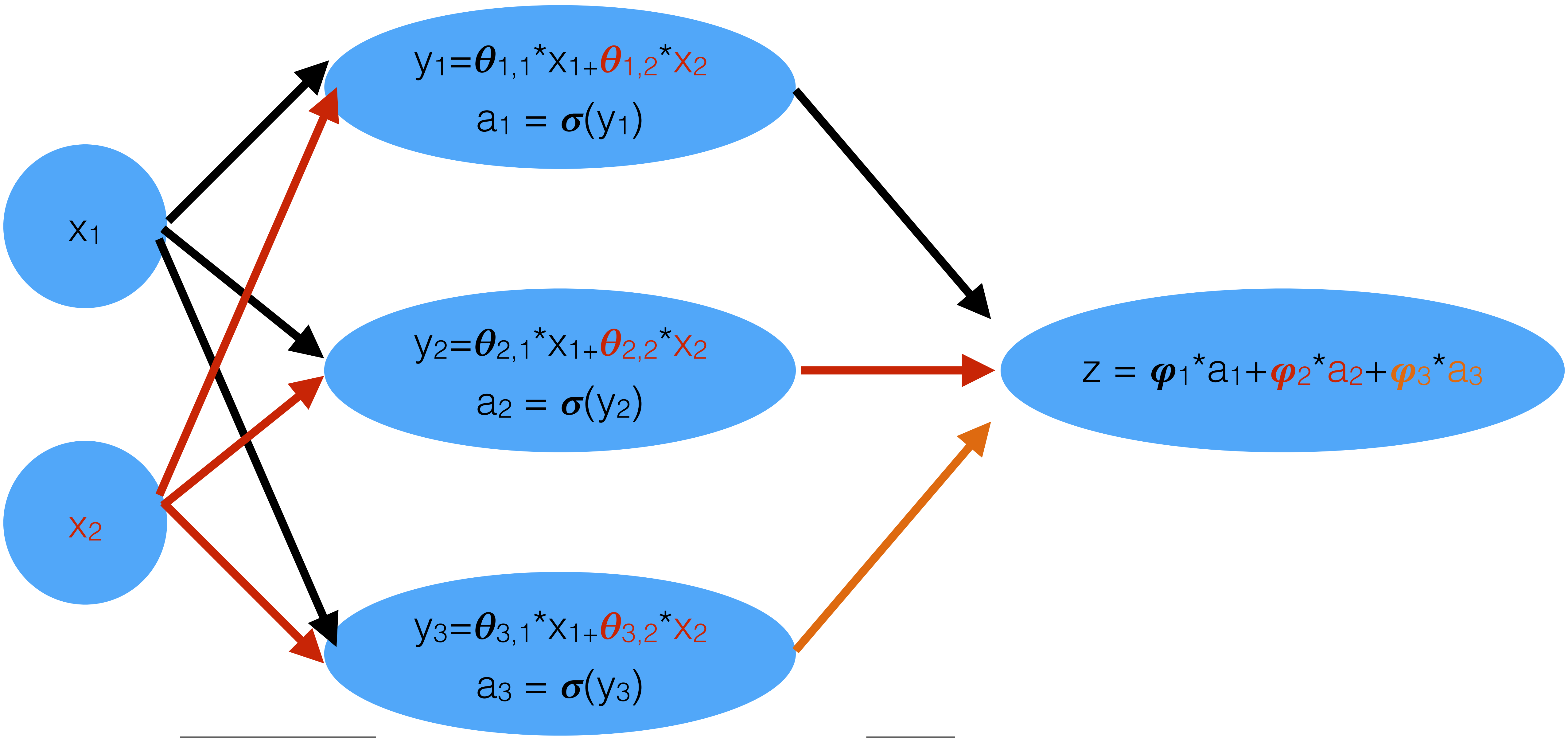
Back-propagation and Parallel Implementation

Overview

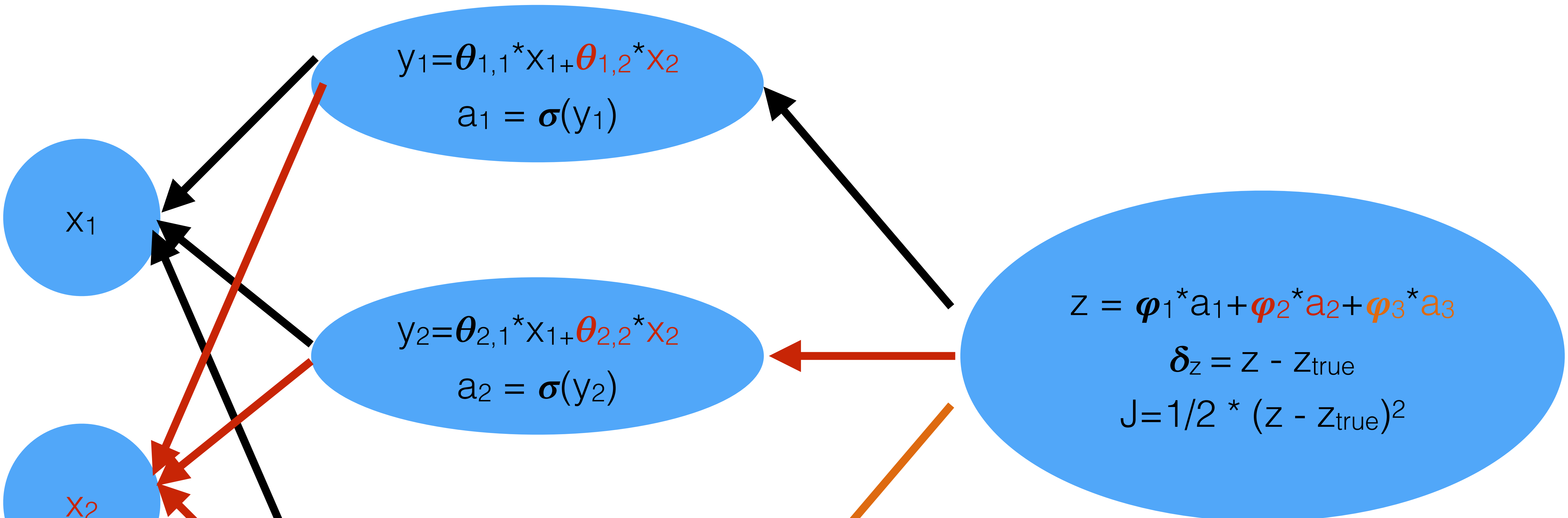
- Back-propagation review
- MPI Implementation
- Parameter Server Implementation

