KUBECON + CLOUDNATIVECON NORTH AMERICA 2026 · CONTRIBFEST

DYNAMIA.AI
Creators of HAMi

GPU Virtualization: First Setup and Understanding the Device Plugin / DRA

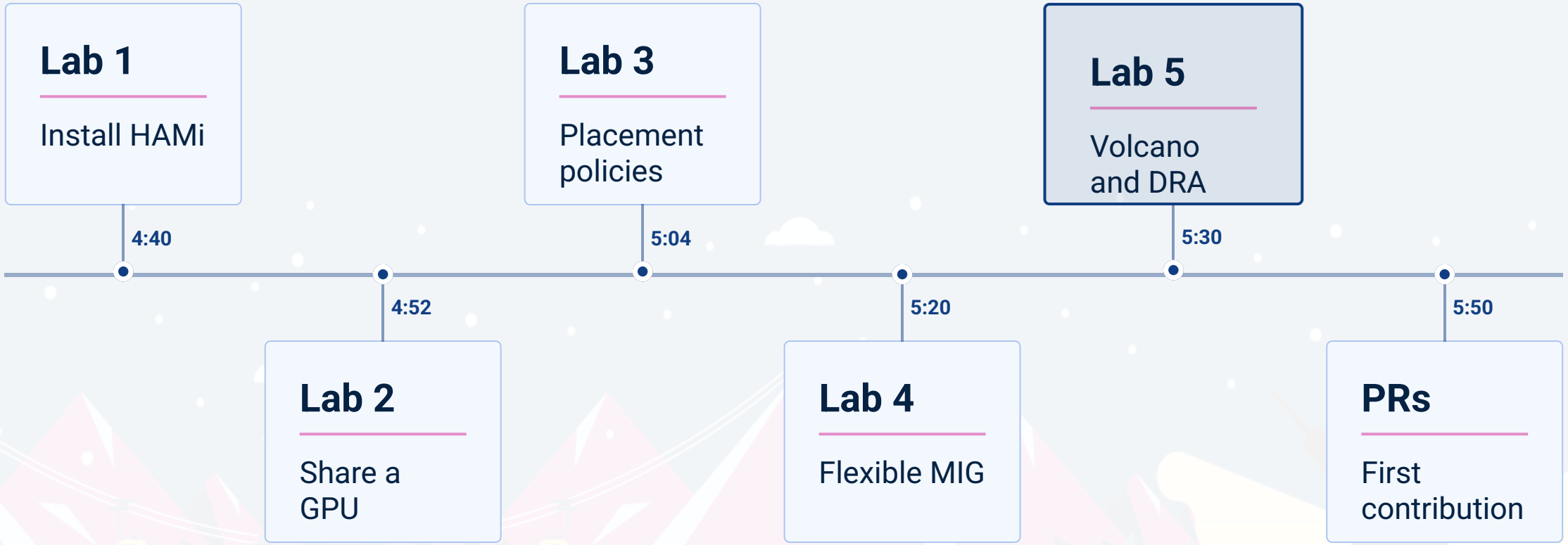
Reza Jelveh
Solution Architect, Dynamia AI - Makers of HAMi
github.com/fishman linkedin.com/in/rezajelveh

Mengxuan Li
CTO, Dynamia AI - HAMi maintainer



The 80 minutes

Tuesday, November 10, 4:40 to 6:00 pm · Salt Palace 255 D



Before the session

Everything runs on your laptop. The GPUs are fake.

```
git clone https://github.com/Project-HAMi/hami-workshop
cd hami-workshop/examples/containerdays-singapore
git clone https://github.com/fishman/hami-mock-monitor
docker pull kindest/node:v1.37.0 busybox:1.36 ghcr.io/llm-d/llm-d-inference-sim:v0.12.0
```

- Needs Docker (or OrbStack / Colima), 4 CPU and 8 GB free, plus `kind`, `kubectl`, `helm`, `jq`, `git`
- Go 1.27+ only to build HAMi or contribute code. A GitHub account for Lab 4
- Conference Wi-Fi is slow. Pull images at home

DYNAMIA.AI
Creators of HAMi

Why share GPUs

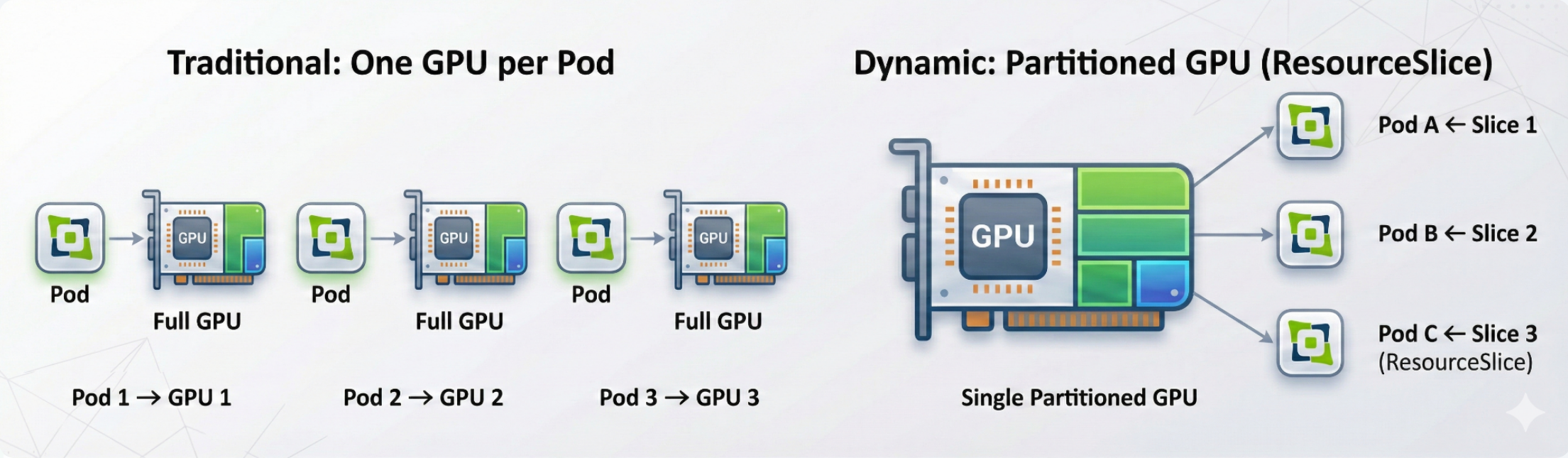


One GPU per task is the default

What it costs

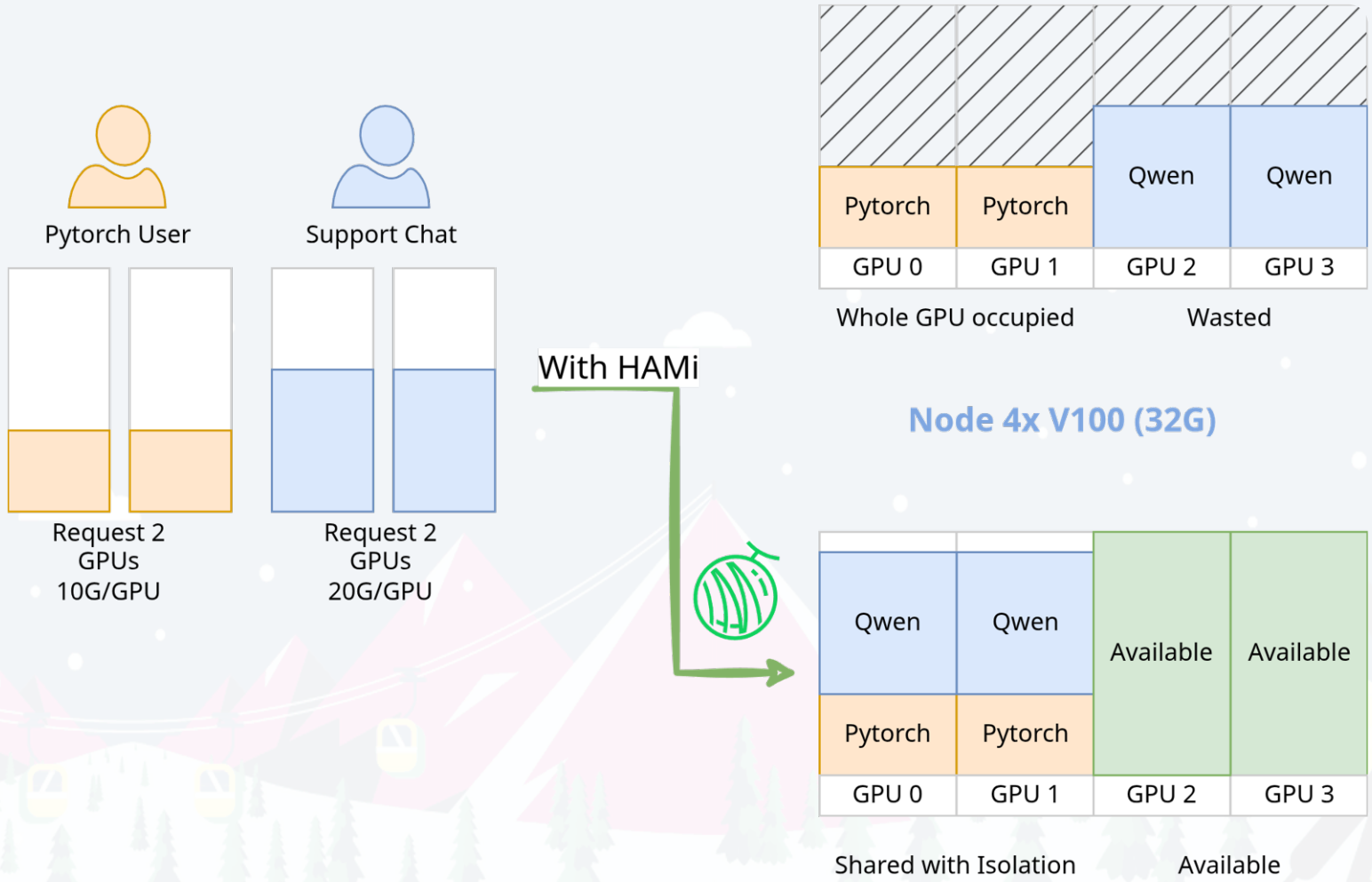
Kubernetes allocates GPUs atomically: one whole device, one task.

