



Flink



Spark

# Flink vs. Spark

**Slim Baltagi** [@SlimBaltagi](https://twitter.com/SlimBaltagi)

Director of Big Data Engineering, Fellow  
Capital One



**FlinkForward**

— BERLIN 12/13 OCT 2015 —

# Agenda

**I. Motivation for this talk**

**II. Apache Flink vs. Apache Spark?**

**III. How Flink is used at Capital One?**

**IV. What are some key takeaways?**

# **I. Motivation for this talk**

- 1. Marketing fluff**
- 2. Confusing statements**
- 3. Burning questions & incorrect or outdated answers**
- 4. Helping others evaluating Flink vs. Spark**

# 1. Marketing fluff

— UP TO —  
**80%**  
**OFF**

# 1. Marketing fluff



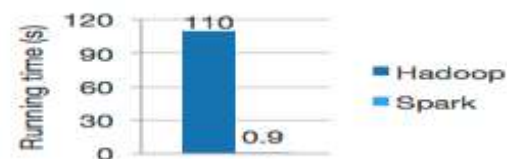
[Download](#) [Libraries](#) [Documentation](#) [Examples](#) [Community](#) [FAQ](#)

**Apache Spark™** is a fast and general engine for large-scale data processing.

## Speed

Run programs up to 100x faster than Hadoop MapReduce in memory, or 10x faster on disk.

Spark has an advanced DAG execution engine that supports cyclic data flow and in-memory computing.



Logistic regression in Hadoop and Spark

## About Apache Flink



### Open Source

- Started in 2009 by the Berlin-based database research groups
- In the Apache Incubator since mid 2014

### Fast + Reliable

- Fast, general purpose distributed data processing system
- Up to 100x faster than Hadoop

### Ready to use

- Programming APIs for Java and Scala
- Tested on large clusters

## 2. Confusing statements

- “**Spark** is already an excellent piece of software and is advancing very quickly. **No vendor — no new project — is likely to catch up. Chasing Spark would be a waste of time**, and would **delay availability of real-time analytic and processing services for no good reason.**” Source: MapReduce and Spark, **Mike Olson. Chief Strategy Officer, Cloudera.** December, 30<sup>th</sup> 2013

<http://vision.cloudera.com/mapreduce-spark/>

- “**Goal: one engine for all data sources, workloads and environments.**” Source: Slide 15 of ‘New Directions for Apache Spark in 2015’, **Matei Zaharia. CTO, Databricks.** February 20<sup>th</sup> , 2015. <http://www.slideshare.net/databricks/new-directions-for-apache-spark-in-2015>

### 3. Burning questions & incorrect or outdated answers

- "Projects that depend on **smart optimizers rarely work well in real life.**" Curt Monash, **Monash Research**.

January 16, 2015 <http://www.computerworld.com/article/2871760/big-data-digest-how-many-hadoops-do-we-really-need.html>

- "Flink is basically a Spark alternative out of Germany, which I've been **dismissing as unneeded**". Curt Monash, **Monash Research**, March 5, 2015.

<http://www.dbms2.com/2015/03/05/cask-and-cdap/>

- "Of course, this is all a bullish argument for Spark (or **Flink, if I'm wrong to dismiss its chances as a Spark competitor**)."

Curt Monash, **Monash Research**, September 28, 2015. <http://www.dbms2.com/2015/09/28/the-potential-significance-of-cloudera-kudu/>

### 3. Burning questions & incorrect or outdated answers

- “The benefit of Spark's micro-batch model is that you get full fault-tolerance and "exactly-once" processing for the entire computation, meaning it can recover all state and results even if a node crashes. Flink and Storm don't provide this...” Matei Zaharia. CTO, Databricks. May 2015 <http://www.kdnuggets.com/2015/05/interview-matei-zaharia-creator-apache-spark.html>
- “I understand Spark Streaming uses micro-batching. Does this increase latency? While Spark does use a micro-batch execution model, this does not have much impact on applications...” <http://spark.apache.org/faq.html>



## 4. Help others evaluating Flink vs. Spark

- Besides the marketing fluff, the confusing statements, the incorrect or outdated answers to burning questions, the **little information on the subject of Flink vs. Spark is available piecemeal!**
- While evaluating different stream processing tools at Capital One, we built a **framework listing categories and over 100 criteria to assess these stream processing tools.**
- In the next section, I'll be sharing this framework and use it to **compare Spark and Flink on a few key criteria.**
- We hope this will be beneficial to you as well when selecting Flink and/or Spark for stream processing.

# Agenda

**I. Motivation for this talk**

**II. Apache Flink vs. Apache Spark?**

**III. How Flink is used at Capital One?**

**IV. What are some key takeaways?**

## **II. Apache Flink vs. Apache Spark?**

- 1. What is Apache Flink?**
- 2. What is Apache Spark?**
- 3. Framework to evaluate Flink and Spark**
- 4. Flink vs. Spark on a few key criteria**
- 5. Future work**

# 1.What is Apache Flink?



- Squirrel: **Animal**. In harmony with other animals in the Hadoop ecosystem (Zoo): elephant, pig, python, camel,...
  - Squirrel: reflects the **meaning of the word 'Flink'**: German for “nimble, swift, speedy” which are also characteristics of the squirrel.
  - Red **color**. In harmony with red squirrels in Germany to reflect its root at German universities
  - **Tail**: colors matching the ones of the feather symbolizing the Apache Software Foundation.
- Commitment to build Flink in the open source!?**

# 1.What is Apache Flink?



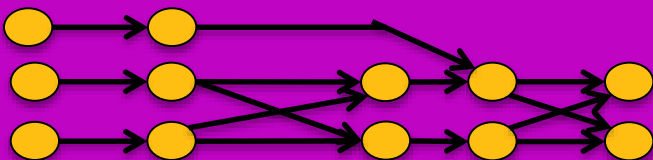
- “Apache Flink is an open source platform for distributed stream and batch data processing.”

