Distributed Learning

Akash Dhasade, Anne-Marie Kermarrec

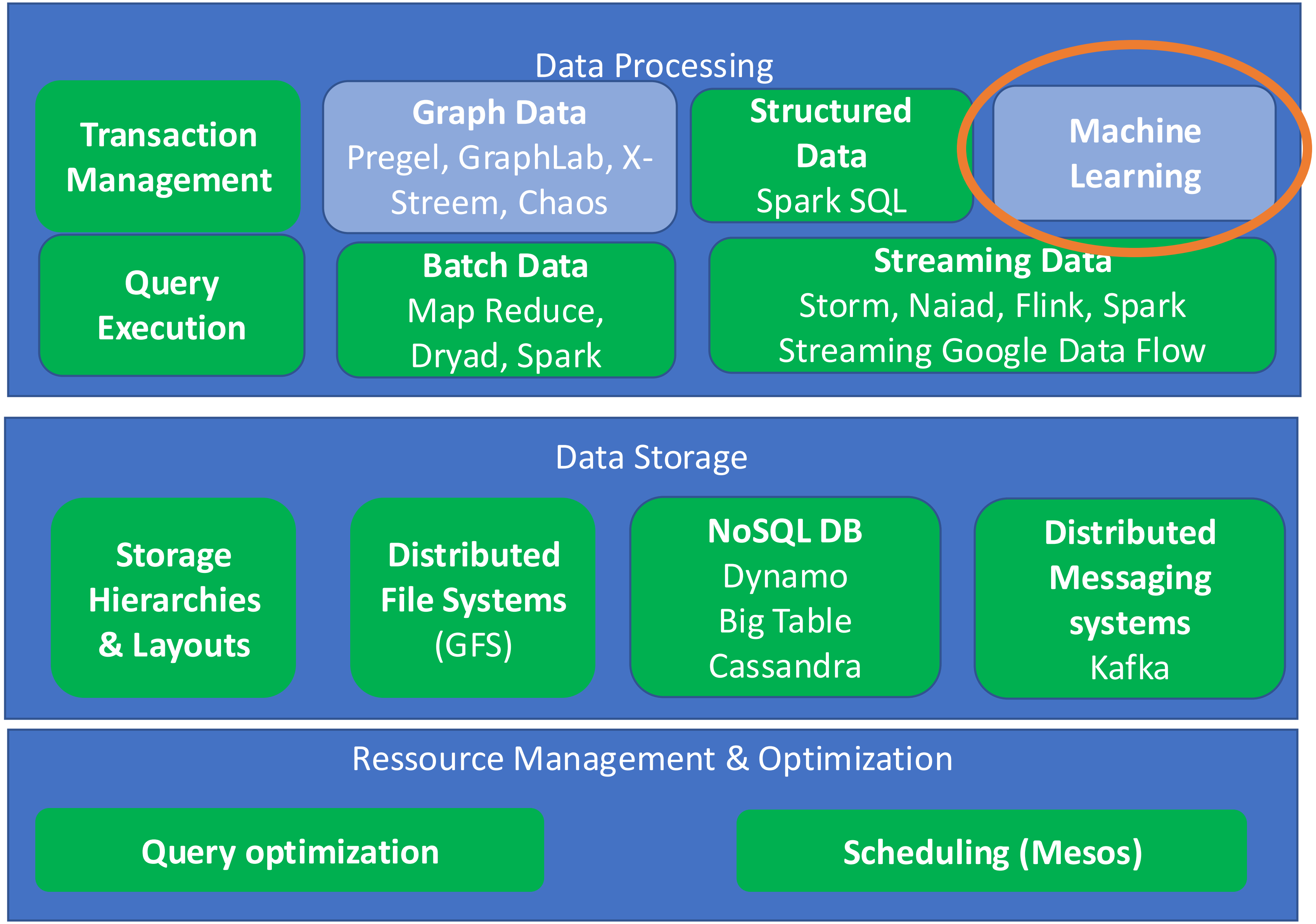
Where are we?

Gossip Protocols

Consistency protocols
CAP Theorem

Distributed/decentralized systems

Data science software stack



Data Processing

Transaction Management

Graph Data
Pregel, GraphLab, X-Streem, Chaos

Structured Data
Spark SQL

Machine Learning

Query Execution

Batch Data
Map Reduce, Dryad, Spark

Streaming Data
Storm, Naiad, Flink, Spark Streaming Google Data Flow

Data Storage

Storage Hierarchies & Layouts

Distributed File Systems
(GFS)

NoSQL DB
Dynamo
Big Table
Cassandra

Distributed Messaging systems
Kafka

Ressource Management & Optimization

Query optimization

Scheduling (Mesos)

Machine Learning

Web search, spam detection, recommendation systems, advertizing, voice recognition, image classification, document analysis, NLP

Learn models from examples: training data

Can be expressed with an objective function

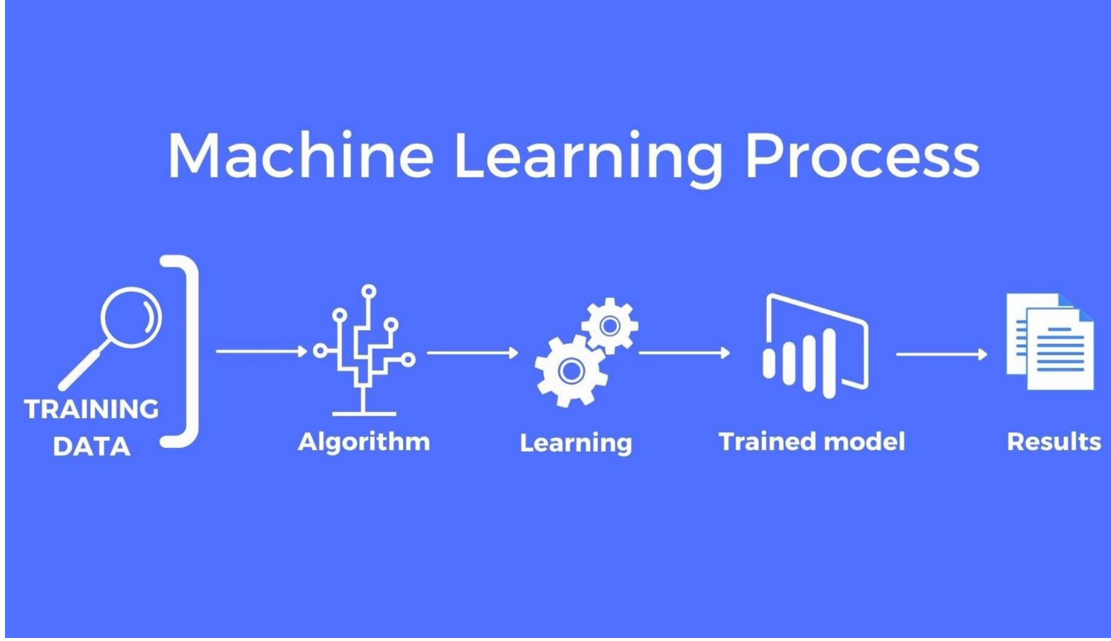
Learning algorithm typically minimizes an objective function

Starts from an initial model

Iteratively refines this model

Stops when optimal solution found or considered converged

Challenges



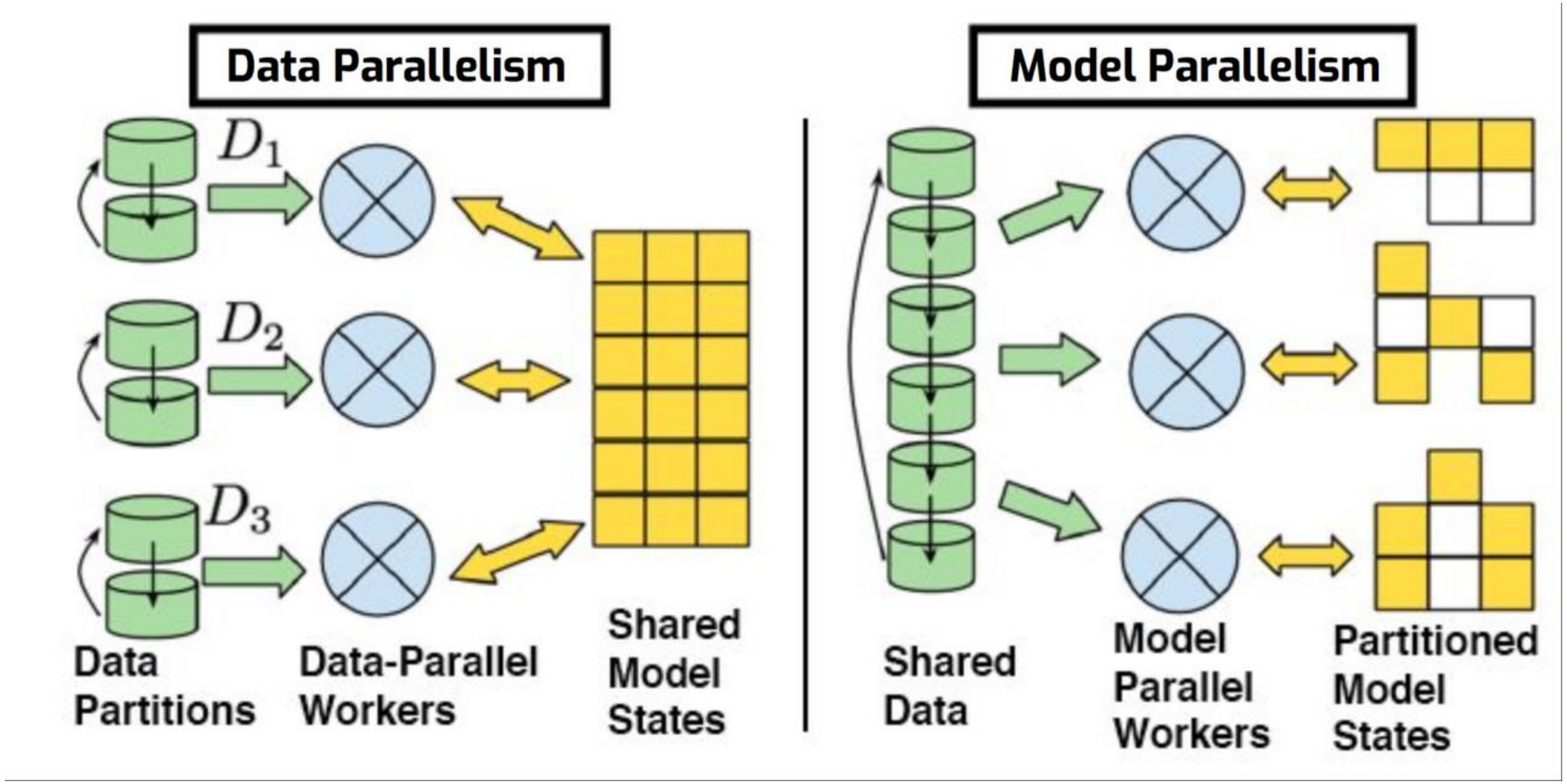
- Training Data is **Large** – 1TB to 1PB

- Complex Models with **Billions** and **Trillions** of Parameters
  - GPT-3: 175 billion
  - GPT-4: > 1 trillion
- No single machine can process such large models

Distributed machine Learning

- Datacenters
  - Model parallelism
  - Data parallelism
- Federated learning
- Decentralized learning

Model versus Data Parallelism



Data vs Model Parallelism in TensorFlow, from Illia Polosukhin slide deck.

Model training

$$maximize \quad \frac{1}{n}\sum_{i=1}^{n} p(y_i|x_i, w)$$

Example

Given input

Maximize $p(3|x_0,w)$

For a training dataset containing n samples $(x_i, y_i), 1 \leq i \leq n$, the training objective is:

$$\min_{w \in \mathbb{R}^d} f(w) \qquad \text{where } f(w) \overset{\text{def}}{=} \frac{1}{n}\sum_{i=1}^{n} f_i(w)$$

$f_i(w) = l(x_i, y_i, w)$ is the loss of the prediction on example (x_i, y_i)

Gradient Descent

Iterative optimization algorithm: iteratively adjust the parameters of a model

Step 1: Take random values for the parameters

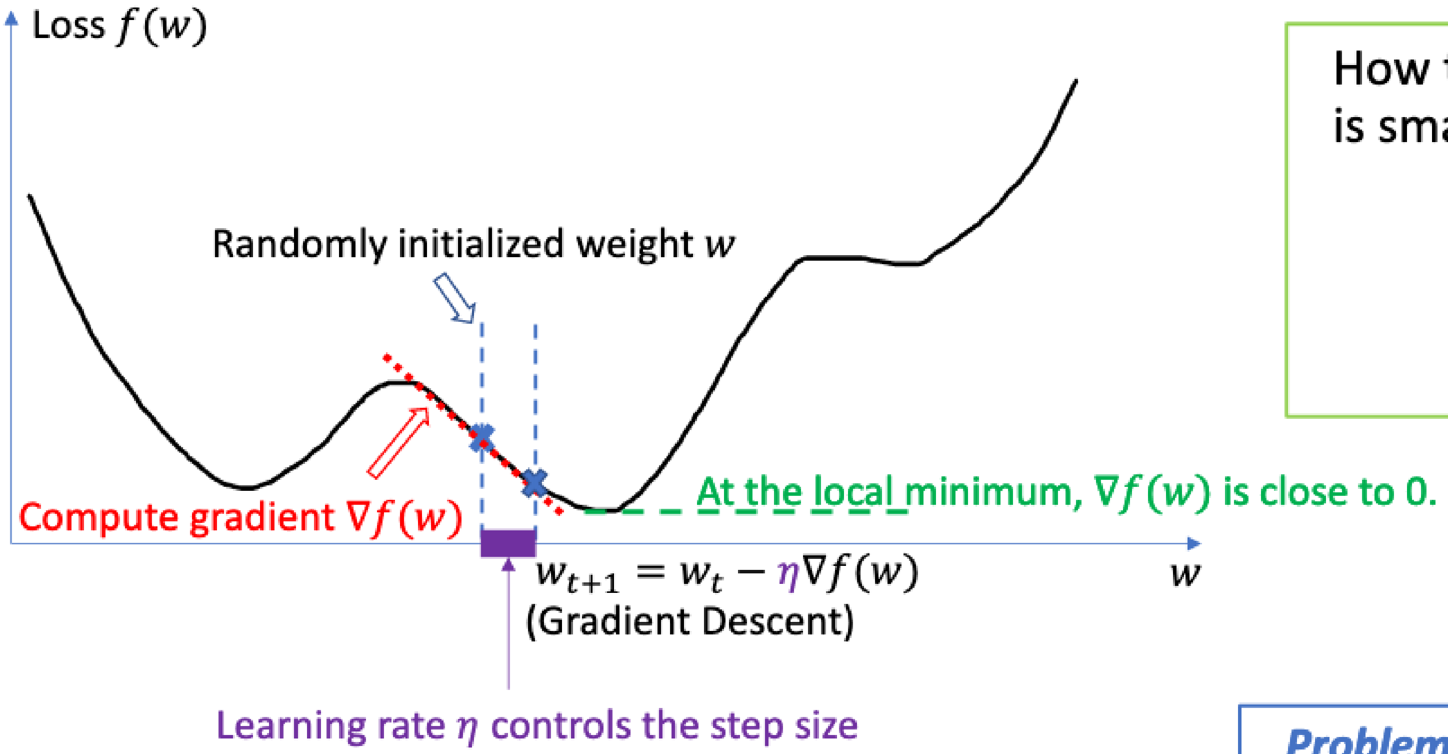
Step 2: Compute the gradients of the function wrt the parametres

Step 3: Calculate the step size: Step Size = negative gradient (Slope) x learning rate

Step 5: Calculate the New Parameters = Old parameters – Step Size

May take very long when millions of data points. Example use 23000 genes to predict if someone has a disease: 23000 derivatives to compute X 1M samples:
23 B terms at each step * 1000

Gradient Descent



Loss $f(w)$

Randomly initialized weight w

Compute gradient $\nabla f(w)$

At the local minimum, $\nabla f(w)$ is close to 0.

