Imperial College London

LSDS
Large-Scale Data & Systems Group

What Can Machine Learning Systems Learn From Data Analytics?

Peter Pietzuch

Imperial College London

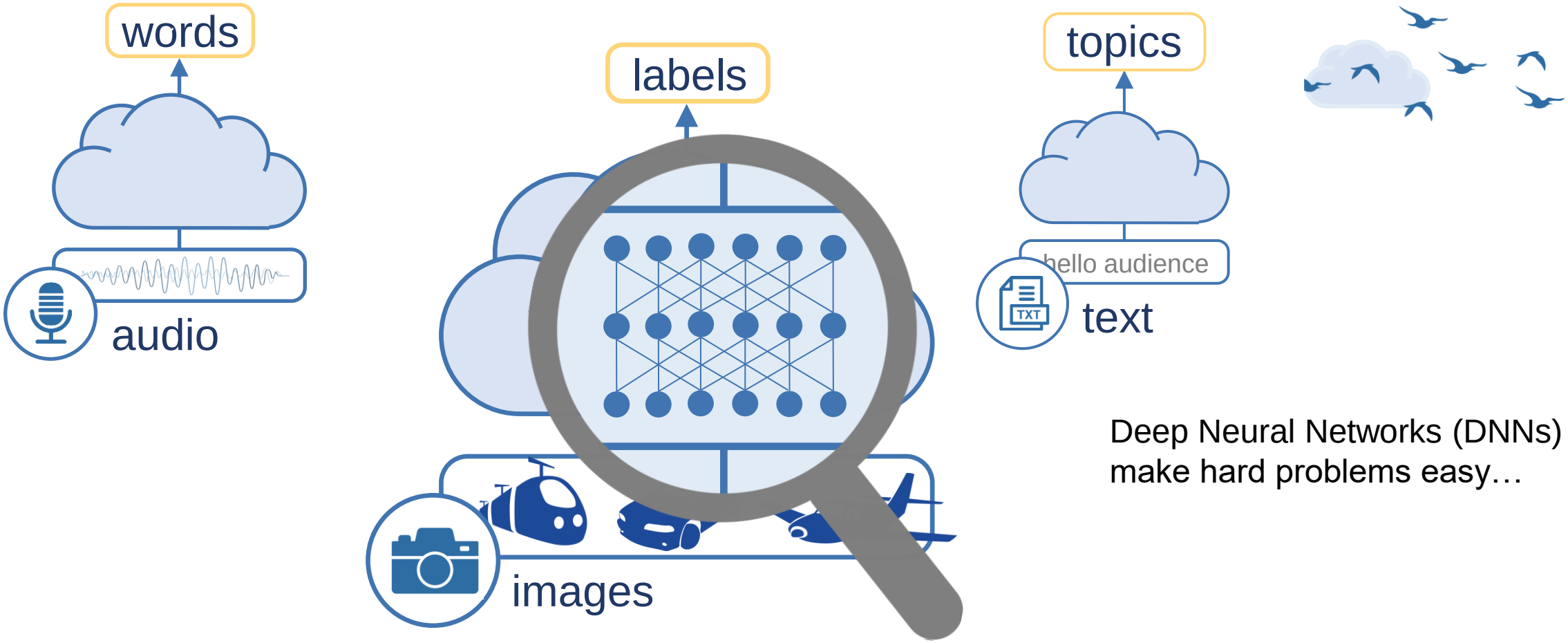
http://lsds.doc.ic.ac.uk
<prp@imperial.ac.uk>

Joint work with Luo Mai, Guo Li, Marcel Wagenländer, Konstantinos Fertakis, Andrei-Octavian Brabete

Huawei Global Software Technology Summit – July 2022

Deep Learning Is Eating The World

Revolutionised **vision**, **speech recognition**, **natural language processing**, …



Deep Neural Networks (DNNs) make hard problems easy…

DALL E Mini Model for Artistic Image Generation

AI drawing from text input: “Hamster skateboarding in space”

Go to https://www.craiyon.com

Training Deep Neural Networks (DNNs)

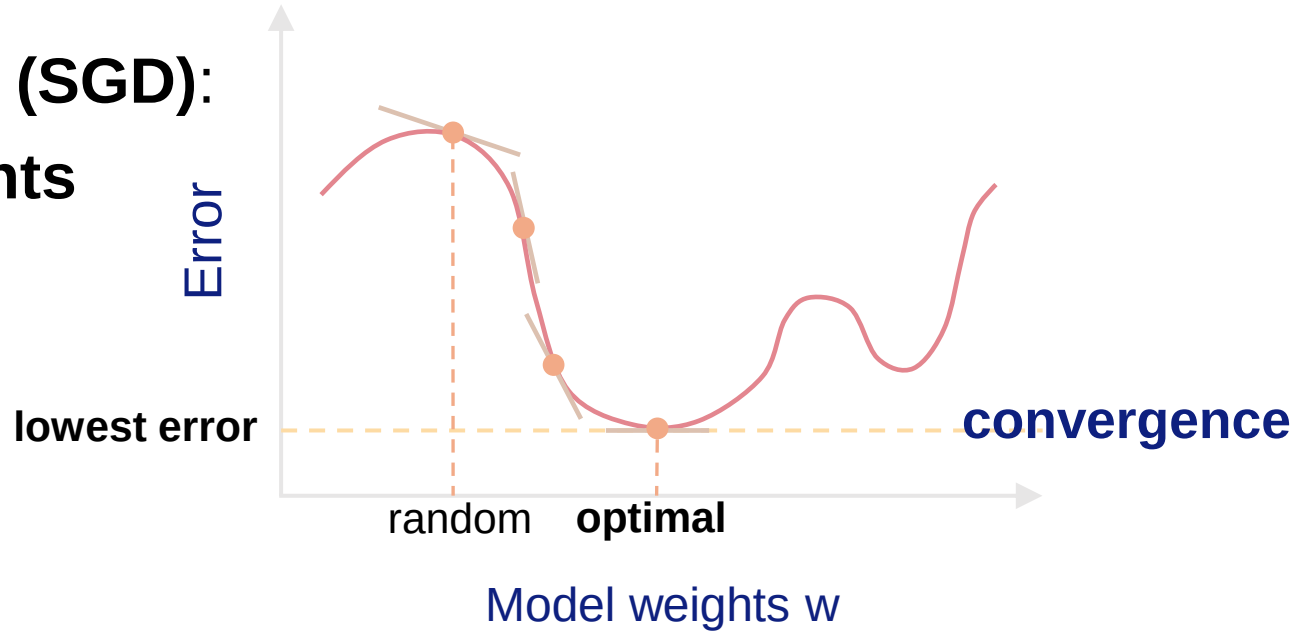
DNN models are **trained** by giving **examples** (instead of programming)

Obtain DNN model that minimises **classification error**

When DNN output is wrong, tweak its parameters

Use **Stochastic Gradient Descent (SGD)**:

1. Begin with **random model weights**
2. Consider **batch** of training data
3. Iteratively calculate **gradients** & update **model weights**

