Scalable AI: Building AI/HPC systems for pre-training

Fall 2026
UC Berkeley

Matt Nightingale - linkedin.com/in/matt-night/
Sean Smith - linkedin.com/in/seanwssmith/

Lecture 11: Building AI/HPC Systems for Pre-Training

Building and maintaining a GPU fleet for frontier-scale pre-training

Learning Goals

By the end of this lecture you should be able to:

1/ Explain the architectural drivers and parallelism requirements that necessitate GPU clusters for pre-training.

2/ Describe the key components of pre-training clusters: Compute, Network, Storage, Observability

3/ Understand the basic methodology for cluster bring up and validation including NCCL & Workload Benchmarking

4/ Interpret various infra and workload level metrics including MTBI, Goodput, OFU, and HFU

Roadmap for today

Part A: The Framing: What do we mean by a cluster, why do we need a cluster?

Part B: Cluster Bring Up and Testing

Part C: Measuring Success as a cluster operator

Part A:

The Framing: Why do we need a cluster? What is a cluster?

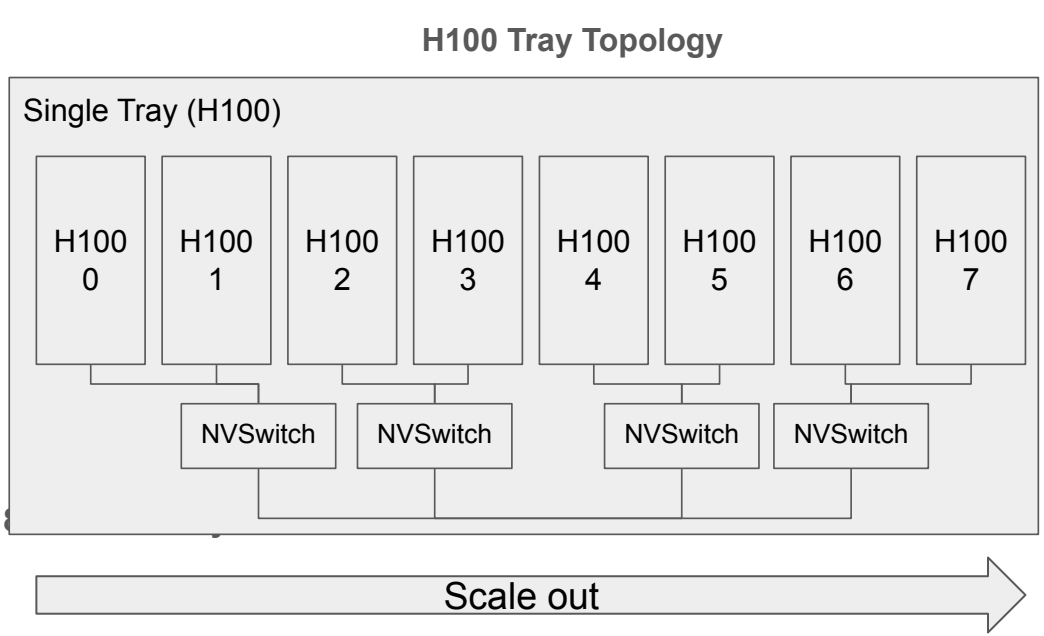
Why we need a cluster…

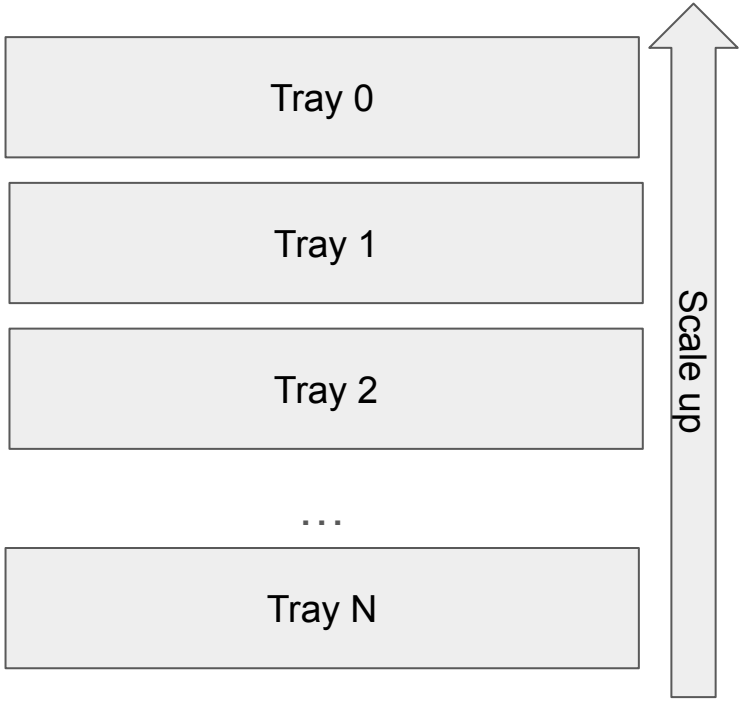
Limitations of a single node

- Two main limitations - memory and flops.
- Memory
  - As models scale in size, we implement model parallelism techniques such as PP, TP.
  - These techniques allow us to increase the size of models beyond a single gpu (80GB) but what happens when we hit the single node memory footprint of 640GB (8x H100) or 1152GB (4 x GB300)?
- Flops
  - A single H100 can do 1,979 TFlops (dense, FP8). How do we parallelize the pre-training of a huge (multi-trillion) parameter dataset?
  - As we increase the size of the dataset we implement Data Parallel (DP) techniques.
  - DP scales linearly i.e. each additional GPU allows us to process incrementally more Flops (with some performance degradation due to network overhead)
  - We can measure scaling efficiency with the following formula:

$$E = \frac{T_{compute}}{T_{compute} + T_{communication}}$$

Scale out vs Scale Up





Why we need a cluster…

Weak Scaling

In High Performance Computing (HPC) **weak scaling** refers to when adding more compute & more data scales linearly i.e. **Data Parallel**. This can be measured by **Gustafson's Law**:

$$\mathrm{S(N)} = (1 - p) + p \cdot N$$

S(N) = Scaled speedup using N processors.

N = Number of processors or processing nodes.

p = Fraction of the execution time that is parallelizable (measured on a single processor).

Assuming network is low overhead - we can scale **infinitely***!

Why we need a cluster…

Strong Scaling

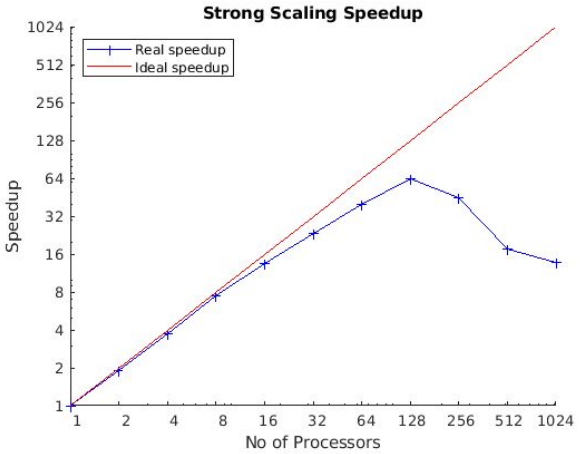
In HPC - **strong scaling** is when we add more compute but the problem size stays fixed i.e. **model parallelism.** This can be measured by Amdahl's law:

$$\text{Speedup} = \frac{1}{(1-P)+\frac{P}{N}}$$

- **P**: The proportion of the program that can be parallelized (e.g. .8).
- (1-P): The serial portion that cannot be parallelized (e.g., .2).
- **N**: The number of processors being used.

Strong scaling is often limited by network bandwidth.

$$\text{Max Speedup} = \frac{1}{(1-P)}$$

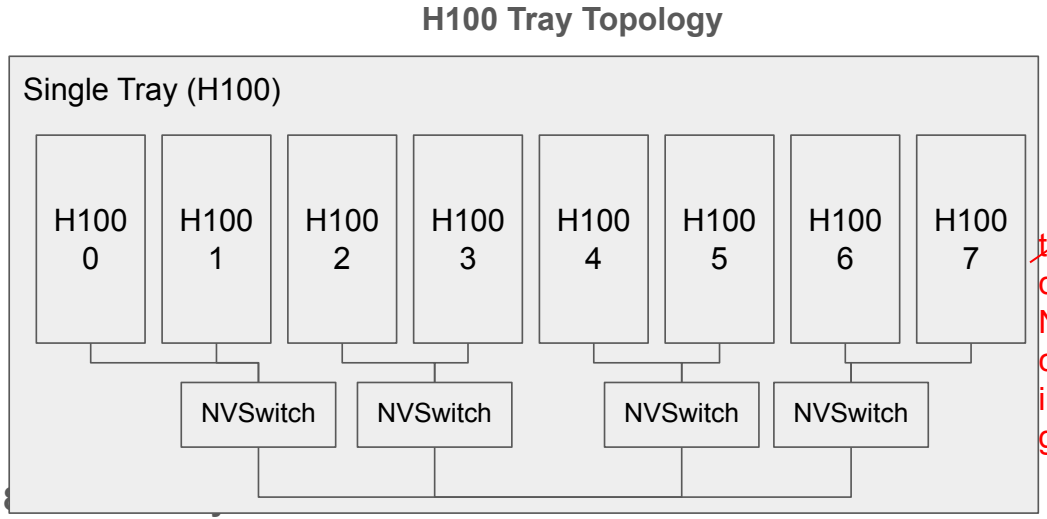


Why we need a cluster…

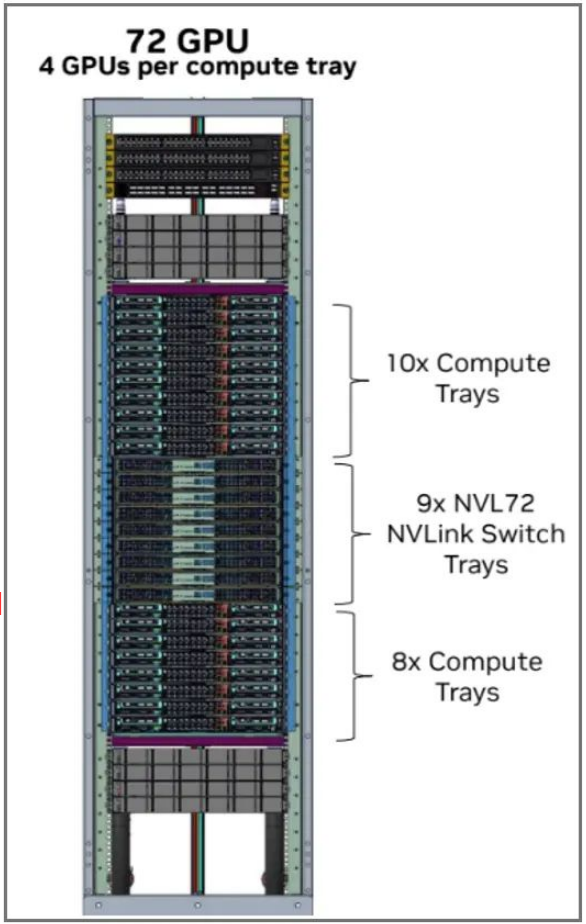
Strong scaling matters…

We want to stay here as much as possible for model parallelism

- HBM memory is 3,350 GB/s (67x faster)
- NVLink is 900 GB/s (18x faster)
- Infiniband 400 GB/s (1x baseline)

