Other Distributed Frameworks

Shannon Quinn

Distinction

1. General Compute Engines
   - Hadoop
2. User-facing APIs
   - Cascading
   - Scalding

Alternative Frameworks

1. Apache Mahout
2. Apache Giraph
3. GraphLab
4. Apache Storm
5. Apache Tez
6. Apache Flink

Alternative Frameworks

1. Apache Mahout
2. Apache Giraph
3. GraphLab
4. Apache Storm
5. Apache Tez
6. Apache Flink

Apache Mahout

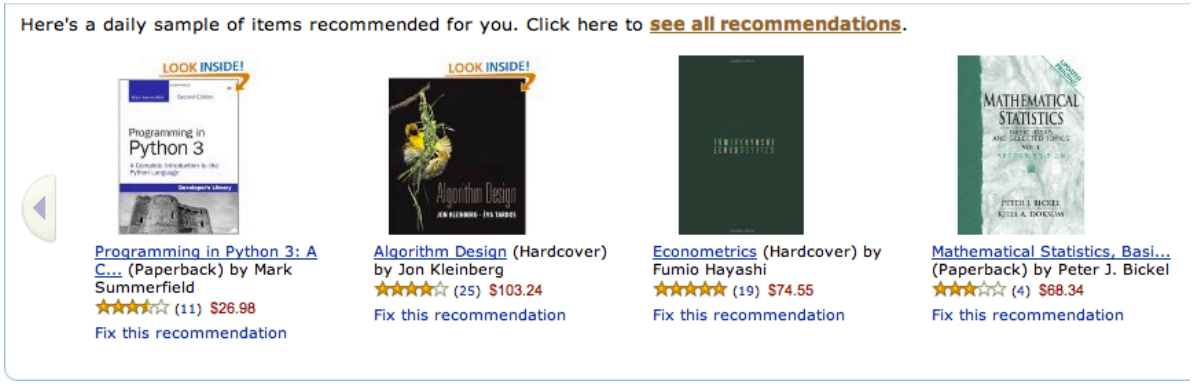
- A Tale of Two Frameworks

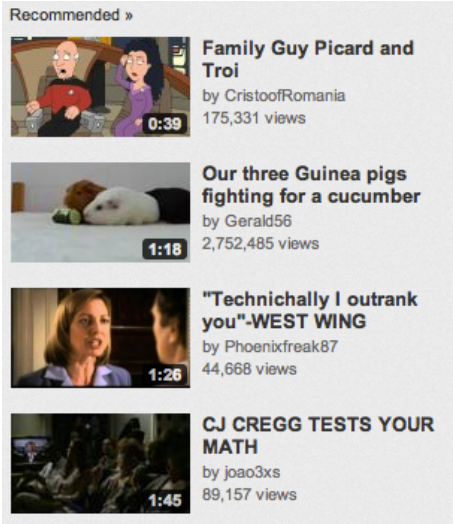
1. Distributed machine learning on Hadoop
   - 0.1 to 0.9
2. "Samsara"
   - New in 0.10+

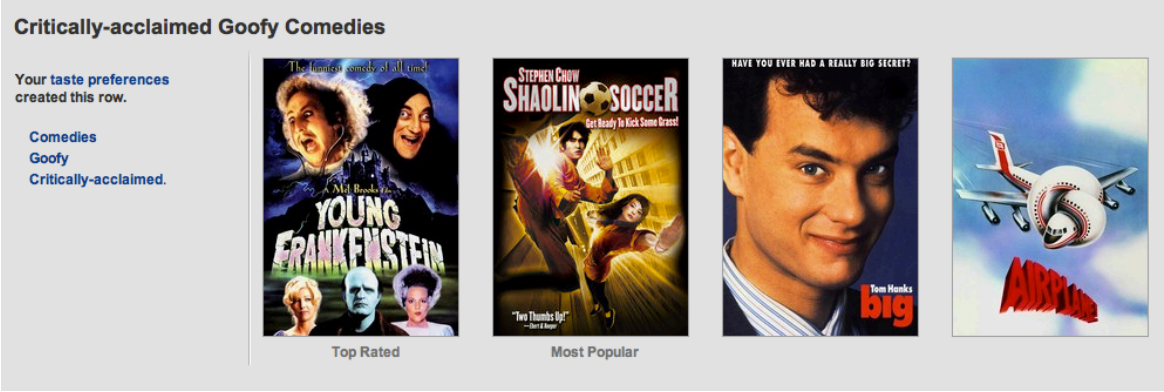
Machine learning on Hadoop

- Born out of the Apache Lucene project
- Built on Hadoop (all in Java)
- Pragmatic machine learning at scale

1: Recommendation

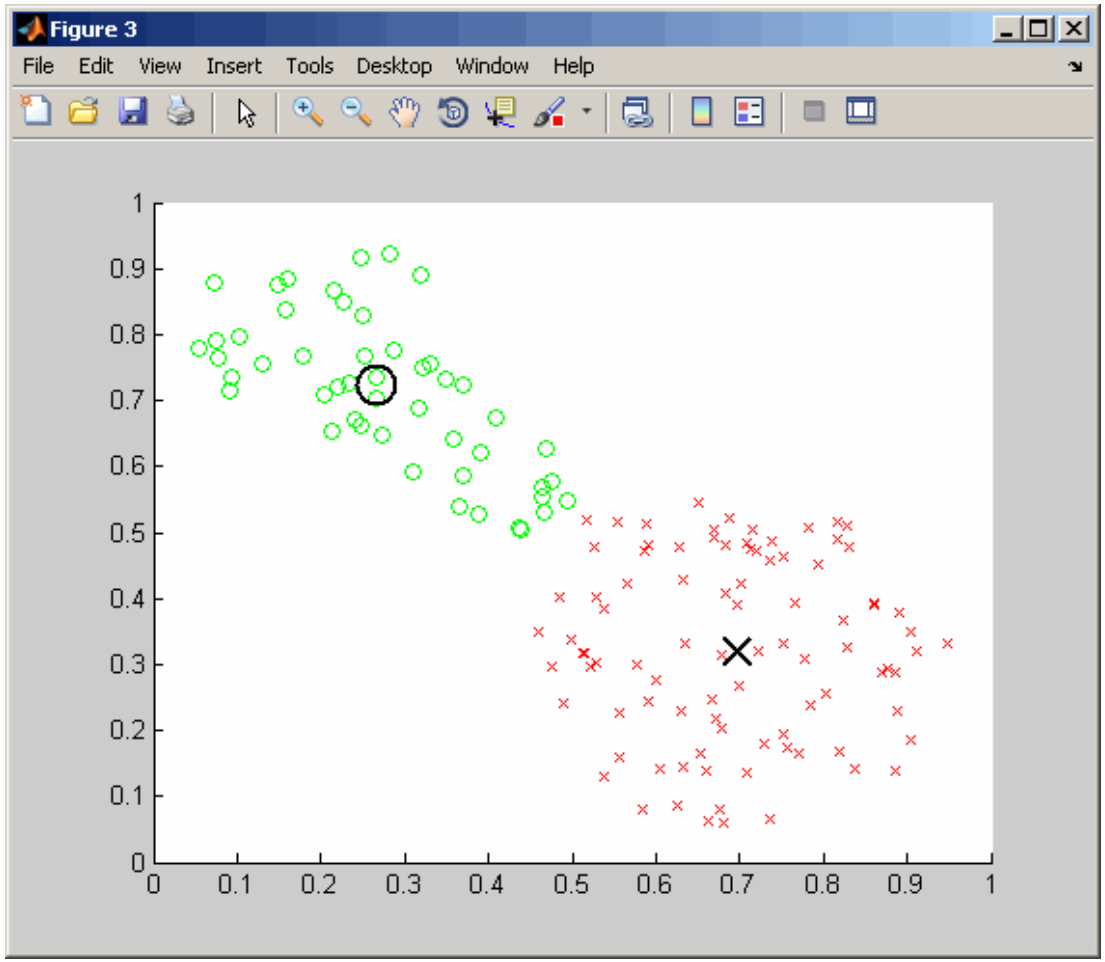






2: Classification

3: Clustering



Other MapReduce algorithms

- Dimensionality reduction
  - Lanczos
  - SSVD
  - LDA
- Regression
  - Logistic
  - Linear
  - Random Forest
- Evolutionary algorithms

Mahout-Samsara

- Programming "environment" for distributed machine learning
- R-like syntax
- Interactive shell (like Spark)
- Under-the-hood algebraic optimizer
- ***Engine-agnostic***
  - Spark
  - H2O
  - Flink
  - ?

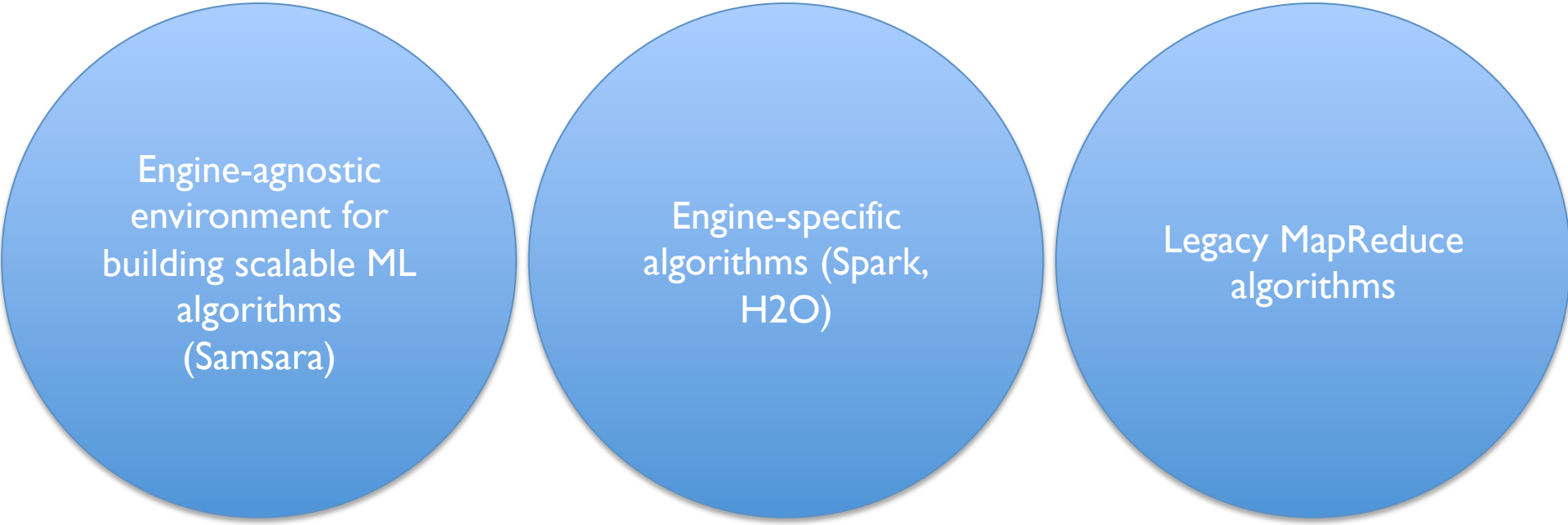
Mahout-Samsara

$$\mathbf{G} = \mathbf{B}\mathbf{B}^\top - \mathbf{C} - \mathbf{C}^\top + \mathbf{s}_q \mathbf{s}_q^\top \boldsymbol{\xi}^\top \boldsymbol{\xi}$$

```
val g = bt.t %*% bt - c - c.t + (s_q cross s_q) * (xi dot xi)
```

Mahout

- 3 main components



Mahout

- v0.10.0 released April 11 (as in, 5 days ago!)

- 0.10.1
  - More base linear algebra functionality
- 0.11.0
  - Compatible with Spark 1.3
- 1.0
  - ?

