Parallel Computing for Machine Learning
(Part 1)

Shusen Wang

Why parallel computing for ML?

- Deep learning models are big: ResNet-50 has 25M parameters. ResNet-50 2500
- Big models are trained on big data, e.g., ImageNet has 14M images.

Why parallel computing for ML?

- Deep learning models are big: ResNet-50 has 25M parameters.

- Big models are trained on big data, e.g., ImageNet has 14M images.

- Big model + big data ➜ Big computation cost.

- Example: Training ResNet-50 on ImageNet (run 90-epochs) using a single NVIDIA M40 GPU takes 14 days.

Why parallel computing for ML?

- Deep learning models are big: ResNet-50 has 25M parameters.

- Big models are trained on big data, e.g., ImageNet has 14M images.

- Big model + big data ➜ Big computation cost.

- Example: Training ResNet-50 on ImageNet (run 90-epochs) using a single NVIDIA M40 GPU takes 14 days.

- Parallel computing: using multiple processors to make the computation faster (in terms of wall-clock time.)

Toy Example: Least Squares Regression

Linear Predictor

- Inputs: $\mathbf{x} \in \mathbb{R}^d$ (e.g., features of a house).
- Prediction: $f(\mathbf{x}) = \mathbf{x}^T \textcolor{red}{\mathbf{w}}$ (e.g., housing price).

Linear Predictor

- Inputs: $\mathbf{x} \in \mathbb{R}^d$ (e.g., features of a house).
- Prediction: $\boxed{f(\mathbf{x}) = \mathbf{x}^T \mathbf{w}}$ (e.g., housing price).

- $f(\mathbf{x}) = w_1 x_1 + w_2 x_2 + \cdots + w_d x_d$
- $w_1, w_2, \cdots, w_d$: weights
- x_1: # of bedrooms
- x_2: # of bathroom
- x_3: square feet
- x_4: age of house
- …

Linear Predictor

- Inputs: $\mathbf{x} \in \mathbb{R}^d$ (e.g., features of a house).
- Prediction: $f(\mathbf{x}) = \mathbf{x}^T \mathbf{w}$ (e.g., housing price).

Features of a House
$\mathbf{x} \in \mathbb{R}^d$

Price = $0.5M

Prediction:
$f(\mathbf{x}) = \mathbf{x}^T \mathbf{w}$

Linear Predictor

- Inputs: $\mathbf{x} \in \mathbb{R}^d$ (e.g., features of a house).
- Prediction: $f(\mathbf{x}) = \mathbf{x}^T \mathbf{w}$ (e.g., housing price).

Question: How to find $\mathbf{w}$?

Features of a House
$\mathbf{x} \in \mathbb{R}^d$

Price = $0.5M

Prediction:
$f(\mathbf{x}) = \mathbf{x}^T \mathbf{w}$

Least Squares Regression

- Inputs: $\mathbf{x} \in \mathbb{R}^d$ (e.g., features of a house).

- Prediction: $f(\mathbf{x}) = \mathbf{x}^T \mathbf{w}$ (e.g., housing price).

Question: How to find **w**?

- Training inputs: $\mathbf{x}_1, \cdots, \mathbf{x}_n \in \mathbb{R}^d$.

- Training targets: $y_1, \cdots, y_n \in \mathbb{R}$.

$\cdots$

totally n houses

Least Squares Regression

- Inputs: $\mathbf{x} \in \mathbb{R}^d$ (e.g., features of a house).

- Prediction: $f(\mathbf{x}) = \mathbf{x}^T \mathbf{w}$ (e.g., housing price).

Question: How to find $\mathbf{w}$?

- Training inputs: $\mathbf{x}_1, \cdots, \mathbf{x}_n \in \mathbb{R}^d$.

- Training targets: $y_1, \cdots, y_n \in \mathbb{R}$.

- Loss function: $L(\mathbf{w}) = \sum_{i=1}^{n} \frac{1}{2} \left(\mathbf{x}_i^T \mathbf{w} - y_i \right)^2$.

Least Squares Regression

- Inputs: $\mathbf{x} \in \mathbb{R}^d$ (e.g., features of a house).

- Prediction: $f(\mathbf{x}) = \mathbf{x}^T \mathbf{w}$ (e.g., housing price).

Question: How to find $\mathbf{w}$?

- Training inputs: $\mathbf{x}_1, \cdots, \mathbf{x}_n \in \mathbb{R}^d$.

- Training targets: $y_1, \cdots, y_n \in \mathbb{R}$.

- Loss function: $L(\mathbf{w}) = \sum_{i=1}^{n} \frac{1}{2} \left(\mathbf{x}_i^T \mathbf{w} - y_i \right)^2$.

- Least squares regression: $\mathbf{w}^\star = \underset{\mathbf{w}}{\operatorname{argmin}} \, L(\mathbf{w})$.

Parallel Gradient Descent for Least Squares

Parallel Gradient Descent

Example: GD for least squares regression model

- Loss function: $L(\mathbf{w}) = \sum_{i=1}^{n} \frac{1}{2} \left(\mathbf{x}_i^T \mathbf{w} - y_i \right)^2.$

Parallel Gradient Descent

Example: GD for least squares regression model

- Loss function: $L(\mathbf{w}) = \sum_{i=1}^{n} \frac{1}{2}\left(\mathbf{x}_i^T \mathbf{w} - y_i\right)^2.$

Gradient: $g(\mathbf{w}) = \frac{\partial\, L(\mathbf{w})}{\partial\, \mathbf{w}} = \sum_{i=1}^{n} \frac{\partial\, \frac{1}{2}\left(\mathbf{x}_i^T \mathbf{w} - y_i\right)^2}{\partial\, \mathbf{w}} = \sum_{i=1}^{n}\left(\mathbf{x}_i^T \mathbf{w} - y_i\right)\mathbf{x}_i$

Parallel Gradient Descent

Example: GD for least squares regression model

- Loss function: $L(\mathbf{w}) = \sum_{i=1}^{n} \frac{1}{2} \left(\mathbf{x}_i^T \mathbf{w} - y_i \right)^2.$

- Gradient: $g(\mathbf{w}) = \sum_{i=1}^{n} g_i(\mathbf{w}),$ where $g_i(\mathbf{w}) = \left(\mathbf{x}_i^T \mathbf{w} - y_i \right) \mathbf{x}_i.$

Parallel Gradient Descent

Example: GD for least squares regression model

- Loss function: $L(\mathbf{w}) = \sum_{i=1}^{n} \frac{1}{2}\left(\mathbf{x}_i^T \mathbf{w} - y_i\right)^2.$

- Gradient: $g(\mathbf{w}) = \sum_{i=1}^{n} g_i(\mathbf{w}),$ where $g_i(\mathbf{w}) = \left(\mathbf{x}_i^T \mathbf{w} - y_i\right)\mathbf{x}_i.$

- Gradient descent: $\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \cdot g(\mathbf{w}_t).$

Parallel Gradient Descent

Example: GD for least squares regression model

- Loss function: $L(\mathbf{w}) = \sum_{i=1}^{n} \frac{1}{2} \left(\mathbf{x}_i^T \mathbf{w} - y_i \right)^2.$

- Gradient: $g(\mathbf{w}) = \sum_{i=1}^{n} g_i(\mathbf{w}),$ where $g_i(\mathbf{w}) = \left(\mathbf{x}_i^T \mathbf{w} - y_i \right) \mathbf{x}_i.$

- Gradient descent: $\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \cdot g(\mathbf{w}_t).$

- The bottleneck of GD is at computing the gradient.

- It is expensive if #samples and #parameters are both big.

Parallel Gradient Descent

Example: GD for least squares regression model

- Loss function: $L(\mathbf{w}) = \sum_{i=1}^{n} \frac{1}{2} \left(\mathbf{x}_i^T \mathbf{w} - y_i \right)^2.$

- Gradient: $g(\mathbf{w}) = \sum_{i=1}^{n} g_i(\mathbf{w})$, where $g_i(\mathbf{w}) = \left(\mathbf{x}_i^T \mathbf{w} - y_i \right) \mathbf{x}_i.$

Parallel GD using 2 processors

- $g(\mathbf{w}) = g_1(\mathbf{w}) + g_2(\mathbf{w}) + \cdots + g_{\frac{n}{2}}(\mathbf{w}) + g_{\frac{n}{2}+1}(\mathbf{w}) + \cdots + g_{n-1}(\mathbf{w}) + g_n(\mathbf{w}).$

Parallel Gradient Descent

Example: GD for least squares regression model

- Loss function: $L(\mathbf{w}) = \sum_{i=1}^{n} \frac{1}{2} \left(\mathbf{x}_i^T \mathbf{w} - y_i \right)^2.$

- Gradient: $g(\mathbf{w}) = \sum_{i=1}^{n} g_i(\mathbf{w}),$ where $g_i(\mathbf{w}) = \left(\mathbf{x}_i^T \mathbf{w} - y_i \right) \mathbf{x}_i.$

Parallel GD using 2 processors

- $g(\mathbf{w}) = g_1(\mathbf{w}) + g_2(\mathbf{w}) + \cdots + g_{\frac{n}{2}}(\mathbf{w}) + g_{\frac{n}{2}+1}(\mathbf{w}) + \cdots + g_{n-1}(\mathbf{w}) + g_n(\mathbf{w}).$

Processor 1

Parallel Gradient Descent

Example: GD for least squares regression model

- Loss function: $L(\mathbf{w}) = \sum_{i=1}^{n} \frac{1}{2}\left(\mathbf{x}_i^T \mathbf{w} - y_i\right)^2$.