- A 1 GB task blocks an 80 GB device
- Most GPUs sit idle most of the time
- No advanced scheduling yet: no binpack, no spread, no topology



What is HAMi

One GPU, many tasks



DYNAMIA.AI
Creators of HAMi

HAMi Capabilities

Six things HAMi brings to GPU scheduling

Heterogeneous Management

Manage GPU, NPU, MLU, TPU 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.

Asking for a slice

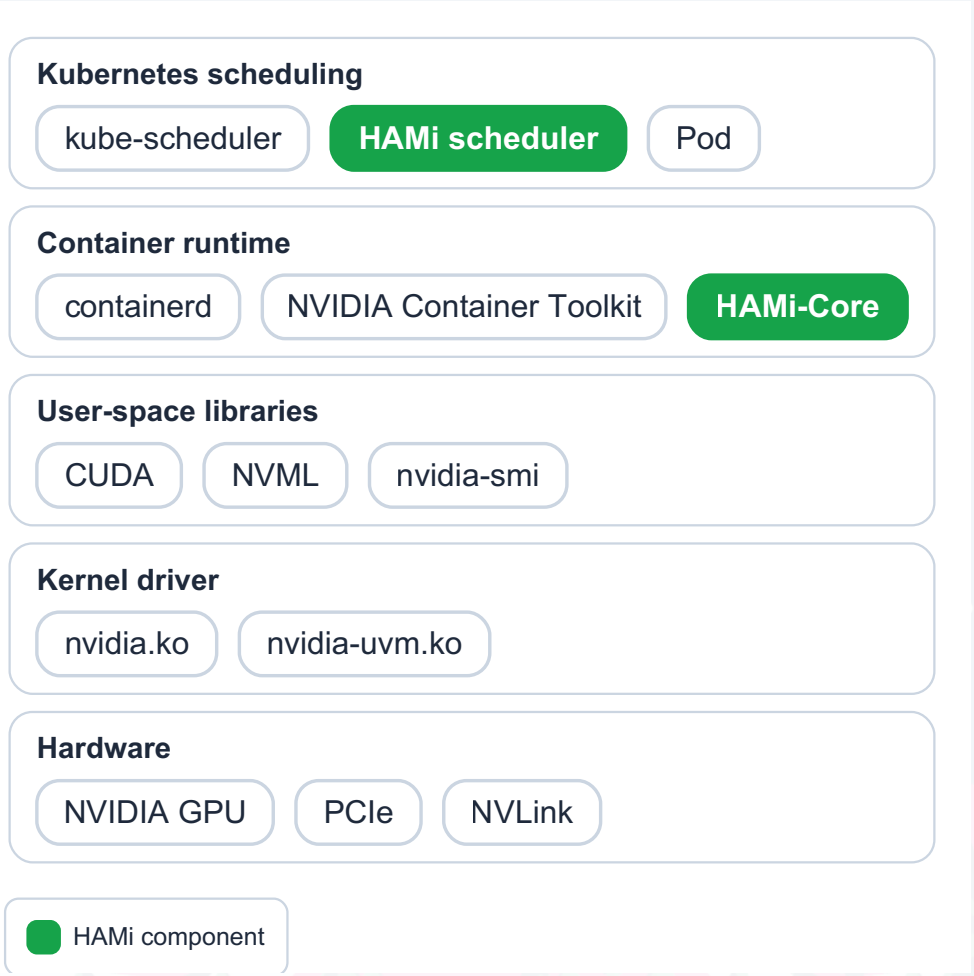
One API across vendors, no code changes

- **NVIDIA, Ascend, Cambricon, Hygon, Iluvatar** and more
- **Fine-grained:** memory as small as 1 MB, compute as small as 1% of a card
- **Transparent to tasks:** no code changes required
- **Hard resource isolation** inside containers

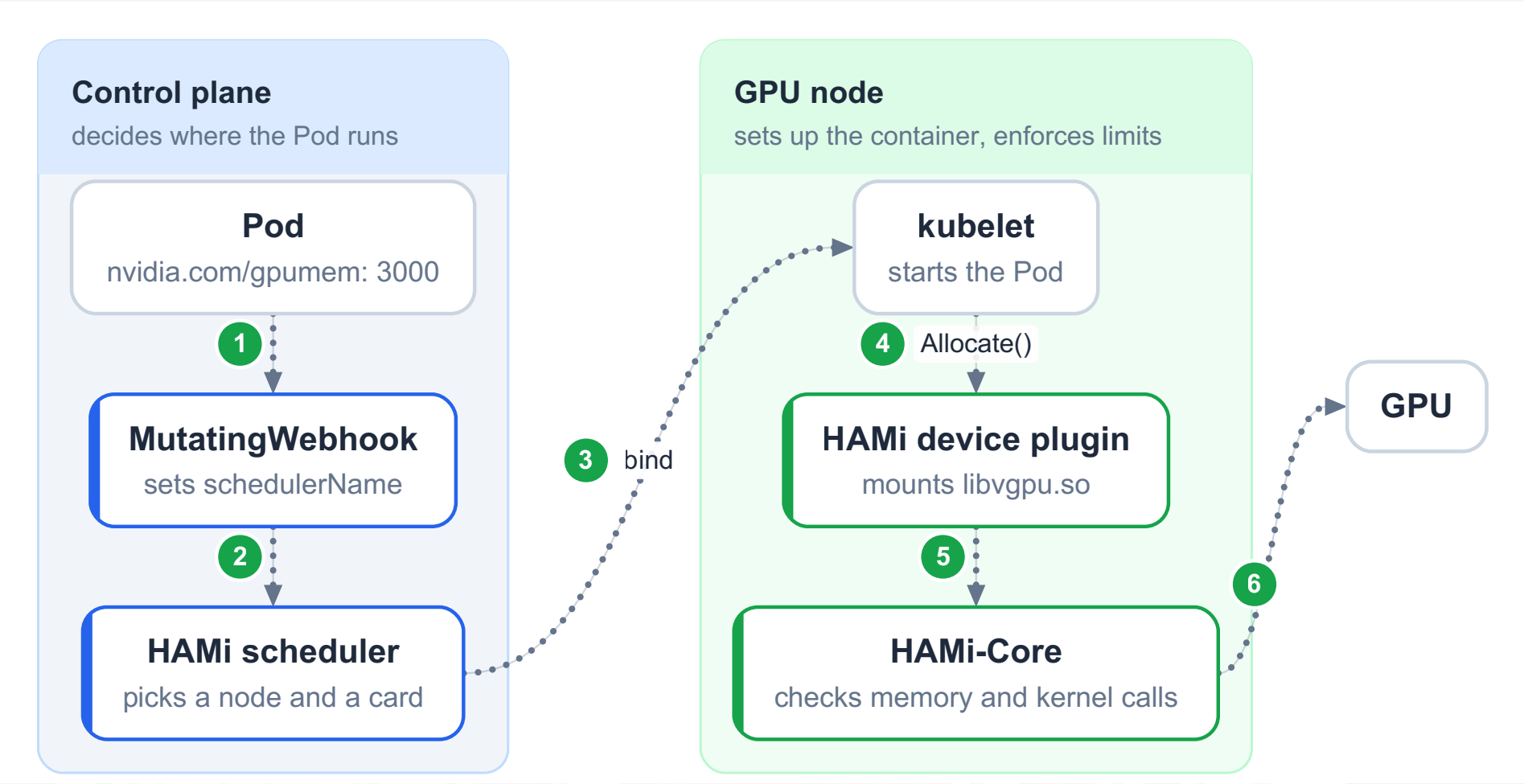
```yaml
# NVIDIA: 15 GB and 30% of one card
resources:
  limits:
    nvidia.com/gpu: 1
    nvidia.com/gpumem: 15000
    nvidia.com/gpucores: 30
```