Mahout features by engine

	Single Machine	MapReduce	Spark	H2O	Flink
Mahout Math-Scala Core Library and Scala DSL					
Mahout Distributed BLAS. Distributed Row Matrix API with R and Matlab like operators. Distributed ALS, SPCA, SSVD, thin-QR. Similarity Analysis.			x	x	*in development*
Mahout Interactive Shell					
Interactive REPL shell for Spark optimized Mahout DSL			x		
Dimensionality Reduction *note: most scala-based dimensionality reduction algorithms are available through the* Mahout Math-Scala Core Library for all engines					
Singular Value Decomposition	x	x	x	x	
Lanczos Algorithm	*deprecated*	*deprecated*			
Stochastic SVD	x	x	x	x	
PCA (via Stochastic SVD)	x	x	x	x	
QR Decomposition	x	x	x	x	

Mahout features by engine

Collaborative Filtering			
User-Based Collaborative Filtering	x		x
Item-Based Collaborative Filtering	x	x	x
Matrix Factorization with ALS	x	x	
Matrix Factorization with ALS on Implicit Feedback	x	x	
Weighted Matrix Factorization, SVD++	x		
Classification			
Logistic Regression - trained via SGD	x		
Naive Bayes / Complementary Naive Bayes		x	x
Random Forest		x	
Hidden Markov Models	x		
Multilayer Perceptron	x		

Mahout features by engine

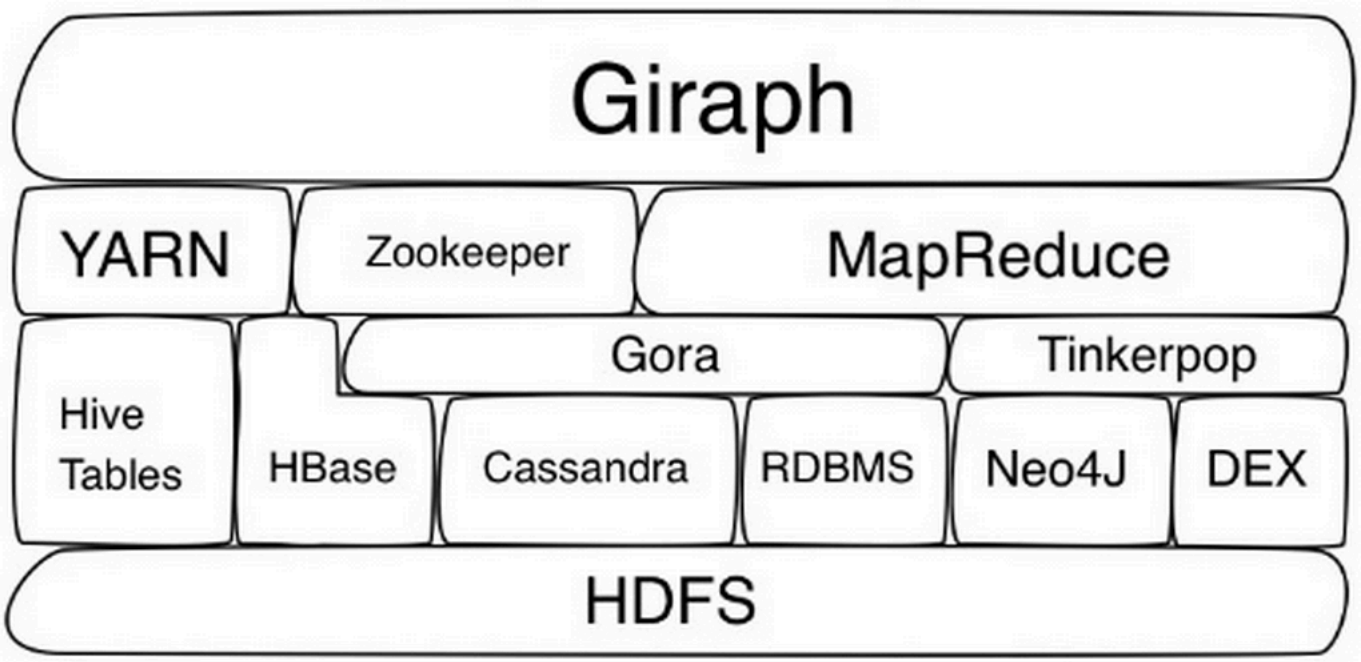
- No engine-agnostic clustering algorithms yet
  - Still the domain of legacy MapReduce
- H2O and especially Flink still highly experimental

Alternative Frameworks

1. Apache Mahout
2. Apache Giraph
3. GraphLab
4. Apache Storm
5. Apache Tez
6. Apache Flink

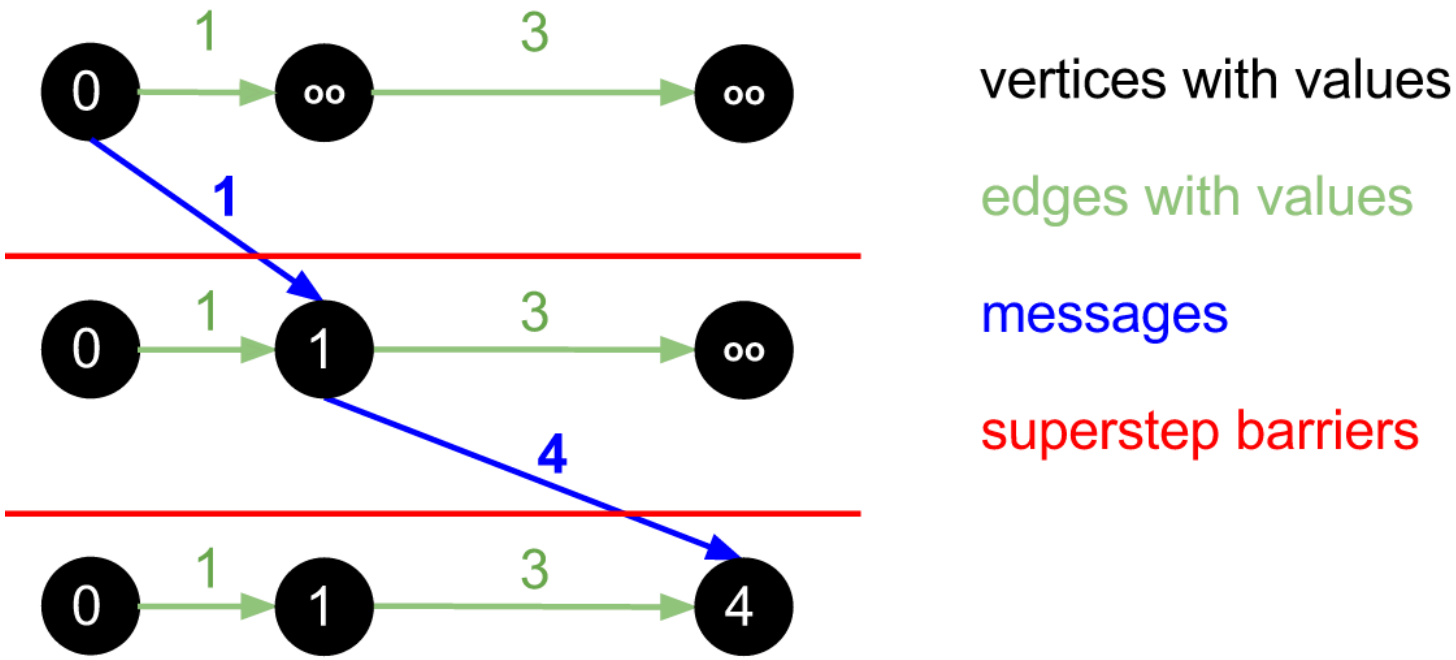
Apache Giraph

- Vertex-centric alternative to Hadoop
  - Runs on Hadoop



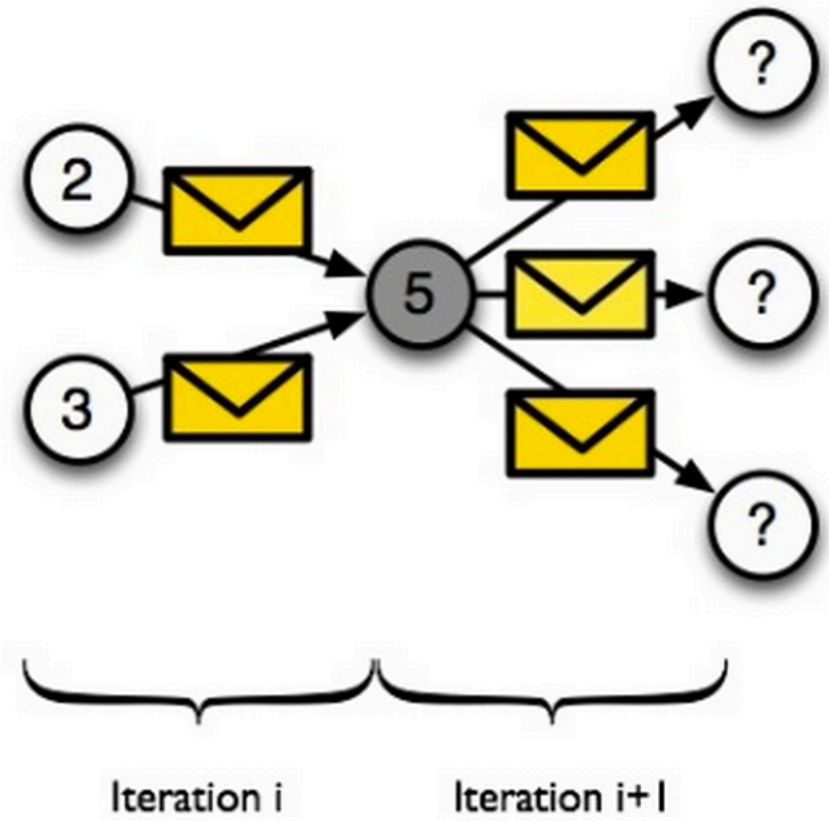
Giraph

- "...an iterative graph processing system built for high scalability."
- Bulk-synchronous Parallel (BSP) model of distributed computation



Bulk-synchronous Parallel

- Vertex-centric model



Giraph terminology

- Superstep
  - Sequence of iterations
  - Each “active” vertex invokes a compute() method
    - receives messages sent to the vertex in the previous superstep,
    - computes using the messages, and the vertex and outgoing edge values, which may result in modifications to the values, and
    - may send messages to other vertices.

Shortest path

- Example compute() method

```
public void compute(Iterable<DoubleWritable> messages) {
  double minDist = Double.MAX_VALUE;
  for (DoubleWritable message : messages) {
    minDist = Math.min(minDist, message.get());
  }
  if (minDist < getValue().get()) {
    setValue(new DoubleWritable(minDist));
    for (Edge<LongWritable, FloatWritable> edge : getEdges()) {
      double distance = minDist + edge.getValue().get();
      sendMessage(edge.getTargetVertexId(), new DoubleWritable(distance));
    }
  }
  voteToHalt();
}
```

Giraph terminology

- Barrier
  - The messages sent in any current superstep get delivered to the destination vertices only in the next superstep
  - Vertices start computing the next superstep after every vertex has completed computing the current superstep

