DiFacto — Distributed Factorization Machines

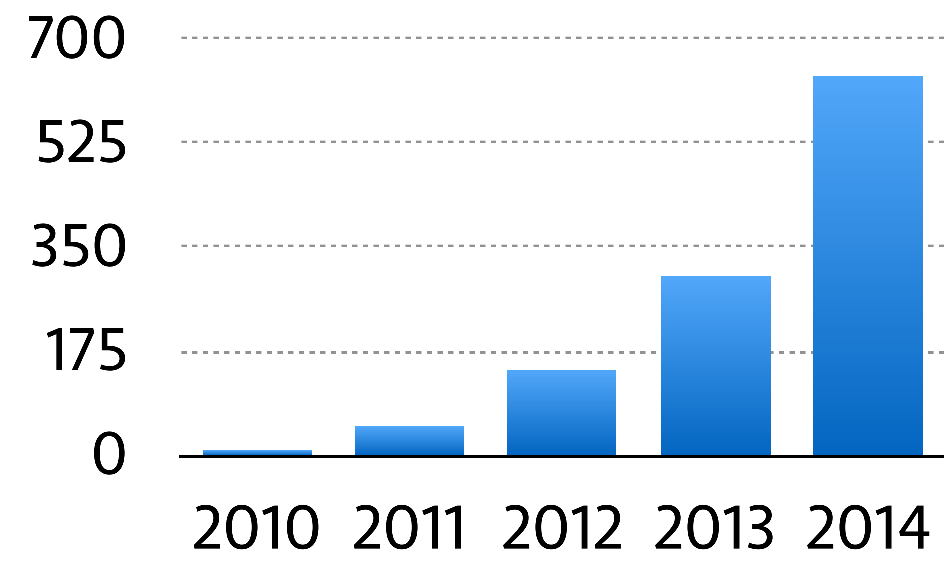
Mu Li

Joint work with Ziqi Liu, Alex Smola, and Yu-Xiang Wang

Carnegie Mellon University

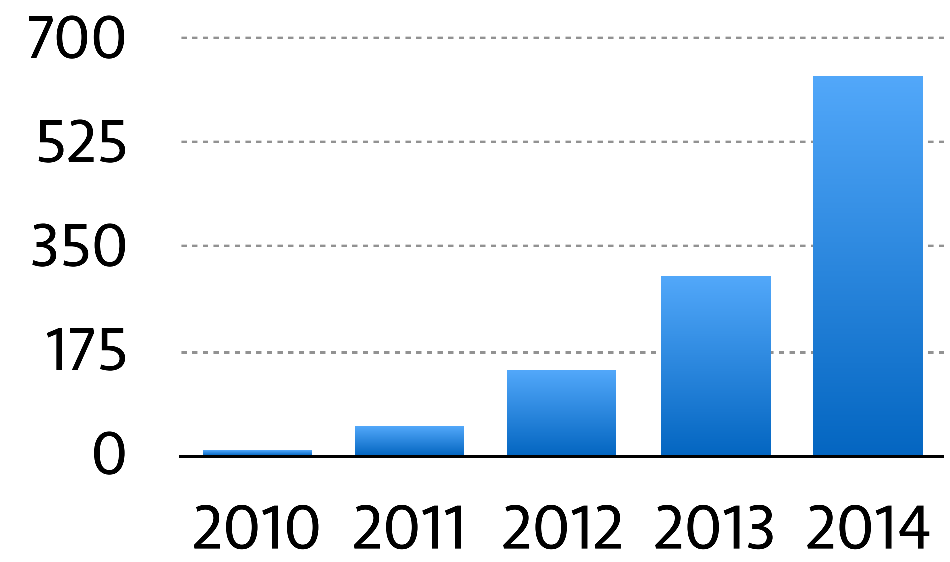
Linear model is widely used for large-scale datasets

Training data size (TB) of linear model
for Ads CTR estimation
in an Internet company

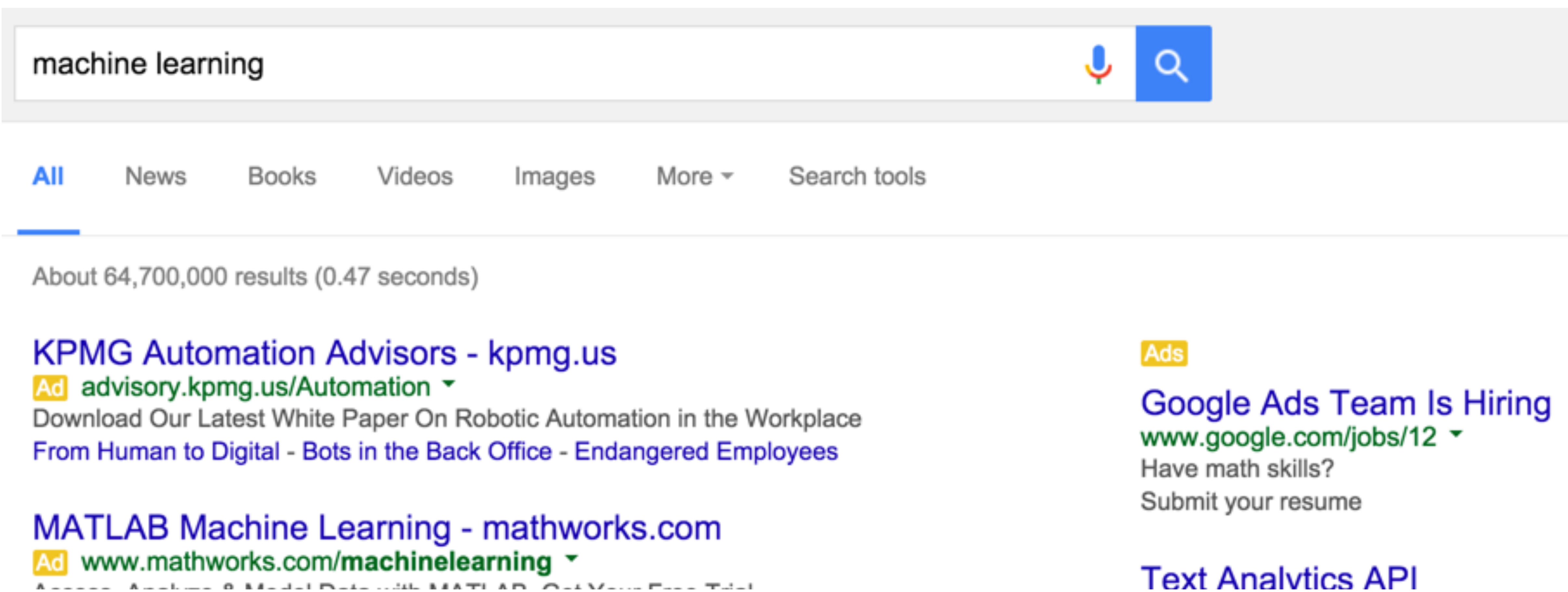


Linear model is widely used for large-scale datasets

Training data size (TB) of linear model for Ads CTR estimation in an Internet company

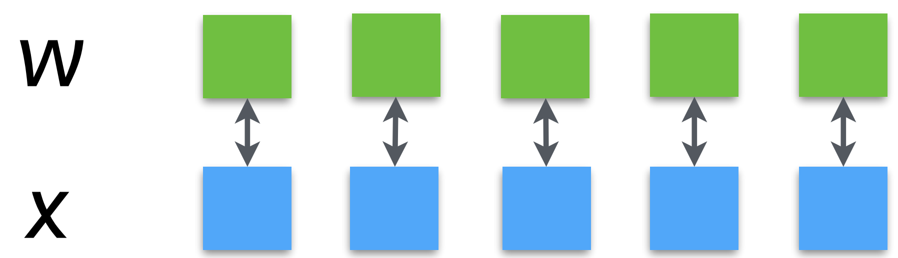


Reach model capacity limit when the data go very large



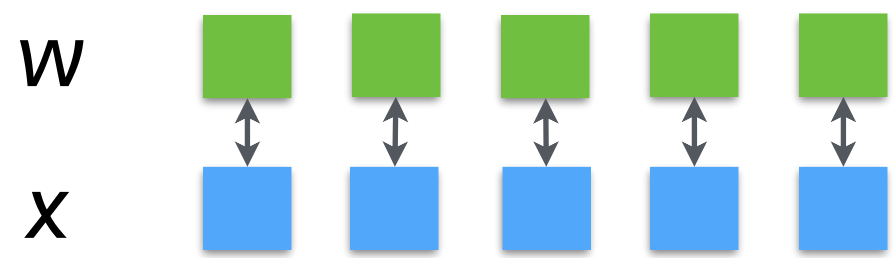
Factorization Machine

- Linear model predicts by $f(x) = \sum_i x_i w_i$



Factorization Machine

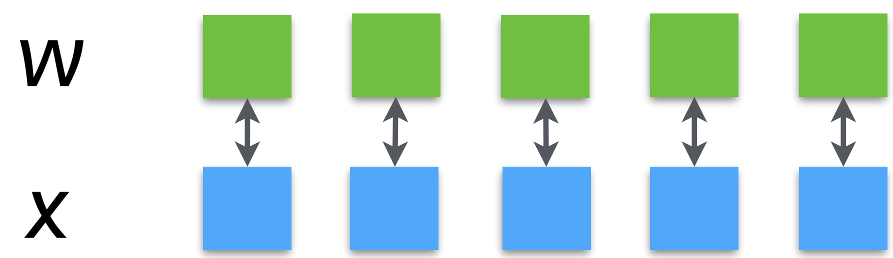
- Linear model predicts by $f(x) = \sum_i x_i w_i$



- Factorization machine (Rendle et al, '10) adds a *k*-dimensional embedding

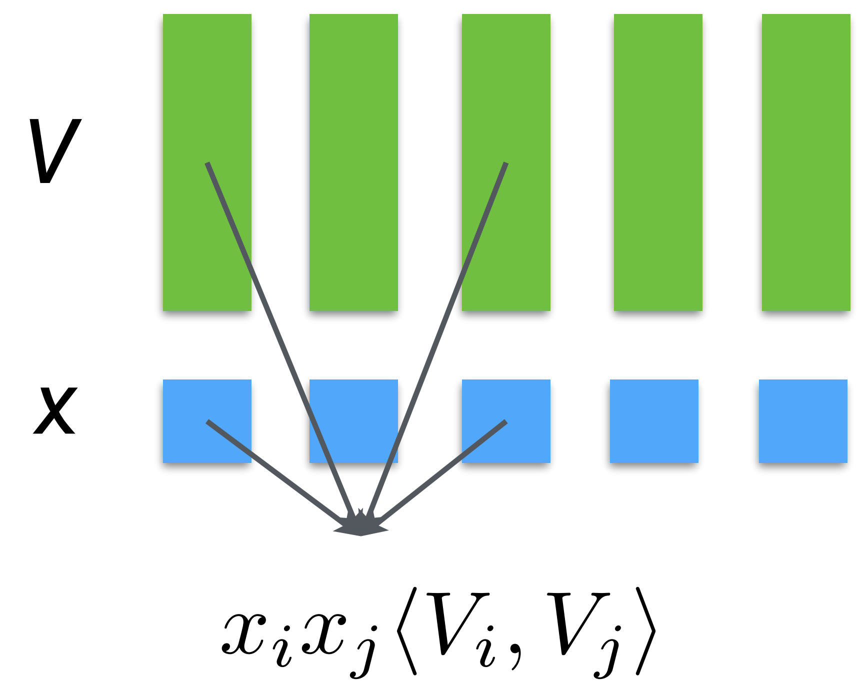
Factorization Machine

- Linear model predicts by $f(x) = \sum_i x_i w_i$



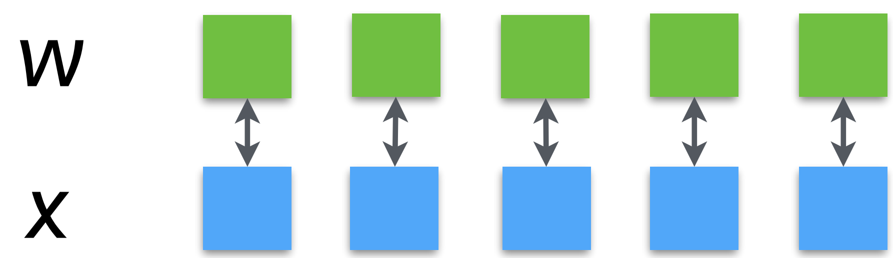
- Factorization machine (Rendle et al, '10) adds a *k*-dimensional embedding

$$f(x) = \sum_i x_i w_i + \sum_{i<j} x_i x_j \langle V_i, V_j \rangle$$



Factorization Machine

- Linear model predicts by $f(x) = \sum_i x_i w_i$

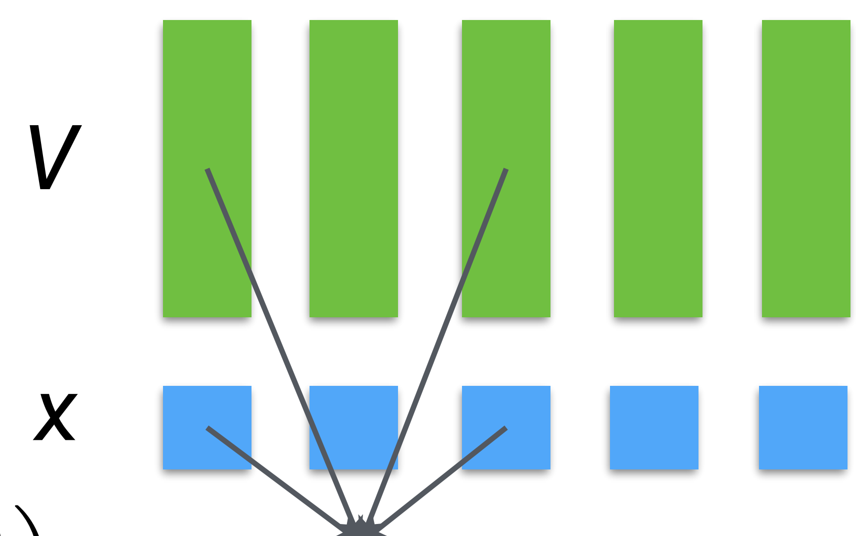


- Factorization machine (Rendle et al, '10) adds a *k*-dimensional embedding

$$f(x) = \sum_i x_i w_i + \sum_{i<j} x_i x_j \langle V_i, V_j \rangle$$

(can go beyond second-order)

$$+ \sum_{i<j<k} x_i x_j x_k \operatorname{tr}\left(V_i^{(3)} \otimes V_j^{(3)} \otimes V_k^{(3)} \right)$$



$x_i x_j \langle V_i, V_j \rangle$

The Challenge

Both computation and storage costs of k-dimension FM are k times larger than linear model

- On Criteo CTR dataset with 1.5B examples and 0.36B features

k	model size	FLOP per data pass	time for a single CPU (in theory)
100	288GB	10 P	5 hours
1,000	3TB	100 P	50 hours

The Challenge

Both computation and storage costs of k-dimension FM are k times larger than linear model

- On Criteo CTR dataset with 1.5B examples and 0.36B features

k	model size	FLOP per data pass	time for a single CPU (in theory)
100	288GB	10 P	5 hours
1,000	3TB	100 P	50 hours

Key Contributions

1. Reduce model capacity by exploring data sparsity
2. Highly efficient distributed training

The Challenge

Both computation and storage costs of k-dimension FM are k times larger than linear model

- On Criteo CTR dataset with 1.5B examples and 0.36B features

k	model size	FLOP per data pass	time for a single CPU (in theory)
100	288GB	10 P	5 hours
1,000	3TB	100 P	50 hours

