Learning from Big Data

TYSON CONDIE
UCLA

“Machine Learning is Programming by Example”

Used when:

- Programming is hard (e.g. topic detection, bioinformatics)
- Program changes all the time (recommender systems, antispam)

Machine Learning

Supervised
- Classification
- Regression
- Recommender

Unsupervised
- Clustering
- Dimensionality reduction
- Topic modeling

Machine Learning Workflow

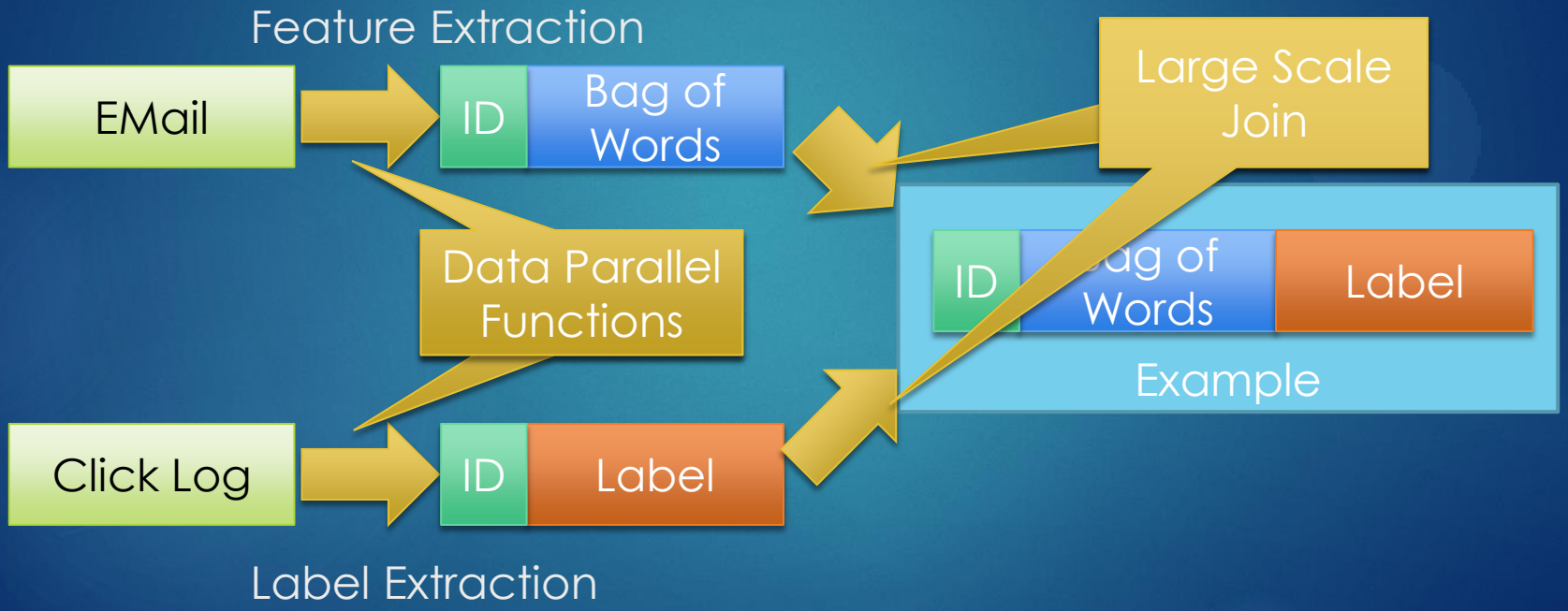
Step I: Example Formation

Feature and Label Extraction

Step II: Modeling

Step III: Evaluation (and eventually Deployment)

Example Formation at Scale



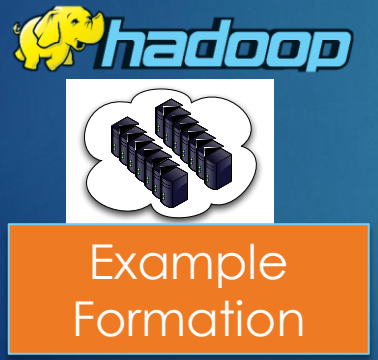
Machine Learning Workflow

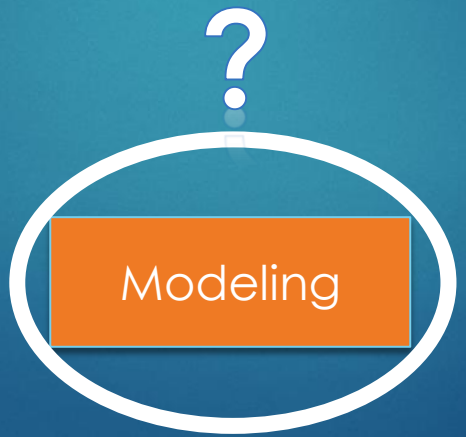
Step I: Example Formation

Feature and Label Extraction

Step II: Modeling

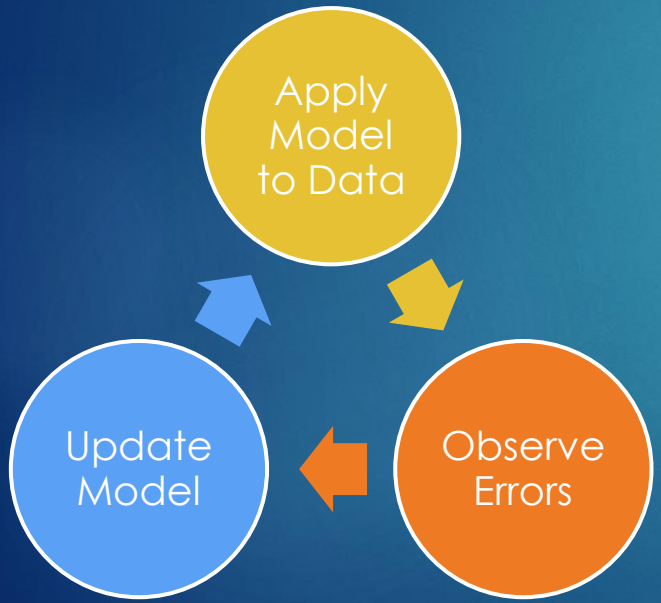
Step III: Evaluation





Modeling (30,000ft)

Learning is Iterative



Computational models

Statistical Query Model: Algorithm operates on global statistics of the dataset through aggregations.

Graphical Models: Algorithm operators on local (per-node) statistics of the dataset through message passing.

