

PyTorch Conference 2026

Lessons at scale: building a PyTorch inference stack with open-source components

From a single forward pass to fleet scale, mapped to the hardware and fabric beneath it.

Oscar Savolainen, PhD

Senior Staff Specialized AI Engineer, Nscale

What we'll build, layer by layer

01 Physical data center layout the map we keep coming back to

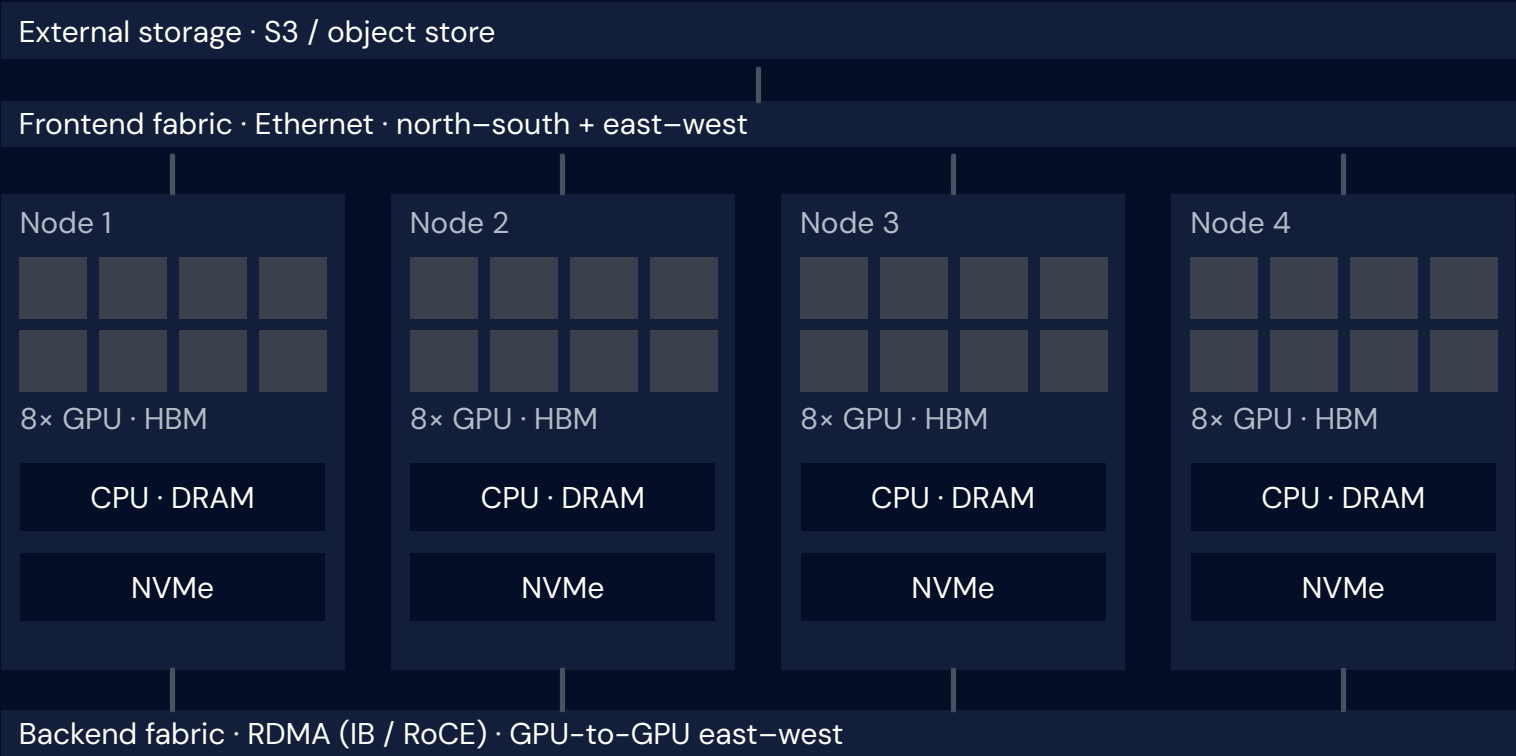
In vLLM / SGLang, out of the box

- 02 Weight loading
- 03 Dynamic batching, web server and KV cache
- 04 Compilation
- 05 Multi-node

Needs a higher-order orchestrator

- 06 GPU HBM management → storage hierarchy
- 07 Multi-instance and autoscaling
- 08 Cache-aware routing
- 09 AI gateway and rate limiting
- 10 Cold start optimization

How a compute cluster is laid out



Compute nodes: GPUs, CPUs and local NVMe.

Backend fabric: GPU to GPU, over RDMA.

Frontend fabric: everything else, in and out and node to node.

External storage: S3 / object store.

We'll keep coming back to this picture.

Open source

Kubernetes · KubeRay · NVIDIA GPU Operator · NVIDIA Network Operator

Physical data center layout

Inside a node: three tiers of memory



GPU HBM: the fastest and smallest. Weights and KV cache live here.

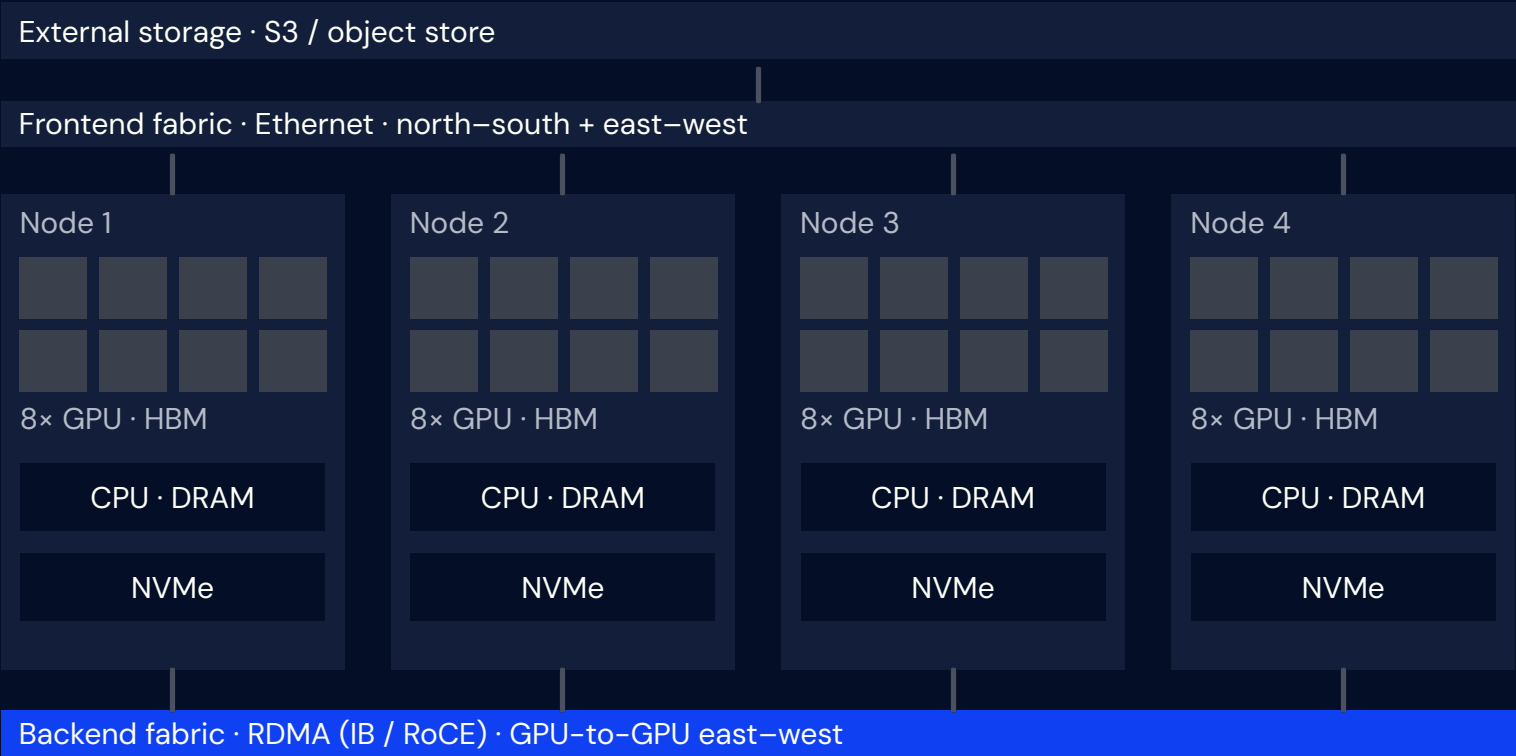
CPU DRAM: much larger, slower. Runtime, tokenization, offload space.

