FLARE: Anomaly Diagnostics for Divergent LLM Training in GPU Clusters of Thousand-Plus Scale

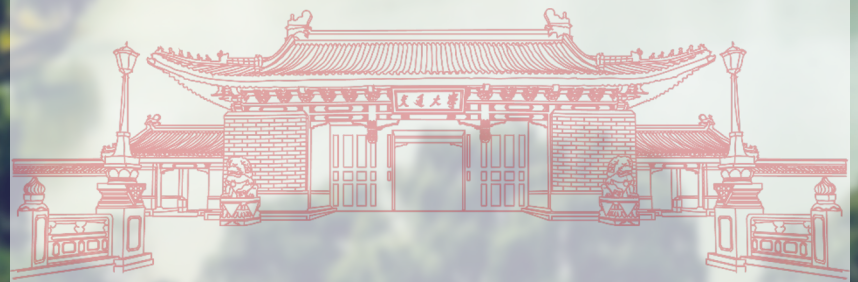
Weihao Cui, Ji Zhang, Han Zhao, Chao Liu, Jian Sha, **Bo Sang**

Bingsheng He, Minyi Guo, Quan Chen

Tuesday, March 24, 2026

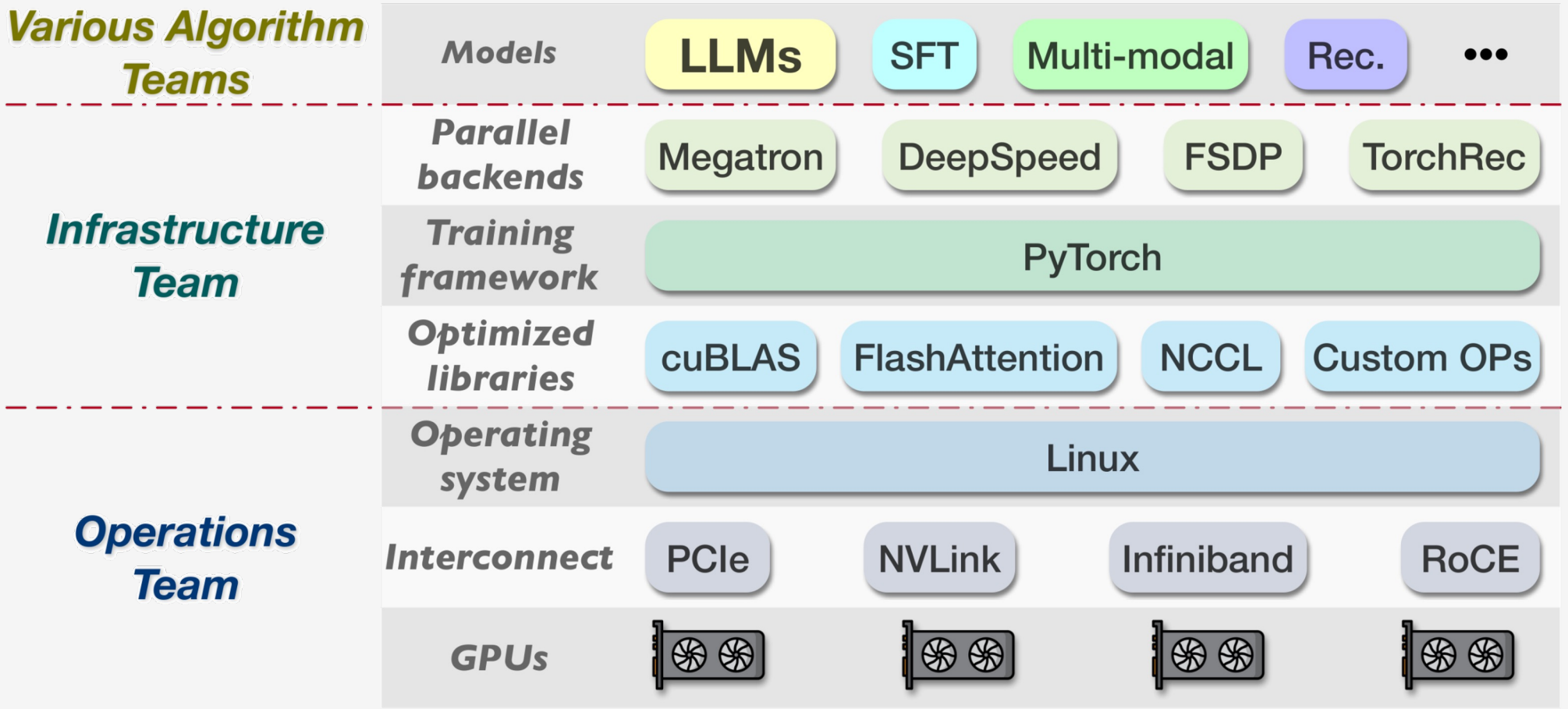
01

Motivation



LLM training at thousand-GPU scale is a multi-team system

- Algorithm, infrastructure, operations teams share the stack
  — **anomaly can cross all three.**



Regressions are the unsolved anomaly class

	Errors	Fail-slows	Regressions
Symptom	Hang or crash	Sudden throughput drop	**Persistent, subtle slowdown**
Owner team	Operations	Operations	**Algorithm + Infrastructure**
3-month count	127 / 3047 jobs	57 / 3047 jobs	**78 / 3047 jobs**

3 months · 6,000 GPUs

- Errors and fail-slows are acute and well-studied — **regressions are persistent and unsolved.**

No existing tool spans the full stack and regression

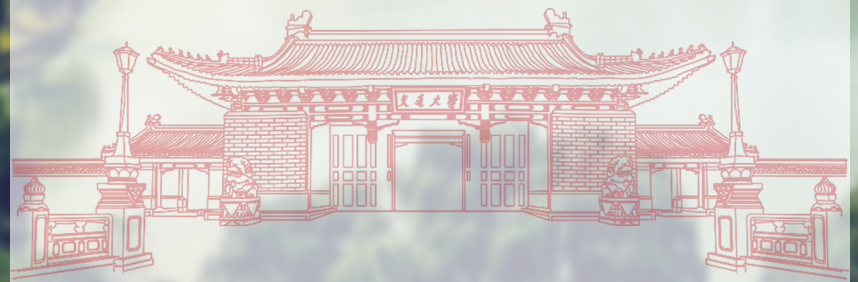
上海交通大學 SHANGHAI JIAO TONG UNIVERSITY

MegaScale	C4D	Greyhound	Holmes	FLARE
Tracing scope **Full stack, but intrusive**	*Tracing scope* **Comm. traffic only**	*Tracing scope* **Comm. only**	*Tracing scope* **Fail-slow localization**	*Tracing scope* **Full-stack, non-intrusive**
Extensibility / diagnosis **Megtron only**	*Extensibility / diagnosis* **Backend-agnostic**	*Extensibility / diagnosis* **Backend-agnostic**	*Extensibility / diagnosis* **Not full stack**	*Extensibility / diagnosis* **Regression-aware**

