ECE826: Part 2

So far we talked about

- Generalization: why would expect our models to work
- SGD/GD: under what conditions do they work
- Why can we optimize neural networks?

Not a lot of algorithmic design principles for large-scale learning

Advances and Challenges in Distributed Machine Learning

We'll see a lot of principles for scaling up, and designing the "plumbing" of deep learning systems

Overview

- Multicore vs. Distributed
- Algorithms of choice
- Open challenges with Performance Gains/Analysis
- Communication bottlenecks
- Straggler Nodes
- Robustness

Stochastic Gradient Descent

$$\min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} \ell(\mathbf{w}; \mathbf{z}_i)$$

loss for data point i

- Idea ('50s, '60s [Robbins, Monro], [Widrow, Hoff]):
  Sample a data point + locally optimize.

SGD: An *Über*-algorithm

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \gamma \cdot \nabla \ell(\mathbf{w}_k; \mathbf{z}_{i_k})$$

Stochastic Gradient Descent

Different names and flavors

ML / Optimization / Statistics / EE

Perceptron

Incremental Gradient

Back Propagation (NNs)

Oja's iteration (PCA)

LMS Filter

...

Has been around for a while, for good reasons:

- Robust to noise
- Simple to implement
- Near-optimal learning performance *
- Small computational foot-print

Stochastic Gradient Descent

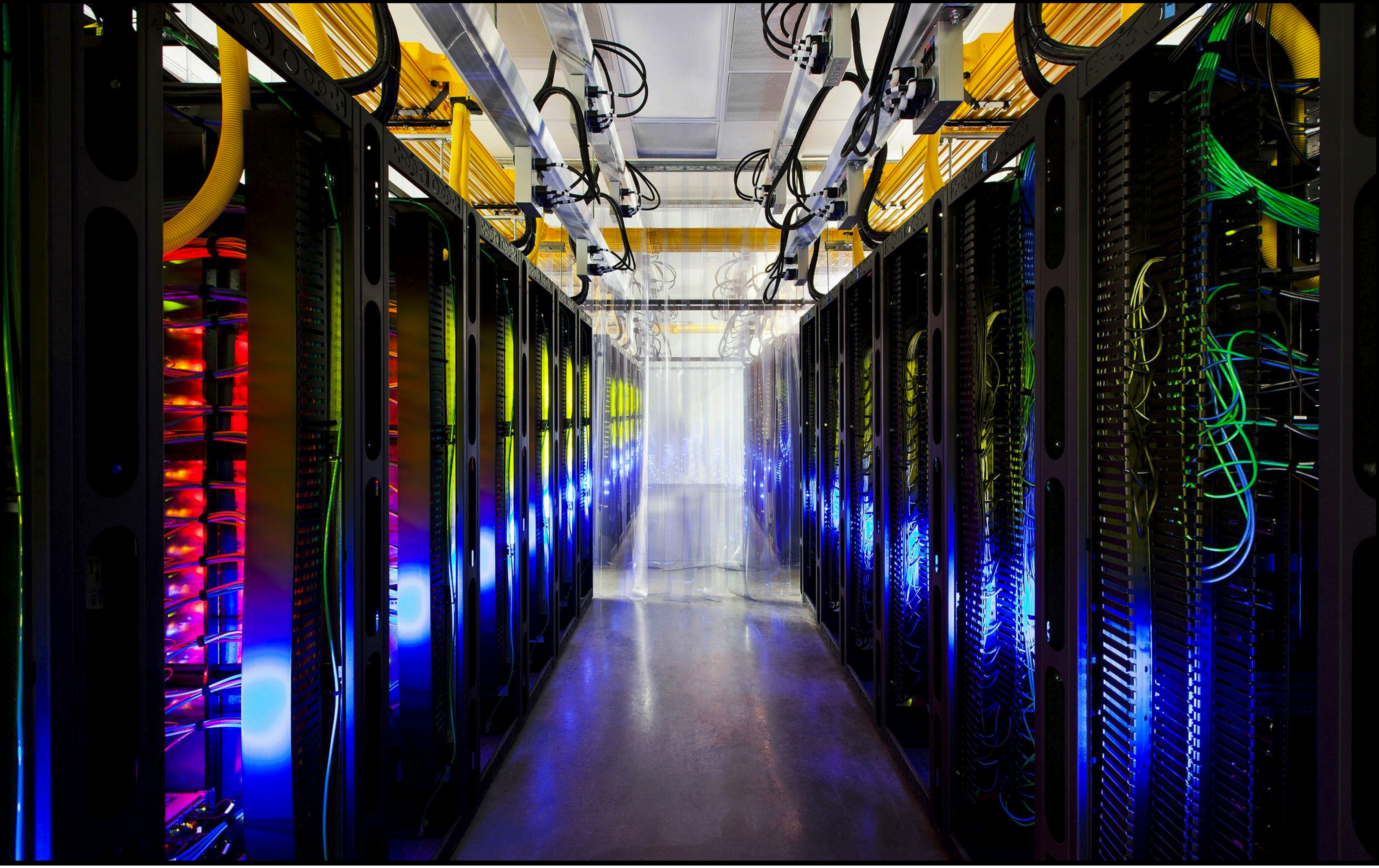
GPT-3 would take 288 years to train on a single Tesla V100 GPU

[source: https://arxiv.org/pdf/2104.04473.pdf]

Goal:

Speed up Machine Learning

Idea:
Train at scale



System Setup

Parallel vs. Distributed

- Parallel (CPU/multicore GPU)
    - Single machine, many cores (usually up to 10-100s)
    - Shared memory (all cores have access to RAM)
    - Comm. to RAM is cheap

- Distributed
    - Many machines (usually up to100-1000s) connected via network
    - Shared-nothing architecture (each node has its own resources)
    - Communication costs non-negligible

Scaling up vs. Scaling out

Scaling up vs. out

<u>What we'd ideally like</u>

• Cores	∞
• RAM	∞
• Comm. Cost	0
• Cost to build	0

<u>Feasible solutions:</u>

Scaling up:

- Getting the largest machine possible, with maxed out RAM

Scaling out:

- Getting a bunch of machines, and linking them together

Scaling up vs. out

Scaling up

Pros:
RAM comm. cheap (no network)
Less impl. overheads
Less power/smaller footprint

Cons:
100s cores/machine = expensive
Smaller fault tolerance
Limited upgradability

Scaling out

Pros:
Much cheaper (especially on Ec2)
Can replace faulty parts
Better fault tolerance (if it matters)

Cons:
Network bound
Major implementation overheads
Large power footprint

Should I buy or rent?

Price Comparisons for 4 GPUs racks

- Sin
- Alt

Flops/$ (1-yr analysis with 50% occupancy)

	Hyperplane-A100 (25% annual depreciation)	Hyperplane-A100 (100% annual depreciation)	Scalar-A100 (25% annual depreciation)	Scalar-A100 (100% annual depreciation)	p4d (qll upfront)	p4d (partial upfront)	p4d (no upfront)	p4d (on demand)
Total Cost 1-yr	$55,534	$160,534	$47,279	$138,779	$164,955	$168,321	$176,737	$143,544
Total Petaflops 1-yr	2,459,808	2,459,808	2,459,808	2,459,808	2,459,808	2,459,808	2,459,808	2,459,808
Petaflops/$	44.3	15.3	52.0	17.7	14.9	14.6	13.9	17.1

web services™ | EC2

Google Cloud Platform

Microsoft Azure

source https://lambdalabs.com/blog/tesla-a100-server-total-cost-of-ownership/

Price Comparisons for 4 GPUs racks

- Sin
- Alt

Flops/$ (1-yr analysis with 50% occupancy)

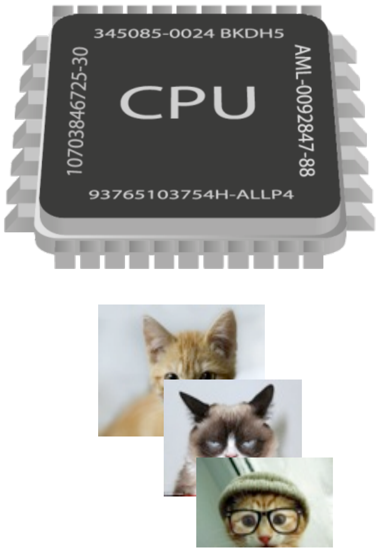
	Hyperplane-A100 (25% annual depreciation)	Hyperplane-A100 (100% annual depreciation)	Scalar-A100 (25% annual depreciation)	Scalar-A100 (100% annual depreciation)	p4d (qll upfront)	p4d (partial upfront)	p4d (no upfront)	p4d (on demand)
Total Cost 1-yr	$55,534	$160,534	$47,279	$138,779	$164,955	$168,321	$176,737	$143,544
Total Petaflops 1-yr	2,459,808	2,459,808	2,459,808	2,459,808	2,459,808	2,459,808	2,459,808	2,459,808
Petaflops/$	44.3	15.3	52.0	17.7	14.9	14.6	13.9	17.1

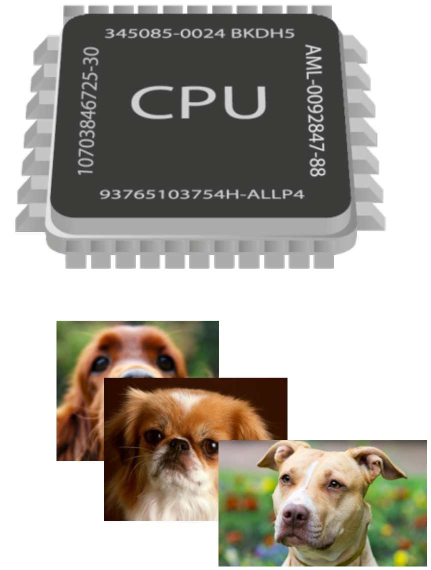
Doesn't sound fun

source https://lambdalabs.com/blog/tesla-a100-server-total-cost-of-ownership/

How to distributed the compute effort?

Distribute the effort!





Several issues

Theory

- Many models weaker than one
- Delays and Slow Nodes
- Communication Costs

Practice

- Barriers to entry / High Cost
- Implementation Overhead
- Nontrivial choice of ML-framework

How to Parallelize

$$w_{k+1} = w_k - \gamma \nabla \ell_{s_k}(w_k; x_i)$$

- Parallelize computation of one update?

  Issue:

  computing $\nabla \ell_i(w; x_i)$ cheap (even for deep nets) [O(d)]

- Parallel Updates?

  Issue: SGD is inherently serial….

Q: Can we parallelize inherently serial algorithms?

Simple idea: mini-batch SGD

- Compute multiple gradients in parallel

$$w_{k+1} = w_k - \gamma \nabla \ell_{s_k^1}(w_k; x_i)$$

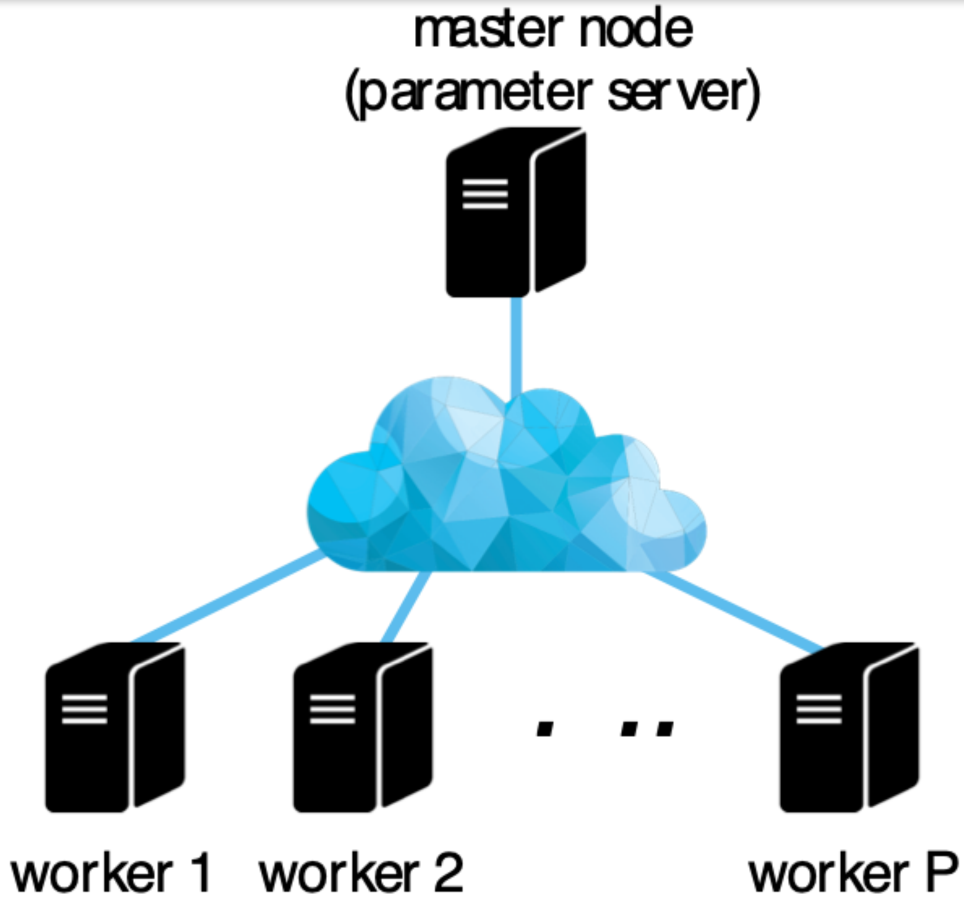
$$w_{k+1} = w_k - \gamma \nabla \ell_{s_k^2}(w_k; x_i)$$

$$w_{k+1} = w_k - \gamma \nabla \ell_{s_k^3}(w_k; x_i)$$

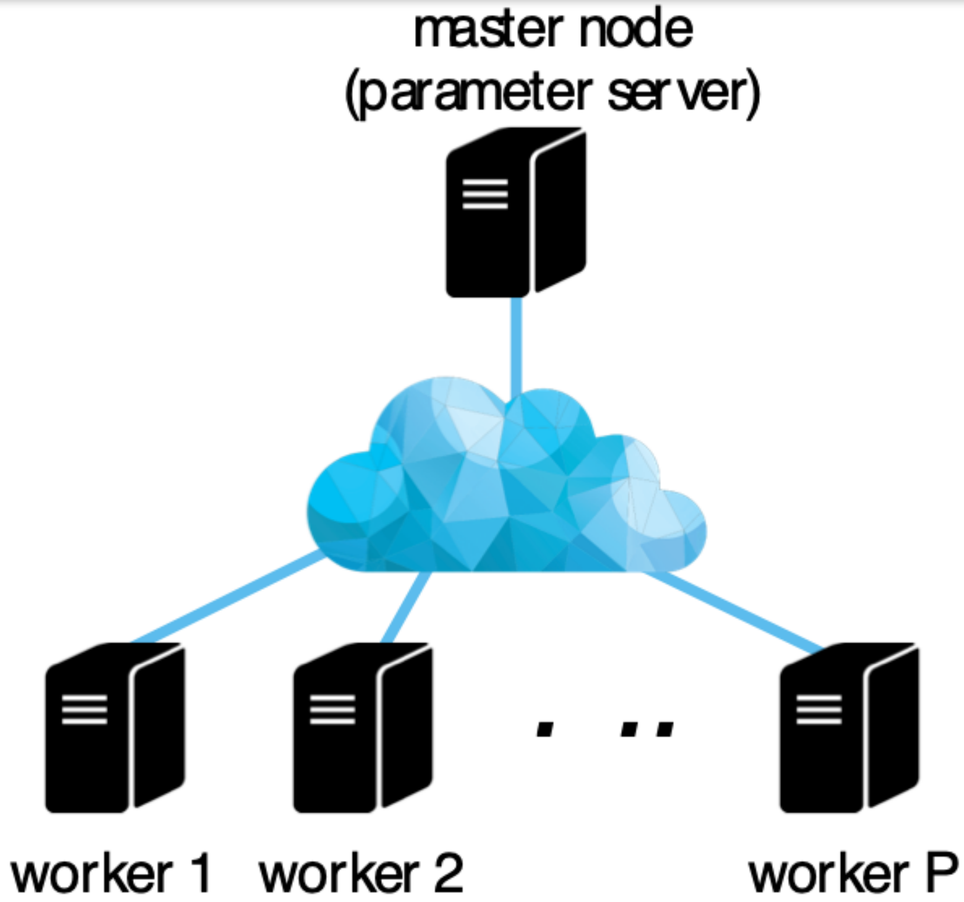
$$w_{k+1} = w_k - \gamma \nabla \ell_{s_k^4}(w_k; x_i)$$

Q: Does it perform the same as SGD?