Local NVMe: fast persistent disk, per node. A warm cache for weights and artifacts.

Inside a node (or a whole NVL72 rack), GPUs talk over NVLink.

Open source NVIDIA GPU Operator · DCGM Exporter · KubeRay · Ray resource scheduling

Backend fabric: GPU to GPU

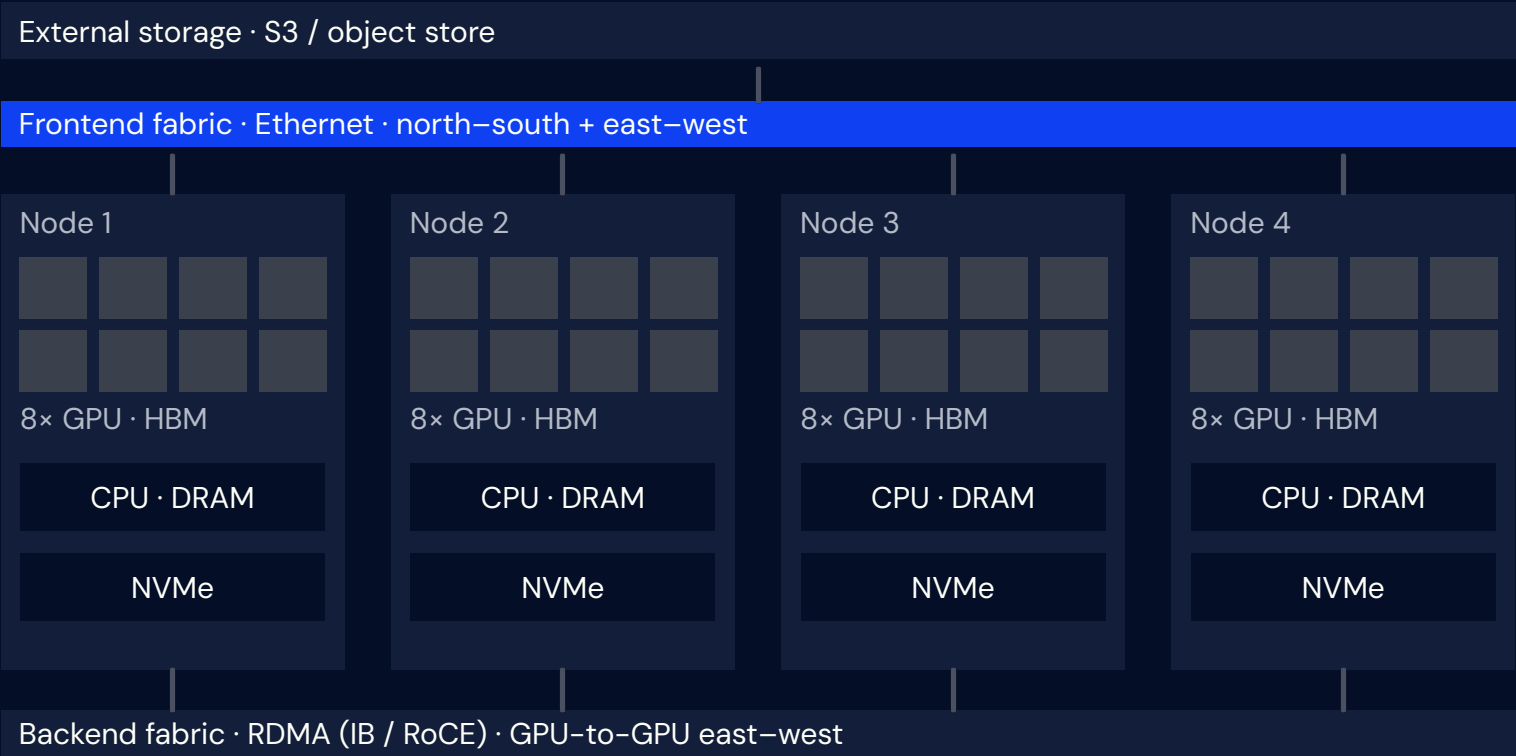


- Dedicated east-west:** a fabric built for GPU-to-GPU traffic.
- RDMA:** InfiniBand or RoCE. Memory to memory, bypassing the CPU.
- One rail per GPU:** a dedicated NIC for each GPU.
- Carries:** collectives, activations between model shards, KV and weight transfers.