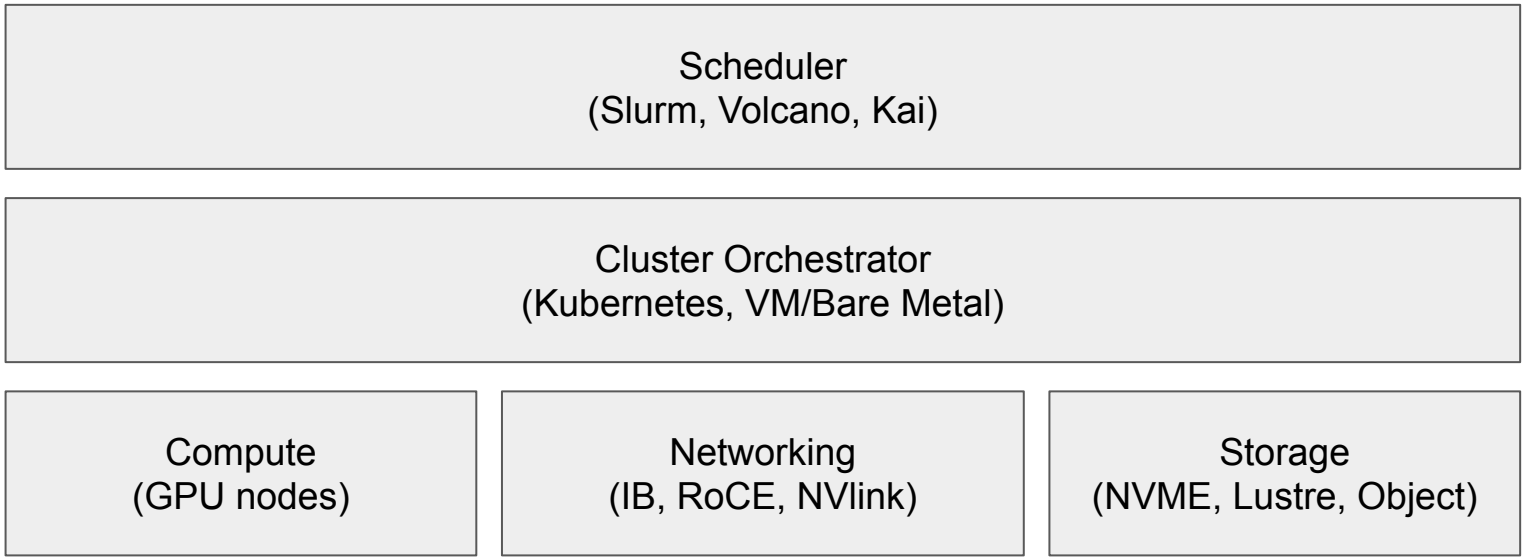


tray to rack level domain allows NVLink connections to increase 8 -> 72 gpus



What is a cluster…

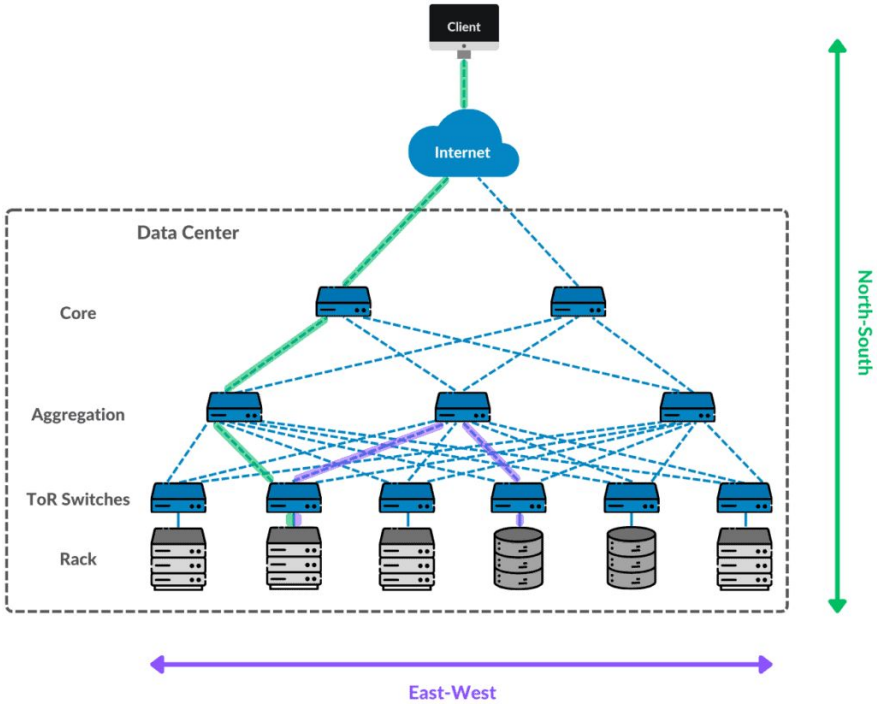
Components of a HPC Cluster



What is a cluster…

Network Layout

- Within a **tray**, fast communication between every **GPU**
  - Typically NVLink (for Blackwell + GPUs)
- Within a **rack**, fast communication between every **compute tray**
  - Typically NVLink through NVLink switches
- Determining how to group **groups of racks**
  - Scale out topology or scalable units
  - An NVLink domain may be used when supported by the rack scale architecture
- **East West** networking (Infiniband, RoCE, EFA)
  - Fast communication **between groups of racks**
  - Fast communication to Lustre through separate storage fabric
- **North South** networking (TCP)
  - Traffic to external services through private endpoints or the Internet



The faster these systems are, the bigger models we can train!

East West Networking

- East/West networking determines scaling efficiency for our models. This is where we'll spend most of our time.
  - Networking fabric is measured on two criteria, bandwidth & latency
  - All modern networks provide GPU to GPU Remote Direct Memory Access (**RDMA**). This allows the gpus to communicate directly and skip the host OS.
  - Common options are:
    - Infiniband - Nvidia product, based on UDP over fiber - 200GB/s per-gpu and <1us latency.
    - RoCEv2 - stands for RDMA over Converged Ethernet - 200GB/s per-gpu 2-5us latency
    - EFA (AWS specific) - 200GB/s per-gpu and 10-15us latency.

https://elice.io/en/resources/blog/infiniband-vs-rocev2-benchmark

Fat Tree Topology (East West)

T2 Spine

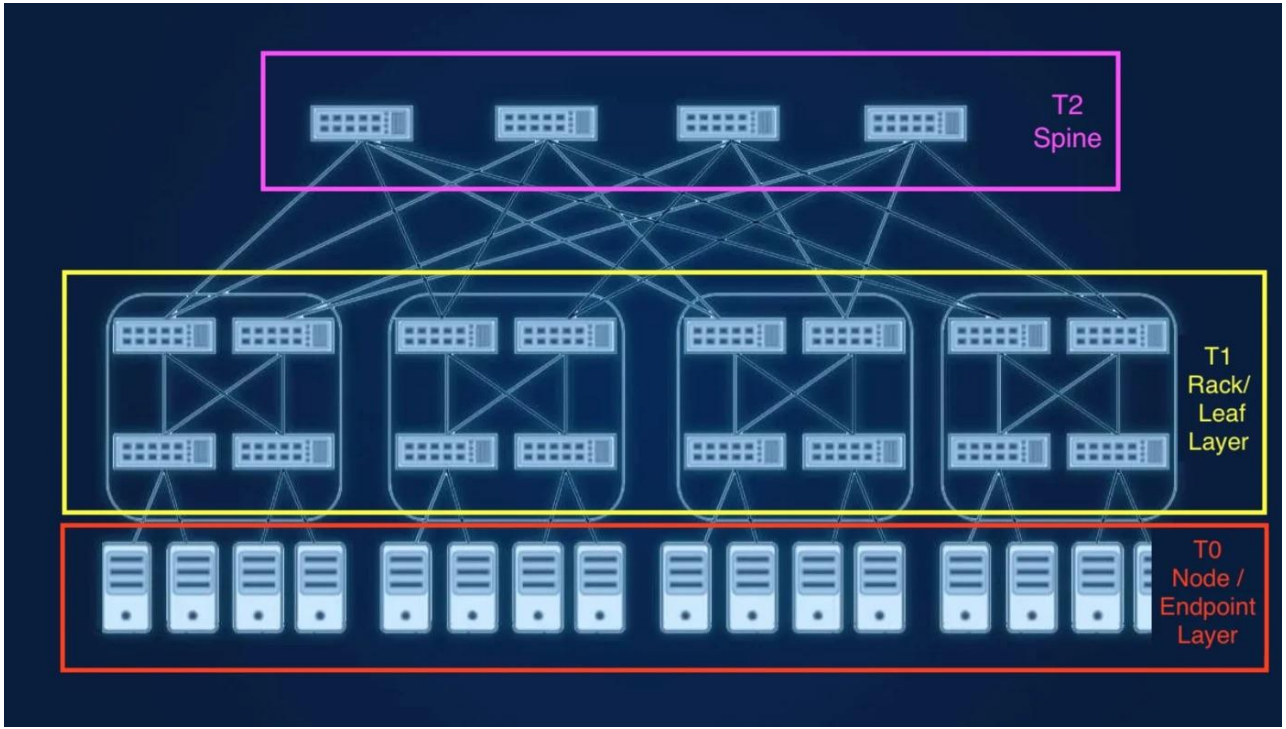
Spine switches interconnect the leaf switches using InfiniBand (some OCI) or Ethernet with RoCE (most OCI, GCP, NCPs), or EFA (AWS)

T1 Rack/Leaf Layer

Connected through “Top of Rack” (ToR) switches InfiniBand or Ethernet with RoCE. This layer provides scale out connectivity across nodes and racks.

