GIANT: Globally Improved Approximate Newton Method for Distributed Optimization

Shusen Wang

UC Berkeley

Joint work with **Fred Roosta**, **Peng Xu**, and **Michael Mahoney**

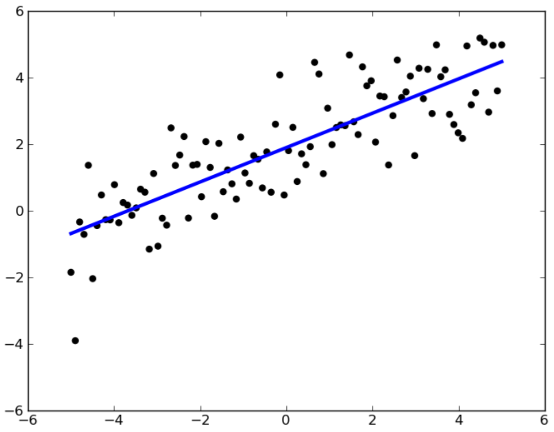
Background & Motivation

Optimization

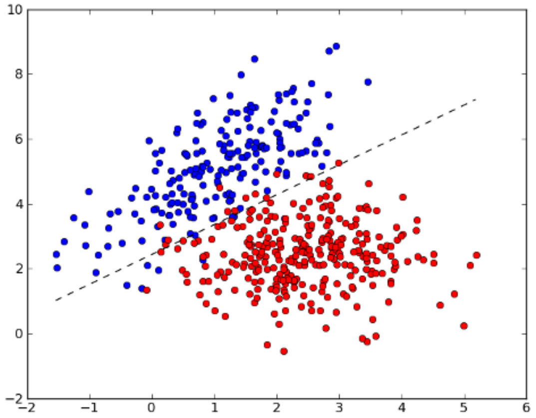
- We consider the *empirical risk minimization* problem:

$$\min_{\mathbf{w}\in\mathbb{R}^d} \left\{ f(\mathbf{w}) \triangleq \frac{1}{n}\sum_{j=1}^{n} l(\mathbf{w}\,;\, \mathbf{x}_j, y_j) + r(\mathbf{w}) \right\}$$

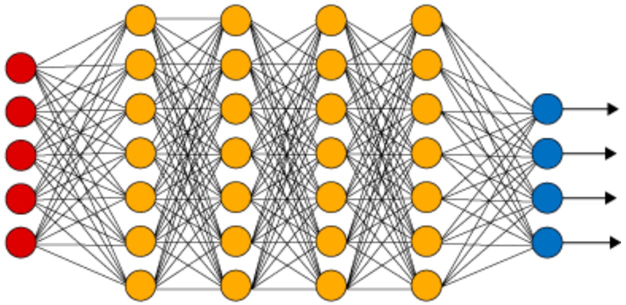
- Examples:



Linear Regression



Linear Classification



Neural Networks

Optimization

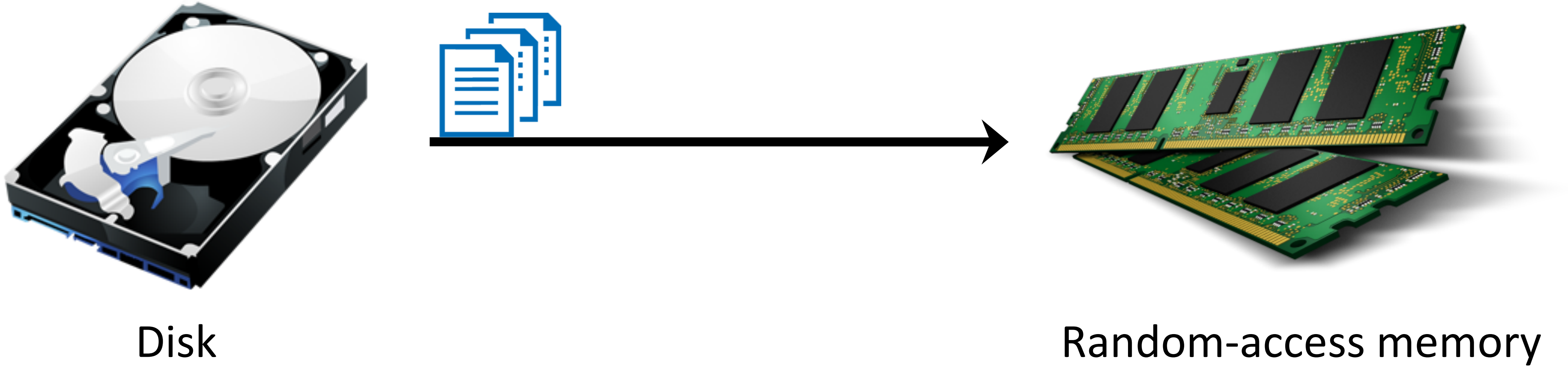
- How to solve the optimization problem $\min_{\mathbf{w}} f(\mathbf{w})$?

   1. Write some code / find a package.

Optimization

- How to solve the optimization problem $\min_{\mathbf{w}} f(\mathbf{w})$?

1. Write some code / find a package.
2. Load data to memory.



Optimization

- How to solve the optimization problem $\min_{\mathbf{w}} f(\mathbf{w})$?

1. Write some code / find a package.
2. Load data to memory.
3. Run the code.

Optimization

- How to solve the optimization problem $\min_{\mathbf{w}} f(\mathbf{w})$?

    1. Write some code / find a package.
    2. Load data to memory.
    3. Run the code.

- What if the data do not fit in memory?

Optimization

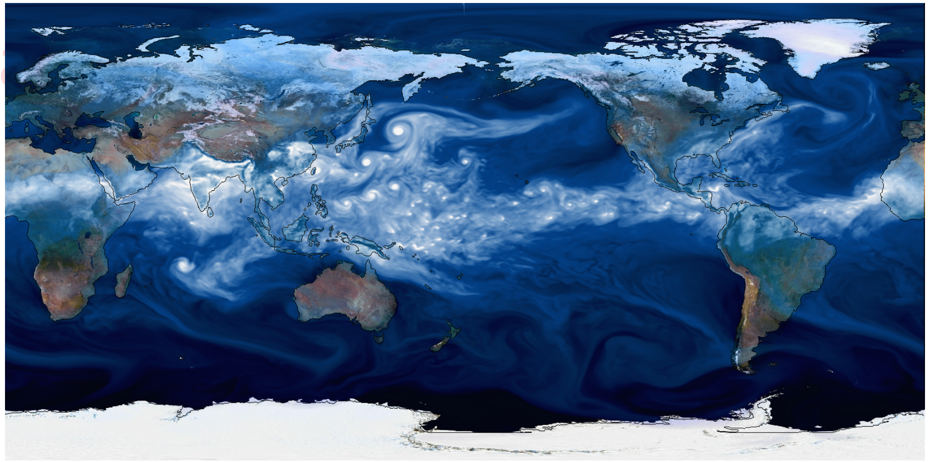
- How to solve the optimization problem $\min_{\mathbf{w}} f(\mathbf{w})$?

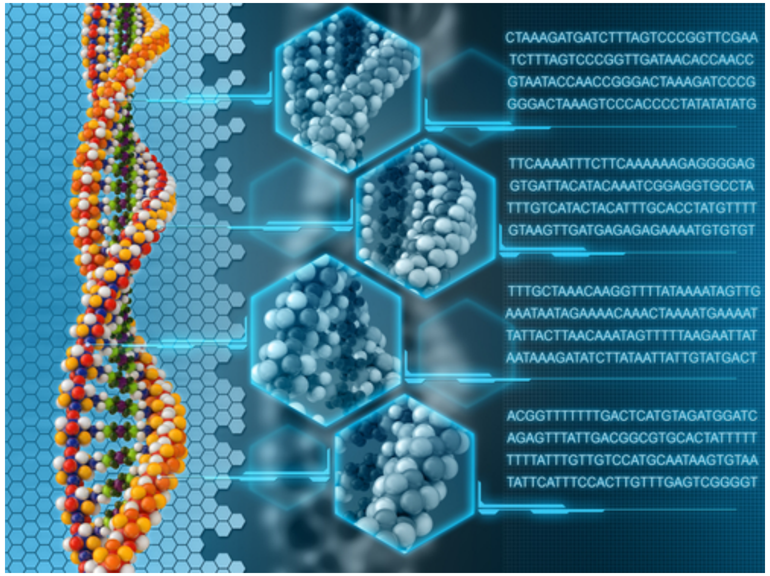
  1. Write some code / find a package.
  2. Load data to memory.
  3. Run the code.

- What if the data do not fit in memory?
- What if the computation is too expensive for a single machine?

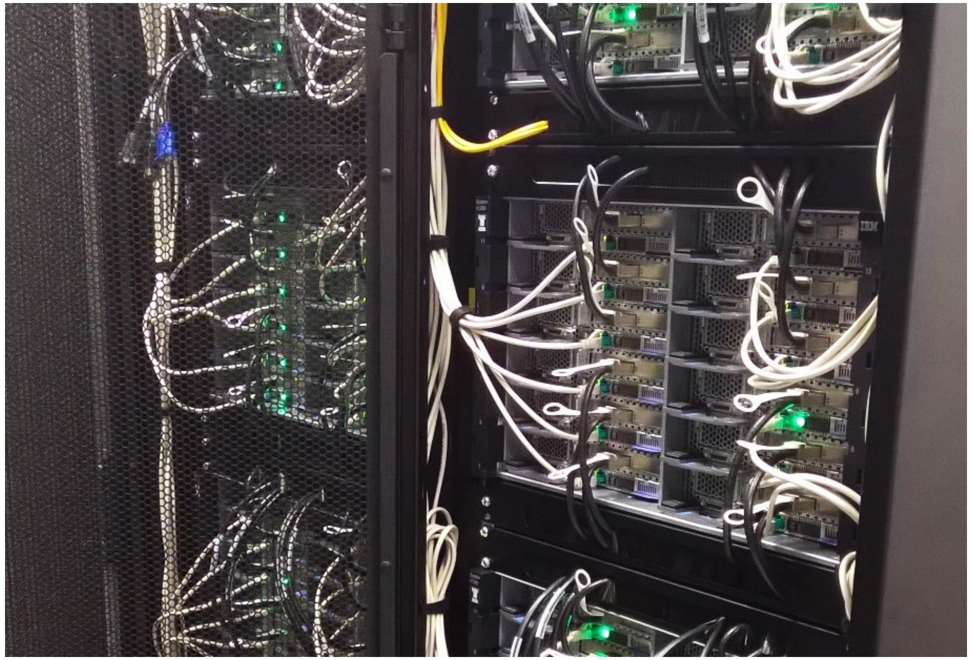
Optimization

- How to s... the opt...ati... problem $\min_{\mathbf{w}} f($...
  1. ...me ... / find a package
  2. Loa... dat... to ...ory
  3. Run the code.
- What if the data do not fit in memory?





Distributed Optimization



Computer clusters

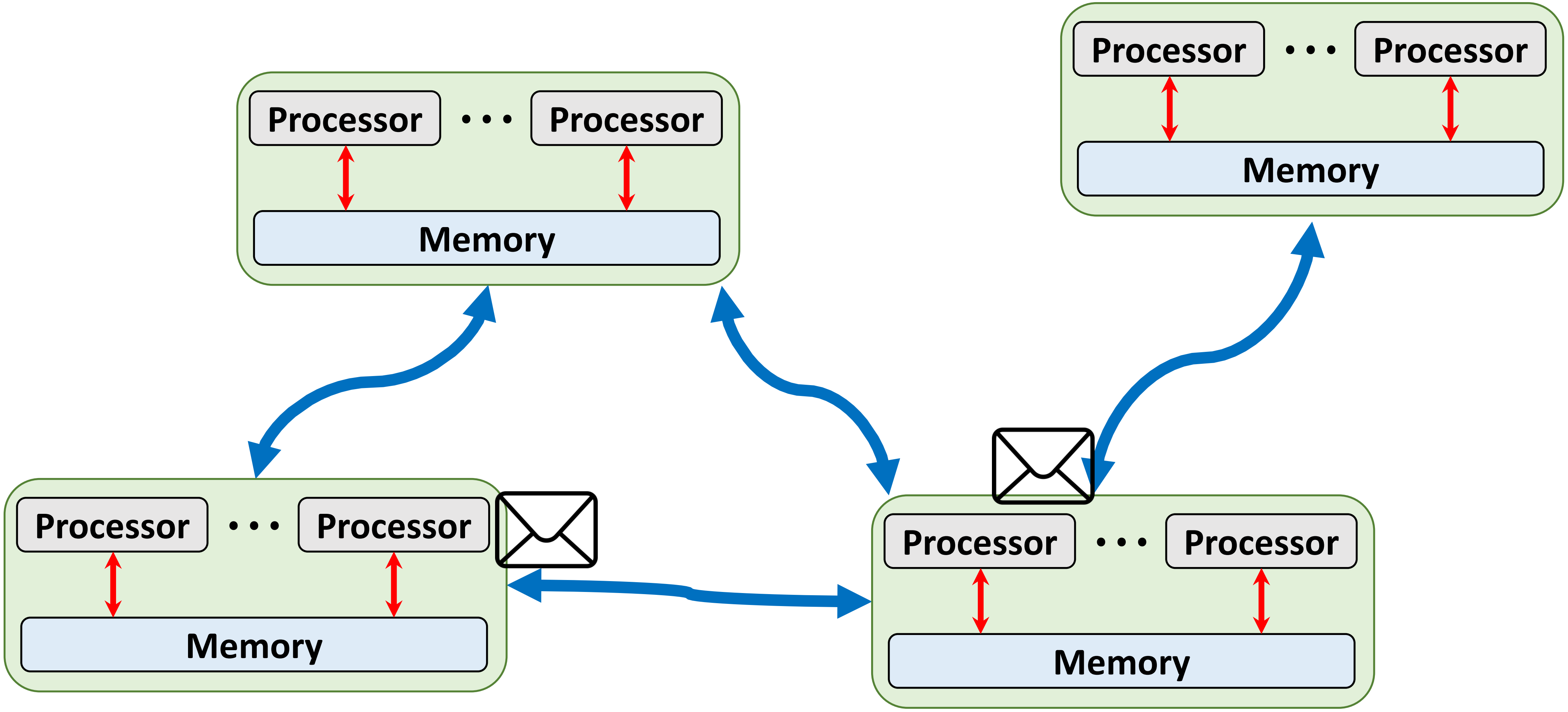
Distributed Optimization

AMAZON EC2

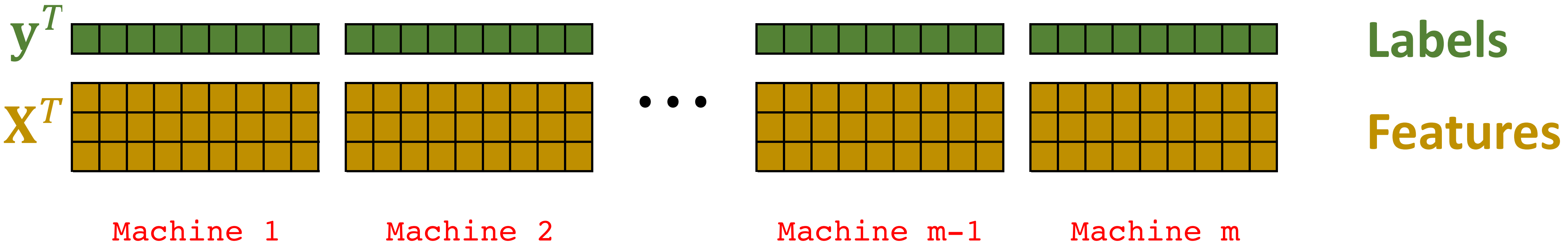
amazon web services

Supercomputer

Distributed Optimization



Distributed Optimization



- $(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_n, y_n)$ are split among m machines.

Distributed Optimization

Ideally,

- $\frac{1}{m}$ of the data fit in the memory of one machine;
- each machine does $\frac{1}{m}$ of the computation ➡ mx Speedup .

Distributed Optimization

Ideally,

- $\frac{1}{m}$ of the data fit in the memory of one machine;
- each machine does $\frac{1}{m}$ of the computation ➡ ~~mx Speedup~~ .

Do not overlook the communication!

Distributed Optimization: Example

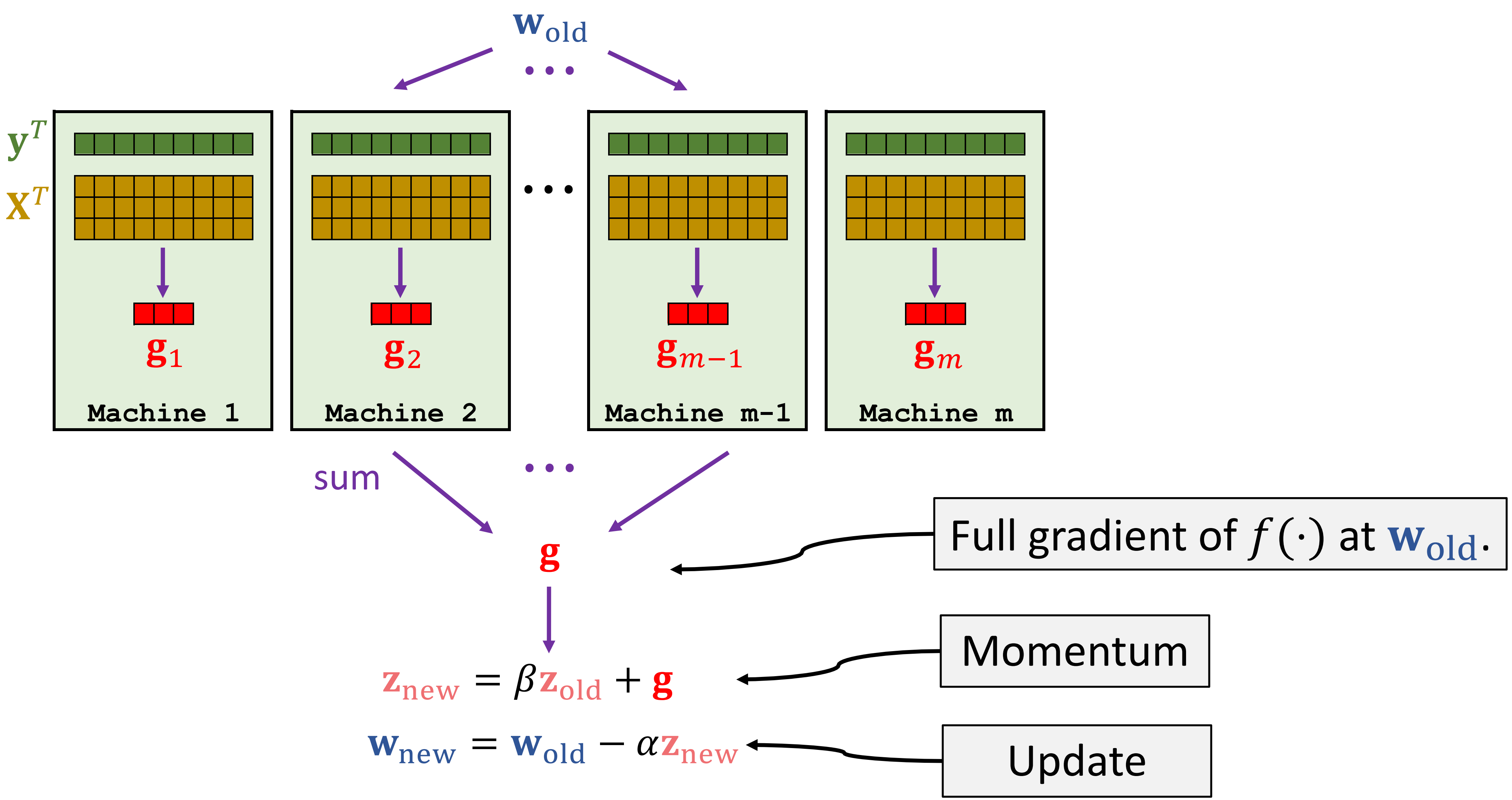
Solve the problem:

$$\min_{\mathbf{w}\in\mathbb{R}^d} \left\{ f(\mathbf{w}) \triangleq \frac{1}{n}\sum_{j=1}^{n} l\big(\mathbf{w}\,;\,\mathbf{x}_j, y_j\big) + r(\mathbf{w}) \right\}$$