$w_{t+1} = w_t - \eta \nabla f(w)$

(Gradient Descent)

w

Learning rate η controls the step size

How to stop? – when the update is small enough – converge.

$$\| w_{t+1} - w_t \| \leq \epsilon$$

or $$\| \nabla f(w_t) \| \leq \epsilon$$

***Problem**: Usually the number of training samples n is large – slow convergence*

Stochastic Gradient Descent (SGD)

- At each step of gradient descent, instead of processing all training samples, randomly pick a small subset (mini-batch) of training samples x_k,y_k.

$$w_{t+1} \leftarrow w_t - \eta \nabla f(w_t; x_k, y_k)$$

- Compared to gradient descent, SGD may take more steps to converge, but each step is much faster

Parameter server

Data Parallelism

Context

- Distributed optimization and inference for ML problems
- Cloud-computing settings
    - Machines may be unreliable
    - Jobs may be preempted
    - Data may be lost
    - Varying performance (network and computation)

Design features

General-purpose framework exploiting specific datatypes of ML problems

Parameters stored in a distributed database (KVS) accessible through the network

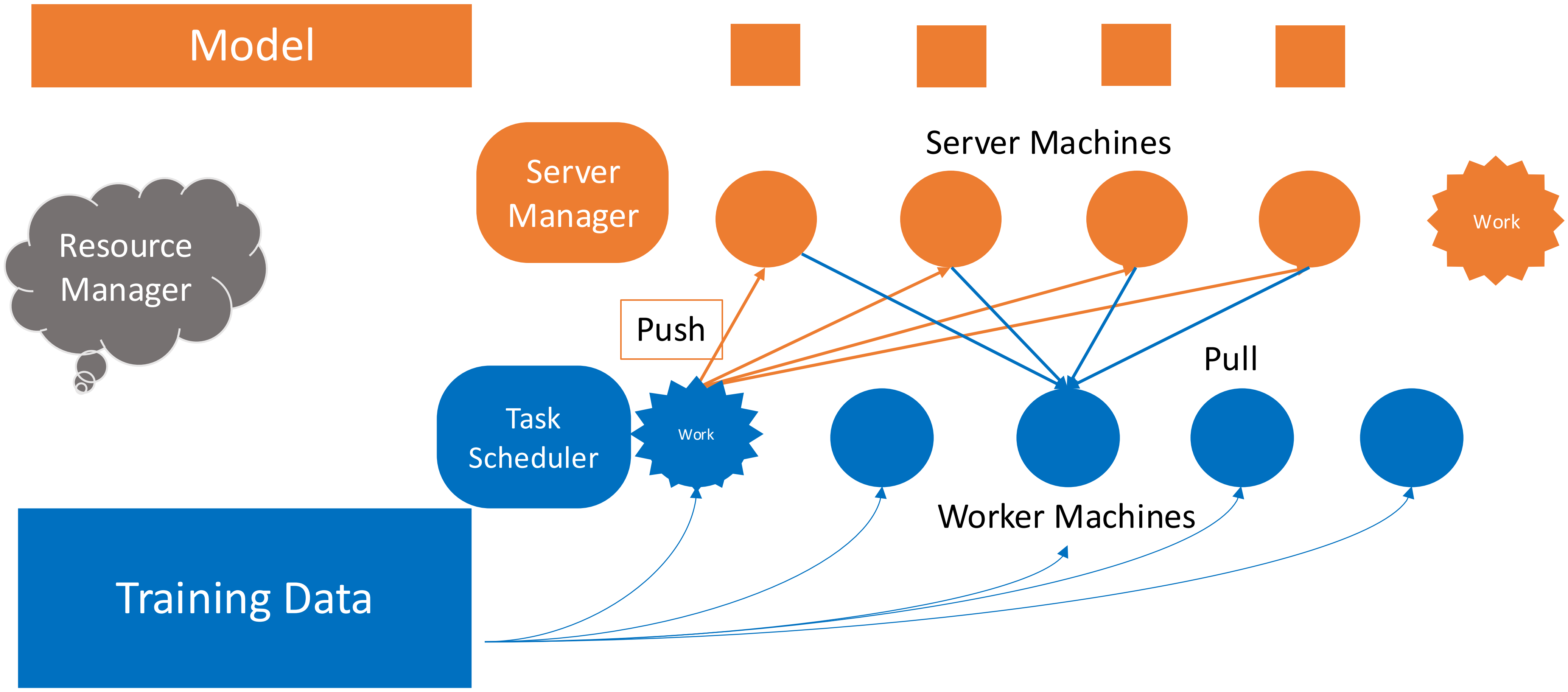
1. **Efficient communication**: asynchronous communication model
2. **Elastic scalability**: new nodes can be added without restarting the running framework (use of a DHT)
3. **Fault-tolerance** and durability: optimized data replication architecture for fast node failure recovery
4. **Ease of use:** globally shared parameters represented as linear algebra vectors or matrices rather than individual key-value pairs

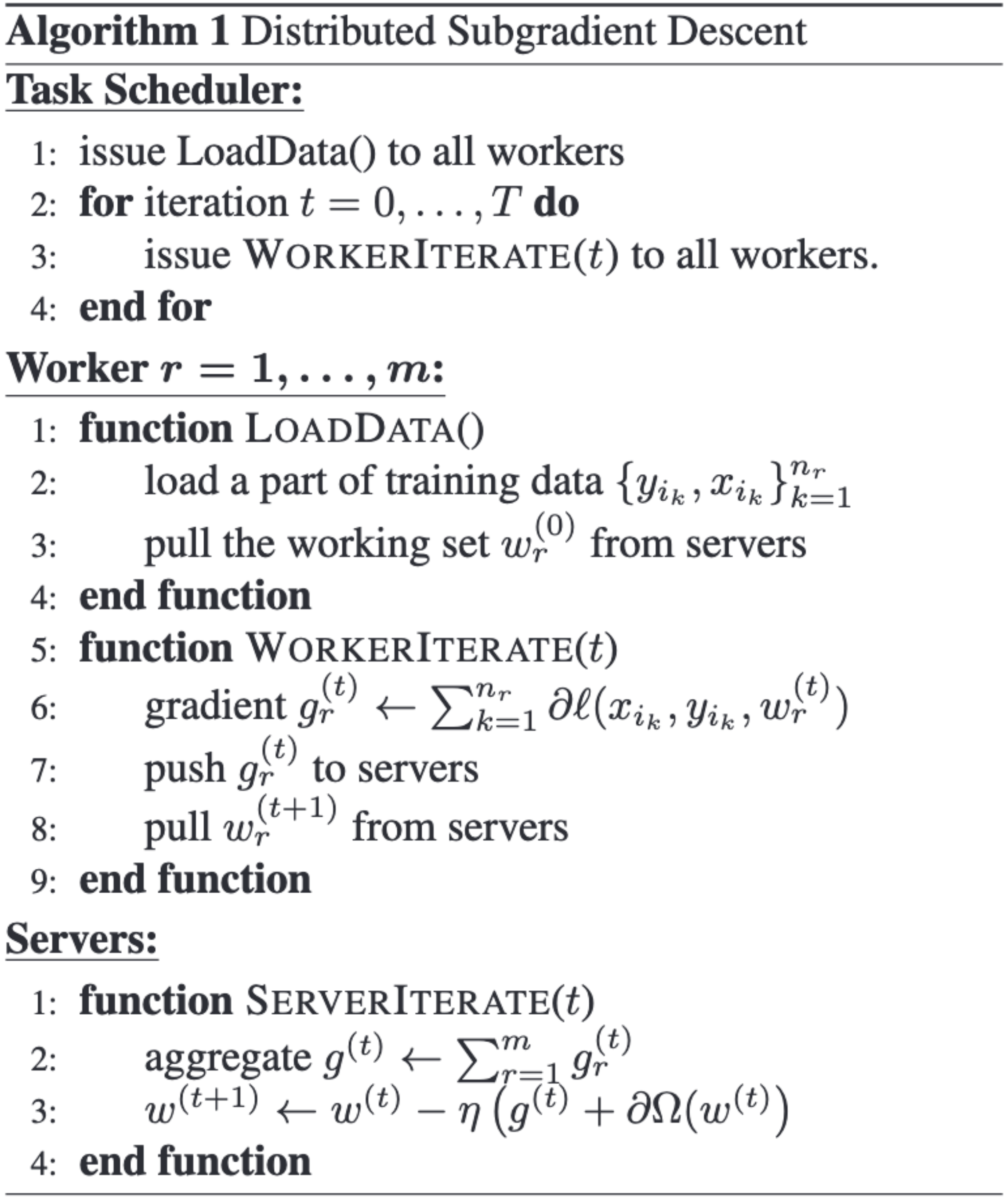
Features
Efficient Communication
Elastic Scalability
Fault Tolerance
Ease of Use

Architecture

- Server nodes
  - maintain a partition of the globally shared parameters
  - communicate with each other to replicate or migrate parameters
  - perform bookeeping and global aggregation steps
- Client nodes
  - perform the bulk of the computation
  - store locally a portion of the training data
  - communicate with server nodes to update and retrieve the shared parameters

Architecture





Algorithm 1 Distributed Subgradient Descent

```
Task Scheduler:
issue LoadData() to all workers
for iteration $t = 0, \ldots, T$ do
    issue WorkerIterate($t$) to all workers.
end for

Worker $r = 1, \ldots, m$:
function LoadData()
    load a part of training data $\{y_{i_k}, x_{i_k}\}_{k=1}^{n_r}$
    pull the working set $w_r^{(0)}$ from servers
end function
function WorkerIterate($t$)
    gradient $g_r^{(t)} \leftarrow \sum_{k=1}^{n_r} \partial \ell(x_{i_k}, y_{i_k}, w_r^{(t)})$
    push $g_r^{(t)}$ to servers
    pull $w_r^{(t+1)}$ from servers
end function

Servers:
function ServerIterate($t$)
    aggregate $g^{(t)} \leftarrow \sum_{r=1}^{m} g_r^{(t)}$
    $w^{(t+1)} \leftarrow w^{(t)} - \eta \left( g^{(t)} + \partial \Omega(w^{(t)}) \right)$
end function
```

- Workers get the Assigned training data
- Workers **Pull** the Working set of Model
- Iterate until Stop:
    - Workers **Compute** Gradients
    - Workers **Push** Gradients
    - Servers **Aggregate** into current model
    - Workers **Pull** updated model

Shared parameters: Key-Value vectors

- Model Parameters are represented as Key – Value pairs
- Use of vector semantics to send large amount of data in bulk: Batch several key-value pairs required to compute a vector/matrix instead of sending them one by one
- Exploit vector/matrix structure for linear algebra operation

Range Push and Pull

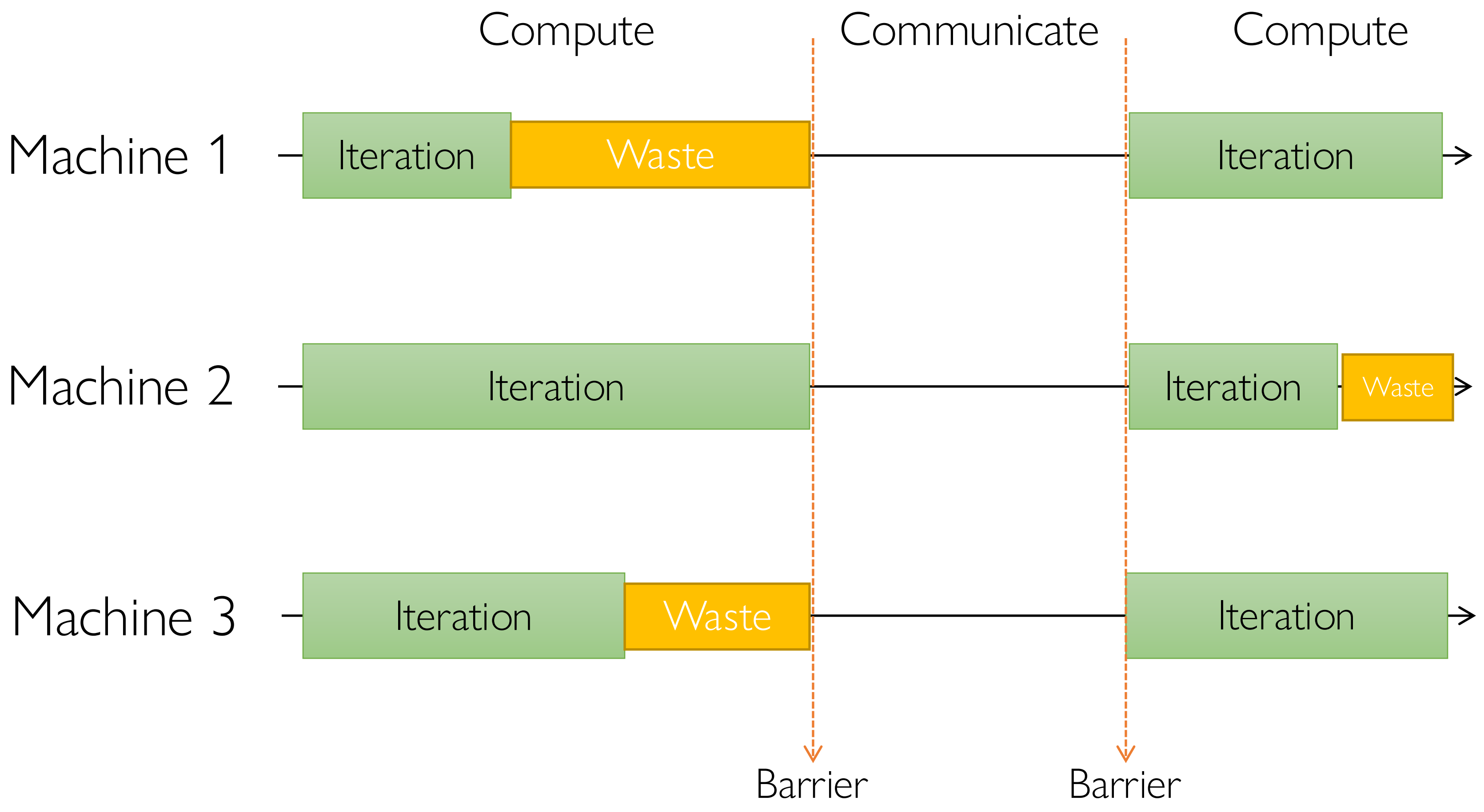
- Data communication between workers and servers: *PUSH* and P*ULL* operations.

