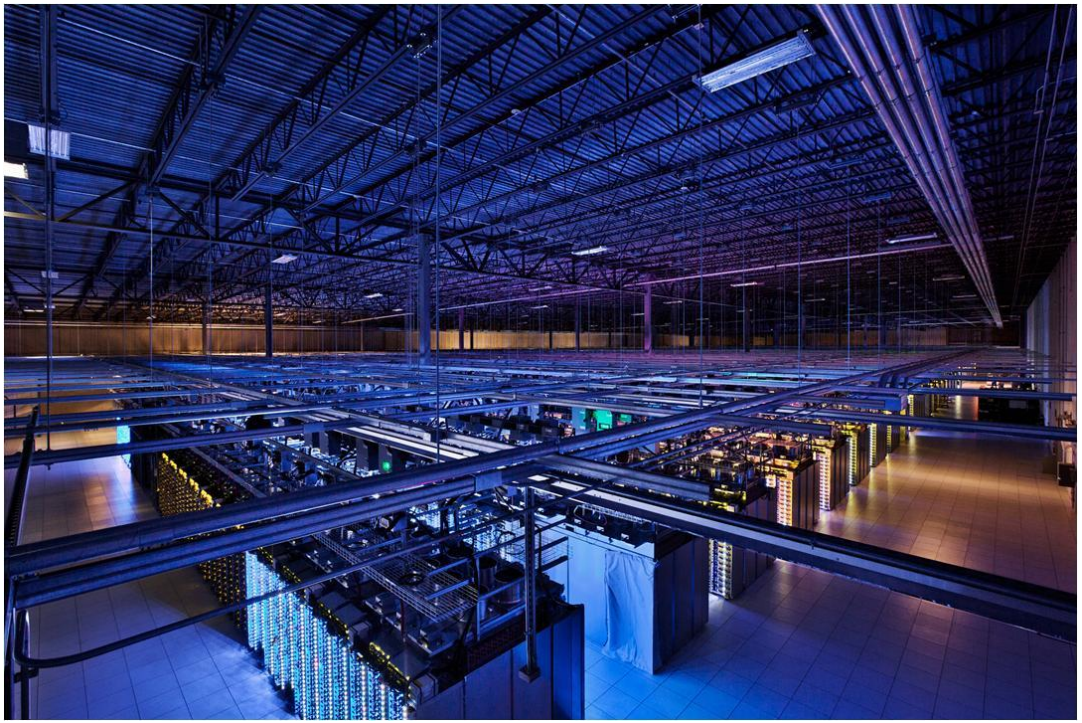
CWI
Centrum Wiskunde & Informatica

# Large-Scale Data Engineering

## Spark and MLLIB



# OVERVIEW OF SPARK

# What is Spark?

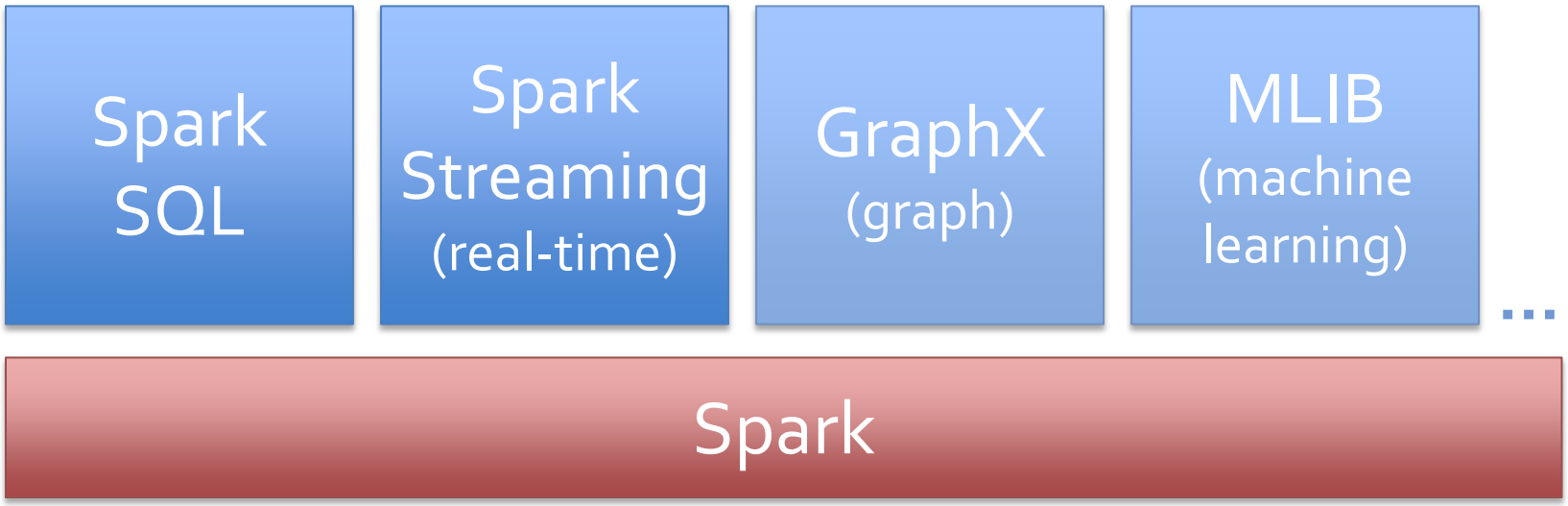
- Fast and expressive cluster computing system interoperable with Apache Hadoop
- Improves efficiency through:
  - In-memory computing primitives
  - General computation graphs
- Improves usability through:
  - Rich APIs in Scala, Java, Python
  - Interactive shell

→ Up to 100 × faster (2-10 × on disk)

→ Often 5 × less code

# The Spark Stack

- Spark is the basis of a wide set of projects in the Berkeley Data Analytics Stack (BDAS)

