



CONFERENCE

—NORTH AMERICA 2026—

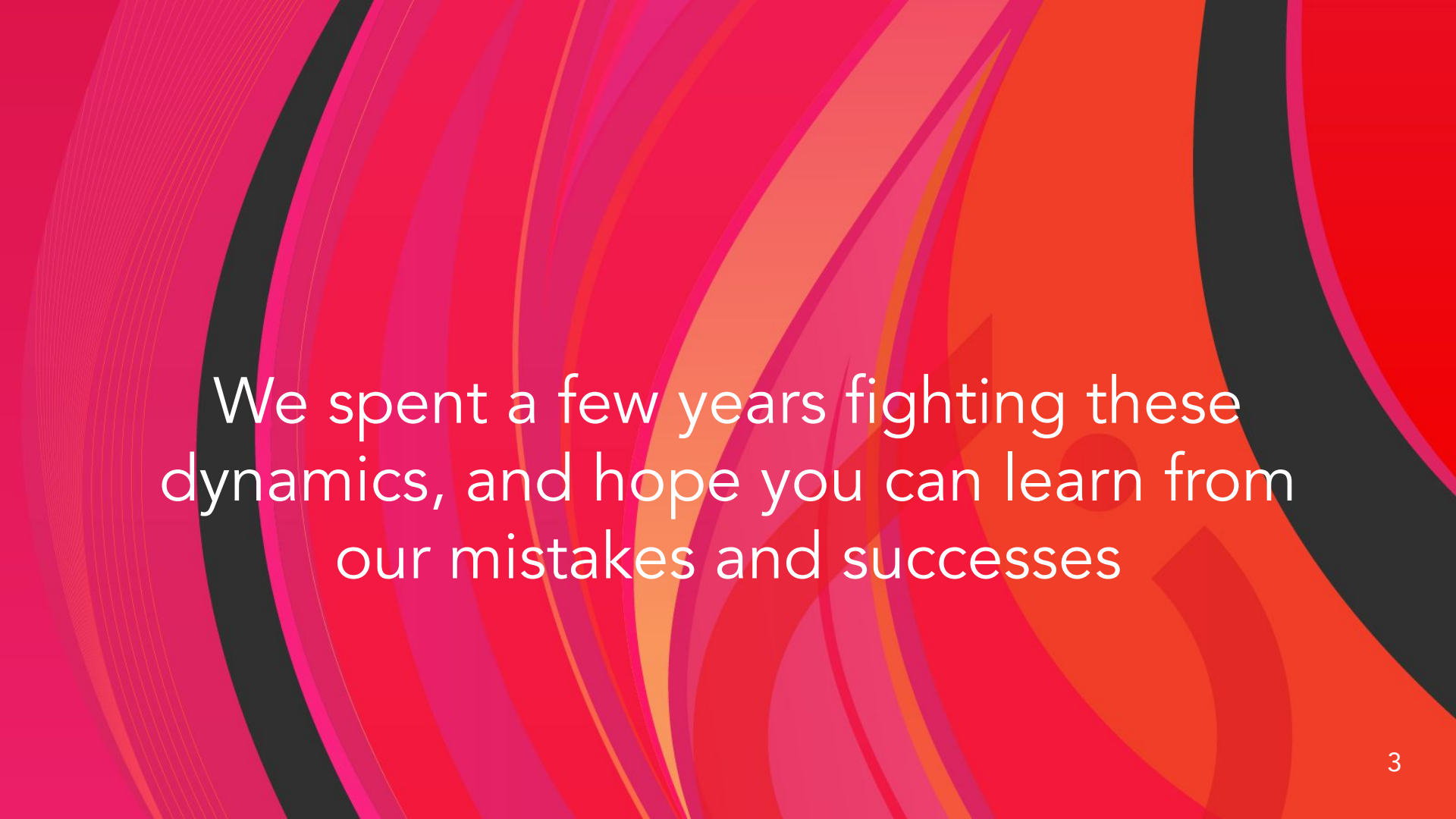
Scaling PyTorch dataloaders 10x: A case study

Ian Stenbit · Dylan Doblar · Christine Cheng

NVIDIA · Alpamayo ML Infrastructure

Let's talk about dataloaders

- Dataloading can limit training throughput
 - Large models often spend most of the step in forward / backward
 - Smaller models leave less time to prepare the next batch
- Our inputs need substantial preparation
 - Video must be decoded, and geometries turned into model tensors
 - Storing everything as ready-to-use tensors would be expensive
- Fast components don't guarantee fast training
 - File access and CPU preparation can vary from batch to batch
 - One late rank can leave other ranks waiting



We spent a few years fighting these dynamics, and hope you can learn from our mistakes and successes

