Let There Be Topology-Awareness in Kube-Scheduler!!

Swati Sehgal

Senior Software Engineer

KNI Telco-5G Compute team

Email: swsehgal@redhat.com

Slack: swsehgal

Github: swatisehgal

Red Hat

Today's Agenda

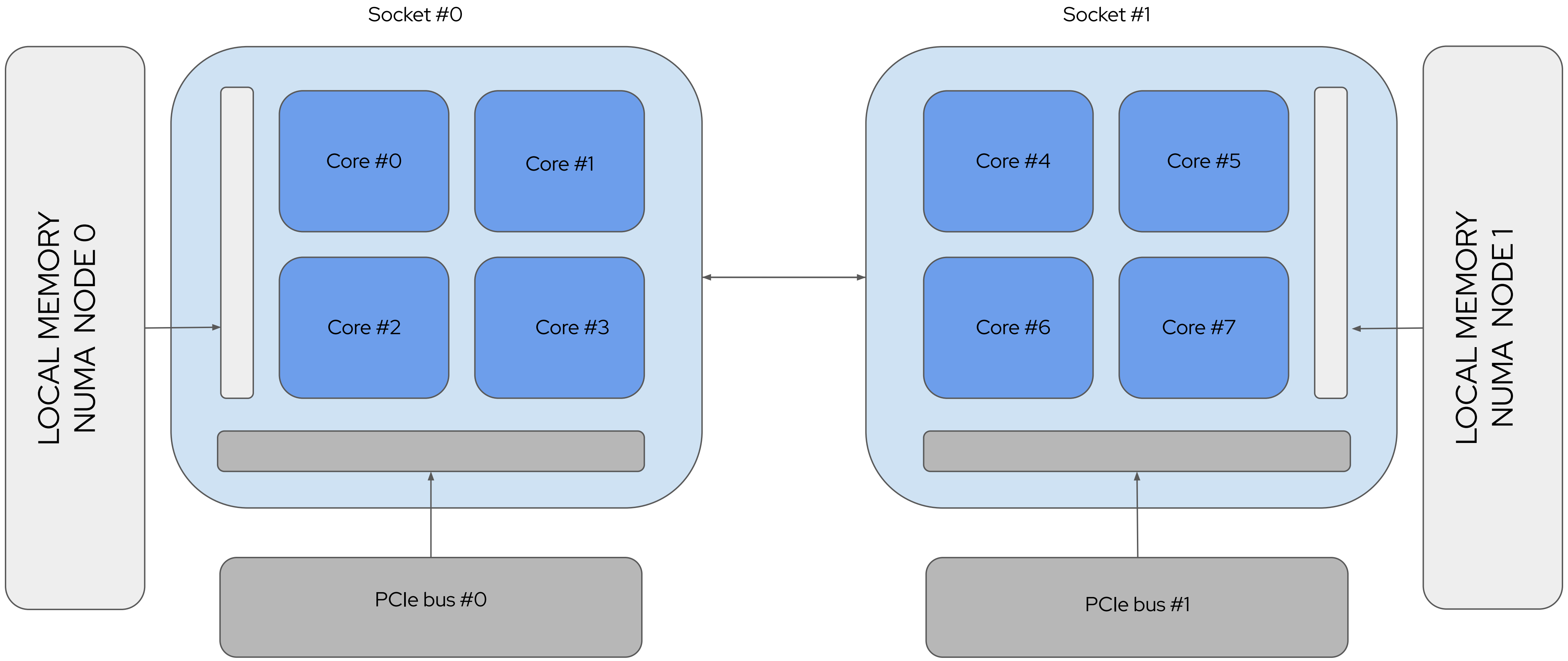
Red Hat

Today's Agenda

1. **Hardware Topology**
2. Topology Alignment in Kubernetes
3. Topology-unawareness of Kubernetes Default Scheduler
4. How the Kubernetes Default Scheduler works
5. Topology aware scheduling - proposed Solution
6. Use cases

Red Hat

Hardware Topology: What is NUMA ?



Why is NUMA Alignment needed?