Back-propagation Review



$$\theta = \begin{bmatrix} \theta_{1,1} & \theta_{1,2} \\ \theta_{2,1} & \theta_{2,2} \\ \theta_{3,1} & \theta_{3,2} \end{bmatrix} \qquad \varphi = \begin{bmatrix} \varphi_1 \\ \varphi_2 \\ \varphi_3 \end{bmatrix}$$

Back-propagation Review



$$\varphi_1 = \varphi_1 - \lambda^* \partial J/\partial \varphi_1$$

$$\partial J/\partial \varphi_1 = (z - z_{true}) * a_1 = \delta_z * a_1$$

$$\theta = \begin{bmatrix} \theta_{1,1} & \theta_{1,2} \\ \theta_{2,1} & \theta_{2,2} \\ \theta_{3,1} & \theta_{3,2} \end{bmatrix}$$

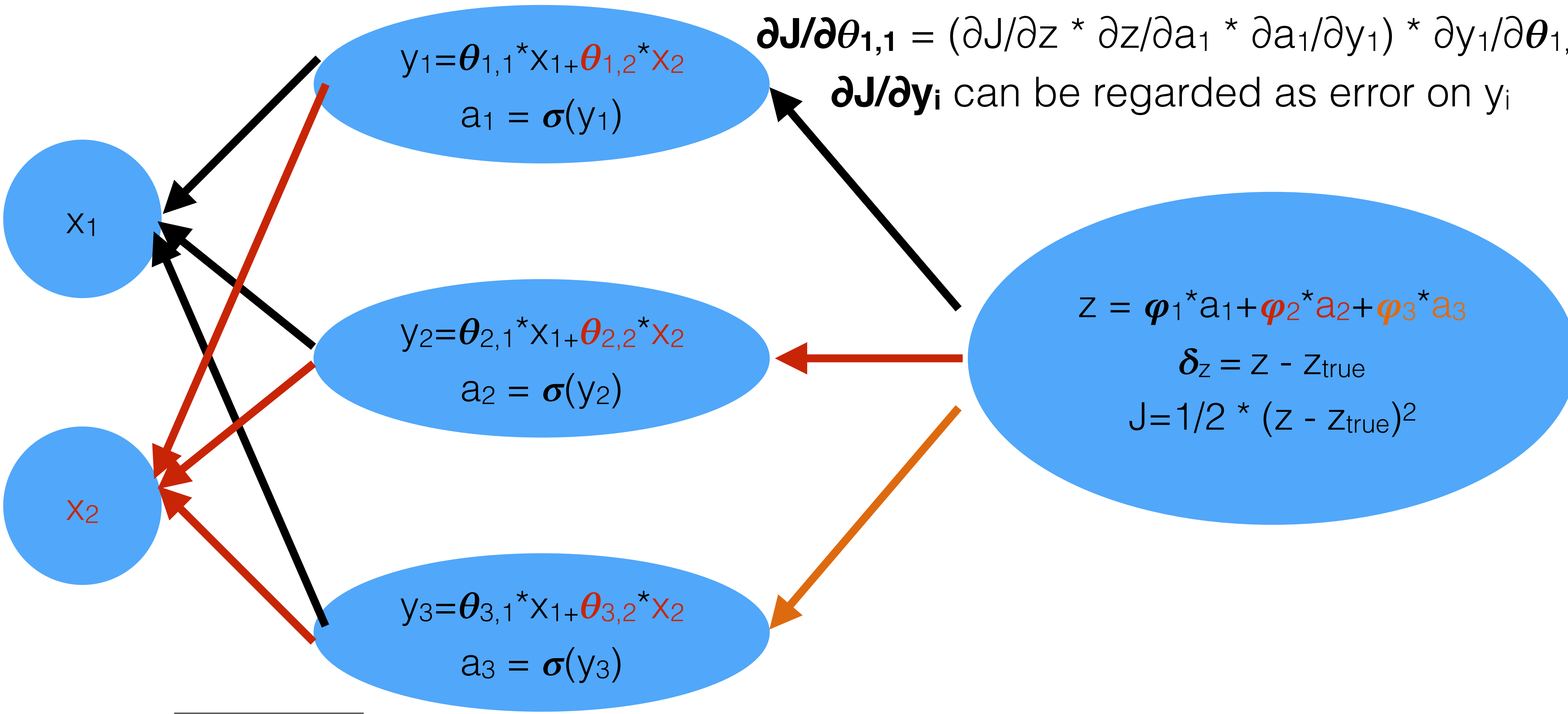
$$\varphi = \begin{bmatrix} \varphi_1 & \varphi_2 & \varphi_3 \end{bmatrix}$$

