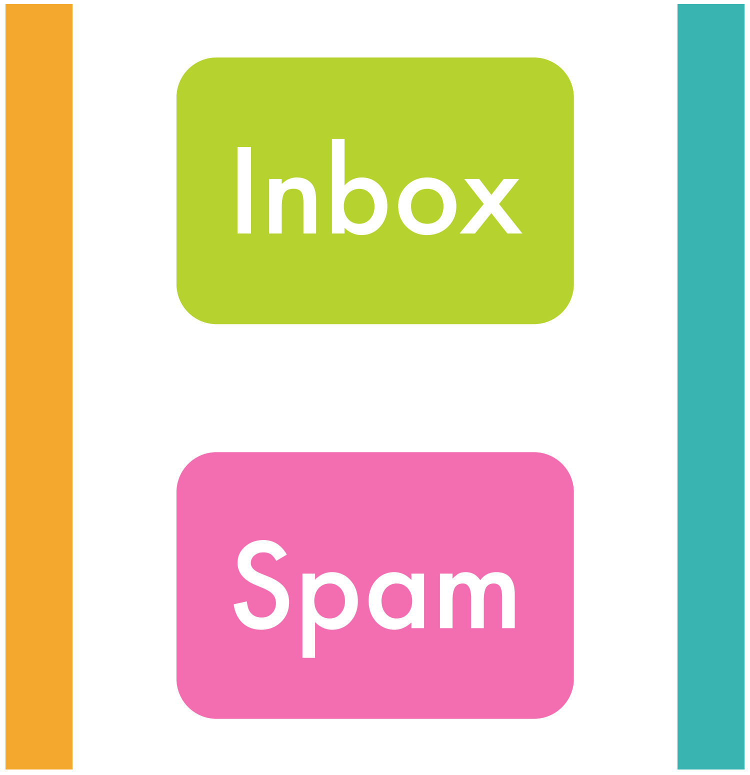
Templates

for scalable data analysis

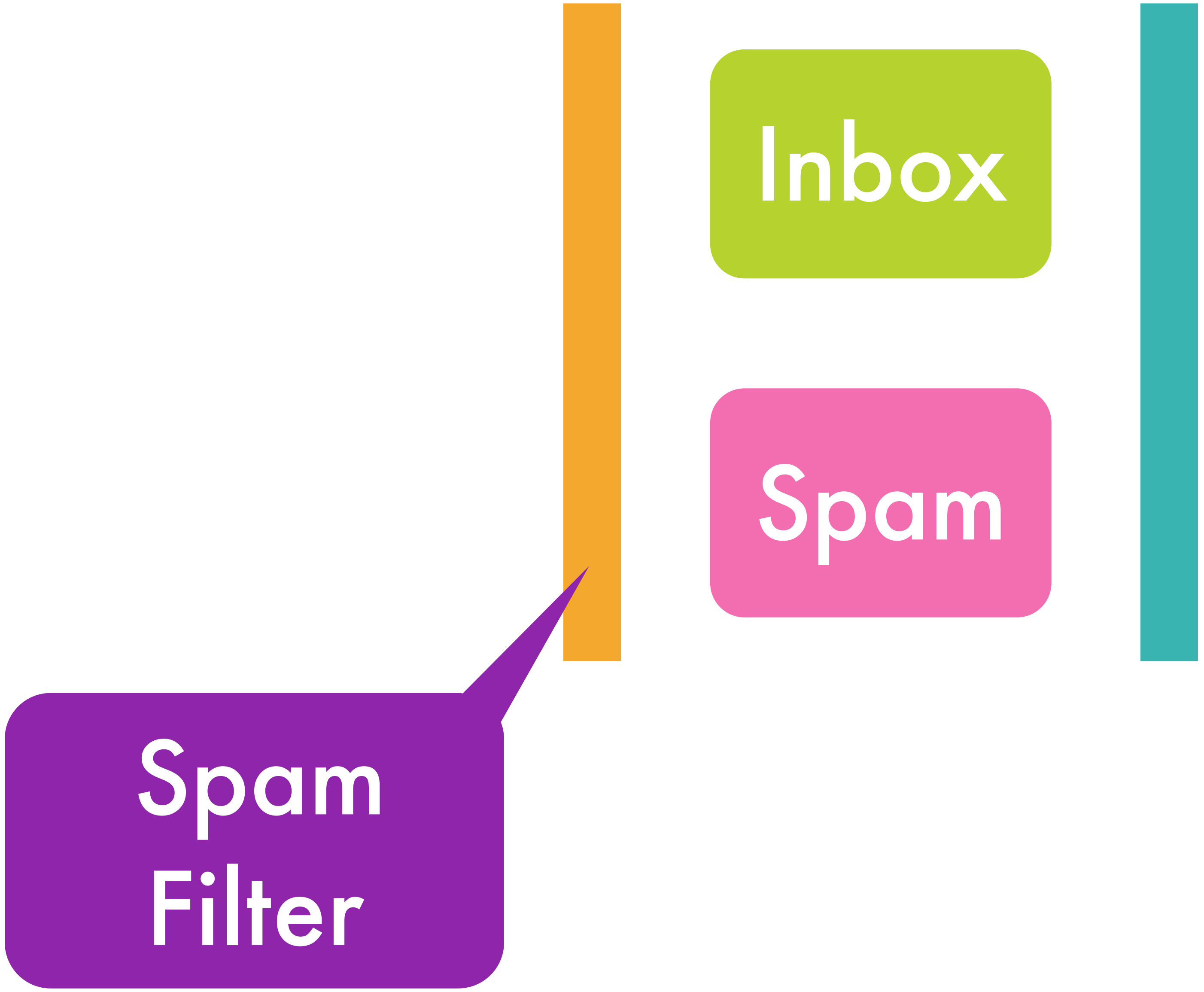
2 Synchronous Templates

Amr Ahmed, Alexander J Smola, **Markus Weimer**
Yahoo! Research & UC Berkeley & ANU

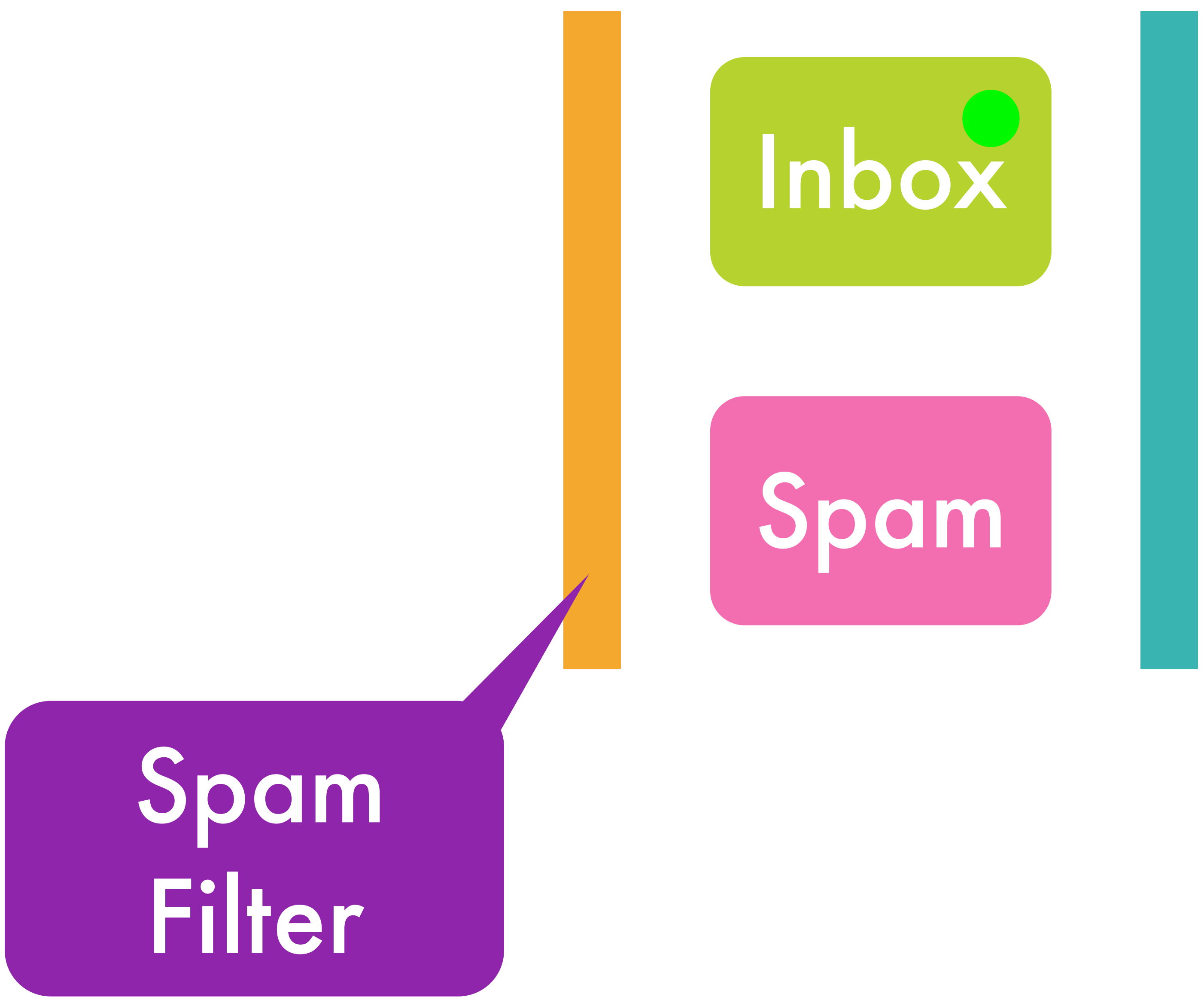
Running Example



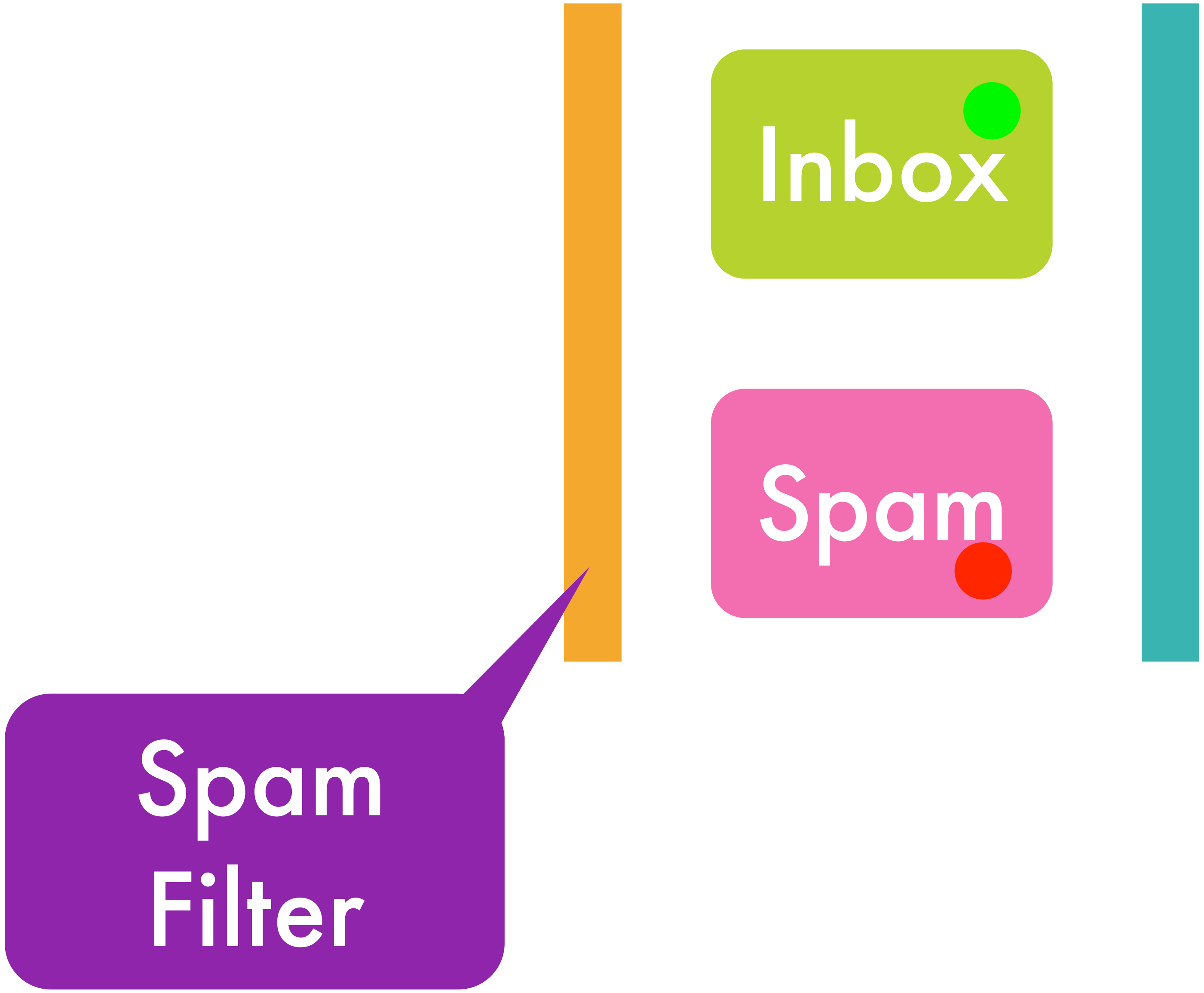
Running Example



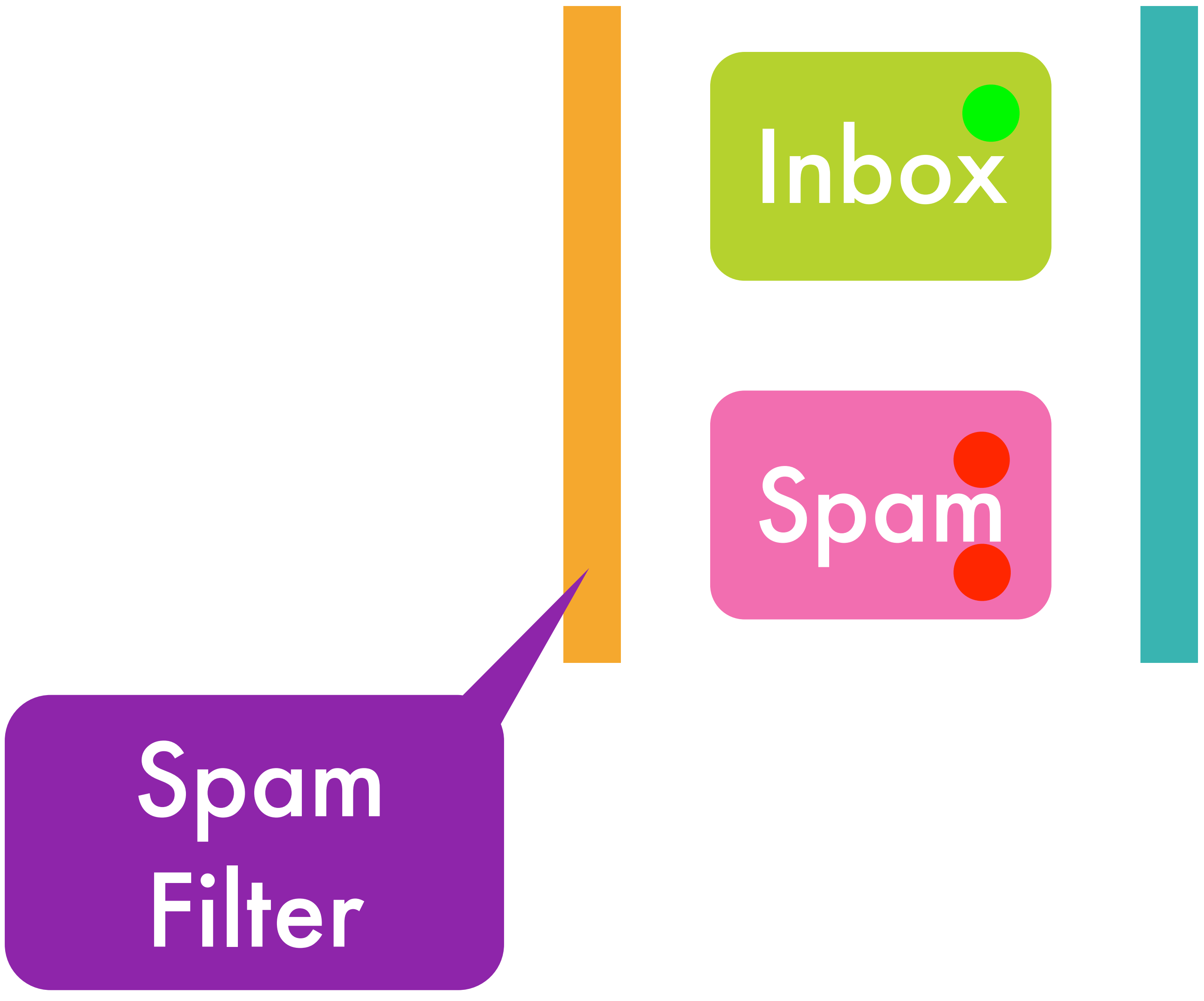
Running Example



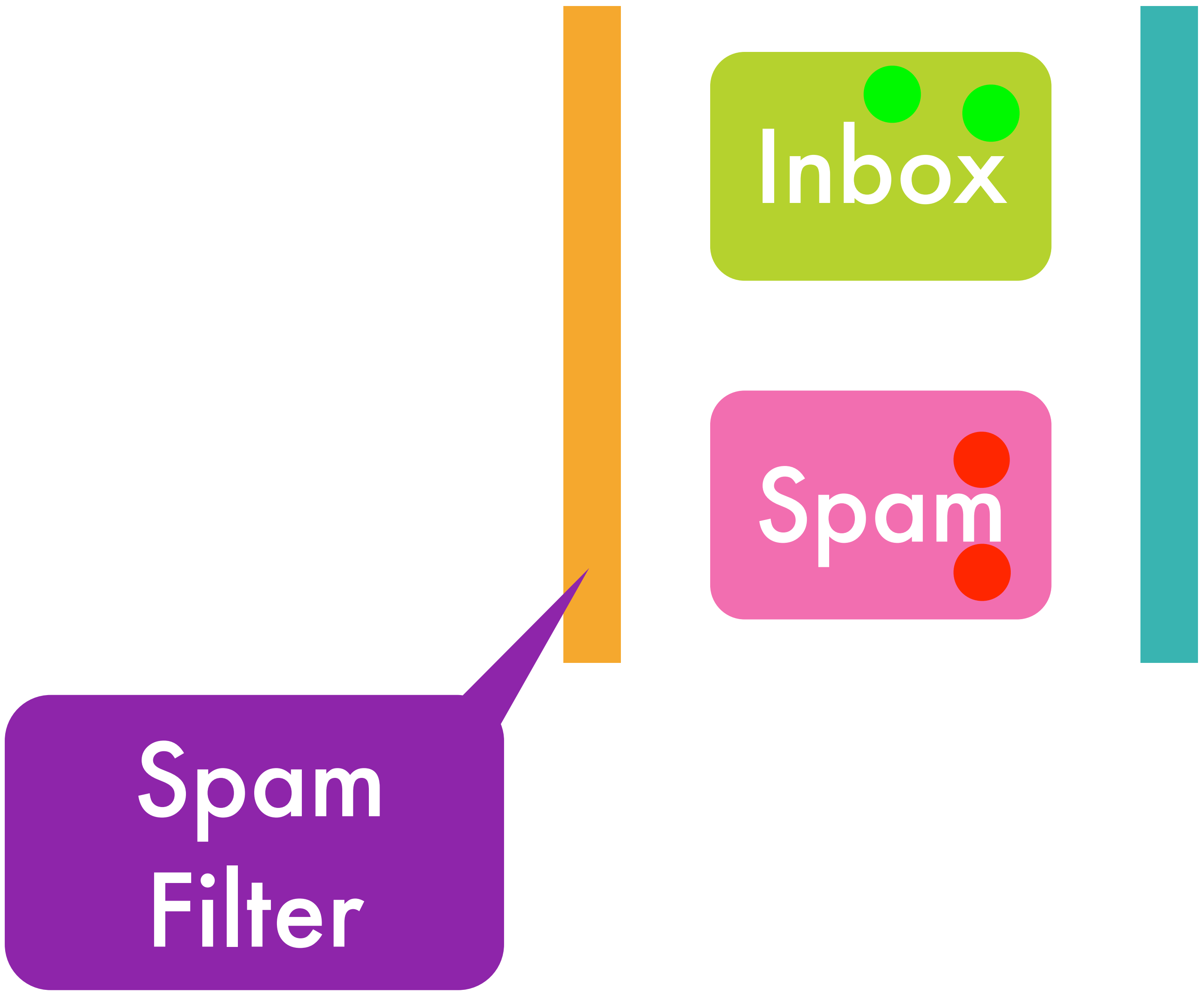
Running Example



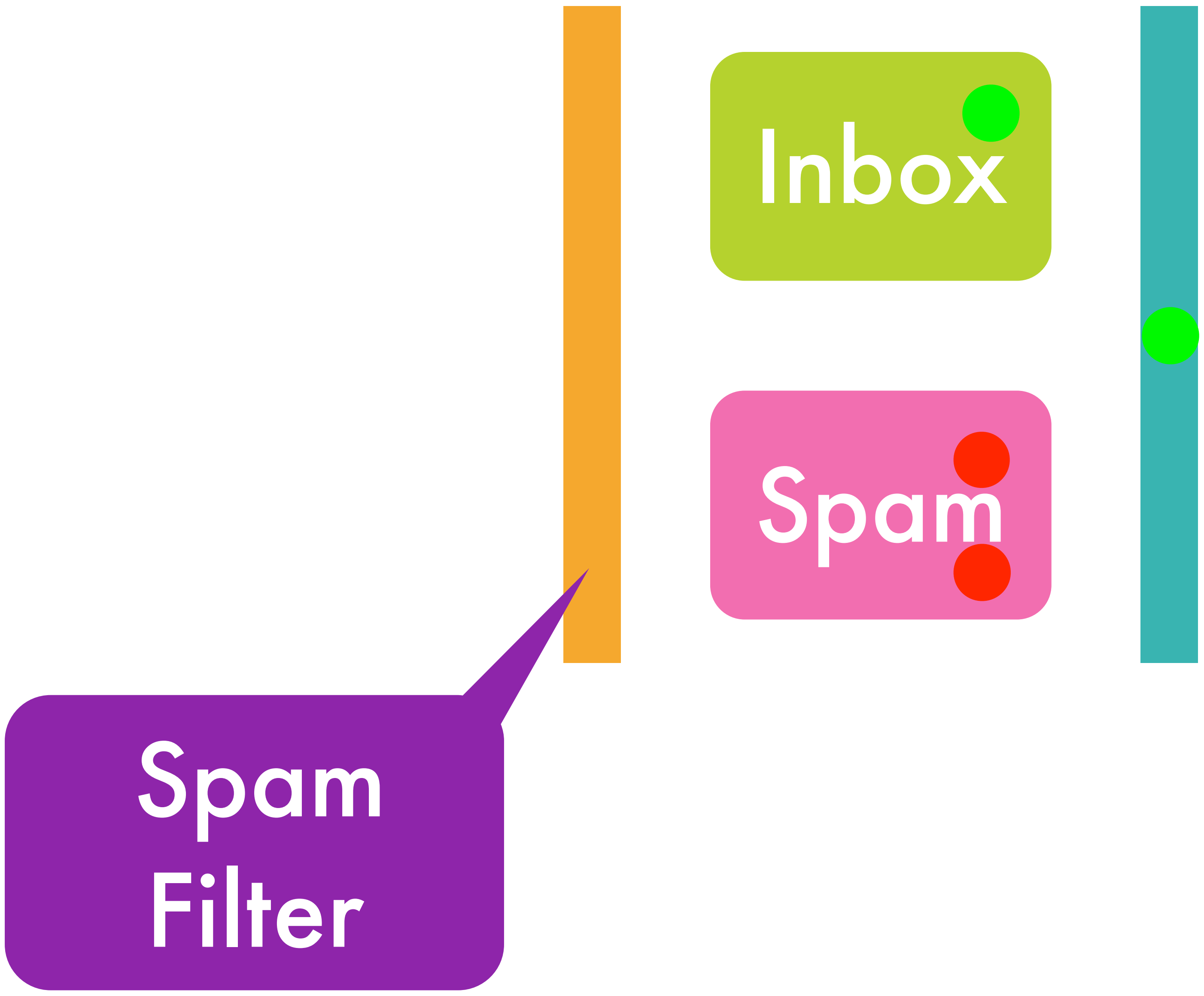
Running Example



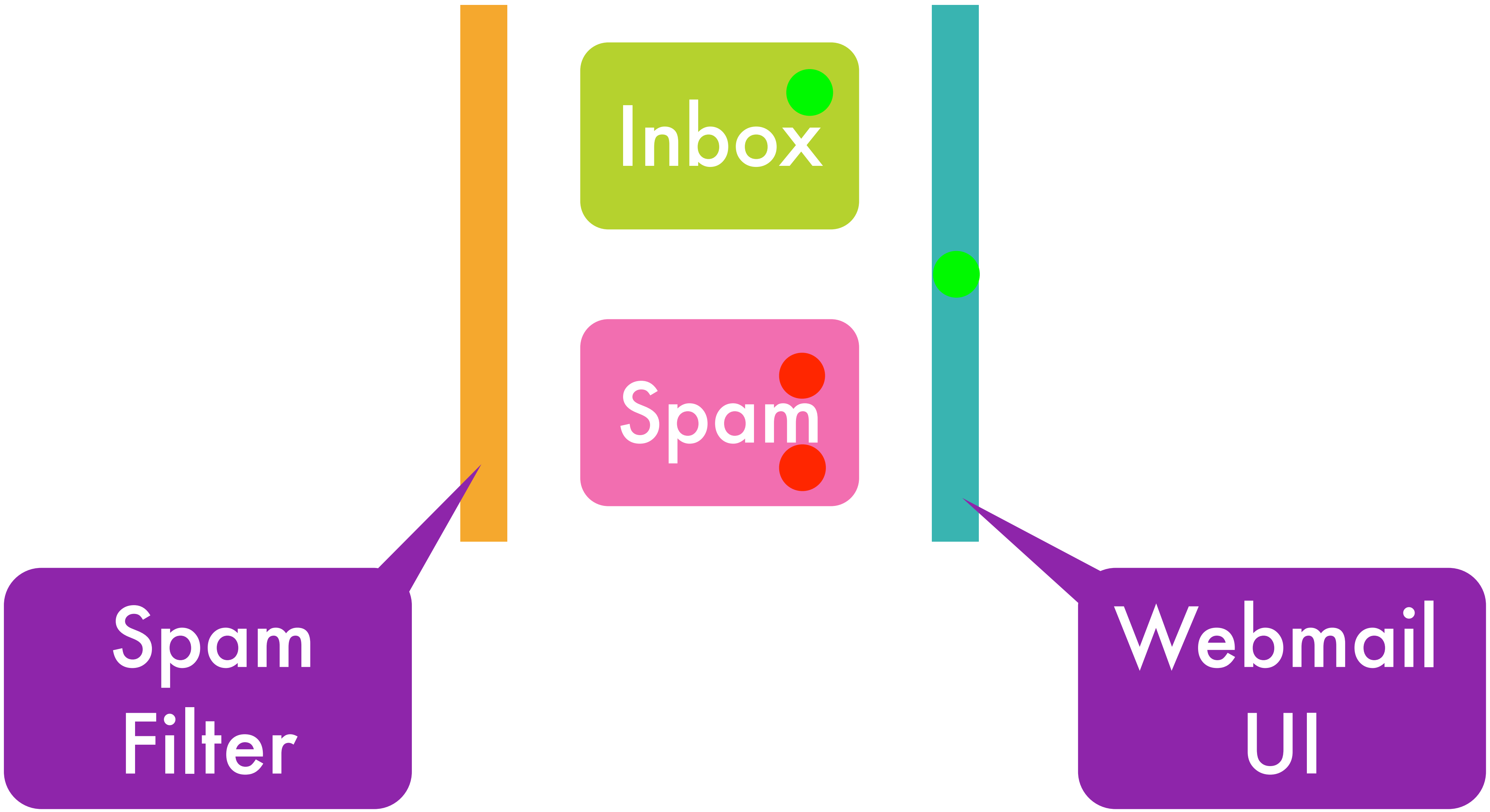
Running Example



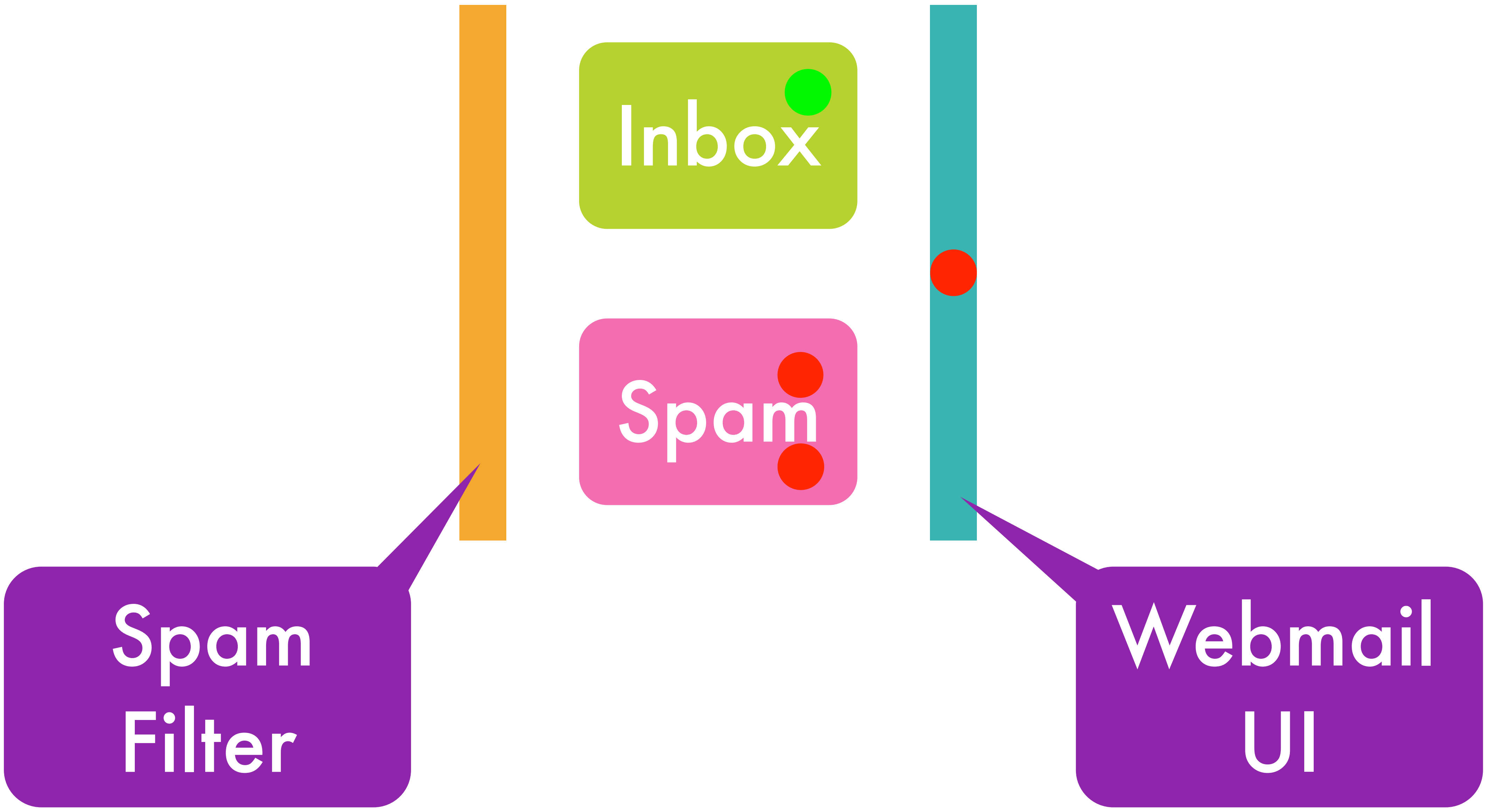
Running Example



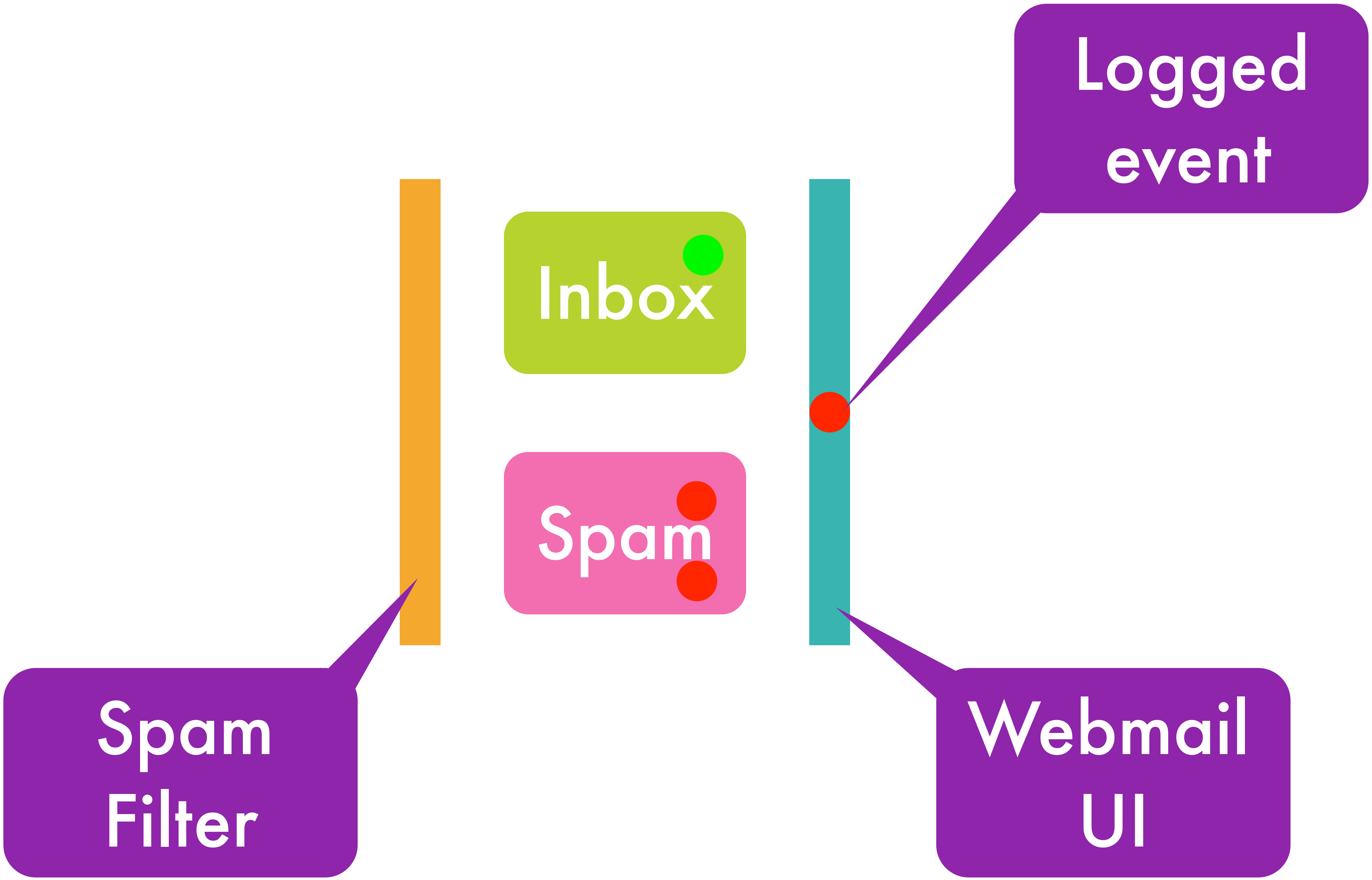
Running Example



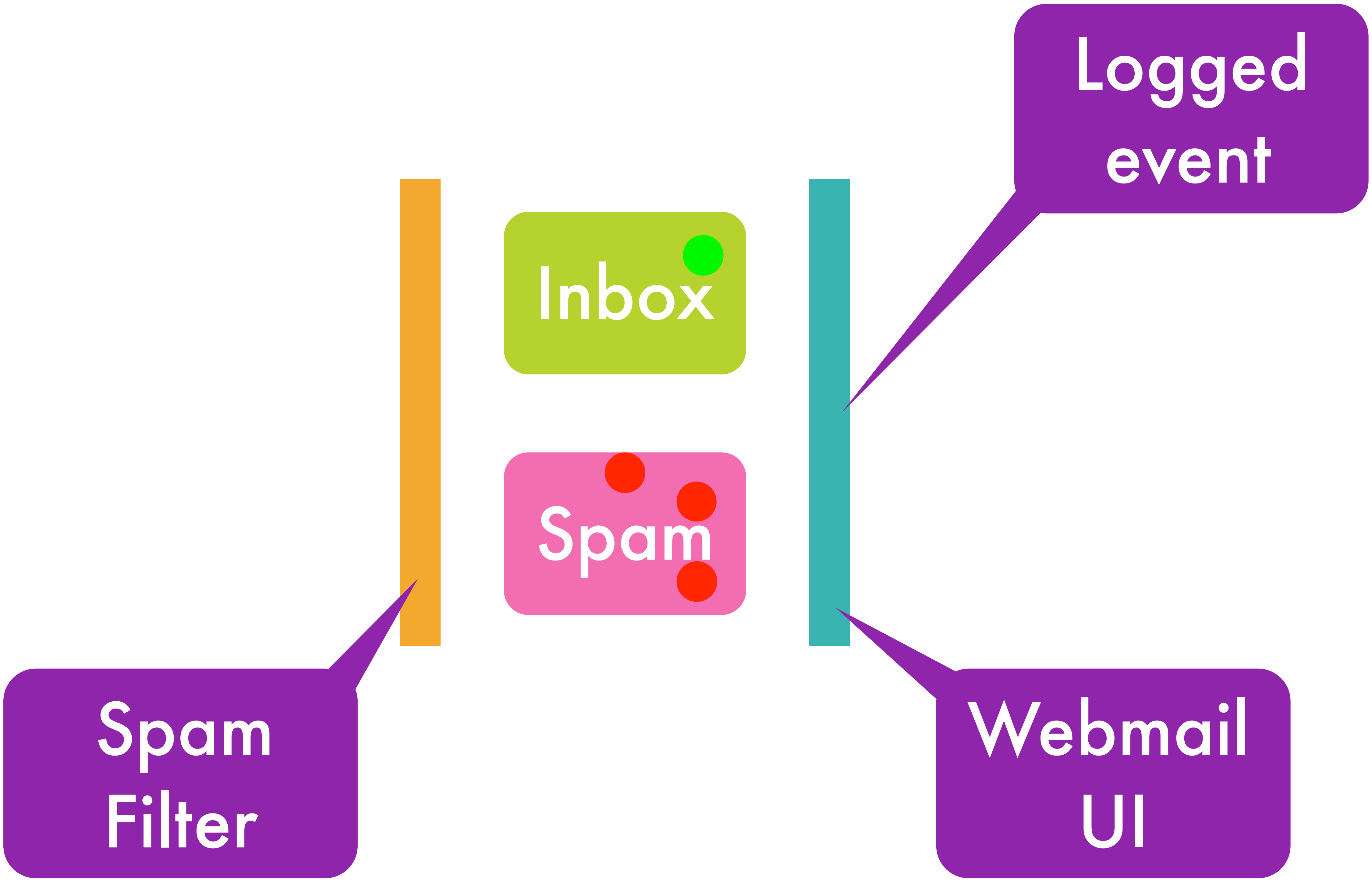
Running Example



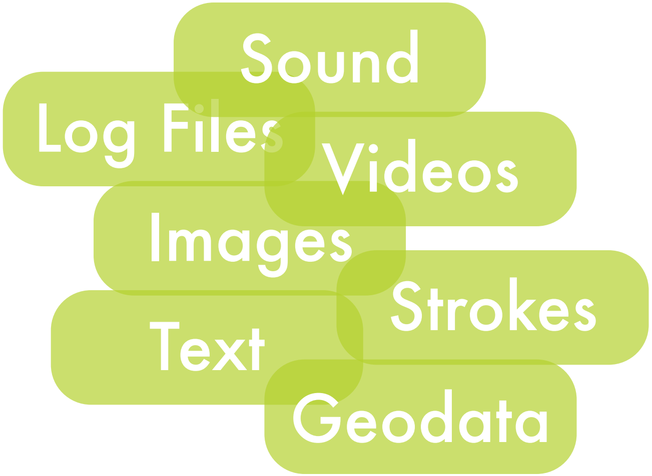
Running Example



Running Example

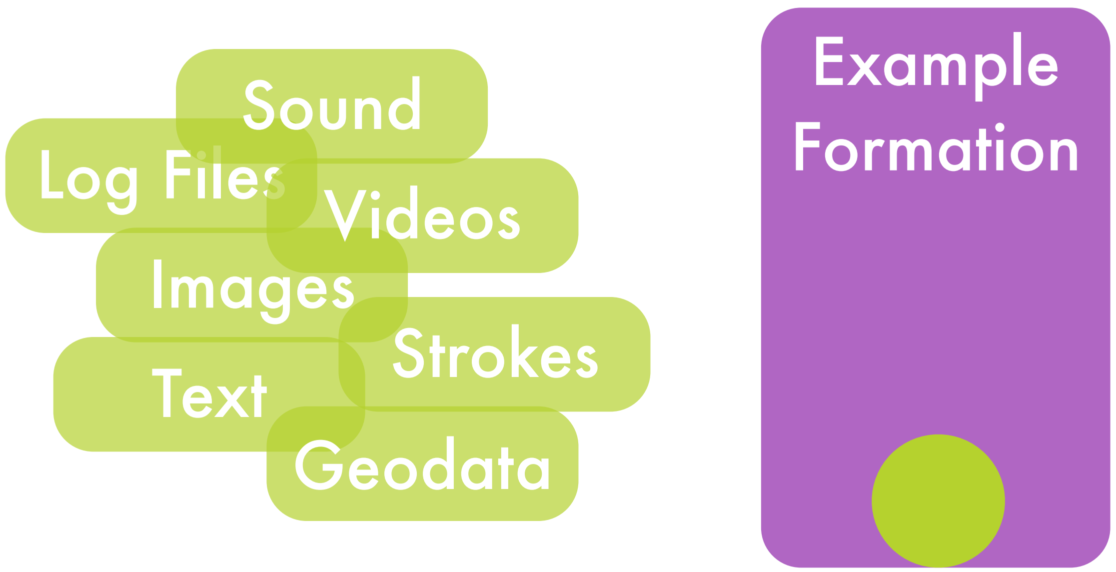


ML Workflow



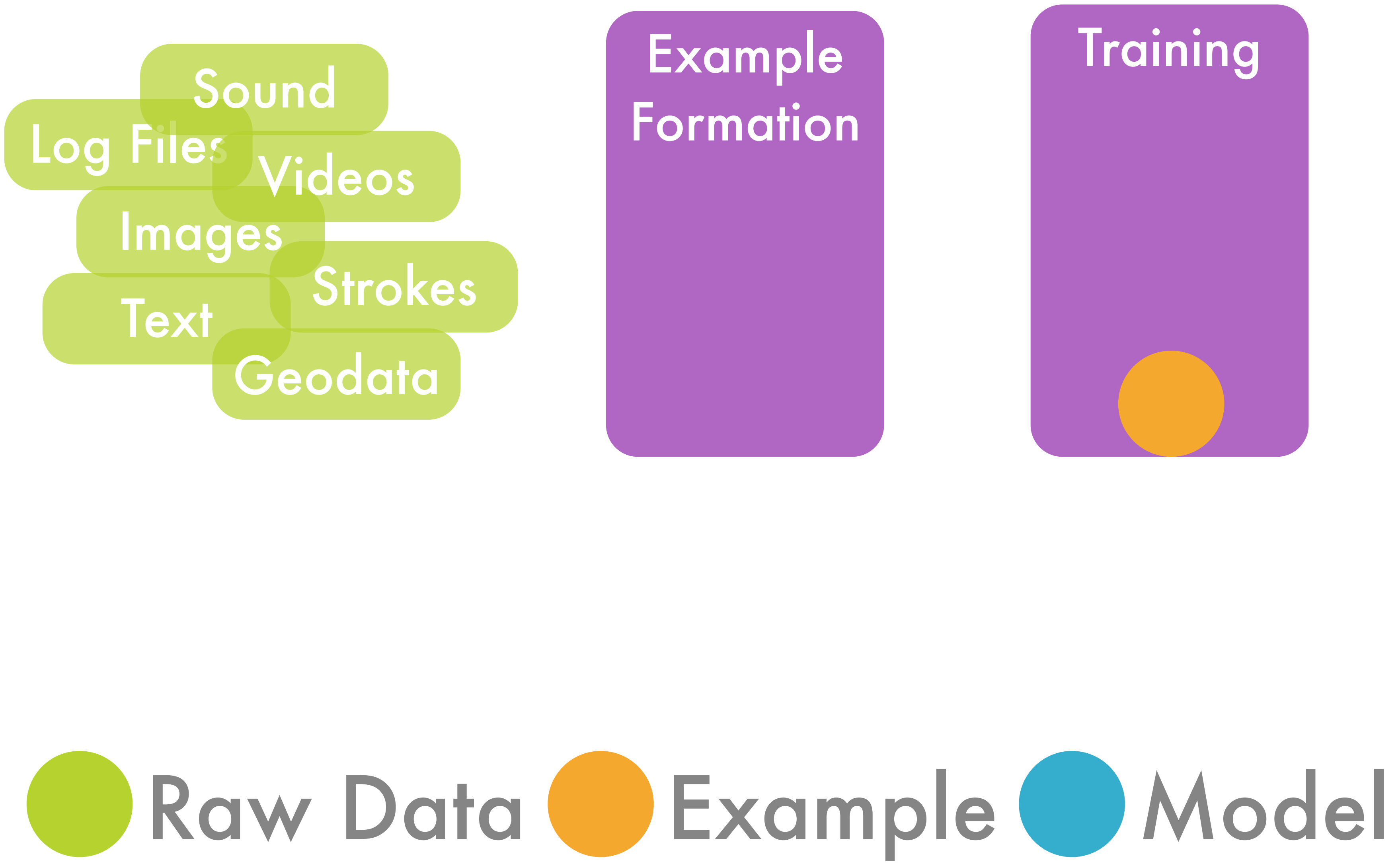
Raw Data Example Model

ML Workflow

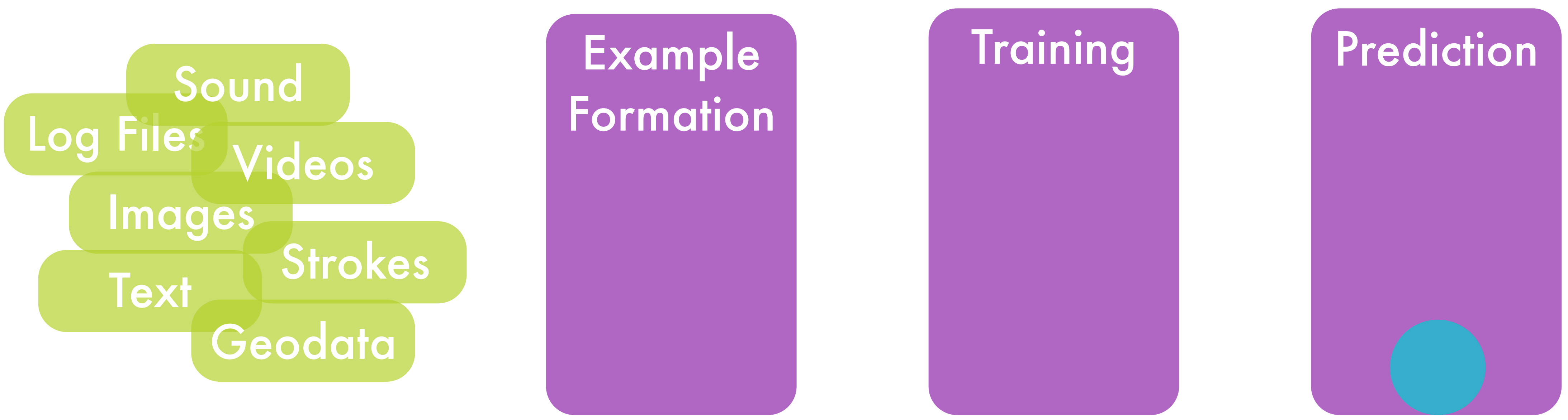


Raw Data Example Model

ML Workflow

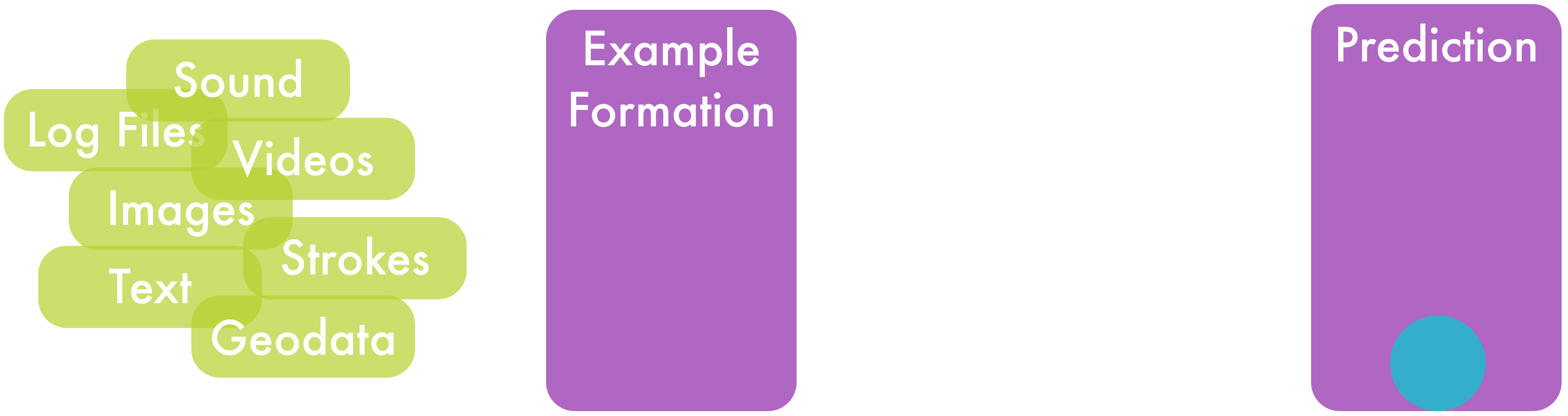

Sound
Log Files
Videos
Images
Strokes
Text
Geodata
Example Formation
Training
Raw Data
Example
Model

