

A possible future: DRA and KAI

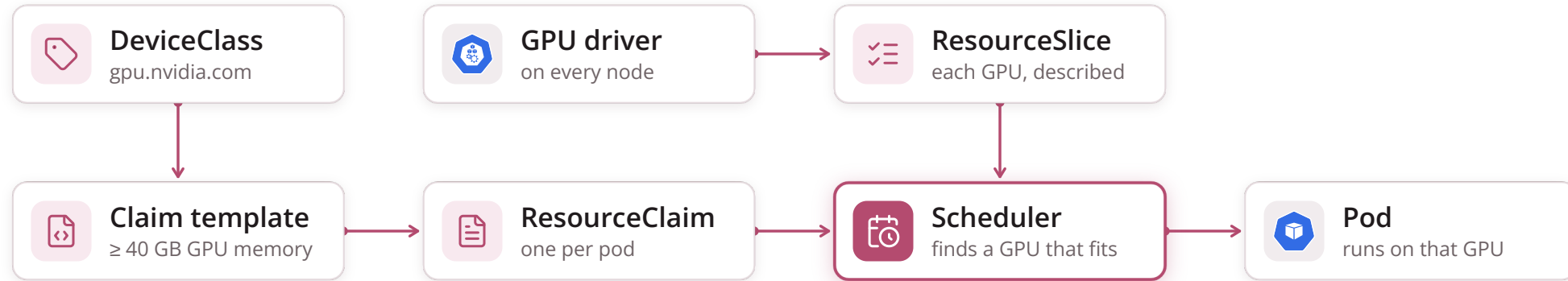
What the platform could do if GPUs were described, not counted

What we cannot say today

	today	could help
 "a card with enough memory"	GPUs are counted; a kind of card is only a node label	DRA
 "this recipe needs more"	every pod asks for the same	DRA claims
 "use what is free"	the window is fixed and admitted whole	KAI elastic jobs, Kueue elastic workloads
 "each team its share"	one pool, first come first served	KAI queues, Kueue cohorts
 "part of a card"	whole cards only	MIG through DRA, KAI fractions



DRA — claim a GPU by what it is



The driver describes every GPU. A ResourceSlice lists each device with its product name and memory, so the cluster knows what a card is, not only how many.

A pod asks for what it needs. A claim template says "a GPU with more than 40 GB"; each pod gets its own claim, and the scheduler finds a card that fits.



A pipeline asks for what its model needs

a claim template – the need, not the machine

```
kind: ResourceClaimTemplate
metadata:
  name: large-model
spec:
  spec:
    devices:
      requests:
      - name: gpu
        exactly:
          deviceClassName: gpu.nvidia.com
          selectors:
          - cel:
              expression: >-
                device.capacity['gpu.nvidia.com'].memory
                  .isGreaterThan(quantity("40Gi"))
```

Any card that has it will do. The recipe says how much GPU memory its models want; named sizes become a claim.

Kueue still counts. It charges a claim to quota by its device class, not by the expression — one quota per kind of card needs one class per kind.

Not yet in Kueue: a claim with a fallback list.



Split a card, share a card

	what it gives	memory isolation	status
MIG, set up in advance	fixed slices of one card, each its own device	yes, in hardware	on by default in the NVIDIA DRA driver
MIG, on demand	slices cut when a claim asks for one	yes, in hardware	alpha
Time-slicing	pods take turns on one card	none	alpha
MPS	processes run side by side	thread share and pinned-memory limits only	alpha
KAI GPU fractions	a pod asks for half a card, or a number of MiB	none	off by default

Only MIG keeps one model out of another's memory. Everything else trusts each pod to stay within what it asked for.



KAI Scheduler — a scheduler built for GPUs

A second scheduler, beside the default one. It places the pods that ask for it, with its own queues and pod groups. It began inside Run:ai and is now a CNCF sandbox project.

A Job opts in from its pod template:

```
template:
  metadata:
    labels:
      kai.scheduler/queue: transcription
  spec:
    schedulerName: kai-scheduler
```

What it adds:

- queues per team — quota, over-quota weight, a limit
- gang scheduling — a group starts together, or at least a minimum of it
- elastic jobs — a minimum that must run, the rest reclaimable
- GPU fractions, bin-packing and consolidation
- topology-aware placement



What KAI would give us



More volumes per card. Small models share one GPU: four or five volumes where today there is one.

