Distributed Optimization for Machine Learning

[The Good, the Bad, and the Hyperparameters]

Dan Alistarh

IST Austria & NeuralMagic, Inc.

European Research Council

Established by the European Commission

Hydra 2020

The Machine Learning “Cambrian Explosion”

CAT, DOG, DUCK

Image Classification
& Segmentation

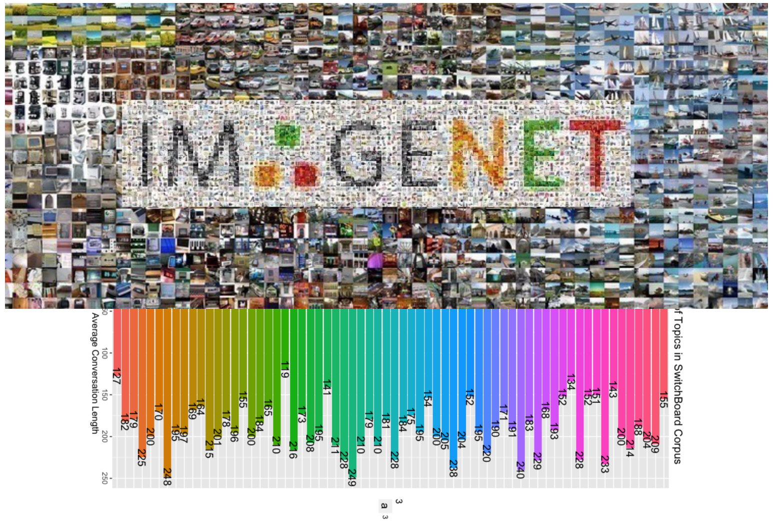
Speech Recognition
& Translation

Strategic Games
(Reinforcement Learning)

Even if progress stalls, **ML is here to stay**:
existing technologies already have significant industry adoption.

Three Factors

Great Ideas



High Quality Data

NVIDIA
aws
Google Cloud Platform

Efficient Computation

Distributed/parallel computing is the key enabler of computational speedups.

IMAGENET

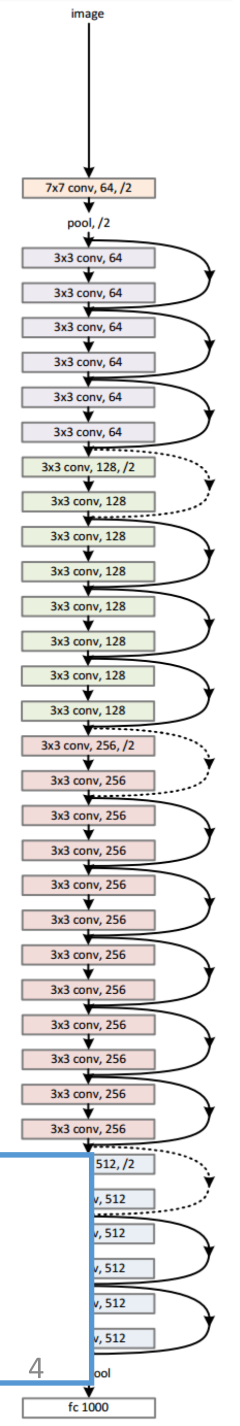
Distribution is Key

Training Deep Neural Networks Efficiently

- **Large Datasets:**
  - ImageNet: **1.3 Million images**
    Google OpenImages: **9 Million images**
  - NIST2000 Switchboard dataset: **2000 hours**
    Proprietary speech datasets: **> 30.000 hours (3.5 years)**

  ➢ **Distributed training is necessary**

- **Large Models**:
  - **ResNet-152** [He et al. 2015]: 152 layers**, 60 million parameters**
  - **LACEA** [Yu et al. 2016]: 22 layers, **65 million parameters**



Is efficient distributed machine learning a **solved problem**?

The Scalability Problem

CSCS: Europe's Top Supercomputer (World 4^{th})

- 4500+ GPU Nodes, state-of-the-art interconnect

Task:

- Image Classification (ResNet-152 on ImageNet)
- Single Node time (TensorFlow): **19 days**
- 1024 Nodes: **25 minutes** (***in theory***)

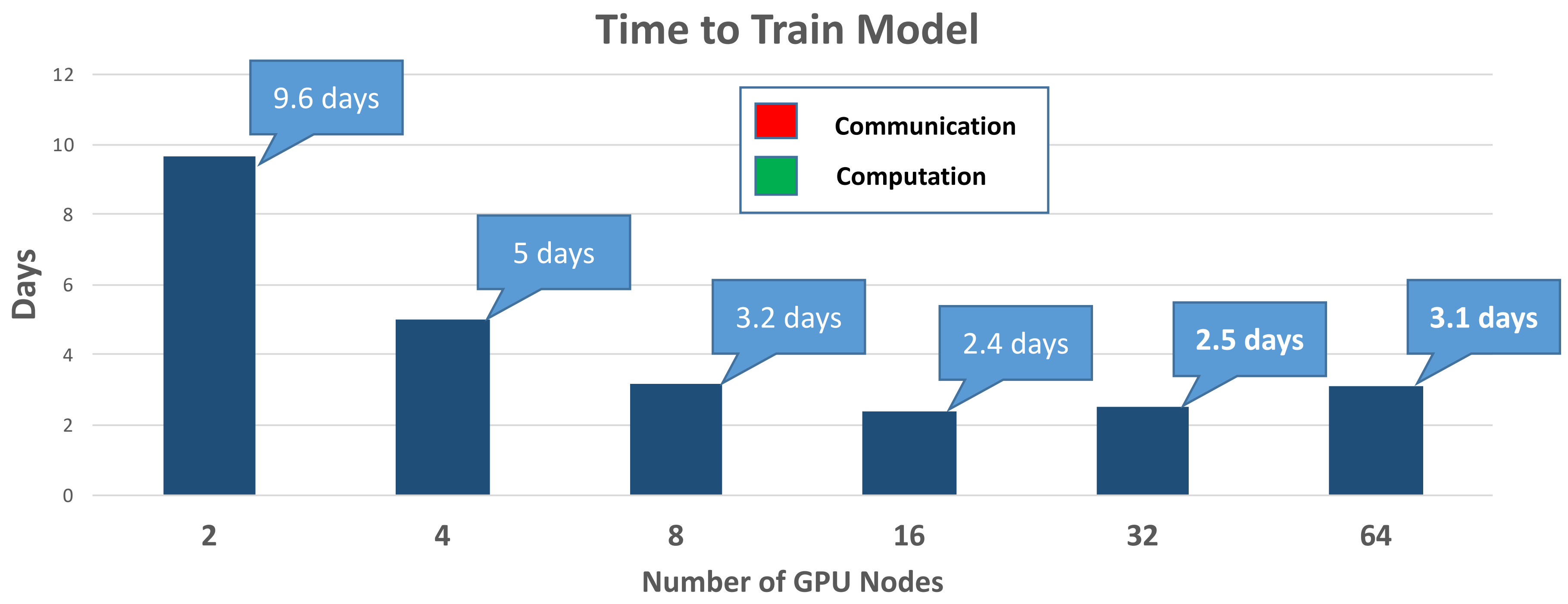
The Scalability Problem

CSCS: Europe's Top Supercomputer (World 4th)

- 4500+ GPU Nodes, state-of-the-art interconnect

Task:

- Image Classification (ResNet-152 on ImageNet)
- Single Node time (TensorFlow): **19 days**
- 1024 Nodes: **25 minutes** (***in theory***)



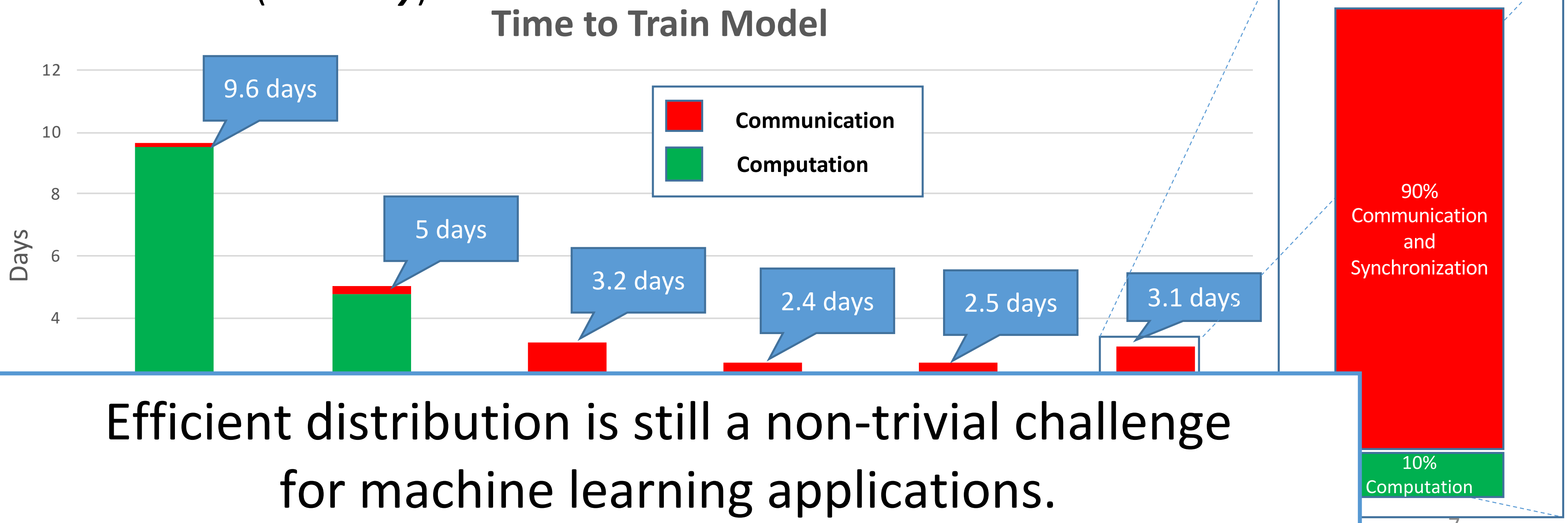
The Scalability Problem

CSCS: Europe's Top Supercomputer (World 4th)