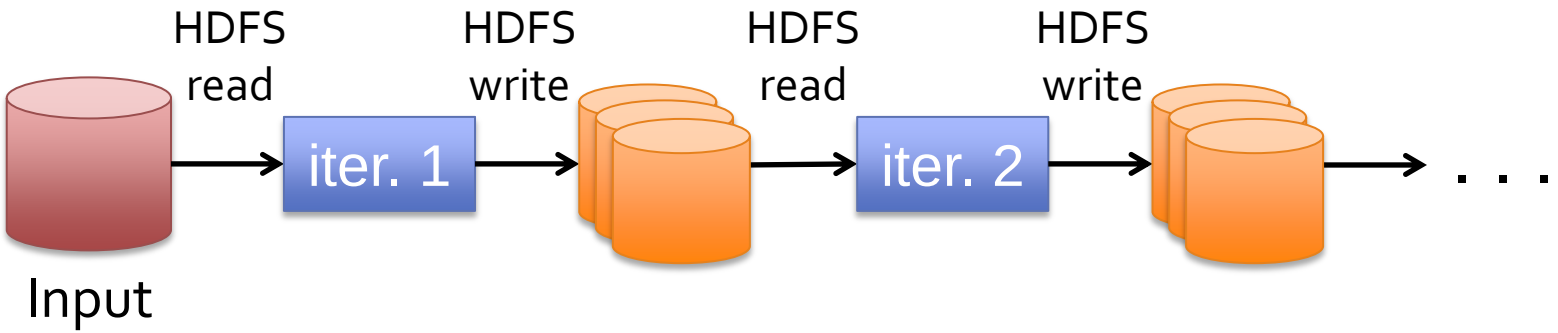


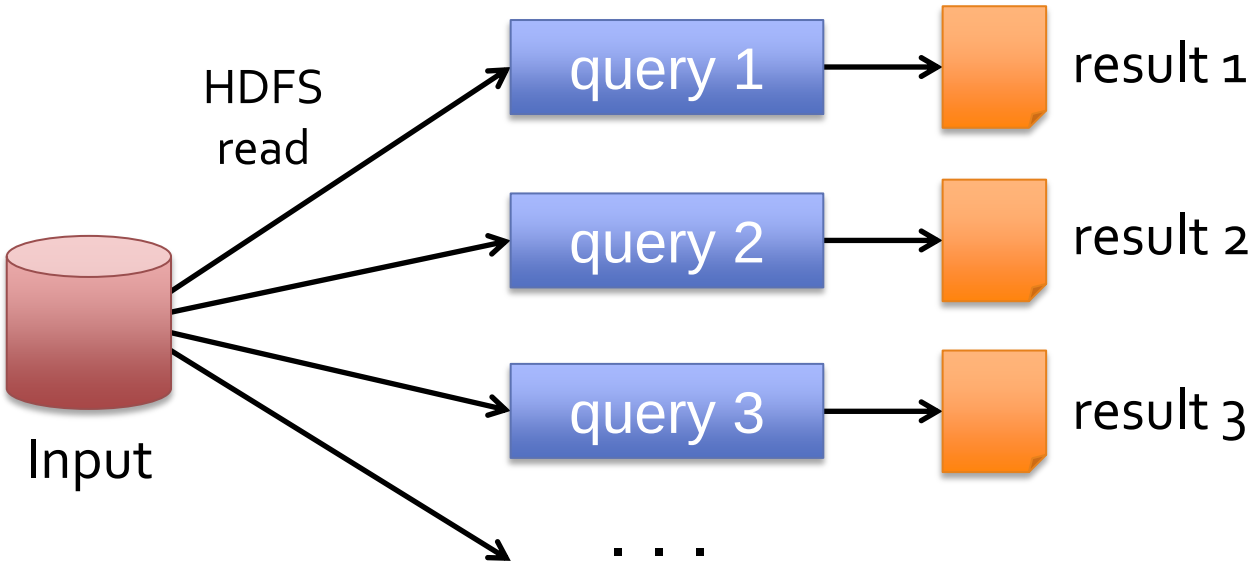

More details: amplab.berkeley.edu

# Why a New Programming Model?

- MapReduce greatly simplified big data analysis
- But as soon as it got popular, users wanted more:
  - More **complex**, multi-pass analytics (e.g. ML, graph)
  - More **interactive** ad-hoc queries
  - More **real-time** stream processing
- All 3 need faster **data sharing** across parallel jobs

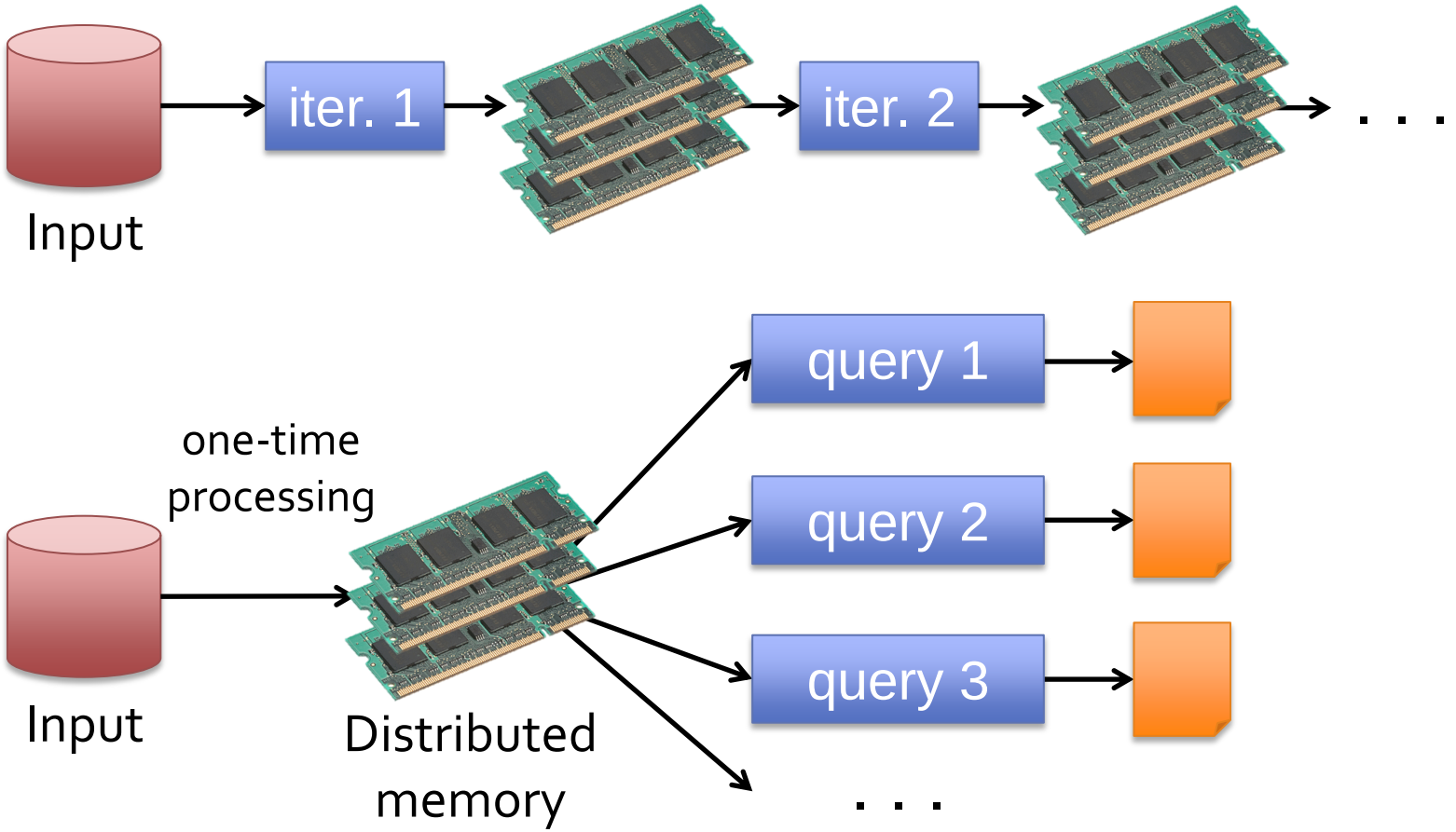
# Data Sharing in MapReduce




**Slow** due to replication, serialization, and disk IO

# Data Sharing in Spark




**~10×** faster than network and disk

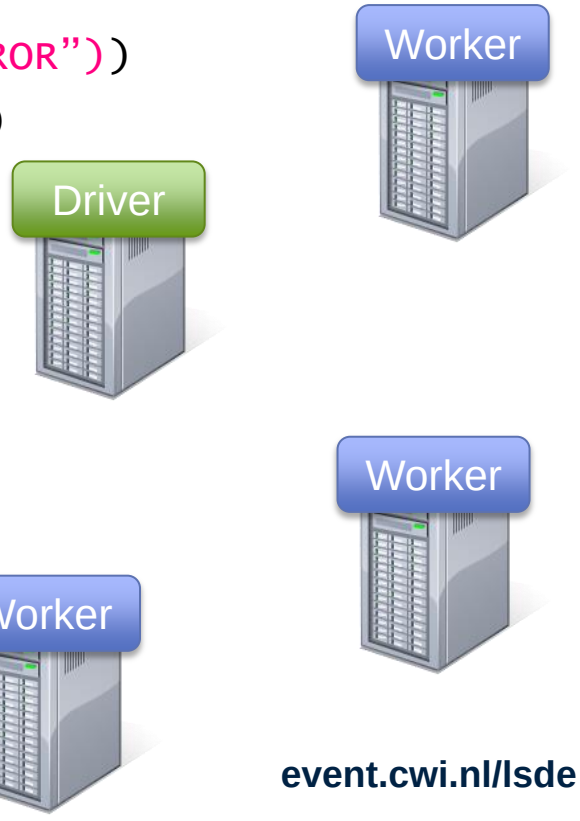
# Spark Programming Model

- Key idea: *resilient distributed datasets (RDDs)*
  - Distributed collections of objects that can be cached in memory across the cluster
  - Manipulated through parallel operators
  - Automatically *recomputed* on failure
- Programming interface
  - Functional APIs in Scala, Java, Python
  - Interactive use from Scala shell