Back-propagation Review

$$\theta_{1,1} = \theta_{1,1} - \lambda^* \partial J/\partial\theta_{1,1}$$

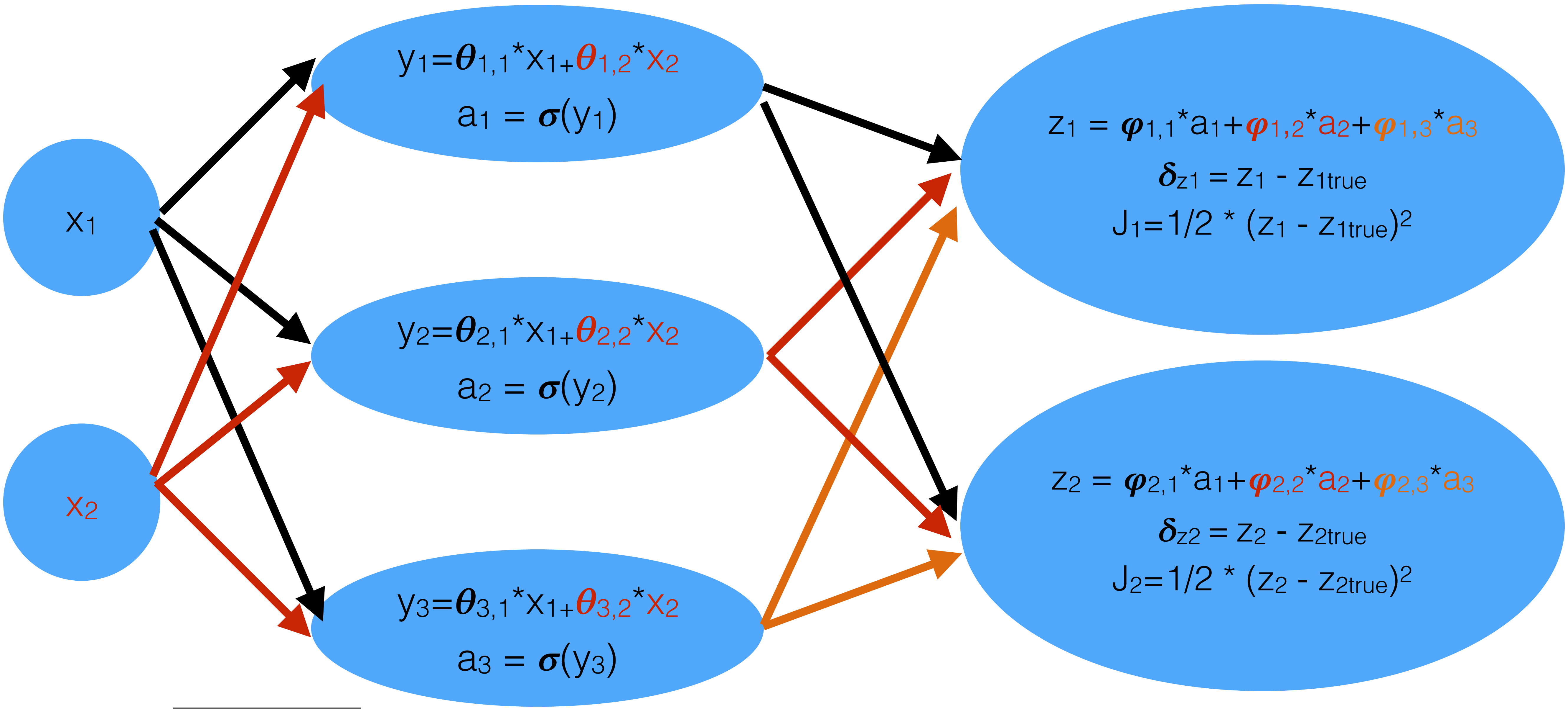
$$\partial J/\partial\theta_{1,1} = (\partial J/\partial z * \partial z/\partial a_1 * \partial a_1/\partial y_1) * \partial y_1/\partial\theta_{1,1}$$

$\partial J/\partial y_i$ can be regarded as error on y_i



$$\theta = \begin{bmatrix} \theta_{1,1} & \theta_{1,2} \\ \theta_{2,1} & \theta_{2,2} \\ \theta_{3,1} & \theta_{3,2} \end{bmatrix} \qquad \varphi = \begin{bmatrix} \varphi_1 & \varphi_2 & \varphi_3 \end{bmatrix}$$

Back-propagation Review



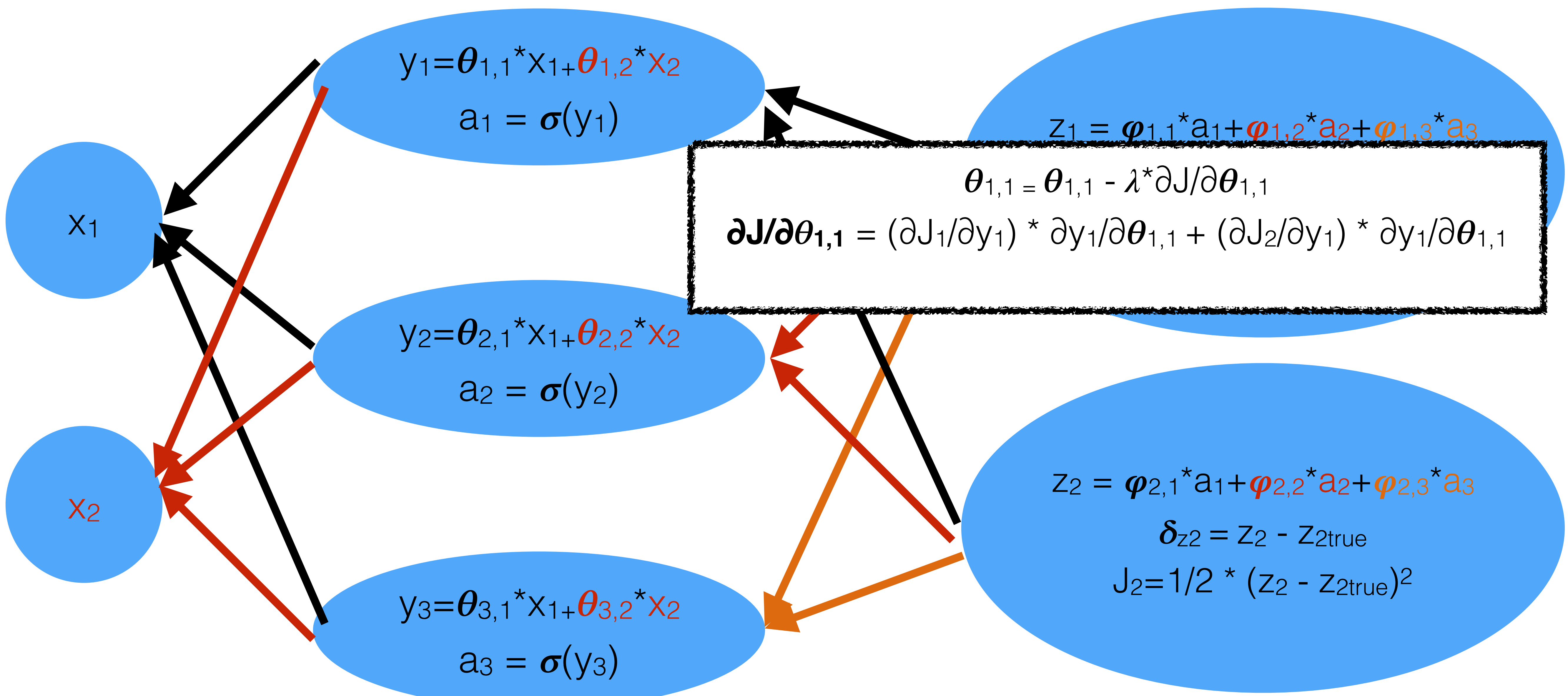
$\theta=$

$\theta_{1,1}$	$\theta_{1,2}$
$\theta_{2,1}$	$\theta_{2,2}$
$\theta_{3,1}$	$\theta_{3,2}$

