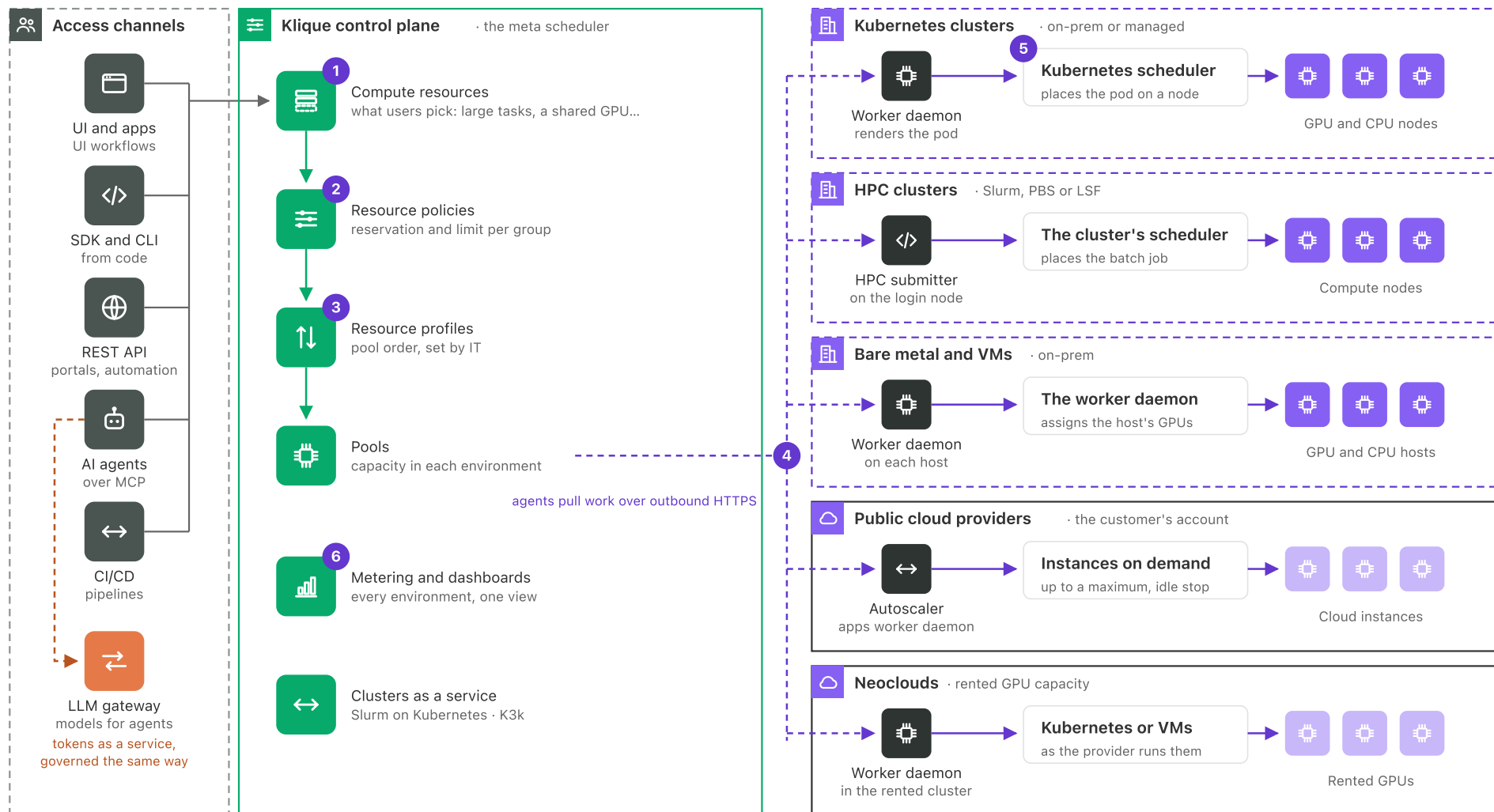


Meta-Scheduler Reference Architecture

This reference architecture covers the following scenario: an enterprise has GPUs in several places, including Kubernetes clusters, HPC clusters, bare-metal servers, public cloud providers and neoclouds. Users and AI agents submit work through common access channels to compute resources named for what they give, such as multi-node training, large tasks or a shared GPU. IT decides through policy where each compute resource runs: one Klique control plane places each task in an environment, where a Klique agent turns it into that environment's own unit of work. Klique also offers clustering services: Slurm on Kubernetes, and K3k, Kubernetes in Kubernetes. The document covers the components, how capacity is shared between groups, and the governance layers Klique adds above the scheduler.

CONTROL PLANE One, on-prem, scheduling across every environment	ENVIRONMENTS Kubernetes, Slurm, PBS and LSF, bare metal and VMs, public cloud providers, neoclouds
HARDWARE CPUs, and NVIDIA and AMD GPUs, whole or shared	AUDIENCE IT architects and sales engineers

How to read the diagram: numbered badges follow one task from submission to the node that runs it.