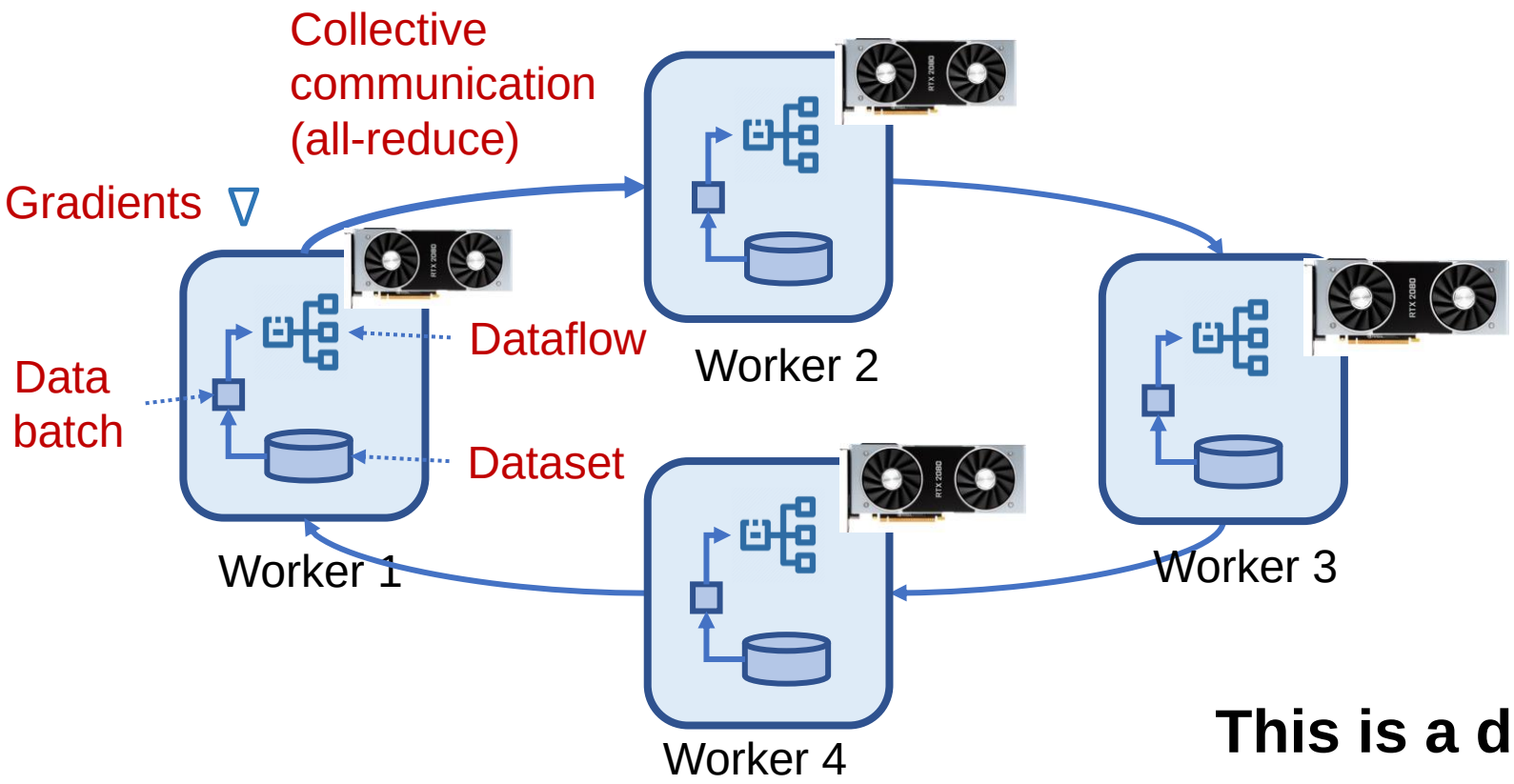


Deep Learning on GPUs

GPUs are good at parallelising gradient computation

Distributed DNN Training Systems

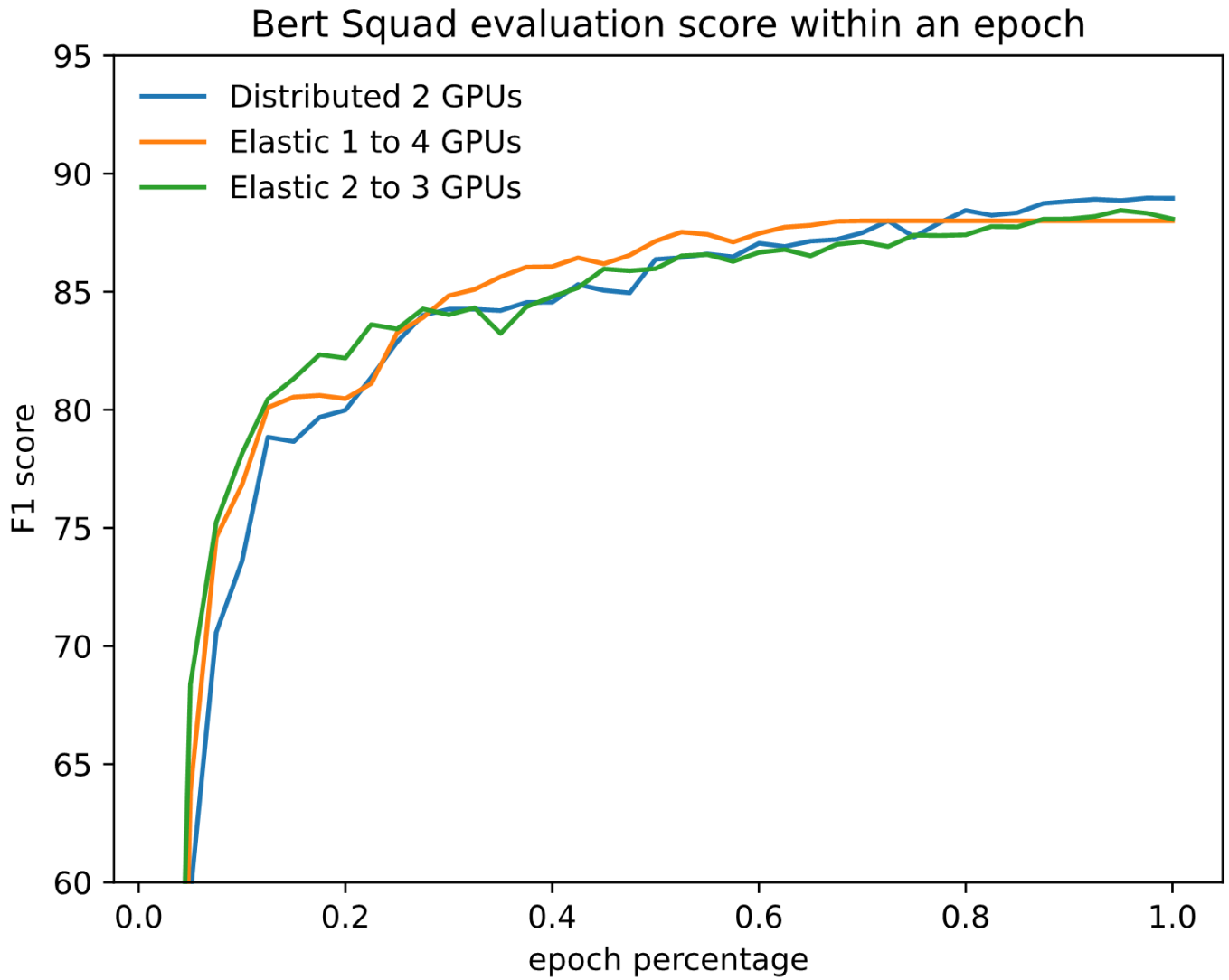
Exploiting **data parallelism** during training



This is a data system!

Using Different GPU Numbers for Training

Convergence curve for BERT NLP model depends on GPU number

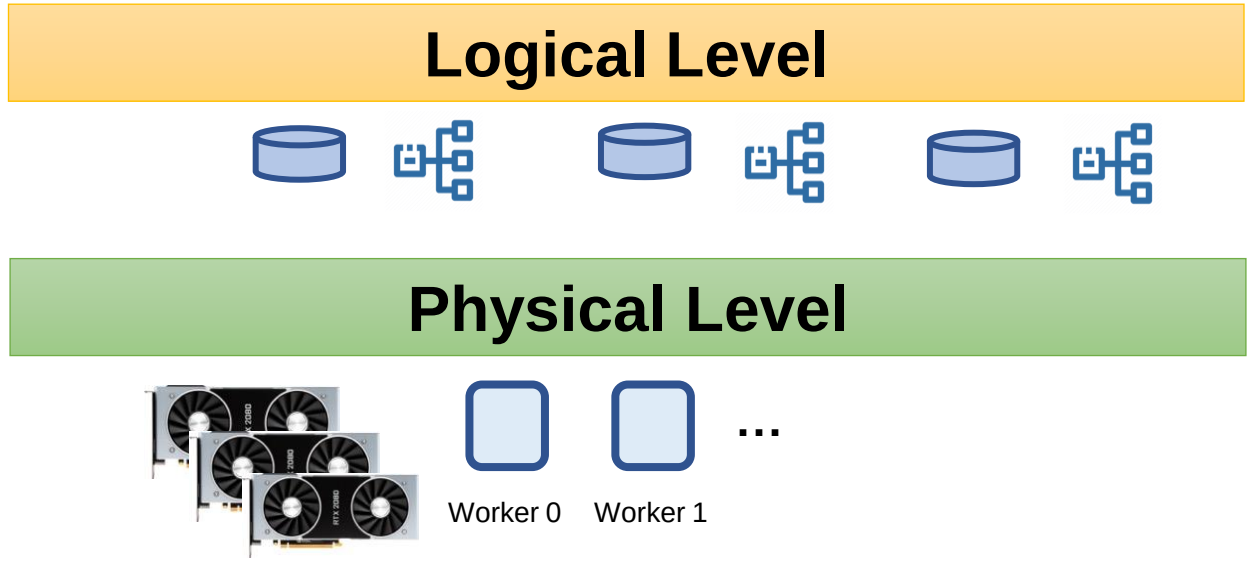


But What About Data Abstractions in DBMS?

Query/application 1

Query/application 2

Query/application 3



DNN Training Requires Many Parameters

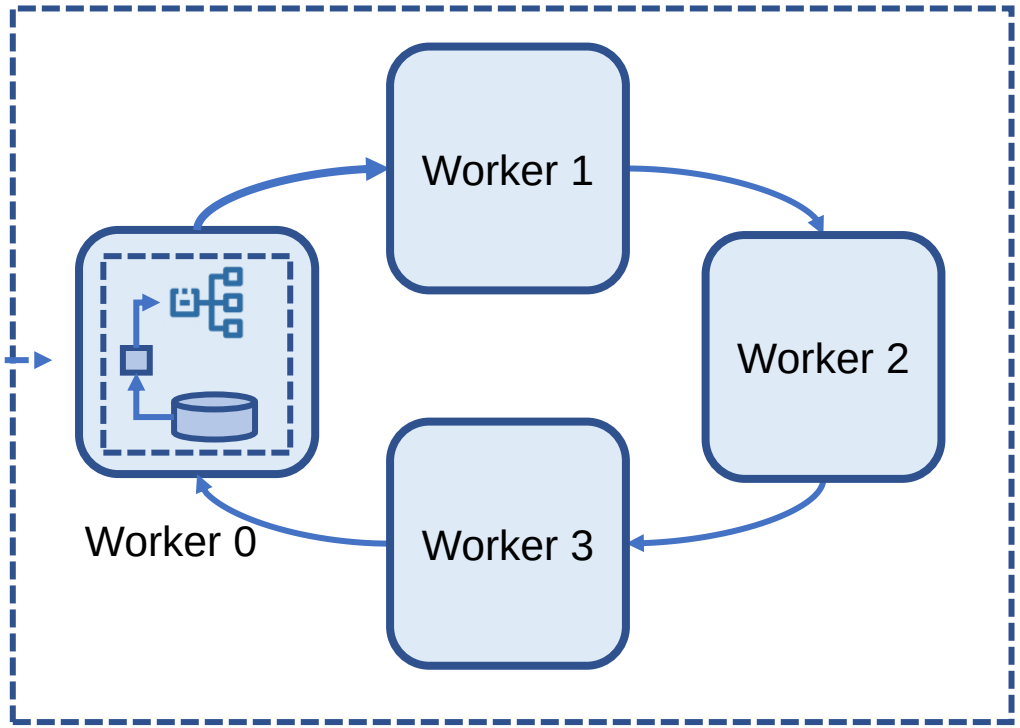
Users tune **hyper-parameters** and **system parameters** to optimise **time-to-accuracy**

Hyper-parameters

- Batch size
- Learning rate
- …

Small batch or large batch size?

Statistical Efficiency



System parameters

- Number of workers
- Communication topology
- …

Ring or binary-tree?

Hardware Efficiency

Tuning Parameters During Training

Empirical parameter tuning	**Issue**
“Change batch size at epoch 30, 60 and 90 when training with ImageNet.” [1]	Dataset-specific
“Linearly scale the learning rate with the #workers when training ResNet models.” [2]	Model-specific
“Set the topology to a ring by default.” [3]	Cluster-specific

[1] Dynamic Mini-batch SGD for Elastic Distributed Training: Learning in the Limbo of Resources, 2020
[2] Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour, 2018
[3] Horovod: fast and easy distributed deep learning in TensorFlow, 2018

System Parameters: Elastic Training

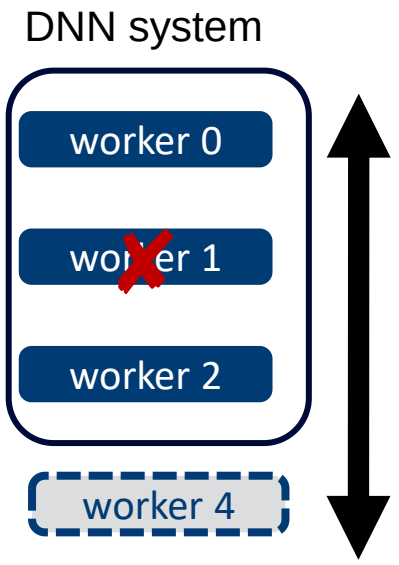
Distributed DNN training is **resource-intensive**

Elastic training: use **cheapest** hardware resources (GPUs) **when available**

Example: Training Megatron-LM[3]

- Training of BERT-like model
- 512 NVIDIA V100 GPUs
- One epoch (68,507 iterations) takes 2.1 days

