

a storage system
that does not “store”

TensorHub: Scalable and Elastic Weight Transfer for LLM RL Training

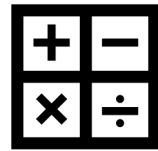
Chenhao Ye, Huaizheng Zhang, Mingcong Han, Baoquan Zhong, Xiang Li,
Qixiang Chen, Xinyi Zhang, Weidong Zhang, Kaihua Jiang, Wang Zhang, He Sun,
Wencong Xiao, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau



Reinforcement Learning Unlocks LLM Capabilities



Reasoning



Math



Code



Tool Use



Alignment

DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning

December 20, 2024 Publication Release Safety

September 8, 2026 Research Publication

**On the Navier–Stokes
Millennium Prize Problem**

Co

Alignment:
enables
models

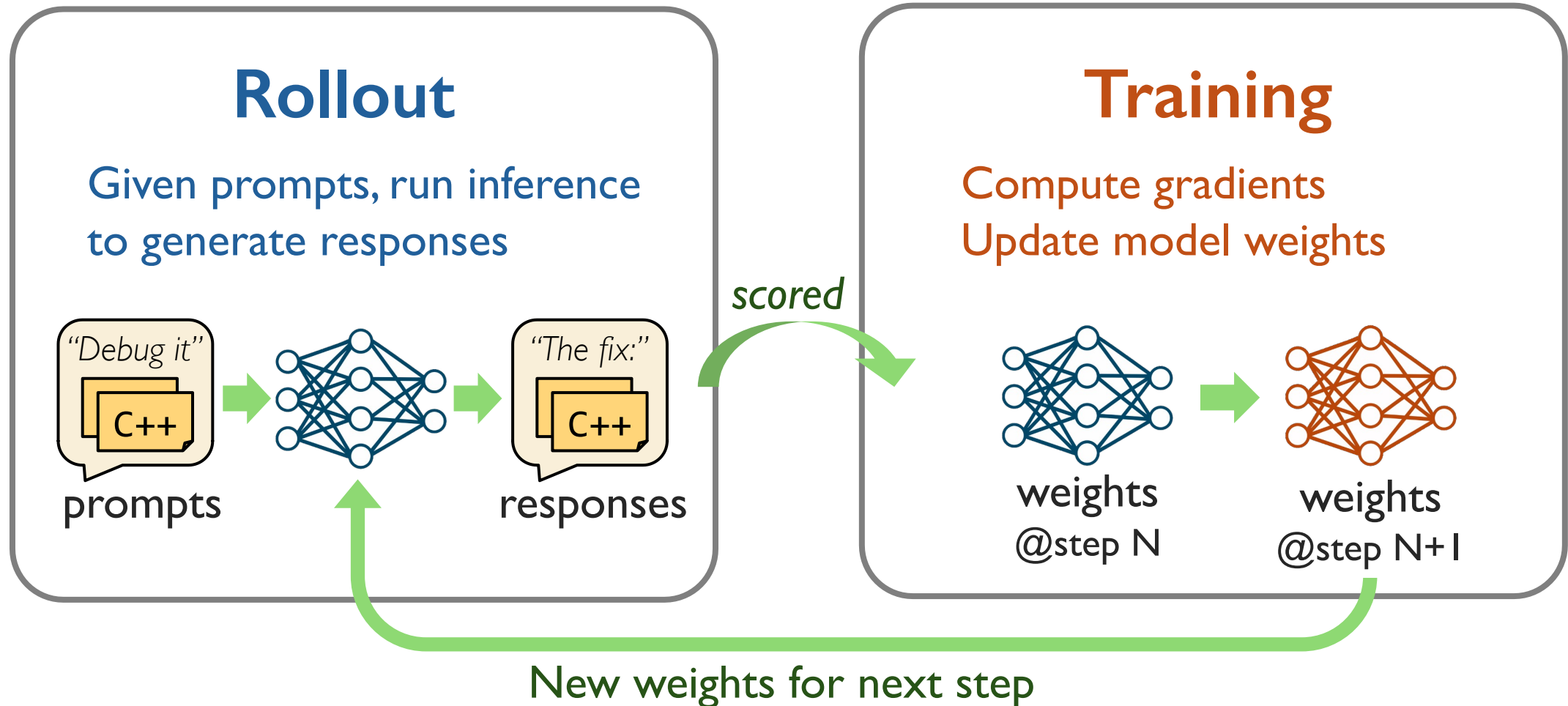
ies models, which
o reason over them.

work on
ChatGPT

universal interface for AI to interact with the digital world.

Pro, Business, and Enterprise users today, and Plus users soon.

RL Workload: Rollout-Training Loop

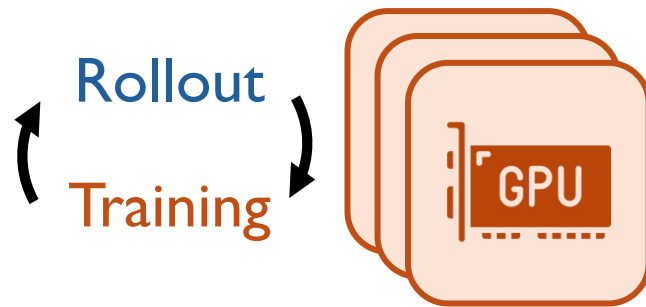


Scheduling RL Tasks on GPU Workers

Co-located, Time-Shared

on same GPU workers

e.g., HybridFlow [EuroSys '25]



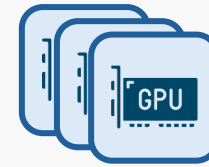
Step time breakdown



Dominant in time

→ Scale out rollout resources

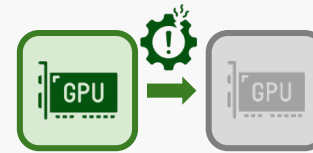
Scaling Out Rollout on diverse compute resources



Dedicated GPUs

Rollout in a tight loop

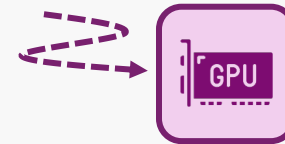
e.g., Laminar [EuroSys '26]



Spot GPUs

Preemptible at any time

e.g., RLBoost [NSDI '26]



Remote Datacenter's GPUs

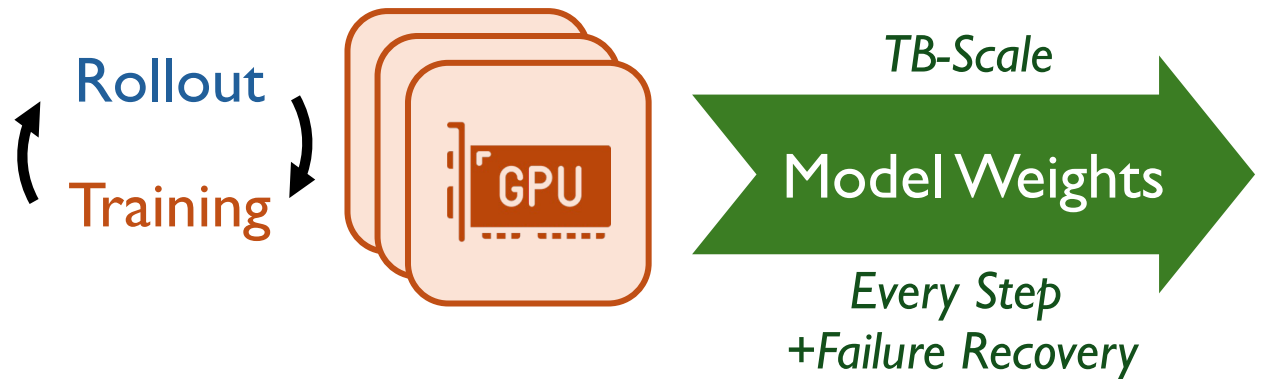
Over slow inter-DC network

e.g., StreamRL

Scheduling RL Tasks on GPU Workers

Co-located, Time-Shared

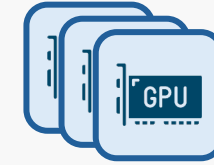
on same GPU workers
e.g., HybridFlow [EuroSys '25]



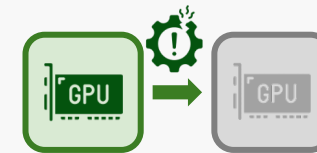
Weight transfer system must be
fast, scalable, and elastic

Scaling Out Rollout on diverse compute resources

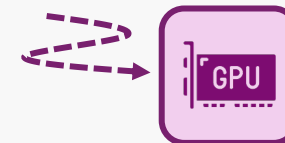
Dedicated GPUs



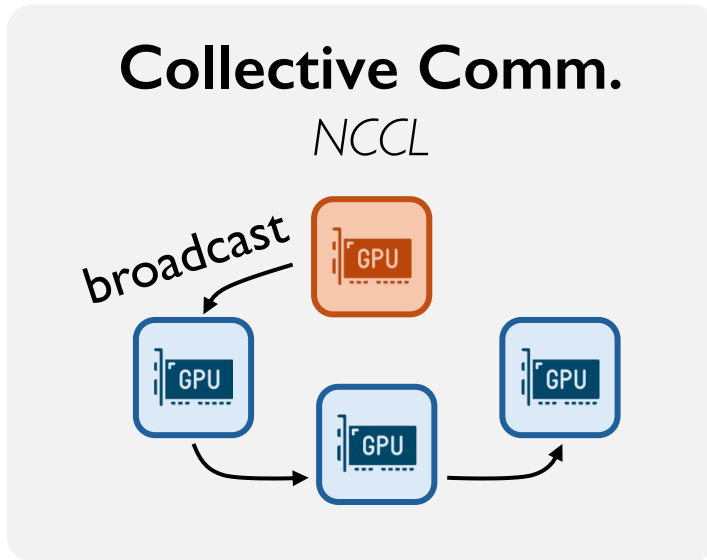
Spot GPUs



Remote Datacenter's GPUs



Existing Weight Transfer Approaches



✓ **Topology Aware**

✗ **Not Elastic/Fault Tolerant**

Static membership

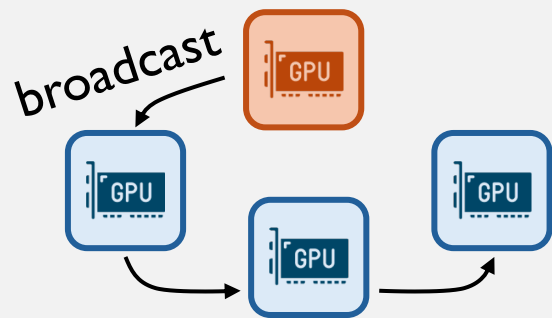
No runtime add/remove

 **Trainer** (worker)  **Rollout** (worker)

Existing Weight Transfer Approaches

Collective Comm.