$\varphi=$

$\varphi_{1,1}$	$\varphi_{1,2}$	$\varphi_{1,3}$
$\varphi_{2,1}$	$\varphi_{2,2}$	$\varphi_{2,3}$

Back-propagation Review



$\theta=$

$\theta_{1,1}$	$\theta_{1,2}$
$\theta_{2,1}$	$\theta_{2,2}$
$\theta_{3,1}$	$\theta_{3,2}$

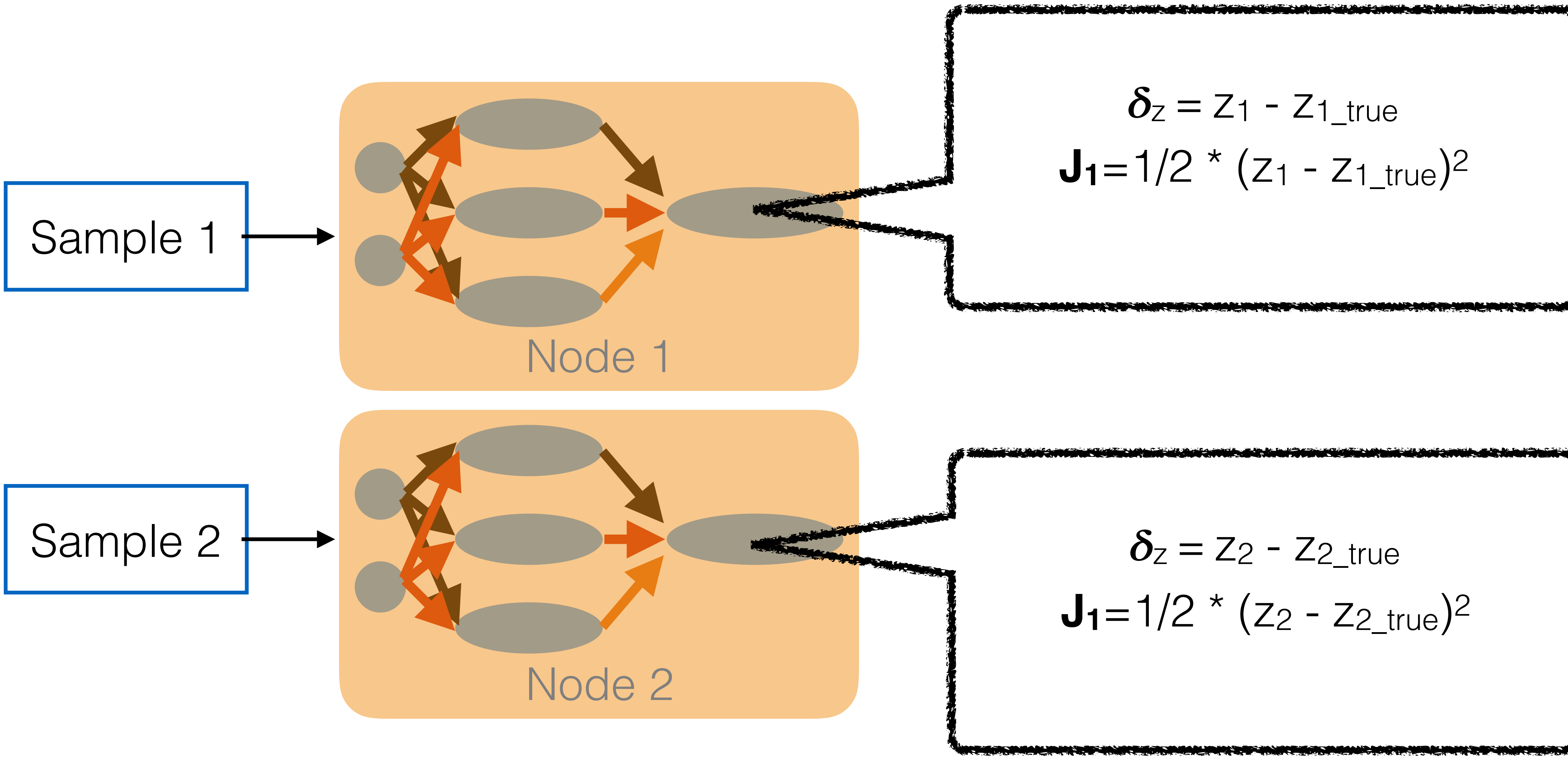
$\varphi=$

$\varphi_{1,1}$	$\varphi_{1,2}$	$\varphi_{1,3}$
$\varphi_{2,1}$	$\varphi_{2,2}$	$\varphi_{2,3}$