- Gradient: $g(\mathbf{w}) = \sum_{i=1}^{n} g_i(\mathbf{w})$, where $g_i(\mathbf{w}) = \left(\mathbf{x}_i^T \mathbf{w} - y_i\right)\mathbf{x}_i$.

Parallel GD using 2 processors

- $g(\mathbf{w}) = g_1(\mathbf{w}) + g_2(\mathbf{w}) + \cdots + g_{\frac{n}{2}}(\mathbf{w}) + g_{\frac{n}{2}+1}(\mathbf{w}) + \cdots + g_{n-1}(\mathbf{w}) + g_n(\mathbf{w})$.

Processor 1

Processor 2

Parallel Gradient Descent

Example: GD for least squares regression model

- Loss function: $L(\mathbf{w}) = \sum_{i=1}^{n} \frac{1}{2}\left(\mathbf{x}_i^T \mathbf{w} - y_i\right)^2.$

- Gradient: $g(\mathbf{w}) = \sum_{i=1}^{n} g_i(\mathbf{w}),$ where $g_i(\mathbf{w}) = \left(\mathbf{x}_i^T \mathbf{w} - y_i\right)\mathbf{x}_i.$

Parallel GD using 2 processors

- $g(\mathbf{w}) = \underbrace{g_1(\mathbf{w}) + g_2(\mathbf{w}) + \cdots + g_{\frac{n}{2}}(\mathbf{w})}_{= \tilde{\mathbf{g}}_1} + \underbrace{g_{\frac{n}{2}+1}(\mathbf{w}) + \cdots + g_{n-1}(\mathbf{w}) + g_n(\mathbf{w})}_{= \tilde{\mathbf{g}}_2}.$

Parallel Gradient Descent

Example: GD for least squares regression model

- Loss function: $L(\mathbf{w}) = \sum_{i=1}^{n} \frac{1}{2} \left(\mathbf{x}_i^T \mathbf{w} - y_i \right)^2$.

- Gradient: $g(\mathbf{w}) = \sum_{i=1}^{n} g_i(\mathbf{w})$, where $g_i(\mathbf{w}) = \left(\mathbf{x}_i^T \mathbf{w} - y_i \right) \mathbf{x}_i$.

Parallel GD using 2 processors

- $g(\mathbf{w}) = \underbrace{g_1(\mathbf{w}) + g_2(\mathbf{w}) + \cdots + g_{\frac{n}{2}}(\mathbf{w})}_{= \tilde{\mathbf{g}}_1} + \underbrace{g_{\frac{n}{2}+1}(\mathbf{w}) + \cdots + g_{n-1}(\mathbf{w}) + g_n(\mathbf{w})}_{= \tilde{\mathbf{g}}_2}$.

Aggregate: $g(\mathbf{w}) = \tilde{\mathbf{g}}_1 + \tilde{\mathbf{g}}_2$.

Communication

Two Ways of Communication
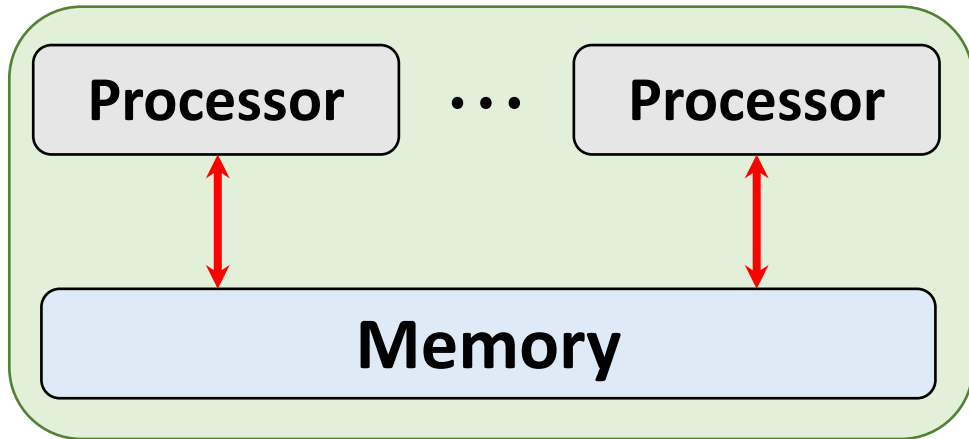
Two Ways of Communication

