
KubeCon
CloudNativeCon
North America 2022
BUILDING FOR THE ROAD AHEAD
DETROIT 2022
TiKV
TiKV
HELM

BUILDING FOR THE ROAD AHEAD

DETROIT 2022

Don't be greedy

Rightsize your Kubernetes cluster with Prometheus

Jesús Samitier
David Lorite

KubeCon | CloudNativeCon
North America 2022
BUILDING FOR THE ROAD AHEAD
DETROIT 2022

David Lorite

Jesús Samitier

Integration Engineers
PromCat.io maintainers
at Sysdig

October 24-28, 2021

PROMCAT

A resource catalog for enterprise-class Prometheus monitoring

A PROJECT BY sysdig

https://promcat.io

FILTER

SELECT YOUR CATEGORY

- [ ] Available
- [ ] AWS
- [ ] CI-CD
- [ ] Coming soon
- [ ] Containers
- [ ] Database
- [ ] Host
- [ ] Hpa
- [ ] Kubernetes
- [ ] Load-balancer
- [ ] Logging
- [ ] Message-broker
- [ ] Network
- [ ] Observability
- [ ] OpenShift
- [ ] PHP

APACHE SOLR
Solr is an open-source enterprise-search platform.
COMING SOON

APACHE
The Apache HTTP Server Project