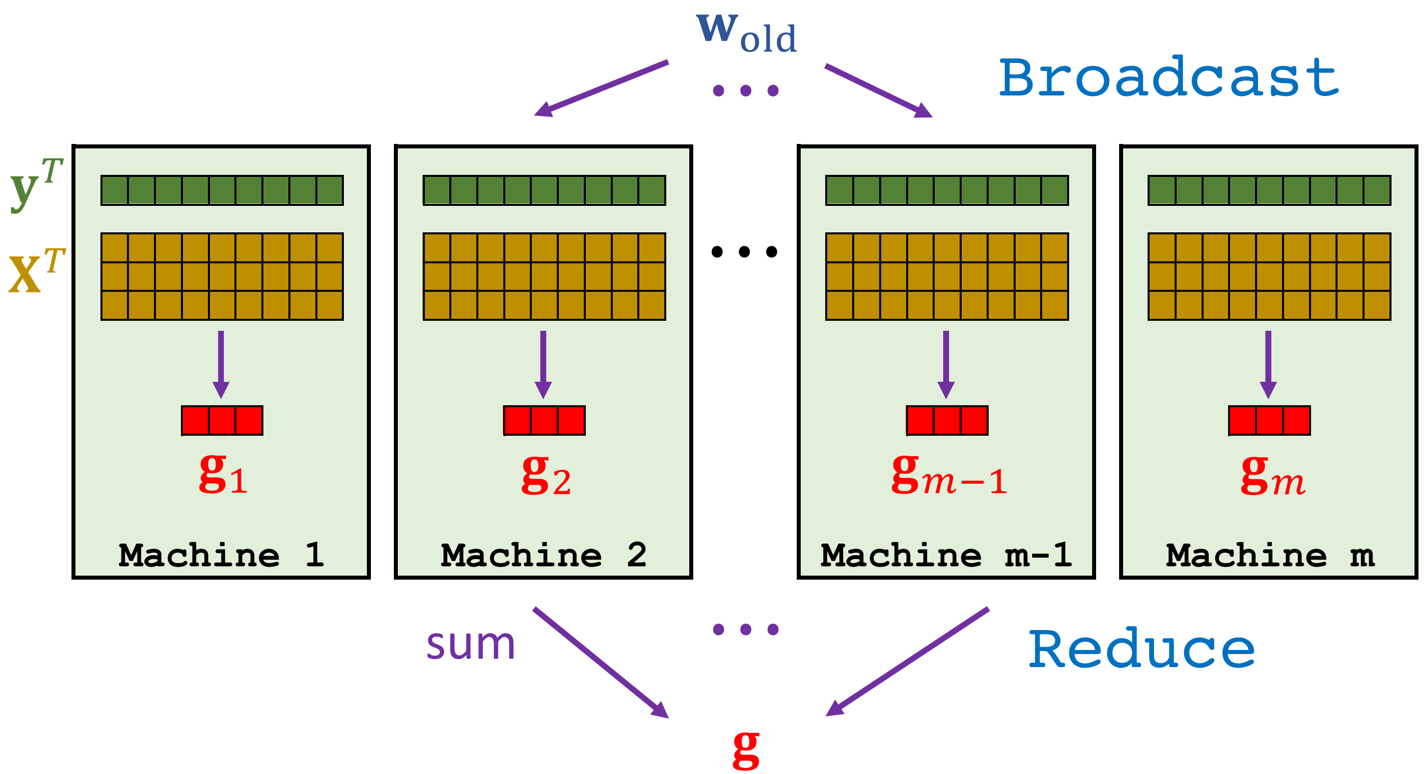
Accelerated Gradient Descent (AGD) repeats:

1. Compute gradient: $g = \nabla f(\mathbf{w}_{\text{old}});$
2. Update momentum: $\mathbf{z}_{\text{new}} = \beta \mathbf{z}_{\text{old}} + g, \quad 0 \leq \beta < 1;$
3. Update model: $\mathbf{w}_{\text{new}} = \mathbf{w}_{\text{old}} - \alpha \mathbf{z}_{\text{new}}.$

Warm-up: Distributed AGD



Warm-up: Distributed AGD

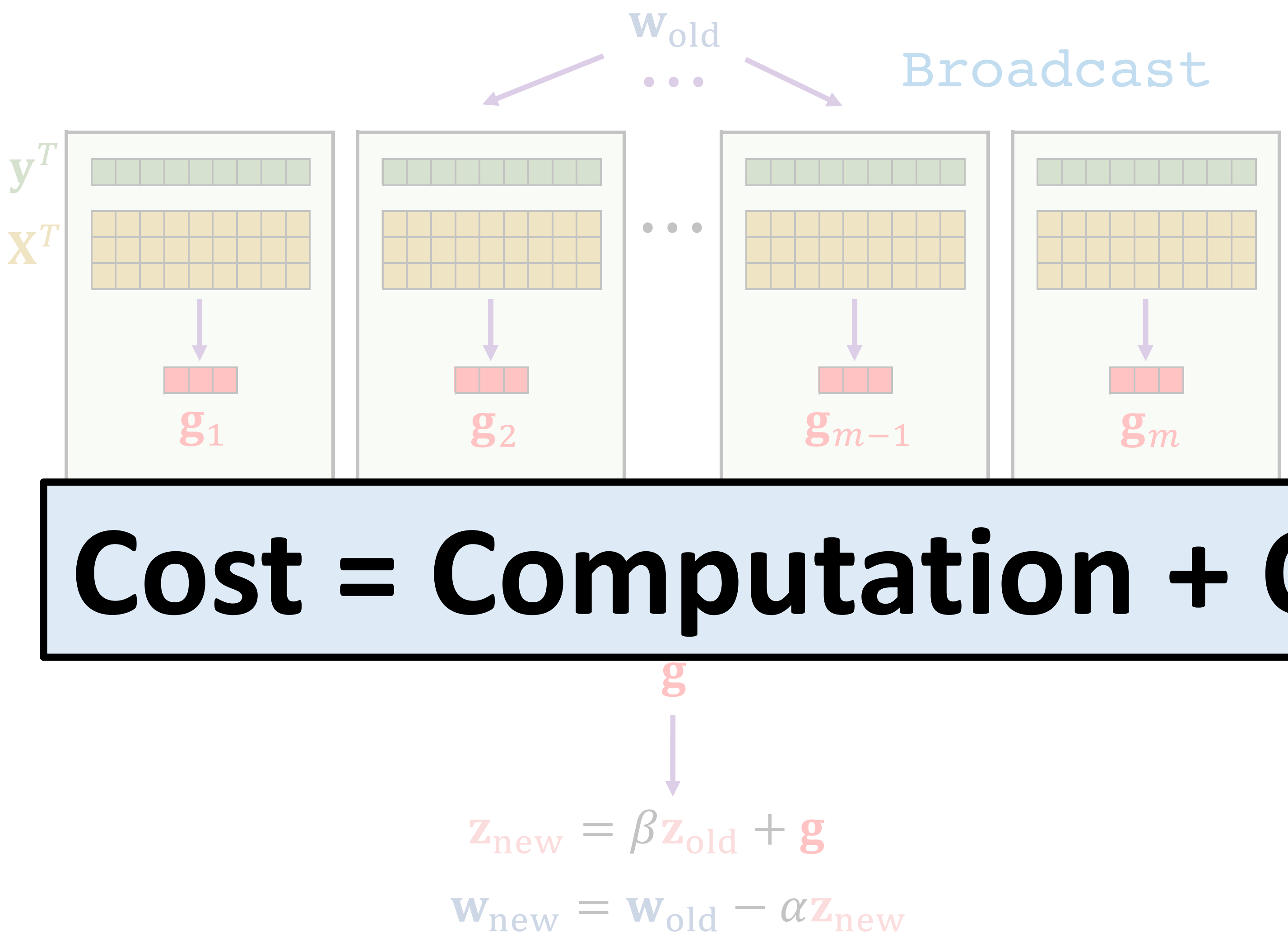


$$\mathbf{z}_{\text{new}} = \beta \mathbf{z}_{\text{old}} + \mathbf{g}$$

$$\mathbf{w}_{\text{new}} = \mathbf{w}_{\text{old}} - \alpha \mathbf{z}_{\text{new}}$$

- Time complexity: $O\left(\frac{nd}{m}\right)$ FLOPs per iteration.
- One Broadcast and one Reduce per iteration.
- Lots of iterations to converge ➔ lots of communications.

Warm-up: Distributed AGD



- Time complexity: $O\left(\frac{nd}{m}\right)$ FLOPs per iteration.
- One Broadcast and one Reduce
- Lots of iterations to converge → lots of communications.

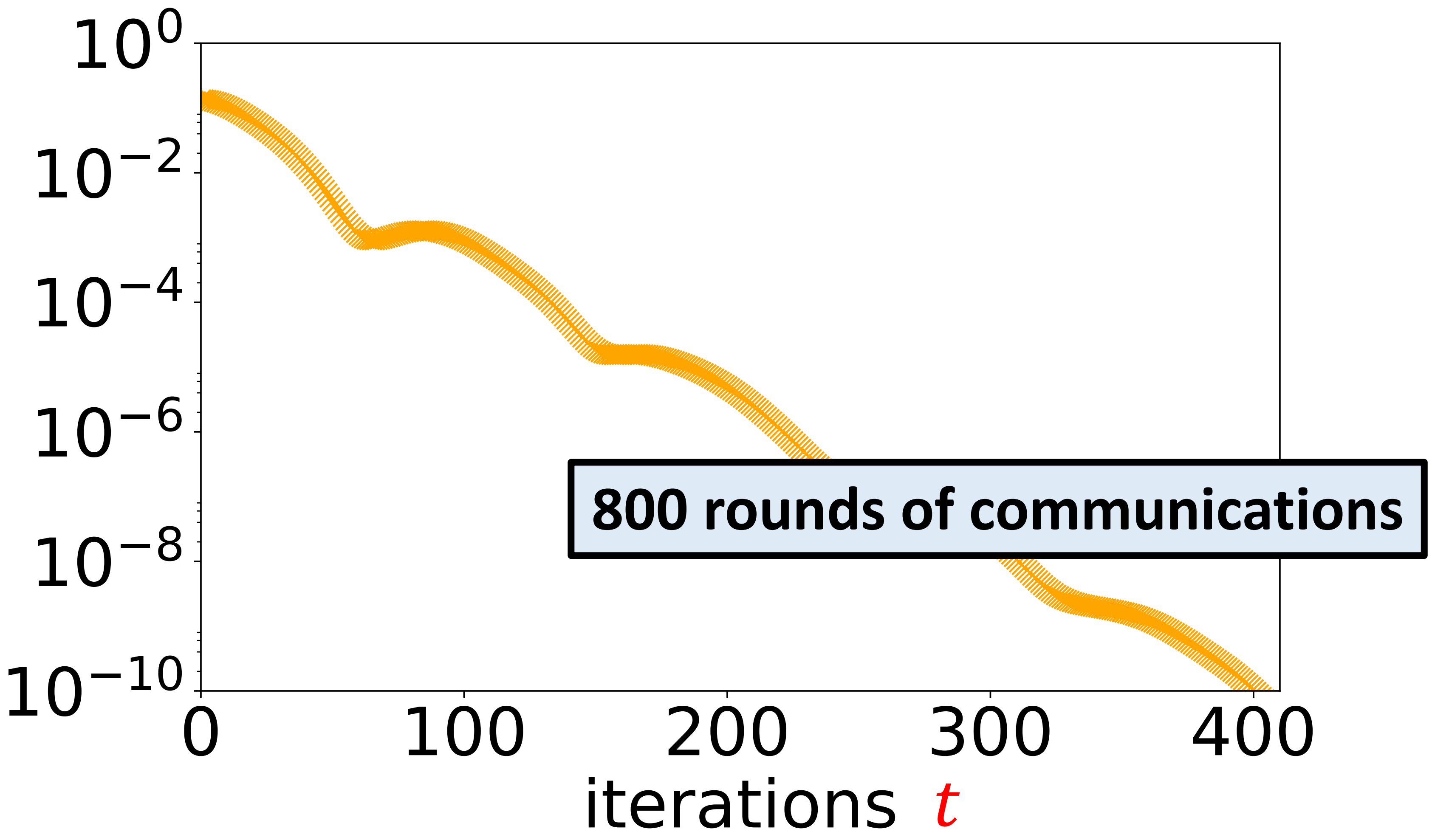
Cost = Computation + Communication

AGD for ℓ_2-Regularized Logistic Regression

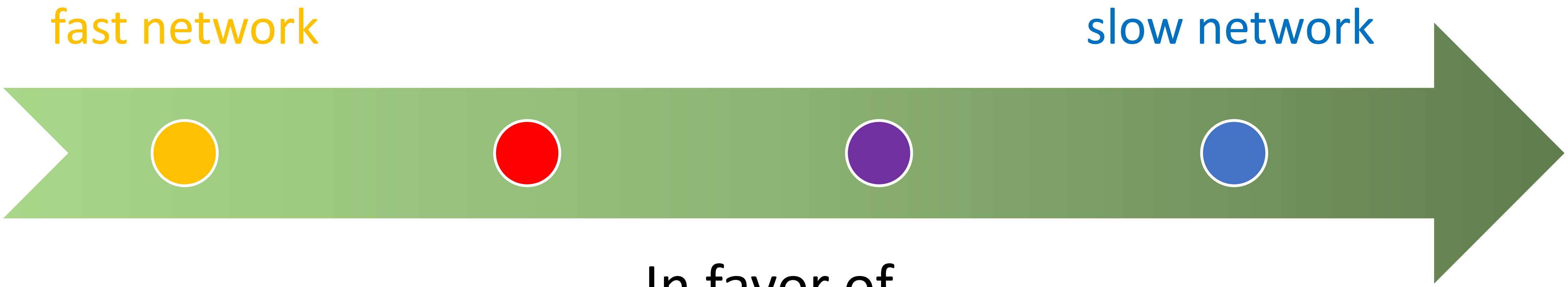
$$f(\mathbf{w}_t) - f(\mathbf{w}^\star)$$

iteration

optimal solution



FLOPs versus Communications



In favor of

- SGD
- SCD

- Mini-Batch SGD
- Block SCD

- Full GD
- L-BFGS

Single Machine	Cluster	Federated Learning

FLOPs versus Communications

In favor of

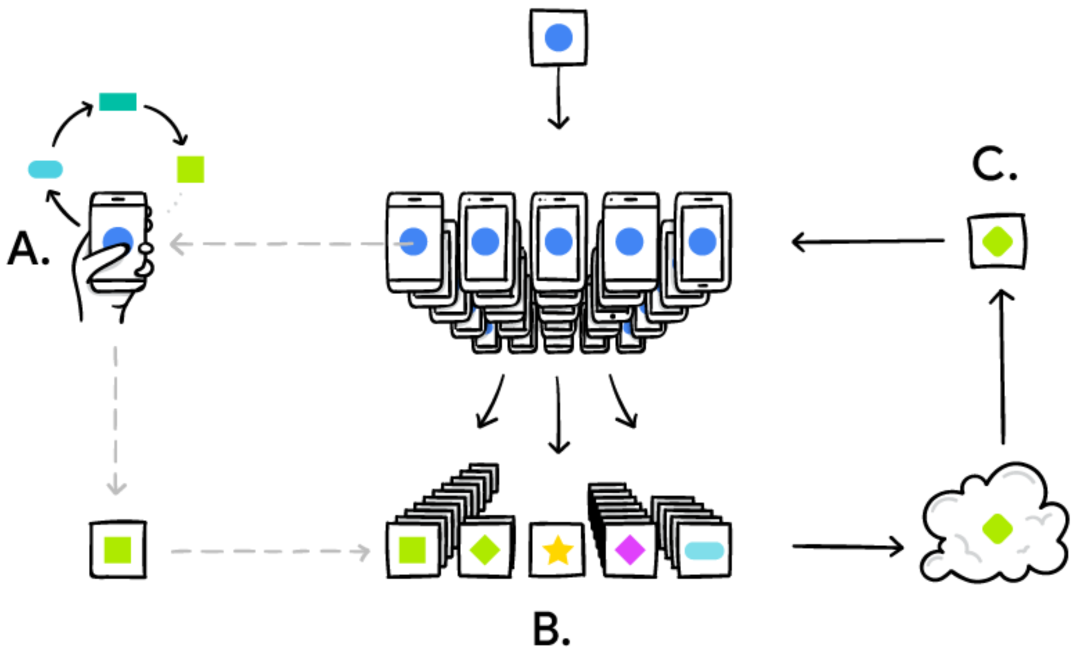
- SGD
- SCD

- Mini-Batch SGD
- Block SCD

- Full GD
- L-BFGS

Single Machine	Cluster	Federated Learning

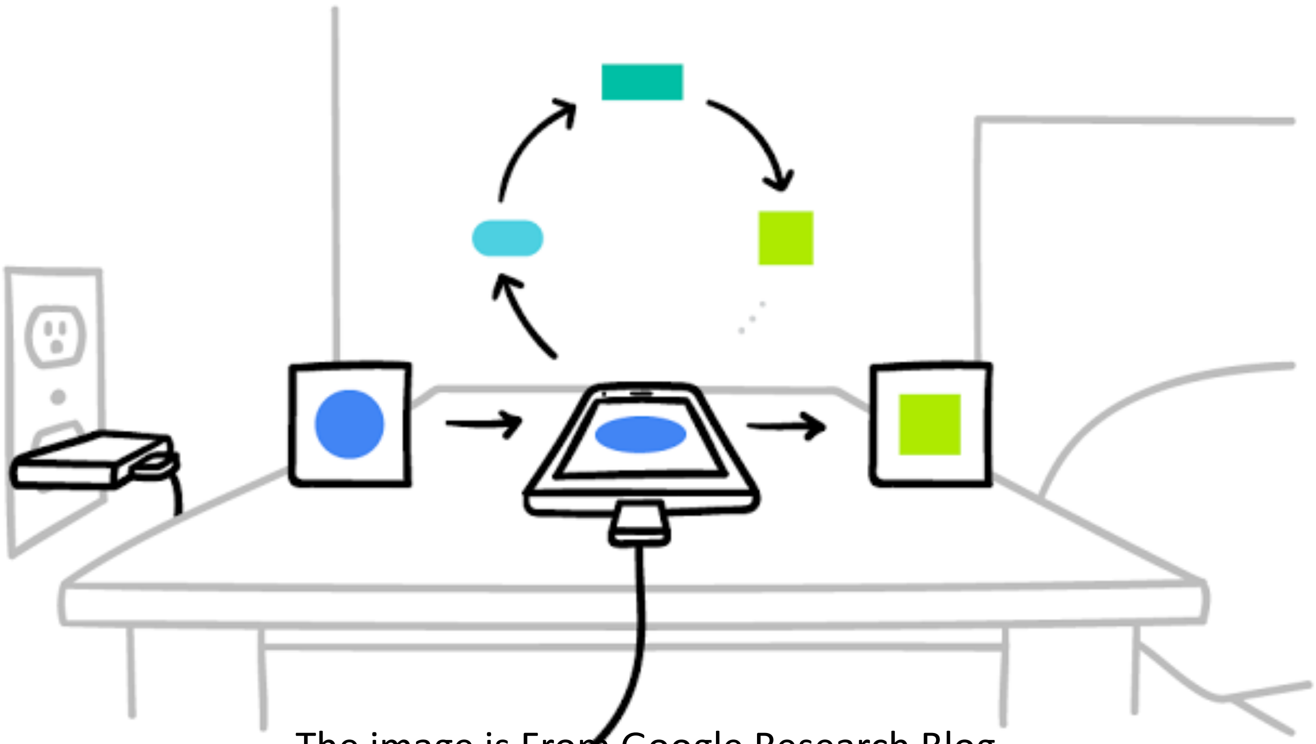
FLOPs versus Communications



The image is From Google Research Blog

Single Machine | Cluster | **Federated Learning**

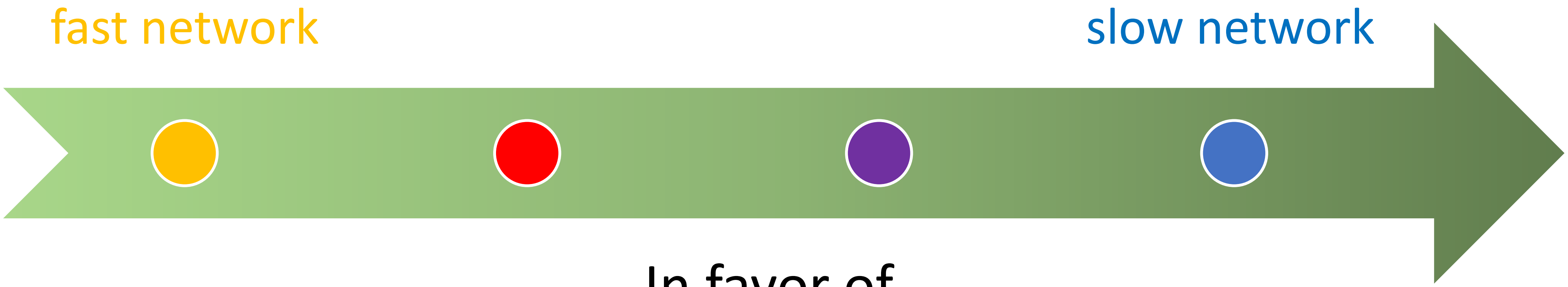
FLOPs versus Communications



The image is From Google Research Blog

Single Machine | Cluster | Federated Learning

FLOPs versus Communications



In favor of

- SGD
- SCD

- Mini-Batch SGD
- Block SCD

- Full GD
- L-BFGS

?

?

Single Machine	Cluster	Federated Learning

Distributed Optimization

Summary

1. For big-data problems, distributed optimization is very useful.

2. If the network is slow, then communication is the bottleneck.

    - Recall: `Cost ≈ Computation + Communication`

Communication-Efficient Optimization

Motivation

Basic ideas:

1. Let worker machines do lots of local computations.
2. Communicate as few as possible.

Prior Work

Existing communication-efficient methods:

- CoCoA
- DANE
- AIDE

⋮

They make assumptions, e.g.,

- objective function is strongly convex and Lipschitz smooth

Reference:

1. Smith, Forte, Ma, Takac, Jordan, & Jaggi. CoCoA: A General Framework for Communication-Efficient Distributed Optimization.
2. Shamir, Srebro, & Zhang. Communication Efficient Distributed Optimization using an Approximate Newton-type Method. In *ICML*, 2014.
3. Reddi, Konečný, Richtárik, Póczós, & Smola. AIDE: Fast and Communication Efficient Distributed Optimization.

