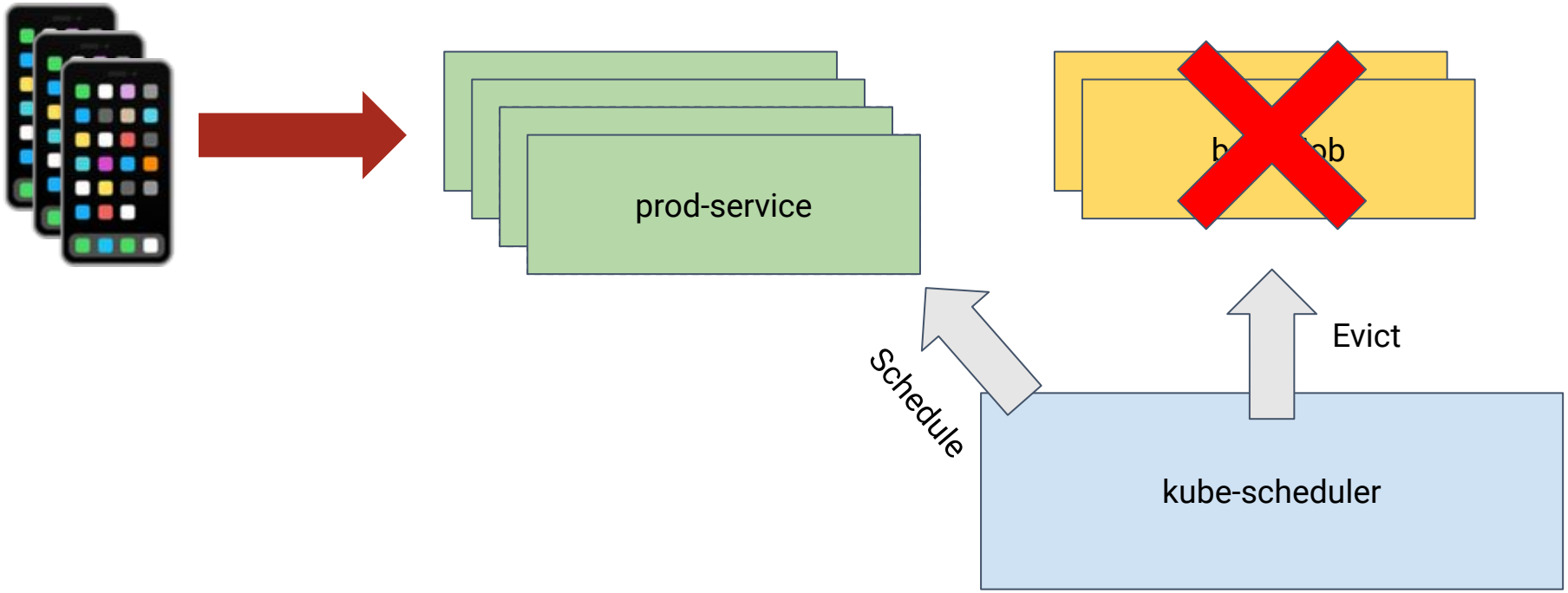


Capacity Planning Challenges

Can we do this?

Using Horizontal Pod Autoscaler (HPA) + PriorityClass:

- Higher PriorityClass will evict Batch workloads automatically

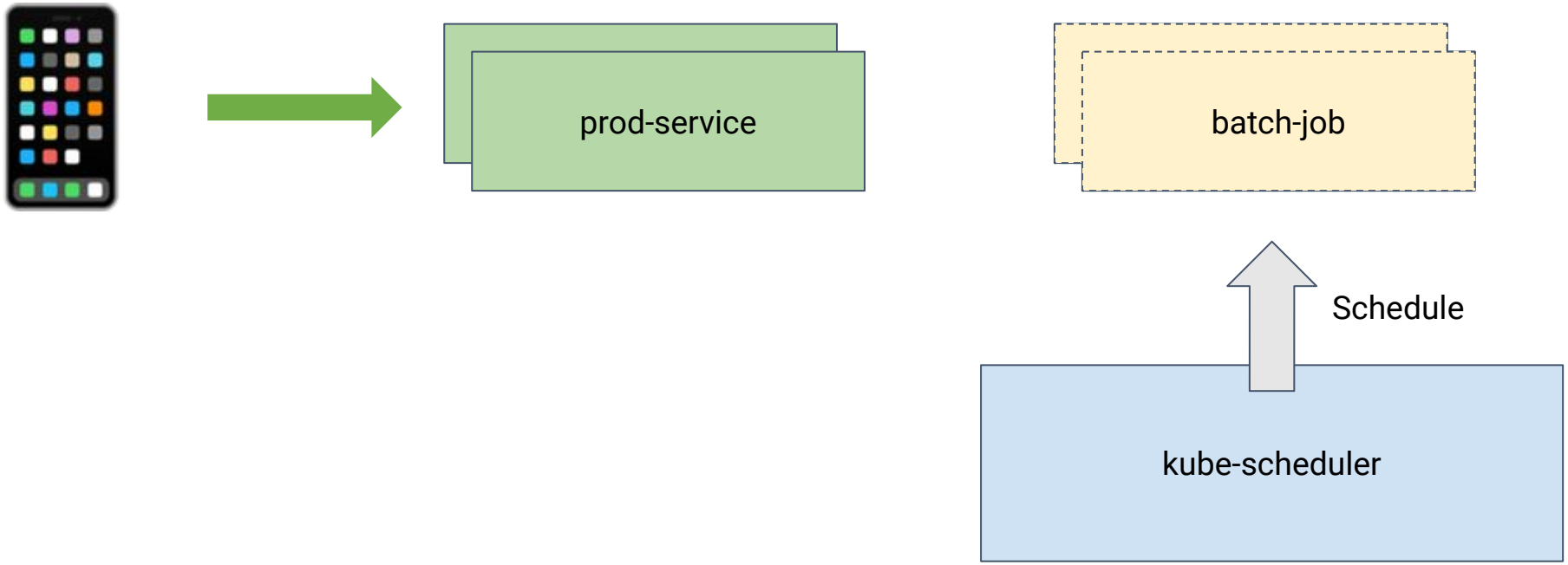


Capacity Planning Challenges

Can we do this?

Using Horizontal Pod Autoscaler (HPA) + PriorityClass:

- Batch workloads will only be scheduled during troughs



Capacity Planning Challenges

Nope…

- Too slow to support **rapid growths** in traffic spikes
- Cannot run non-K8s workloads unless we **reprovision entire nodes**
- Frequent batch pod eviction causes **wasted CPU utilization**

Colocation on Kubernetes

Allow Batch jobs to **reclaim unused resources** from Prod services

while ensuring Prod's **stability and performance** during usage spikes

Recipe for Colocation

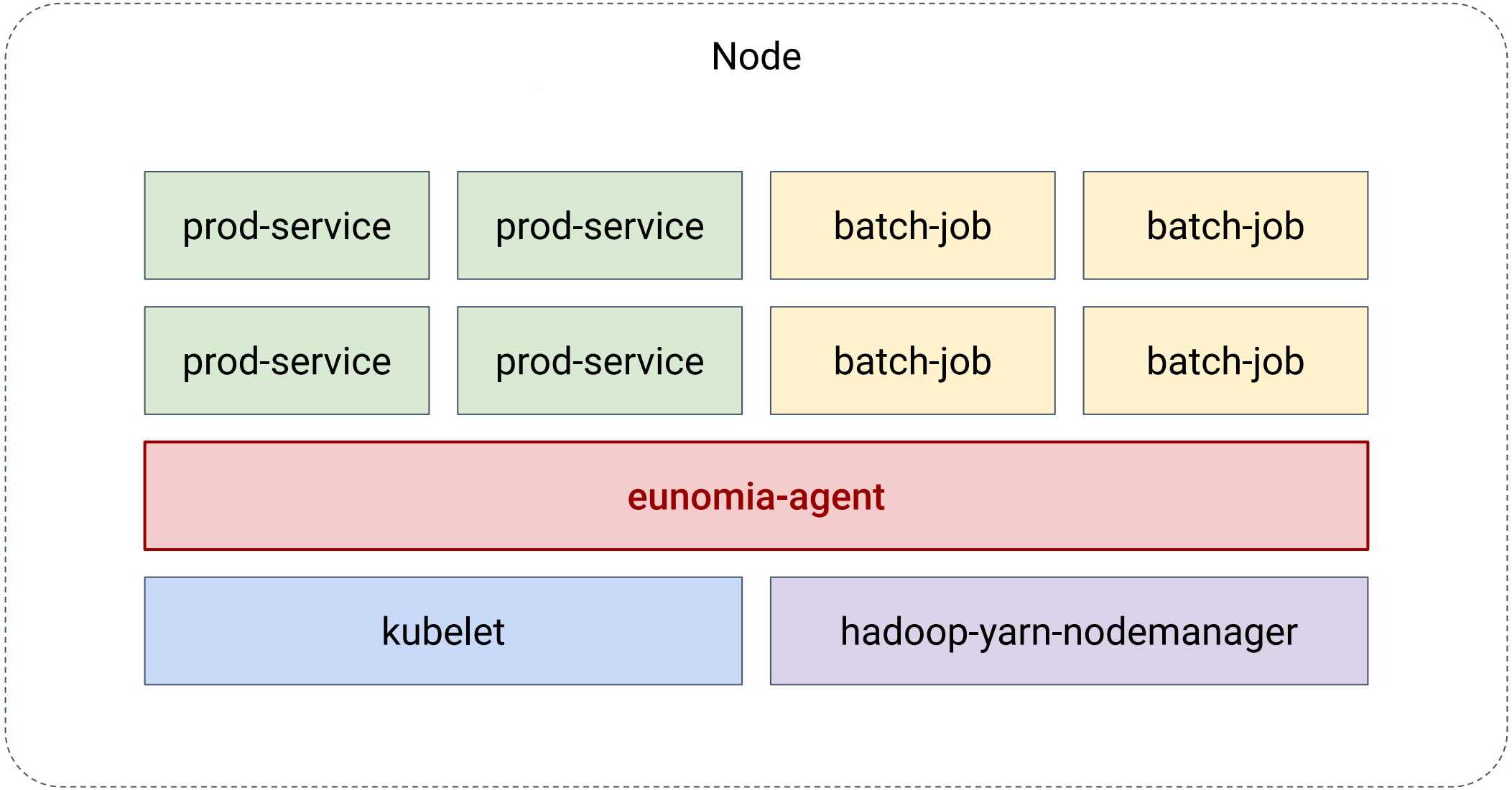
Allocation

- **Monitoring** the node to estimate reclaimable resources
- **Scheduling** of Batch workloads to the node