AWS ALB
AWS Application Load Balancer

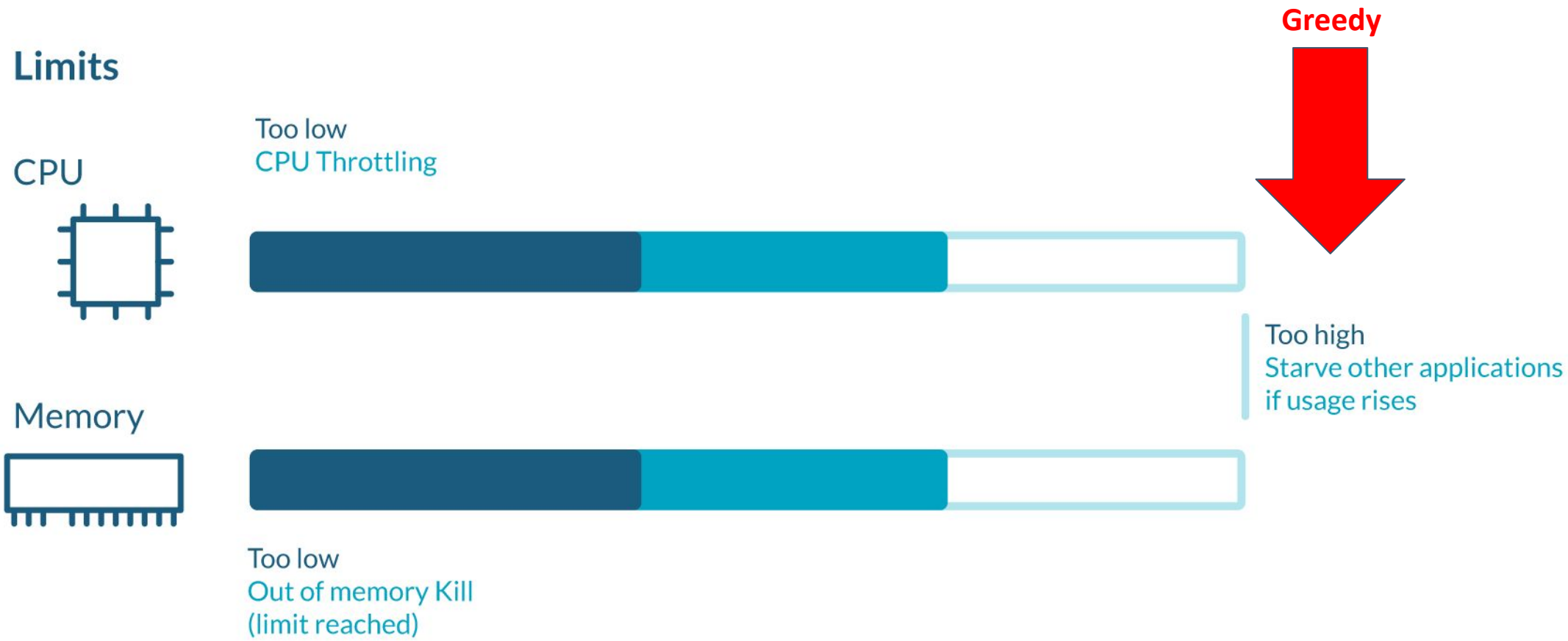
AWS EBS
AWS Elastic Block Store

AWS EKS
AWS EKS with Fargate
COMING SOON

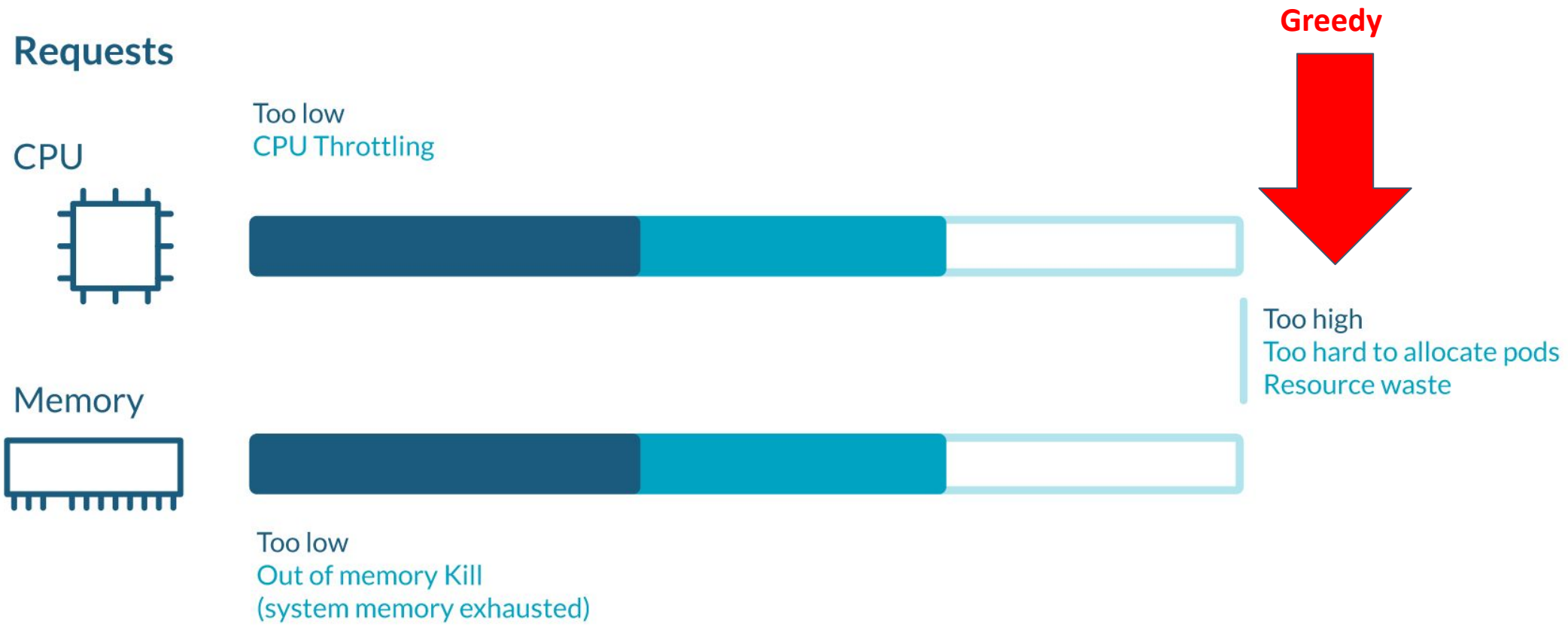
AWS ELB
AWS Elastic Load Balancing

The greedy developer problem

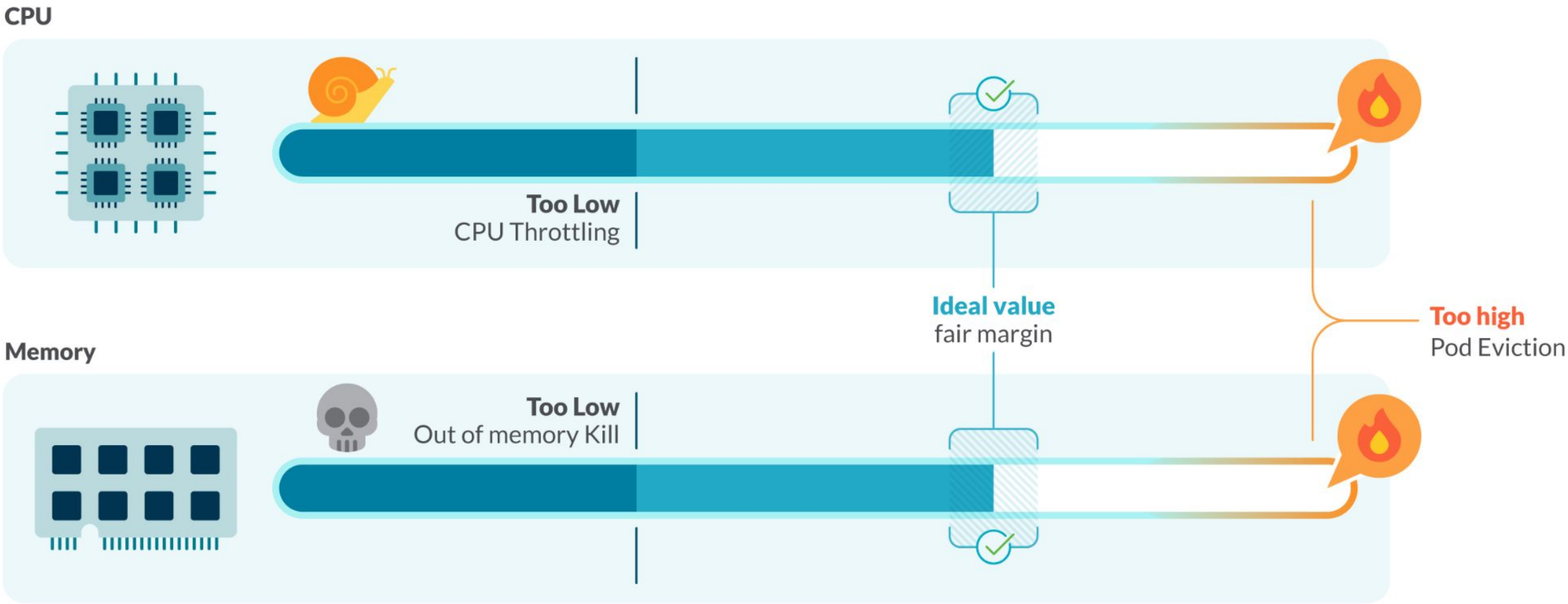
The greedy developer problem



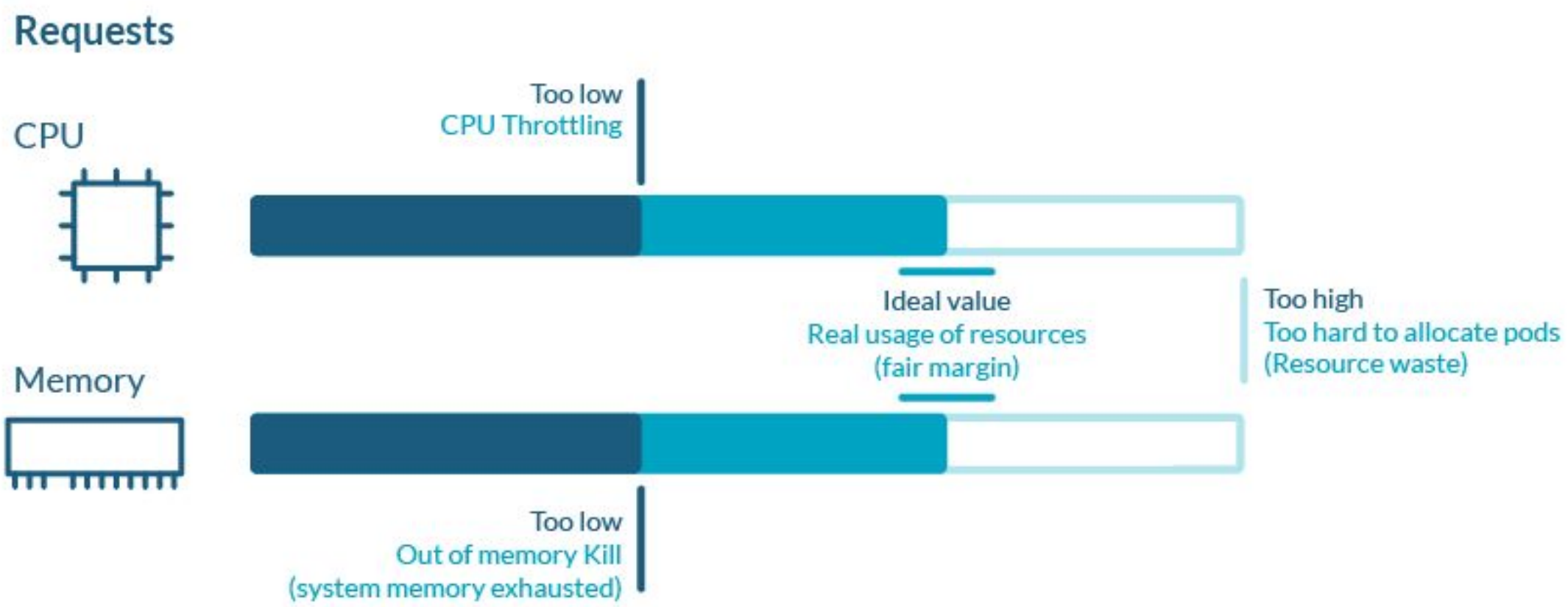
The greedy developer problem



What are the limits?



What are the requests?

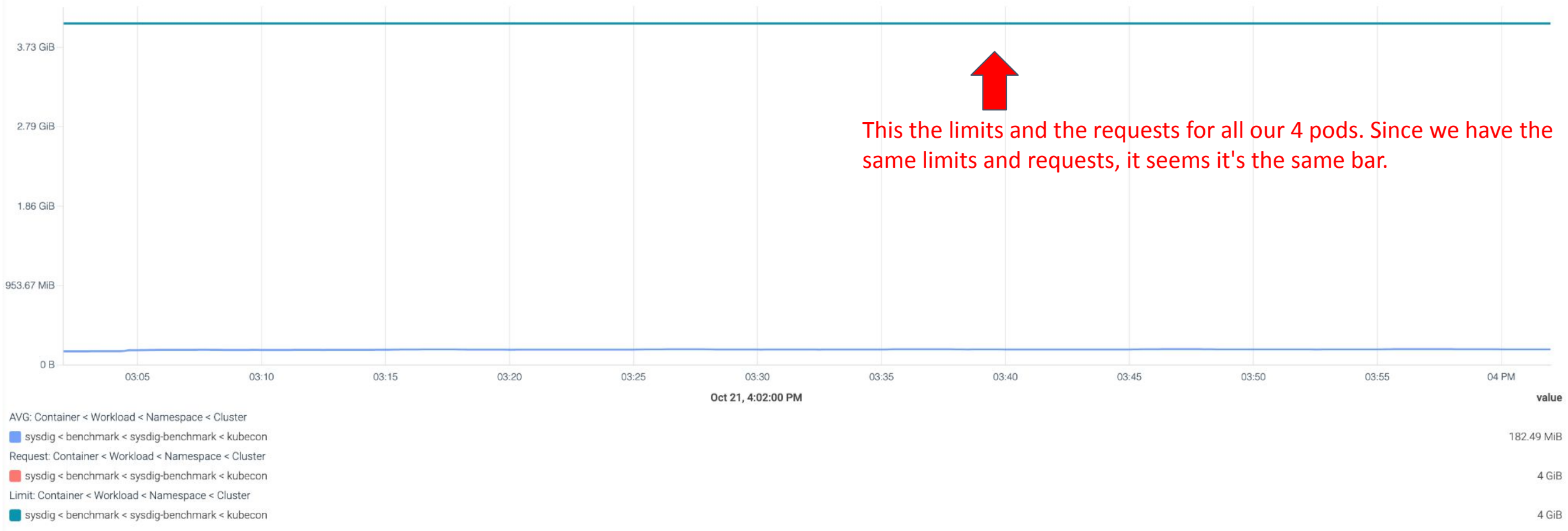


How looks like in a real scenario


```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: benchmark
  labels:
    app: sysdig-benchmark
spec:
  selector:
    matchLabels:
      app: sysdig-benchmark
    spec:
      containers:
      - name: sysdig
        image: sysdig
        resources:
          limits:
            memory: 1Gi
            cpu: 1
          requests:
           memory: 1Gi
           cpu: 1
```

