COSCUP 2026 - Track: Golang TW x Cloud Native

Running Multiple AI Workloads on One GPU with HAMi

Architecture and Gotchas

Reza Jelveh
Solution Architect, Dynamia AI - Makers of HAMi
github.com/fishman linkedin.com/in/rezajelveh

DYNAMIA.AI
Creators of HAMi

Part 1: The Problem

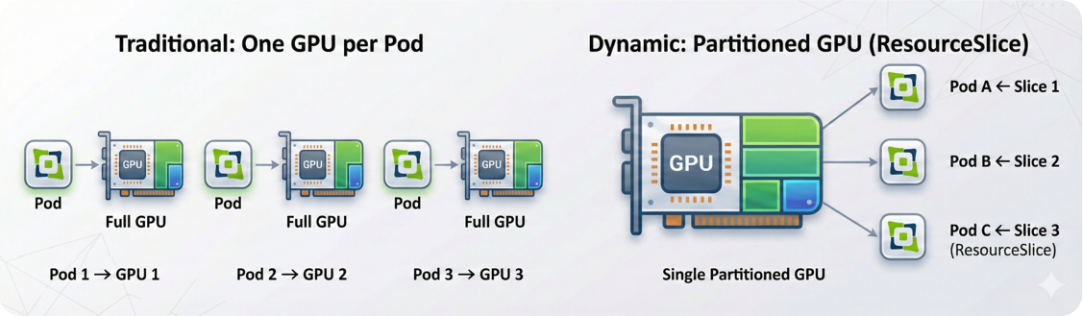
GPUs are expensive, and Kubernetes does not share them well

The Problem

One GPU per task is the default

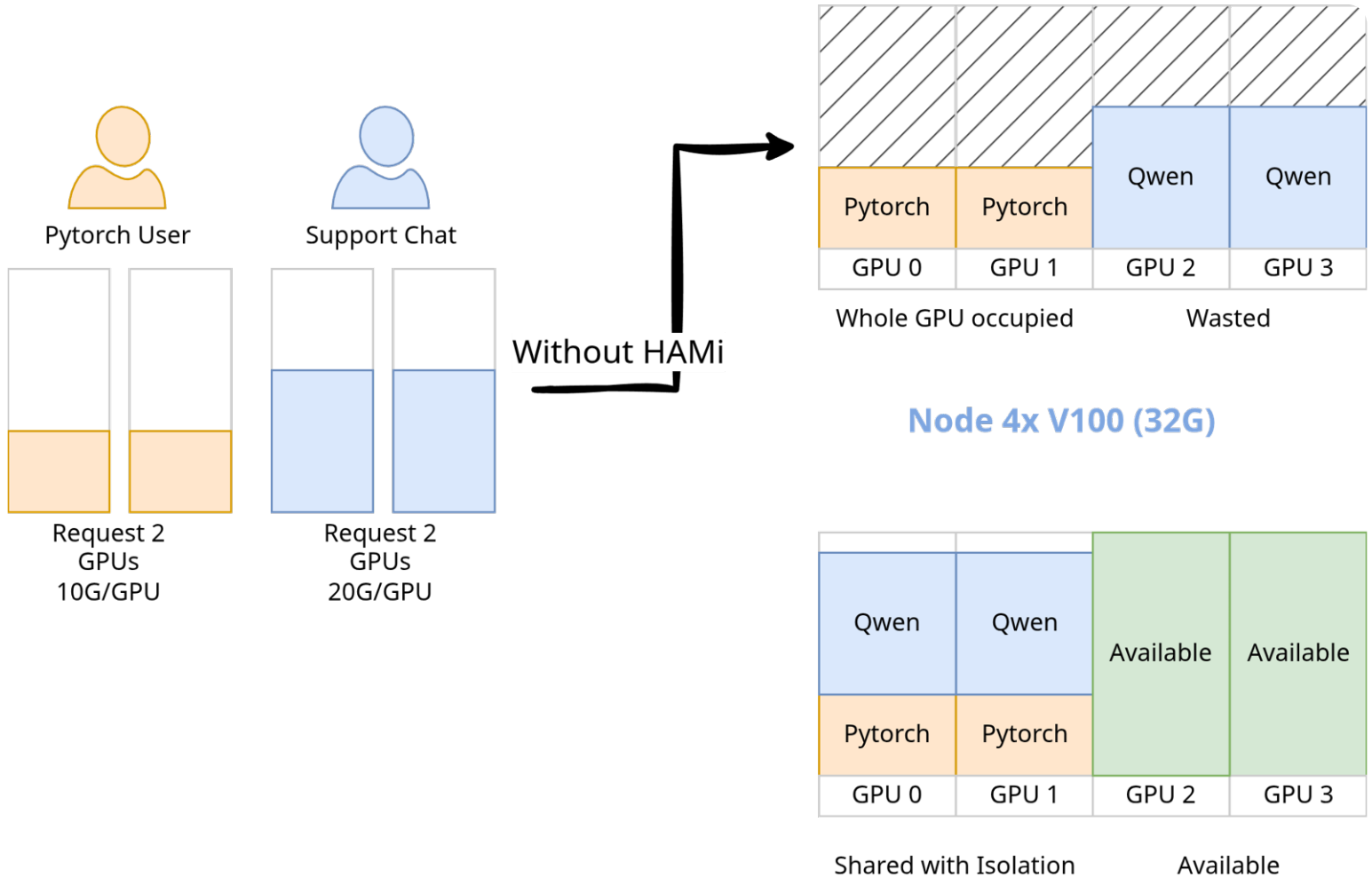
Kubernetes allocates GPUs atomically: one whole device, one task.