NCCL



✓ Topology Aware

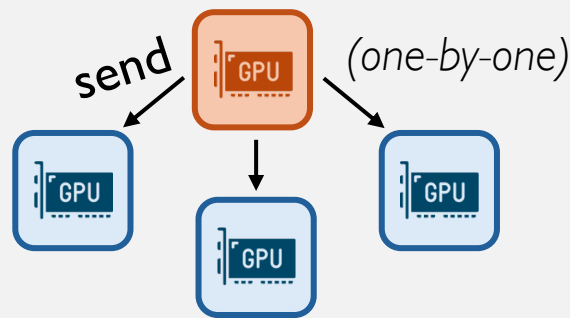
✗ Not Elastic/Fault Tolerant

Static membership

No runtime add/remove

Point-to-Point Comm.

UCX



✓ Elastic/Fault Tolerant

✗ Not Topology Aware

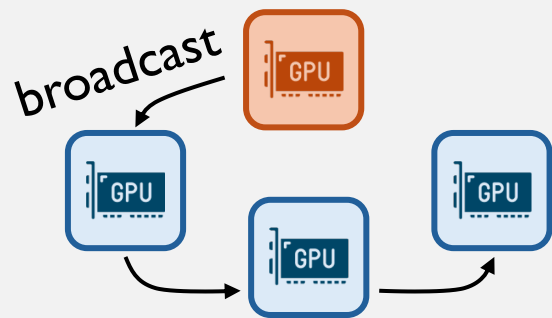
Contention on trainers' network bandwidth

 **Trainer** (worker)  **Rollout** (worker)

Existing Weight Transfer Approaches

Collective Comm.

NCCL

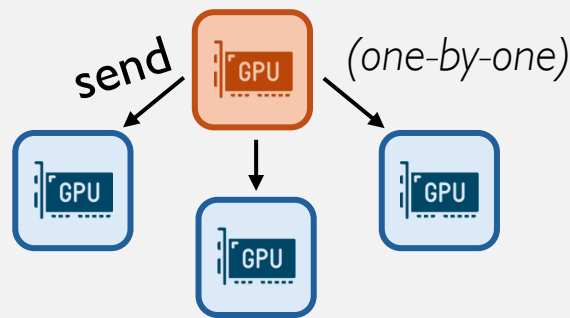


✓ Topology Aware

✗ Not Elastic/Fault Tolerant

Point-to-Point Comm.

UCX



✓ Elastic/Fault Tolerant

✗ Not Topology Aware

✗ Temporal Coupling b/w Trainer & Rollout

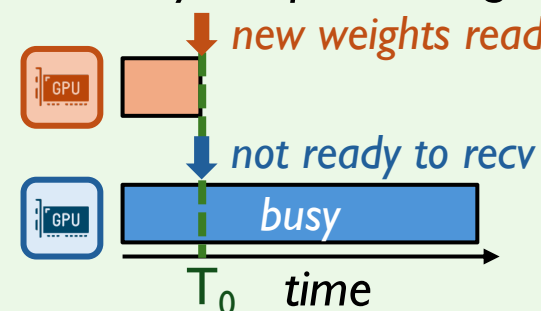
Requires synchronous coordination

 **Trainer** (worker)  **Rollout** (worker)

Example: why coupling hurts

Trainer: new weights ready at T_0

Rollout: busy with a long response; not ready to update weights at T_0



😓 *Either:*

- Trainer waits, OR

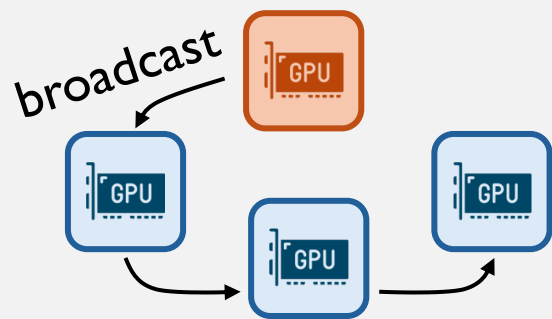
- Rollout aborts inference

**Coupling → Forced Synchrony
→ Hurt Utilization**

Existing Weight Transfer Approaches

Collective Comm.

NCCL

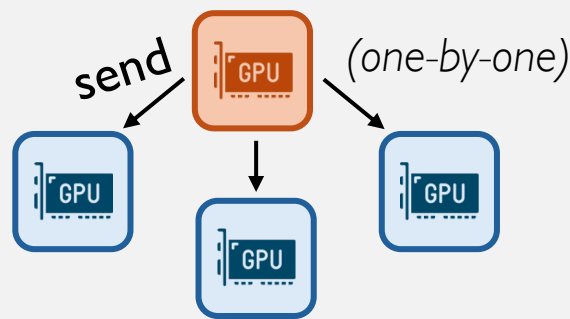


✓ Topology Aware

✗ Not Elastic/Fault Tolerant

Point-to-Point Comm.

UCX



✓ Elastic/Fault Tolerant

✗ Not Topology Aware

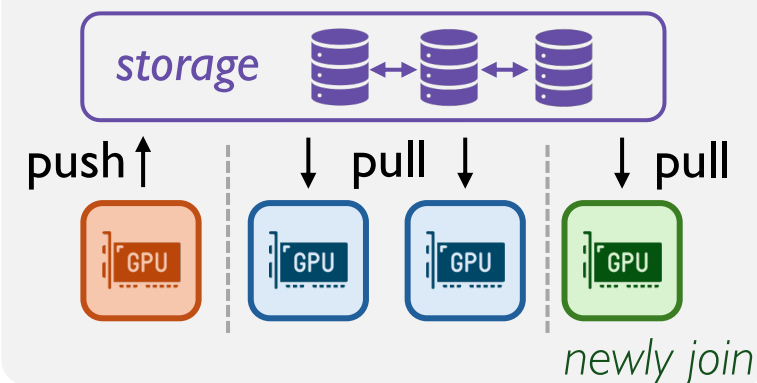
✗ Temporal Coupling b/w Trainer & Rollout

Requires synchronous coordination

 **Trainer** (worker)  **Rollout** (worker)

Distributed Storage

Parameter Server



✓ Trainer-Rollout Decoupling!

Trainers push, w/ waiting

Rollouts pull on demand

Rollouts join w/o trainers' awareness

✗ Amplified Data Movement

✗ Expensive Storage Space

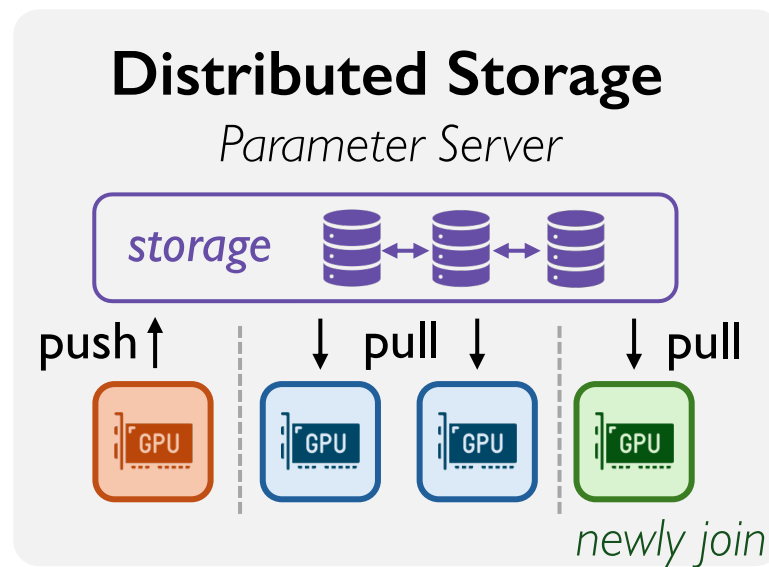
Existing Weight Transfer Approaches

Why storage slow, fundamentally?

On write, storage owns a copy for lifecycle control

- *Copy to storage-managed space*
→ *Extra data movement*
- *Keep the storage-owned copy*
→ *Extra storage space*

Can we retain decoupling of storage
w/o movement/space overhead? 🤔



✓ **Trainer-Rollout Decoupling!**

Trainers push, w/ waiting

Rollouts pull on demand

Rollouts join w/o trainers' awareness

✗ **Amplified Data Movement**

✗ **Expensive Storage Space**

Idea: Reference-Oriented Storage (ROS)

ROS does not own a data copy

It references the *in-memory copy*
shared with the application
rollout inference engine

Key observation: Model weights are...

