



KubeCon



CloudNativeCon

Europe 2026

#KubeCon #CloudNativeCon

The Symbiosis of Storage and Workloads: Longhorn for Hyperconverged Block Storage

Lessons from operating large-scale storage at NAVER

Jangseon Ryu – NAVER Corp.

Jinhong Kim – NAVER Corp.



Who are we?



Senior Technical Lead
NAVER Corp



Software Engineer
NAVER Corp

Who is NAVER?

NAVER operates one of the largest digital ecosystems in South Korea.

Our services span search, commerce, fintech, content, and enterprise cloud platforms used by millions of users. Running these services requires operating large-scale cloud-native infrastructure

1. Major Portal & Information

Naver Search: South Korea's No. 1 search engine providing integrated search results (images, videos, etc.).

Naver News: Hub for major media articles featuring the customized 'Newsstand' subscription.

Naver Map: Comprehensive location-based service for route finding, navigation, and public transit.

2. Community & Social

Naver Cafe: Online community service for users gathering around shared interests.

Naver Blog: Personal media platform for sharing daily life and professional expertise.

3. Shopping & Financial

Naver Shopping: High-scale e-commerce platform for comparing and purchasing registered products.

Naver Pay: Simple payment service supporting online/offline transactions, wire transfers, and asset management.

4. Entertainment & Content

Naver Webtoon / Web Novel: Global digital platform for popular comic and novel content.

CHZZK: High-traffic, game-focused live streaming platform.

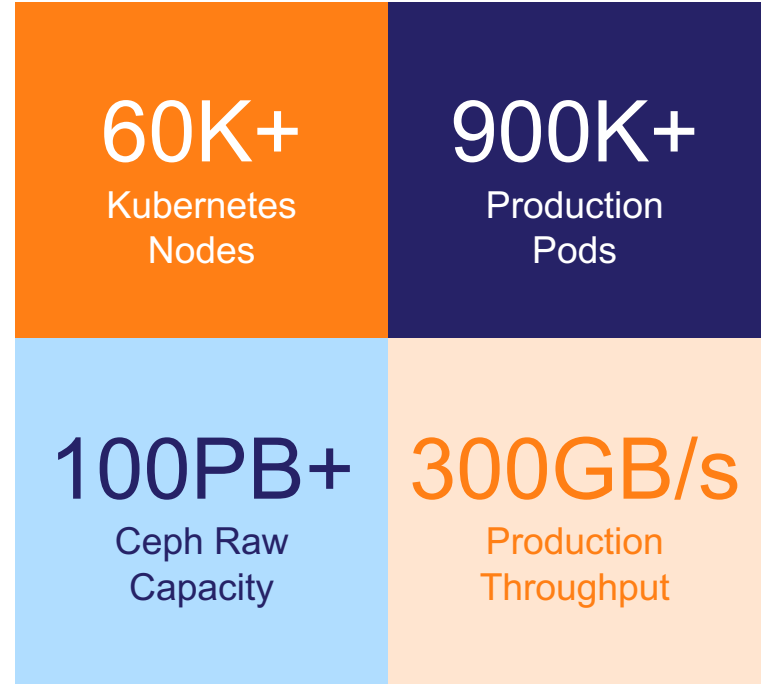
5. Business & Professional

Naver Cloud: Enterprise-grade cloud infrastructure providing IaaS, PaaS, and SaaS.

NAVER WORKS: All-in-one enterprise collaboration solution integrating email, messenger, and calendar.

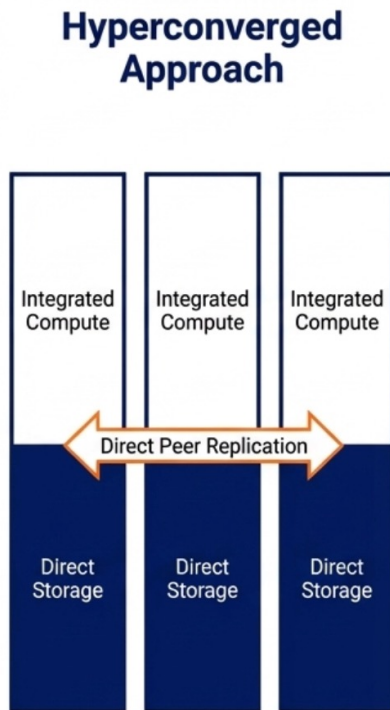
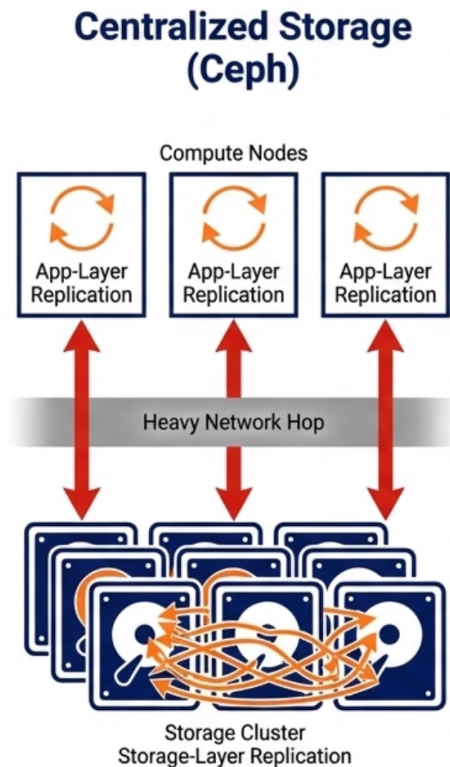
NAVER at Scale

- 60,000+ Kubernetes nodes and 900,000+ Production Pods
- 100 Petabytes of raw Ceph Storage capacity (5,000 nodes, 50,000 disks)
- 250,000 Persistent Volumes served
- 300 GB/s throughput handled reliably in production



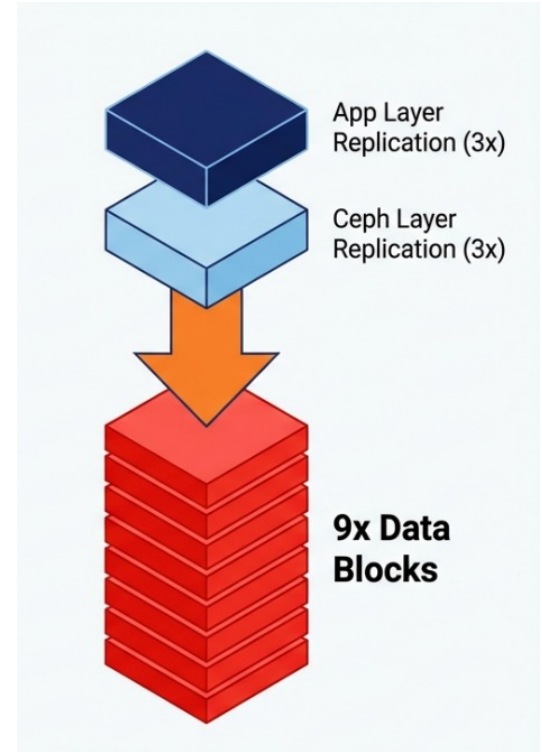
Ceph Works Well — But Not for Every Workload

- Ceph is a mature, highly scalable platform that powers our shared multi-cluster storage.
- But centralized storage is not universal. Many distributed applications already replicate data.
- Stacking application replication on storage replication creates redundancy and inefficiencies.
- This pushed us to rethink the model toward a hyperconverged approach.



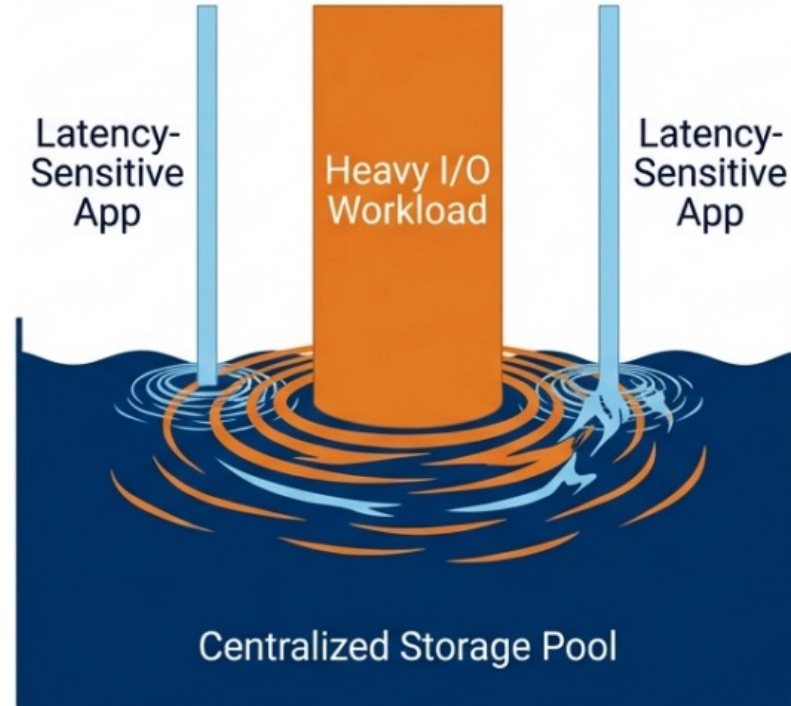
Problem #1: Duplicate Replication

- Ceph relies on 3-way replication or 1.5x erasure coding
- Distributed apps often independently replicate their own data (2x or 3x)
- Stacking these layers leads to immense cost inefficiency (up to 9 total copies)
- Causes severe I/O amplification and degrades overall storage performance



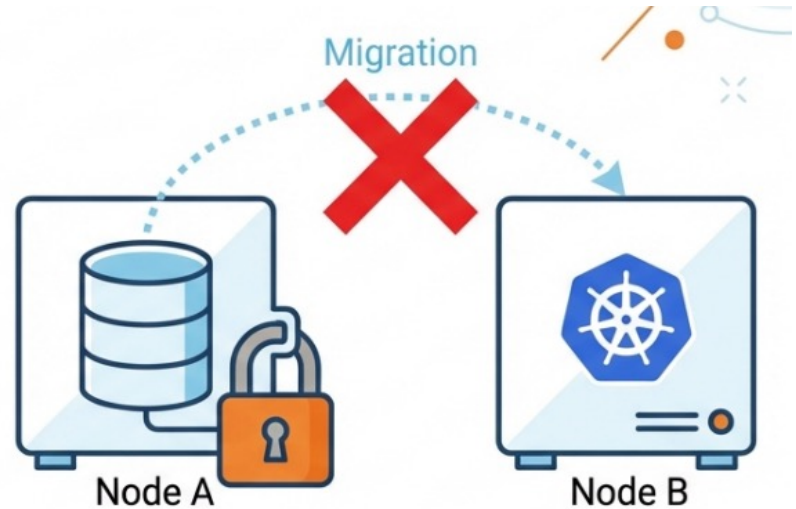
Problem #2: Performance Interference

- Applications assume volumes act as completely independent disks
- In reality, all volumes share the same underlying centralized pool
- Hardware issues, rebuilds, or heavy I/O from neighbors cause latency spikes
- Unacceptable environment for latency-sensitive, critical workloads



