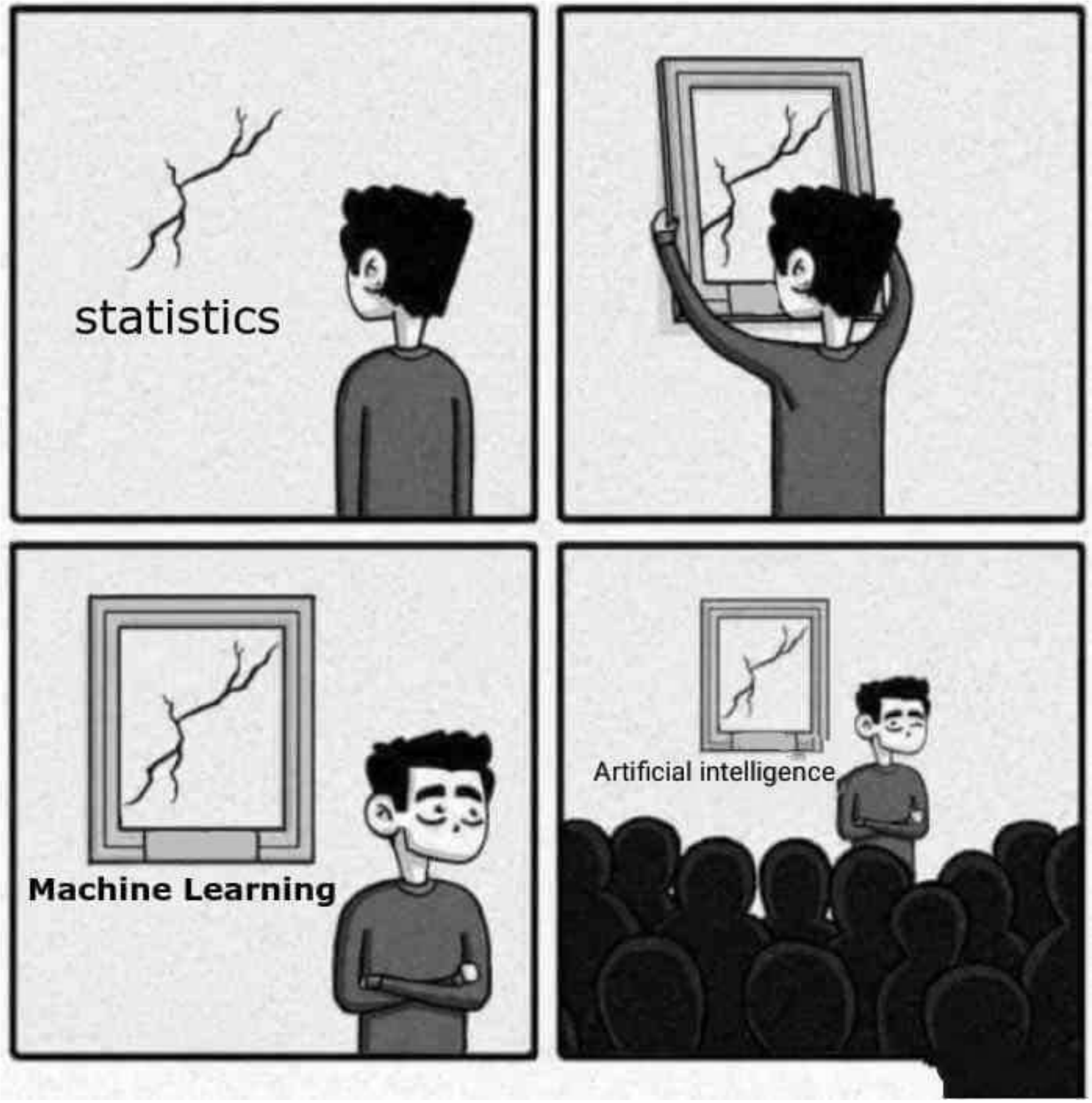
Distributed Machine Learning

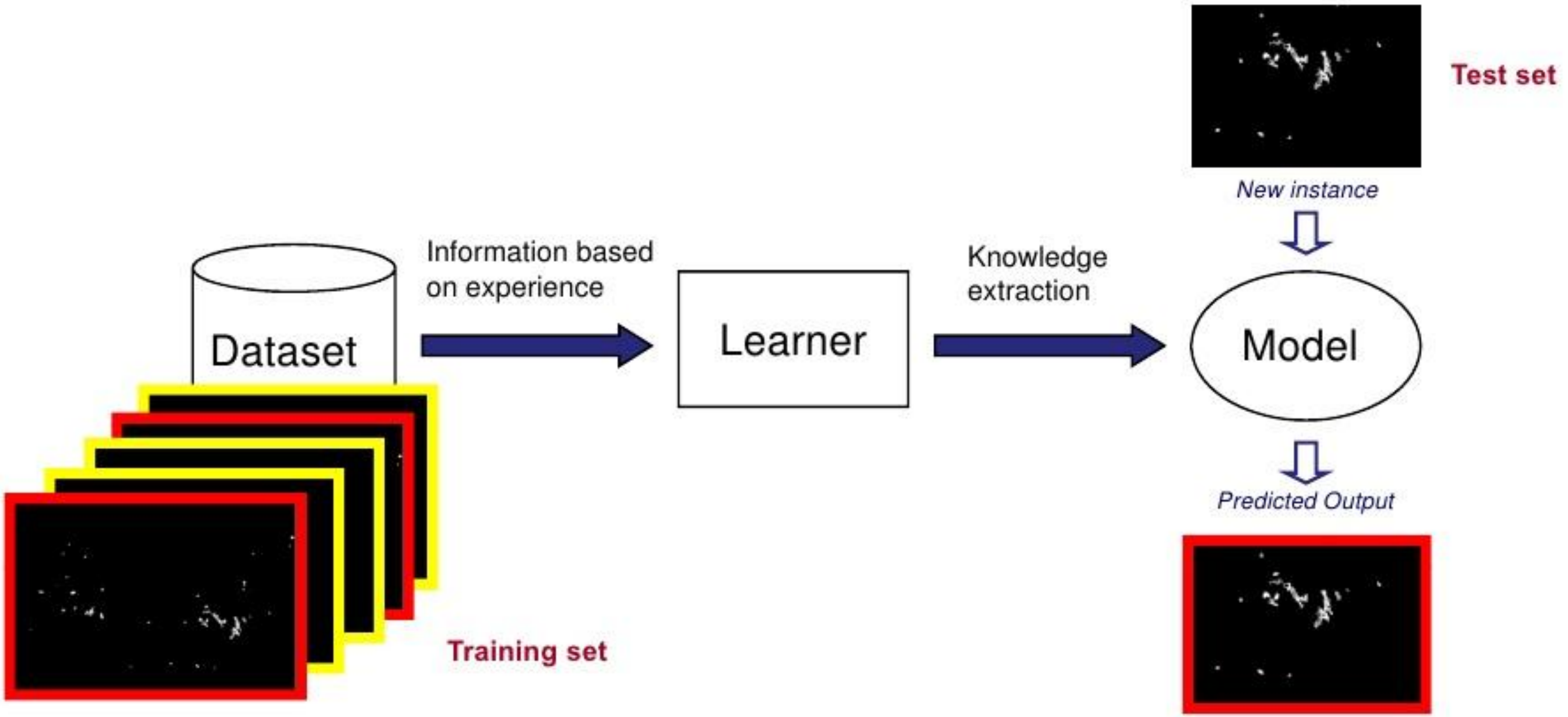
Georgios Damaskinos

2018

Machine Learning ?



Machine Learning "in a nutshell"



Machine Learning algorithm

Cost Functions

Root Mean Squared Error (RMSE)

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Root Mean Squared Log Error (RMSLE)

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(\log(p_i + 1) - \log(a_i + 1))^2}$$

prediction

actual

Machine Learning algorithm

$$h_\theta(x) = \sum_{j=0}^{n} \theta_j x_j$$

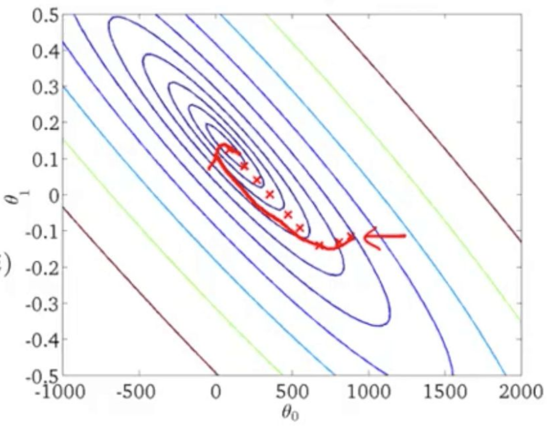
$$J_{train}(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

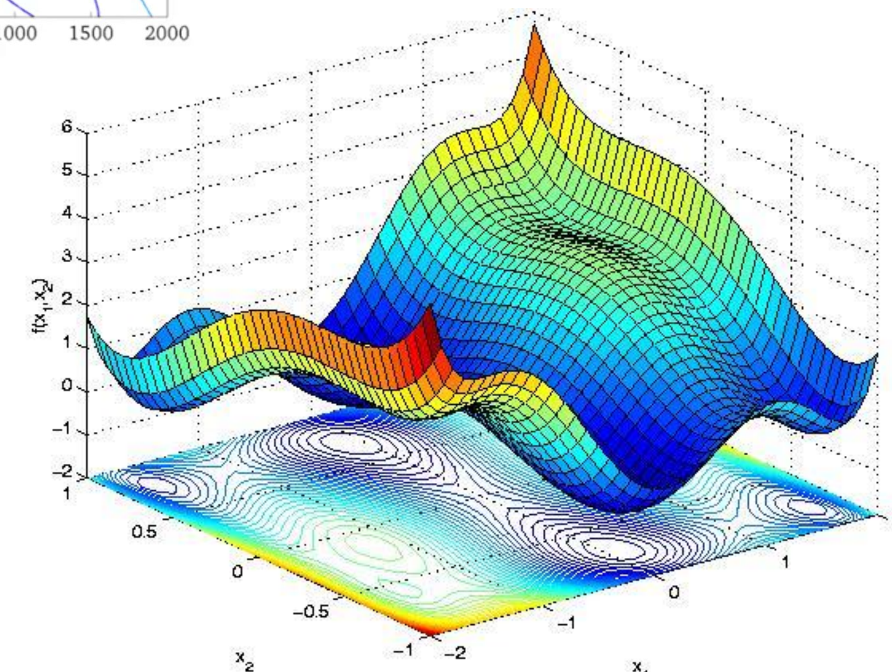
Repeat {

$$\theta_j := \theta_j - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

(for every $j = 0, \ldots, n$)

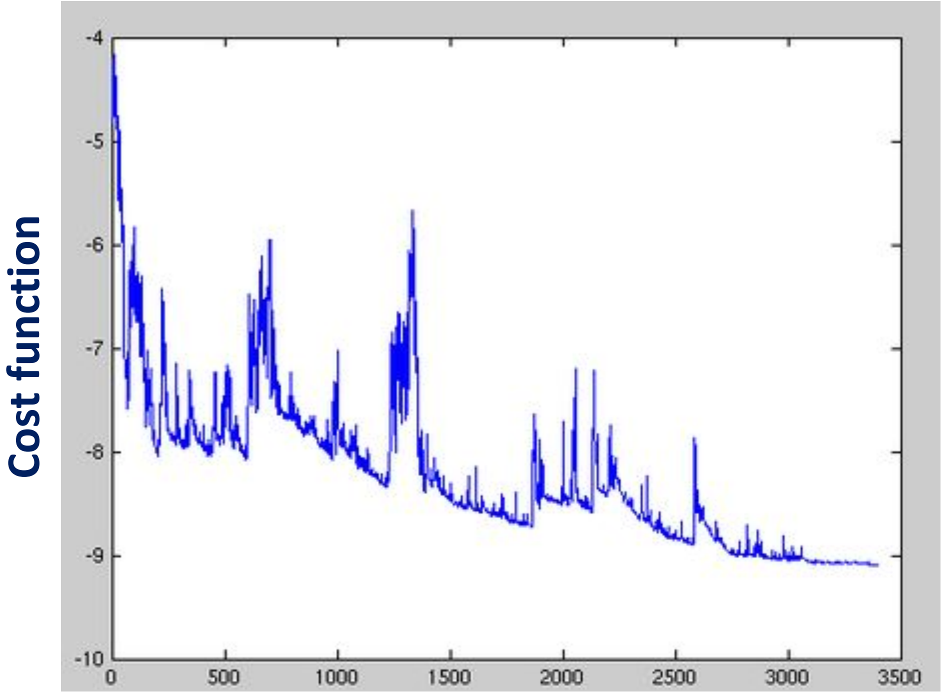
}





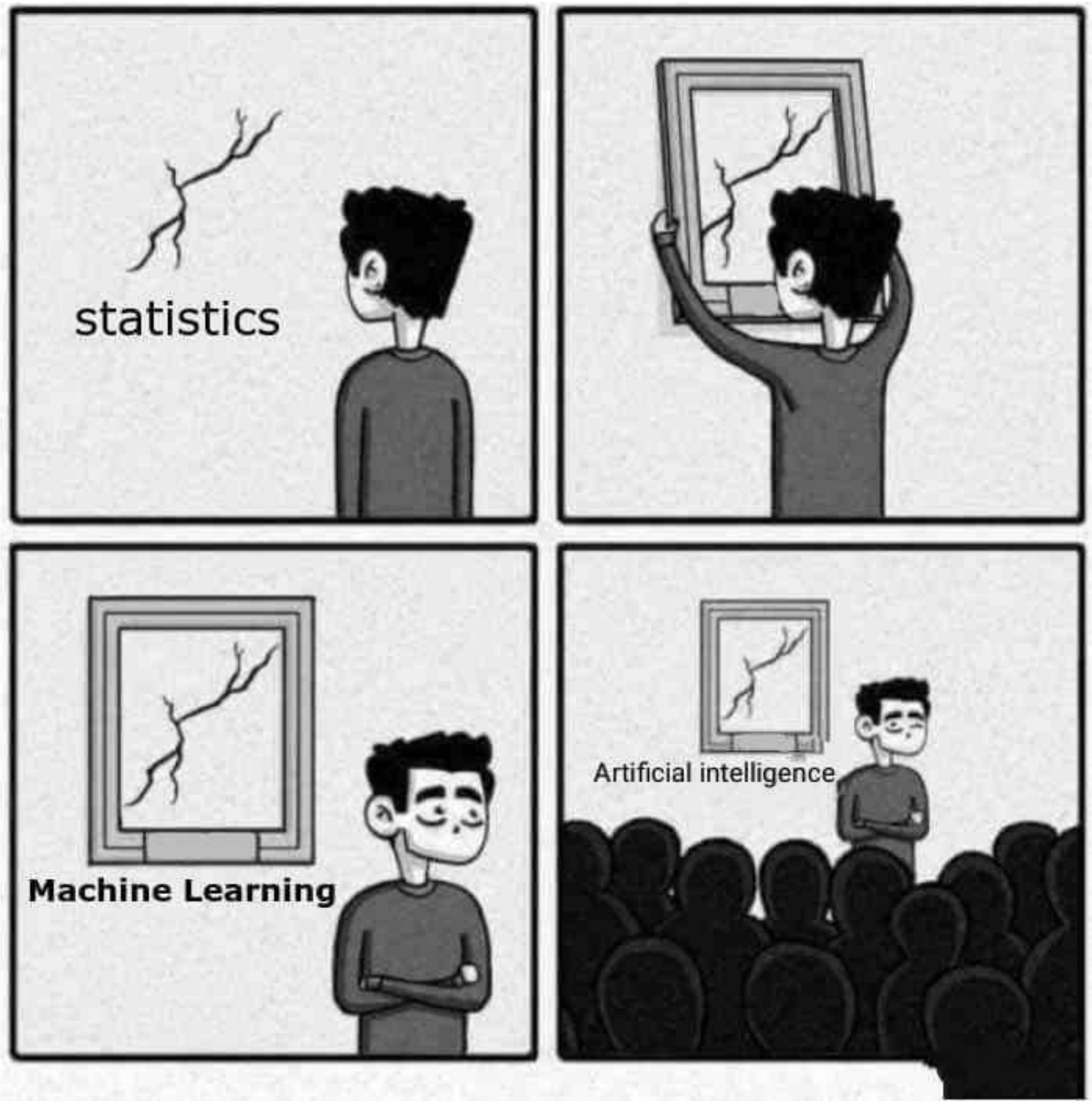
Safety?

Safety?

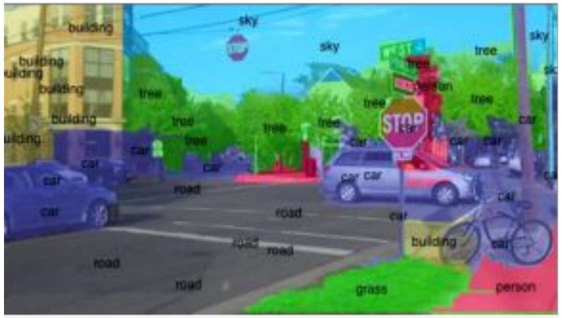


Convergence

Machine Learning ?



Think big!



"It's not who has the best algorithm that wins. It's who has the most data."

Andrew Ng

Think big!

