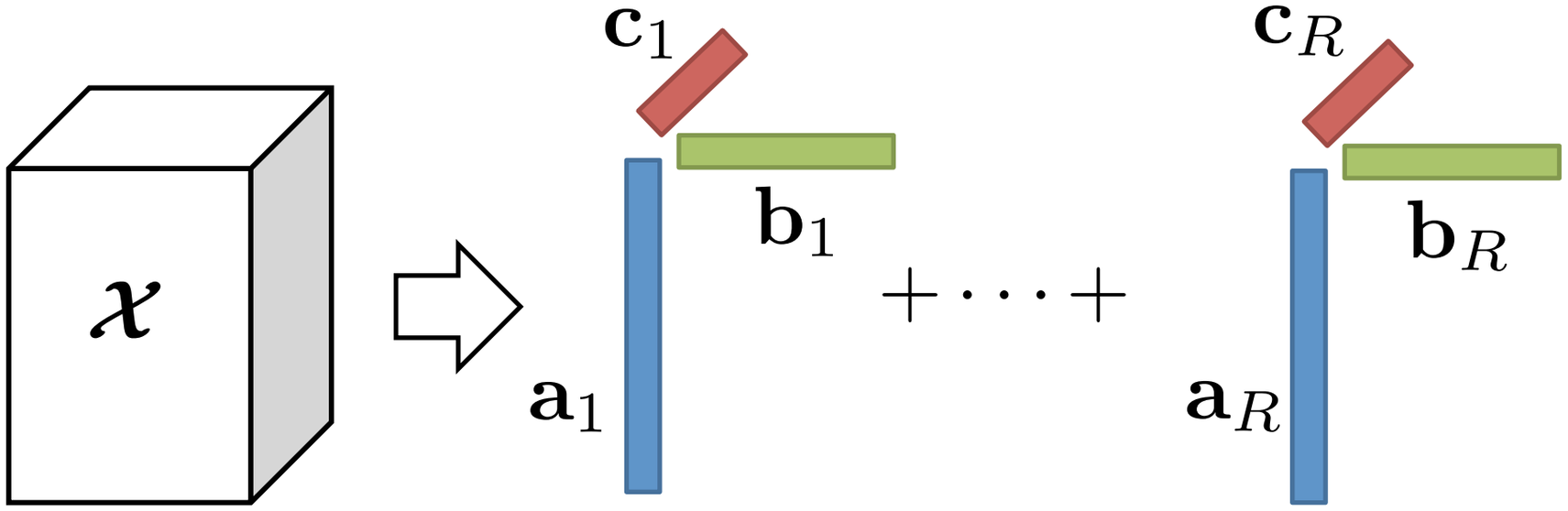
FlexiFaCT: Scalable Flexible Factorization of Coupled Tensors on Hadoop

CS 584: Big Data Analytics

Material adapted from ICASSP 2014 Tutorial
by Nicholas Sidiropoulos & Vagelis Papelxakis
(http://www.cs.cmu.edu/~epapalex/tutorials/icassp14.html)

CP decomposition



$$\min_{\mathbf{A},\mathbf{B},\mathbf{C}} ||\mathcal{X} - \sum_k \mathbf{a}_k \circ \mathbf{b}_k \circ \mathbf{c}_k||_F^2$$

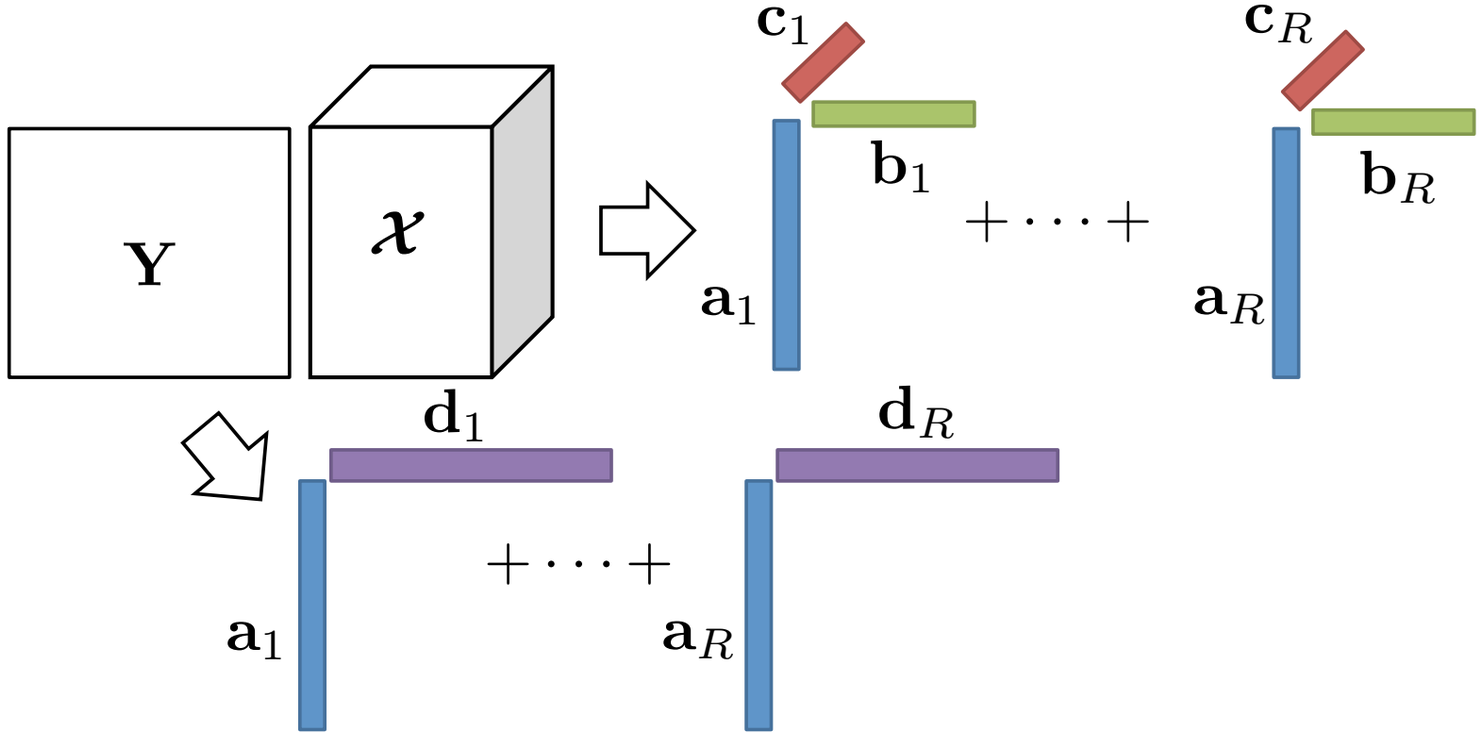
Motivation for Coupled Tensors

- Not all data can be captured in single tensor
  - Social network about who messages whom, who becomes friends with whom, and when
    - How can you incorporate side information like demographic information?
- Not all data needs to be stored as a tensor either, some might be matrices

Coupled Matrix-Tensor Factorization (CMTF)

Shared mode between matrix and tensor with same low-rank basis

$$\min ||\boldsymbol{\mathcal{X}} - [\![\mathbf{A}; \mathbf{B}; \mathbf{C}]\!]||_F^2 + ||\mathbf{Y} - \mathbf{A}\mathbf{D}^\intercal||_F^2$$



Existing CMTF Algorithms

- Traditional (Single Processor)
  - Alternating Least Squares
  - Gradient Descent (Acar, Kolda, and Dunlavy, 2011)
- Large-scale
  - Turbo-SMT (Papalexakis et al., 2014) - sampling based

