
KubeCon
CloudNativeCon
Europe 2023
TiKV

WG Batch: What's new and What's Next?

Swati Sehgal, @swsehgal, Red Hat,
Aldo Culquicondor, @alculquicondor, Google

What is the WG Batch

- Forum to discuss enhancements to better support batch workloads in core Kubernetes (eg. HPC, AI/ML, data analytics, CI).
- A goal is to reduce fragmentation in the Kubernetes batch ecosystem.
- Stakeholders:
  - SIG Scheduling
  - SIG Apps
  - SIG Node
  - SIG Autoscaling
- Other participants: ecosystem developers from Kubeflow, Armada, Yunikorn, among others.
- In scope:

Additions to the batch APIs (Job, CronJob)

Job queuing primitives

Primitives to maximize clusters utilization

Support for specialized hardware

Additions to the batch APIs

Additions to the batch APIs (Job, CronJob)

Job queuing primitives

Primitives to maximize clusters utilization

Support for specialized hardware

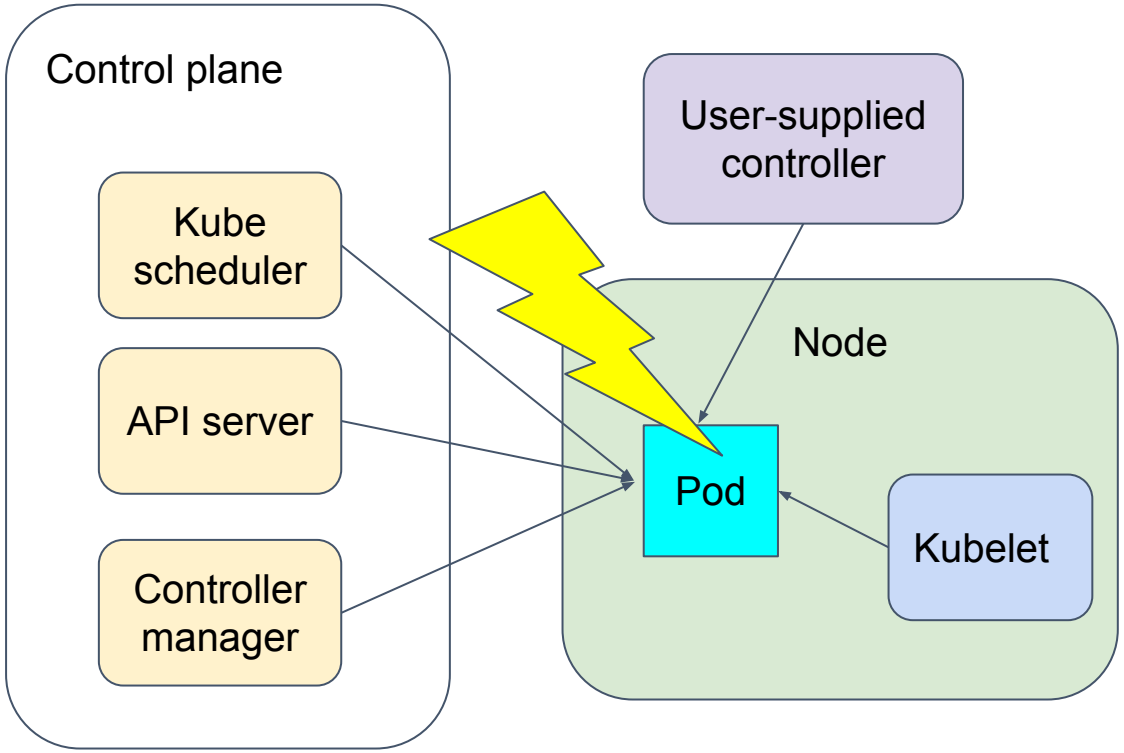
Job: More scalable than ever

- Job tracking with finalizers reached General Availability in Kubernetes 1.26 🎉
- Prior to this feature, the job controller lost progress of Jobs when Pods were deleted
  - 💥 Incompatible with Pod GC
- Pods get a finalizer `batch.kubernetes.io/job-tracking` before counting in the Job status.
- Indexed Jobs of up to 100k Pods can be processed in minutes 🚀

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: sample-job
spec:
  parallelism: 10
  completions: 100
  completionMode: Indexed
  template:
    spec:
      containers:
      - name: job
        image: foo
status:
  active: 10
  completedIndexes: 0-9
  ready: 2
  succeeded: 10
  startTime: "2023-04-11T19:08:27Z"
```

Blogpost: [Job Tracking, to Support Massively Parallel Batch Workloads, Is Generally Available]

Job: Pod Failure Policies



Pod disruption scenarios

```yaml
backoffLimit: 6
podFailurePolicy:
  rules:
  - action: FailJob
    onPodConditions:
    - type: ConfigIssue
  - action: Ignore
    onPodConditions:
    - type: DisruptionTarget
  - action: FailJob
    onExitCodes:
      operator: In
      values: [1,2,126,127,128,139]
  - action: Ignore
    onExitCodes:
      operator: In
      values: [130,137,138,147]
```

- **New** in 1.27: kubelet guarantees all Pods reach a terminal phase

[Enabling HPC and ML Workloads with the Latest Kubernetes Job Features]

Job: Mutable completions

- `completions` is now mutable for Indexed Jobs
- Requirement: `parallelism` and `completions` need to match
- Use case: elastic jobs, such as Pytorch

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: elastic-job
spec:
  parallelism: 100
  completions: 100
  completionMode: Indexed
  template:
    spec:
      containers:
      - name: job
        image: foo
```