- FLARE combines **full-stack tracing, backend extensibility, and automated regression diagnosis.**

02

Challenges



Full-stack tracing and backend extensibility conflict

Full-stack coverage

Collect runtime data across Python, C++, CUDA, NCCL

Tends to require intrusive, per-backend hooks

conflict

Backend extensibility

Support Megatron, FSDP, DeepSpeed, TorchRec, …

Demands minimal changes to each backend

Example: MegaScale

Low-overhead tracing via codebase patching

- ✓ **Megatron** — *patched, traced*
- ✗ **FSDP** — *needs another patch*
- ✗ **DeepSpeed** — *needs another patch*
- ✗ **TorchRec** — *needs another patch*

Every new backend ⟶ another codebase patch

Regressions resist both detection and attribution

1

Detection: throughput alone misses regressions.

- Regressions are persistent, subtle slowdowns — no acute drop to flag.
- Algorithm teams assume jobs run normally; throughput monitors stay quiet.

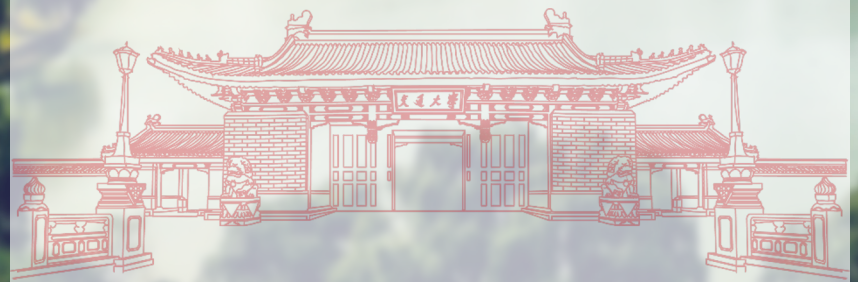
2

Attribution: linking a regression to a code change is non-obvious.

- Same code can be benign or harmful — e.g., Python GC depends on where it fires.
- Exsiting metrics fail to catch the regression well.

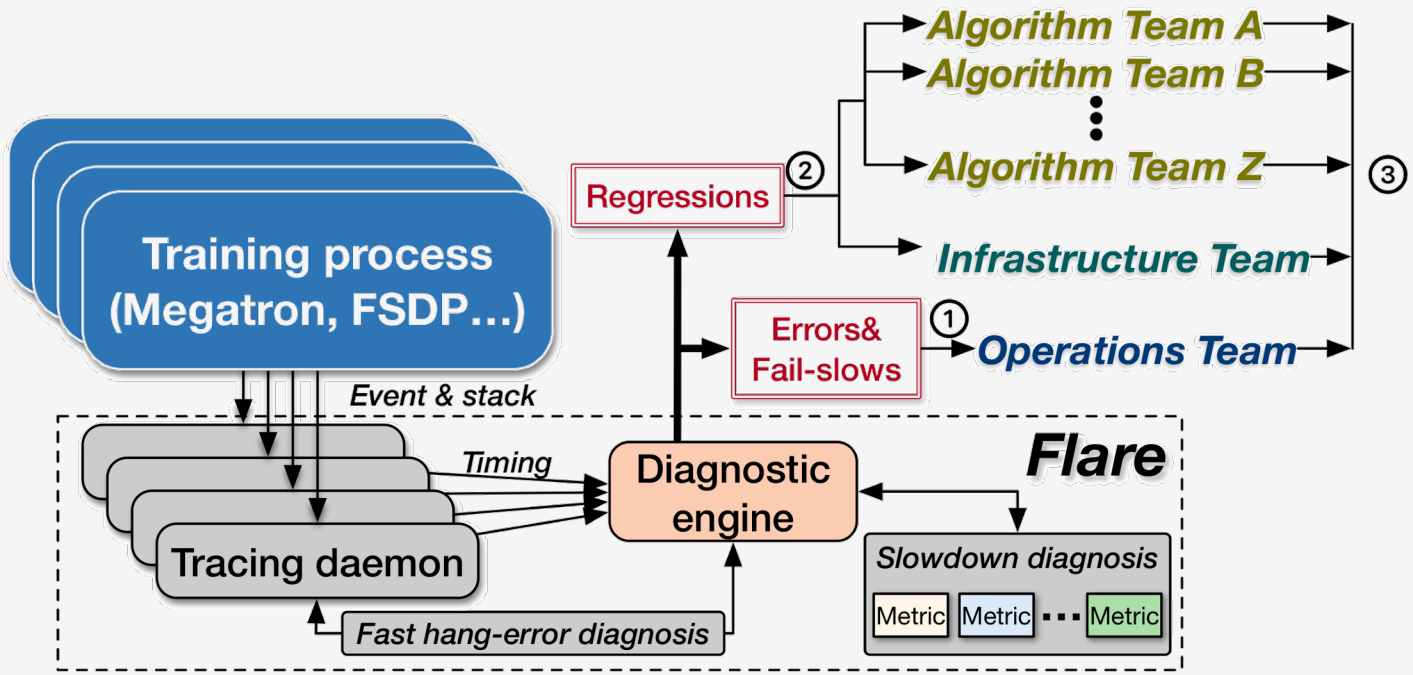
03

Design of Flare



FLARE

Lightweight tracing + automated triage across the training stack — **cross-team collaboration becomes the exception, not the default.**



Tracing daemon

Per-process, backend-extensible, full-stack runtime data

Intercepts Python APIs + C++ kernels; records timings, call stacks, input layouts