<https://flink.apache.org/>

- See also the definition in

Wikipedia: [https://en.wikipedia.org/wiki/Apache\\_Flink](https://en.wikipedia.org/wiki/Apache_Flink)

# 1. What is Apache Flink?

APIs & LIBRARIES	<div>FlinkML</div> <div>Table</div> <div>Gelly</div> <div>Hadoop M/R</div> <div>Google Dataflow</div> <div>MRQL</div> <div>Cascading (wIP)</div> <div>Zeppelin</div>			<div>Table</div> <div>SAMOA</div> <div>Dataflow (wIP)</div> <div>Storm</div>		
	<div>DataSet (Java/Scala/Python)</div> <div>Batch Processing</div>			<div>DataStream (Java/Scala)</div> <div>Stream Processing</div>		
SYSTEM	<div>Batch Optimizer</div>			<div>Stream Builder</div>		
				<div>Runtime - Distributed Streaming Dataflow</div>		
DEPLOY	<div>Local</div> <div>Single JVM</div> <div>Embedded</div> <div>Docker</div>			<div>Cluster</div> <div>Standalone</div> <div>YARN, Tez,</div> <div>Mesos (WIP)</div>		
				<div>Cloud</div> <div>Google's GCE</div> <div>Amazon's EC2</div> <div>IBM Docker Cloud, ...</div>		
STORAGE	<div>Files</div> <div>Local</div> <div>HDFS</div> <div>S3, Azure Storage</div> <div>Tachyon</div>			<div>Databases</div> <div>MongoDB</div> <div>HBase</div> <div>SQL</div> <div>...</div>		
				<div>Streams</div> <div>Flume</div> <div>Kafka</div> <div>RabbitMQ</div> <div>...</div>		

## 2. What is Apache Spark?



- “Apache Spark™ is a fast and general engine for large-scale data processing.”

<http://spark.apache.org/>

- See also definition in Wikipedia:

[https://en.wikipedia.org/wiki/Apache\\_Spark](https://en.wikipedia.org/wiki/Apache_Spark)

- Logo was picked to reflect **Lightning-fast cluster computing**

## 2. What is Apache Spark?

*Spark* 



Scala



Java



python



ML Pipelines

DataFrames

Spark SQL

Spark  
Streaming

MLlib

GraphX

Spark Core

Data Sources



cassandra



APACHE  
HBASE



hive



{JSON}



MySQL

elasticsearch



# **3. Framework to evaluate Flink and Spark**

**1. Background**

**2. Fit-for-purpose Categories**

**3. Organizational-fit Categories**

**4. Miscellaneous/Other Categories**

# **1. Background**

## **1.1 Definition**

## **1.2 Origin**

## **1.3 Maturity**

## **1.4 Version**

## **1.5 Governance model**

## **1.6 License model**

## **2. Fit-for-purpose Categories**

**2.1 Security**

**2.2 Provisioning & Monitoring Capabilities**

**2.3 Latency & Processing Architecture**

**2.4 State Management**

**2.5 Processing Delivery Assurance**

**2.6 Database Integrations, Native vs. Third party connector**

**2.7 High Availability & Resiliency**

**2.8 Ease of Development**

**2.9 Scalability**

**2.10 Unique Capabilities/Key Differentiators**

## **2. Fit-for-purpose Categories**

### **2.1 Security**

**2.1.1 Authentication, Authorization**

**2.1.2 Data at rest encryption (data persisted in the framework)**

**2.1.3 Data in motion encryption (producer ->framework -> consumer)**

**2.1.4 Data in motion encryption (inter-node communication)**

## **2. Fit-for-purpose Categories**

### **2.2 Provisioning & Monitoring Capabilities**

**2.2.1 Robustness of Administration**

**2.2.2 Ease of maintenance: Does technology provide configuration, deployment, scaling, monitoring, performance tuning and auditing capabilities?**

**2.2.3 Monitoring & Alerting**

**2.2.4 Logging**

**2.2.5 Audit**

**2.2.6 Transparent Upgrade: Version upgrade with minimum downtime**

## **2. Fit-for-purpose Categories**

### **2.3 Latency & Processing Architecture**

**2.3.1 Supports tuple at a time, micro-batch, transactional updates and batch processing**

**2.3.2 Computational model**

**2.3.3 Ability to reprocess historical data from source**

**2.3.4 Ability to reprocess historical data from native engine**

**2.3.5 Call external source (API/database calls)**

**2.3.6 Integration with Batch (static) source**

**2.3.7 Data Types (images, sound etc.)**

**2.3.8 Supports complex event processing and pattern detection vs. continuous operator model (low latency, flow control)**

## 2. Fit-for-purpose Categories

### 2.3 Latency & Processing Architecture

**2.3.9 Handles stream imperfections (delayed)**

**2.3.10 Handles stream imperfections (out-of-order)**

**2.3.11 Handles stream imperfections (duplicate)**

**2.3.12 Handles seconds, sub-second or millisecond event processing (Latency)**

**2.3.13 Compression**

**2.3.14 Support for batch analytics**

**2.3.15 Support for iterative analytics (machine learning, graph analytics)**

**2.3.16 Data lineage provenance (origin of the owner)**

**2.3.17 Data lineage (accelerate recovery time)**

## **2. Fit-for-purpose Categories**

### **2.4 State Management**

**2.4.1 Stateful vs. Stateless**

**2.4.2 Is stateful data Persisted locally vs. external database vs. Ephemeral**

**2.4.3 Native rolling, tumbling and hopping window support**

**2.4.4 Native support for integrated data store**



## **2. Fit-for-purpose Categories**

### **2.5 Processing Delivery Assurance**

**2.5.1 Guarantee (At least once)**

**2.5.2 Guarantee (At most once)**

**2.5.3 Guarantee (Exactly once)**

**2.5.4 Global Event order guaranteed**

**2.5.5 Guarantee predictable and repeatable  
outcomes( deterministic or not)**

## **2. Fit-for-purpose Categories**

### **2.6 Database Integrations, Native vs. Third party connector**

**2.6.1 NoSQL database integration**

**2.6.2 File Format (Avro, Parquet and other format support)**

**2.6.3 RDBMS integration**

**2.6.4 In-memory database integration/ Caching integration**

## 2. Fit-for-purpose Categories

### 2.7 High Availability & Resiliency

**2.7.1 Can the system avoid slowdown due to straggler node**

**2.7.2 Fault-Tolerance (does the tool handle node/operator/messaging failures without catastrophically failing)**

**2.7.3 State recovery from in-memory**

**2.7.4 State recovery from reliable storage**

**2.7.5 Overhead of fault tolerance mechanism (Does failure handling introduce additional latency or negatively impact throughput?)**

