AUMOVIO

The Hitchhiker’s Guide to Kubernetes Scheduling and Orchestration

DON’T PANIC.

KCD Budapest 2026

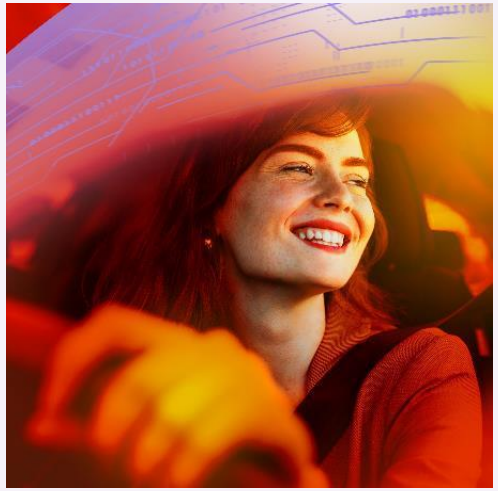
Máté Hajnal

AUMOVIO Autonomous Mobility Hungary

31 October 2026

Hitchhiking is awkward.

Two strangers. One car.
Absolutely no idea what could go wrong.



“Seems trustworthy.”



A few questions to establish trust...

Where are you going?

What’s your name?

How are you feeling?

Shall I open the trunk for your baggage?

A few questions to establish trust...

Where are you going?

What's your name?

How are you feeling?

Shall I open the trunk for your baggage?
Let me move the zip ties, duct tape and spade.



A few questions to establish trust...

Where are you going?

What's your name?

How are you feeling?

Shall I open the trunk for your baggage?
Let me move the zip ties, duct tape and spade.

What's your blood type, exactly?



A few questions to establish trust...

Where are you going?

What's your name?

How are you feeling?

Shall I open the trunk for your baggage?
Let me move the zip ties, duct tape and spade.

What's your blood type, exactly?

Do any illnesses or diseases run in your family?



The really risky part: introducing myself



Senior Machine Learning Operations Engineer at AUMOVIO

Golden Kubeastronaut

LFX Mentor

CNCF Volcano Core Contributor

Kubernetes scheduling and Distributed Systems

So I keep it simple.



~~Senior Machine Learning Operations Engineer at AUMOVIO~~

~~Golden Kubeastronaut~~

~~LFX Mentor~~

~~CNCF Volcano Core Contributor~~

~~Kubernetes scheduling and Distributed Systems~~

Software Engineer.

I help build technology for self-driving cars and trucks at AUMOVIO

Asking SI for guidance

The first attempt

I asked a higher intelligence to choose my Kubernetes scheduler.

7,500,000 tokens later...



Asking SI for guidance

The second attempt

Maybe I simply phrased the question incorrectly. Surely an AI can handle a retry.

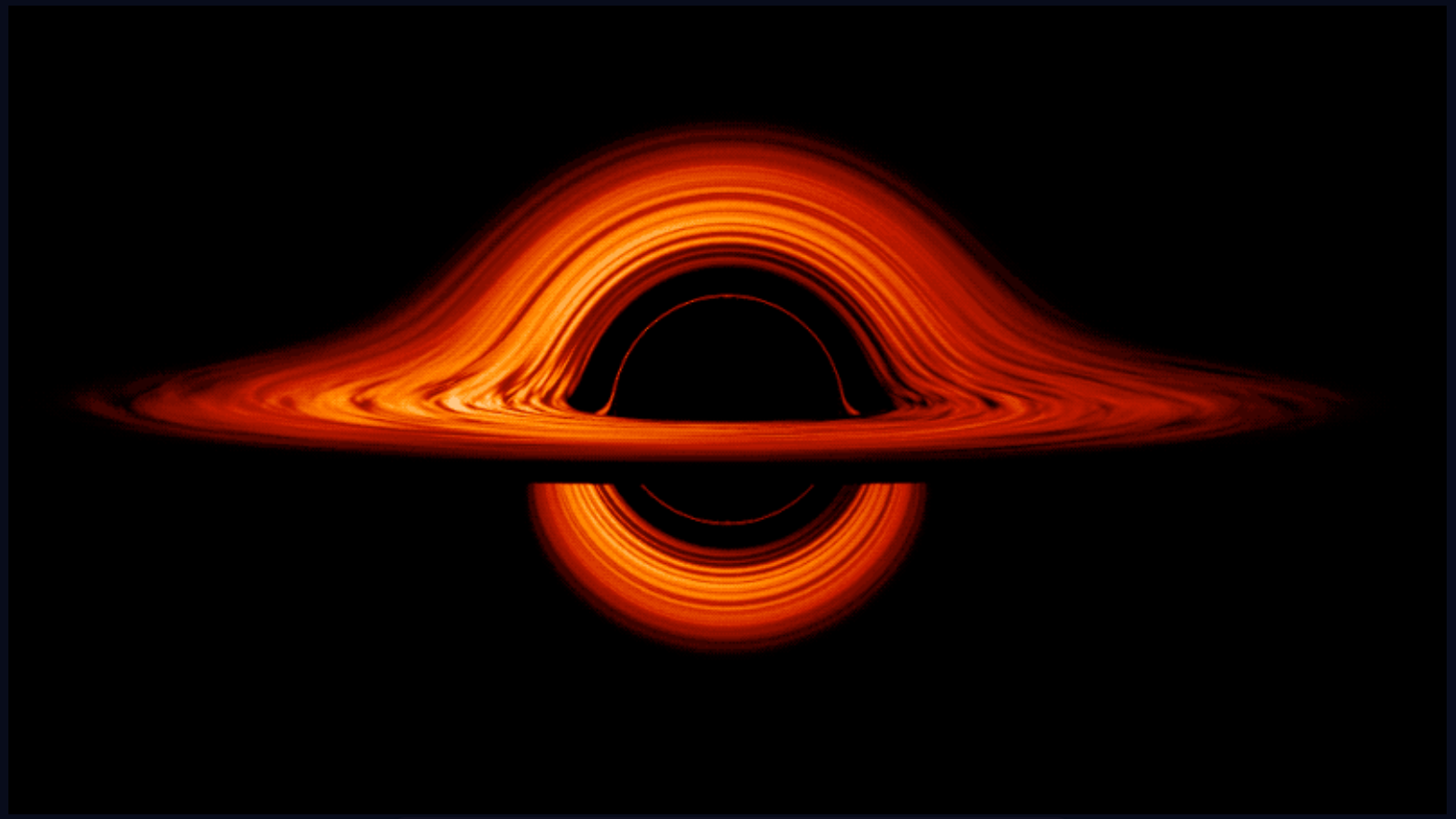
7,500,000 tokens later...