# Example: Log Mining

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

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda x: x.startswith("ERROR"))
messages = errors.map(lambda x: x.split('\t')[2])
messages.cache()
```

# Lambda Functions

```
errors = lines.filter(lambda x: x.startswith("ERROR"))
messages = errors.map(lambda x: x.split('\t')[2])
```

Lambda function ← functional programming!

= implicit function definition

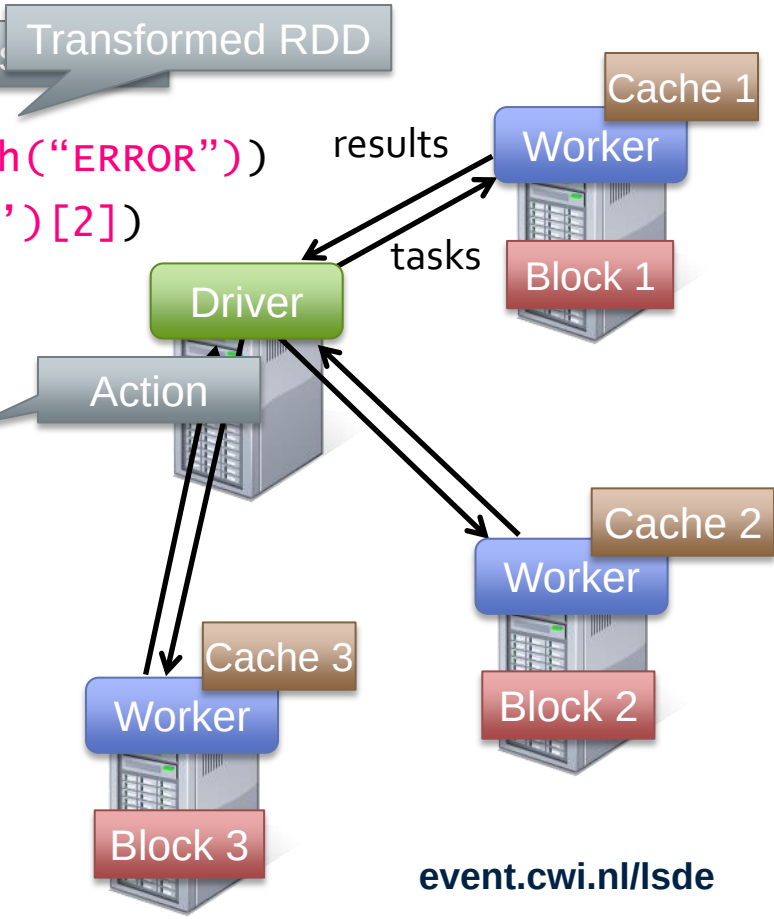
```
bool detect_error(string x) {
    return x.startswith("ERROR");
}
```

# Example: Log Mining

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

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda x: x.startswith("ERROR"))
messages = errors.map(lambda x: x.split('\t')[2])
messages.cache()

messages.filter(lambda x: "foo" in x).count
messages.filter(lambda x: "bar" in x).count
```



