

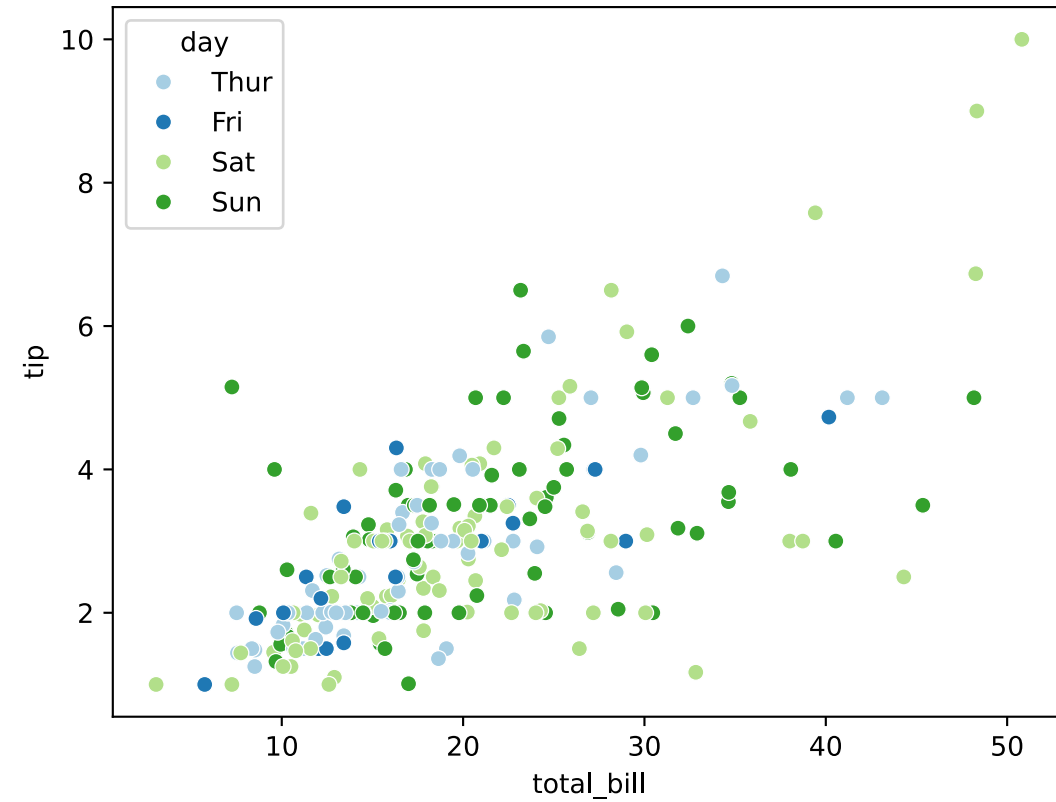
Seaborn

DATA VISUALIZATION WITH PYTHON

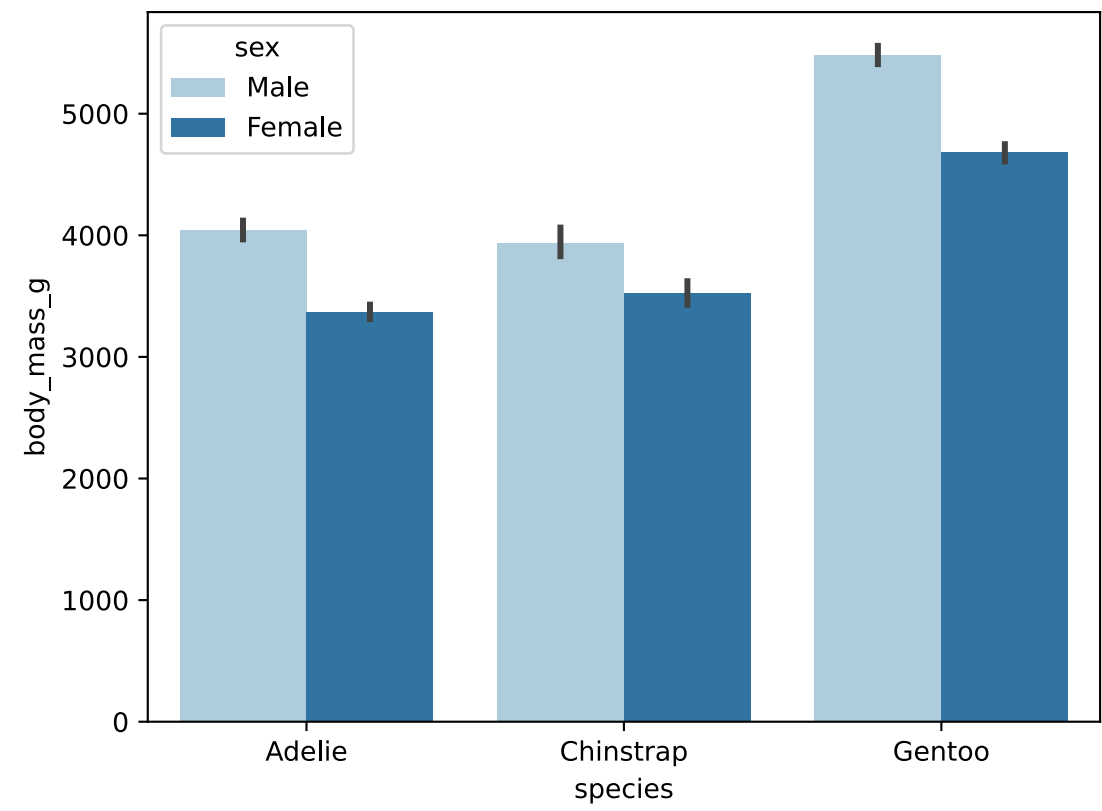
Decktile

Code-to-image rendering

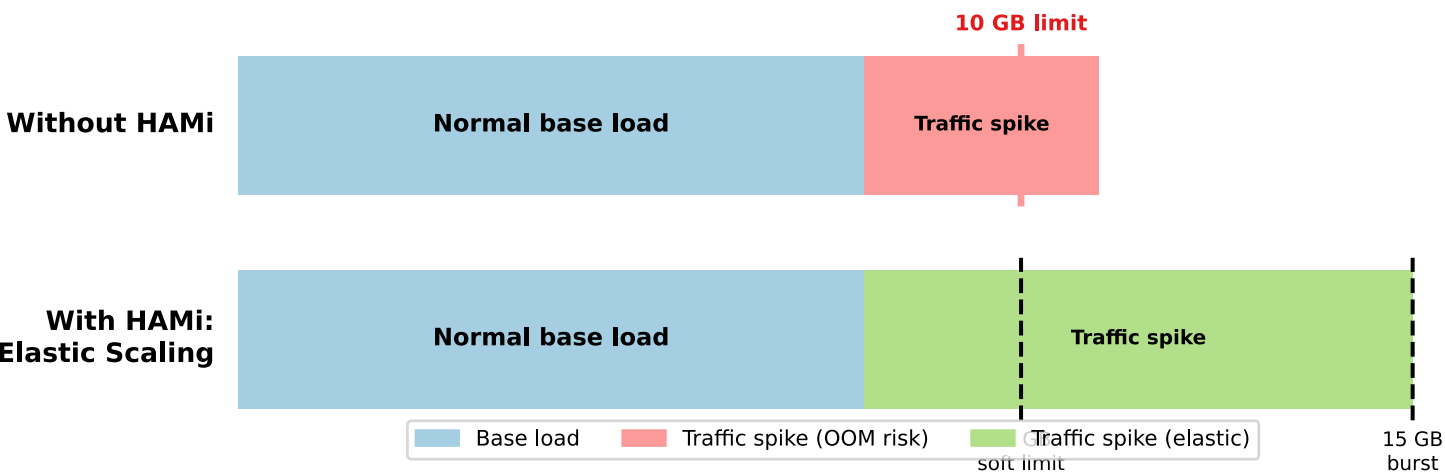