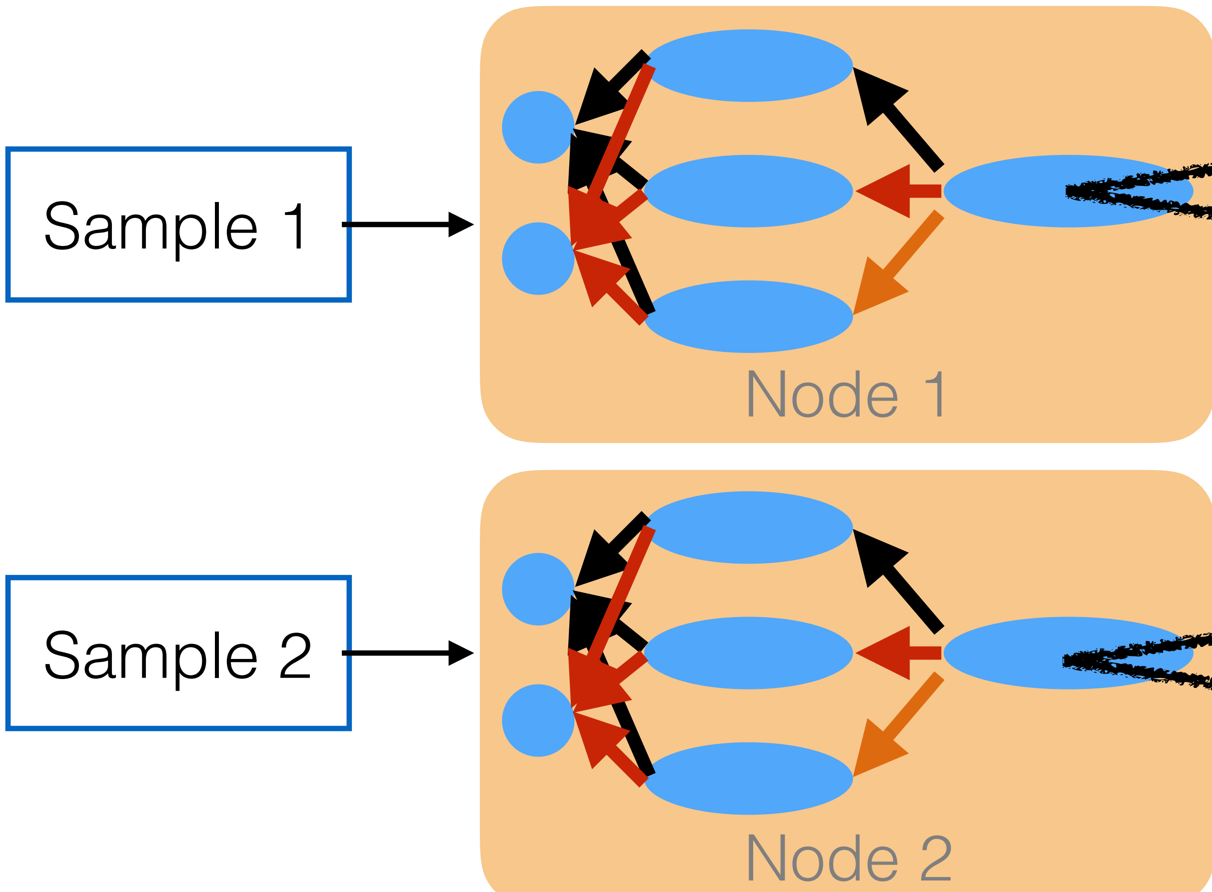
Intel MLSL Implementation

- MPI based library MLSL
- Supports Caffe, Theano
- Data parallelism based
- It is as simple as an MPI_allreduce between forward propagation and back propagation

Intel MLSL Implementation



Intel MLSL Implementation



$$\delta_{z1} = z_1 - z_{1\_true}$$

$$J_1 = 1/2 * (z_1 - z_{1\_true})^2$$

$$\partial J_1 / \partial \varphi_1 = (z_1 - z_{1\_true}) * a_1 = \delta_z * a_1$$

$$\text{sum\_grad} = \text{MPI\_allreduce}(\delta_z, +)$$

$$\varphi_1 = \varphi_1 - \lambda^* \delta_z * a_1$$

$$\delta_{z2} = z_2 - z_{2\_true}$$

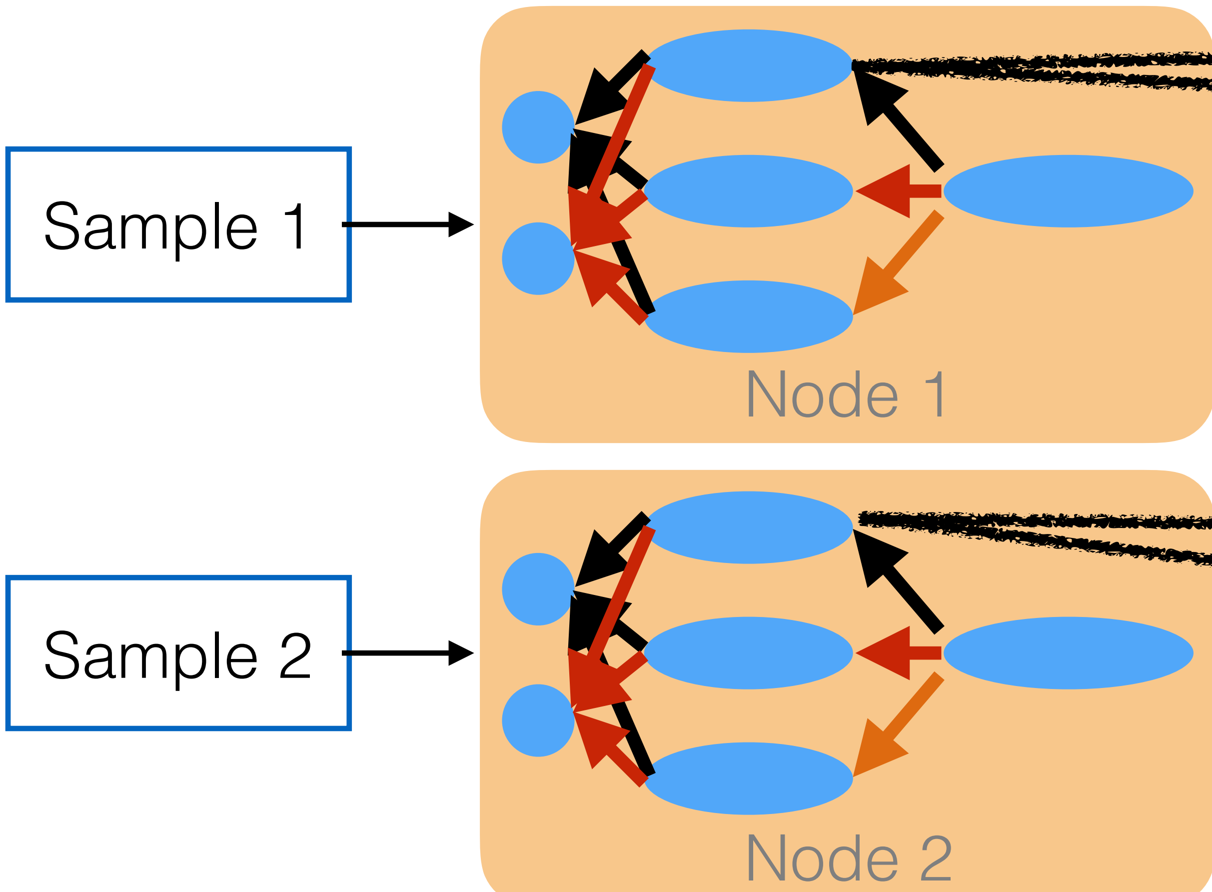
$$J_1 = 1/2 * (z_2 - z_{2\_true})^2$$

$$\partial J_2 / \partial \varphi_1 = (z_2 - z_{2\_true}) * a_1 = \delta_z * a_1$$

$$\text{sum\_grad} = \text{MPI\_allreduce}(\delta_z, +)$$

$$\varphi_1 = \varphi_1 - \lambda^* \delta_z * a_1$$

Intel MLSL Implementation



$$\frac{\partial J}{\partial \theta_{1,1}} = \partial J/\partial z * \partial z/\partial a_1 * \partial a_1/\partial y_1 * \partial y_1/\partial \theta_{1,1}$$

