Imperial College London

LSDS
Large-Scale Data & Systems Group

Crossbow: Scaling Deep Learning on Multi-GPU Servers

Peter Pietzuch

with Alexandros Koliousis, Luo Mai, Pijika Watcharapichat, Matthias Weidlich, Paolo Costa

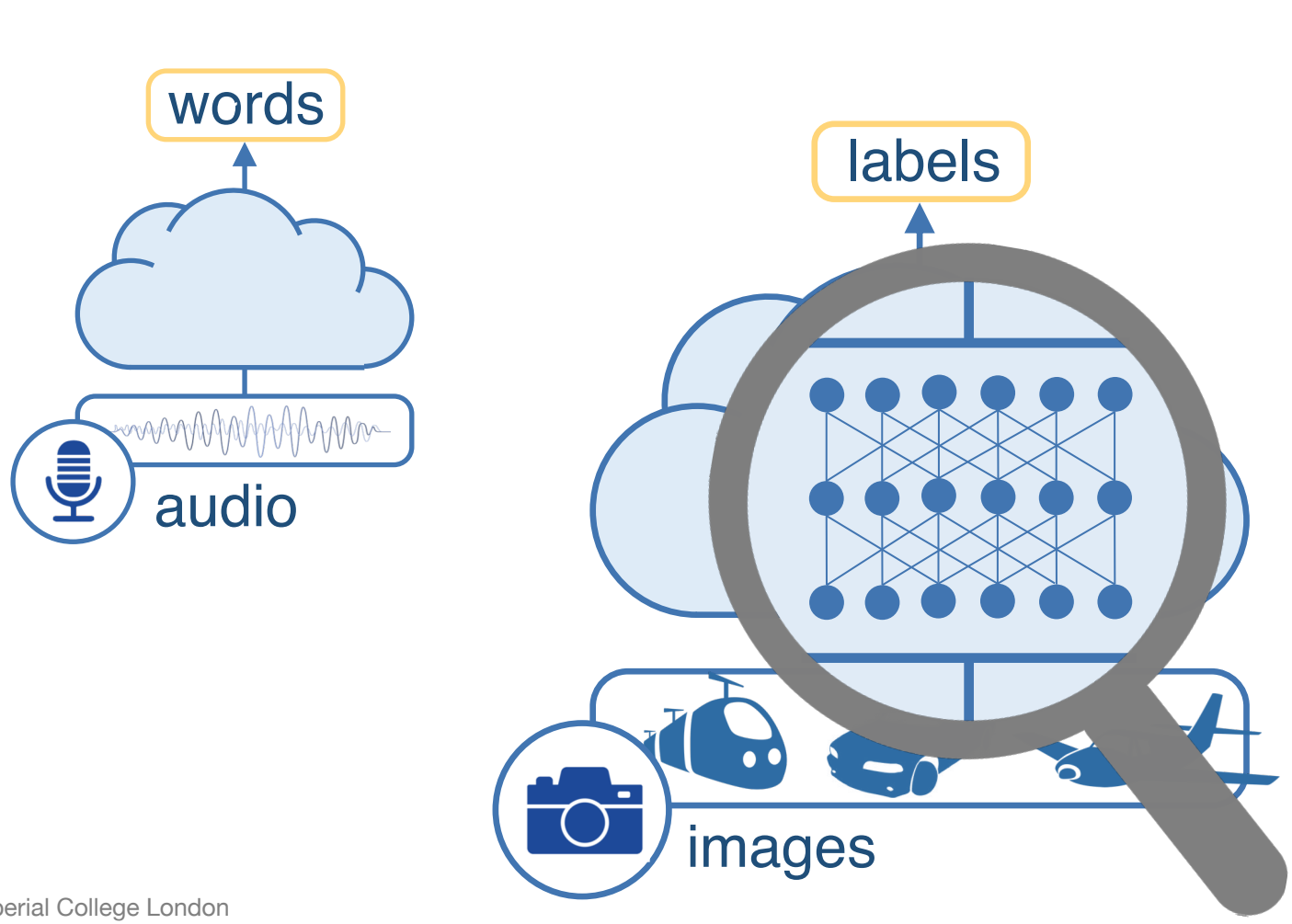
Imperial College London
http://lsds.doc.ic.ac.uk
<prp@imperial.ac.uk>

CASTOR Software Days – Stockholm, Sweden – October 2019

Machine Learning with Deep Neural Networks (DNN)

Revolutionised solutions in **vision**, **speech recognition**, …

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



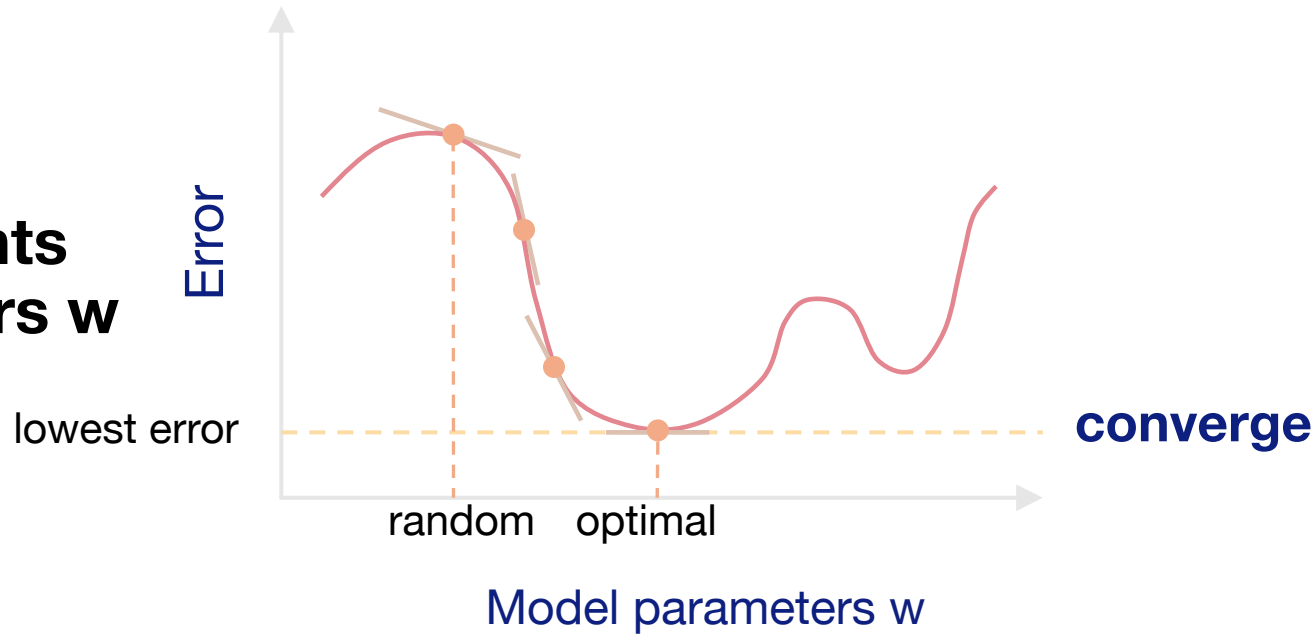
When DNN output is wrong, tweak its parameters

Training DNNs

Obtain DNN model that minimises **classification error**

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

1. Begin with **random model**
2. Consider **mini-batch** of training data
3. Iteratively calculate **gradients** & update **model parameters w**



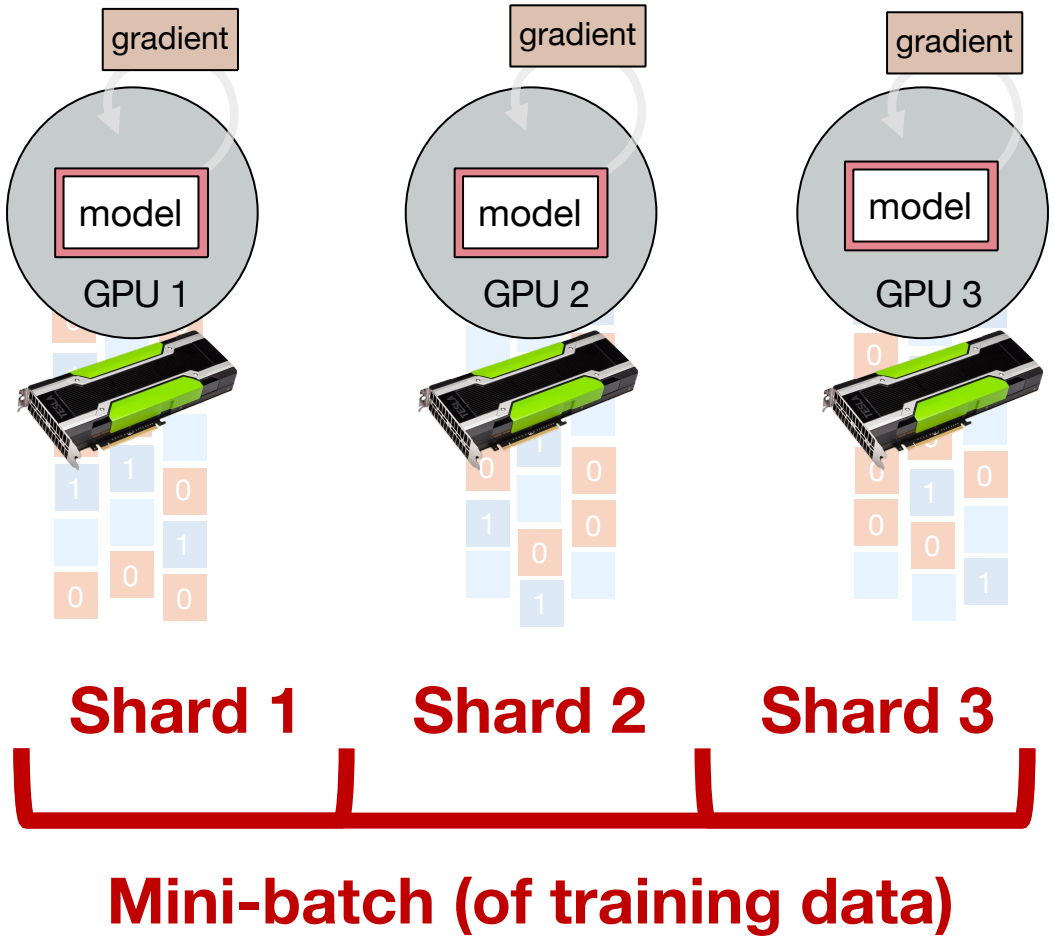
Training DNNs on GPUs

GPUs are good at parallelising gradient computation

Training DNNs in Parallel with GPUs

With large datasets, speed up by calculating gradients on **multiple GPUs**

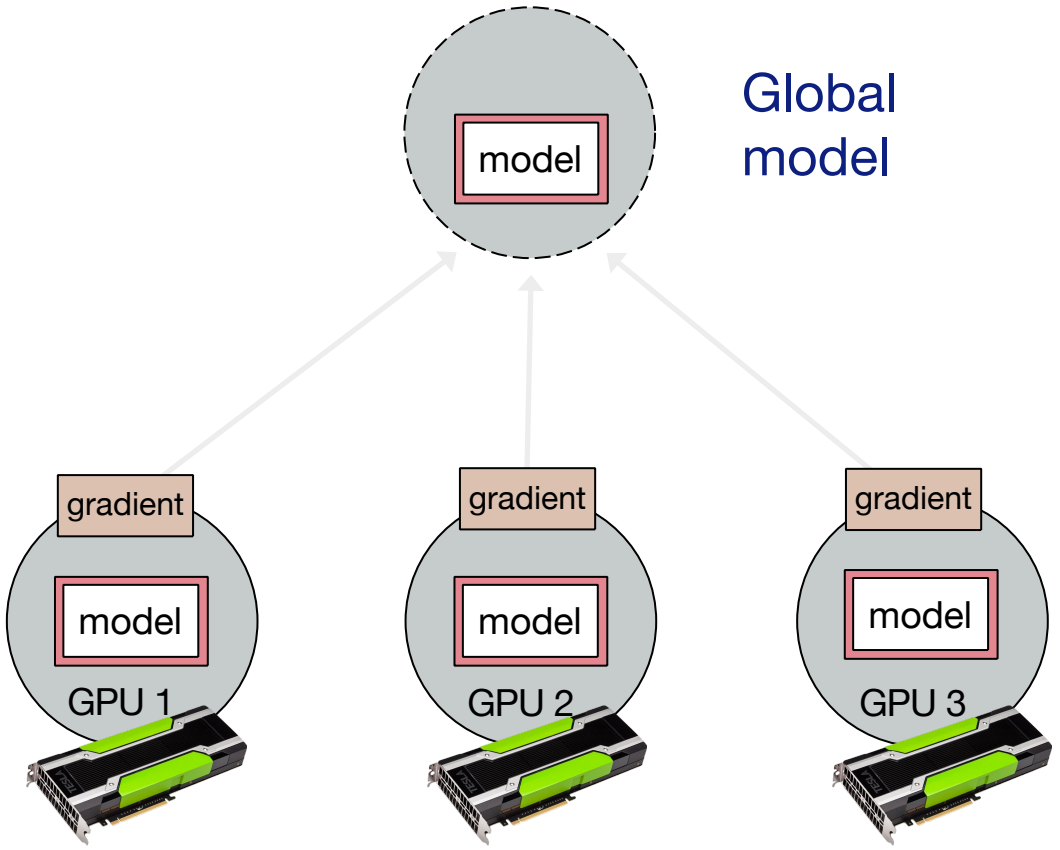
Every GPU has **model replica** with a copy of model parameters (or weights)



But model replicas would **diverge** over time…

Model Synchronisation among GPUs

Parameter server: Maintains **global model**



GPUs:

1. Send gradients to **update global model**
2. **Synchronise** local model replicas with global model

The Problem with Large Batch Sizes

Yann LeCun
@ylecun

Training with large mini-batches is bad for your health.