Many more: Custom solutions

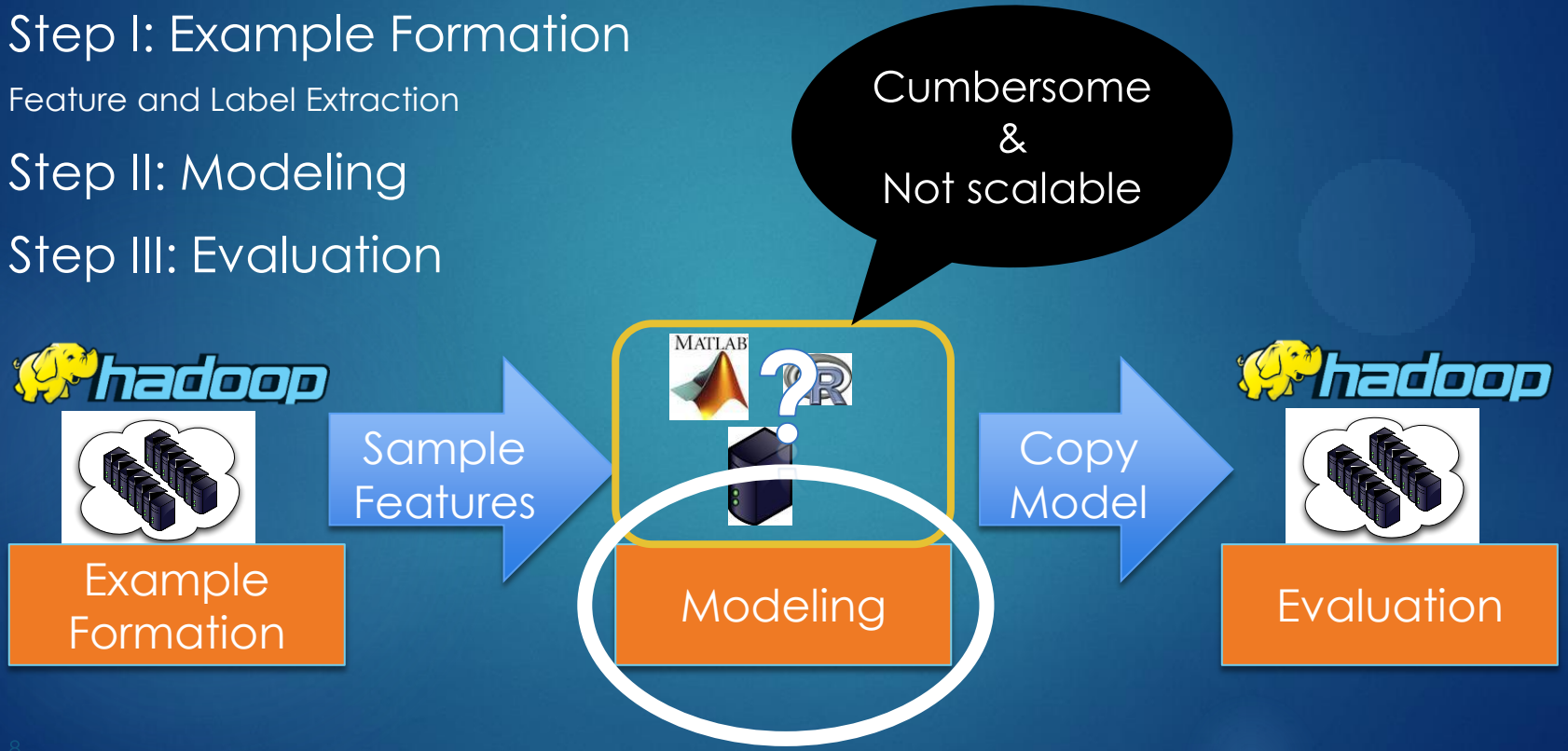
Machine Learning Workflow

Step I: Example Formation

Feature and Label Extraction

Step II: Modeling

Step III: Evaluation



Distributed Learning

Machine Learning in MapReduce?

+ MapReduce model fits statistical query model learning

- Hadoop MR does not support iterations (30x slowdown compared to others)
- Hadoop MR does not match other forms of algorithms

"Solution": Map only jobs

1. Allocate a set of map tasks
2. Instantiate learning algorithm
3. Execute iterative algorithm until convergence
4. Release mappers

➔ Hadoop Abuse

Hadoop Abusers 1: (All)Reduce and friends

Decision Trees on Hadoop

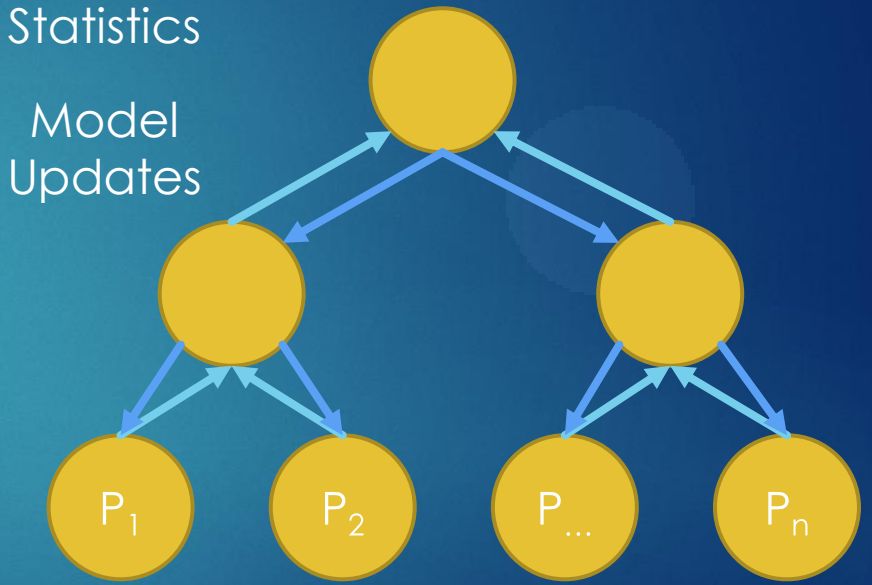
Jerry Ye et al.

- Runs OpenMPI on a map only job
- Uses HDFS for coordination/bootstrap

Vowpal Wabbit

John Langford et al.

- AllReduce on a map-only job
- Uses custom server for coordination/bootstrap
- Constructs a binary aggregation tree
- Optimizes node selection via redundant tasks



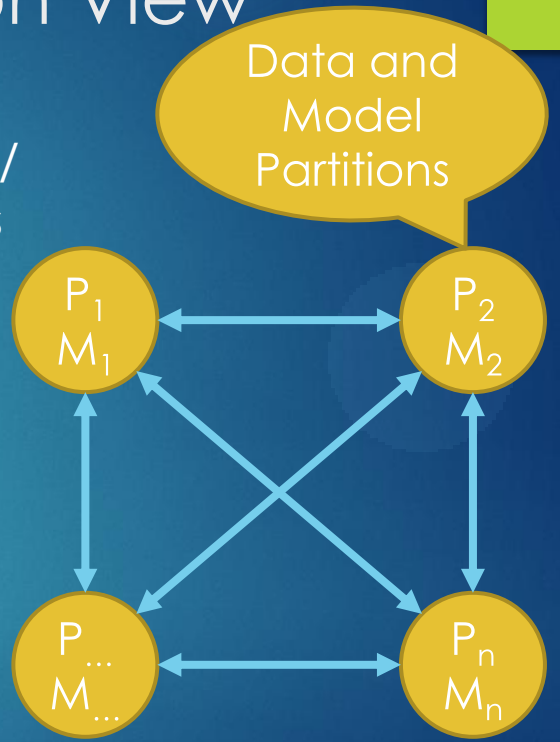
Hadoop Abusers 2: The Graph View

Apache Giraph
Avery Chen et al.

- Open source implementation of Pregel on a map-only job
- Uses Zookeeper for coordination/bootstrap
- Graph computation using "think like a vertex" UDFs
- Executes BSP message passing algorithm

Yahoo! LDA (YLDA)
Alex Smola and Shravan Narayanamurthy

- Instantiate Graphical Model on a map-only job
- Uses HDFS for coordination/bootstrap
- Coordinate global parameters via shared memory
  - Version 1: memcached. Version 2: ICE