Example: Image Classification

<u>Data</u>:

ImageNet: 1.3 Million training images (224 x 224 x 3)

<u>Model</u>:

ResNet-152: 60.2 Million parameters (model size)

Training time (single node):

TensorFlow: **19 days!!**

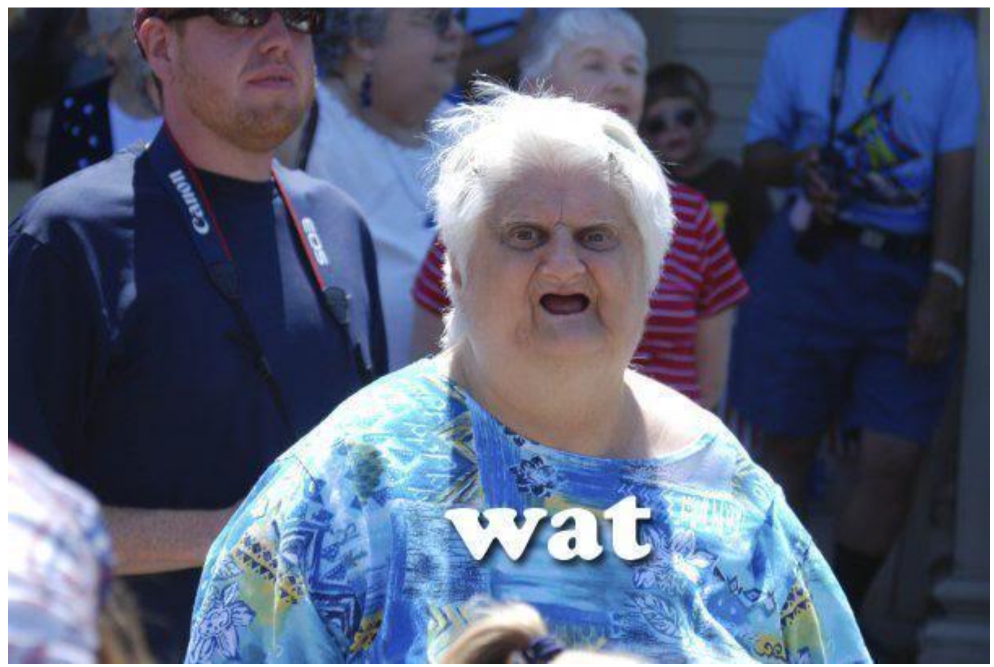
Think big!

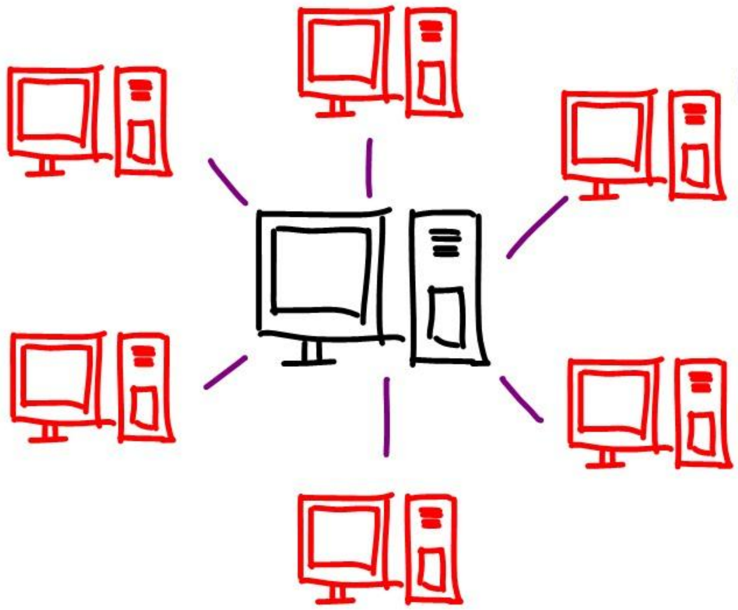
Example: Image Classification

Data:
    ImageNet: 1.3 Million training images (224 x 224 x 3)

Model:
    ResNet-152: 60.2 Million parameters (model size)

Training time (single node):
    TensorFlow: **19 days!!**





Distributed Computing

Performance?

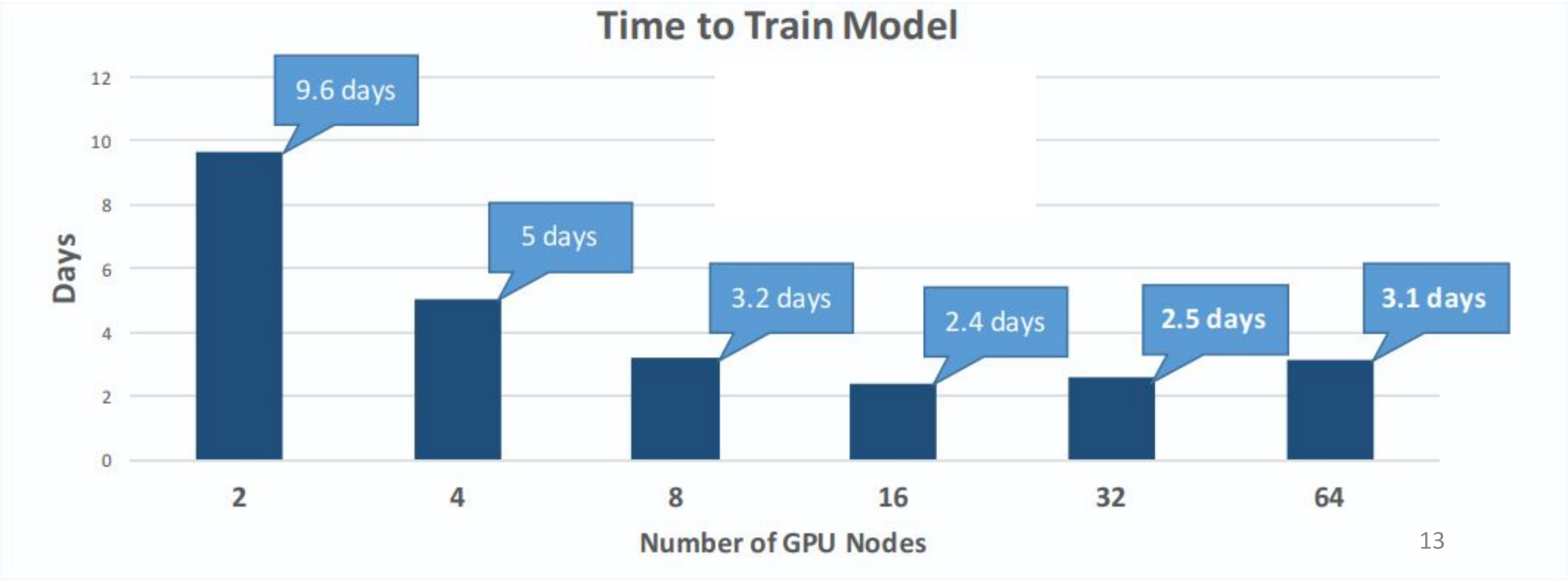
Training time (single node):

TensorFlow: **19 days**

Training time (distributed):

1024 Nodes (theoretical): *25 minutes*

CSCS (3rd top supercomputer, 4500+ GPUs, state-of-the-art interconnect:



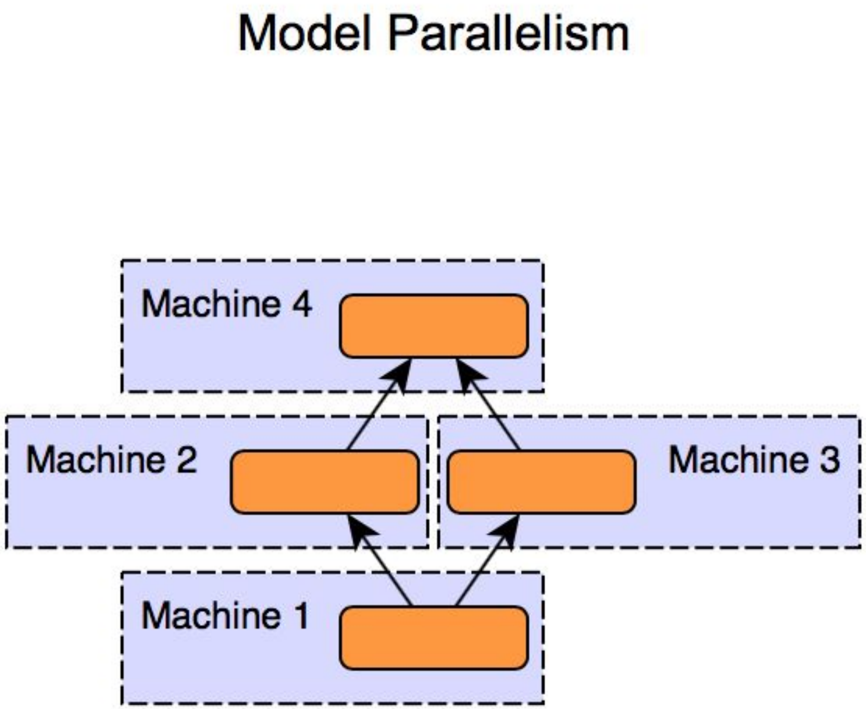
Performance?

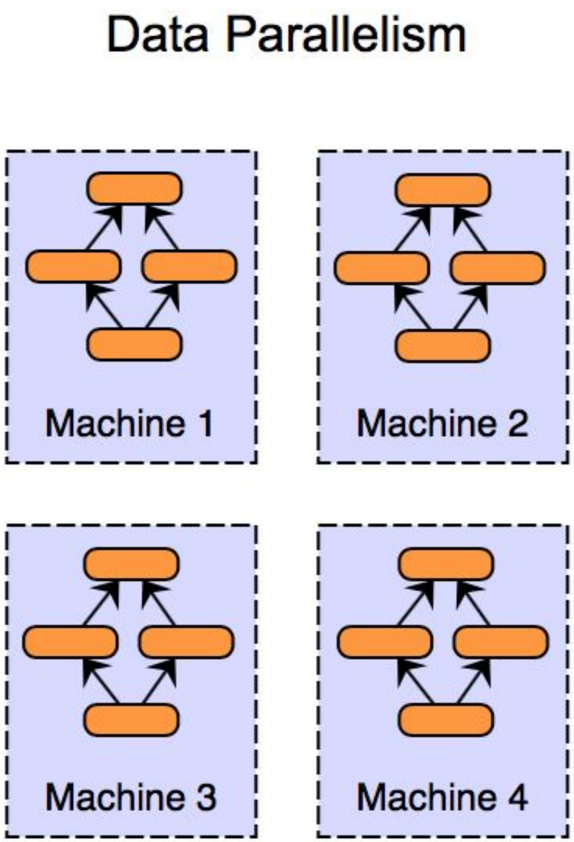
State-of-the-art (ResNet-50): **1 hour** [GP+17]

- Batch size = 8192
- 256 GPUs

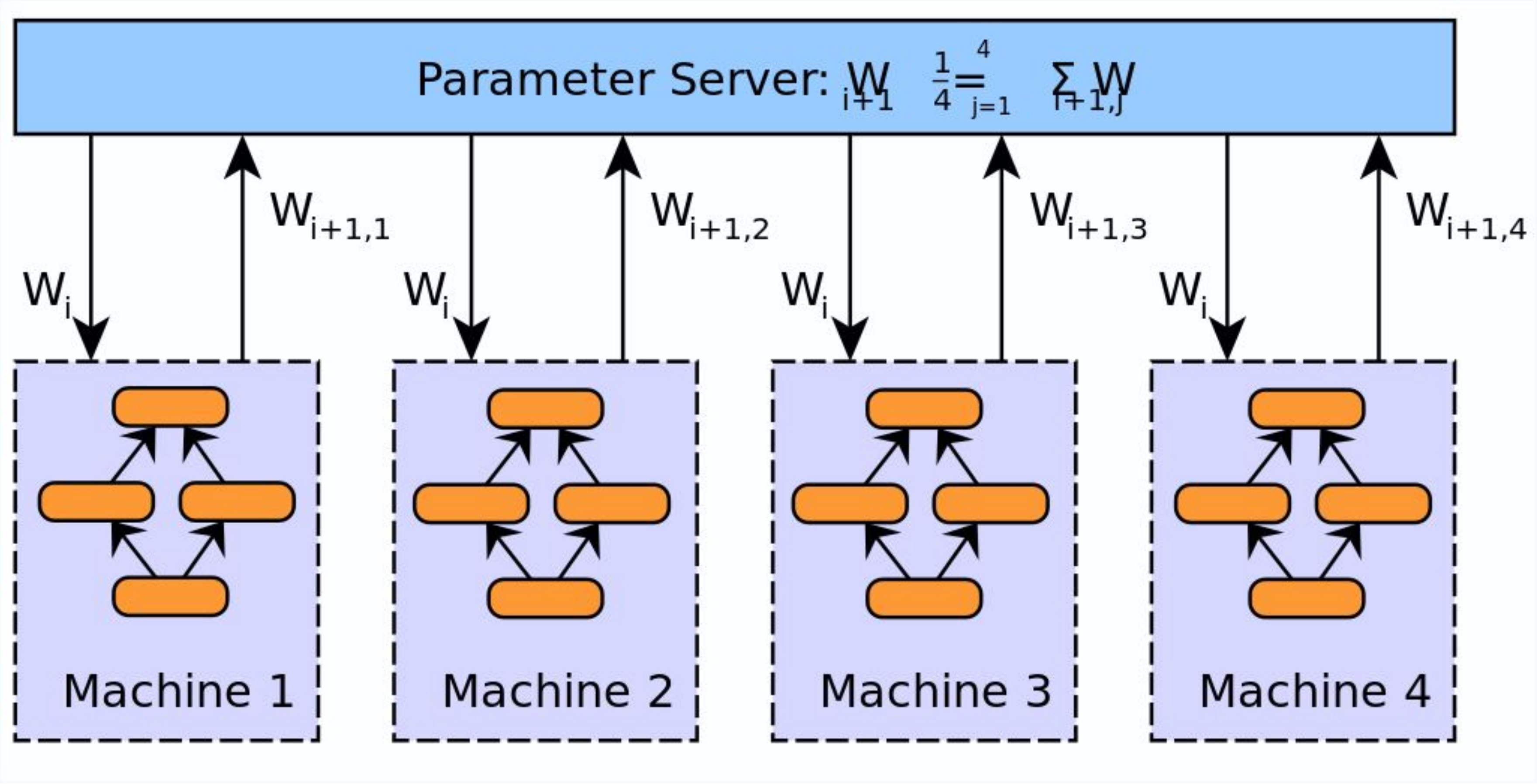
[GP+17] Goyal, Priya, et al. "Accurate, large minibatch SGD: training imagenet in 1 hour." 2017

Distributed … how?