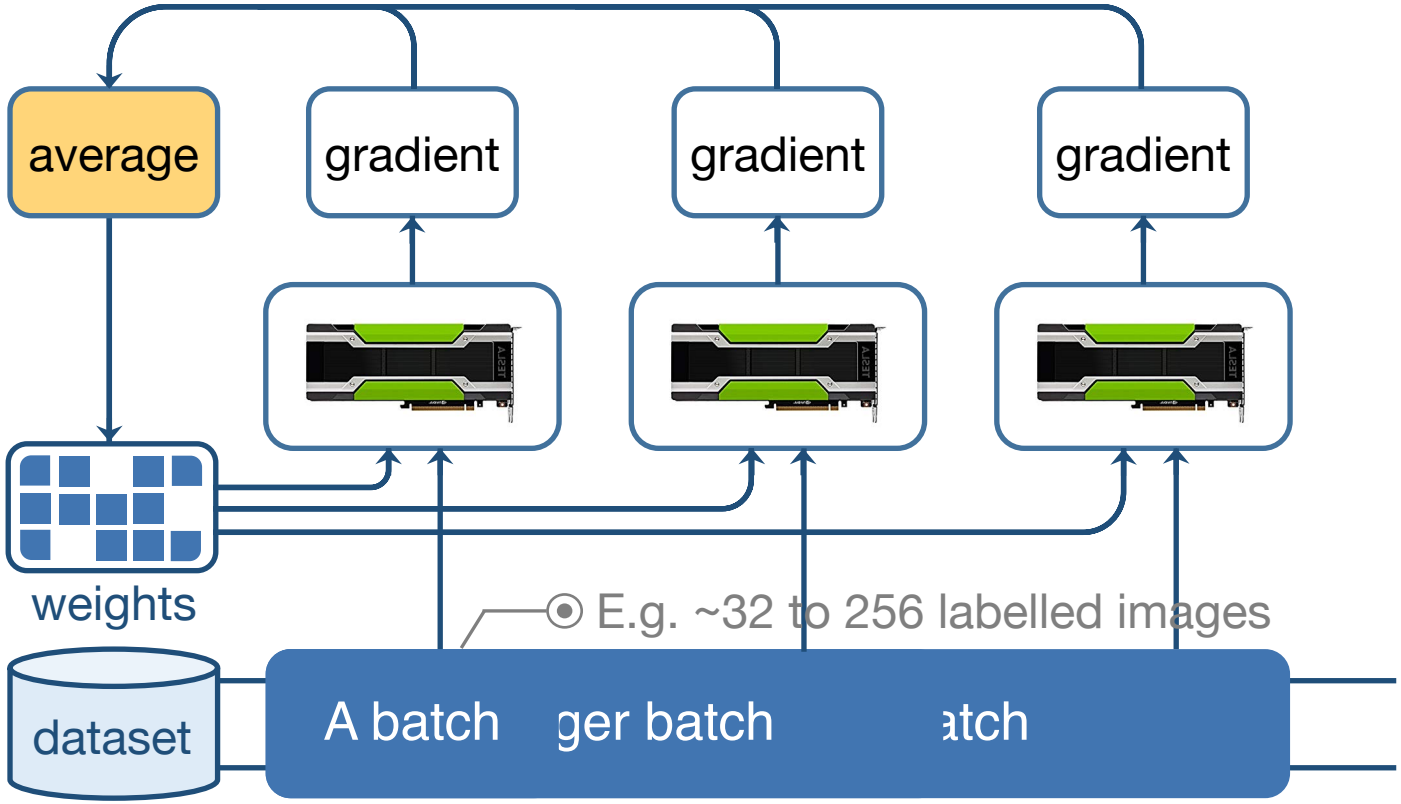
More importantly, it's bad for your test error.

Friends don't let friends use mini-batches larger than 32.

2:00 PM – 26 Apr 2018

447 1.2K

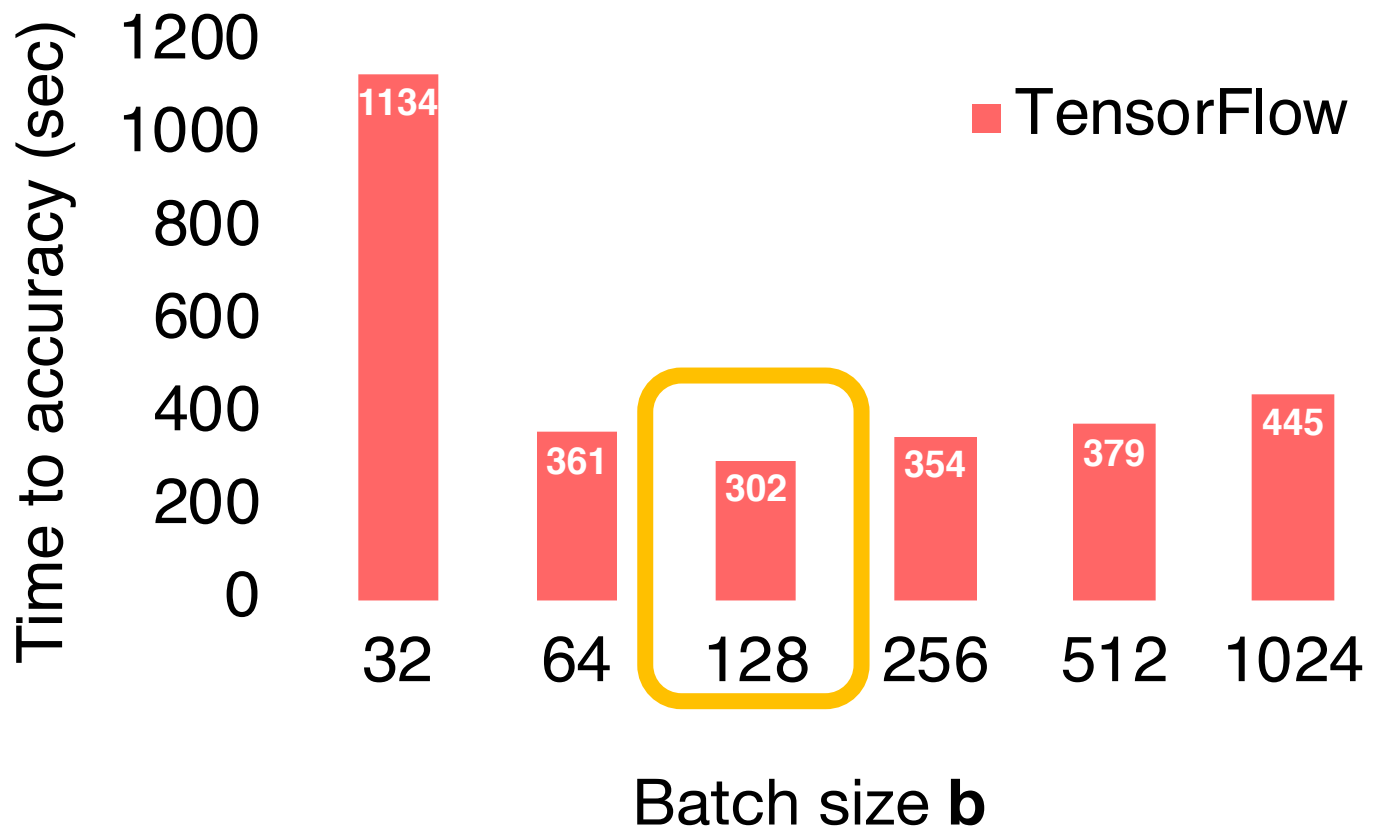
Why Use Large Batch Sizes?



Keep work per GPU **constant** to scale

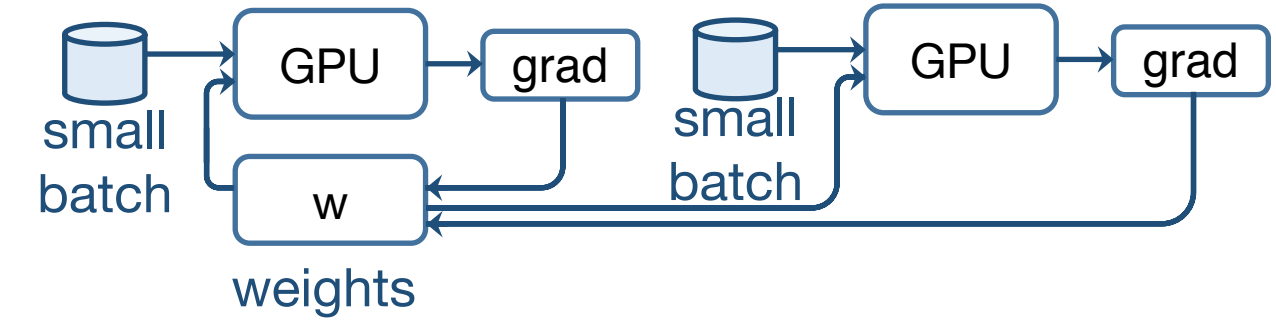
What is the Best Batch Size on a GPU?

