Scaling Distributed Machine Learning

with System and Algorithm Co-design

Mu Li
AWS
03/2017

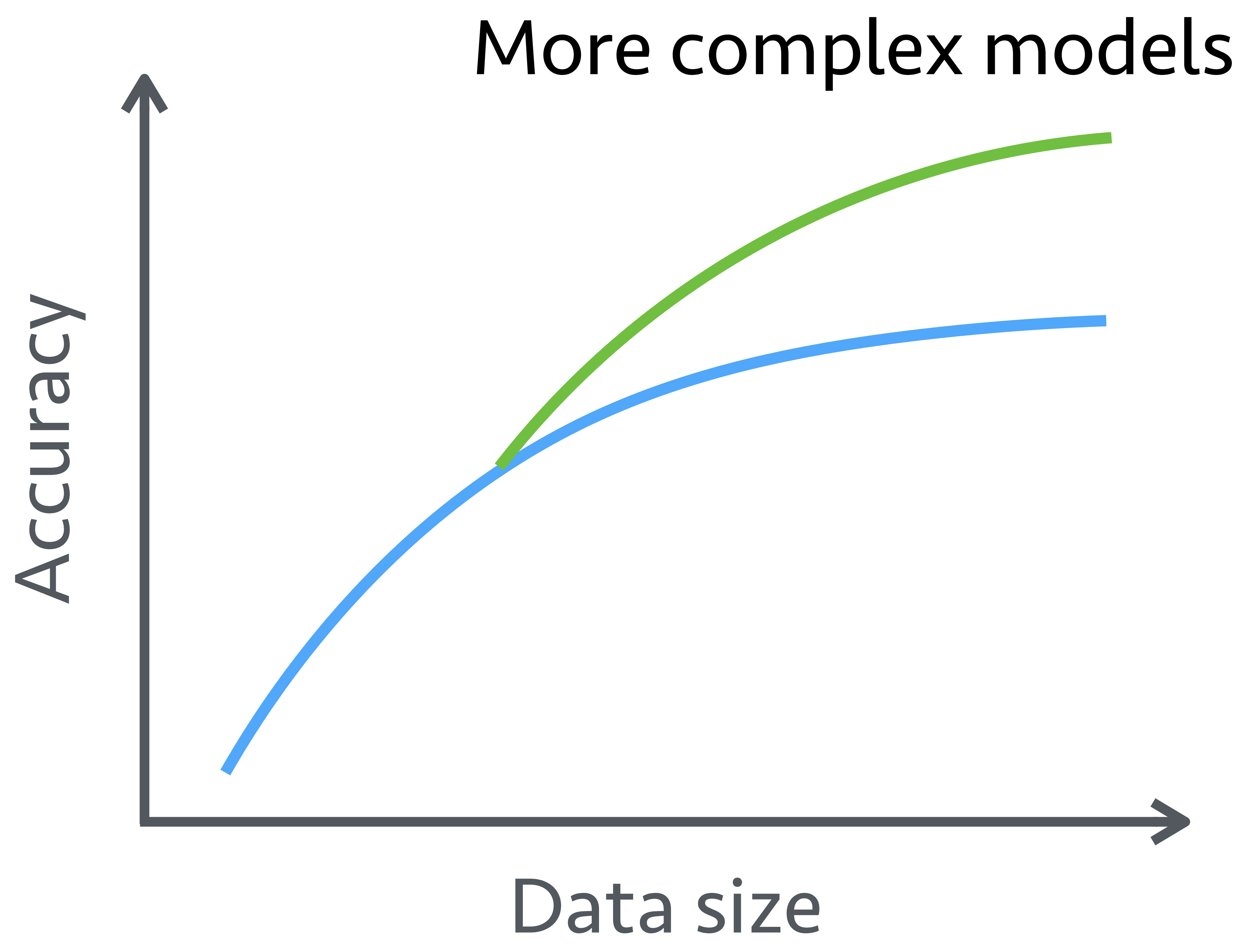
$$\min_{w} \sum_{i=1}^{n} f_i(w)$$

Large-scale problems

- Distributed systems
- Large scale optimization methods

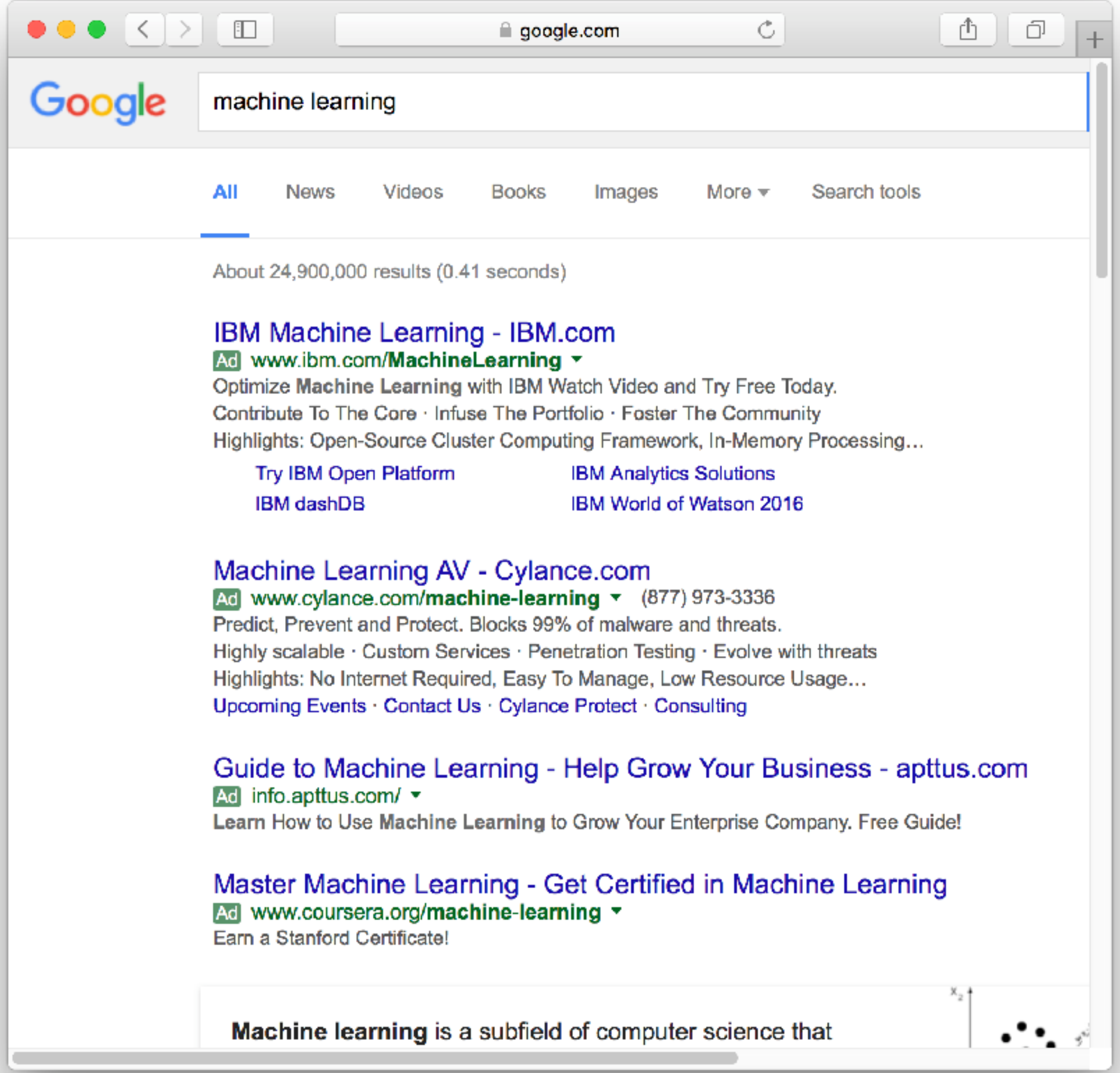
Large Scale Machine Learning

- ✧ Machine learning learns from data
- ✧ More data
  - ✓ better accuracy
  - ✓ can use more complex models



Ads Click Prediction

- ✧ Predict if an ad will be clicked
- ✧ Each ad impression is an example
- ✧ Logistic regression
  - ✓ Single machine processes 1 million examples per second



Ads Click Prediction

- Predict if an ad will be clicked
- Each ad impression is an example
- Logistic regression
  - Single machine processes 1 million examples per second
- A typical industrial size problem has
  - 100 billion examples
  - 10 billion unique features

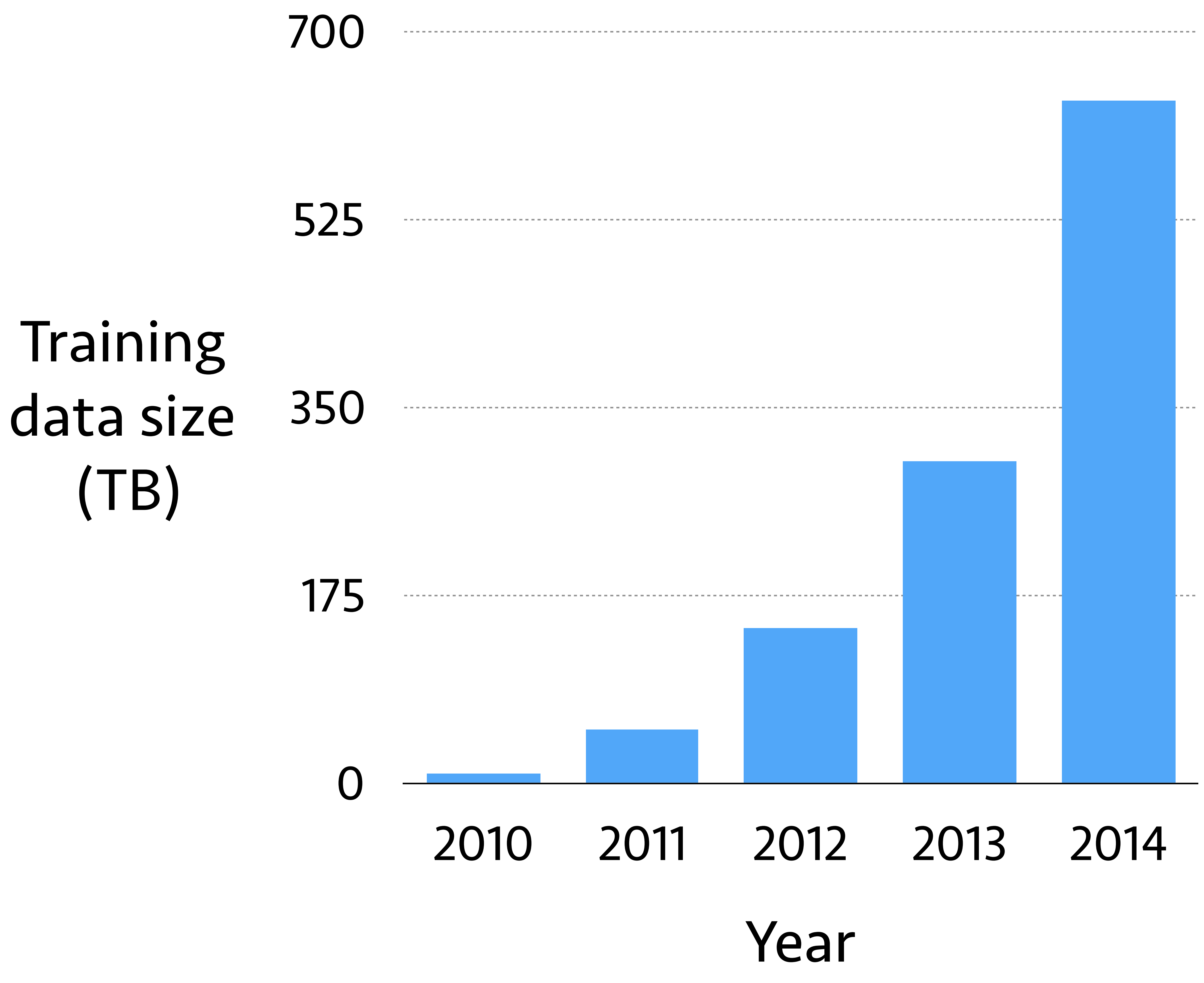
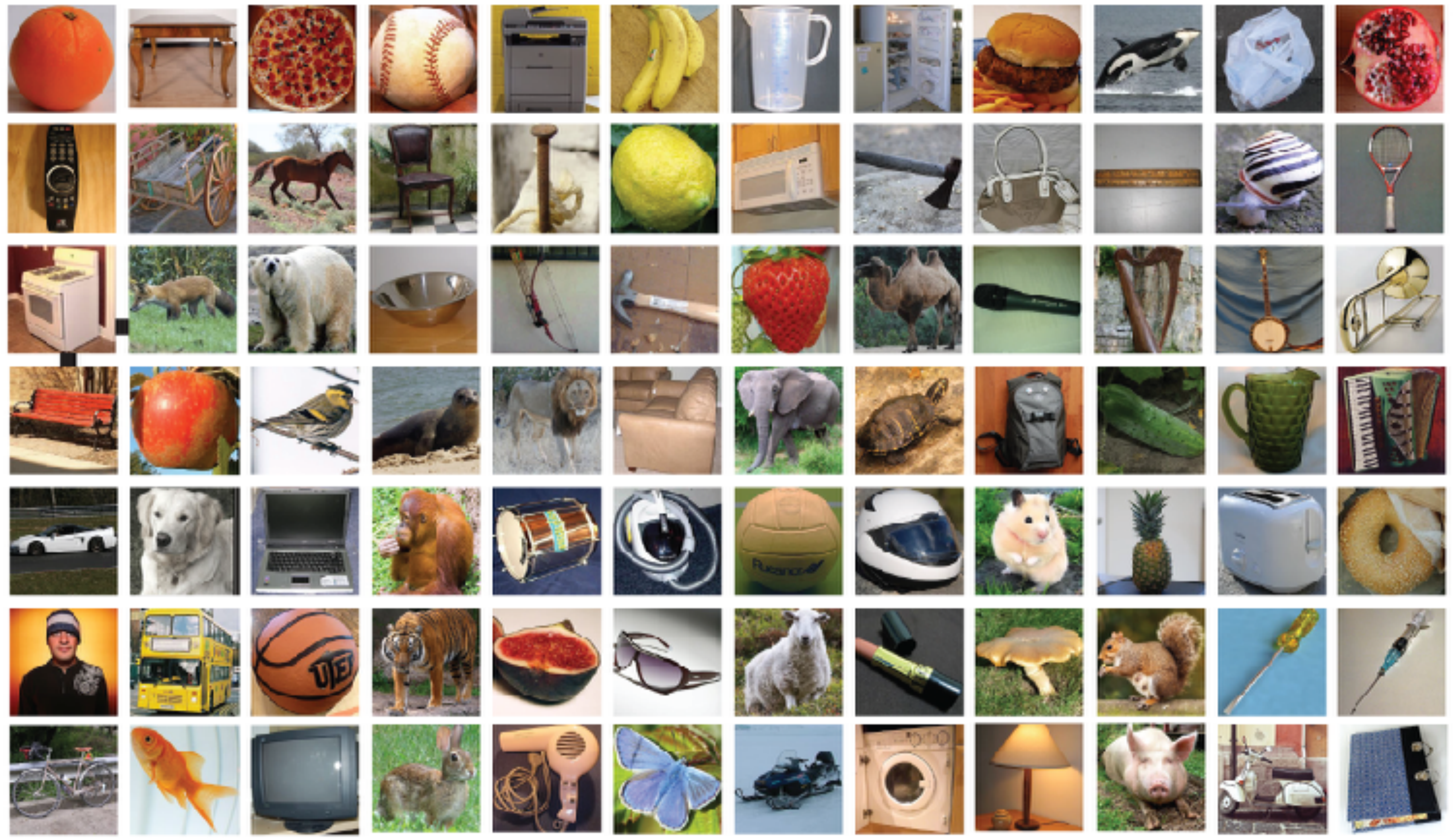