Key Contributions

1. Reduce model capacity by exploring data sparsity
2. Highly efficient distributed training

Key Take Away

Make large-scale FM as cheap as linear model

Statistic Model

Distributed Optimization

Evaluation

Key Observation

- High-dimensional datasets are often extremely sparse
  - The count of feature occurrence often obeys a pow law distribution. e.g the Criteo dataset

occurrence >	1	10	100
# features (%)	53%	5.8%	0.49%

Key Observation

- High-dimensional datasets are often extremely sparse
  - The count of feature occurrence often obeys a pow law distribution. e.g the Criteo dataset

occurrence >	1	10x	10	10x	100
# features (%)	53%		5.8%		0.49%

Key Observation

- High-dimensional datasets are often extremely sparse
  - The count of feature occurrence often obeys a pow law distribution. e.g the Criteo dataset

occurrence >	1	10x	10	10x	100
# features (%)	53%		5.8%		0.49%

- If feature *i* appears less than *k+1* times in the data, then the problem of estimating (w_i, V_i) is underdetermined

Key Observation

- High-dimensional datasets are often extremely sparse
  - The count of feature occurrence often obeys a pow law distribution. e.g the Criteo dataset

occurrence >	1	10x	10	10x	100
# features (%)	53%		5.8%		0.49%

- If feature *i* appears less than *k+1* times in the data, then the problem of estimating (w_i, V_i) is underdetermined

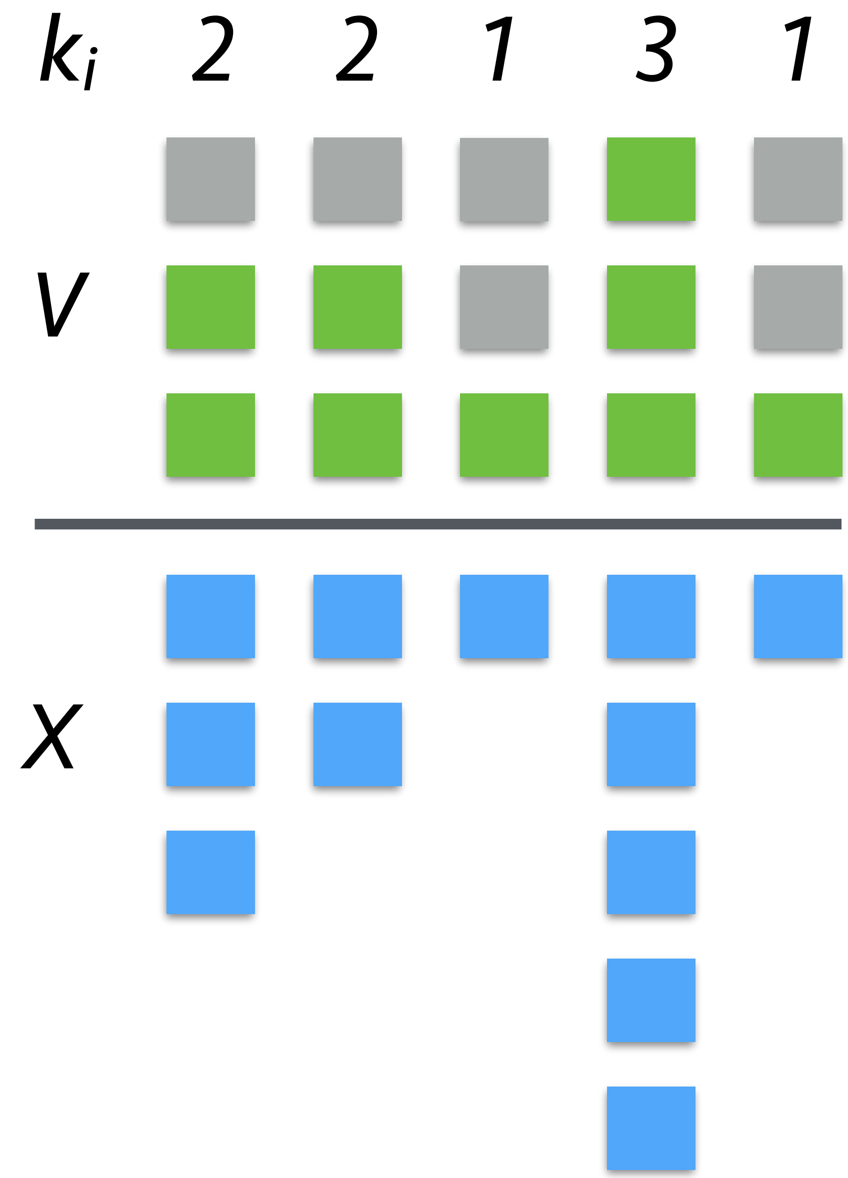
Our solution

Data and model adaptive regulazations to reduce model capacity on unimportant features

Memory Adaptive Constraints

- Limit the effective embedding dimension k_i for "tail" feature i

$$V_{ij} = 0 \text{ for all } j > k_i$$



Memory Adaptive Constraints

- Limit the effective embedding dimension k_i for "tail" feature i

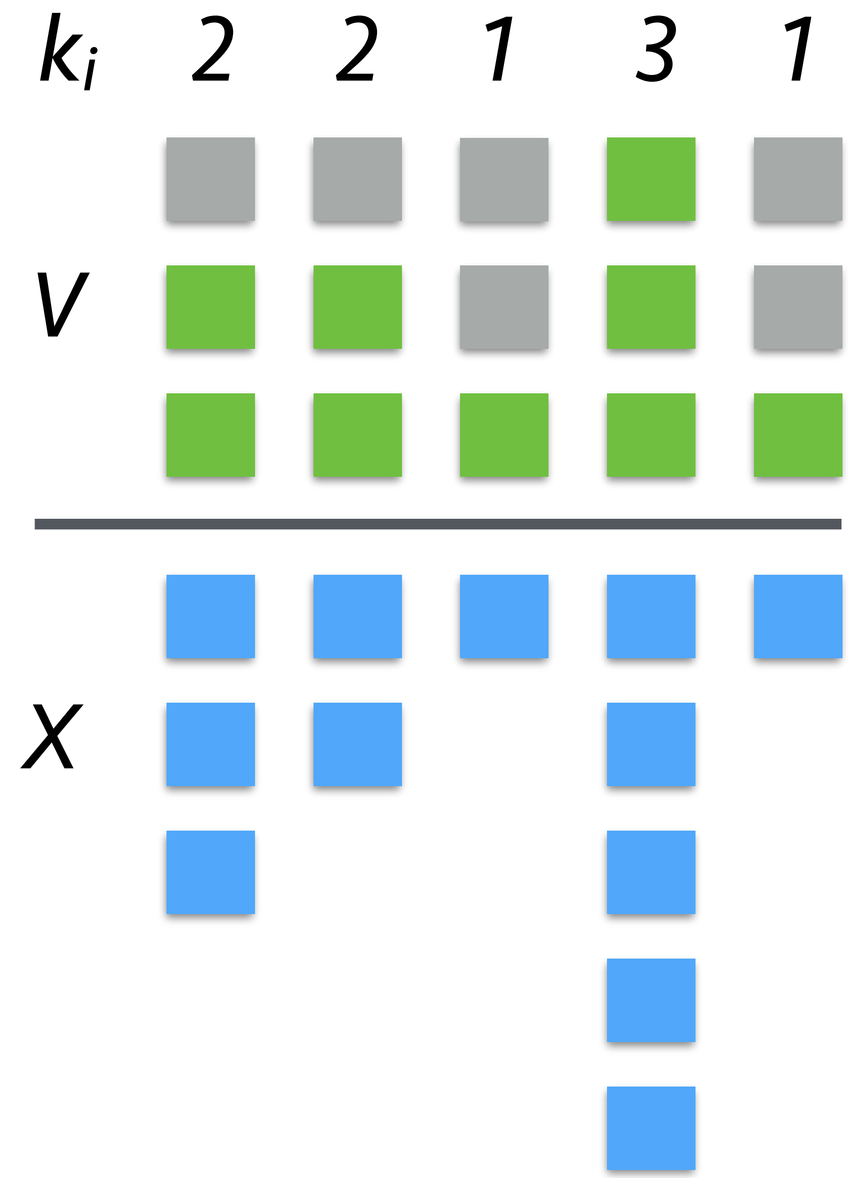
$$V_{ij} = 0 \text{ for all } j > k_i$$

- Choose k_i according to the number of occurrence of feature i, n_i
  - simple choice:

$$k_i = \begin{cases} k & \text{if } n_i \geq k \\ 0 & \text{otherwise} \end{cases}$$

  - three levels:

$$k_i = \begin{cases} 10k & \text{if } n_i \geq 10k \\ k & \text{if } 10k > n_i \geq k \\ \min(n_i, k) & \text{otherwise} \end{cases}$$

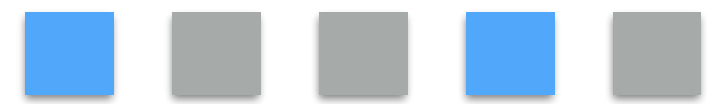


Sparse Regularization

- Model adaptive capacity control by sparse regularization
- Encourage a sparse linear term

$$\lambda_1 \left\| w \right\|_1$$

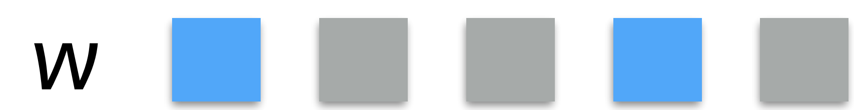
w



Sparse Regularization

- Model adaptive capacity control by sparse regularization
- Encourage a sparse linear term

$$\lambda_1 \|w\|_1$$



- Need sparse V too:
  - Structured sparsity on V

$$\sum_i \left[w_i^2 + \|V_i\|_2^2\right]^{\frac{1}{2}} + \|V_i\|_2$$

  - A simpler solution

$$V_i = 0 \text{ if } w_i = 0$$

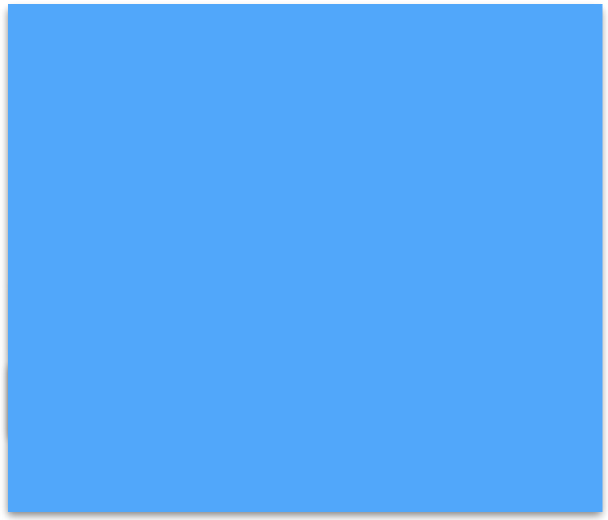
Statistic Model

Distributed Optimization

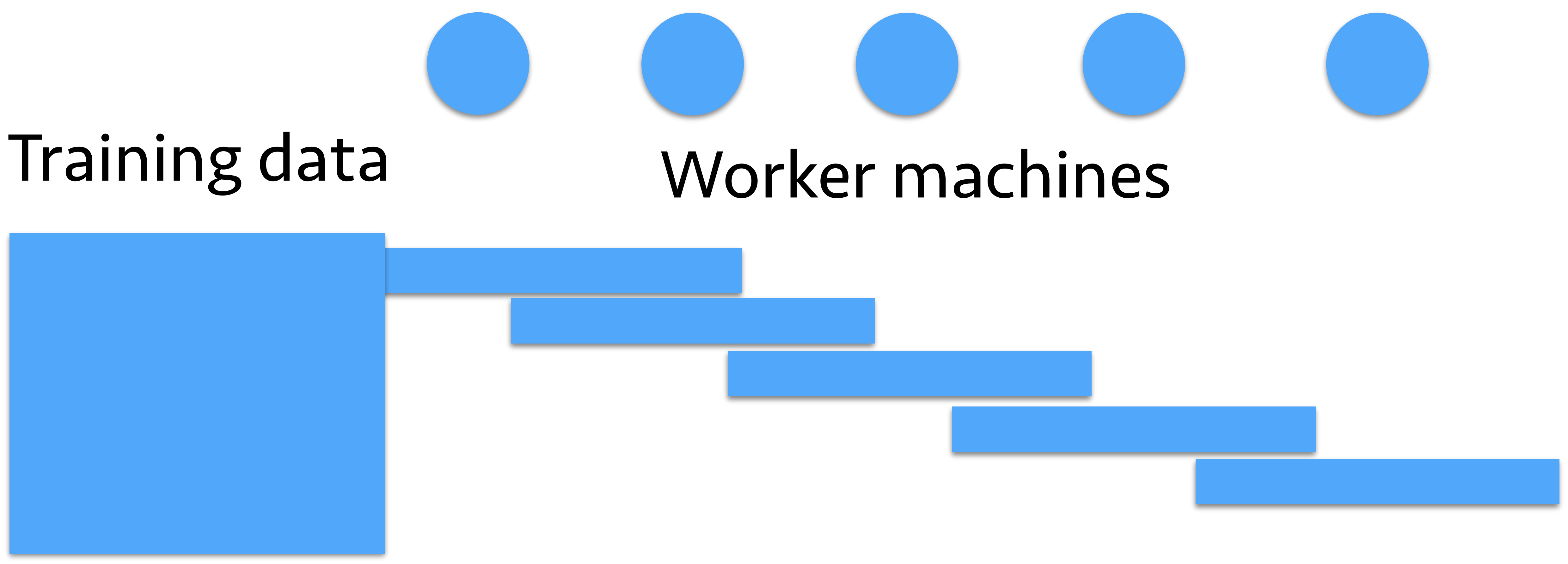
Evaluation

Distributed by Parameter Server (Li et al, '14)