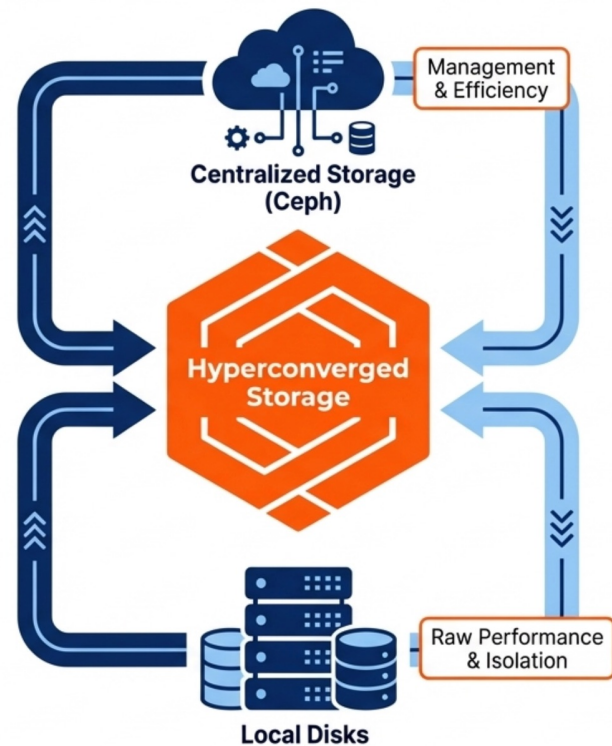
Problem #3 – The Limits of Local Disk

- Local disks fix replication and interference, but introduce Node Lock-in
- Volume become tightly bound to specific physical hardware
- Node failures cause immediate data unavailability; maintenance requires complex manual workflows
- Cannot share unused capacity across clusters (no thin or over-provisioning)



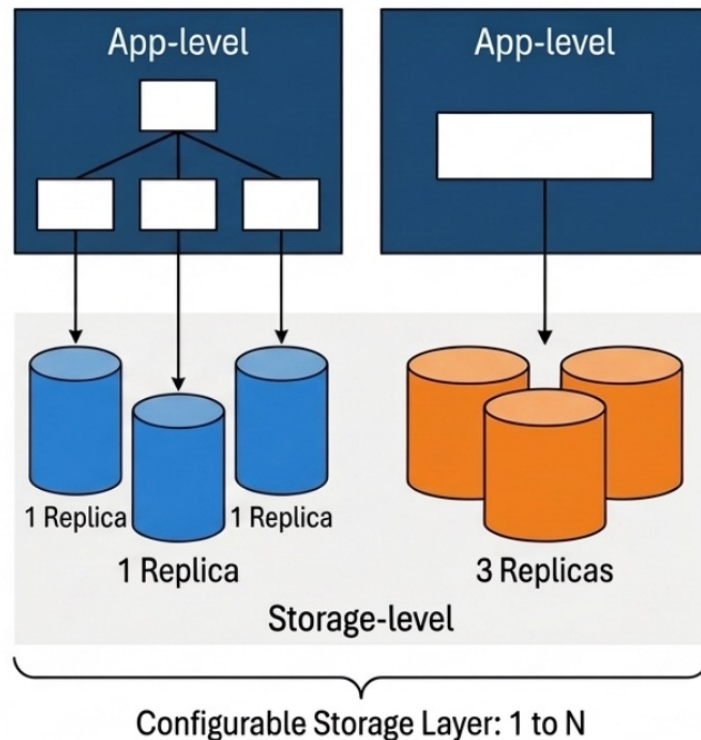
Why We Needed Hyperconverged Storage

- Find the balance between centralized storage and local disks.
- Avoid double replication when apps already replicate.
- Reduce noisy neighbor interference.
- Maintain locality while allowing volumes to move.
- Thin provisioning and shared capacity.
- Stable performance for sensitive workloads.



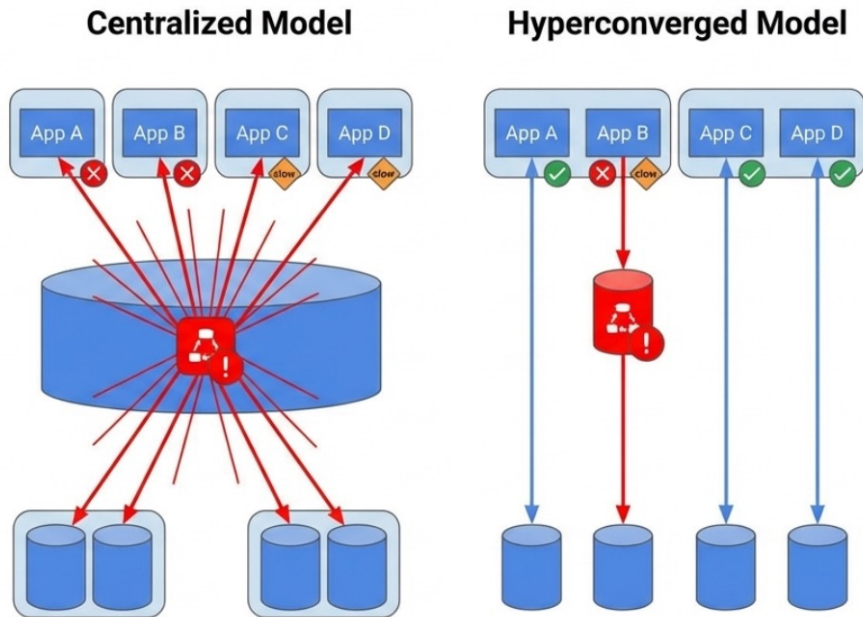
First Goal: Flexible Replication for Different Workloads

- Different workloads require entirely different replication strategies.
- App-level Replication: Distributed applications already replicate their own data. These only need 1 storage replica.
- Storage-level Replication : Legacy or simpler workloads without built-in replication require 2 or 3 replicas to prevent data loss.
- The Unifying Solution: Storage must be completely configurable (1 to N) based on specific workload needs.



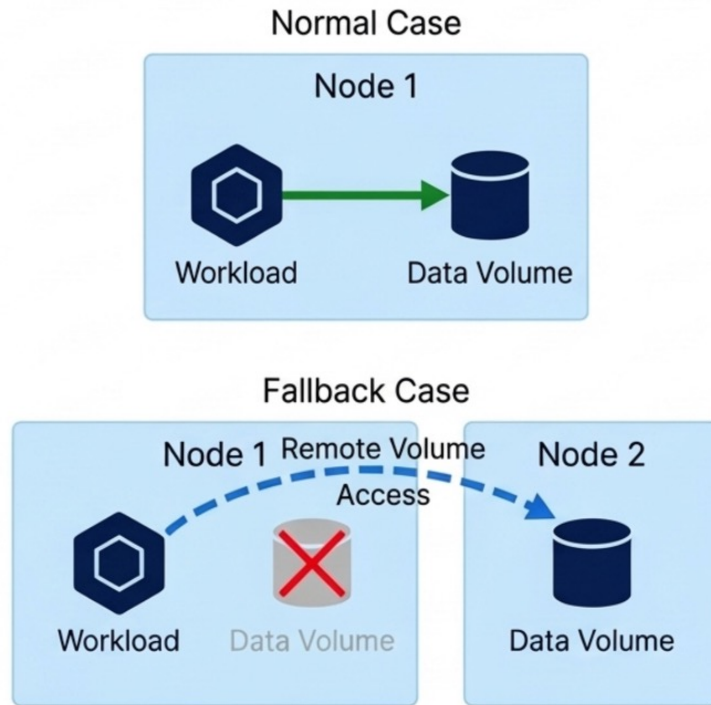
Second Goal: Better Workload Isolation

- Problem: Centralized storage systems (like Ceph) rely on shared resource pools, creating cluster-wide performance interference across volumes.
- Solution: The hyperconverged model fundamentally shifts this by distributing storage resources directly across specific nodes and individual disks.
- Benefit: This physical and logical separation guarantees superior workload isolation and neutralizes the impact of noisy neighbors.



Third Goal: Locality with Service Continuity

- Placing storage on the same node as the workload enables locality and delivers very low latency.
- However, nodes can fail, and locality should never break service availability.
- If the local node becomes unavailable, the workload can access its data remotely. Latency may increase slightly, but the service continues to run.

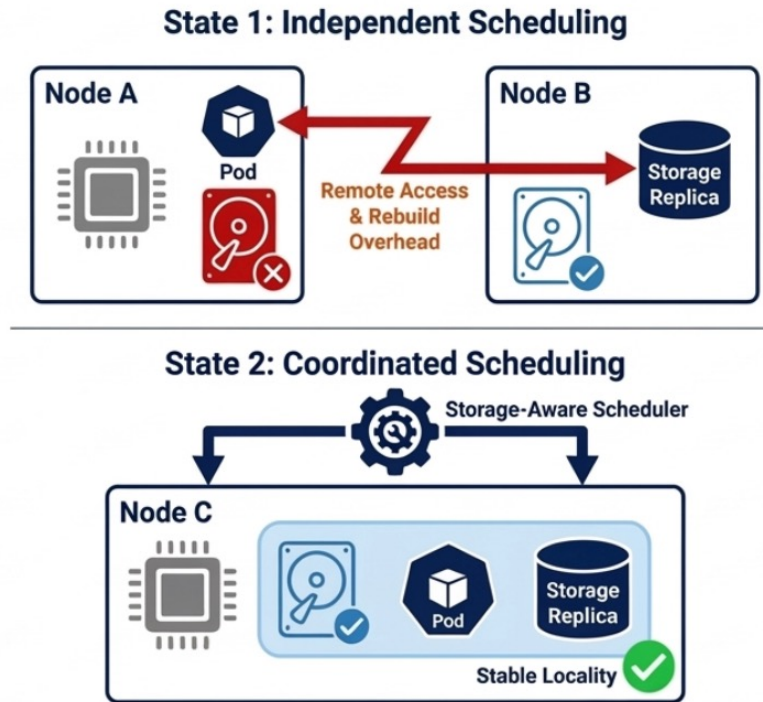


Fourth Goal: Coordinated Scheduling for Stable Locality

- Kubernetes schedules Pods based on compute resources, while Longhorn schedules replicas based on storage availability. These decisions are independent.