Isolation

- **Suppressing** of Batch workloads during spikes in Prod usage
- **Evicting** of Batch workloads if necessary

Recipe for Colocation



Inside Eunomia Agent

Unused Resource Allocation

How to identify and schedule reclaimable resources

Reclaiming Unused Resources

CPU Resources

Allocatable: 48

Reclaiming Unused Resources

Allocatable:	48
Scheduled:	45

Reclaiming Unused Resources

Allocatable:	48
Scheduled:	45
Actual Usage:	7.2

Reclaiming Unused Resources

Allocatable:	48
Scheduled:	45
Actual Usage:	7.2
Estimated Usage + Buffer:	12.7

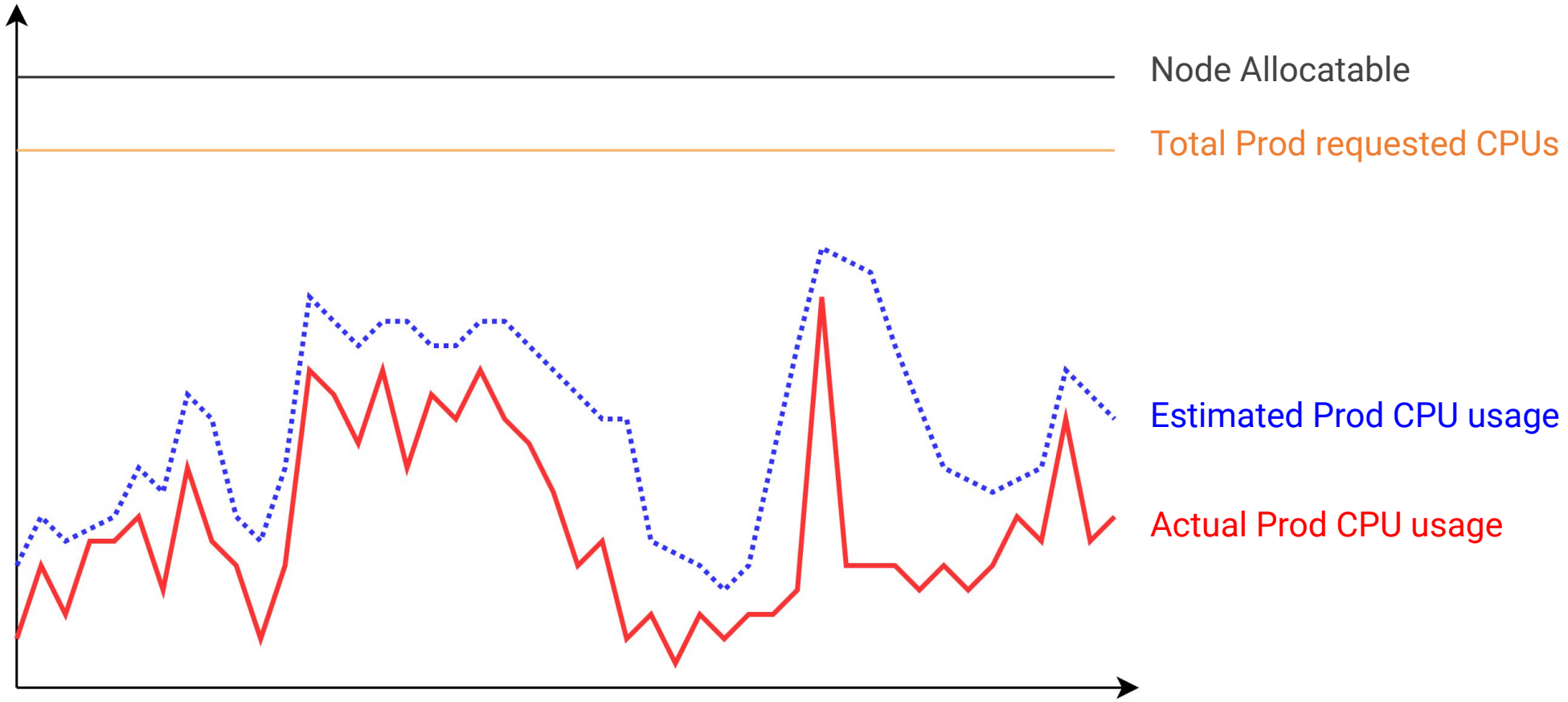
Reclaiming Unused Resources

Allocatable:	48
Scheduled:	45
Actual Usage:	7.2
Estimated Usage + Buffer:	12.7

Potentially Reclaimable: 35.3 CPUs

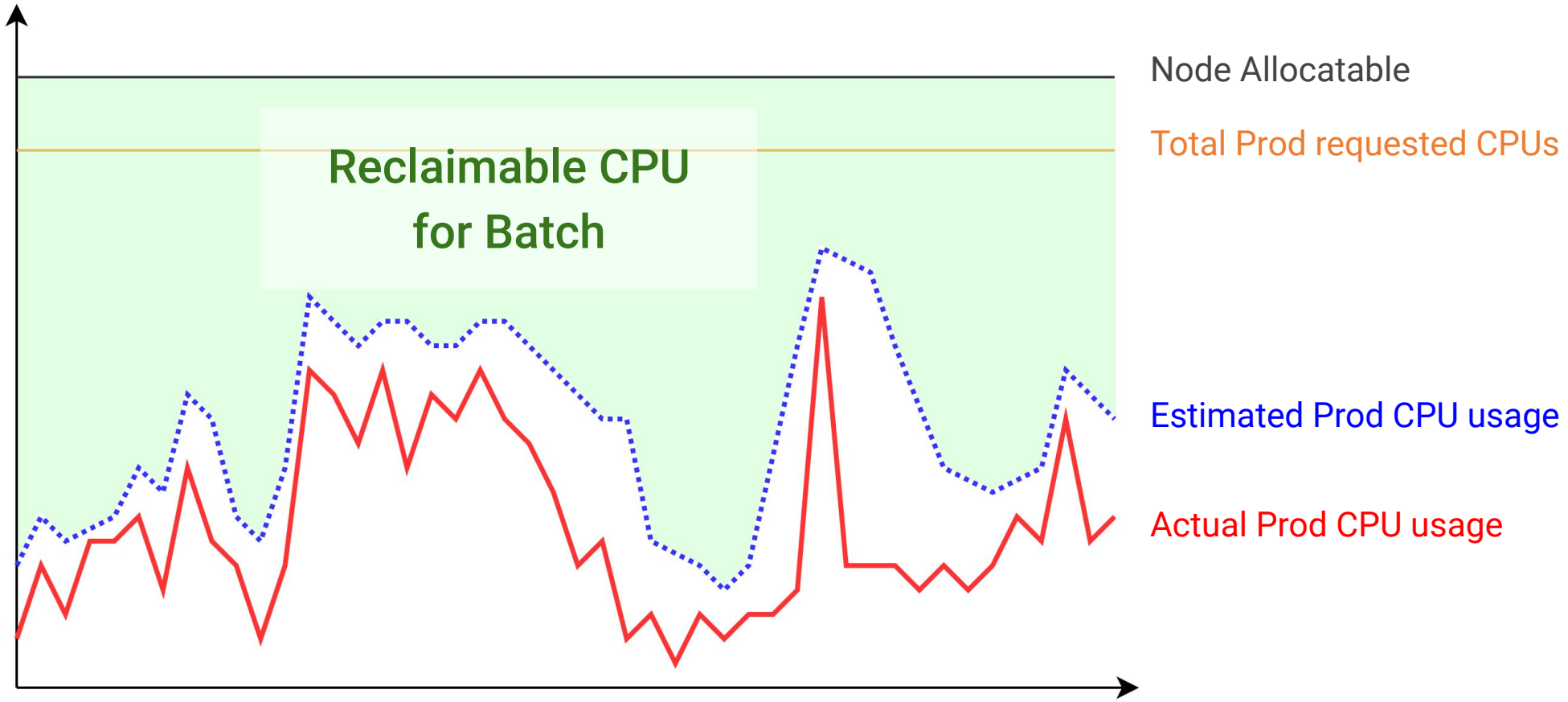
Reclaiming Unused Resources

Dynamic estimation of Prod workloads' CPU usage with exponentially weighted moving average over time



Reclaiming Unused Resources

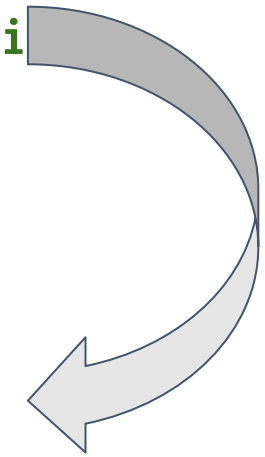
Varying levels of **reclaimed resources** for Batch workloads at different times depending on Prod real-time usage



Reclaiming Unused Resources

Real-time reporting of **reclaimable resources** for Batch workloads using *Extended Resources*

```
allocatable:
  cpu: 48
  memory: 128Gi
  resources.eunomia.io/batch-cpu: 35
  resources.eunomia.io/batch-memory: 108Gi
```



CPU/memory burst
in Prod services

```
allocatable:
  cpu: 48
  memory: 128Gi
  resources.eunomia.io/batch-cpu: 16
  resources.eunomia.io/batch-memory: 64Gi
```

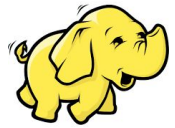
Scheduling Reclaimed Resources

NodeStatus

```
allocatable:
  cpu: 48
  resources.eunomia.io/batch-cpu: 16
```


Can schedule Pods:

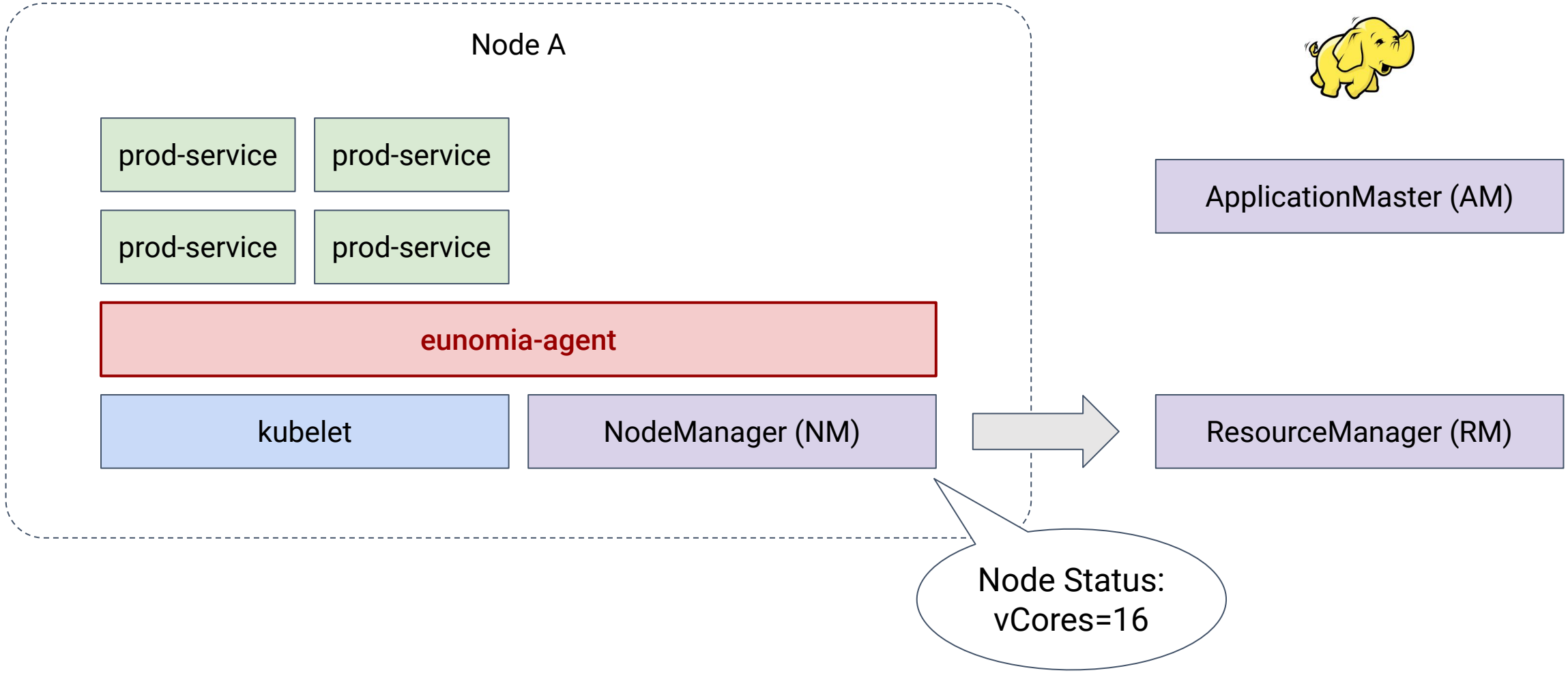
```
spec:
  containers:
  - name: my-batch-job
    resources:
      requests:
        resources.eunomia.io/batch-cpu: 16
```

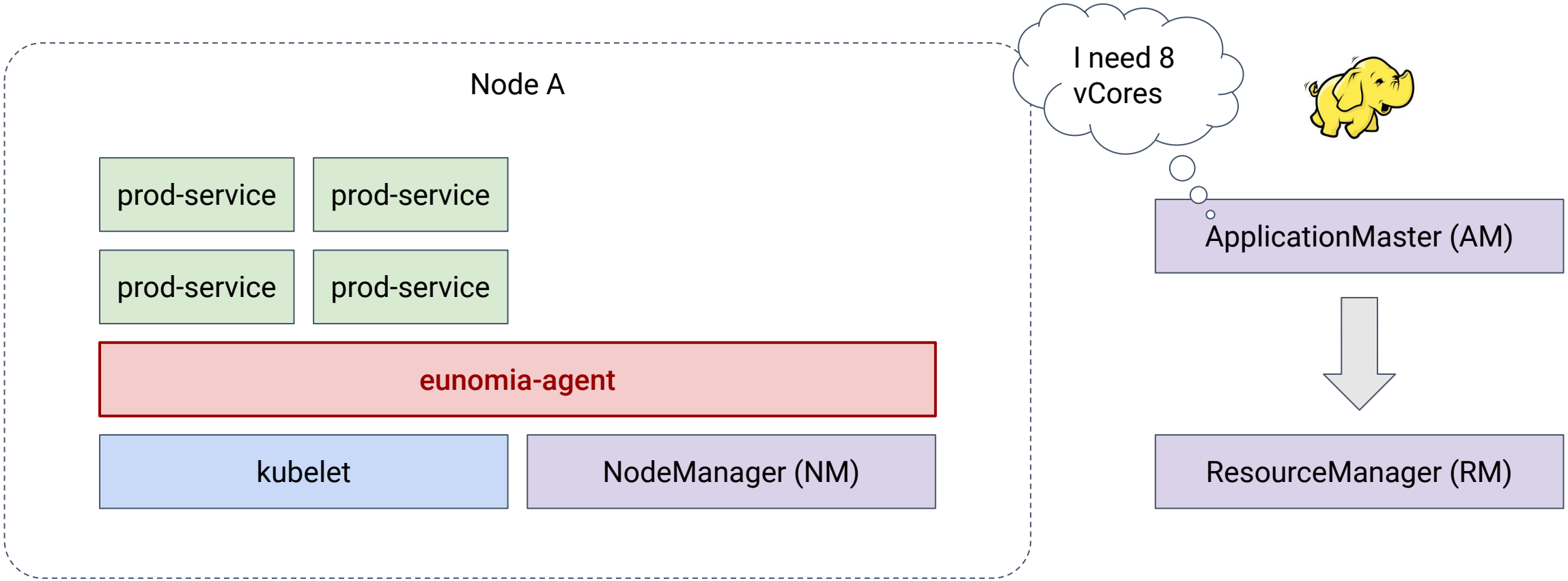
Hadoop YARN:

Use a custom API to notify YARN NodeManager to set `vCores=16`.