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

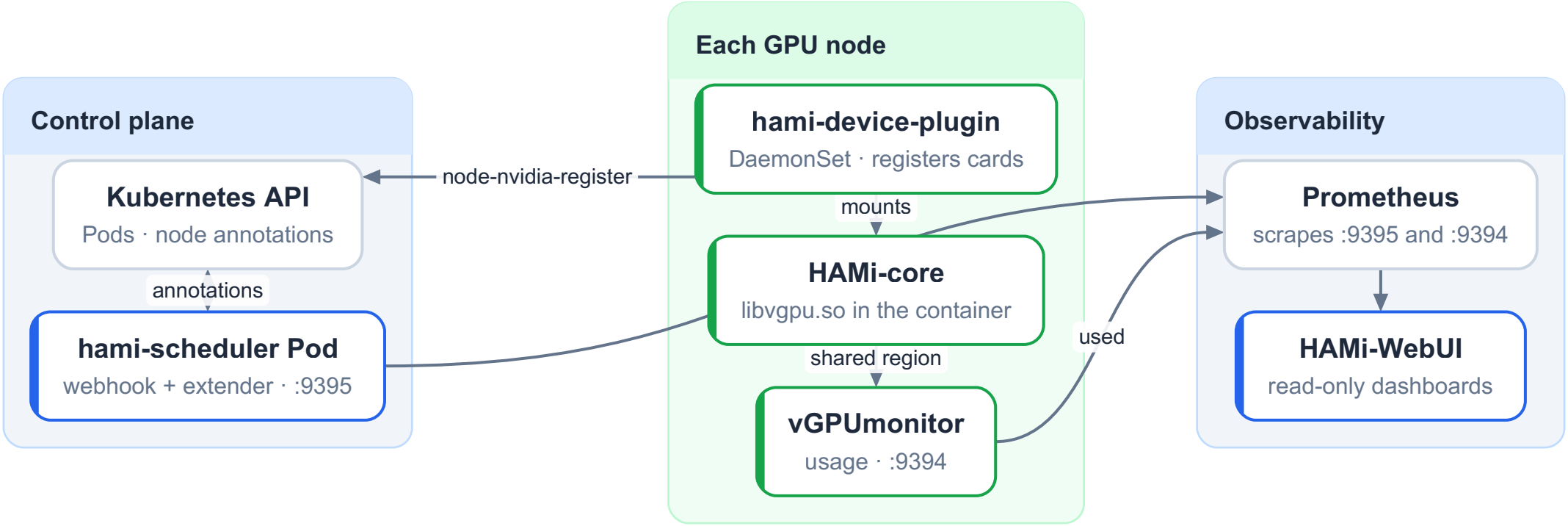
Where HAMi sits in the GPU stack



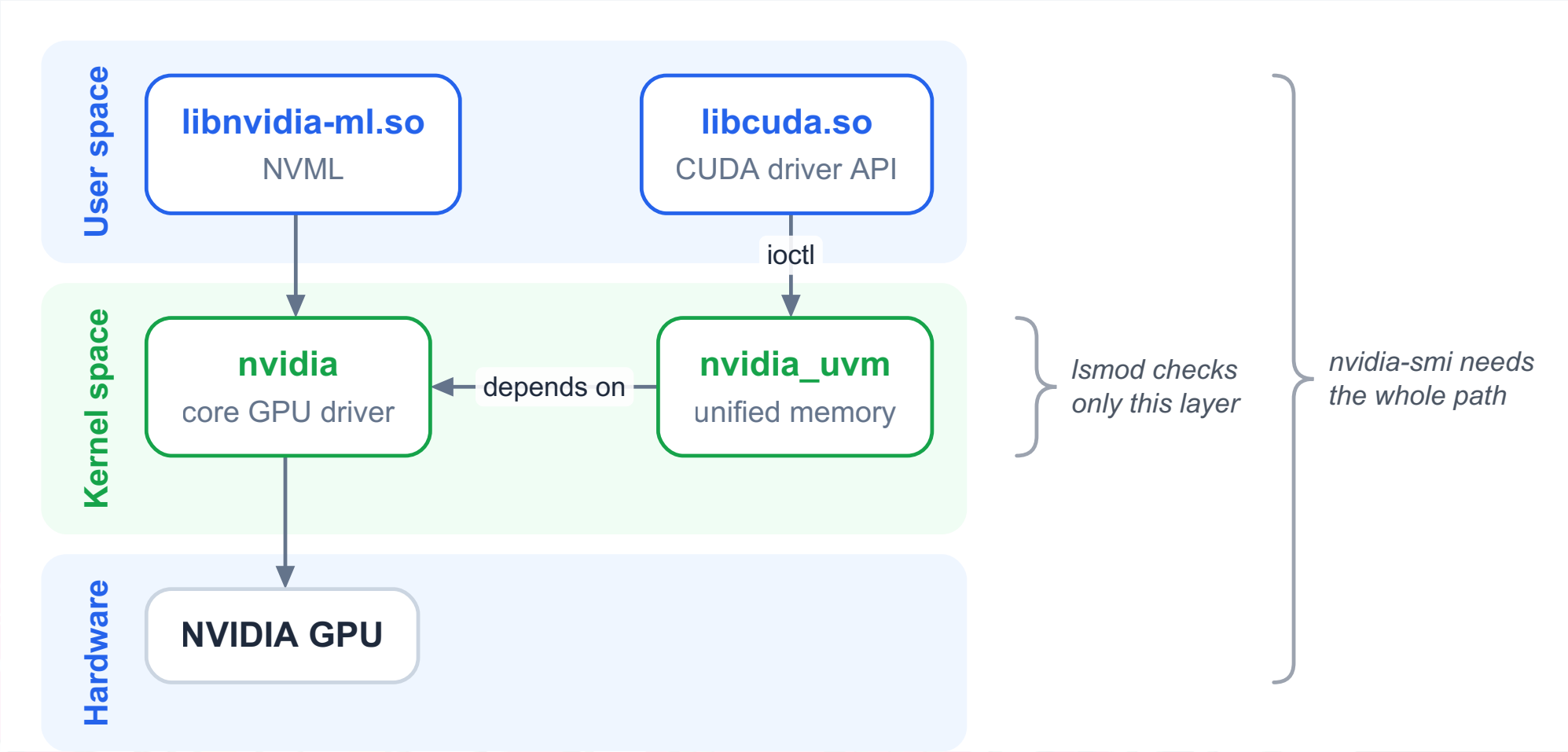
A Pod's path to the GPU



How the HAMi components work together

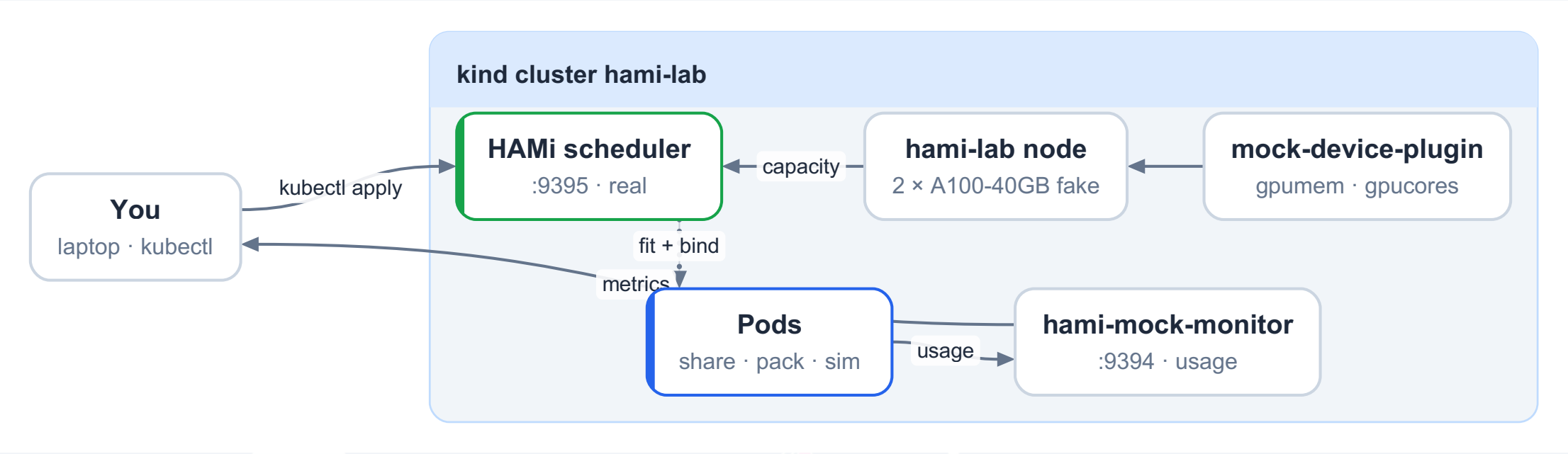


Why `nvidia-smi` can fail when `lsmod` looks fine



HAMi-Core intercepts at the CUDA and NVML layer, above the kernel driver.

Lab setup: real scheduler, fake GPUs



mock-device-plugin

Turns a node annotation into `nvidia.com/gpumem` and `nvidia.com/gpucores` capacity. The real device plugin writes that annotation from NVML.

hami-mock-monitor

Produces HAMi's usage metrics (normally from vGPUmonitor and HAMi-core) from a load source instead of a GPU.

DYNAMIA.AI
Creators of HAMi

Lab 1: Install HAMi



Lab 1: one command

```
./setup.sh
```

The script:

- Creates a kind cluster `hami-lab` and labels the node gpu=on
- Installs HAMi 2.11.0 with the real device plugin off and the mock plugin on
- Fakes two A100-40GB cards. Two pieces:the count resource `nvidia.com/gpu=20`: the mock plugin's health gate, 2 cards x 10 splits
- the node annotation `hami.io/node-nvidia-register` from `fake-gpus.json`, what the real device plugin writes after it reads NVML
- Builds and installs hami-mock-monitor with `completeBinds=true`

Expected at the end:

```
{ "nvidia.com/gpu": "20", "nvidia.com/gpucores": "200",
  "nvidia.com/gpumem": "81920", "nvidia.com/gpumem-percentage": "200" }
```

Lab 1: look around

```
kubectl -n kube-system get pods | grep hami
kubectl -n kube-system get cm hami-scheduler-device -o yaml | less
kubectl get node hami-lab-control-plane \
  -o jsonpath='{.metadata.annotations.hami\.io/node-nvidia-register}' | jq
```

```
[{"id":"GPU-mock-0","index":0,"count":10,"devmem":40960,"devcore":100,
  "type":"NVIDIA-A100-SXM4-40GB","health":true,"numa":0,"mode":"hami-core"}, ...]
```

- `count`: how many Pods may share the card. `devmem` in MB, `devcore` in percent
- The card list lives on the node, written by the device plugin. The scheduler reads it at `Filter`

DYNAMIA.AI
Creators of HAMi

Lab 2: Share a GPU



Lab 2: three Pods, two cards

```
kubectl apply -f share.yaml -f too-big.yaml
kubectl get pods -o wide
```

`share-1..3` each ask for 15000 MB and 30% of a card, and run. `too-big` asks for 45000 MB, more than any 40960 MB card, and stays `Pending`:

```
kubectl get events --field-selector involvedObject.name=too-big | tail -2
# 0/1 nodes are available: 1 2/2 CardInsufficientMemory.
```

45000 MB fits the node total, but no single 40960 MB card.

Lab 2: read what HAMi decided

```
for p in share-1 share-2 share-3; do
  echo -n "$p "; kubectl get pod $p -o jsonpath='{.metadata.annotations.hami\.io/vgpu-devices-allocated}{"\n"}'
done
# share-1 GPU-mock-1,NVIDIA,15000,30:;
# share-2 GPU-mock-0,NVIDIA,15000,30:;
# share-3 GPU-mock-1,NVIDIA,15000,30:;
```

The format is `uuid,type,memory_mb,cores:` per device, `;` per container. The device plugin reads it in `Allocate`.

The interception chain

Four steps, one node

- The admission **webhook** rewrites `schedulerName` to `hami-scheduler`
- The scheduler **extender** at `Filter` reads the node annotation `hami.io/node-nvidia-register` (the card list from Lab 1), checks memory and core headroom per card, and picks a physical card UUID