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: elastic-job
spec:
  parallelism: 90
  completions: 90
  completionMode: Indexed
  template:
    spec:
      containers:
      - name: job
        image: foo
```

Job: What's next

- 🚧 Subproject (in SIG Apps): JobSet
  - Use case: jobs that have Pods with different roles, such as driver-workers.
  - Each role is implemented as an Indexed Job.
  - Automates pod-to-pod communication and auth keys.
  - Corollary: #116993 creating environment variables from a file
- 💡 KEP3850: backoffLimit per Index
  - Use case: Parallel applications where each worker process and independent piece of data.
  - Corollary: #109131 retry specific indexes
- 💡 KEP3939: Count terminating Pods as active.
  - Use case: Tightly coupled applications that don't support more than one worker per Index.

Job: What's next

- 💡Open discussions:
  - #113221 Mutable scheduling directives when suspended
  - #111948 Deadline for Pending Pods
  - #115716 DaemonJob: Run at least once per Node
  - #115066 Stateful Indexed Job: Automate creation of PVCs for each index.

Job queueing and cluster utilization

Additions to the batch APIs (Job, CronJob)

Job queuing primitives

Primitives to maximize clusters utilization

Support for specialized hardware

Kueue: cloud-native job queueing

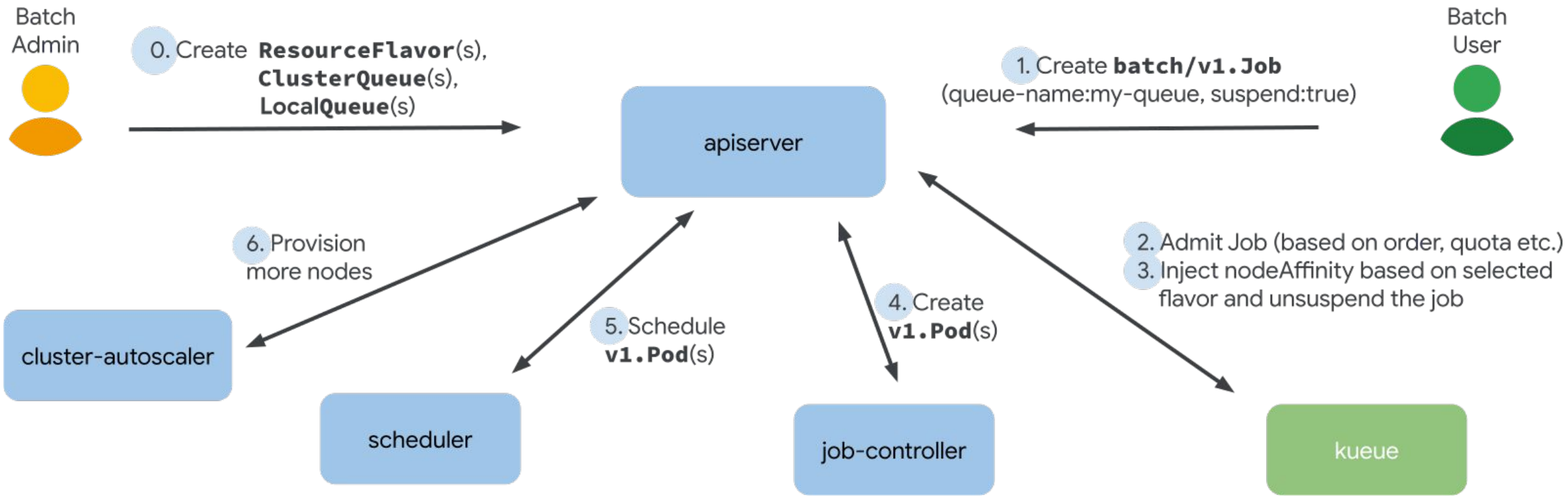
A Kubernetes-native job queueing system, offering:

- Resource quota management, with borrowing and preemption semantics.
- Resource fungibility in heterogeneous clusters.
- Support for k8s batch/v1.Job and kubeflow's MPIJob.
- Extension points and libraries for supporting custom job CRDs.
- More Job integrations coming soon

Latest version: v0.3 🎉

Kueue: overview

Design principle: compatibility and separation of concerns with standard k8s components: kube-scheduler, kube-controller-manager, cluster-autoscaler.



Kueue: What's next

- ✨ Roadmap for v0.4
  - Improvements to WaitForPodsReady
  - Improvements to Preemption
  - [Nice-to-have] Support for Ray and kubeflow
- 💡 Open k8s discussions:
  - KEP3838 Mutable scheduling directives for gated Pods
  - #107294 A suspend/queueuing subresource

Batch+HPC Day:
Building a Batch System for the Cloud with Kueue

Support for specialized hardware

Additions to the batch APIs (Job, CronJob)

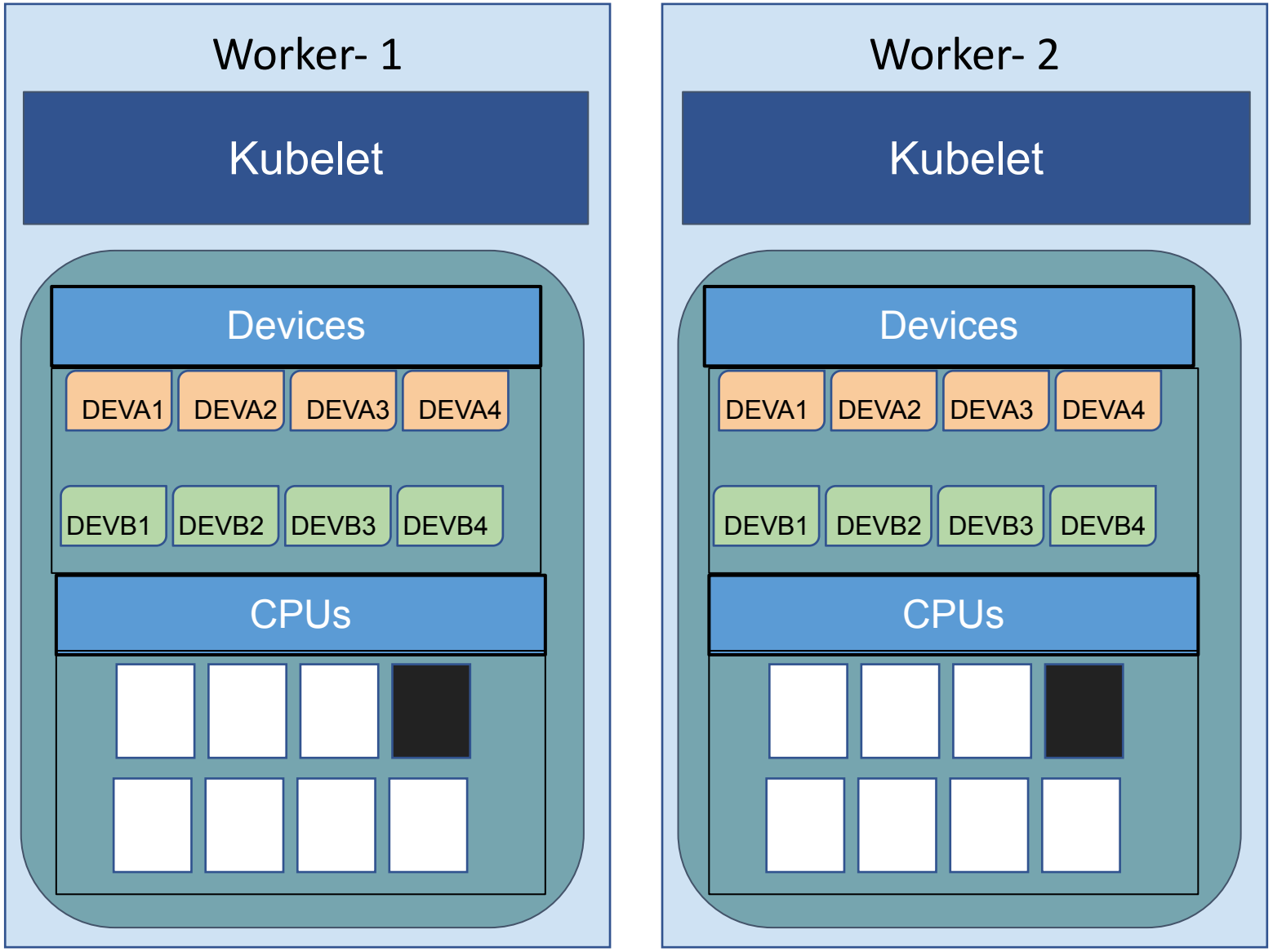
Job queuing primitives

Primitives to maximize clusters utilization

Support for specialized hardware

Topology-aware Scheduling - Problem statement

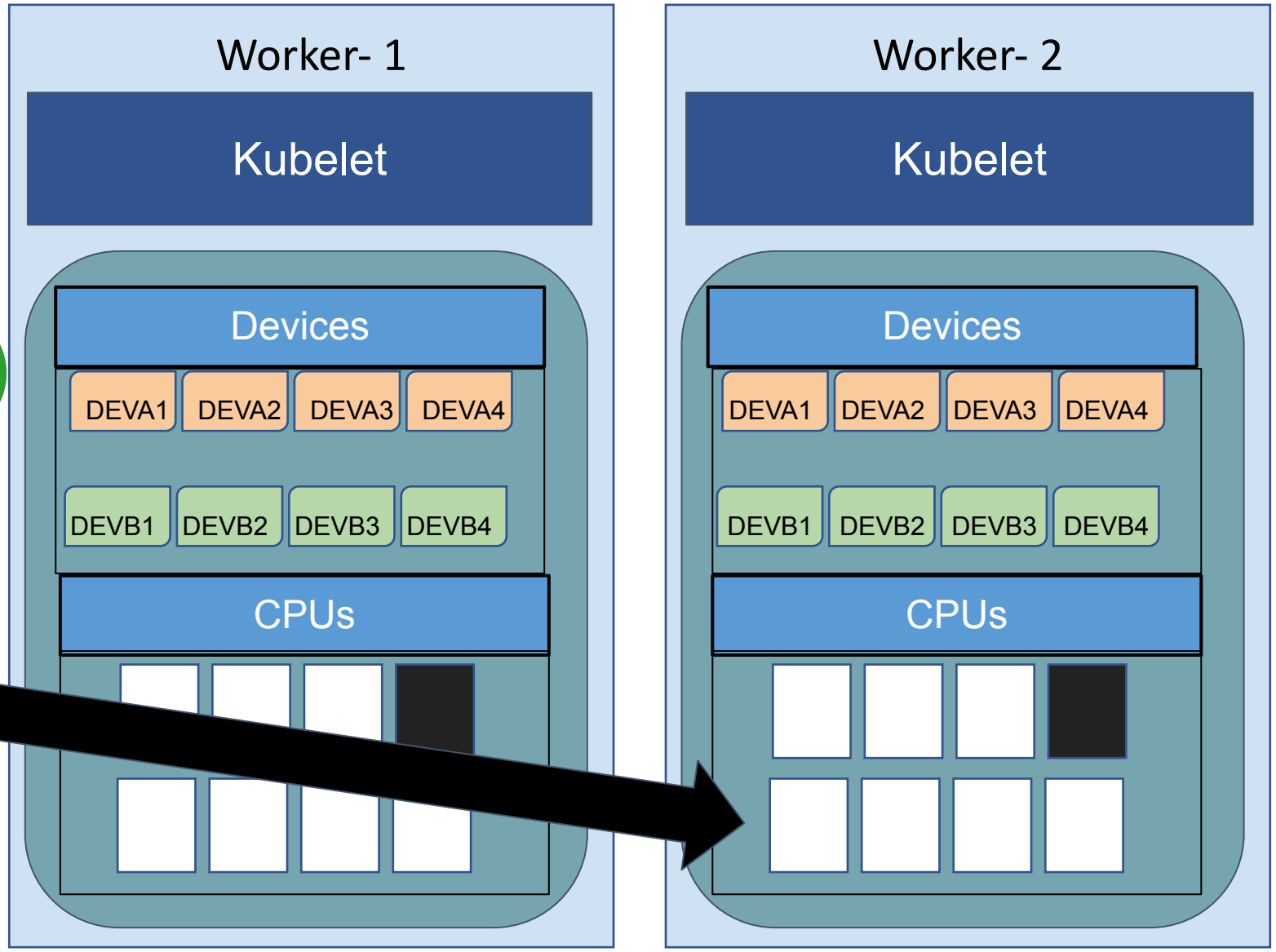

