HADOOP 2.0: SPARK OVERVIEW

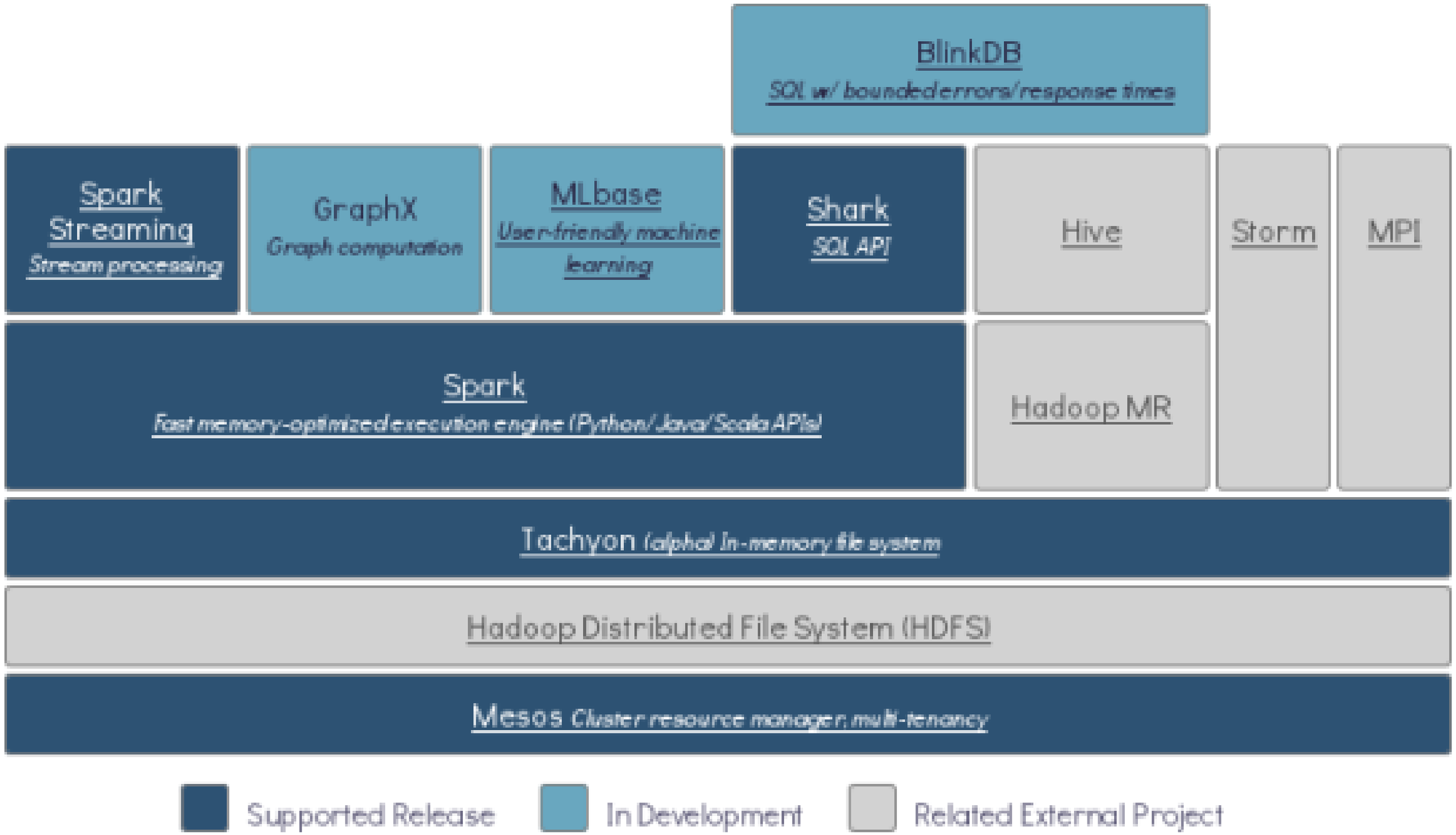
Вовченко Алексей,

alexey.vovchenko@gmail.com

к.т.н., с.н.с.

ВМК МГУ, ИПИ РАН

BDAS – Berkley Data Analytics Stack



DATA PROCESSING GOALS

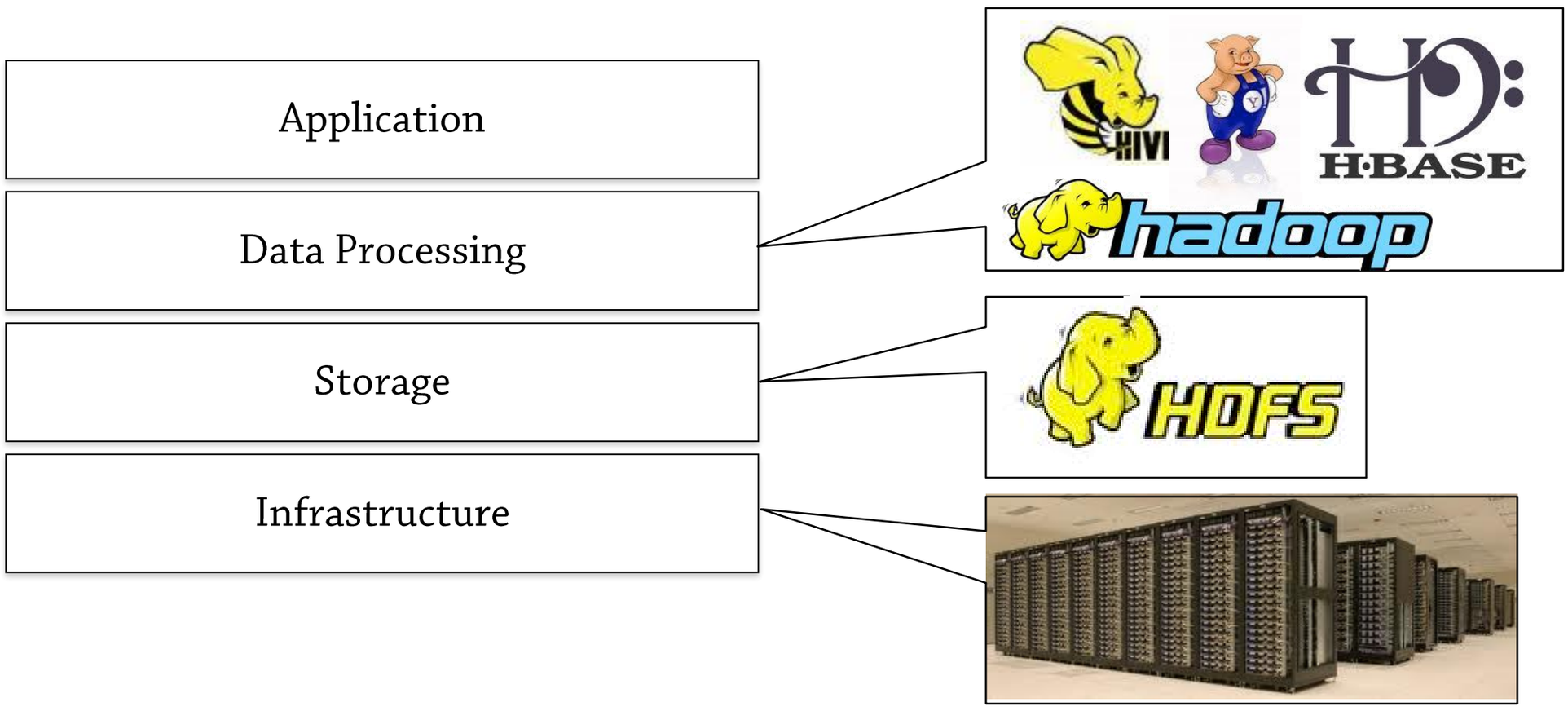
- **Low latency (interactive) queries on historical data:** enable faster decisions
  - E.g., identify why a site is slow and fix it