**2.7.6 Multi-site support (multi-region)**

**2.7.7 Flow control: backpressure tolerance from slow operators or consumers**

**2.7.8 Fast parallel recovery vs. replication or serial recovery on one node at a time**

# 2. Fit-for-purpose Categories

## 2.8 Ease of Development

2.8.1 SQL Interface

2.8.2 Real-Time debugging option

2.8.3 Built-in stream oriented abstraction (streams, windows, operators , iterators - expressive APIs that enable programmers to quickly develop streaming data applications)

2.8.4 Separation of application logic from fault tolerance

2.8.5 Testing tools and framework

2.8.6 Change management: multiple model deployment ( E.g. separate cluster or can one create multiple independent redundant streams internally)

2.8.7 Dynamic model swapping (Support dynamic updating of operators/topology/DAG without restart or service interruption)

2.8.8 Required knowledge of system internals to develop an application

2.8.9 Time to market for applications

2.8.10 Supports plug-in of external libraries

2.8.11 API High Level/Low Level

2.8.12 Easy to configuration

2.8.13 GUI based abstraction layer

## 2. Fit-for-purpose Categories

### 2.9 Scalability

**2.9.1 Supports multi-thread across multiple processors/cores**

**2.9.2 Distributed across multiple machines/servers**

**2.9.3 Partition Algorithm**

**2.9.4 Dynamic elasticity - Scaling with minimum impact/performance penalty**

**2.9.5 Horizontal scaling with linear performance/throughput**

**2.9.6 Vertical scaling (GPU)**

**2.9.7 Scaling without downtime**

# **3. Organizational-fit Categories**

**3.1 Maturity & Community Support**

**3.2 Support Languages for Development**

**3.3 Cloud Portability**

**3.4 Compatibility with Native Hadoop  
Architecture**

**3.5 Adoption of Community vs. Enterprise  
Edition**

**3.6 Integration with Message Brokers**

# 3. Organizational-fit Categories

## 3.1 Maturity & Community Support

3.1.1 Open Source Support

3.1.2 Maturity (years)

3.1.3 Stable

3.1.4 Centralized documentation with versioning support

3.1.5 Documentation of programming API with good code examples

3.1.6 Centralized visible roadmap

3.1.7 Community acceptance vs. Vendor driven

3.1.8 Contributors

# 3. Organizational-fit Categories

## 3.2 Support Languages for Development

3.2.1 Language technology was built on

3.2.2 Language supported to access  
technology



# 3. Organizational-fit Categories

## 3.3 Cloud Portability

**3.3.1 Ease of migration between cloud vendors**

**3.3.2 Ease of migration between on premise to cloud**

**3.3.3 Ease of migration from on premise to complete cloud services**

**3.3.4 Cloud compatibility (AWS, Google, Azure)**

# 3. Organizational-fit Categories

## 3.4 Compatibility with Native Hadoop Architecture

**3.4.1 Implement on top of Hadoop YARN vs. Standalone**

**3.4.2 Mesos**

**3.4.3 Coordination with Apache Zookeeper**

# 3. Organizational-fit Categories

## 3.5 Adoption of Community vs. Enterprise Edition

### 3.5.1 Open Source

### 3.5.2 Enterprise Support

# **4. Miscellaneous/Other Categories**

**4.1 Best Suited for**

**4.2 Key use case scenarios**

**4.3 Companies using technology**

## **4. Flink vs. Spark on a few key criteria**

- 1. Streaming Engine**
- 2. Iterative Processing**
- 3. Memory Management**
- 4. Optimization**
- 5. Configuration**
- 6. Tuning**
- 7. Performance**

## 4.1. Streaming Engine

- Many time-critical applications need to process **large streams of live data** and provide results in **real-time**.

For example:

- ✓ Financial Fraud detection
  - ✓ Financial Stock monitoring
  - ✓ Anomaly detection
  - ✓ Traffic management applications
  - ✓ Patient monitoring
  - ✓ Online recommenders
- Some claim that 95% of streaming use cases can be handled with micro-batches!? Really!!!

# 4.1. Streaming Engine



- **Spark's micro-batching isn't good enough!**
- **Ted Dunning, Chief Applications Architect at MapR, talk at the Bay Area Apache Flink Meetup on August 27, 2015**

<http://www.meetup.com/Bay-Area-Apache-Flink-Meetup/events/224189524/>

- ✓ Ted described several **use cases** where batch and **micro batch processing** is **not appropriate** and described why.
- ✓ He also described what a **true streaming solution** needs to provide for solving these problems.
- ✓ These use cases were taken from **real industrial situations**, but the descriptions drove down to **technical details** as well.

## 4.1. Streaming Engine



- “I would consider **stream data analysis** to be a **major unique selling proposition** for **Flink**. Due to its **pipelined architecture**, Flink is a perfect match for big data stream processing in the Apache stack.” – **Volker Markl**

Ref.: On Apache Flink. Interview with Volker Markl, June 24<sup>th</sup> 2015

<http://www.odpms.org/blog/2015/06/on-apache-flink-interview-with-volker-markl/>

- Apache **Flink uses streams for all workloads**: streaming, SQL, micro-batch and batch. Batch is just treated as a finite set of streamed data. This makes **Flink** the **most sophisticated** distributed open source **Big Data processing engine** (not the most **mature** one yet!).