- 4500+ GPU Nodes, state-of-the-art interconnect

Task:

- Image Classification (ResNet-152 on ImageNet)
- Single Node time (TensorFlow): **19 days**
- 1024 Nodes: **25 minutes** (***in theory***)

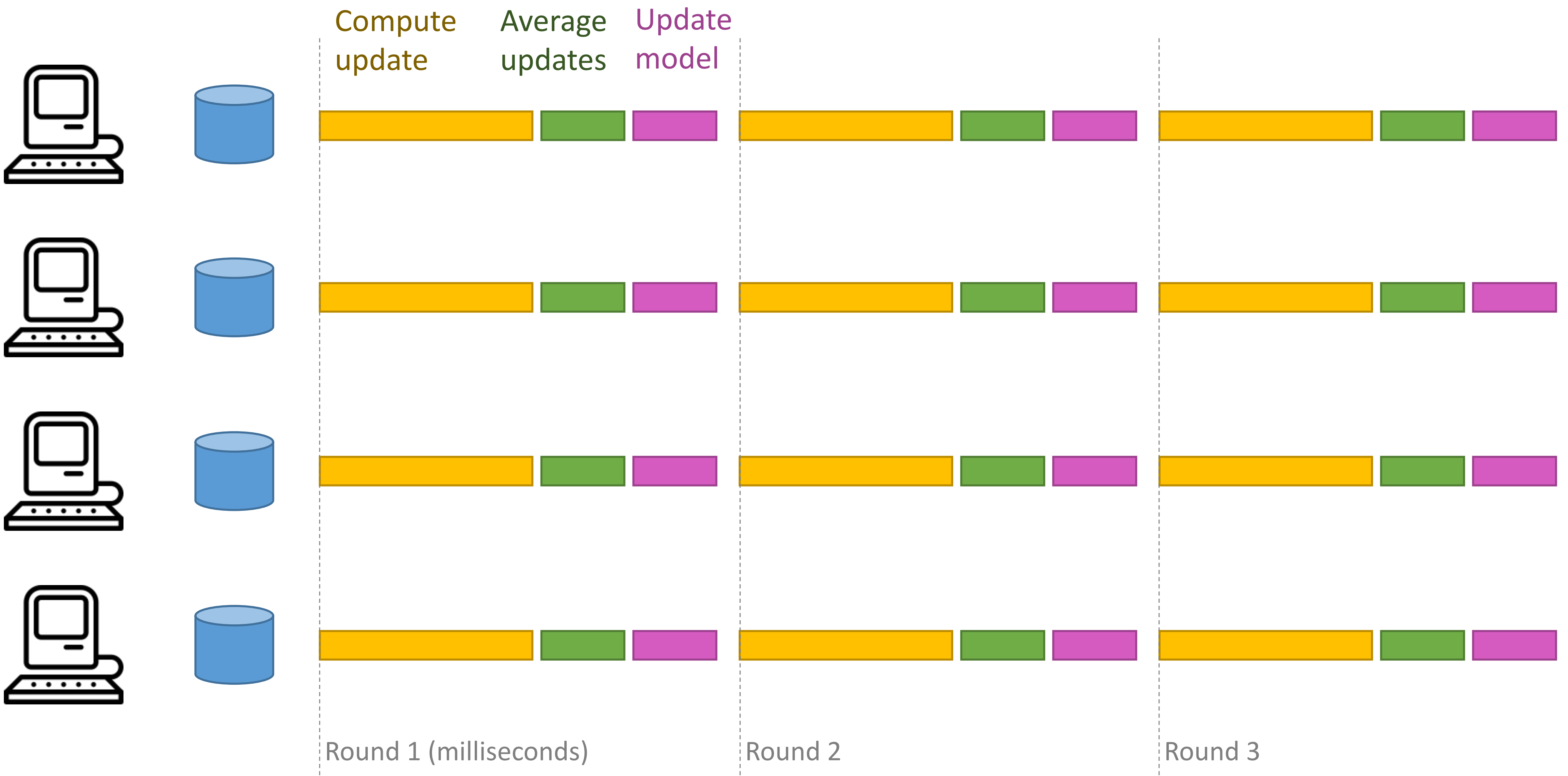


Efficient distribution is still a non-trivial challenge for machine learning applications.

The Algorithm: Parallel Stochastic Gradient Descent

Synchronous Message-Passing System

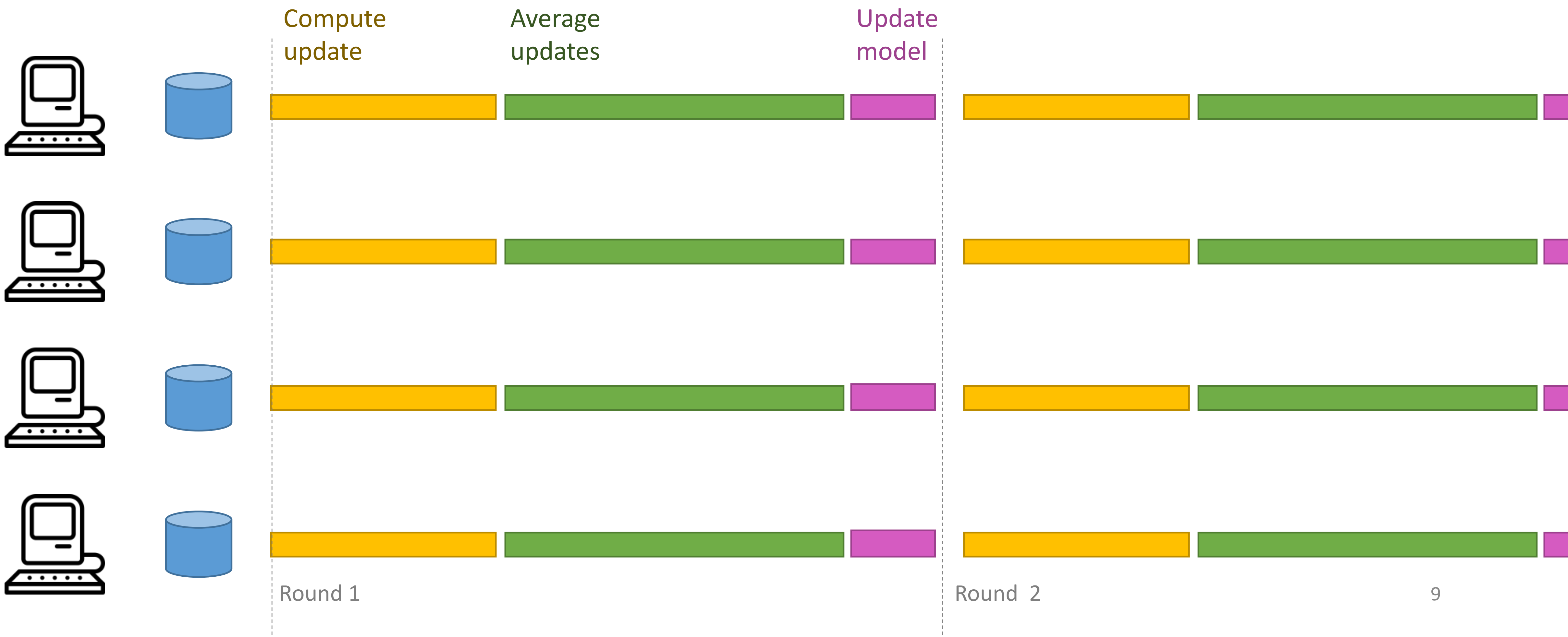
- **n** nodes, fully-connected communication topology



Parallel SGD (**large** models)