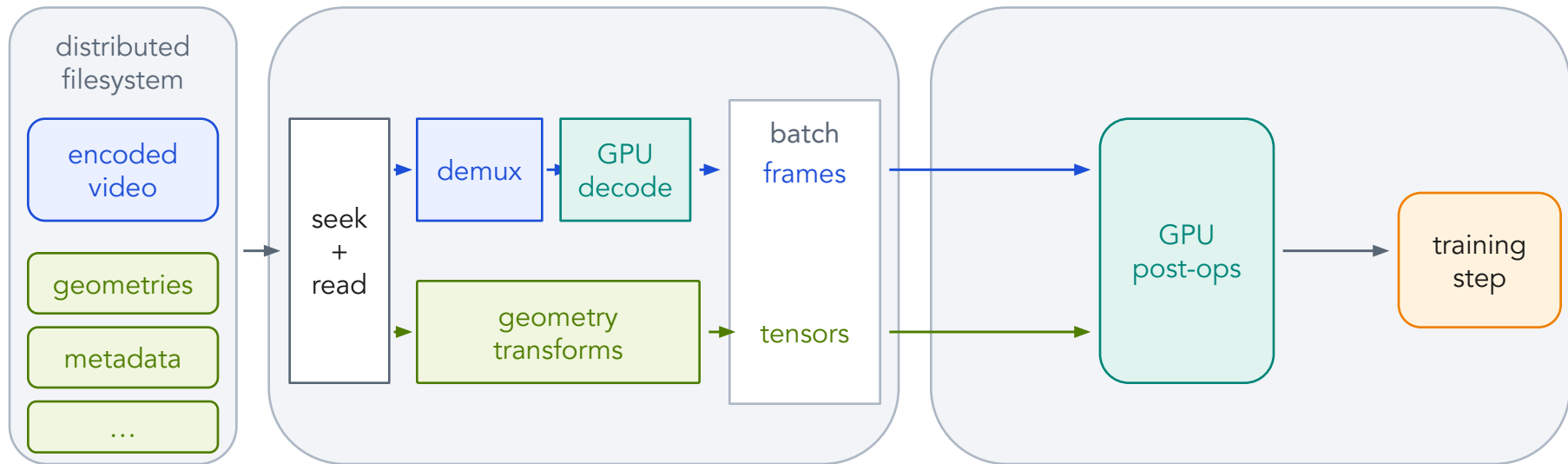
First - our problem, and how we measured
it (incorrectly)
Technical details later.

Our training workload

- a few PB of data; each storage clip has video and 4D geometry data and produces several samples
- relatively small models (~0.5–2B parameters), ~500 GPU-day -> ~50GPU-day

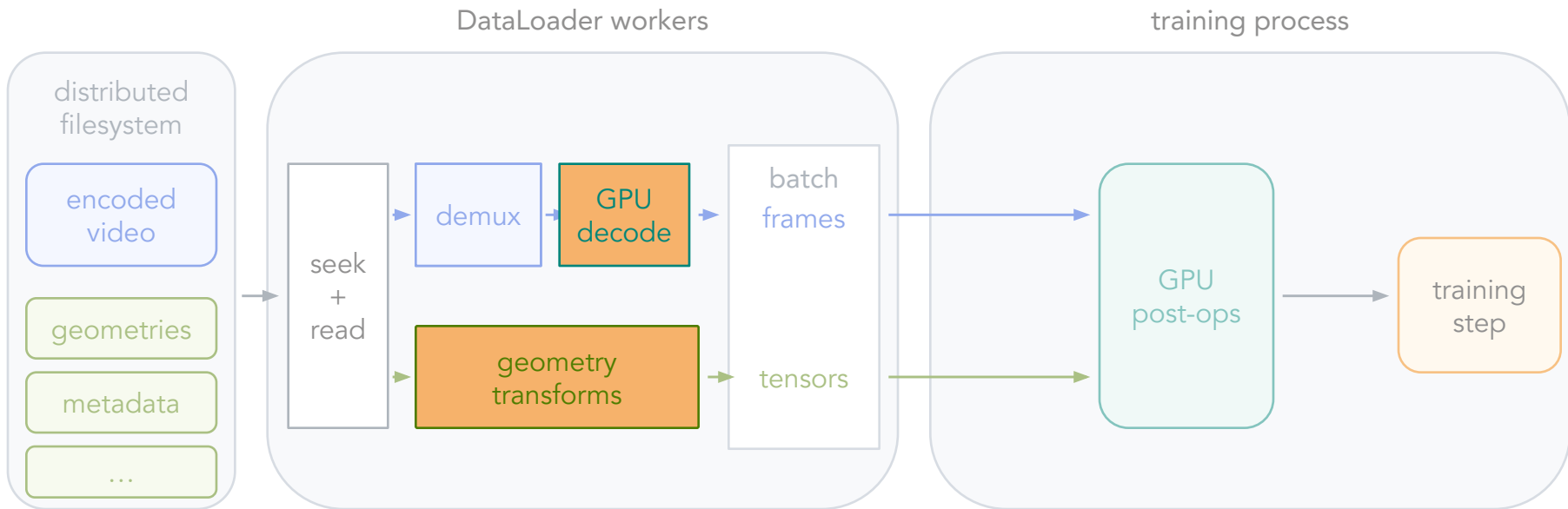
DataLoader workers

training process



Initial profiling

- Video processing and geometric transforms dominated CPU time averaged across workers
 - We calculated that we could be model-bound by optimizing these