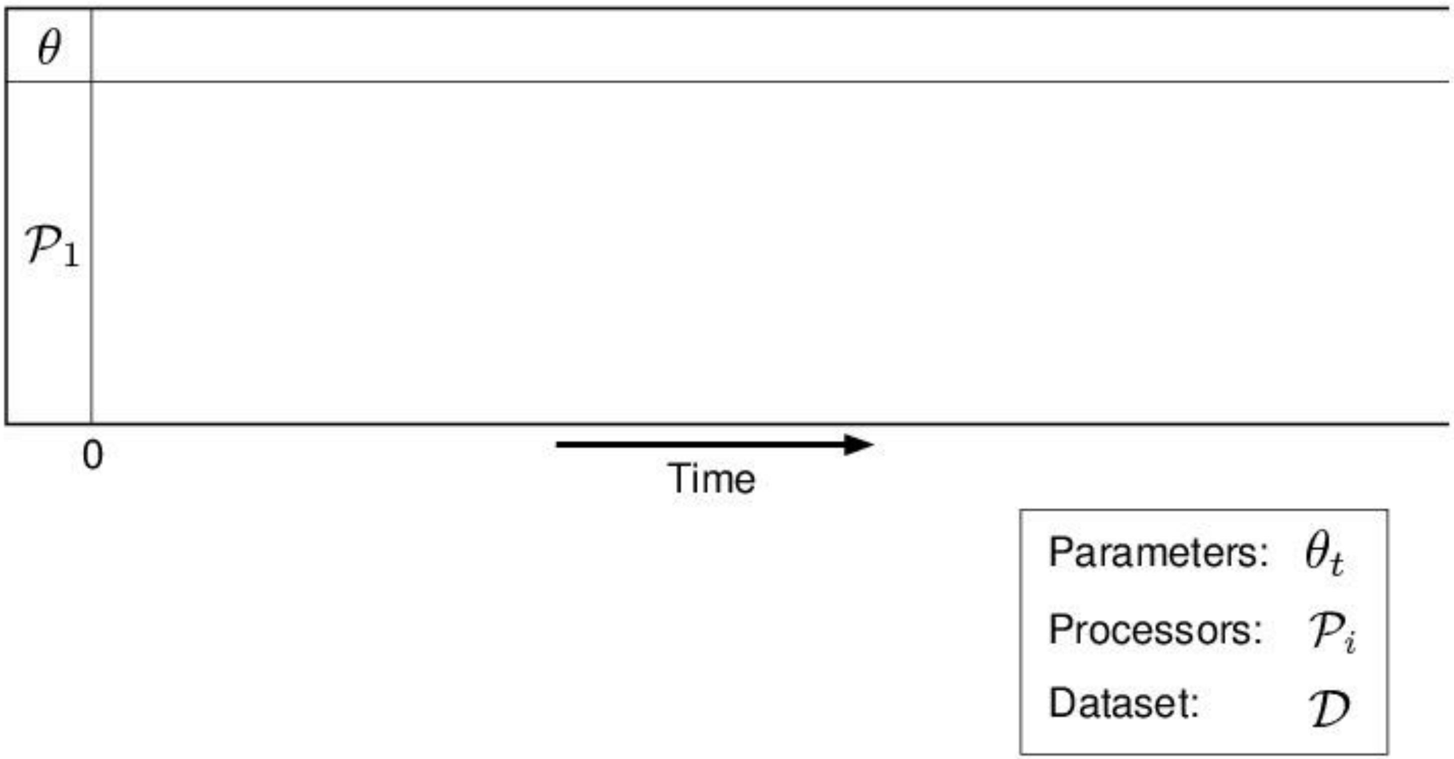




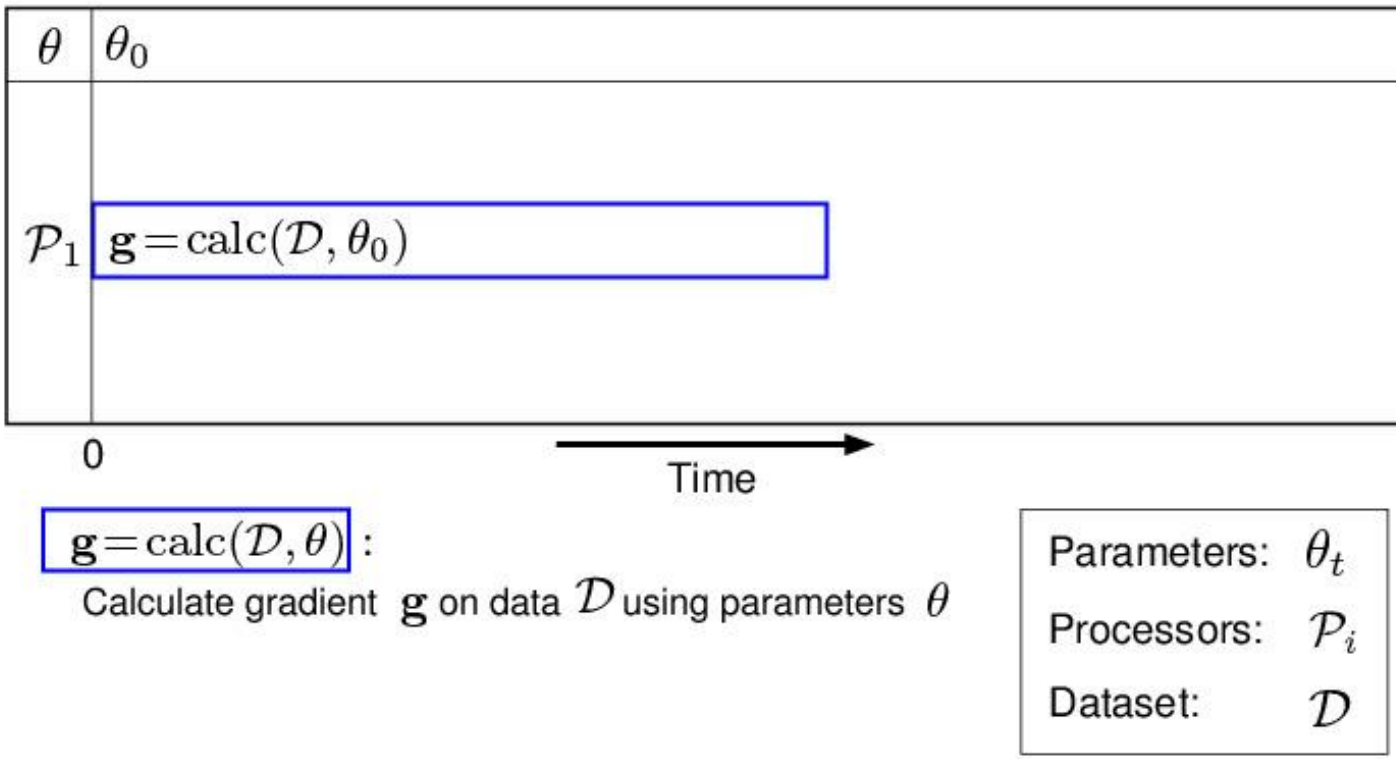
Data Parallelism



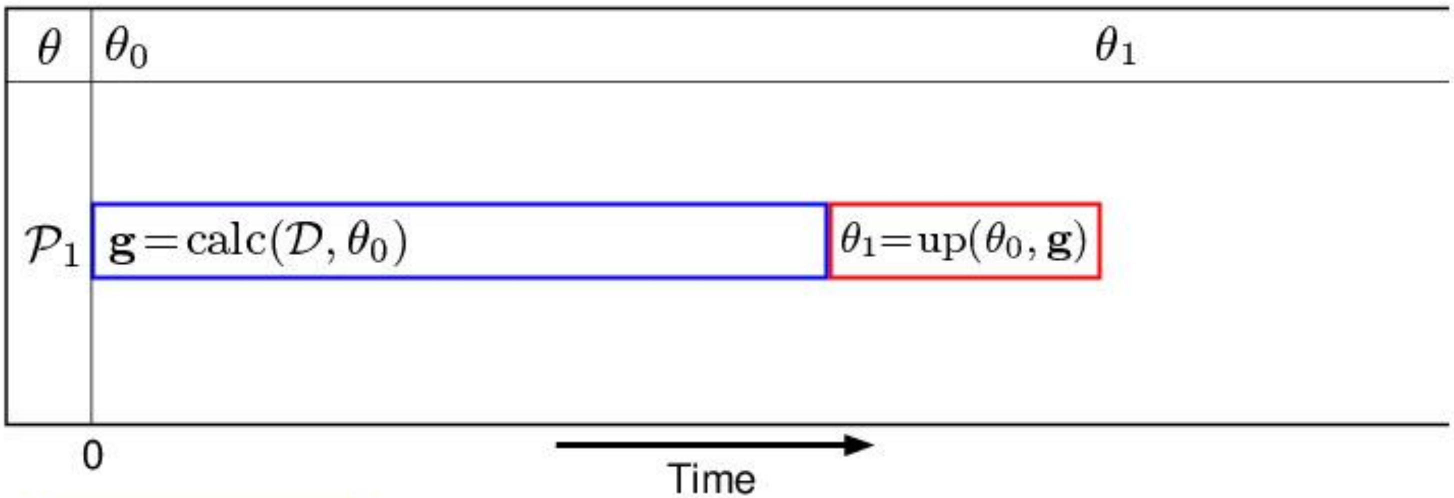
Batch Learning



Batch Learning



Batch Learning



$\boxed{\mathbf{g} = \mathrm{calc}(\mathcal{D}, \theta)}$:

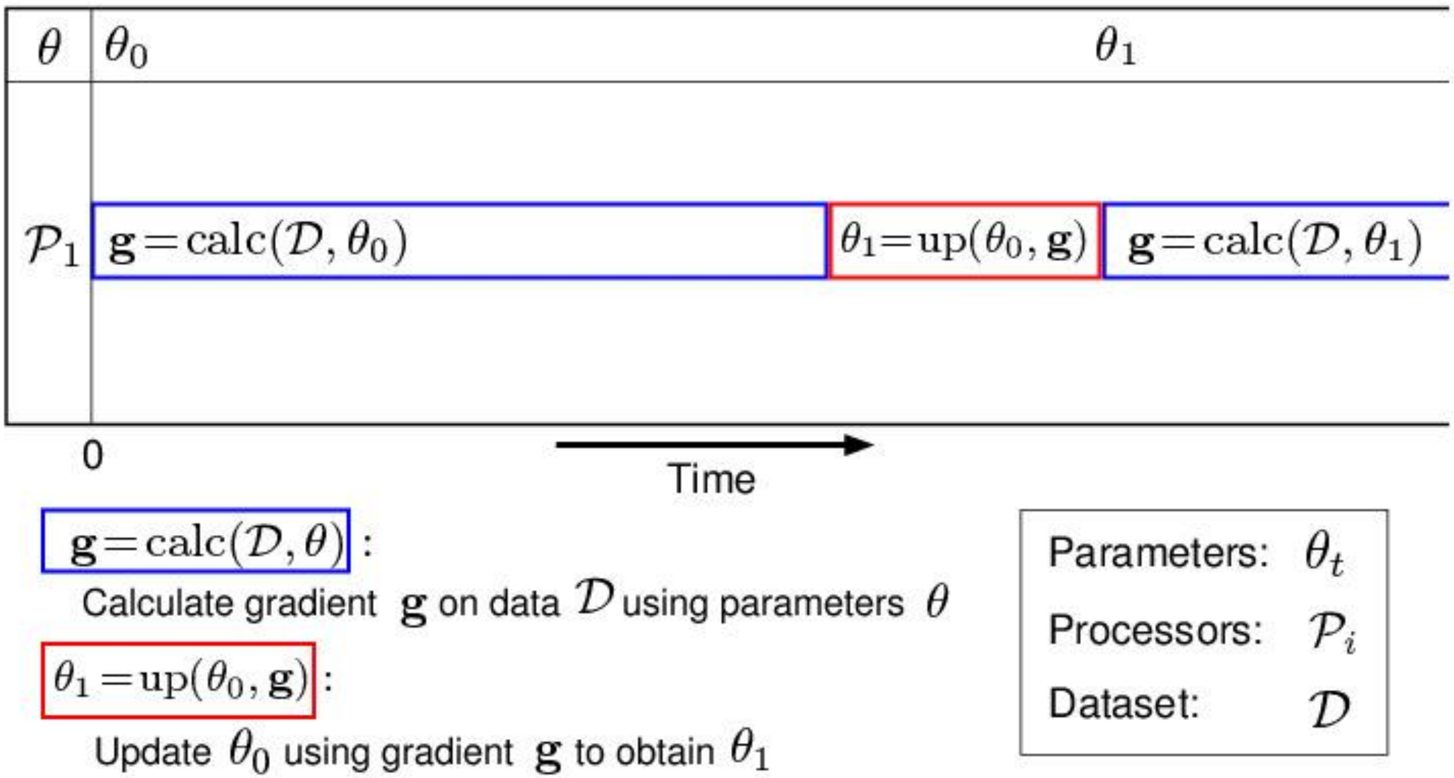
Calculate gradient $\mathbf{g}$ on data $\mathcal{D}$ using parameters θ

$\boxed{\theta_1 = \mathrm{up}(\theta_0, \mathbf{g})}$:

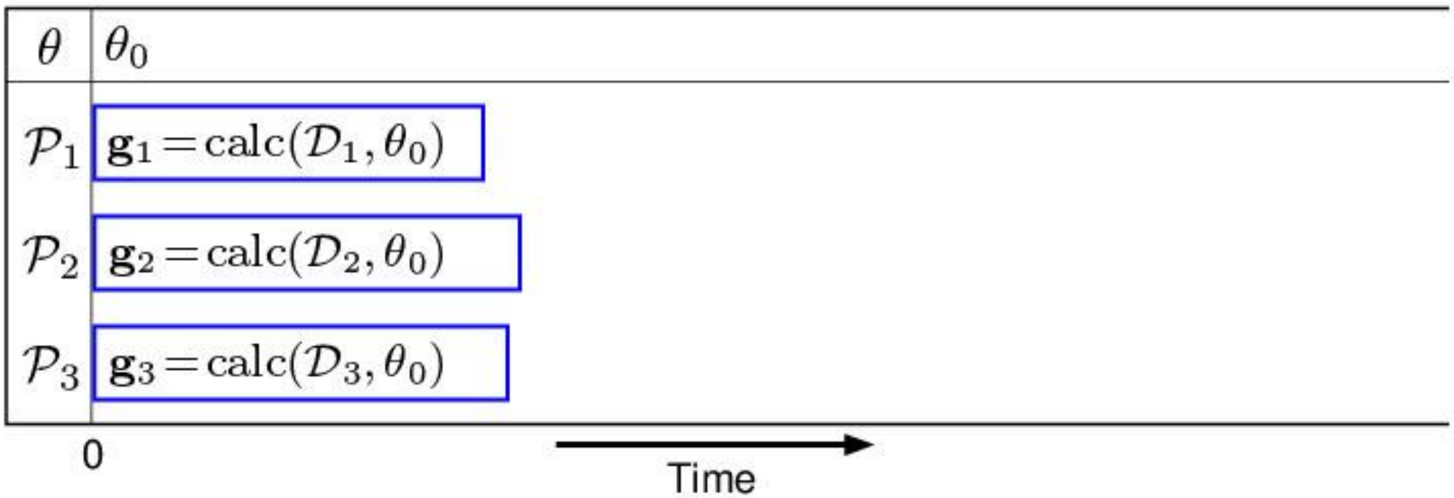
Update θ_0 using gradient $\mathbf{g}$ to obtain θ_1