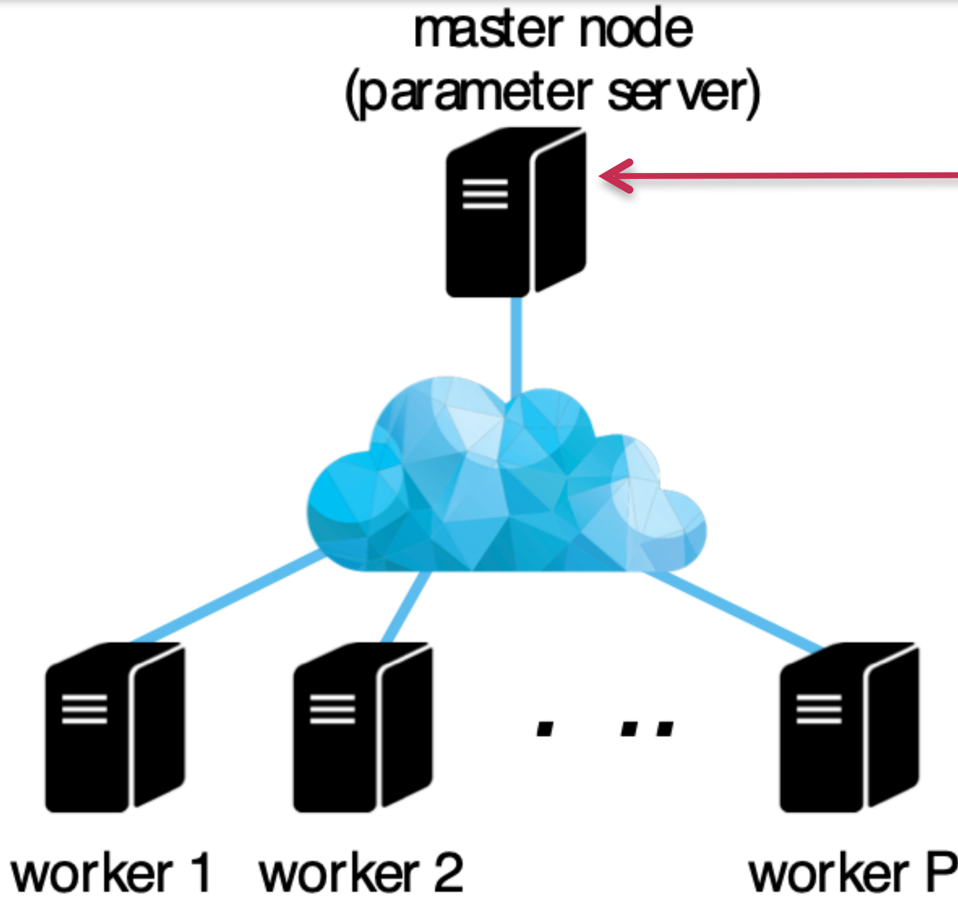
The Master-worker Setting



Algorithm of choice: minibatch SGD



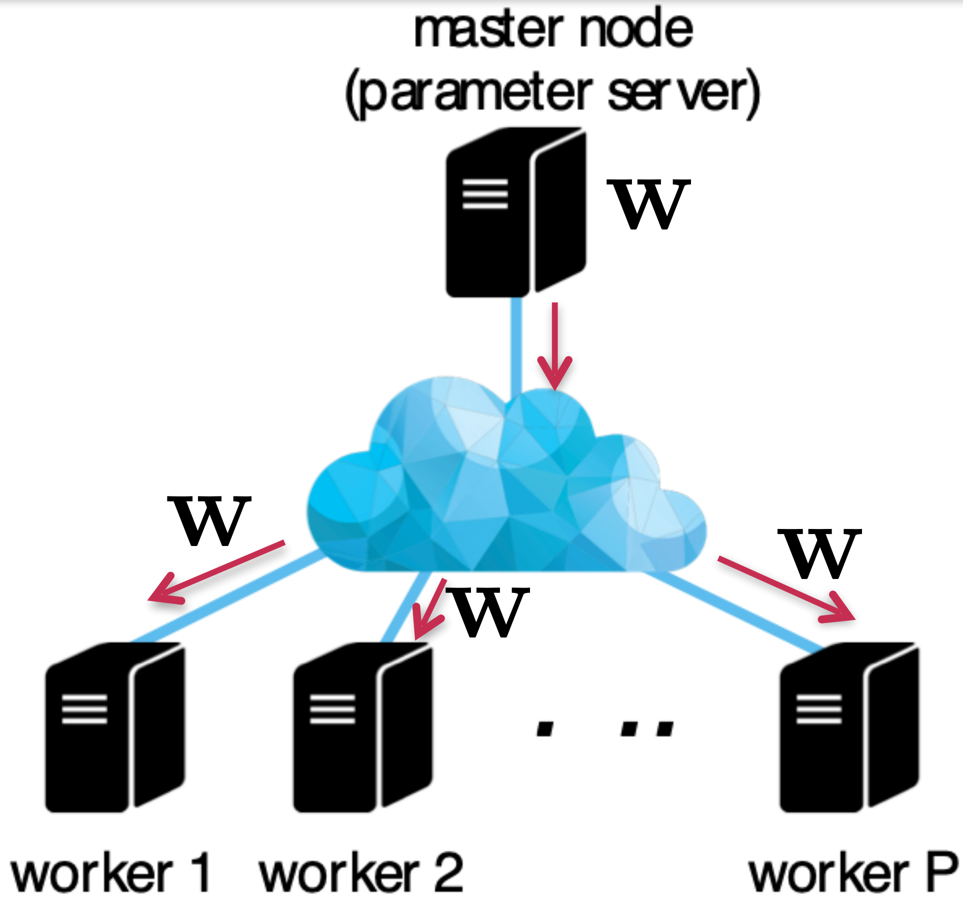
Algorithm of choice: minibatch SGD



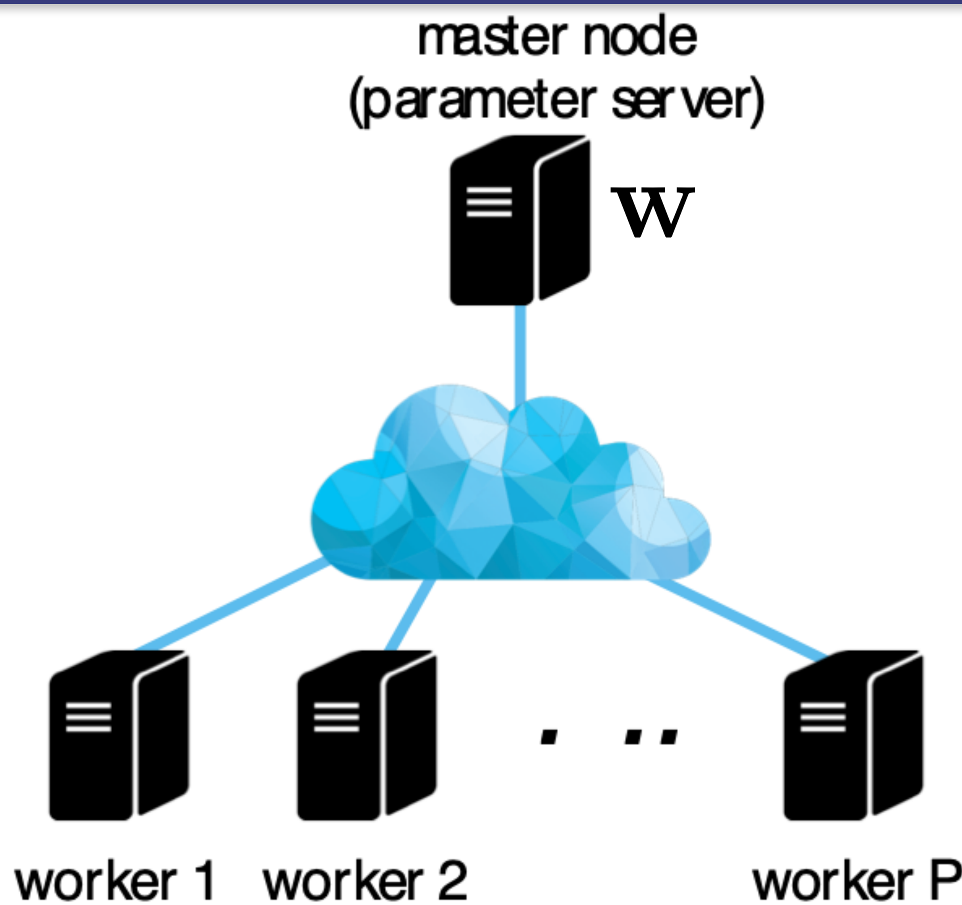
Stores the model

Gradient Computations

Algorithm of choice: minibatch SGD

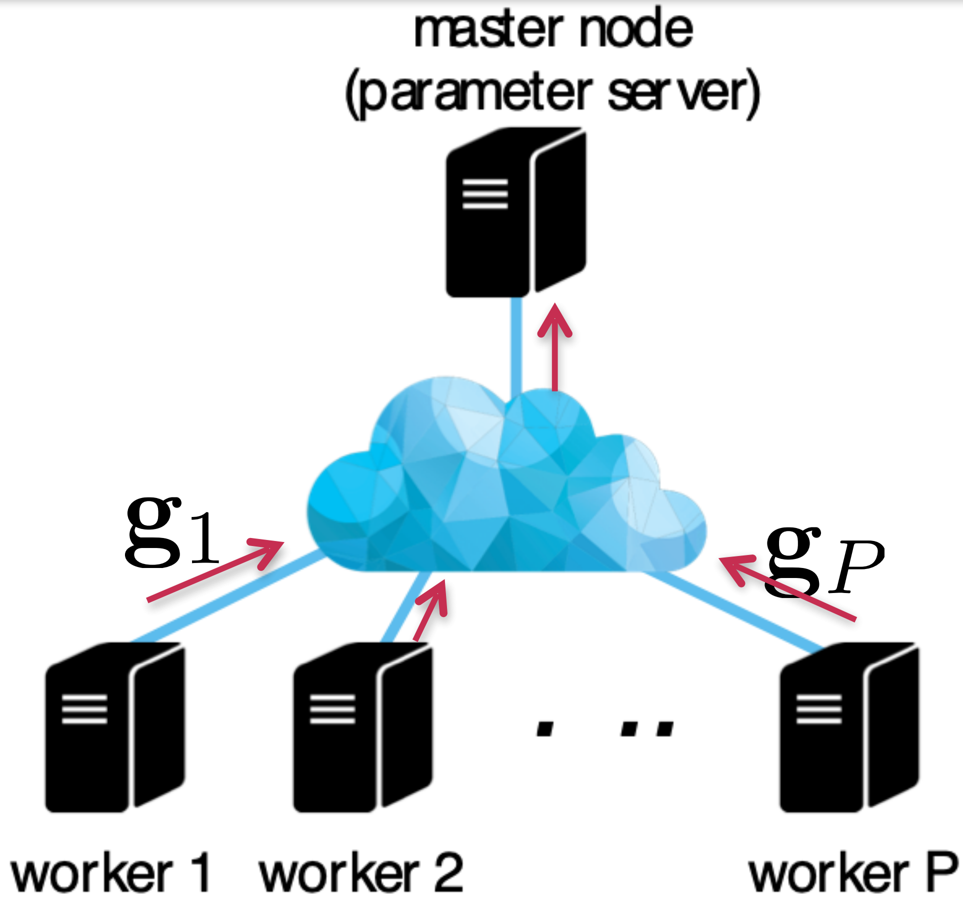


Algorithm of choice: minibatch SGD

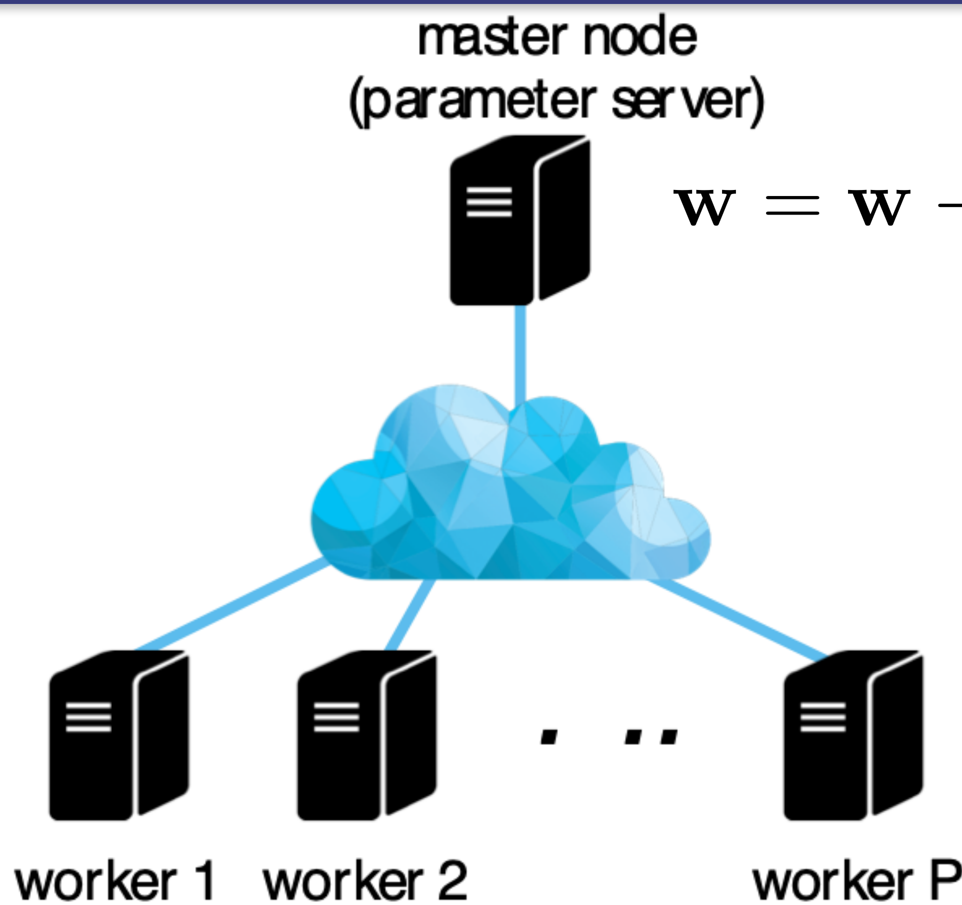


$$\mathbf{g}_1 = \sum_{i \in \mathcal{S}_1} \nabla \ell(\,(\mathbf{x}_i, y_i); \mathbf{w}\,) \qquad \mathbf{g}_P = \sum_{i \in \mathcal{S}_P} \nabla \ell(\,(\mathbf{x}_i, y_i); \mathbf{w}\,)$$