## 4.2. Iterative Processing

**Why Iterations?** Many Machine Learning and Graph processing **algorithms need** iterations! For example:

➤ **Machine Learning** Algorithms

- ✓ Clustering (K-Means, Canopy, ...)
- ✓ Gradient descent (Logistic Regression, Matrix Factorization)

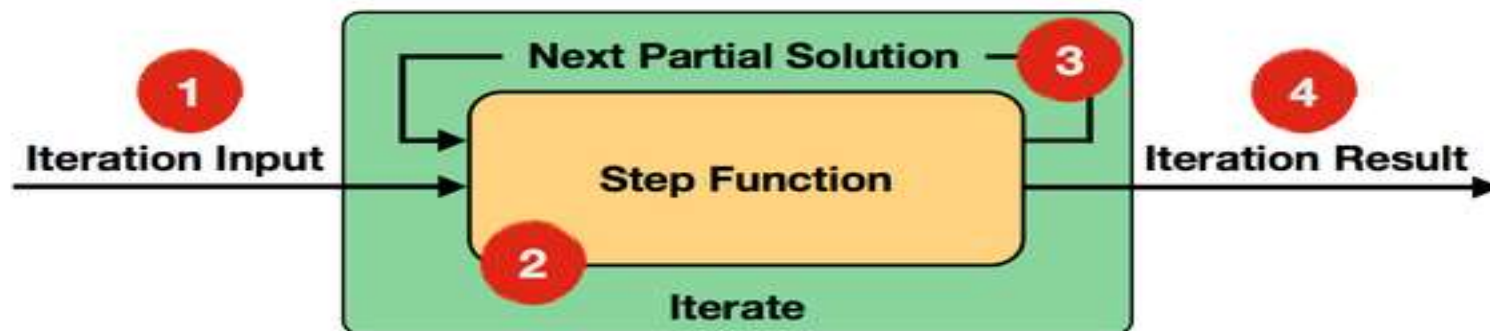
➤ **Graph Processing** Algorithms

- ✓ Page-Rank, Line-Rank
- ✓ Path algorithms on graphs (shortest paths, centralities, ...)
- ✓ Graph communities / dense sub-components
- ✓ Inference (Belief propagation)

## 4.2. Iterative Processing



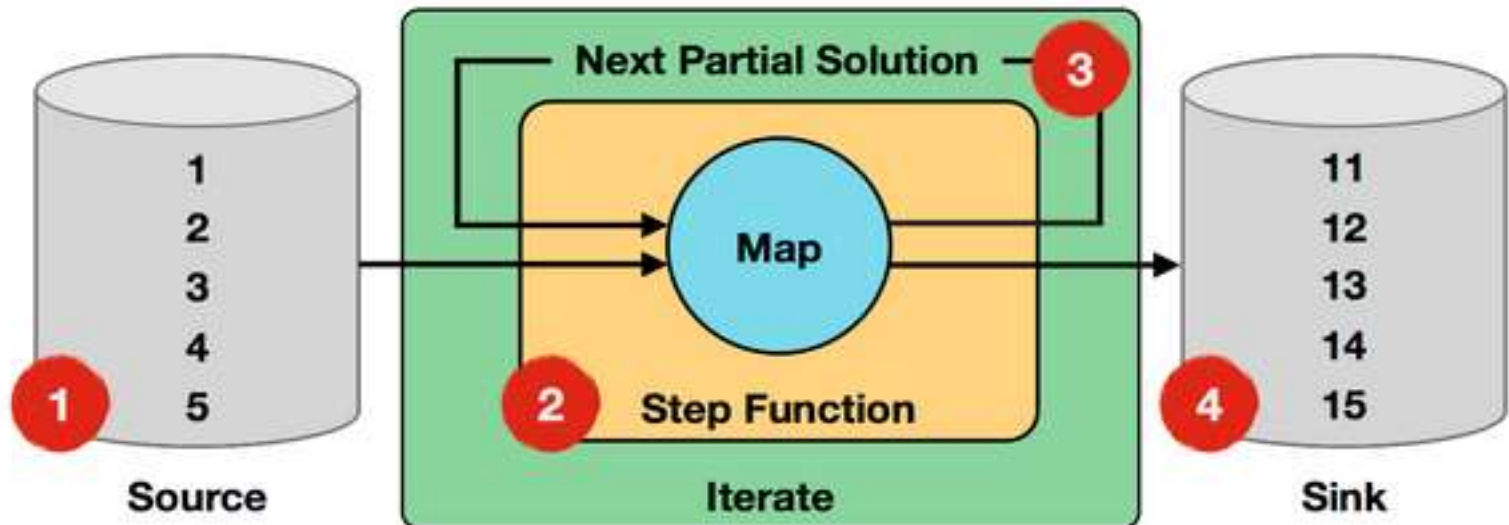
- Flink's API offers two dedicated iteration operations: **Iterate** and **Delta Iterate**.
- Flink executes programs with **iterations** as **cyclic data flows**: a data flow program (and all its operators) is scheduled **just once**.
- In each iteration, the step function **consumes** the entire input (the **result of the previous iteration**, or the **initial data set**), and **computes** the next version of the partial solution



## 4.2. Iterative Processing



- **Delta iterations run only** on **parts** of the **data** that is changing and can significantly **speed up many machine learning** and **graph algorithms** because the work in each iteration decreases as the number of iterations goes on.