- At `Bind` it writes `hami.io/vgpu-devices-allocated` onto the Pod, sets `hami.io/bind-phase=allocating` and takes the node lock `hami.io/mutex.lock`
- The device plugin's `Allocate` mounts `/dev/nvidia*` and the vGPU files, reading the target from the Pod annotation, then marks `success` and releases the lock

> *The mock plugin skips step 4's completion, which is why hami-mock-monitor's `completeBinds` exists. Without it the next Pod waits up to 5 minutes for the lock.*

On a real node, step 4 also injects `libvgpu.so` via `ld.so.preload`, and HAMi-core enforces the memory and core limits from that moment on.

DYNAMIA.AI
Creators of HAMi

Lab 3: Placement policies



Binpack and spread

Two axes, two policies, one annotation each

GPU binpack

Several workloads fill one card, each with its own memory and compute slice. Keeps whole cards free for the job that cannot share.

Best for: small workloads, batch, dev and test.

GPU spread

One workload per card. No shared compute or memory bandwidth, so a noisy neighbour stays on its own card.

Best for: latency-sensitive serving.

Axis	Annotation	binpack	spread
Which node	`hami.io/node-scheduler-policy`	fewer nodes, autoscaler frees machines	across nodes, smaller blast radius
Which card	`hami.io/gpu-scheduler-policy`	same card	across cards

Lab 3: binpack

```
kubectl delete -f share.yaml -f too-big.yaml
kubectl apply -f binpack.yaml
for p in pack-1 pack-2; do
  echo -n "$p "; kubectl get pod $p -o jsonpath='{.metadata.annotations.hami\.io/vgpu-devices-allocated}{"\n"}'
done
```

Both Pods report the same `GPU-mock-X`: two workloads sharing one card. Frees whole cards for big jobs.

Lab 3: spread

```
kubectl delete -f binpack.yaml
kubectl apply -f spread.yaml
for p in spread-1 spread-2; do
  echo -n "$p "; kubectl get pod $p -o jsonpath='{.metadata.annotations.hami\.io/vgpu-devices-allocated}{"\n"}'
done
```

One Pod per card. Same cluster, same requests, different policy, different placement.

```
metadata:
  annotations:
    hami.io/gpu-scheduler-policy: spread   # or binpack, per Pod
```

Mutex: start on an idle card

`hami.io/gpu-scheduler-policy: mutex`, new in HAMi v2.10.0 (#2011)

- A filter: only cards with zero tenants are eligible. One small workload on a card is enough to rule it out
- No idle card: the Pod stays `Pending` with `ExclusiveDeviceAllocateConflict`
- Checked **at placement only**. A later plain Pod can still join the card

When it is necessary

- Latency-sensitive inference that must not share a die, but should not have to request the whole card
- Spread is only a preference: it takes the least-used card, even a busy one
- `mutex,topology-aware`: idle cards first, then the best NVLink-linked set among them
- For exclusivity over the Pod's lifetime, request all of the card's memory and cores, or pin it with `nvidia.com/use-gpuuuid`

Lab 3: mutex

```
kubectl apply -f spread.yaml          # spread-1 and spread-2: one Pod per card
kubectl apply -f mutex.yaml           # asks for an idle card
kubectl get events --field-selector involvedObject.name=mutex-1 | tail -1
# 0/1 nodes are available: 1 2/2 ExclusiveDeviceAllocateConflict.
kubectl delete pod spread-2           # free one card
kubectl get pod mutex-1 -o jsonpath='{.metadata.annotations.hami\.io/vgpu-devices-allocated}'
# GPU-mock-0,NVIDIA,15000,30:;
```

Both cards have room for `mutex-1`'s memory, but neither is empty. It waits until one is.