I'M GOING ON AN ADVENTURE!

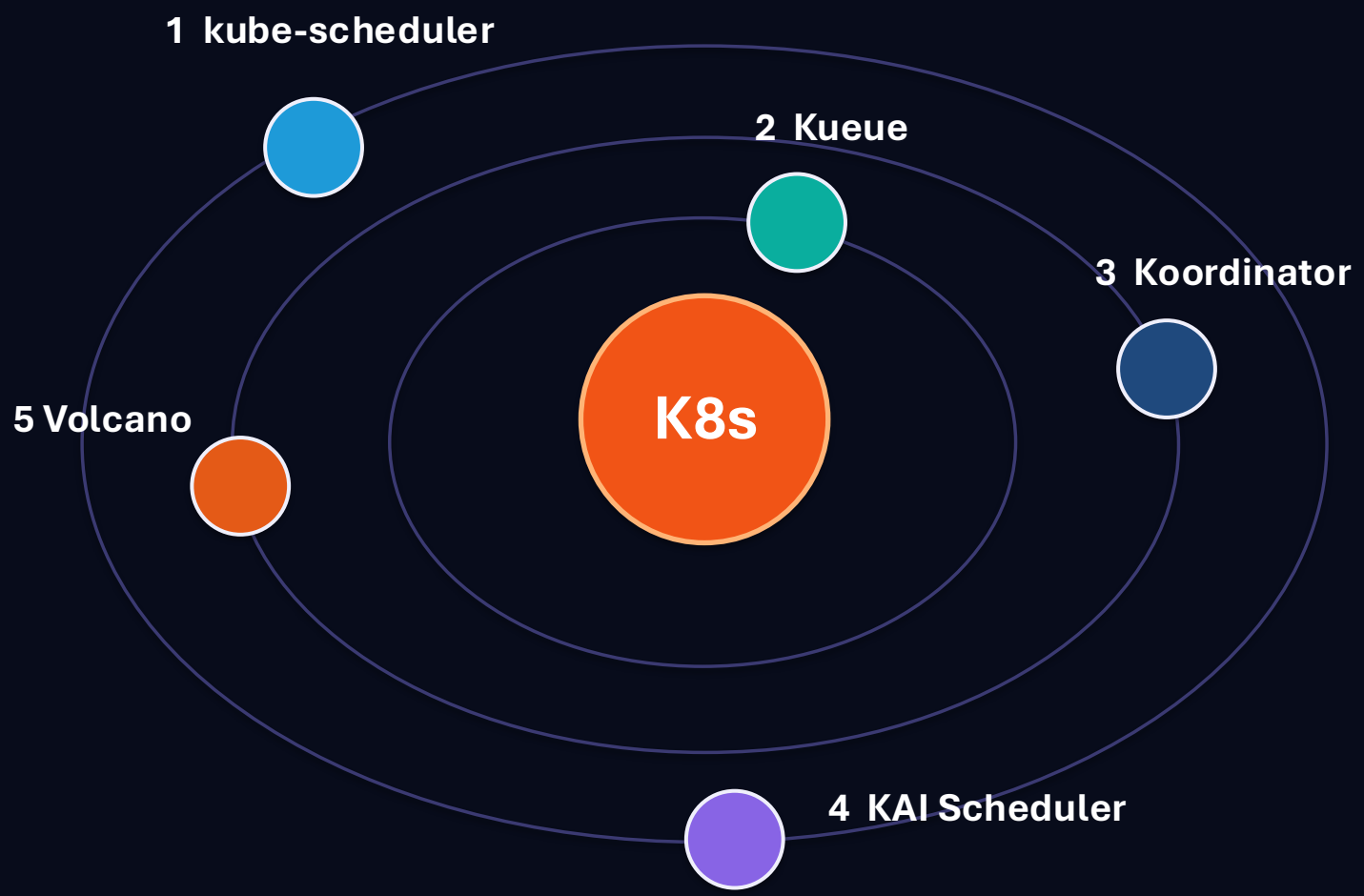
WELCOME TO THE KUBERNETES GALAXY



KUBERNETES

FIVE SOLAR SYSTEMS. ONE EXPEDITION.

Our planned route through the scheduling universe



UNCHARTED SYSTEMS

Apache YuniKorn
NVIDIA Run:ai
Fluid
HAMI

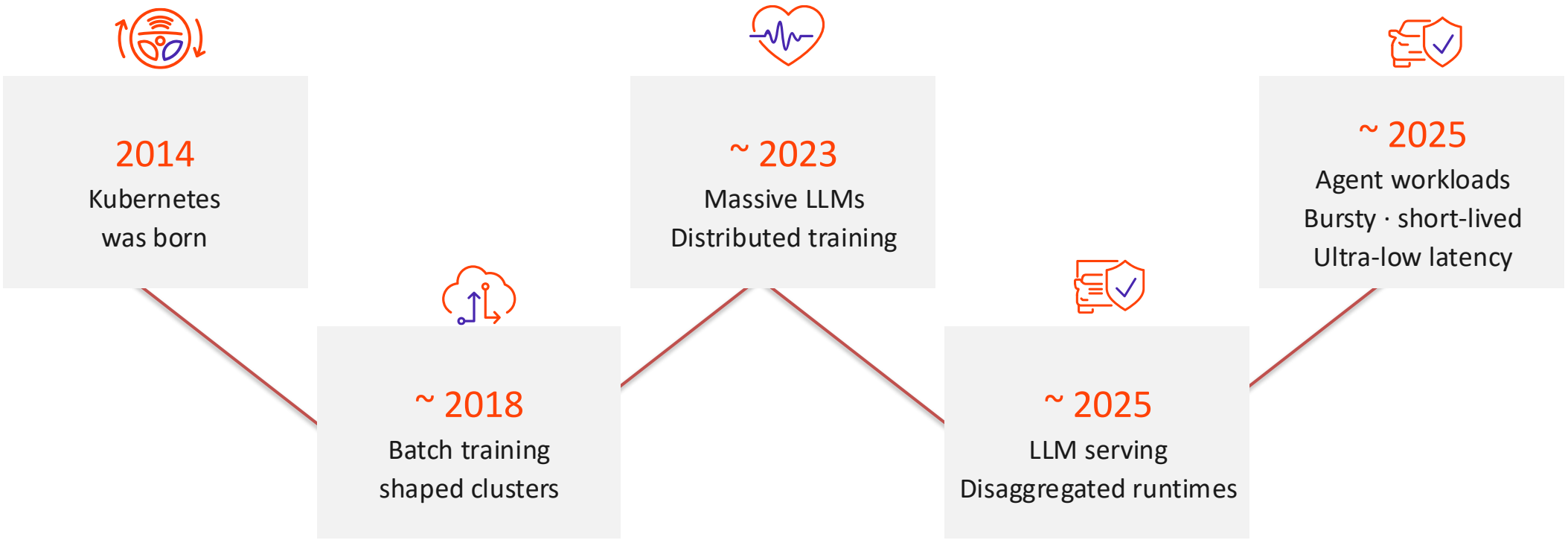
BEYOND ONE CLUSTER
MultiKueue
Karmada
Volcano Global
Cluster Autoscaler*
Karpenter
KubeAdmiral
Open Cluster Management