- **Documentation on iterations with Apache Flink**

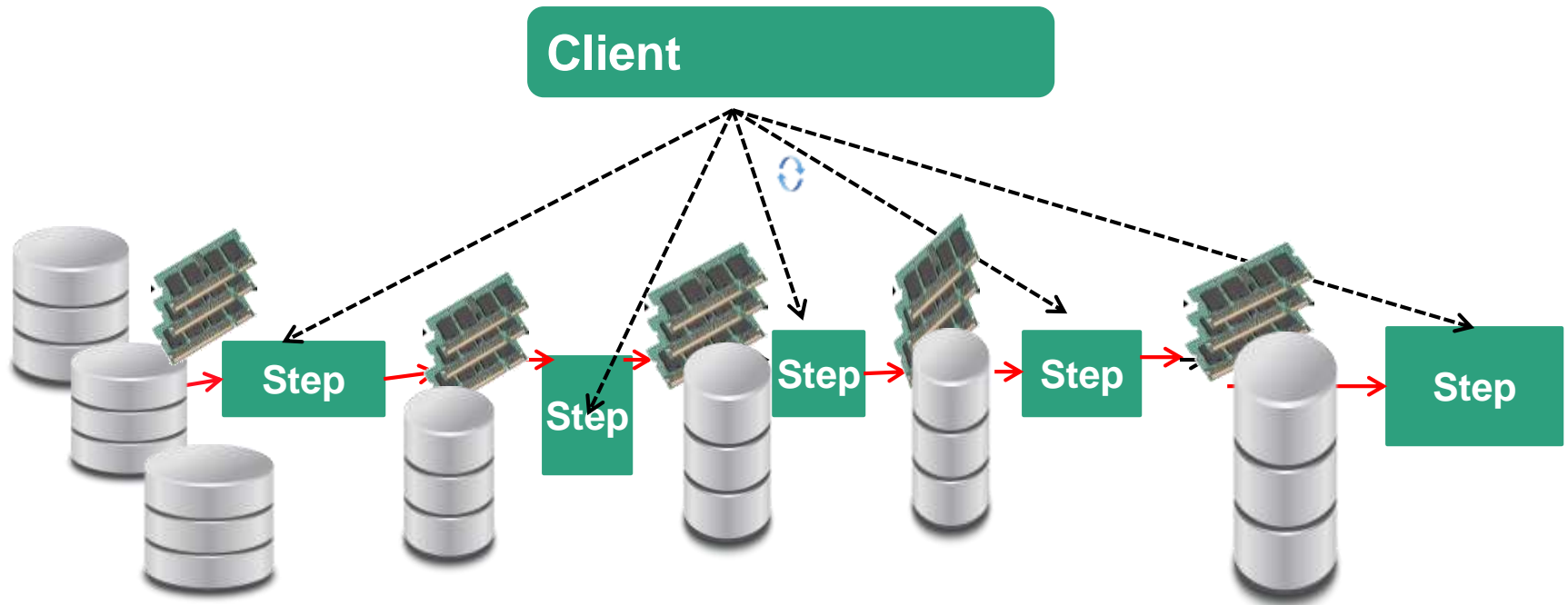
<http://ci.apache.org/projects/flink/flink-docs-master/apis/iterations.html>

## 4.2. Iterative Processing



**Non-native iterations** in Hadoop and **Spark** are implemented as regular **for-loops outside the system.**

```
for (int i = 0; i < maxIterations; i++) {  
    // Execute MapReduce job  
}
```



## 4.2. Iterative Processing

- Although **Spark** caches data across iterations, it still needs to **schedule** and **execute** a **new set of tasks** for **each iteration**.



- **Spinning Fast Iterative Data Flows** - Ewen et al. 2012 :  
[http://vldb.org/pvldb/vol5/p1268\\_stephanewen\\_vldb2012.pdf](http://vldb.org/pvldb/vol5/p1268_stephanewen_vldb2012.pdf) The Apache Flink model for incremental iterative dataflow processing. **Academic paper**.



- Recap of the paper, June 18, 2015 <http://blog.acolyer.org/2015/06/18/spinning-fast-iterative-dataflows/>

- **Documentation** on **iterations** with Apache

**Flink** <http://ci.apache.org/projects/flink/flink-docs-master/apis/iterations.html>

## 4.3. Memory Management

➤ Question: Spark vs. Flink low memory available?

➤ Question answered on

**stackoverflow.com** <http://stackoverflow.com/questions/31935299/spark-vs-flink-low-memory-available>



➤ The same question still unanswered on the Apache Spark Mailing List!! <http://apache-flink-user-mailing-list-archive.2336050.n4.nabble.com/spark-vs-flink-low-memory-available-t2364.html>



## 4.3. Memory Management



### Features:

- **C++ style** memory management **inside the JVM**
- User data stored in serialized **byte arrays** in JVM
- Memory is allocated, de-allocated, and used strictly using an internal **buffer** pool implementation.

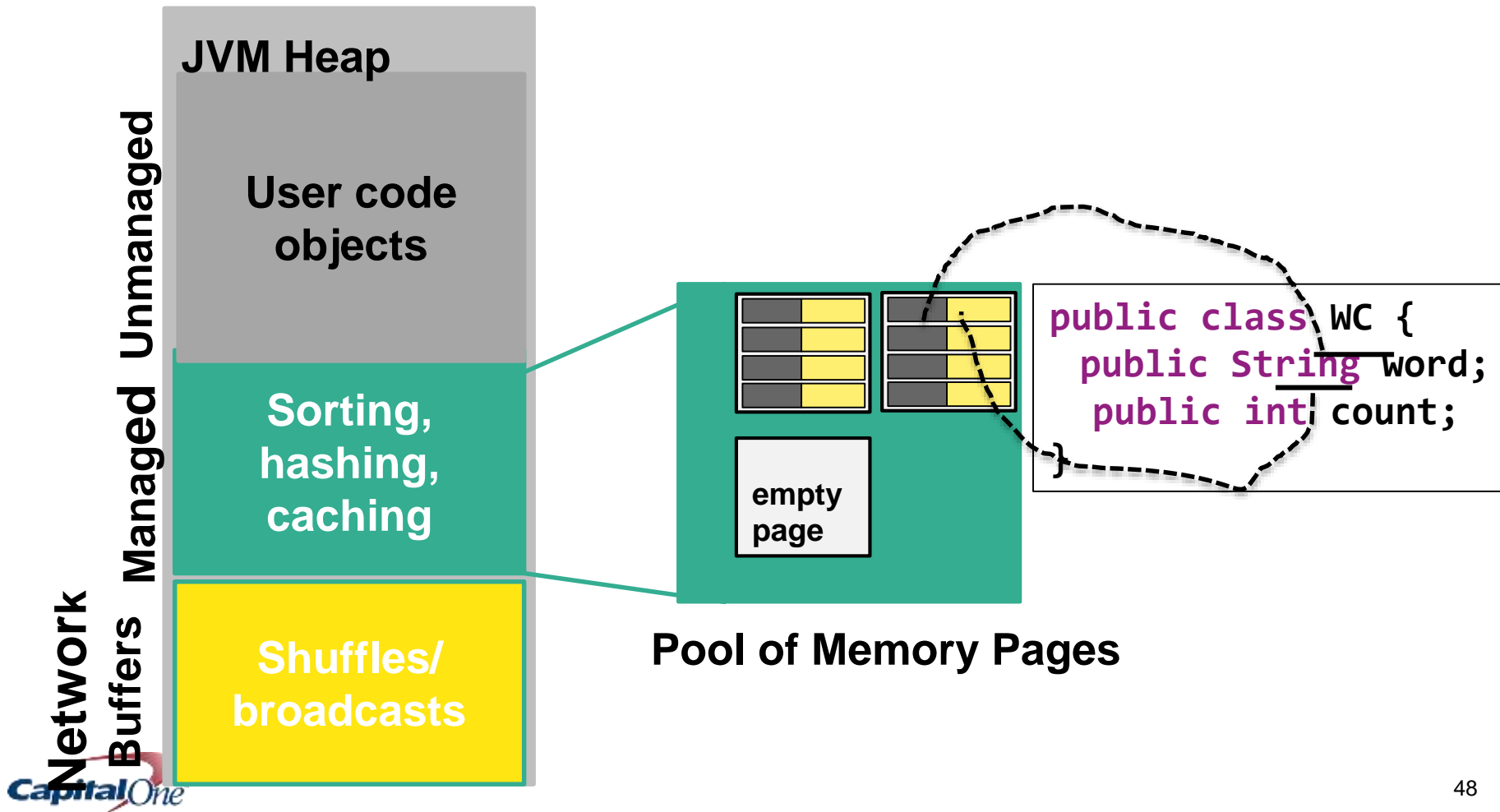
### Advantages:

1. Flink will **not throw an OOM** exception on you.
2. **Reduction of Garbage Collection (GC)**
3. Very **efficient disk spilling** and network transfers
4. **No Need for runtime tuning**
5. **More reliable and stable performance**

## 4.3. Memory Management



Flink contains its **own memory management stack**.  
To do that, Flink contains its **own type extraction**  
and **serialization** components.





## 4.3. Memory Management



- **Peeking into Apache Flink's Engine Room - by Fabian Hüske, March 13, 2015** <http://flink.apache.org/news/2015/03/13/peeking-into-Apache-Flinks-Engine-Room.html>
- **Juggling with Bits and Bytes - by Fabian Hüske, May 11, 2015**  
<https://flink.apache.org/news/2015/05/11/Juggling-with-Bits-and-Bytes.html>
- **Memory Management (Batch API) by Stephan Ewen-May 16, 2015**  
<https://cwiki.apache.org/confluence/pages/viewpage.action?pageId=53741525>
- **Flink added an Off-Heap option for its memory management component in Flink 0.10:**  
<https://issues.apache.org/jira/browse/FLINK-1320>

## 4.3. Memory Management



- Compared to Flink, Spark is still behind in custom memory management but is catching up with its project Tungsten for Memory Management and Binary Processing: manage memory explicitly and eliminate the overhead of JVM object model and garbage collection. April 28, 2014 <https://databricks.com/blog/2015/04/28/project-tungsten-bringing-spark-closer-to-bare-metal.html>
- It seems that Spark is adopting something similar to Flink and the initial Tungsten announcement read almost like Flink documentation!!

