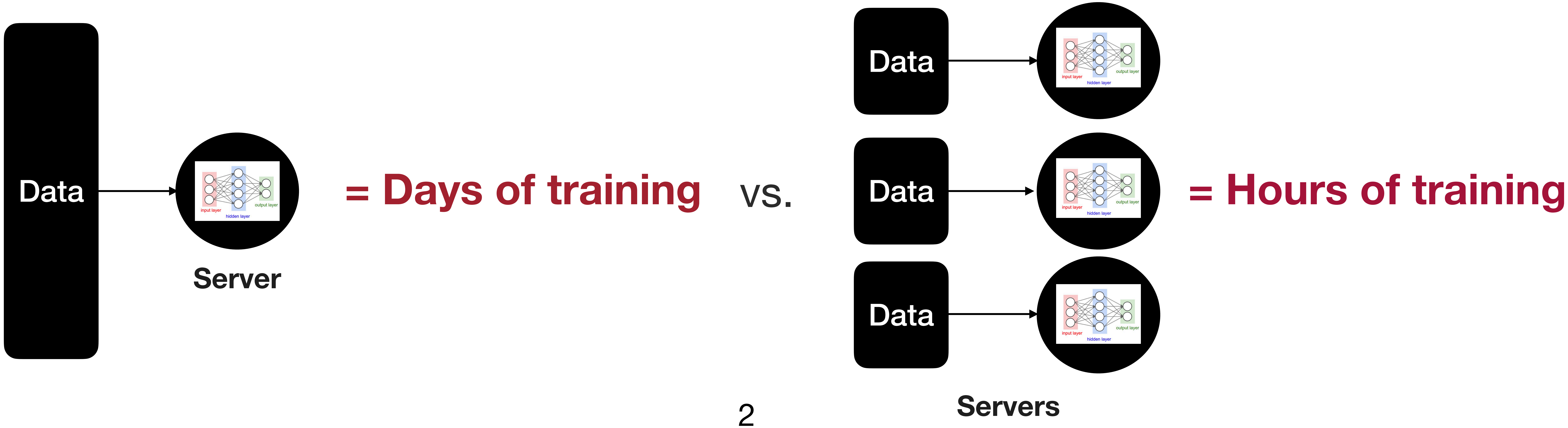
Taming Resource Heterogeneity in Distributed ML training with Dynamic Batching

Sahil Tyagi and Prateek Sharma
Intelligent Systems Engineering,
Luddy School of Informatics, Computing and Engineering,
Indiana University Bloomington, USA

Why is Distributed Training important ?

- Advances in deeper and complex machine learning models increases computational needs. E.g., ResNet, VGG, Transformers, BERT
- State of the art models train on billions of parameters, increasing training time. E.g., GPT-3 has 175 billion parameters!
- Distributing training across multiple servers decreases model training time