- A 1 GB task blocks an 80 GB device
- Most GPUs sit idle most of the time
- DRA (Dynamic Resource Allocation) is stable, but still work in progress
- No advanced scheduling yet: no binpack, no spread, no topology



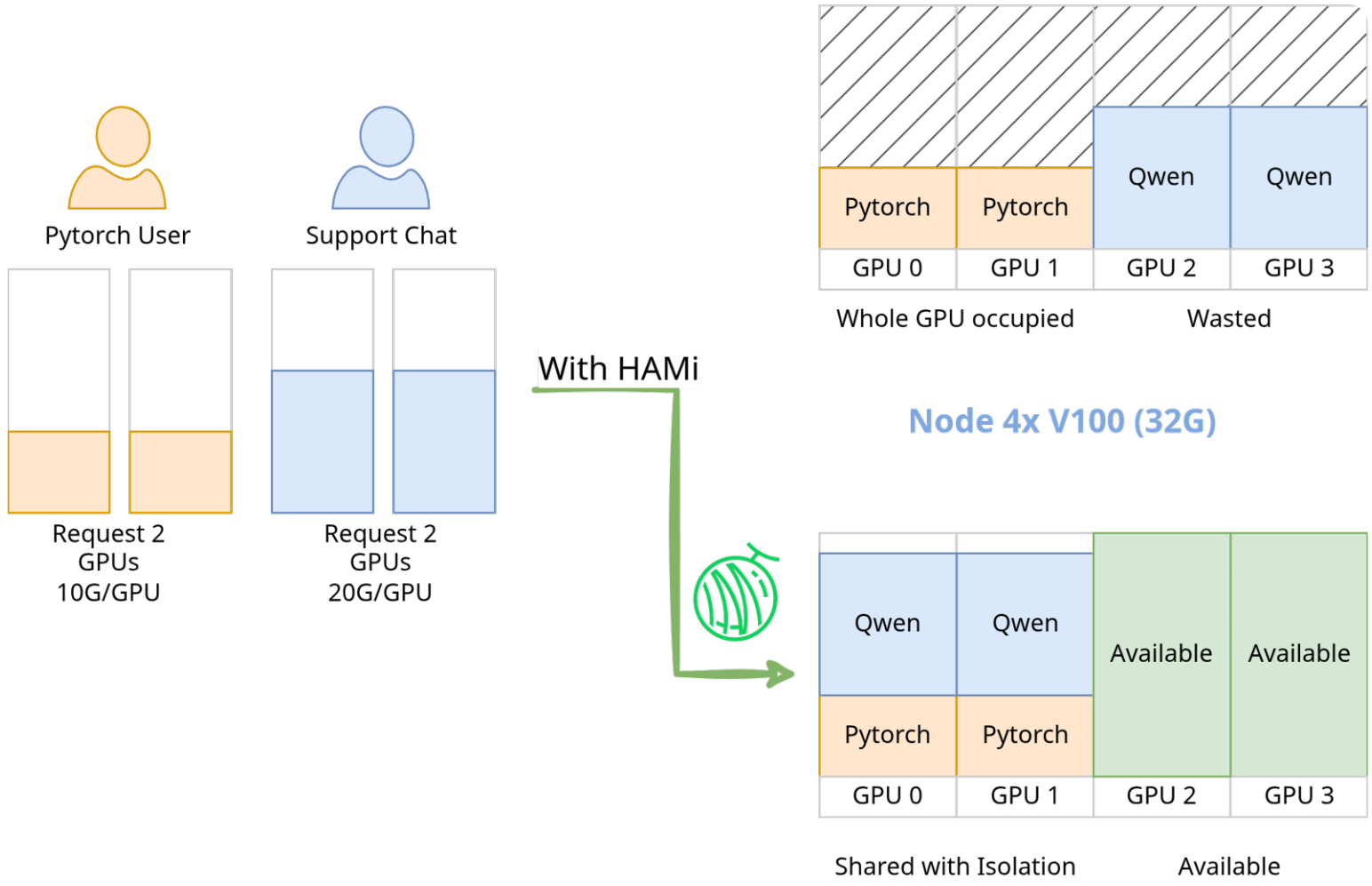
What is HAMi

Before: one GPU, one task



What is HAMi

After: one GPU, many tasks



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

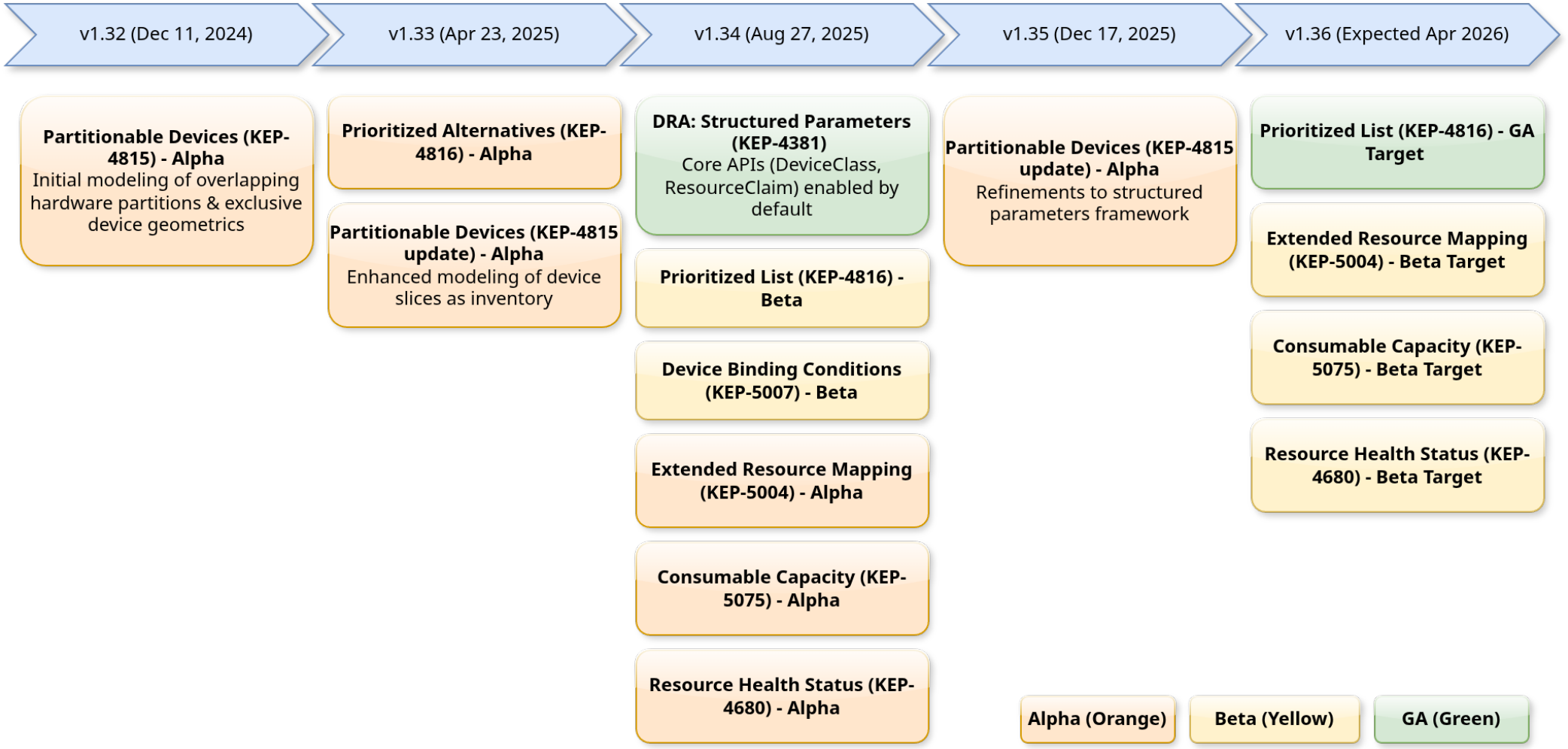
Unified observability, 50% GPU utilization, 10x workloads running, 10x GPU