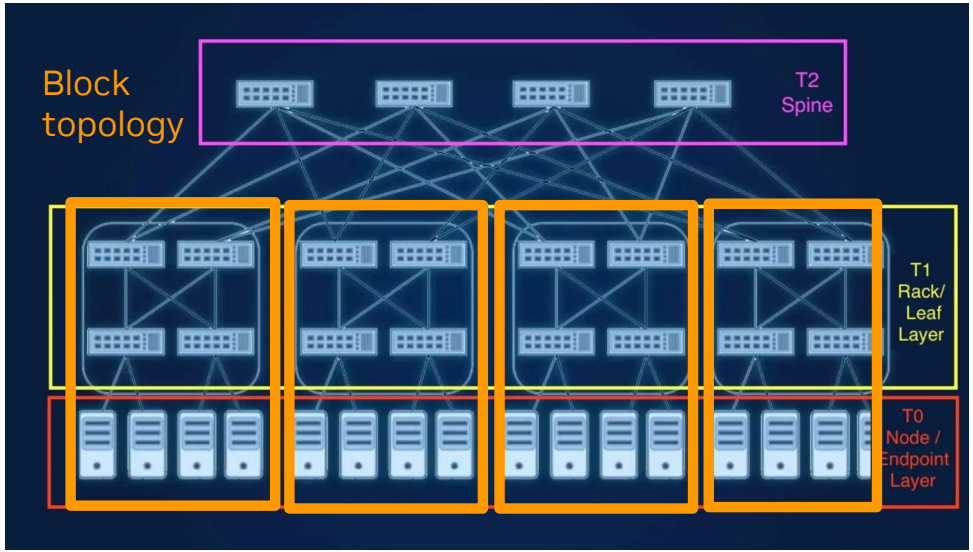
T0 Node/ Endpoint Layer

Compute endpoints include GPUs and network adapters such as HCAs, NICs, or EFA devices. NVLink and NVSwitch form the separate scale up domain within a node or supported rack scale system.

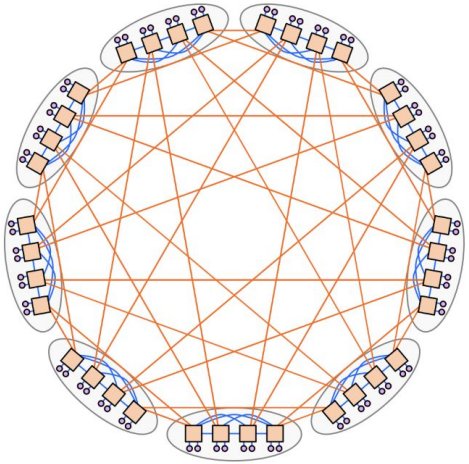


Block/Dragonfly Topology (East West)

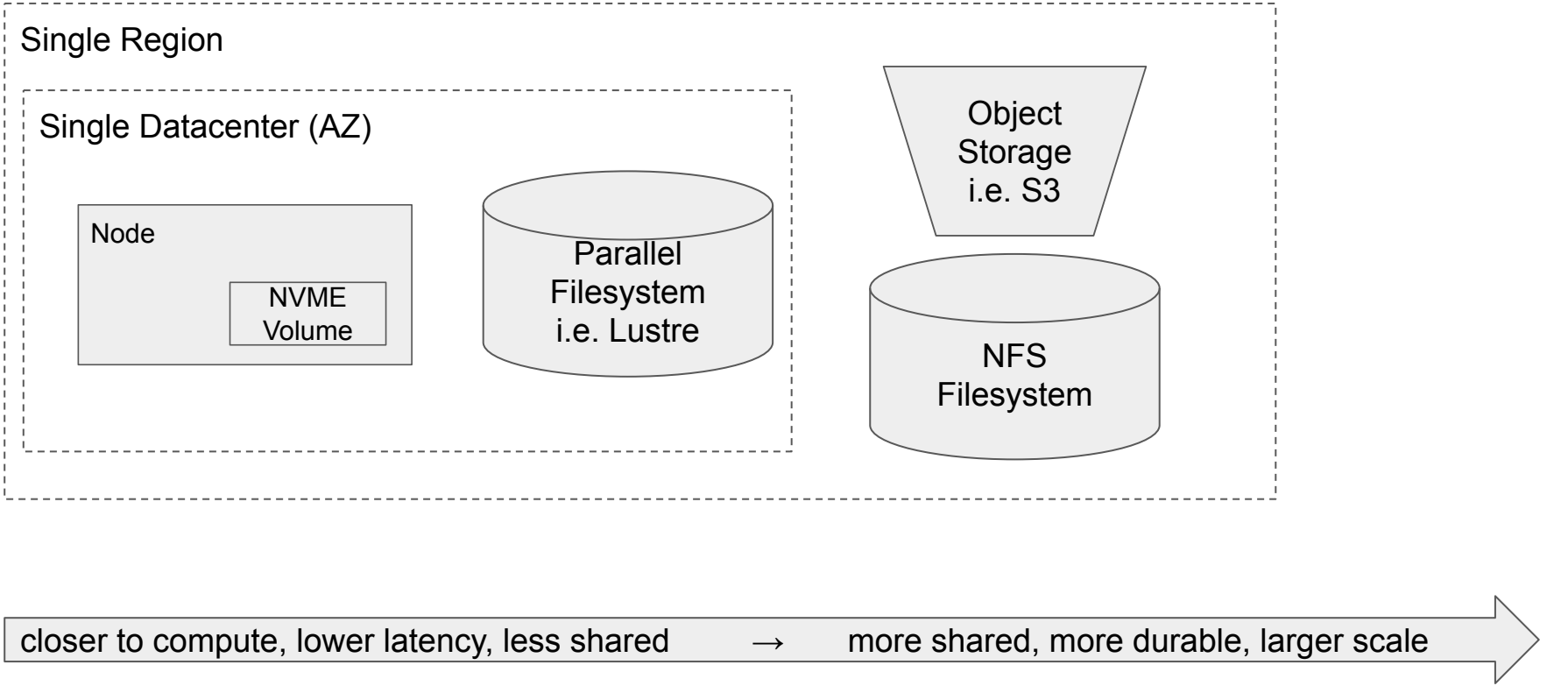
- **Block topology**
  Built for liquid-cooled mega-nodes (Blackwell, Hopper NVL72)
- Skips traditional switches entirely
- Uses atomic, hardware-fenced boundaries (e.g., fixed blocks of 18)



Dragonfly-inspired topology
Designed for next-generation massive scale



AI/HPC Cluster storage architecture



AI/HPC Cluster storage hierarchy

The same training workflow often touches multiple storage tiers, each optimized for different bottlenecks

Local Scratch

Node-local NVMe, SSD, tmpfs

Node-attached storage used by one job on one machine at a time.

Best For

Temporary working data, caches, container builds & high-IOPS access close to CPU/GPU.

Throughput: 14 GB/s

Parallel FS

Lustre, WEKA, VAST

Distributed shared filesystem for cluster-scale POSIX access and high aggregate bandwidth.

Best For

Active multi-node training data, checkpointing, and heavy parallel I/O streams.

Throughput: 3-10GB/s

Shared NFS

ZFS, NetApp ONTAP

Simple shared POSIX filesystem that multiple cluster nodes mount over the network.

Best For

Shared datasets, team directories, and workflows prioritizing ease of access over raw throughput.

Throughput: .5-2.5GB/s

Object Store

S3, GCS, Azure Blob

API-accessed bucket storage providing massive scalability and durability.

Best For

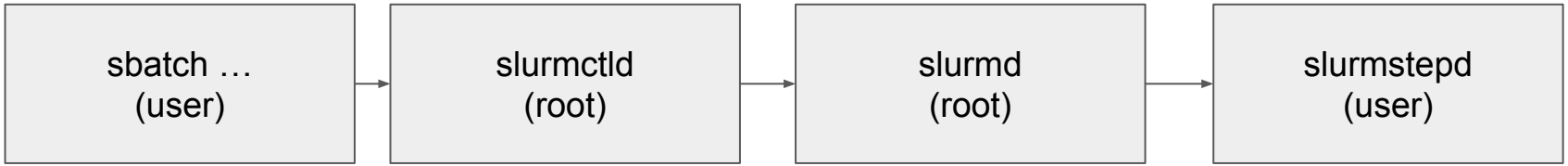
Massive durable datasets, long-term archiving, portability, and cross-cluster sharing.

Throughput: .1-.5 GB/s
(*scales linearly)

Storage speeds are aprox per-node. Source: Google Gemini

Access Control

- User Identities are handled centrally (i.e. LDAP, or Active Directory). Jobs are submitted to slurmctld by the user which calls slurmd (running as root) which then spawns a subprocess running as the user.



- File permissions is standard POSIX permissions, i.e. rwx for user, group, other

```
-  rwx  r-x  r--
|   |    |    └─ Others: Read-only
|   |    └────── Group:  Read & Execute
|   └─────────── Owner:  Read, Write, & Execute
└─────────────── Type:   (-) Regular file, (d) Directory, (l) Symlink
```

- Directories are organized on each shared filesystem (NFS and Parallel FS) by org_id and user

```
/home/org_id/<user>
/scratch/org_id/<user>
```

AI/HPC node software stack

Bottom: fixed by the cloud provider, shared by every job on the cluster ↑ **Top:** chosen by researcher, ships with the job, changes every run

CONTAINER IMAGE (OCI) everything here ships with your job — run by containerd / Docker / Enroot

- **Workload** your job · model + data pipeline · checkpointing · nccl-tests / validation harness
- **Framework & runtime** PyTorch · JAX · Megatron / NeMo · vLLM / TensorRT-LLM · DeepSpeed · Triton
- **Collective comms** NCCL · NCCL network plugin (libfabric / UCX) · NVSHMEM · MPI · UCC
- **CUDA user space** cUDA runtime · cuBLAS · cuDNN · CUTLASS · NVTX — but not the kernel driver

The portable layer, and the only one researchers fully control.

The same image runs on any cloud only if the interfaces below it match: driver to CUDA, and NCCL to its network plugin to the fabric.