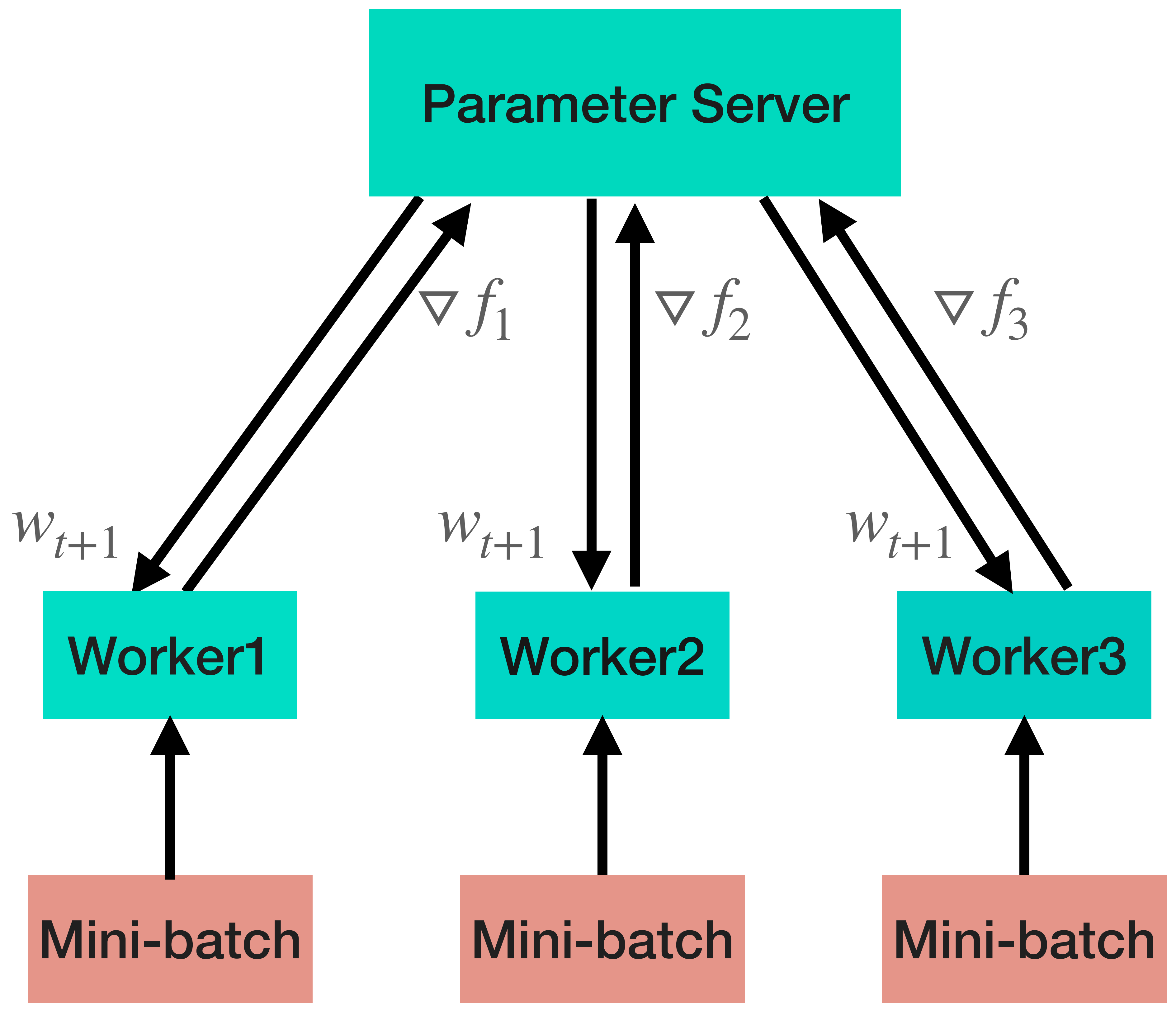


Distributed Training with Parameter Servers

- Workers: servers iteratively train model on a mini-batch of data independently (data parallelism)
    - Gradients computed using optimization like Mini-batch Stochastic Gradient Descent (SGD)
- Parameter Servers: compute mean of gradients from workers for given step; apply updates and re-distribute updated weights
    - Gradients applied to weights and resume training to next step; process repeats again
- Synchronous/BSP: Training at every step is halted until all gradient updates are received