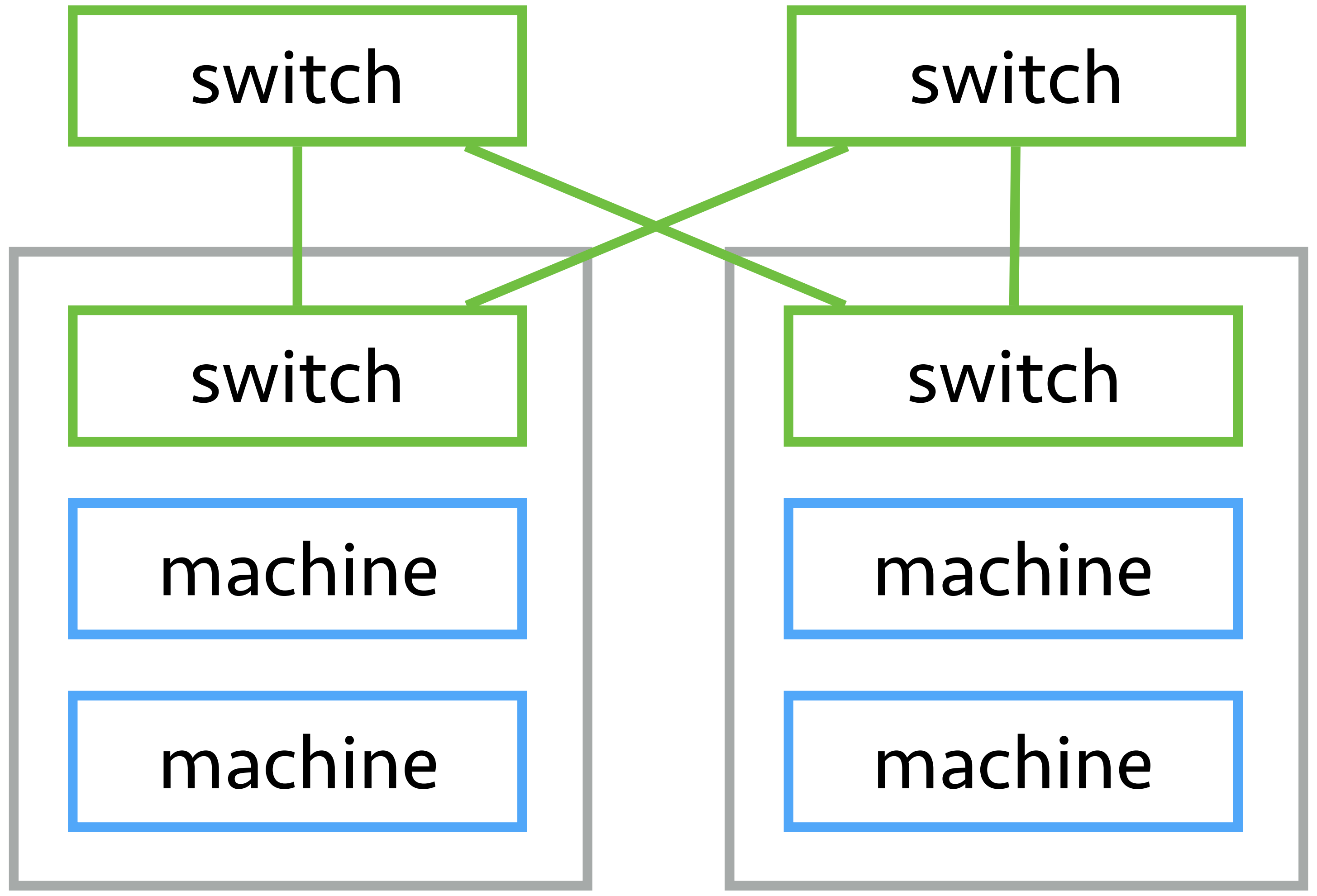
Image Recognition

- Recognize the object in an image
- Convolutional neural network
- A state-of-the-art network
  - Hundreds of layers
  - Billions of floating-point operation for processing a single image



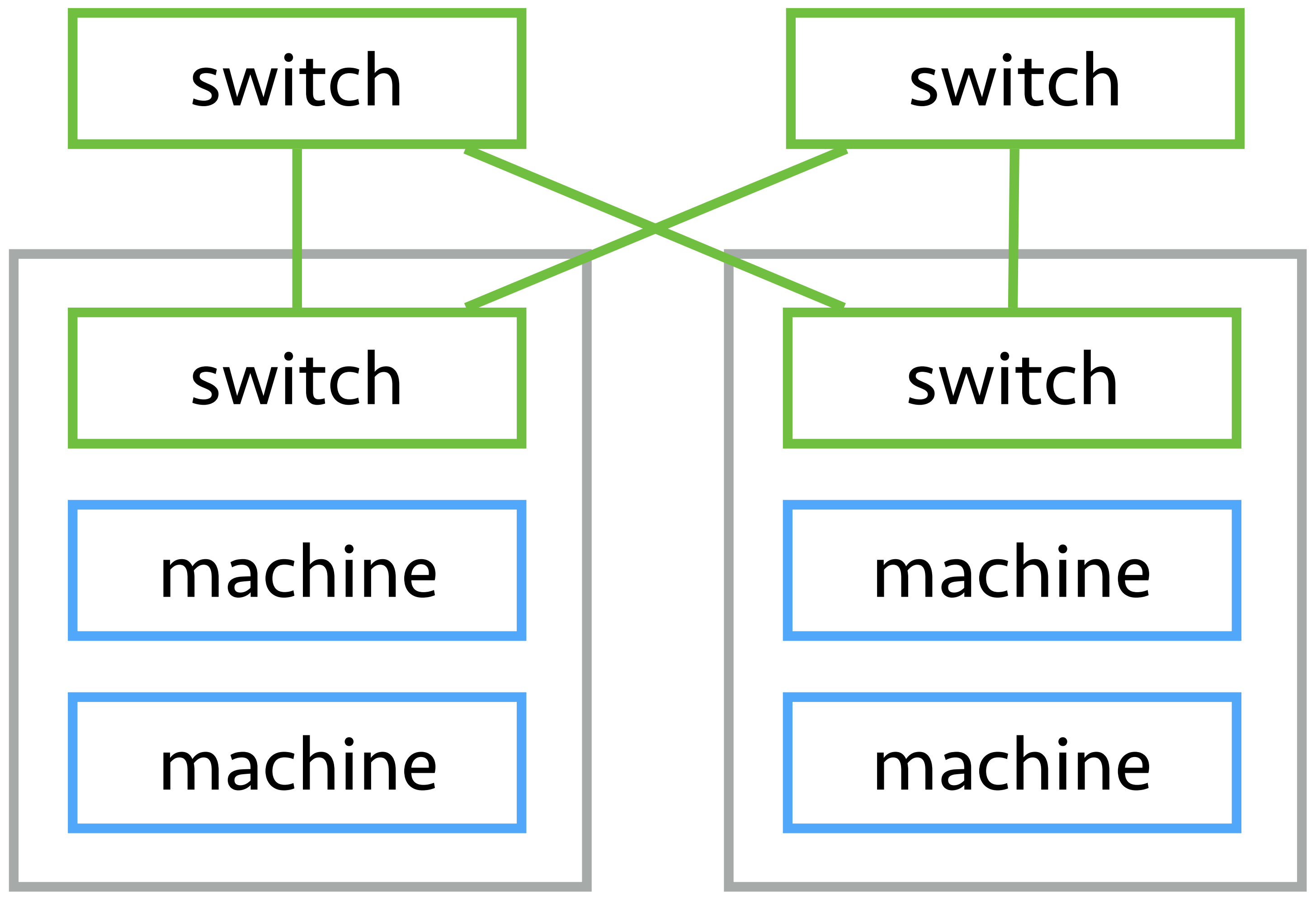
Distributed Computing for Large Scale Problems

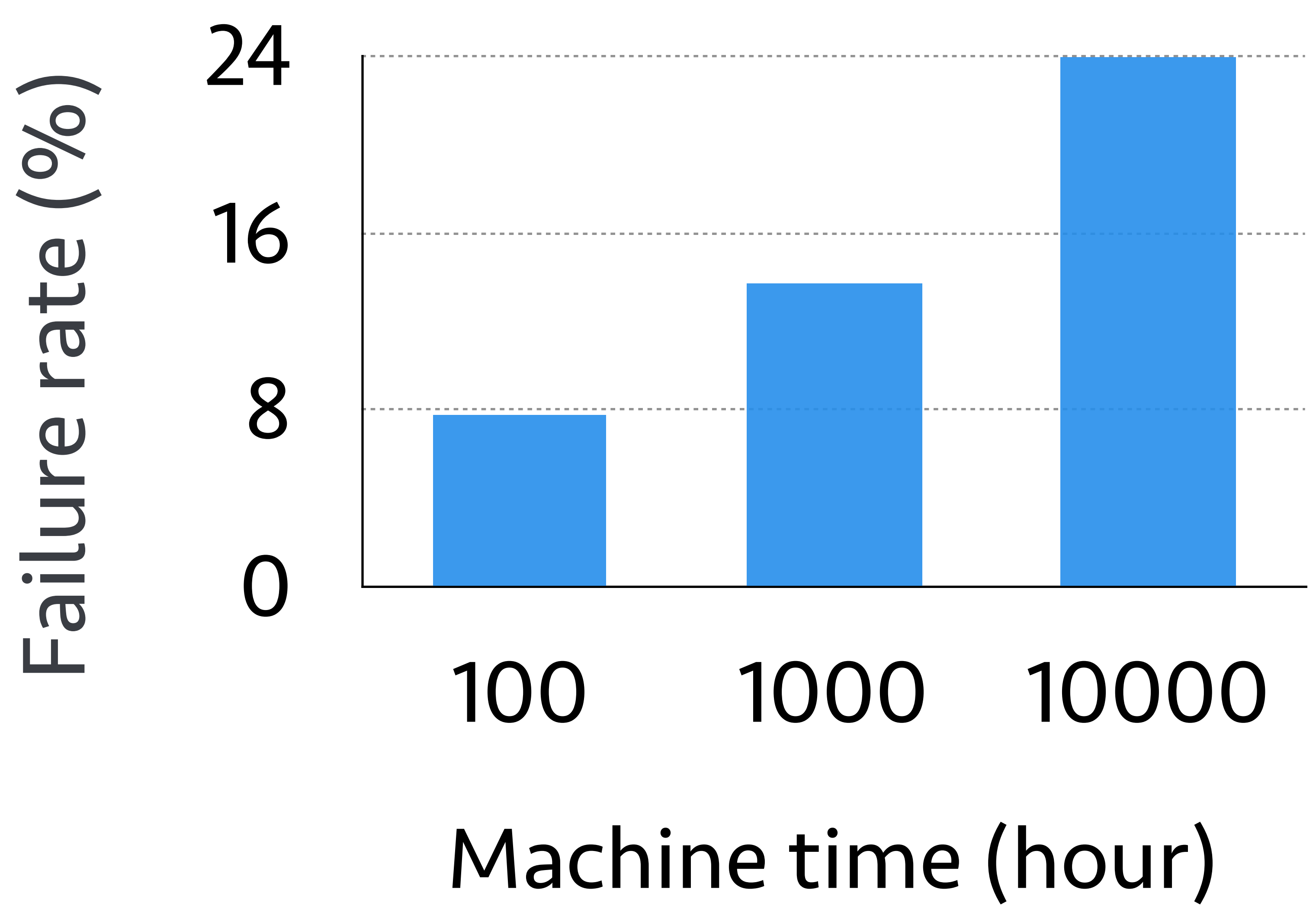
- Distribute workload among many machines
- Widely available thanks to cloud providers (AWS, GCP, Azure)



Distributed Computing for Large Scale Problems

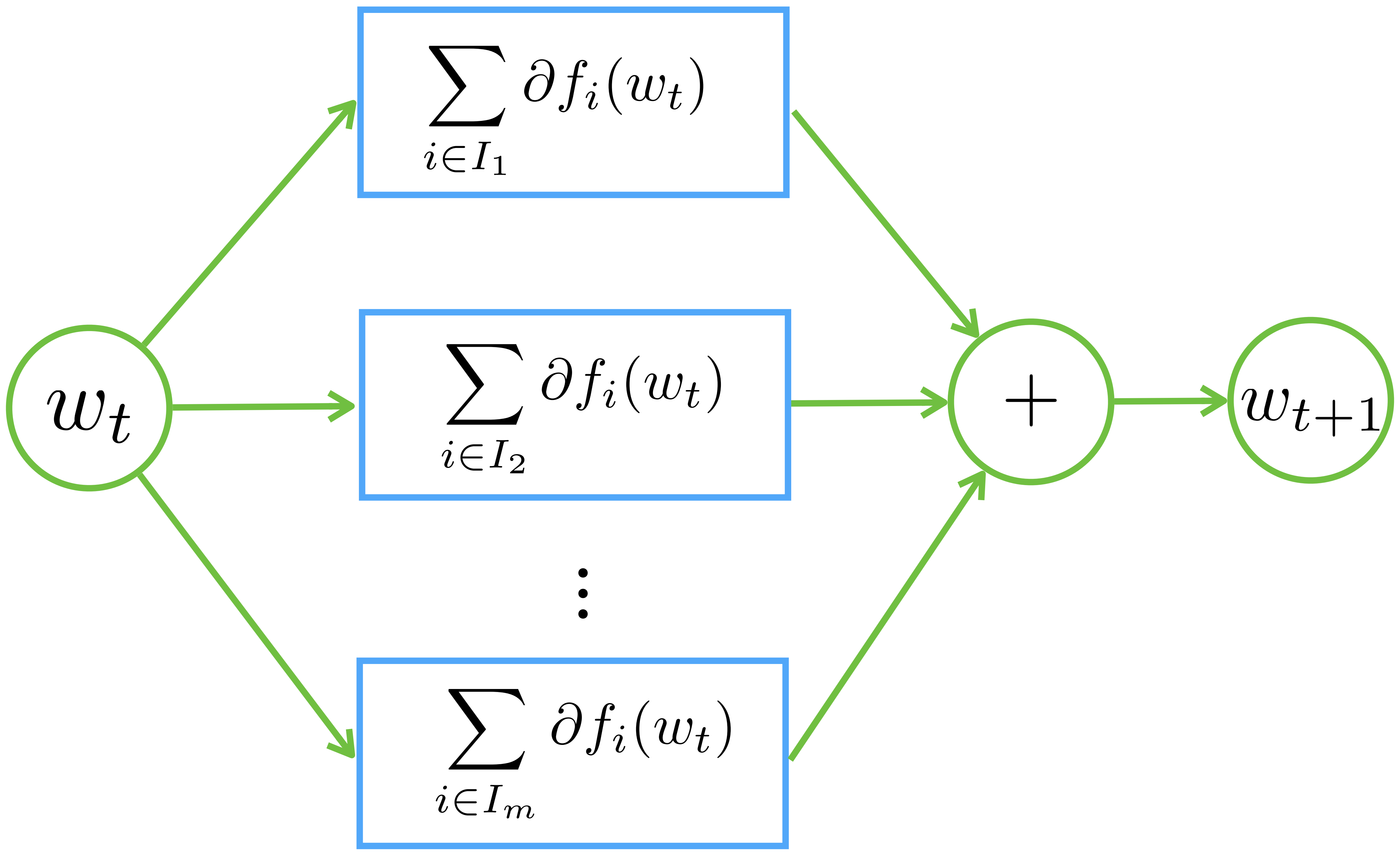
- Distribute workload among many machines
- Widely available thanks to cloud providers (AWS, GCP, Azure)
- Challenges
  - ✓ Limited communication bandwidth (10x less than memory bandwidth)
  - ✓ Large synchronization cost (1ms latency)
  - ✓ Job failures





