KubeCon | CloudNativeCon

North America 2020

Virtual

Optimizing Storage Assignment via Pod Scheduling Under Disturbance Factors

Kenji Morimoto, Cybozu

About Me

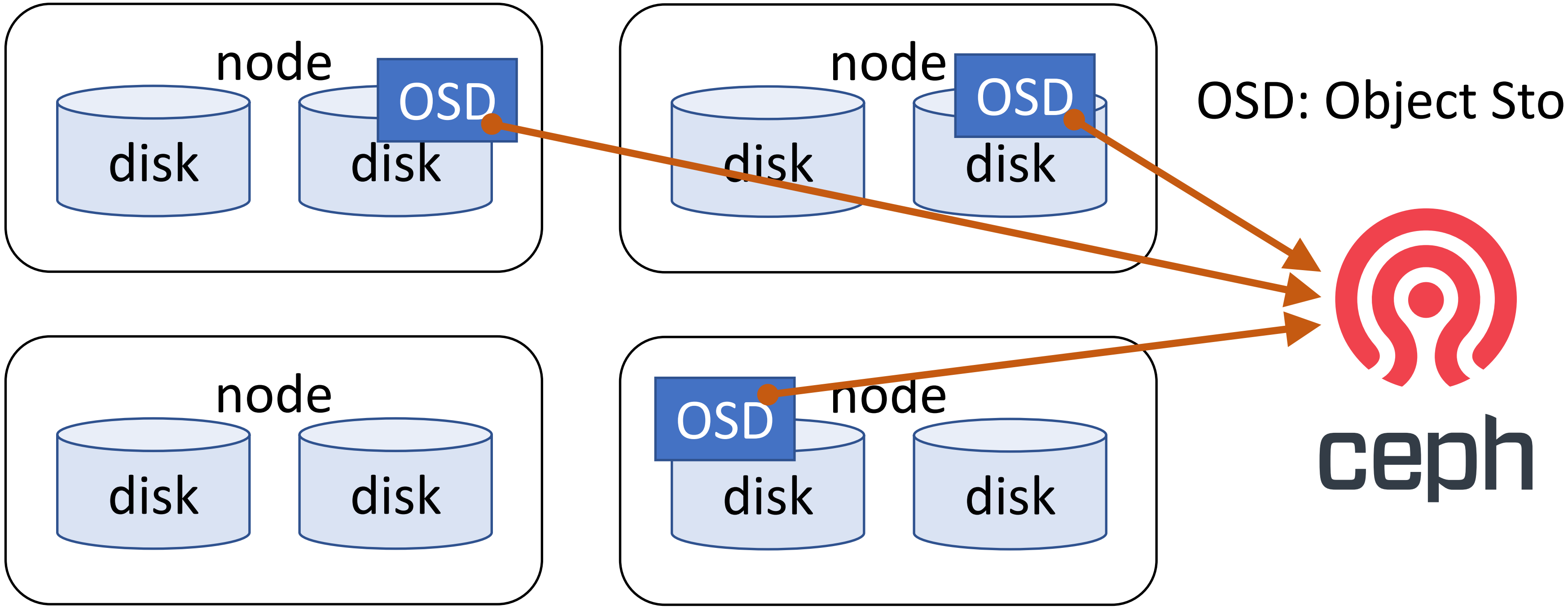
- Kenji Morimoto
  - github.com: morimoto-cybozu
- Worked as an infrastructure engineer for 8 years
  - Running 2,000+ servers on-premise
- Renewing the infrastructure with K8s

Agenda

- Challenge: Running distributed storage system on K8s
  - Recap: Volume management in K8s
  - Problem: How to place storage devices optimally
- Basic idea: WaitForFirstConsumer + Pod topology spread constraints
- Implementation using whenUnsatisfiable stanza
- Tuning: kube-scheduler configuration for the optimal placement
- Demo

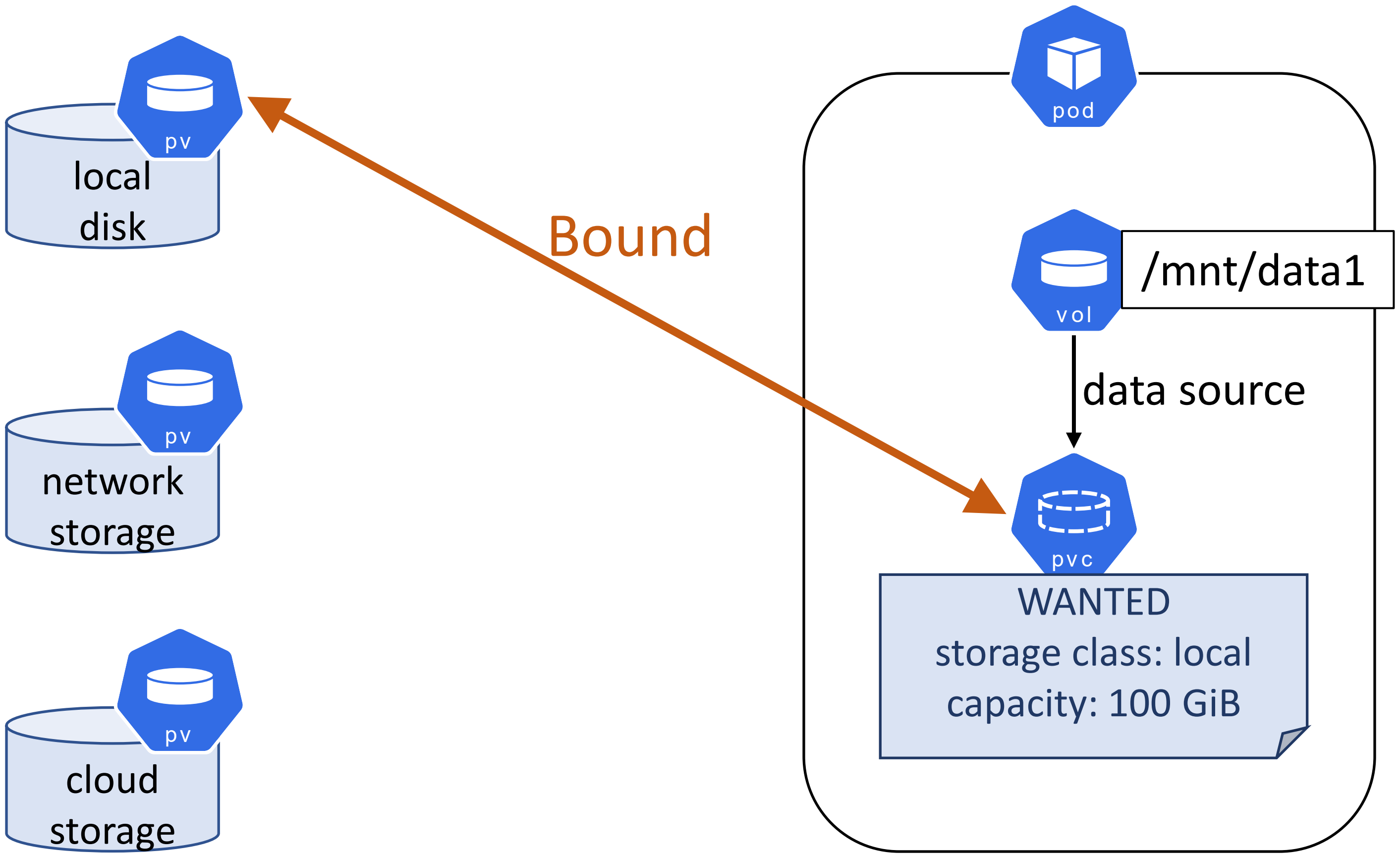
Distributed storage

- Distributed storage system organizes node-local storage devices
- It’s tedious work to add/remove storage devices manually