Cost on Azure: $92,613



GPU workers added/removed during training

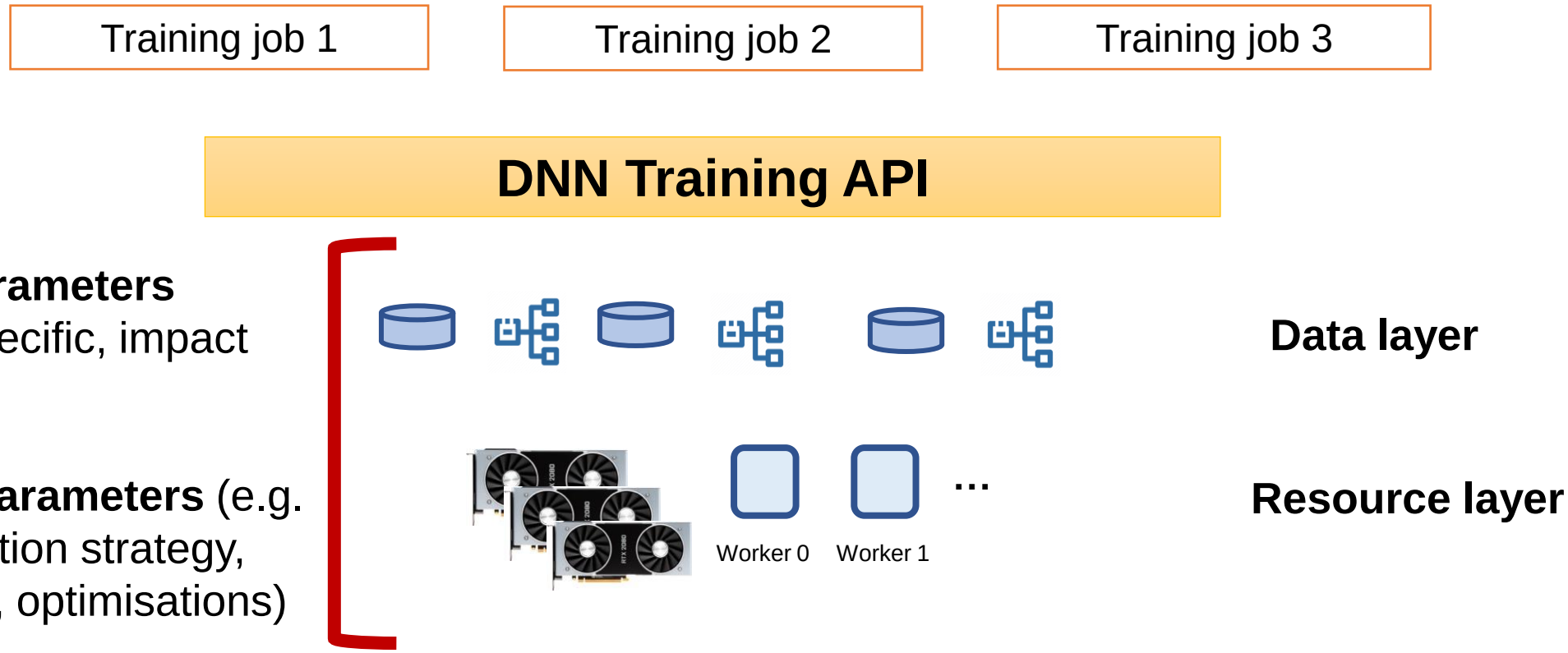
→ Must cope with changing GPU devices

→ Must adapt communication topologies

→ Must ensure consistency of training process

[3]Shoeybi, Mohammad, et al. Megatron-LM: Training Multi-billion Parameter Language Models using GPU Model Parallelism, 2017

Current Abstractions in DNN Training Systems

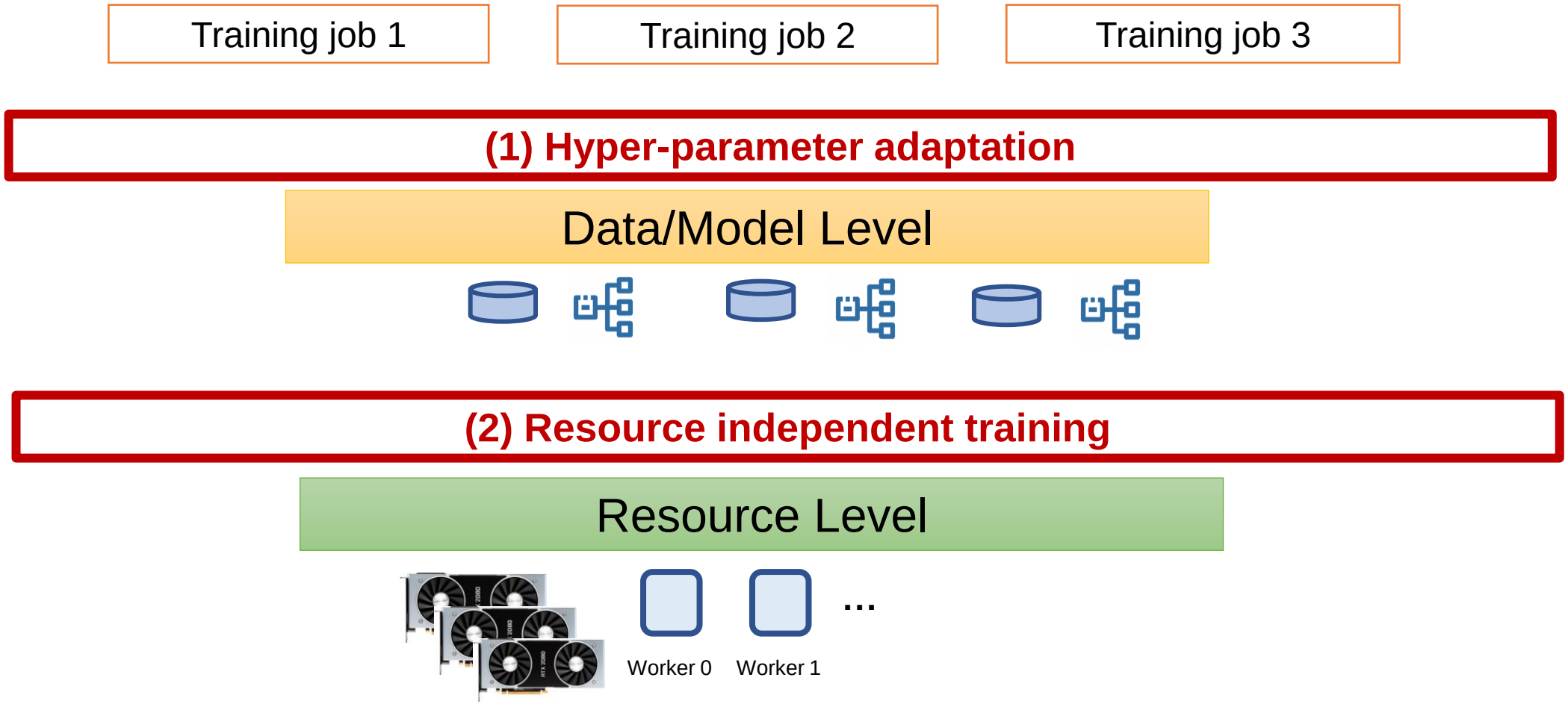


(1) Hyper-parameters (model-specific, impact accuracy)

(2) System parameters (e.g. parallelisation strategy, resources, optimisations)

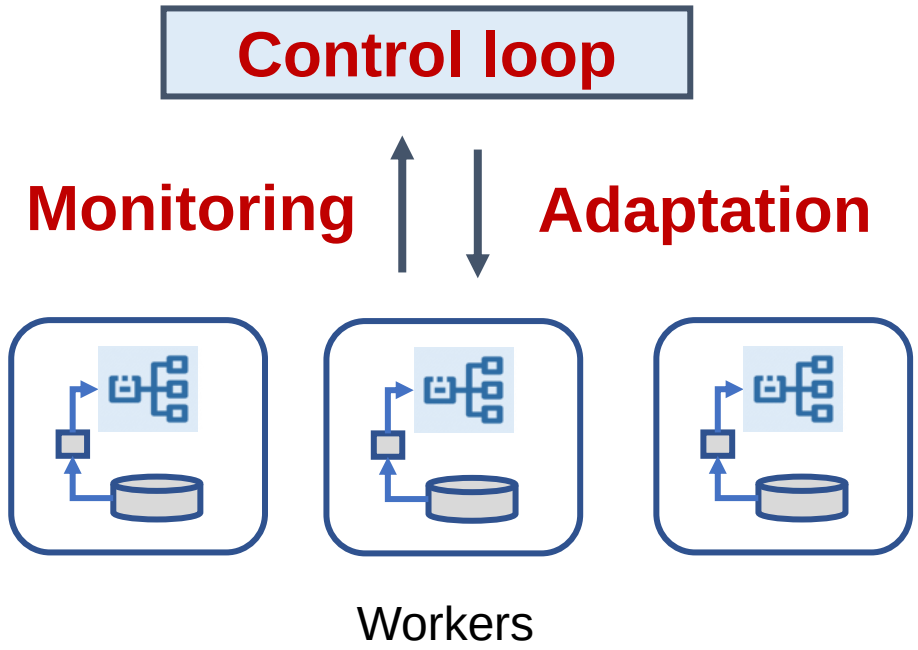
➔ Hyper- and System parameters couple Data and Resource layers

New Abstractions in DNN Training Systems



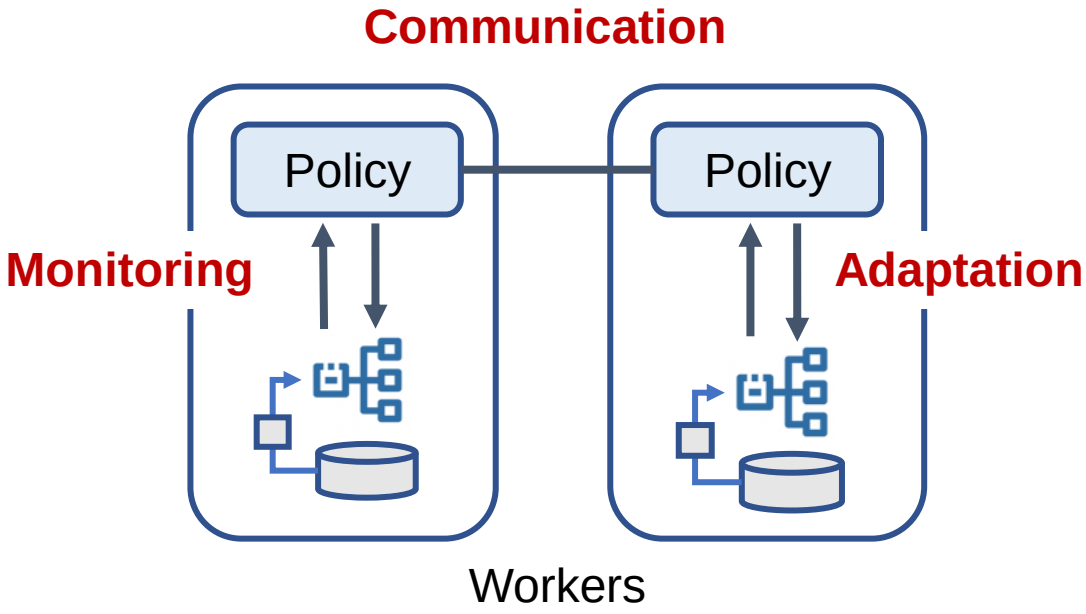
1. Hyper-Parameter Adaptation

Express Adaptation as Control Loop



Control loop monitors workers and uses monitored metrics to change parameters

Adaptation Policies



Write adaptation policies using **expressive API functions:**

Monitoring	Communication	Adaptation
• `grad_noise_scale` • `grad_variance` • …	• `allreduce` • `broadcast` • …	• `resize` • `set_tree` • …

