

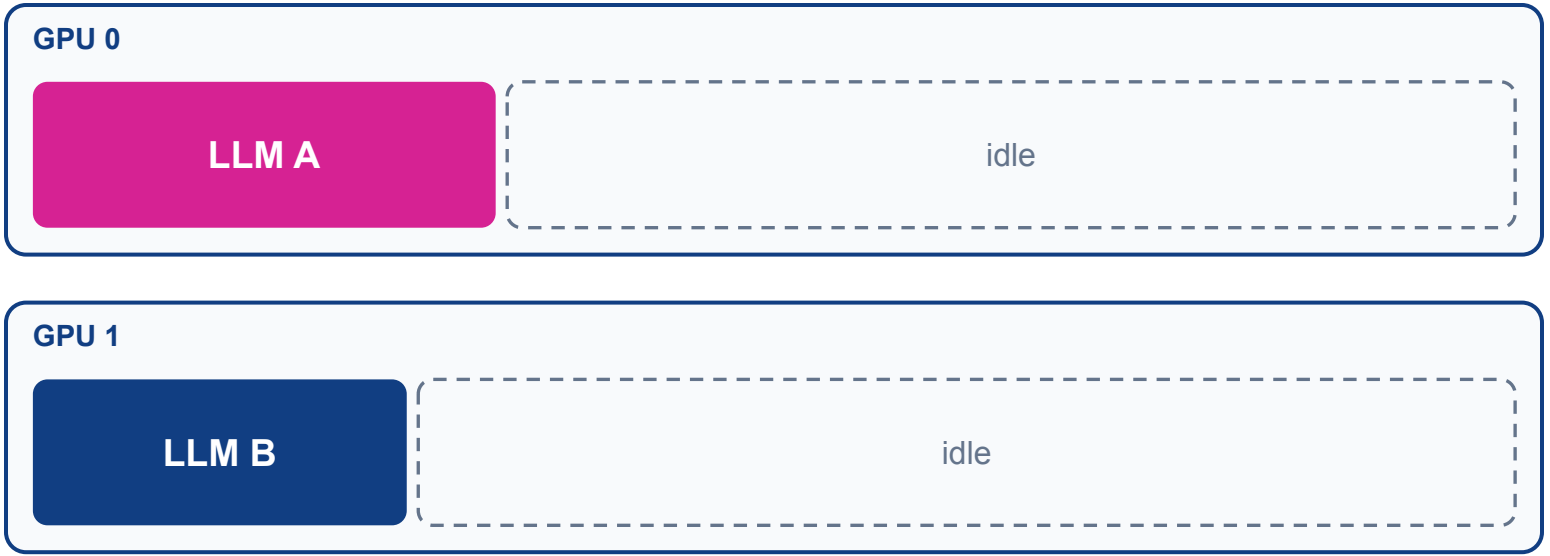
Fixing GPU Time-Slicing on Kubernetes:
Dynamic, Memory-Safe Sharing for AI Inference

