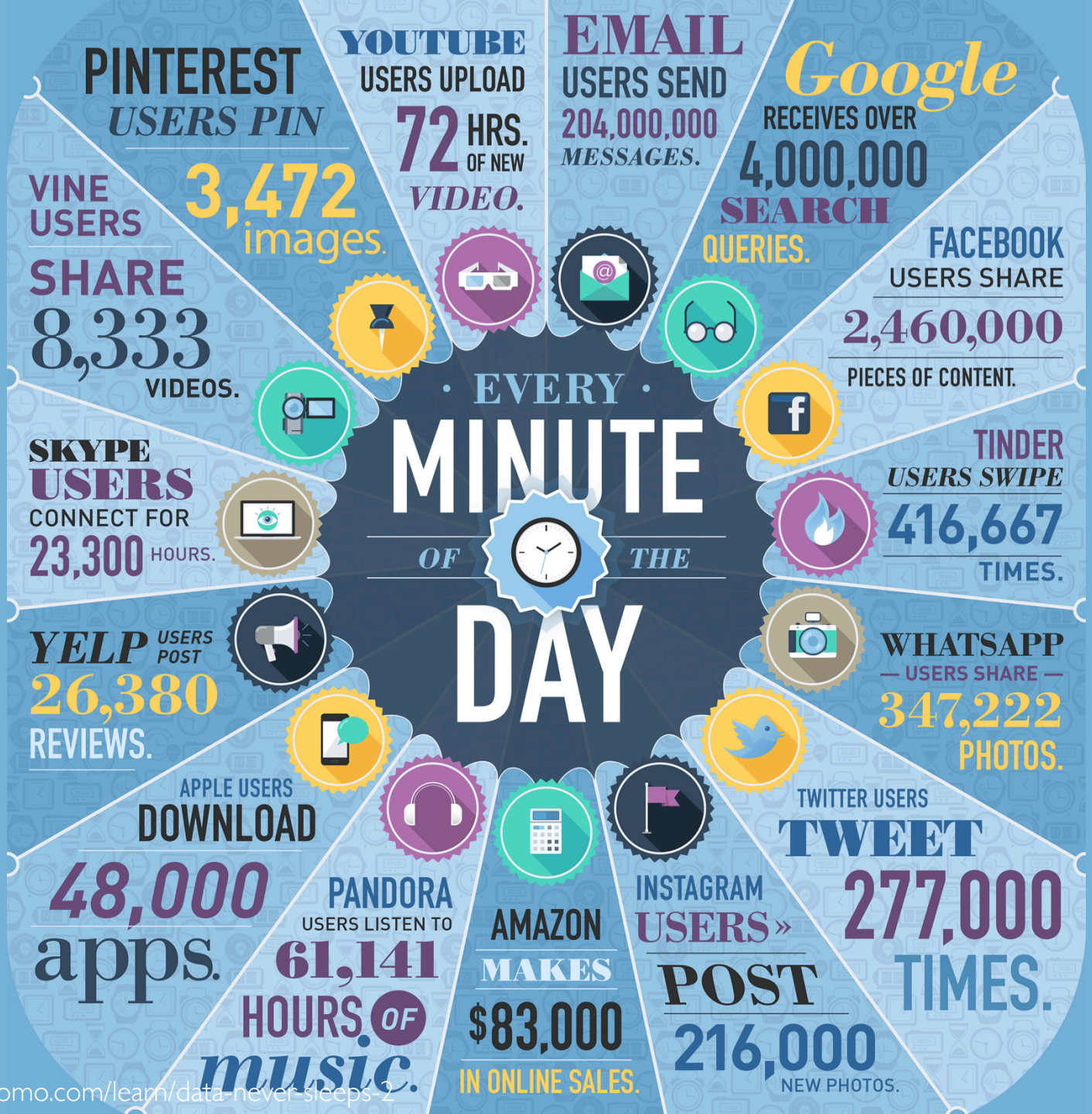
EMERGING SYSTEMS FOR LARGE-SCALE MACHINE LEARNING

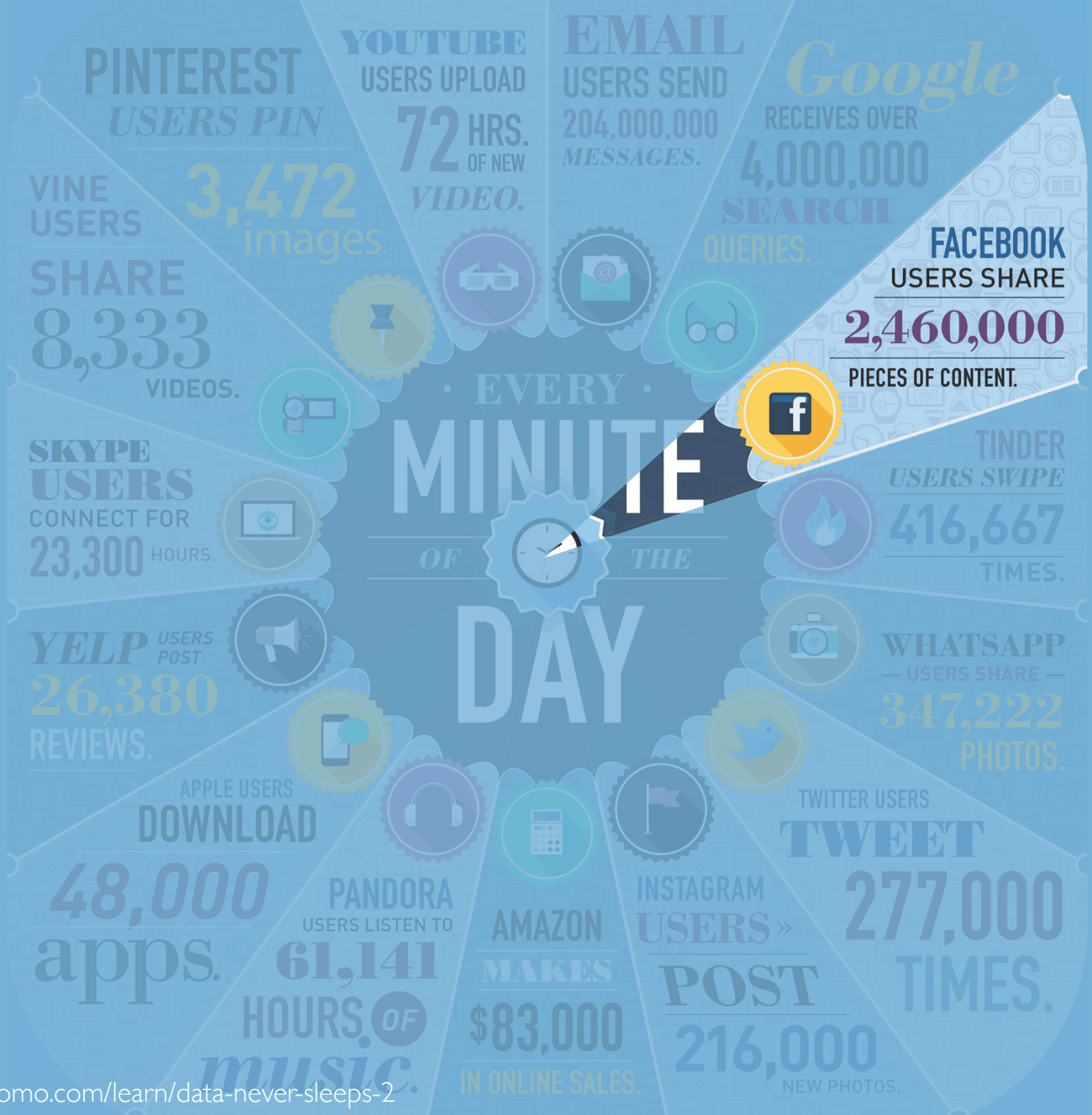
Joseph E. Gonzalez
Postdoc, UC Berkeley AMPLab
Co-founder, GraphLab Inc.
Ph.D. 2012, CMU
jegonzal@eecs.berkeley.edu

Slides (draft):
http://tinyurl.com/icml14-sysml

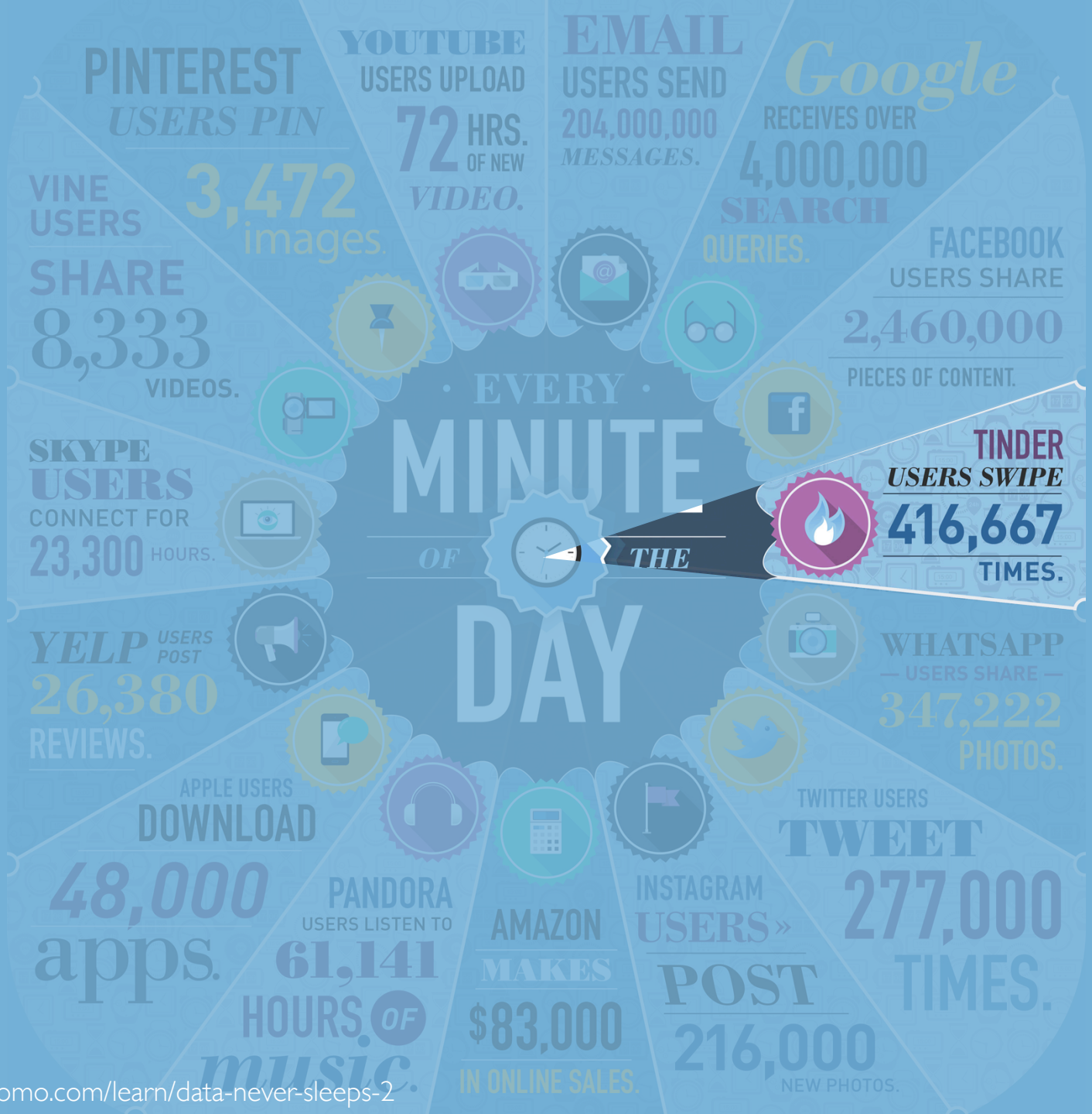
ICML'14 Tutorial



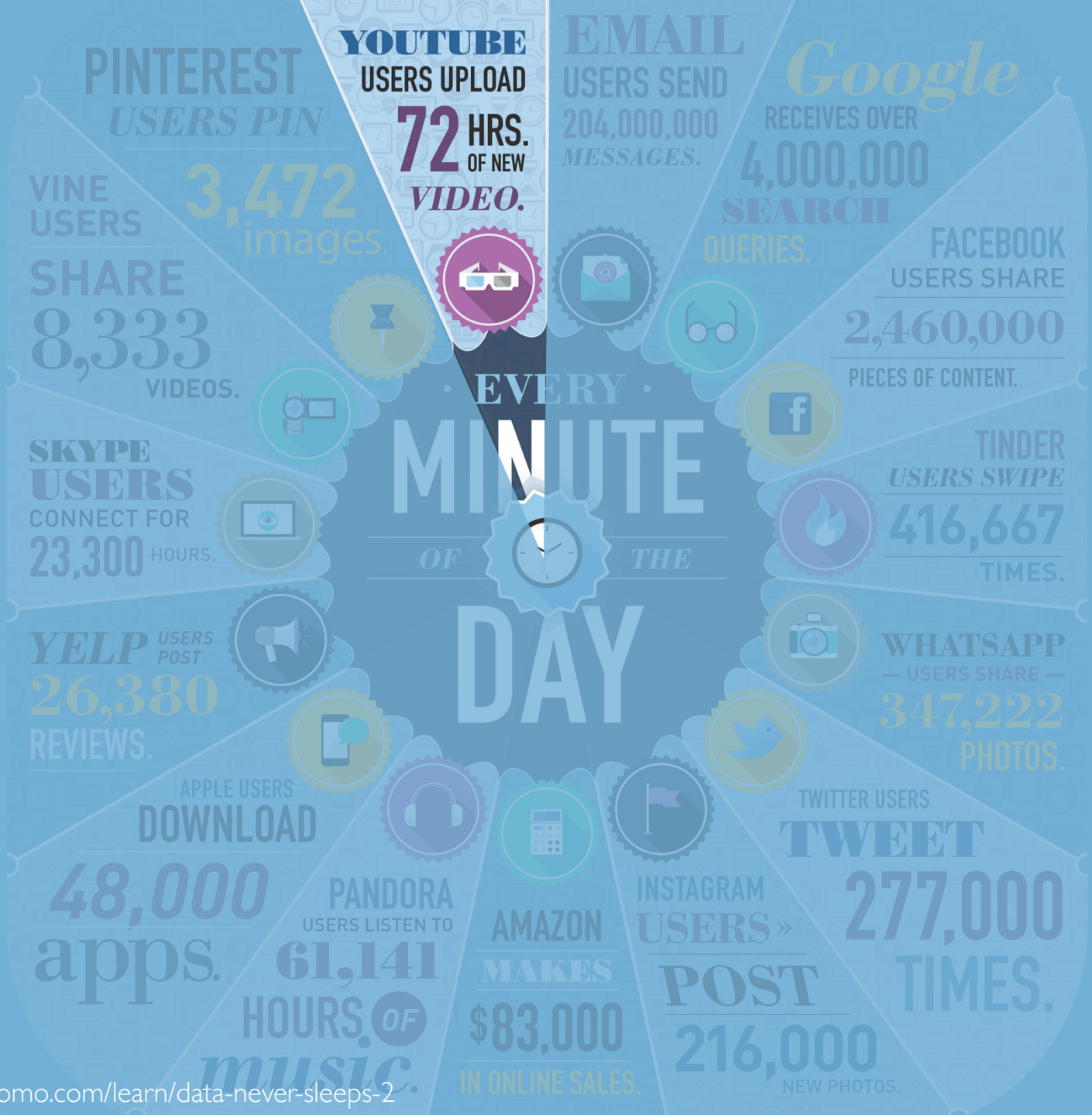
http://www.domo.com/learn/data-never-sleeps-2



http://www.domo.com/learn/data-never-sleeps-2



http://www.domo.com/learn/data-never-sleeps-2



http://www.domo.com/learn/data-never-sleeps-2

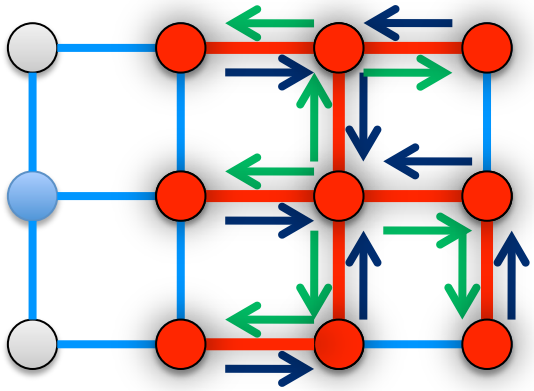
My story …

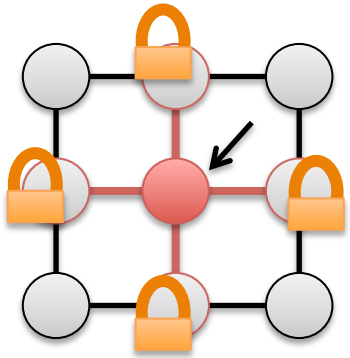
Machine Learning → Learning Systems

As a young graduate student

As a young graduate student I worked on parallel algorithms for inference in graphical models:

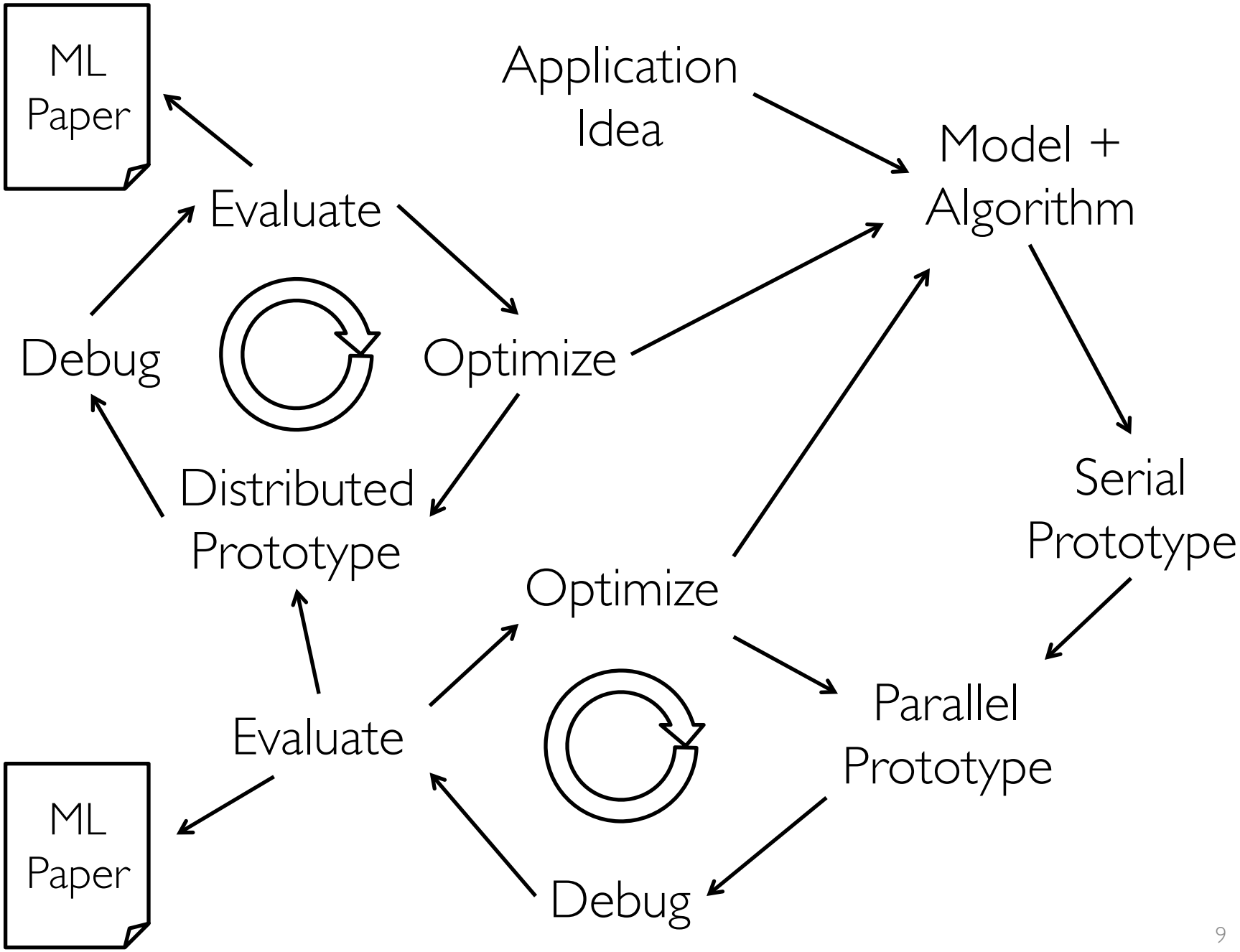
Belief Propagation





Gibbs Sampling

I designed and implemented parallel learning algorithms on top of low level primitives …


ML Paper
Application Idea
Model + Algorithm
Evaluate
Debug
Optimize
Distributed Prototype
Serial Prototype
Optimize
Parallel Prototype
Evaluate
ML Paper
Debug

Advantages of the Low-Level Approach

Extract maximum performance from hardware

Enable exploration of more complex algorithms

- Fine grained locking
- Atomic data-structures
- Distributed coordination protocols

My implementation is better than your implementation.

Limitations of the Low-Level Approach

Repeatedly address the same system challenges

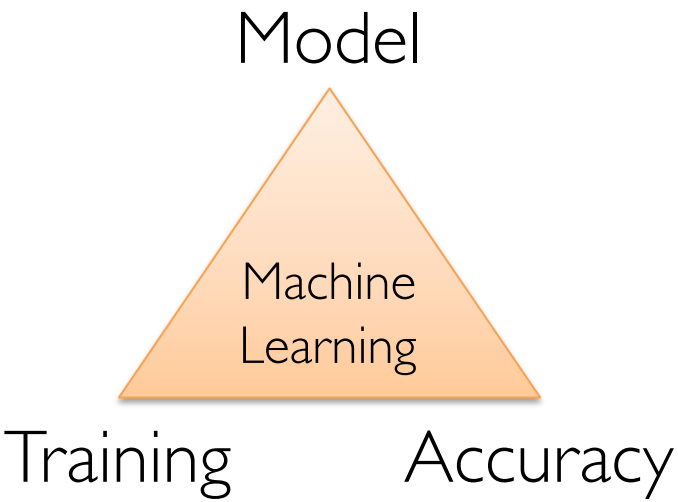
Algorithm conflates learning and system logic

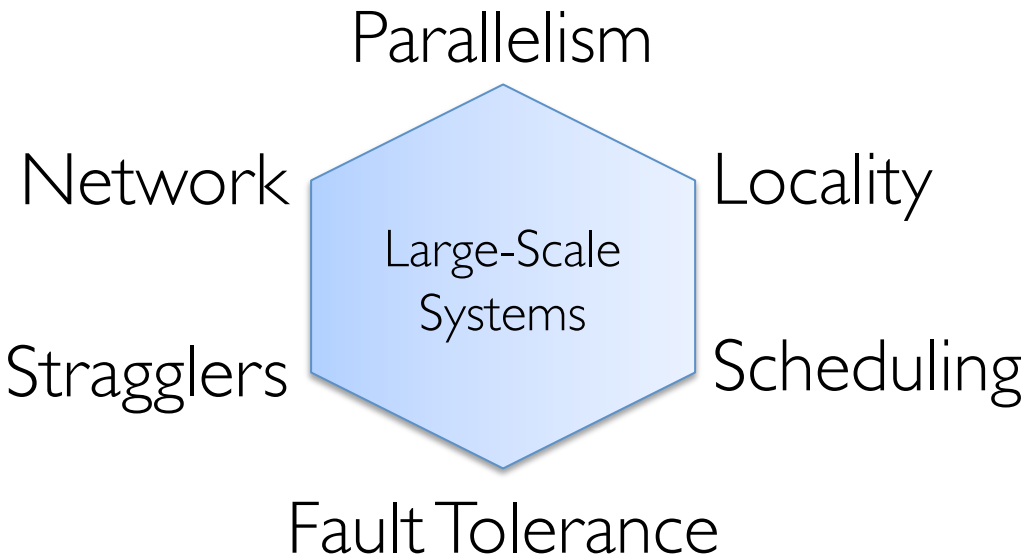
Difficult to debug and extend

Typically does not address issues at scale:
hardware failure, stragglers, …

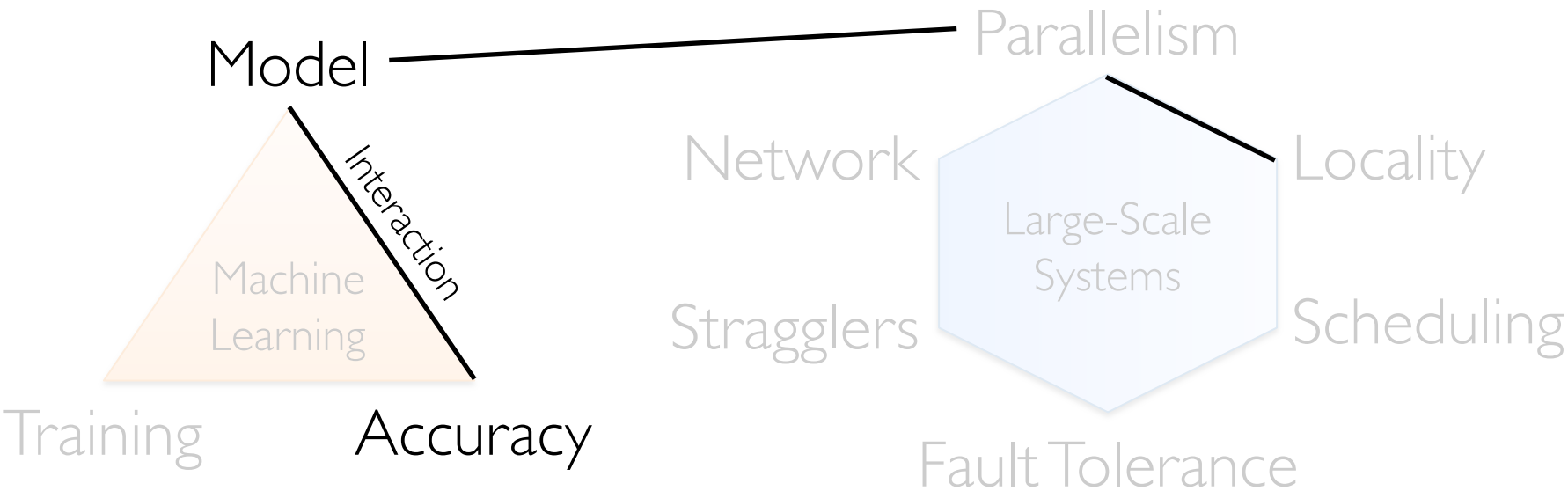
Months of tuning and engineering
for one problem.

Design Complexity

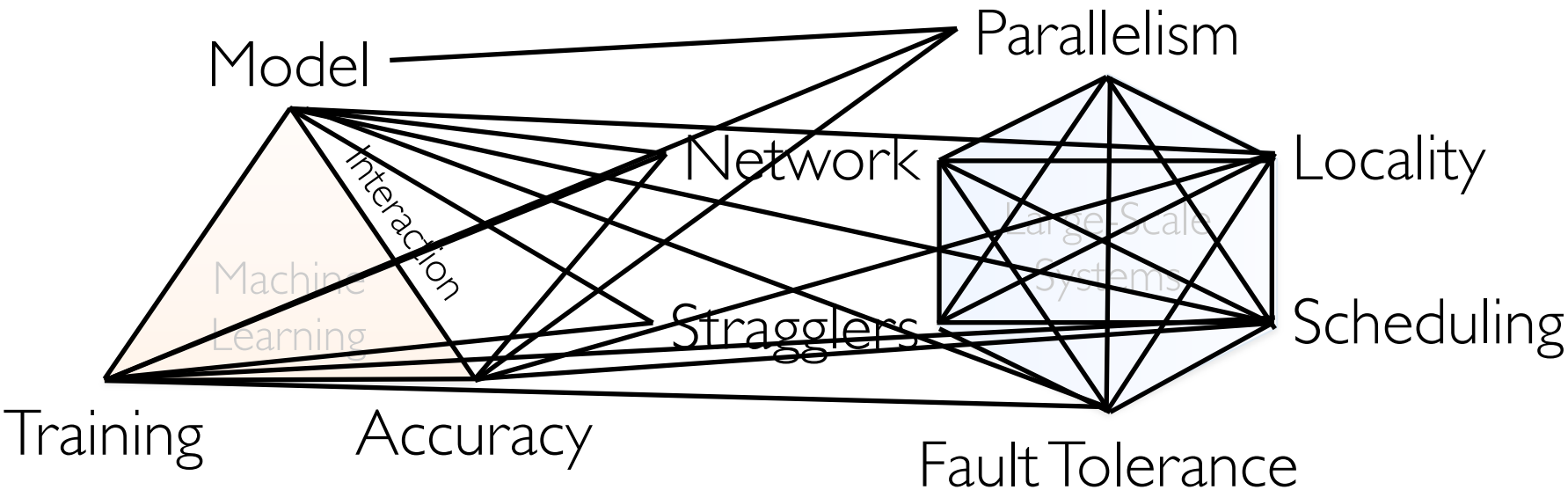




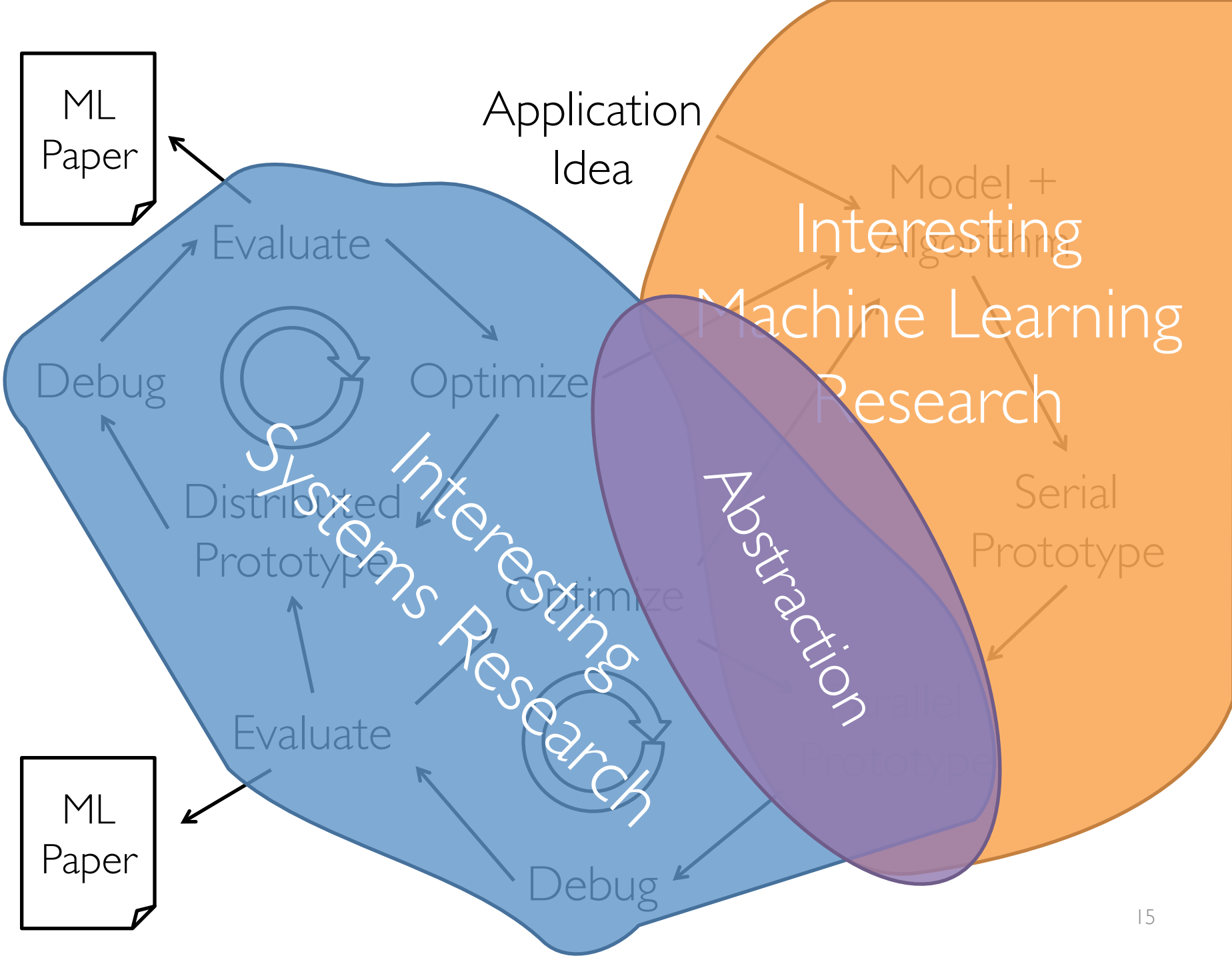
Design Complexity



Design Complexity



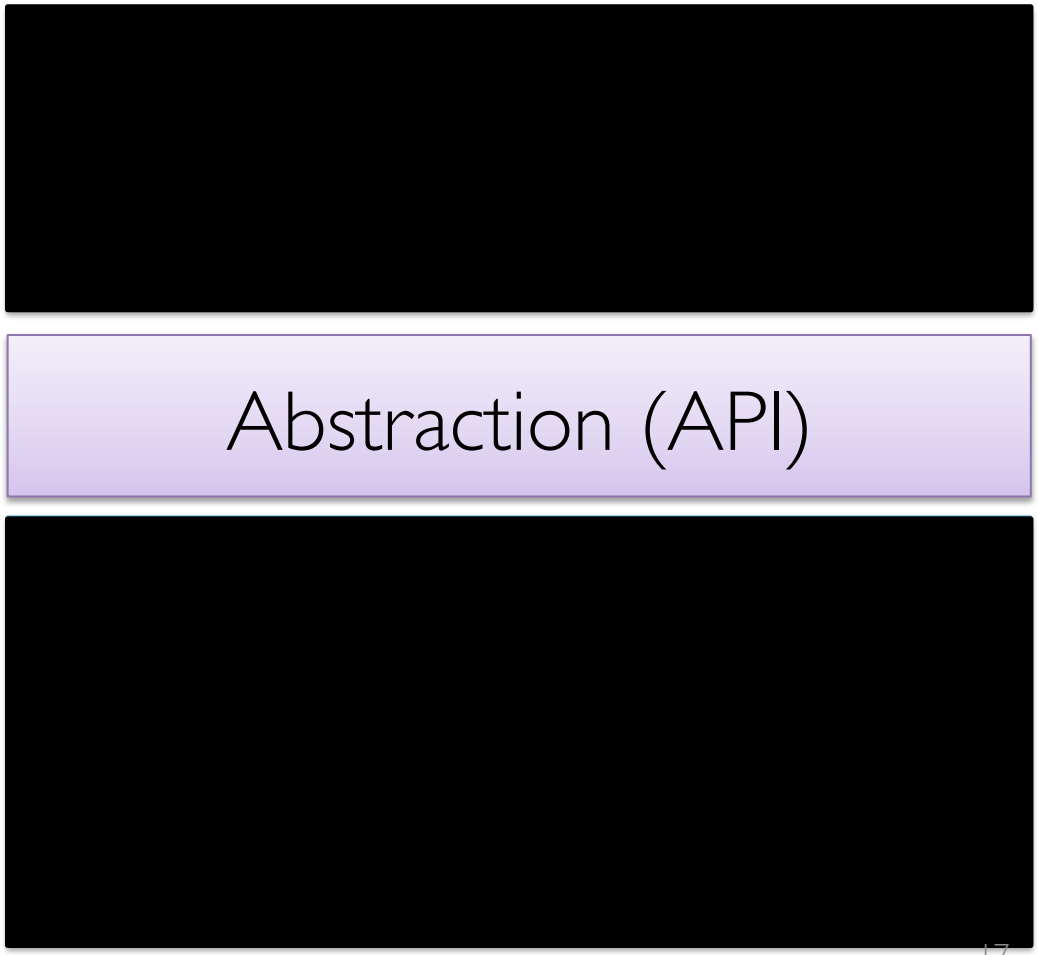
Learning systems combine the complexities of machine learning with system design


ML Paper
Application Idea
Evaluate
Debug
Optimize
Distributed Prototype
Evaluate
Debug
Optimize
Interesting Systems Research
Model + Algorithm
Serial Prototype
Parallel Prototype
Interesting Machine Learning Research
Abstraction
ML Paper

Black Box

Learning Systems

Black Box



Managing Complexity Through Abstraction

Identify common patterns

Define a narrow interface

Exploit limited abstraction to address system design challenges

Learning Algorithm

Common Patterns

Abstraction (API)

System

1. Parallelism
2. Data Locality
3. Network
4. Scheduling
5. Fault Tolerance
6. Stragglers

Junction Tree Inf. CoEM ALS Matrix Fact.

Common Pattern

Belief Propagation Gibb Sampling

→ Graph Parallel Abstraction →

GraphLab

The GraphLab project allowed us to:

- Separate algorithm and system design
- Optimize system for many applications at once
- Accelerate research in large-scale ML

Outline of the Tutorial

1. Distributed Aggregation: Map-Reduce
   Data Parallel
2. Iterative Machine Learning: Spark

3. Large Shared Models: Parameter Server
   Model Parallel

4. Graphical Computation: GraphLab to GraphX
   Graph Parallel

What is not covered

Linear Algebra Patterns: BLAS/ScaLAPACK

- core of high-performance computing
- communication avoiding & randomized algorithms
- Joel Tropp Tutorial (right now)

GPU Accelerated Systems

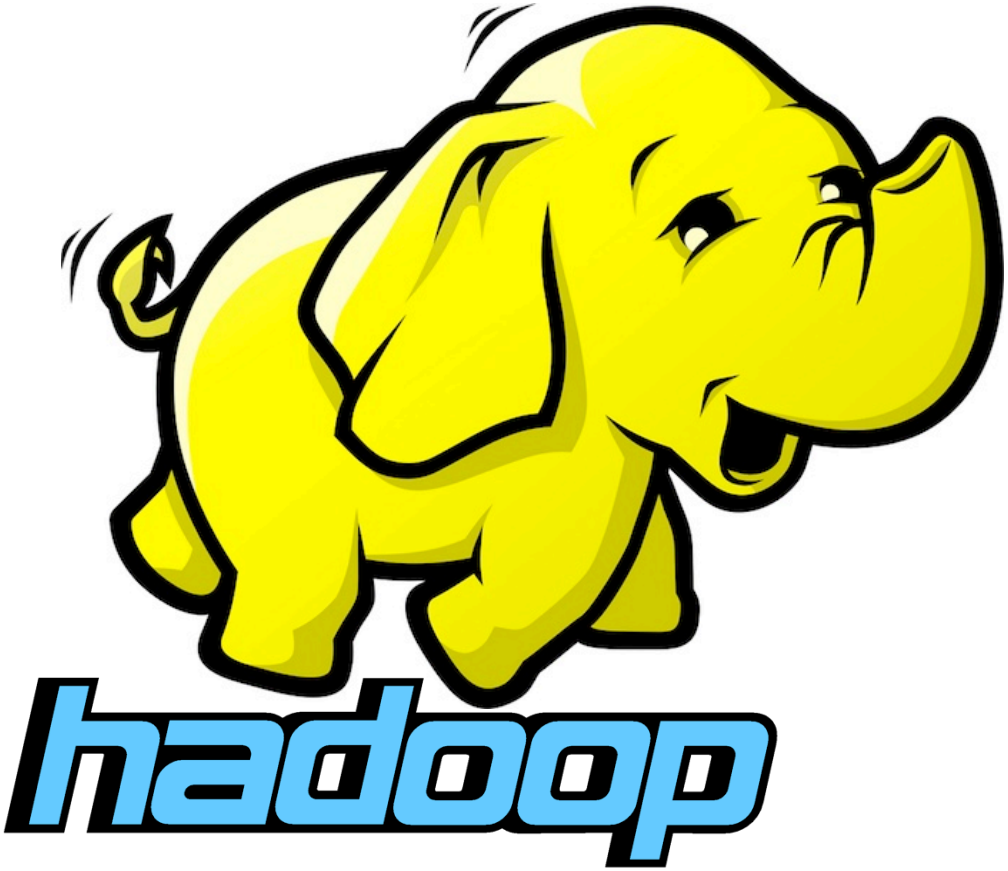
- converging to BLAS patterns

Probabilistic Programming

- See tutorial 5

Elephant in the Room

Map-Reduce



Aggregation Queries

Common Pattern

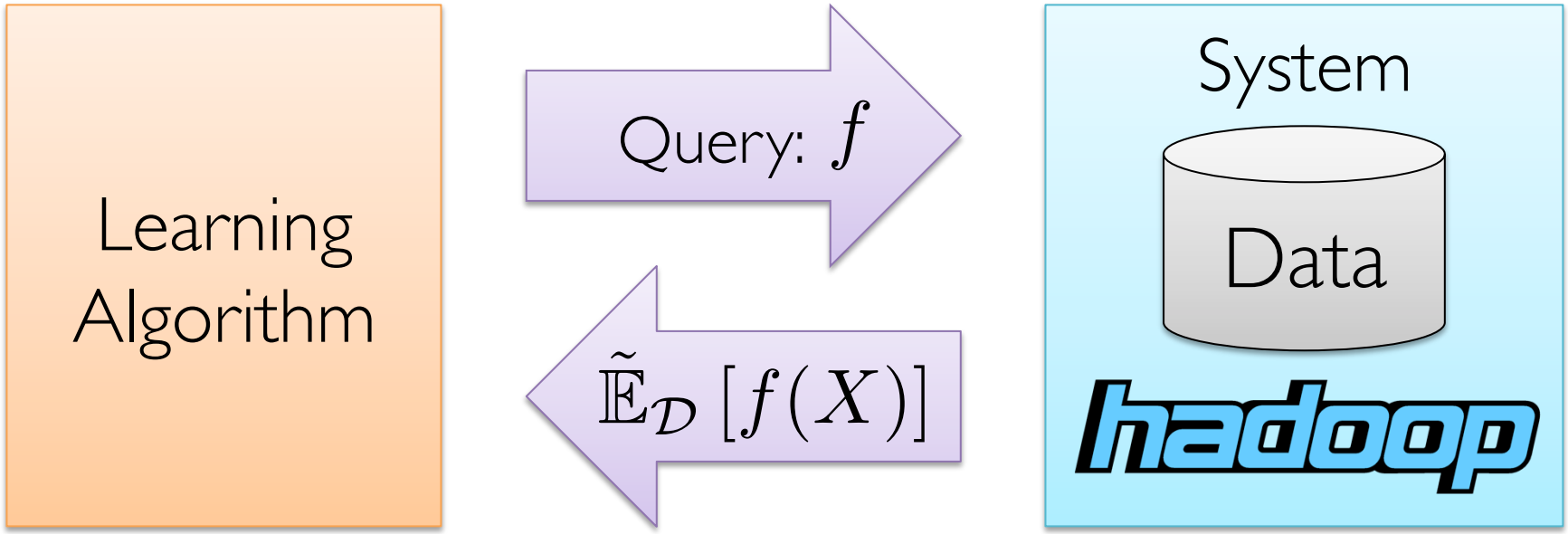
$$\tilde{\mathbb{E}}_{\mathcal{D}}\left[f(X)\right] = \frac{1}{n}\sum_{i=1}^{n} f(x_i)$$

Abstraction: Map, Reduce

System

hadoop

Learning from Aggregation Statistics



- D. Caragea et al., *A Framework for Learning from Distributed Data Using Sufficient Statistics and Its Application to Learning Decision Trees*. Int. J. Hybrid Intell. Syst. 2004
- Chu et al., *Map-Reduce for Machine Learning on Multicore*. NIPS'06.