Diagnostic engine

Fast hang-error diagnosis + aggregated metrics for slowdowns

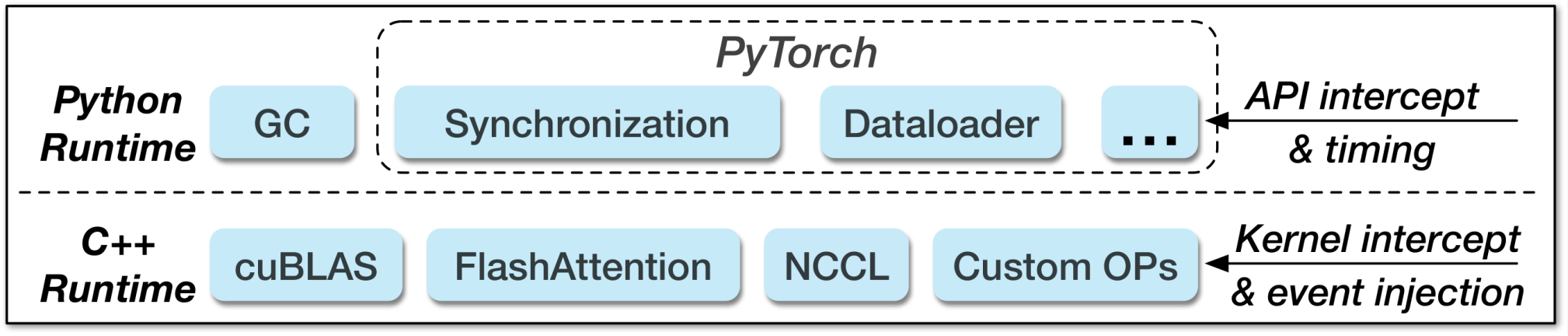
Routes anomalies to the owning team with narrowed root causes

Auto-routing

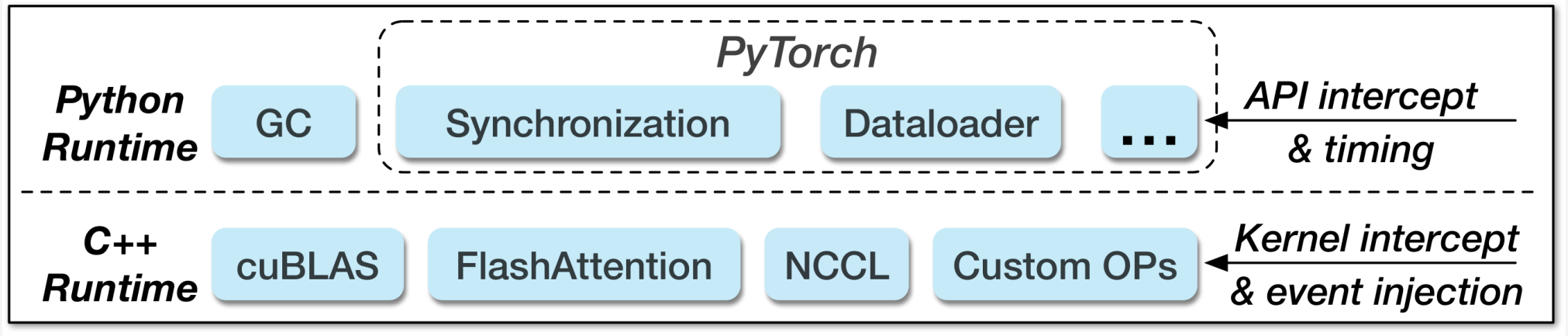
Errors & fail-slows → Operations

Regressions → Algorithm / Infrastructure

Plug-and-play instrumentation: no backend patching



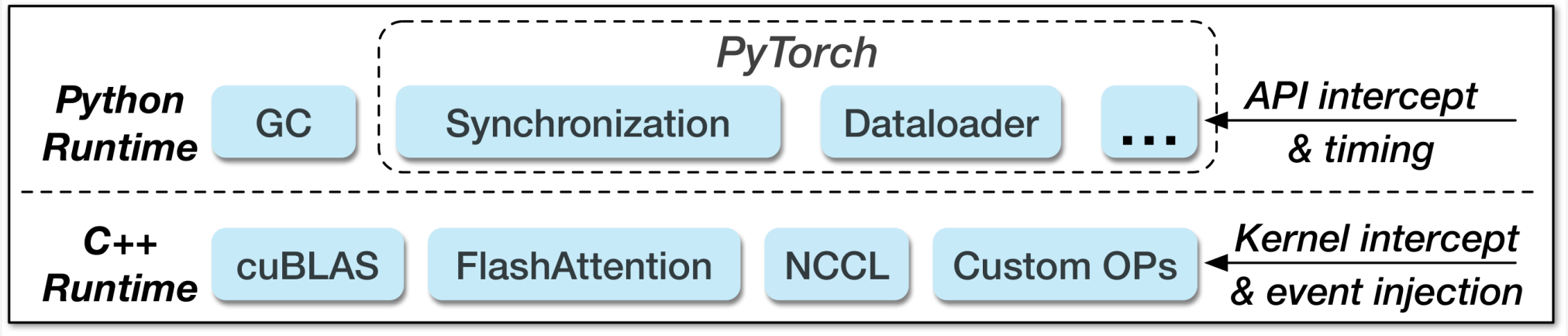
Plug-and-play instrumentation: no backend patching



Python APIs — CPython PyEval_SetProfile on bytecode

- GC, dataloader, GPU sync — no backend source change
- New APIs added by env var: TRACED_PYTHON_API=...

Plug-and-play instrumentation: no backend patching



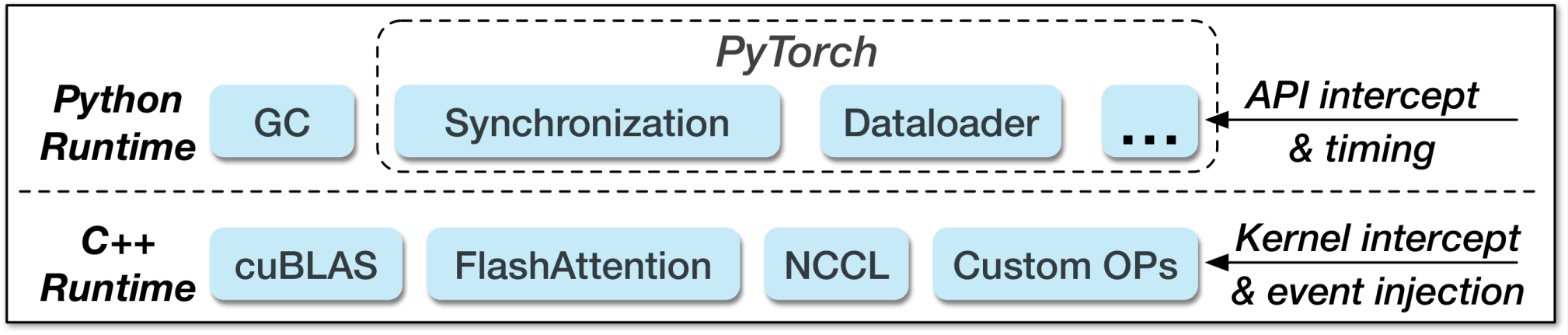
Python APIs — CPython PyEval_SetProfile on bytecode

- GC, dataloader, GPU sync — no backend source change
- New APIs added by env var: TRACED_PYTHON_API=...