Immutable *during inference*

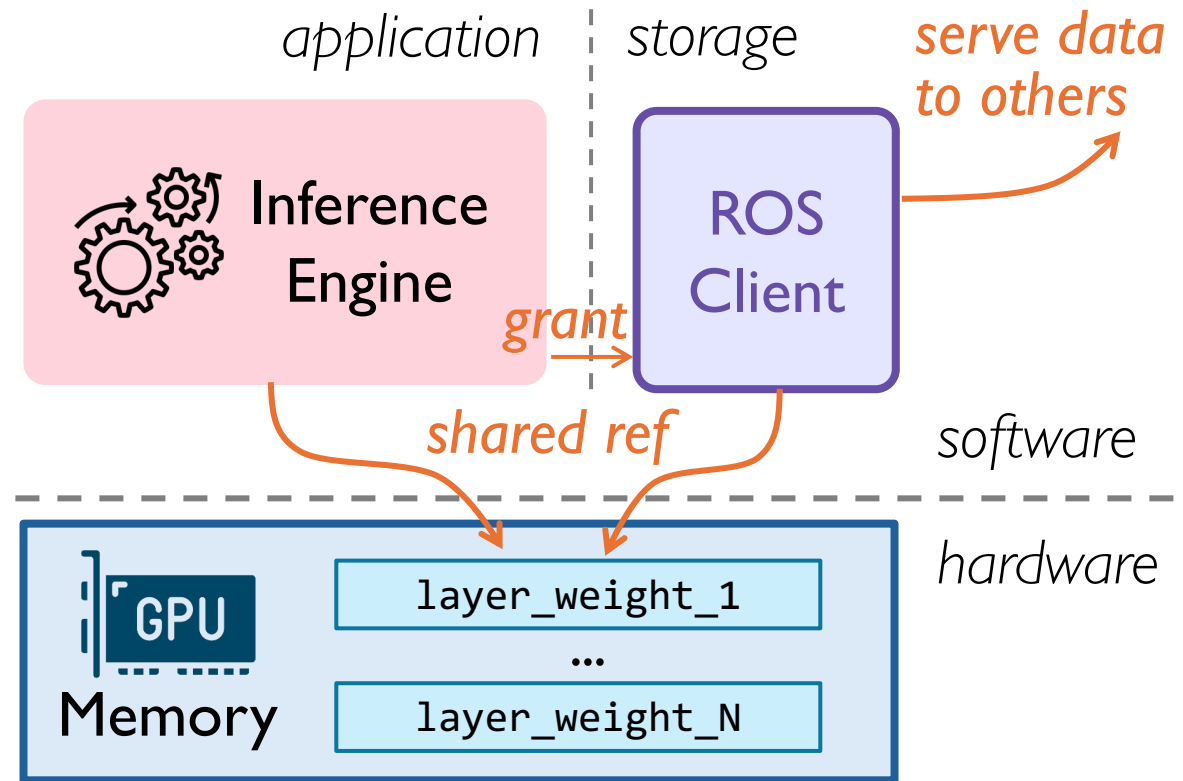
→ **Shared references are safe!**

Highly replicated *due to data parallelism*

→ **Free redundancy!**

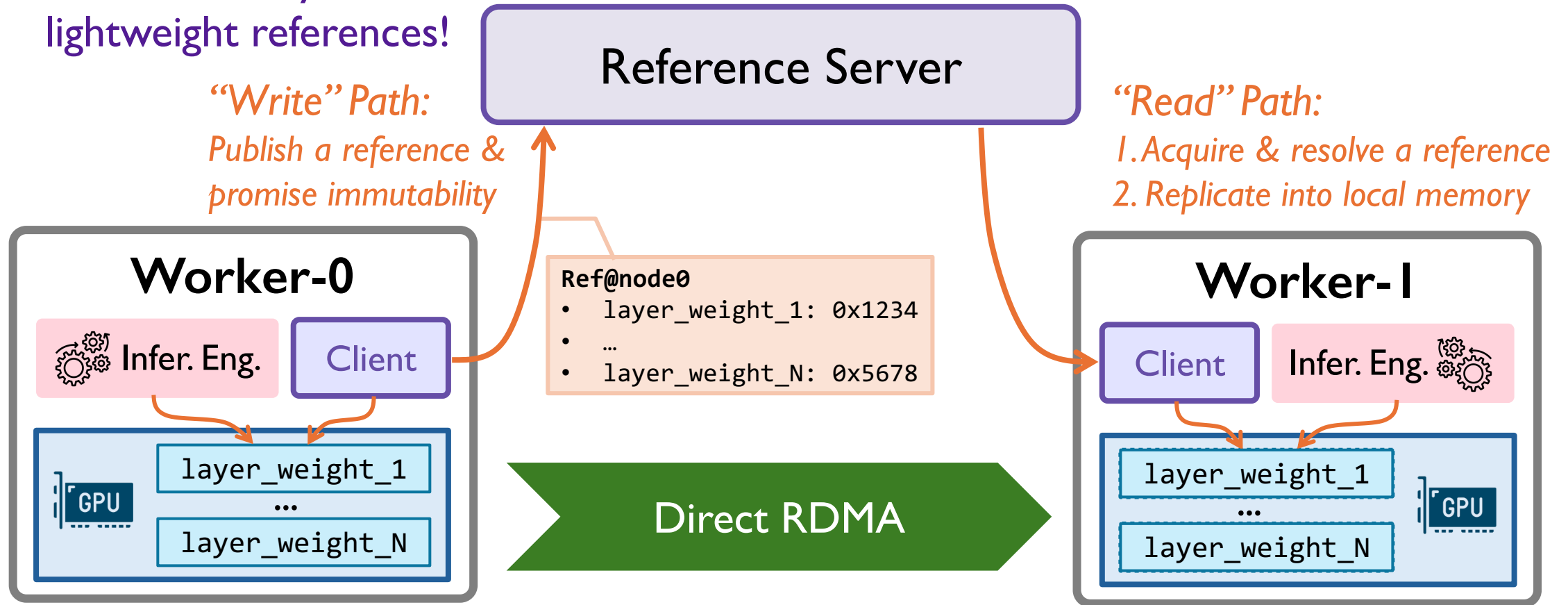
Short-lived *as new version at each step*

→ **Persistence not required!**



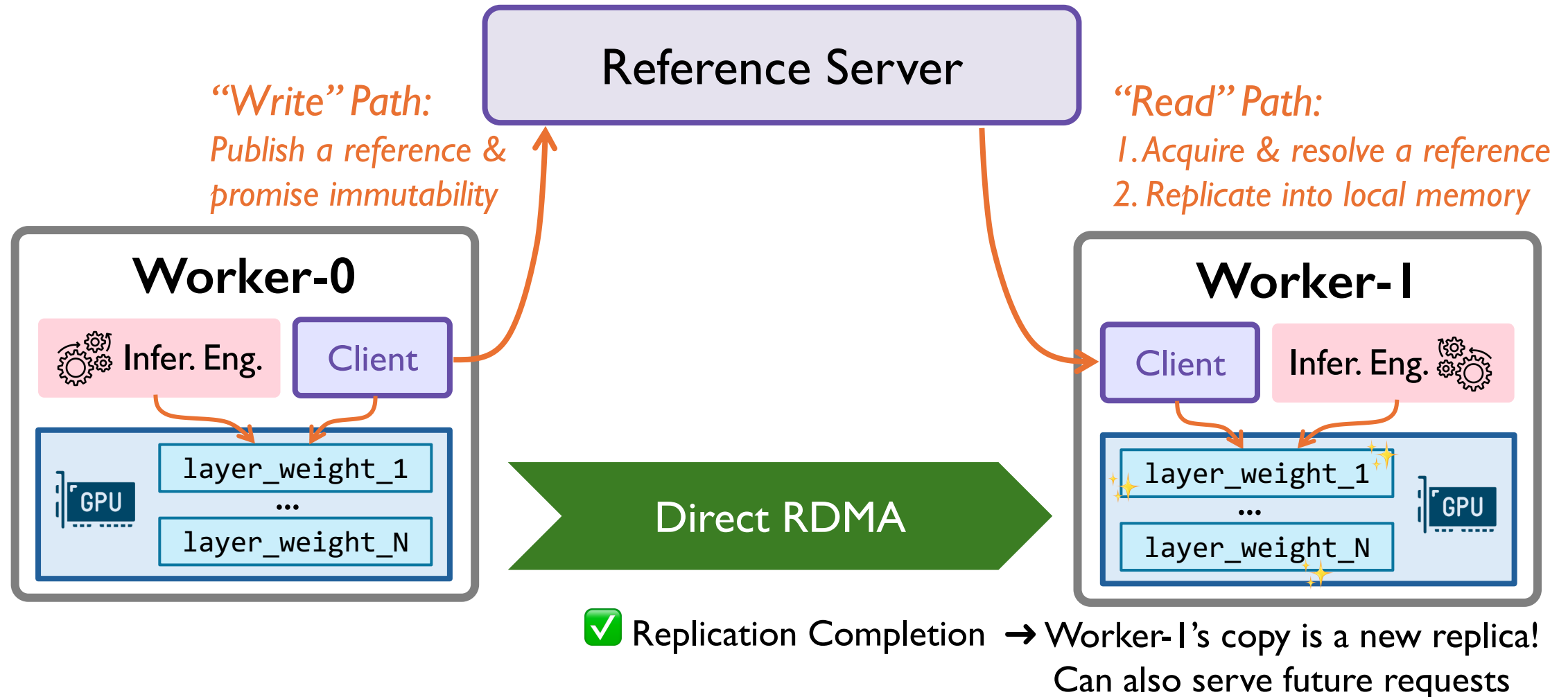
Idea: Reference-Oriented Storage (ROS)

Ref server only stores
lightweight references!



Heavyweight data transfer is directly b/w clients
No intermediate hop!

Idea: Reference-Oriented Storage (ROS)



TensorHub: ROS Made Live

We built **TensorHub**, a production-quality system based on the **ROS** idea

Goals:

- Easy-to-Use
- High-Performance
- Consistent
- Reliable

TensorHub features:

- Concise Versioning APIs
- Topology-Aware Transfer
- Model-Parallel Consistency
- Fault Tolerance & Recovery

TensorHub: ROS Made Live

We built **TensorHub**, a production-quality system based on the **ROS** idea

Goals:

- Easy-to-Use
- High-Performance
- Consistent
- Reliable

TensorHub features:

- Concise Versioning APIs
- Topology-Aware Transfer
- Model-Parallel Consistency
- Fault Tolerance & Recovery