Better, not new: team queues with a guaranteed quota, a weight for idle GPUs and a hard limit — Kueue's cohorts cover most of it.



Start with what is free. A campaign starts with one pod and grows; what it borrowed goes back when a team needs it.

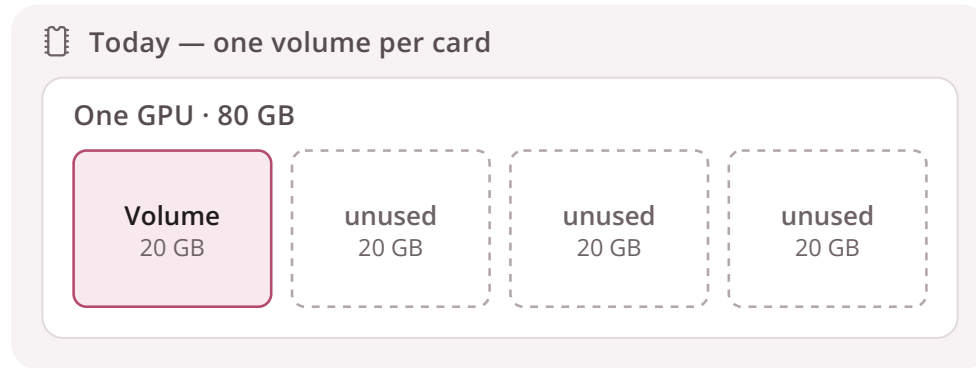


Whole nodes kept free. Pods are packed onto few nodes, so work that needs a full node still fits.

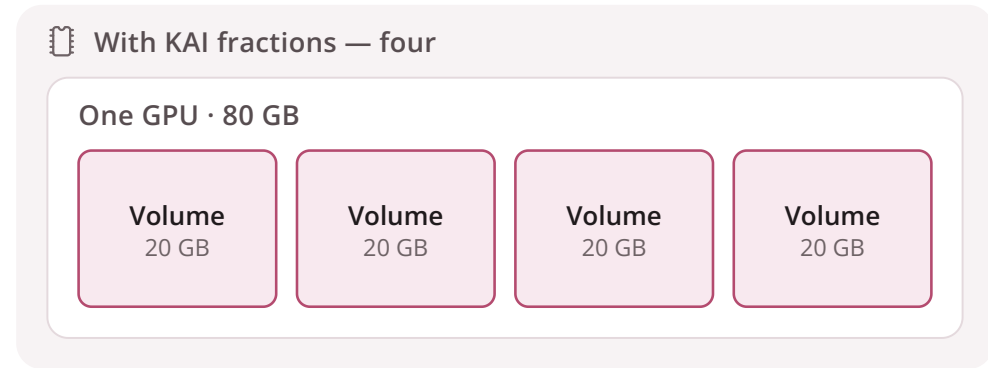
Not from KAI: GPUs described by what they are — its DRA support is still on its roadmap.