Distributed Optimization for Large Scale ML

$$\min_w \sum_{i=1}^{n} f_i(w) \quad \Rightarrow \quad \bigcup_{i=1}^{m} I_i = \{1, 2, \ldots, n\}$$



- Challenges
  - ✓ Massive communication traffic
  - ✓ Expensive global synchronization

Scaling Distributed Machine Learning

Distributed Systems

- ✓ Large data size, complex models
- ✓ Fault tolerant
- ✓ Easy to use

Large Scale Optimization

- ✓ Communication efficient
- ✓ Convergence guarantee

Scaling Distributed Machine Learning

Distributed Systems

Parameter Server
for machine learning

MXNet
for deep learning

Large Scale Optimization

DBPG
for non-convex non-smooth f_i

EMSO
for efficient minibatch SGD

Scaling Distributed Machine Learning

Distributed Systems

Parameter Server
for machine learning

MXNet
for deep learning

Large Scale Optimization

DBPG
for non-convex non-smooth f_i

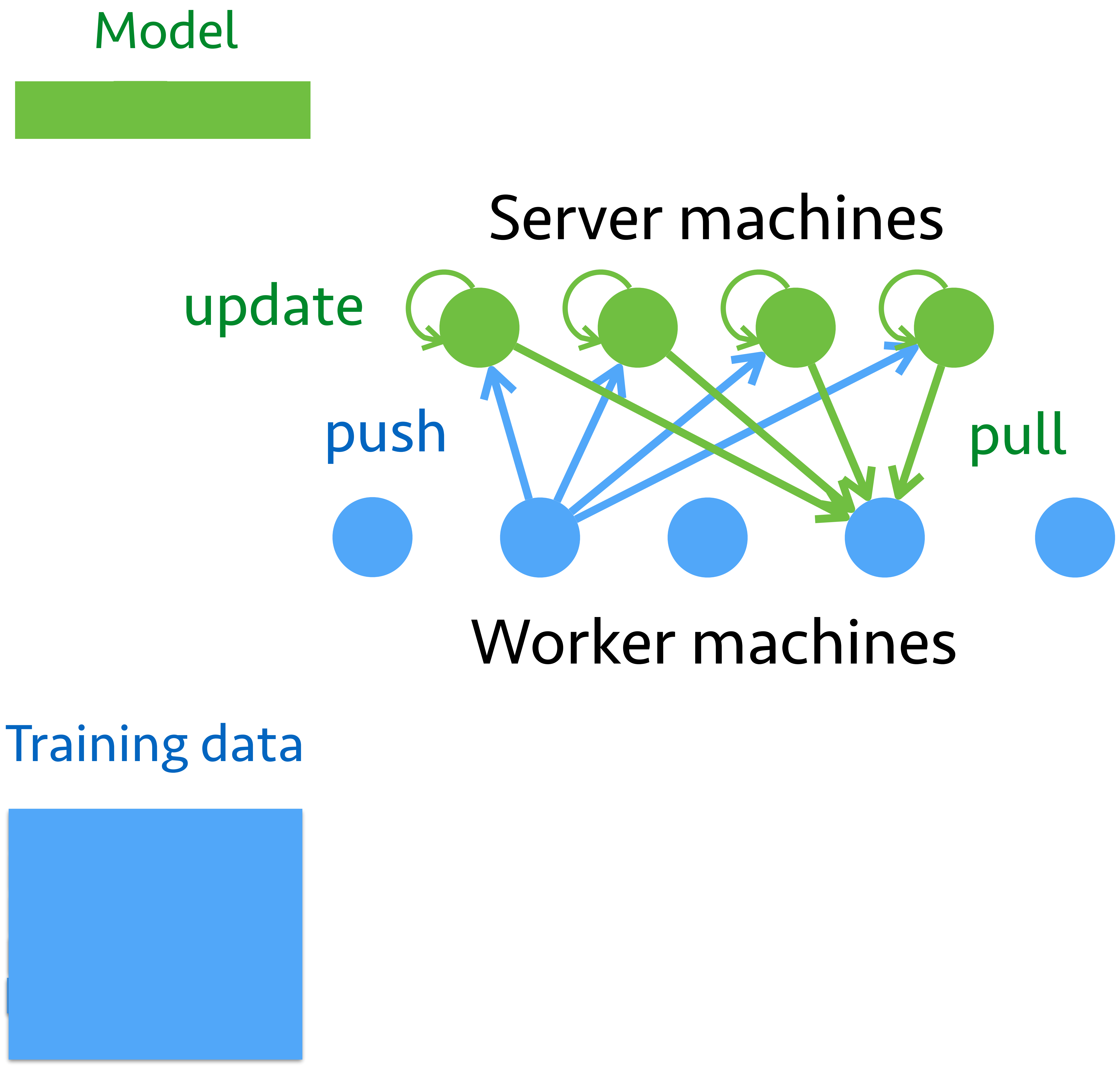
EMSO
for efficient minibatch SGD

Existing Open Source Systems in 2012

- MPI (message passing interface)
  - Hard to use for sparse problems
  - No fault tolerance
- Key-value store, e.g. redis
  - Expensive individual key-value pair communication
  - Difficult to program on the server side
- Hadoop/Spark
  - BSP data consistency makes efficient implementation challenging

Parameter Server Architecture

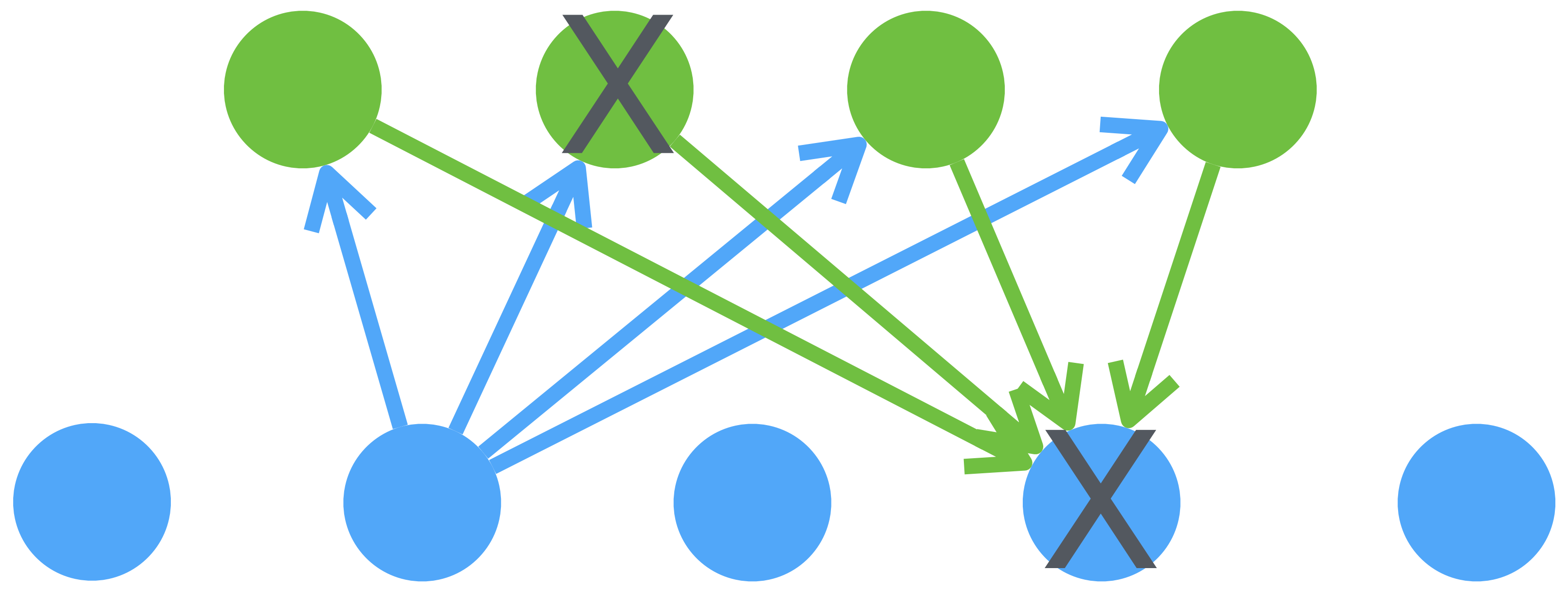
[Smola'10, Dean'12]



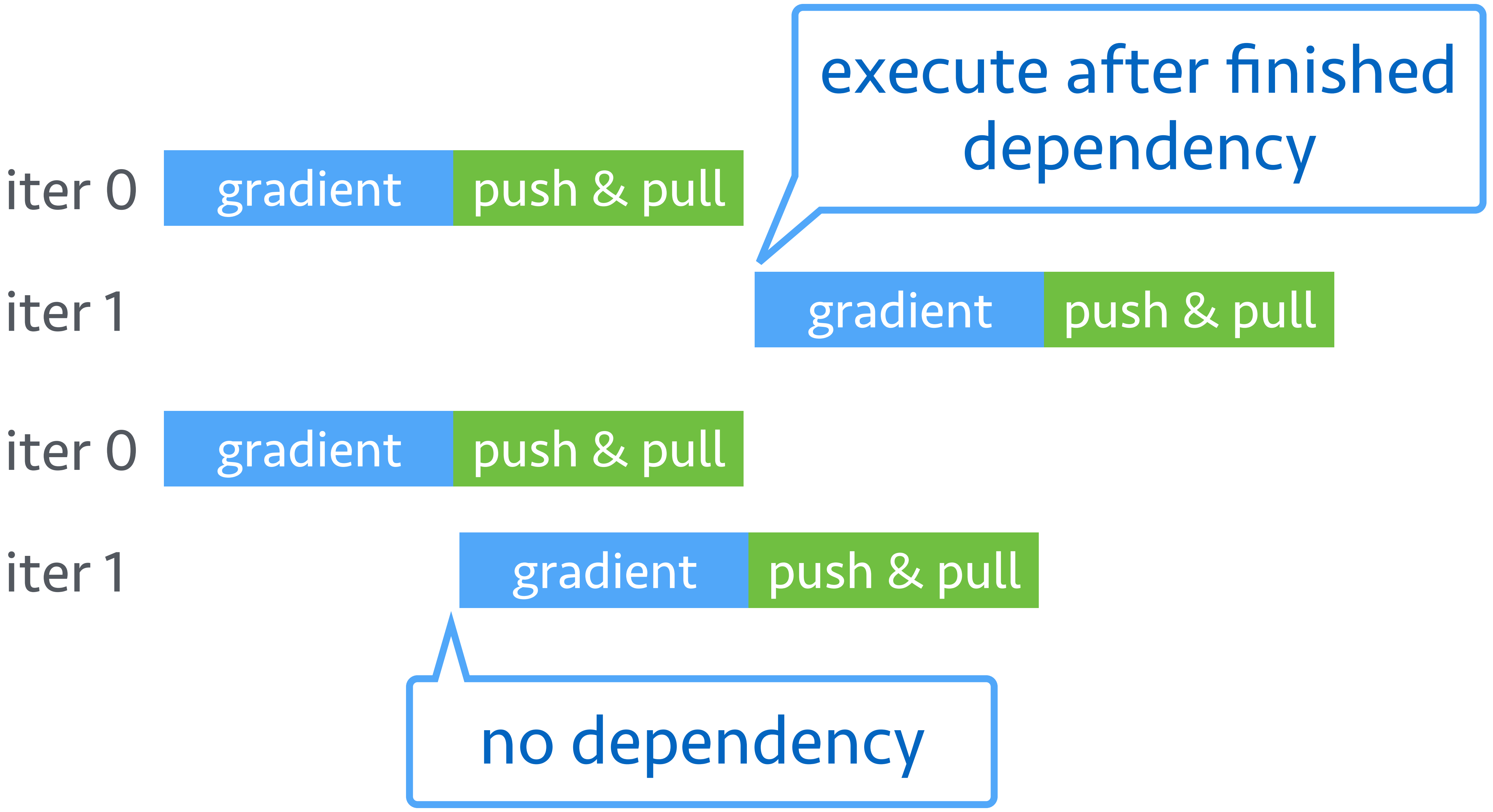
Keys Features of our Implementation

[Li et al, OSDI'14]

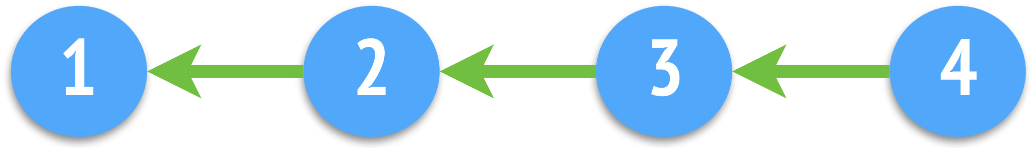
- ✧ Trade off data consistency for speed
  - ✓ Flexible consistency models
  - ✓ User-defined filters
- ✧ Fault tolerance with chain replication