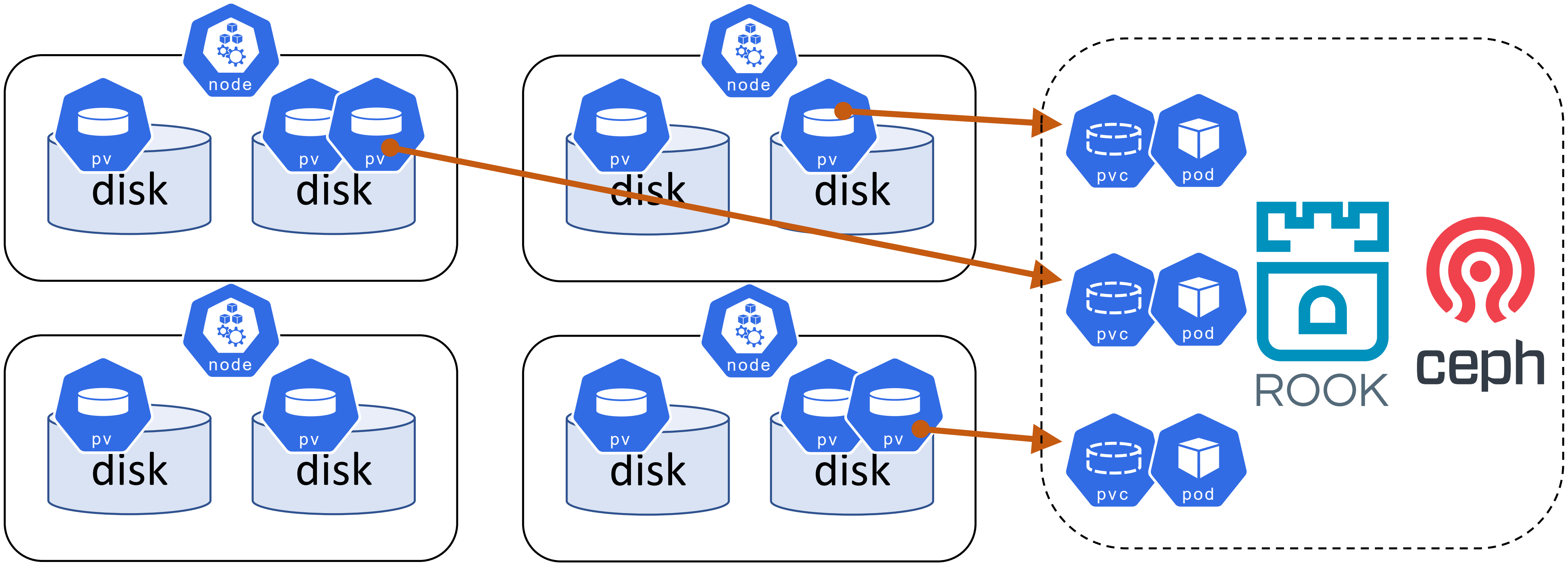
OSD: Object Storage Device

Recap: PVs and PVCs



How Rook places local storage

- Rook has a mode to acquire PVs through PVCs

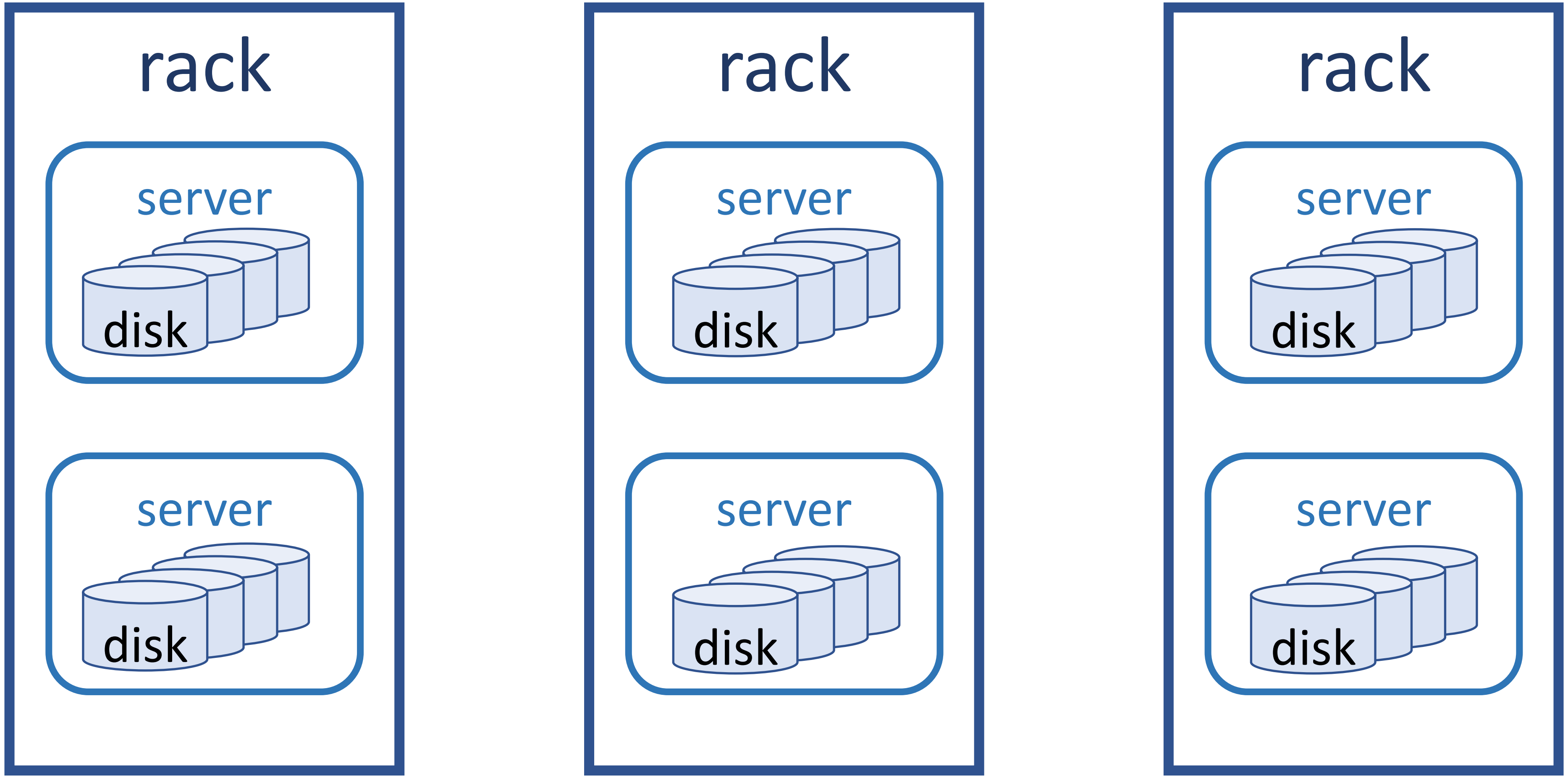


Challenge

- **There is no standard profile** to deploy distributed storage systems on K8s yet
- Distributed storage systems are responsible for replicating data across failure domains for robustness
- Distributing local storage devices evenly is **up to the administrators**

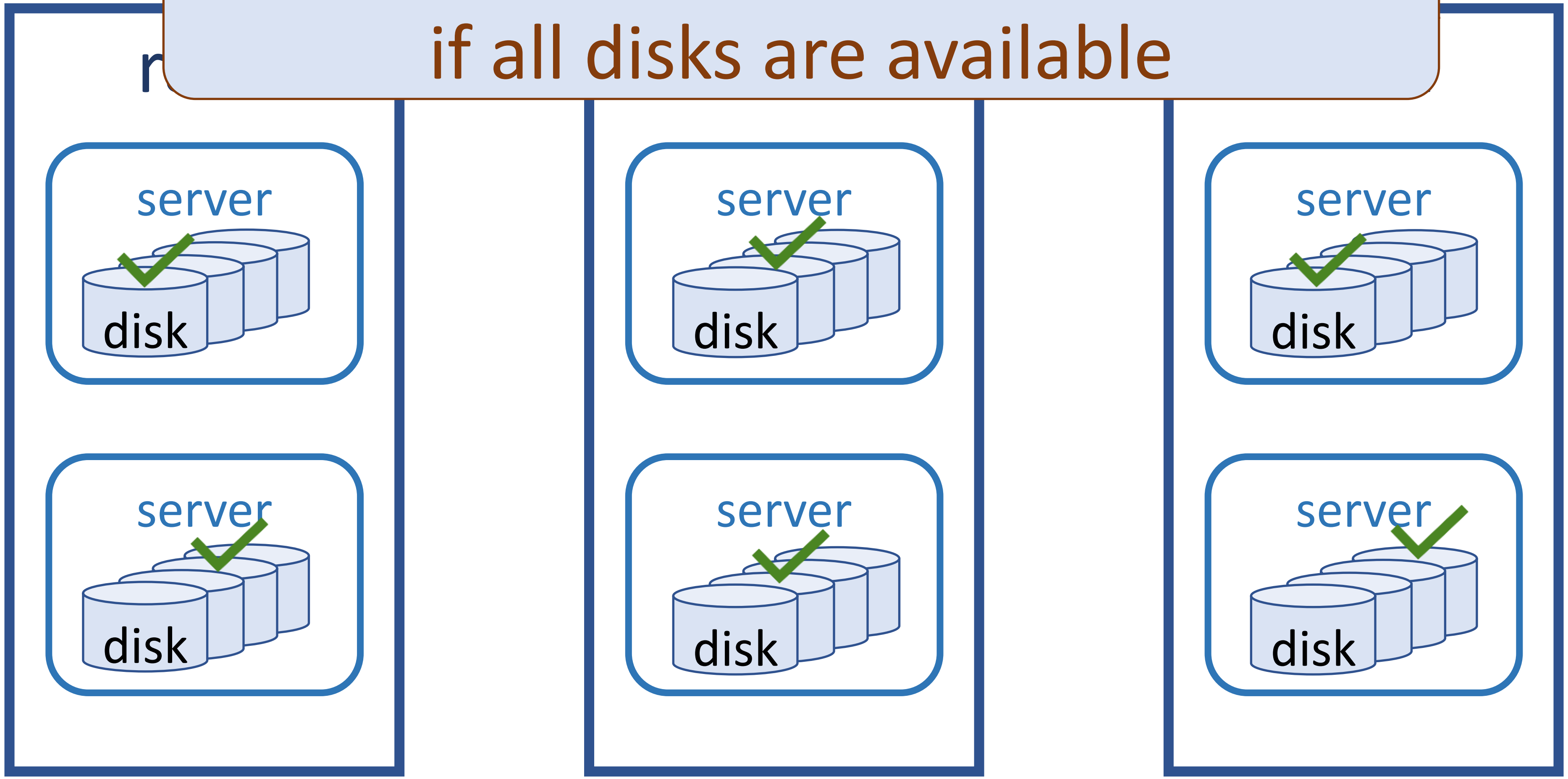
- **Challenge: Distributing PVs for local disks evenly through PVCs**

Uneven local storage availability



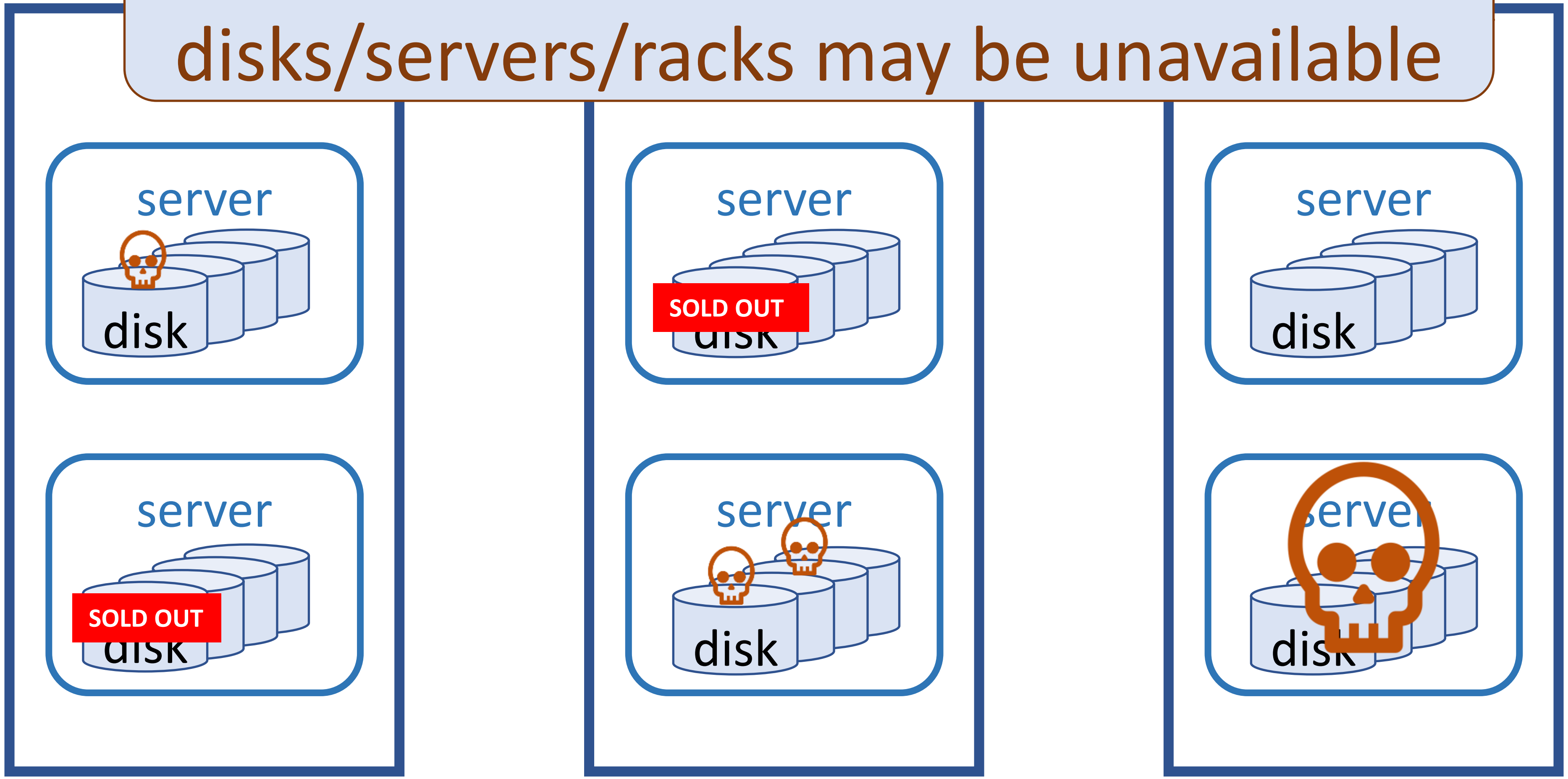
Uneven local storage availability

It’s easy to achieve even distribution if all disks are available



Uneven local storage availability

It’s not easy in reality because some disks/servers/racks may be unavailable



Problem

- **K8s does not care about storage assignment**
  - kube-scheduler handles Pod scheduling, but not storage assignment
- In contrast to storage, K8s provides a rich set of Pod scheduling features
  - Resource requirements
  - Node selectors
  - Pod affinity / anti-affinity
  - Taints and tolerations