Since v2.10.0 policies chain: filters first (`mutex`, `topology-aware`), then sort keys in the order written (`binpack`, `spread`, `numa`). `mutex,binpack,numa`: an idle card, then the best-packed of those, NUMA as the tiebreaker.

Where topology information comes from

Every vendor: the node's device plugin reads it, an annotation carries it, the scheduler scores it



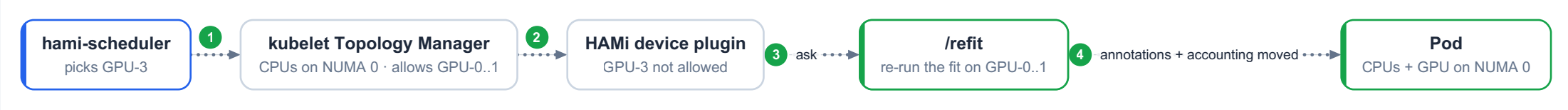
Topology, vendor by vendor

- **NVIDIA:** `ENABLE_TOPOLOGY_SCORE=true` scores every card pair from NVML into `hami.io/node-nvidia-score`. Policy `topology-aware`: one card gets the weakest-linked card, N cards get the best-linked set
- **AMD:** `xgmiPeers` and numa per GPU in `hami.io/node-amd-register`. Pod annotations `amd.com/numa-bind`, `amd.com/link-bind`, or policy `topology-aware`
- **MetaX:** the vendor plugin writes `metax-tech.com/gpu.topology.scores` and `.losses`. MetaXLink first, then PCIe switch
- **Kunlunxin, AWS Neuron:** no link data; the layout is fixed, so the scheduler reads it from card index (NUMA halves of 8, Neuron rings)

```yaml
metadata:
  annotations:
    hami.io/gpu-scheduler-policy: topology-aware
```

CPU and GPU on the same NUMA node

The scheduler picks the GPU, kubelet picks the CPUs. They have to agree.



- `enableNumaTopology`: the device plugin advertises each card's NUMA node, so the Topology Manager can restrict the choice
- When that excludes the scheduler's pick, the plugin calls the scheduler's `/refit`: same policy, allowed cards only, reservation moved
- Before the refit, the container ran on one GPU while HAMi's annotations and accounting recorded another

NUMA alignment: turning it on

```yaml
# kubelet config
cpuManagerPolicy: static
topologyManagerPolicy: single-numa-node
topologyManagerScope: container
```

```yaml
metadata:
  annotations:
    hami.io/numa-alignment: best-effort   # or strict
```

- Helm: `devices.nvidia.enableNumaTopology`, `devicePlugin.numaRefit.enabled`, and per node `enablegetpreferredallocation`. All off by default
- `best-effort`: refit fails, kubelet's choice stands and is logged. `strict`: refit fails, the allocation fails
- Not the same as `nvidia.com/numa-bind`, which keeps a multi-GPU Pod's cards on one NUMA node. Policy `numa` sorts by NUMA node in a chain such as `numa,binpack`

DYNAMIA.AI
Creators of HAMi

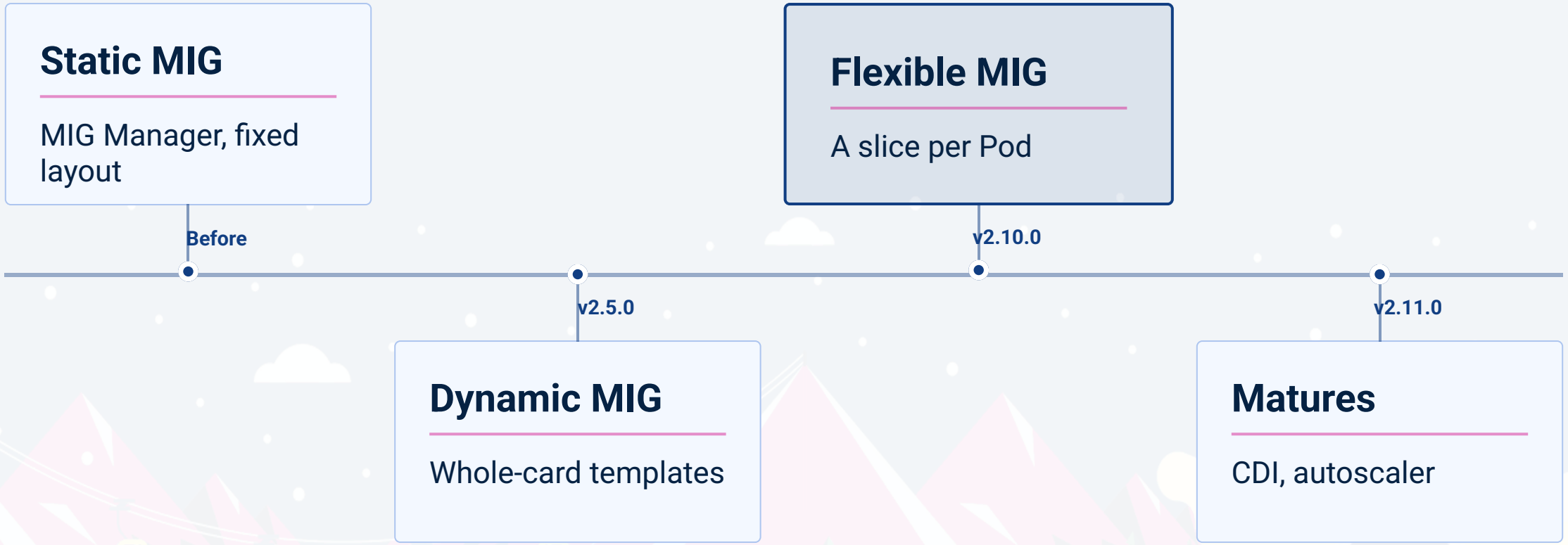
Lab 4: Flexible MIG

MIG slices per Pod, no whole-GPU templates



MIG in HAMi

From pre-carved layouts to a slice per Pod



v2.5.0: Dynamic MIG

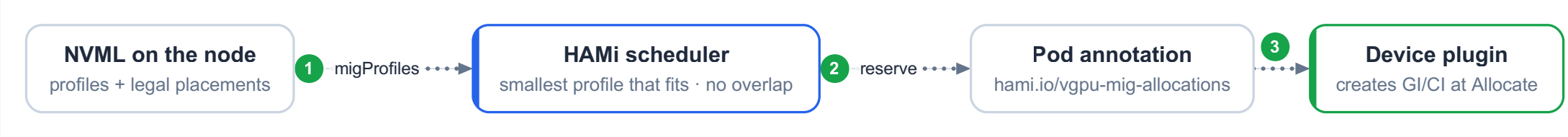
Templates, switched by the device plugin

- `knownMigGeometries` lists whole-GPU templates per model, with each profile's memory and count, e.g. A100-40GB `2g.10gb ×3 + 1g.5gb ×1`
- The scheduler takes the **first template that fits** the job; the device plugin applies it with mig-parted
- Same API as HAMi-core: `nvidia.com/gpu` + `nvidia.com/gpumem`, `nvidia.com/vgpu-mode: mig` to force a MIG node

The catch: a template covers the whole card. A new profile mix means switching the template, so the card has to be empty first

v2.10.0: Flexible MIG

NVML says what is possible, the scheduler reserves, the device plugin creates



- Profiles and legal placements come from NVML; `migProfileAllowlist` only says which profiles are allowed
- The scheduler picks the smallest allowed profile that fits the `gpumem` request, at a placement that does not overlap
- The reservation lives in a Pod annotation, so it survives device-plugin restarts. The instance is removed when the Pod ends

v2.10.0: a Flexible MIG Pod

DYNAMIA.AI
Creators of HAMi

```
# node config: "operatingmode": "mig"
# device config: which profiles are allowed, not their sizes
nvidia:
  migProfileAllowlist:
    - models: ["A100-SXM4-40GB"]
      profiles: ["1g.5gb", "2g.10gb", "3g.20gb", "7g.40gb"]
```

```
metadata: { annotations: { nvidia.com/vgpu-mode: "mig" } }
# limits: { nvidia.com/gpu: 1, nvidia.com/gpumem: 8000 }
```

8000 MiB gets the smallest allowed profile that fits, 2g . 10gb. Same Pod spec as Lab 2, hard isolation.

v2.11.0: Flexible MIG matures

- **v2.10.0 limits:** MIG didn't work with CDI device injection; multi-GPU MIG not yet validated
- **v2.11.0:** a CDI spec per MIG instance (#3070), placement scoring with backtracking (#2855), `nvidia.com/mig-profile-preference` per Pod (#3014), Cluster Autoscaler support via `reportNodeCapacity` (#2874)
- **Pick MIG** for hard isolation: separate memory, cache and fault domains. **Pick HAMi-core** for 1 MiB slices and oversubscription
- **Still true:** free slices but no legal contiguous placement leaves the Pod `Pending`; HAMi never moves a running instance. Ampere or newer