availability. AMD MI355X: 80% of B200 perf at ~1/3 the cost. Not everyone needs Vera Rubin.

Part 2: The Solution

No code changes. No kernel modules. No vendor lock-in.

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.

DYNAMIA.AI
Creators of HAMi

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

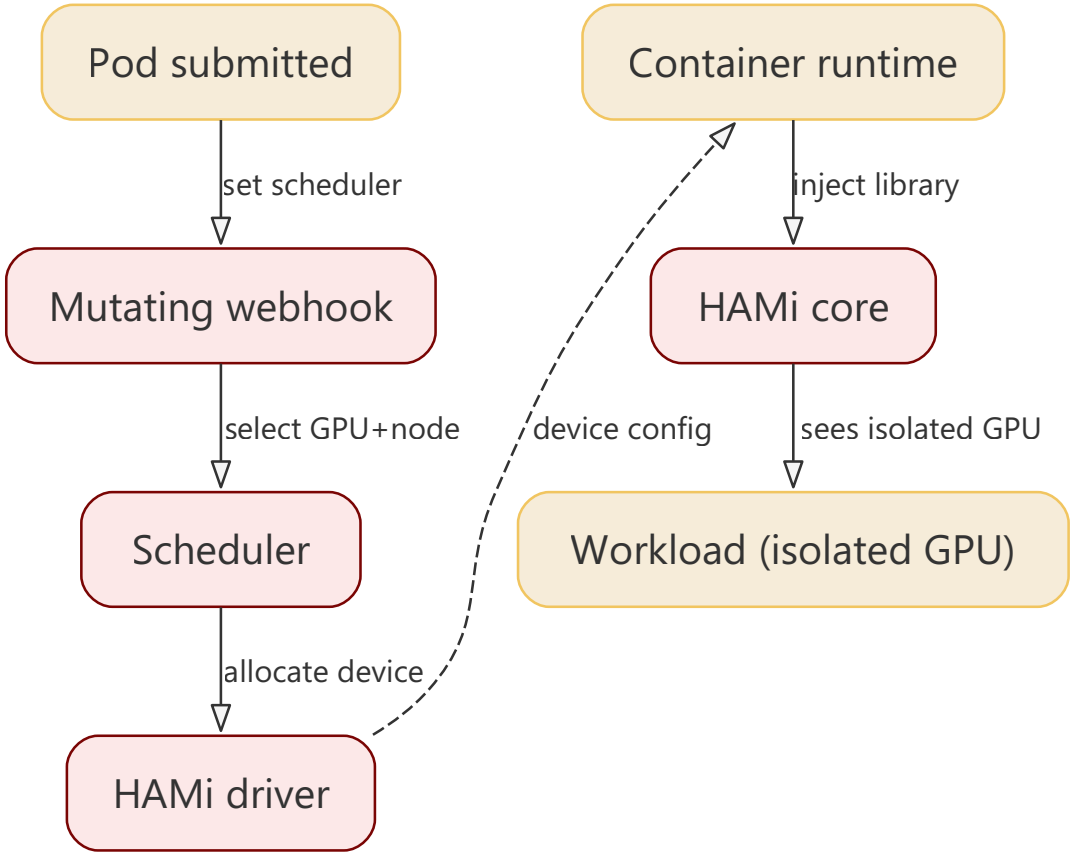
```yaml
# Ascend 910C: 8GB + 20% compute
resources:
  limits:
    huawei.com/Ascend910C: "1"
    huawei.com/Ascend910C-core: "20"
    huawei.com/Ascend910C-memory: "8192"
```