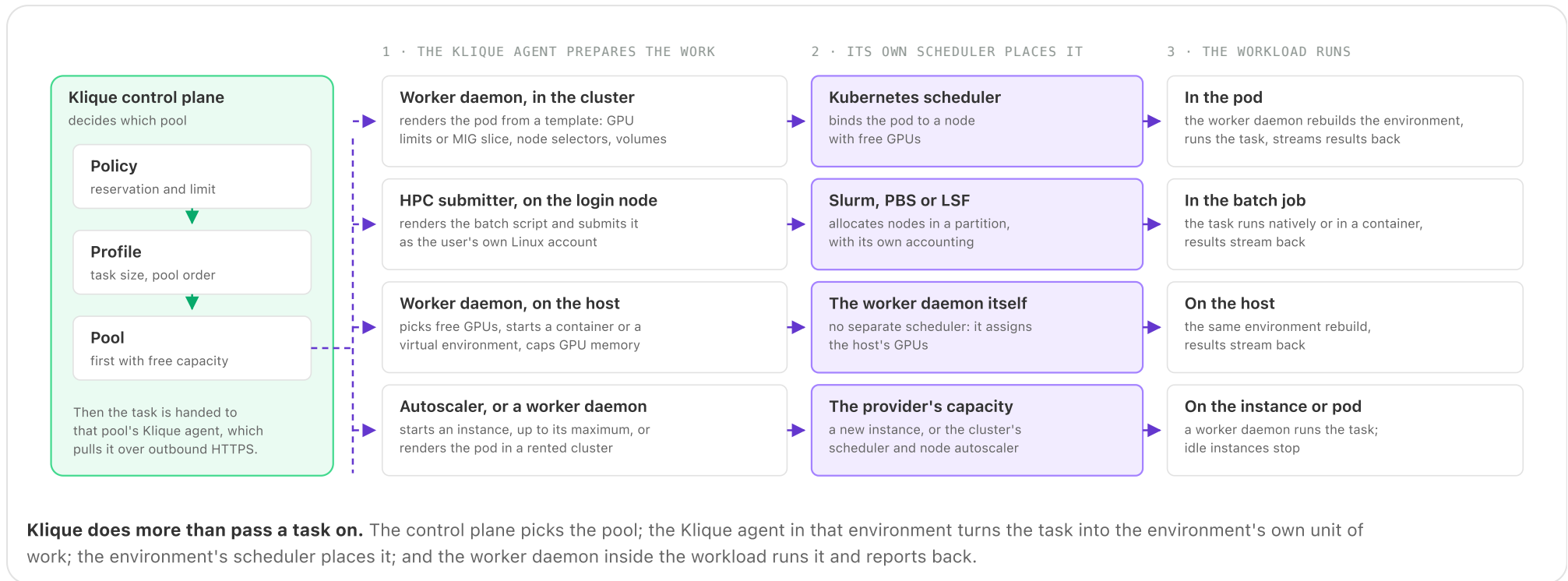
STEP	DETAIL	LOCATION	CONNECTION
1 Submit	Work arrives through an access channel: the UI and its apps and workflows, the SDK and CLI, the REST API, AI agents over MCP, or a CI/CD pipeline. It goes to a compute resource the user may use, such as one for multi-node training or a shared GPU. The GPU as a Service reference architecture covers each access channel. AI agents get their models from the Klique LLM gateway, covered in the LLM Gateway and Tokenomics reference architecture .	Control plane	Caller → API, HTTPS
2 Policy	The group's resource policy is checked. Work within the group's reservation goes first, and the group can use idle capacity up to its limit.	Control plane	None: a scheduling decision
3 Profile	The compute resource's profile sets the task size and lists pools in order. IT owns that order, so IT decides where the work runs; the task goes to the first pool with free capacity.	Control plane	None: a scheduling decision
4 Pull	The agent in that pool pulls the task over outbound HTTPS: a Klique worker daemon on Kubernetes, bare metal, VMs or a neocloud, the HPC submitter on an HPC login node, or an autoscaler, which first starts an instance in a public cloud account.	The pool's environment	Agent → control plane, outbound
5 Place	On Kubernetes the worker daemon renders a pod and the Kubernetes scheduler places it. On a bare-metal host the worker daemon starts a container on free GPUs. On Slurm, PBS or LSF the HPC submitter on the login node turns the task into a batch job from the batch template, and the cluster's own scheduler places it.	The pool's environment	Inside the environment
6 Report	Logs, metrics and GPU-hours flow back to the control plane and are recorded per user, project and pool.	Control plane	Agent → control plane, outbound

Business value

- **For the platform team: one way to use every GPU and CPU.** Kubernetes clusters, HPC clusters, bare-metal servers, public cloud providers and neoclouds are pools behind the same compute resources, reached through the same access channels. Adding an environment means installing an agent and defining a pool. Users and their code do not change.
- **For the business: work runs where capacity is free.** Ordered pools send work to owned capacity first and to rented capacity when it is full. A group's reservation holds in every environment.
- **For governance and finance: one set of controls.** Identity, policies, metering and billing apply to every task in every environment. Token budgets, guardrails and self-serve services sit above the scheduler on the same platform.

Two levels of scheduling

Klique is a meta scheduler. Users choose a compute resource by what it gives, for example multi-node training, large tasks or a shared GPU for a session, through whichever access channel suits them. IT decides, through policies and pool order, which pools serve each compute resource. Klique then picks the pool for each task, from the group's policy, the task size and the order of the profile's pools. A Klique agent then hands the work to that environment, where its own scheduler places it. Klique works alongside these schedulers rather than replacing them: a Slurm cluster keeps its partitions and accounting, and a Kubernetes cluster keeps its scheduler.