These decisions are independent.

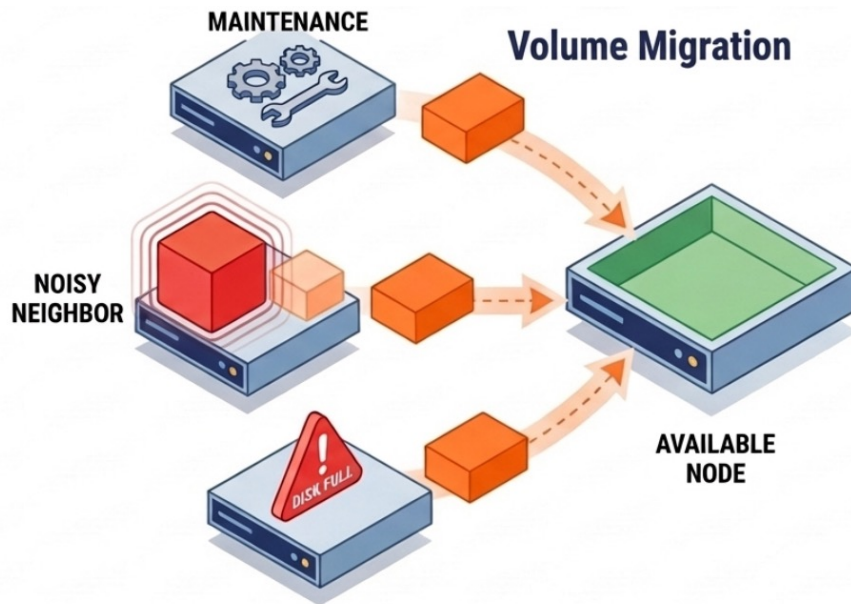
- Breakage: breaks locality. Pods may land on nodes without enough disk capacity. Remote access, unnecessary rebuilding
- Solution: Coordinated Scheduling. Make scheduling storage-aware so Pod and storage placement align.



Fifth Goal:

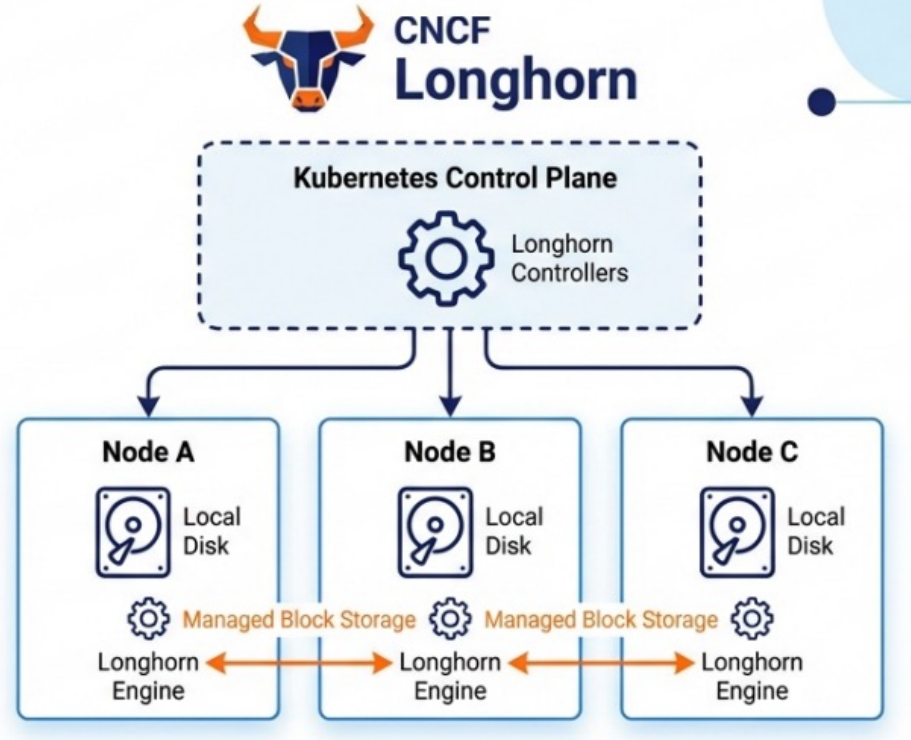
Storage Mobility for Flexible Operations

- Flexible Operations: Storage mobility enables agile infrastructure operations
- Operational Scenarios: Handle node maintenance, capacity limits, and noisy neighbors.
- Beyond Local Disks: Unlike node-bound local disks, Longhorn volumes can move when needed.



Solution: Longhorn

- Evaluated multiple CNCF storage projects to fit the hyperconverged model
- Selected Longhorn: A cloud-native distributed block storage for Kubernetes
- Runs entirely inside Kubernetes with native controllers and engines
- Utilized local disks but exposes them flexibly via standard CSI interfaces



KubeCon



CloudNativeCon

Europe 2026

Engineering the Solution

The Problem

- Pod scheduling (Compute-driven) vs. Storage scheduling (Capacity-driven) are decoupled, leading to "Locality Mismatch "

The Impact

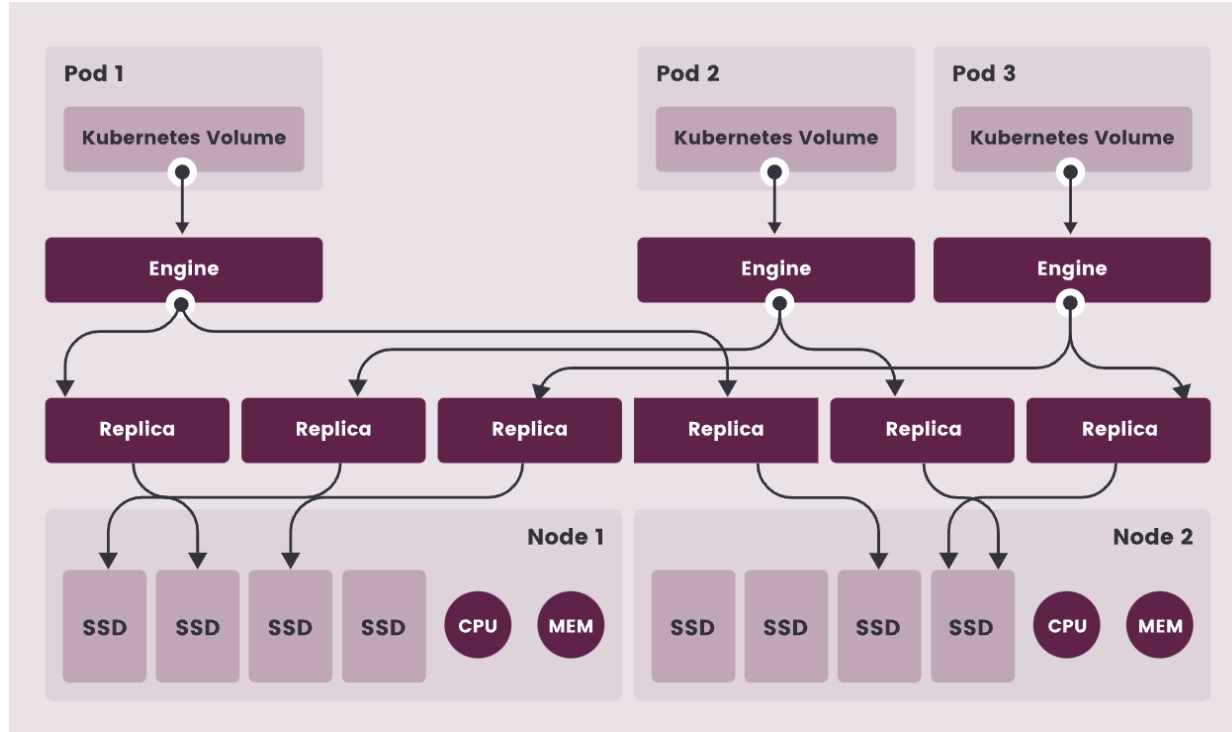
- Increased latency, high network traffic, and frequent, avoidable data rebuilding.

The Strategy

- Synchronizing Pod and storage placement to ensure that the workload and its data always reside on the same node or zone.

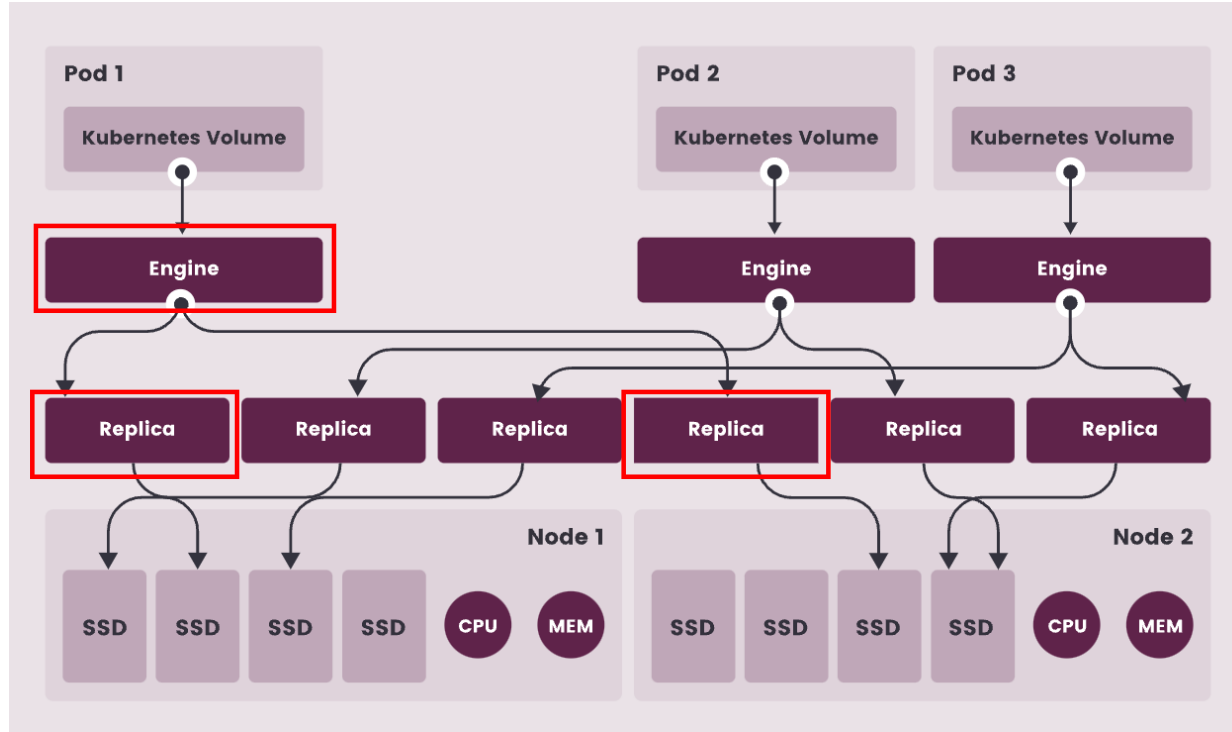


Longhorn Architecture



*<https://longhorn.io/>

Longhorn Architecture



*<https://longhorn.io/>

Where Should Replica Be placed?