C++ kernels — LD_PRELOAD function hooks

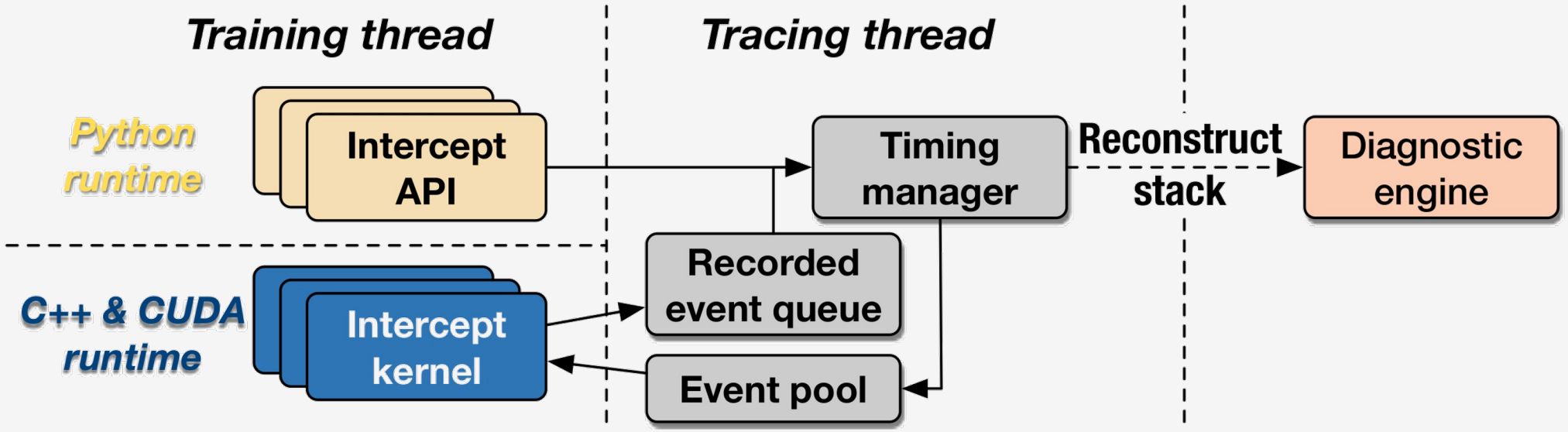
- Matmul + collective kernels dominate thousand-GPU training
- Registered once via C++ API; shared across backends

Plug-and-play instrumentation: no backend patching



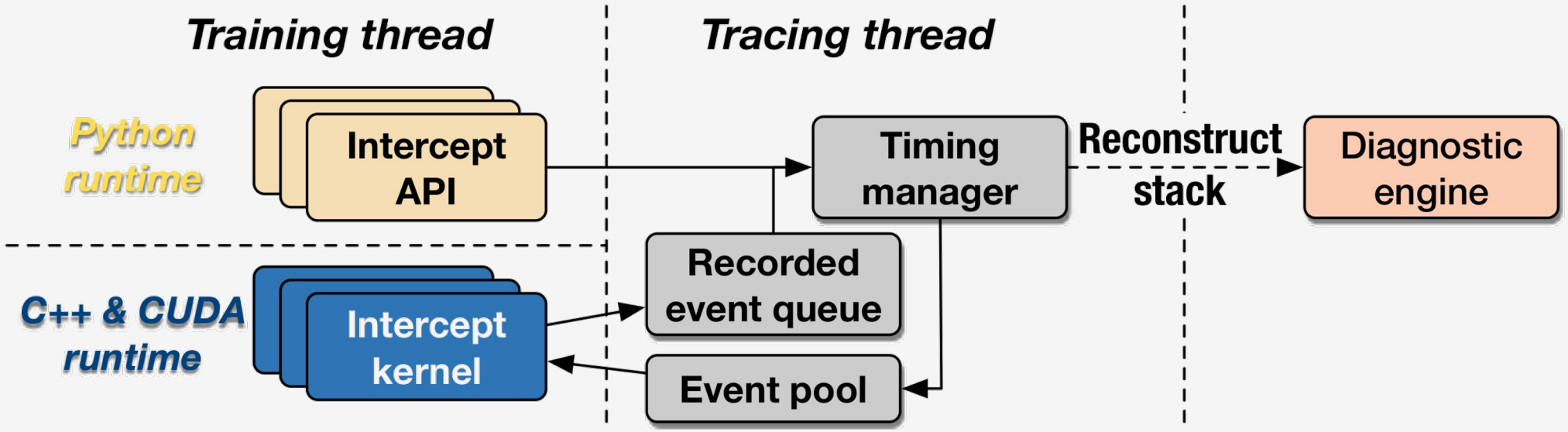
✓ **Zero backend patching — one tracer works across Megatron, FSDP, DeepSpeed, TorchRec.**

Async CUDA-event timing with stack reconstruction



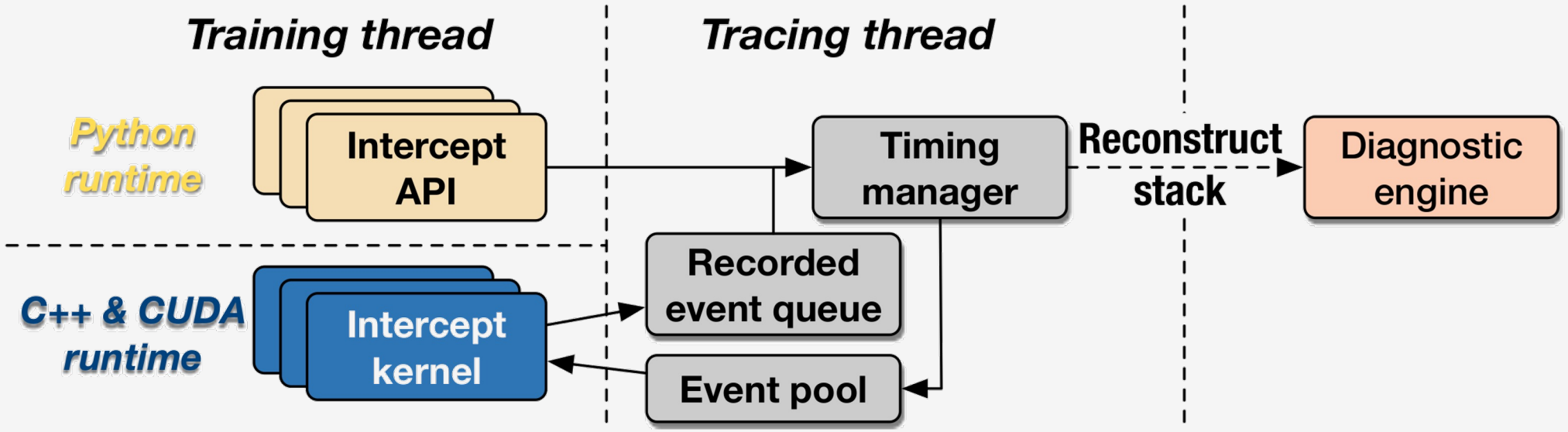
- Python APIs: timestamp start / end inline — synchronous, direct.