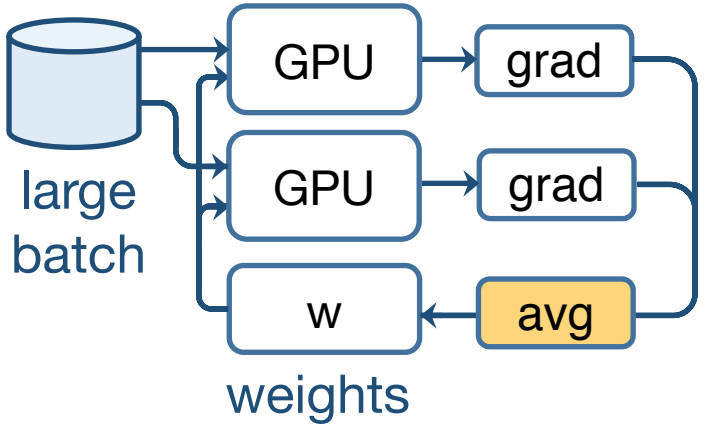
ResNet-32 on NVIDIA Titan X GPU

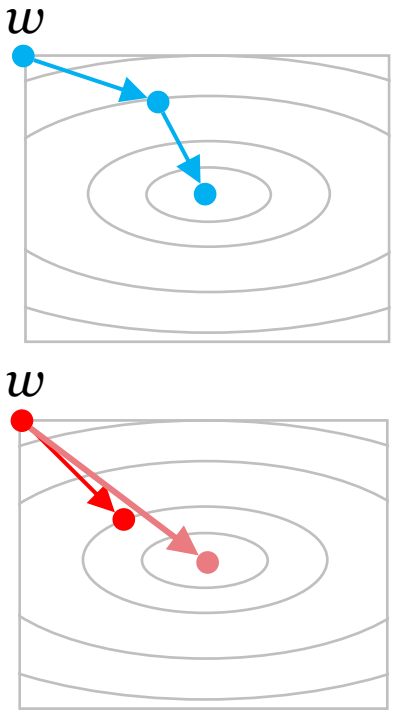


Training DNNs Favours Small Batch Sizes

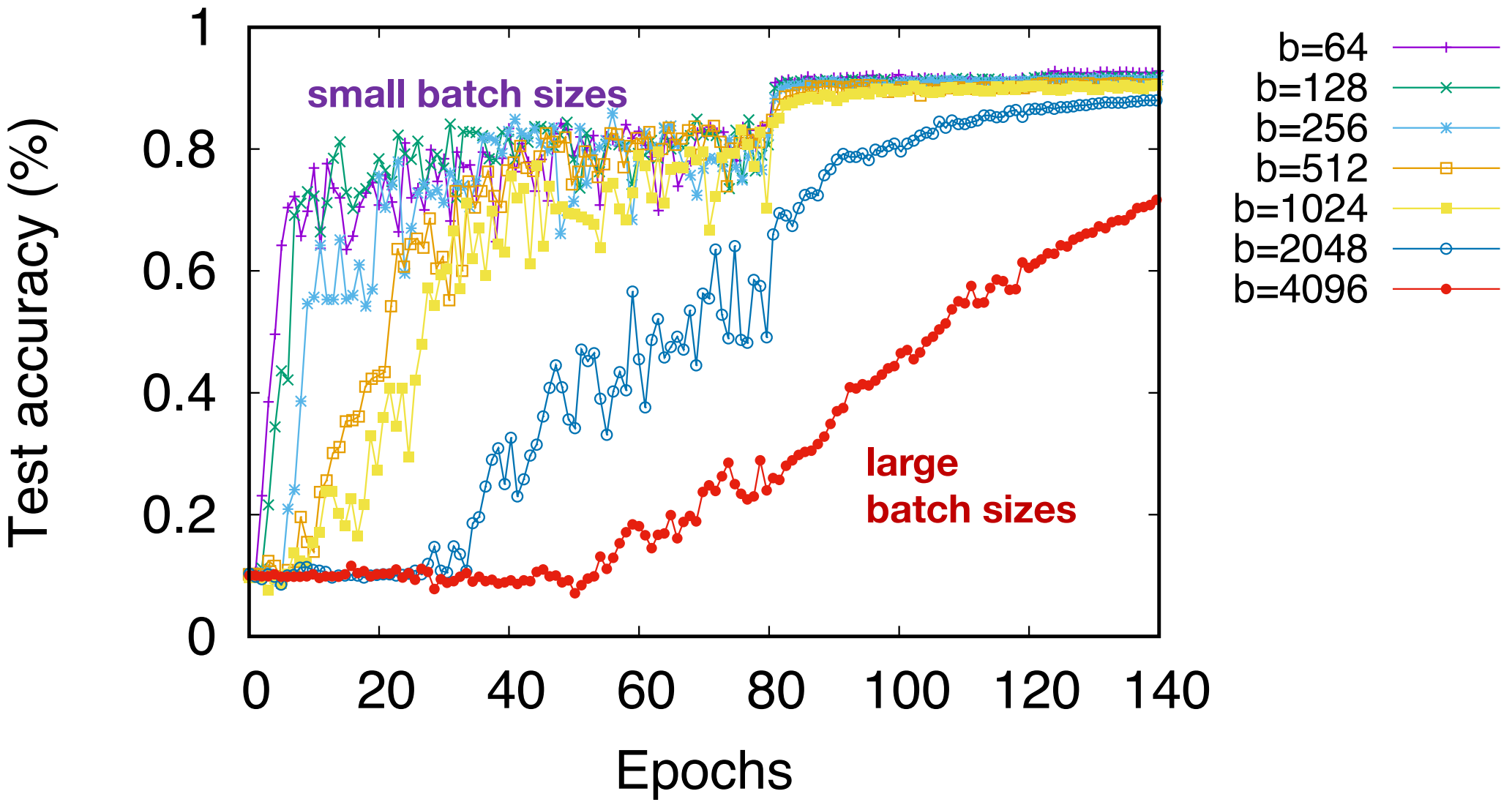
We want frequent, less “noisy” updates



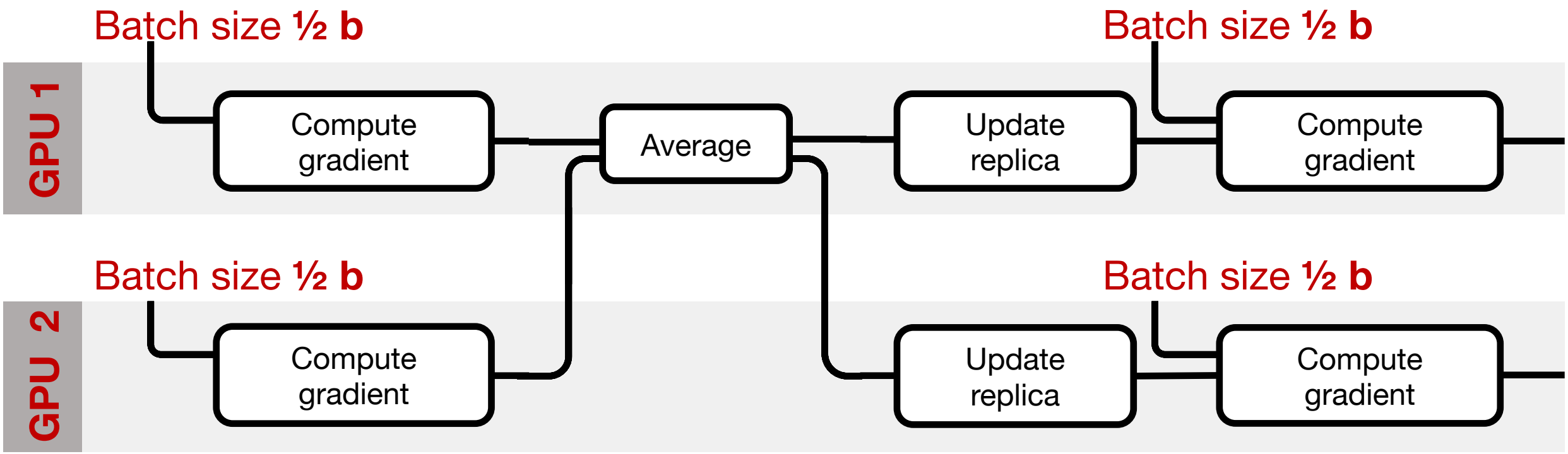




Statistical Efficiency Needs Small Batch Sizes

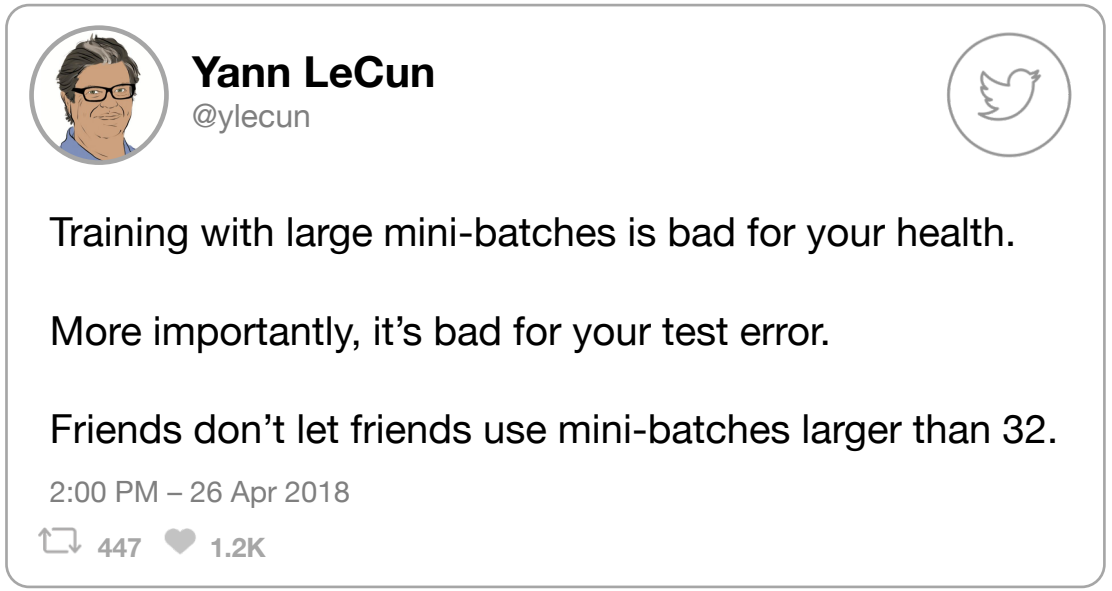


Hardware Efficiency Needs Large Batch Sizes



Keep **work per GPU constant** → **increase batch size** with #GPUs

Tension between Hardware & Statistical Efficiency



Practitioners increase batch size due to **hardware efficiency**

But best batch size depends on both **hardware** & **statistical efficiency**

Current Practice: Hyper-Parameter Tuning

Adjust hyper-parameters (eg learning rate, momentum etc) to avoid reduction in statistical efficiency

Linear scaling rule:
"When mini-batch size is multiplied by *k*, multiply learning rate by *k*"

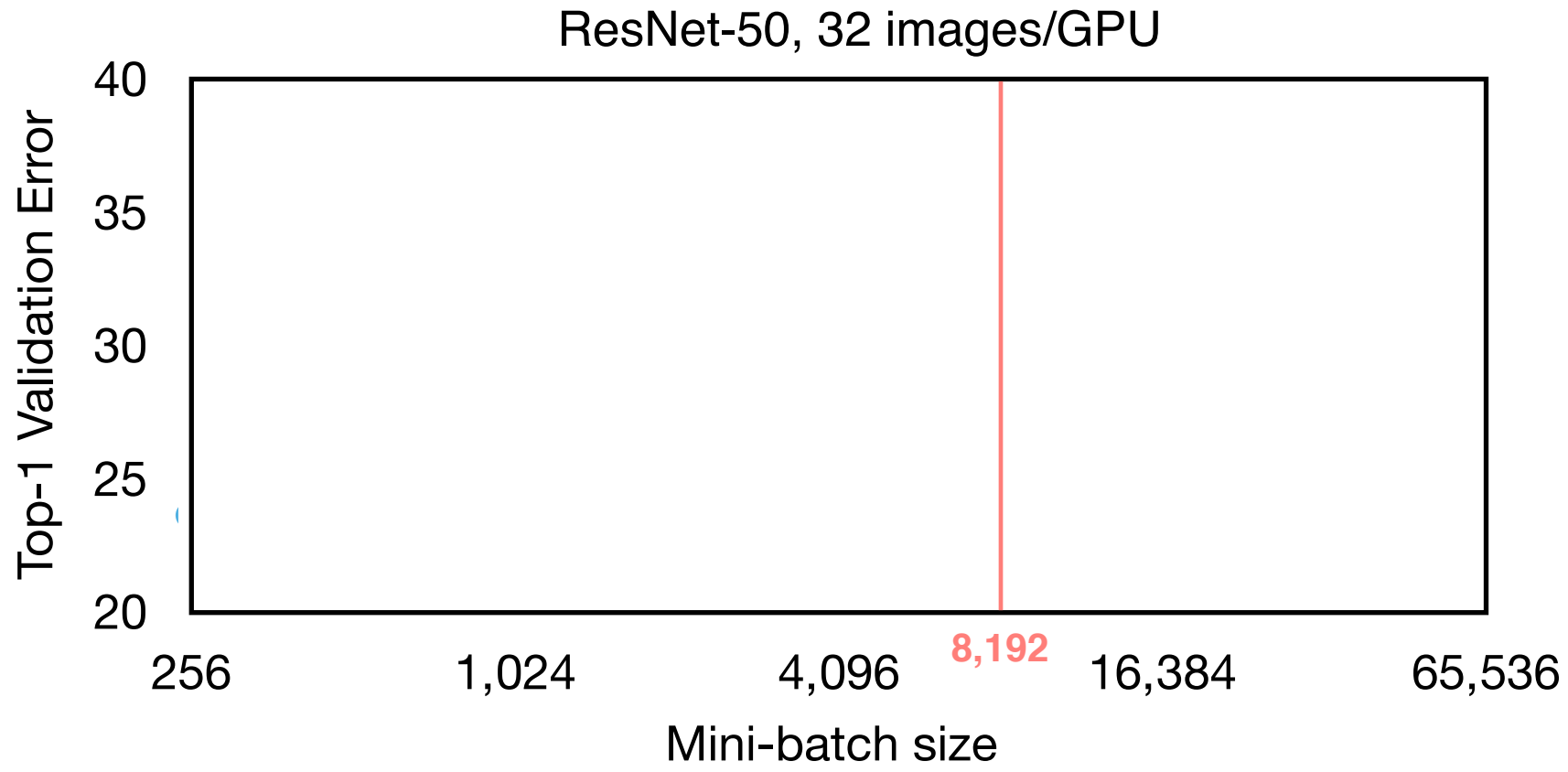
Goyal et al. (2017)

Drawbacks

- Manual, labour-intensive process
- Highly model specific – not portable and does not work for some models
- Less effective for very large batch sizes…

Limits of Hyper-Parameter Tuning

"When mini-batch size is multiplied by *k*, multiply learning rate by *k*"



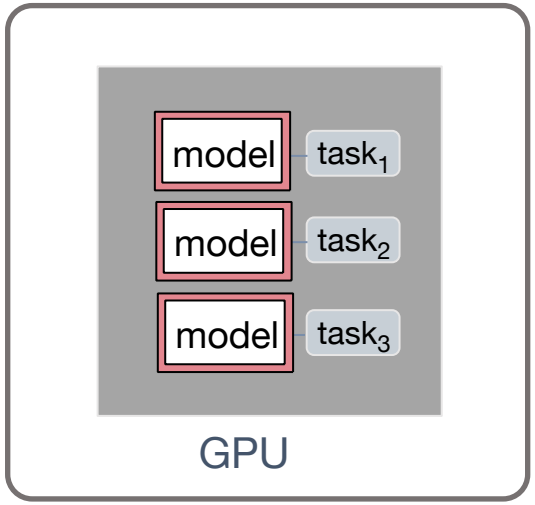
Fundamental Challenge of GPU Scaling

"If batch size could be made arbitrarily large while still training effectively, then training is amenable to standard weak scaling approaches. However, if the training rate of some models is restricted to small batch sizes, then we will **need to find other algorithmic and architectural approaches** to their acceleration."

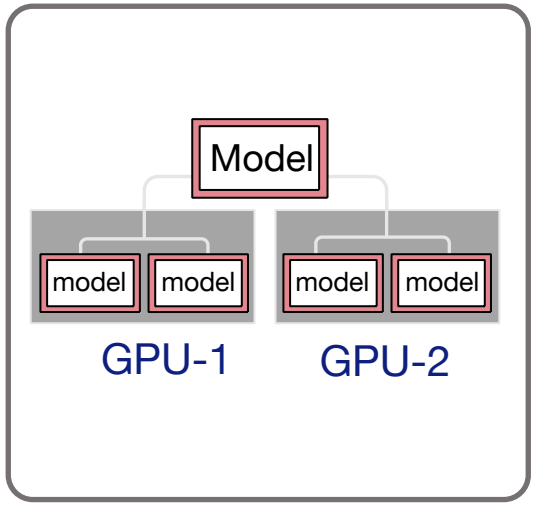
– J. Dean, D. Patterson et al., "A New Golden Age in Computer Architecture", IEEE Micro, 2018

How to design a deep learning system that scales training with multiple GPUs, even when the preferred batch size is small?

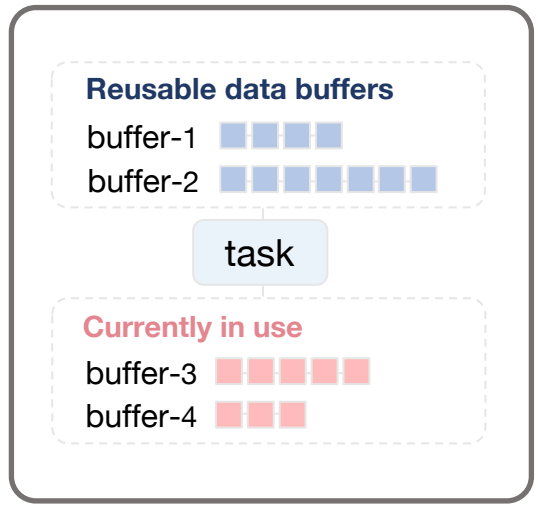
(1) How to increase hardware efficiency with small batches?