Open source NCCL · NIXL · UCX · NVIDIA Network Operator

Physical data center layout

Frontend fabric: everything else

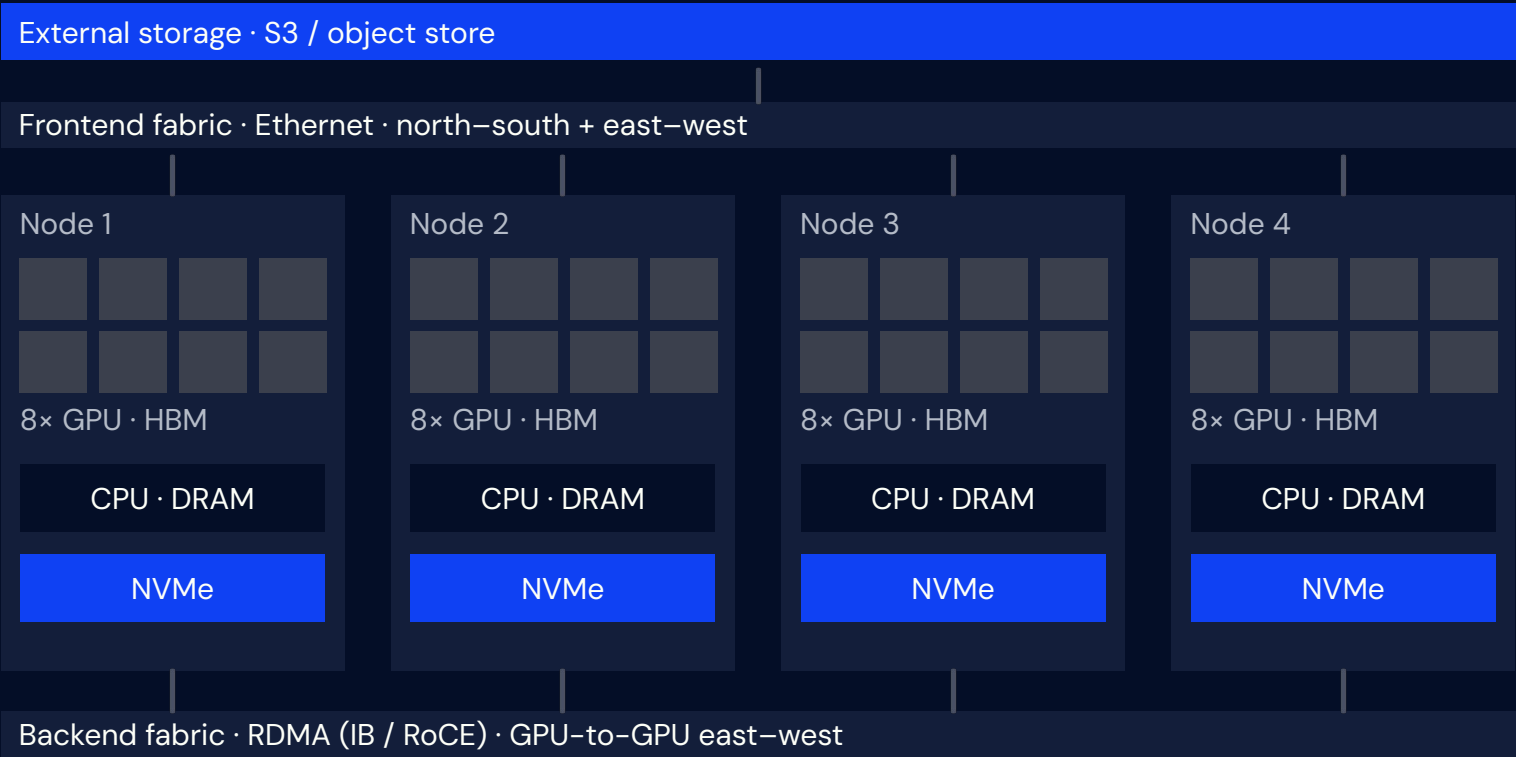


- North-south:** in and out of the cluster.
- East-west too:** node to node, when both sit inside the same network boundary.
- User traffic:** API requests in, tokens streamed out.
- Control and storage I/O:** SSH, Kubernetes, monitoring, object storage.

Open source Envoy Gateway · Cilium · Kubernetes Gateway API · Ray Serve proxy

Physical data center layout

Storage: local and external



Object storage: durable, effectively unlimited, shared by every node.

Reached over: the frontend fabric, unless you wire it up over RDMA.

Local NVMe: fast, but only on the node that has it.

Bandwidth vs durability: we'll trade these off later.

Open source Ceph · MinIO · JuiceFS · Ray Data

Physical data center layout

Where do we get the weights?

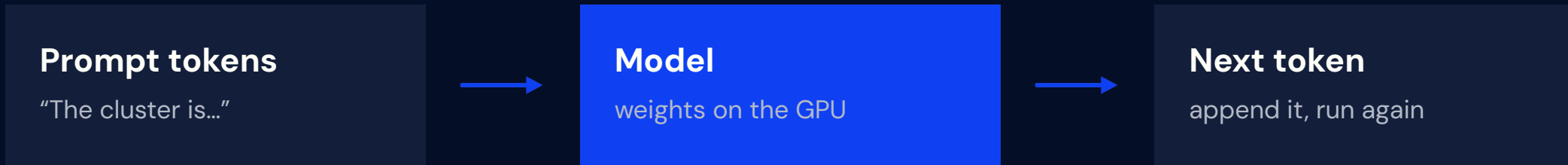


Open source

Hugging Face Hub

In vLLM / SGLang · Weight loading

Start simple: one forward pass



No batching

One request at a time. Decode is memory-bound, so the GPU sits mostly idle.

No KV cache

Every new token recomputes attention over the whole sequence. Cost grows with every step.

Open source

PyTorch · Hugging Face Transformers

In vLLM / SGLang · Dynamic batching, web server and KV cache

Add continuous batching and a KV cache

Every forward pass: whoever is ready joins the batch.

KV cache: each request keeps its keys and values. One more token per step.

Finished? It leaves, and a waiting request takes the slot.

Open source

vLLM · SGLang · TensorRT-LLM · Ray Serve LLM

In vLLM / SGLang · Dynamic batching, web server and KV cache

torch.compile is an easy optimization lever

01

Eager PyTorch

Runs op by op from Python.
Great UX: why researchers love it.



02

TorchDynamo

Traces your Python into a graph (a DAG) of the whole workload.



03

TorchInductor

Compiles fused kernels:
Triton by default, other backends available.