- **Low latency queries on live data (streaming):** enable decisions on real-time data
  - E.g., detect & block worms in real-time (a worm may infect **1mil** hosts in **1.3sec**)

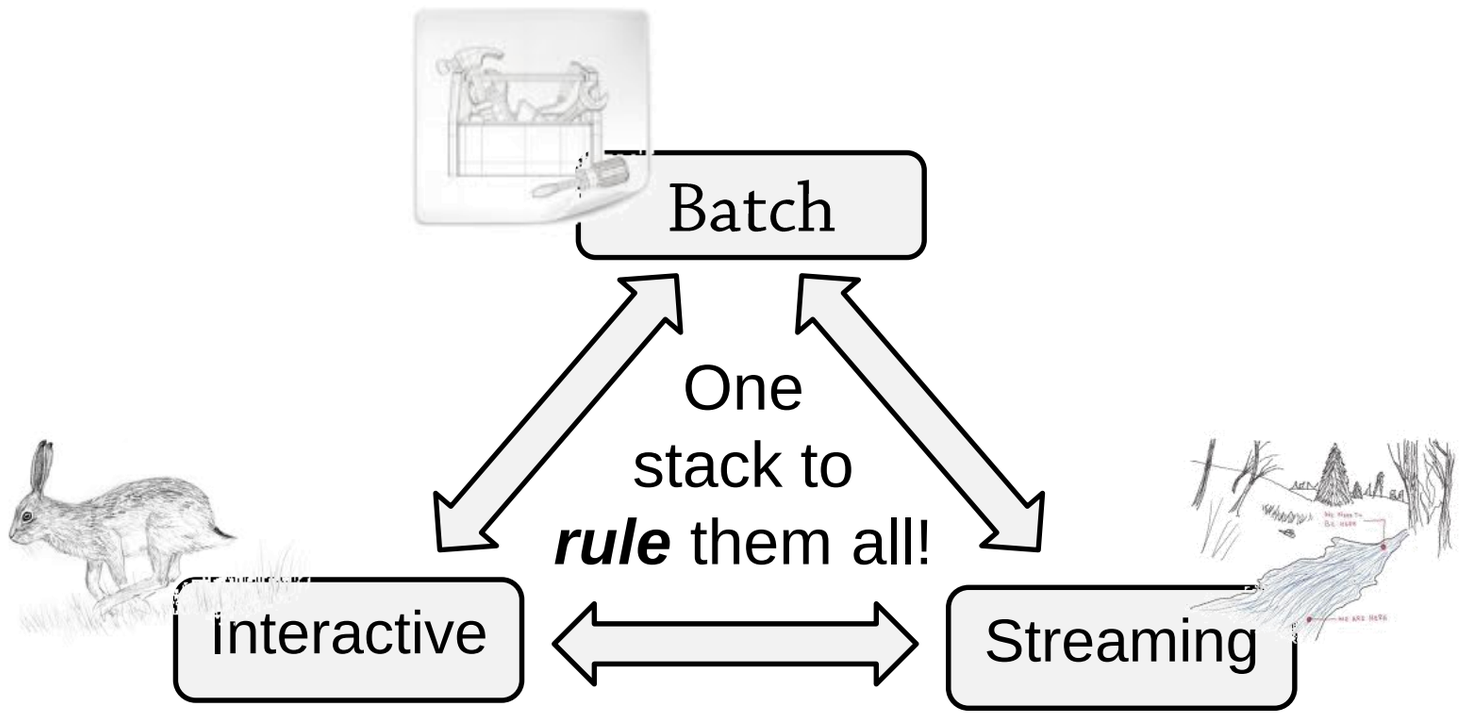
- **Sophisticated data processing:** enable "better" decisions
  - E.g., anomaly detection, trend analysis

TODAY'S OPEN ANALYTICS STACK…

- ..mostly focused on large on-disk datasets: great for ***batch*** but ***slow***



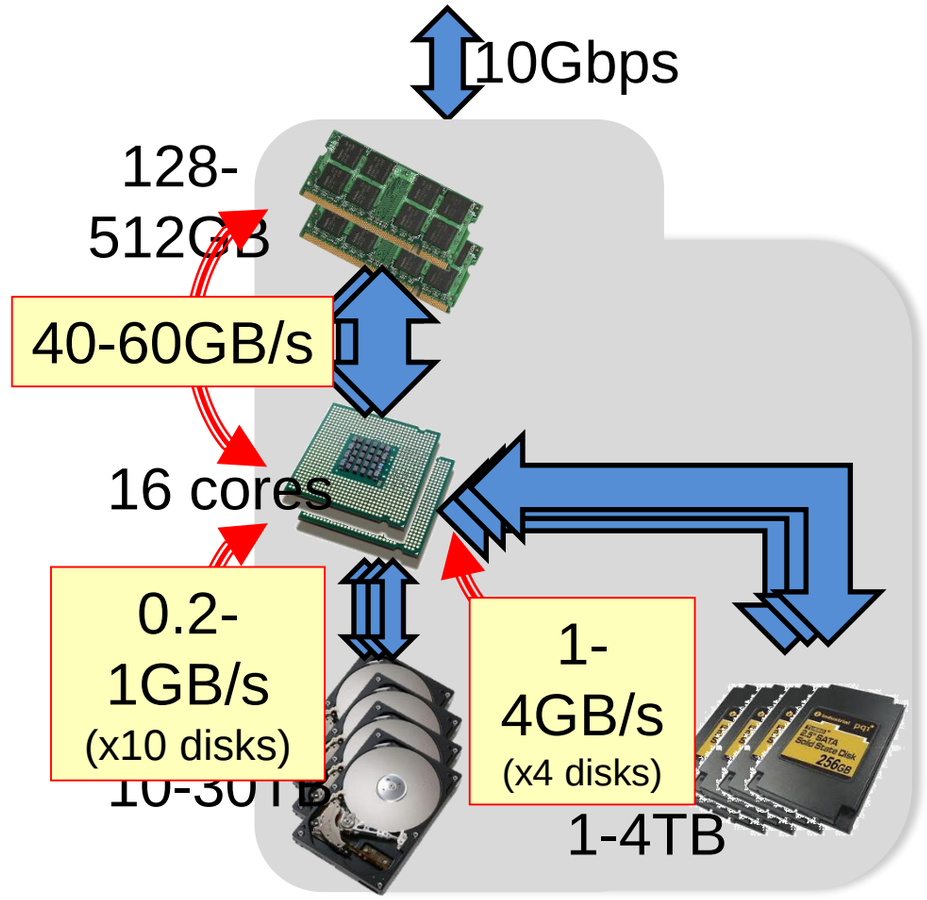
GOALS



- ***Easy*** to combine ***batch***, ***streaming***, and ***interactive***e computations
  - ***Easy*** to develop ***sophisticated*** algorithms
- ***Compatible*** with existing open source ecosystem (Hadoop/HDFS)

APPROACH: SUPPORT INTERACTIVE AND STREAMING COMP.