FlexiFaCT Optimization Objective

- Frobenius norm

$$||\mathcal{X} - \sum_k \mathbf{u}_k \circ \mathbf{v}_k \circ \mathbf{w}_k||_F^2 + ||\mathbf{Y} - \mathbf{U}\mathbf{A}^\top||_F^2$$

- Frobenius + sparsity

$$||\mathcal{X} - \sum_k \mathbf{u}_k \circ \mathbf{v}_k \circ \mathbf{w}_k||_F^2 + ||\mathbf{Y} - \mathbf{U}\mathbf{A}^\top||_F^2 +$$
$$\lambda(||\mathbf{U}||_1 + ||\mathbf{V}||_1 + ||\mathbf{W}||_1 + ||\mathbf{A}||_1)$$

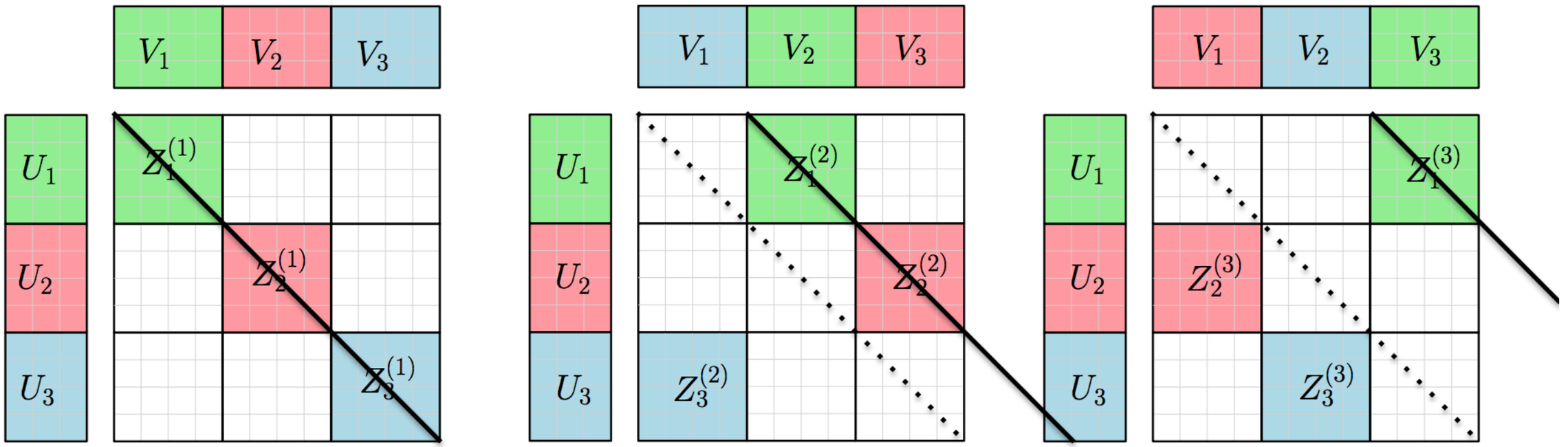
- Frobenius + sparsity + nonnegative

$$||\mathcal{X} - \sum_k \mathbf{u}_k \circ \mathbf{v}_k \circ \mathbf{w}_k||_F^2 + ||\mathbf{Y} - \mathbf{U}\mathbf{A}^\top||_F^2 +$$
$$\lambda(||\mathbf{U}||_1 + ||\mathbf{V}||_1 + ||\mathbf{W}||_1 + ||\mathbf{A}||_1)$$
$$\text{s.t. } \mathbf{U}, \mathbf{V}, \mathbf{W}, \text{ or } \mathbf{A} \geq 0$$

FlexiFaCT Algorithm Overview

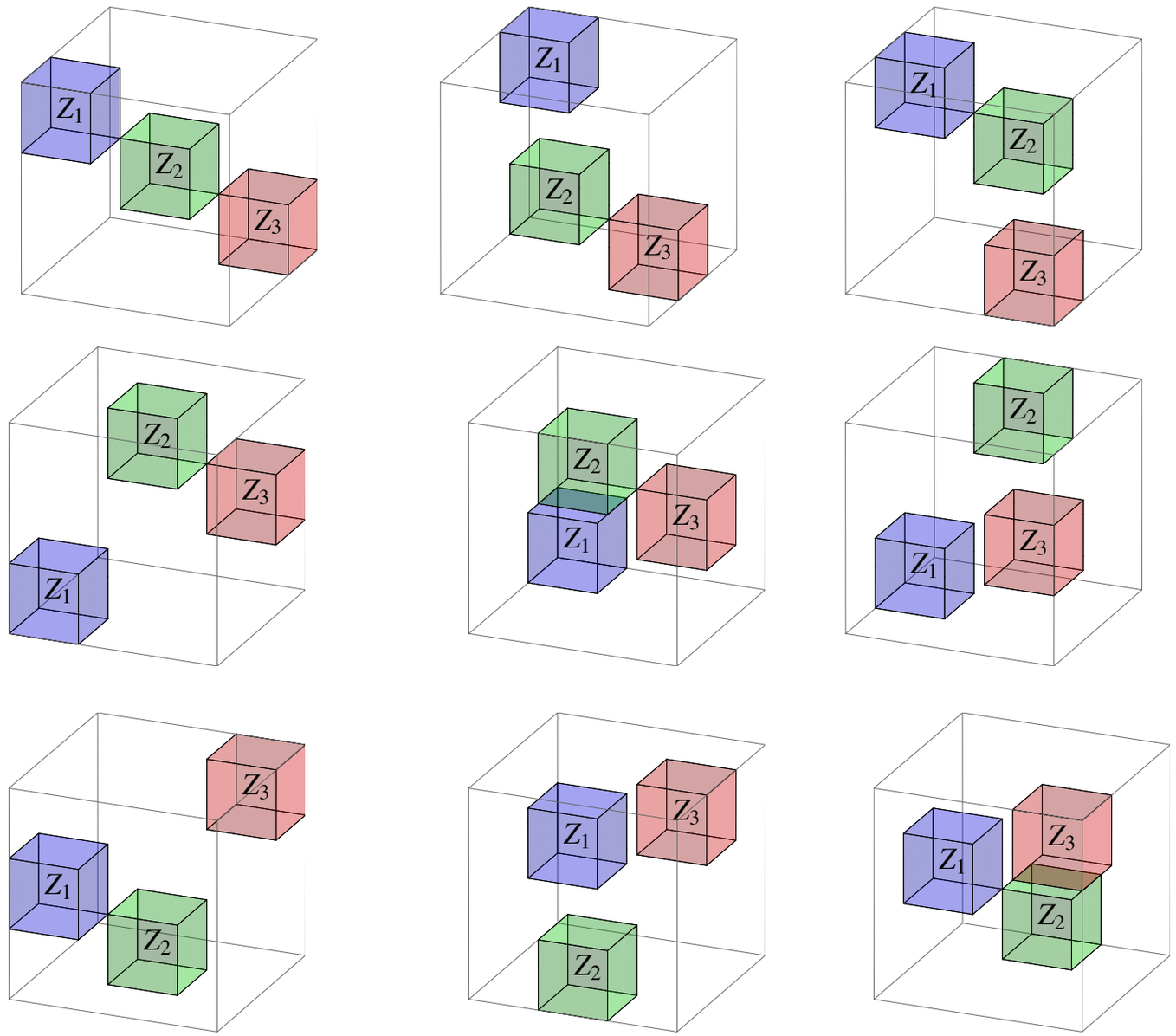
- Solve for parameters using SGD
  - For the case where there is sparsity and/or non-negative use projection to solve each iteration
- Use blocks to parallelize SGD
- Implement the algorithm using MapReduce framework to achieve scalability