Scatter Plot



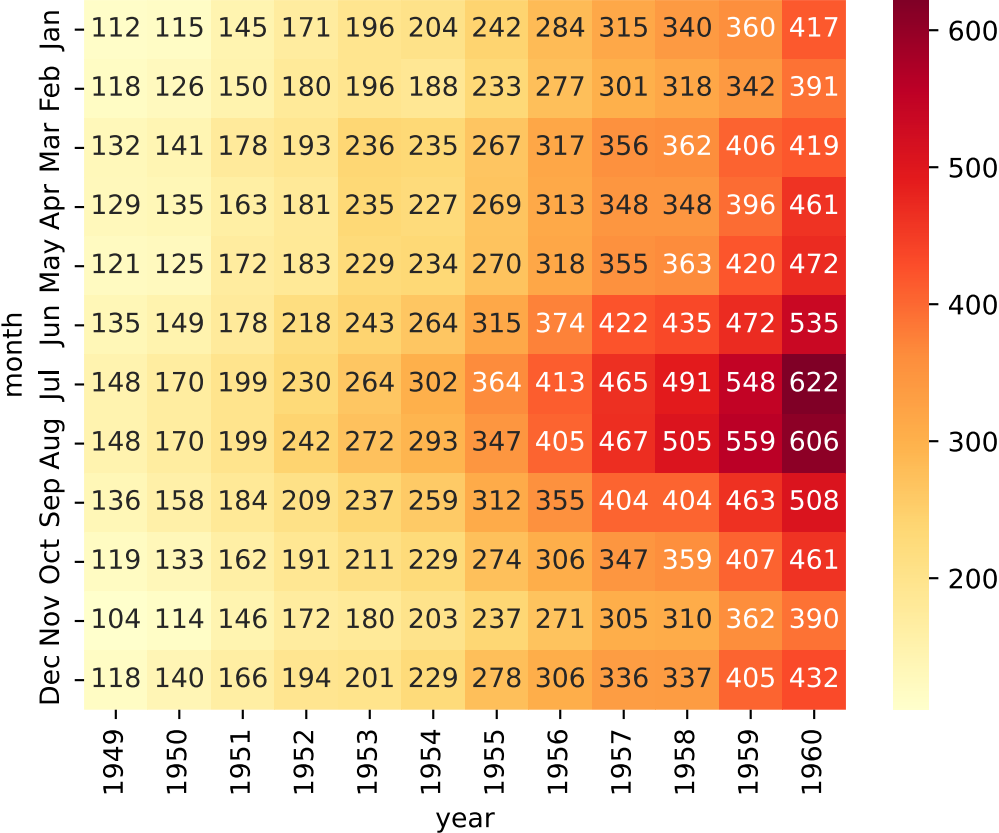
Bar Plot



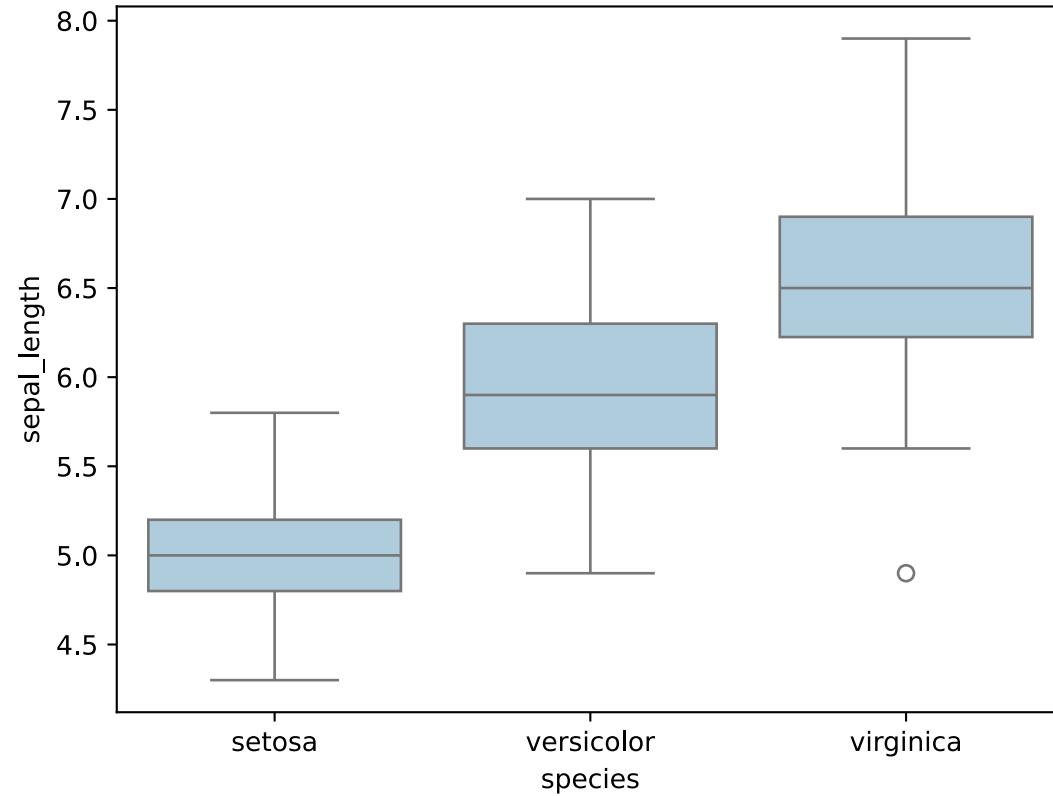
Memory Oversubscription



Heatmap



Distribution



- ▶ Seaborn builds on matplotlib