Agenda

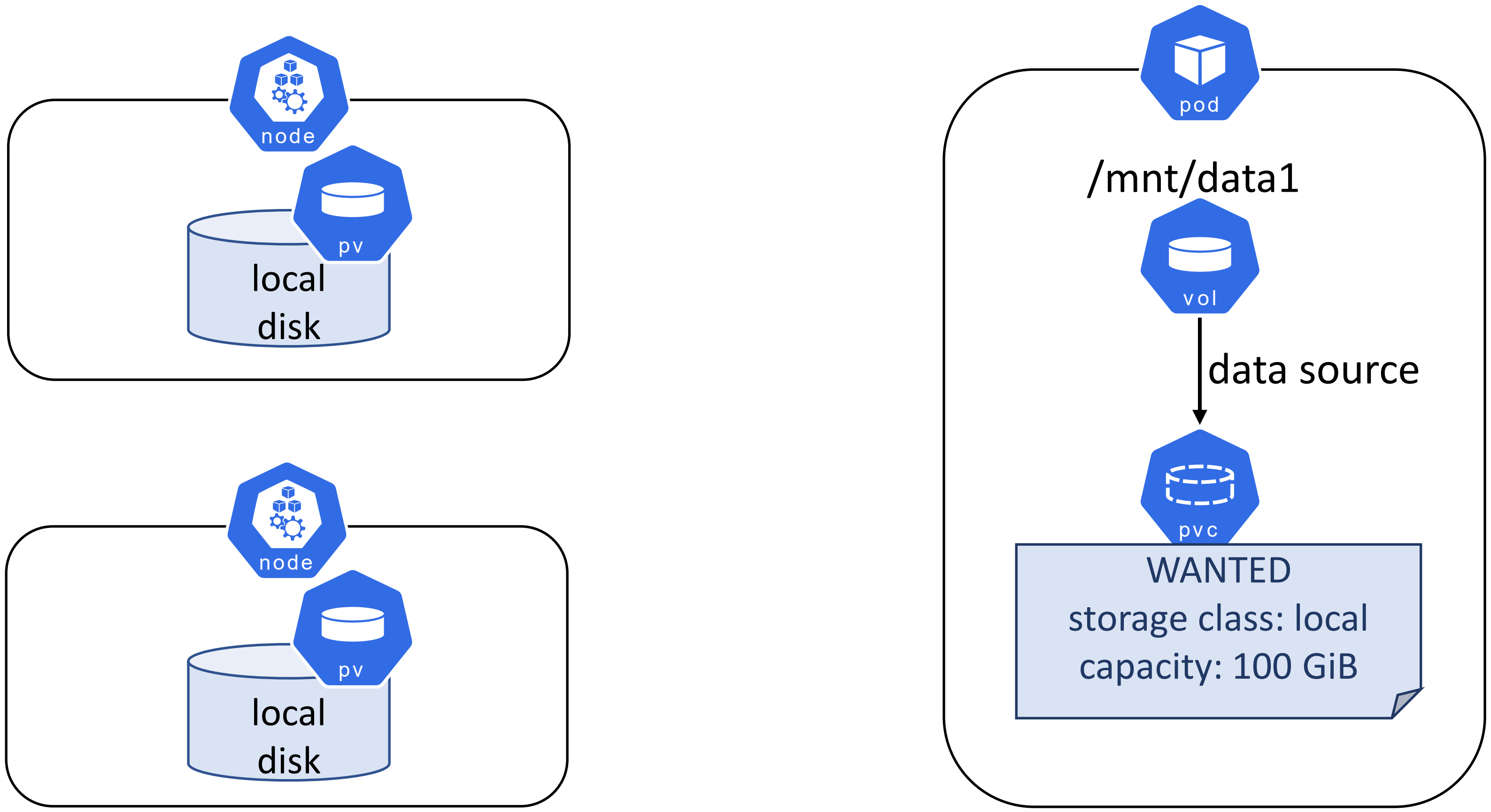
- Challenge: Running distributed storage system on K8s
  - Recap: Volume management in K8s
  - Problem: How to place storage devices optimally
- **Basic idea: WaitForFirstConsumer + pod topology spread constraints**
- Implementation using whenUnsatisfiable stanza
- Tuning: kube-scheduler configuration for the optimal placement
- Demo

Basic idea

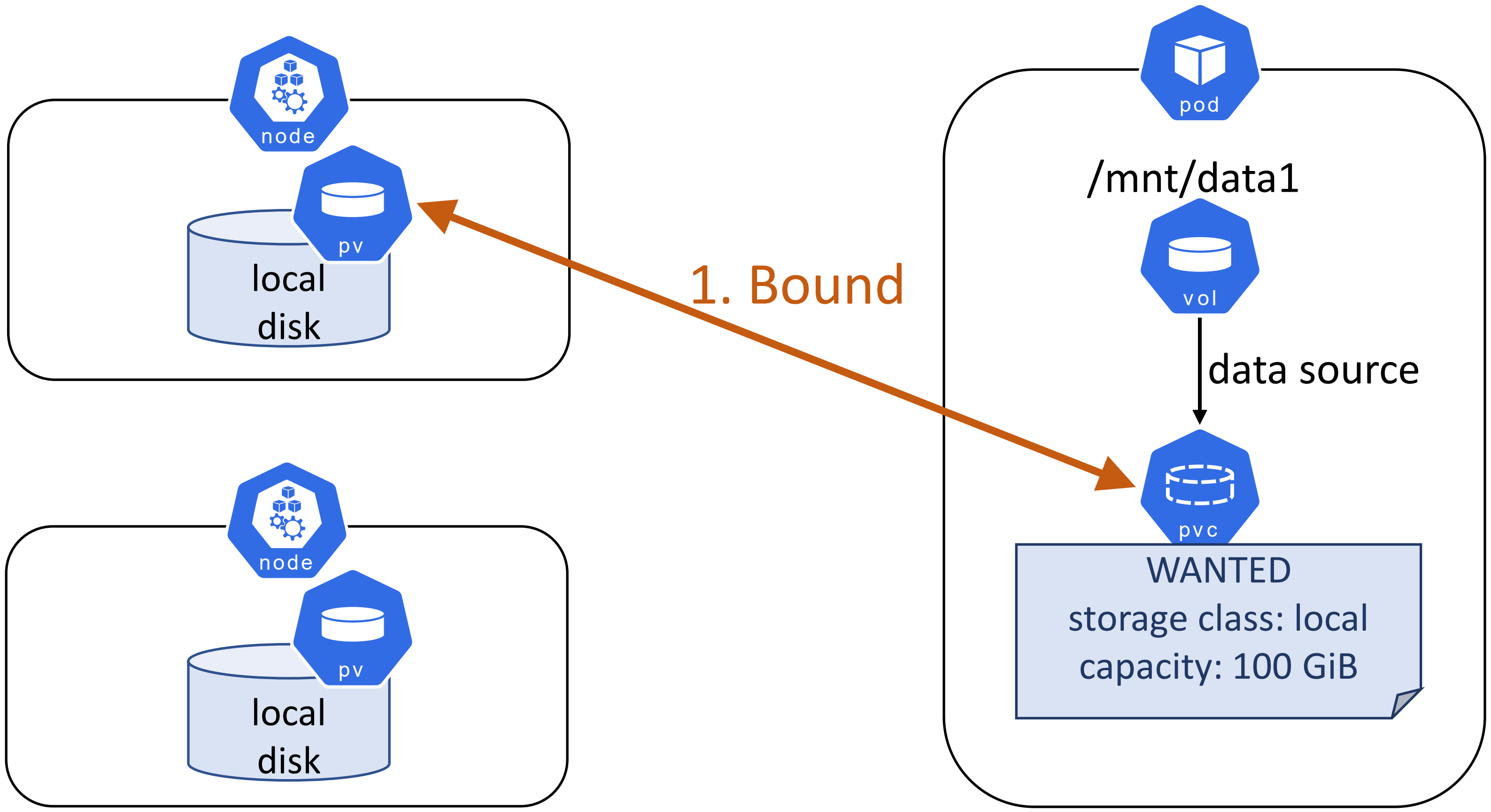
- Use “volumeBindingMode: WaitForFirstConsumer” in StorageClass
  - *… the WaitForFirstConsumer mode which will delay the binding and provisioning of a PersistentVolume until a Pod using the PersistentVolumeClaim is created.*

    https://kubernetes.io/docs/concepts/storage/storage-classes/
- 2 values for volumeBindingMode
  - Immediate (default)
  - WaitForFirstConsumer