Synchronous Message-Passing System

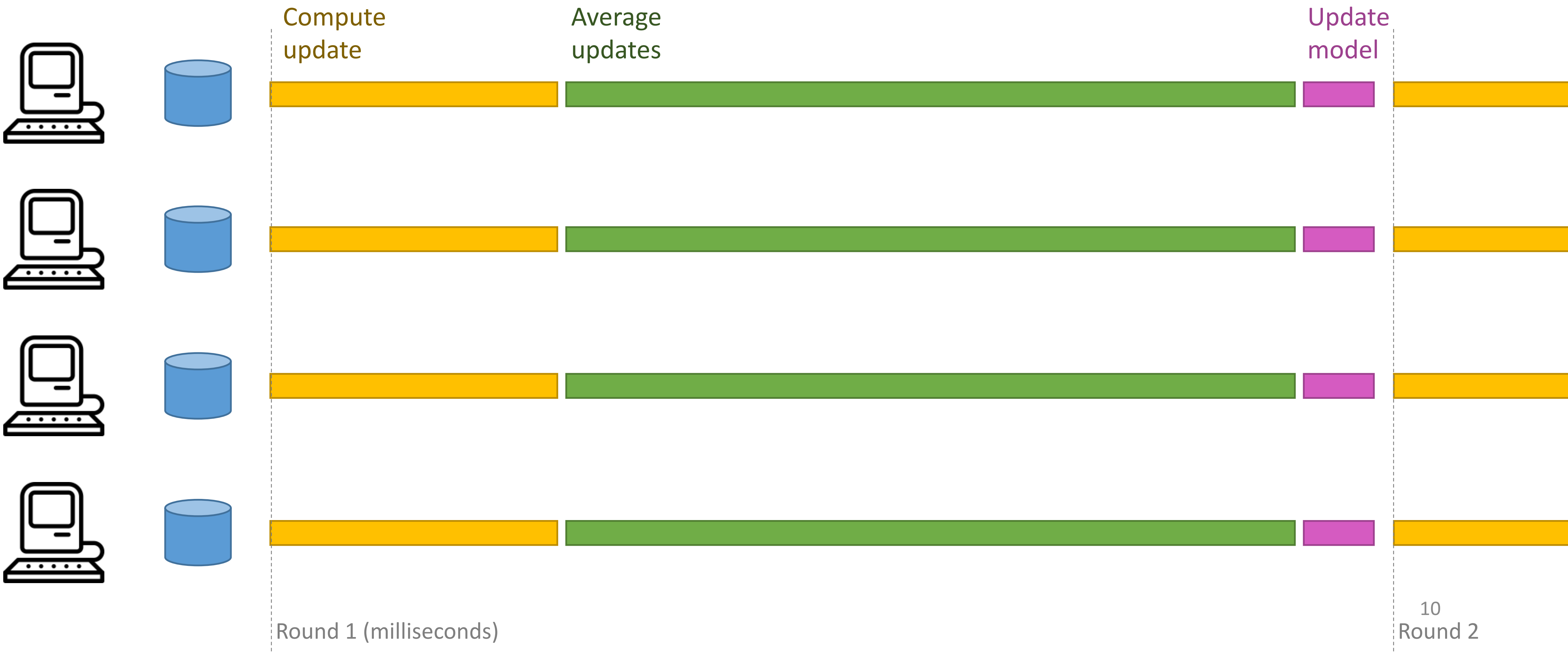
- **n** nodes, fully-connected communication topology



Parallel SGD (*really large* models)

Synchronous Message-Passing System

- **n** nodes, fully-connected communication topology



Today's Talk

Synchronization-Efficient Algorithms for Scalable Machine Learning

Quantization — **Sparsification** — **Efficient Aggregation**

Trade-offs: compression vs **convergence** vs **parametrization.**

ScaleML: An open-source framework implementing these techniques

Overview & Open Problems

The General Setting

Given:

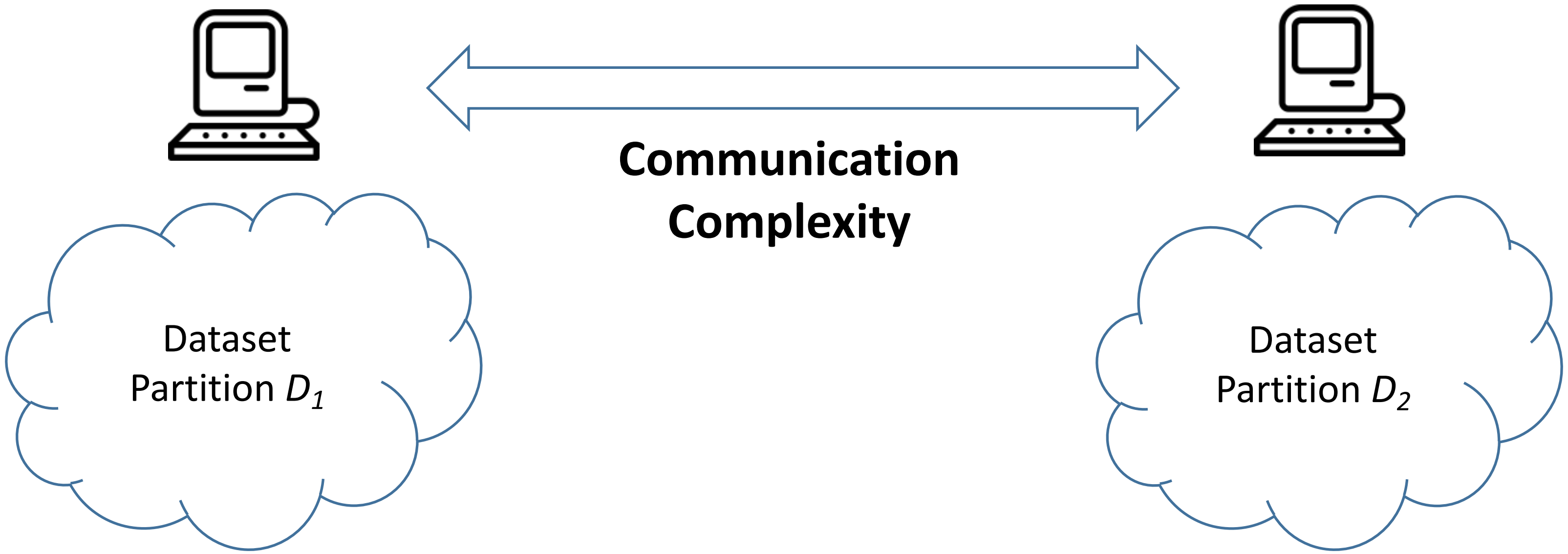
- **n** nodes, synchronous message-passing, fully-connected topology
- Dataset ***D***: node p_i is assigned dataset partition $\boldsymbol{D_i}$
- Loss function **Loss(x, e)** = how “good” is the prediction of model **x** on example **e**

Wanted:

$$\text{model } \boldsymbol{x} \text{ minimizing } f(\boldsymbol{x}) = f_1(\boldsymbol{x}) + f_2(\boldsymbol{x})$$

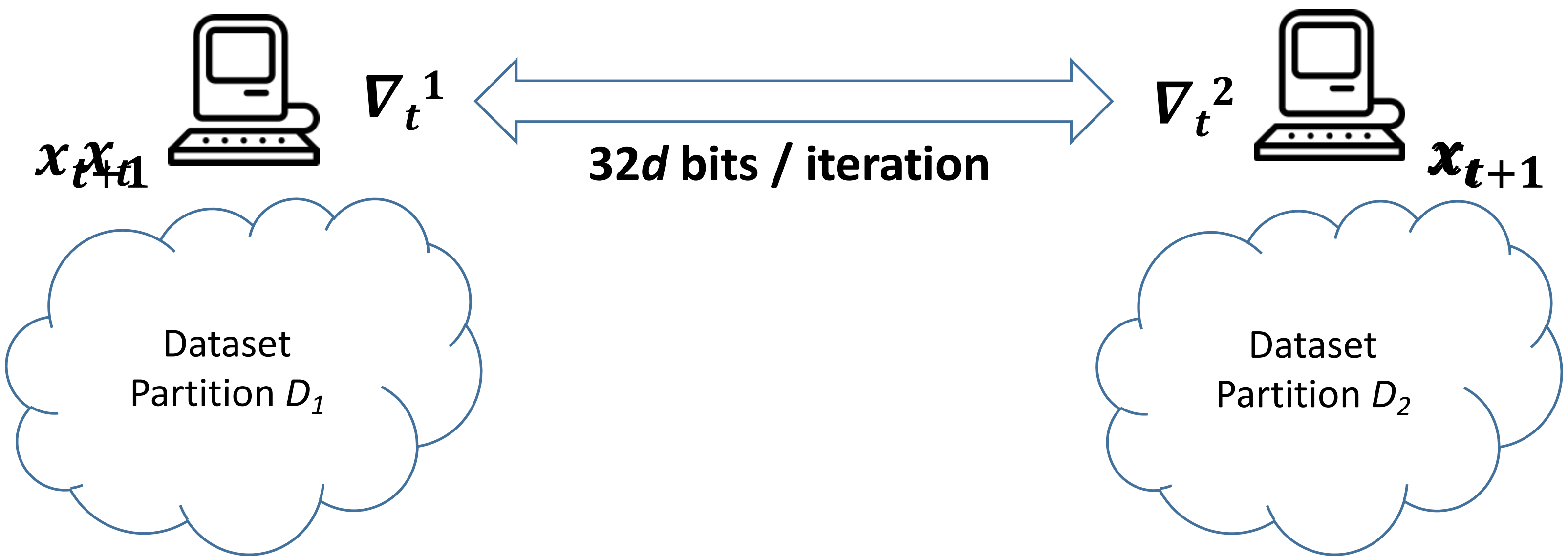
$f_1(x) = \mathbf{E}_{\text{e from D1}}$ [Loss(x, e)]

$f_2(x) = \mathbf{E}_{\text{e from D2}}$ [Loss(x, e)]



The Algorithm: Data-Parallel Stochastic Gradient Descent

- Each node maintains a copy of the "model/parameter" $\boldsymbol{x}$
- In each iteration $\boldsymbol{t}$, until convergence:
  - Each node *i* selects a sample e_i uniformly at random from D_i
  - It **computes** the update $\boldsymbol{\nabla_t}^{\boldsymbol{i}}$ = the **gradient** of x_t at e_i w.r.t. the Loss
  - Nodes **average** their updates: $\boldsymbol{\nabla_t} = (\boldsymbol{\nabla_t}^{\mathbf{1}} + \boldsymbol{\nabla_t}^{\mathbf{2}})/\mathbf{2}$
  - **Update model:** $\boldsymbol{x_{t+1}} = \boldsymbol{x_t} - \boldsymbol{\eta_t \nabla_t}$, where $\boldsymbol{\eta_t}$ is the learning rate.