Flexible Consistency Model

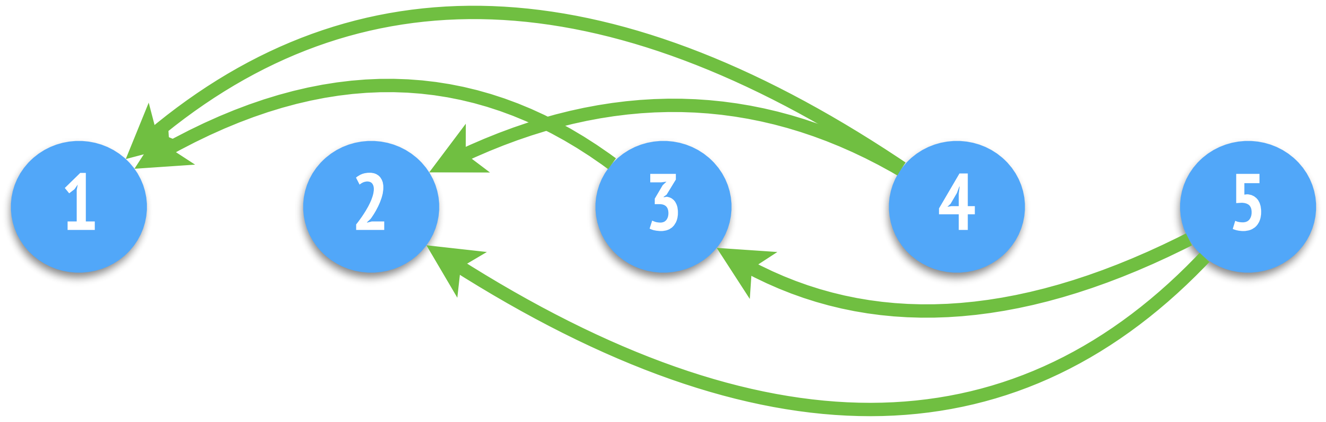


Sequential / BSP

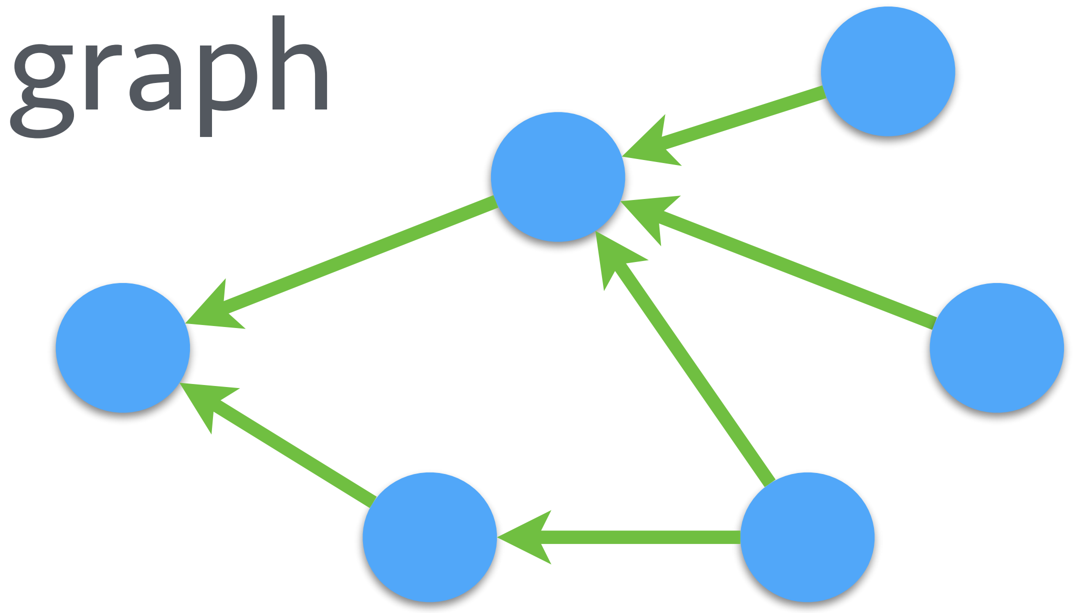


Eventual /
Total asynchronous
[Smola 10]

Bounded delay / SSP
[Langford 09, Cipar 13]

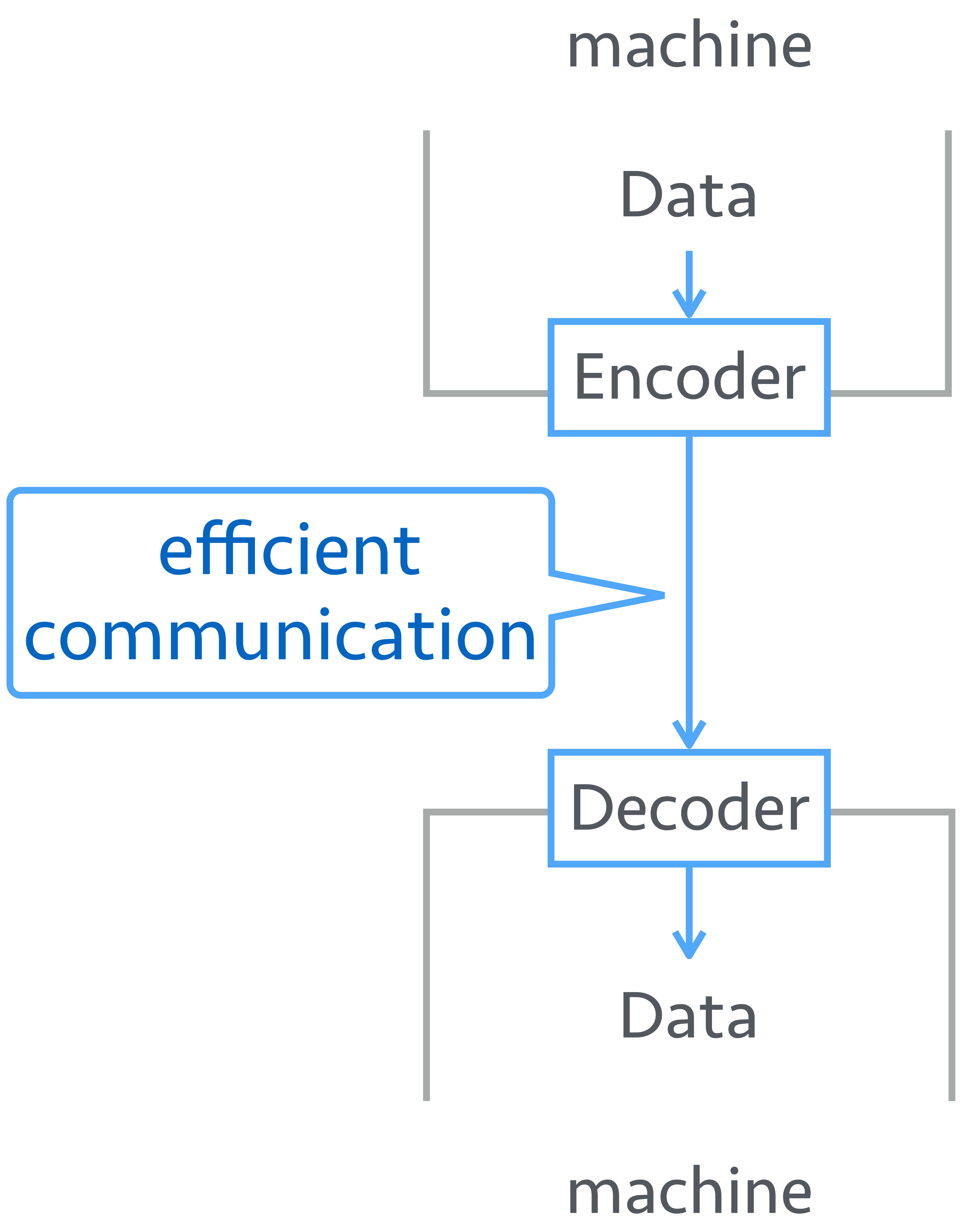


Flexible models via
task dependency graph



User-defined Filters

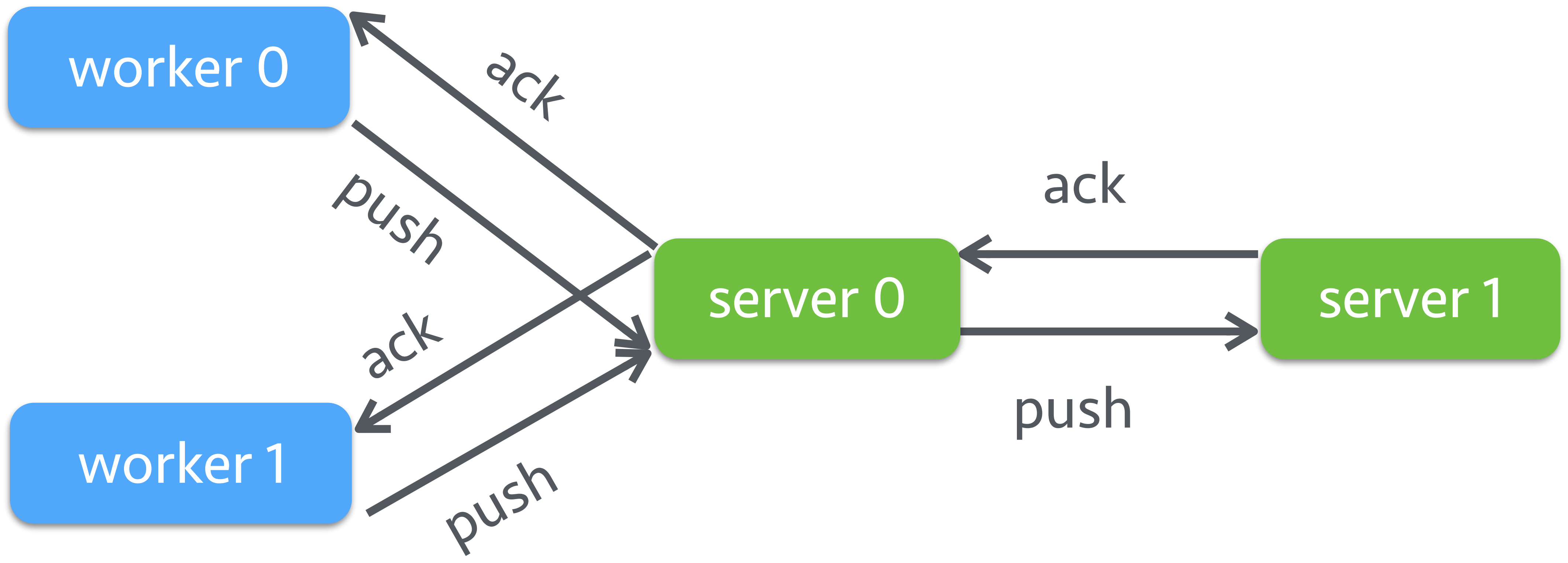
- User defined encoder/decoder for efficient communication
- Lossless compression
  - General data compression: LZ, LZR, ..
- Lossy compression
  - Random skip
  - Fixed-point encoding



Fault Tolerance with Chain Replication

- Model is partitioned by consistent hashing
- Chain replication

- Option: aggregation reduces backup traffic



Scaling Distributed Machine Learning

Distributed Systems

Parameter Server
for machine learning

MXNet
for deep learning

Large Scale Optimization

DBPG
for non-convex non-smooth f_i

EMSO
for efficient minibatch SGD

Proximal Gradient Method

[Combettes'09]

$$\min_{w\in\Omega} \sum_{i=1}^{n} f_i(w) + h(w)$$

- ✓ f_i: continuously differentiable but not necessarily convex
- ✓ h: convex but possibly non-smooth

✧ Iterative update

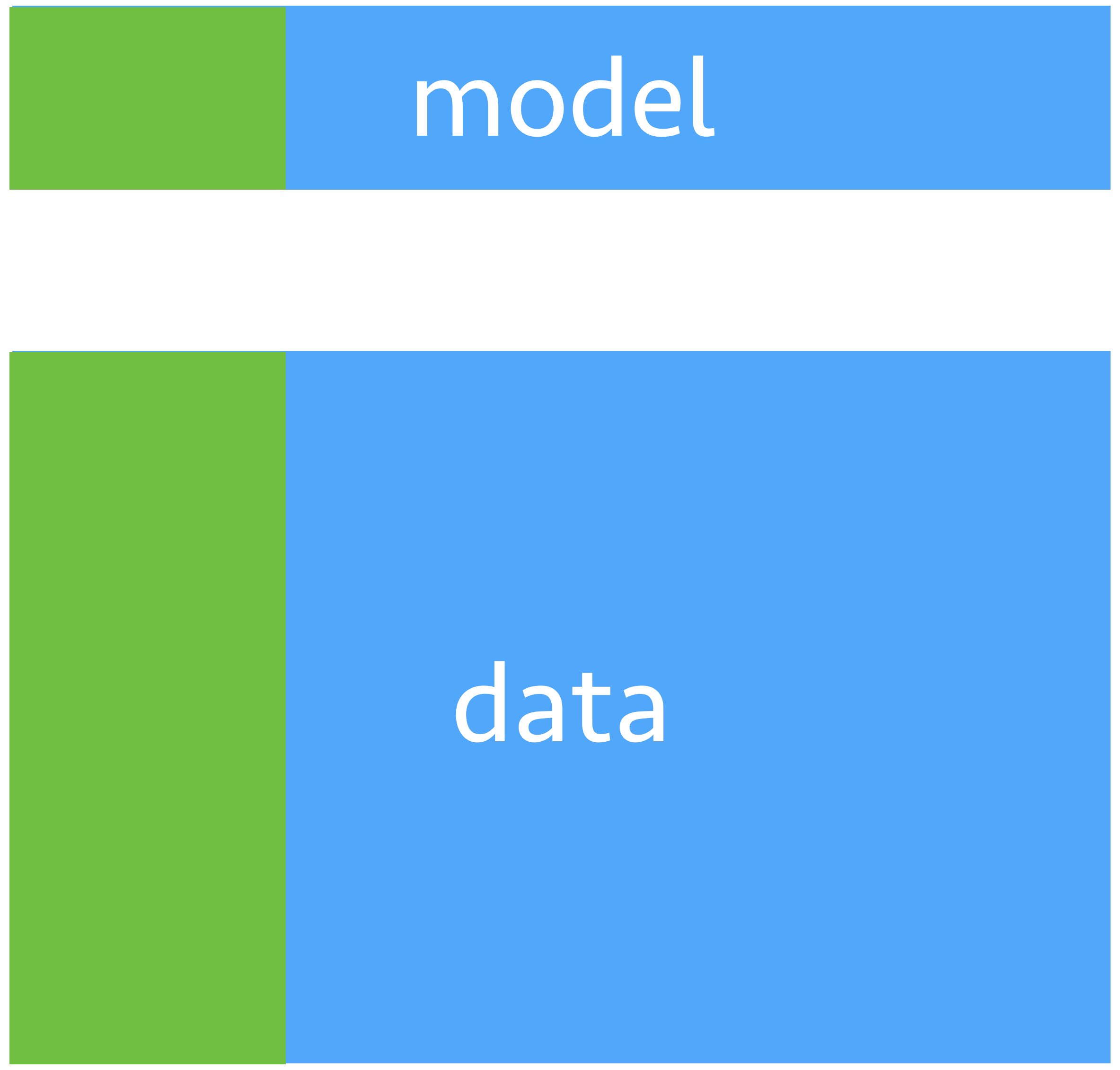
$$w_{t+1} = \mathrm{Prox}_{\gamma_t}\left[w_t - \eta_t \sum_{i=1}^{n} f_i(w_t)\right]$$

where $\mathrm{Prox}_\eta(x) := \operatorname*{argmin}_{y\in\Omega} h(y) + \frac{1}{2\eta}\|x-y\|^2$

Delayed Block Proximal Gradient

[Li et al, NIPS'14]

- Algorithm design tailored for parameter server implementation
  - Update a block of coordinates each time
  - Allow delay among blocks
  - Use filters during communication
- Only 300 lines of codes



Convergence Analysis

- Assumptions:
  - Block Lipschitz continuity: within block L_{var}, cross blocks L_{cor}
  - Delay is bounded by τ
  - Lossy compressions such as random skip filter and significantly-modified filter
- DBPG converges to a stationary point if the learning rate is chosen as

