HAMI - A CNCF INCUBATION PROJECT

Shared GPU Scheduling & Proactive Autoscaling

A Production Blueprint for 1000+ GPUs

Jeonghyun Kim
AI Engineer, SNOW Corp.
github.com/jeonghyunkeem linkedin.com/in/jeonghyun-kim-2399a6203

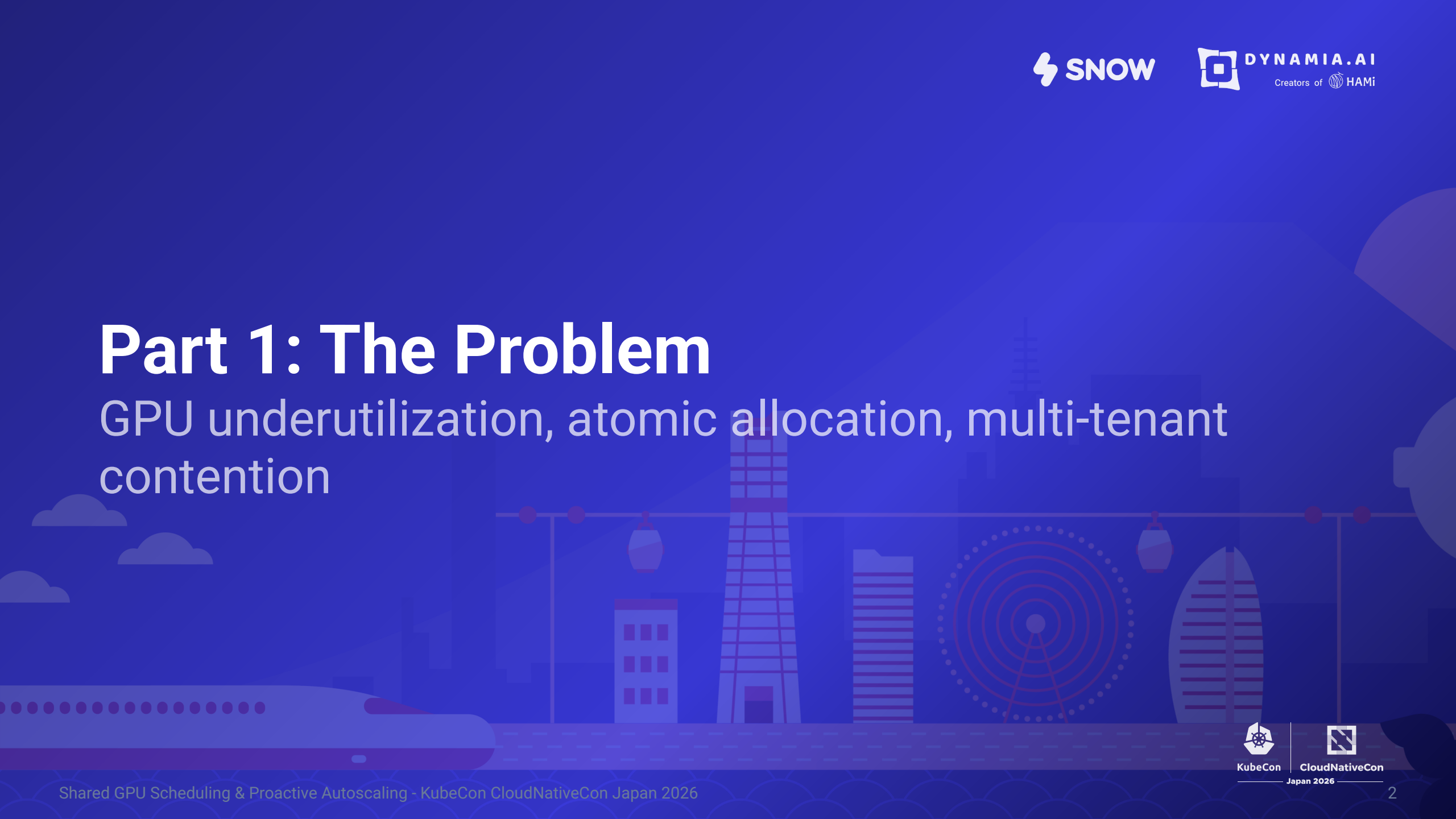
Reza Jelveh
GTM & Solution Architecture @ Dynamia AI - Makers of HAMi
github.com/rezajelveh linkedin.com/in/rezajelveh

SNOW

DYNAMIA.AI
Creators of HAMi

Part 1: The Problem

GPU underutilization, atomic allocation, multi-tenant contention



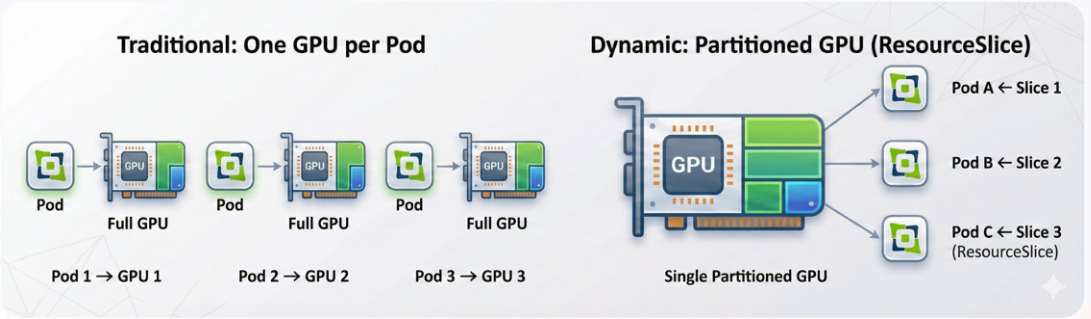
KubeCon CloudNativeCon Japan 2026

The Problem

Atomic GPU allocation wastes silicon

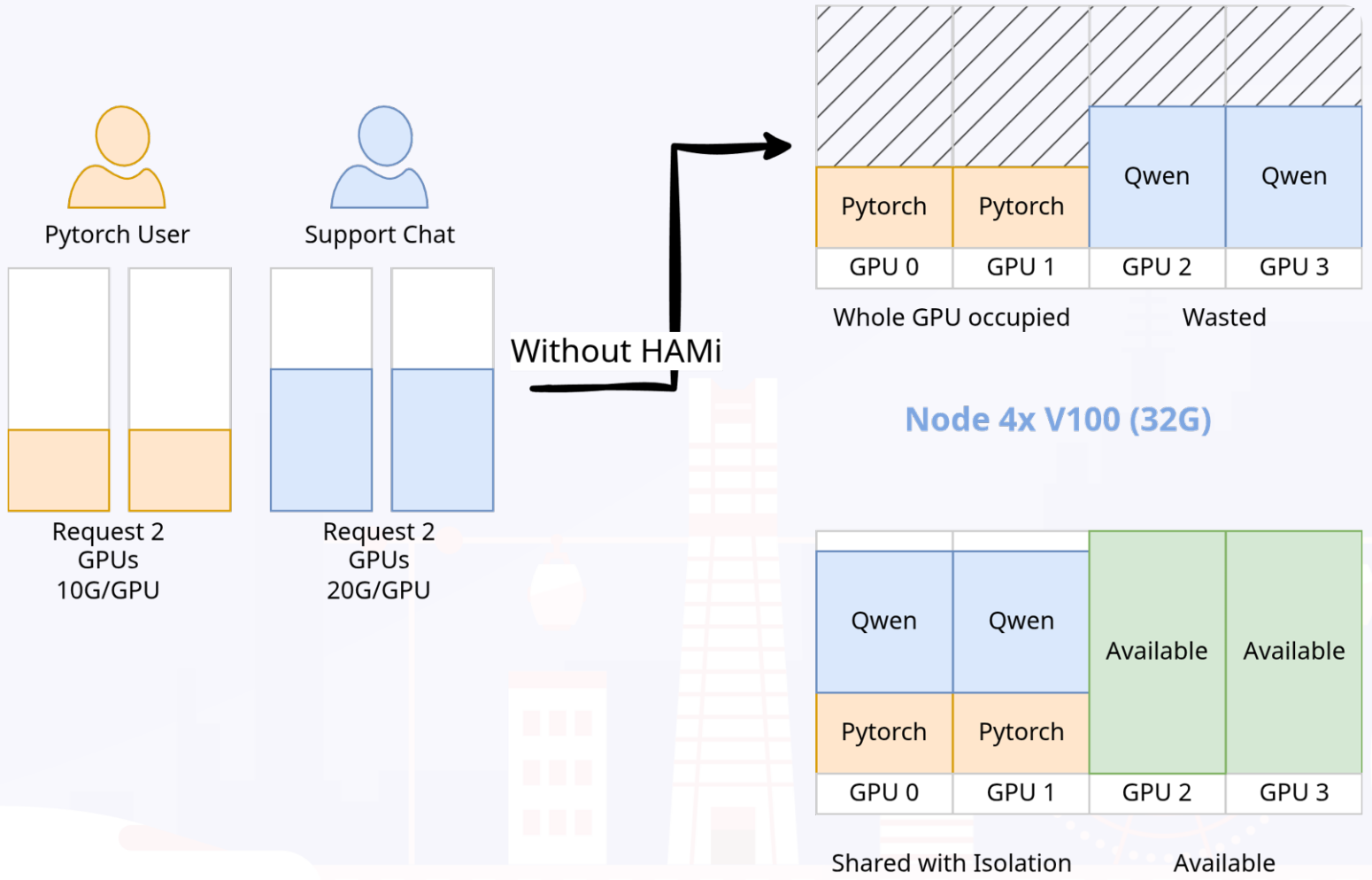
Kubernetes allocates GPUs atomically. DRA + HAMi fix this.