Alternative Frameworks

1. Apache Mahout
2. Apache Giraph
3. GraphLab
4. Apache Storm
5. Apache Tez
6. Apache Flink

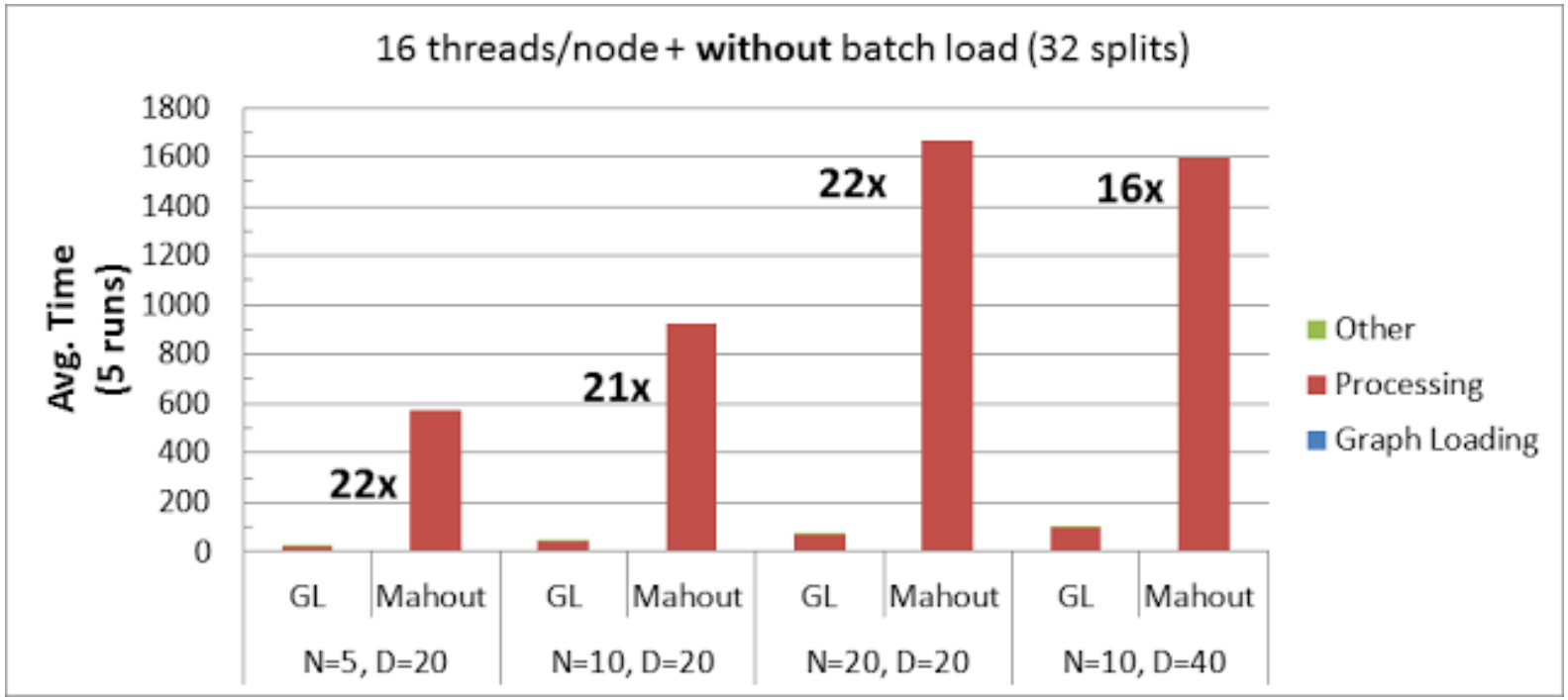
GraphLab / Dato

- Began as a PhD thesis at Carnegie Mellon University
- Like Mahout, a Tale of Two Frameworks

1. GraphLab 1.0, 2.0
   - Vertex-centric alternative to Hadoop for graph analytics (a la Apache Giraph)
2. Dato, GraphLab Create
   - ???
   - SaaS: front-facing Python API for interacting with [presumably] C++ backend on AWS

GraphLab: the early years

- Envisioned as a vertex-centric alternative to Hadoop and, in particular, Mahout
- Built in C++
- Liked to compare apples and oranges…



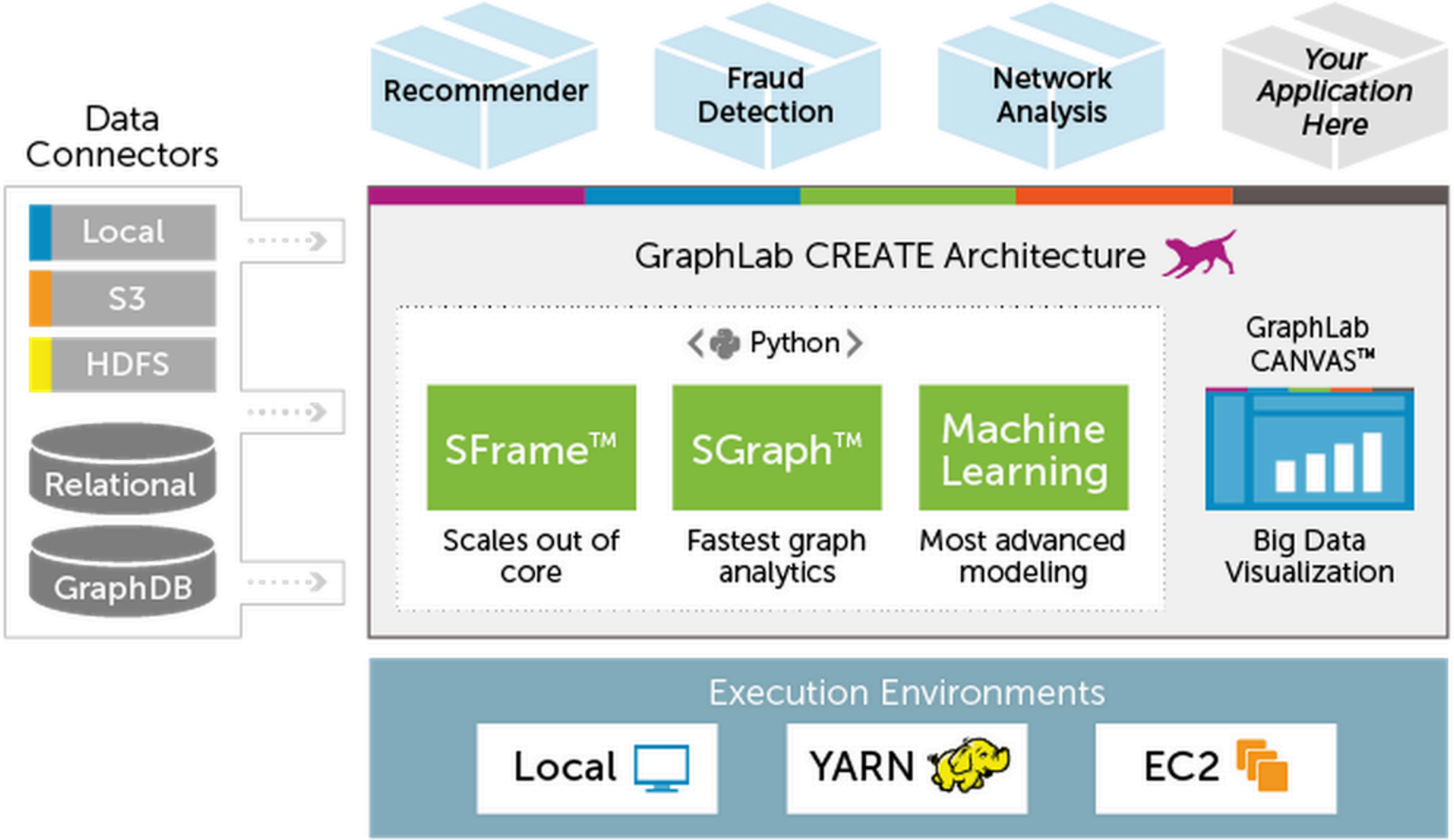
GraphLab to Dato

- Data Engineering
  - Extraction, transformation
  - Visualization
- Data Intelligence
  - Recommendation
  - Clustering
  - Classification
- Deployment
  - Creating services

Dato data structures

- SArray
  - An *immutable, homogeneously* typed array object backed by persistent storage. SArray is ***scaled to hold data that are much larger than the machine's main memory***. It fully supports missing values and random access. The data backing an SArray is located on the same machine as the GraphLab Server process. Each column in an SFrame is an SArray.
- SFrames
  - A tabular, ***column-mutable*** dataframe object that can scale to big data. The data in SFrame is stored column-wise on the GraphLab Server side, and is stored on persistent storage (e.g. disk) to avoid being constrained by memory size. ***Each column in an SFrame is a size-immutable SArray, but SFrames are mutable in that columns can be added and subtracted with ease***. An SFrame essentially acts as an ordered dict of SArrays.
- SGraph
  - A scalable graph data structure. The SGraph data structure allows arbitrary dictionary attributes on vertices and edges, provides flexible vertex and edge query functions, and seamless transformation to and from SFrame.

Architecture



GraphLab Create

- "Five-line recommender"

```
import graphlab as gl
url = 'http://s3.amazonaws.com/dato-datasets/movie_ratings/training_data.csv'
data = gl.SFrame.read_csv(url, column_type_hints={"rating":int})
data.show()
model = gl.recommender.create(data, user_id="user", item_id="movie", target="rating")
results = model.recommend(users=None, k=5)
```

user	movie	score	rank
Marco Smith	Sex and the City: Season 6: Part 2	5.53567966735	1
Marco Smith	Sex and the City: Season 4	5.21305689132	2
Marco Smith	Sex and the City: Season 5	5.1795544318	3
Marco Smith	Sex and the City: Season 6: Part 1	5.15331420219	4
Marco Smith	Friends: Season 4	5.01739624059	5
Zion Smith	High Noon	3.73848405397	1
Zion Smith	The Treasure of the Sierra Madre	3.5382391355	2
Zion Smith	Dial M for Murder	3.44573309934	3
Zion Smith	Band of Brothers	3.43282082593	4
Zion Smith	The Dish	3.40498876488	5

Alternative Frameworks

1. Apache Mahout
2. Apache Giraph
3. GraphLab
4. Apache Storm
5. Apache Tez
6. Apache Flink

Apache Storm

- "…doing for realtime processing what Hadoop did for batch processing."
- Distributed realtime computation system
- Reliably process unbounded streams of data
- Common use cases
  - Realtime analytics
  - Online learning
  - Distributed RPC
  - [Your use case here]