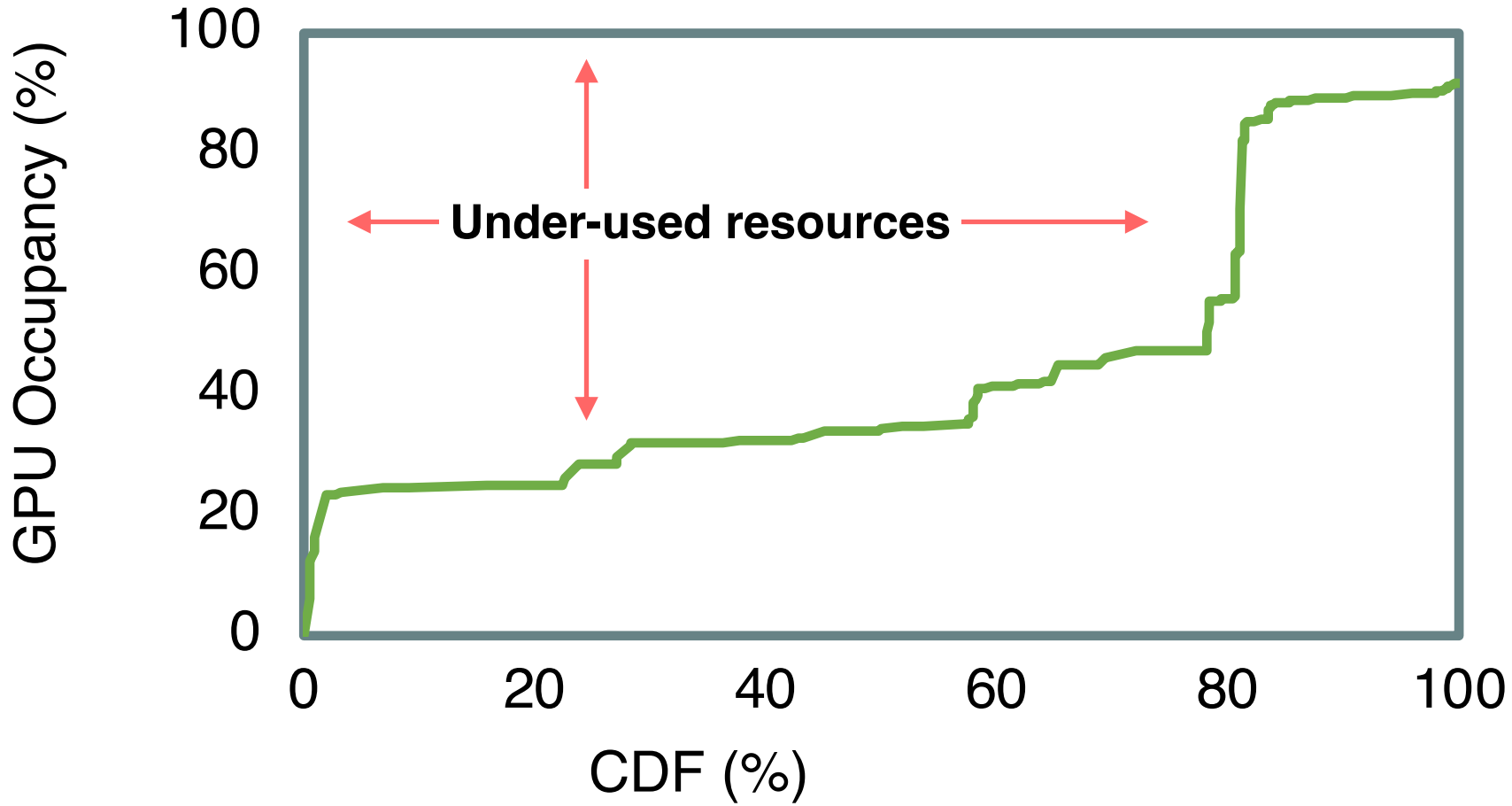
(2) How to synchronise model replicas?



(3) How to reduce scheduling & synchronisation overheads?



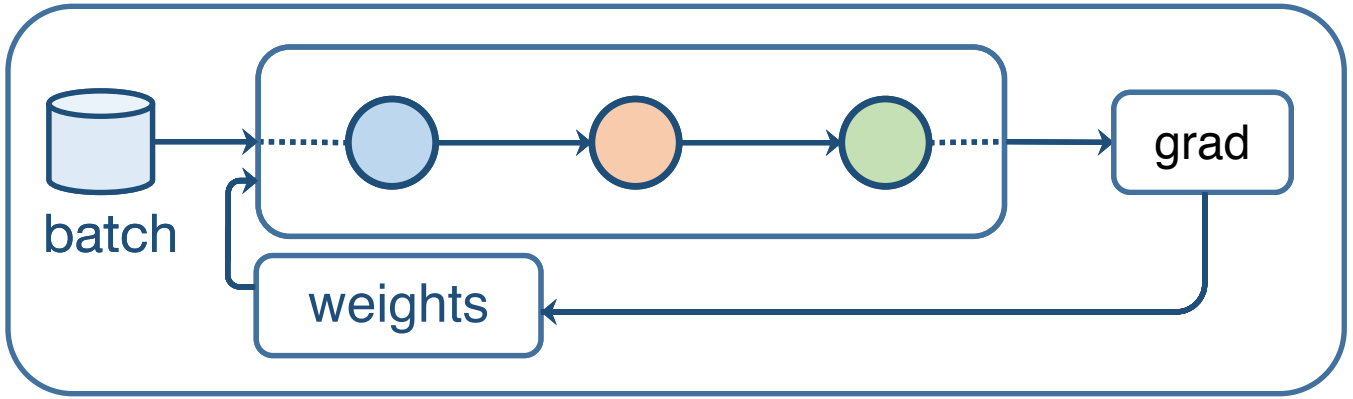
Problem: Small Batch Sizes Underutilise GPUs



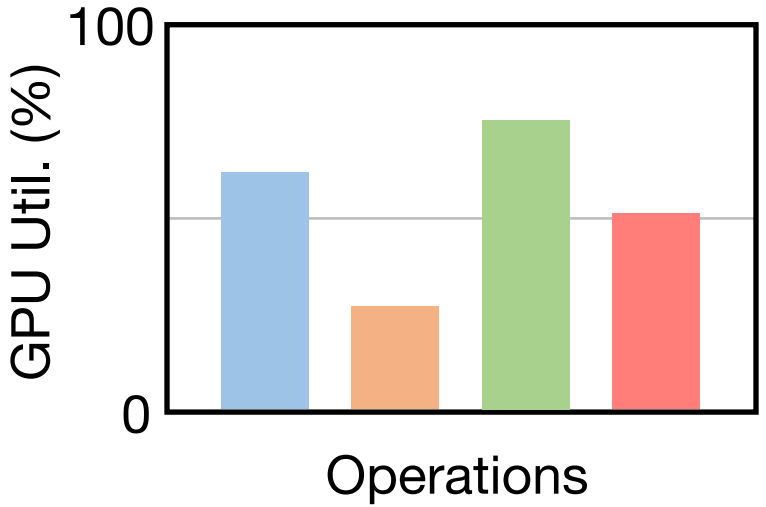
How to Process Small Batches Efficiently?