Parameters:	θ_t
Processors:	$\mathcal{P}_i$
Dataset:	$\mathcal{D}$

Batch Learning



Parallel Batch Learning



- Partition Data
- Parallel Compute on Partitions

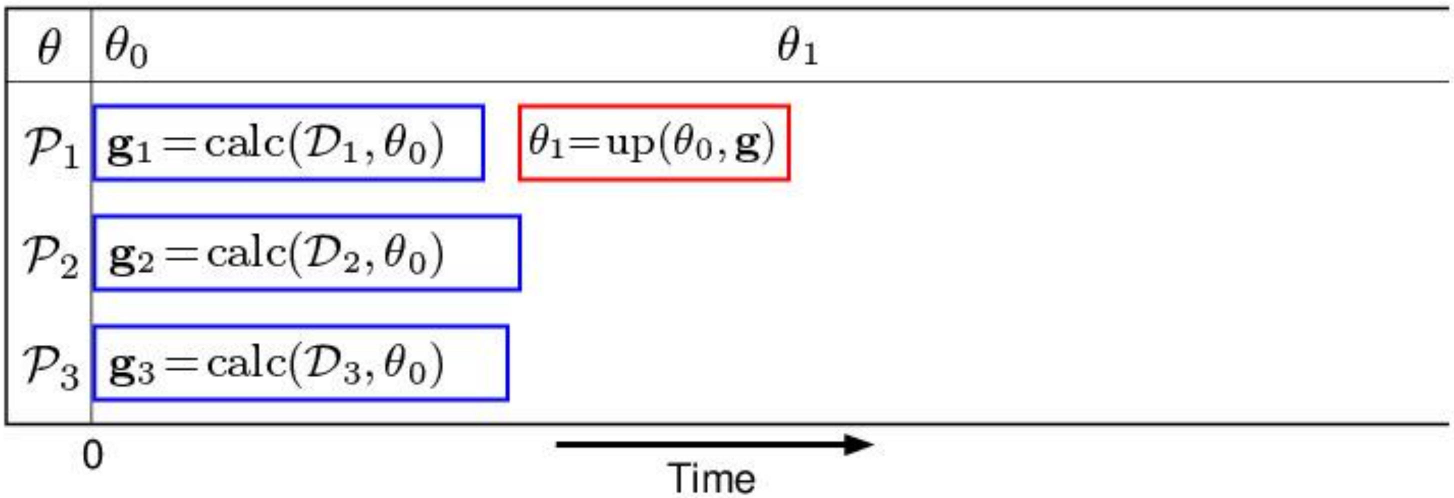
Parameters: θ_t

Processors: $\mathcal{P}_i$

Dataset: $\mathcal{D} = \mathcal{D}_1 \cup \mathcal{D}_2 \cup \mathcal{D}_3$

Gradient: $\mathbf{g} = \mathbf{g}_1 + \mathbf{g}_2 + \mathbf{g}_3$

Parallel Batch Learning



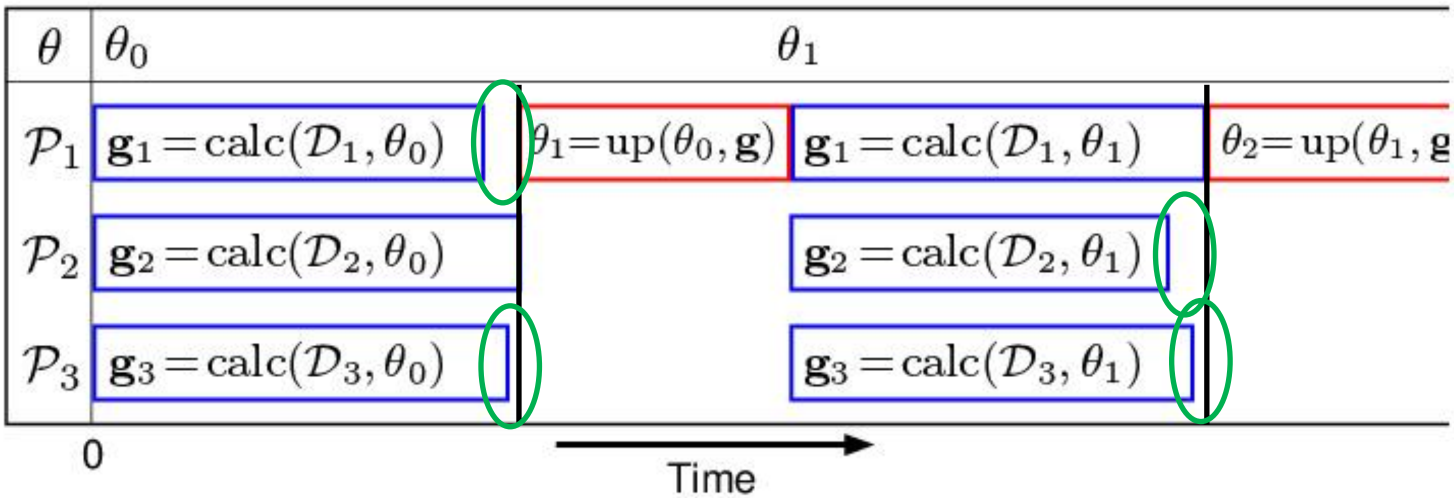
Parameters: θ_t

Processors: $\mathcal{P}_i$

Dataset: $\mathcal{D} = \mathcal{D}_1 \cup \mathcal{D}_2 \cup \mathcal{D}_3$

Gradient: $\mathbf{g} = \mathbf{g}_1 + \mathbf{g}_2 + \mathbf{g}_3$

Parallel Batch Learning



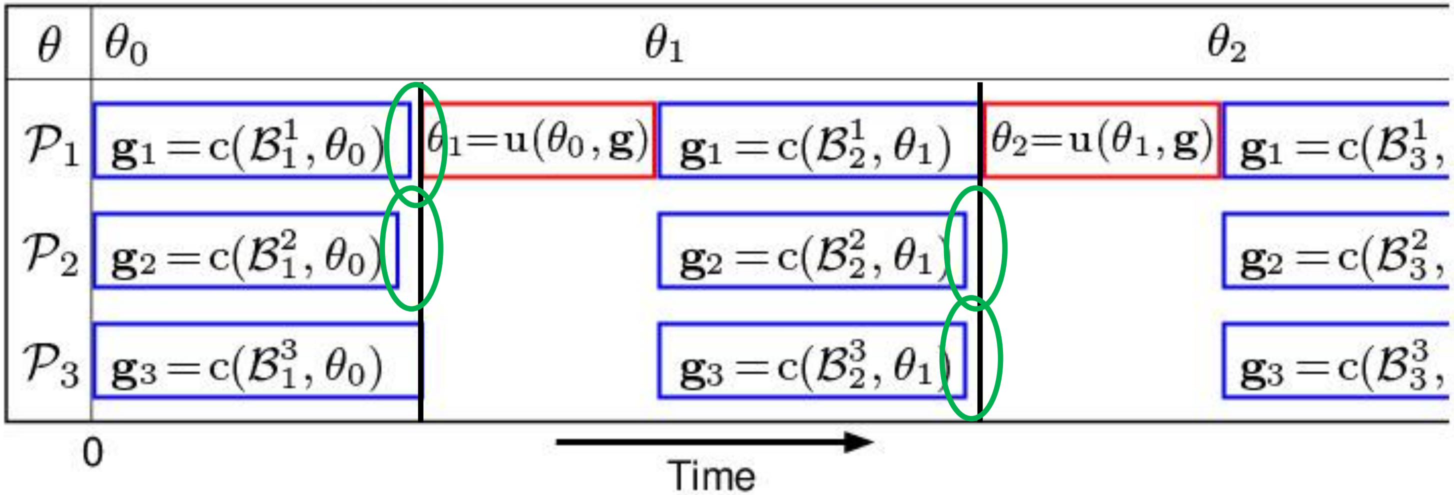
Parameters: θ_t

Processors: $\mathcal{P}_i$

Dataset: $\mathcal{D} = \mathcal{D}_1 \cup \mathcal{D}_2 \cup \mathcal{D}_3$

Gradient: $\mathbf{g} = \mathbf{g}_1 + \mathbf{g}_2 + \mathbf{g}_3$

Parallel Synchronous Mini-Batch Learning



- More frequent updates

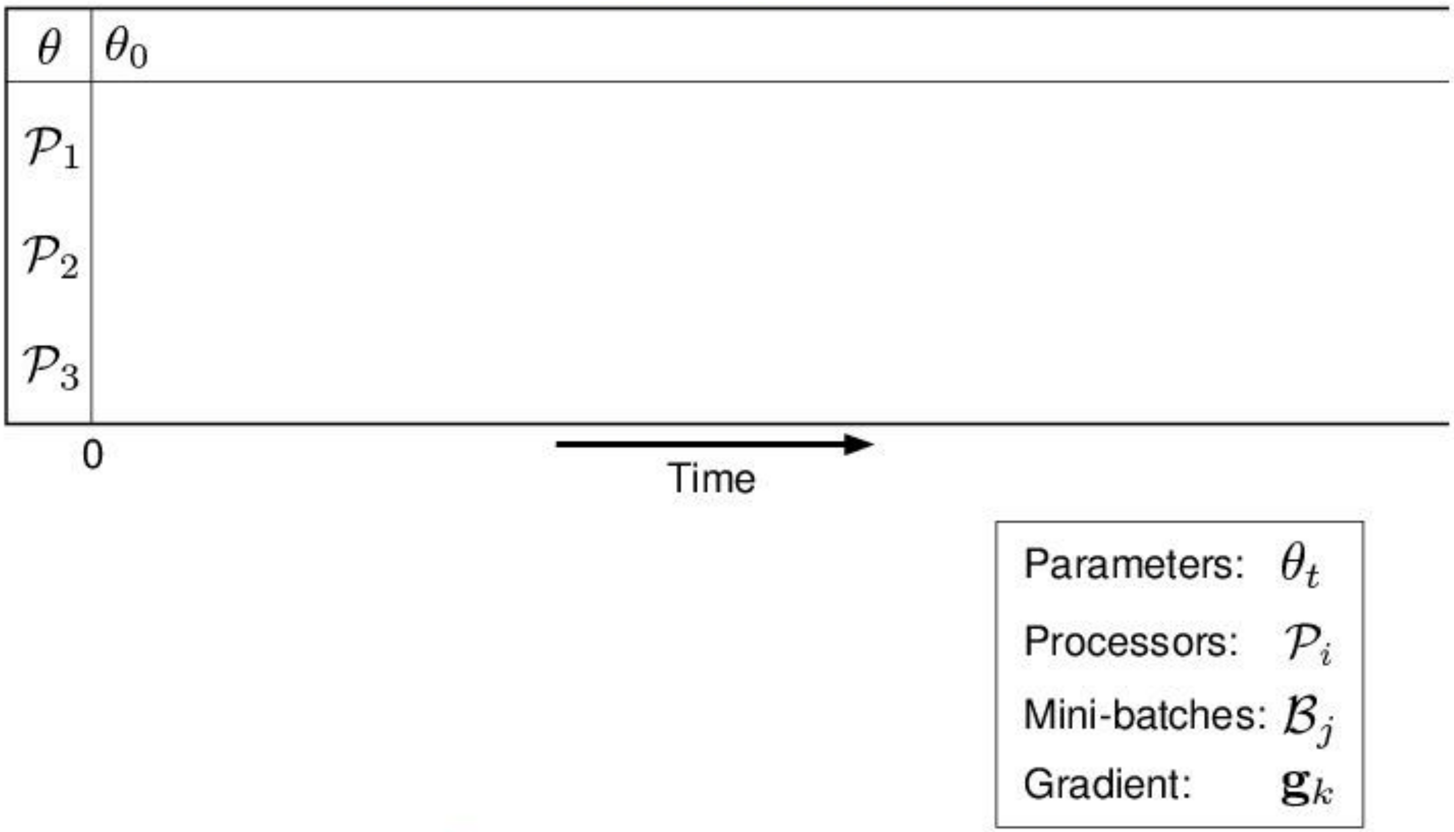
Parameters: θ_t

Processors: $\mathcal{P}_i$

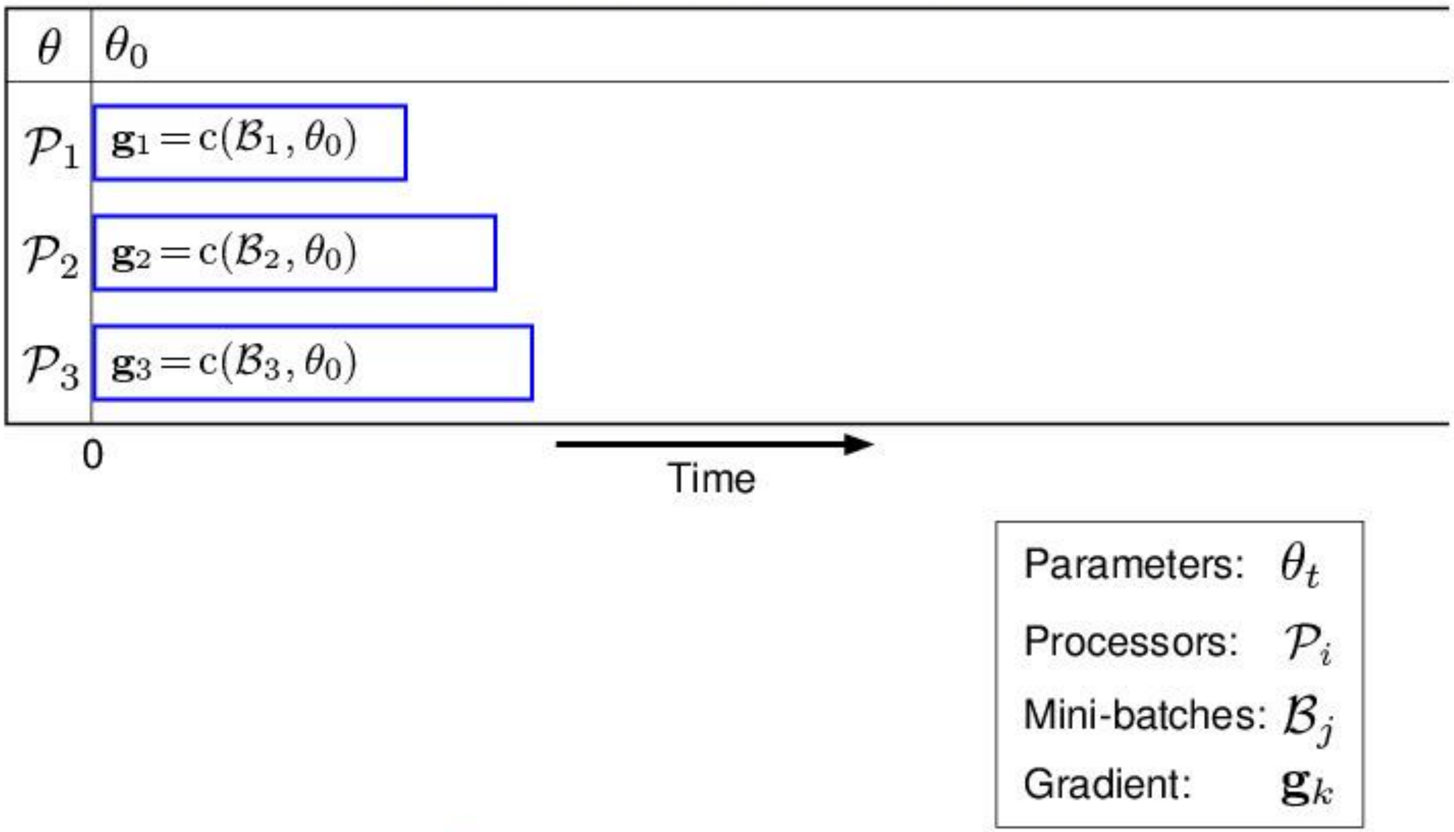
Mini-batches: $\mathcal{B}_t = \mathcal{B}_t^1 \cup \mathcal{B}_t^2 \cup \mathcal{B}_t^3$

Gradient: $\mathbf{g} = \mathbf{g}_1 + \mathbf{g}_2 + \mathbf{g}_3$

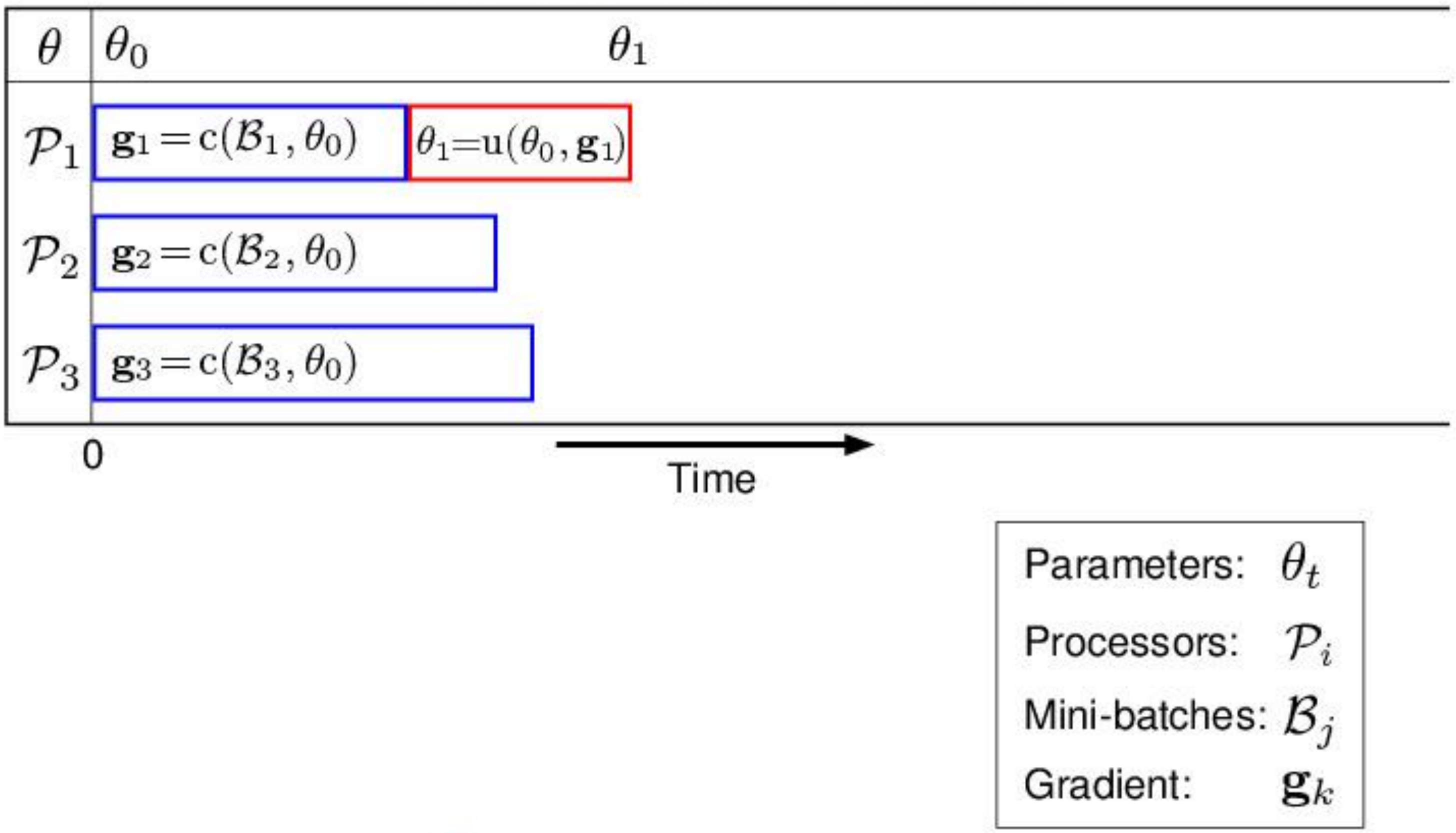
Parallel Asynchronous Mini-Batch Learning



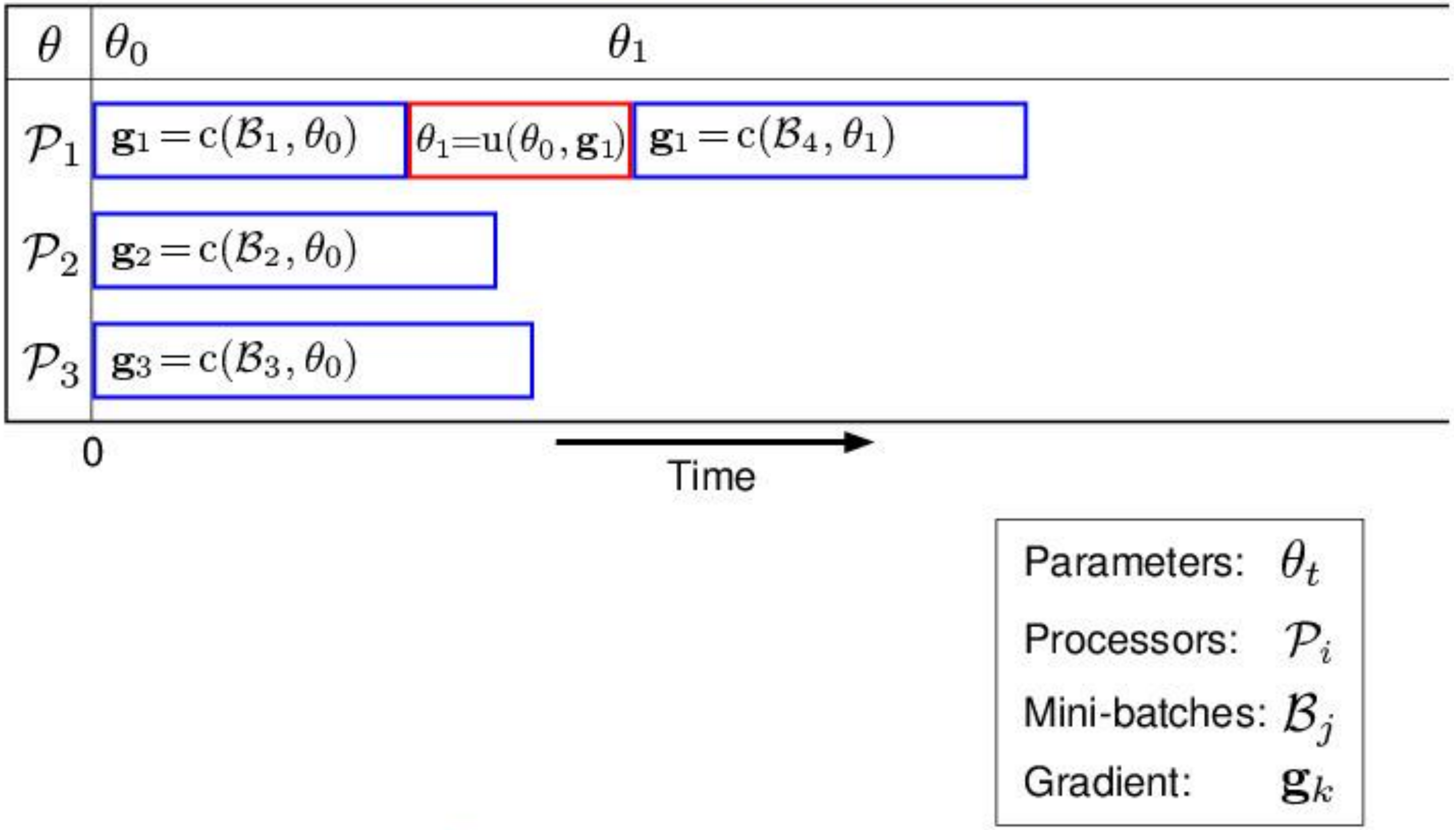
Parallel Asynchronous Mini-Batch Learning