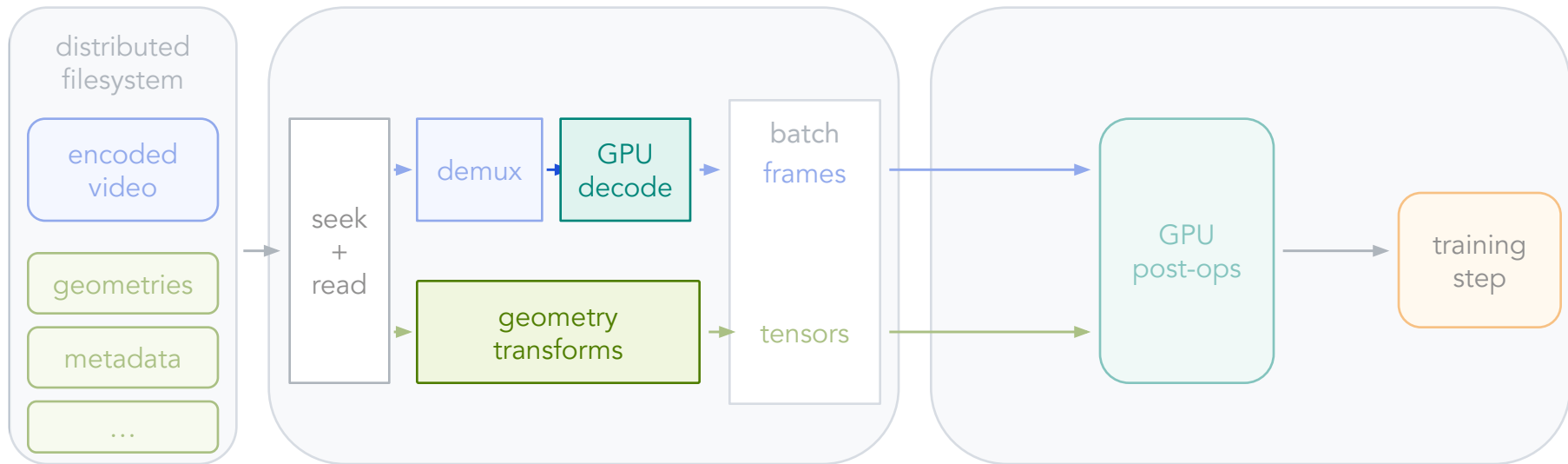


Optimizing the expensive pieces

- We optimized gpu decoding and geometric transforms enough to theoretically become model-bound
- Overall training throughput improved, but only marginally