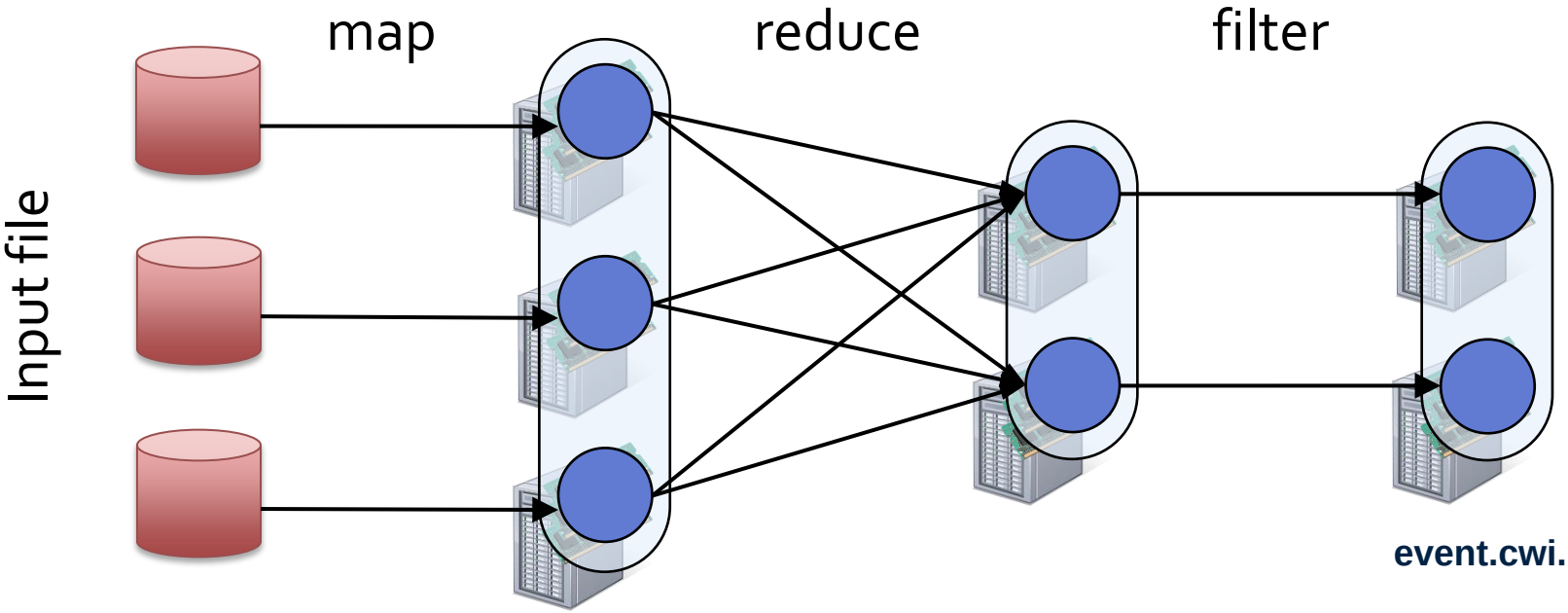

**Result:** scaled to 1 TB data in 5-7 sec (vs 170 sec for on-disk data)

**Result:** full-text search of Wikipedia in <1 sec (vs 20 sec for on-disk data)

# Fault Tolerance

RDDs track *lineage* info to rebuild lost data

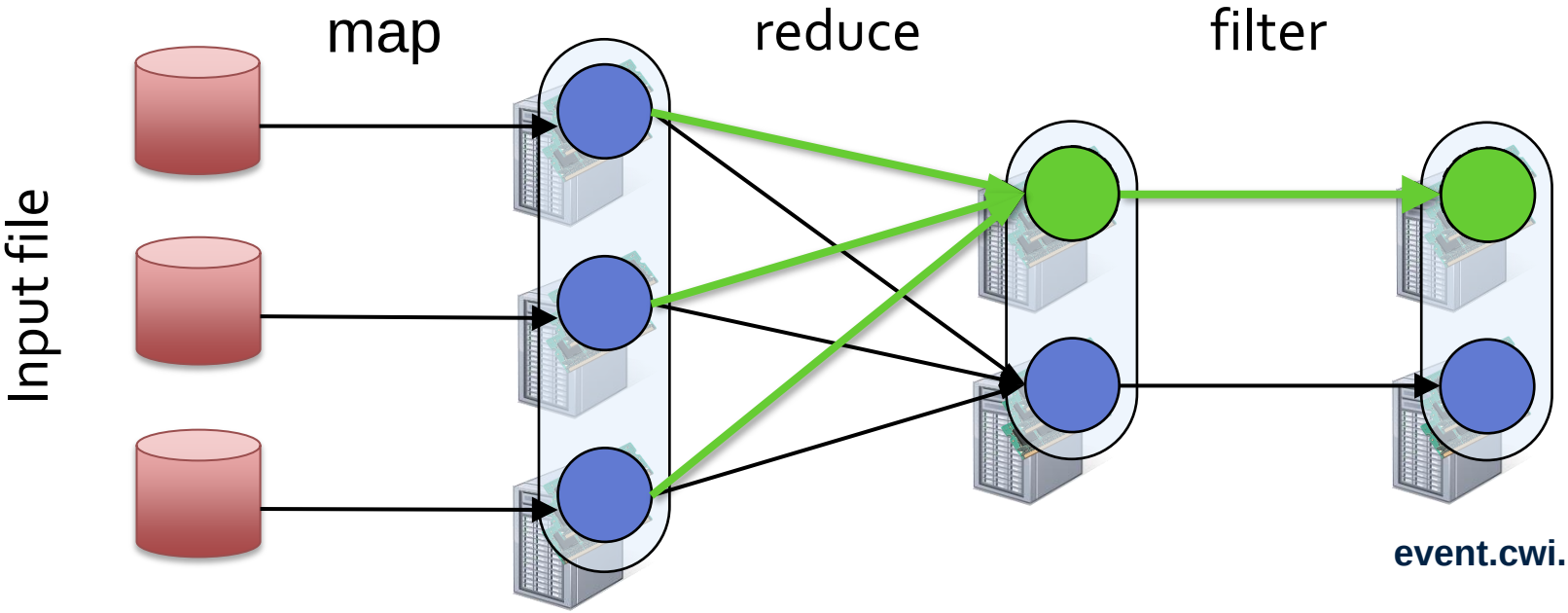
```
file.map(lambda rec: (rec.type, 1))
    .reduceByKey(lambda x, y: x + y)
    .filter(lambda (type, count): count > 10)
```



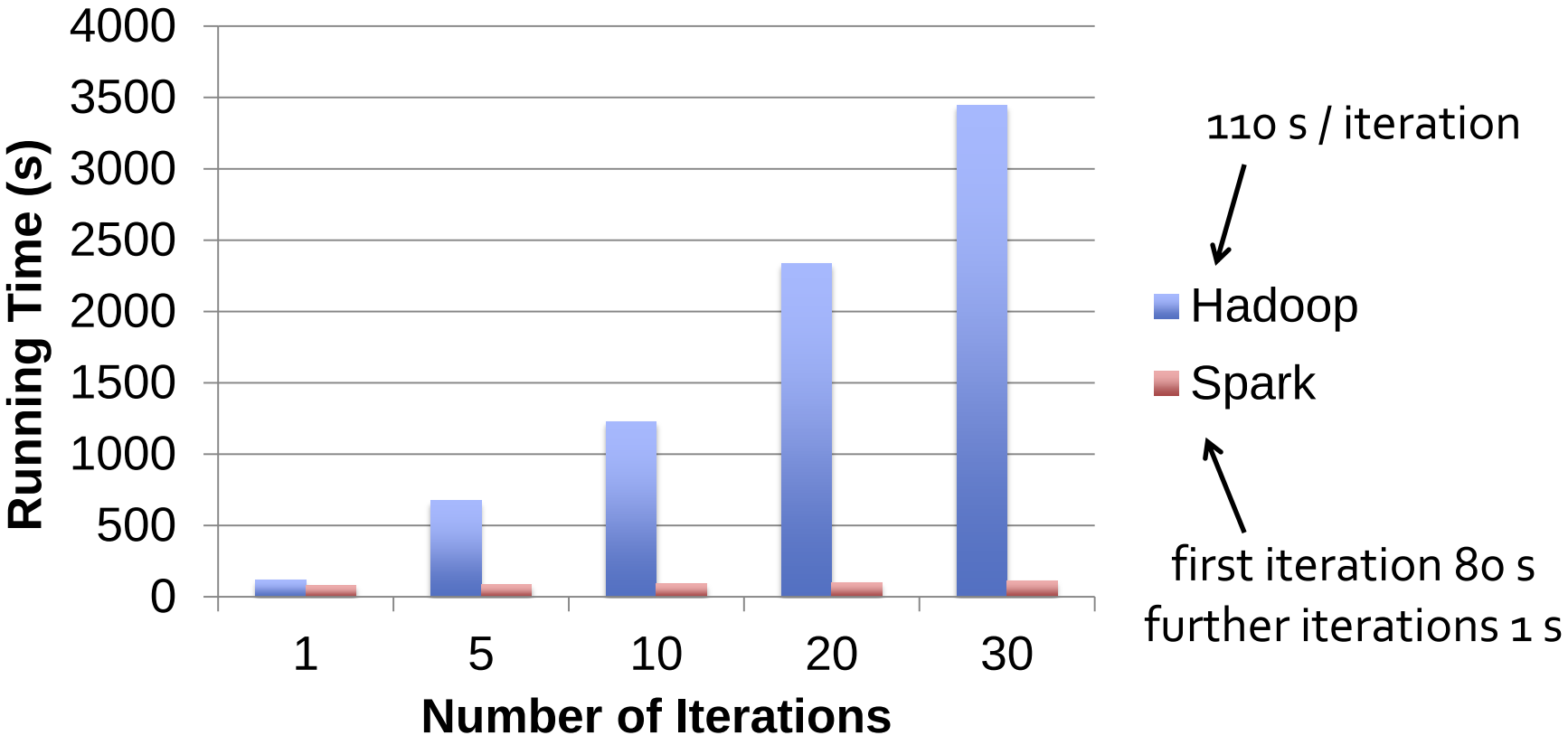
# Fault Tolerance

RDDs track *lineage* info to rebuild lost data

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



# Example: Logistic Regression



# Spark in Scala and Java

```scala
// Scala:

val lines = sc.textFile(...)
lines.filter(x => x.contains("ERROR")).count()
```

```java
// Java:

JavaRDD<String> lines = sc.textFile(...);
lines.filter(new Function<String, Boolean>() {
  Boolean call(String s) {
    return s.contains("error");
  }
}).count();
```

# Supported Operators

- `map`
- `filter`
- `groupBy`
- `sort`
- `union`
- `join`
- `leftOuterJoin`
- `rightOuterJoin`
- `reduce`
- `count`
- `fold`
- `reduceByKey`
- `groupByKey`
- `cogroup`
- `cross`
- `zip`