ML Workflow



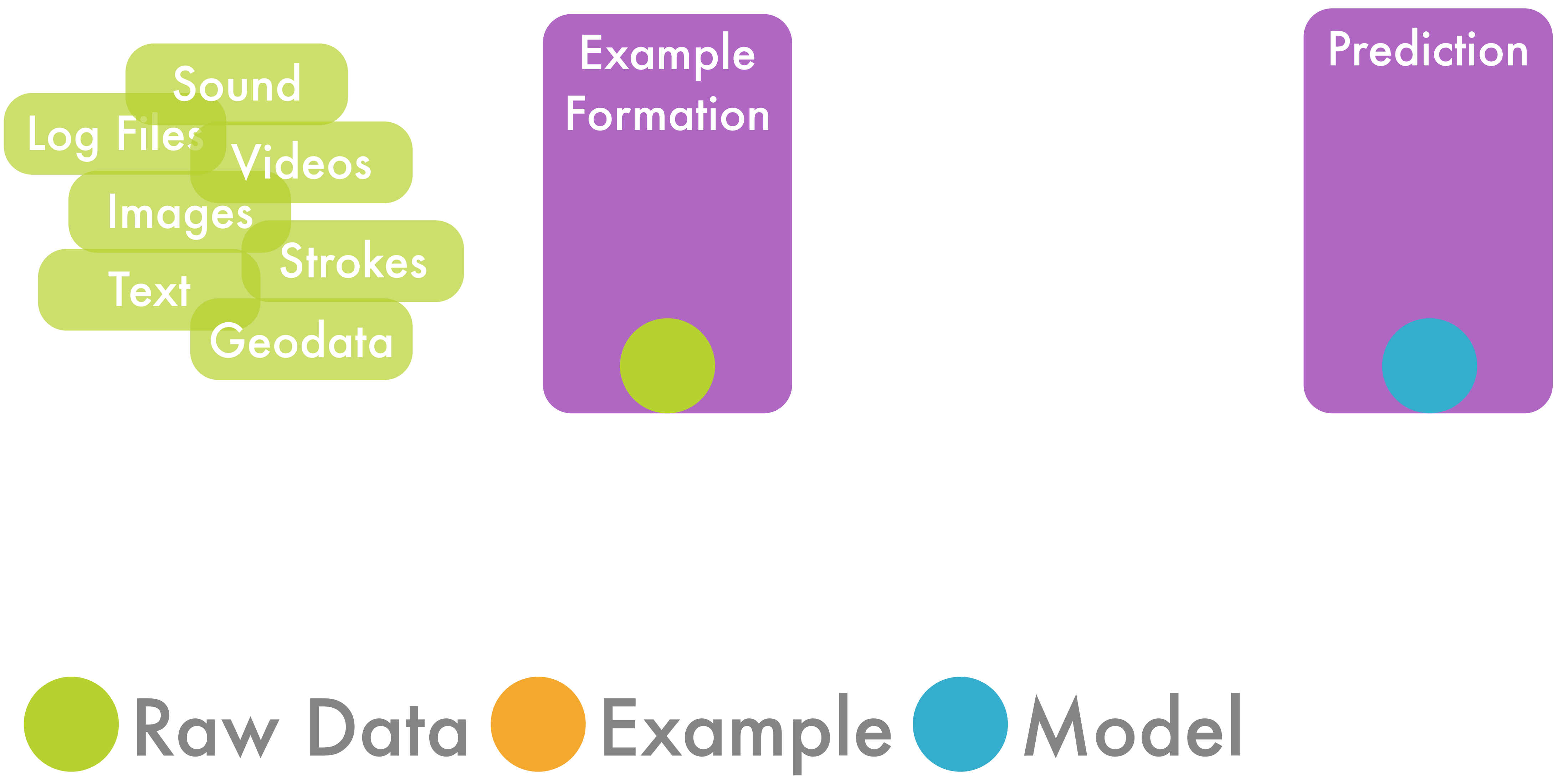
Raw Data Example Model

ML Workflow

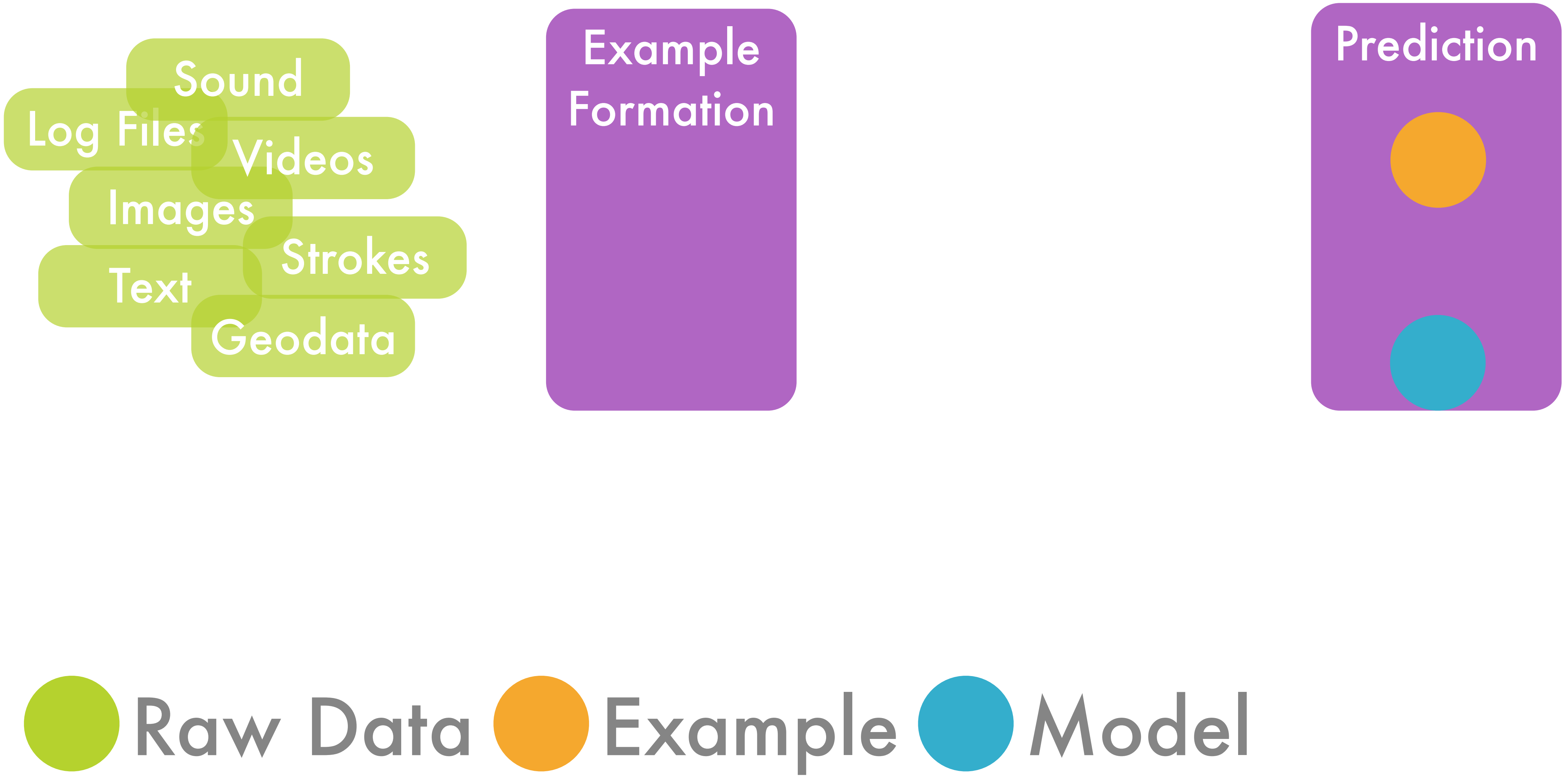


Raw Data Example Model

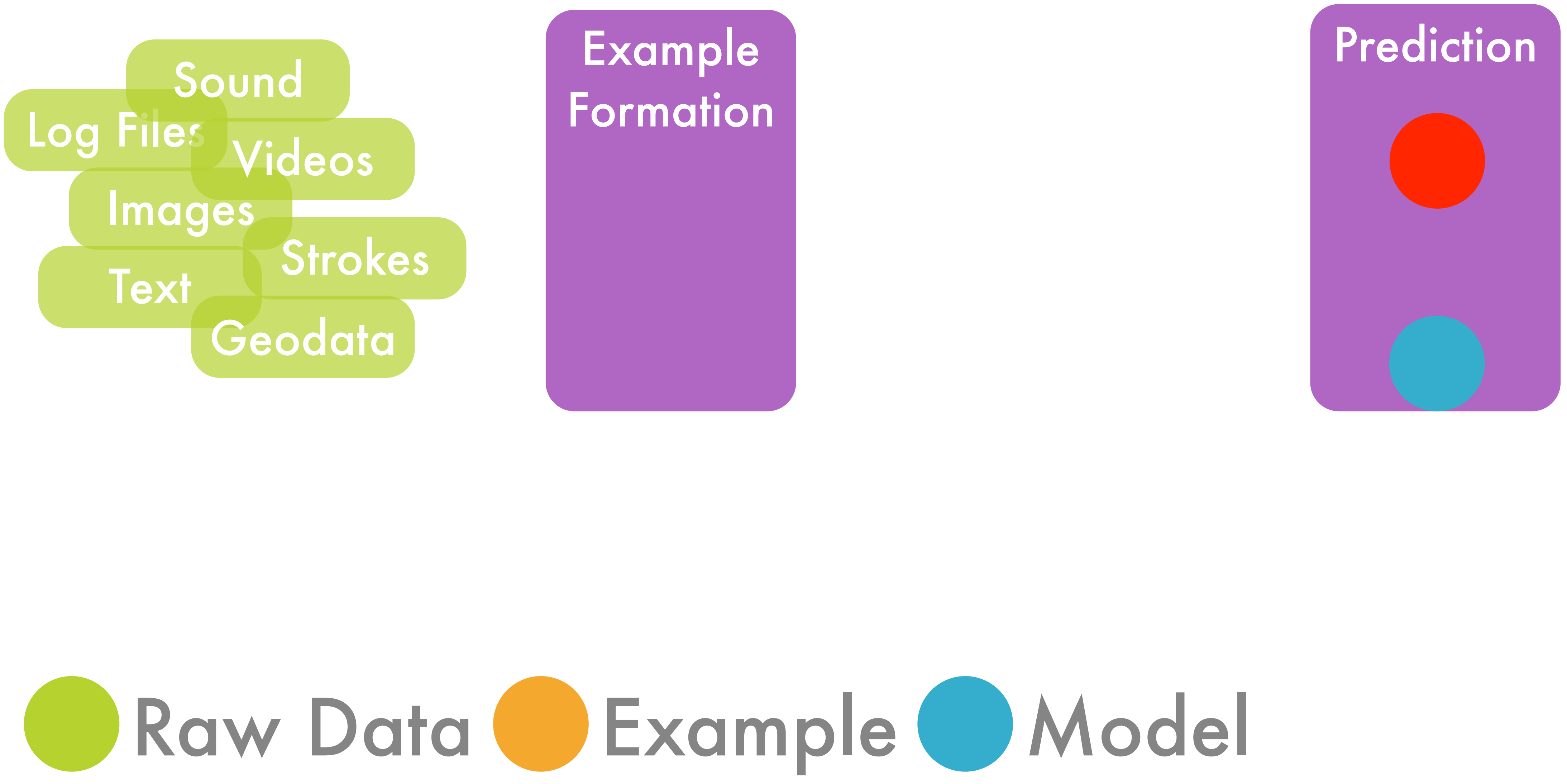
ML Workflow


Sound
Log Files
Videos
Images
Strokes
Text
Geodata
Example Formation
Prediction
Raw Data
Example
Model

ML Workflow

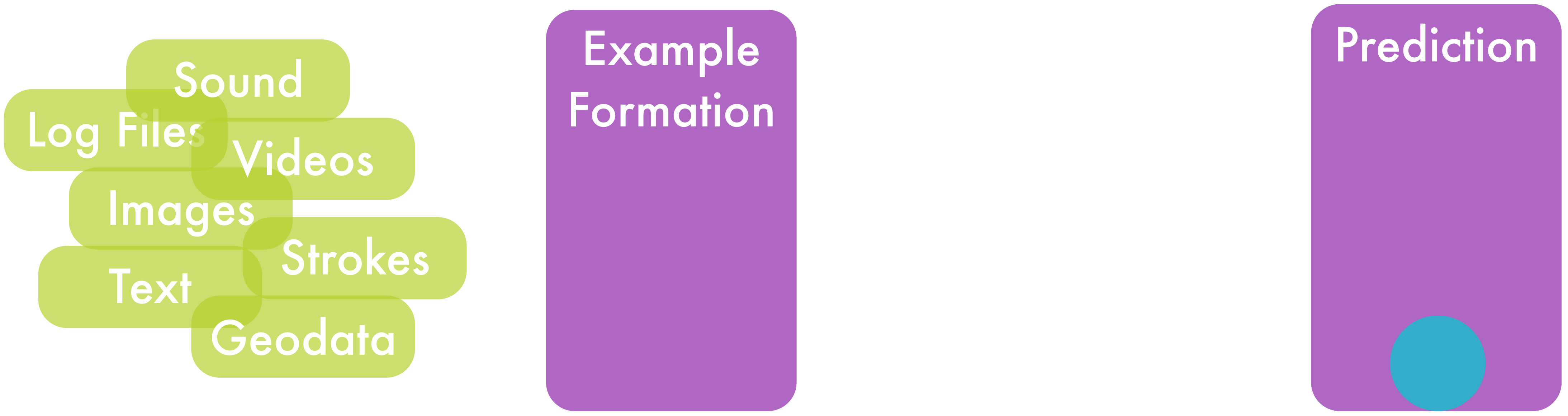

Sound
Log Files
Videos
Images
Strokes
Text
Geodata
Example Formation
Prediction
Raw Data
Example
Model

ML Workflow

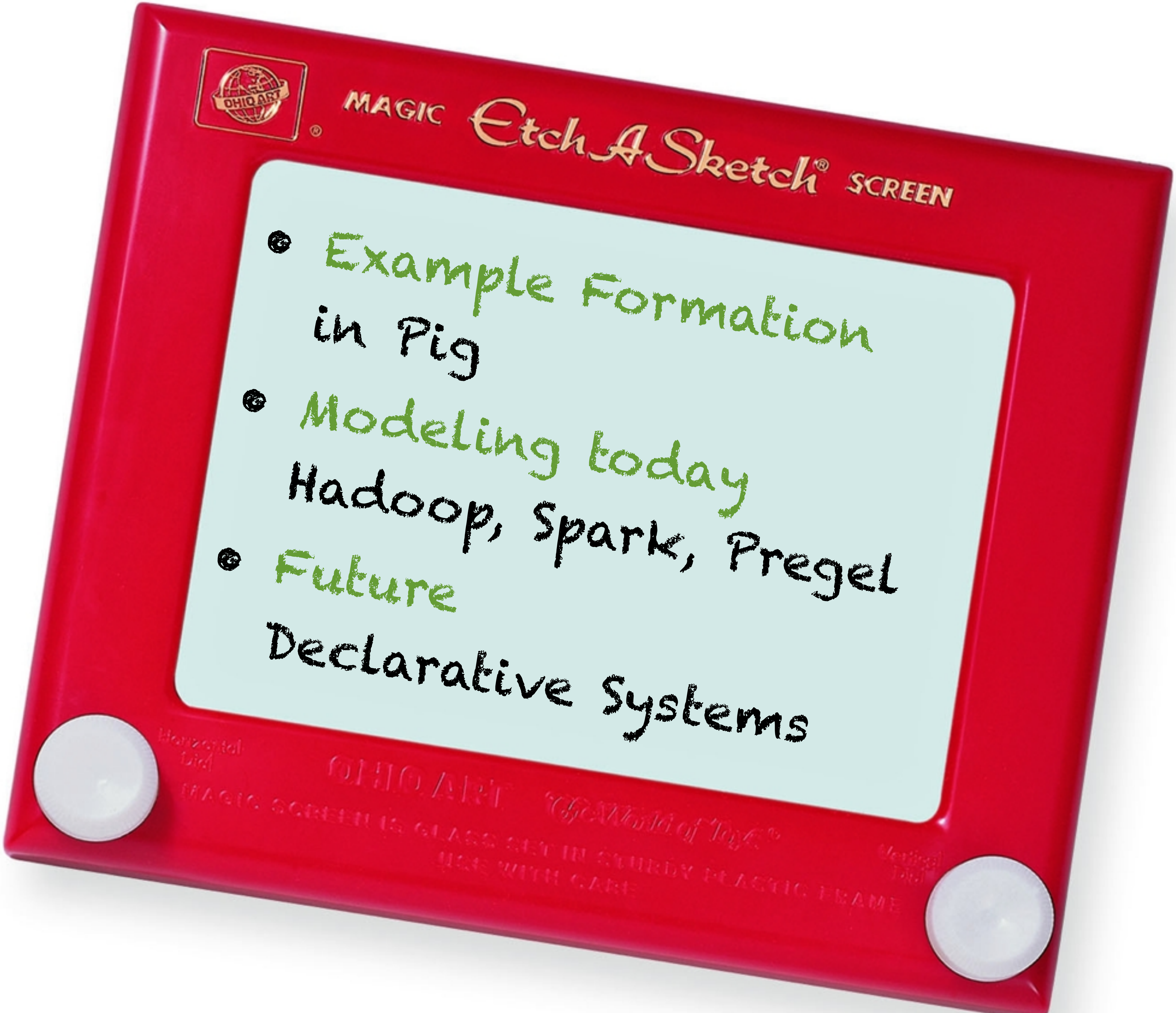


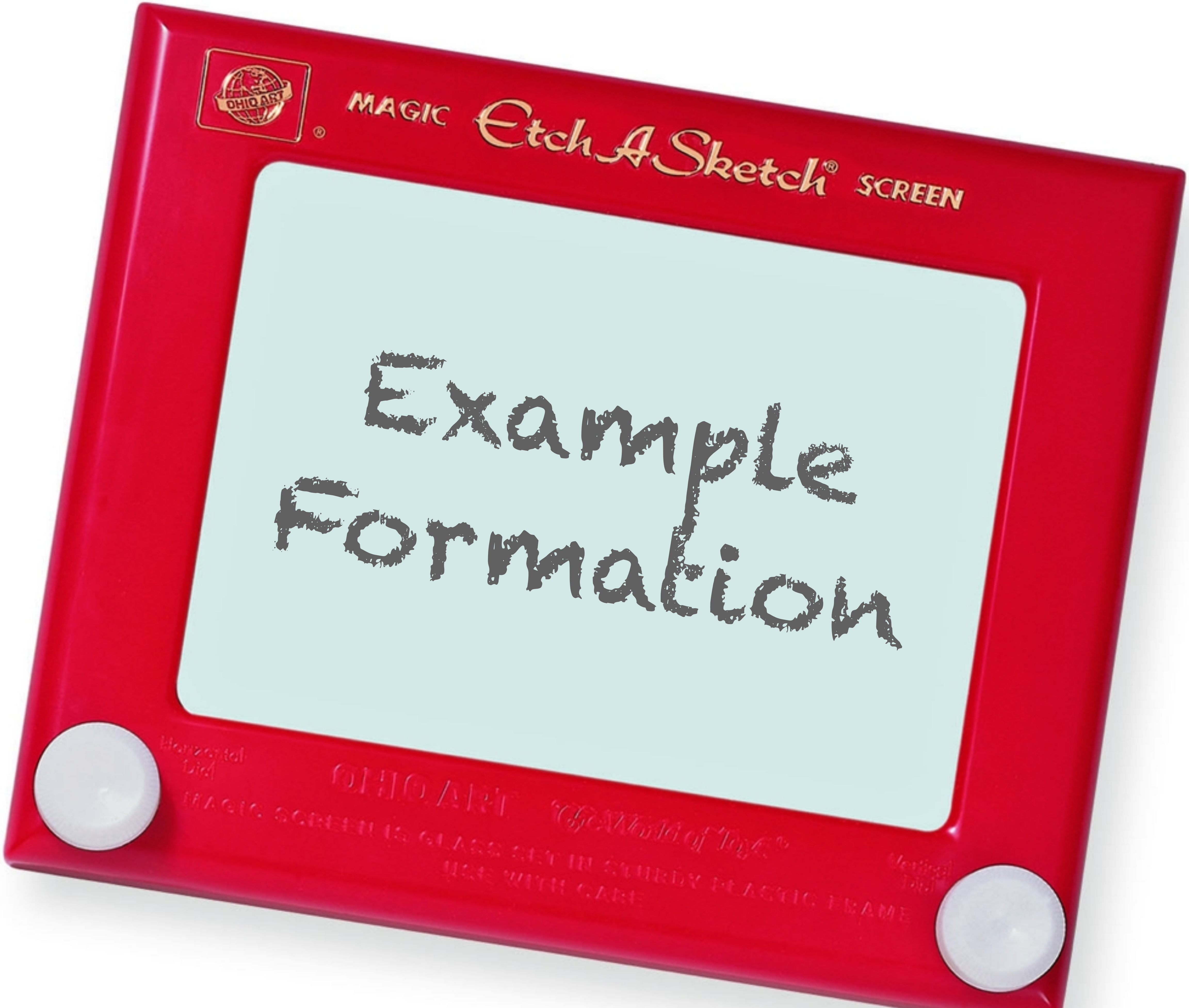
Sound
Log Files
Videos
Images
Strokes
Text
Geodata
Example Formation
Prediction
Raw Data
Example
Model

ML Workflow



Raw Data Example Model


MAGIC Etch A Sketch® SCREEN
OHIO ART
• Example Formation in Pig
• Modeling today Hadoop, Spark, Pregel
• Future Declarative Systems