Julien Semaan



Two small models, two GPUs

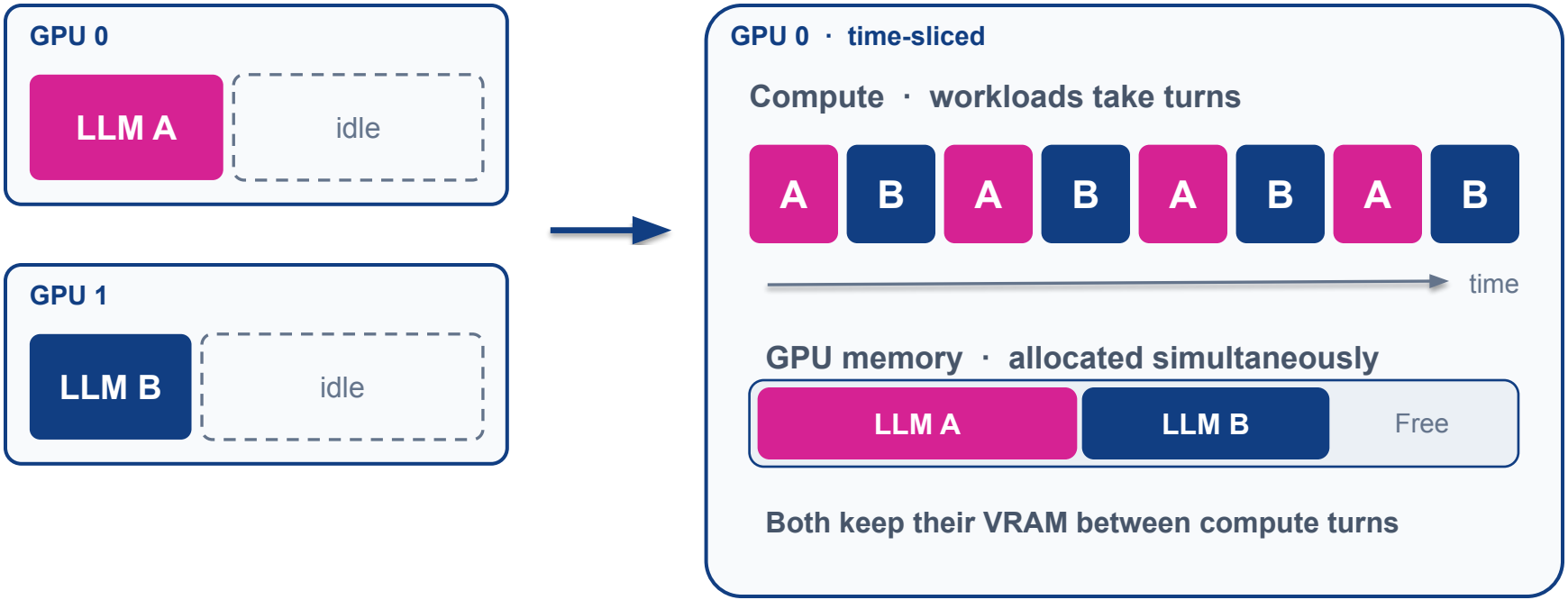
Each pod runs a single model and gets a whole GPU, and most of it sits idle



Put both models on the same GPU

KubeCon CloudNativeCon North America 2026

Each model still stays in one pod, but 2 pods can run on the same GPU



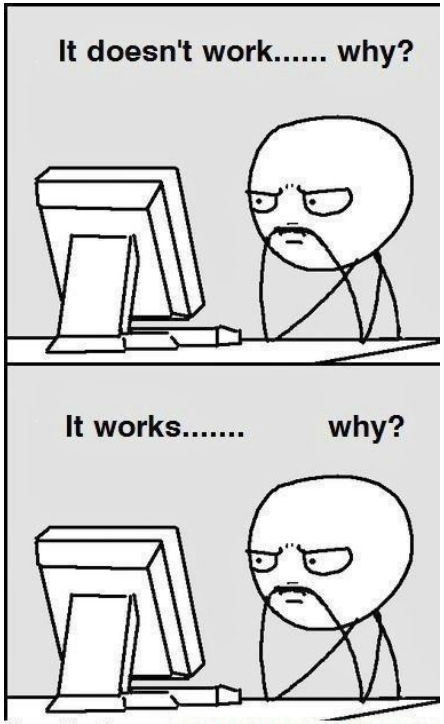
You set it up: not trivial, but it works