`sample`

`take`

`first`

`partitionBy`

`mapWith`

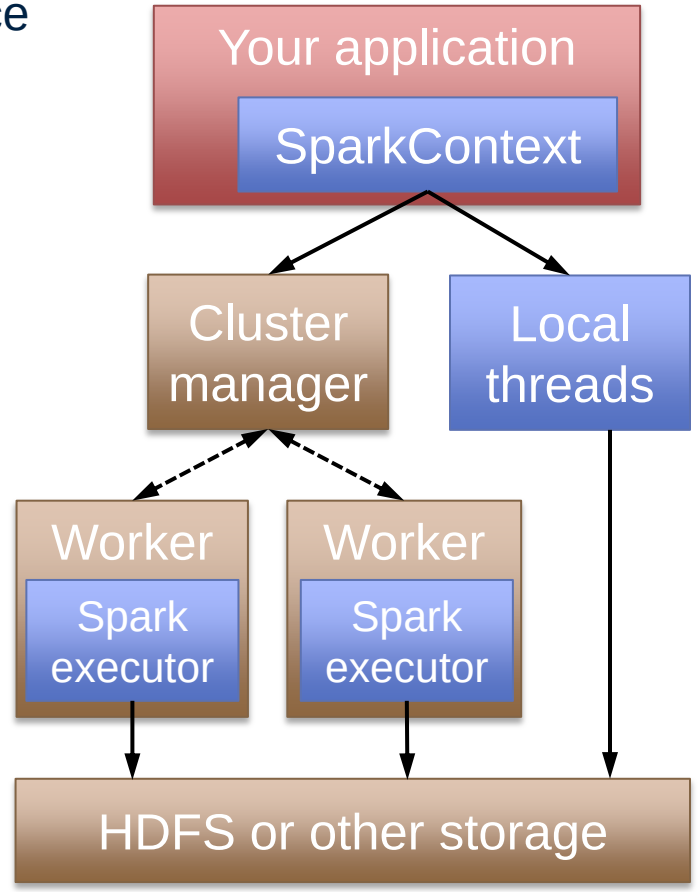
`pipe`

`save`

`...`

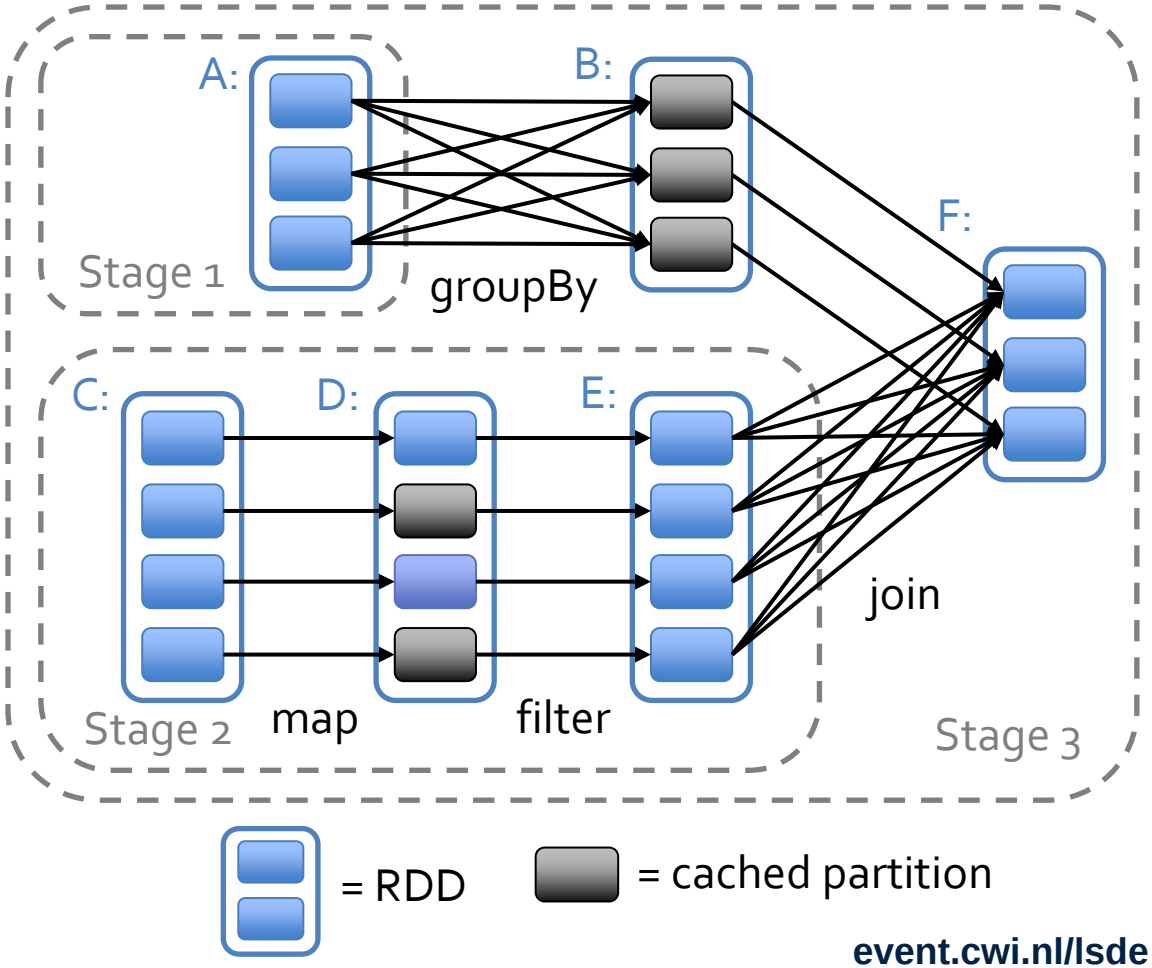
# Software Components

- Spark client is library in user program (1 instance per app)
- Runs tasks locally or on cluster
  - Mesos, YARN, standalone mode
- Accesses storage systems via Hadoop InputFormat API
  - Can use HBase, HDFS, S3, …



# Task Scheduler

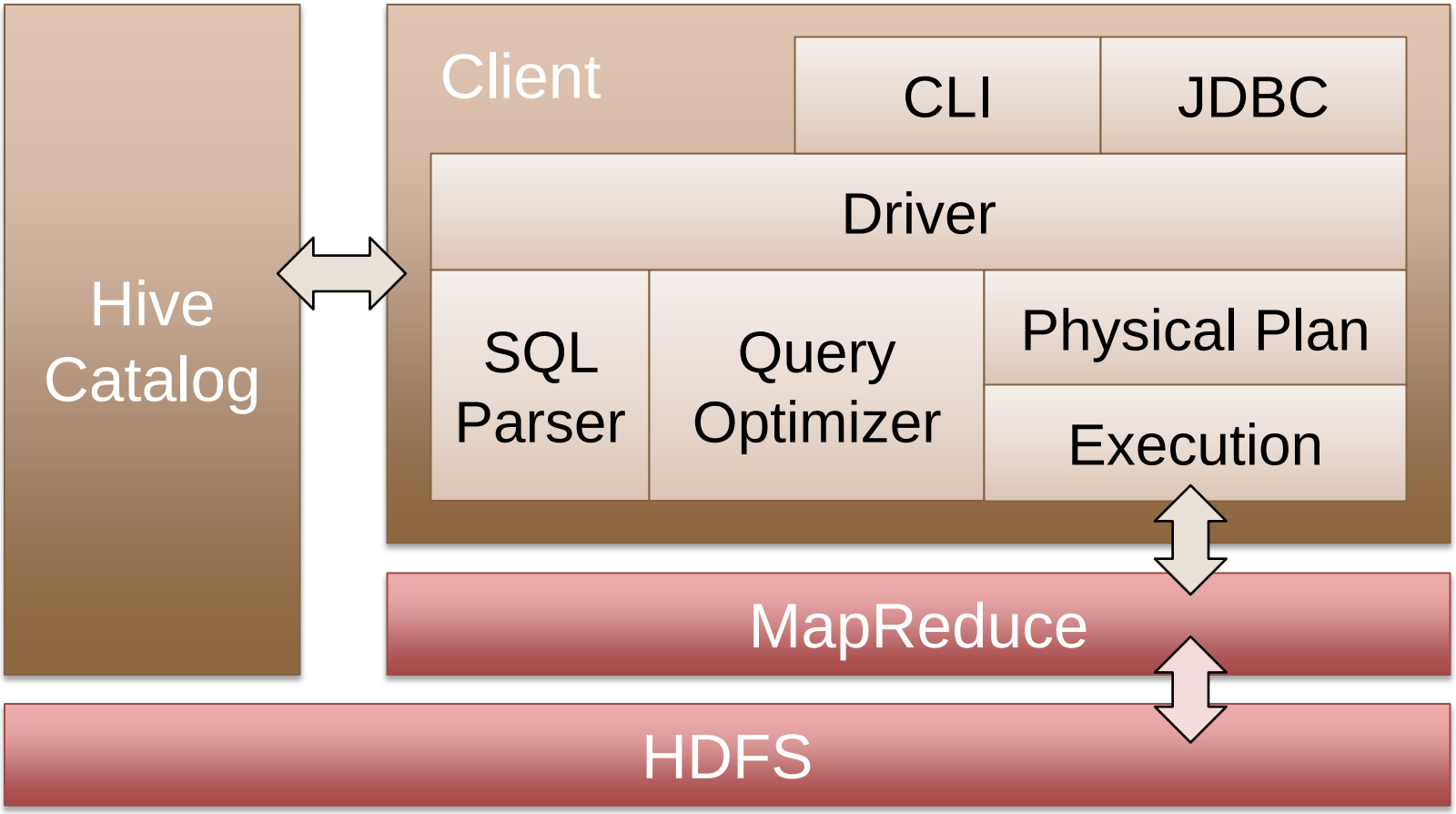
- General task graphs
- Automatically pipelines functions
- Data locality aware
- Partitioning aware to avoid shuffles