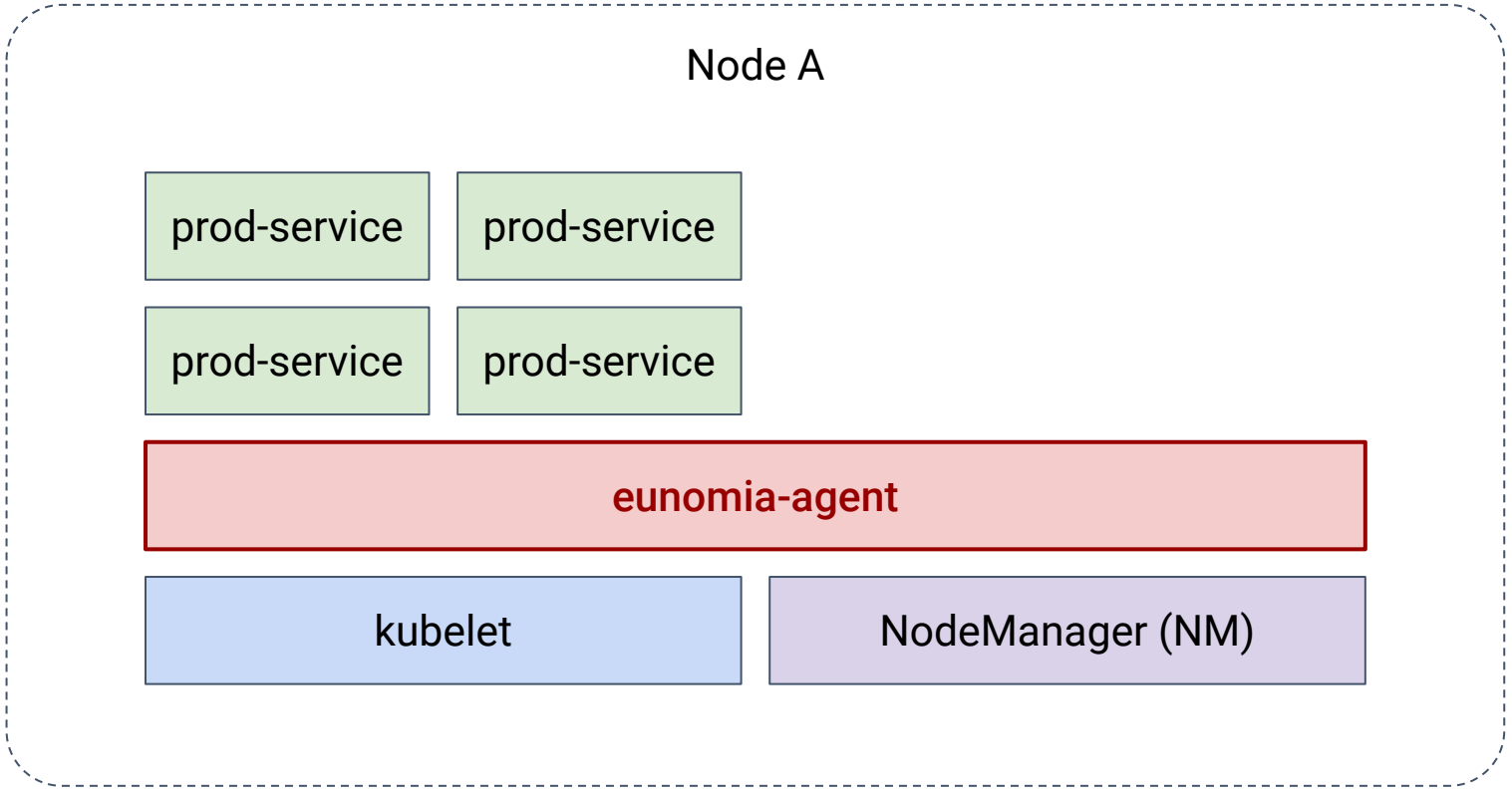
Scheduling Reclaimed Resources

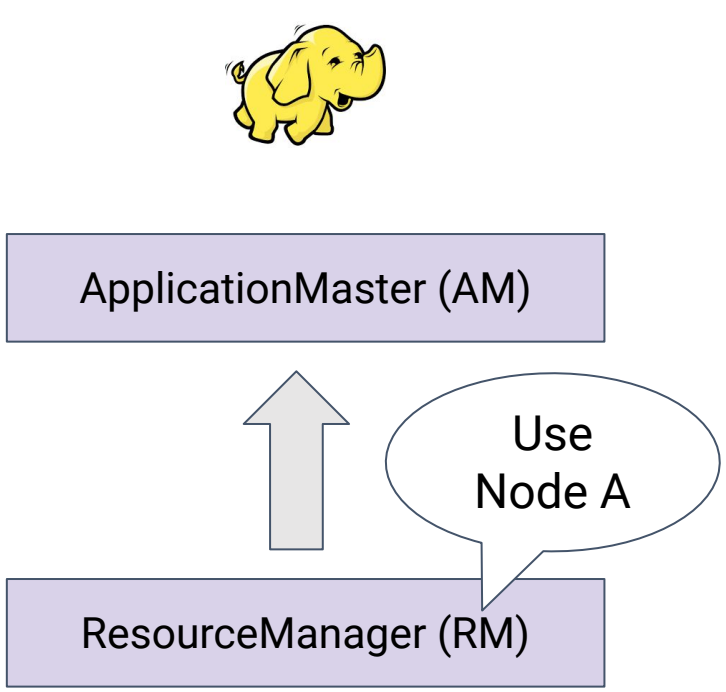


Scheduling Reclaimed Resources

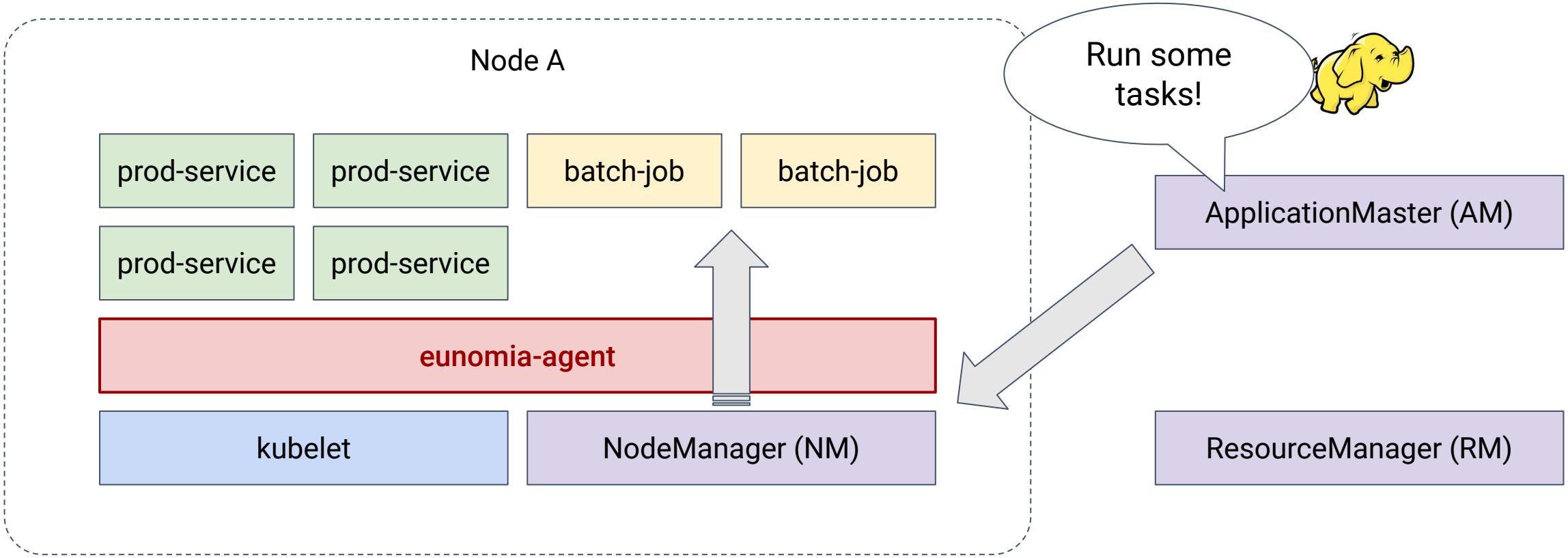


Scheduling Reclaimed Resources



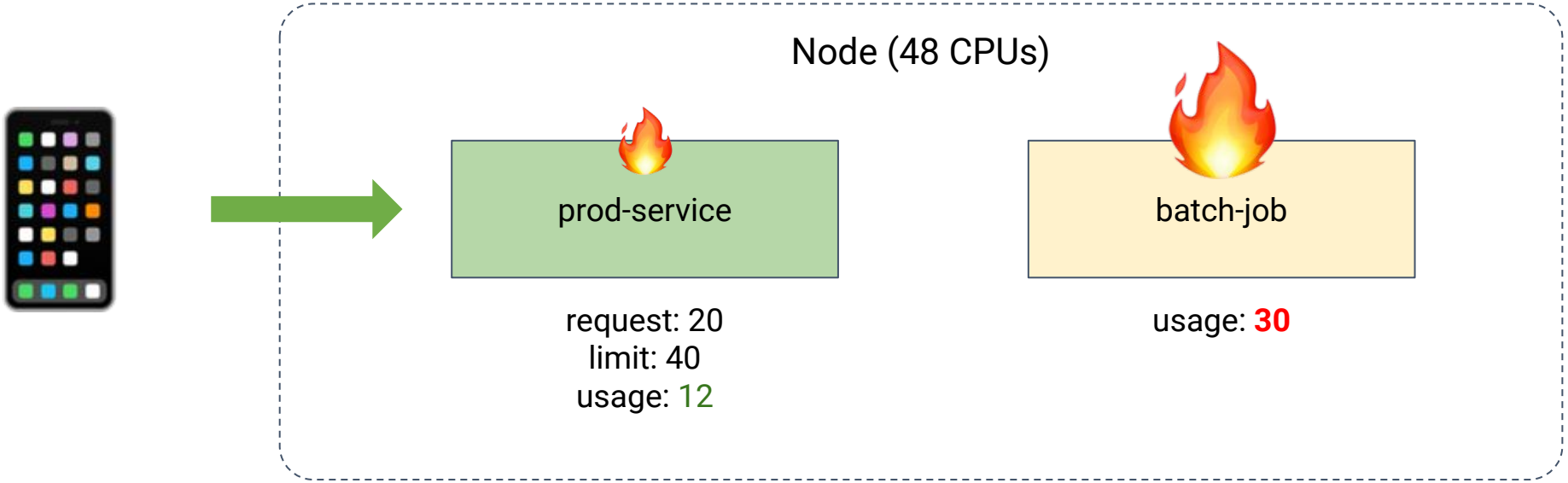


Scheduling Reclaimed Resources



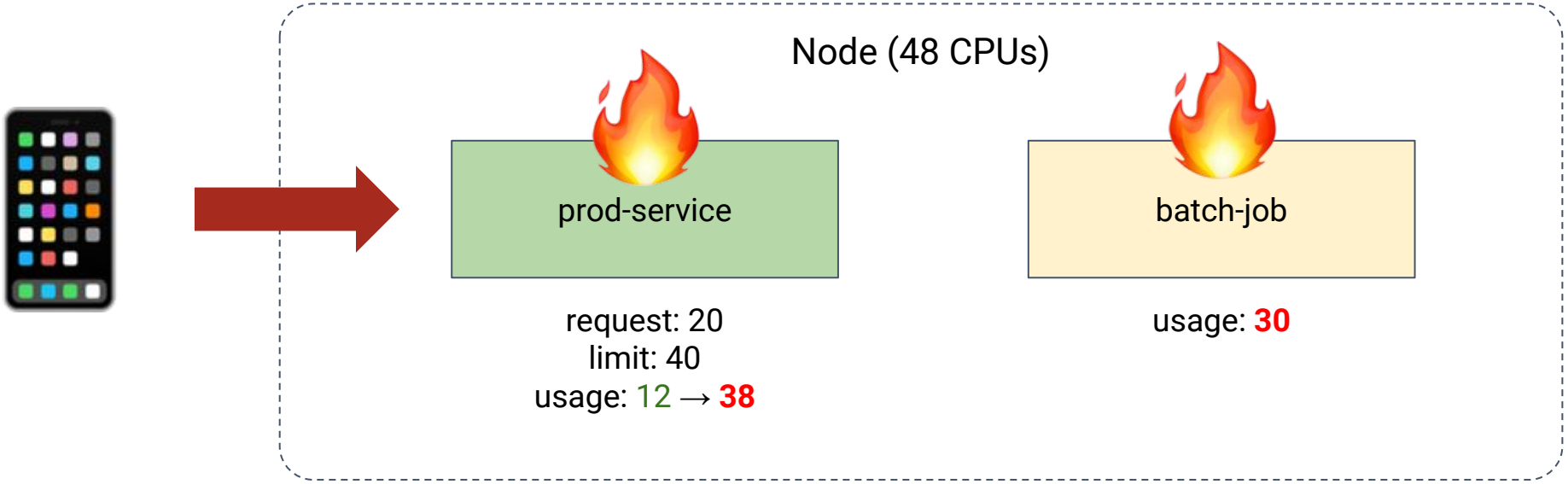
Wait a minute…

When Prod services burst in CPU, won't the Batch job containers compete with Prod service containers for resources?



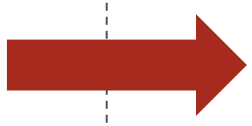
Wait a minute…

When Prod services burst in CPU, won't the Batch job containers compete with Prod service containers for resources?



In other words…

How can we ensure that Prod services will have enough CPU to support rapid traffic bursts?



Isolating Noisy Neighbors

How to minimize effects caused by Batch workloads

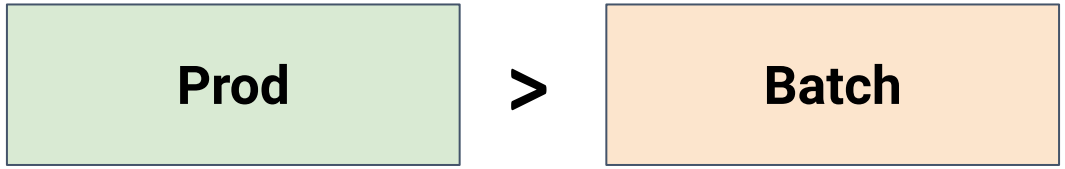
Isolating Noisy Neighbors

WorkloadQoS: Quality of Service (QoS) classes based on latency requirements

WorkloadQoS	Description	Example Use Cases
ProdGuaranteed	• Reserved CPUSet • CPU and memory NUMA alignment • Unconditionally suppress Mid ~ Batch	Highly critical services, control plane components
ProdBurstable	• Share CPUs with other ProdBurstable • Unconditionally suppress Mid ~ Batch	Stateless web servers
ProdRelaxed	• Suppress Mid ~ Batch	DaemonSet services
Mid	• Relatively stable resources • Suppress Batch	Internal web services, non-business critical services
Batch	• Dynamic, unstable resources	Low priority batch jobs, cron jobs, big data jobs, video/image transcoding