Learning from Aggregation Statistics

Query Function:

$$f : \mathcal{X} \to \mathbb{R}^d$$

System Executes:

$$\tilde{\mathbb{E}}_{\mathcal{D}}\left[f(X)\right] = \frac{1}{n}\sum_{i=1}^{n} f(x_i)$$

on data $\mathcal{D} = \{x_1, \ldots, x_n\}$

Example Statistics

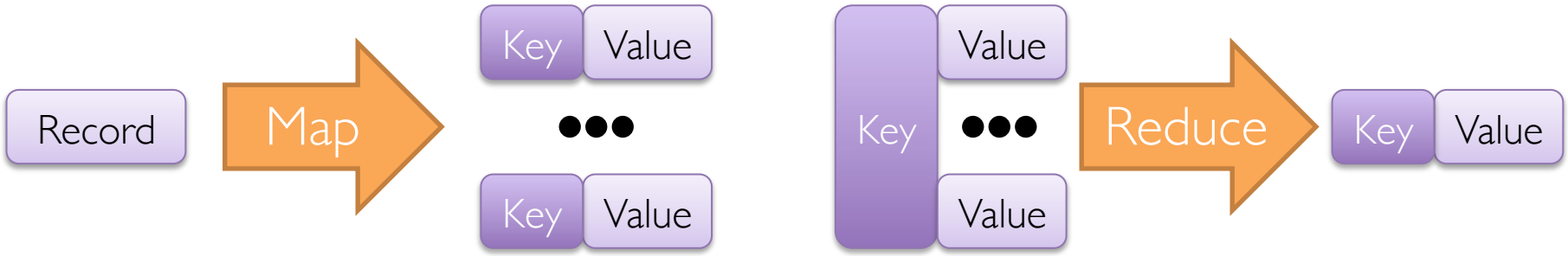
Sufficient Statistics (e.g., E[X], E[X^2]): $$\frac{1}{n}\sum_{i=1}^{n} x_i$$

Empirical loss: $$\frac{1}{n}\sum_{i=1}^{n} l(y, h(x))$$

Gradient of the loss: $$\frac{1}{n}\sum_{i=1}^{n} \nabla_w l(y, h_w(x))\Big|_{w=w^{(t)}}$$

Map-Reduce Abstraction

[Dean & Ghemawat, OSDI'04]



Example: *Word-Count*

```
Map(docRecord) {
  for (word in docRecord) {
    emit (word, 1)
  }
}
```

Key Value

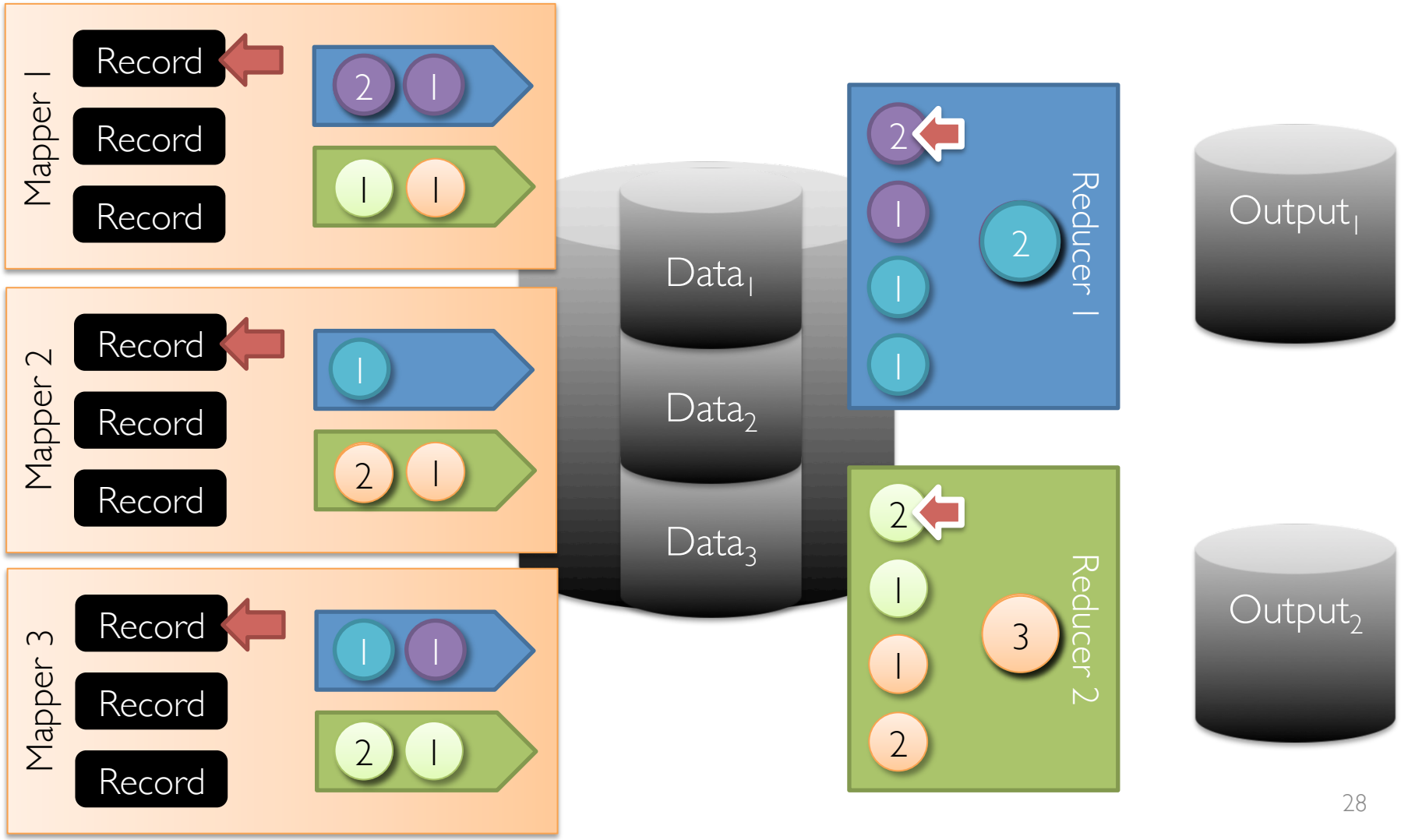
```
Reduce(word, counts) {
  emit (word, SUM(counts))
}
```

Map: Idempotent

Reduce: Commutative and Associative

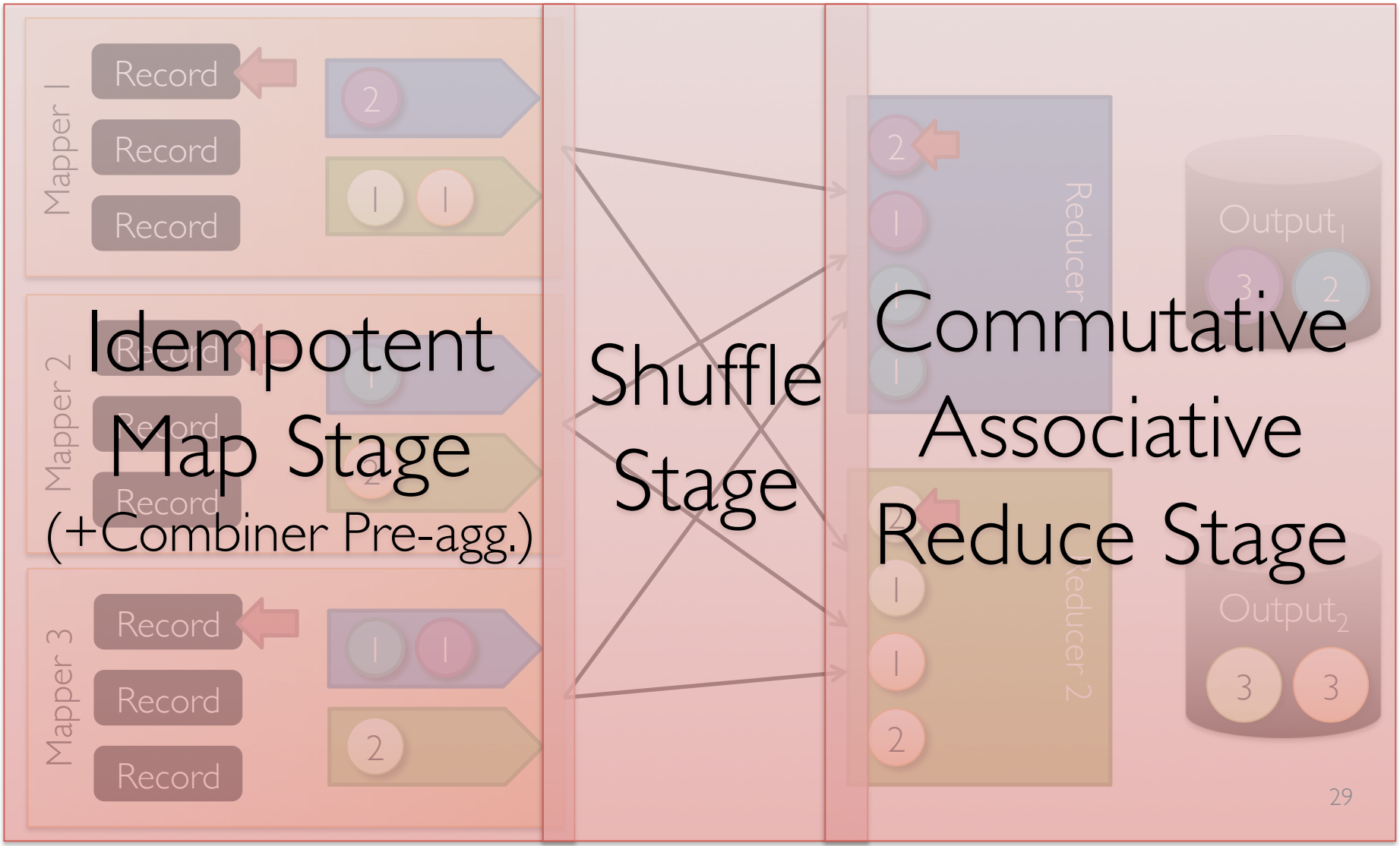
Map-Reduce System

[Dean & Ghemawat, OSDI'04]



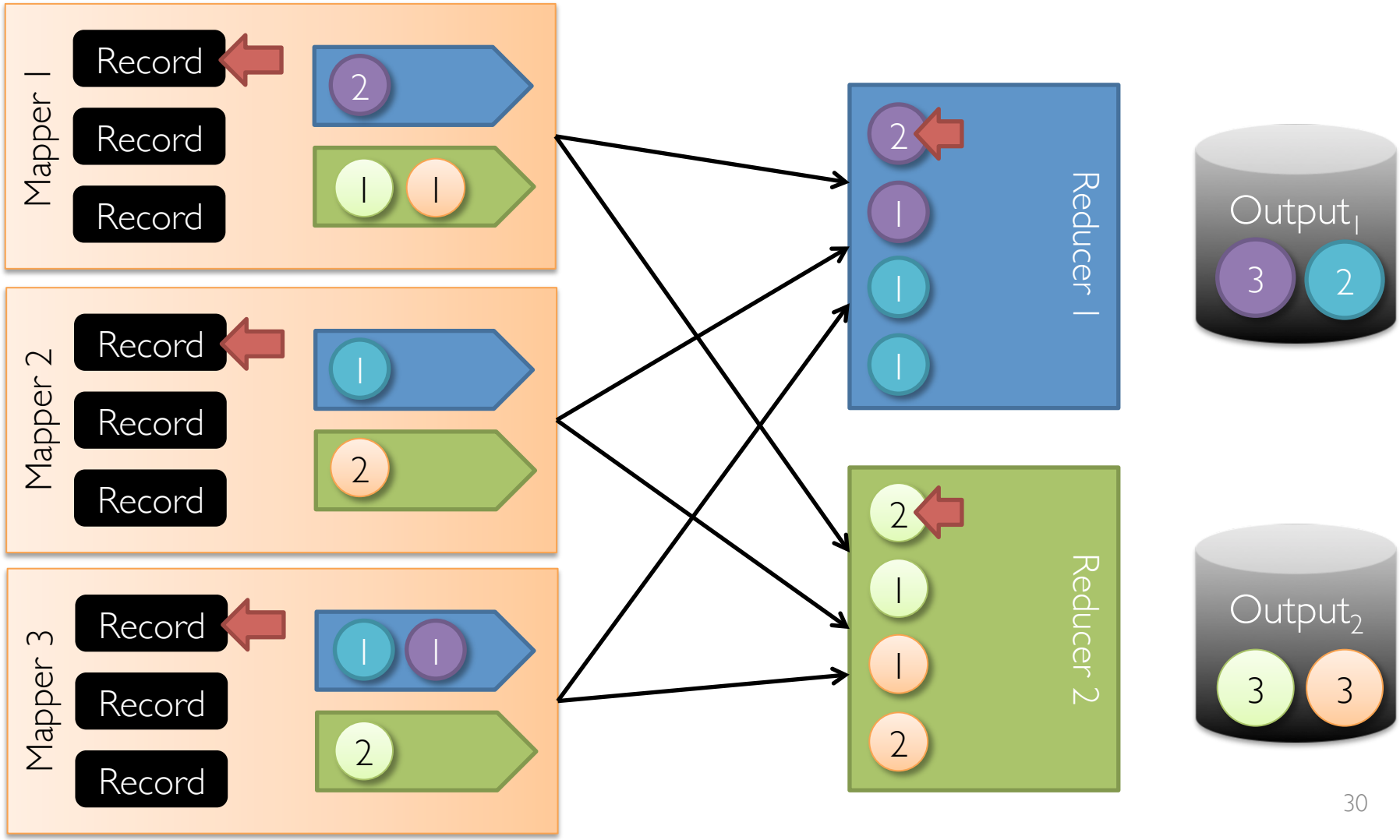
Map-Reduce System

[Dean & Ghemawat, OSDI'04]



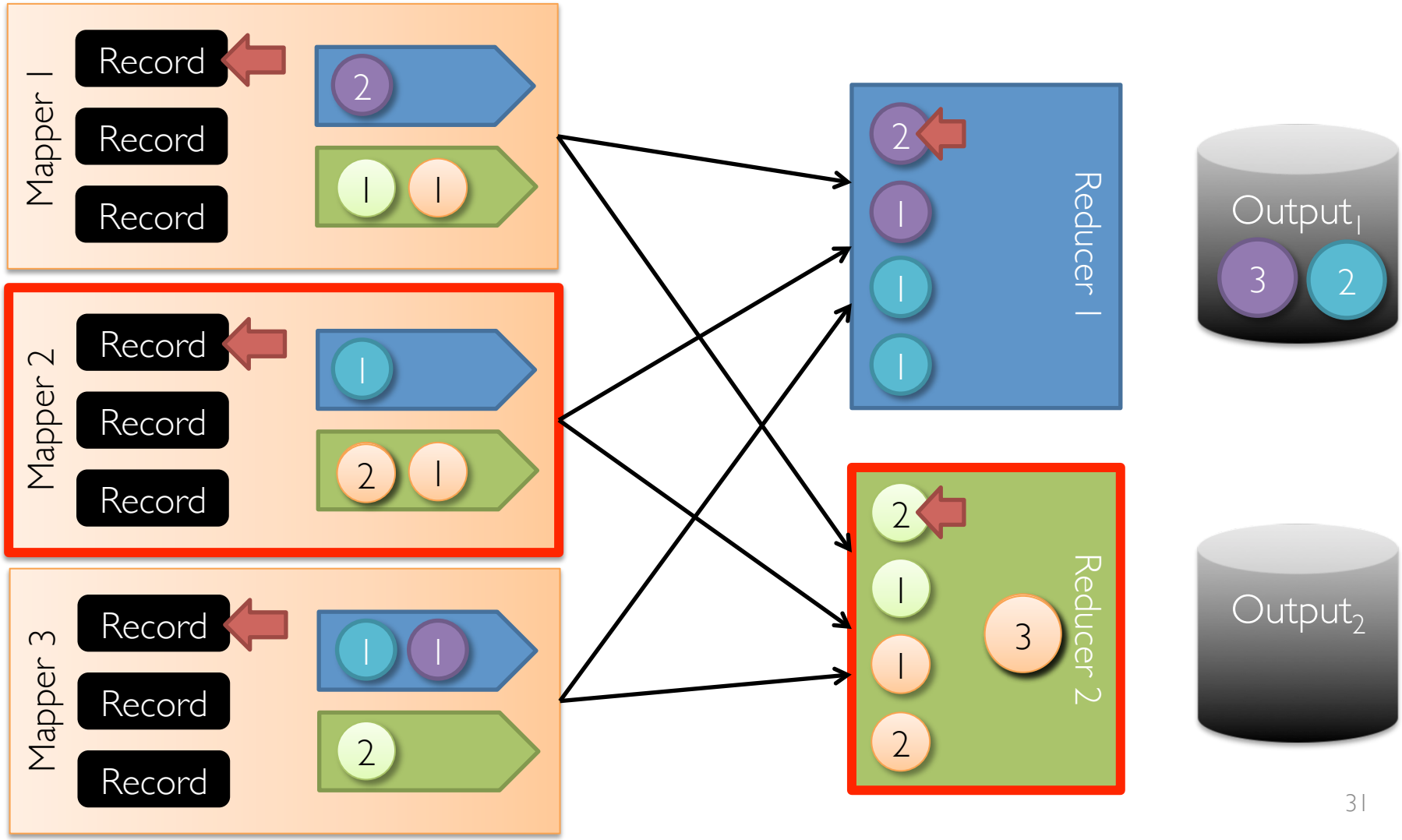
Map-Reduce Fault-Recovery

[Dean & Ghemawat, OSDI'04]



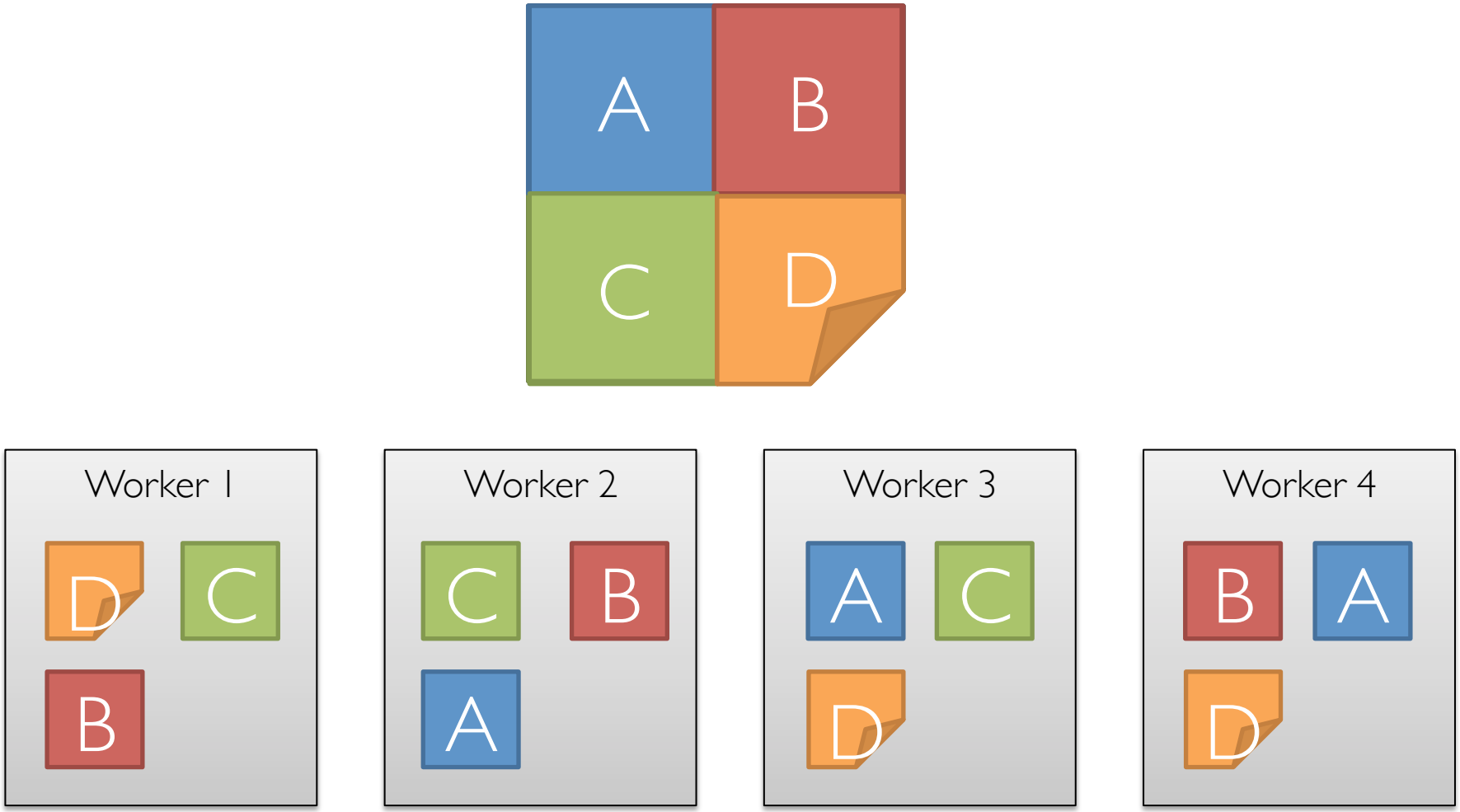
Map-Reduce Fault-Recovery

[Dean & Ghemawat, OSDI'04]



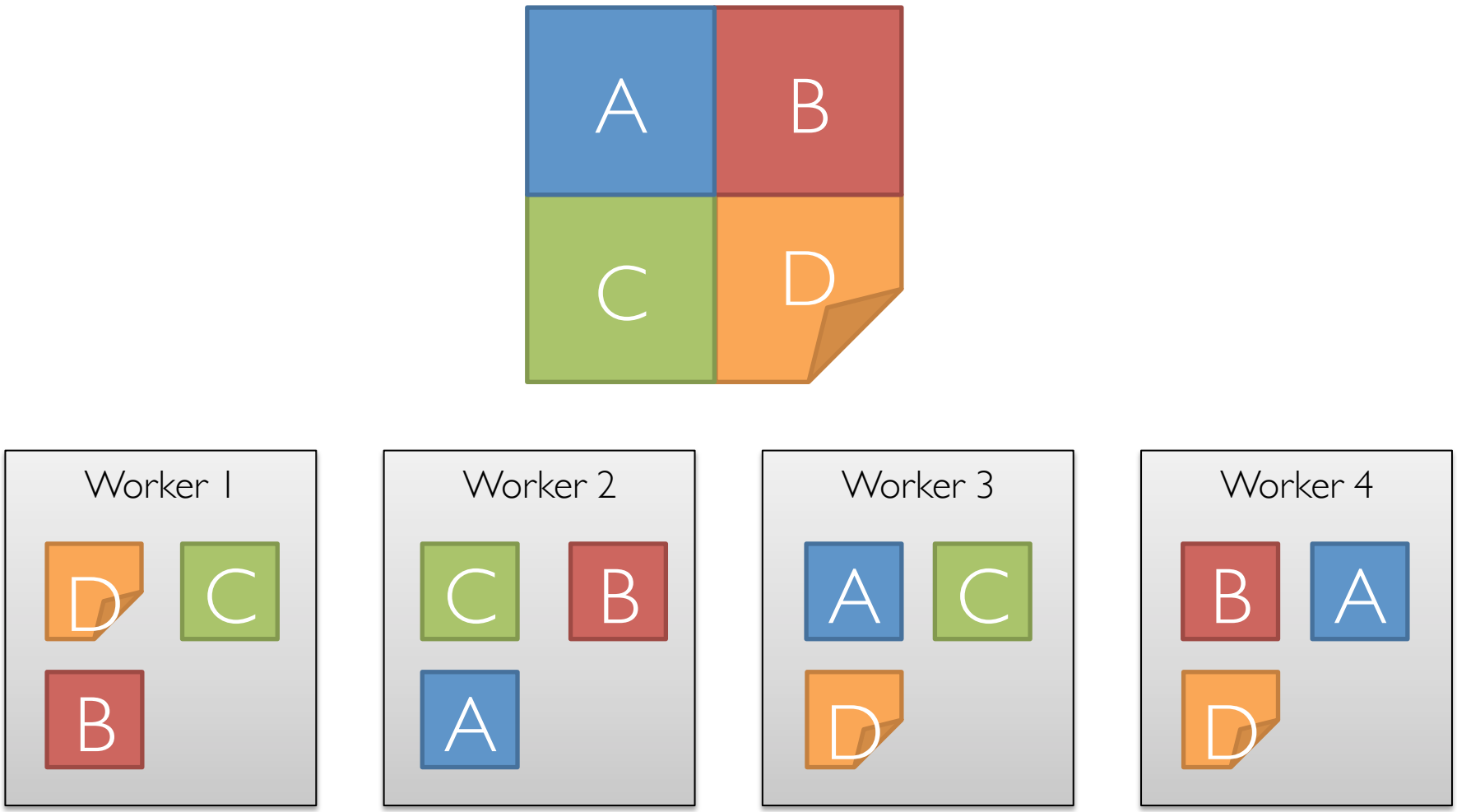
Distributed File Systems

[Ghemawat et al., SOSP'03]



Distributed File Systems

[Ghemawat et al., SOSP'03]



Important Systems Theme

What functionality can we remove?

Learning algorithm cannot directly access data.

- Restrict computation to:

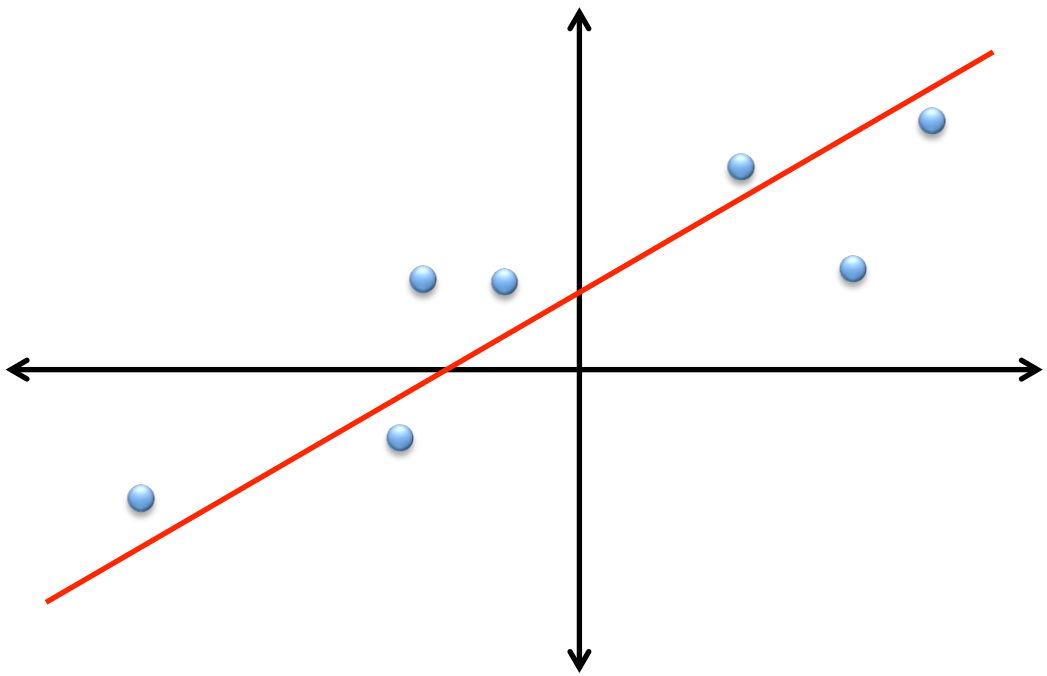
$$\tilde{\mathbb{E}}_{\mathcal{D}}\left[f(X)\right] = \frac{1}{n}\sum_{i=1}^{n} f(x_i)$$

System controls interaction with data:

- Distribute computation and data access
- Fault tolerance & straggler mitigation

Example:

Least Squares Regression



Least-Squares Regression with Aggregation Statistics

Objective:

Big

$$\hat{\theta}_{\text{MLE}} = \arg\min_{\theta \in \mathbb{R}^d} \sum_{i=1}^{n} (y_i - \theta^T x_i)^2$$

Small

Solution (Normal Equations):

$$\hat{\theta}_{\text{MLE}} = (X^T X)^{-1} (X^T Y)$$

Deriving the Aggregation Stats.

$$\hat{\theta}_{\text{MLE}} = (X^T X)^{-1} (X^T Y)$$

Aggregation Statistics:

$$\hat{\theta}_{\text{MLE}} = \left(\sum_{i=1}^{n} \underbrace{x_i x_i^T}_{f_1} \right)^{-1} \left(\sum_{i=1}^{n} \underbrace{x_i y_i}_{f_2} \right)$$

$$\frac{O(nd^2)}{\#\text{mappers}}$$

```
Map( (x,y) record ) {
  emit ("xx", x * Trans(x))
  emit ("xy", x * y)
}
```

```
Reduce(key, mats) {
  emit (key, SUM(mats))
}
```

$$\theta^*_{\text{MLE}} = (X^T X)^{-1} (X^T Y)$$

Deriving the Aggregation Stats.

$$\hat{\theta}_{\text{MLE}} = (X^T X)^{-1} (X^T Y)$$

Aggregation Statistics:

$$\hat{\theta}_{\text{MLE}} = \left(\sum_{i=1}^{n} \underbrace{x_i x_i^T}_{f_1} \right)^{-1} \left(\sum_{i=1}^{n} \underbrace{x_i y_i}_{f_2} \right)$$

$$\frac{O(nd^2)}{\#\text{mappers}}$$

Solve linear system on the master:

$$\hat{\theta}_{\text{MLE}} = \left(\begin{array}{c} d \\ \square \\ d \end{array} \right)^{-1} \left(\begin{array}{c} d \\ \square \\ 1 \end{array} \right) = \begin{array}{c} d \\ \square \\ 1 \end{array}$$

Inversion doesn't depend on *n*

$$O(d^3)$$

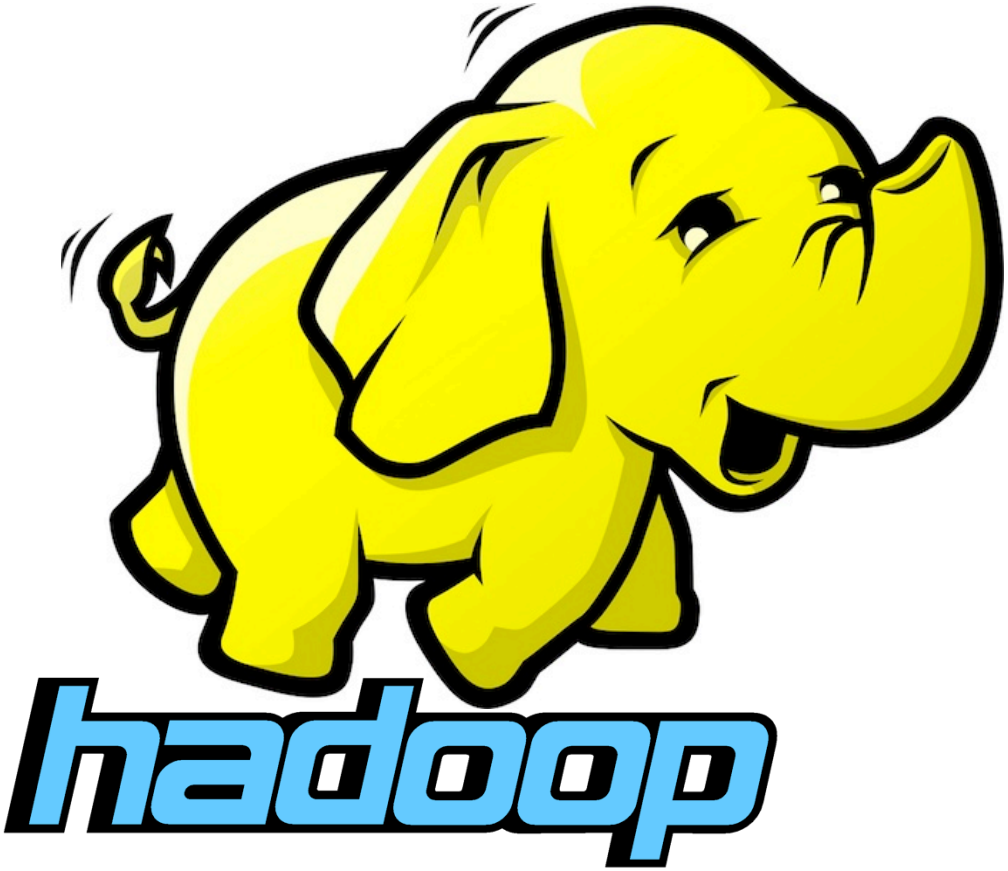
Apache Mahout

Open-Source Library of Algorithms on Hadoop

ALS Matrix Fact.	Naïve Bayes
SVD	PCA
Random Forests	Spectral Clustering
LDA	Canopy Clustering
K-Means	~~Logistic Regression?~~

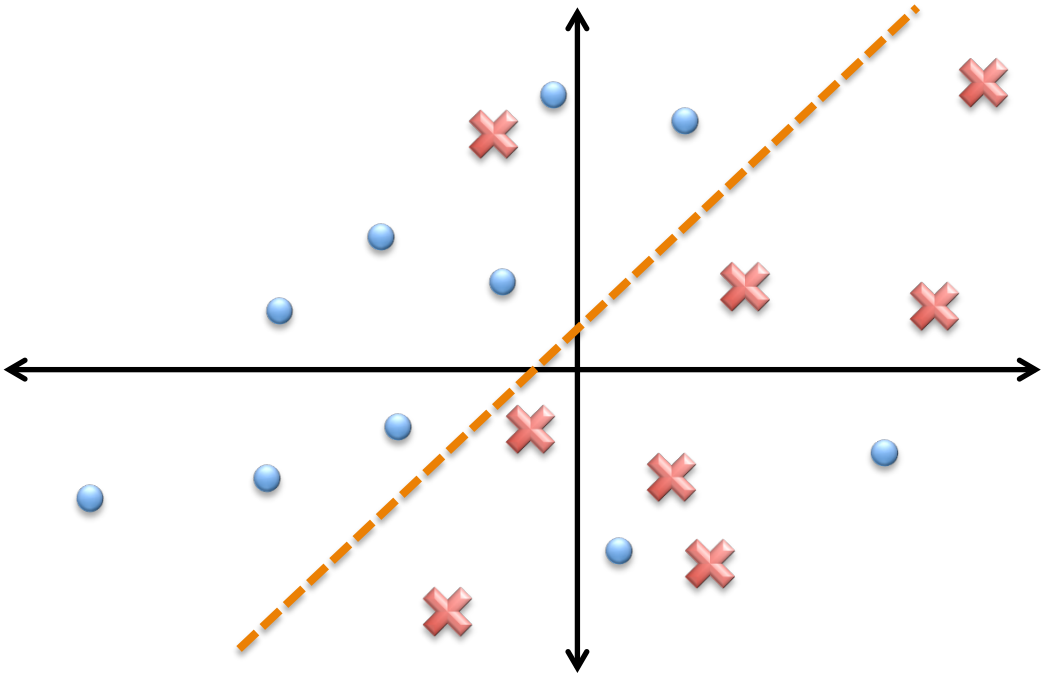
Limitations?

Map-Reduce



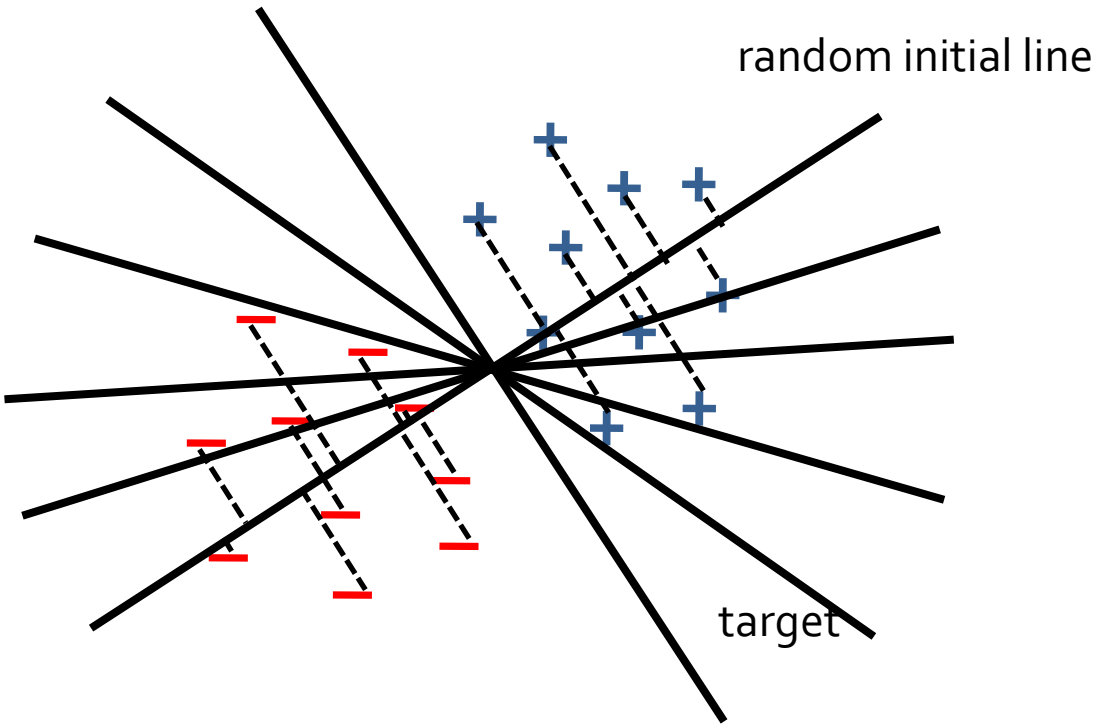
Why not

Logistic Regression?