04

Runtime

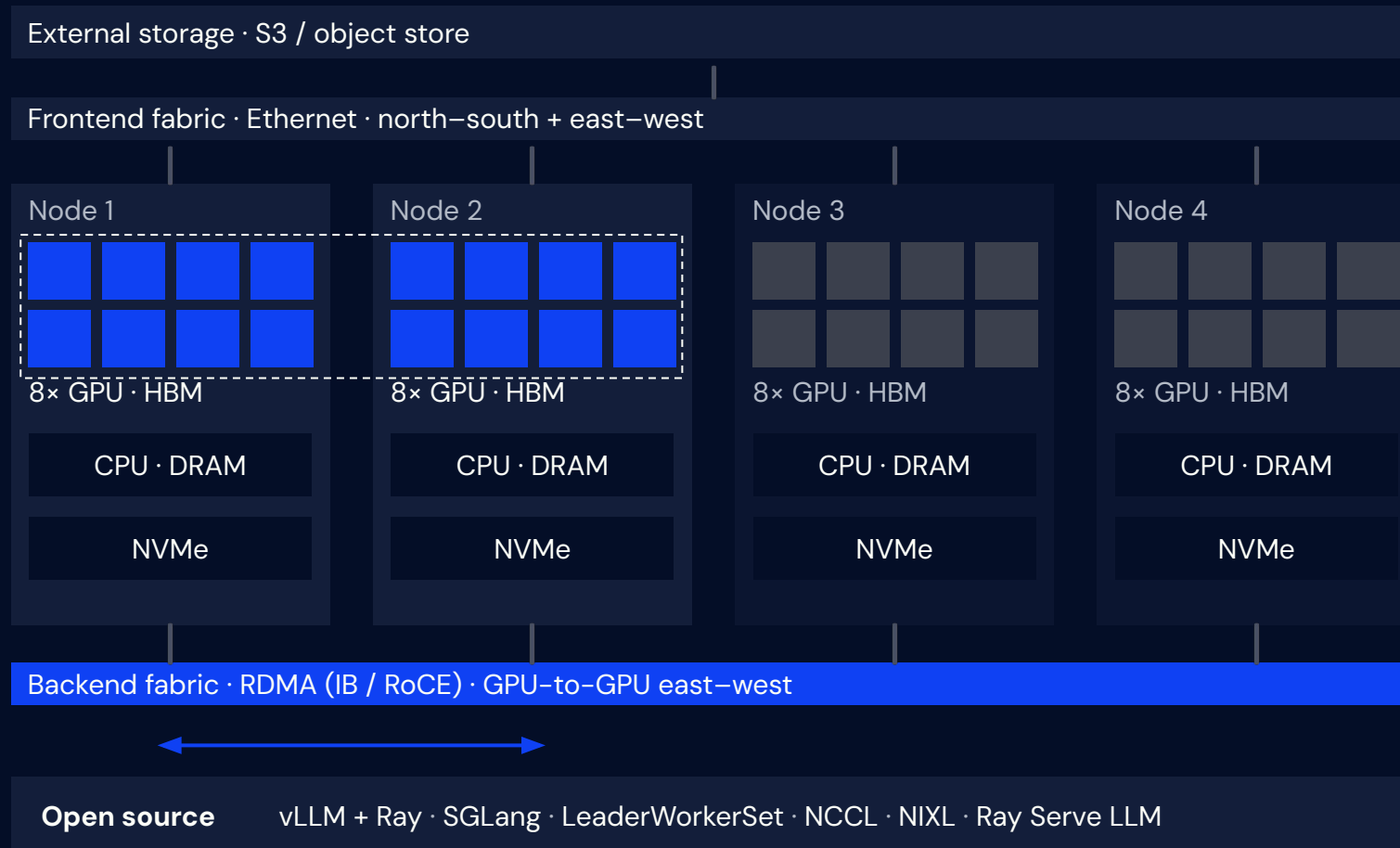
Executes the compiled graphs instead of individual ops.

Compilers want the whole graph to reason across. Eager mode only shows them one op at a time.

Open source

PyTorch · TorchDynamo · TorchInductor · Triton

When the model doesn't fit on one node



Shard across nodes: tensor parallel inside a node over NVLink; pipeline or expert parallel across nodes.

Every forward pass now crosses the backend fabric over RDMA.

Keep shards together: Ray placement groups or Kubernetes LeaderWorkerSet.

Put it behind a web server and... we have a very basic version of vLLM and SGLang

HTTP API

OpenAI-compatible, streaming

Scheduler

continuous batching

KV cache manager

paged blocks, prefix reuse

Model executor

kernels on the GPU

Up to here: vLLM and SGLang, out of the box. **From here:** an orchestrator (Ray Serve, Dynamo, KServe...).

Open source

vLLM · SGLang · Ray Serve LLM

In vLLM / SGLang · Checkpoint

Part 2

Needs a higher-order orchestrator

Past a single vLLM or SGLang instance: GPU memory, gateways, autoscaling, routing and cold starts.

First problem: managing GPU memory at scale

One GPU's HBM (illustrative proportions)



Fixed costs

Model weights, runtime and CUDA context, compiled kernels and CUDA graphs.

The variable cost

KV cache. It grows with every token of every in-flight request.

In an agentic world, context keeps growing. Long context is the norm.

Open source

vLLM (PagedAttention) · SGLang (RadixAttention)

Needs a higher-order orchestrator · GPU HBM management → storage hierarchy

The KV cache grows until it doesn't fit

Every request grows a token per forward pass.

HBM full: the scheduler preempts a request to let the others finish.

Preempted: evicted, requeued, and its prefill done all over again.

Open source

vLLM · SGLang · Ray Serve LLM

Needs a higher-order orchestrator · GPU HBM management → storage hierarchy

Offload KV while the session is idle

GPU HBM



Queue

CPU DRAM



A, B, C decoding. D is waiting: no HBM left.

Agentic loop: the model works, then makes a tool call.

Meanwhile: move its KV to CPU DRAM and give the HBM to a queued request.

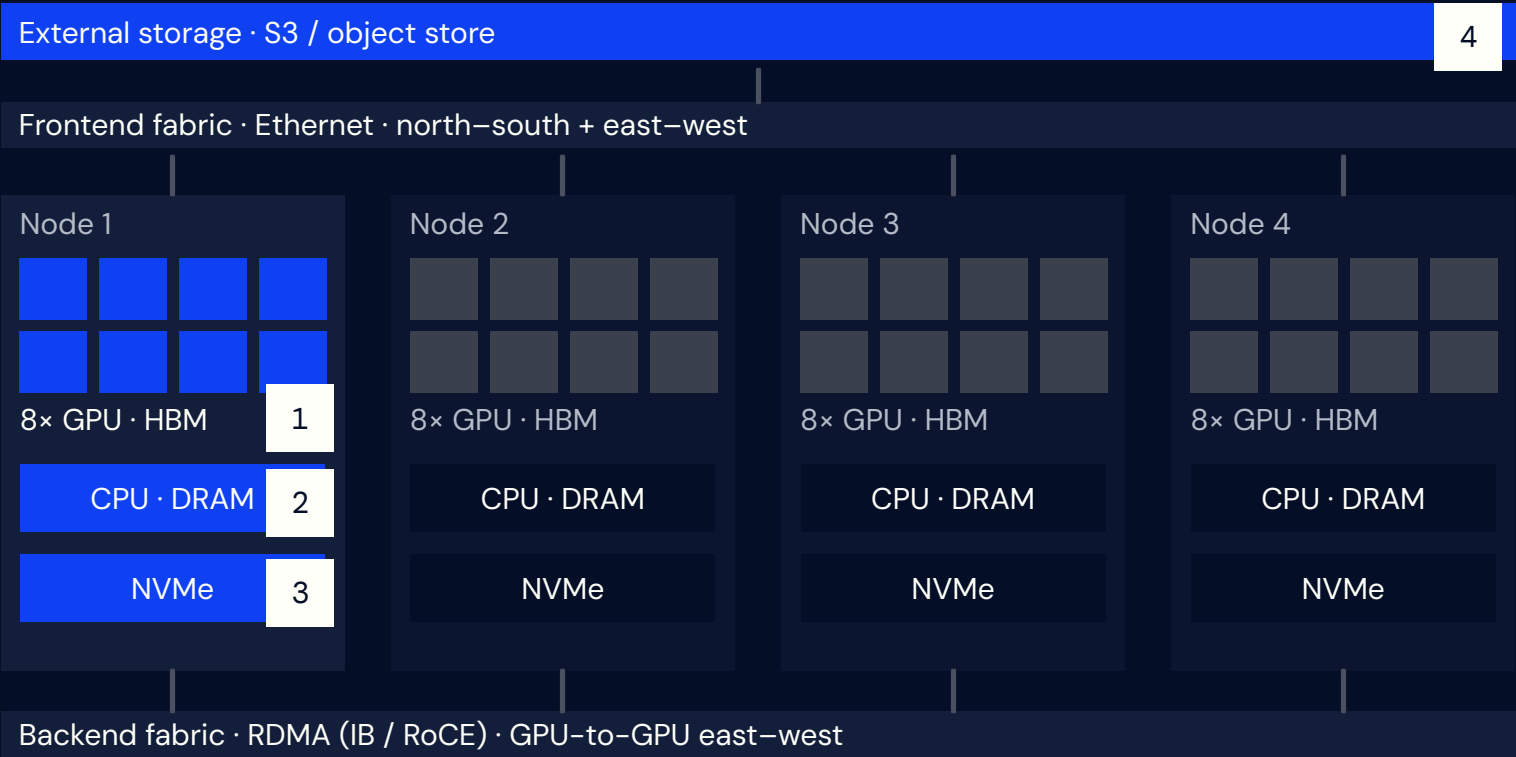
Tool returns: reload the KV instead of recomputing it.