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

How HAMi Works

From pod submission to isolated GPU

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

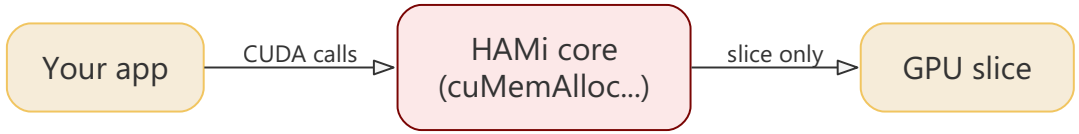


The Magic: CUDA Hijacking

Your app does not change

Your app calls CUDA. HAMi answers with a slice of the GPU.

- Small library loaded before your app (`LD_PRELOAD`)
- Intercepts CUDA calls, no app code changes
- No kernel modules, no driver changes
- Works for any framework: PyTorch, TensorFlow, vLLM



Why Memory Isolation Matters

One bad task must not kill its neighbors

⚠ Without HAMi

Tasks share a GPU with no borders. One greedy task eats all memory and kills the neighbors. Multi-tenant means risky.

With HAMi

Each task sees only its slice. Memory is a hard limit, enforced by the hijacked CUDA calls: every allocation is checked against your slice.

Oversubscription

Idle memory can be swapped to host RAM, so more models fit. Great for inference, not for active training.

Per-task limits

```yaml
resources:
  limits:
    nvidia.com/gpu: 1
    nvidia.com/gpumem: 3000
```

GPU Sharing Approaches

MIG vs HAMi vs NVIDIA DRA

Capability	MIG	HAMi	NVIDIA DRA
Sub-MIG slicing	✗	✓	✗
Dynamic repartition	✗	✓	✓
Multi-vendor	✗	✓	✗
Advanced scheduling	✗	✓	✗
No code changes	✗	✓	✗

HAMi differentiates with symbolic hijacking (1 MiB granularity on NVIDIA), consumable capacities for flexible requests, and multi-vendor support. HAMi-DRA builds on NVIDIA's upstream DRA driver and supports multiple DRA drivers.

DYNAMIA.AI
Creators of HAMi

Consumable Capacities

GPU resources as a pool you draw from

Memory and compute, separate axes

Request MiB of VRAM and percent of the GPU's SMs. What you use is what you get, no fixed profiles.

Scheduler packs by remaining capacity

Every GPU reports memory and compute left. Binpack and spread policies fit requests into the gaps.

Enforced on hijacked calls

HAMi-core caps every allocation at your slice. Inside the pod, nvidia-smi shows 0 MiB / 4000 MiB.

Same semantics on DRA (v2.8+)