Logistic Regression

Iterative batch gradient descent.

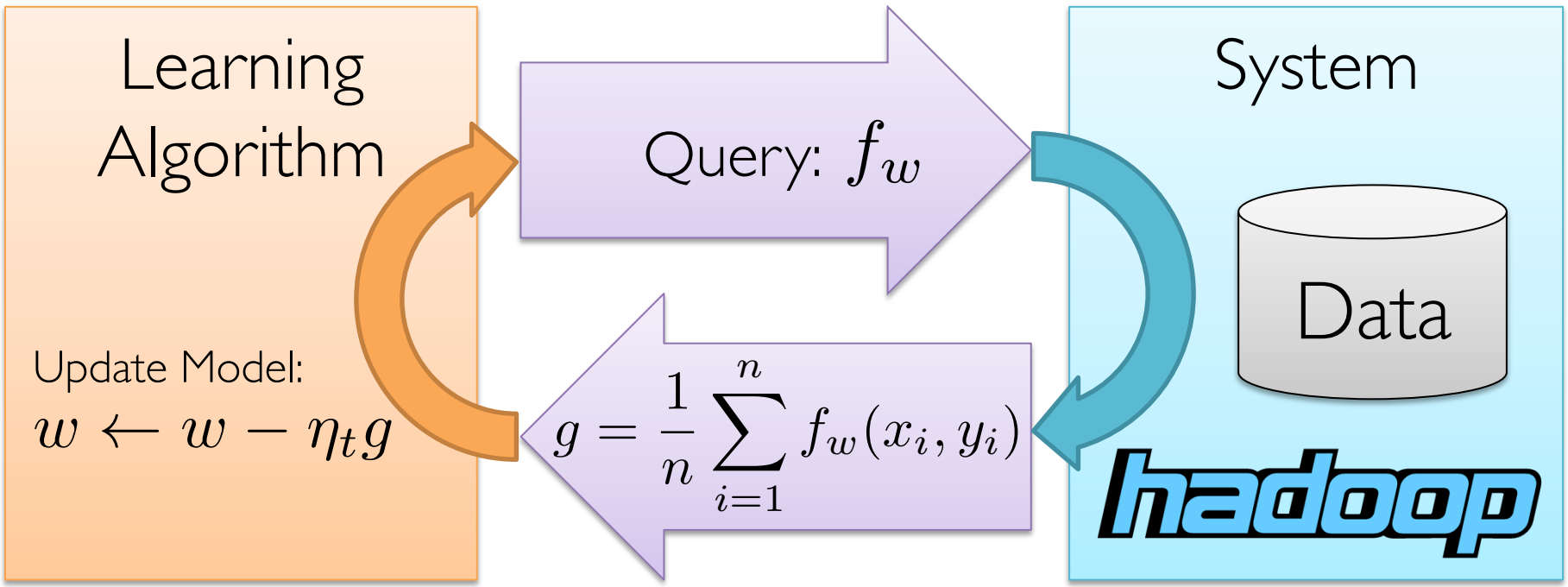


Slide provided by M. Zaharia

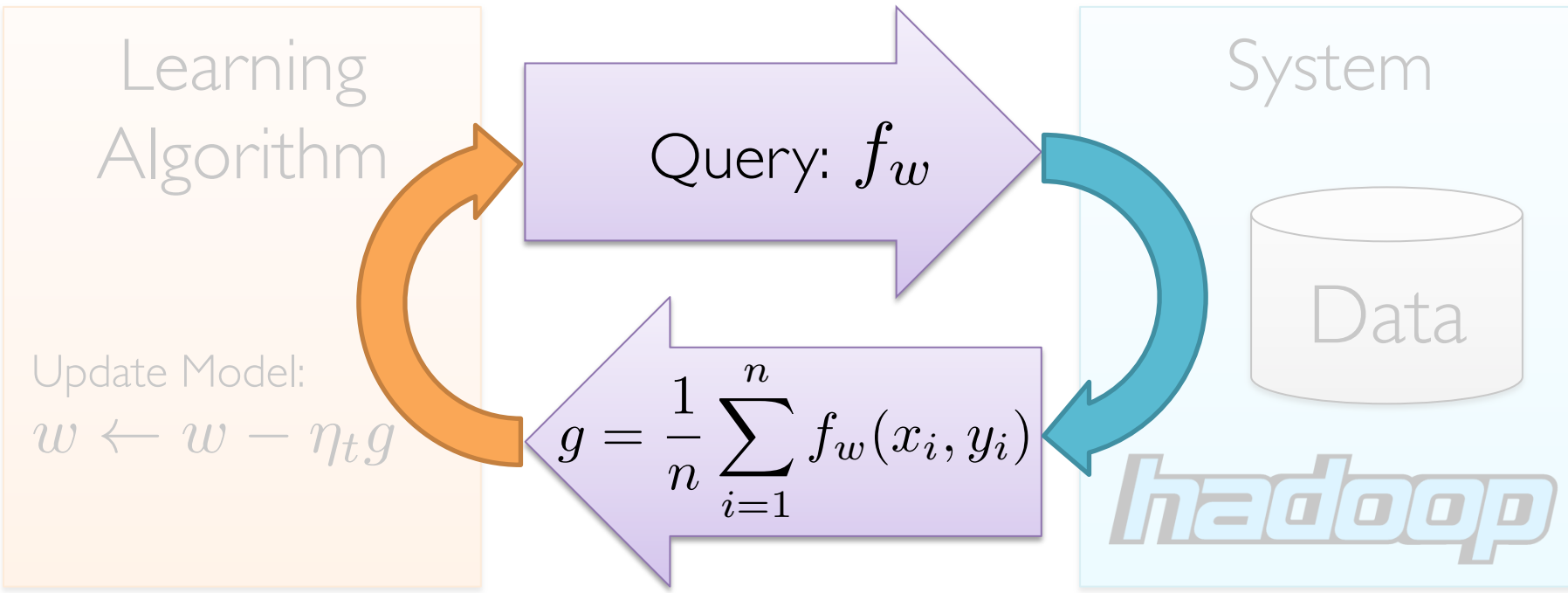
Logistic Regression in Map-Reduce

Gradient descent:

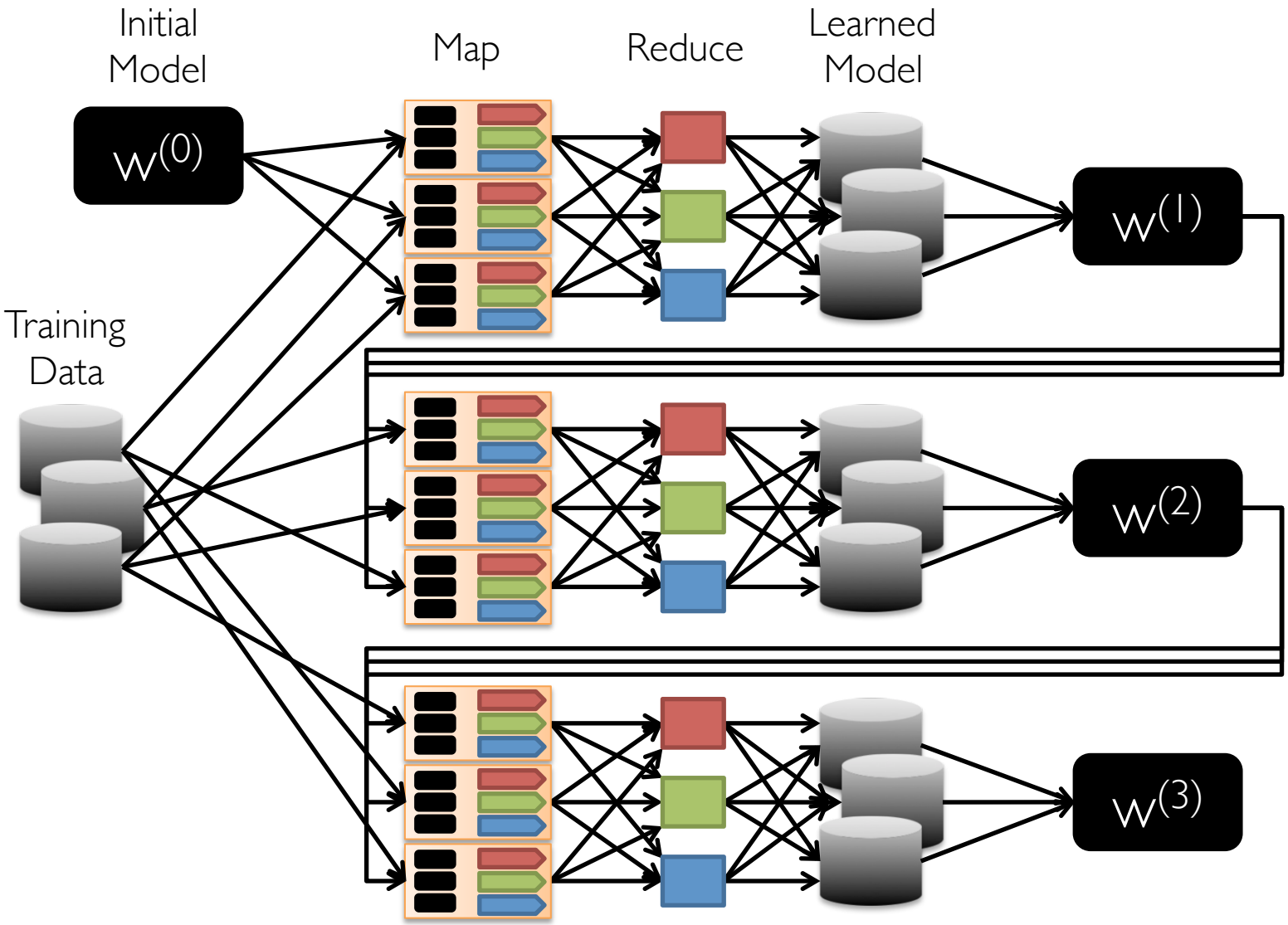
$$f_w(x, y) = \nabla \log L(y, h_w(x))$$



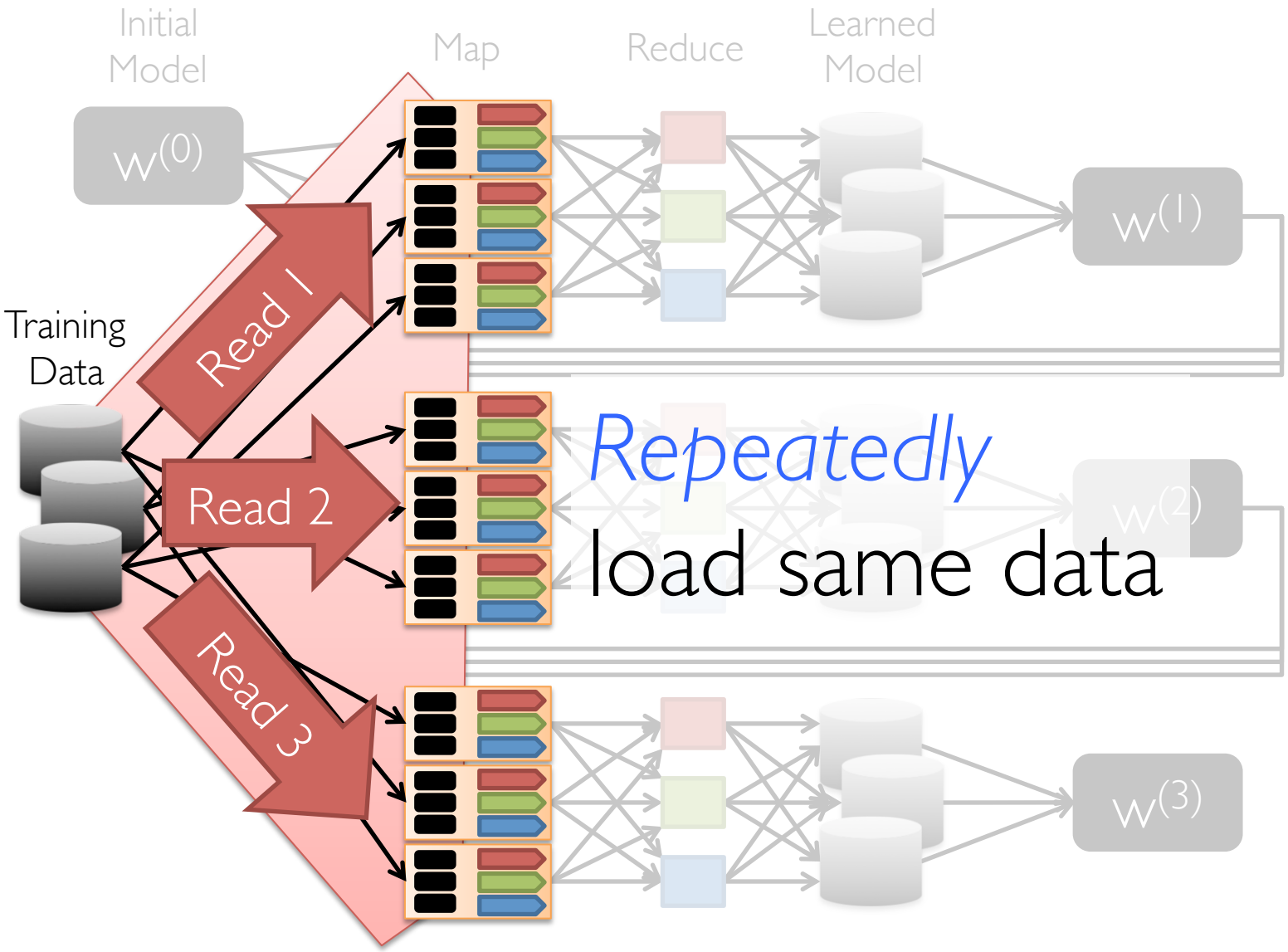
Map-Reduce is not optimized for iteration and multi-stage computation



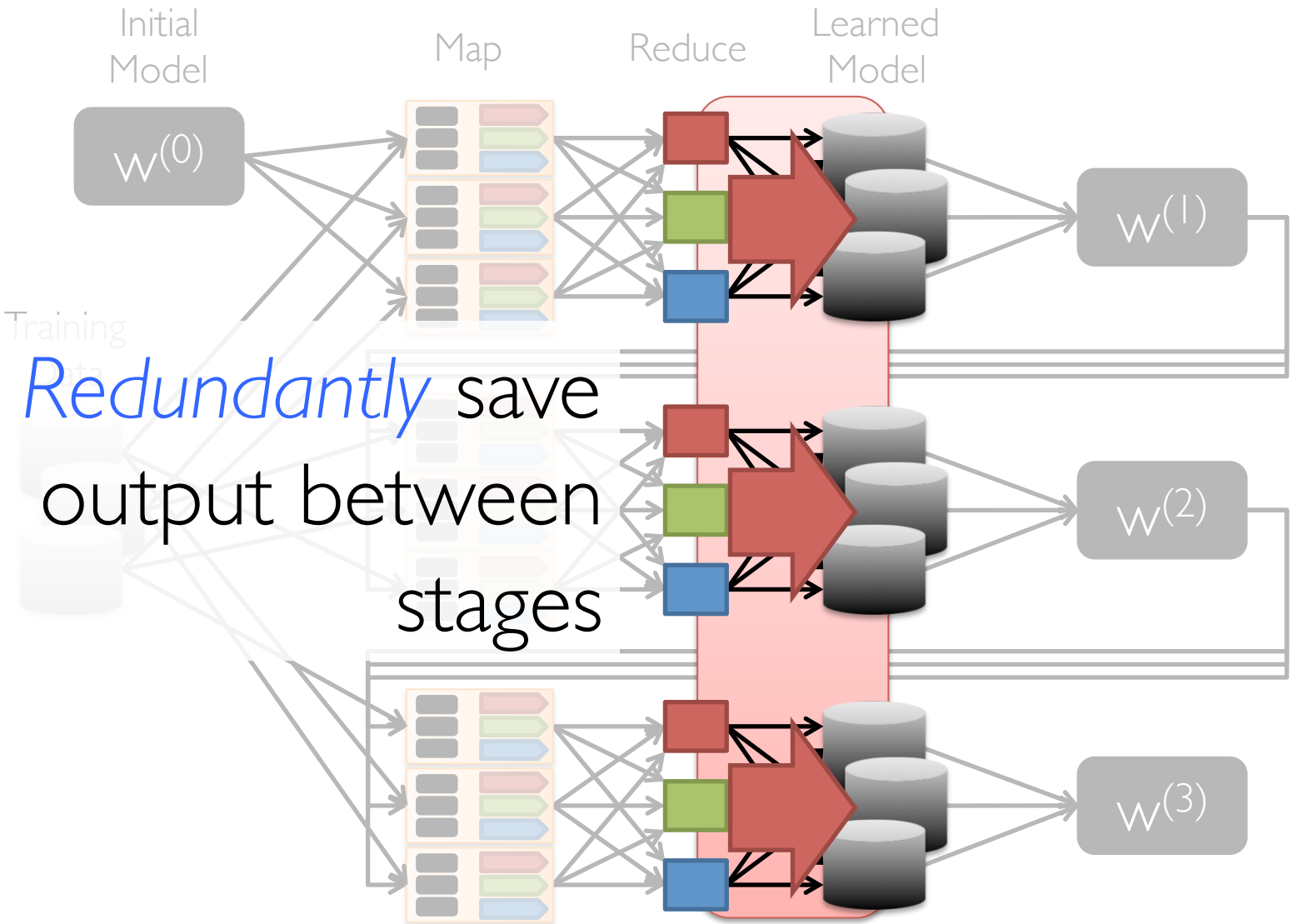
Iteration in Map-Reduce



Cost of Iteration in Map-Reduce



Cost of Iteration in Map-Reduce



Redundantly save output between stages

Spark

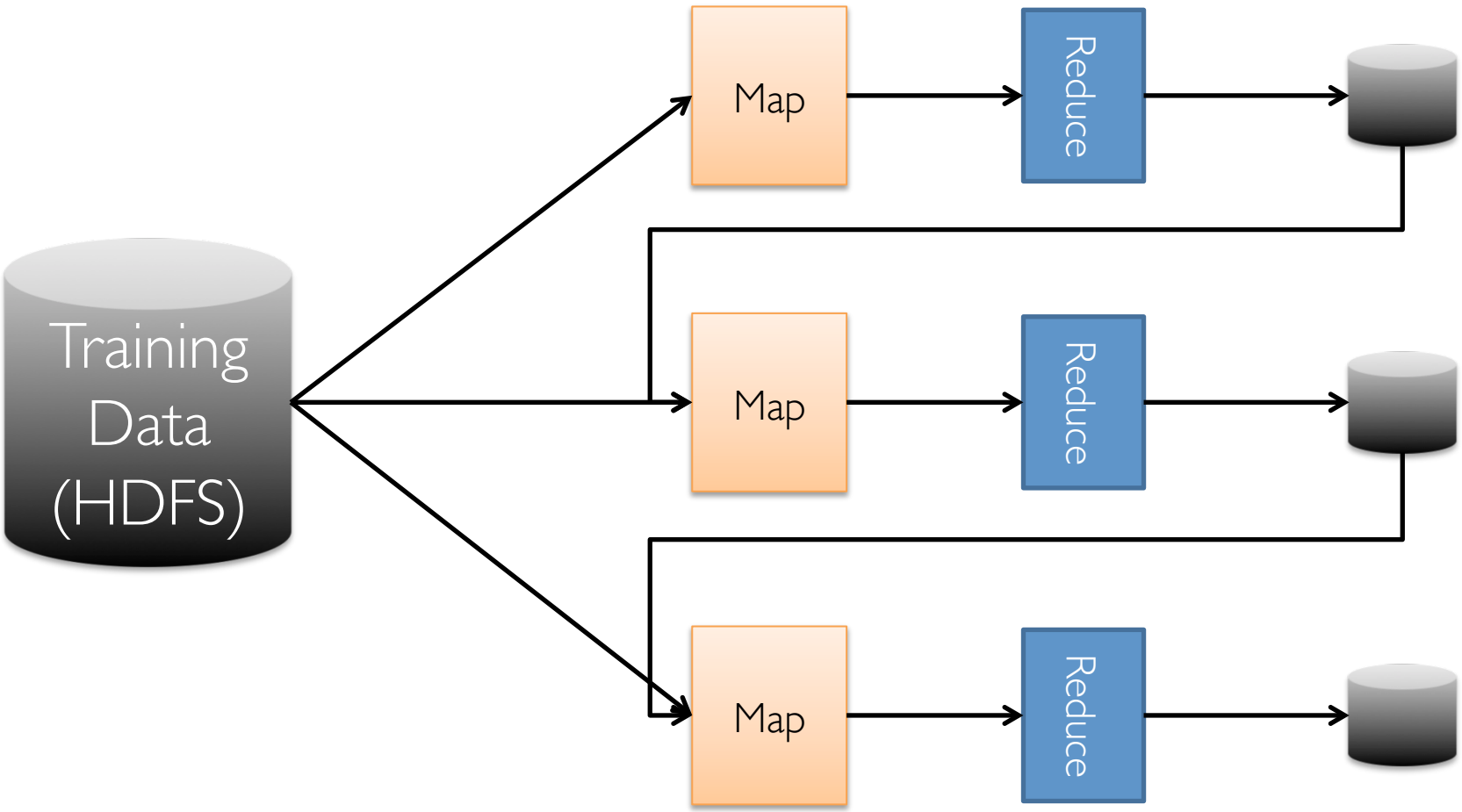
In-Memory Dataflow System

Iteration and Multi-stage computation

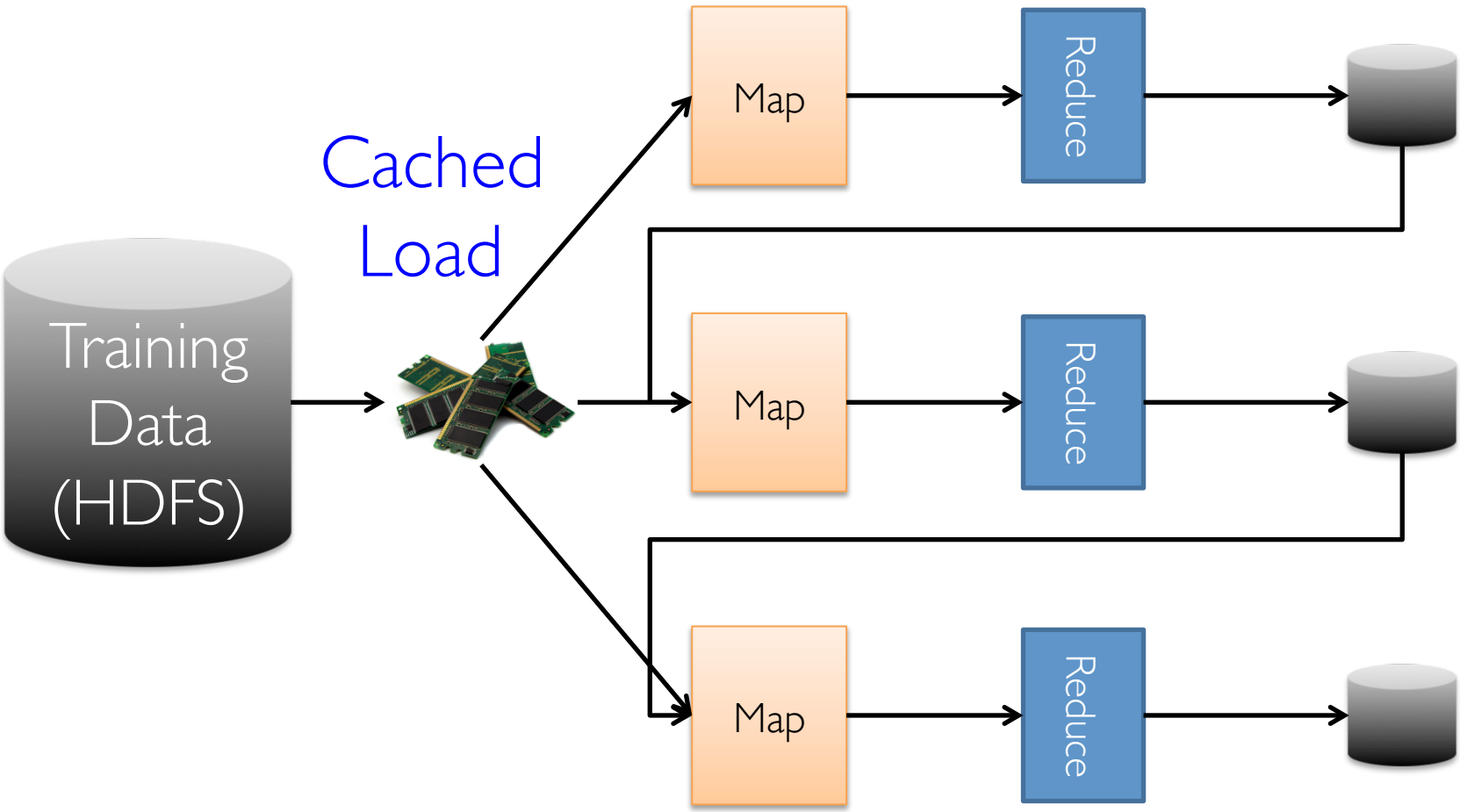
M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. *Spark: cluster computing with working sets*. HotCloud'10

M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M.J. Franklin, S. Shenker, I. Stoica. *Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing*, NSDI 2012

Dataflow View

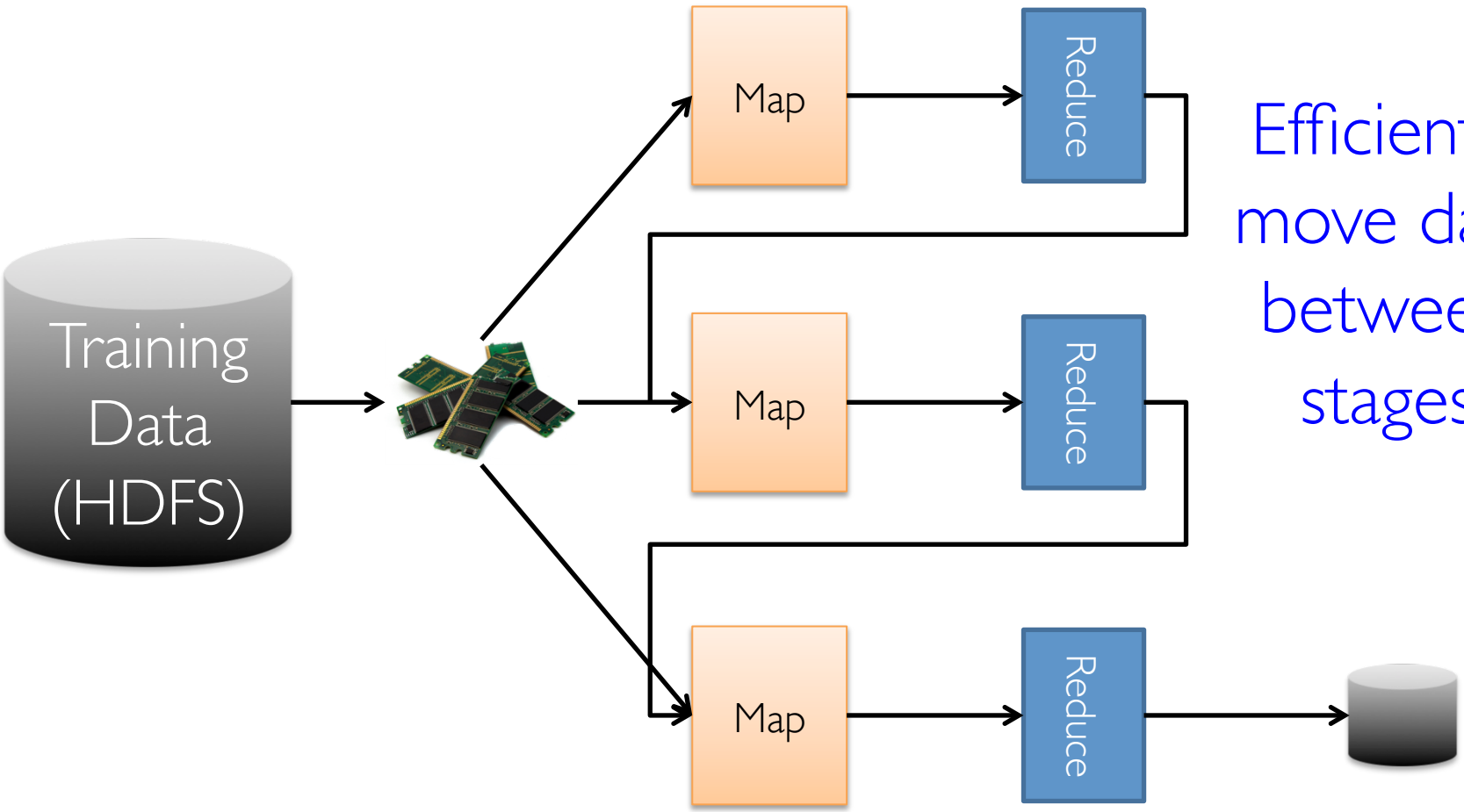


Memory Opt. Dataflow



10-100× faster than network and disk

Memory Opt. Dataflow View



Efficiently move data between stages

In-Memory Data-Flow Systems

Common Pattern:
Multi-Stage Aggregation

Abstraction: *Dataflow Ops. on Immutable datasets*

System

Spark

What is Spark?

Fault-tolerant distributed dataflow framework

Improves efficiency through:

» In-memory computing primitives
» Pipelined computation

→ Up to 100× faster (2-10× on disk)

Improves usability through:

» Rich APIs in Scala, Java, Python
» Interactive shell

→ 2-5× less code

Slide provided by M. Zaharia

Spark Programming Abstraction

Write programs in terms of transformations on distributed datasets

Resilient Distributed Datasets (RDDs)

» Distributed collections of objects that can stored in memory or on disk

» Built via parallel transformations (map, filter, …)

» Automatically rebuilt on failure

Slide provided by M. Zaharia

Example: Log Mining

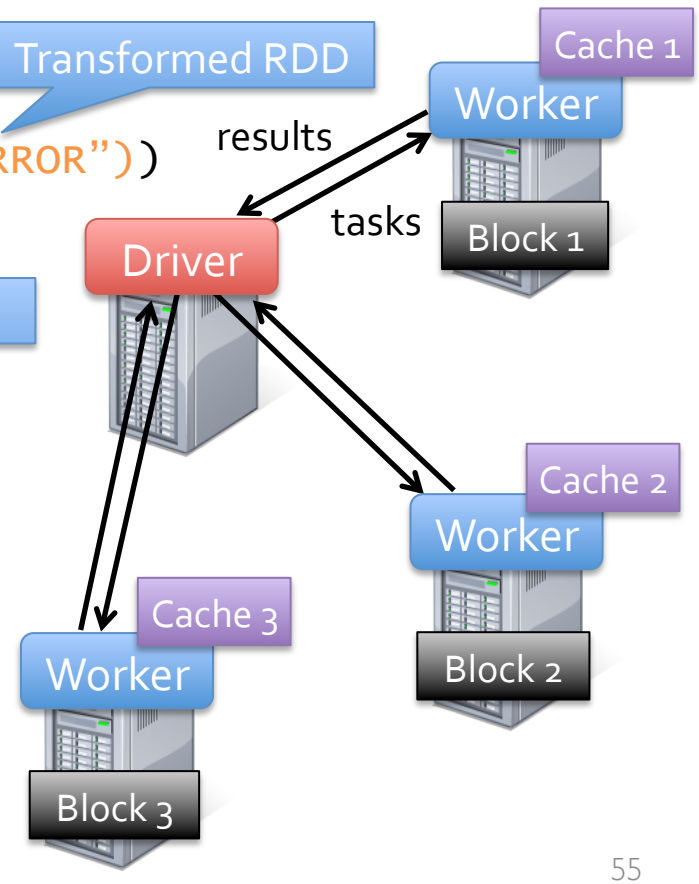
Load error messages from a log into memory, then interactively search for various patterns

Base RDD

Transformed RDD

```
lines = spark.textFile("hdfs://log")
errors = lines.filter(x => x.startsWith("ERROR"))
msgs = errors.map(x => x.split('\t')(2))
              .cache()

msgs.filter(x => x.contains("foo")).count
msgs.filter(x => x.contains("bar")).count
```

Action



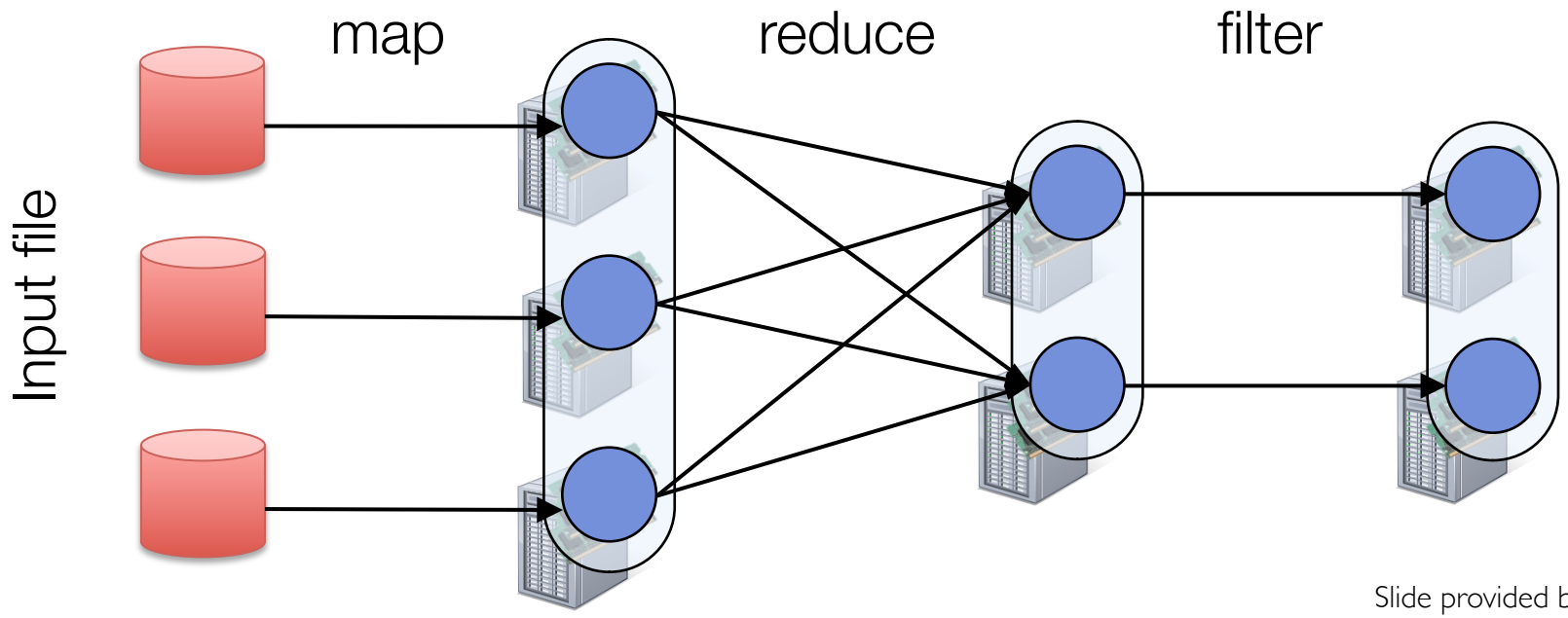
Result: scaled to 1 TB data in 5-7 sec (vs 170 sec for on-disk data)

Slide provided by M. Zaharia

Fault Tolerance

RDDs track *lineage* info to rebuild lost data

```
file.map(lambda rec: (rec.type, 1))
    .reduceByKey(lambda x, y: x + y)
    .filter(lambda (type, count): count > 10)
```

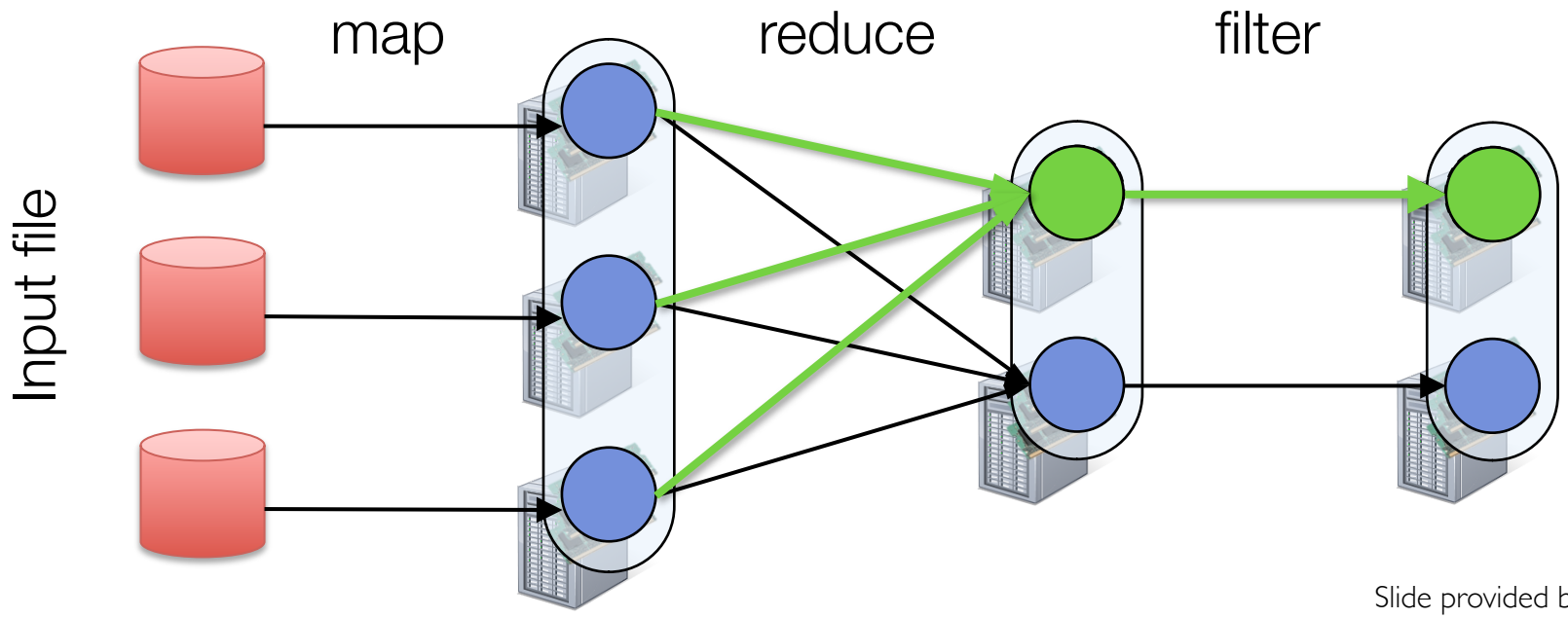


Slide provided by M. Zaharia

Fault Tolerance

RDDs track *lineage* info to rebuild lost data

```
file.map(lambda rec: (rec.type, 1))
    .reduceByKey(lambda x, y: x + y)
    .filter(lambda (type, count): count > 10)
```



Slide provided by M. Zaharia

Abstraction: *Dataflow Operators*

map	reduce	sample
filter	count	take
groupBy	fold	first
sort	reduceByKey	partitionBy
union	groupByKey	mapWith
join	cogroup	pipe
leftOuterJoin	cross	save
rightOuterJoin	zip	...