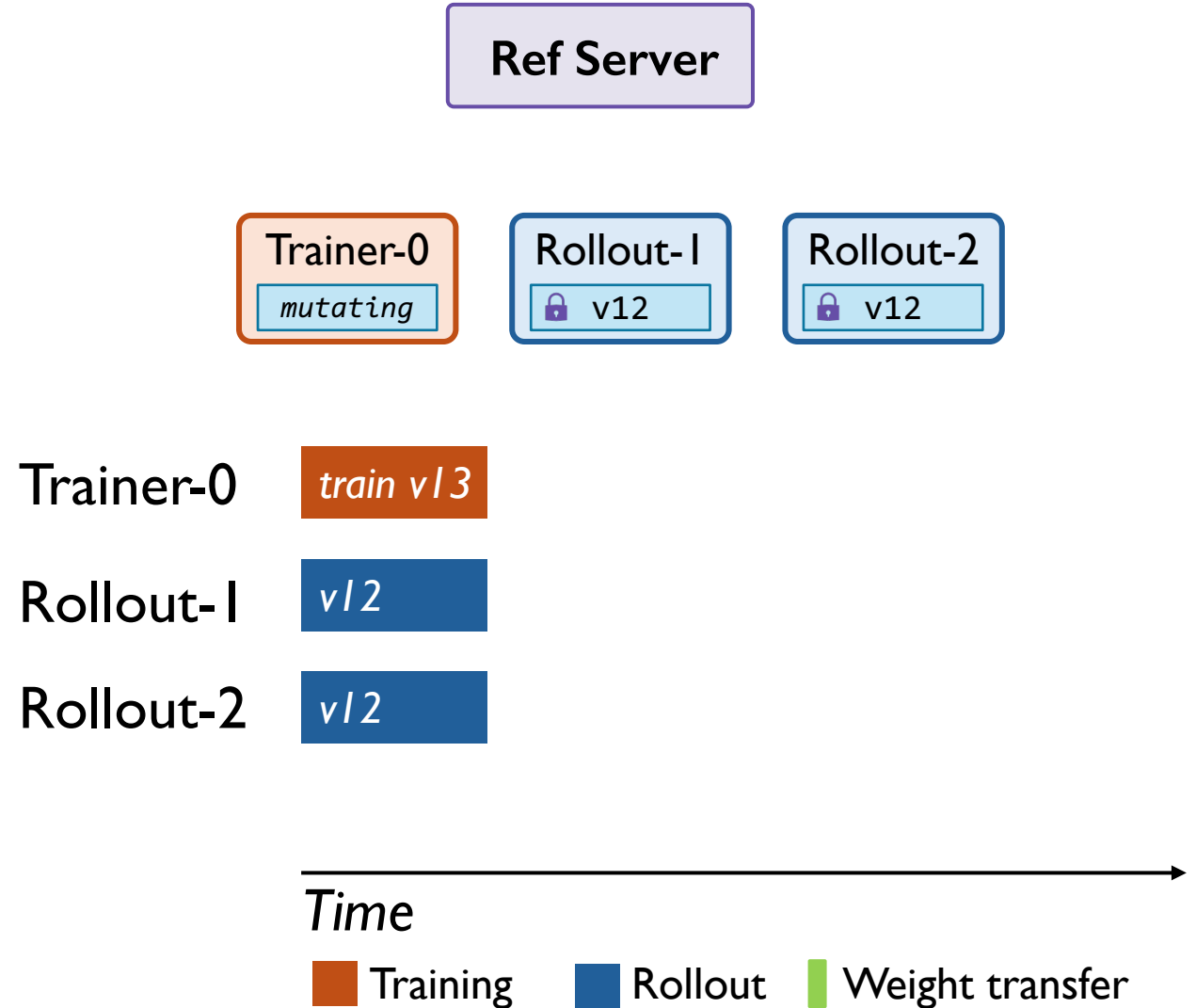
TensorHub: ROS Made Live

↑ publish(v)

⊘ unpublish()

↓ replicate(v)

↻ update(v)