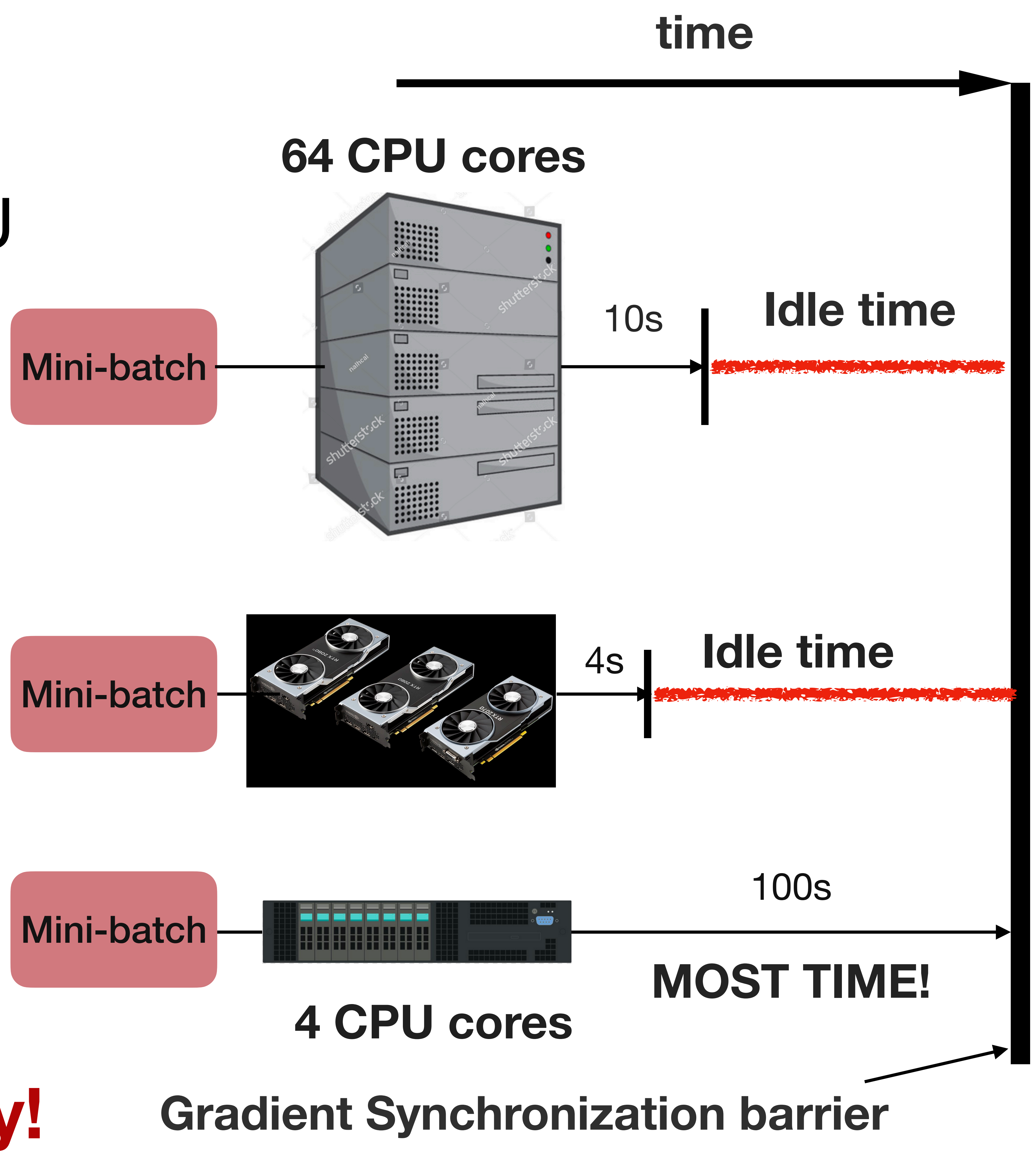
$$\nabla f = mean(\nabla f_1, \nabla f_2, \nabla f_3)$$

$$w_{t+1} = w_t - \lambda \nabla f$$



What Happens if Cluster is Heterogeneous ?

- Heterogeneity: Servers in a cluster with different compute capabilities.
  - E.g., NVIDIA GPU, 64-core CPU and 4-core CPU
  - Omnipresent; be it shared cloud or data centers
- Iteration time: time taken by each worker to compute gradients on a mini-batch
- Iteration time is lower on more capable servers and vice versa



Synchronization increase training time significantly!