Rule of thumb: user space can be containerized, anything that touches hardware cannot.

HOST USER SPACE node-managed daemons, runtimes and libraries

Fabric user space	Container runtime	Scheduler agent	Telemetry & tools
verbs · libfabric · UCX RDMA provider for the NIC	containerd / Docker / Enroot + NVIDIA Container Toolkit	slurmd + Pyxis, or kubelet + device & network plugins	DCGM · node exporter nsys / ncu · health checks

The container toolkit is the seam: at container start it injects the driver's user-space libraries and /dev/nvidia* into the image.

KERNEL SPACE — DRIVERS & MODULES hardware-coupled, fixed by the node image

GPU driver	GPUDirect RDMA	RDMA NIC driver	Storage & block
nvidia.ko / open GPU modules sets the CUDA version ceiling	nvidia-peermem · gdrdrv NIC to GPU memory directly	mlx5 / irdma / CSP NIC RoCE or InfiniBand verbs	Lustre / GPFS / NFS client NVMe · overlayfs

Cannot be containerized. Mismatches here are the usual cause of code that runs fine on one GPU and hangs at 512.

HOST OS Linux node image (Ubuntu / RHEL / Rocky / CSP-optimized) · pinned kernel · systemd · cgroups v2 · hugepages & IOMMU settings

Chosen once per cluster. Everything above inherits its kernel.

COMPUTE SERVICE the provider's unit of delivery — the one genuinely cloud-specific layer

BMaaS — bare metal as a service	VMaaS — VM as a service
you get the node itself: you own the OS image, drivers and firmware policy	hypervisor with GPU passthrough or SR-IOV; provider owns firmware and host

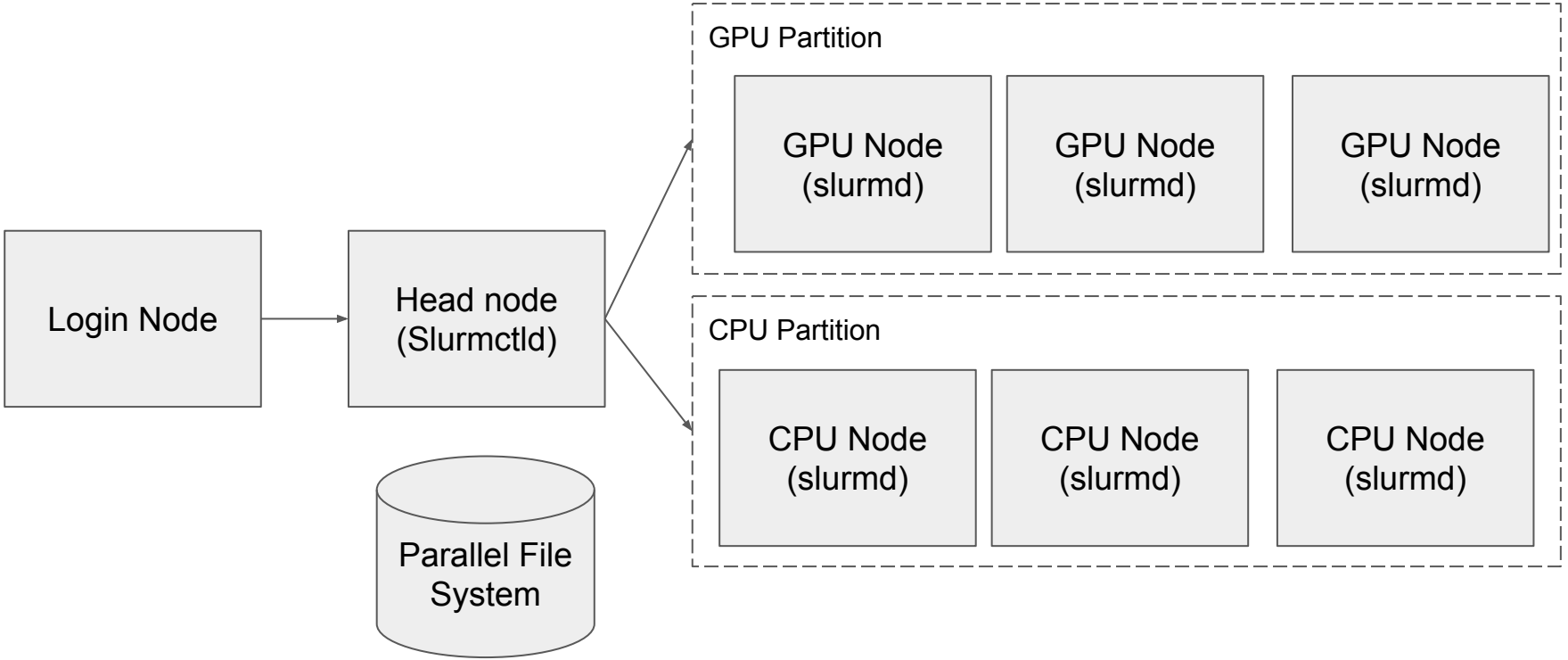
This choice sets where the provider's responsibility stops and yours begins.

8x h100 SXM system used as reference

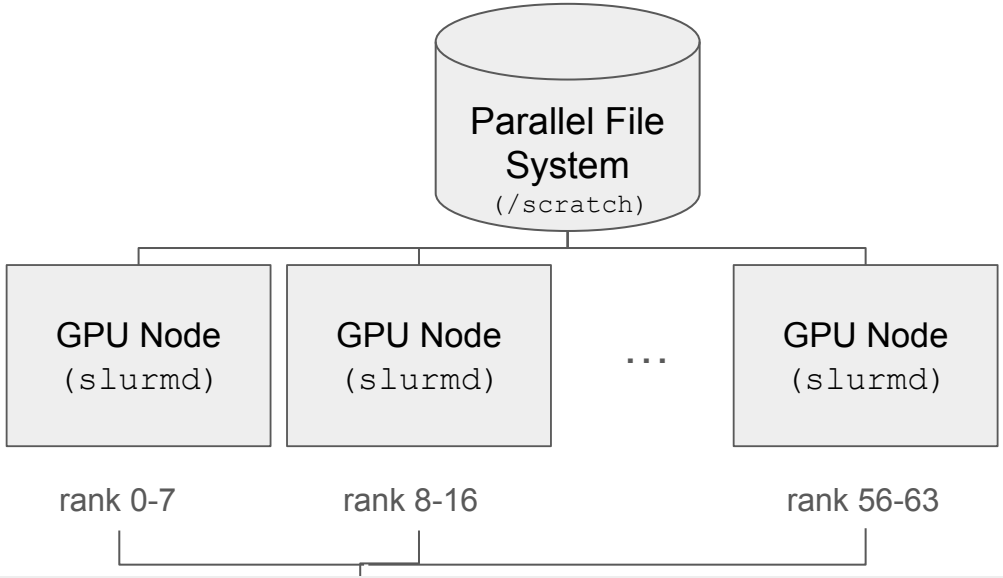
Part B: Cluster Bring Up and Testing

How do we bring GPU clusters to life?

HPC/AI Scheduler Architecture (Slurm)



What a job looks like (Slurm)



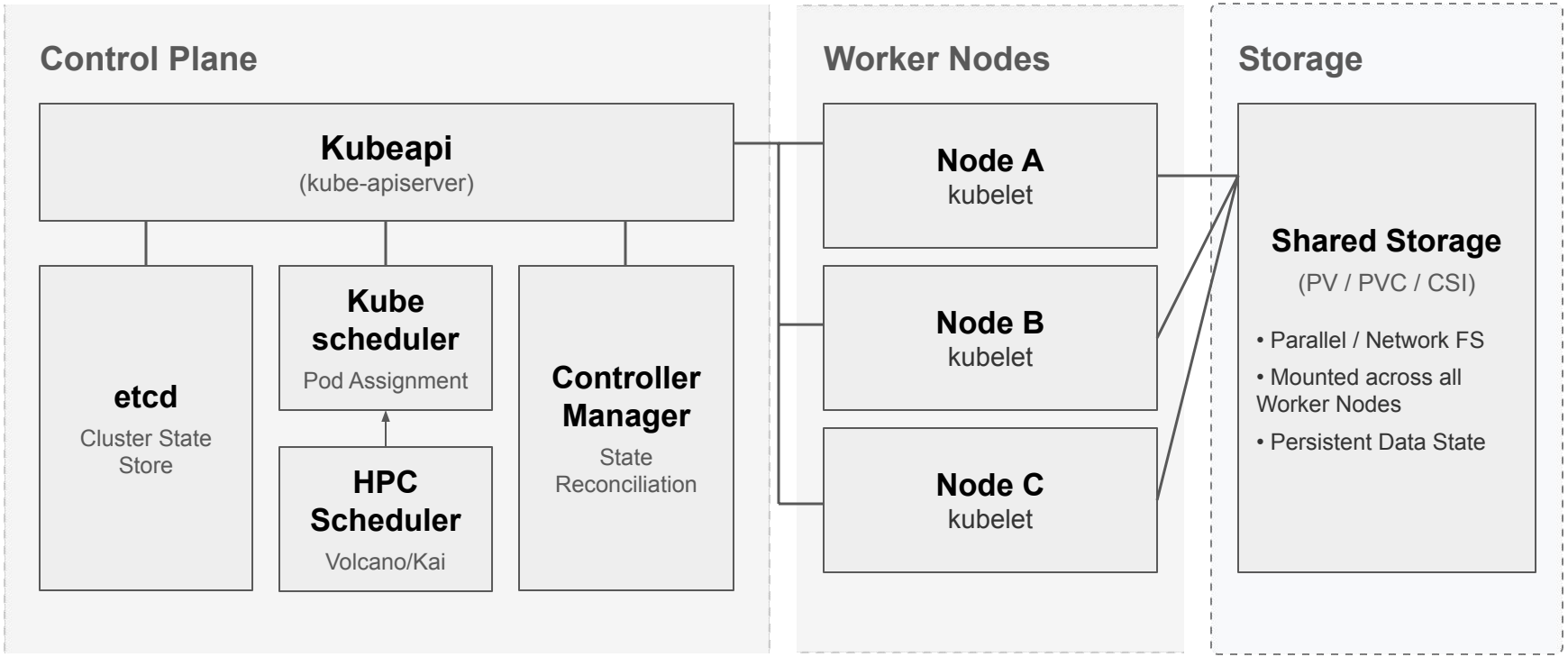
- Each node starts a single container
- Each container spawns 8 processes (known as ranks)
- Torch does the coordination (known as rendezvous)

```
srun -N 8
--ntasks-per-node=1
--gpus-per-node=8
–container-mount /scratch:/scratch --container-image="nvcr.io/nvidia/pytorch:25.11-py3"
torchrun --nproc_per_node=8 --nnodes=8 --rdzv_id=$SLURM_JOB_ID --rdzv_backend=c10d
--rdzv_endpoint="$(scontrol show hostnames $SLURM_JOB_NODELIST | head -n 1):29500" train.py …
```