Or Deschedulers

* capacity provisioning, not a scheduler

The workload drift, Big Bang

From Kubernetes to agentic workloads



One control plane. Multiple execution patterns. Throughput AND latency.

Internal

OUR HITCHHIKER ARRIVES

Small request, big expectations.

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: hitchhiker
spec:
  schedulerName: default-scheduler
  # nodeName: worker-42
  containers:
  - name: ford-prefect
    image: nginx:alpine
    resources:
      requests:
        cpu: "500m"
        memory: 42Gb
# status:
#   nominatedNodeName: worker-42
```

```
Inspect: kubectl get pod hitchhiker -o yaml
```

The request

The scheduler uses declared requests when evaluating Node capacity — not the Pod's future actual utilization.

Everything else is a wrapper

Deployment, ReplicaSet, StatefulSet, DaemonSet, Job, CronJob, JobSet, LeaderWorkerSet, TrainJob, PyTorchJob, TFJob, MPIJob, JAXJob, XGBoostJob, RayJob, RayCluster, RayService, SparkApplication, FlinkDeployment, Argo Workflow, Tekton PipelineRun, KServe InferenceService, Knative Service, Agent Sandbox, AppWrapper, Volcano Job

Workload API — the itinerary, not the passengers

Kubernetes v1.35 Alpha → v1.36 redesigned → v1.37 Beta

```yaml
apiVersion: scheduling.k8s.io/v1beta1
kind: Workload
metadata:
  name: hitchhiker-training
spec:
  podGroupTemplates:
  - name: workers
    schedulingPolicy:
      gang:
        minCount: 4
```

Feature gate: GenericWorkload (off by default)

STATIC SCHEDULING TEMPLATE

Defines policy for a group of Pods. No individual Pod placement or runtime state.

```
gang.minCount: 4
```

Four seats required before departure.

PodGroup API — all passengers board together

Standalone runtime API since v1.36 · Beta in Kubernetes v1.37 (26 August 2026)

```yaml
apiVersion: scheduling.k8s.io/v1beta1
kind: PodGroup
metadata:
  name: hitchhiker-workers
spec:
  schedulingPolicy:
    gang:
      minCount: 4

# Each member Pod:
spec.schedulingGroup.podGroupName:
  hitchhiker-workers
```

Feature gate: GenericWorkload (off by default)

RUNTIME SCHEDULING UNIT

Scheduler evaluates the gang together.
If fewer than 4 Pods can fit, no Pod is bound.



All-or-nothing placement

CompositePodGroup API – 1.37 alpha

Agent Sandbox — a home for autonomous travelers

New workload API · Kubernetes SIG Apps · CRD, not a core Pod kind

sandbox.yaml

```yaml
apiVersion: agents.x-k8s.io/v1beta1
kind: Sandbox
metadata:
  name: hitchhiker-agent
spec:
  podTemplate:
    spec:
      runtimeClassName: gvisor
      containers:
        - name: agent
          image: python:3.12-slim
          command: ["sleep", "infinity"]
```

Why not a Deployment?

1 Singleton + stateful
Stable identity and persistent workspace

2 Isolation boundary
RuntimeClass: gVisor / Kata Containers

3 Agent lifecycle
Create, pause/resume, clean up; optional warm pools

Sandbox controller → Pod → kube-scheduler

FIRST STOP: THE DEFAULT SCHEDULER

A hitchhiker needs a ride. A Pod needs a Node.

POD

Where can I go?

FILTER

Which Nodes are feasible?

SCORE

Which one is best?

BIND

Your ride has arrived.

A scheduler is a matchmaker with a very strict checklist.

Finding the right ride: placement constraints

hitchhiker-pod.yaml | spec excerpt

```
nodeSelector:
  kubernetes.io/arch: amd64
affinity:
  podAffinity:
    preferredDuringSchedulingIgnoredDuringExecution:
    - weight: 80
      podAffinityTerm:
        labelSelector:
          matchLabels: {zoey: deschanel}
        topologyKey: trillian
  podAntiAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
    - labelSelector:
        matchLabels: {app: hitchhiker}
      topologyKey: kubernetes.io/hostname
tolerations:
  - key: zaphod
    operator: Equal
    value: beeblebrox
    effect: NoSchedule
```

Four questions for the scheduler

01 nodeSelector

Hard node-label match: only amd64 nodes.

02 podAffinity

Prefer to ride with Pods labeled app=cache.

03 podAntiAffinity

Never share a node with app=hitchhiker Pods.

04 Toleration - taints

Sometimes the node is picky, and you need to tolerate it

Our hitchhiker gets picky about placement

Node affinity + topology spread control placement; a PDB protects availability during voluntary disruptions.

POD TEMPLATE • placement constraints