```
apiVersion: v1
kind: Pod
Spec:
  containers:
  - name: app
    Resources:
      Requests:
        cpu: 1
        memory: 2Gi
        example.com/deviceA: 1
        example.com/deviceB: 1
      Limits:
        cpu: 1
        memory: 2Gi
        example.com/deviceA: 1
        example.com/deviceB: 1
```



Topology-aware Scheduling - Problem statement

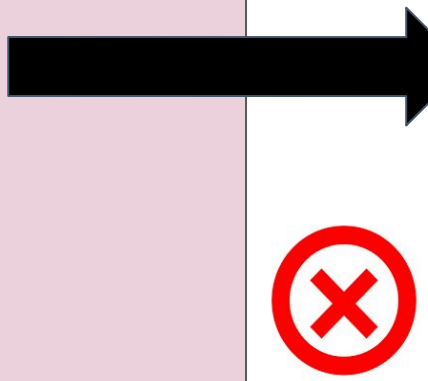

```
apiVersion: v1
kind: Pod
Spec:
  containers:
  - name: app
    Resources:
      Requests:
        cpu: 1
        memory: 2Gi
        example.com/deviceA: 1
        example.com/deviceB: 1
      Limits:
        cpu: 1
        memory: 2Gi
        example.com/devic
        example.com/deviceB: 1
```

Pod Running!

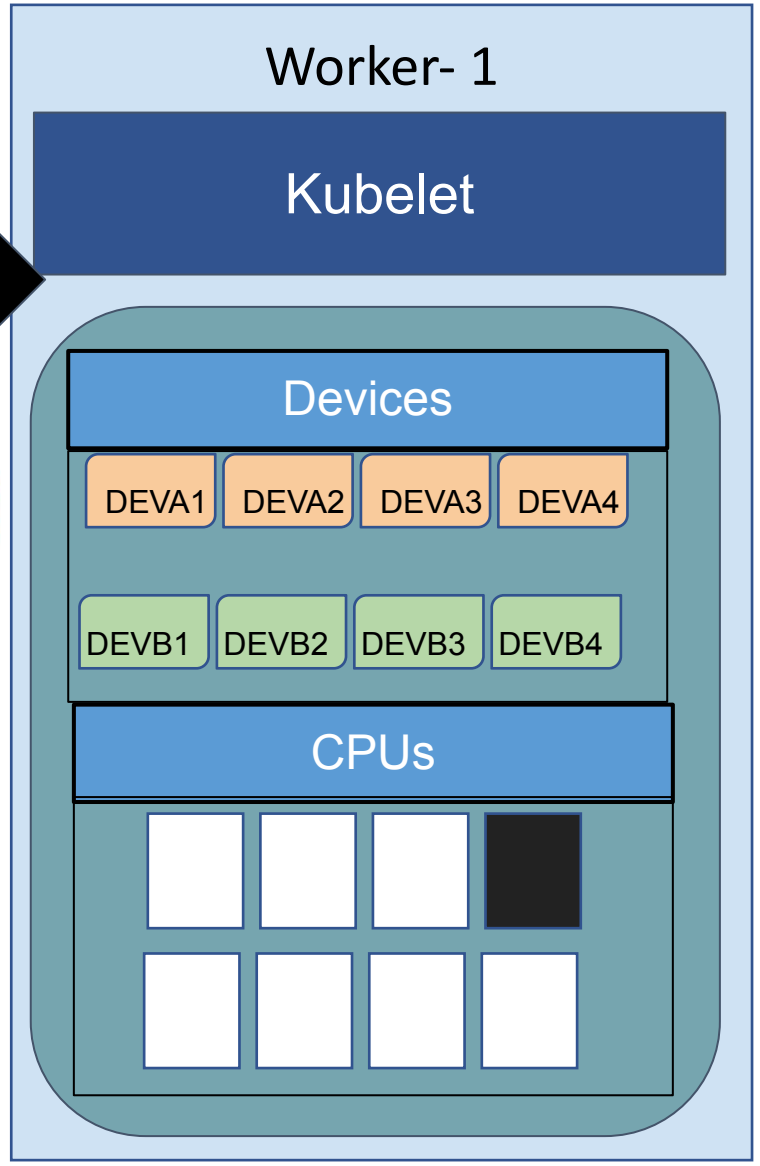


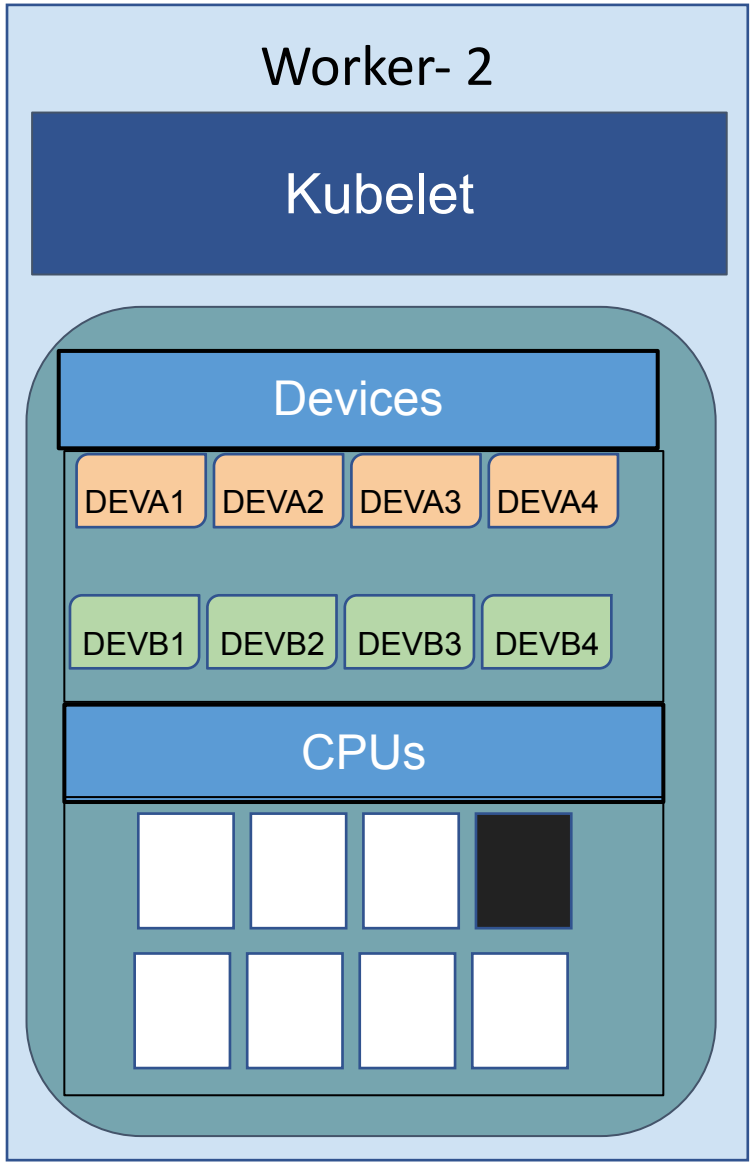
Topology-aware Scheduling - Problem statement


```
apiVersion: v1
kind: Pod
Spec:
  containers:
  - name: app
    Resources:
      Requests:
        cpu: 1
        memory: 2Gi
        example.com/deviceA: 1
        example.com/deviceB: 1
      Limits:
        cpu: 1
        memory: 2Gi
        example.com/deviceA: 1
        example.com/deviceB: 1
```



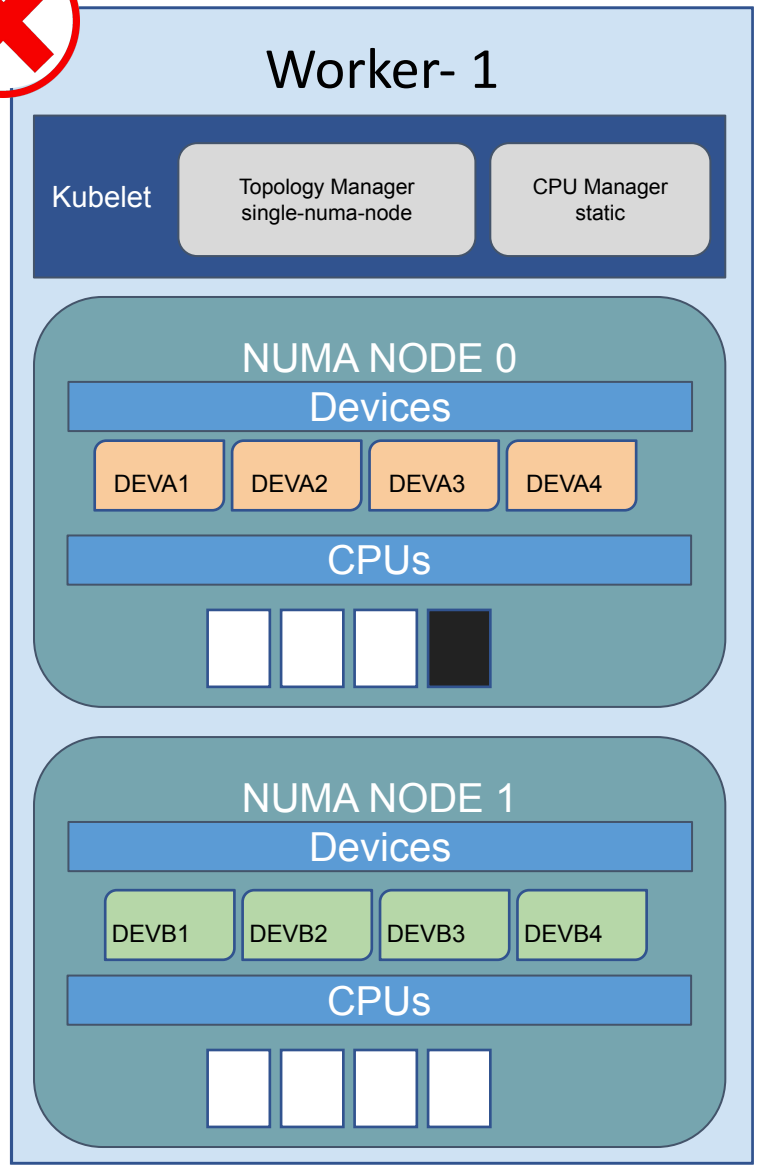
Pod Rejected!





Topology-aware Scheduling - Zooming in