- PS minimizes network traffic by using RANGE based *PUSH* and *PULL.*

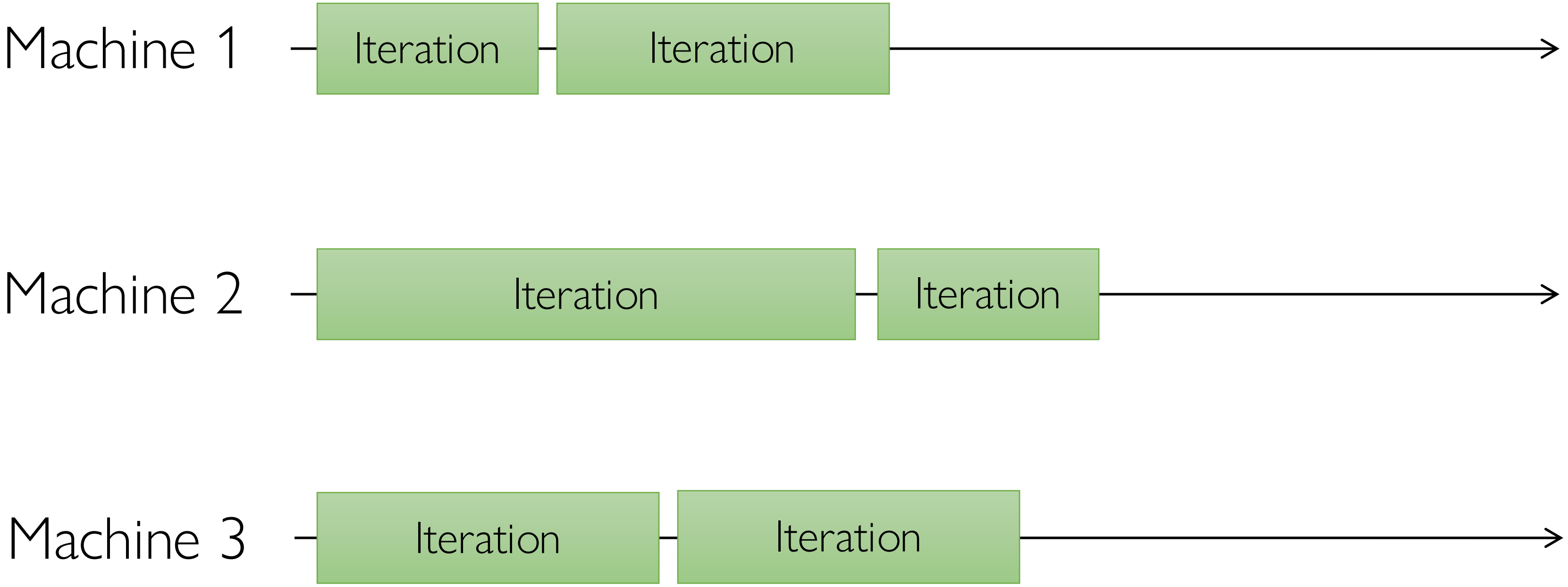
- Example: Let ***w*** denote parameters of some model
    - *w.push(**Range,** dest)*
    - *w.pull**(Range**, dest**)***
    - These methods will send/receive all existing entries of ***w*** with ***keys*** in ***Range***
- Non blocking operations
    - The caller inserts its requests in a queue and resume computation

Synchronous Execution

This is conceptual



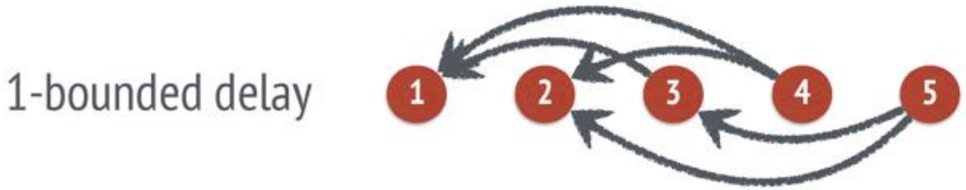
Asynchronous Execution



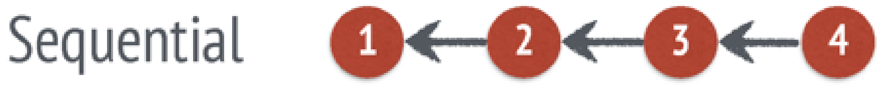
Enable more frequent coordination on parameter values

Flexible Consistency model

- Asynchronous communication may lead to inconsistencies
- PS provides flexible data consistencies models for applications to select
  - Eventual : the PS never stalls regardless of resource availability
  - Bounded Delay



  - Sequential: 0-bounded-delay, fully synchronous model

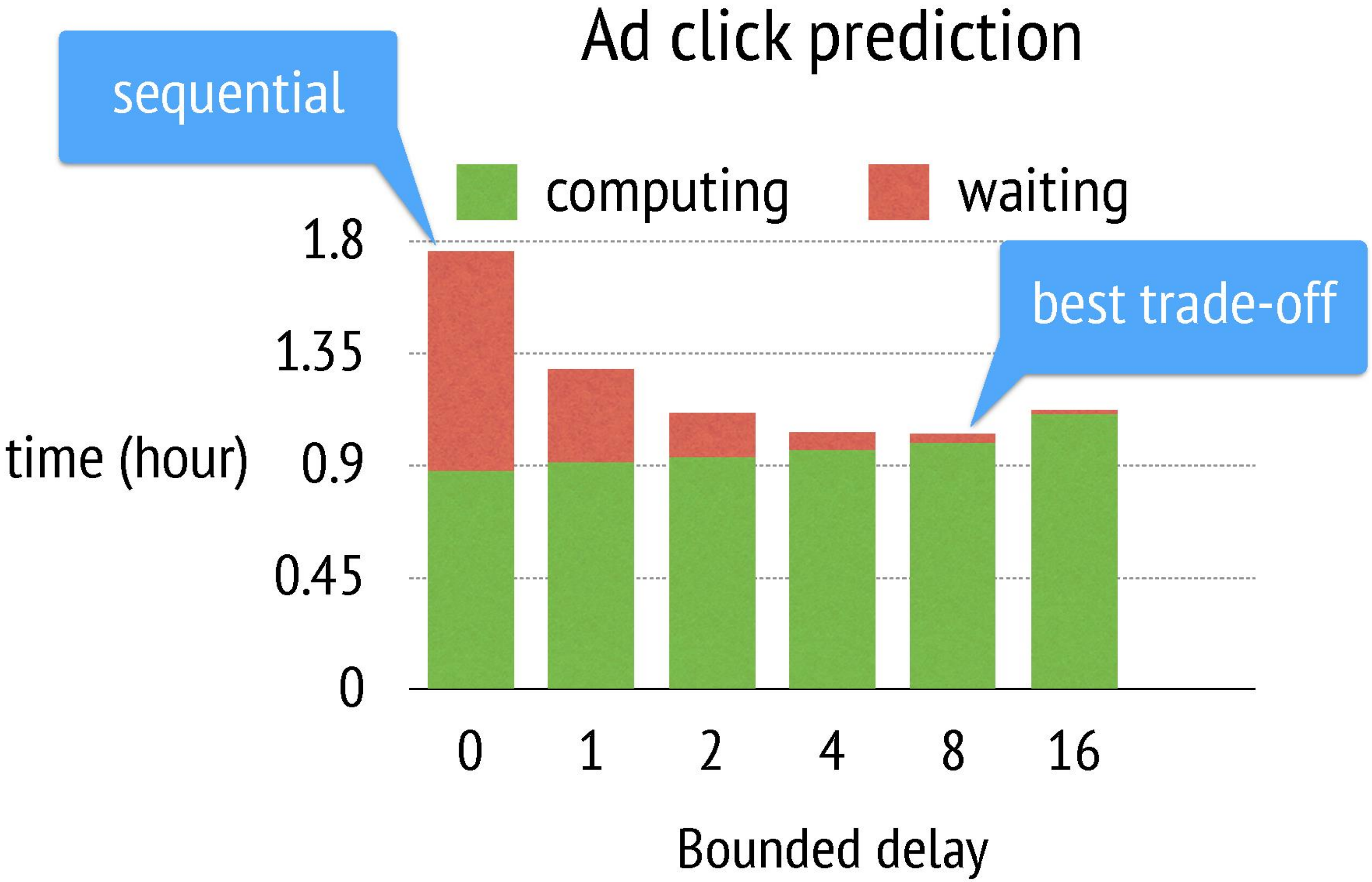


Stale synchronous parallel (SSP):

- Stale synchronous parallel (SSP):
- Global clock time *t*
- Parameters workers “get” *can* be out of date
- but can’t be older than ***t-τ***

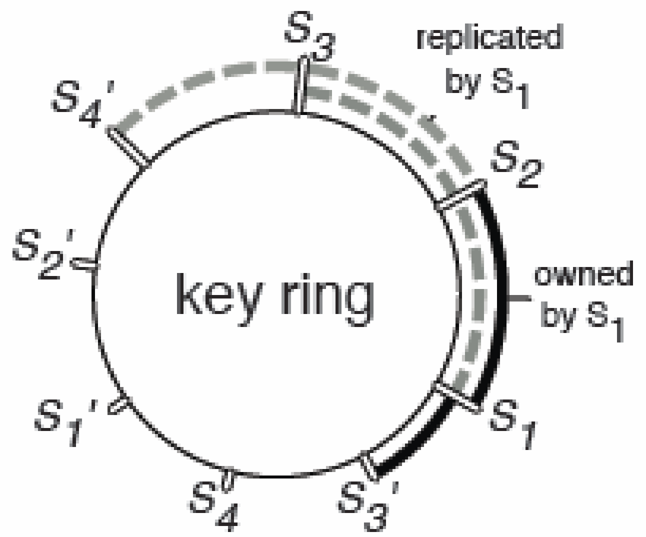
- ***τ*** controls “staleness”
  - ∞ : fully asynchronouS
  - 1: synchronous

Results from [Li et al., 2014]



Consistent Hashing & Replication

- Use of DHT range partitioning
- Servers hashed in the ring
- Virtual servers for load balancing
- Server nodes store a replica of (Key, value) pairs on *k* nodes counter clockwise to it.



Summary

Efficient Communication:

- Batching (key,value) pairs in Linear Algebra objects
- Caching keys at worker and server nodes for local access

Flexible Consistency Models:

- Can choose between Sequential, Eventual, and Bounded delay consistency models
- Allows for tradeoffs between System Performance and Algorithmic Convergence

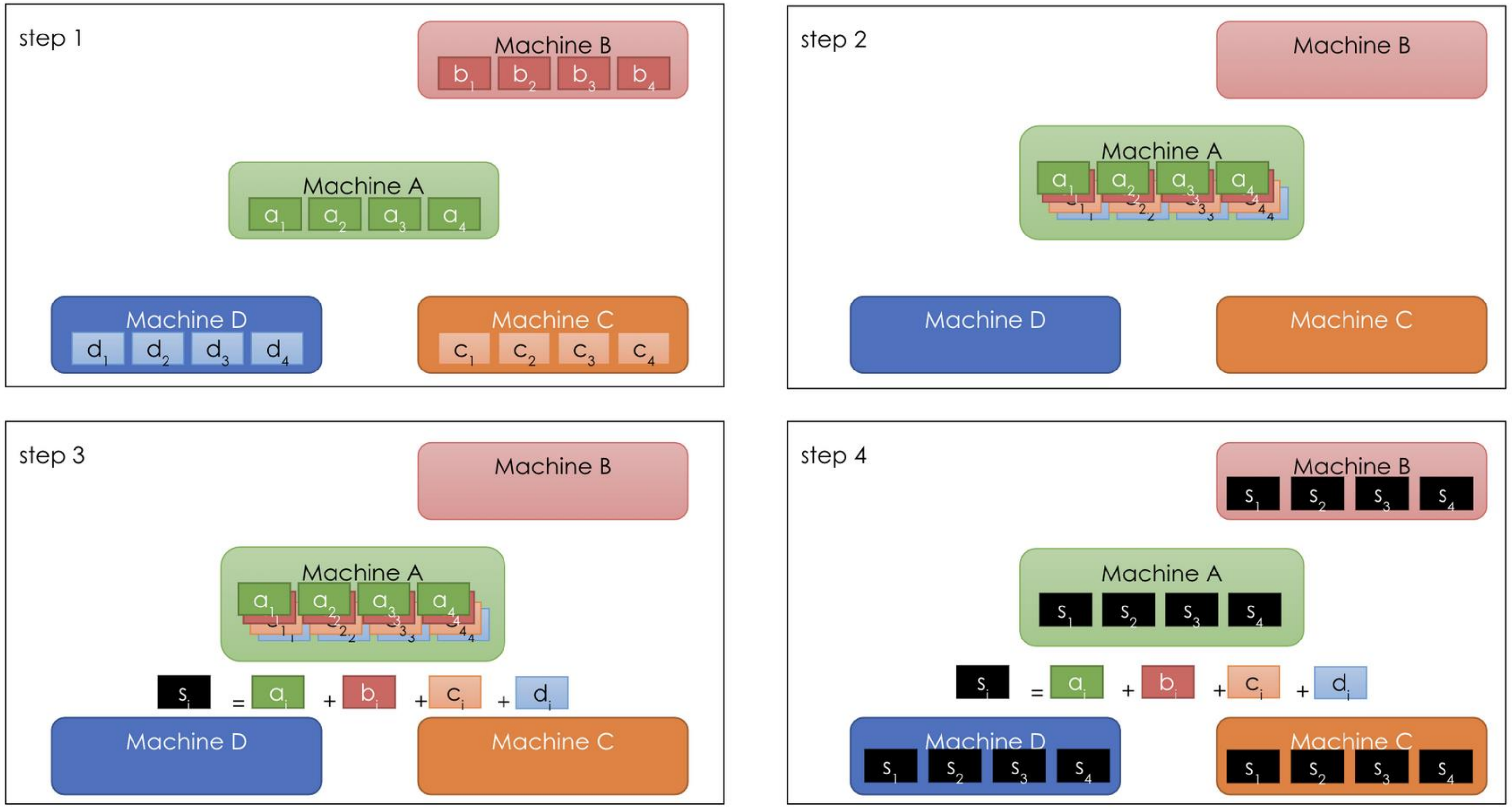
Fault Tolerance and Durability:

- Replication of data in Servers
- Failed workers can restart at the point of failure by using vector clocks

Ease of Use:

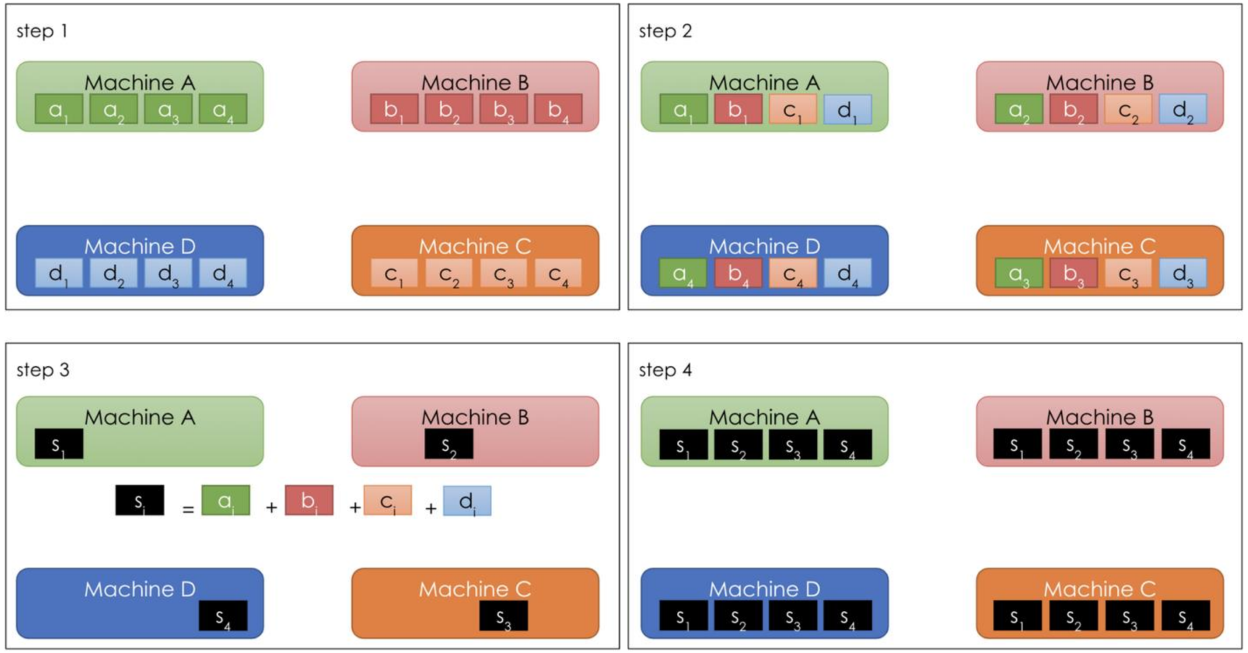
- Linear Algebra objects allow for intuitive implementation of tasks

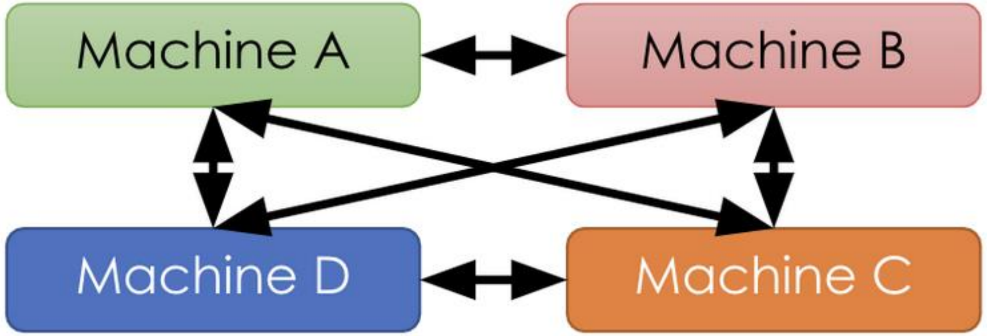
Classical Parameter Server



Bandwidth for Machine A=(N-1)*P where N is the number of machines and P is the total number of parameters. The bandwidth for machine A increases as we add more machines or have more parameters.

All Reduce



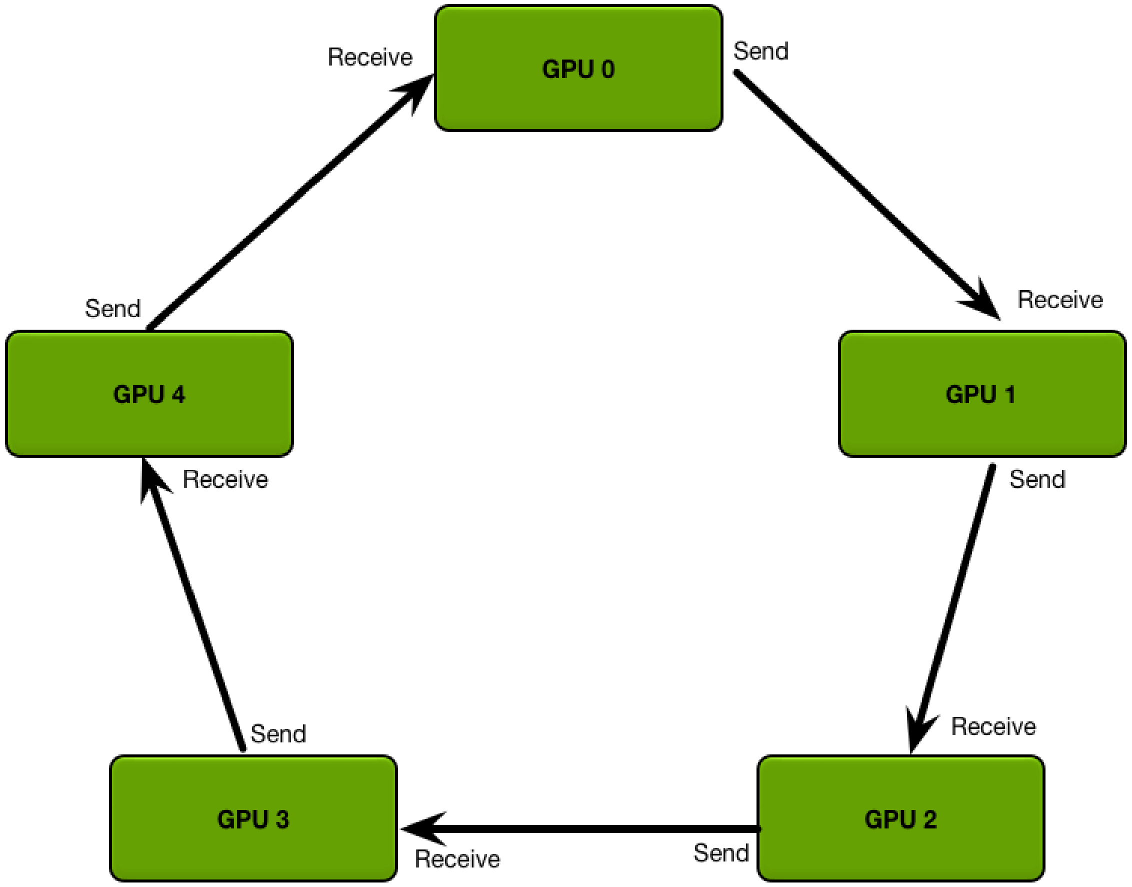


This results in a bandwidth of (N-1)*P/N for all machines, which is smaller than the one in parameter server (N-1)*P.

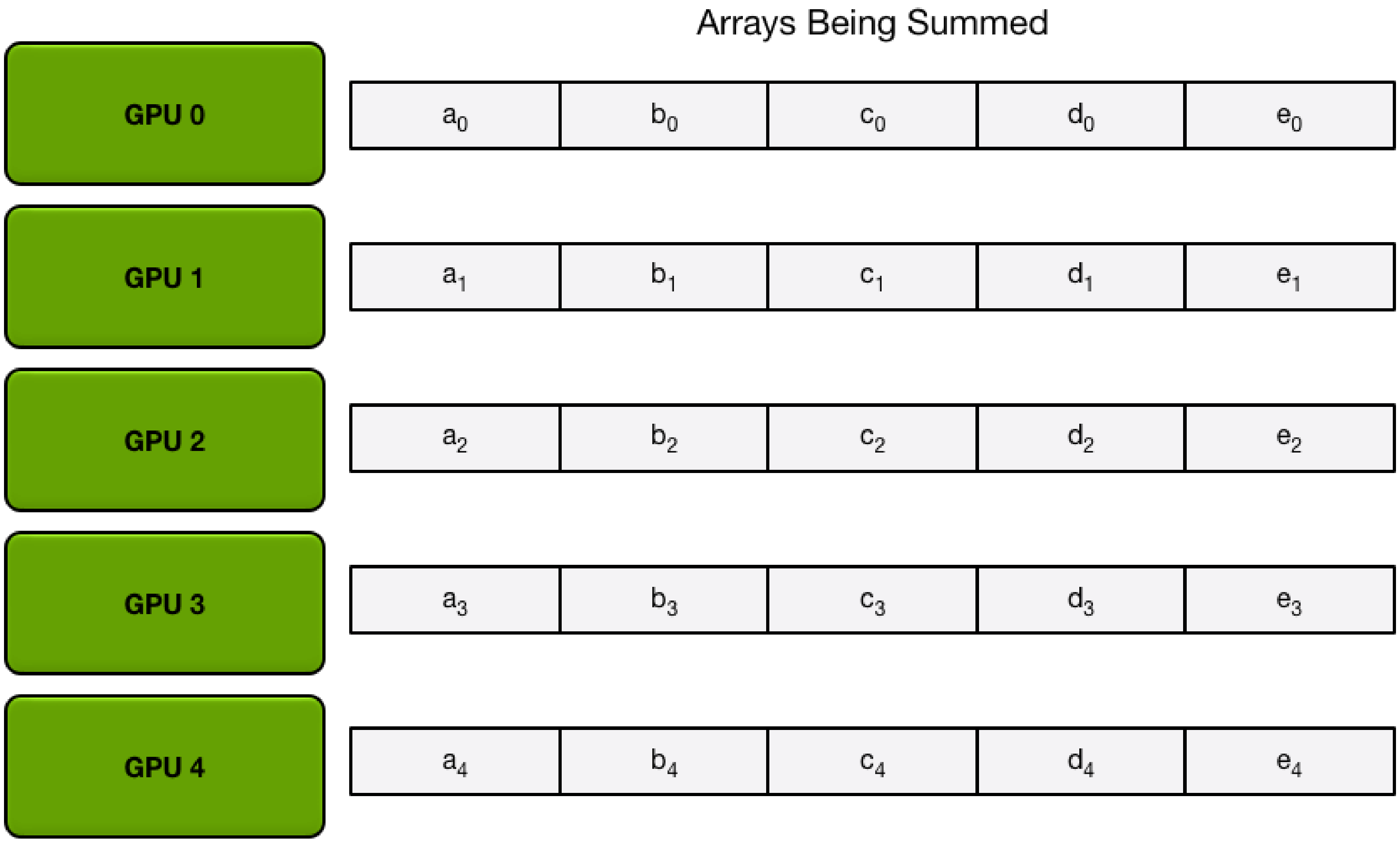
HPC All reduce

Images from Andrew Gibiansky

Scatter Reduce Phase



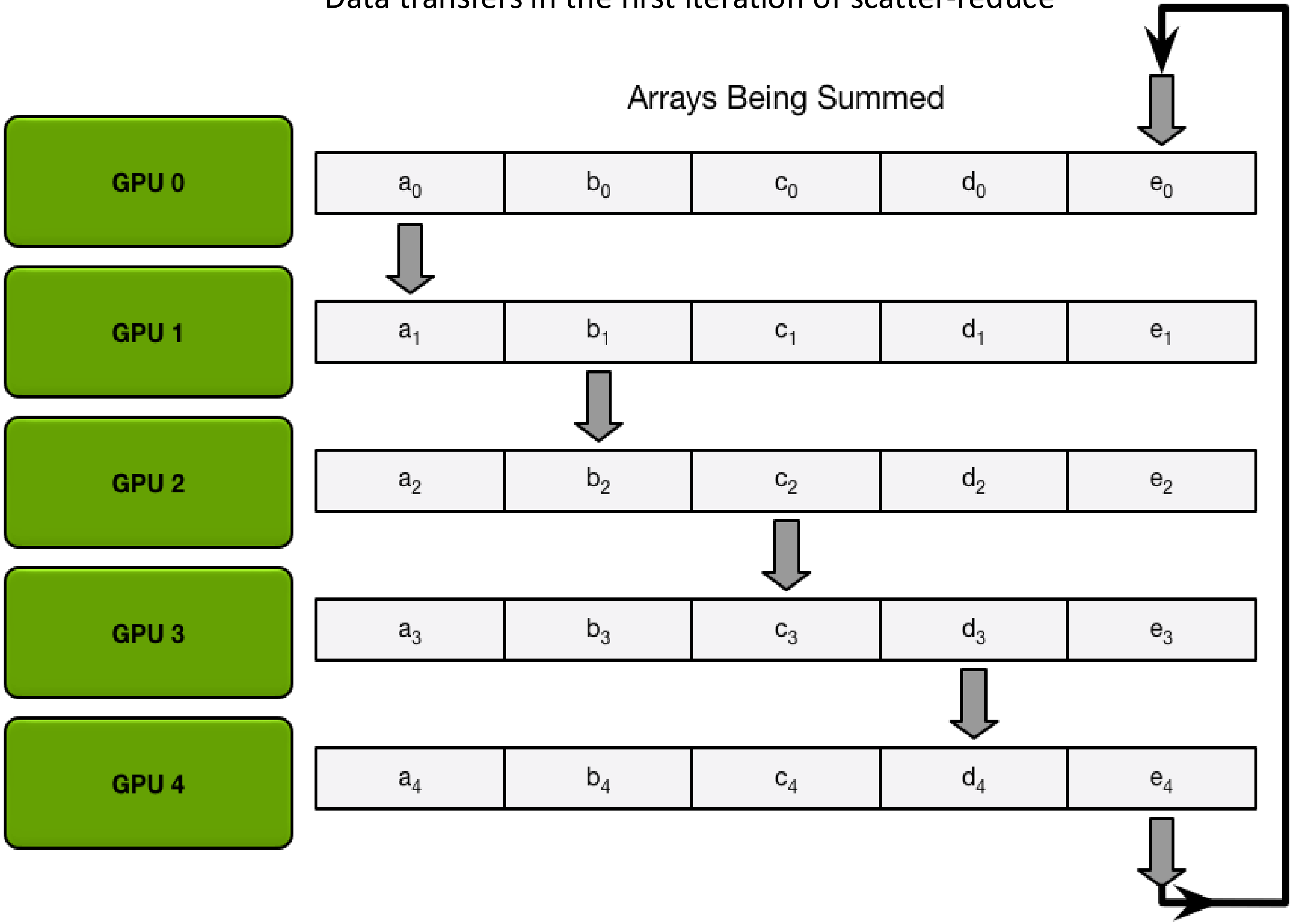
Partitioning of an array in N chunks



Arrays Being Summed

GPU 0	a_0	b_0	c_0	d_0	e_0
GPU 1	a_1	b_1	c_1	d_1	e_1
GPU 2	a_2	b_2	c_2	d_2	e_2
GPU 3	a_3	b_3	c_3	d_3	e_3
GPU 4	a_4	b_4	c_4	d_4	e_4