Processor $\cdots$ Processor

Memory

Processor $\cdots$ Processor

Memory

Node 1

Processor $\cdots$ Processor

Memory

Node 2
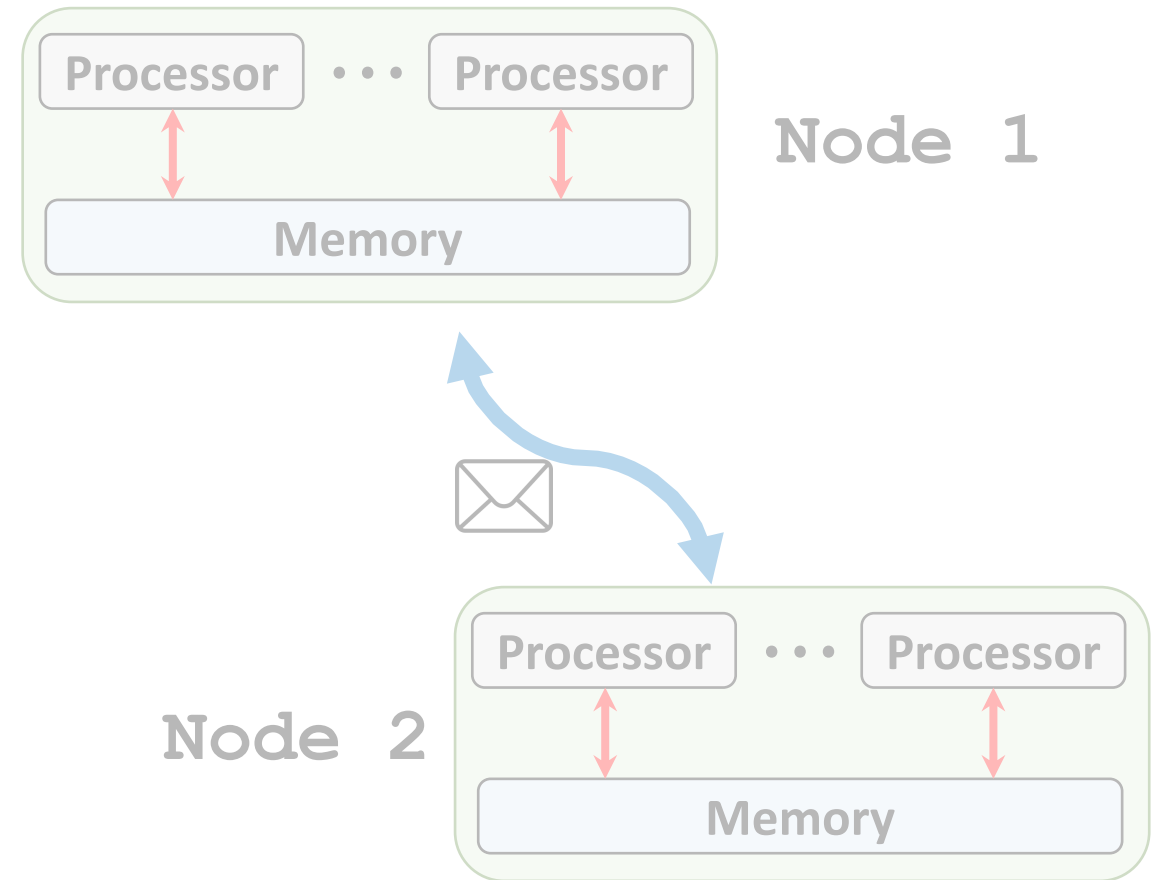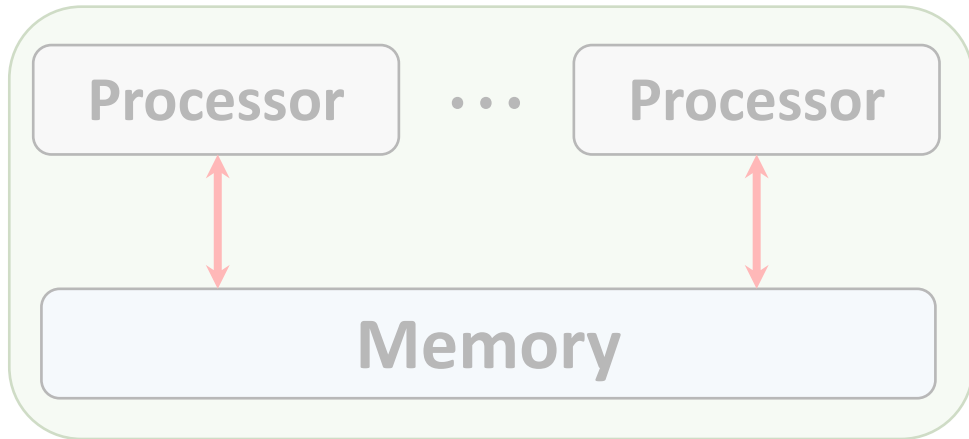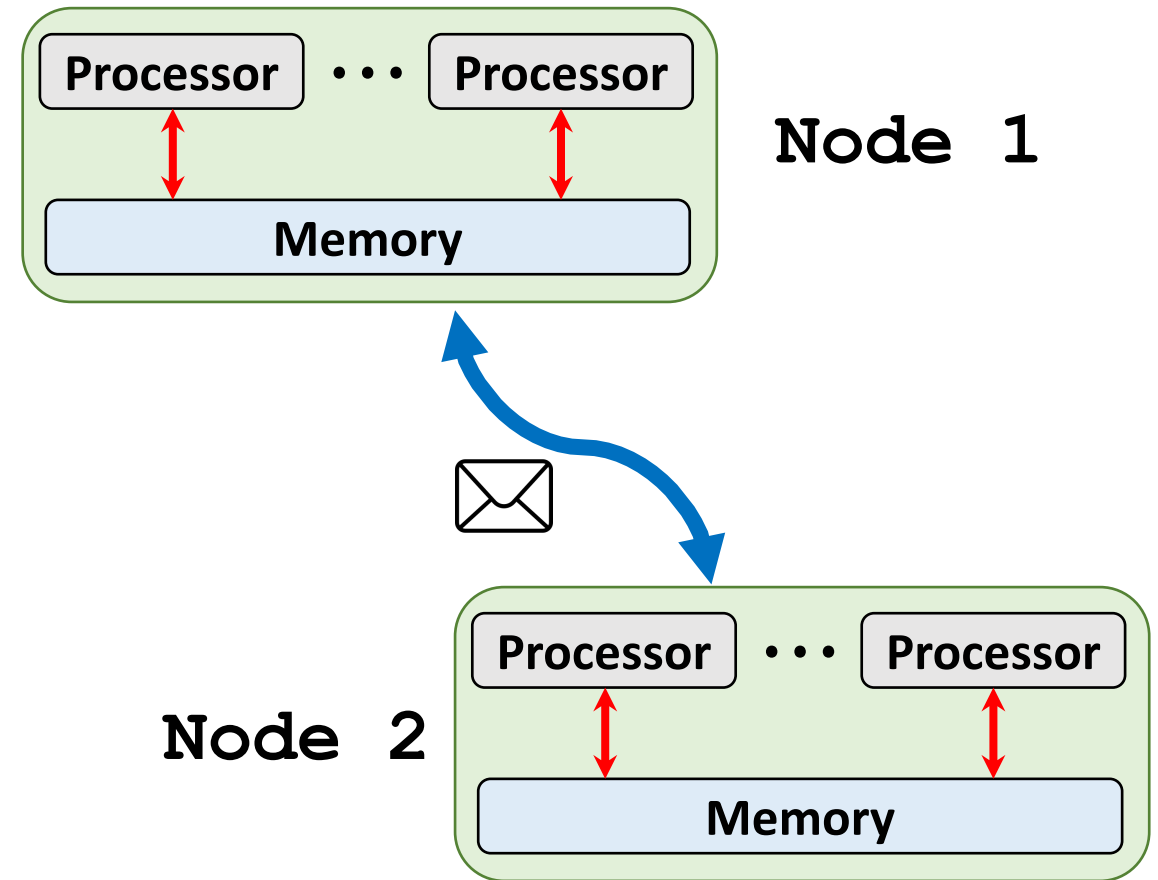
Client-Server Architecture
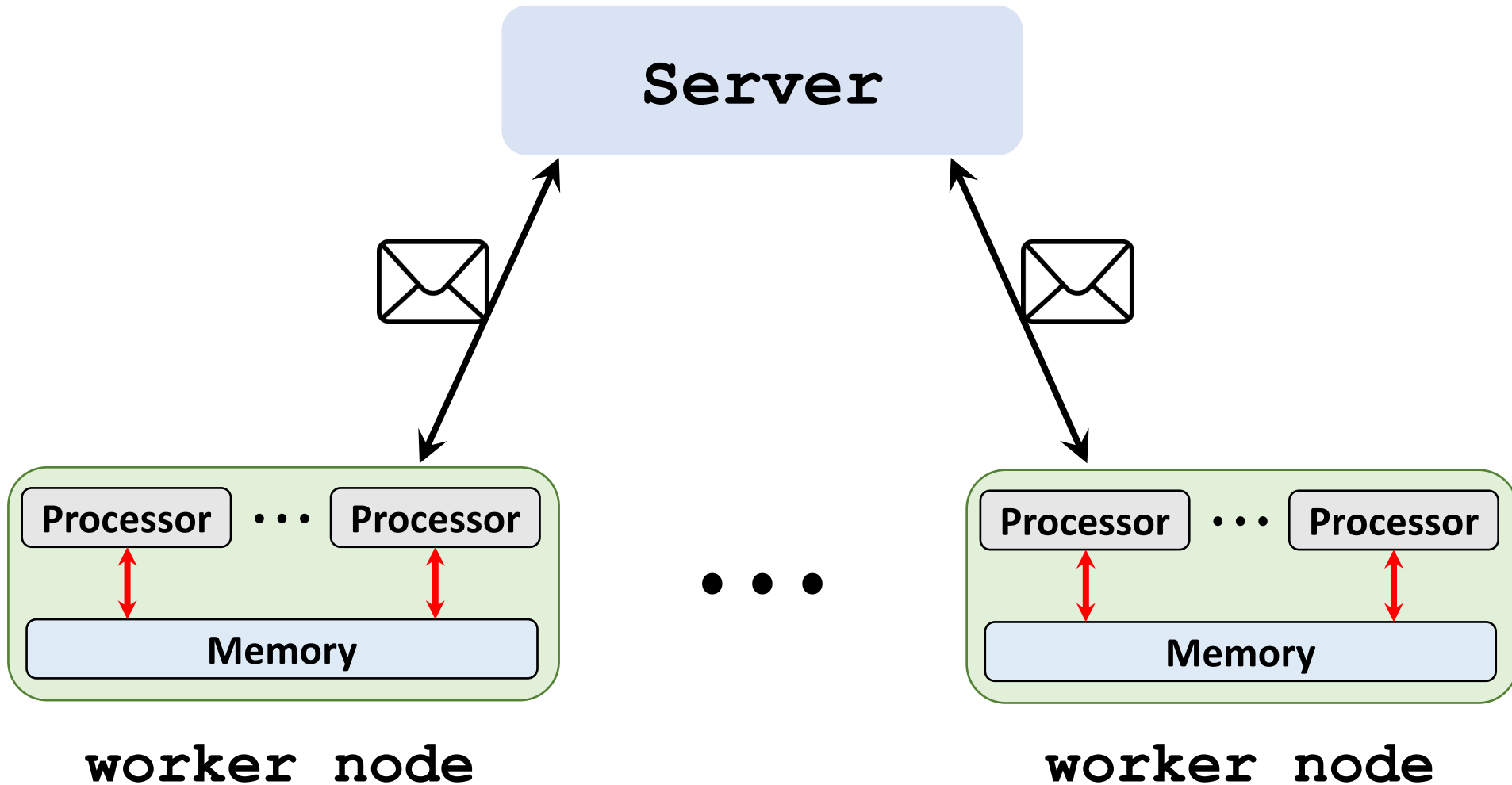
Peer-to-Peer Architecture
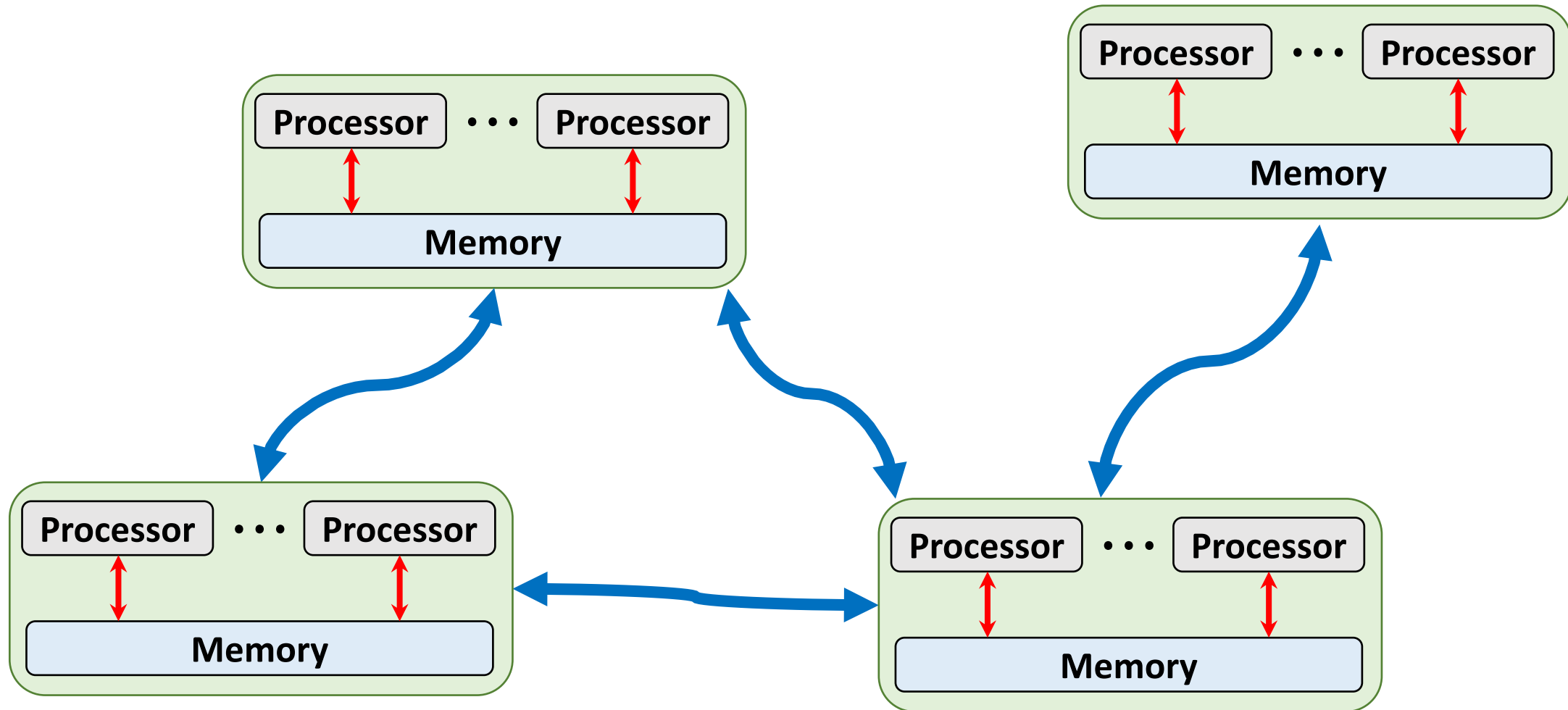
Synchronous Parallel Gradient Descent