Problems with this Approach

Problems for the Abusers

Fault Tolerance Mismatch

Resource Allocation Mismatch

Cumbersome integration with M/R

Every Abuser has to implement …

- Networking
- Cluster Membership
- Bulk data transfers
- …

Problems for the Cluster

Abusers Violate MapReduce assumptions

Network usage bursts in (All)Reduce

The Abusers are disrespectful of other users

Hoarding of resources

Graph Analytics Systems

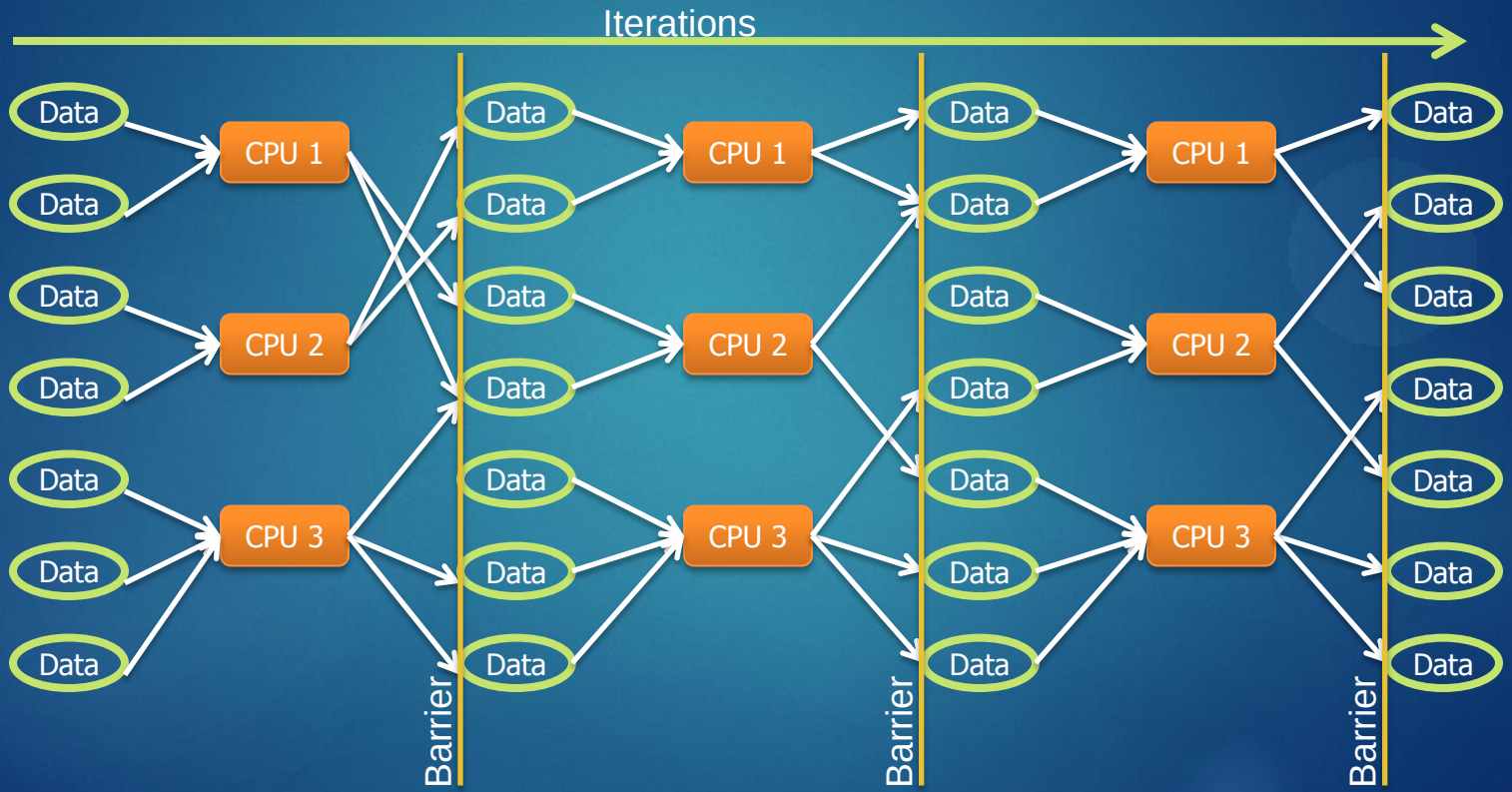
Processing Big Graphs

- Options
  - Create custom distributed infrastructure
  - Just use MapReduce
  - Use a graph library: BGL, LEDA, NetworkX
- Problems
  - Custom solutions do not generalize well
  - MapReduce is not the right programming model
    - And it is inefficient!
  - Graph libraries cannot handle problems at scale

Google Pregel

- Graph processing Framework
  - High scalability
  - Fault-tolerance
  - Graph-oriented programming model
- Inspired by Valiant's Bulk Synchronous Parallel (BSP) model
- The Pregel name honors Leonhard Euler
  - The Bridges of Königsberg, which inspired his famous theorem, spanned the Pregel river

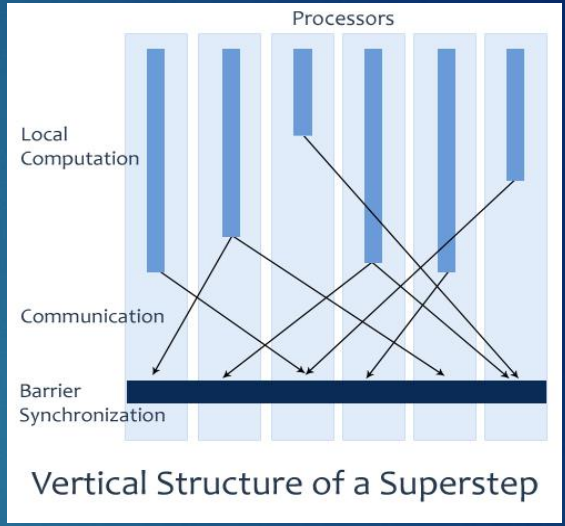
Bulk Synchronous Parallel Model



Pregel programming model

- Model of Computation
  - Input: a graph of vertices and edges
  - Each vertex holds a modifiable user defined value
  - Each edge is associated with a source vertex, value and a destination vertex

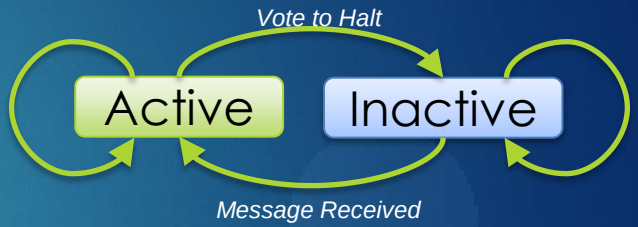
- Runtime executes a sequence of iterations called *Supersteps*
  - A user-defined function F is executed at each vertex V
  - F can read messages sent to V in superstep S – 1 and send message to other vertices, which will be received at superstep S + 1
  - F can modify the state of V and its outgoing edges
  - F can change the topology of the graph



Vertical Structure of a Superstep

Algorithm Termination

- Vote to halt protocol
  - In superstep 0, every vertex is active
  - All active vertices participate in the superstep
  - A vertex deactivates itself by "voting to halt"
  - A vertex is reactivated if it receives an external message



Vertex State Machine

- Program terminates when all vertices are simultaneously inactive and there are no messages in transit

The Pregel API in C++

- A Pregel program is written by subclassing the vertex class:

```
template <typename VertexValue,
typename EdgeValue,
typename MessageValue>

class Vertex {
public:
     virtual void Compute(MessageIterator* msgs) = 0;

     const string& vertex_id() const;
     int64 superstep() const;
     const VertexValue& GetValue();
     VertexValue* MutableValue();
     OutEdgeIterator GetOutEdgeIterator();

     void SendMessageTo(const string& dest_vertex,
     const MessageValue& message);

     void VoteToHalt();
};
```

To define the types for vertices, edges and messages

Override the compute function to define the computation at each superstep

To get the value of the current vertex

To modify the value of the vertex

To pass messages to other vertices

Pregel Code for PageRank


```
class PageRankVertex : public Vertex<double, void, double> {
Public:
virtual void Compute(MessageIterator* msgs) {
  if (superstep() >= 1) {
      double sum = 0;
      for (; !msgs->Done(); msgs->Next())
            sum += msgs->Value();
            *MutableValue() = 0.15 / NumVertices() + 0.85 * sum;
      }

      if (superstep() < 30) {
            const int64 n = GetOutEdgeIterator().size();
            SendMessageToAllNeighbors(GetValue() / n);
      } else {
            VoteToHalt();
      }
  } else {
      SendMessageToAllNeighbors(1f / NumVerticies());
  }
};
```

Vertex type with a value and message type of double, and no edge value

Compute my tentative PageRank

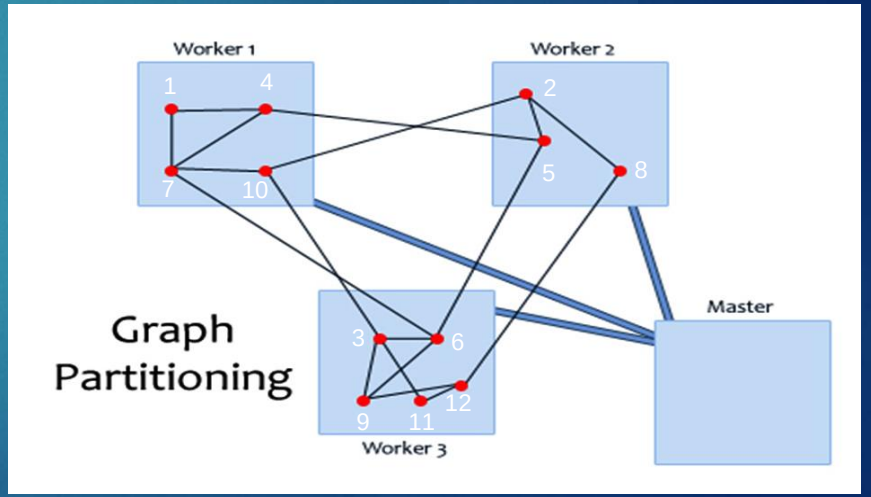
Send my tentative PageRank/n to my neighbors