Algorithm of choice: minibatch SGD

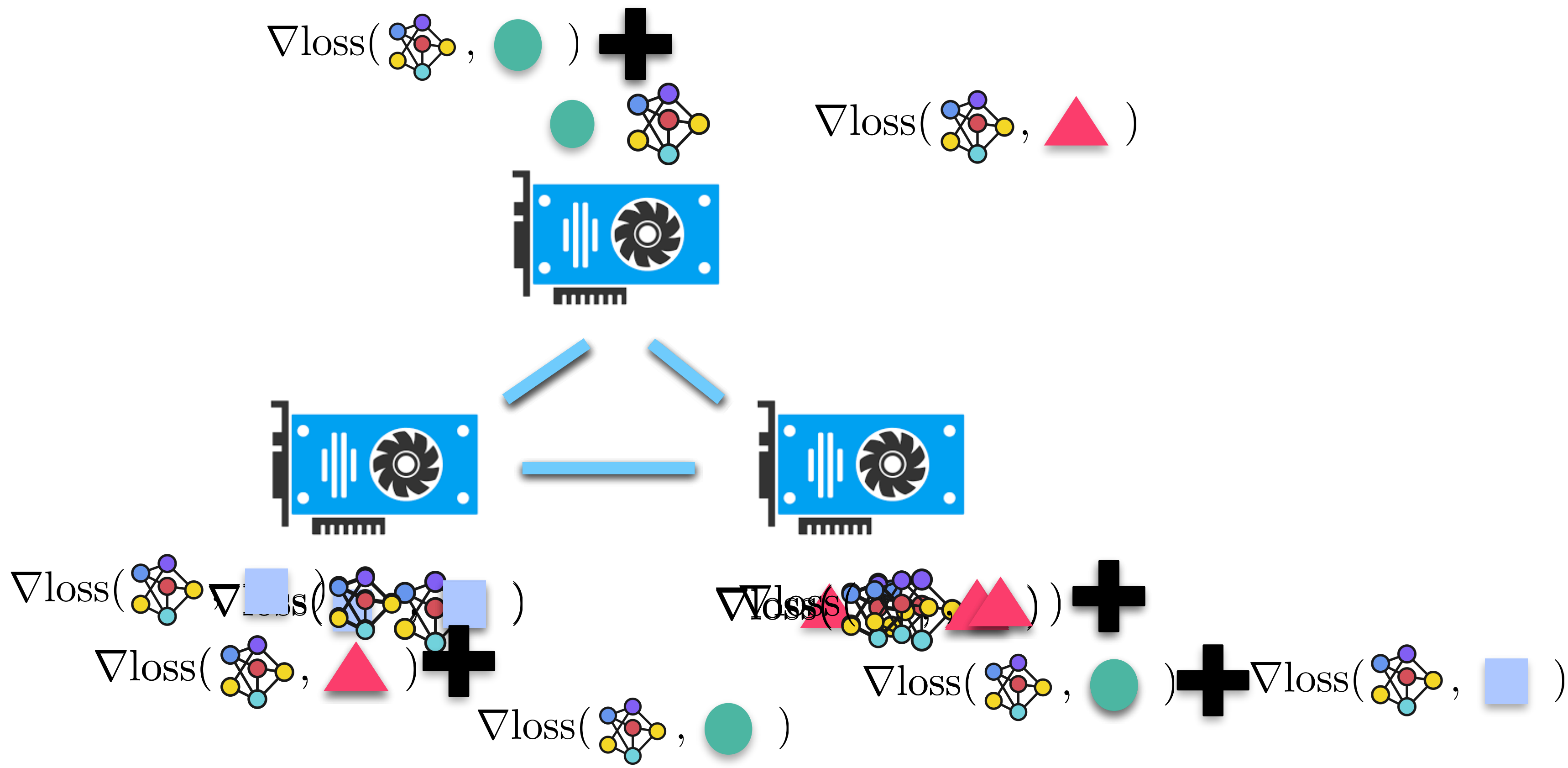


Algorithm of choice: minibatch SGD



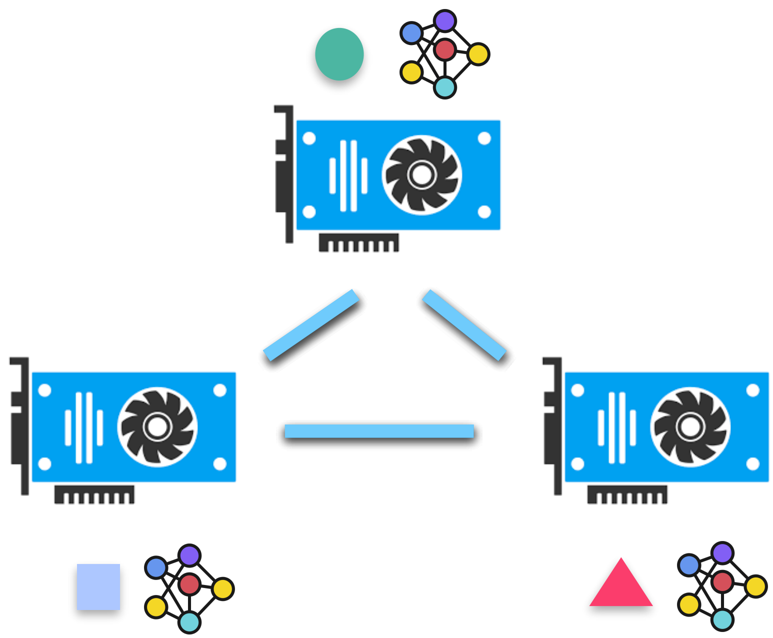
$$\mathbf{w} = \mathbf{w} - \gamma \sum_{i=1}^{P} \mathbf{g}_i$$

All-reduce



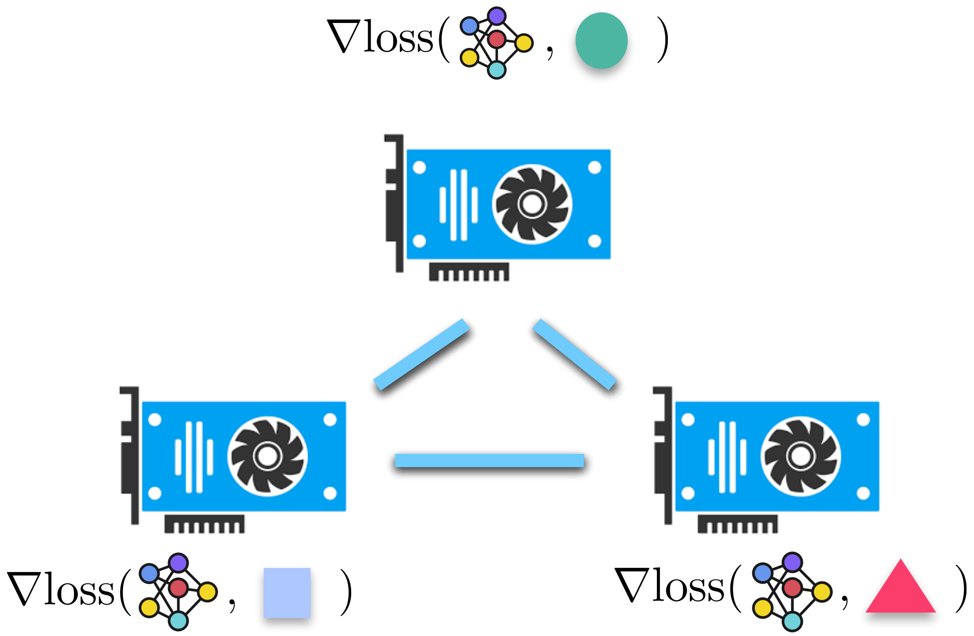
Sergeev, et al. Horovod: fast and easy distributed deep learning in TensorFlow

All-reduce



Sergeev, et al. Horovod: fast and easy distributed deep learning in TensorFlow

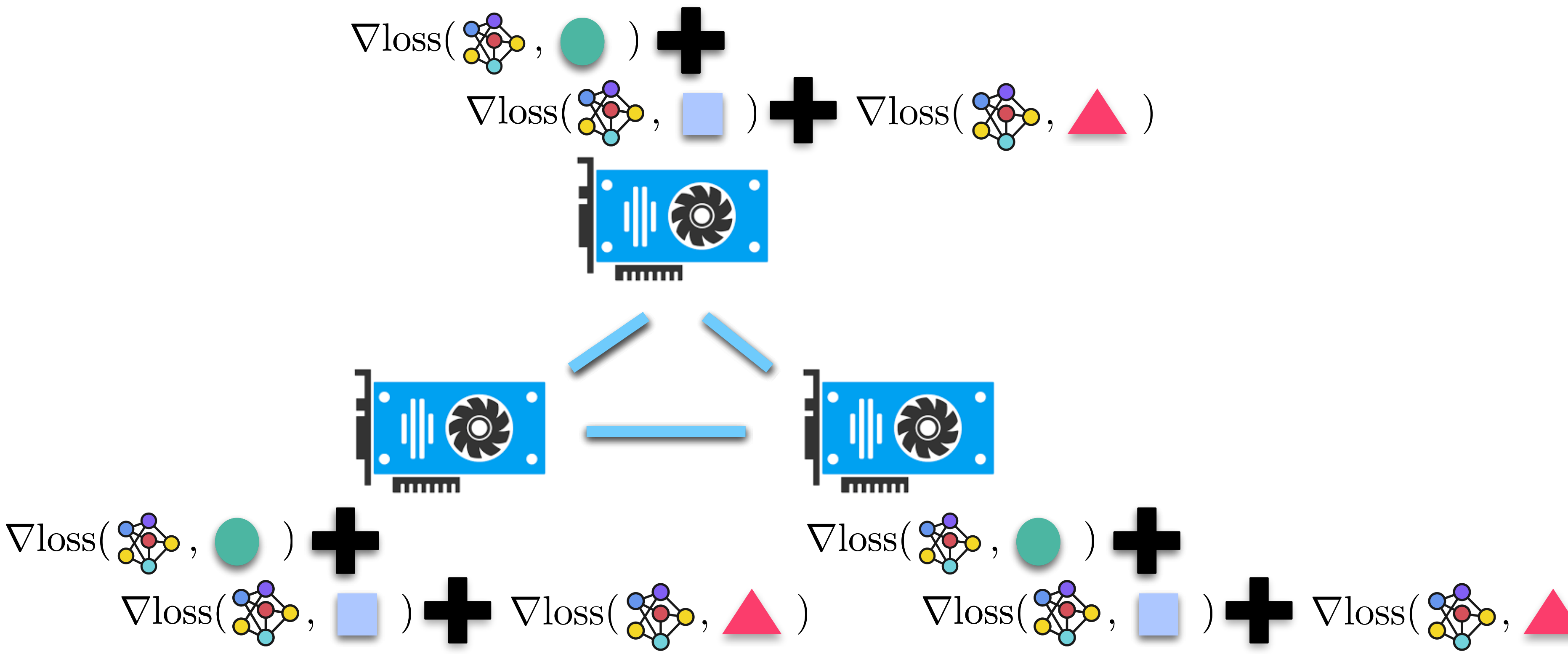
All-reduce



Sergeev, et al. Horovod: fast and easy distributed deep learning in TensorFlow

All-reduce

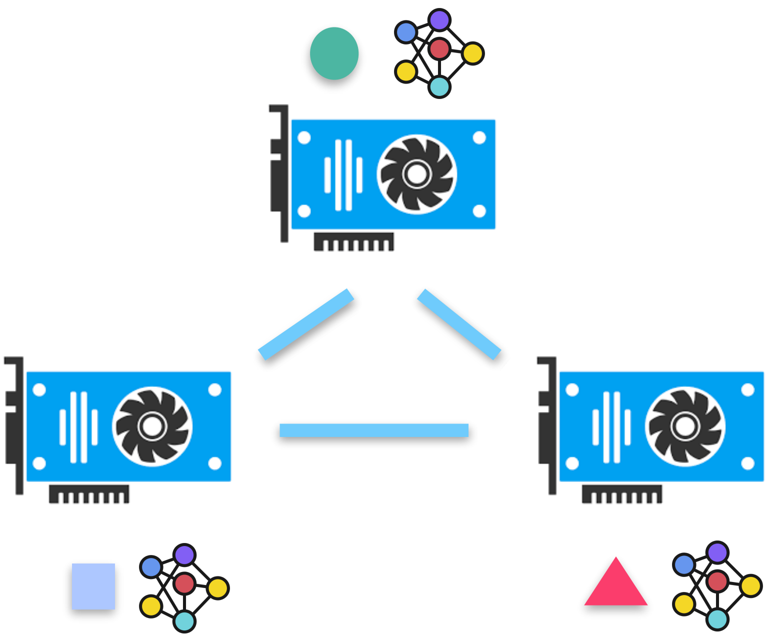
after K-1 push rounds



Sergeev, et al. Horovod: fast and easy distributed deep learning in TensorFlow

All-reduce, the server-free case

All-reduce SGD



Repeat until convergence

[Cotter, Shamir, Srebro, Sridharan, NIPS11]
[Dekel, Gilad-Bachrach, Shamir, Xiao, JMLR 2012]
[Friedlander and Schmidt, SIAM JSC 2012]

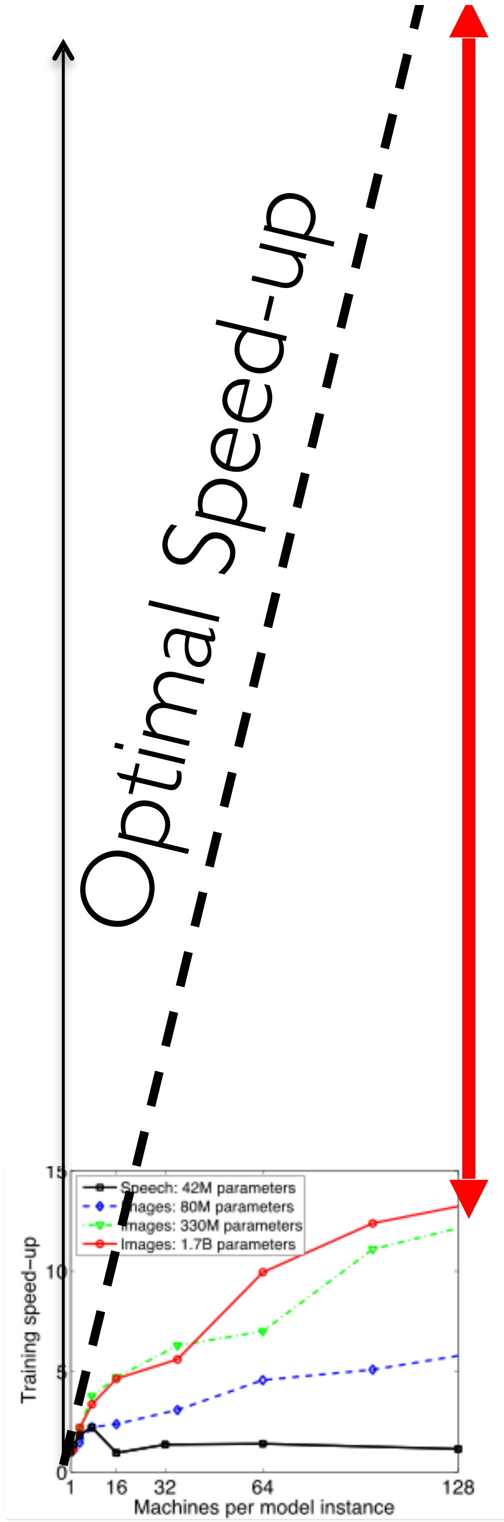
[Takác, Bijral, Richtárik, Srebro, ICML 2013]
[Li, Zhang, Chen, Smola, KDD 2014]
[Jain, Kakade, Kidambi, Netrapalli, Sidford, arxiv'16]
[De, Yadav, Jacobs, Goldstein, arxiv'16]

The Ideal Speedup Should be Proportional to #Compute Nodes

Many Questions….

- How fast does distributed-SGD converge?
- How can we measure speed?
- How can we update the model faster?
- How can we reduce communication?
- What happens with delayed nodes?
- Does fault tolerance matter?

Compute and Communication Bottlenecks



Reality ~100x worse than optimal

WHY?

“Large Scale Distributed Deep Networks” [Dean et al., NIPS 2012]

How to analyze parallel algorithms?