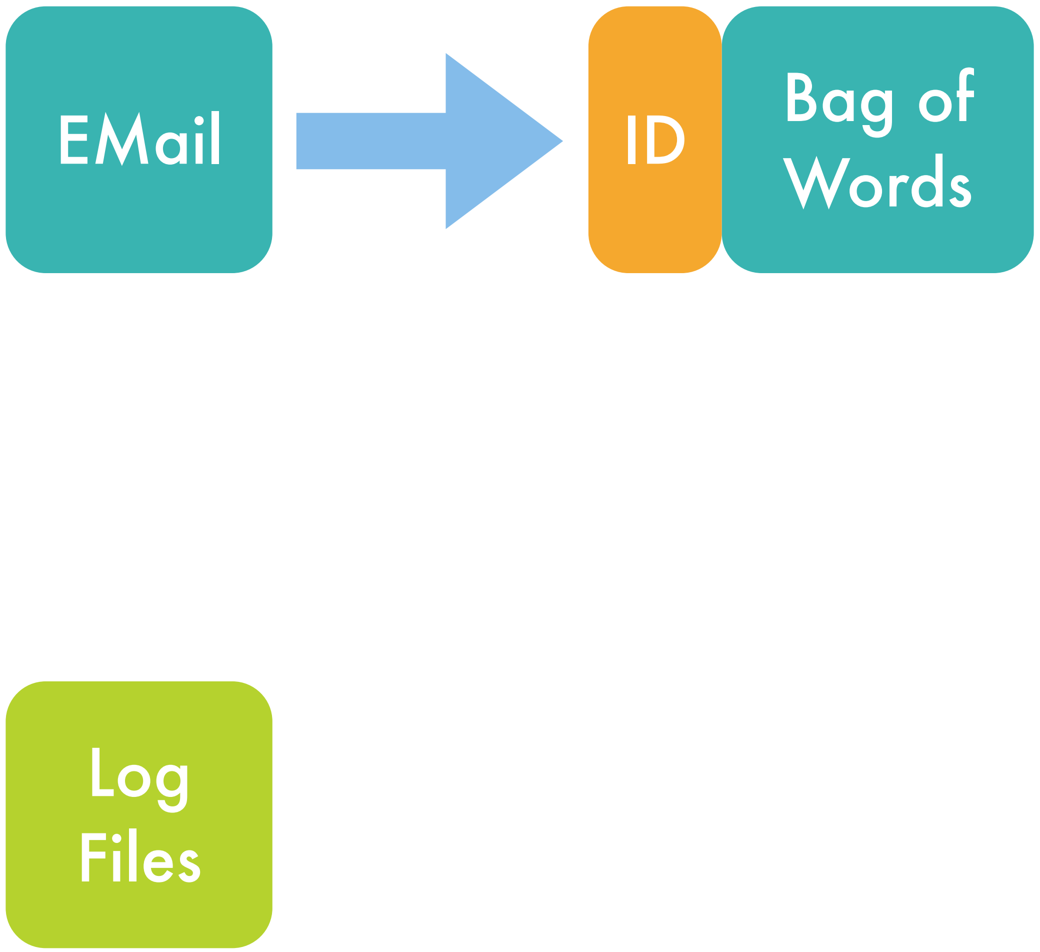

OHIO ART
MAGIC Etch A Sketch SCREEN
Example
Formation

Example Formation

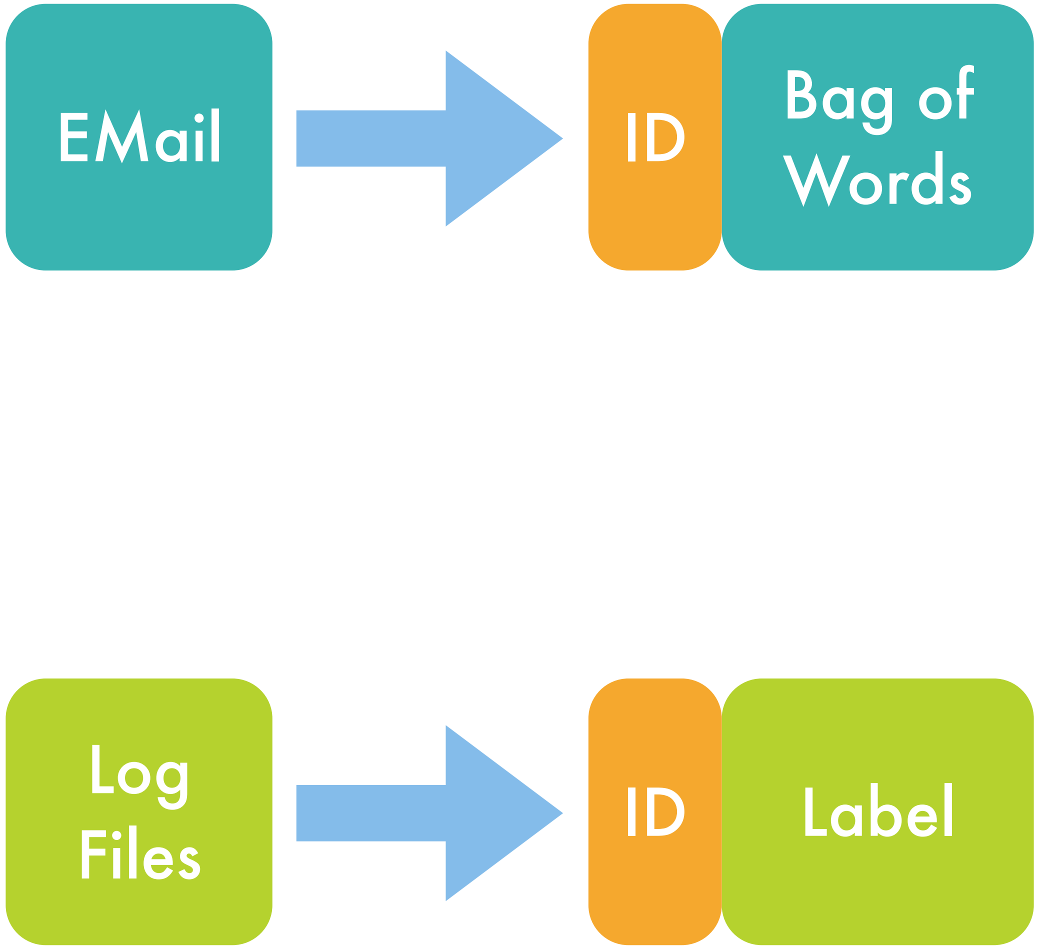
EMail

Log Files

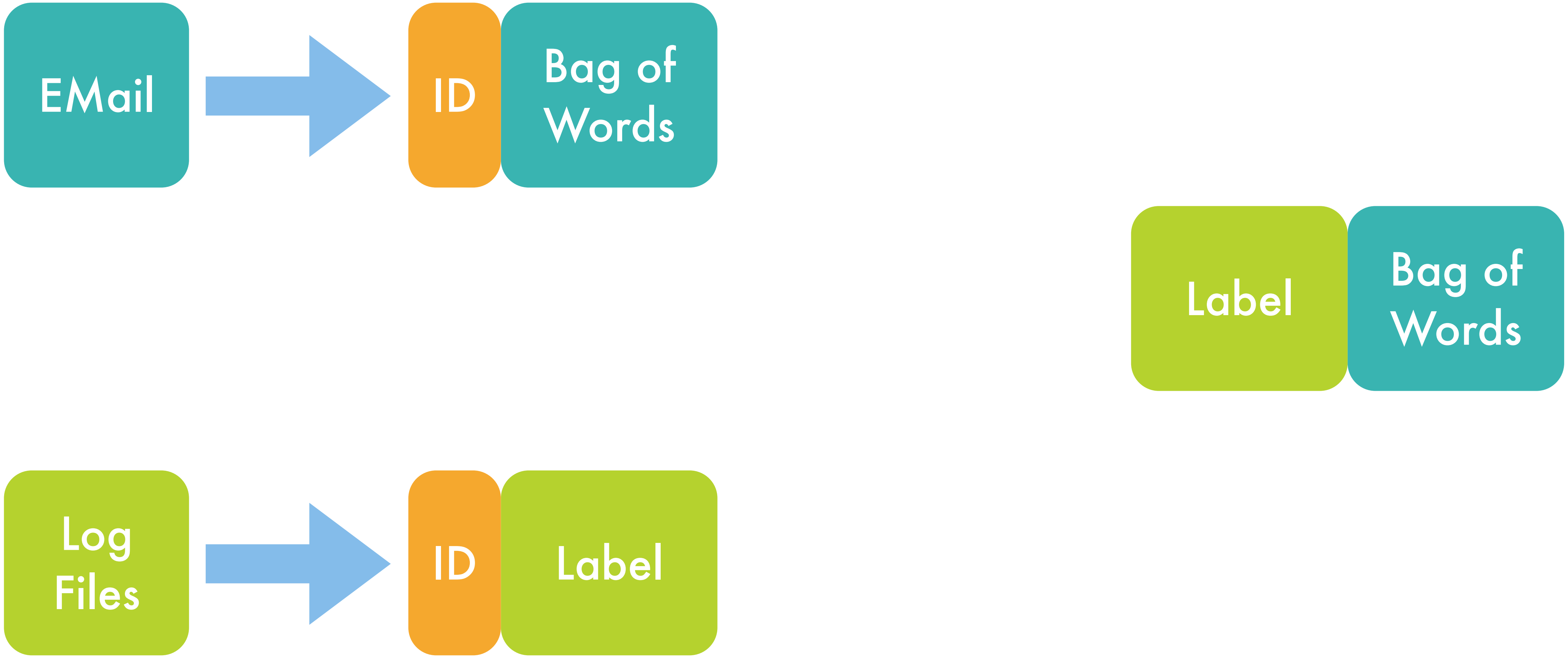
Example Formation



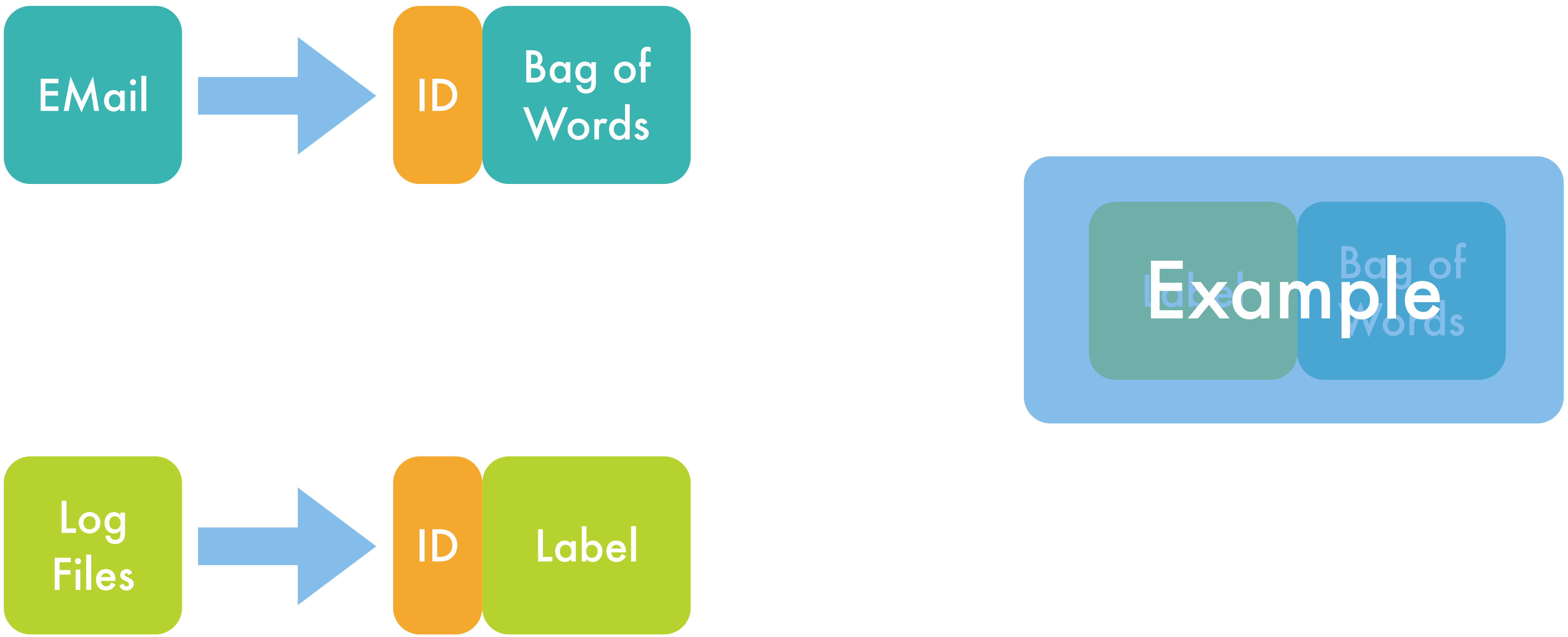
Example Formation



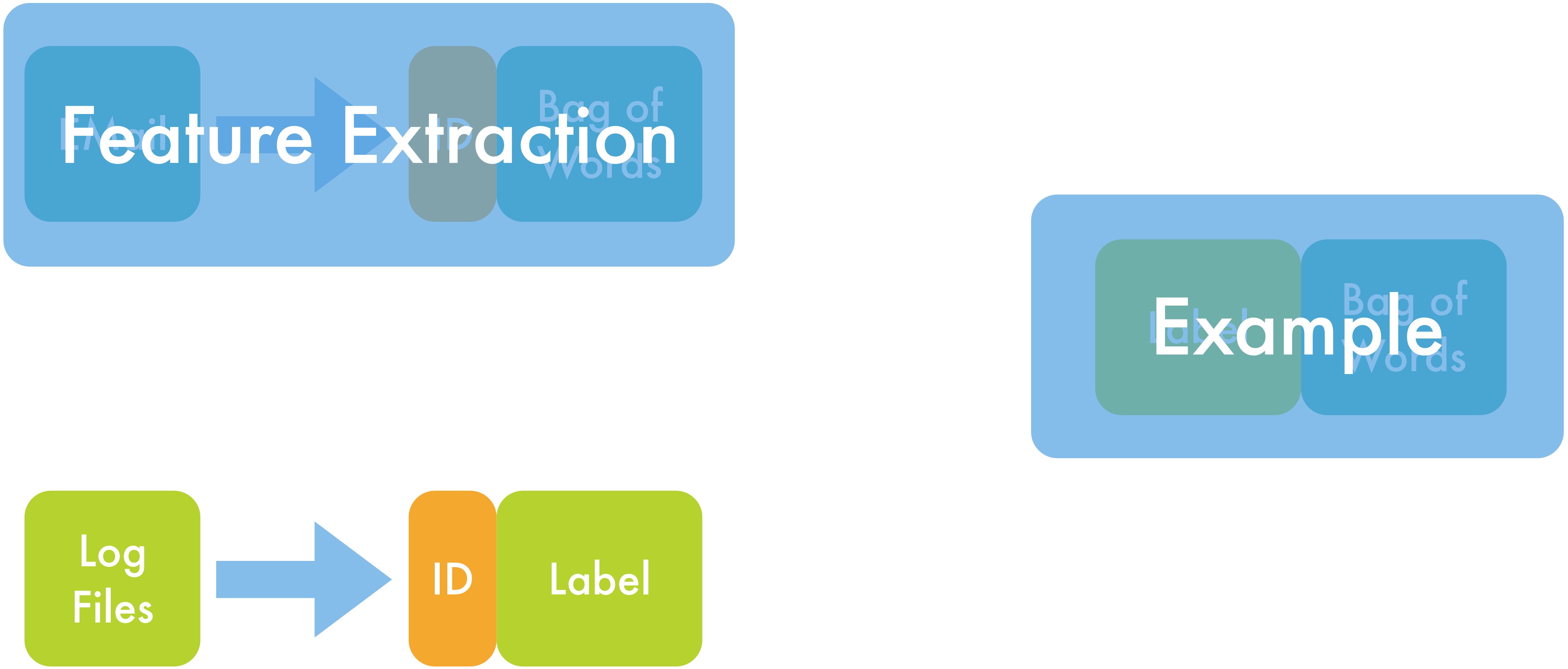
Example Formation



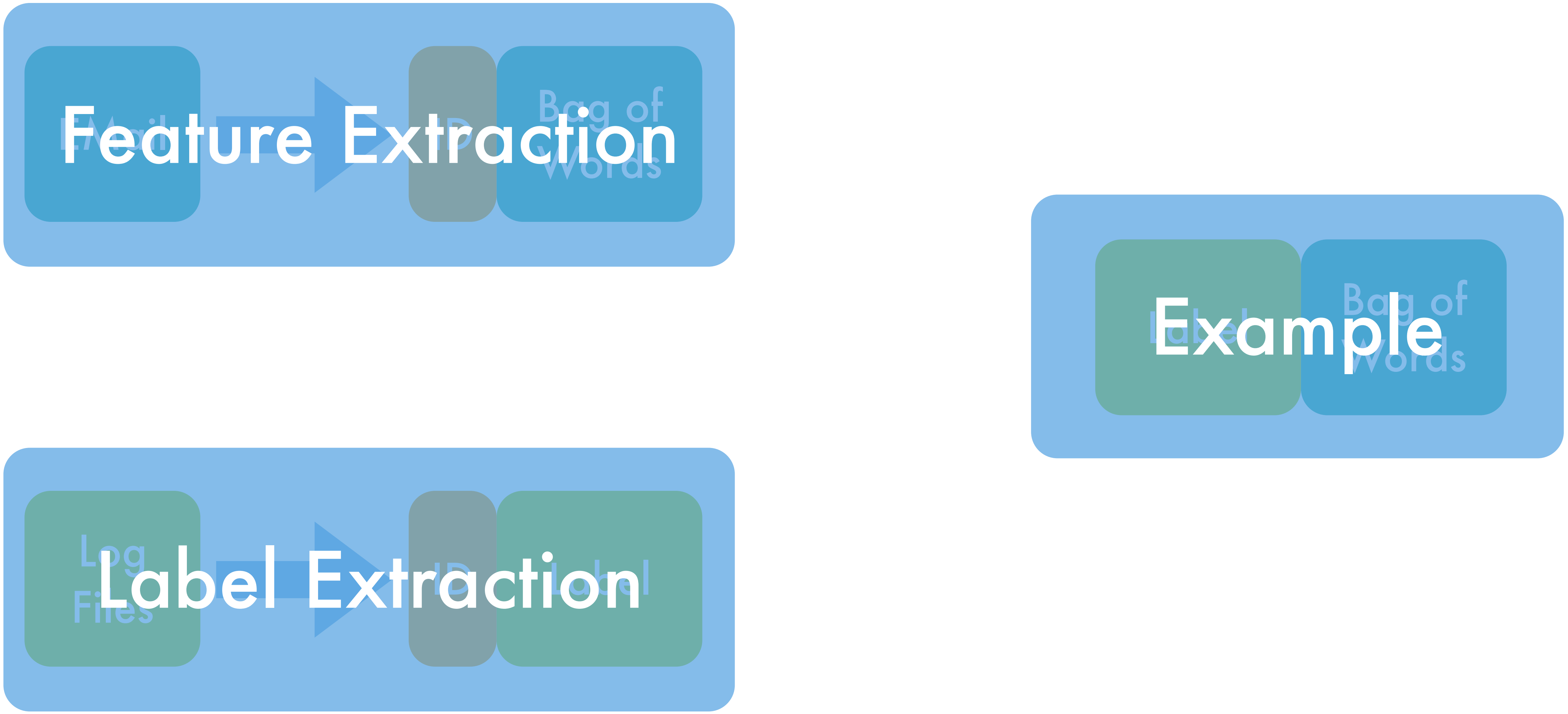
Example Formation



Example Formation



Example Formation

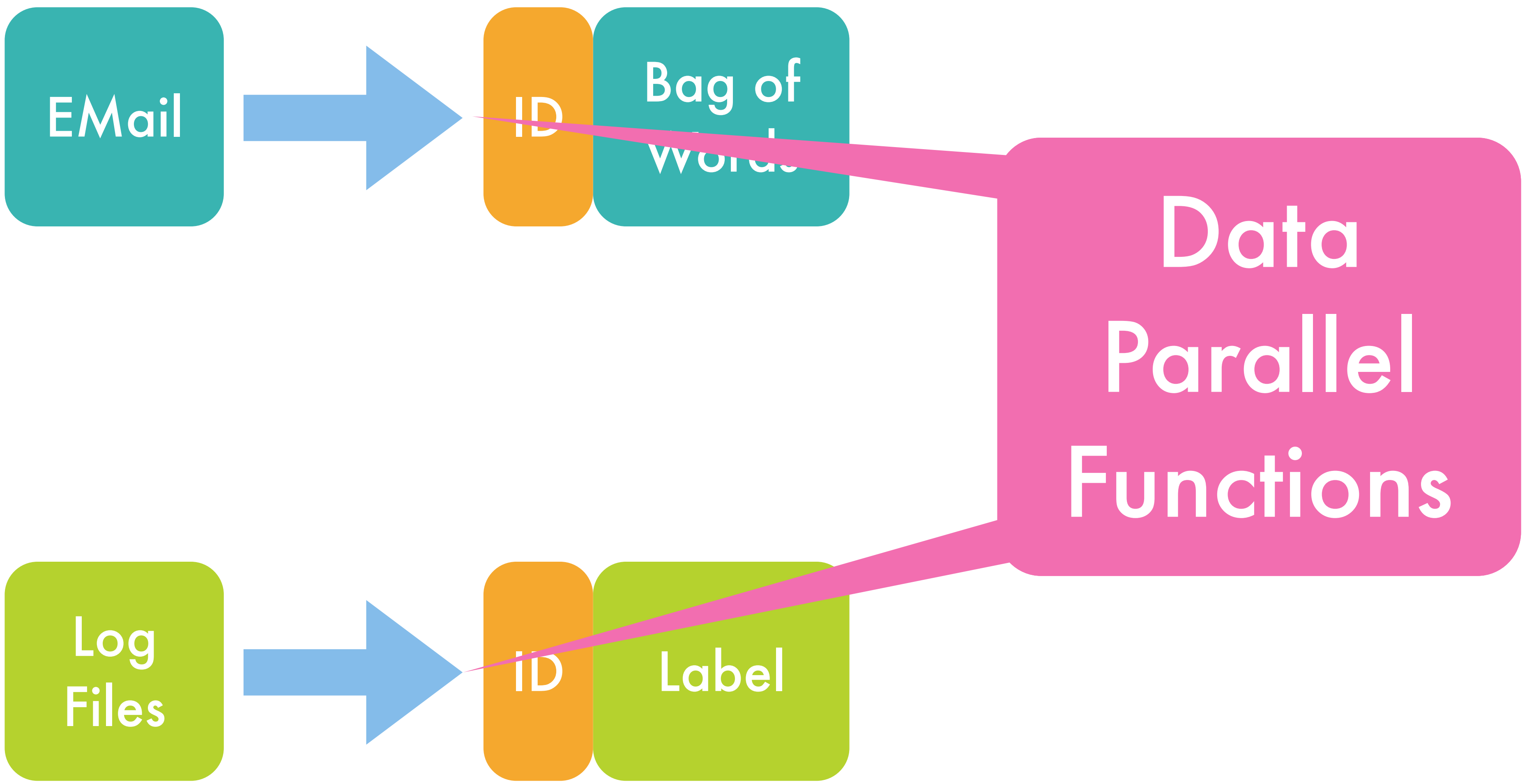


Requirements

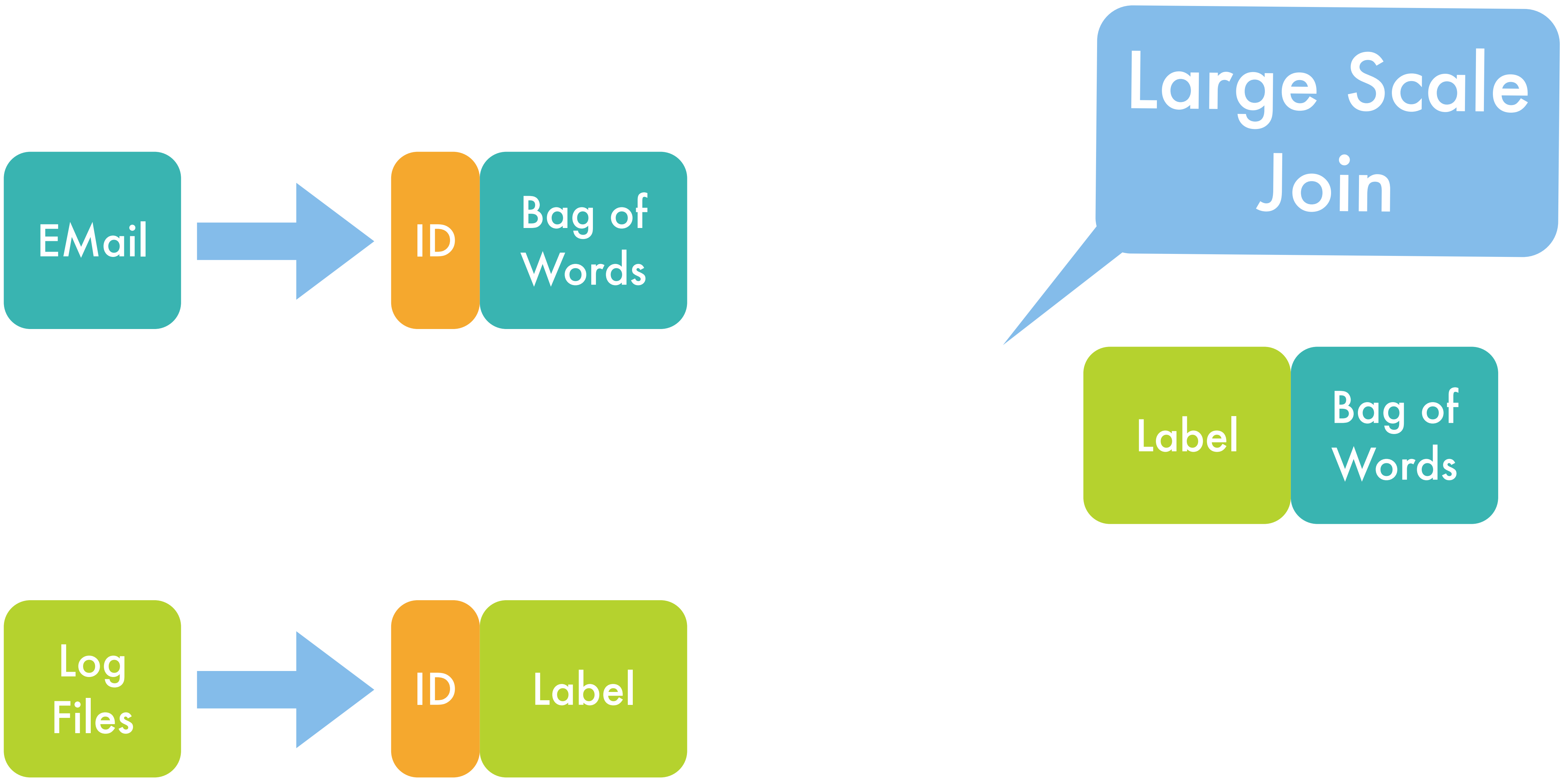
EMail

Log Files

Requirements



Requirements



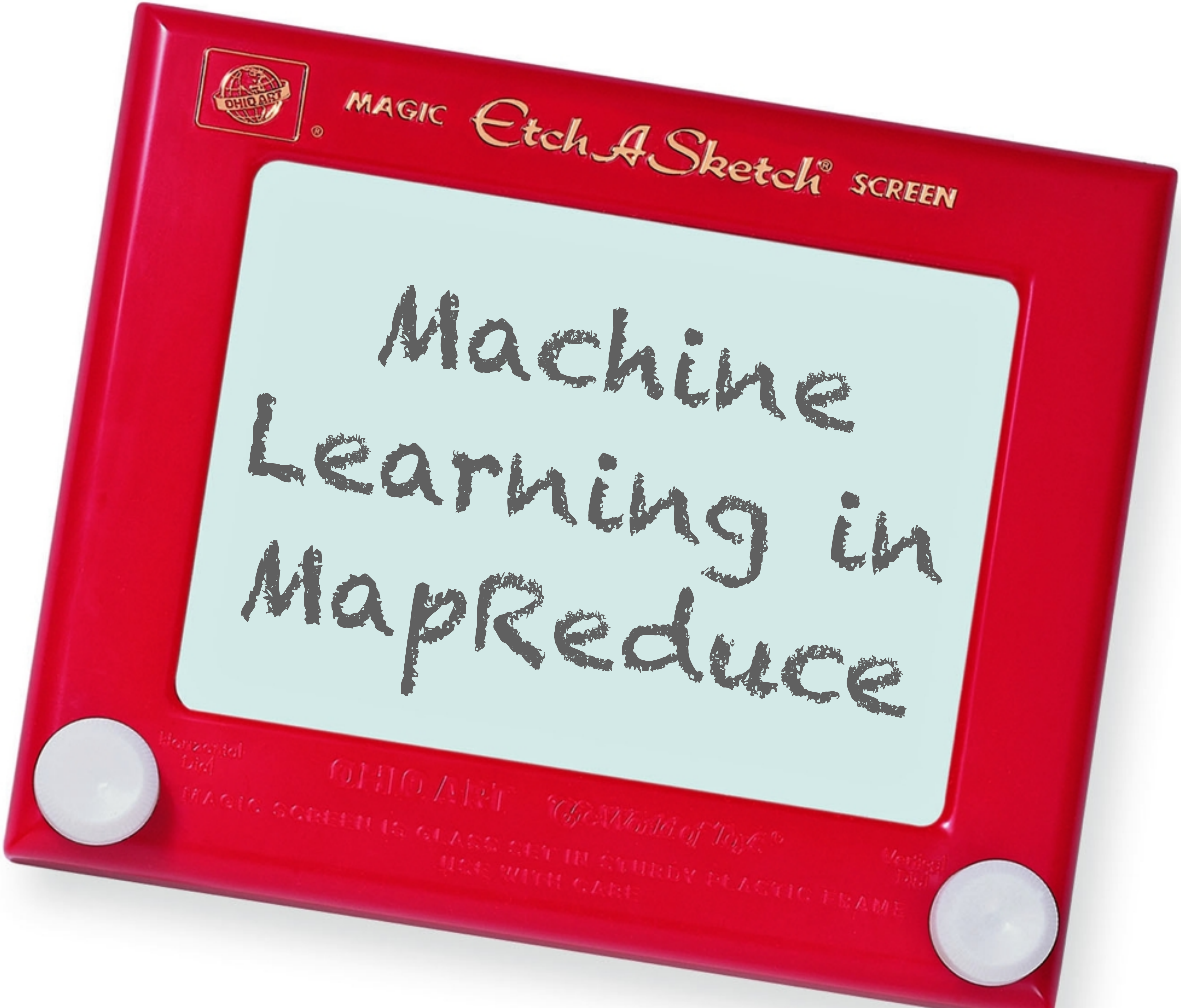
Apache Pig

- Relational Query Language
- Similar to SQL
- Performs runtime optimizations
- Executes Queries on Apache Hadoop
- Developed and heavily used by Yahoo!
- Open Source (Apache)

Pig: Example Formation

- **Feature and Label Extraction**
  - User Defined Function
  - Applied via `FOREACH … GENERATE`

- **Example formation**
  - `JOIN` between the outputs of the above

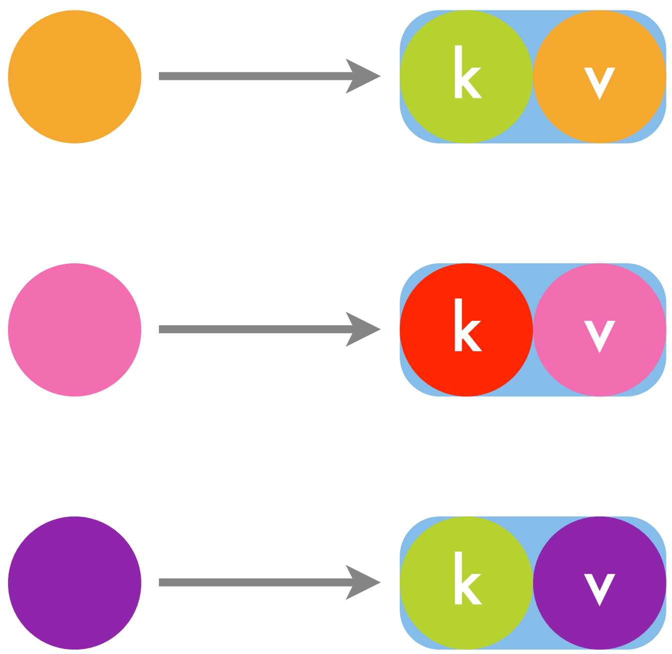

OHIO ART
MAGIC Etch A Sketch® SCREEN
Machine Learning in MapReduce
OHIO ART

MapReduce

- Parallel, Distributed programming framework
- User defines two functions:
  - `map(x)` emits `(key, value)` pairs
  - `reduce(k, x[])` gets all values for a key, produces output

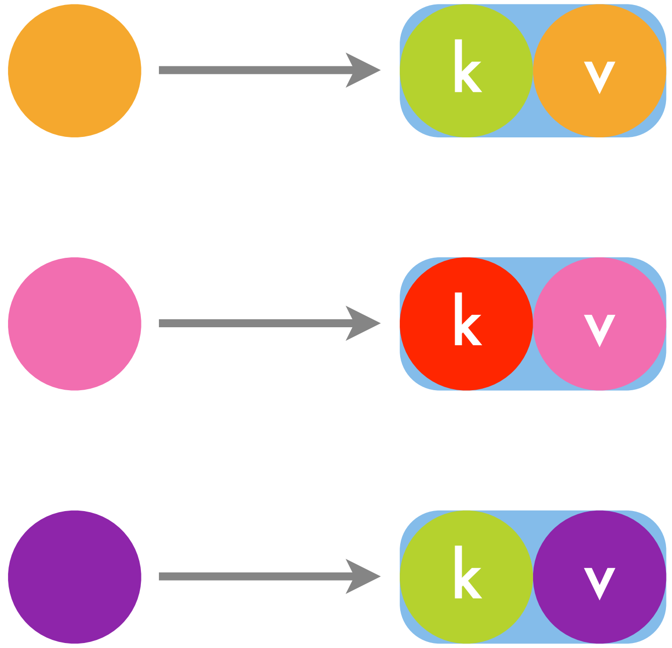
MapReduce

MapReduce



Map

MapReduce

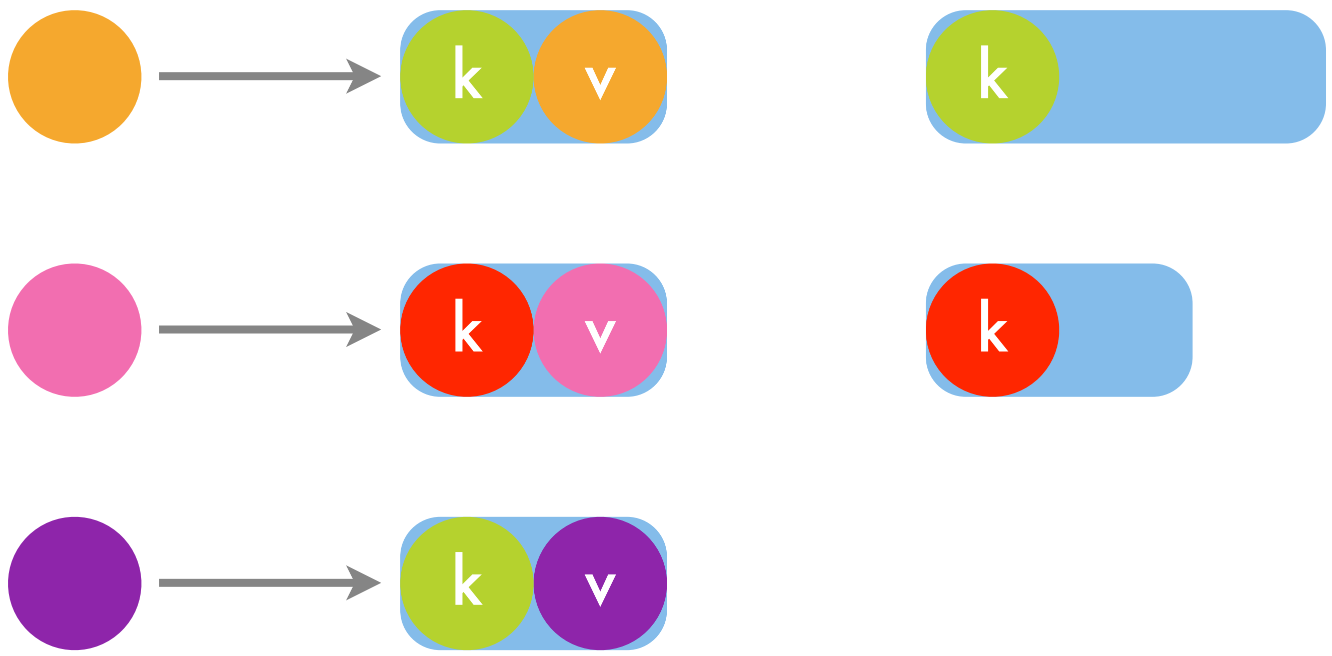


Map

GroupBy (Shuffle)

Reduce

MapReduce

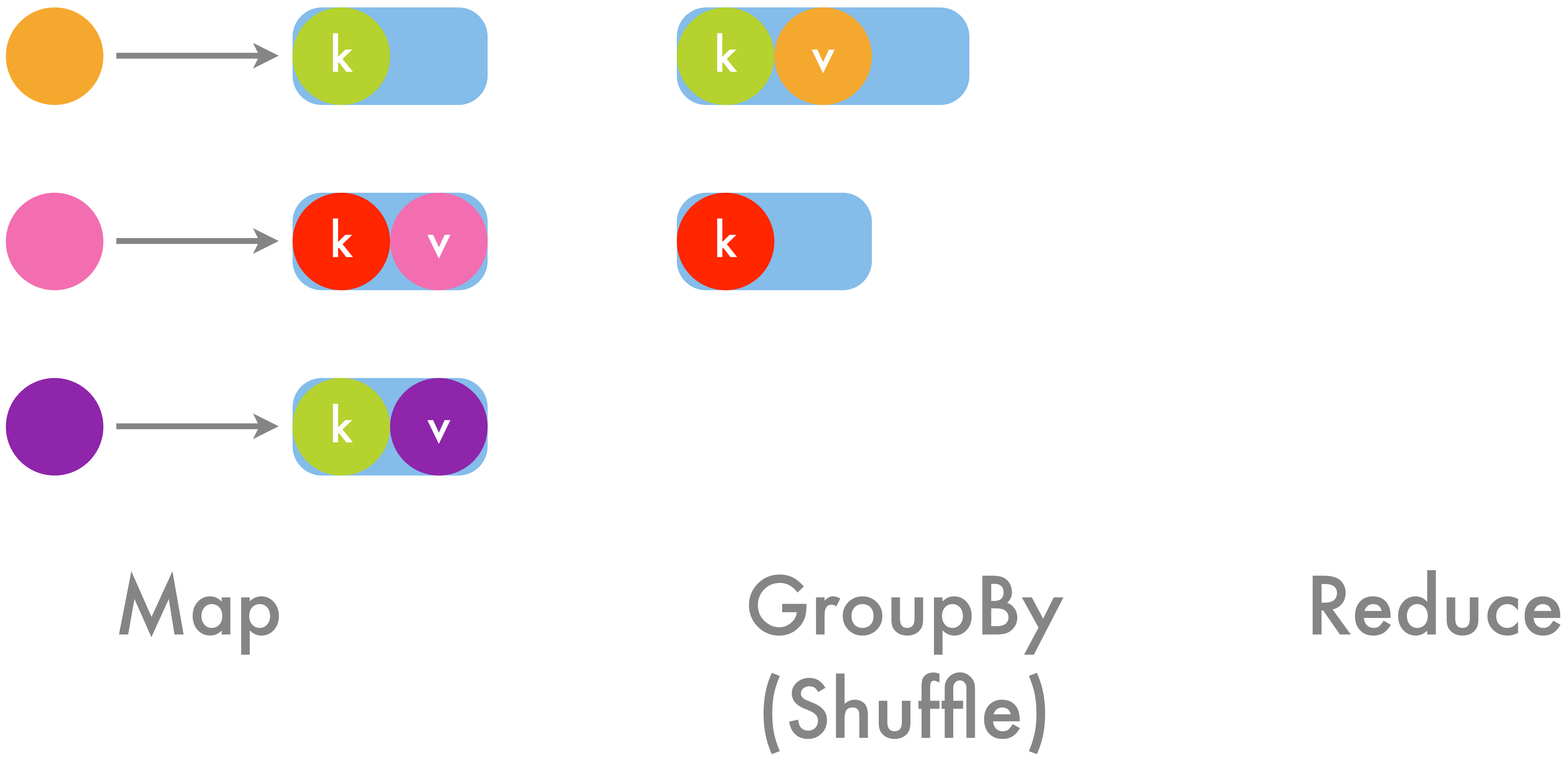


Map

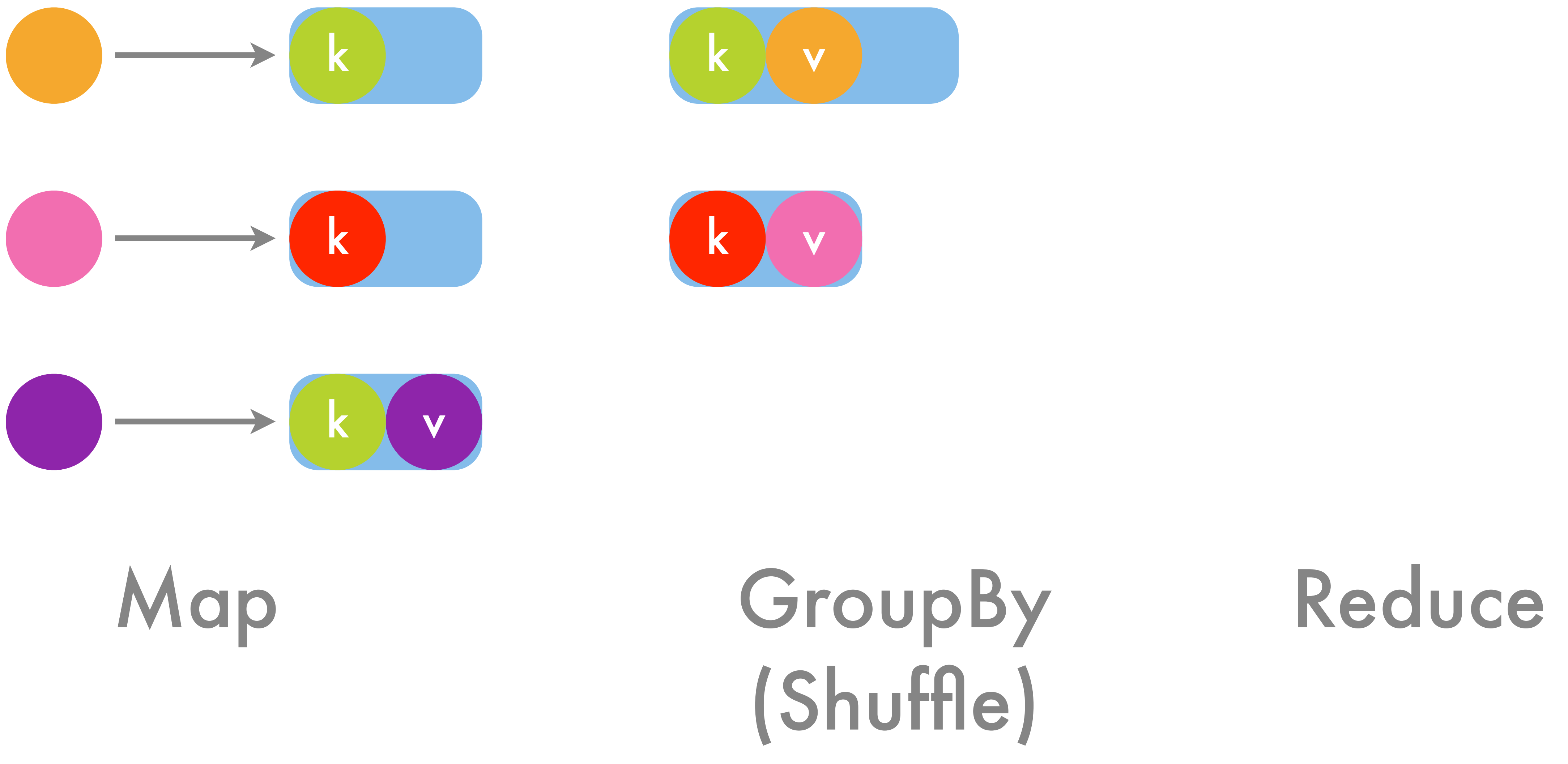
GroupBy
(Shuffle)

Reduce

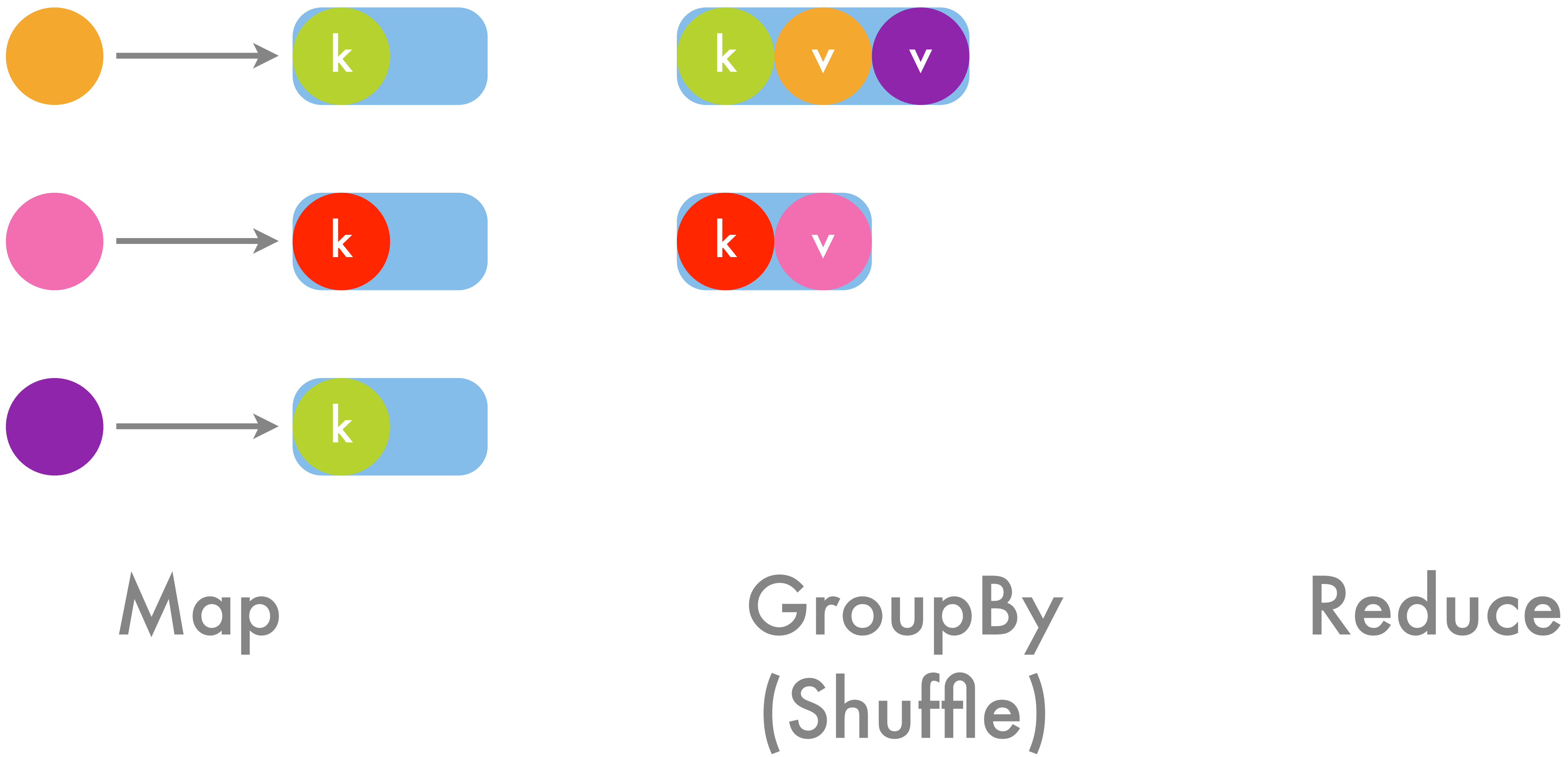
MapReduce



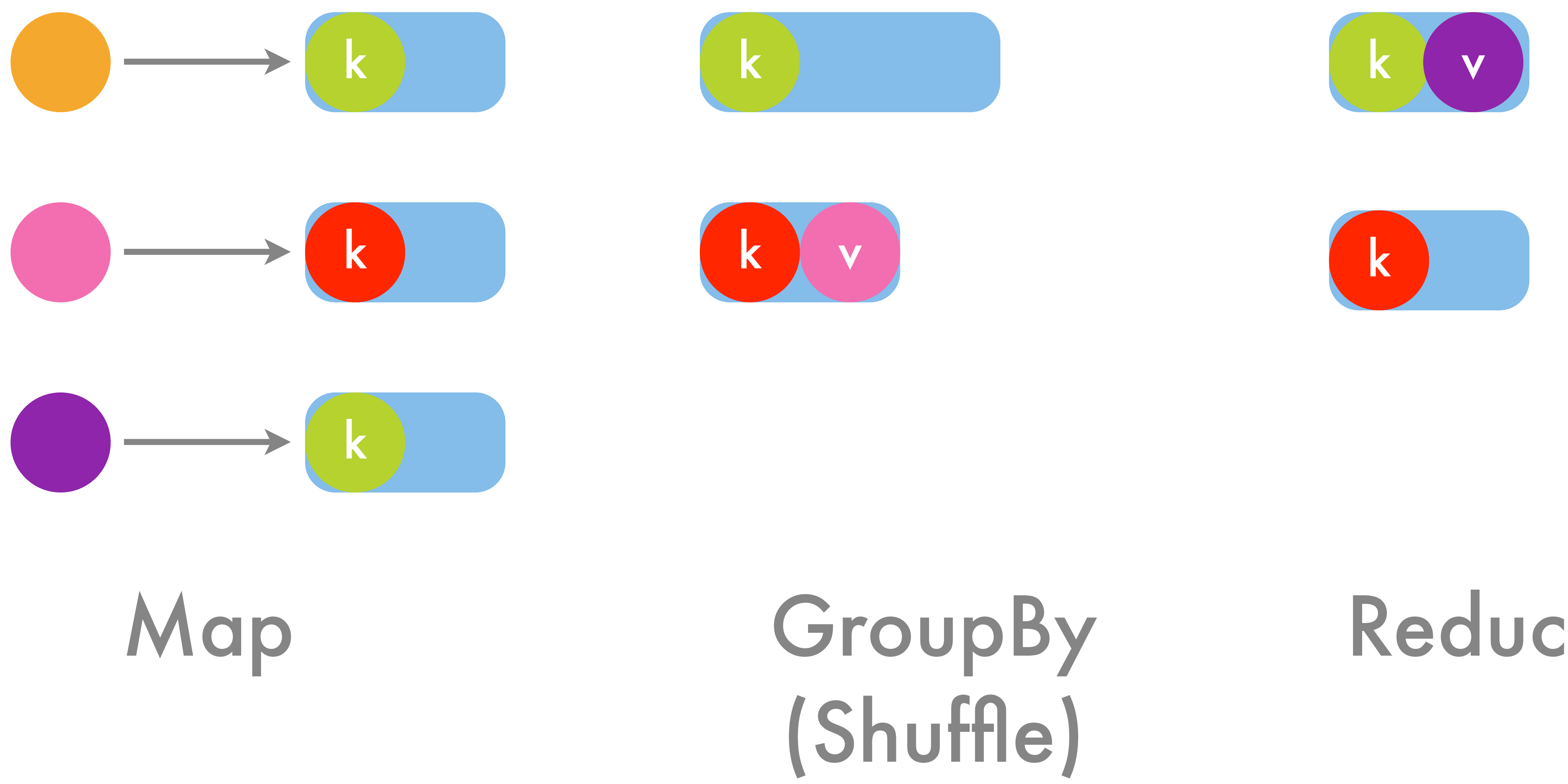
MapReduce



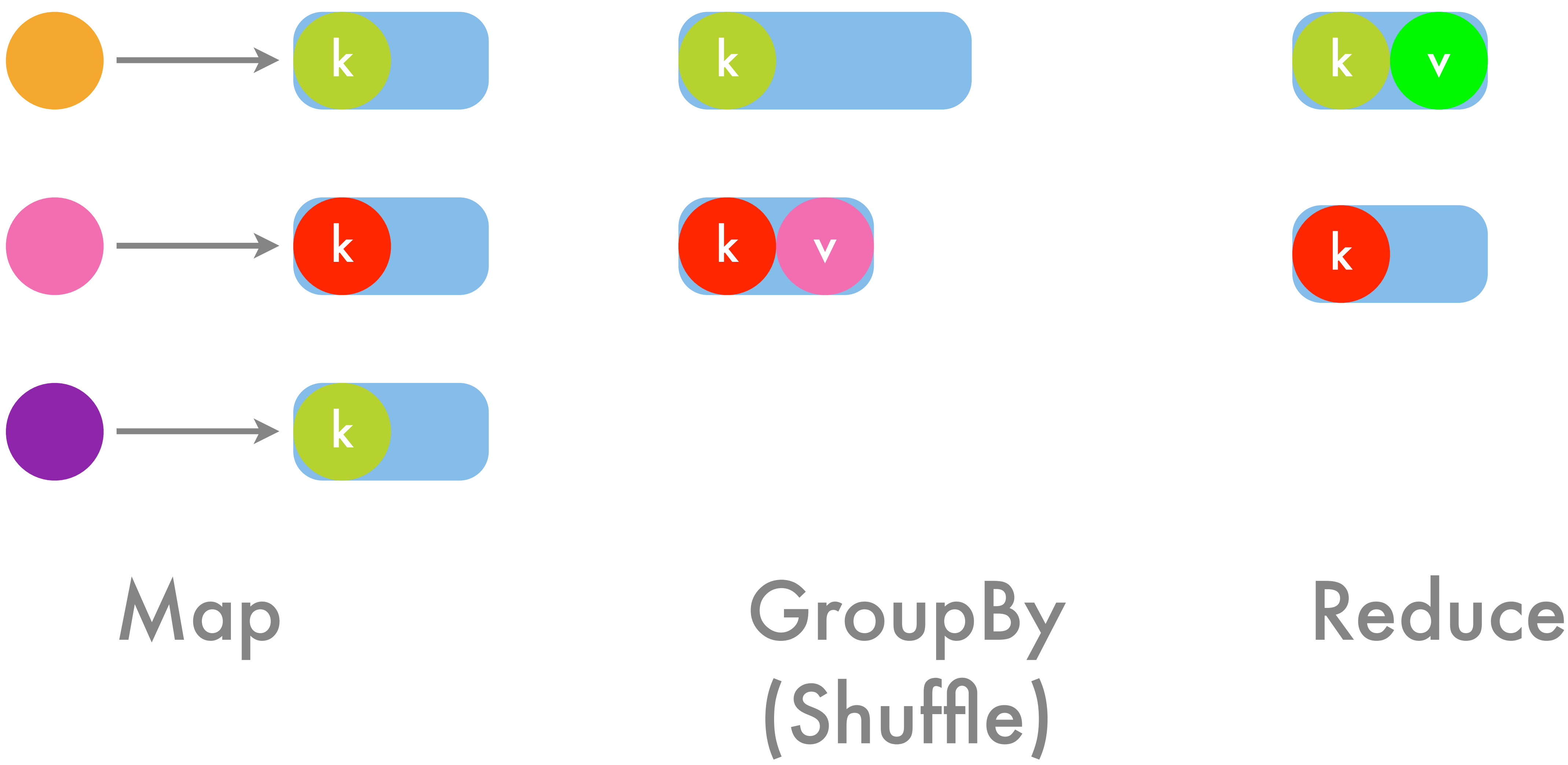
MapReduce



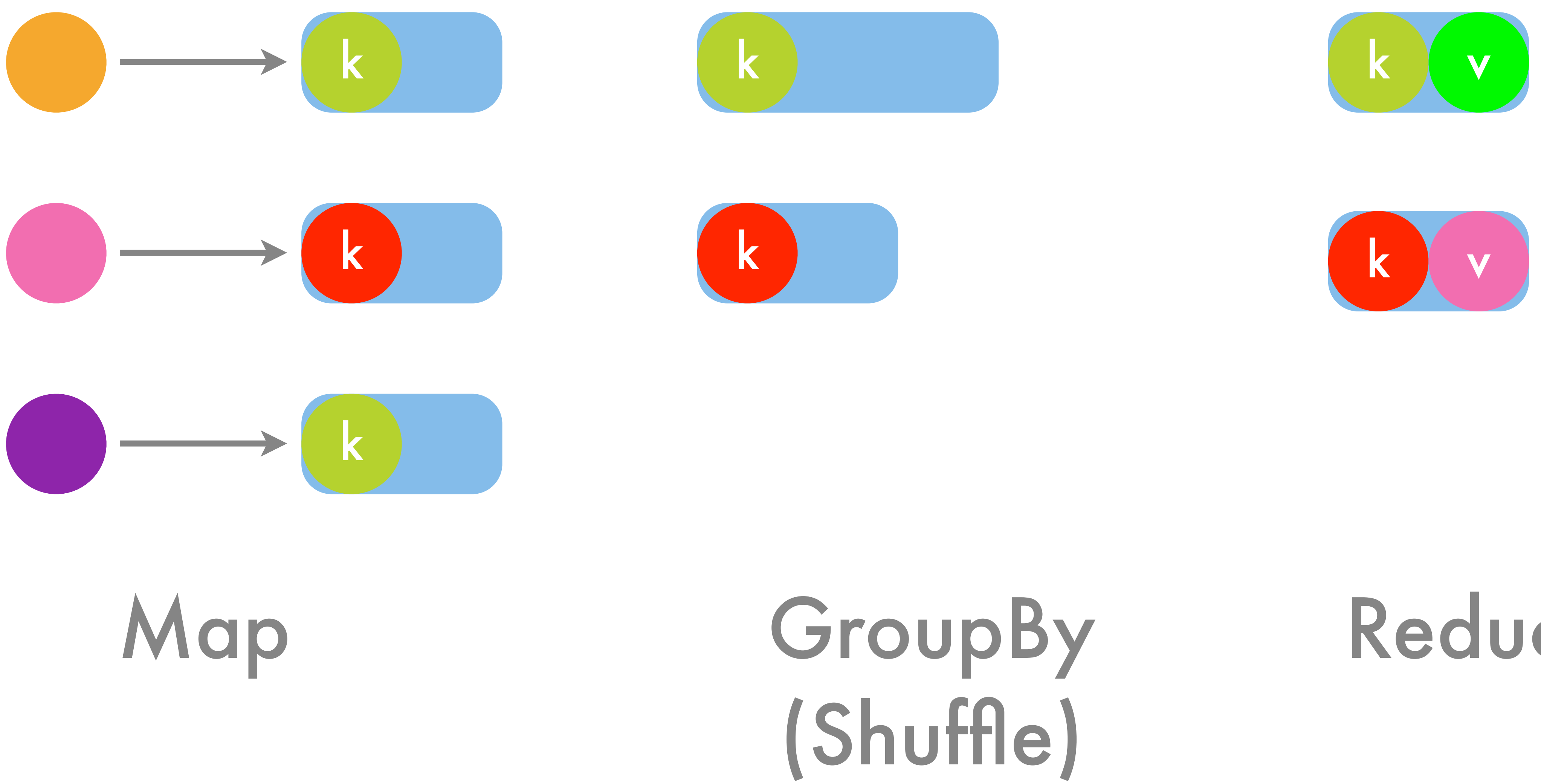
MapReduce



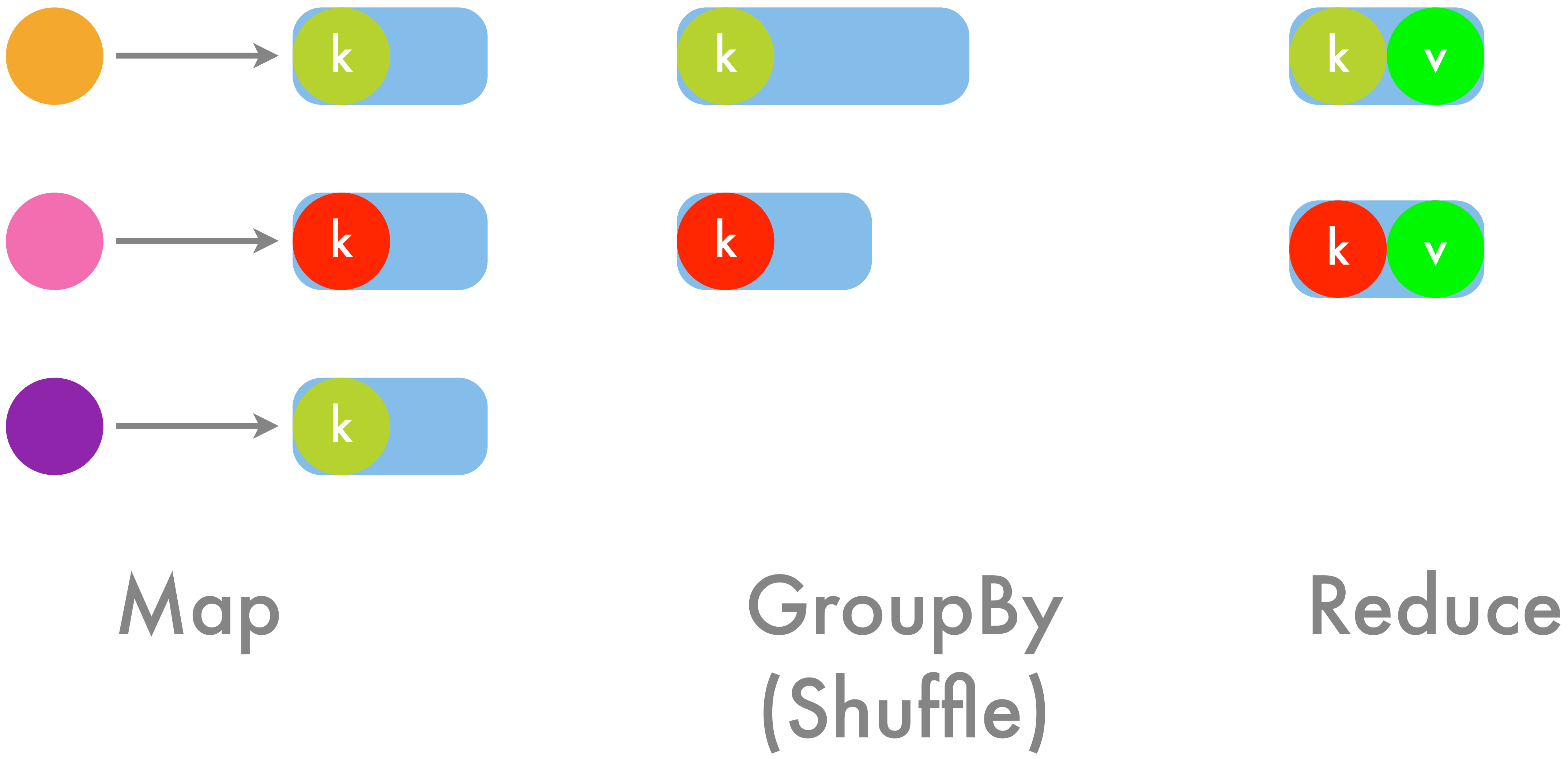
MapReduce



MapReduce



MapReduce



hadoop

- Open Source MapReduce Implementation:
  - **HDFS**: Distributed FileSystem
  - **YARN**: Resource Management
  - **MapReduce**: Programming Framework

http://hadoop.apache.org

New in Hadoop .23

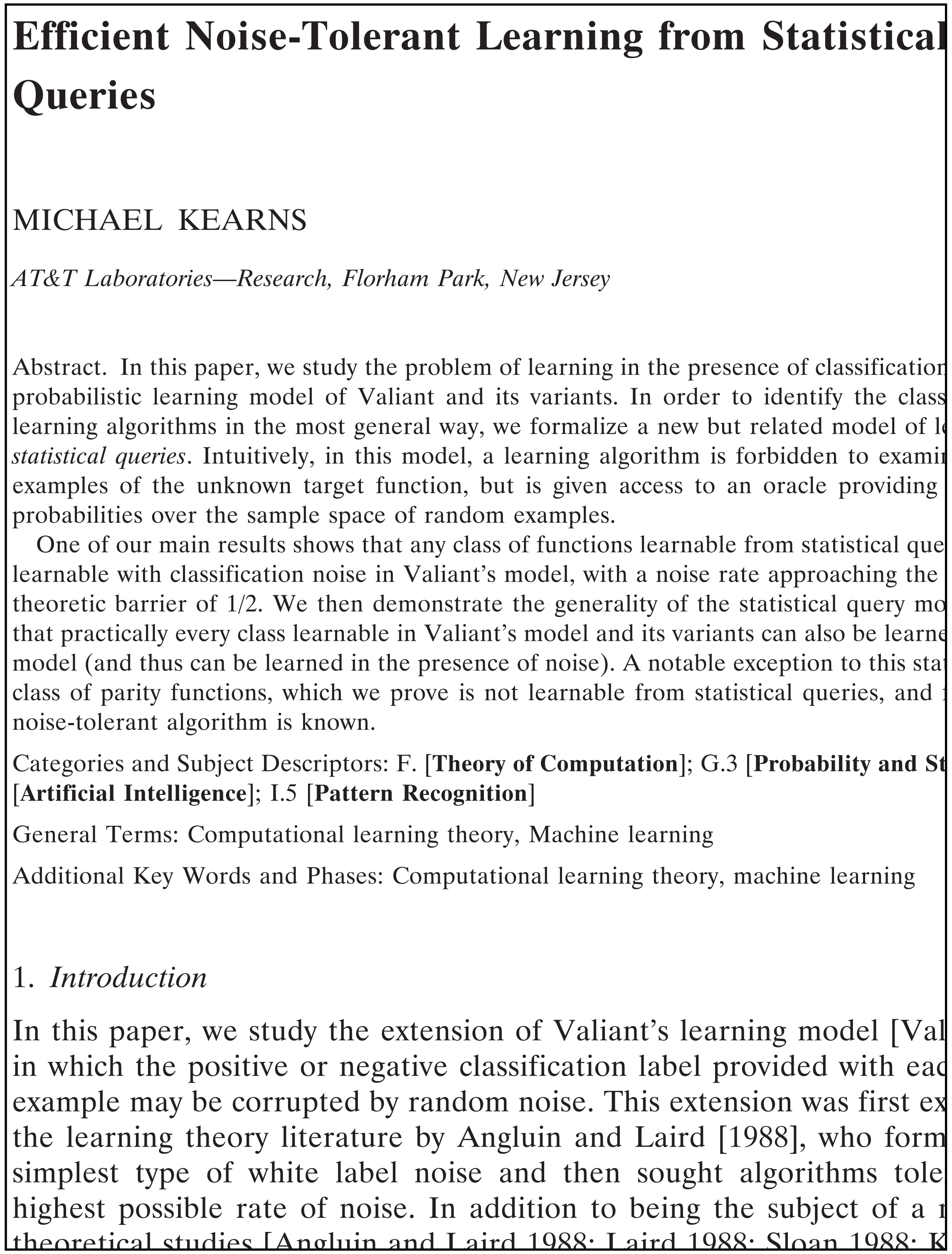
- Open Source MapReduce Implem
  - **HDFS**: Distributed FileSystem
  - **YARN**: Resource Management
  - **MapReduce**: Programming Framework

http://hadoop.apache.org

MapReduce for ML

- Learning algorithm can access the learning problem only through a statistical query oracle

- The statistical query oracle returns an estimate of the expectation of a function f(x,y) (averaged over the data distribution).