```
apiVersion: v1
kind: Pod
Spec:
  containers:
  - name: app
    Resources:
      Requests:
        cpu: 1
        memory: 2Gi
        example.com/deviceA: 1
        example.com/deviceB: 1
      Limits:
        cpu: 1
        memory: 2Gi
        example.com/deviceA: 1
        example.com/deviceB: 1
```



Topology-aware Scheduling - Phase 1

An out-of-tree solution.

Development started in 2020 (alongside Kubernetes v1.22)

1. Topology-aware Scheduler Plugin:

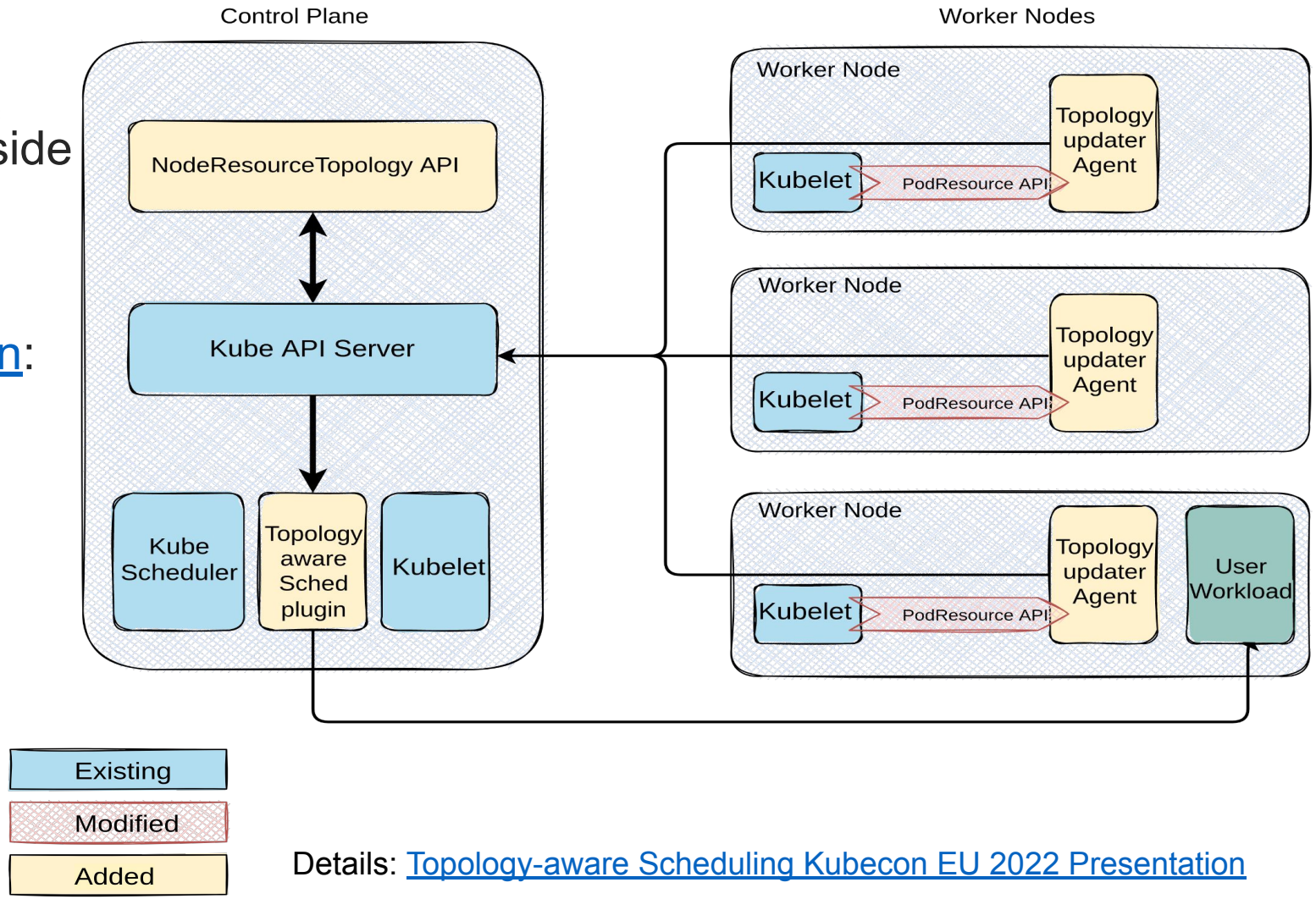
   (filter + scoring)

2. NodeResourceTopology API

3. Topology Updater Agent

   a. Node Feature Discovery

   b. Resource Topology Exporter



Details: Topology-aware Scheduling Kubecon EU 2022 Presentation

Topology-aware Scheduling Phase 2

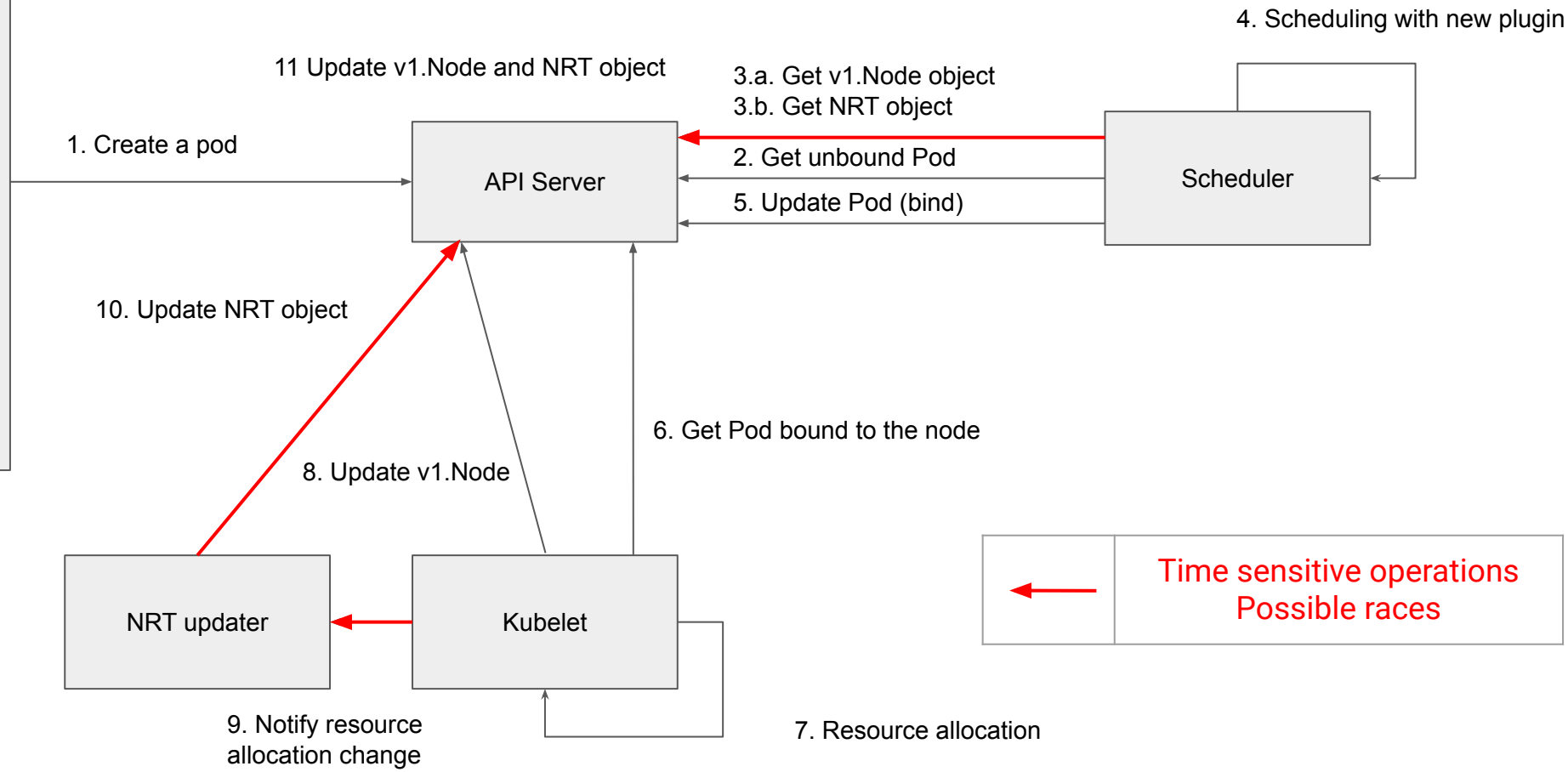