DYNAMIA.AI
Creators of HAMi

Lab 5: Volcano and DRA

Gang scheduling on shared GPUs, then a different resource model



Volcano: gang jobs on shared GPUs

```yaml
kind: Job                    # batch.volcano.sh/v1alpha1
spec:
  schedulerName: volcano
  minAvailable: 2            # both workers or none
  queue: vgpu-queue
  tasks:
  - replicas: 2              # each container limits:
      # volcano.sh/vgpu-number: 1, vgpu-memory: 2000, vgpu-cores: 30
```

- Volcano `deviceshare` (`VGPUEnable: true`) plus the volcano-vgpu-device-plugin. No HAMi scheduler
- HAMi-core still isolates inside the container
- Gang scheduling and per-team queue `capability` on fractional GPUs

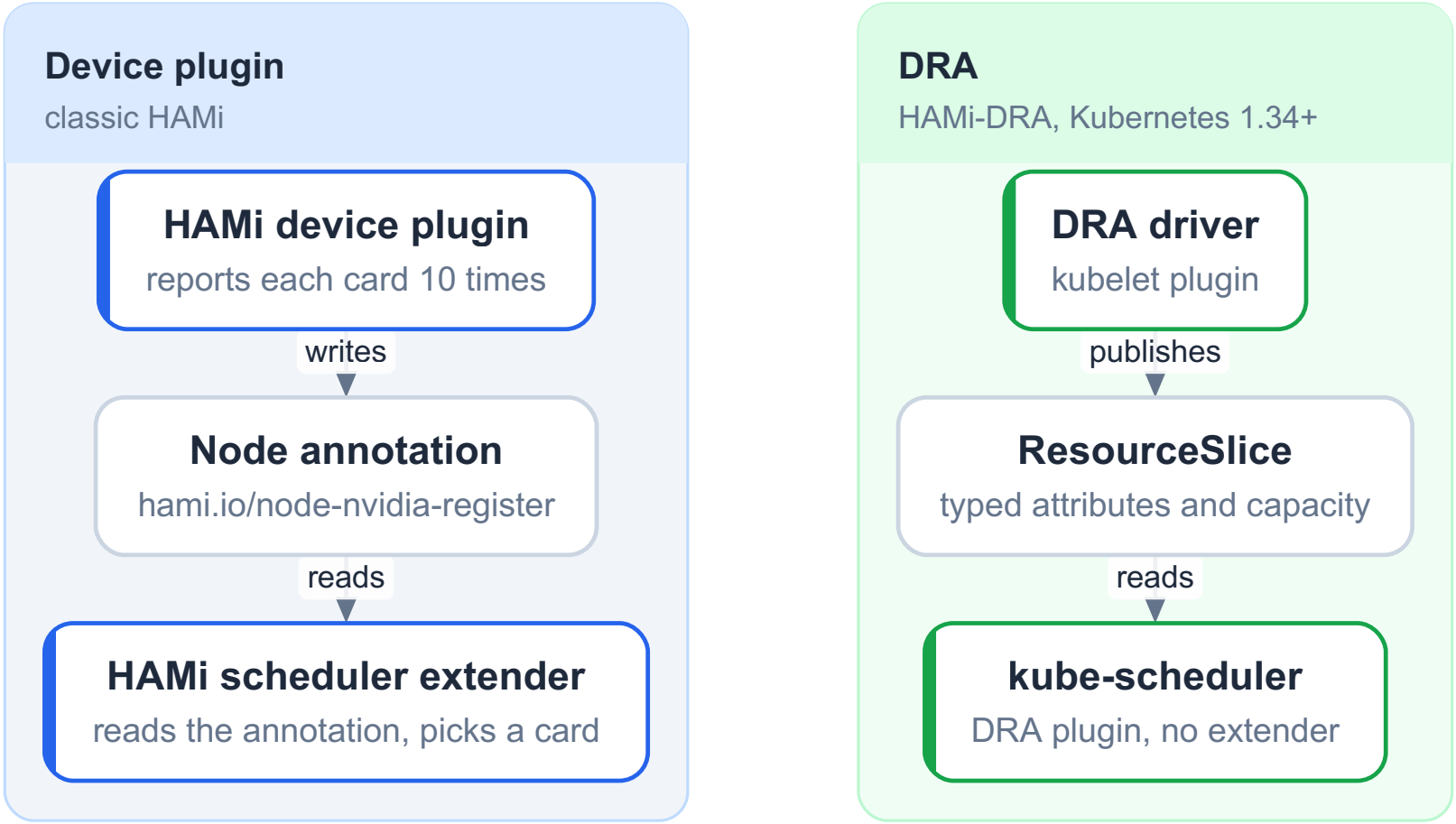
Volcano: install the vGPU plugin

Project-HAMi/volcano-vgpu-device-plugin

```
# volcano-scheduler-configmap, namespace volcano-system
- name: deviceshare
  arguments:
    deviceshare.VGPUEnable: true
    deviceshare.SchedulePolicy: binpack   # or spread
```

```
git clone https://github.com/Project-HAMi/volcano-vgpu-device-plugin
cd volcano-vgpu-device-plugin   # no Helm repo published yet
helm install volcano-vgpu-device-plugin deployments/helm/volcano-vgpu-device-plugin \
  --set nvmlMock.enabled=true   # no GPUs: scheduling and registration only
kubectl get node <node> -o jsonpath='{.status.allocatable}'   # volcano.sh/vgpu-number, vgpu-memory
```

Device plugin vs DRA



The four DRA objects

DYNAMIA.AI
Creators of HAMi

ResourceSlice

Published by the DRA driver. Lists each device with its attributes and capacity.

DeviceClass

A named set of devices, selected with CEL. HAMi: `hami-core-gpu.project-hami.io`

ResourceClaim

A Pod's request for devices, like a PersistentVolumeClaim.

ResourceClaimTemplate

Kubernetes creates one ResourceClaim per Pod from it.

Consumable capacity: one card, many claims

DYNAMIA.AI
Creators of HAMi

```yaml
capacity:
  cores:
    value: "100"
    requestPolicy: { validRange: { min: "0", max: "100", step: "1" } }
  memory:
    value: 15Gi
    requestPolicy: { validRange: { min: 1Mi, max: 15Gi, step: 1Mi } }
```

- A Tesla T4 published as one device with two capacities, plus `allowMultipleAllocations: true`
- The scheduler does the accounting and enforces min, max and step. Extended resources accept any number
- Feature gate `DRAConsumableCapacity`: alpha in 1.34, on by default since 1.36

A claim for 4Gi and 30 cores

```yaml
apiVersion: resource.k8s.io/v1
kind: ResourceClaim
metadata: { name: single-gpu-0, namespace: test-dra }
spec:
  devices:
    requests:
    - name: single-gpu
      exactly:
        deviceClassName: hami-core-gpu.project-hami.io
        capacity:
          requests: { cores: 30, memory: "4Gi" }
```

```
| N/A   59C    P8   16W /  70W |   0MiB /   4096MiB |   0%   Default |
```

`nvidia-smi` in the Pod reports 4096 MiB: HAMi-Core enforces the claim. A second Pod's claim lands on the same card, same UUID, its own `shareID`.

HAMi-DRA: keep the HAMi Pod spec

DYNAMIA.AI
Creators of HAMi



- NVIDIA, Hygon HCU, AMD and Ascend in one deployment (`deviceVendors`)
- `resourceClaimTemplate: true` lets Kueue admit the Pods. `batch/v1` Jobs are not covered yet
- Needs Kubernetes 1.34+, `DRAConsumableCapacity`, and CDI in containerd or CRI-O

github.com/Project-HAMi/HAMi-DRA

DRA and the schedulers above it

DYNAMIA.AI
Creators of HAMi

Kueue

Admits ResourceClaimTemplates. Needs the HAMi DeviceClass in `deviceClassMappings`.

Volcano

Today: `deviceshare` plus the vGPU device plugin. Gang and queue capacity on shared GPUs.

Open questions for the room: who decides partitioning, the driver or the scheduler? How do quotas count fractional devices?