Pod and Storage(replicas) were handled independently

- Pods were scheduled by Computing resource(kube-scheduler)
- Storage were scheduled by Storage resource(Longhorn)

Two possible approaches

1. Place replicas on the Pod's node
2. Place the Pod on the replicas' node
 - Can **conflict with** Pod affinity policies

Where Should Replica Be placed?

Pod and Storage(replicas) were handled independently

- Pods were scheduled by Computing resource(kube-scheduler)
- Storage were scheduled by Storage resource(Longhorn)

Two possible approaches

1. **Place replicas on the Pod's node**
2. Place the Pod on the replicas' node
 - Can **conflict with** Pod affinity policies

DataLocality paramater

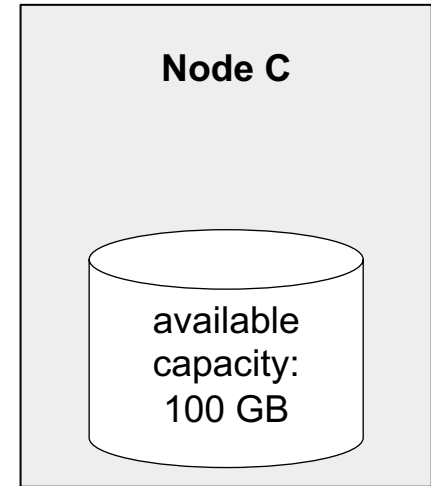
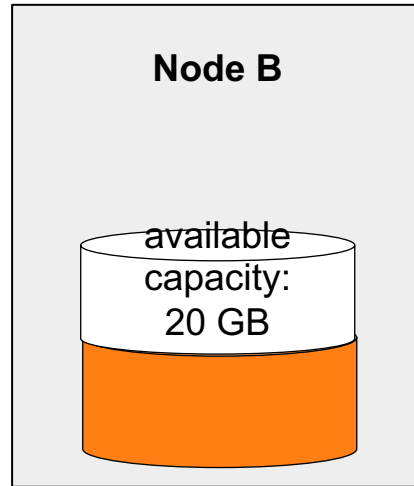
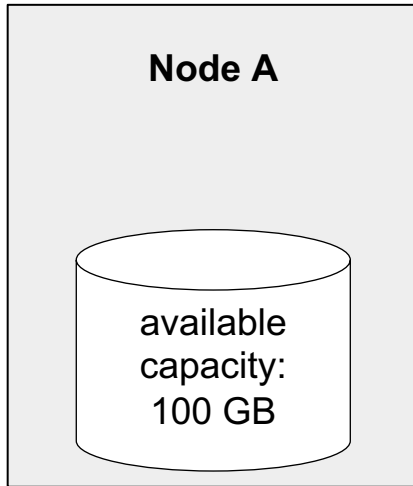
```
kind: StorageClass
provisioner: driver.longhorn.io
volumeBindingMode: WaitForFirstConsumer
parameters:
  numberOfReplicas: "1"
  dataLocality: "best-effort"
  v2DataEngine: true
```

- disabled
 - Replicas are placed independently of the workload node
- **best-effort**
 - Longhorn tries to place one replica on the workload node
- strict-local
 - Longhorn places the only one replica on the workload node