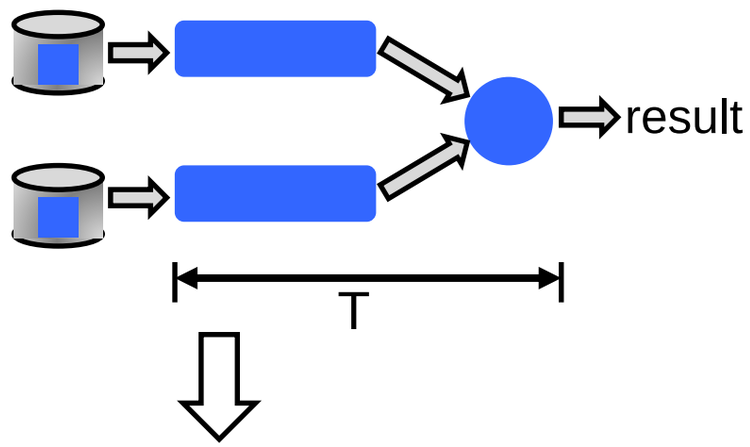
- Aggressive use of ***memory***
- Why?
  1. Memory transfer rates >> disk or even SSDs
     - Gap is growing especially w.r.t. disk
  2. Many datasets already fit into memory
     - The inputs of over 90% of jobs in Facebook, Yahoo!, and Bing clusters fit into memory
     - E.g., 1TB = 1 billion records @ 1 KB each
  3. Memory density (still) grows with Moore's law
     - RAM/SSD hybrid memories at horizon



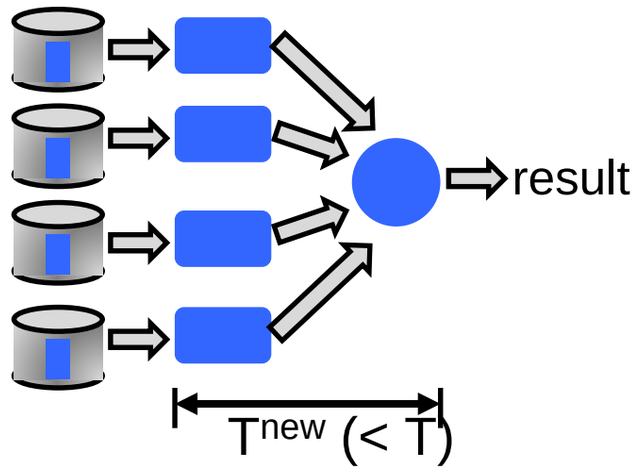
High end datacenter node

APPROACH: SUPPORT INTERACTIVE AND STREAMING COMP.

- Increase ***parallelism***
- Why?
  - Reduce work per node → improve latency

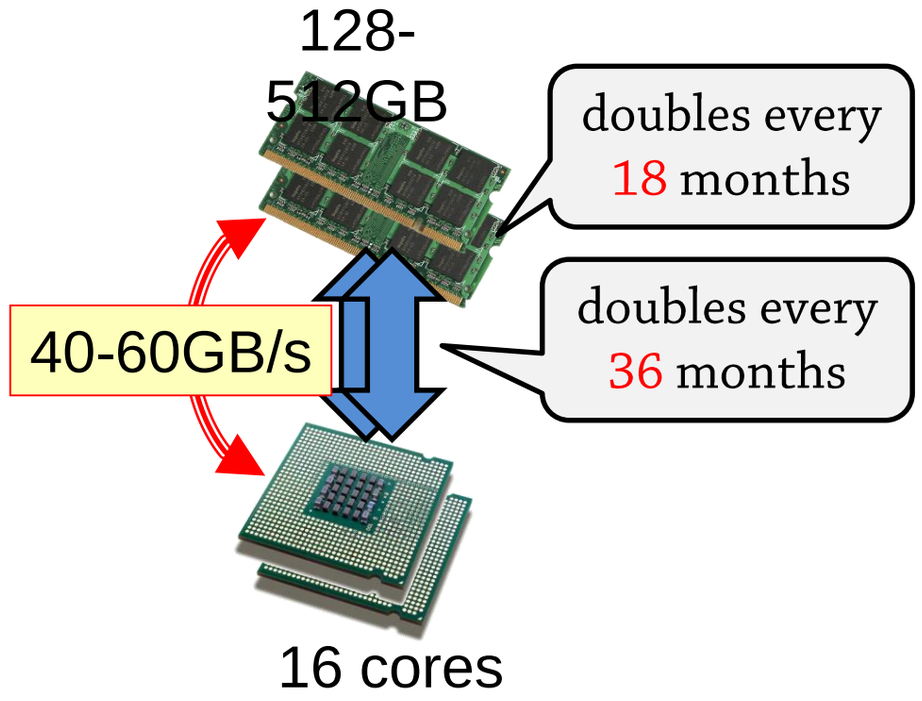


- Techniques:
  - Low latency parallel scheduler that achieve high locality
  - Optimized parallel communication patterns (e.g., shuffle, broadcast)
  - Efficient recovery from failures and straggler mitigation



APPROACH: SUPPORT INTERACTIVE AND STREAMING COMP.

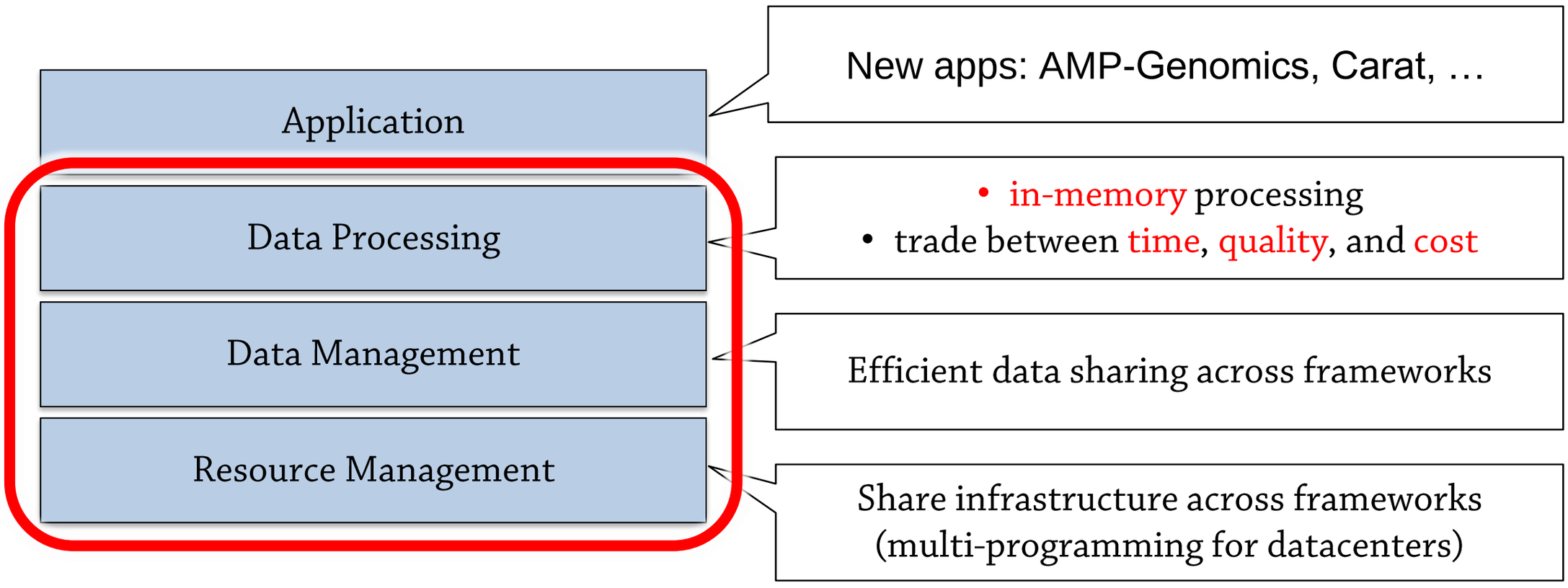
- Trade between result ***accuracy*** and ***response times***
- Why?
  - In-memory processing does not guarantee interactive query processing
    - E.g., ~10's sec just to scan 512 GB RAM!
    - Gap between memory capacity and transfer rate increasing
- Challenges:
  - accurately estimate error and running time for…
  - … arbitrary computations