- Main measure of performance

$$\text{speedup} = \frac{\text{Time of serial } \mathcal{A} \text{ to accuraccy } \epsilon}{\text{Time of parallel } \mathcal{A} \text{ to accuraccy } \epsilon}$$

Example: Gradient Descent

Serial

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \gamma \cdot \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(\mathbf{x}_k)$$

Parallel

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \gamma \cdot \frac{1}{n} \left(\sum_{i_1=1}^{P} \nabla f_{i_1}(\mathbf{x}_k) + \ldots + \sum_{i_P=n-P+1}^{n} \nabla f_{i_P}(\mathbf{x}_k) \right)$$

- Convergence is invariant of allocation
- Both algorithms reach to same accuracy after T iterations
- Speedup is independent of covergence rate

How to analyze parallel algorithms?

- Main measure of performance

$$\text{speedup} = \frac{\text{Time of serial } \mathcal{A} \text{ to accuraccy } \epsilon}{\text{Time of parallel } \mathcal{A} \text{ to accuraccy } \epsilon}$$

Example: Gradient Descent

Serial $$\mathbf{x}_{k+1} = \mathbf{x}_k - \gamma \cdot \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(\mathbf{x}_k)$$

Embarrassingly Parallel O(#cores) speedup

Parallel $$\mathbf{x}_{k+1} = \mathbf{x}_k - \gamma \cdot \frac{1}{n}\left(\sum_{i_1=1}^{P} \nabla f_{i_1}(\mathbf{x}_k) + \ldots + \sum_{i_P=n-P+1}^{n} \nabla f_{i_P}(\mathbf{x}_k)\right)$$

- Convergence is invariant of allocation
- Both algorithms reach to same accuracy after T iterations
- Speedup is independent of covergence rate

Not true for mini-batch SGD

speedups for minibatch SGD

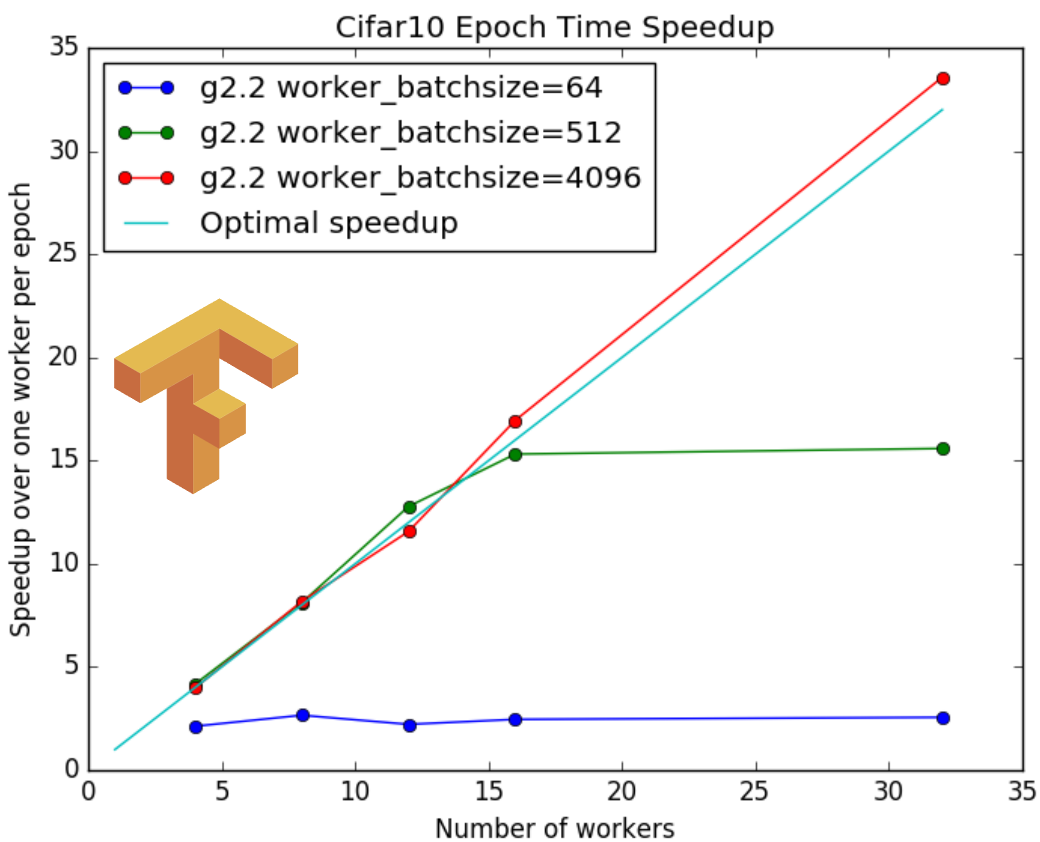
Two factors control run-time

Time to accuracy ε =
[time per data pass] X [#passes to accuracy ε]

speedups for minibatch SGD

"[…] on more than 8 machines […]
network overhead starts to dominate […]"

- TL;DR: Communication is the Bottleneck
- Why?

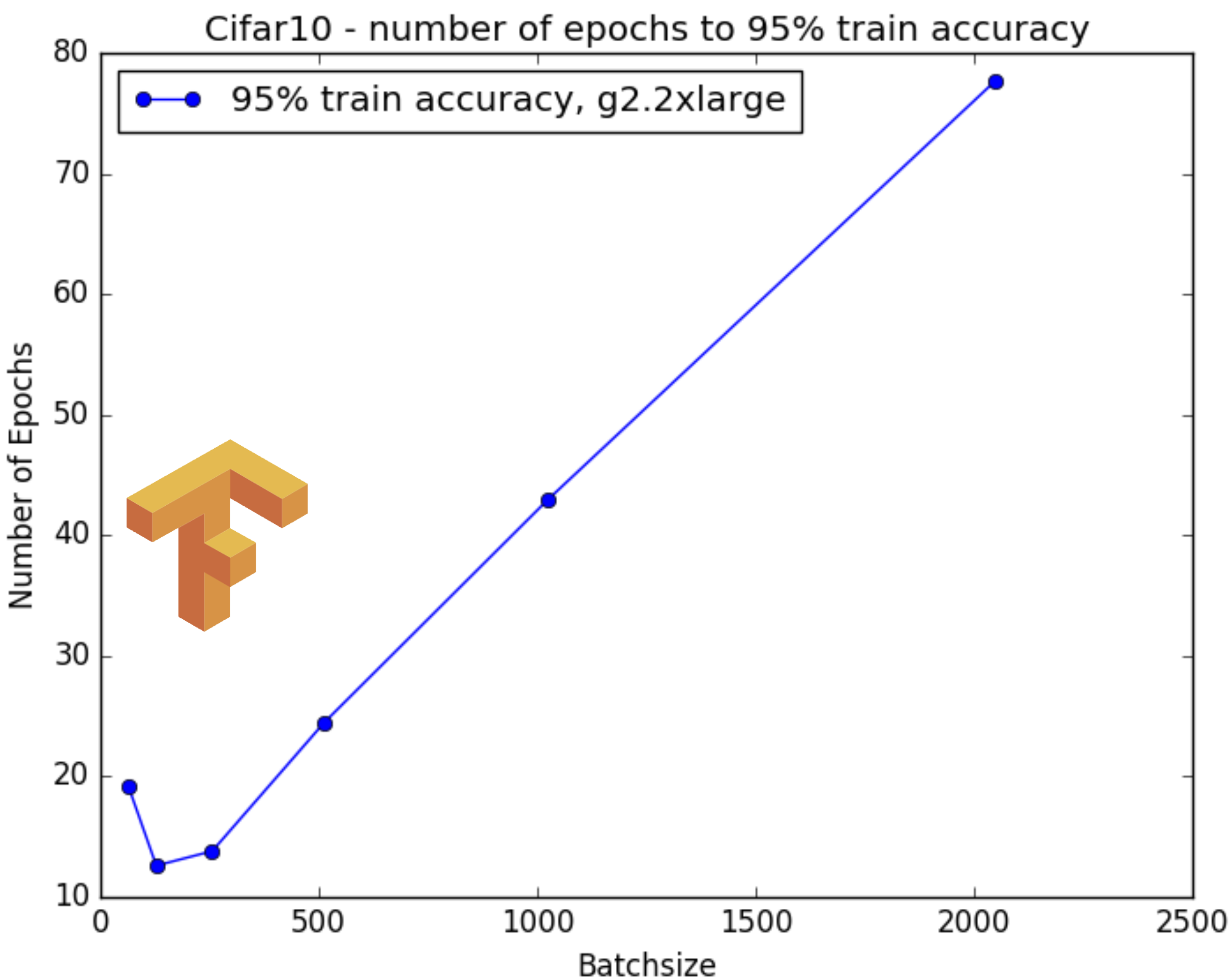


Time per pass:
time for dataset_size/batch_size
distributed iterations

Bigger Batch
$\Rightarrow$ Less Communication
(smaller time per epoch)

What's wrong with Large Batches?

- If small batch is bad, then maximize it



Large Batch
⇒ worse train error
(more #passes to accuracy ε)

Large Batch
=> some generalization issues

[Keskar, Mudigere, Nocedal, Smelyanskiy, Tang, ICLR 2017]

How to Analyze mini-batch?

- Measure of performance

$$\text{worst case speedup} = \frac{\text{bound on \#iter of SGD to } \epsilon}{\text{bound on \#iter of Parallel SGD to } \epsilon}$$

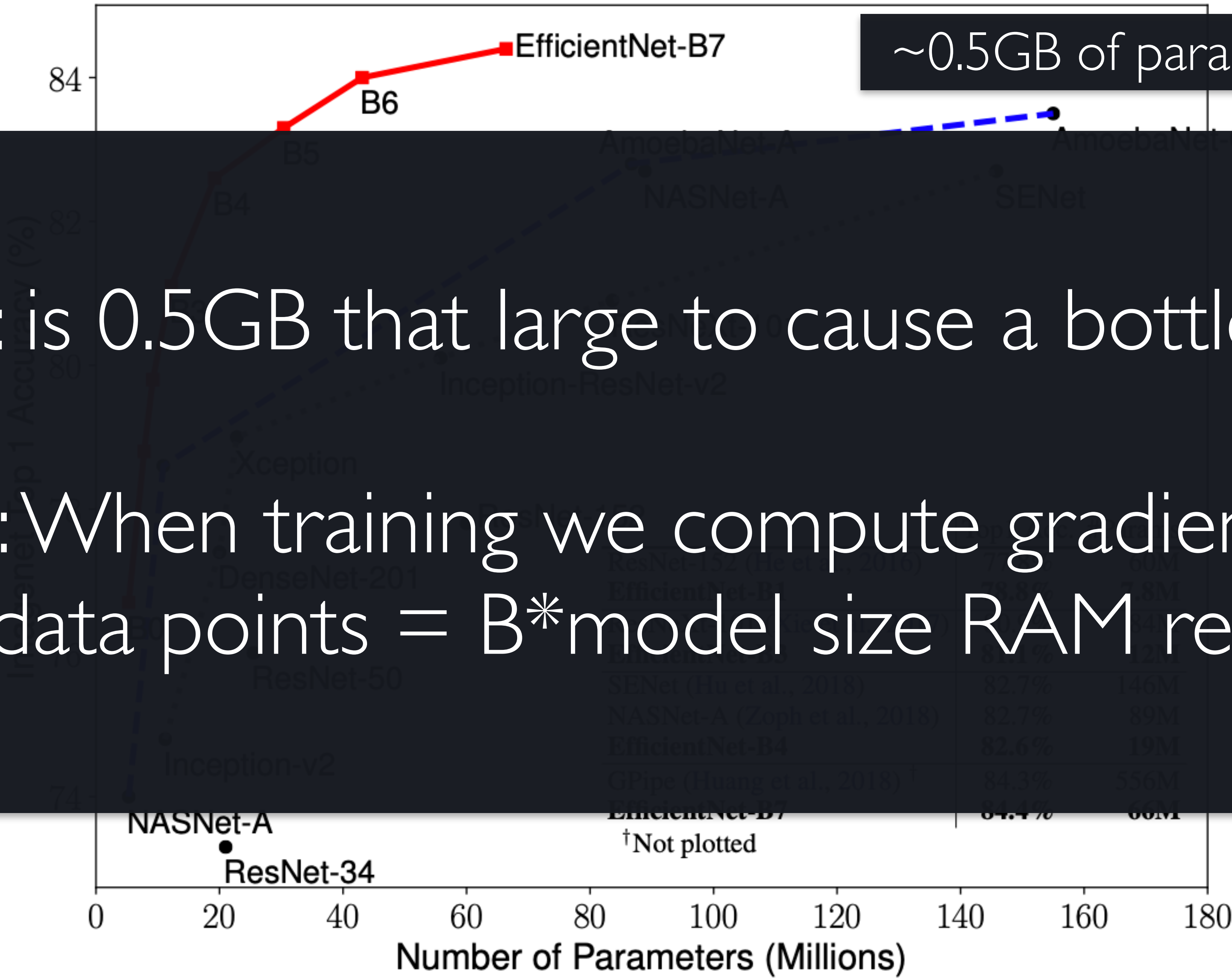
Main Question:
How does minibatch SGD compare against serial SGD?

Main questions:
Convergence after T gradient computations

Answer is Complicated: Depends on Problem
Generally if batch B_0 > B(data, loss)
Minibatch SGD offers no speedup.

Modern Architectures are huge
make everything slower

Model sizes (vision)



~0.5GB of parameters

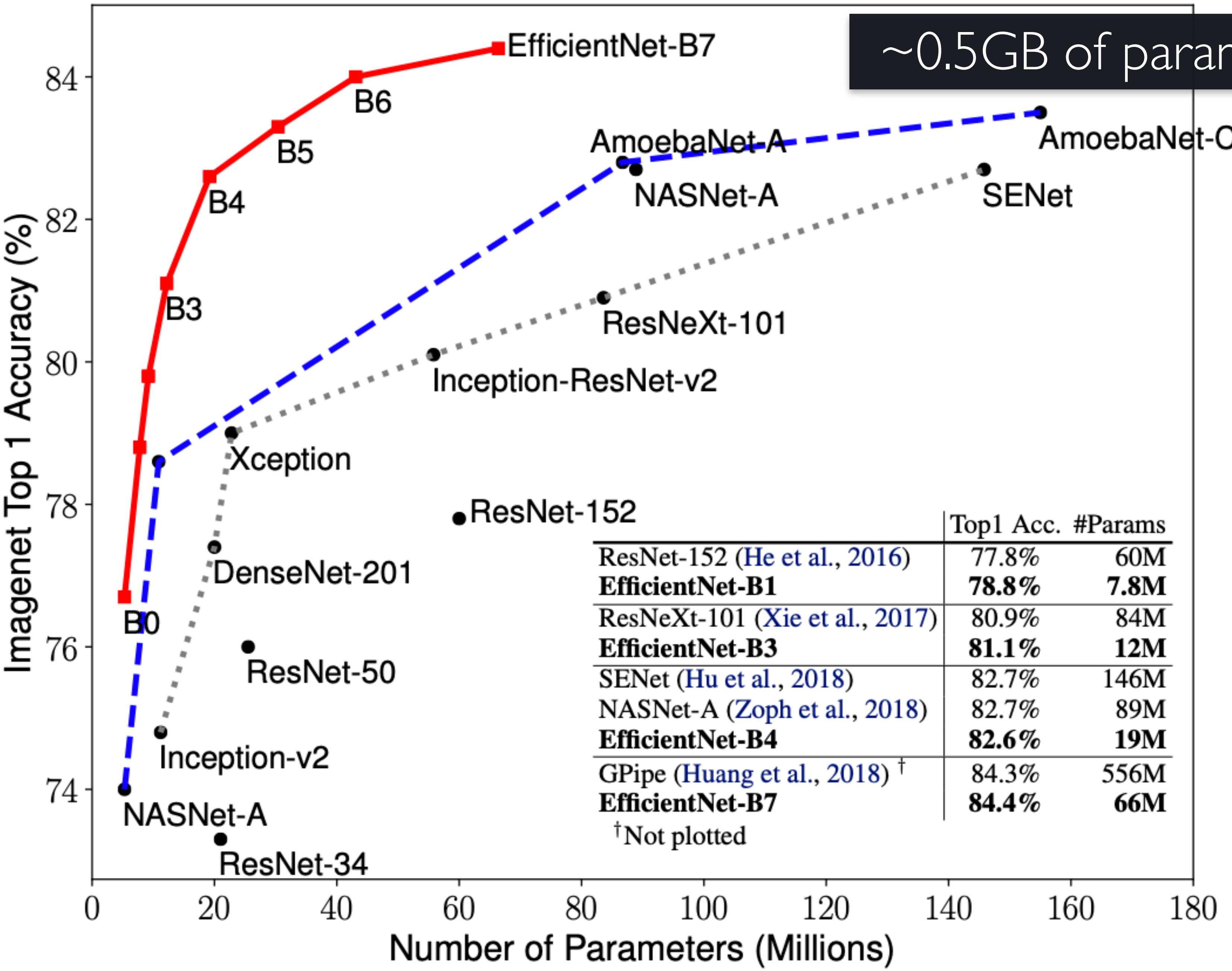
Q: is 0.5GB that large to cause a bottleneck?