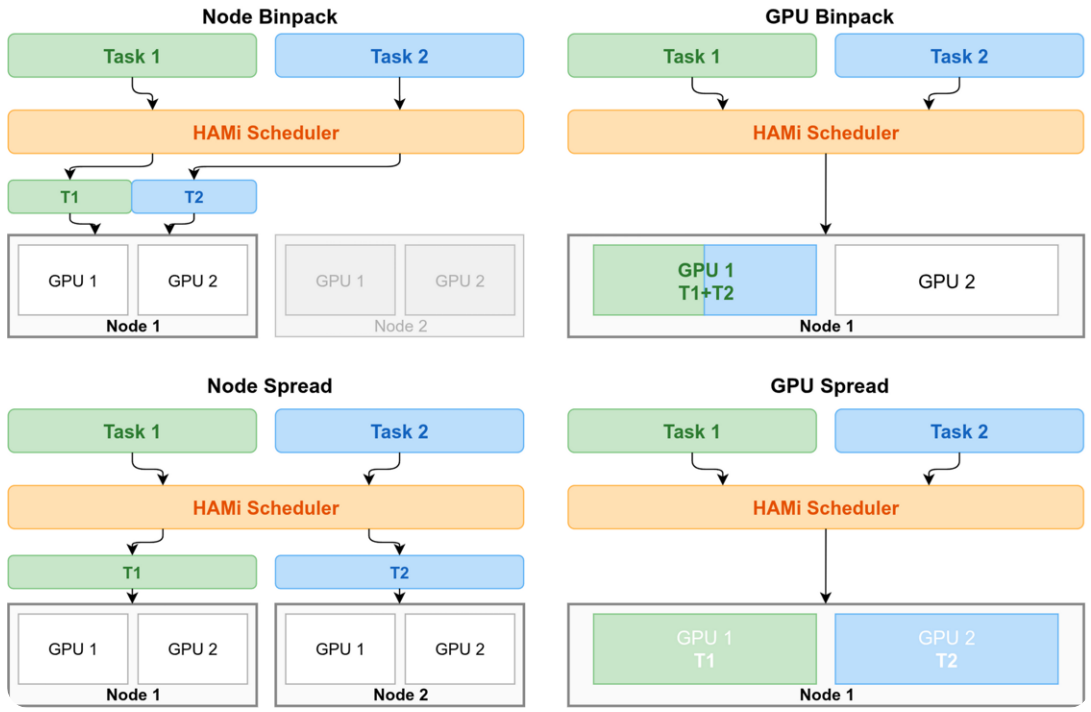
Typed ResourceSlice capacities (memory step 1 MiB, cores 0-100). Requests become ResourceClaims.

```
    nvidia.com/gpu: 1           # one vGPU slice
    nvidia.com/gpumem: 4000     # 4000 MiB of VRAM
    nvidia.com/gpucores: 30     # 30% of the GPU's SMs
```

Scheduling Policies

Binpack & Spread

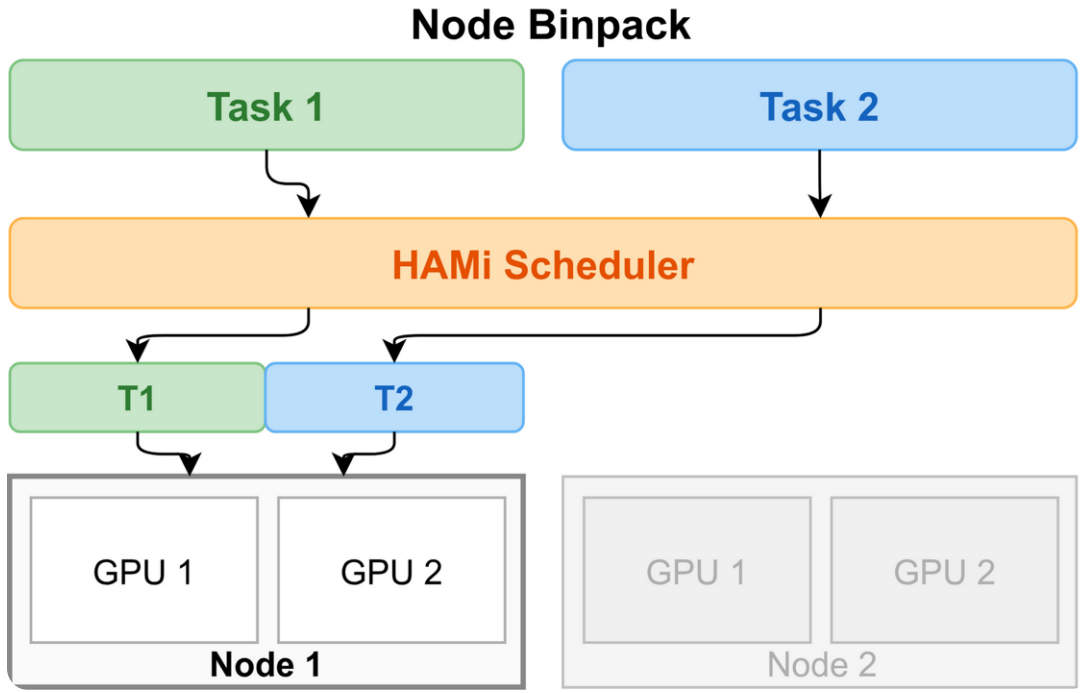
- **Node binpack** frees whole machines: reduces cost, helps cluster autoscaler
- **Node spread** isolates faults: HA across zones, blast radius control
- **GPU binpack** prevents fragmentation: frees entire GPUs for training
- **GPU spread** protects tail latency: reduces HBM and NVLink contention
-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