BDAS Approach

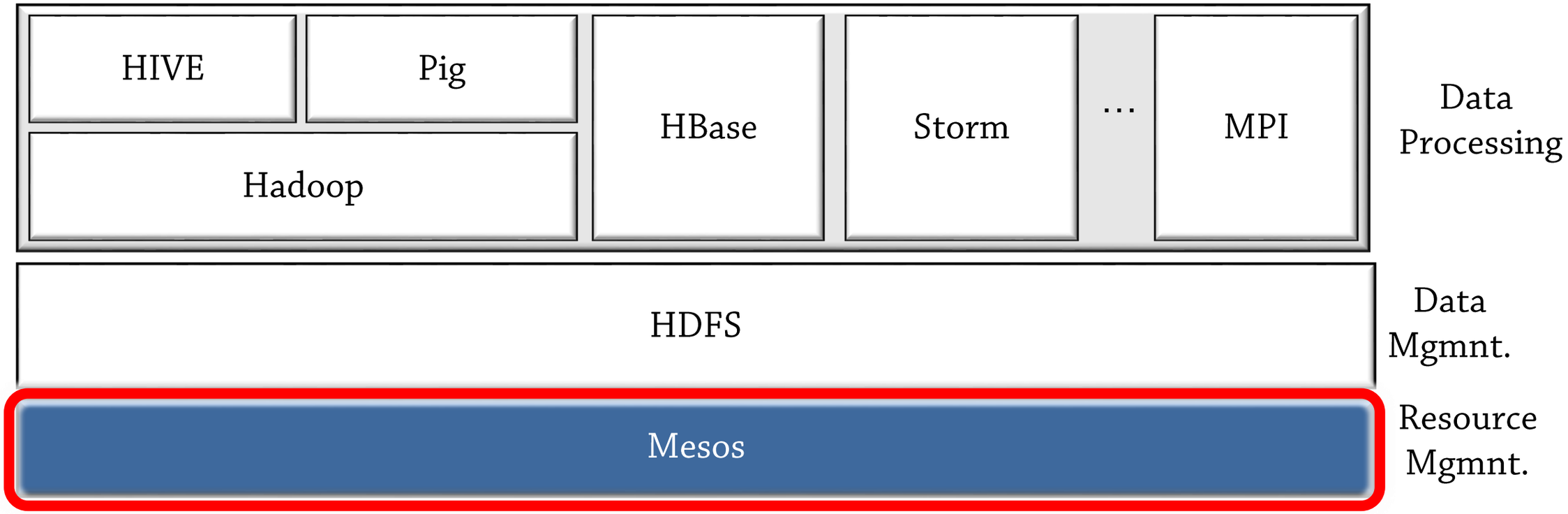
- ***Easy*** to combine ***batch***, ***streaming***, and ***interactive*** computations
  - Single execution model that ***supports*** all computation models
- ***Easy*** to develop ***sophisticated*** algorithms
  - Powerful Python and Scala shells
  - High level abstractions for graph based, and ML algorithms
- ***Compatible*** with existing open source ecosystem (Hadoop/HDFS)
  - Interoperate with existing storage and input formats (e.g., HDFS, Hive, Flume, ..)
  - Support existing execution models (e.g., Hive, GraphLab)

BERKELEY DATA ANALYTICS STACK (BDAS)



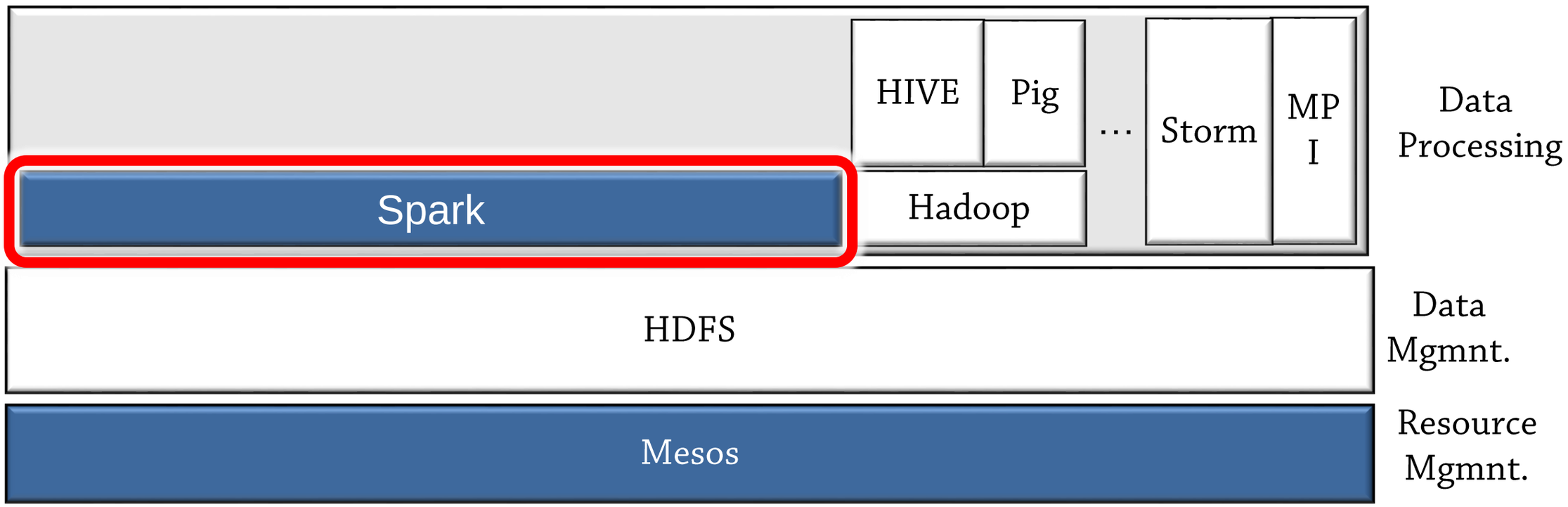
MESOS [RELEASED, V0.9]