A: When training we compute gradients on B data points = B*model size RAM required

[Source: http://proceedings.mlr.press/v97/tan19a/tan19a.pdf]

Model sizes (vision)

~0.5GB of parameters

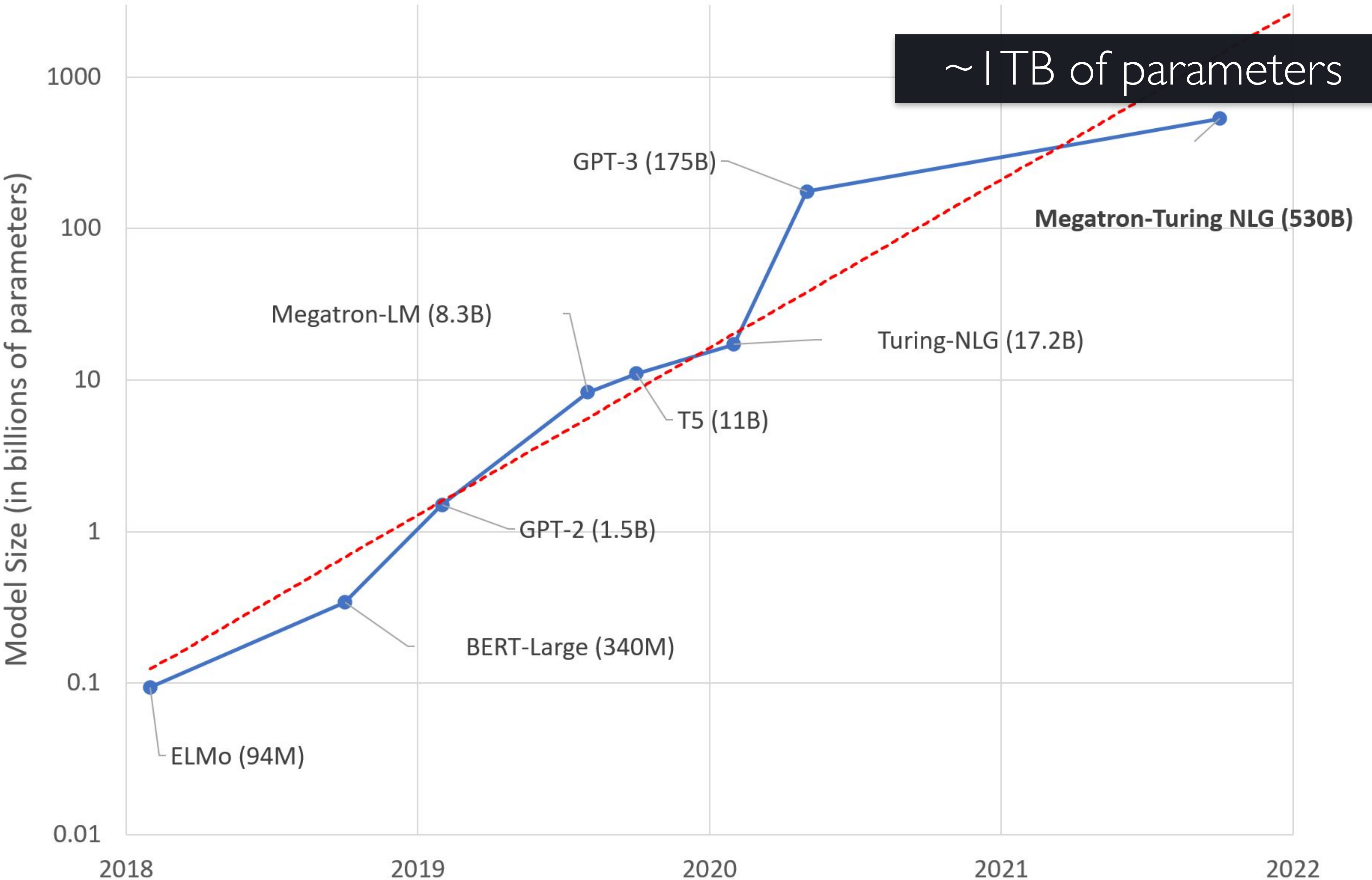


	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**78.8%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.1%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**82.6%**	**19M**
GPipe (Huang et al., 2018) †	84.3%	556M
EfficientNet-B7	**84.4%**	**66M**

†Not plotted

[Source: http://proceedings.mlr.press/v97/tan19a/tan19a.pdf]

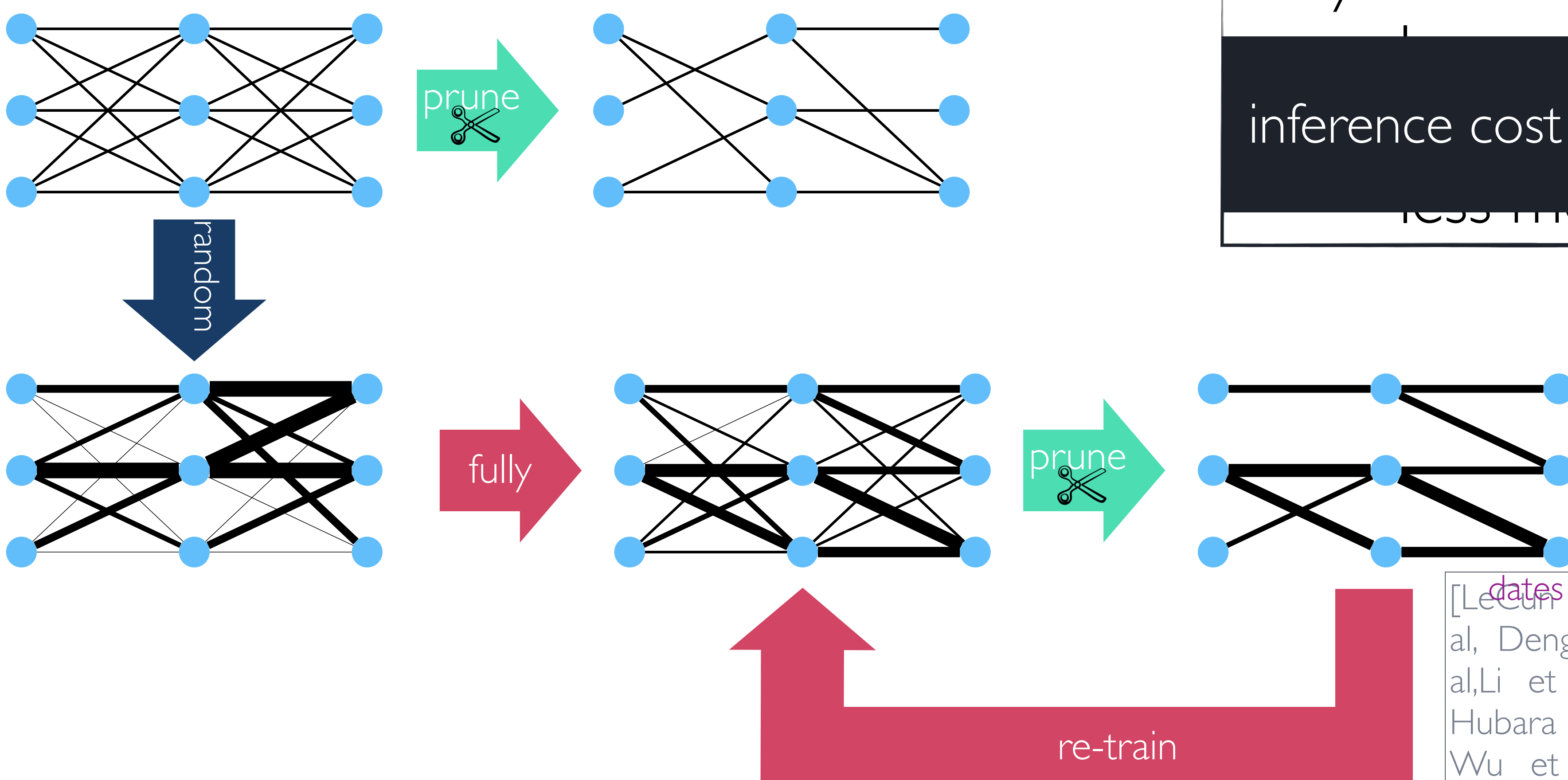
Model sizes (language)



[Source: https://tinyurl.com/Megatron-Turing-NLG]

compression by sparsity

Network Pruning, 1980-2018



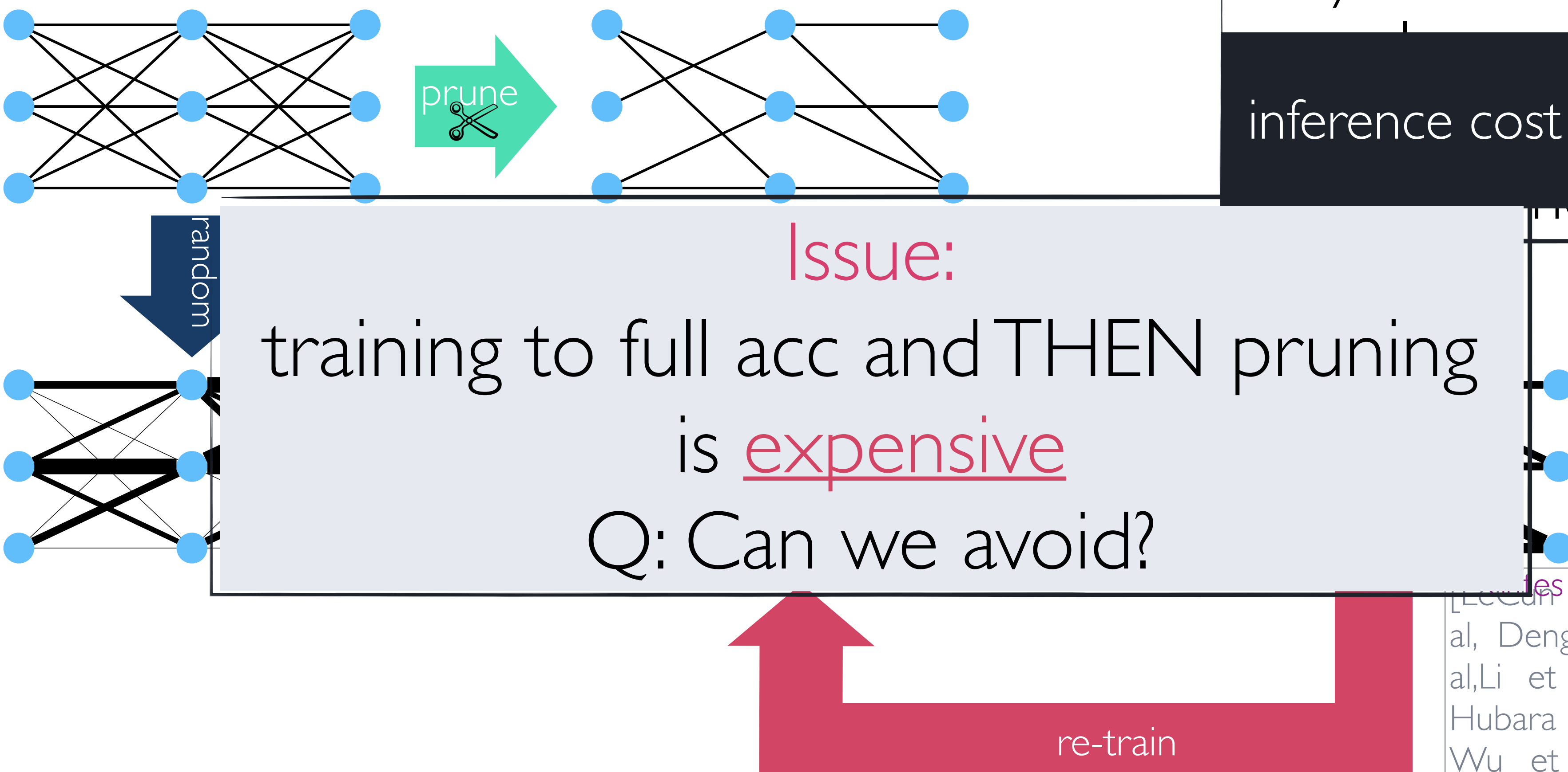
Why?

inference cost

dates back to 80s

[LeCun et al, Hassibi et al, Deng et al, Han et al, Li et al, Wen et al, Hubara et al, He et al, Wu et al, Zhu et al, Cheng et al, Blalock et al, Levin et al, Mozer et al]

Network Pruning, 1980-2018



Why?

inference cost

ergy
mpute
emory

Issue:

training to full acc and THEN pruning is expensive

Q: Can we avoid?