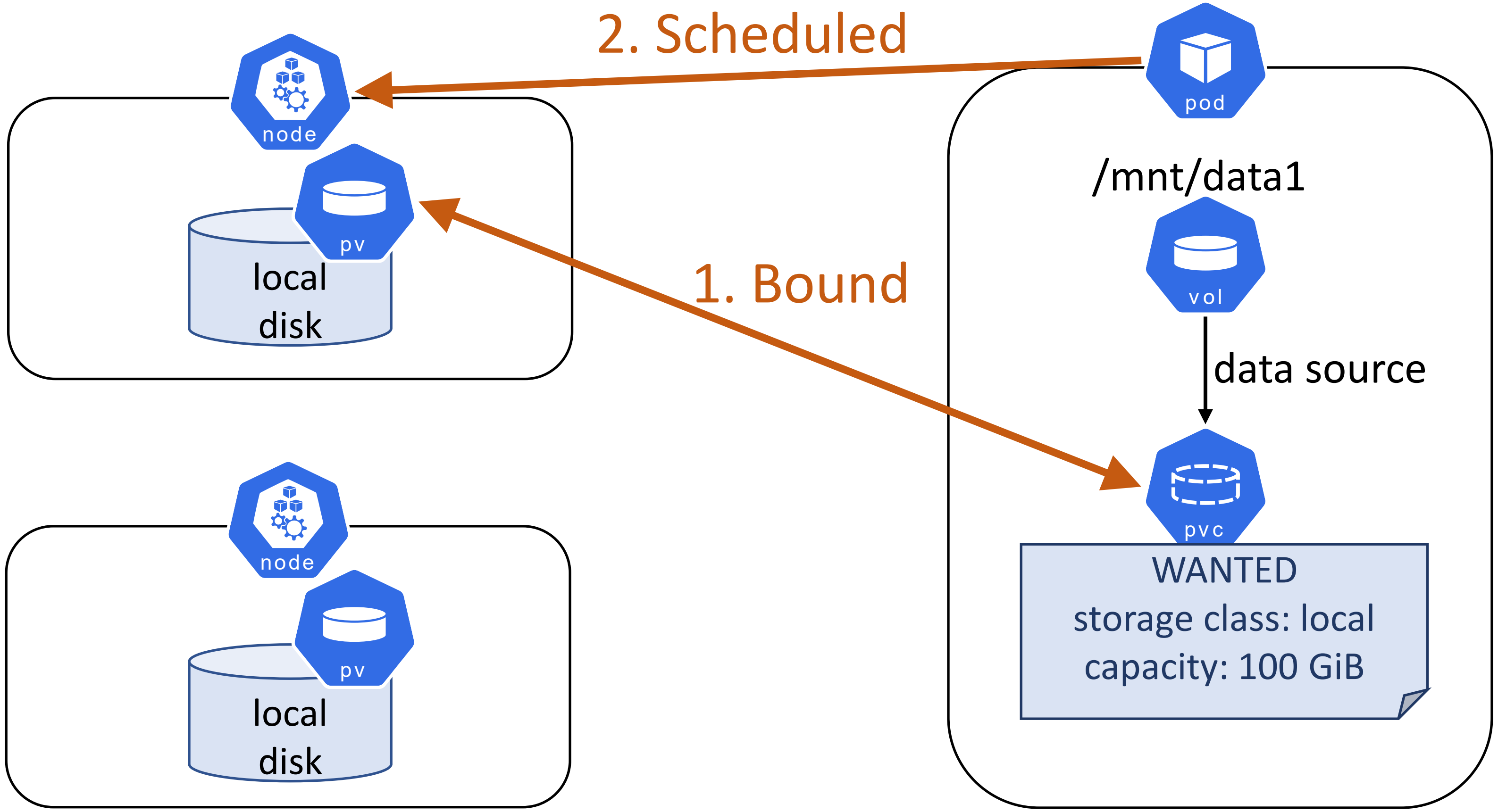
volumeBindingMode: Immediate (default)



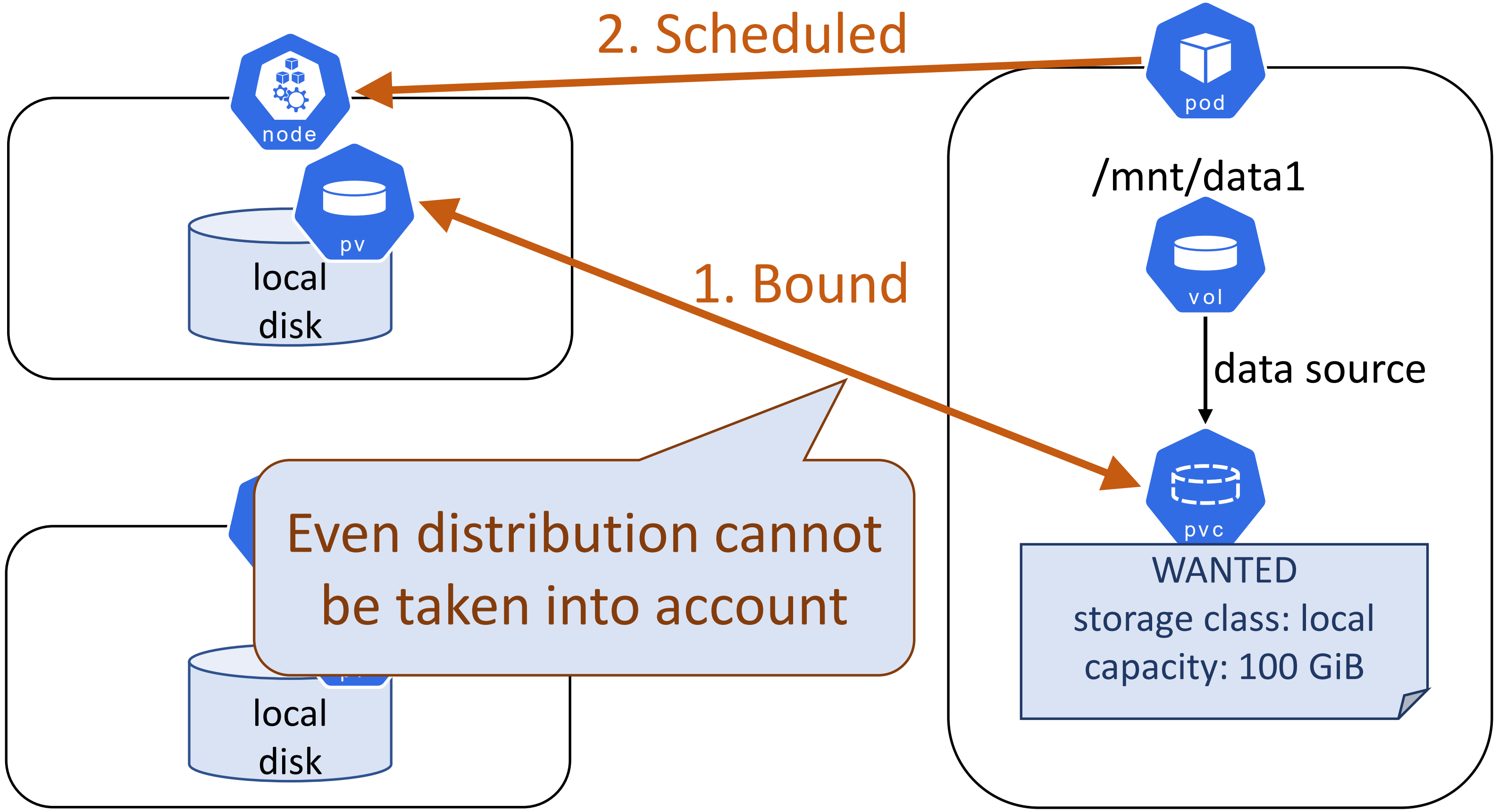
volumeBindingMode: Immediate (default)



volumeBindingMode: Immediate (default)



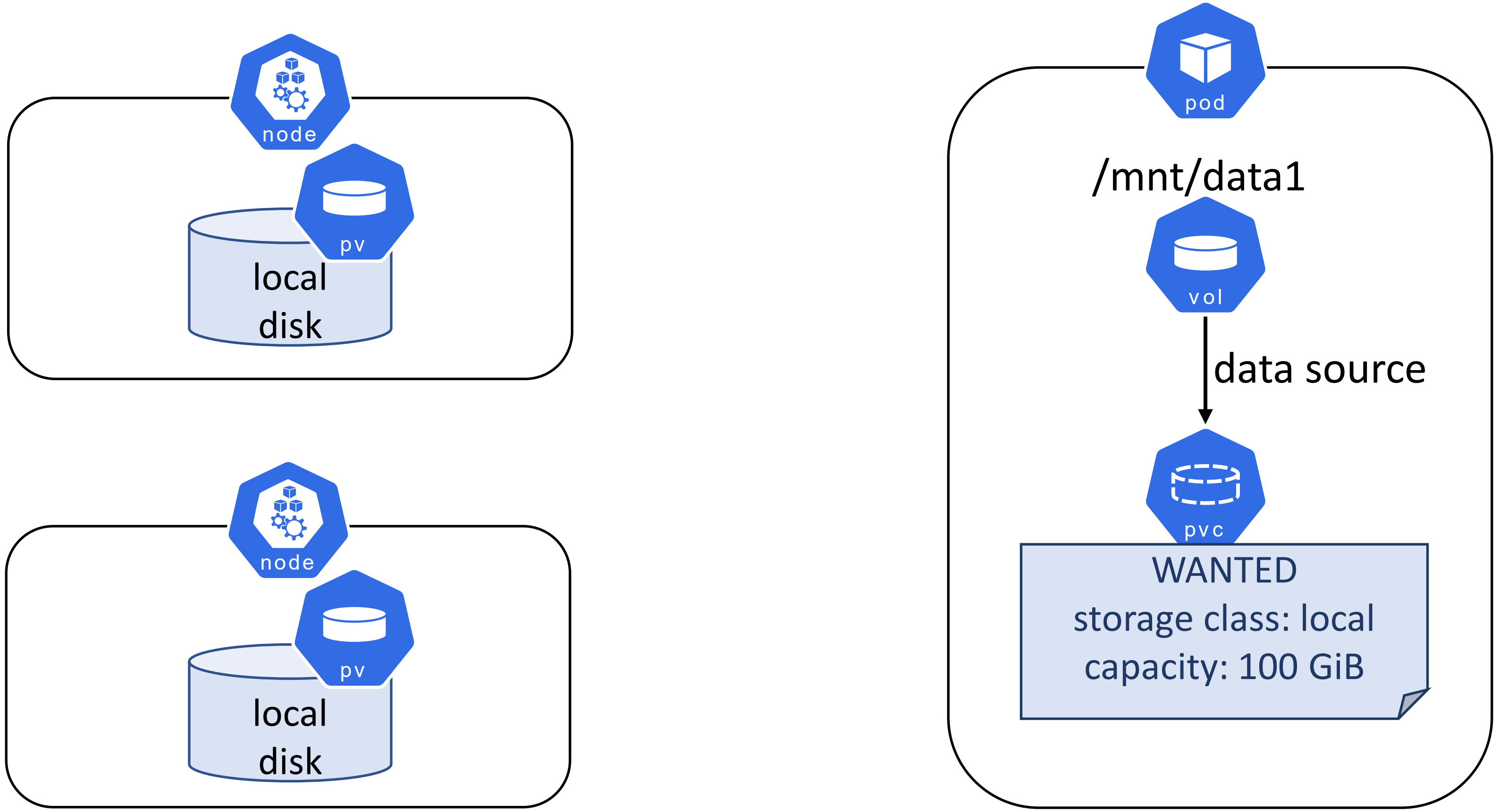
volumeBindingMode: Immediate (default)