Efficient Noise-Tolerant Learning from Statistical Queries

MICHAEL KEARNS

AT&T Laboratories—Research, Florham Park, New Jersey

Abstract. In this paper, we study the problem of learning in the presence of classification
probabilistic learning model of Valiant and its variants. In order to identify the class
learning algorithms in the most general way, we formalize a new but related model of l
statistical queries. Intuitively, in this model, a learning algorithm is forbidden to exami
examples of the unknown target function, but is given access to an oracle providing
probabilities over the sample space of random examples.
One of our main results shows that any class of functions learnable from statistical que
learnable with classification noise in Valiant's model, with a noise rate approaching the
theoretic barrier of 1/2. We then demonstrate the generality of the statistical query mo
that practically every class learnable in Valiant's model and its variants can also be learne
model (and thus can be learned in the presence of noise). A notable exception to this sta
class of parity functions, which we prove is not learnable from statistical queries, and
noise-tolerant algorithm is known.

Categories and Subject Descriptors: F. [**Theory of Computation**]; G.3 [**Probability and St**
[**Artificial Intelligence**]; I.5 [**Pattern Recognition**]

General Terms: Computational learning theory, Machine learning

Additional Key Words and Phases: Computational learning theory, machine learning

1. *Introduction*

In this paper, we study the extension of Valiant's learning model [Val
in which the positive or negative classification label provided with eac
example may be corrupted by random noise. This extension was first ex
the learning theory literature by Angluin and Laird [1988], who form
simplest type of white label noise and then sought algorithms tole
highest possible rate of noise. In addition to being the subject of a
theoretical studies [Angluin and Laird 1988; Laird 1988; Sloan 1988; K

MapReduce for ML

- Rephrase oracle in summation form.
- **Map:** Calculate function estimates over sub-groups of data.
- **Reduce:** Aggregate the function estimates from various sub-groups.

Map-Reduce for Machine Learning on Multicore

Cheng-Tao Chu * chengtao@stanford.edu

Sang Kyun Kim * skkim38@stanford.edu

Yi-An Lin * ianl@stanford.edu

YuanYuan Yu * yuanyuan@stanford.edu

Gary Bradski *† garybradski@gmail

Andrew Y. Ng * ang@cs.stanford.edu

Kunle Olukotun * kunle@cs.stanford.edu

* CS. Department, Stanford University 353 Serra Mall,
Stanford University, Stanford CA 94305-9025.
† Rexee Inc.

Abstract

We are at the beginning of the multicore era. Computers will have increasingly many cores (processors), but there is still no good programming framework for these architectures, and thus no simple and unified way for machine learning to take advantage of the potential speed up. In this paper, we develop a broadly applicable parallel programming method, one that is easily applied to *many* different learning algorithms. Our work is in distinct contrast to the tradition in machine learning of designing (often ingenious) ways to speed up a *single* algorithm at a time. Specifically, we show that algorithms that fit the Statistical Query model [15] can be written in a certain "summation form," which allows them to be easily parallelized on multicore computers. We adapt Google's map-reduce [7] paradigm to demonstrate this parallel speed up technique on a variety of learning algorithms including locally weighted linear regression (LWLR), k-means, logistic regression (LR), naive Bayes (NB), SVM, ICA, PCA, gaussian discriminant analysis (GDA), EM, and backpropagation (NN). Our experimental results show basically linear speedup with an increasing number of processors.

1 Introduction

Frequency scaling on silicon—the ability to drive chips at ever higher clock rates—is beginning to hit a power limit as device geometries shrink due to leakage, and simply because CMOS consumes power every time it changes state [9, 10]. Yet Moore's law [20], the density of circuits doubling every generation, is projected to last between 10 and 20 more years for silicon based circuits [10].

Example: Gradient

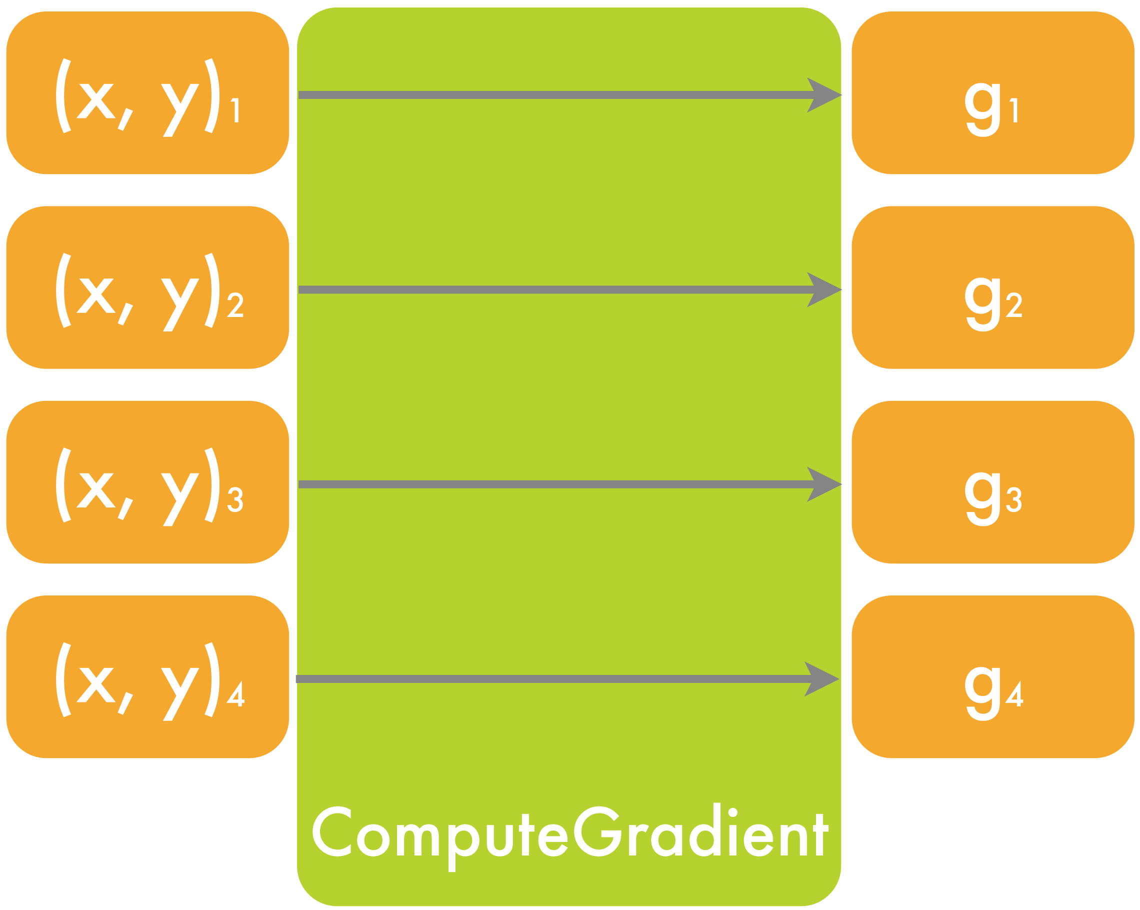
$(x, y)_1$

$(x, y)_2$

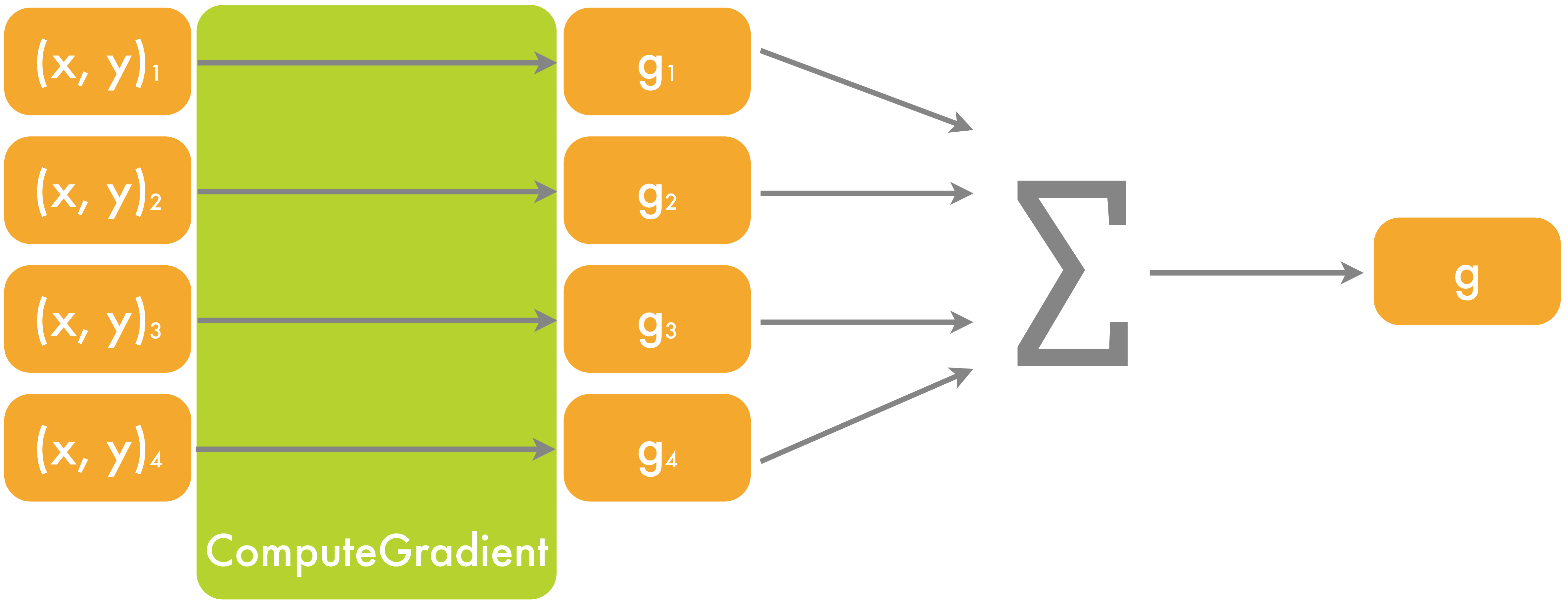
$(x, y)_3$

$(x, y)_4$

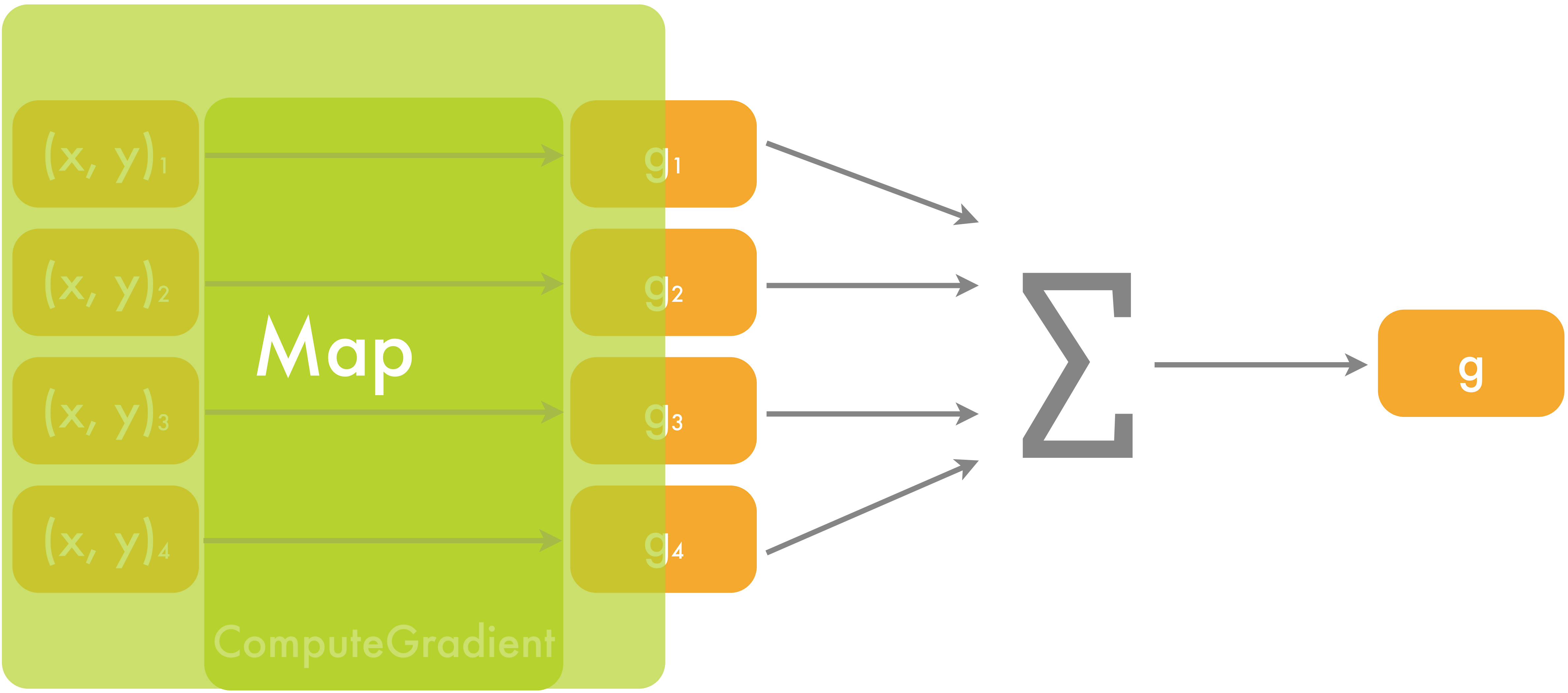
Example: Gradient



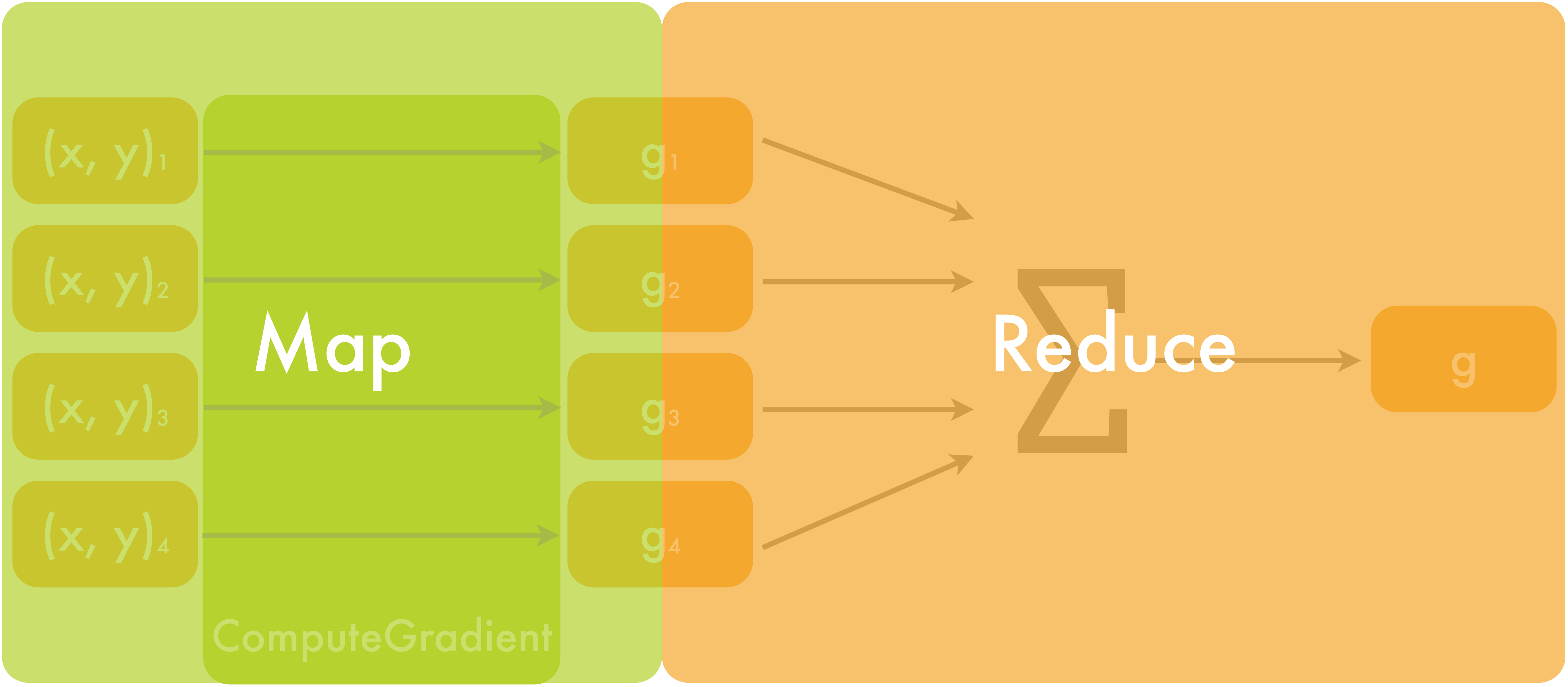
Example: Gradient



Example: Gradient



Example: Gradient



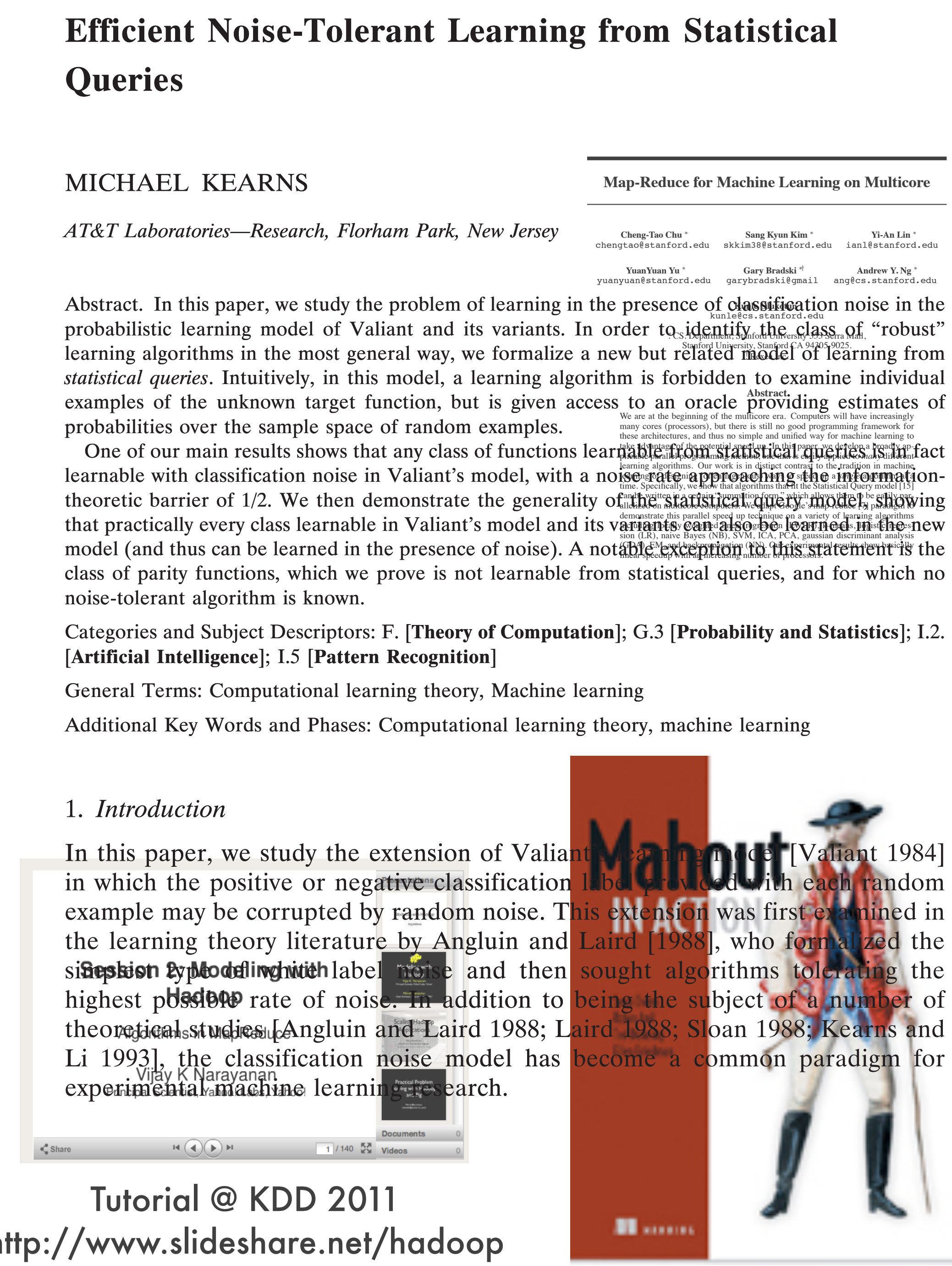
- Machine Learning Library
- Implementations of many algorithms, both on Hadoop MapReduce and stand-alone
- Open Source (Apache)
- Welcoming, helpful community

http://mahout.apache.org

mahout

- **Recommender Systems**, e.g.
  - User and Item based recommenders
  - Collaborative Filtering
- Clustering (K-Means, Mean Shift, ...)
- Topic Models (LDA)
- Supervised ML
  - (Logistic) Regression
  - Linear SVMs
  - Decision Trees and Forests

Further Reading



Efficient Noise-Tolerant Learning from Statistical Queries

MICHAEL KEARNS

AT&T Laboratories—Research, Florham Park, New Jersey

Abstract. In this paper, we study the problem of learning in the presence of classification noise in the probabilistic learning model of Valiant and its variants. In order to identify the class of "robust" learning algorithms in the most general way, we formalize a new but related model of learning from *statistical queries*. Intuitively, in this model, a learning algorithm is forbidden to examine individual examples of the unknown target function, but is given access to an oracle providing estimates of probabilities over the sample space of random examples.

One of our main results shows that any class of functions learnable from statistical queries is in fact learnable with classification noise in Valiant's model, with a noise rate approaching the information-theoretic barrier of 1/2. We then demonstrate the generality of the statistical query model, showing that practically every class learnable in Valiant's model and its variants can also be learned in the new model (and thus can be learned in the presence of noise). A notable exception to this statement is the class of parity functions, which we prove is not learnable from statistical queries, and for which no noise-tolerant algorithm is known.

Categories and Subject Descriptors: F. [**Theory of Computation**]; G.3 [**Probability and Statistics**]; I.2. [**Artificial Intelligence**]; I.5 [**Pattern Recognition**]

General Terms: Computational learning theory, Machine learning

Additional Key Words and Phases: Computational learning theory, machine learning

1. *Introduction*

In this paper, we study the extension of Valiant's learning model [Valiant 1984] in which the positive or negative classification label provided with each random example may be corrupted by random noise. This extension was first examined in the learning theory literature by Angluin and Laird [1988], who formalized the simplest type of white label noise and then sought algorithms tolerating the highest possible rate of noise. In addition to being the subject of a number of theoretical studies [Angluin and Laird 1988; Laird 1988; Sloan 1988; Kearns and Li 1993], the classification noise model has become a common paradigm for experimental machine learning research.

Map-Reduce for Machine Learning on Multicore

Cheng-Tao Chu chengtao@stanford.edu
Sang Kyun Kim skkim38@stanford.edu
Yi-An Lin ianl@stanford.edu
YuanYuan Yu yuanyuan@stanford.edu
Gary Bradski garybradski@gmail
Andrew Y. Ng ang@cs.stanford.edu

Abstract

Tutorial @ KDD 2011
http://www.slideshare.net/hadoop

Further Reading

Efficient Noise-Tolerant Learning from Statistical Queries

MICHAEL KEARNS

AT&T Laboratories—Research, Florham Park, New Jersey

Abstract. In this paper, we study the problem of learning in the presence of classification noise in the probabilistic learning model of Valiant and its variants. In order to identify the class of "robust" learning algorithms in the most general way, we formalize a new but related model of learning from *statistical queries*. Intuitively, in this model, a learning algorithm is forbidden to examine individual examples of the unknown target function, but is given access to an oracle providing estimates of probabilities over the sample space of random examples.

One of our main results shows that any class of functions learnable from statistical queries is in fact learnable with classification noise in Valiant's model, with a noise rate approaching the information-theoretic barrier of 1/2. We then demonstrate the generality of the statistical query model, showing that practically every class learnable in Valiant's model and its variants can also be learned in the new model (and thus can be learned in the presence of noise). A notable exception to this statement is the class of parity functions, which we prove is not learnable from statistical queries, and for which no noise-tolerant algorithm is known.

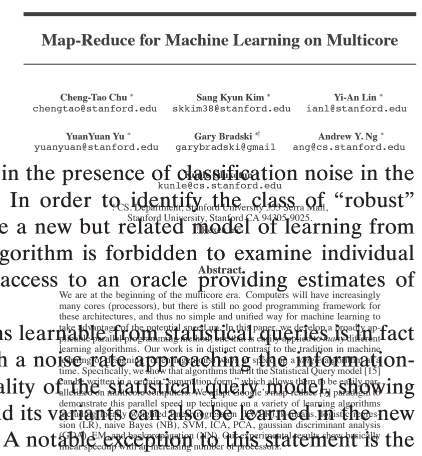
Categories and Subject Descriptors: F. [**Theory of Computation**]; G.3 [**Probability and Statistics**]; I.2. [**Artificial Intelligence**]; I.5 [**Pattern Recognition**]

General Terms: Computational learning theory, Machine learning

Additional Key Words and Phases: Computational learning theory, machine learning

1. *Introduction*

In this paper, we study the extension of Valiant's learning model [Valiant 1984] in which the positive or negative classification label provided with each random example may be corrupted by random noise. This extension was first examined in the learning theory literature by Angluin and Laird [1988], who formalized the simplest type of white label noise and then sought algorithms tolerating the highest possible rate of noise. In addition to being the subject of a number of theoretical studies [Angluin and Laird 1988; Laird 1988; Sloan 1988; Kearns and Li 1993], the classification noise model has become a common paradigm for experimental machine learning research.

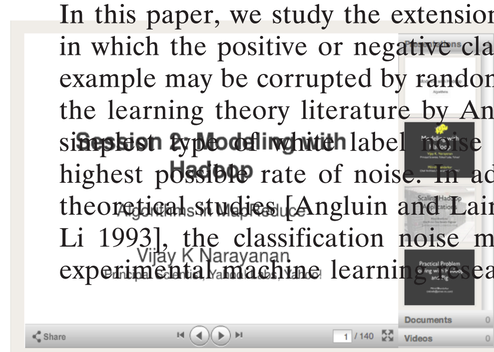


Map-Reduce for Machine Learning on Multicore

Cheng-Tao Chu chengtao@stanford.edu | Sang Kyun Kim skkim38@stanford.edu | Yi-An Lin ianl@stanford.edu

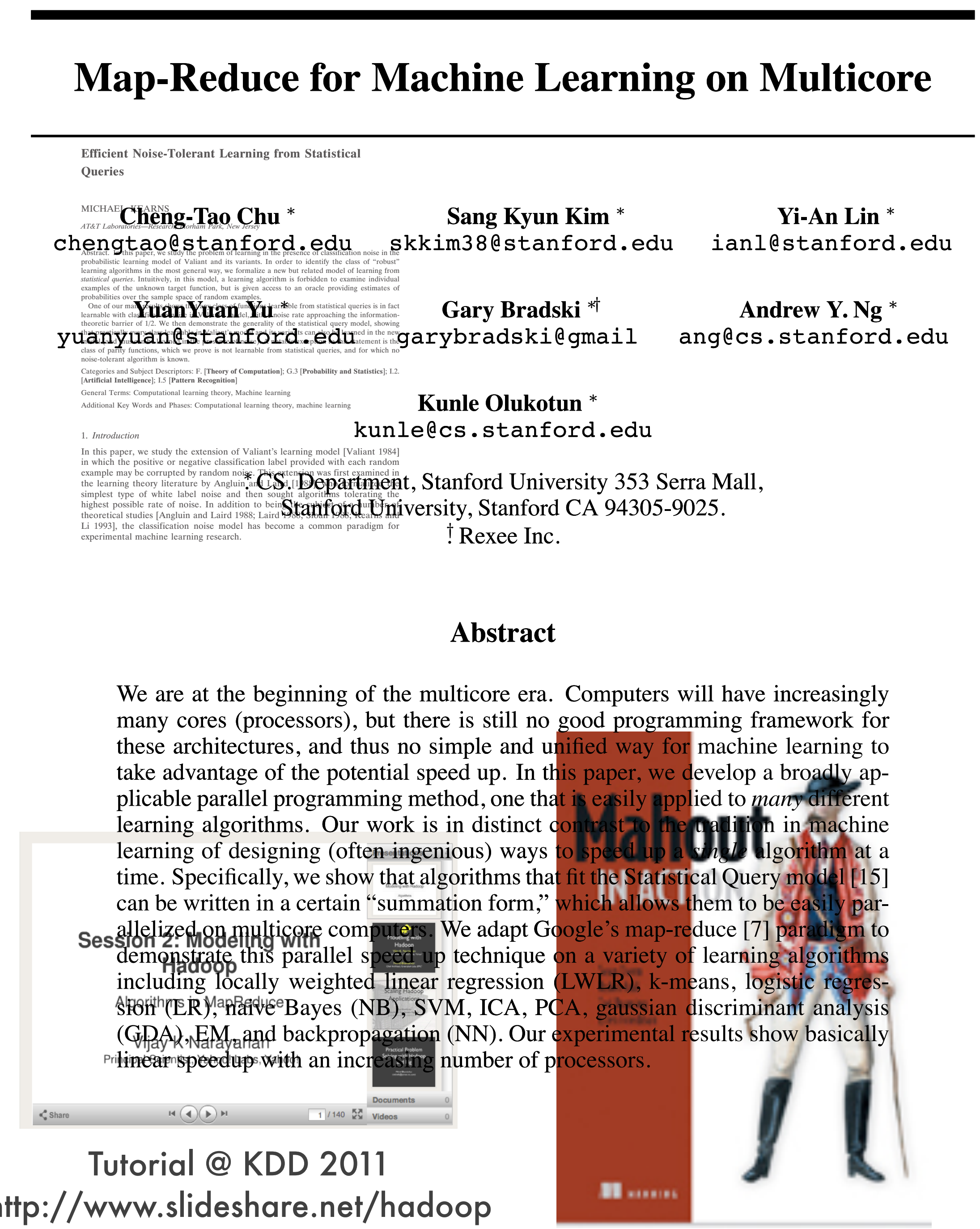
YuanYuan Yu yuanyuan@stanford.edu | Gary Bradski garybradski@gmail | Andrew Y. Ng ang@cs.stanford.edu

Abstract



Tutorial @ KDD 2011
http://www.slideshare.net/hadoop

Further Reading



Efficient Noise-Tolerant Learning from Statistical Queries

Map-Reduce for Machine Learning on Multicore

Cheng-Tao Chu * chengtao@stanford.edu

Sang Kyun Kim * skkim38@stanford.edu

Yi-An Lin * ianl@stanford.edu

YuanYuan Yu * yuanyuan@stanford.edu

Gary Bradski *† garybradski@gmail

Andrew Y. Ng * ang@cs.stanford.edu

Kunle Olukotun * kunle@cs.stanford.edu

* CS. Department, Stanford University 353 Serra Mall,
Stanford University, Stanford CA 94305-9025.
† Rexee Inc.

Abstract

We are at the beginning of the multicore era. Computers will have increasingly many cores (processors), but there is still no good programming framework for these architectures, and thus no simple and unified way for machine learning to take advantage of the potential speed up. In this paper, we develop a broadly applicable parallel programming method, one that is easily applied to *many* different learning algorithms. Our work is in distinct contrast to the tradition in machine learning of designing (often ingenious) ways to speed up a *single* algorithm at a time. Specifically, we show that algorithms that fit the Statistical Query model [15] can be written in a certain "summation form," which allows them to be easily parallelized on multicore computers. We adapt Google's map-reduce [7] paradigm to demonstrate this parallel speed up technique on a variety of learning algorithms including locally weighted linear regression (LWLR), k-means, logistic regression (LR), naive Bayes (NB), SVM, ICA, PCA, gaussian discriminant analysis (GDA), EM, and backpropagation (NN). Our experimental results show basically linear speedup with an increasing number of processors.

Tutorial @ KDD 2011
http://www.slideshare.net/hadoop

Further Reading

Efficient Noise-Tolerant Learning from Statistical Queries

Map-Reduce for Machine Learning on Multicore

Cheng-Tao Chu *
chengtao@stanford.edu

Sang Kyun Kim *
skkim38@stanford.edu

Yi-An Lin *
ianl@stanford.edu

YuanYuan Yu *
yuanyuan@stanford.edu

Gary Bradski *†
garybradski@gmail

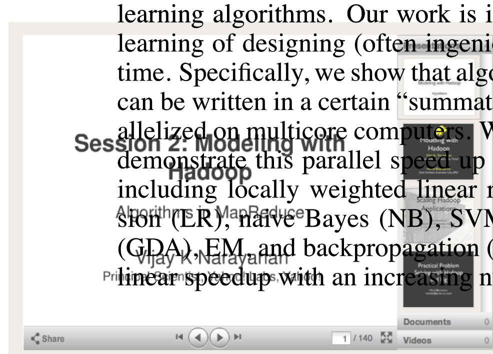
Andrew Y. Ng *
ang@cs.stanford.edu

Kunle Olukotun *
kunle@cs.stanford.edu

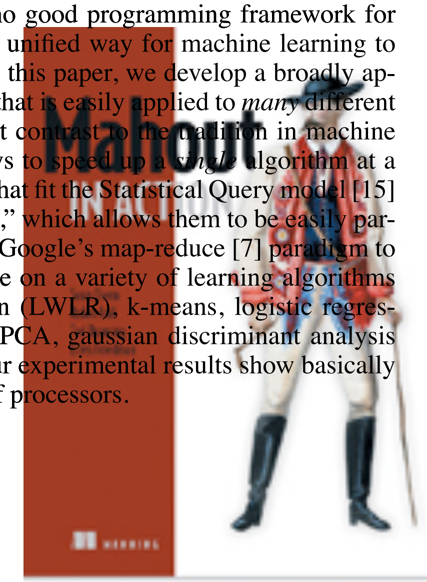
* CS. Department, Stanford University 353 Serra Mall,
Stanford University, Stanford CA 94305-9025.
† Rexee Inc.

Abstract

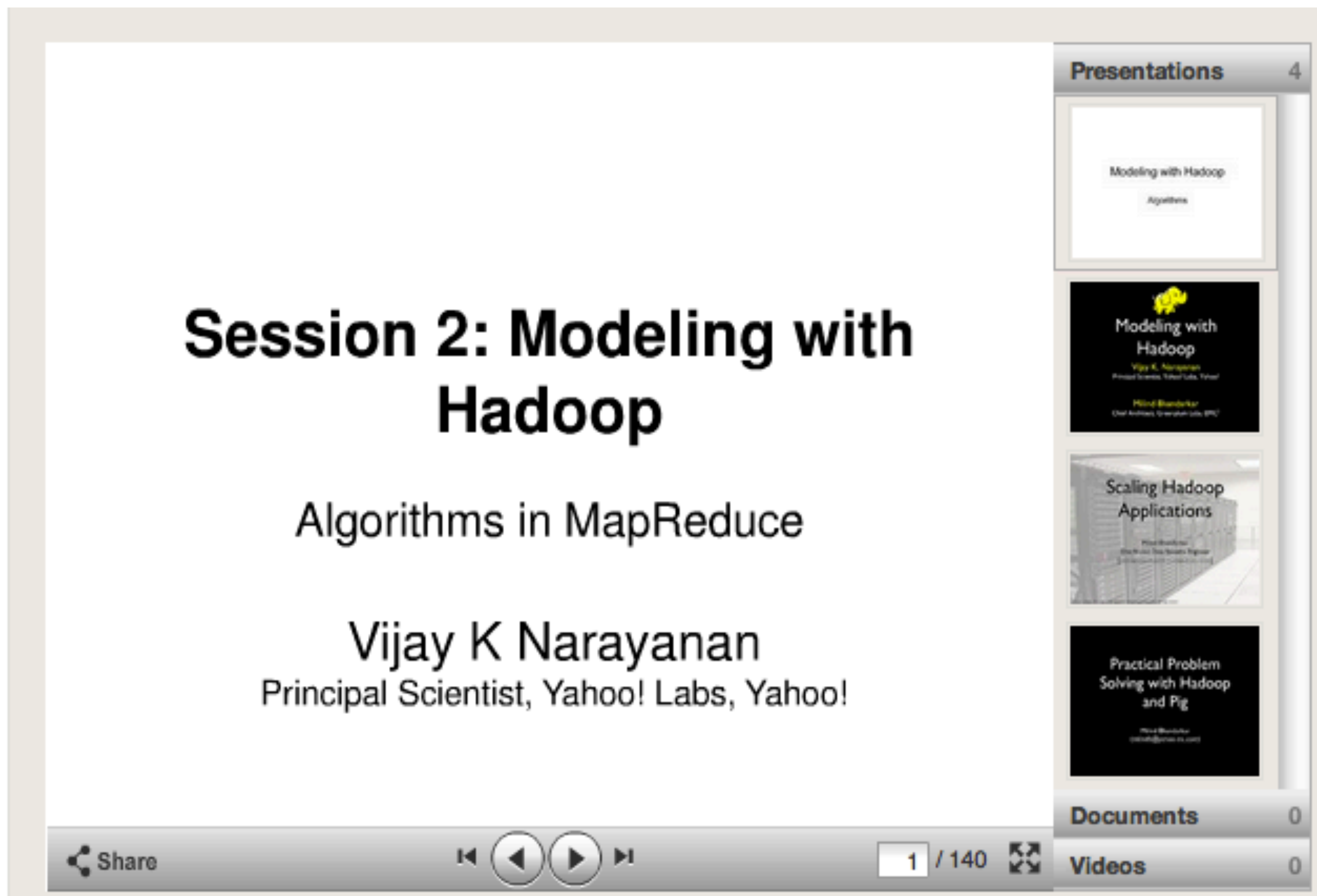
We are at the beginning of the multicore era. Computers will have increasingly many cores (processors), but there is still no good programming framework for these architectures, and thus no simple and unified way for machine learning to take advantage of the potential speed up. In this paper, we develop a broadly applicable parallel programming method, one that is easily applied to *many* different learning algorithms. Our work is in distinct contrast to the tradition in machine learning of designing (often ingenious) ways to speed up a *single* algorithm at a time. Specifically, we show that algorithms that fit the Statistical Query model [15] can be written in a certain "summation form," which allows them to be easily parallelized on multicore computers. We adapt Google's map-reduce [7] paradigm to demonstrate this parallel speed up technique on a variety of learning algorithms including locally weighted linear regression (LWLR), k-means, logistic regression (LR), naive Bayes (NB), SVM, ICA, PCA, gaussian discriminant analysis (GDA), EM, and backpropagation (NN). Our experimental results show basically linear speedup with an increasing number of processors.