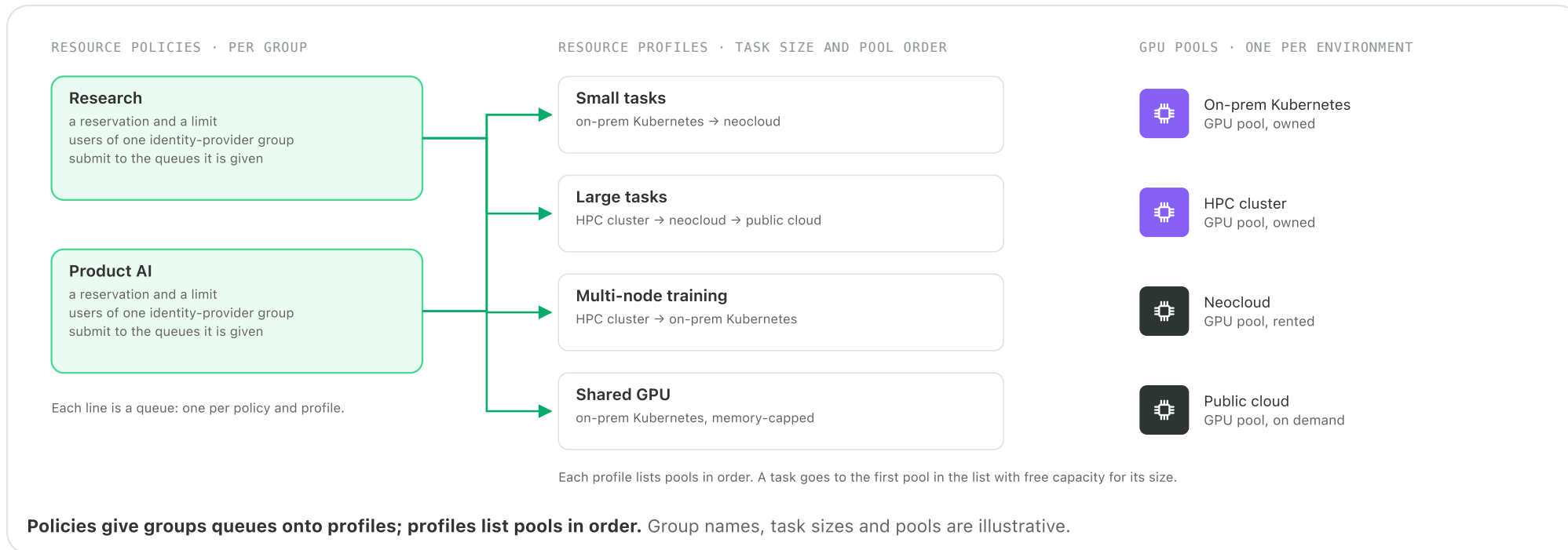


The resource manager in the control plane does logical scheduling and accounting: it counts GPUs against pools and policies. The Klique agent in each pool then does the environment-specific work. On Kubernetes the worker daemon renders the pod from a template, with GPU limits or a MIG slice, node selectors and volumes. On an HPC cluster the HPC submitter renders the batch script and submits it as the user's own Linux account. On bare metal the worker daemon picks free GPUs and starts a container or virtual

environment. Inside the running workload, the worker daemon rebuilds the task's environment, runs it and streams logs, metrics and results back.

Pools, profiles and policies

Administrators describe capacity once, in the Resource Configuration and Resource Policies settings. Users then submit to compute resources and receive what their group's policy allows.



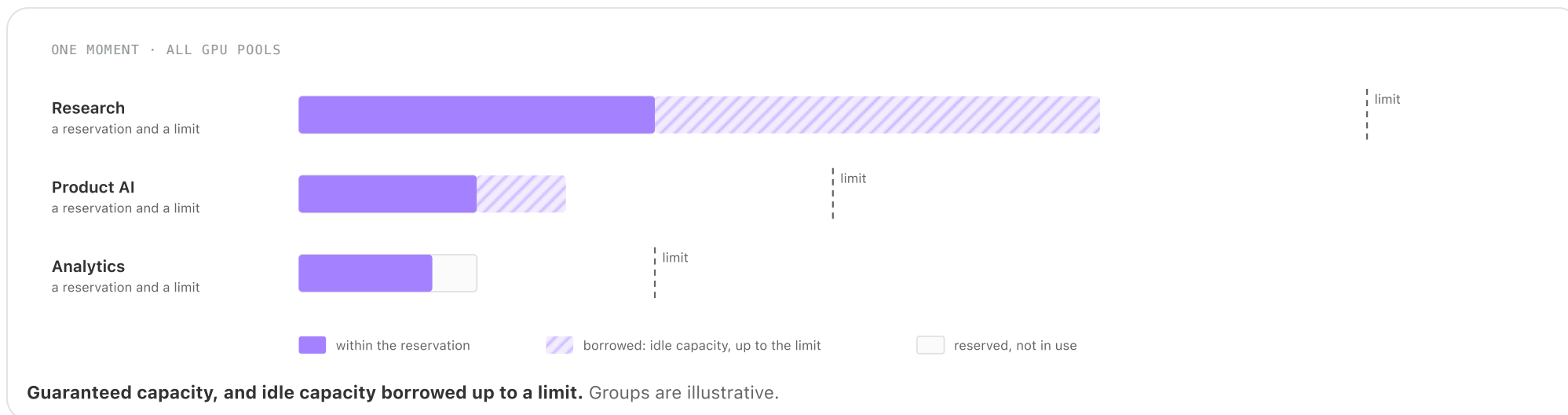
OBJECT	WHAT IT IS	SET BY
Pool	Capacity in one environment, counted in whole GPUs: a Kubernetes cluster, an HPC cluster or partition, a group of bare-metal hosts, a cloud autoscaler or a neocloud.	Administrator
Resource profile	A task size, fixed or read from the task, and an ordered list of pools. Within a pool, profiles have an execution priority.	Administrator
Resource policy	A user group's reservation and limit, and the profiles it may use. Within a profile, policies have a queuing priority.	Administrator
Compute resource	The queue a group submits to, one per policy and profile, named for what it gives, such as multi-node training or a shared GPU. Users pick it; IT decides where it runs.	Created when a profile is connected to a policy
Worker daemon or HPC submitter	The agent that pulls tasks for a pool and runs them in its environment.	Installed in each environment

Pools are sized in whole GPUs whatever the work. A task on a share of a GPU, or on a MIG slice, counts as that share, so the accounting stays aligned with the hardware.