Batch Gradient Logistic Regression

```
val data = spark.textFile("hdfs://data")
                .map(readPoint).cache()

var w = Vector.random(D)

for (i <- 1 to ITERATIONS) {
  val gradient = data.map(p =>
    (1 / (1 + exp(-p.y*(w dot p.x))) - 1) * p.y * p.x
  ).reduce(_ + _)
  w -= gradient
}

println("Final w: " + w)
```

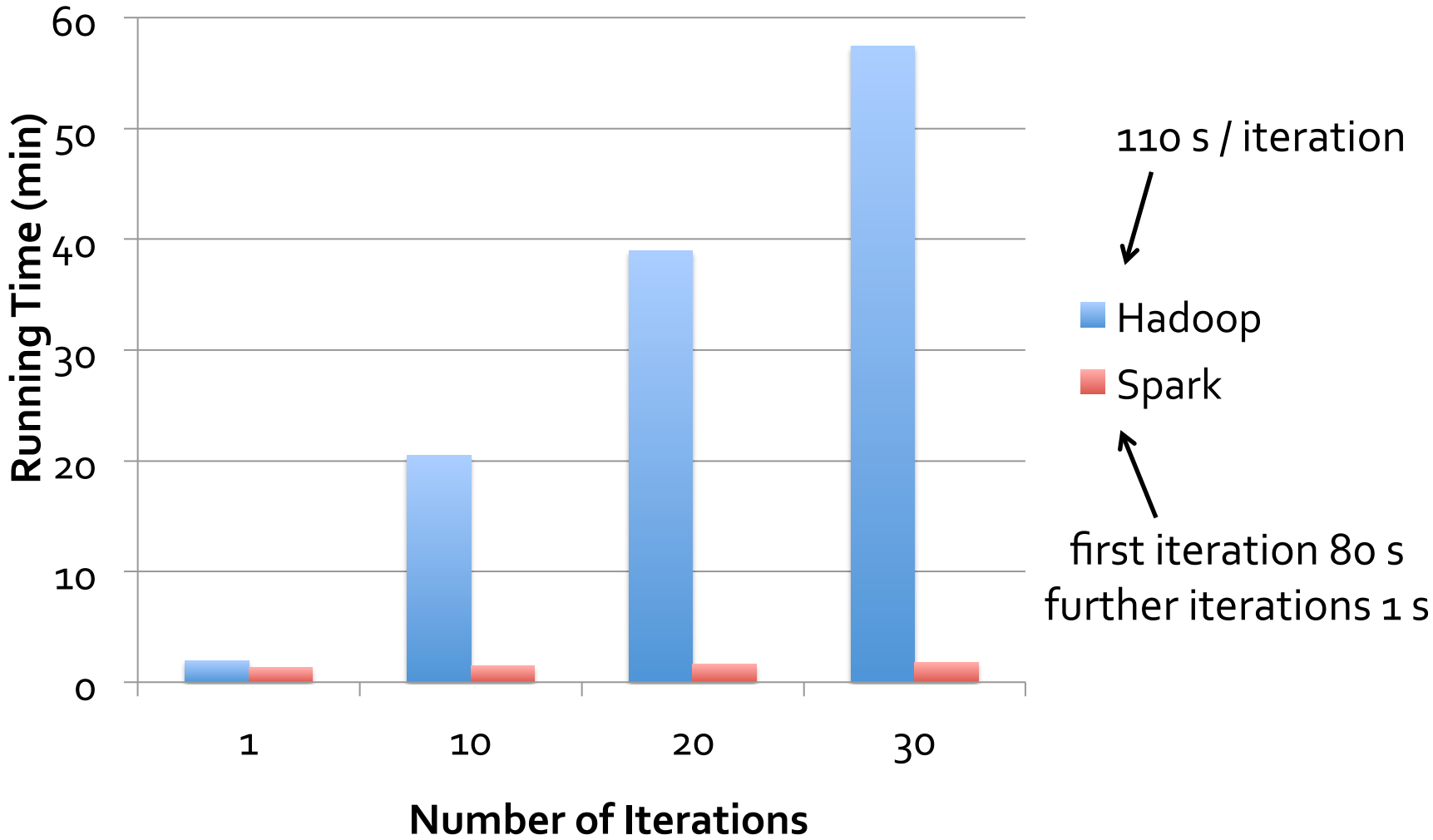
Load data in memory once

Initial parameter vector

Repeated Map-Reduce steps for gradient descent

Slide provided by M. Zaharia

Logistic Regression Performance



29 GB dataset on 20 EC2 m1.xlarge machines (4 cores each)

Slide provided by M. Zaharia

MLlib + MLbase

MLlib: high quality library for ML algorithms

» Included in Apache Spark

MLbase: make ML accessible to non-experts

» Automatically pick best algorithm

» Allow developers to easily add and test new algorithms

E. Sparks, A. Talwalkar, V. Smith, J. Kottalam, X. Pan, J. Gonzalez, Michael Franklin, Michael Jordan, Tim Kraska. *MLI: An API for Distributed Machine Learning*. ICDM'13

T. Kraska, A. Talwalkar, J. C. Duchi, R. Griffith, M. J. Franklin, and M. I. Jordan. *MLbase: A Distributed Machine-learning System.* CIDR'13

Mahout Moves to Spark

On 25 April 2014 - Goodbye MapReduce

The Mahout community decided to move its codebase onto modern data processing systems that offer a richer programming model and more efficient execution than Hadoop MapReduce. Mahout will therefore reject new MapReduce algorithm implementations from now on.

We are building our future implementations on top of a DSL for linear algebraic operations which has been developed over the last months. Programs written in this DSL are automatically optimized and executed in parallel on Apache Spark.

Other Related Systems

Microsoft Dryad and Naiad:

- http://research.microsoft.com/en-us/projects/dryad/
- http://research.microsoft.com/en-us/projects/naiad/

Hyracks: http://hyracks.org

Stratosphere: http://stratosphere.eu

MADlib: http://madlib.net

Hadoop Tez: http://hortonworks.com/hadoop/tez/

See publication lists at each of the sites

Outline of the Tutorial

1. Distributed Aggregation: Map-Reduce
2. Iterative Machine Learning: Spark
3. Large Shared Models: Parameter Server
4. Graphical Computation: GraphLab to GraphX

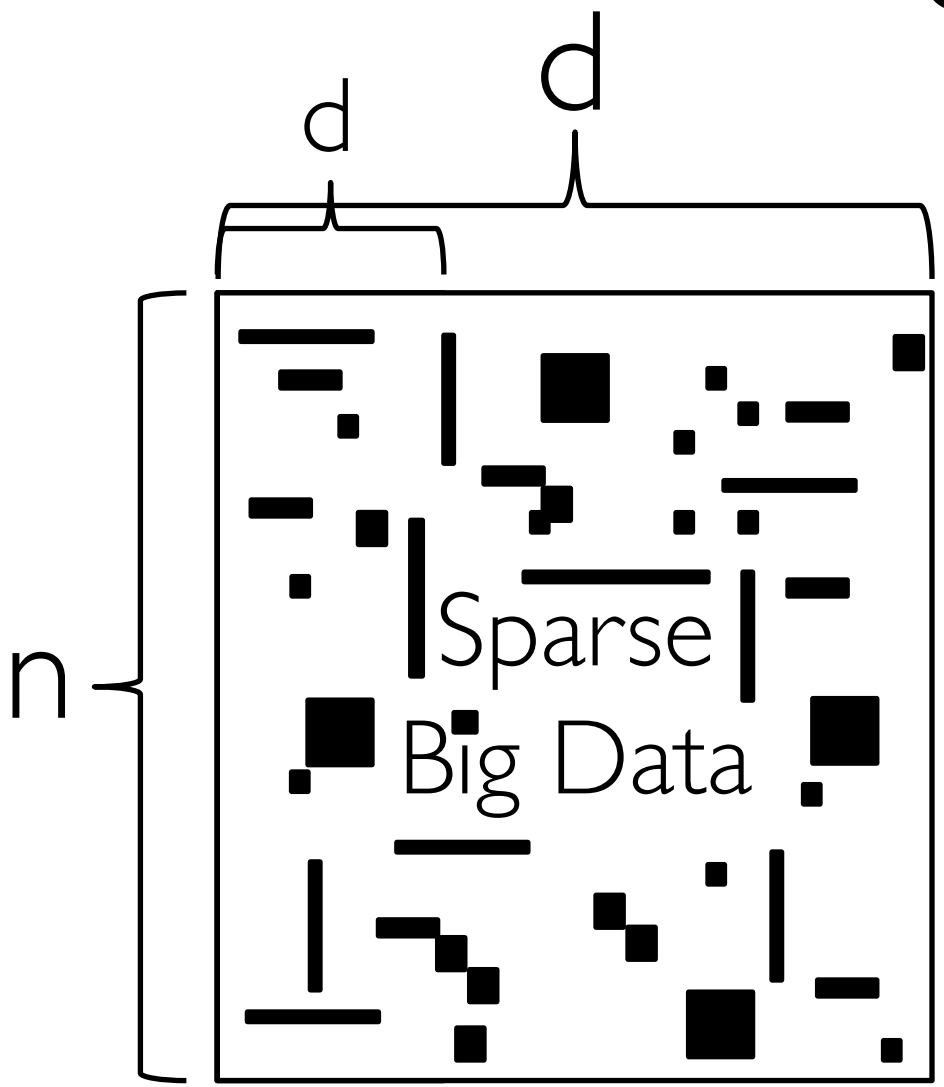
Big Models and Online Algorithms

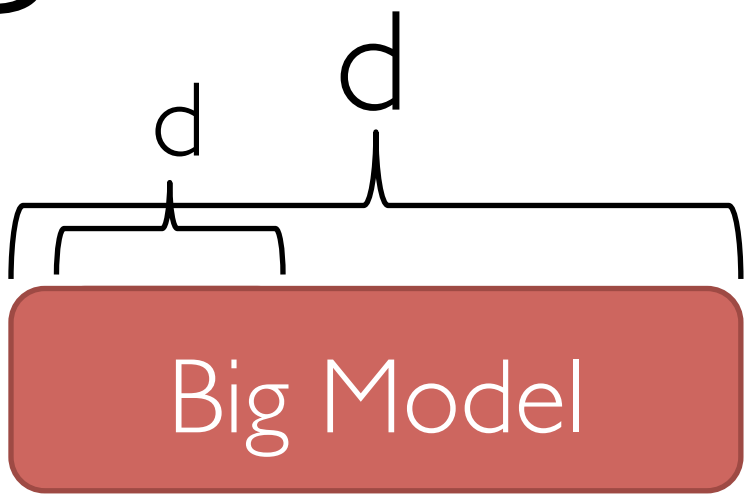
Parameter Servers

A. Smola and S. Narayanamurthy. *An architecture for parallel topic models.* VLDB'10

A. Ahmed, M. Aly, J. Gonzalez, S. Narayanamurthy, and A. J. Smola.
Scalable inference in latent variable models. WSDM '12

Small → Big Models





$d \approx n$

Examples

Spam prediction using bi-grams:

- Weight vector in $(\#\text{Words})^2$

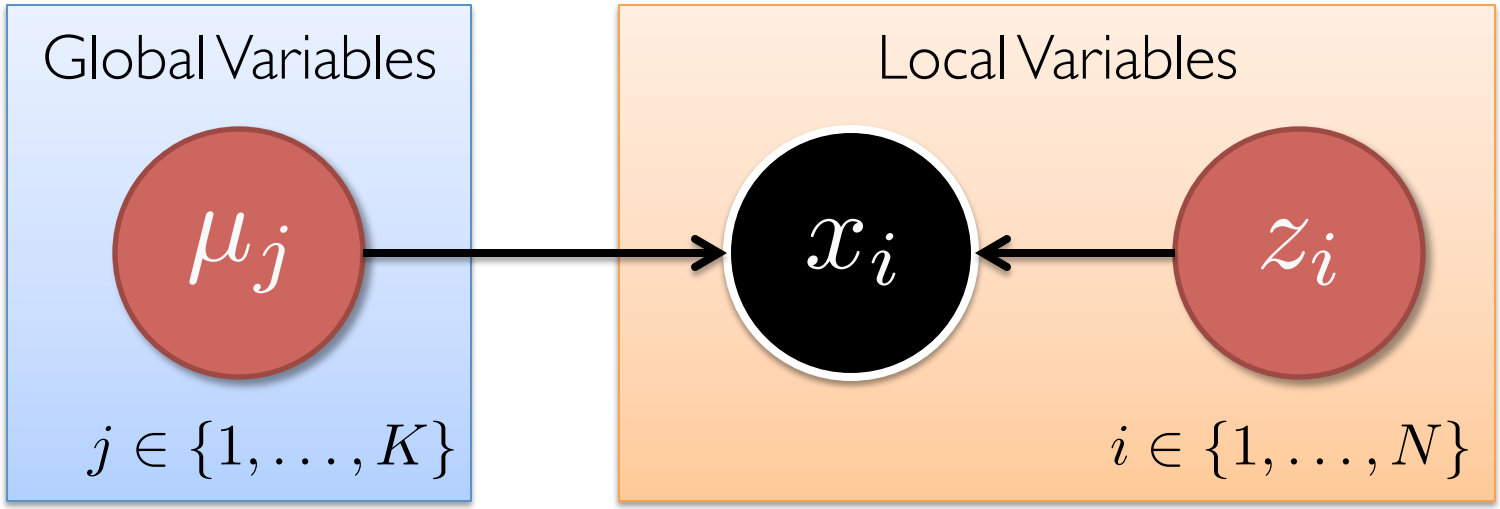
Deep Learning:

- Billions of model parameters

Topic Modeling (LDA):

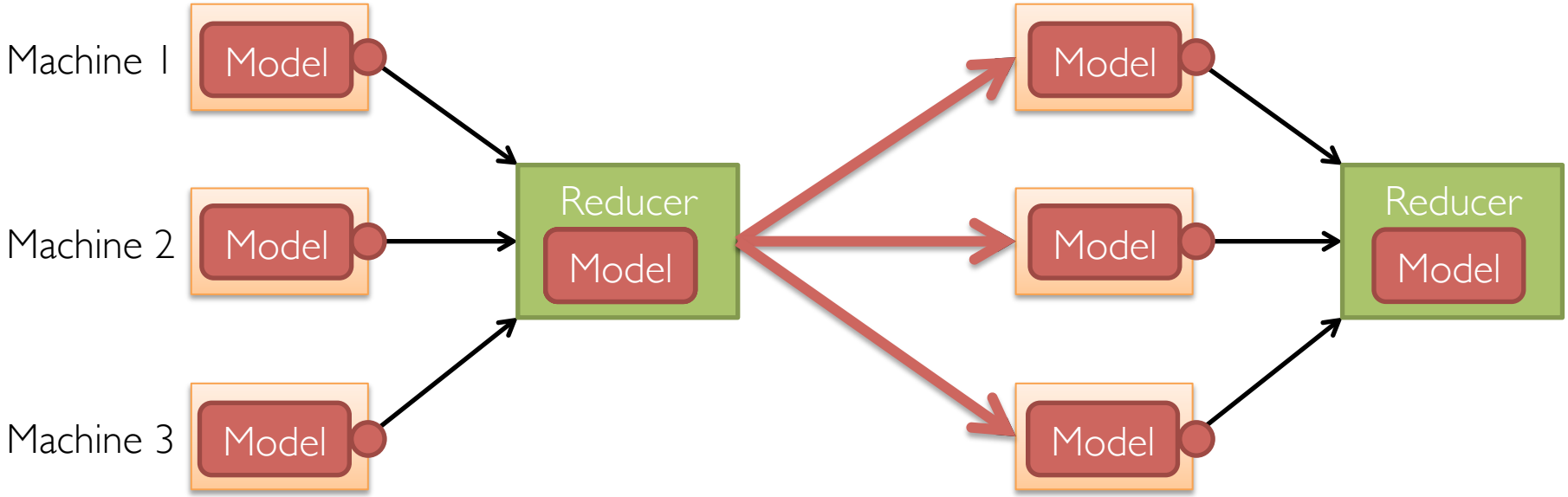
- Distribution over words for each topic

Common Pattern
Latent Var. Models



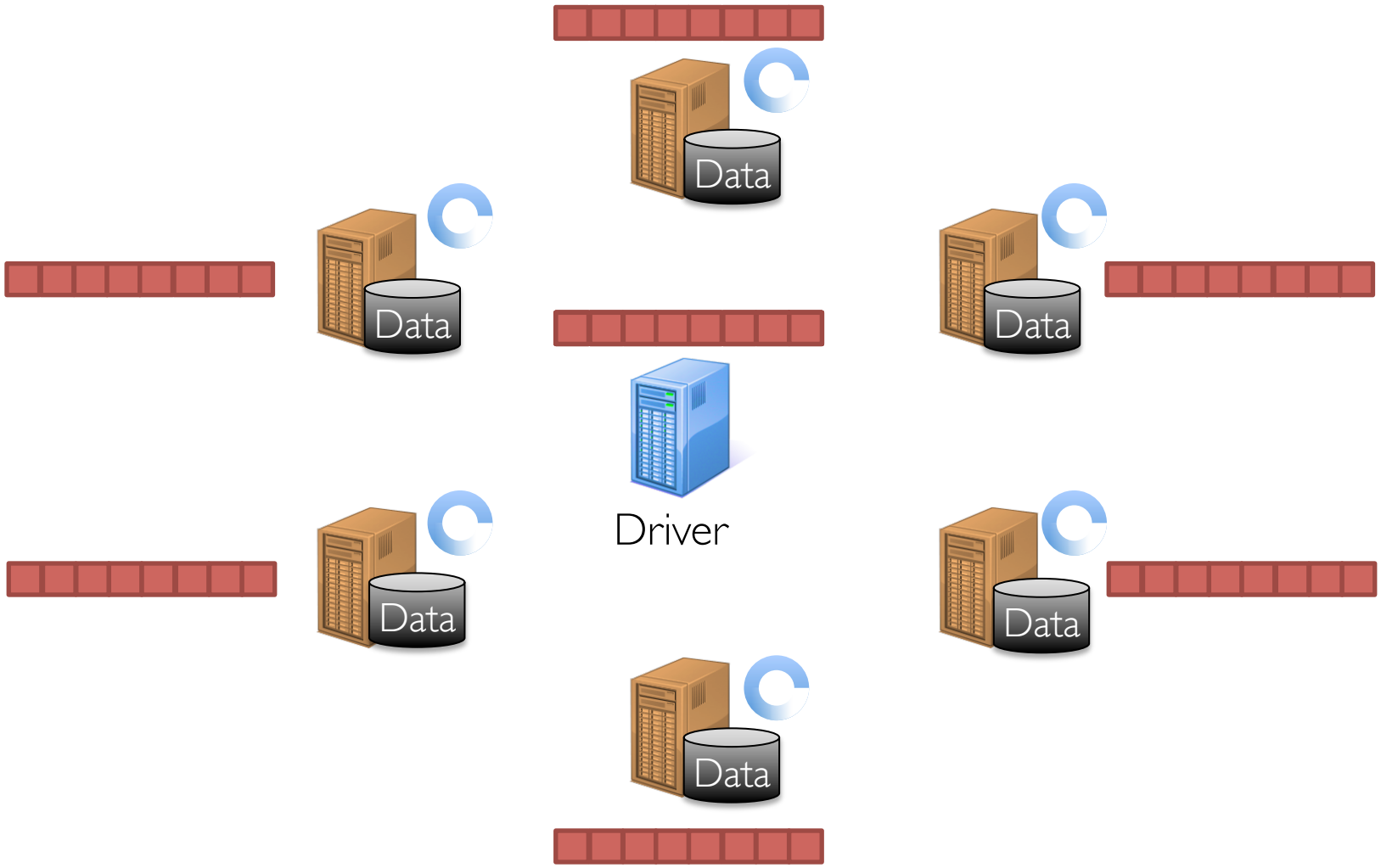
Challenge of Big Models

Example (Gradient Descent):

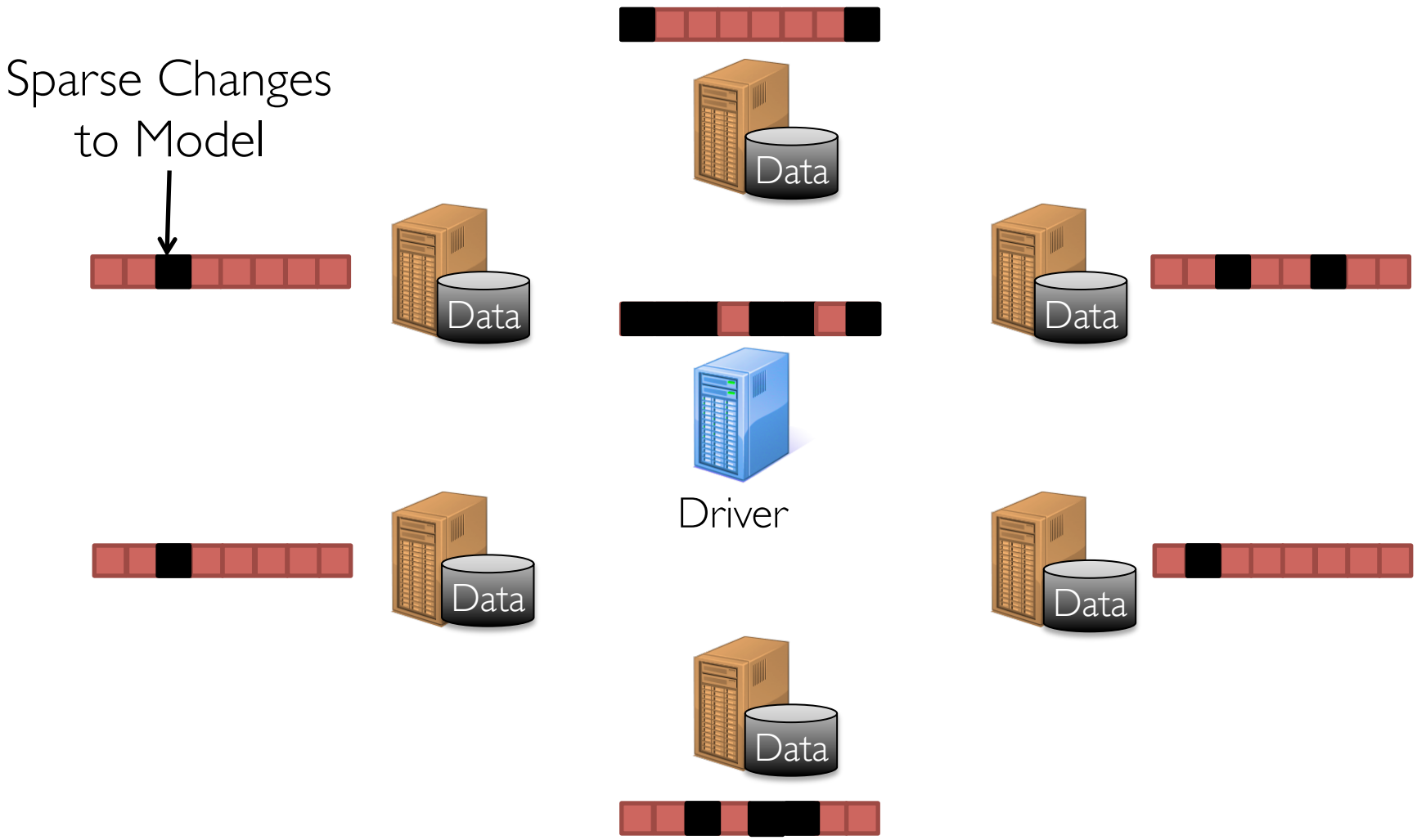


Broadcast and store
a copy of the model each iteration

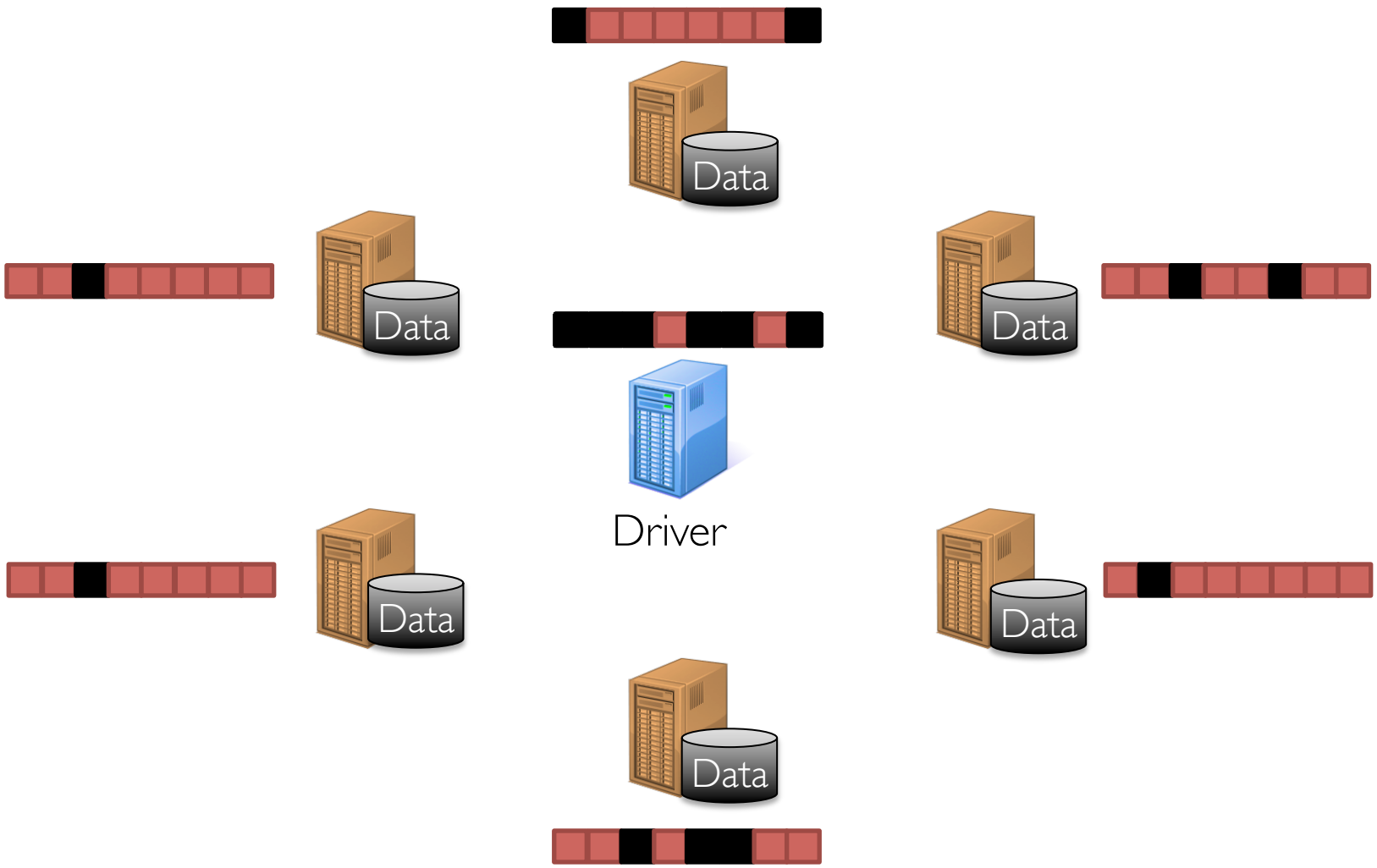
Challenge of Big Models



Challenge of Big Models

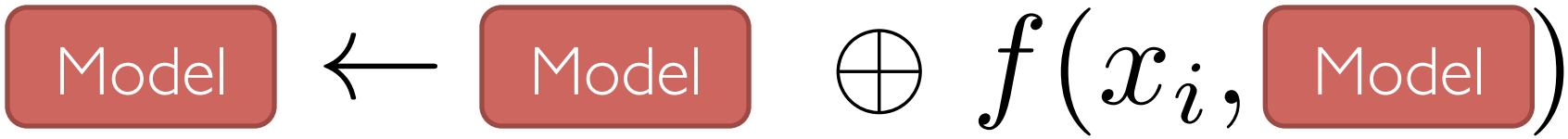


Challenge of Big Models



Online Algorithms

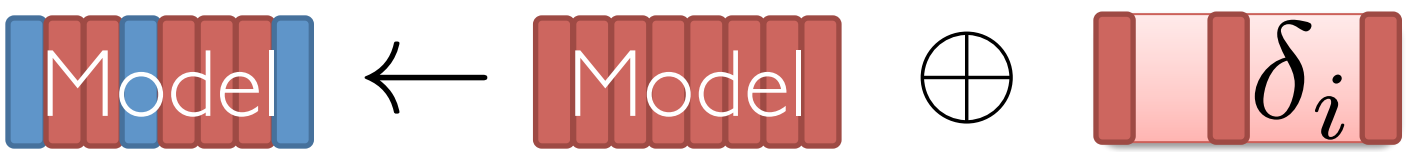
Example: Stochastic Gradient Descent



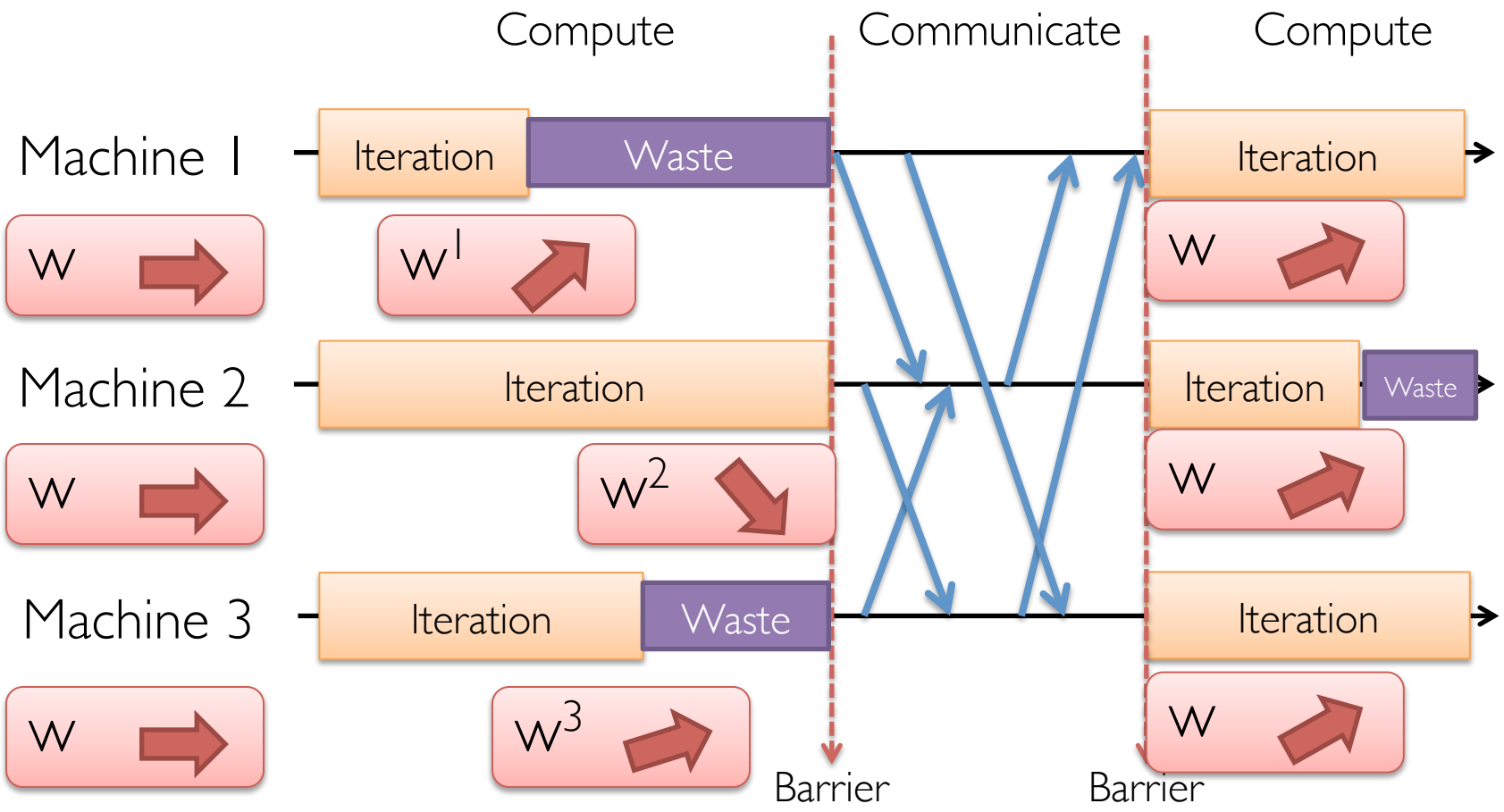
Sparse updates:

1. Comp. depends on a small part of model:

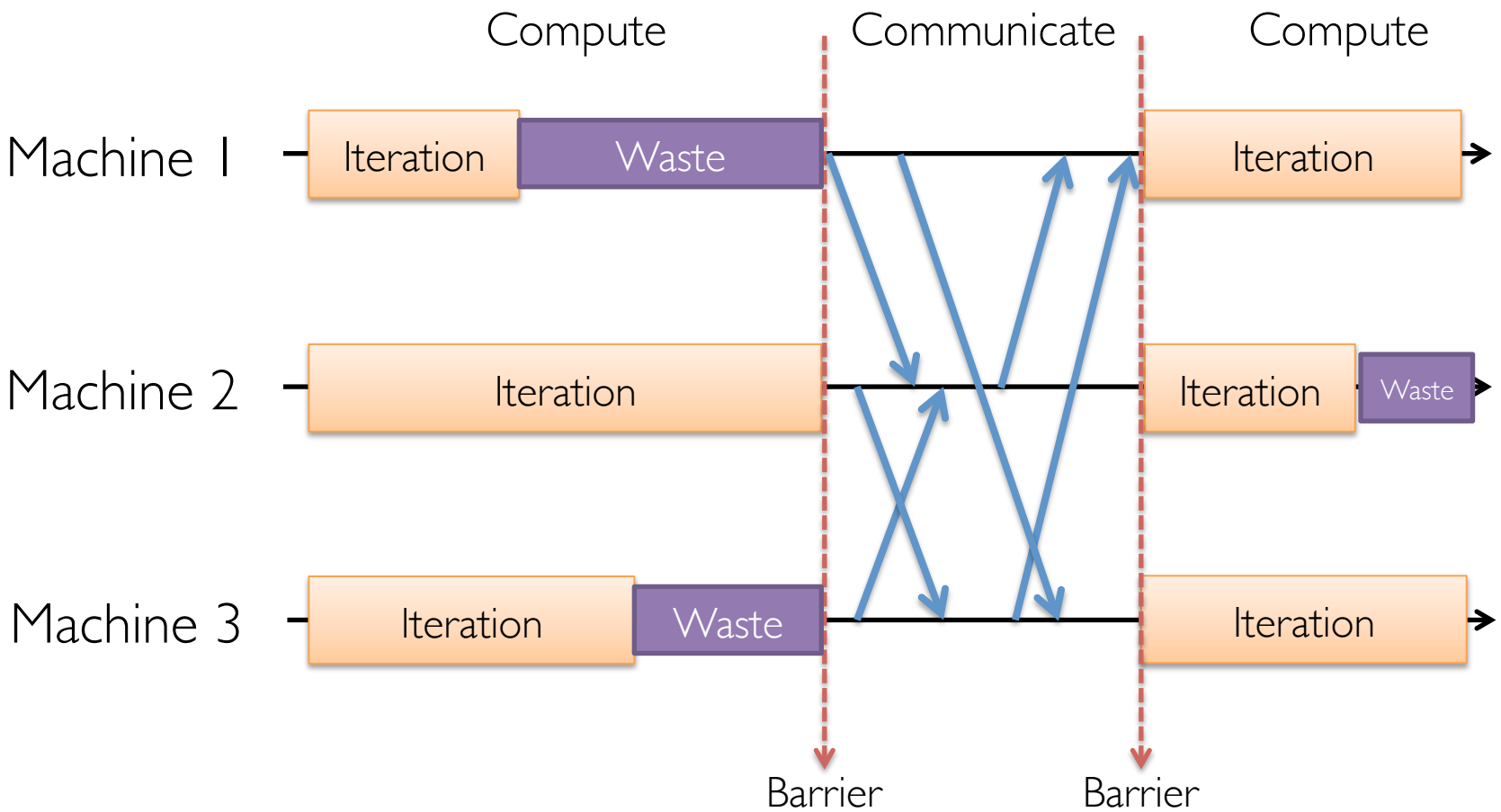
2. Sparse *additive* model update:



Bulk Synchronous Execution

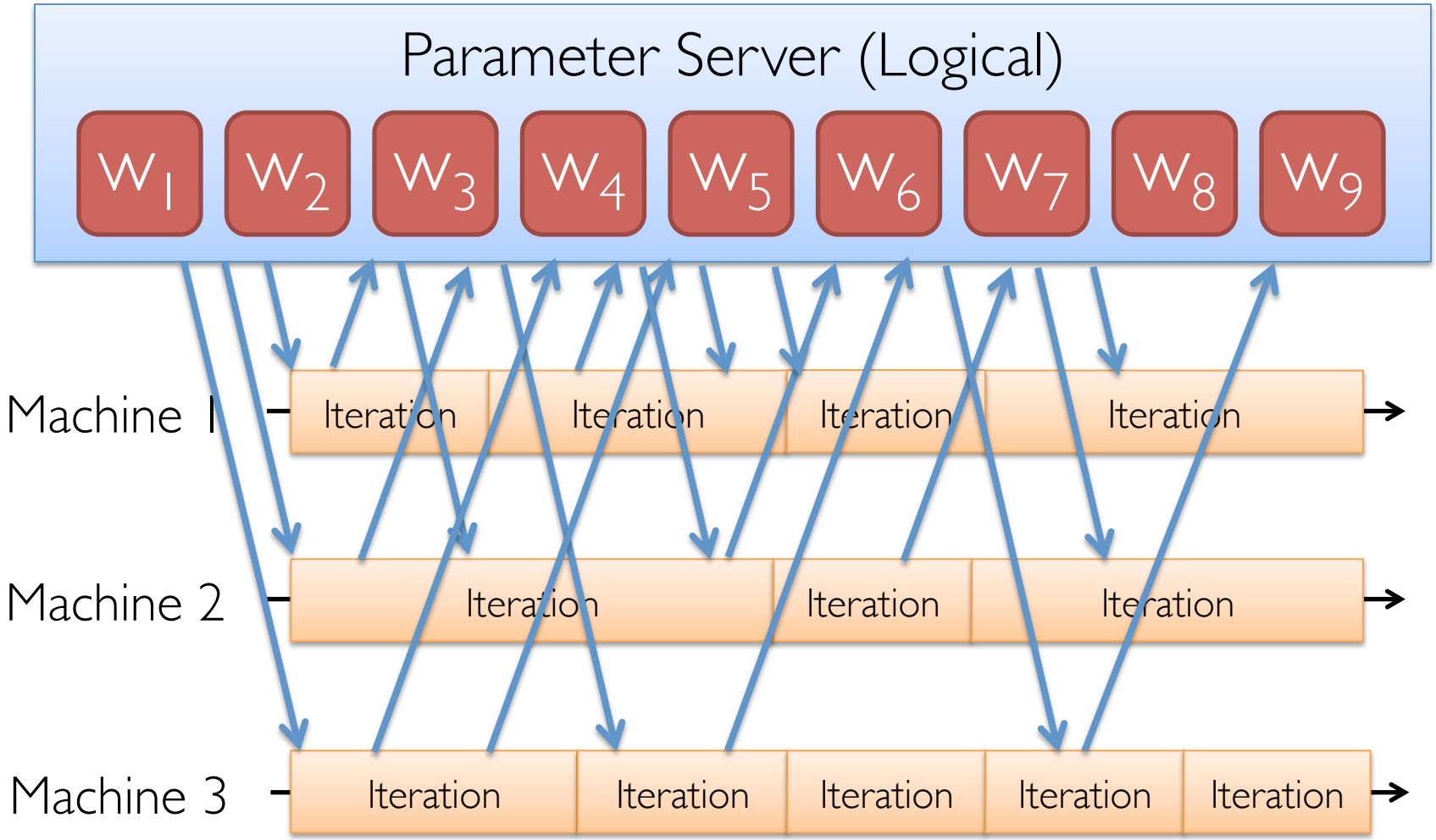


Asynchronous Execution



Enable more frequent coordination on parameter values

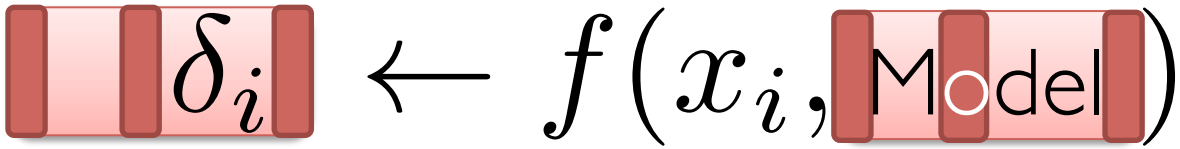
Asynchronous Execution



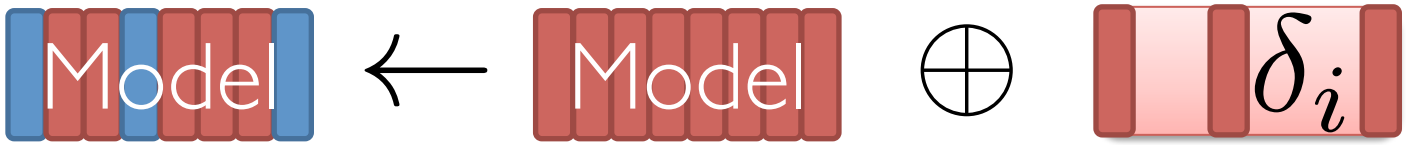
Parameter Server Abstraction

Key-Value API with two basic operations:

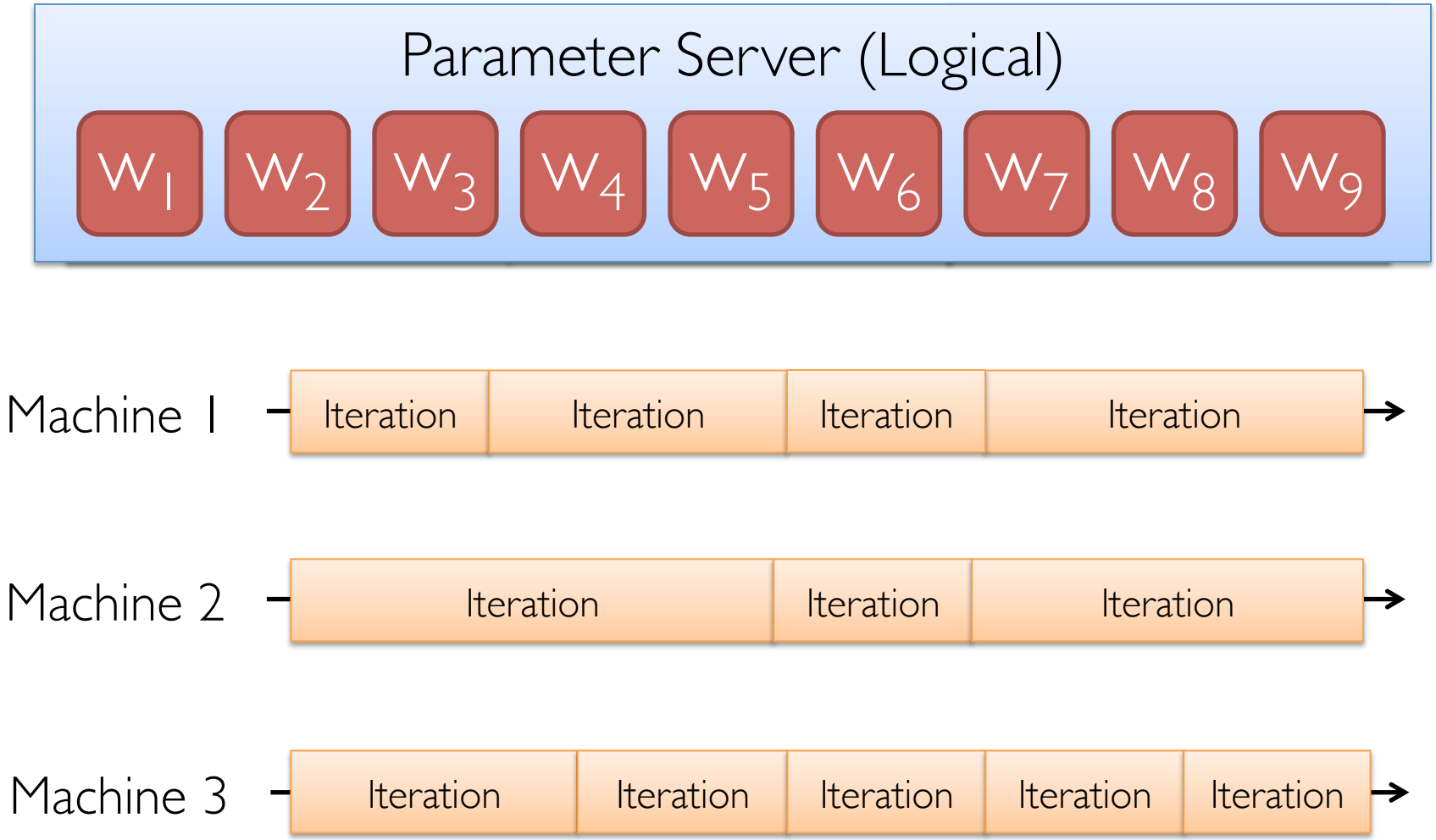
1. get(key) → value



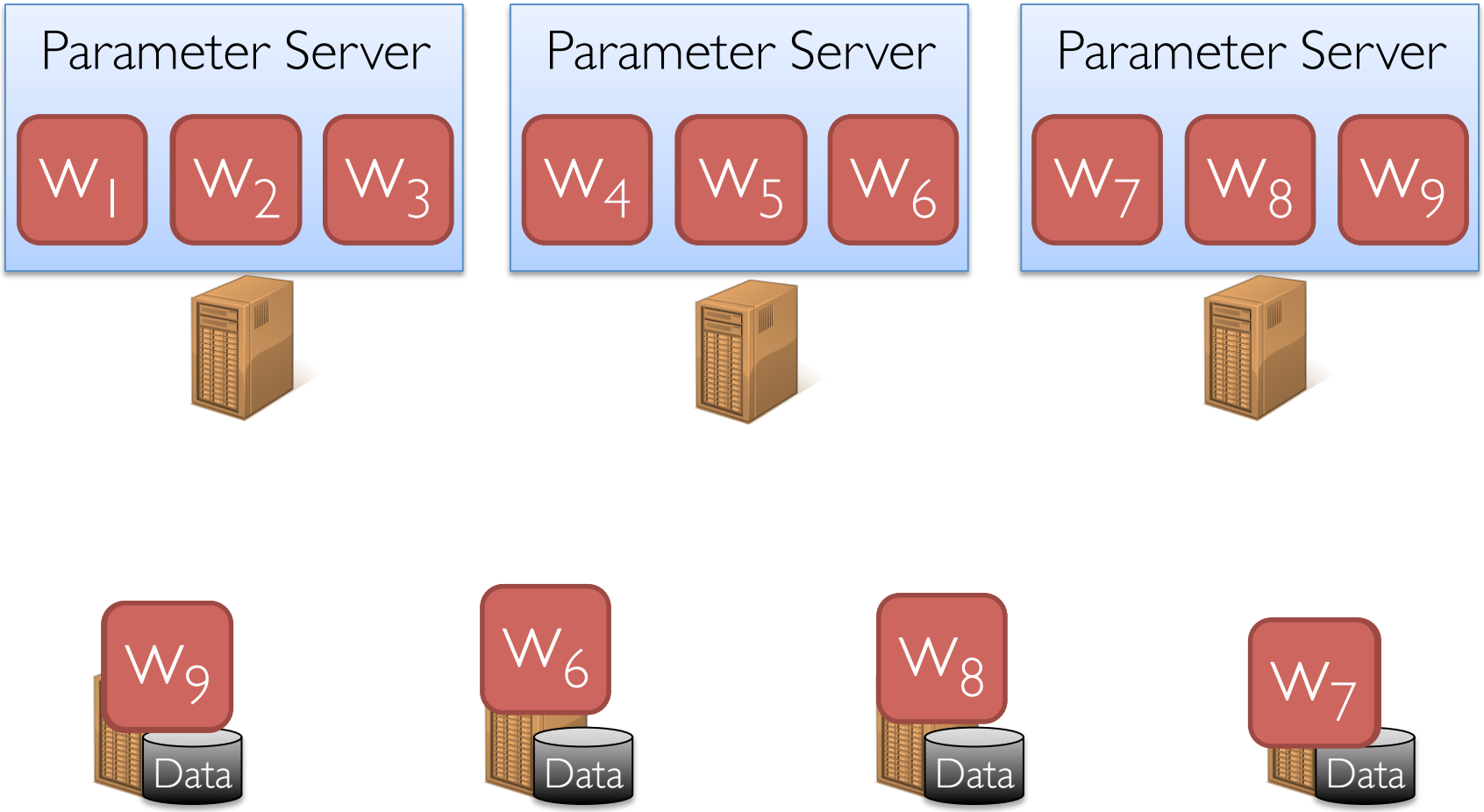
2. add(key, delta)



Split Model Across Machines

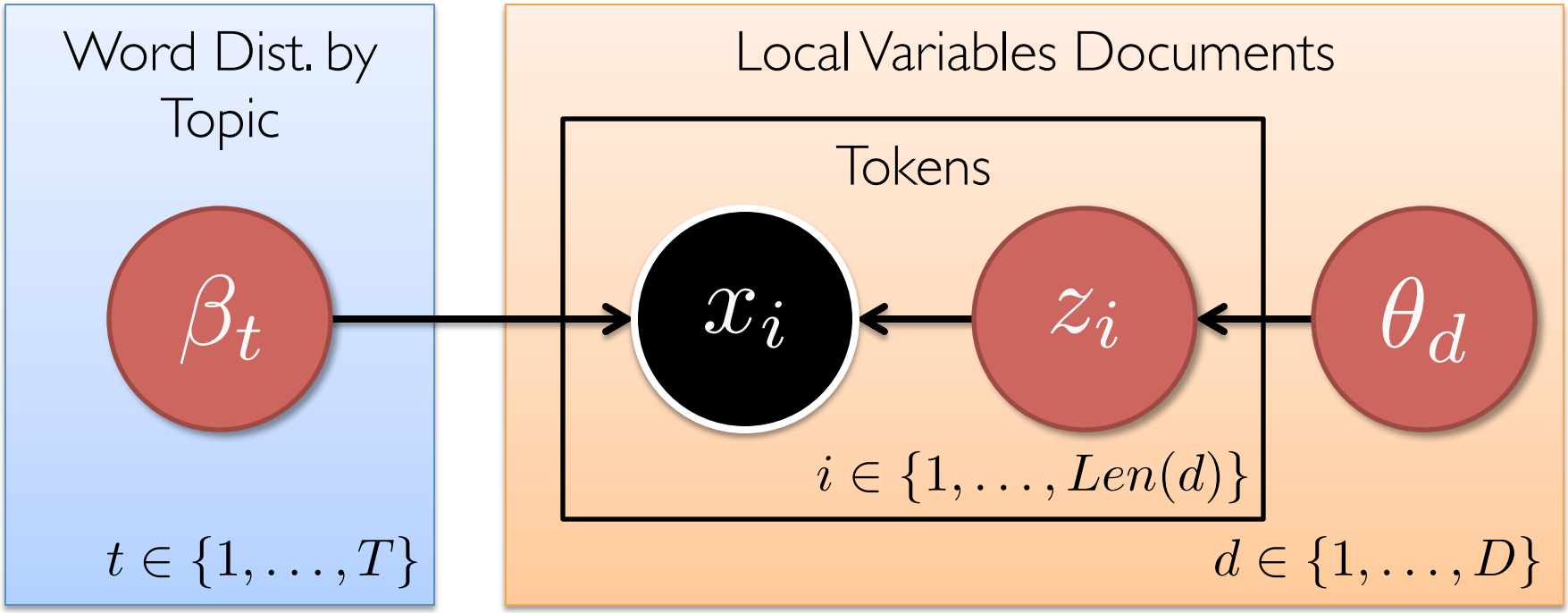


Split Model Across Machines



Split Data Across Machines

Example: Topic Modeling with LDA



Maintained by the Parameter Server

Maintained by the Workers Nodes

Gibbs Sampling for LDA

Title: *Oh, The Places You'll Go!*

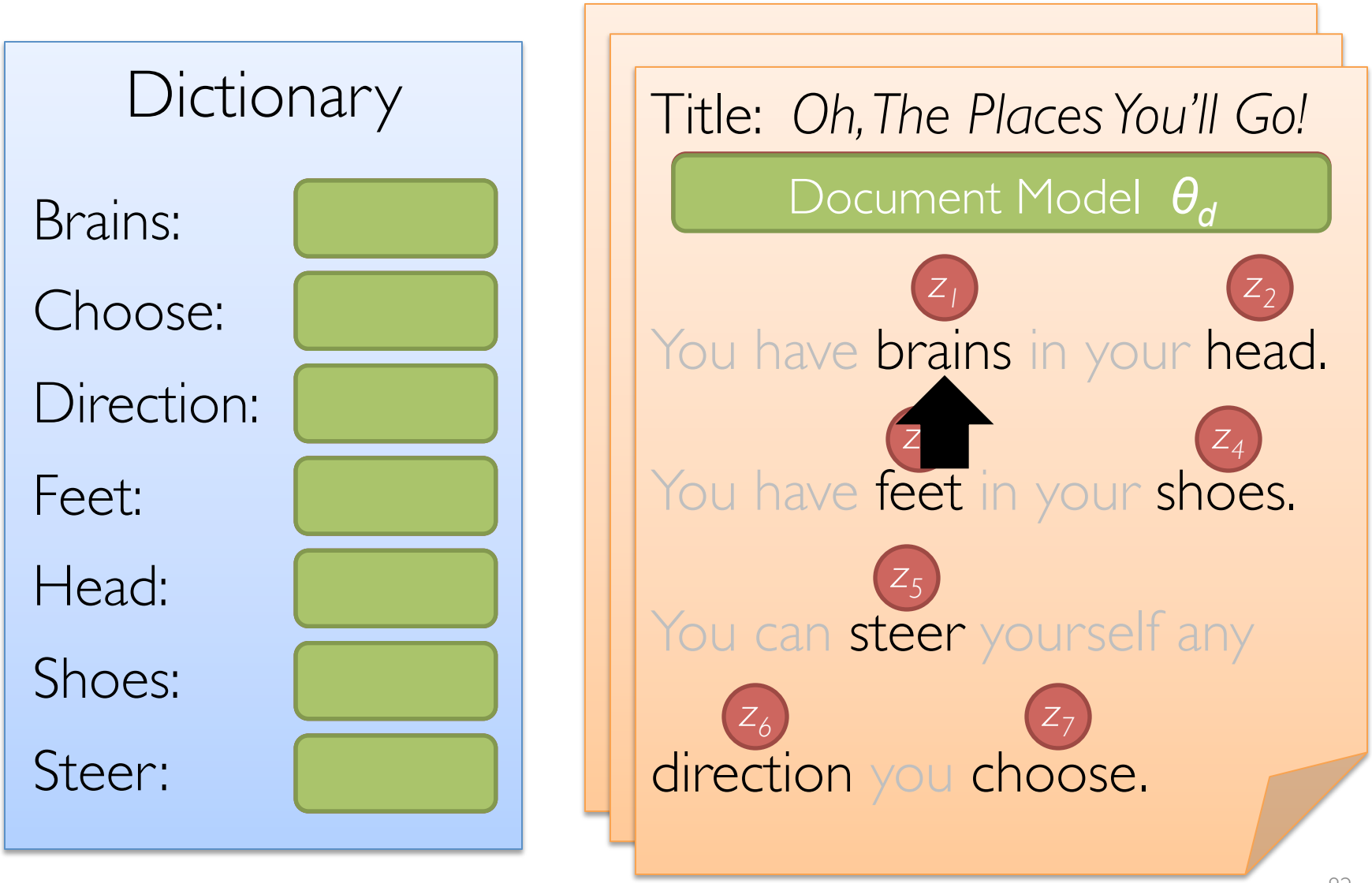
You have brains in your head.

You have feet in your shoes.

You can steer yourself any

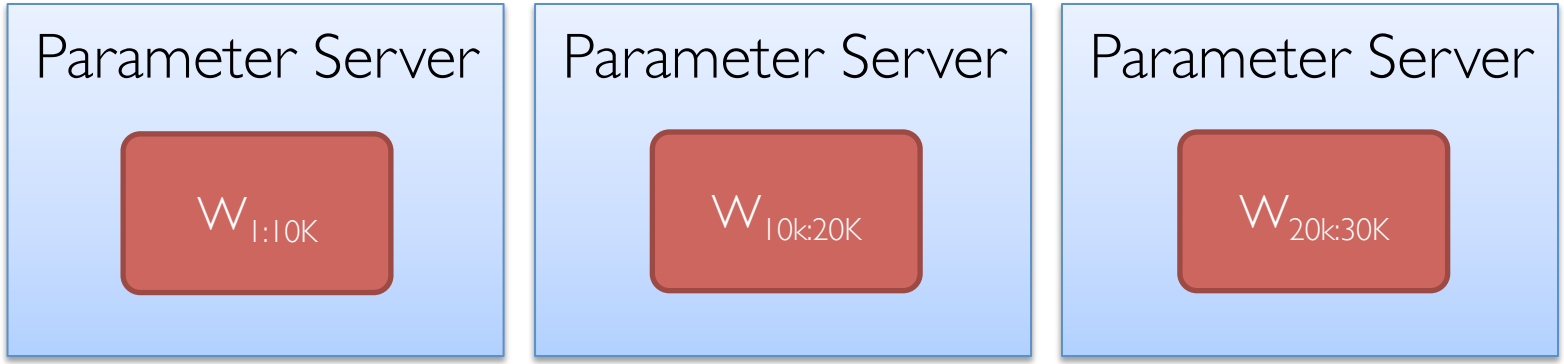
direction you choose.

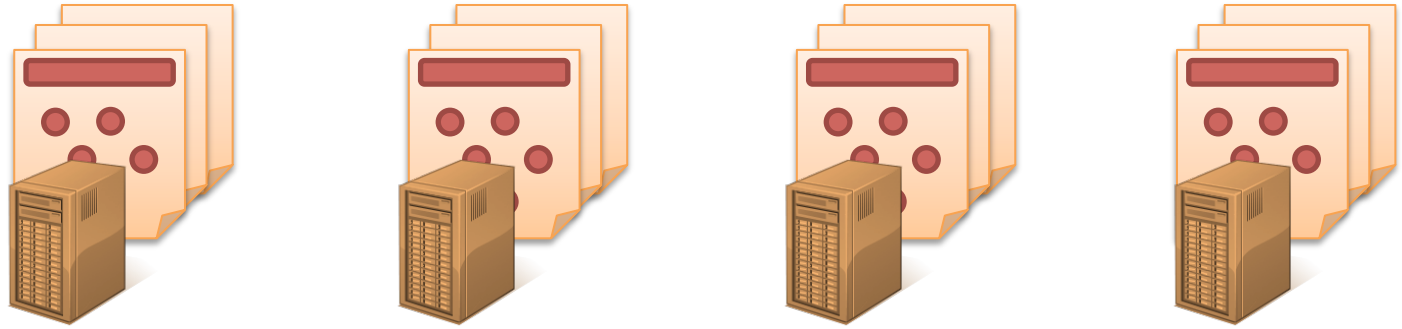
Gibbs Sampling for LDA



Ex: Collapsed Gibbs Sampler for LDA

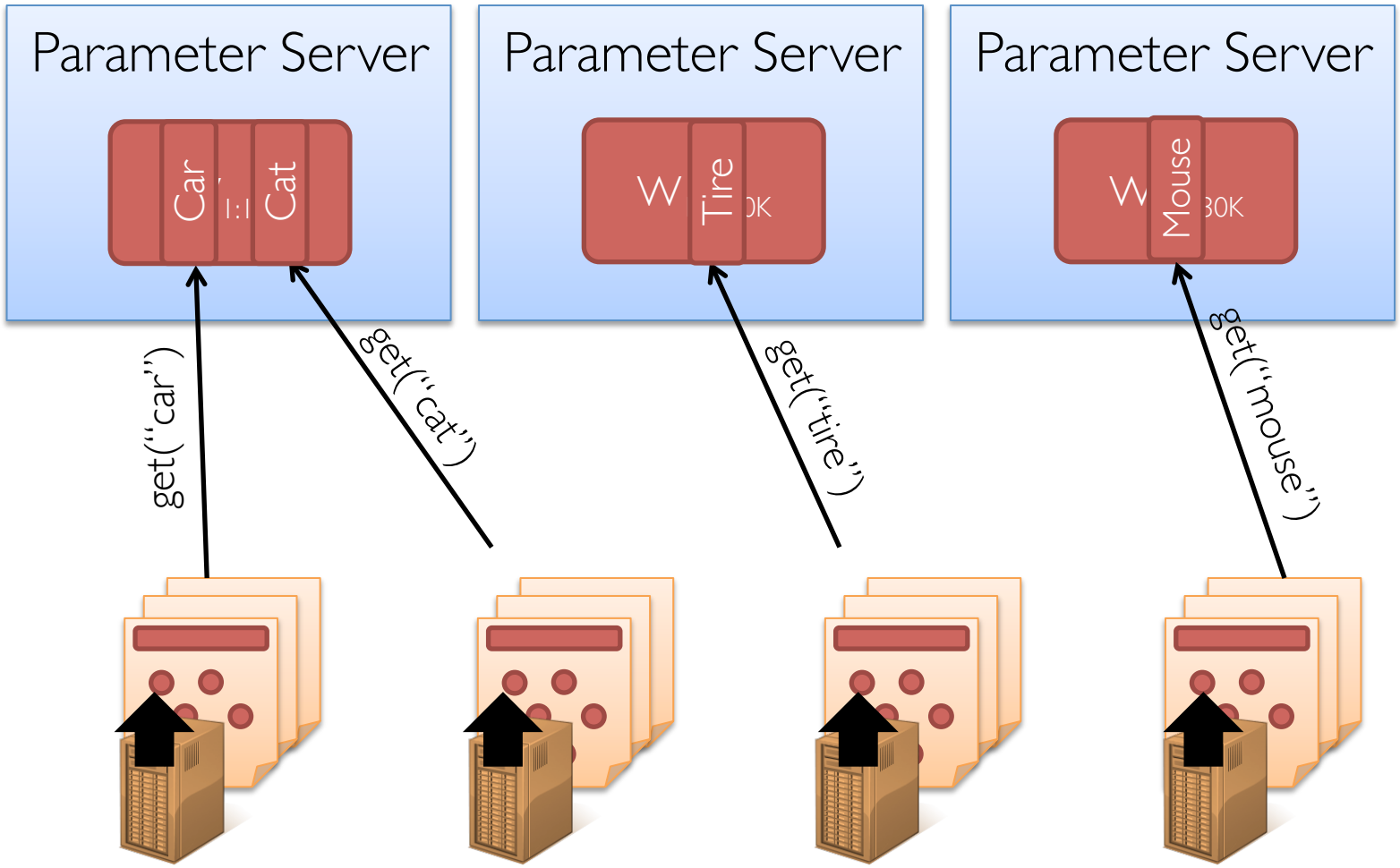
Partitioning the model and data





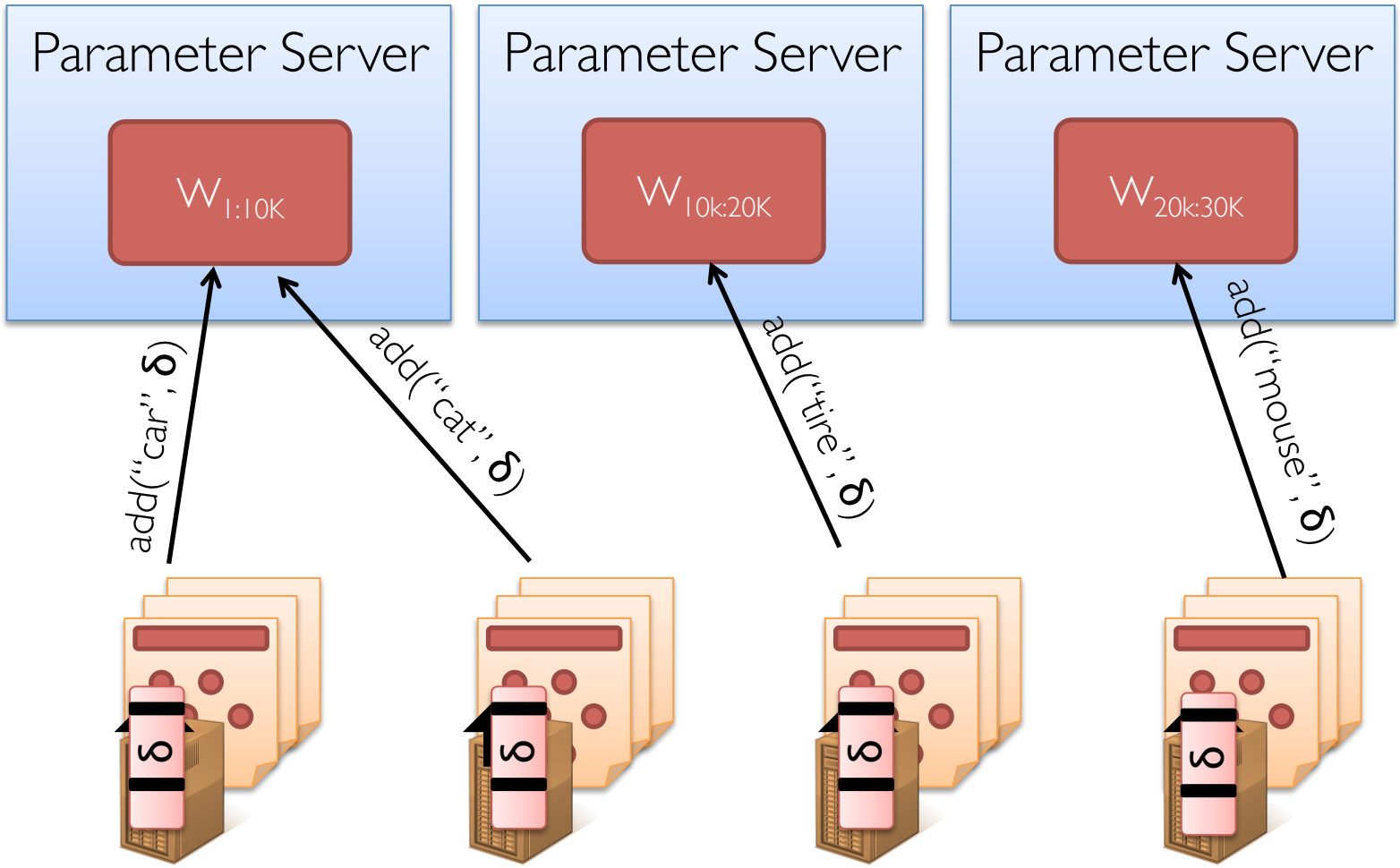
Ex: Collapsed Gibbs Sampler for LDA

Get model parameters and compute update



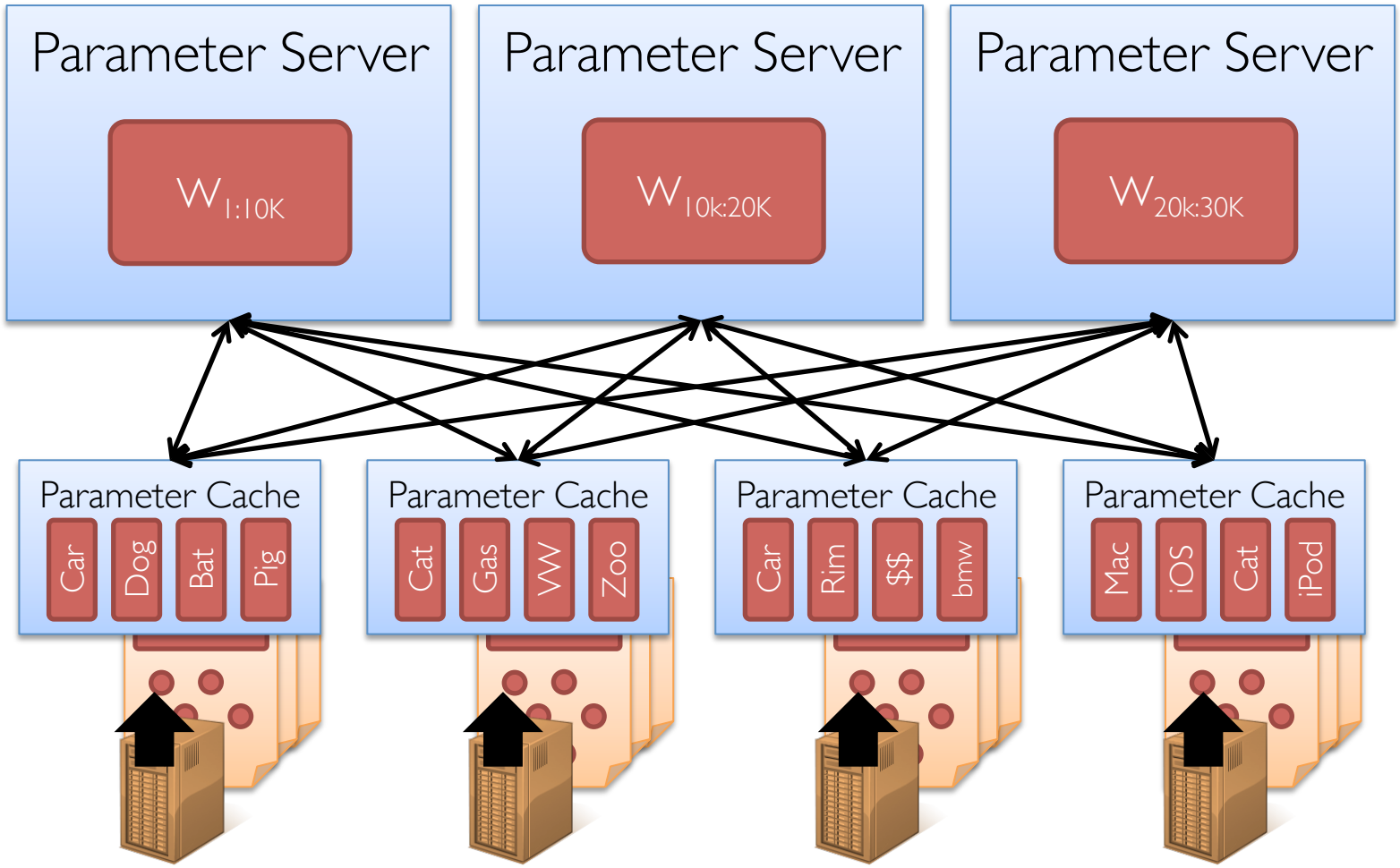
Ex: Collapsed Gibbs Sampler for LDA

Send changes back to the parameter server



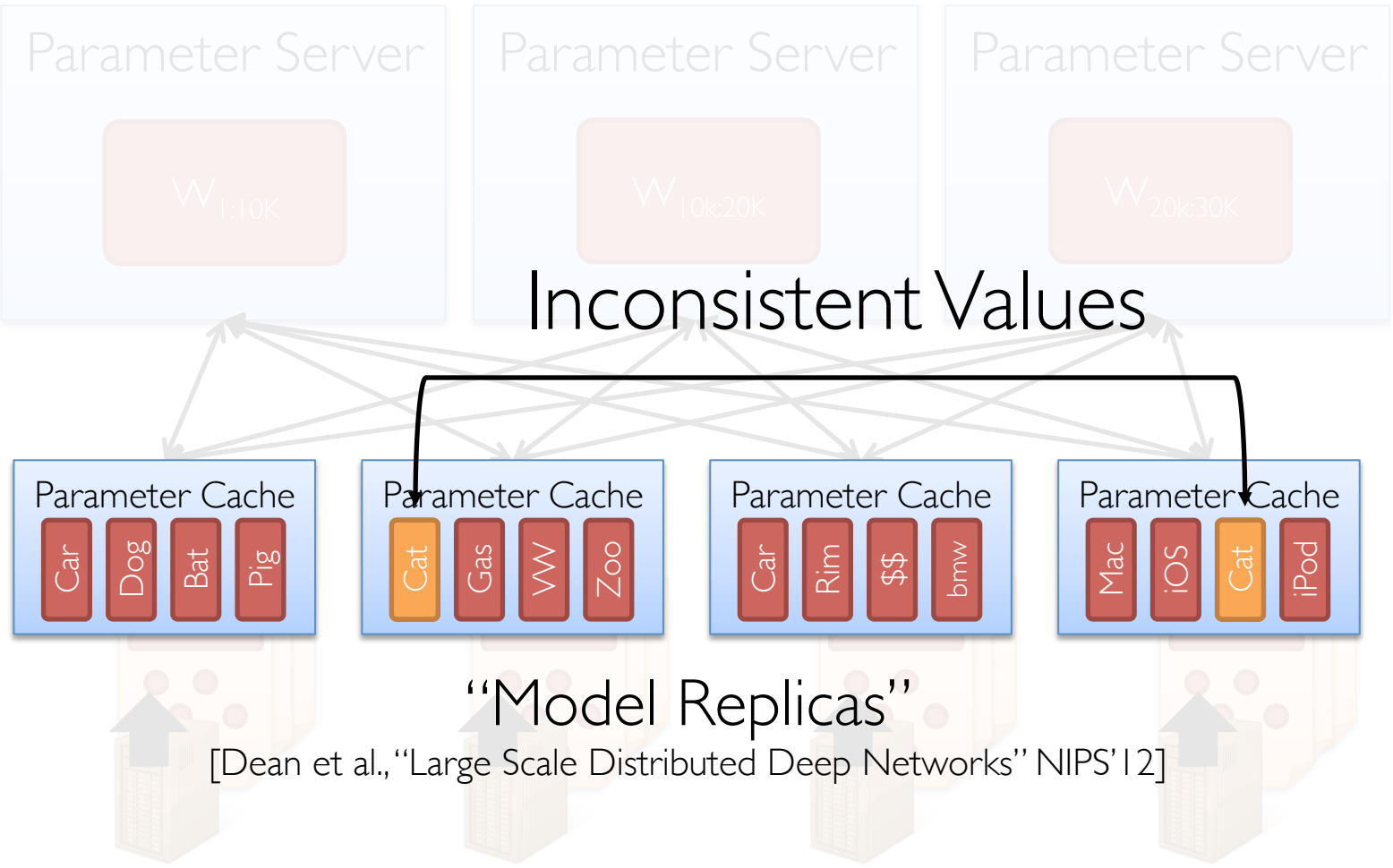
Ex: Collapsed Gibbs Sampler for LDA

Adding a caching layer to collect updates



Ex: Collapsed Gibbs Sampler for LDA

Inconsistent model replicas



"Model Replicas"

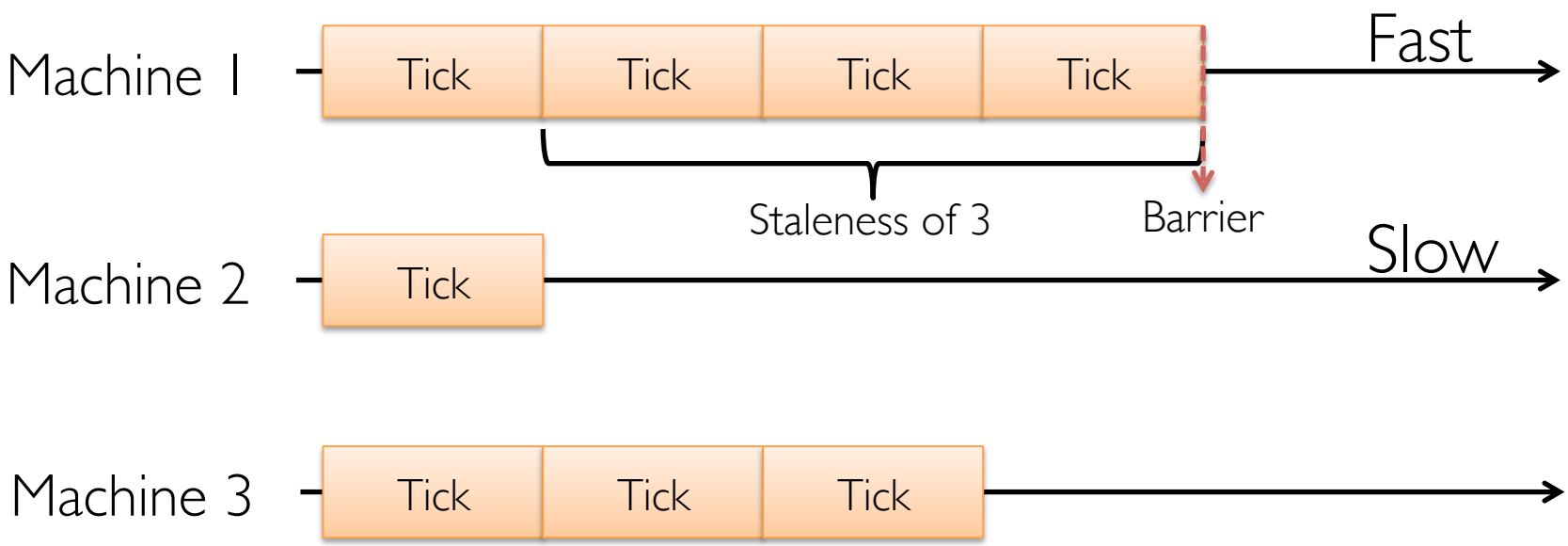
[Dean et al., "Large Scale Distributed Deep Networks" NIPS'12]

Bounding Staleness

Ho et al. *"More Effective Distributed ML via a Stale Synchronous Parallel Parameter Server."* NIPS'13

Slow-down fast workers

Force periodic cache synchronization



Bounding Staleness

Ho et al. *"More Effective Distributed ML via a Stale Synchronous Parallel Parameter Server."* NIPS'13

Slow-down fast workers

Force periodic cache synchronization

Currently the analysis only:

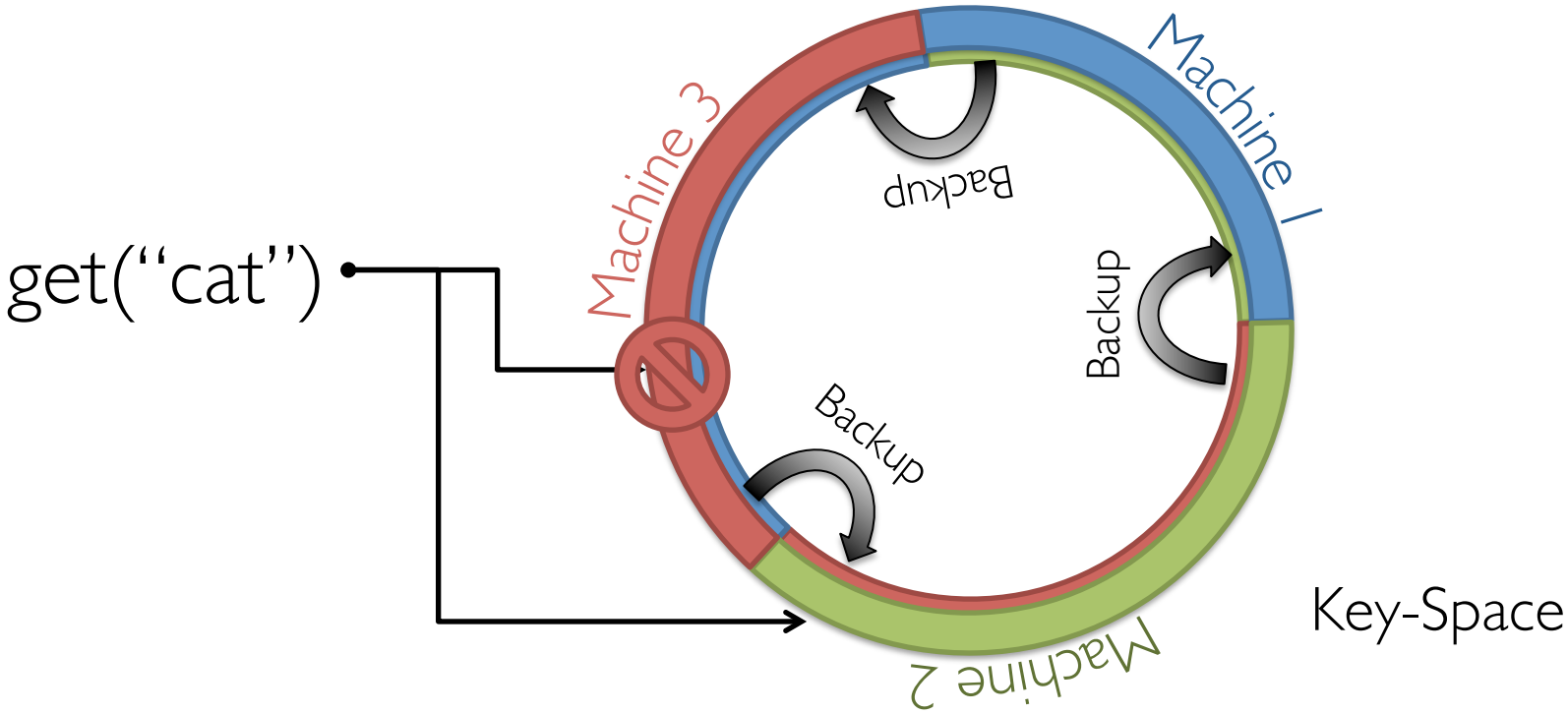
- applies to convex functions
- characterizes the average objective value

Opportunity for more research.