Node Binpack

Concentrate tasks, free whole machines

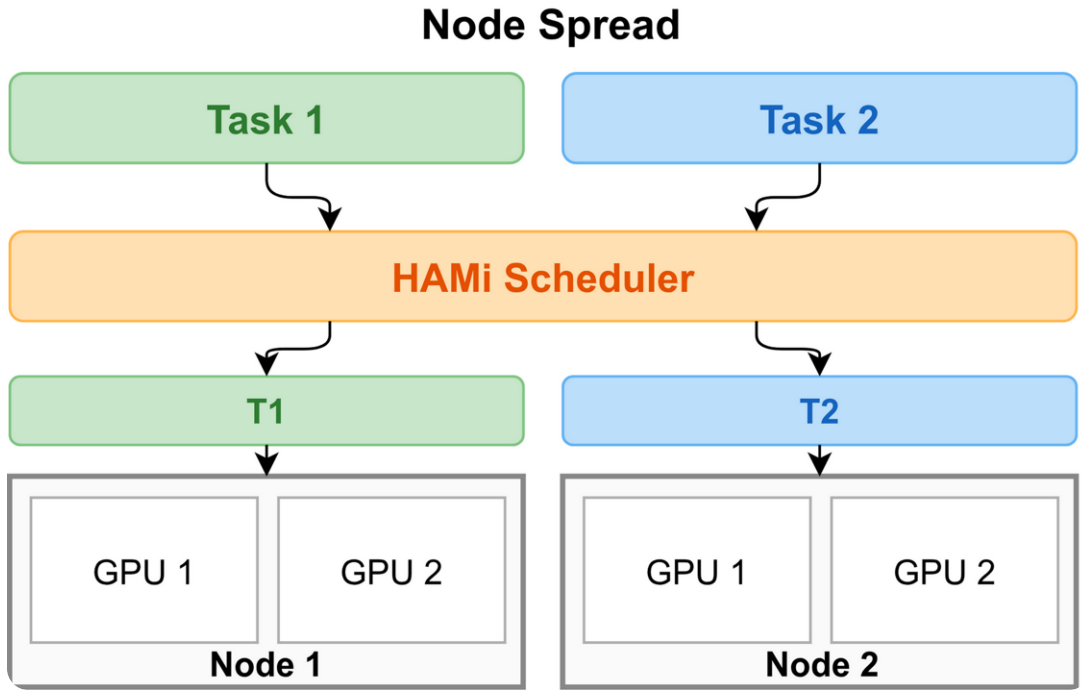
- Fills the fullest nodes first, emptiest last
- Frees whole machines: unused nodes stay idle, ready to power down
- Cuts cost: fewer active nodes, less power, smaller footprint
- Pairs with cluster autoscaling: empty nodes are removed automatically
- Best for: cost-sensitive clusters, batch and offline jobs, dev and test fleets



Node Spread

One task per node, faults stay local

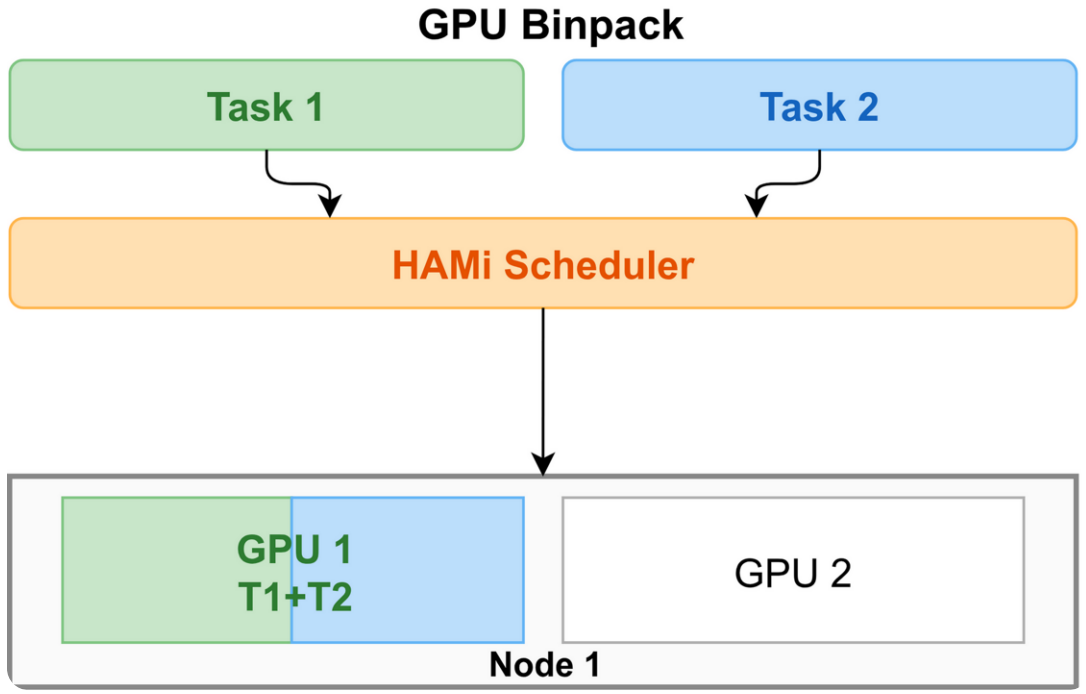
- One task per node: load balances across the cluster
- Fault isolation: a node crash or reboot takes down one task, not all
- Survives maintenance: node drains hit one replica at a time
- Use it for: production inference with SLAs. Losing a node costs one replica, not the whole model