Send my initial PageRank estimate

Pregel runtime

- Master/Worker Architecture
  - Similar to GFS and MapReduce
  - Master partitions the graph and schedules supersteps
  - Workers execute active vertices

- Graph partitioning
  - The input graph is divided into partitions
  - Default partition function: hash(VertexID) mod N
    - Where N is the # of partitions



Fault Tolerance in Pregel

- Fault tolerance is achieved through checkpointing
  - At the start of every superstep the master may instruct the workers to save the state of their partitions in a stable storage (e.g., GFS)
- Master uses ping messages to detect worker failures
- If a worker fails, the master reassigns vertices/input to another available worker and restarts the superstep
  - The new worker reloads the partition state from the most recent checkpoint

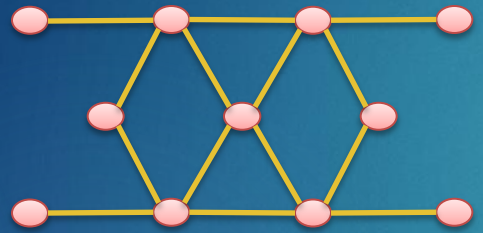
ASYNCHRONOUS GRAPH PROCESSING SYSTEM

Motivation for GraphLab

- Recap: Shortcomings of Google MapReduce
  - Programming model does not fit Graph computations
  - Overheads of running jobs iteratively --- disk access and startup costs
- Shortcomings of Google Pregel
  - BSP model requires synchronous computation
  - Slowest machine determines computation speed

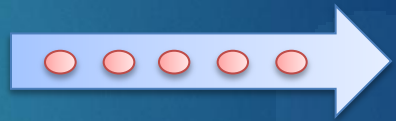
GraphLab

- A framework for parallel machine learning

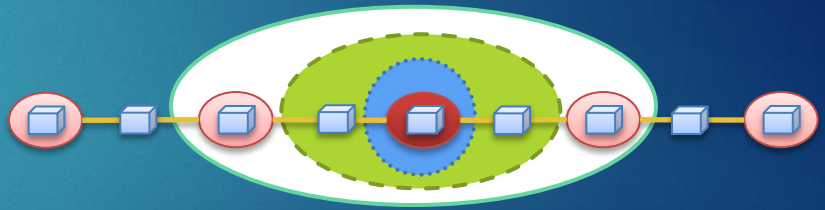


Data Graph

Shared Data Table



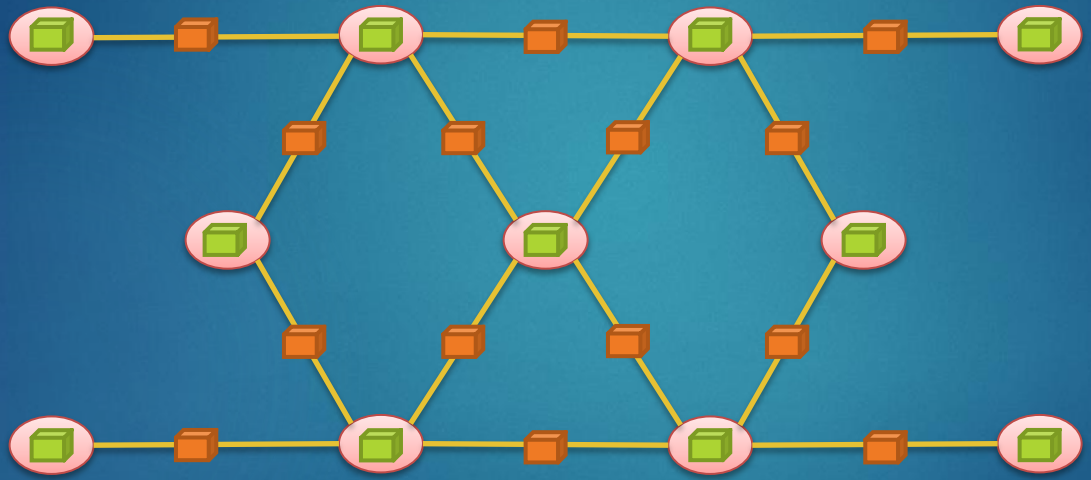
Scheduling



Update Functions and Scopes

Data Graph

- Graphs associate data at every vertex and edge

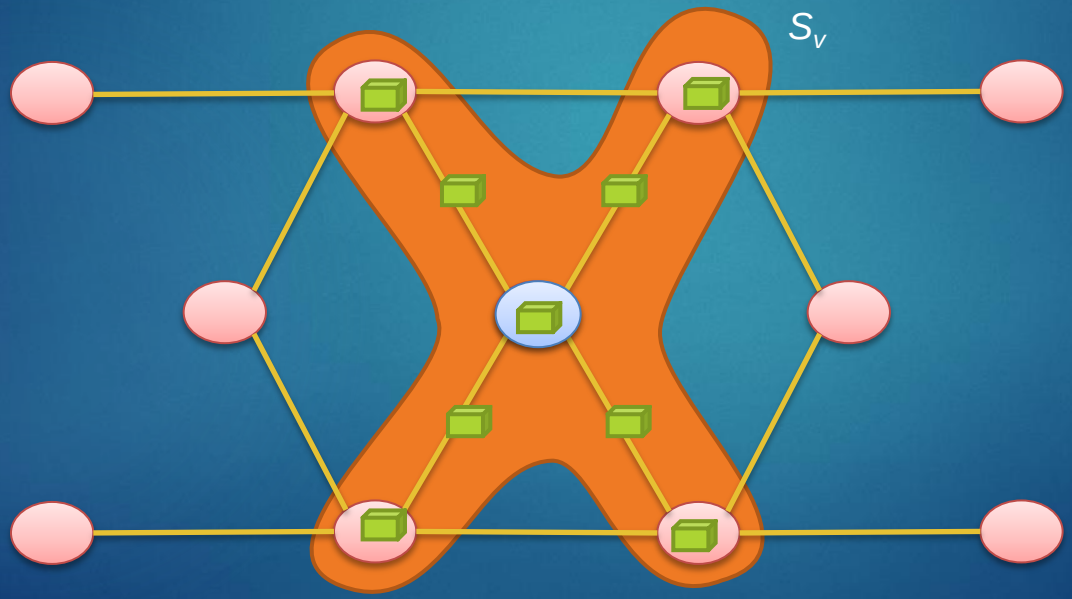


Data Graph

Arbitrary blocks of data can be assigned to vertices and edges

Update Functions

- Data graph is modified via update functions
  - The update function can modify a vertex v and its neighborhood (or scope S_v)



Shared Data Table

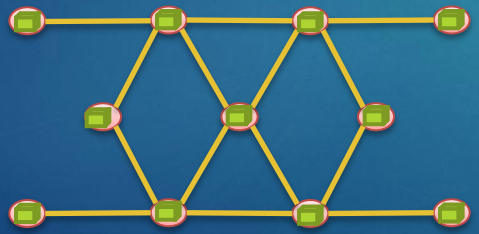
- Certain algorithms require global information shared among all vertices (Algorithm Parameters, Statistics, etc.)
  - GraphLab exposes a Shared Data Table (SDT)
- SDT is an associative map between keys and data blocks
  - T[Key] => Data Block