KubeCon CloudNativeCon North America 2026

NVIDIA device plugin · time-slicing config

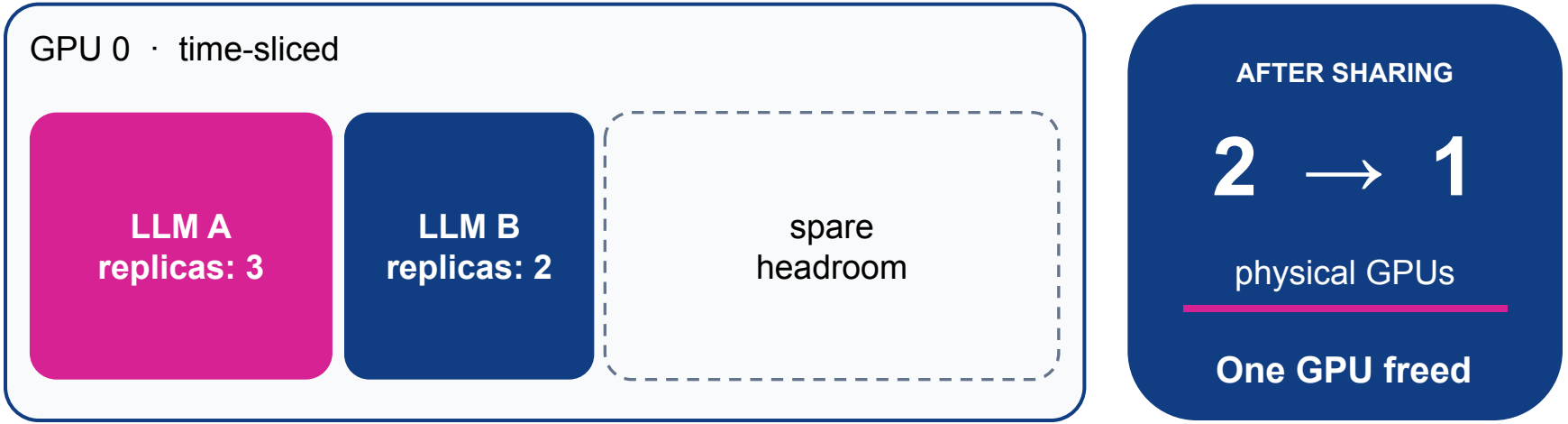
```
version: v1
sharing:
  timeSlicing:
    resources:
    - name: nvidia.com/gpu
      replicas: 5
```

One physical GPU is now seen by Kubernetes as 5 schedulable nvidia.com/gpu



Two models, one GPU, room to grow

Same two models, on a single GPU, with headroom left over



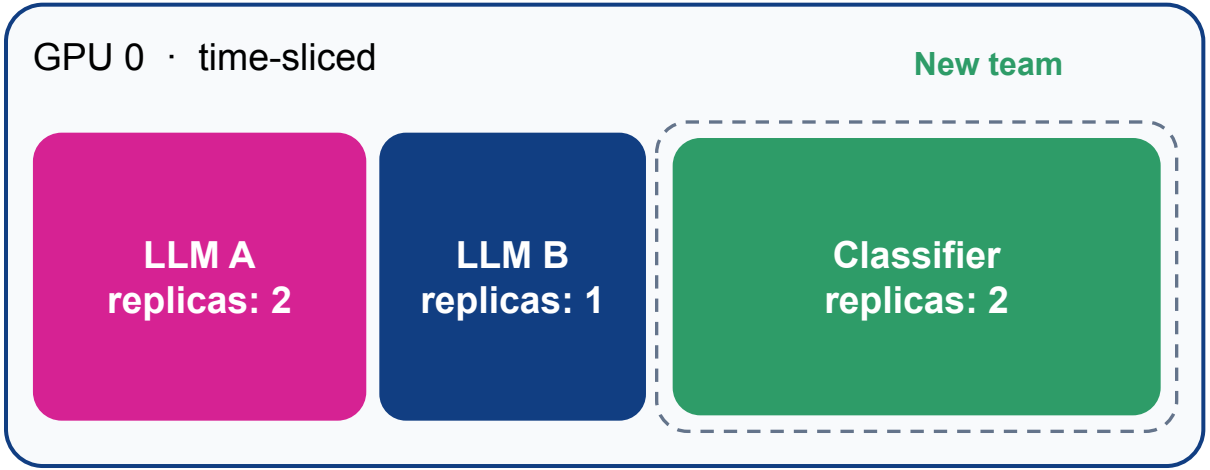
All 5 GPU accesses assigned. The physical GPU still has headroom.


GREAT JOB ON INCREASING
SHAREHOLDER VALUE
imgflip.com


YALL GOT ANY MORE OF THOSE
SPARE GPU CAPACITY
imgflip.com

In theory, it fits

You do your math, everything looks fine.



WHAT KUBERNETES SEES

5 / 5

GPU accesses

✓ **Schedules fine**

From the scheduler's perspective, everything fits.

KubeCon CloudNativeCon
North America 2026

Let's see how that turns out

LIVE DEMO (not)



Unfortunate outcome



A fifth of a GPU isn't a fifth of anything

KubeCon CloudNativeCon North America 2026

With replicas: 5, Kubernetes counts five accesses, not five protected 20% shares.