Open source

LMCache · NVIDIA Dynamo KVBM · Mooncake · vLLM KV connectors · Ray Serve LLM

Needs a higher-order orchestrator · GPU HBM management → storage hierarchy

Now we have a storage hierarchy



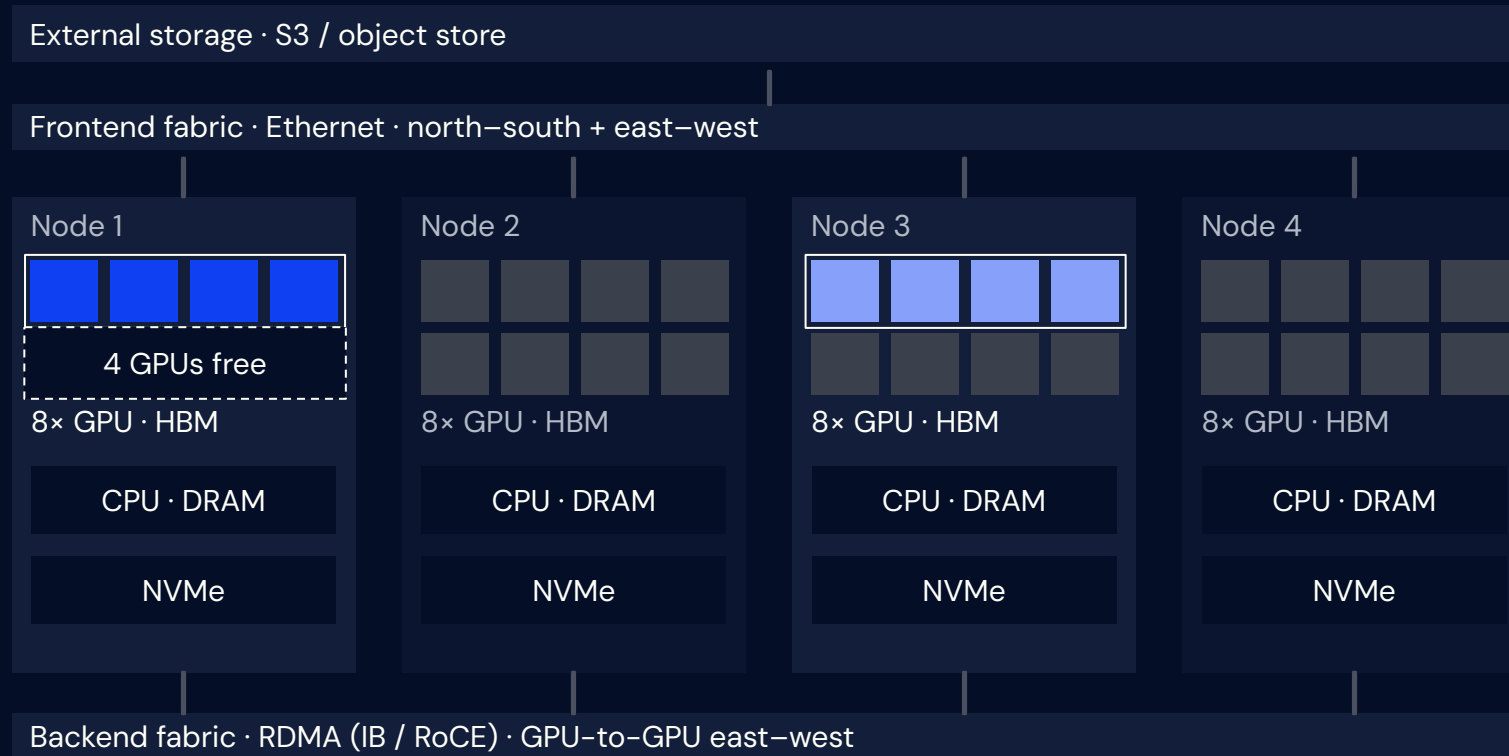
- 1 · **GPU HBM** where the KV is used.
- 2 · **CPU DRAM** next to the GPU, large.
- 3 · **Local NVMe** bigger again, per node.
- 4 · **S3** shared and durable, reached over the network.

Each step down: bigger, slower, further away.

Open source LMCache · NVIDIA Dynamo KVBM · Mooncake Store · llm-d KV offloading

Needs an orchestrator · GPU HBM management → storage hierarchy

Multiple instances, on different nodes



Each replica: 4 GPUs. Both would fit on Node 1.

Anti-affinity forces the copy onto another node anyway, so one node failure can't take out both.