Pod Scheduling Workflow

- `volumeBindingMode: WaitForFirstConsumer`
- `dataLocality: "best-effort"`

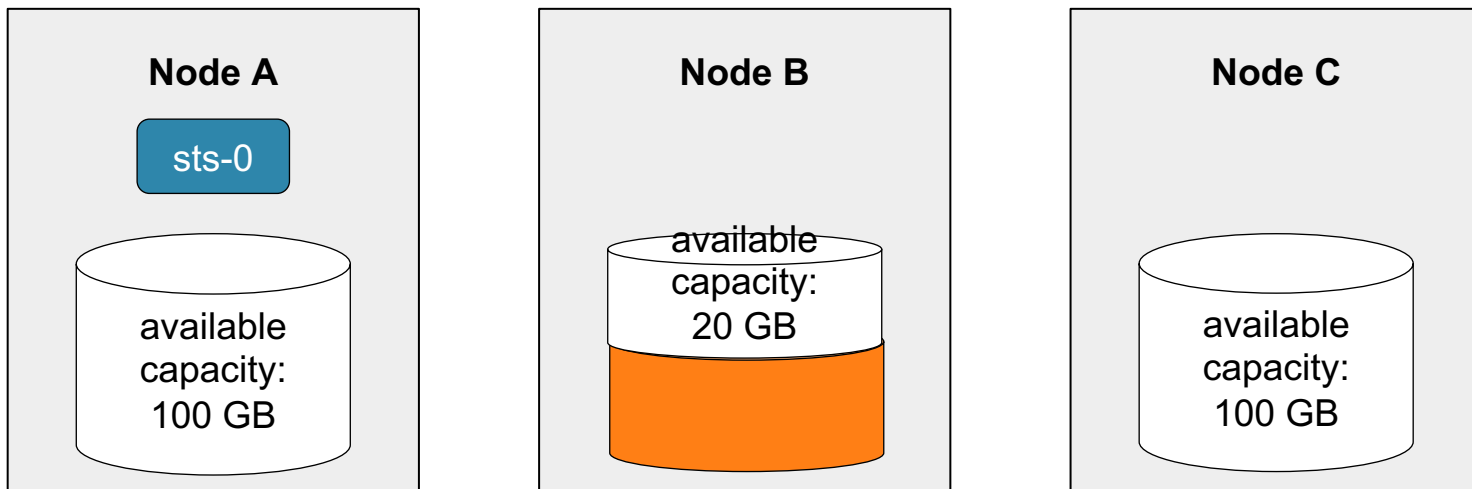
1. Pod **sts-0** with **40 GB** PVC



Pod Scheduling Workflow

- `volumeBindingMode: WaitForFirstConsumer`
- `dataLocality: "best-effort"`

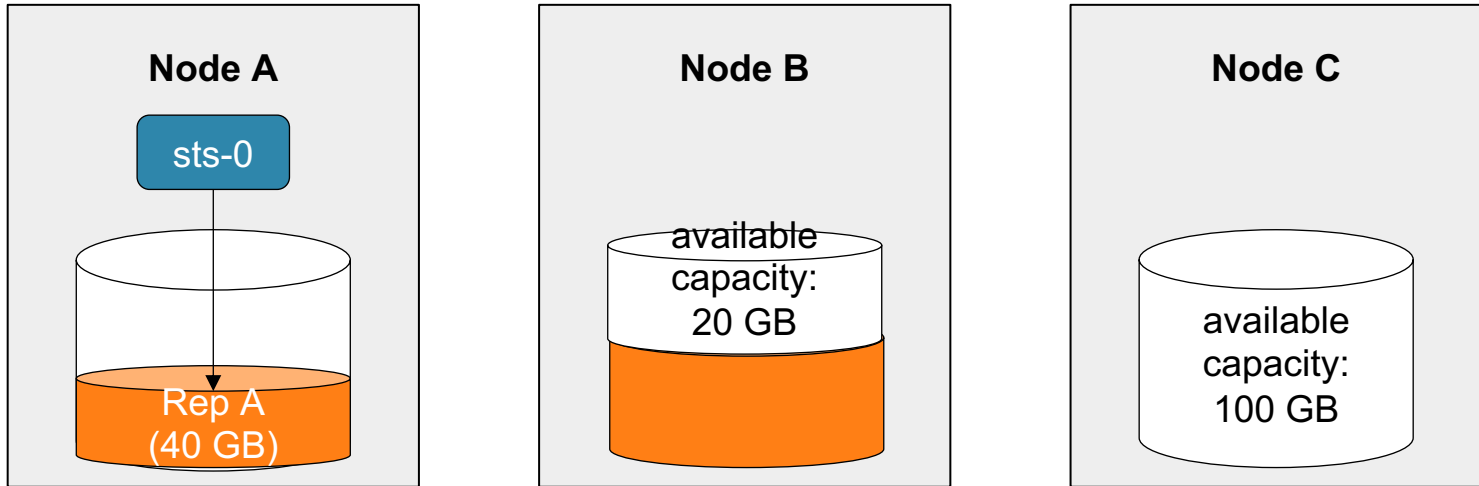
1. Pod **sts-0** with **40 GB** PVC



Pod Scheduling Workflow

- `volumeBindingMode: WaitForFirstConsumer`
- `dataLocality: "best-effort"`

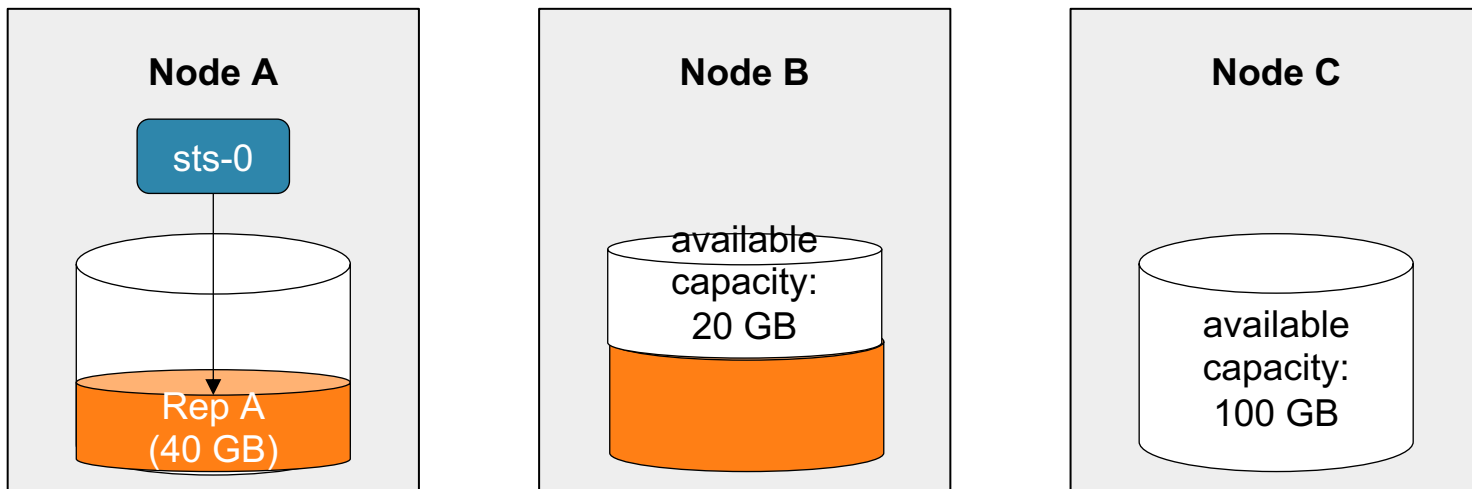
1. Pod **sts-0** with **40 GB** PVC



Pod Scheduling Workflow

- `volumeBindingMode: WaitForFirstConsumer`
- `dataLocality: "best-effort"`

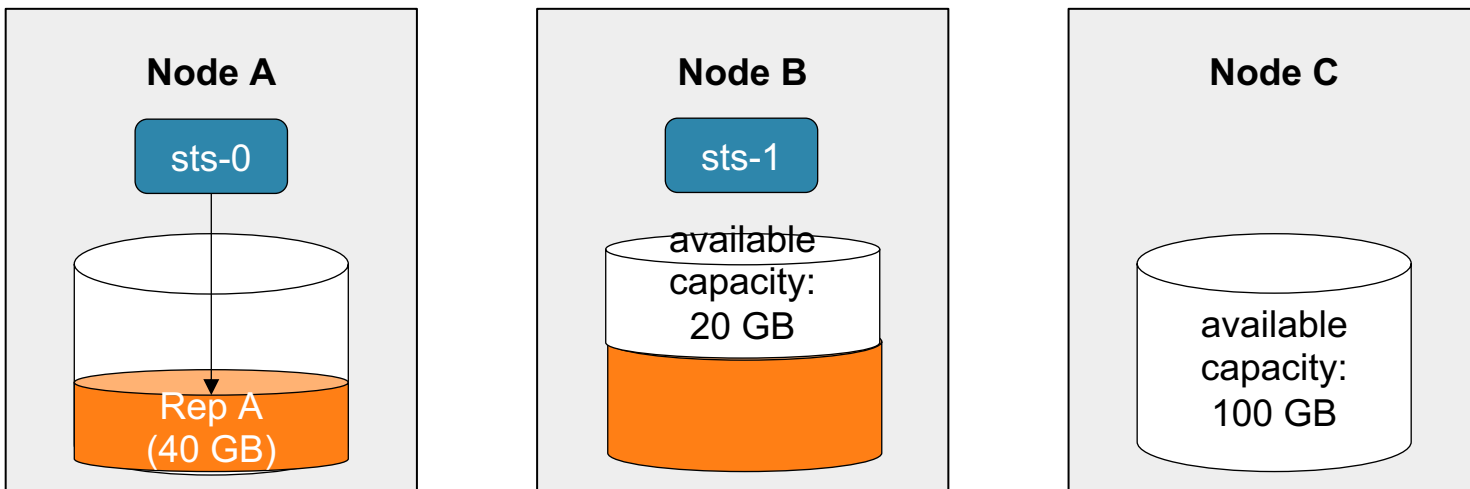
2. Pod **sts-1** with **40 GB** PVC



Pod Scheduling Workflow

- `volumeBindingMode: WaitForFirstConsumer`
- `dataLocality: "best-effort"`

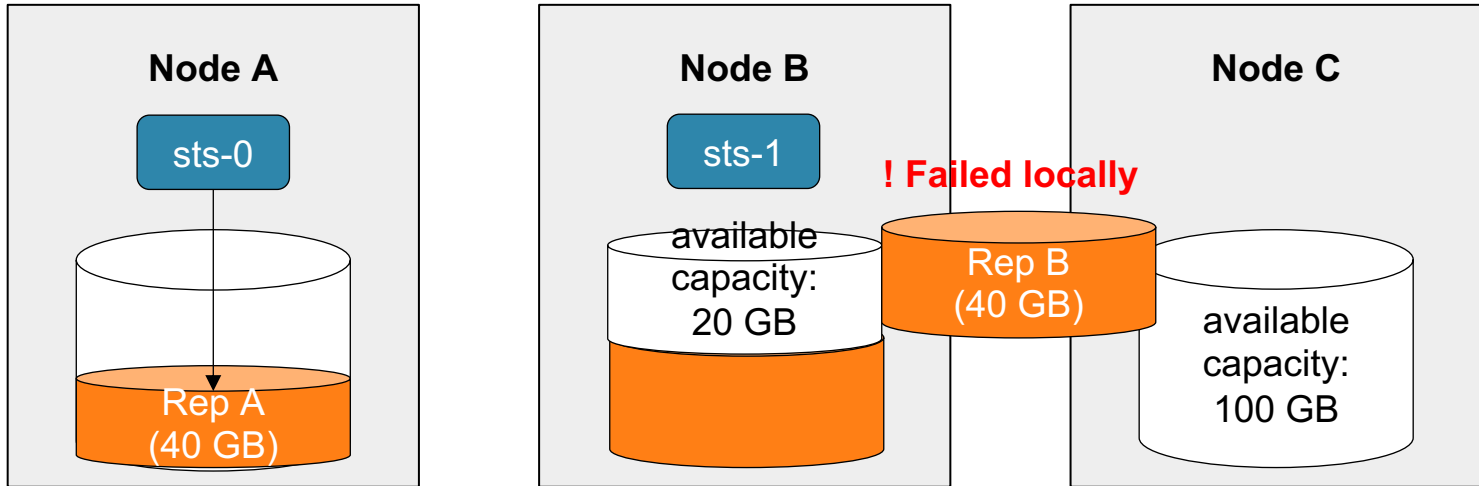
2. Pod **sts-1** with **40 GB** PVC



Pod Scheduling Workflow

- `volumeBindingMode: WaitForFirstConsumer`
- `dataLocality: "best-effort"`

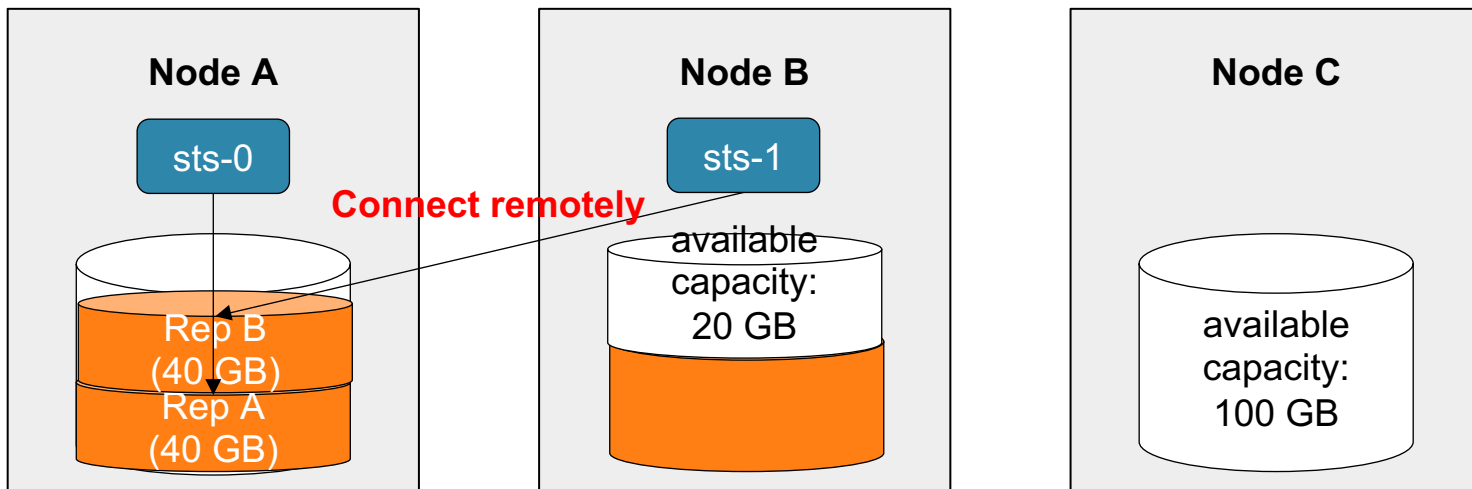
2. Pod **sts-1** with **40 GB** PVC



Pod Scheduling Workflow

- `volumeBindingMode: WaitForFirstConsumer`
- `dataLocality: "best-effort"`

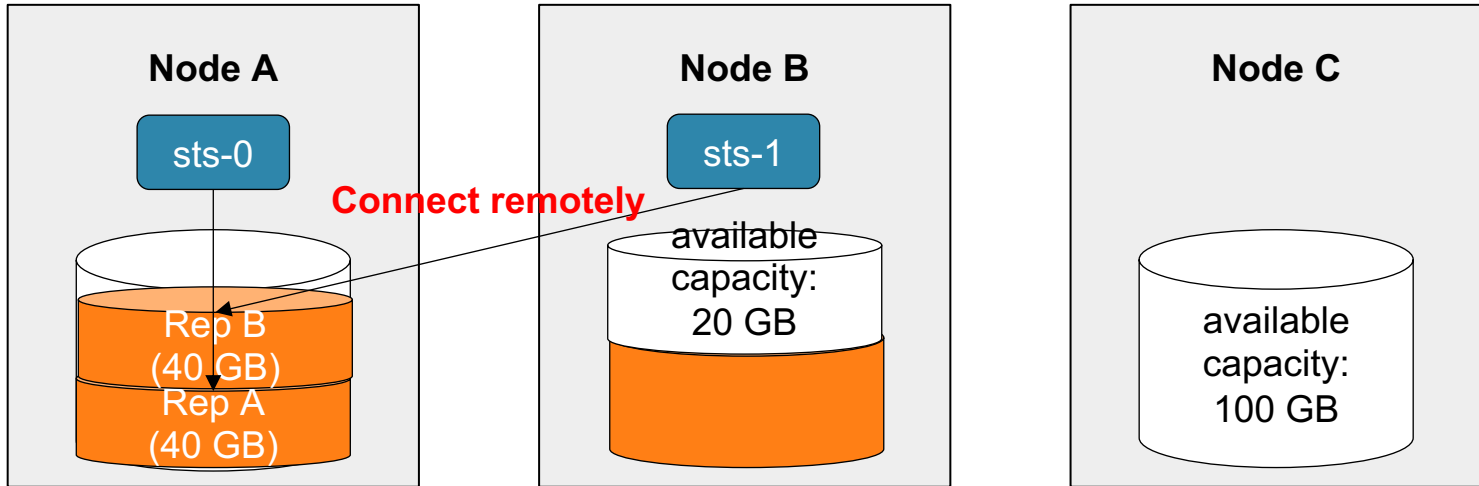
2. Pod **sts-1** with **40 GB** PVC



Pod Scheduling Workflow

Limitation: kube-scheduler is not storage-capacity-aware

Remote replica access can undermine StatefulSet availability.