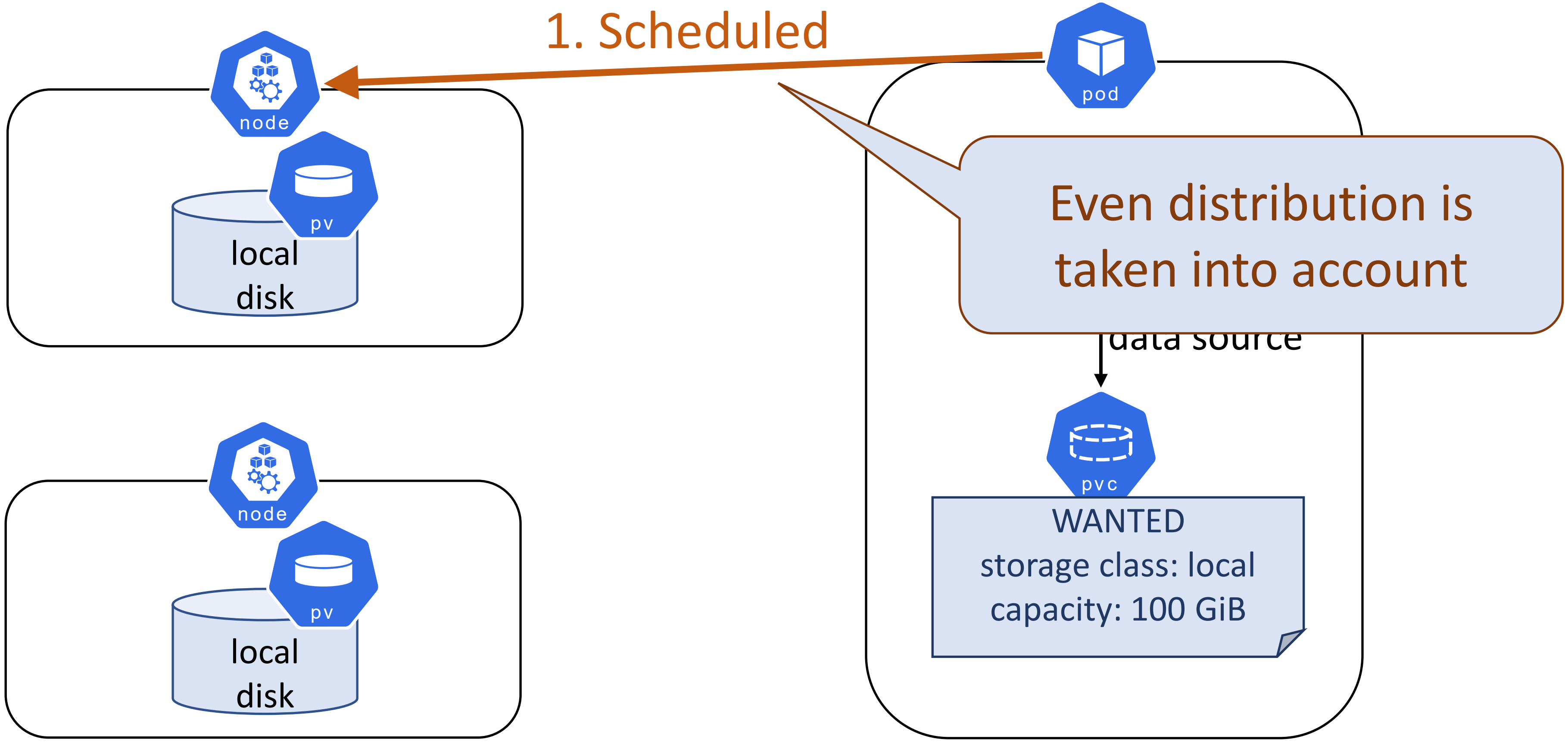
WANTED
storage class: local
capacity: 100 GiB

Even distribution cannot be taken into account

volumeBindingMode: WaitForFirstComsumer

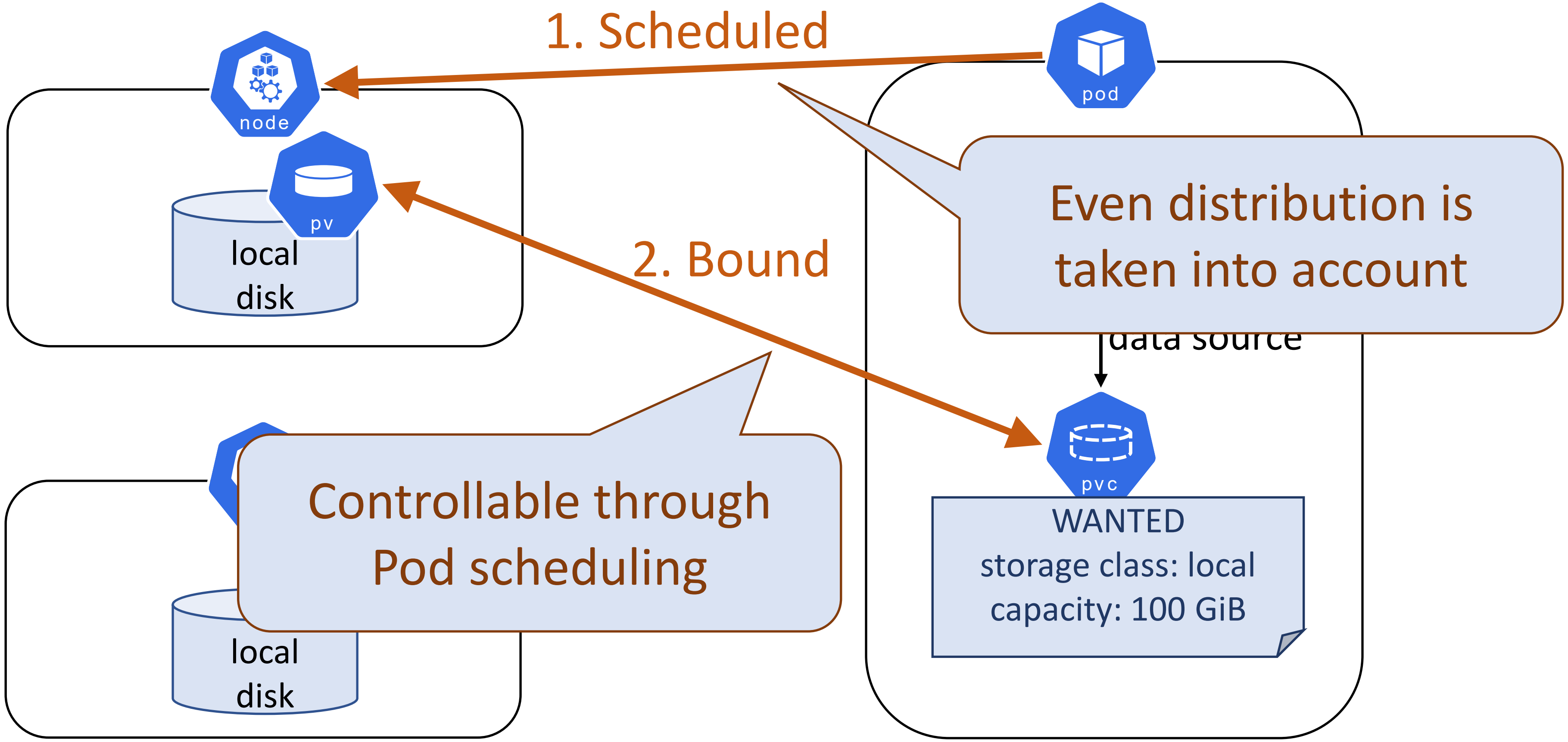


volumeBindingMode: WaitForFirstComsumer



Even distribution is taken into account

volumeBindingMode: WaitForFirstComsumer



Even distribution is taken into account

Controllable through Pod scheduling

Basic idea

- Use “volumeBindingMode: WaitForFirstConsumer” in StorageClass
- **Translate** the problem of **storage allocation** into the problem of **Pod scheduling**
  - Now we can utilize K8s’s rich set of Pod scheduling

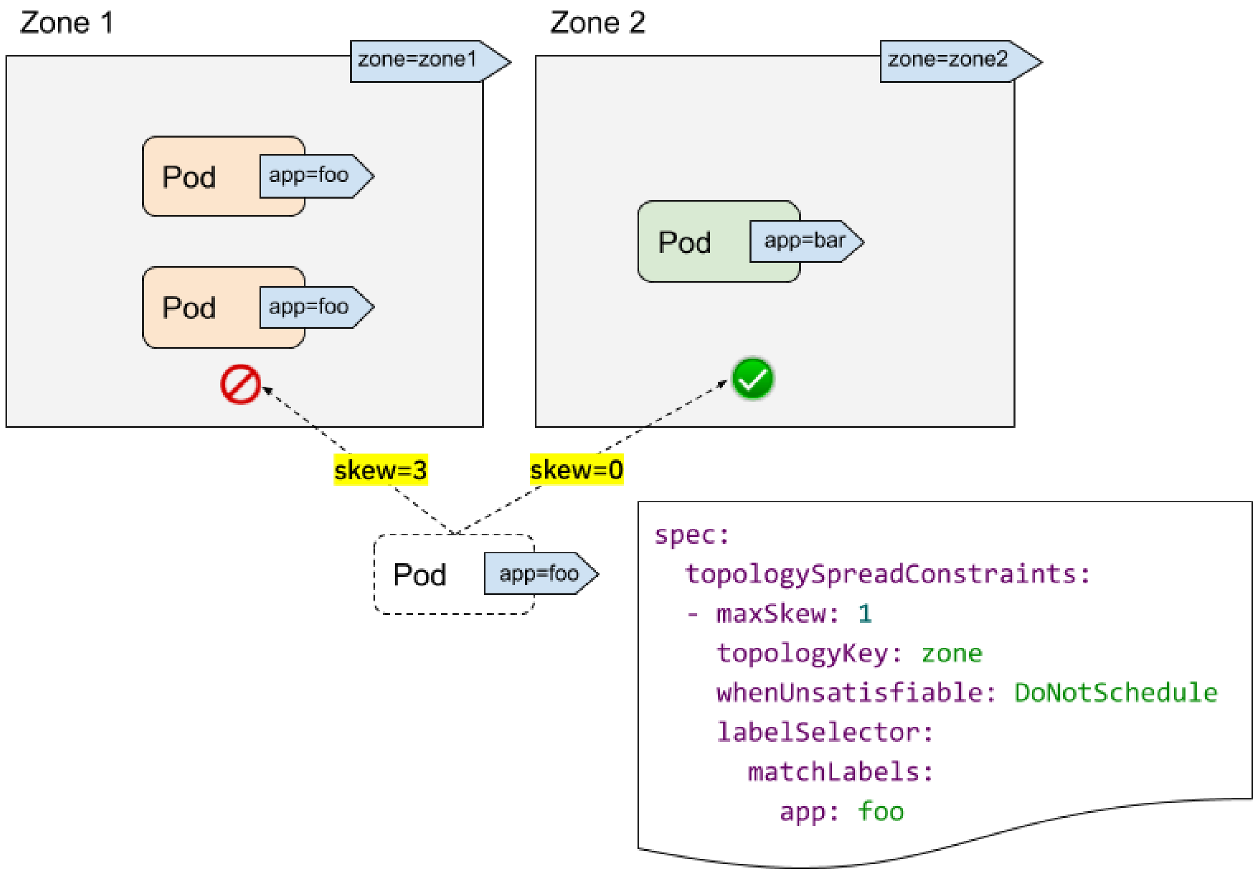
- Original challenge: Distributing PVs for local disks evenly through PVCs
- **Translated challenge: Distributing Pods with PVCs evenly**