- 1 GB task blocks an 80 GB device
- Over-provisioning is the default: idle silicon
- DRA can request GPUs but requires MIG to slice
- Why HAMi if DRA slices? Covered in Part 2



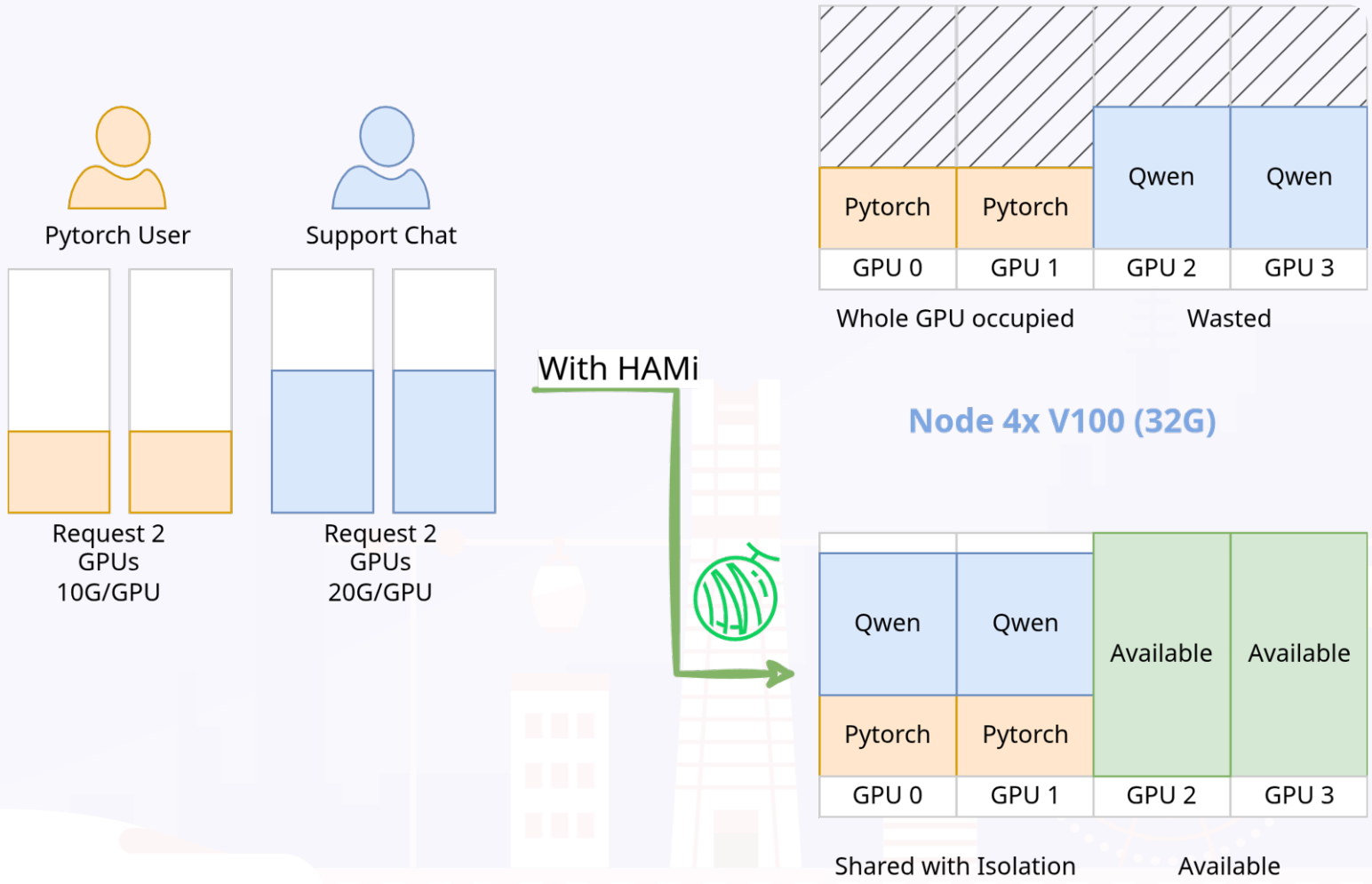
What is HAMi

Static allocation, one GPU per task



What is HAMi

Fractional vGPUs, multiple tasks per device



The GPU Challenge

What breaks, what HAMi needs to solve

Problem

- GPUs are scarce, allocated whole
- Vendors locked in, supply tight
- Utilization stuck at 10%
- No central observability
- Fragmented inference workloads

Requirements

- Hardware agnostic: one API, any accelerator
- Fractional GPU: fine-grained slices, multiple tasks per device
- Advanced scheduling: binpack, spread, topology-aware
- Unified observability across vendors

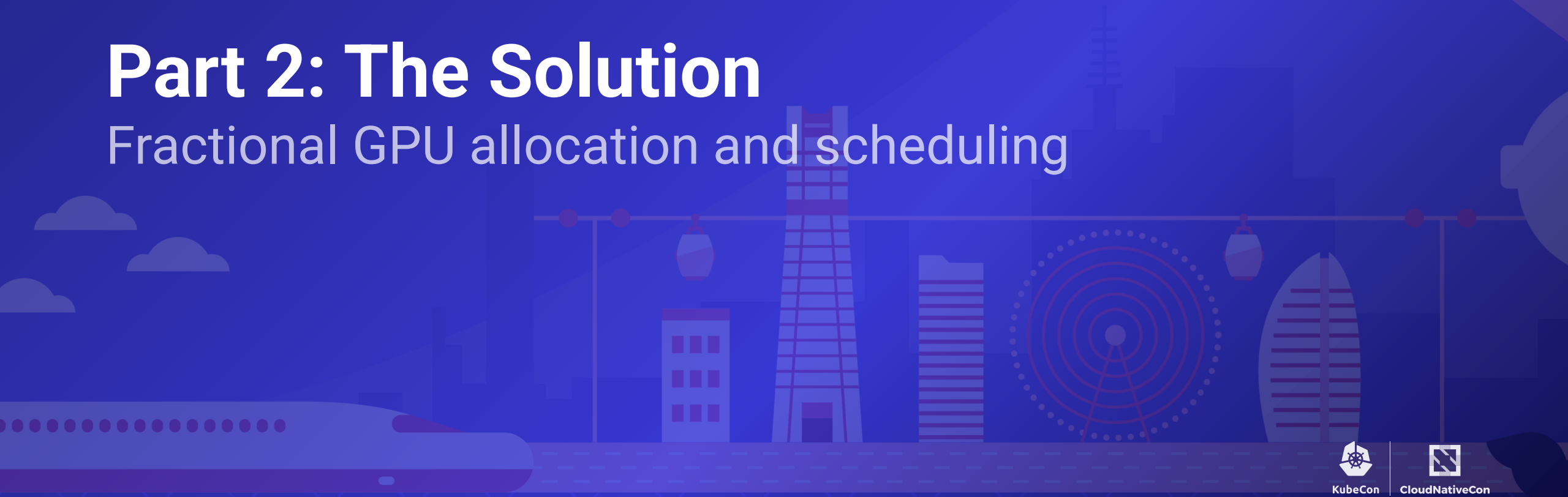
Unified observability, 50% GPU utilization, 10x workloads running, 10x GPU availability. AMD MI355X: 80% of B200 perf at ~1/3 the cost. Not everyone needs Vera Rubin.