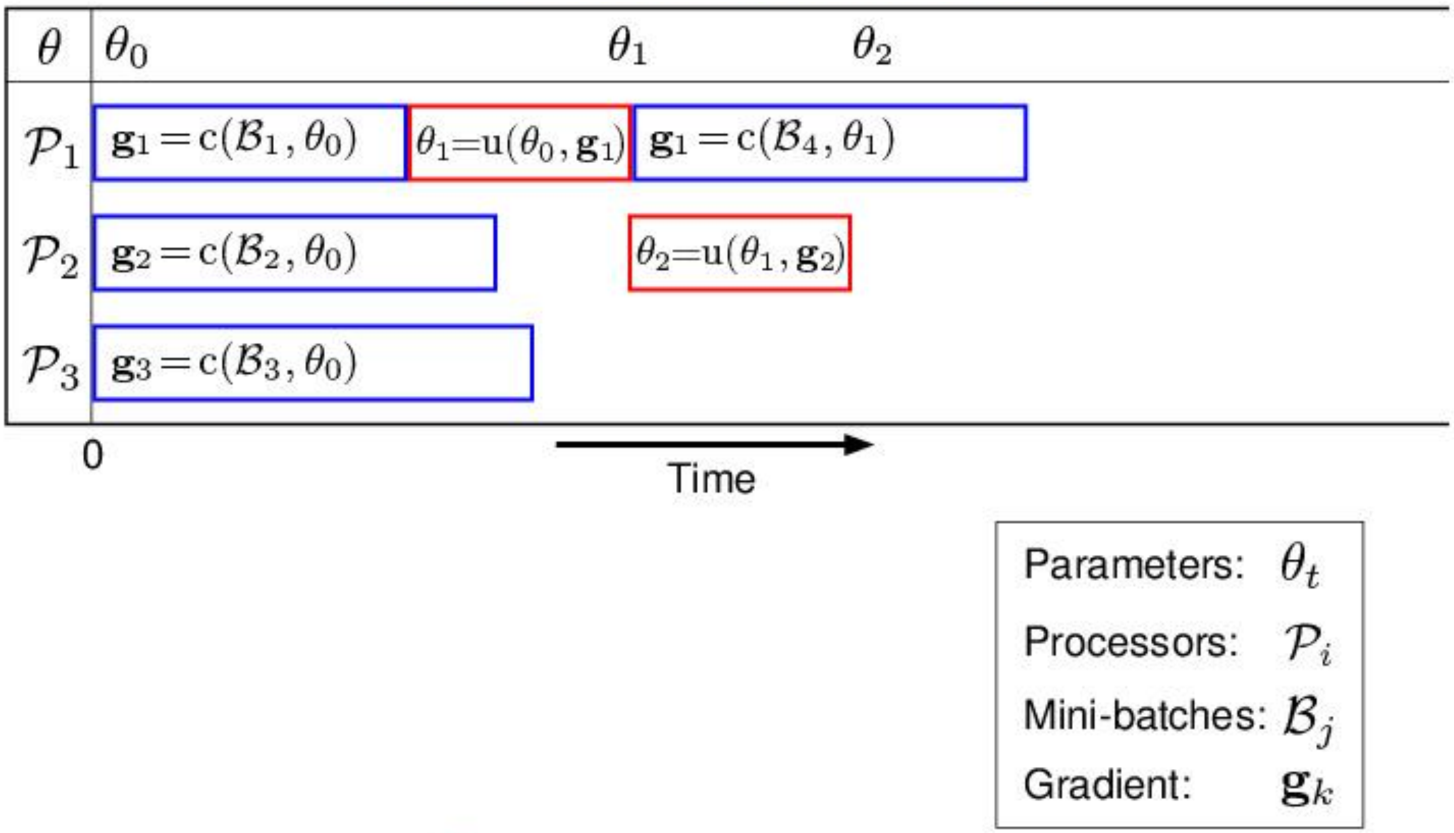
Parallel Asynchronous Mini-Batch Learning



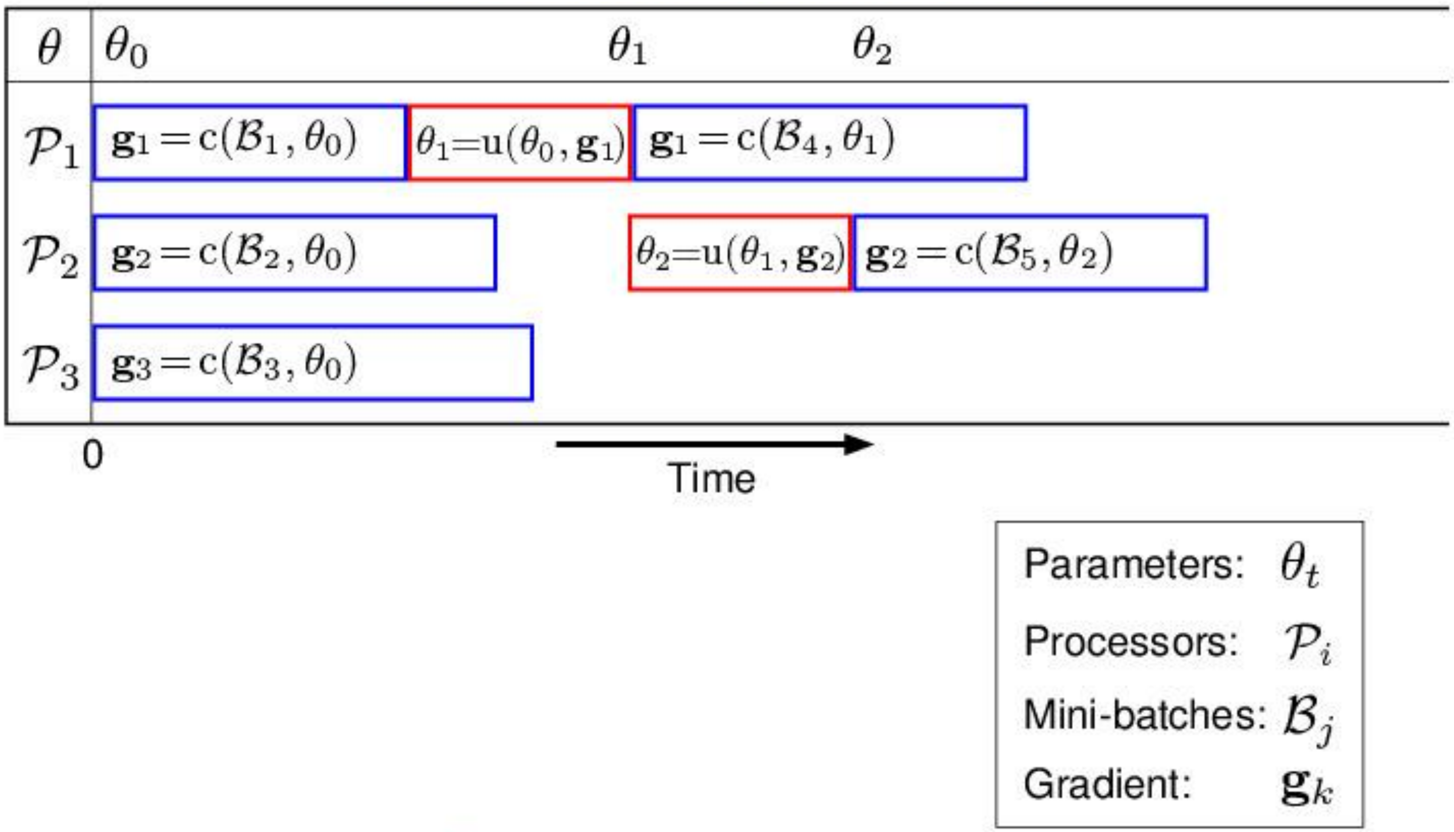
Parallel Asynchronous Mini-Batch Learning



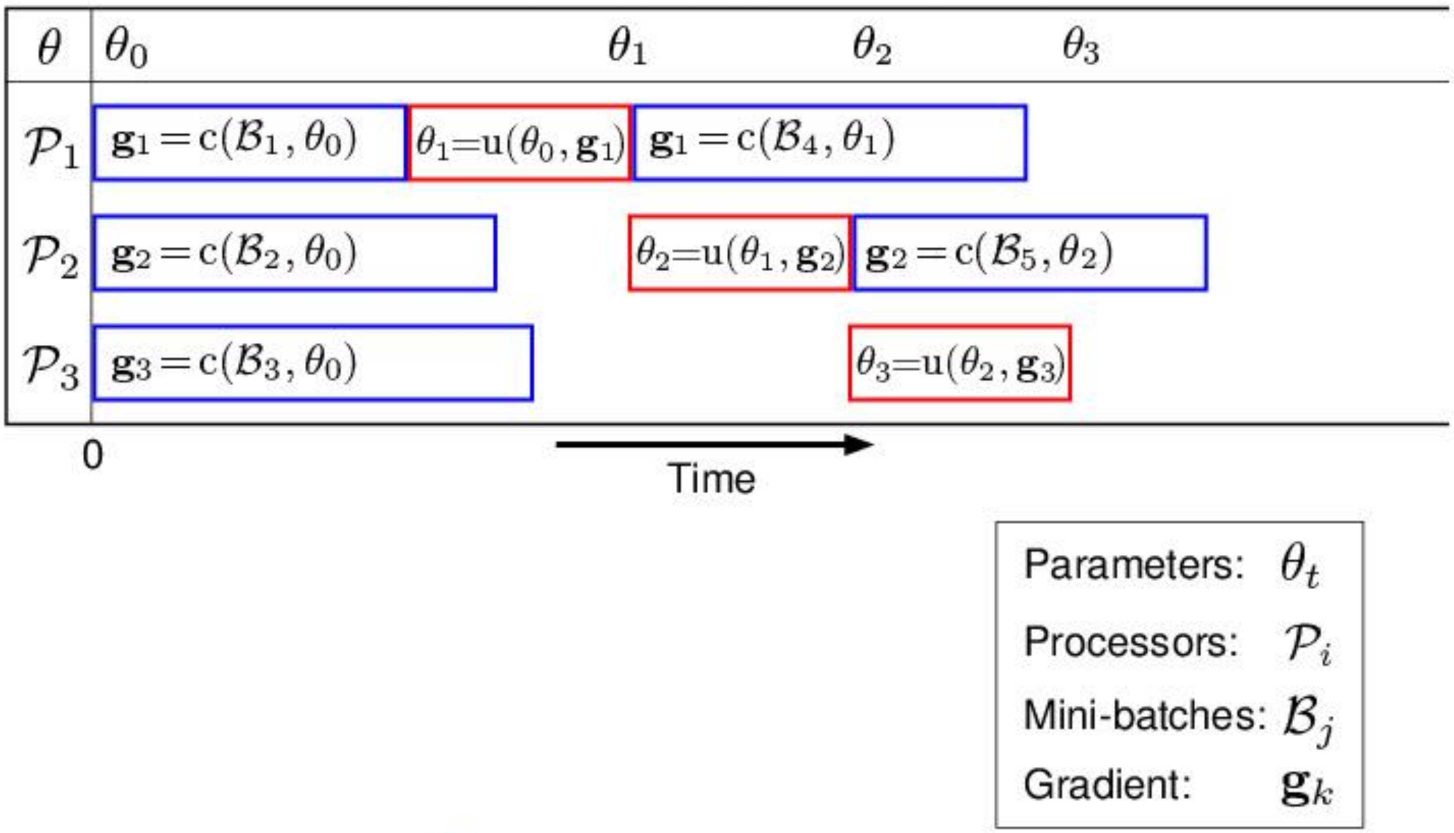
Parallel Asynchronous Mini-Batch Learning



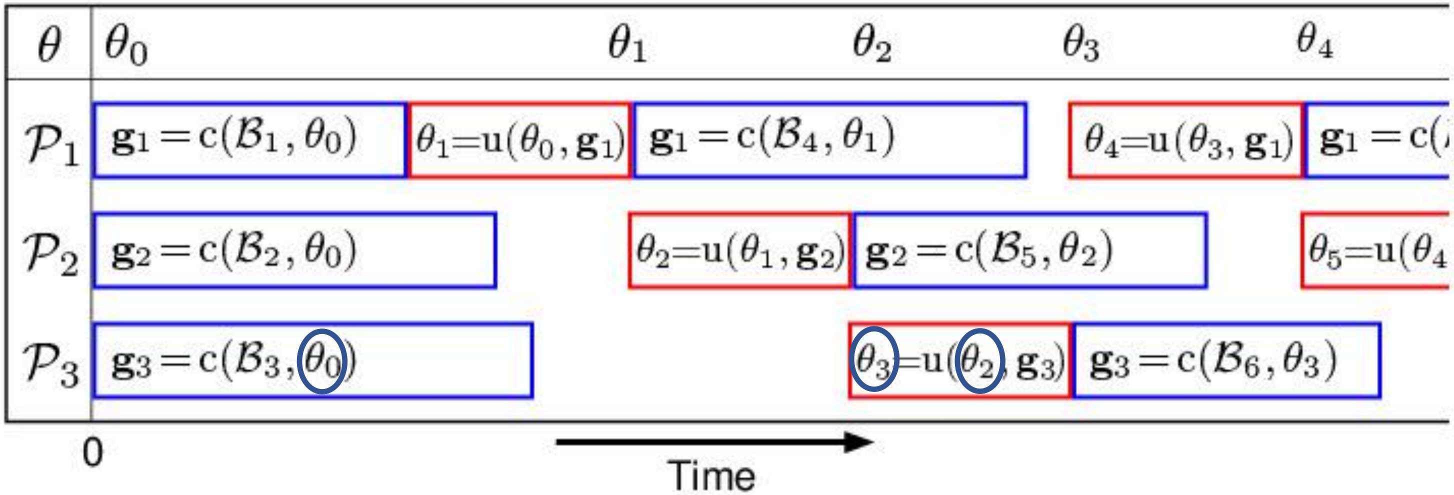
Parallel Asynchronous Mini-Batch Learning



Parallel Asynchronous Mini-Batch Learning



Parallel Asynchronous Mini-Batch Learning



- Gradients computed using ***stale*** parameters
- Increased utilization
- Central lock

Parameters:	θ_t
Processors:	$\mathcal{P}_i$
Mini-batches:	$\mathcal{B}_j$
Gradient:	$\mathbf{g}_k$