```yaml
metadata:
  labels: {app: hitchhiker}
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: kubernetes.io/arch
            operator: In
            values: [amd64]
          - key: nodepool
            operator: NotIn
            values: [spot]
  topologySpreadConstraints:
  - maxSkew: 1
    topologyKey: topology.kubernetes.io/zone
    whenUnsatisfiable: DoNotSchedule
    labelSelector:
      matchLabels: {app: hitchhiker}
  containers:
  - name: traveler
    image: nginx:alpine
```

PDB • separate API object

```yaml
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata:
  name: hitchhiker-pdb
spec:
  minAvailable: 2
  selector:
    matchLabels:
      app: hitchhiker
```

Key distinctions

NotIn = exclude matching node labels
maxSkew = maximum allowed imbalance

PDB ≠ scheduling constraint

Pod Pending for 47 minutes

The universe is full of cars. It can happen that none of them has a free seat.



Don’t Panic!

Check the Events

The scheduler does not invent capacity. It reports why none of the rides work.

SECOND STOP: KUEUE

Before picking a car, can you even start the journey?

NO TICKET?
NO DEPARTURE.

WORKLOAD ADMISSION

KUEUE: THE TICKET OFFICE

Logical quota before physical placement.



LocalQueue

Project-facing queue

ClusterQueue

Quota + ResourceFlavors

In a Cohort

Admitted Workload

Job can start

kube-scheduler

Place the Pods

Admission ≠ Pod placement

Control through labels and annotations naturally

The meta scheduler



```yaml
apiVersion: v1
kind: Pod
metadata:
  name: hitchhiker
  labels:
    kueue.x-k8s.io/queue-name: user-queue
    kueue.x-k8s.io/pod-group-name: "group-name"
  annotations:
    kueue.x-k8s.io/pod-group-total-count: "2"
spec:
  containers:
  - name: traveler
    image: nginx:alpine
    resources:
      requests:
        cpu: "500m"
        memory: 42Gb
```

Queue Name

Preemption control via a Local Queues

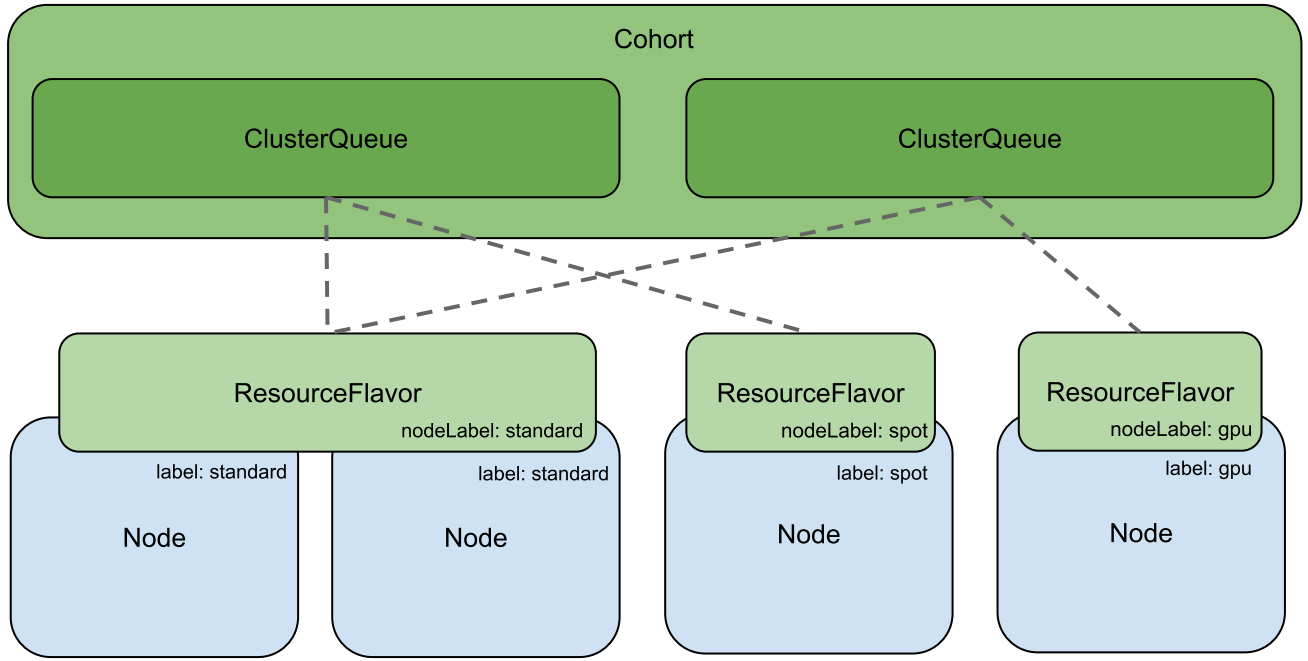
Special Workload API and Job management with Seperate CRDs

Support for a lot of workloads: BatchJob, Kubeflow training jobs, RayJob, RayCluster, JobSet, AppWrappers

From Cohorts to ResourceFlavors

Queues are the heart of Kueue

```
apiVersion: kueue.x-k8s.io/v1beta2
kind: ClusterQueue
metadata:
  name: "team-b-cq"
spec:
  namespaceSelector: {} # match all.
  cohortName: "team-ab"
  resourceGroups:
  - coveredResources: ["cpu"]
    flavors:
    - name: "default-flavor"
      resources:
      - name: "cpu"
        nominalQuota: 12
        lendingLimit: 1
  preemption:
    reclaimWithinCohort: Any
    borrowWithinCohort:
      policy: LowerPriority
      maxPriorityThreshold: 100
```



THIRD STOP: KOORDINATOR

More useful work per node — without sacrificing critical services.



SAME CLUSTER.

DIFFERENT PRIORITIES.

CLOUD-NATIVE CO-LOCATION

QoS: Kubernetes vs. Koordinator

Kubernetes answers resource guarantees; Koordinator adds co-location execution policies



Kubernetes QoS

Guaranteed

Requests = limits for CPU & memory

Burstable

Some resources requested / limited

BestEffort

No CPU or memory requests/limits

co-location semantics

Koordinator QoS

SYSTEM	System services; constrained use	—
LSE	Latency-sensitive, exclusive	Guaranteed
LSR	Latency-sensitive, reserved; bound CPU	Guaranteed
LS	Latency-sensitive, shared resources	Guaranteed / Burstable
BE	Low-cost batch; may be suppressed	BestEffort

Independent field: metadata.labels["koordinator.sh/qosClass"]

Internal

How Koordinator works

Scheduling choices in the control plane + QoS enforcement on every node



CONTROL PLANE

koord-scheduler

QoS / load-aware / quotas

koord-manager

Profiles / SLO controllers

koord-descheduler

Rebalance hot nodes

NODE PLANE

koordlet (DaemonSet)

Profiling · interference · isolation

Workloads on Kubernetes nodes

Latency-sensitive + batch + AI together

Internal

Beyond QoS: ElasticQuota and more

Shared-cluster fairness + operational efficiency



ElasticQuota