One batch per GPU →

Not enough data and instruction parallelism for every operator

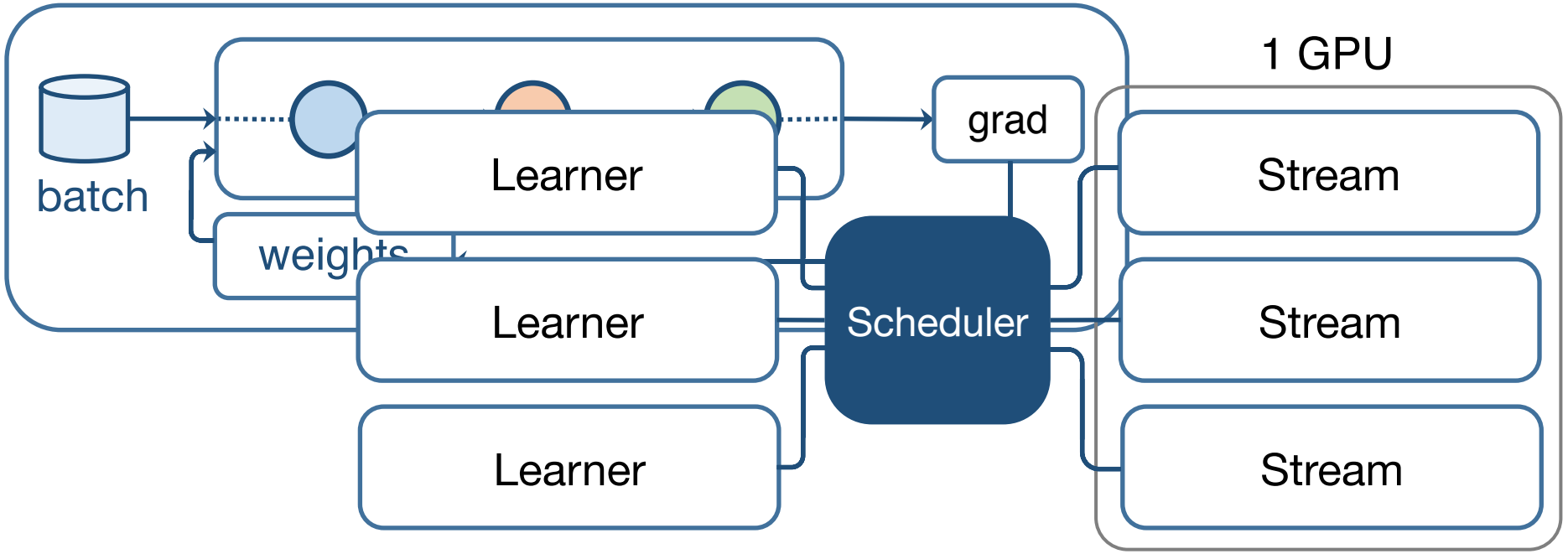


One per GPU

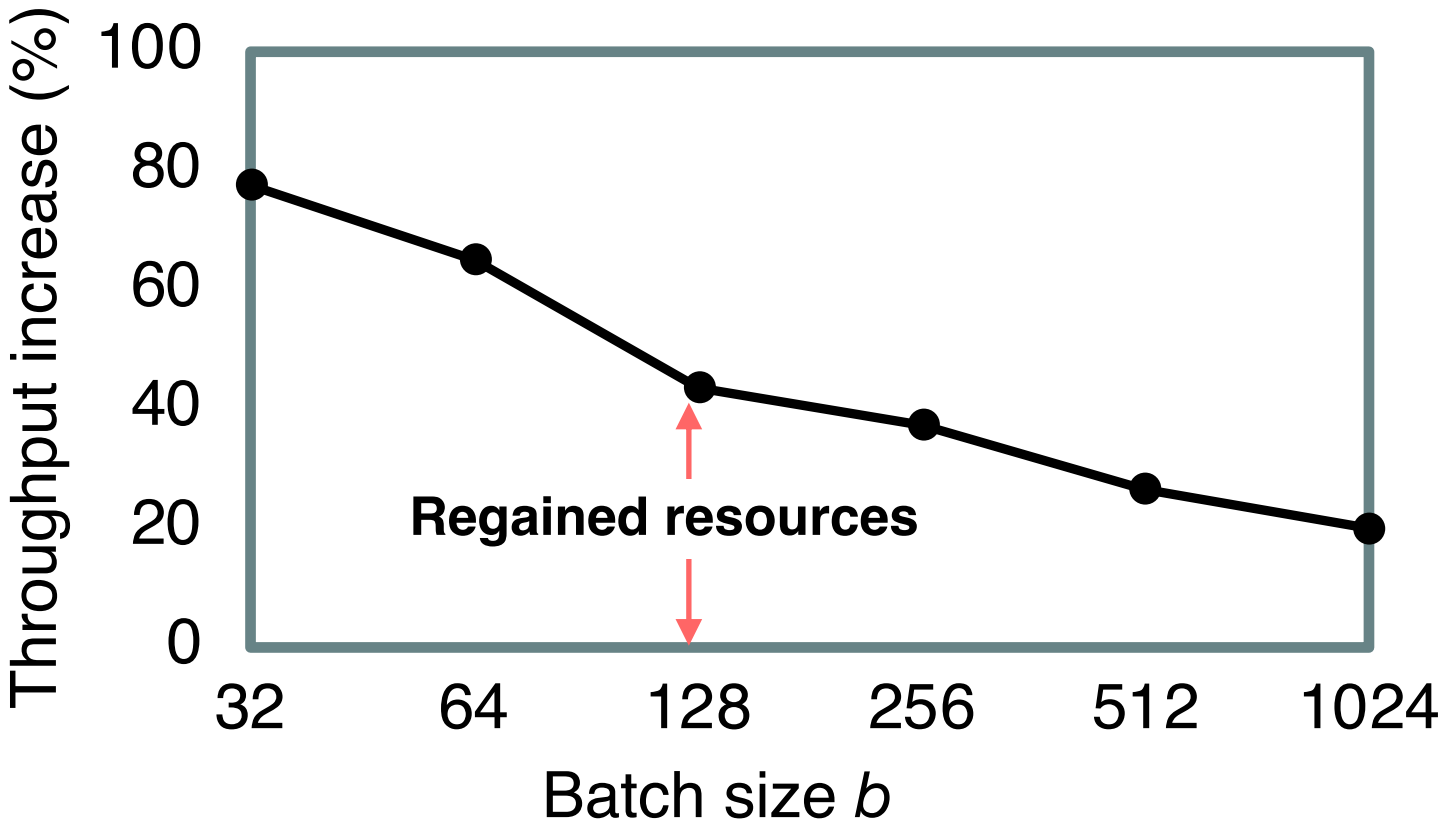


Idea: Train Multiple Model Replicas per GPU

One learning process (or **learner**) per **GPU stream**

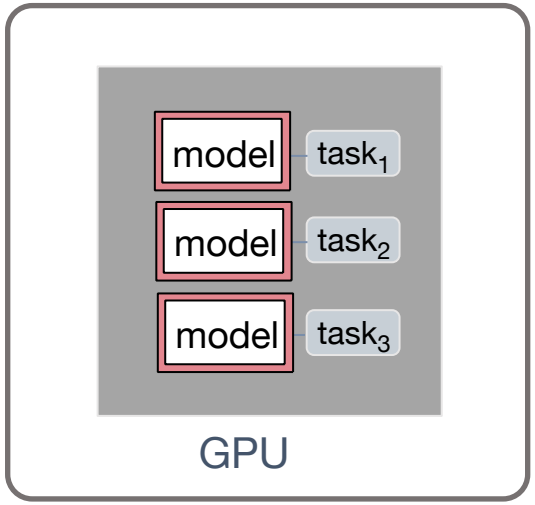


Effect of Training Multiple Model Replicas per GPU



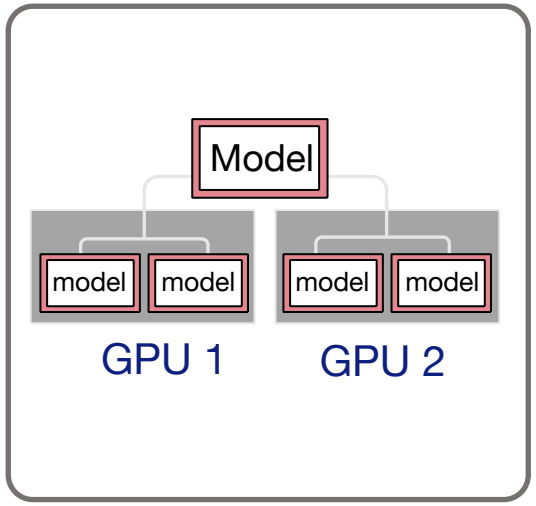
But now we must synchronise a large number of learners/model replicas...

(1) How to increase efficiency with small batches?



Train multiple model replicas per GPU

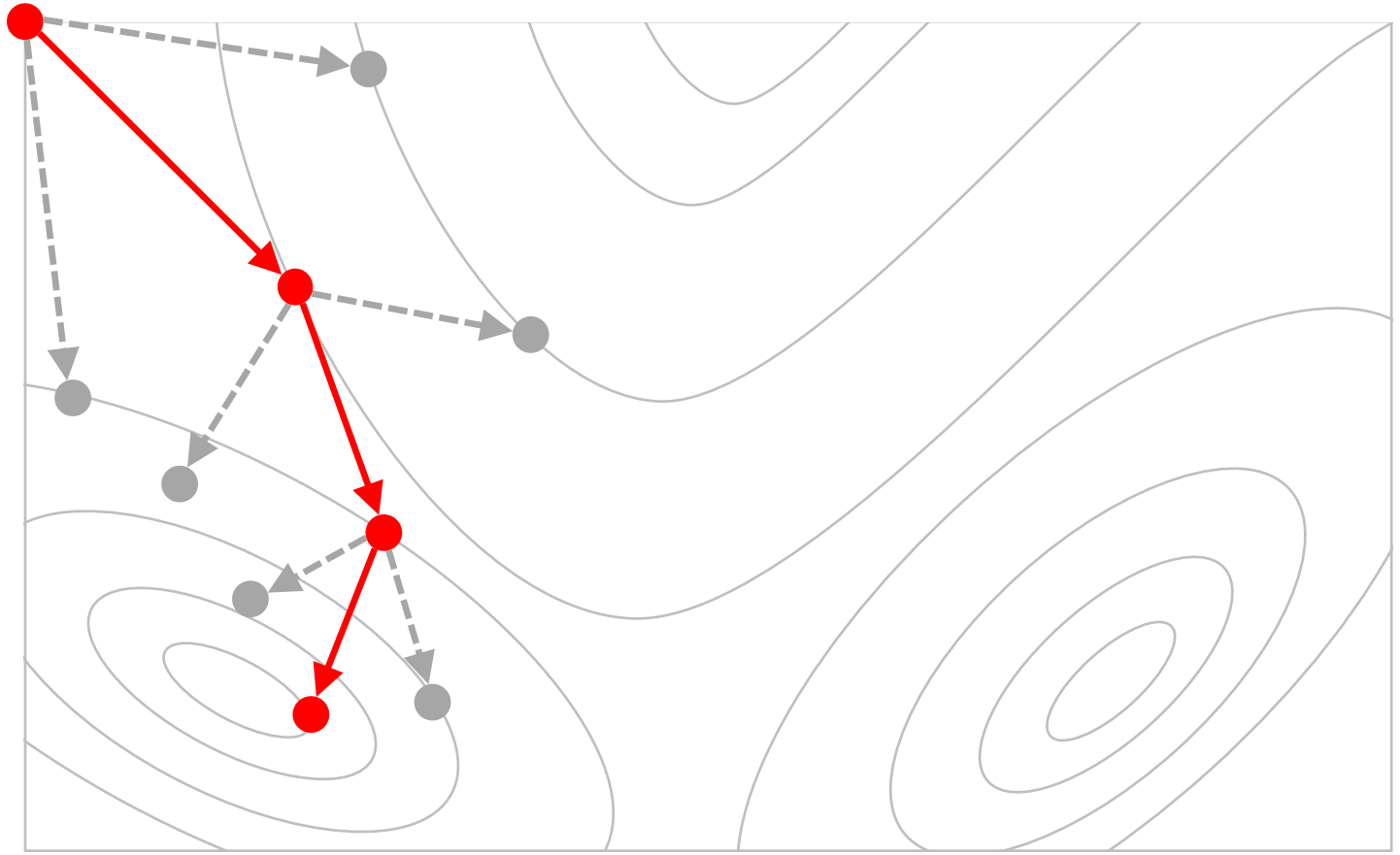
(2) How to synchronise model replicas?



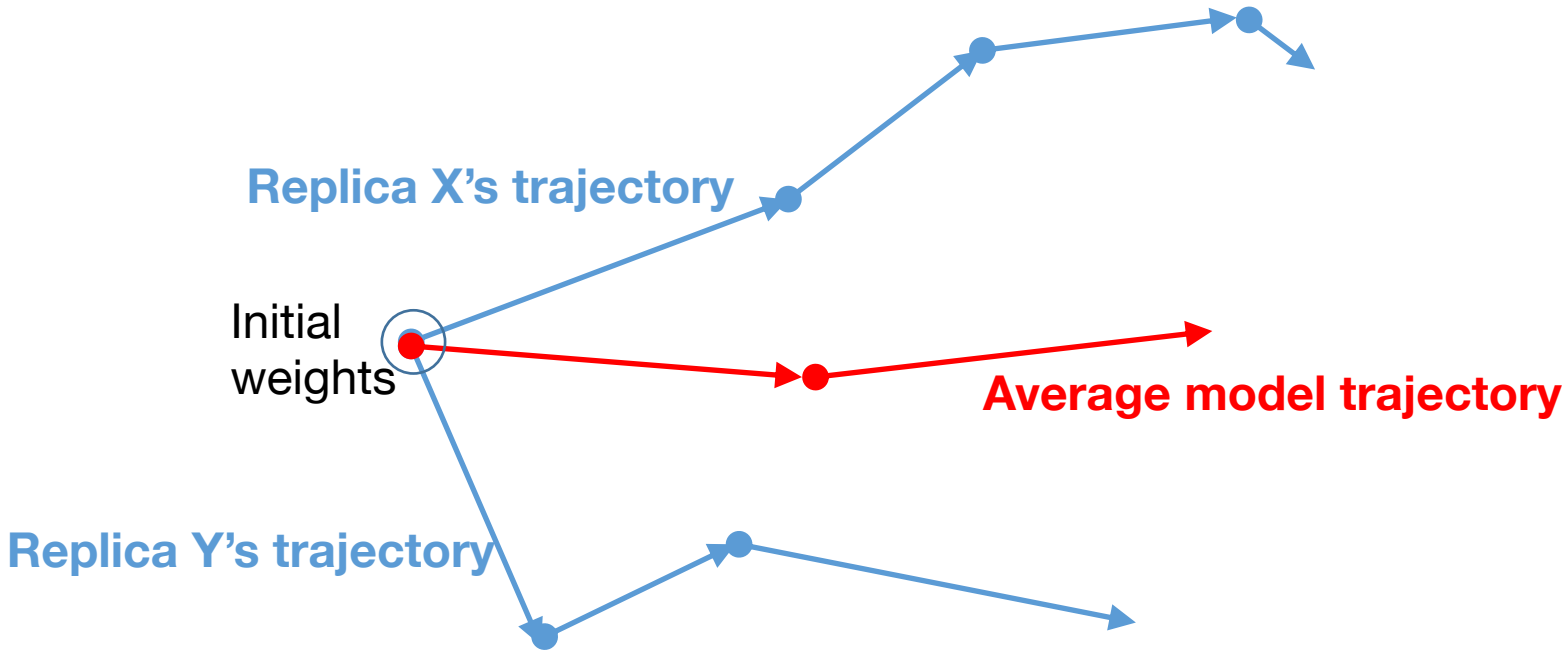
Problem: Why not Synchronous Parallel SGD?

All learners **always** start from the same point

Limited exploration of parameter space



Idea: Maintain Independent Model Replicas



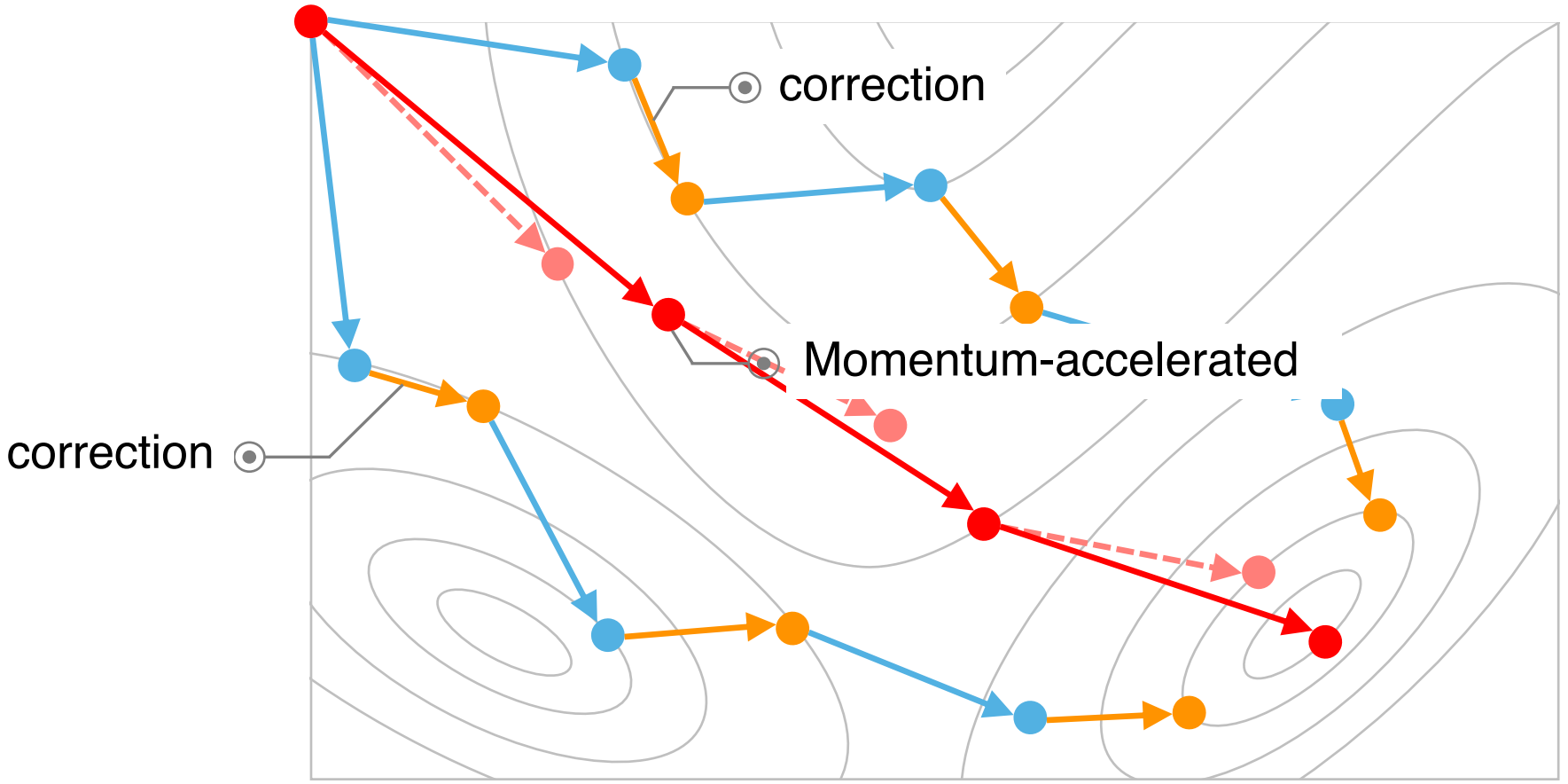
Benefits:

- Increased exploration of space through parallelism
- Each model replica uses small batch size

Crossbow: Synchronous Model Averaging

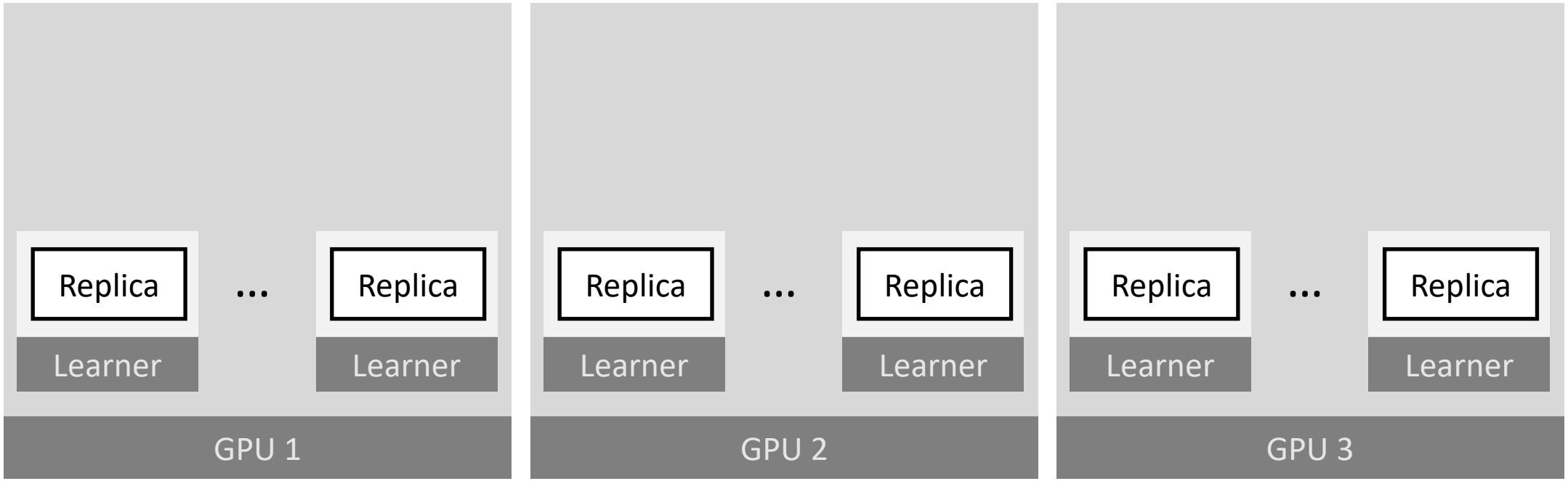
Allow learners to diverge but **correct** trajectories based on **average model**

Accelerate average model trajectory with **momentum** to find minima faster



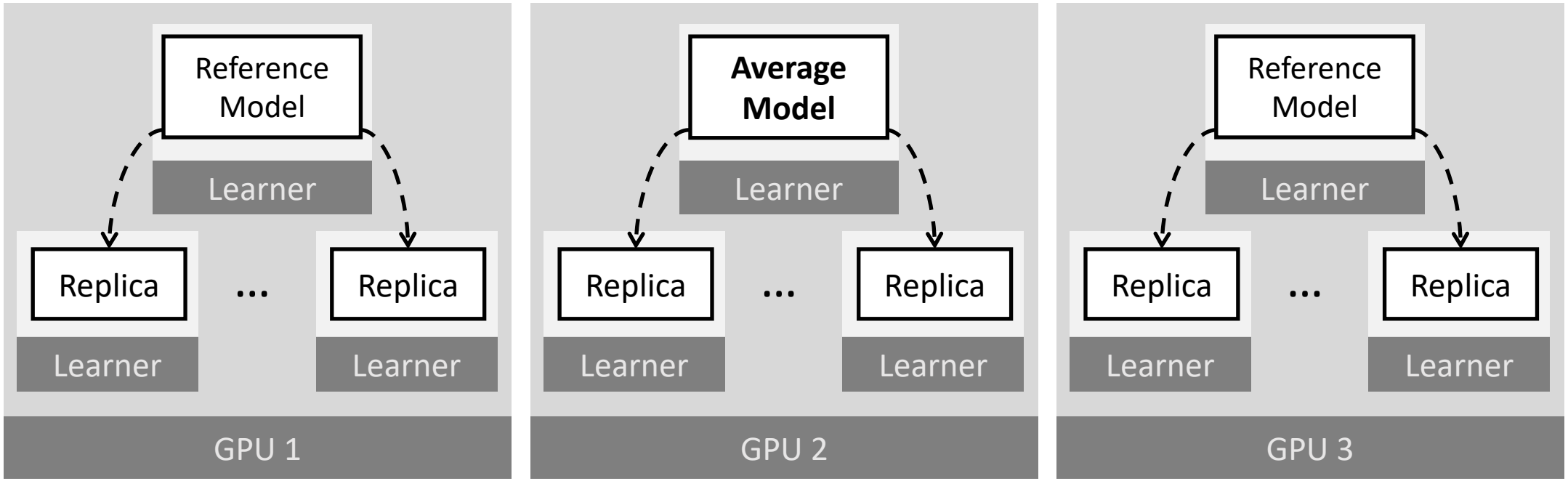
GPUs with Synchronous Model Averaging

Synchronously apply corrections to model replicas



GPUs with Synchronous Model Averaging

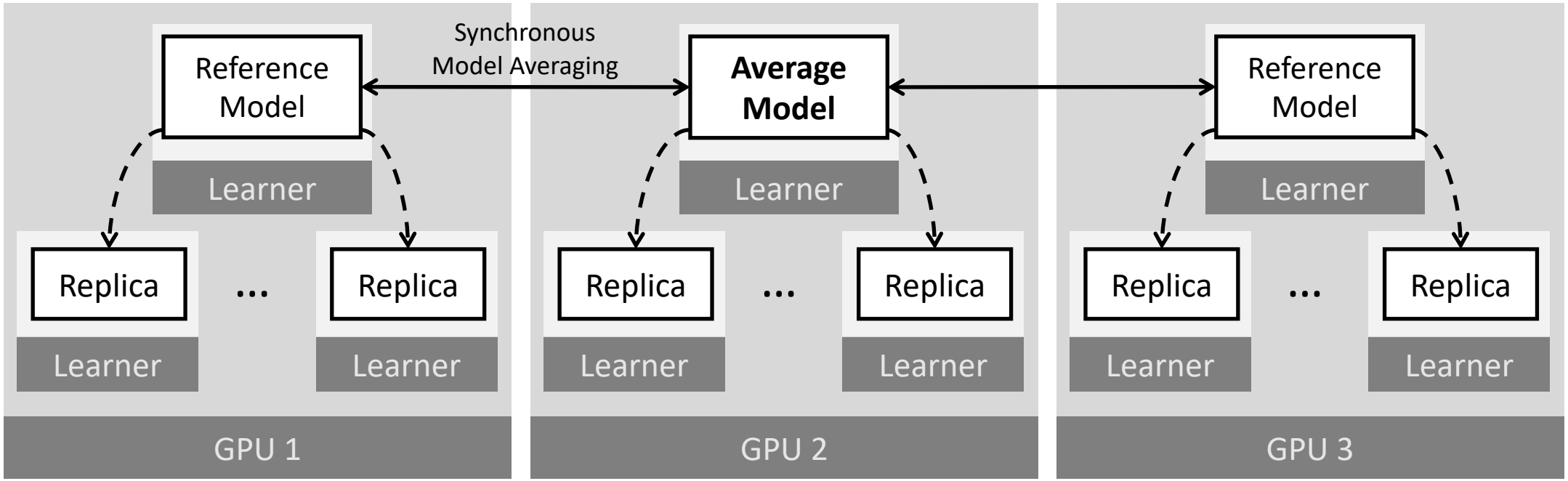
Synchronously apply corrections to model replicas



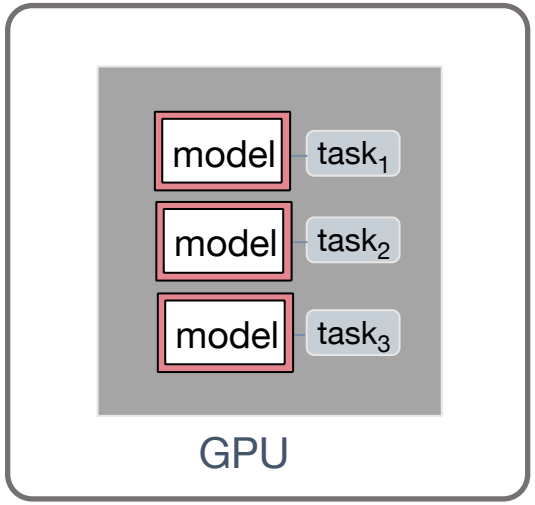
GPUs with Synchronous Model Averaging

Ensures **consistent** view of average model

Takes **GPU bandwidth** into account during synchronisation

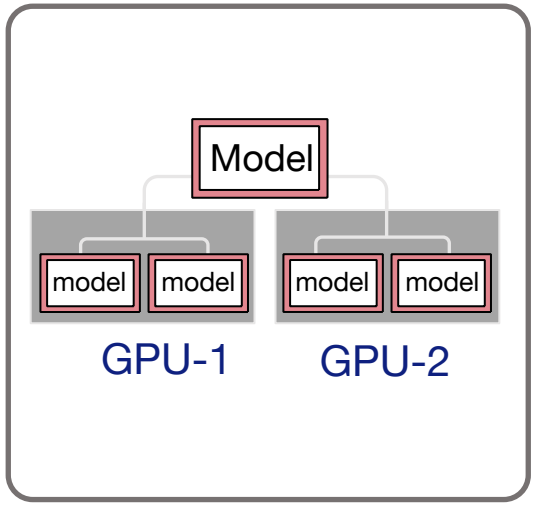


(1) How to increase efficiency with small batches?



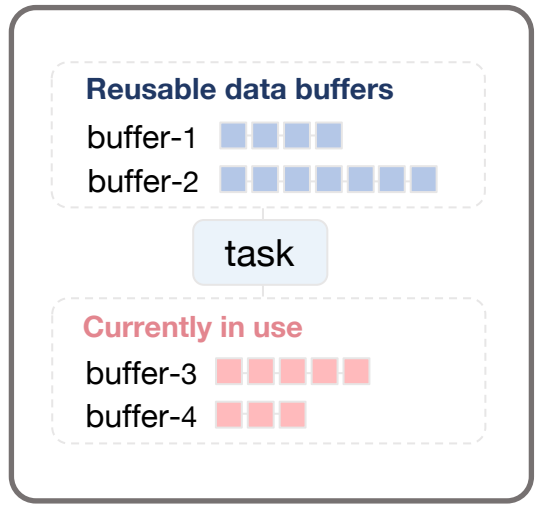
Train multiple model replicas per GPU

(2) How to synchronise model replicas?



Use synchronous model averaging

(3) How to reduce scheduling and synchronisation overheads?