Impact of Heterogeneity on Training Time

- We test a broad spectrum of test workloads on two clusters with three workers each (with same cumulative CPU cores)

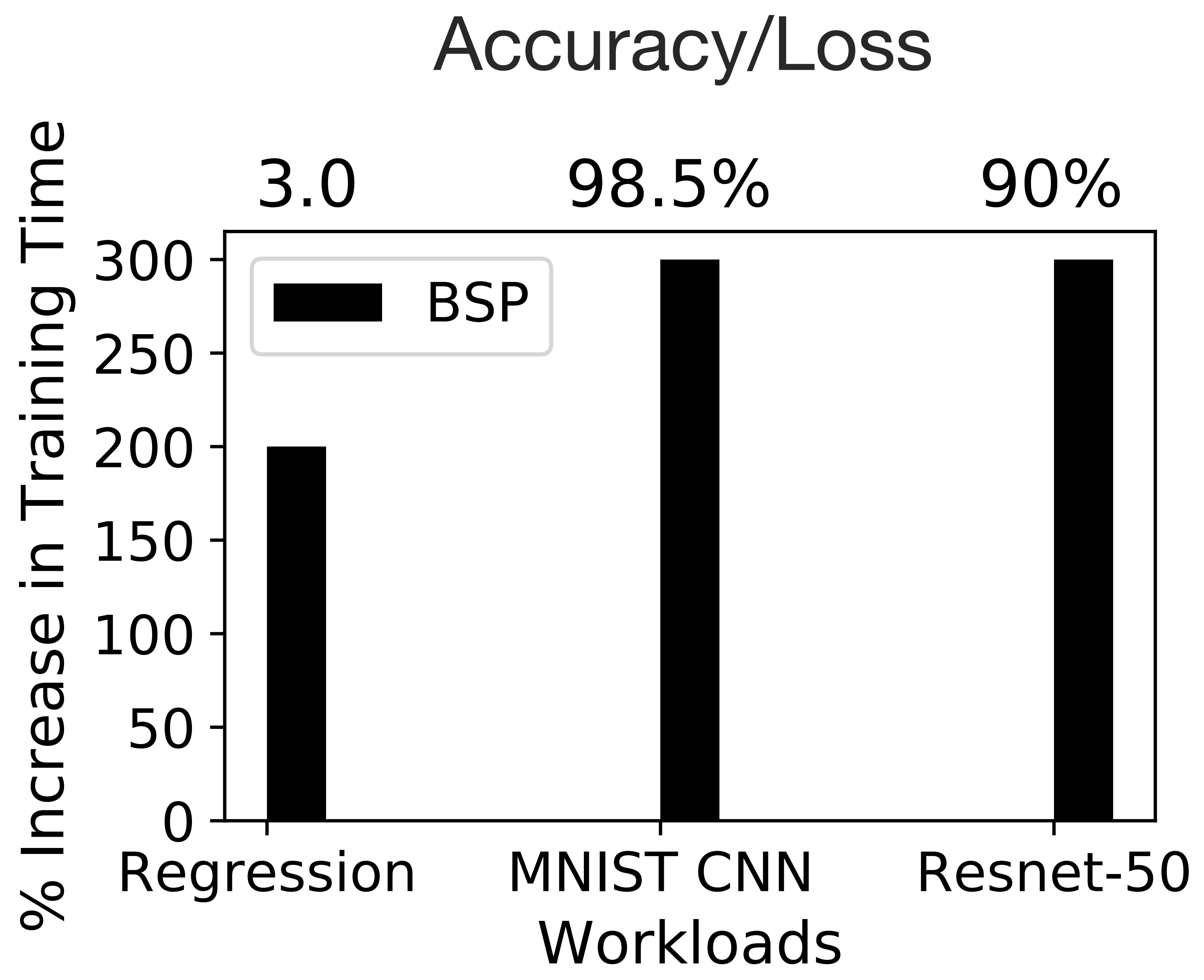
Homogeneous

13 CPU cores | 13 CPU cores | 13 CPU cores = 39 cores

vs.