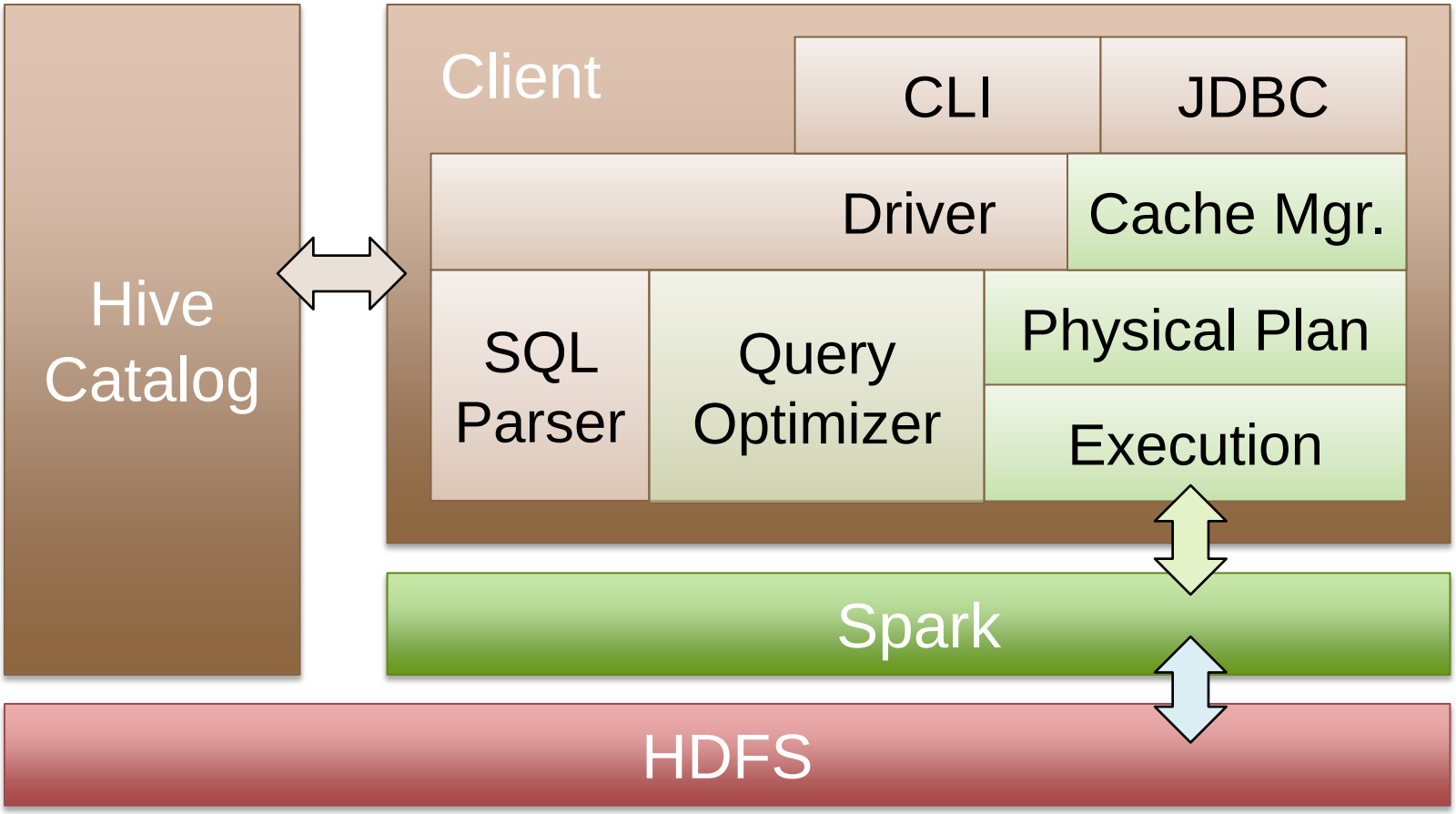
# Spark SQL

- Columnar SQL analytics engine for Spark
  - Support both SQL and complex analytics
  - Up to 100X faster than Apache Hive
- Compatible with Apache Hive
  - HiveQL, UDF/UDAF, SerDes, Scripts
  - Runs on existing Hive warehouses
- In use at Yahoo! for fast in-memory OLAP

# Hive Architecture



# Spark SQL Architecture




[Engle et al, SIGMOD 2012]

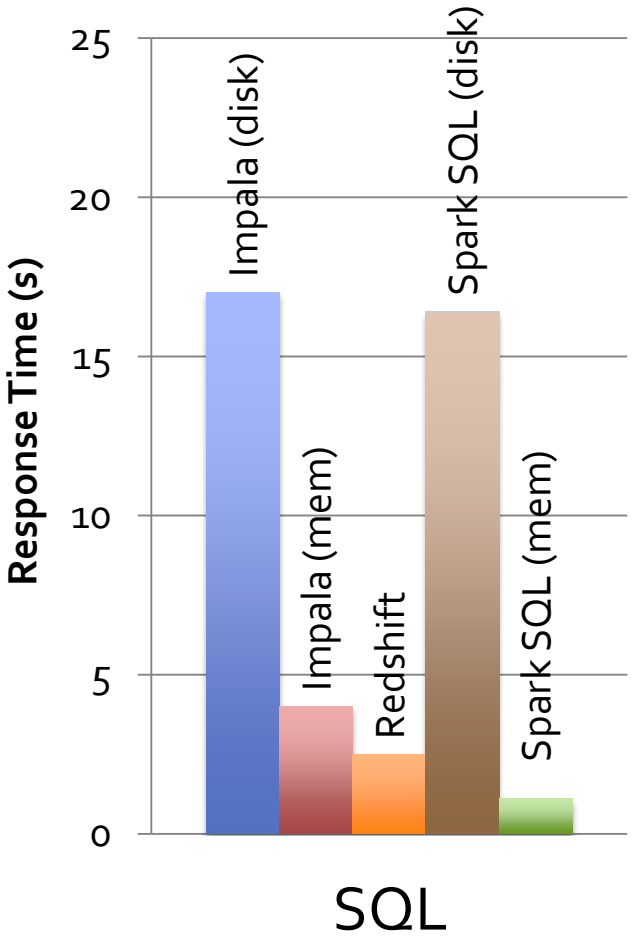
# What Makes it Faster?

- Lower-latency engine (Spark OK with 0.5s jobs)
- Support for general DAGs
- Column-oriented storage and compression
- New optimizations (e.g. map pruning)