Shared Data Table

- The shared data table is updated using the ***sync mechanism***

Sync Mechanism

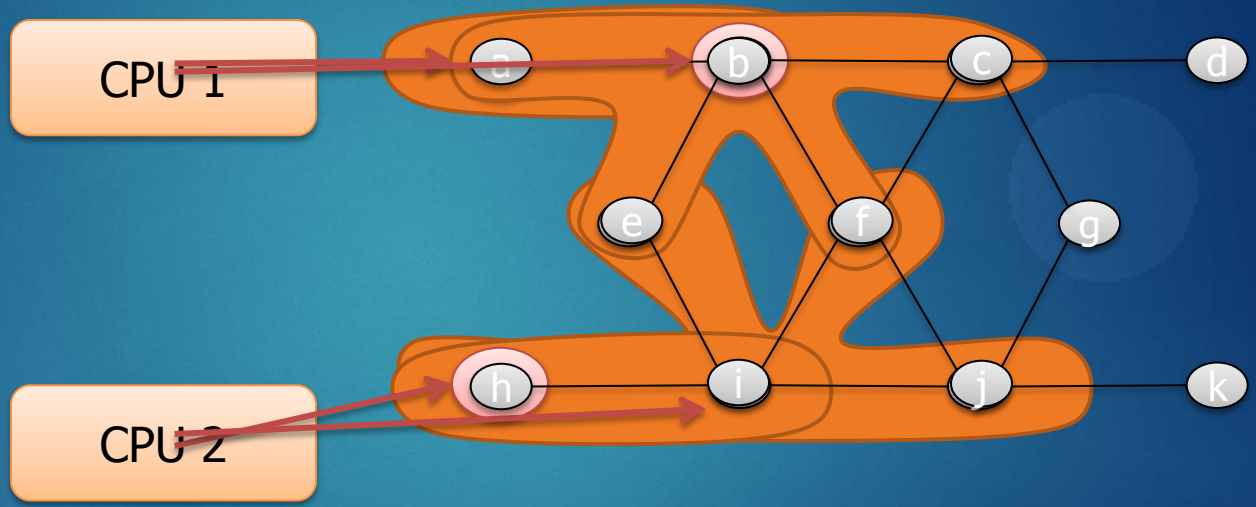
- Similar to "Reduce" in MapReduce
  - User defines fold, merge and apply functions that are triggered during a global sync mechanism
- **Fold** allows the user to aggregate information across all vertices
- **Merge** (optionally) allows a parallel tree reduction
  - Similar to "Combiners" in MapReduce
- **Apply** finalizes the result from fold/merge and commits to the SDT



Shared Data Table

Scheduling in GraphLab

Scheduler



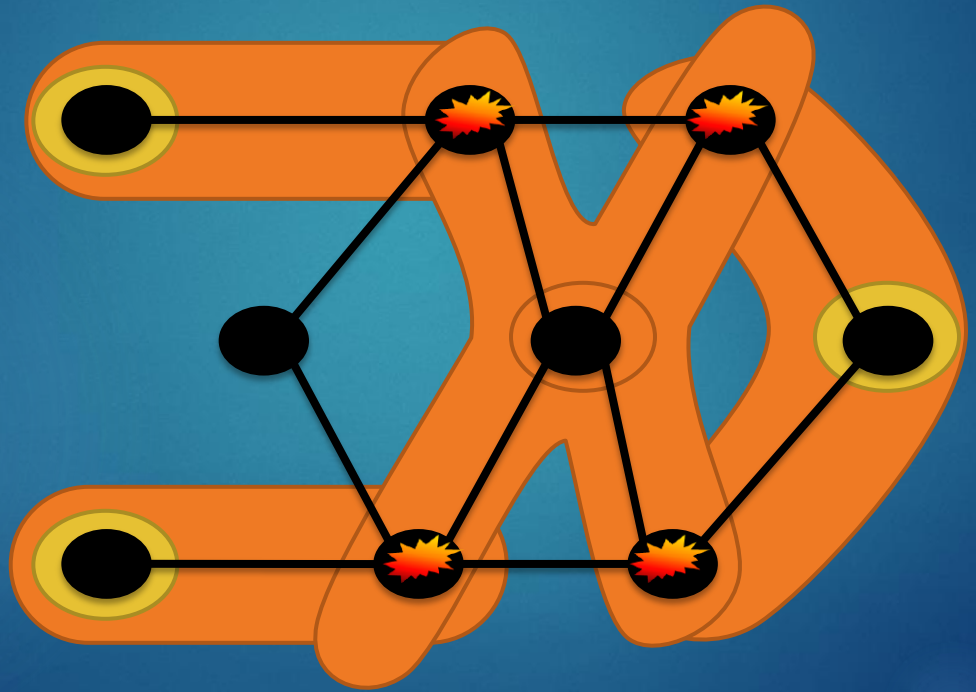
The process repeats until the scheduler is empty

Scheduling in GraphLab

- Base (Vertex) schedulers in GraphLab
  - Synchronous and Round-robin
  - Custom schedulers can also be defined
- Termination Assessment
  - If the scheduler has no remaining tasks
  - Or, a termination function can be defined to check for (algorithmic) convergence

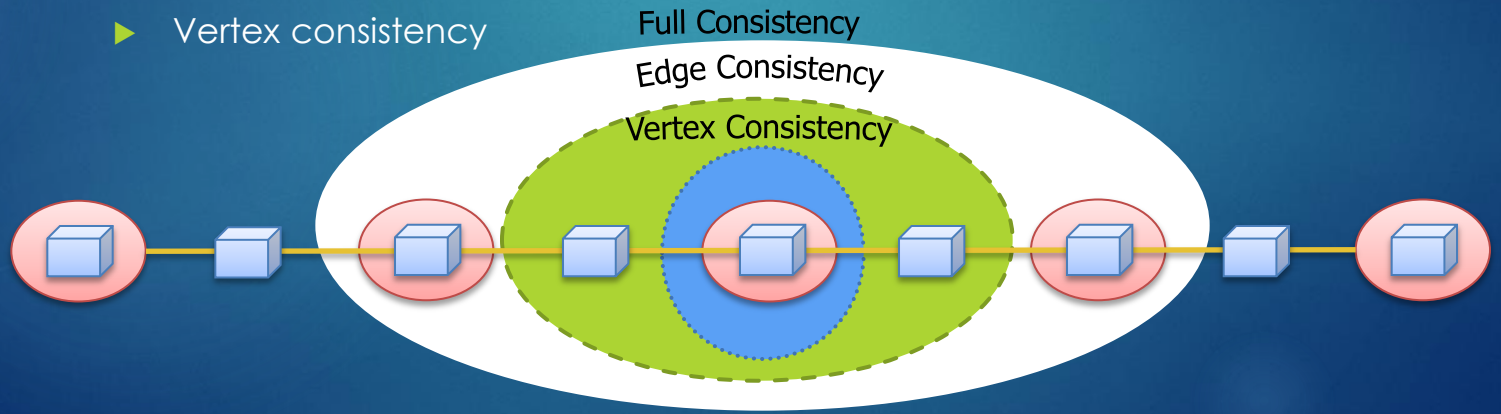
Need for Consistency Models

- How much can computation overlap?



Consistency Models in GraphLab

- GraphLab guarantees sequential consistency
  - Garanteed to give the same result in a sequential execution of the computational steps
- Consistency models
  - Full consistency
  - Edge consistency
  - Vertex consistency