Example: **Distributed Mean Estimation**

- Given distribution $\boldsymbol{D}$, find a parameter $\boldsymbol{x} \in \mathbb{R}^d$ which minimizes
$$\mathbf{E}_{e \text{ in } D}\left[||x - e||^2\right].$$
- In each iteration $\boldsymbol{t}$ until convergence:
  - Each node *i* selects a sample e_i uniformly at random from its local set
  - It **computes** the gradient of its estimate $\boldsymbol{\nabla_t}^{\boldsymbol{i}} = \boldsymbol{e_i} - \boldsymbol{xt}$
  - Nodes **average** their gradients to obtain $\boldsymbol{\nabla_t = (e_1 + e_2)/2 - xt}$, **and update their estimates by** $\boldsymbol{x_{t+1} = x_t - \eta_t \nabla_t}$.

The SGD algorithm remains roughly the same whether we are optimizing complex neural networks or solving classic regression.

Why does averaging / parallelism help?

Intuition: two random samples are better than one!

A Bit of Theory (1)

- Assume we wish to minimize a differentiable function $f : \mathbb{R}^d \text{->} \mathbb{R}$
- We apply the classic SGD iteration

$$\boldsymbol{x_{t+1}} = \boldsymbol{x_t} - \boldsymbol{\eta_t \nabla_t (xt)}, \quad \text{where } \boldsymbol{E}_{e \text{ in } D}[\boldsymbol{\nabla_t (xt)}] = \boldsymbol{\nabla f(x_t)}.$$

- Let $\boldsymbol{E}\left[||\boldsymbol{\nabla_t(x)} - \boldsymbol{\nabla f(x)}||^2\right] \leq \boldsymbol{\sigma^2}$ (variance bound)

<u>Theorem</u> [e.g. Bubeck15]**:** Given f **convex** and **smooth**, and $R^2 = ||x_0 - x^*||^2$.
If we run SGD for $\boldsymbol{T} = \boldsymbol{\mathcal{O}}(\boldsymbol{R^2} \frac{\boldsymbol{2\sigma^2}}{\boldsymbol{\varepsilon^2}})$ iterations, then

$$\boldsymbol{E}\left[f\left(\frac{1}{T}\sum_{t=0}^{T} x_t\right)\right] - f(x^*) \leq \varepsilon.$$

A Bit of Theory (2)

- Assume we wish to minimize a differentiable function f : $\mathbb{R}^d$-> $\mathbb{R}$
- We apply the classic SGD iteration

$$\boldsymbol{x_{t+1}} = \boldsymbol{x_t} - \boldsymbol{\eta_t \nabla_t (xt)}, \quad \text{where } \boldsymbol{E_{e \text{ in } D}[\nabla_t(xt)] = \nabla f(x_t)}.$$

- Assume we are ***averaging*** over ***P*** gradient estimators.
  Then $\boldsymbol{E}\left[||\boldsymbol{\nabla_t}(\boldsymbol{x}) - \boldsymbol{\nabla f}(\boldsymbol{x})||^2\right] \leq \boldsymbol{\sigma^2/P}$.

Trade-off: lower variance versus **cost of averaging.**

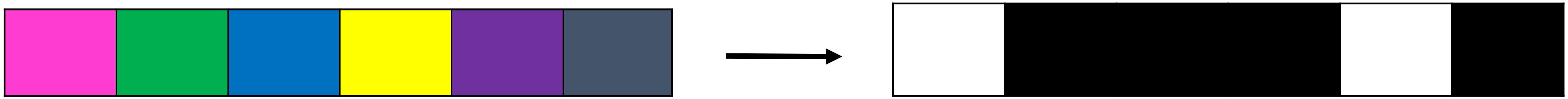
Theorem [e.g. Bubeck15]**:** Given f **convex** and **smooth**, and $R^2 = ||x_0 - x^*||^2$.
If we run SGD for $\boldsymbol{T} = \boldsymbol{\mathcal{O}}(\ \boldsymbol{R^2} \frac{\boldsymbol{2\sigma^2}}{\boldsymbol{P\varepsilon^2}})$ iterations, then

$$\boldsymbol{E}\left[f\left(\frac{1}{T}\sum_{t=0}^{T} x_t\right)\right] - f(x^*) \leq \varepsilon.$$

Today’s Talk

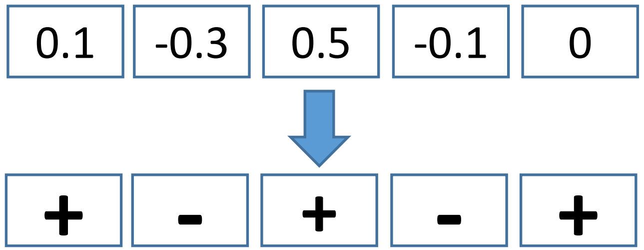
Communication-Efficient Algorithms for Scalable Machine Learning

Method 1: Gradient Quantization



1BitSGD Quantization
[Microsoft Research, Seide et al. 2014]

Gradient vector at a node:



$avg_+ = +0.2$

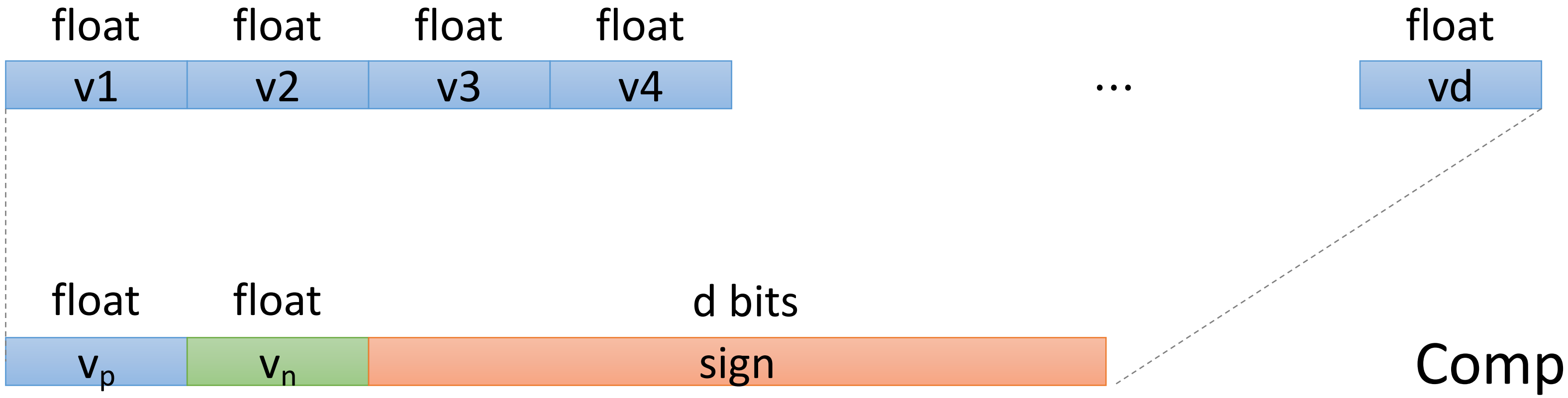
$avg_- = -0.2$

Quantization function

$$Q_i(v) = \begin{cases} avg_+ & \text{if } v_i \geq 0, \\ avg_- & \text{otherwise} \end{cases}$$

where $avg_+ = \text{mean}([v_i \text{ for } i: v_i \geq 0])$, $avg_- = \text{mean}([v_i \text{ for } i: v_i < 0])$

Accumulate the quantization error locally, and apply to the next update!



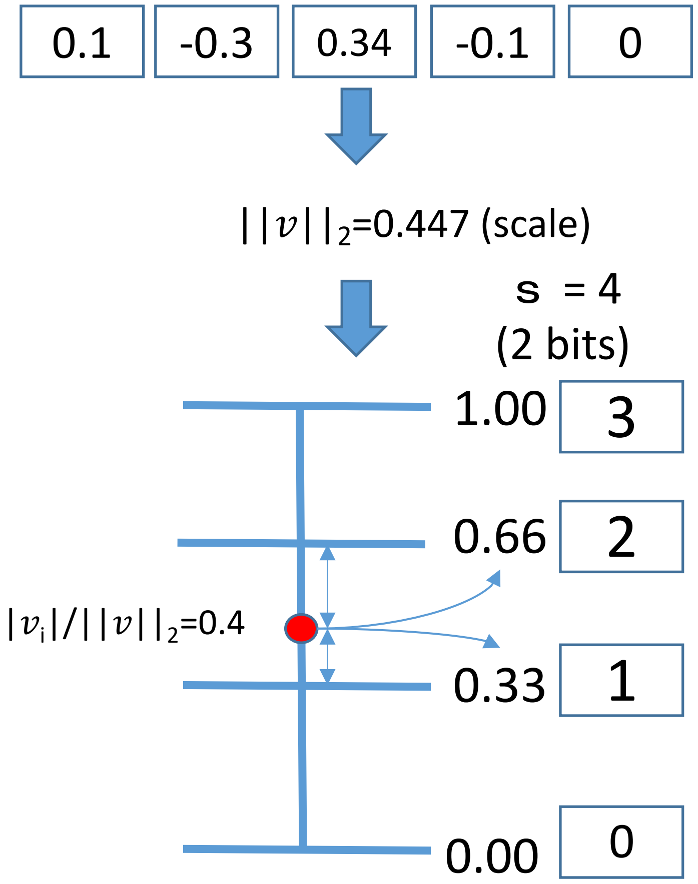
Compression rate ≈ 32x

Does not always converge!