Heterogeneous

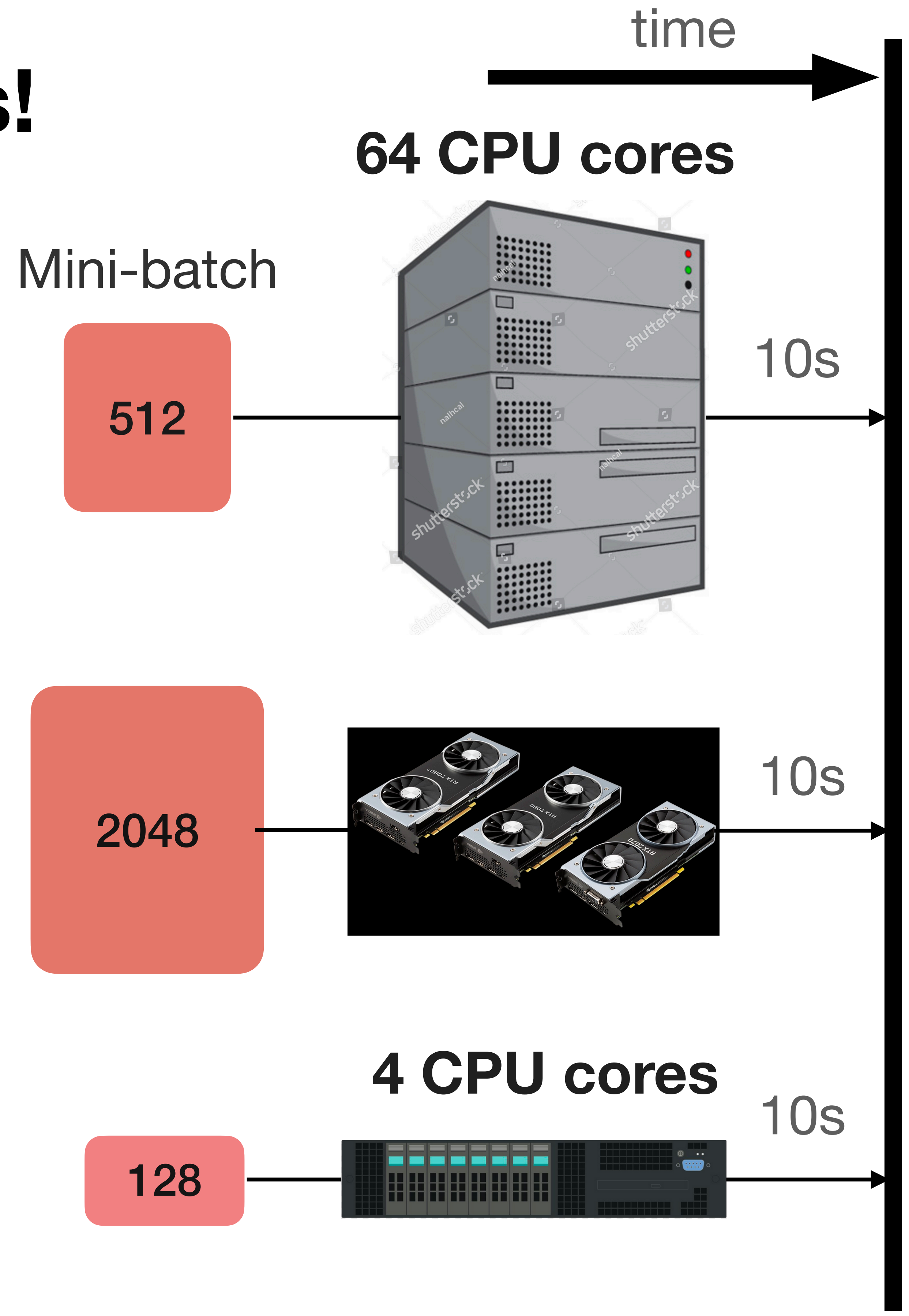
11 CPU cores | 4 CPU cores | 24 CPU cores = 39 cores



ResNet-50 and 1-layer CNN take 3x time in a heterogeneous cluster!
Linear Regression takes 2x more time than a homogeneous cluster!