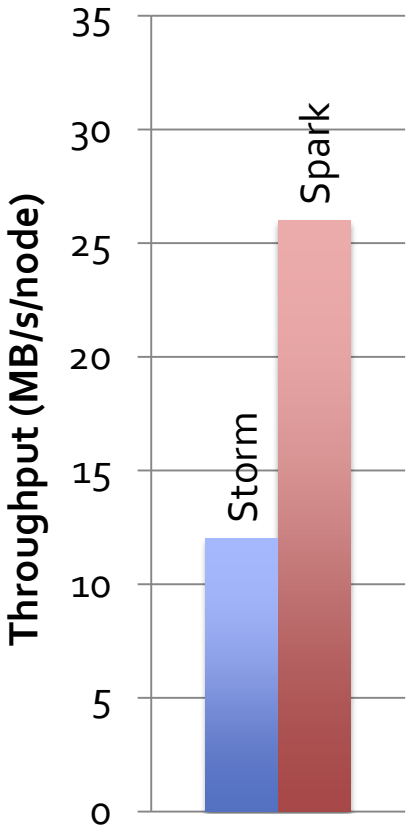
# Other Spark Stack Projects

- **Spark Streaming:** stateful, fault-tolerant stream processing (out since Spark 0.7)
- ```
  sc.twitterStream(...)
    .flatMap(_.getText.split(" "))
    .map(word => (word, 1))
    .reduceByWindow("5s", _ + _)
  ```
- **MLlib:** Library of high-quality machine learning algorithms (out since 0.8)
  ```

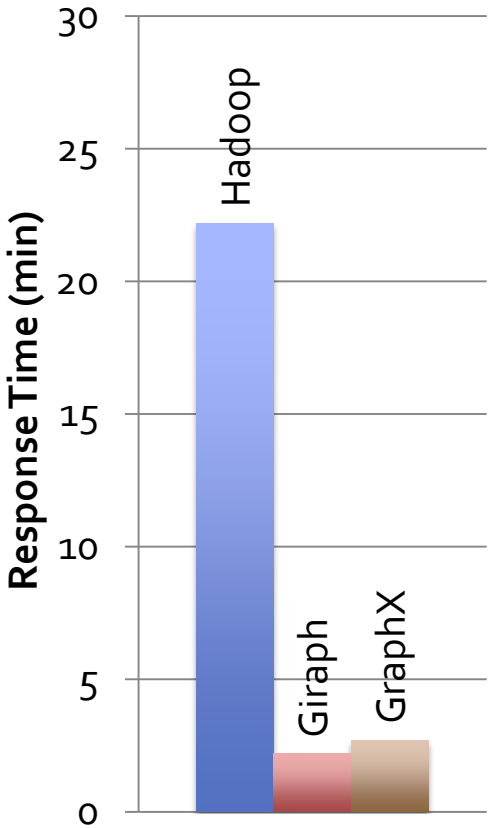
# Performance




SQL




Streaming

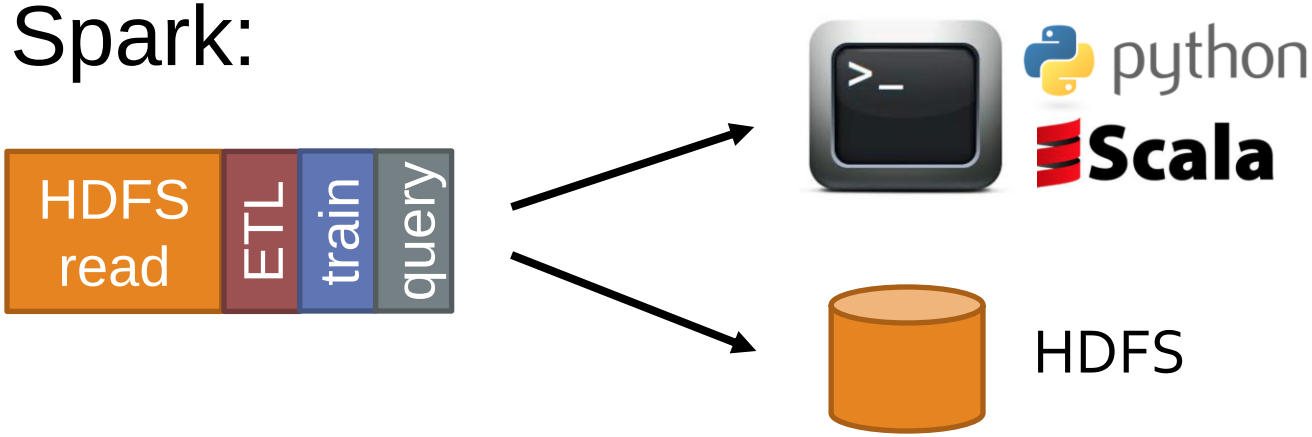



Graph

# What it Means for Users

- Separate frameworks:


Spark:



# Conclusion

- Big data analytics is evolving to include:
  - More **complex** analytics (e.g. machine learning)
  - More **interactive** ad-hoc queries
  - More **real-time** stream processing
- Spark is a fast platform that *unifies* these apps
- More info: spark-project.org

# SPARK MLLIB

# What is MLLIB?

MLlib is a Spark subproject providing machine learning primitives:

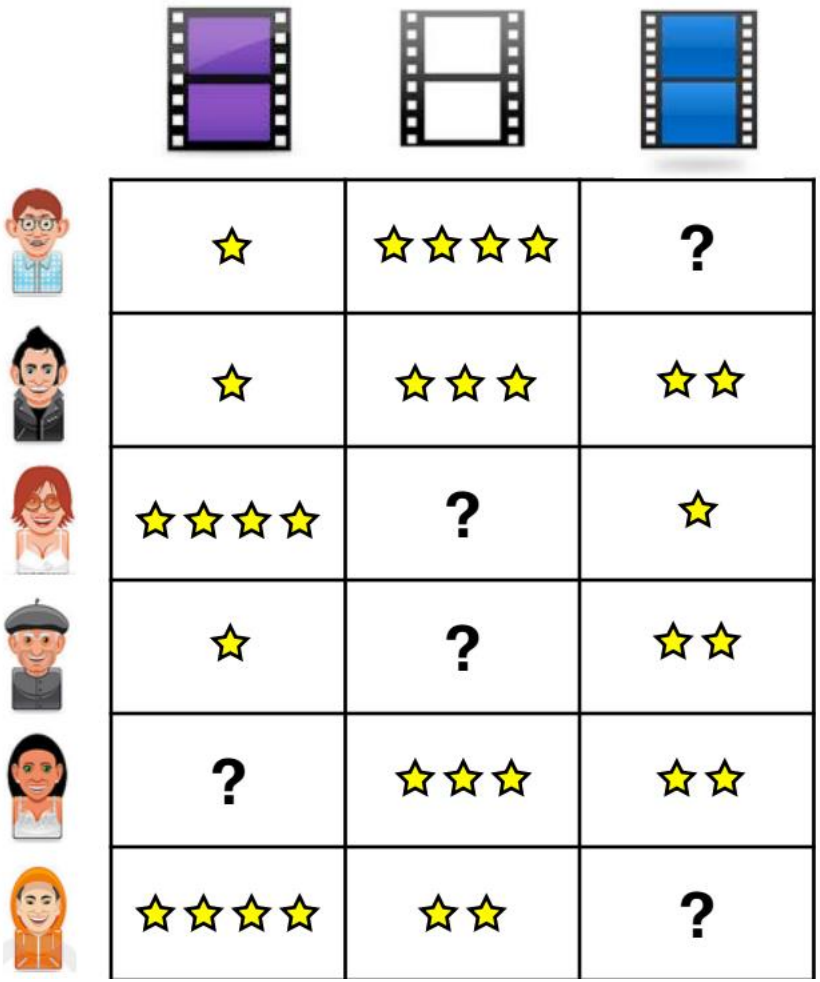
- initial contribution from AMPLab, UC Berkeley
- shipped with Spark since version 0.8

# What is MLLIB?

**Algorithms**:

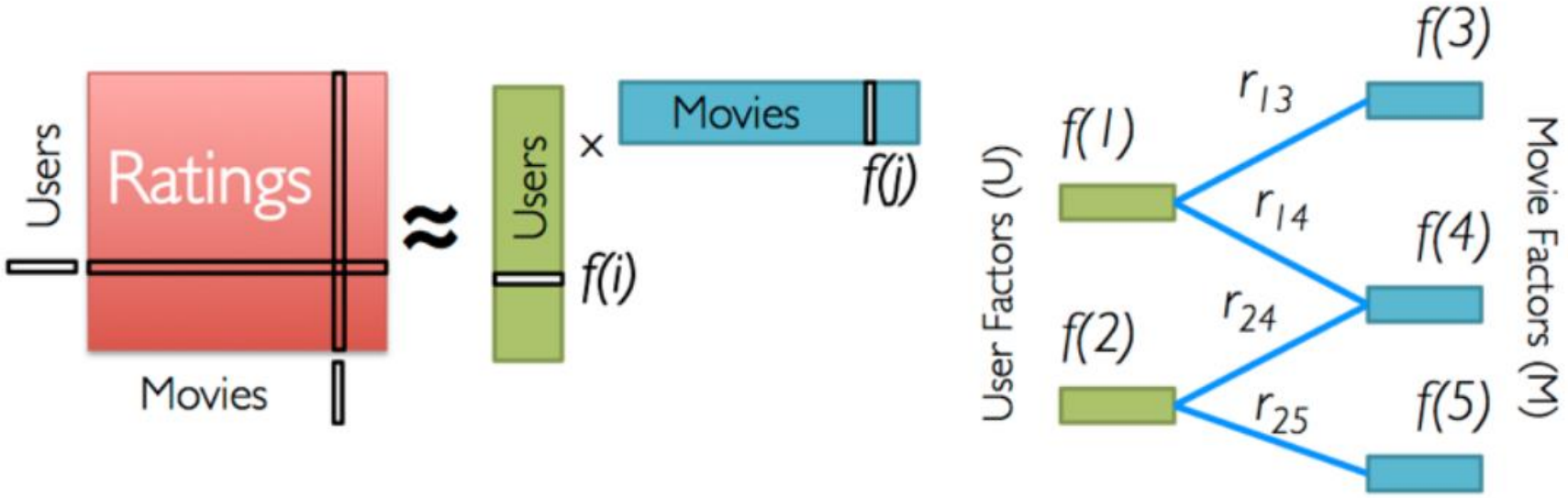
- **classification**: logistic regression, linear support vector machine (SVM), naive Bayes
- **regression**: generalized linear regression (GLM)
- **collaborative filtering**: alternating least squares (ALS)
- **clustering**: k-means
- **decomposition**: singular value decomposition (SVD), principal component analysis (PCA)

# Collaborative Filtering




- Recover a rating matrix from a subset of its entries.

# Alternating Least Squares (ALS)




Iterate:

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

# Collaborative Filtering in Spark MLLIB

