Efficient Large-Scale Optimization by Population

Dan Alistarh
IST Austria

The Plan for Today

1. A Quick Introduction to Population Protocols

2. The Scaling Challenge in Distributed Machine Learning

3. Large-Scale Optimization via Population Protocols

What are Population Protocols?

A model for large-scale
decentralized distributed computation.

Three ingredients:

- Nodes
- Communication
- Computation

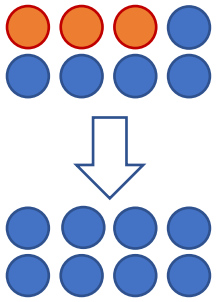
Computational Model

Population Protocols [AADFP'04, Dijkstra Award 2020]

- **Nodes** are ***simple, identical agents***
  - Each node is ***an anonymous*** finite state automaton
  - E.g., **a *molecule*** or ***a cell***

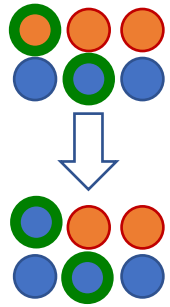
- **Interactions** are ***pairwise***, and follow a ***fair scheduler***
  - Communication graph is ***fully connected (clique)***
  - At each step, an **edge** chosen ***uniformly at random*** ("well-mixed solution")
  - Nodes update their state following interactions

- **Computation** is performed ***collectively***
  - The system ***should converge*** to configurations satisfying meaningful predicates on the input
  - No "fixed" decision time



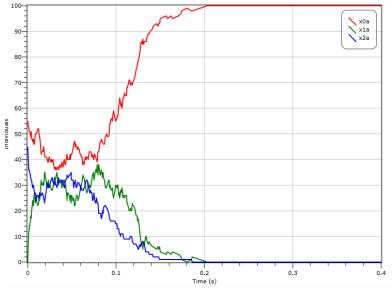
- A.k.a. Chemical Reaction Networks, Petri Net varieties

Complexity Measures



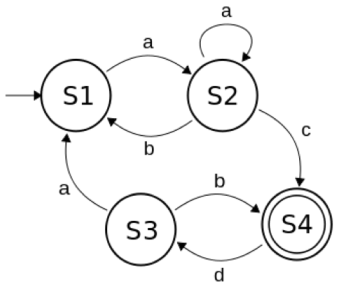
1. Time Complexity

- **Step** = a ***single pair*** interacts
  - Chosen uniformly at random
- **Parallel convergence time**
  - **#rounds** to convergence / **# nodes**
  - Alternative ***continuous-time*** definition exists

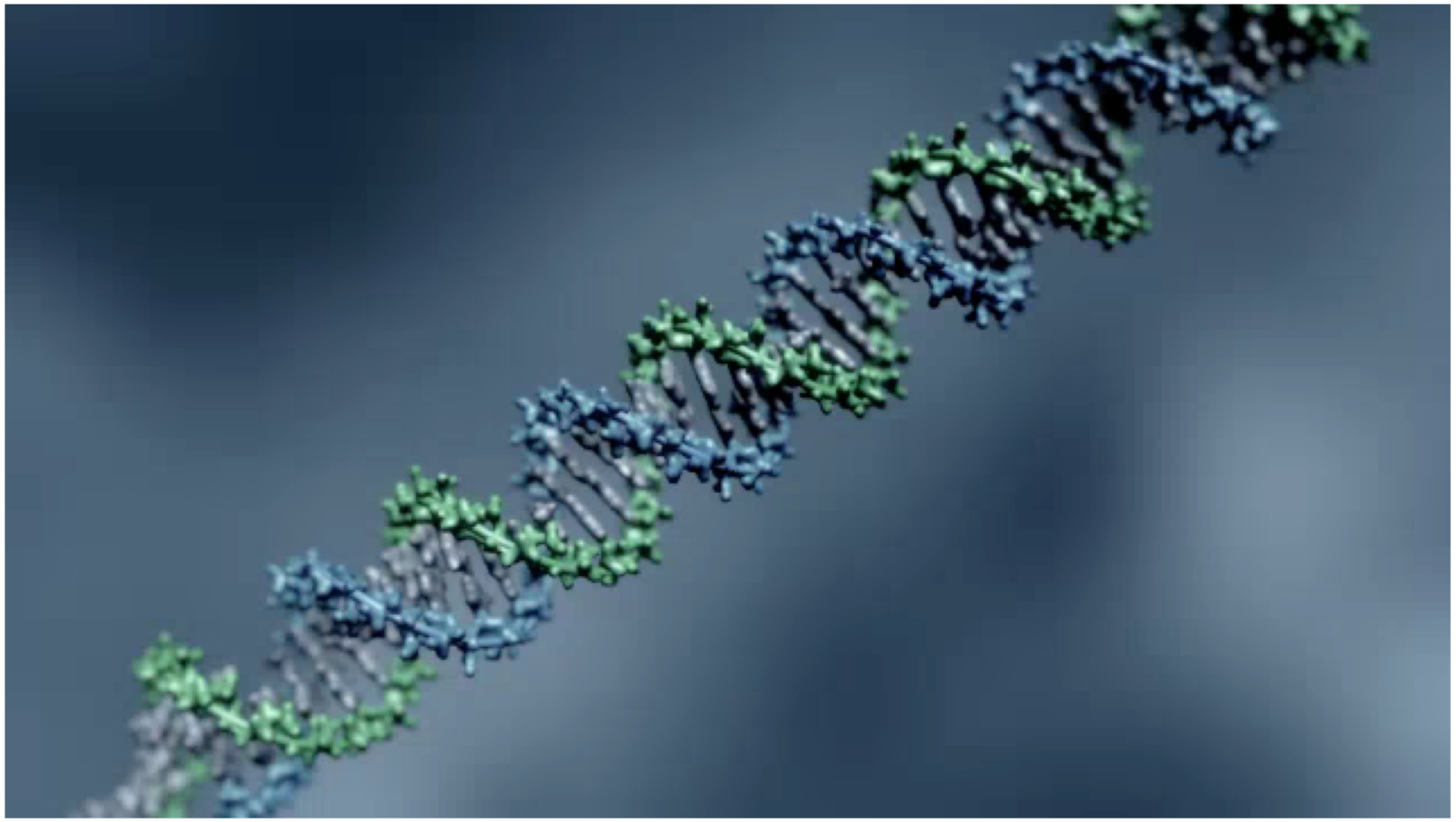


2. Space Complexity

- Number of ***distinct states*** per automaton
- Alternatively, #**memory bits** to encode state



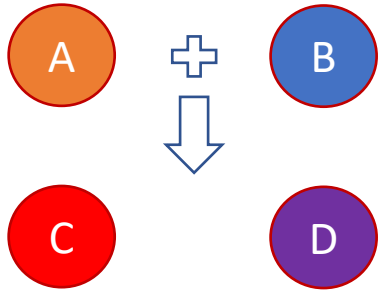
Implementable!