⋮

Prior Work

Existing communication-efficient methods:

- CoCoA
- DANE
- AIDE

⋮

Recall **Accelerated Gradient Descent (AGD)**

- $O\left(\sqrt{\kappa}\log\frac{1}{\epsilon}\right)$ iterations
- 2 communications per iteration
- $O\left(\frac{nd}{m}\right)$ FLOPs per iterations

Baseline!

Prior Work

Existing communication-efficient methods:

- CoCoA
- DANE
- AIDE

⋮

Recall **Accelerated Gradient Descent (AGD)**

- $O\left(\sqrt{\kappa}\log\frac{1}{\epsilon}\right)$ iterations
- 2 communications per iteration
- $O\left(\frac{nd}{m}\right)$ FLOPs per iterations

Baseline!

Do their convergence bounds beat **AGD**?

Prior Work

Existing communication-efficient methods:

- CoCoA
- DANE
- AIDE

⋮

Recall **Accelerated Gradient Descent (AGD)**

- $O\left(\sqrt{\kappa}\log\frac{1}{\epsilon}\right)$ iterations
- 2 communications per iteration
- $O\left(\frac{nd}{m}\right)$ FLOPs per iterations

Baseline!

Do their convergence bounds beat **AGD**?

- In terms of **communication, NO!**
- In terms of **computation, NO!**

Prior Work

Existing communication-efficient methods:

- CoCoA
- DANE
- AIDE

If the objective function is quadratic, then DANE = GIANT!

GIANT: Overview

Globally Improved Approximate Newton (GIANT)

- GIANT is a distributed 2nd-order method.
- Each iteration has 4 rounds of communications.
  - `Broadcast` or `Reduce` of one vector.
- Much faster convergence than AGD in terms of communication.
  - Assume the objective function is strongly convex and Lipschitz smooth.

Globally Improved Approximate Newton (GIANT)

- Assume the objective function is strongly convex and Lipschitz smooth.

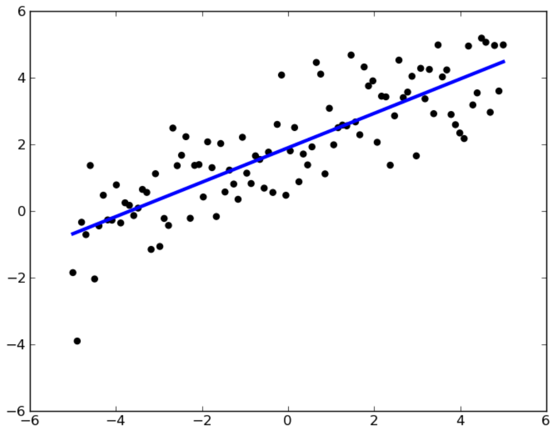
Globally Improved Approximate Newton (GIANT)

- Assume the objective function is strongly convex and Lipschitz smooth.

- **Examples**

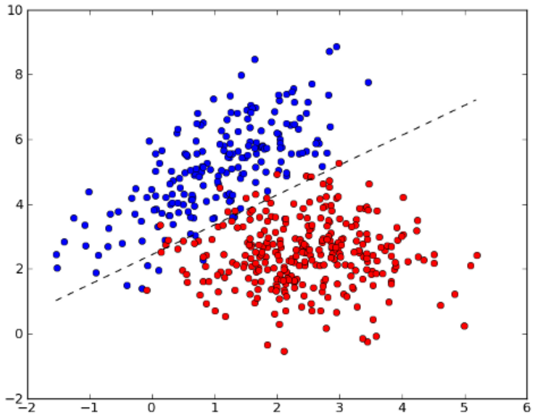
Linear regression

$$f(\mathbf{w}) = \frac{1}{2n}\sum_{j=1}^{n}\left(\mathbf{w}^T\mathbf{x}_j - y_j\right)^2 + \gamma\|\mathbf{w}\|_2^2$$



Logistic regression

$$f(\mathbf{w}) = \frac{1}{2n}\sum_{j=1}^{n}\log\left(1 + e^{-y_j\mathbf{w}^T\mathbf{x}_j}\right) + \gamma\|\mathbf{w}\|_2^2$$



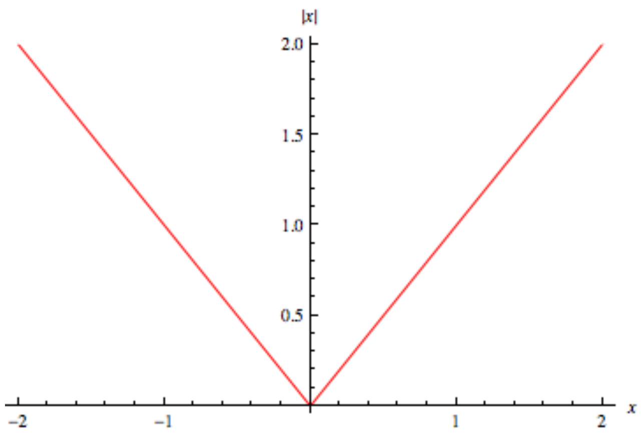
Globally Improved Approximate Newton (GIANT)

- Assume the objective function is strongly convex and Lipschitz smooth.

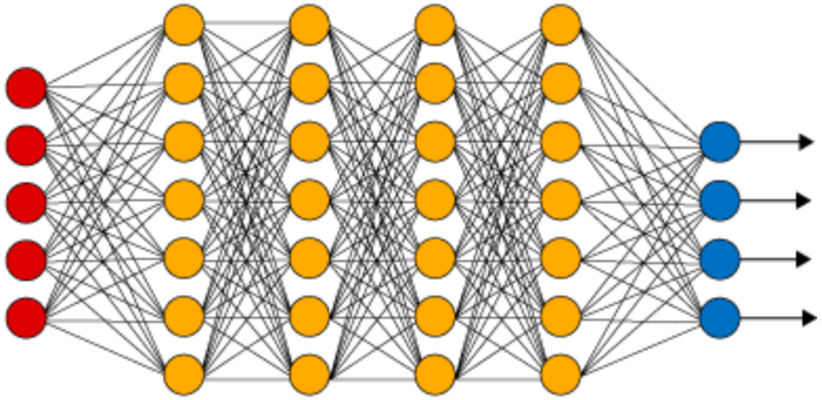
- **Counter-examples**

LASSO

$$f(\mathbf{w}) = \frac{1}{2n}\sum_{j=1}^{n}\left(\mathbf{w}^T\mathbf{x}_j - y_j\right)^2 + \gamma\|\mathbf{w}\|_1$$



Neural Networks



Globally Improved Approximate Newton (GIANT)

- Assume the objective function is strongly convex and Lipschitz smooth.

- **Counter-examples**

LASSO

The regularization is non-smooth!

Neural Networks

The objective is non-convex!

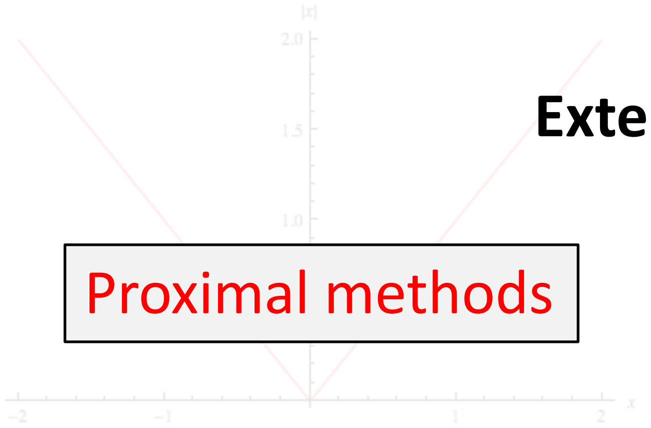
Globally Improved Approximate Newton (GIANT)

- Assume the objective function is strongly convex and Lipschitz smooth.

- **Counter-examples**

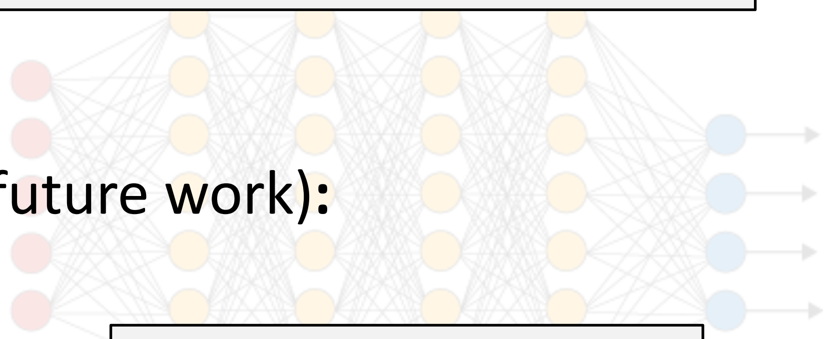
LASSO

The regularization is non-smooth!



Neural Networks

The objective is non-convex!



Extensions of GIANT (our future work):

Proximal methods

Trust-region method

Globally Improved Approximate Newton (GIANT)

- Assume the objective function is strongly convex and Lipschitz smooth.

GIANT: Algorithm Description

Warm-up: Newton-CG

- Repeat until convergence
  1. Compute gradient $\mathbf{g}$ and Hessian $\mathbf{H}$;
  2. Solve $\mathbf{Hp} = \mathbf{g}$ by running tens/hundreds of CG steps;
  3. Update $\mathbf{w} \leftarrow \mathbf{w} - \alpha \mathbf{p}$ (find α by line search).

GIANT: Algorithm Derivation

Recall: Newton's direction is $\mathbf{p} = \mathbf{H}^{-1}\mathbf{g}$.

In parallel, form the approximations:

$\widetilde{\mathbf{H}}_1 \approx \mathbf{H}$ $\widetilde{\mathbf{H}}_2 \approx \mathbf{H}$ $\cdots$ $\widetilde{\mathbf{H}}_{m-1} \approx \mathbf{H}$ $\widetilde{\mathbf{H}}_m \approx \mathbf{H}$

In parallel, compute

$\widetilde{\mathbf{p}}_1 = \widetilde{\mathbf{H}}_1^{-1}\mathbf{g}$ $\widetilde{\mathbf{p}}_2 = \widetilde{\mathbf{H}}_2^{-1}\mathbf{g}$ $\cdots$ $\widetilde{\mathbf{p}}_{m-1} = \widetilde{\mathbf{H}}_{m-1}^{-1}\mathbf{g}$ $\widetilde{\mathbf{p}}_m = \widetilde{\mathbf{H}}_m^{-1}\mathbf{g}$

$\widetilde{\mathbf{p}} = \frac{1}{m}\sum_i \widetilde{\mathbf{p}}_i = \left(\frac{1}{m}\sum_i \widetilde{\mathbf{H}}_i^{-1}\right)\mathbf{g}$ approximates $\mathbf{p} = \mathbf{H}^{-1}\mathbf{g}$

GIANT: Algorithm Derivation

Recall: Newton's direction is $\mathbf{p} = \mathbf{H}^{-1}\mathbf{g}$.

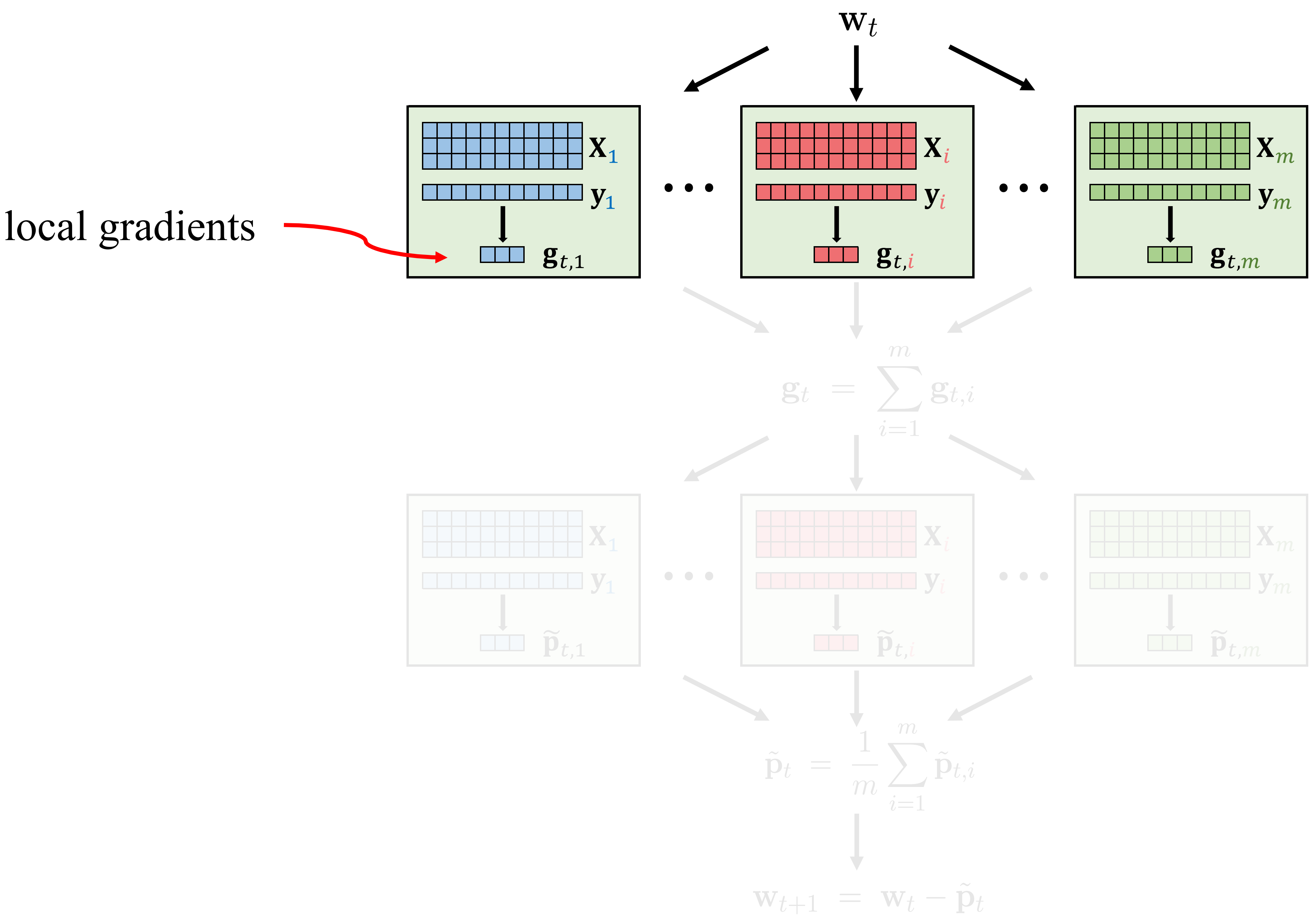
$\tilde{\mathbf{p}} = \frac{1}{m}\sum_i \tilde{\mathbf{p}}_i = \left(\frac{1}{m}\sum_i \widetilde{\mathbf{H}}_i^{-1}\right)\mathbf{g}$ approximates $\mathbf{p} = \mathbf{H}^{-1}\mathbf{g}$

GIANT: Algorithm Derivation

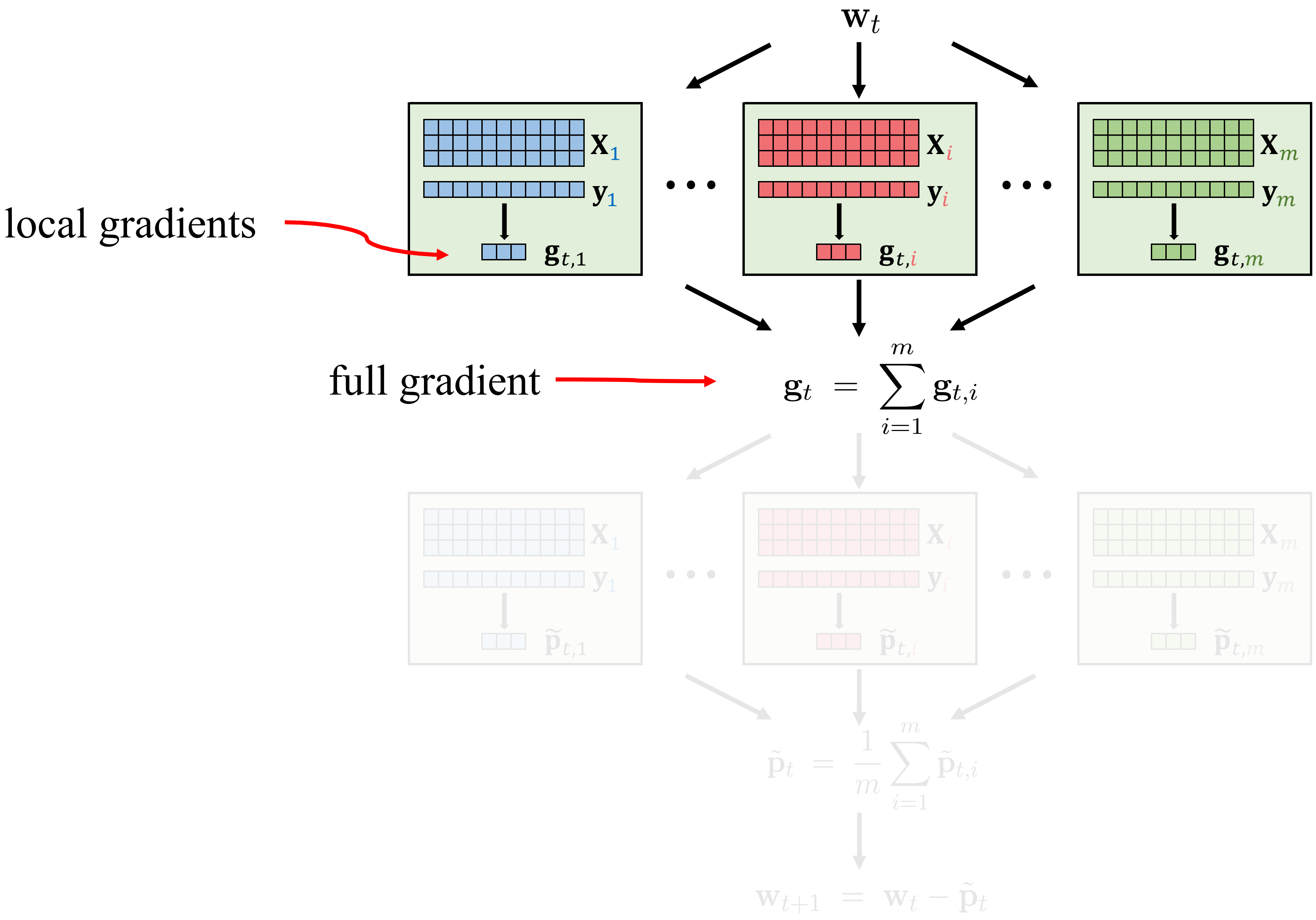
Recall: Newton's direction is $\mathbf{p} = \mathbf{H}^{-1}\mathbf{g}$.

$\tilde{\mathbf{p}} = \frac{1}{m}\sum_i \tilde{\mathbf{p}}_i = \left(\frac{1}{m}\sum_i \tilde{\mathbf{H}}_i^{-1}\right)\mathbf{g}$ approximates $\mathbf{p} = \mathbf{H}^{-1}\mathbf{g}$

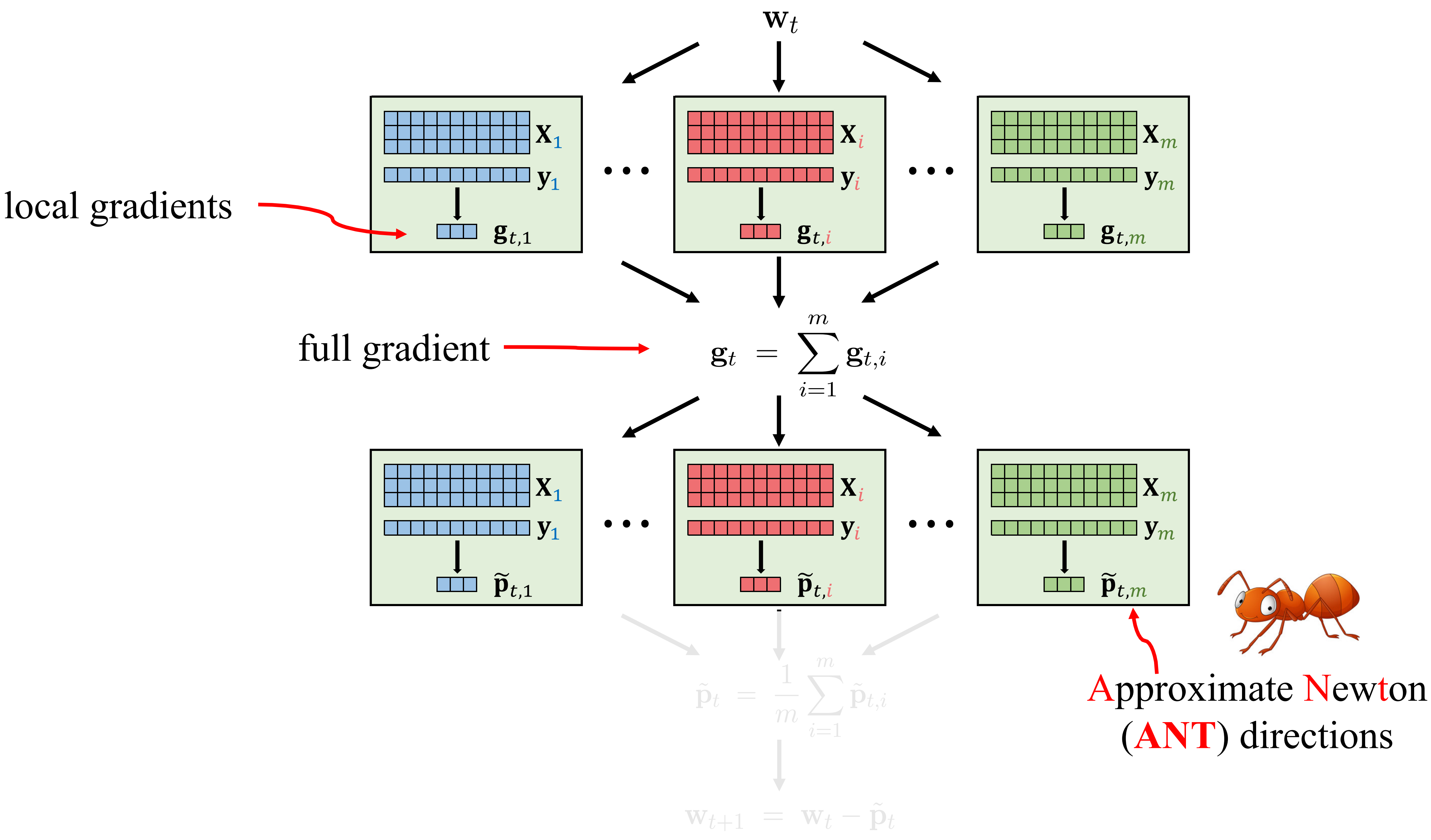
- GIANT uses the exact gradient $\mathbf{g}$.
- GIANT approximates the Hessian matrix $\mathbf{H}$ by $\left(\frac{1}{m}\sum_i \tilde{\mathbf{H}}_i^{-1}\right)^{-1}$.