SNOW

DYNAMIA.AI
Creators of HAMi

Part 2: The Solution

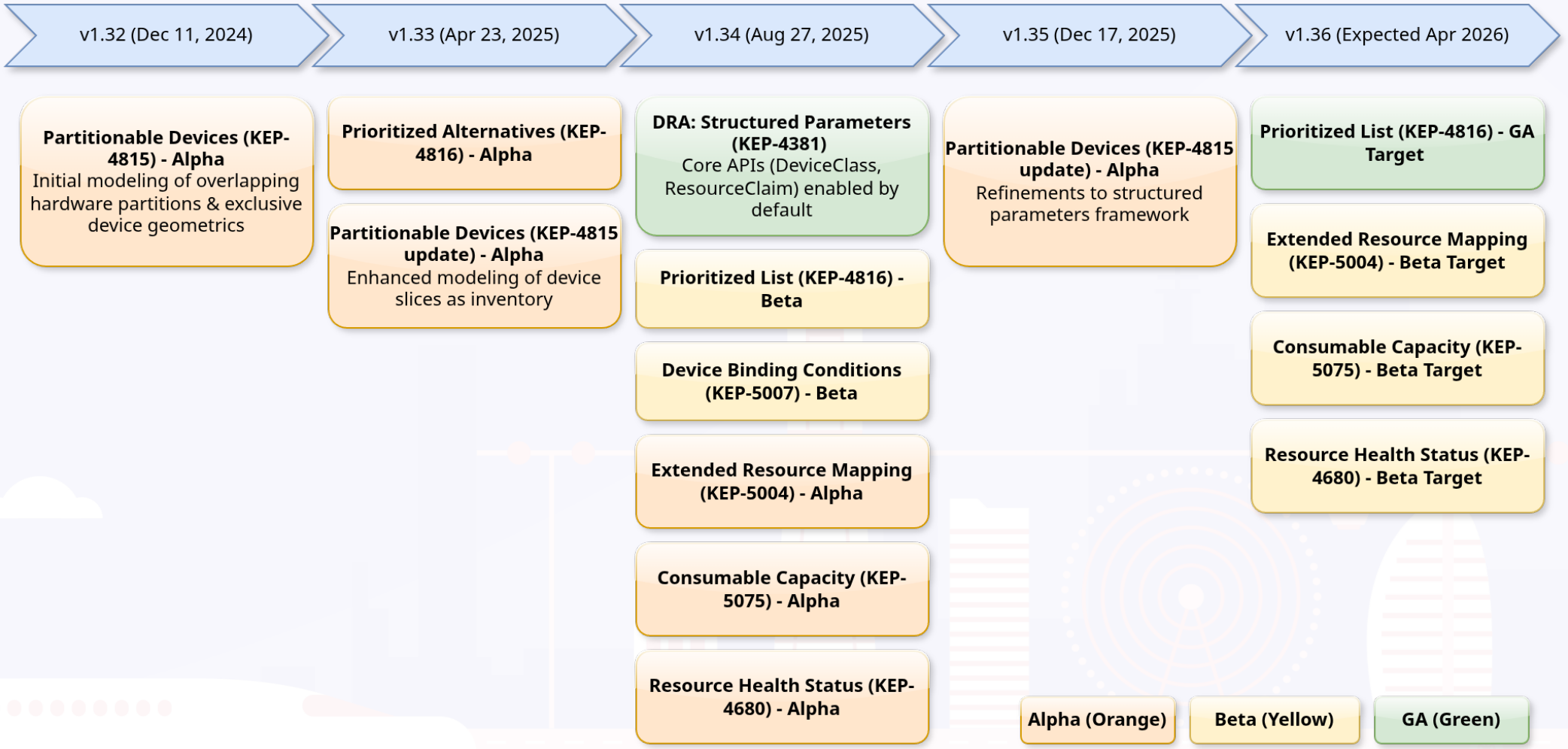
Fractional GPU allocation and scheduling



KubeCon CloudNativeCon Japan 2026

DRA Feature Timeline

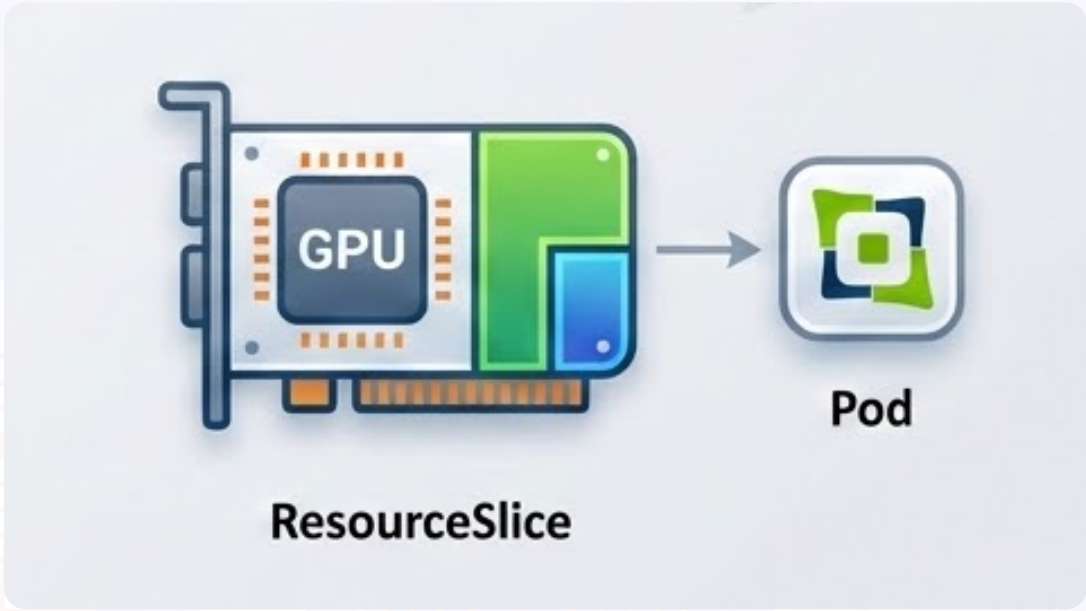
KEPs and their development status: not covered today



DRA: ResourceSlice

Per-node device inventory

- **ResourceSlice:** lists all available devices on a node with their attributes
- DRA drivers publish slices; scheduler reads them to match claims to devices
- Tied to a node via `nodeName`, supports topology and NUMA attributes



DRA: ResourceClaim

A standardized way to request hardware: not just GPUs. Stable in K8s 1.34.

- **DeviceClass:** groups devices with identical resource models. **ResourceClaim:** a workload's ticket to hardware. **ResourceClaimTemplate:** reusable blueprint, auto-creates a claim per Pod.

HAMi Capabilities

Six things HAMi brings to GPU scheduling

Heterogeneous Management

Manage GPU, NPU, MLU, and other accelerators in one workflow.

Hard Isolation

Slice memory and compute with hard isolation at runtime.

Advanced Scheduling

Binpack, spread, and topology-aware placement policies.

Kubernetes Native

Kubernetes-native APIs, DRA, and CDI support.

Resource Isolation & QoS

Memory and core quotas for fair, stable sharing.

Unified Monitoring

Consistent metrics and visibility across vendors.

GPU Sharing

Dynamic fine-grained device slicing

- **NVIDIA, Ascend, Cambricon, Hygon, Iluvatar** supported
- **Fine-grained:** as small as 1MB device memory, 1% computing cores
- **Transparent to tasks:** no code changes required
- **Hard resource isolation** inside containers
- One API across vendors: same YAML, any accelerator