Async CUDA-event timing with stack reconstruction



- Python APIs: timestamp start / end inline — synchronous, direct.
- GPU kernels: CUDA events enqueued at interception; status polled by background thread — zero interference with the training thread.

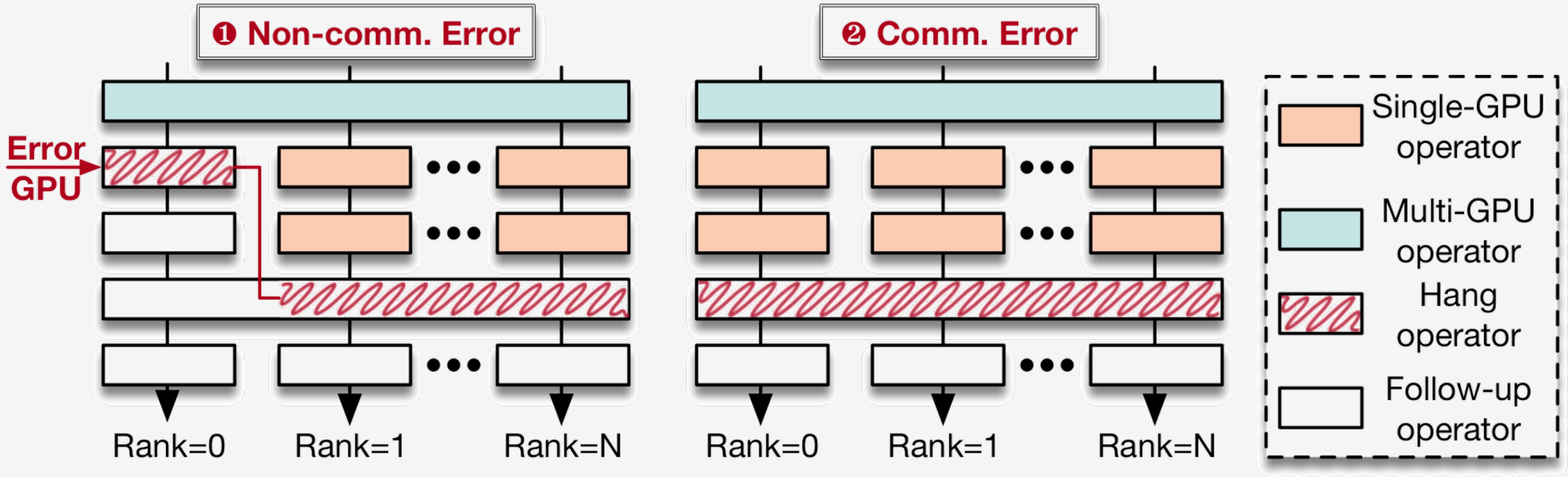
Async CUDA-event timing with stack reconstruction



- Python APIs: timestamp start / end inline — synchronous, direct.
- GPU kernels: CUDA events enqueued at interception; status polled by background thread — zero interference with the training thread.
- Call stack reconstructed post-hoc from Python ↔ C++ timestamp overlap — preserves causal links for diagnosis.

Hang-error diagnosis

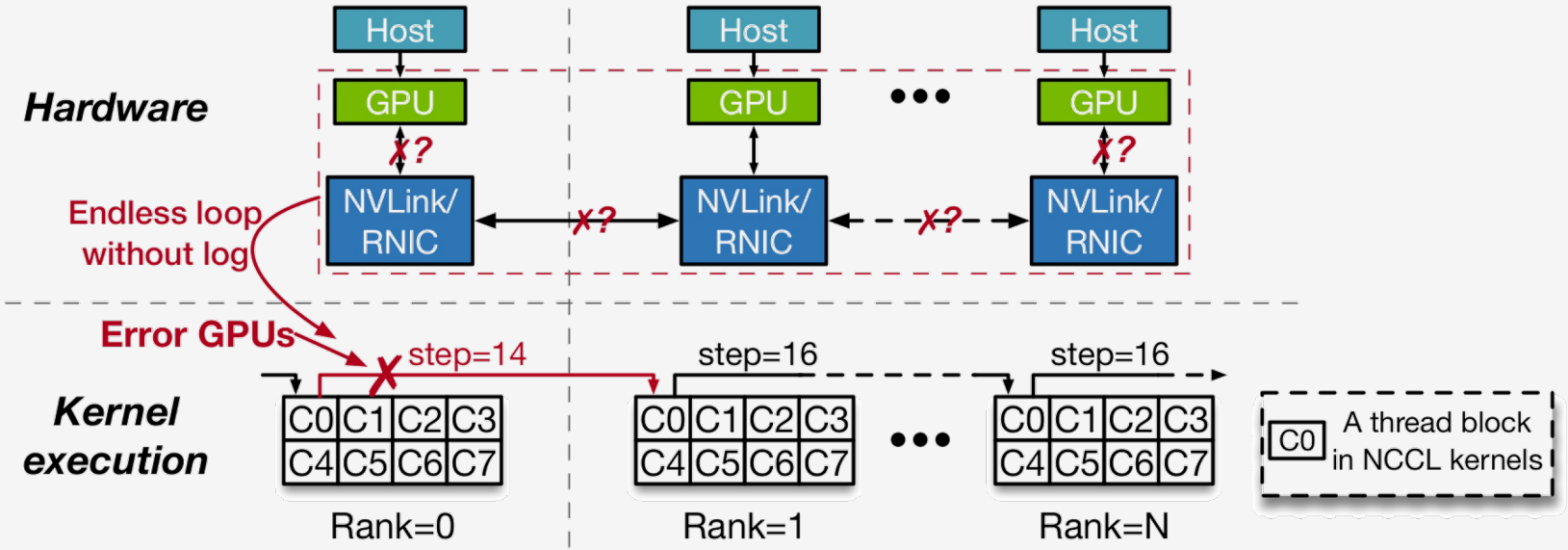
- Step 1: compare call stacks across ranks — non-communication hang ↔ the diverging rank is faulty.