- Management platform that allows multiple framework to share cluster
- Compatible with existing open analytics stack
- Deployed in production at Twitter on 3,500+ servers



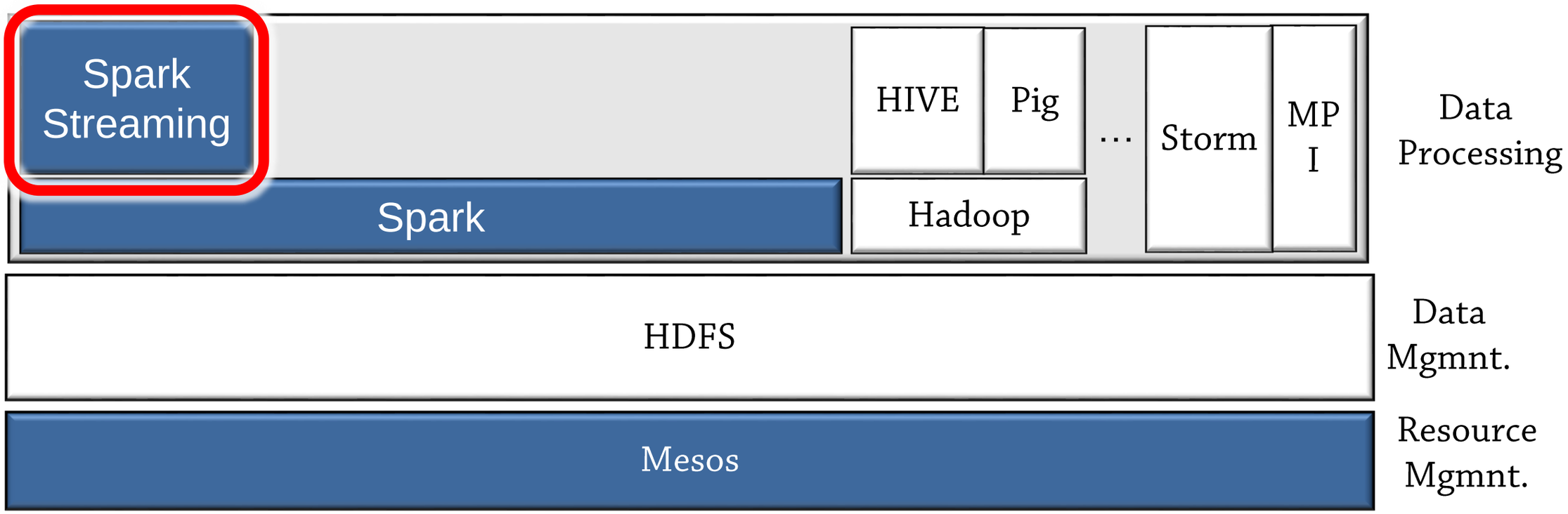
SPARK [RELEASE, V0.7]

- In-memory framework for **interactive** and **iterative** computations
  - Resilient Distributed Dataset **(RDD)**: fault-tolerance, in-memory storage abstraction
- Scala interface, Java and Python APIs



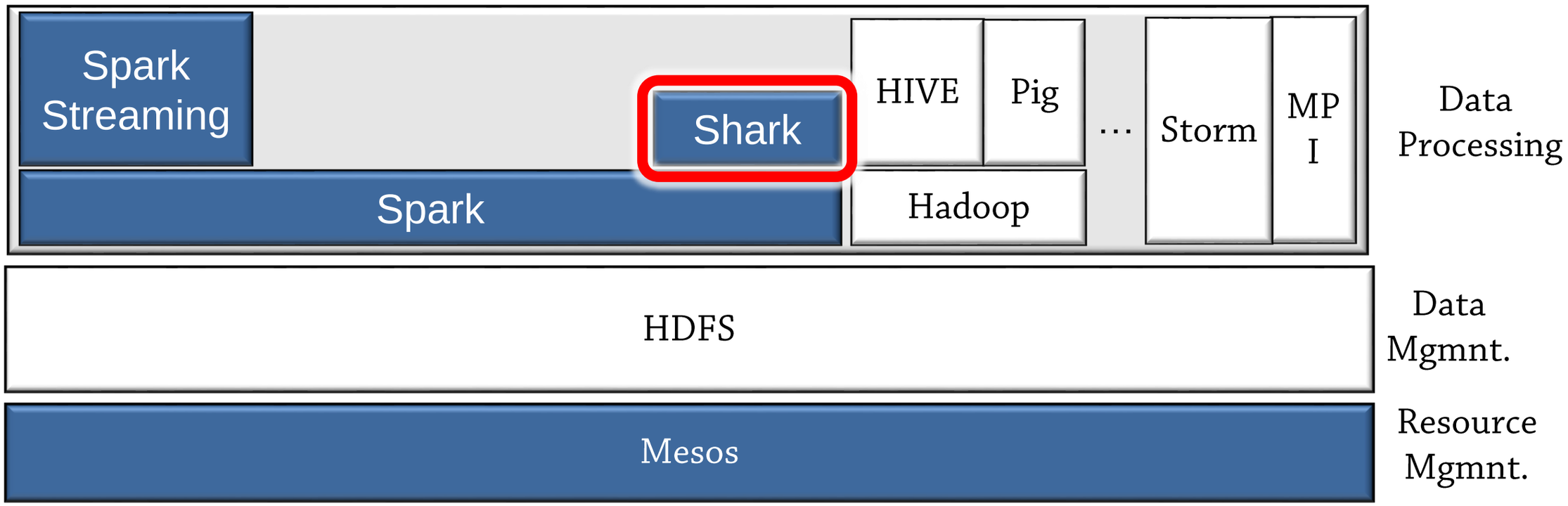
SPARK STREAMING [ALPHA RELEASE]

- Large scale streaming computation
- Ensure exactly one semantics
- Integrated with Spark → unifies ***batch***, ***interactive***, and ***streaming*** computations!