Improvements were made across the stack for stabilization of the solution

Updates were made to the three main components:

1. **Topology-aware Scheduler Plugin**
2. **NodeResourceTopology API**
3. Topology Updater Agent: **Node Feature Discovery**

Time sensitive operations

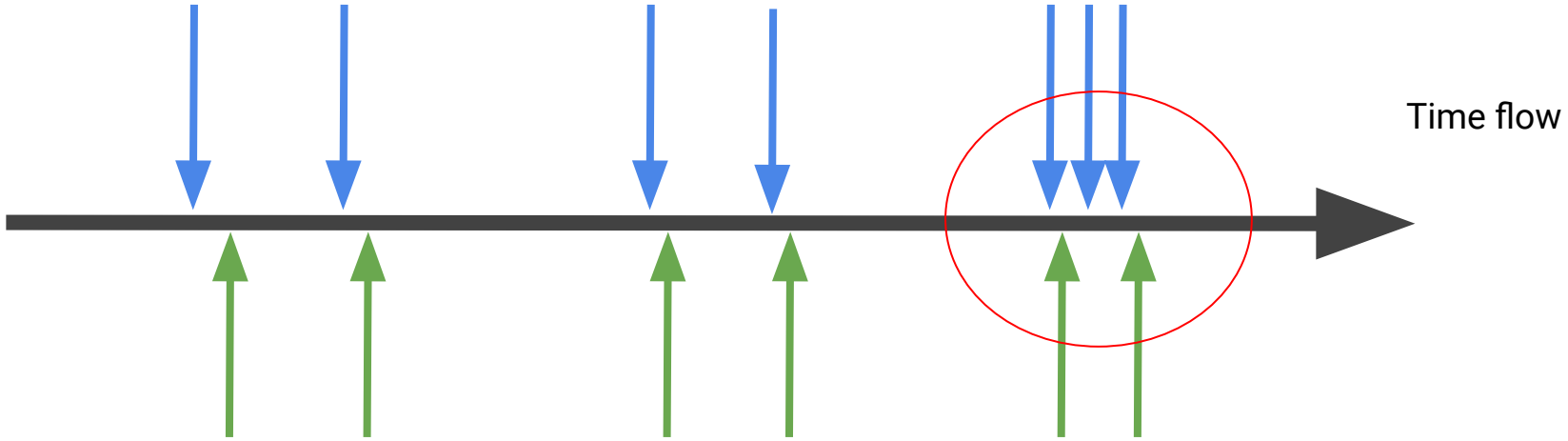
```
apiVersion: v1
kind: Pod
Spec:
  containers:
  - name: dpdk-app1
    Resources:
      Requests:
        cpu: 4
        memory: 2Gi
        hugepages-1Gi : “2Gi”
        intel.com/vf: “1”
      Limits:
        ...
```



NRT = Node Resource Topology

Race between scheduler and updater

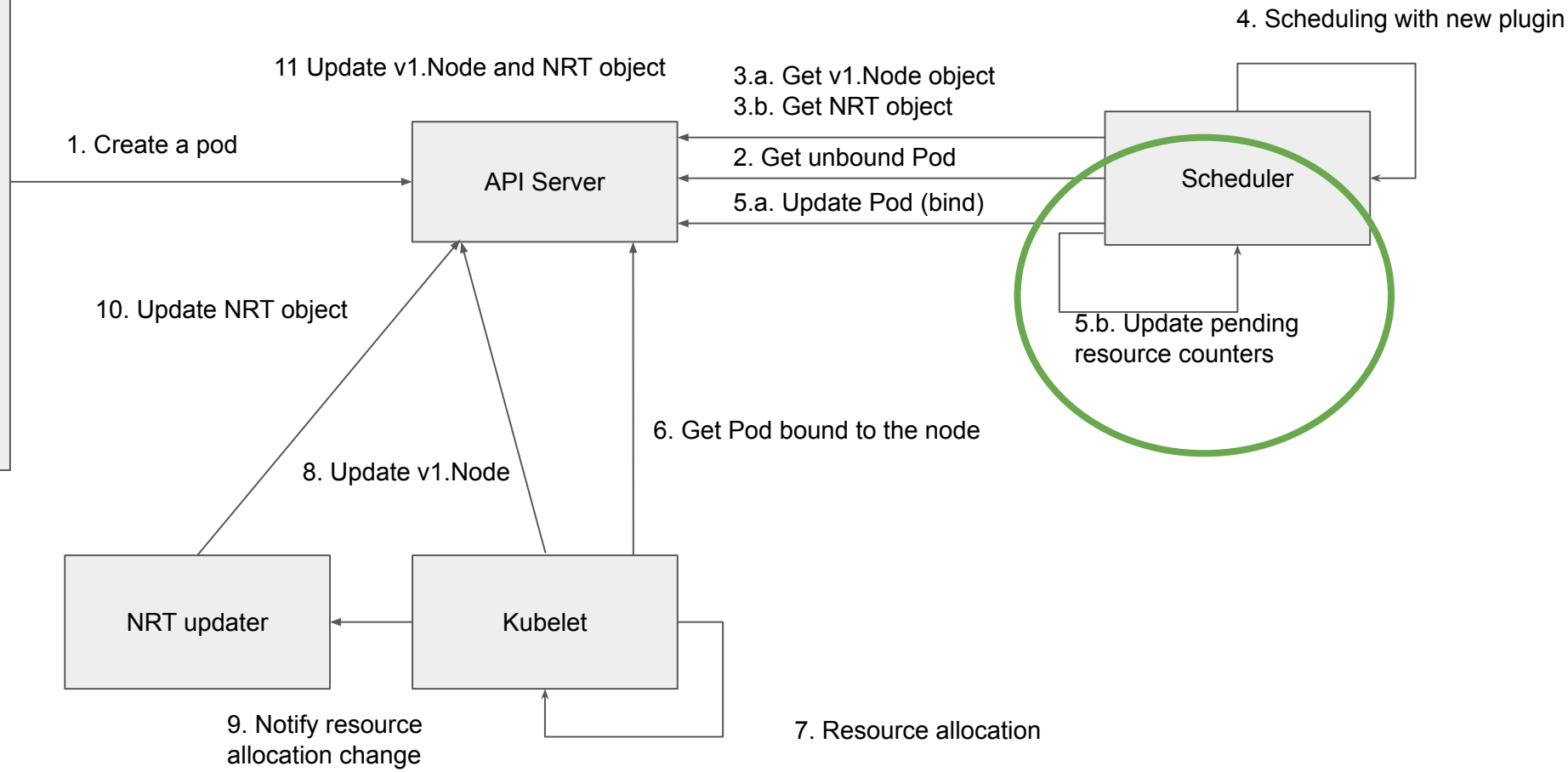
Pod to be scheduled - each arrow is a new pod



Node Resource Topology (NRT) objects updates, as seen by the scheduler - each arrow is update

Scheduler plugin: Local cache with reserve plugin