Powered by Storm

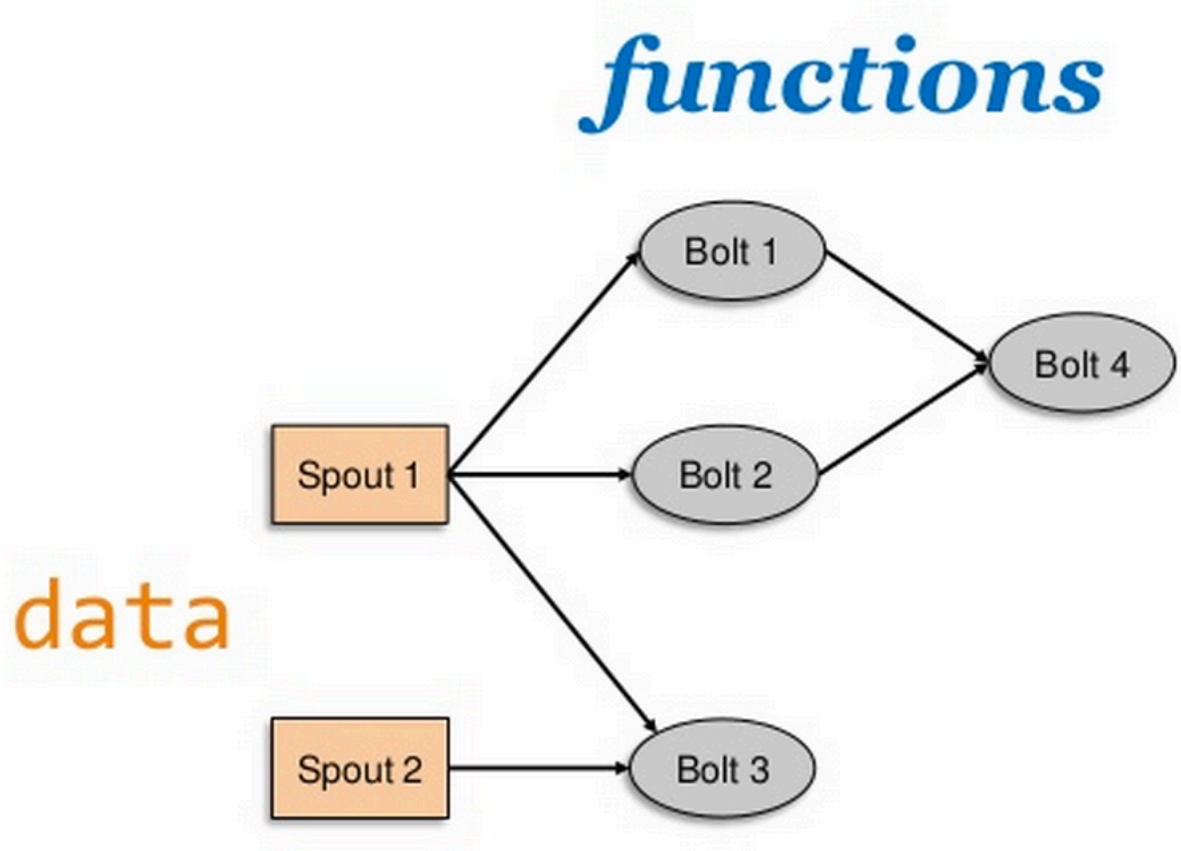
and many others

Storm terminology

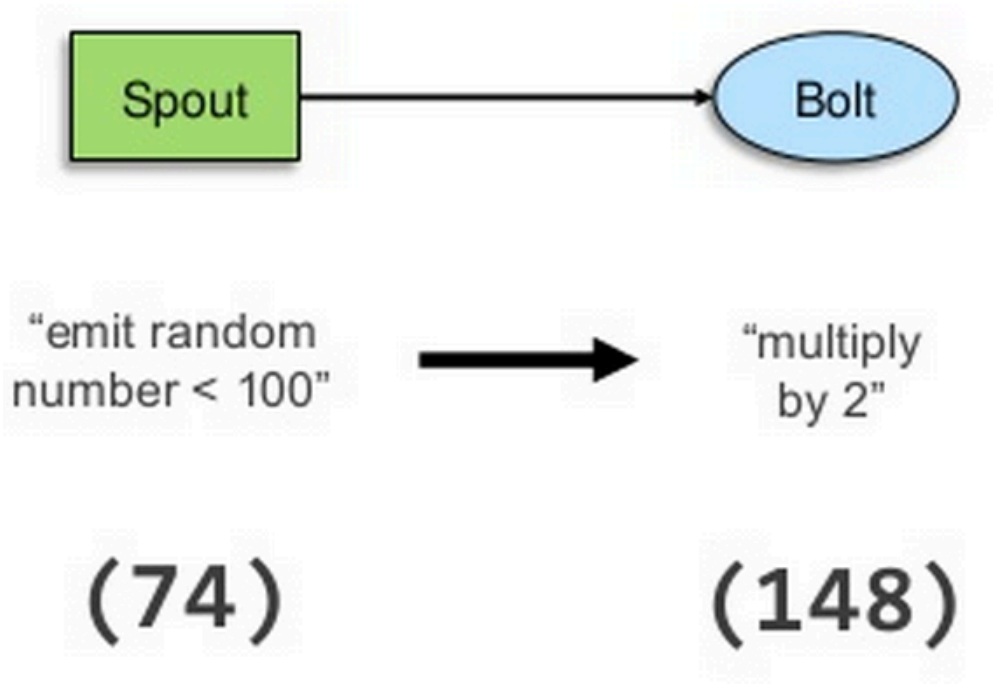
- Spouts
  - Source of streaming data
  - Kestrel, RabbitMQ, Kafka, JMS, databases (brokers)
  - Twitter Streaming API
- Bolts
  - Processes input streams to produce output streams
  - Functions, filters, joins, aggregations
- Topologies
  - Network of sprouts (vertices) and bolts (edges)
  - Arbitrarily complex multi-stage streaming operation
  - Run indefinitely once deployed

Storm

Topology



Storm



Storm

Spout

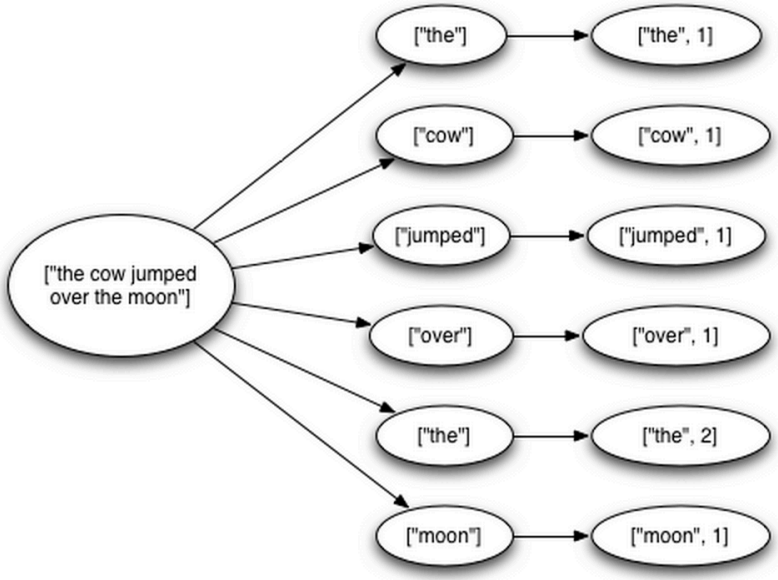
```
1  public void nextTuple() {
2      final Random rand = new Random(); // normally this should be an instance field
3      int nextRandomNumber = rand.nextInt(100);
4      collector.emit(new Values(nextRandomNumber)); // auto-boxing
5  }
```

Bolt

```
1   Override
2   public void prepare(Map conf, TopologyContext context, OutputCollector collector) {
3       this.collector = collector;
4   }
5
6   @Override
7   public void execute(Tuple tuple) {
8       Integer inputNumber = tuple.getInteger(0);
9       collector.emit(tuple, new Values(inputNumber * 2)); // auto-boxing
10      collector.ack(tuple);
11  }
12
13  @Override
14  public void declareOutputFields(OutputFieldsDeclarer declarer) {
15      declarer.declare(new Fields("doubled-number"));
16  }
```

Performance

- 1M 100-byte messages per second per node
- Storm automatically restarts workers that fail
  - Workers which cannot be restarted on the original node are restarted on different nodes
  - Nimbus and Supervisor
- Guarantees each tuple will be fully processed



Highly configurable

- Usable with [virtually] any language
- Thrift definition for defining topologies
  - Thrift is language-agnostic, so topologies are as well
- So are spouts and bolts!
  - Non-JVM languages communicate over JSON protocols
  - Adapters available for Ruby, Python, JavaScript, and Perl

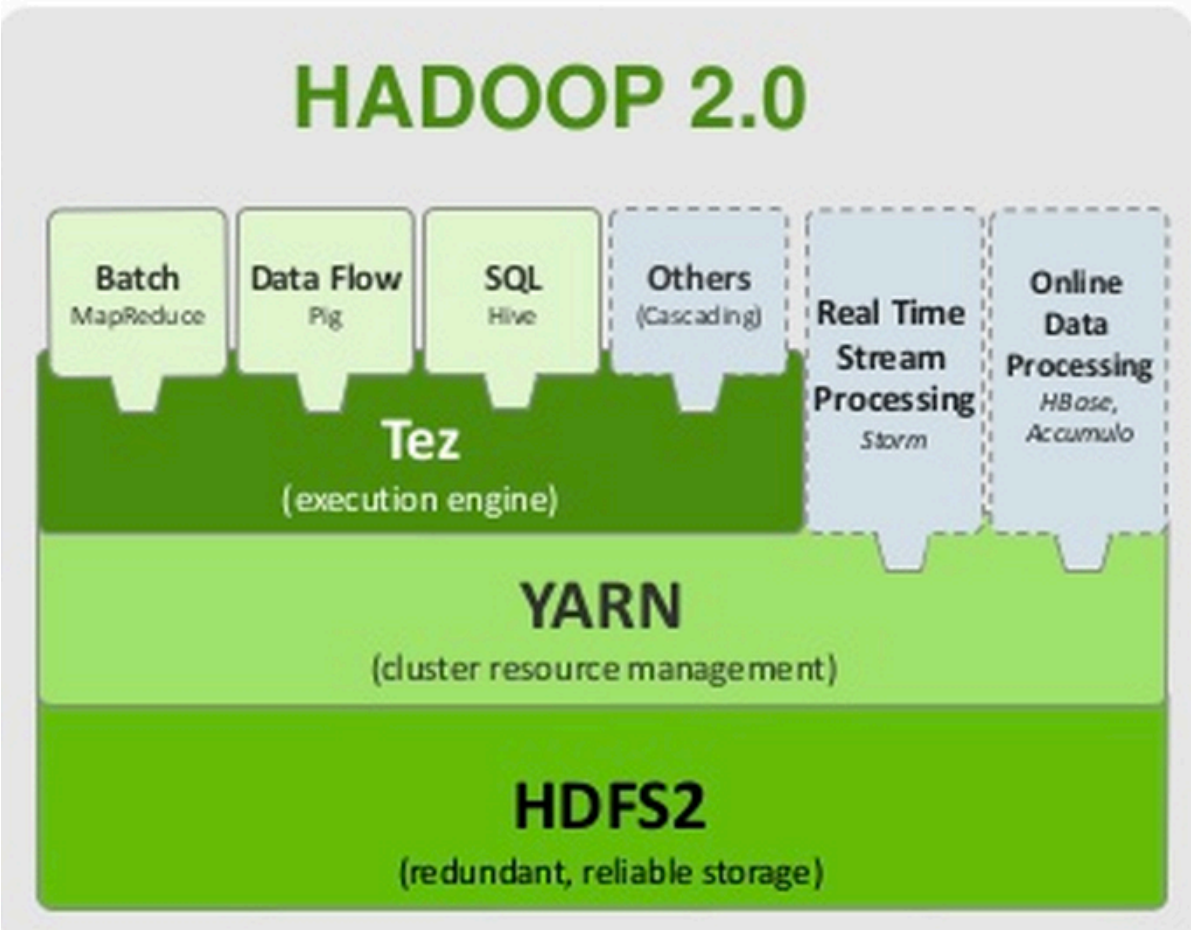
Alternative Frameworks

1. Apache Mahout
2. Apache Giraph
3. GraphLab
4. Apache Storm
5. Apache Tez
6. Apache Flink

Apache Tez

- "…aimed at building an application framework which allows for a complex directed-acyclic-graph [DAG] of tasks for processing data."
- Distributed execution framework
- Express computation as a data flow graph
- Built on Hadoop's YARN