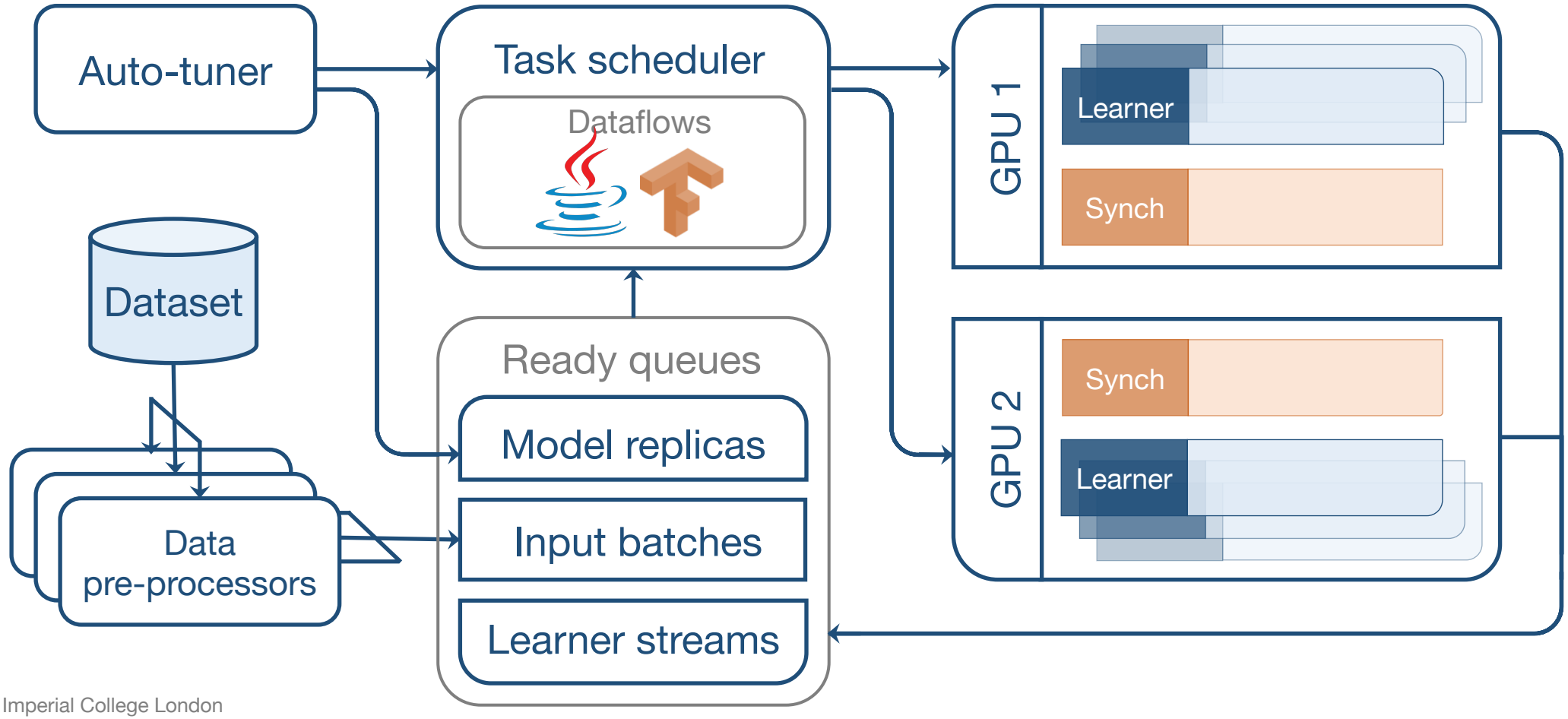


Crossbow Architecture

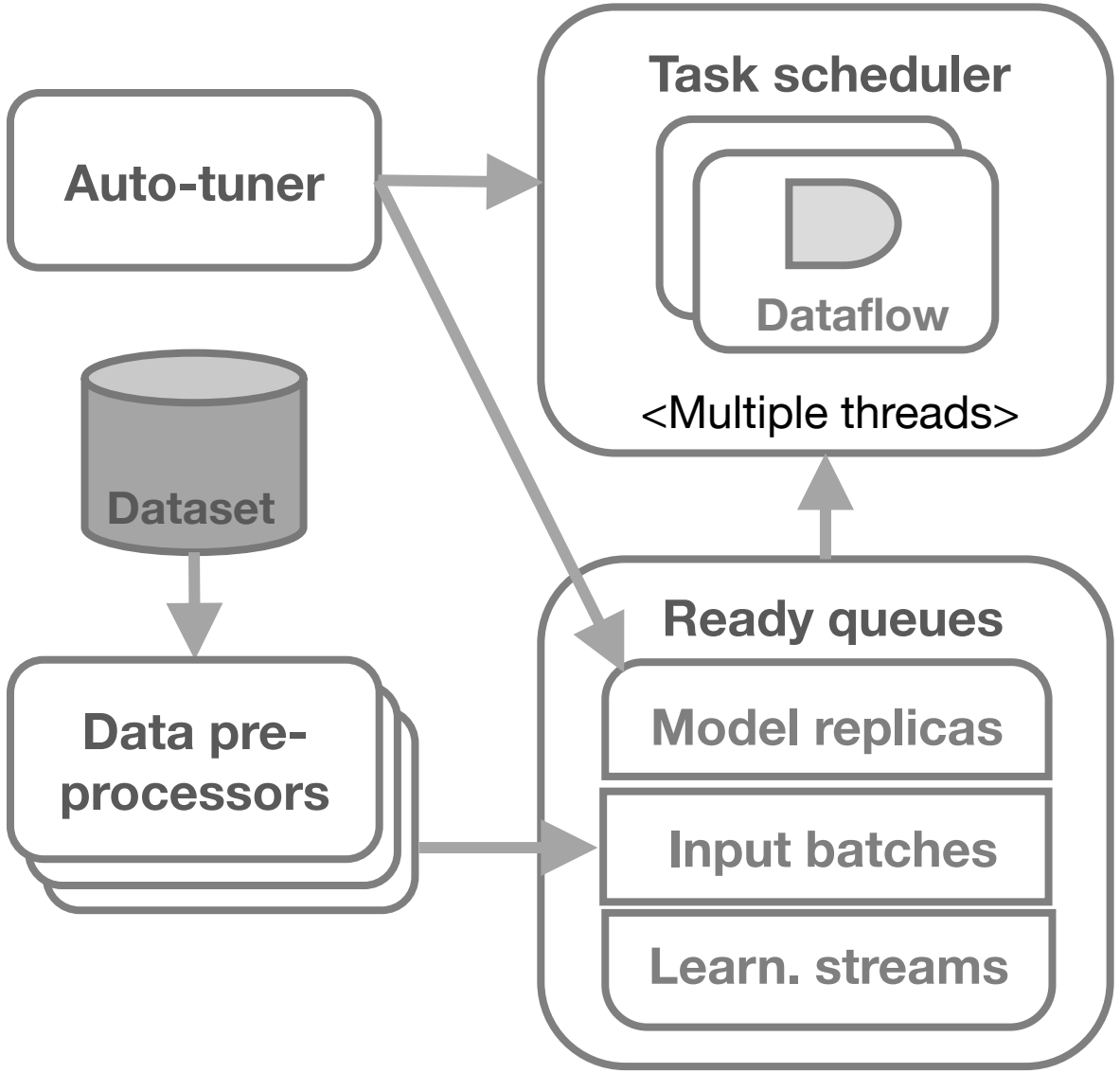
Java 24K LOC

C/C++ 15K LOC

Integration with TensorFlow



Efficient Task Scheduling

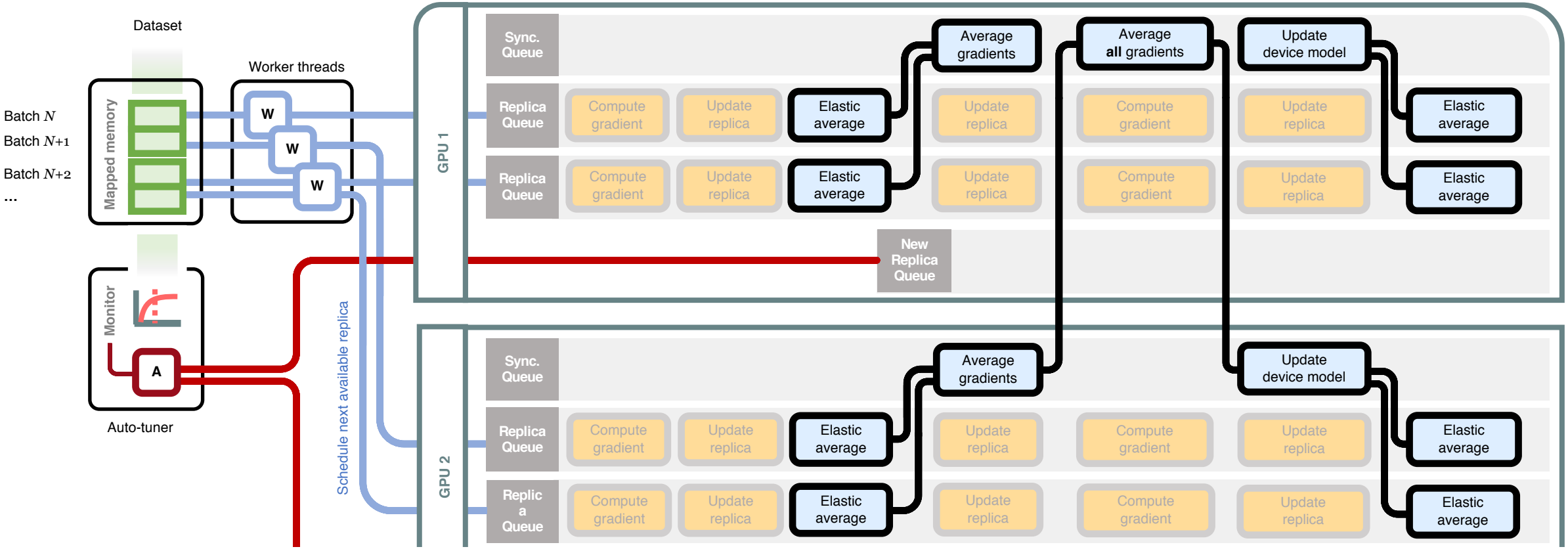


Execute **compute** and **synchronisation** tasks

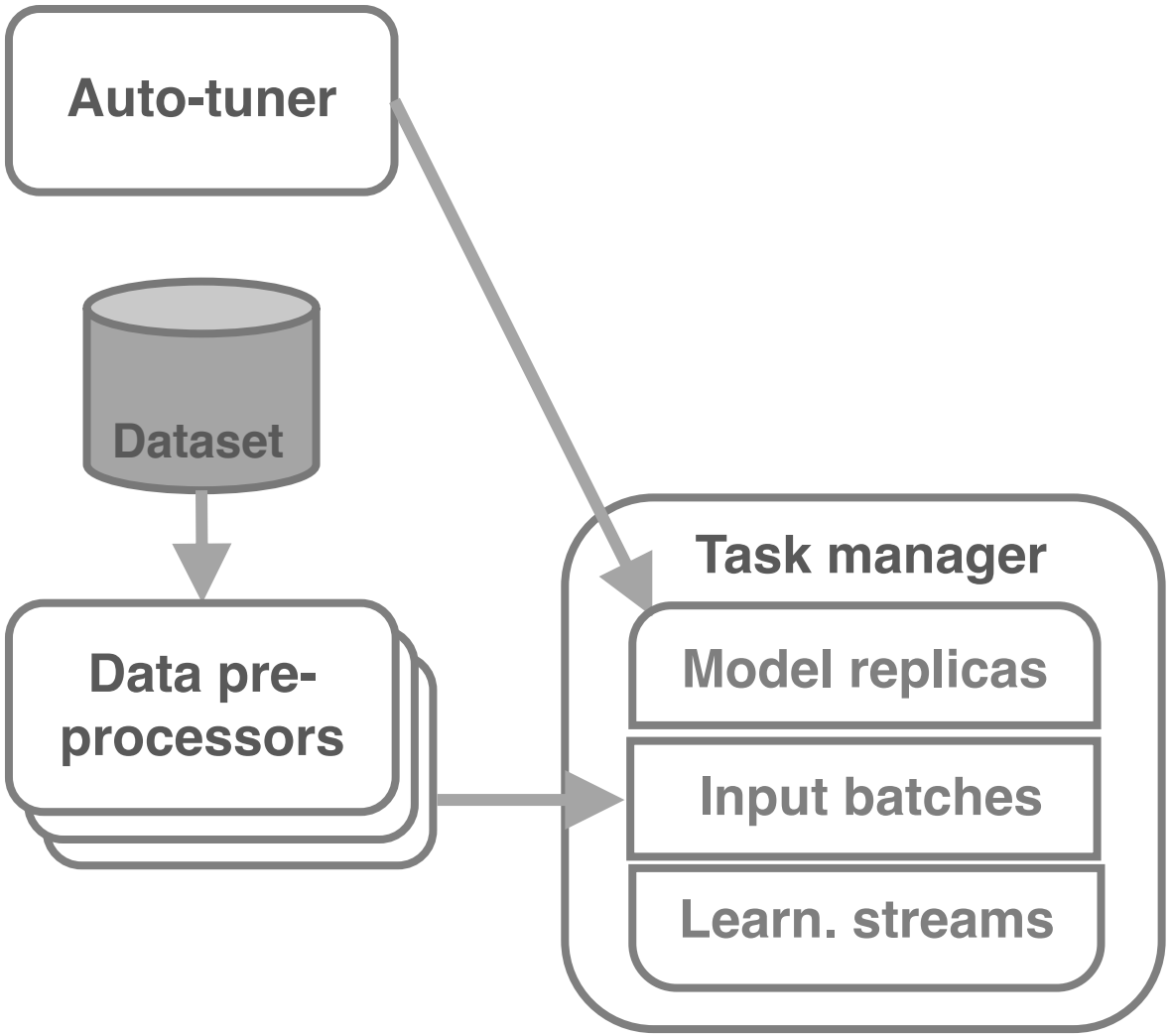
Fine-grained concurrency

Need efficient scheduler to feed all GPUs with tasks

Interleaving Compute & Synchronisation Tasks



Auto-Tuning the Number of Model Replicas



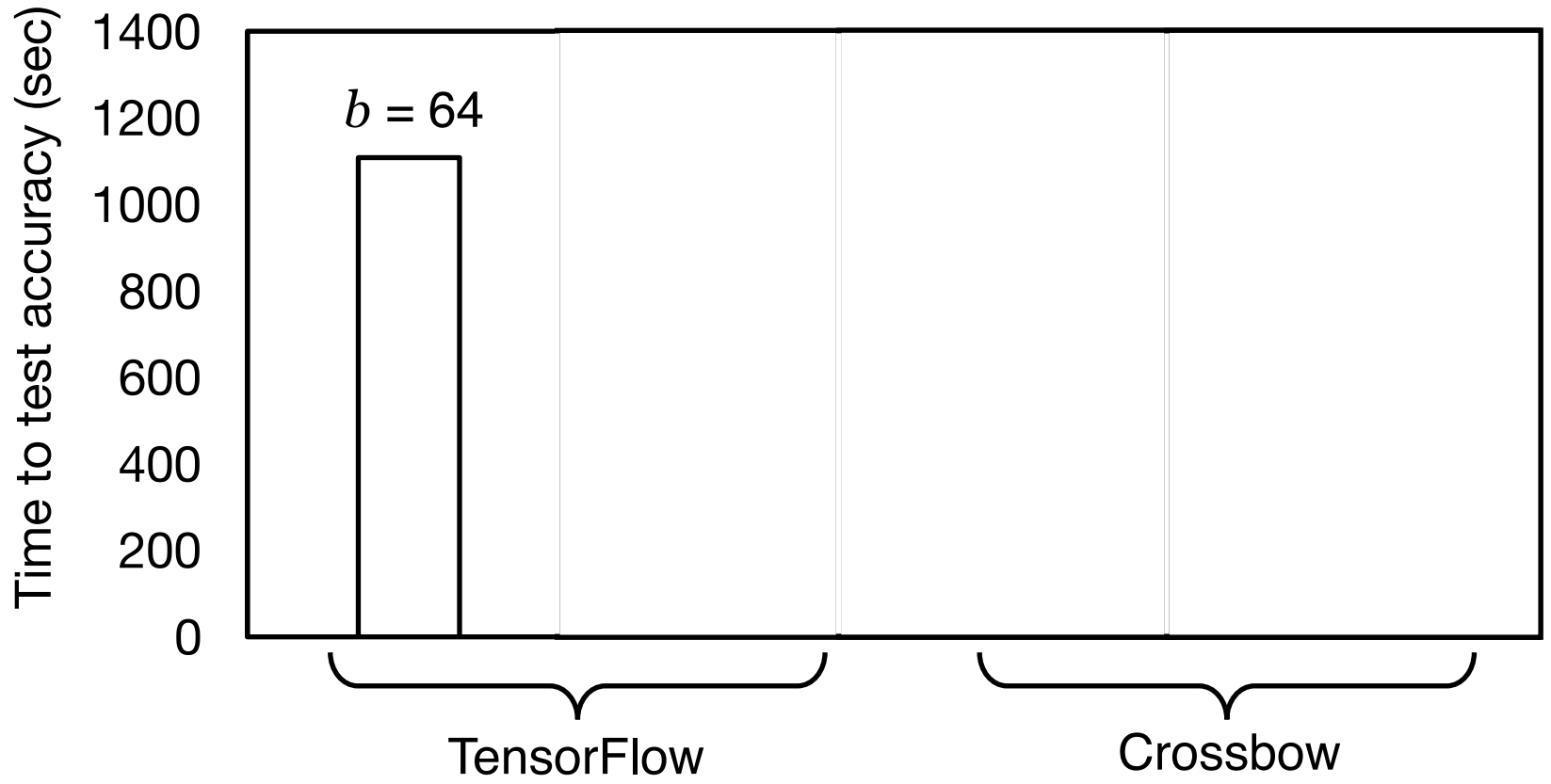
Monitor training throughput

Dynamically adust number of learners

Uses object pooling & lazy materialization

Experimental Evaluation

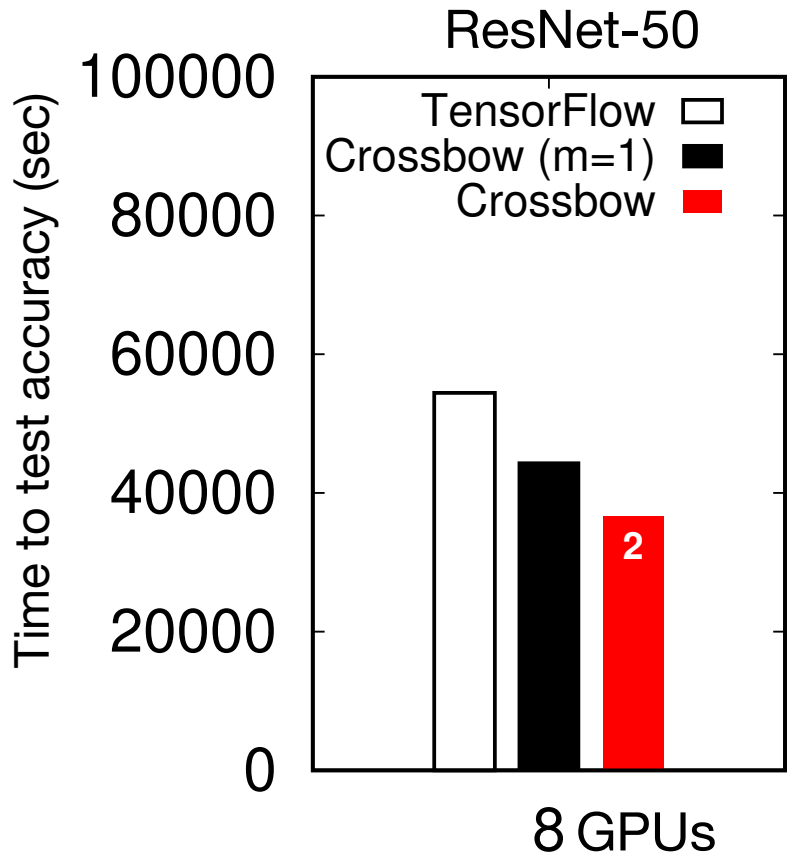
Does Crossbow Train Effectively with Small Batch Sizes?



ResNet-32 with ImageNet dataset on 1 Titan X GPU

Multiple learners per GPU improve hardware efficiency

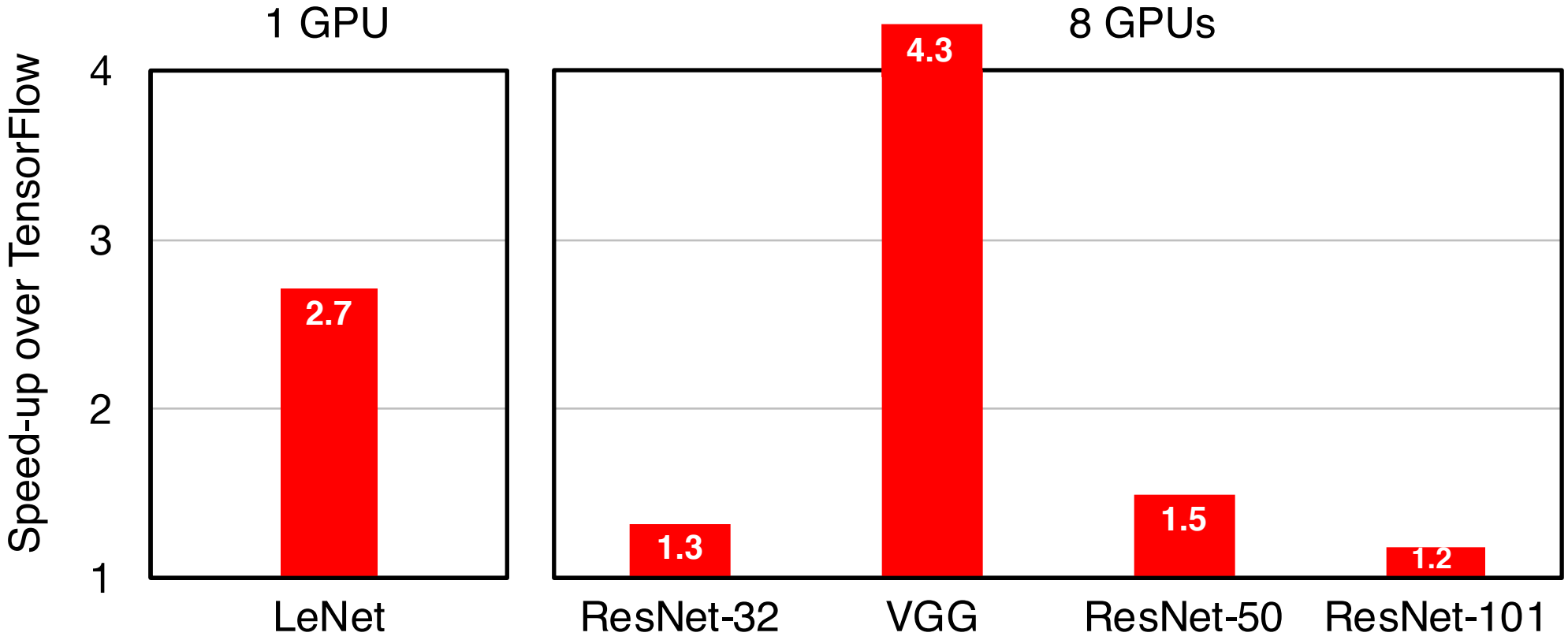
Does Crossbow Scale to multiple GPUs?



ResNet-50 with ImageNet dataset on 8 Titan X GPUs