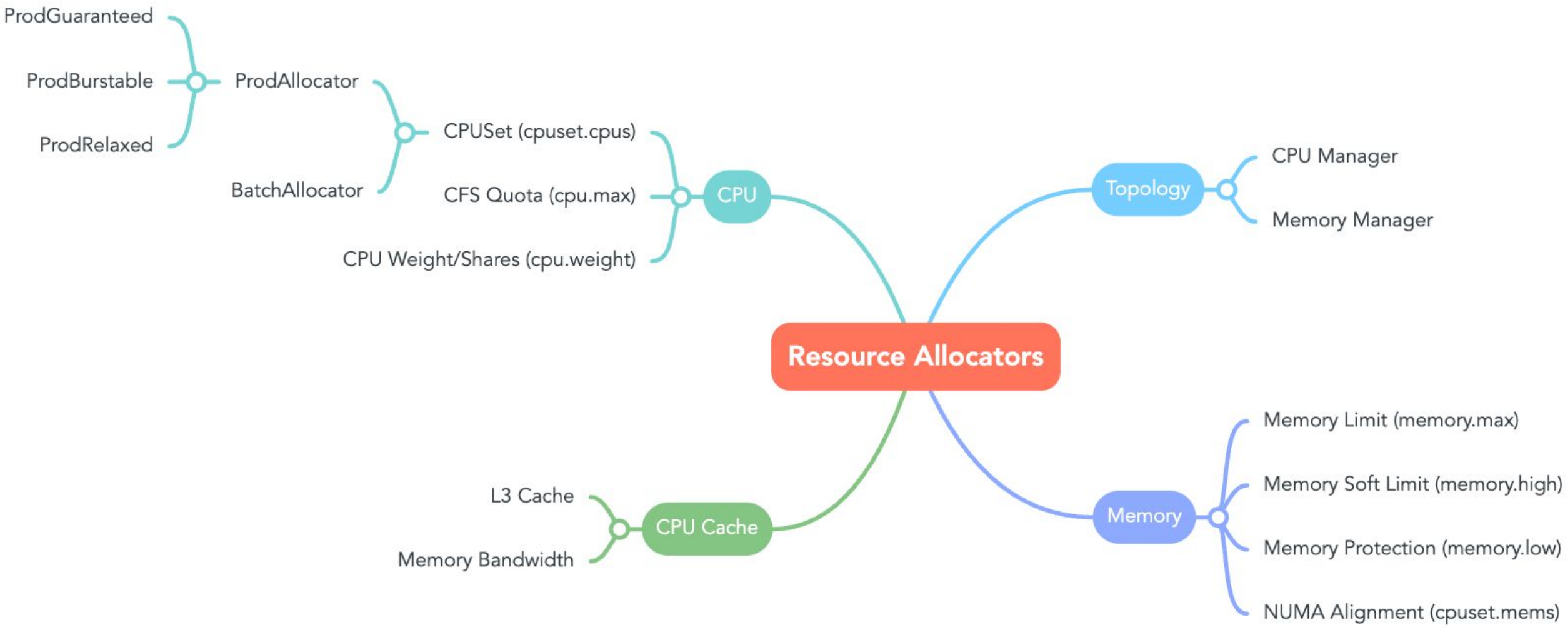
Isolating Noisy Neighbors

WorkloadQoS: Quality of Service (QoS) classes based on latency requirements



Prod is able to **suppress** Batch in real-time
using Linux kernel features.

Isolating Noisy Neighbors



Suppression and Eviction

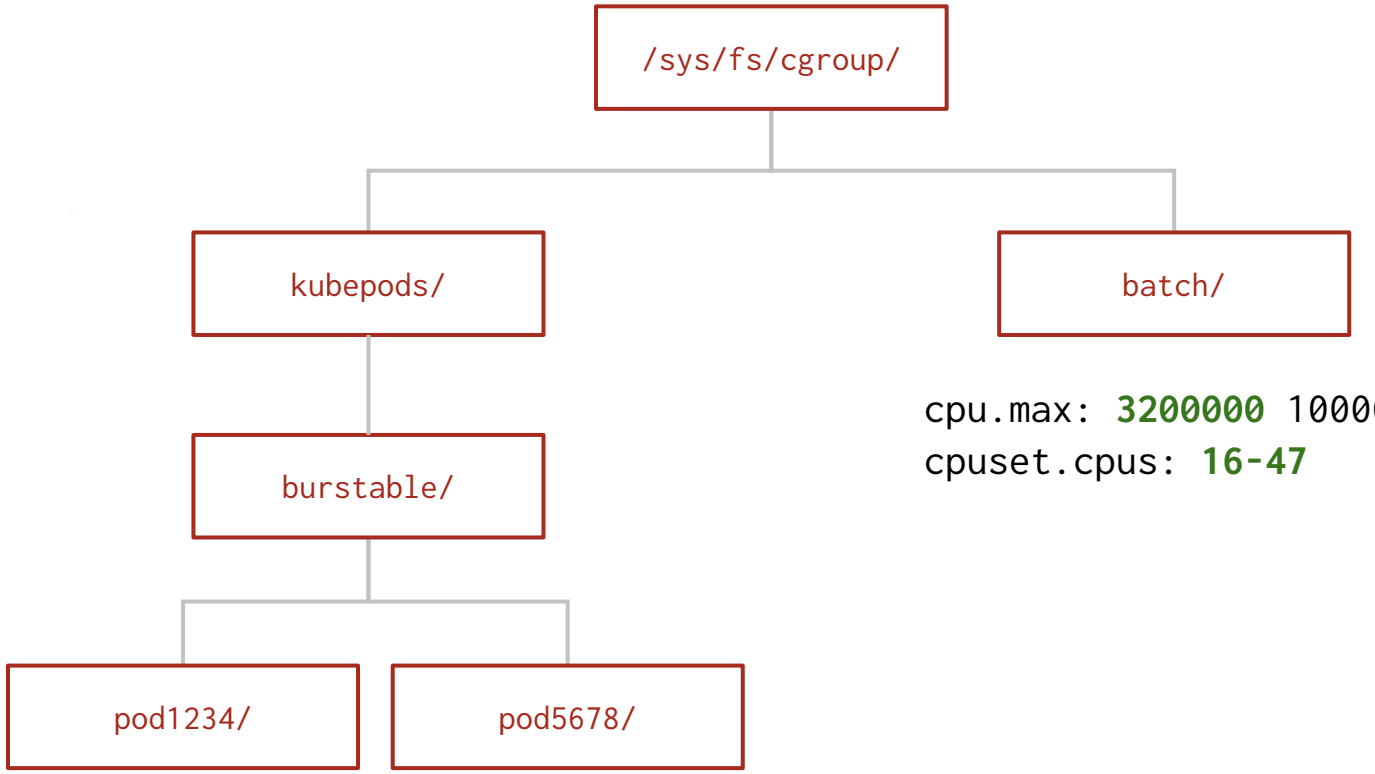
Total CPU Cores: **48**

Prod CPUs Allocated: **40**

Current CPU Usage: **12**

CPU Estimate: **16**

Reclaimable CPUs: **32**



Suppression and Eviction

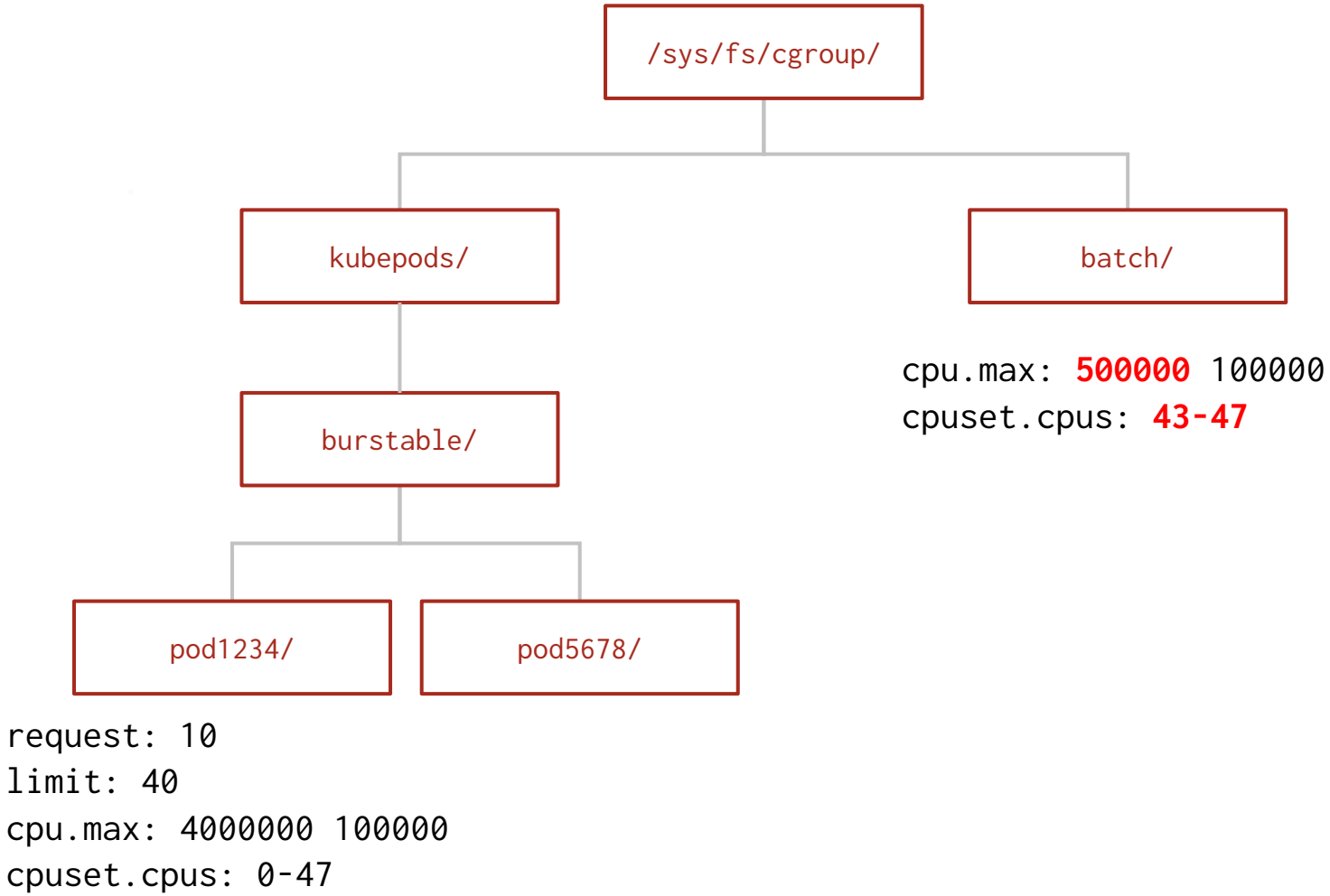
Total CPU Cores: **48**

Prod CPUs Allocated: **40**

Current CPU Usage: **38**

CPU Estimate: **43**

Reclaimable CPUs: **5**



Batch workloads can be **suppressed immediately**
when Prod has temporary CPU spikes.