GPU Binpack

Fill one GPU, free the rest

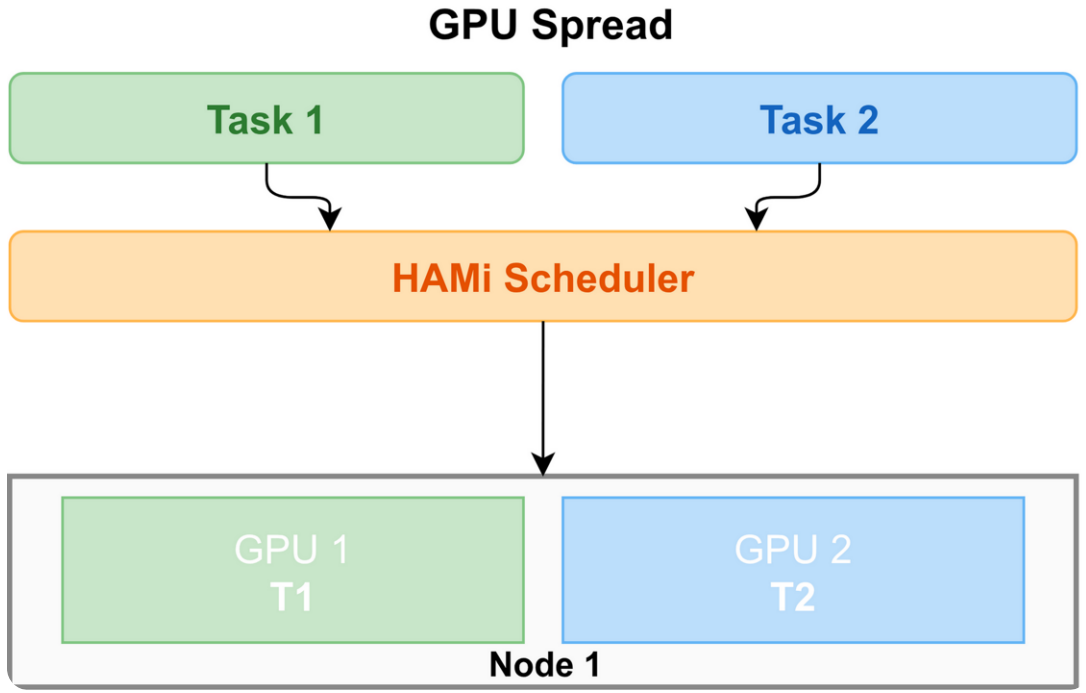
- Several tasks share one GPU, each with its own slice of memory and SMs
- Prevents fragmentation: scattered small tasks no longer block whole GPUs
- Frees entire GPUs for training jobs that cannot share
- Best for: fleets of small workloads - inference endpoints, batch jobs, dev and test



GPU Spread

One task per GPU, no shared bottlenecks

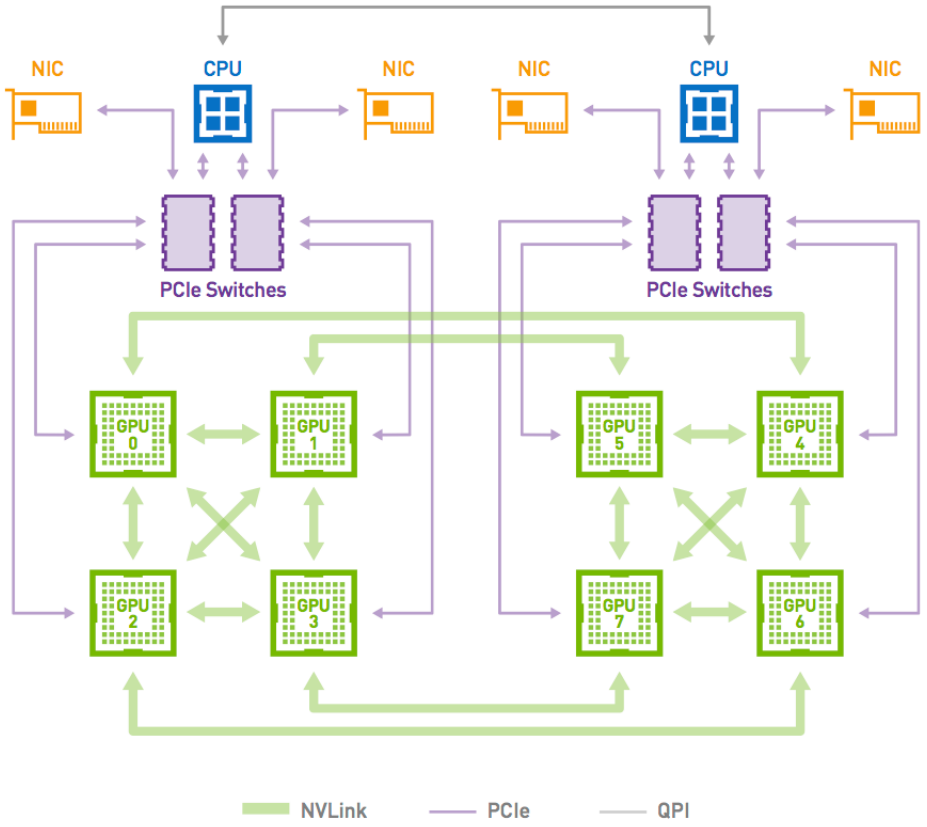
- One task per GPU: compute and HBM bandwidth are never shared
- Isolates noisy neighbors: a thrasher on one GPU does not slow the others
- Protects tail latency: no cross-tenant NVLink or HBM contention
- Best for: latency-sensitive serving with strict SLOs



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



Workload-Aware Scheduling

Upstream Kubernetes is catching up

- **What it is:** a Workload API so the scheduler treats a group of pods as one unit
- **Gang scheduling (v1.35, alpha):** all-or-nothing placement, no half-started training jobs
- **GPU scheduling (v1.36):** workload-aware placement on top of DRA
- **Today:** advanced policies (binpack, spread, topology) still live in plugins and tools like HAMi

Gotchas

Hardware limits and Kubernetes gaps

MIG is not available everywhere

MIG needs recent data-center GPUs and fixed profiles. Software slicing works on any device.