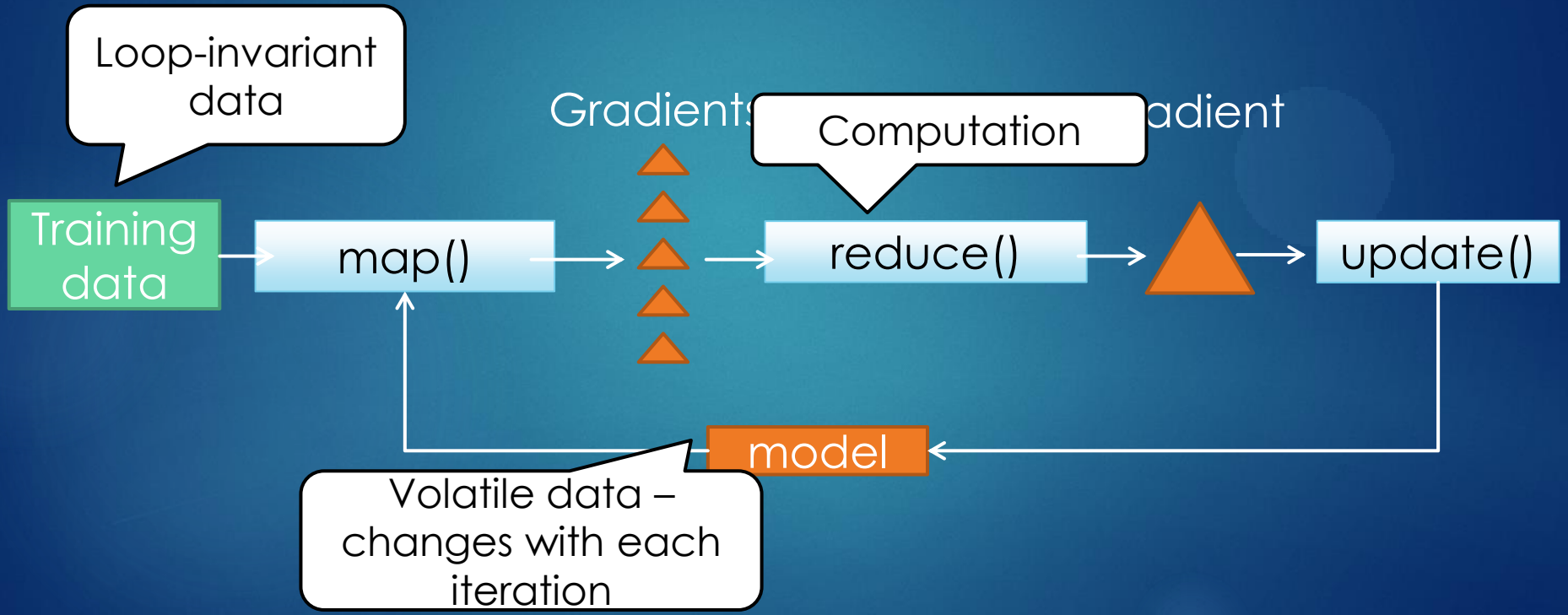


Dataflow Systems

Dataflow Systems

- Computation expressed as graph of operators
  - An operator transforms input to some output
  - Data flows along edges of the graph
- Iteration is captured as a cycle in the dataflow
  - Think of a RA query plan + UDFs + iteration

Example: BGD


Loop-invariant data
Training data
map()
Gradients
Computation
adient
reduce()
update()
model
Volatile data – changes with each iteration

Baseline: Hadoop

- One iteration is a DAG of MR jobs
- Iteration logic resides in the application

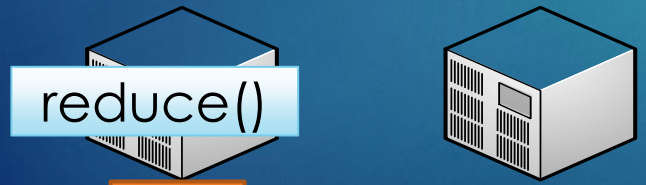
BGD in Hadoop

Training data

1 2

Model

a'



b Gradient

Iteration 1

Application

- Apply update()
- Continue?

Iteration 2

Shortcomings of Baseline

- Every iteration re-reads loop-invariant data
  - Batch Gradient Descent: Training data
  - PageRank: Links table
- Intrinsic overheads of Map/Reduce (cost to start-up tasks and controller)
  - Empirical evidence on startup cost of MapReduce job in Hadoop is in the order of a minute.

Haloop: Hadoop + Loops

Y. Bu et al., "HaLoop: Efficient Iterative Data Processing on Large Clusters", VLDB 2010

- Extended MapReduce API to express iteration:
  - Designate the loop body
  - Specify loop termination (max iterations or convergence)
  - Designate loop-invariant data
- Haloop optimizes access to loop-invariant data
  - Physical caching and indexing of data
  - Loop-aware job scheduling

Data Access in Haloop

Training Data

1 2

Model

a'

Iteration 1

Iteration 2

map()

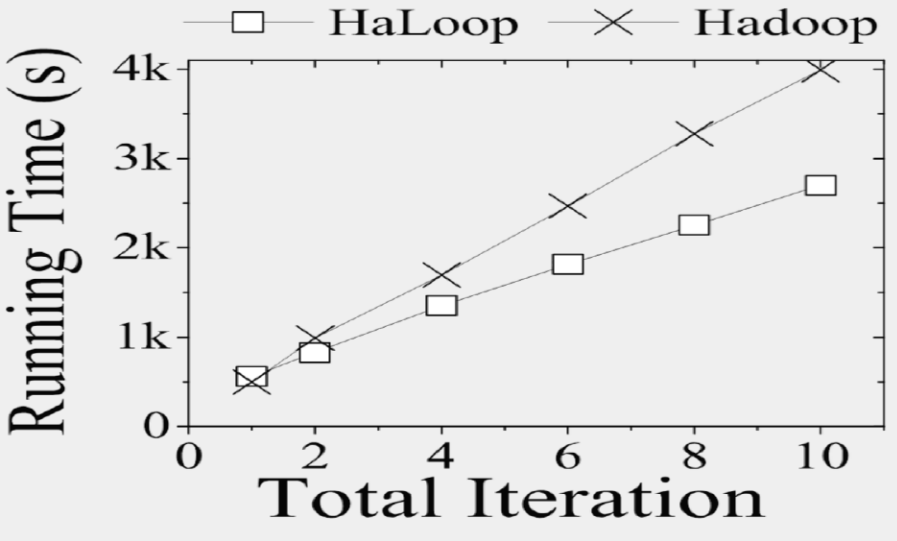
map()

No need to scan training data

reduce()

b

Gradient



Spark: In-memory dataflows

- Key notion: Resilient Distributed Dataset (RDD)
  - Resilient = recoverable if failures occur
  - Distributed = partitioned across nodes
  - Also, immutable
- RDDs can be created from:
  1. scanning data in stable storage, or
  2. running an operator on other RDDs
- A dataflow consumes RDDs and outputs RDDs

M. Zaharia, et al., "Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing", NSDI 2012

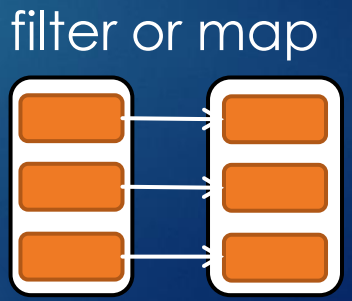


Illustration: BGD in Spark

Input Data

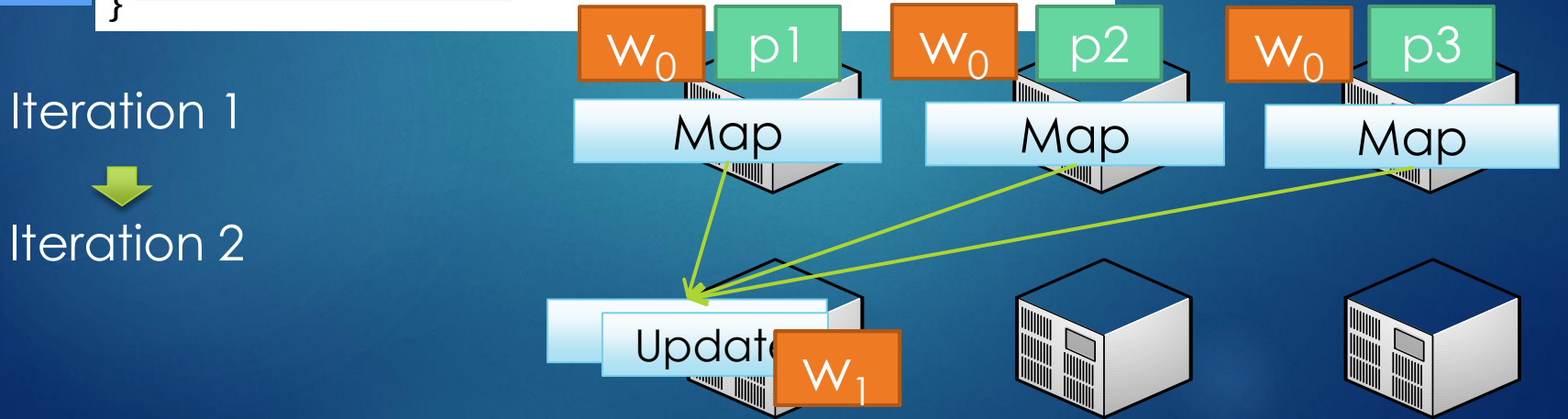
```
val points = spark.textFile(...)
                  .map(parsePoint).persist()
var w = // random initial vector
for (i <- 1 to ITERATIONS) {
  val gradient = points.map{ p =>
    p.x * (1/(1+exp(-p.y*(w dot p.x))) - 1) * p.y
  }.reduce((a, b) => a+b)
  w -= gradient
}
```