- ▶ Tight integration with pandas DataFrames
- ▶ Statistical visualizations out of the box
- ▶ Auto-generated from fenced code blocks

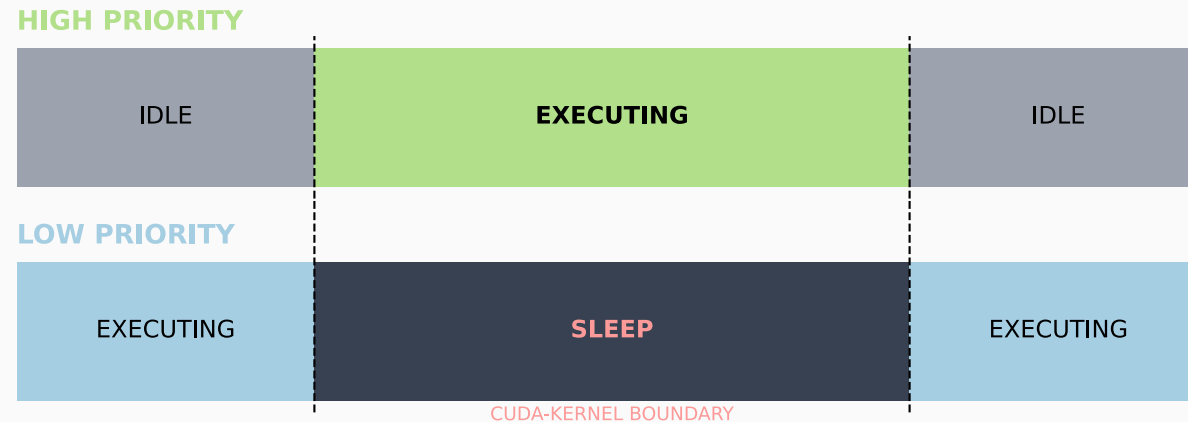
Explainer charts

HAND-DRAWN MATPLOTLIB, FROM A REAL TALK

Priority Preemption

High priority pauses low priority at kernel boundaries

High-priority tasks preempt low-priority ones at CUDA kernel boundaries: no wasted compute, clean context switch:



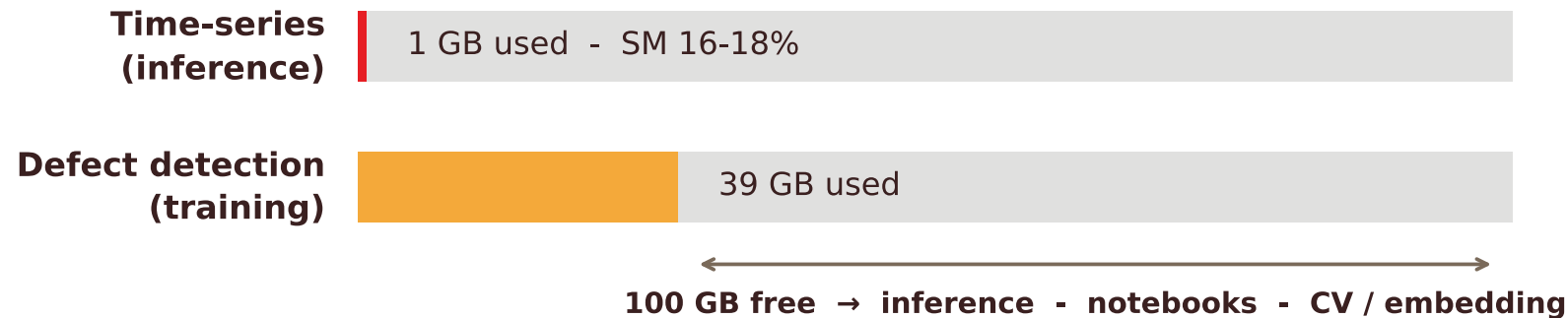
The Problem We Measured

Asking for a GPU is not the same as using it

Two real jobs. Each one held **a whole H200, 141 GB**:

GPU MEMORY - ALLOCATED VS USED

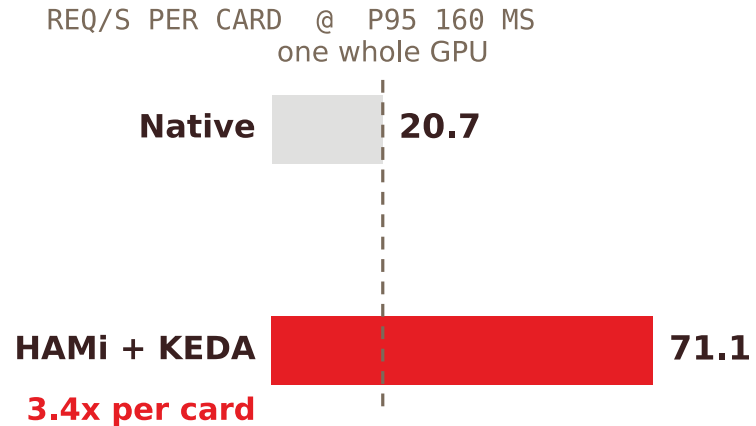
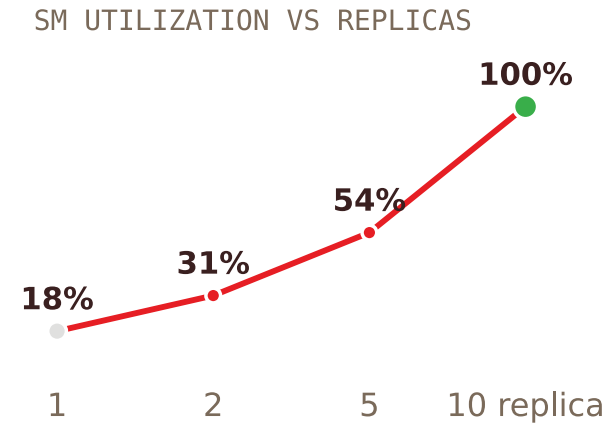
one whole H200 = 141 GB



- ▶ The time-series job leaves **140 GB and 80% of the GPU** doing nothing
 - ▶ Detection training took **39 GB** - it still left most of the card unused
 - ▶ **Many of our GPUs have no MIG**: L40, L40S, A6000, ...
- | [Kubernetes gives you one choice: nvidia.com/gpu](https://nvidia.com/gpu): 1. The whole card, or nothing.