Tutorial @ KDD 2011
http://www.slideshare.net/hadoop



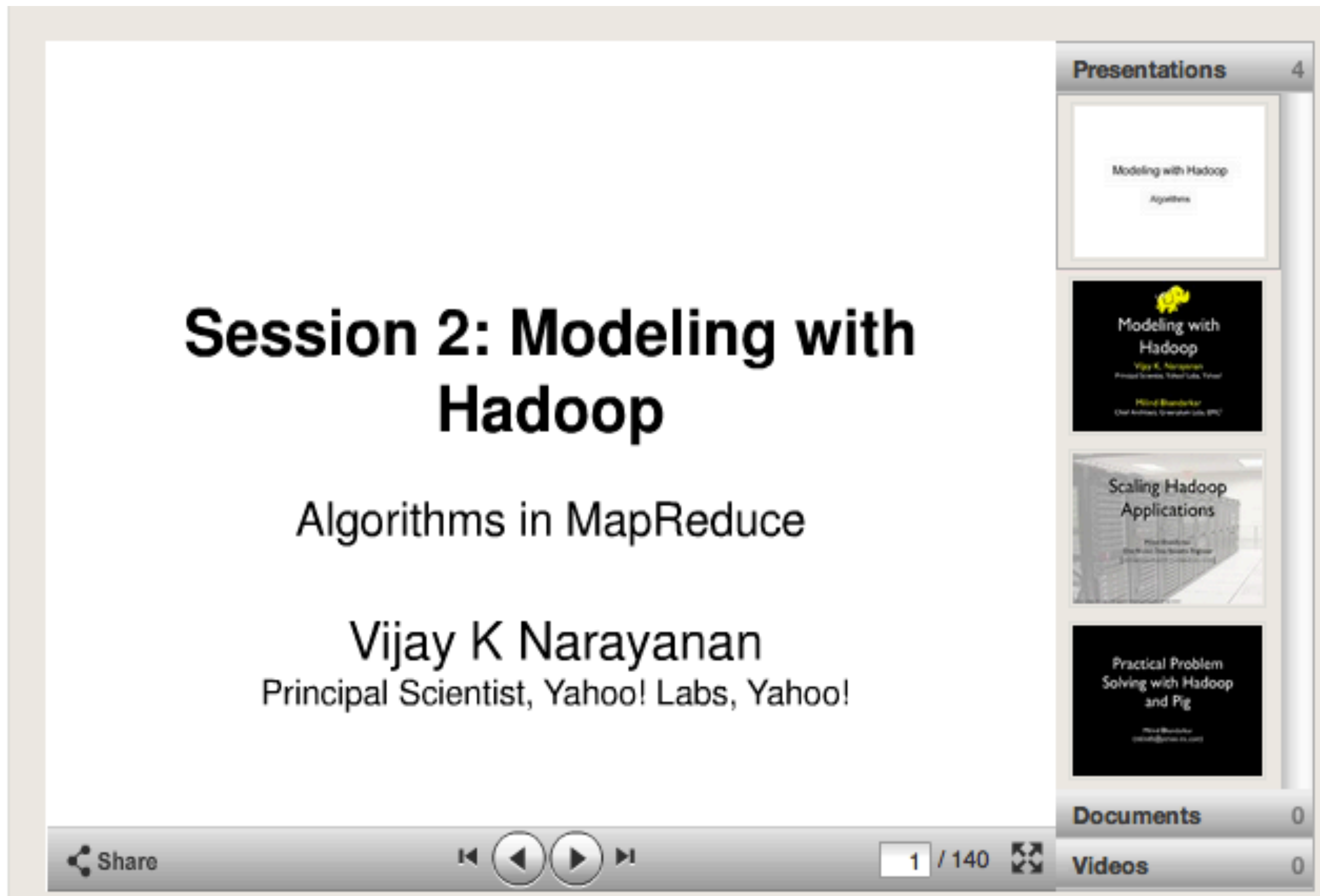
Further Reading



Tutorial @ KDD 2011

http://www.slideshare.net/hadoop

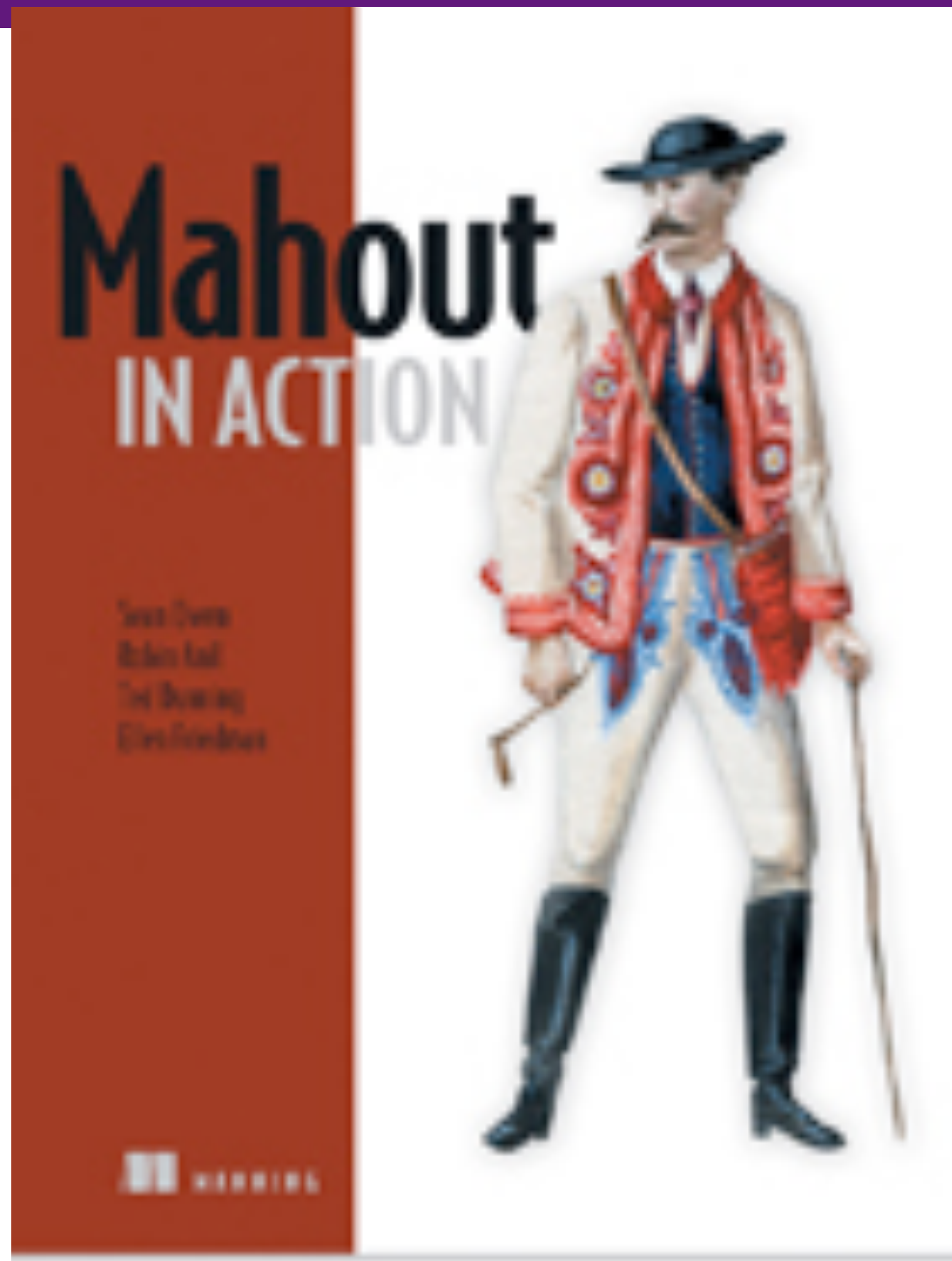
Further Reading



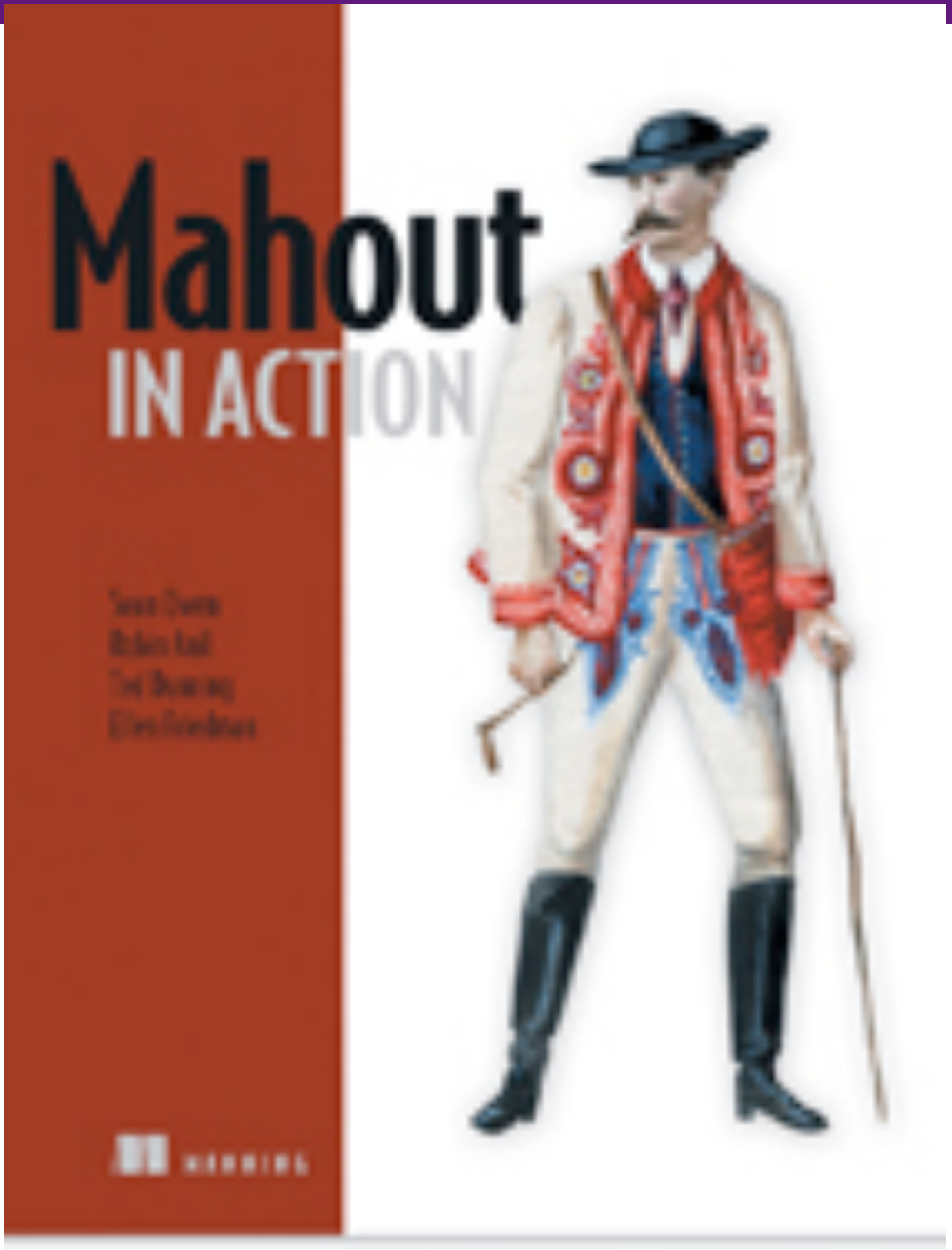
Tutorial @ KDD 2011

http://www.slideshare.net/hadoop

Further Reading



Further Reading



Further Reading

Efficient Noise-Tolerant Learning from Statistical Queries

MICHAEL KEARNS

AT&T Laboratories—Research, Florham Park, New Jersey

Abstract. In this paper, we study the problem of learning in the presence of classification noise in the probabilistic learning model of Valiant and its variants. In order to identify the class of "robust" learning algorithms in the most general way, we formalize a new but related model of learning from *statistical queries*. Intuitively, in this model, a learning algorithm is forbidden to examine individual examples of the unknown target function, but is given access to an oracle providing estimates of probabilities over the sample space of random examples.

One of our main results shows that any class of functions learnable from statistical queries is in fact learnable with classification noise in Valiant's model, with a noise rate approaching the information-theoretic barrier of 1/2. We then demonstrate the generality of the statistical query model, showing that practically every class learnable in Valiant's model and its variants can also be learned in the new model (and thus can be learned in the presence of noise). A notable exception to this statement is the class of parity functions, which we prove is not learnable from statistical queries, and for which no noise-tolerant algorithm is known.

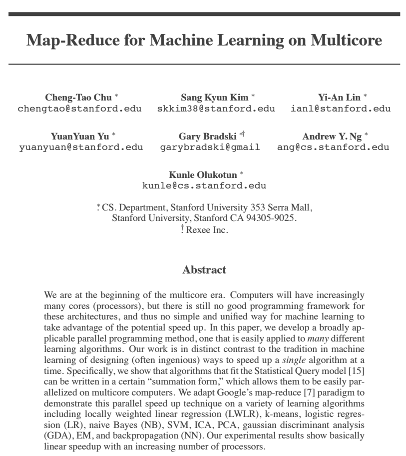
Categories and Subject Descriptors: F. [**Theory of Computation**]; G.3 [**Probability and Statistics**]; I.2. [**Artificial Intelligence**]; I.5 [**Pattern Recognition**]

General Terms: Computational learning theory, Machine learning

Additional Key Words and Phases: Computational learning theory, machine learning

1. *Introduction*

In this paper, we study the extension of Valiant's learning model [Valiant 1984] in which the positive or negative classification label provided with each random example may be corrupted by random noise. This extension was first examined in the learning theory literature by Angluin and Laird [1988], who formalized the simplest type of white label noise and then sought algorithms tolerating the highest possible rate of noise. In addition to being the subject of a number of theoretical studies [Angluin and Laird 1988; Laird 1988; Sloan 1988; Kearns and Li 1993], the classification noise model has become a common paradigm for experimental machine learning research.



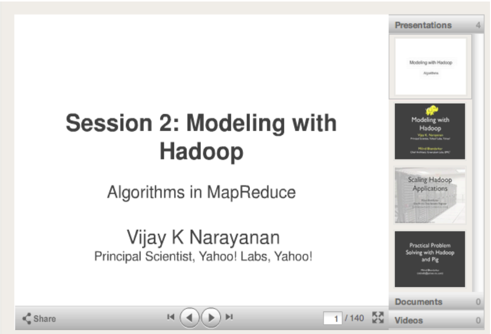
Map-Reduce for Machine Learning on Multicore

Cheng-Tao Chu * chengtao@stanford.edu
Sang Kyun Kim * skkim38@stanford.edu
Yi-An Lin * ianl@stanford.edu
YuanYuan Yu * yuanyuan@stanford.edu
Gary Bradski *† garybradski@gmail
Andrew Y. Ng * ang@cs.stanford.edu
Kunle Olukotun * kunle@cs.stanford.edu

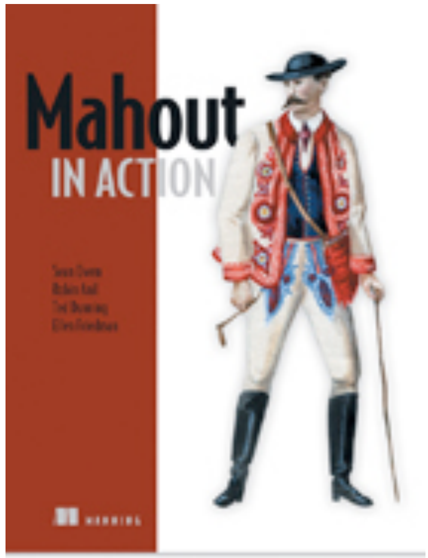
* CS. Department, Stanford University 353 Serra Mall,
Stanford University, Stanford CA 94305-9025.
† Rexee Inc.

Abstract

We are at the beginning of the multicore era. Computers will have increasingly many cores (processors), but there is still no good programming framework for these architectures, and thus no simple and unified way for machine learning to take advantage of the potential speed up. In this paper, we develop a broadly applicable parallel programming method, one that is easily applied to *many* different learning algorithms. Our work is in distinct contrast to the tradition in machine learning of designing (often ingenious) ways to speed up a *single* algorithm at a time. Specifically, we show that algorithms that fit the Statistical Query model [15] can be written in a certain "summation form," which allows them to be easily parallelized on multicore computers. We adapt Google's map-reduce [7] paradigm to demonstrate this parallel speed up technique on a variety of learning algorithms including locally weighted linear regression (LWLR), k-means, logistic regression (LR), naive Bayes (NB), SVM, ICA, PCA, gaussian discriminant analysis (GDA), EM, and backpropagation (NN). Our experimental results show basically linear speedup with an increasing number of processors.



Tutorial @ KDD 2011
http://www.slideshare.net/hadoop



Trouble

- ML is iterative
- Each iteration is a Job
- Overhead per job (>45s)
  - Scheduling
  - Program Distribution
  - Data Loading and Parsing
  - State Transfer

Map-Reduce for Machine Learning on Multicore

Cheng-Tao Chu *
chengtao@stanford.edu

Sang Kyun Kim *
skkim38@stanford.edu

Yi-An Lin *
ianl@stanford.edu

YuanYuan Yu *
yuanyuan@stanford.edu

Gary Bradski *†
garybradski@gmail

Andrew Y. Ng *
ang@cs.stanford.edu

Kunle Olukotun *
kunle@cs.stanford.edu

* CS. Department, Stanford University 353 Serra Mall,
Stanford University, Stanford CA 94305-9025.
† Rexee Inc.

Abstract

We are at the beginning of the multicore era. Computers will have increasingly many cores (processors), but there is still no good programming framework for these architectures, and thus no simple and unified way for machine learning to take advantage of the potential speed up. In this paper, we develop a broadly applicable parallel programming method, one that is easily applied to *many* different learning algorithms. Our work is in distinct contrast to the tradition in machine learning of designing (often ingenious) ways to speed up a *single* algorithm at a time. Specifically, we show that algorithms that fit the Statistical Query model [15] can be written in a certain "summation form," which allows them to be easily parallelized on multicore computers. We adapt Google's map-reduce [7] paradigm to demonstrate this parallel speed up technique on a variety of learning algorithms including locally weighted linear regression (LWLR), k-means, logistic regression (LR), naive Bayes (NB), SVM, ICA, PCA, gaussian discriminant analysis (GDA), EM, and backpropagation (NN). Our experimental results show basically linear speedup with an increasing number of processors.

1 Introduction

Frequency scaling on silicon—the ability to drive chips at ever higher clock rates—is beginning to hit a power limit as device geometries shrink due to leakage, and simply because CMOS consumes power every time it changes state [9, 10]. Yet Moore's law [20], the density of circuits doubling every generation, is projected to last between 10 and 20 more years for silicon based circuits [10].

Trouble

- ML is iterative
- Each iteration is a Job
- Overhead per job (>45s)
  - Scheduling
  - Program Distribution
  - Data Loading and Parsing
  - State Transfer

Map-Reduce for Machine Learning on Multicore

Cheng-Tao Chu * chengtao@stanford.edu
Sang Kyun Kim * skkim38@stanford.edu
Yi-An Lin * ianl@stanford.edu
YuanYuan Yu * yuanyuan@stanford.edu
Gary Bradski *† garybradski@gmail
Andrew Y. Ng * ang@cs.stanford.edu
Kunle Olukotun * kunle@cs.stanford.edu

* CS. Department, Stanford University 353 Serra Mall,
Stanford University, Stanford CA 94305-9025.
† Rexee Inc.

Abstract

We are at the beginning of the multicore era. Computers will have increasingly many cores (processors), but there is still no good programming framework for these architectures, and thus no simple and unified way for machine learning to take advantage of the potential speed up. In this paper, we develop a broadly applicable parallel programming method, one that is easily applied to *many* different learning algorithms. Our work is in distinct contrast to the tradition in machine learning of designing (often ingenious) ways to speed up a *single* algorithm at a time. Specifically, we show that algorithms that fit the Statistical Query model [15] can be written in a certain "summation form," which allows them to be easily parallelized on multicore computers. We adapt Google's map-reduce [7] paradigm to demonstrate this parallel speed up technique on a variety of learning algorithms including locally weighted linear regression (LWLR), k-means, logistic regression (LR), naive Bayes (NB), SVM, ICA, PCA, gaussian discriminant analysis (GDA), EM, and backpropagation (NN). Our experimental results show basically linear speedup with an increasing number of processors.

1 Introduction

Frequency scaling on silicon—the ability to drive chips at ever higher clock rates—is beginning to hit a power limit as device geometries shrink due to leakage, and simply because CMOS consumes power every time it changes state [9, 10]. Yet Moore's law [20], the density of circuits doubling every generation, is projected to last between 10 and 20 more years for silicon based circuits [10].


MAGIC EtchASketch® SCREEN
Beyond
MapReduce
today
OHIO ART

Solutions

- Local (subsampling)
- MPI
- Spark
- Pregel

Subsampling

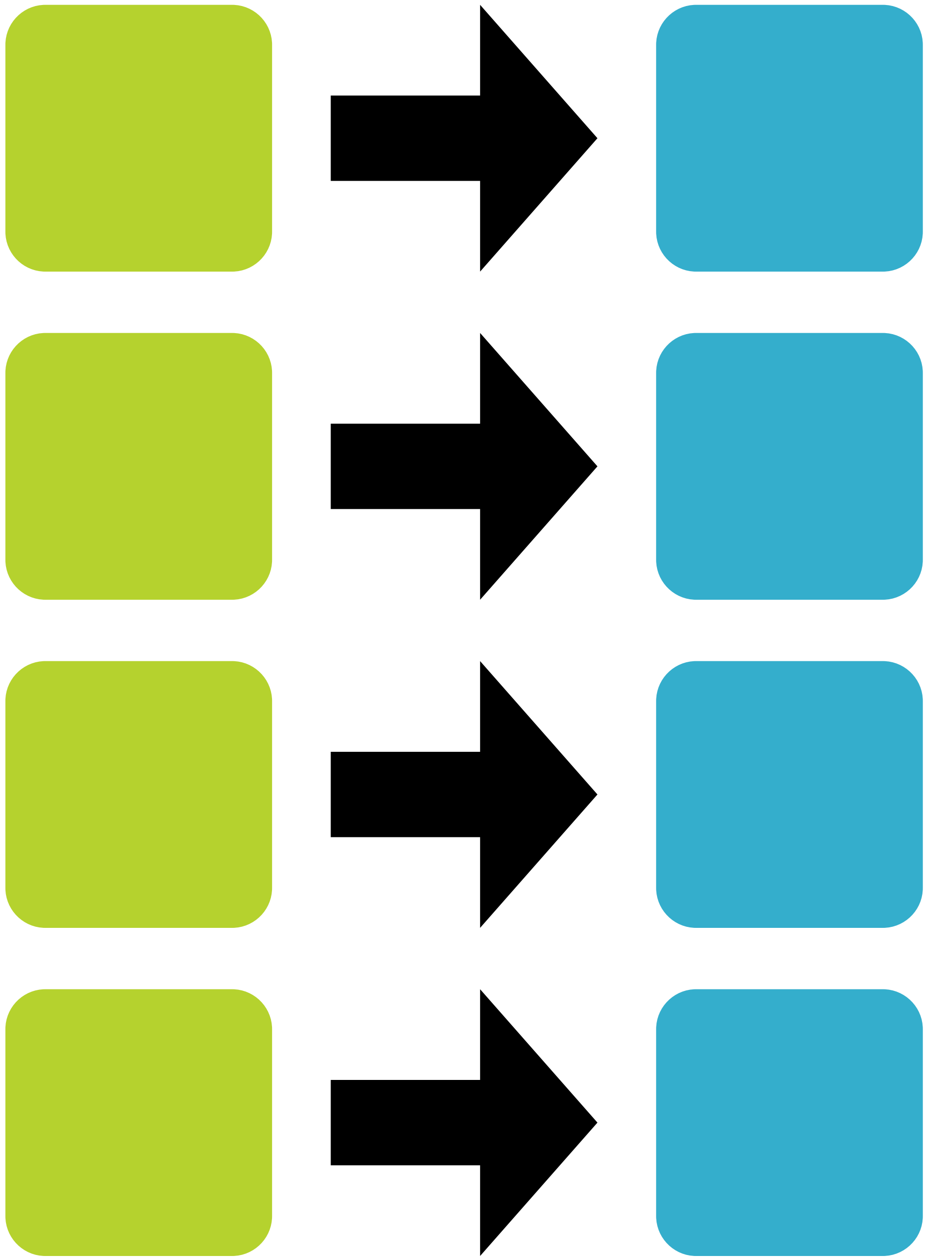
- Form examples on the cluster
- Subsample the data on the cluster
- Train a model on a single machine

Model Averaging

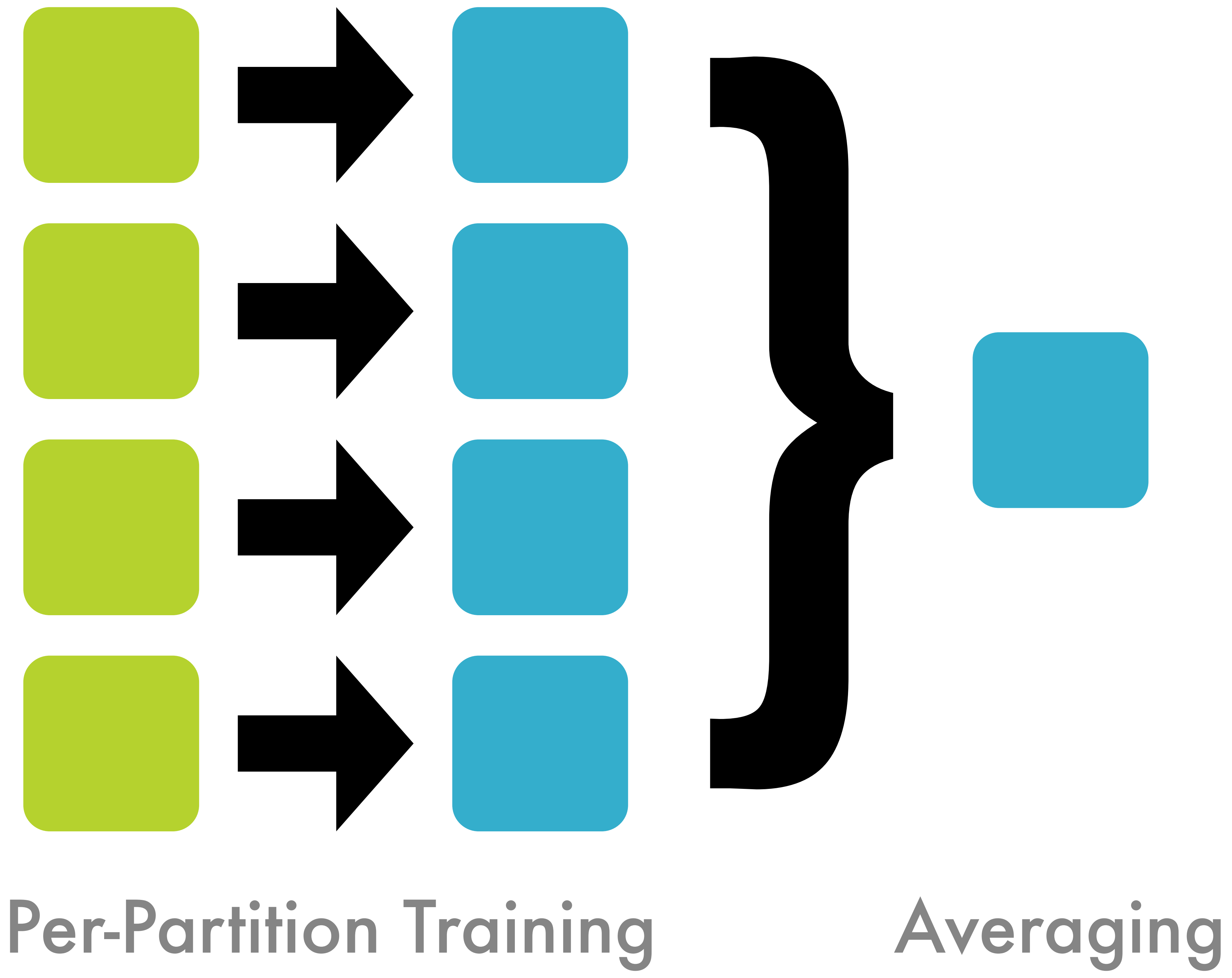
Per-Partition Training

Averaging

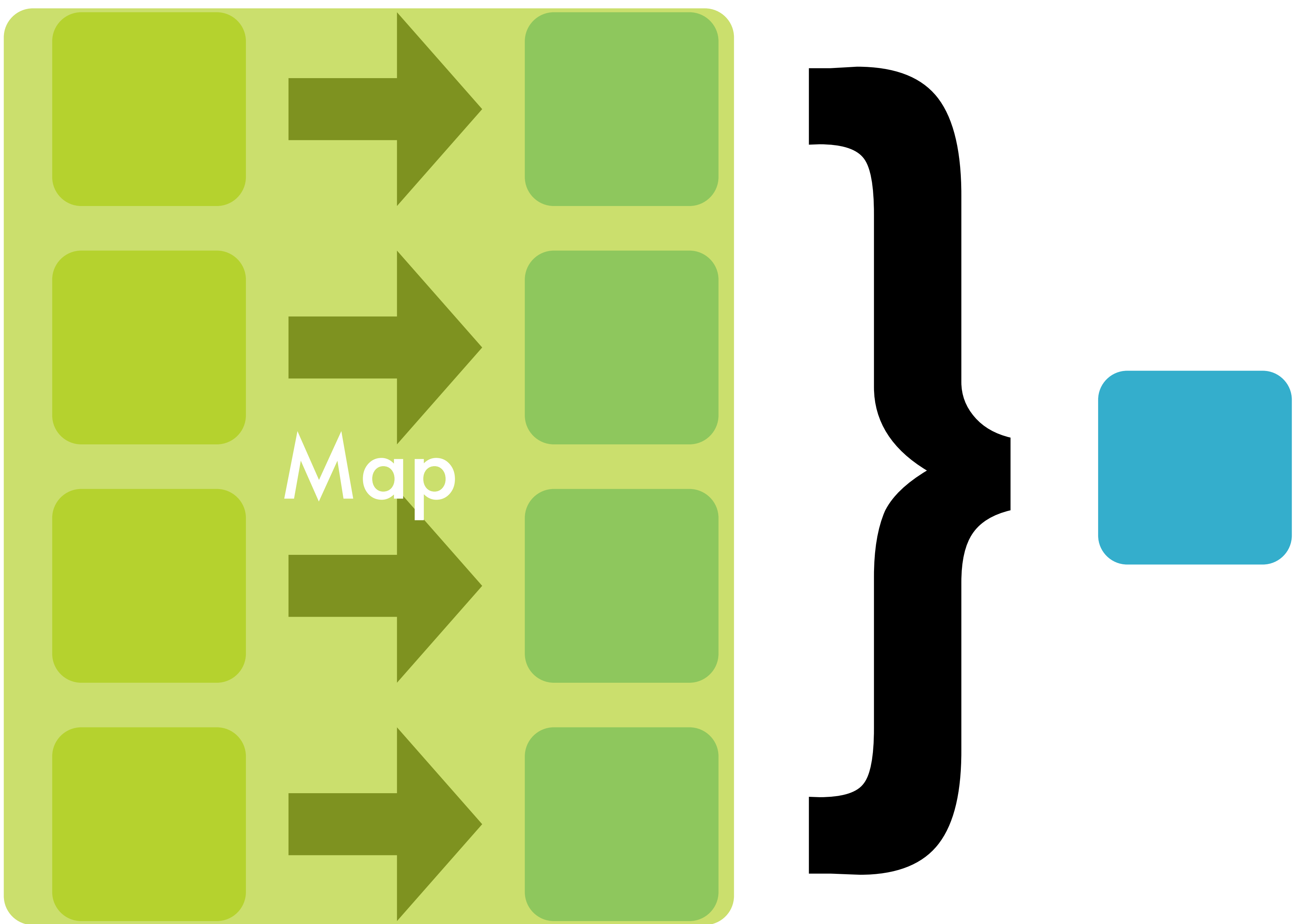
Model Averaging



Model Averaging



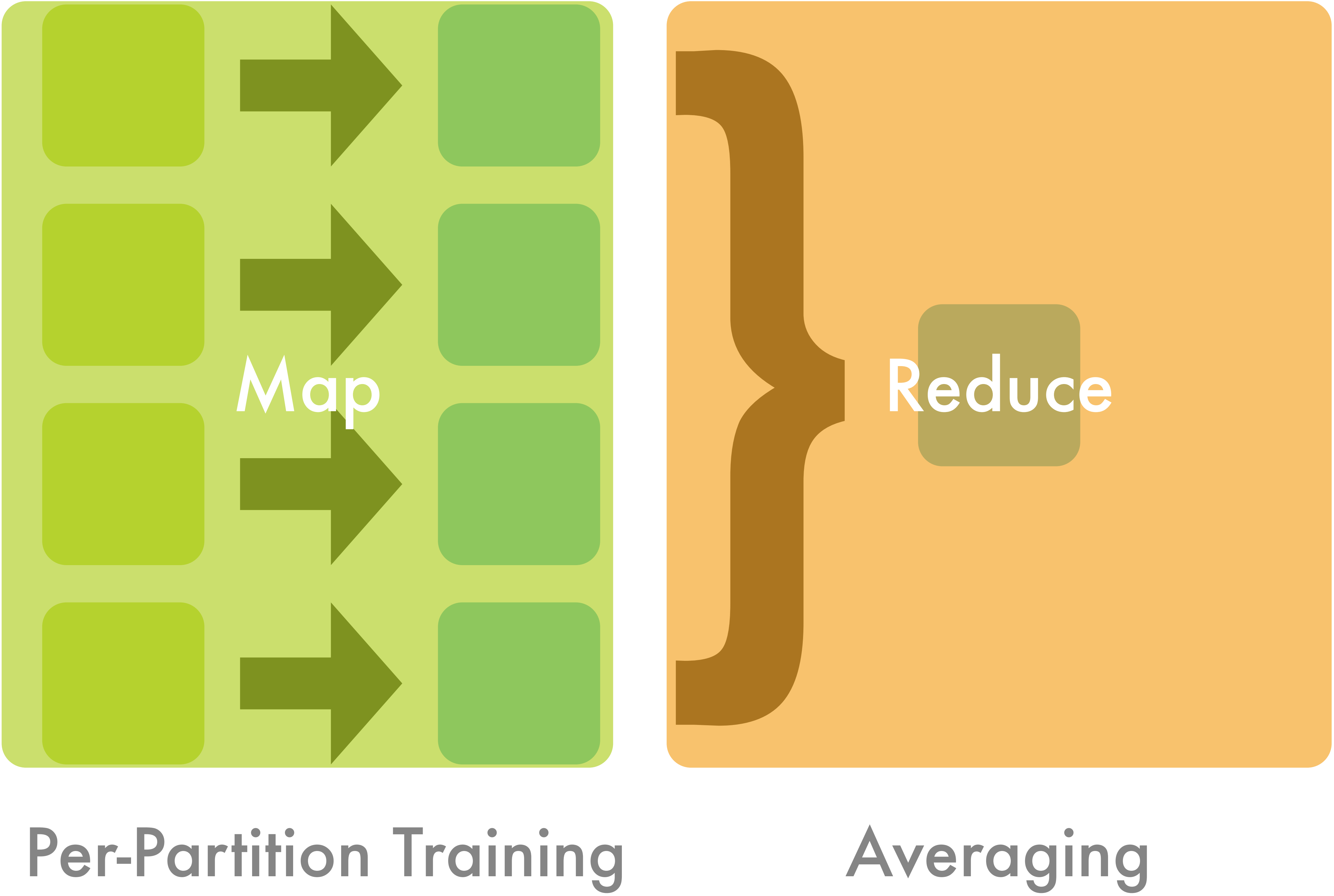
Model Averaging



Per-Partition Training

Averaging

Model Averaging



Message Passing Interface

- Mature HPC standard

- Supported on many clusters (e.g. OpenMPI)
- Available in C, Fortran and Scripting Languages

- Key operation here: AllReduce

AllReduce

AllReduce

... AllReduce()

... AllReduce()

... AllReduce()

AllReduce

… AllReduce()

… AllReduce()

… AllReduce()

AllReduce

... AllReduce()

... AllReduce()

... AllReduce()

AllReduce

... AllReduce()

... AllReduce()

... AllReduce()

... AllReduce()

... AllReduce()

... AllReduce()

AllReduce

... AllReduce()

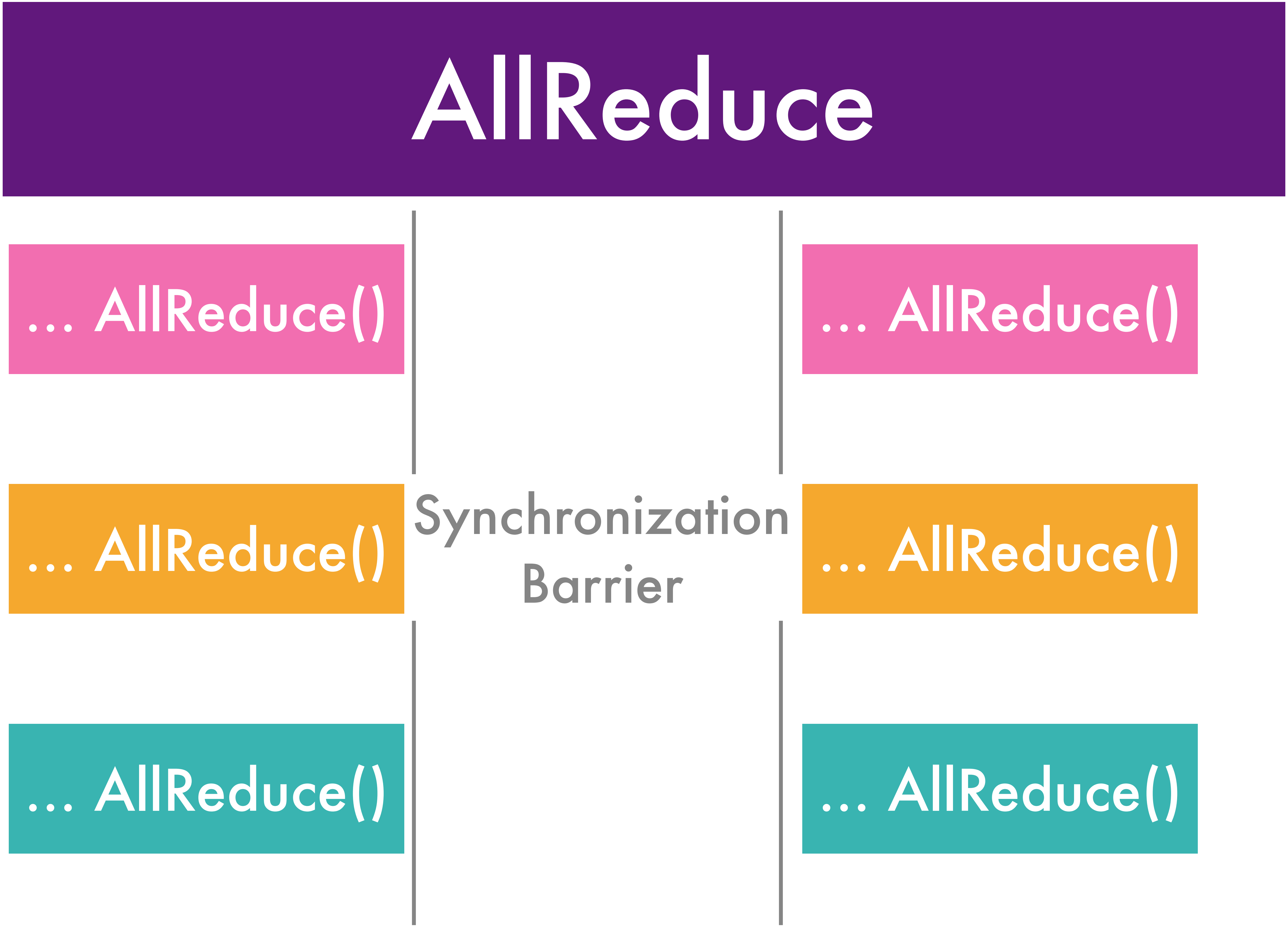
... AllReduce()

... AllReduce()

... AllReduce()

... AllReduce()

... AllReduce()


AllReduce
... AllReduce()
... AllReduce()
... AllReduce()
Synchronization
Barrier
... AllReduce()
... AllReduce()
... AllReduce()

AllReduce

... AllReduce()

... AllReduce()

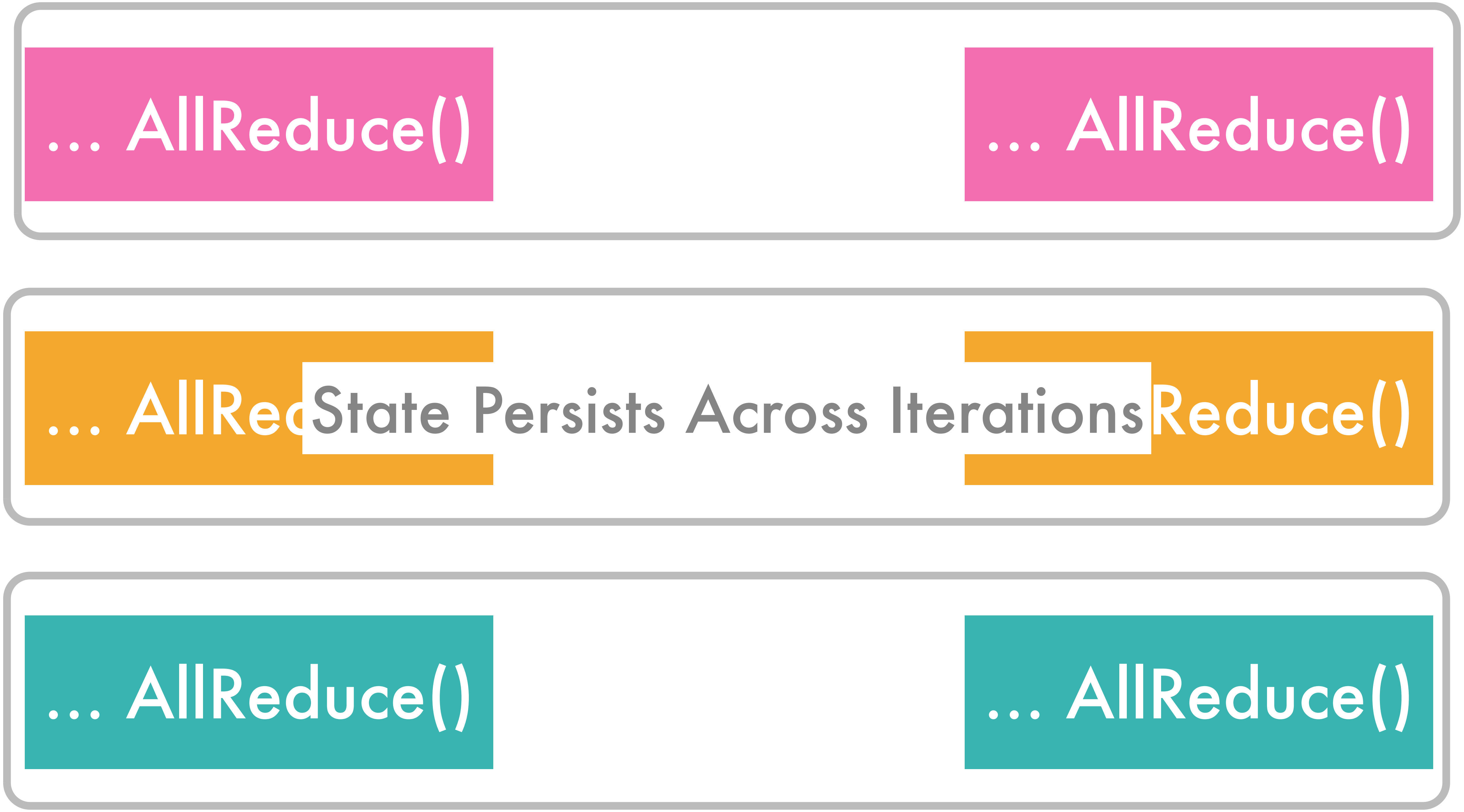
... AllReduce()

... AllReduce()

... AllReduce()

... AllReduce()

AllReduce



... AllReduce()
... AllReduce()
State Persists Across Iterations
Reduce()
... AllReduce()
... AllReduce()

Hadoop AllReduce

- Use Hadoop for
  - Data local scheduling
  - Good machine identification
- Use MPI for
  - AllReduce
- 30x Speedup over Hadoop MapReduce

A Reliable Effective Terascale Linear Learning System

Alekh Agarwal
Department of EECS
UC Berkeley
alekh@cs.berkeley.edu

Olivier Chapelle
Yahoo! Research
Santa Clara, CA
chap@yahoo-inc.com

Miroslav Dudík
Yahoo! Research
New York, NY
mdudik@yahoo-inc.com

John Langford
Yahoo! Research
New York, NY
jl@yahoo-inc.com

ABSTRACT

We present a system and a set of techniques for learning linear predictors with convex losses on terascale datasets, with trillions of features,[1] billions of training examples and millions of parameters in an hour using a cluster of 1000 machines. Individually none of the component techniques is new, but the careful synthesis required to obtain an efficient implementation is a novel contribution. The result is, up to our knowledge, the most scalable and efficient linear learning system reported in the literature. We describe and thoroughly evaluate the components of the system, showing the importance of the various design choices.

1. INTRODUCTION

Distributed machine learning is a research area that has seen a growing body of literature in recent years. Much work focuses on problems of the form

$$\min_{\mathbf{w}\in\mathbb{R}^d} \sum_{i=1}^{n} \ell(\mathbf{w}^\top \mathbf{x}_i; y_i) + \lambda R(\mathbf{w}), \qquad (1)$$

where $\mathbf{x}_i$ is the feature vector of the i-th example, y_i is the label, $\mathbf{w}$ is the linear predictor, ℓ is a loss function and R a regularizer. Much of this work exploits the natural decomposability over examples in (1), partitioning the examples over different nodes in a distributed environment such as a cluster.

Perhaps the simplest learning strategy when the number of samples n is very large is to subsample a smaller set of examples that can be tractably learned with. However, this strategy only works if the problem is simple enough or the number of parameters is very small. The setting of interest here is when a large number of samples is really needed to learn a good model, and distributed algorithms are a natural choice for such scenarios.

Some prior works (McDonald et al., 2010; Zinkevich e 2010) consider online learning with averaging and I et al. (2010a) propose gossip-style message passing rithms extending the existing literature on distributed vex optimization (Bertsekas and Tsitsiklis, 1989). Lan et al. (2009) analyze a delayed version of distributed o learning. Dekel et al. (2010) consider mini-batch versic online algorithms which are extended to delay-based up in Agarwal and Duchi (2011). A recent article of Boyd (2011) describes an application of the ADMM techniqu distributed learning problems. GraphLab (Low et al., is a parallel computation framework on graphs. More cl related to our work is that of Teo et al. (2007) who use to parallelize a bundle method for optimization.

However, all of the aforementioned approaches see leave something to be desired empirically when deploy large clusters. In particular their throughput—measur the input size divided by the wall clock running tim smaller than the the I/O interface of a single machin almost all parallel learning algorithms (Bekkerman e 2011, Part III, page 8). The I/O interface is an upper b on the speed of the fastest sequential algorithm sinc sequential algorithms are limited by the network inte in acquiring data. In contrast, we were able to achi throughput of 500M features/s, which is about a facto faster than the 1Gb/s network interface of any one no

An additional benefit of our system is its compati with MapReduce clusters such as Hadoop (unlike MPI-l systems) and minimal additional programming effort to allelize existing learning algorithms (unlike MapReduc proaches).