Using MapReduce

MapReduce
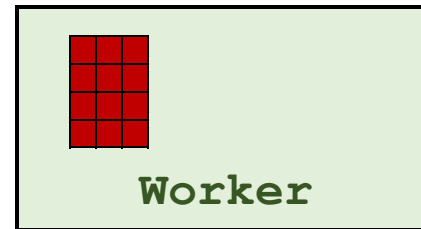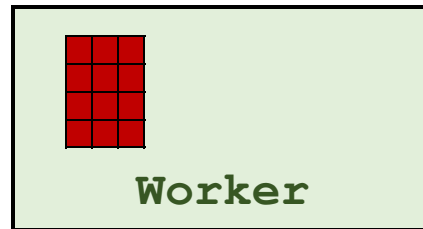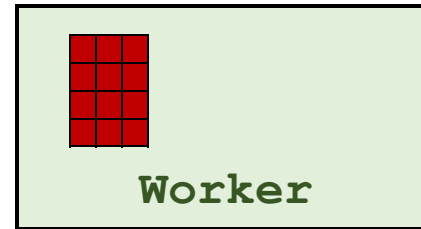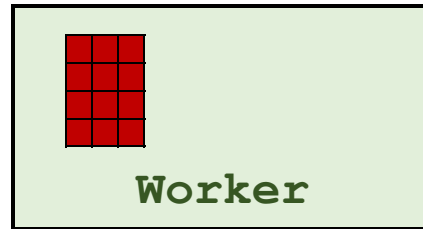
- **MapReduce** is a programming model and software system developed by Google [1].

- **Characters:** client-server architecture, message-passing communication, and bulk synchronous parallel.

Reference

1. Dean and Ghemawat: MapReduce: simplified data processing on large clusters. *Communications of the ACM,* 2008.

MapReduce

- **MapReduce** is a programming model and software system developed by Google [1]. MapReduce [1]

- **Characters:** client-server architecture, message-passing communication, and bulk synchronous parallel.

- **Apache Hadoop** [2] is an open-source implementation of MapReduce.

Reference

1. Dean and Ghemawat: MapReduce: simplified data processing on large clusters. *Communications of the ACM,* 2008.

2. https://hadoop.apache.org/

MapReduce

- **MapReduce** is a programming model and software system developed by Google [1].

- **Characters:** client-server architecture, message-passing communication, and bulk synchronous parallel.

- **Apache Hadoop** [2] is an open-source implementation of MapReduce.

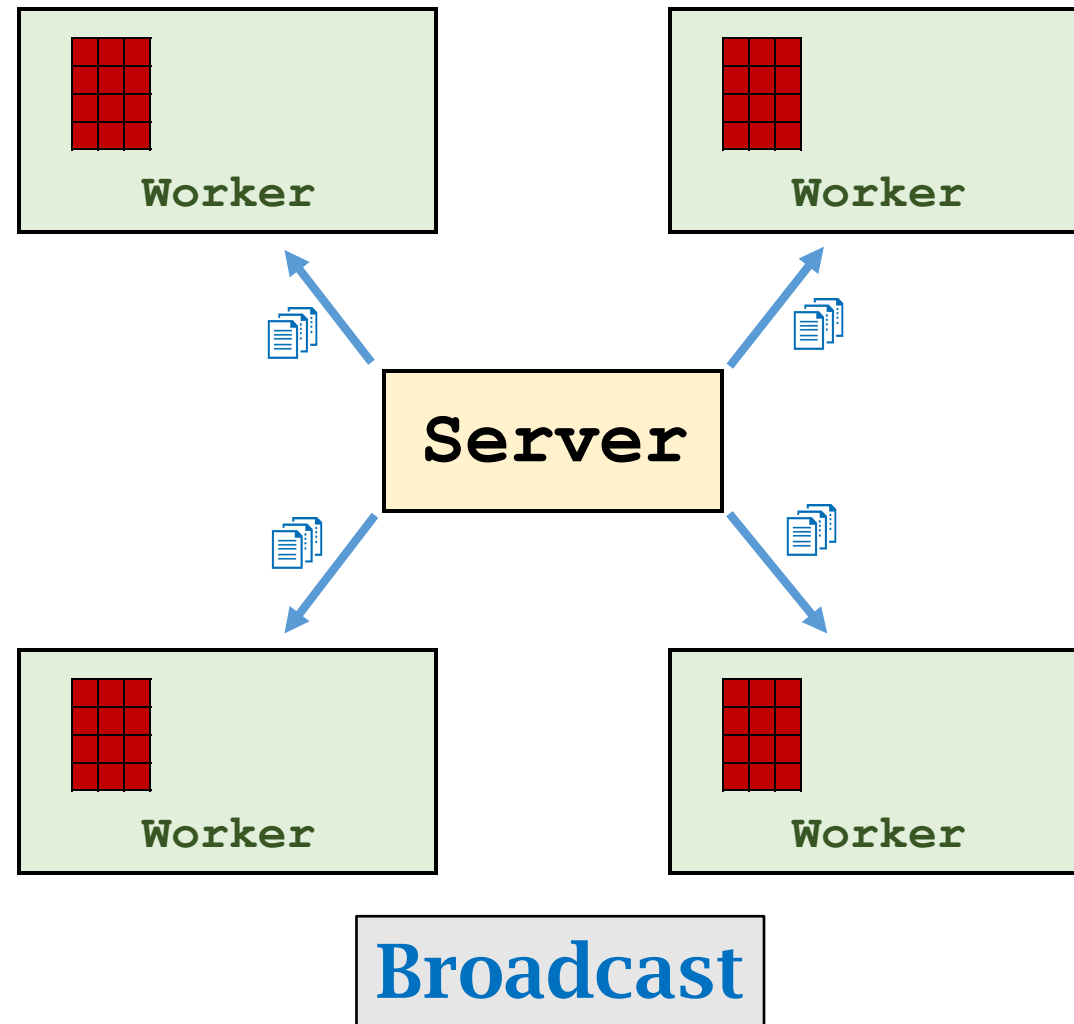- **Apache Spark** [3] is an improved open-source MapReduce.