Distributed ML

- Parallelism
  - Model
  - Data
- Learning
  - Synchronous
  - Asynchronous

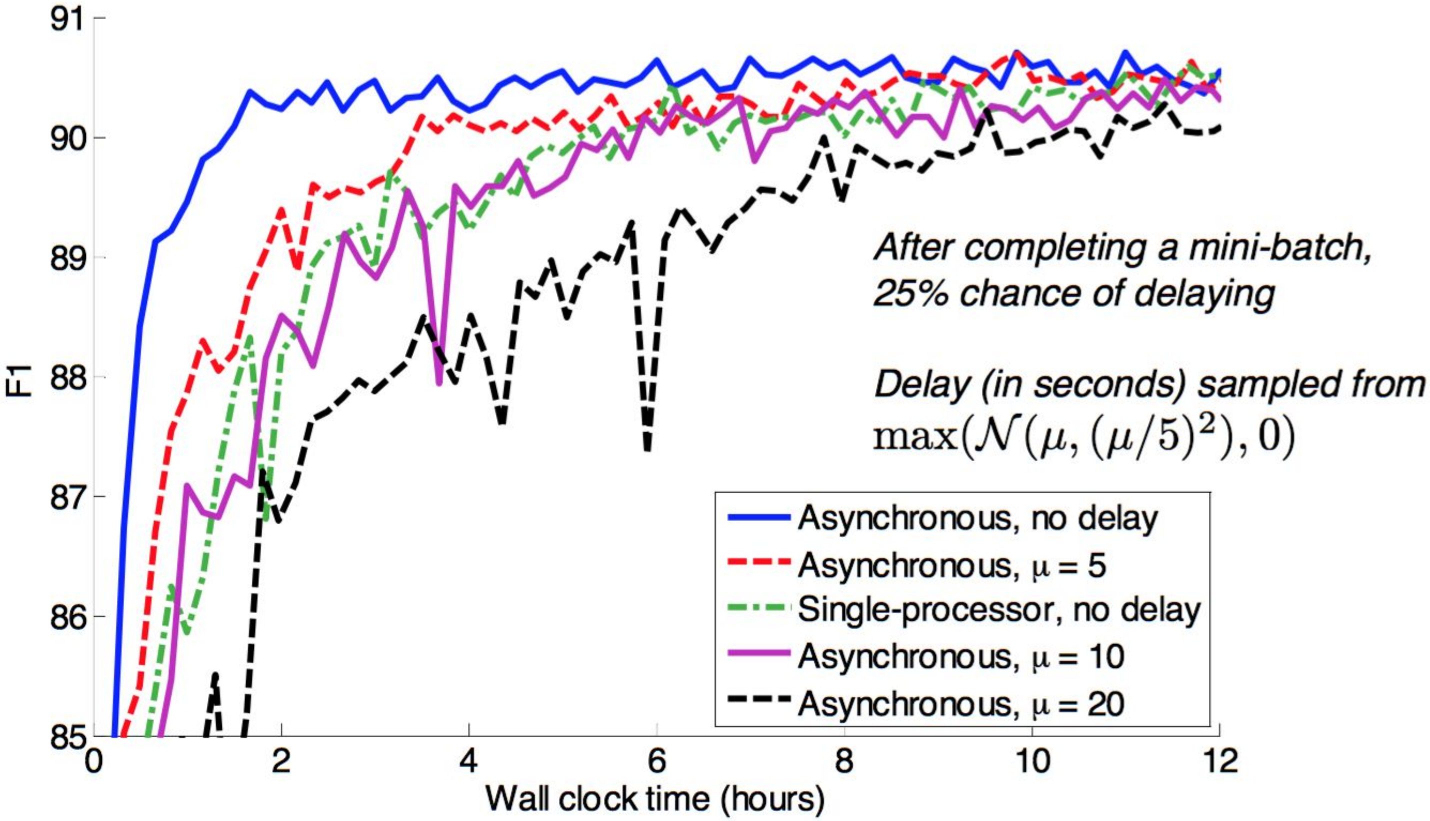
Distributed ML: Challenges

1. **Scalability**

2. **Privacy**

3. **Security**

Scalability - Asynchrony

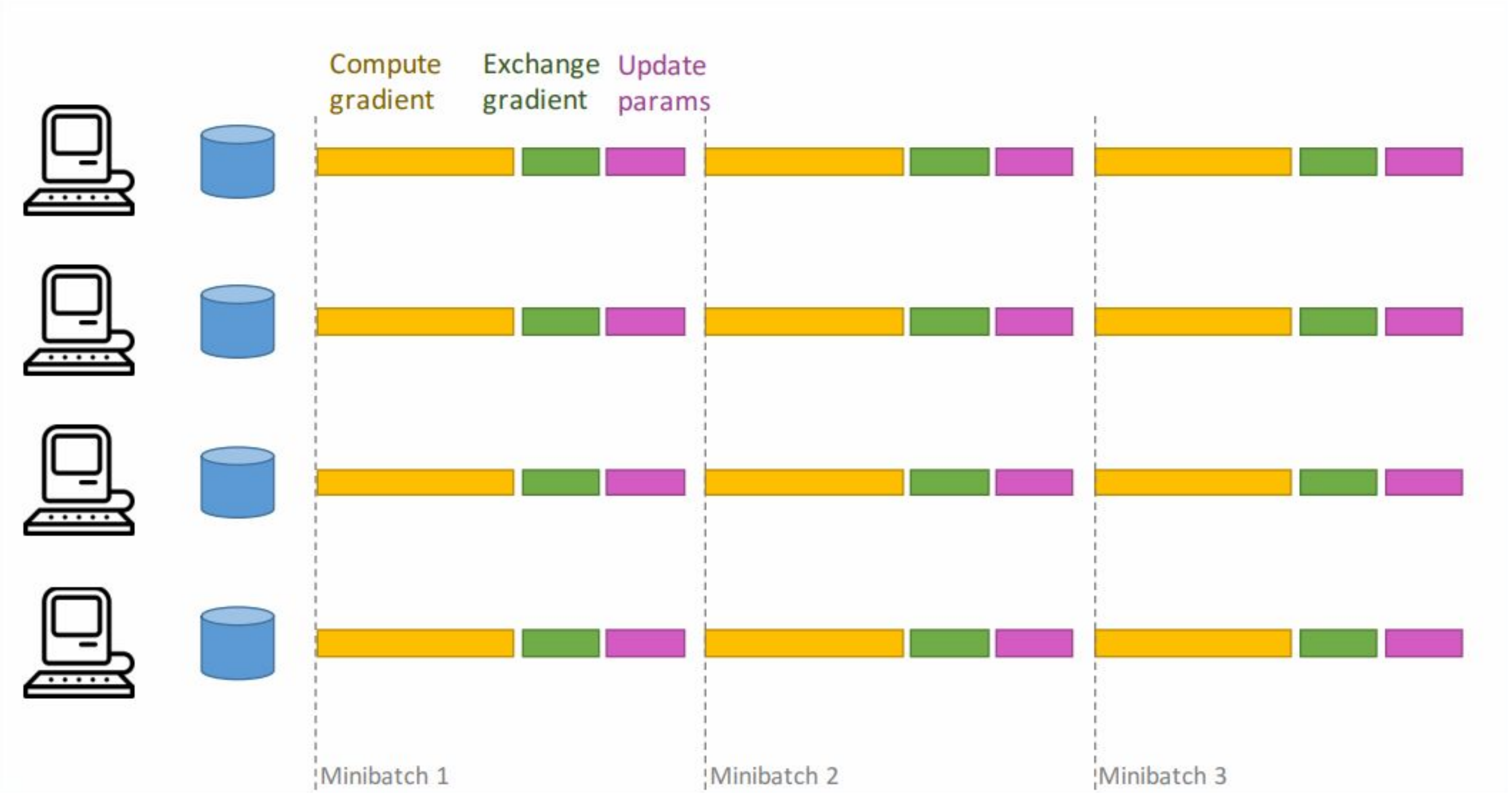


Avg. time per mini-batch = 0.62 s

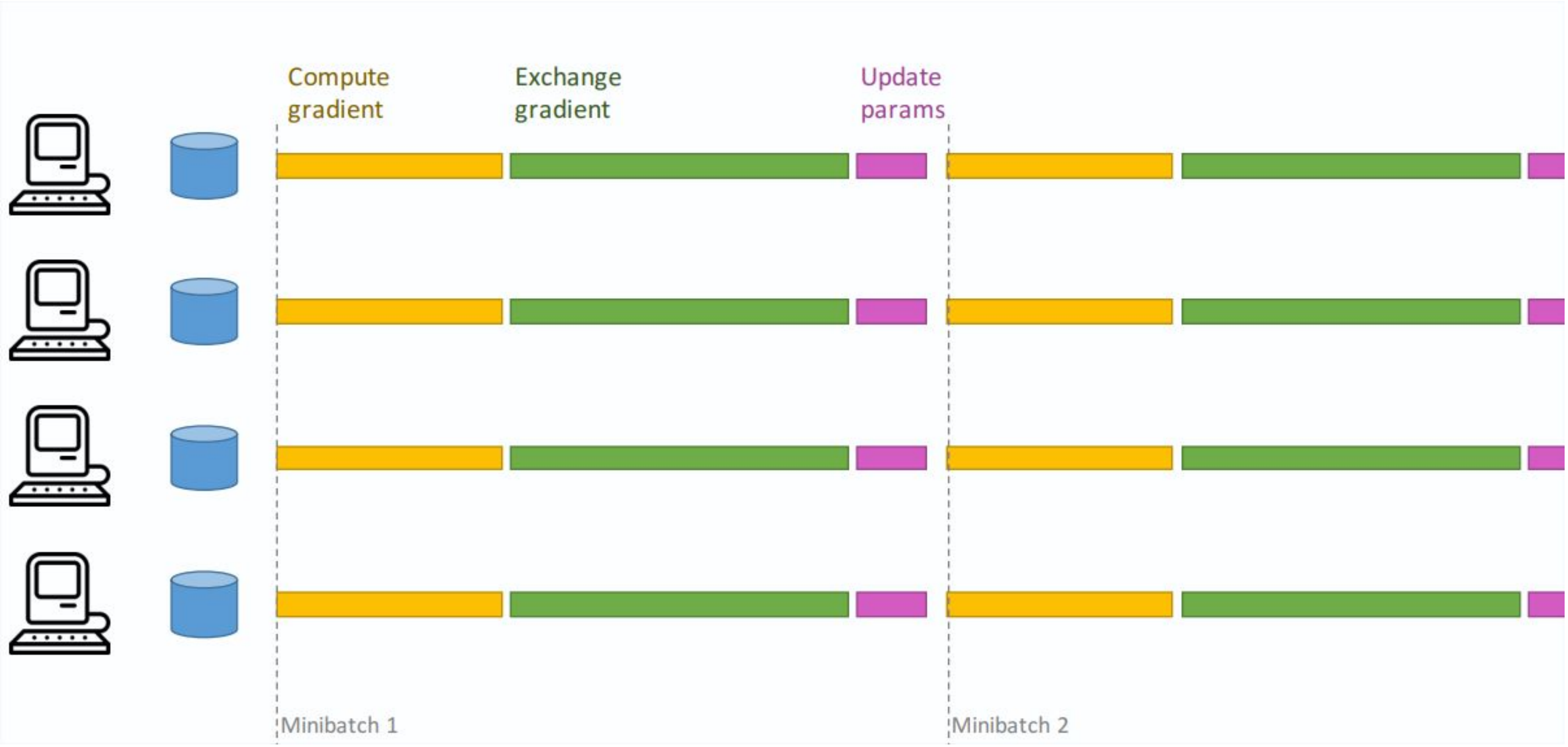
Scalability - Communication

ImageNet classification (ResNet-152):
Model/update size = **~ 250MB**

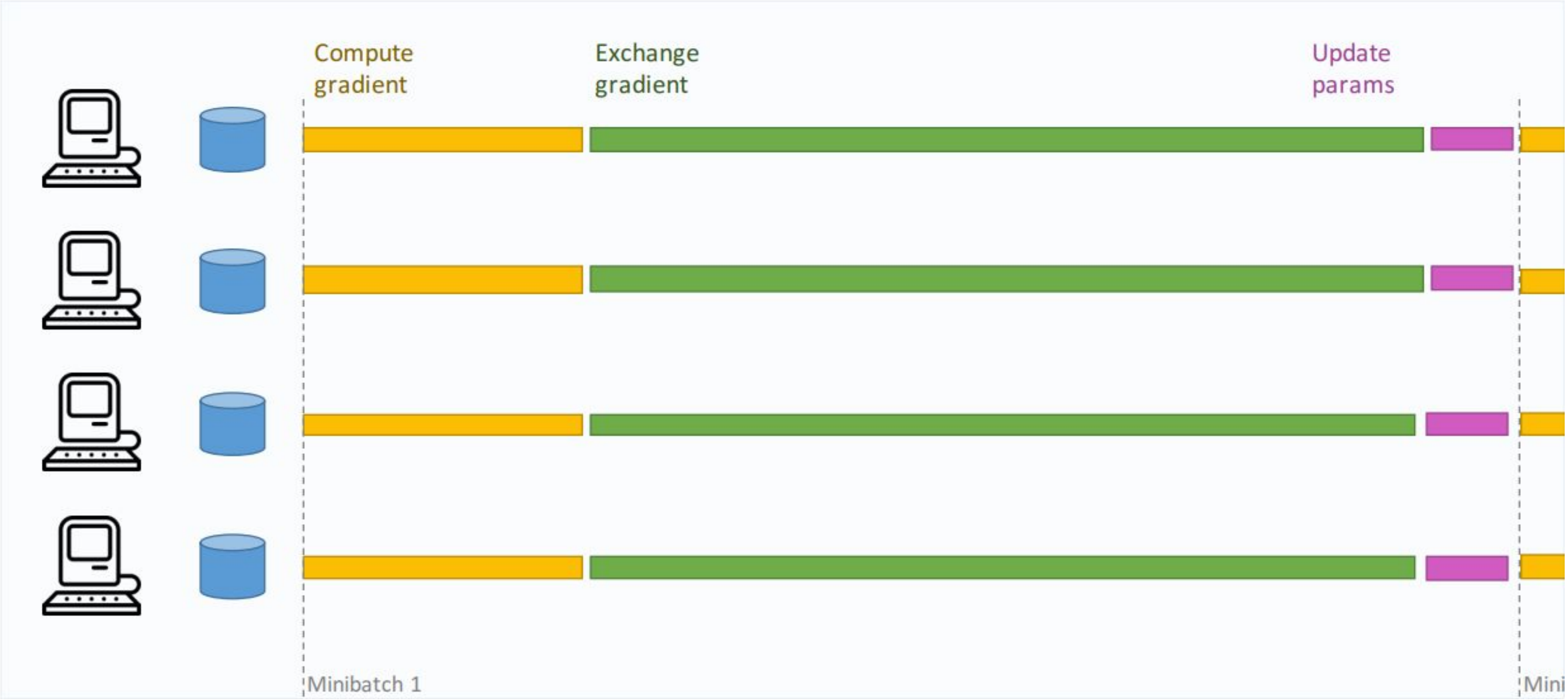
Scalability - Communication



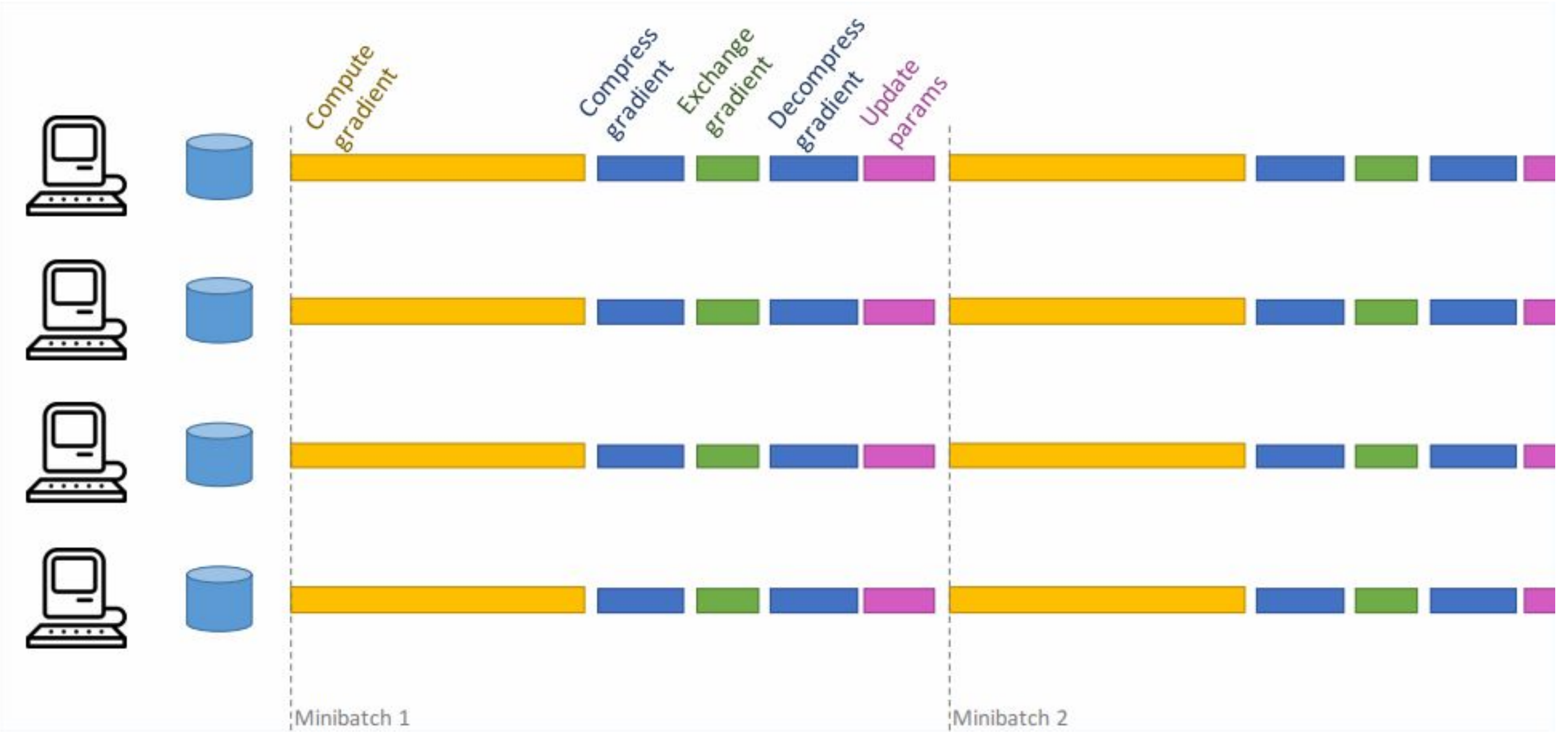
Scalability - Communication



Scalability - Communication



Scalability - Communication



Scalability - Communication

ImageNet classification (ResNet-152):
Mode/update size = **~ 250MB**

Compression

- Distillation [PPA+18]
- Quantization [DGL+17]
  - SignSGD [BJ+18]

[DGL+17] Alistarh, Dan, et al. "QSGD: Communication-efficient SGD via gradient quantization and encoding." NIPS 2017.
[PPA+18] Polino, Antonio, Razvan Pascanu, and Dan Alistarh. "Model compression via distillation and quantization." ICLR 2018.
[BJ+18] Bernstein, Jeremy, et al. "signSGD: compressed optimisation for non-convex problems." ICML 2018.

Distributed ML: Challenges

1. **Scalability**
   a. Asynchrony
   b. Communication efficiency

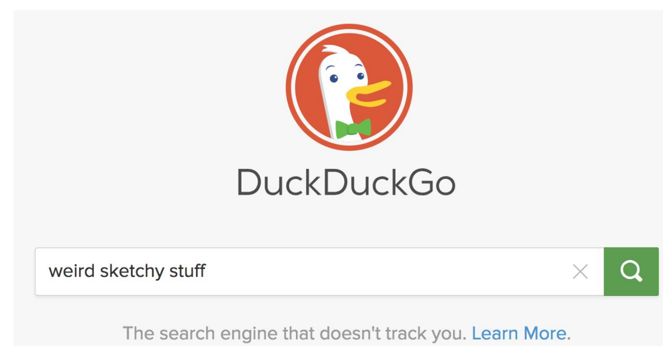
2. **Privacy**

3. **Security**

- Medical data

- Photos

- Search logs



• • •

Privacy

Differential Privacy

- Decentralized Learning [BGT+18]
- Compression <-> DP [AST+18]

Local Privacy

- MPC

[BGT+18] Bellet, A., Guerraoui, R., Taziki, M., & Tommasi, M.. Personalized and Private Peer-to-Peer Machine Learning. AISTATS 2018.
[AST+18] Agarwal, N., Suresh, A. T., Yu, F., Kumar, S., & Mcmahan, H. B. (2018). cpSGD: Communication-efficient and differentially-private distributed SGD. NIPS 2018.