Courtesy of the Microsoft Research Biological Computation Group

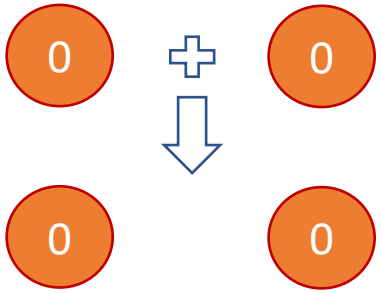
What can they compute?

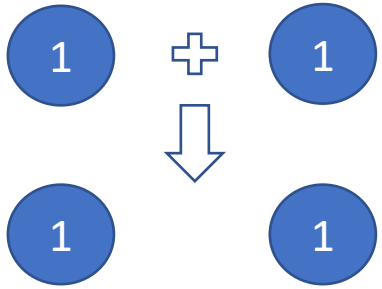
We can perform interactions of the type:

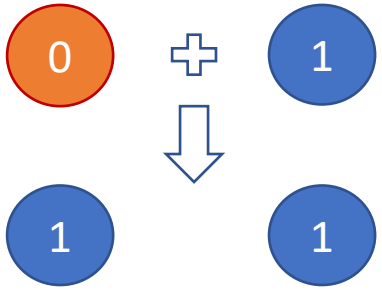


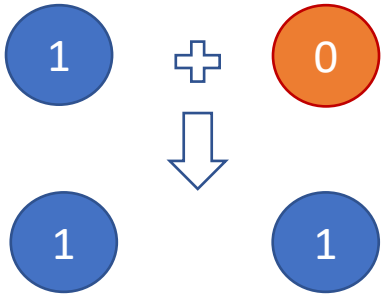
Example: the **OR** function

- **Initial** states: 0 or 1
- **Final** state:
  - If there exists a 1, then all 1.
  - Otherwise, all 0
- Protocol:







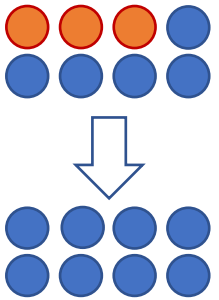


Can they compute anything *interesting*?

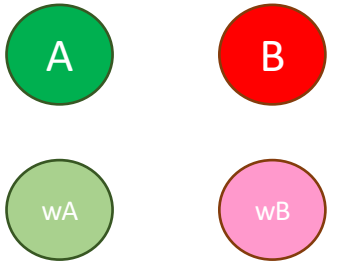
Theorem [AADFP]: The set of predicates computable by a population protocol is exactly the set of predicates expressible in Presburger arithmetic.

Majority ("Consensus")

- **Initial** states A, B
- **Output**:
  - **A** if **#A** > **#B** initially.
  - **B**, otherwise.
- **Fundamental task**
  - **Complexity**: [AAE08] & [DV12]; [PVV09] & [MNRS14]
  - Natural computation:
    the cell cycle switch implements approximate majority [CC12]
  - Implementation in DNA: [CDS+13, Nature Nanotechnology]

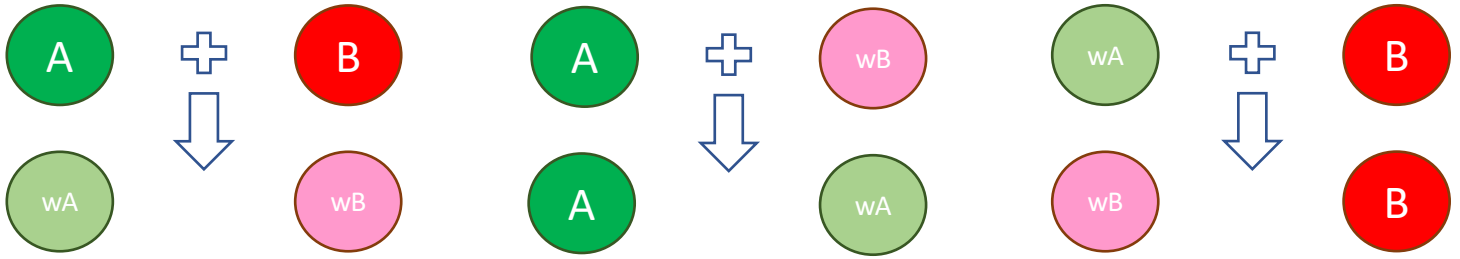


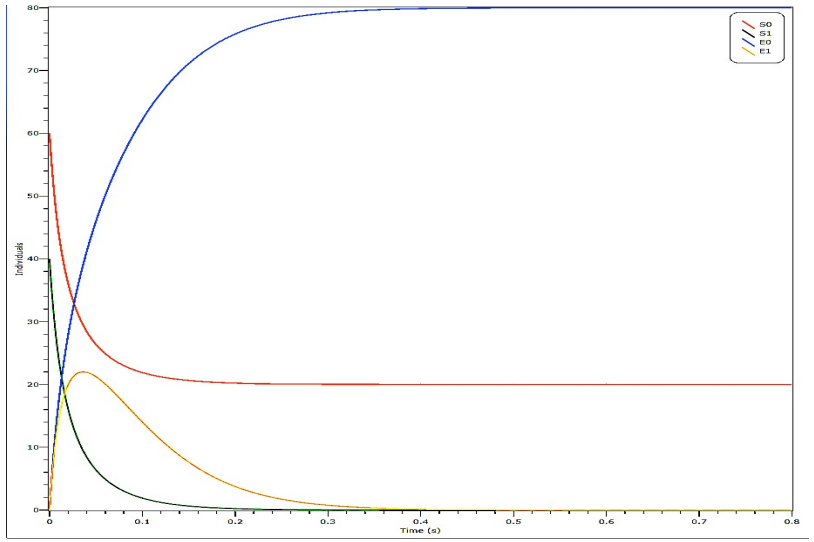
Solving Majority



4-State Exact Majority [PVV09] [MNRS14]

- Protocol:





Discrepancy/margin:
ε = |#A - #B| / n
Can be as small as
ε = O(1 / n).

Theorem [Draief & Vojnovic '12]: Given n nodes and discrepancy ε, the parallel running time of 4EM is O((log n) / ε).

Think of n = 6.023 x 10^{23}.

Can be Θ(n log n) ***parallel time*** if ε = constant / n.