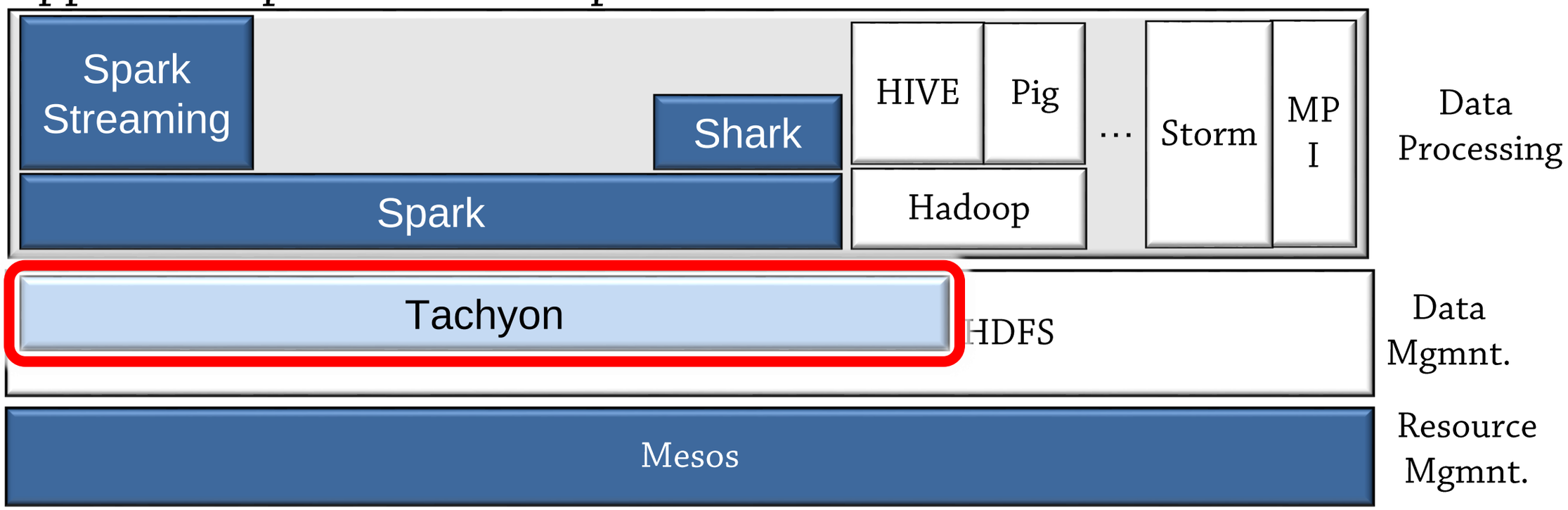
SHARK [RELEASE, V0.2]

- HIVE over Spark: SQL-like interface (supports Hive 0.9)
  - up to 100x faster for in-memory data, and 5-10x for disk
- In tests on hundreds node cluster at YAHOO!



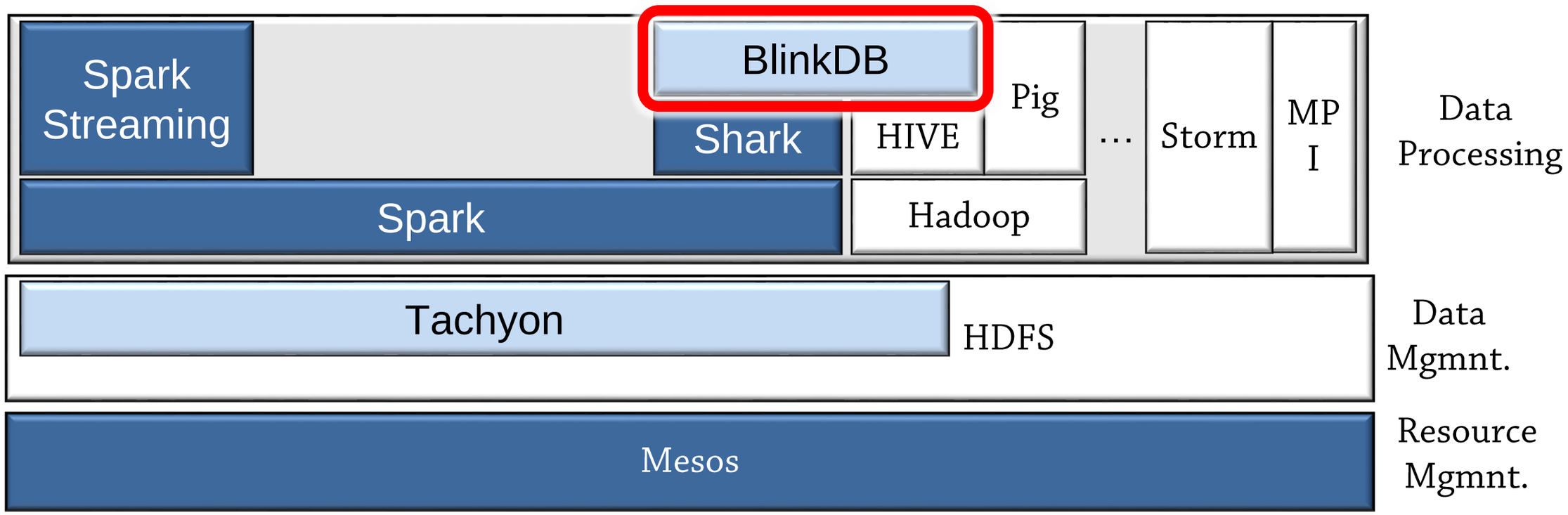
TACHYON

- High-throughput, fault-tolerant in-memory storage
- Interface compatible to HDFS
- Support for Spark and Hadoop



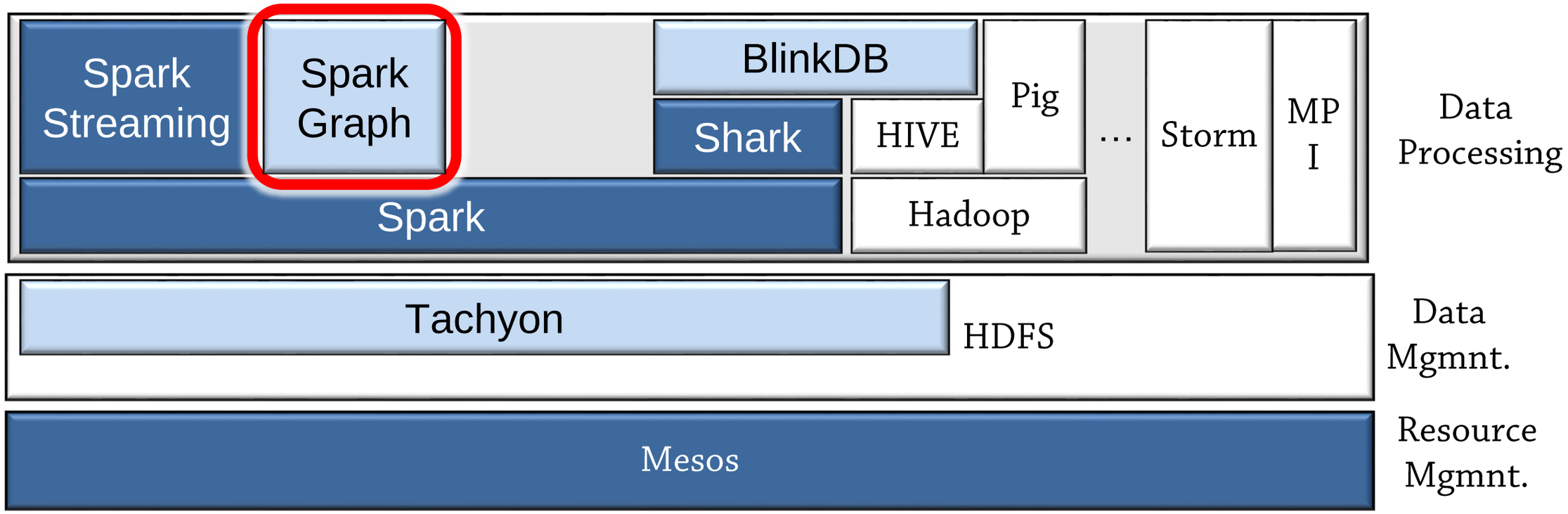
BLINKDB

- Large scale approximate query engine
- Allow users to specify **error** or **time** bounds
- Preliminary prototype starting being tested at Facebook