DataLoader workers

training process

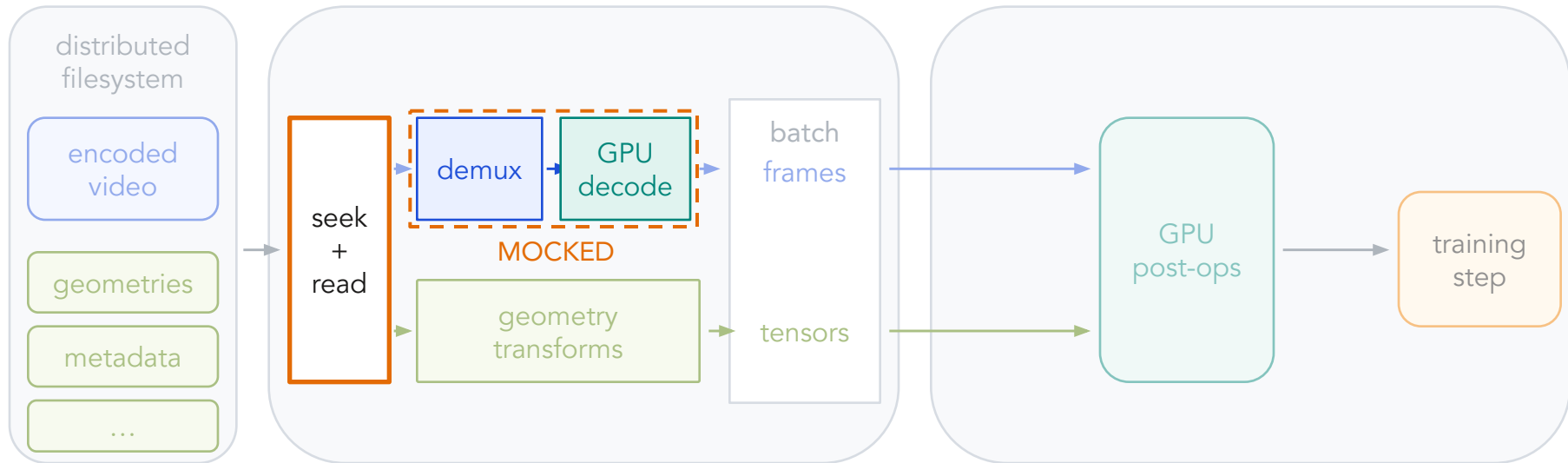


Ablating the dataloader

- Mock out individual pieces of the dataloading pipeline, one at a time
 - Result: seek+read is our main culprit, despite using <20% of total CPU time... why?

DataLoader workers

training process

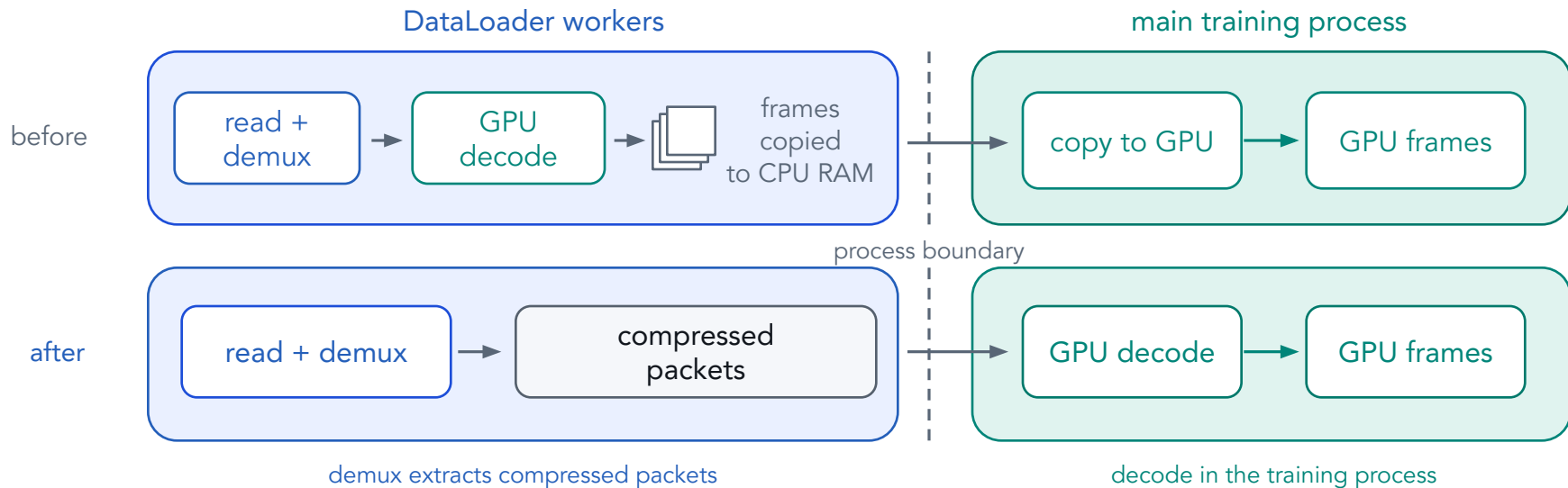


Now, the technical details we promised

How did we speed up video and geometry?
How did we keep I/O from holding up training?

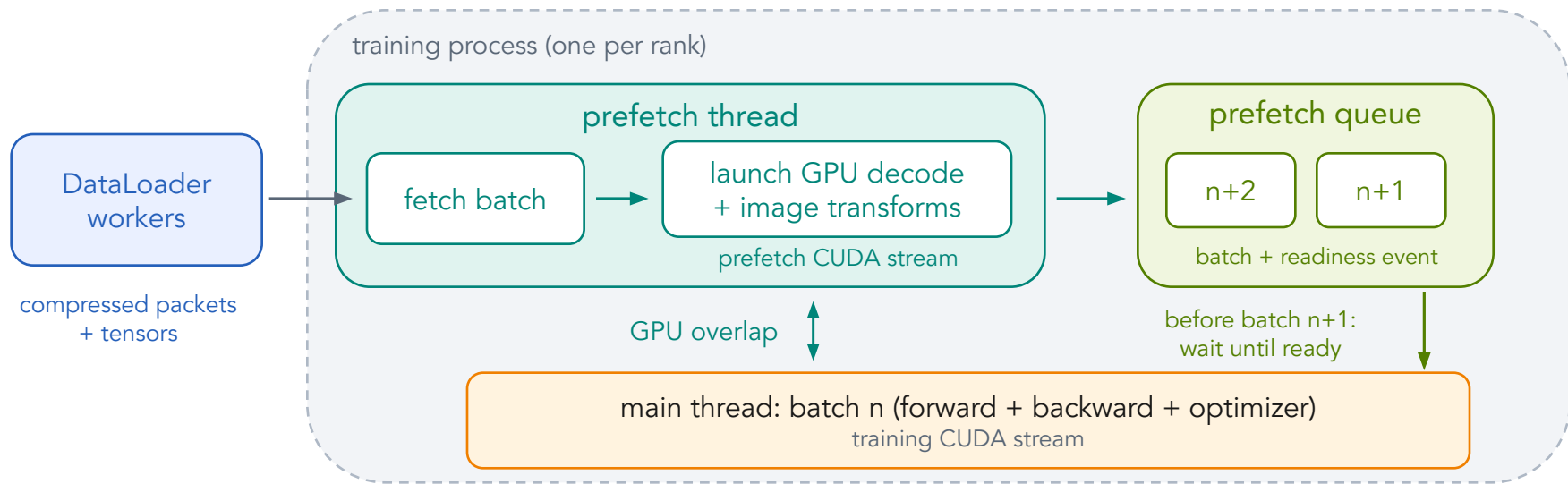
Keep video compressed across the worker boundary