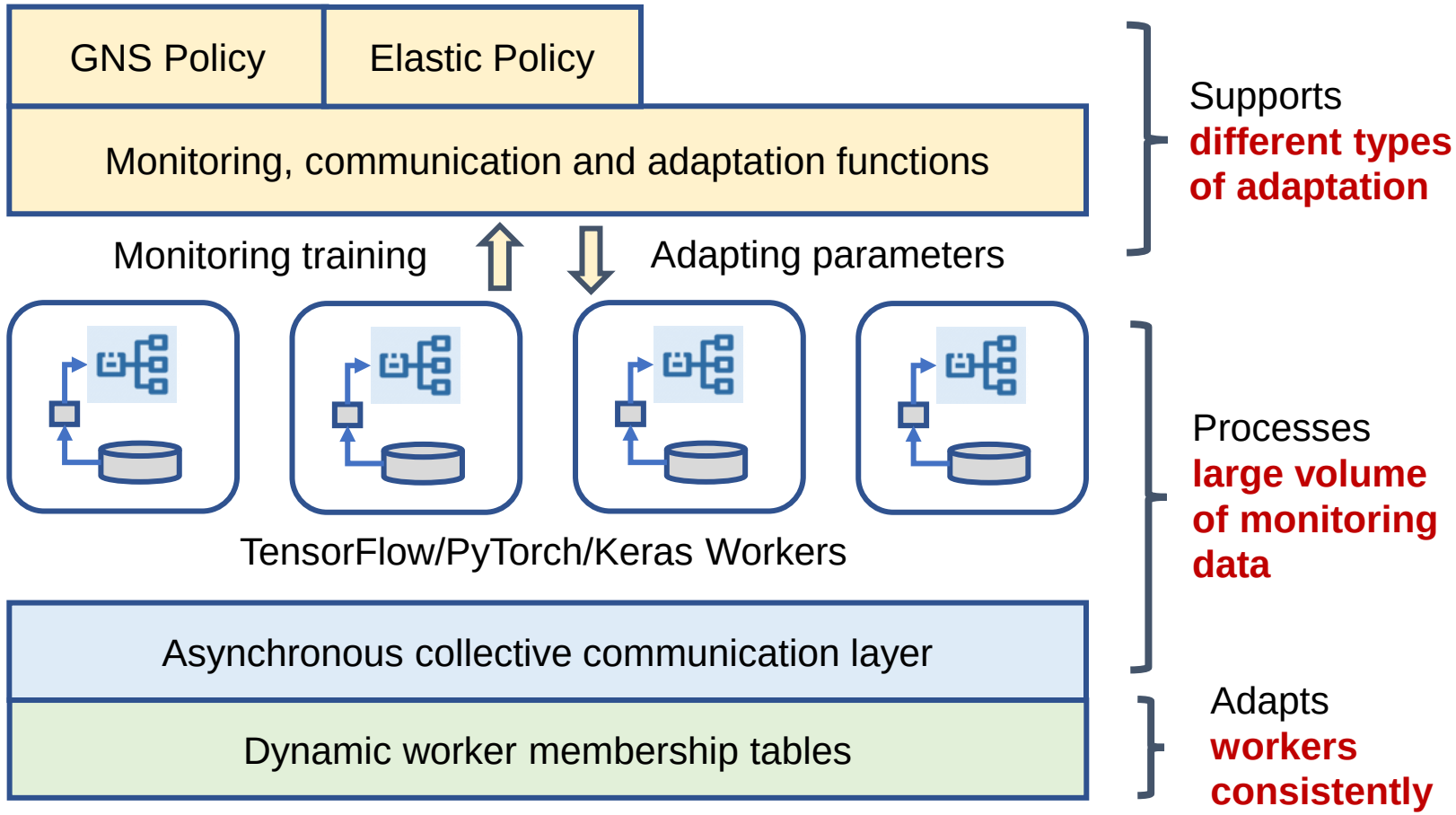
KungFu: Adaptive distributed DNN Training

Key ideas:

Supporting adaptation policies

Monitoring inside dataflow graph

Distributing parameter updates



Example: Adaptation Policy for Batch Size

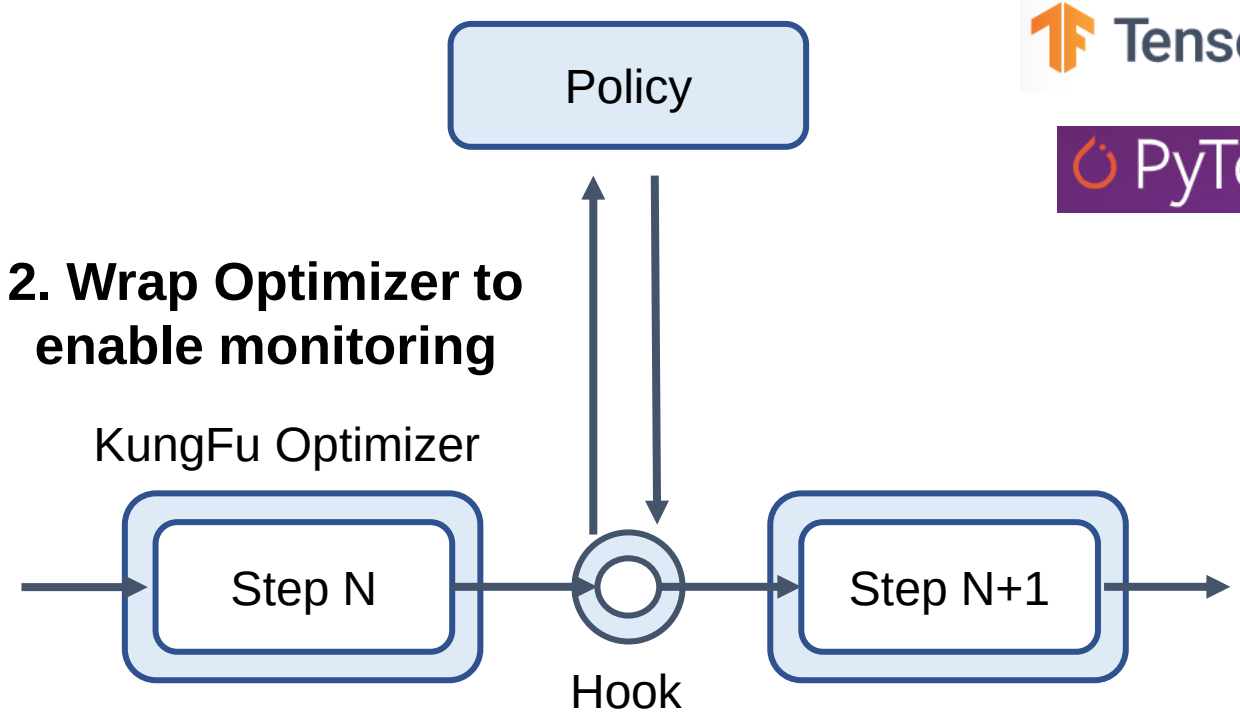
1. Adaptation logic in policy

```
import kungfu as kf

class GNSPolicy(kf.BasePolicy)
  def after_step(self):
    gns = kf.grad_noise_scale()
    avg = kf.allreduce(gns, `avg`)
    if avg > self.prev:
      kf.resize(kf.size() + 1)
```

```
opt = SGDOptimizer(…)
opt = kf.Optimizer(opt)
```

```
hook = kf.Hook(GNSPolicy(…))
model, data = …
model.train(data, opt, hook)
```

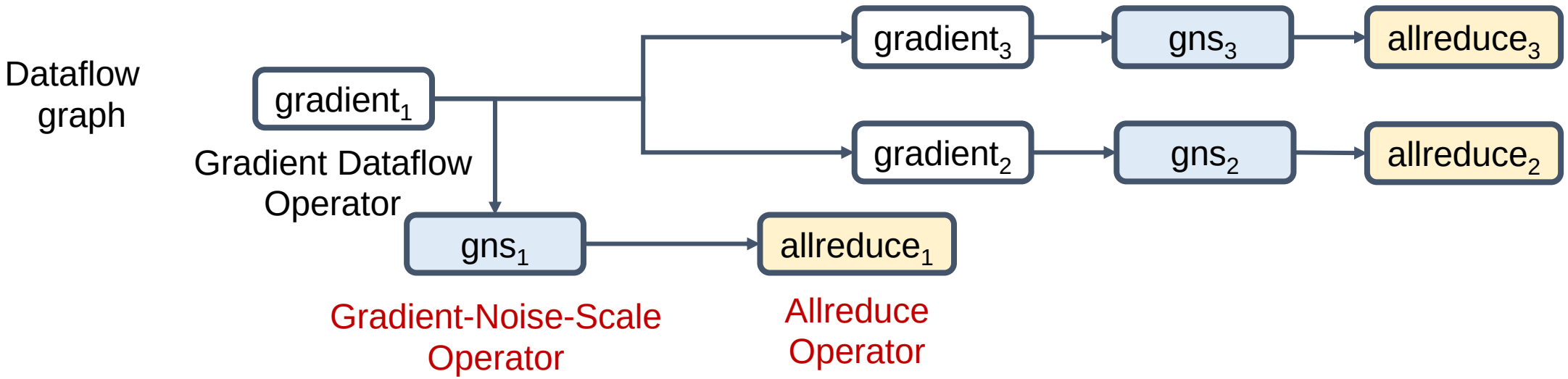


3. KungFu Hooks add policy

Efficient Monitoring During Training

Problem: High monitoring cost reduces adaptation benefit

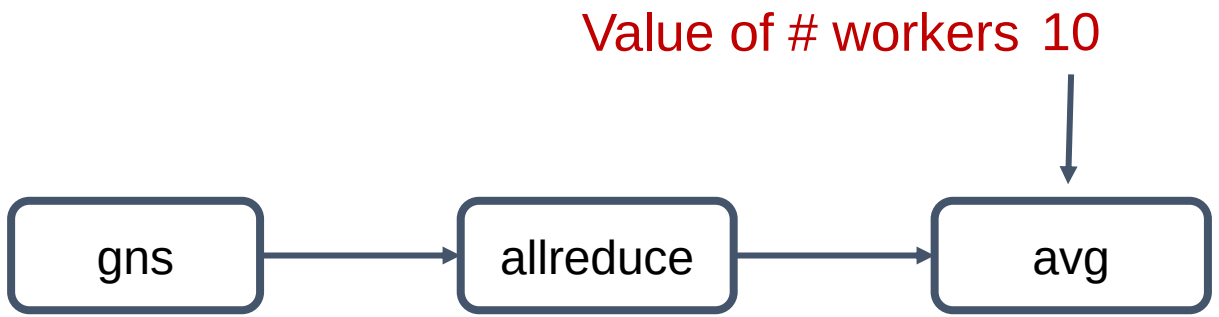
Idea: Include monitoring operators inside dataflow