Reference

1. Dean and Ghemawat: MapReduce: simplified data processing on large clusters. *Communications of the ACM,* 2008.

2. https://hadoop.apache.org/

3. Zaharia and others: Apache Spark: a unified engine for big data processing. *Communications of the ACM,* 2016.

MapReduce

MapReduce

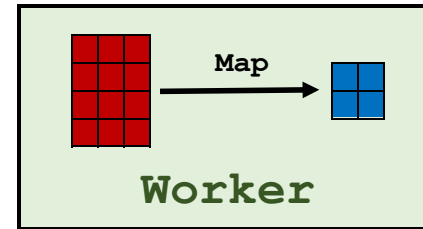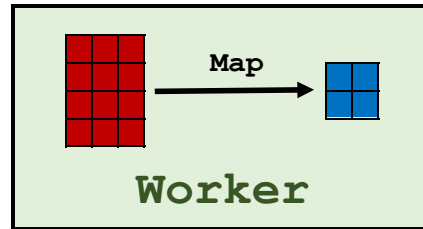
Server
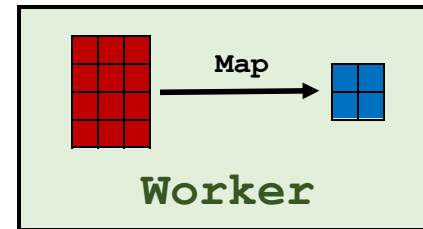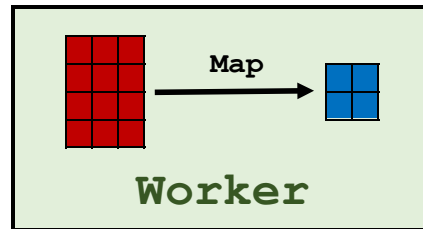
Worker   Worker

Worker   Worker

Broadcast

MapReduce

MapReduce

Data Parallelism

Targets: $\{y_i\}$

Inputs: $\{\mathbf{x}_i\}$
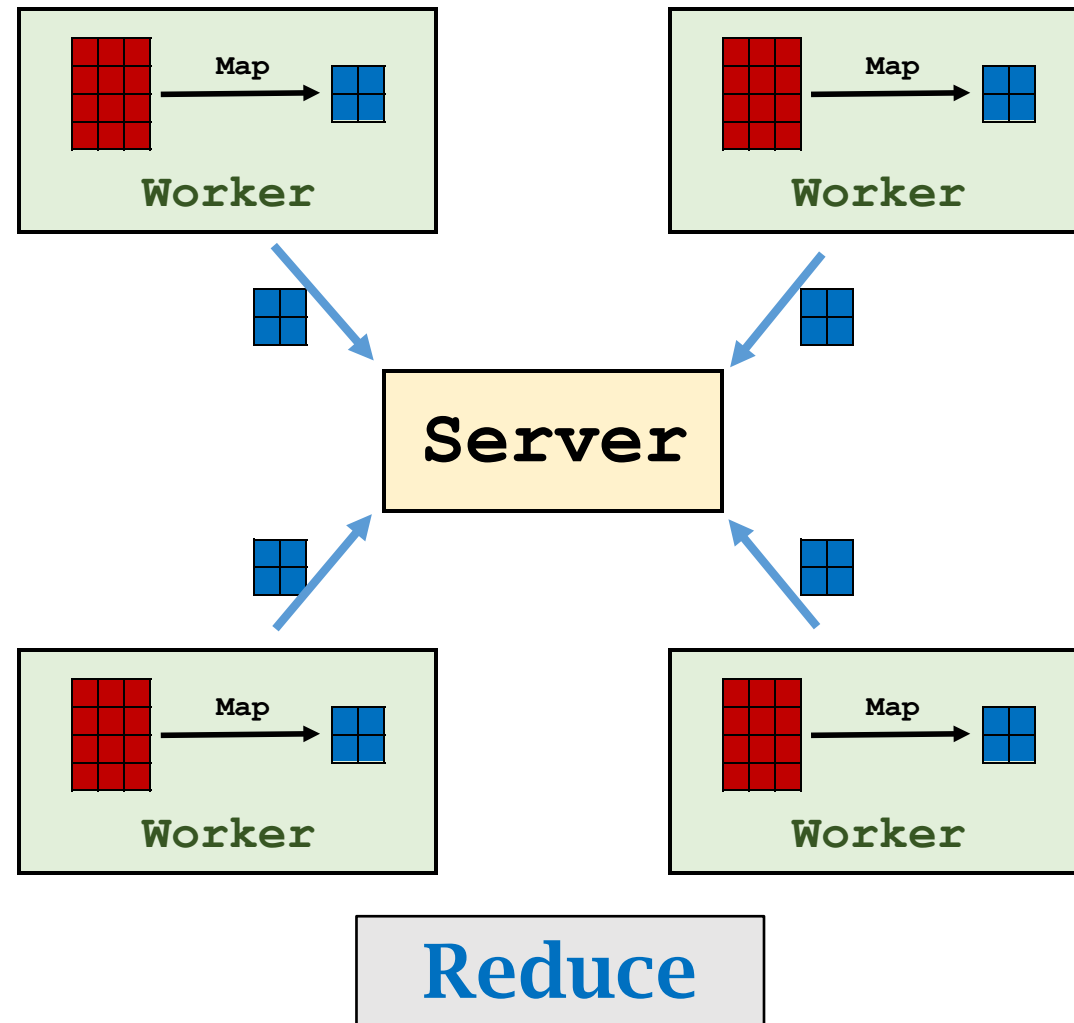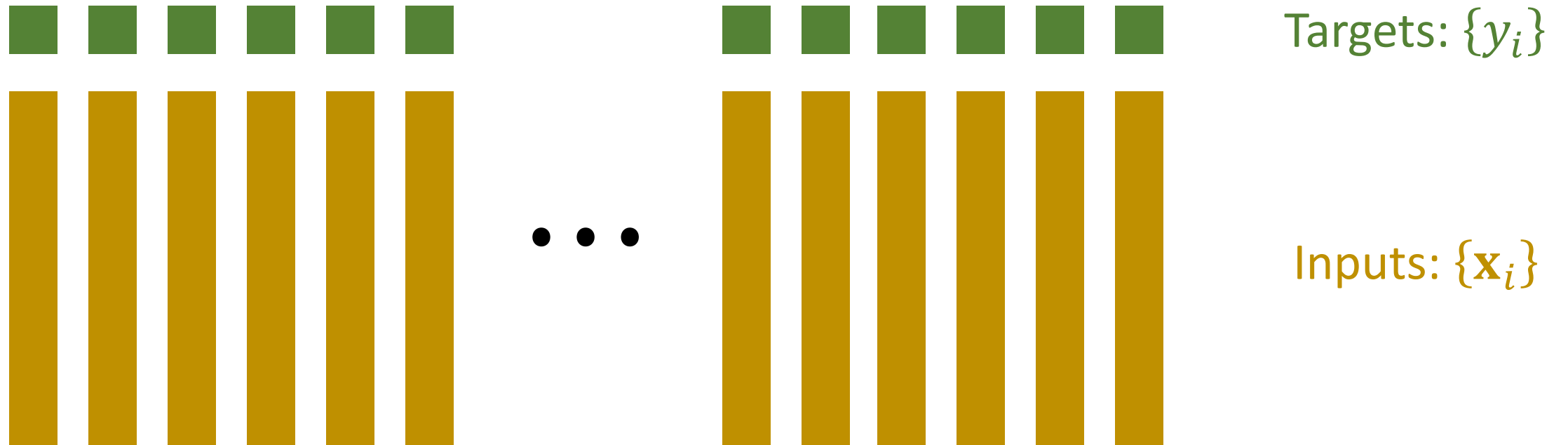
Data Parallelism