Non-comm hang: call-stack divergence pinpoints faulty rank

Hang-error diagnosis

- Step 1: compare call stacks across ranks — non-communication hang ↔ the diverging rank is faulty.
- Step 2: for communication hang, attach CUDA-GDB to halted processes, read ring-step registers from SASS in parallel — minimum step points to the faulty link, complexity O(1).



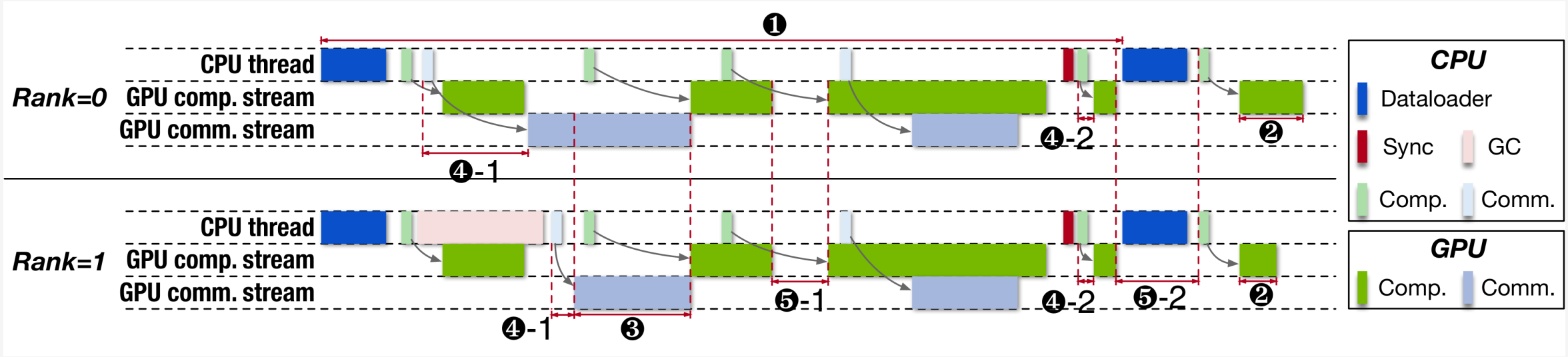
Comm hang: CUDA-GDB reads ring-allreduce step registers in parallel

Hang-error diagnosis

- Step 1: compare call stacks across ranks — non-communication hang ↔ the diverging rank is faulty.
- Step 2: for communication hang, attach CUDA-GDB to halted processes, read ring-step registers from SASS in parallel — minimum step points to the faulty link, complexity O(1).