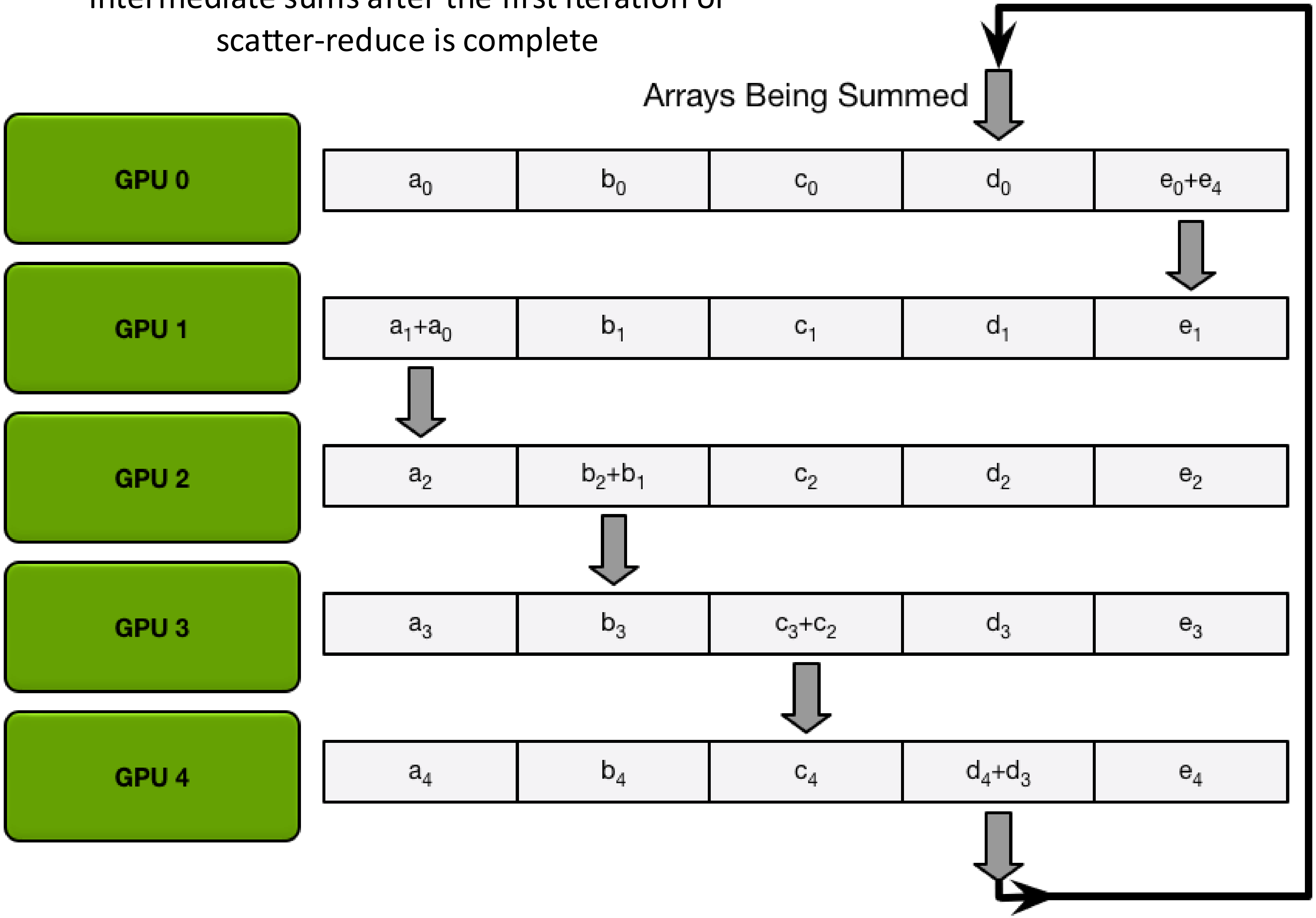
Data transfers in the first iteration of scatter-reduce



Arrays Being Summed

GPU 0	a_0	b_0	c_0	d_0	e_0
GPU 1	a_1	b_1	c_1	d_1	e_1
GPU 2	a_2	b_2	c_2	d_2	e_2
GPU 3	a_3	b_3	c_3	d_3	e_3
GPU 4	a_4	b_4	c_4	d_4	e_4

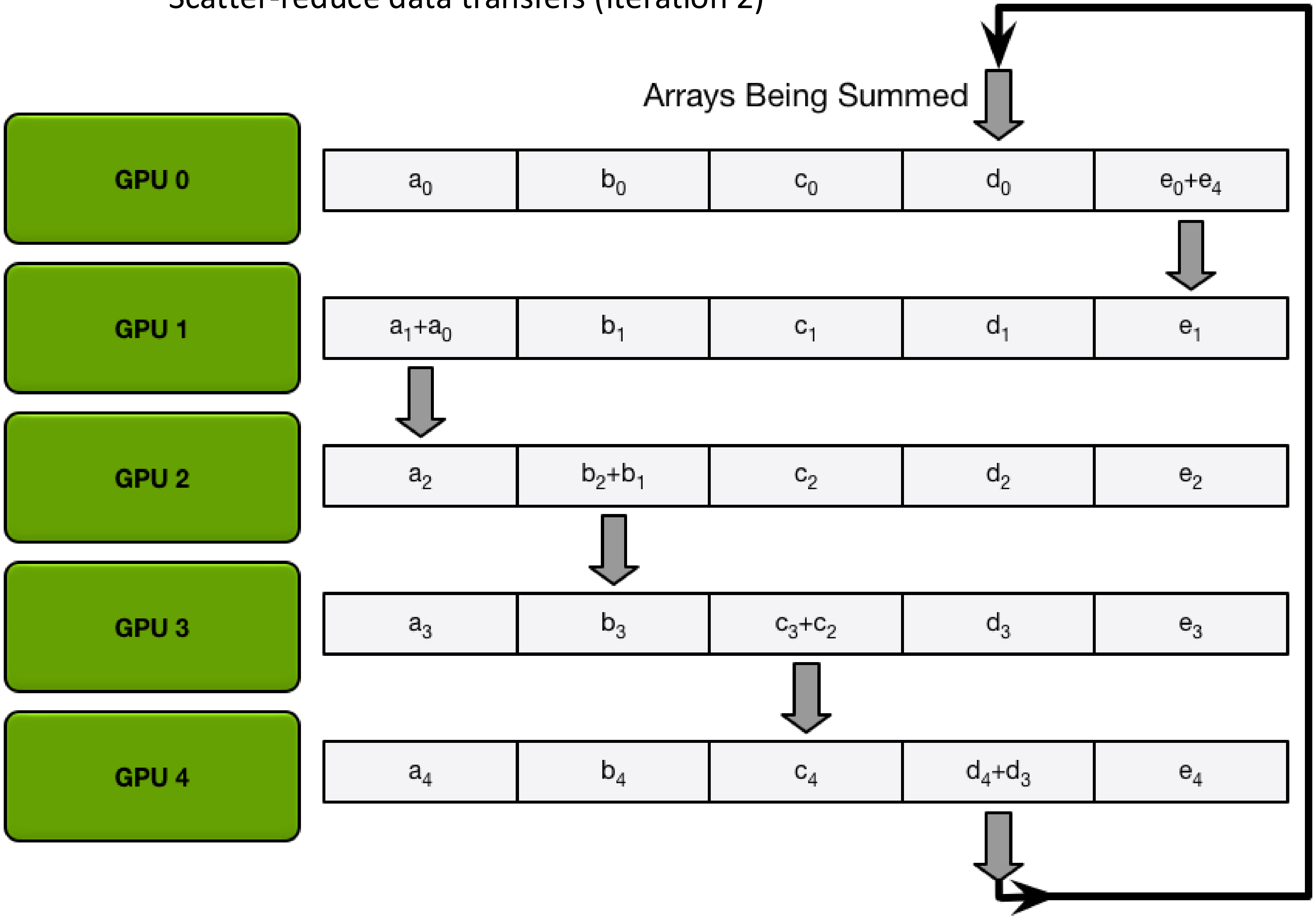
Intermediate sums after the first iteration of scatter-reduce is complete



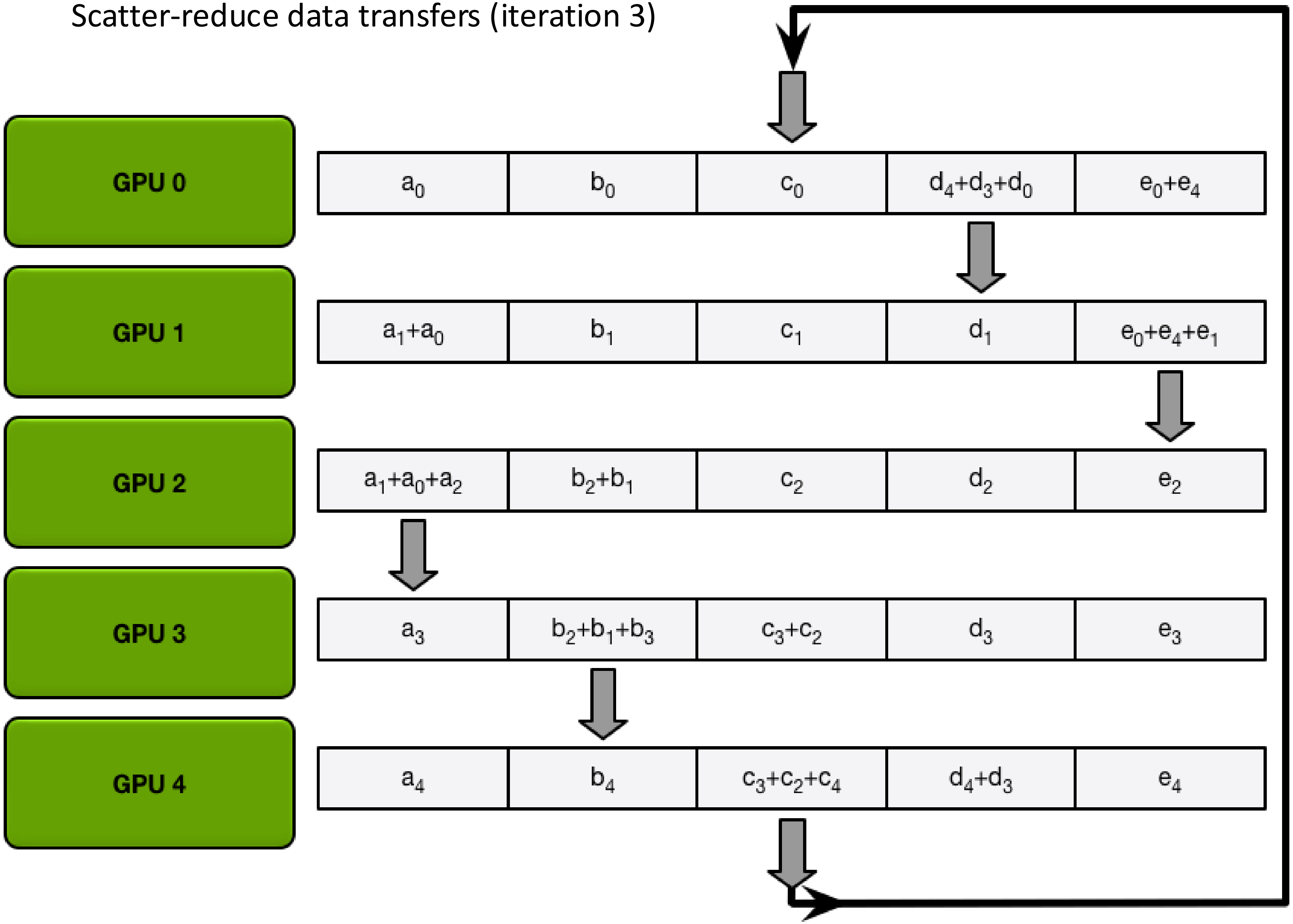
Arrays Being Summed

GPU 0	a_0	b_0	c_0	d_0	e_0+e_4
GPU 1	a_1+a_0	b_1	c_1	d_1	e_1
GPU 2	a_2	b_2+b_1	c_2	d_2	e_2
GPU 3	a_3	b_3	c_3+c_2	d_3	e_3
GPU 4	a_4	b_4	c_4	d_4+d_3	e_4

Scatter-reduce data transfers (iteration 2)

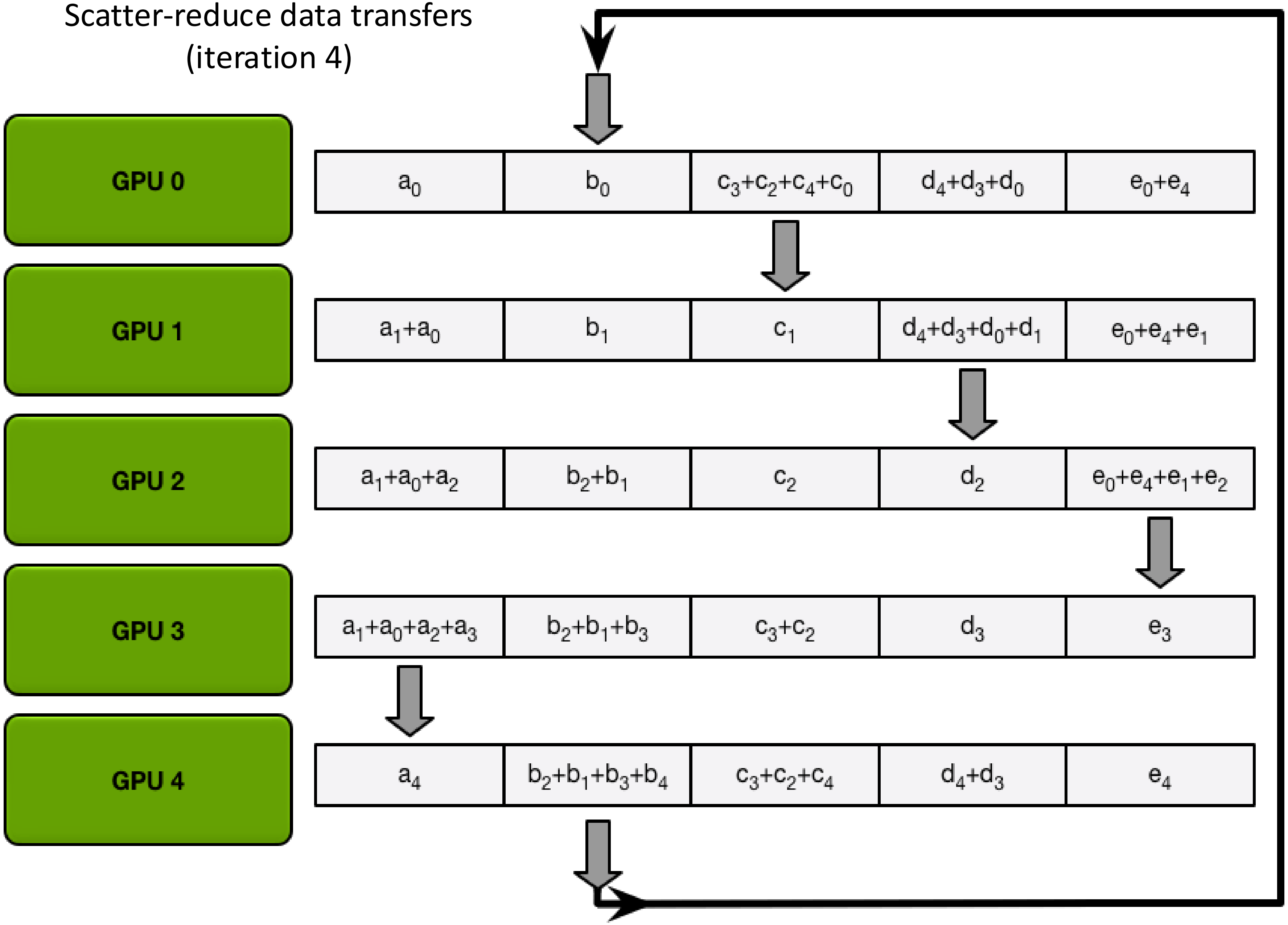


Scatter-reduce data transfers (iteration 3)