The State of Majority

Algorithm/Lower Bound	States	Convergence Time
4-State Exact Majority [PVV09, MNRS14]	**4**	**Super-linear in n**
[MNRS14]	**3**	$\Omega(n)$

The State of Majority

Algorithm/Lower Bound	States	Convergence Time
4-State Exact Majority [PVV09, MNRS14]	4	Super-linear in n
[MNRS14]	3	$\Omega(n)$
Average-and-Conquer [AGV15]	$\frac{Ct.}{\epsilon}$	O(polylog n)
[AGV15]	4	$\Omega(n)$

The State of Majority

Algorithm/Lower Bound	States	Convergence Time
4-State Exact Majority [PVV09, MNRS14]	4	**Super-linear in n**
[MNRS14]	**3**	$\Omega(n)$
Average-and-Conquer [AGV15]	$\frac{Ct.}{\epsilon}$	**O(polylog n)**
[AGV15]	**4**	$\Omega(n)$
Split-Join Algorithm [AAEGR17]	$\mathbf{O}(\log^2 n)$	$\mathbf{O}(\log^3 n)$
[AAEGR17]	$\leq 1/2(\log\log n)$	$\Omega\left(\frac{n}{\text{polylog } n}\right)$

Simplified & Improved to $\mathbf{O}(\log^2 n)$ by Berenbrink et al., 2017

The State of Majority

Algorithm/Lower BoundO	States	Convergence Time
4-State Exact Majority [PVV09, MNRS14]	4	**Super-linear in n**
[MNRS14]	**3**	$\Omega(n)$
Average-and-Conquer [AGV15]	$\frac{Ct.}{\epsilon}$	**O(polylog n)**
[AGV15]	**4**	$\Omega(n)$
Split-Join Algorithm [AAEGR17]	$\mathbf{O}(\log^2 n)$	$\mathbf{O}(\log^3 n)$
[AAEGR17]	$\leq 1/2(\log\log n)$	$\Omega\left(\frac{n}{\text{polylog } n}\right)$
[AAG18]	**Θ (log n)**	O ($\log^2$ n)
[BKKP20]	**O(log n)**	O($\log^{3/2}$ n)

Simplified & Improved to $\mathbf{O}(\log^2 n)$ by Berenbrink et al., 2017

Taking a Step Back

Population Protocols Induce Non-Trivial Space-Time Trade-Offs

- Another good example: *Leader Election* [AAEGR17], [BGK20]
  - Time bound: **Θ (log n);** Space bound: **Θ (log log n)**

- Interesting model of ***extremely large-scale parallelism***

Can we apply them to anything else that's interesting?

The Plan for Today

1. A Quick Introduction to Population Protocols

2. The Scaling Challenge in Distributed Machine Learning

3. Large-Scale Optimization via Population Protocols

The Machine Learning “Cambrian Explosion”

CAT, DOG, DUCK

Image Classification
& Segmentation

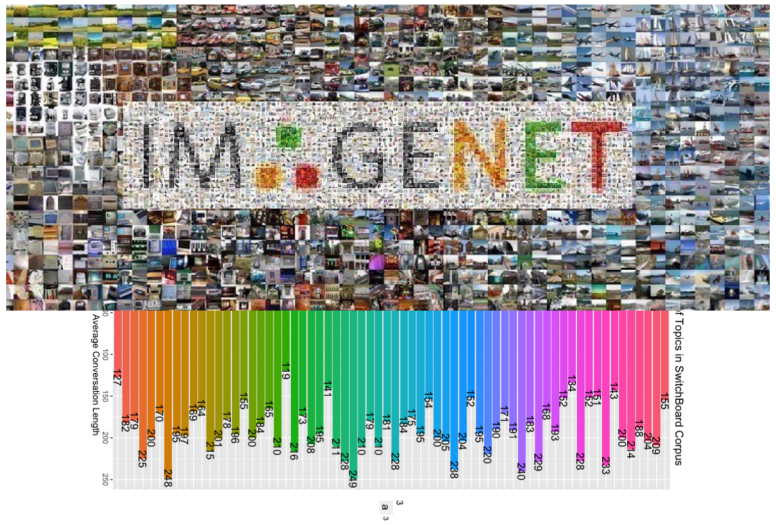
Speech Recognition
& Translation

Strategic Games
(Reinforcement Learning)

Machine Learning is here to stay:
existing technologies already have significant industry adoption.

Three Factors

Great Ideas



High Quality Data

Efficient Computation

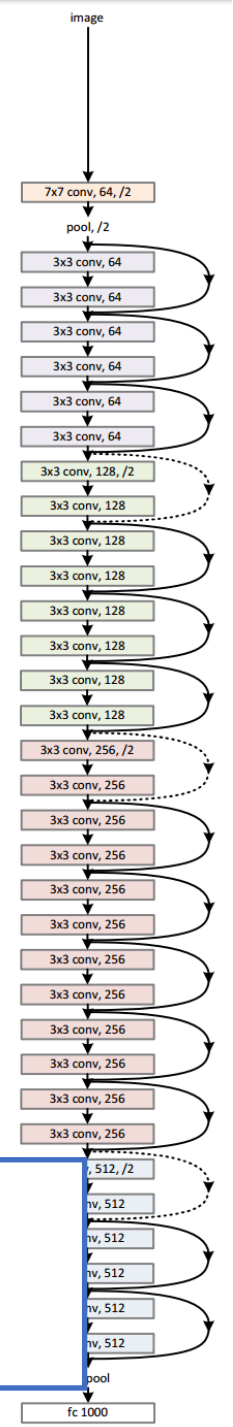
Distributed/parallel computing is the key enabler of computational speedups.

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 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***)

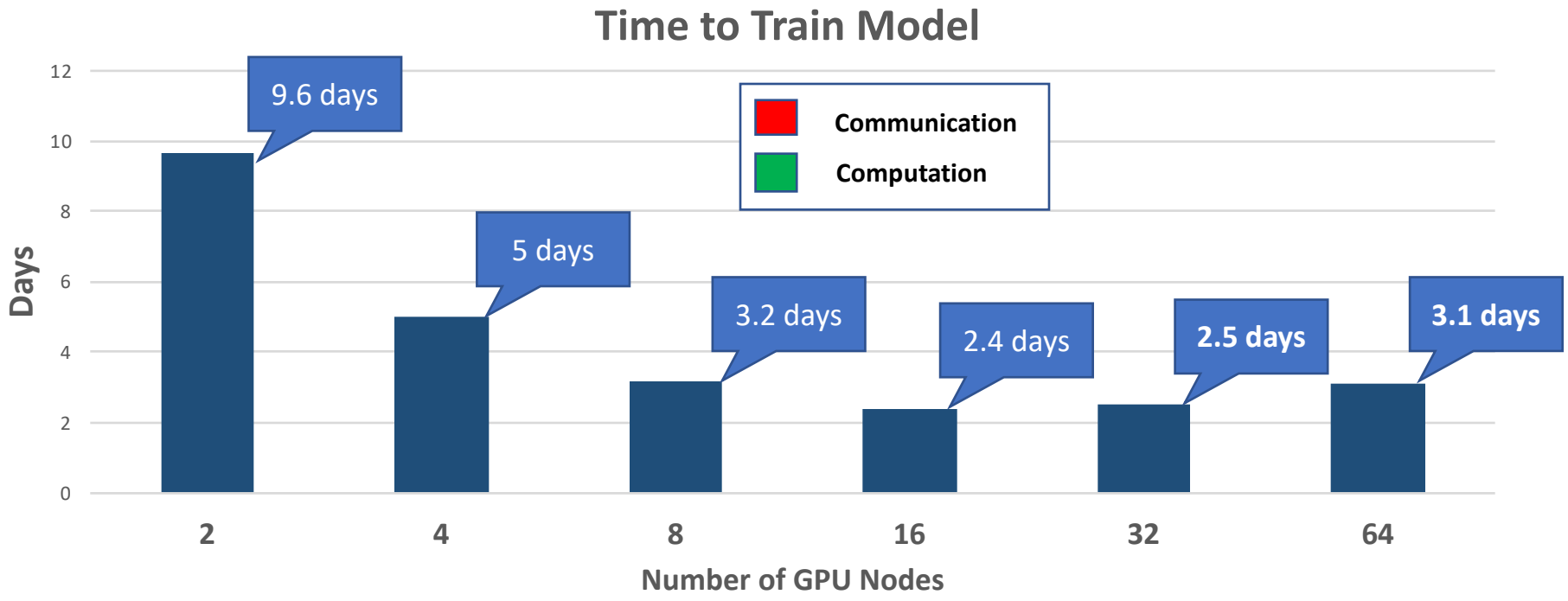
The Scalability Problem

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***)