The software stack



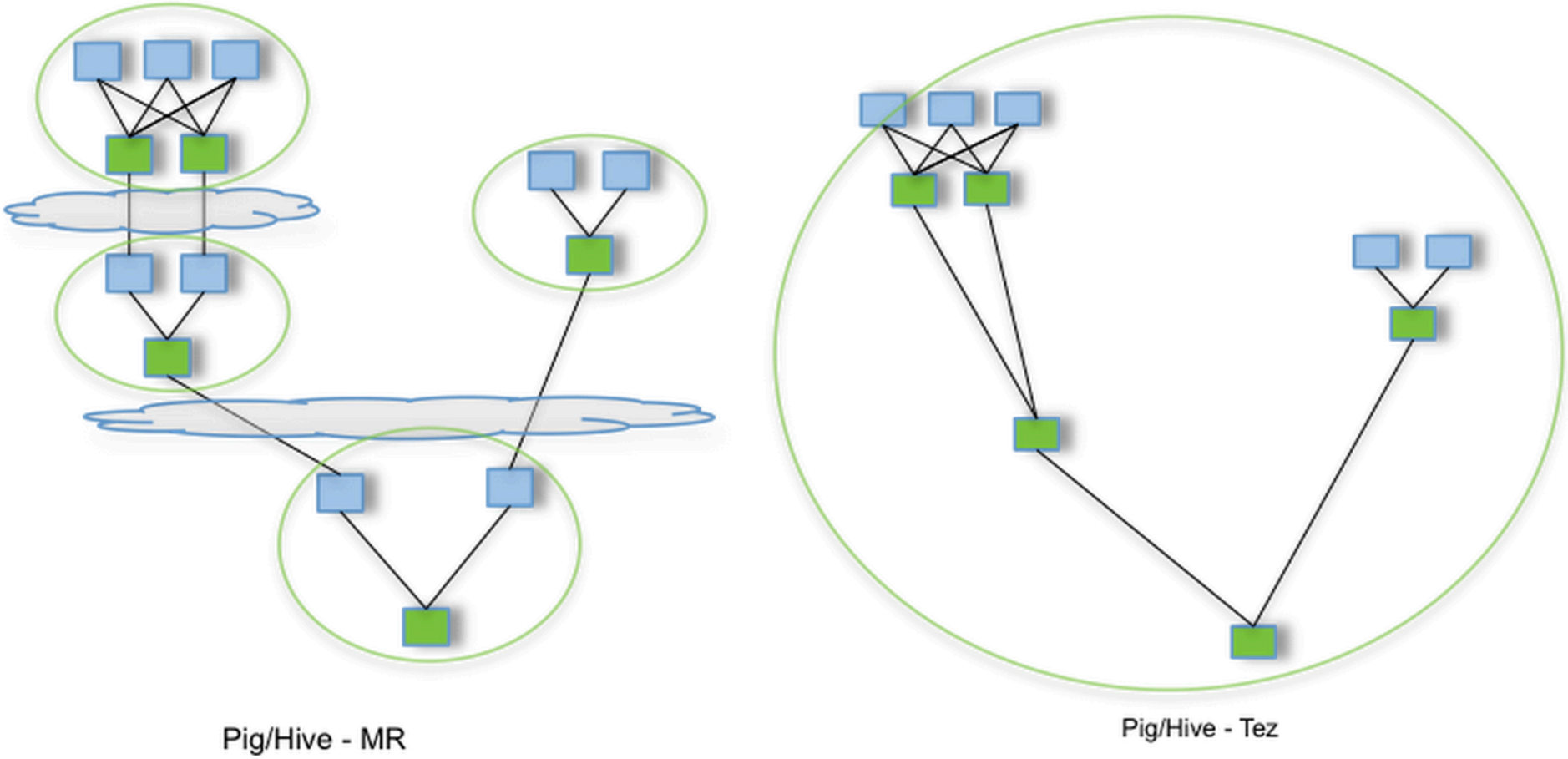
Apache Tez

- Separates application logic from parallel execution, resource allocation, and fault tolerance

App	• Custom application logic • Custom data format • Custom data transfer technology
Tez	• Distributed parallel execution • Negotiating resources from the Hadoop framework • Fault tolerance and recovery • Horizontal scalability • Resource elasticity • Shared library of ready-to-use components • Built-in performance optimizations • Security

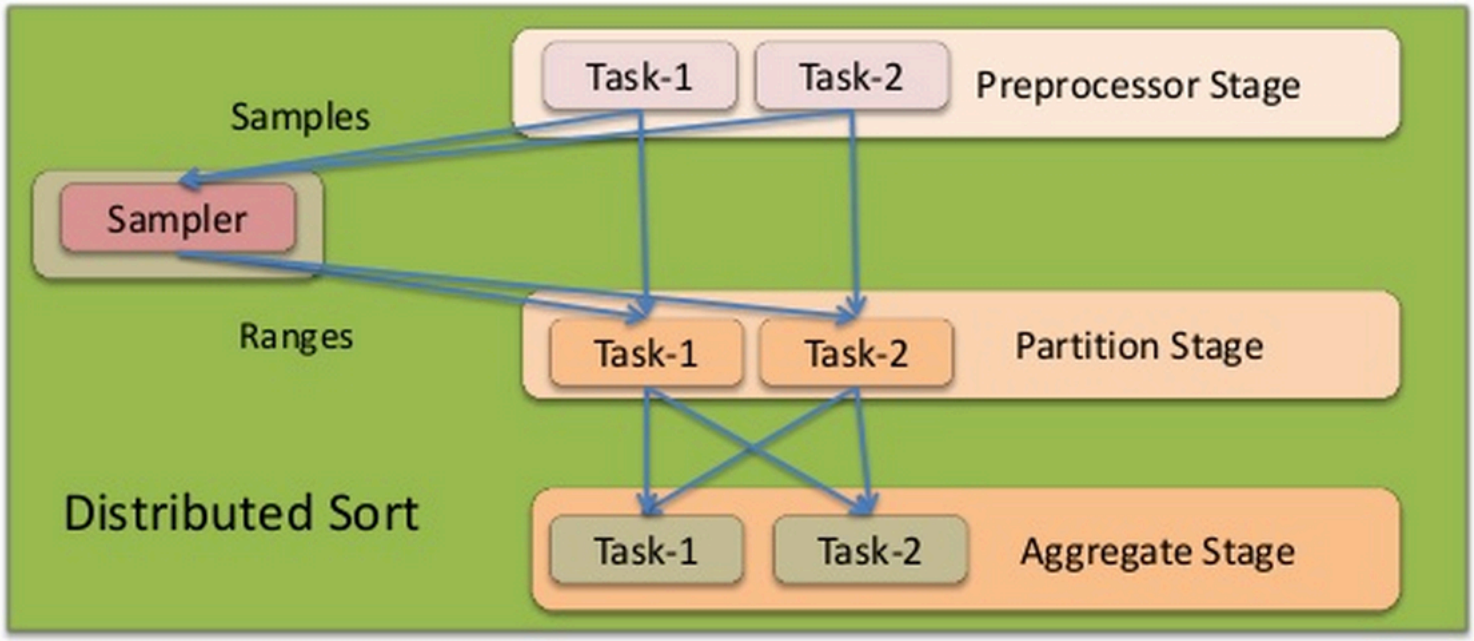
Workflow optimization

- Workflows that previous required multiple MR passes can be done in only one



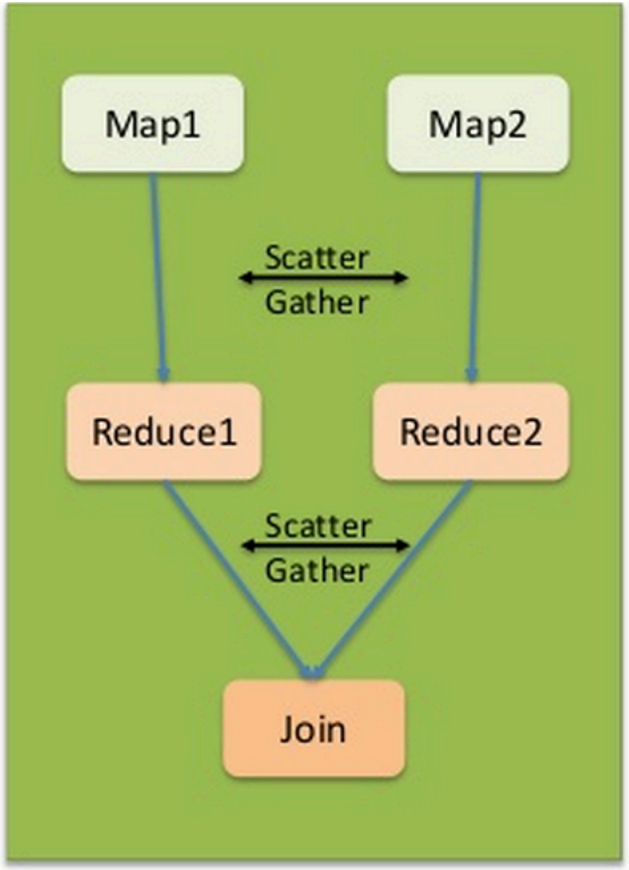
Directed acyclic execution

- Vertices are data transformations
- Edges are data movement



Directed acyclic execution

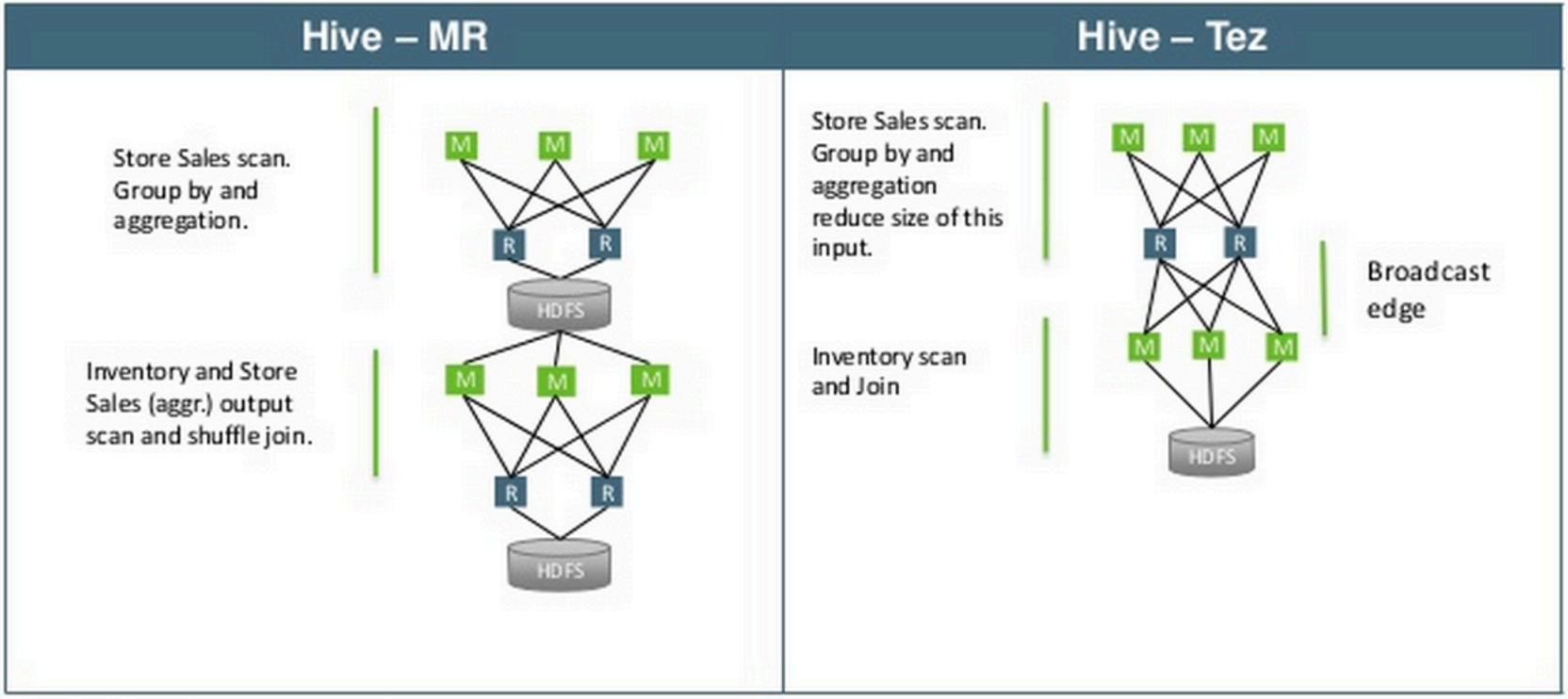
```
// Define DAG
DAG dag = new DAG();

// Define Vertex
Vertex Map1 = new Vertex(Processor.class);

// Define Edge
Edge edge = Edge(Map1, Reduce1,
SCATTER_GATHER, PERSISTED, SEQUENTIAL,
Output.class, Input.class);

// Connect them
dag.addVertex(Map1).addEdge(edge)...
```



Comparison

```
SELECT ss.ss_item_sk, ss.ss_quantity, avg_price, inv.inv_quantity_on_hand
FROM (select avg(ss_sold_price) as avg_price, ss_item_sk, ss_quantity_sk from store_sales
group by ss_item_sk) ss
JOIN inventory inv
ON (inv.inv_item_sk = ss.ss_item_sk);
```

Hive : Broadcast Join



Comparison

- Read/write barrier between successive computations
- Overhead of launching new job
- map() reads at the start of every job