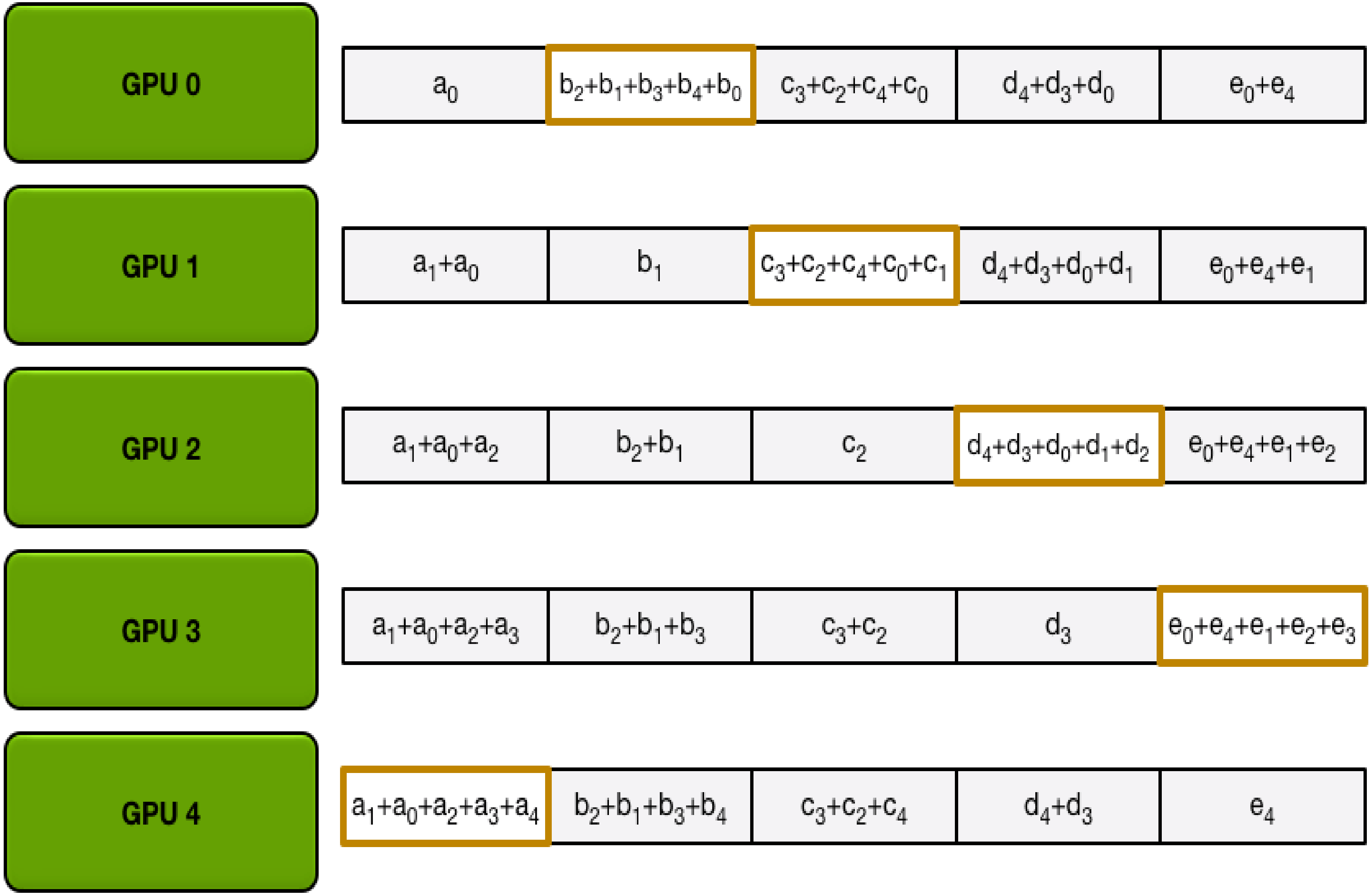
GPU 0	a_0	b_0	c_0	$d_4+d_3+d_0$	e_0+e_4
GPU 1	a_1+a_0	b_1	c_1	d_1	$e_0+e_4+e_1$
GPU 2	$a_1+a_0+a_2$	b_2+b_1	c_2	d_2	e_2
GPU 3	a_3	$b_2+b_1+b_3$	c_3+c_2	d_3	e_3
GPU 4	a_4	b_4	$c_3+c_2+c_4$	d_4+d_3	e_4

Scatter-reduce data transfers (iteration 4)



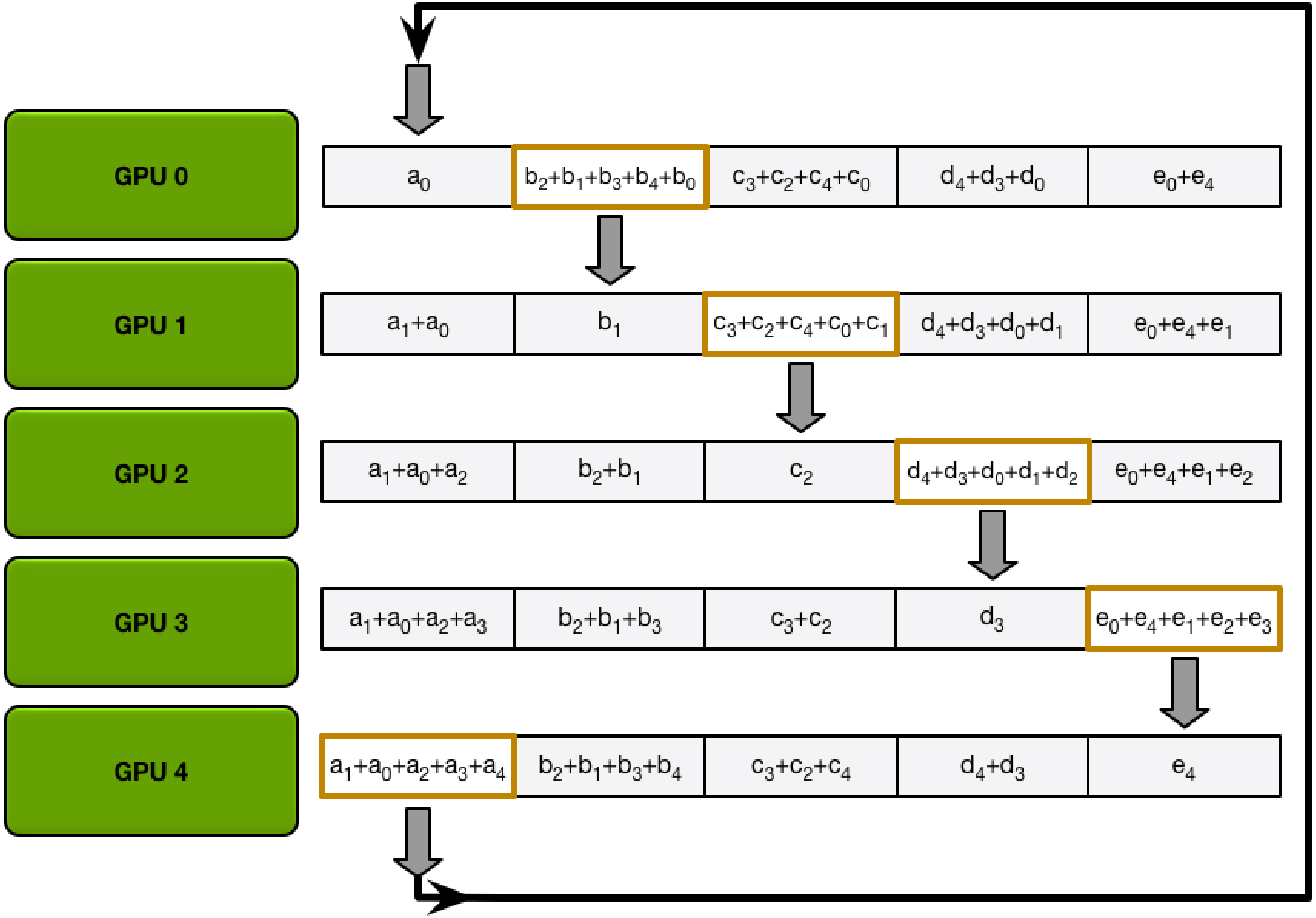
GPU 0	a_0	b_0	$c_3+c_2+c_4+c_0$	$d_4+d_3+d_0$	e_0+e_4
GPU 1	a_1+a_0	b_1	c_1	$d_4+d_3+d_0+d_1$	$e_0+e_4+e_1$
GPU 2	$a_1+a_0+a_2$	b_2+b_1	c_2	d_2	$e_0+e_4+e_1+e_2$
GPU 3	$a_1+a_0+a_2+a_3$	$b_2+b_1+b_3$	c_3+c_2	d_3	e_3
GPU 4	a_4	$b_2+b_1+b_3+b_4$	$c_3+c_2+c_4$	d_4+d_3	e_4

Final state after all scatter-reduce transfers

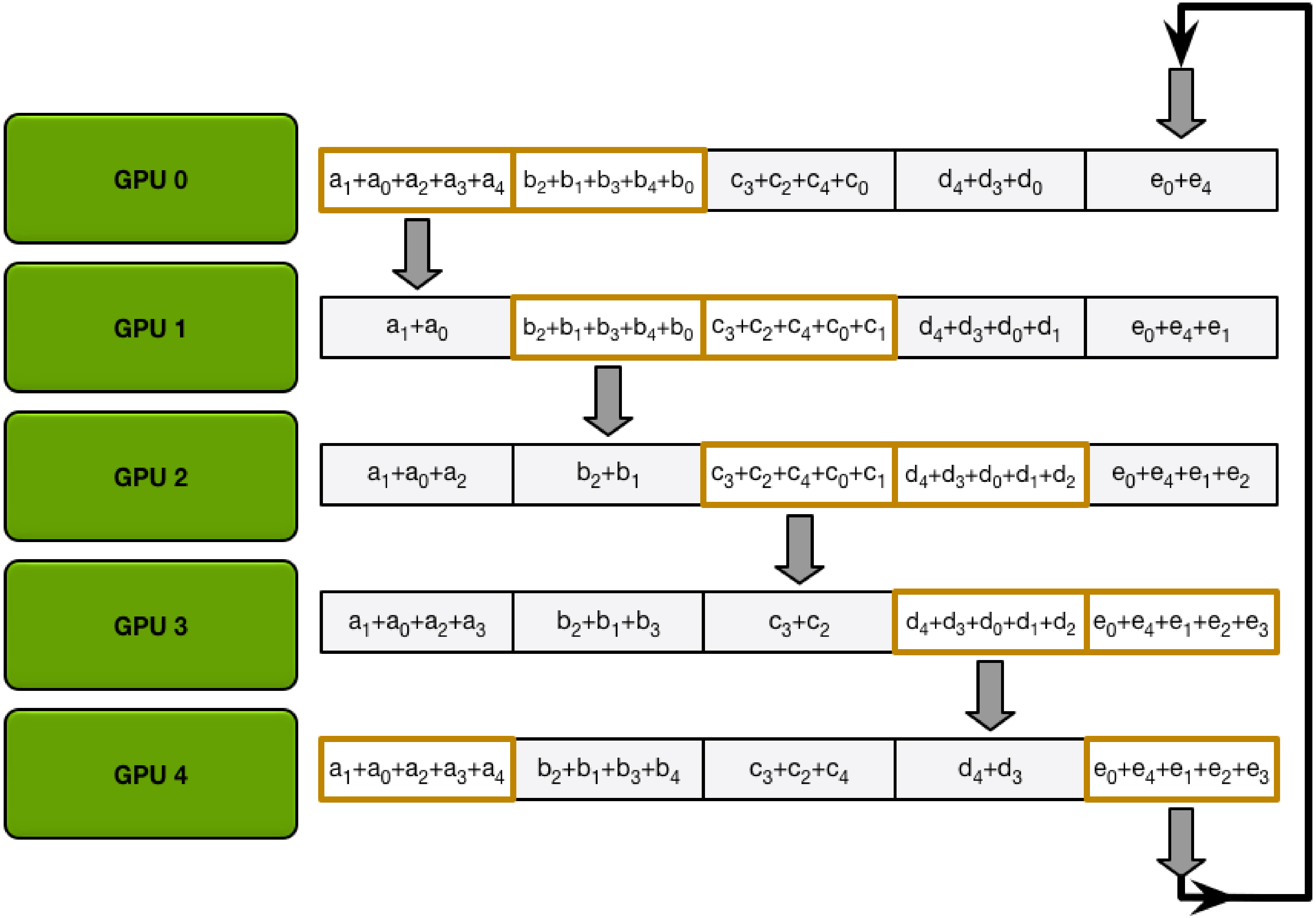


GPU 0	a_0	$\boxed{b_2+b_1+b_3+b_4+b_0}$	$c_3+c_2+c_4+c_0$	$d_4+d_3+d_0$	e_0+e_4
GPU 1	a_1+a_0	b_1	$\boxed{c_3+c_2+c_4+c_0+c_1}$	$d_4+d_3+d_0+d_1$	$e_0+e_4+e_1$
GPU 2	$a_1+a_0+a_2$	b_2+b_1	c_2	$\boxed{d_4+d_3+d_0+d_1+d_2}$	$e_0+e_4+e_1+e_2$
GPU 3	$a_1+a_0+a_2+a_3$	$b_2+b_1+b_3$	c_3+c_2	d_3	$\boxed{e_0+e_4+e_1+e_2+e_3}$
GPU 4	$\boxed{a_1+a_0+a_2+a_3+a_4}$	$b_2+b_1+b_3+b_4$	$c_3+c_2+c_4$	d_4+d_3	e_4