Pod scheduling criteria

- Anti-affinity
  - Only consider whether Pods overlap or not
  - Cannot handle the case of multiple Pods/PVs in one Node
- ✓ Pod Topology Spread Constraints
  - Alpha in K8s 1.16, beta in 1.18, stable in 1.19
  - Compute scheduling score based on the skew
  - *You can use topology spread constraints to control how Pods are spread across your cluster among failure-domains such as regions, zones, nodes, and other user-defined topology domains*

    https://kubernetes.io/docs/concepts/workloads/pods/pod-topology-spread-constraints/

Pod Topology Spread Constraints



```
spec:
  topologySpreadConstraints:
  - maxSkew: 1
    topologyKey: zone
    whenUnsatisfiable: DoNotSchedule
    labelSelector:
      matchLabels:
        app: foo
```

skew = Pods **number** matched in **current** topology - **min** Pods matches in a topology

https://kubernetes.io/blog/2020/05/introducing-podtopologyspread/

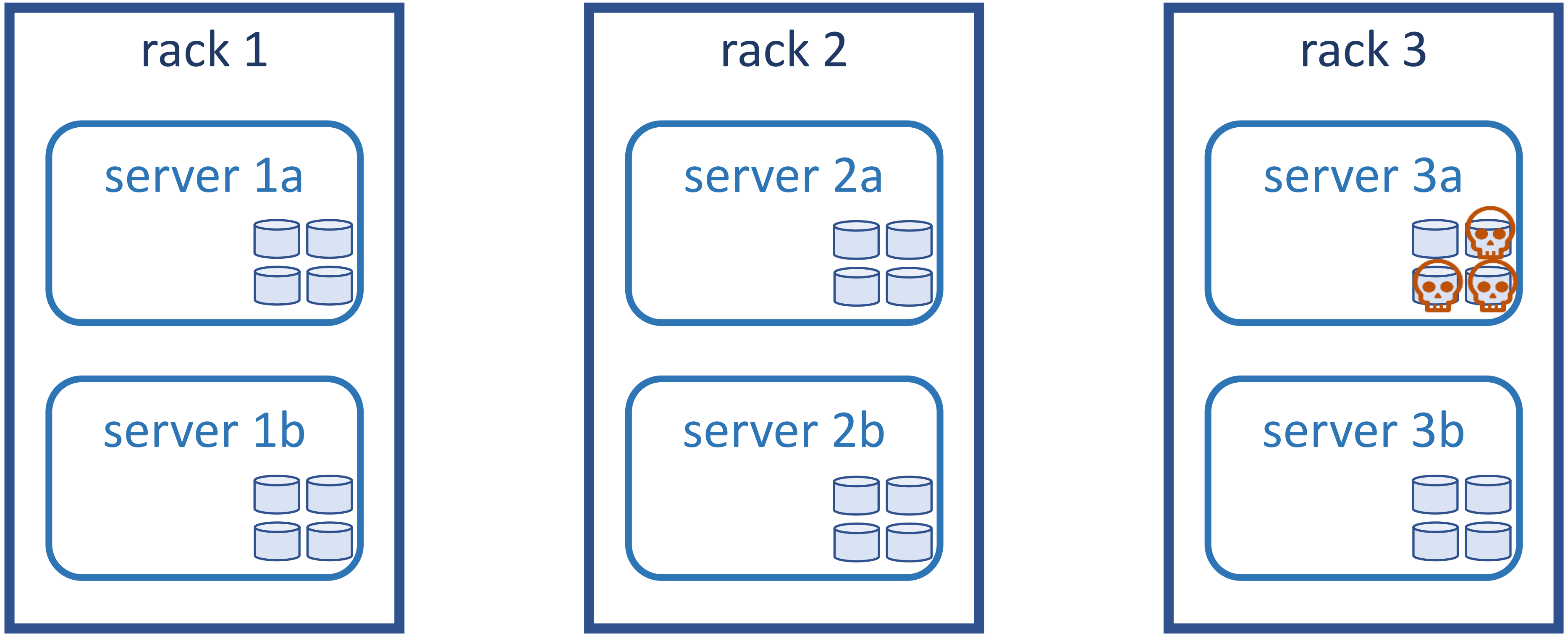
Agenda

- Challenge: Running distributed storage system on K8s
  - Recap: Volume management in K8s
  - Problem: How to place storage devices optimally
- Basic idea: WaitForFirstConsumer + pod topology spread constraints
- **Implementation using whenUnsatisfiable stanza**
- Tuning: kube-scheduler configuration for the optimal placement
- Demo

Evenness in the real world

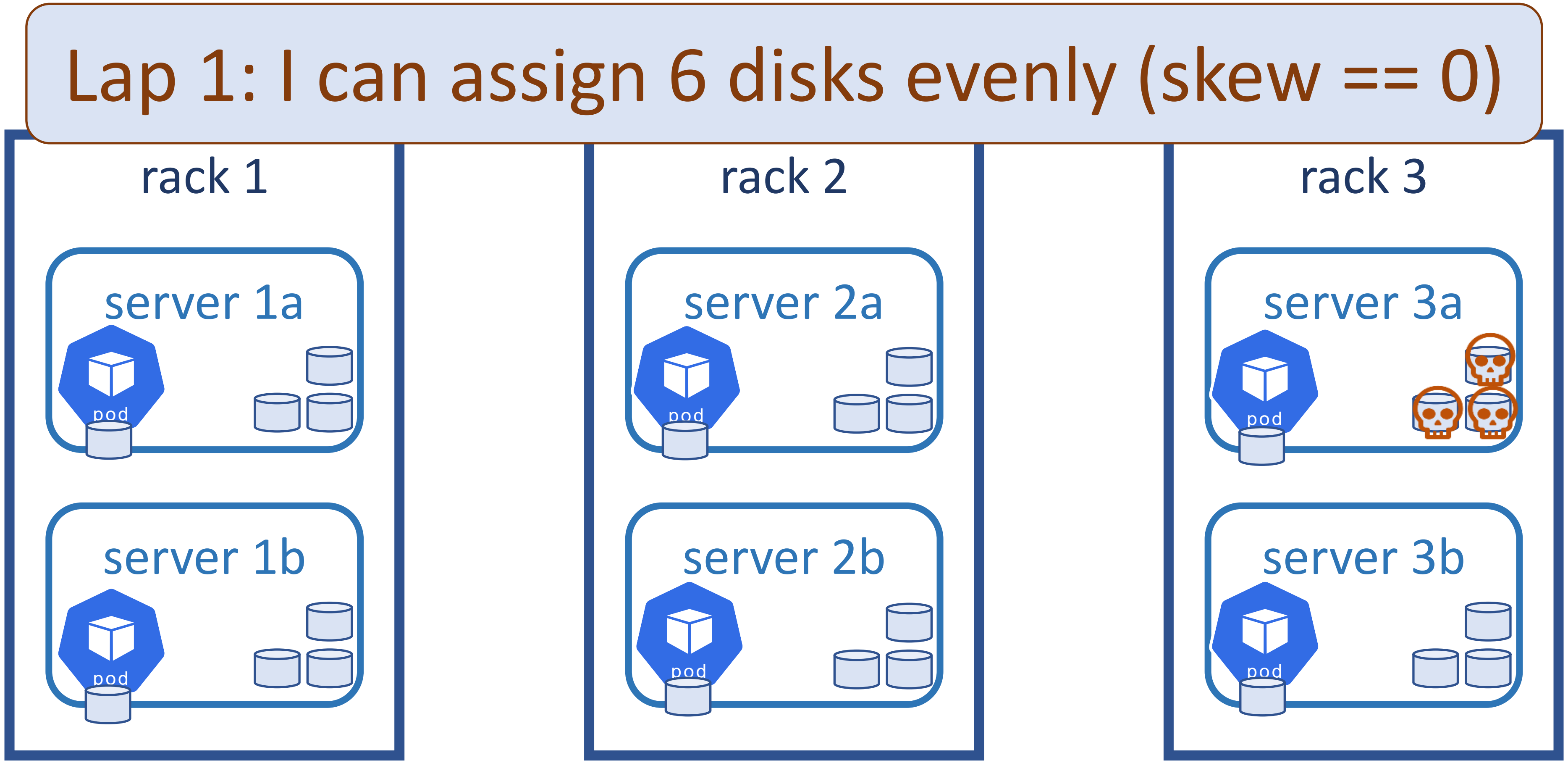
- Strict evenness is not desirable in the real world

Constraints: maxSkew = 1 for racks && maxSkew = 1 for servers



Evenness in the real world

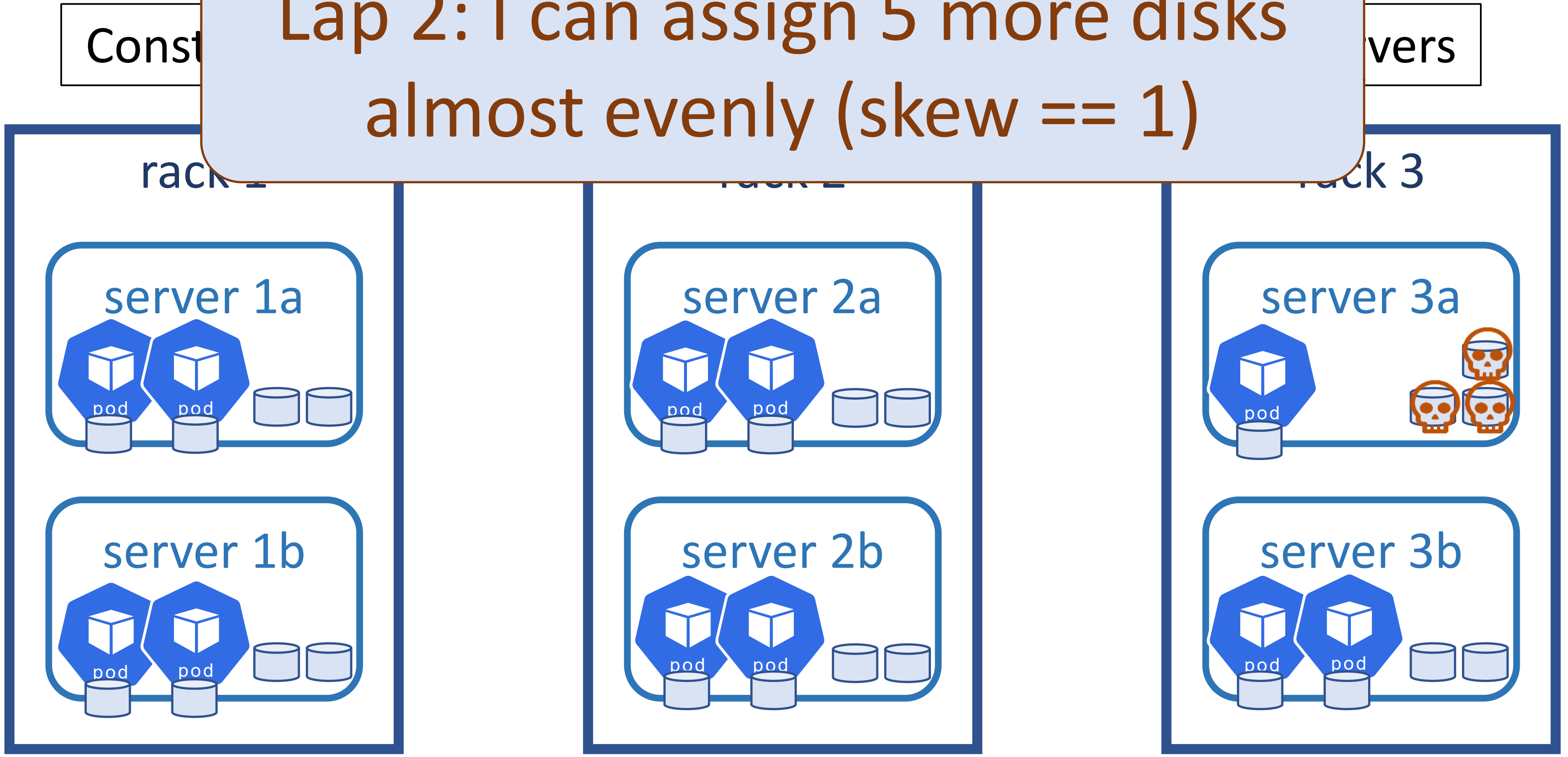
- Strict evenness is not desirable in the real world