- Engine has a global picture of the workflow

Alternative Frameworks

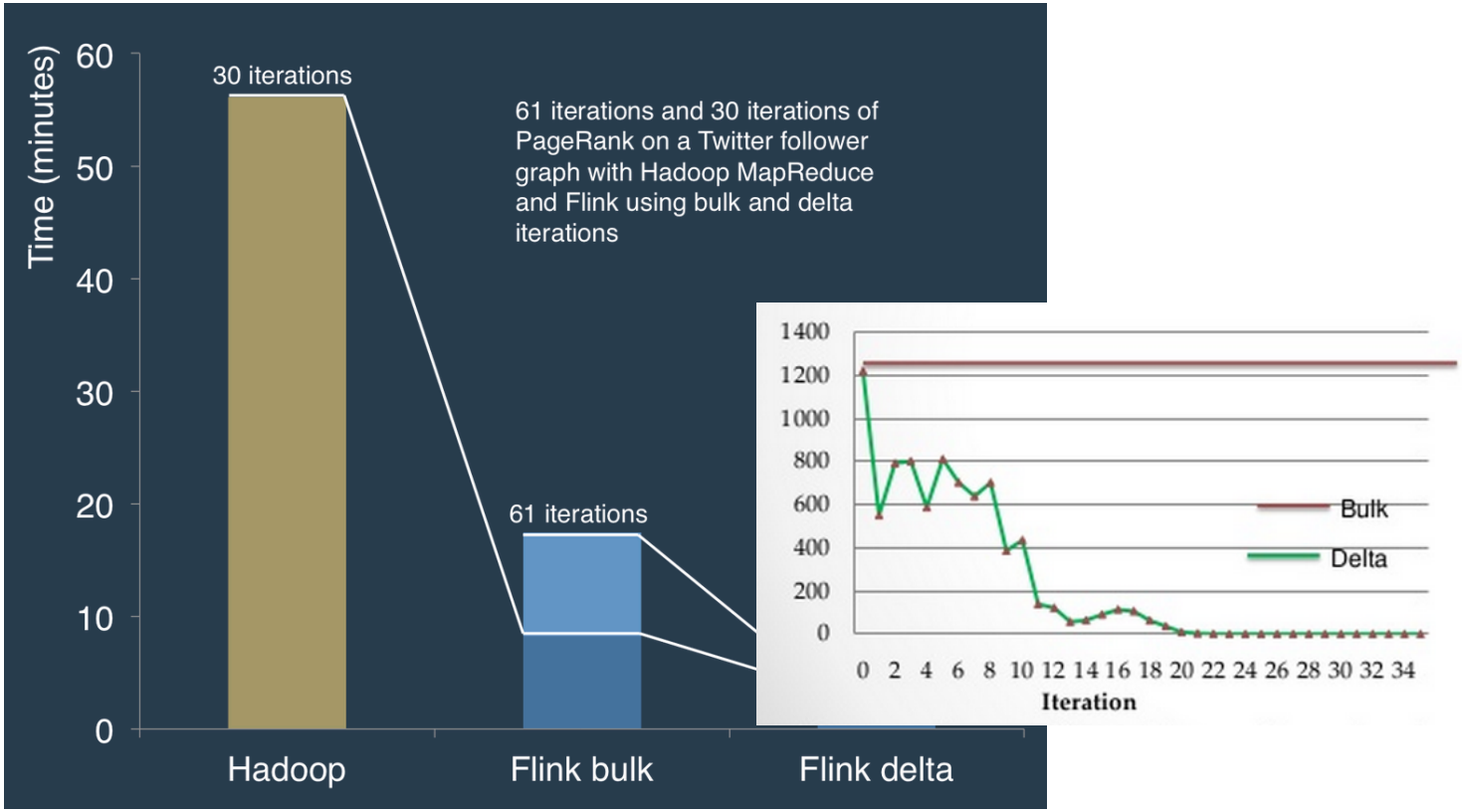
1. Apache Mahout
2. Apache Giraph
3. GraphLab
4. Apache Storm
5. Apache Tez
6. Apache Flink

Apache Flink

- [formerly StratoSphere]
- “Fast and reliable large-scale data processing engine”
- Incubation in April 2014, TLP in December 2014

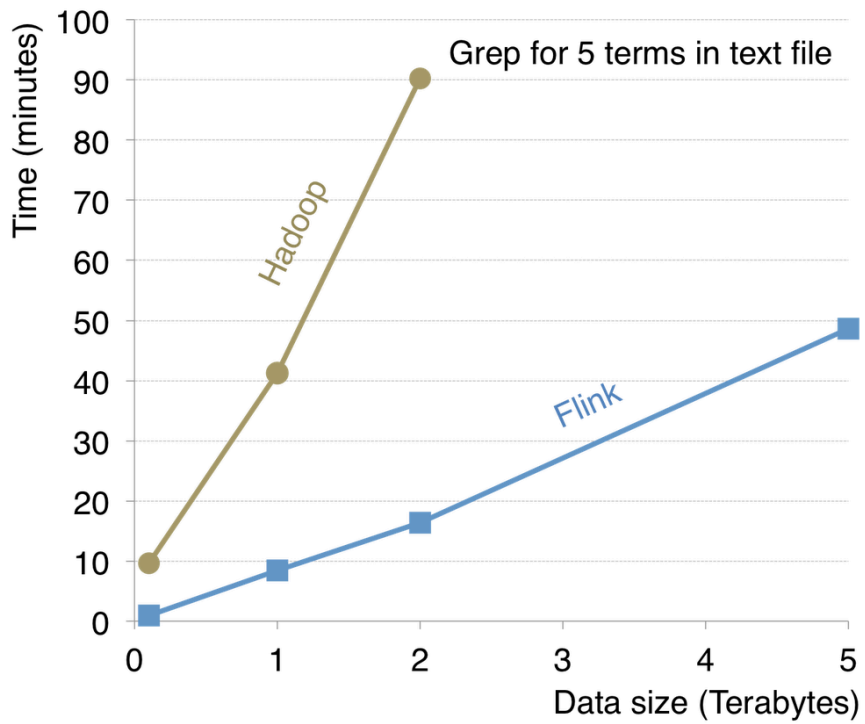
Selling points

- Fast
  - In-memory computations (like Spark)
  - Integrates iterative processing



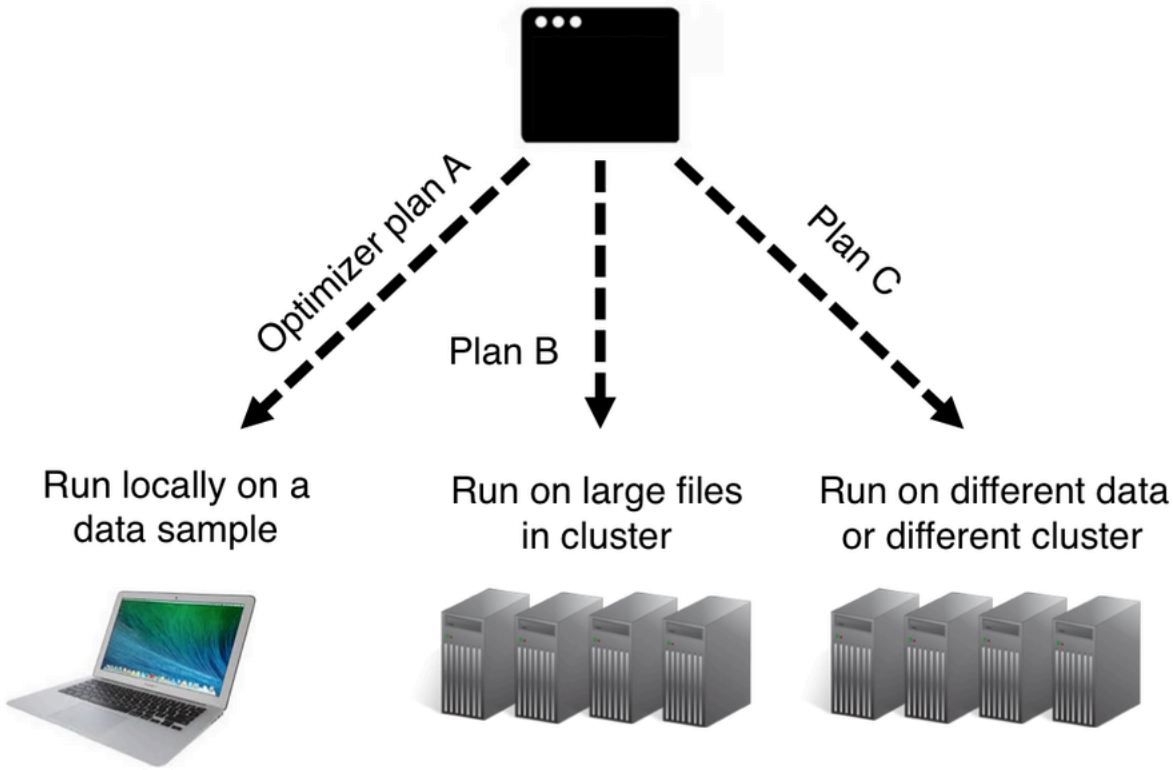
Selling points

- Reliable and scalable
  - Designed to keep working when memory runs out
  - Contains its own memory management, serialization, and type inference frameworks



Selling points

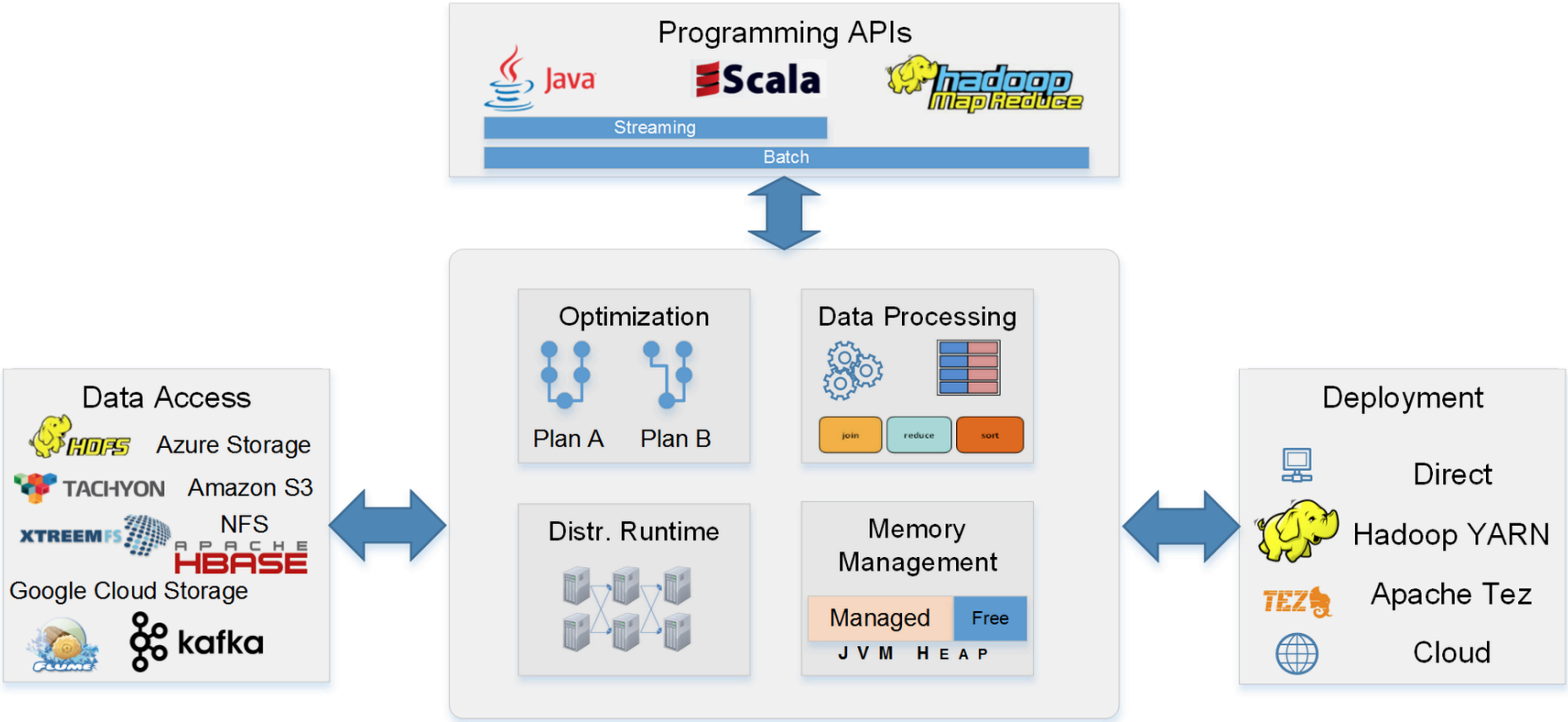
- Ease of use
  - Very few configuration options required
  - Infers most of the configuration itself