transfer compressed packets between processes instead of full frames



Prefetch batches on a background thread

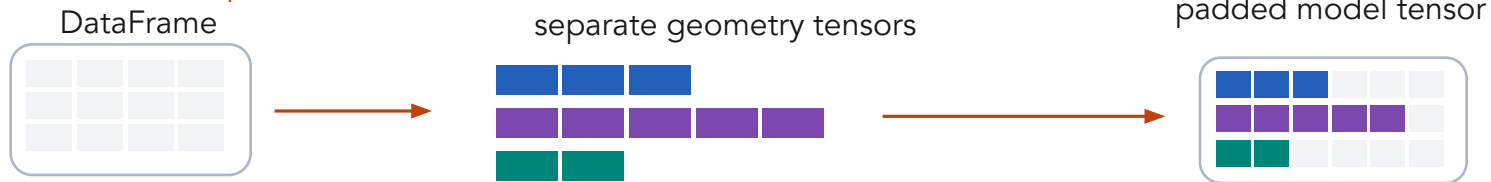
fetch and prepare upcoming batches while training runs



previously: one-batch lookahead driven by the training loop

Construct geometry tensors directly

before: convert, split, then assemble



after: transform the flat coordinates and fill directly
flat coordinate values



video decoding and geometric transforms got much faster, but training throughput improved only modestly...