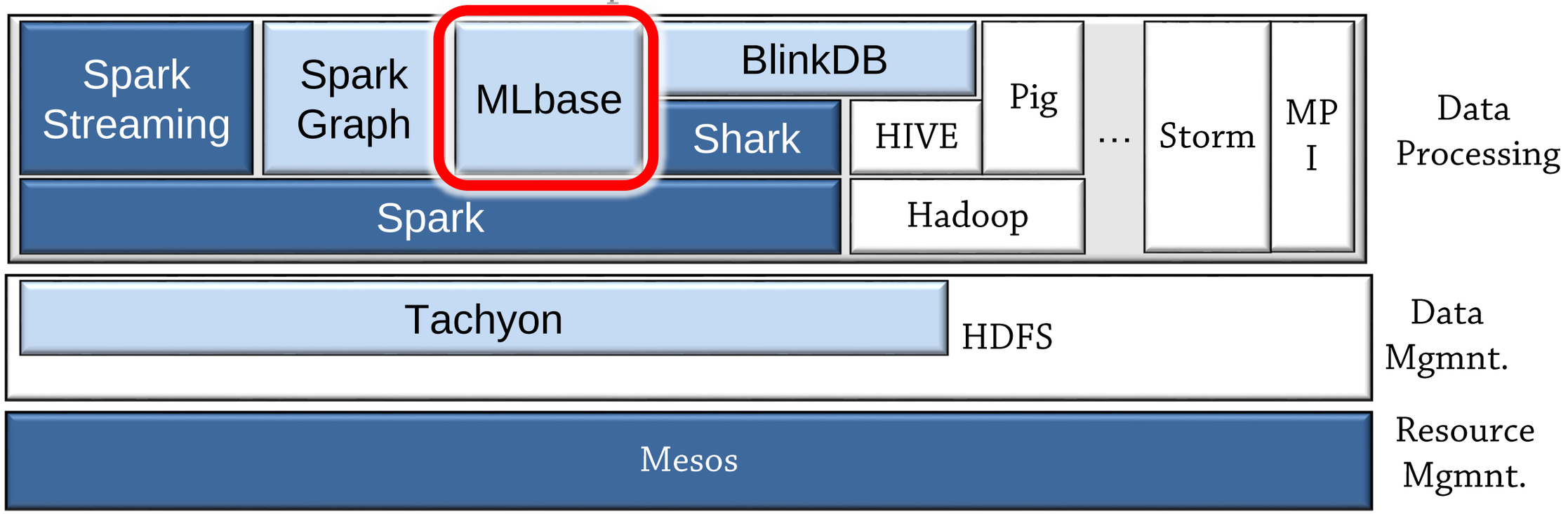
SPARK GRAPH

- **GraphLab API** and **Toolkits** on top of Spark
- Fault tolerance by leveraging Spark



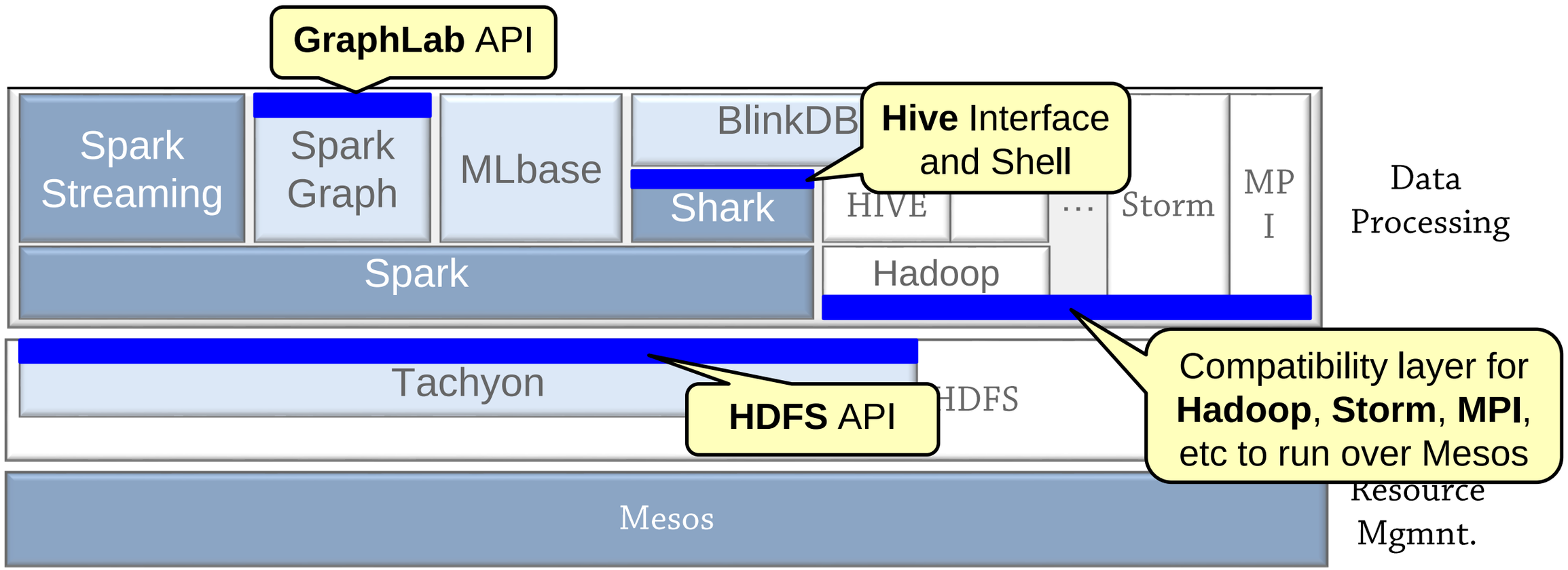
MLBASE

- Declarative approach to ML
- Develop scalable ML algorithms
- Make ML accessible to non-experts