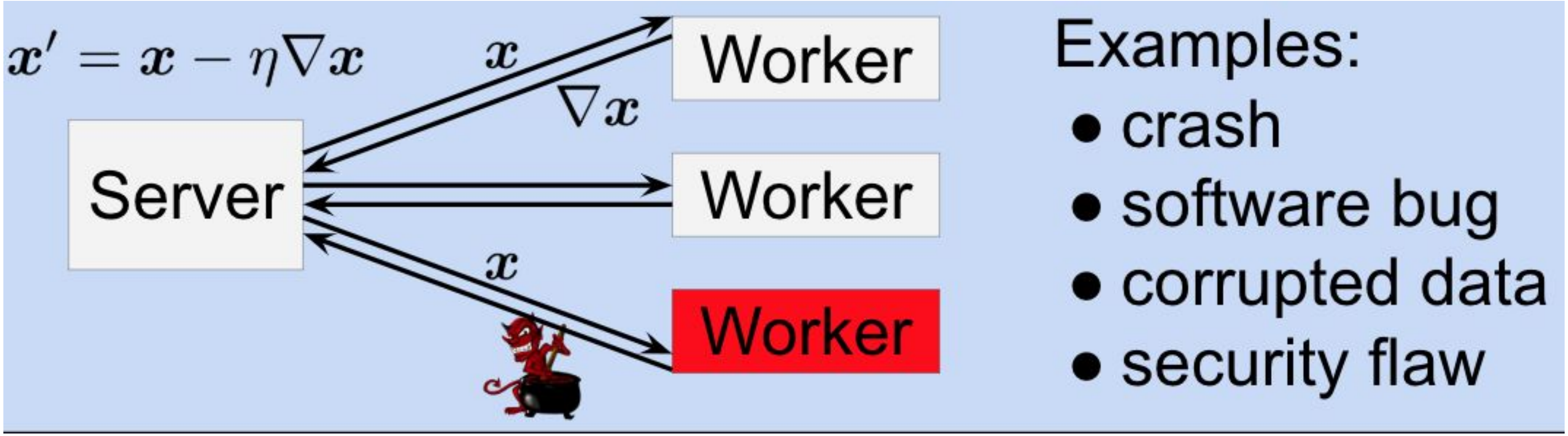
Distributed ML: Challenges

1. **Scalability**
    a. Asynchrony
    b. Communication efficiency

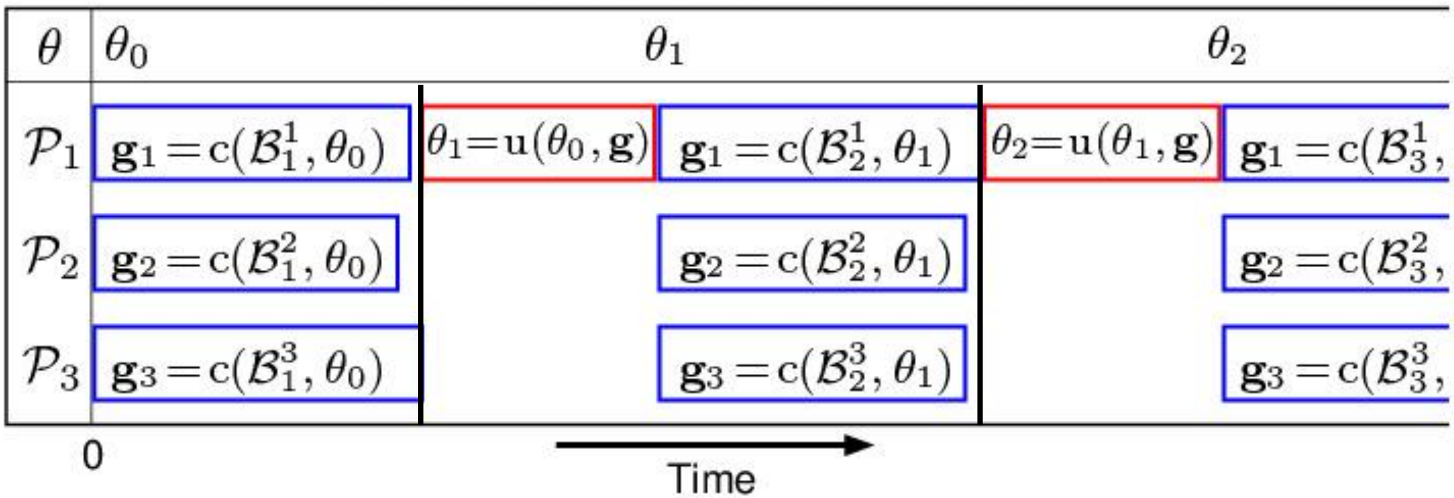
2. **Privacy**
    a. Differential Privacy
    b. Local Privacy

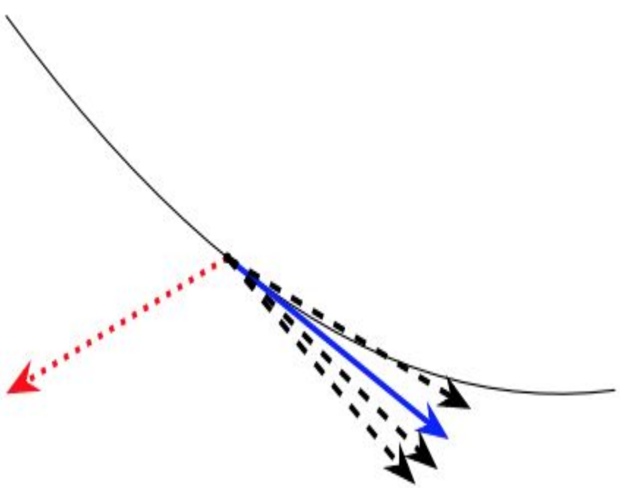
3. **Security**

Security: Byzantine worker



Security: Synchronous BFT





Parameters: θ_t

Processors: $\mathcal{P}_i$

Mini-batches: $\mathcal{B}_t = \mathcal{B}_t^1 \cup \mathcal{B}_t^2 \cup \mathcal{B}_t^3$

Gradient: $\mathbf{g} = \mathbf{g}_1 + \mathbf{g}_2 + \mathbf{g}_3$

Security: Synchronous BFT

Krum

[Blanchard, Peva, Mhamdi, E. M. E., Rachid Guerraoui, and Julien Stainer. "Machine learning with adversaries: Byzantine tolerant gradient descent." NIPS. 2017.]

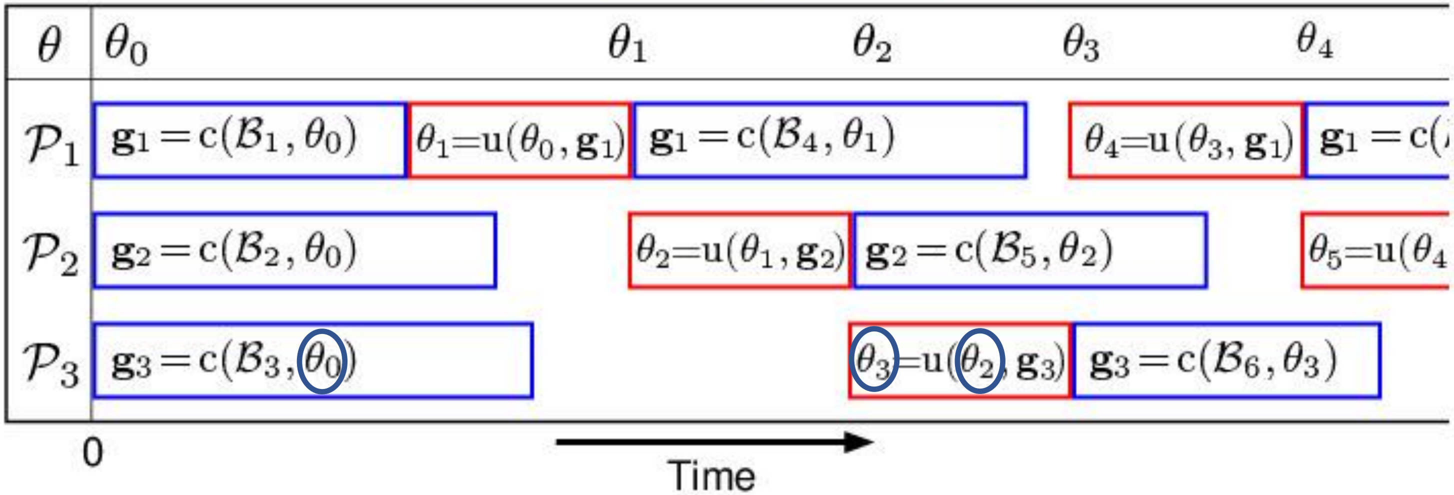
- Byzantine resilience against f/n workers, 2f + 2 < n
- Provable convergence (i.e., *safety*)

How?

1. Worker i: score(i) = $\sum_{n-f-2 \; closest \; vectors \; to \; G_i} ||G_i - Gj||^2$
   Select gradient with minimum score
2. m-Krum

Majority + Squared distance-based decision -> BFT

Security: Asynchronous BFT



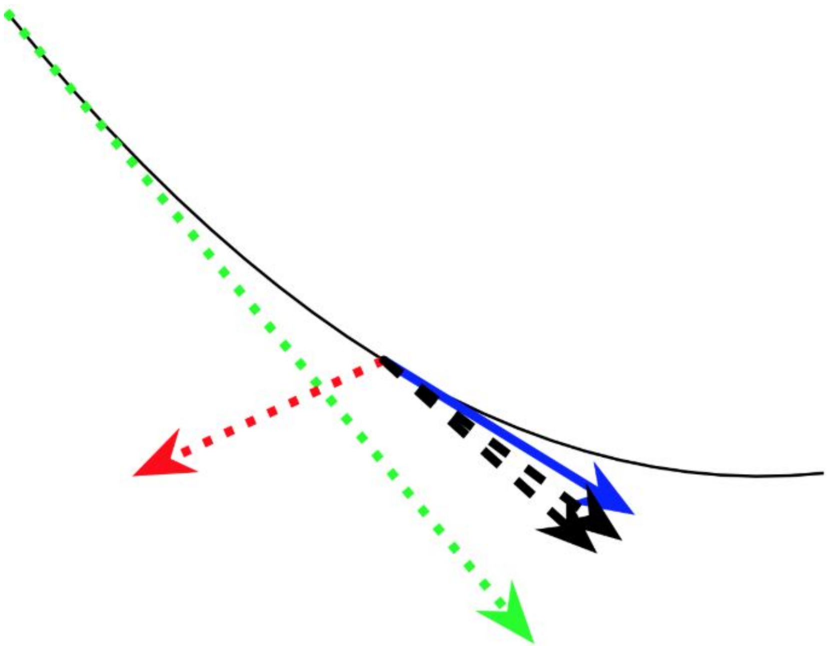
stale

Parameters: θ_t

Processors: $\mathcal{P}_i$

Mini-batches: $\mathcal{B}_j$

Gradient: $\mathbf{g}_k$



Security: Asynchronous BFT

Kardam

[Damaskinos, G., Mhamdi, E. M. E., Guerraoui, R., Patra, R., & Taziki, M. Asynchronous Byzantine Machine Learning. ICML 2018.]

- Byzantine resilience against f/n workers, $3f < n$
- Optimal slowdown: $\frac{n-2f}{n-f} \leq SL \leq \frac{n-f}{n}$
- Provable (almost sure) convergence (i.e., *safety*)

How?

1. Lipschitz Filtering Component => Byzantine resilience
2. Staleness Dampening Component => Asynchronous convergence

Asynchrony can be viewed as Byzantine behavior

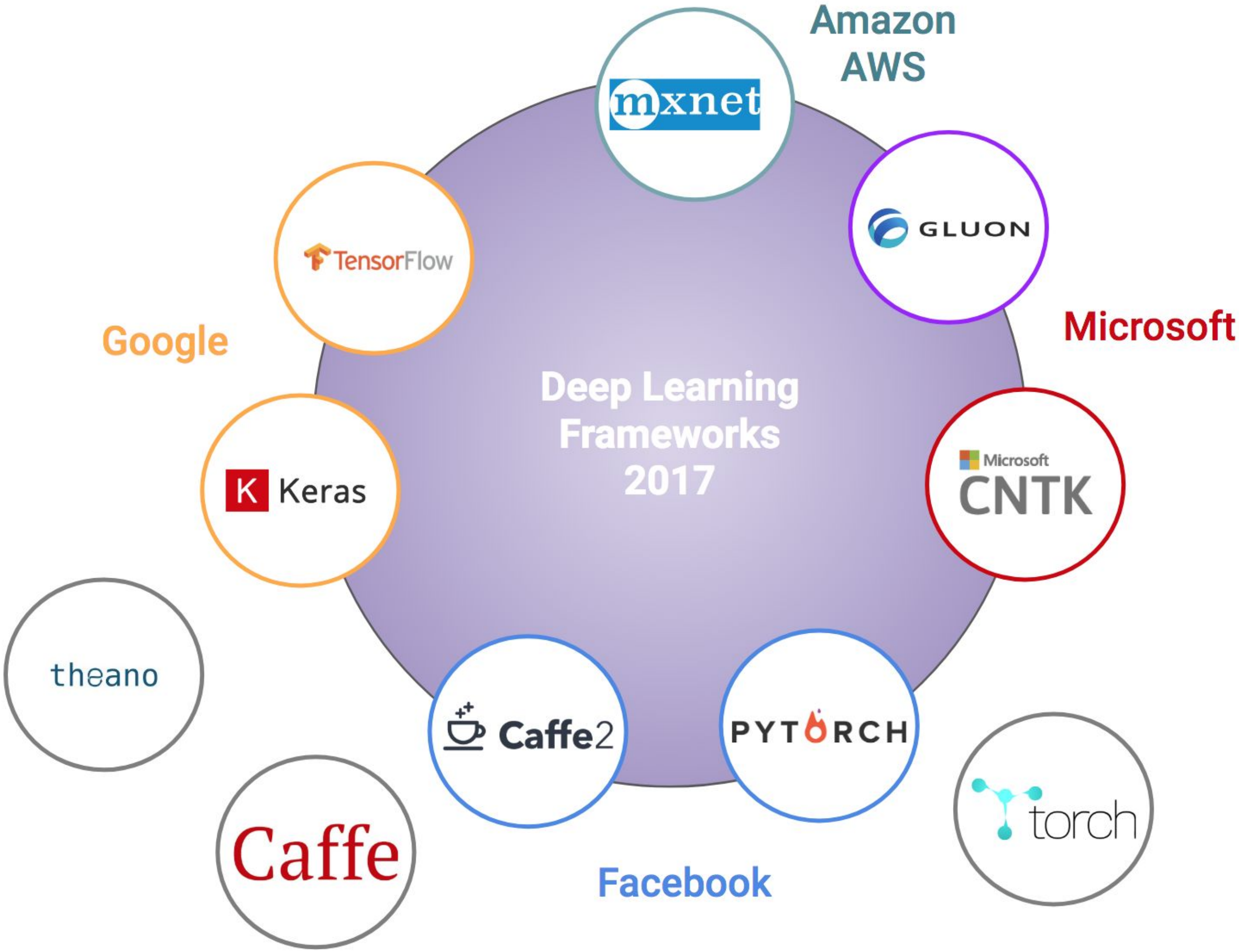
Distributed ML: Challenges

1. **Scalability**
    a. Asynchrony
    b. Communication efficiency

2. **Privacy**
    a. Differential Privacy
    b. Local Privacy

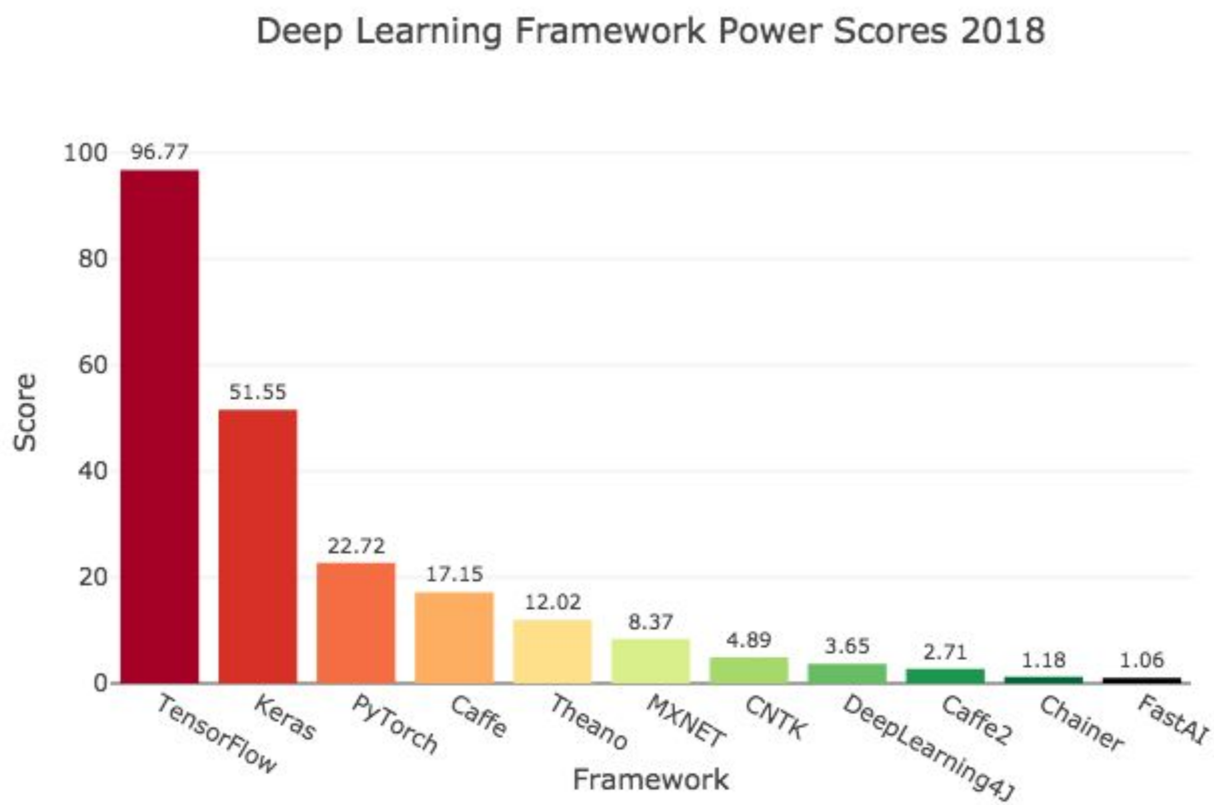
3. **Security**
    a. Synchronous BFT
    b. Asynchronous BFT

Distributed ML: Frameworks



Tensorflow: Why?