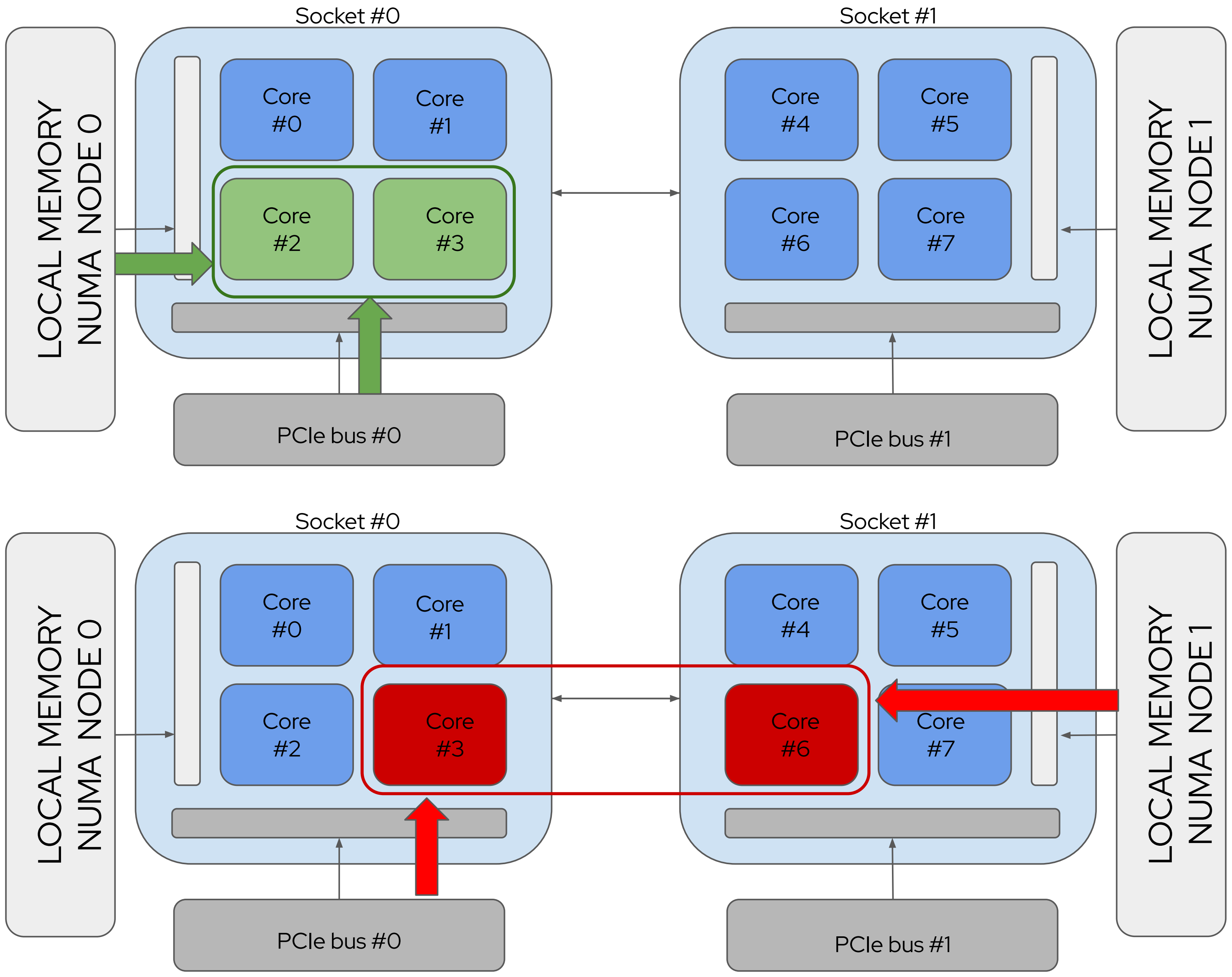
NUMA alignment of CPUs and devices allows workloads to run in an environment optimized for low-latency.

- Application in the field of Telco 5G, ML, AI and data analytics require NUMA alignment
- DPDK based networking applications require resources from the same NUMA node for optimal performance

Today's Agenda

Red Hat

NUMA Alignment in Kubernetes



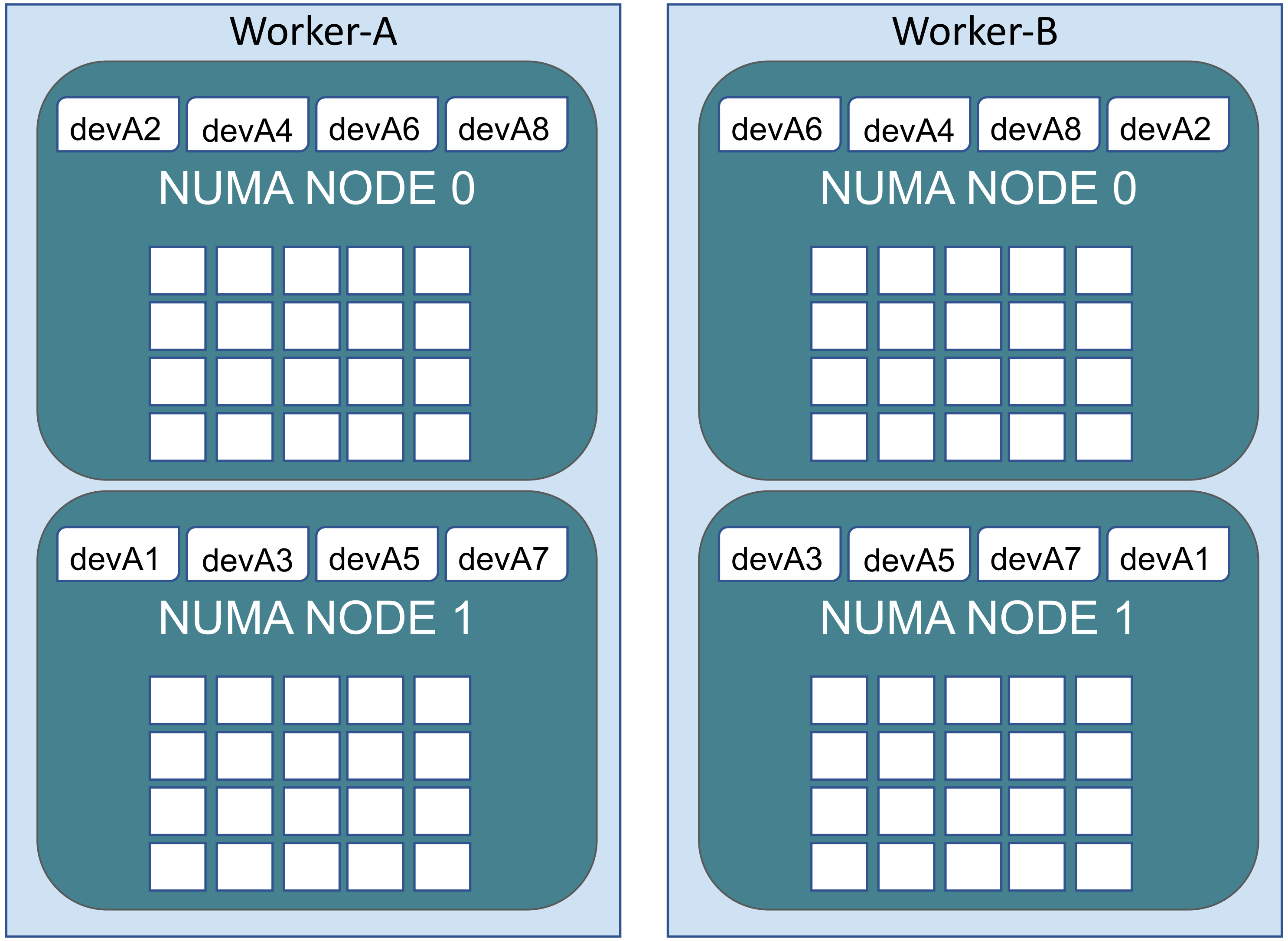
```
kind: Pod
metadata:
  name: my-pod
spec:
  containers:
  - name: my-container
    resources:
      requests:
        example.com/deviceA: 1
        memory: "64Mi"
        cpu: 2
      limits:
       example.com/deviceA: 1
       memory: "64Mi"
       cpu: 2
```