Monitoring takes advantage of **optimisations** in dataflow engines and **collective communication** support

Changing System Parameters

Problem: Parameter adaptation affects **state consistency**



Dataflow for averaging GNS

Value may be stale

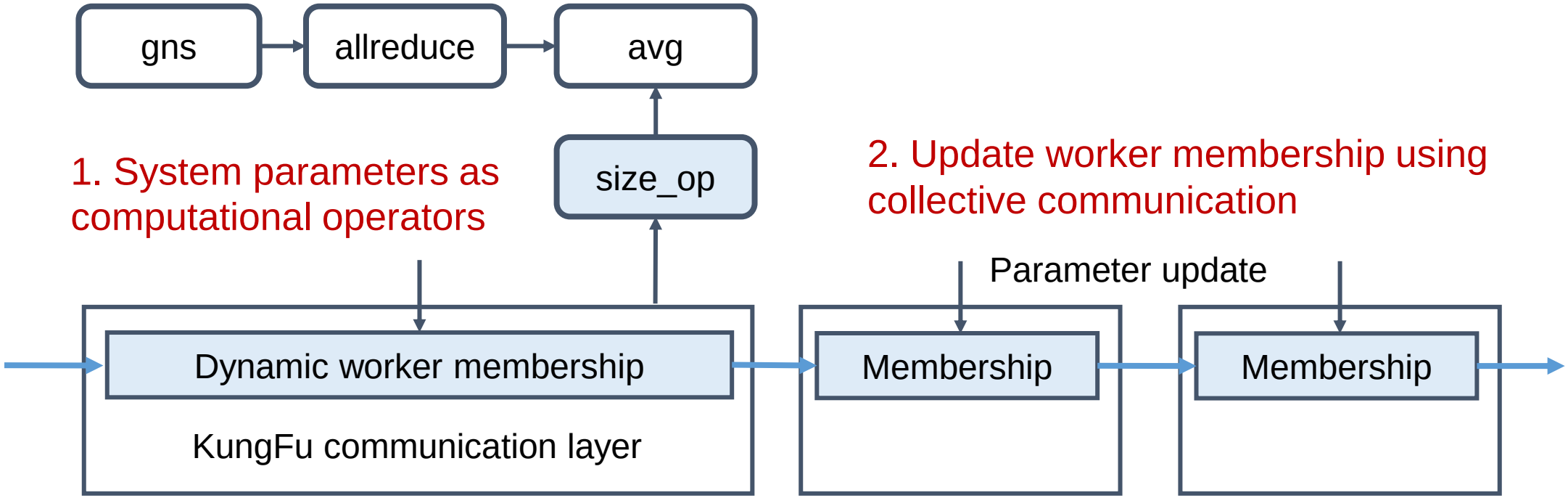
Other system parameters:

- Worker rank
- Communication topology
- …

Changing system parameters therefore typically requires **system restart**

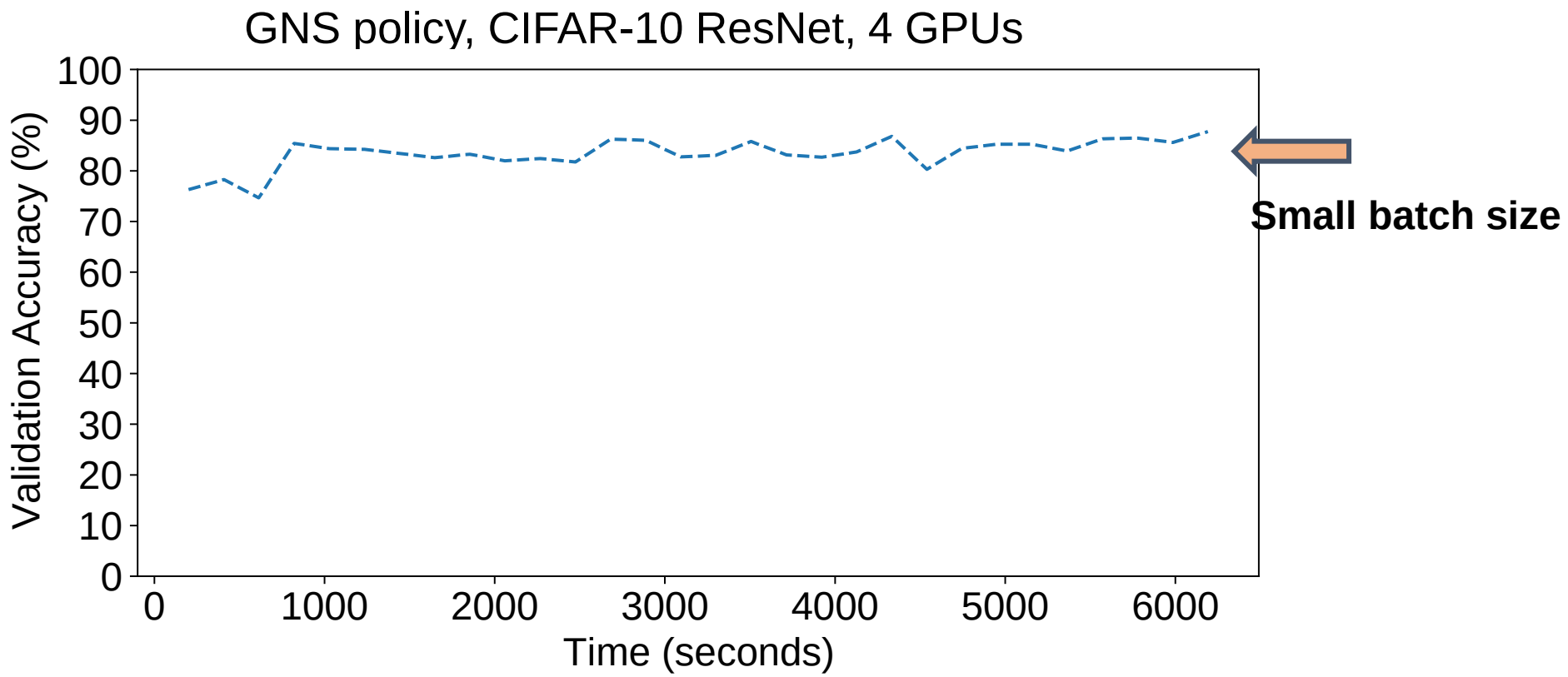
Distributed Mechanism for Changing Parameters

Idea: Decouple system parameters from dataflow state



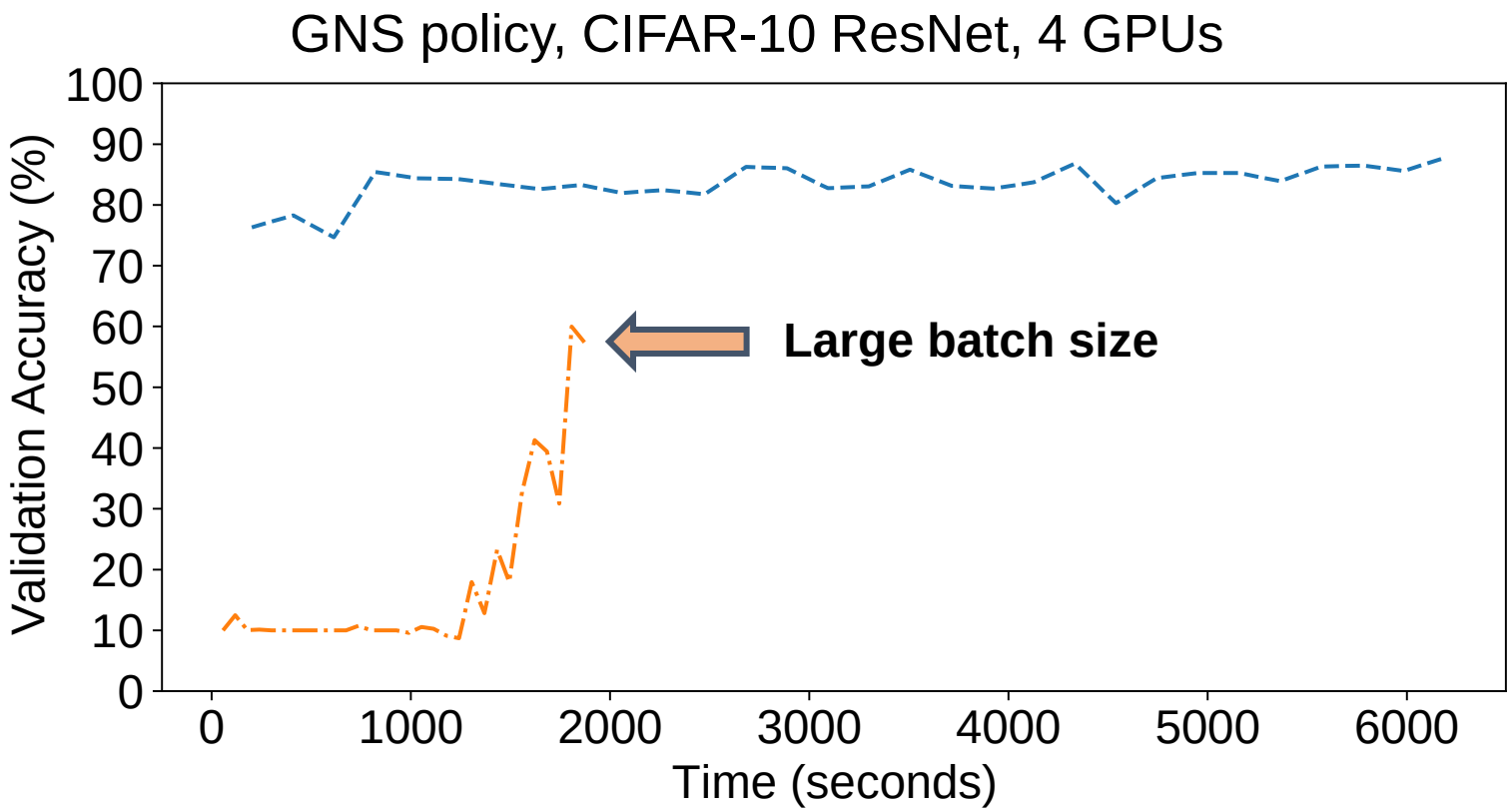
Always obtains up-to-date view of system parameters

How Effectively Does KungFu Adapt?