- Partition the data among worker nodes. (A node has a subset of data.)
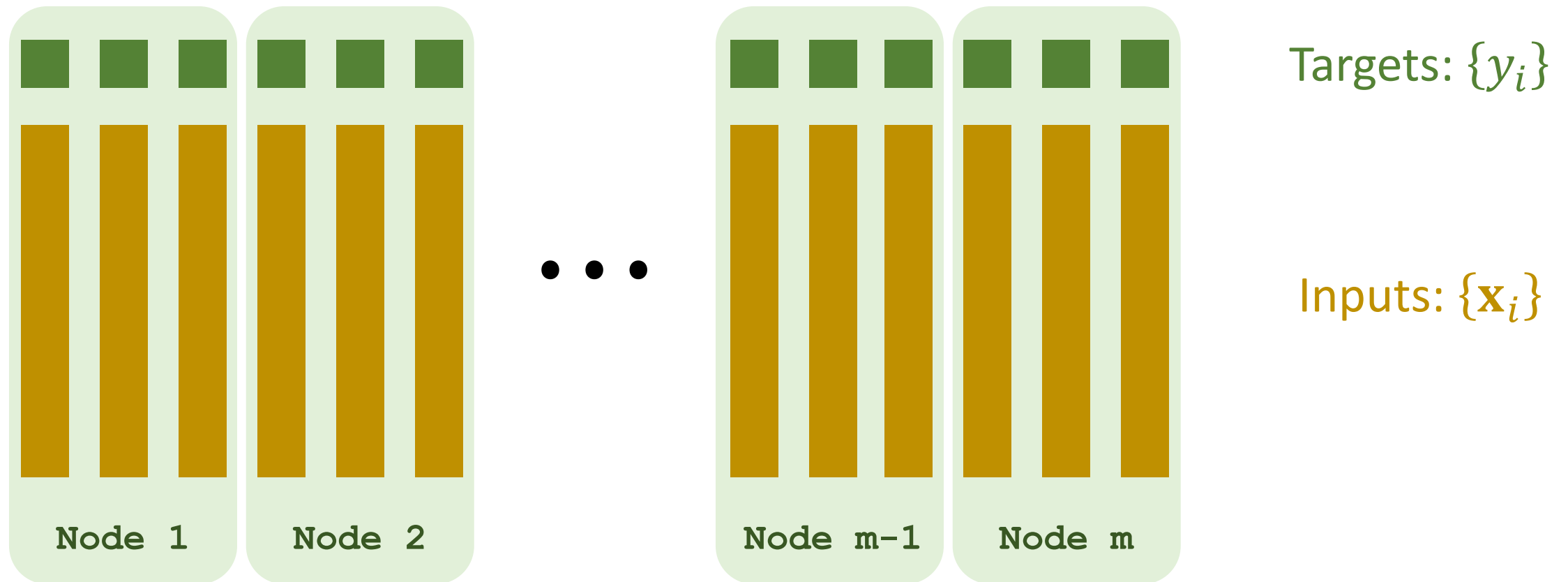
Targets: $\{y_i\}$

Inputs: $\{\mathbf{x}_i\}$

Parallel Gradient Descent Using MapReduce

- **Broadcast:** Server broadcast the up-to-date parameters $\mathbf{w}_t$ to workers.

- **Map:** Workers do computation locally.

  - Map $(\mathbf{x}_i, y_i, \mathbf{w}_t)$ to $\mathbf{g}_i = \left(\mathbf{x}_i^T \mathbf{w}_t - y_i\right) \mathbf{x}_i.$
  - Obtain n vectors: $\mathbf{g}_1, \mathbf{g}_2, \mathbf{g}_3, \cdots, \mathbf{g}_n.$
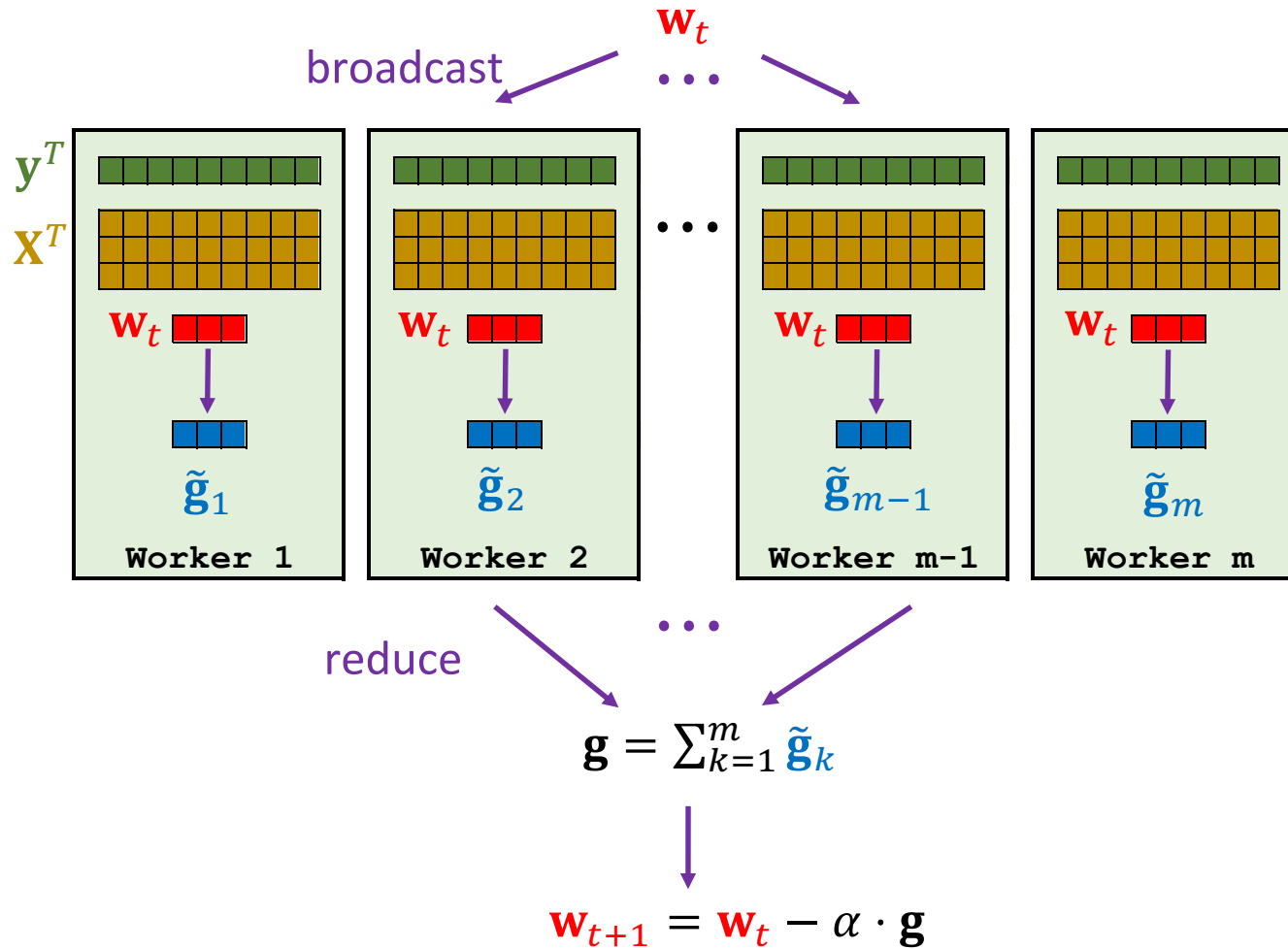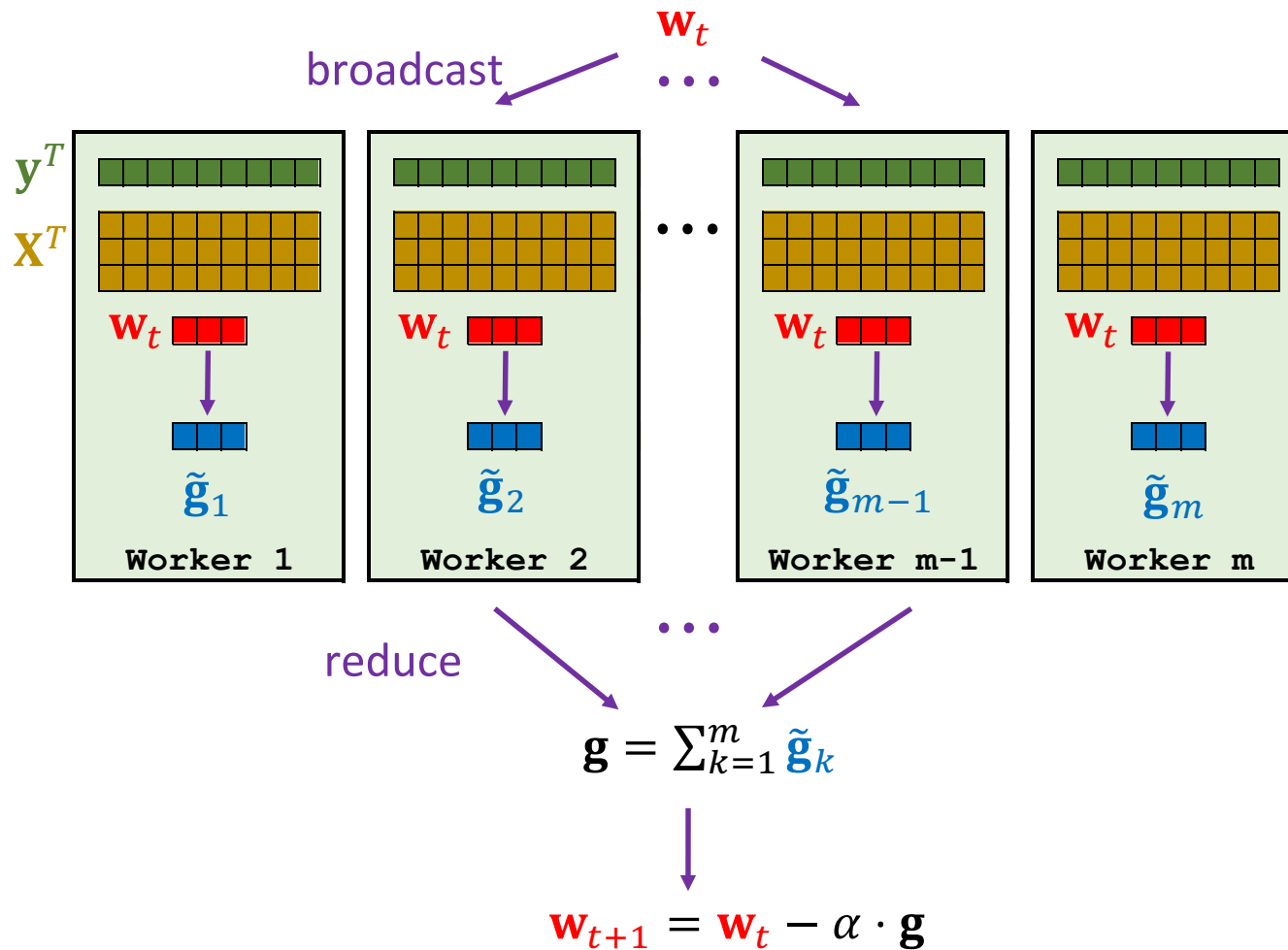
Parallel Gradient Descent Using MapReduce