The Result: 3.4x More Work Per GPU

Fix the SLA, scale replicas, fill the empty GPU



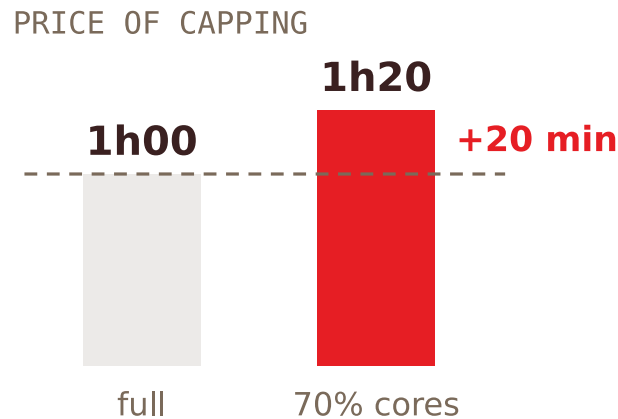
One pod uses only **18% of the card**. Pack **10 replicas** on it and the GPU hits **100%** - same latency.

- ▶ **HAMi does not make one pod faster** - it fills the GPU that was sitting empty
- ▶ Bigger batches did not help either: **32 to 512**, GPU still 7-10%
- ▶ Same 71 req/s (H200): **1 GPU instead of 3.4** → **~71% fewer cards**, **~51% less power**, **~\$7k/month** at an example \$4/GPU-h

One Card, One Day

One slice guaranteed, on purpose - and what it costs

Workload: an AI Notebook training YOLO11 - ~17k images, batch 64 → **39 GB** used. Batch is sized for **accuracy**, not to fill the card - so the spare **~100 GB is genuinely free**, not just idle. We fence **~30%** and cap the notebook to **70% of the cores**.

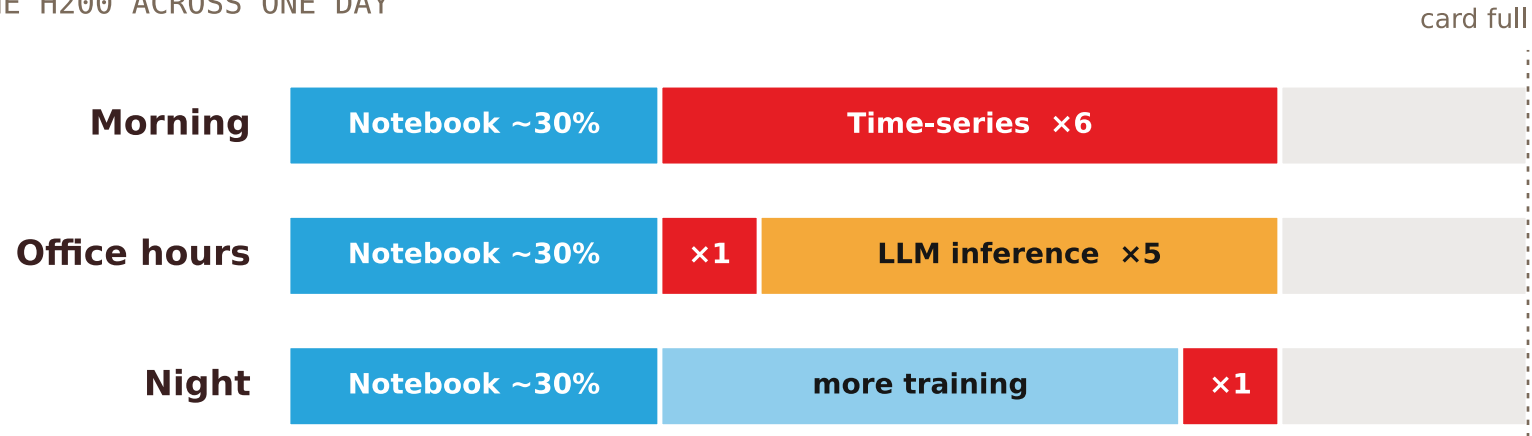


- ▶ 🗄️ **Memory is fenced.** The notebook sees only its own slice - it cannot OOM the neighbours
- ▶ ⏸️ **Cores are capped** with force - the freed share is **reserved**, not "maybe"
- ▶ ⌚ **The price:** ~20 minutes, on purpose - *a small model; a big one costs much more*