Fault Tolerance

M. Li et al. *Parameter Server for Distributed Machine Learning, Big Learning* Workshop, NIPS'13

Consistent Hashing:



Parameter Server Implementations

ParameterServer.org: Alex Smola's Lab

- C++, Apache License
- Code: https://github.com/mli/parameter_server

Petuum.org: Eric Xing's Lab

- C++, BSD License
- Code: https://github.com/sailinglab/petuum

Applications of Parameter Servers

Distributed Gibbs Sampling: A. Ahmed et al., *Scalable inference in latent variable models.* WSDM '12

Stochastic Gradient Descent

Deep Learning: Dean et al., *Large Scale Distributed Deep Networks.* NIPS'12

Matrix Factorization: Ho et al. *More Effective Distributed ML via a Stale Synchronous Parallel Parameter Server.* NIPS'13

Specialization for SGD

Vowpal Wabbit: http://hunch.net/~vw/

- Primary use is fast online linear optimization
- Distributed optimization tools

Bismark: X. Feng, A. Kumar, B. Recht, and C. Ré. *Towards a unified architecture for in-RDBMS analytics.* SIGMOD'12

- In database incremental gradient descent

Limitations of the Parameter Server

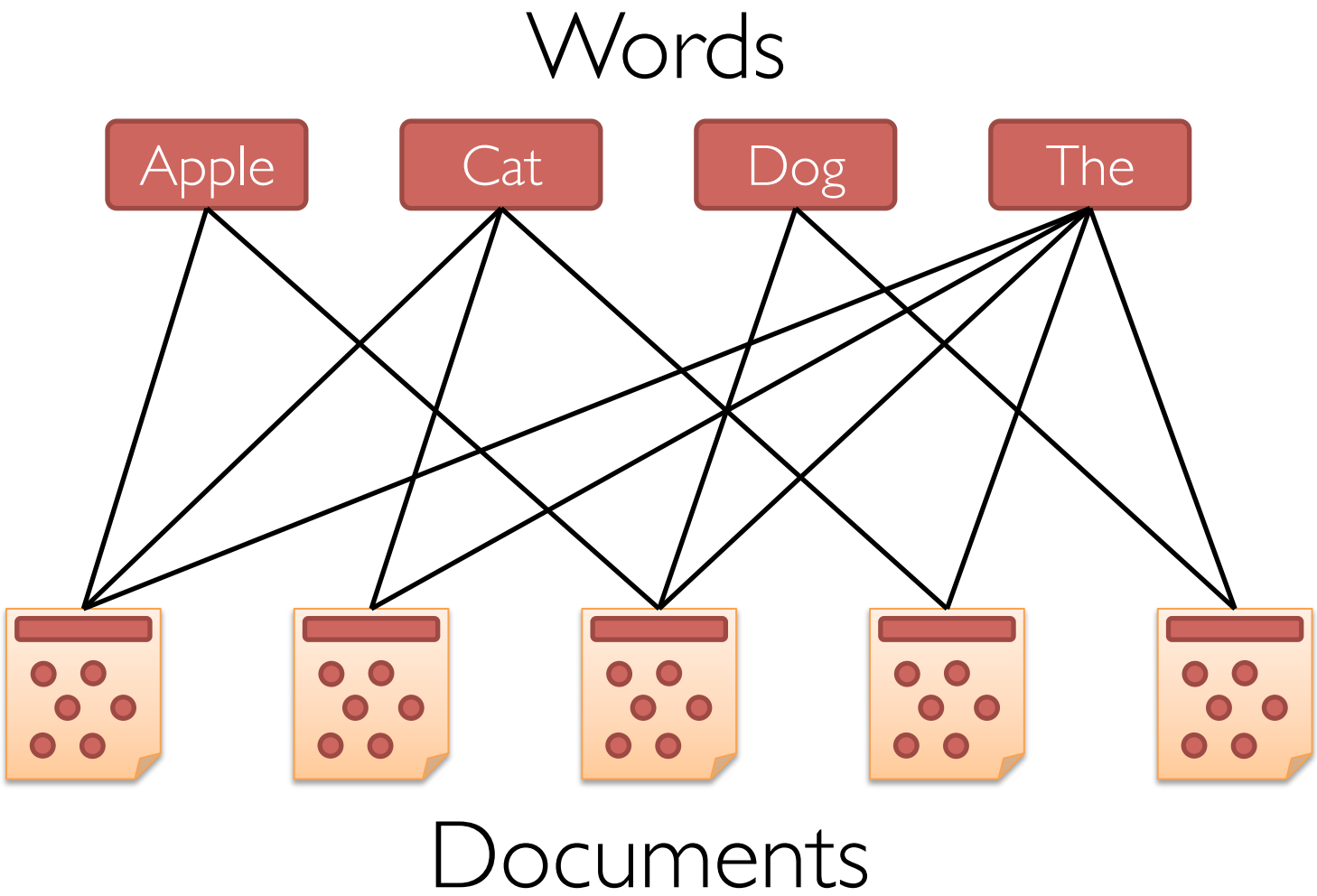
Does not address the data/worker management:

- Data-partitioning, recovery, stragglers
- Opportunity: *Dataflow Integration (e.g., Spark)*

Asynchronous model is complicated to debug

Does not capture static dependency structure between data and parameters

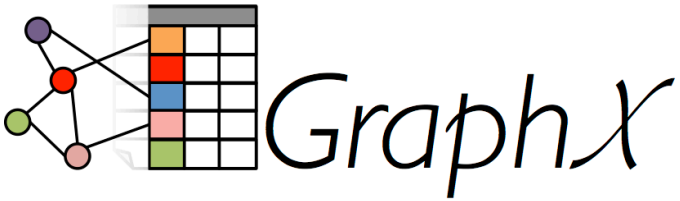
Static Data Dependencies



Outline of the Tutorial

1. Distributed Aggregation: Map-Reduce
2. Iterative Machine Learning: Spark
3. Large Shared Models: Parameter Server
4. Graphical Computation: GraphLab to GraphX

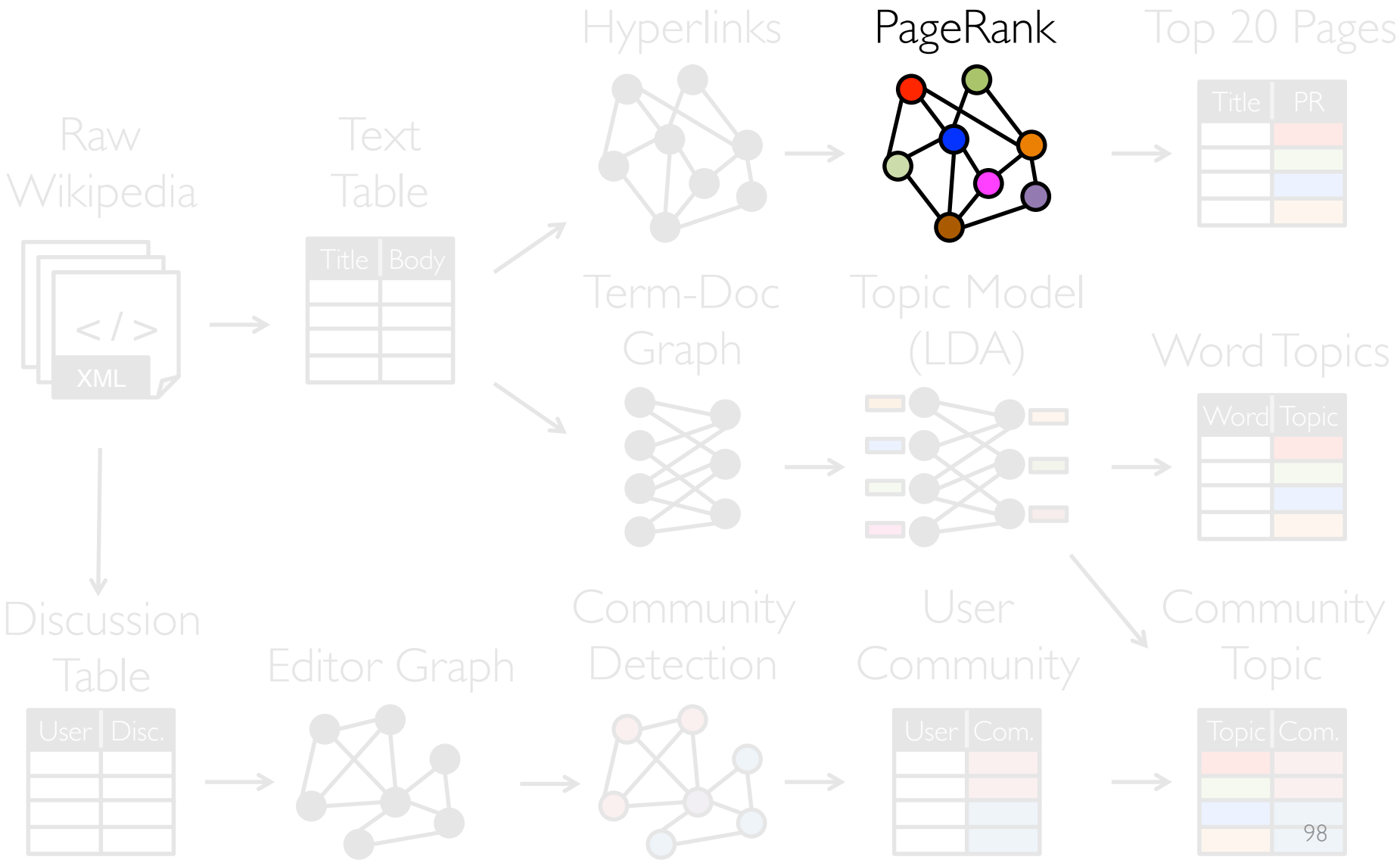
Graph-Structured Big Models



R. Xin, J. Gonzalez, M. Franklin, I. Stoica., *GraphX: A Resilient Distributed Graph System on Spark.* SIGMOD GRADES'13

J. Gonzalez, Y. Low, H. Gu, D. Bickson, C. Guestrin. *PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs.* OSDI'12

Graphs are Central to Analytics



PageRank: Identifying Leaders

$$R[i] = 0.15 + \sum_{j \in \mathrm{Nbrs}(i)} w_{ji} R[j]$$

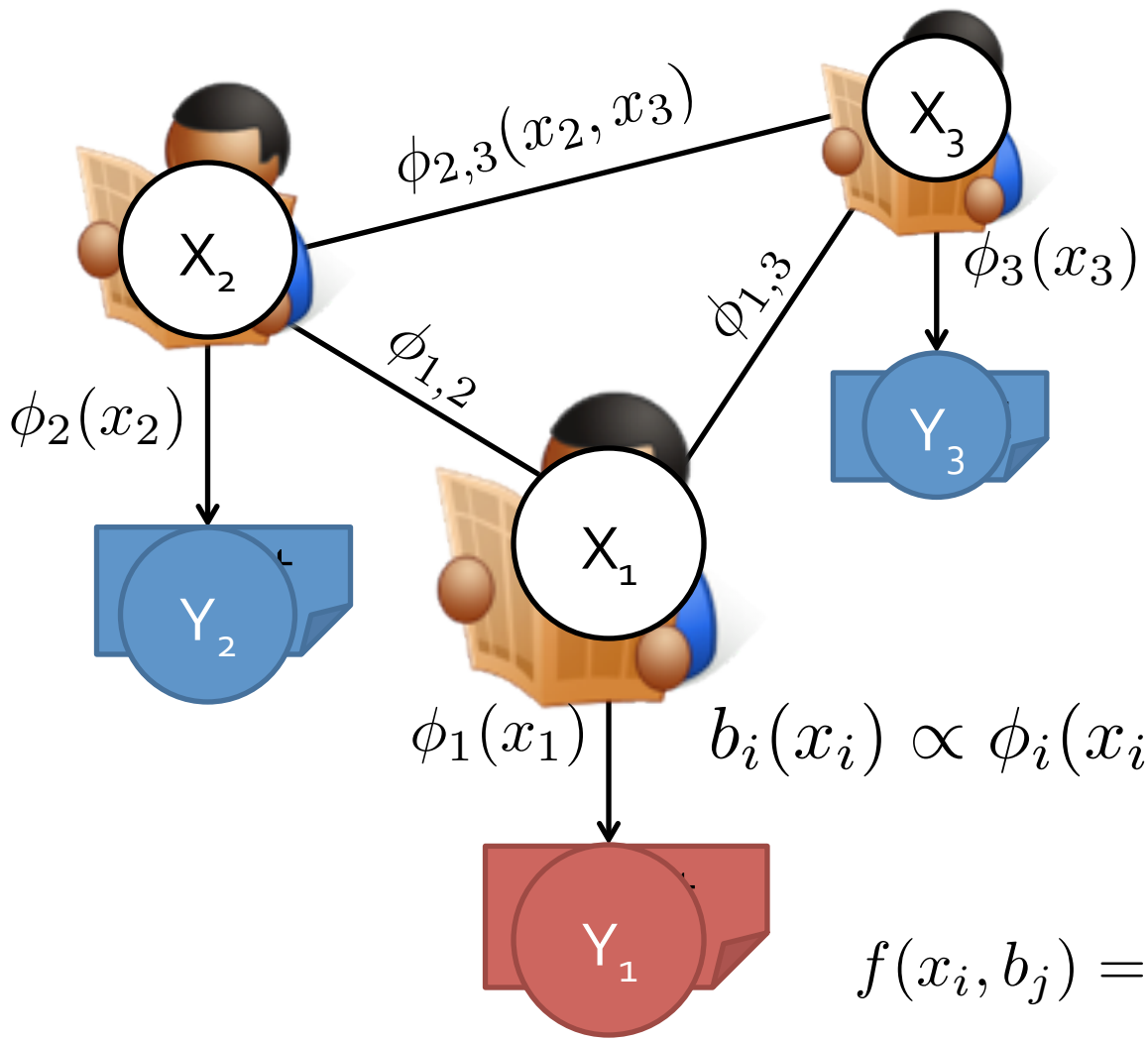
Rank of user *i*

Weighted sum of neighbors' ranks

Update ranks in parallel

Iterate until convergence

Mean Field Algorithm

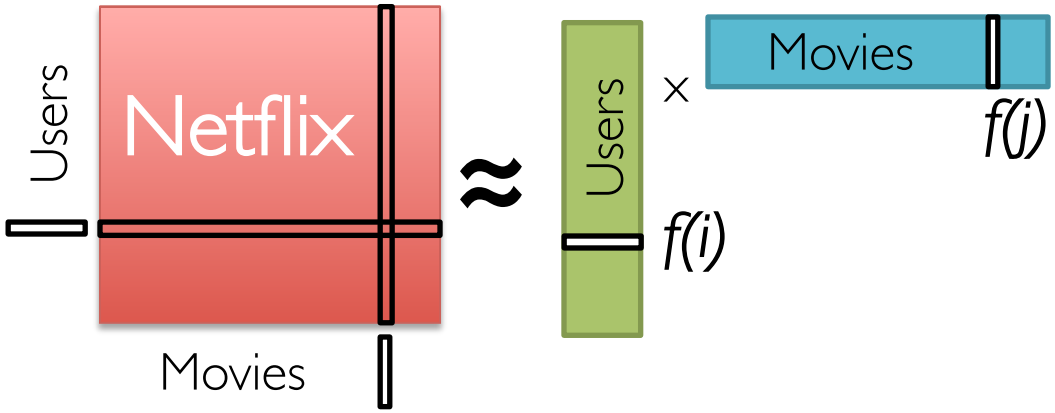


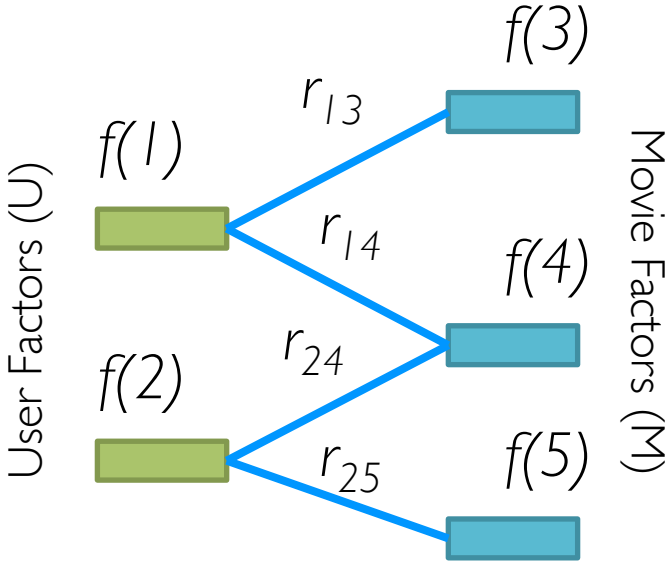
$$b_i(x_i) \propto \phi_i(x_i) \exp\left(\overbrace{\sum_{j \in N_i} f(x_i, b_j)}^{\text{Sum over Neighbors}}\right)$$

$$f(x_i, b_j) = \sum_{x_j} b_j(x_j) \log \phi_{i,j}(x_i, x_j)$$

Recommending Products

Low-Rank Matrix Factorization:

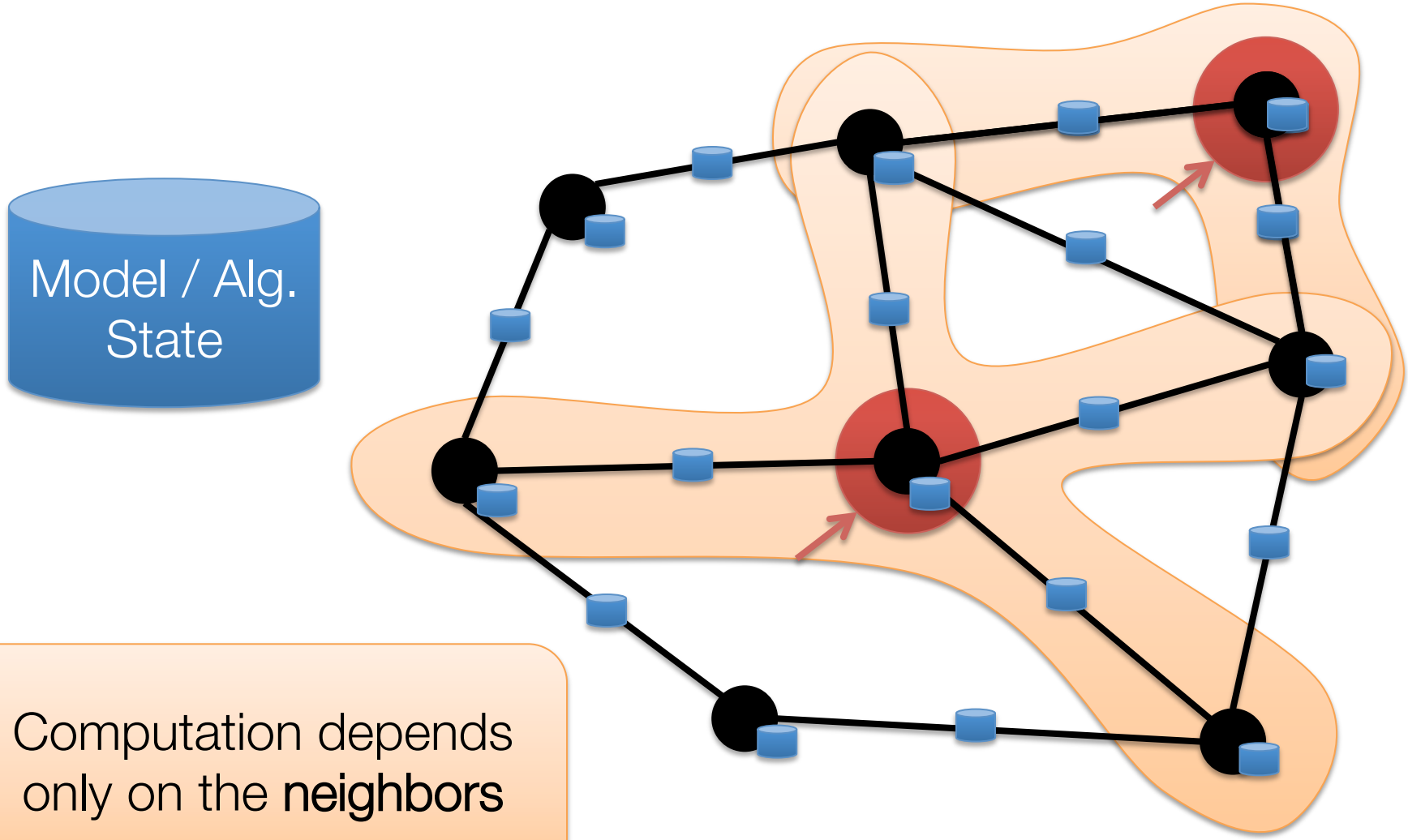




Iterate:

$$f[i] = \arg\min_{w \in \mathbb{R}^d} \sum_{j \in \mathrm{Nbrs}(i)} \left(r_{ij} - w^T f[j]\right)^2 + \lambda ||w||_2^2$$

The Graph-Parallel Pattern



Computation depends only on the **neighbors**

Many Graph-Parallel Algorithms

- Collaborative Filtering
  - Alternating Least Squares
  - Stochastic Gradient Descent
  - Tensor Factorization
- Structured Prediction
  - Loopy Belief Propagation
  - Max-Product Linear Programs
  - Gibbs Sampling
- Semi-supervised ML
  - Graph SSL
  - CoEM
- Community Detection
  - Triangle Counting
  - K-core Decomposition
  - K-Truss
- Graph Analytics
  - PageRank
  - Personalized PageRank
  - Shortest Path
  - Graph Coloring
- Classification
  - Neural Networks

MACHINE LEARNING

SOCIAL NETWORK ANALYSIS

GRAPH ALGORITHMS

Graph-Parallel Systems

GraphLab Pregel APACHE GIRAPH

Expose specialized APIs to simplify graph programming.

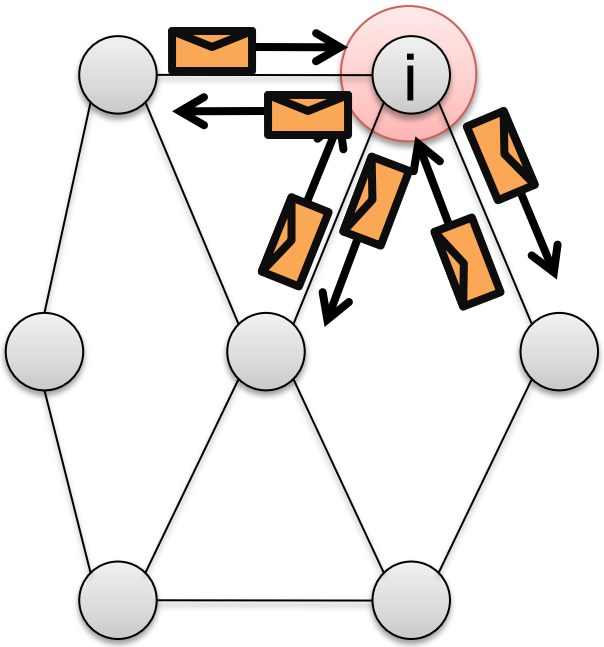
"Think like a Vertex."

- Pregel [SIGMOD'10]

The Pregel (Push) Abstraction

Vertex-Programs interact by sending **messages**.

```
Pregel_PageRank(i, messages) :
  // Receive all the messages
  total = 0
  foreach( msg in messages) :
    total = total + msg

  // Update the rank of this vertex
  R[i] = 0.15 + total

  // Send new messages to neighbors
  foreach(j in out_neighbors[i]) :
    Send  msg(R[i]) to vertex j
```

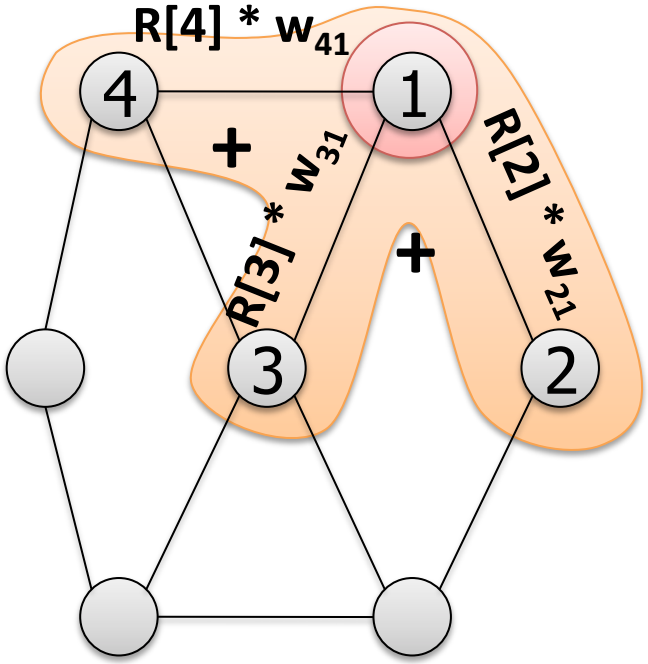


Malewicz et al. [PODC'09, SIGMOD'10]

The GraphLab (Pull) Abstraction

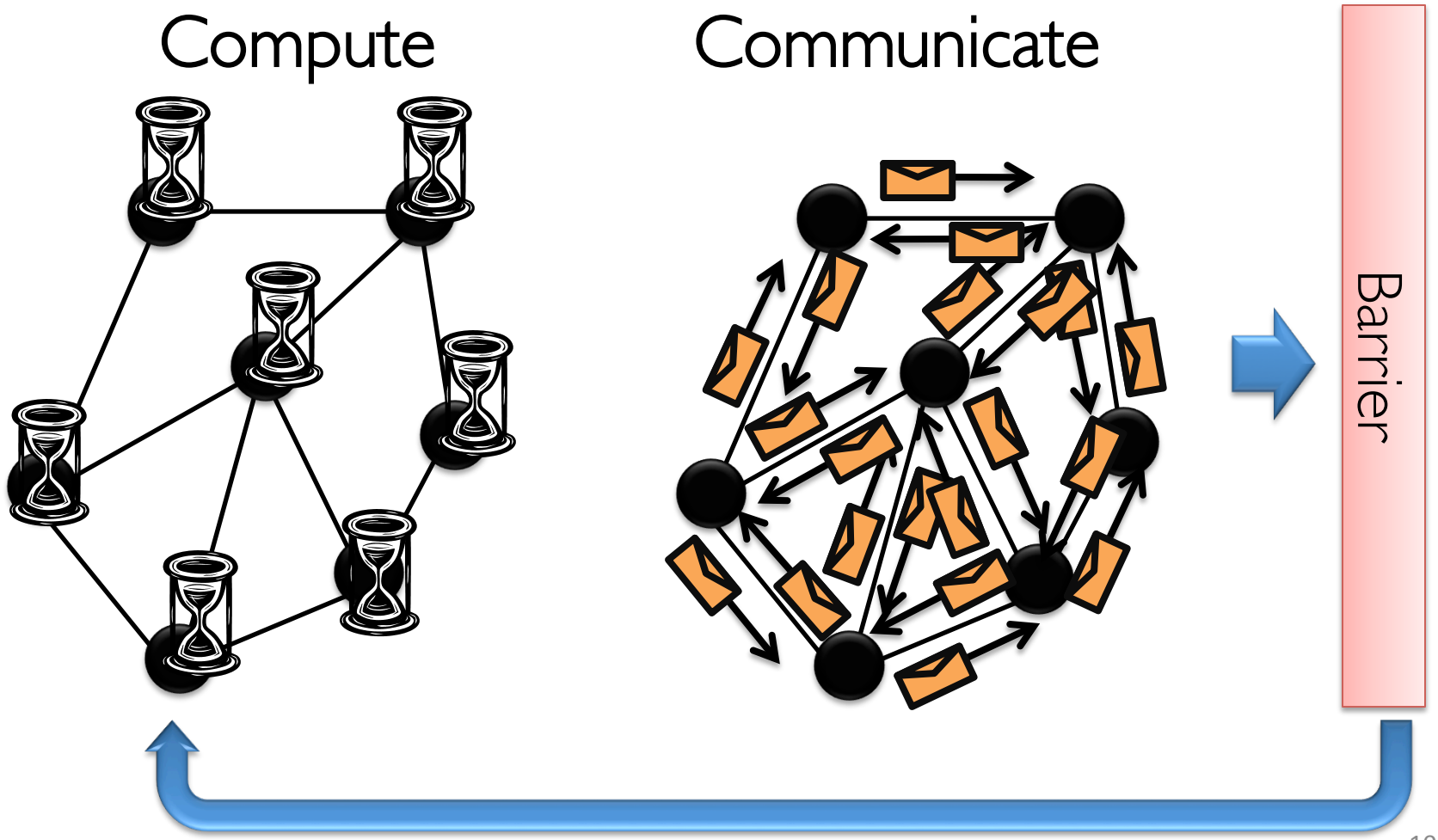
Vertex Programs directly **access** adjacent vertices and edges

```
GraphLab_PageRank(i)
  // Compute sum over neighbors
  total = 0
  foreach( j in neighbors(i)):
    total = total + R[j] * w_ji

  // Update the PageRank
  R[i] = 0.15 + total
```



Data movement is managed by the system and not the user.

Iterative Bulk Synchronous Execution

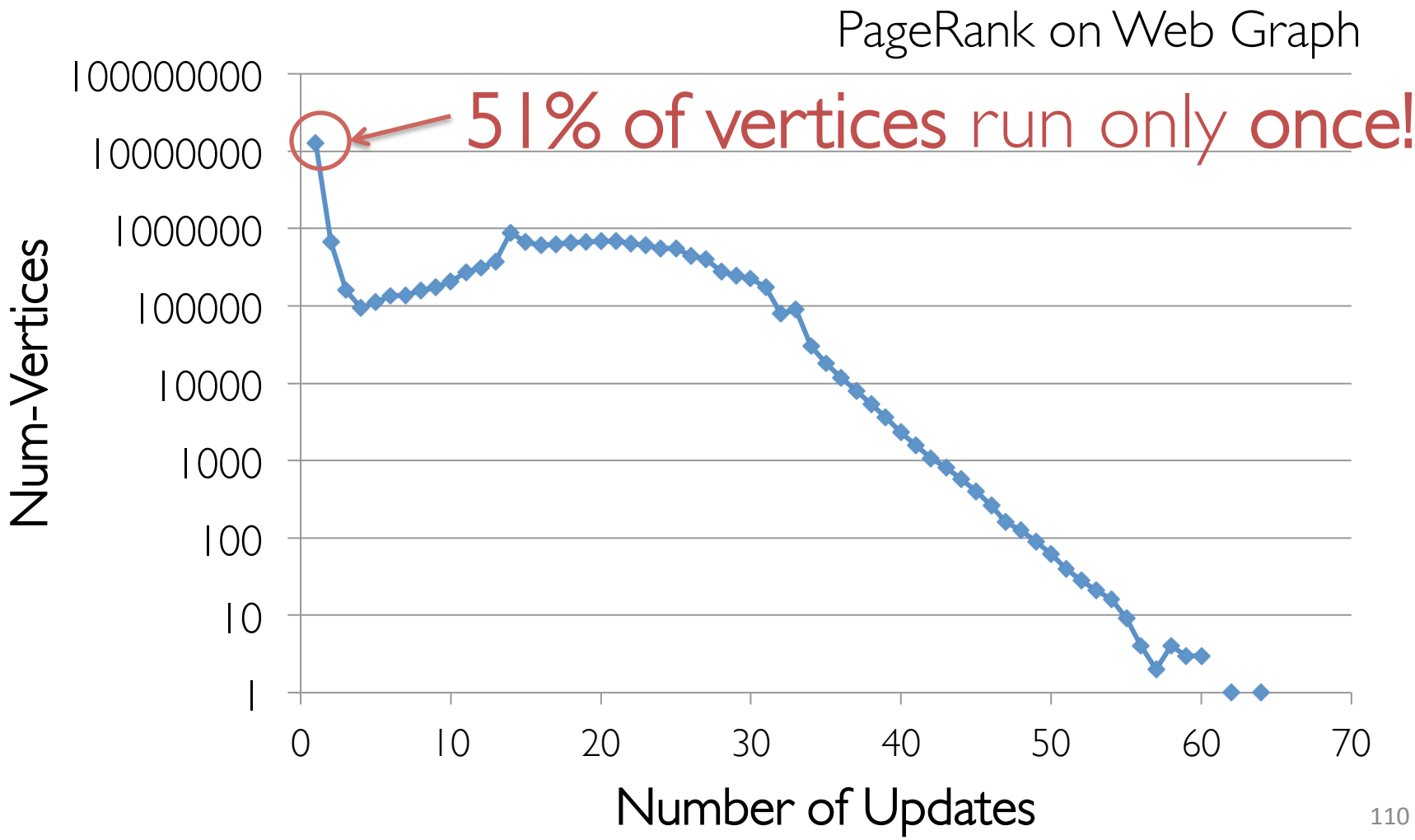


Graph-Parallel Systems

GraphLab Pregel APACHE GIRAPH

Exploit graph structure to achieve orders-of-magnitude performance gains over more general data-parallel systems.

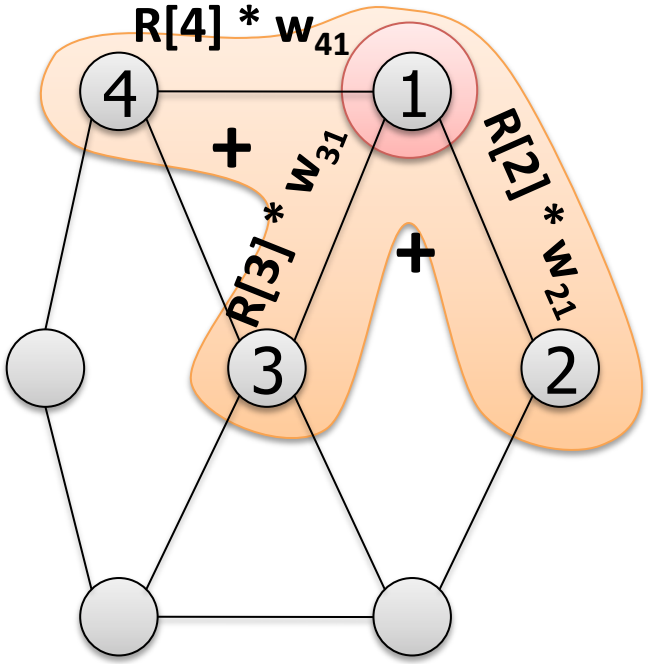
Shrinking Working Sets



The GraphLab (Pull) Abstraction

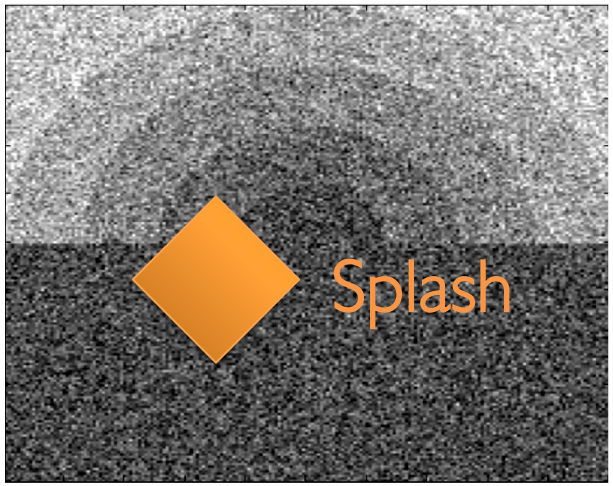
Vertex Programs directly **access** adjacent vertices and edges

```
GraphLab_PageRank(i)
  // Compute sum over neighbors
  total = 0
  foreach( j in neighbors(i)):
    total = total + R[j] * w_ji

  // Update the PageRank
  R[i] = 0.15 + total

  // Trigger neighbors to run again
  if R[i] not converged then
   signal nbrsOf(i) to be recomputed
```

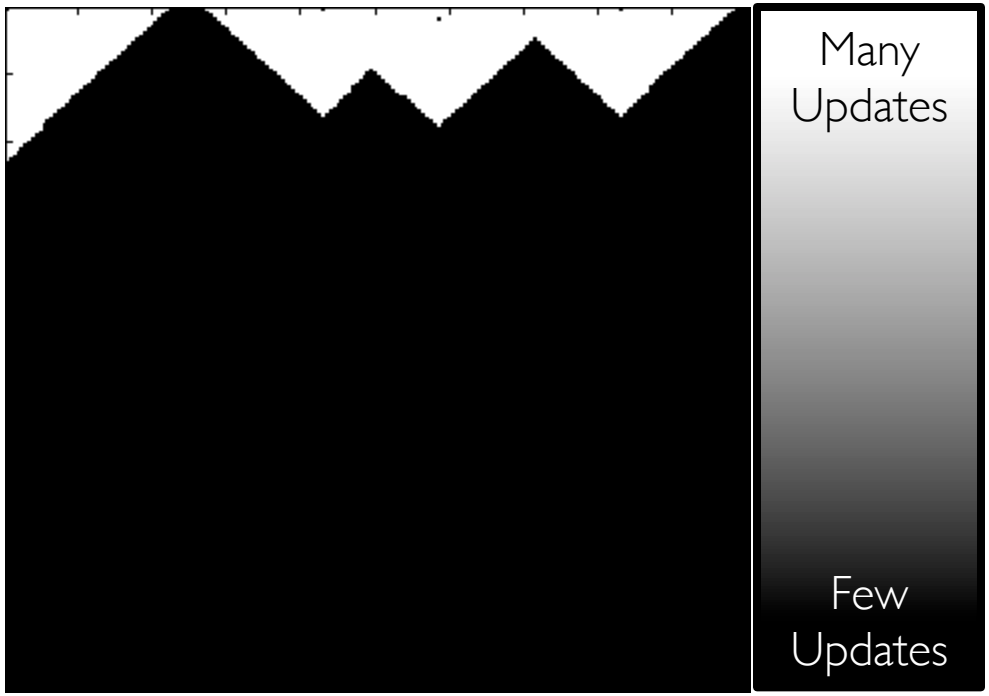


Trigger computation *only* when necessary.