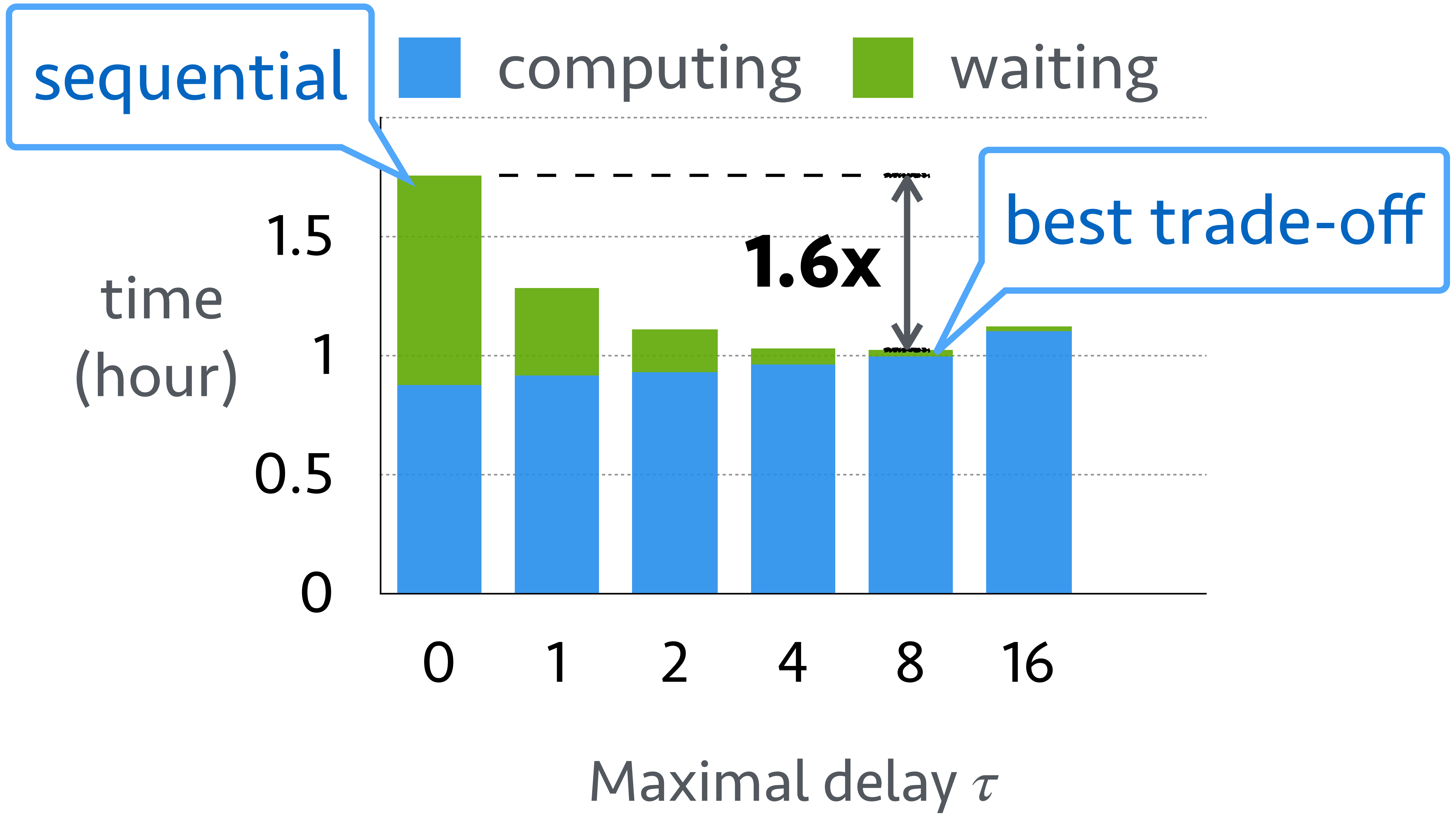
$$\eta_t < \frac{1}{L_{\text{var}} + \tau L_{\text{cor}}}$$

Experiments on Ads Click Prediction

- Real dataset used in production
  - 170 billion examples, 65 billion unique features, 636 TB in total
- 1000 machines
- Sparse logistic regression

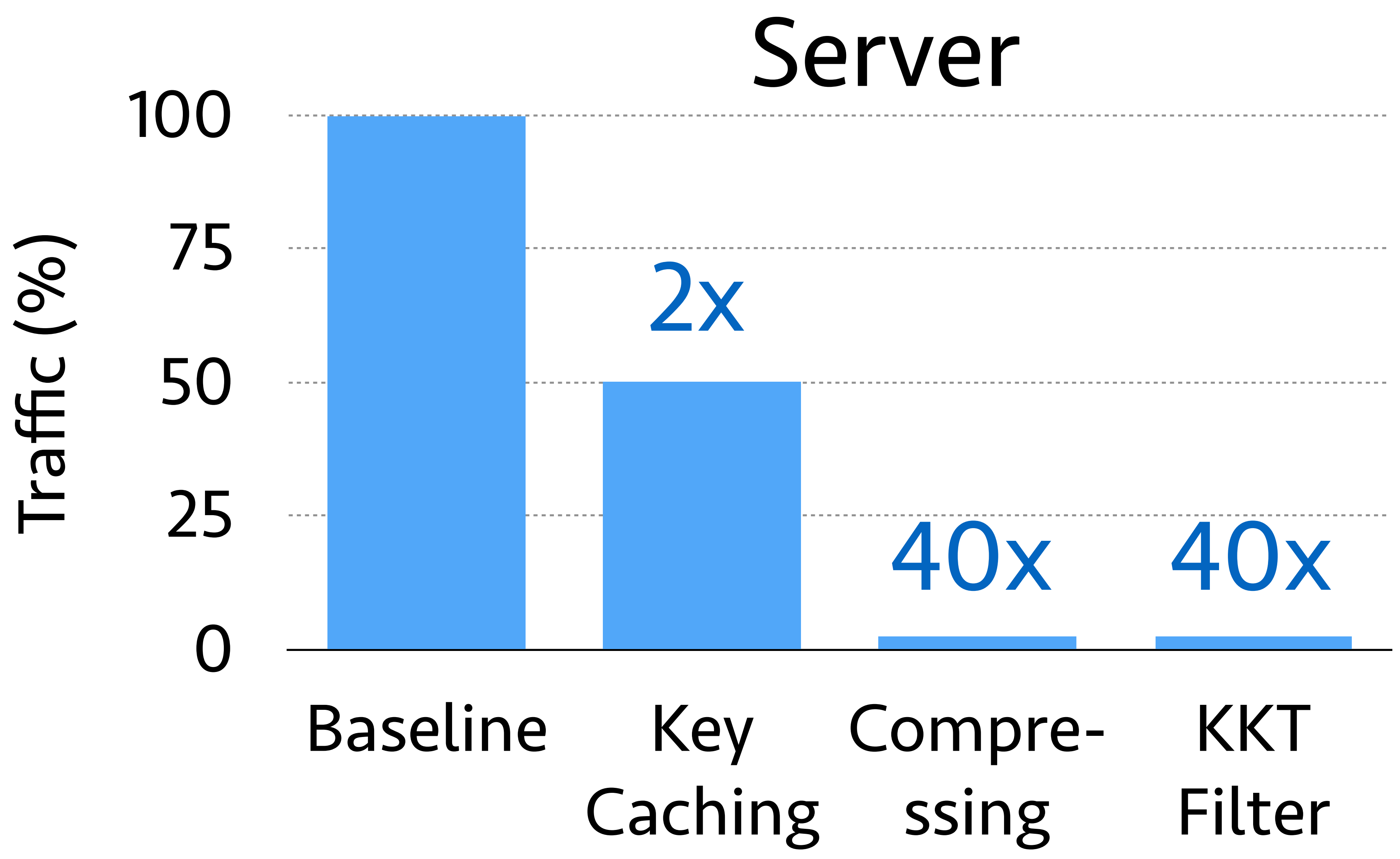
$$\min_{w} \sum_{i=1}^{n} \log(1 + \exp(-y_i \langle x_i, w \rangle)) + \lambda \|w\|_1$$

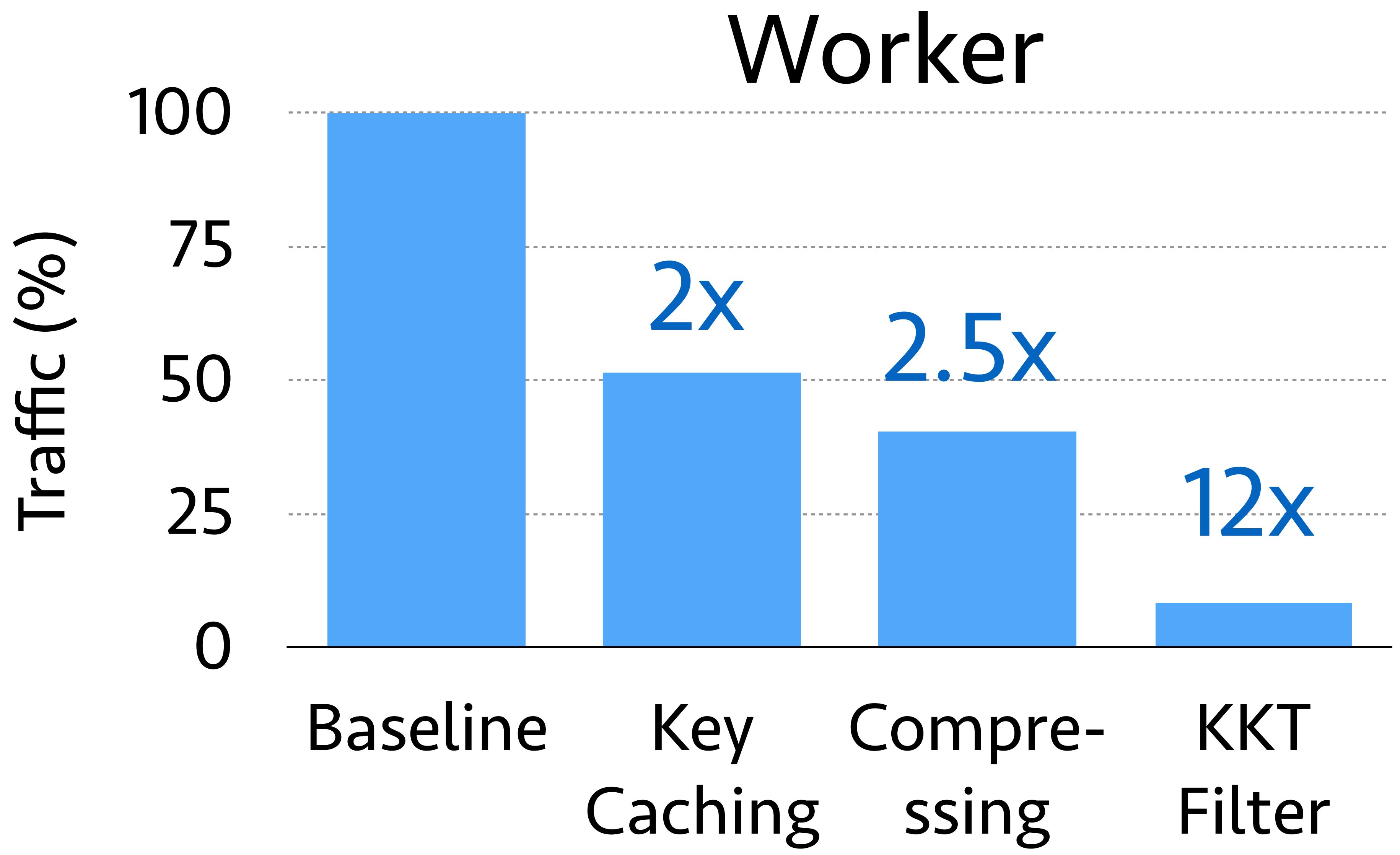
Time to achieve the same objective value



Filters to Reduce Communication Traffic

- Key caching
  - Cache feature IDs on both sender and receiver
- Data compression
- KKT filter
  - Shrink gradient to 0 based on the KKT condition





Scaling Distributed Machine Learning

Distributed Systems

Parameter Server
for machine learning

MXNet
for deep learning

Large Scale Optimization

DBPG
for non-convex non-smooth f_i

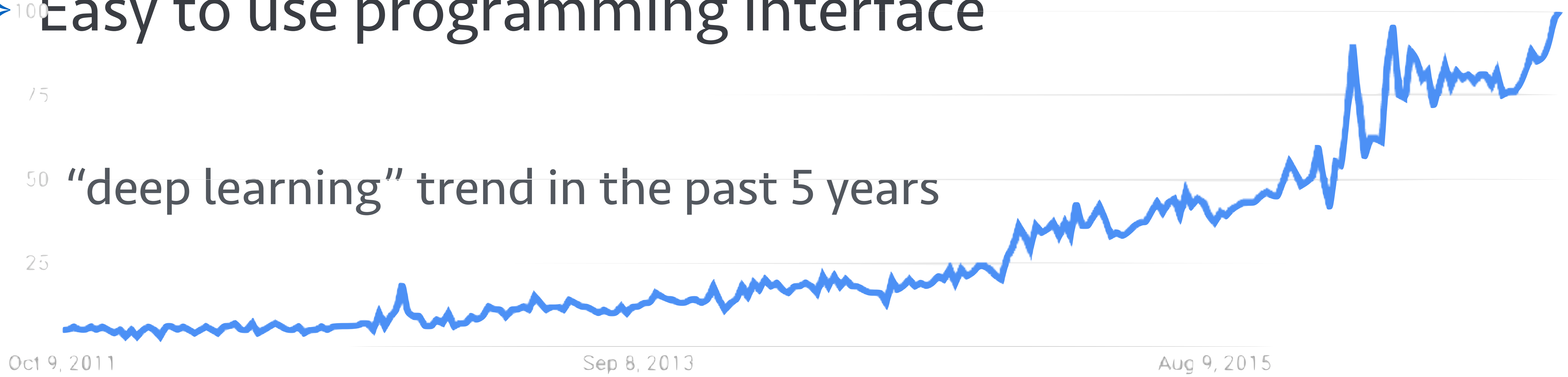
EMSO
for efficient minibatch SGD

Deep Learning is Unique

- Complex workloads

- Heterogeneous computing

- Easy to use programming interface



"deep learning" trend in the past 5 years

Key Features of MXNet

[Chen et al, NIPS'15 workshop]
(corresponding author)

- Easy-to-use front-end
  - ✓ Mixed programming
- Scalable and efficient back-end
  - ✓ Computation and memory optimization
  - ✓ Auto-parallelization
  - ✓ Scaling to multiple GPU/machines

Mixed Programming

- Declarative programs are easy to optimize
  - e.g. TensorFlow, Theano, Caffe, …

```
import mxnet as mx
net = mx.symbol.Variable('data')
net = mx.symbol.FullyConnected(
          data=net, num_hidden=128)
net = mx.symbol.SoftmaxOutput(data=net)
model = mx.module.Module(net)
model.forward(data=c)
model.backward()
```

Good for defining the neural network

- Imperative programming is flexible
  - e.g. Numpy, Matlab, Torch, …

```
import mxnet as mx
a = mx.nd.zeros((100, 50))
b = mx.nd.ones((100, 50))
c = a * b
print(c)
c += 1
```