- **Broadcast:** Server broadcast the up-to-date parameters $\mathbf{w}_t$ to workers.

- **Map:** Workers do computation locally.

  - Map $(\mathbf{x}_i, y_i, \mathbf{w}_t)$ to $\mathbf{g}_i = \left(\mathbf{x}_i^T \mathbf{w}_t - y_i\right) \mathbf{x}_i$.
  - Obtain n vectors: $\mathbf{g}_1, \mathbf{g}_2, \mathbf{g}_3, \cdots, \mathbf{g}_n$.

- **Reduce:** Compute the sum: $\mathbf{g} = \sum_{i=1}^{n} \mathbf{g}_i$.

  - Every worker sums all the $\{\mathbf{g}_i\}$ stored in its local memory to get a vector.
  - Then, the server sums the resulting m vectors. (There are m workers.)

- Server updates the parameters: $\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \cdot \mathbf{g}$.

Parallel Gradient Descent Using MapReduce

Parallel Gradient Descent Using MapReduce

$\mathbf{w}_t$

broadcast $\quad \cdots$

$\mathbf{y}^T$

$\mathbf{X}^T$

$\mathbf{w}_t$ $\mathbf{w}_t$ $\mathbf{w}_t$ $\mathbf{w}_t$

$\cdots$

$\tilde{\mathbf{g}}_1$ $\tilde{\mathbf{g}}_2$ $\tilde{\mathbf{g}}_{m-1}$ $\tilde{\mathbf{g}}_m$

Worker 1   Worker 2   Worker m-1   Worker m

reduce $\quad \cdots$

$$\mathbf{g} = \sum_{k=1}^{m} \tilde{\mathbf{g}}_k$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \cdot \mathbf{g}$$

- Every worker stores $\frac{1}{m}$ of the data.
- Every worker does $\frac{1}{m}$ of the computation.
- Is the runtime reduced to $\frac{1}{m}$?

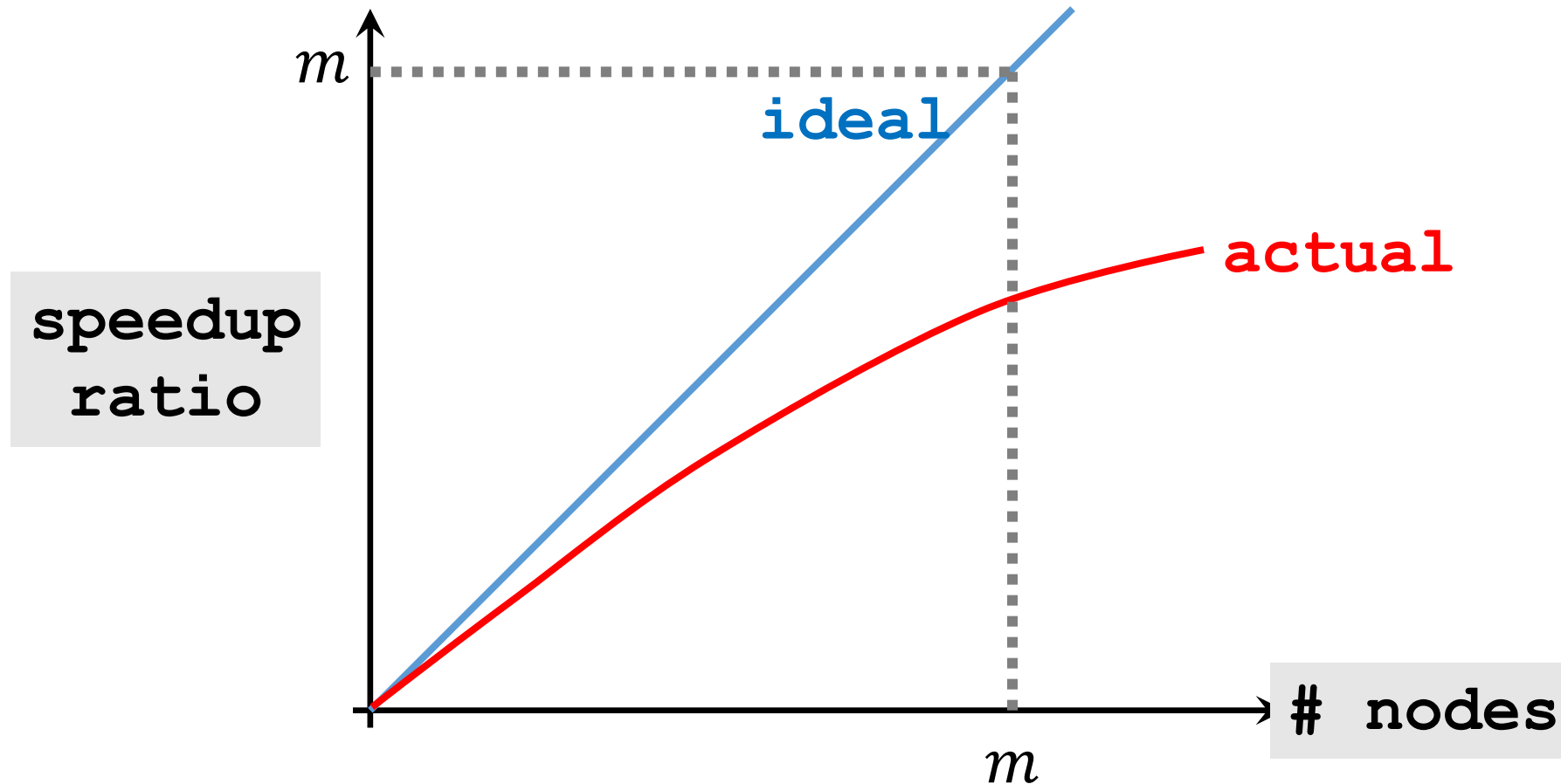- No. Because communication and synchronization must be considered.