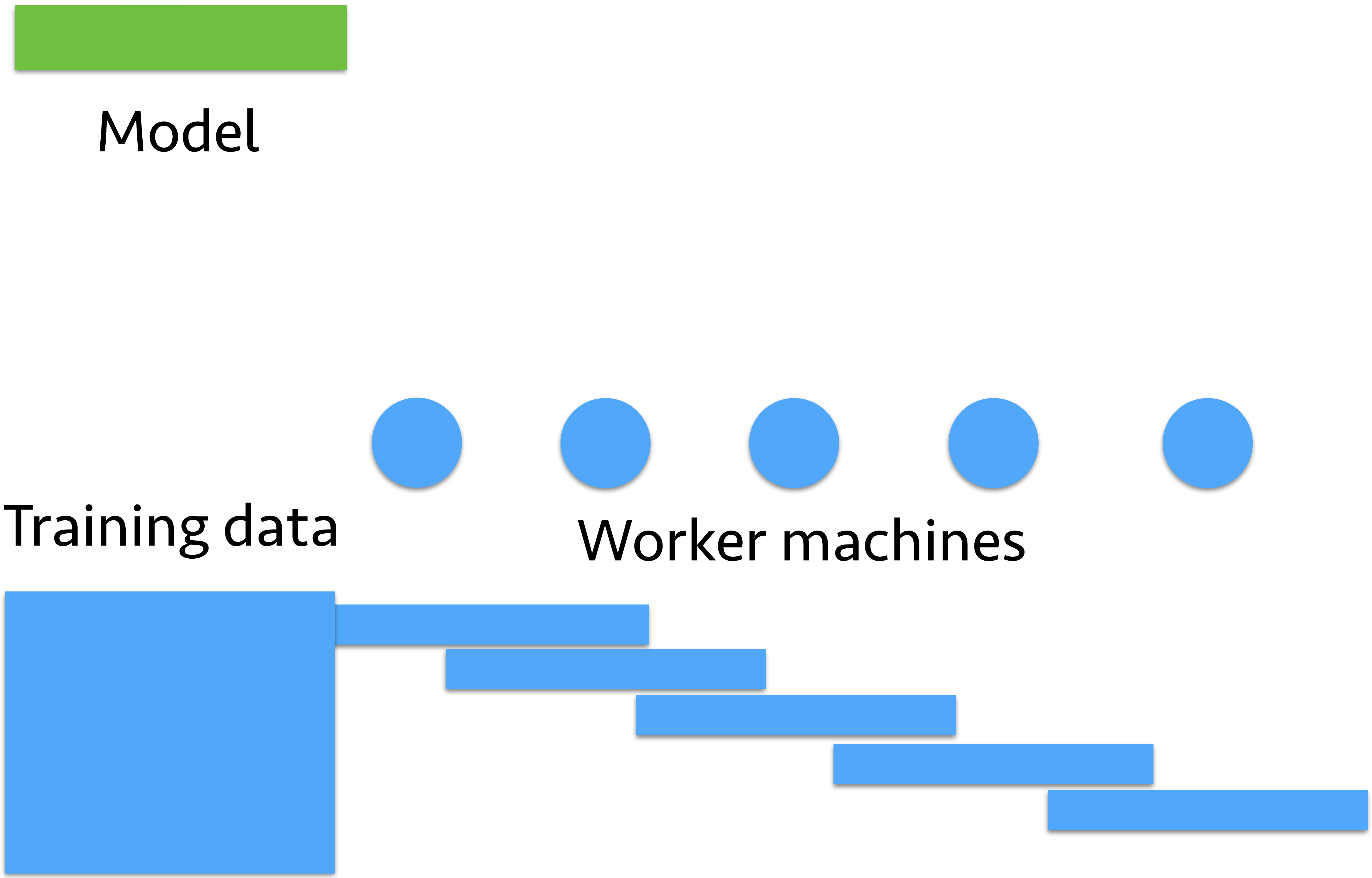
Training data



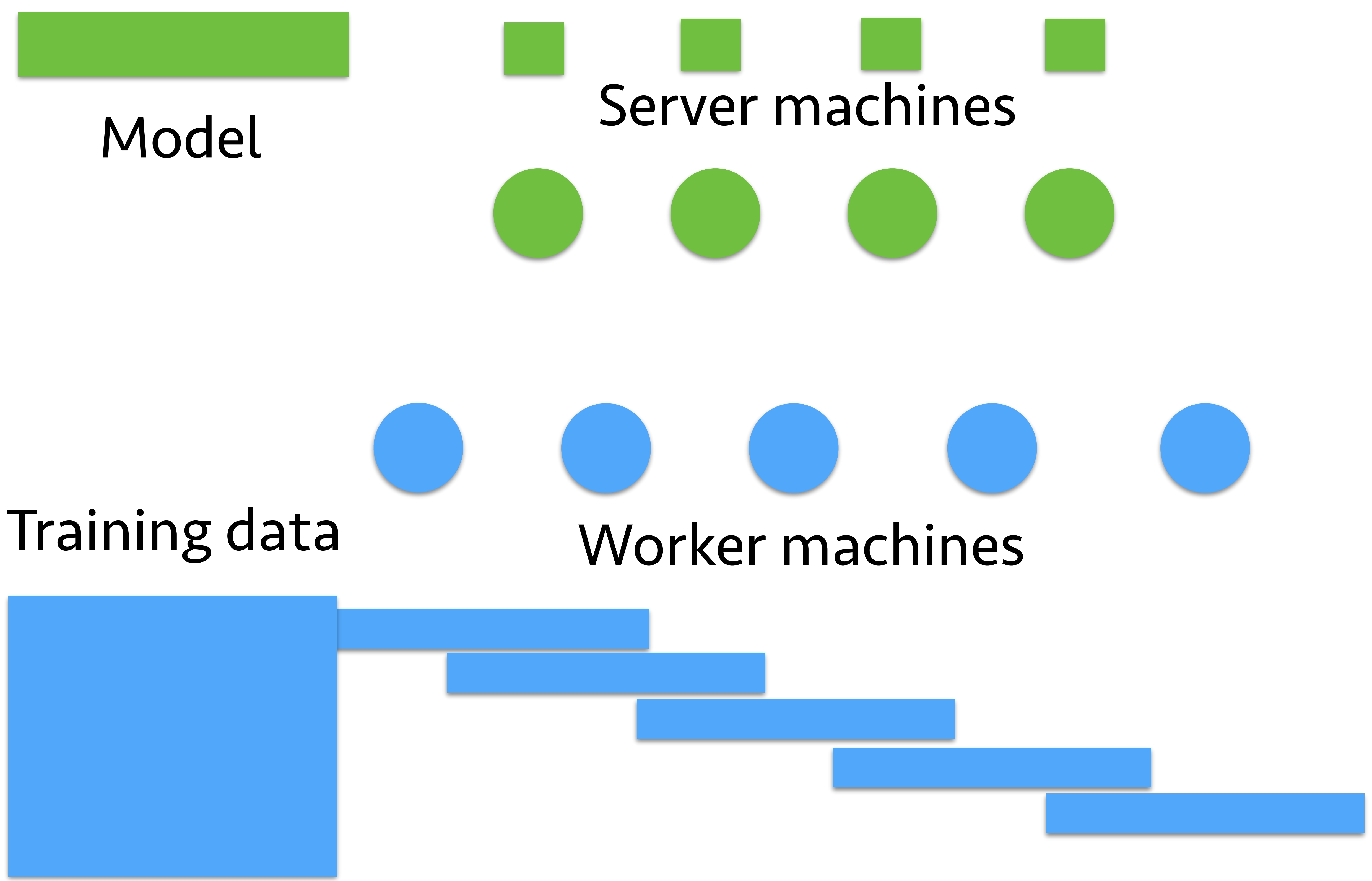
Distributed by Parameter Server (Li et al, '14)



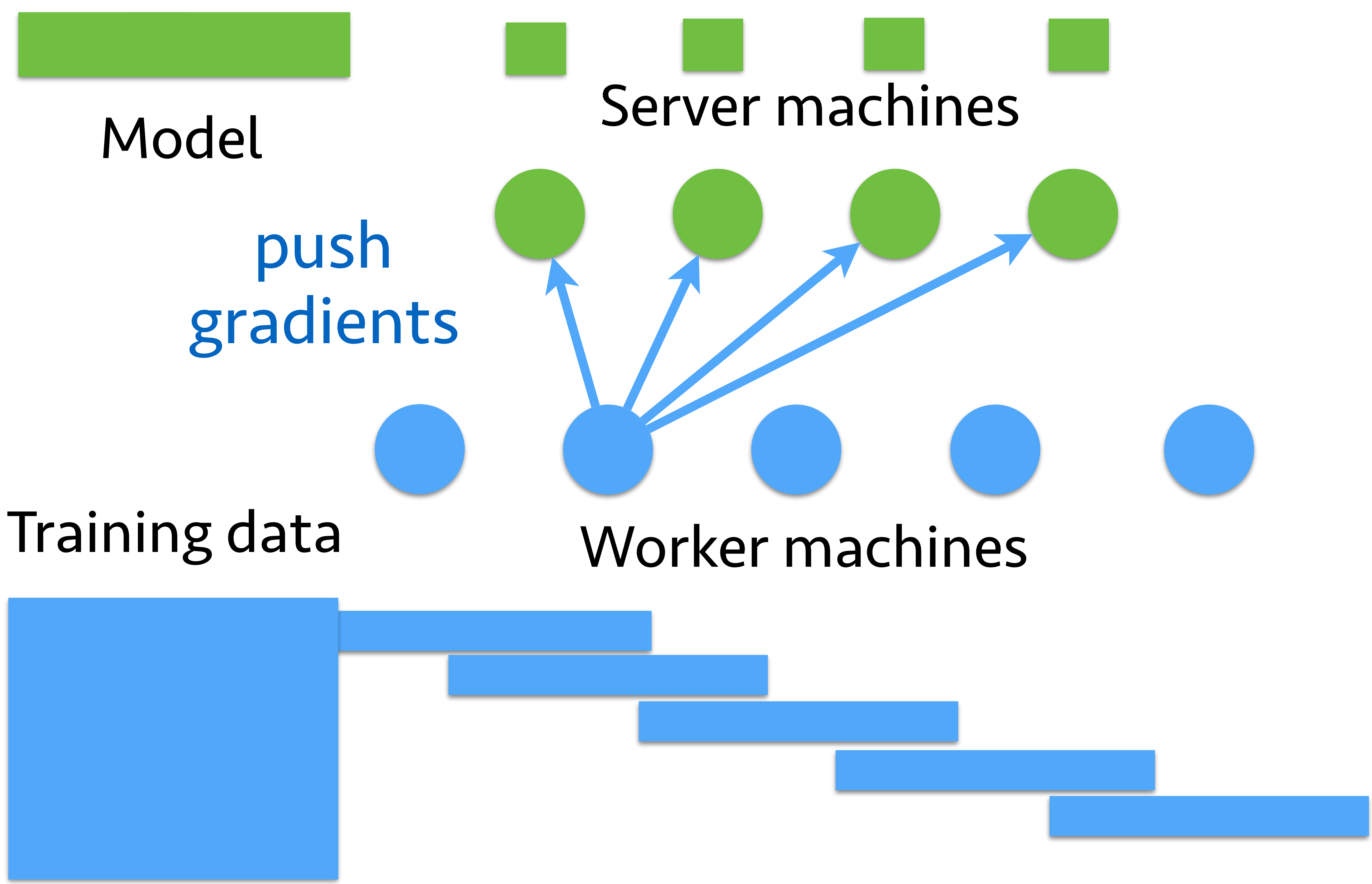
Distributed by Parameter Server (Li et al, '14)



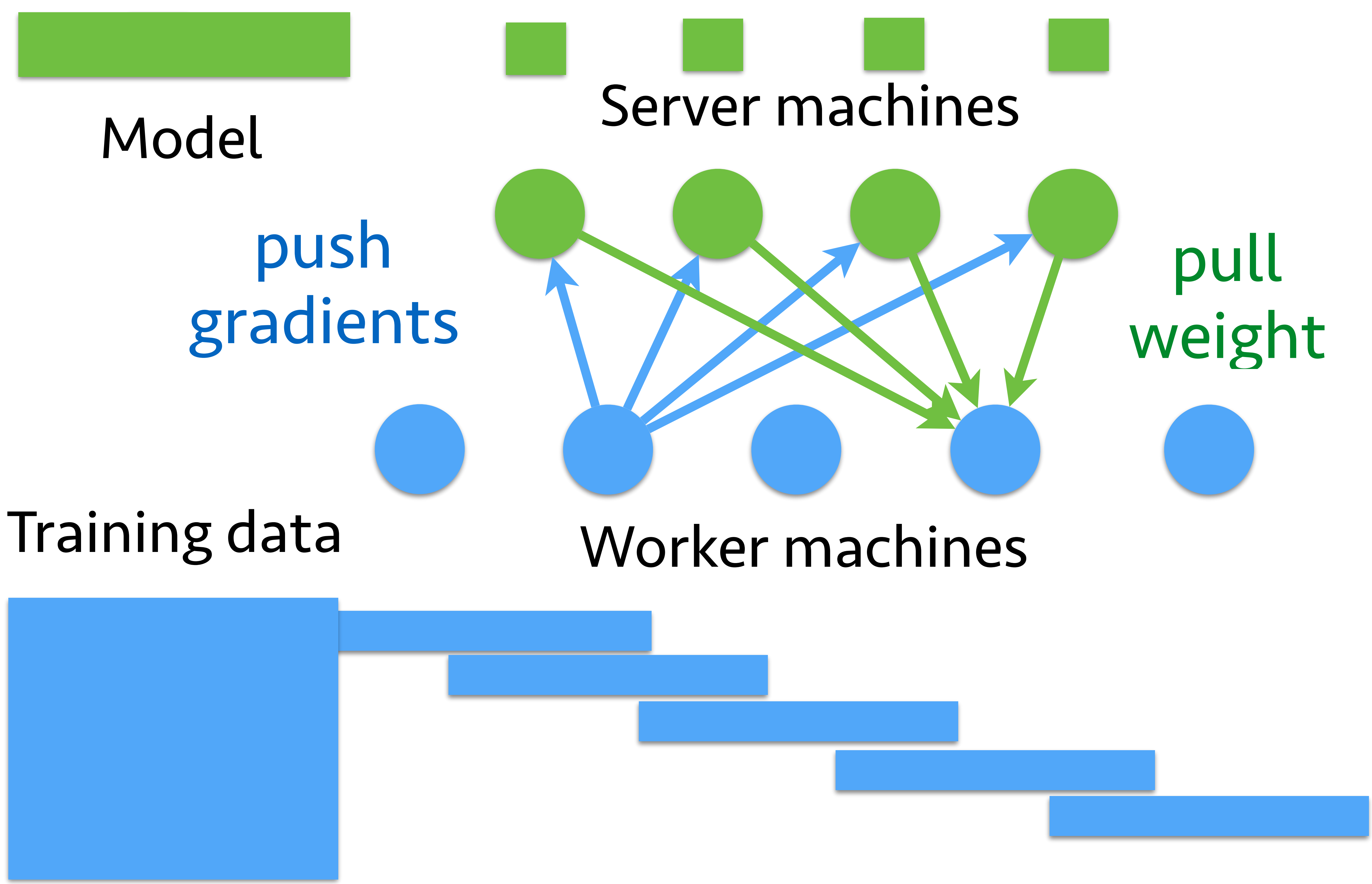
Distributed by Parameter Server (Li et al, ’14)



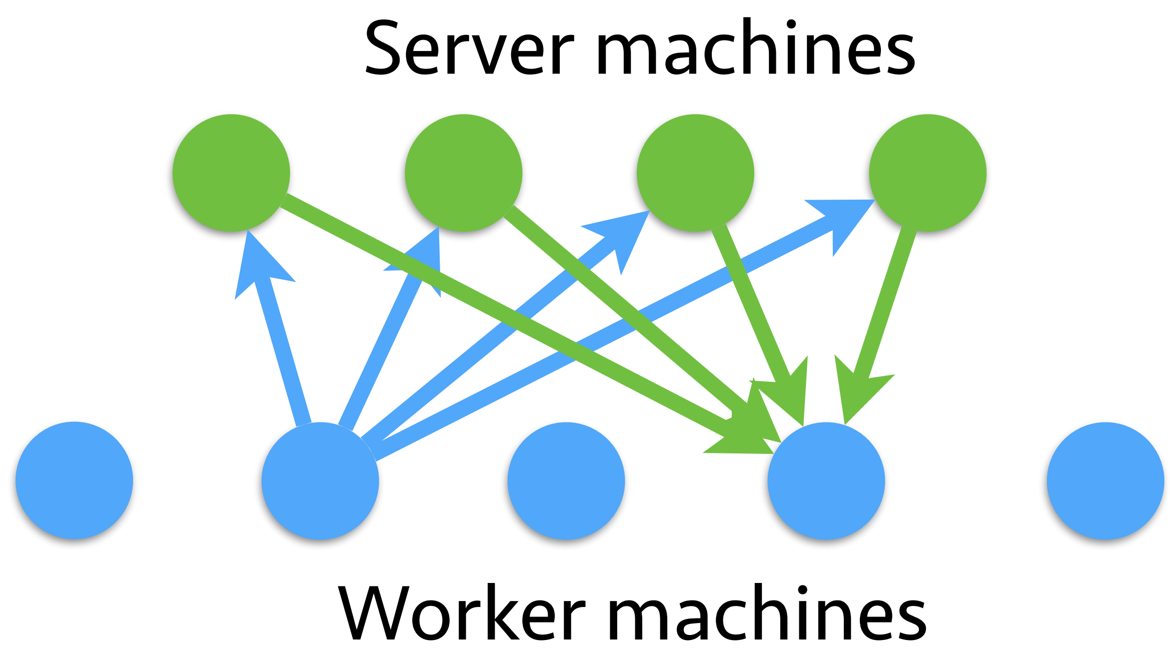
Distributed by Parameter Server (Li et al, '14)