Cluster Principles (Slurm)

- All nodes are the same, only differences are in user space (application level)
- No users have admin (sudo) privileges
- Scheduler allocates capacity exclusively i.e. single user, single job, single gpu
- Jobs read & write data from shared filesystems. No persistent local storage.
- GPU and CPU nodes are fungible, i.e. easy to kill & respawn
- Workloads are containerized (95% of the time)
- Multi-node jobs are launched as a single process per-gpu i.e. rank 0-7/node
- Networking topology (east/west) is tied to the scheduler so it can allocate in the same NVL Domain (rack scale)
- All nodes are connected via passwordless SSH (i.e. ~/.ssh/authorized_keys)

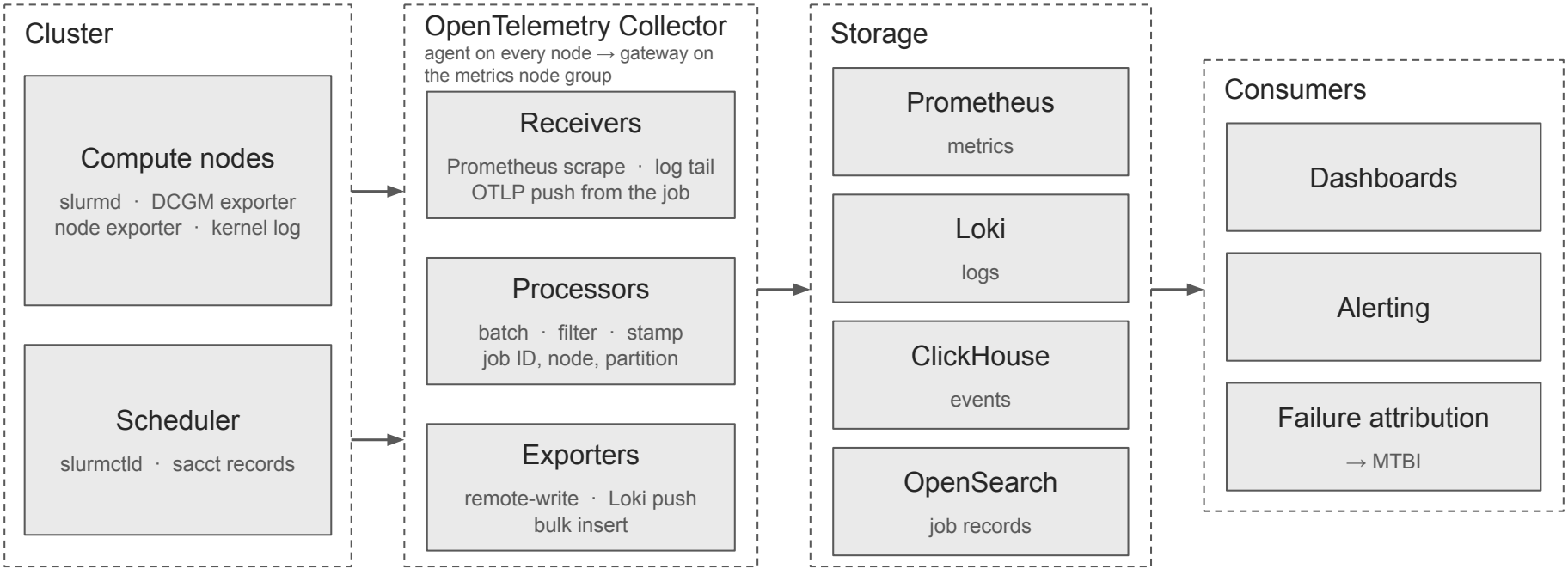
Basic Architecture of a Cluster (Kubernetes)



Cluster Principles (Kubernetes)

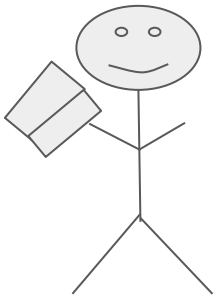
- Kubernetes (k8s) was built as a **distributed system** for container orchestration. It was designed for a web-server like applications where containers (pods) can be scaled up and down.
- For training it's lacking lacks several key features:
  - It lacks "gang scheduling" i.e. all-or-nothing scheduling. This is solved by adding a external scheduler such as Volcano or Kai that informs the kube scheduler.
  - It lacks a straightforward jobs interface for end users, this is added by a 3rd party UI such as Skypilot, Lepton ect.
- There are several advantages:
  - Modern well-supported stack, used by frontier labs at scale
  - Resiliency inherent in the architecture
  - Ideally suited for inference and heterogeneous jobs like RL (CPU+GPU)

AI/HPC Cluster Observability Architecture



Phases of a cluster bring up (Infra Operator):

Hardware validation	Functional Validation	Workload Validation
DGEMM cublasMatMult NCCL nvBandwidth Local storage FIO	System checks Auth Slurm configuration Enroot setup Network storage FIO Smoke tests (NCCL, Sniff, nvBandwidth)	NCCL Parameters Application (Model) Proxies (MLPerf, HPL) Storage Testing Custom testing



Researcher onboarding!

cuBLAS Matrix Multiplication Validation

Test Suite Overview

Numerical Accuracy & Hardware Health

- Detects hardware-level floating point errors, GPU memory corruption, and silent data corruption (SDC) under heavy compute loading.

Peak Compute Throughput (FLOPs)

- Measures dense GEMM performance to establish upper-bound teraflops/petaflops baselines across FP32, FP16, and BF16 precision types.

Stress Testing & Thermal Limits

- Sustains maximum Tensor Core utilization to uncover power throttling, thermal issues, or clock frequency degradation under workload.

Execution & Verification Metrics

EXAMPLE CLI EXECUTION

```
cublasGemmTest -m 8192 -n 8192 -k 8192
-precision=fp16 ...
```

WHAT WE EXAMINE

- **Compute Performance:** Validate achieved TFLOPS against theoretical peak for target precision.

```
# Testing GEMM: M=8192, N=8192, K=8192 | Precision: FP16 Tensor Core
# Matrix A: (8192x8192) Matrix B: (8192x8192) Matrix C: (8192x8192)
# Size(N) Precision Time(ms) GFLOPS TFLOPS ResErrors Status
8192 FP16 3.241 339451 339.45 0 OK
8192 BF16 3.258 337680 337.68 0 OK
8192 FP32 12.894 85302 85.30 0 OK
# Verification Check : 0 errors detected across 1000 iterations
# Peak Compute Achieved : 339.45 TFLOPS (98.2% of theoretical max)
```

NCCL Tests & Fabric Validation

Test Suite Catalog

- **Loopback Tests**
  Validates local GPU-to-NIC path & rail alignment; tests bidirectional saturation under full bi-directional traffic.
- **NCCL AllReduce**
  Exercises Data Parallel (DP) gradient exchange patterns across NVLink and InfiniBand/RoCE fabrics.
- **NCCL AllToAll**
  Stresses MoE (Mixture-of-Experts) token routing across rail-optimized and multi-plane topologies.

Execution & Verification Metrics

EXAMPLE CLI EXECUTION

all_reduce_perf -b 256M -e 16G -f 2 -g 8 …

alltoall_perf -b 256M -e 16G -f 2 -g 8 …

What we examine

- **Bandwidth:** Scale up to 16GB and look at max in-place busbw - compare this number to theoretical max for the fabric (all_reduce)

```
#
#                                                              out-of-place                       in-place
#       size         count      type   redop    root     time   algbw   busbw #wrong     time   algbw   busbw
#wrong
#        (B)    (elements)                               (us)  (GB/s)  (GB/s)            (us)  (GB/s)  (GB/s)
   268435456      67108864     float     sum      -1   1097.0  244.69  428.21      0   1094.7  245.20  429.11
0
   536870912     134217728     float     sum      -1   2115.8  253.74  444.04      0   2121.6  253.05  442.85
0
  1073741824     268435456     float     sum      -1   4033.6  266.20  465.84      0   4033.4  266.21  465.88
0
  2147483648     536870912     float     sum      -1   7960.1  269.78  472.12      0   7949.1  270.15  472.77
0
```

Application Testing (Pre-training Workload)

Test Overview & Objectives

Real-World Pre-Training Stress Test

- Uses dense 8B / 56B / Ultra models to simulate real-world AI researcher workloads on the cluster.

End-to-End Stack Validation

- Stresses compute, NVLink, InfiniBand/RoCE network fabric, container environments, and storage simultaneously.

Catch System-Level Regressions

- Exposes complex multi-node issues (e.g., slow GPUs, filesystem locks, bad images) that isolated microbenchmarks miss.

Execution & Performance Metrics

EXAMPLE WORKLOAD RUN

```
torchrun --nproc_per_node=8 pretrain.py ...
```

KEY EVALUATION CRITERIA

- **TFLOP/s / Step Time:** Compared against baseline targets set by statistical QA analysis.
- **Scaling Efficiency:** Evaluates multi-node throughput from single rack up to full cluster scale.
- **Straggler Detection:** Identifies single slow nodes or GPUs stalling large collective operations.