DSGD for MF (Gemulla, 2011)



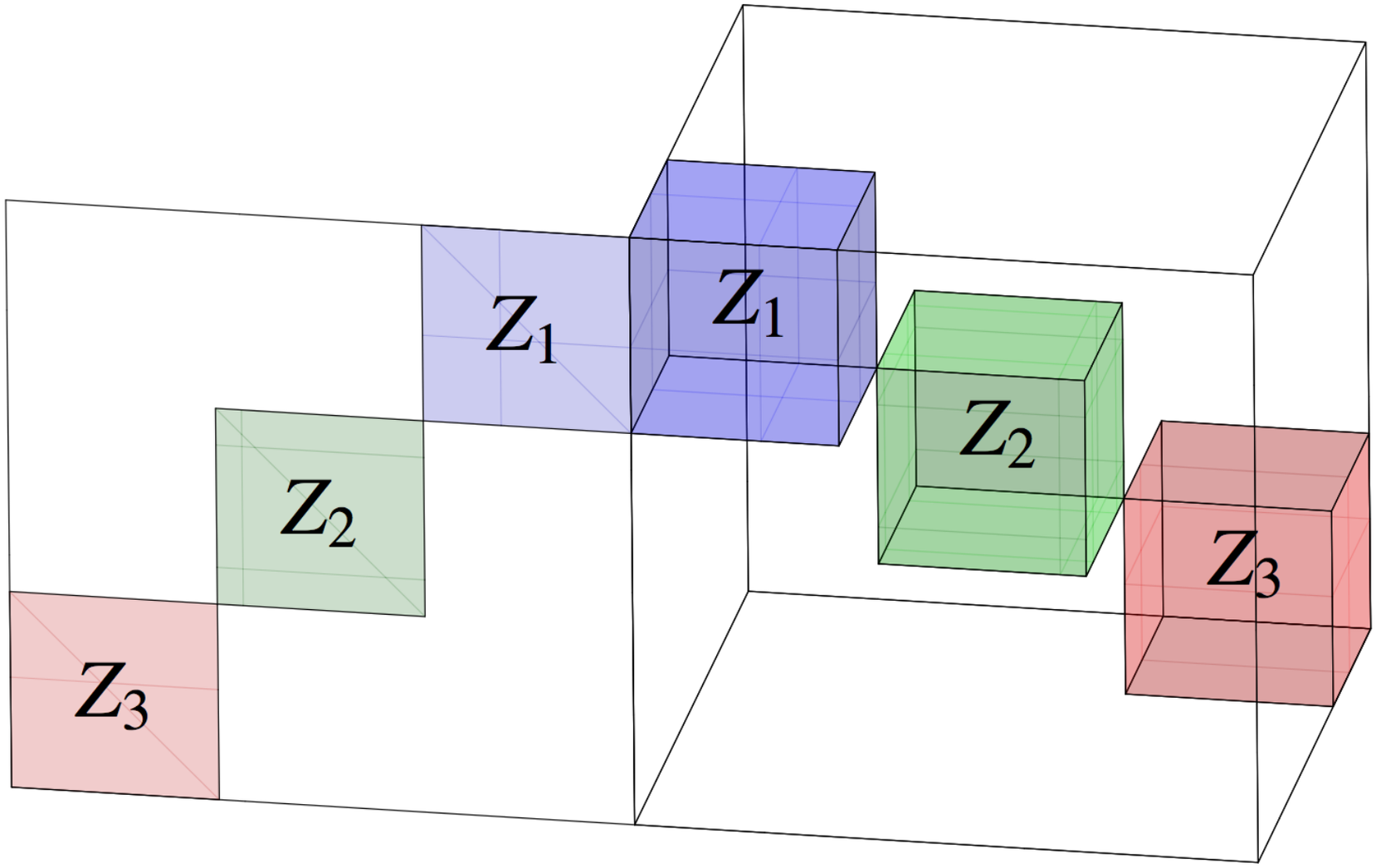
- Partition data into d x d blocks
- Process each block in stratum in parallel
- Process strata sequentially

Finding Blocks for TF



For d = 3 blocks per stratum, need d^2 strata

Blocks for CMTF



MapReduce Framework

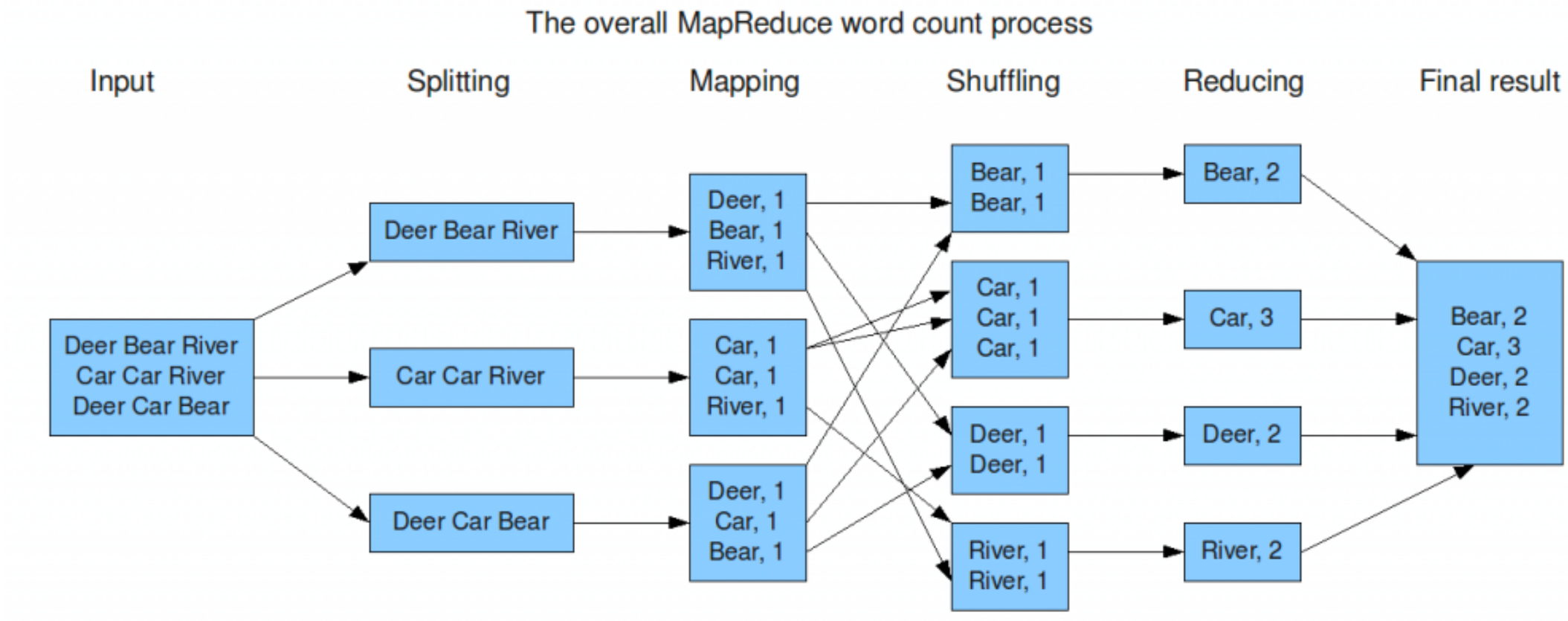
- Distributed file system (GFS - Google File System) where files are stored in the cloud