Red Hat

Today's Agenda

Red Hat

Topology-unawareness of Kubernetes Default Scheduler

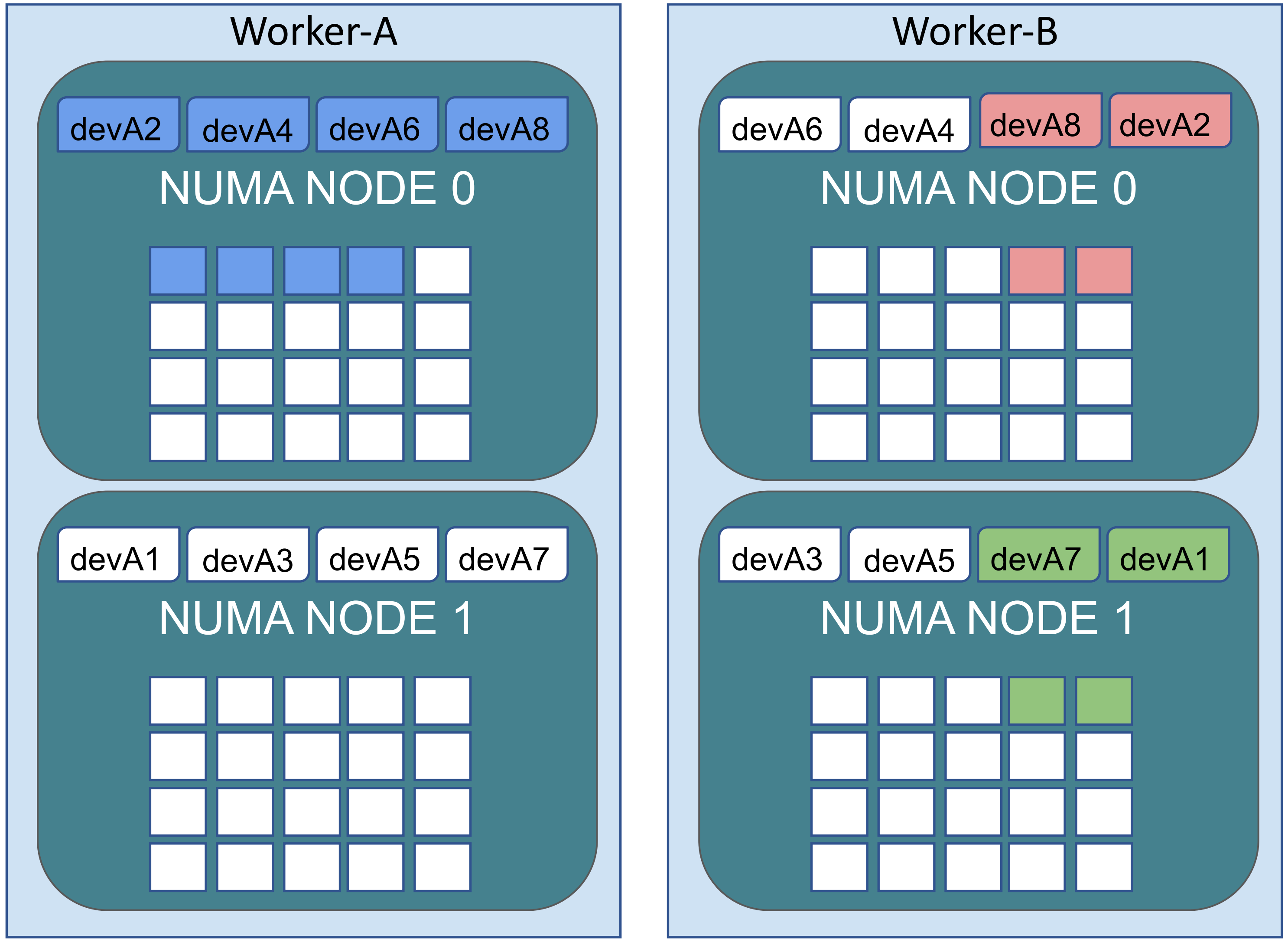


Kube-Scheduler's view

	Device A	CPU
Worker-A	8	40
Worker-B	8	40

* Environment configured with Topology Manager single-numa-node policy

Topology-unawareness of Kubernetes Default Scheduler



Kube-Scheduler's view

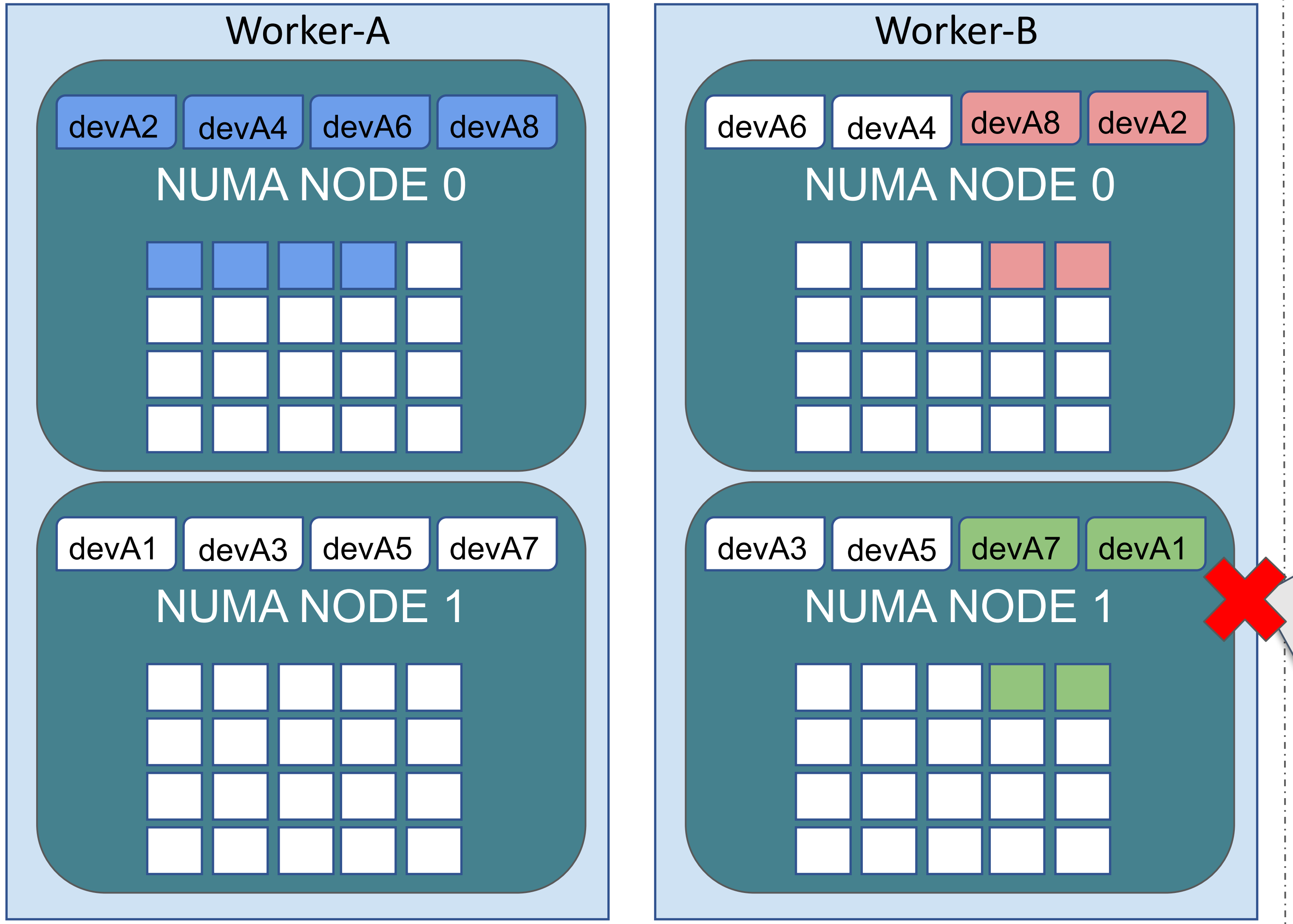
	Device A	CPU
Worker-A	**4**	**36**
Worker-B	**4**	**36**

```
resources:
  requests:
     deviceA: 2
     Cpu: 2
     memory: 200Mi
  limits:
     deviceA: 2
     cpu: 2
     memory: 200Mi
```

```
resources:
  requests:
     deviceA: 2
     cpu: 2
     memory: 200Mi
  limits:
     deviceA: 2
     cpu: 2
     memory: 200Mi
```

```
resources:
  requests:
     deviceA: 5
     cpu: 4
     memory: 200Mi
  limits:
     deviceA: 4
     cpu: 4
     memory: 200Mi
```