## 4.4 Optimization

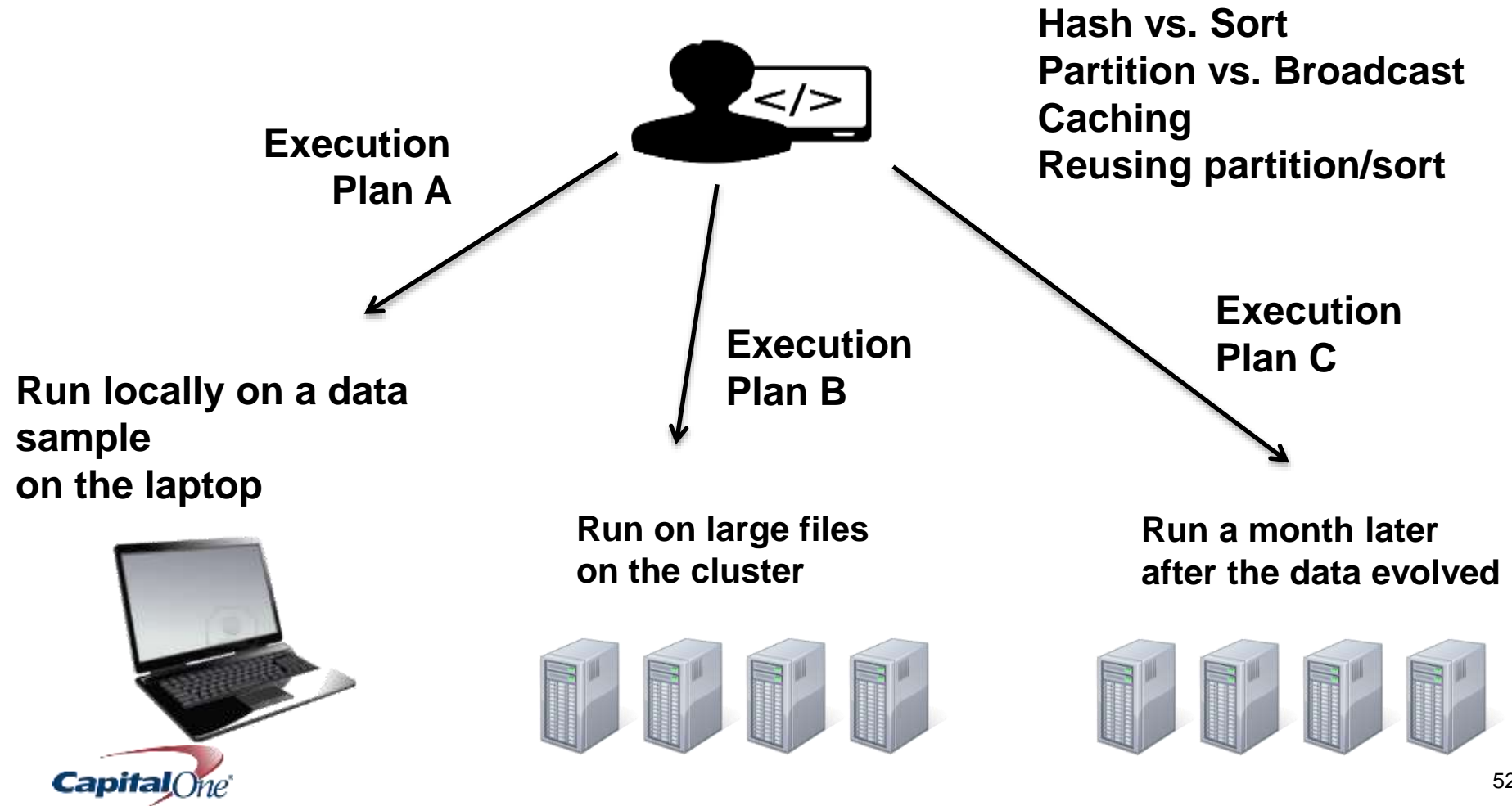


- Apache Flink comes with an **optimizer** that is independent of the actual programming interface.
- It chooses a **fitting execution strategy** depending on the inputs and operations.
- Example: the "**Join**" operator will **choose between** partitioning and broadcasting the data, as well as between running a sort-merge-join or a hybrid hash join algorithm.
- This helps you **focus on your application logic** rather than parallel execution.
- **Quick introduction to the Optimizer**: section 6 of the paper: 'The Stratosphere platform for big data analytics' [http://stratosphere.eu/assets/papers/2014-VLDBJ\\_Stratosphere\\_Overview.pdf](http://stratosphere.eu/assets/papers/2014-VLDBJ_Stratosphere_Overview.pdf)

# 4.4 Optimization



What is **Automatic Optimization**? The system's built-in optimizer takes care of finding the best way to execute the program in any environment.



## 4.4 Optimization



- In contrast to Flink's built-in automatic optimization, **Spark jobs** have to be **manually optimized** and adapted to specific datasets because you need to manually control partitioning and caching if you want to get it right.
- **Spark SQL** uses the **Catalyst optimizer** that supports both rule-based and **cost-based optimization**. References:
  - ✓ Spark SQL: Relational Data Processing in Spark [http://people.csail.mit.edu/matei/papers/2015/sigmod\\_spark\\_sql.pdf](http://people.csail.mit.edu/matei/papers/2015/sigmod_spark_sql.pdf)
  - ✓ Deep Dive into Spark SQL's Catalyst Optimizer <https://databricks.com/blog/2015/04/13/deep-dive-into-spark-sqls-catalyst-optimizer.html>

## 4.5. Configuration



- **Flink** requires **no memory thresholds** to **configure**
  - ✓ **Flink** manages its own **memory**
- **Flink** requires **no complicated network configurations**
  - ✓ **Pipelining engine** requires much less memory for data exchange
- **Flink** requires **no serializers to be configured**
  - ✓ **Flink** handles its own **type extraction** and data representation