- Everything is viewed as <key, value> pairs

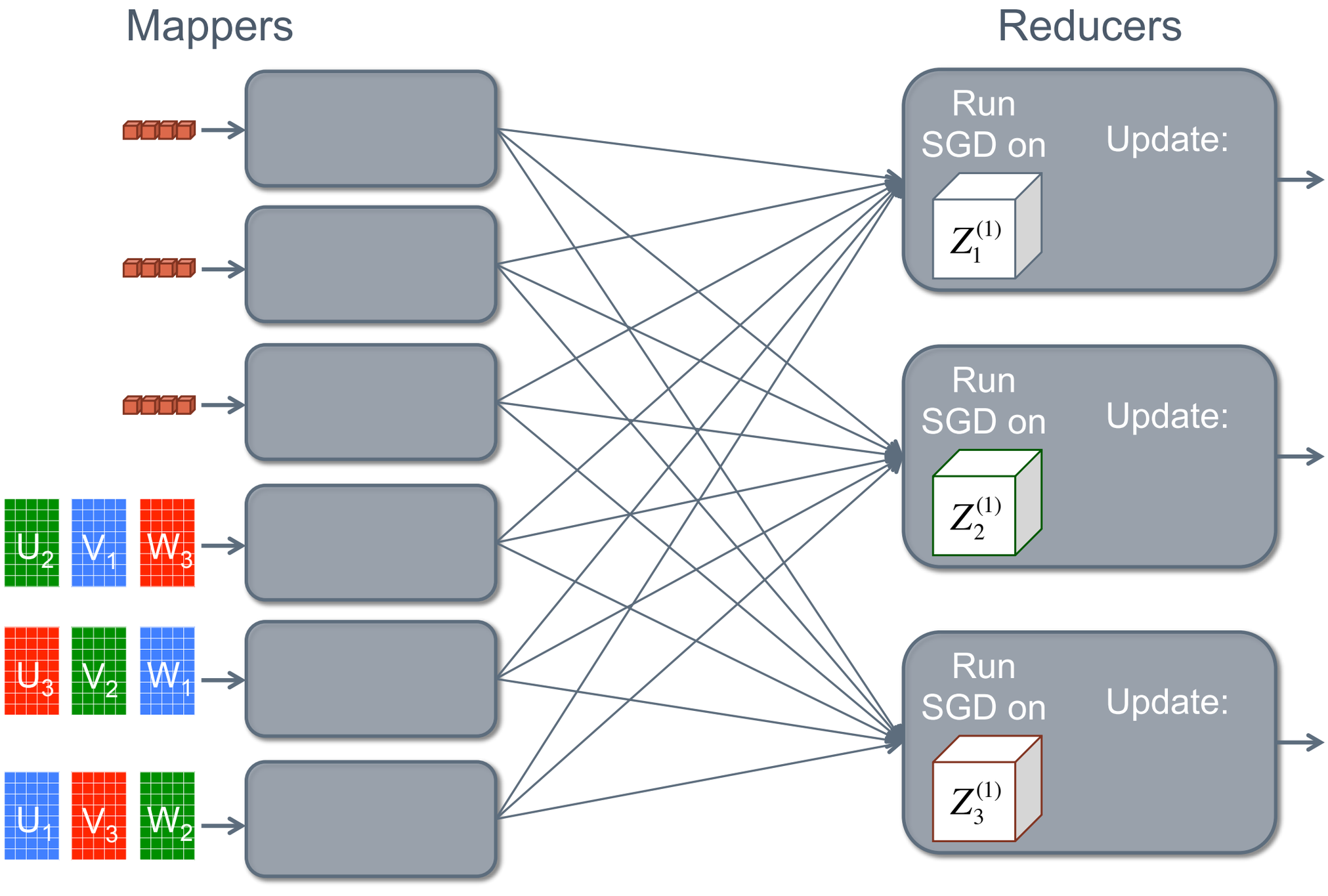
- Provides a Map() function - gathers data records with same key to one worker machine

- Provides a Reduce() function - tells system how to combine values of all records with same key