Good for updating and interacting with the neural network

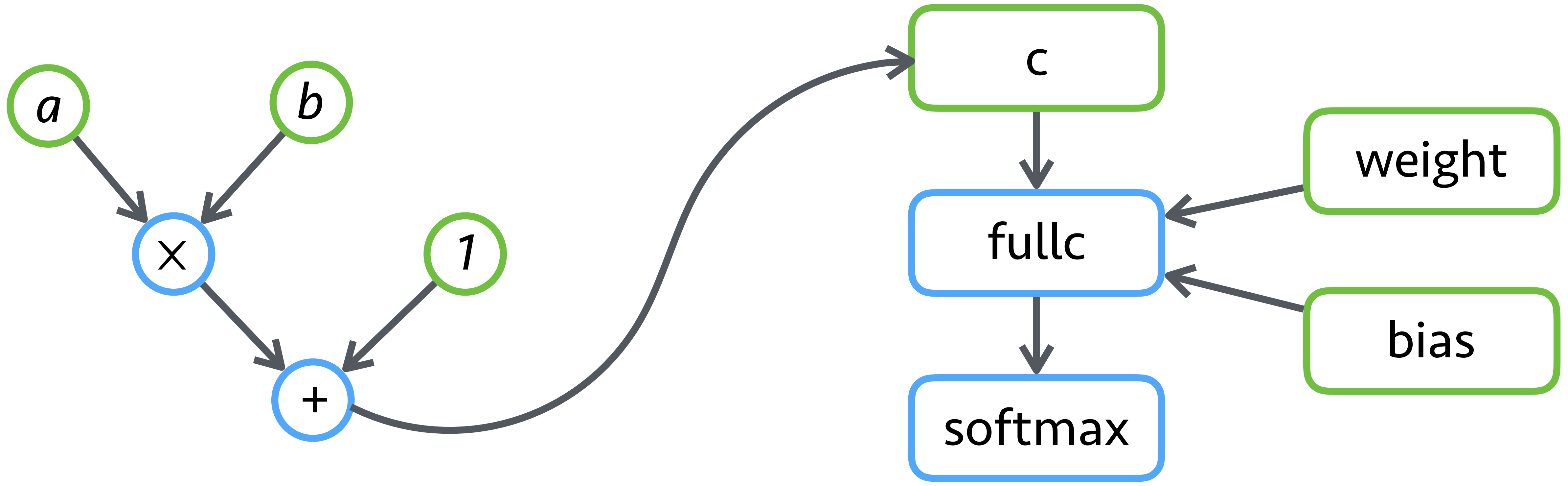
Back-end System

```python
import mxnet as mx
a = mx.nd.zeros((100, 50))
b = mx.nd.ones((100, 50))
c = a * b
c += 1
```

```python
import mxnet as mx
net = mx.symbol.Variable('data')
net = mx.symbol.FullyConnected(
        data=net, num_hidden=128)
net = mx.symbol.SoftmaxOutput(data=net)
texec = mx.module.Module(net)
texec.forward(data=c)
texec.backward()
```

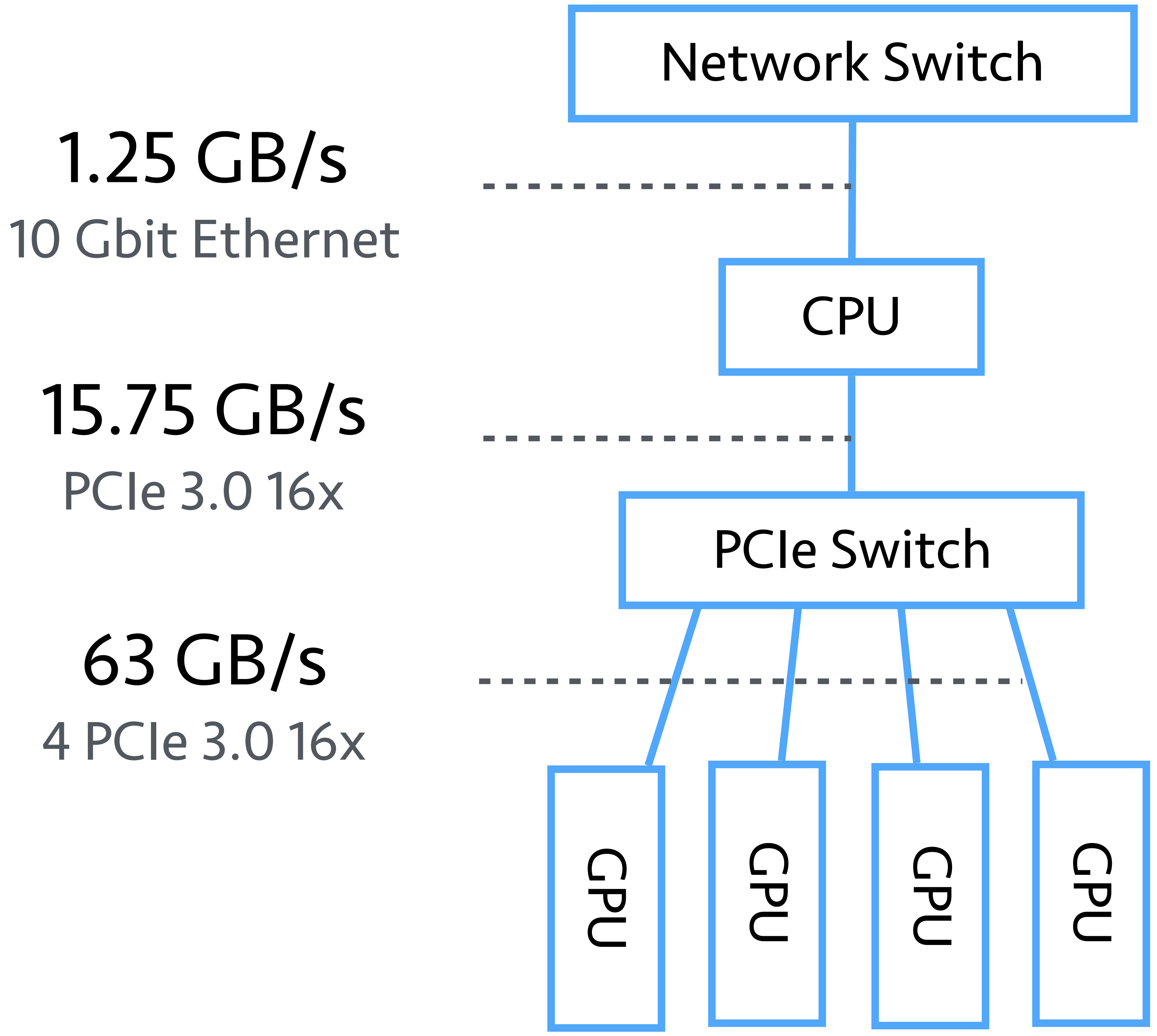
Front-end

Back-end

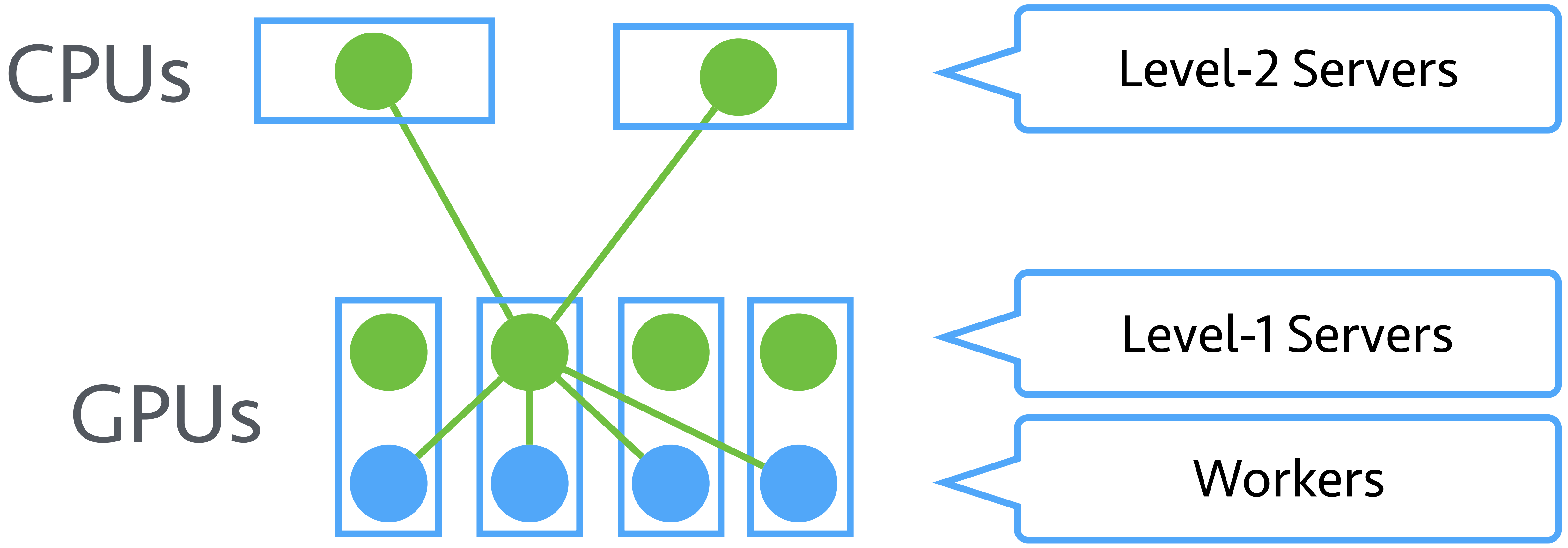


- Optimization
  - ✓ Memory optimization
  - ✓ Operator fusion and runtime compilation
- Scheduling
  - ✓ Auto-parallelization

Scale to Multiple GPU Machines



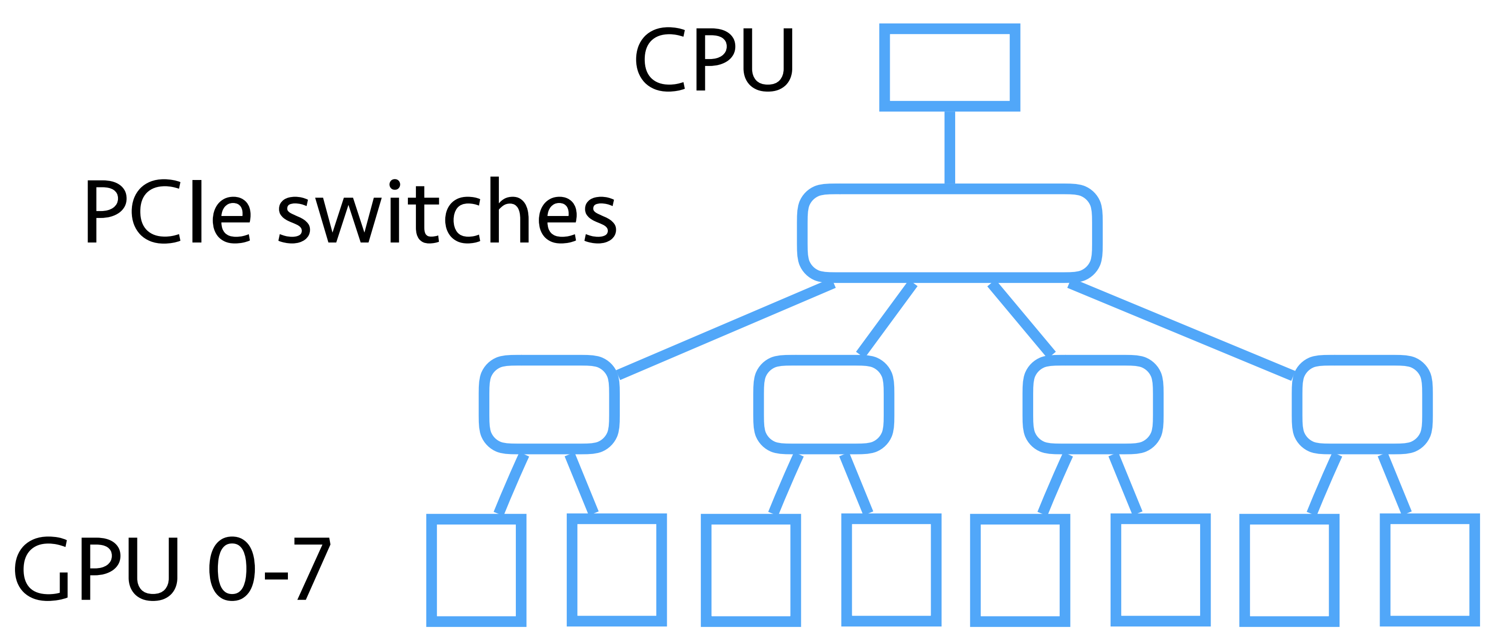
Hierarchical parameter server



- 1000 lines of codes

Experiment Setup

- IMAGENET
  - 1.2 million images with 1000 classes
- Resnet 152-layer model
- EC2 P2.8 xlarge
  - 8 K80 GPUs per machine



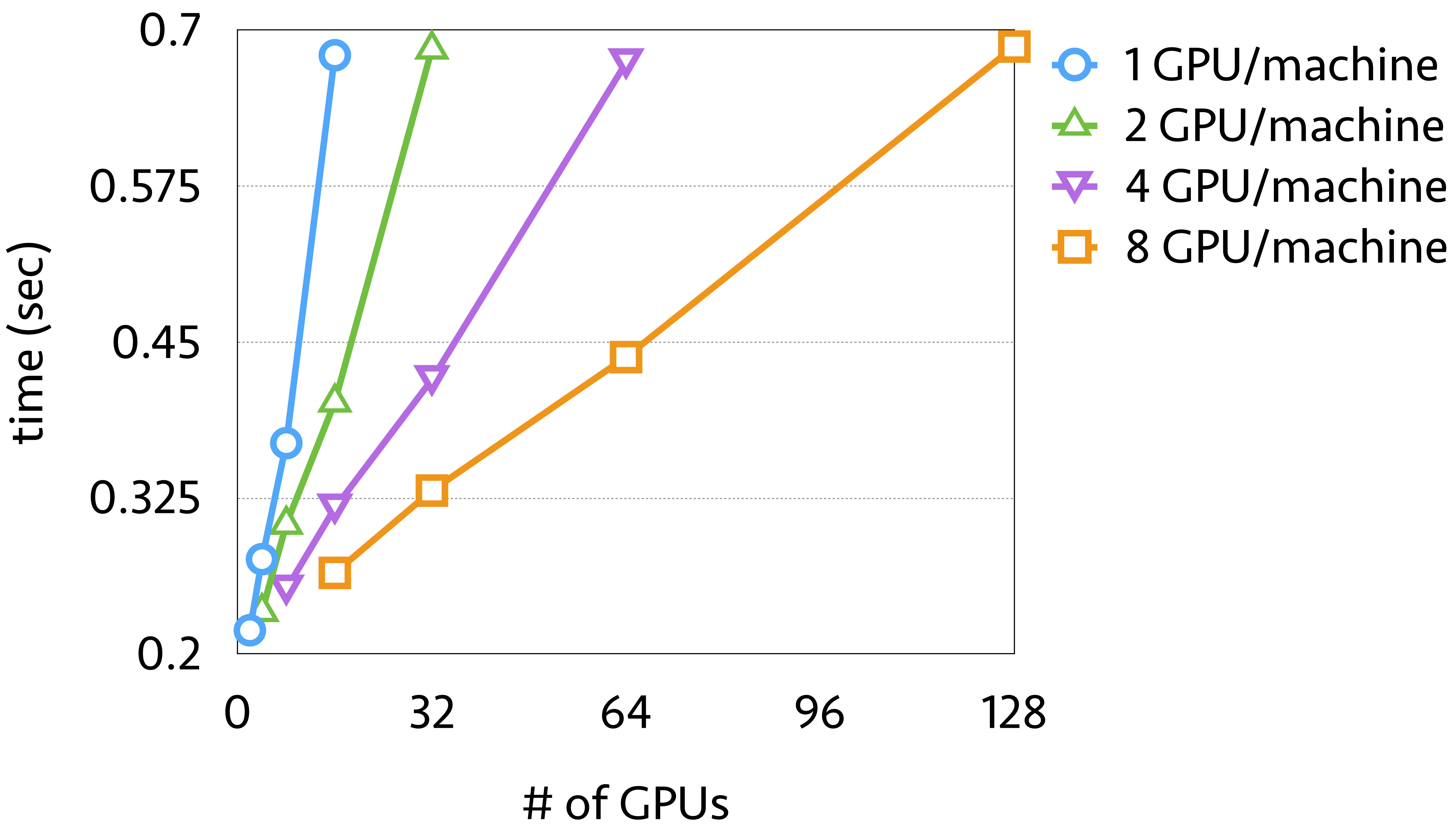
- Minibatch SGD
  - Draw a random set of examples I_t at iteration t
  - Update