One of the key components in our system is a comn cation infrastructure that efficiently accumulates and b casts values across all nodes of a computation. It is func ally similar to MPI AllReduce (hence we use the name) it takes advantage of and is compatible with Hadoop so : restar ution sp

[1]The number of zero entries in th

http://hunch.net/~vw

MPI: Conclusion

- **The Good**
  - Computational Performance
  - Well established software available
- **The Bad**
  - No fault tolerance
- **The Ugly**
  - Ignorance of shared clusters
  - Systems-Level decisions at the algorithm layer

Spark: Intro

- Open Source cluster computation framework
- Developed at UC Berkeley by the AMP Lab
- Aimed at interactive and iterative use cases
- 30x faster than Hadoop for those
- User interface: Embedded Domain Specific Language in Scala

http://spark-project.org/

Spark: Example

```
val points = spark.textFile(...).
                   map(parsePoint).
                   partitionBy(HashPartitioner(NODES)).
                   cache()

var w = Vector.random(D)

for (i <- 1 to ITERATIONS) {
  val gradient = points.map(computeGradient(_,w)).reduce(_ + _)

  w -= gradient
}
```

Spark: Example

```
val points = spark.textFile(...).
                  map(parsePoint).
                  partitionBy(HashPartitioner(NODES)).
                  cache()

var w = Vector.random(D)

for (i <- 1 to ITERATIONS) {
  val gradient = points.map(computeGradient(_,w)).reduce(_ + _)

  w -= gradient
}
```

Spark: Example

Loads data into (distributed) main memory

```
val points = spark.textFile(...).
                  map(parsePoint).
                  partitionBy(HashPartitioner(NODES)).
                  cache()

var w = Vector.random(D)

for (i <- 1 to ITERATIONS) {
  val gradient = points.map(computeGradient(_,w)).reduce(_ + _)

  w -= gradient
}
```

Spark: Example

```
val points = spark.textFile(...).
                   map(parsePoint).
                   partitionBy(HashPartitioner(NODES)).
                   cache()

var w = Vector.random(D)

for (i <- 1 to ITERATIONS) {
  val gradient = points.map(computeGradient(_,w)).reduce(_ + _)

  w -= gradient
}
```

Spark: Example

```
val points = spark.textFile(...).
                   map(parsePoint).
                   partitionBy(HashPartitioner(NODES)).
                   cache()

var w = Vector.random(D)

for (i <- 1 to ITERATIONS) {
  val gradient = points.map(computeGradient(_,w)).reduce(_ + _)

  w -= gradient
}
```

Spark: Example

```
val points = spark.textFile(...).
                  map(parsePoint).
                  partitionBy(HashPartitioner(NODES)).
                  cache()

var w = Vector.random(D)

for (i <- 1 to ITERATIONS) {
  val gradient = points.map(computeGradient(_,w)).reduce(_ + _)

  w -= gradient
}
```

Computes a gradient per data point

Spark: Example

```
val points = spark.textFile(...).
                   map(parsePoint).
                   partitionBy(HashPartitioner(NODES)).
                   cache()

var w = Vector.random(D)

for (i <- 1 to ITERATIONS) {
  val gradient = points.map(computeGradient(_,w)).reduce(_ + _)

  w -= gradient
}
```

Computes a gradient per data point

Sums them up

Spark: Example

```
val points = spark.textFile(...).
                   map(parsePoint).
                   partitionBy(HashPartitioner(NODES)).
                   cache()

var w = Vector.random(D)

for (i <- 1 to ITERATIONS) {
  val gradient = points.map(computeGradient(_,w)).reduce(_ + _)

  w -= gradient
}
```

Spark: Example

```
val points = spark.textFile(...).
                  map(parsePoint).
                  partitionBy(HashPartitioner(NODES)).
                  cache()

var w = Vector.random(D)

for (i <- 1 to ITERATIONS) {
  val gradient = points.map(computeGradient(_,w)).reduce(_ + _)

  w -= gradient
}
```

Spark: Example

```
val points = spark.textFile(...).
                   map(parsePoint).
                   partitionBy(HashPartitioner(NODES)).
                   cache()

var w = Vector.random(D)

for (i <- 1 to ITERATIONS) {
  val gradient = points.map(computeGradient(_,w)).reduce(_ + _)

  w -= gradient
}
```

Trouble!
(physical layer shows through)

Spark: Conclusion

- **The Good**
  - Speed (ca. MPI speed)
  - Fault Tolerance
  - Ease of Programming
- **The Bad**
  - Main Memory Assumption
- **The Ugly**
  - Systems aspects creep up

Pregel

- Graph Computation framework
- Developed by Google
- Per vertex function `update()` processes incoming messages and sends new ones
- Computation is Bulk Synchronous Parallel

Pregel: A System for Large-Scale Graph Processing

Grzegorz Malewicz, Matthew H. Austern, Aart J. C. Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski
Google, Inc.
{malewicz,austern,ajcbik,dehnert,ilan,naty,gczaj}@google.com

TRACT

practical computing problems concern large graphs. ard examples include the Web graph and various so- tworks. The scale of these graphs—in some cases bil- f vertices, trillions of edges—poses challenges to their t processing. In this paper we present a computa- model suitable for this task. Programs are expressed equence of iterations, in each of which a vertex can messages sent in the previous iteration, send mes- o other vertices, and modify its own state and that of going edges or mutate graph topology. This vertex- approach is flexible enough to express a broad set of hms. The model has been designed for efficient, scal- nd fault-tolerant implementation on clusters of thou- of commodity computers, and its implied synchronic- kes reasoning about programs easier. Distribution- l details are hidden behind an abstract API. The result mework for processing large graphs that is expressive sy to program.

gories and Subject Descriptors

Programming Techniques]: Concurrent Program- -*Distributed programming*; D.2.13 [**Software Engi- ng**]: Reusable Software—*Reusable libraries*

ral Terms

, Algorithms

vords

outed computing, graph algorithms

NTRODUCTION

Internet made the Web graph a popular object of is and research. Web 2.0 fueled interest in social net- Other large graphs—for example induced by trans- ion routes, similarity of newspaper articles, paths of disease outbreaks, or citation relationships among pub scientific work—have been processed for decades. Frequ applied algorithms include shortest paths computation ferent flavors of clustering, and variations on the page theme. There are many other graph computing pro of practical value, *e.g.*, minimum cut and connected co nents.

Efficient processing of large graphs is challenging. algorithms often exhibit poor locality of memory access little work per vertex, and a changing degree of paral over the course of execution [31, 39]. Distribution over machines exacerbates the locality issue, and increase probability that a machine will fail during computation spite the ubiquity of large graphs and their commercia portance, we know of no scalable general-purpose sy for implementing arbitrary graph algorithms over arb graph representations in a large-scale distributed en ment.

Implementing an algorithm to process a large grap ically means choosing among the following options:

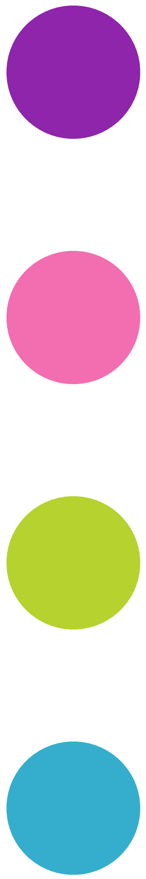
1. Crafting a custom distributed infrastructure, typ requiring a substantial implementation effort that be repeated for each new algorithm or graph repr tation.
2. Relying on an existing distributed computing plat often ill-suited for graph processing. MapReduc for example, is a very good fit for a wide array of scale computing problems. It is sometimes us mine large graphs [11, 30], but this can lead to optimal performance and usability issues. The models for processing data have been extended cilitate aggregation [41] and SQL-like queries [4 but these extensions are usually not ideal for gra gorithms that often better fit a message passing m
3. Using a single-computer graph algorithm library as BGL [43], LEDA [35], NetworkX [25], JDSL Stanford GraphBase [29], or FGL [16], limitin scale of problems that can be addressed.
4. Using an existing parallel graph system. The Pa

Giraph

- Apache Open Source implementation of Pregel
- Runs on Hadoop, (ab)uses mappers to do so
- Used at LinkedIn and Facebook

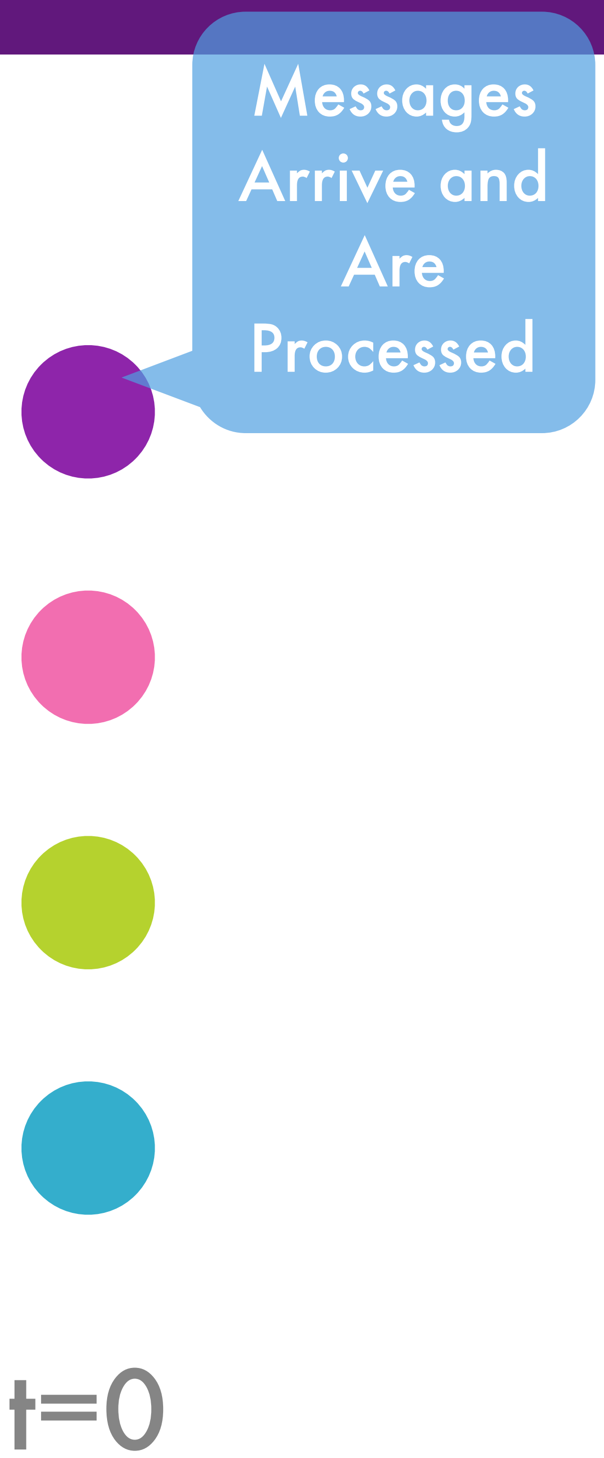
http://incubator.apache.org/giraph/

Pregel Visually

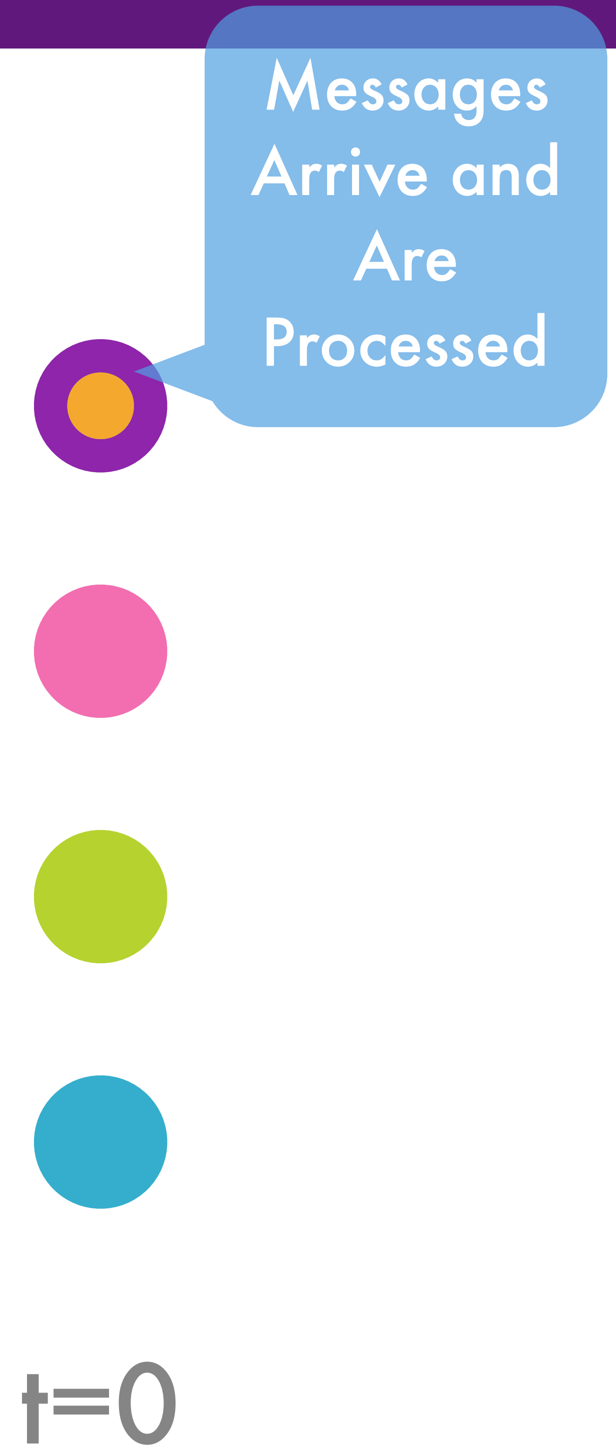


t=0

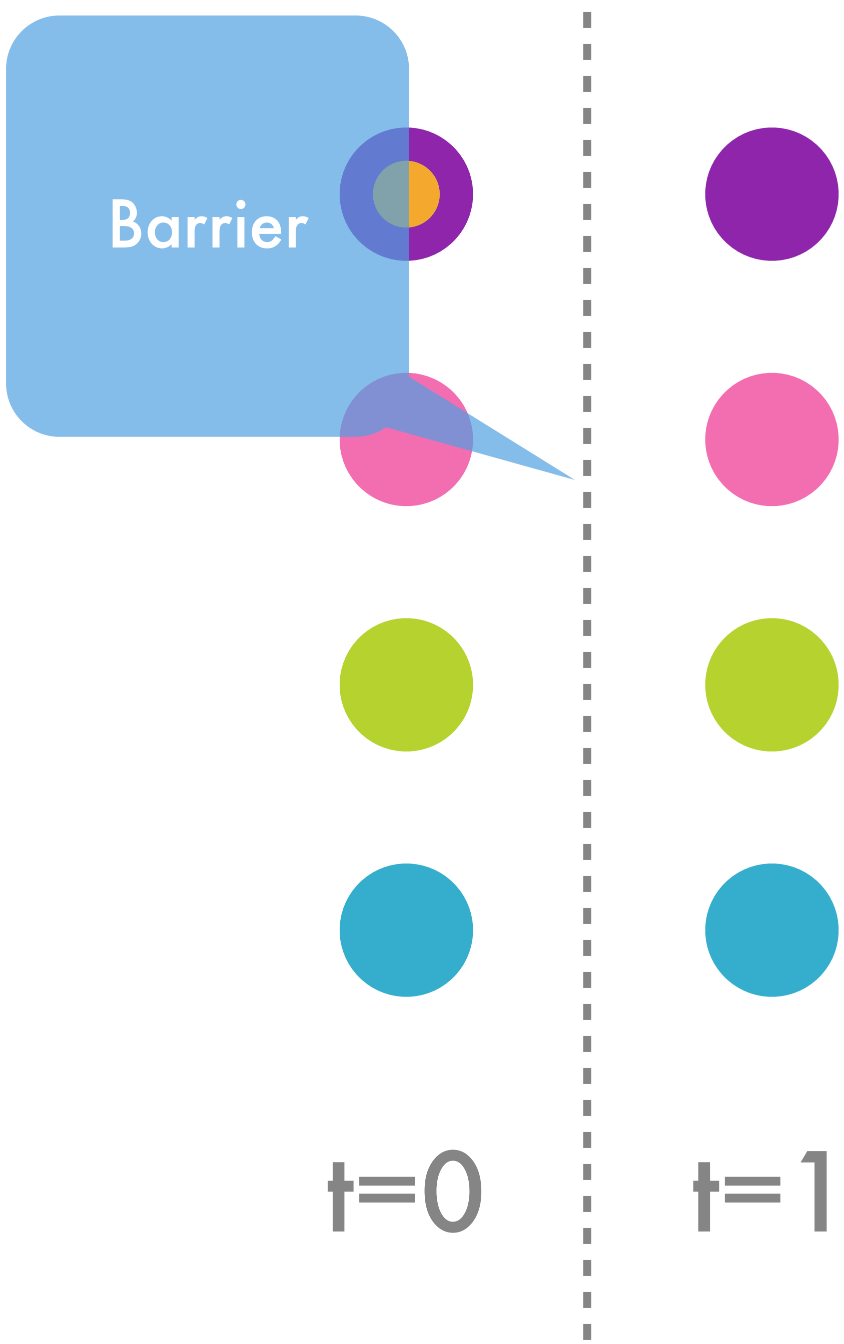
Pregel Visually



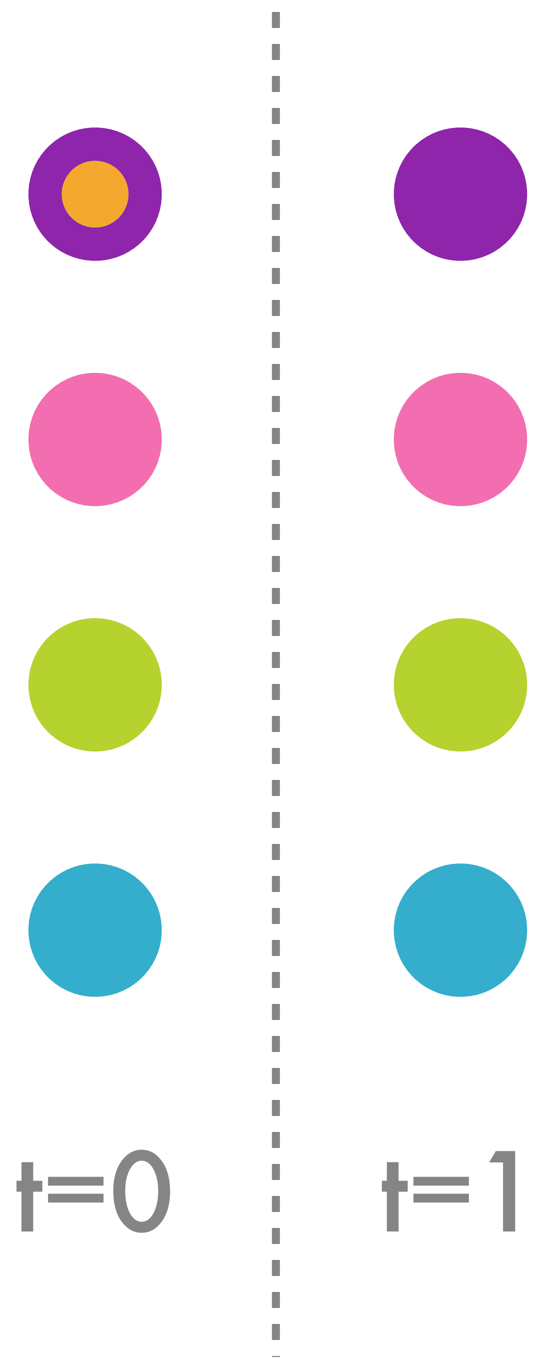
Pregel Visually



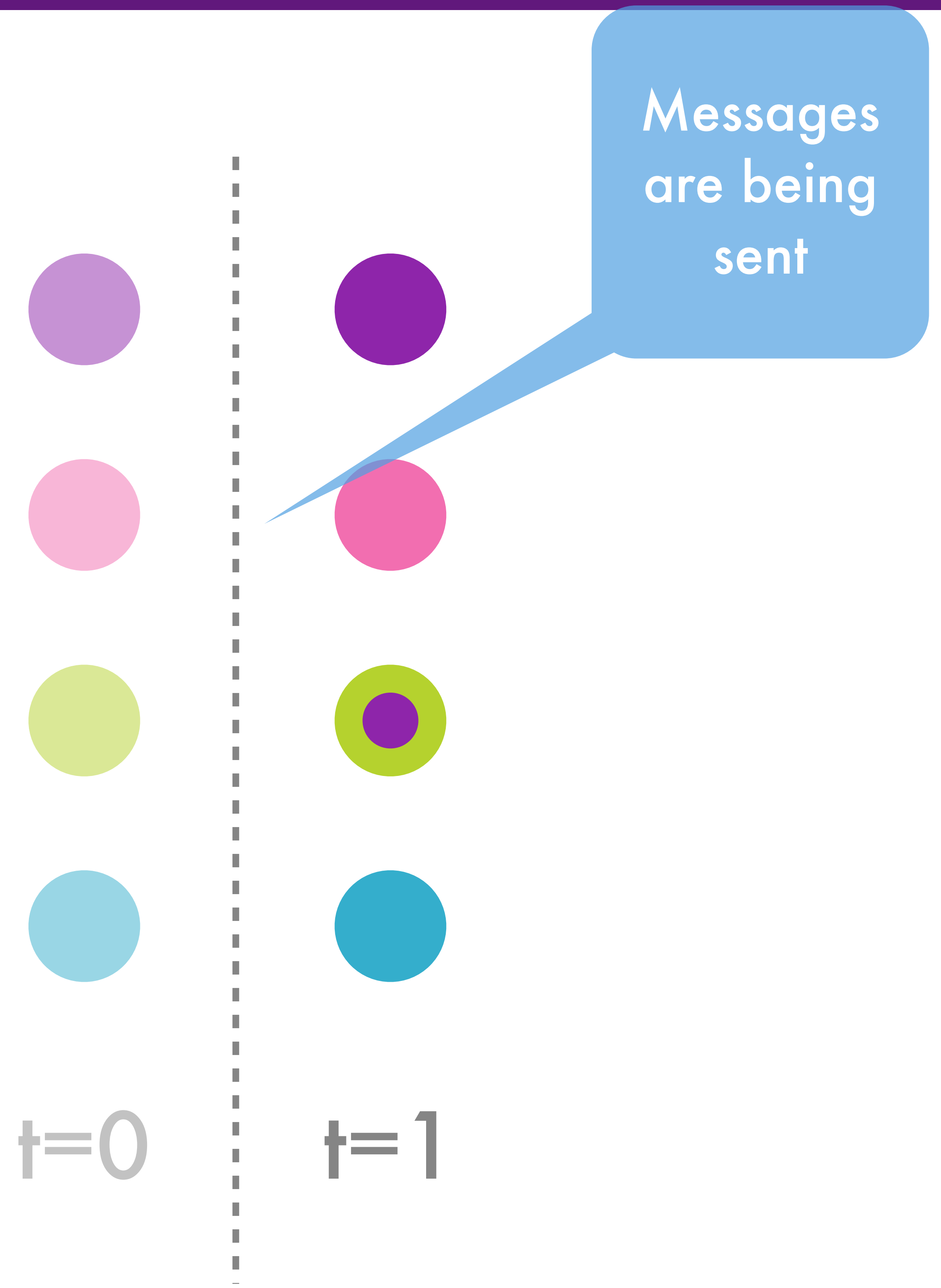
Pregel Visually



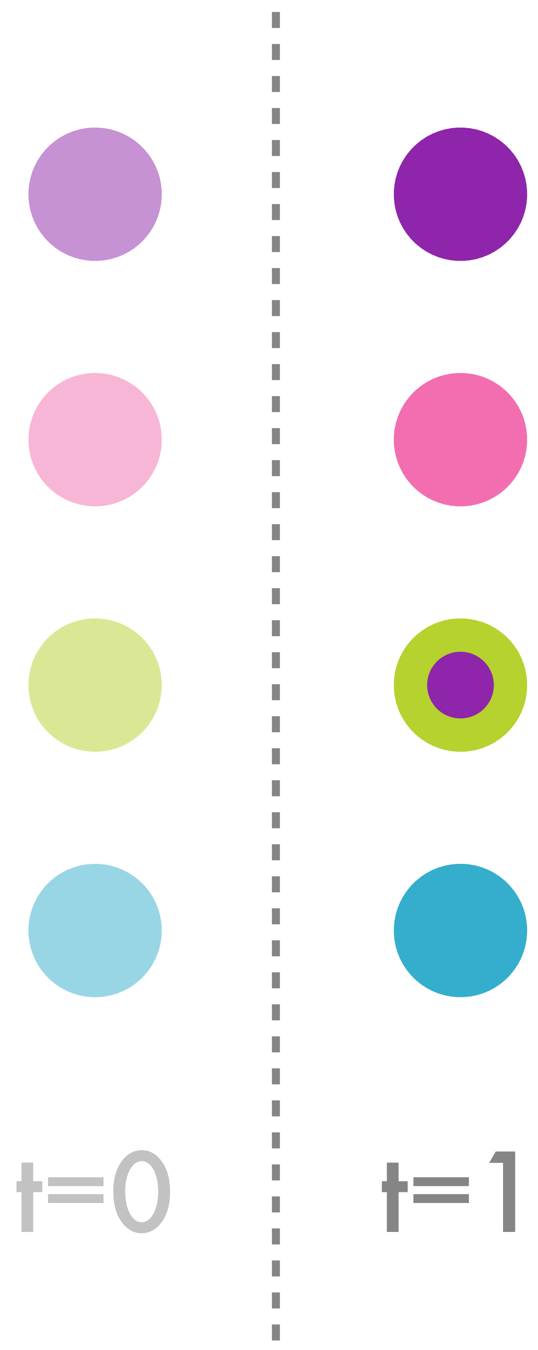
Pregel Visually



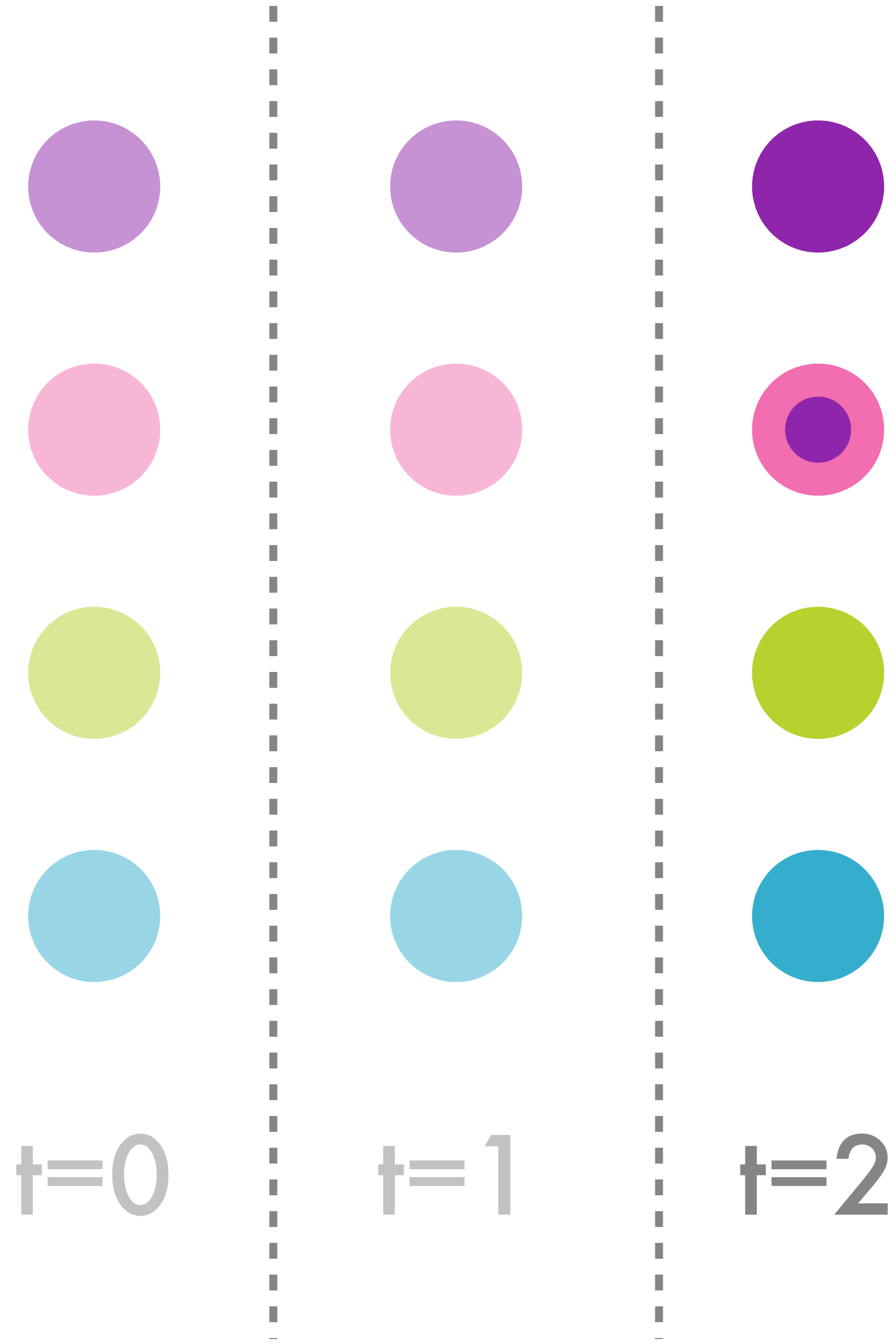
Pregel Visually



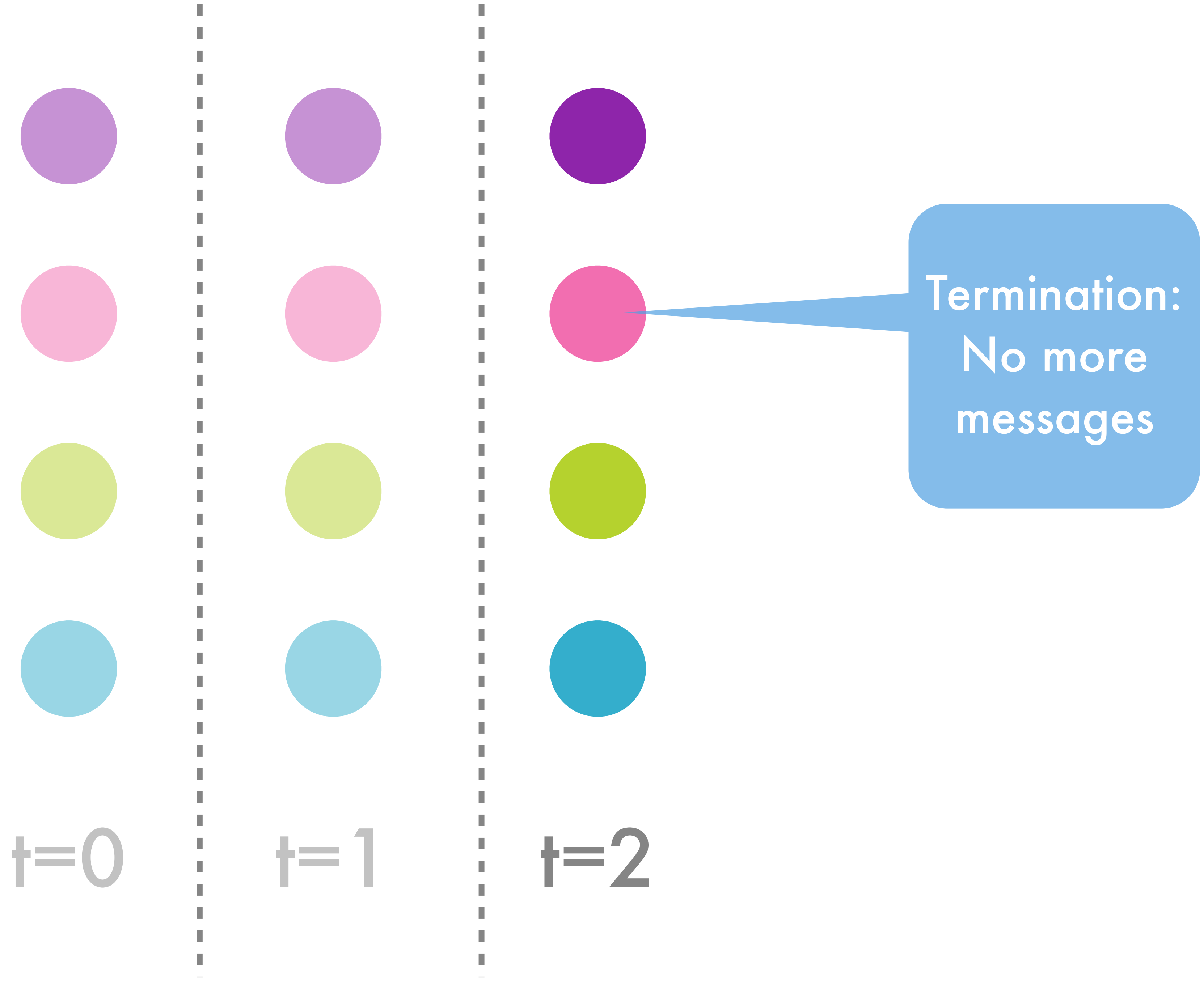
Pregel Visually



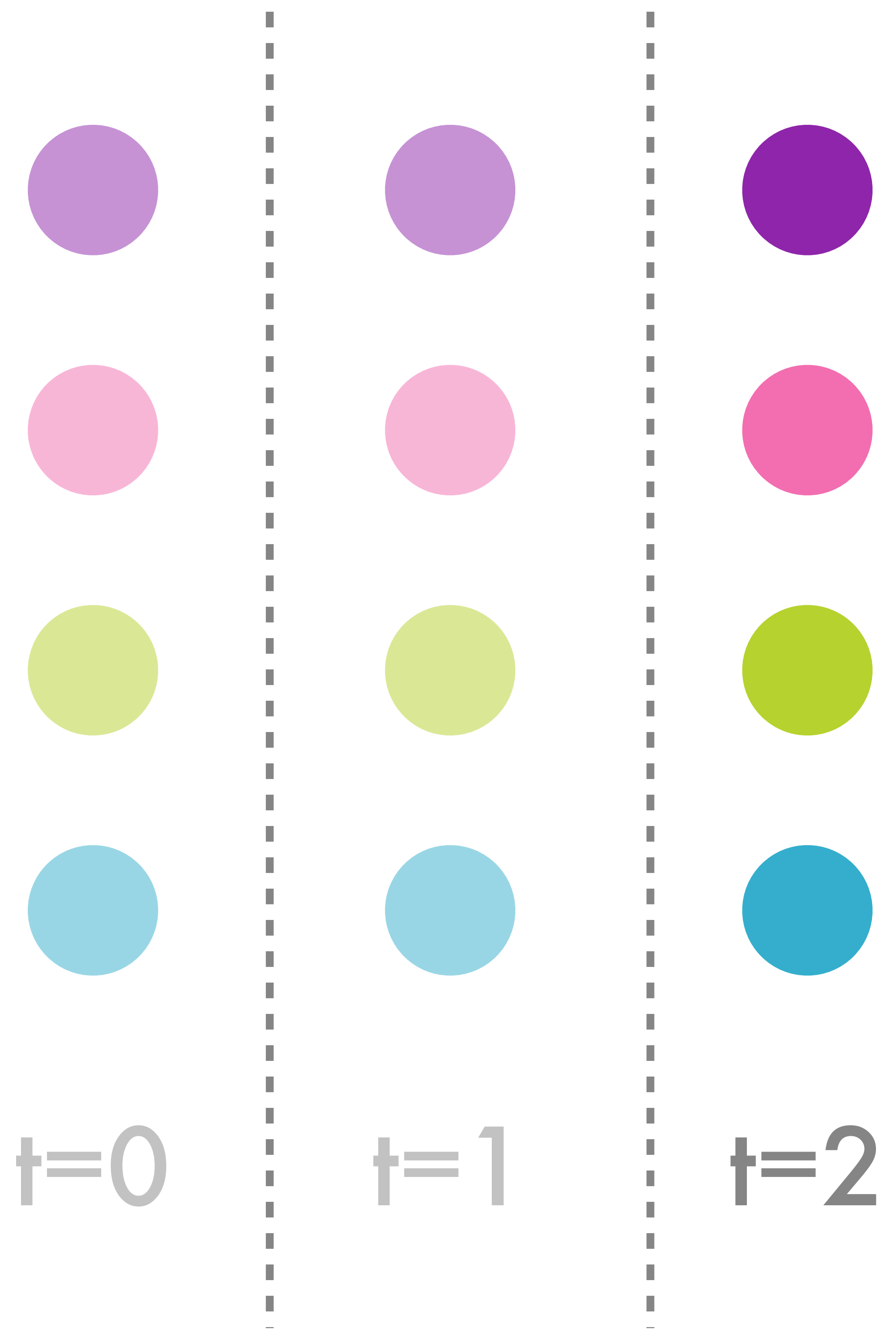
Pregel Visually



Pregel Visually



Pregel Visually



Pregel: PageRank

- `update()` receives the PageRank of all neighbors
- Updates its local PageRank
- Sends new PageRank around if it changed enough

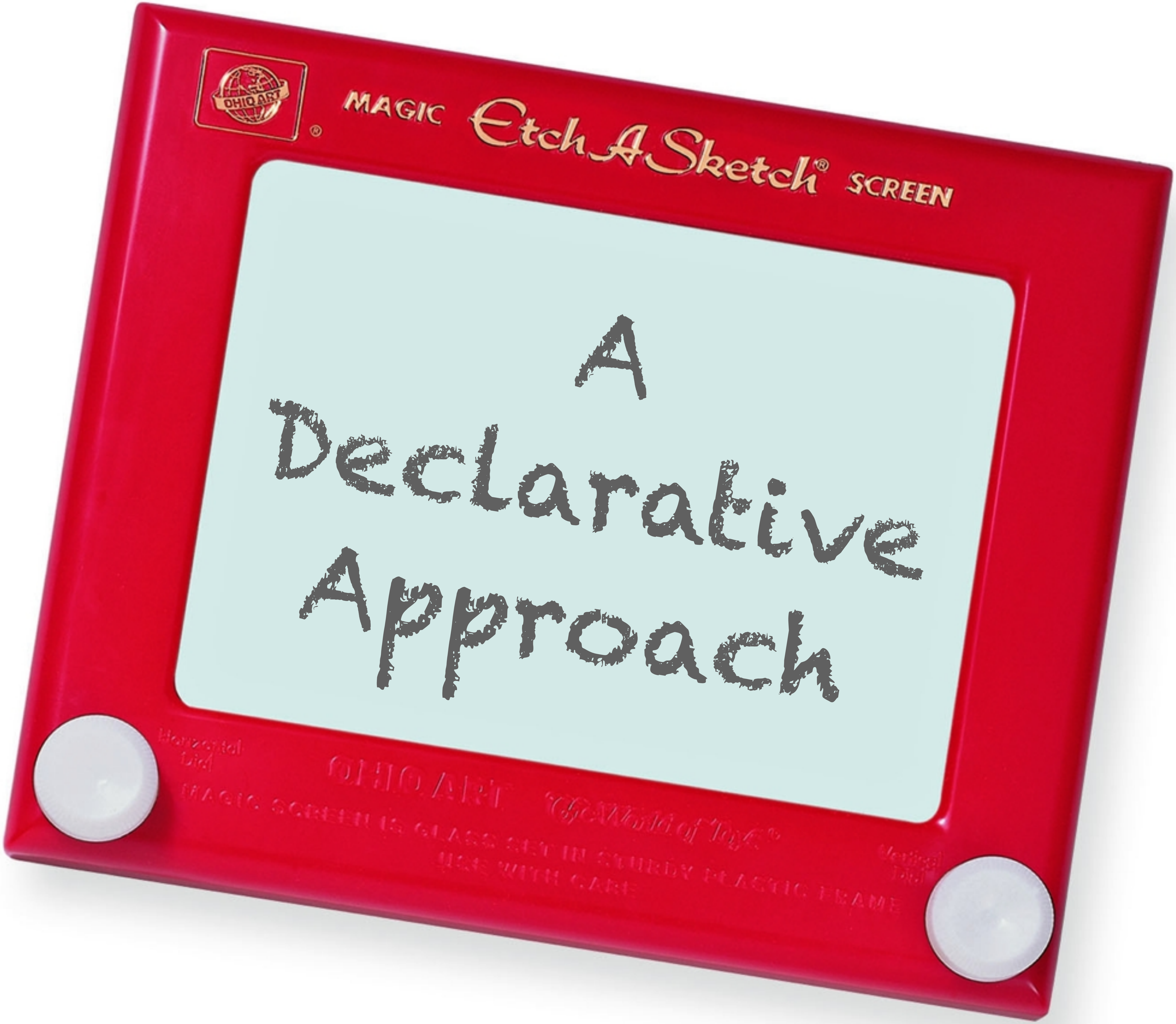
Pregel: Conclusion

- **The Good**
  - Excellent Map for Graph problems
  - Fast
- **The Bad**
  - Memory Model
  - Main Memory Assumption
- **The Ugly**
  - Wrong computational model (stay for the afternoon)

Open Problems

- No complete isolation of user / systems code
  - Unlike MapReduce

- No **one** system for example formation and modeling
  - Learning Effort
  - Orchestration
  - Wasted resources in distributed clusters


OHIO ART
MAGIC Etch A Sketch® SCREEN
A
Declarative
Approach
OHIO ART

Joint Work With

Yingyi Bu, Vinayak Borkar, Michael J. Carey
University of California, Irvine

Joshua Rosen, Neoklis Polyzotis
University of California, Santa Cruz

Joshua Rosen, Neoklis Polyzotis
University of California, Santa Cruz

Goals

- **Unify Example Formation and Modeling**
  - Relational Algebra Operators
  - Iteration Support
  - A unified runtime

- **Increase Productivity via high-level language**
  - Insulate the user from the systems aspects
  - Debugging and IDE support