```
trainset =
  sc.textFile("s3n://bads-music-dataset/train_*.gz")
    .map(lambda l: l.split('\t'))
    .map(lambda l: Rating(int(l[0]), int(l[1]), int(l[2])))

model = ALS.train(trainset, rank=10, iterations=10) # train

testset =                                    # load testing set
  sc.textFile("s3n://bads-music-dataset/test_*.gz")
    .map(lambda l: l.split('\t'))
    .map(lambda l: Rating(int(l[0]), int(l[1]), int(l[2])))

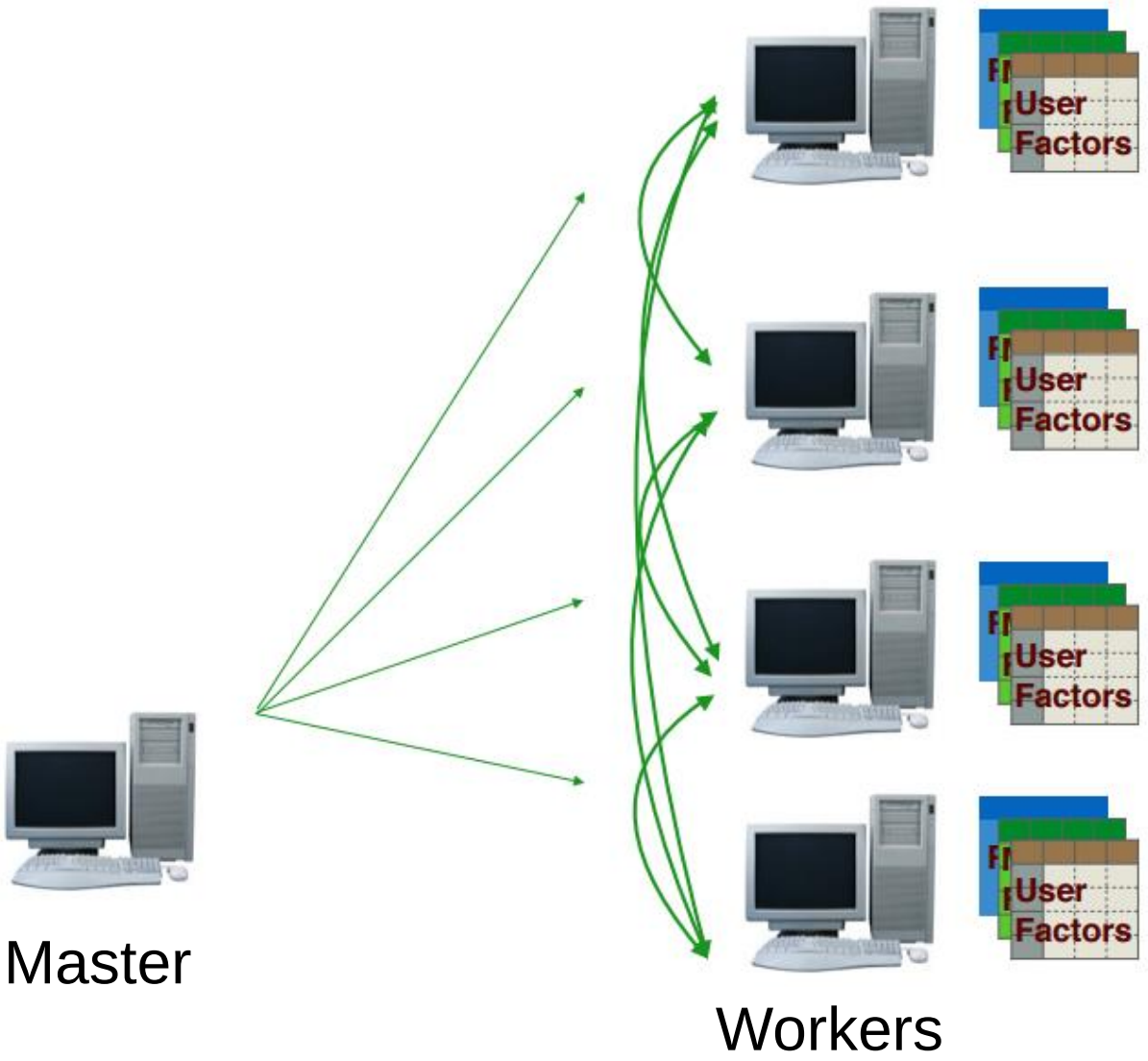
# apply model to testing set (only first two cols) to predict
predictions =
   model.predictAll(testset.map(lambda p: (p[0], p[1])))
        .map(lambda r: ((r[0], r[1]), r[2]))
```

# Spark MLLIB – ALS Performance

| System | Wall-clock /me (seconds) |
|---|---|
| Matlab | 15443 |
| Mahout | 4206 |
| GraphLab | 291 |
| MLlib | 481 |

- Dataset: Netflix data
- Cluster: 9 machines.
- MLlib is an order of magnitude faster than Mahout.
- MLlib is within factor of 2 of GraphLab.

# Spark Implementation of ALS




- Workers load data
- Models are instantiated at workers.
- At each iteration, models are shared via join between workers.
- Good scalability.
- Works on large datasets

# Spark SQL + MLLIB

```
// Data can easily be extracted from existing sources,
// such as Apache Hive.
val trainingTable = sql("""
  SELECT e.action,
         u.age,
         u.latitude,
         u.longitude
  FROM Users u
  JOIN Events e
  ON u.userId = e.userId""")

// Since `sql` returns an RDD, the results of the above
// query can be easily used in MLlib.
val training = trainingTable.map { row =>
  val features = Vectors.dense(row(1), row(2), row(3))
  LabeledPoint(row(0), features)
}
```

# MLLIB Pointers

- **Website**: http://spark.apache.org
- **Tutorials**: http://ampcamp.berkeley.edu
- **Spark Summit**: http://spark-summit.org
- **Github**: https://github.com/apache/spark
- **Mailing lists**: user@spark.apache.org dev@spark.apache.org