Topology-unawareness of Kubernetes Default Scheduler



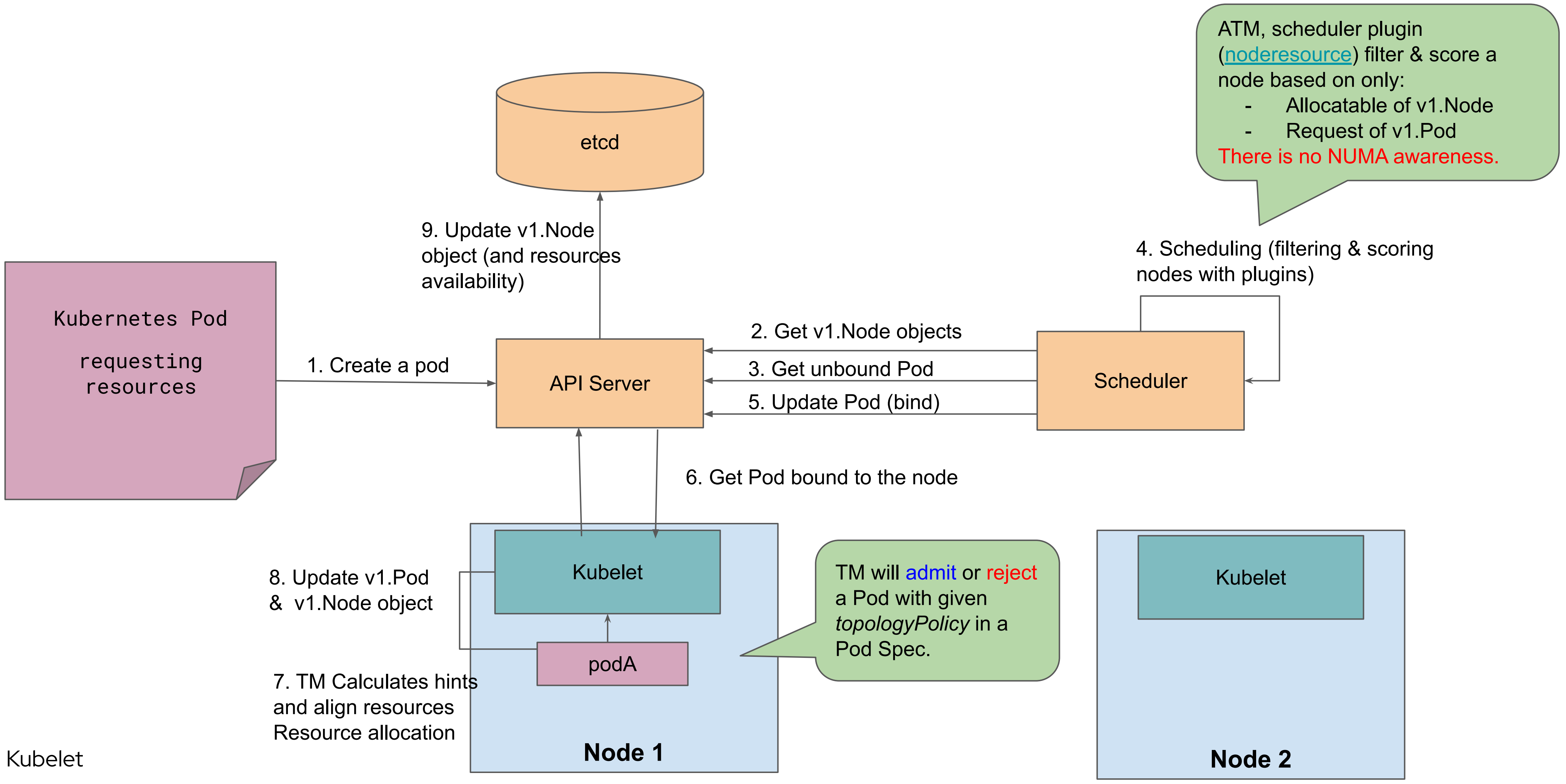
Kube-Scheduler's view

	Device A	CPU
Worker-A	**4**	**36**
Worker-B	**4**	**36**

```
kind: Pod
metadata:
  name: pod-with-my-scheduler
   c:
  schedulerName: my-scheduler
  containers:
  - name: pod-with-my-scheduler
    resources:
      requests:
        example.com/deviceA: 4
        cpu: 4
      limits:
       example.com/deviceA: 4
        cpu: 4
```

Today's Agenda

How the Kubernetes Default Scheduler works


ATM, scheduler plugin (noderesource) filter & score a node based on only:
- Allocatable of v1.Node
- Request of v1.Pod