```
apiVersion:
scheduling.sigs.k8s.io/v1alpha1
kind: ElasticQuota
metadata:
  name: team-ai
  namespace: ml
spec:
  min:
    cpu: "8"
  max:
    cpu: "32"
```

Guaranteed floor → borrow unused capacity → upper limit

Other capabilities

Load-aware scheduling

Avoid hot spots from actual utilization

Gang & device scheduling

Workload cohorts, GPU/NUMA topology

Resource reservation

Reduce fragmentation and enable moves

Co-location + descheduling

Recover idle capacity, rebalance load

Koord-Queue

Job queuing integrated with ElasticQuota

Internal

FOURTH STOP: KAI SCHEDULER

Our hitchhiker demands a GPU-powered spaceship.

“I WOULD LIKE HALF A GPU, PLEASE.”

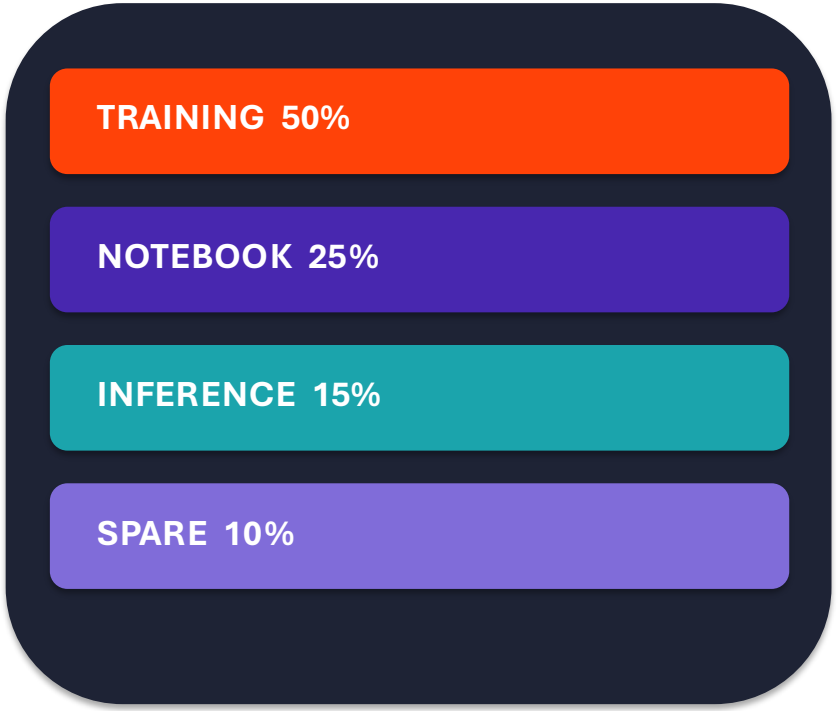
— Every AI developer, eventually



GPU-AWARE SCHEDULING

KAI: FIRST CLASS, ECONOMY, AND SHARED SEATS

GPU sharing makes the resource model more complicated.



Scheduling fraction

Placement and accounting can allocate a portion of a GPU to a workload.

Enforcement matters

GPU memory limits and compute-sharing semantics depend on configured runtime integrations.

Fractional allocation ≠ guaranteed performance isolation.

KAI: A FRACTIONAL-GPU HITCHHIKER

A deliberately simplified workload example.