Seide et al (2014) "1-Bit Stochastic Gradient Descent and its Application to Data-Parallel Distributed Training of Speech DNNs"

Stochastic Quantization

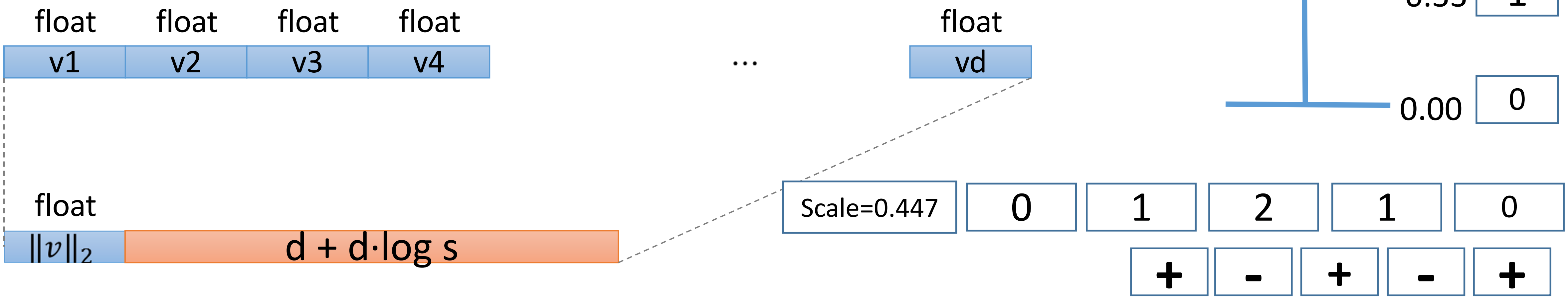
[Alistarh, Grubic, Li, Tomioka, Vojnovic, NeurIPS17]



Quantization function

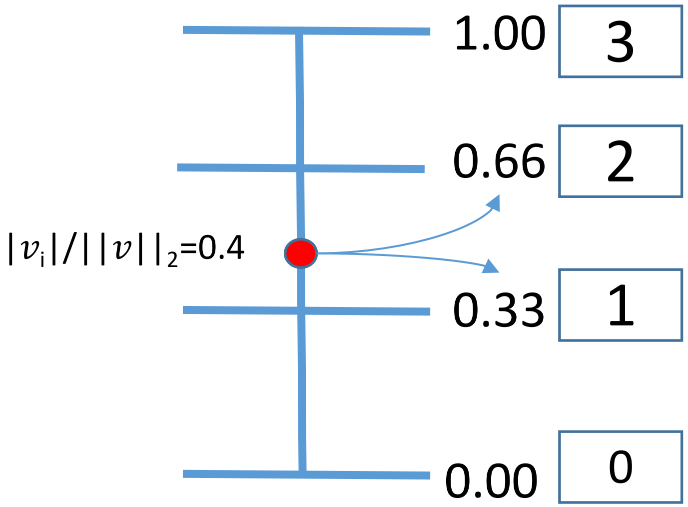
$$Q[v; s] = \|v\|_2 \cdot \mathrm{sgn}(v_i) \cdot \xi_i(v, s)$$

The random variable ξ_i encodes the integer quantization level for v_i.



Compression ratio ≈ 32/(log s+1)

QSGD Properties



Quantization function

$$Q[v_i; s] = \|v\|_2 \cdot \operatorname{sgn}(v_i) \cdot \xi_i(v, s)$$

- Properties

1. **Unbiasedness**

$$E\big[Q[v_i; s]\big] = v_i$$

-> ensures convergence since $E[Q[\nabla_t(x_t)]] = \nabla f(x_t)$.

2. **Sparsity**

$$E[nonzeros\ (Q[\vec{v}, s]) \leq s^2 + \sqrt{d}$$

-> intuitively ensures some compression

3. **Variance bound**

Trade-off between sparsity and variance!

$$E[\|Q[v; s]\|_2^2] \leq \left(1 + \frac{\sqrt{d}}{s}\right) \cdot \|v\|_2^2$$

-> bounded variance -> *fast* convergence

(Variance increase is 2 for $s = \sqrt{d}$)

QSGD Compression

<u>Informal Claim</u> [QSGD]: There exists a setting of parameters for which QSGD converges at **most 2x slower** than the full-precision baseline, and sends more than **10x less bits** per iteration.

Recently [Ramezani et al., 2019] improved on these guarantees.

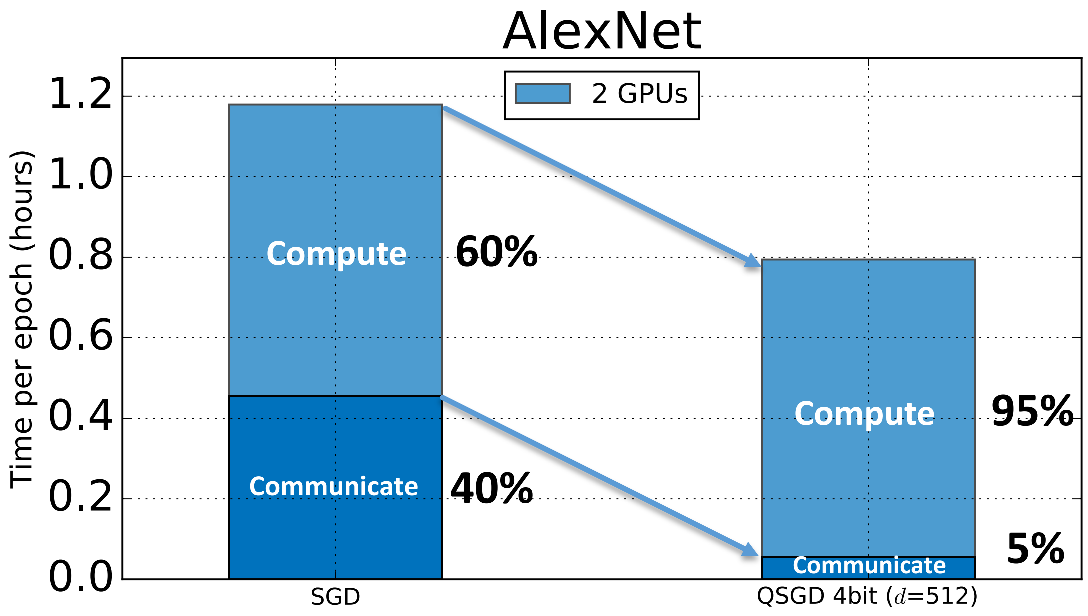
Proof sketch:

- **Idea1**: Assume we are implementing $s = \sqrt{d}$ integer quantization levels. We notice that very few vector entries can be quantized to **the top integers: values are normalized with respect to $\|v\|_2$, so not all can be large.**
- **Idea2**: The resulting "plain text" is a sequence of integers of different frequencies. We can use **custom arithmetic coding** to encode this sequence efficiently.

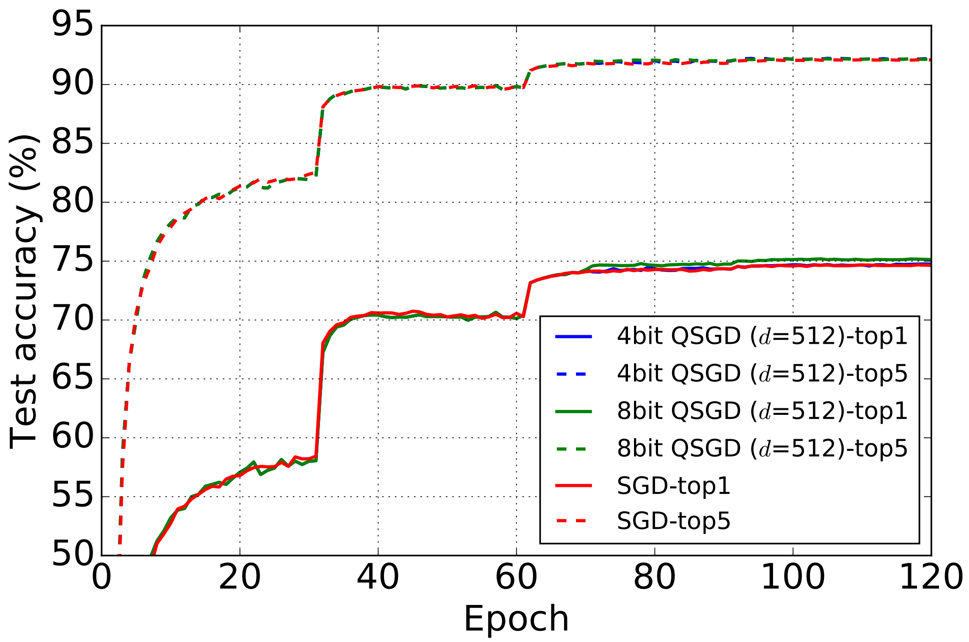
<u>Note</u>: The QSGD compression-variance trade-off is *tight:*
Any algorithm sending < B bits per round will induce *d* / B additional variance.
QSGD can match the Ω (d (log d + log (1 / ε))) bit lower bound of [Tsitsiklis & Luo, 1986].