How to Minimize the Effect of Heterogeneity ?

- **Key idea: Equalize the iteration times among the workers!**
- Assign mini-batch size on workers proportional to worker throughput
- Throughput ratio in clusters is approximated as:
  - On CPU clusters: ratio of CPU-core count
  - On CPU-GPU mix and GPU clusters: ratio of floating point operations per second (FLOPS)



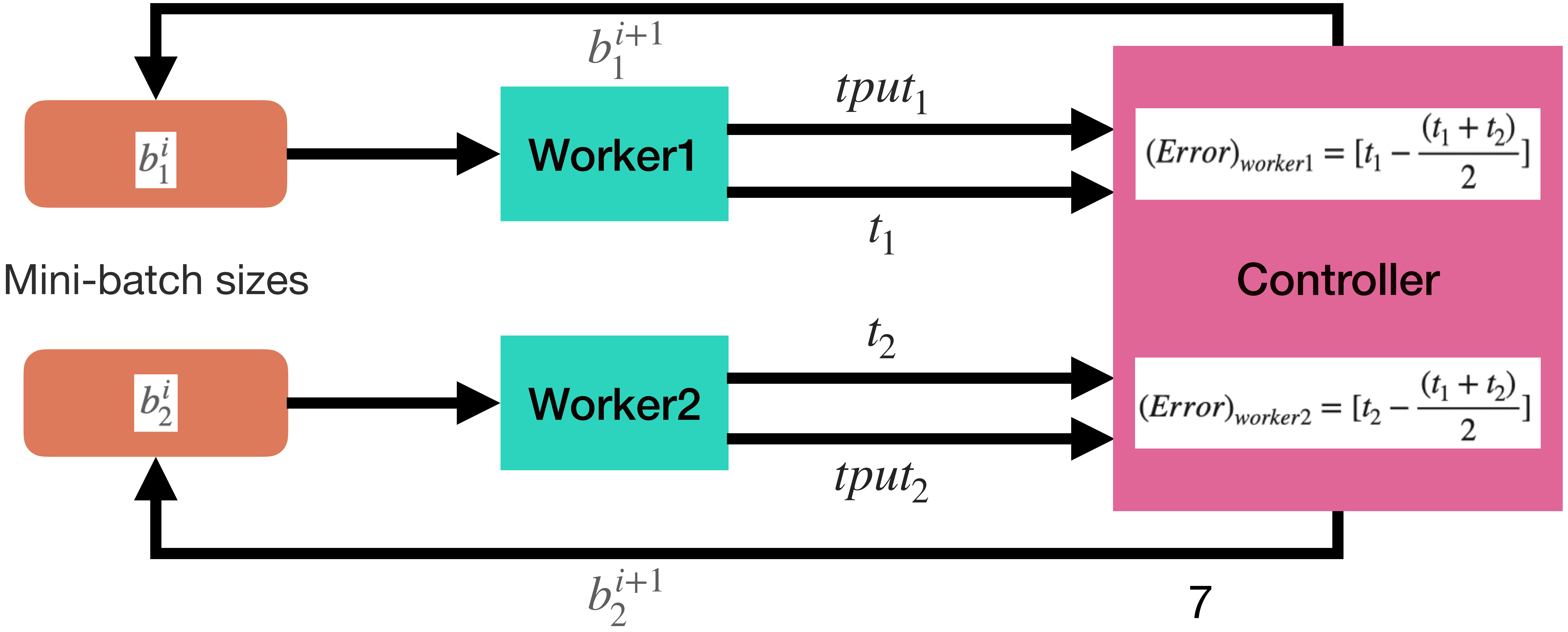
Fast worker; big mini-batch size
Slow worker; small mini-batch size

Near-equal time on all workers

Mini-batch Size Controller

- A one-time throughput approximated mini-batch adjustment not enough:
  - Approximated throughput is different from actual training throughput
  - Server resources may change dynamically during training (interference, overcommitment etc.)
- We use a proportional controller that $b_k^{i+1} = b_k^i + \triangle(b_k^i)$

$$\triangle(b_k) = -(Throughput \times Error) \quad such\ that \quad Error = (worker's\ iteration\ time \; - \; cluster's\ average\ iteration\ time)$$



If Error < 0, increase mini-batch.