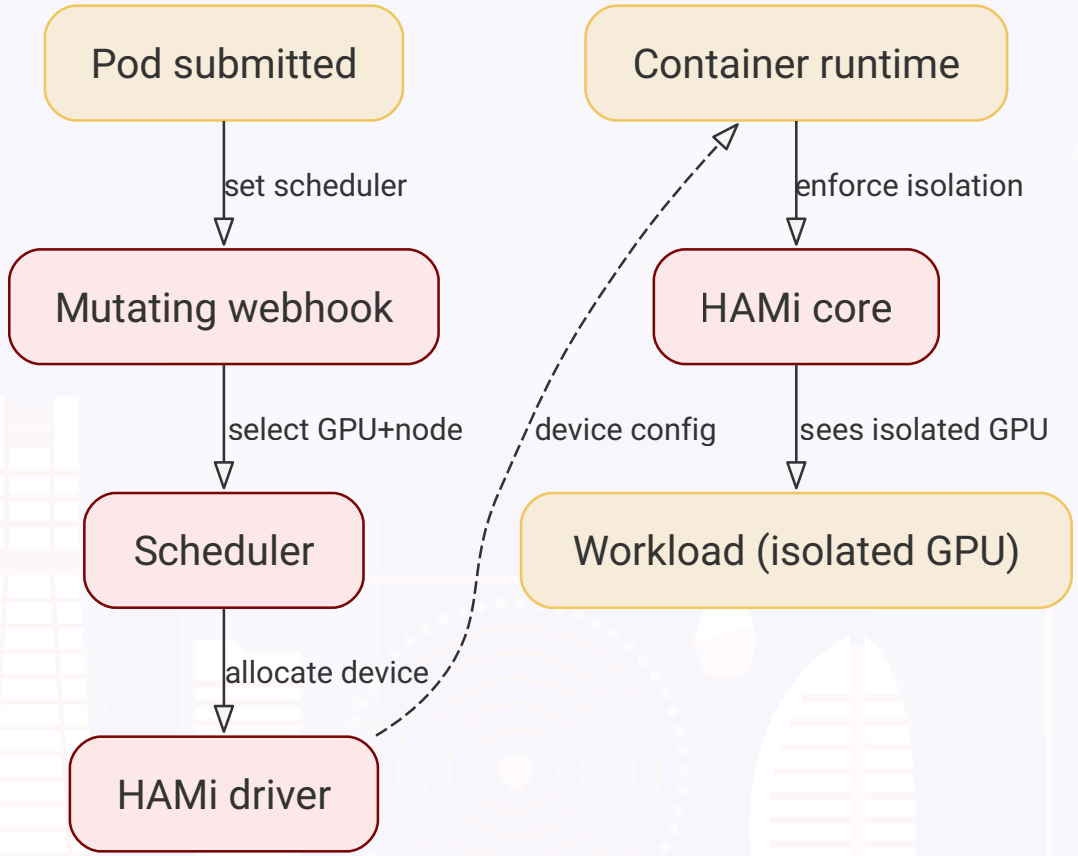
```yaml
# Ascend 910C: 8GB + 20% compute
resources:
  limits:
    huawei.com/Ascend910C: "1"
    huawei.com/Ascend910C-core: "20"
    huawei.com/Ascend910C-memory: "8192"

# NVIDIA: 3GB on any GPU
resources:
  limits:
    nvidia.com/gpu: 1
    nvidia.com/gpumem: 3000
```

How HAMi Works

Pod creation to GPU isolation

HAMi intercepts pod creation and GPU allocation through seven stages:

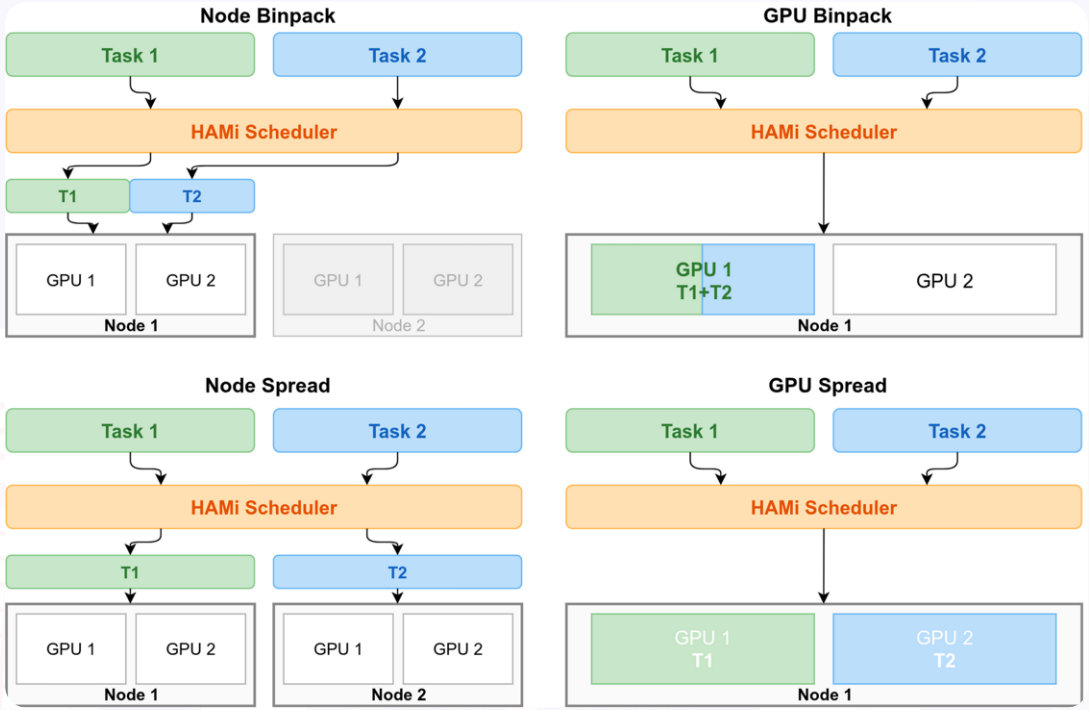
- **Mutating webhook:** detects GPU requests, routes pod to HAMi scheduler
- **Scheduler:** selects GPU and node via HAMi scheduler extender
- **HAMi driver:** generates device config
- **Container runtime:** reads config, injects HAMi-Core library
- **HAMi core:** enforces isolation in-process via symbolic hijacking



Scheduling Policies

Binpack & Spread

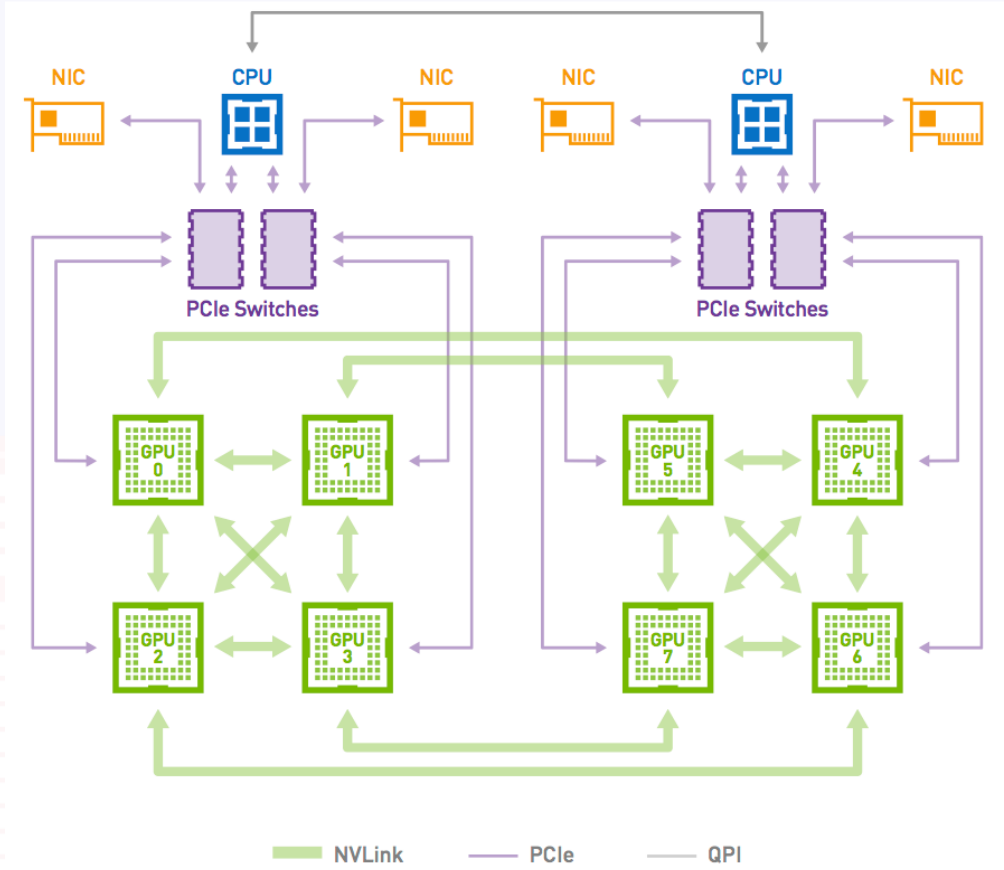
- **Node binpack** frees whole machines: reduces cost, helps cluster autoscaler
- **Node spread** isolates faults: HA across zones, blast radius control
- **GPU binpack** prevents fragmentation: frees entire GPUs for training
- **GPU spread** protects tail latency: reduces HBM and NVLink contention
- Advanced scheduling works with standalone HAMi; DRA mode can use Yunikorn