More volumes per card



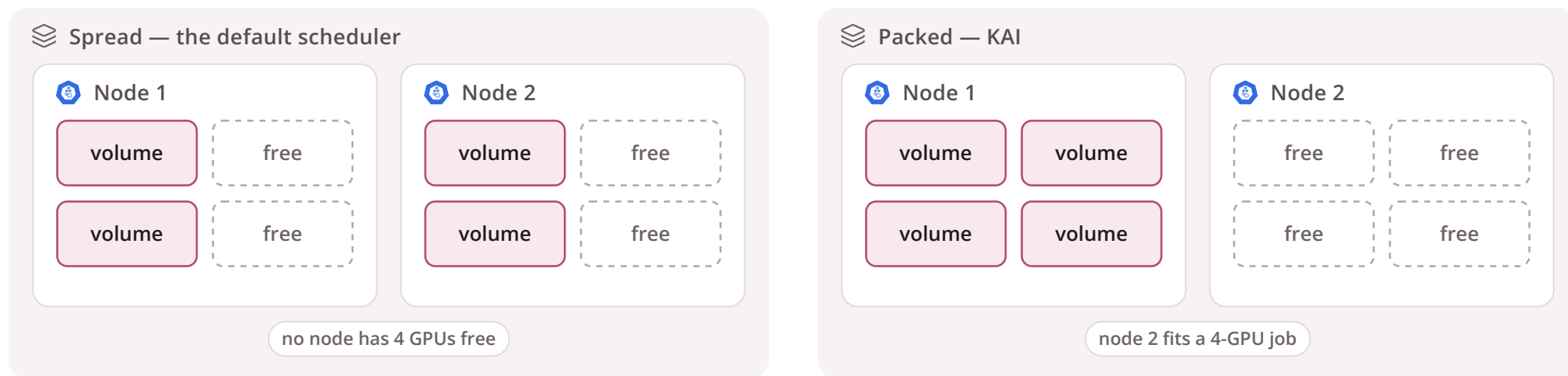
A pod asks for part of a card — a share, or a number of MiB — and KAI places several on one GPU.



The catch: nothing stops a pod from using more than its share, and its neighbour runs out of memory. A pipeline needs a known, steady memory use first.



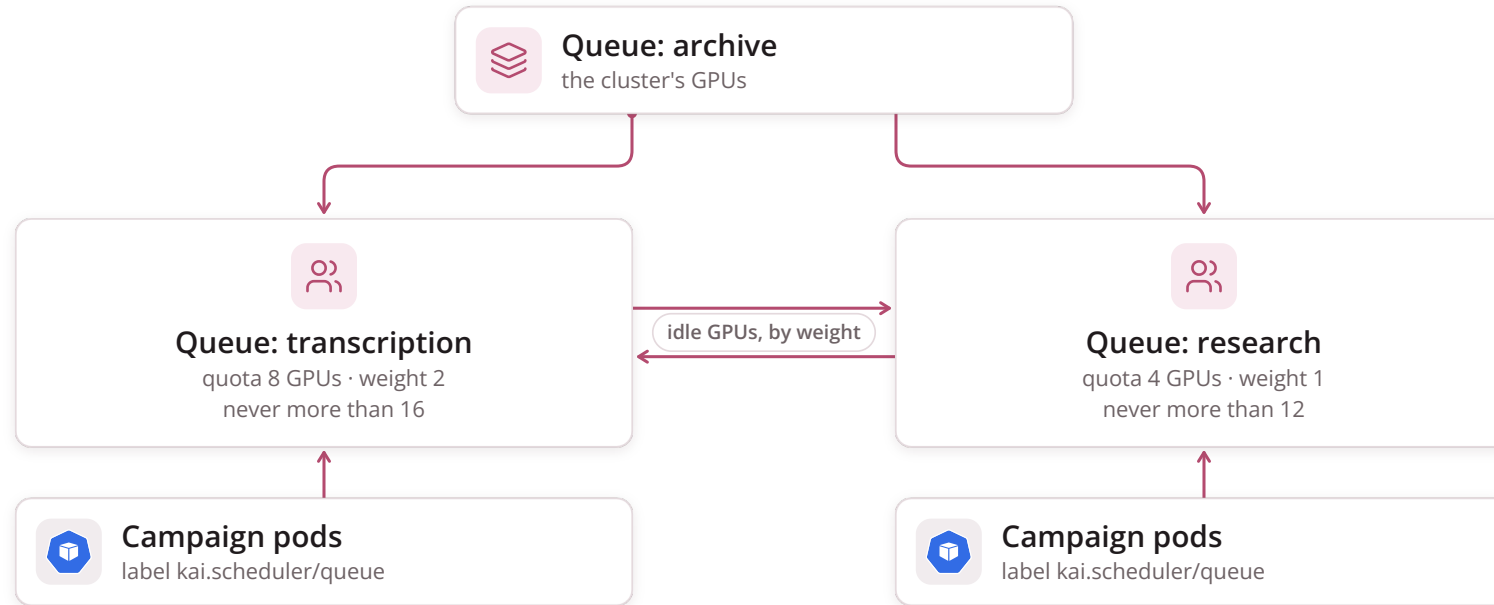
Whole nodes kept free



The default scheduler spreads pods; KAI packs them and can move them to close gaps. It matters when the cluster also runs work that needs several GPUs on one node — training, or a large model served.