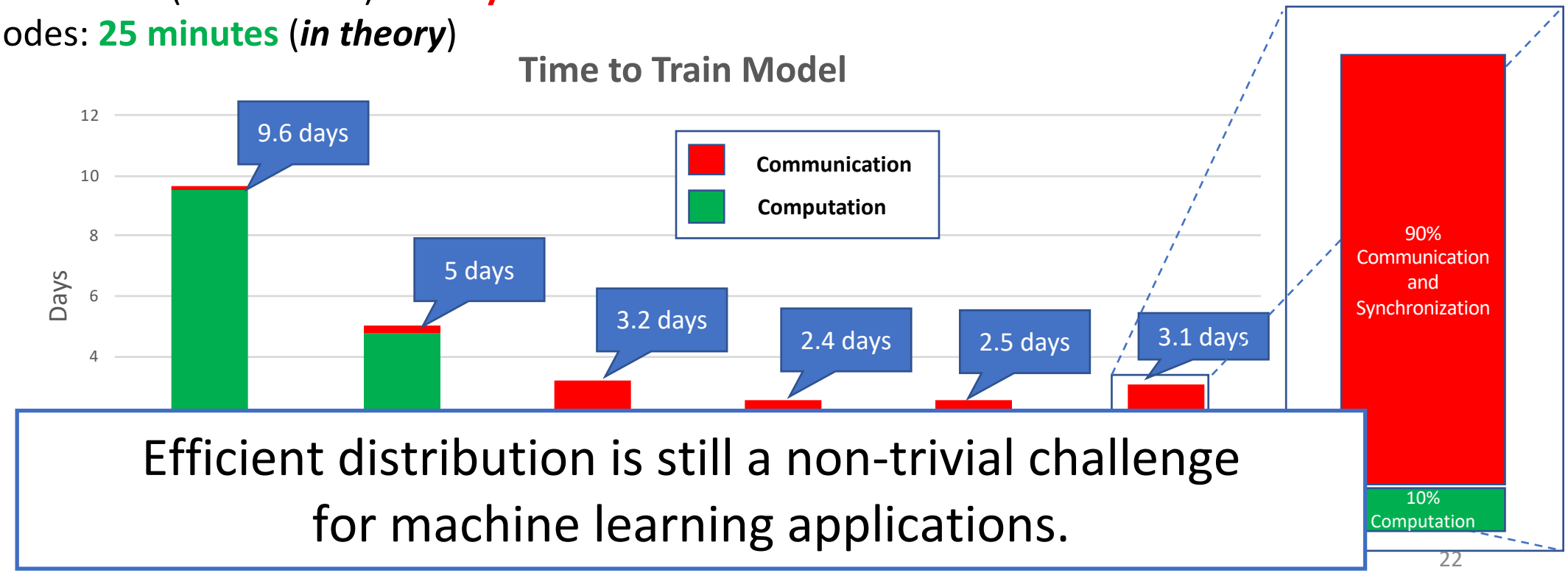
The Scalability Problem

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***)

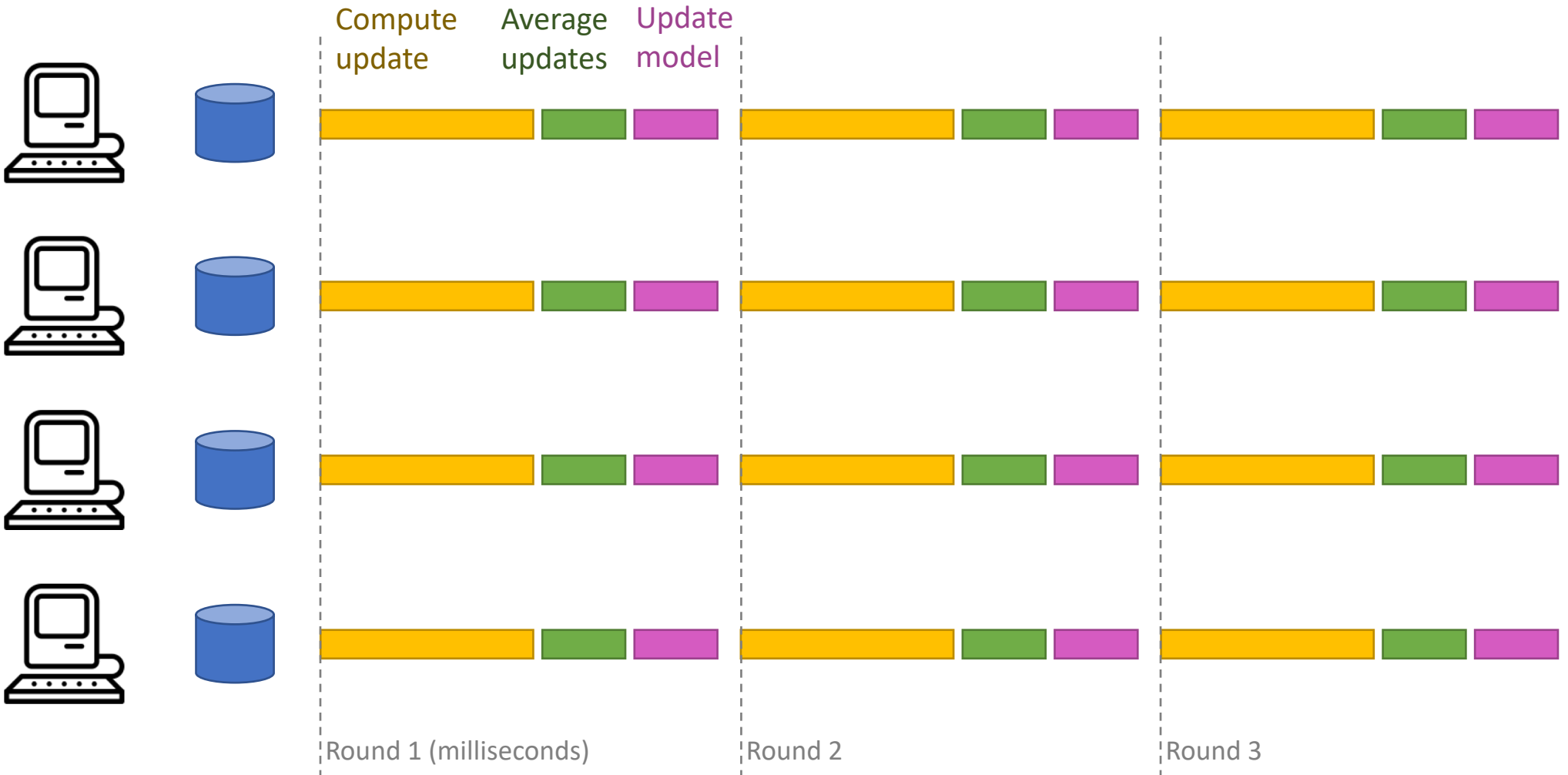


Efficient distribution is still a non-trivial challenge for machine learning applications.