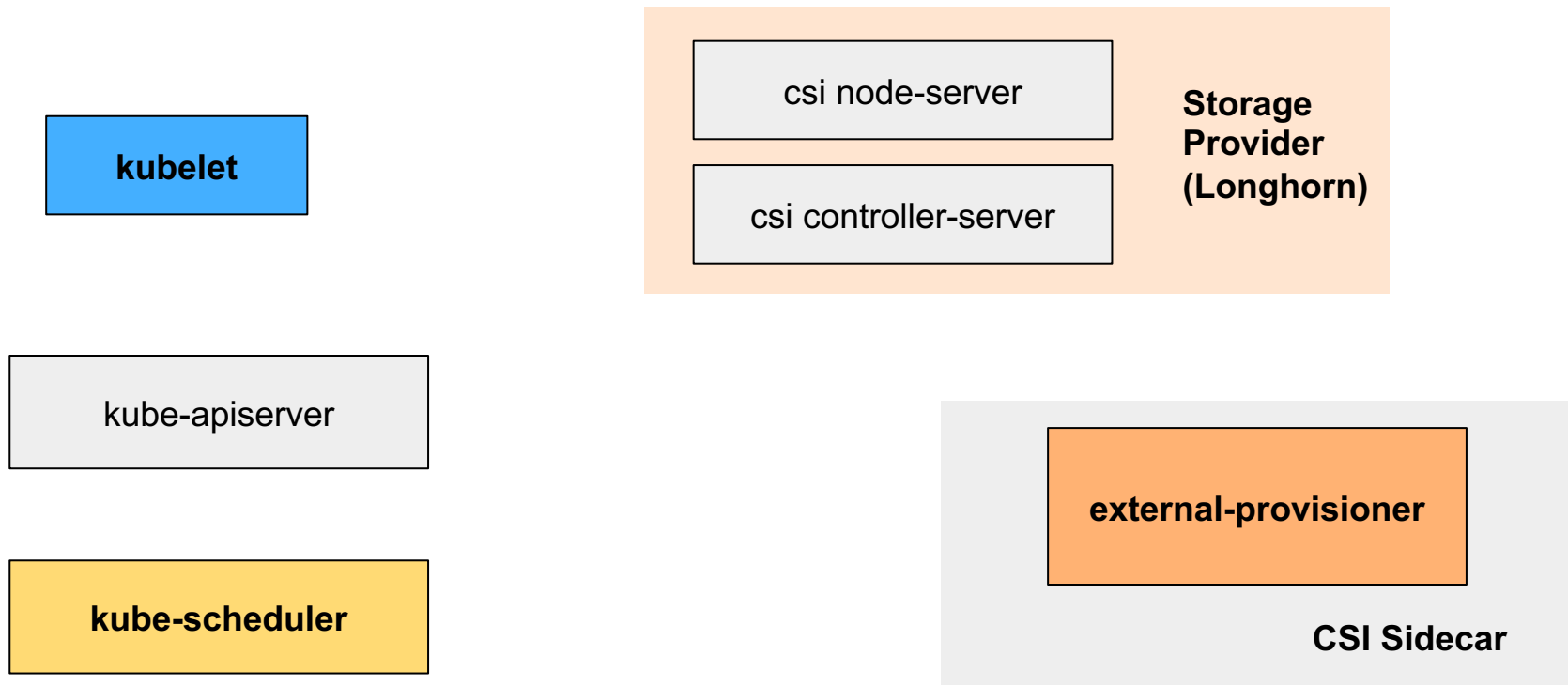


CSIStorageCapacity

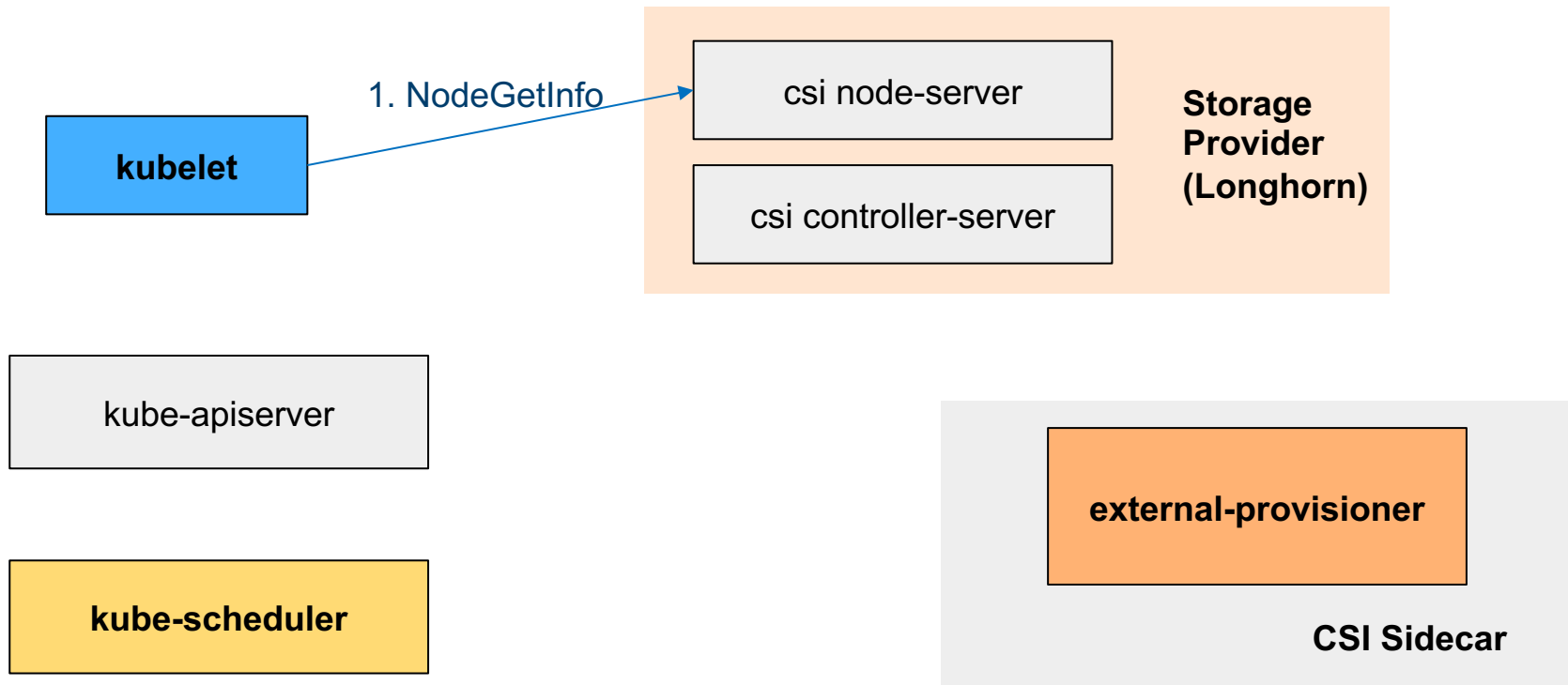
```
kind: CSIStorageCapacity
storageClassName: longhorn
nodeTopology:
  matchExpressions:
    - key: kubernetes.io/hostname
      values: ["node-1"]
capacity: "100Gi"
```

- storageClass - which SC this capacity belongs to
- nodeTopology – which node(or zone) this capacity applies to
- Capacity – available storage reported by **GetCapacity** RPC