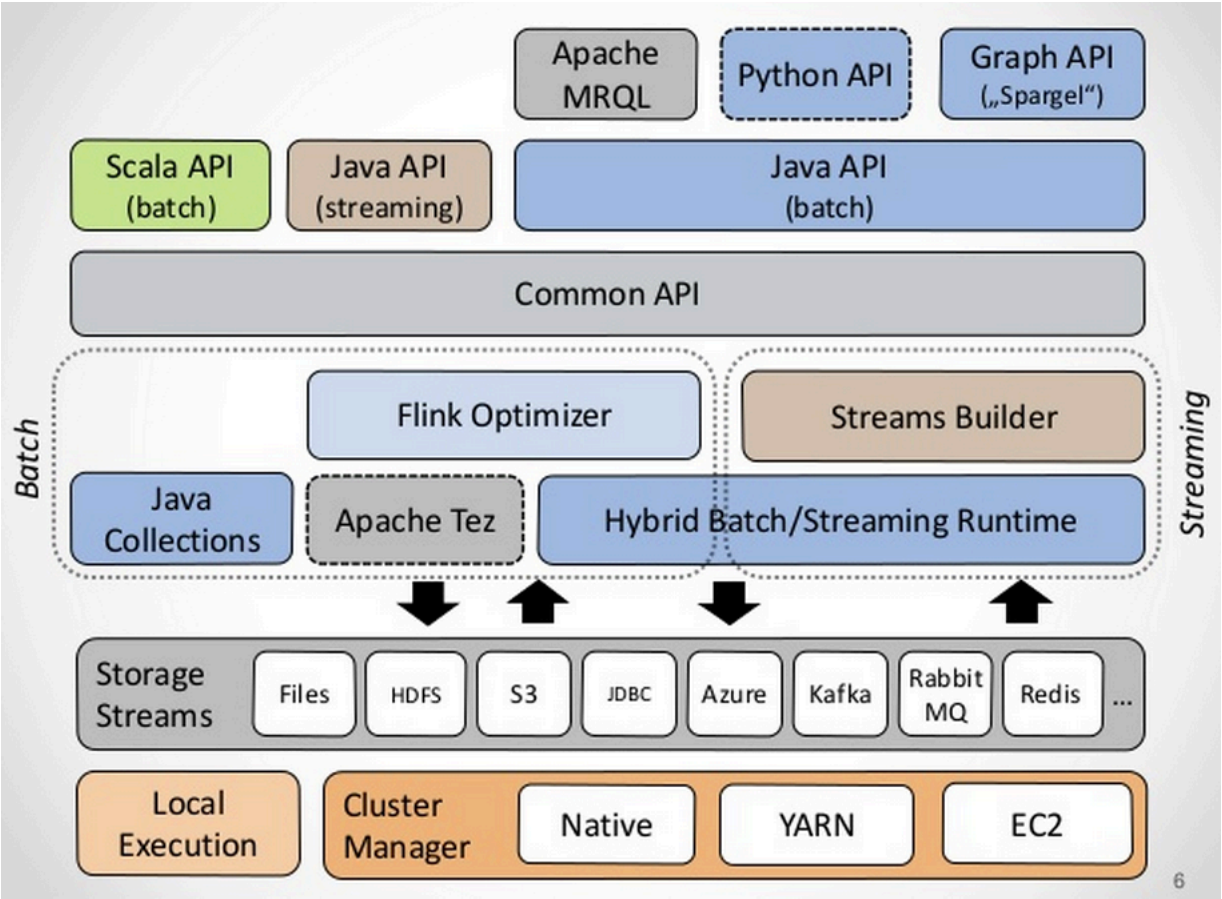
Ease of use

- No memory thresholds to configure
  - Flink manages its own memory
- Requires no network configuration
  - Only needs slave information
- Needs no configured serializers
  - Flink handles this internally
- Programs automatically adjust to data type
  - Flink's internals dynamically choose execution strategies

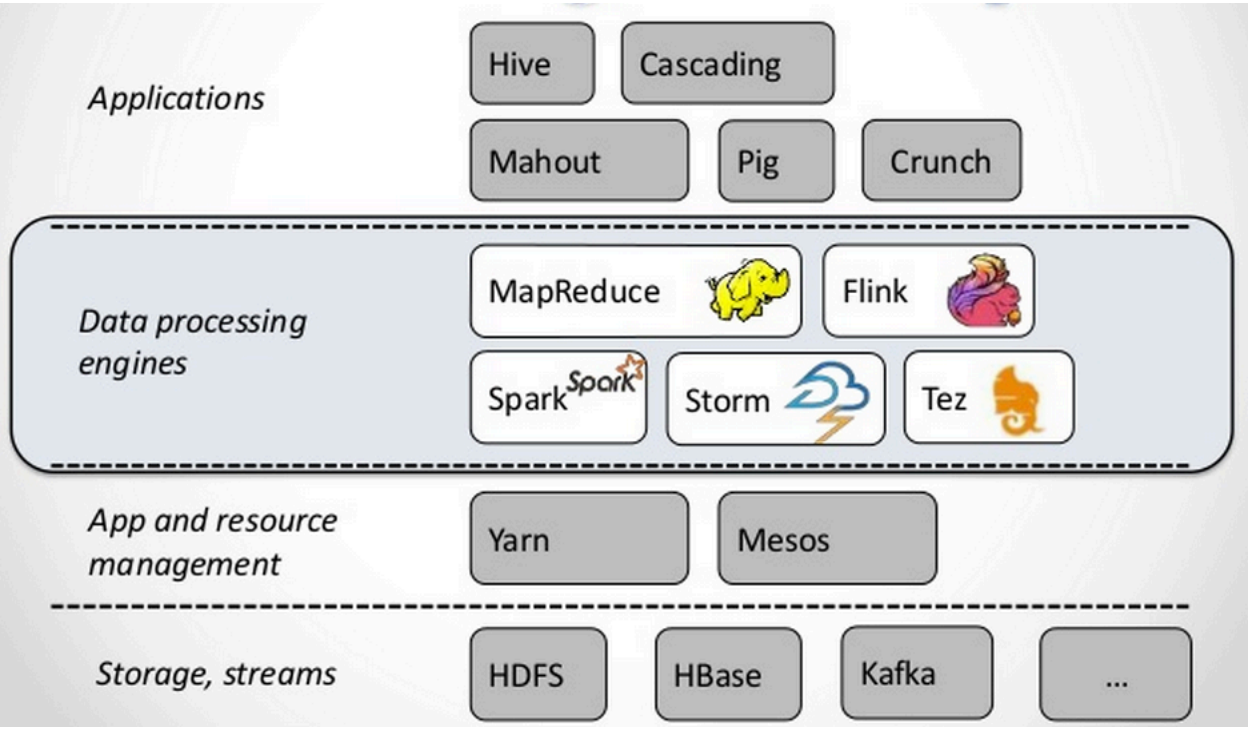
Architecture



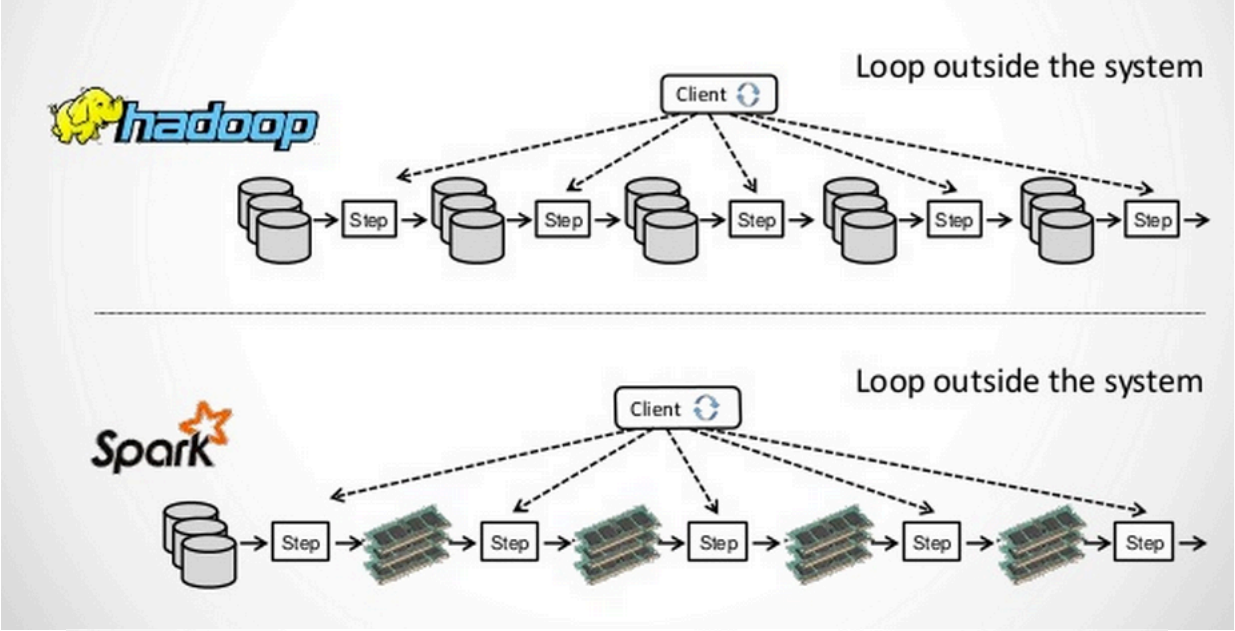
Architecture

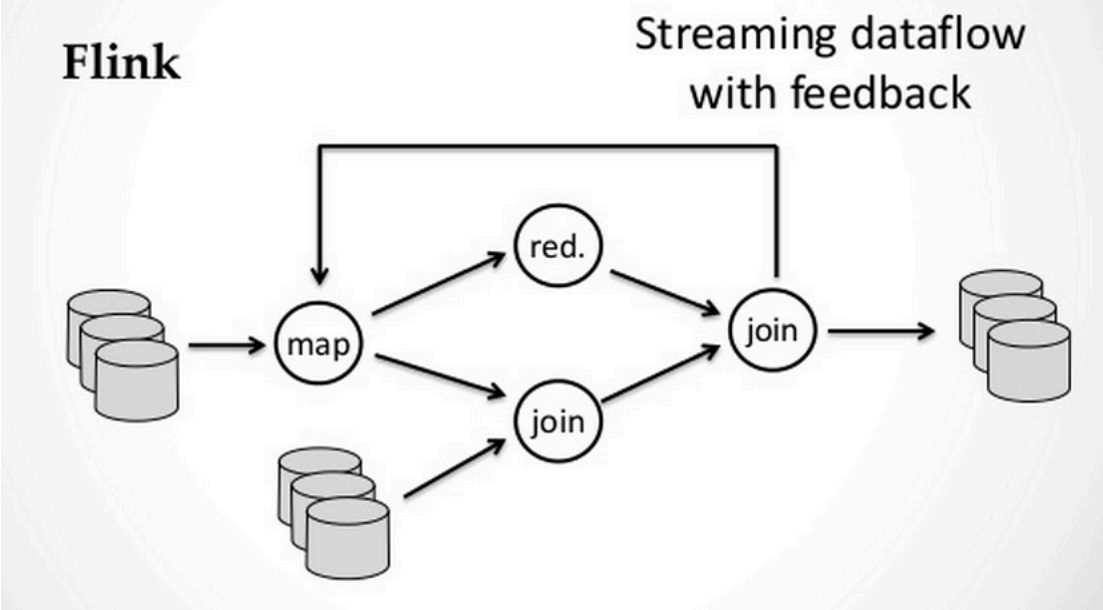


Software stack



Flink engine





Flink engine

	hadoop MapReduce	TEZ	Spark	Flink
API	MapReduce on k/v pairs	k/v pair Readers/Writers	Transformations on k/v pair collections	Iterative transformations on collections
Paradigm	MapReduce	DAG	RDD	Cyclic dataflows
Optimization	none	none	Optimization of SQL queries	Optimization in all APIs
Execution	Batch sorting	Batch sorting and partitioning	Batch with memory pinning	Stream with out-of-core algorithms

Flink engine

- On-the-fly program optimization

```
DataSet<Tuple...> large = env.readCsv(...);
DataSet<Tuple...> medium = env.readCsv(...);
DataSet<Tuple...> small = env.readCsv(...);

DataSet<Tuple...> joined1 = large.join(medium).where(3).equals(1)
                            .with(new JoinFunction() { ... });

DataSet<Tuple...> joined2 = small.join(joined1).where(0).equals(2)
                            .with(new JoinFunction() { ... });

DataSet<Tuple...> result = joined2.groupBy(3).aggregate(MAX, 2);
```

Possible execution

1) Partitioned hash-join

2) Broadcast hash-join

3) Grouping /Aggregation reuses the partitioning from step (1) → No shuffle!!!