K8s GPU APIs are still young

DRA is stable, but advanced scheduling (binpack, spread, topology) is still missing from the platform.

DYNAMIA.AI
Creators of HAMi

Part 3: Production Use Cases

What real teams do with HAMi

Where HAMi Is Today

Production metrics from CNCF case studies

100%
Hardware pool utilization
Baike Holdings

10x
GPU utilization in CI
NIO

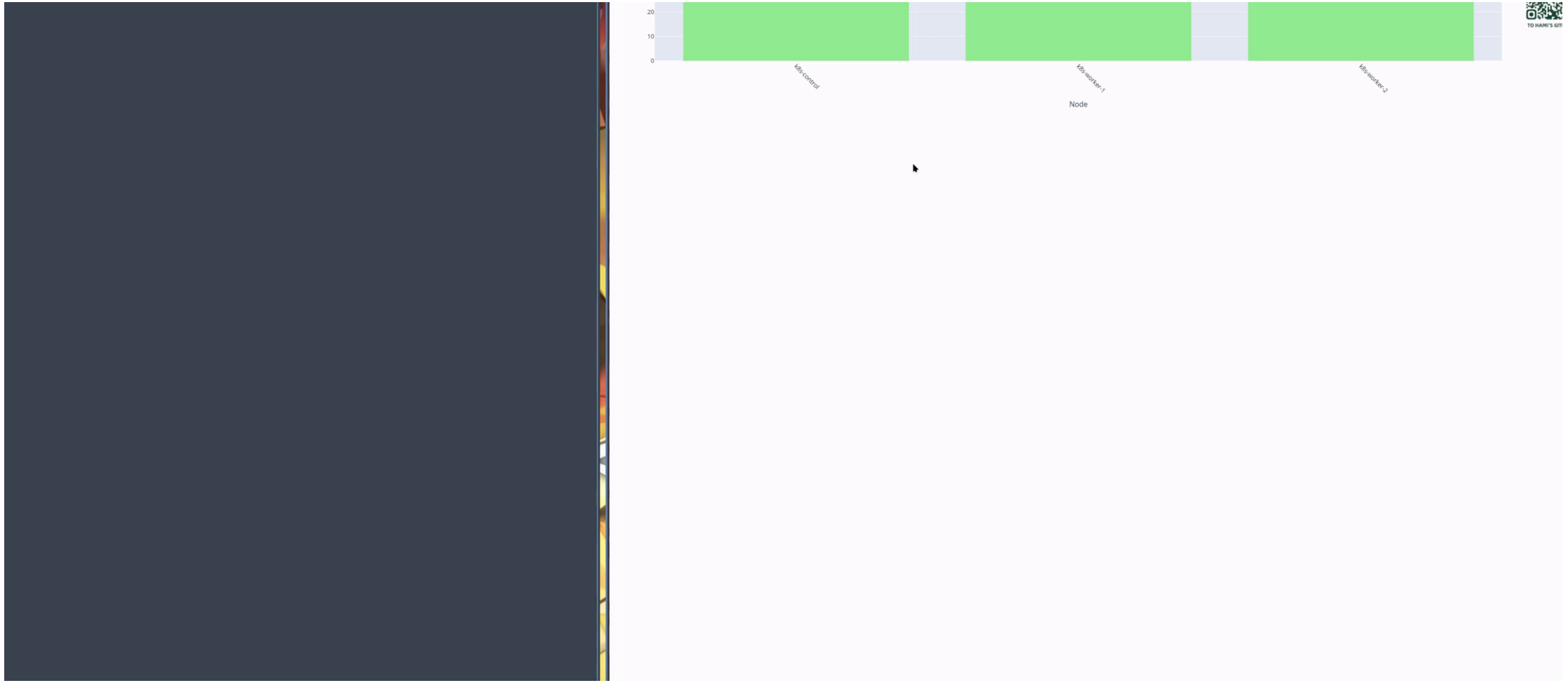
30%
Fewer GPU hours
China Merchants Bank

10,000+
Pods running concurrently
Baike Holdings

China Merchants Bank - SNOW Corp. - NIO - KE Holdings - DaoCloud - SF Technology - Prep Education - cncf.io/case-studies

Demo

3 nodes x 2 A100s: MIG, YOLO, and two vLLMs



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

Thank You

Questions? Try HAMi

github.com/Project-HAMi/HAMi

Reza Jelveh

Solution Architect, Dynamia AI - Makers of HAMi

github.com/fishman linkedin.com/in/rezajelveh