Example: Hadoop/ MapReduce



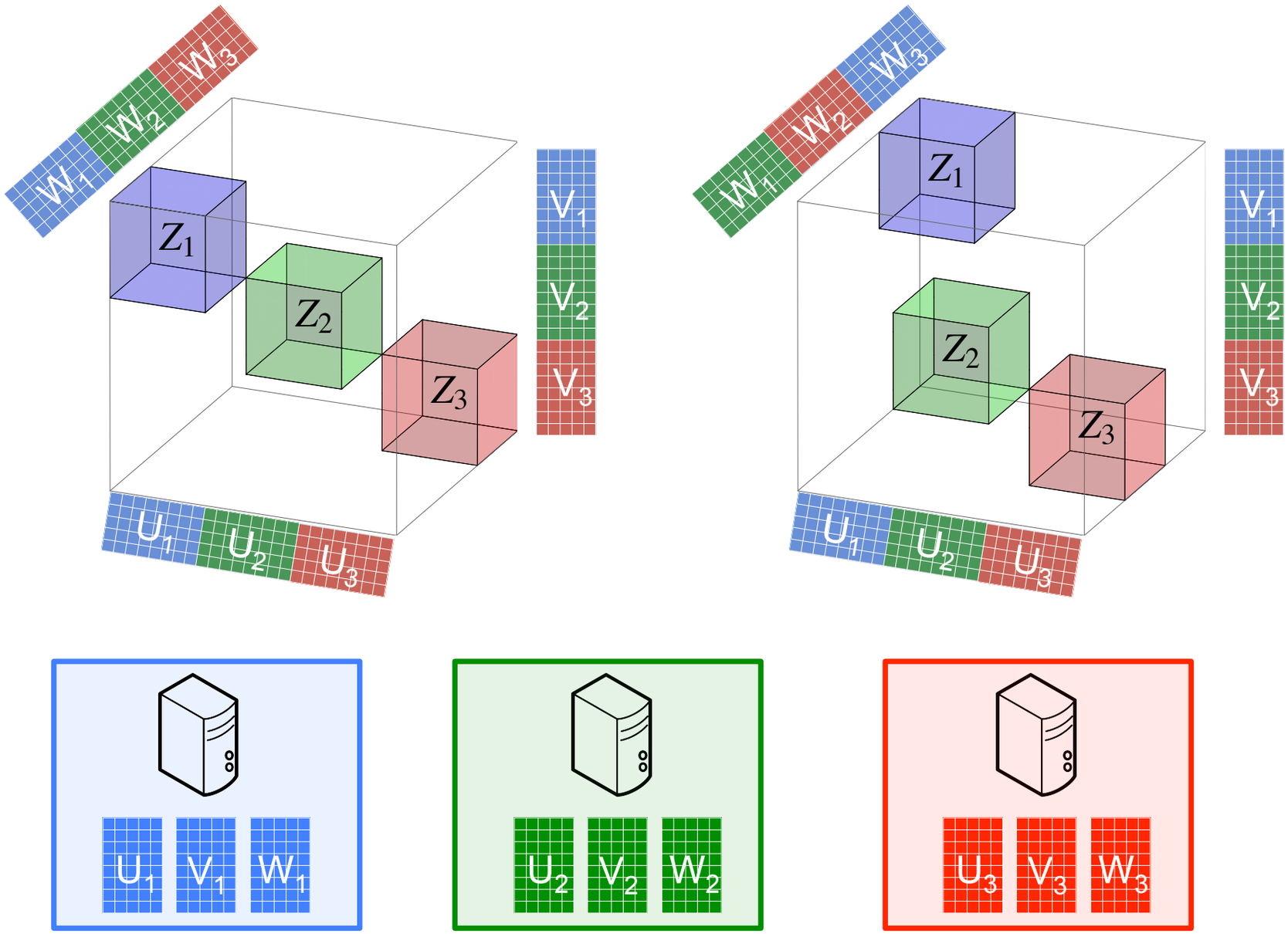
Bad Hadoop Algorithm



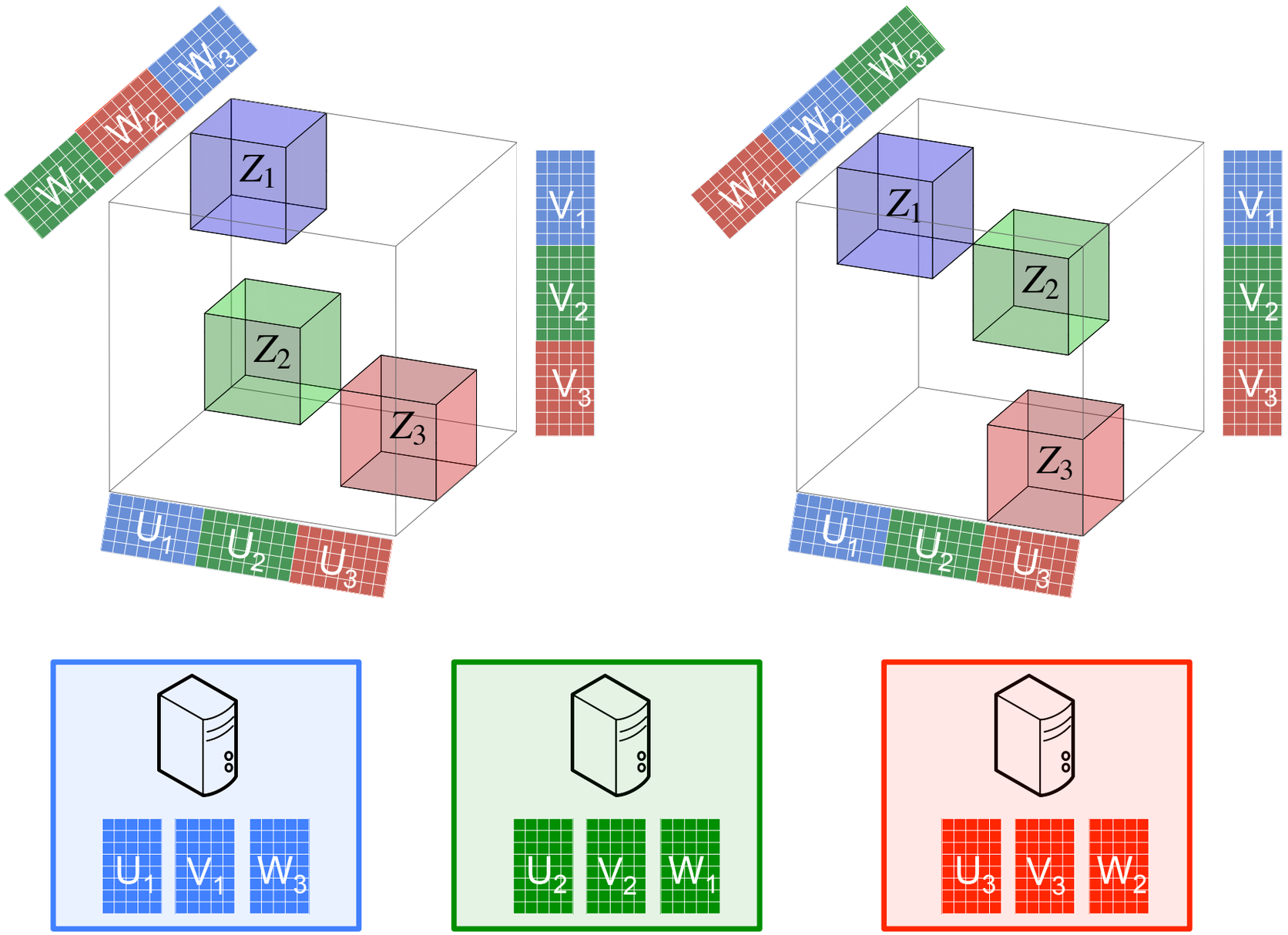
Hadoop Challenges

- MapReduce is typically very bad for iterative algorithms
  - T x d^2 iterations
- Sizable overhead per Hadoop job
- Little flexibility