Model

Map

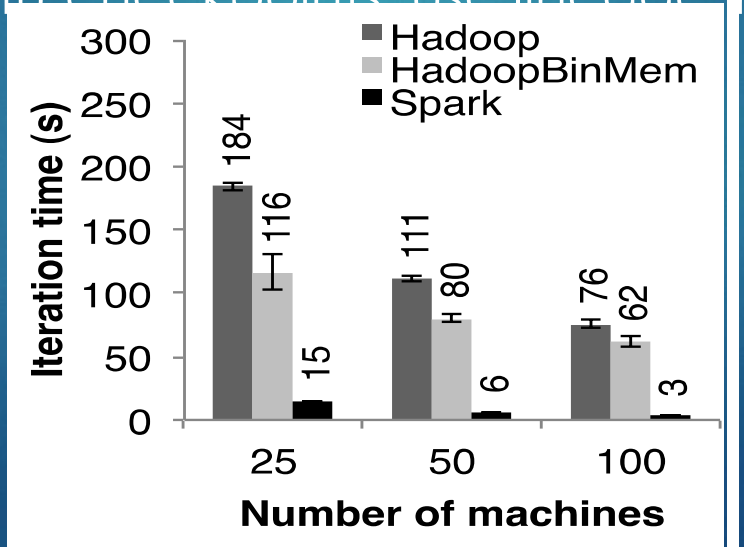
Reduce

Update



Spark's Execution Engine

- Loop-invariant data can be pinned in memory
- Computation gets scheduled near data
- No (implicit) checkpoints, use lineage instead

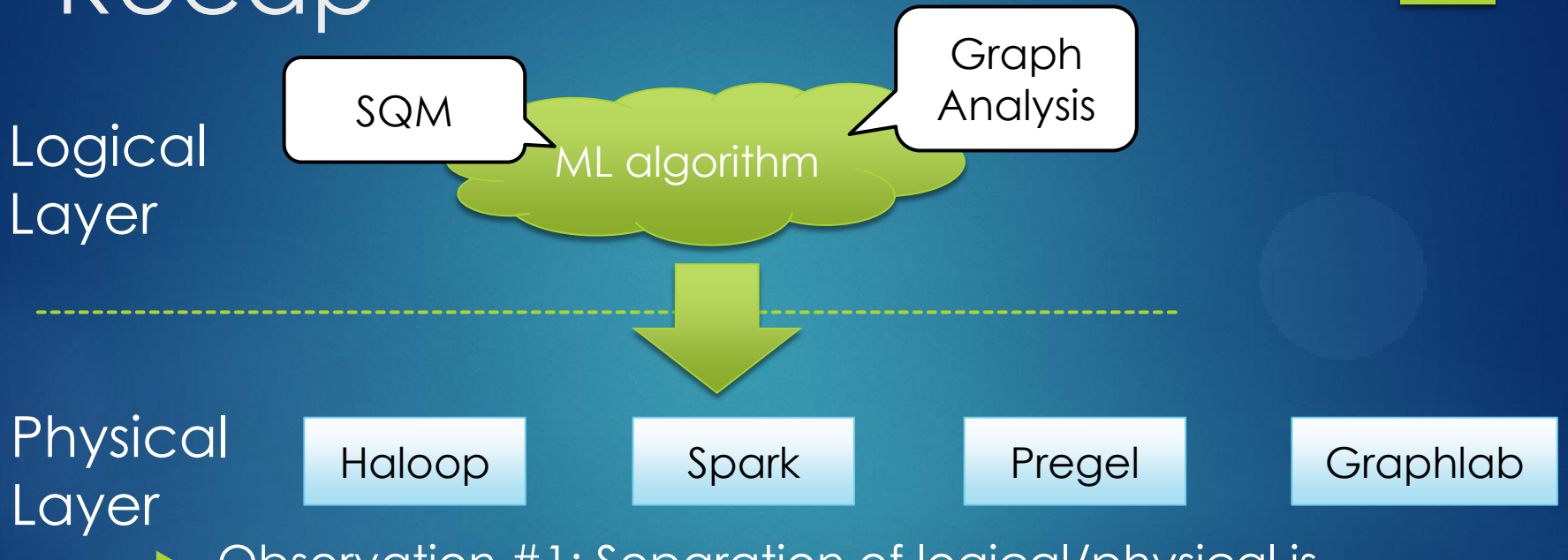


Other Efforts

- REX: Similar to Stratosphere
- Madlib: SQL library for matrix multiplication
- VW: Specialized tool for linear models
- DistBelief: Specialized system for NN learning
- Mahout: A library of ML algorithms over Hadoop
- Hyracks*