```
apiVersion: v1
kind: Pod
Spec:
  containers:
  - name: dpdk-app1
    Resources:
      Requests:
        cpu: 4
        memory: 2Gi
        hugepages-1Gi : “2Gi”
        intel.com/vf: “1”
      Limits:
        ...
```

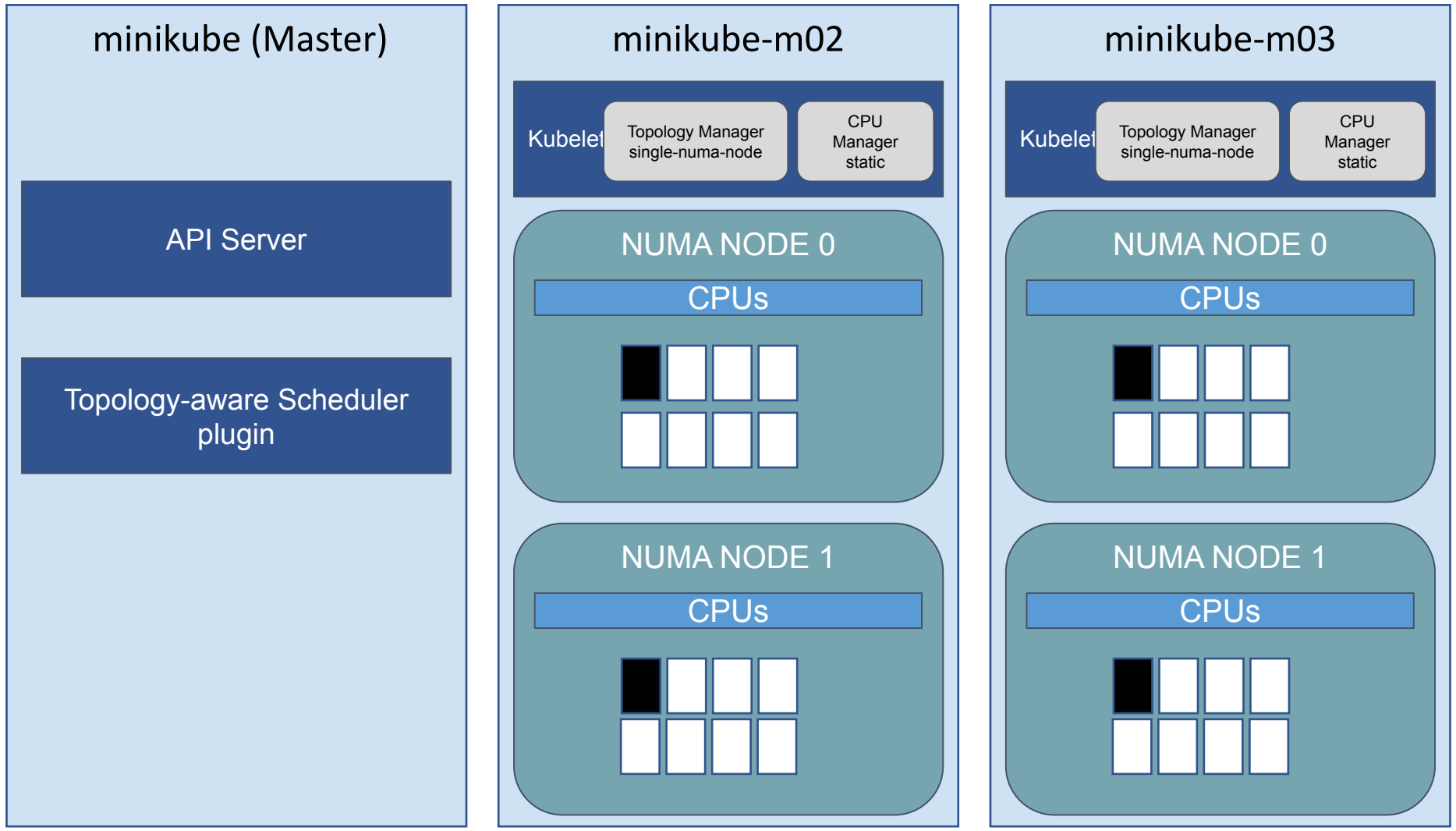


NRT = Node Resource Topology

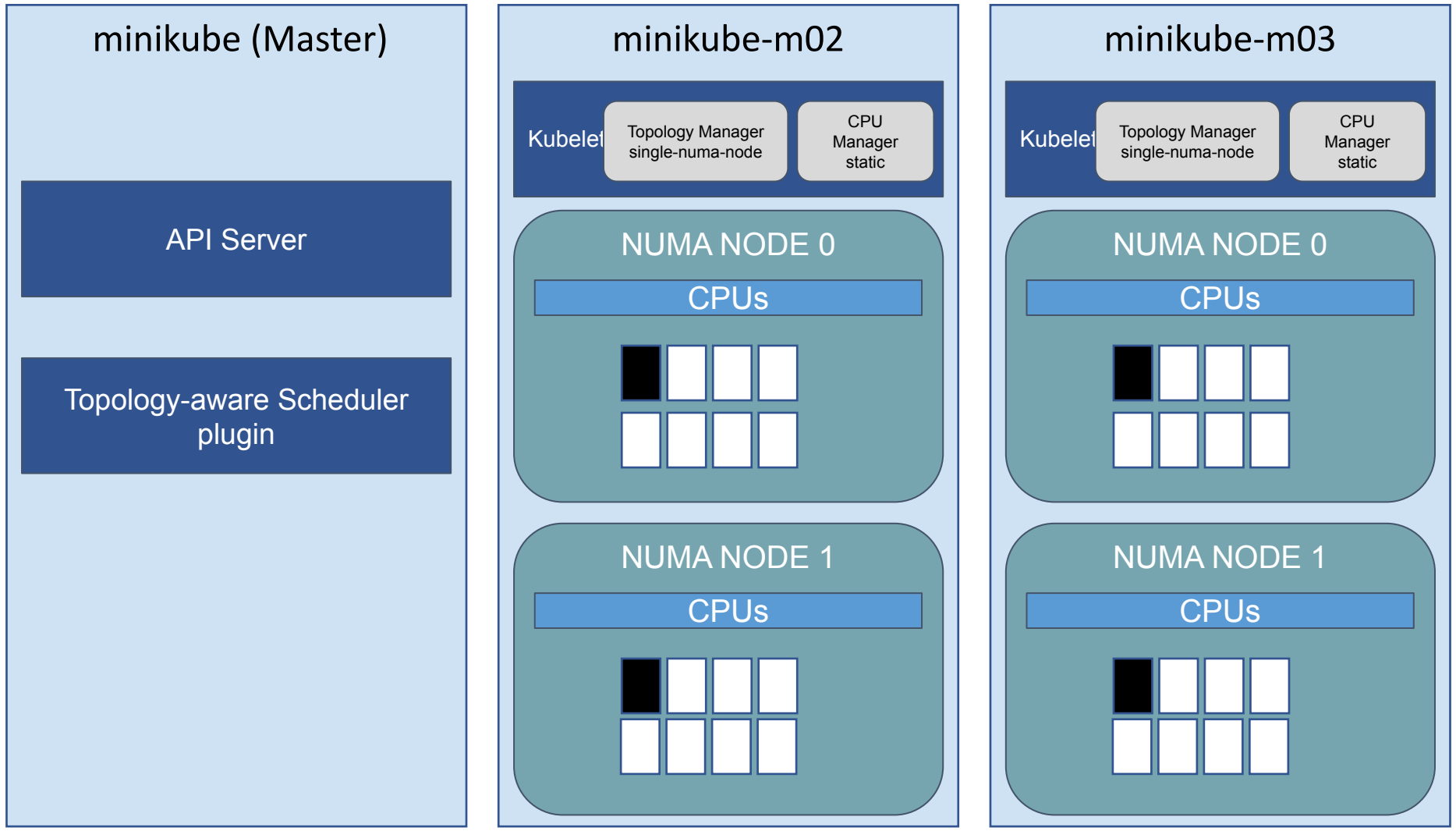
Solution: Local cache with reserve plugin

1. Keep a cache of resources **allocated but not yet reported in NRT objects by the updater** (unreported resources). Cache is updated after a pod is scheduled.
2. Account the unreported resources **against ALL the NUMA zones** on the given node as the scheduler cannot foresee the NUMA node from which resources are allocated.
3. **Invalidate the cache** on each NRT object changes (e.g. updates)

Demo: System setup



Demo: Burst of pods being scheduled

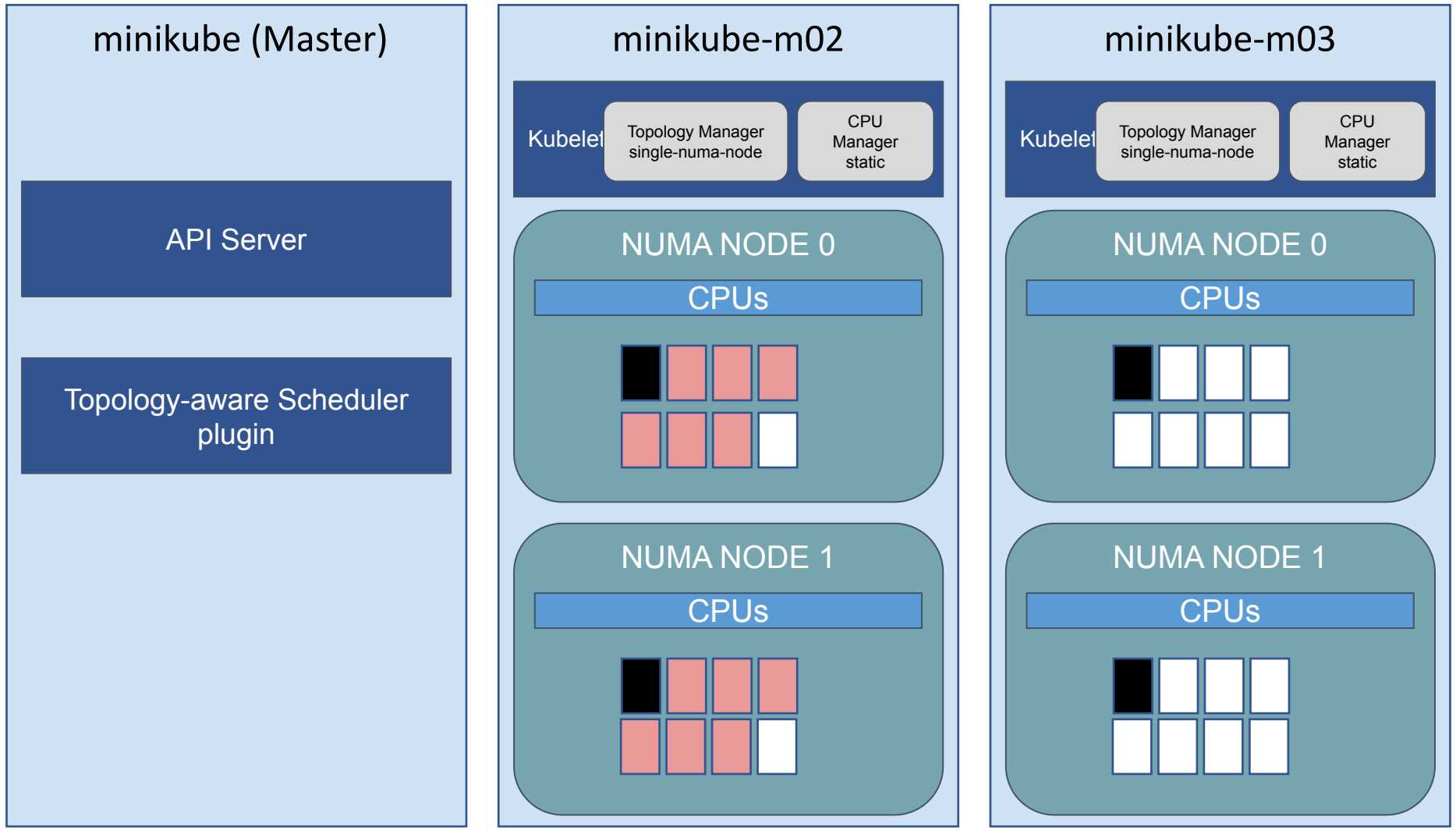