In the Klique documentation: [Resource Management](#) covers policies, profiles and pools; [Resource Governance](#) covers reservations, limits and quotas; [Governed Compute](#) covers how queues reach each environment; and [Klique Orchestrator](#) covers the agent installed in each environment, called the worker daemon here.

Reservations and limits across environments

A resource policy gives a group a reservation, the GPUs guaranteed to it, and a limit, the most it can use at once. A group can run past its reservation on idle capacity, up to its limit. When a group below its reservation queues work, that work is placed first, and borrowed work can be preempted to make room; preempted work goes back to the front of its queue. Policies sit above the pools, so a group's reservation holds whichever environment runs its work.



Reservation

The GPUs a group is guaranteed. Reservations across groups should not exceed the capacity of the pools behind them.

Limit

The most a group can use at once, including borrowed capacity. A profile larger than the limit cannot be connected to the policy.

Execution priority

Within one pool, which profile's work runs first, for example small tasks before large ones.

Queuing priority

Within one profile, which group's work runs first when several groups are waiting.

Task size set by the task

A profile can read the size from a task parameter, so one profile covers tasks of different sizes. A default applies when the task sets none.

Every environment, one model

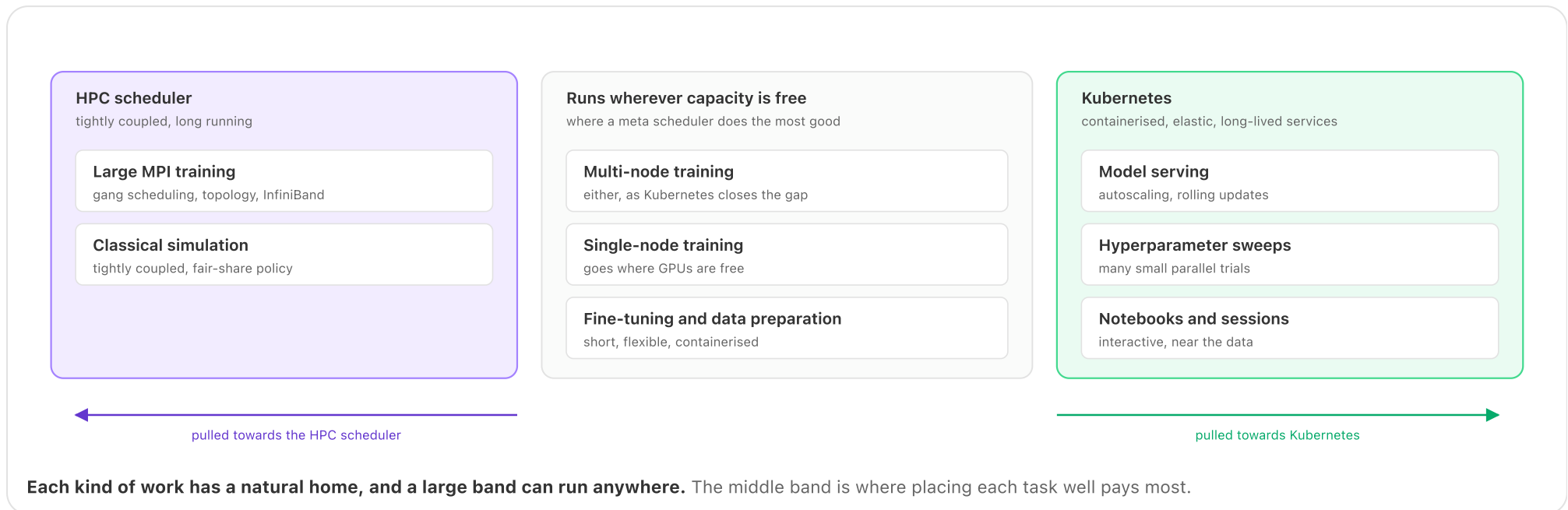
Each environment runs a Klique agent that pulls work for its pool. Clusters as a service sit beside them: Klique can stand up a Slurm cluster on Kubernetes, or a K3k virtual Kubernetes cluster, for a team that wants its own scheduler or API. The agent connects outbound to the control plane, so environments need no inbound ports. On an HPC cluster the compute nodes need outbound HTTPS to the control plane too, not only the login node, because the task's process on each node reports results, logs and metrics home. Adding an environment is the same each time: install the agent, define a pool and connect it to profiles.

ENVIRONMENT	KLIQUE COMPONENT	PLACES THE WORK	HOW A TASK RUNS
Kubernetes, on-prem	A worker daemon per cluster or namespace, installed with Helm	The Kubernetes scheduler	A pod from the compute resource's template, with GPU limits, node selectors and volumes
Slurm, PBS or LSF	The HPC submitter, one per cluster, on the login node	The cluster's own scheduler, with its own partitions and accounting	A batch job from a template, submitted with sbatch, qsub or bsub; the task runs natively or in a Singularity or Pyxis container
Bare metal and VMs	A worker daemon on each host	The worker daemon, on the host's free GPUs	A container in Docker mode, or a virtual environment on the host
Public cloud VMs	An autoscaler, run by the apps worker daemon, in the customer's account	One instance per task, of the instance type the autoscaler instance sets	An instance started on demand, up to the autoscaler's maximum instances, and stopped after an idle timeout
Managed Kubernetes, neoclouds	A worker daemon per cluster	The Kubernetes scheduler, with the cluster autoscaler adding nodes	Pods, as on any Kubernetes cluster
Slurm on Kubernetes	A Klique clustering service, launched on a Kubernetes cluster	Slurm, running in pods	Existing sbatch scripts, with worker pods that scale with demand
K3k	A Klique clustering service: Kubernetes in Kubernetes, launched inside a host cluster	The virtual cluster's own Kubernetes API, on the host cluster's nodes	A team gets its own Kubernetes cluster and namespaces without its own hardware

How public cloud and managed Kubernetes capacity is added and removed, including VM autoscaling, managed Kubernetes node autoscaling and pool order across on-prem and cloud, is covered in the [Cloud Autoscaler reference architecture](#).