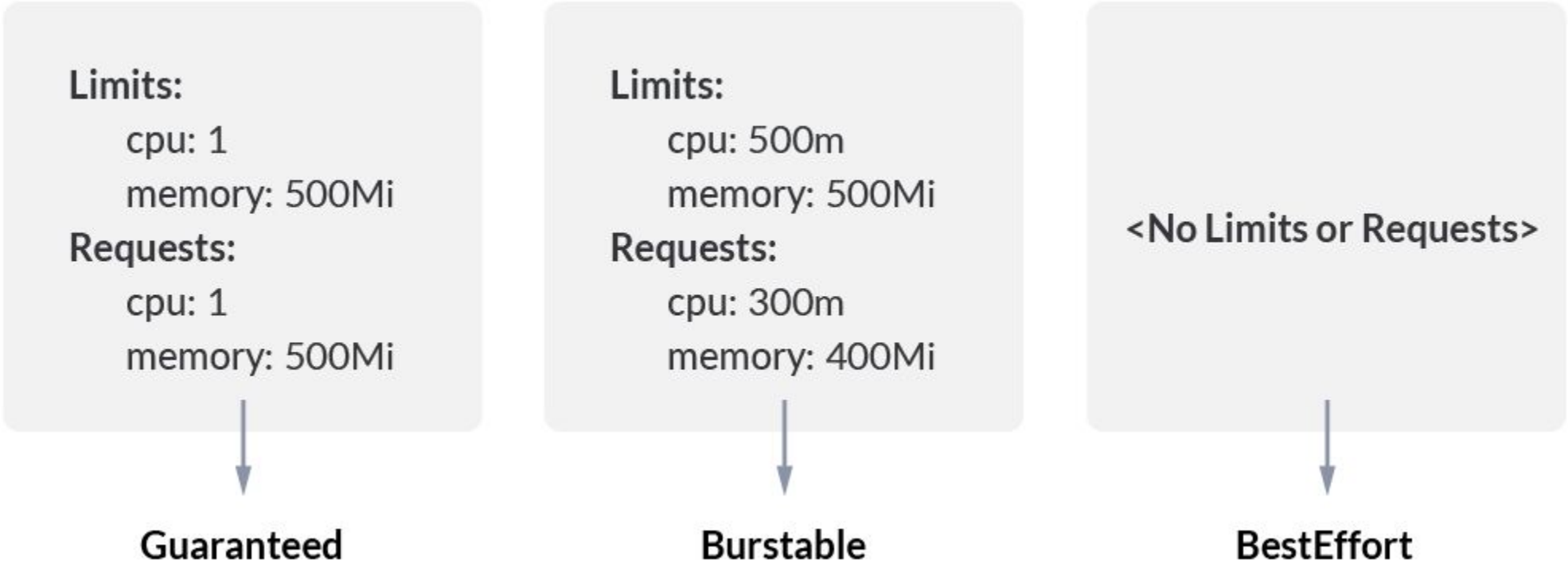
```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: benchmark
  labels:
    app: sysdig-benchmark
spec:
  selector:
    matchLabels:
      app: sysdig-benchmark
    spec:
      containers:
      - name: sysdig
        image: sysdig
        resources:
          limits:
            memory: 1Gi
            cpu: 1
          requests:
            memory: 1Gi
            cpu: 1
```

How looks like in a real scenario



This the limits and the requests for all our 4 pods. Since we have the same limits and requests, it seems it's the same bar.

QoS Classes


Limits:
cpu: 1
memory: 500Mi
Requests:
cpu: 1
memory: 500Mi
Guaranteed
Limits:
cpu: 500m
memory: 500Mi
Requests:
cpu: 300m
memory: 400Mi
Burstable
<No Limits or Requests>
BestEffort

QoS Classes

Guaranteed

For all containers:

- Limits set for CPU and Memory
- Requests set for CPU and Memory
- CPU Limits = CPU Requests
- Memory Limits = Memory Requests

Quite unlikely to be evicted

Example:
Database

Burstable

For at least a container:

- Limit set for CPU or Memory
- Requests set for CPU or Memory

Less likely to be evicted

Example: Api Gateway

BestEffort

For all containers:

- No Limits or Requests set

More likely to be evicted

Example:
Any non critical applications

A Pod evicted might not be that bad

- Unless your application is running in just one pod, don't worry about Pod Eviction.
- Pod Eviction is a natural thing in Kubernetes.
- After a change, the scheduler will arrange the pods.
- Don't worry about where's your pod.

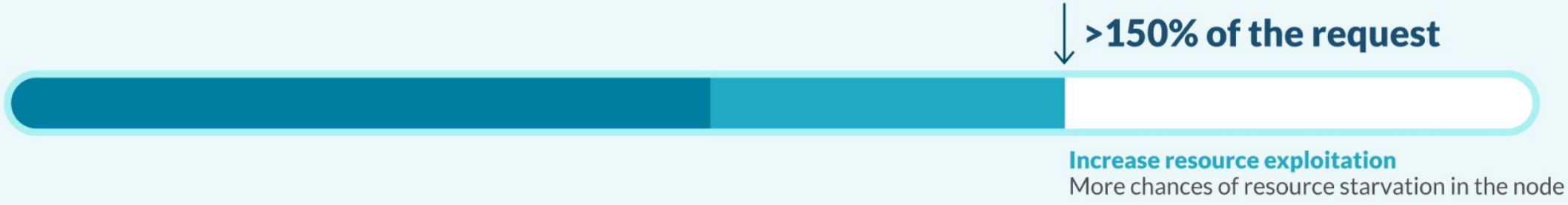
Quick review

Usually setting up limits and requests in your workloads is a good idea.