Does it actually work?

- **Amazon EC2 p2.xlarge multi-GPU server**
- **AlexNet model (60M params)** x **ImageNet dataset** x **2 GPUs**
- **QSGD 4bit quantization (s = 16)**
- **No additional hyperparameter tuning**



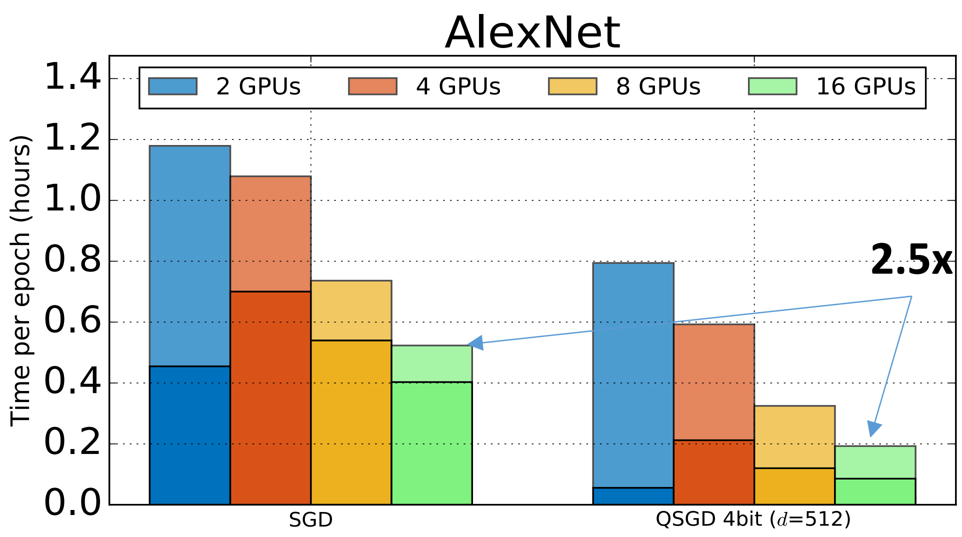
SGD vs QSGD on AlexNet.

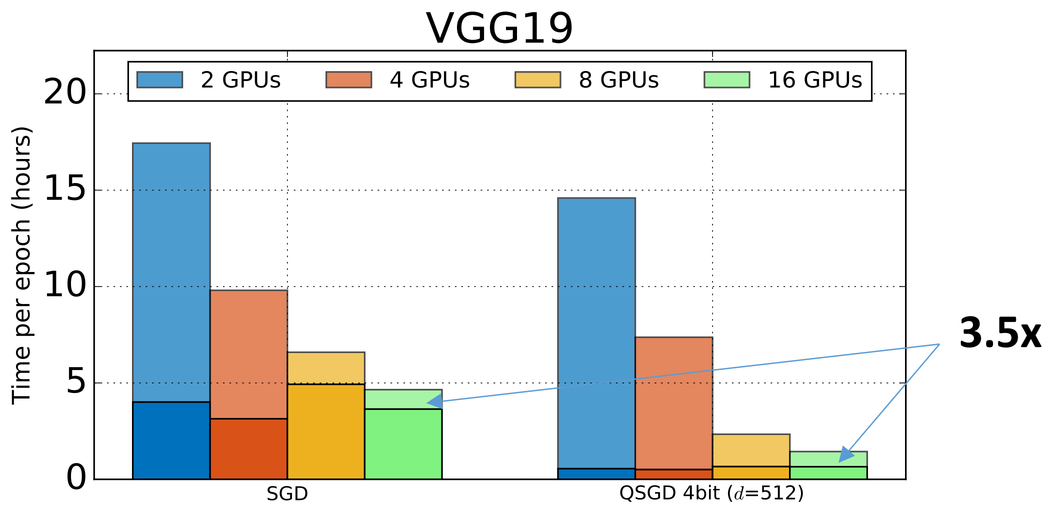


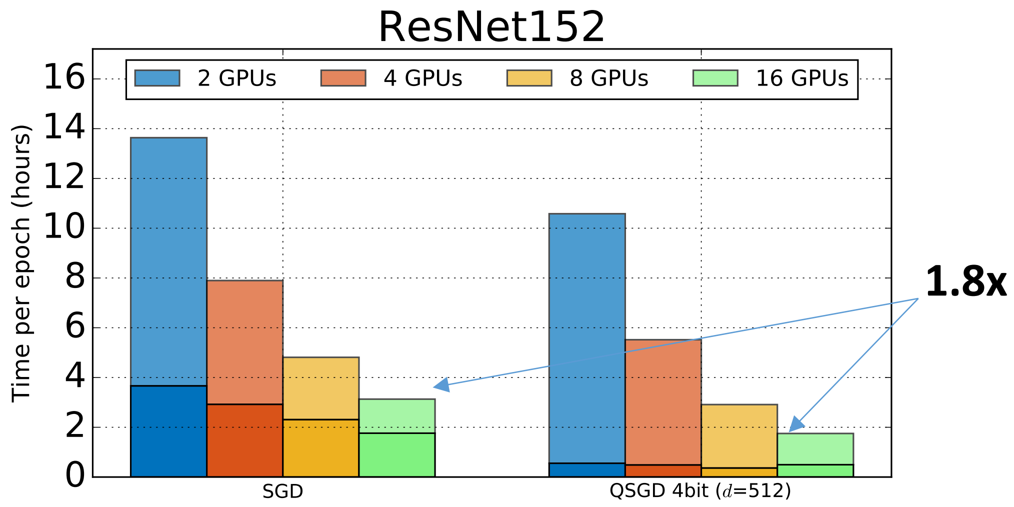
AlexNet on ImageNet: Accuracy

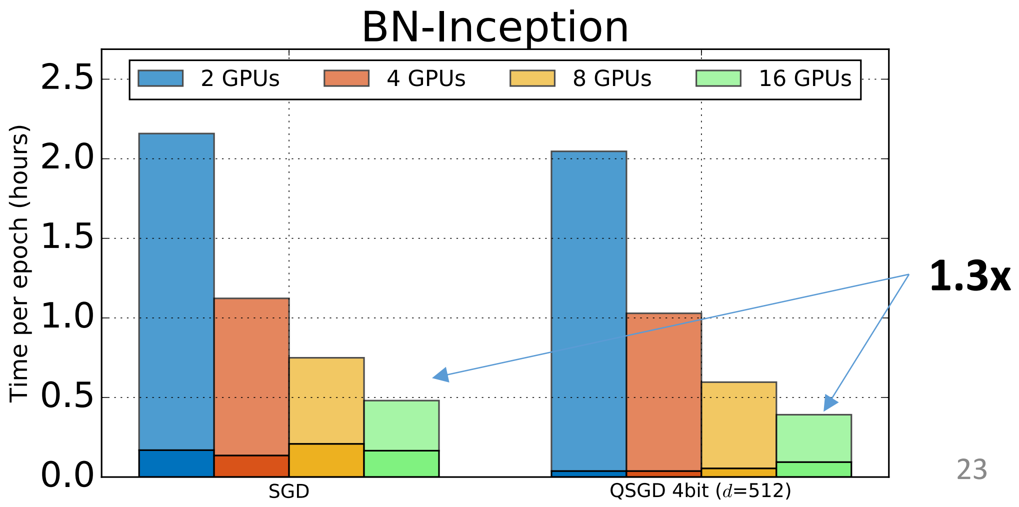
Experiments: “Strong” Scaling

- State-of-the-art image classification on ImageNet, 16-GPU EC2 server



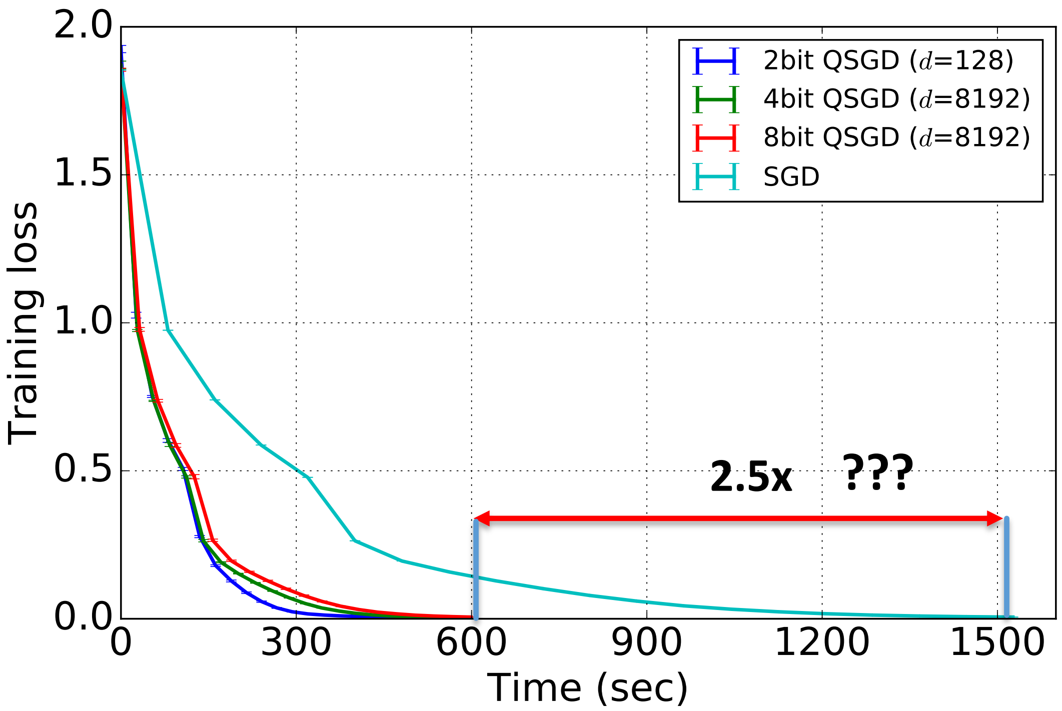






Accuracy vs Time on a Speech Model (LSTM)