$$w_{t+1} = w_t - \frac{\eta_t}{|I_t|} \sum_{i \in I_t} \partial f_i(w_t)$$

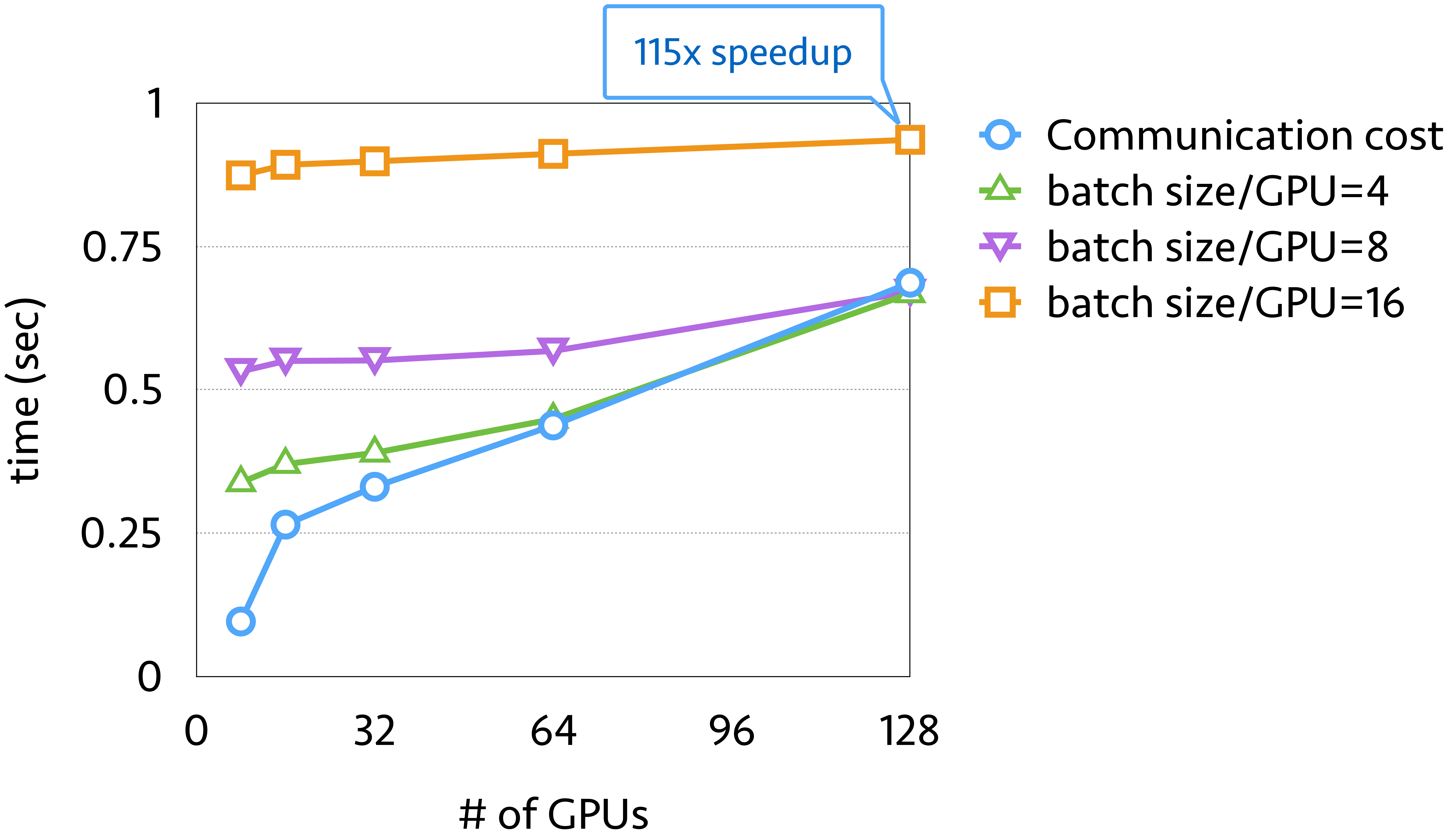
- Synchronized updating

Communication Cost

- Fix #GPUs per machine

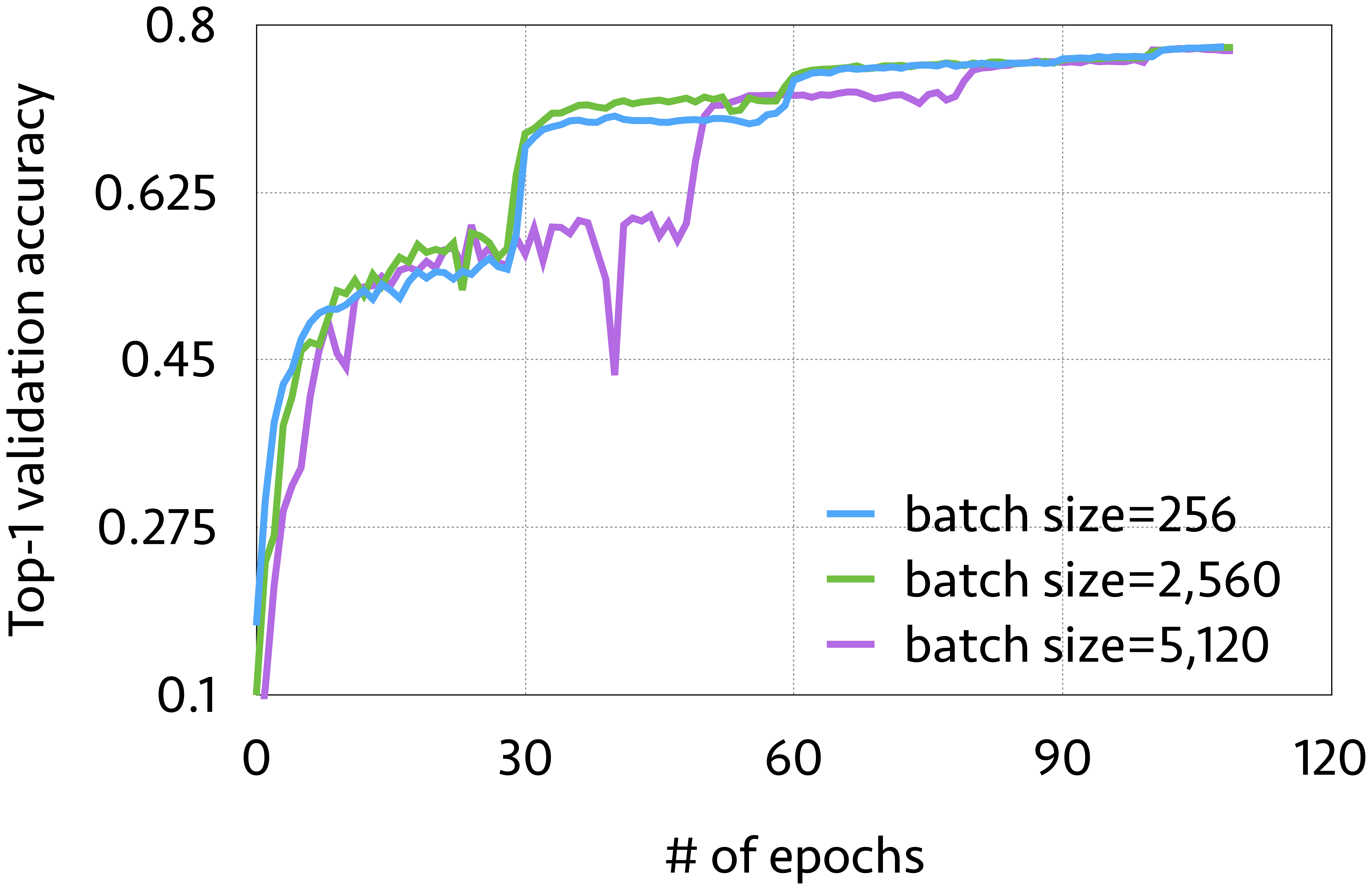


Scalability



Convergence

- Increase learning rate by 5x
- Increase learning rate by 10x, decrease it at epoch 50, 80



Scaling Distributed Machine Learning

Distributed Systems

Parameter Server
for machine learning

MXNet
for deep learning

Large Scale Optimization

DBPG
for non-convex non-smooth f_i

EMSO
for efficient minibatch SGD

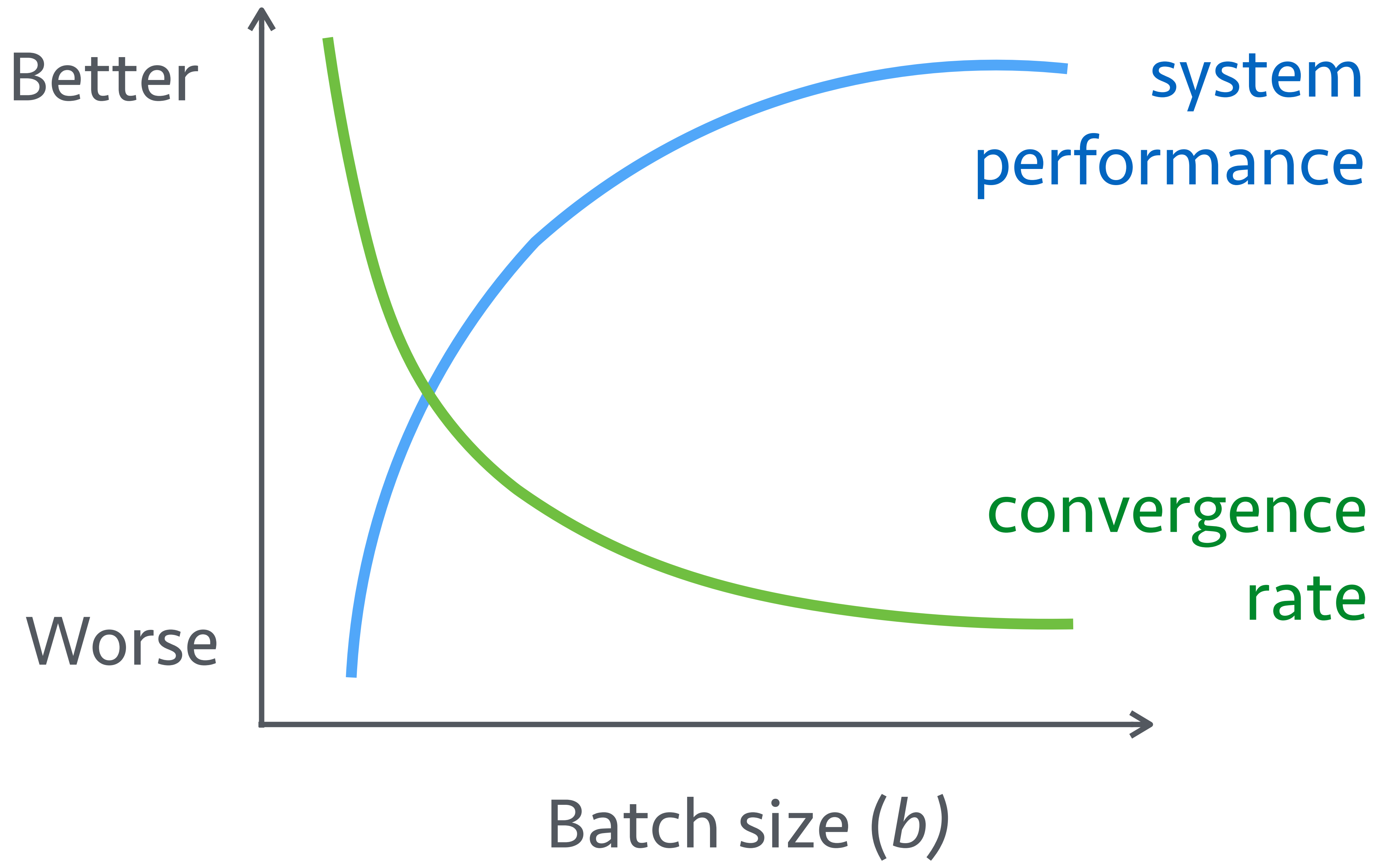
Minibatch SGD

- Large batch size *b* in SGD
  - Better parallelization within a batch
  - Less switching/communication cost