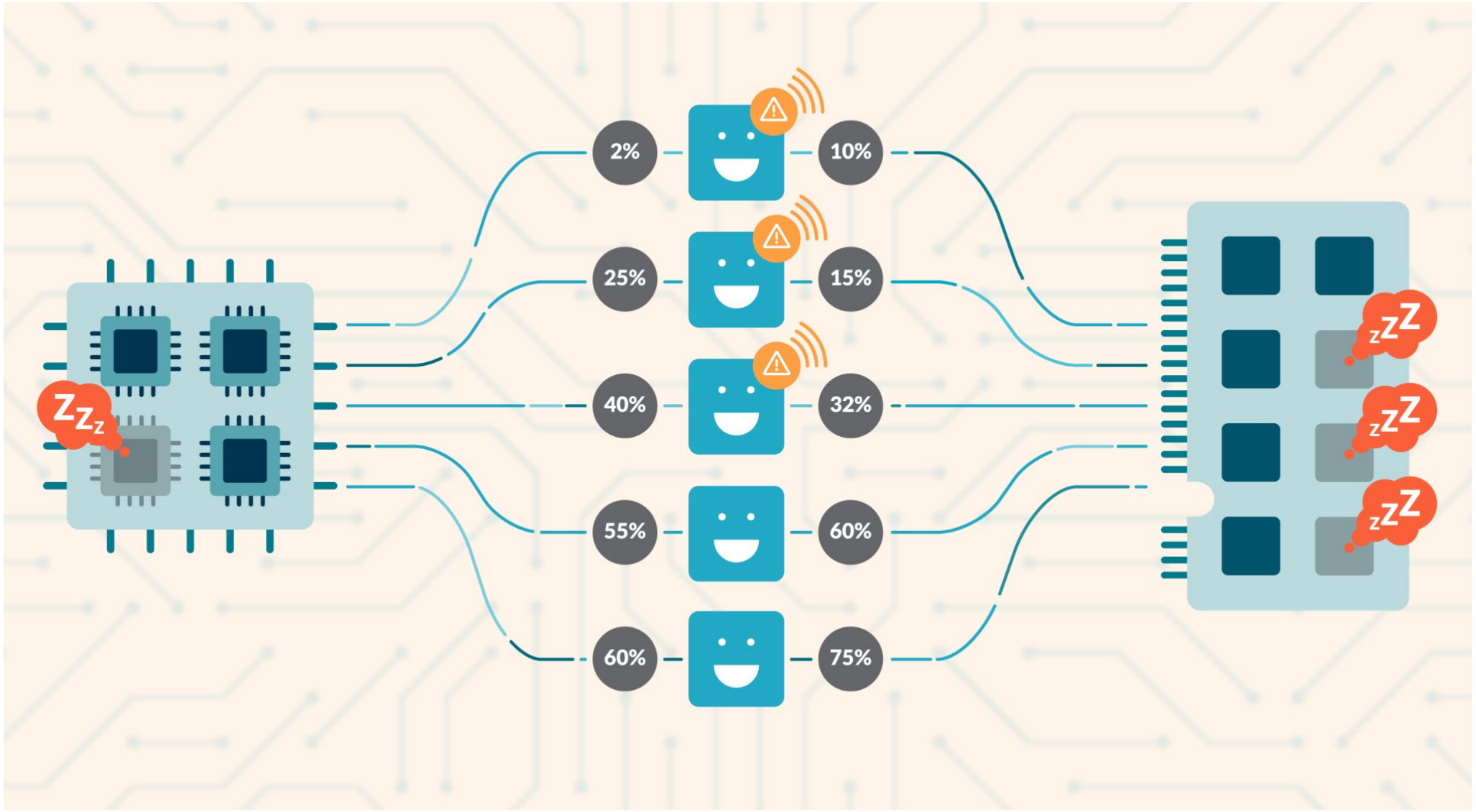
Limit rightsize strategies

Conservative

Aggressive



Request rightsize strategies



Why are we doing this?

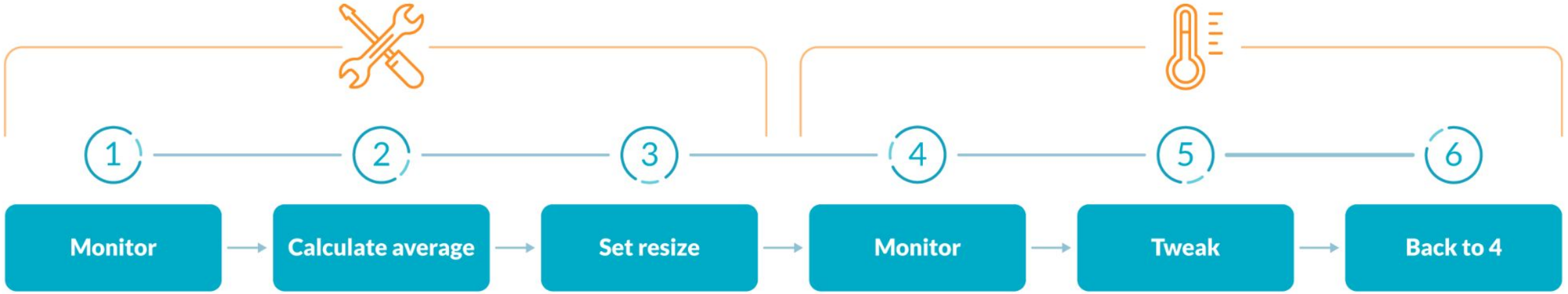
Rightsizing requests will help to

- Have a better understanding of the application.
- Unravel hidden problems in our application, that were masked by the high resource availability.
- Make the most of the resources.
- And yes, save some money!