Lap 1: I can assign 6 disks evenly (skew == 0)

Evenness in the real world

- Strict evenness is not desirable in the real world



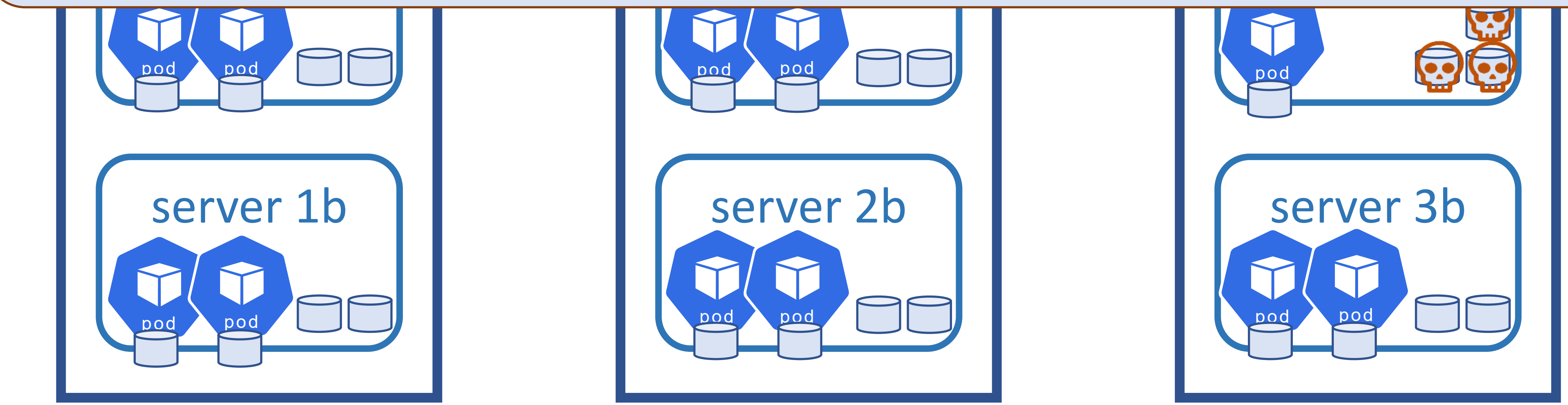
Lap 2: I can assign 5 more disks almost evenly (skew == 1)

Evenness in the real world

- Strict evenness is not desirable in the real world

Constraints: maxSkew = 1 for racks && maxSkew = 1 for servers

I cannot assign any more disks due to constraints;
Do I need to stop assignment here?



Relaxing the constraints

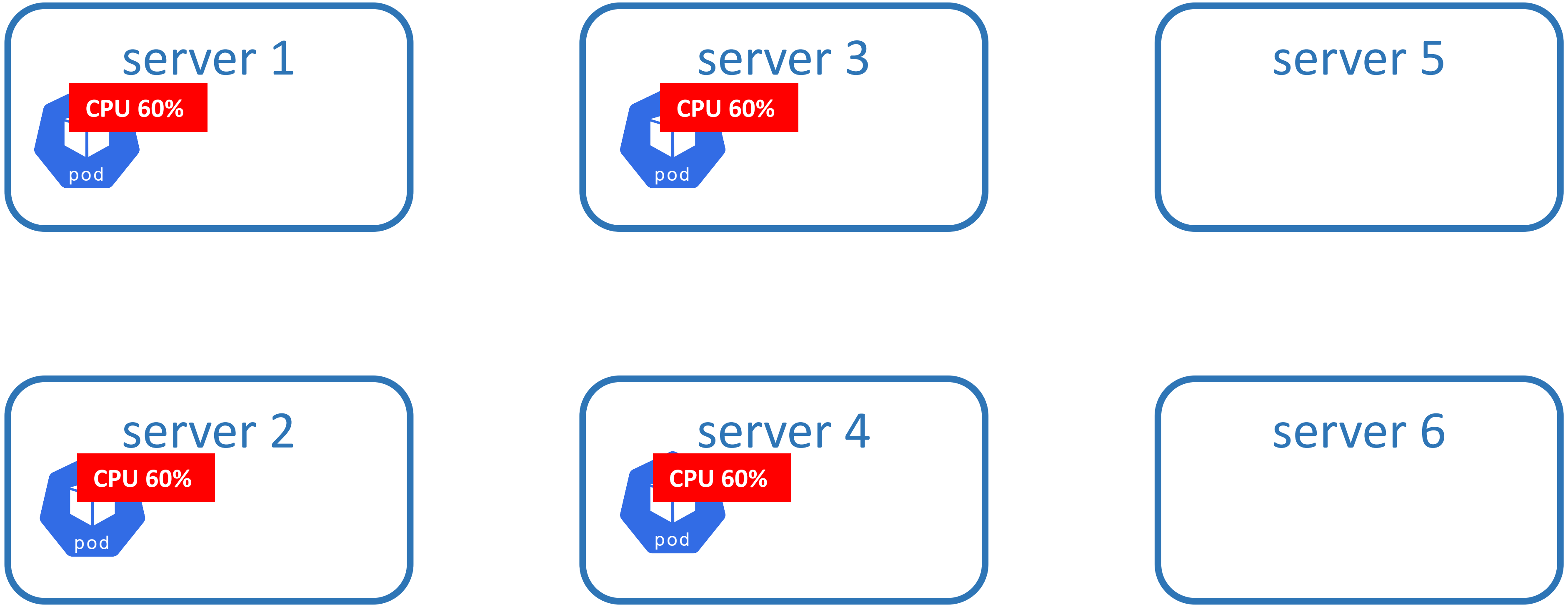
- whenUnsatisfiable indicates how to deal with a Pod if it doesn't satisfy the spread constraint:
  - DoNotSchedule (default): not to schedule the Pod
  - ✓ ScheduleAnyway: to still schedule the Pod while prioritizing nodes that ***minimize the skew***
- We tried ScheduleAnyway and …

Expected behavior of ScheduleAnyway

- If satisfiable, kube-scheduler ***always*** schedules the Pod within the constraints