KUBERNETES VIEW

5 logical GPU accesses



PHYSICAL GPU MEMORY

One VRAM pool, no memory limits

Before traffic increases



Classifier KV cache grows



Shared VRAM exhausted → OOMs

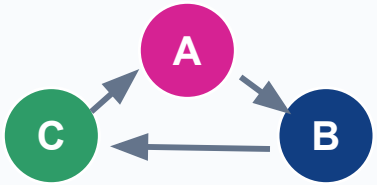
20% of a GPU isn't necessarily 20% of anything

Why not just use MIG or MPS?

KubeCon CloudNativeCon North America 2026

Time-slicing

DEVICE PLUGIN



No dependencies

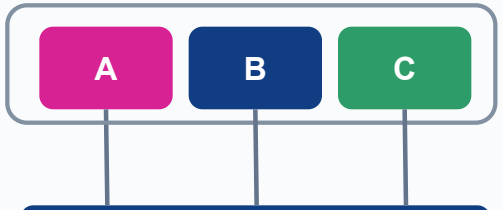
Compute: sequential turns

Memory: shared, no caps

Shared fault domain

MPS

CUDA RUNTIME



MPS control daemon (required)

Compute: concurrent kernels

Memory: per-client limits

Runtime isolation boundary

MIG

HARDWARE PARTITIONS



Pre-partitioning required
Not available on all GPUs

Compute: fixed instances

Memory: dedicated VRAM

Hardware isolation boundary

Meet KAI Scheduler

KubeCon CloudNativeCon North America 2026

A CNCF Sandbox project for GPU-aware scheduling on Kubernetes.

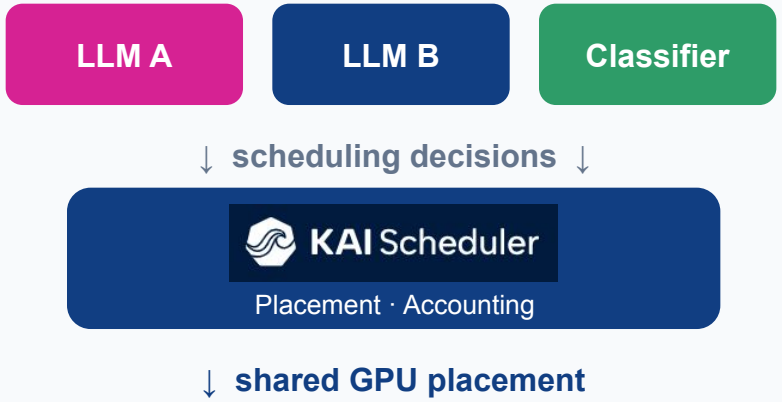
WHAT KAI HANDLES

Placement
Bin-packing and GPU topology

Scheduling Fairness
Queues, quotas, priorities

Fraction accounting
Workload-aware GPU memory requests

OUR THREE WORKLOADS

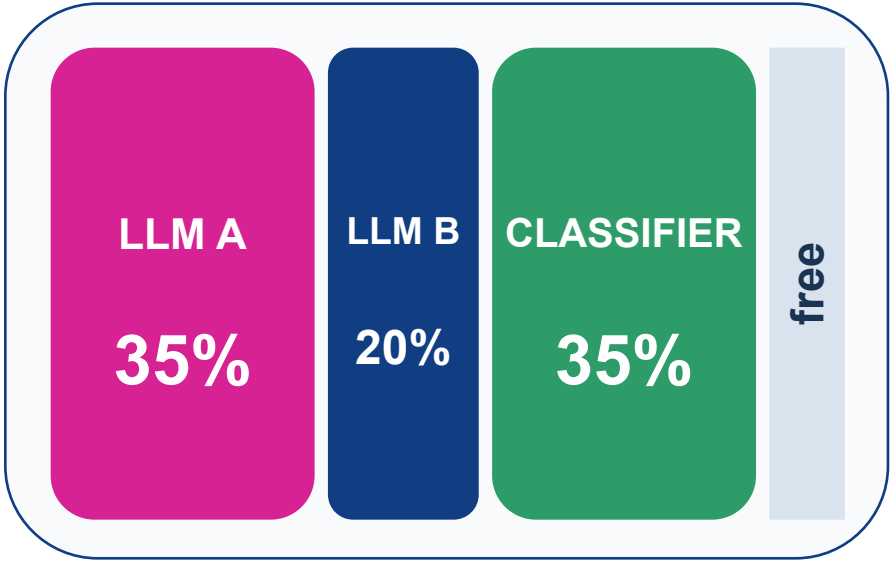


KAI decides where workloads run and accounts for what they request.