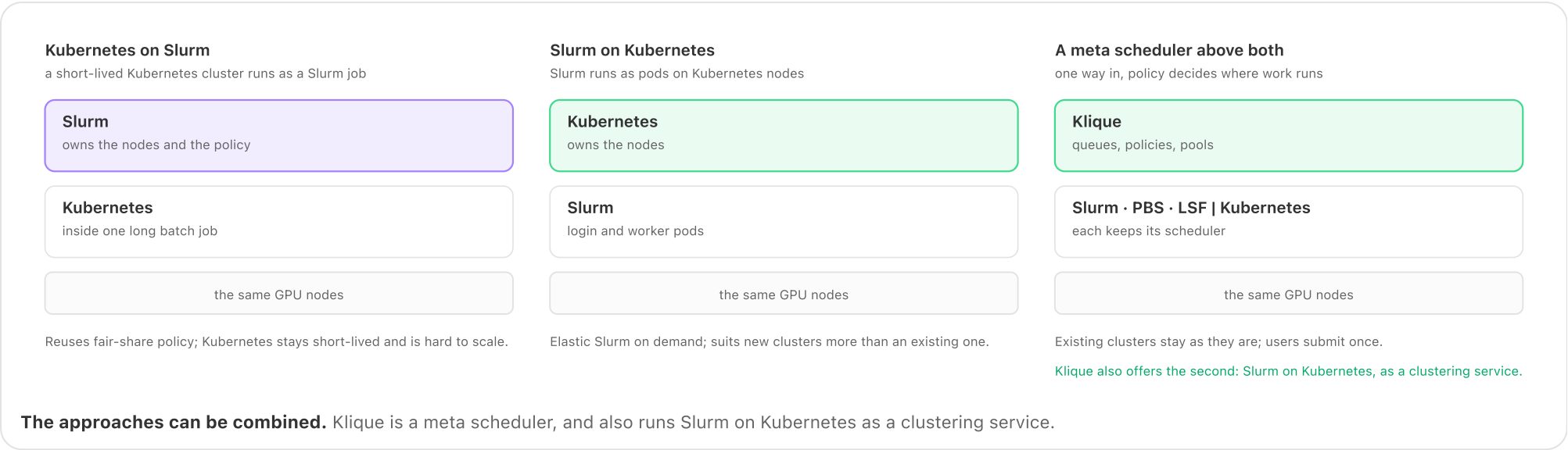
Usage patterns

AI work now spans two kinds of scheduler. HPC schedulers were built for tightly coupled work on a shared cluster, with fair-share queues, gang scheduling and placement that follows the network topology. Kubernetes was built for containerised, elastic, long-lived services. Most estates run both, often on the same kind of GPUs, and each kind of work has a natural home.



The middle band is where a meta scheduler does the most good. Single-node training, fine-tuning and data preparation go to whichever environment has free GPUs, and multi-node training can run on either side as Kubernetes gains the batch features HPC schedulers have long had. On HPC a task runs natively or in a Singularity or Pyxis container, so the same image can run under Slurm and on Kubernetes.

Three ways to bring HPC and Kubernetes together



Where mixed estates break, and how Klique closes the gaps

Data and storage

Data on a parallel filesystem, while Kubernetes expects volumes and object storage, often ends in copies in both worlds. Present the same shared filesystem to every pool; Klique attaches volumes to tasks through templates, so a task finds its data at the same path wherever it runs.

Identity

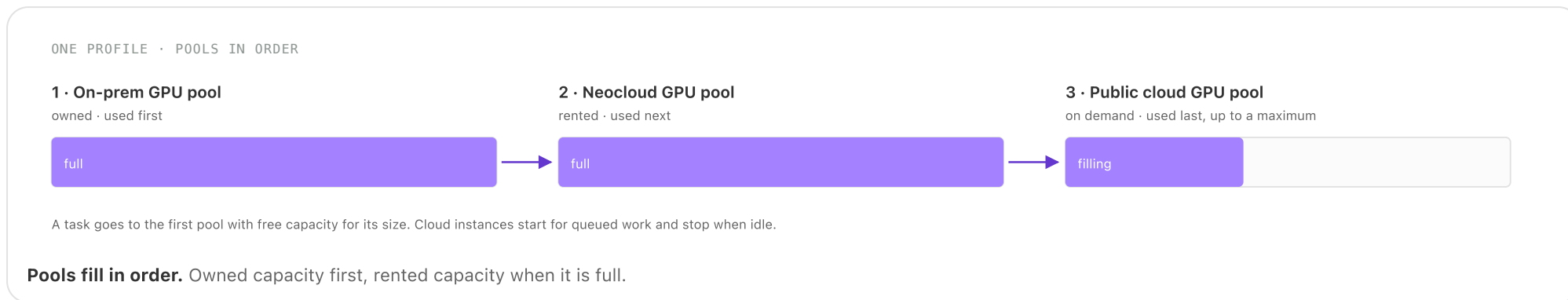
HPC clusters use Linux accounts and directory services; Kubernetes uses service accounts and RBAC. Klique gives one sign-on and one set of roles for every environment, and on Slurm each batch job can run as the submitting user's own Linux account.