The Algorithm: Parallel Stochastic Gradient Descent

Synchronous Message-Passing System

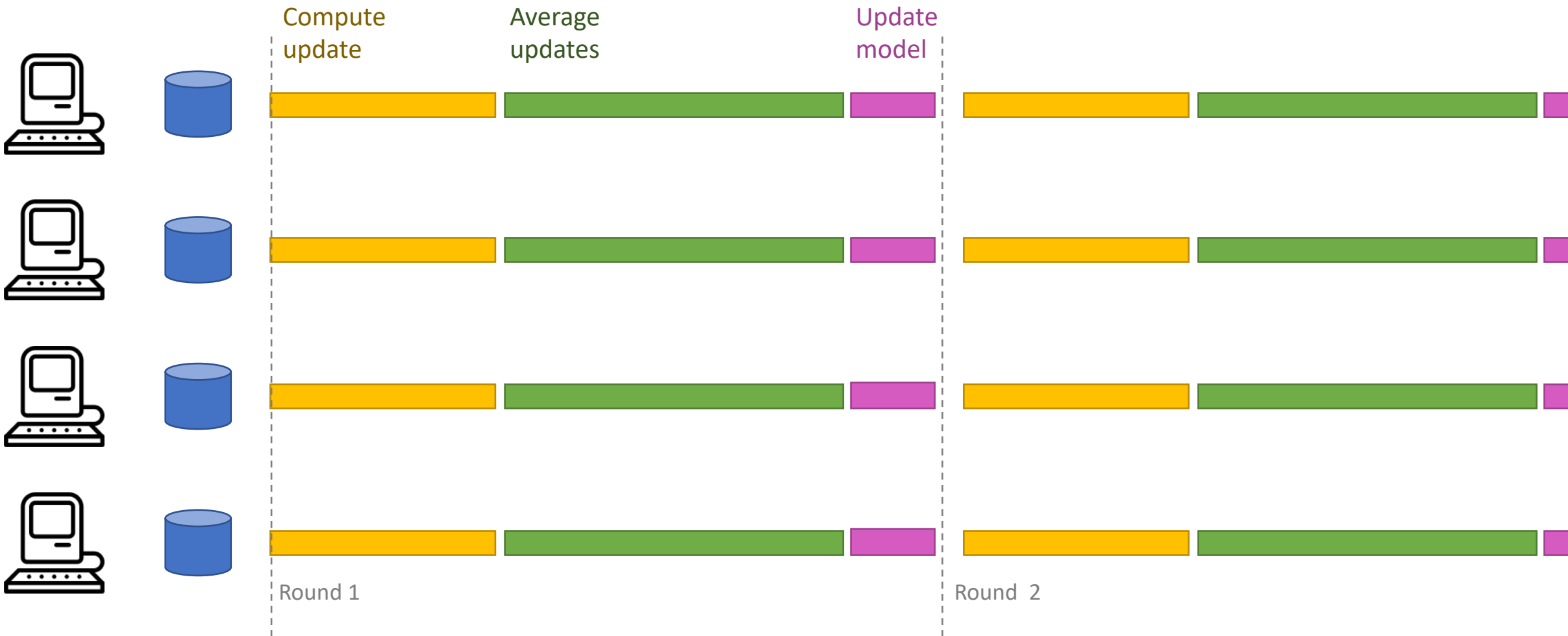
- **n** nodes, fully-connected communication topology



Parallel SGD (**large** models)

Synchronous Message-Passing System

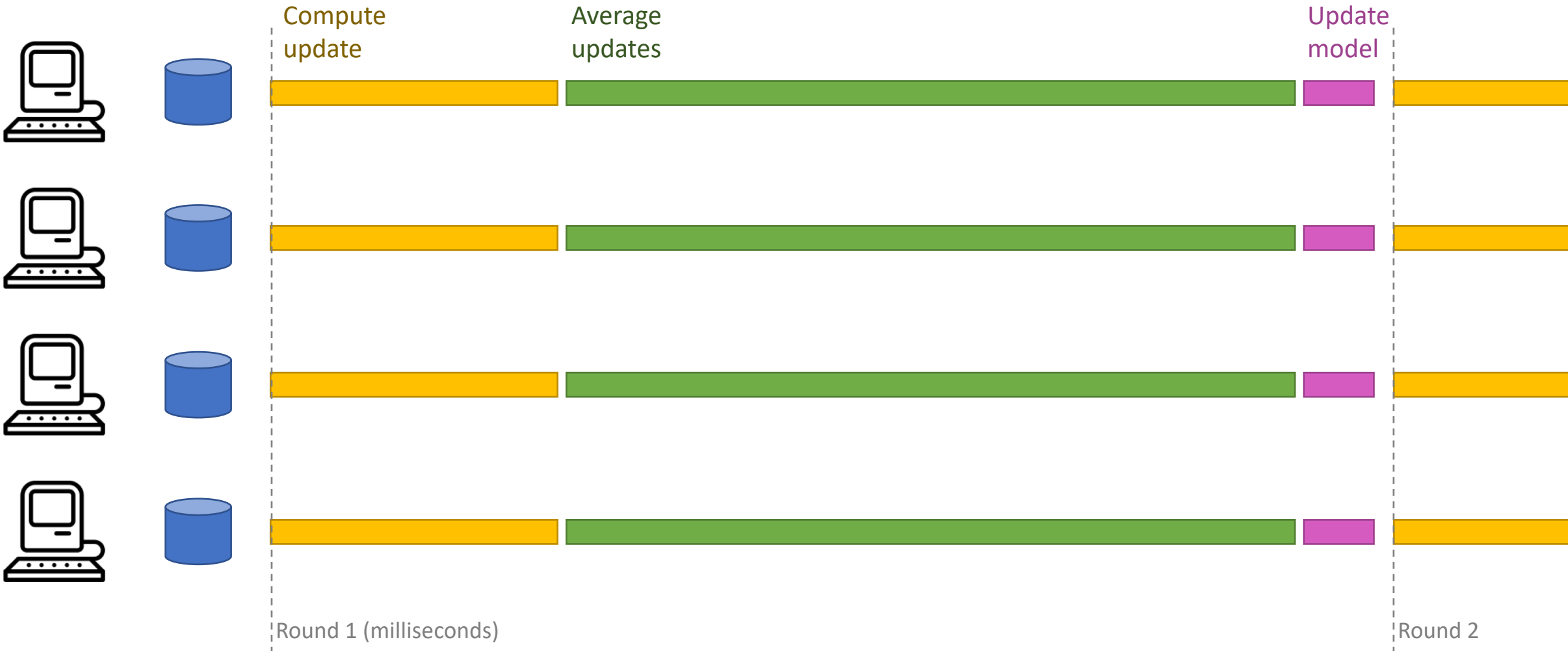
- **n** nodes, fully-connected communication topology