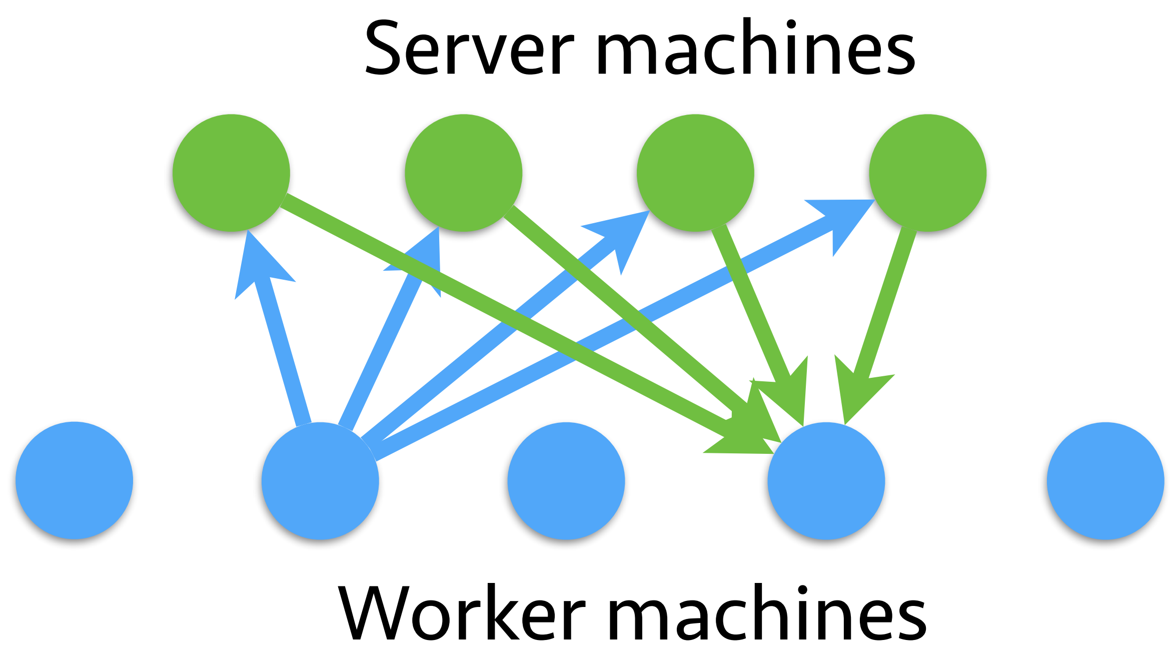
Distributed by Parameter Server (Li et al, '14)



Distributed Asynchronous SGD



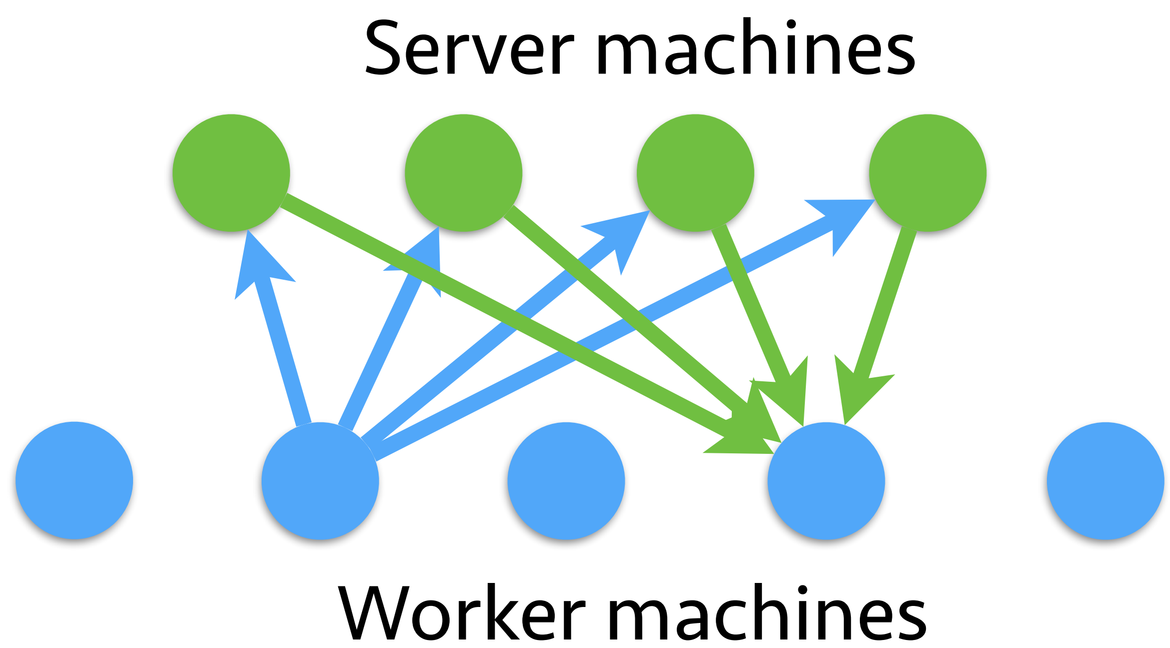
Distributed Asynchronous SGD



Workers run independently

- For each iteration
  - read a new minibatch
  - pull weights from the servers
  - compute gradients
  - push gradients into the servers

Distributed Asynchronous SGD



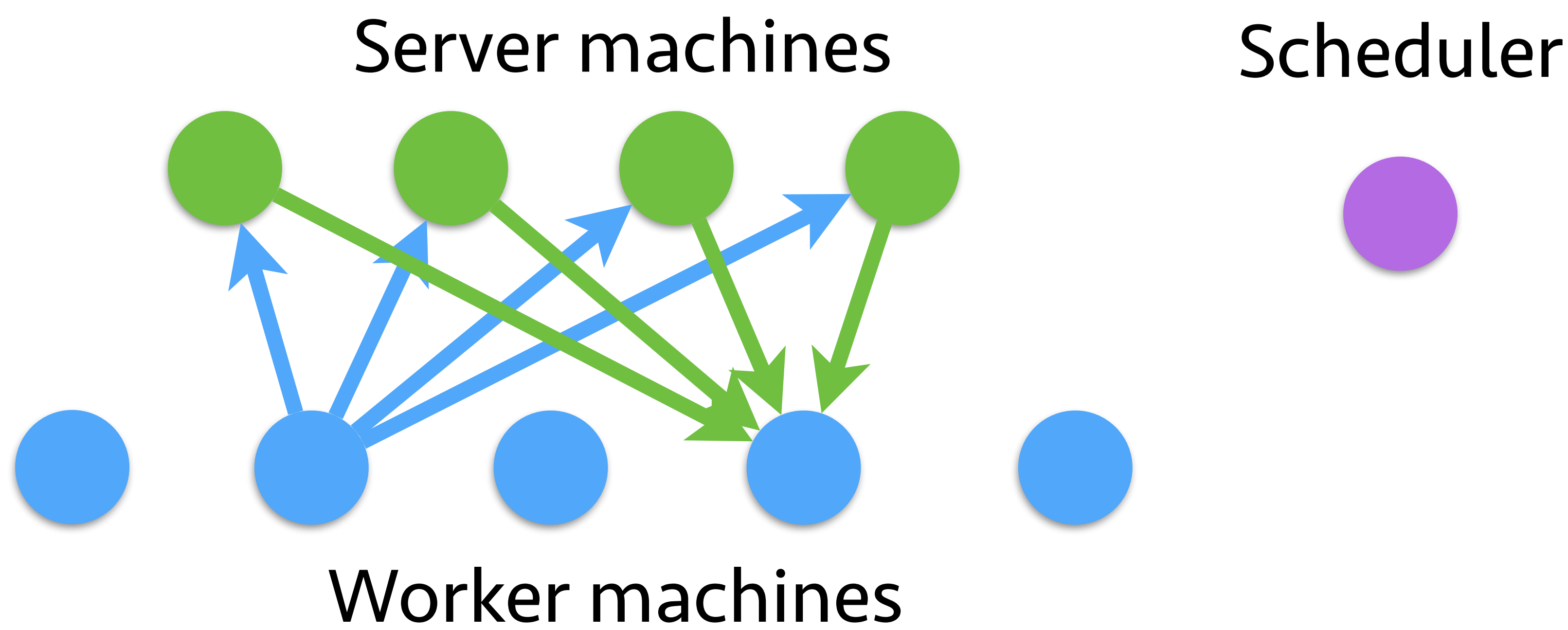
Workers run independently

- For each iteration
  - read a new minibatch
  - pull weights from the servers
  - compute gradients
  - push gradients into the servers

Servers update weights:

- Update *V* by adagrad
- Update *w* by FTRL

Distributed Asynchronous SGD



Workers run independently

- For each iteration
  - read a new minibatch
  - pull weights from the servers
  - compute gradients
  - push gradients into the servers

Servers update weights:

- Update V by adagrad
- Update w by FTRL

Scheduler node:

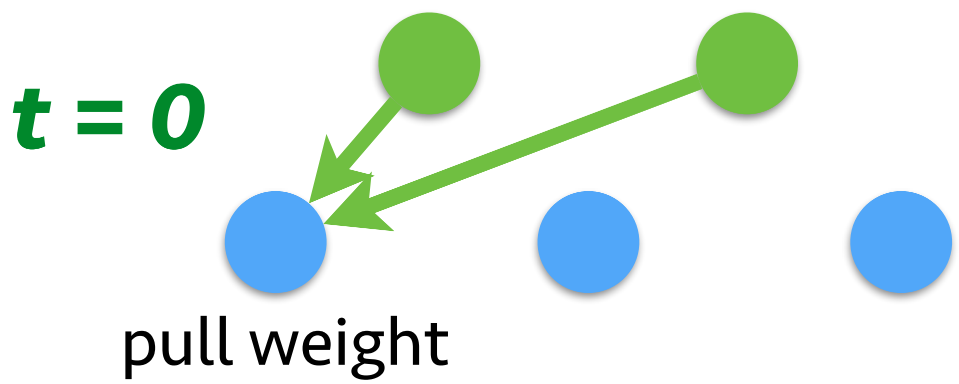
- manages load balanced
- achieves fault tolerance

Data consistency

- Async SGD trade-off data consistency for system performance

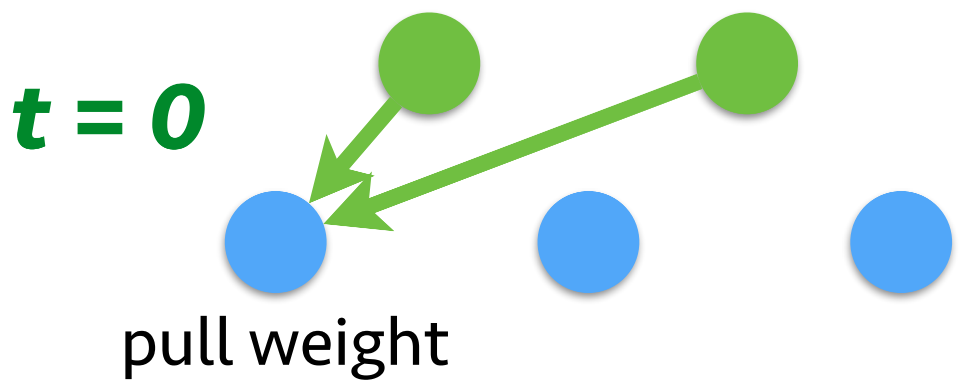
Data consistency

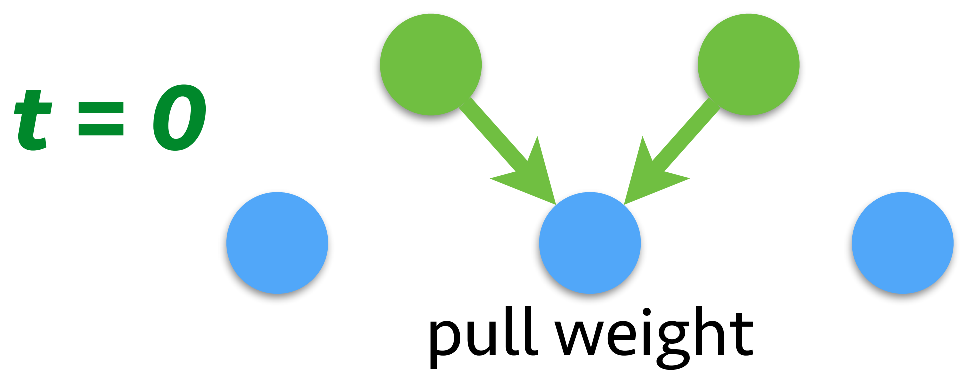
- Async SGD trade-off data consistency for system performance



Data consistency

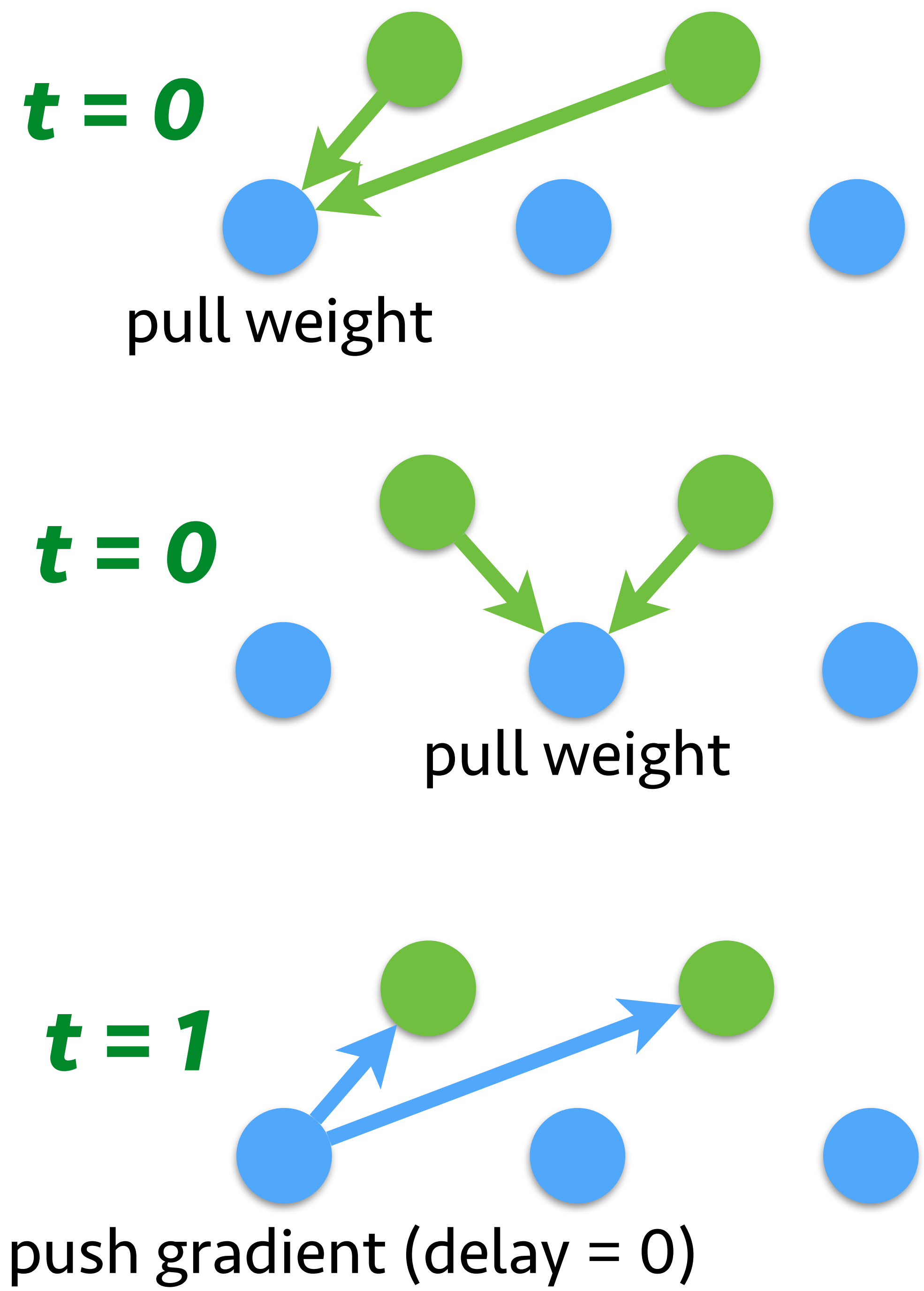
- Async SGD trade-off data consistency for system performance