MLPerf as a cluster validation tool

ML Commons

- **Open source**
  Reference implementations from MLCommons, on GitHub. Clone and run.
- **Measures time-to-train**
  Fixed model, dataset and target quality — so compute, fabric, storage and framework are all on the hook at once.
- **An external reference point**
  v6.0 published 95 systems from 24 organizations. Compare your cluster against the field, not just against yourself.
- **Published numbers are a ceiling**
  Closed-division submissions are tuned for weeks by vendor performance engineers.

https://github.com/mlcommons/training

NVIDIA Performance on MLPerf 6.0's AI Benchmarks: Multi Node, Closed Division

Network	Time to Train (mins)	MLPerf Quality Target	GPU	Server	MLPerf ID	Precision	Dataset	Framework	GPU Version
Llama 3.1 405B	58.3	5.6 log perplexity	512x GB300	Theia-cmh (8x NVIDIA GB300 NVL72)	6.0-0013	Mixed	c4/en/3.0.1	NVIDIA NeMo	NVIDIA Blackwell Ultra GPU (GB300)
Llama 3.1 405B	18.5	5.6 log perplexity	2,048x GB300	Theia-cmh (32x NVIDIA GB300 NVL72)	6.0-0012	Mixed	c4/en/3.0.1	NVIDIA NeMo	NVIDIA Blackwell Ultra GPU (GB300)
Llama 3.1 405B	9.8	5.6 log perplexity	4,096x GB300	CoreWeave_GB300_1024x4_nccl2297	6.0-0004	Mixed	c4/en/3.0.1	PyTorch	NVIDIA Blackwell Ultra GPU (GB300)
Llama 3.1 405B	18.8	5.6 log perplexity	2,560x GB200	Tyche-hsg (40x NVIDIA GB200 NVL72)	6.0-0019	Mixed	c4/en/3.0.1	NVIDIA NeMo	NVIDIA Blackwell GPU (GB200)
Llama 3.1 405B	10.0	5.6 log perplexity	5,120x GB200	Tyche-hsg (80x NVIDIA GB200 NVL72)	6.0-0021	Mixed	c4/en/3.0.1	NVIDIA NeMo	NVIDIA Blackwell GPU (GB200)
Llama 3.1 405B	7.1	5.6 log perplexity	8,192x GB200	Azure GB200 (128x NVIDIA GB200 NVL72)	6.0-0001	Mixed	c4/en/3.0.1	NVIDIA NeMo	NVIDIA Blackwell GPU (GB200)

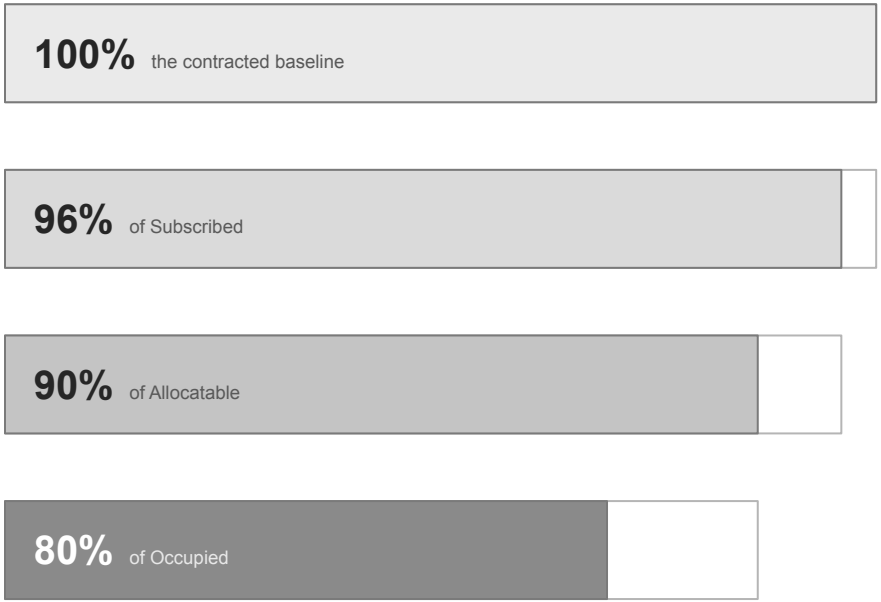
MLPerf® is a trademark of MLCommons®. Figures from MLPerf Training v6.0, June 2026.

Source:https://developer.nvidia.com/deep-learning-performance-training-inference/training

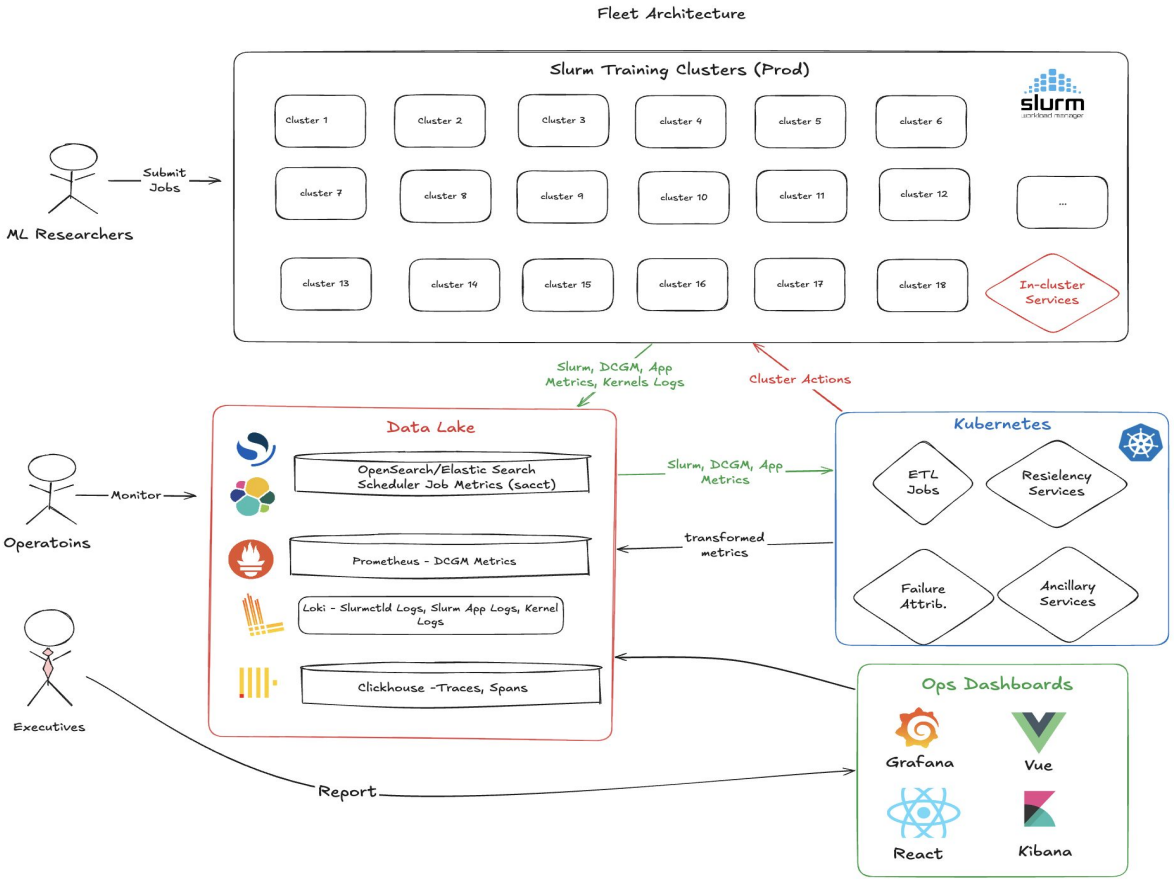
Part C: Measuring Success as an Infra Operator

Once a cluster is up, how do we keep it humming along?

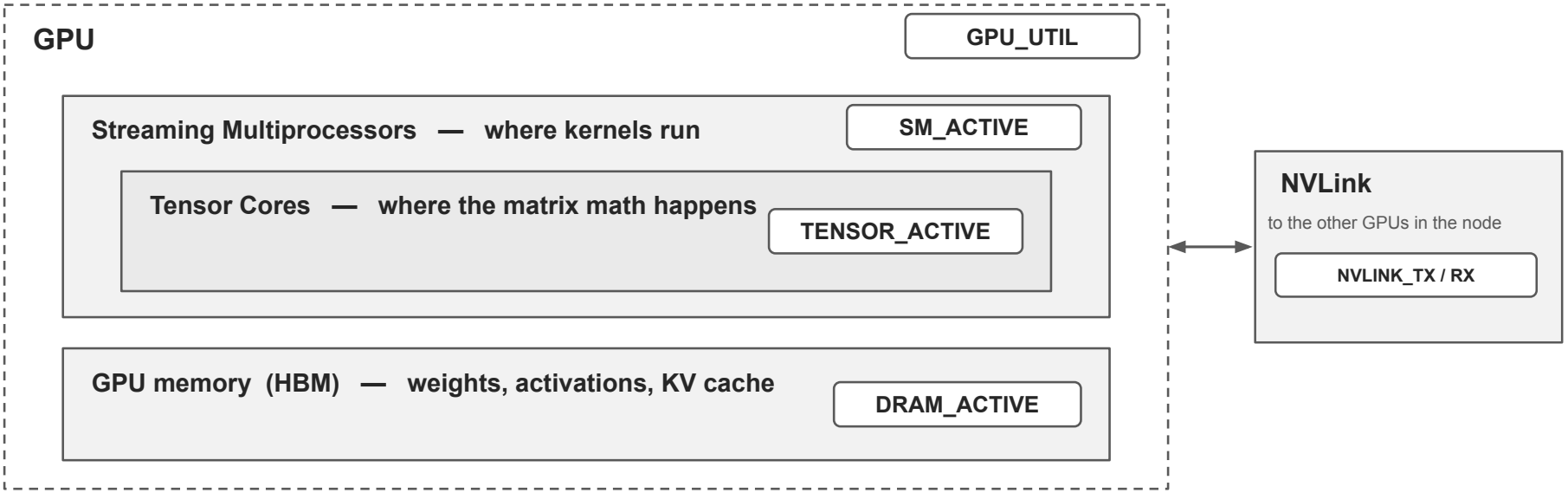
Measuring success: Cluster Efficiency Cascade



Fleet Architecture - Workload and Cluster Observability (Slurm)



GPU Monitoring with DCGM (part 1)



Next: what each of these counters actually measures — and why the one most people quote is the one on the outside box.