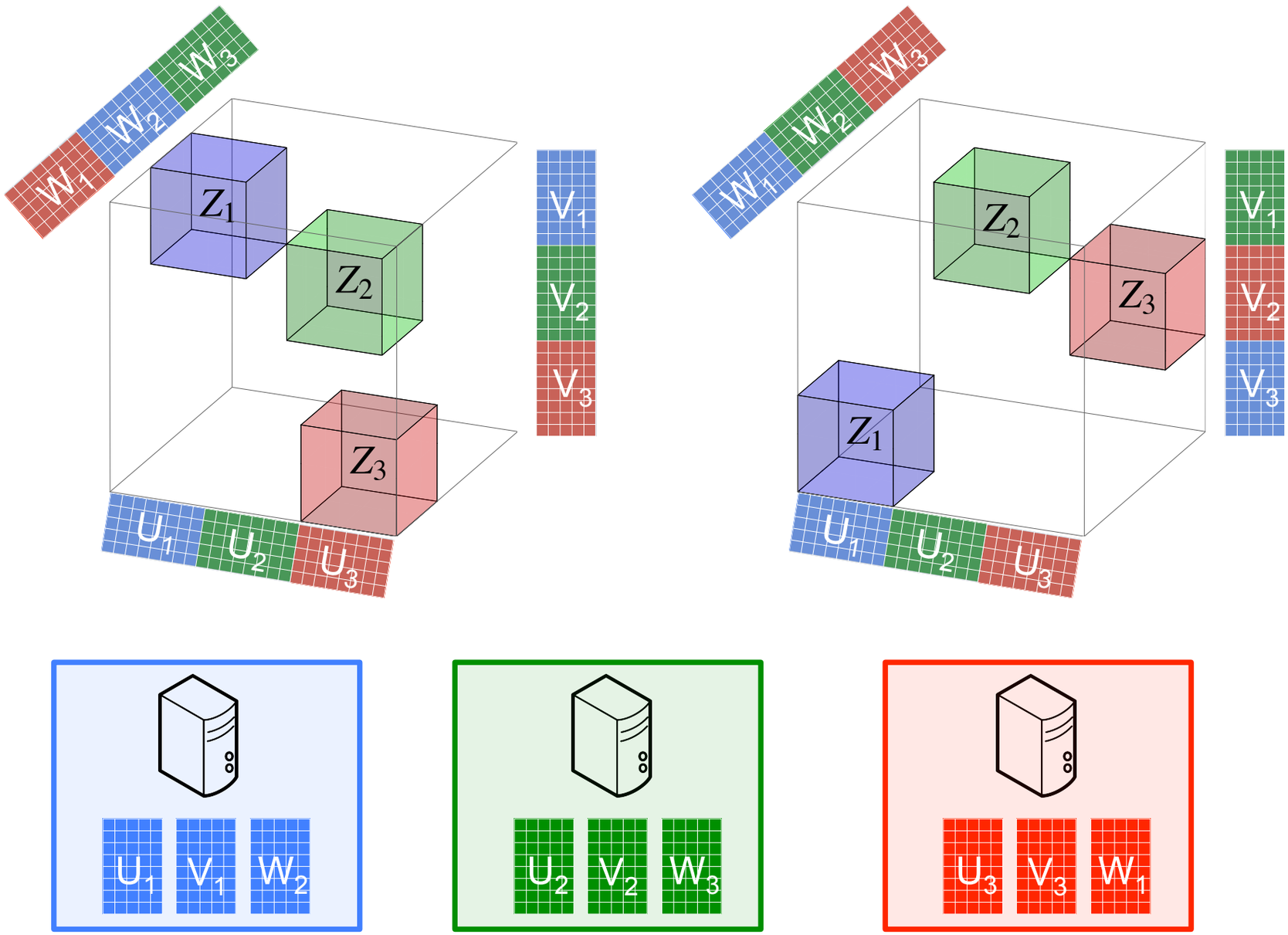
FlexiFaCT Algorithm using Blocks



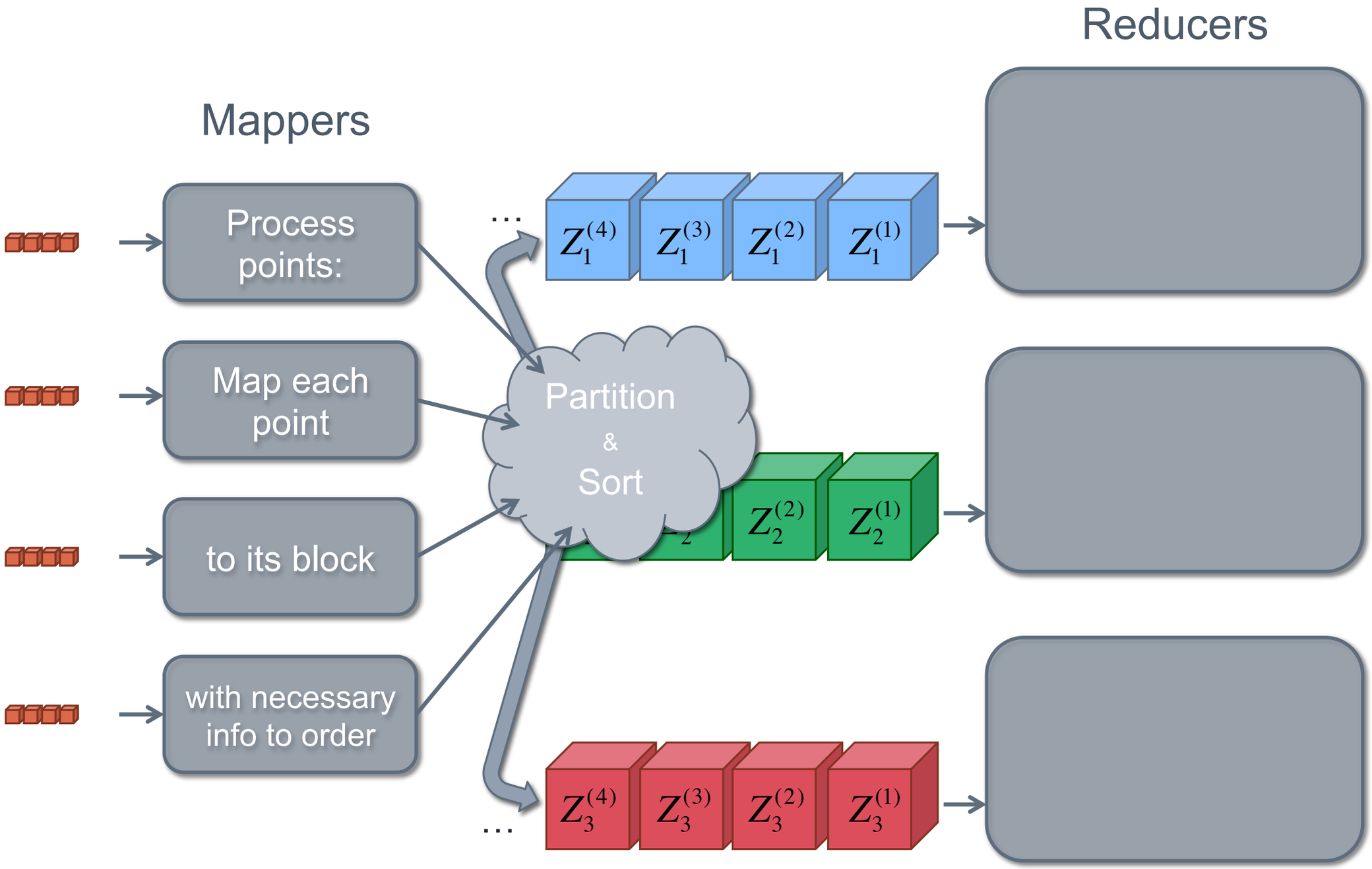
FlexiFaCT Algorithm using Blocks (2)



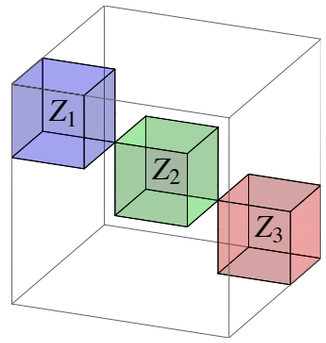
FlexiFaCT Algorithm using Blocks (3)



FlexiFaCT in Hadoop



FlexiFaCT in Hadoop (2)



Reducers

Mappers

Process points:

Map each point

to its block

with necessary info to order

Partition & Sort

$Z_1^{(5)}$ $Z_1^{(4)}$ $Z_1^{(3)}$ $Z_1^{(2)}$ $Z_1^{(1)}$

Update: U_1 V_1 W_1

$Z_2^{(3)}$ $Z_2^{(2)}$ $Z_2^{(1)}$

Run SGD on

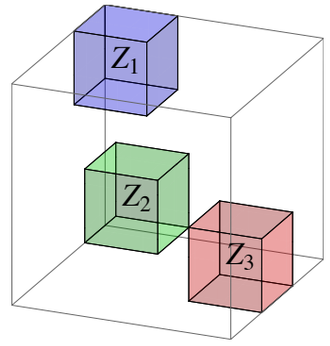
Update: U_2 V_2 W_2

$Z_3^{(5)}$ $Z_3^{(4)}$ $Z_3^{(3)}$ $Z_3^{(2)}$ $Z_3^{(1)}$

Run SGD on

Update: U_3 V_3 W_3

FlexiFaCT in Hadoop (3)



Mappers

Process points:

Map each point

to its block

with necessary info to order

Partition & Sort

$Z_1^{(5)}$ $Z_1^{(4)}$ $Z_1^{(3)}$

$Z_2^{(3)}$

$Z_3^{(5)}$ $Z_3^{(4)}$ $Z_3^{(3)}$

Reducers

Run SGD on $Z_1^{(2)}$ Update: U_1 V_1 W_1

HDFS

Run SGD on $Z_2^{(2)}$ Update: U_2 V_2 W_2

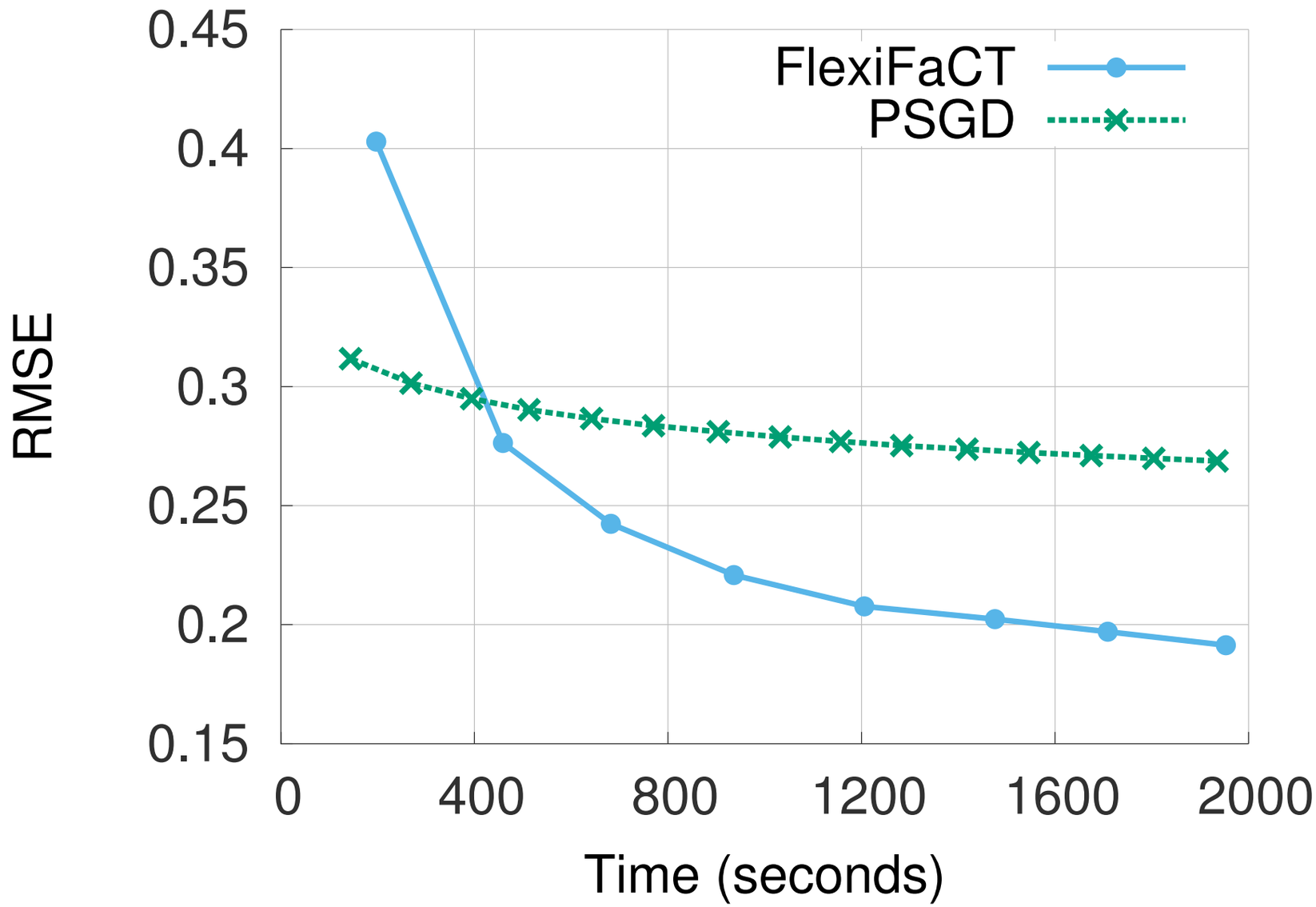
HDFS

Run SGD on $Z_3^{(2)}$ Update: U_3 V_3 W_3

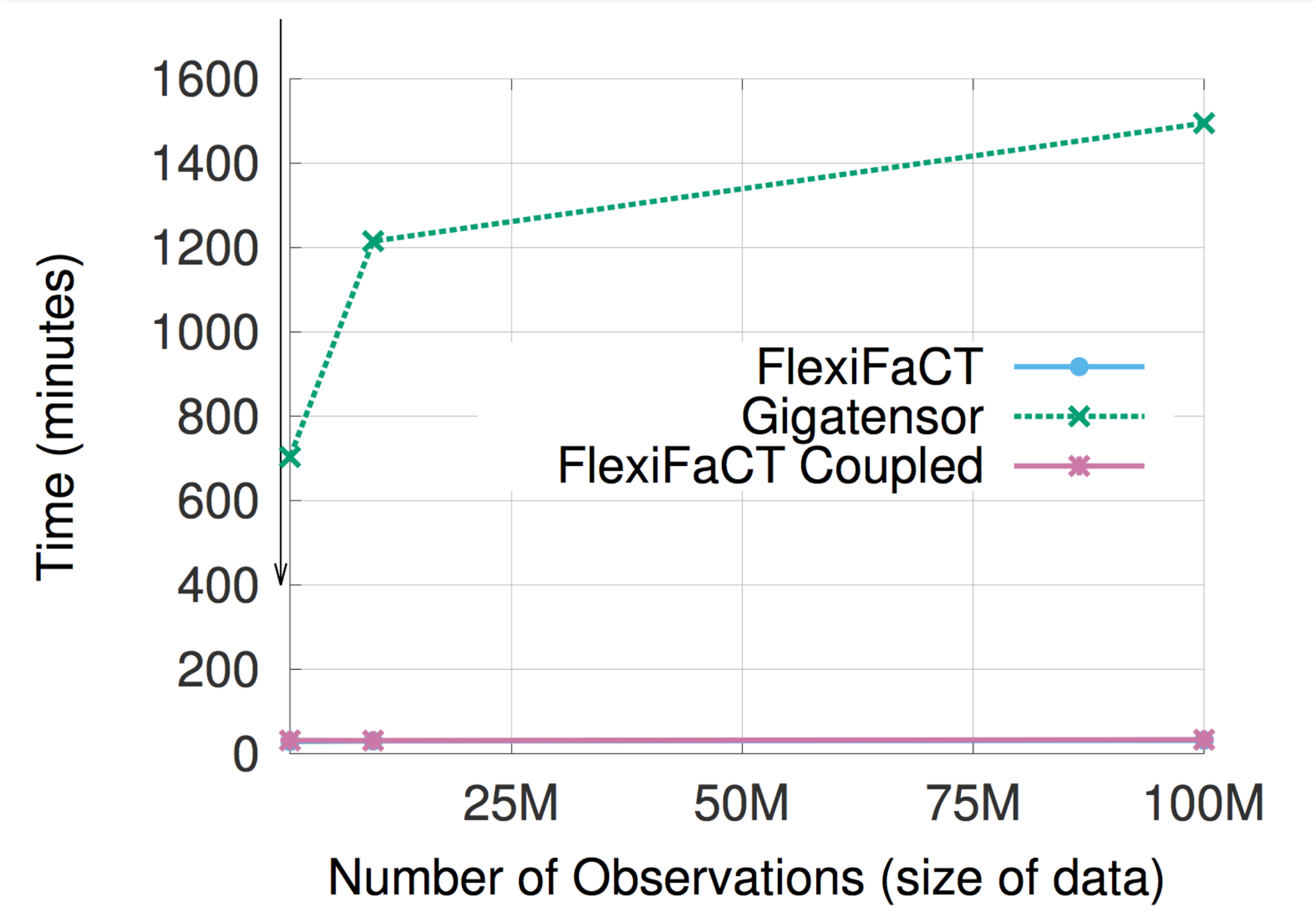
Experimental Evaluation

- Comparison with
  - PSGD (Zinkevich et al., 2010)
  - GigaTensor (Kang et al., 2012)
- Implementation in Hadoop 0.20.1
- 24 machines / reducers
- Synthetic data