Parallel SGD (*large* models and high node counts)

Synchronous Message-Passing System

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



The Plan for Today

1. A Quick Introduction to Population Protocols

2. The Scaling Challenge in Distributed Machine Learning

3. Large-Scale Optimization via Population Protocols

The General Setting

Given:

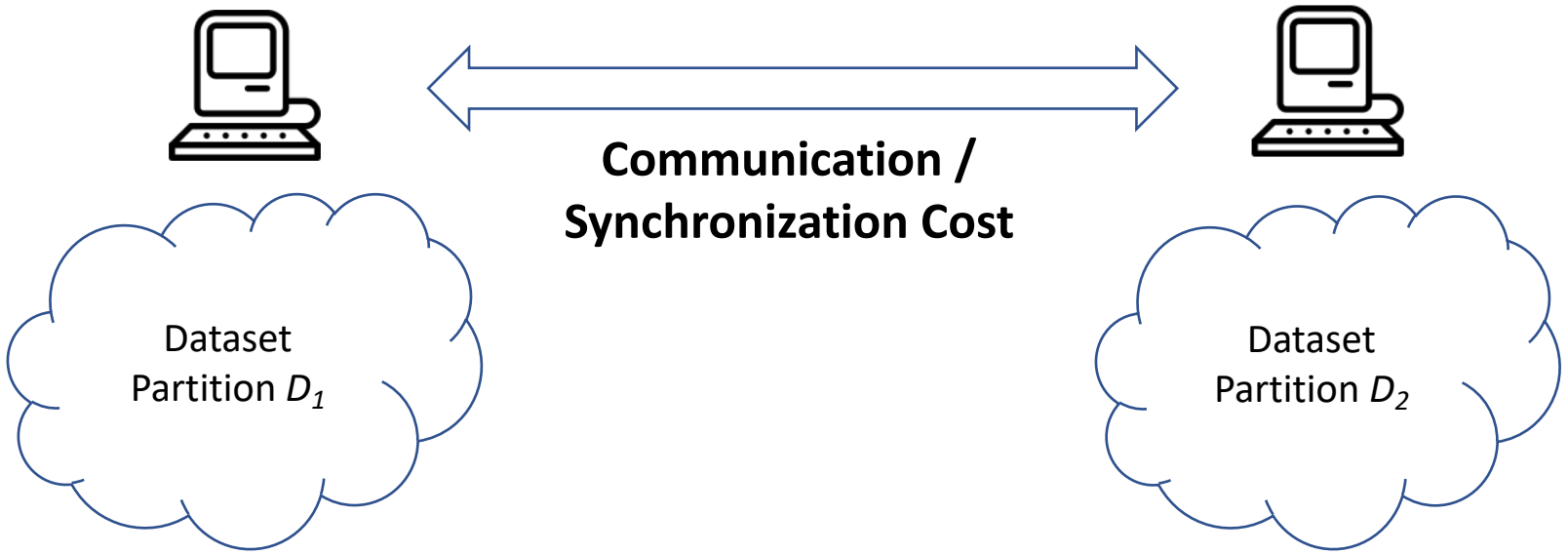
- **n** nodes, message-passing, fully-connected topology
- Dataset ***D***: node p_i is assigned dataset partition $\boldsymbol{D_i}$
- Loss function **Loss(x, e)** = how "good" is the prediction of model **x** on example **e**

Wanted:

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

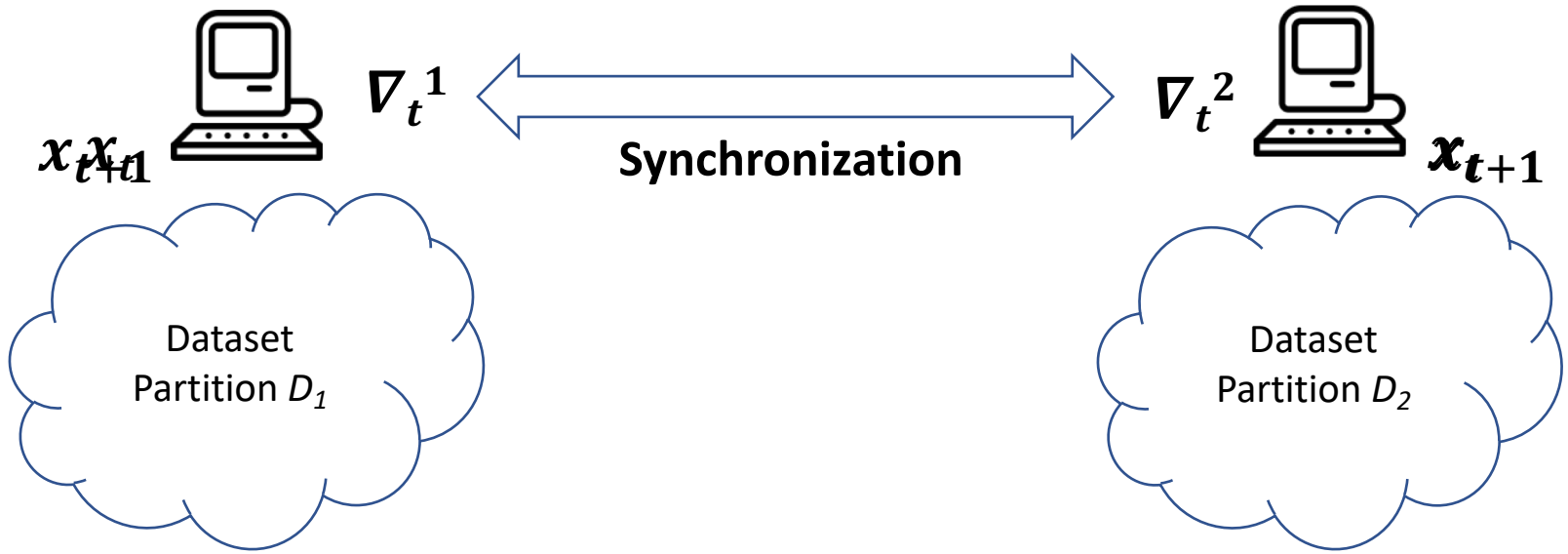
$f_1(x) = \mathbf{E}_{\text{e from D1}}$ [Loss(x, e)]

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



The Classic Algorithm: 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 $\boldsymbol{\nabla_t{}^i}$ = the gradient of x_t at e_i w.r.t. the Loss
  - Nodes **average** their updates: $\boldsymbol{\nabla_t = (\nabla_t{}^1 + \nabla_t{}^2)/2}$
  - **Update model:** $\boldsymbol{x_{t+1} = x_t - \eta_t \nabla_t}$, where $\boldsymbol{\eta_t}$ is the learning rate.