```yaml
apiVersion: v1
kind: Pod
metadata:
  name: half-gpu-hitchhiker
  labels:
    kai.scheduler/queue: default-queue
  annotations:
    gpu-fraction: "0.5"
spec:
  schedulerName: kai-scheduler
  containers:
  - name: traveler
    image: nvidia/cuda:13.0.2-base-ubi8
    command: ["nvidia-smi"]
```

Mode-dependent

Queue support with RunAI CRDs

This example uses legacy fraction annotation supported by relevant KAI GPU-sharing modes. Cluster setup and supported modes are required.

One car. Two hitchhikers. What could go wrong?

FIFTH STOP: VOLCANO

The first batch scheduler

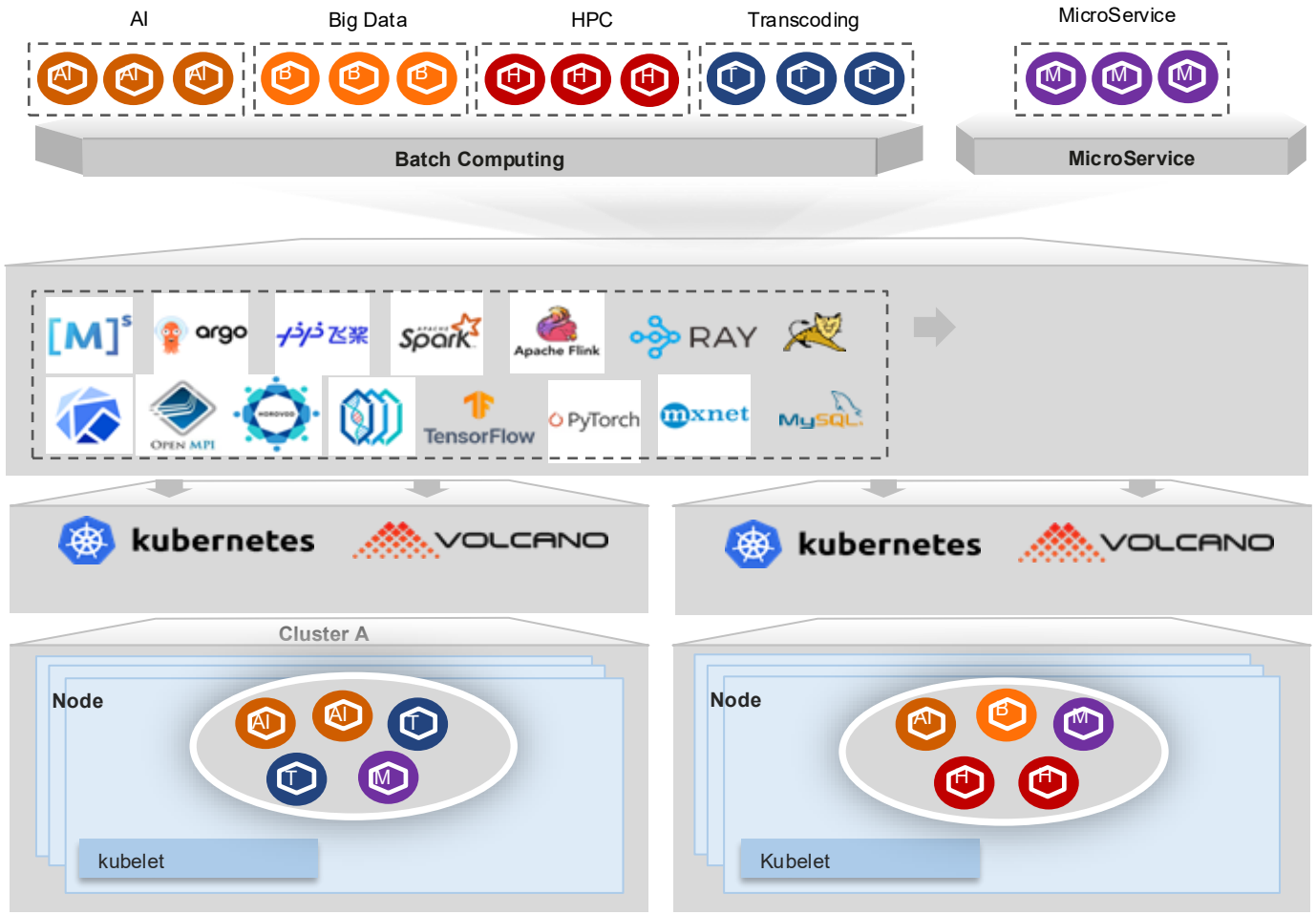
ONE SEAT IS EASY.
42 SEATS AT ONCE?



GANG SCHEDULING

Volcano is Versatile

Throw me anything, I am pan dimensional mice



01 Unified Scheduling

Native workloads & Mainstream Framework full support

02 Colocation

New QoS, enables online and offline workloads

03 Queue Management

Multi Level Queue management & Namespaces Queues

04 Lots of Scheduling Policies

Binback, Gang, Numa-Aware, DeviceShare, Fair-share

05 Heterogeneous Device Scheduling

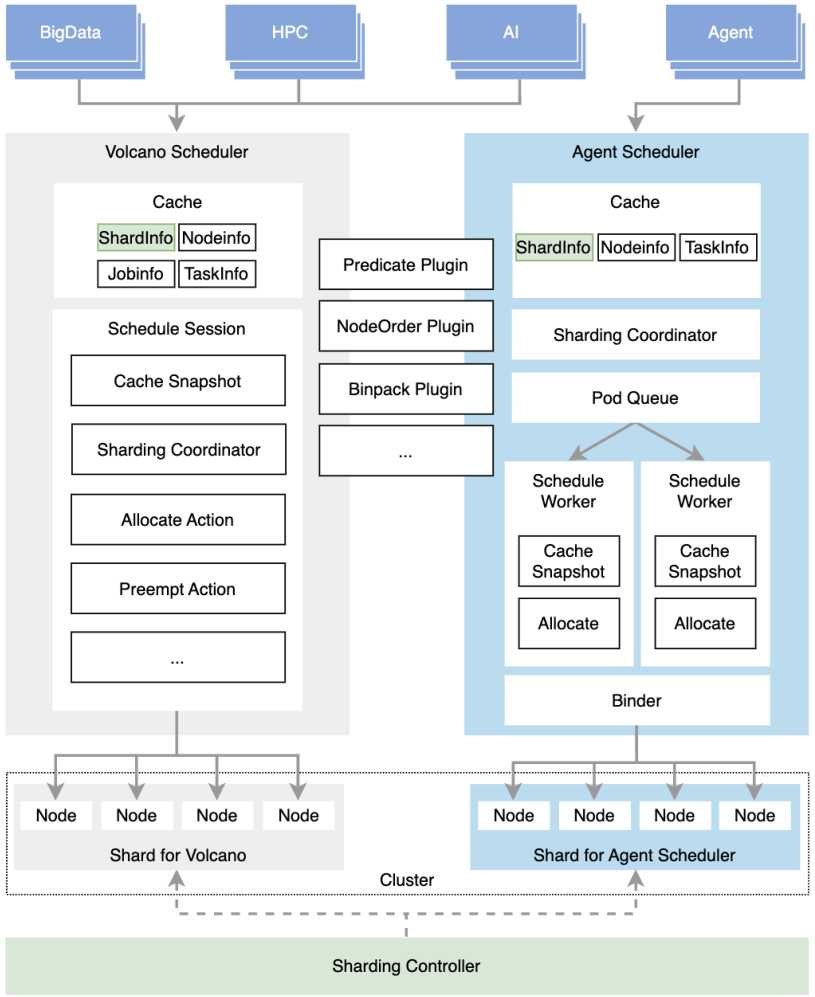
GPU, NPU, TPU support

06 Fractional GPU scheduling

With extensive support for HAMI

Volcano Agent Scheduler

Fast scheduling for latency-sensitive agent workloads



Fast path

Parallel scheduling

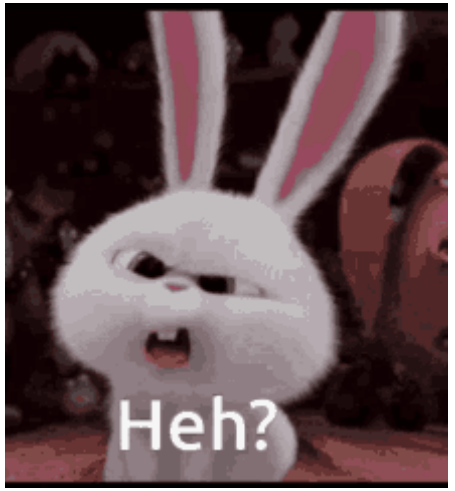
Workers handle independent Pod decisions.

Optimistic concurrency

Version-based checks avoid stale bindings.

Dynamic sharding

Candidate Nodes can be redistributed.



NodeShard CRD

A coordination contract between scheduling paths



Dynamic candidate Node pools

nodeshard.yaml