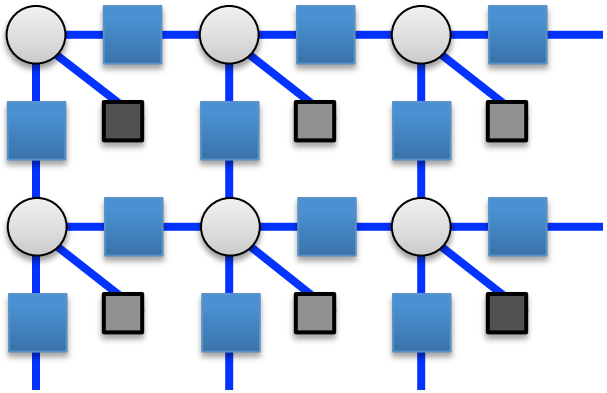
Shrinking Working Sets in Graphical Model Inference



Synthetic Noisy Image



Vertex Updates

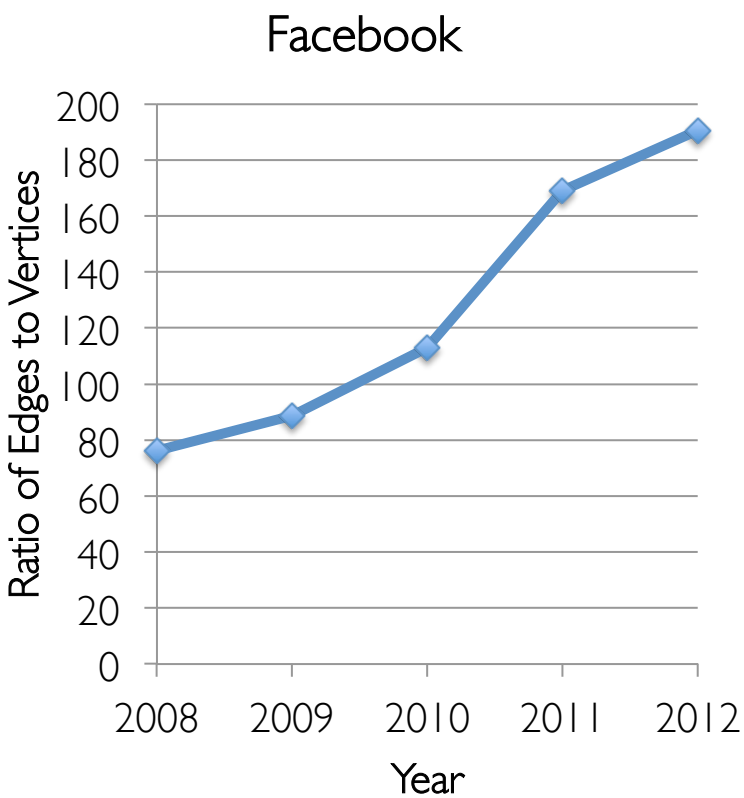


Factor Graph

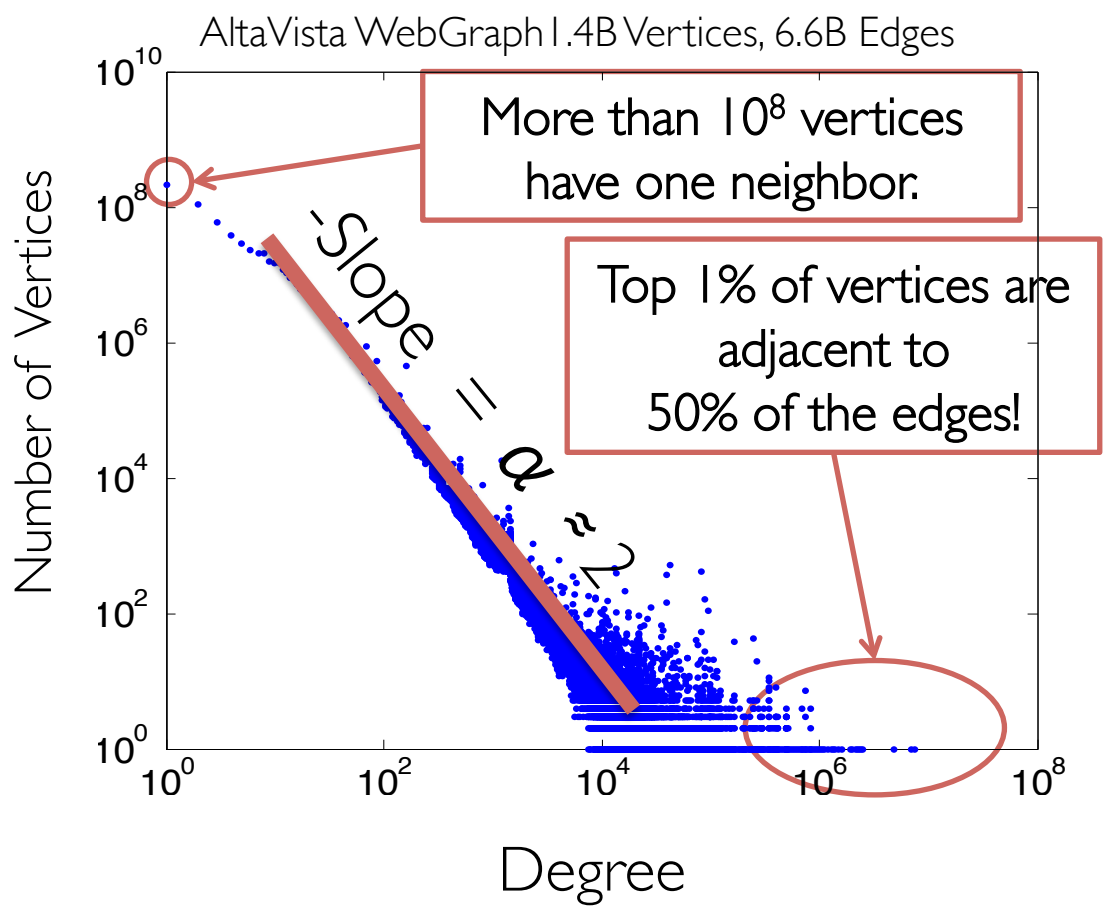
Algorithm identifies and focuses on hidden sequential structure

Real-World Graphs

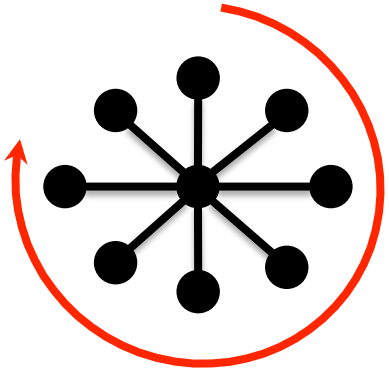
Edges >> Vertices



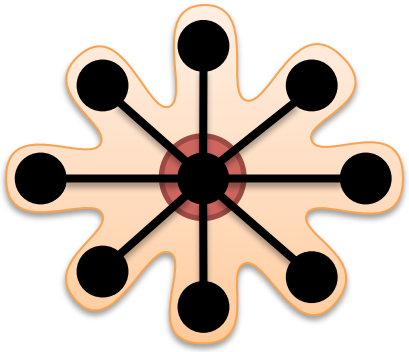
Power-Law Degree Distribution



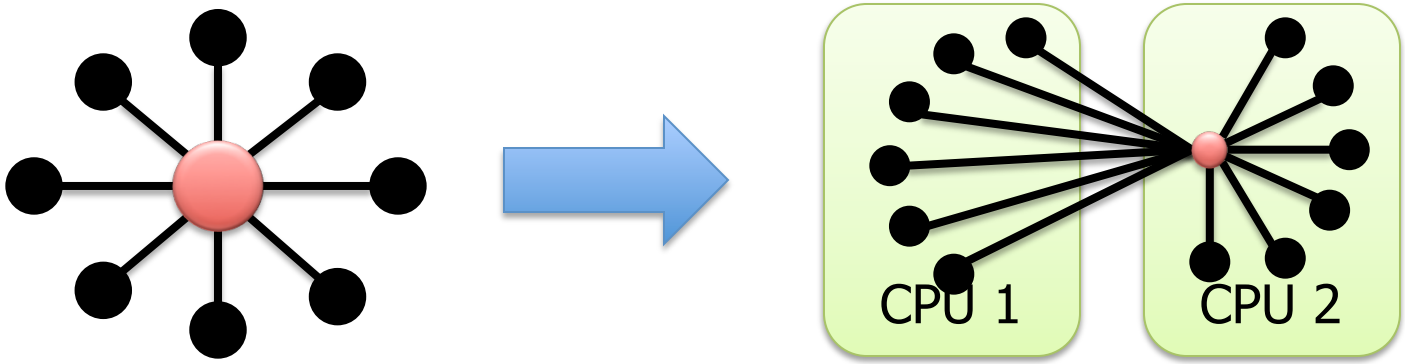
Challenges of High-Degree Vertices



Sequentially process edges



Touches a large fraction of graph



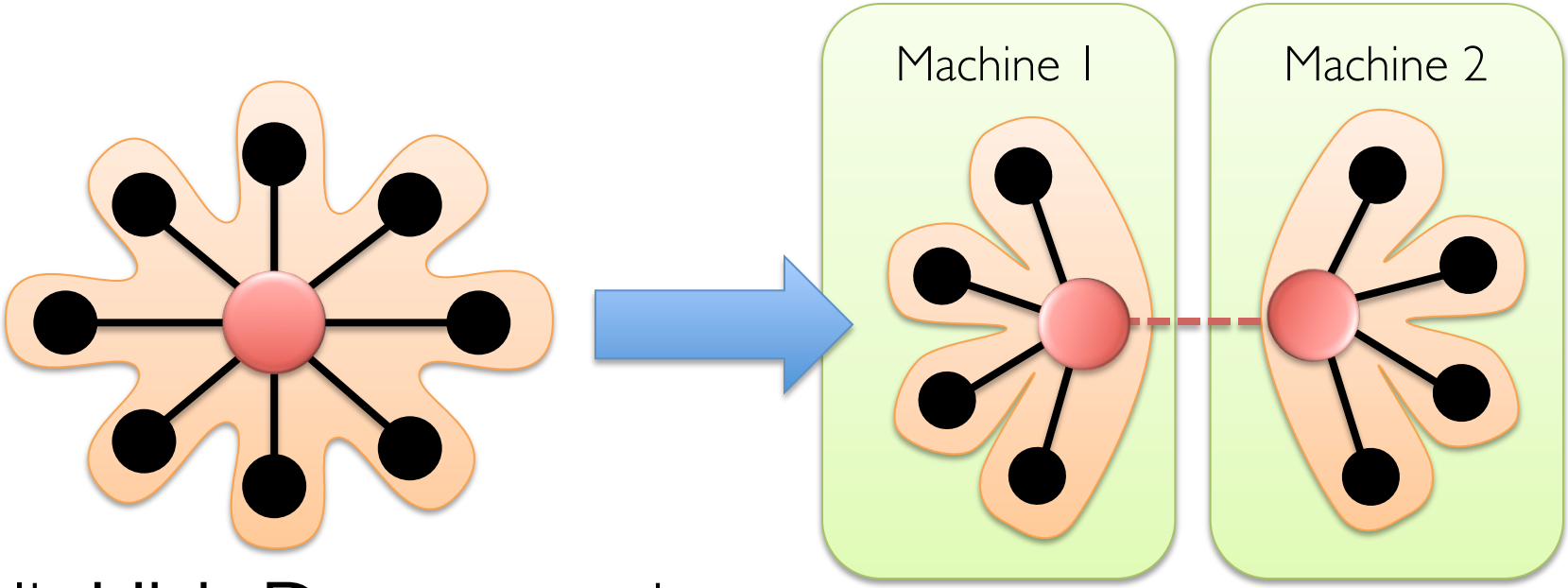
Provably Difficult to Partition

GraphLab

(PowerGraph, OSDI'12)

Program This

Run on This



Split **High-Degree** vertices

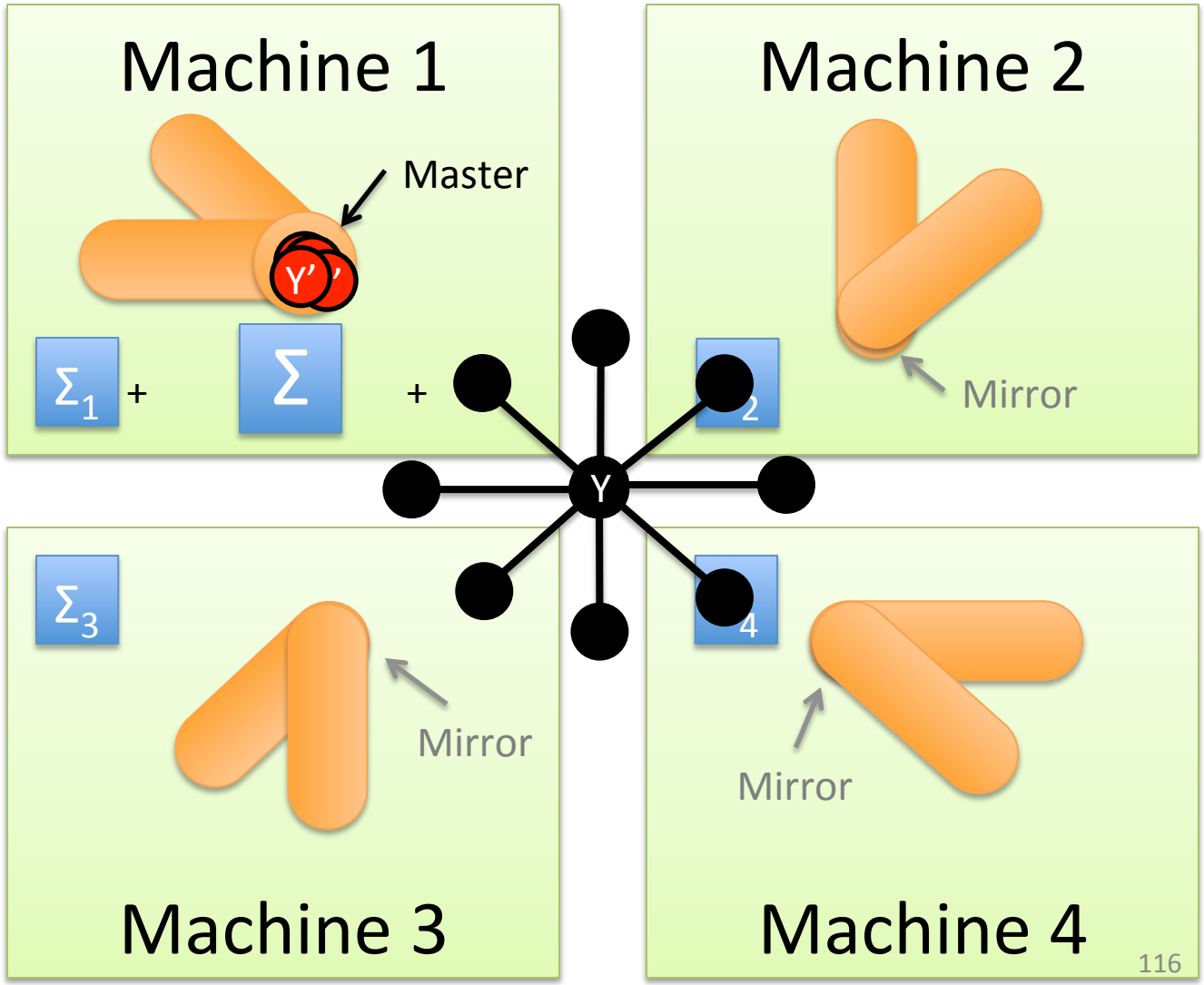
New Abstraction → *<u>Equivalence</u> on Split Vertices*

GAS Decomposition

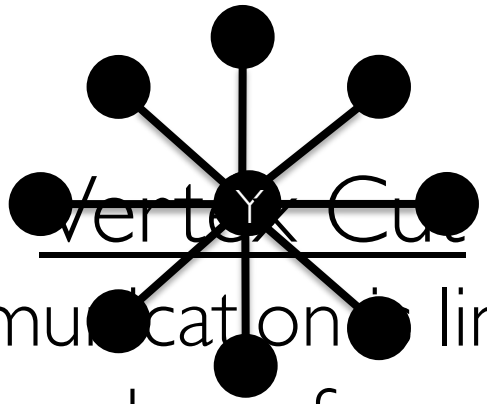
Gather

Apply

Scatter



Minimizing Communication in PowerGraph



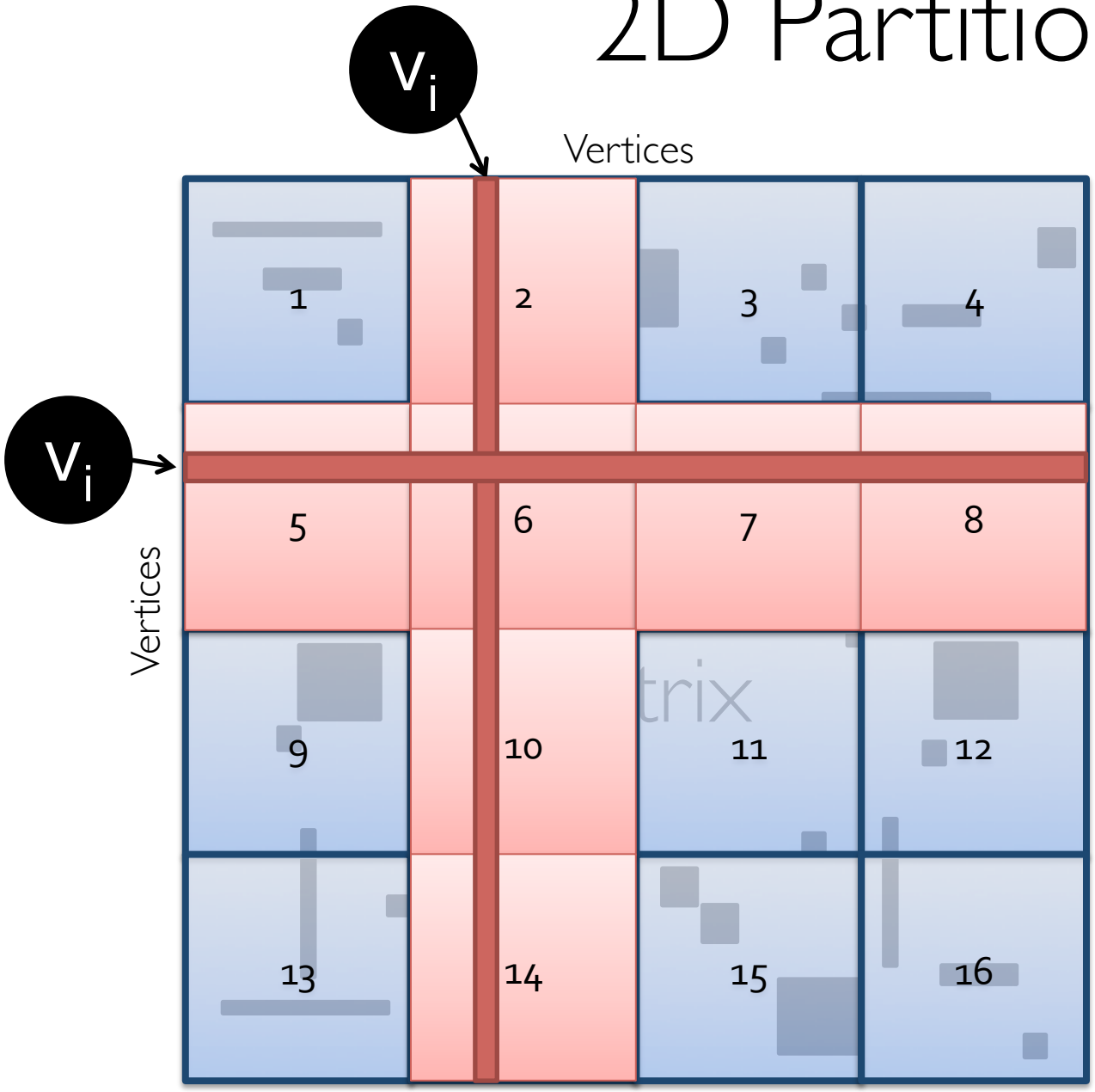
Vertex Cut

Communication is linear in the number of machines each vertex spans.

Total communication upper bound:

$$O\left(\#\text{vertices}\sqrt{\#\text{machines}}\right)$$

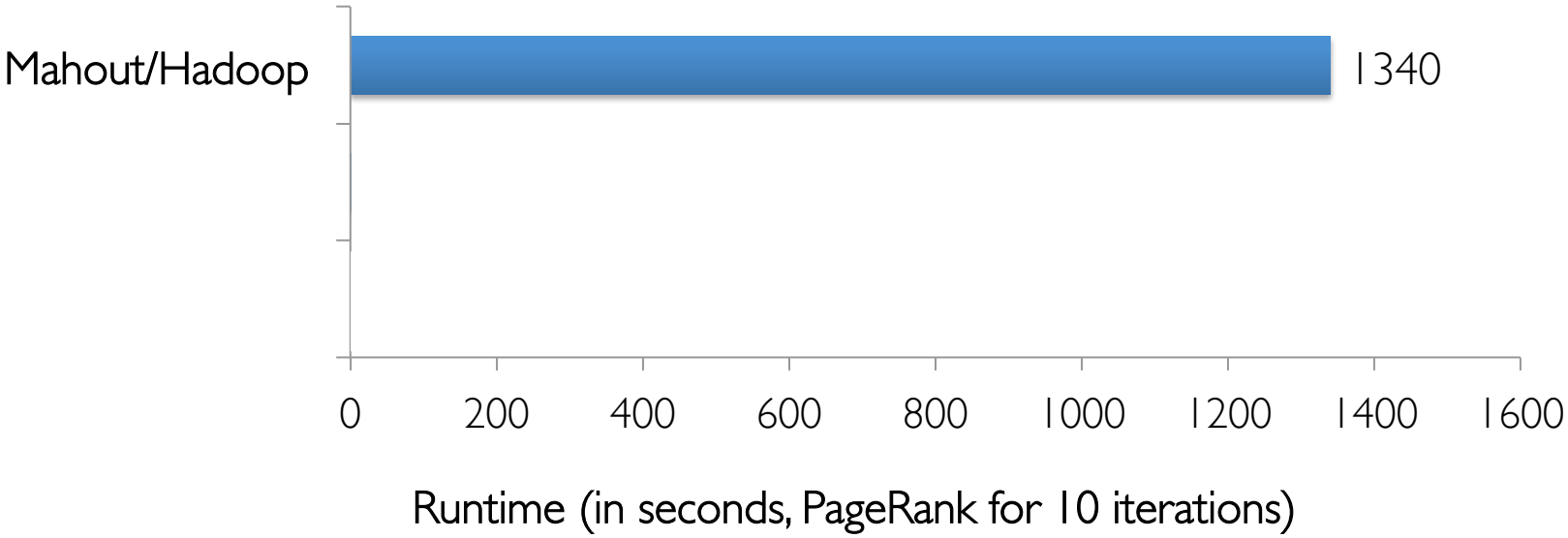
2D Partitioning



16 Machines

v_i only has neighbors on 7 machines

PageRank on the Live-Journal Graph



Spark is *4x faster* than Hadoop
GraphLab is *16x faster* than Spark

Triangle Counting on Twitter

40M Users, 1.4 Billion Links

Counted: 34.8 Billion Triangles

Hadoop [WWW'11]

1536 Machines
423 Minutes

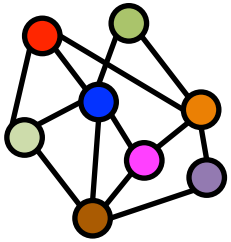
GraphLab

64 Machines
15 Seconds

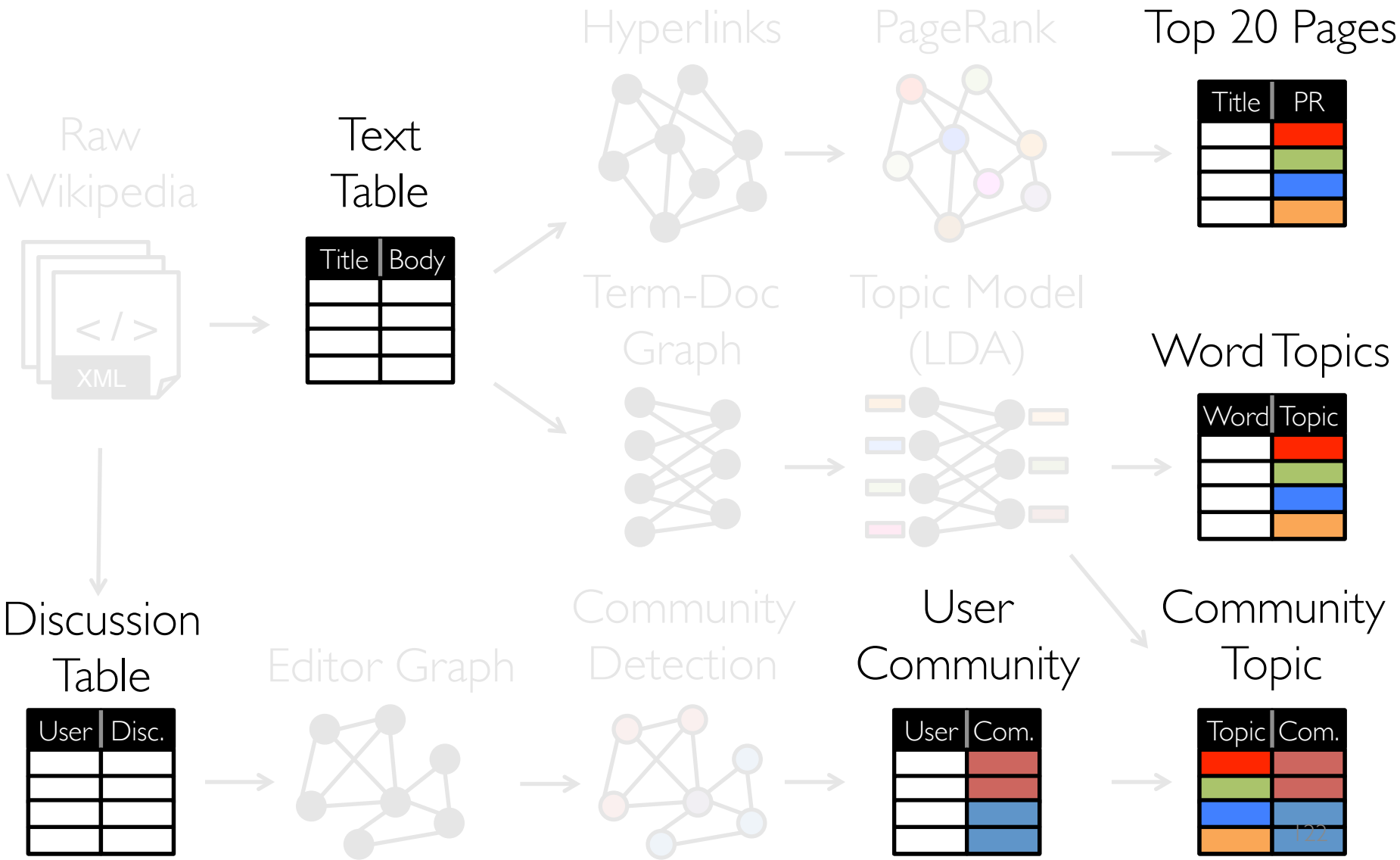
1000 x Faster

S. Suri and S. Vassilvitskii, "Counting triangles and the curse of the last reducer," WWW'11

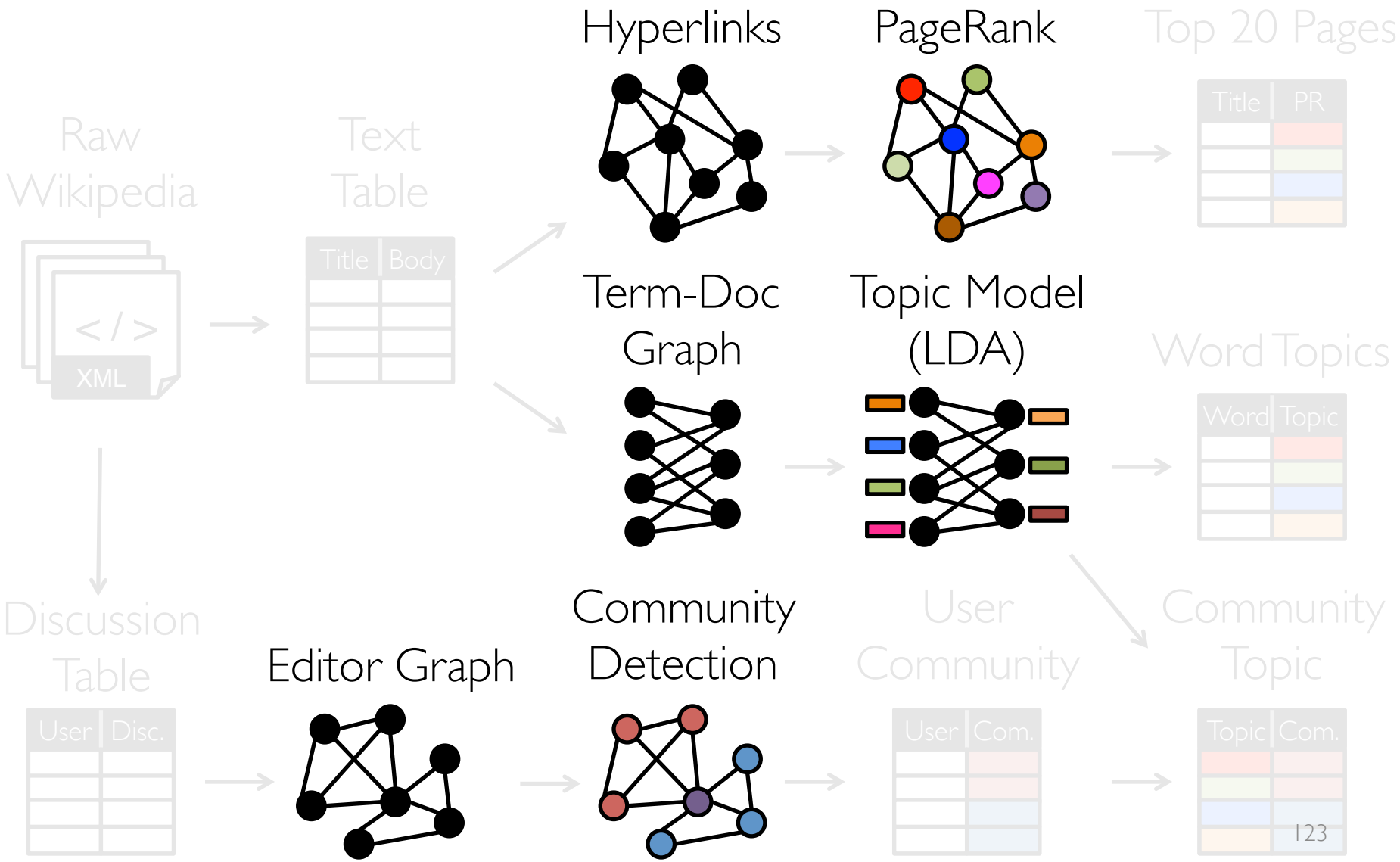
PageRank



Tables

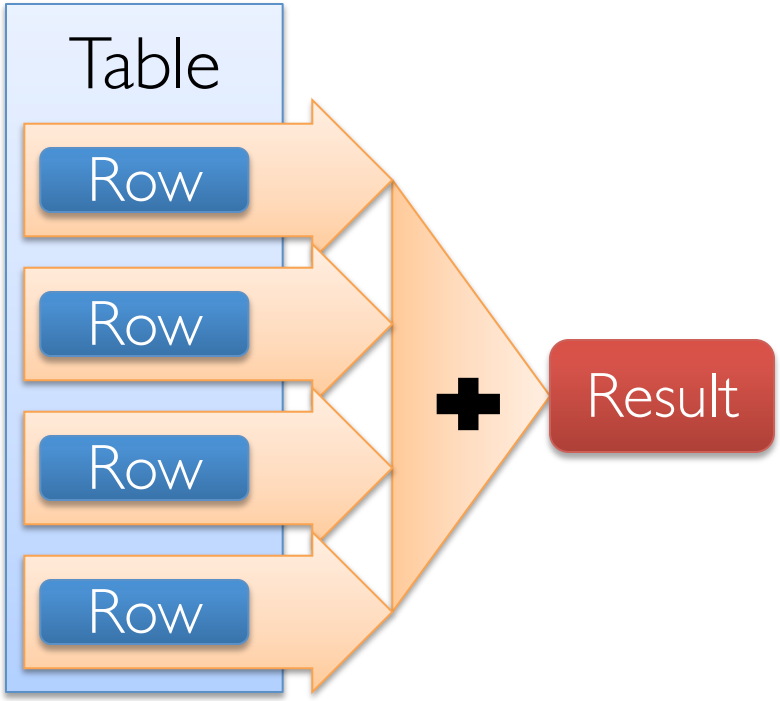

Raw Wikipedia
XML
Text Table
Title
Body
Hyperlinks
PageRank
Top 20 Pages
Title
PR
Term-Doc Graph
Topic Model (LDA)
Word Topics
Word
Topic
Discussion Table
User
Disc.
Editor Graph
Community Detection
User Community
User
Com.
Community Topic
Topic
Com.

Graphs

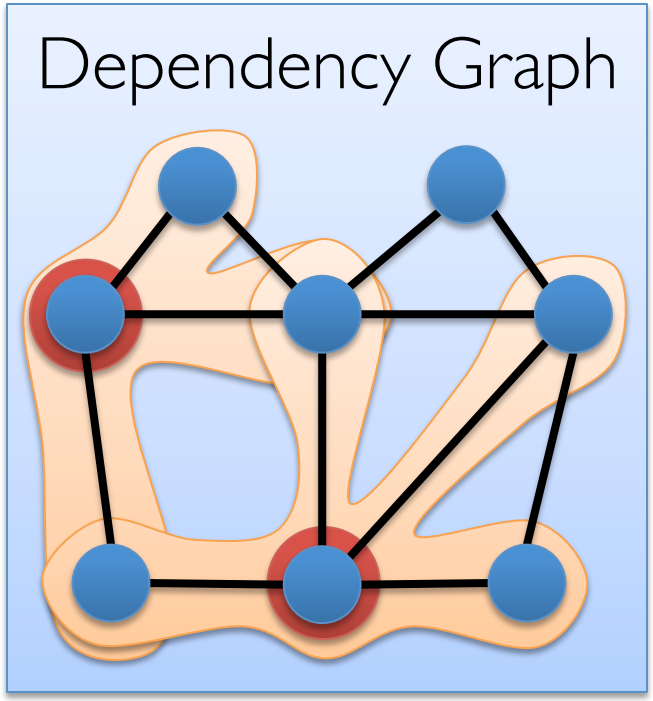

Hyperlinks
PageRank
Top 20 Pages
Title
PR
Raw Wikipedia
XML
Text Table
Title
Body
Term-Doc Graph
Topic Model (LDA)
Word Topics
Word
Topic
Discussion Table
User
Disc.
Editor Graph
Community Detection
User Community
User
Com.
Community Topic
Topic
Com.

Separate Systems to Support Each View

Table View



Graph View



Having separate systems for each view is difficult to use and inefficient

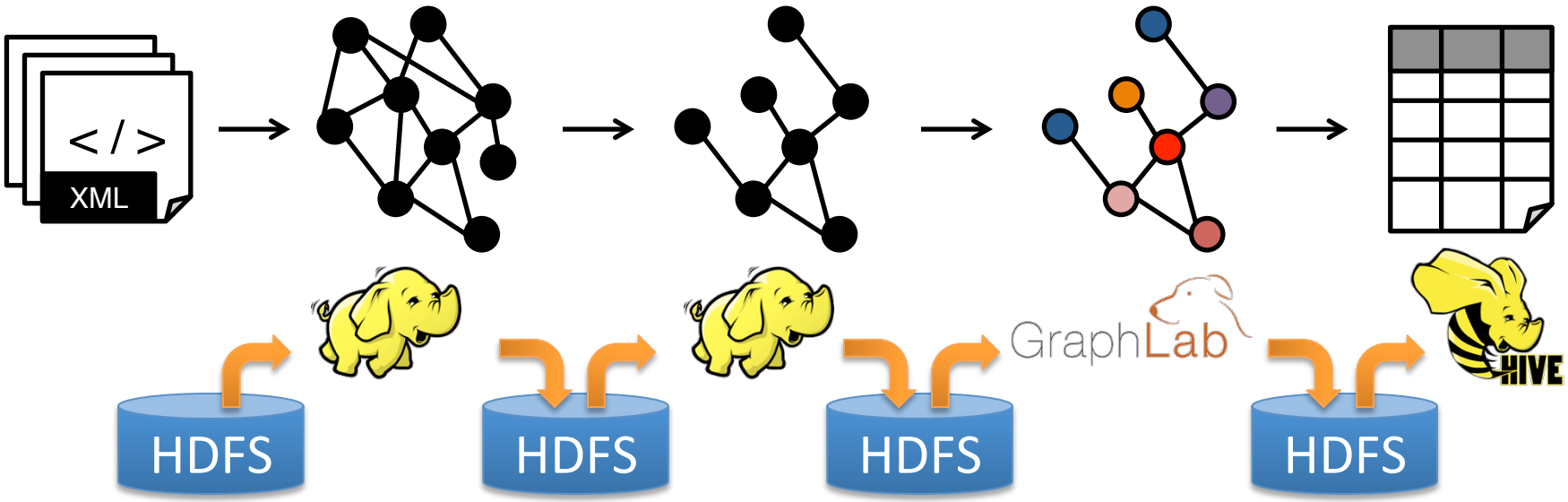
Difficult to Program and Use

Users must *Learn*, *Deploy*, and *Manage* multiple systems

Leads to brittle and often complex interfaces

Inefficient

Extensive data movement and duplication across the network and file system



Limited reuse internal data-structures across stages

GraphX Solution: Tables and Graphs are *views* of the *same physical* data

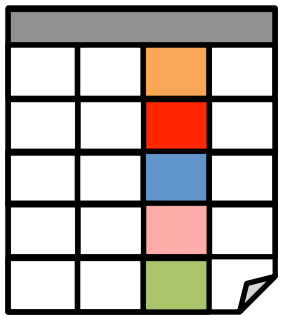
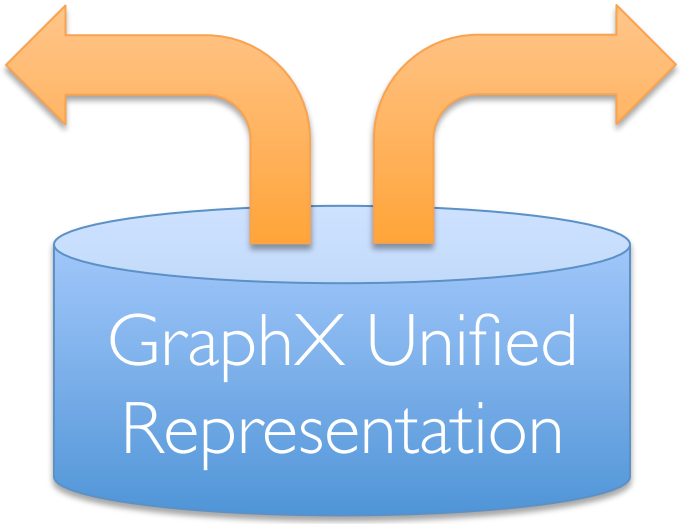
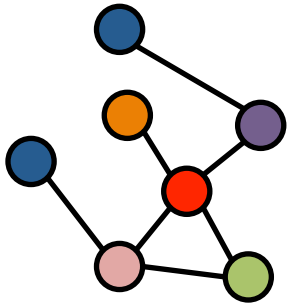


Table View





Graph View

Each view has its own operators that exploit the semantics of the view to achieve efficient execution

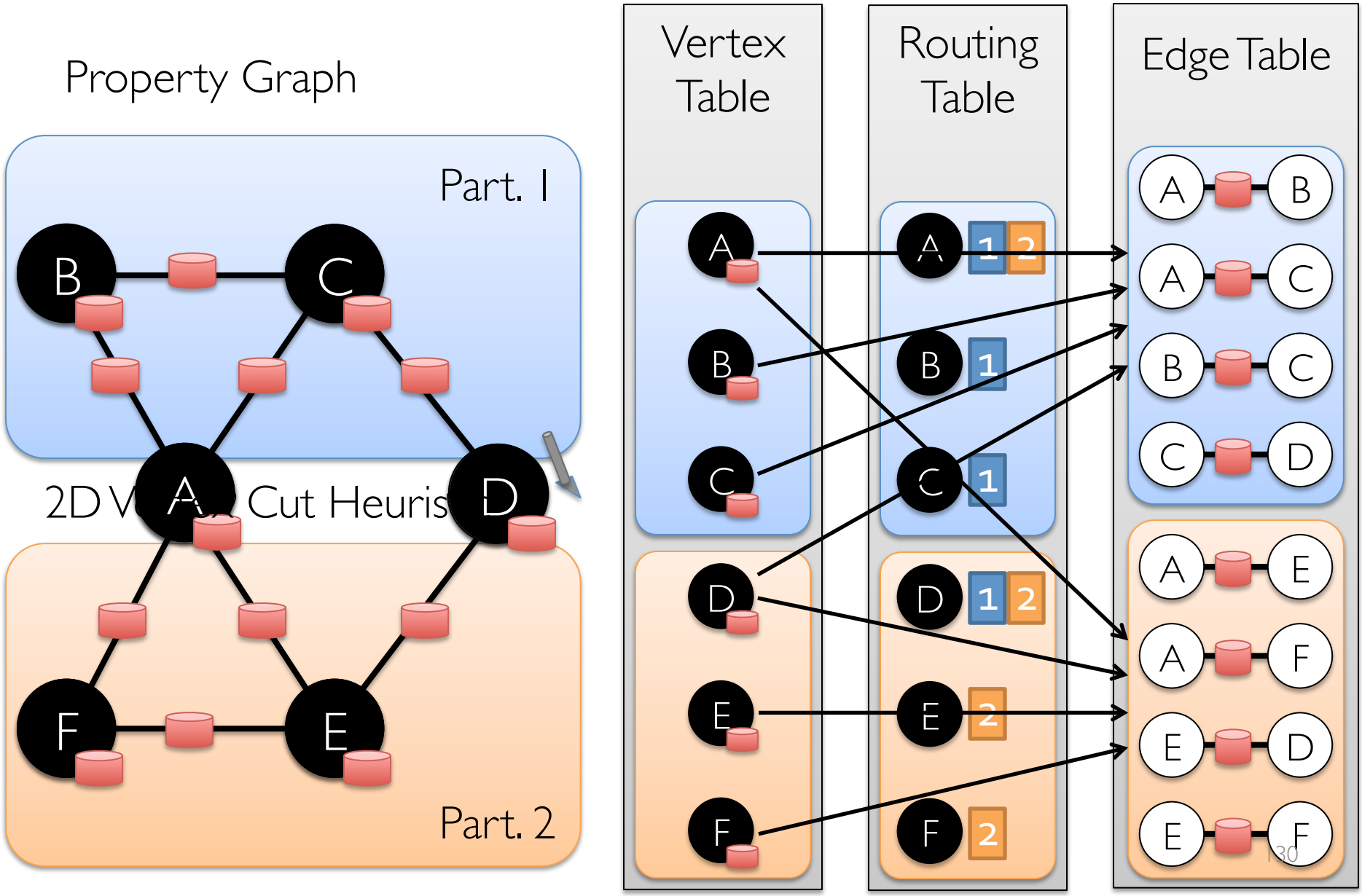
Graphs → Dataflow

1. Encode graphs as distributed tables (RDDs)
2. Express graph computation in relational ops.
3. Recast graph systems optimizations as:
   1. Distributed join optimization
   2. Incremental materialized maintenance

Integrate Graph and Table data processing systems.