There is no NUMA awareness.
etcd
9. Update v1.Node object (and resources availability)
4. Scheduling (filtering & scoring nodes with plugins)
Kubernetes Pod requesting resources
1. Create a pod
API Server
2. Get v1.Node objects
3. Get unbound Pod
5. Update Pod (bind)
Scheduler
6. Get Pod bound to the node
8. Update v1.Pod & v1.Node object
Kubelet
podA
7. TM Calculates hints and align resources Resource allocation
Node 1
TM will admit or reject a Pod with given topologyPolicy in a Pod Spec.
Kubelet
Node 2

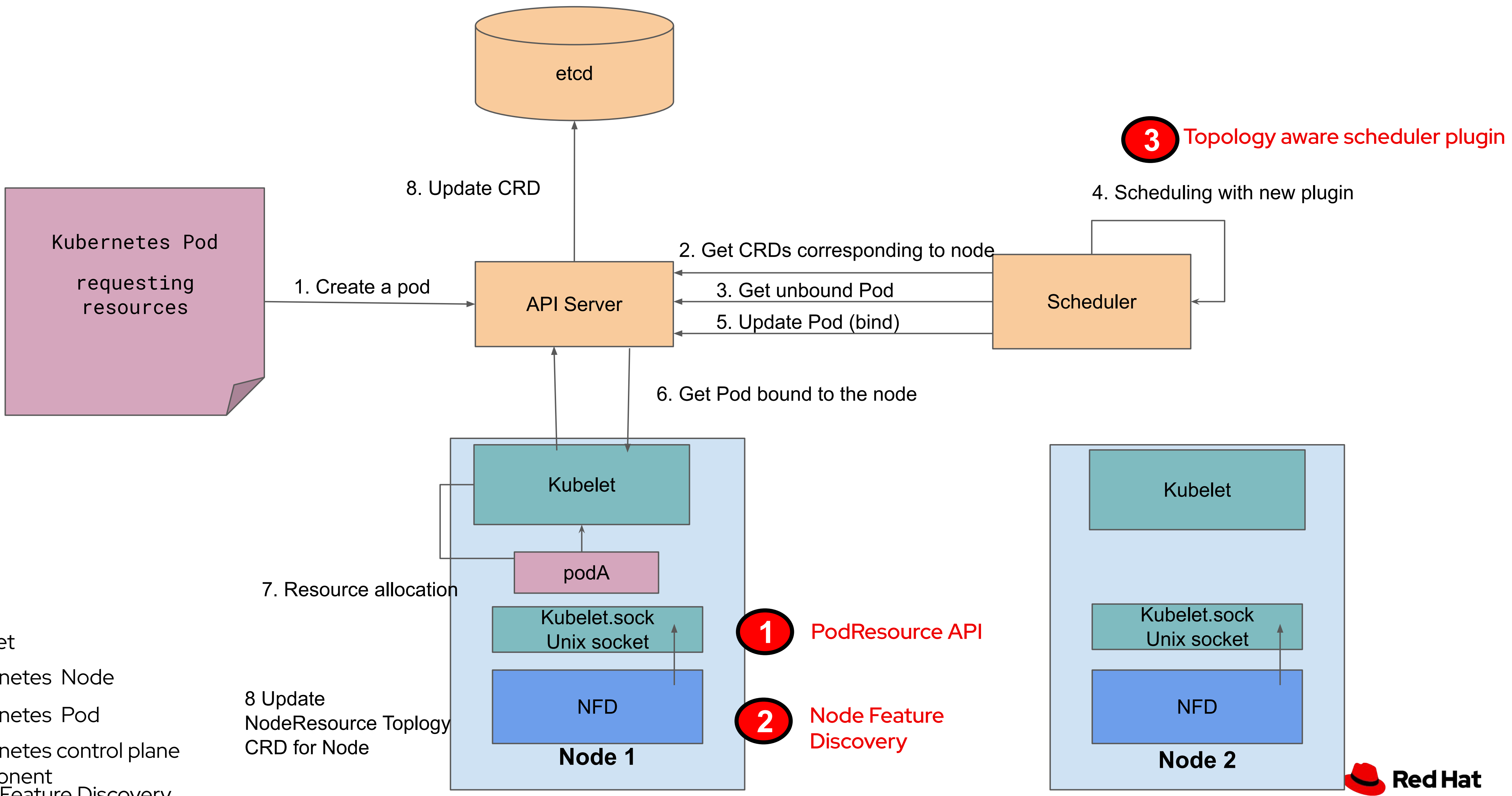
● Kubelet
● Kubernetes Node
● Kubernetes Pod
● Kubernetes control plane component