✓ **Sub-5-min localization vs >30 min blind NCCL-test at thousand-GPU scale.**

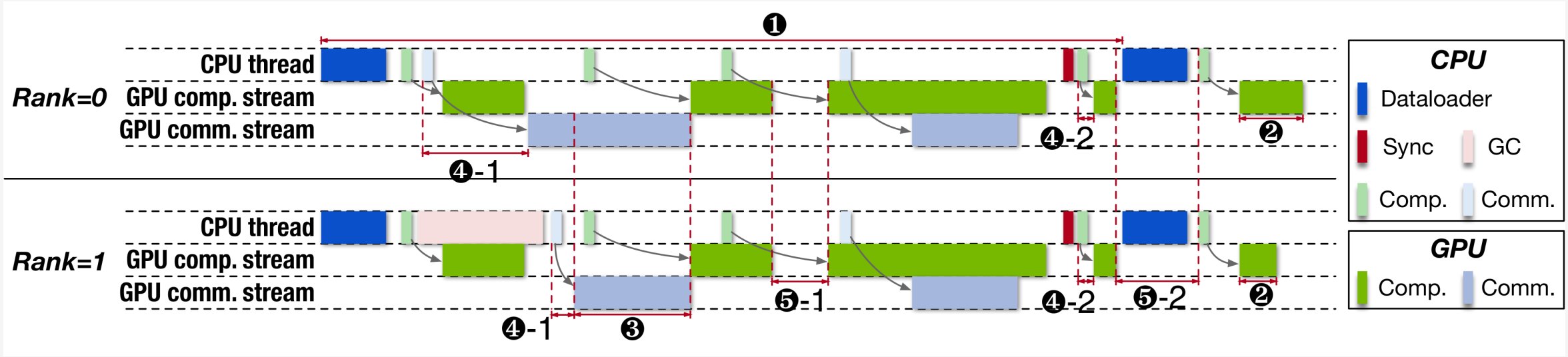
Five aggregated metrics for diagnostics



1 Training throughput

Macro — fail-slows

Five aggregated metrics for diagnostics



1 Training throughput

Macro — fail-slows

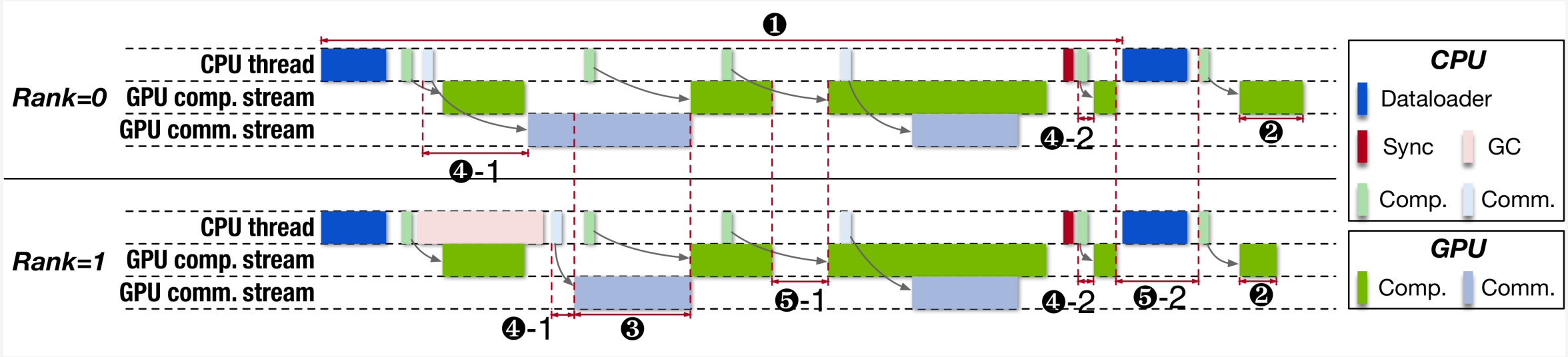
2 FLOPS

Computation kernel regression

3 Bandwidth

Communication kernel regression

Five aggregated metrics for diagnostics



1 Training throughput

Macro — fail-slows

2 FLOPS

Computation kernel regression

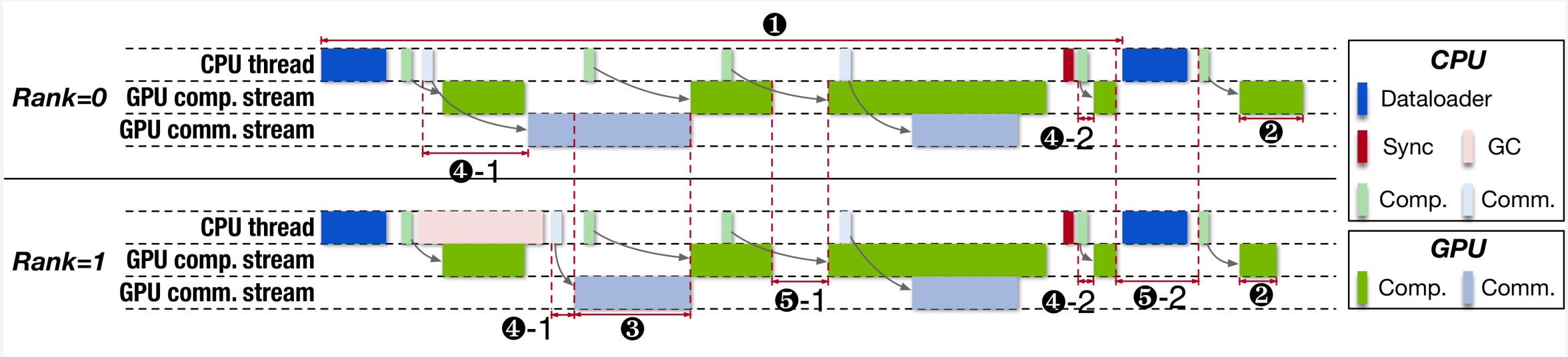
3 Bandwidth

Communication kernel regression

4 Issue latency dist.

Kernel-issue stall (Wasserstein)

Five aggregated metrics for diagnostics



1 Training throughput

Macro — fail-slows

2 FLOPS

Computation kernel regression

3 Bandwidth

Communication kernel regression

4 Issue latency dist.

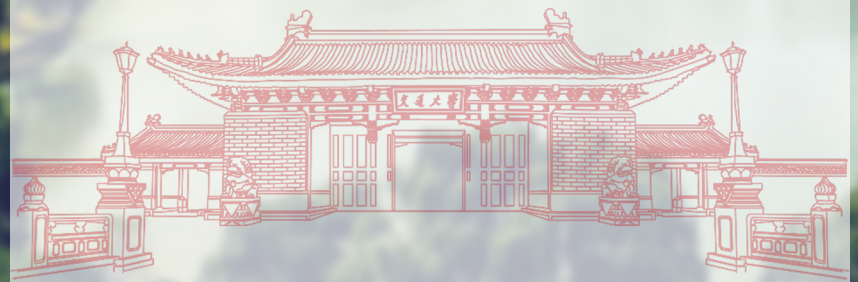
Kernel-issue stall (Wasserstein)