Operations

Two submission paths, two queues and two monitoring stacks double the work for users and administrators. Klique gives one way in, one view of every queue, one dashboard and one ledger across environments.

Placement patterns

Pool order is set per profile, not per user. A task goes to the first pool in the order with free capacity for its size.



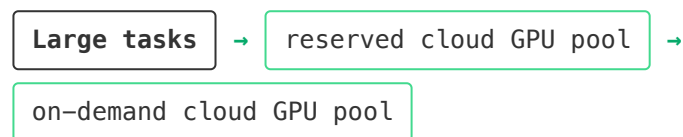
The patterns below are common starting points; in each line the first item is the compute resource and the rest is the pool order.

Owned capacity first, rented capacity for peaks



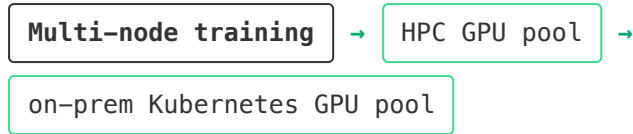
Work fills the owned GPUs first. When they are full it goes to the neocloud, then to on-demand cloud instances, up to the autoscaler's maximum instances.

Committed capacity before on-demand



Committed hours are billed whether they are used or not, so they come first. On-demand instances start only when the reserved capacity is full.

Multi-node training on HPC or Kubernetes



Large distributed tasks go to the HPC cluster first, where its scheduler places all the nodes together, and to multi-node scheduling on Kubernetes when the cluster is busy.

Interactive sessions near the data



Sessions stay on the pool next to the data and the shared storage, on a memory-capped share of a GPU.

A compute resource per GPU class



Each GPU class gets its own compute resource, so a task asks for the class it needs and lands on any pool that has it.

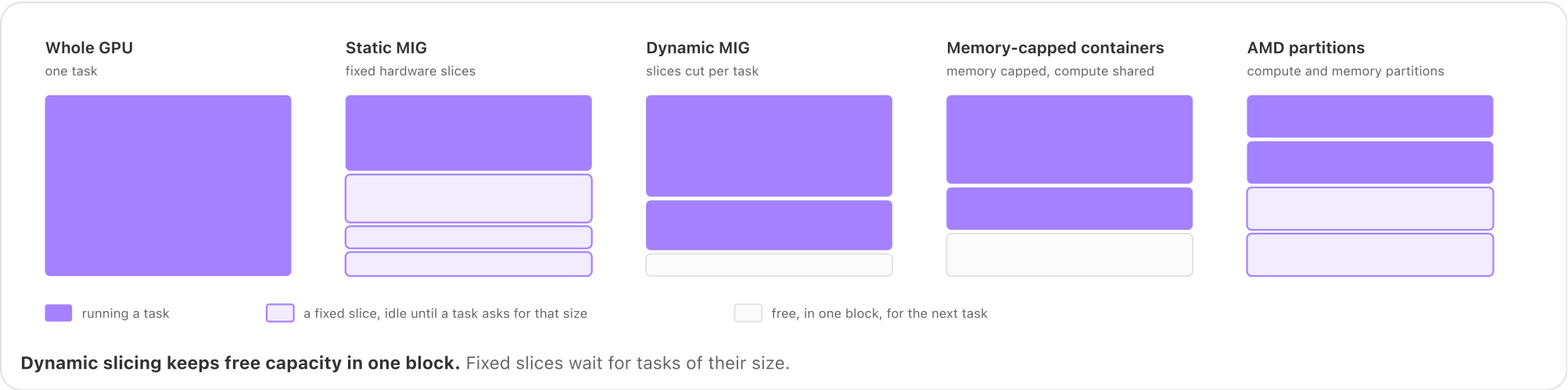
Data that stays in a region



The project's profiles list only pools in the permitted region, so its work cannot be placed elsewhere. Its output destination is set to storage in that region; by default outputs go to the control plane's file server. Code comes from git or the image, secrets come from vaults and are injected at start, and logs and metrics stream home. The [Cloud Autoscaler reference architecture](#) covers where the work lives.