Today's Agenda

Key components of Topology aware scheduling

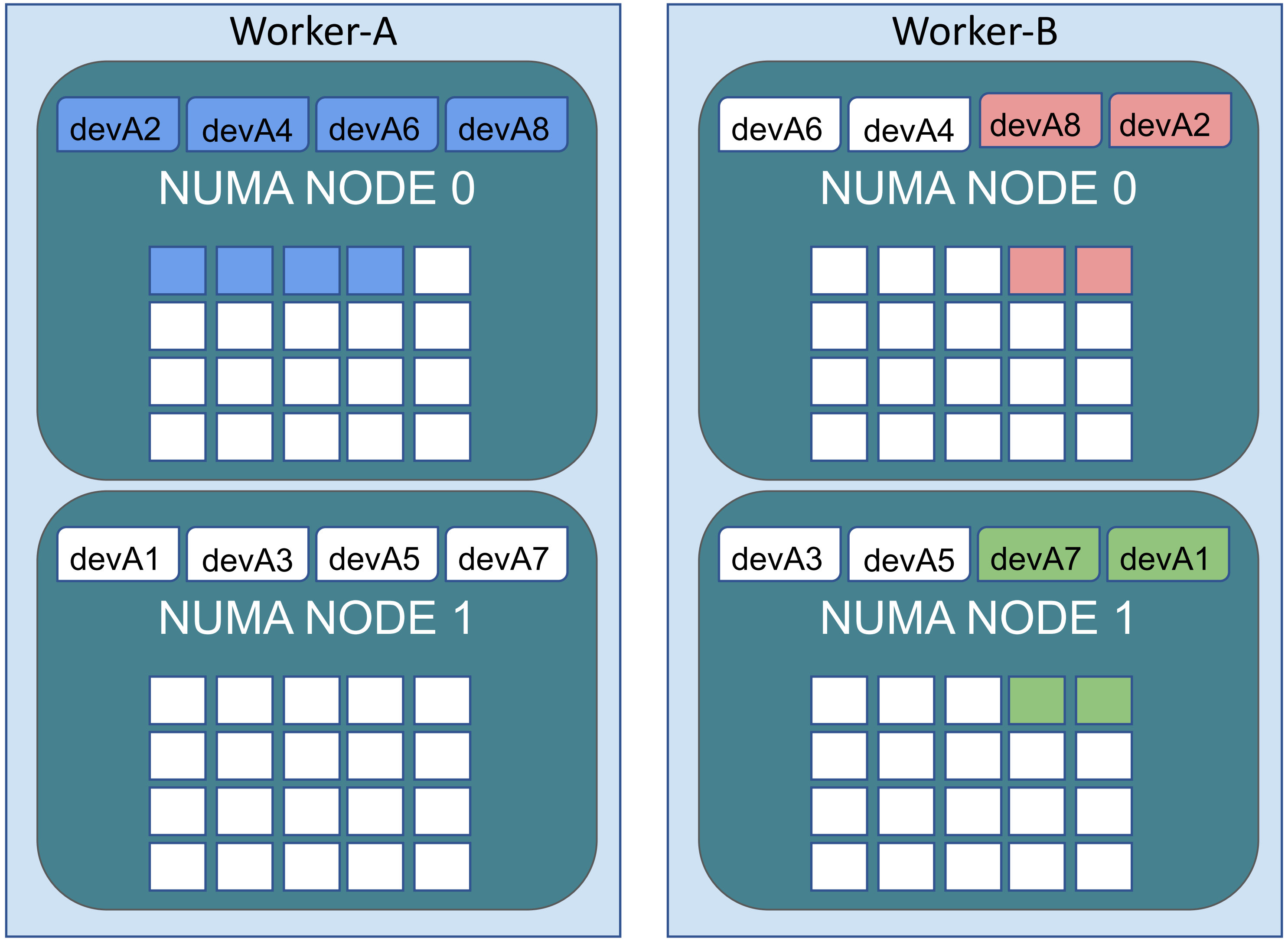
1. PodResource API
2. Node Feature Discovery
3. Topology aware scheduler plugin
4. NodeResourceTopology API

End to end Proposed Solution: Topology Aware Scheduling



●	Kubelet
●	Kubernetes Node
●	Kubernetes Pod
●	Kubernetes control plane component
●	Node Feature Discovery