Reduce file opens and small reads

1. match the reader to our access pattern

off-the-shelf reader

18+ small read calls / file

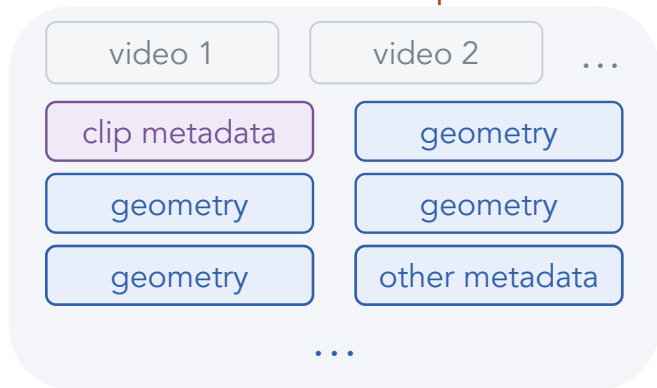


custom reader

read whole packed file into RAM
parse + access in memory

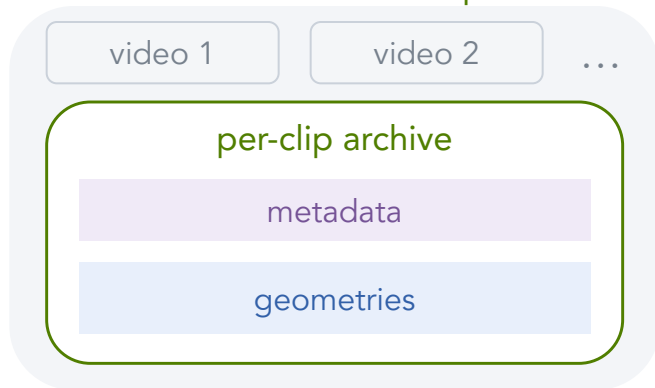
2. fewer files per clip

before: one clip



pack per clip

after: the same clip



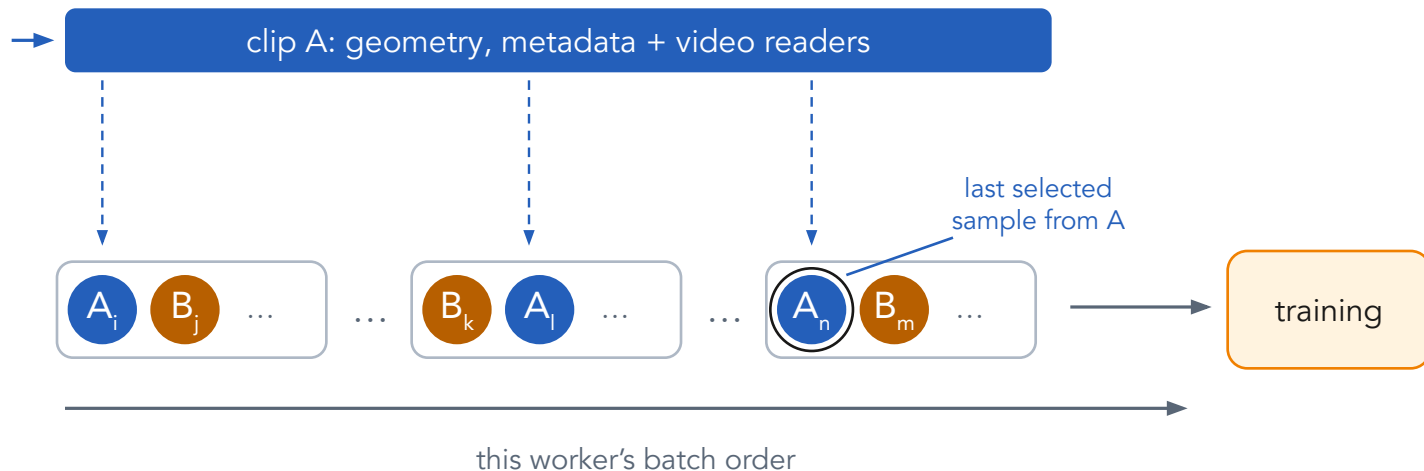
Reuse clip data with I/O-aware sampling

more active clips $\rightarrow$ more mixing, more worker memory

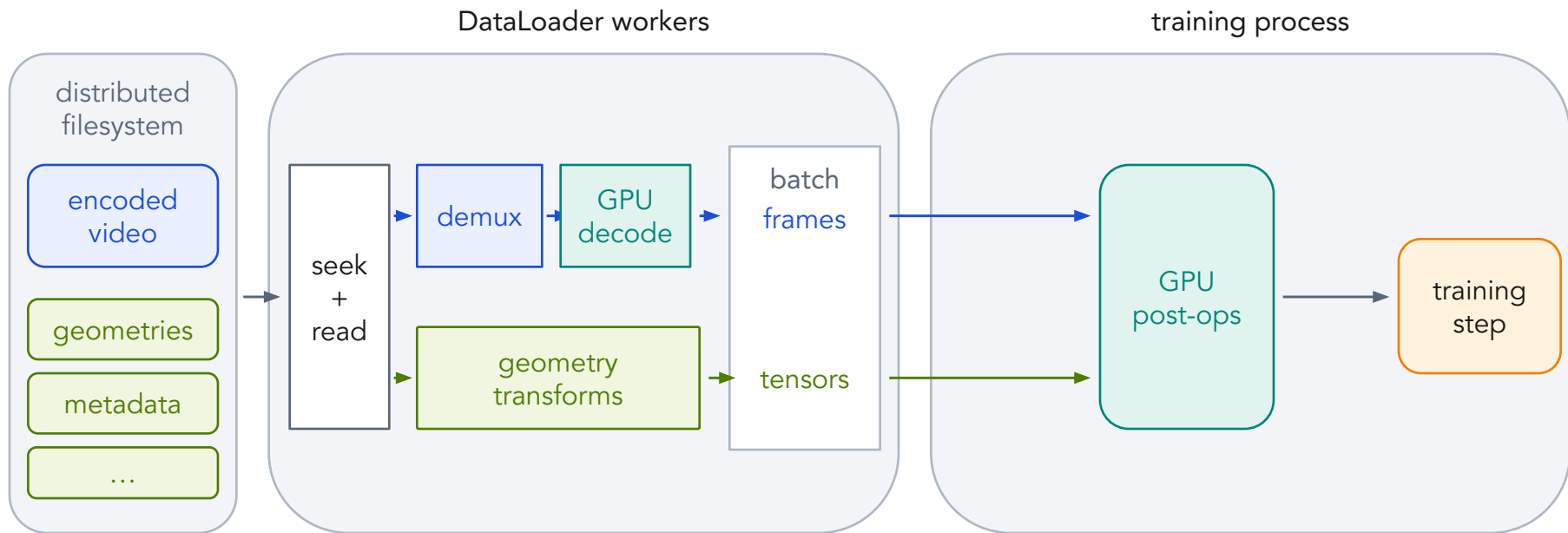
kept in worker memory

first use:
load data +
open video readers

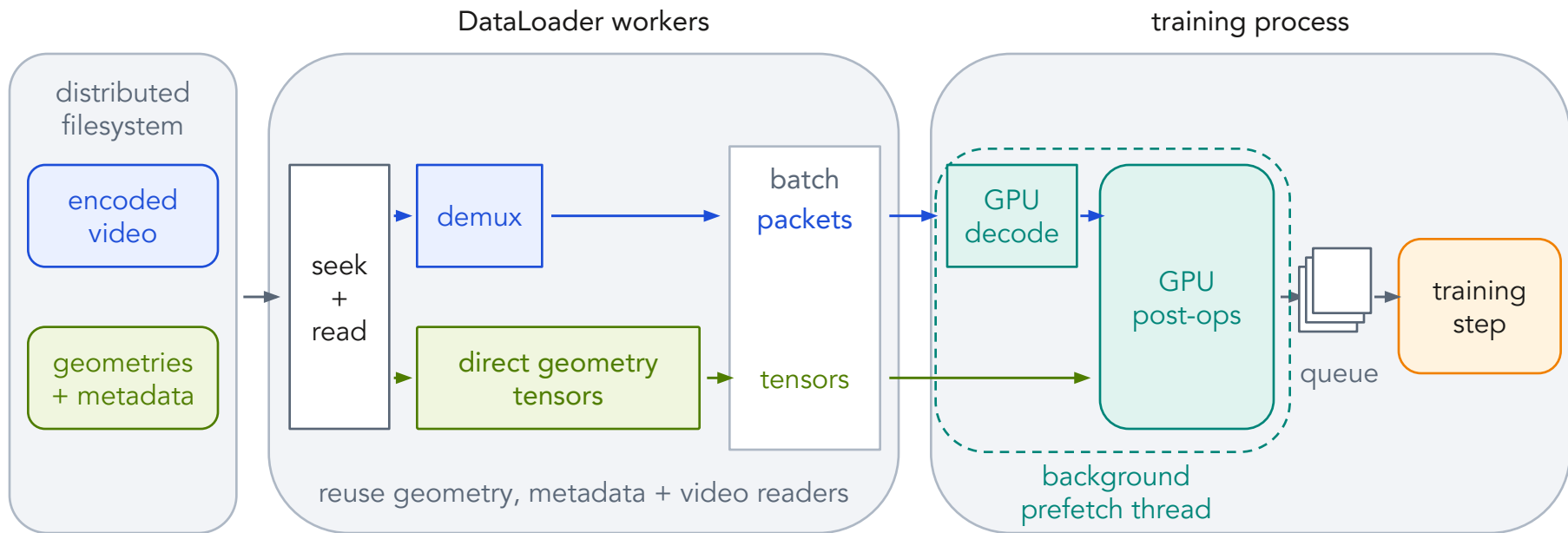
worker batches



The original dataloader



The redesigned dataloader