Example: Distributed Mean Estimation

- Given distribution ***D***, find a parameter $\boldsymbol{x} \in \mathbb{R}^d$ which minimizes
$$\mathbf{E}_{e \text{ in } D}\left[\left|\left|x - e\right|\right|^2\right].$$
- 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 $\boldsymbol{\nabla_t^i = e_i - xt}$
  - 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!

Can we leverage populations to optimize faster?

PopSGD: Distributed Optimization in a Population

The issue: in a population, the "model" x_t can no longer be consistent! Interactions are pairwise:

For each interaction between $< i, j >$:

- Each node i has its own estimate ("model") $x_t{}^i$

$$\boldsymbol{x^i}_{t+1} = \boldsymbol{x^i}_t - \boldsymbol{\eta_t \nabla_t{}^i}$$

$$\boldsymbol{x^j}_{t+1} = \boldsymbol{x^j}_t - \boldsymbol{\eta_t \nabla_t{}^j}$$

- If i and j meet, they take an ***update step*** and then ***average*** their estimates:

$$\boldsymbol{x^i}_{t+1} = \boldsymbol{x^j}_{t+1} = \boldsymbol{(x^i_t + x^j_t)/2 - \eta_t(\nabla_t{}^i + \nabla_t{}^j)/2}$$

Does this still converge? Does it yield any *speedup*?

Note: similar algorithms considered for mean estimation via gossip [BGPS06].

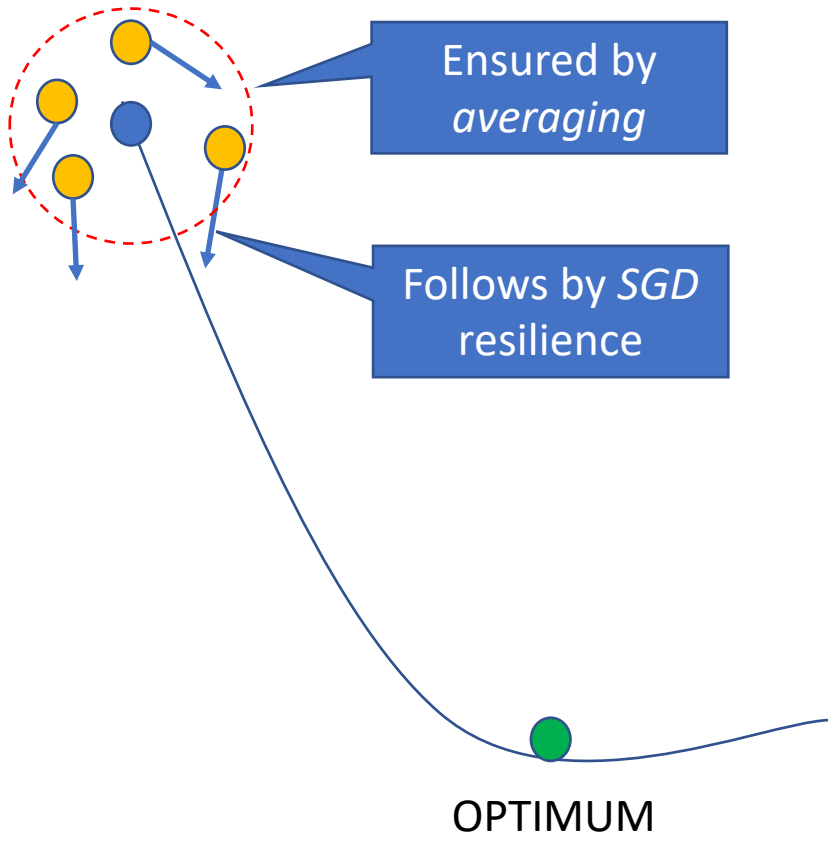
A High-Level View of the Analysis

Theorem [informal]: For **convex** objective functions f, given **large enough #iterations T**, the PopSGD algorithm **converges n times** faster than SGD.

- We keep track of the *mean of the models:*

$$\boldsymbol{\mu_t} = \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} \boldsymbol{x^i_t}$$

- **Step 1**: Show that *the models themselves stay concentrated around* $\boldsymbol{\mu_t}$
  - A **multi-dimensional** load-balancing process!
- **Step 2**: SGD is resilient to gradients being taken at noisy versions of $\boldsymbol{\mu_t}$
  - Despite noise, its convergence is still proportional to the **total** number of steps taken by the **entire** population

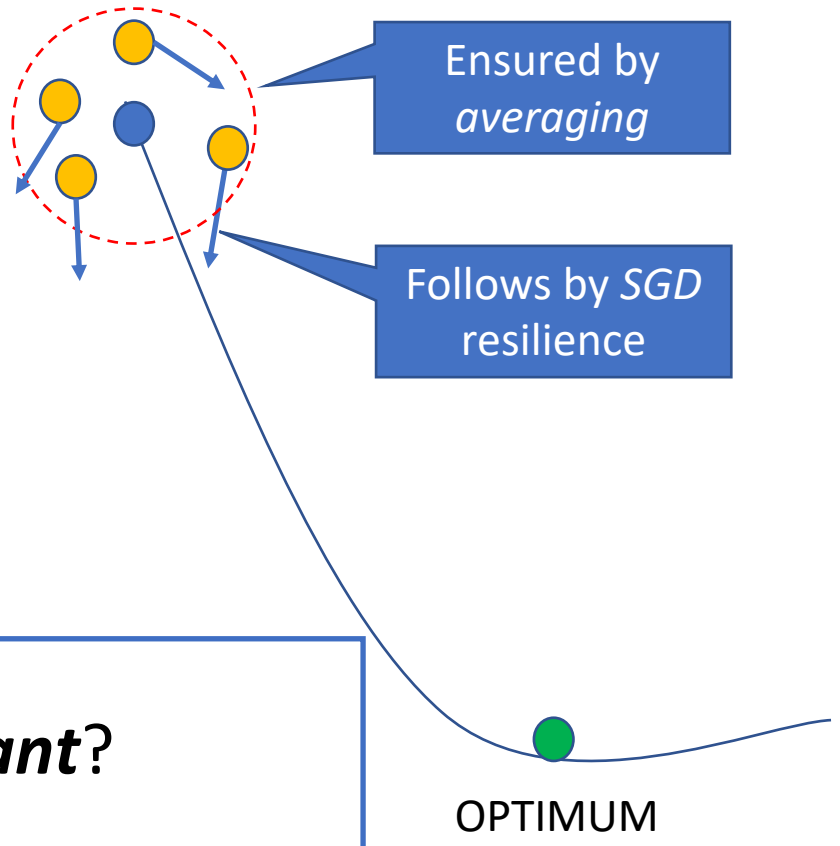


Discussion

Theorem [informal]: For **convex** objective functions f, given **large enough #iterations T**, the PopSGD algorithm **converges n times** faster than SGD.