Flexible fractioning, no enforcement

KAI fractions act like Kubernetes resource requests.
They guide pod placement, not runtime enforcement.

```
apiVersion: v1
kind: Deployment
metadata:
  name: llm-a
  labels:
    kai.scheduler/queue: default-queue
  annotations:
    gpu-fraction: "0.35"
spec:
  schedulerName: kai-scheduler
  ….
```



What happens in the GPU stays in the GPU

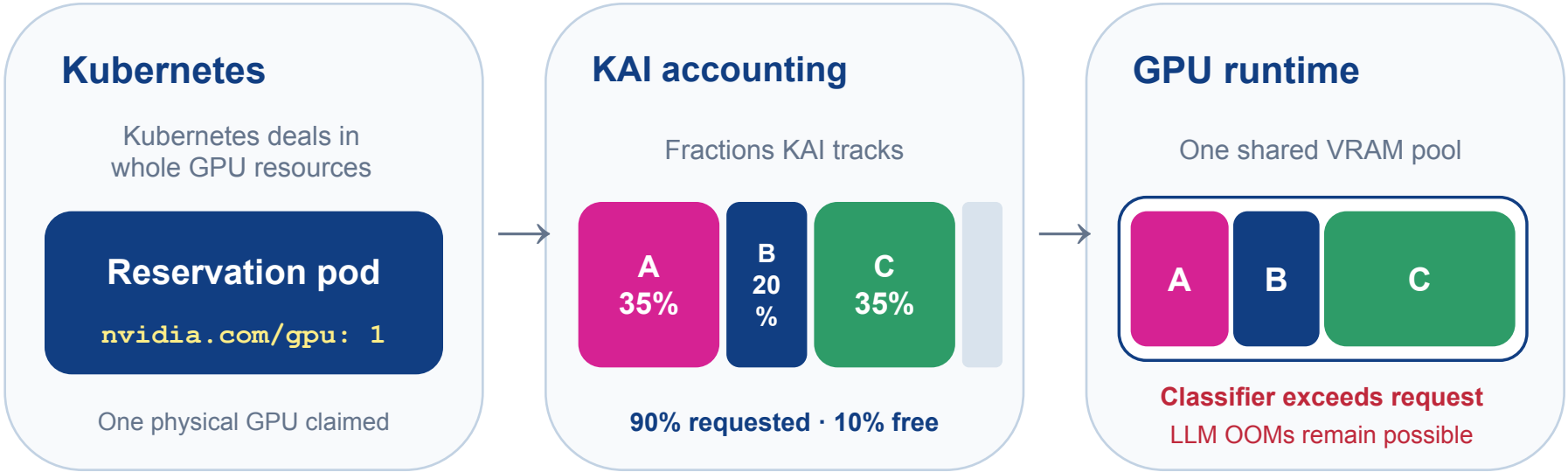
Everything should be fine, right?



Scheduling a fraction isn't enforcing a fraction

KubeCon CloudNativeCon North America 2026

KAI accounts for GPU memory requests. It doesn't enforce them at runtime.