Topology-unawareness of Kubernetes Default Scheduler



Kube-Scheduler's view

	Device A	CPU
Worker-A	4	36
Worker-B	4	36

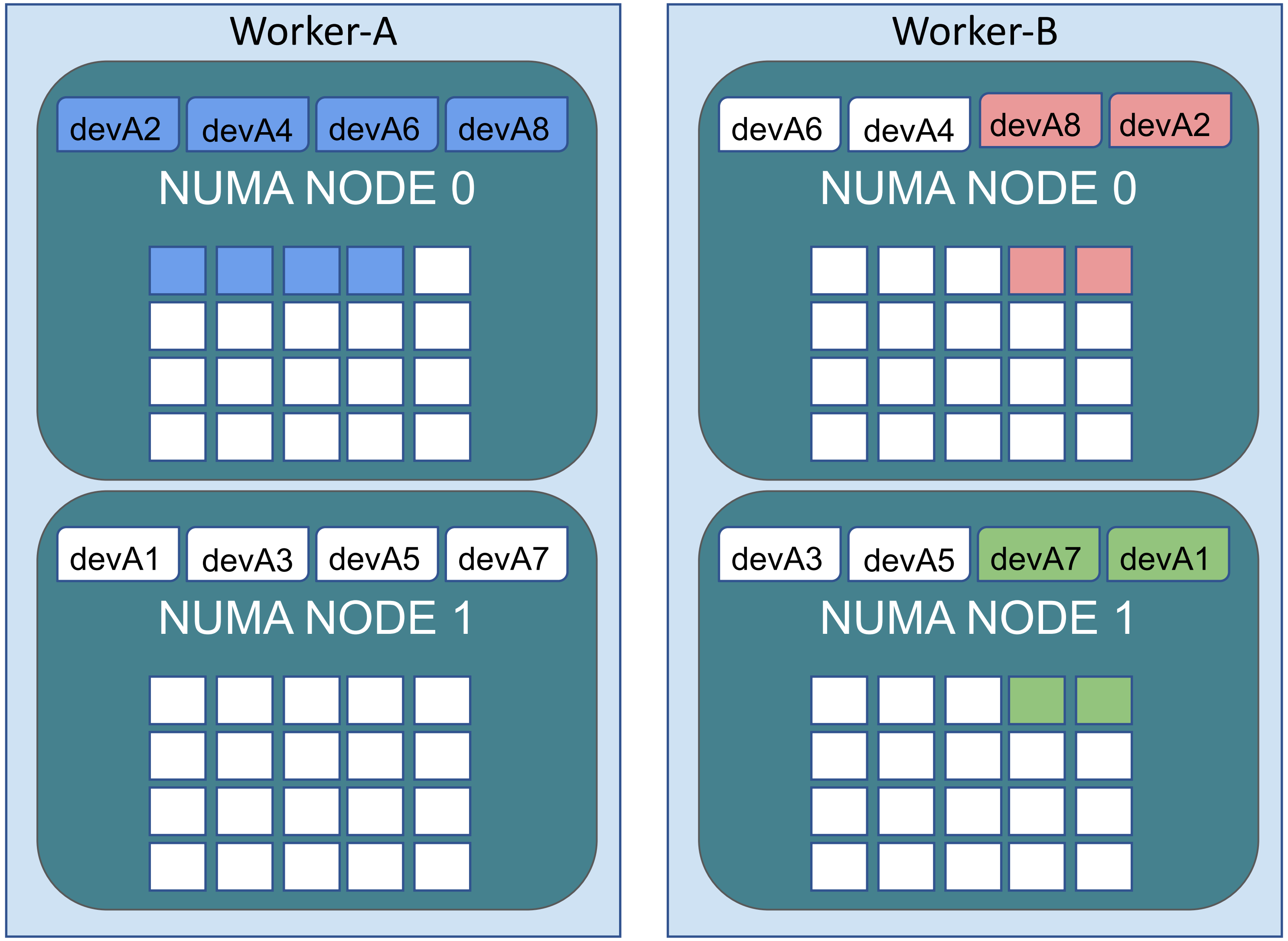
```
resources:
  requests:
    deviceA: 2
    Cpu: 2
    memory: 200Mi
  limits:
    deviceA: 2
    cpu: 2
    memory: 200Mi
```

```
resources:
  requests:
    deviceA: 2
    cpu: 2
    memory: 200Mi
  limits:
    deviceA: 2
    cpu: 2
    memory: 200Mi
```

```
resources:
  requests:
    deviceA: 5
    cpu: 4
    memory: 200Mi
  limits:
    deviceA: 4
    cpu: 4
    memory: 200Mi
```