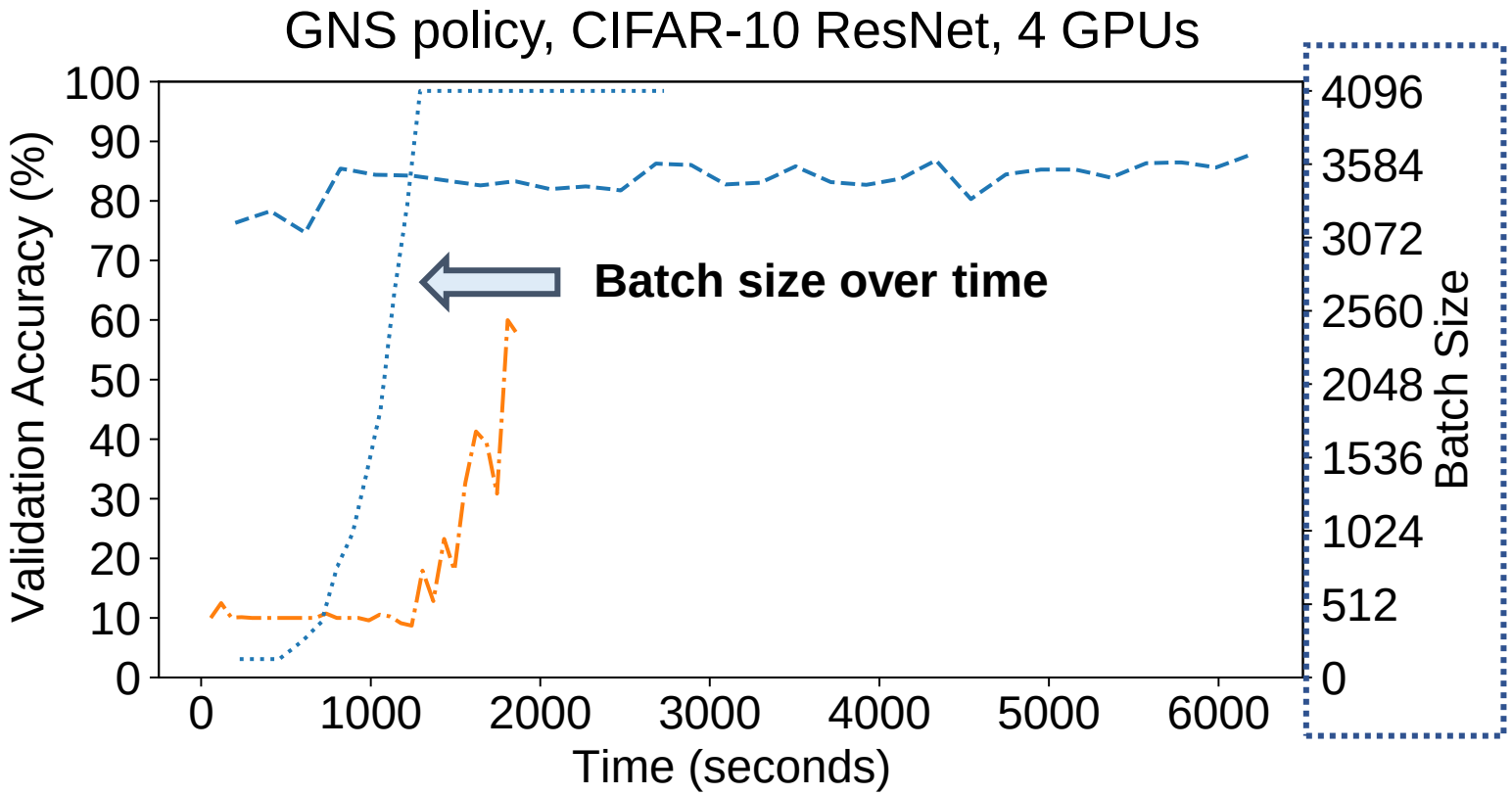
Small batch size reaches high accuracy, but converges slowly

How Effectively Does KungFu Adapt?



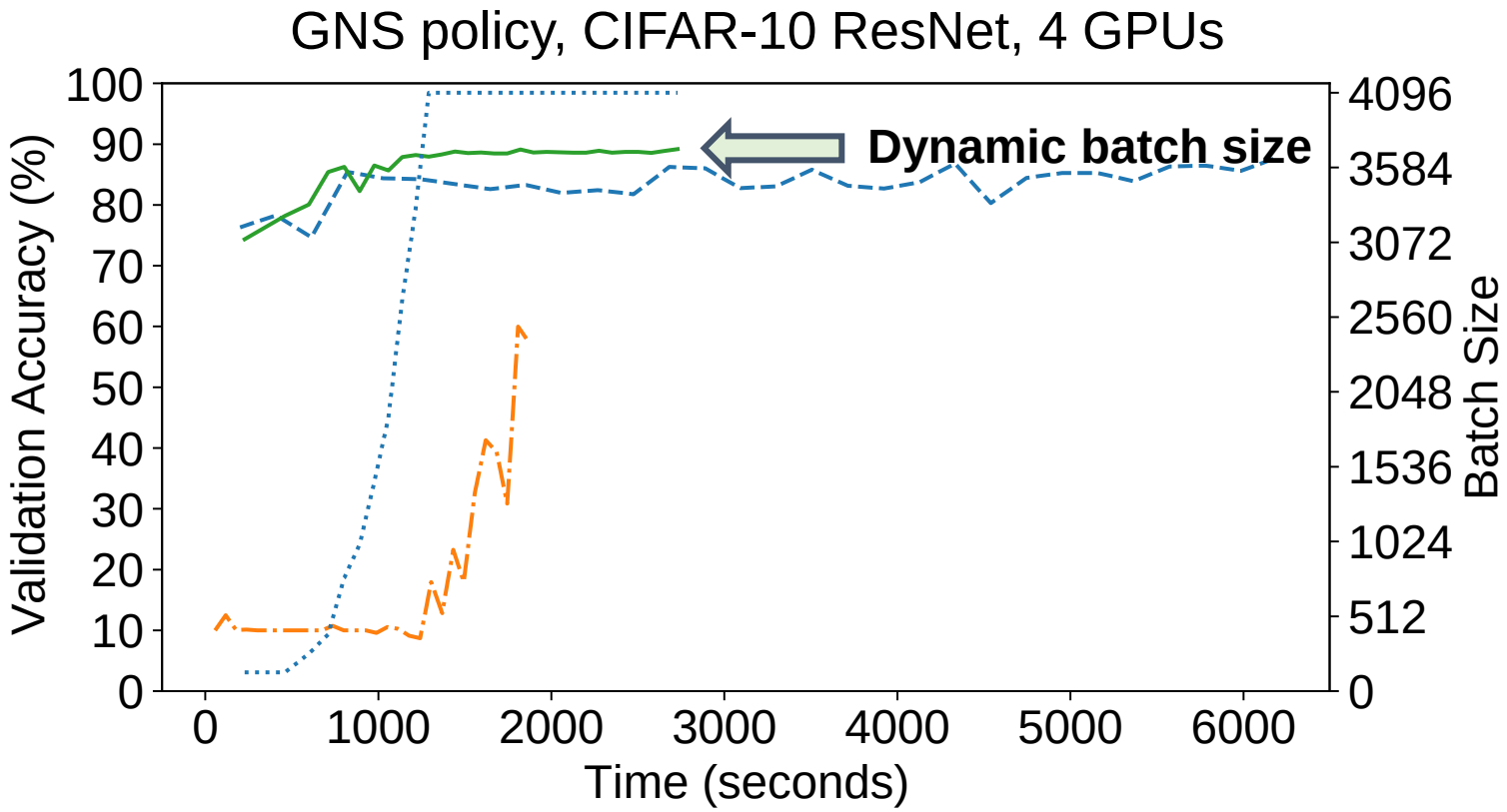
Large batch size finishes quickly, but accuracy suffers

How Effectively Does KungFu Adapt?



GNS predicts how effective batch size should increase during training

How Effectively Does KungFu Adapt?



Adaptation Policy has low overhead due to **embedded monitoring**

(2) Resource Independent Training

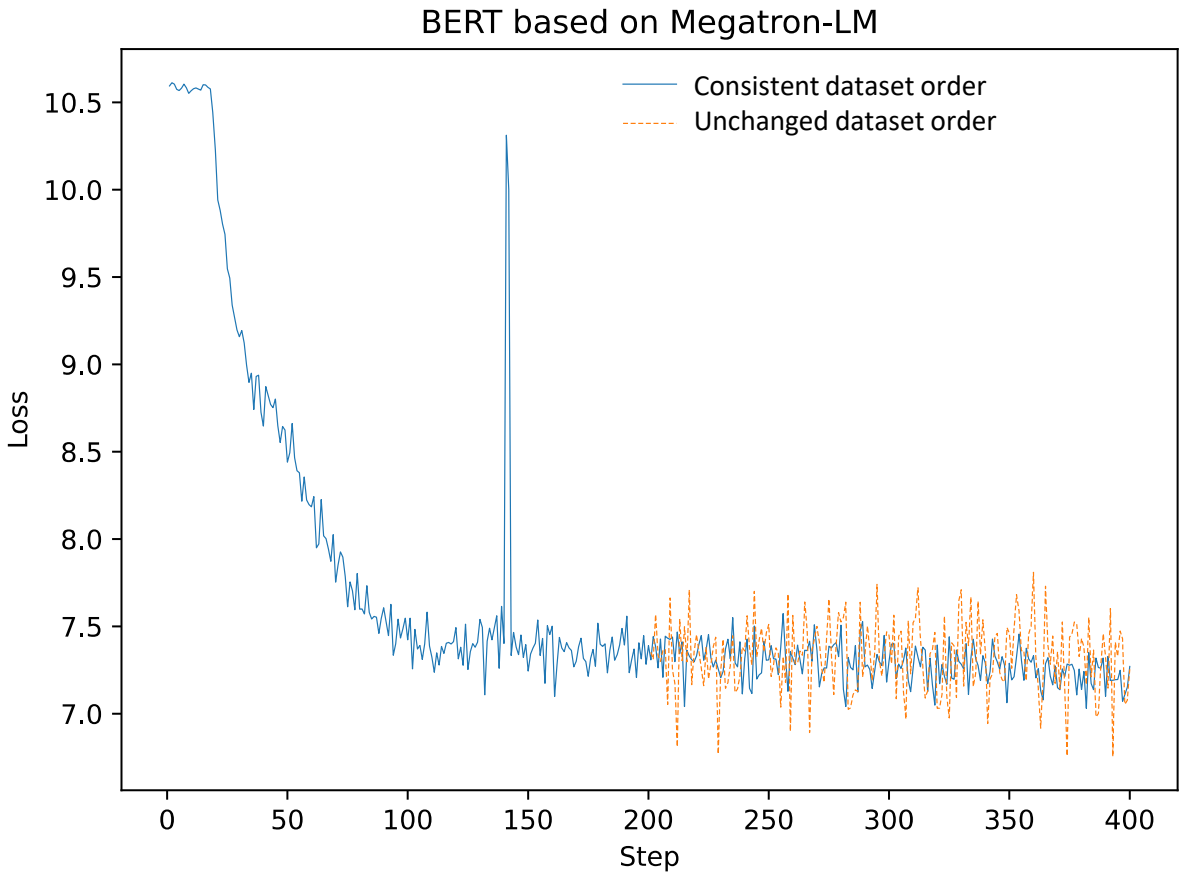
How is Training Affected by Resource Changes?