$$\theta_{1,1} = \theta_{1,1} - \lambda * \partial J/\partial \theta_{1,1}$$

Identical

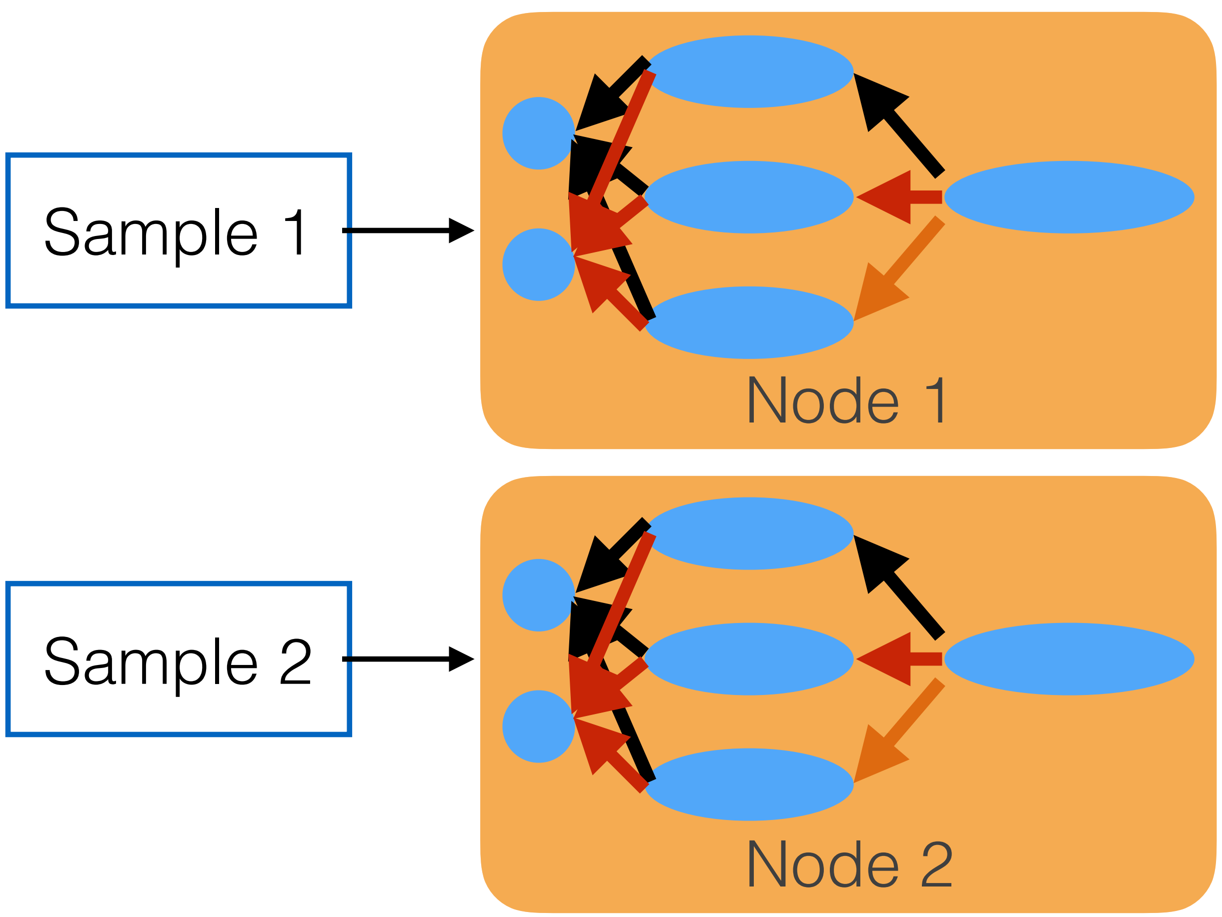
$$\frac{\partial J}{\partial \theta_{1,1}} = \partial J/\partial z * \partial z/\partial a_1 * \partial a_1/\partial y_1 * \partial y_1/\partial \theta_{1,1}$$

$$\theta_{1,1} = \theta_{1,1} - \lambda * \partial J/\partial \theta_{1,1}$$

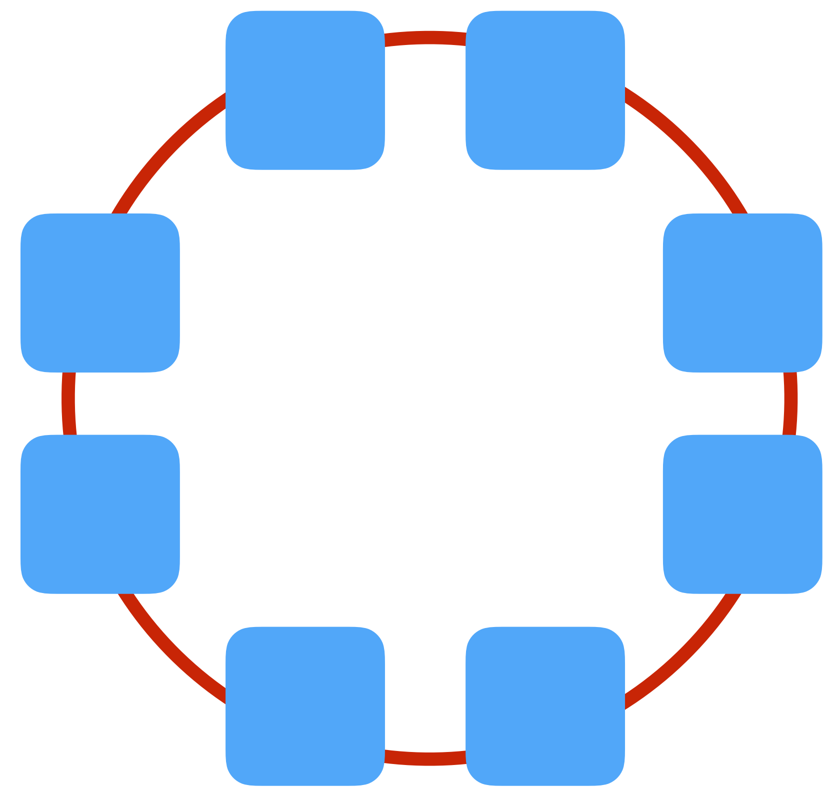
Parameter Server

- Uses a separate set of nodes as key-value stores
- Data parallelism based
- Training workers synchronize via push and pull operations
- Key innovation: consistency model