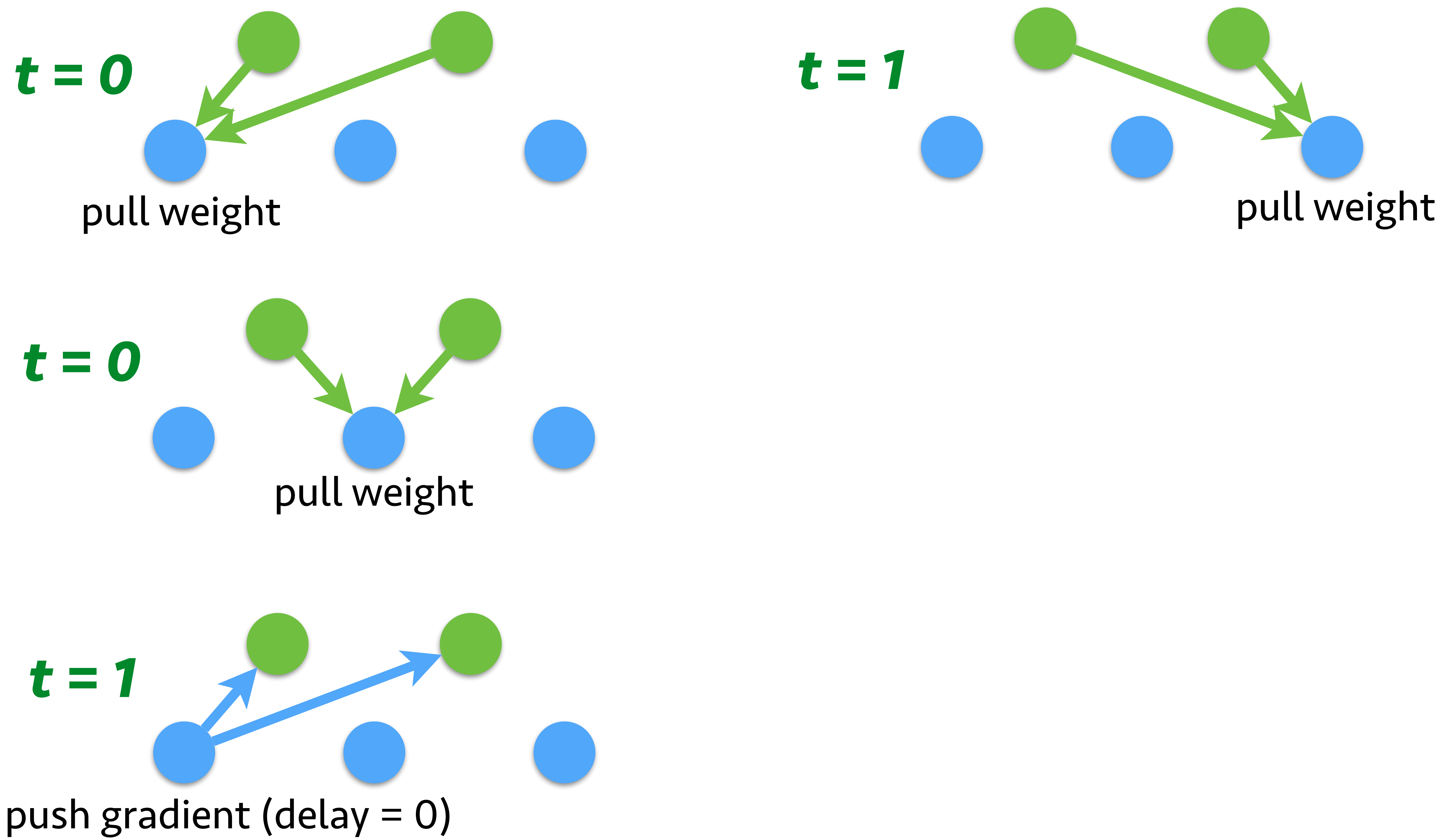
Data consistency

- Async SGD trade-off data consistency for system performance



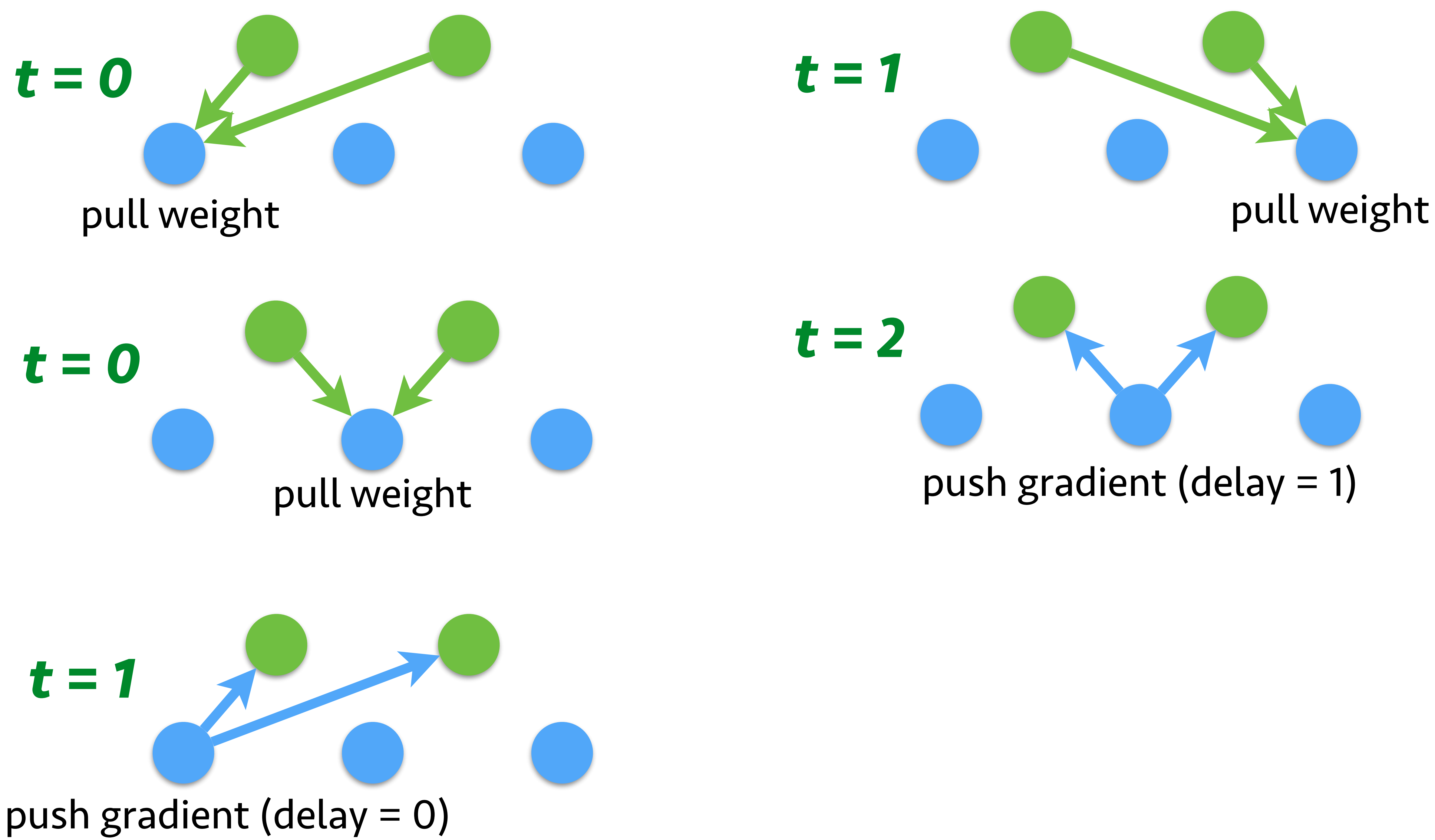
Data consistency

- Async SGD trade-off data consistency for system performance



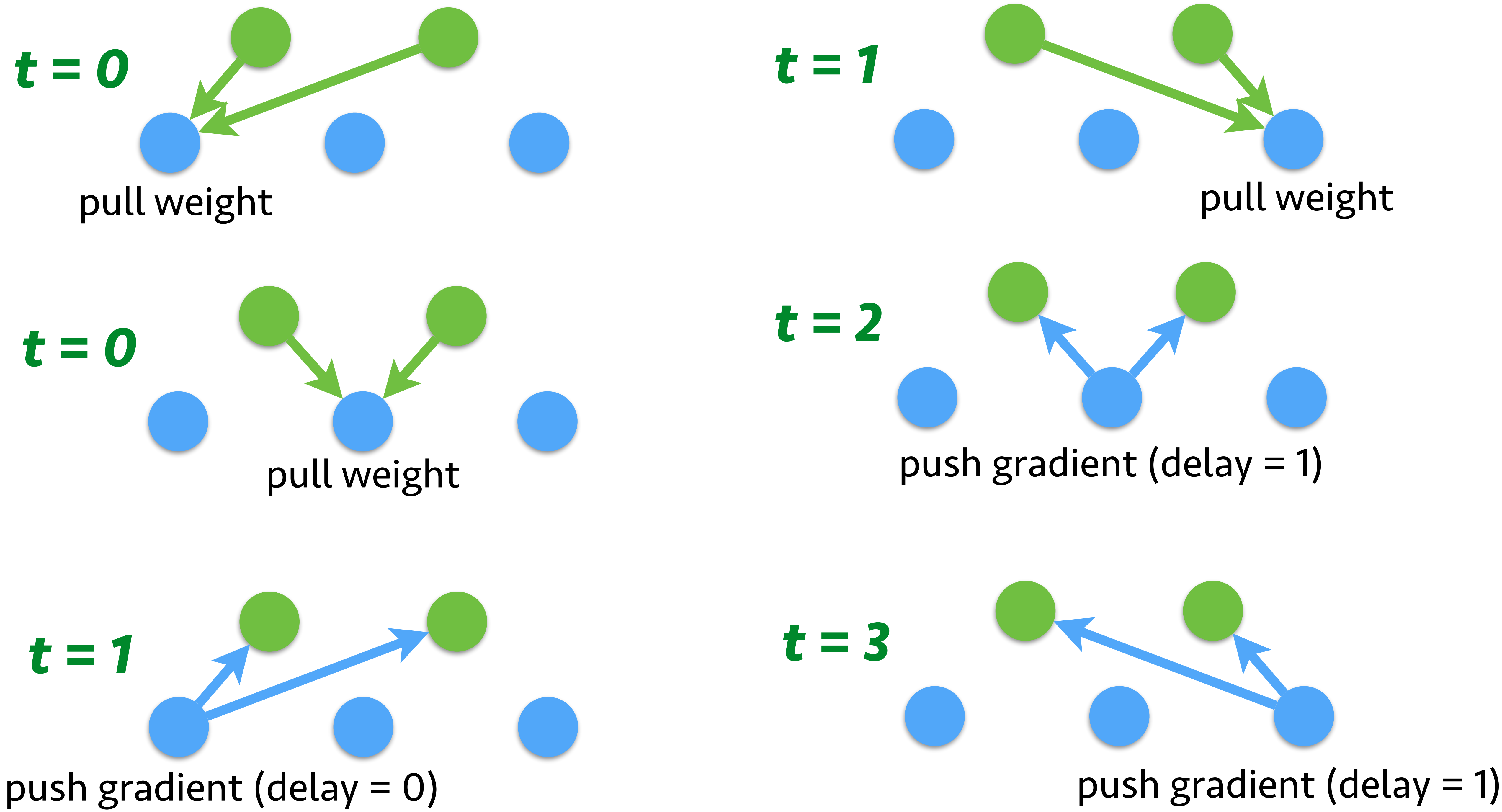
Data consistency

- Async SGD trade-off data consistency for system performance



Data consistency

- Async SGD trade-off data consistency for system performance



Theoretical Analysis

Theoretical Analysis

- Key assumptions
  - maximal delay is upper-bonded by τ
  - stochastic gradient is L-Lipschitz, and variance is bounded by σ^2

Theoretical Analysis

- Key assumptions
  - maximal delay is upper-bonded by τ
  - stochastic gradient is L-Lipschitz, and variance is bounded by σ^2
- If choose a constant step-size $\eta = \sqrt{\frac{C}{L\tau\sigma^2}}$ then for every $T \geq 4LC(\tau+1)^2/\sigma$

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\|\nabla f(x_t)\|^2 \leq 4\sqrt{\frac{CL}{T}}\sigma$$

  - LHS is an intuitive measure of distance from stationary point
  - Delay may slow convergence
  - We can use a large learning rate for large minibatch size or sparse dataset