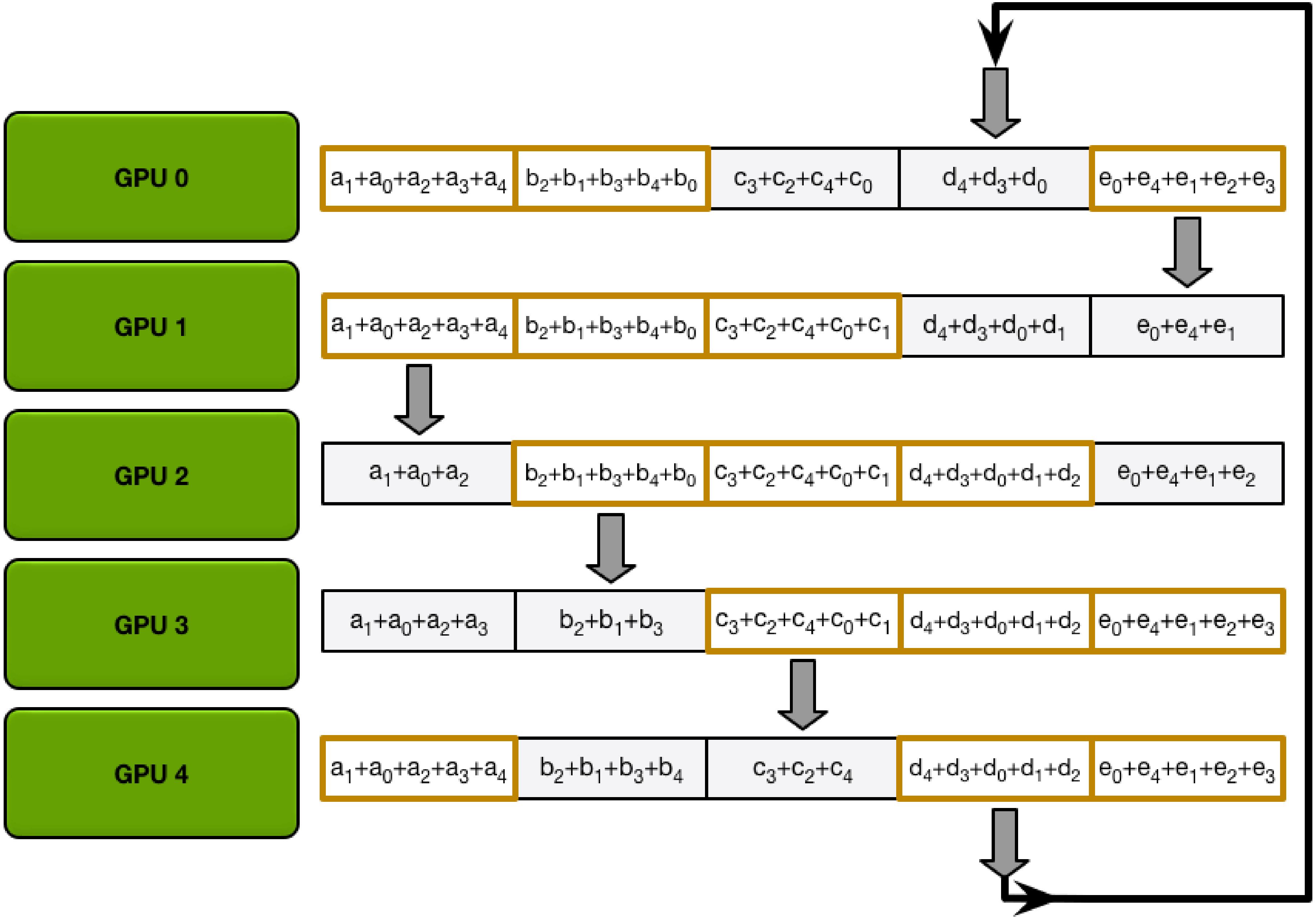
Allgather phase



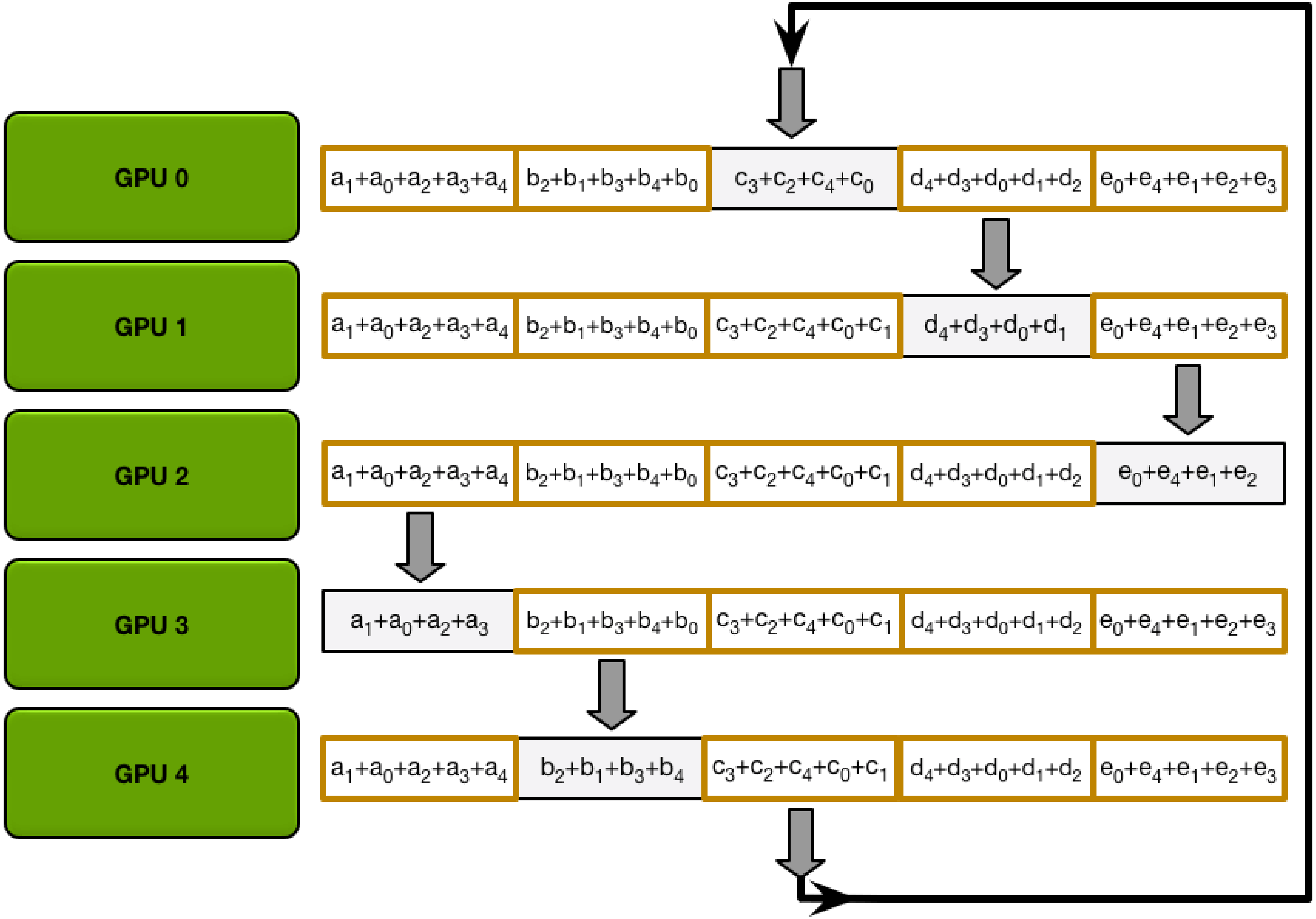
GPU 0	a_0	$b_2+b_1+b_3+b_4+b_0$	$c_3+c_2+c_4+c_0$	$d_4+d_3+d_0$	e_0+e_4
GPU 1	a_1+a_0	b_1	$c_3+c_2+c_4+c_0+c_1$	$d_4+d_3+d_0+d_1$	$e_0+e_4+e_1$
GPU 2	$a_1+a_0+a_2$	b_2+b_1	c_2	$d_4+d_3+d_0+d_1+d_2$	$e_0+e_4+e_1+e_2$
GPU 3	$a_1+a_0+a_2+a_3$	$b_2+b_1+b_3$	c_3+c_2	d_3	$e_0+e_4+e_1+e_2+e_3$
GPU 4	$a_1+a_0+a_2+a_3+a_4$	$b_2+b_1+b_3+b_4$	$c_3+c_2+c_4$	d_4+d_3	e_4



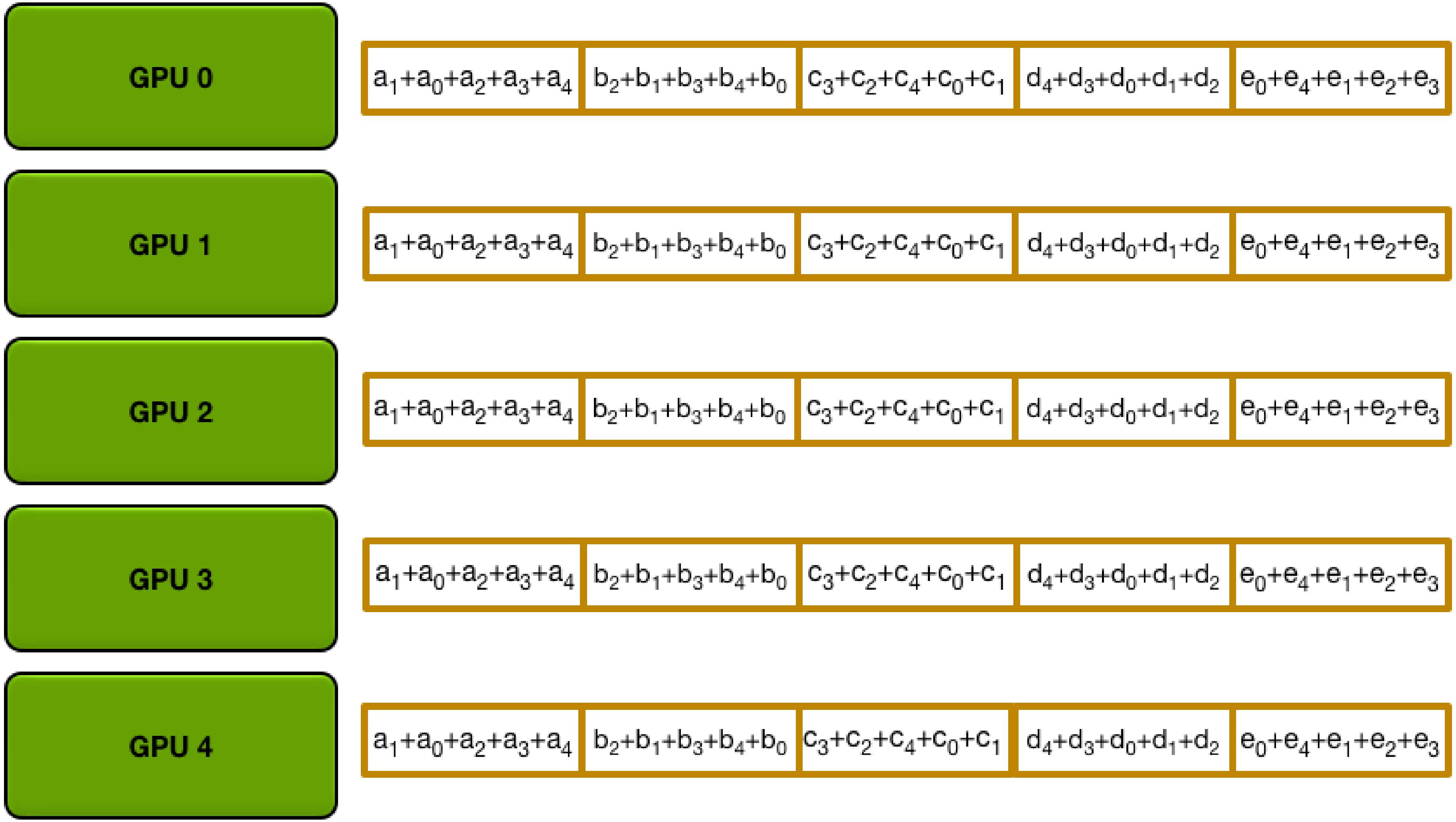
GPU 0	$a_1+a_0+a_2+a_3+a_4$	$b_2+b_1+b_3+b_4+b_0$	$c_3+c_2+c_4+c_0$	$d_4+d_3+d_0$	e_0+e_4
GPU 1	a_1+a_0	$b_2+b_1+b_3+b_4+b_0$	$c_3+c_2+c_4+c_0+c_1$	$d_4+d_3+d_0+d_1$	$e_0+e_4+e_1$
GPU 2	$a_1+a_0+a_2$	b_2+b_1	$c_3+c_2+c_4+c_0+c_1$	$d_4+d_3+d_0+d_1+d_2$	$e_0+e_4+e_1+e_2$
GPU 3	$a_1+a_0+a_2+a_3$	$b_2+b_1+b_3$	c_3+c_2	$d_4+d_3+d_0+d_1+d_2$	$e_0+e_4+e_1+e_2+e_3$
GPU 4	$a_1+a_0+a_2+a_3+a_4$	$b_2+b_1+b_3+b_4$	$c_3+c_2+c_4$	d_4+d_3	$e_0+e_4+e_1+e_2+e_3$



GPU 0	$a_1+a_0+a_2+a_3+a_4$	$b_2+b_1+b_3+b_4+b_0$	$c_3+c_2+c_4+c_0$	$d_4+d_3+d_0$	$e_0+e_4+e_1+e_2+e_3$
GPU 1	$a_1+a_0+a_2+a_3+a_4$	$b_2+b_1+b_3+b_4+b_0$	$c_3+c_2+c_4+c_0+c_1$	$d_4+d_3+d_0+d_1$	$e_0+e_4+e_1$
GPU 2	$a_1+a_0+a_2$	$b_2+b_1+b_3+b_4+b_0$	$c_3+c_2+c_4+c_0+c_1$	$d_4+d_3+d_0+d_1+d_2$	$e_0+e_4+e_1+e_2$
GPU 3	$a_1+a_0+a_2+a_3$	$b_2+b_1+b_3$	$c_3+c_2+c_4+c_0+c_1$	$d_4+d_3+d_0+d_1+d_2$	$e_0+e_4+e_1+e_2+e_3$
GPU 4	$a_1+a_0+a_2+a_3+a_4$	$b_2+b_1+b_3+b_4$	$c_3+c_2+c_4$	$d_4+d_3+d_0+d_1+d_2$	$e_0+e_4+e_1+e_2+e_3$



GPU 0	$a_1+a_0+a_2+a_3+a_4$	$b_2+b_1+b_3+b_4+b_0$	$c_3+c_2+c_4+c_0$	$d_4+d_3+d_0+d_1+d_2$	$e_0+e_4+e_1+e_2+e_3$
GPU 1	$a_1+a_0+a_2+a_3+a_4$	$b_2+b_1+b_3+b_4+b_0$	$c_3+c_2+c_4+c_0+c_1$	$d_4+d_3+d_0+d_1$	$e_0+e_4+e_1+e_2+e_3$
GPU 2	$a_1+a_0+a_2+a_3+a_4$	$b_2+b_1+b_3+b_4+b_0$	$c_3+c_2+c_4+c_0+c_1$	$d_4+d_3+d_0+d_1+d_2$	$e_0+e_4+e_1+e_2$
GPU 3	$a_1+a_0+a_2+a_3$	$b_2+b_1+b_3+b_4+b_0$	$c_3+c_2+c_4+c_0+c_1$	$d_4+d_3+d_0+d_1+d_2$	$e_0+e_4+e_1+e_2+e_3$
GPU 4	$a_1+a_0+a_2+a_3+a_4$	$b_2+b_1+b_3+b_4$	$c_3+c_2+c_4+c_0+c_1$	$d_4+d_3+d_0+d_1+d_2$	$e_0+e_4+e_1+e_2+e_3$