Parameter Server

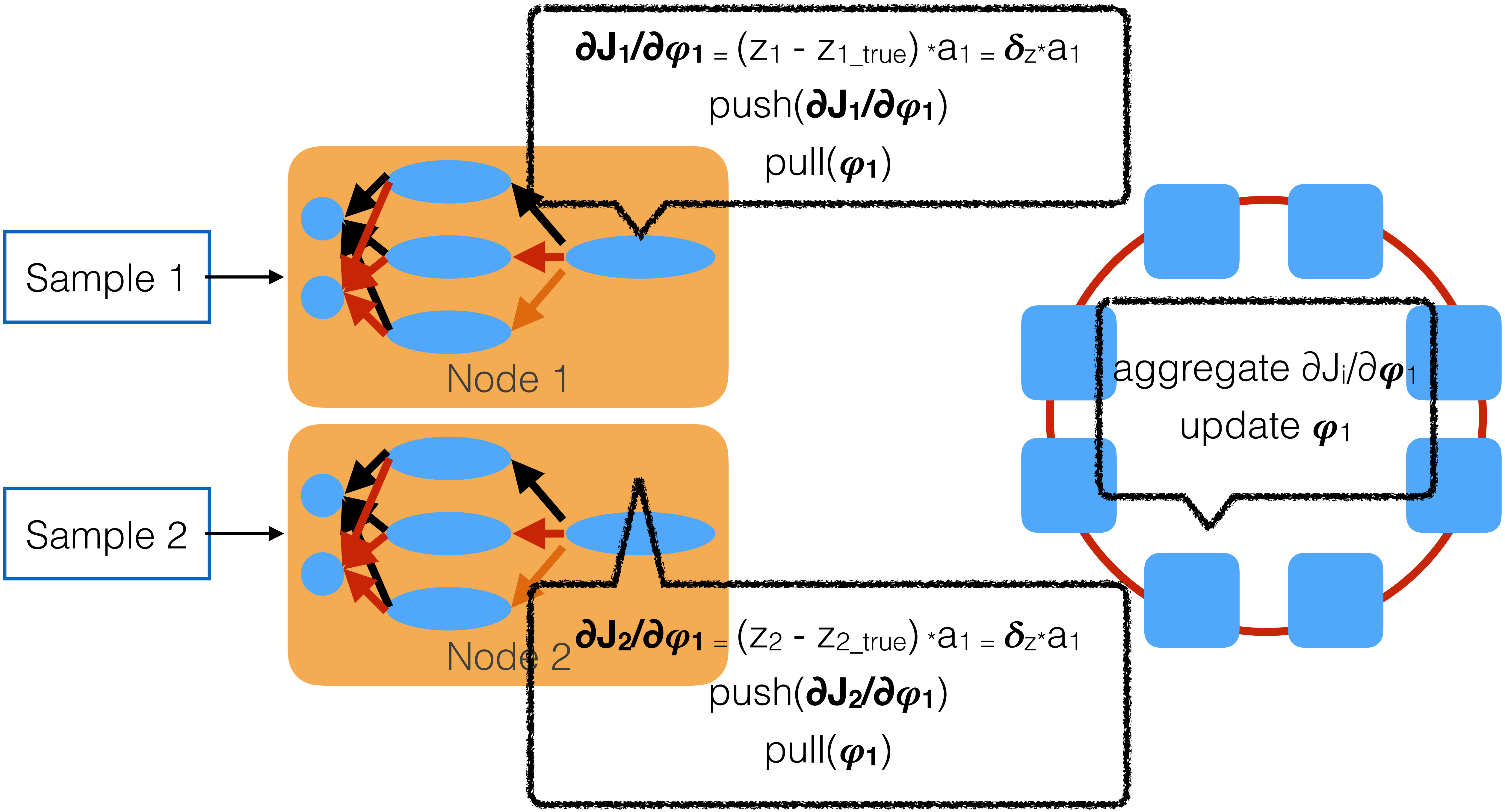


Key-Value pairs:
(parameter_name, value)



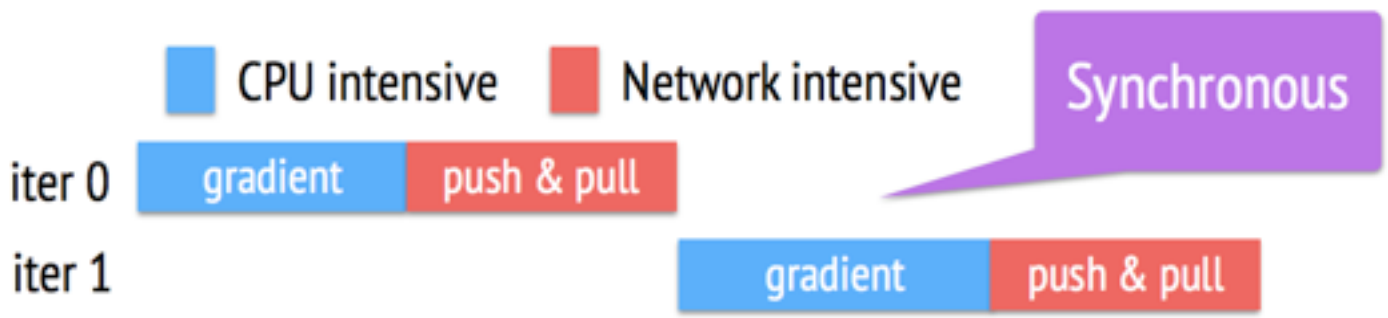
Parameter Server

Parameter Server

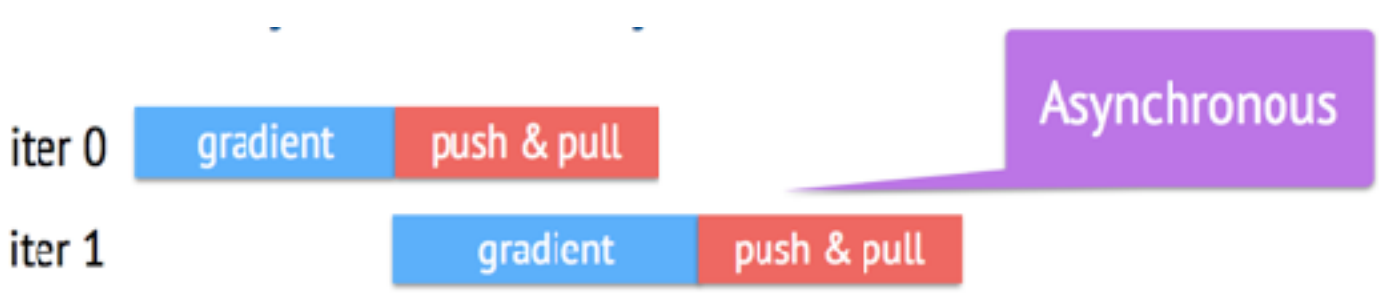


Task

- a push/pull/user defined function (an iteration)
- “execute-after-finished” dependency