New in v2.11.0: Remote GPU

CUDA on a node with no GPU, served over the network by Lupine

- **Client nodes** need no GPU and no driver: an init container delivers the Lupine client libraries and `nvidia-smi`, HAMi injects `LUPINE_SERVER`
- **GPU nodes** with `operatingmode: remote` serve Lupine instead of advertising local devices. Pools by Node label, picked with `hami.io/remote-server`
- `nvidia.com/remote-gpu` reserves a whole server; `nvidia.com/remote-gpu-memory` caps memory per GPU through HAMi-core

```
command: ["vllm", "serve", "Qwen/Qwen3-8B"]   # on a GPU-less node
resources: { limits: { nvidia.com/remote-gpu: 2, nvidia.com/remote-gpu-memory: 20480 } }
```

New in v2.11.0: AMD Instinct sharing

The NVIDIA resource model, hard limits in the container

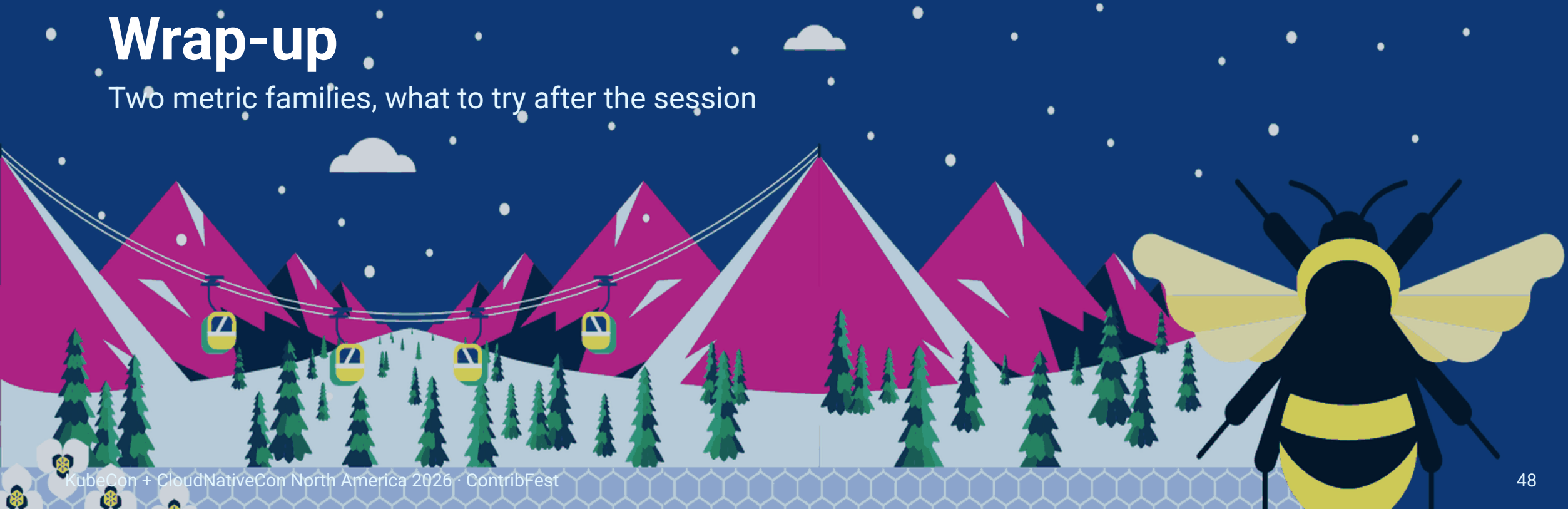
- **Memory:** `amd.com/gpumem` or `amd.com/gpumem-percentage`, enforced by amd-hami-core
- **Compute:** `amd.com/gpucores` sets the CU share through `HSA_CU_MASK`
- **Placement:** `amd.com/numa-bind`, XGMI-linked cards with `amd.com/link-bind`, and `binpack`, `spread`, `mutex`, `topology-aware`
- **Node side:** amd-device-plugin v0.0.4 adds SPX/DPX/QPX/CPX compute partitions; amd-dra-driver brings the same limits to DRA

```
resources: { limits: { amd.com/gpu: 1, amd.com/gpumem: 24000, amd.com/gpucores: 50 } }
```

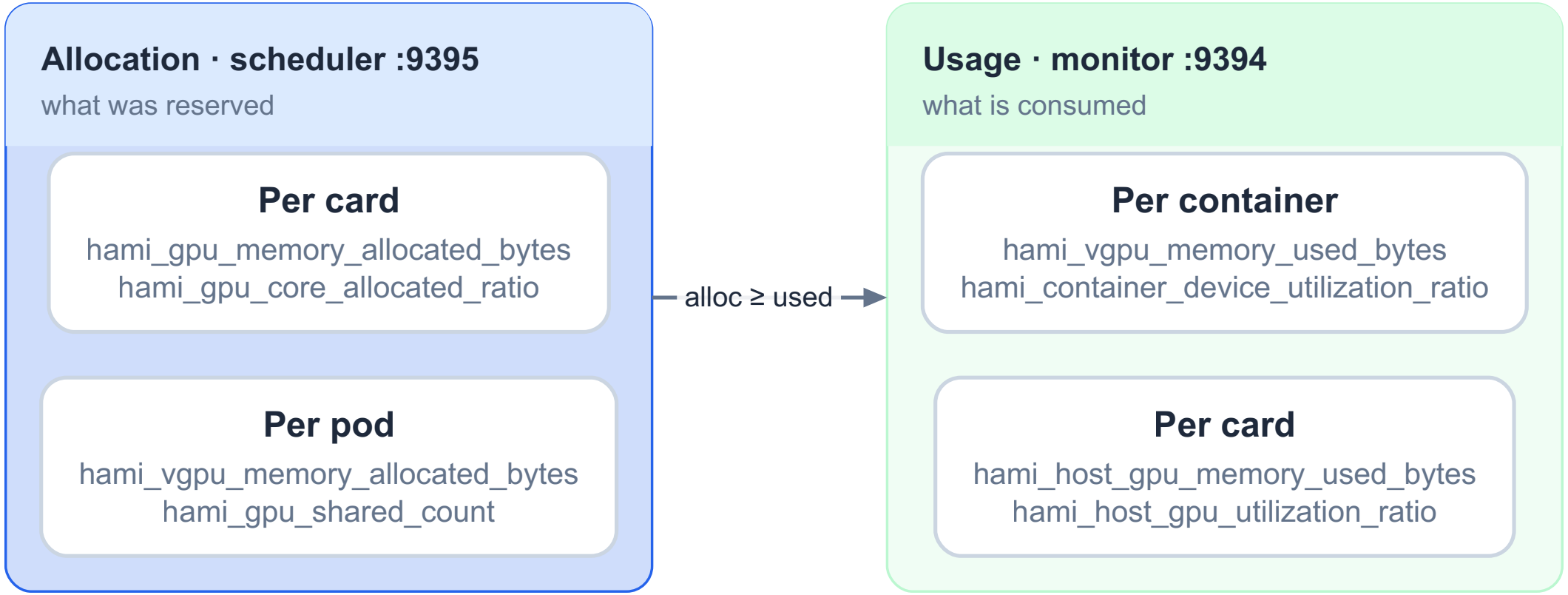
DYNAMIA.AI
Creators of HAMi

Wrap-up

Two metric families, what to try after the session



Reserved is not used




```
./metrics.sh scheduler   # allocation, port 9395
./metrics.sh monitor     # usage, port 9394
```

Give the mock a realistic load

```
kubectl apply -f sim.yaml      # llm-d-inference-sim, requests oscillate 0 to 5
watch -n 5 "./metrics.sh monitor | grep 'pod=\"sim\"'"
```

```
hami_container_device_utilization_ratio{pod="sim"} 26.4
hami_node_gpu_memory_allocated_ratio{device_uuid="GPU-mock-1"} 0.73
hami_host_gpu_utilization_ratio{device_uuid="GPU-mock-1"}      20.0
```

- `sim` is a vLLM-compatible inference simulator: utilization moves between 0 and 33 (load x allocated cores)
- Memory stays at 0.9 x the limit, because vLLM preallocates its KV cache

Question. Card `GPU-mock-1` shows 73% memory allocated but 20% used. Which number goes on a capacity dashboard, and which on a cost report?