```yaml
apiVersion: shard.volcano.sh/v1alpha1
kind: NodeShard
metadata:
  name: volcano
spec:
  nodesDesired:
    - node1
    - node2
status:
  nodesInUse:
    - node1
    - node0
  nodesToRemove:
    - node0
  nodesToAdd:
    - node3
```

sharding.yaml

```yaml
sharding.yaml: |
   schedulerConfigs:
     - name: agent-scheduler
       type: agent
       cpuUtilizationMin: 0.7
       cpuUtilizationMax: 1.0
       preferWarmupNodes: true
       minNodes: 1
       maxNodes: 100
     - name: volcano
       type: volcano
       cpuUtilizationMin: 0.0
       cpuUtilizationMax: 0.69
       preferWarmupNodes: false
       minNodes: 1
       maxNodes: 100
```

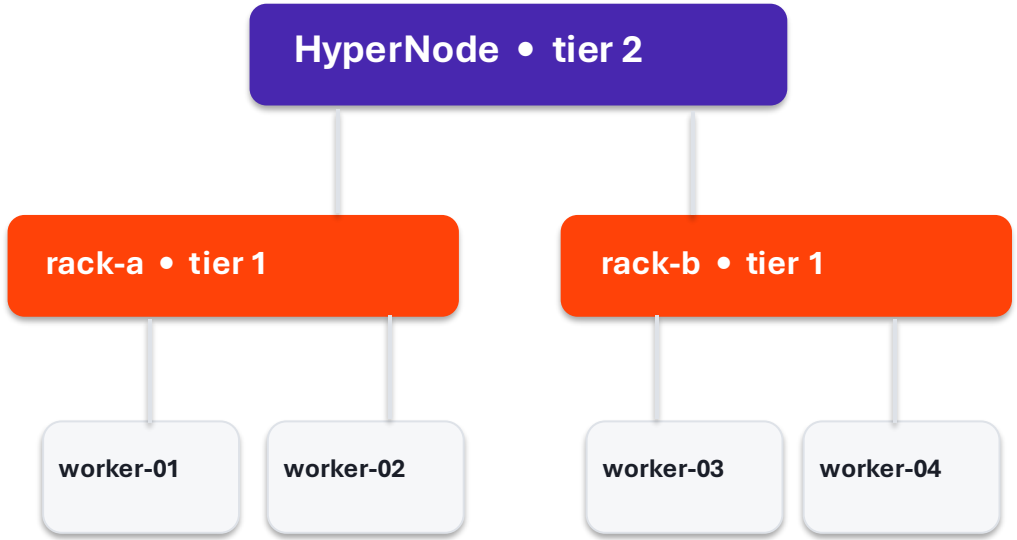
HyperNode — network topology as an API

Hierarchical performance domains for network-aware placement

hypernode.yaml

```yaml
apiVersion: topology.volcano.sh/v1alpha1
kind: HyperNode
metadata:
  name: rack-a
spec:
  tier: 1
  tierName: rack
  members:
    - type: Node
      selector:
        exactMatch:
          name: worker-01
    - type: Node
      selector:
        exactMatch:
          name: worker-02
```

A tree of performance domains



Lower tier = tighter network locality

Autodiscovery

Discovery may maintain the same CRDs automatically.

PodGroup topology anti-affinity

Keep each gang together — separate matching PodGroups across HyperNodes

podgroup.yaml • complete PodGroup resource