Red Hat

Topology-aware Scheduling in Kubernetes



Topology-aware Scheduler's view

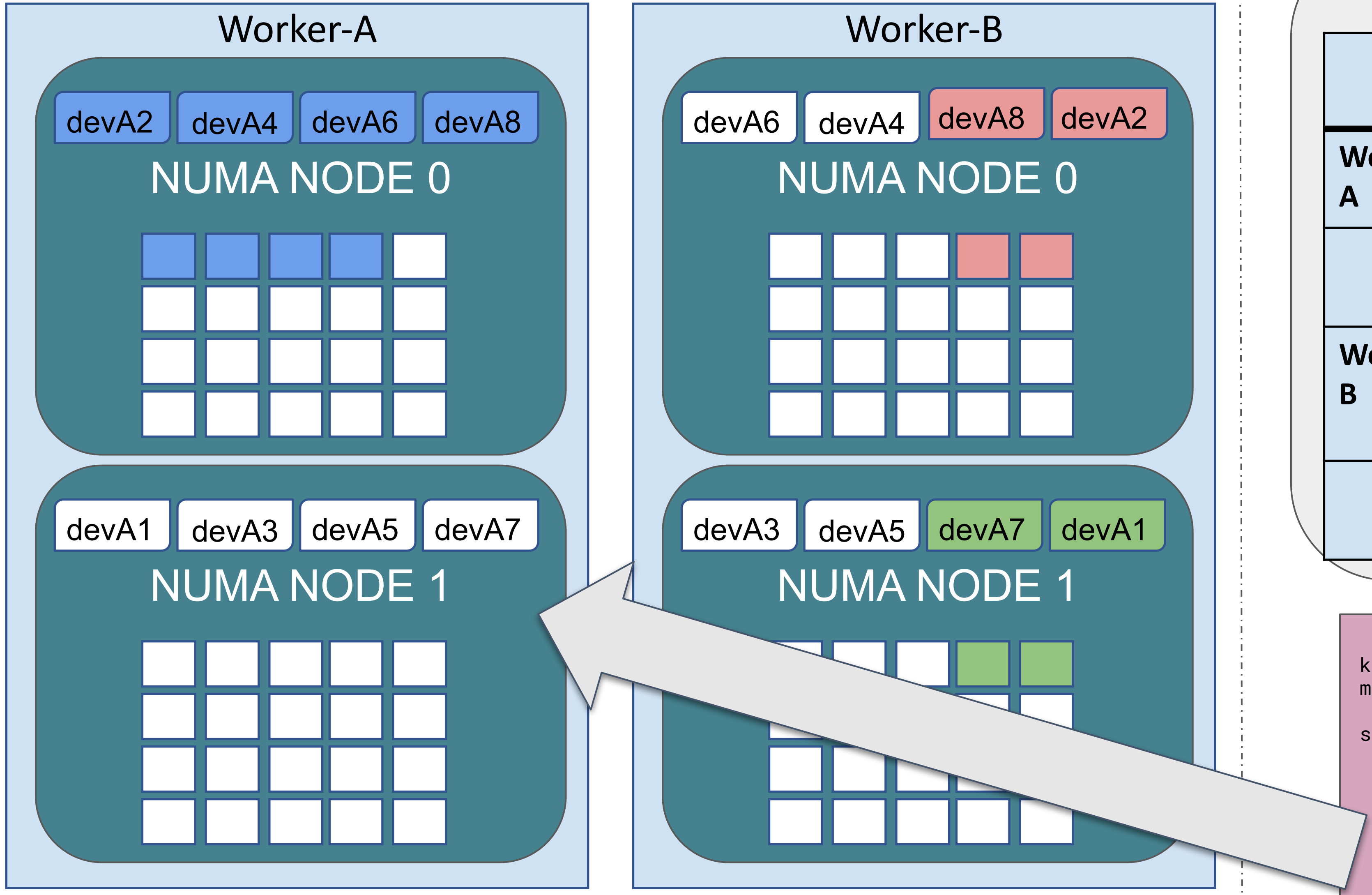
		Device A	CPU
Worker-A	Numa Node 0	0	16
	Numa Node 1	4	20
Worker-B	Numa Node 0	2	18
	Numa Node 1	2	18

```
resources:
  requests:
     deviceA: 2
     Cpu: 2
     memory: 200Mi
  limits:
     deviceA: 2
     cpu: 2
     memory: 200Mi
```

```
resources:
  requests:
     deviceA: 2
     cpu: 2
     memory: 200Mi
  limits:
     deviceA: 2
     cpu: 2
     memory: 200Mi
```

```
resources:
  requests:
     deviceA: 5
     cpu: 4
     memory: 200Mi
  limits:
     deviceA: 4
     cpu: 4
     memory: 200Mi
```

Topology-aware Scheduling in Kubernetes



Topology-aware Scheduler's view

		Device A	CPU
Worker-A	**Numa Node 0**	**0**	**16**
	Numa Node 1	**4**	**20**
Worker-B	**Numa Node 0**	**2**	**18**
	Numa Node 1	**2**	**18**

```
kind: Pod
metadata:
  name: pod-with-my-scheduler
spec:
  schedulerName: my-scheduler
  containers:
  - name: pod-with-my-scheduler
    resources:
      requests:
        example.com/deviceA: 4
        cpu: 4
      limits:
       example.com/deviceA: 4
        cpu: 4
```

Current Status

1. **PodResource API**
   - Proposed Pod Resource API enhancements (KEP) Merged ✓
   - Introduce device topology and cpuids info as part of Podresource API (PR) Merged ✓
   - GetAllocatableResources() support being targeted for K8s 1.21 release 🕒
2. **Node Feature Discovery**
   - Resource Topology Enablement (RTE) (KEP, code) ✓
   - Initial Discussions with NFD maintainers and stakeholders (Issue, Proposal Doc) ✓
   - NFD development work in progress here 🕒
   - Initial Demo: here ✓
3. **Topology aware scheduler plugin** (KEP, code) 🕒
4. **NodeResourceTopology API** (PR) – still being discussed in the community 🕒

Red Hat

Today's Agenda

For more detail: Use case Doc.

vRAN User plane (Nokia)

- requirement is to process packets with extreme high bandwidth. To achieve this we use DPDK with SR-IOV.
- The pods handling the user traffic require the SR-IOV VF, the memory huge pages and the CPU resources from the same NUMA node.

Performance-intensive high throughput network applications (Samsung)

- strict requirements must be met (of speed, latency, availability) for containerized 5G deployments
- MEC (Multi-access Edge Computing) - to deliver low latency access to edge applications

Cloudnative Network Function cluster level NUMA alignments (Intel)

Two workload numa alignments requirements:

- Full: CPU / HugePage / SRIOV VF have to be in the same NUMA Zone
- Native: CPU / HugePages must be aligned. SRIOV VFs can come from either of the NUMA zones.

GPUDirect scheduling (ViaSat)

- GPU, NIC, and CPU must be in the same NUMA zone.
- The GPU and NIC must be sharing a single-hop on the PCI bus

- Learn more: here
- Use Cases: here
- K8s slack: https://kubernetes.slack.com/archives/C012XSGFZQE
- Email: swsehgal@redhat.com, Slack: swsehgal

Demos

- Topology aware scheduling with Resource Topology Exporter: Demo
- NFD exposing hardware topology through CRDs: Demo

Thank you

Red Hat is the world's leading provider of enterprise open source software solutions. Award-winning support, training, and consulting services make Red Hat a trusted adviser to the Fortune 500.

linkedin.com/company/red-hat

youtube.com/user/RedHatVideos

facebook.com/redhatinc

twitter.com/RedHat