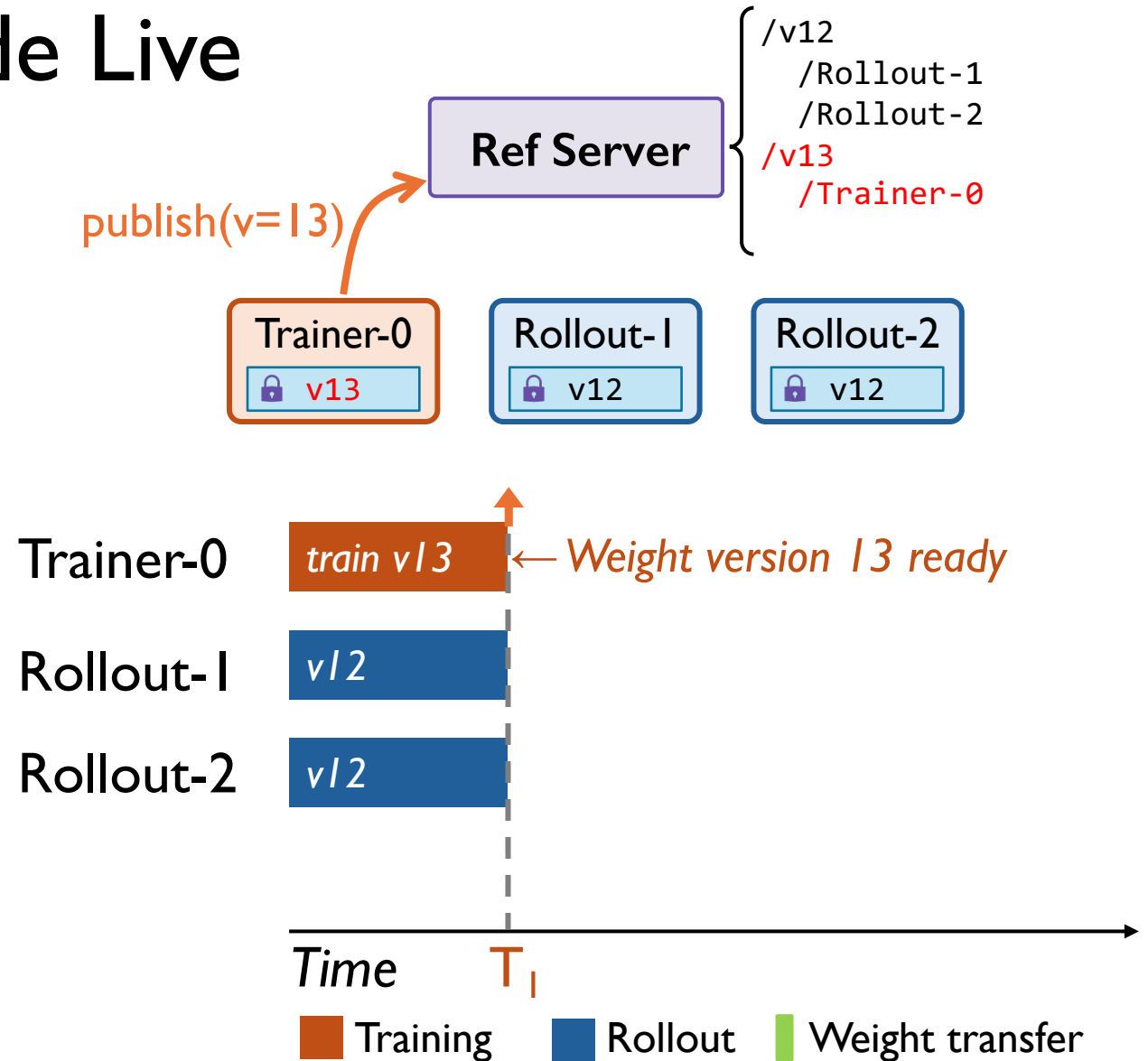
TensorHub: ROS Made Live

↑ **publish(v)**
*Make weights visible as version v;
Promise immutability* 🔒

⊘ **unpublish()**

↓ **replicate(v)**

↻ **update(v)**



TensorHub: ROS Made Live

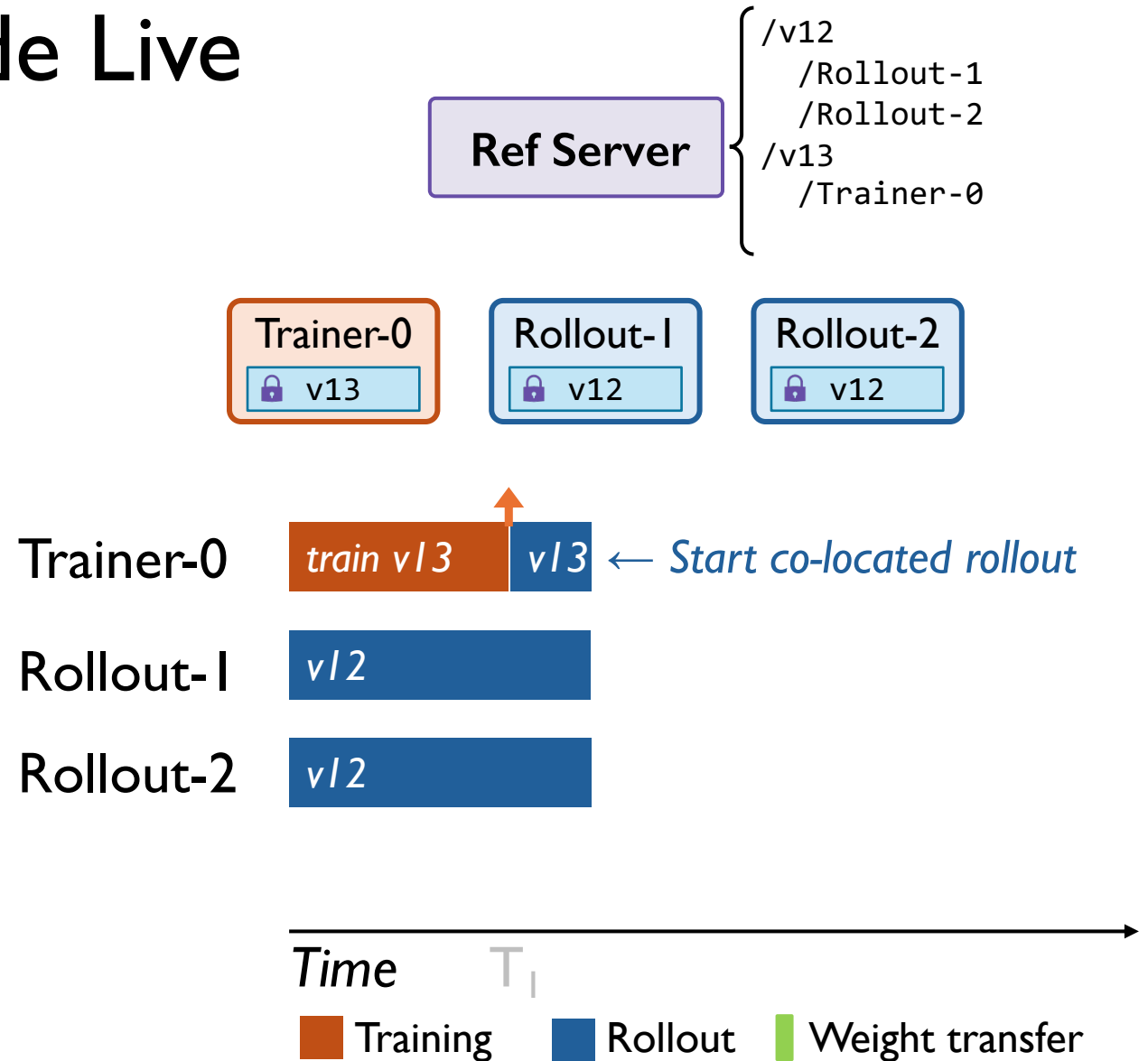
↑ **publish(v)**

*Make weights visible as version v;
Promise immutability 🔒*

⊘ **unpublish()**

↓ **replicate(v)**

↻ **update(v)**



TensorHub: ROS Made Live

↑ **publish(v)**

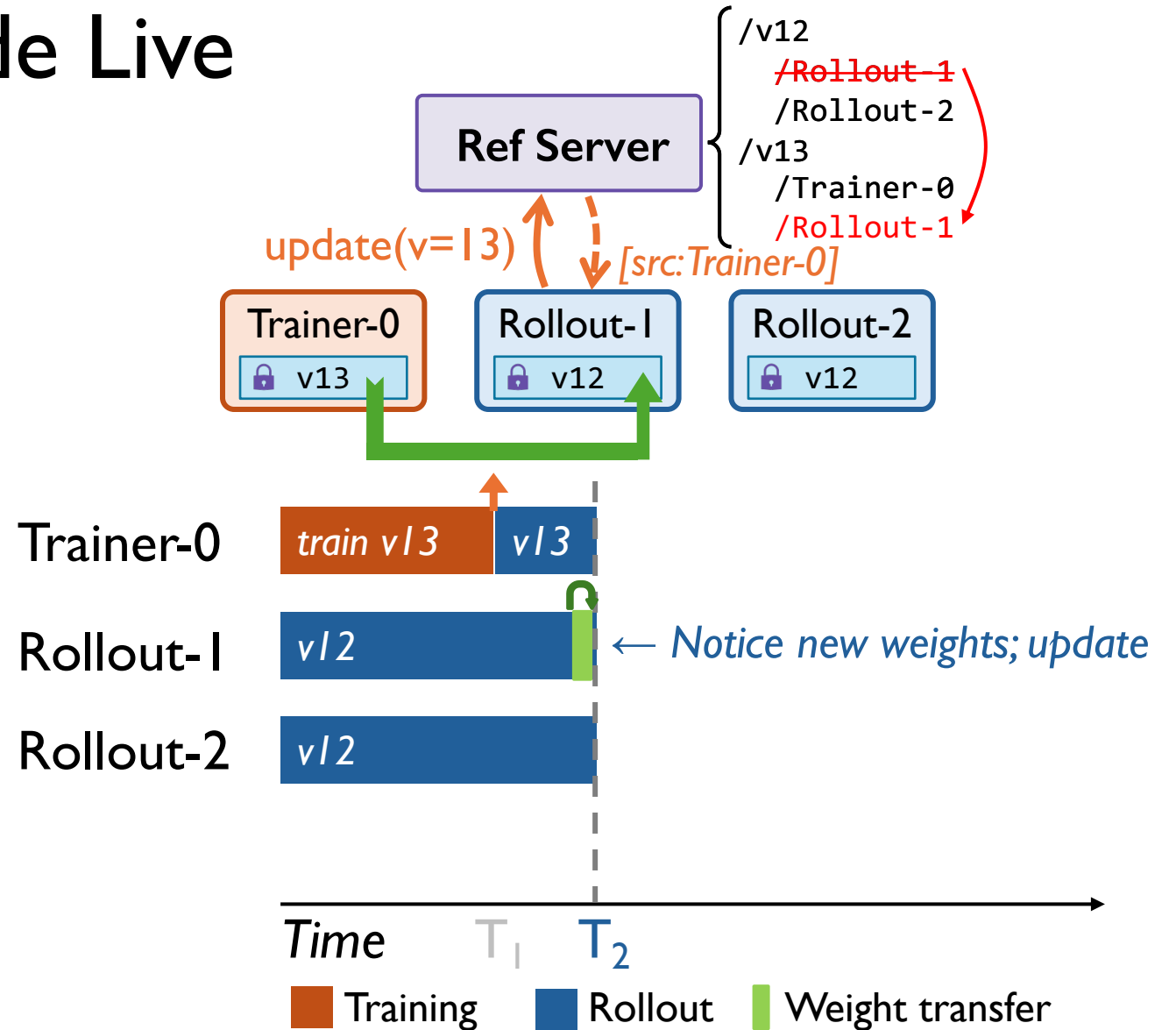
*Make weights visible as version v;
Promise immutability 🔒*

⊘ **unpublish()**

↓ **replicate(v)**

↻ **update(v)**

Atomically switch to new version v



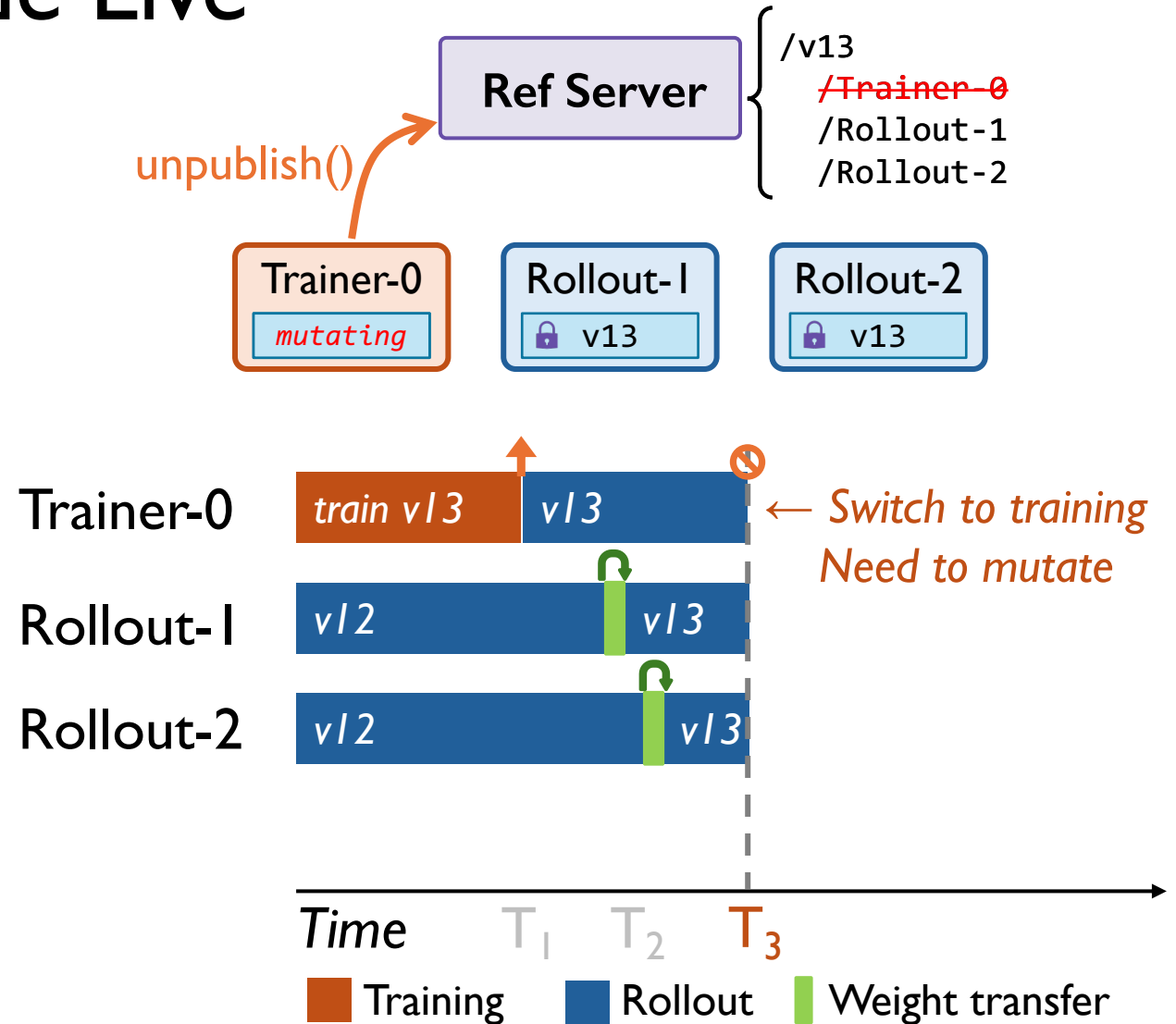
TensorHub: ROS Made Live

↑ **publish(v)**
*Make weights visible as version v;
Promise immutability* 🔒

⊘ **unpublish()**
*Revoke visibility;
Can mutate afterwards*

↓ **replicate(v)**

↻ **update(v)**
Atomically switch to new version v



TensorHub: ROS Made Live

↑ publish(v)

Make weights visible as version v;
Promise immutability 🔒

⊘ unpublish()