We need a plan

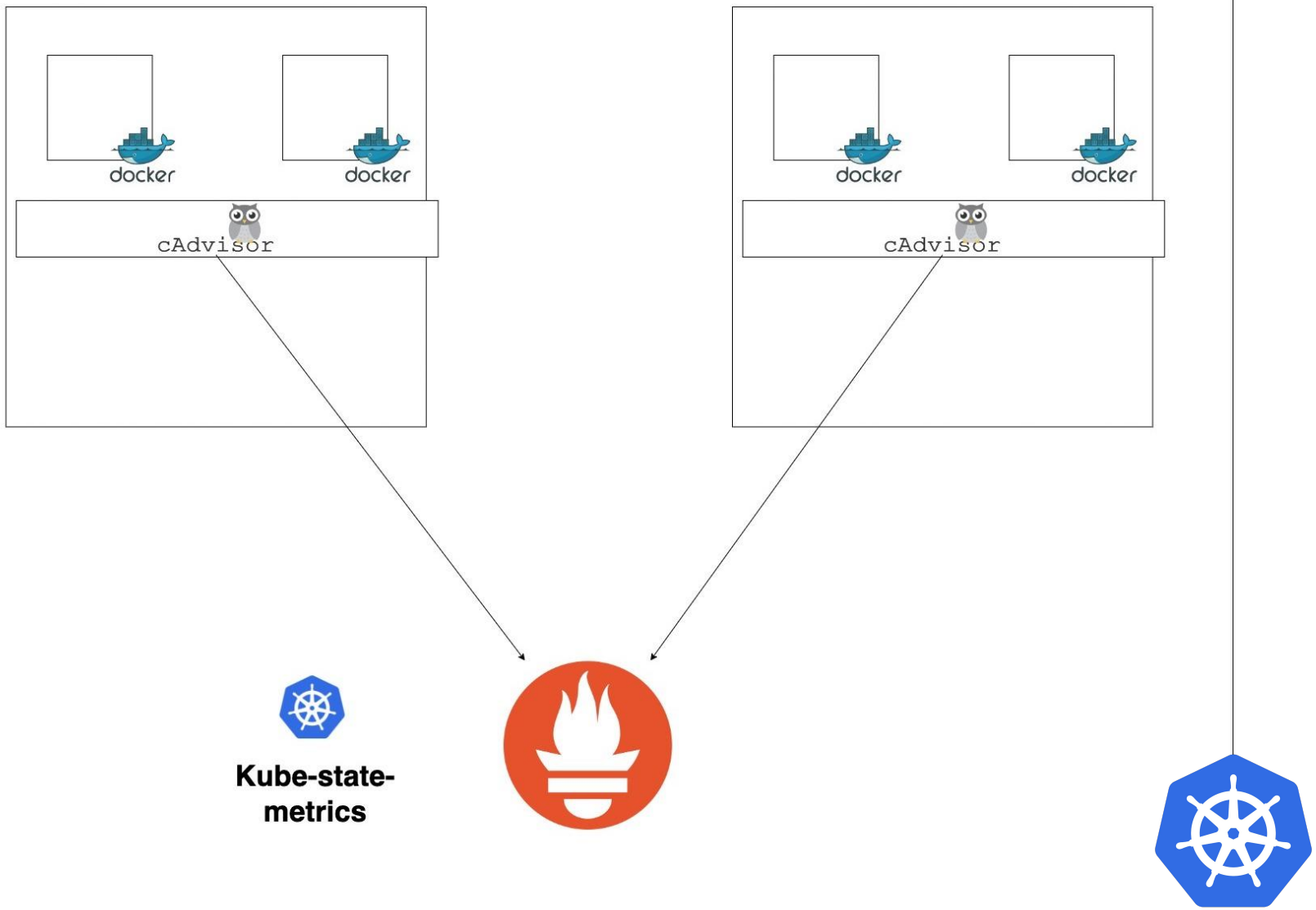
We have a plan

Rightsizing the workloads



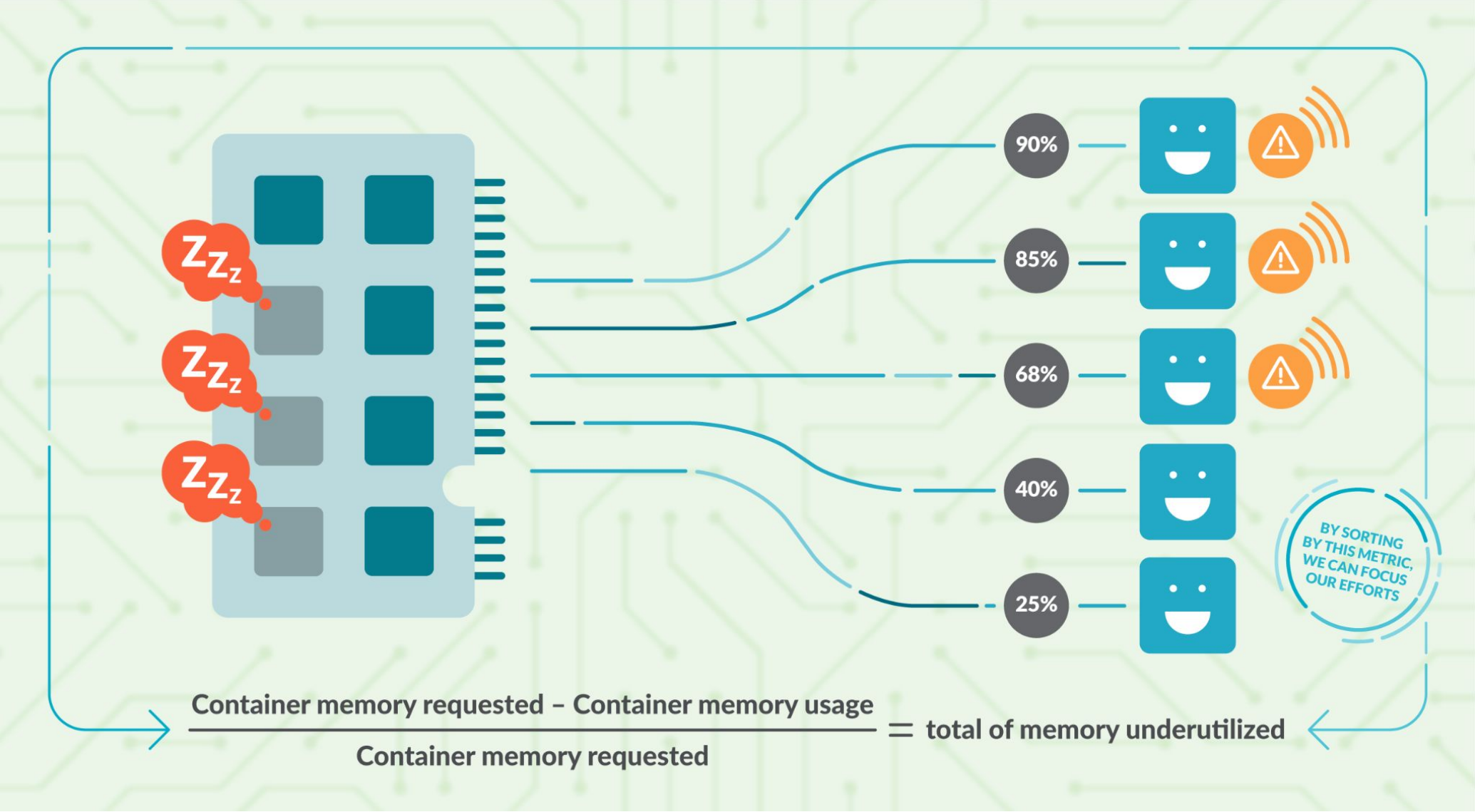
What do you need

The tools


docker
docker
cAdvisor
docker
docker
cAdvisor
Kube-state-metrics

First, detect unused resources

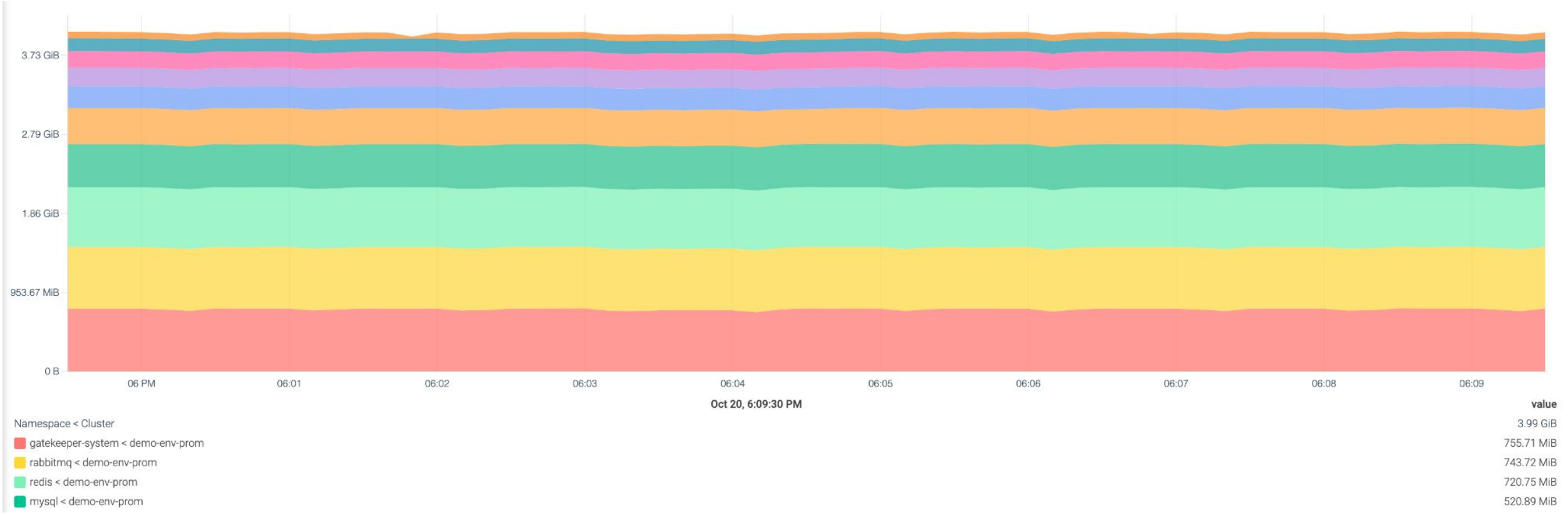
Calculate % unused Memory



Detect unused Memory


```
sum by(namespace, container, pod) (
    (
      rate(
        container_memory_usage_bytes{container!="POD",container!=""}[30m]
      )
    - on(namespace, pod, container) group_left()
      avg by(namespace, pod, container) (
        kube_pod_container_resource_requests{resource="memory"}
      )
    ) * -1 > 0
)
```

Detect underutilized Memory



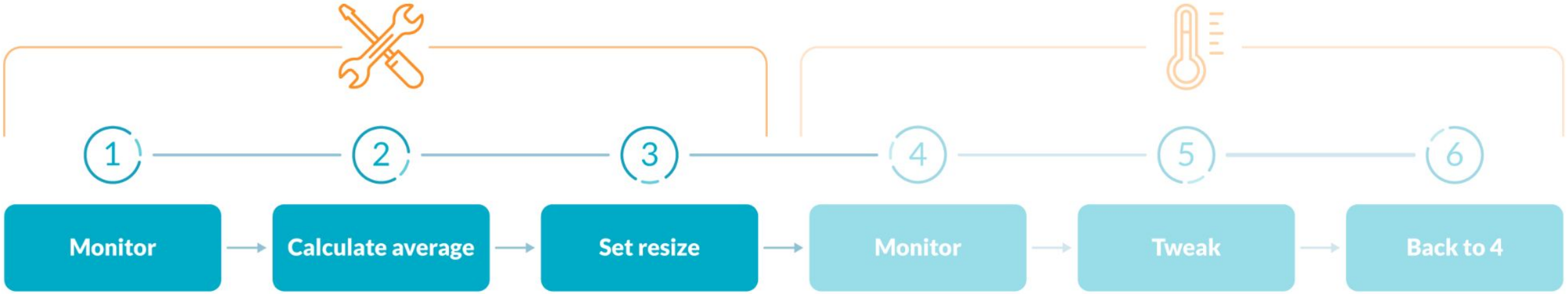
If you aggregate it just by namespace, you can have the

~~`git`~~ `kubectl blame`

Let's take some action: It's rightsizing time!