Popularity

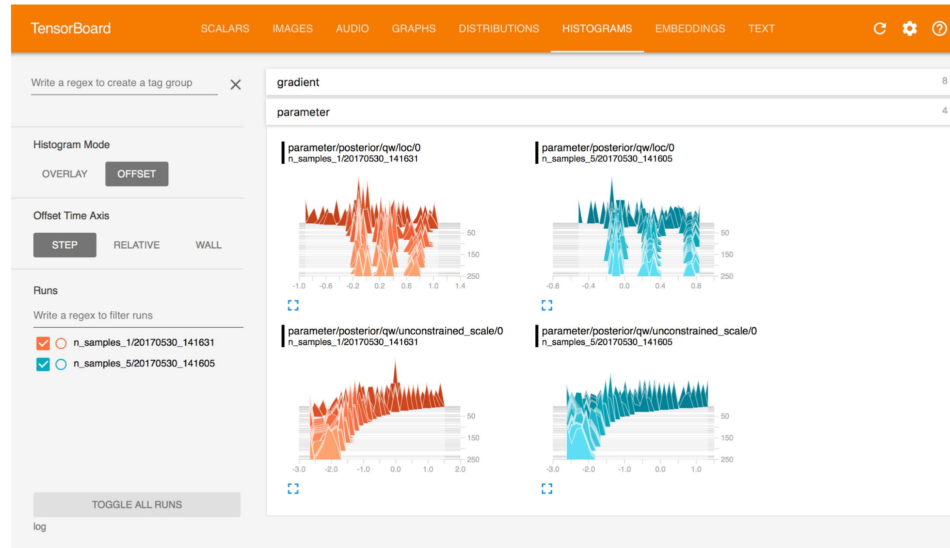


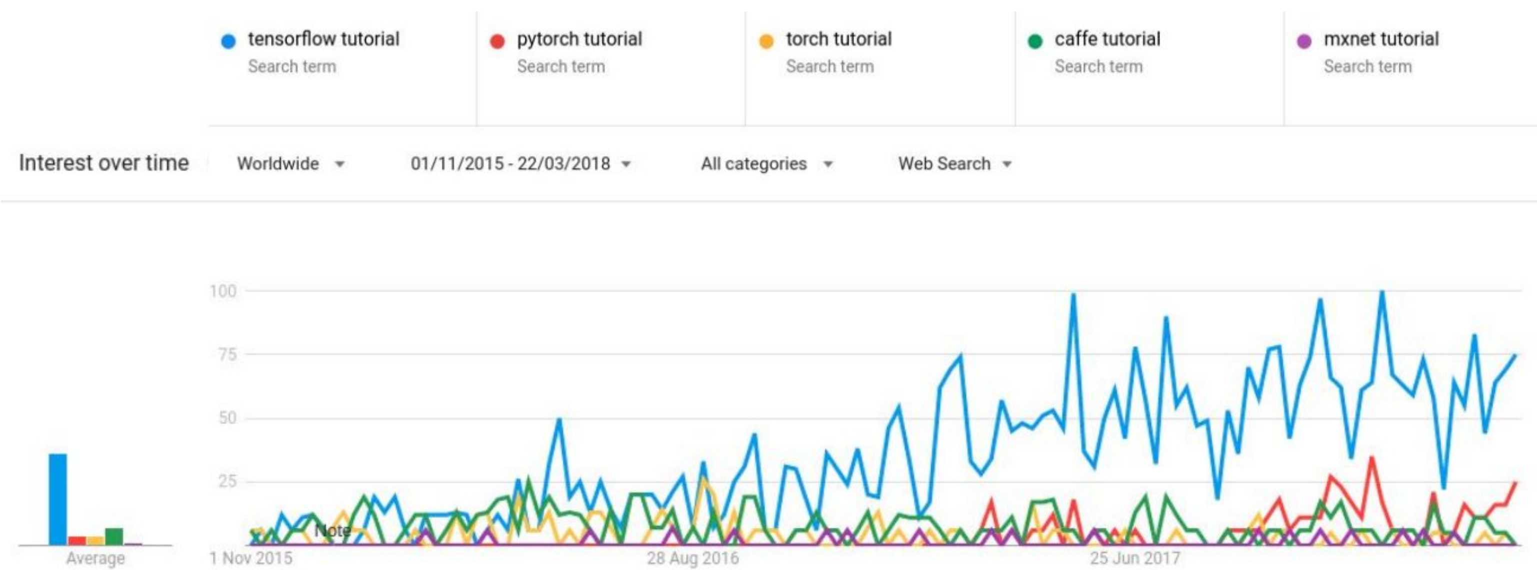
https://towardsdatascience.com/deep-learning-framework-power-scores-2018-23607ddf297a

Tensorflow: Why?

Support

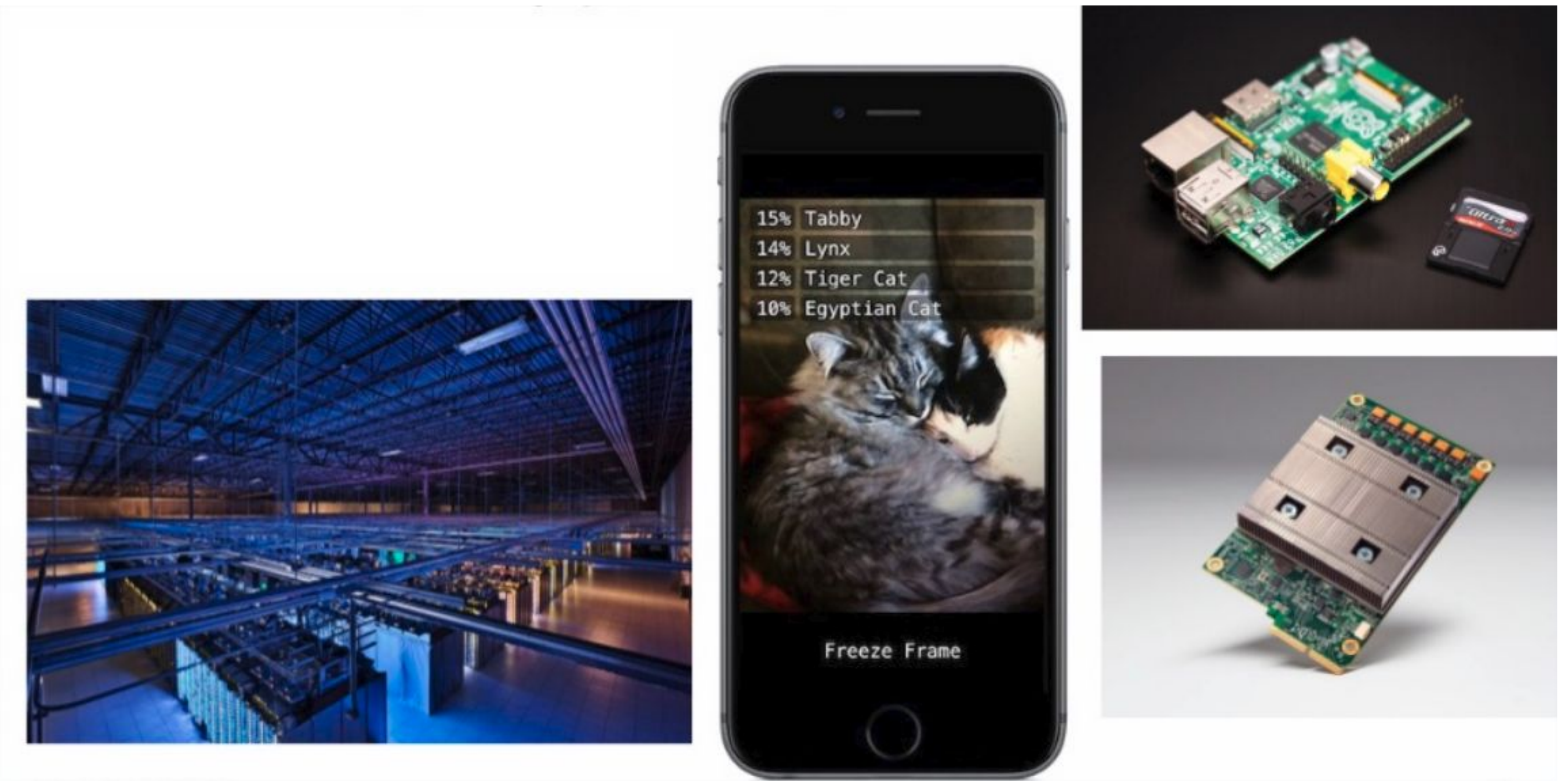
- Visualization tools
- Documentation
- Tutorials





Tensorflow: Why?

<u>Portability - Flexibility - Scalability</u>



Tensorflow: What is it?

- Dataflow graph computation
- Automatic differentiation (also for while loops [Y+18])

[Y+18] Yu, Yuan, et al. "Dynamic control flow in large-scale machine learning." EuroSys. ACM, 2018.

Tensor?

Multidimensional array of numbers

Examples:

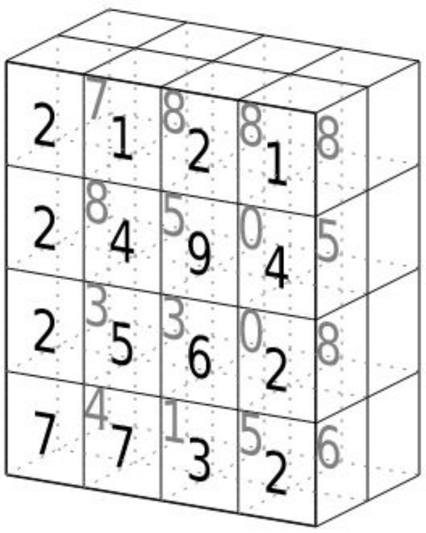
- A scalar
- A vector
- A matrix

't'
'e'
'n'
's'
'o'
'r'

tensor of dimensions [6]
(vector of dimension 6)

3	1	4	1
5	9	2	6
5	3	5	8
9	7	9	3
2	3	8	4
6	2	6	4

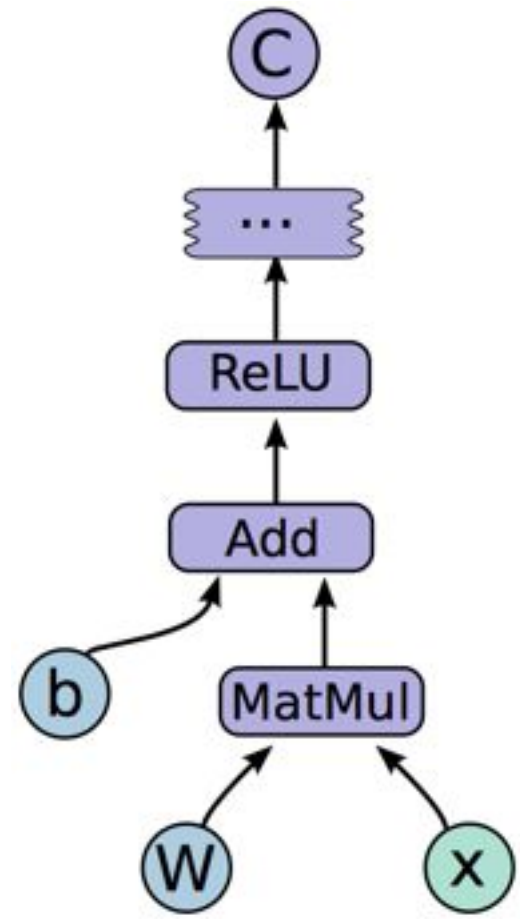
tensor of dimensions [6,4]
(matrix 6 by 4)



tensor of dimensions [4,4,2]

DataFlow?

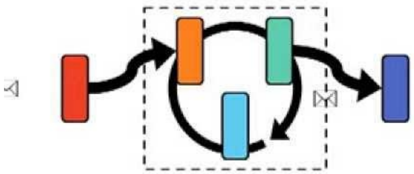
- Computations are **graphs**
  - Nodes: *Operations*
  - Edges: *Tensors*
- Program phases:
  - Construction: create the graph
  - Execution: push data through the graph



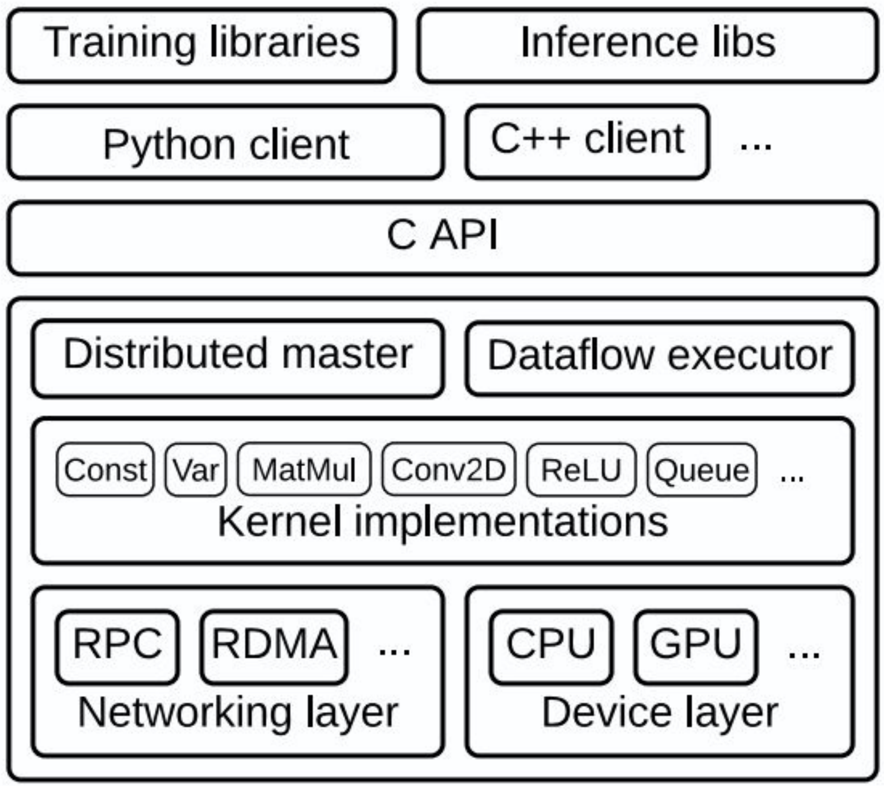
Tensorflow VS DataFlow Frameworks

- Batch Processing
- Relaxed consistency
- Simplicity
  - No join operations
  - Input diff => new batch

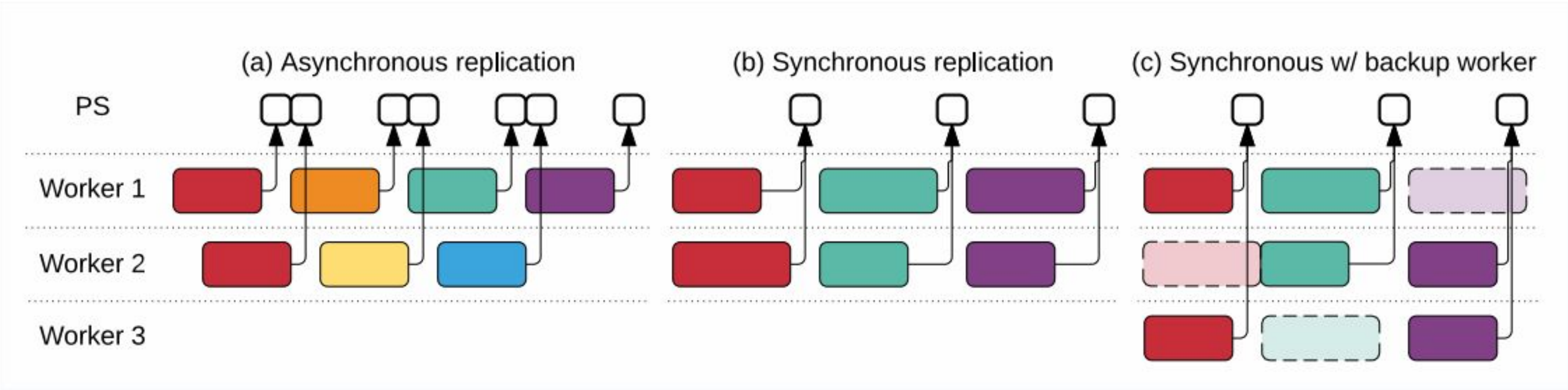
Naiad



Architecture



Learning



(a) Asynchronous replication
(b) Synchronous replication
(c) Synchronous w/ backup worker
PS
Worker 1
Worker 2
Worker 3


Start
DEMO

TensorFlow BFT ? No!

How can we make it BFT?

[Damaskinos G., El Mhamdi E., Guerraoui R., Guirguis A., Rouault S.]