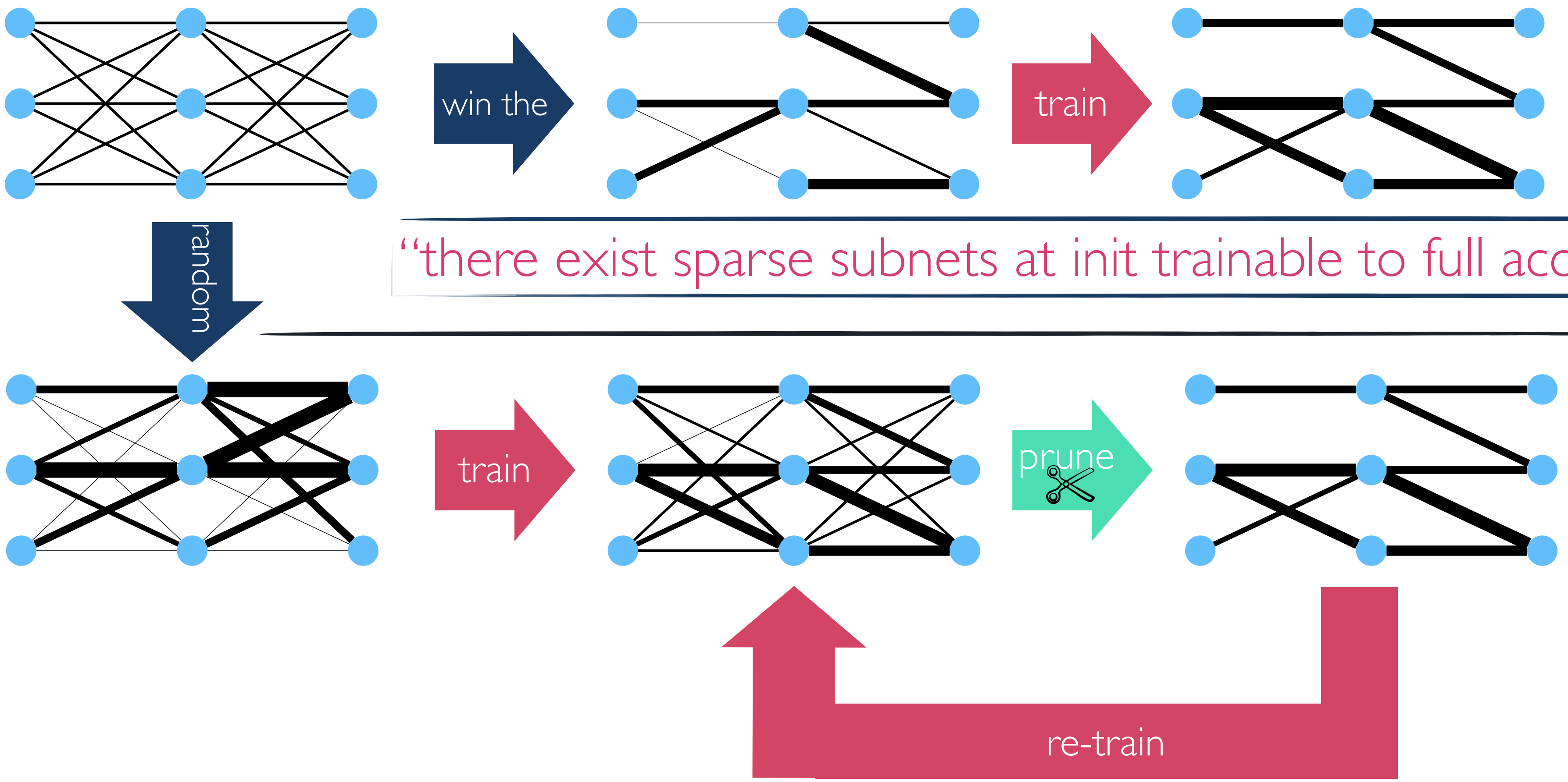
tes back to 80s

[LeCun et al, Hassibi et al, Deng et al, Han et al, Li et al, Wen et al, Hubara et al, He et al, Wu et al, Zhu et al, Cheng et al, Blalock et al, Levin et al, Mozer et al]

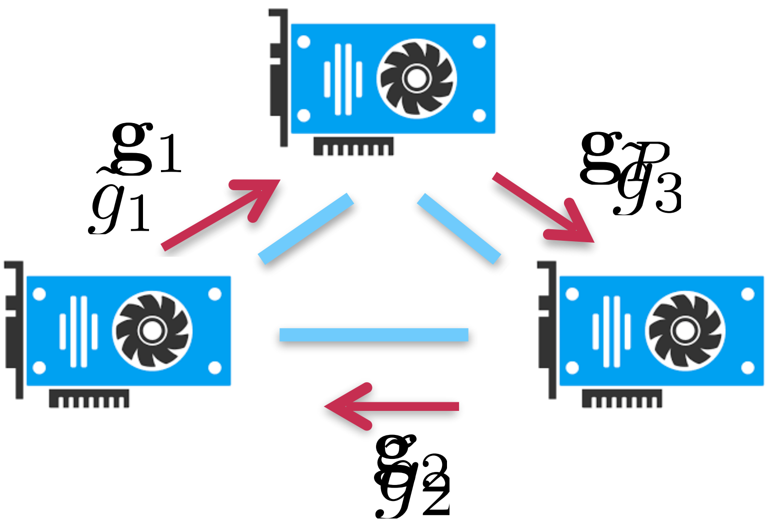
Lottery Ticket Hypothesis (LTH)

Frankle, Carbin, ICLR 2019

"there exist sparse subnets at init trainable to full accuracy"*



Gradient Compression



Communication / worker:

O(size of gradient) * 32 bits

Gradient Quantization

Quantize to precision:

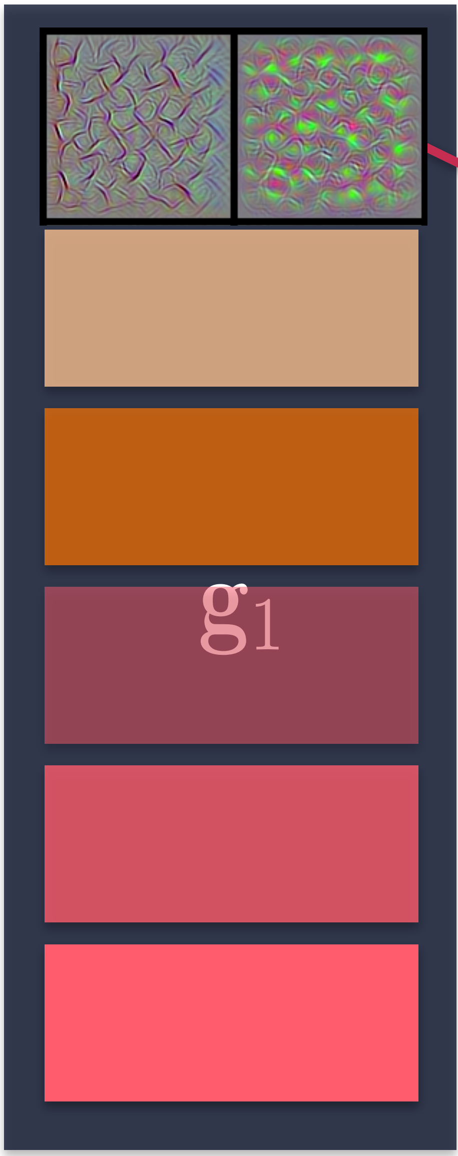
O(size of gradient) * 2/4/8 bits

Gradient Sparsification

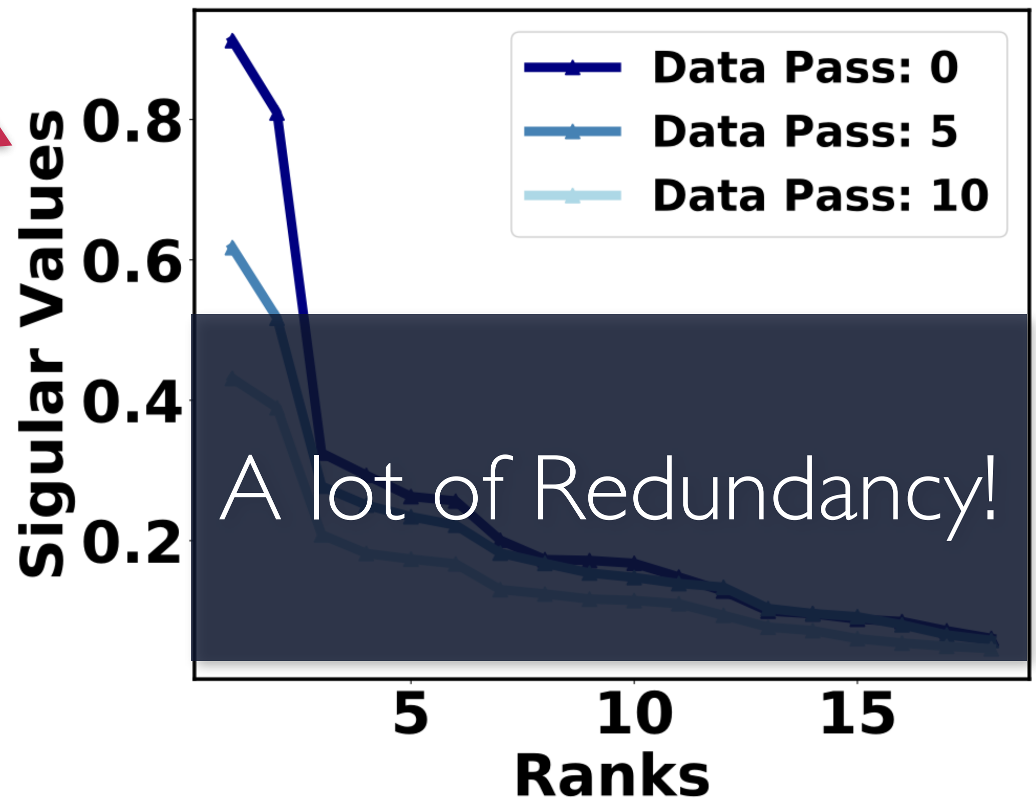
Sparsified Gradients (k-sparse):

O(k)

Gradients nearly-sparse in SVD



- Observation: Fast decaying spectral profile (SVD)



- *What if we compress in the spectral domain?*

Hongyi Wang et al. "ATOMO: Communication-efficient Learning via Atomic Sparsification", NeurIPS 2018.

LoRA: Low-Rank Adaptation of Large Language Models

Edward Hu* **Yelong Shen*** **Phillip Wallis**
Zeyuan Allen-Zhu **Yuanzhi Li** **Shean Wang** **Weizhu Chen**
Microsoft Corporation
{edwardhu, yeshe, phwallis, zeyuana, yuanzhil
swang, wzchen}@microsoft.com

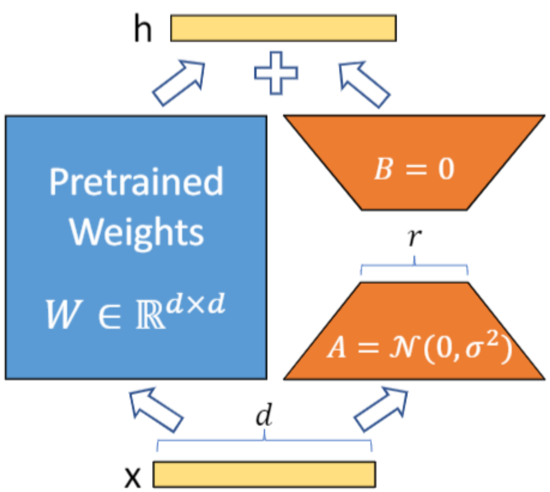
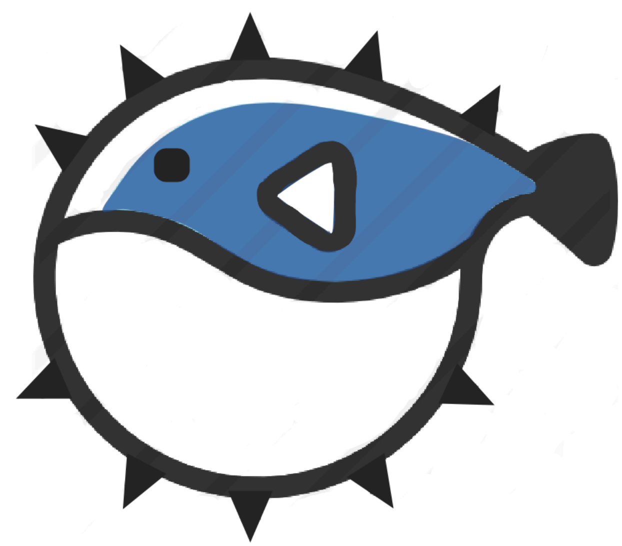


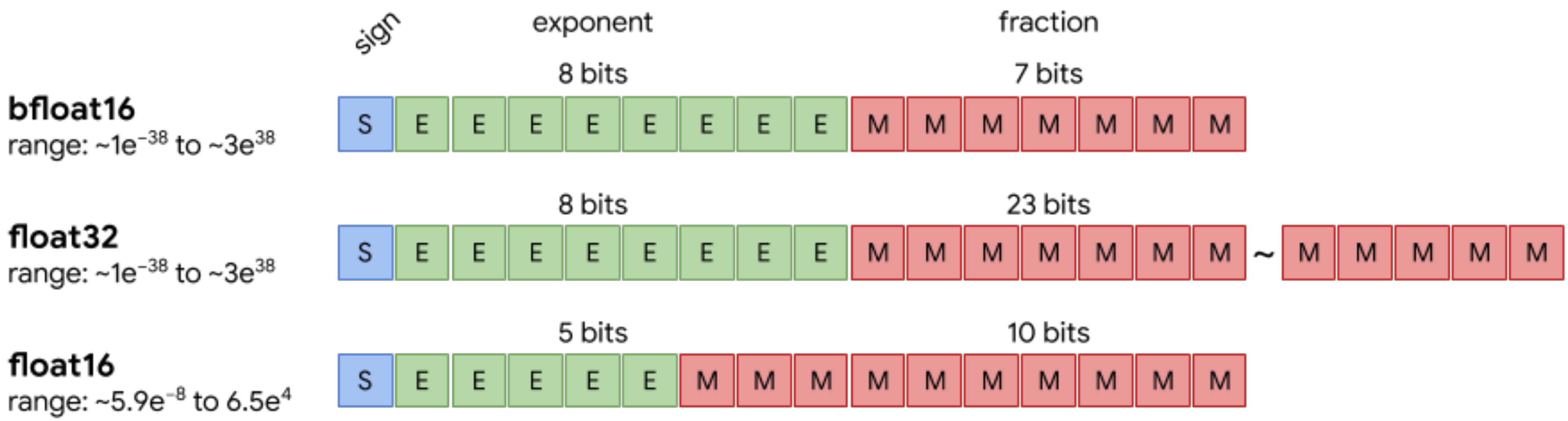
Figure 1: Our reparametrization. We only train A and B.

Pufferfish: Communication-efficient Models At No Extra Cost

Hongyi Wang, [1] **Saurabh Agarwal,** [1] **Dimitris Papailiopoulos** [2]



compression by quantization

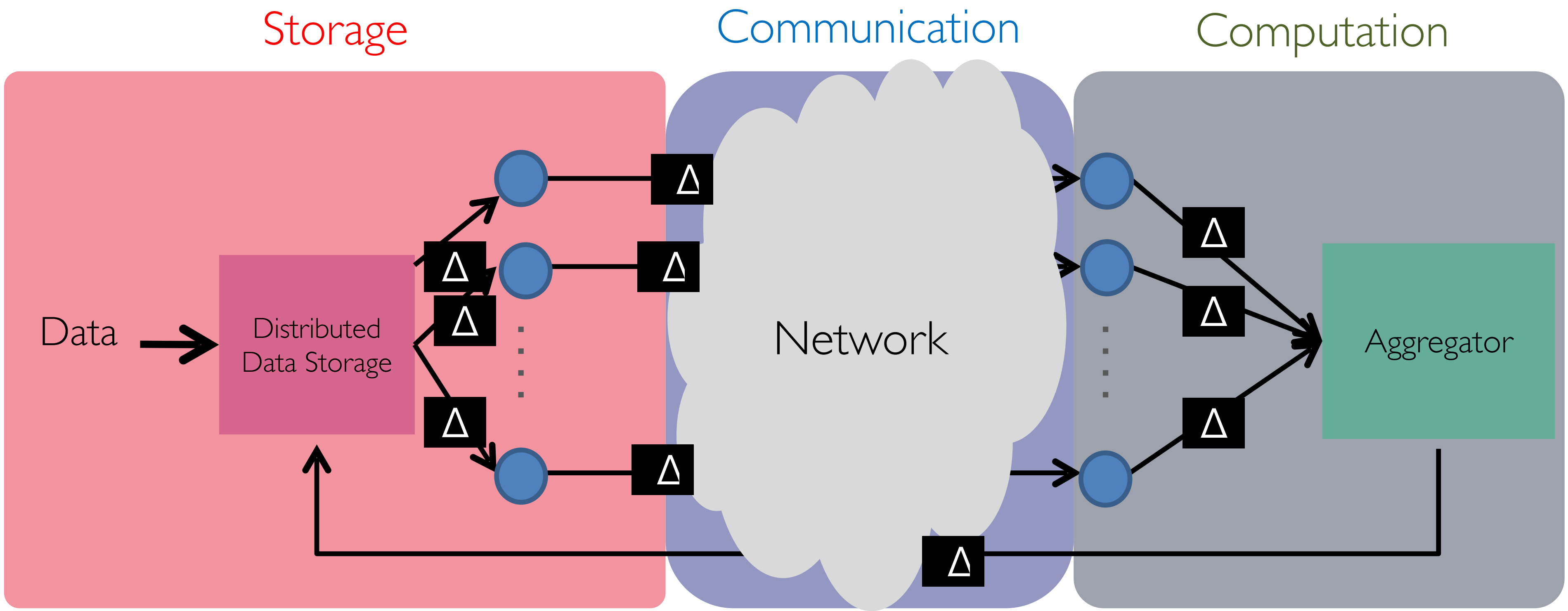


Makes model smaller!