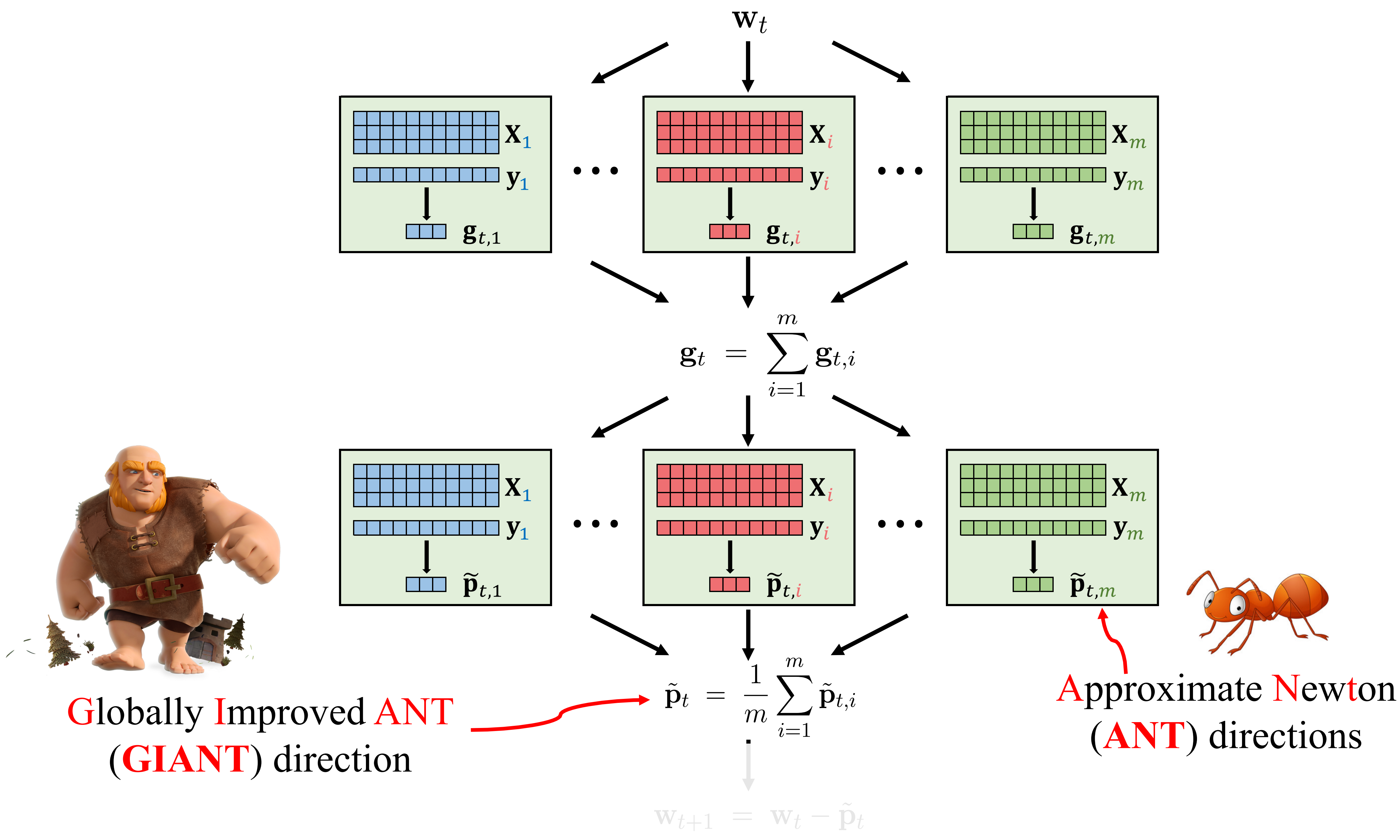


w_t
local gradients
X_1
y_1
g_{t,1}
X_i
y_i
g_{t,i}
X_m
y_m
g_{t,m}

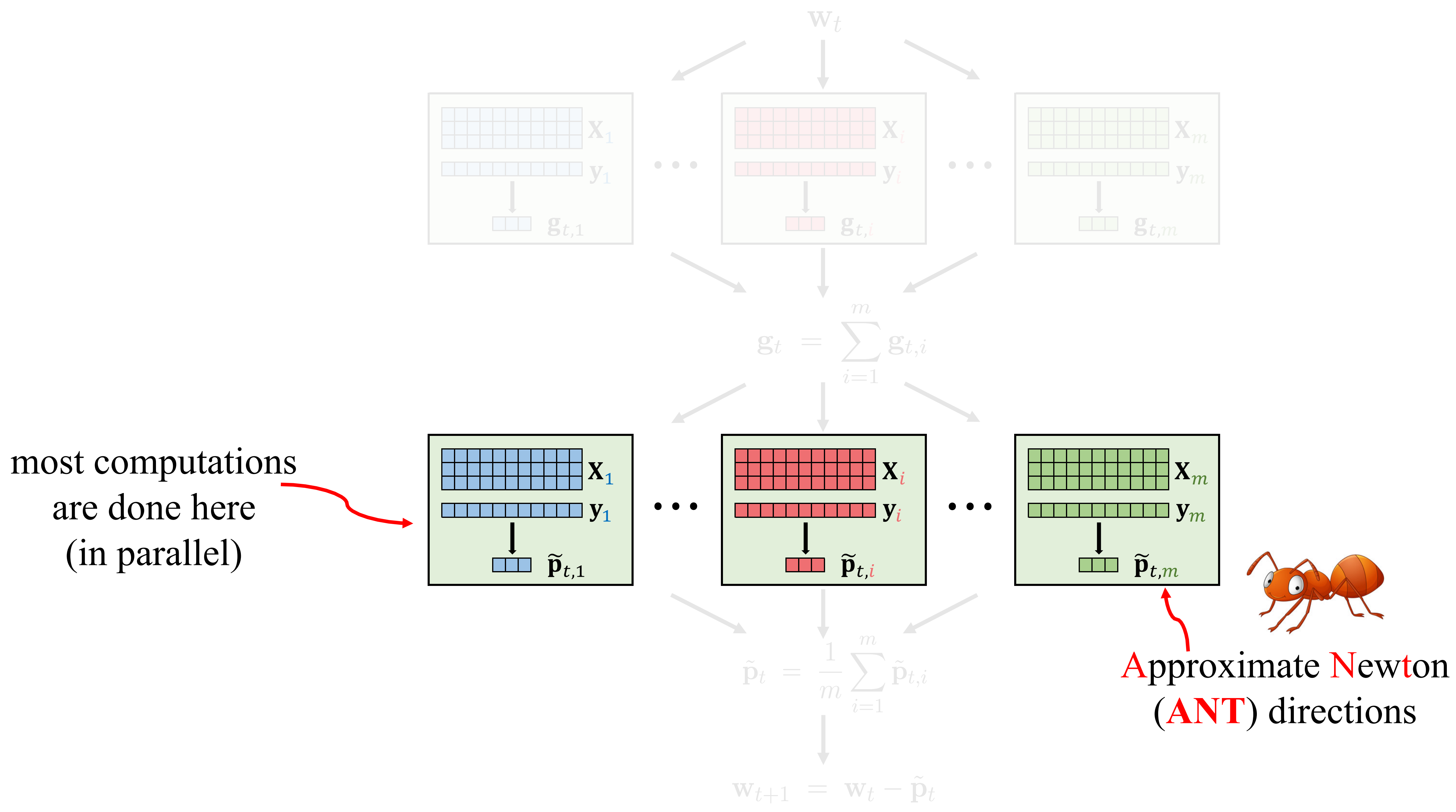


$\mathbf{w}_t$
$\mathbf{X}_1$
$\mathbf{y}_1$
$\mathbf{g}_{t,1}$
$\mathbf{X}_i$
$\mathbf{y}_i$
$\mathbf{g}_{t,i}$
$\mathbf{X}_m$
$\mathbf{y}_m$
$\mathbf{g}_{t,m}$
local gradients
full gradient
$\mathbf{g}_t = \sum_{i=1}^{m} \mathbf{g}_{t,i}$
$\tilde{\mathbf{p}}_{t,1}$
$\tilde{\mathbf{p}}_{t,i}$
$\tilde{\mathbf{p}}_{t,m}$
$\tilde{\mathbf{p}}_t = \frac{1}{m} \sum_{i=1}^{m} \tilde{\mathbf{p}}_{t,i}$
$\mathbf{w}_{t+1} = \mathbf{w}_t - \tilde{\mathbf{p}}_t$


$\mathbf{w}_t$
$\mathbf{X}_1$ $\mathbf{y}_1$ $\mathbf{g}_{t,1}$
$\mathbf{X}_i$ $\mathbf{y}_i$ $\mathbf{g}_{t,i}$
$\mathbf{X}_m$ $\mathbf{y}_m$ $\mathbf{g}_{t,m}$
local gradients
full gradient
$\mathbf{g}_t = \sum_{i=1}^{m} \mathbf{g}_{t,i}$
$\widetilde{\mathbf{p}}_{t,1}$ $\widetilde{\mathbf{p}}_{t,i}$ $\widetilde{\mathbf{p}}_{t,m}$
$\tilde{\mathbf{p}}_t = \frac{1}{m}\sum_{i=1}^{m} \tilde{\mathbf{p}}_{t,i}$
$\mathbf{w}_{t+1} = \mathbf{w}_t - \tilde{\mathbf{p}}_t$
Approximate Newton
(ANT) directions

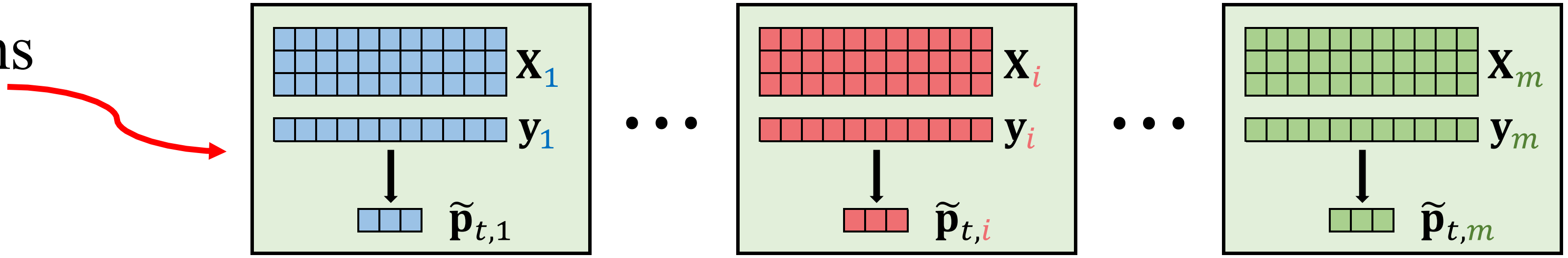


$\mathbf{w}_t$
$\mathbf{X}_1$
$\mathbf{y}_1$
$\mathbf{g}_{t,1}$
$\mathbf{X}_i$
$\mathbf{y}_i$
$\mathbf{g}_{t,i}$
$\mathbf{X}_m$
$\mathbf{y}_m$
$\mathbf{g}_{t,m}$
$\mathbf{g}_t = \sum_{i=1}^{m} \mathbf{g}_{t,i}$
$\mathbf{X}_1$
$\mathbf{y}_1$
$\tilde{\mathbf{p}}_{t,1}$
$\mathbf{X}_i$
$\mathbf{y}_i$
$\tilde{\mathbf{p}}_{t,i}$
$\mathbf{X}_m$
$\mathbf{y}_m$
$\tilde{\mathbf{p}}_{t,m}$
$\tilde{\mathbf{p}}_t = \frac{1}{m} \sum_{i=1}^{m} \tilde{\mathbf{p}}_{t,i}$
$\mathbf{w}_{t+1} = \mathbf{w}_t - \tilde{\mathbf{p}}_t$
Globally Improved ANT
(GIANT) direction
Approximate Newton
(ANT) directions

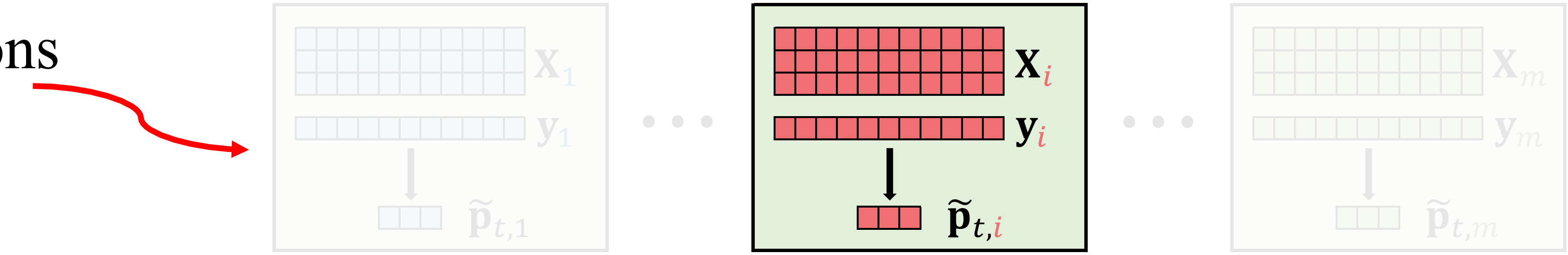


$\mathbf{w}_t$
$\mathbf{X}_1$
$\mathbf{y}_1$
$\mathbf{g}_{t,1}$
$\mathbf{X}_i$
$\mathbf{y}_i$
$\mathbf{g}_{t,i}$
$\mathbf{X}_m$
$\mathbf{y}_m$
$\mathbf{g}_{t,m}$
$\mathbf{g}_t = \sum_{i=1}^{m} \mathbf{g}_{t,i}$
most computations
are done here
(in parallel)
$\mathbf{X}_1$
$\mathbf{y}_1$
$\tilde{\mathbf{p}}_{t,1}$
$\mathbf{X}_i$
$\mathbf{y}_i$
$\tilde{\mathbf{p}}_{t,i}$
$\mathbf{X}_m$
$\mathbf{y}_m$
$\tilde{\mathbf{p}}_{t,m}$
Approximate Newton
(ANT) directions
$\tilde{\mathbf{p}}_t = \frac{1}{m} \sum_{i=1}^{m} \tilde{\mathbf{p}}_{t,i}$
$\mathbf{w}_{t+1} = \mathbf{w}_t - \tilde{\mathbf{p}}_t$

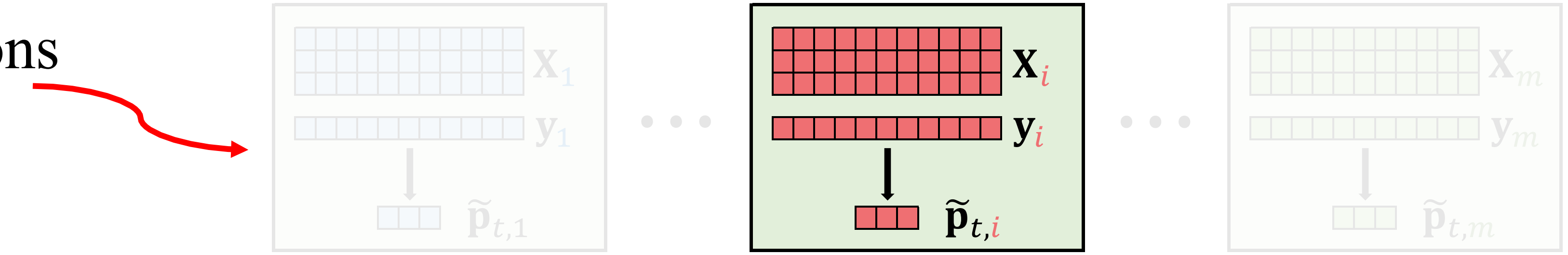
most computations
are done here
(in parallel)



most computations
are done here
(in parallel)



most computations
are done here
(in parallel)



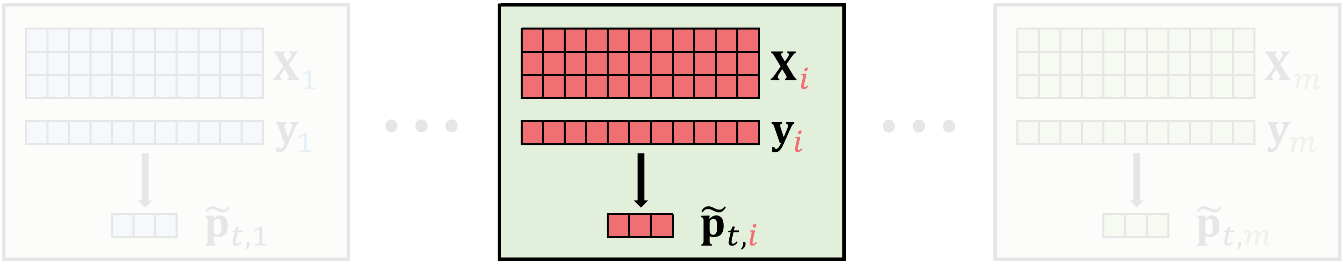
Naïve approach:

1. Form local Hessian $\widetilde{\mathbf{H}}_i \in \mathbb{R}^{d\times d}$
2. Invert $\widetilde{\mathbf{H}}_i$
3. The ANT direction $\tilde{\mathbf{p}}_{t,i} = \widetilde{\mathbf{H}}_i^{-1} \, \mathbf{g}_t$

It is inefficient!