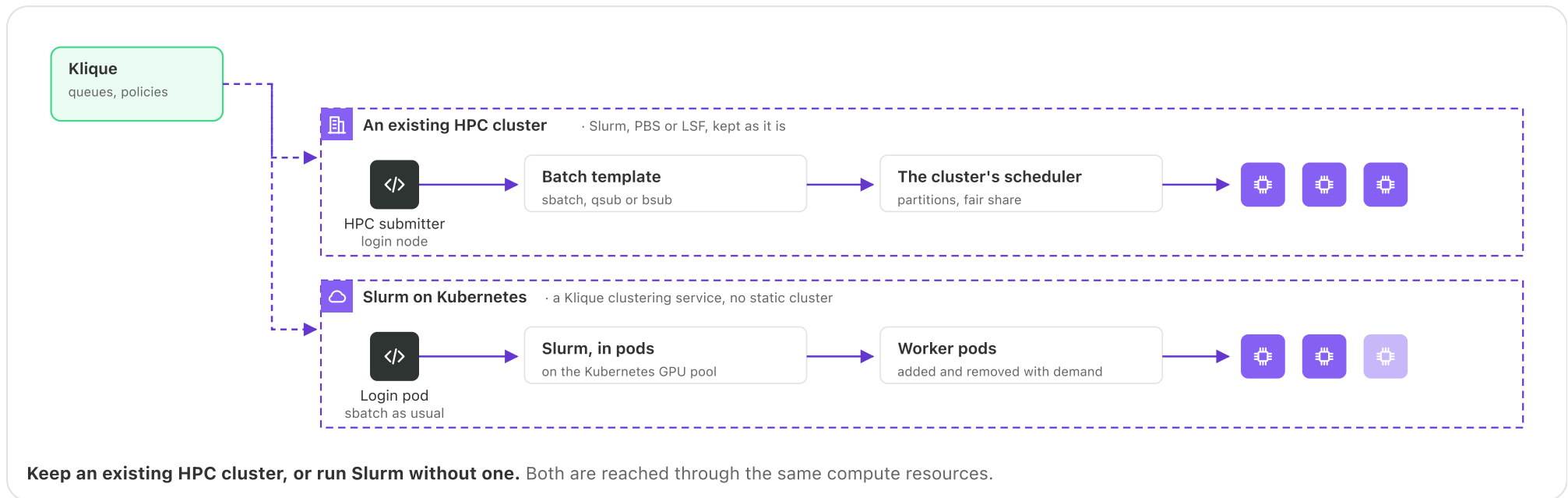
GPU sharing on any hardware

A GPU can be shared between tasks in several ways, depending on the hardware and the environment. Shares are counted in the pool like whole GPUs, so a task on a share of a GPU uses that share of the group's reservation.



METHOD	ISOLATION	WHERE
Static MIG	Hardware slices in a fixed layout; tasks request a slice	NVIDIA GPUs with MIG, on Kubernetes
Dynamic MIG	Hardware slices created per task; the GPU is repartitioned as demand changes	NVIDIA GPUs with MIG, on Kubernetes
Memory-capped containers	GPU memory capped per container; compute shared by time slicing	Any NVIDIA GPU: on Kubernetes, and on bare metal and VMs in Docker mode
AMD partitions	Compute and memory partitions set on the GPU	AMD Instinct GPUs
CPU offload for model serving	Idle models move to CPU memory, so one GPU serves more models	Model endpoints on vLLM, SGLang and llama.cpp

Multi-node and HPC work



On HPC clusters

The batch template can take the node count and other directives from the task. The HPC submitter submits it as a batch job with the cluster's own command, and the task runs natively or in a container on the nodes the scheduler allocates.

As the user's own account

On Slurm, the HPC submitter's user map (`agent.slurm.user_map`) runs each batch job as the Linux account mapped to the Klisque user who submitted it. A user with no mapping is refused unless a default account is set. PBS and LSF follow the same submitter pattern, an HPC submitter on the login node with a batch template. The [GPU as a Service reference architecture](#) covers HPC user mapping and sessions.

On Kubernetes

Multi-node scheduling runs one task across pods on several nodes. The task states the GPUs per node and the number of nodes.

Slurm on Kubernetes, a clustering service

Klique can run a Slurm cluster as pods on Kubernetes, with a login pod and worker pods that scale with demand. Existing sbatch scripts work unchanged.

Kubernetes in Kubernetes (K3k), a clustering service

Klique can run a virtual Kubernetes cluster inside a host cluster, so a team or tenant gets its own Kubernetes API and namespaces. Its pods run on the host cluster's nodes.

What Klique adds to an HPC cluster

Queue visibility and reordering, environment provisioning, experiment tracking and GPU sharing. Users submit through Klique instead of logging in to the cluster.

Layers above the scheduler

Scheduling GPUs on a Kubernetes cluster is one layer. Klique covers that layer in every environment and adds governance and services above it, on the same platform and with the same identity.

KLIQUE ADDS

Agent access and secure sandboxes	MCP server · agents act with the user's identity, quota and meter · coding agents in sessions, calling models with the user's own Klique token
Metering and billing	GPU-hours, tokens and storage metered per tenant, project and user · exported to billing or chargeback
Development lifecycle	Experiment tracking · data versioning · pipelines · model registry
Tokens and models	LLM gateway · routing rules · token budgets and wallets · guardrails
Self-serve services	App catalogue · remote IDEs · model serving · Slurm and Kubernetes clustering services
Tenants	Isolated tenants · Platform Management Center · each tenant's share and usage
Identity and access	Single sign-on · RBAC on projects, queues and apps · service accounts · vaults
Scheduling across environments	Kubernetes, Slurm, PBS, LSF, bare metal, VMs, public cloud providers, neoclouds · ordered pools · NVIDIA and AMD GPU sharing

A GPU SCHEDULER ON ONE CLUSTER COVERS

GPU scheduling, one cluster	Queues · quotas · GPU fractions · priority · on one Kubernetes cluster. Klique covers this layer too, in every environment.
-----------------------------	---

Scheduling is the bottom layer. Every layer above it applies to work in every environment.

Scheduling across environments

Kubernetes, Slurm, PBS, LSF, bare metal, VMs, public cloud providers and neoclouds behind the same compute resources, with ordered pools and GPU sharing on NVIDIA and AMD hardware.

Identity and access

Single sign-on with the company's identity provider, RBAC on projects, queues and apps, service accounts for automation, and vaults for secrets injected into tasks.

Tenants

One control plane can serve many isolated tenants, such as business units or the customers of a provider, each with its own users, projects, compute resources, policies and worker daemons. The provider sets each tenant's share, and the Platform Management Center shows compute, usage and cost across tenants. The [Multi-Tenant reference architecture](#) covers this in detail.