The Rest Follows The Load

One notebook fixed all day - inference flexes by the hour

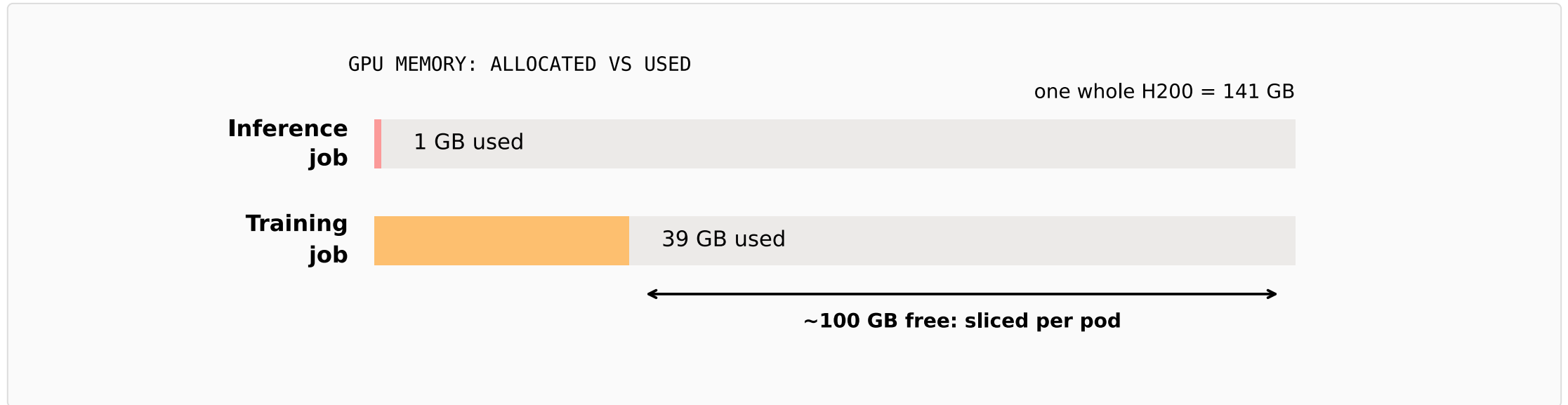
ONE H200 ACROSS ONE DAY



- ▶ **🔒 The notebook never moves.** Its slice is fixed all day → training stays stable and guaranteed
- ▶ **🔄 KEDA hands the freed space off through the day.** The morning **burst fades** → cores go to **LLM <20B** → one time-series instance stays for scattered load → at night, spare **auto-scales more notebooks** for extra overnight training. *Never idle.*