1. Multiply two matrices to form $\widetilde{\mathbf{H}}_i$
2. Invert the dense matrix $\widetilde{\mathbf{H}}_i$

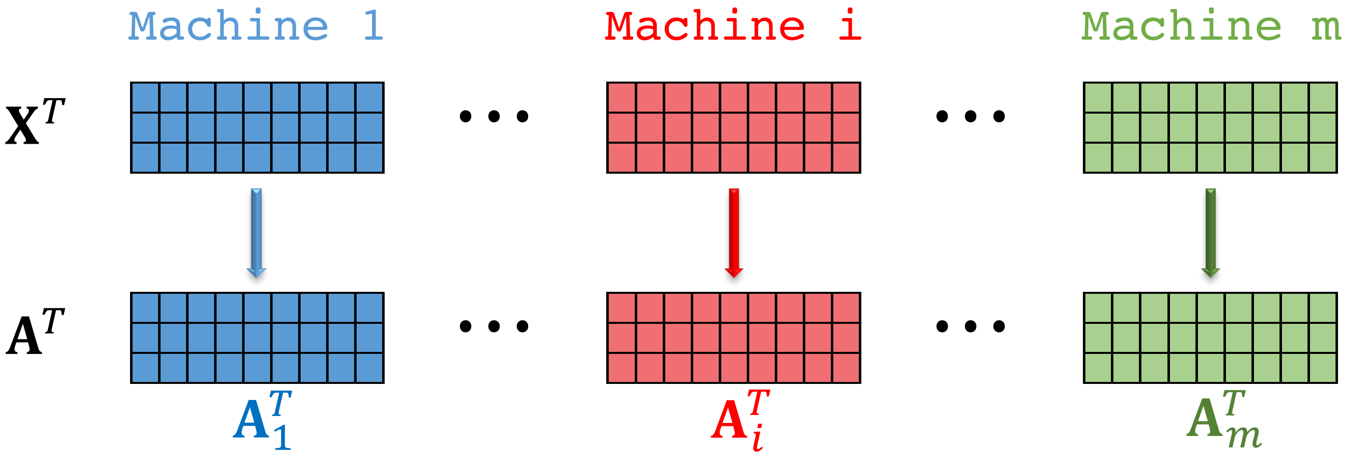
most computations are done here (in parallel)



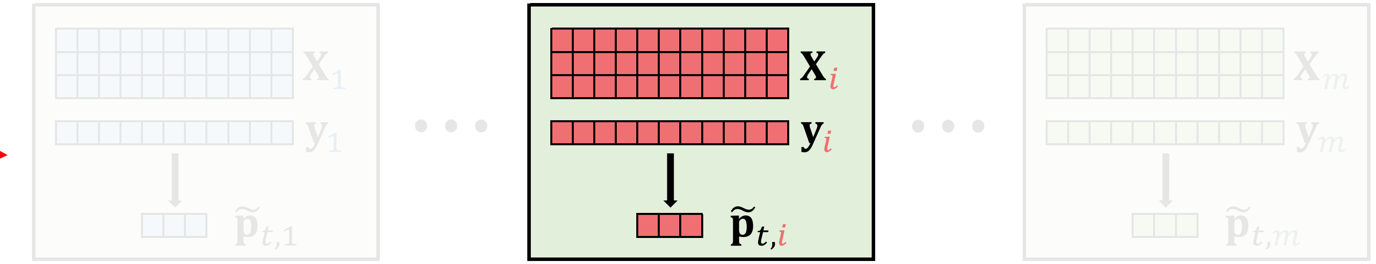
Fact: For the problem

$$\min_{\mathbf{w}\in\mathbb{R}^d} \quad \left\{ f(\mathbf{w}) \triangleq \frac{1}{n}\sum_{j=1}^{n} l\big(\mathbf{w}\,;\,\mathbf{x}_j, y_j\big) + \gamma\|\mathbf{w}\|_2^2 \right\} \quad ,$$

the local Hessian can be written as $\widetilde{\mathbf{H}}_i = \mathbf{A}_i^T\mathbf{A}_i + \gamma\mathbf{I}_d$.



most computations
are done here
(in parallel)



Fact: For the problem

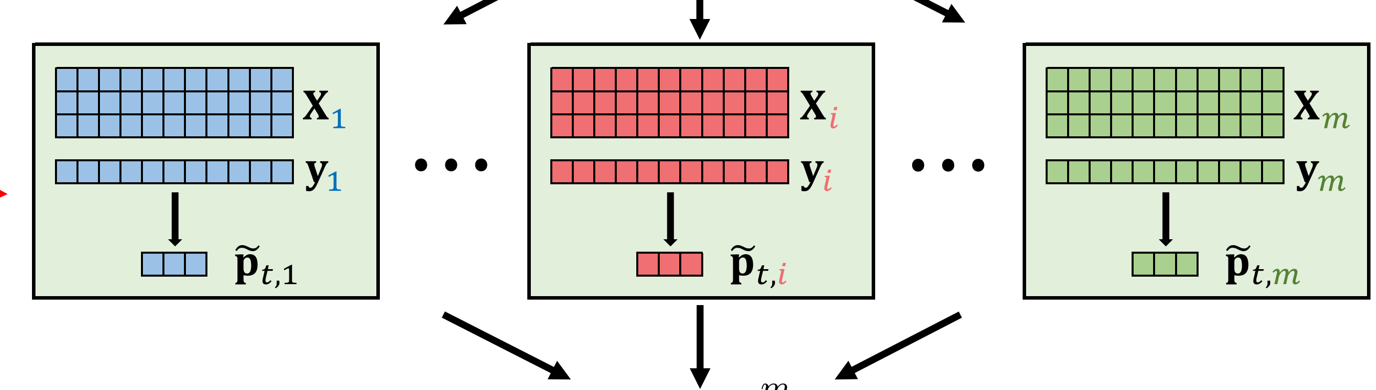
$$\min_{\mathbf{w}\in\mathbb{R}^d} \left\{ f(\mathbf{w}) \triangleq \frac{1}{n}\sum_{j=1}^{n} l(\mathbf{w}\,;\,\mathbf{x}_j, y_j) + \gamma\|\mathbf{w}\|_2^2 \right\} ,$$

the local Hessian can be written as $\widetilde{\mathbf{H}}_i = \mathbf{A}_i^T\mathbf{A}_i + \gamma\mathbf{I}_d$.

Local solver:

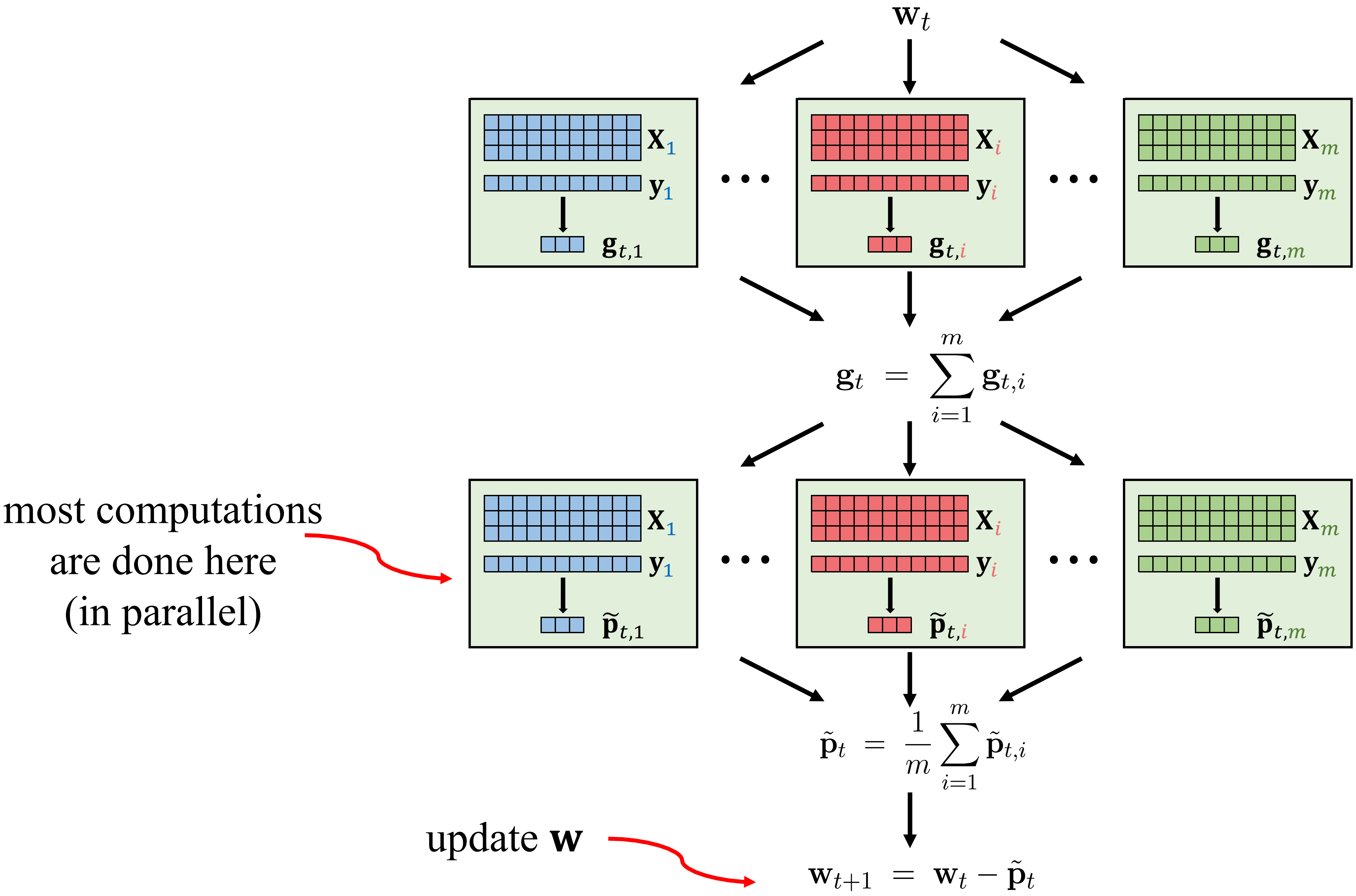
- Inexactly solve $(\mathbf{A}_i^T\mathbf{A}_i + \gamma\mathbf{I}_d)\mathbf{p} = \mathbf{g}_t$ by taking q CG steps.
- Cost: $2q$ matrix-vector products.

most computations are done here (in parallel)



$$\tilde{\mathbf{p}}_t = \frac{1}{m}\sum_{i=1}^{m} \tilde{\mathbf{p}}_{t,i}$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \tilde{\mathbf{p}}_t$$



$\mathbf{w}_t$
$\mathbf{X}_1$
$\mathbf{y}_1$
$\mathbf{g}_{t,1}$
$\mathbf{X}_i$
$\mathbf{y}_i$
$\mathbf{g}_{t,i}$
$\mathbf{X}_m$
$\mathbf{y}_m$
$\mathbf{g}_{t,m}$
$\mathbf{g}_t = \sum_{i=1}^{m} \mathbf{g}_{t,i}$
most computations are done here (in parallel)
$\tilde{\mathbf{p}}_{t,1}$
$\tilde{\mathbf{p}}_{t,i}$
$\tilde{\mathbf{p}}_{t,m}$
$\tilde{\mathbf{p}}_t = \frac{1}{m} \sum_{i=1}^{m} \tilde{\mathbf{p}}_{t,i}$
update **w**
$\mathbf{w}_{t+1} = \mathbf{w}_t - \tilde{\mathbf{p}}_t$

GIANT: Experiments

Settings

- Solve the ℓ_2-regularized logistic regression:

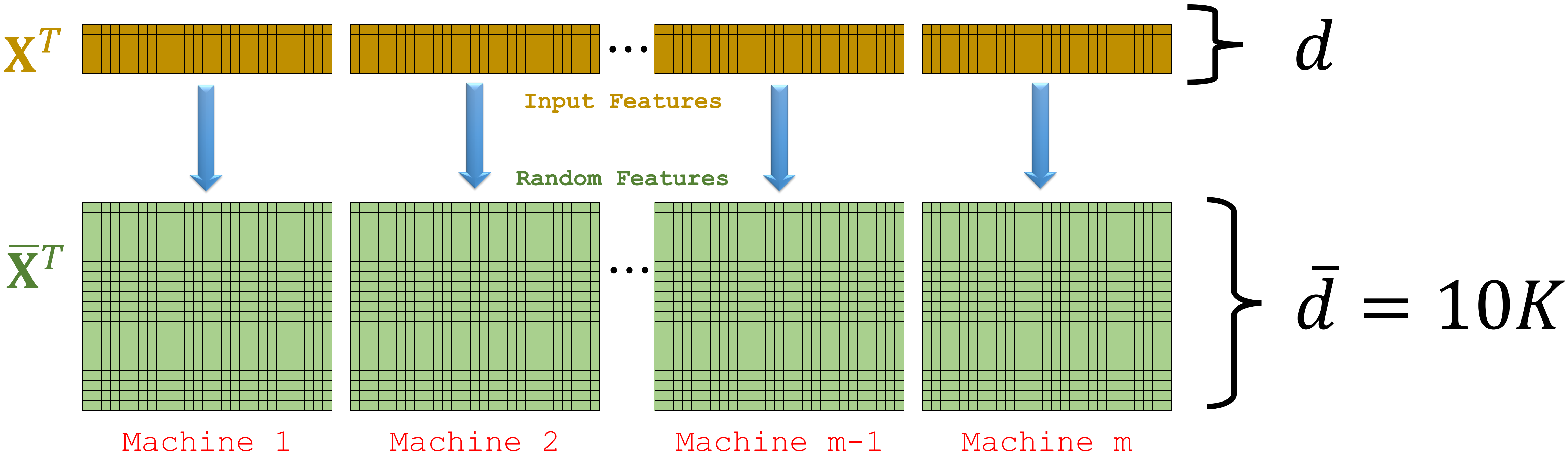
$$\min_{\mathbf{w}\in\mathbb{R}^d} \quad \left\{ f(\mathbf{w}) \triangleq \frac{1}{n}\sum_{j=1}^{n} \log\left(1 + e^{-y_j \mathbf{x}_j^T \mathbf{w}}\right) + \frac{\gamma}{2}\|\mathbf{w}\|_2^2 \right\}$$

Datasets

- Covtype: $n = 581\text{K}, d = 54$.
- Epsilon: $n = 500\text{K}, d = 2\text{K}$.
- 80% for training, 20% for test.

Datasets

- Covtype: $n = 581\text{K}, d = 54$.
- Epsilon: $n = 500\text{K}, d = 2\text{K}$.
- 80% for training, 20% for test.



Compared Methods

- Accelerated gradient descent (AGD)
  - choose *step size* from {0.1, 1, 10, 100}
  - choose *momentum* from {0.5, 0.9, 0.95, 0.99, 0.999}

Compared Methods

- Accelerated gradient descent (AGD)
- Limited memory BFGS (a quasi-Newton method)
  - choose *number of history* from {30, 100, 300}
  - line search is used

Compared Methods

- Accelerated gradient descent (AGD)
- Limited memory BFGS
- DANE (another Newton-type method) [Shamir et al. 2014]
  - local solver: SVRG (a stochastic optimization method)
  - choose *step size of SVRG* from {0.1, 1, 10, 100}
  - choose *max. iteration of SVRG* from {30, 100, 300}

Reference:

Shamir, Srebro, & Zhang. Communication Efficient Distributed Optimization using an Approximate Newton-type Method. In *ICML*, 2014.

Compared Methods

- Accelerated gradient descent (AGD)
- Limited memory BFGS
- DANE (another Newton-type method)
- GIANT
  - local solver: conjugate gradient (CG)
  - choose *max iteration of CG* from {30, 100, 300}

Compared Methods

Accelerated gradient descent (AGD)	2 Tuning Parameters
Limited memory BFGS	1 Tuning Parameter
DANE (another Newton-type method)	2 Tuning Parameters
GIANT	1 Tuning Parameter

Experiment Environment

- Spark 2.1.1 + Scala 2.11.8

Experiment Environment

- Spark 2.1.1 + Scala 2.11.8
- Cori Supercomputer (Cray XC40)

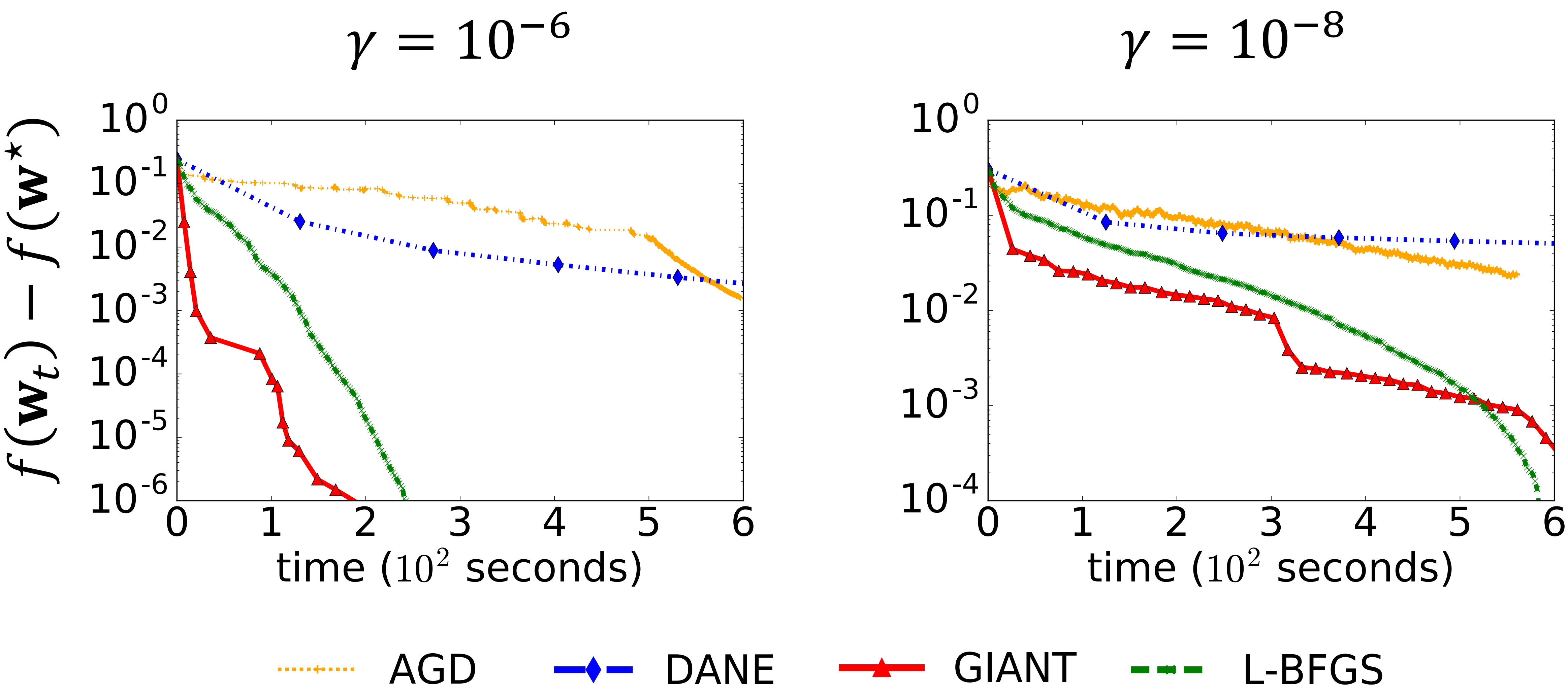
NERSC National Energy Research Scientific Computing Center

BERKELEY LAB Bringing Science Solutions to the World

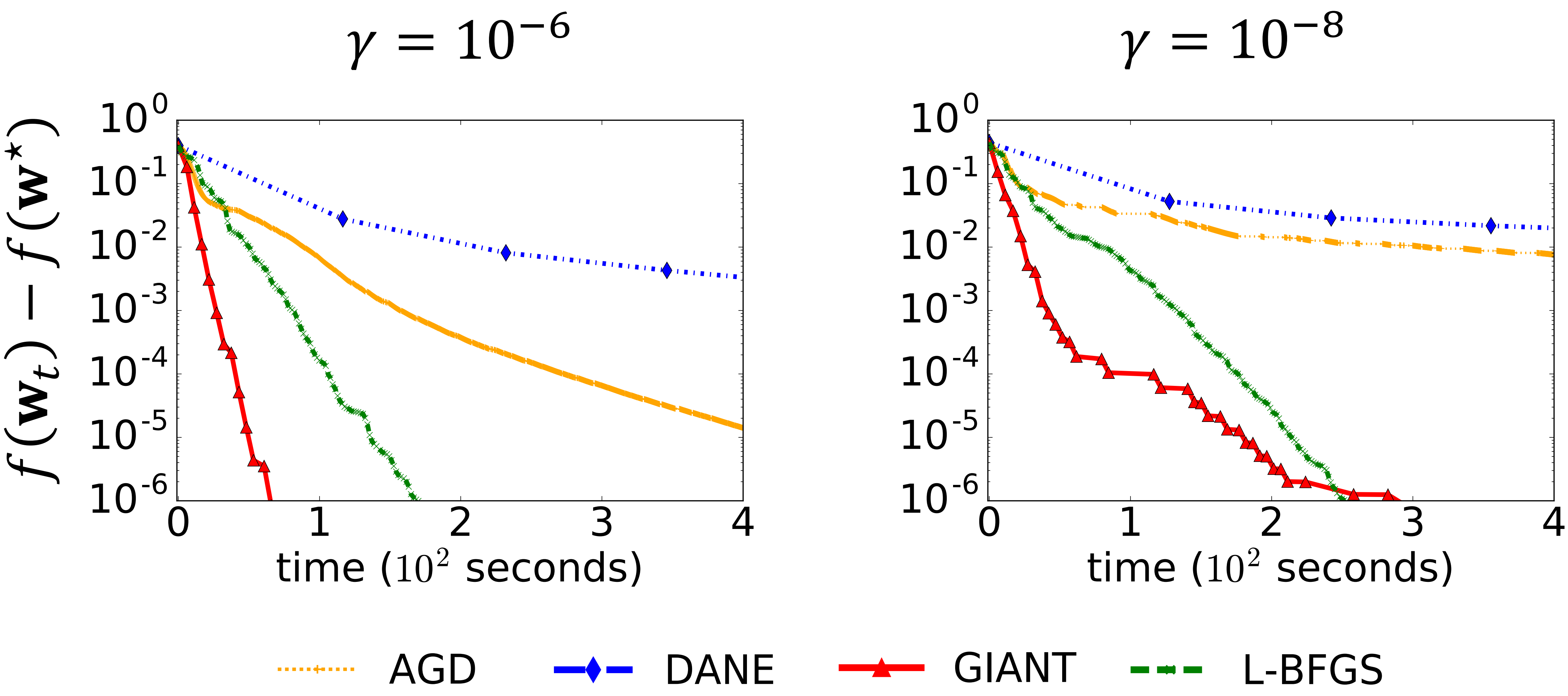
Experiment Environment

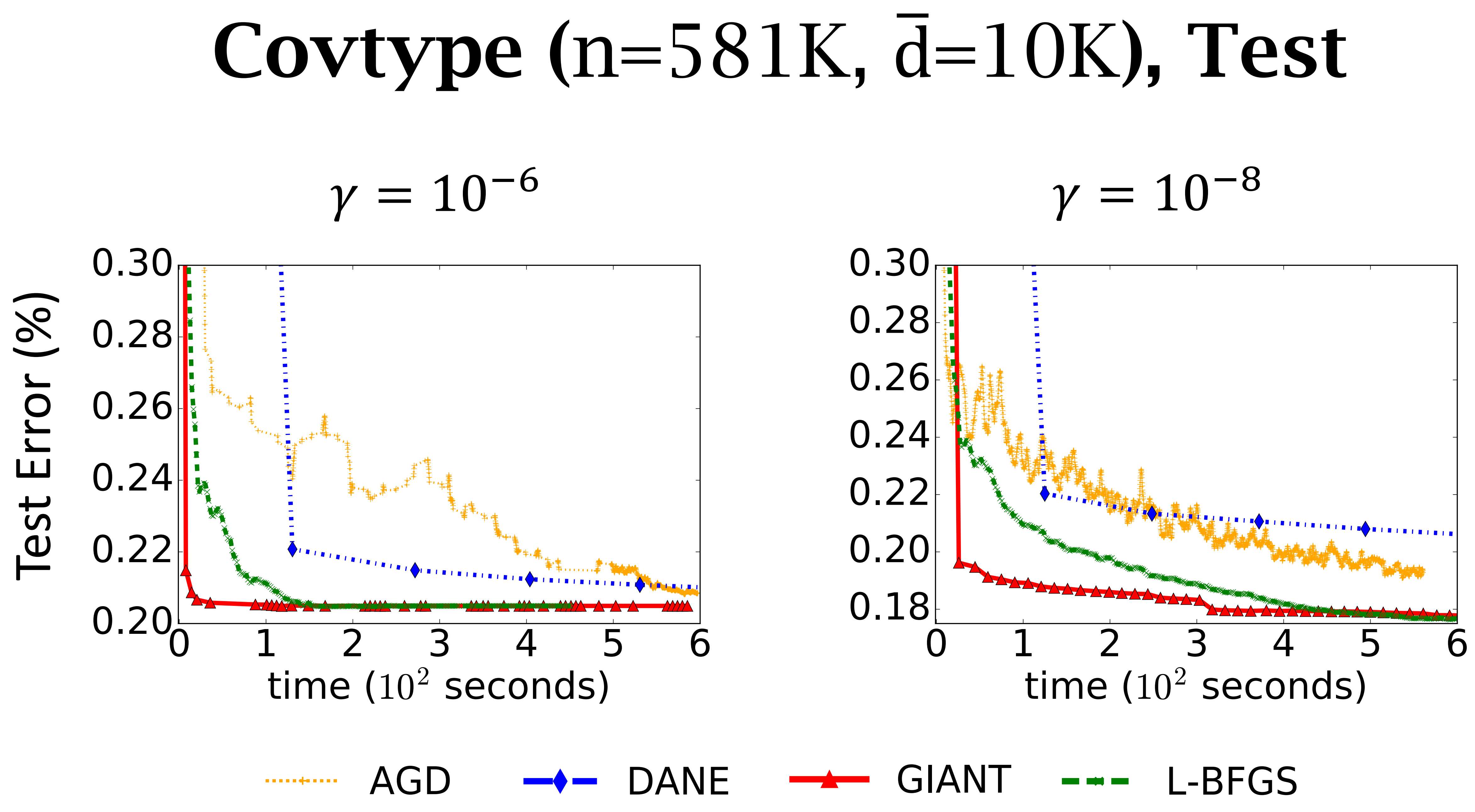
- Spark 2.1.1 + Scala 2.11.8
- Cori Supercomputer (Cray XC40)
  - 128 GB Memory / node
  - 32 Cores / node
- Use 15 nodes (480 CPU cores)