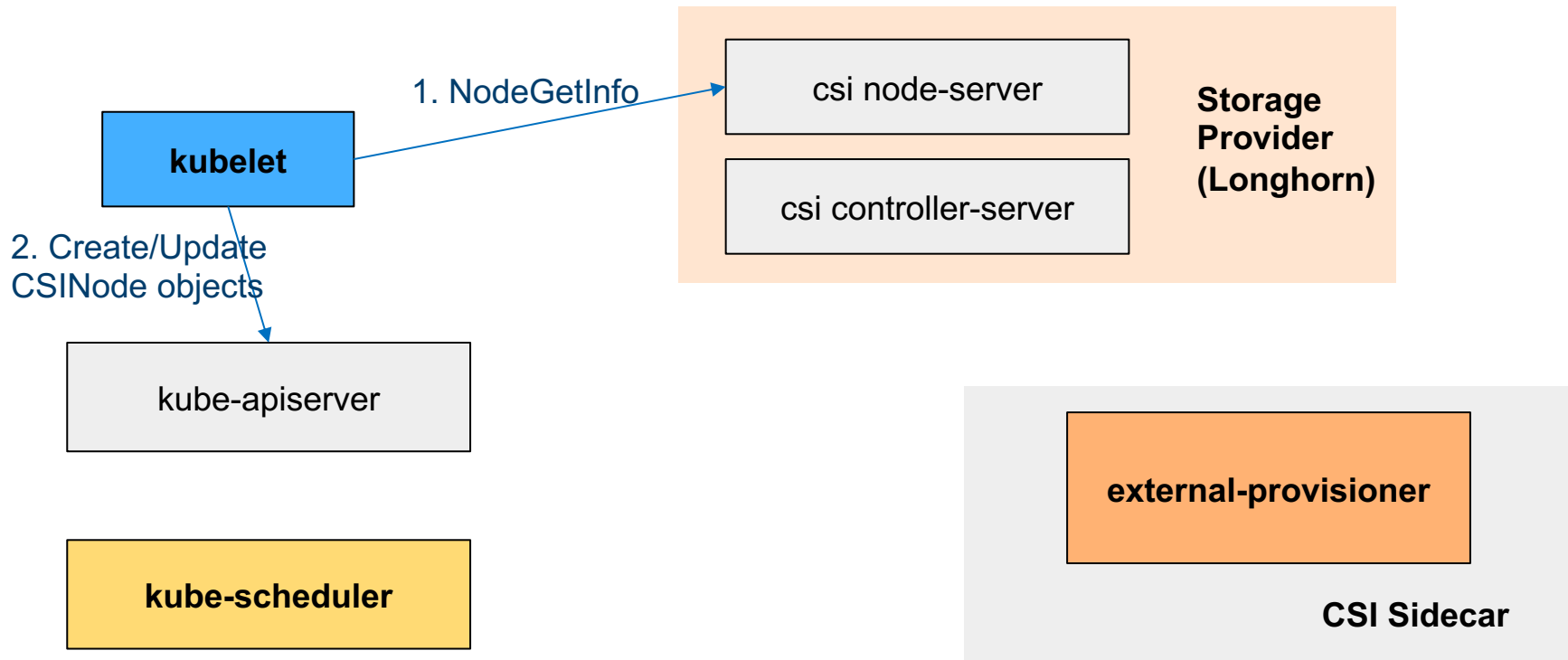
CSIStorageCapacity Flow



CSIStorageCapacity Flow

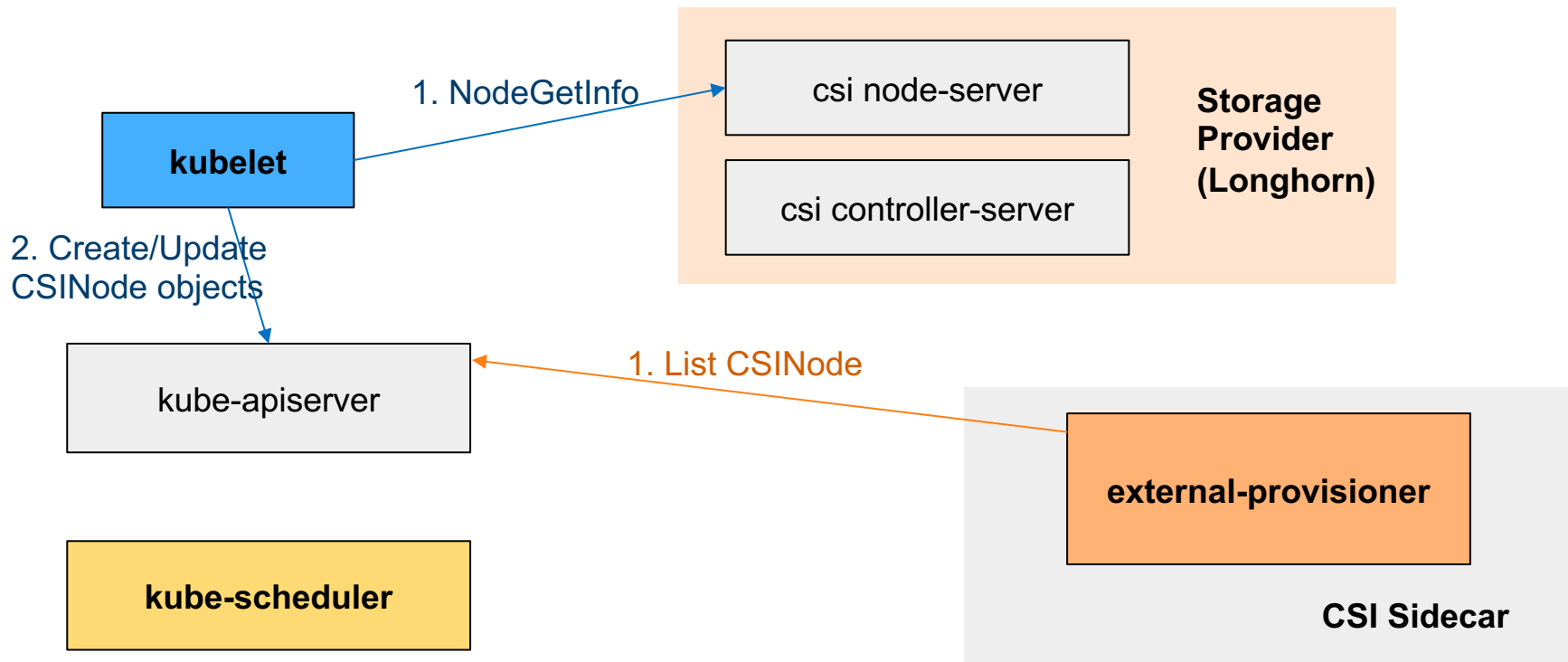


CSIStorageCapacity Flow

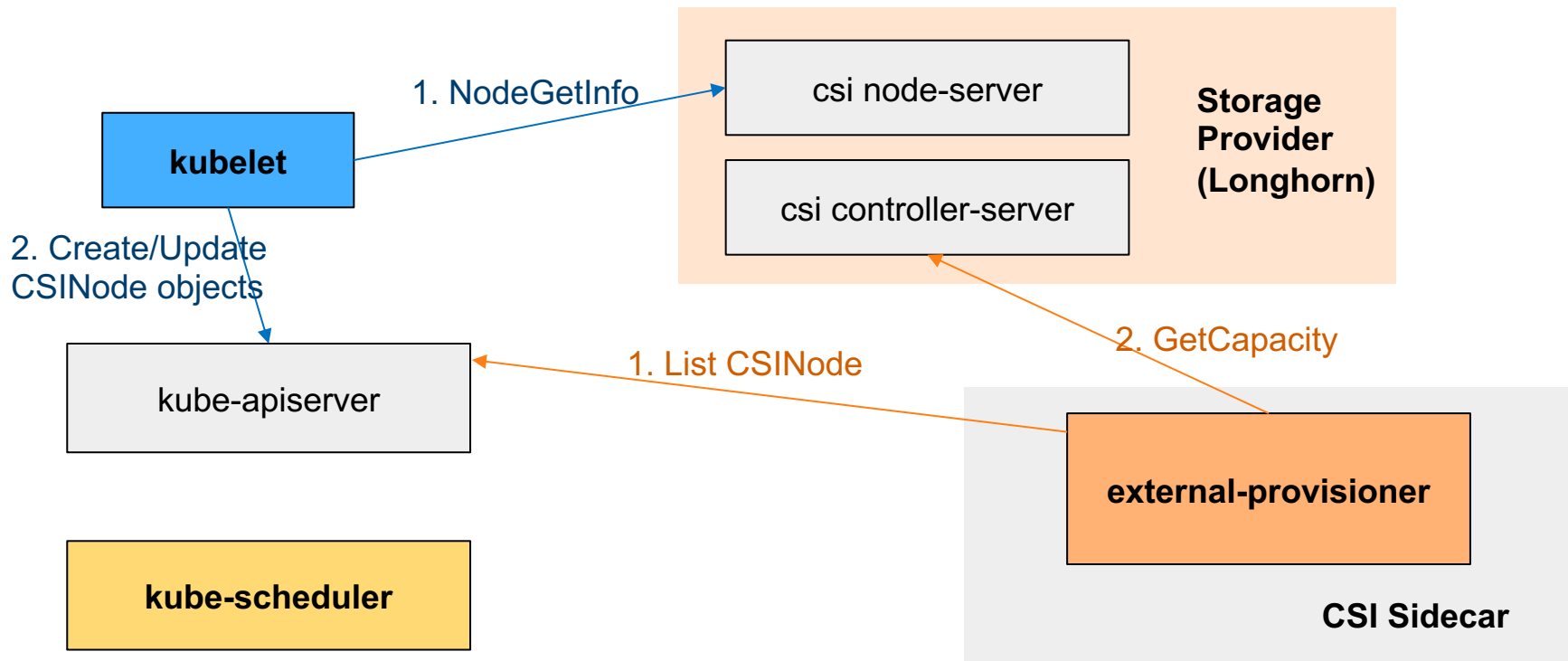


*KEP-1472: Storage Capacity Constraints for Pod Scheduling

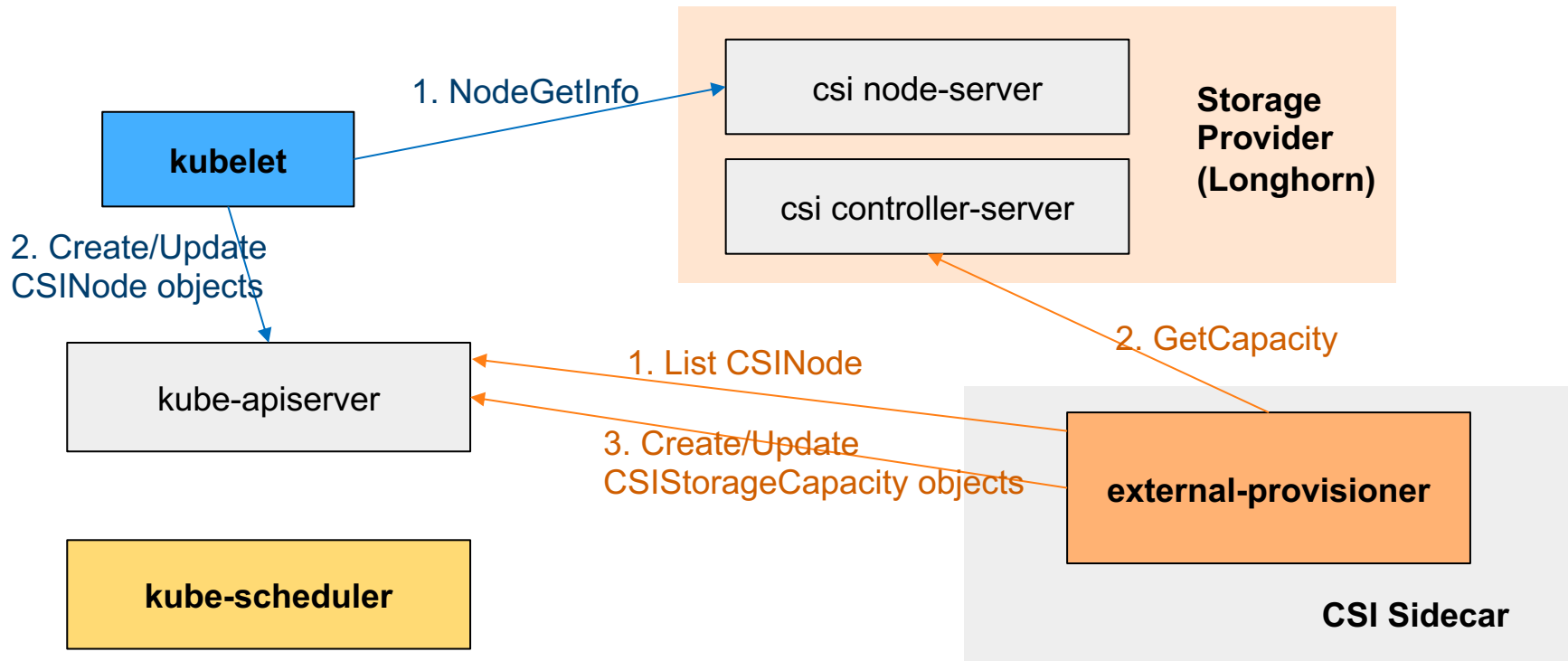
CSIStorageCapacity Flow



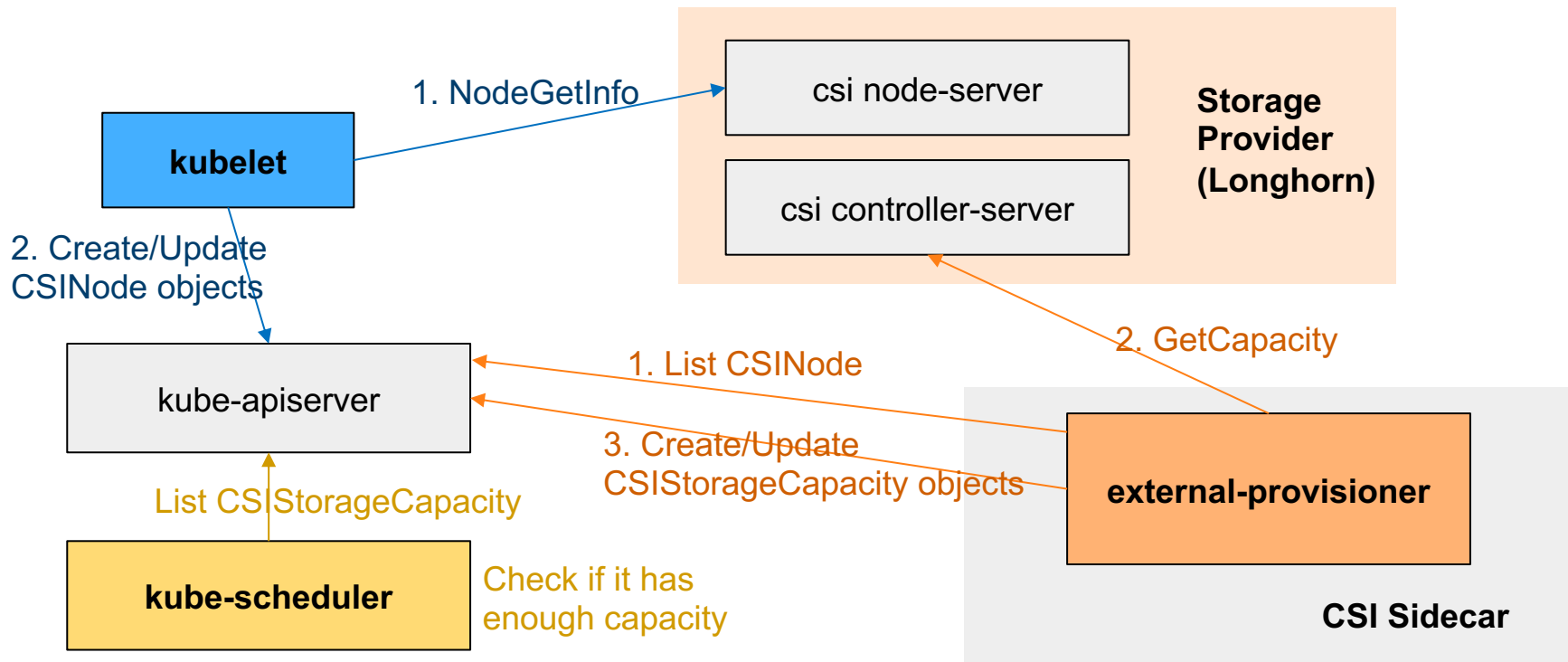
CSIStorageCapacity Flow



CSIStorageCapacity Flow



CSIStorageCapacity Flow

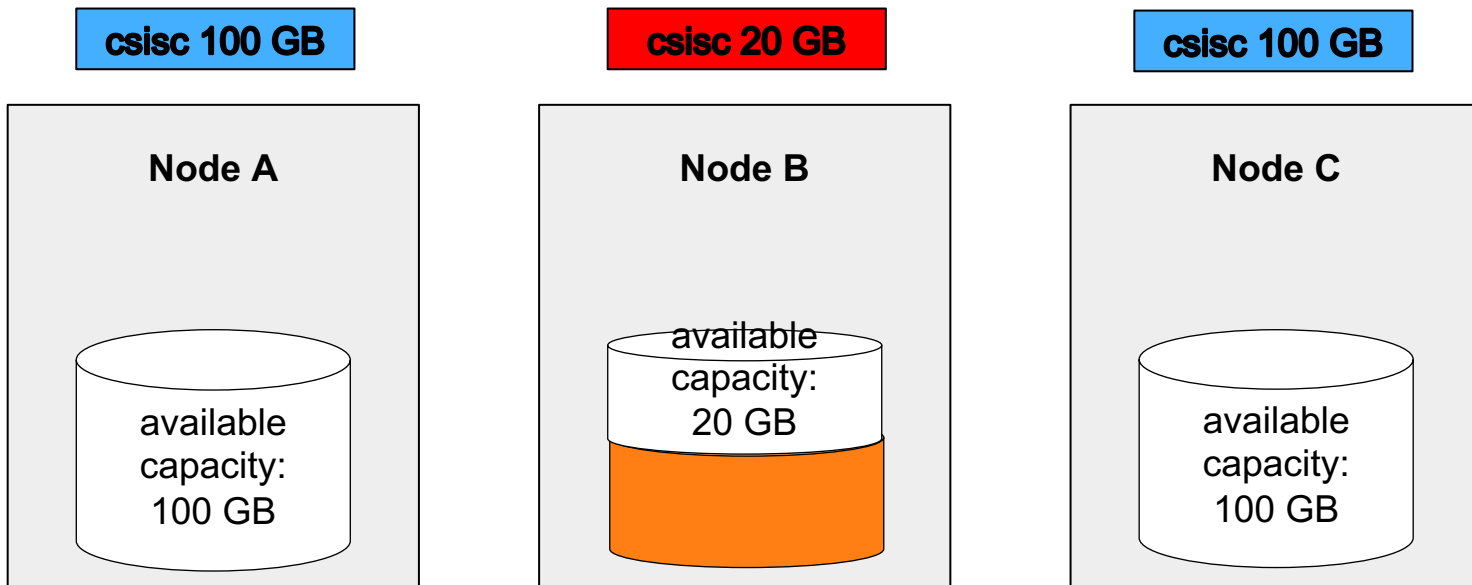


GetCapacity()

```
function GetCapacity(nodeID):  
    ## STEP 1: Node Validation  
  
    node = Fetch node from Longhorn API  
  
    IF node is NOT Ready/Schedulable → return 0  
    IF scheduling disabled OR Eviction → return 0  
    IF node.Tags  $\nrightarrow$  nodeSelector → return 0  
  
    ## STEP 2: Per-Disk Capacity Calculation & Aggregation  
    capacity = 0  
    FOR EACH disk IN node.disks:  
        SKIP IF scheduling disabled / Eviction / unhealthy / tag mismatch  
        schedulable = (disk.Max - disk.Reserved) × OverProvision% / 100 - disk.Scheduled  
        capacity = MAX(capacity, schedulable) ← not SUM, takes MAX  
  
    RETURN capacity
```

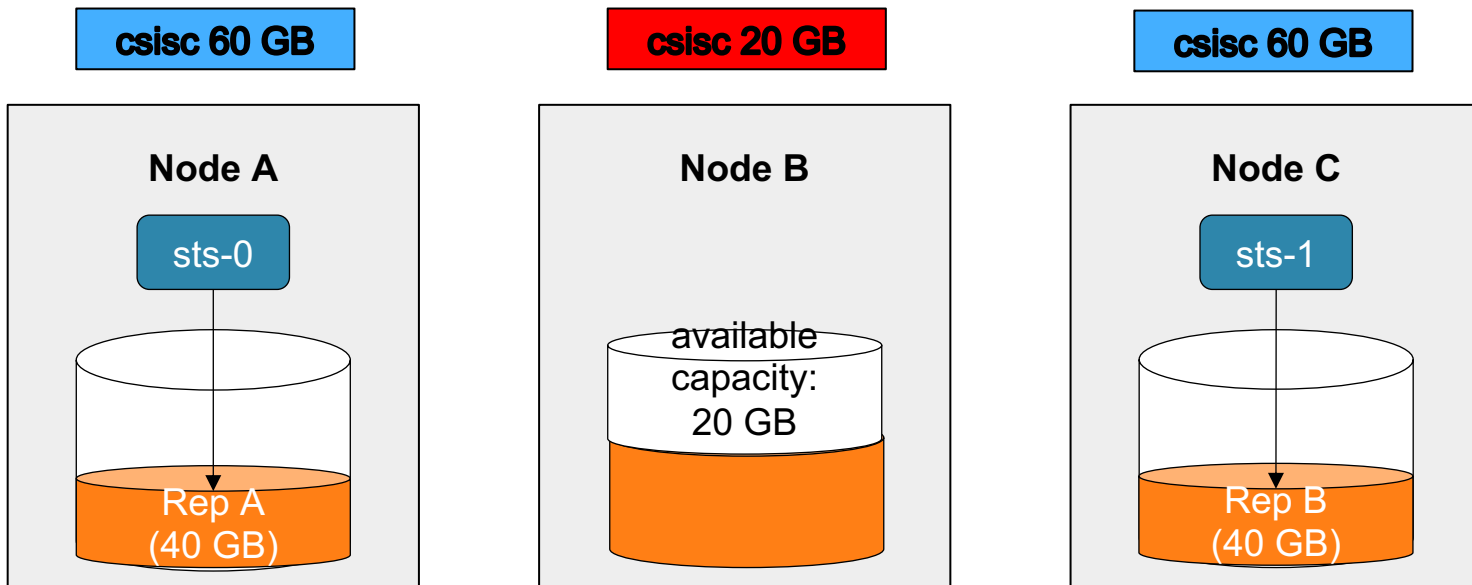
Pod Scheduling Workflow(/w csisc)

kube-scheduler filters out nodes with insufficient storage capacity