Approach

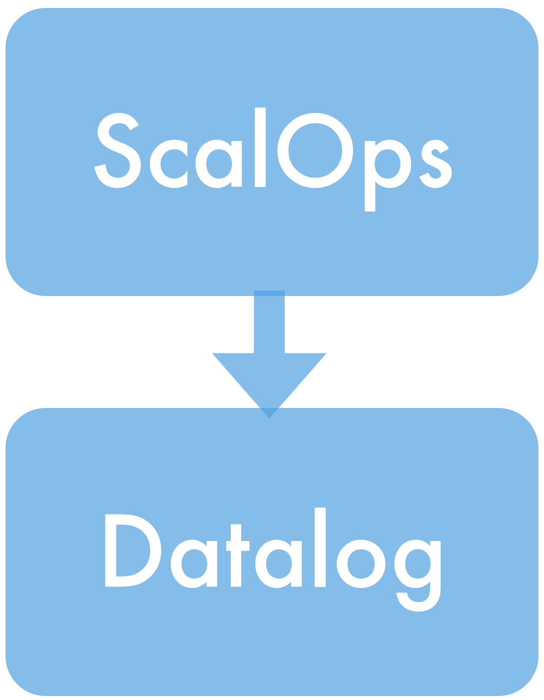
Approach

ScalOps

High Level Language

Relational Algebra and Loops

Approach



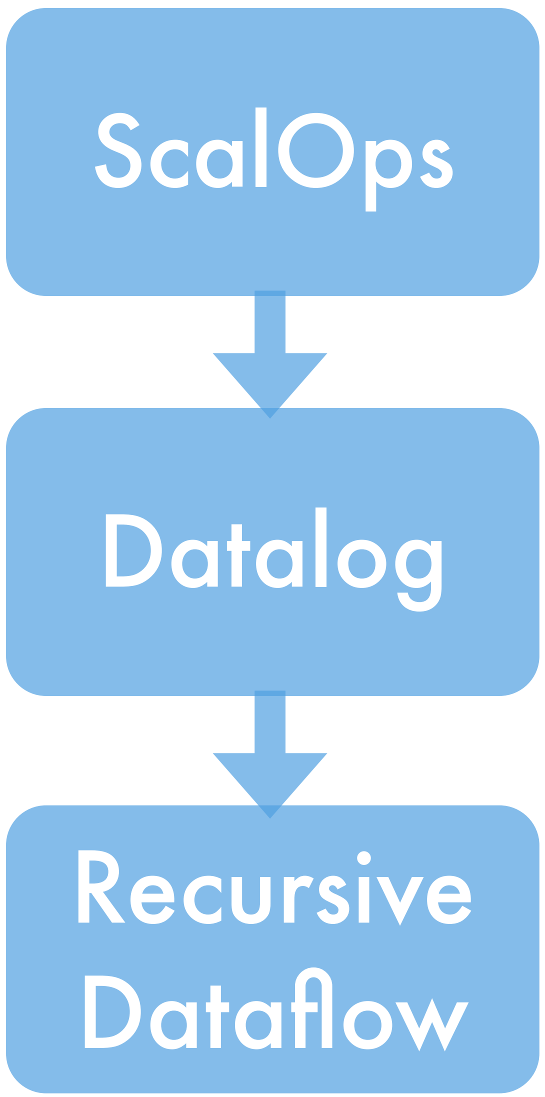
High Level Language

Relational Algebra and Loops

Declarative Language

Captures the Recursive Dataflow

Approach



High Level Language

Relational Algebra and Loops

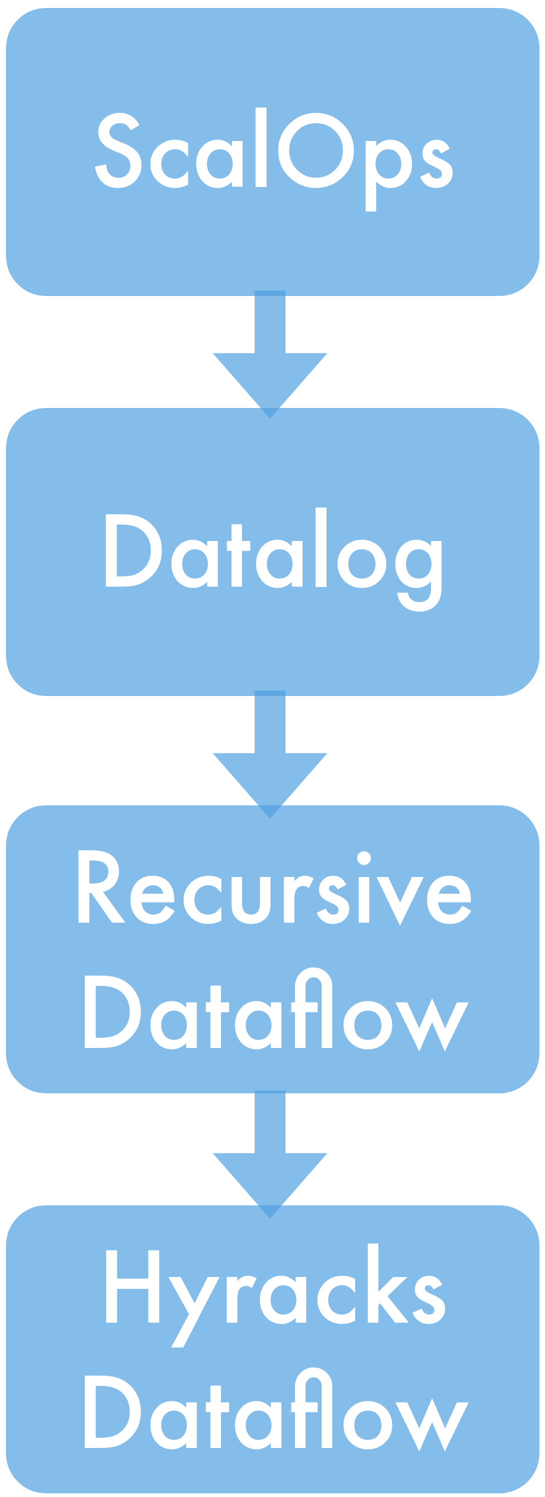
Declarative Language

Captures the Recursive Dataflow

Suite of data-parallel operators

Selected by an Optimizer / Compiler

Approach



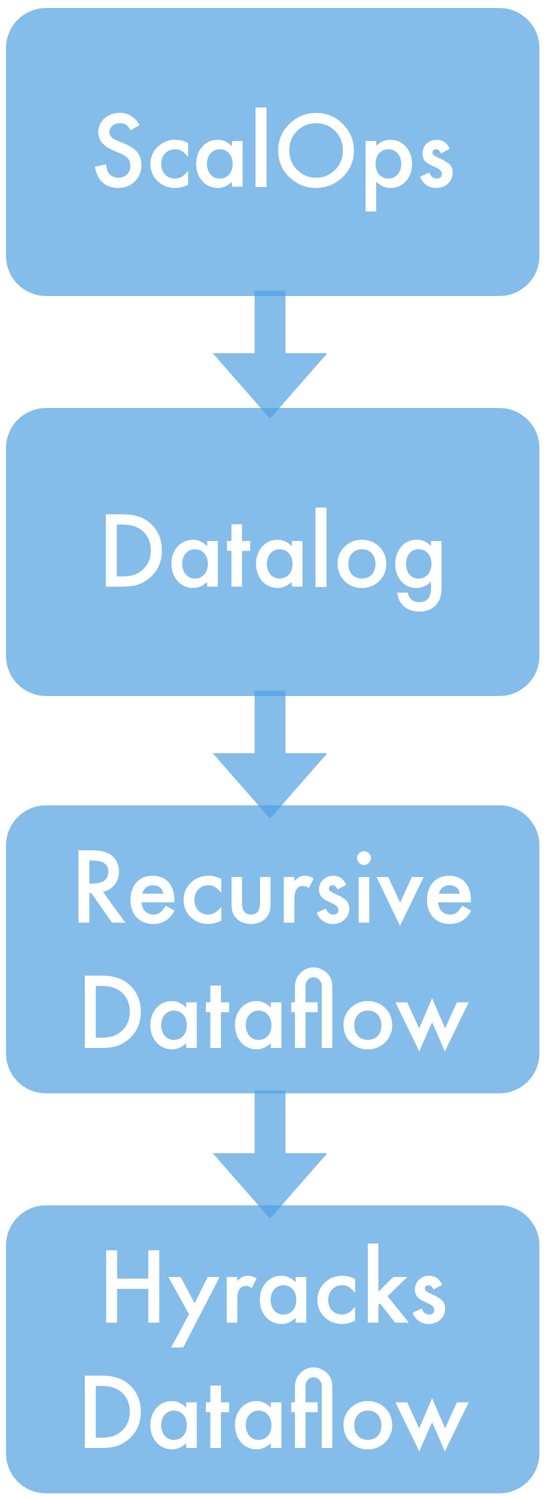
High Level Language
Relational Algebra and Loops

Declarative Language
Captures the Recursive Dataflow

Suite of data-parallel operators
Selected by an Optimizer / Compiler

Unified Runtime
Scalability + High performance

Approach



High Level Language
Relational Algebra and Loops

Declarative Language
Captures the Recursive Dataflow

Suite of data-parallel operators
Selected by an Optimizer / Compiler

Unified Runtime
Scalability + High performance

Approach

ScalOps — **High Level Language**
Relational Algebra and Loops

Datalog — **Declarative Language**
Captures the Recursive Dataflow

Recursive Dataflow — **Suite of data-parallel operators**
Selected by an Optimizer / Compiler

Hyracks Dataflow — **Unified Runtime**
Scalability + High performance

ScalOps

- Internal Domain Specific Language (DSL)
  - Written in Scala
- Relational Algebra (Filter, Join, GroupBy, ...)
- Iteration support
- Rich UDF support
  - Inline Scala function calls / literals
  - Byte-code compatible with Java
- Support in major IDEs

BGD in ScalOps

```
def train(xy:Table[Example],
          compute_grad:(Example, Vector) => Vector,
          compute_loss:(Example, Vector) => Double) = {

  class Env(w:VectorType, lastError:DoubleType, delta:DoubleType) extends Environment

  val initialValue = new Env(VectorType.zeros(1000), Double.MaxValue, Double.MaxValue)

  loop(initialValue, (env: Env) => env.delta < eps) { env => {
      val gradient = xy.map(x=>compute_grad(x,env.w)).reduce(_+_)
      val loss     = xy.map(x=>compute_loss(x,env.w)).reduce(_+_)
      env.w       -= gradient
      env.delta    = env.lastLoss - loss
      env.lastLoss = loss
      env
    }
  }
}
```

BGD in ScalOps

```
def train(xy:Table[Example],
          compute_grad:(Example, Vector) => Vector,
          compute_loss:(Example, Vector) => Double) = {

  class Env(w:VectorType, lastError:DoubleType, delta:DoubleType) extends Environment

  val initialValue = new Env(VectorType.zeros(1000), Double.MaxValue, Double.MaxValue)

  loop(initialValue, (env: Env) => env.delta < eps) { env => {
      val gradient = xy.map(x=>compute_grad(x,env.w)).reduce(_+_)
      val loss     = xy.map(x=>compute_loss(x,env.w)).reduce(_+_)
      env.w       -= gradient
      env.delta    = env.lastLoss - loss
      env.lastLoss = loss
      env
    }
  }
}
```

BGD in ScalOps

```
def train(xy:Table[Example],
          compute_grad:(Example, Vector) => Vector,
          compute_loss:(Example, Vector) => Double) = {

  class Env(w:VectorType ... DoubleType) extends Environment

  val initialValue = new ... ouble.MaxValue, Double.MaxValue)

  loop(initialValue, (env: Env) => env.delta < eps) { env => {
    val gradient = xy.map(x=>compute_grad(x,env.w)).reduce(_+_)
    val loss     = xy.map(x=>compute_loss(x,env.w)).reduce(_+_)
    env.w       -= gradient
    env.delta    = env.lastLoss - loss
    env.lastLoss = loss
    env
  }
  }
}
```

Table is our Dataset type

BGD in ScalOps

```
class Example(x:Vector, y:Double)
```

```
def train(xy:Table[Example],
          compute_grad:(Example, Vector) => Vector,
          compute_loss:(Example, Vector) => Double) = {

  class Env(w:VectorType                DoubleType) extends Environment

  val initialValue = new                ouble.MaxValue, Double.MaxValue)

  loop(initialValue, (env: Env) => env.delta < eps) { env => {
      val gradient = xy.map(x=>compute_grad(x,env.w)).reduce(_+_)
      val loss     = xy.map(x=>compute_loss(x,env.w)).reduce(_+_)
      env.w       -= gradient
      env.delta    = env.lastLoss - loss
      env.lastLoss = loss
      env
    }
  }
}
```

Table is our Dataset type

BGD in ScalOps

```
def train(xy:Table[Example],
          compute_grad:(Example, Vector) => Vector,
          compute_loss:(Example, Vector) => Double) = {

  class Env(w:VectorType, lastError:DoubleType, delta:DoubleType) extends Environment

  val initialValue = new Env(VectorType.zeros(1000), Double.MaxValue, Double.MaxValue)

  loop(initialValue, (env: Env) => env.delta < eps) { env => {
      val gradient = xy.map(x=>compute_grad(x,env.w)).reduce(_+_)
      val loss     = xy.map(x=>compute_loss(x,env.w)).reduce(_+_)
      env.w       -= gradient
      env.delta    = env.lastLoss - loss
      env.lastLoss = loss
      env
    }
  }
}
```

BGD in ScalOps

Gradient Function

```
def train(xy:Table[Example],
          compute_grad:(Example, Vector) => Vector,
          compute_loss:(Example, Vector) => Double) = {

  class Env(w:VectorType, lastError:DoubleType, delta:DoubleType) extends Environment

  val initialValue = new Env(VectorType.zeros(1000), Double.MaxValue, Double.MaxValue)

  loop(initialValue, (env: Env) => env.delta < eps) { env => {
      val gradient = xy.map(x=>compute_grad(x,env.w)).reduce(_+_)
      val loss     = xy.map(x=>compute_loss(x,env.w)).reduce(_+_)
      env.w       -= gradient
      env.delta    = env.lastLoss - loss
      env.lastLoss = loss
      env
    }
  }
}
```

BGD in ScalOps

Gradient Function

Loss Function

```
def train(xy:Table[ xample],
          compute_grad:(Example, Vector) => Vector,
          compute_loss:(Example, Vector) => Double) = {

  class Env(w:Vecto Type, lastError:DoubleType, delta:DoubleType) extends Environment

  val initialValue                          0), Double.MaxValue, Double.MaxValue)

  loop(initialValu                          s) { env => {
      val gradient                          .w)).reduce(_+_)
      val loss     = xy.map(x=>compute_loss(x,env.w)).reduce(_+_)
      env.w       -= gradient
      env.delta    = env.lastLoss - loss
      env.lastLoss = loss
      env
    }
  }
}
```

BGD in ScalOps

```
def train(xy:Table[Example],
          compute_grad:(Example, Vector) => Vector,
          compute_loss:(Example, Vector) => Double) = {

  class Env(w:VectorType, lastError:DoubleType, delta:DoubleType) extends Environment

  val initialValue = new Env(VectorType.zeros(1000), Double.MaxValue, Double.MaxValue)

  loop(initialValue, (env: Env) => env.delta < eps) { env => {
      val gradient = xy.map(x=>compute_grad(x,env.w)).reduce(_+_)
      val loss     = xy.map(x=>compute_loss(x,env.w)).reduce(_+_)
      env.w       -= gradient
      env.delta    = env.lastLoss - loss
      env.lastLoss = loss
      env
    }
  }
}
```

BGD in ScalOps

Compute gradient

```
def train(xy:Table[Example],
                                  r) => Vector,
                                  ) => Double) = {

  class E                         bleType, delta:DoubleType) extends Environment

  val ini                         .zeros(1000), Double.MaxValue, Double.MaxValue)

  loop(initialValue, (env: Env) => env.delta < eps) { env => {
      val gradient = xy.map(x=>compute_grad(x,env.w)).reduce(_+_)
      val loss     = xy.map(x=>compute_loss(x,env.w)).reduce(_+_)
      env.w       -= gradient
      env.delta    = env.lastLoss - loss
      env.lastLoss = loss
      env
    }
  }
}
```

BGD in ScalOps

Compute gradient

Sum it up

```
def train(xy:Table[Example],
          ...r) => Vecto...
          ...) => Doub...

  class E...bleType, ...ronment

  val ini....zeros(10...axValue)

  loop(initialValue, (env: Env) => env.delta < eps) { env => {
      val gradient = xy.map(x=>compute_grad(x,env.w)).reduce(_+_)
      val loss     = xy.map(x=>compute_loss(x,env.w)).reduce(_+_)
      env.w       -= gradient
      env.delta    = env.lastLoss - loss
      env.lastLoss = loss
      env
    }
  }
}
```

BGD in ScalOps

```
def train(xy:Table[Example],
                                      r) => Vecto
                                      ) => Doub

  class E                              bleType,                              ronment

  val ini                             .zeros(10                              axValue)

  loop(initialValue, (env: Env) => env.delta < eps) { env => {
      val gradient = xy.map(x=>compute_grad(x,env.w)).reduce(_+_)
      val loss     = xy.map(x=>compute_loss(x,env.w)).reduce(_+_)
      env.w        -= gradient
      env.delta    = env.lastLoss - loss
      env.lastLoss = loss
      env
    }
  }
}
```

Compute gradient

Sum it up

Update the model

BGD in ScalOps

```
def train(xy:Table[Example],
          compute_grad:(Example, Vector) => Vector,
          compute_loss:(Example, Vector) => Double) = {

  class Env(w:VectorType, lastError:DoubleType, delta:DoubleType) extends Environment

  val initialValue = new Env(VectorType.zeros(1000), Double.MaxValue, Double.MaxValue)

  loop(initialValue, (env: Env) => env.delta < eps) { env => {
      val gradient = xy.map(x=>compute_grad(x,env.w)).reduce(_+_)
      val loss     = xy.map(x=>compute_loss(x,env.w)).reduce(_+_)
      env.w       -= gradient
      env.delta    = env.lastLoss - loss
      env.lastLoss = loss
      env
    }
  }
}
```

BGD in ScalOps

Compute loss

```
def train(xy:Table[Example],
          compute_grad:(Example, Vector) => Vector,
          compute_loss:(Example, Vector) => Double) = {

  class E… bleType, delta:DoubleType) extends Environment

  val ini… .zeros(1000), Double.MaxValue, Double.MaxValue)

  loop(initialValue, (env: Env) => env.delta < eps) { env => {
      val gradient = xy.map(x=>compute_grad(x,env.w)).reduce(_+_)
      val loss     = xy.map(x=>compute_loss(x,env.w)).reduce(_+_)
      env.w       -= gradient
      env.delta    = env.lastLoss - loss
      env.lastLoss = loss
      env
    }
  }
}
```

BGD in ScalOps

```
def train(xy:Table[Example],
          compute_grad:(Example, Vector) => Vector,
          compute_loss:(Example, Vector) => Double) = {

  class E                              bleType,                              ronment

  val ini                              .zeros(10                             axValue)

  loop(initialValue, (env: Env) =   env.delta < eps) { env => {
      val gradient = xy.map(x=>compute_grad(x,env.w)).reduce(_+_)
      val loss     = xy.map(x=>compute_loss(x,env.w)).reduce(_+_)
      env.w       -= gradient
      env.delta    = env.lastLoss - loss
      env.lastLoss = loss
      env
    }
  }
}
```

Compute loss

Sum it up

BGD in ScalOps

```
def train(xy:Table[Example],
          compute_grad:(Example, Vector) => Vector,
          compute_loss:(Example, Vector) => Double) = {

  class E                              bleType,                              ronment

  val ini                              .zeros(10                             axValue)

  loop(initialValue, (env: Env) => env.delta < eps) { env => {
    val gradient = xy.map(x=>compute_grad(x,env.w)).reduce(_+_)
    val loss     = xy.map(x=>compute_loss(x,env.w)).reduce(_+_)
    env.w       -= gradient
    env.delta    = env.lastLoss - loss
    env.lastLoss = loss
    env
   }
  }
}
```

Compute loss

Sum it up

Update convergence

BGD in ScalOps

```
def train(xy:Table[Example],
          compute_grad:(Example, Vector) => Vector,
          compute_loss:(Example, Vector) => Double) = {

  class Env(w:VectorType, lastError:DoubleType, delta:DoubleType) extends Environment

  val initialValue = new Env(VectorType.zeros(1000), Double.MaxValue, Double.MaxValue)

  loop(initialValue, (env: Env) => env.delta < eps) { env => {
      val gradient = xy.map(x=>compute_grad(x,env.w)).reduce(_+_)
      val loss     = xy.map(x=>compute_loss(x,env.w)).reduce(_+_)
      env.w       -= gradient
      env.delta    = env.lastLoss - loss
      env.lastLoss = loss
      env
    }
  }
}
```

BGD in ScalOps

Shared Loop State

```
le, Vector) => Vector,
ompute_loss:(Example, Vector) => Double) = {

  class Env(w:VectorType, lastError:DoubleType, delta:DoubleType) extends Environment

  val initialValue = new Env(VectorType.zeros(1000), Double.MaxValue, Double.MaxValue)

  loop(initialValue, (env: Env) => env.delta < eps) { env => {
      val gradient = xy.map(x=>compute_grad(x,env.w)).reduce(_+_)
      val loss     = xy.map(x=>compute_loss(x,env.w)).reduce(_+_)
      env.w       -= gradient
      env.delta    = env.lastLoss - loss
      env.lastLoss = loss
      env
    }
  }
}
```

BGD in ScalOps

```
def train(xy:Table[Example],
          compute_grad:(Example, Vector) => Vector,
          compute_loss:(Example, Vector) => Double) = {

  class Env(w:VectorType, lastError:DoubleType, delta:DoubleType) extends Environment

  val initialValue = new Env(VectorType.zeros(1000), Double.MaxValue, Double.MaxValue)

  loop(initialValue, (env: Env) => env.delta < eps) { env => {
      val gradient = xy.map(x=>compute_grad(x,env.w)).reduce(_+_)
      val loss     = xy.map(x=>compute_loss(x,env.w)).reduce(_+_)
      env.w       -= gradient
      env.delta    = env.lastLoss - loss
      env.lastLoss = loss
      env
    }
  }
}
```

BGD in ScalOps

Initializer

```
def train(xy:Table[Example],
                e, Vector) => Vector,
                e, Vector) => Double) = {

                :Error:DoubleType, delta:DoubleType) extends Environment

  val init   lValue = new Env(VectorType.zeros(1000), Double.MaxValue, Double.MaxValue)

  loop(initialValue, (env: Env) => env.delta < eps) { env => {
      val gradient = xy.map(x=>compute_grad(x,env.w)).reduce(_+_)
      val loss     = xy.map(x=>compute_loss(x,env.w)).reduce(_+_)
      env.w       -= gradient
      env.delta    = env.lastLoss - loss
      env.lastLoss = loss
      env
    }
  }
}
```

BGD in ScalOps

Initializer

Loop Condition

```
def train(xy:Table[Example],
                 e, V
                 e, V

                 :Erro                                          ) extends Environment

  val initialValue = new Env(VectorType.zeros(1000), Double.MaxValue, Double.MaxValue)

  loop(initialValue, (env: Env) => env.delta < eps) { env => {
      val gradient = xy.map(x=>compute_grad(x,env.w)).reduce(_+_)
      val loss     = xy.map(x=>compute_loss(x,env.w)).reduce(_+_)
      env.w       -= gradient
      env.delta    = env.lastLoss - loss
      env.lastLoss = loss
      env
    }
  }
}
```

BGD in ScalOps

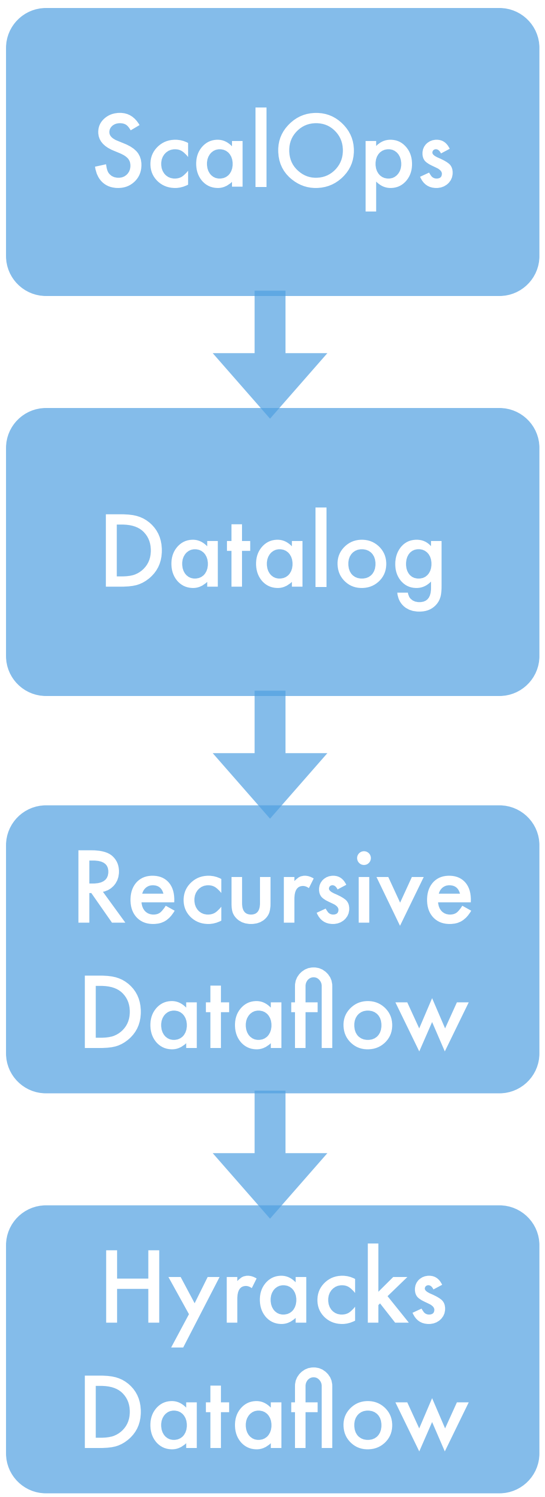
Initializer

Loop Condition

Loop Body

```
def train(xy:Table[Example],
                 e, V
                 e, V

                :Erro                              ) extends Environment

  val init   lValue = new Env(VectorType.zeros 1000), Double.MaxValue, Double.MaxValue)

  loop(initialValue, (env: Env) => env.delta < eps) { env => {
      val gradient = xy.map(x=>compute_grad(x,env.w)).reduce(_+_)
      val loss     = xy.map(x=>compute_loss(x,env.w)).reduce(_+_)
      env.w       -= gradient
      env.delta    = env.lastLoss - loss
      env.lastLoss = loss
      env
    }
  }
}
```

Approach



High Level Language
Relational Algebra and Loops

Declarative Language
Captures the Recursive Dataflow

Suite of data-parallel operators
Selected by an Optimizer / Compiler

Unified Runtime
Scalability + High performance

Approach

ScalOps — High Level Language
Relational Algebra and Loops

Datalog — Declarative Language
Captures the Recursive Dataflow

Recursive Dataflow — Suite of data-parallel operators
Selected by an Optimizer / Compiler

Hyracks Dataflow — Unified Runtime
Scalability + High performance

BGD in ScalOps

```
def train(xy:Table[Example],
          compute_grad:(Example, Vector) => Vector,
          compute_loss:(Example, Vector) => Double) = {

  class Env(w:VectorType, lastError:DoubleType, delta:DoubleType) extends Environment

  val initialValue = new Env(VectorType.zeros(1000), Double.MaxValue, Double.MaxValue)

  loop(initialValue, (env: Env) => env.delta < eps) { env => {
      val gradient = xy.map(x=>compute_grad(x,env.w)).reduce(_+_)
      val loss     = xy.map(x=>compute_loss(x,env.w)).reduce(_+_)
      env.w       -= gradient
      env.delta    = env.lastLoss - loss
      env.lastLoss = loss
      env
    }
  }
}
```

Automatic Optimizations

```
def train(xy:Table[Example],
          compute_grad:(Example, Vector) => Vector,
          compute_loss:(Example, Vector) => Double) = {

  class Env(w:VectorType, lastError:DoubleType, delta:DoubleType) extends Environment

  val initialValue = new Env(VectorType.zeros(1000), Double.MaxValue, Double.MaxValue)

  loop(initialValue, (env: Env) => env.delta < eps) { env => {
      val gradient = xy.map(x=>compute_grad(x,env.w)).reduce(_+_)
      val loss     = xy.map(x=>compute_loss(x,env.w)).reduce(_+_)
      env.w       -= gradient
      env.delta    = env.lastLoss - loss
      env.lastLoss = loss
      env
    }
  }
}
```

Automatic Optimizations

Merge into one MapReduce

```
def train(
                      => Vector,
                      => Double) = {

  class En                              leType, delta:DoubleType) extends Environment

  val initialValue =  ew Env(VectorType.zeros(1000), Double.MaxValue, Double.MaxValue)

  loop(initialValue, (env: Env) => env.delta < eps) { env => {
      val gradient = xy.map(x=>compute_grad(x,env.w)).reduce(_+_)
      val loss     = xy.map(x=>compute_loss(x,env.w)).reduce(_+_)
      env.w       -= gradient
      env.delta    = env.lastLoss - loss
      env.lastLoss = loss
      env
    }
  }
}
```

BGD in ScalOps

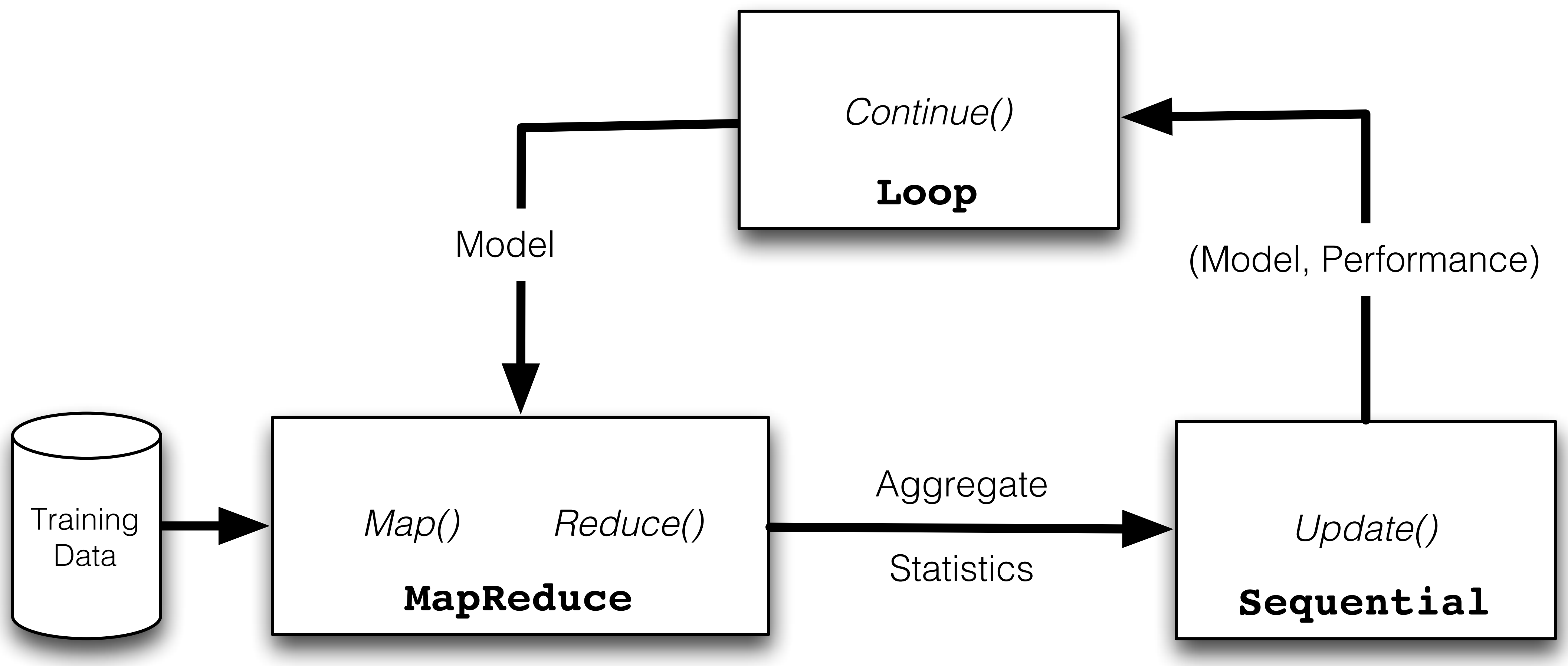
```
def train(xy:Table[Example],
          compute_grad:(Example, Vector) => Vector,
          compute_loss:(Example, Vector) => Double) = {

  class Env(w:VectorType, lastError:DoubleType, delta:DoubleType) extends Environment

  val initialValue = new Env(VectorType.zeros(1000), Double.MaxValue, Double.MaxValue)

  loop(initialValue, (env: Env) => env.delta < eps) { env => {
      val gradient = xy.map(x=>compute_grad(x,env.w)).reduce(_+_)
      val loss     = xy.map(x=>compute_loss(x,env.w)).reduce(_+_)
      env.w       -= gradient
      env.delta    = env.lastLoss - loss
      env.lastLoss = loss
      env
    }
  }
}
```

BGD in ScalOps

```
def train(xy:Table[Example],
          compute_grad:(Example, Vector) => Vector,
                                         => Double) = {

  class En                                 leType, delta:DoubleType) extends Environment

  val init                                 zeros(1000), Double.MaxValue, Double.MaxValue)

  loop(initialValue,   env: Env) => env.delta < eps) { env => {
      val gradient =  xy.map(x=>compute_grad(x,env.w)).reduce(_+_)
      val loss     =  xy.map(x=>compute_loss(x,env.w)).reduce(_+_)
      env.w        -= gradient
      env.delta     = env.lastLoss - loss
      env.lastLoss = loss
      env
    }
  }
}
```

Merge into one Operator

Logical Plan



Approach

ScalOps	**High Level Language** Relational Algebra and Loops
↓ Datalog	**Declarative Language** Captures the Recursive Dataflow
↓ Recursive Dataflow	**Suite of data-parallel operators** Selected by an Optimizer / Compiler
↓ Hyracks Dataflow	**Unified Runtime** Scalability + High performance

Approach

ScalOps — High Level Language
Relational Algebra and Loops

Datalog — Declarative Language
Captures the Recursive Dataflow

Recursive Dataflow — Suite of data-parallel operators
Selected by an Optimizer / Compiler

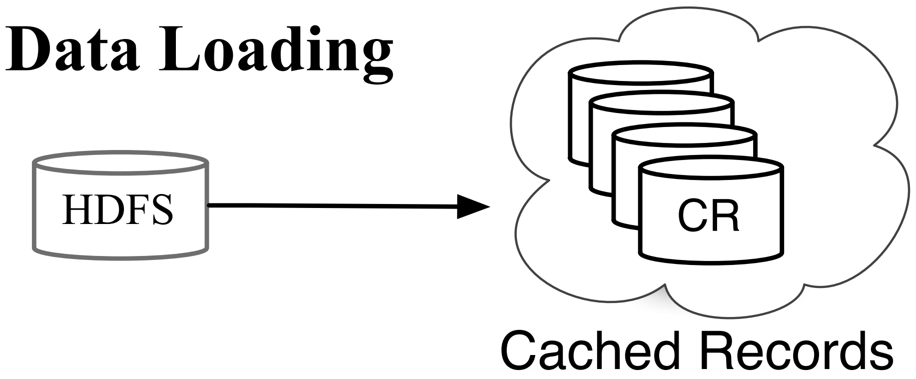
Hyracks Dataflow — Unified Runtime
Scalability + High performance

Some Optimizations

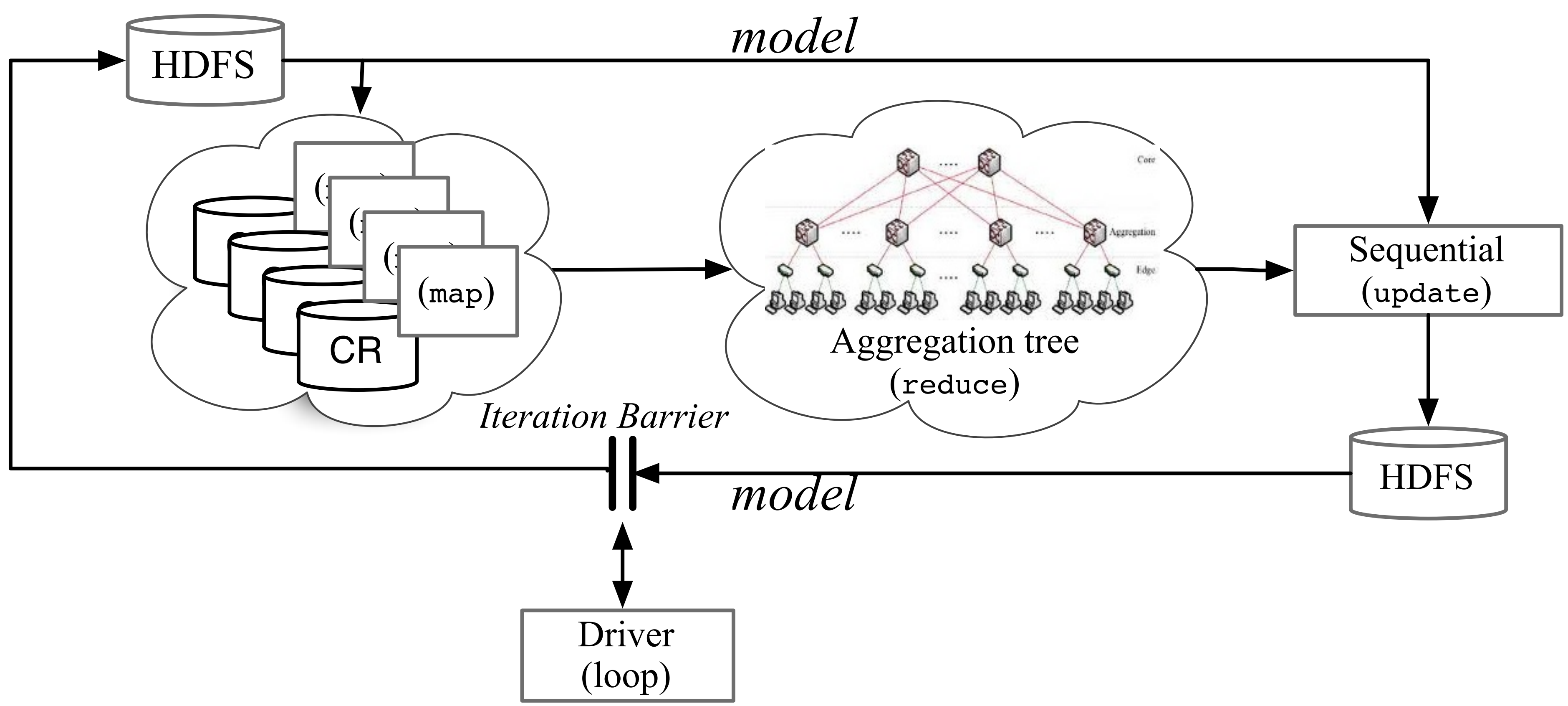
- Caching, Rocking
- Scheduling: Data-Local, Iteration-Aware
- Avoid (de-)serialization
- Minimize #network connections
- Pipelining