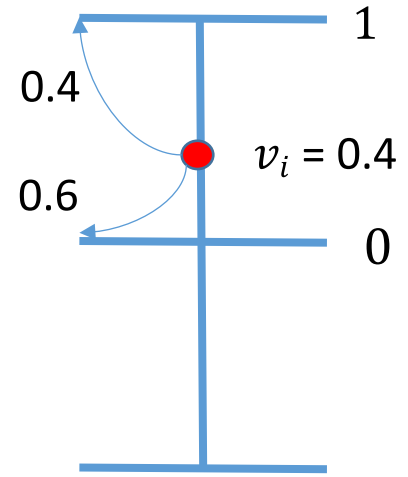
- Speech Recognition Dataset (CMU AN4)
- Encoder-Decoder LSTM Model
- **2 GPU nodes**



Speech Recognition (3-Layer LSTM)

Summary: Quantization

[1bitSGD, 2014], [QSGD, NeurIPS17], [TernGrad, NeurIPS17], [NUQSGD, 2019]

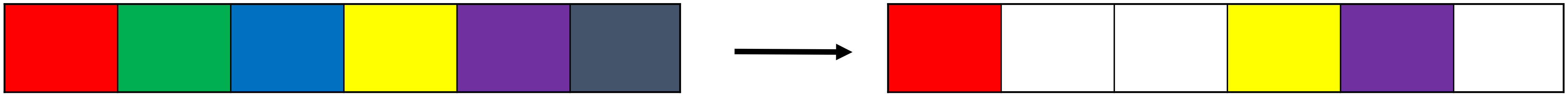


1. **How much compression?**
   - Usually **< 32x**, since it's just bit width reduction
   - Cannot do better without large variance <-> convergence loss
2. **Does it guarantee convergence/accuracy?**
   - **Theory:** Yes (QSGD). Under strong assumptions (1bitSGD).
   - **Practice**: Extensive testing (30'000 node hours) shows QSGD (4bit) preserves accuracy for all neural networks [Grubic et al., EDBT18].
3. **Do they need additional parameter tuning?**
   - **TernGrad, 1BitSGD:** Yes.
   - **QSGD**: No.

Talk Outline

Communication-Efficient Algorithms for Scalable Machine Learning.

Method 2: Structured Sparsification

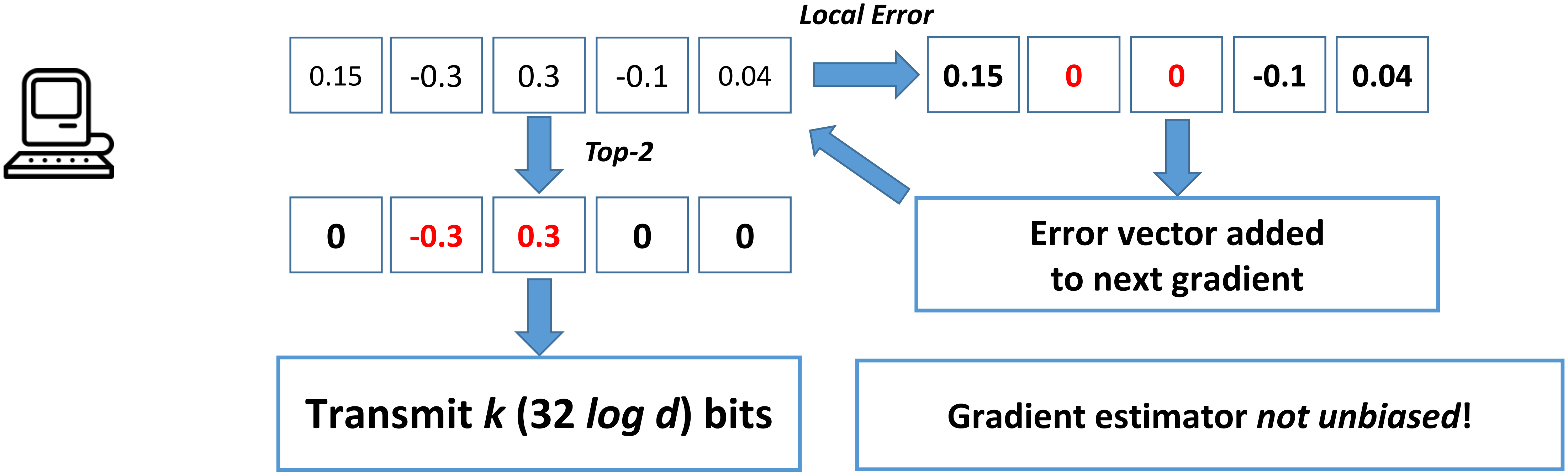


Method 2: Structured Sparsification

[Strom, 2016; Dryden et al., 2017; Aji & Heafield, 2017; Alistarh & Grubic 2018; Lin et al., 2018]

Fix an integer parameter *k*

- *Only send top k from each gradient vector, in order of absolute values*
- *Accumulate the unsent values locally*



Method 2: Sparsification

[Dryden et al., 2016; Aji & Heafield, 2017; Lin et al., 2018]

1. **How much compression?**
   - **d / (*k log d*),** potentially huge
2. **Does it still guarantee convergence?**
   - **Experimentally:** up to **400x** compression with no accuracy loss via extremely careful parameter tuning **[Lin et al. 2018, ICLR18]**
   - **Theory: Yes! [Konstantinov et al., NeurIPS 2018]**
3. **Do they need additional parameter tuning?**
   - **Yes [Deep Gradient Compression: Lin et al., ICLR18]**
   - **No [ScaleML: Renggli et al., Supercomputing '19]**

Sparsification with Error Correction Converges

[Konstantinov et al., NeurIPS 2018, journal version in preparation]

<u>Informal Claim</u> [TopK SGD]: Under analytic assumptions, given **any smooth, (non-convex) function *f***, there exists a learning rate sequence s.t. TopK SGD ensures

$$min_{t=1,\dots,T}\ E\left[\left|\left|\nabla f(x_t)\right|\right|^2\right] \xrightarrow{T\to\infty} 0.$$

This suggests that TopK SGD will eventually converge to a local minimum of *f*.

Convergence rate depends linearly on the "density" parameter k / d.

Notes:

1. **The above guarantee is the best we can hope for in the non-convex case.**
2. **The technical argument reveals that TopK is a special case of *asynchronous SGD* [Hogwild!: Niu et al., 2011]**
3. **Key for convergence:** how much ***gradient norm*** is transferred in the **TopK**

This also works in the concurrent setting: threads have to write less!

Summary so far

Algorithmic methods for scalable distributed machine learning.

Quantization

Sparsification

Can provide order-of-magnitude communication reduction!

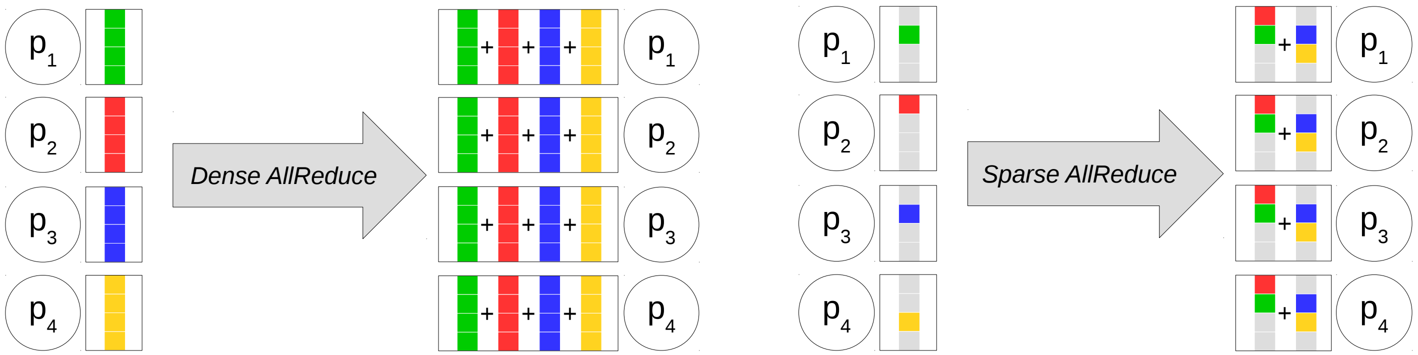
But how about software support?