Scheduling Policies

Topology-Aware

- **NVLink 3 (A100):** 600 GB/s, 12 links
- **NVLink 4 (H100/H200):** 900 GB/s bidirectional across 18 links
- **NVLink 5 (B200/B300):** 1.8 TB/s, 14x PCIe 5.0
- **NVLink 6 (Rubin):** ~3.6 TB/s target
- **PCIe 5.0 x16:** 128 GB/s. **PCIe 6.0:** 242 GB/s
- **HAMi topology policy:** prefers NVLink (NVIDIA), HCCS (Ascend), and other high-speed interconnects, avoids PCIe bridge pairs



GPU Sharing Approaches

MIG vs HAMi vs HAMi+DRA vs NVIDIA DRA

Capability	MIG	HAMi	HAMi+DRA	NVIDIA DRA
Pre-configured templates	✗	✓	✓	✓
Dynamic MIG repartition	✗	✓	✓	✓
Symbolic hijacking	✗	✓	✓	✗
Consumable capacities	✗	✓	✓	✗
Multi-vendor	✗	✓	✓	✗
Advanced scheduling	✗	✓	✗	✗

NVIDIA DRA supports MIG, MPS, and VFIO passthrough with dynamic repartitioning. HAMi differentiates with symbolic hijacking for sub-MIG slicing (1MB), consumable capacities for flexible resource requests, and multi-vendor support. **HAMi-DRA supports multiple DRA drivers**. Its NVIDIA driver builds on NVIDIA's upstream, adding consumable capacities.

SNOW

DYNAMIA.AI
Creators of HAMi

Part 3: SNOW Corp. at Scale

200M users, 1000+ GPUs, and the limits of static Docker

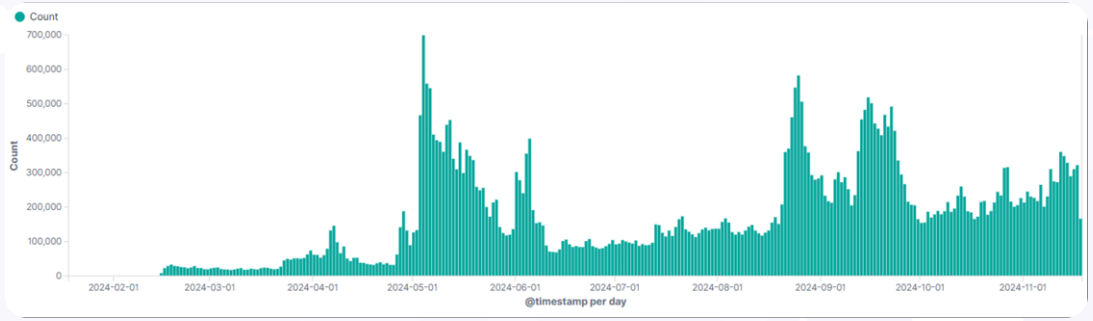
KubeCon CloudNativeCon Japan 2026

The Challenge at Scale

200M Users, 1000+ GPUs, 1200+ Workflows

SNOW Corp., subsidiary of NAVER, manages 1000+ A100 GPUs serving 200M users across three top-ranked GenAI applications - SNOW, EPIK, B612 - handling extreme traffic volatility from viral AI trends.

- 3 apps in a16z Top 50 Gen AI Mobile Apps
- #1 Camera/Photo app in Korea, Japan, Vietnam
- 1.5B+ cumulative downloads



Three Workload Challenges

Continuous evolution, traffic volatility, heterogeneous workflows

Continuous Service Evolution

Market leadership needs a constant stream of new GenAI filters: rapid model deployment and frequent updates with no service interruption.

Extreme Traffic Volatility

Viral AI trends such as the Ghibli Filter trigger unpredictable surges up to 700%, making static capacity planning impossible.

Heterogeneous Inference Workflows

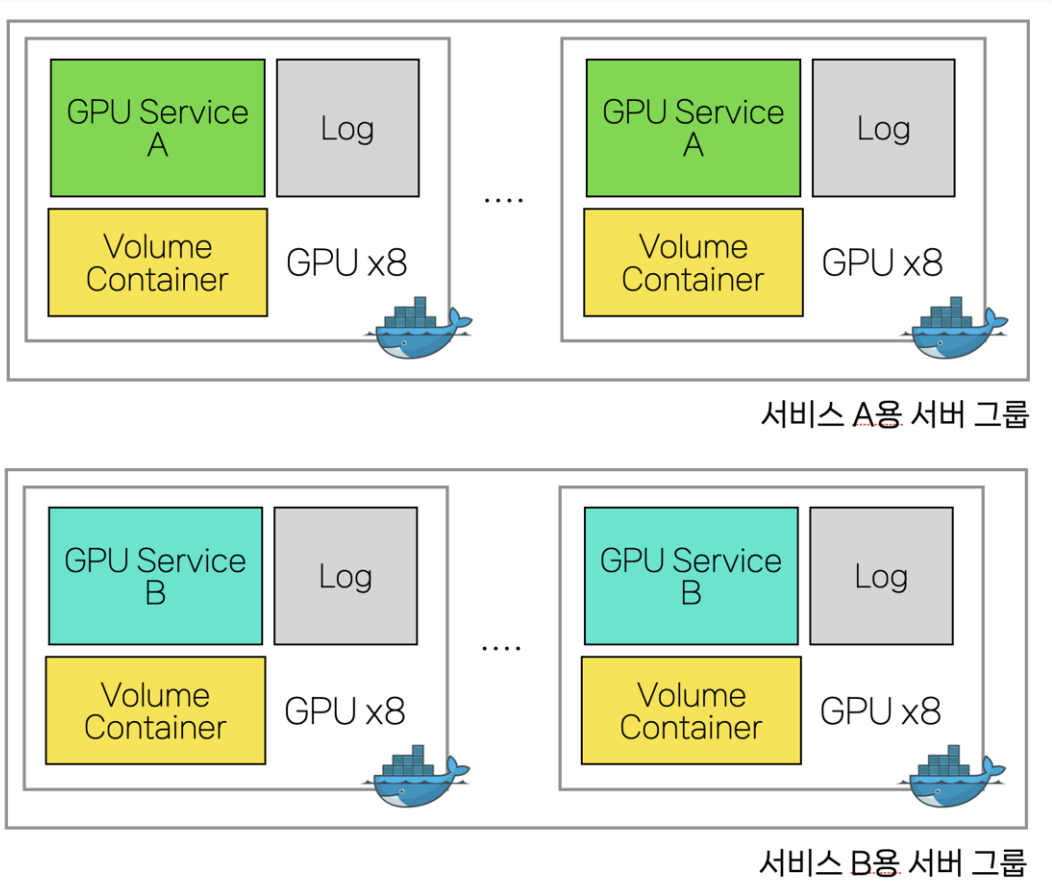
Some filters fine-tune on a user's appearance before generating from it; others are inference only. Mixed pipeline shapes and resource profiles make one-size-fits-all allocation inefficient.

The Legacy: Static Docker

Manual GPU binding per host, and what it cost

Isolated Docker hosts with local volume containers and manual GPU binding, with no centralized control plane.

- ⚠ **System instability:** no centralized monitoring or recovery, directly impacting 200M users
- **Inefficient utilization:** static allocation fragmented GPUs and wasted capacity
- **Operational overload:** no automatic recovery, so manual intervention drove up cost



Infrastructure Evolution

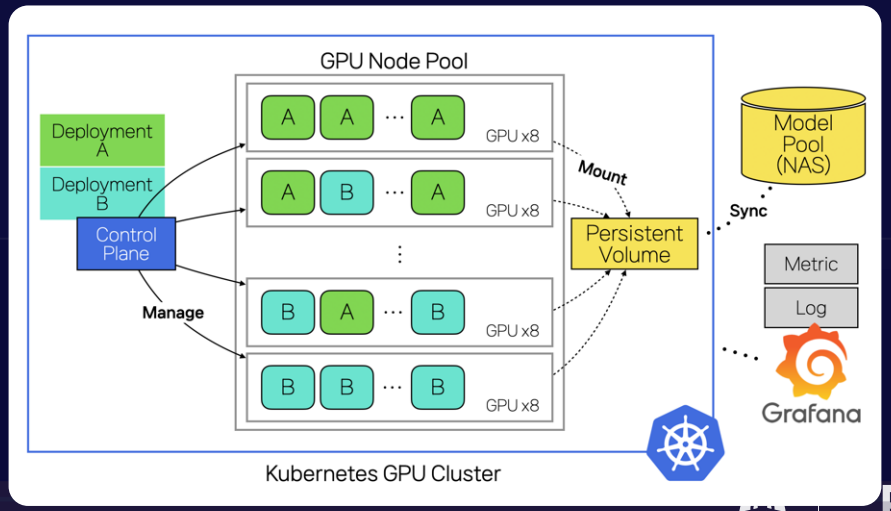
From Docker silos to Kubernetes + HAMi

AS-IS: Legacy Docker

- Manual GPU binding
- Isolated hosts
- Static allocation
- Low utilization.