From pipeline changes to training results

How much faster did training get?
What did we learn along the way?

Results: training throughput grew about 10×

Training throughput improved about 10× over two years as we changed the system and its configurations. We didn't measure each change's contribution separately.

1×  ≈10×

PIPELINE CHANGES BEHIND THE PRODUCTION TRAJECTORY

1

less
repeated work

2

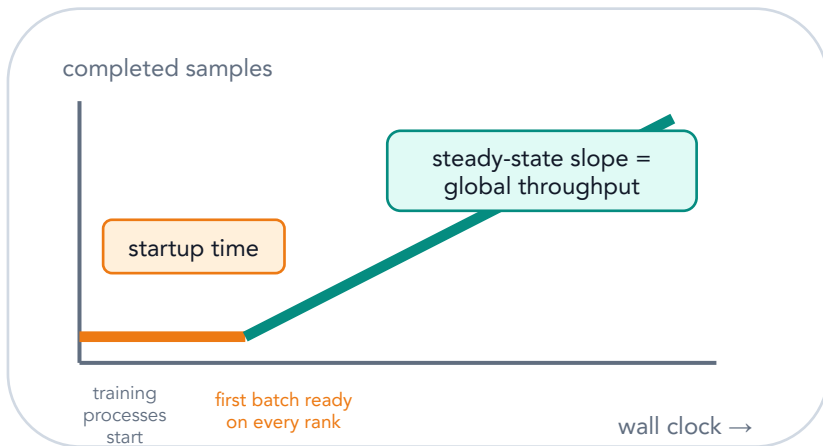
reuse
clip data

3

overlap preparation
with training

Lesson 1: measure the training job

Track startup time and sustained global samples/sec under comparable conditions.



STARTUP MATTERS AT SCALE

Device failures can trigger restarts.

Each restart repeats initialization.