GPU Monitoring with DCGM (part 2)

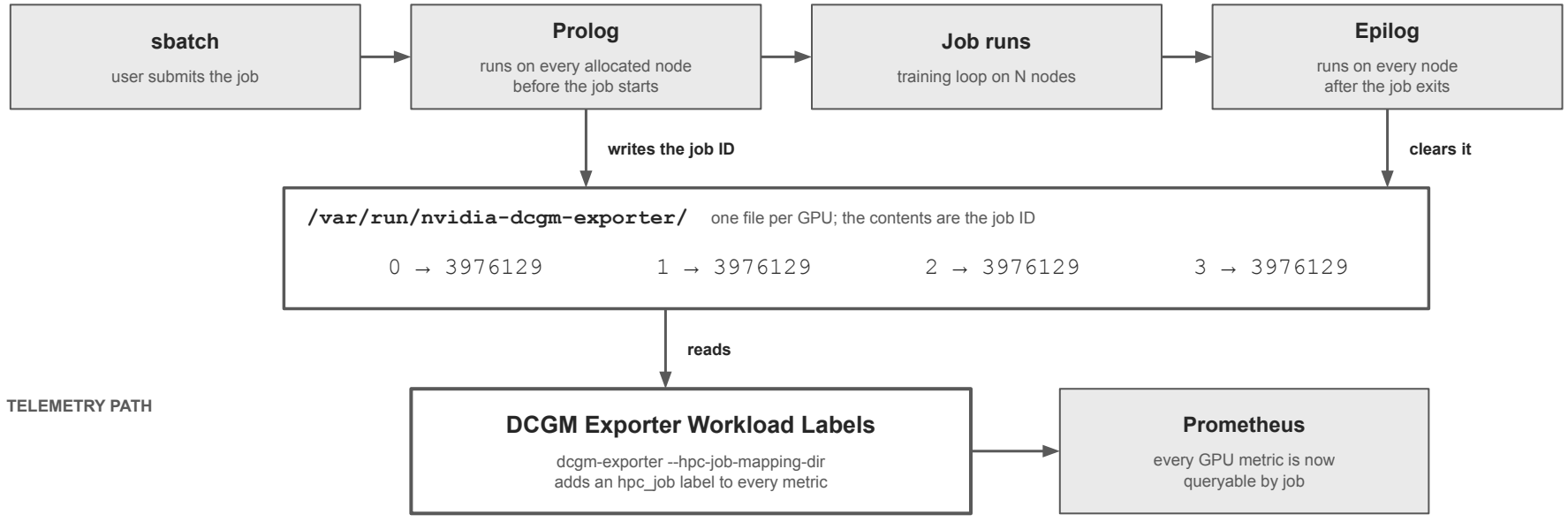
What the counter actually measures	**Matmul-heavy step** a healthy training iteration	**Memory-bound kernels** small batch, elementwise ops	**All-reduce in flight** gradients crossing NVLink	**Stalled on a straggler** NCCL kernel spin-waiting
GPU_UTIL `DCGM_FI_DEV_GPU_UTIL`	**100%**	**100%**	**100%**	**100%**
SM_ACTIVE `DCGM_FI_PROF_SM_ACTIVE`	**0.94**	**0.88**	**0.22**	**0.18**
TENSOR_ACTIVE `DCGM_FI_PROF_PIPE_TENSOR_ACTIVE`	**0.61**	**0.03**	**0.00**	**0.00**
DRAM_ACTIVE `DCGM_FI_PROF_DRAM_ACTIVE`	**0.42**	**0.87**	**0.08**	**0.02**
NVLINK_TX_BYTES `DCGM_FI_PROF_NVLINK_TX_BYTES`	**~0**	**~0**	**310 GB/s**	**~0**

PROF_* fields are ratios from 0 to 1; GPU_UTIL is a percentage. GR_ENGINE_ACTIVE behaves like a finer-grained GPU_UTIL — still blind to what the SMs are doing. Values are illustrative.

Attributing DCGM metrics to Slurm Jobs (Workload Labels)

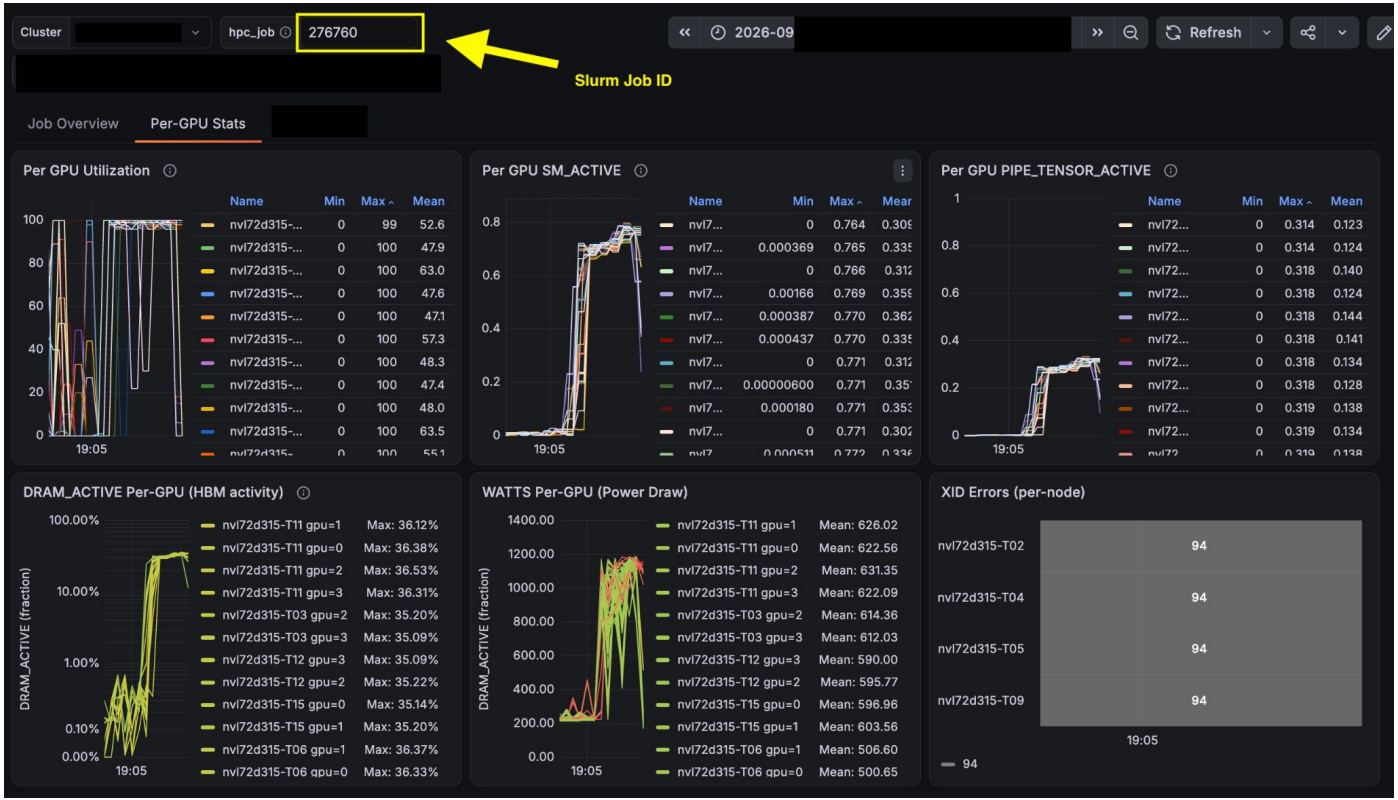
DCGM Exporter Workload Labels — putting a Slurm job ID on every GPU metric

Prolog writes the job ID to disk; the exporter reads it. **That one file turns per-GPU telemetry into per-job telemetry.**




```
DCGM_FI_PROF_PIPE_TENSOR_ACTIVE{gpu="3", Hostname="node412", hpc_job="3976129"}    0.61
```

DCGM Workload Labels



Measuring Success: Failure Attribution

Key requirement: can the platform answer why a job failed?

A job exits non-zero. Was it your code, the framework, the network, or a GPU? **Until the platform can answer that, nothing above it is measurable.**

EXACTLY ONE OUTCOME PER JOB

Fail: user code	**Fail: framework**	**Fail: infra software**	**Fail: infra hardware**	**Unknown**

Unknown is the number to watch. It is the fraction of your cluster you cannot reason about at all.

From evidence to attribution

The user sees one line. The classifier sees four streams — **and has to find the first mover.**

WHAT THE USER SEES

```
srun: error: node412: task 24: Exited with exit code 1
```

WHAT THE CLASSIFIER SEES

T+0s **Job logs** stdout / stderr

```
NCCL WARN Timeout, ring 0, 511 ranks
```

T+2s **System logs** journalctl

```
slurmd: JOB 3976129 CANCELLED
```

T−40s **Kernel log** dmesg

```
NVRM: Xid (PCI:0000:8a:00): 45, pid=1553724
```

T+5s **Scheduler records** sacct

```
3976129  FAILED  1:0  node[412-419]
```

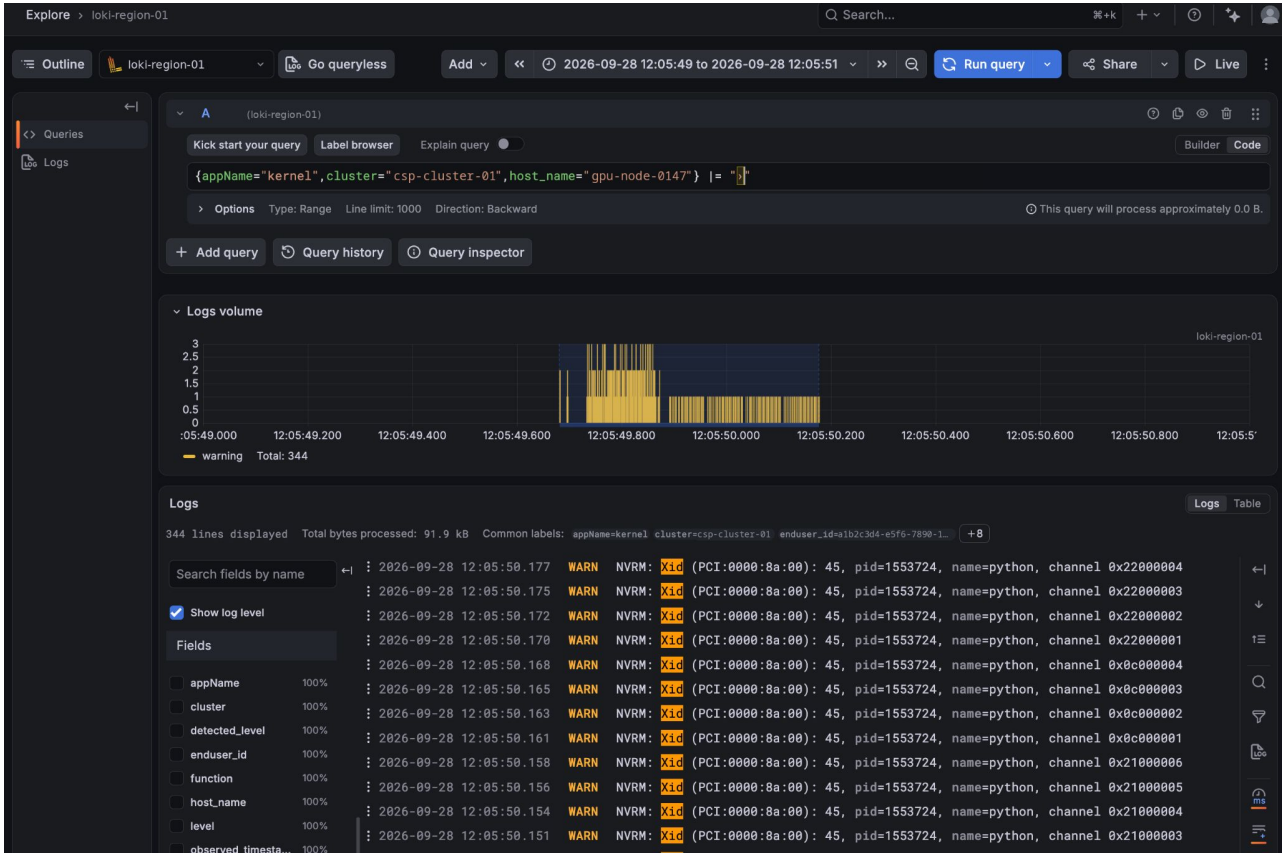
→ **Rules engine**

Orders the evidence in time and picks the first mover.