- Small batch size *b*
  - Faster convergence

$$O(1/\sqrt{N} + b/N)$$

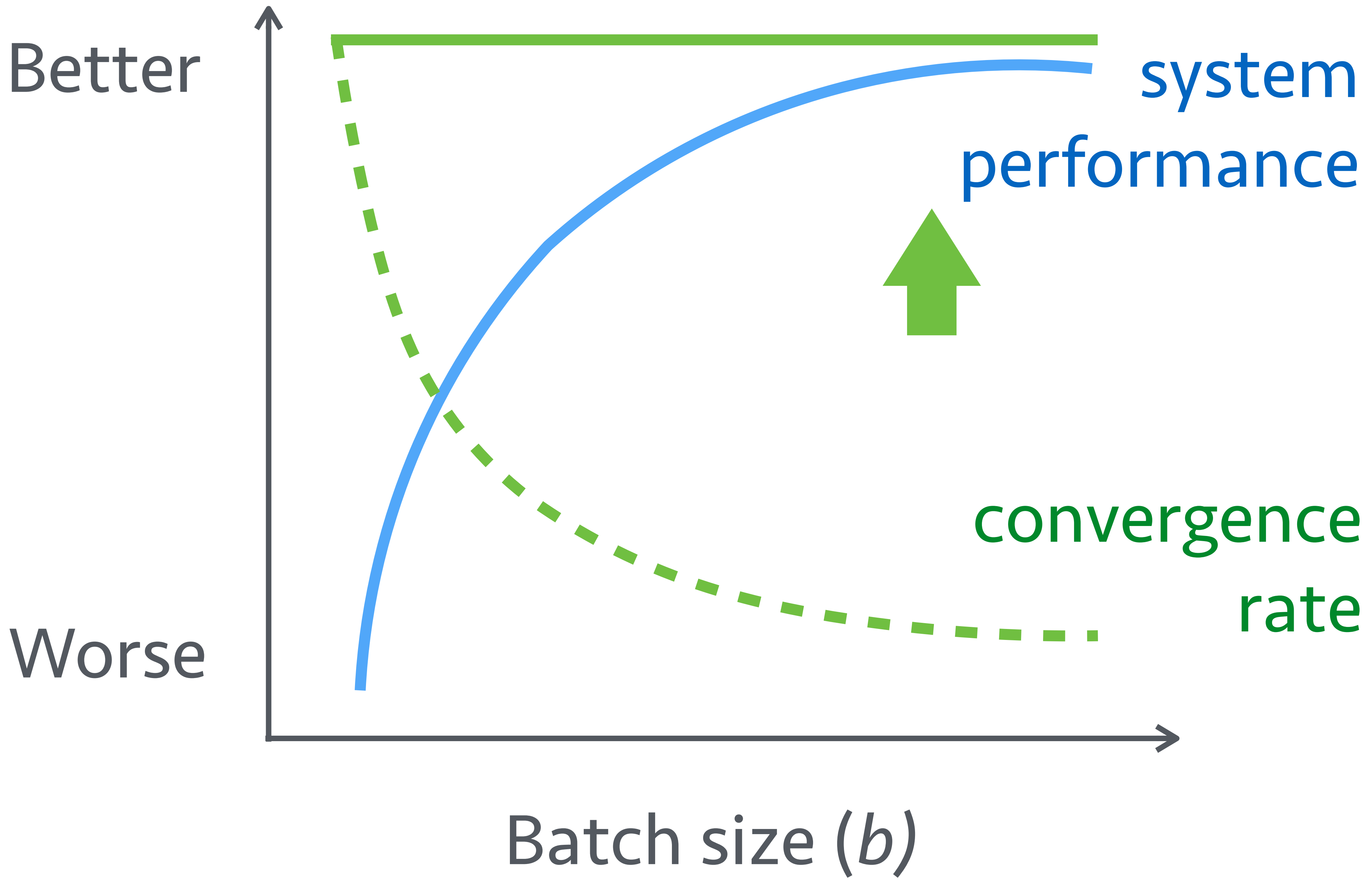
N: number of examples processed



Motivation

[Li et al, KDD'14]

- Improve converge rate for large batch size
  - ✓ Example variance decreases with batch size
  - ✓ Solve a more "accurate" optimization subproblem over each batch



Efficient Minibatch SGD

- Define $f_{I_t}(w) := \sum_{i \in I_t} f_i(w)$. Minibatch SGD solves

first-order approximation

conservative penalty

$$w_t = \operatorname*{argmin}_{w \in \Omega} \left[f_{I_t}(w_{t-1}) + \langle \partial f_{I_t}(w_{t-1}), w - w_{t-1} \rangle + \frac{1}{2\eta_t} \|w - w_{t-1}\|_2^2 \right]$$

- EMSO solves the subproblem at each iteration

$$w_t = \operatorname*{argmin}_{w \in \Omega} \left[f_{I_t}(w) + \frac{1}{2\eta_t} \|w - w_{t-1}\|_2^2 \right]$$

exact objective

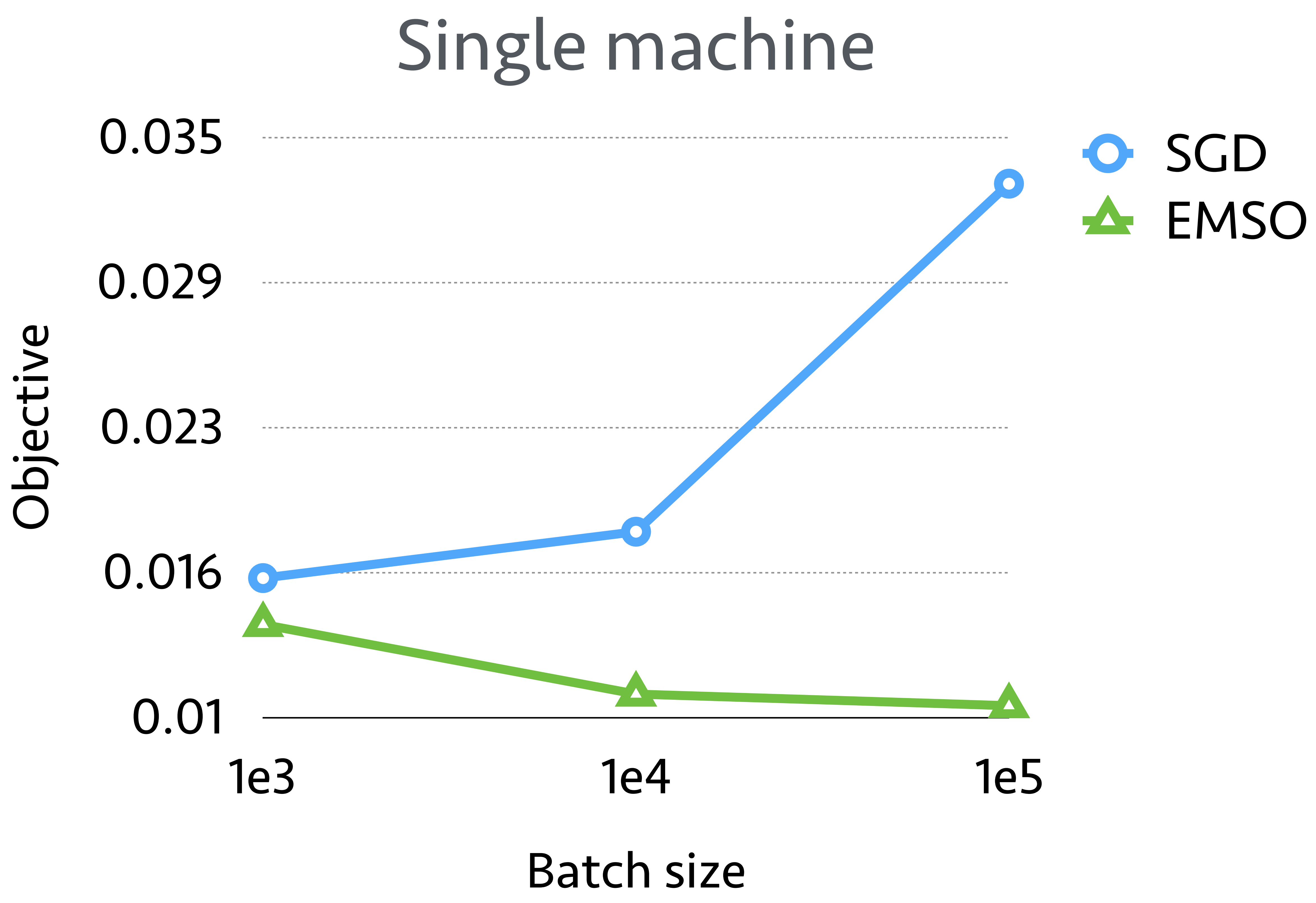
- *For convex f_i, choose $\eta_t = O(b/\sqrt{N})$. EMSO has convergence rate*

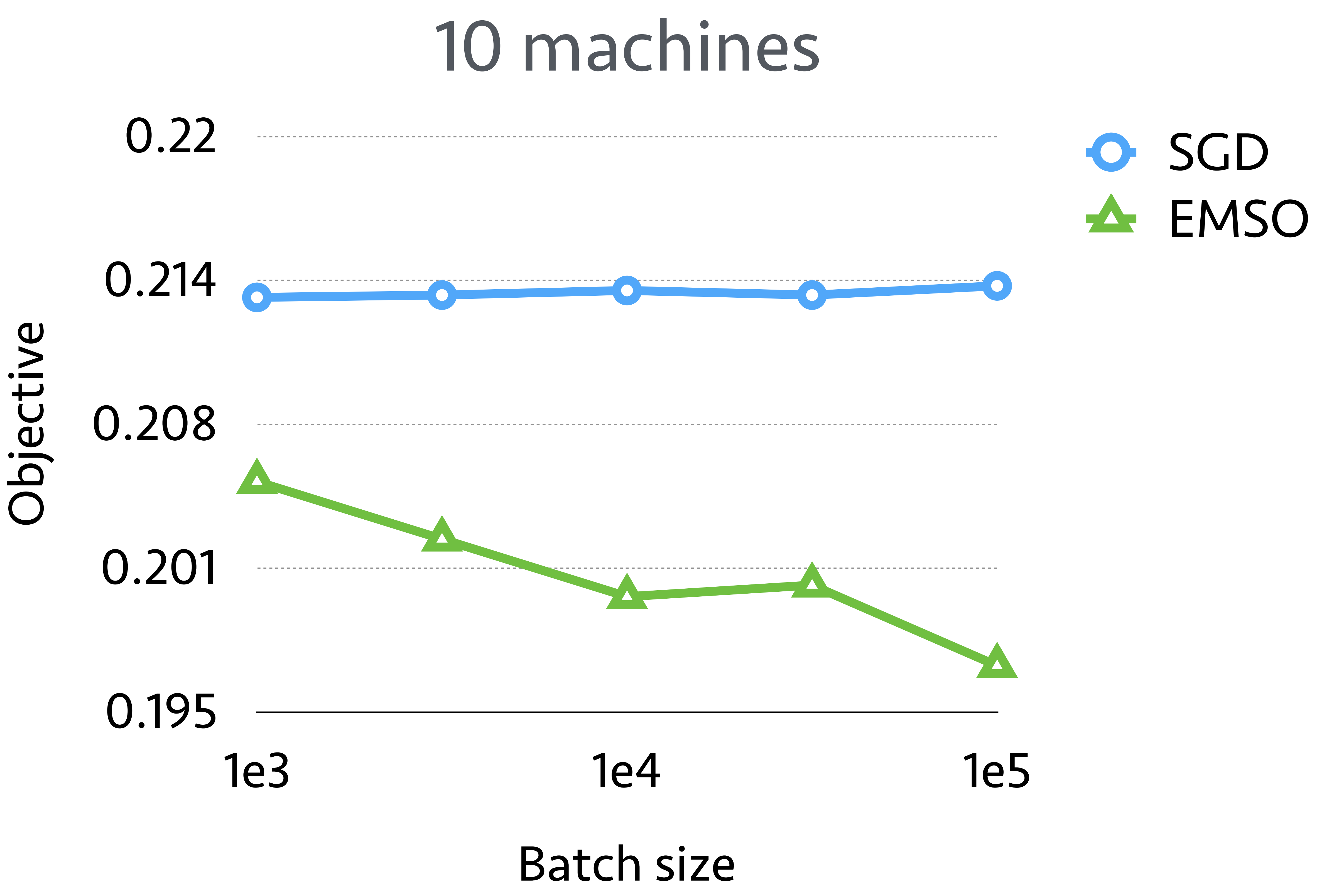
$$O(1/\sqrt{N})$$

(compared to $O(1/\sqrt{N} + b/N)$)

Experiment

- Ads click prediction with fixed run time





Extended to deep learning in [Keskar et al, arXiv'16]

Large-scale problems $\min_{w} \sum_{i=1}^{n} f_i(w)$

Reduce communication cost

- Distributed systems
- Large scale optimization

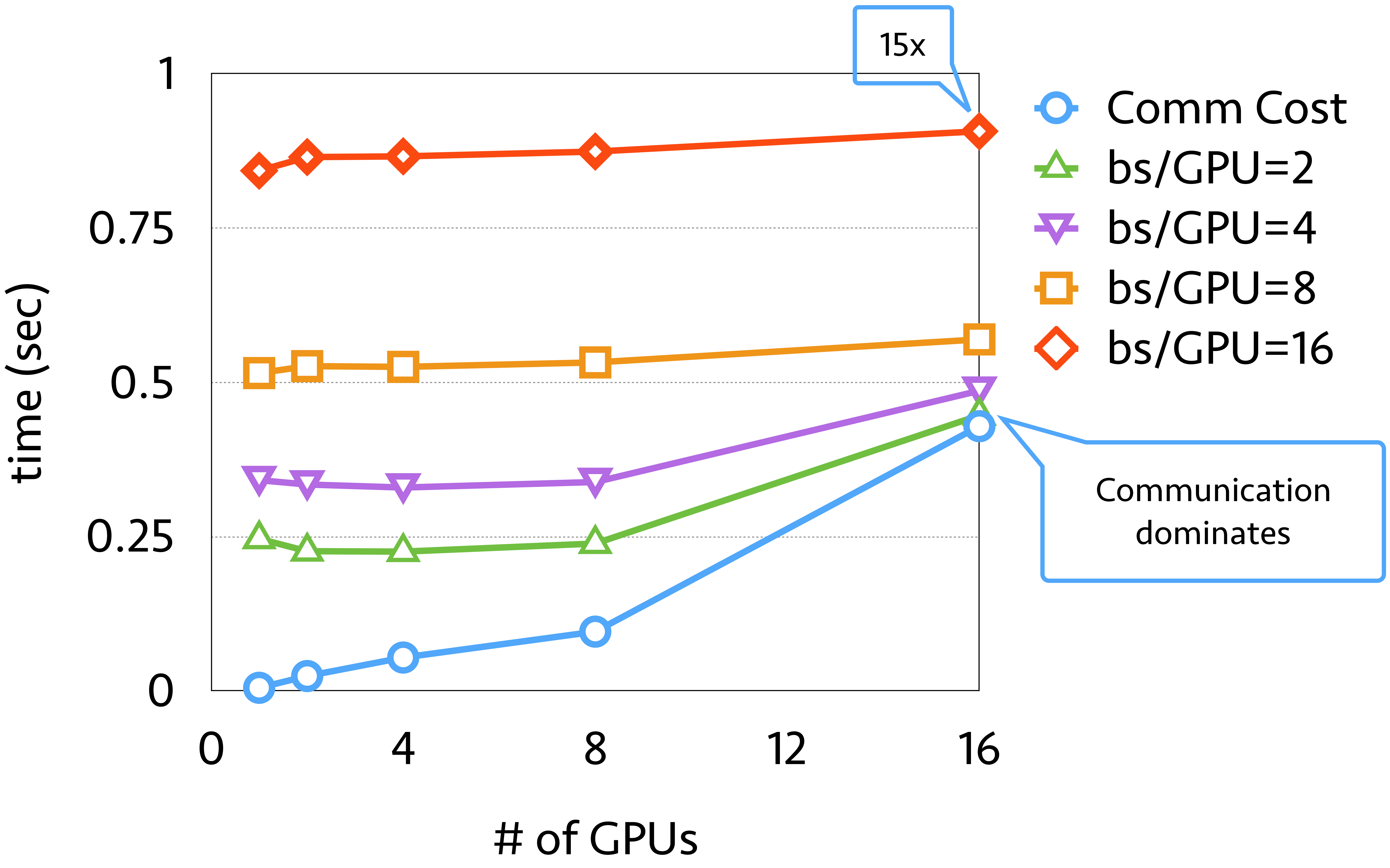
Co-design

- ✓ Communicate less
- ✓ Message compression
- ✓ Relaxed data consistency

With appropriate computational frameworks and algorithm design, distributed machine learning can be made simple, fast, and scalable, both in theory and in practice.

Backup slides

Scaling to 16 GPUs in a Single Machine



Compare to a L-BFGS Based System