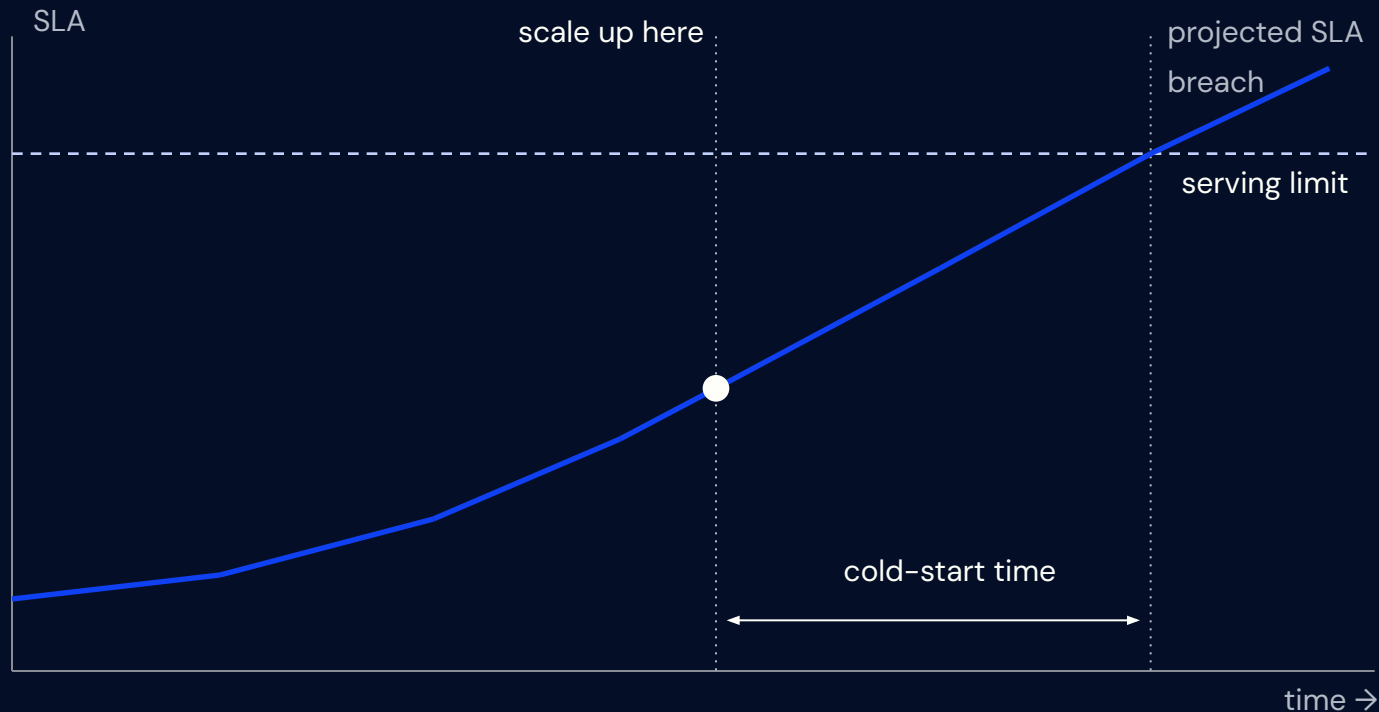
```
Kubernetes:  
podAntiAffinity on  
kubernetes.io/hostname
```

```
Ray Serve:  
max_replicas_per_node=1
```

Open source Kubernetes anti-affinity · KubeRay · Ray Serve autoscaling

Needs an orchestrator · Multi-instance and autoscaling

Autoscaling: scale based on demand



Concurrent requests: in-flight per replica. Simple, and the usual default.

KV-cache pressure: how full each replica's KV cache is right now.

Fill rate: how fast it's filling. The trend, not just the level.

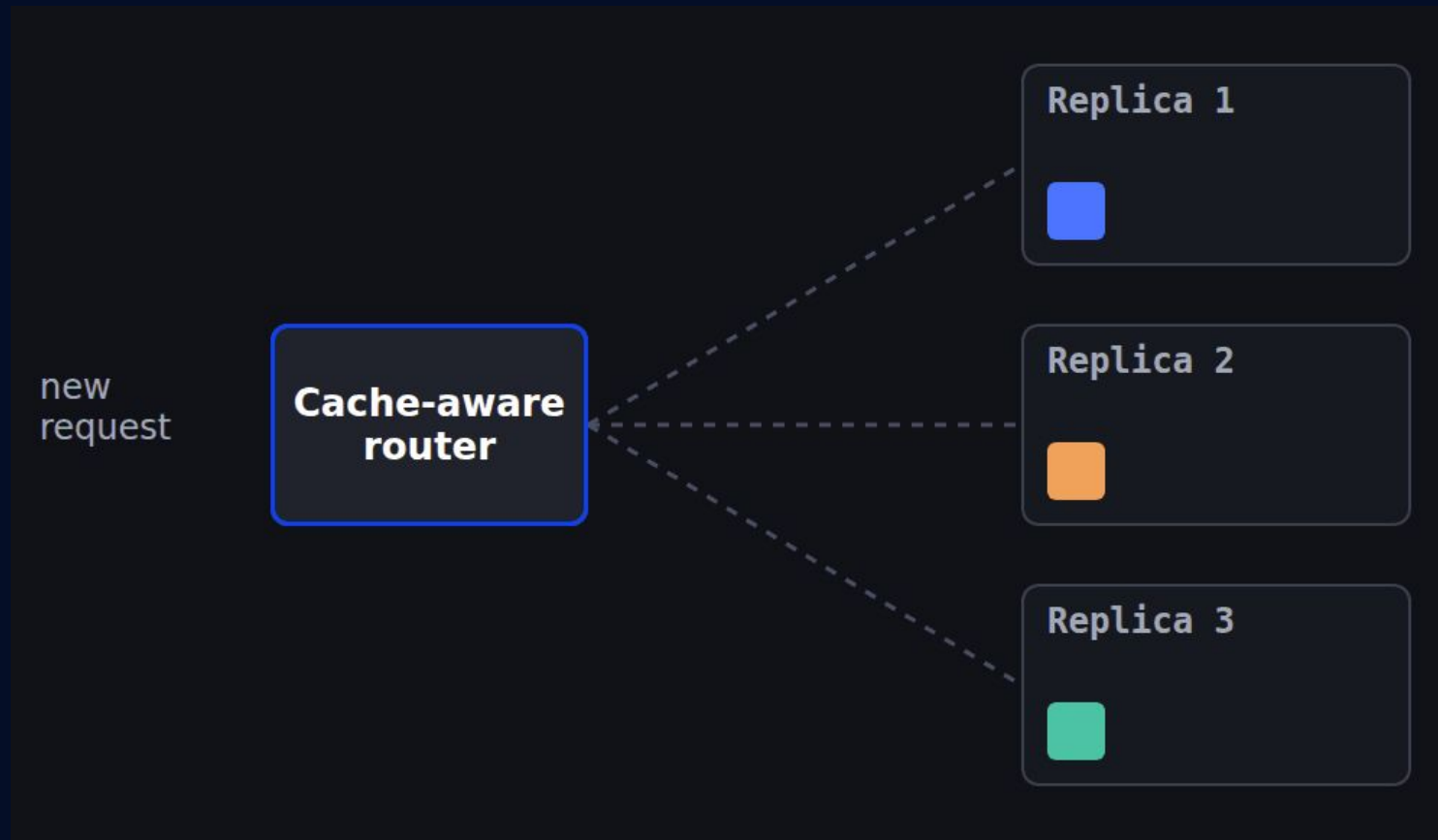
Cold-start time: trigger early enough that the new replica is ready before you hit the SLA wall.

Open source

Kubernetes HPA · KEDA · KServe autoscaling · Ray Serve autoscaling

Needs a higher-order orchestrator · Multi-instance and autoscaling

Cache-aware routing



Route to the prefix: send each request where its KV already lives.

Miss: recompute the whole prefix. Slower TTFT, wasted GPU.

So the router must know where every block is: a distributed caching problem.