TO-BE: Kubernetes + HAMi

- Centralized control plane
- Dynamic node pools
- vGPU sharing enabled
- Automated recovery + scaling



SNOW

DYNAMIA.AI
Creators of HAMi

Part 4: Methodology

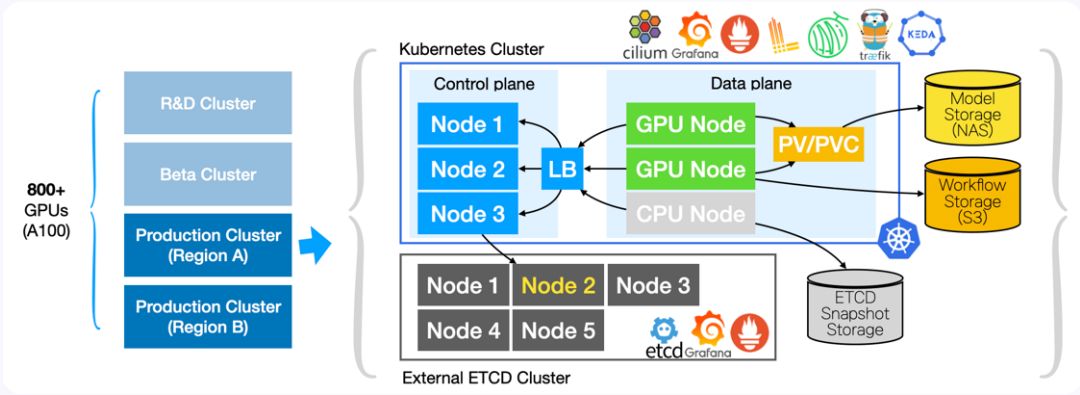
How SNOW Fixed It

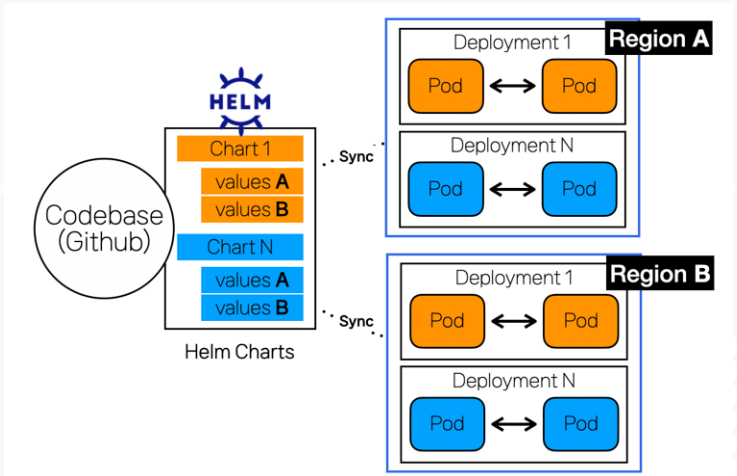
KubeCon CloudNativeCon Japan 2026

Building a Cloud-Native Foundation

Multi-region on-premise HA with decoupled ETCD

Project	Role
Kubernetes	Container orchestration
Cilium	High-performance CNI
HAMi	GPU sharing scheduler
KEDA	Autoscaling
Helm	Service configuration
Prometheus + Grafana	Monitoring
Traefik	Ingress / reverse proxy

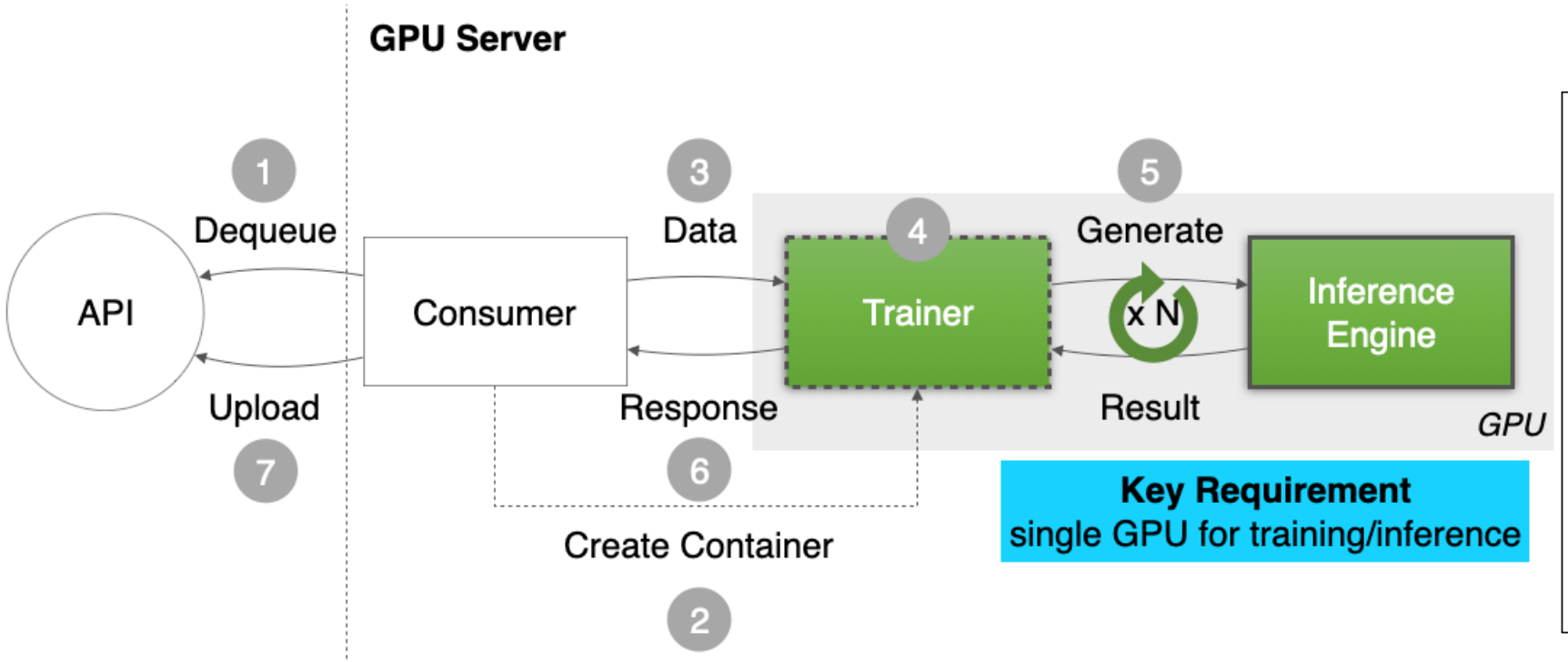




GPU Sharing: The Migration Hurdle

Train-to-Inference pipeline blocked by GPU isolation

Kubernetes' strict GPU isolation blocked the sequential "Train-to-Inference" pipeline.




```
apiVersion: v1
kind: Pod
metadata:
  name: train_and_inference
spec:
  containers:
    - name: train
      resources:
        limits:
          nvidia.com/gpu: 1
    - name: inference
      resources:
        limits:
          nvidia.com/gpu: 1
```

Problem: Default scheduler cannot share one GPU across containers.