Achieve performance parity with specialized systems.

Distributed Graphs as Distributed Tables



Spark Dataflow Operators

Inherited from Spark:

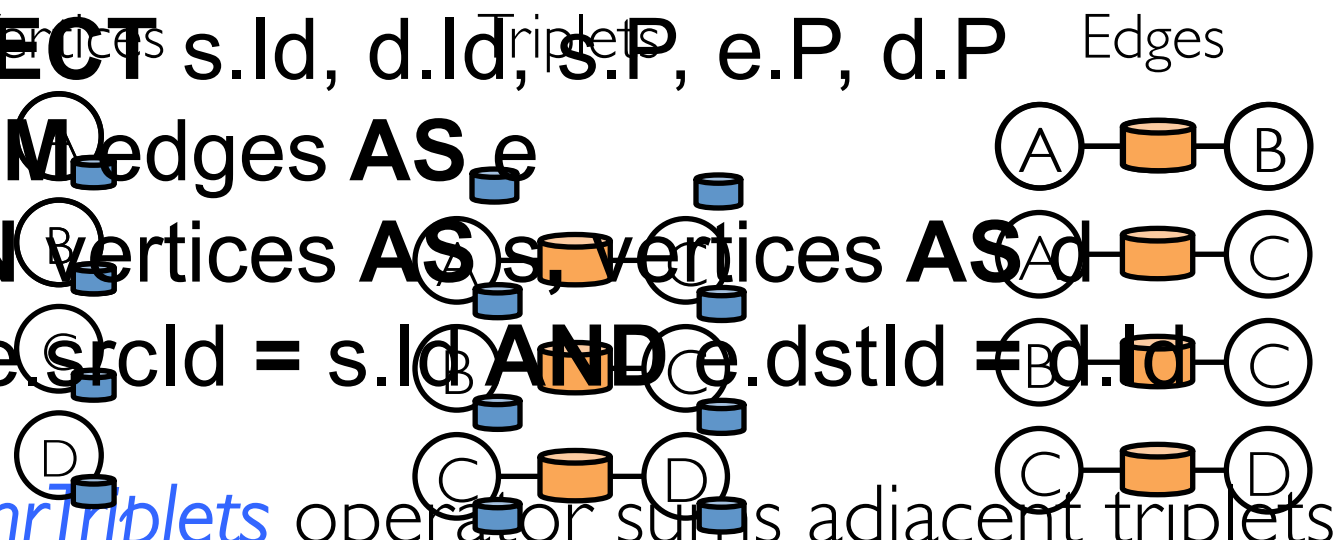
`map`	`reduce`	`sample`
`filter`	`count`	`take`
`groupBy`	`fold`	`first`
`sort`	`reduceByKey`	`partitionBy`
`union`	`groupByKey`	`mapWith`
`join`	`cogroup`	`pipe`
`leftOuterJoin`	`cross`	`save`
`rightOuterJoin`	`zip`	`...`

Graph Operators

```
class Graph [ V, E ] {
  def Graph(vertices: Table[ (Id, V) ],
            edges: Table[ (Id, Id, E) ])
  // Table Views ------------------
  def vertices: Table[ (Id, V) ]
  def edges: Table[ (Id, Id, E) ]
  def triplets: Table [ ((Id, V), (Id, V), E) ]
  // Transformations ------------------------------
  def reverse: Graph[V, E]
  def subgraph(pV: (Id, V) => Boolean,
               pE: Edge[V,E] => Boolean): Graph[V,E]
  def mapV(m: (Id, V) => T ): Graph[T,E]
  def mapE(m: Edge[V,E] => T ): Graph[V,T]
  // Joins ----------------------------------------
  def joinV(tbl: Table [(Id, T)]): Graph[(V, T), E ]
  def joinE(tbl: Table [(Id, Id, T)]): Graph[V, (E, T)]
  // Computation ----------------------------------
  def mrTriplets(mapF: (Edge[V,E]) => List[(Id, T)],
                 reduceF: (T, T) => T): Graph[T, E]
}
```

Triplets Join Vertices and Edges

The *triplets* operator joins vertices and edges:



SELECT s.Id, d.Id, s.P, e.P, d.P
FROM edges **AS** e
JOIN vertices **AS** s, vertices **AS** d
ON e.srcId = s.Id **AND** e.dstId = d.Id

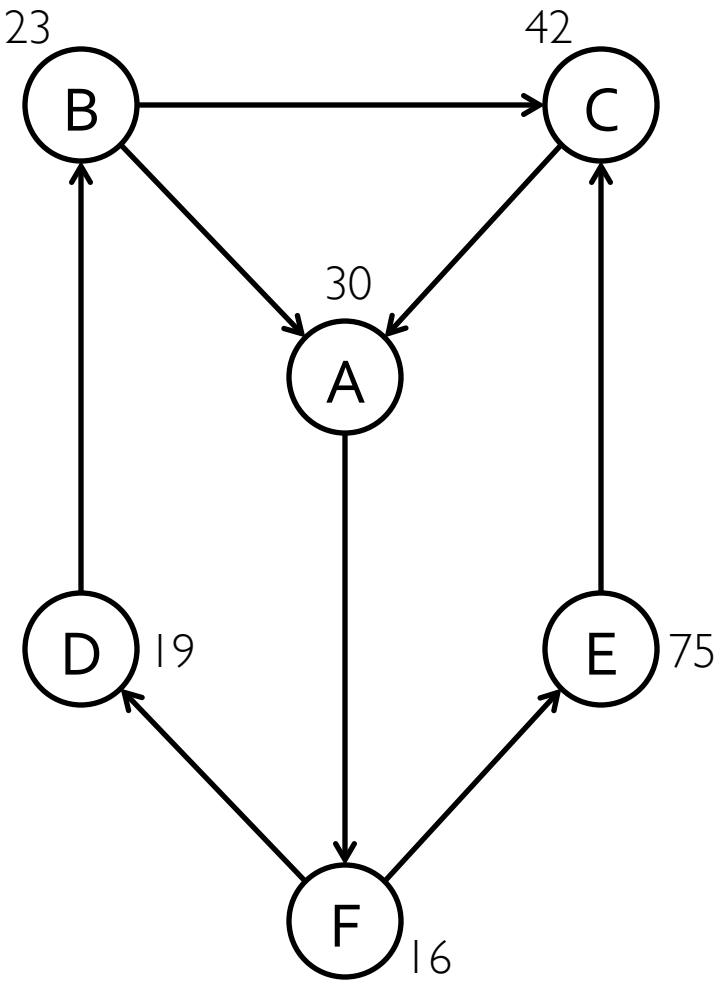
The *mrTriplets* operator sums adjacent triplets.

SELECT t.dstId, *reduce*(*map*(t)) **AS** sum
FROM triplets **AS** t **GROUPBY** t.dstId

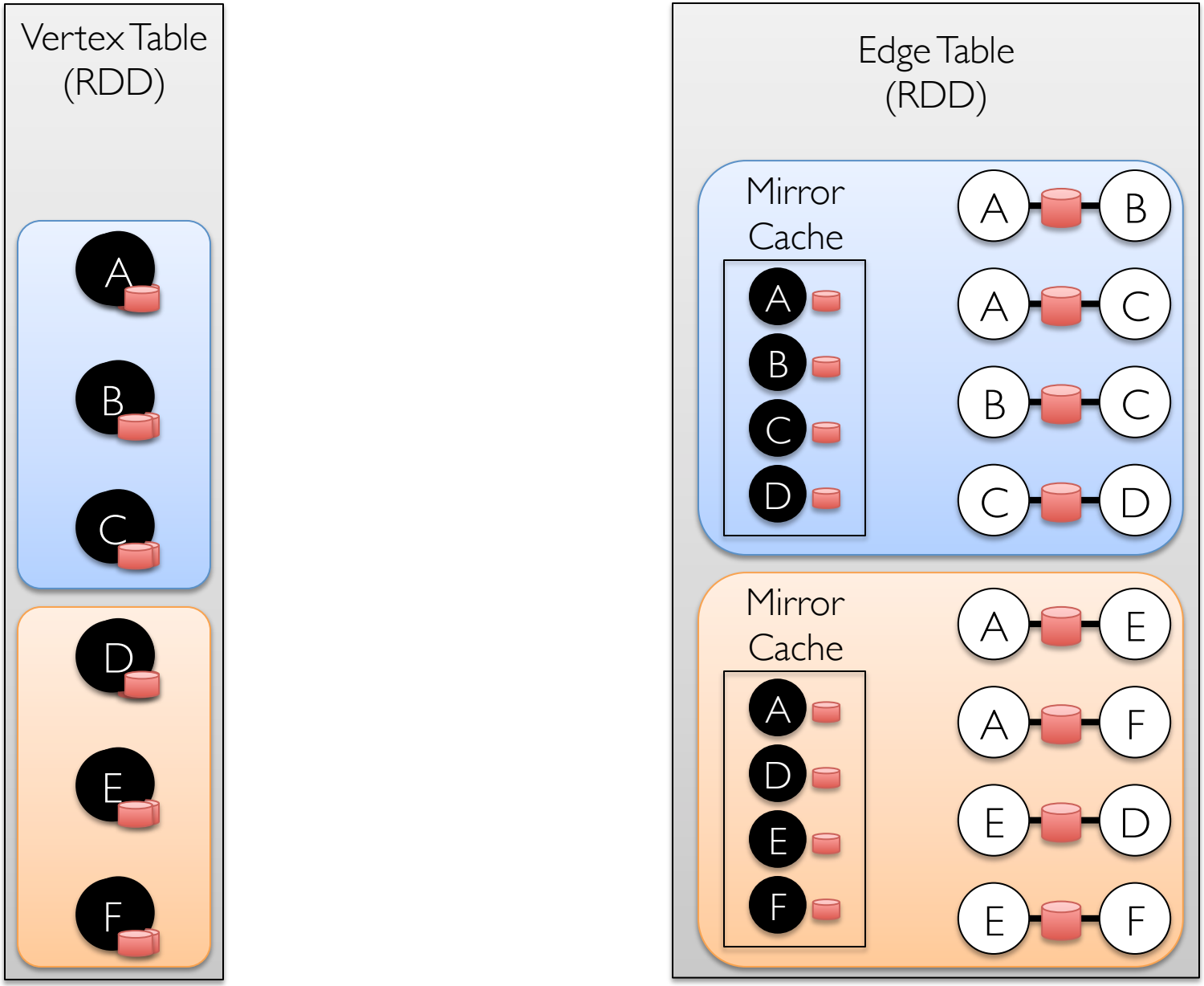
Example: Oldest Follower

Calculate the number of older followers for each user?

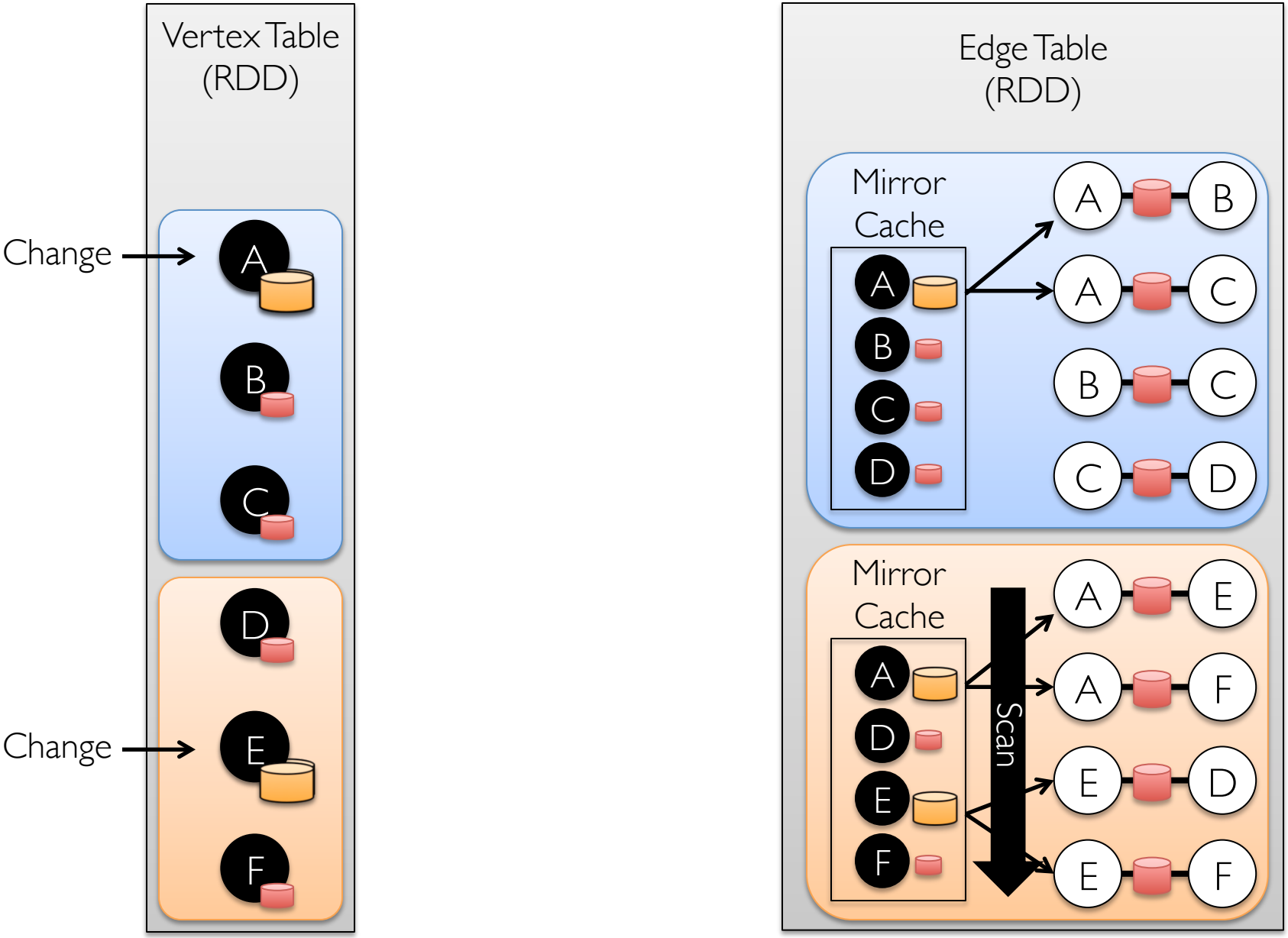
```
val olderFollowerAge = graph
  .mrTriplets(
    e => // Map
     if(e.src.age < e.dst.age) {
       (e.srcId, 1)
     else { Empty }
    ,
    (a,b) => a + b // Reduce
  )
  .vertices
```



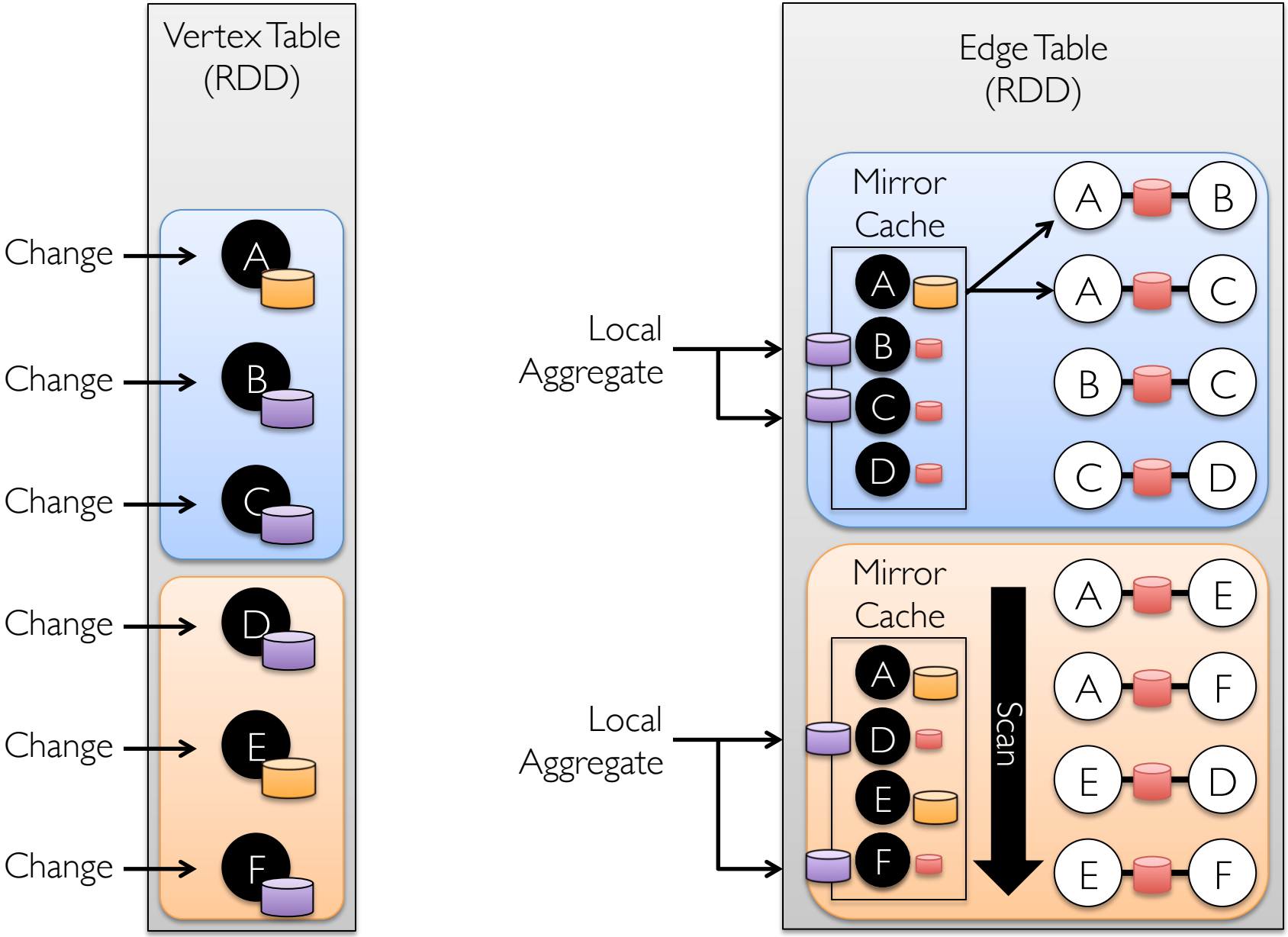
Caching for Iterative mrTriplets



Incremental Updates for Iterative mrTriplets



Aggregation for Iterative mrTriplets



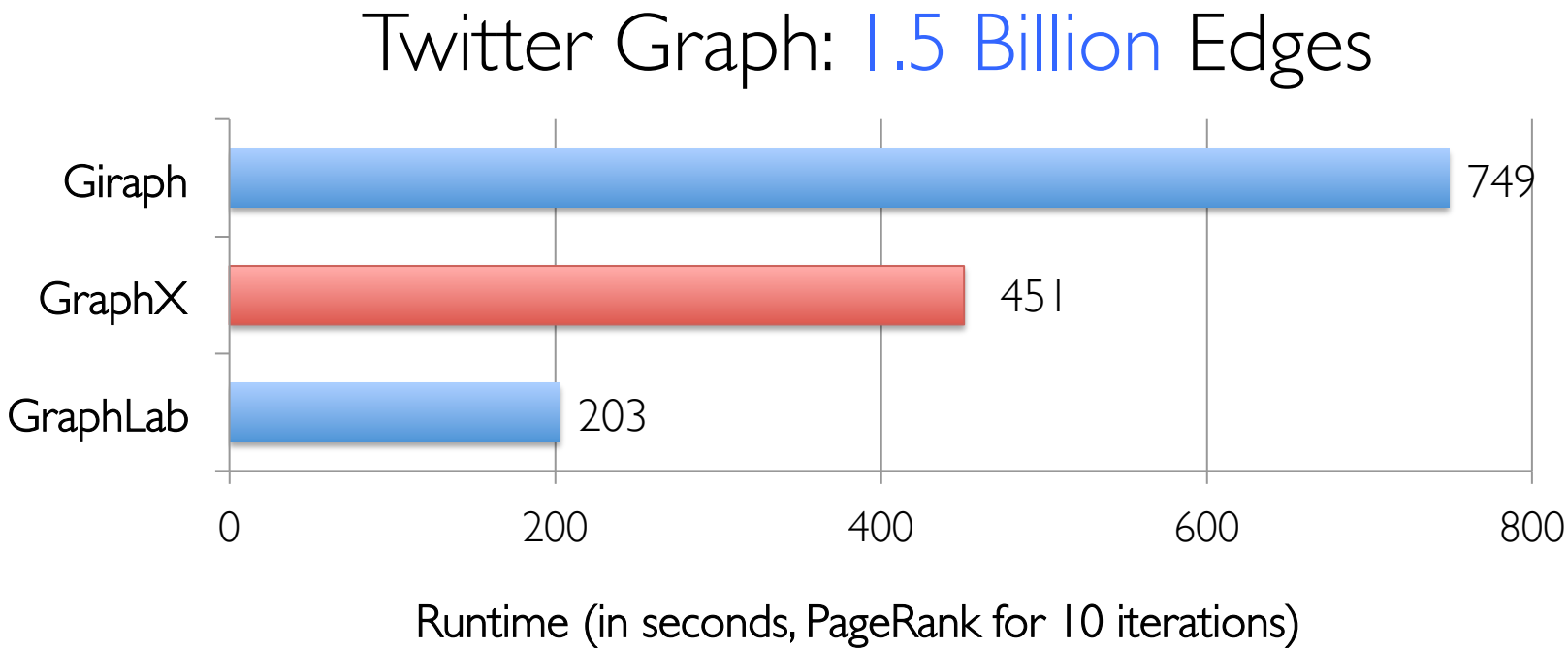
PageRank in GraphX

```
// Load and initialize the graph
val graph = GraphBuilder.text("hdfs://web.txt")
val prGraph = graph.joinVertices(graph.outDegrees)

// Implement and Run PageRank
val pageRank =
  prGraph.pregel(initialMessage = 0.0, iter = 10)(
    (oldV, msgSum) => 0.15 + 0.85 * msgSum,
    triplet => triplet.src.pr / triplet.src.deg,
    (msgA, msgB) => msgA + msgB)
```

Example Analytics Pipeline

```
// Load raw data tables
val verts = sc.textFile("hdfs://users.txt").map(parserV)
val edges = sc.textFile("hdfs://follow.txt").map(parserE)
// Build the graph from tables and restrict to recent links
val graph = new Graph(verts, edges)
val recent = graph.subgraph(edge => edge.date > LAST_MONTH)
// Run PageRank Algorithm
val pr = graph.PageRank(tol = 1.0e-5)
// Extract and print the top 25 users
val topUsers = verts.join(pr).top(25).collect
topUsers.foreach(u => println(u.name + '\t' + u.pr))
```

GraphX scales to larger graphs



GraphX is roughly 2x slower than GraphLab

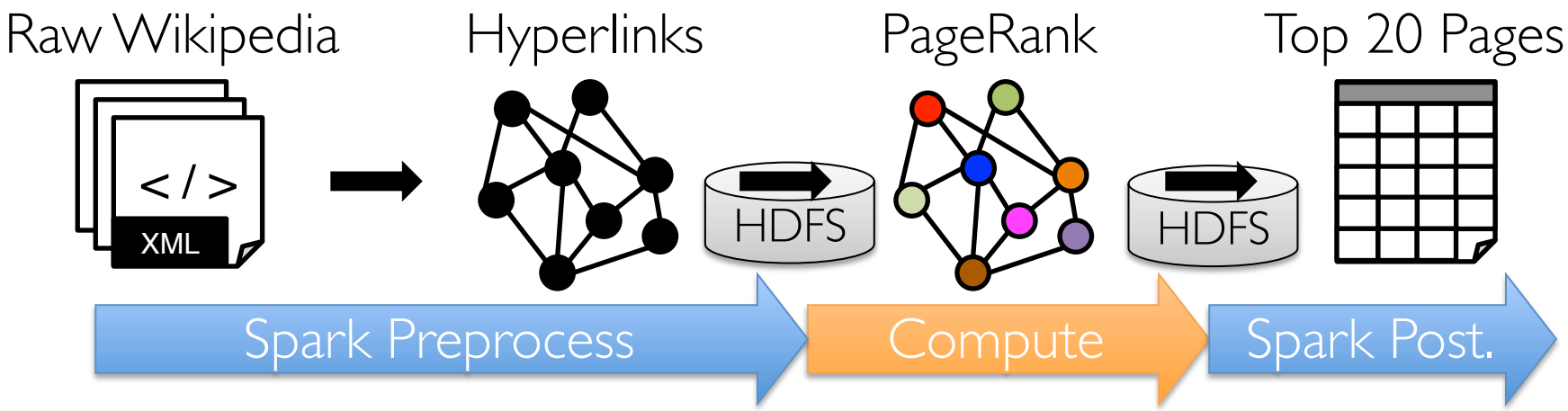
» Scala + Java overhead: Lambdas, GC time, …

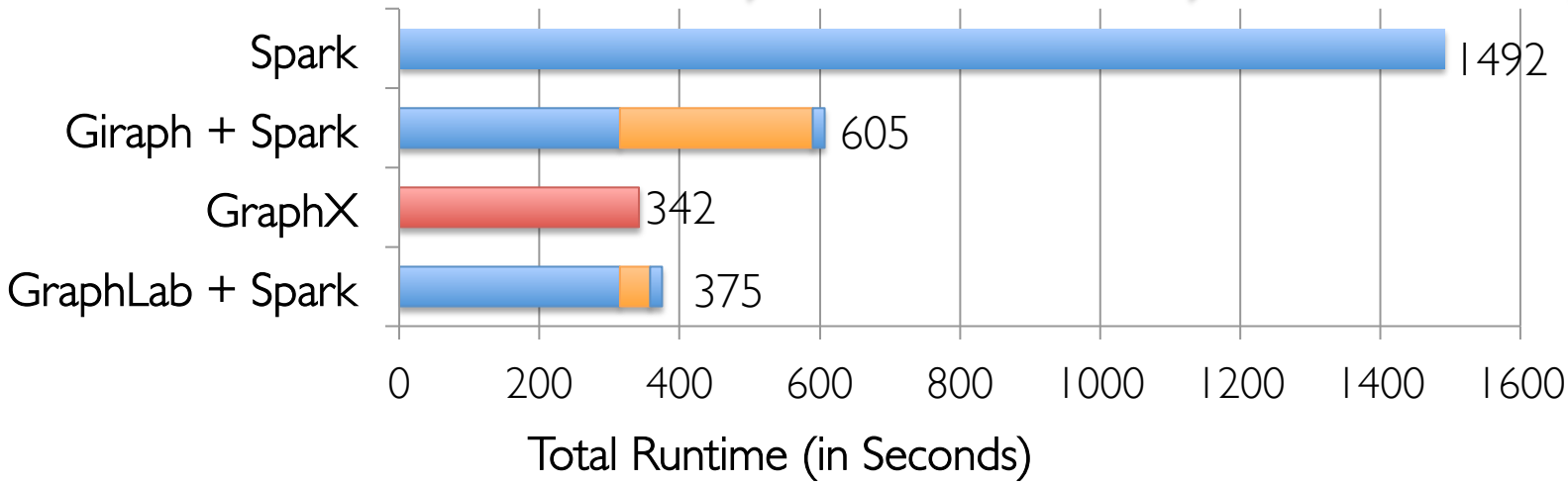
» No shared memory parallelism: 2x increase in comm.

PageRank is just one stage….

What about a pipeline?

A Small Pipeline in GraphX



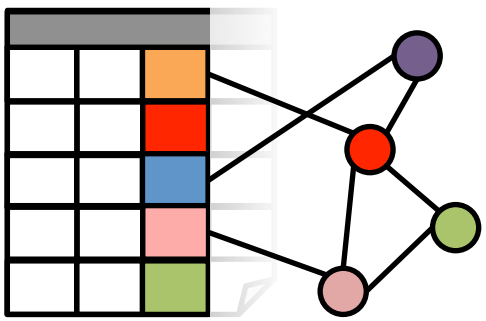


Timed end-to-end GraphX is *faster* than GraphLab

GraphX: Unified Graph Analytics

New API

Blurs the distinction between Tables and Graphs



New System

Combines Data-Parallel Graph-Parallel Systems

APACHE GIRAPH

Spark

GraphLab

Enabling users to easily and efficiently express the entire graph analytics pipeline

Current Limitations of GraphX

No support for asynchronous computation

- Favor determinism over speed

Not optimized for out-of-core processing

GraphLab Create (GraphLab + GraphX):

- Supports asynchrony and out-of-core processing
- Currently not distributed

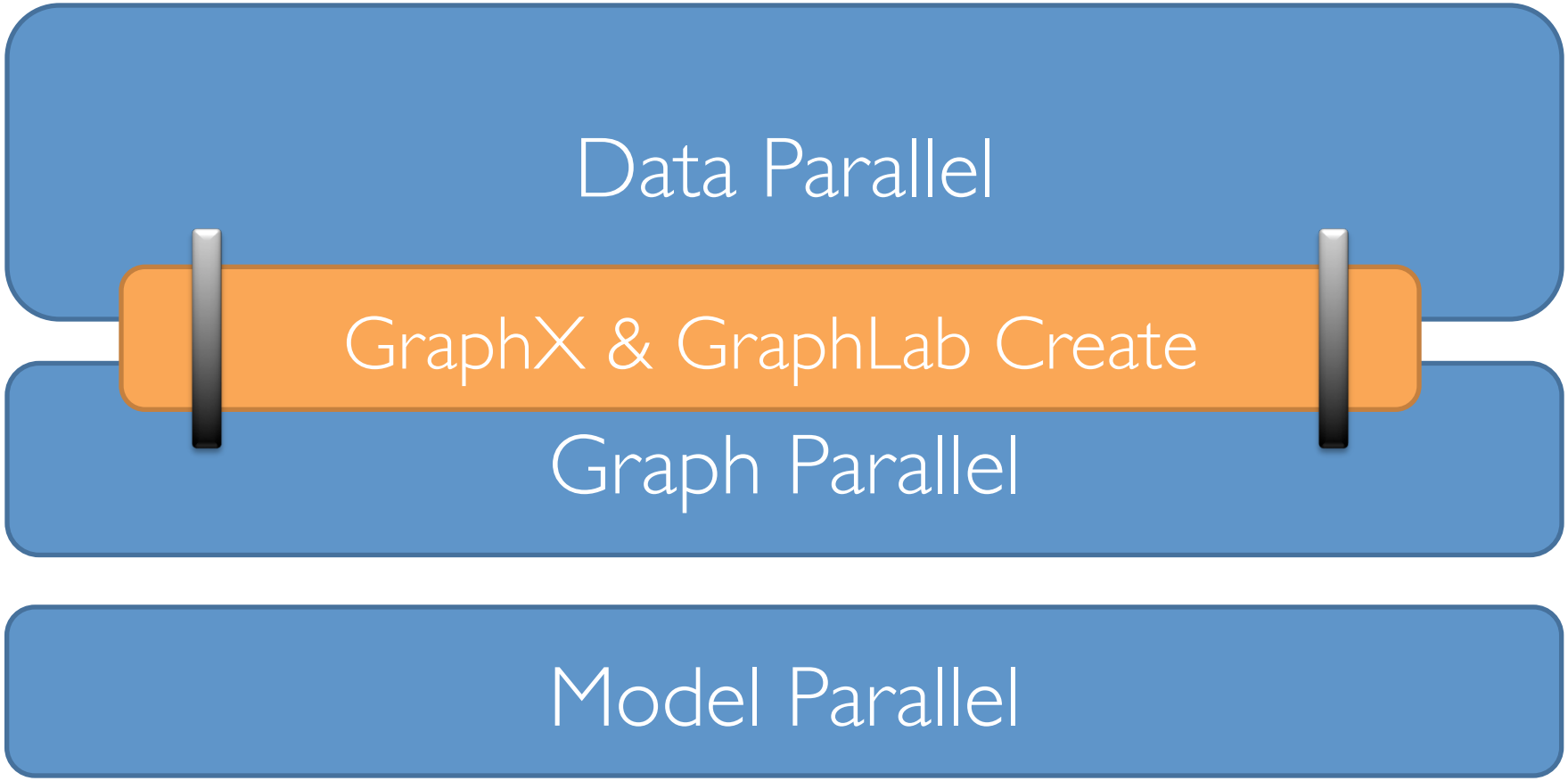
Outline of the Tutorial

Data Parallel

Model Parallel

Graph Parallel

Outline of the Tutorial



Future Directions

Themes in Learning Systems

Optimize for common patterns: aggregation, iteration, large-models, and graphs

- *Others?*

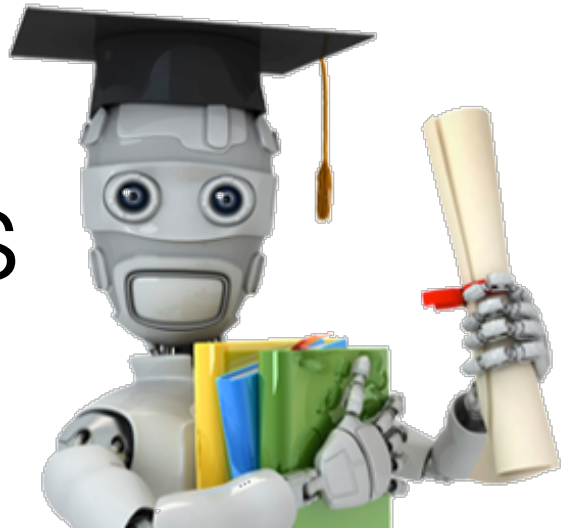
Leverage hardware trends: in-memory comp and elastic compute on commodity hardware

- RDMA, SSDs?

Tradeoff accuracy and runtime with sampling and asynchrony

Research Opportunities

Database Systems Research



Machine Learning Research

Concurrent State Mngmt.

Sampling and Approx. Query Processing

Query Optimization

Concurrency Control

Coordination Free (Parameter Server):

Provably fast and correct under key assumptions.

Optimistic Concurrency Control:

Provably correct and fast under key assumptions.

Database Systems
Improve Efficiency

X. Pan, J. Gonzalez, S. Jegelka, T. Broderick, M. Jordan. *Optimistic Concurrency Control for Distributed Unsupervised Learning*. NIPS'13

BlinkDB

Exploit sampling for fast, approximate answers with error bars:

```
SELECT avg(sessionTime)
FROM Table
WHERE city='San Francisco'
WITHIN 2 SECONDS
```

Queries with Time Bounds

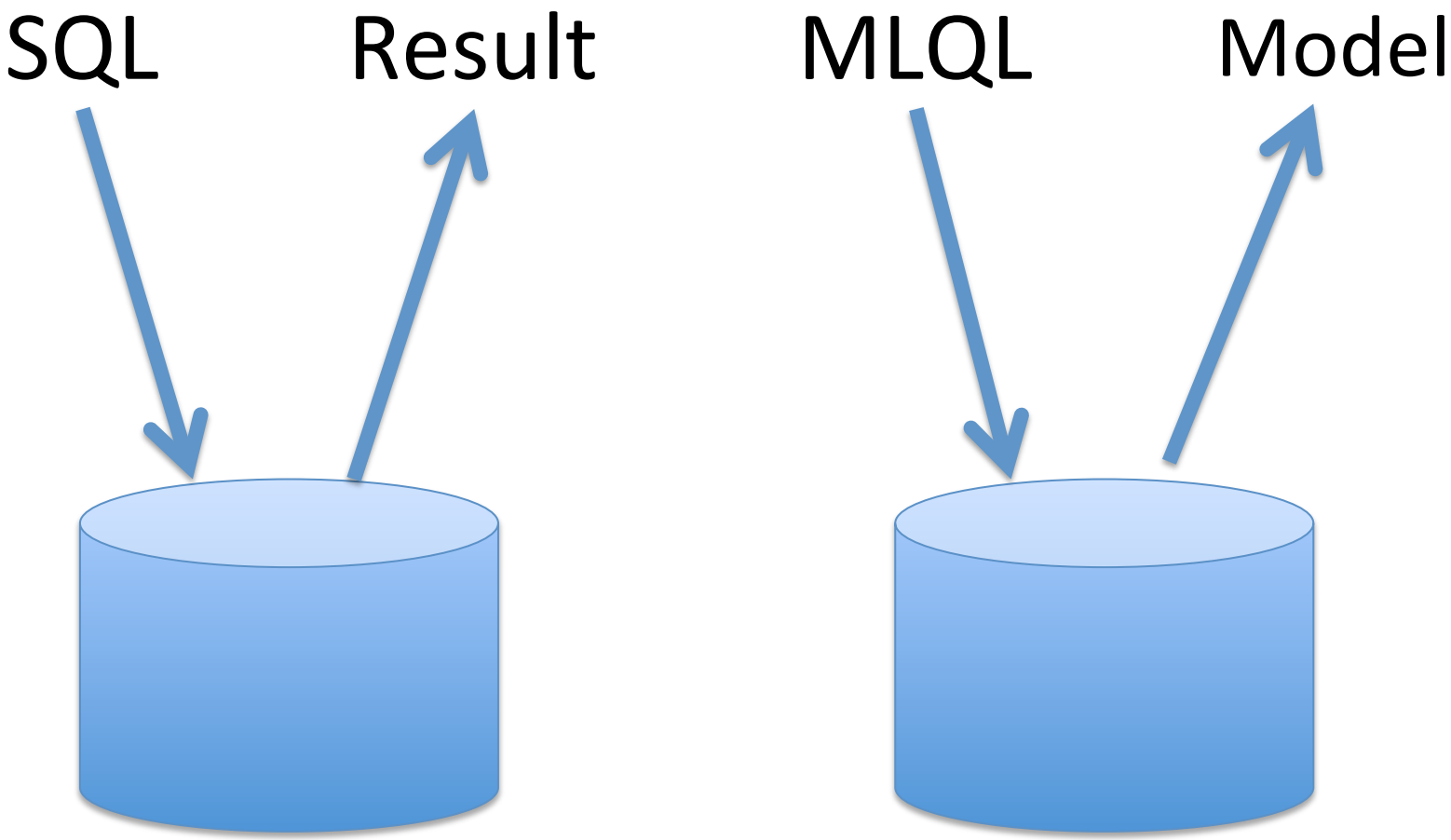
```
SELECT avg(sessionTime)
FROM Table
WHERE city='San Francisco'
ERROR 0.1 CONFIDENCE 95.0%
```

Queries with Error Bounds

Can we do the same for learning?

Agarwal et al., BlinkDB: Queries with Bounded Errors and Bounded Response Times on Very Large Data. ACM EuroSys 2013,

Insight: A Declarative Approach to ML



T. Kraska, A. Talwalkar, J. C. Duchi, R. Griffith, M. J. Franklin, and M. I. Jordan. *MLbase: A Distributed Machine-learning System.* CIDR'13


Systems Research
Research Opportunity
Machine Learning Research

Thank You

Questions?

Joseph E. Gonzalez
jegonzal@eecs.berkeley.edu

Slides (with animations) available at
http://eecs.berkeley.edu/~jegonzal


SUPPORT
VECTOR
MACHINES
REPEA
POWE
LAWS
END
DUALITY
GAP
Map Reduce
Map Reuse
Map Recycle
BAYSIANS
AGAINST
DISCRIMINATION
FREE
VARIABLES!
BAN
GENETIC
ALGORITHMS

How Systems Researchers Build Systems

Define the Problem

» Identify constraints and abstract the problem

Propose Solution: *Simple Idea*

» Don't try to solve everything

Implement the System

» Reuse existing systems wherever possible

Evaluation

» Support the design decisions

» What are the tradeoffs and limitations?