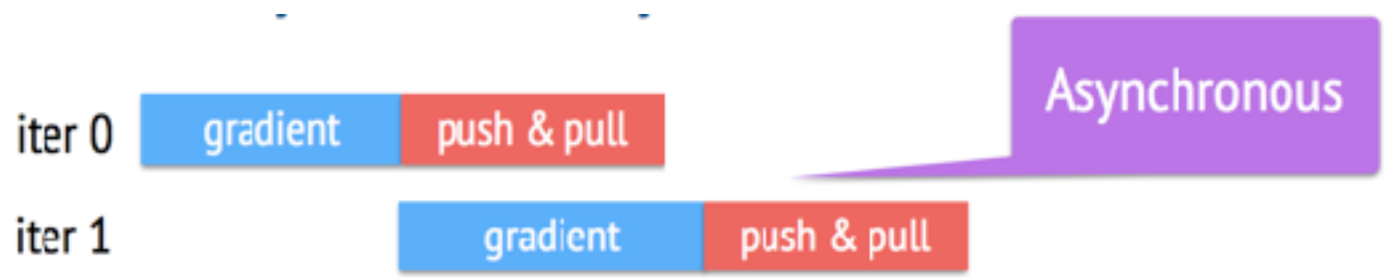


- executed asynchronously



Flexible Iteration Dependency

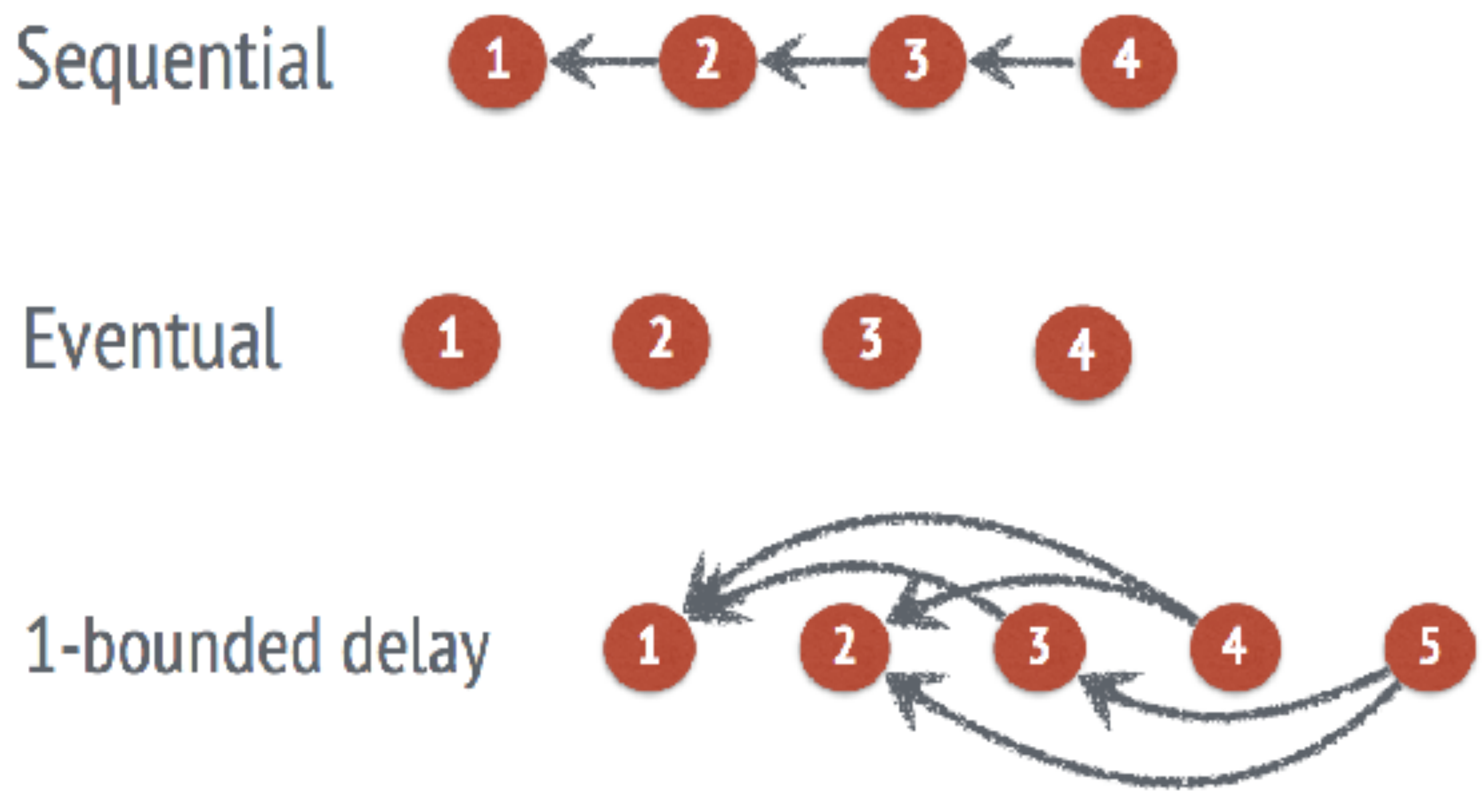
- Executed asynchronously



- Iteration 1 uses the old parameters as in Iteration 0, and obtains the same gradient.
- It is likely to slow down the convergence progress
- Some algorithms are less sensitive to this type of inconsistency

Flexible Iteration Dependency

- Sequential
- Eventual
- Bounded Delay



  - τ: maximal delay time. A new iteration will be blocked until all previous tasks τ time ago have been finished.
  - $\tau = 0$ —> Sequential, $\tau = \infty$ —> Eventual

Flexible Iteration Dependency

- Bounded Delay consistency model is referred as Stale Synchronous Parallel (SSP) consistency model.

Practical Scalability

- MPI based solution
  - Intel MLSL — measuring now

- Parameter Server based solution
  - Poseidon — 8 GPU nodes 4-4.5x speedup
  - Common ML algorithm — 5000 workers and 1000 parameter servers