Partitioned ≈ Reduce-side
Broadcast ≈ Map-side

37

WordCount!

- Uses Scala, just like Spark

```
case class Word (word: String, frequency: Int)

val env = ExecutionEnvironment.getExecutionEnvironment()

val lines = env.readTextFile(...)

lines
    .flatMap {line => line.split(" ").map(word => Word(word,1))}
    .groupBy("word").sum("frequency”)
    .print()

env.execute()
```

Flink API

- map, flatMap, filter, groupBy, reduce, reduceGroup, aggregate, join, coGroup, cross, project, distinct, union, iterate, iterateDelta…
- All Hadoop InputFormats supported
- Windowing functions for streaming data
- Counters, accumulators, broadcast variables
- Local standalone mode for testing/debugging

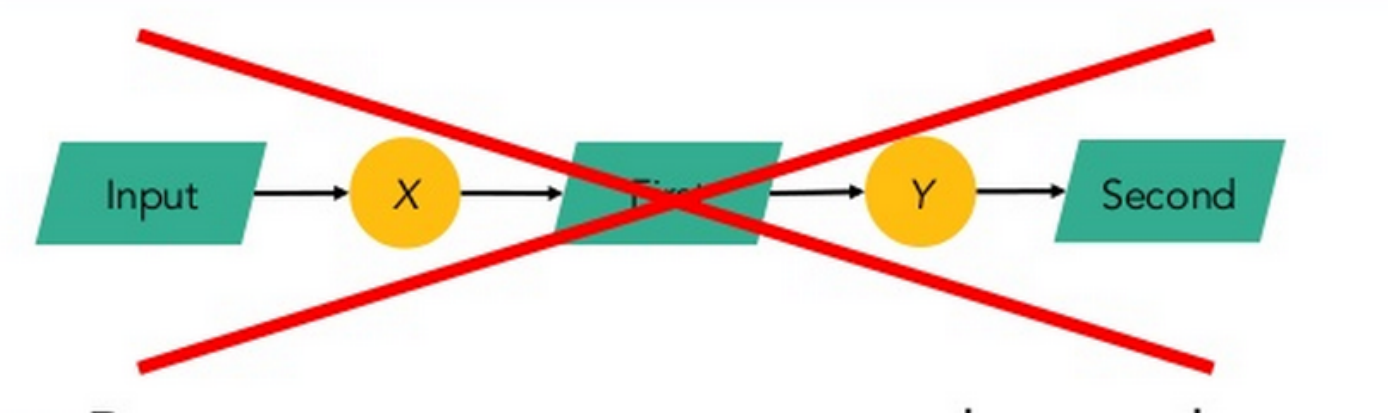
Flink philosophy

- Developers made a concerted effort to ***hide*** internals from Flink users

- The Good
  - Anyone who has had OutOfMemoryExceptions in Spark will probably agree this is a very good thing
- The Bad
  - Execution model is much more complicated than Hadoop or Spark

Flink internals

- Programs are *not* executed eagerly



- Flink compiles program to an "execution plan"
  - Essentially a pipeline, rather than a staged or batched execution

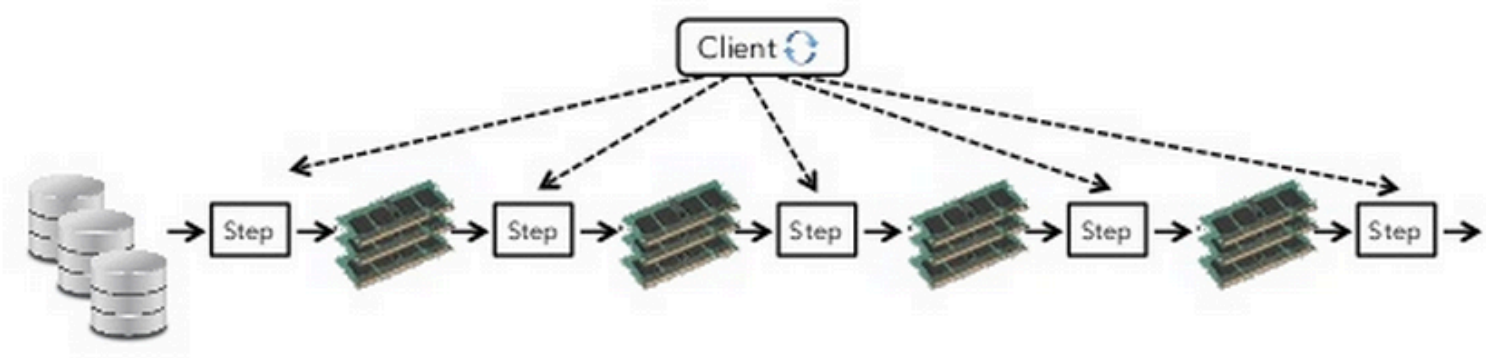
Iterative processing on Flink

```
DataSet<Page> pages = ...
DataSet<Neighborhood> edges = ...
DataSet<Page> oldRanks = pages; DataSet<Page> newRanks;

for (i = 0; i < maxIterations; i++) {
  newRanks = update(oldRanks, edges)
  oldRanks = newRanks
}

DataSet<Page> result = newRanks;
```

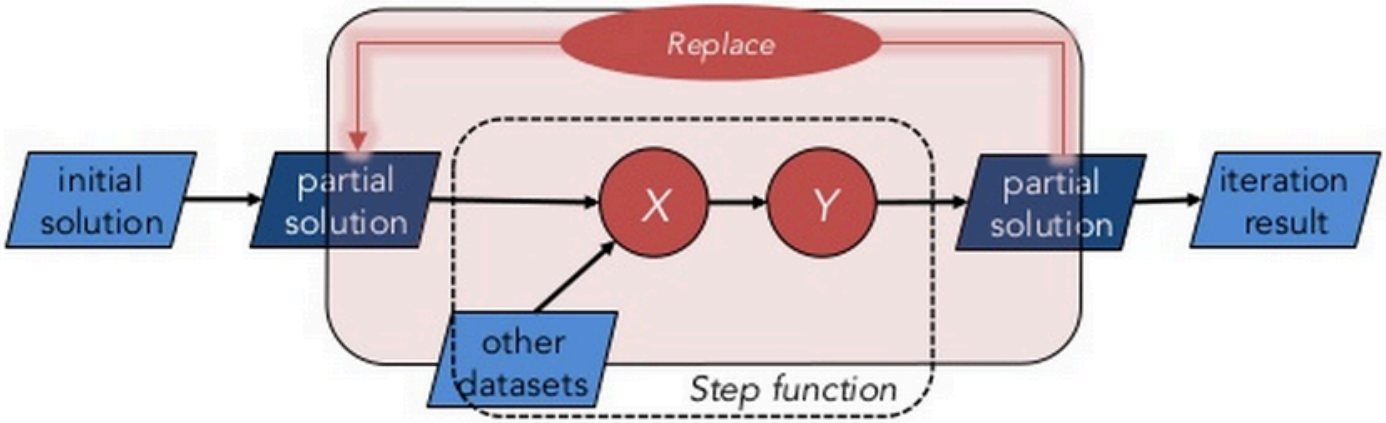
```
DataSet<Page> update (DataSet<Page> ranks, DataSet<Neighborhood> adjacency) {
  return oldRanks
    .join(adjacency)
    .where("id").equalTo("id")
    .with ( (page, adj, out) -> {
      for (long n : adj.neighbors)
        out.collect(new Page(n, df * page.rank / adj.neighbors.length))
    })
    .groupBy("id")
    .reduce ( (a, b) -> new Page(a.id, a.rank + b.rank) );
```

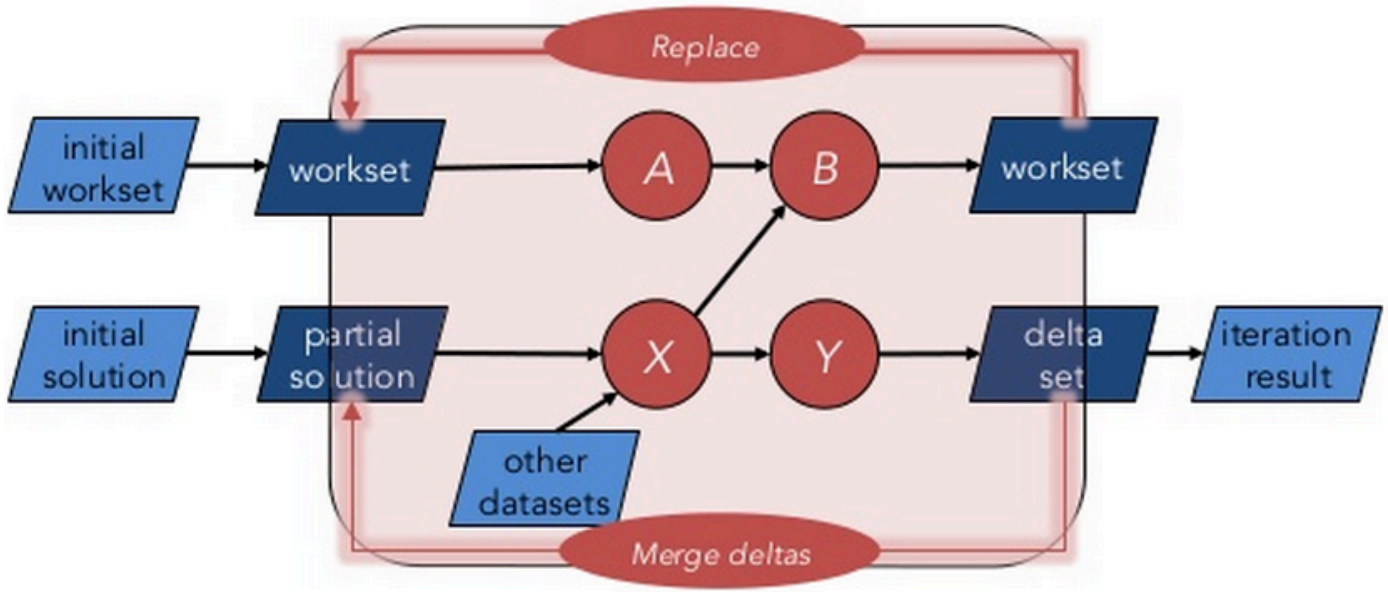
Iterative processing

- Hadoop, Spark, etc
  - Iterate by unrolling: loop submits one job per iteration
  - Data reuse by caching in memory and/or disk



Iterate natively [with delta]





Flink summary

- Flink decouples API from execution
  - Same program can be executed in many different ways
  - Ideally users do not care about this
- Pipelined execution, native iterations, program optimizer, serialized data manipulation
- Equivalent or better performance to Spark

Resources

- Apache Mahout
  - http://mahout.apache.org/users/sparkbindings/home.html
- Apache Giraph
  - http://www.slideshare.net/ClaudioMartella/giraph-at-hadoop-summit-2014
- GraphLab / Dato
  - https://dato.com/
- Apache Storm
  - http://www.slideshare.net/miguno/apache-storm-09-basic-training-verisign
- Apache Tez
  - http://www.slideshare.net/Hadoop_Summit/w-1205phall1saha
- Apache Flink
  - https://flink.apache.org/material.html