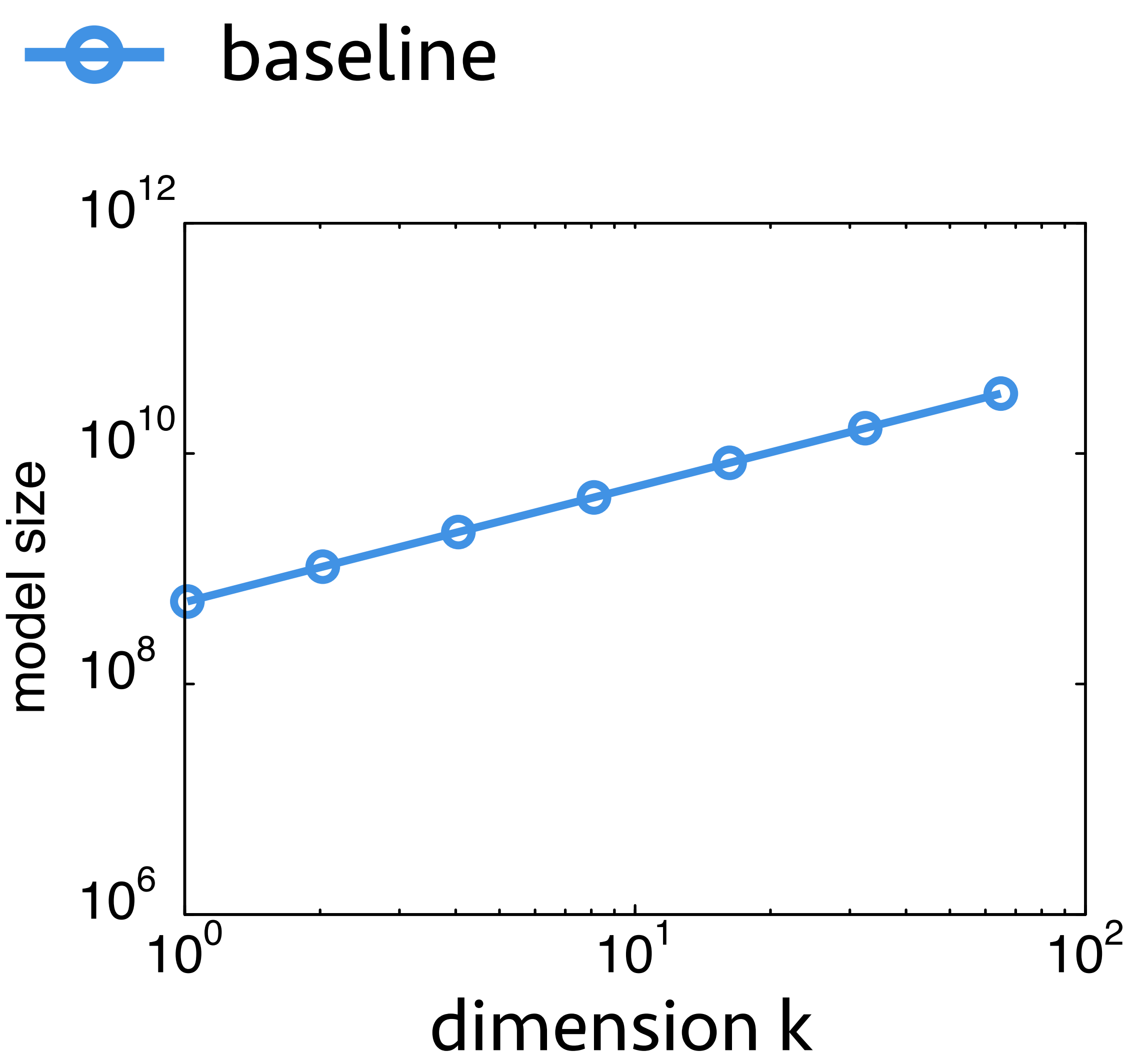
Statistic Model

Distributed Optimization

Evaluation

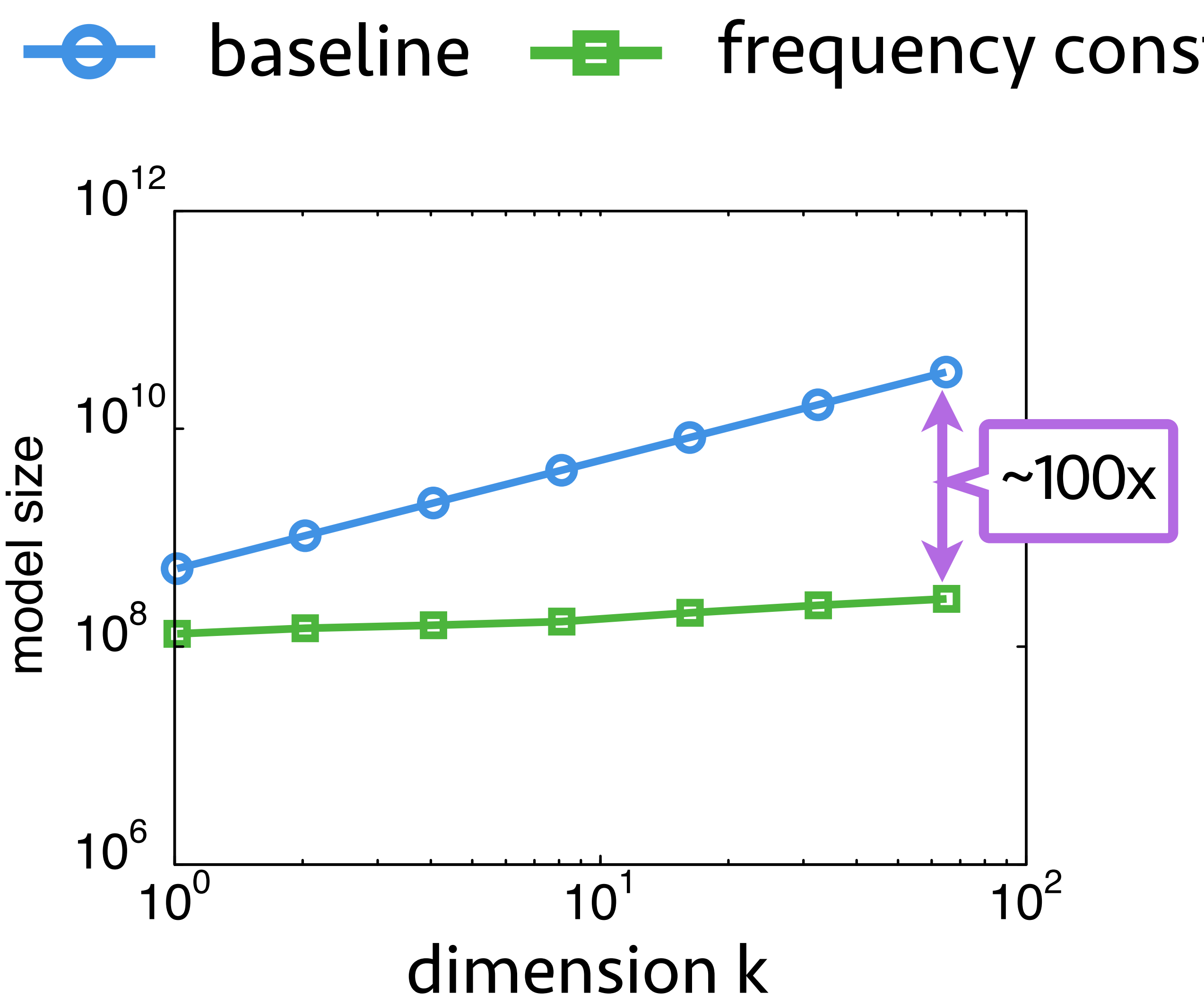
Adaptive Memory

- Criteo dataset: 1.5B examples, 360M features
- Run on 10 AWS EC2 machines



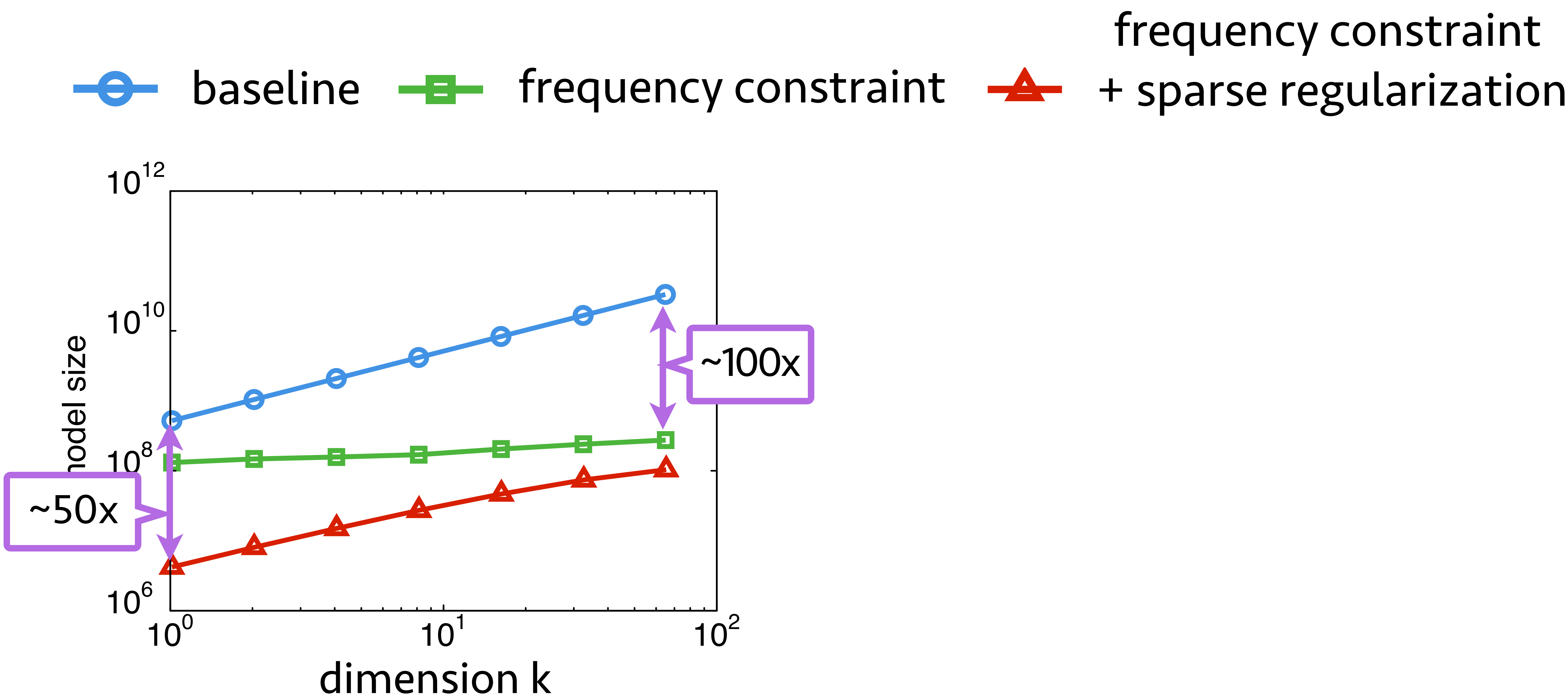
Adaptive Memory

- Criteo dataset: 1.5B examples, 360M features
- Run on 10 AWS EC2 machines



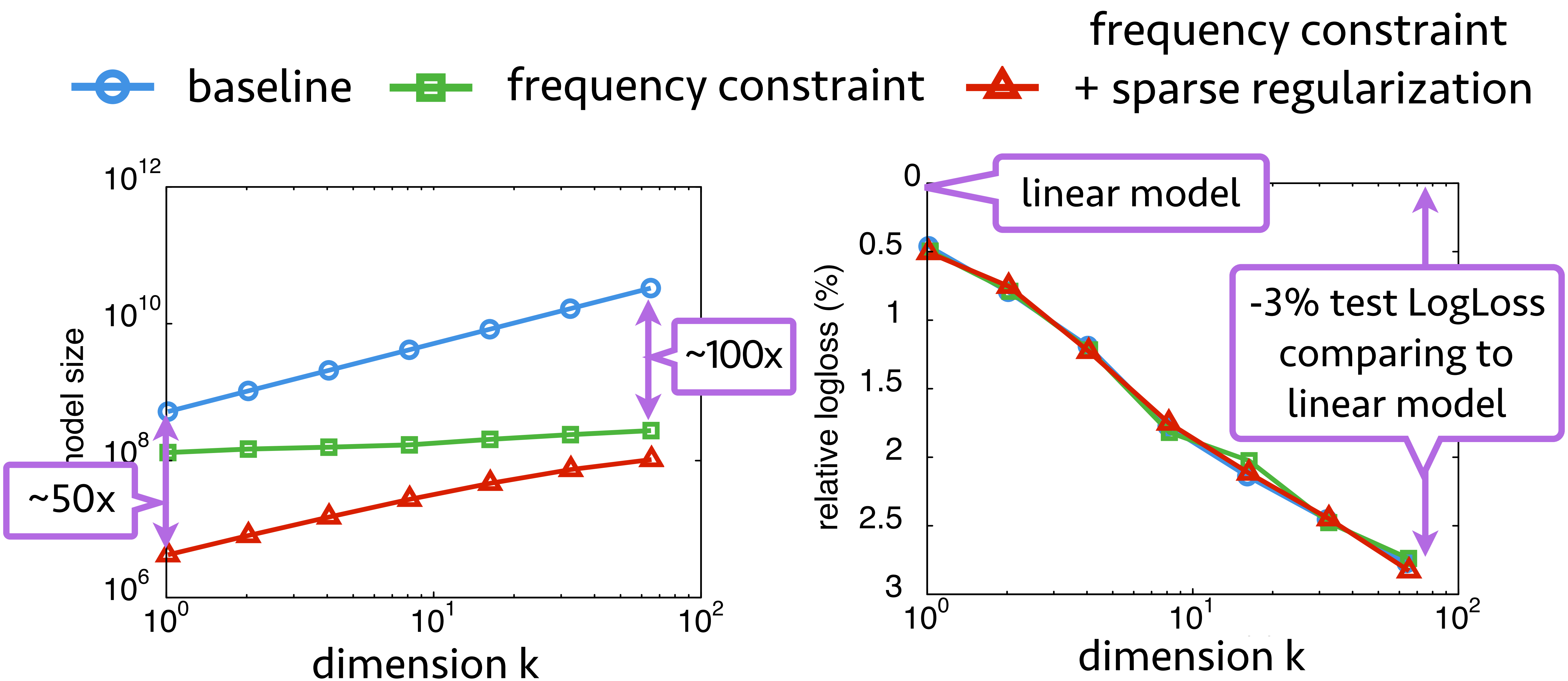
Adaptive Memory

- Criteo dataset: 1.5B examples, 360M features
- Run on 10 AWS EC2 machines



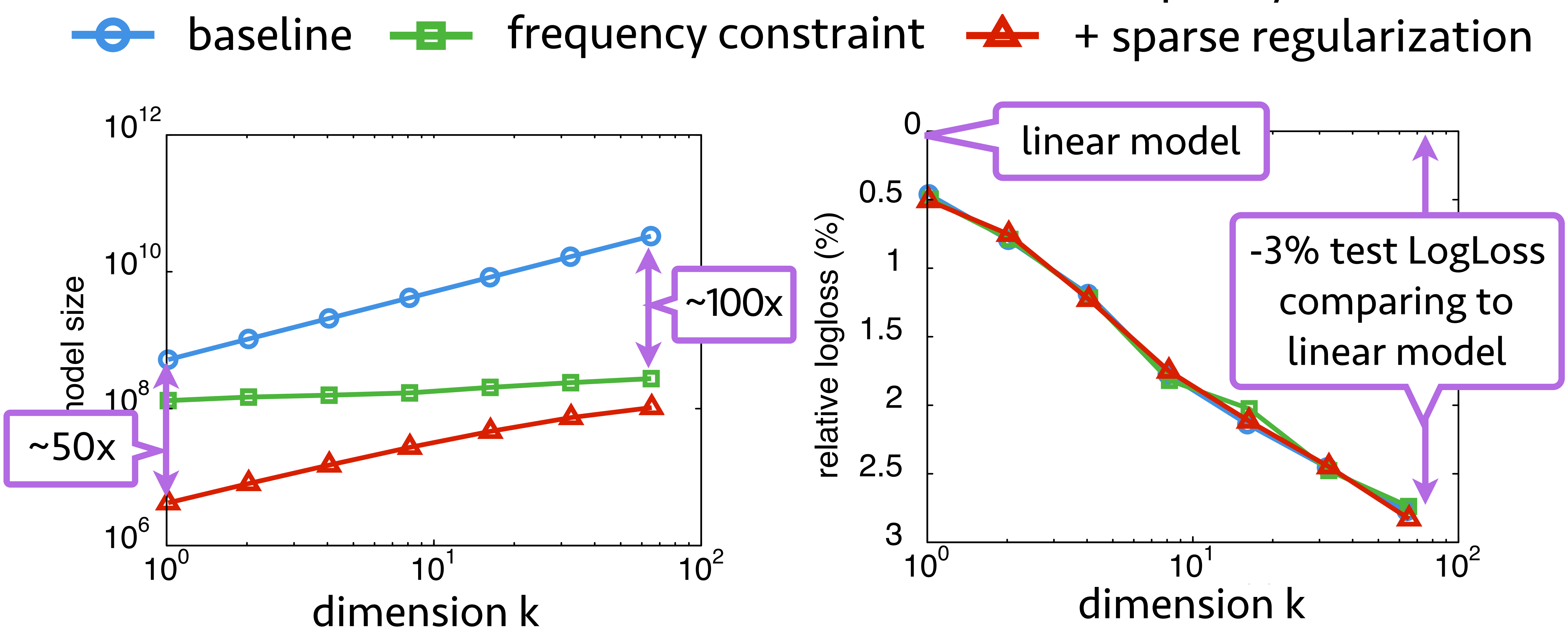
Adaptive Memory

- Criteo dataset: 1.5B examples, 360M features
- Run on 10 AWS EC2 machines



Adaptive Memory

- Criteo dataset: 1.5B examples, 360M features
- Run on 10 AWS EC2 machines



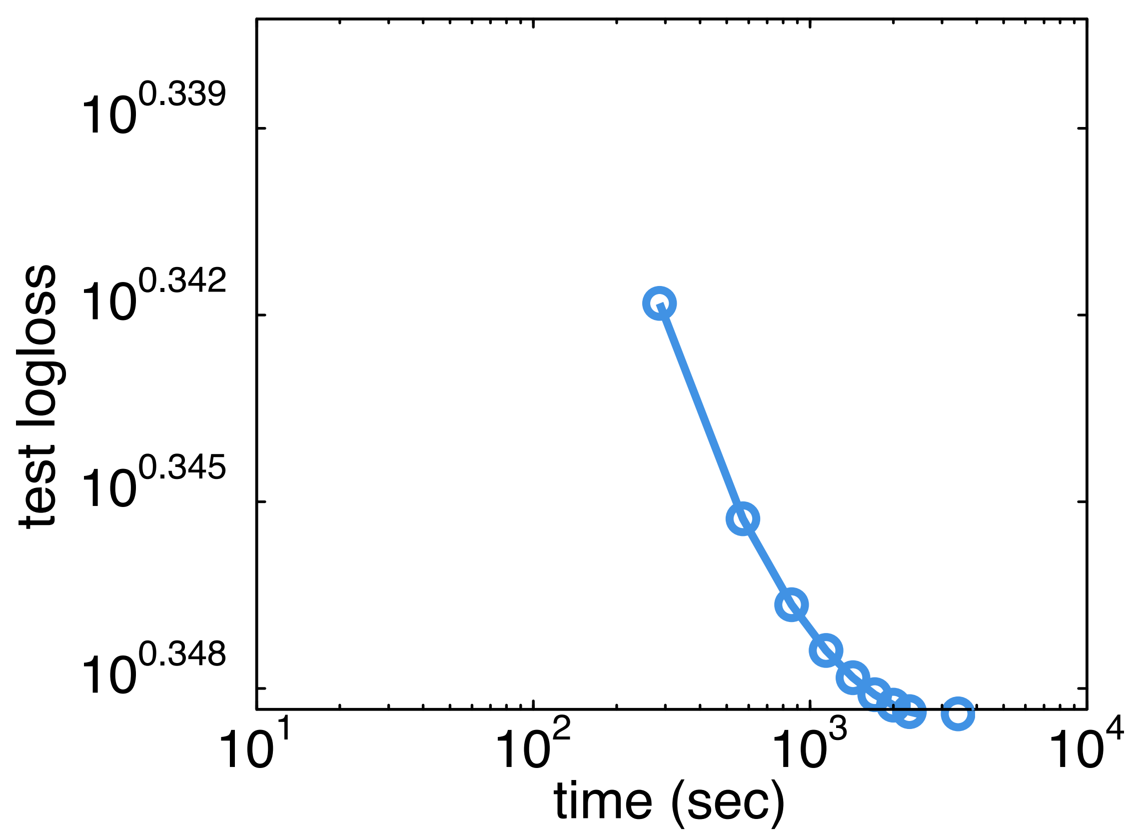
only 2x more computation cost
for FM with *k=100* comparing to linear model