DNN model convergence changes with number of GPU devices

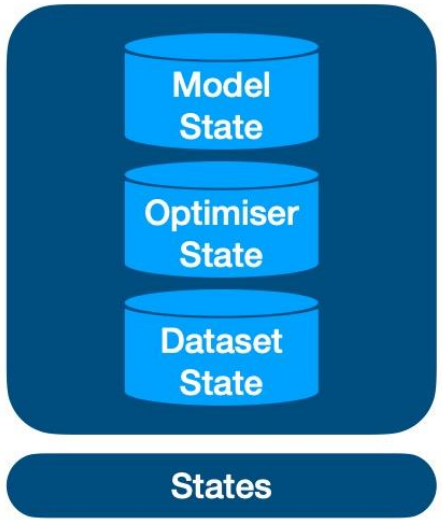
Experiment

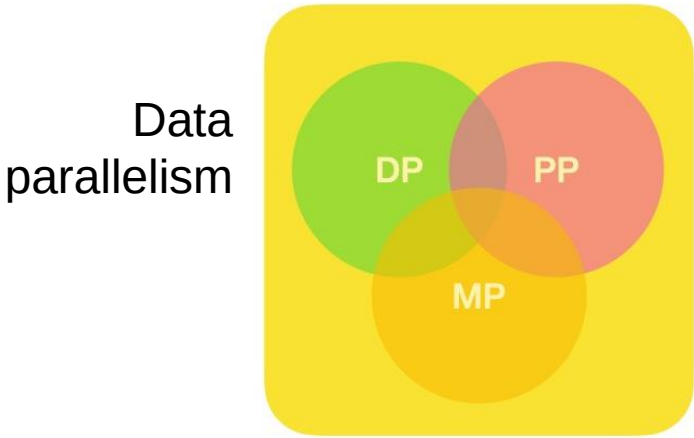
- After changing GPUs, **dataset order** is affected
- New dataset order **impacts convergence**

➔ **Changing resources may impact any training state**



Types of State Affected by Resources





Changing number of GPUs requires, e.g.:

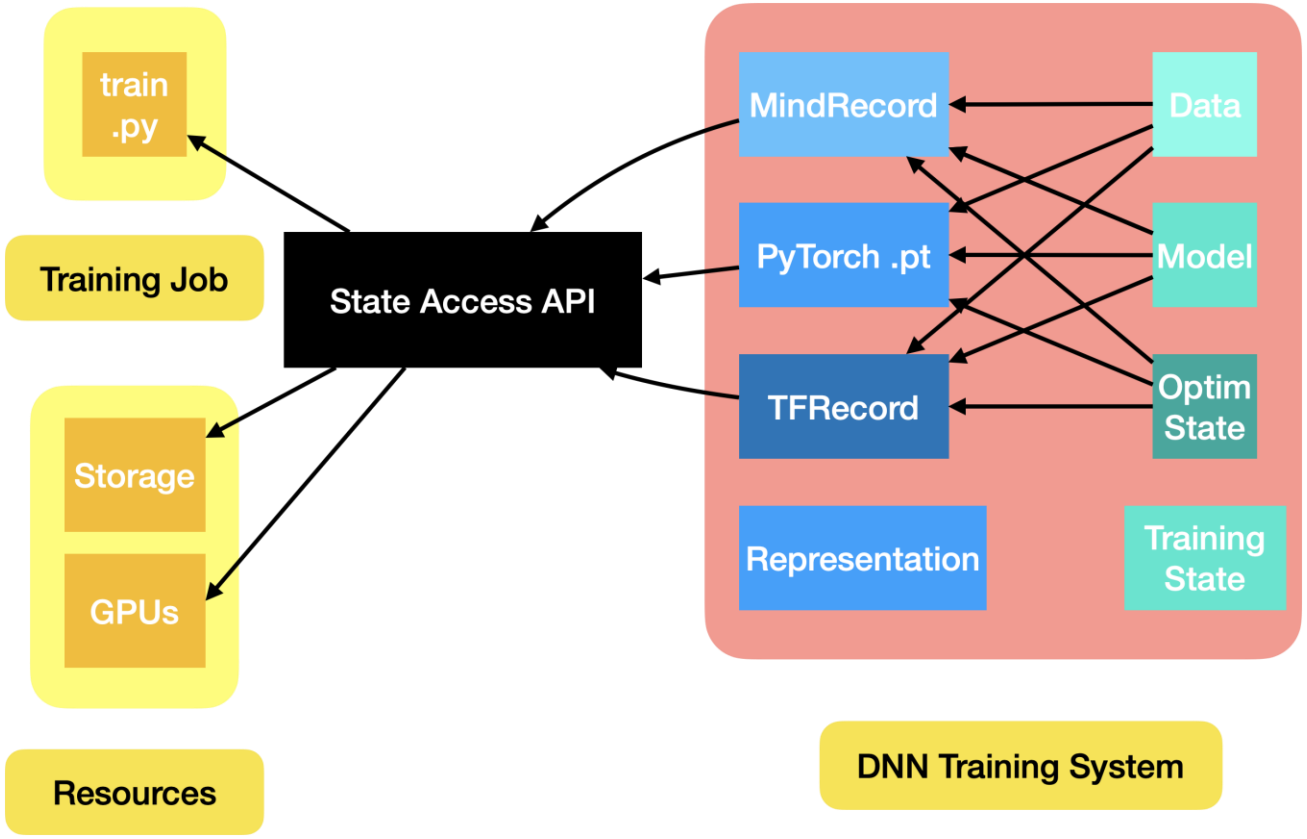
- **redistribution** of **dataset & model states**
- **update** of device-specific **batch size**

Transformations depend on type of parallelism used

Idea: Externalise Training State

Update training state when resources change to ensure consistent training

Requires **access** to training state outside of DNN system



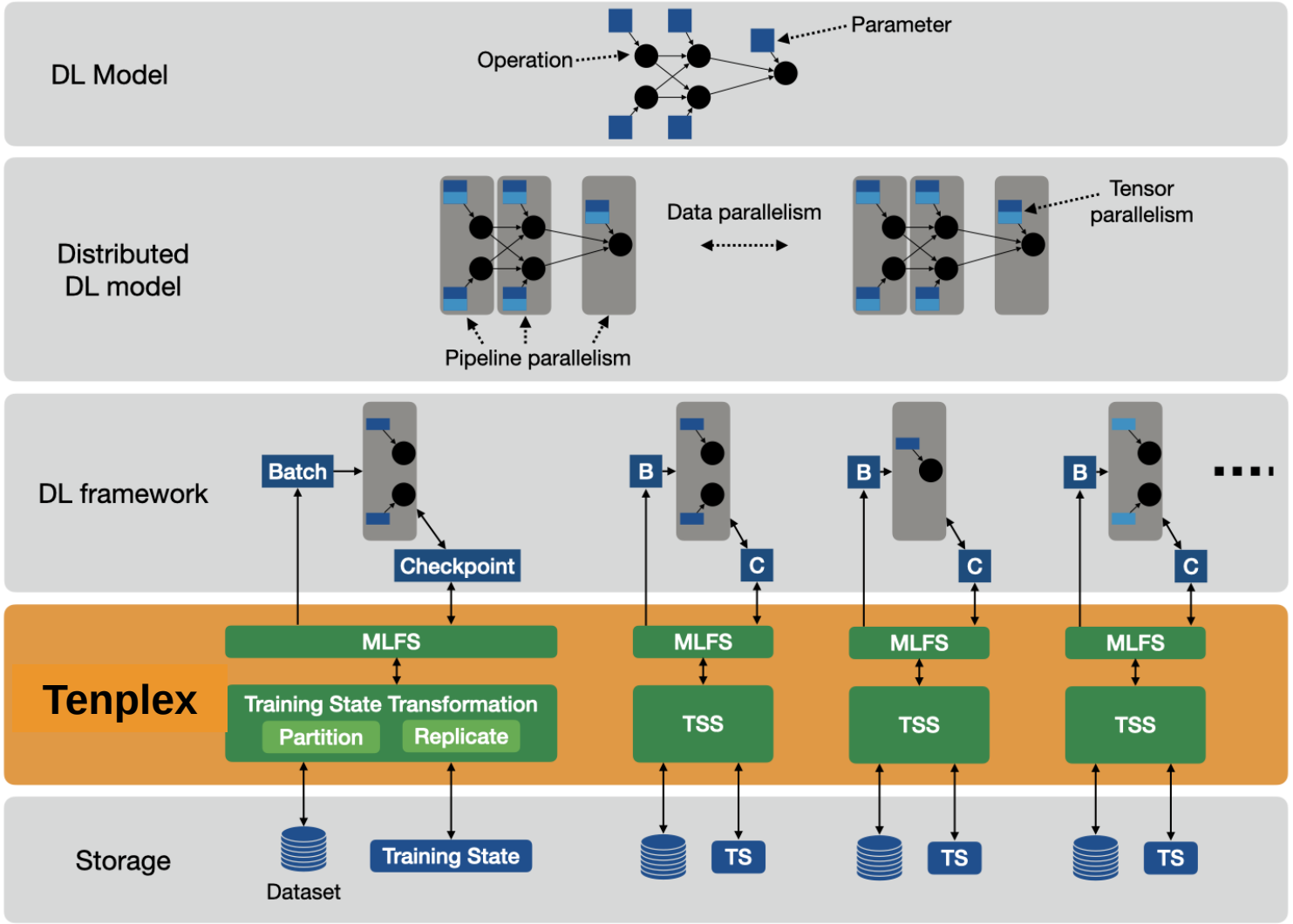
Tenplex: Resource-Independent DNN Training