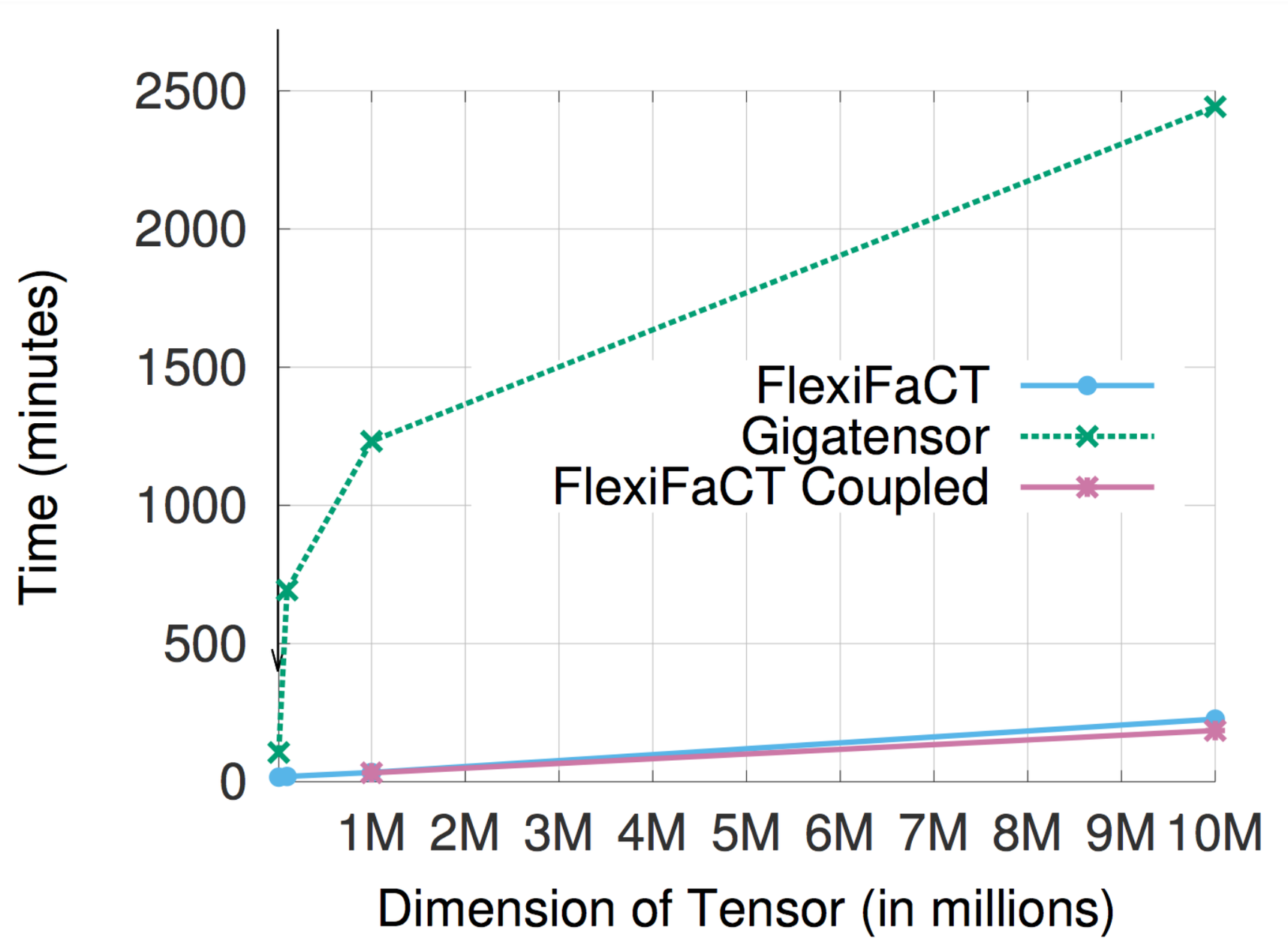
Results: Convergence



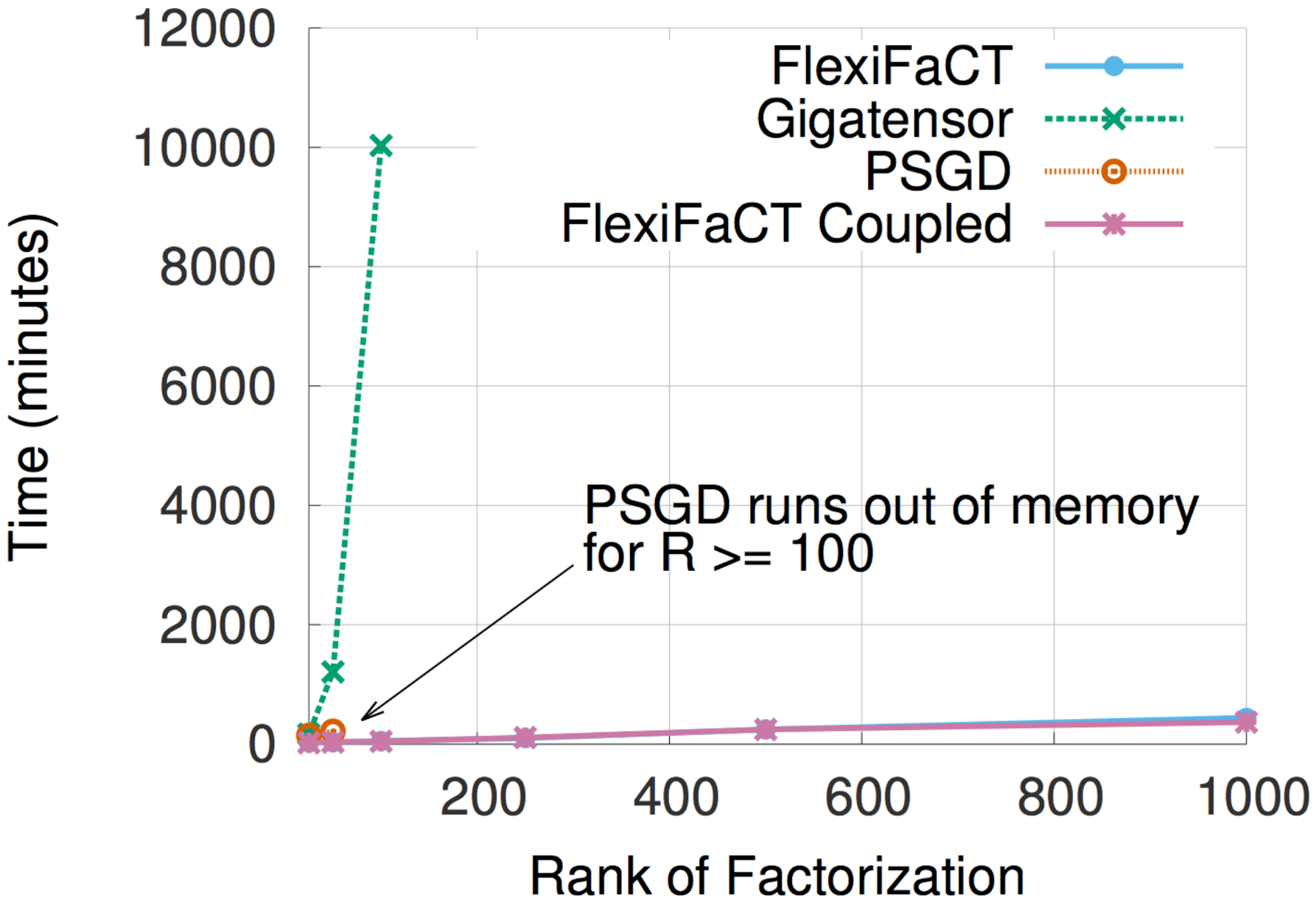
Results: Data Size Scalability



Results: Tensor Dimension Scalability



Results: Rank Scalability



FlexiFaCT Summary

- Versatility: wide spectrum of settings for matrices, tensors, coupled tensor-matrix settings as well as different loss functions

- Scalability: scales well with both input size as well as number of model parameters

- Convergence: proof of convergence even with constraints like non-negativity

- Usability and reproducibility: runs on stock Hadoop with open-sourced code