Built in: llm-d, KServe

LLMInferenceService, NVIDIA Dynamo, Ray Serve LLM, SGLang router.

Open source

Ray Serve LLM routers · KServe LLMInferenceService · llm-d · NVIDIA Dynamo · SGLang router

Needs a higher-order orchestrator · Cache-aware routing

An AI gateway in front of everything

Route by model

Read the model name from the request body and send it to the right pool.

Meter tokens

Track input and output tokens per user, per model.

Fail over

Fall back to another pool or provider when one is down.

Open source

Envoy AI Gateway · Envoy Gateway · Gateway API Inference Extension · LiteLLM · Ray Serve

Needs a higher-order orchestrator · AI gateway and rate limiting

Rate limiting: keep one user from starving the rest

Requests

Requests per minute. Simple, but a short and a 100k-token request count the same.

Tokens

Input and output tokens per minute. The common default for LLM APIs.

Spend

A cost budget per user or team, priced per model.

Concurrency

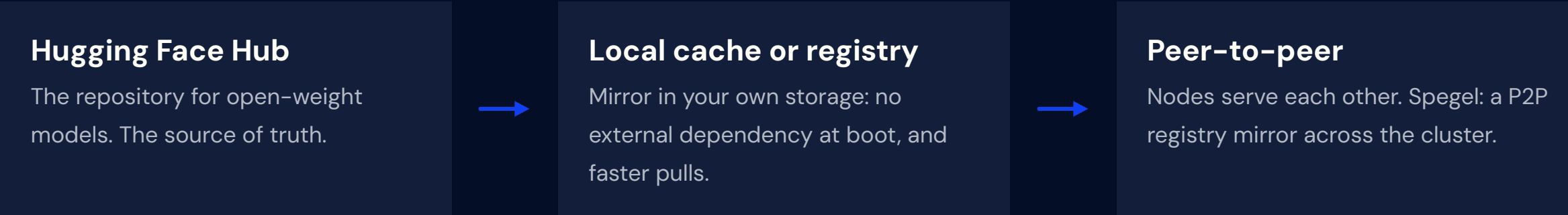
Cap in-flight requests per user, whatever their size.

Open source

Envoy AI Gateway · Envoy rate limit service · LiteLLM · Redis

Needs a higher-order orchestrator · AI gateway and rate limiting

How to load weights quickly?



Local NVMe makes P2P viable
With multiple TB of NVMe per node, every node can keep warm weights and serve them to its peers.

One node's NVMe

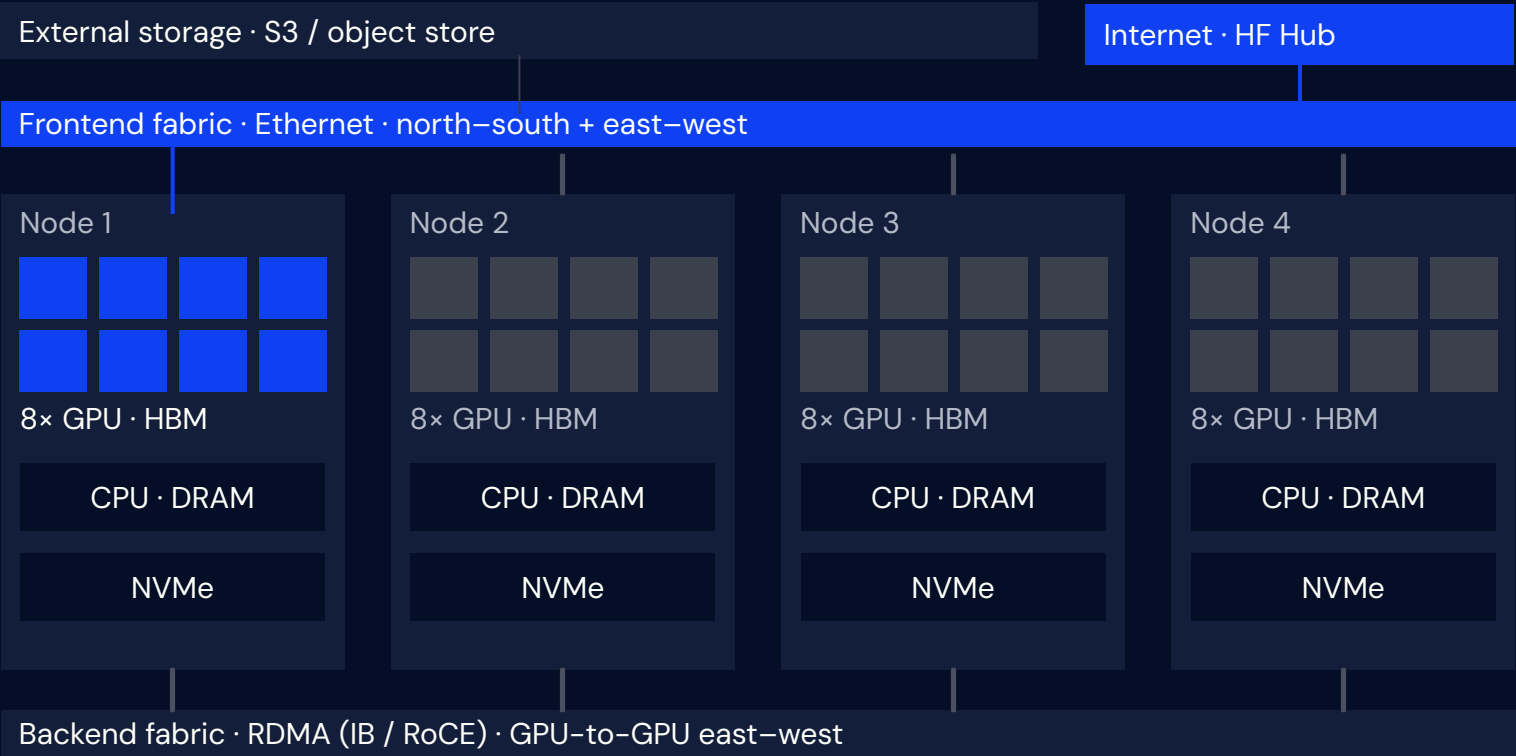
KV-cache tier	Model weights	Free
---------------	---------------	------

Open source

Hugging Face Hub · Spegel · Dragonfly · Run:ai Model Streamer · Ray Serve LLM

In vLLM / SGLang · Weight loading

Option 1: straight from Hugging Face



Path: internet → frontend fabric → node.

Easy: nothing to run yourself.

But: the slowest path, rate limits, and an external dependency every time you boot.

Open source

huggingface_hub · hf_transfer

In vLLM / SGLang · Weight loading

Option 2: from your own object storage



Path: object storage → frontend fabric, north-south → node.

Better: durable, under your control, no internet in the boot path.