New abstraction for accessing training state: **Virtual file system (VFS)**

- **Intercepts** state accesses by DNN system through VFS API
- **Transforms** state to maintain resource independence

Supports data, model, pipeline, hybrid parallelism

Compatible with MindSpore, PyTorch, Megatron-LM, DeepSpeed



State Access Through VFS Abstraction

http://<work-host>:<port>/dataset/<progress>/<worker-id>/…/<tensor>?range=

stored distributedly, mounted as Fuse:

/<mnt>/ds/<start-progress>/<worker-id>/meta.txt # e.g. batch size
/<mnt>/ds/<start-progress>/<worker-id>/filelist.txt # list of files, one per line
/<mnt>/ds/<start-progress>/<worker-id>/<data.part-1>
/<mnt>/ckpt/<worker-id>/<progress>/<mp-rank>?/<pp-rank>?/<model.part-1>
/<mnt>/ckpt/<worker-id>/<progress>/<mp-rank>?/<pp-rank>?/<optimizer.part-2>

Unified state addressing with file path/URL:

All state assigned URL

Simplified access API:

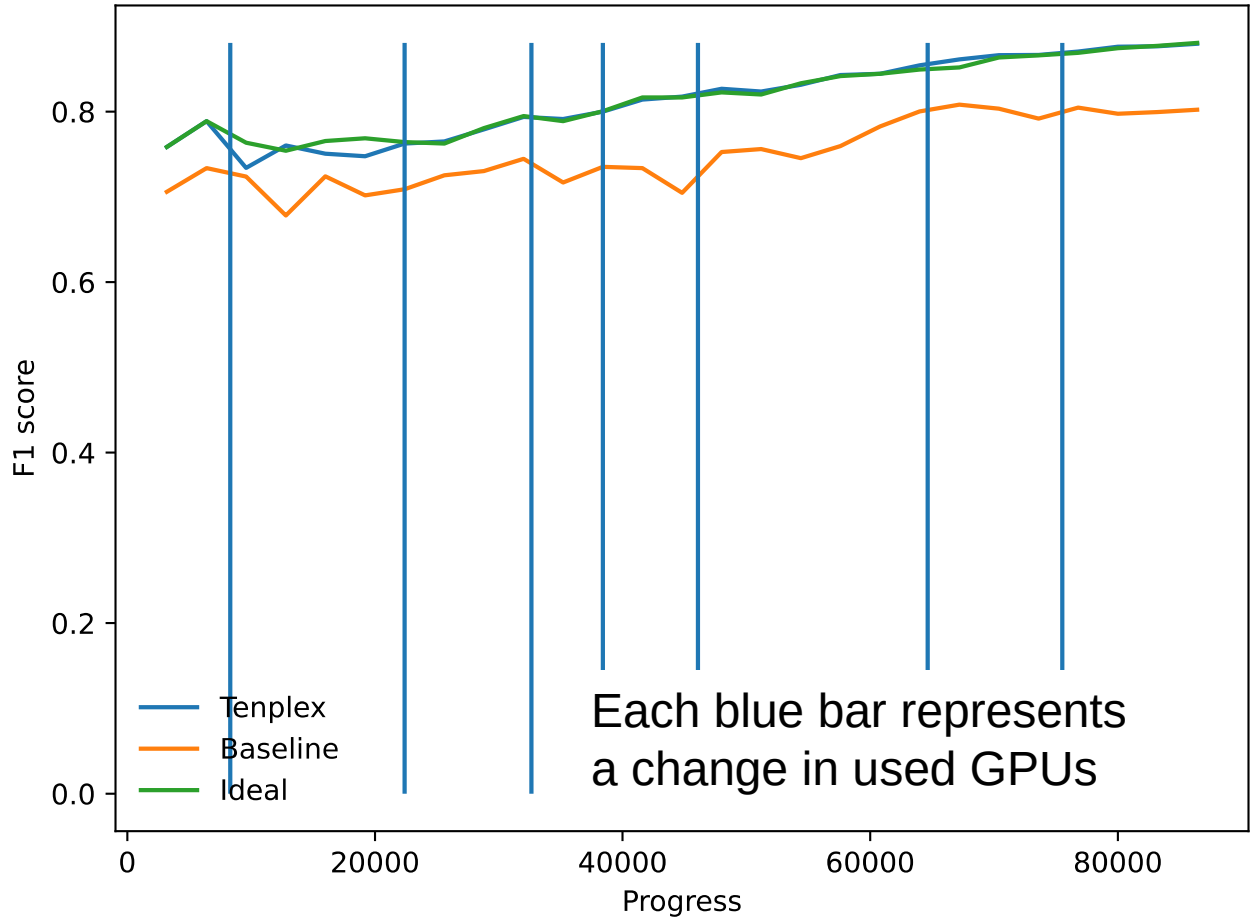
Training job/controller can access/migrate state via:

- plain File API
- or HTTP Request

```
def main(...):
    progress, worker_id, mp_rank, ... = init()
    btsz = int(open(f'/{mnt}/ds/{progress}/{worker_id}/batch-size.txt').read())
    files = open(f'/{mnt}/ds/{progress}/{worker_id}/filelist.txt').read().split('\n')
    ds = create_ds(files, batch_size=btsz)
    m = build_model(…)
    m.load(f'/{mnt}/{worker_id}/{progress}/{mp_rank}/data.pt')
    m.train(ds)
```

Resource Independent Elastic Training with Tenplex

Tenplex ensures consistent training when elastically scaling GPU workers



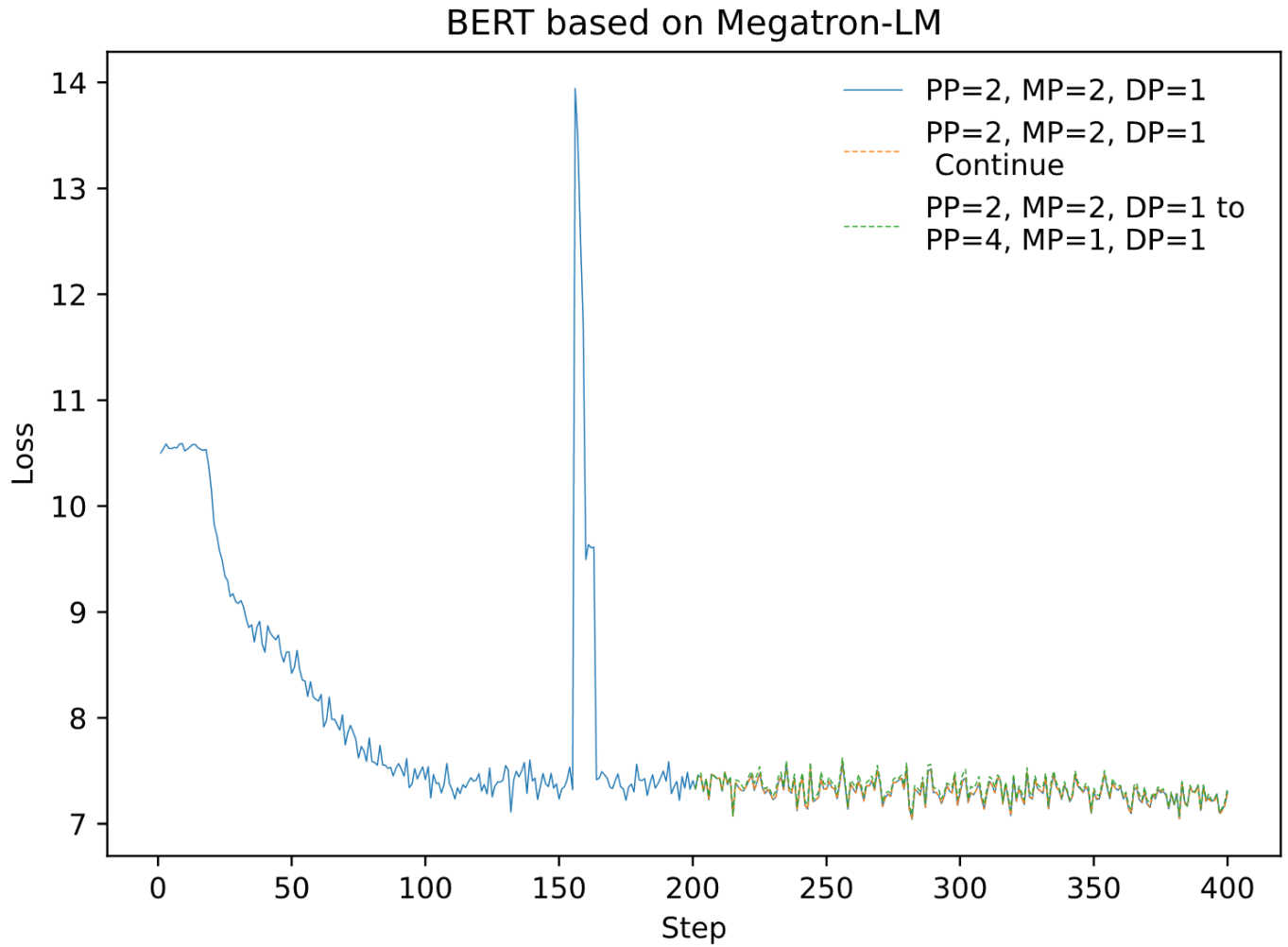
Consistent Convergence with Tenplex

Tenplex achieves consistent convergence with different **hybrid parallelism**:

Data parallelism (DP)

Model parallelism (MP)

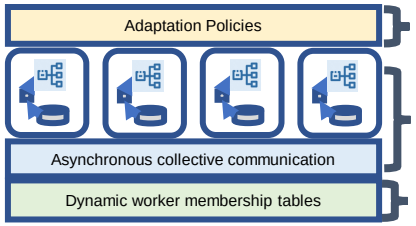
Pipeline parallelism (PP)



Summary:
New Abstractions for DNN Training Systems

Current DNN systems lack abstractions for adaptation & resource independence

(1) KungFu makes DNN training **adaptive**

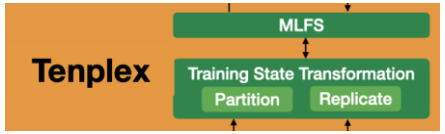


Decouple adaptation from training job

Take advantage of efficient dataflow execution

Provide powerful distributed primitives

(2) Tenplex decouples training state from GPU resources



Externalise state from DNN training system

Exposes virtual file system (VFS) abstraction

Transforms state to be resource independent

KungFu @ Github

https://github.com/lsds/KungFu

Thank You — Any Questions?

Peter Pietzuch
prp@imperial.ac.uk