```
kind: Pod
metadata:
  name: testpod1
spec:
  schedulerName: my-scheduler
  containers:
  - name: testcnt1
    resources:
      requests:
        memory: 128Mi
        cpu: 6
      limits:
       memory: 128Mi
        cpu: 6
```

```
kind: Pod
metadata:
  name: testpod2
spec:
  schedulerName: my-scheduler
  containers:
  - name: testcnt2
    resources:
      requests:
        memory: 128Mi
        cpu: 6
      limits:
       memory: 128Mi
        cpu: 6
```

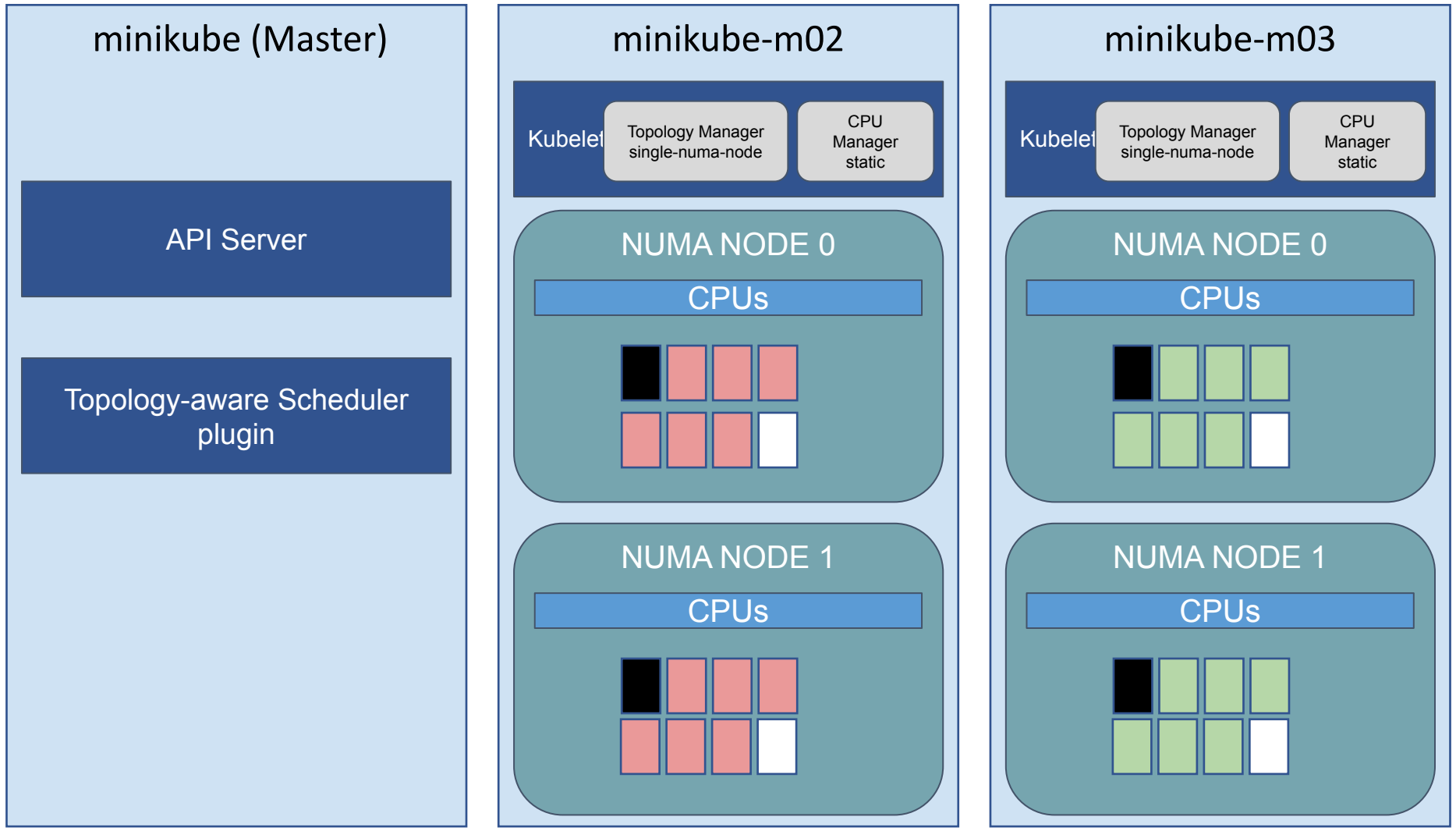
```
kind: Pod
metadata:
  name: testpod3
spec:
  schedulerName: my-scheduler
  containers:
  - name: testcnt3
    resources:
      requests:
        memory: 128Mi
        cpu: 6
      limits:
       memory: 128Mi
        cpu: 6
```

Demo: testPod1 scheduled




```
kind: Pod
metadata:
  name: testpod1
spec:
  schedulerName: my-scheduler
  containers:
  - name: testcnt1
    resources:
      requests:
        memory: 128Mi
        cpu: 6
      limits:
       memory: 128Mi
        cpu: 6
```

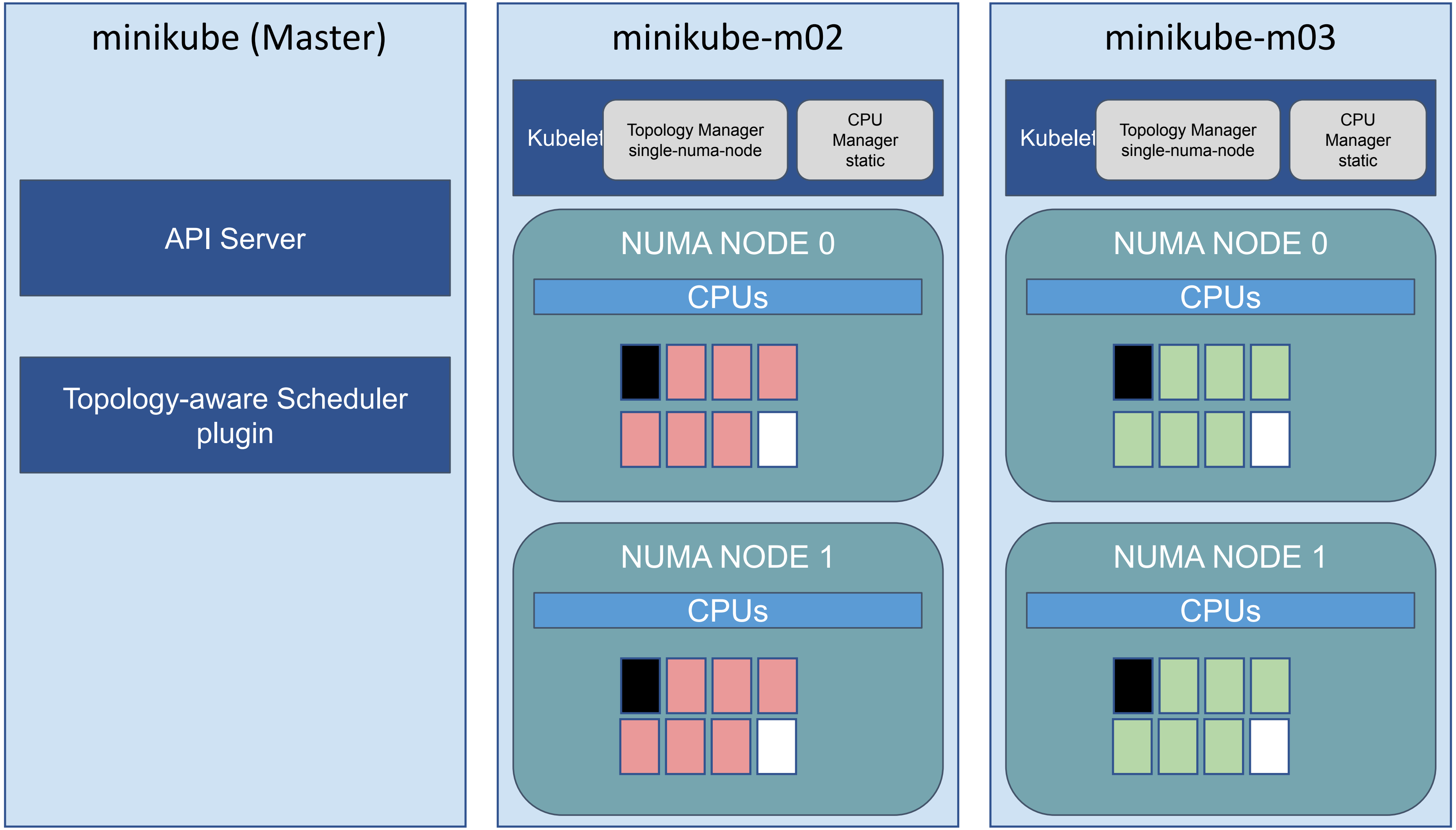
Demo: testPod2 scheduled