Self-serve services

An app catalogue with remote IDEs, model serving and custom apps, and clustering services for teams that want their own Slurm or Kubernetes cluster, all on governed compute and reached through internal routes.

Tokens and models

An LLM gateway in front of self-hosted and public models, with routing rules, token budgets and wallets, and guardrails. Its self-hosted models run on the same GPU pools this scheduler manages. See the [LLM Gateway and Tokenomics reference architecture](#).

Development lifecycle

Experiment tracking, data versioning, pipelines and a model registry, recorded for every task whichever environment ran it.

Metering and billing

Klique meters GPU-hours, tokens and storage per tenant, project and user, and exports the usage to the billing or chargeback system.

Agent access and secure sandboxes

An MCP server, so AI agents can submit and manage work with the identity, quota and metering of the user they act for. Coding agents run in sessions: credentials come from the platform's vaults, and model calls go through the LLM gateway, where a user's gateway key is their own Klisque token. Compute and tokens are governed in one place, under the same identity and budgets.

Visibility and metering

Orchestration dashboard

Available and used GPUs, CPUs and workers in every environment, grouped by category and group, with utilization over time and an event log.

Policy view for users

Users see the policies that apply to them: current usage, reservation, limit and the compute resources available to them.

Across tenants

The Platform Management Center shows orchestration, usage and cost for every tenant in the deployment.

Metering

GPU-hours are recorded per user, project and pool, by GPU class. Klique meters usage per tenant and exports it to the billing or chargeback system.

Compared with a cluster scheduler

A GPU scheduler for one Kubernetes cluster and a meta scheduler solve different problems. The left column describes a typical Kubernetes GPU scheduler; check a specific product's documentation for its details.

CAPABILITY	A GPU SCHEDULER ON ONE KUBERNETES CLUSTER	KLIQUE
Where work runs	Pods on one Kubernetes cluster	Kubernetes clusters, Slurm, PBS and LSF clusters, bare metal, VMs, public cloud providers and neoclouds
When a cluster is full	Work waits in the cluster	Work goes to the next pool in the profile's order, including rented capacity
GPU sharing	Fractions on the cluster's GPUs	Static and dynamic MIG, memory-capped containers and AMD partitions
Quotas	Per project, in the cluster	Reservation and limit per group, held across every pool
Identity	Cluster accounts and roles	Single sign-on, RBAC, service accounts and identities for agents
Model tokens	Not covered	LLM gateway with routing rules, budgets and guardrails
Billing	Not covered	GPU-hours, tokens and storage metered per tenant, project and user, and exported to the billing or chargeback system
Services for users	Workloads on the cluster	App catalogue, remote IDEs, coding agents, model serving and experiment tracking

Design considerations

Match each agent to its profiles

The resource manager counts GPUs; the agent applies them. Set GPU limits in the pod or batch template to match each profile, or tasks are counted correctly but run with the wrong allocation.

Size pools to the hardware

Count each pool in whole GPUs, whatever the mix of whole, fractional and MIG work it runs. The scheduler treats a pool as full when its declared capacity is used, so keep the declared capacity equal to the real GPUs.

Keep reservations within capacity

The sum of reservations on a pool should not exceed what the pool holds, or a reservation cannot always be met.

Order pools by cost and locality

Owned capacity first, committed capacity next, on-demand last. Keep interactive sessions near the data.

Plan for preemption

Borrowed work can be preempted when a group below its reservation needs capacity. Preempted work goes back to the front of its queue. Long tasks should save checkpoints.

Map HPC users

On Slurm, map Klique users to Linux accounts in the user map so batch jobs run as the person who submitted them, and set a default account if unmapped users should not be refused.

Set cloud limits

Give each autoscaler instance an instance type, a maximum instance count and an idle timeout. A cloud pool with no running instances still takes work, because the autoscaler registers as its listener and scales from zero. The [Cloud Autoscaler reference architecture](#) covers these settings.

Allow outbound only

Agents connect out to the control plane over HTTPS; no environment needs inbound ports from it. On HPC clusters, allow outbound HTTPS from the compute nodes as well as the login node.

Glossary

Meta scheduler	A scheduler that decides which environment runs a task, above the schedulers that place it on nodes.
Compute resource	What a user submits work to, named for the allocation it gives, such as multi-node training or a shared GPU: a queue, one per policy and profile. IT decides which pools serve it.
Access channel	A way to submit work: the UI and its apps and workflows, the SDK and CLI, the REST API, AI agents over MCP, or CI/CD pipelines.
Pool	Capacity in one environment, counted in whole GPUs.
Resource profile	A task size and an ordered list of pools.
Resource policy	A user group's reservation and limit, and the profiles it may use.
Reservation	The GPUs a group is guaranteed.
Limit	The most a group can use at once, including idle capacity it borrows.
Klique worker daemon	The agent that pulls tasks for a pool and runs them on Kubernetes, bare metal, VMs or a neocloud. The Klique documentation calls it the Klique Orchestrator.
HPC submitter	The agent on an HPC cluster's login node that turns tasks into batch jobs for Slurm, PBS or LSF.
Apps worker daemon	The agent that runs app controllers, including cloud autoscalers.

Autoscaler	Starts cloud instances for queued work in the customer's account, of the instance type it sets and up to its maximum instances, and stops them after an idle timeout.
Clustering services	Slurm on Kubernetes, and Kubernetes in Kubernetes (K3k): clusters Klique runs on a Kubernetes cluster for teams that want their own scheduler or Kubernetes API.
Neocloud	A provider that rents GPU capacity, usually as Kubernetes clusters or VMs.

Klique AI · Meta-scheduler reference architecture · one control plane scheduling across every environment. Group names and pools are illustrative.