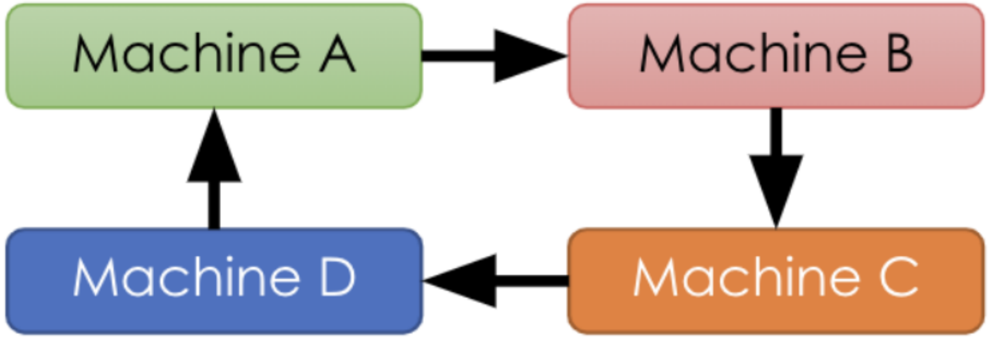


GPU 0	$a_1+a_0+a_2+a_3+a_4$	$b_2+b_1+b_3+b_4+b_0$	$c_3+c_2+c_4+c_0+c_1$	$d_4+d_3+d_0+d_1+d_2$	$e_0+e_4+e_1+e_2+e_3$
GPU 1	$a_1+a_0+a_2+a_3+a_4$	$b_2+b_1+b_3+b_4+b_0$	$c_3+c_2+c_4+c_0+c_1$	$d_4+d_3+d_0+d_1+d_2$	$e_0+e_4+e_1+e_2+e_3$
GPU 2	$a_1+a_0+a_2+a_3+a_4$	$b_2+b_1+b_3+b_4+b_0$	$c_3+c_2+c_4+c_0+c_1$	$d_4+d_3+d_0+d_1+d_2$	$e_0+e_4+e_1+e_2+e_3$
GPU 3	$a_1+a_0+a_2+a_3+a_4$	$b_2+b_1+b_3+b_4+b_0$	$c_3+c_2+c_4+c_0+c_1$	$d_4+d_3+d_0+d_1+d_2$	$e_0+e_4+e_1+e_2+e_3$
GPU 4	$a_1+a_0+a_2+a_3+a_4$	$b_2+b_1+b_3+b_4+b_0$	$c_3+c_2+c_4+c_0+c_1$	$d_4+d_3+d_0+d_1+d_2$	$e_0+e_4+e_1+e_2+e_3$

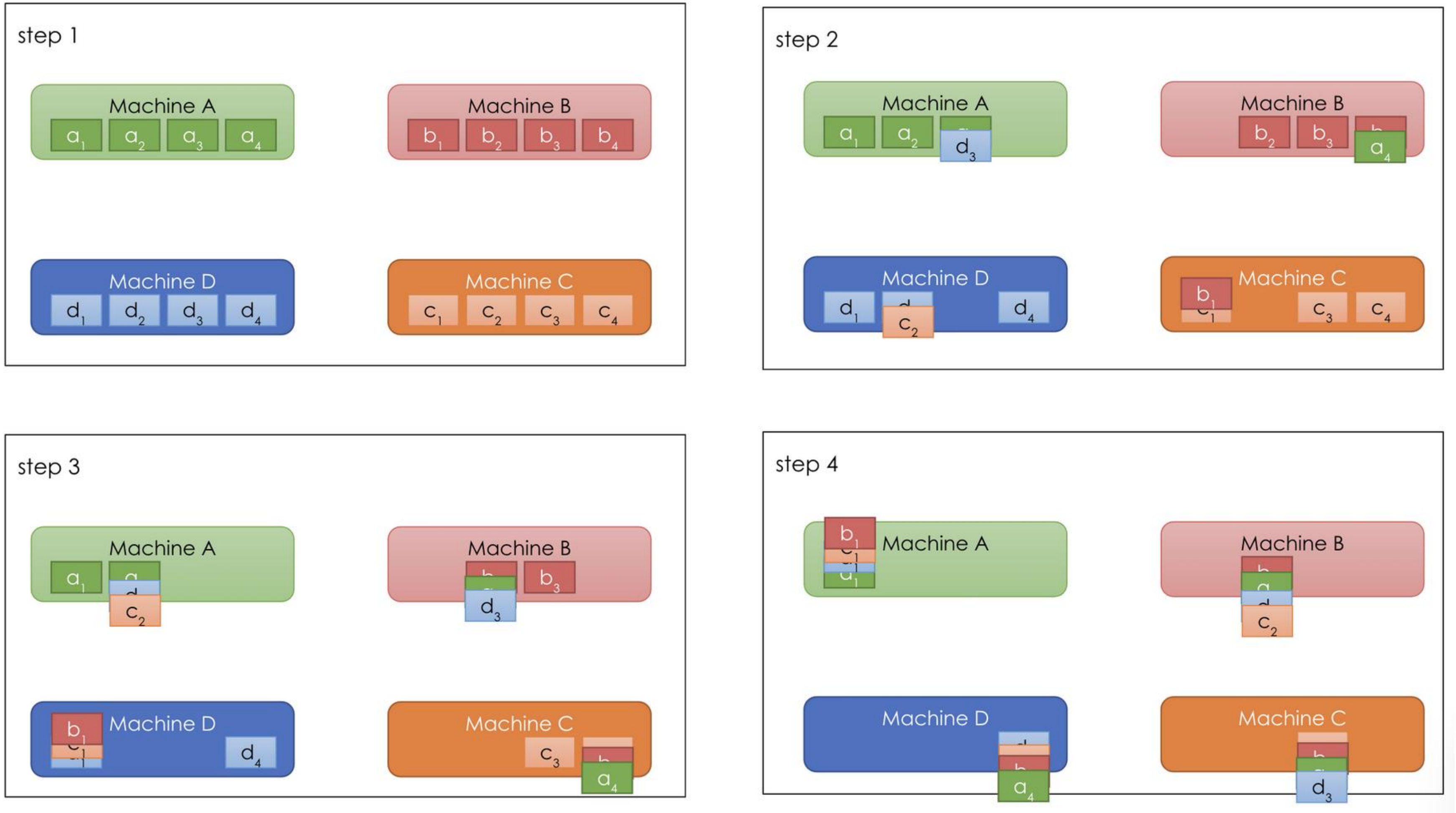
Data Transfers

- Each of the N GPUs send and receive:
  - N-1 times for the scatter-reduce
  - N-1 times for the allgather
  - Each time, the GPUs will send P / N values, where P is the total number of values in array being summed across the different GPUs.
- Data Transferred=$2(N-1)P/N$
- which, crucially, is independent of the number of GPUs.

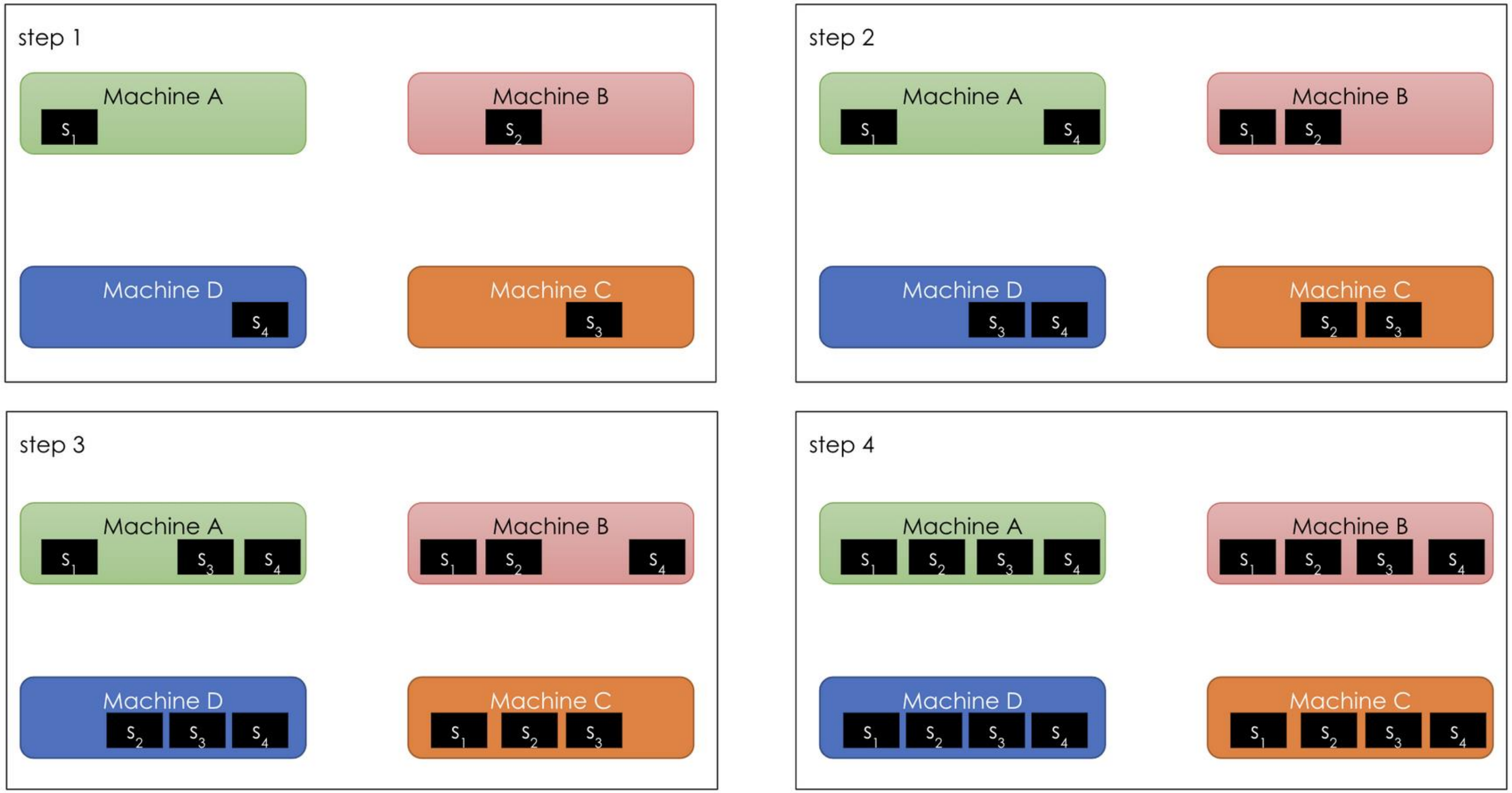
Ring AllReduce



Ring AllReduce: Scatter



Ring AllReduce: Gather



Ring AllReduce

- At each round, we have a constant bandwidth of P that does not depend on the total number of machines N , thus more scalable.
- The limiting factor in Ring AllReduce is the number of rounds of communication, which equals to N-1. As there are more machines, it may take longer for each cycle.

What if one cares about privacy?

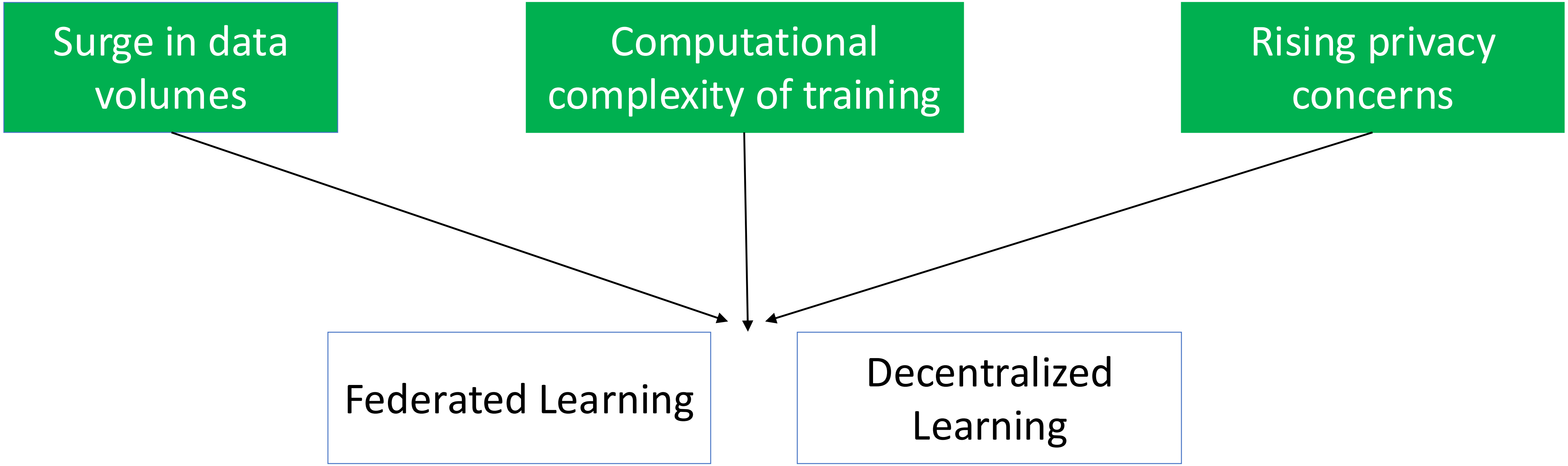
Everybody cares

- In 2016, Uber paid $148 million to settle the investigation on a data breach that exposed the personal information of over half a million drivers.
- In 2020, Google was fined $57 million for a GDPR violation
- Healthcare industry
- FinTech
- Autonomous cars
- Fraudulent behaviour in insurance
- IoT
- End users

Privacy: A rising concern

- Data born at the edge
- **Private data**: all the photos a user takes and everything they type on their mobile keyboard, including passwords, URLs, messages, etc.
- Data owned and processed by GAFAMs
- Users are more and more reluctant to share their data.

A shift towards distributed/decentralized learning



Surge in data volumes

Computational complexity of training

Rising privacy concerns

Federated Learning

Decentralized Learning

Basic Principle: Let the data stay where it is, learn by exchanging gradients/models

References

- Li *et al.* Scaling Distributed Machine Learning with the Parameter server. OSDI 2014
- Narayanan et al. PipeDream: generalized pipeline parallelism for DNN training. SOSP 2019
- Li *et al*. Parameter Server for distributed Machine Learning. Big Learning NIPS workshop, 2013
- McMahan, H. Brendan, Eider Moore, Daniel Ramage, and Seth Hampson. Communication-efficient learning of deep networks from decentralized data. AISTATS, 2017.
- Bonawitz *et al.* Towards Federated Learning at Scale: System Design. SysML 2019
- Karimireddy *et al*. SCAFFOLD: Stochastic Controlled Averaging for Federated Learning . ICML 2020
- Li et al. Federated Optimization in Heterogeneous networks. MLSys 2020
- Jianyu Wang, Qinghua Liu, Hao Liang, Gauri Joshi, and H. Vincent Poor. Tacking the objective inconsistency in heterogeneous federated optimization. In *Advances in Neural Information Processing Systems*, volume 33, pages 7611-7623, 2020.

Some slides courtesy of Aurick Qiao, Joseph Gonzalez, Wei Dai, and Jinliang Wei , Akash Dhasade
Some slides on Federated learning inspired/borrowed from Min Du
Some pictures from Ju Yang