## 4.6. Tuning

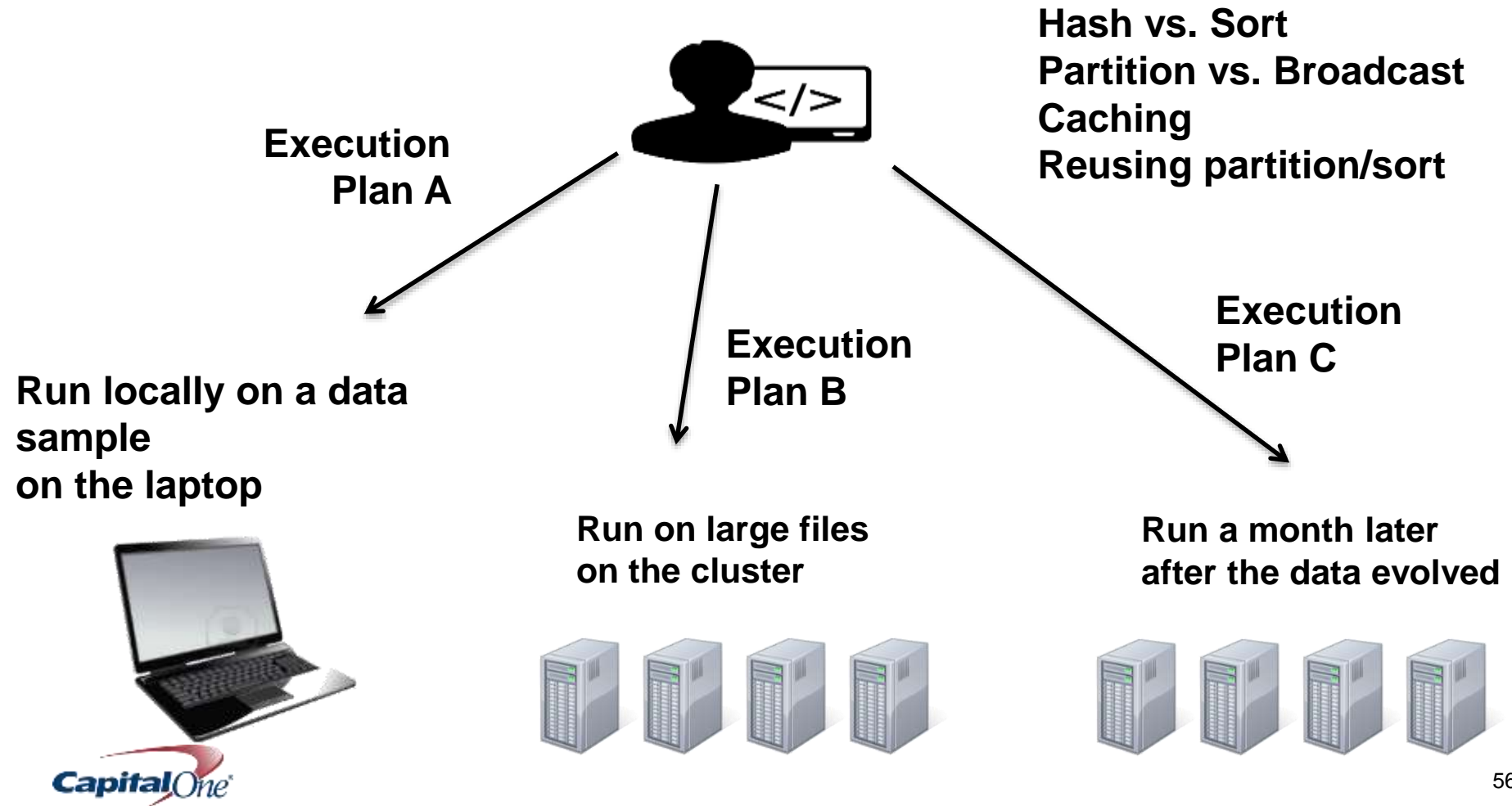


- According to **Mike Olsen**, Chief Strategy Officer of **Cloudera** Inc. “**Spark is too knobby — it has too many tuning parameters**, and they need constant adjustment as workloads, data volumes, user counts change. Reference: <http://vision.cloudera.com/one-platform/>
- Tuning Spark Streaming for Throughput By Gerard Maas from Virdata. December 22, 2014  
<http://www.virdata.com/tuning-spark/>
- Spark Tuning:  
<http://spark.apache.org/docs/latest/tuning.html>

## 4.6. Tuning



What is **Automatic Optimization**? The system's built-in optimizer takes care of finding the best way to execute the program in any environment.





# 7. Performance



- Why Flink provides a better performance?
  - ✓ Custom memory manager
  - ✓ Native closed-loop iteration operators make graph and machine learning applications run much faster.
  - ✓ Role of the built-in automatic optimizer. For example: more efficient join processing.
  - ✓ Pipelining data to the next operator in Flink is more efficient than in Spark.

- See benchmarking results against Flink here:

<http://www.slideshare.net/sbaltagi/why-apache-flink-is-the-4g-of-big-data-analytics-frameworks/87>



## 5. Future work

- The **framework from Capital One to evaluate stream processing tools** is being refined and will be **published** at <http://www.capitalone.io/>
- The **assessment** of the major open source streaming tools will be **published** as well as a live document continuously updated by **Capital One**.
- I also have a work in progress on comparing Spark and Flink as **multi-purpose Big Data analytics framework**
- Check **my blog** at <http://www.SparkBigData.com>
- Check also **my slide decks** on the Flink and Spark on <http://slideshare.net/sbaltagi>

# Agenda

**I. Motivation for this talk**

**II. Apache Flink vs. Apache Spark?**

**III. How Flink is used at Capital One?**

**IV. What are some key takeaways?**

# III. How Flink is used at Capital One?

- We started our journey with Apache Flink at **Capital One** while researching and contrasting stream processing tools in the **Hadoop ecosystem** with a particular interest in the ones providing **real-time stream processing capabilities** and not just micro-batching as in Apache Spark.
- While learning more about Apache Flink, we discovered some **unique capabilities** of Flink which differentiate it from other Big Data analytics tools not only **for Real-Time streaming** but also for **Batch processing**.
- **We evaluated Apache Flink Real-Time stream processing capabilities in a POC.**

### III. How Apache Flink is used at Capital One?

- Where are we in our Flink journey?
  - ✓ Successful installation of Apache Flink 0.9 in our Pre-Production cluster running on CDH 5.4 with security and High Availability enabled.
  - ✓ Successful installation of Apache Flink 0.9 in a 10 nodes R&D cluster running HDP.
  - ✓ Successful completion of Flink POC for real-time stream processing. The POC proved that propriety system can be replaced by a combination of tools: Apache Kafka, Apache Flink, Elasticsearch and Kibana in addition to advanced real-time streaming analytics.

### III. How Apache Flink is used at Capital One?

➤ What are the **opportunities** for using Apache Flink at Capital One?

1. Real-Time streaming analytics
2. Cascading on Flink
3. Flink's MapReduce Compatibility Layer
4. Flink's Storm Compatibility Layer
5. Other Flink libraries (**Machine Learning** and **Graph processing**) once they come out of beta.

# III. How Apache Flink is used at Capital One?

## ➤ Cascading on Flink:

- ✓ **First release of Cascading on Flink** was announced recently by Data Artisans and Concurrent. It will be supported in upcoming Cascading 3.1.
- ✓ **Capital One** is the **first** company **verifying** this release on **real-world Cascading data flows** with a **simple configuration switch** and **no code re-work needed!**
- ✓ **This is a good example of doing analytics on bounded data sets (Cascading) using a stream processor (Flink)**
- ✓ **Expected advantages** of performance boost and less resource consumption.
- ✓ **Future work is to support 'Driven' from Concurrent Inc. to provide performance management for Cascading data flows running on Flink.**

### III. How Apache Flink is used at Capital One?

- Flink's **compatibility layer for Storm**:
  - ✓ We can **execute existing Storm topologies** using Flink as the underlying engine.
  - ✓ We can **reuse** our application code (**bolts** and **spouts**) inside Flink programs.
- Flink's libraries (**FlinkML** for **Machine Learning** and **Gelly** for Large scale **graph processing**) can be used along Flink's DataStream API and DataSet API for our end to end big data analytics needs.



# Agenda

**I. Motivation for this talk**

**II. Apache Flink vs. Apache Spark?**

**III. How Flink is used at Capital One?**

**IV. What are some key takeaways?**

# III. What are some key takeaways?

- Neither Flink nor Spark will be the single analytics framework that will solve every Big Data problem!
- By design, Spark is not for real-time stream processing while Flink provides a true low latency streaming engine and advanced DataStream API for real-time streaming analytics.
- Although Spark is ahead in popularity and adoption, Flink is ahead in technology innovation and is growing fast.
- It is not always the most innovative tool that gets the largest market share, the Flink community needs to take into account the market dynamics!
- Both Spark and Flink will have their sweet spots despite their “Me too syndrome”.

# Thanks!

- To **all of you** for attending!
- To **Capital One** for giving me the opportunity to meet with the growing Apache Flink family.
- To the **Apache Flink community** for the great spirit of collaboration and help.
- **2016 will be the year of Apache Flink!**
- **See you at FlinkForward 2016!**