Compare to LibFM

- LibFM is a widely used library for FM (Rendle et al)

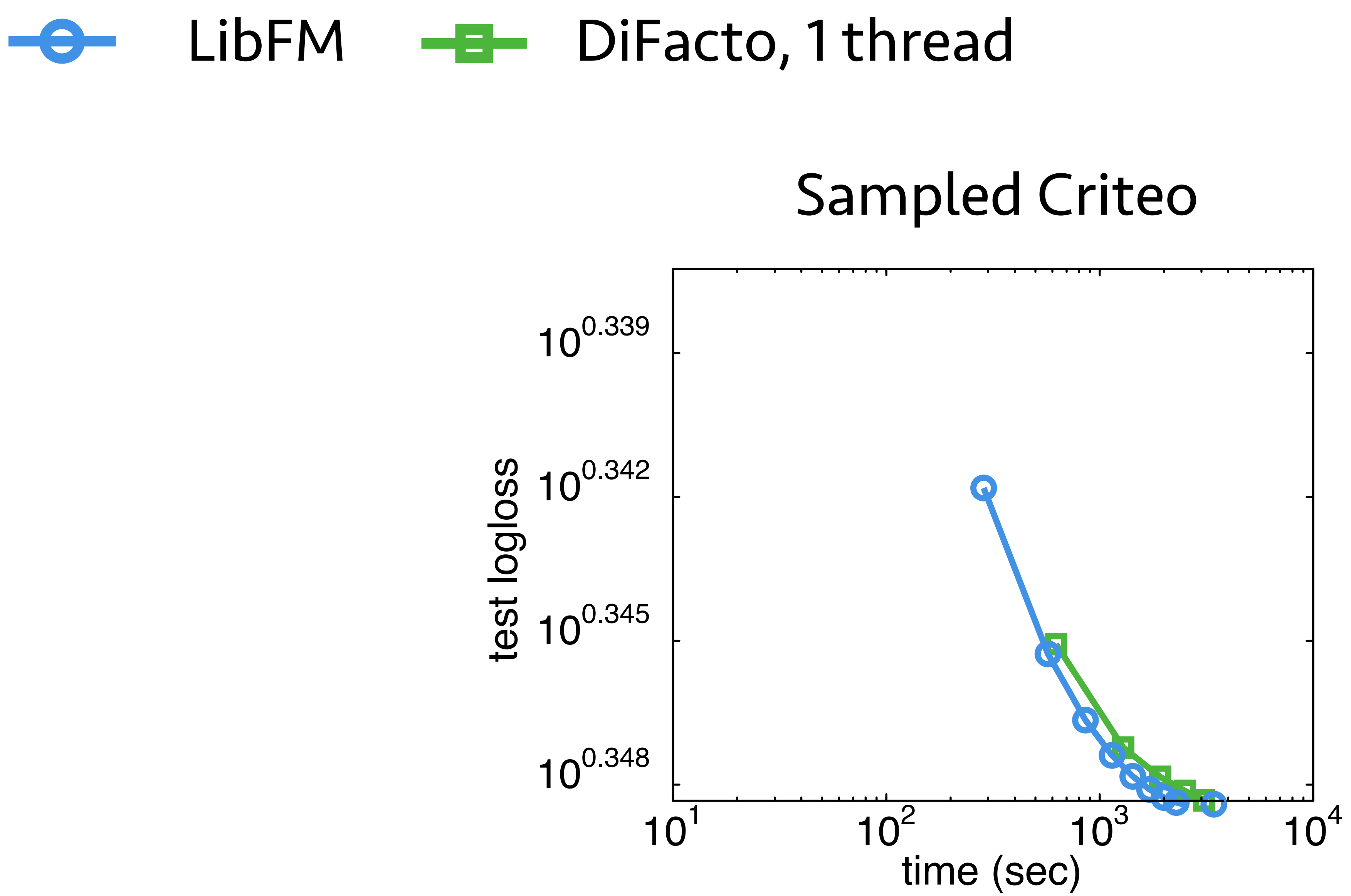
LibFM

Sampled Criteo



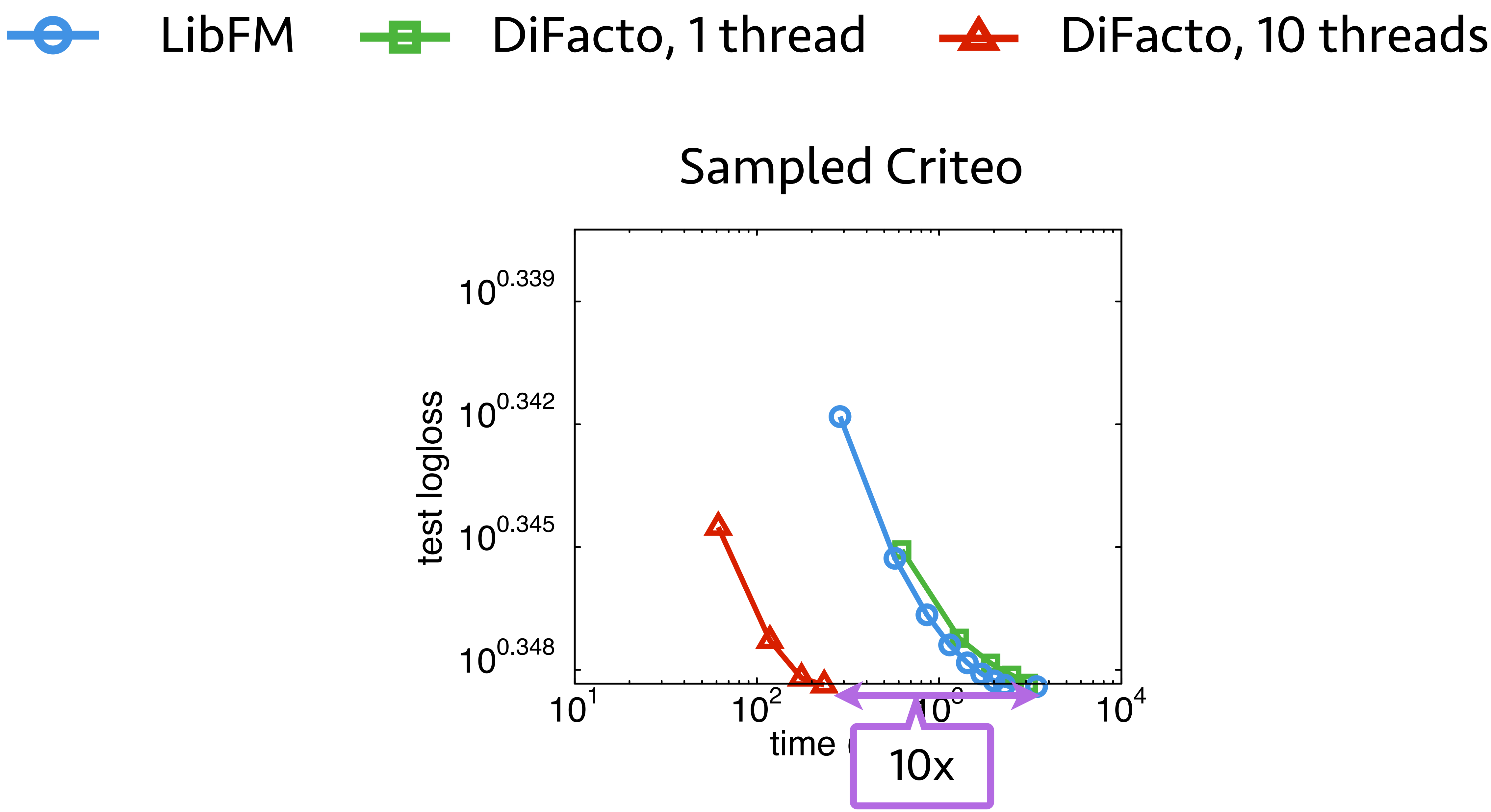
Compare to LibFM

- LibFM is a widely used library for FM (Rendle et al)



Compare to LibFM

- LibFM is a widely used library for FM (Rendle et al)

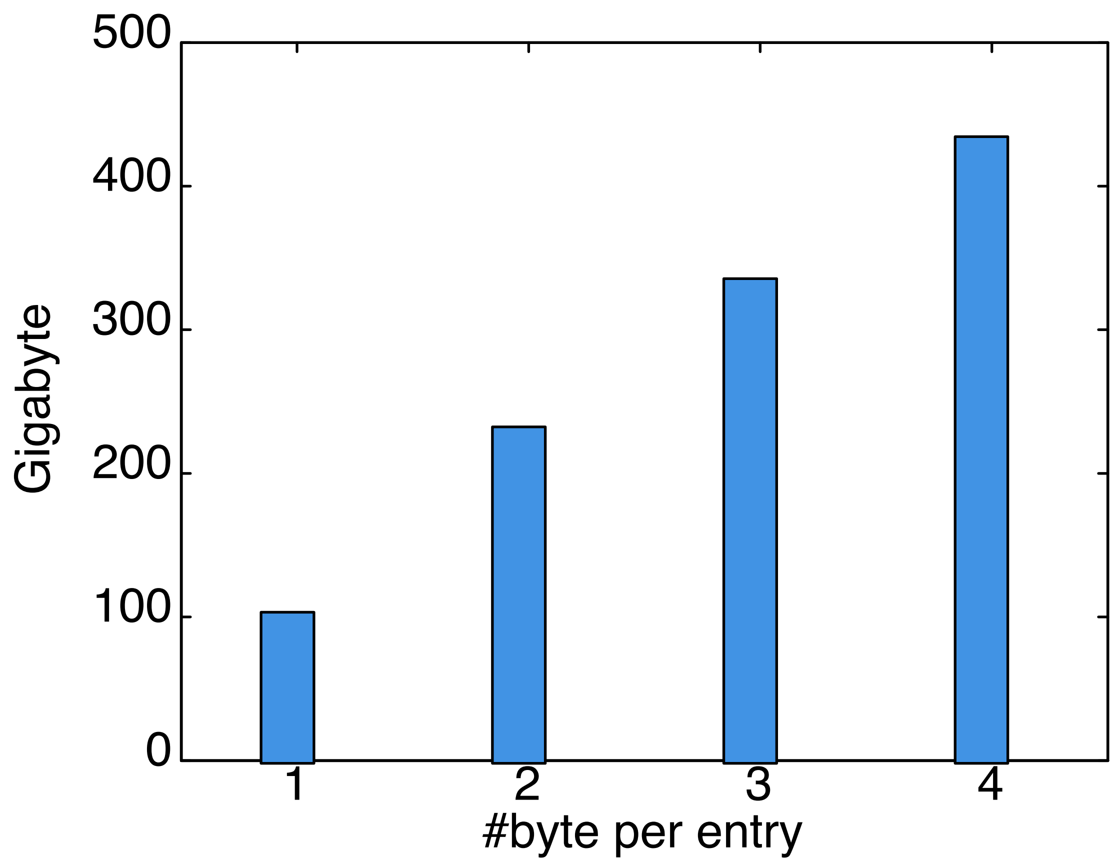


Fixed-point Compression

- Quantize float into n-bytes integer with randomized rounding during communication

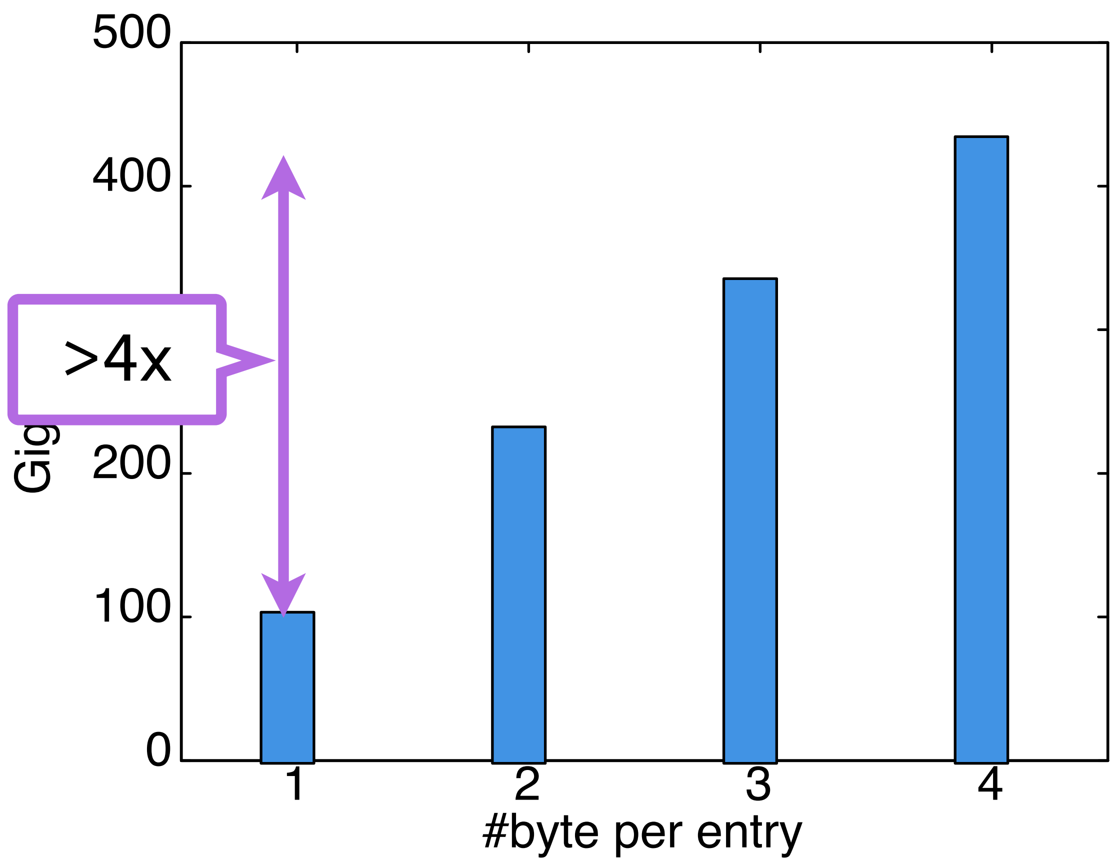
Fixed-point Compression

- Quantize float into n-bytes integer with randomized rounding during communication