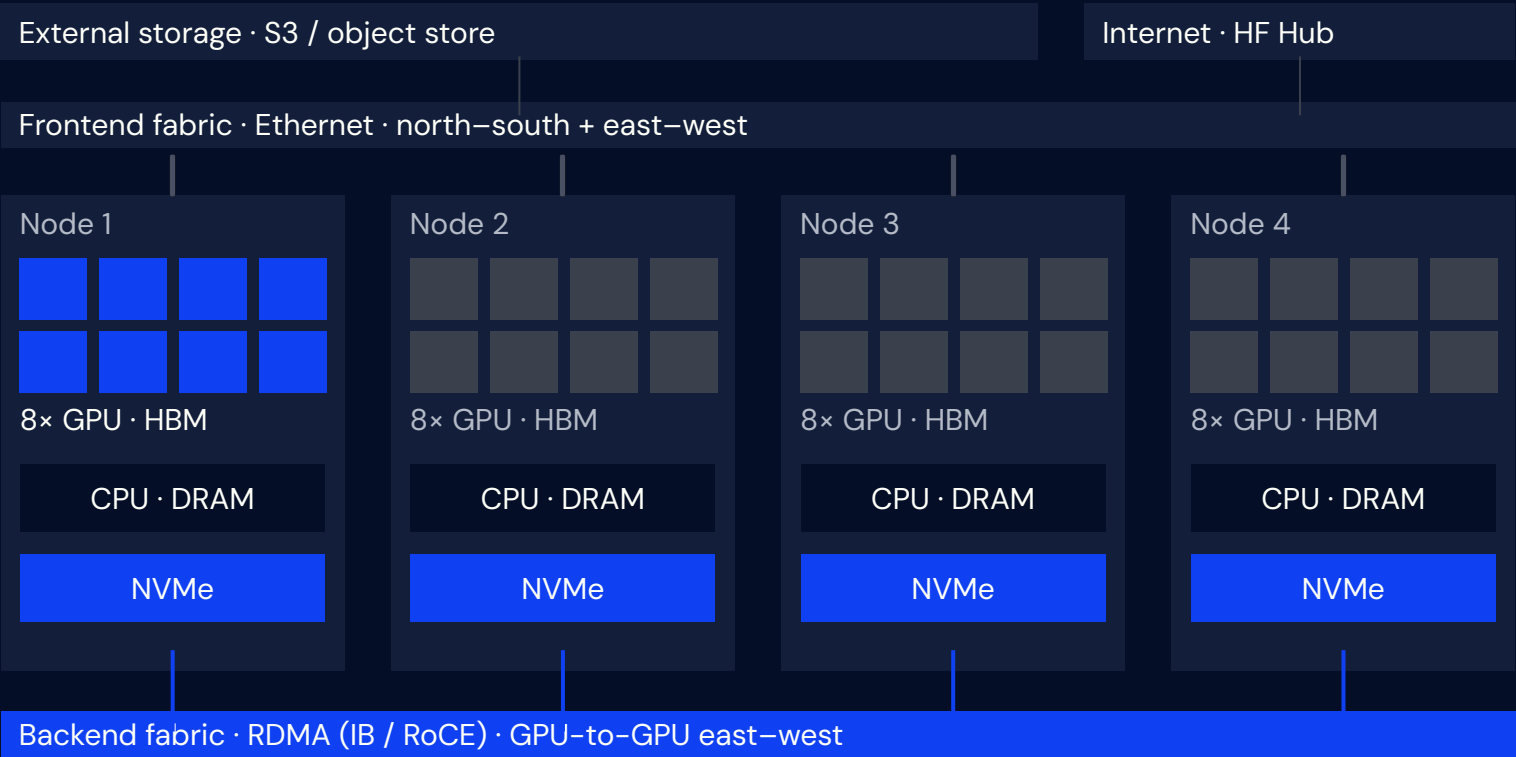
But: every new replica pulls the full model over shared bandwidth if done naively.

Open source

Run:ai Model Streamer · MinIO · Ceph

In vLLM / SGLang · Weight loading

Option 3: from peers, over RDMA



Path: shards on every node's NVMe → backend fabric, east-west → the new replica.

A distributed registry: every node holds part of the cache, not just the nodes already running this model.

Fastest: many sources in parallel over RDMA, and it scales with the cluster.

Open source

Spegel · Dragonfly · NIXL · NVIDIA Dynamo ModelExpress

In vLLM / SGLang · Weight loading

Compile ahead of time, ship the artifacts

Eager: one op at a time



We want compilation, not compilation time on every cold start.

Compile once, AOT, for each model, GPU type and framework version.

Store alongside the weights: load them from NVMe, S3 or peers, just like weights.

Open source

torch.compile Mega-Cache · AOTInductor · vLLM compile cache · Ray Serve LLM

Needs a higher-order orchestrator · Cold start optimization

Three ways to make cold starts faster

Cache closer

Pull from a local cache, such as P2P NVMe over RDMA or on-premise s3, rather than from HF.

Do work AOT

Prepare as much work AOT as you can, e.g. shard your weights so each GPU pulls only its own section, or compile AOT.

Snapshot and restore

Checkpoint an initialised worker, GPU memory and CUDA context included, and restore new ones from it: Dynamo Snapshot.

Open source

Run:ai Model Streamer · NVIDIA Dynamo ModelExpress · Dynamo Snapshot · NIXL · Spegel

Needs a higher-order orchestrator · Cold start optimization

Or... have someone else run all of this

Deploy from a UI

Pick a model, get a working, scaled endpoint.

KV offload and routing

Tiered cache and cache-aware routing, handled.

Placement and rate limits

KV-aware scheduling and per-user budgets.

Fast, resilient restarts

Weights and compiled artifacts, pre-warmed.

Shameless plug: Nscale's Inference Management Service, built for both serverless and dedicated inference. Go to **console.nscale.com** or talk to us to get started.

PyTorch Conference 2026

Thank you

Questions welcome.

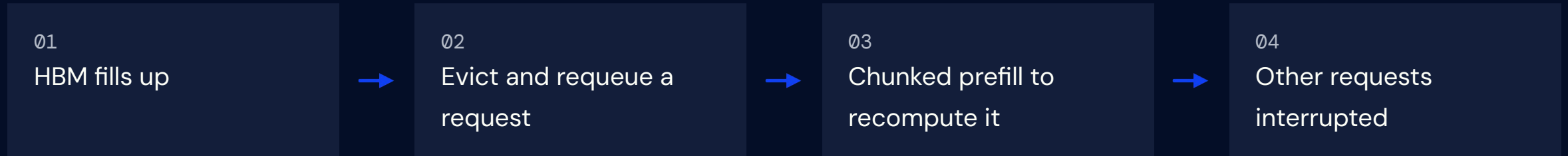


console.nscale.com

Oscar Savolainen, PhD

Senior Staff Specialized AI Engineer, Nscale

HBM preemption is expensive, and it compounds



The bill

Worse TTFT, worse inter-token latency, lower throughput, and compute spent twice on the same prefill.

The worst case

A death spiral: everything is preempted, nothing finishes. DoS.

Open source

vLLM · SGLang · Ray Serve LLM

Needs a higher-order orchestrator · GPU HBM management → storage hierarchy