Rightsize the requests

Rightsizing the workloads



Rightsize the requests

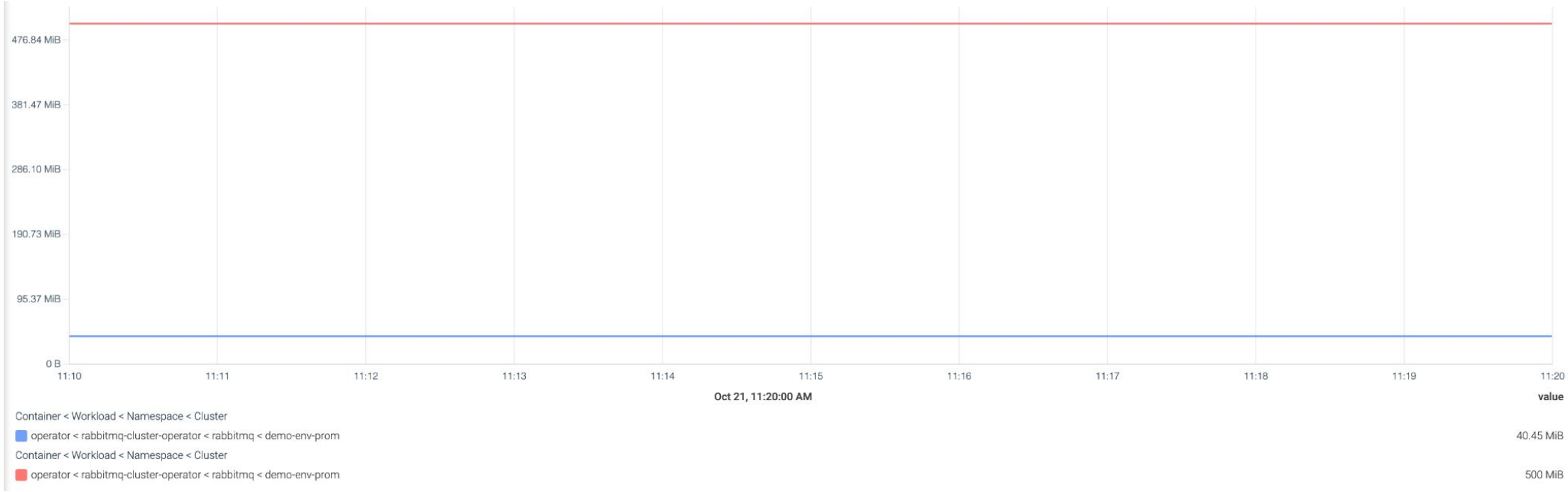
KubeCon CloudNativeCon North America 2022

Conservative Strategy:

Calculates the maximum resource usage for all the containers

A nice starting point. Go from this, and start lowering requests from there.

```
max by (namespace,owner_name,container)((rate(container_cpu_usage_seconds_total{container!="POD",container!=""}[5m])) *
on(namespace,pod) group_left(owner_name) avg by
(namespace,pod,owner_name)(kube_pod_owner{owner_kind=~"DaemonSet|StatefulSet|Replicaset"}))
```



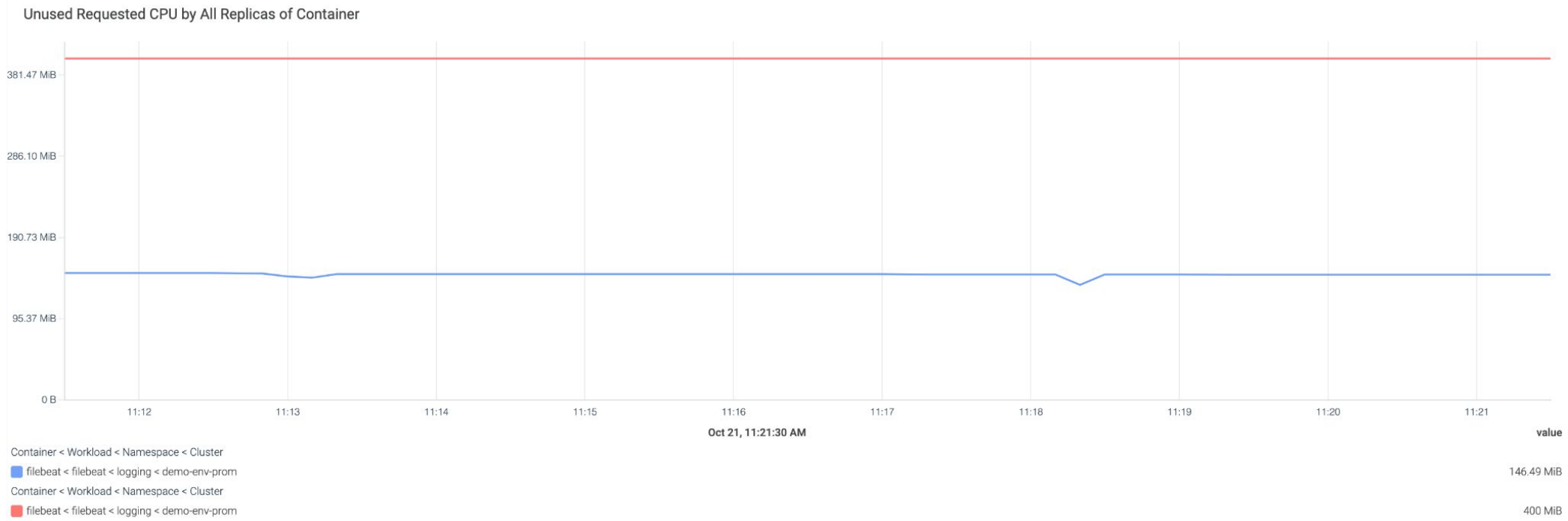
Rightsize the requests

KubeCon CloudNativeCon North America 2022

Aggressive strategy:

Calculates the average resource usage for all the containers

```
avg by (namespace,owner_name,container)((rate(container_cpu_usage_seconds_total{container!="POD",container!=""}[5m])) *
on(namespace,pod) group_left(owner_name) avg by
(namespace,pod,owner_name)(kube_pod_owner{owner_kind=~"DaemonSet|StatefulSet|Replicaset"}))
```



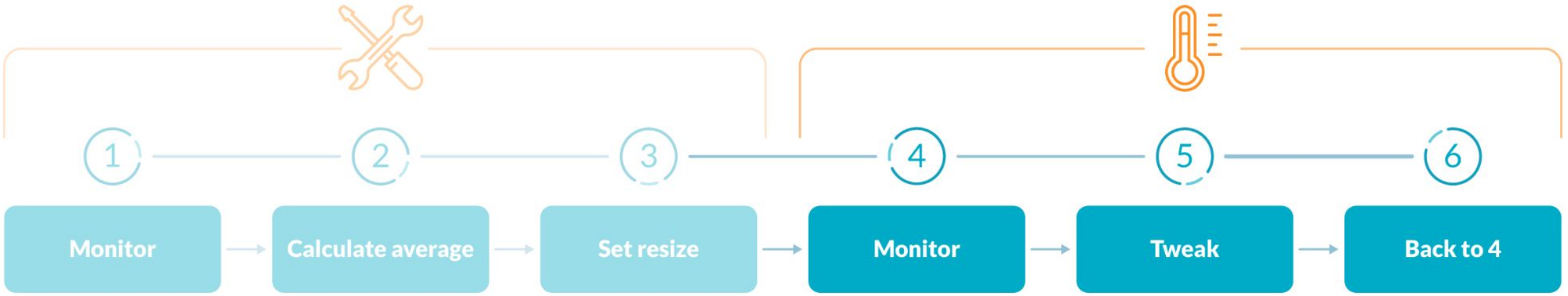
First resize


```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: benchmark
  labels:
    app: sysdig-benchmark
spec:
  selector:
    matchLabels:
      app: sysdig-benchmark
    spec:
      containers:
      - name: sysdig
        image: sysdig
        resources:
          limits:
            memory: 1Gi
            cpu: 1
          requests:
            memory: 1Gi
            cpu: 1
```