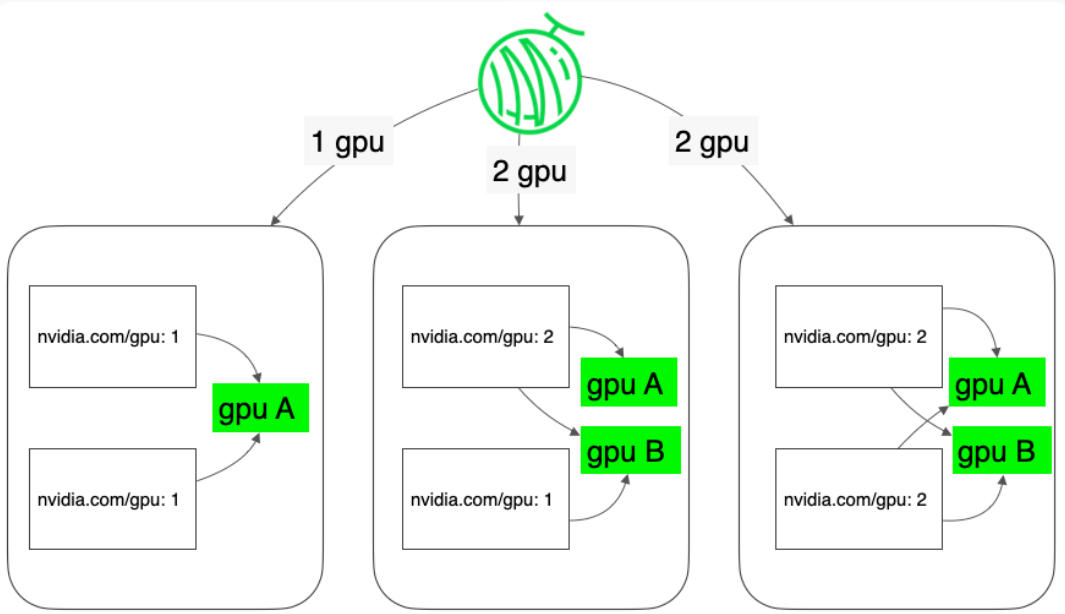
Without HAMi: 2x GPU usage or massive code rewrite.

Migration Solution: HAMi vGPU

Device sharing without code changes

- **Device sharing:** Multiple containers share one GPU concurrently
- **Zero code changes:** Install via Helm, assign GPUs in chart
- **Kubernetes-native:** Parallel with kube-scheduler, no conflicts

Result: flexible GPU scheduling comparable to Docker, with enhanced utilization and stability.



Proactive GPU Orchestration

Why conventional metrics miss GPU saturation

SNOW's inference fleet is heterogeneous, so utilization and saturation are not the same signal.

Inaccurate Saturation Signal

CPU and RAM miss real service load. Even DCGM GPU utilization is unreliable: varying workflow intensities mean high utilization does not imply saturation, and vice versa.

Lagging Indicator

KEDA's built-in RabbitMQ scaler triggers on queue length, so it only reacts once a backlog exists.

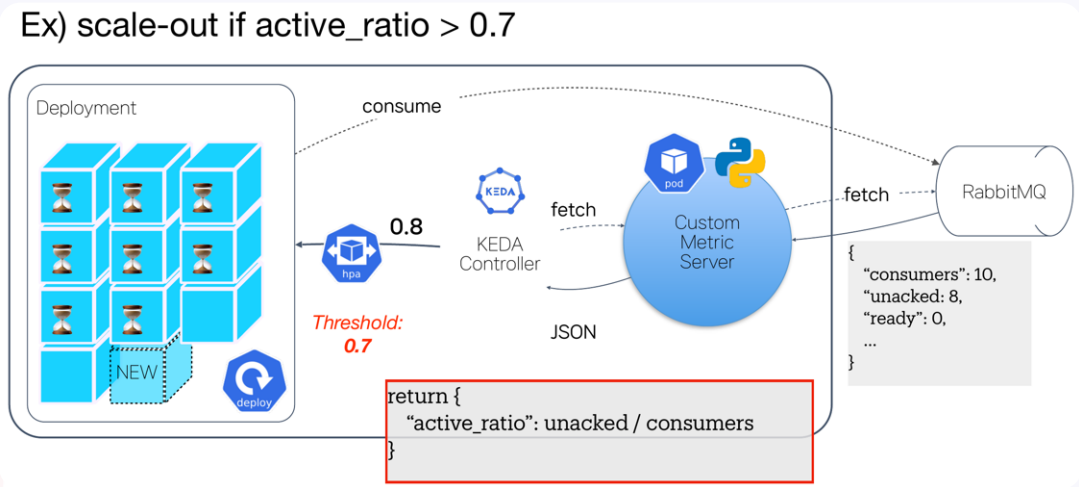
With a 60 second model warm-up, that is already too late: requests are throttled before new workers can serve.

Custom KEDA Metric Server

Consumer Saturation: unacked messages / active consumers

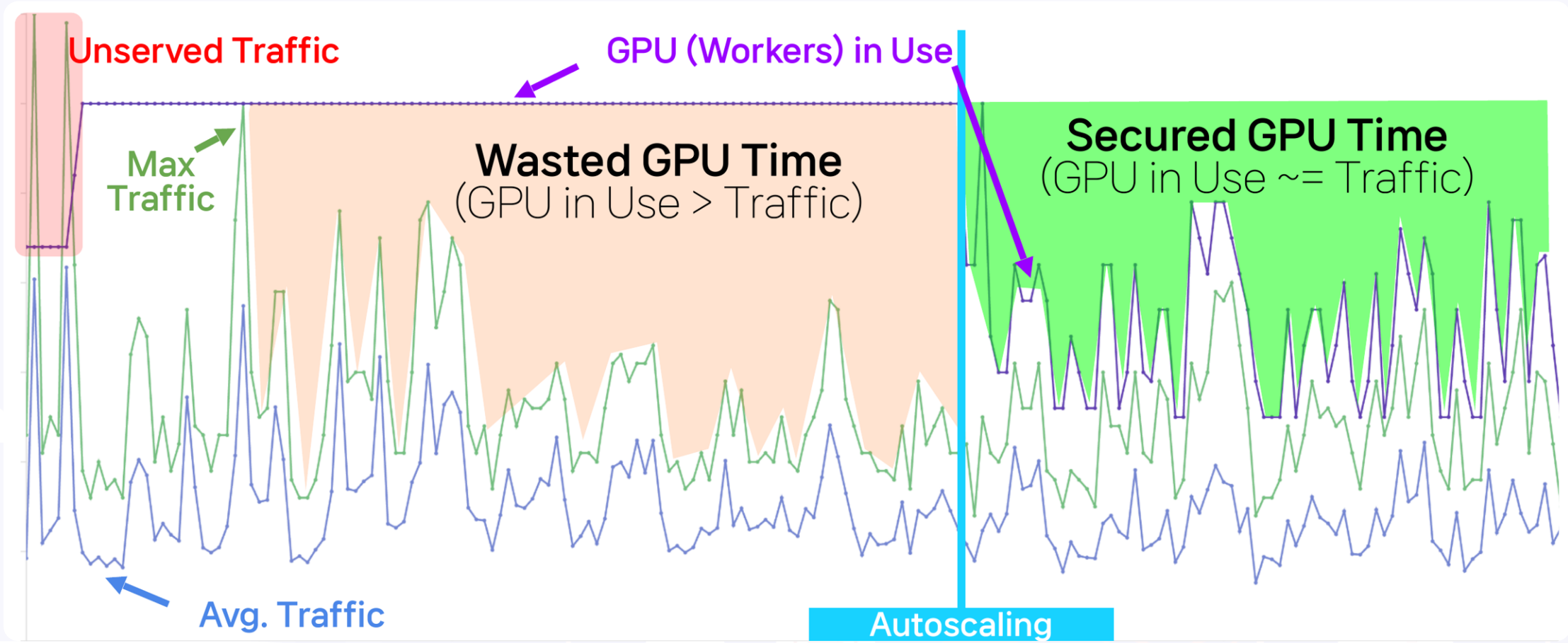
SNOW built a lightweight Python metric server that exposes real-time RabbitMQ consumer state to the Kubernetes HPA through KEDA's Metrics API.

- `active_ratio = unacked / consumers`, scaling out above a **0.7** threshold
- Provisions GPUs **before** the pool saturates, creating a buffer that absorbs the 60s warm-up
- Longer stabilization and cooldown windows prevent premature scale-in during traffic lulls



From Wasted to Secured GPU Time

GPU allocation tracks real-time user traffic



Static provisioning burns **wasted GPU time** when supply sits above traffic (orange), and still drops **unserved spikes** (red). After autoscaling, allocation tracks demand closely (green).