Physical Plan

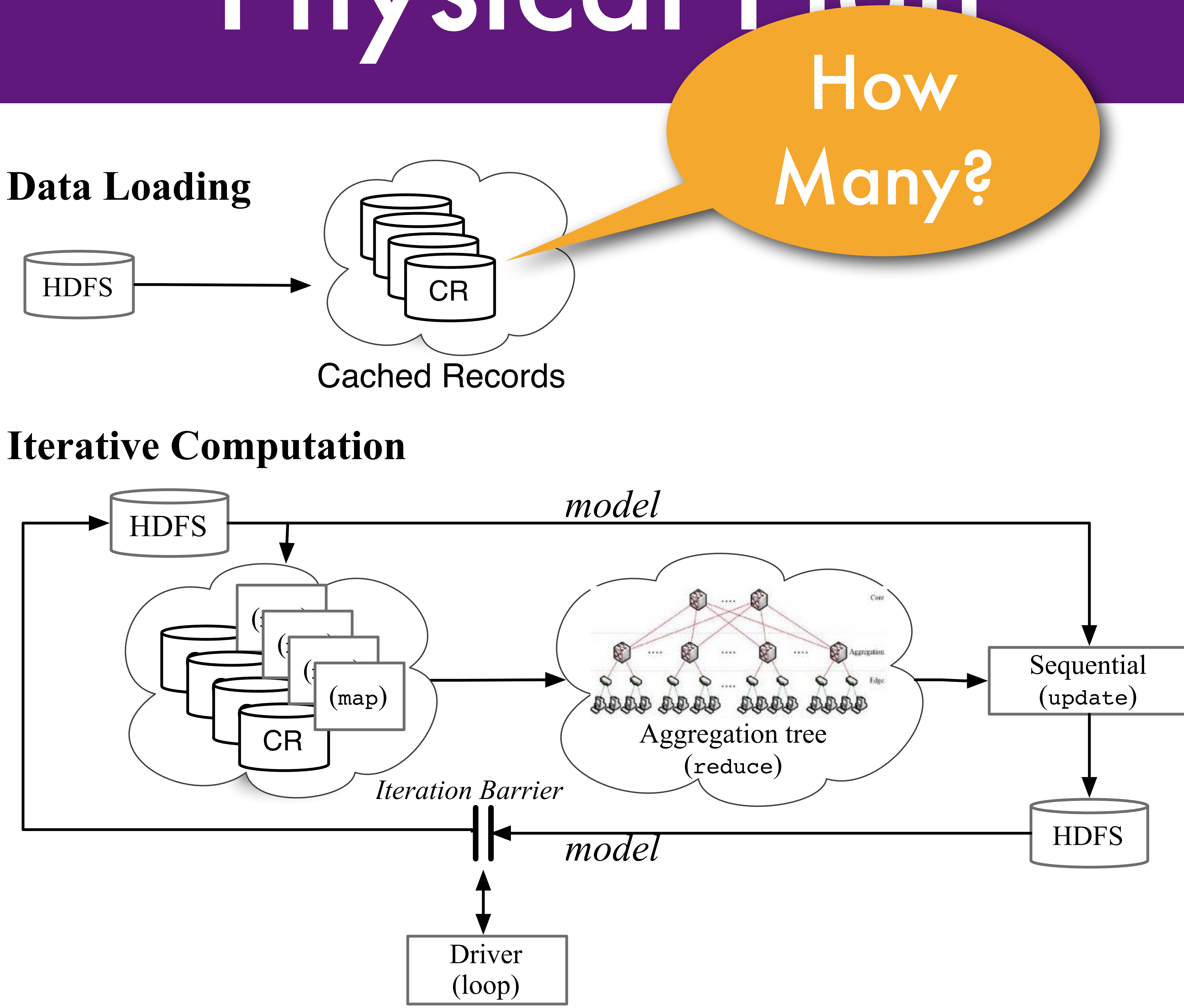
Data Loading



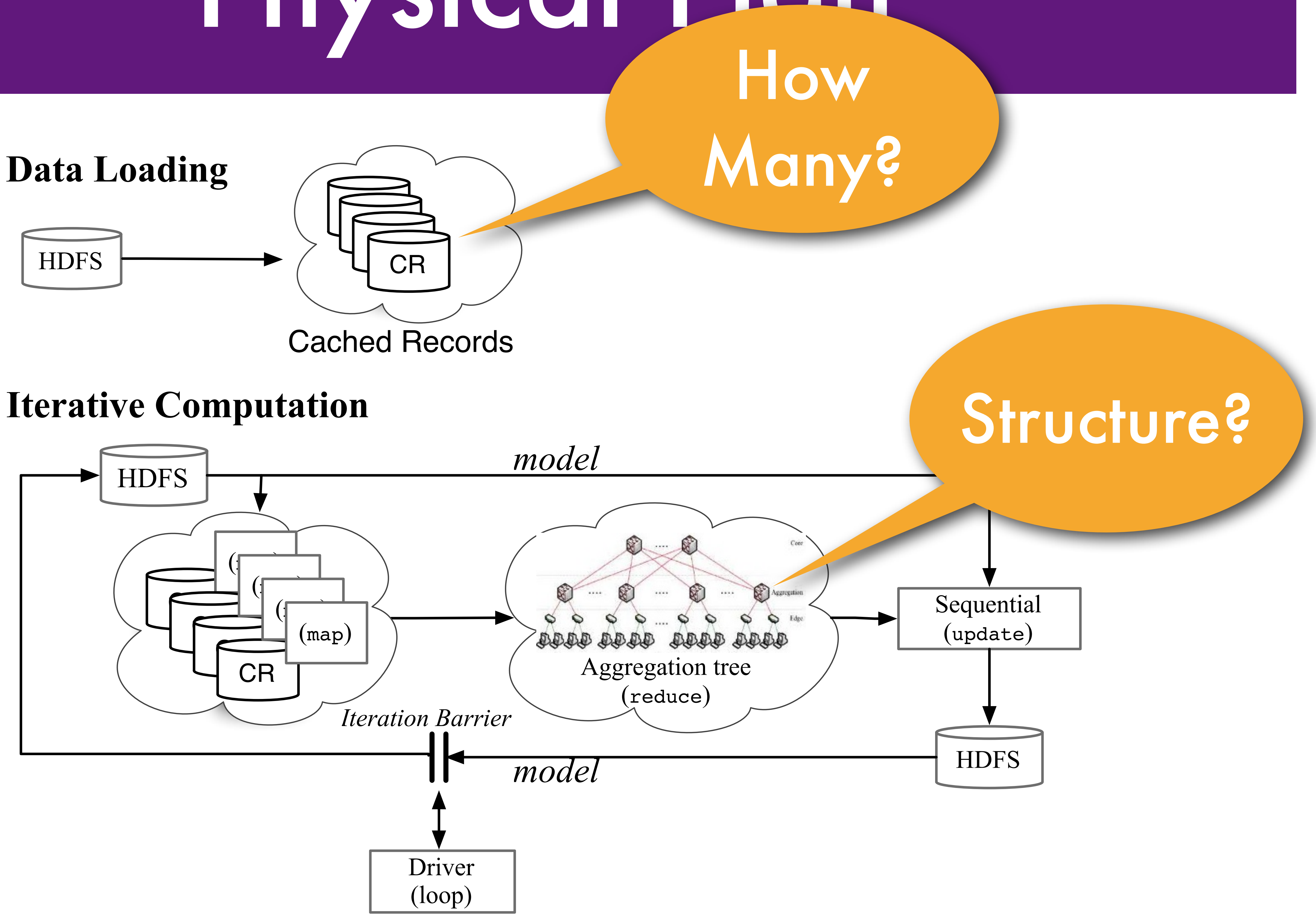
Iterative Computation



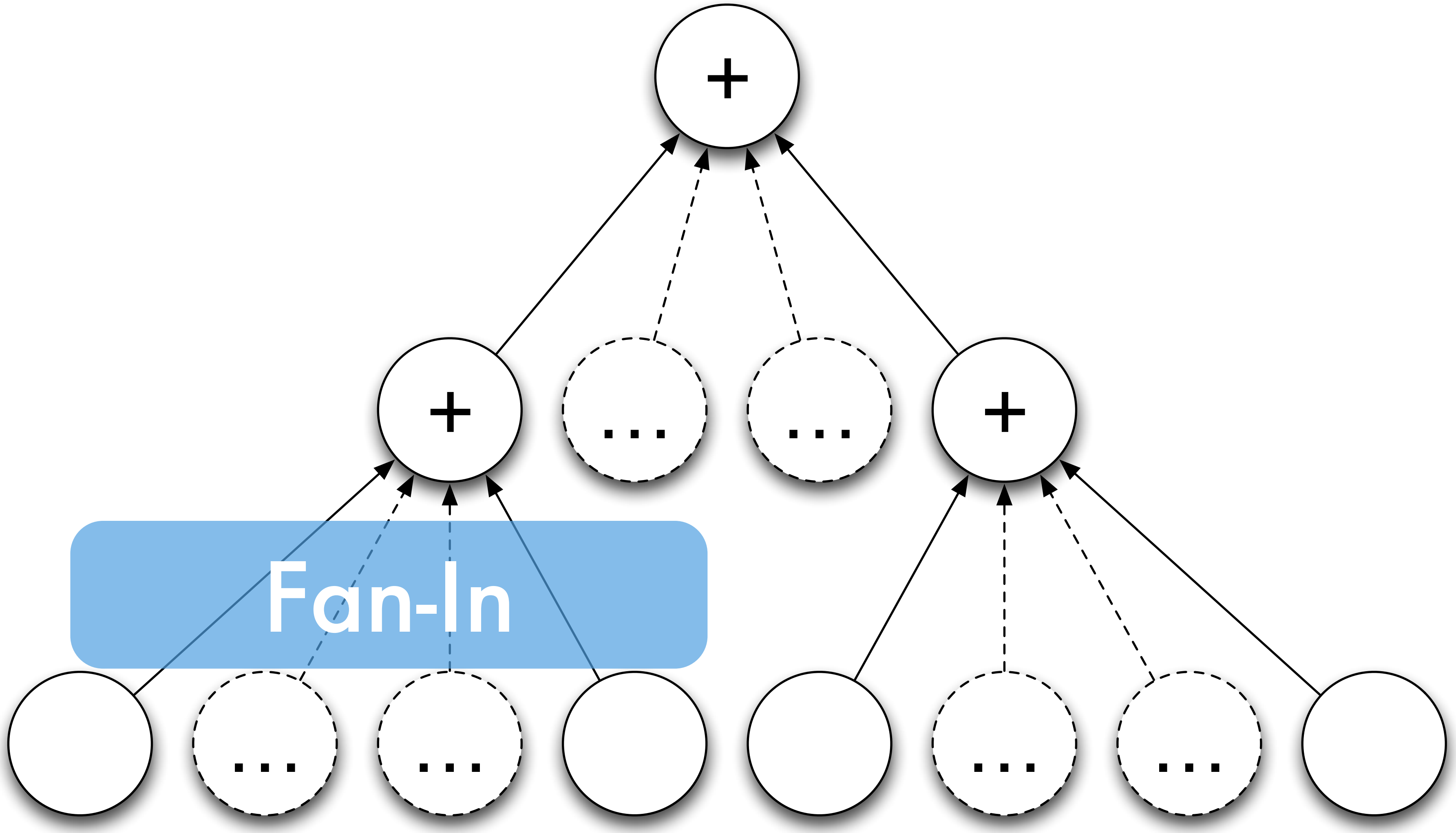
Physical Plan



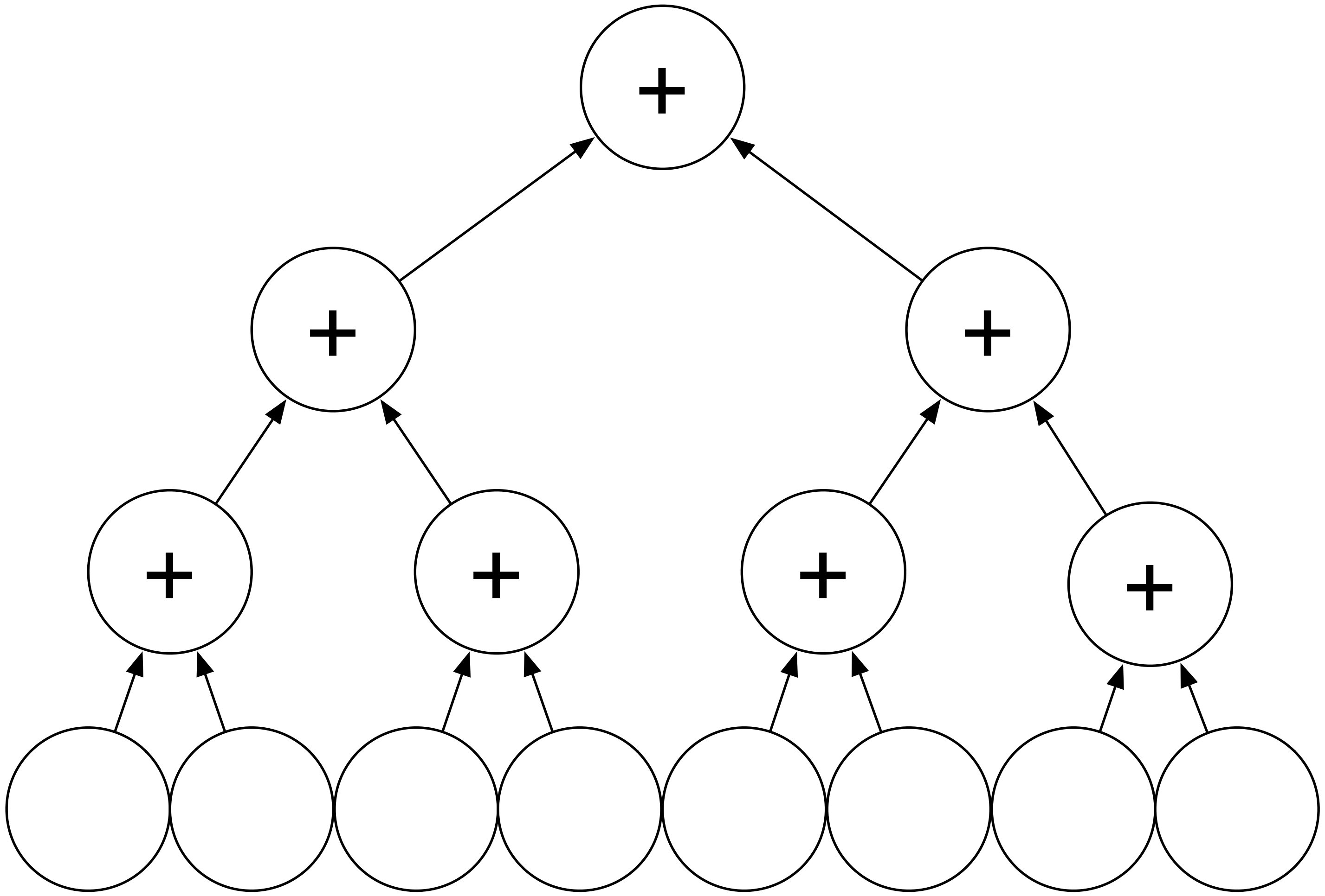
Physical Plan



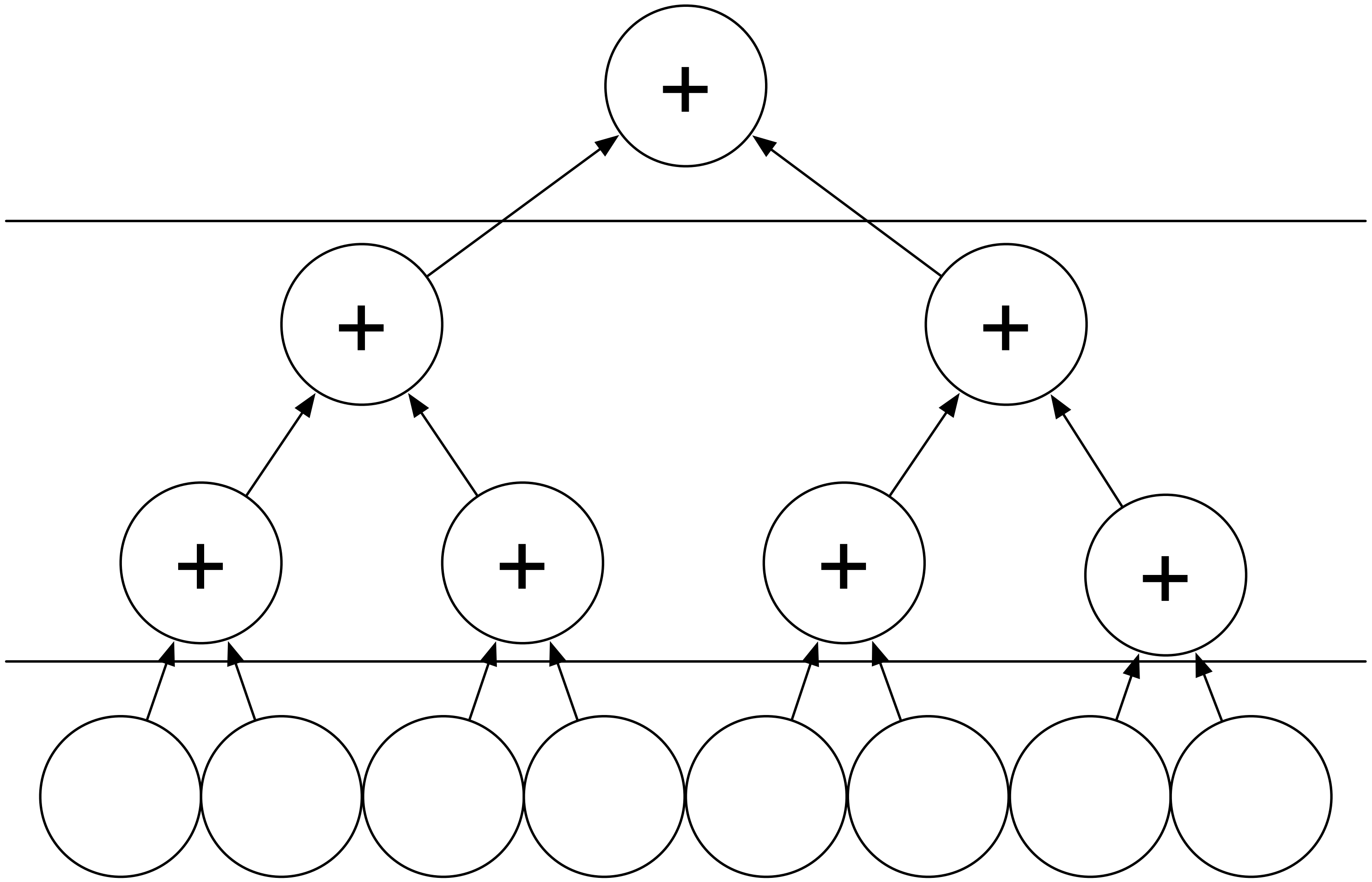
Fan-In



Fan-In

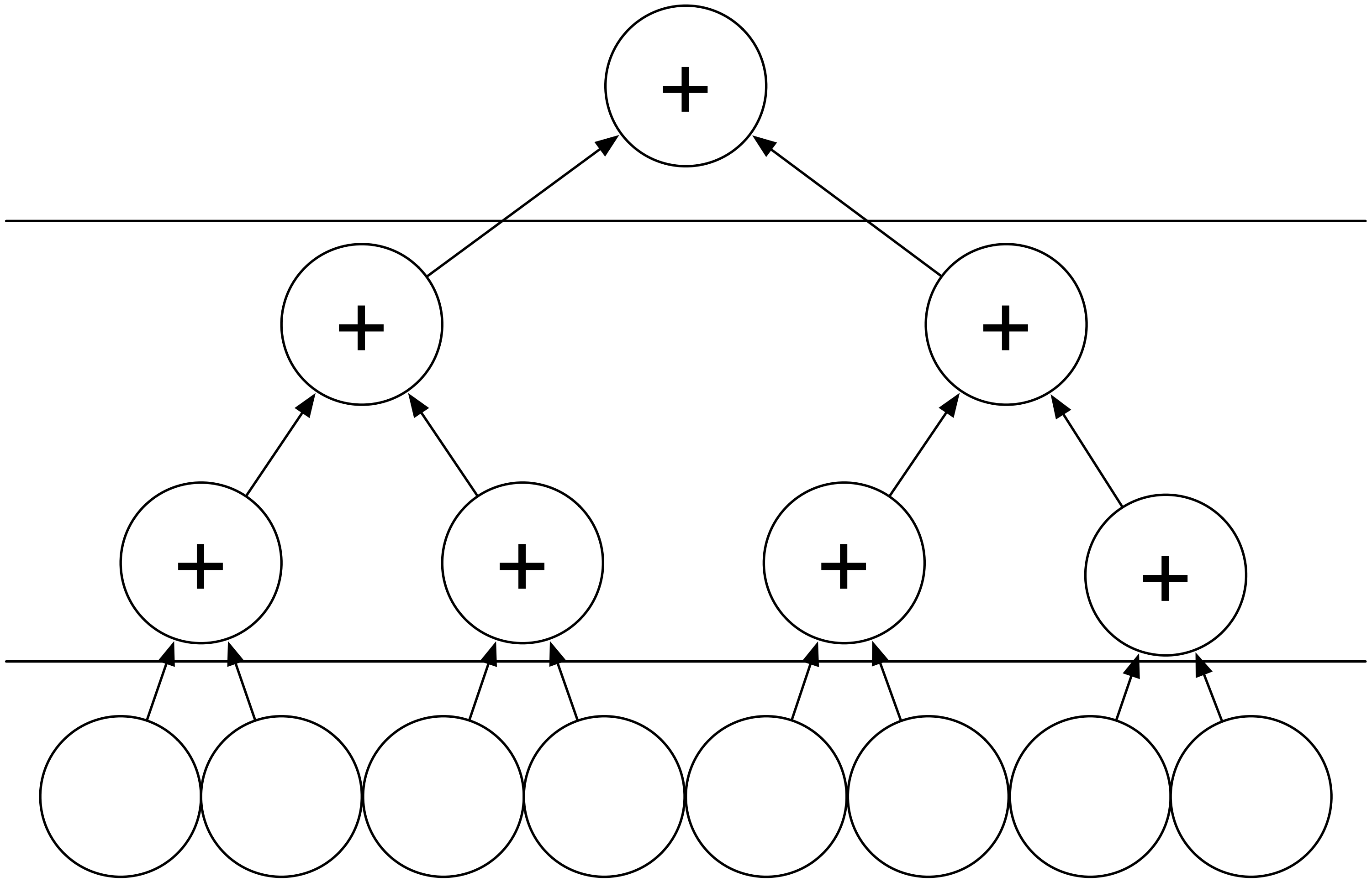


Fan-In: Blocking



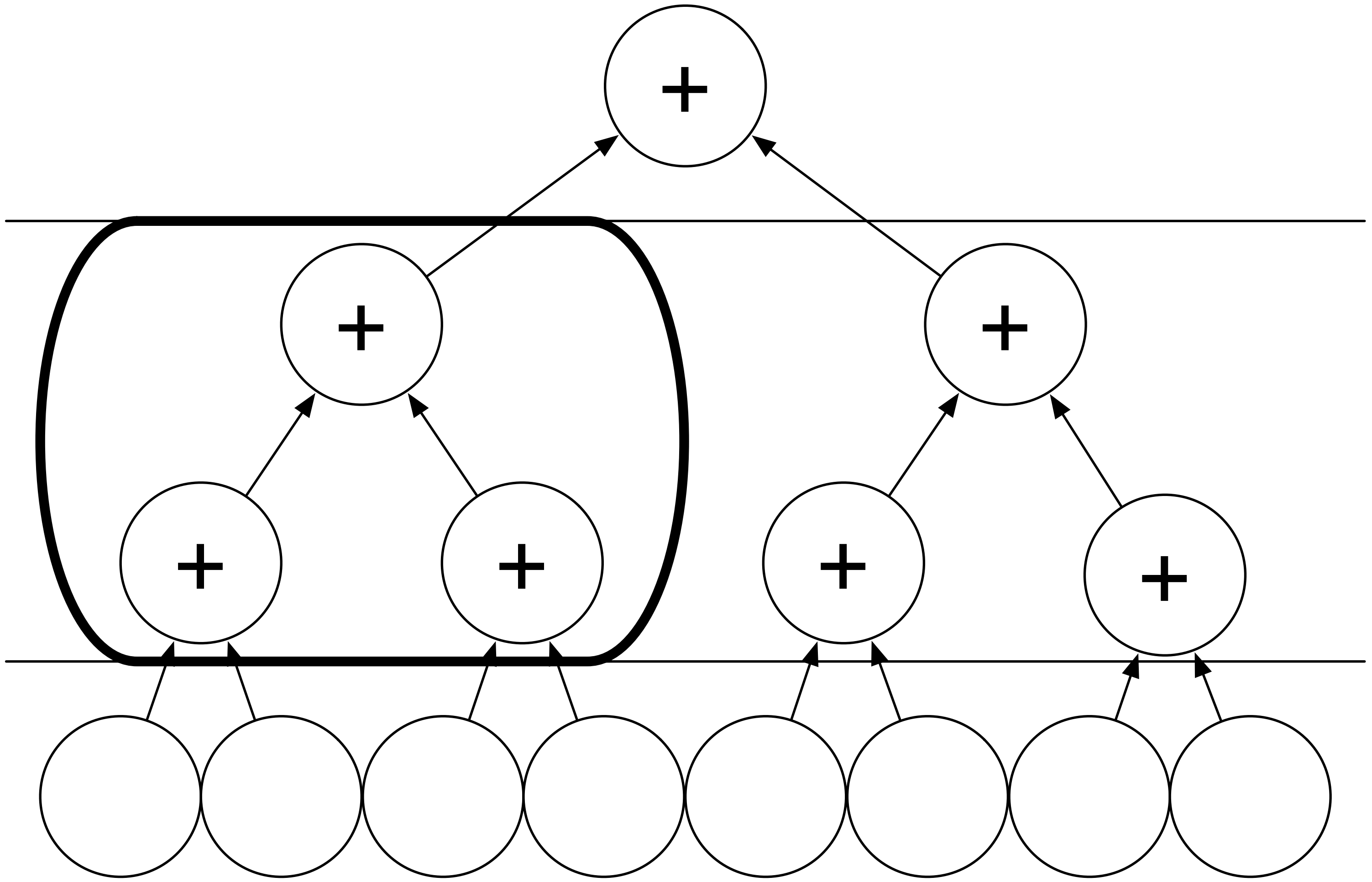
Fan-In: Blocking

$$h = \log_f(N) = \frac{\ln(N)}{\ln(f)}$$



Fan-In: Time per Level

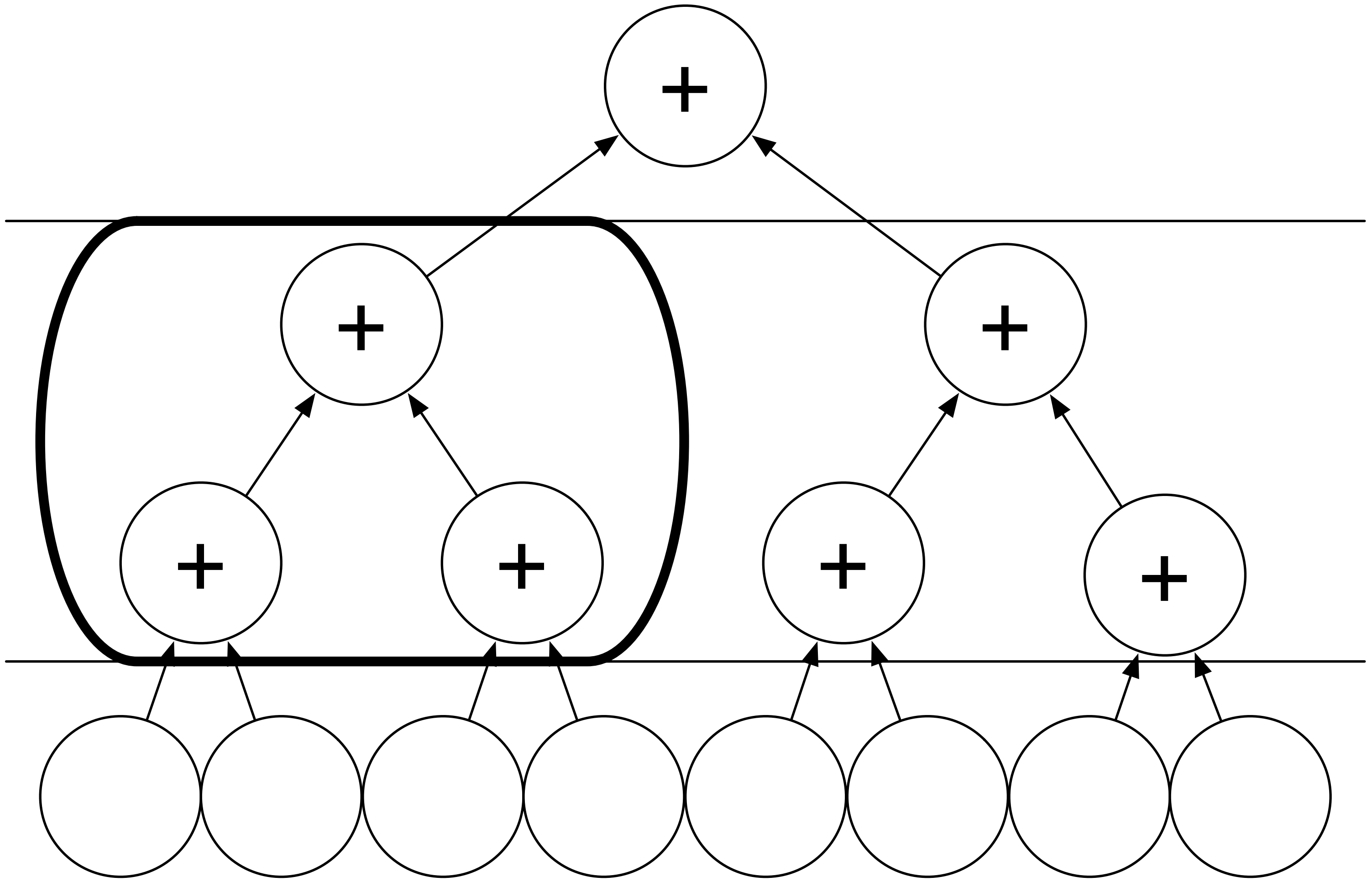
$$h = \log_f(N) = \frac{\ln(N)}{\ln(f)}$$



Fan-In: Time per Level

$t = fA$

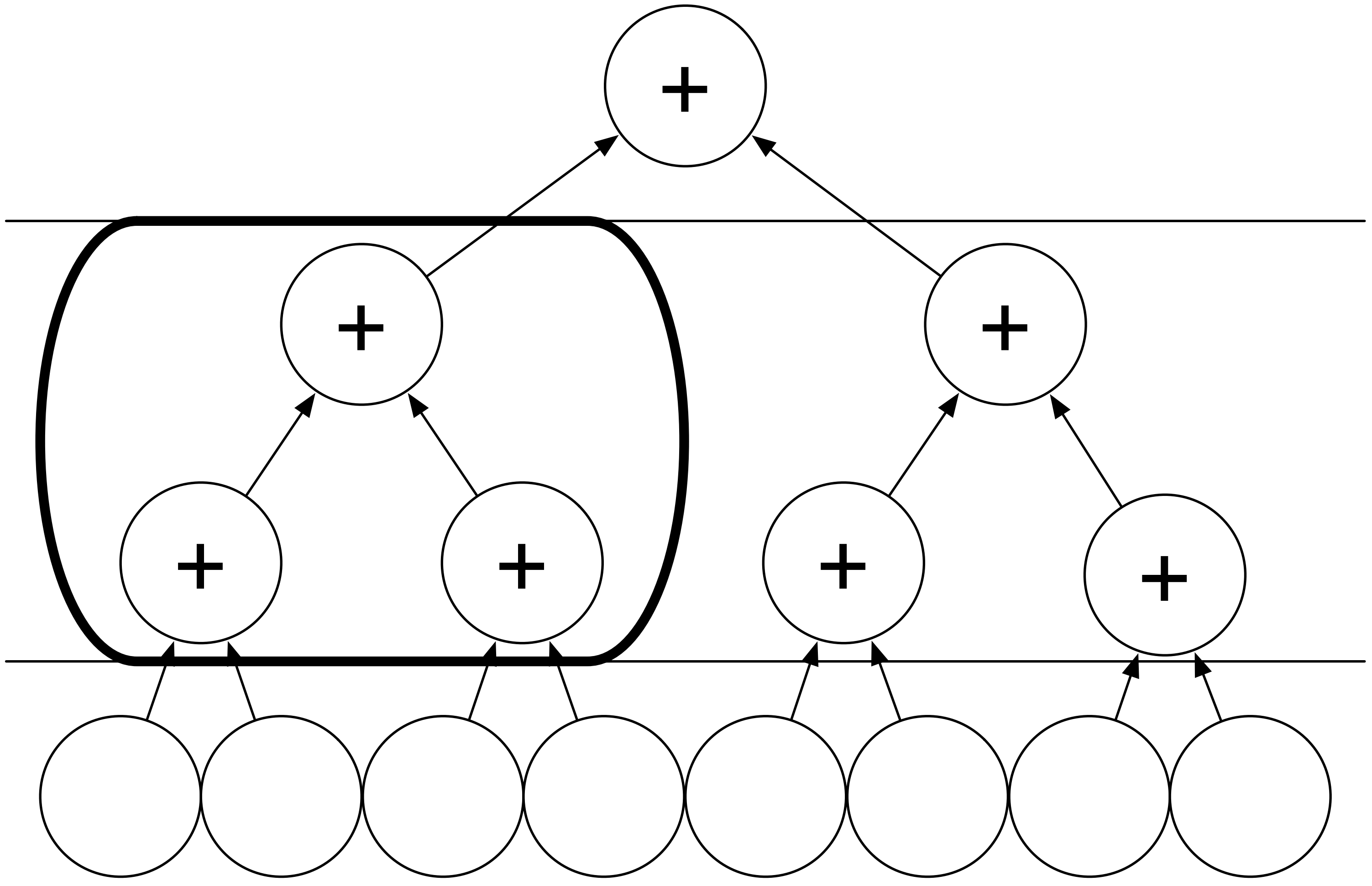
$h = \log_f(N) = \frac{\ln(N)}{\ln(f)}$



Fan-In: Total Time

$t = fA$

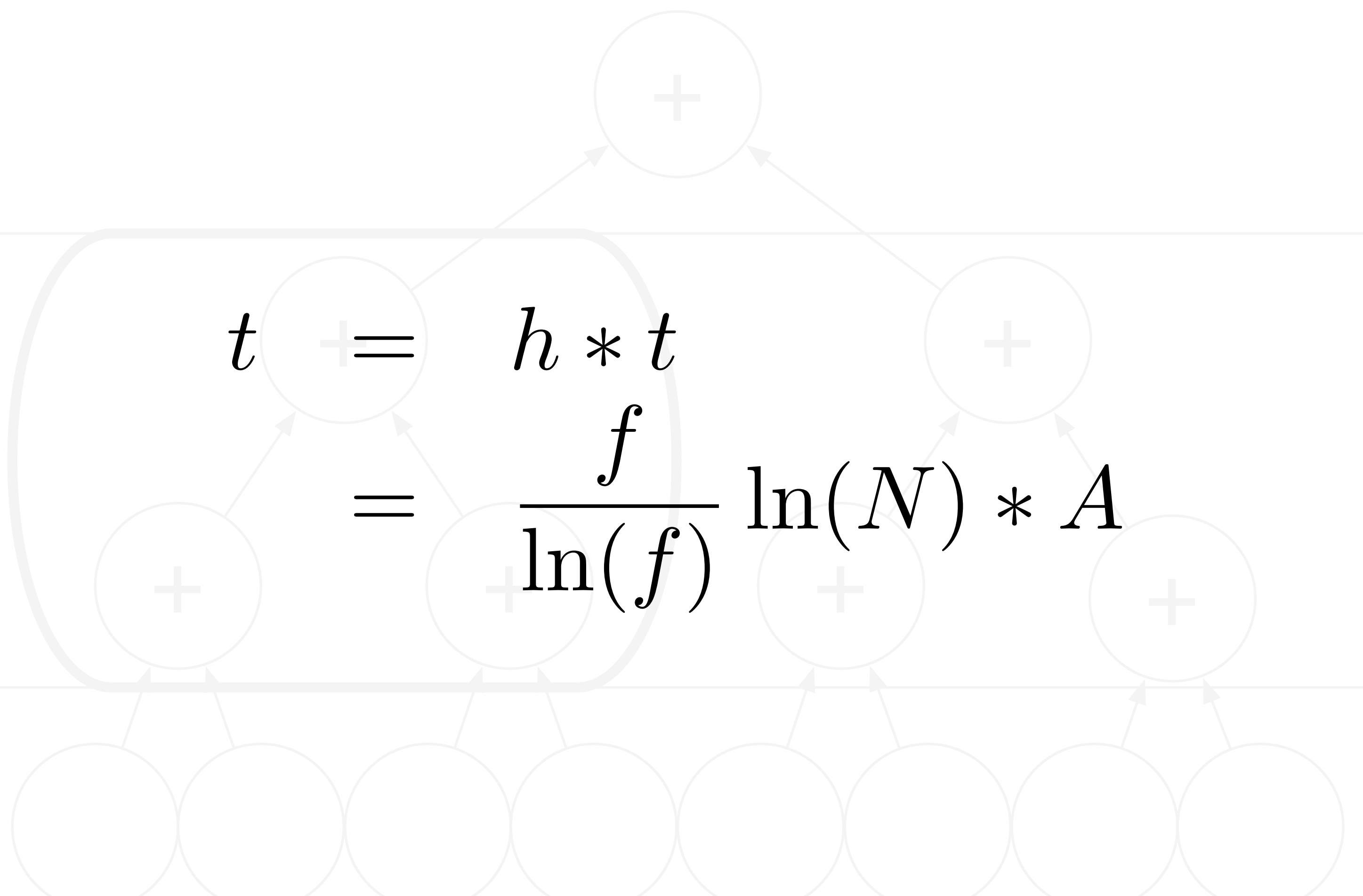
$h = \log_f(N) = \frac{\ln(N)}{\ln(f)}$



Fan-In: Total Time

$$t = fA$$

$$h = \log_f(N) = \frac{\ln(N)}{\ln(f)}$$

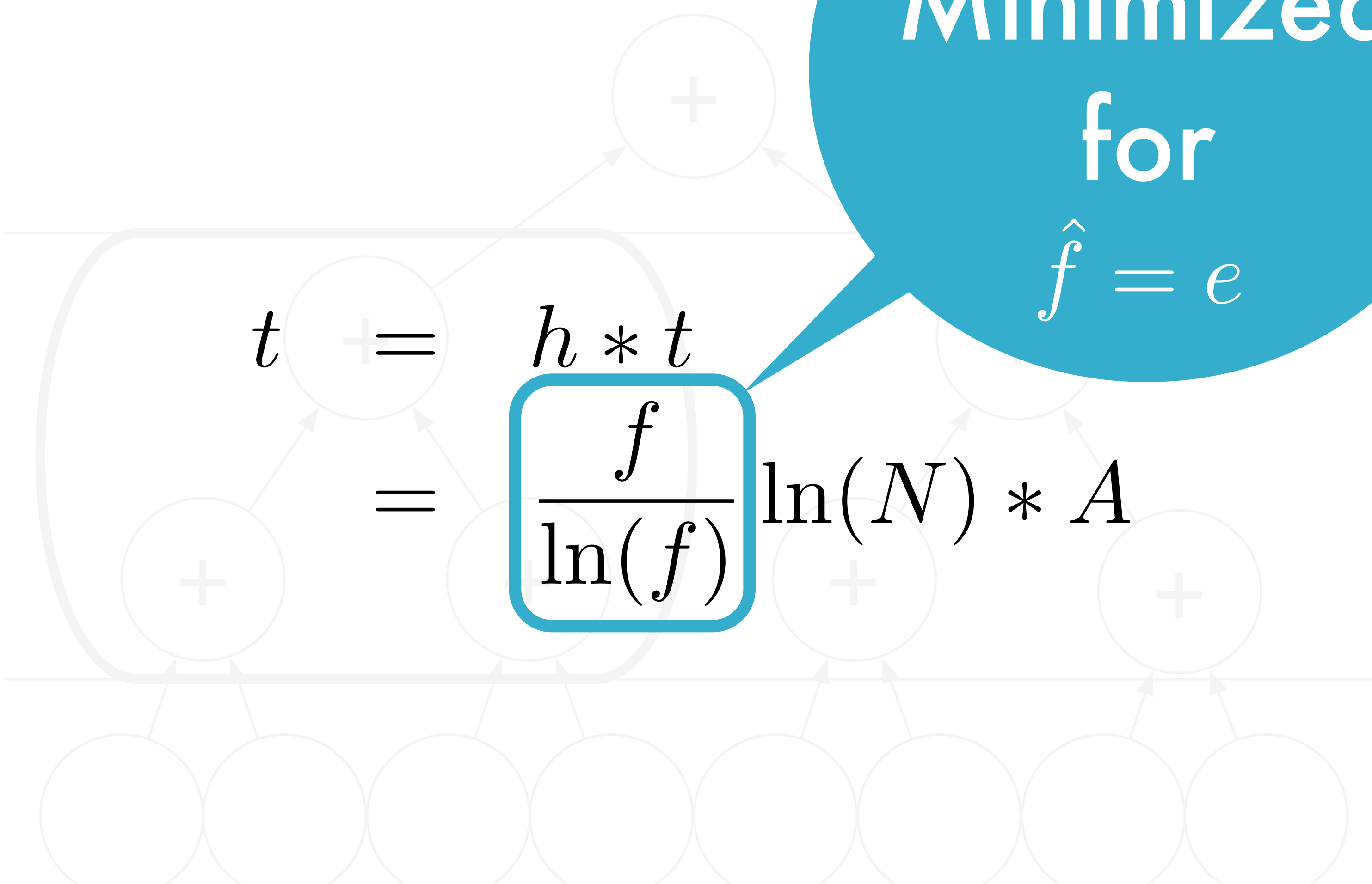


$$\begin{aligned} t &= h * t \\ &= \frac{f}{\ln(f)} \ln(N) * A \end{aligned}$$

Fan-In: Total Time

$$t = fA$$

$$) = \frac{\ln(N)}{\ln(f)}$$



$$t = h * t$$

$$= \frac{f}{\ln(f)} \ln(N) * A$$

Minimized for $\hat{f} = e$

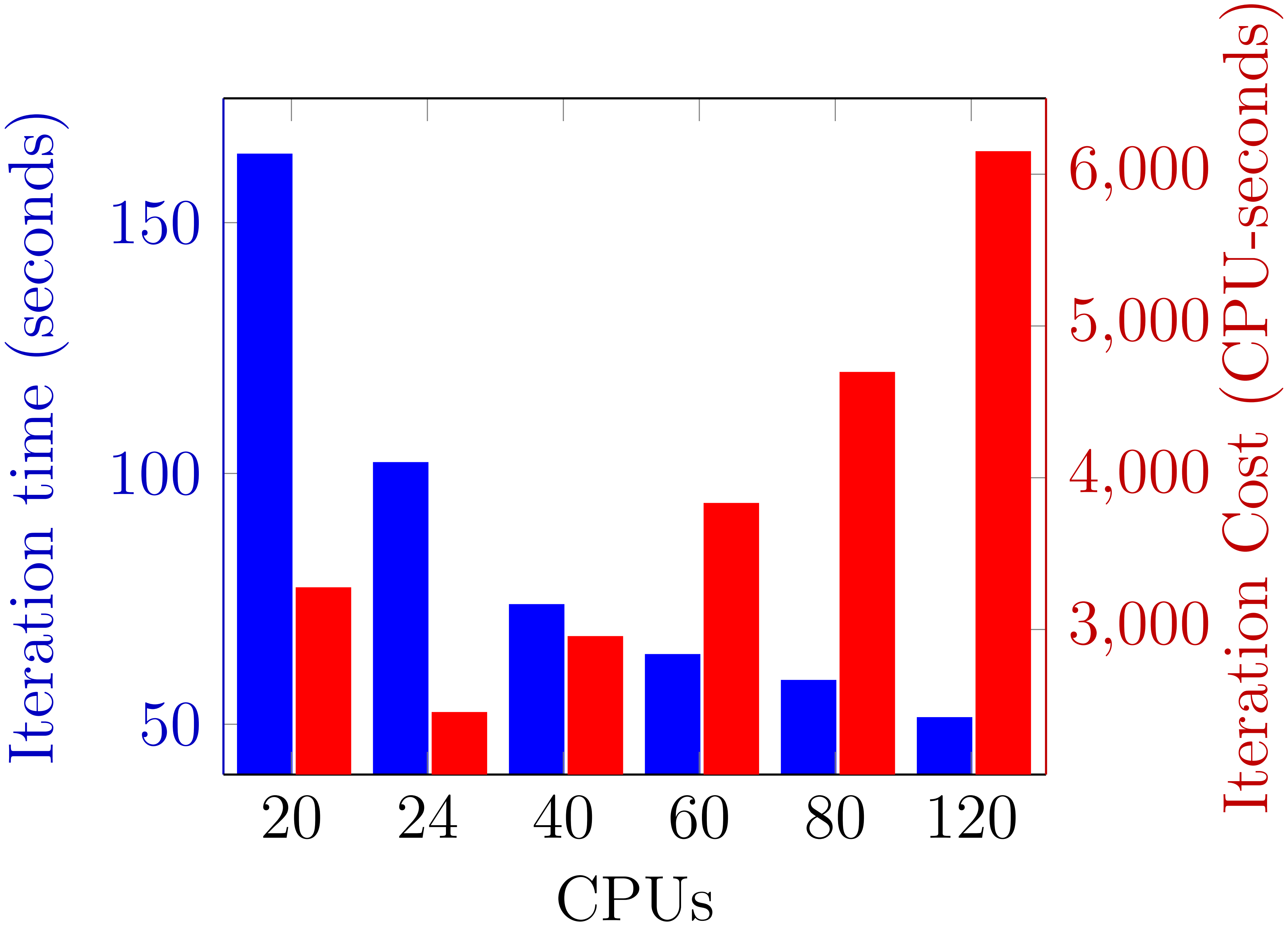
Partitioning

- Aggregation time *increases* logarithmically with number of machines
- Map time *decreases* linearly with the number of machines
- Closed form solutions available (but omitted here)

Evaluation

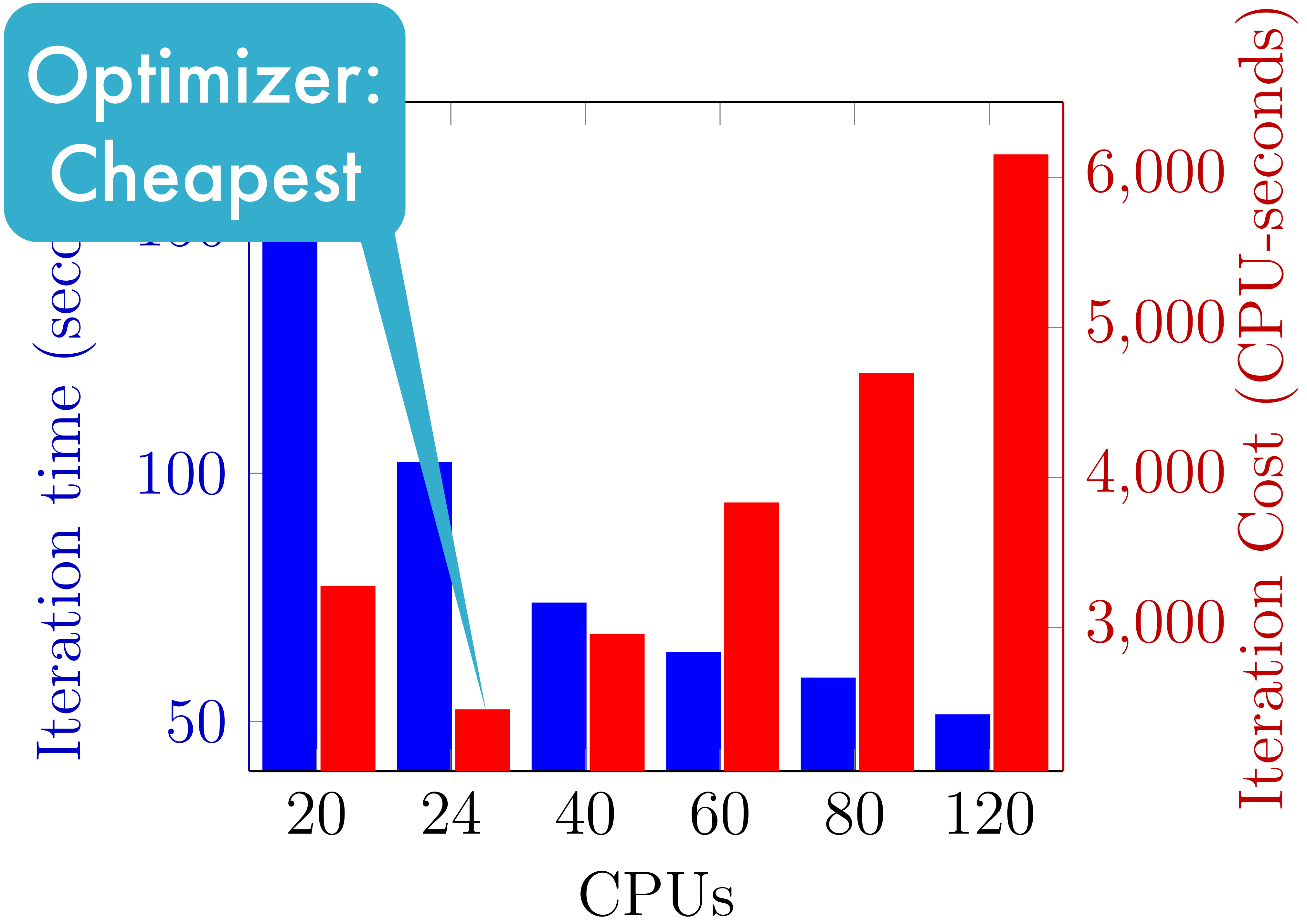
- As fast as
  - Vowpal Wabbit
  - Spark
- Faster than Hadoop (doh!)
- Much, much less code

Evaluation



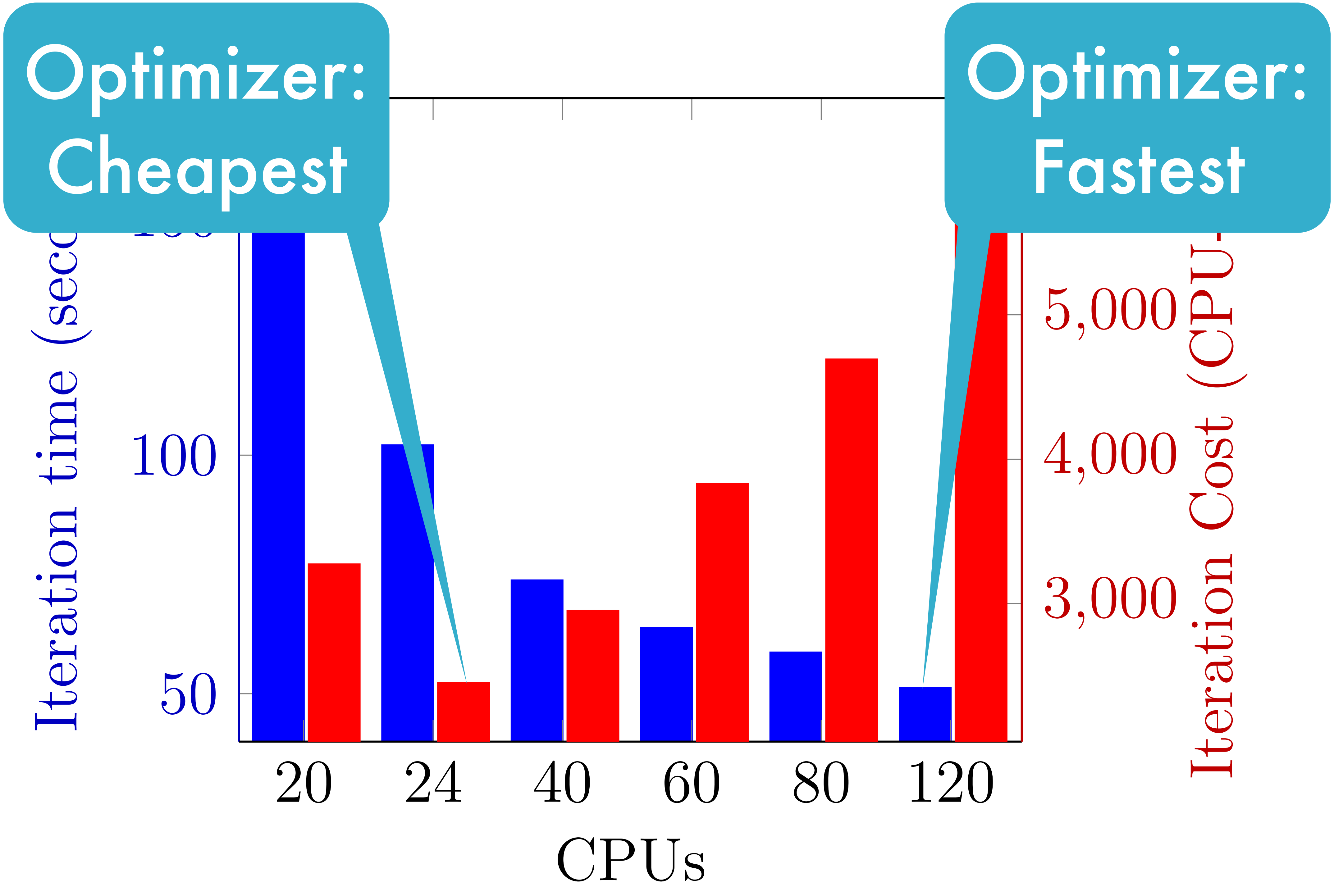
Iteration time (seconds)
150
100
50
6,000
5,000
4,000
3,000
Iteration Cost (CPU-seconds)
20
24
40
60
80
120
CPUs

Evaluation



Optimizer: Cheapest
Iteration time (seco
100
50
6,000
5,000
4,000
3,000
Iteration Cost (CPU-seconds)
20
24
40
60
80
120
CPUs

Evaluation


Optimizer: Cheapest
Optimizer: Fastest
Iteration time (seco
100
50
5,000
4,000
3,000
Iteration Cost (CPU-
20
24
40
60
80
120
CPUs

Summary

- **Example Formation**
  - Use Pig
- **Modeling**
  - Hadoop (maybe not)
  - Subsampling (now)
  - Spark / Pregel (now)
  - ScalOps (as soon as we are done)