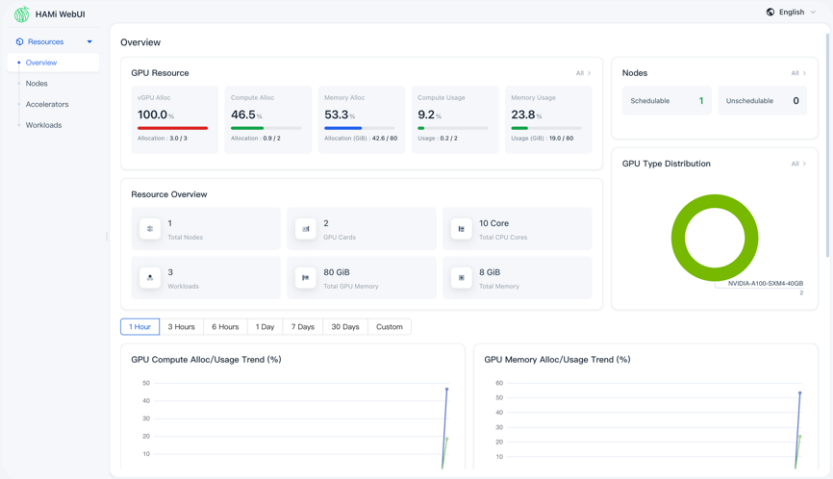
After the session: HAMi WebUI

`metrics.sh` is fine for a terminal, but the HAMi WebUI turns the same numbers into dashboards. It needs a Prometheus in the cluster:

```
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
helm repo add hami-webui https://project-hami.github.io/HAMi-WebUI
helm upgrade --install prometheus prometheus-community/kube-prometheus-stack \
  --version 75.15.1 -n monitoring --create-namespace --set grafana.enabled=false --wait
kubectl apply -f monitoring.yaml   # scrape :9395 and :9394
kubectl apply -f webui-rules.yaml  # translate metric names for the WebUI
helm upgrade --install my-hami-webui hami-webui/hami-webui \
  --set externalPrometheus.enabled=true --set dcgm-exporter.enabled=false \
  -n kube-system --wait
kubectl -n kube-system port-forward svc/my-hami-webui 3000:3000
```

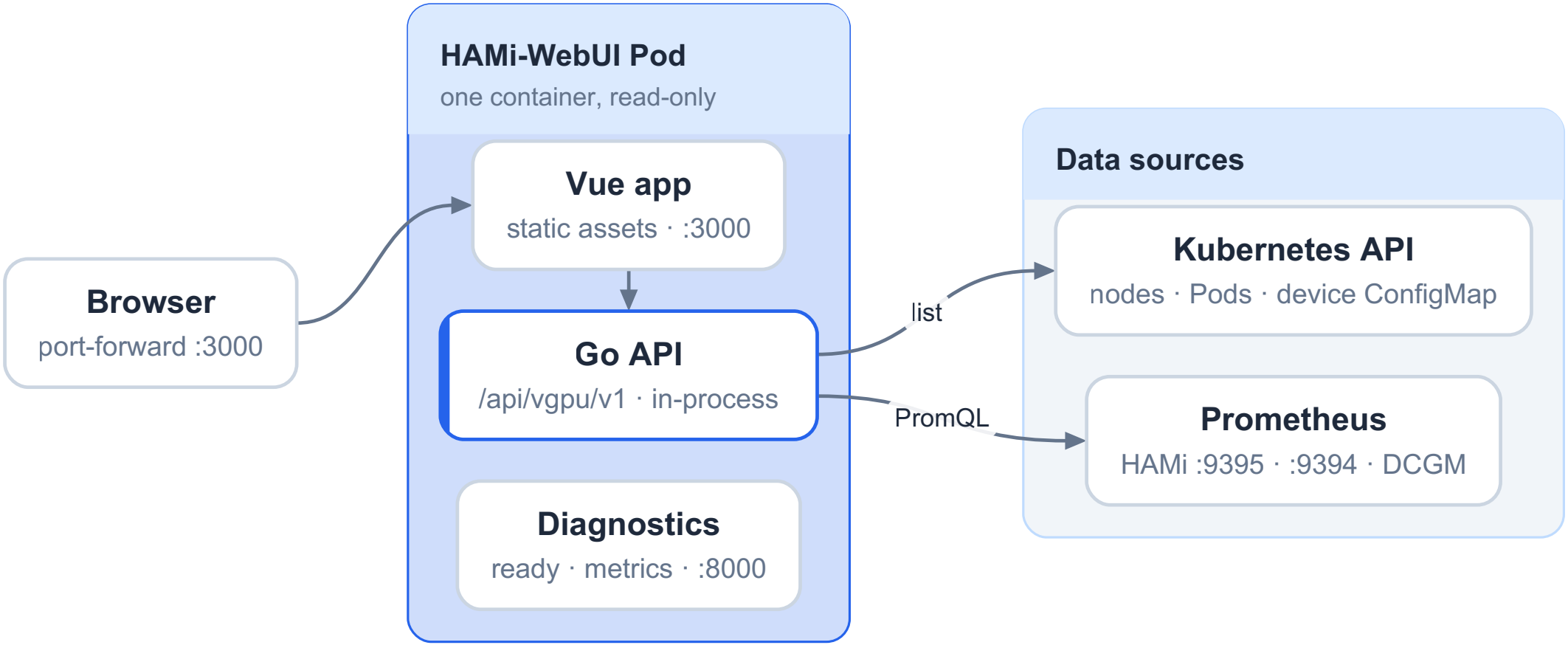
Open http://localhost:3000.

Why `webui-rules.yaml`



- The WebUI queries vGPUmonitor's metric names (`hami_memory_size`, `hami_core_util`, `hami_container_vgpu_allocated`, ...)
- This lab's scheduler and mock monitor expose the newer byte/ratio names (`hami_gpu_memory_limit_bytes`, `hami_host_gpu_utilization_ratio`, ...)
- The PrometheusRule records one set as the other. Without it every allocation and usage panel stays empty

How the WebUI integrates



Hack on the WebUI

```
git clone https://github.com/Project-HAMi/HAMi-WebUI && cd HAMi-WebUI
make bootstrap && make -C server bootstrap   # pnpm deps, Go generators (+ protoc)
make -C server run   # Go API on :8000, uses kubeconfig + server/config/config.yaml
make dev             # Vite on :3000, proxies /api/vgpu/v1 to :8000
make verify && make -C server verify         # before the PR
```

- Point `server/config/config.yaml` at the lab's Prometheus (port-forward `:9090`)
- Go, Node and pnpm versions are pinned in the repo. `git commit -s`

After the session: contribute

DYNAMIA.AI
Creators of HAMi

HAMi

`make test`. Start in `Fit`, `PatchAnnotations` (`pkg/device/nvidia/device.go`) or `Filter`, `Bind` (`pkg/scheduler/scheduler.go`). `git commit -s`.

HAMi-DRA

The webhook that turns `nvidia.com/gpu` requests into DRA ResourceClaims. Needs Kubernetes 1.34+.

mock-device-plugin

Complete the bind in `Allocate` (a patch is prepared in `hami-mock-monitor/upstream/mock-device-plugin`), or add a vendor.

hami-mock-monitor

Other vendors (Ascend, Hygon), a Grafana dashboard for the lab, a published image.

Look for good `first issue`. Open a draft PR before you leave. Disclose AI help in the PR.

Clean up

```
kind delete cluster --name hami-lab
```

Where HAMi Is Today

Production metrics from CNCF case studies

DYNAMIA.AI
Creators of HAMi

0

Driver, kernel, code changes needed
SF Technology

1-2 GB

Memory per small-model inference service
KE Holdings

4x

GPU utilization: 5-10% to 30-50%
NIO

90%

GPU infra managed by HAMi on RTX 4070/4090
Prep Edu

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

DYNAMIA.AI
Creators of HAMi

Community & Adopters

Devices, integrations, and who uses HAMi

Open Source, CNCF Backed, Production Ready

5k	700k	500+	27
Github Stars	Docker Pulls	Contributors	Contributor Countries

kubernetes VOLCANO Kueue koordinator KAI Scheduler COZYSTACK

Ecosystem & Device Support

NVIDIA AMD Ascend Cambricon 寒武纪 HYGON 中科海光 天数智芯 Iluvatar CoreX METAX 沐曦 摩尔线程 MOORE THREADS 昆仑芯 KUNLUNXIN Enflame 燧原科技 壁仞科技 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 顺丰科技 思特奇 SI-TECH SNOW viettel

DYNAMIA.AI
Creators of HAMi

Thank you

github.com/Project-HAMi/HAMi · project-hami.io · lab: github.com/Project-HAMi/hami-workshop

Reza Jelveh
Global Market & Solution Architecture, Dynamia AI
github.com/fishman linkedin.com/in/rezajelveh
alias tzcentral="timedatectl set-timezone US/Central"

Mengxuan Li
CTO, Dynamia AI - HAMi maintainer