```
kind: Pod
metadata:
  name: testpod1
spec:
  schedulerName: my-scheduler
  containers:
  - name: testcnt1
    resources:
      requests:
        memory: 128Mi
        cpu: 6
      limits:
       memory: 128Mi
        cpu: 6
```

```
kind: Pod
metadata:
  name: testpod2
spec:
  schedulerName: my-scheduler
  containers:
  - name: testcnt2
    resources:
      requests:
        memory: 128Mi
        cpu: 6
      limits:
       memory: 128Mi
        cpu: 6
```

Demo: testPod3 pending



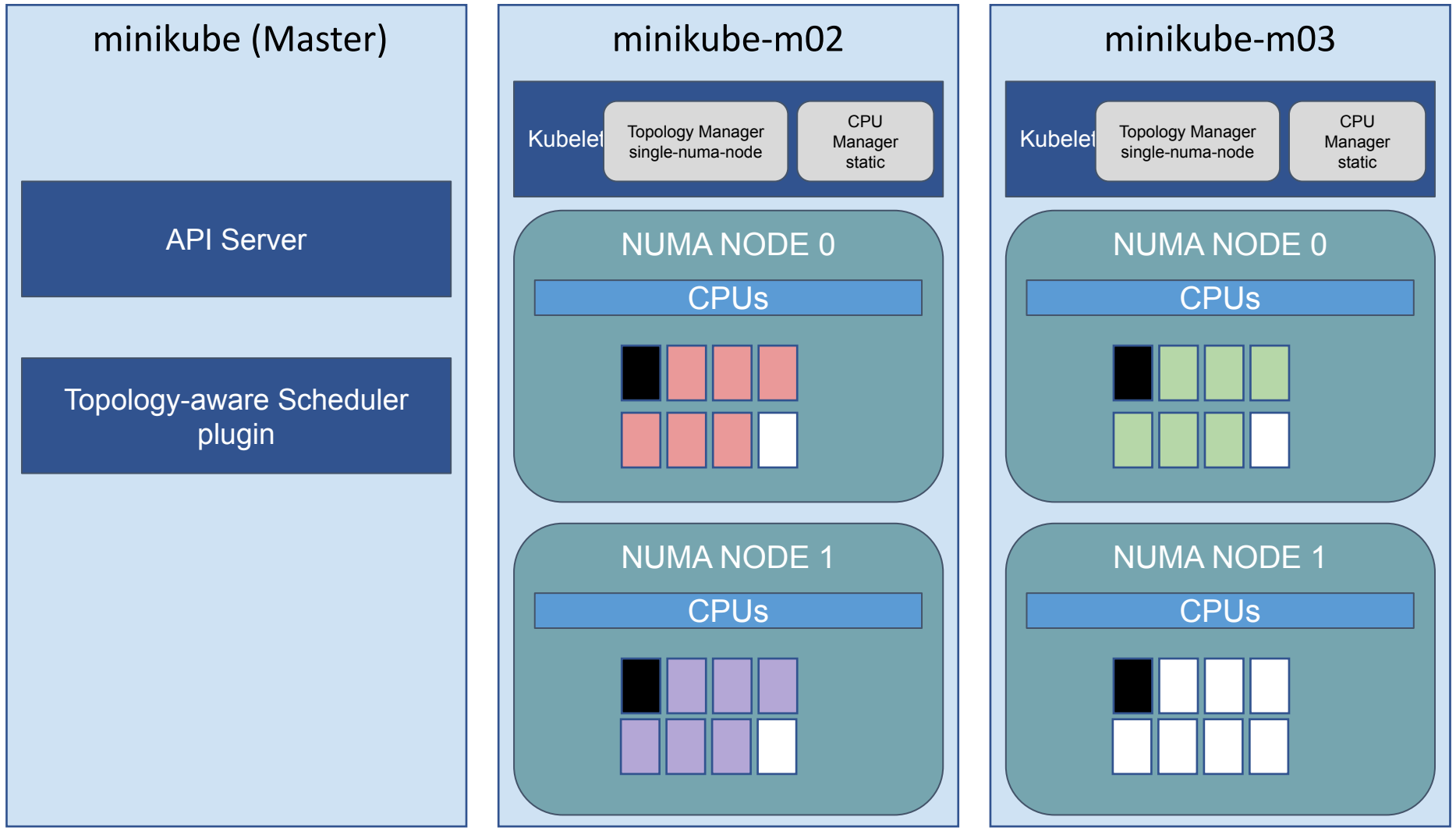

```
kind: Pod
metadata:
  name: testpod1
spec:
  schedulerName: my-scheduler
  containers:
  - name: testcnt1
    resources:
      requests:
        memory: 128Mi
        cpu: 6
      limits:
       memory: 128Mi
        cpu: 6
```

```
kind: Pod
metadata:
  name: testpod2
spec:
  schedulerName: my-scheduler
  containers:
  - name: testcnt2
    resources:
      requests:
        memory: 128Mi
        cpu: 6
      limits:
       memory: 128Mi
        cpu: 6
```

```
kind: Pod
metadata:
  name: testpod3
spec:
  schedulerName: […]ler
  containers:
  - name: te[…]
    resou[…]
        […]y: 128Mi
        […]: 6
      […]mits:
       memory: 128Mi
        cpu: 6
```

PENDING

Demo: After reconciliation




```
kind: Pod
metadata:
  name: testpod1
spec:
  schedulerName: my-scheduler
  containers:
  - name: testcnt1
    resources:
      requests:
        memory: 128Mi
        cpu: 6
      limits:
       memory: 128Mi
        cpu: 6
```

```
kind: Pod
metadata:
  name: testpod2
spec:
  schedulerName: my-scheduler
  containers:
  - name: testcnt2
    resources:
      requests:
        memory: 128Mi
        cpu: 6
      limits:
       memory: 128Mi
        cpu: 6
```

```
kind: Pod
metadata:
  name: testpod3
spec:
  schedulerName: my-scheduler
  containers:
  - name: testcnt3
    resources:
      requests:
        memory: 128Mi
        cpu: 6
      limits:
       memory: 128Mi
        cpu: 6
```

Demo

Recording: here

Scheduler Plugin: phase 2 developments

1. Local cache enablement with Reserve plugin (PR#315, PR#437, PR#439)
    a. Different caching strategies: overReserve and DiscardReservedNodes (PR#558)
2. NUMA aware scoring: LeastNUMANodes scoring strategy (KEP, PR)
3. noderesourcetopology: overhaul Topology Manager configuration management #488

NodeResourceTopology API: Phase 2

CRD based API definition used for enabling Topology-aware Scheduling and other use cases!

https://github.com/k8stopologyawareschedwg/noderesourcetopology-api

New v1alpha2 version of the API (PR #25) by incorporating feedback received on v1alpha1:

i. add top-level Attributes

ii. deprecate the TopologyPolicies field in the favour of attributes

Next Step: Gather more feedback and defer extensive cleanups and changes to v1beta1

Node Feature Discovery: Phase 2

NFD and RTE are topology updater agents and work is being done to have feature parity between these software components.

1. Catch up with RTE features (introduce exclude list: PR#949, reactive updates: PR#1031)
2. Enable reserve Plugin Support (PR#1049)
3. Consume NRT API v1alpha2 (update to v1alpha2: PR#1053, policy and scope as attributes: PR#1054)

Next Step: Continue enhancing NFD to make it the primary reference implementation as a topology updater agent.

Topology-aware Scheduling Future plans

1. Scalability Testing
2. Integration with DRA and VPA
3. (more long term) Support in core k8s - also shaped by DRA/VPA integration.

For TAS work: Special Shoutout! 🎉

- Francesco Romani
- Markus Lehtonen
- Piotr Prokop
- Talor Itzhak
- Wei Huang

WG Batch: Getting involved

- ➔ slack.k8s.io #wg-batch
- ➔ wg-batch@k8s.io
- ➔ issues.k8s.io?q=label:wg/batch
- ➔ Participate in WG Batch
  git.k8s.io/community/wg-batch
  - ◆ Meetings every other Thursday at 2PM UTC

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