Suppression and Eviction

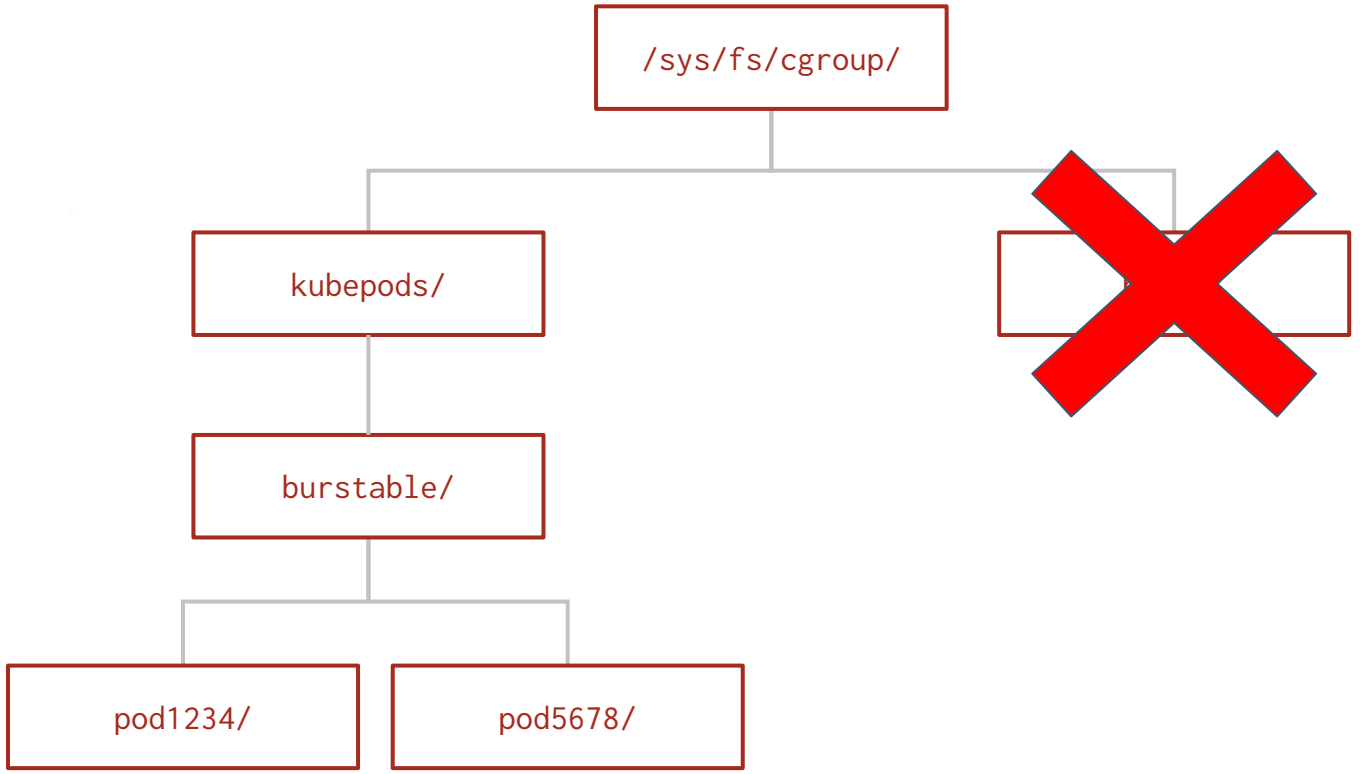
Total CPU Cores: **48**

Prod CPUs Allocated: **40**

Current CPU Usage: **38**

CPU Estimate: **43**

Reclaimable CPUs: **5**

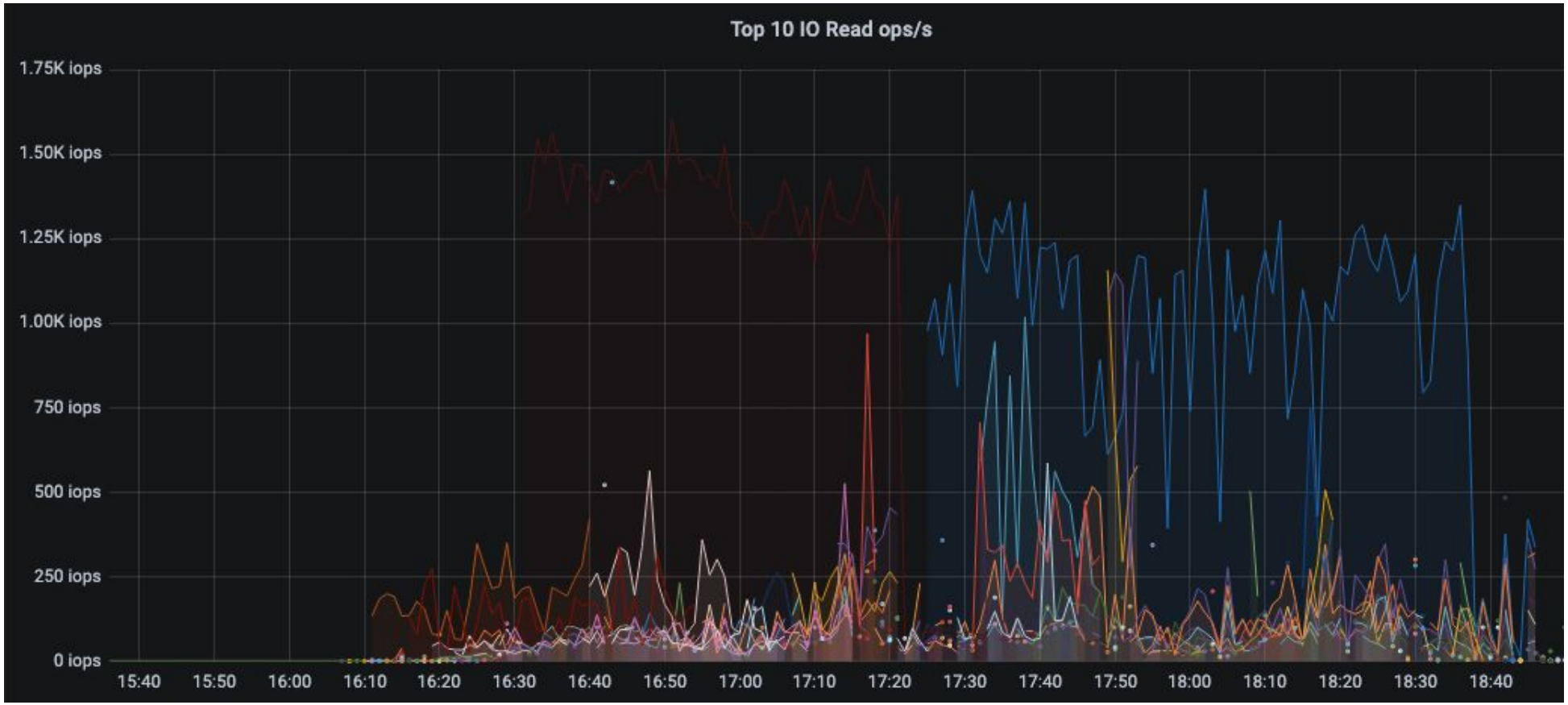



```
request: 10
limit: 40
cpu.max: 4000000 100000
cpuset.cpus: 0-47
```

If the CPU usage remains high for a period of time,
Batch workloads will eventually be **evicted from the node**.

Case Study: cgroup v2 Writeback

Background: We want to control I/O limits for Batch jobs, and ensure that if Batch jobs read/write lots of files, it will not affect Prod services.



Case Study: cgroup v2 Writeback

Background: We want to control I/O limits for Batch jobs, and ensure that if Batch jobs read/write lots of files, it will not affect Prod services.

- cgroup v1: `blkio.throttle.read_bps_device / write_bps_device`
- cgroup v2: `io.max`

Is it really that simple?

Case Study: cgroup v2 Writeback

Issue: Configuring `io.max` will throttle both direct and buffered I/O, which will limit the **writeback rate**.

⇒ Configured `io.max` for Batch containers

⇒ Batch containers write files quickly, but writeback is throttled to `io.max`

⇒ Dirty pages pile up → High memory pressure on the whole system

⇒ Memory is reclaimed from Prod containers instead

⇒ Prod services start stalling

Case Study: cgroup v2 Writeback

Batch container

```
[ root@Thu Apr 06 12:39:58 ] $ # First we add ourselves to the cgroup
[ root@Thu Apr 06 12:40:02 ] $ echo $$ > cgroup.procs
[ root@Thu Apr 06 12:40:05 ] $ # There is currently very little Free memory (mostly in page cache)
[ root@Thu Apr 06 12:40:07 ] $ free -mh
               total        used        free      shared  buff/cache   available
Mem:           124Gi       7.1Gi       1.8Gi       5.0Mi       115Gi       113Gi
Swap:             0B          0B          0B
[ root@Thu Apr 06 12:40:09 ] $ # Limit the write bps to 100 MB/s
[ root@Thu Apr 06 12:40:13 ] $ echo "8:0 wbps=104857600" > io.max
[ root@Thu Apr 06 12:40:17 ] $ # Now write 100GB to disk in the background
[ root@Thu Apr 06 12:40:29 ] $ dd if=/dev/zero of=/data/irvin/test1 bs=1G count=100 &
[1] 765909
```

Prod container

```
[ root@Thu Apr 06 12:50:44 ] $ time dd if=/dev/zero of=/data/irvin/prod-1K bs=1K count=1
1+0 records in
1+0 records out
1024 bytes (1.0 kB, 1.0 KiB) copied, 46.6192 s, 0.0 kB/s

real    0m46.621s
user    0m0.002s
sys     0m0.000s
```

Writing only 1KB can stall for as long as 46s.