Limitations:

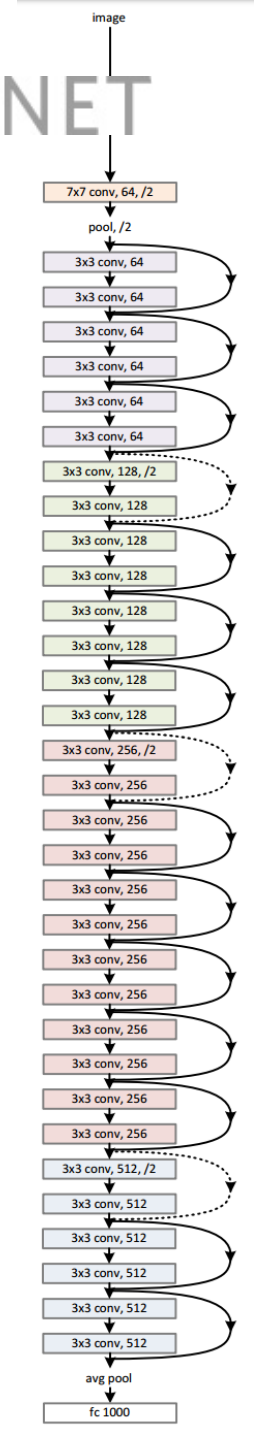
- We are ignoring space / message complexity
- The number of SGD iterations has to be large enough to hide the overhead of mixing
- Gradients assumed to be ***bounded***



Is this result ***practically relevant***?

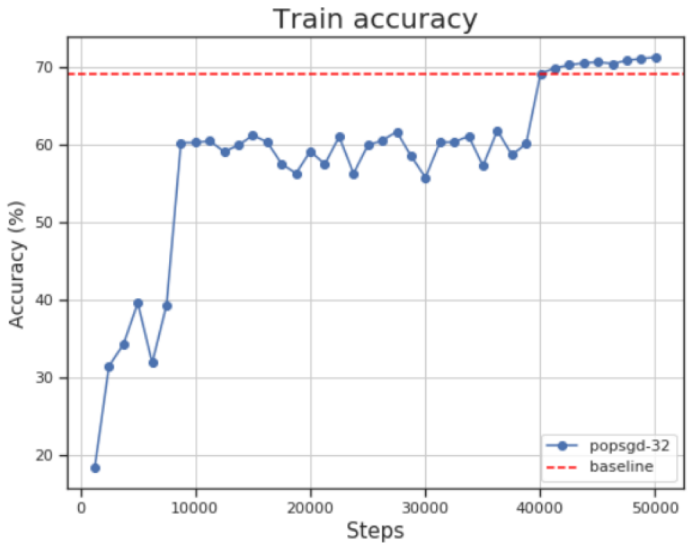
Experimental Setup

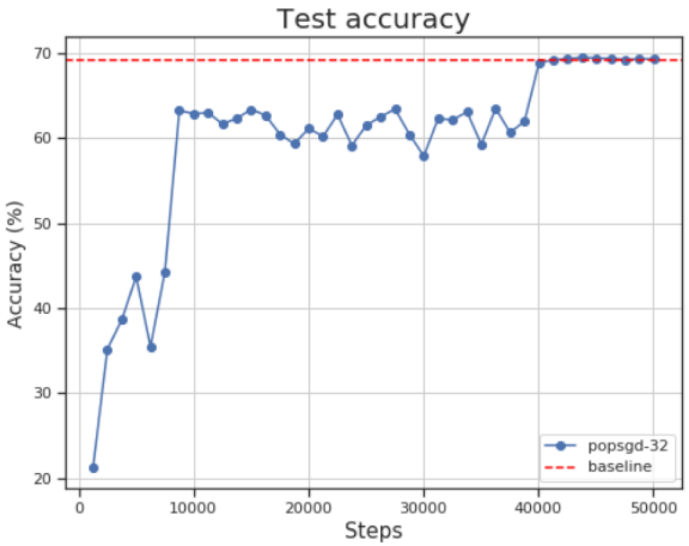
- We train large-scale residual neural networks
  - Image classification on ILSVRC 2012 and CIFAR-10 datasets
- On the Piz Daint Supercomputer
  - Each node is equipped with a state-of-the-art CPU and GPU
- Two major questions:
  - Can PopSGD recover accuracy?
  - Can PopSGD scale to lots of nodes?



Question 1: Accuracy

- Training the ResNet18 network on ImageNet
- 32 nodes, accuracy vs. steps

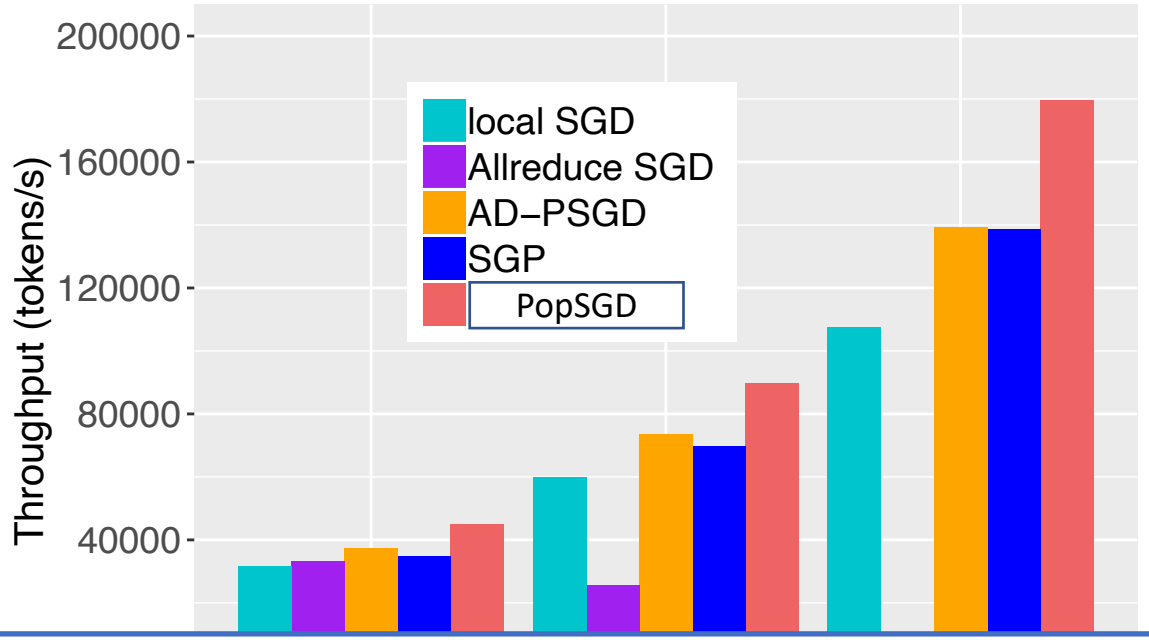




PopSGD can match or exceed the baseline accuracy.

Question 2: Speed

- Training the ResNet18 network on ImageNet
- 16-64 nodes, total samples (tokens) processed per second
- Versus state-of-the-art distribution techniques [PSGD, LocalSGD, AD-PSGD, SGP]



…and it can do it faster than previous techniques.

The Power of the “Right” Model

1. Population Protocols are a minimalistic model of large-scale distributed computing

2. They raise fundamental algorithmic questions, but can also model non-trivial practical settings

3. Example: Large-Scale Optimization by Populations

4. Lots of open questions!

Questions?