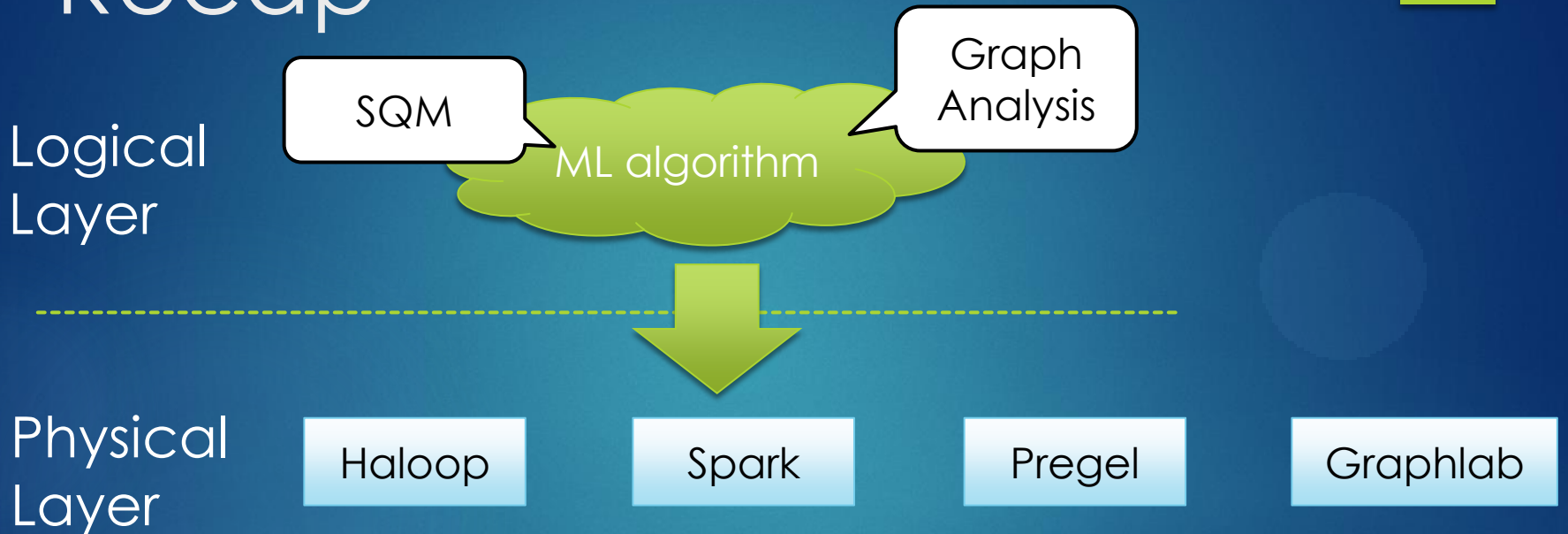
Conclusions and Outlook

Recap



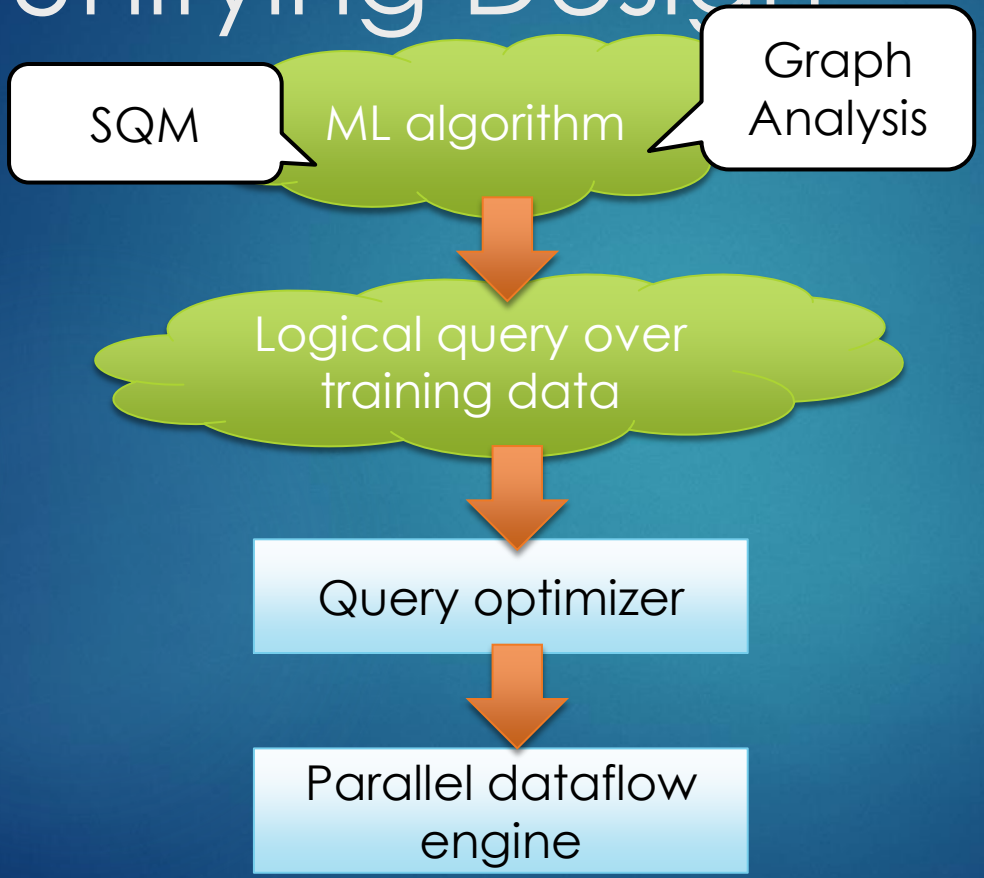
- Observation #1: Separation of logical/physical is reminiscent of query optimization
- Can we automate this translation?

Recap

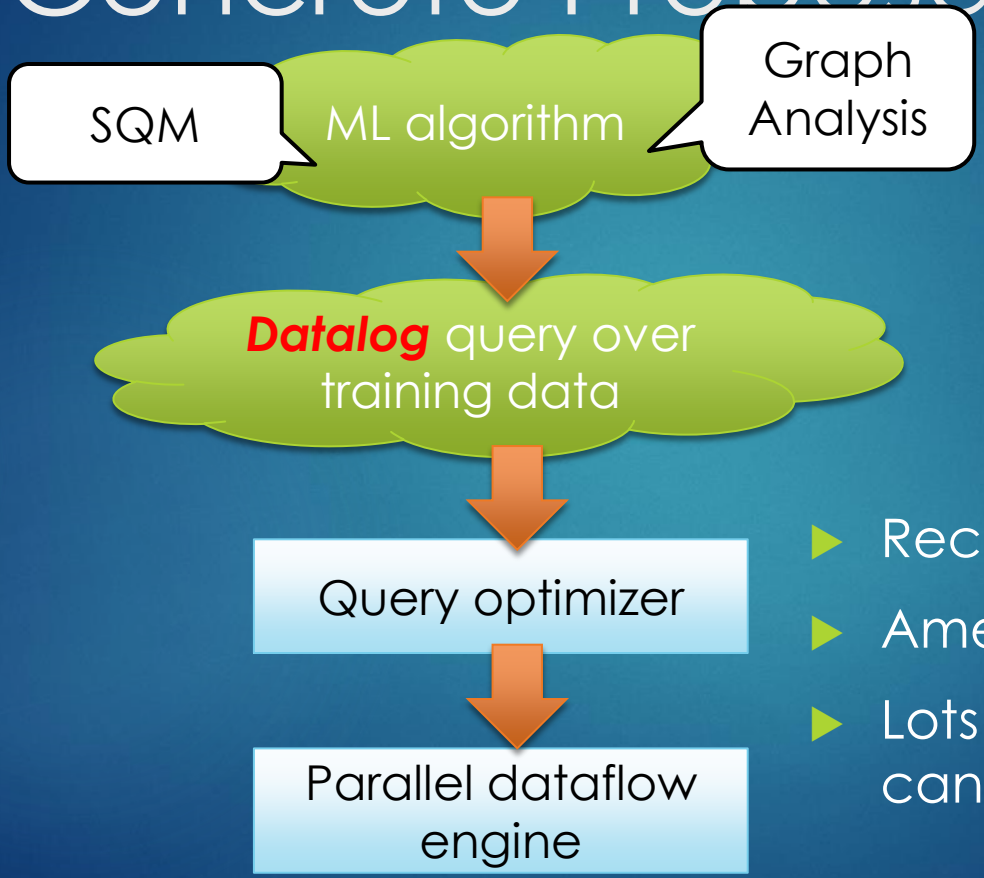


- Observation #2: Systems have to solve the same problems and adopt similar solutions
- Can we isolate these solutions in reusable modules?

A Unifying Design



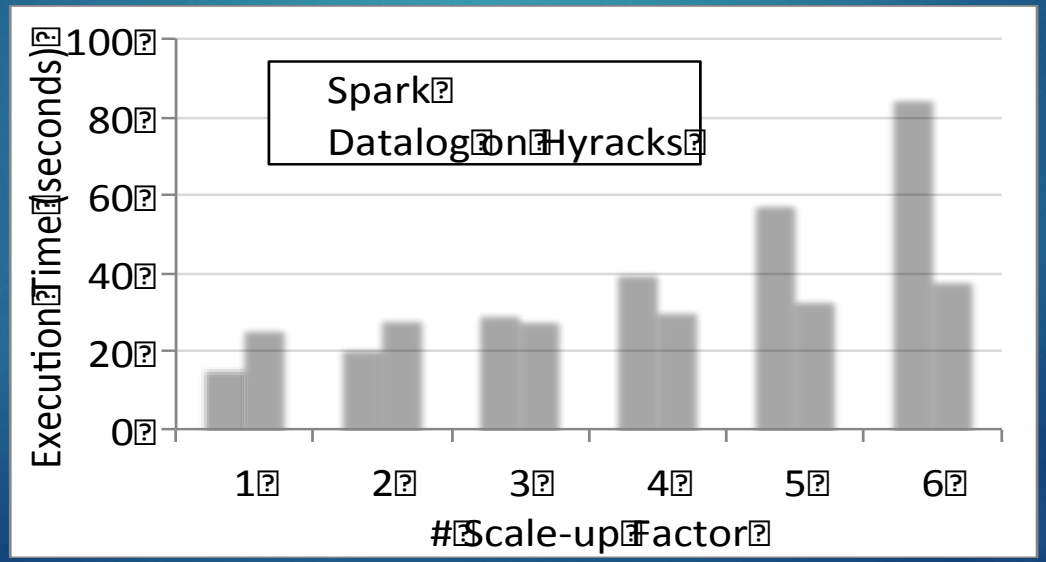
A Concrete Proposal



- Recursion is built-in
- Amenable to optimizations
- Lots of existing work that we can leverage

Version 0.1

- Implementation over Hyracks
- Supports both Iterative-MRU and Pregel
- Standard optimizations + some new tricks



Open Research Questions

- Iteration-aware query processing
  - Cost estimation for recursive computation
  - Cost models (time vs money)
  - Late- vs. early-stage processing
  - Effective handling of UDFs
- Computational models for graph analysis
- Provenance for triage
  - "My model misbehaves – why?"
- Tuning -- who is the "DBA"?
- Fault-awareness as a logical concept
  - Not all faults are catastrophic for ML
  - Algorithm-specific fault-tolerance
- Incremental learning

Thank you!

QUESTIONS?