Memory Slicing

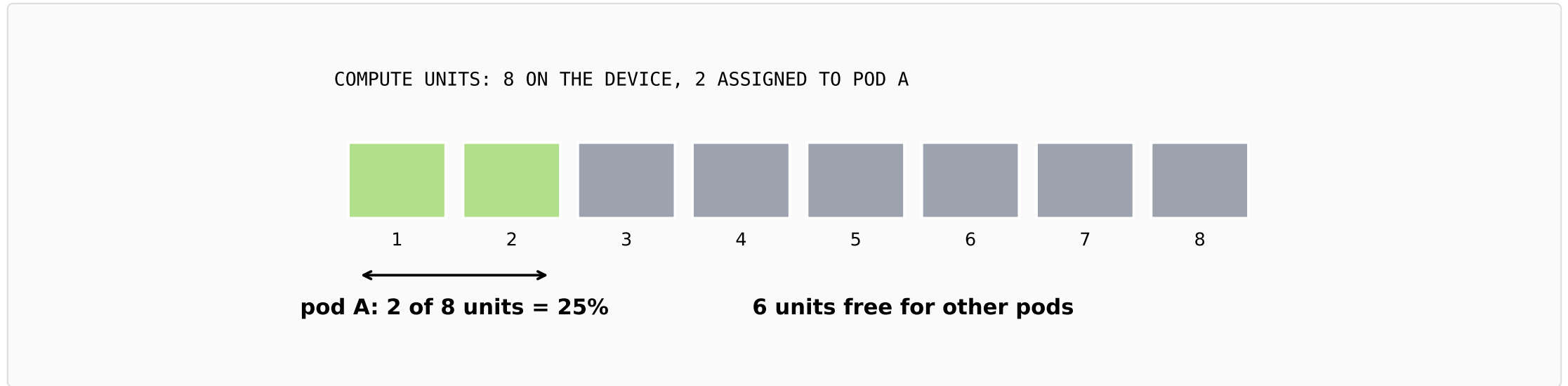
Carve unused memory into per-pod slices



- ▶ One job holds a whole card: 1 GB used of 141 GB
- ▶ Slicing frees the rest: MiB-level partitions per pod

Compute Slicing

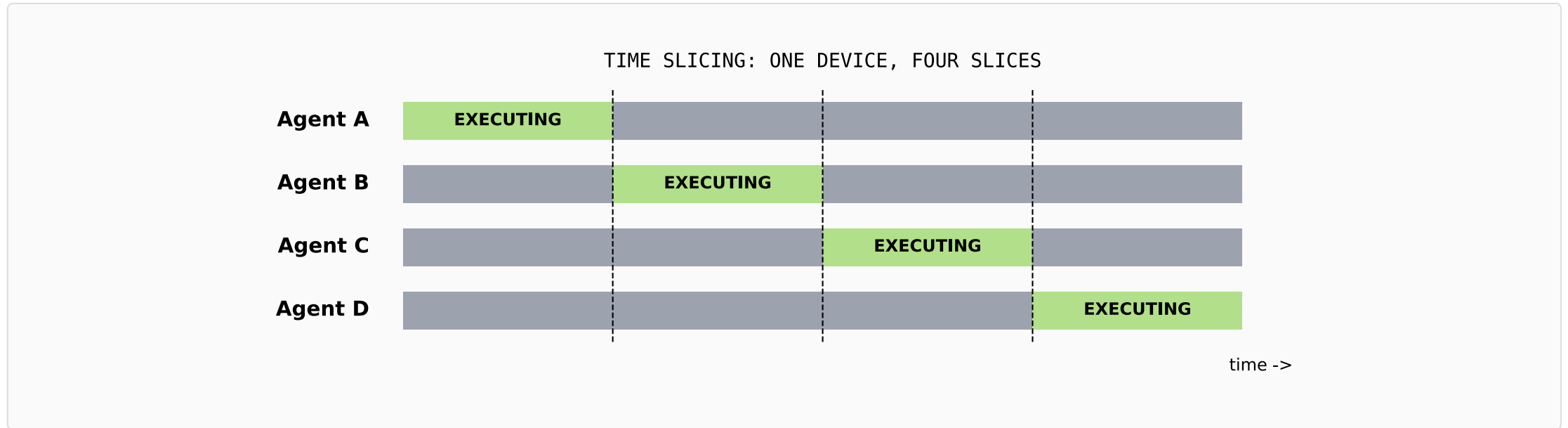
Percent of compute per pod, enforced on every call



- ▶ Each pod gets a fixed % of the device's compute units
- ▶ Hard ceiling on every kernel launch: no bursting above the slice
- ▶ Unused units stay free for other pods

Time Slicing

Workloads take turns at kernel boundaries



- ▶ Rotate compute access over time
- ▶ Kernel boundary is the switch point: no mid-kernel preemption
- ▶ Preemption optional: without it, long kernels run to completion

Hardware Partitioning

Fixed slots, strongest isolation, least flexible

HARDWARE PARTITIONS: MIG, SR-IOV, vXPU, vDCU



SOFTWARE PARTITIONS: HAMI slices



- ▶ MIG, SR-IOV, vXPU, vDCU: hardware partitions, silicon-enforced
- ▶ Memory and compute fenced per slot: strongest isolation