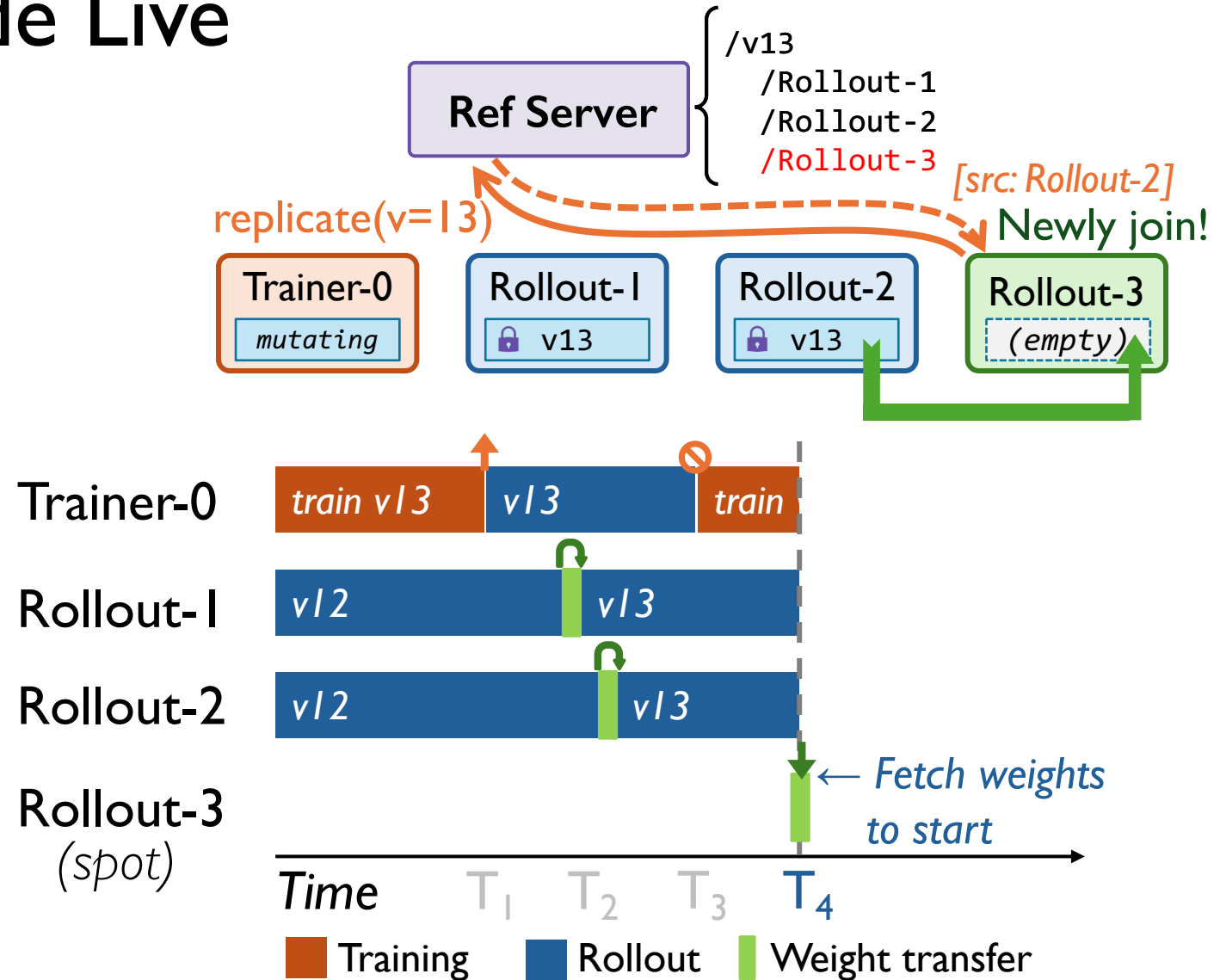
Revoke visibility;
Can mutate afterwards

↓ replicate(v)

Replicate a copy of version v;
Promise immutability, too 🔒

↻ update(v)

Atomically switch to new version v



TensorHub: ROS Made Live

↑ publish(v)

*Make weights visible as version v;
Promise immutability 🔒*

⊘ unpublish()

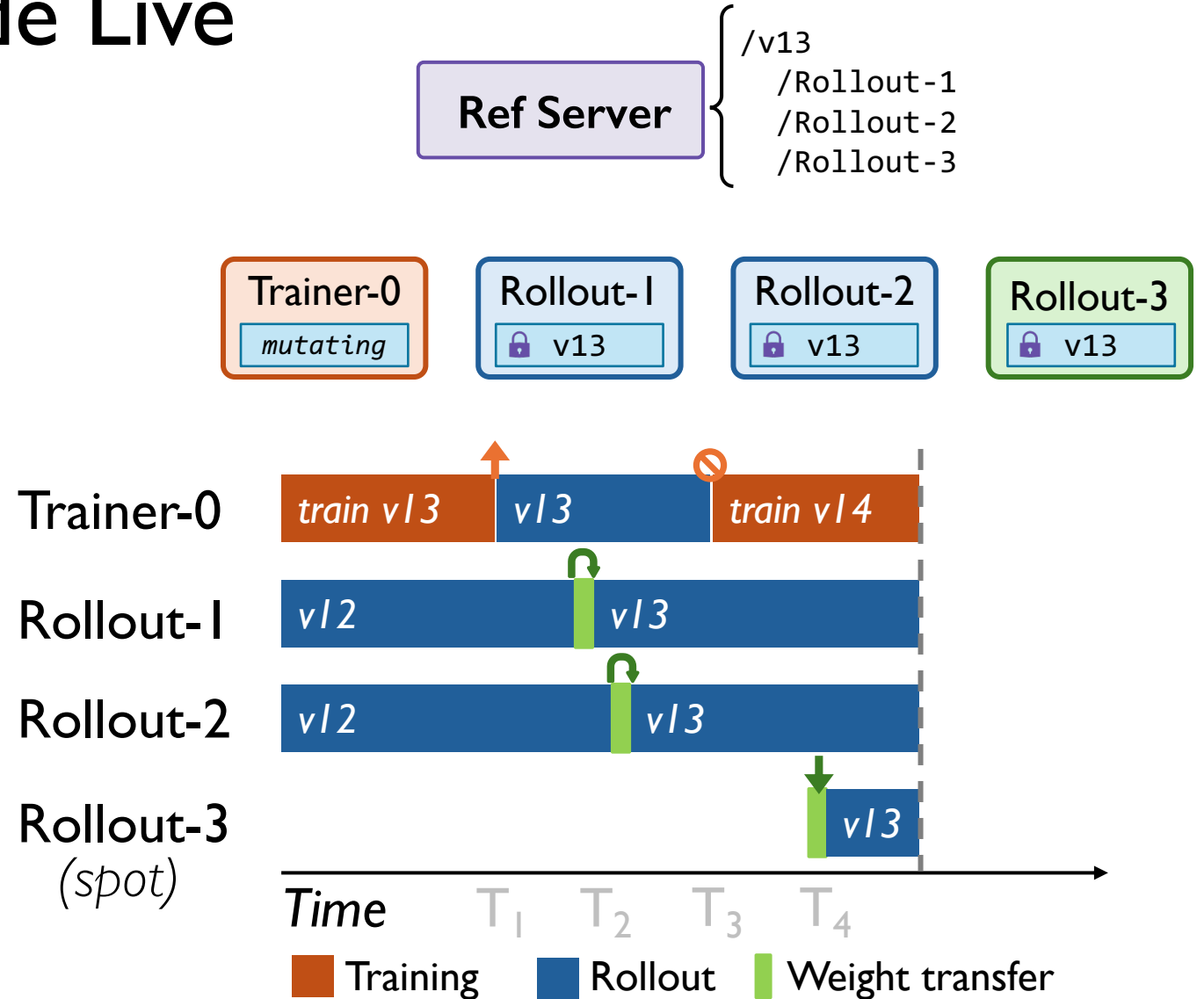
*Revoke visibility;
Can mutate afterwards*

↓ replicate(v)

*Replicate a copy of version v;
Promise immutability, too 🔒*

↻ update(v)

Atomically switch to new version v



TensorHub: ROS Made Live

↑ publish(v)

*Make weights visible as version v;
Promise immutability 🔒*

⊘ unpublish()

*Revoke visibility;
Can mutate afterwards*

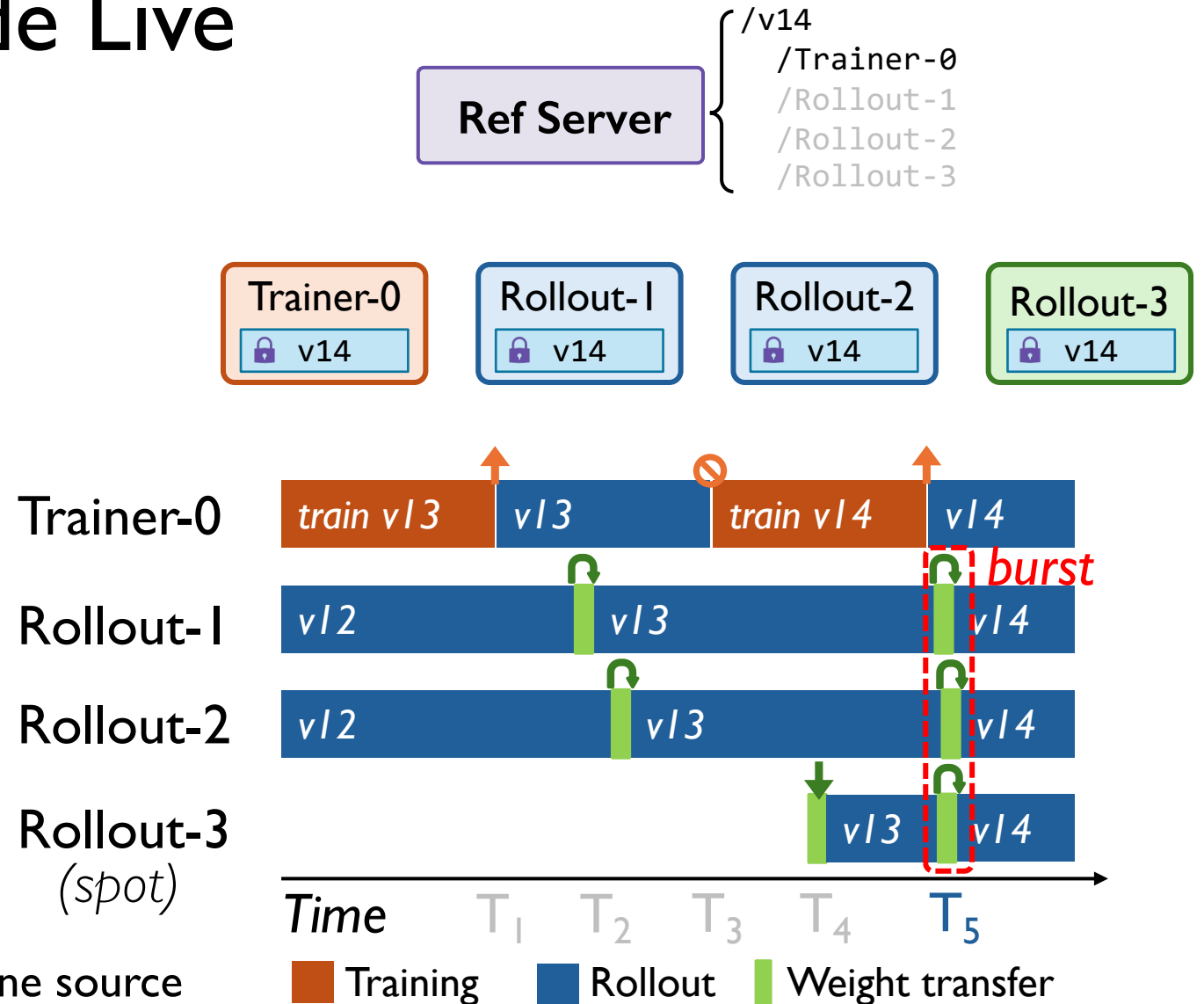
↓ replicate(v)