Neither method supported by communication libraries
|(**MPI** implementations or **NVIDIA NCCL)**

ScaleML: A Scalable Communication Framework for ML

[Renggli, Ashkboos, Aghagolzadeh, Hoefler, Alistarh; Supercomputing 2019]

- **Communication framework with MPI-like semantics**
  - Implements distributed AllReduce operations (a.k.a. MPI collectives)
- Native support for **quantization** and **sparsity**
- **Efficient sparsity support is non-trivial:**
  the underlying sparsity distribution is unknown at runtime

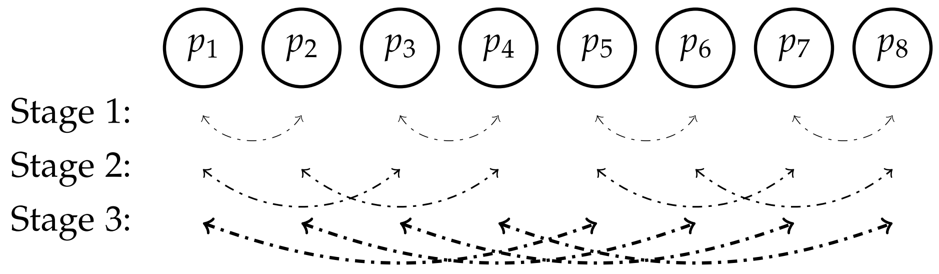


(a) Dense AllReduce (b) Sparse AllReduce

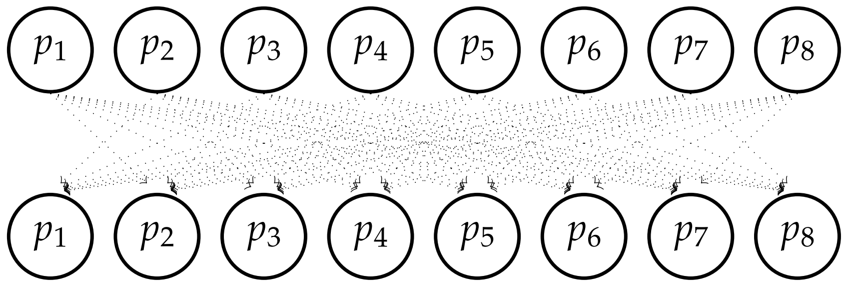
ScaleML: A Scalable Communication Framework for ML

[Renggli, Ashkboos, Aghagolzadeh, Hoefler, Alistarh; SuperComputing19]

- Q1: Can the data become ***dense during aggregation***?
  - Depending on this, **we switch to a dense *quantized* data representation**
- Q2: Is the system ***latency-dominated***, or ***bandwidth-dominated***?
  - Depending on this, we use **completely different communication patterns**



Optimal communication structure for
latency-dominated case
(Recursive Doubling)



Optimal communication structure for
bandwidth-dominated case
(Sparse AllGather)

Practical Performance

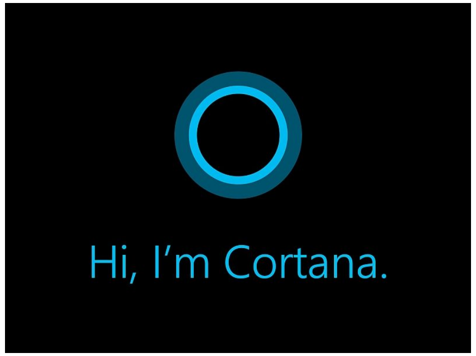
System	Model	#Nodes	Sparsity	End-to-end speedup
CSCS Piz Daint	2-Layer LSTM	8	**99.5% (+8bit QSGD)**	**2.6**x
Amazon EC2 Cluster	2-Layer LSTM	8	**99.5% (+8bit QSGD)**	**7.1**x

End-to-end training speedup
(LSTM for Natural Language Understanding / ATIS dataset)

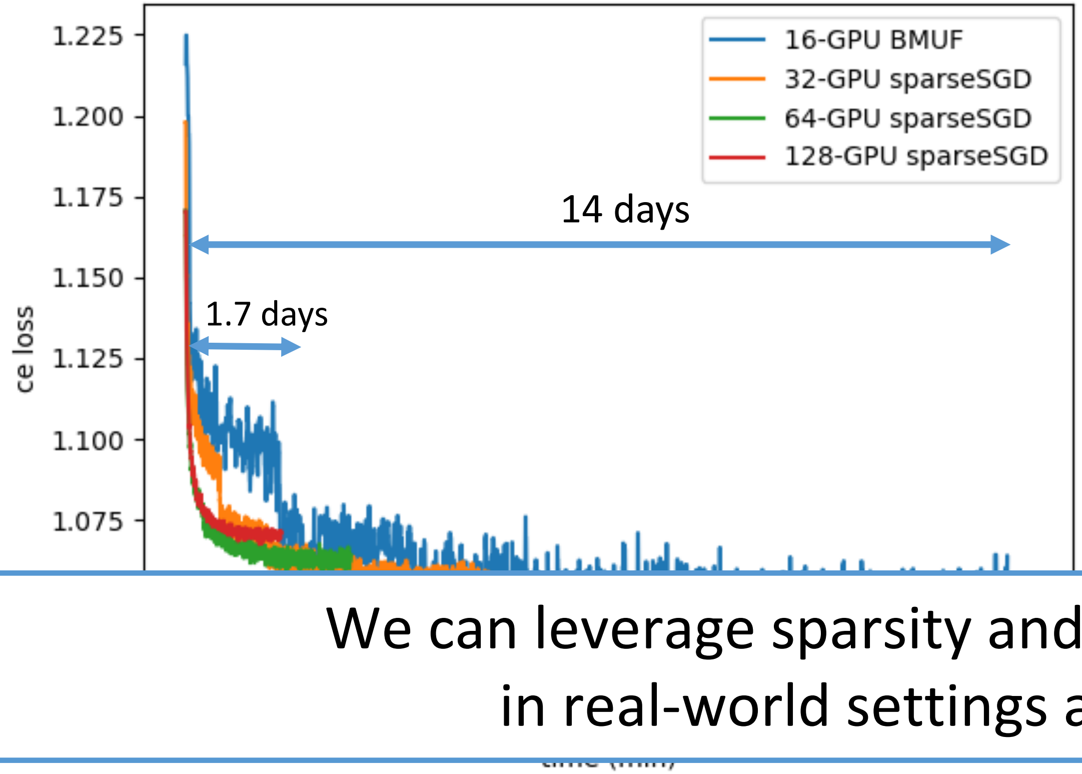
Works well in for a variety of other models/settings.

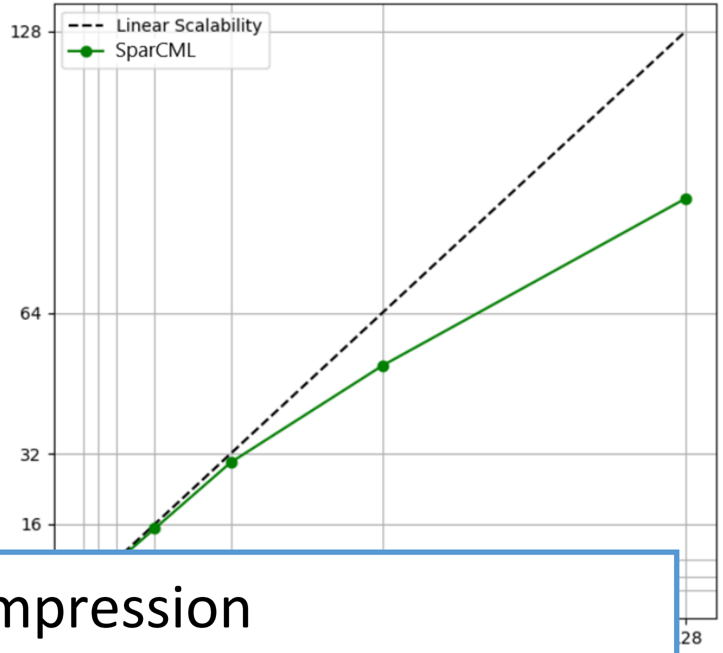
Is this useful **in the real world**?

Does it work at scale?



- Microsoft's Automated Speech Recognition Tool
- State-of-the-art recurrent model
- Baseline: Fine-Tuned Block-Momentum SGD (BMUF) [Zhu et al., 2016]
- We use 99.5% induced sparsity (k = 0.5%) and 8-bit quantization





We can leverage sparsity and compression
in real-world settings as well.

Summary

Algorithmic methods for scalable distributed machine learning

Quantization

Sparsification

Efficient Aggregation

Trade-offs: *compression* vs *speed* vs *parametrization*.

Distributed machine learning is ***wide open***

Topics I Couldn't Cover Today

Asynchronous Machine Learning

E.g. [Bertsekas & Tsitsiklis, 1986],
[Niu et al.; "Hogwild!", 2011]
[De Sa et al., "Async. Gibbs Sampling", 2016]
[Konstantinov et al., "Price of Asynchrony", 2018]

Fault-Tolerant Distributed Machine Learning

E.g. [Su & Vaidya, "Fault-Tolerant Optimization" 2016],
[Blanchard, El Mhamdi, Guerraoui, Steiner, "Byzantine SGD," 2017]
[Alistarh, Allen-Zhu, Li, "Optimal Byzantine SGD," 2018]

Happy to cover these in the Q&A!