If Error > 0, decrease mini-batch.

- In the two worker cluster, for worker1

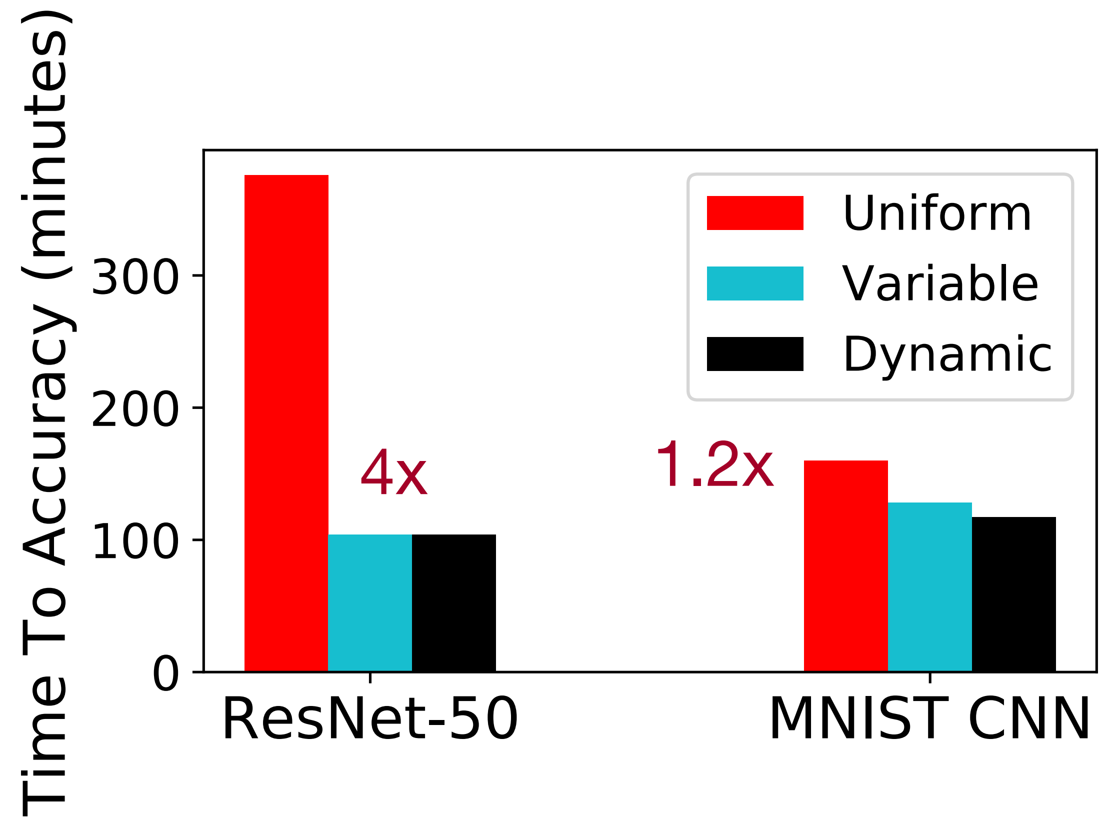
$$\triangle(b_1) = -\,tput_1 \times [t_1 - \frac{(t_1+t_2)}{2}]$$

$$b_1^{i+1} = b_1^i + \triangle(b_1^i)$$

Training on Heterogeneous GPU Clusters

What happens when a cluster contains different types of CPU and GPU workers?