Pod Scheduling Workflow(/w csisc)

kube-scheduler filters out nodes with insufficient storage capacity



Limitations of CSIStorageCapacity

Once the PVC and PV are Bound,
subsequent Pod scheduling does not consider CSIStorageCapacity

- Case 1: Node Drain
 - Pod relocated without considering storage capacity
- Case 2: Pod Restart
 - Pod scheduled regardless of current Replica location
 - Unnecessary data rebuilding occurs

Our Approach: Scheduler Framework

Pod scheduling needed to consider where replicas already exist,
not just storage capacity

- **Why this works**
 - Uses Longhorn state during Pod scheduling
 - Can intervene at multiple scheduling stages (Filter, Score, Reserve, Bind)
- **Trade-offs**
 - Requires version-specific scheduler builds
 - Adds scheduler operational overhead

Our Solution: Scheduling Policy (WIP)

1. Check whether the Pod has an **existing** replica
2. Prefer the **replica node** first
3. If **unschedulable**, fall back to other nodes
4. In fallback or initial placement, filter out nodes with insufficient storage capacity

Conclusion

- Key design considerations for hyperconverged storage
- Storage-aware scheduling for hyperconverged environments



KubeCon



CloudNativeCon

Europe 2026



Thank you

Jangseon Ryu – jangseon.ryu@navercorp.com

Jinhong Kim – jinhong.kim0@navercorp.com