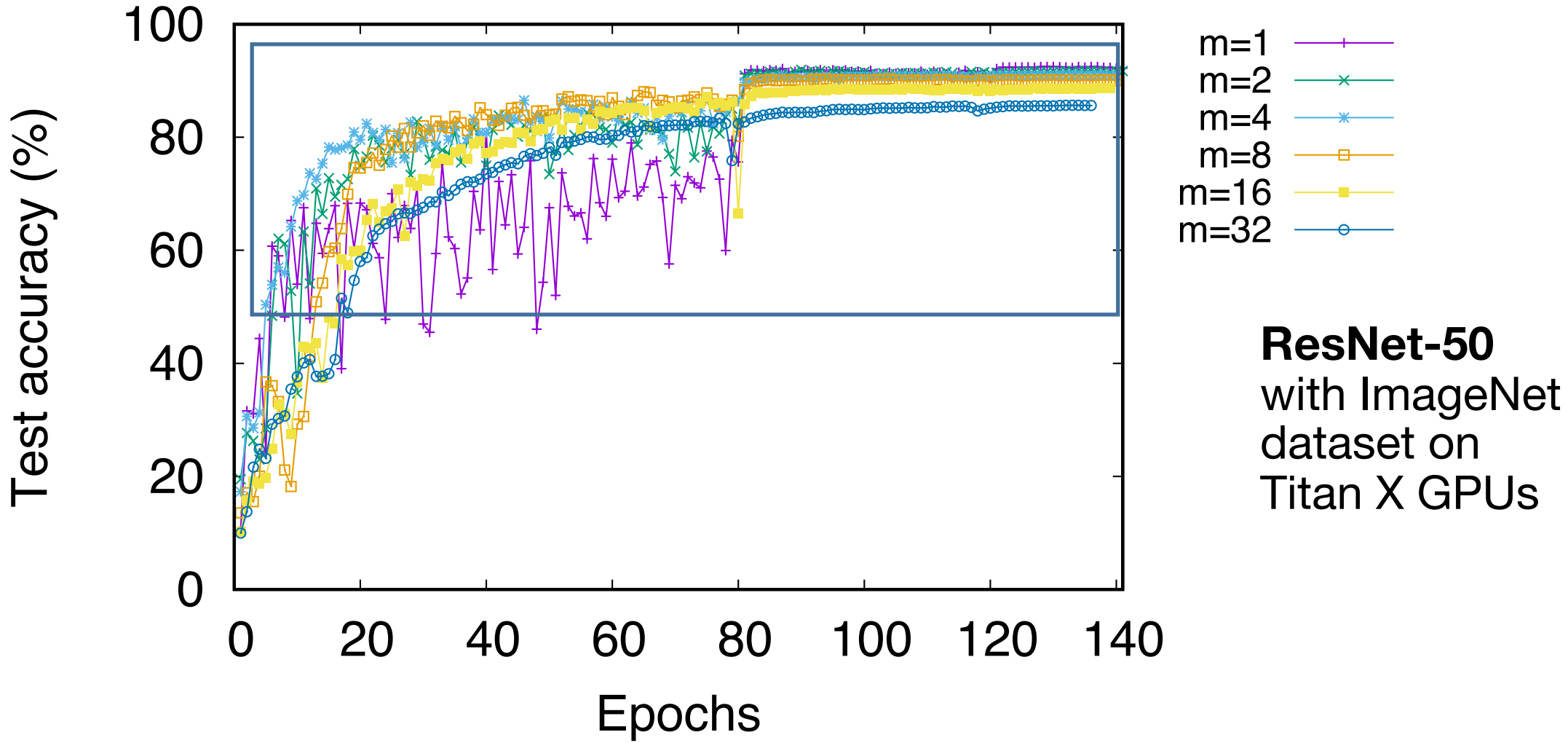
Synchronous Model Averaging improves statistical efficiency

Does Crossbow Train Effectively Across Models?



Training with multiple learners **always better** than training with large batches

What is the Statistical Efficiency with Many Learners?



ResNet-50 with ImageNet dataset on Titan X GPUs

Increasing parallelism has less effect on statistical efficiency

Crossbow: Scaling GPU Deep Learning

github.com/lsds/crossbow

Need to make **training throughput** independent from **hyper-parameters**

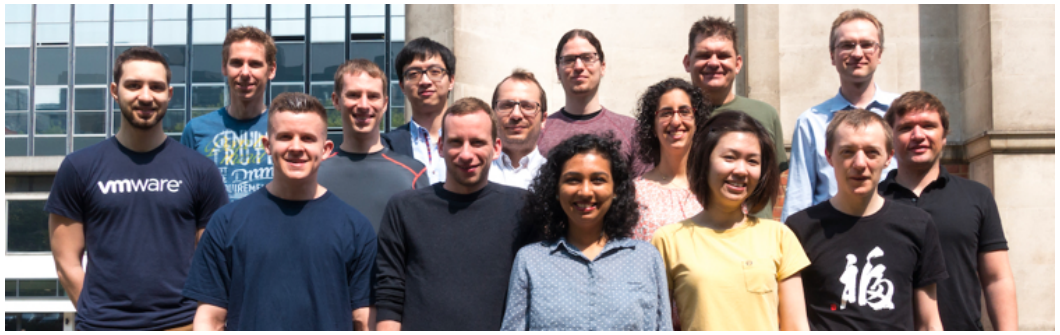
– Rethink the design of future deep learning systems

Crossbow: Scaling DNN training with small batch sizes on many GPUs

– **Multiple model replicas per GPU** for high hardware efficiency

– **Synchronous model averaging** for high statistical efficiency

Exciting research challenges for **next generation deep learning systems**



Thank You — Any Questions?

Peter Pietzuch

https://lsds.doc.ic.ac.uk — prp@imperial.ac.uk