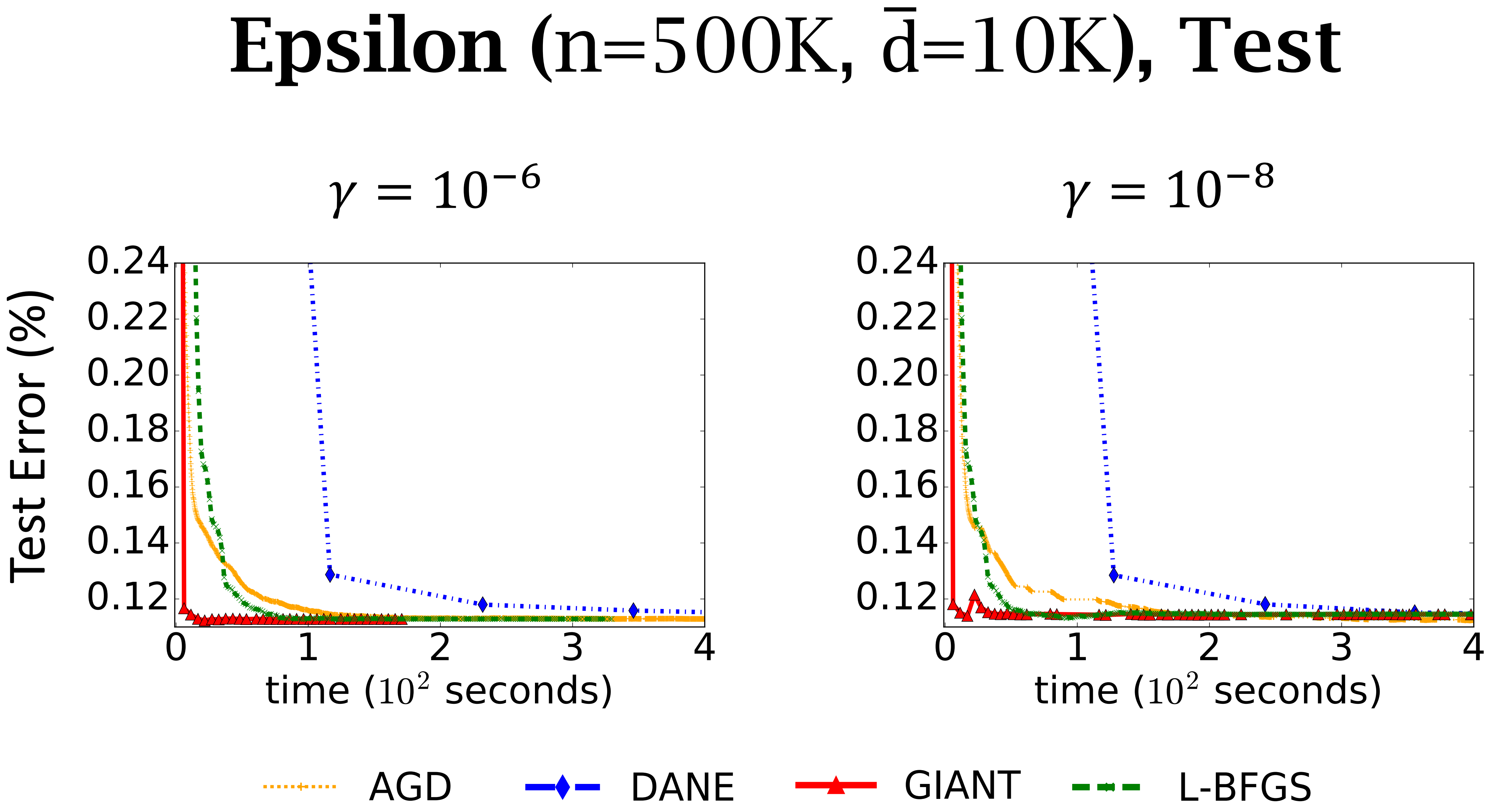
Covtype (n=581K, $\bar{d}$=10K), Training



Epsilon (n=500K, $\bar{d}$=10K), Training




Covtype (n=581K, d̄=10K), Test
$\gamma = 10^{-6}$
$\gamma = 10^{-8}$
Test Error (%)
0.30
0.28
0.26
0.24
0.22
0.20
0.18
0
1
2
3
4
5
6
time (10^2 seconds)
AGD
DANE
GIANT
L-BFGS

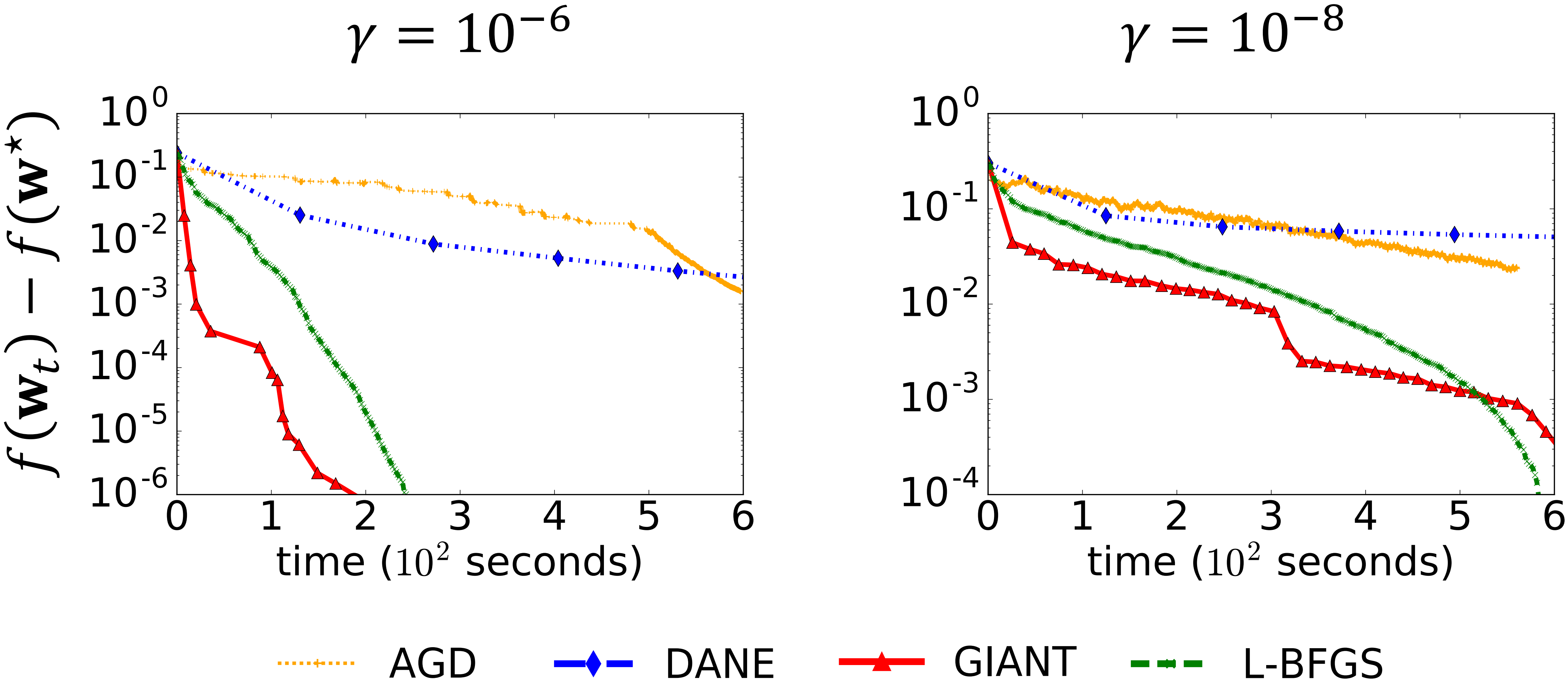


Epsilon (n=500K, d̄=10K), Test
$\gamma = 10^{-6}$
$\gamma = 10^{-8}$
Test Error (%)
0.24
0.22
0.20
0.18
0.16
0.14
0.12
0
1
2
3
4
time (10^2 seconds)
AGD
DANE
GIANT
L-BFGS

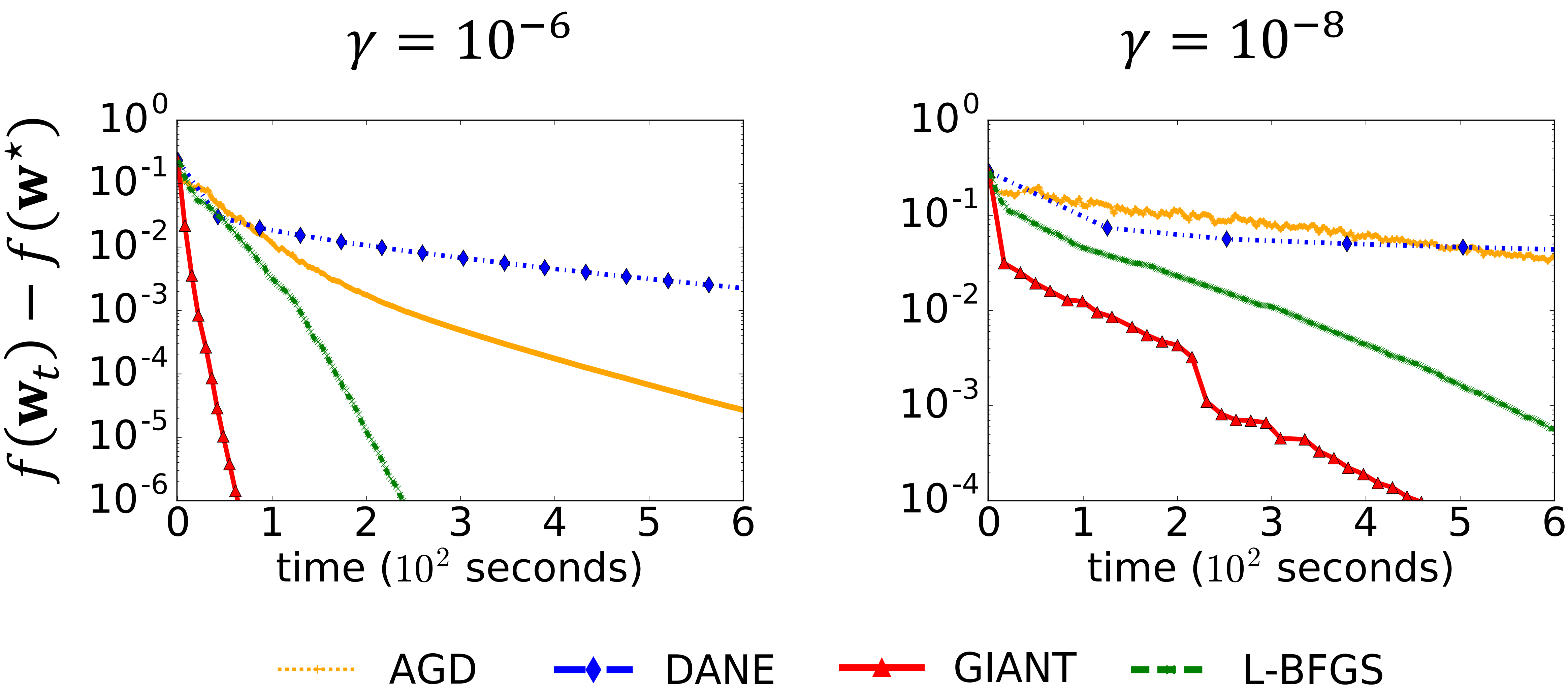
Scaling Experiments

- Make the Covtype data k times larger.
  1. Get k replicates of $\mathbf{X}$ and $\mathbf{y}$;
  2. Inject i.i.d. Gaussian noises to the $kn \times d$ feature matrix;
  3. Do random feature mapping to get 10K features.
- Use k times more nodes.
- Set $k = 5$ and $k = 25$.

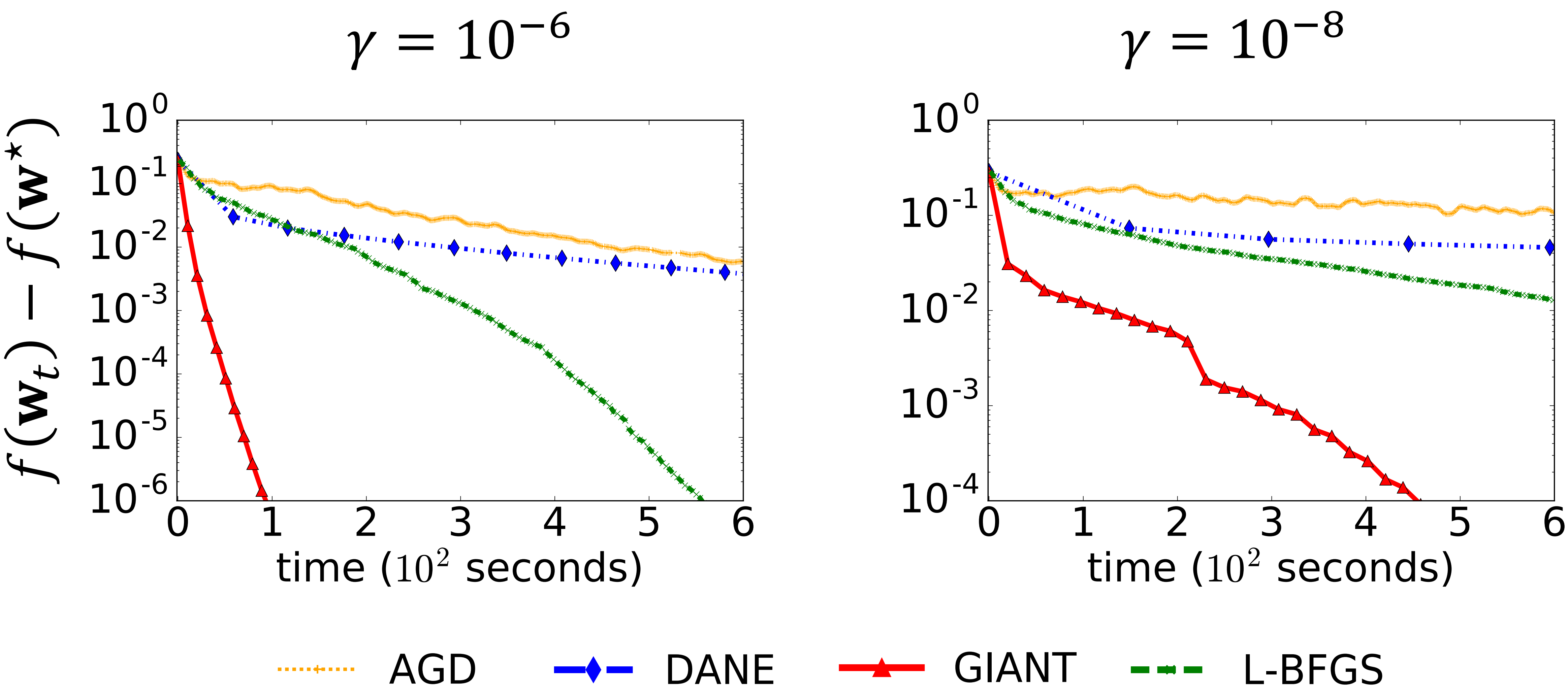
Original Data, 15 Nodes (480 Cores)



5x Larger Data, 75 Nodes (2.4K Cores)



25x Larger Data, 375 Nodes (12K Cores)

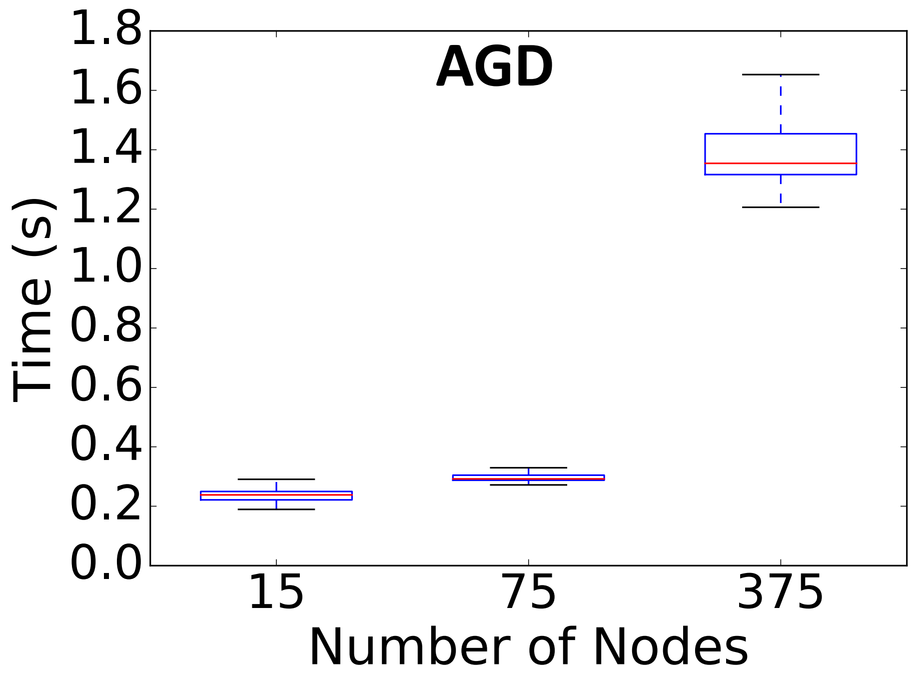


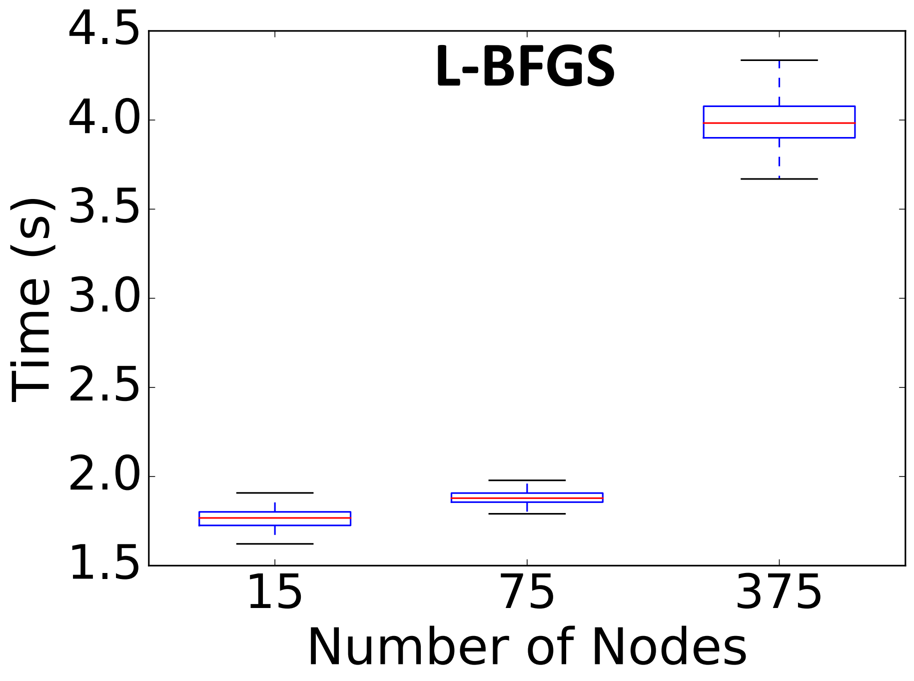
Why is GIANT More Scalable?