SNOW

DYNAMIA.AI
Creators of HAMi

Part 5: Results

What SNOW Achieved

KubeCon CloudNativeCon Japan 2026

Quantitative Results

Fewer GPUs, less idle time, faster recovery

2x

Fewer GPUs for train plus inference pipelines

-55%

Average GPU time per production cluster

-91%

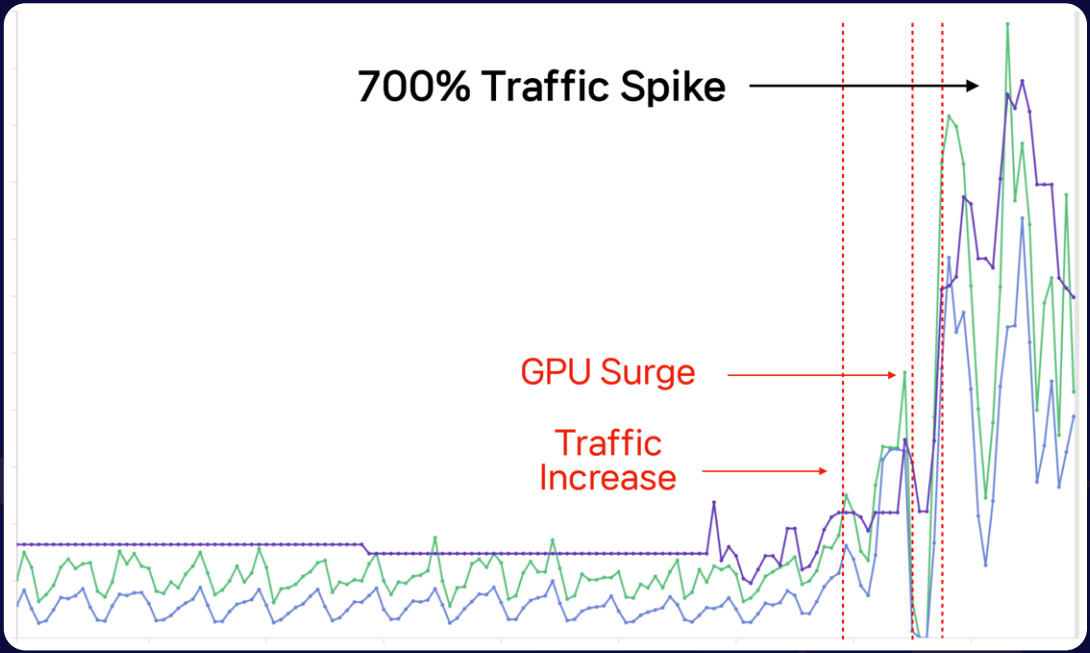
MTTR, from ~2hr to ~10min

-85%

GPU surge errors during peak traffic

Hybrid Cloud Bursting: 700% Spike

Ghibli Filter traffic surge handled with zero downtime



During the viral "Ghibli Filter" trend:

- Traffic tripled in 3 hours on a low-staff Saturday
- Autoscaling initially held, then GPU saturation hit
- Expanded from on-prem to CSP clusters via GitOps
- Unified Helm charts deployed across all regions

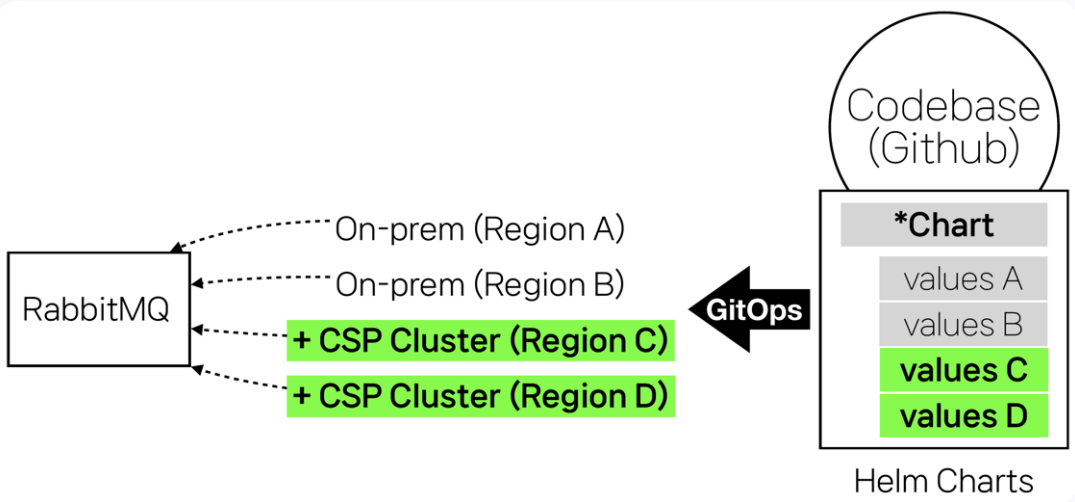
Achieved 7x peak consumption without service interruption.

Multi-Cluster Bursting Architecture

One GitOps pipeline across on-premise and CSP regions

To overcome on-premise capacity limits, the system expanded into Cloud Service Provider regions.

- Identical Helm charts deployed to every cluster via GitOps
- On-premise Regions A and B, burst into CSP Regions C and D
- CSP worker nodes consume from the central RabbitMQ over a secured connection



GPU Monitoring Dashboard

Key Takeaways

Production blueprint, GPU sharing, proactive scaling

Production Blueprint

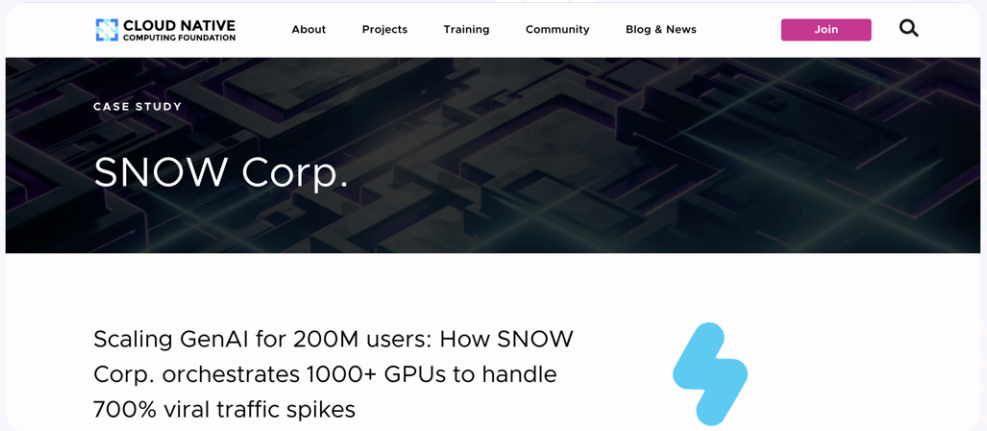
CNCF ecosystem provides production-grade foundation for AI workloads at scale. HAMi + KEDA = proven at 200M users.

GPU Sharing

HAMi enables efficient GPU utilization without code changes - critical for migration from legacy Docker setups.

Proactive Scaling

Custom KEDA metrics beat reactive scaling for GPU workloads with warm-up latency. Consumer Saturation is the key metric.



Full write-up: cncf.io/case-studies/snow-corp

SNOW

DYNAMIA.AI
Creators of HAMi

Part 6: Where HAMi Is Today

Adoption, Case Studies, Community

KubeCon CloudNativeCon Japan 2026

Where HAMi Is Today

Production metrics from CNCF case studies

100%	10x	30%	10,000+
Hardware pool utilization Baike Holdings	GPU utilization in CI NIO	Fewer GPU hours China Merchants Bank	Pods running concurrently Baike Holdings

China Merchants Bank · SNOW Corp. · NIO · KE Holdings · DaoCloud · SF Technology · Prep Education · cncf.io/case-studies · Reach out for help submitting yours.

Community & Adopters

Devices, integrations, and who uses HAMi

Open Source, CNCF Backed, Production Ready

4.1k	325k	500+	27
Github Stars	Docker Pulls	Contributors	Contributor Countries

kubernetes · VOLCANO · Kueue · koordinator · KAI Scheduler · COZYSTACK

Ecosystem & Device Support

NVIDIA · Ascend · Cambricon 寒武纪 · HYGON 海光 · 天数智芯 Iluvatar CoreX · METAX 沐曦 · 摩尔线程 MOORE THREADS · 昆仑芯 KUNLUNXIN · Enflame 燧原科技 · aws · 瀚博半导体

Adopters

4Paradigm · 百度智能云 · 贝壳 · 招商银行 · 中国移动 China Mobile · 中国联通 China unicom · DaoCloud · DYNAMIA · H3C · HUAWEI · LinkedIn · 马上消费 · NIO · PPIO · PREP · SAP · SF TECHNOLOGY · 思特奇 SI-TECH · SNOW · viettel

QUESTIONS

Thank You

Jeonghyun Kim
AI Engineer, SNOW Corp.
github.com/jeonghyunkeem linkedin.com/in/jeonghyun-kim-2399a6203

Reza Jelveh
GTM & Solution Architecture @ Dynamia AI - Makers of HAMi
github.com/rezajelveh linkedin.com/in/rezajelveh