Case Study: cgroup v2 Writeback

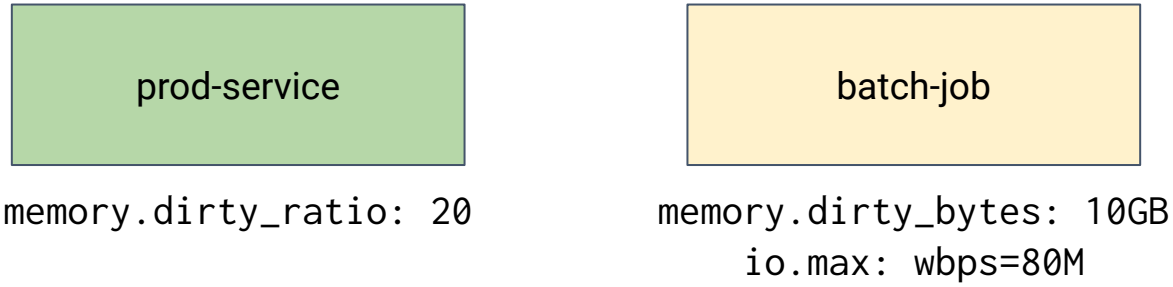
Solution: We need to control the **maximum dirty page size** on a cgroup-level.

⇒ Make **Batch containers stall** once dirty limit is reached

⇒ Use `io.max` in order to control how fast dirty pages can be **flushed**

Implement Linux patches for

per-cgroup **vm.dirty_ratio** / **vm.dirty_bytes**



How can we better ensure the performance of the business line?

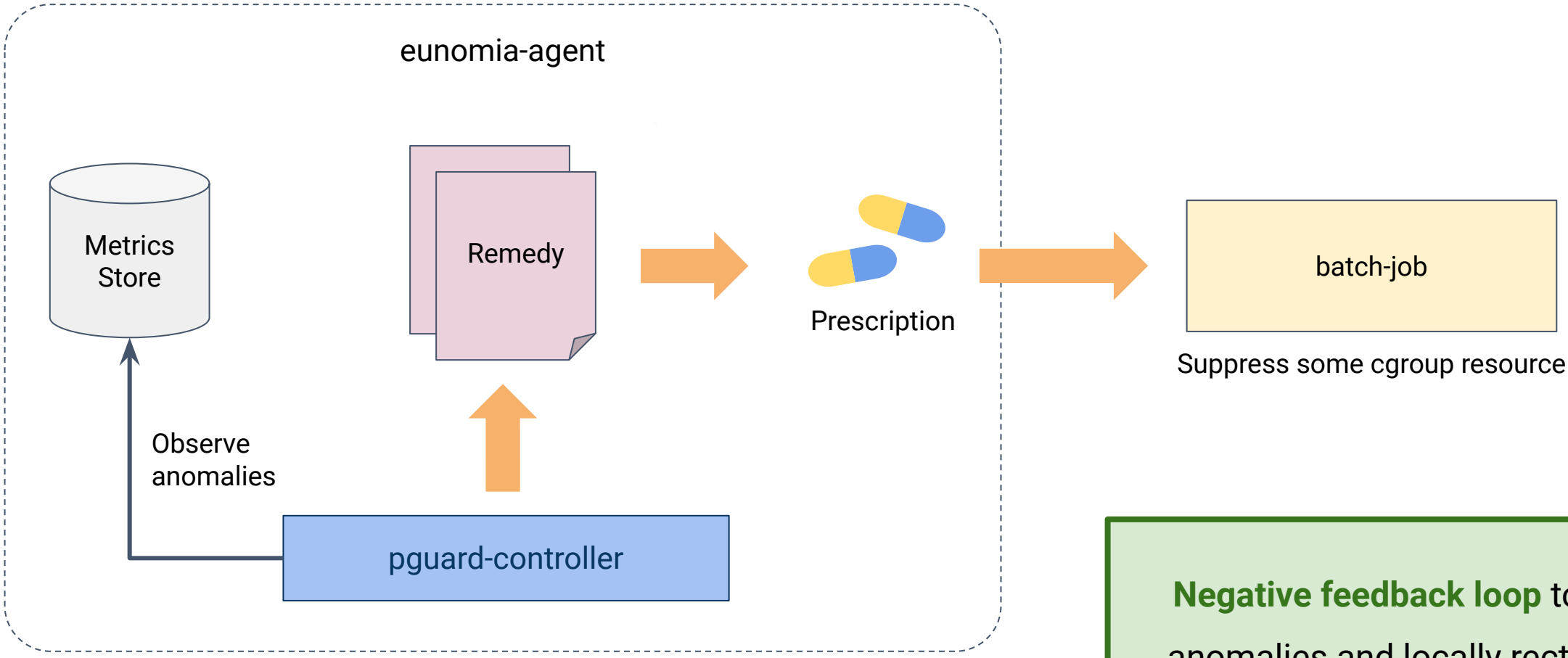
What are the SLOs that we care about?

If these SLOs are violated, how can we rectify it?

Can we allocate containers more efficiently to maximize SLOs?

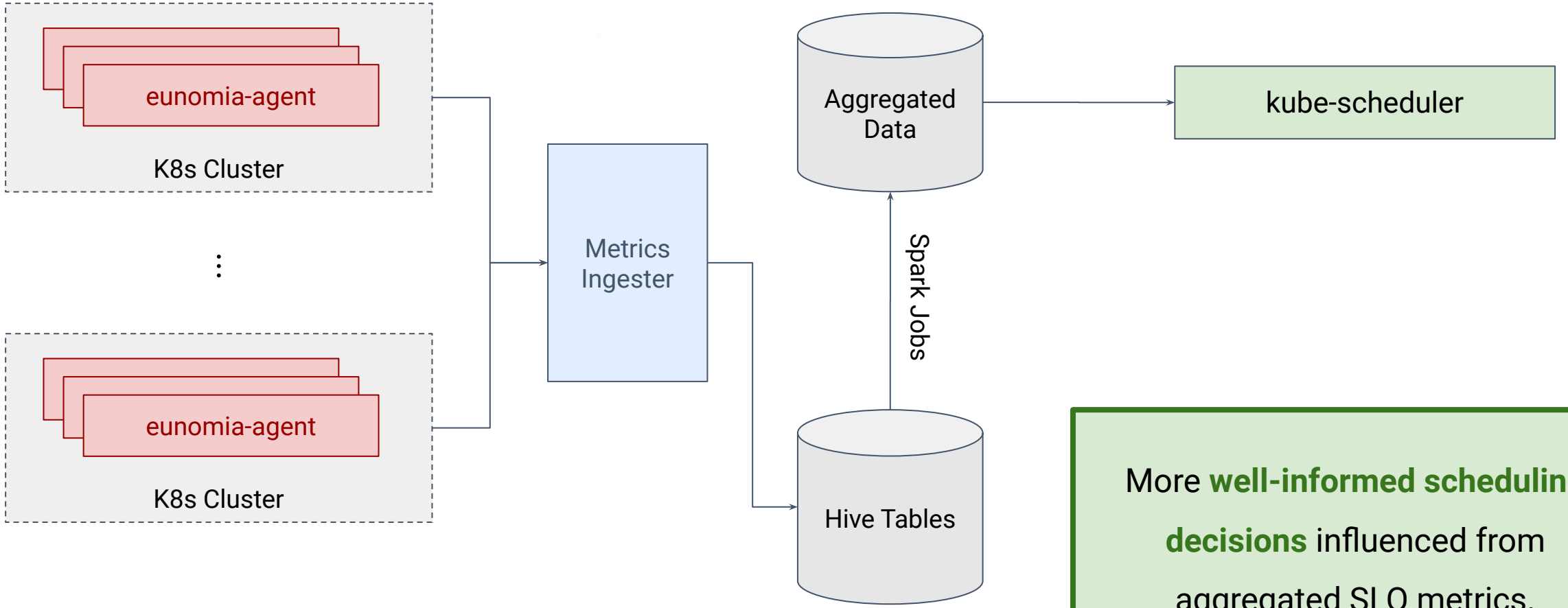
Observability of SLOs

Self-Healing with Performance Guards



Negative feedback loop to detect anomalies and locally rectify node SLO violations automatically.

SLO-Based Scheduling

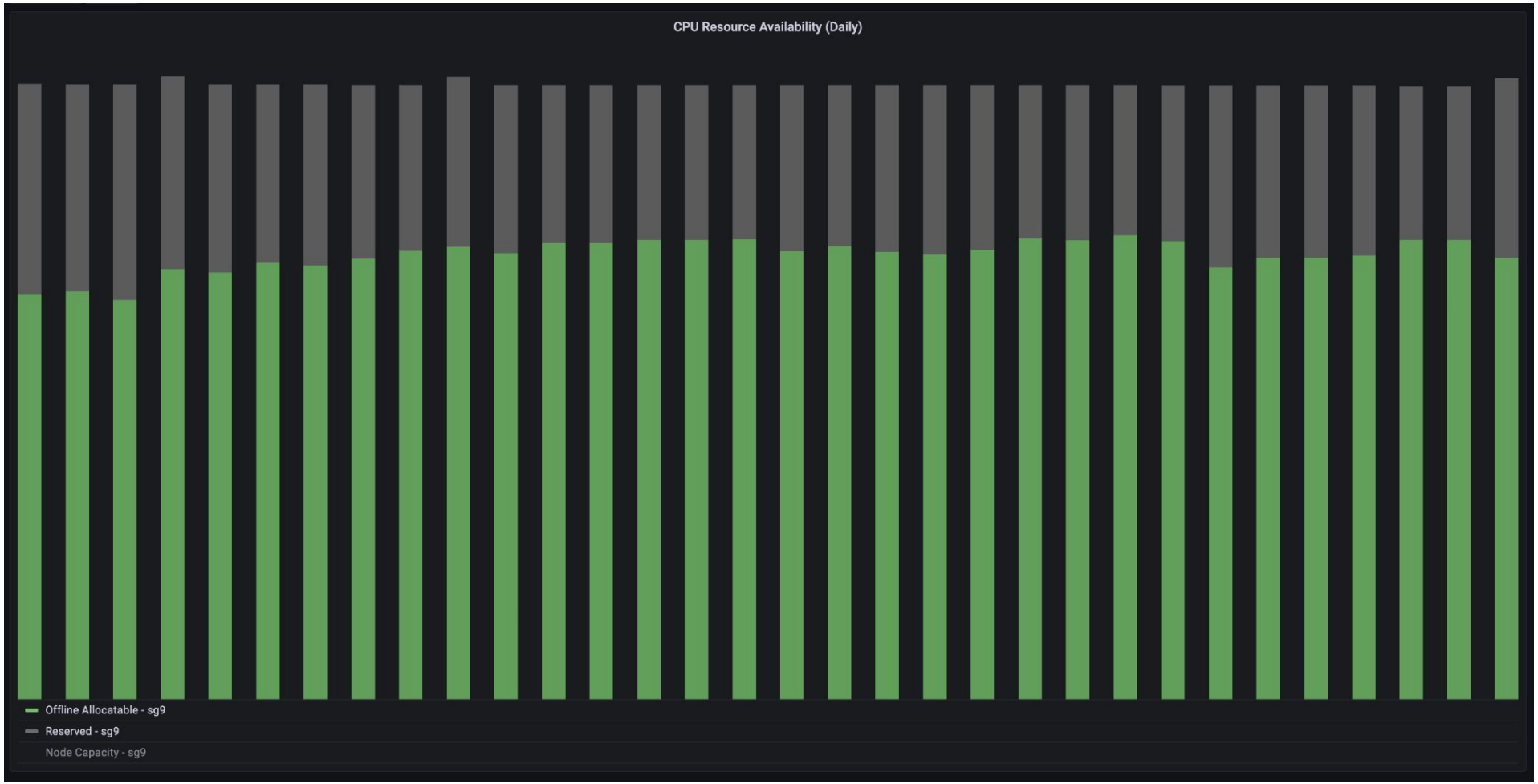


More **well-informed scheduling decisions** influenced from aggregated SLO metrics.