- As #Samples and #Nodes both increases by k times,
  - the **computational** costs remain **the same**;
  - the **communication** costs **increase**.

Why is GIANT More Scalable?

- As #Samples and #Nodes both increases by k times,
  - the computational costs remain the;
  - the communication costs increase.
- Per-iteration time of AGD and L-BFGS increases.

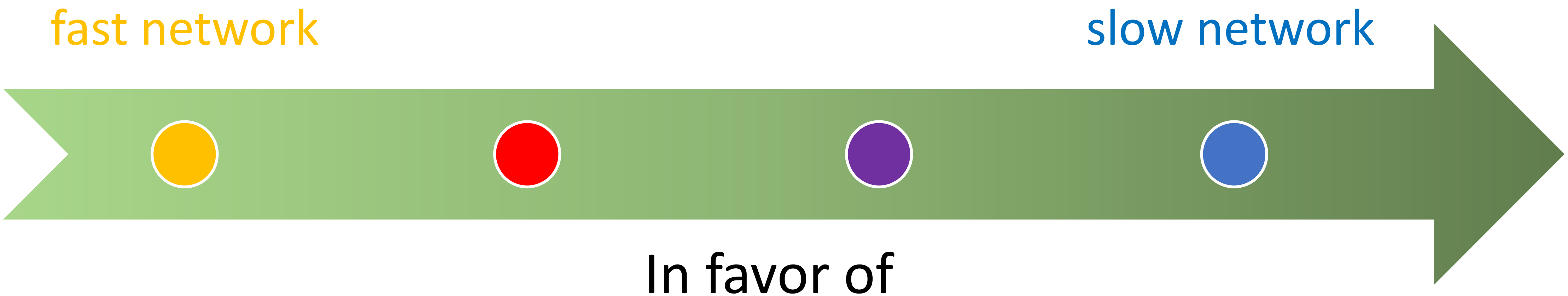




Why is GIANT More Scalable?

- As #Samples and #Nodes both increases by k times,
  - the computational costs remain the;
  - the communication costs increase.
- Per-iteration time of AGD and L-BFGS increases.
- Per-iteration time of GIANT marginally increases.
  - Because GIANT is computation-intensive.

FLOPs versus Communication



In favor of

- SGD
- SCD

- Mini-Batch SGD
- Block SCD

- AGD
- L-BFGS

- GIANT
- DANE

Single Machine | Cluster | Federated Learning

Spark on Supercomputer

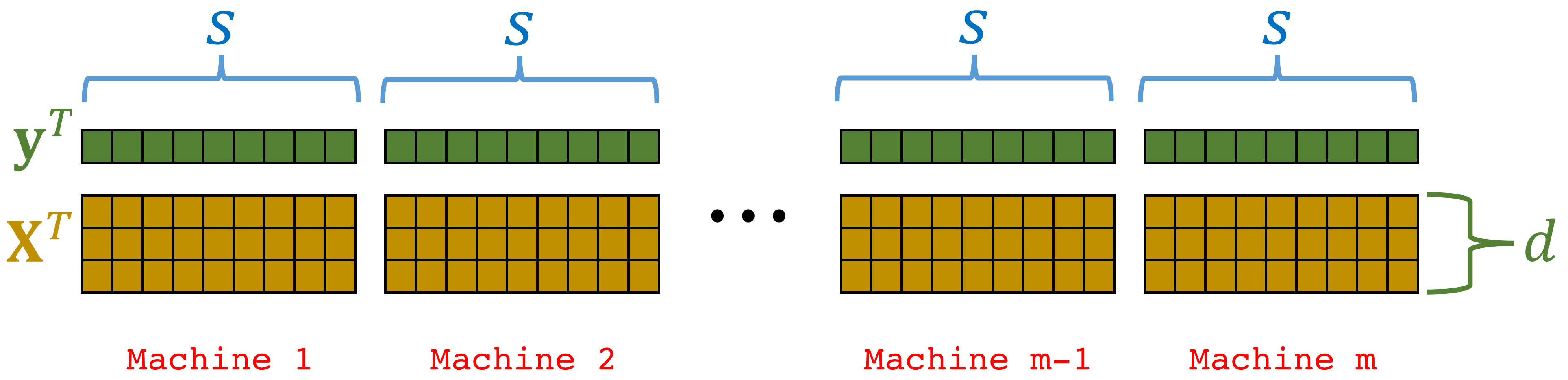
GIANT: Convergence Analysis

Quadratic Loss: Global Convergence

- Objective function: $f(\mathbf{w}) \quad = \quad \frac{1}{2n}\left\|\mathbf{X}\mathbf{w} - \mathbf{y}\right\|_2^2 + \frac{\gamma}{2}\|\mathbf{w}\|_2^2$

Quadratic Loss: Global Convergence

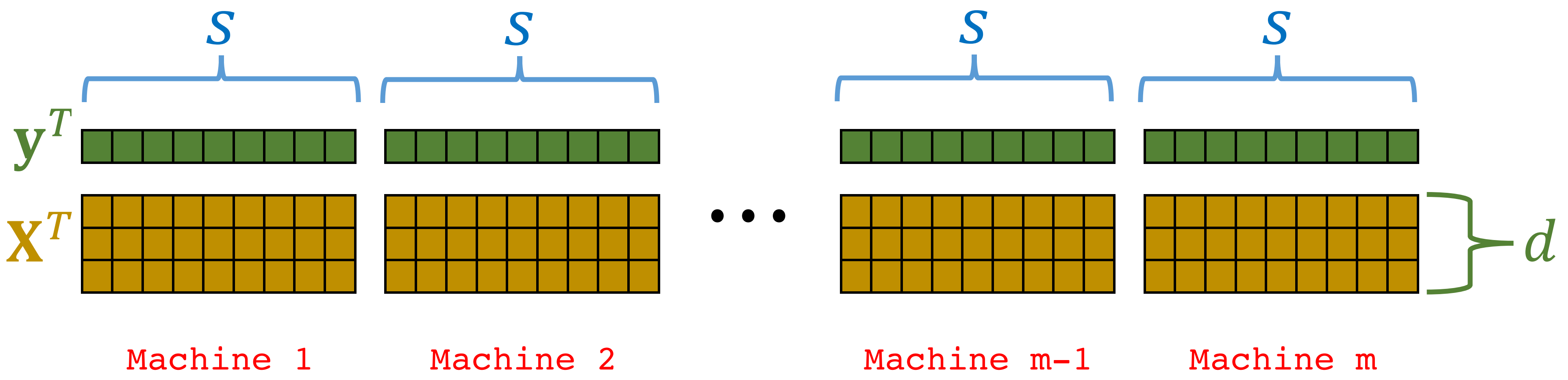
- Objective function: $f(\mathbf{w}) = \frac{1}{2n}\|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 + \frac{\gamma}{2}\|\mathbf{w}\|_2^2$
- Assume $\mathbf{X}$ is "incoherent" (information uniformly spread)
- Assume local sample size is $s = \Theta(\frac{d}{\epsilon^2} \log \frac{md}{\delta})$ for any $\epsilon, \delta \in (0,1)$



Quadratic Loss: Global Convergence

- Objective function: $f(\mathbf{w}) \;=\; \frac{1}{2n}\big\|\mathbf{X}\mathbf{w} - \mathbf{y}\big\|_2^2 + \frac{\gamma}{2}\|\mathbf{w}\|_2^2$
- Assume $\mathbf{X}$ is "incoherent" (information uniformly spread)
- Assume local sample size is $s = \Theta\big(\frac{d}{\epsilon^2}\log\frac{md}{\delta}\big)$ for any $\epsilon, \delta \in (0,1)$
- With probability $1-\delta$ (assume random partition of the data),

$$\frac{\|\mathbf{\Delta}_t\|_2}{\|\mathbf{\Delta}_0\|_2} \;\leq\; \Big(\frac{\epsilon}{\sqrt{m}} + \epsilon^2\Big)^t \sqrt{\kappa}, \qquad \text{where } \; \mathbf{\Delta}_t \triangleq \mathbf{w}_t - \mathbf{w}^\star.$$

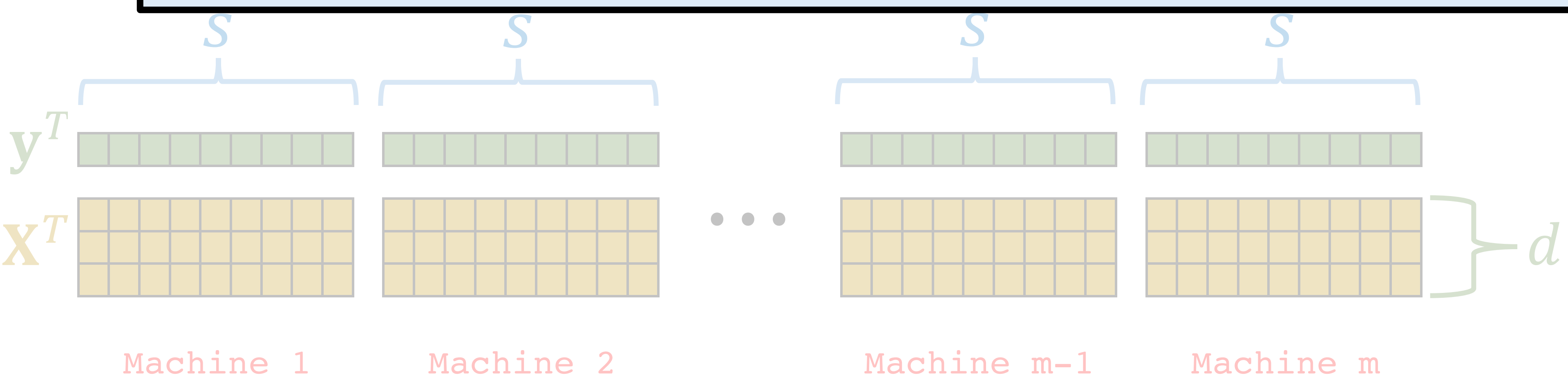


Quadratic Loss: Global Convergence

- Objective function: $f(\mathbf{w}) = \frac{1}{2n}\|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 + \frac{\gamma}{2}\|\mathbf{w}\|_2^2$
- Ass
- Assume local sample size is $s = \Theta(\frac{d}{\epsilon^2}\log\frac{md}{\delta})$ for any $\epsilon, \delta \in (0,1)$
- With probability $1-\delta$ (assume random partition of the data).

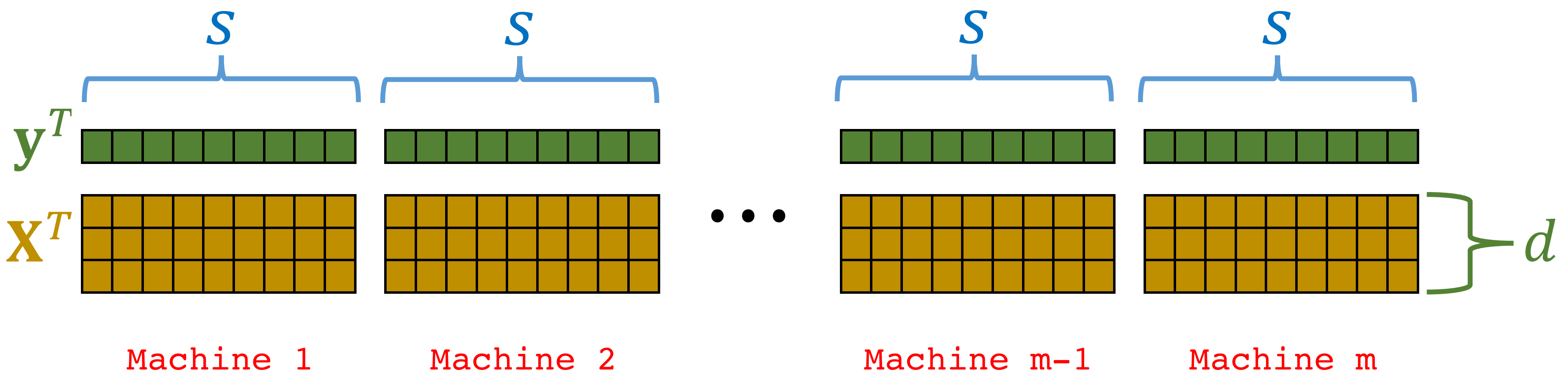
GIANT has $\log\kappa$ dependence.

AGD has $\sqrt{\kappa}$ dependence.



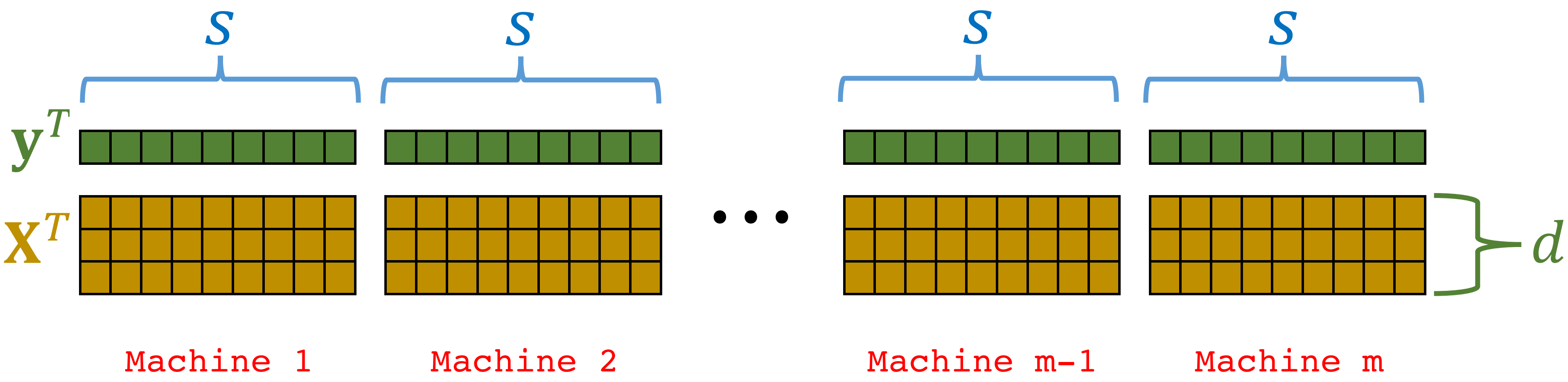
General Smooth Loss: Local Convergence

- Denote $\mathbf{H}_t = \nabla^2 f(\mathbf{w}_t)$ and $\mathbf{H}^\star = \nabla^2 f(\mathbf{w}^\star)$
- Similar "incoherent" assumption (information uniformly spread)
- Assume local sample size is $s = \Theta(\frac{d}{\epsilon^2} \log \frac{md}{\delta})$ for any $\epsilon, \delta \in (0, 1)$



General Smooth Loss: Local Convergence

- Denote $\mathbf{H}_t = \nabla^2 f(\mathbf{w}_t)$ and $\mathbf{H}^\star = \nabla^2 f(\mathbf{w}^\star)$
- Similar "incoherent" assumption (information uniformly spread)
- Assume local sample size is $s = \Theta(\frac{d}{\epsilon^2} \log \frac{md}{\delta})$ for any $\epsilon, \delta \in (0,1)$
- Assume the Hessian is L-Lipchitz: $\left\|\nabla^2 f(\mathbf{w}) - \nabla^2 f(\mathbf{w}')\right\|_2 \leq L\|\mathbf{w} - \mathbf{w}'\|_2$



General Smooth Loss: Local Convergence

- Denote $\mathbf{H}_t = \nabla^2 f(\mathbf{w}_t)$ and $\mathbf{H}^\star = \nabla^2 f(\mathbf{w}^\star)$
- Similar "incoherent" assumption (information uniformly spread)
- Assume local sample size is $s = \Theta(\frac{d}{\epsilon^2} \log \frac{md}{\delta})$ for any $\epsilon, \delta \in (0,1)$
- Assume the Hessian is L-Lipchitz: $\left\| \nabla^2 f(\mathbf{w}) - \nabla^2 f(\mathbf{w}') \right\|_2 \leq L \|\mathbf{w} - \mathbf{w}'\|_2$
- With probability $1-\delta$ (assume random partition of the data),

$$\left\|\mathbf{\Delta}_{t+1}\right\|_2 \leq \max\left\{ \underbrace{\left(\frac{\epsilon}{\sqrt{m}} + \epsilon^2\right) \sqrt{\frac{\sigma_{\max}(\mathbf{H}_t)}{\sigma_{\min}(\mathbf{H}_t)}} \left\|\mathbf{\Delta}_t\right\|_2}_{\text{Linear}}, \underbrace{\frac{2L}{\sigma_{\min}(\mathbf{H}_t)} \left\|\mathbf{\Delta}_t\right\|_2^2}_{\text{Quadratic}} \right\}$$

Inexactly Solving Local Linear System

- Exactly solving $\widetilde{\mathbf{H}}_{t,i}\mathbf{p} = \mathbf{g}_t$ may not be easy.
- Solve $\widetilde{\mathbf{H}}_{t,i}\mathbf{p} = \mathbf{g}_t$ by taking $q = \frac{\sqrt{\kappa}-1}{2}\log\frac{8}{\epsilon_0^2}$ CG steps.
- Recall the bounds of **exact** solver:

Quadratic Loss: $$\frac{\|\mathbf{\Delta}_t\|_2}{\|\mathbf{\Delta}_0\|_2} \leq \left(\frac{\epsilon}{\sqrt{m}} + \epsilon^2\right)^t \sqrt{\kappa}, \quad \text{where } \mathbf{\Delta}_t \triangleq \mathbf{w}_t - \mathbf{w}^\star.$$

General Loss: $$\left\|\mathbf{\Delta}_{t+1}\right\|_2 \leq \max\left\{\left(\frac{\epsilon}{\sqrt{m}} + \epsilon^2\right)\sqrt{\frac{\sigma_{\max}(\mathbf{H}_t)}{\sigma_{\min}(\mathbf{H}_t)}}\left\|\mathbf{\Delta}_t\right\|_2, \ \frac{2L}{\sigma_{\min}(\mathbf{H}_t)}\left\|\mathbf{\Delta}_t\right\|_2^2\right\}$$

Inexactly Solving Local Linear System

- Exactly solving $\widetilde{\mathbf{H}}_{t,i}\mathbf{p} = \mathbf{g}_t$ may not be easy.
- Solve $\widetilde{\mathbf{H}}_{t,i}\mathbf{p} = \mathbf{g}_t$ by taking $q = \frac{\sqrt{\kappa}-1}{2} \log \frac{8}{\epsilon_0^2}$ CG steps.
- Recall the bounds of **exact** solver:

Quadratic Loss: $$\frac{\|\mathbf{\Delta}_t\|_2}{\|\mathbf{\Delta}_0\|_2} \leq \left(\frac{\epsilon}{\sqrt{m}} + \epsilon^2\right)^t \sqrt{\kappa}, \quad \text{where} \quad \mathbf{\Delta}_t \triangleq \mathbf{w}_t - \mathbf{w}^\star.$$

General Loss: $$\big\|\mathbf{\Delta}_{t+1}\big\|_2 \leq \max\left\{\left(\frac{\epsilon}{\sqrt{m}} + \epsilon^2\right)\sqrt{\frac{\sigma_{\max}(\mathbf{H}_t)}{\sigma_{\min}(\mathbf{H}_t)}}\big\|\mathbf{\Delta}_t\big\|_2, \ \frac{2L}{\sigma_{\min}(\mathbf{H}_t)}\big\|\mathbf{\Delta}_t\big\|_2^2\right\}$$

- Bounds of the inexact solver:

$$\frac{\epsilon}{\sqrt{m}} + \epsilon^2 \quad \Longrightarrow \quad \frac{\epsilon}{\sqrt{m}} + \epsilon^2 + \epsilon_0$$

Inexactly Solving Local Linear System

- Exactly solving $\widetilde{\mathbf{H}}_{t,i}\mathbf{p} = \mathbf{g}_t$ may not be easy.
- Solve $\widetilde{\mathbf{H}}_{t,i}\mathbf{p} = \mathbf{g}_t$ by taking $q = \frac{\sqrt{\kappa}-1}{2} \log \frac{8}{\epsilon_0^2}$ CG steps.
- Recall the bounds of **exact** solver:

Quadratic Loss: $$\frac{\|\mathbf{\Delta}_t\|_2}{\|\mathbf{\Delta}_0\|_2} \leq \left(\frac{\epsilon}{\sqrt{m}} + \epsilon^2\right)^t \sqrt{\kappa}, \quad \text{where } \mathbf{\Delta}_t \triangleq \mathbf{w}_t - \mathbf{w}^\star.$$

General Loss: $$\big\|\mathbf{\Delta}_{t+1}\big\|_2 \leq \max\left\{\left(\frac{\epsilon}{\sqrt{m}} + \epsilon^2\right)\sqrt{\frac{\sigma_{\max}(\mathbf{H}_t)}{\sigma_{\min}(\mathbf{H}_t)}}\big\|\mathbf{\Delta}_t\big\|_2, \ \frac{2L}{\sigma_{\min}(\mathbf{H}_t)}\big\|\mathbf{\Delta}_t\big\|_2^2\right\}$$

- Bounds of the inexact solver:

$$\frac{\epsilon}{\sqrt{m}} + \epsilon^2 \quad \Longrightarrow \quad \frac{\epsilon}{\sqrt{m}} + \epsilon^2 + \epsilon_0$$

Outline of Proof

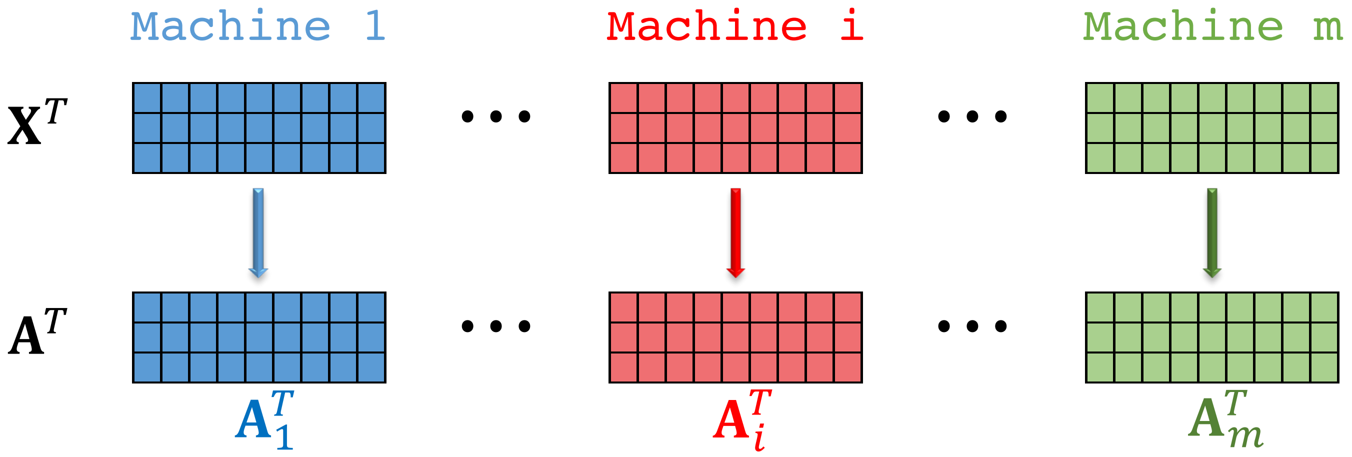
Proof Techniques

Claim 1: Local Hessian $\widetilde{\mathbf{H}}_1, \cdots, \widetilde{\mathbf{H}}_m$ well approximate the true Hessian $\mathbf{H}$.