```yaml
apiVersion: scheduling.volcano.sh/v1beta1
kind: PodGroup
metadata:
  name: inference-a
  namespace: default
  labels:
    app: model-a
spec:
  minMember: 2
  queue: default
  networkTopology:
    mode: hard
    highestTierName: rack
  topologyAffinity:
    podGroupAntiAffinity:
      required:
        - podGroupSelector:
            matchLabels:
              app: model-a
          topologyTierName: rack
```

Two complementary policies

networkTopology

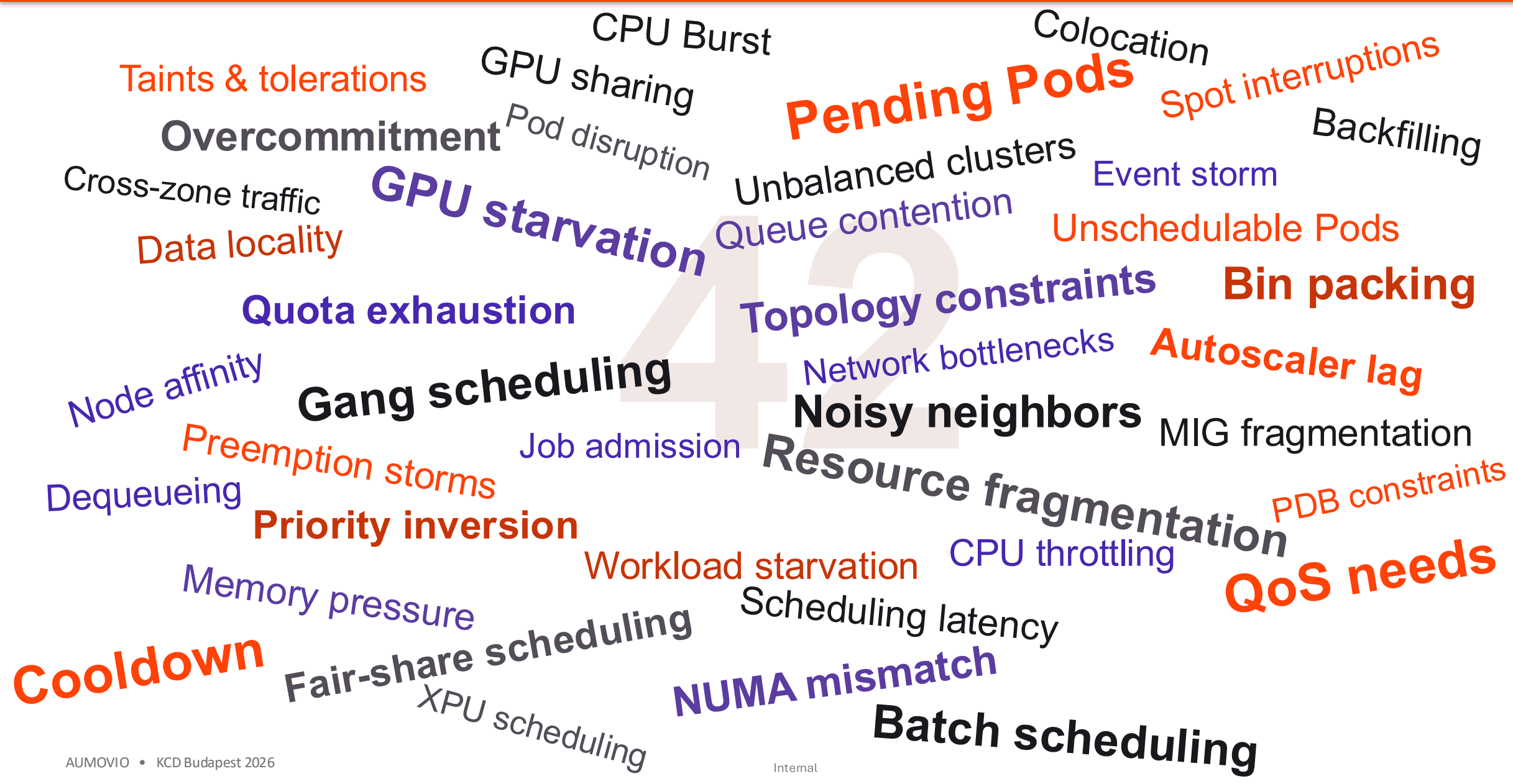
Keep member Pods inside one HyperNode domain.

topologyAffinity

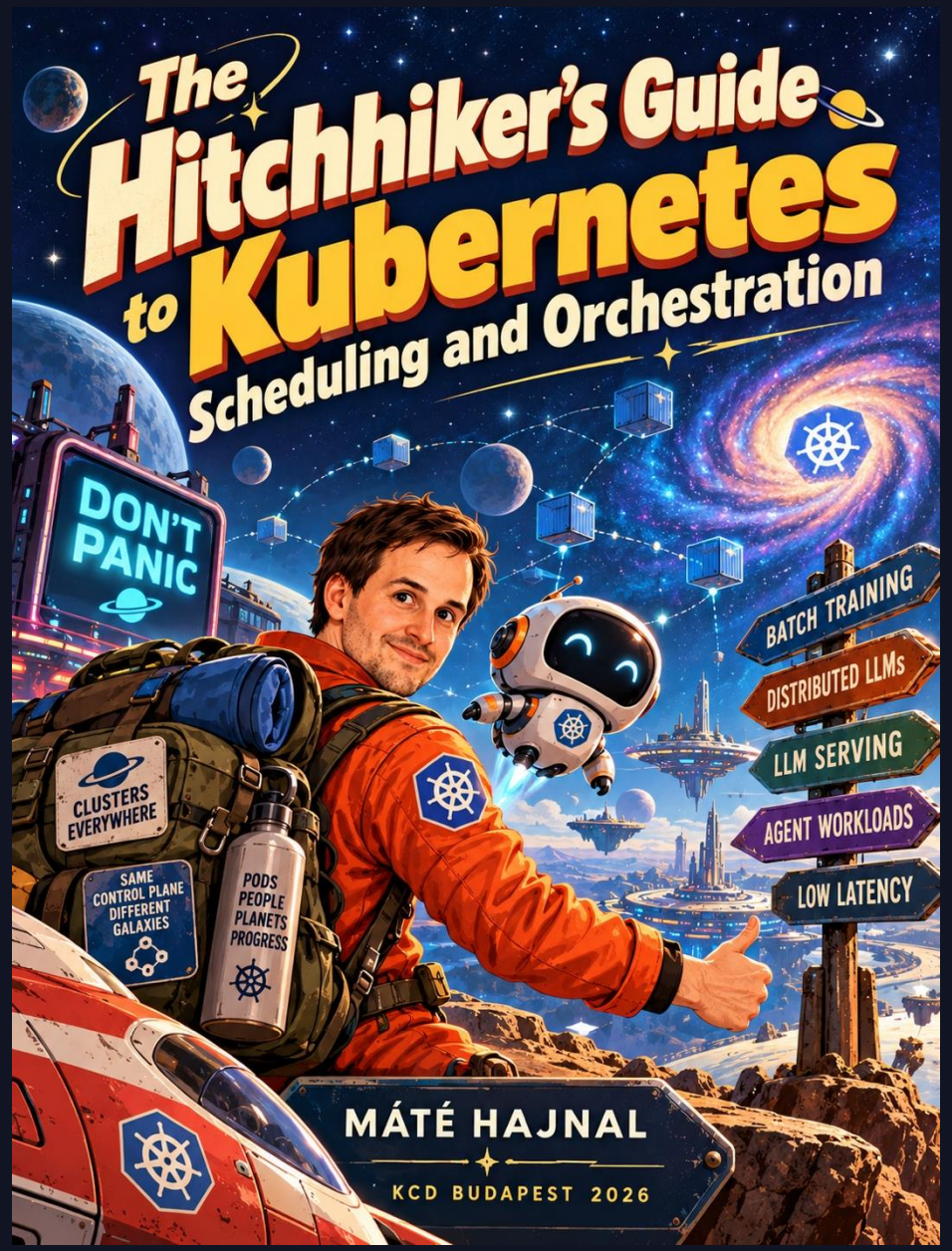
Separate matching PodGroups across those domains.

Requires Volcano v1.16 and the explicit group-topology-affinity plugin.

The problems are…



Internal



SO LONG,
AND THANKS FOR ALL THE GPUs

DON’T PANIC. Schedule responsibly.