Team queues that lend and take back

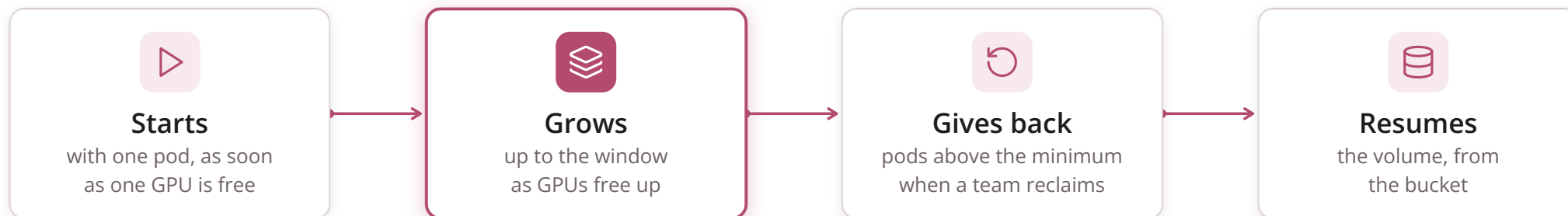


Quota is guaranteed; weight shares the rest. A team always gets its quota; idle GPUs go out by over-quota weight, up to each queue's limit.

Reclaim takes back only what was lent. When a team needs its quota, pods running over someone else's are evicted — never a team's own share. The volume resumes from the bucket.



A window that is a range



The window becomes the most, not the must. A KAI Job starts with a minimum of one pod and grows as GPUs free up; the pods above the minimum are what a reclaim takes.

Kueue has a version too. Elastic workloads change an admitted Job's parallelism without suspending it — beta, and not together with partial admission.



When KAI is worth it

	worth adding KAI?
 one team, a cluster of its own, models that need a whole card	no — Kueue covers it
 small models that fit several to a card	yes — fractions, if memory use is steady
 several teams, bursty demand, a cluster kept full	maybe — start with what is free, and reclaim; Kueue comes close
 a cluster shared with training or served models	yes — packing, and which work may be stopped

Our volumes are a good fit for reclaim: a volume resumes from the bucket, so taking a pod back costs only the page in flight.



Kueue, KAI, or both

	 Kueue	KAI Scheduler
when work may start	admits a Workload against quota	queues gate pods as they are scheduled
where pods run	left to the default scheduler	its own scheduler
teams	cohorts, borrowing limits, fair sharing	a queue tree: quota, over-quota weight, reclaim
kinds of GPU	flavors; DRA device classes in quota	on its roadmap, with DRA for whole GPUs
part of a card	MIG counted through DRA	fractions, by annotation

Both at once means two quota layers. Neither documents an integration; one has to step back — for instance KAI with unlimited queues, only placing what Kueue admitted.



What would change for a team

```
# pipelines/demo-v2.yaml
image: ...@sha256:...
gpu:
  memory: 40Gi      # a claim, not a count
steps: ...
```

```
# campaigns/kyrkobocker-2.yaml
pipeline: demo-v2
window: 8           # at most; starts with what is free
volumes: ...
```

The repository stays. Campaign files, pipelines, pull requests and apply work as they do.

The recipe states its needs; the platform turns them into a claim.

The queue is the team's, set by the platform, not by the repository.



What to watch

- **Fractions share memory.** KAI fractions and time-slicing do not isolate it; a large model can push its neighbour out.
- **GPU allocation in NVIDIA's DRA driver is not yet officially supported**, and its sharing modes are alpha.
- **Kueue with DRA:** no fallback lists, no topology-aware placement, and Kueue cannot see which card a pod will get.
- **The default scheduler does not preempt for DRA devices.**
- **Two quota layers** if Kueue and KAI run together.
- **An elastic window on an Indexed Job** has not been tried.



Any questions?