Can affect the training dynamics

Straggler Nodes

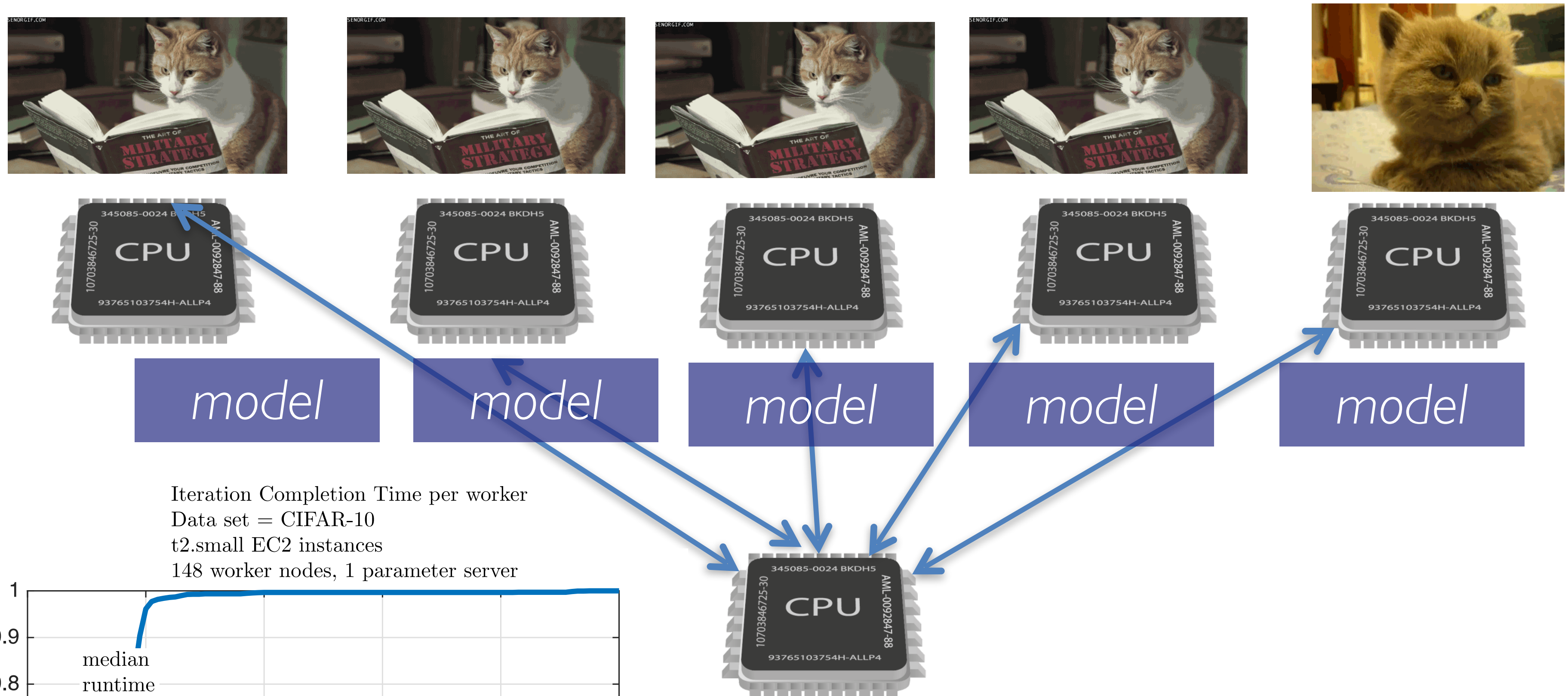
Large-scale Distributed Machine Learning Systems

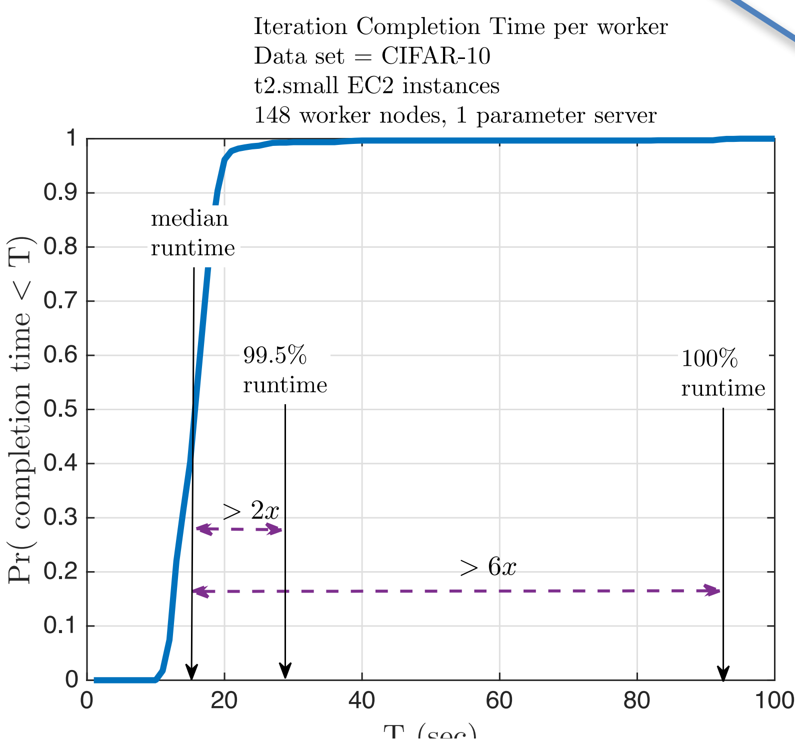


"The scale and complexity of modern Web services make it infeasible to eliminate all latency variability."

Jeff Dean, Google.

Bottleneck 2: Straggling Learners

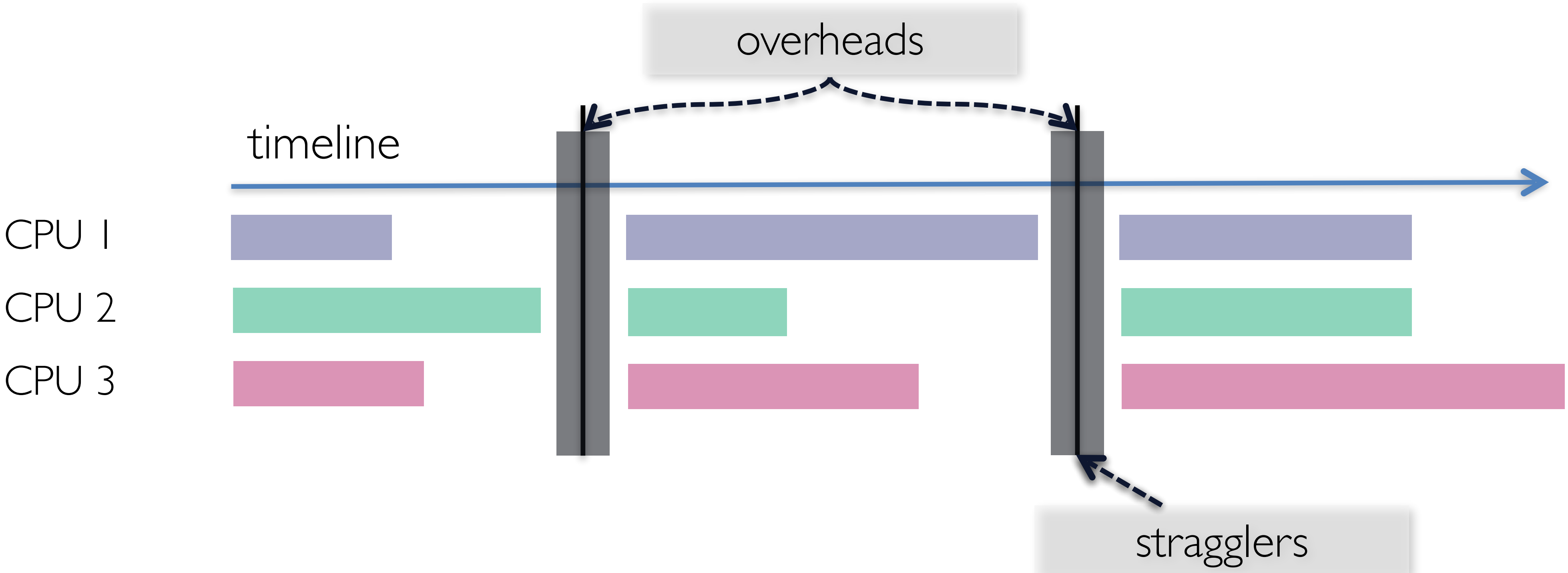




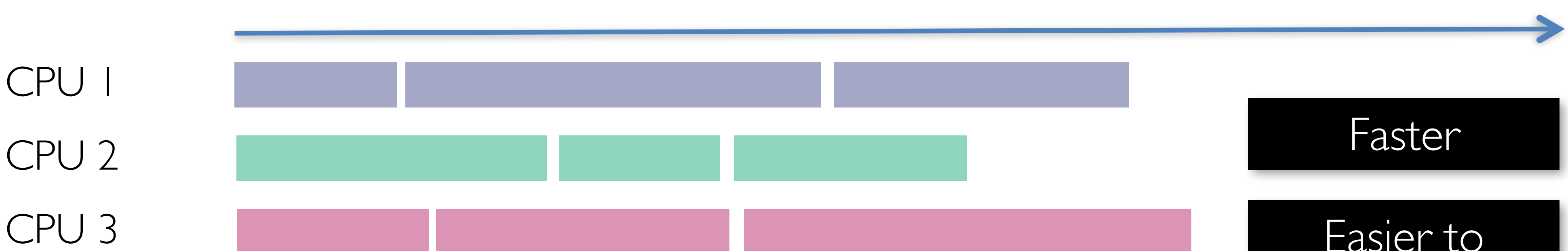
Can we "robustify" distributed ML against stragglers?

Measured on Amazon AWS

A case against Synchronization



Asynchronous World



Faster

Easier to Implement

HOGWILD! 2011

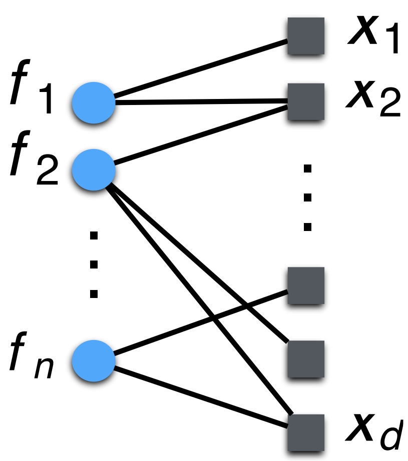
"Run parallel lock-free SGD without synchronization"


Niu

Recht

Ré

Wright



Each processor in parallel

sample function f_i

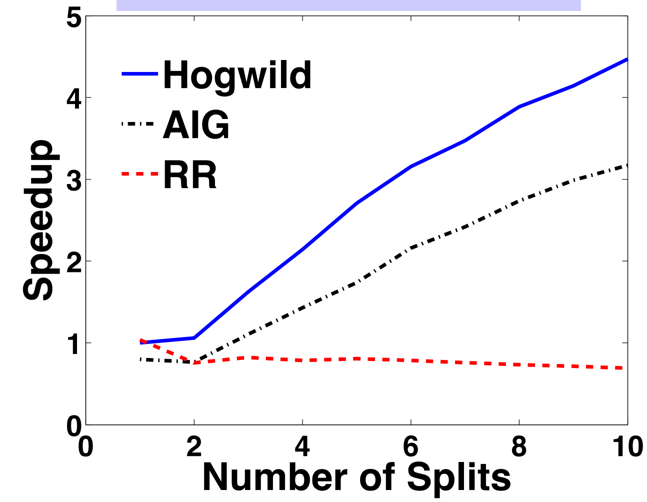
$x =$ read shared memory

$g = -\gamma \cdot \nabla f_i(x)$

for v in the support of f **do**

$\quad x_v \leftarrow x_v + g_v$

SVM RCV1 (10 cores)

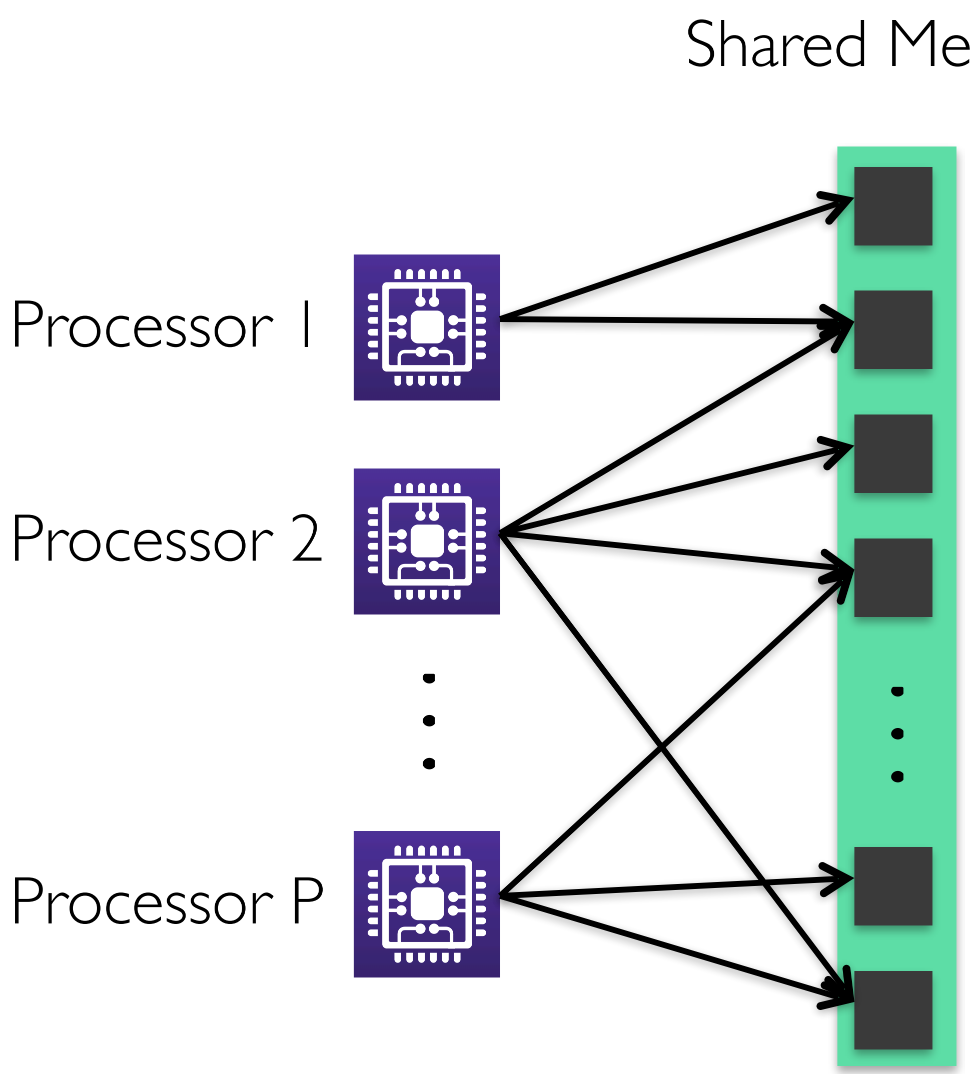


<u>Impact</u>

Google Downpour SGD, Microsoft Project Adam *use HOGWILD!*

Renewed interest on async. optimization

Challenges in Hogwild!