Our result:

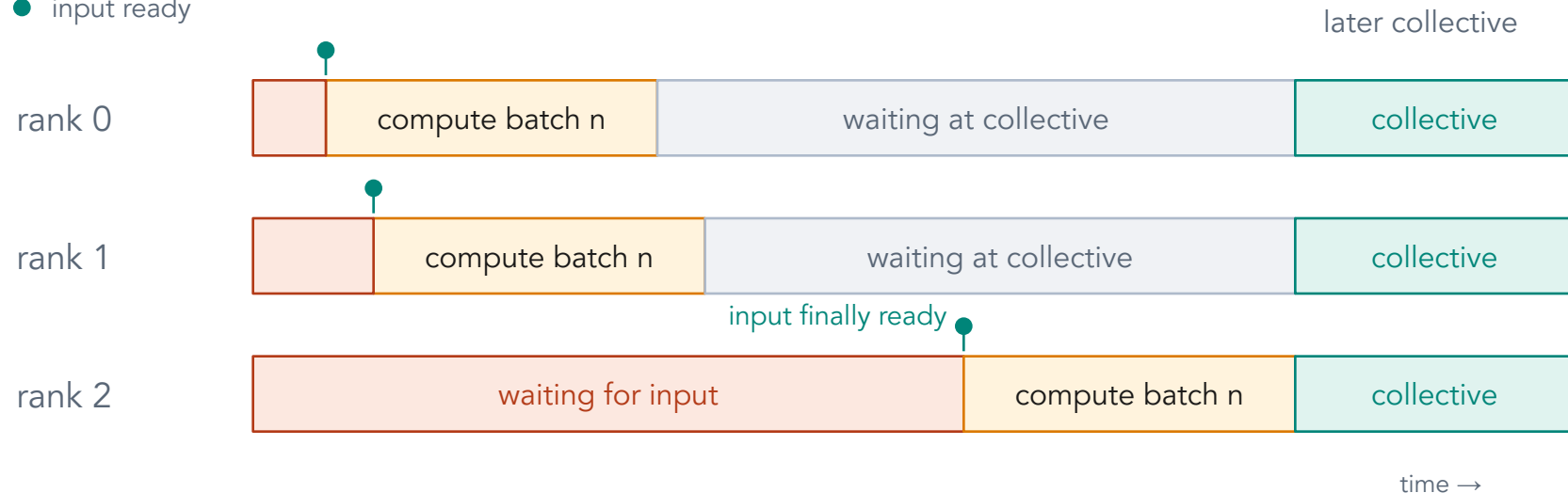
~10× shorter startup (1hr to <5min)
cluster idle from 7% to 1%

Use one-stage ablations to find the limit,
then verify the fix in real training.

Lesson 2: trace the delay that reaches training

A late input on one rank can leave other ranks waiting at a collective.

● input ready



Align step, rank, and batch IDs; record when input is usable by training.

Lesson 3: match the fix to the bottleneck

The limiting stage can change with the pipeline, model, storage, or rank count.

locate the limit → choose a fix → check the gain and cost → repeat

IF STARTUP DOMINATES

Build the sample manifest
in a CPU-only job
before allocating GPUs.

Preparation still takes time;
measure total cost, too.

IF FILE ACCESS DOMINATES

Pack data per clip.
Reuse geometry, metadata,
and video readers across
selected samples.

Reuse depends on sampling
and available worker RAM.

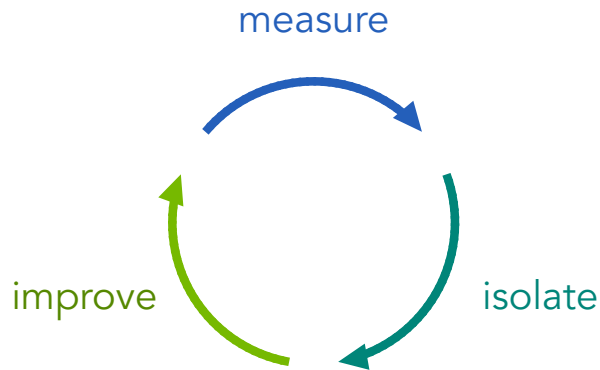
IF GPU PREP DOMINATES

Pass compressed packets.
Overlap GPU preparation
with training.

More lookahead uses memory;
a slow producer still limits.

Measure the gain in your training job

- Use ablations to find the bottleneck
- Measure startup + training throughput
- Compare batch readiness across ranks
- The bottleneck changes as you optimize
- The right fix depends on the workload



Q&A

come chat with us at **NVIDIA booth D1!**

Team posters in the Community Expo

From MP4 to Tensor

Kyle Huang · Mihai Aldén

TorchInductor artifact reuse

Christine Cheng · Kwanghoon An

StageFrontier

Boram Yoon · Aaron Meng · Ville Kallioniemi

Production FP8 Training

Wei Chen · Iain Weissburg