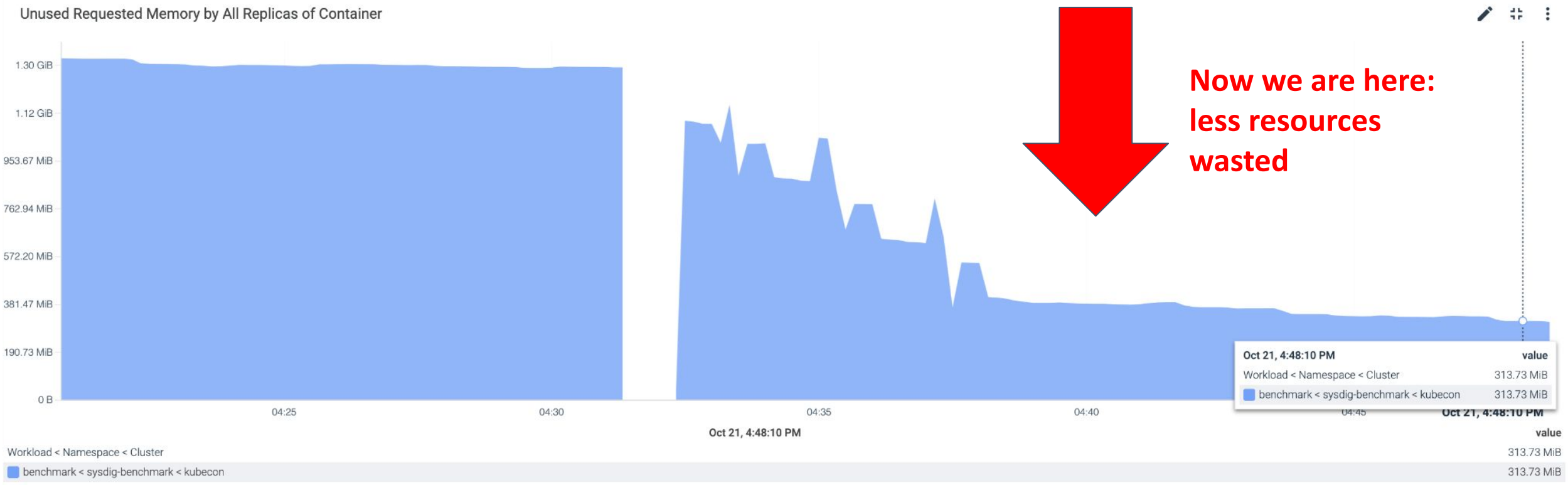
```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: benchmarkt
  labels:
    app: sysdig-benchmark
spec:
  selector:
    matchLabels:
      app: sysdig-benchmark
    spec:
      containers:
      - name: sysdig
        image: sysdig
        resources:
          limits:
            memory: 1Gi
            cpu: 1
          requests:
            memory: 512Mi
            cpu: 500m
```