Issues:

- updates can be old
- results can overwritten
- Speedup saturation
- no "reproducibility"

Convergence analysis is usually a pain

A case against Asynchrony

Under review as a conference paper at ICLR 2017

REVISITING DISTRIBUTED SYNCHRONOUS SGD

Jianmin Chen[*], **Xinghao Pan**[*†], **Rajat Monga, Samy Bengio**
Google Brain
Mountain View, CA, USA
{jmchen,xinghao,rajatmonga,bengio}@google.com

Rafal Jozefowicz
OpenAI
San Francisco, CA, USA
rafal@openai.com

ABSTRACT

Distributed training of deep learning models on large-scale training data is typically conducted with *asynchronous* stochastic optimization to maximize the rate of updates, at the cost of additional noise introduced from asynchrony. In contrast, the *synchronous* approach is often thought to be impractical due to idle time wasted on waiting for straggling workers. We revisit these conventional beliefs in this paper, and examine the weaknesses of both approaches. We demonstrate that a third approach, synchronous optimization with backup workers, can avoid asynchronous noise while mitigating for the worst stragglers. Our approach is empirically validated and shown to converge *faster* and to *better* test accuracies.

A case against Asynchrony

Serializability:
Hogwild Model ≠ SGD Model

cally conducted with *asynchronous* stochastic optimization to maximize the rate of updates, at the cost of additional noise introduced from asynchrony. In contrast, the *synchronous* approach is often thought to be impractical due to idle time wasted on waiting for straggling workers. We revisit these conventional beliefs in this paper, and examine the weaknesses of both approaches. We demonstrate that a third approach, synchronous optimization with backup workers, can avoid asynchronous noise while mitigating for the worst stragglers. Our approach is empirically validated and shown to converge *faster* and to *better* test accuracies.

Robustness During Inference

The Failures of Deep Learning

WHY?

The Failures of Deep Learning

WHY?

Robustness During Training

Robustness: a key Challenge

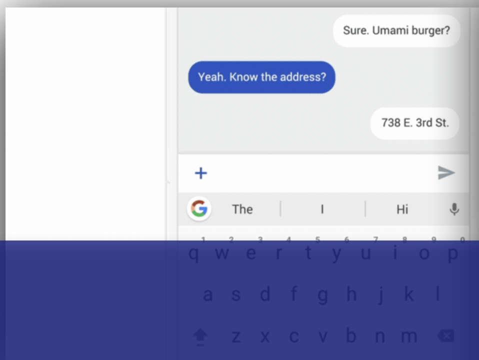
Fig. 5. Facebook global data center locations as of December 2017.

"For [...] training and inference [...] the importance of disaster-readiness cannot be underestimated."

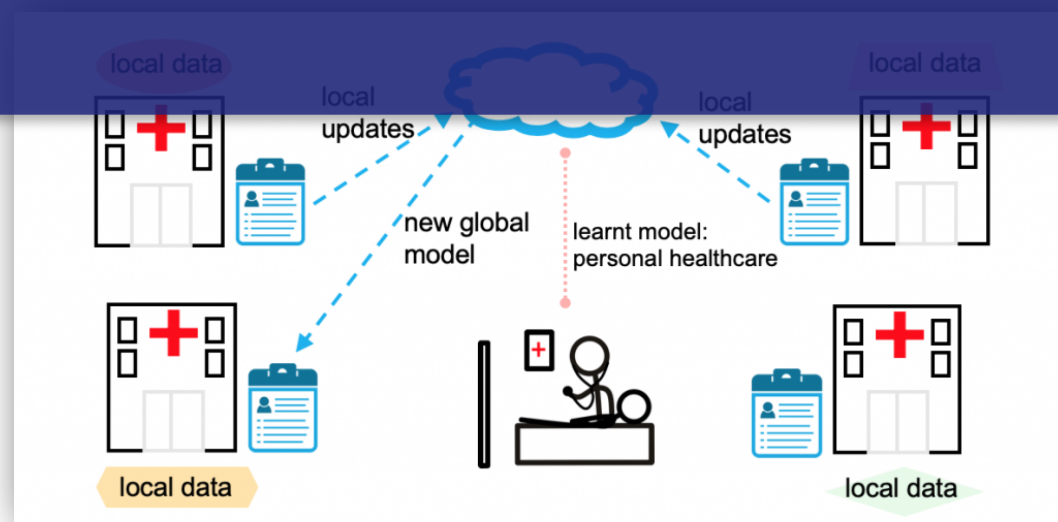
"Adversaries are constantly searching for new [...] ways to bypass our identifiers [...]"

Applied Machine Learning at Facebook: A Datacenter Infrastructure Perspective [fb, HPCA2018]

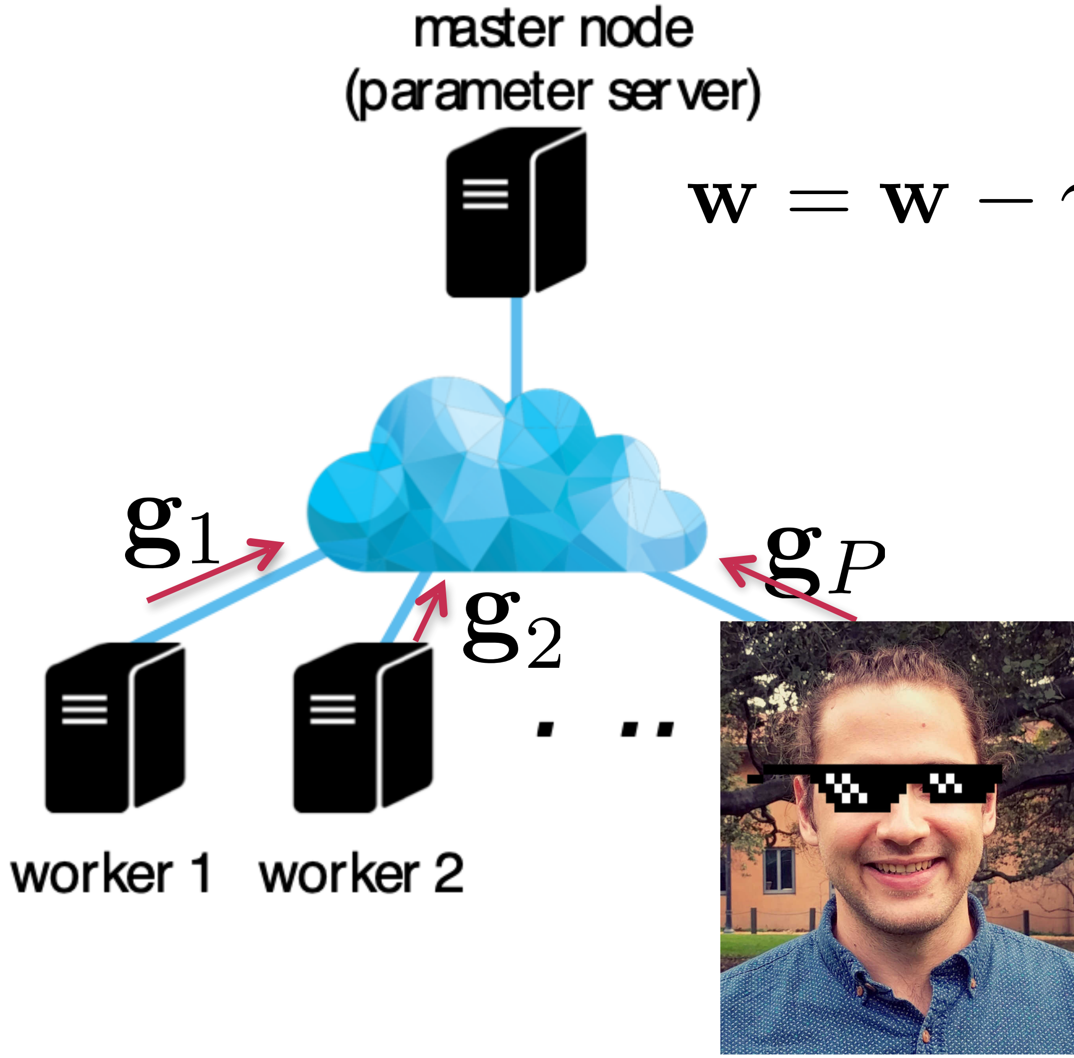
Federated Learning



Data & models not inspected by central authority



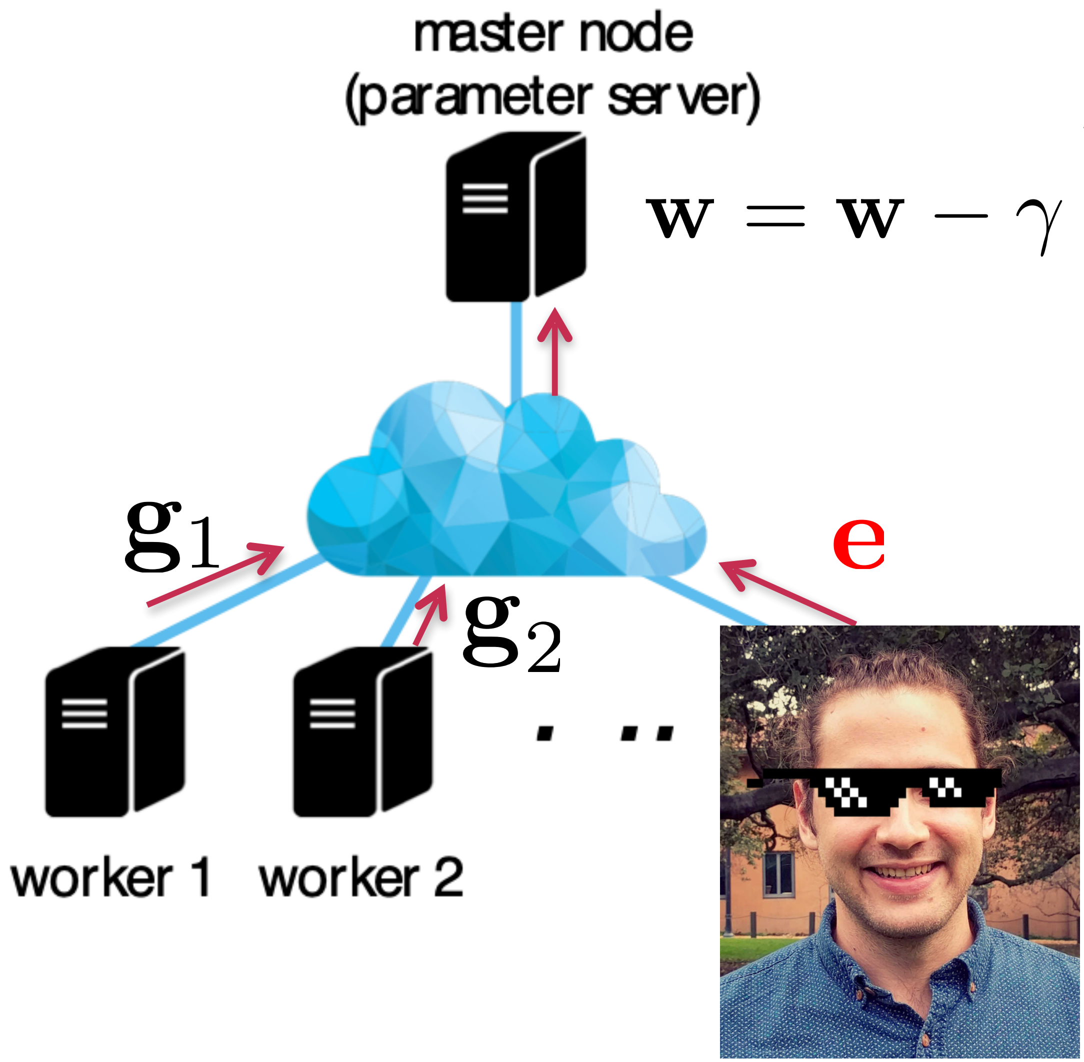
is SGD Robust?



$$\mathbf{w} = \mathbf{w} - \gamma \sum_{i=1}^{P} \mathbf{g}_i$$

is SGD Robust?

$$\mathbf{w} = \mathbf{w} - \gamma \sum_{i=1}^{P-1} \mathbf{g}_i + \mathbf{e}$$



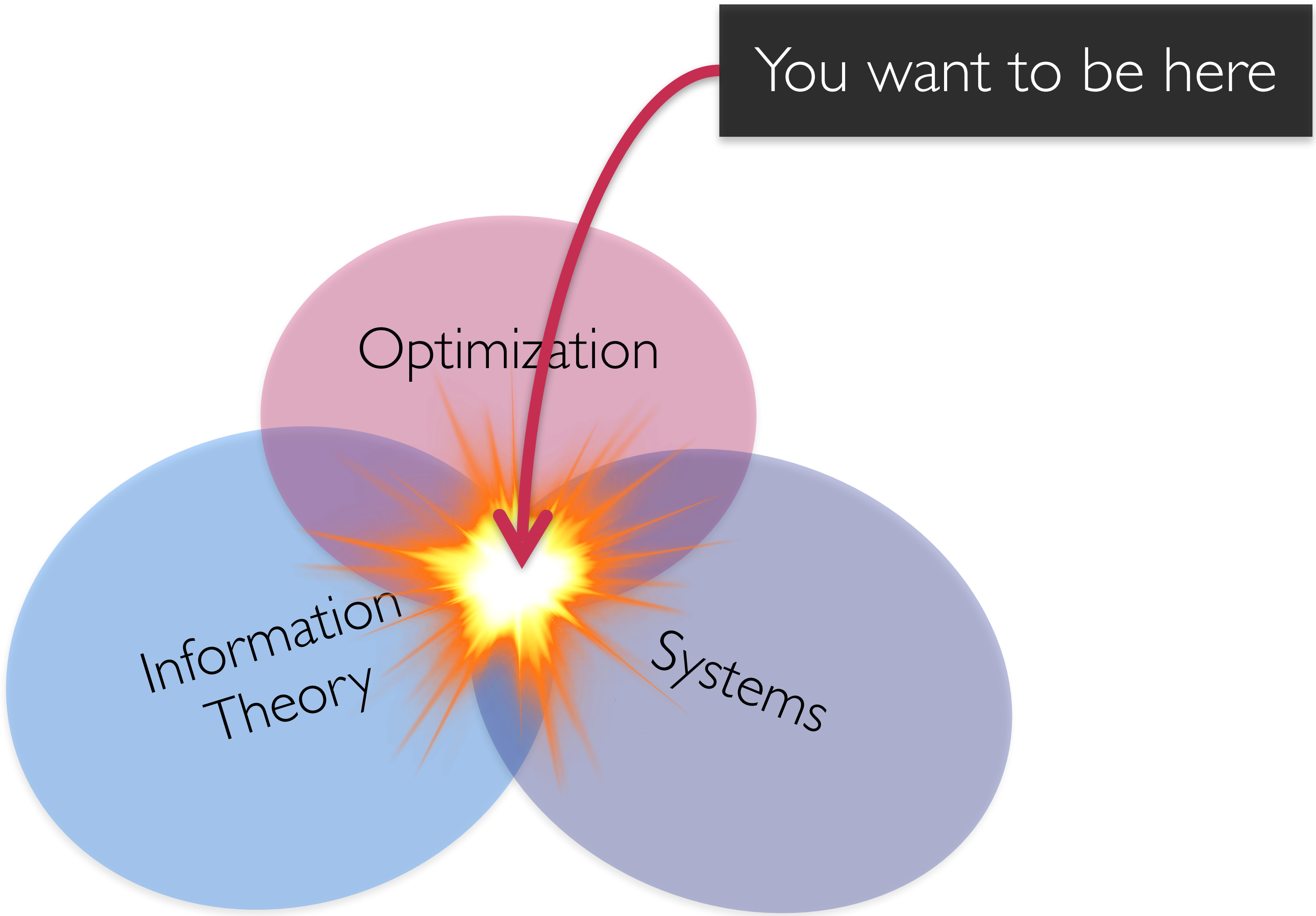
minibatch SGD is not Robust

Can we build a robust version of SGD that is:

+ cheap
+ easy convergence
+ vanilla SGD model = robust SGD mod

What’s coming next

- Understanding mini-batch SGD performance
- Hogwild and theoretical challenges
- Model/Gradient Compression
- Low communication schemes
- Straggler Nodes and ways to overcome them
- Adversarial attacks, why they happen, how to defend


You want to be here
Optimization
Information Theory
Systems