5 Void percentage

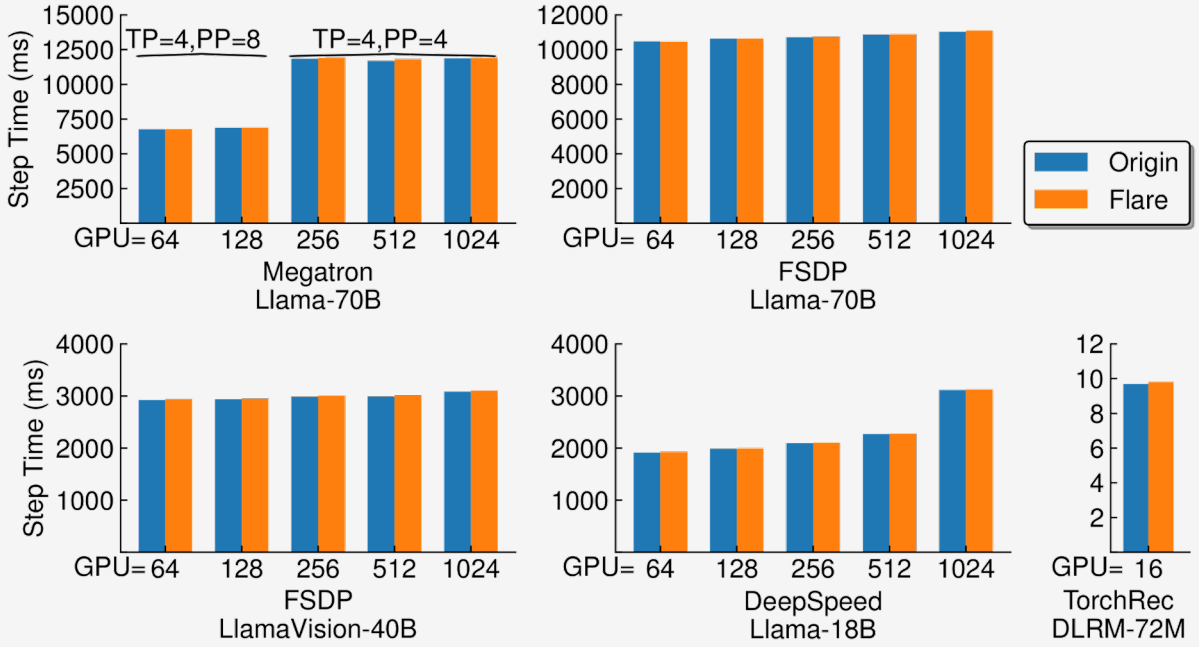
Inter-step + minority kernels

04

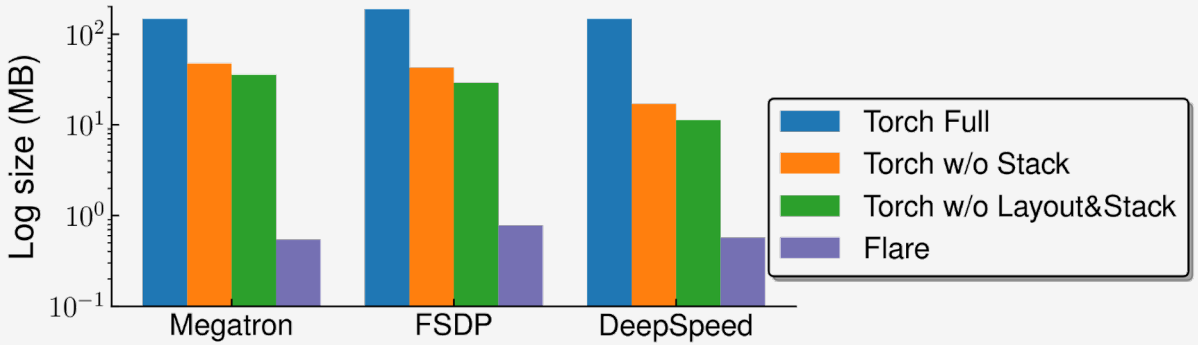
Evaluation



Runtime overhead is negligible at thousand-GPU scale



Latency overhead — 1,024 H800 GPUs, 4 backends



Log-memory overhead — Llama-70B, 16 A100 GPUs

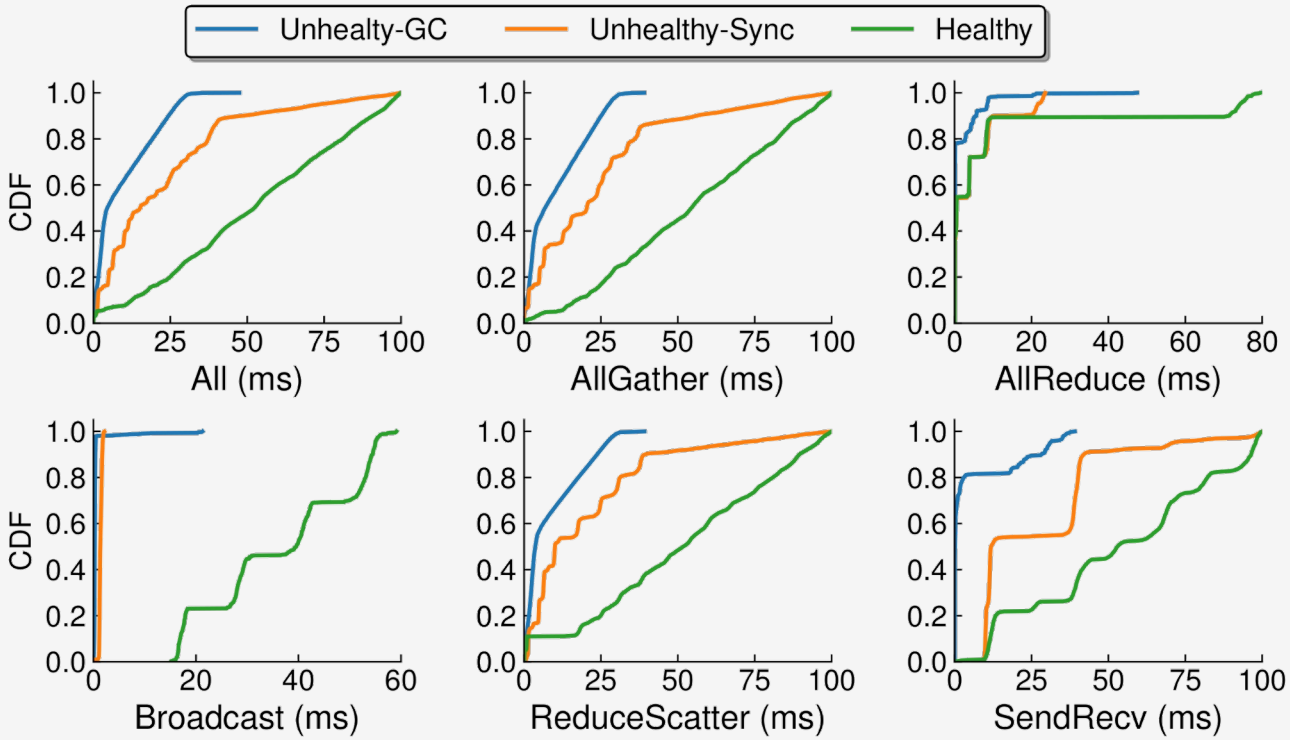
0.43%

Megatron / FSDP / DeepSpeed latency

1.5 MB

Log / GPU, Llama-20B on 1,536 H800

Issue-latency CDFs separate healthy from regression



Llama-20B on 256 H800 GPUs; overall CDF + per-kernel CDFs

Healthy

Linear CDF — issue latency dominated by collective op

Unhealthy-GC

Much steeper CDF — Python GC stalls kernel issues

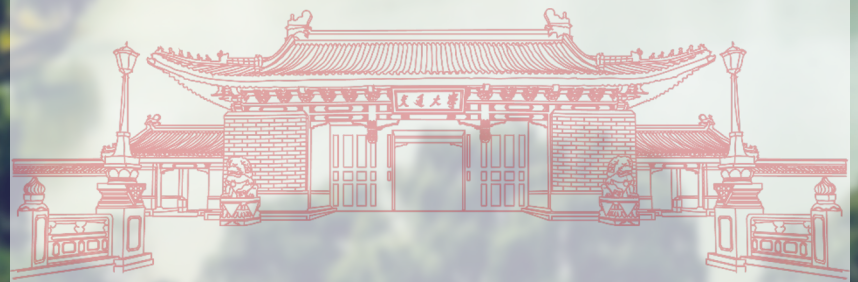
Unhealthy-Sync

Steep CDF — extra GPU sync in forward pass

Wasserstein distance > learned threshold → regression flag

05

Conclusion



FLARE: cluster-scale tracing for diverse LLM training

上海交通大學 SHANGHAI JIAO TONG UNIVERSITY

Plug-and-play tracing

CPython *PyEval_SetProfile* on bytecode + *LD_PRELOAD* function hooks — **no backend source change**.

Regression + hang diagnostics

Five aggregated metrics flag **silent regressions that throughput hides**; intra-kernel SASS scan localizes comm-hang **quickly**.

Production proven

0.43% latency, 1.5 MB/GPU log — **6,000 GPUs, 12 months**; cross-team collaboration on recurrent regressions **reduced**.

上海交通大學

SHANGHAI JIAO TONG UNIVERSITY

Thank You

Q&A

饮 水 思 源　爱 国 荣 校