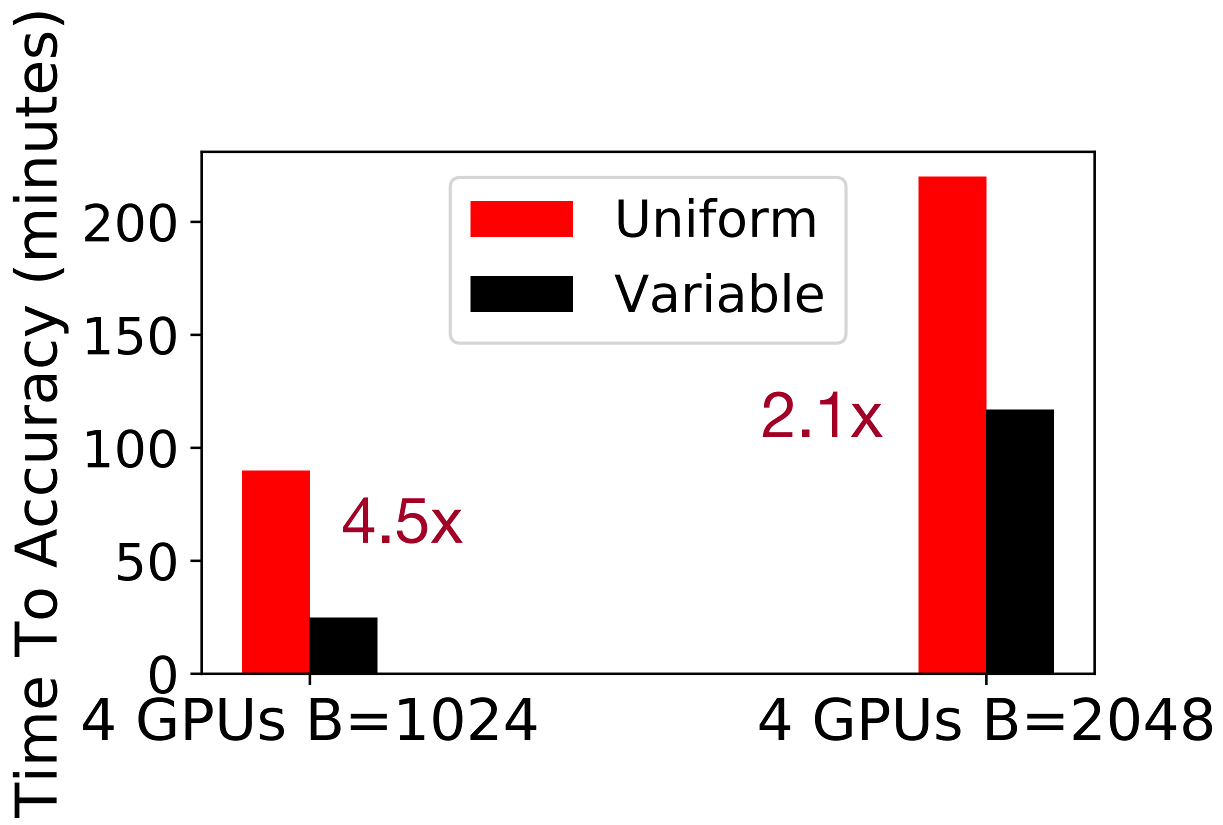
- We use a two worker cluster with NVIDIA Tesla P100 and 48-core Intel Xeon CPU as workers
- Evaluated on conventional TF, hardware throughput based and controller based dynamic mini-batch adjustment



ResNet50 improves by 4x and CNN improves by 1.2x in CPU-GPU mix

What happens when a cluster contains different types of GPU workers only ?

- We test on two NVIDIA Tesla T4 and two NVIDIA Tesla P4



ResNet50 improves by 4.5x and CNN improves by 2.1x in GPU clusters

Thank you!