Proof Techniques

Claim 1: Local Hessian $\widetilde{\mathbf{H}}_1, \cdots, \widetilde{\mathbf{H}}_m$ well approximate the true Hessian $\mathbf{H}$.

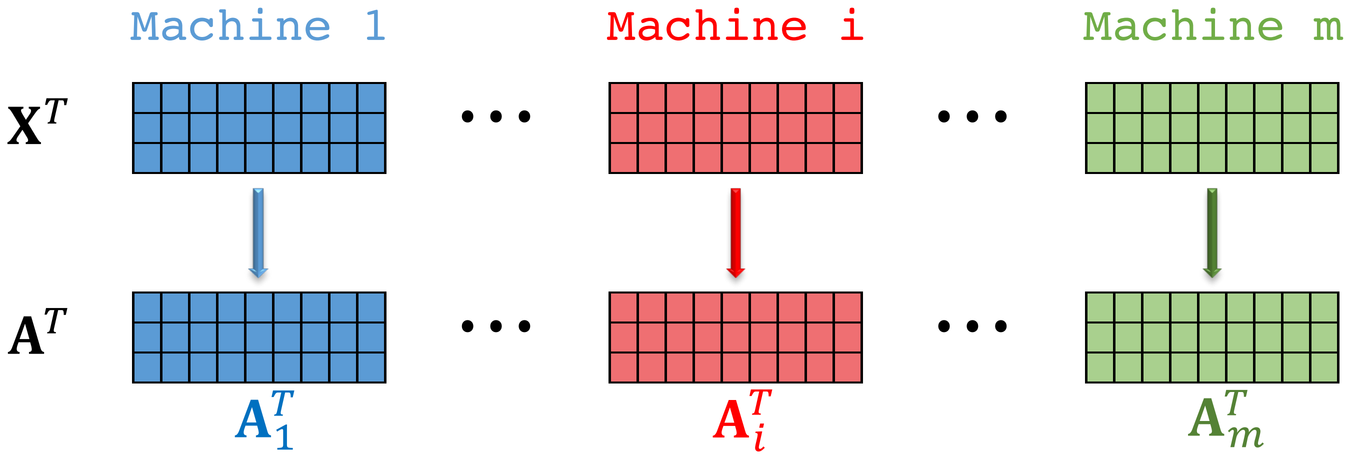
- $\mathbf{H}$ can always be written as $\mathbf{H} = \mathbf{A}^T\mathbf{A} + \gamma \mathbf{I}_d$, for some $\mathbf{A} \in \mathbb{R}^{n \times d}$.



Proof Techniques

Claim 1: Local Hessian $\widetilde{\mathbf{H}}_1, \cdots, \widetilde{\mathbf{H}}_m$ well approximate the true Hessian $\mathbf{H}$.

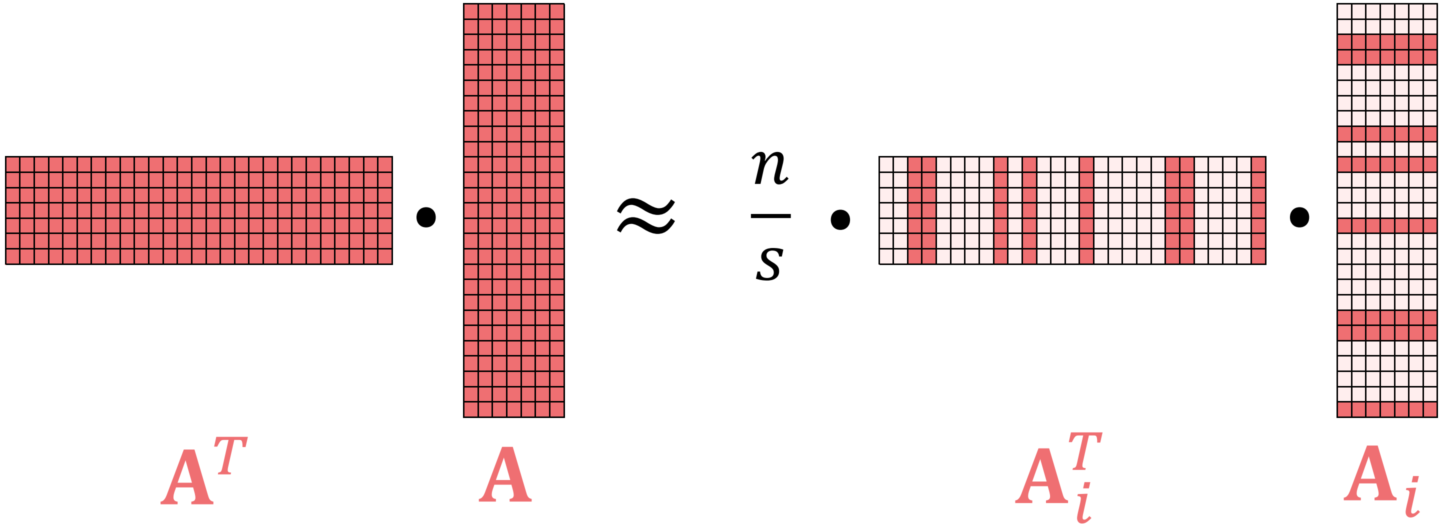
- $\mathbf{H}$ can always be written as $\mathbf{H} = \mathbf{A}^T\mathbf{A} + \gamma\mathbf{I}_d$, for some $\mathbf{A} \in \mathbb{R}^{n \times d}$.
- Assume $\mathbf{A}$ is "incoherent".
- Uniformly partition $\mathbf{A}$ to form $\mathbf{A}_i \in \mathbb{R}^{s \times d}$.



Proof Techniques

Claim 1: Local Hessian $\widetilde{\mathbf{H}}_1, \cdots, \widetilde{\mathbf{H}}_m$ well approximate the true Hessian $\mathbf{H}$.

- $\mathbf{H}$ can always be written as $\mathbf{H} = \mathbf{A}^T\mathbf{A} + \gamma \mathbf{I}_d$, for some $\mathbf{A} \in \mathbb{R}^{n \times d}$.
- Assume $\mathbf{A}$ is "incoherent".
- Uniformly partition $\mathbf{A}$ to form $\mathbf{A}_i \in \mathbb{R}^{s \times d}$.



Proof Techniques

Claim 1: Local Hessian $\widetilde{\mathbf{H}}_1, \cdots, \widetilde{\mathbf{H}}_m$ well approximate the true Hessian $\mathbf{H}$.

- $\mathbf{H}$ can always be written as $\mathbf{H} = \mathbf{A}^T\mathbf{A} + \gamma \mathbf{I}_d$, for some $\mathbf{A} \in \mathbb{R}^{n \times d}$.
- Assume $\mathbf{A}$ is "incoherent".
- Uniformly partition $\mathbf{A}$ to form $\mathbf{A}_i \in \mathbb{R}^{s \times d}$.
- Sufficiently large samples size $s = \Theta(\frac{d}{\epsilon^2} \log \frac{d}{\delta})$
- By matrix Bernstein (concentration inequality), with probability $1 - \delta$,

$$(1-\epsilon)\mathbf{A}^T\mathbf{A} \preccurlyeq \frac{n}{s}\mathbf{A}_i^T\mathbf{A}_i \preccurlyeq (1+\epsilon)\mathbf{A}^T\mathbf{A}.$$

Proof Techniques

Claim 1: Local Hessian $\widetilde{\mathbf{H}}_1, \cdots, \widetilde{\mathbf{H}}_m$ well approximate the true Hessian $\mathbf{H}$.

- $\mathbf{H}$ can always be written as $\mathbf{H} = \mathbf{A}^T\mathbf{A} + \gamma\mathbf{I}_d$, for some $\mathbf{A} \in \mathbb{R}^{n\times d}$.
- Assume $\mathbf{A}$ is "incoherent".
- Uniformly partition $\mathbf{A}$ to form $\mathbf{A}_i \in \mathbb{R}^{s\times d}$.
- Sufficiently large samples size $s = \Theta(\frac{d}{\epsilon^2}\log\frac{d}{\delta})$
- By matrix Bernstein (concentration inequality), with probability $1-\delta$,

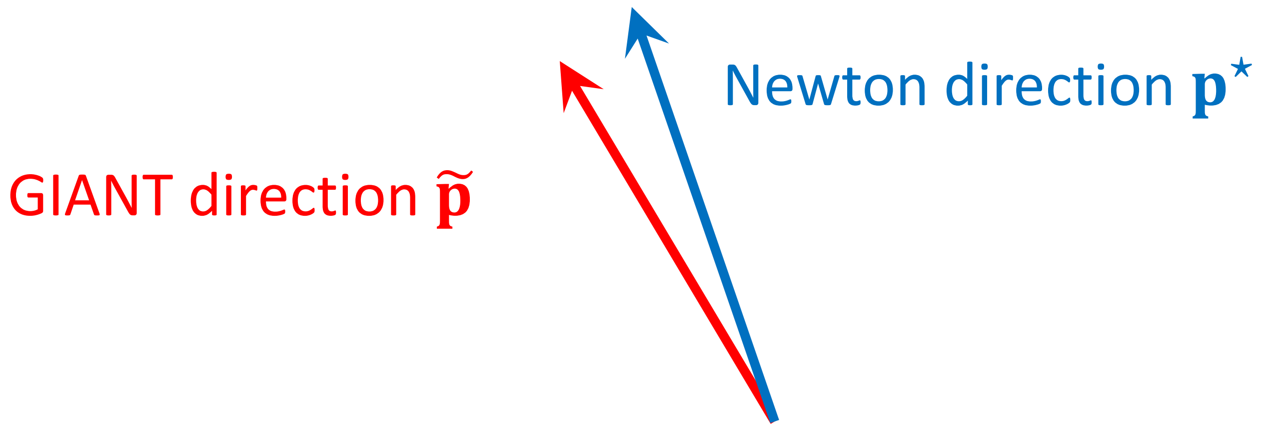
$$(1-\epsilon)\mathbf{A}^T\mathbf{A} \preccurlyeq \frac{n}{s}\mathbf{A}_i^T\mathbf{A}_i \preccurlyeq (1+\epsilon)\mathbf{A}^T\mathbf{A}.$$

- Note that $\widetilde{\mathbf{H}}_i = \frac{n}{s}\mathbf{A}_i^T\mathbf{A}_i + \gamma\mathbf{I}_d \Longrightarrow \widetilde{\mathbf{H}}_i$ well approximates $\mathbf{H}$.

Proof Techniques

Claim 2: The GIANT direction approximates $\mathbf{p}^\star = \mathbf{H}^{-1}\mathbf{g}$.

- Define the quadratic function $\phi(\mathbf{p}) \triangleq \frac{1}{2}\mathbf{p}^T\mathbf{H}\mathbf{p} - \mathbf{p}^T\mathbf{g} \quad (\leq 0)$



Proof Techniques

Claim 2: The GIANT direction approximates $\mathbf{p}^\star = \mathbf{H}^{-1}\mathbf{g}$.

- Define the quadratic function $\phi(\mathbf{p}) \triangleq \frac{1}{2}\mathbf{p}^T\mathbf{H}\mathbf{p} - \mathbf{p}^T\mathbf{g} \quad (\leq 0)$
- The exact Newton direction is $\mathbf{p}^\star = \mathbf{H}^{-1}\mathbf{g} = \operatorname*{argmin}_{\mathbf{p}} \phi(\mathbf{p})$

Proof Techniques

Claim 2: The GIANT direction approximates $\mathbf{p}^\star = \mathbf{H}^{-1}\mathbf{g}$.

- Define the quadratic function $\phi(\mathbf{p}) \triangleq \frac{1}{2}\mathbf{p}^T\mathbf{H}\mathbf{p} - \mathbf{p}^T\mathbf{g} \quad (\leq 0)$
- The exact Newton direction is $\mathbf{p}^\star = \mathbf{H}^{-1}\mathbf{g} = \underset{\mathbf{p}}{\text{argmin}}\ \phi(\mathbf{p})$
- The GIANT directions is $\tilde{\mathbf{p}} \triangleq \frac{1}{m}\sum_{i=1}^{m}\tilde{\mathbf{p}}_i \triangleq \frac{1}{m}\sum_{i=1}^{m}\tilde{\mathbf{H}}_i^{-1}\mathbf{g}$
- Conditioning on **Claim 1** that $\widetilde{\mathbf{H}}_i$ well approximates $\mathbf{H}$, we get

$$\phi(\mathbf{p}^\star) \leq \phi(\tilde{\mathbf{p}}) \leq \left(1 - \alpha^2\right) \cdot \phi(\mathbf{p}^\star), \quad \text{where } \alpha = \left(\frac{\epsilon}{\sqrt{m}} + \epsilon^2\right)$$

Reference:

W, Gittens, & Mahoney: Sketched Ridge Regression: Optimization Perspective, Statistical Perspective, and Model Averaging. In *ICML* 2017.

Proof Techniques

1. Use **Claim 2** that $\phi(\mathbf{p}^\star) \leq \phi(\tilde{\mathbf{p}}) \leq (1 - \alpha^2) \cdot \phi(\mathbf{p}^\star),$ where $\alpha = (\frac{\epsilon}{\sqrt{m}} + \epsilon^2)$
2. Follow the standard convergence analysis of Newton's method.

$\Longrightarrow$ Convergence of GIANT!

Conclusions & Future Work

Conclusions & Future Work

- GIANT's theory beats the existing works.
  - Assume the objective function is strongly convex and Lipschitz smooth.
- GIANT has good empirical performance on computer cluster.
  - Beats AGD, L-BFGS, and DANE.

Conclusions & Future Work

- GIANT's theory beats the existing works.
  - Assume the objective function is strongly convex and Lipschitz smooth.
- GIANT has good empirical performance on computer cluster.

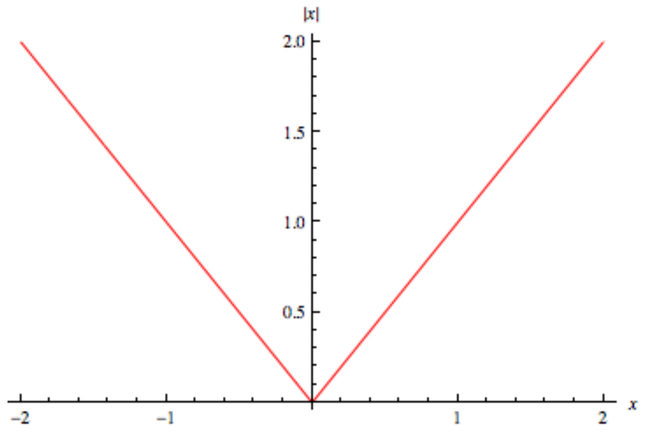
Conclusions & Future Work

- GIANT's theory beats the existing works.
  - Assume the objective function is strongly convex and Lipschitz smooth.
- GIANT has good e[…] computer cluster.

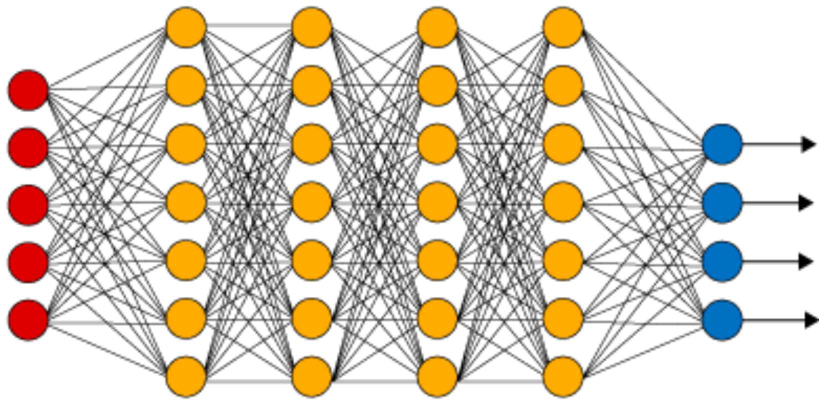
Counter-examples

LASSO

$$f(\mathbf{w}) = \frac{1}{2n}\sum_{j=1}^{n}\left(\mathbf{w}^T\mathbf{x}_j - y_j\right)^2 + \gamma\|\mathbf{w}\|_1$$



Neural Networks



Conclusions & Future Work

- GIANT's theory beats the existing works.
  - Assume the objective function is strongly convex and Lipschitz smooth.
- GIANT has good e... computer cluster.

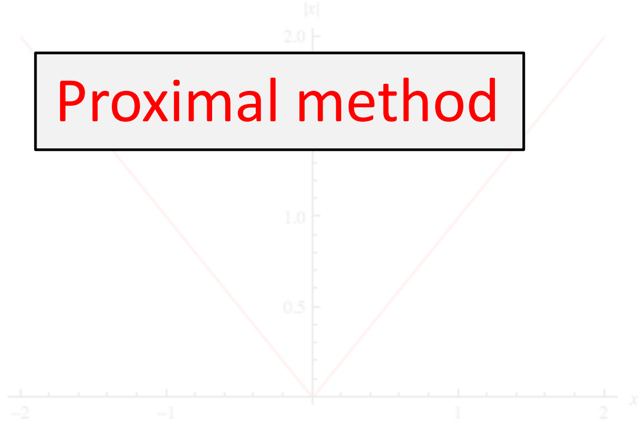
Counter-examples

LASSO

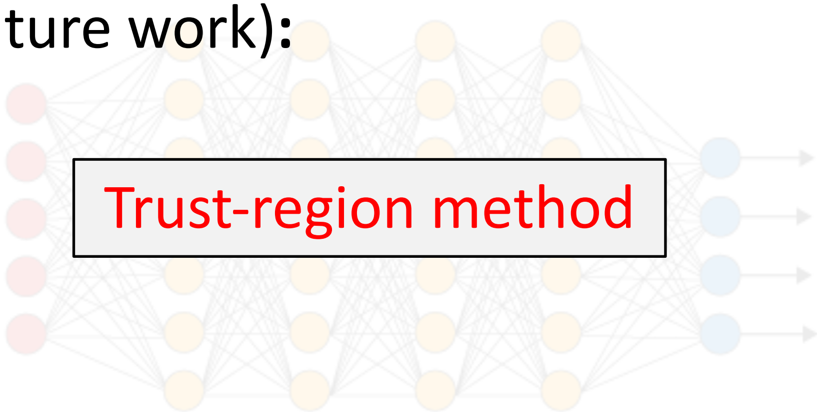
Neural Networks

$$f(\mathbf{w}) = \frac{1}{2n}\sum_{j=1}^{n}(\mathbf{w}^T\mathbf{x}_j - y_j)^2 + \gamma\|\mathbf{w}\|_1$$

Extensions of GIANT (our future work):



Proximal method



Trust-region method

Thank You!