Results

Resources Reclaimed

Reclaimed **more than 70%** of all CPU resources per day for Batch jobs

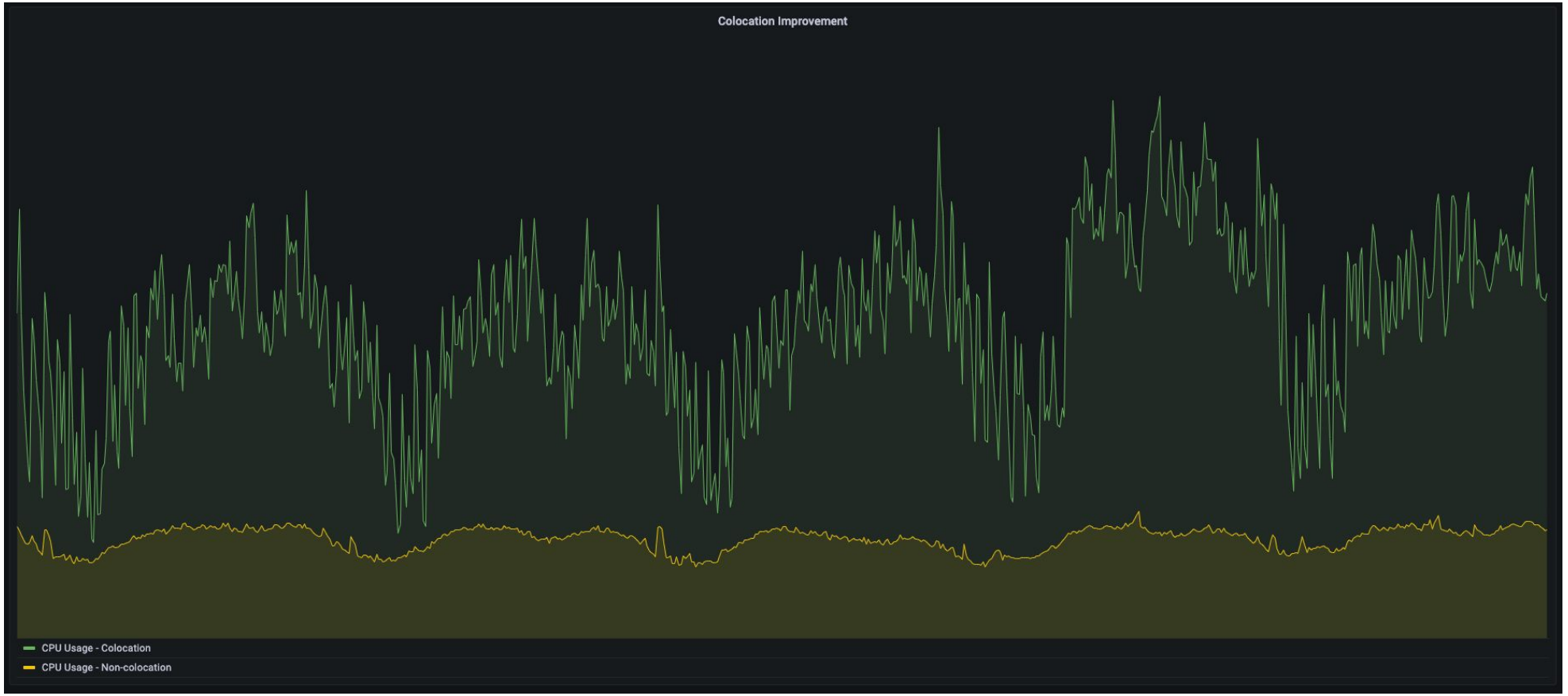


Daily share of Batch allocatable CPU cores (green) out of all allocatable CPU cores.

CPU Utilization Improvement

Up to **4.3x improvement** in peak CPU utilization

Up to **3.2x improvement** in average CPU utilization



CPU usage for colocation nodes vs non-colocation nodes for a single cluster.

Performance and Stability

Less than 5% impact to tail latencies for Prod services

P99 latency for a critical RPC service, on colocation nodes vs non-colocation nodes.

Performance and Stability

Less than 1% of Hadoop YARN jobs failed per day

on colocated K8s clusters

Job failure reasons may include application errors, platform issues as well as eviction as a result of high node pressure.

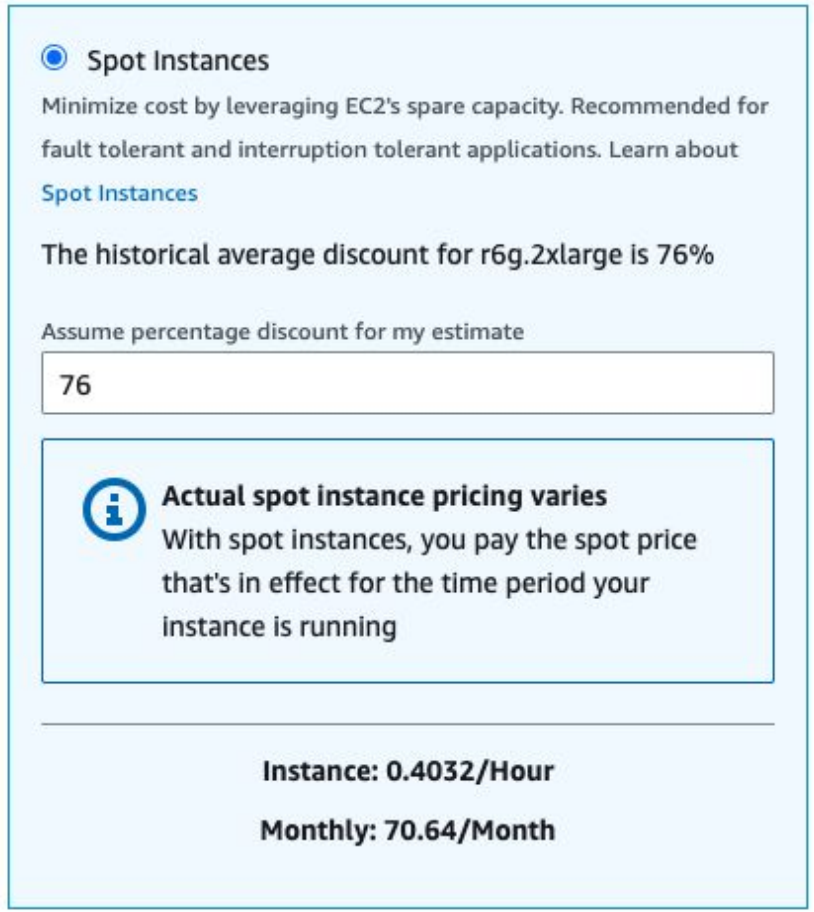
Cost Savings Estimation

For every 10% of CPUs reclaimed, assuming 80 CPU machines:

- 8 CPU cores reclaimed per machine
- EC2 Spot Instance: USD $848/year for r6g.2xlarge (8 vCPUs, 64 GiB)
- Assuming a 5,000 node K8s cluster…

Savings of up to USD $4.2 million/year

for every 8 CPUs reclaimed per node



Takeaways

What we learnt from this journey

Takeaways

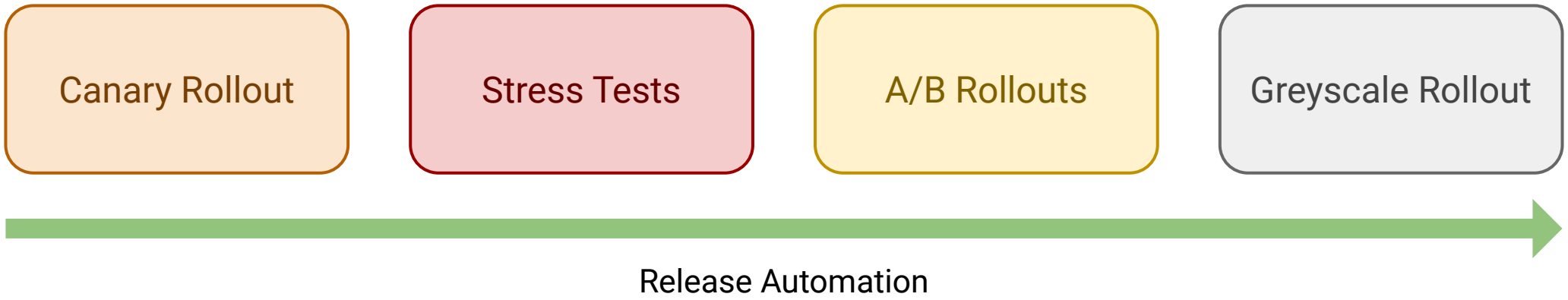
Scalability

- Bottom-up, decentralized agent design worked really well
- Implement in-memory `ListerWatcher` to support CustomResource reconciliation without depending on and overloading kube-apiserver
- Scales to thousands of nodes per cluster ⇒ **>50,000 CR writes/sec in the cluster**

Takeaways

Risk Management

- Modifying cgroup and kernel parameters are risky
- Risks are vastly magnified when deployed to thousands of machines
- Implement risk classification tables with strict release policies



Takeaways

Monitoring and Observability

- Capture as many metrics as possible – all the way from the kernel to the business
- Business SLO monitoring is equally important as node-level monitoring – you will **definitely** know when the business is impacted!
- Comparing business SLOs for the same service on different machines allows us to measure the “**colocation cost**” via A/B testing

KubeCon CloudNativeCon Europe 2023

KubeCon CloudNativeCon North America 2021

Please scan the QR Code above
to leave feedback on this session

Thank you!

For more questions, reach us on CNCF Slack!