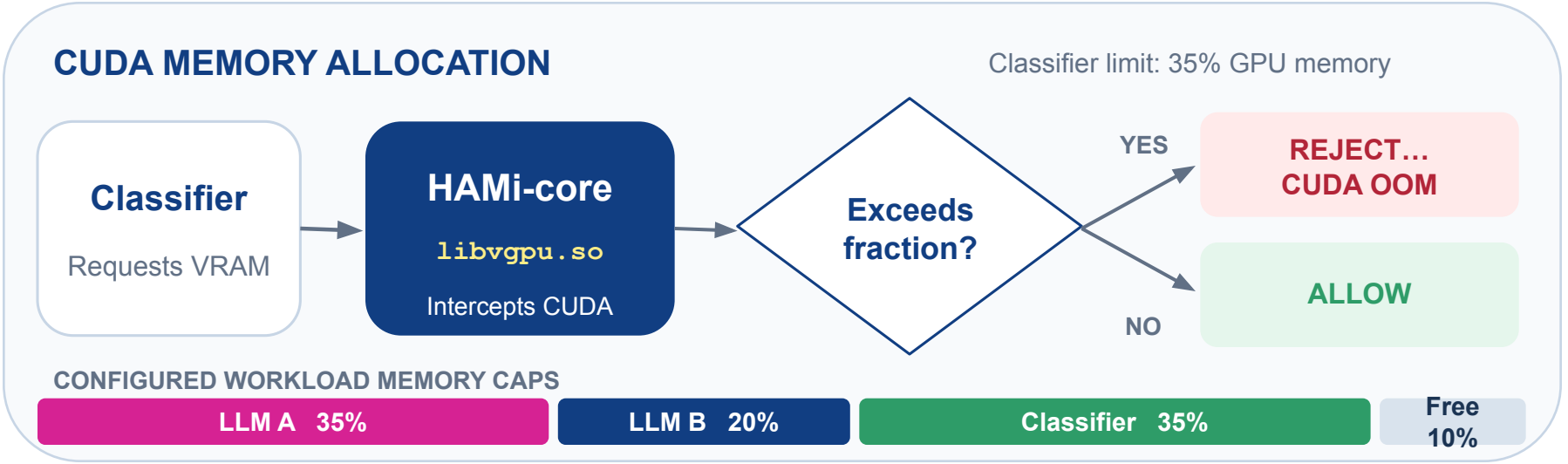


The only runtime boundary is all 3 pods are capped to 1 GPU

HAMi-core: Enforcing GPU memory limits

KubeCon CloudNativeCon North America 2026

An in-container CUDA interception library enforces per-workload GPU memory limits.



Pods can OOM at their limit without exhausting neighbors' VRAM.

HAMi-core supports compute throttling, no KAI fractions support yet.

KAI + HAMi-core: Putting the pieces together

KubeCon CloudNativeCon North America 2026

Example: LLM A requests 35% of an NVIDIA L4 (24 GB).

01 POD REQUEST

LLM A

```
gpu-fraction:
"0.35"
```

35% of GPU memory

→

02 KAI SCHEDULER

Scheduled onto

NVIDIA L4

`23,034 MiB × 0.35`

= 8,061 MiB

→

03 RUNNING CONTAINER

`CUDA_DEVICE_MEMORY_LIMIT`

8061m

≈ 7.87 GiB

HAMi-core enforces

libvgpu.so via kai-resource-isolator

KAI computes a memory cap. HAMi-core enforces it.

HAMi ships a DaemonSet (resource isolator) that injects the library into the pod at admission.

KubeCon CloudNativeCon
North America 2026

Did we fix it?

LIVE DEMO (not)




GREAT SUCCESS!

Same GPU… different blast radius.

The classifier can still fail but no longer takes its neighbors down.

BEFORE

Time-slicing

LLM A	× Error
LLM B	✓ Ready
Classifier	✓ Ready

No per-workload memory limit
Classifier can exhaust shared VRAM

AFTER

KAI + HAMi-core

LLM A	✓ Ready
LLM B	✓ Ready
Classifier	× Error

Per-workload limits enforced
Classifier capped at 35% VRAM

We didn't eliminate the OOM. We contained the blast radius.

GPU sharing is moving fast

KubeCon CloudNativeCon North America 2026

IN OUR DEMO

KAI + HAMi-core

CUDA-level memory limits

JUST GOT HERE

NvFractions CUDA 13.4

NVIDIA driver-enforced GPU memory limits

LOOKING AHEAD

DRA Dynamic MIG

More flexible fractional GPU sharing

References and documentation

- KAI GPU sharing docs
  - https://github.com/kai-scheduler/KAI-Scheduler/tree/main/docs/gpu-sharing
- HAMi-core + KAI initial announcement and getting started
  - https://project-hami.io/blog/hami-core-adopted-by-nvidia-kai-scheduler
- KAI + nvfractions
  - https://developer.nvidia.com/blog/memory-enforced-gpu-sharing-in-kubernetes-with-nvidia-cuda-13-4/
- GPU process exporter (per container GPU metrics)
  - https://github.com/densify-dev/gpu-process-exporter

KubeCon CloudNativeCon
North America 2026

Thank you!

Questions?

SESSION FEEDBACK

Scan to leave feedback