Speedup Ratio

$$\text{speedup ratio} \;=\; \frac{\text{wall clock time using one node}}{\text{wall clock time using } m \text{ nodes}}$$
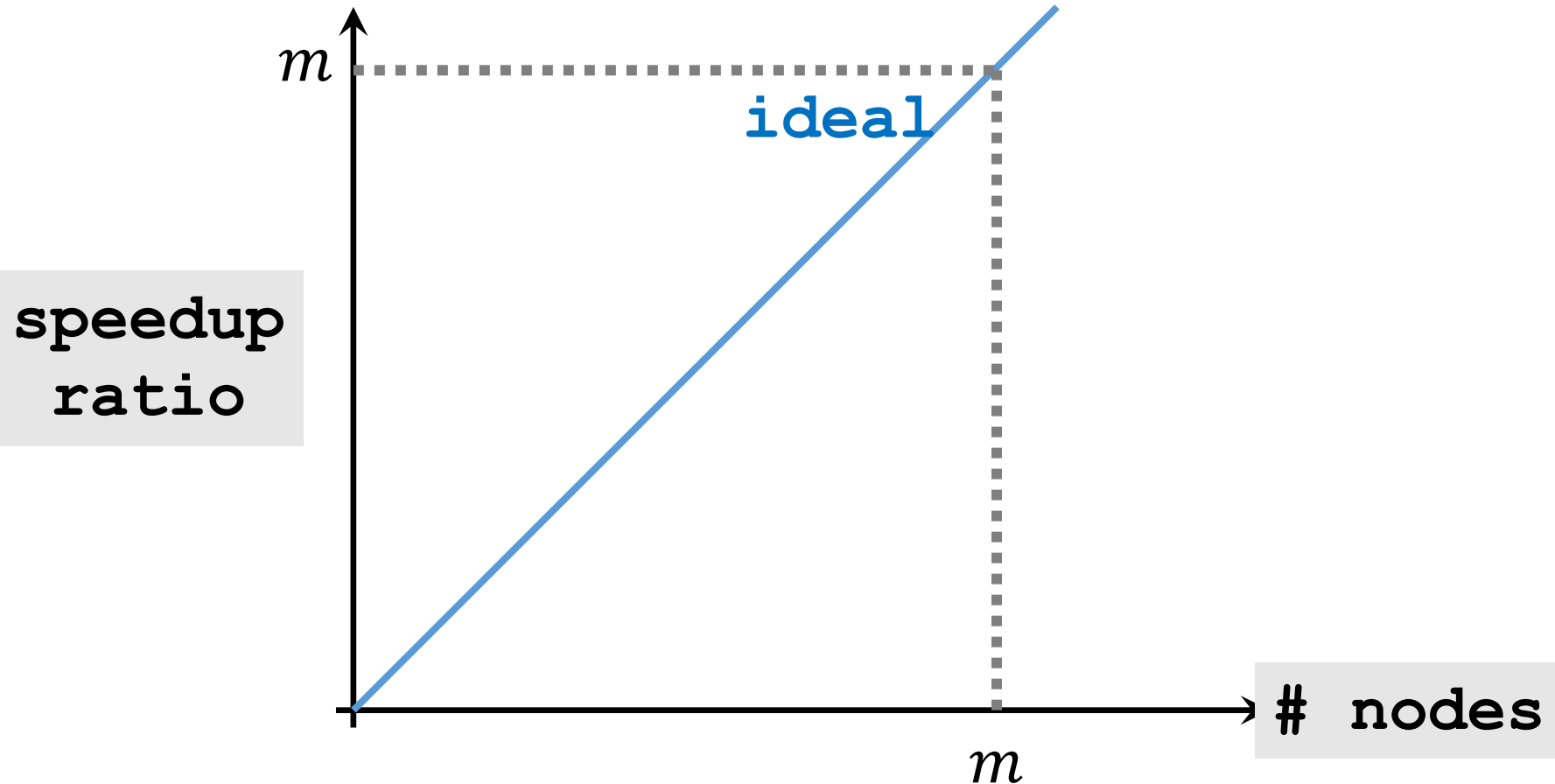
Speedup Ratio

$$\text{speedup ratio} = \frac{\text{wall clock time using one node}}{\text{wall clock time using } m \text{ nodes}}$$

Communication Cost

- Communication complexity: How many words are transmitted between server and workers.

  - Proportional to number of parameters.

 - Grow with number of worker nodes.

Communication Cost

- Communication complexity: How many words are transmitted between server and workers.

  - Proportional to number of parameters.

  - Grow with number of worker nodes.

- Latency: How much time it takes for a packet of data to get from one point to another. (Determined by the compute network.)

Communication Cost

- Communication complexity: How many words are transmitted between server and workers.
    - Proportional to number of parameters.
    - Grow with number of worker nodes.

- Latency: How much time it takes for a packet of data to get from one point to another. (Determined by the compute network.)

- Communication time: $\dfrac{\text{complexity}}{\text{bandwith}} + \text{latency}$.

Bulk Synchronous

Broadcast Computation Reduce
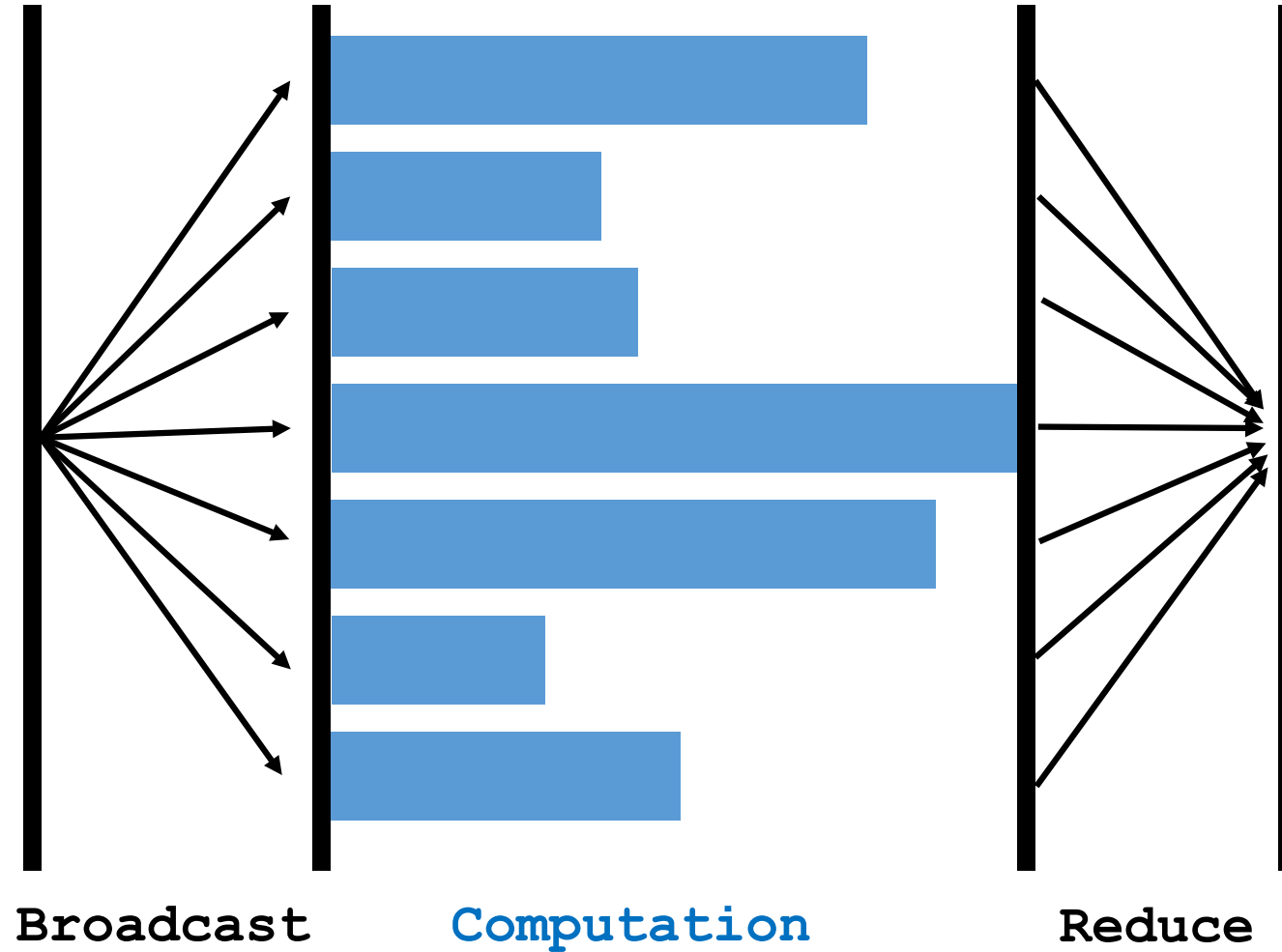
Synchronization Cost

Question: What if a node fails and then restart?

- This node will be much slower than all the others.

- It is called straggler.

- Straggler effect:
  - The wall-clock time is determined by the slowest node.
  - It is a consequence of synchronization.

Recap

- Gradient descent can be implemented using MapReduce.
- Data parallelism: Data are partitioned among the workers.
- One gradient descent step requires a broadcast, a map, and a reduce.

Recap

- Gradient descent can be implemented using MapReduce.

- Data parallelism: Data are partitioned among the workers.

- One gradient descent step requires a broadcast, a map, and a reduce.

- Cost: computation, communication, and synchronization.

- Using m workers, the speedup ratio is lower than m.