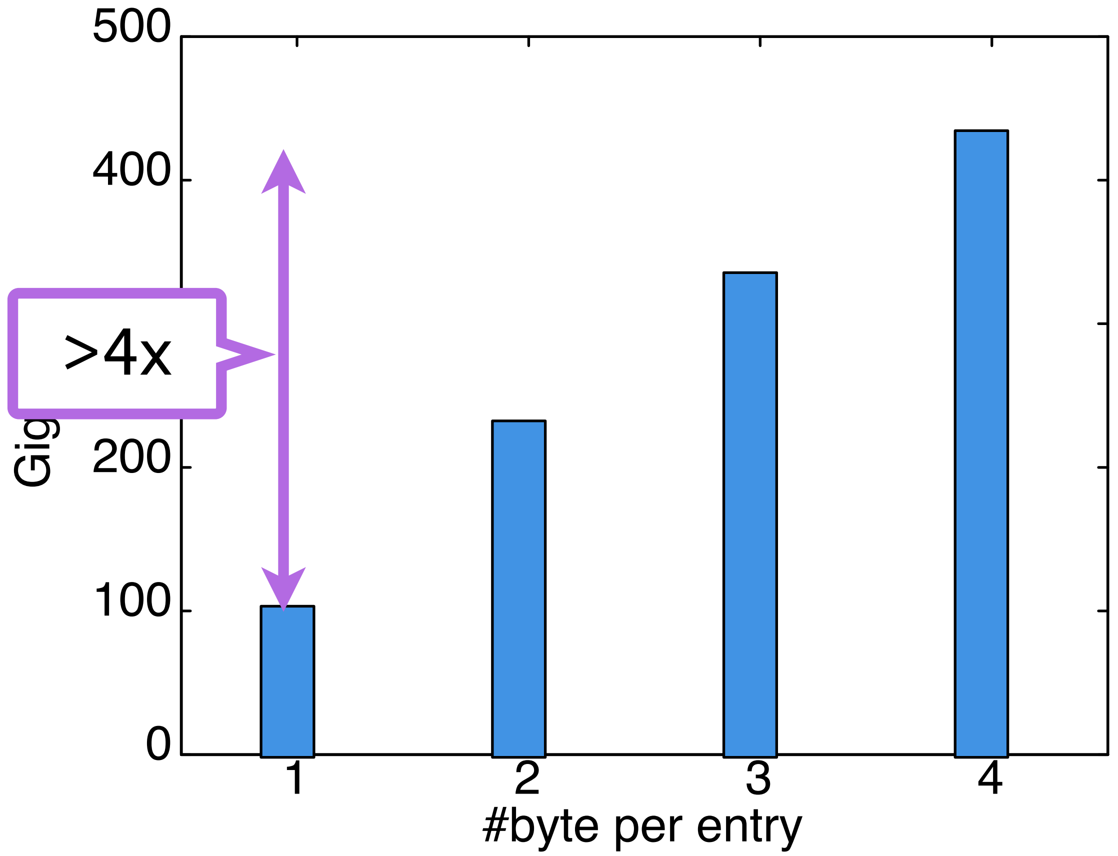
Fixed-point Compression

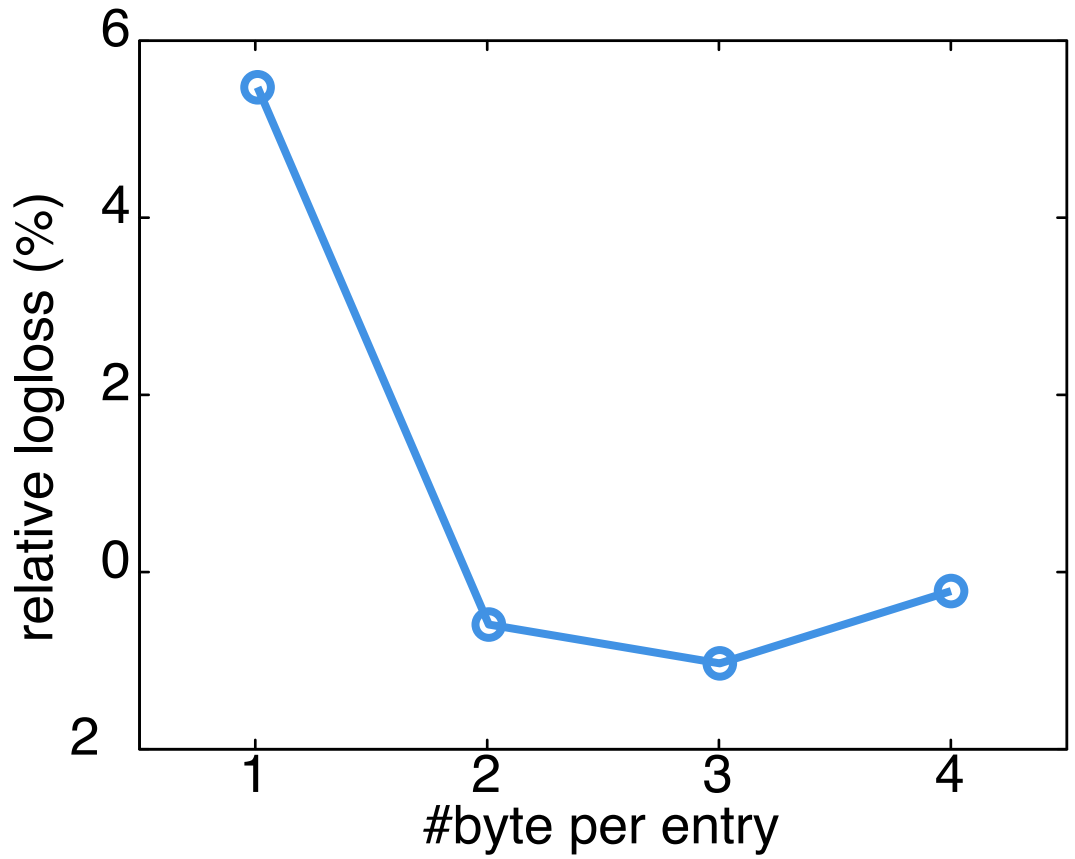
- Quantize float into n-bytes integer with randomized rounding during communication



Fixed-point Compression

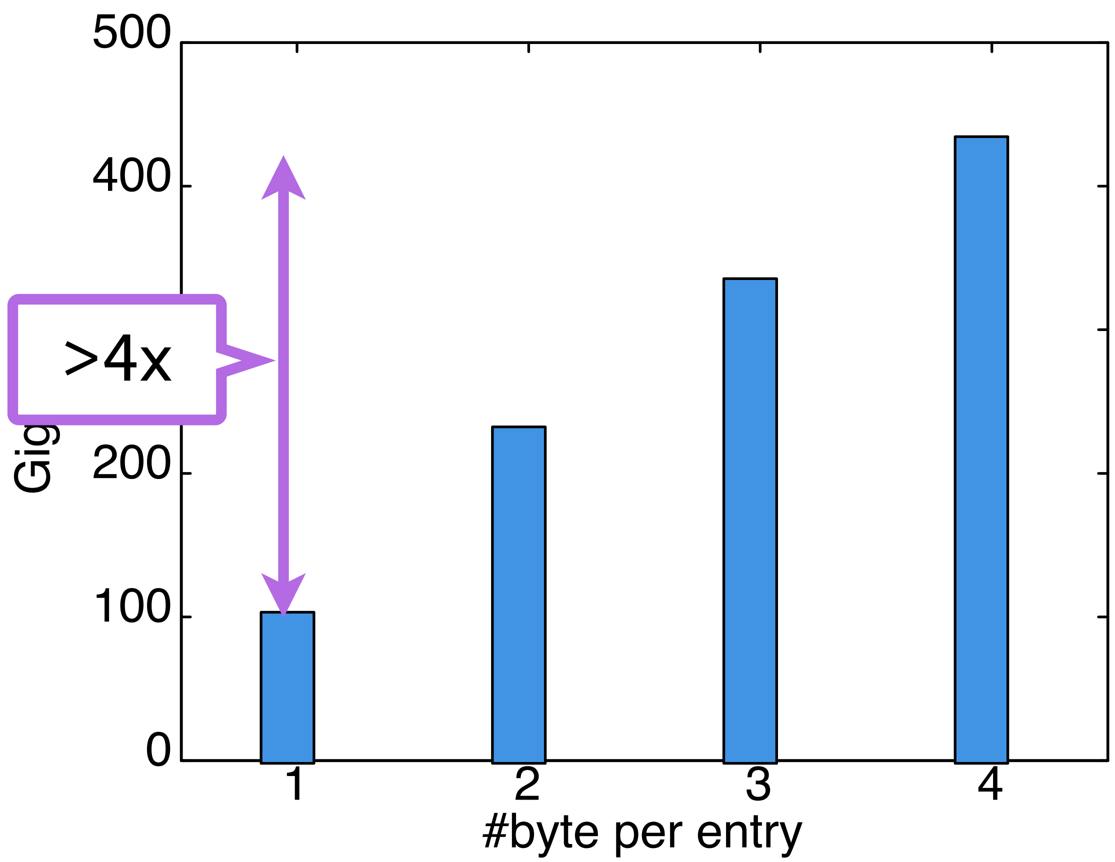
- Quantize float into n-bytes integer with randomized rounding during communication

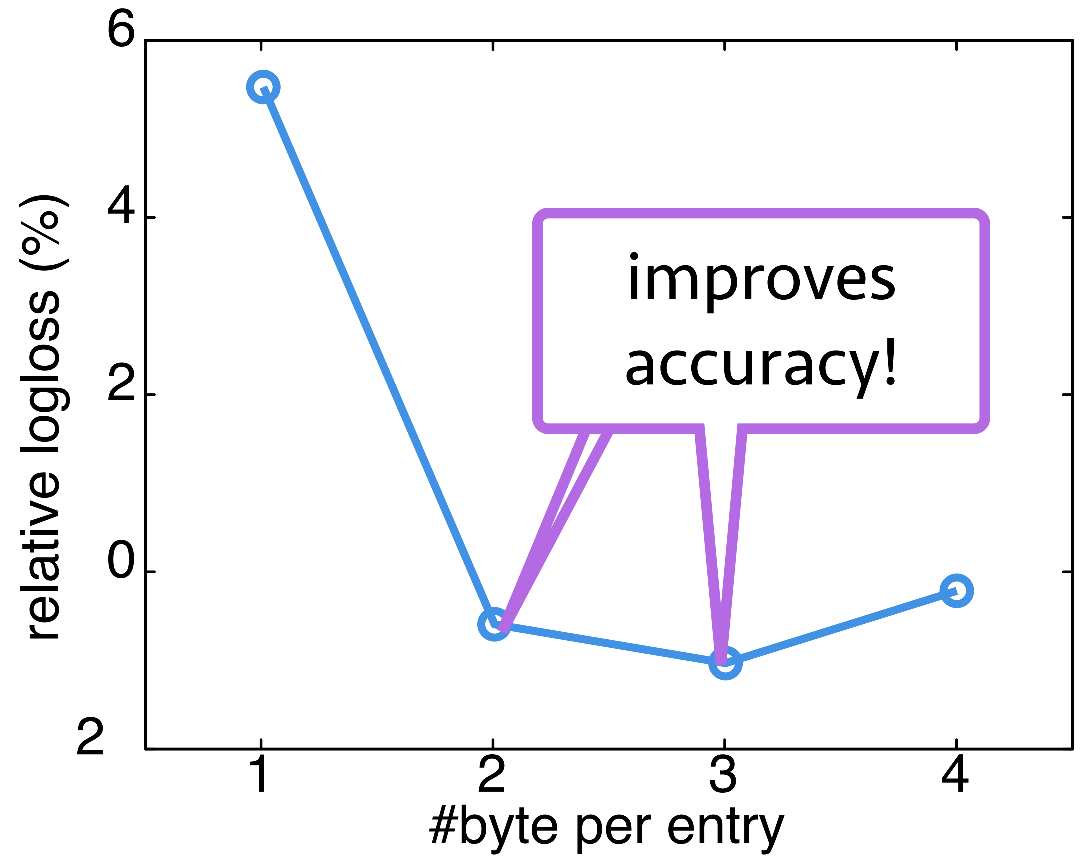




Fixed-point Compression

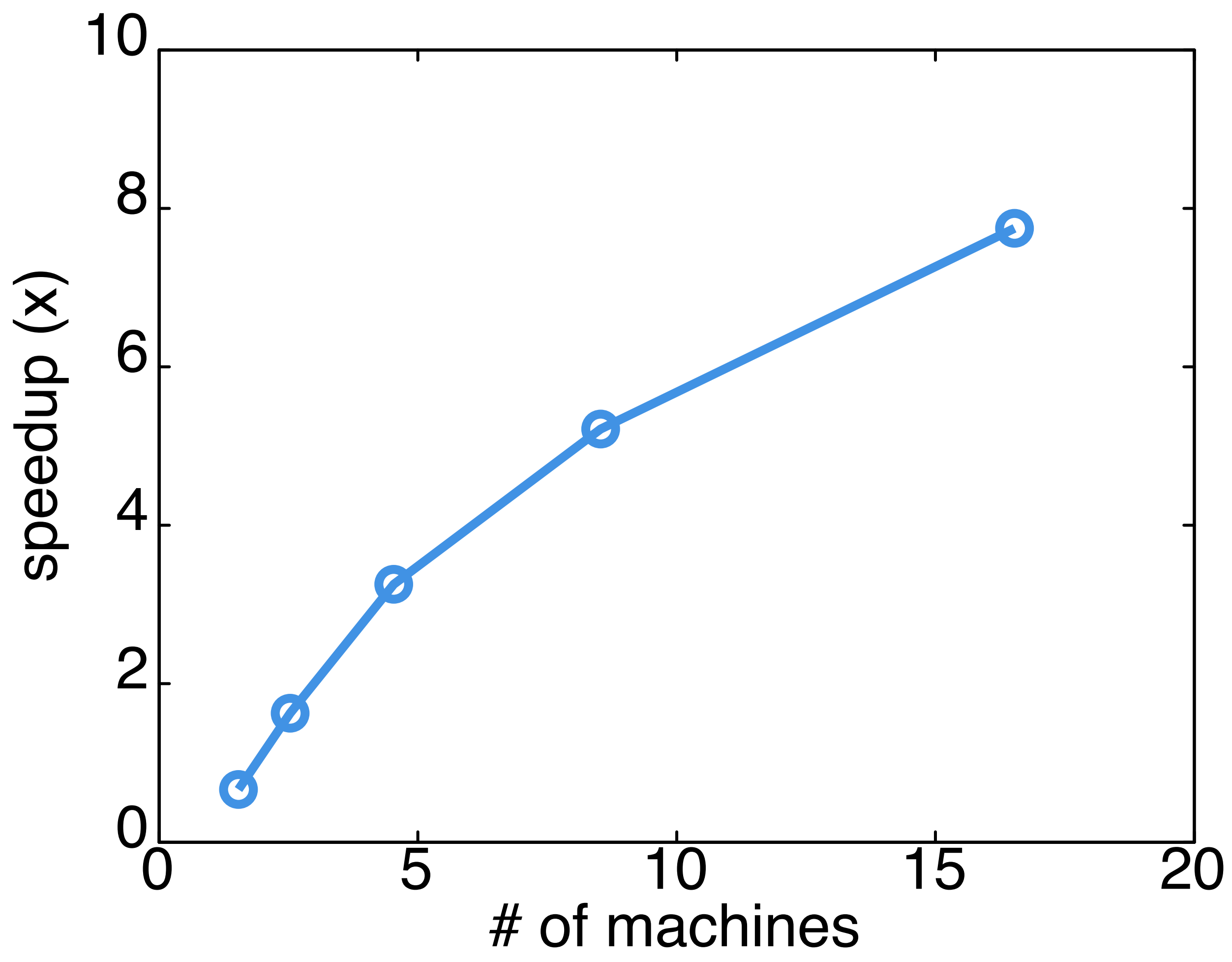
- Quantize float into n-bytes integer with randomized rounding during communication





Scalability

- Scaling from 1 machine to 16 machines



Conclusion

- **Goal:** scale factorization machine into large-scale datasets
- **Solution 1:** Data and model adaptive **regularizations** to reduce model capacity on unimportant features
- **Solution 2:** Efficient **distributed training** by asynchronous SGD using the parameter server framework
- **Results:** FM with a 100-dimension embedding provides significant accuracy improvement over linear model, with only ~2x more computation and storage cost
- **Codes** are publicly available at the DMLC project

dmlc DiFacto https://github.com/dmlc/difacto