→ **Fail: infra hardware**

The Xid came 40 seconds before anything else.

Kernel Logs in Loki (example)



From Failure to MTBI

Mean time between interruptions — the cluster's headline reliability number

MTBF is component-centric: how long until a part breaks. ↔ MTBI is process-centric: how long until a workload is interrupted.

$$\textbf{HW MTBI} = \frac{\text{Occupied GPU-hours}}{\text{Unique infra-HW failures}} \div 1{,}000$$

The denominator counts distinct failed jobs attributed to hardware. Not nodes. Not GPUs. Jobs.

145,000 occupied GPU-hours in a week · 12 job failures attributed to hardware

MTBI = 12.1k GPU-hours

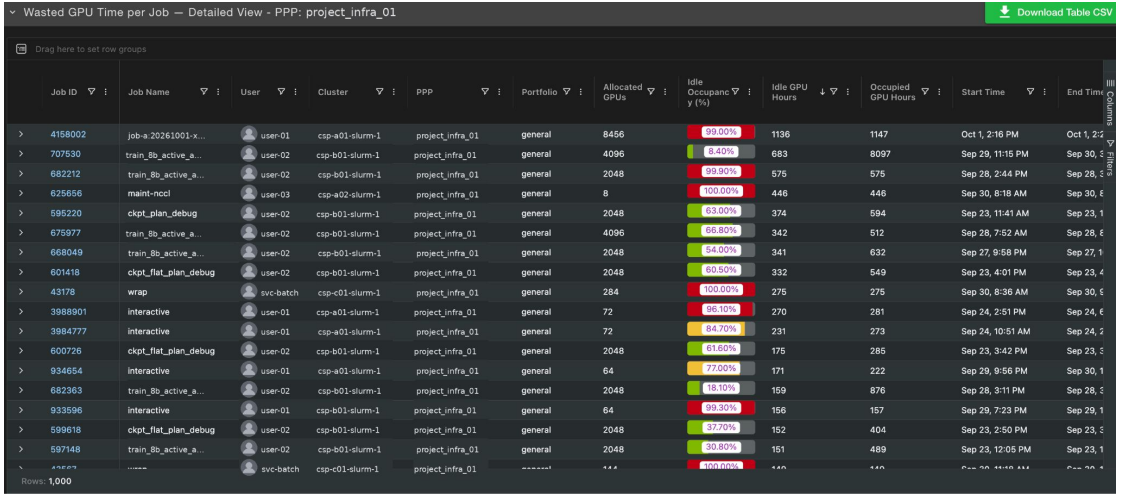
A 512-GPU run burns 512 GPU-hours an hour — so expect an interruption about every day.

Unknown quietly flatters you. A hardware failure never attributed to hardware never enters the denominator — but its GPU-hours still count in the numerator.

Idle Waste - a Researcher Productivity Metric

Idle Waste Definition:

*Time during a running job when the GPU's compute utilization (`DCGM_FI_DEV_GPU_UTIL`) was exactly **0%** for 5 consecutive minutes — a GPU was allocated to a job but doing no work. Aggregated per-job from 5-minute DCGM samples, then summed across all GPUs assigned to the job.*



ML Productivity Goodput

Originating from Google and Harvard as a metric to measure real progress of training jobs over time [1]

Scheduling Goodput x Runtime Goodput x Program Goodput

$$Scheduling\ Goodput = \frac{t_{running}}{t_{running} + t_{waiting}}$$

Capacity & Scheduling

- Availability of the requested capacity
- Bad: wait in queue, good: run on-demand
- Constraint due to the size of the job or imposed limits (e.g. max job size)

Job Scheduler

$$Runtime\ Goodput = 1 - \frac{N \times (t_{lost} + t_{resume})}{T} - \frac{t_{save}}{\tau} - \sum_{i=1}^{M} d_i$$

Reliability

- Time spent iterating (fwd, bwd passes)
- Negative impact due to the warmup, detection, shutdown and insurance tax
  - non-productive time (e.g. data iterations, graph computation, nccl initialization….

Application & Support

$$Program\ Goodput = \frac{FLOPS_{Measured}}{FLOPS_{Theoretical}} = \text{MFU}$$

Performance

- Measures training efficiency and counts the productive flops (bw, fw passes) and is hardware agnostic
- Impacted by optimization or bottlenecks (memory, communication)

Application

[1] Machine Learning Fleet Efficiency: Analyzing and Optimizing Large-Scale Google TPU Systems with ML Productivity Goodput - https://arxiv.org/abs/2502.06982

Scheduling Goodput

Core formula

$$Scheduling\ Goodput = \frac{t_{running}}{t_{running} + t_{waiting}}$$

Time the workload has been running on the system and making use of resources

Total of time during which the workload has been running and waiting to run (i.e. waiting in queue).

Example: 128 nodes job on OCI-HSG t_wait ~26 min, t_running ~18 min

```
paquilanti@oci-hsg-cs-001-login-02:~$ sacct -j 1388651 --
format=JobID,Submit,Start,End,Elapsed,State
JobID               Submit               Start                End    Elapsed      State
------------ ------------------- ------------------- ------------------- ---------- ----------
1388651      2026-01-09T22:09:27 2026-01-09T22:35:30 2026-01-09T22:53:58   00:18:28  COMPLETED
```

$$\frac{1108}{1108+1563} \approx 41.48\%$$

Influenced by

- *Workload requirements*: type of resources, number of resources, time & placement constraints
- *Resources availability*: types of resources and general availability, queues & reservations, time constraints, priority.

Helps to measure

- Scheduling policies, constraints (blocks)
- Capacity needs (type, size) for different job types (ex: CPU nodes for RL - ADLR)

Program Goodput

Assess the workload efficiency in leveraging the hardware resources toward the objective

$$Program\ Goodput = \frac{FLOPs_{Measured}}{FLOPs_{Theoretical}} = \text{MFU}$$

Standard *

- **Model FLOP Utilization (MFU)**
  - Measures productive flops, the forward (fwd) and backward (bwd) passes.

Alternatives

- Overall FLOP Utilization (OFU): Achieved FLOP/s to peak FLOP/s measured over GPU occupied time (Tensor Cores).
- **Hardware FLOP Utilization (HFU):** Measures how much the hardware is busy, Fwd, bwd & recomputation. Minimal with selective recomputation.
- **Tokens / day:** Business metric, does not consider the hardware & implementation.

OFU

Tensor-pipe utilization over allocated time

Definition: fraction of peak tensor throughput achieved, averaged over all allocated GPU-time (occupied time)

OFU = mean_over_30s_buckets(TENSOR_ACTIVE × min(SM_CLOCK, peak_clock) / peak_clock) × 100

Inputs: TENSOR_ACTIVE = DCGM_PIPE_TENSOR_ACTIVE; SM_CLOCK live; peak_clock = per-model SM boost clock (static table). Denominator time = gpu time occupied by the job

Interpretation (GPU-hours-weighted):

- **40%+** well-tuned run
- **20%** cluster wide goal (workloads are diverse)
- **9%** a low average
- **<5%** low utilization - allocated but little tensor compute

Caveats: Depends on per-model idle-power baseline (measured as p50 at sm<0.01) and rated power (empirical busy-p99, not product or CSP spec)

Power Utilization

Power draw vs usable envelop

Definition: per-GPU power normalized between idle floor and rated power. Captures ALL silicon (tensor, CUDA, memory, NVLink) — complements the tensor-only OFU.

$$\text{Power Util} = \text{clamp}\left((\text{POWER_USAGE} - \text{idle_baseline}) / (\text{rated_power} - \text{idle_baseline}), 0, 1 \right) \times 100$$

	Power LOW	Power HIGH
OFU LOW	Idle - allocated, ~no work	Busy on non-tensor work *(memory-bound, NCCL, dataloading)*
OFU HIGH	Small model / clock-throttled	Productive tensor work

Caveats: Depends on per-model idle-power baseline (measured as p50 at sm<0.01) and rated power (empirical busy-p99, not product or CSP spec)

Memory Utilization

Power draw vs usable envelop

Definition: 95th-percentile fraction of HBM in use across the run (p95, not max — robust to transient spikes).

Mem p95 = p95_over_buckets(FB_USED / (FB_USED + FB_FREE) × 100)

Interpretation (GPU-hours-weighted):

- **High mem, low OFU** memory bound: HBM full, compute starved.
- **Low mem** workload under-fills the GPU - placement / right-sizing candidate (may fit smaller/cheaper hardware)
- **High mem, high OFU** memory-bound: HBM full, compute starved

Caveats: FB_USED reflects allocator reservation. FB_TOTAL is not exported fleet wide - denominator is USED + FREE

Resources:

- HuggingFace Ultrascale Playbook
- Slurm SchedMD Overview
- Intro to Lustre Wiki
- ML Commons Training GitHub
- Google Cloud Goodput as a Messure of Productivtiy
- Instant GPU Efficiency Visibility at Fleet Scale