*Replicate a copy of version v;
Promise immutability, too 🔒*

↻ update(v)

Atomically switch to new version v

Challenge: Burst of requests, but only one source

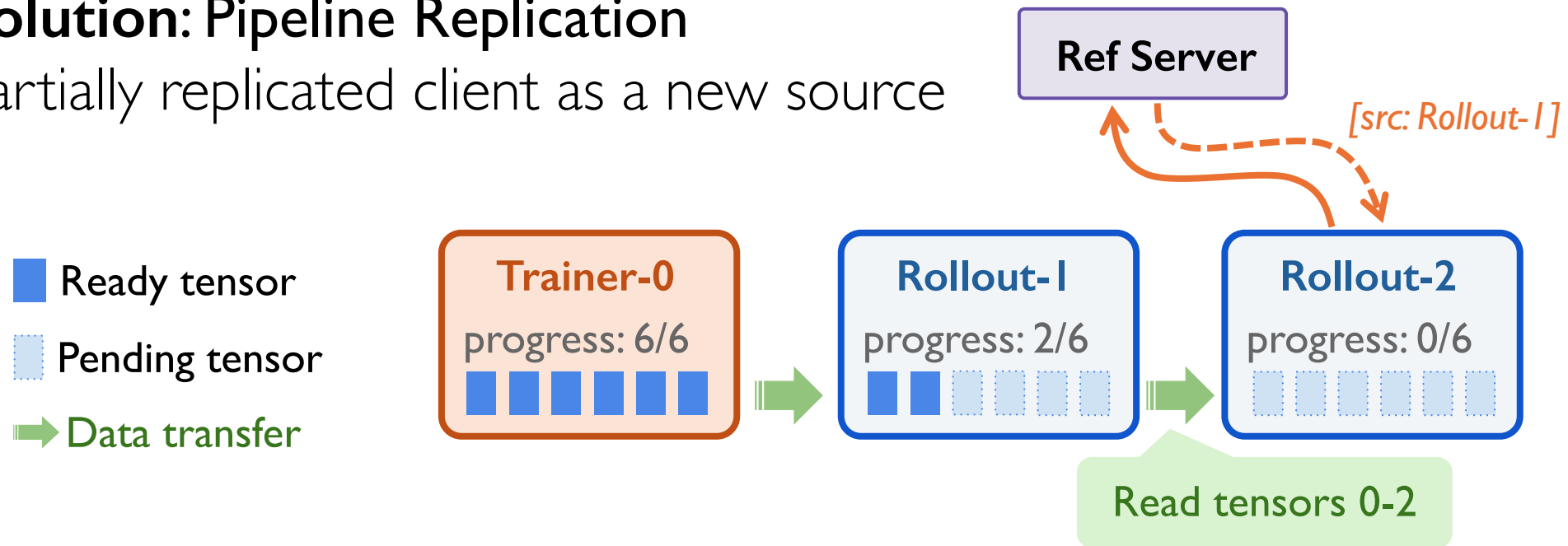


TensorHub: Scalable Replication

Challenge: Burst of requests, but only one source

Solution: Pipeline Replication

Partially replicated client as a new source



TensorHub: Scalable Replication

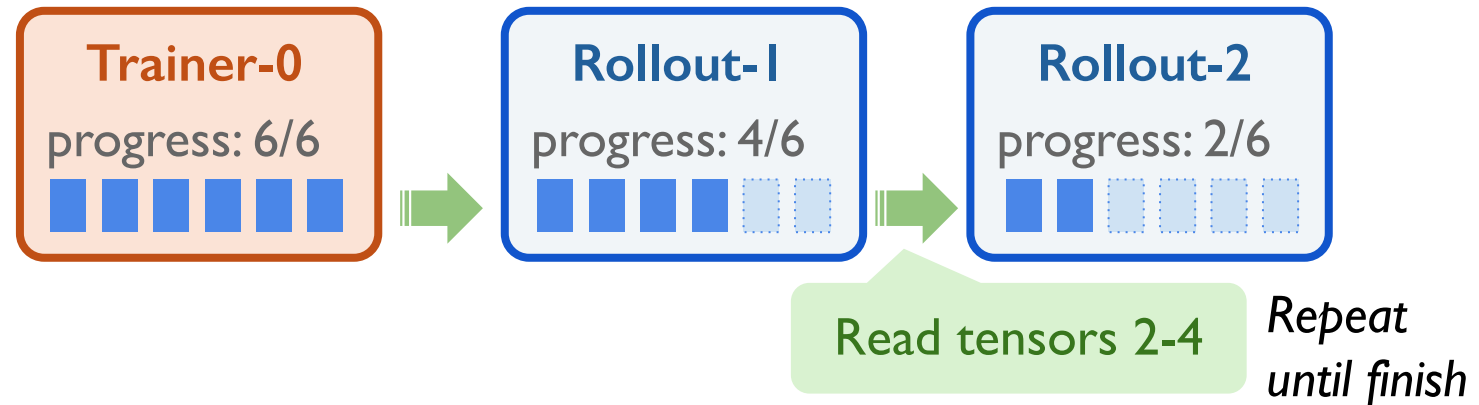
Challenge: Burst of requests, but only one source

Solution: Pipeline Replication

Partially replicated client as a new source

Ref Server

■ Ready tensor
■ Pending tensor
➡ Data transfer



TensorHub: Scalable Replication

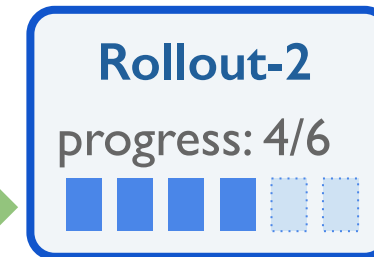
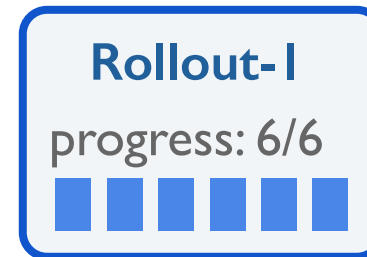
Challenge: Burst of requests, but only one source

Solution: Pipeline Replication

Partially replicated client as a new source

Ref Server

■ Ready tensor
■ Pending tensor
➡ Data transfer



Read tensors 4-6

*Repeat
until finish*

TensorHub: Scalable Replication

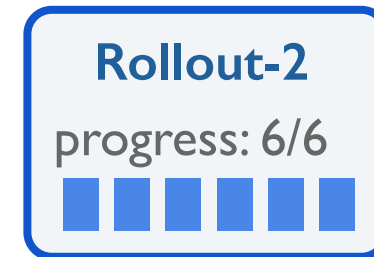
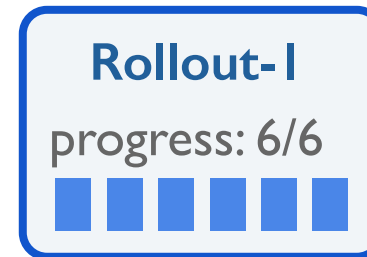
Challenge: Burst of requests, but only one source

Solution: Pipeline Replication

Partially replicated client as a new source

Ref Server

- Ready tensor
- Pending tensor
- ➡ Data transfer



Topology: Dynamic scheduled

No pre-created collective group (unlike NCCL)

More in the Paper

Ensure availability if everyone unpublishes?

→ Retention Protocol

Optimize cross-datacenter transfer?

→ Seeding Replica

Support model-parallel sharding?

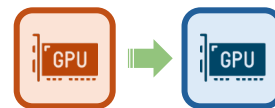
→ SPMD-Aware Transaction

Tolerate worker/server failure?

→ Failure Recovery

Three Case Studies

Standalone Rollout



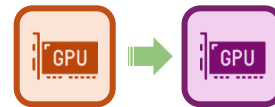
6.7x↓
GPU stall time

Elastic Rollout



4.8x↓
Tail transfer latency

Cross-DC Rollout



19x↓
GPU stall time

More in the Paper

Ensure availability if everyone unpublishes?

→ Retention Protocol

Optimize cross-datacenter transfer?

→ Seeding Replica

Support model-parallel sharding?

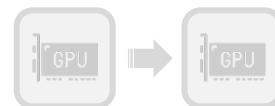
→ SPMD-Aware Transaction

Tolerate worker/server failure?

→ Failure Recovery

Three Case Studies

Standalone Rollout



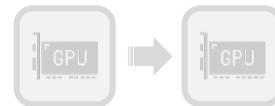
6.7x↓
GPU stall time

Elastic Rollout



4.8x↓
Tail transfer latency

Cross-DC Rollout



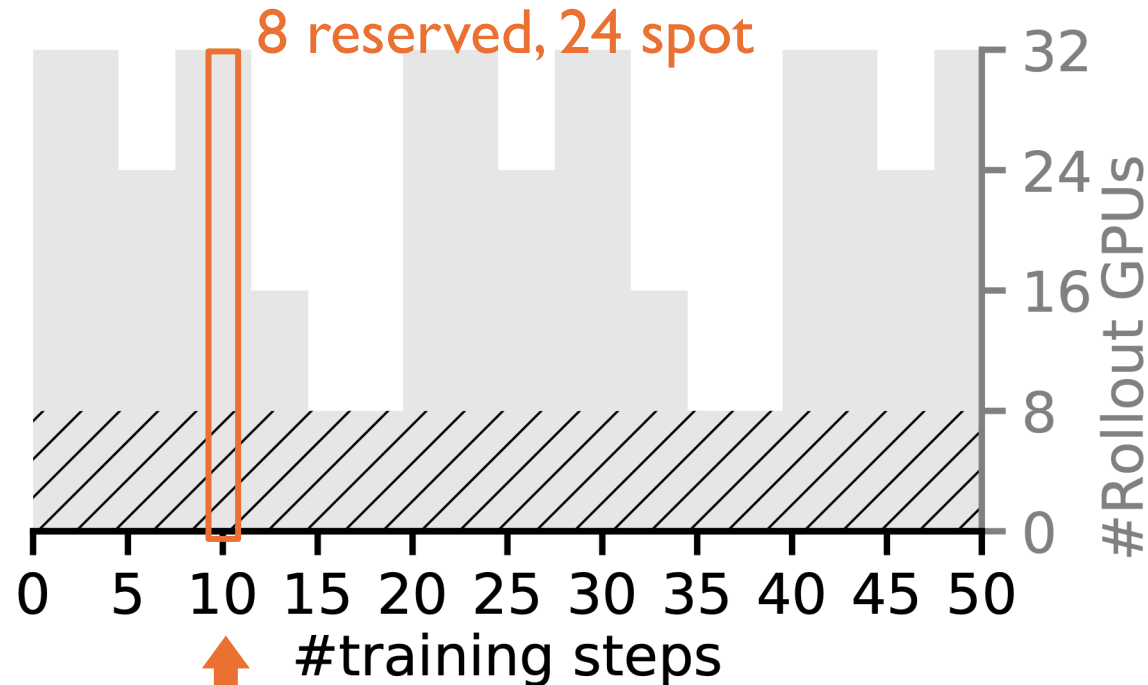
19x↓
GPU stall time

Case Study: Elastic Rollout

Workload: 260B model, 64 trainers, 8-32 rollouts (*dynamically scaled up and down*)