Checking the impact

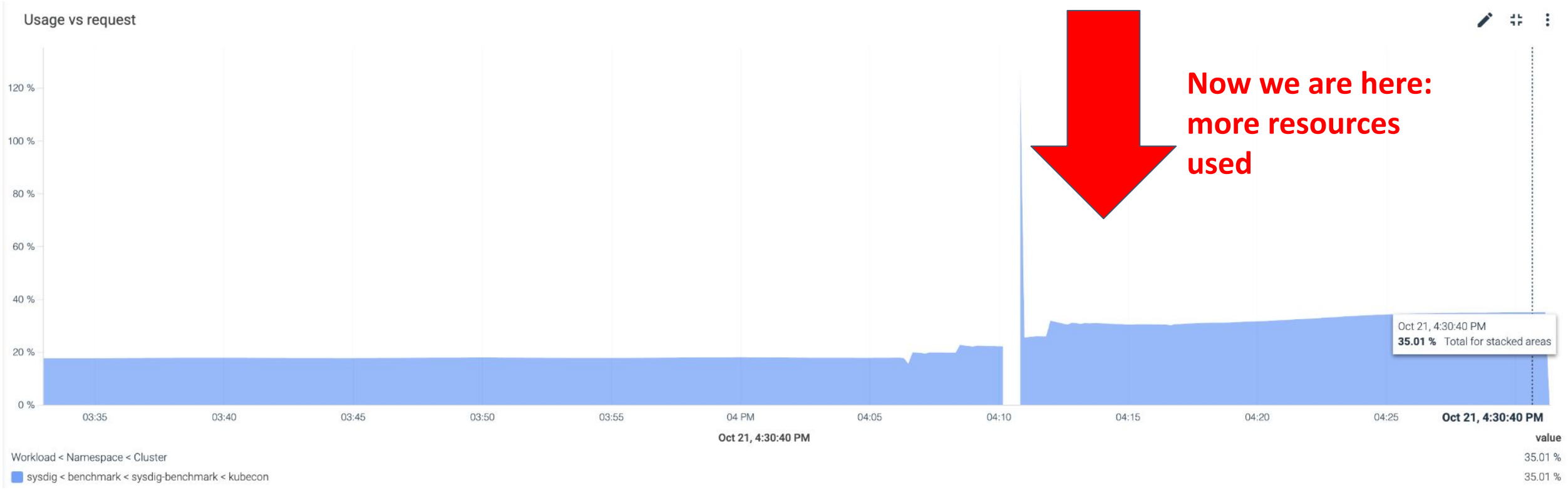
Checking the impact



Checking the impact

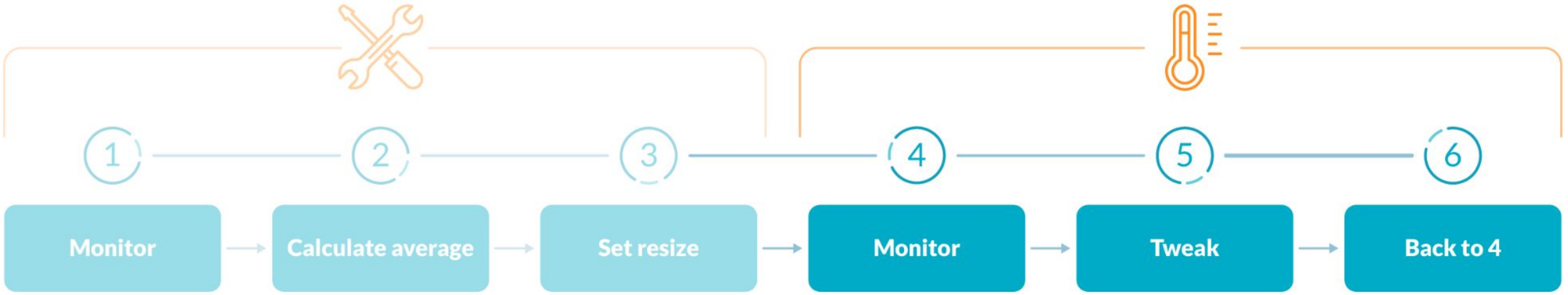


Checking the impact



We're not there yet

Checking the impact

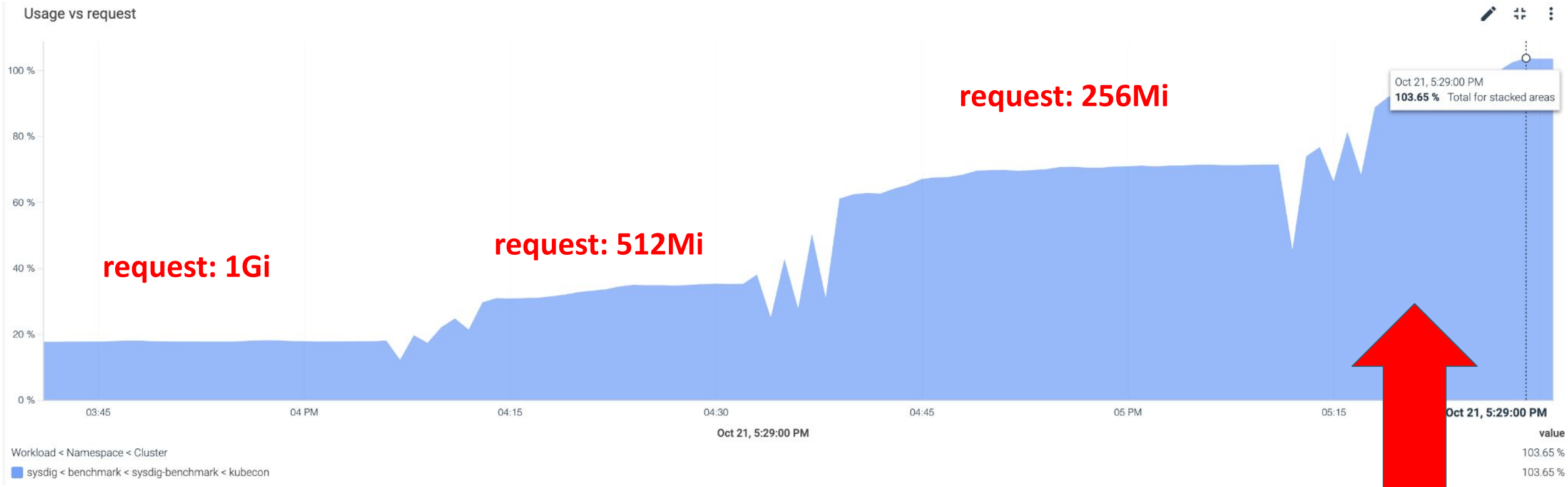


We tweak it AGAIN


```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: benchmark
  labels:
    app: sysdig-benchmark
spec:
  selector:
    matchLabels:
      app: sysdig-benchmark
    spec:
      containers:
      - name: sysdig
        image: sysdig
        resources:
          limits:
            memory: 1Gi
            cpu: 1
          requests:
            memory: 512Mi
            cpu: 500m
```

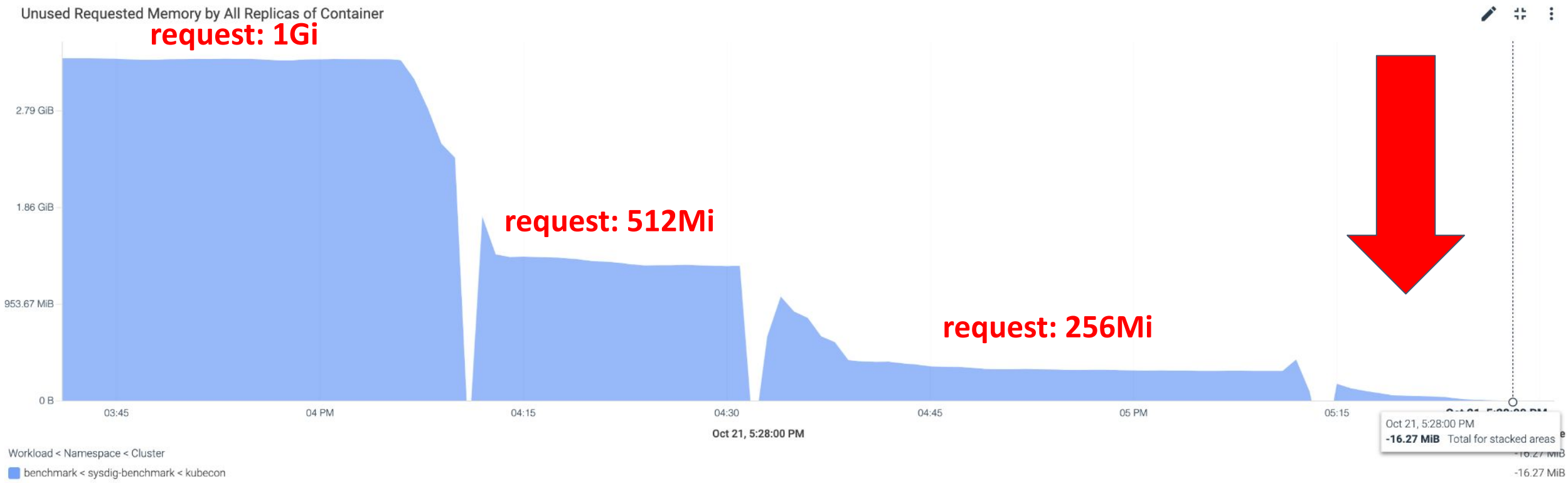
```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: benchmark
  labels:
    app: sysdig-benchmark
spec:
  selector:
    matchLabels:
      app: sysdig-benchmark
    spec:
      containers:
      - name: sysdig
        image: sysdig
        resources:
          limits:
            memory: 1Gi
            cpu: 1
          requests:
            memory: 256Mi
            cpu: 250m
```

Checking the impact



And we could go even further lowering the request to 128Mi

Checking the impact



And we could go even further lowering the request to 128Mi

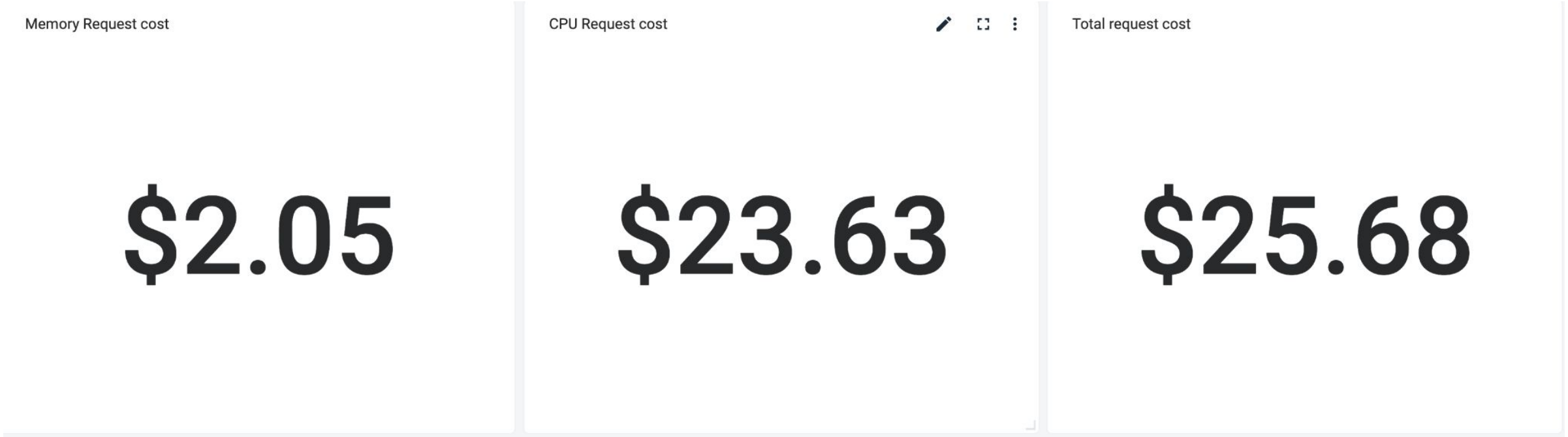
Done!

Real talk

Let's talk about the money

Before rightsizing

After rightsizing



Savings

Total request cost

$104.68

Total request cost

$25.68

Conclusions

- We learned that our application didn't need so many resources.
- We saved $75 after resizing just one workload.
- We started using a workflow easily repeatable to monitor the resource usage of our application.

Thank you!
Questions ?