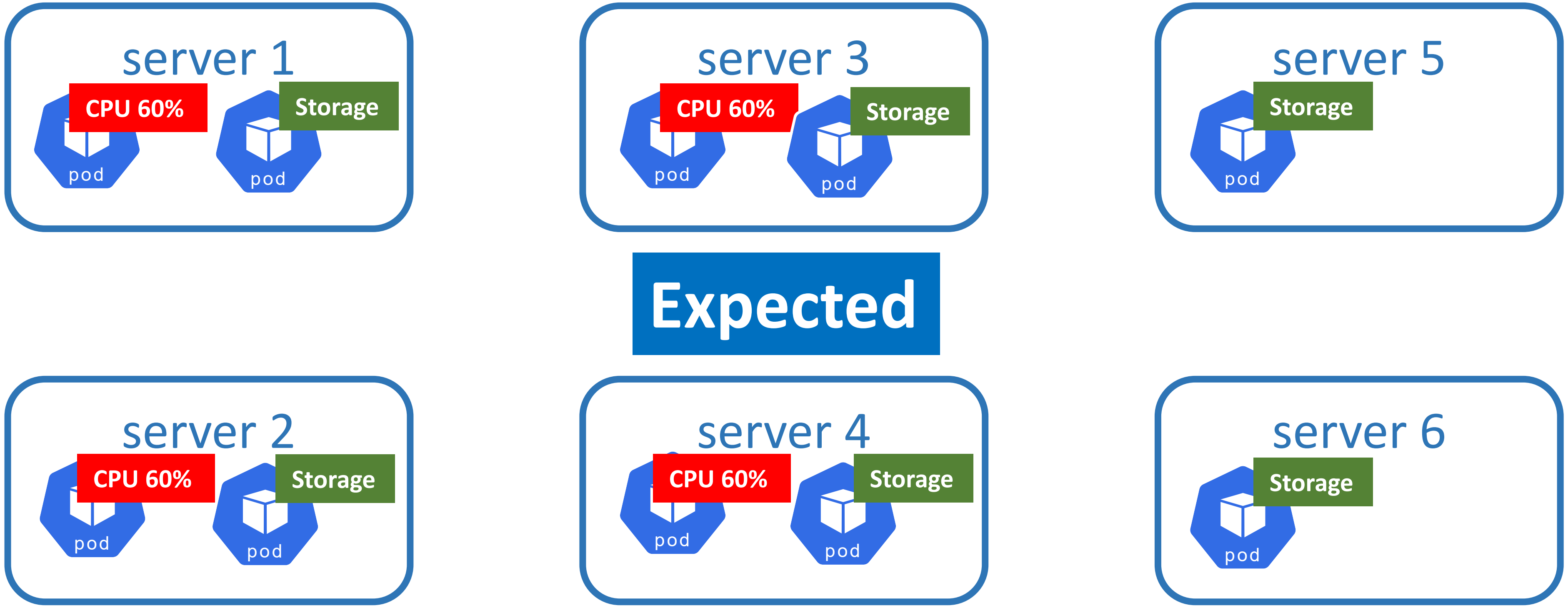
Actual behavior of ScheduleAnyway

Constraint for storage management Pods: maxSkew = 1 for servers



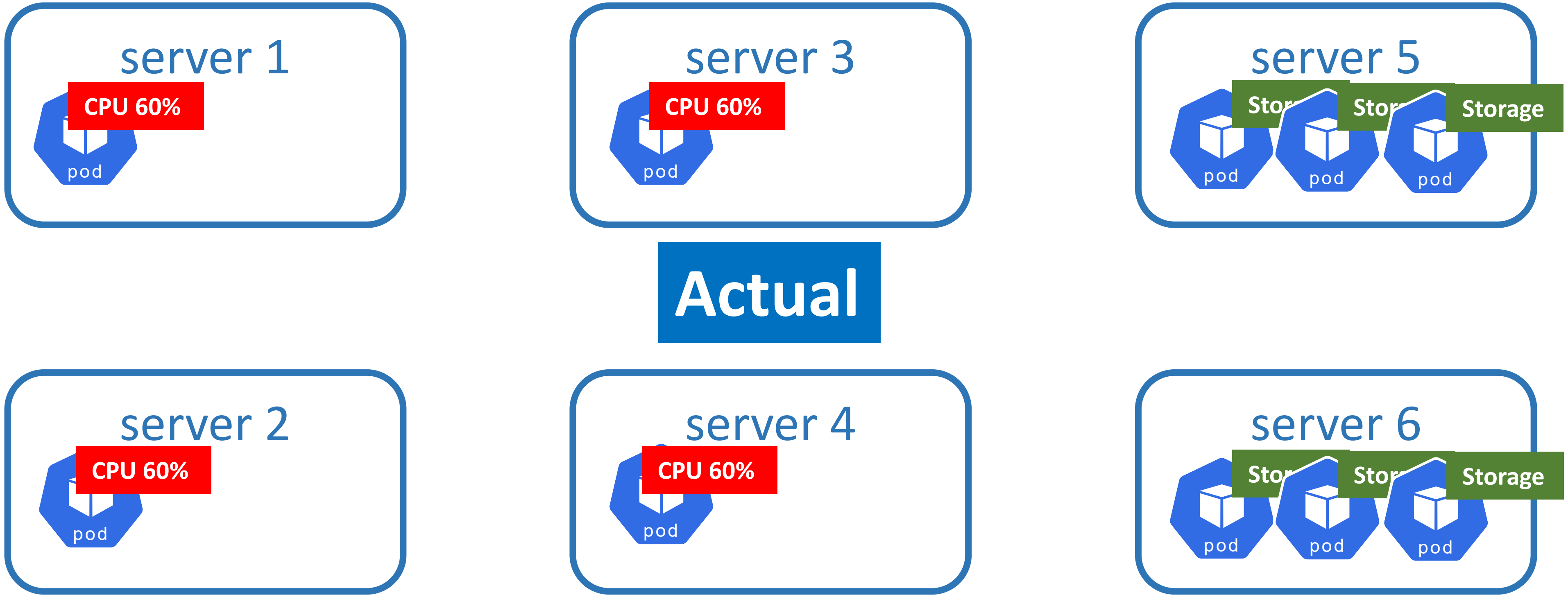
Actual behavior of ScheduleAnyway

Constraint for storage management Pods: maxSkew = 1 for servers



Actual behavior of ScheduleAnyway

Constraint for storage management Pods: maxSkew = 1 for servers



The Truth

With whenUnsatisfiable == ScheduleAnyway

- Expected behavior:
  - If satisfiable, …
  - If not satisfiable, …
- Actual behavior:
  - Whether the constraints are satisfiable or not, kube-scheduler **no longer treats them as real constraints**
  - Instead, they are treated as a part of the **scoring factors**
  - As a result, flattening CPU resource usage can have a higher priority than the Pod Topology Spread Constraints

Tuning of kube-scheduler

- Tune kube-scheduler to weigh the topology spread constraints more heavily
  - K8s 1.17: adjust the scheduling policy
    - Set EvenPodsSpreadPriority's weight to 500
    - The scheduling policy is applied globally, so do it carefully
  - K8s 1.18+: create a new scheduling profile and adjust it
    - 1.18: Set PodTopologySpread's weight to 500
    - 1.19: Disable NodeResourcesBalancedAllocation

Tuning in K8s 1.17

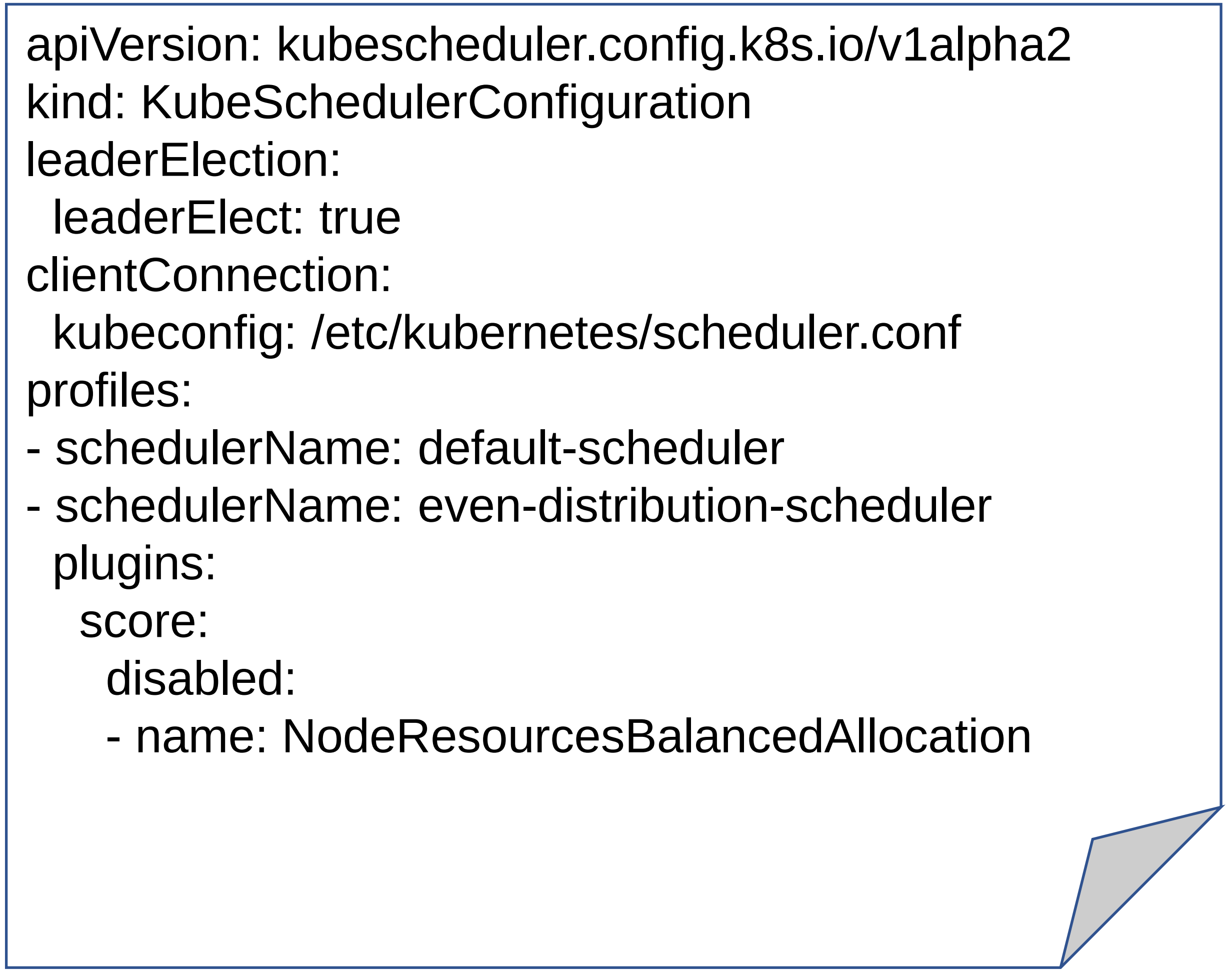
```
apiVersion: kubescheduler.config.k8s.io/v1alpha1
kind: KubeSchedulerConfiguration
leaderElection:
  leaderElect: true
clientConnection:
  kubeconfig: /etc/kubernetes/scheduler/kubeconfig
schedulerName: default-scheduler
algorithmSource:
  policy:
    file:
      path: /etc/kubernetes/scheduler/policy.cfg
```

```
{
  "apiVersion": "v1",
  "kind": "Policy",
  "predicates": null,
  "hardPodAffinitySymmetricWeight": 0,
  "alwaysCheckAllPredicates": false,
  "priorities": [
    {
      "name": "NodePreferAvoidPodsPriority",
      "weight": 100000,
      "argument": null
    },
    {
      "name": "EvenPodsSpreadPriority",
      "weight": 500,
      "argument": null
    },
    {
      "name": "SelectorSpreadPriority",
      "weight": 1,
      "argument": null
    },
    (and other priorities are listed here with weight == 1)
  ]
}
```

Tuning in K8s 1.18

```yaml
apiVersion: kubescheduler.config.k8s.io/v1alpha2
kind: KubeSchedulerConfiguration
leaderElection:
  leaderElect: true
clientConnection:
  kubeconfig: /etc/kubernetes/scheduler.conf
profiles:
- schedulerName: default-scheduler
- schedulerName: even-distribution-scheduler
  plugins:
    score:
      disabled:
      - name: PodTopologySpread
      enabled:
      - name: PodTopologySpread
        weight: 500
```

Tuning in K8s 1.19



```
apiVersion: kubescheduler.config.k8s.io/v1alpha2
kind: KubeSchedulerConfiguration
leaderElection:
  leaderElect: true
clientConnection:
  kubeconfig: /etc/kubernetes/scheduler.conf
profiles:
- schedulerName: default-scheduler
- schedulerName: even-distribution-scheduler
  plugins:
    score:
      disabled:
      - name: NodeResourcesBalancedAllocation
```

Demo

- There are 4 Nodes
- 2 computing Pods are running
- 5 OSD Pods are placed for storage management; are they distributed evenly?
  - With default kube-scheduler
  - With tuned kube-scheduler

Key Takeaways

- Translate local storage distribution into Pod scheduling using WaitForFirstConsumer
- Use Pod topology spread constraints to give better scheduling criteria
- Tune kube-scheduler to prioritize Pod topology spread constraints
  - for K8s 1.17, 1.18, 1.19
- Our configuration for Rook/Ceph is open-sourced
  - https://github.com/cybozu-go/neco-apps, especially under “rook” directory

KEEP CLOUD NATIVE EVERYWHERE

KubeCon | CloudNativeCon

North America 2020

Virtual

FYI

- How to expose node-local storage devices as PVs
  - Local Persistence Volume Static Provisioner
    - https://github.com/kubernetes-sigs/sig-storage-local-static-provisioner
    - Scan local devices according to the specified pathname patterns, and expose matched devices as PVs
    - Used for static preparation of PVs
    - Dynamic binding is applicable
  - TopoLVM
    - https://github.com/topolvm/topolvm
    - Create a Logical Volume of the specified size from the given Volume Group, and expose the LV as a PV
    - Used for dynamic provisioning