Baseline: UCX (*NCCL not support dynamic clusters*)

Right Axis:  Reserved GPUs  Spot GPUs (*only rollout shown*)

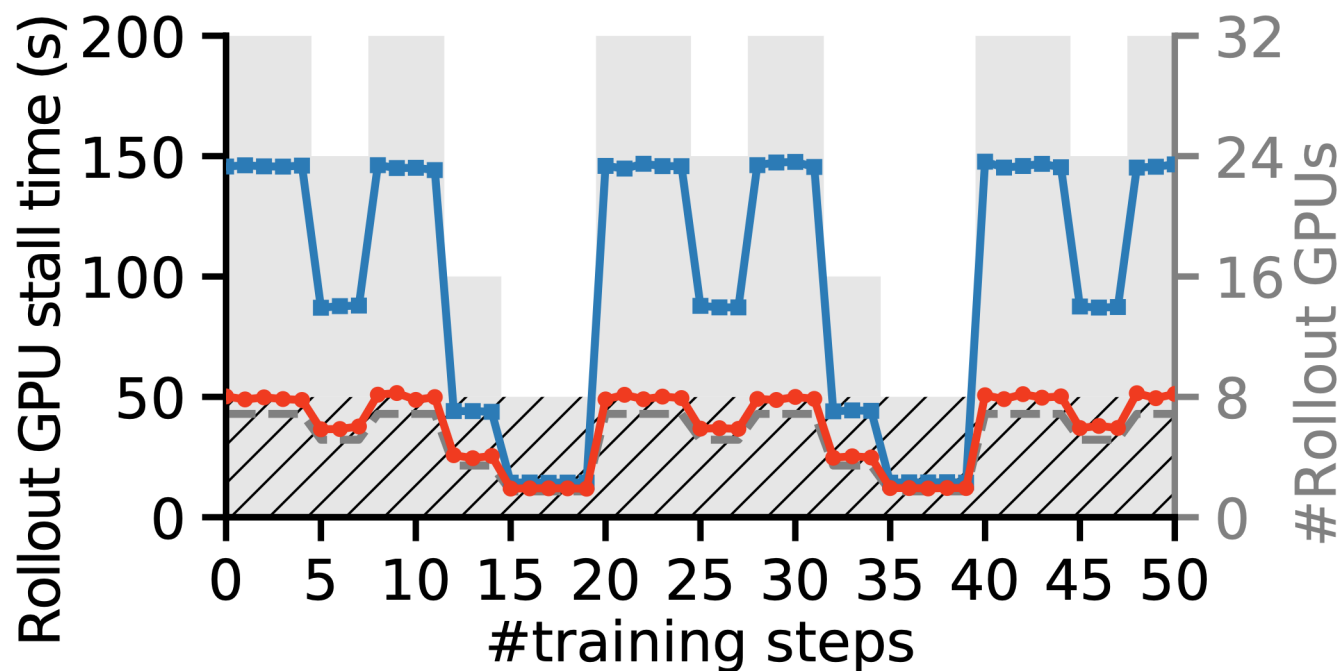


Case Study: Elastic Rollout

Workload: 260B model, 64 trainers, 8-32 rollouts (*dynamically scaled up and down*)

Baseline: UCX (*NCCL not support dynamic clusters*)

Left Axis: ■ UCX ● TensorHub --- RDMA Ideal
Right Axis: Reserved GPUs Spot GPUs (*only rollout shown*)



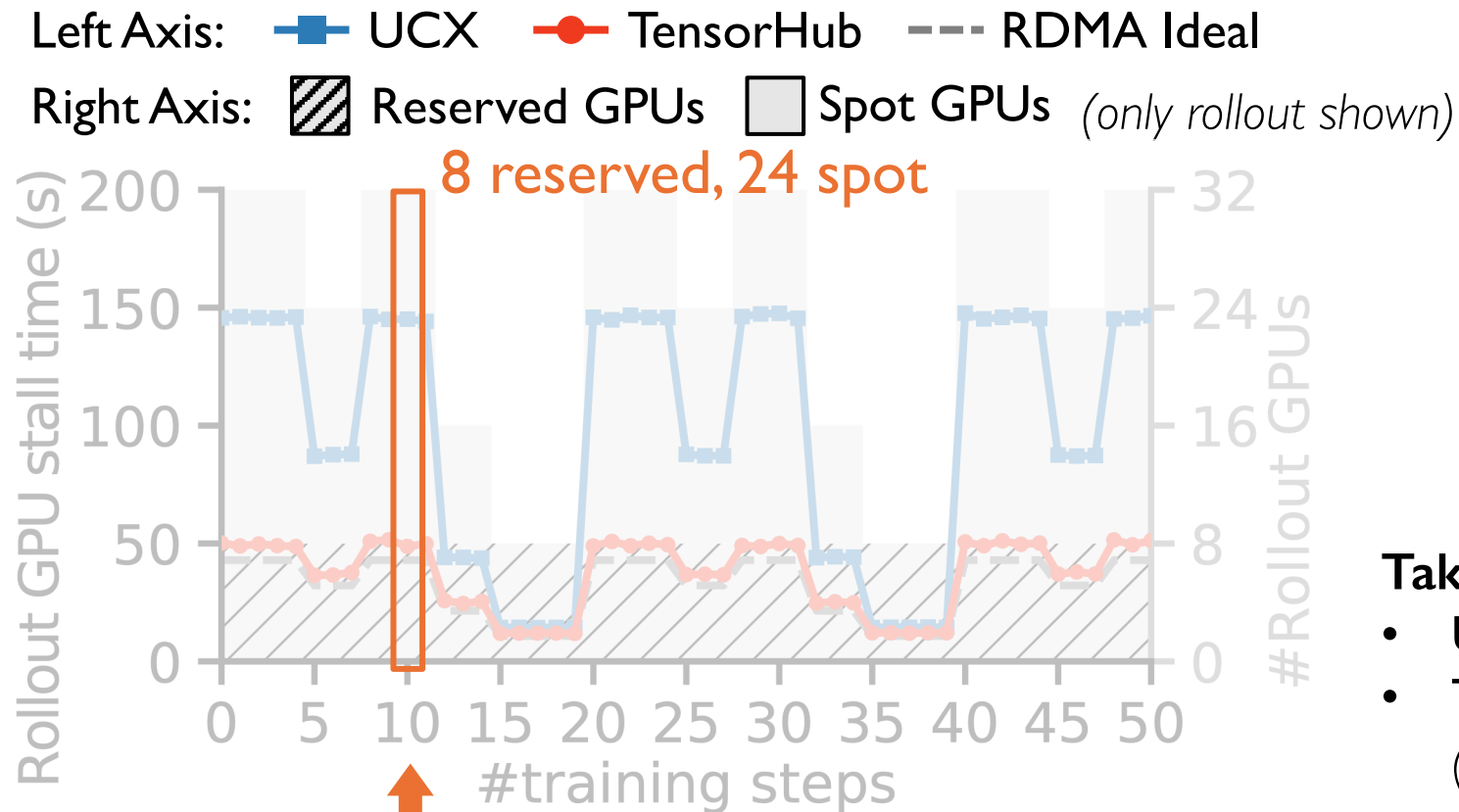
Takeaway:

- UCX suffers from contention
- TensorHub load-balanced (*close to RDMA optimal*)

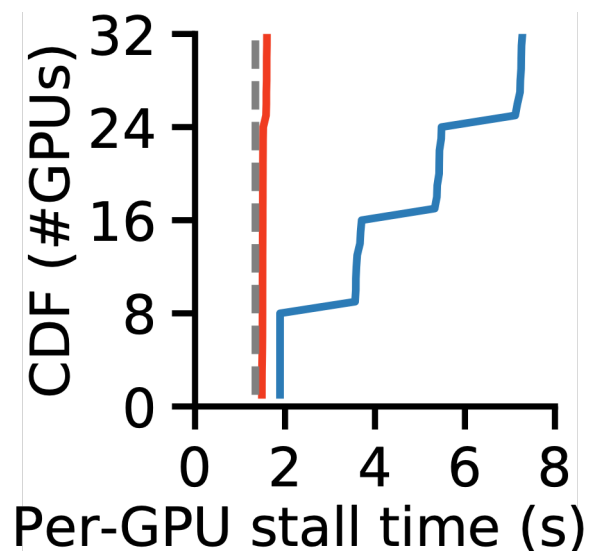
Case Study: Elastic Rollout

Workload: 260B model, 64 trainers, 8-32 rollouts (*dynamically scaled up and down*)

Baseline: UCX (*NCCL not support dynamic clusters*)



Latency CDF at step 10



Takeaway:

- UCX suffers from contention
- TensorHub load-balanced (*close to RDMA optimal*)

Conclusion

- **Reference-Oriented Storage (ROS)**

- Exploit *highly replicated, immutable, short-lived* weights
- No ownership → No extra movement/space

- **TensorHub**

- Concise versioning APIs
- Topology-aware transfer
- Model-parallel consistency
- Fault tolerance & recovery

RL systems are increasingly complex ...

Clean Abstraction → Decoupling

Usability++

Scalability++

Reliability++