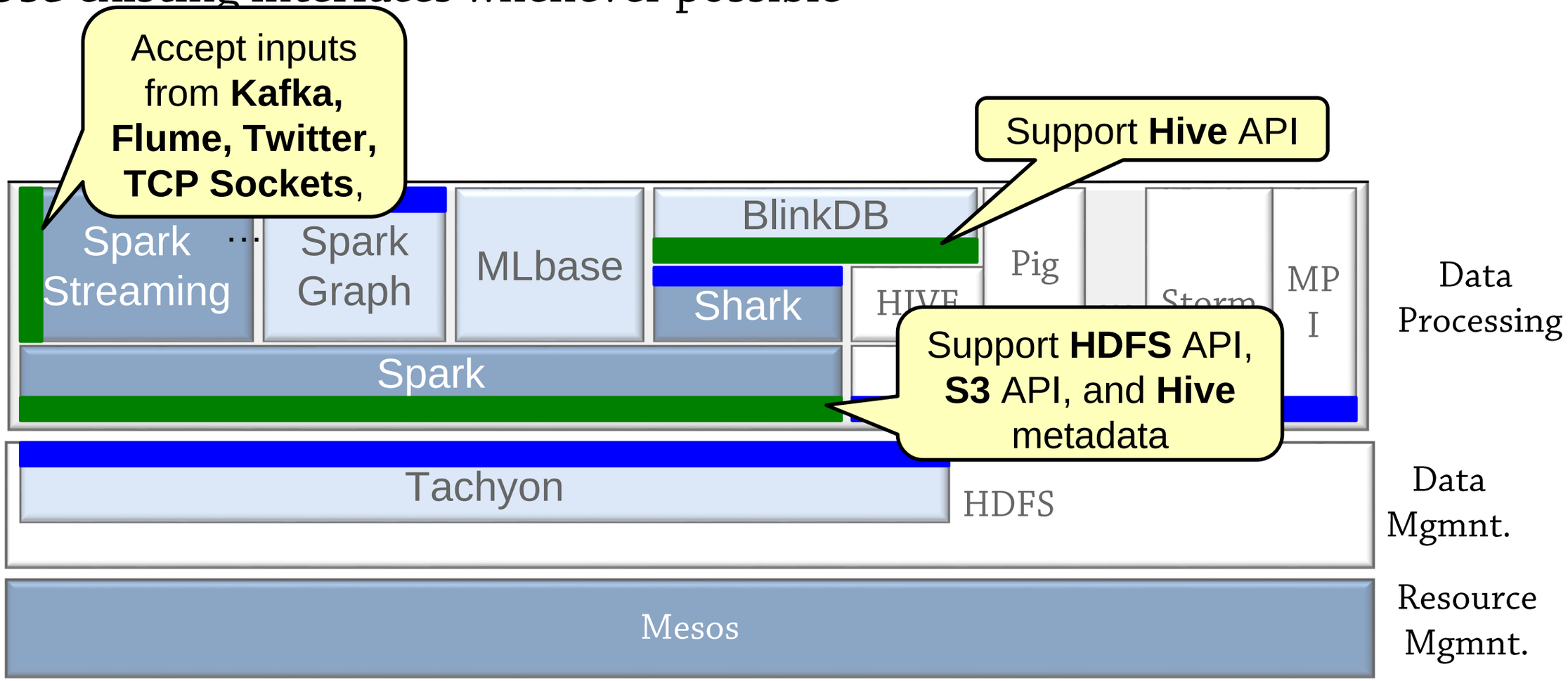
COMPATIBLE WITH OPEN SOURCE ECOSYSTEM

- ***Support*** existing interfaces whenever possible



COMPATIBLE WITH OPEN SOURCE ECOSYSTEM

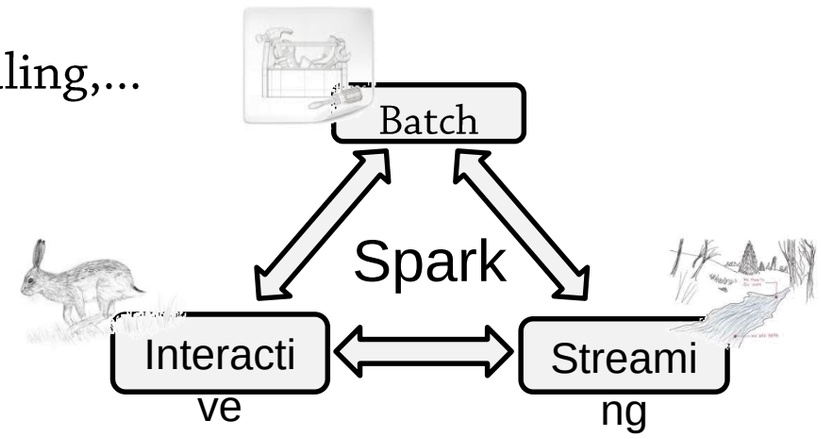
- ***Use*** existing interfaces whenever possible



SUMMARY BDAS

Holistic approach to address next generation of Big Data challenges!

- Support ***interactive*** and ***streaming*** computations
  - In-memory, fault-tolerant storage abstraction, low-latency scheduling,…
- ***Easy*** to combine ***batch***, ***streaming***, and ***interactive*** computations
  - Spark execution engine supports all comp. models
- ***Easy*** to develop ***sophisticated*** algorithms
  - Scala interface, APIs for Java, Python, Hive QL, …
  - New frameworks targeted to graph based and ML algorithms
- ***Compatible*** with existing open source ecosystem
- ***Open source*** (Apache/BSD) and fully committed to release ***high quality*** software
  - Three-person software engineering team lead by Matt Massie (creator of Ganglia, 5th Cloudera engineer)



Spark and Shark

- High-Speed In-Memory Analytics over Hadoop and Hive Data

What is Spark?

- Not a modified version of Hadoop
- Separate, fast, MapReduce-like engine
  - In-memory data storage for very fast iterative queries
  - General execution graphs and powerful optimizations
  - Up to 40x faster than Hadoop
- Compatible with Hadoop's storage APIs
  - Can read/write to any Hadoop-supported system, including HDFS, HBase, SequenceFiles, etc

Project History

- Spark project started in 2009, open sourced 2010
- Shark started summer 2011, alpha April 2012
- In use at Berkeley, Princeton, Klout, Foursquare, Conviva, Quantifind, Yahoo! Research & others
- 200+ member meetup, 500+ watchers on GitHub

Why a New Programming Model?

- MapReduce greatly simplified big data analysis
- But as soon as it got popular, users wanted more:
  - More **complex**, multi-stage applications (e.g. iterative graph algorithms and machine learning)
  - More **interactive** ad-hoc queries
- Both multi-stage and interactive apps require faster **data sharing** across parallel jobs

About Scala

High-level language for JVM

>> Object-oriented + Functional programming (FP)

Statically typed

>> Comparable in speed to Java

>> no need to write types due to type inference

Interoperates with Java

>> Can use any Java class, inherit from it, etc;

>> Can also call Scala code from Java

Quick Tour

Declaring variables:

```
var x: Int = 7
var x = 7  // type inferred

val y = “hi”  // read-only
```

Java equivalent:

```
int x = 7;

final String y = “hi”;
```

Functions:

```
def square(x: Int): Int = x*x

def square(x: Int): Int = {
  x*x
}

def announce(text: String) {
  println(text)
}
```

Last expression in block returned

Java equivalent:

```
int square(int x) {
  return x*x;
}

void announce(String text) {
  System.out.println(text);
}
```

Quick Tour

Generic types:

```
var arr = new Array[Int](8)

var lst = List(1, 2, 3)
 // type of lst is List[Int]
```

Java equivalent:

```
int[] arr = new int[8];

List<Integer> lst =
  new ArrayList<Integer>();
lst.add(...)
```

Indexing:

```
arr(5) = 7

println(lst(5))
```

Java equivalent:

```
arr[5] = 7;

System.out.println(lst.get(5));
```

Quick Tour

Processing collections with functional programming:

```
val list = List(1, 2, 3)

list.foreach(x => println(x))  // prints 1, 2, 3
list.foreach(println)          // same

list.map(x => x + 2)  // => List(3, 4, 5)
list.map(_ + 2)       // same, with placeholder notation

list.filter(x => x % 2 == 1)  // => List(1, 3)
list.filter(_ % 2 == 1)       // => List(1, 3)

list.reduce((x, y) => x + y)  // => 6
list.reduce(_ + _)            // => 6
```

Function expression (closure)

All of these leave the list unchanged (List is Immutable)

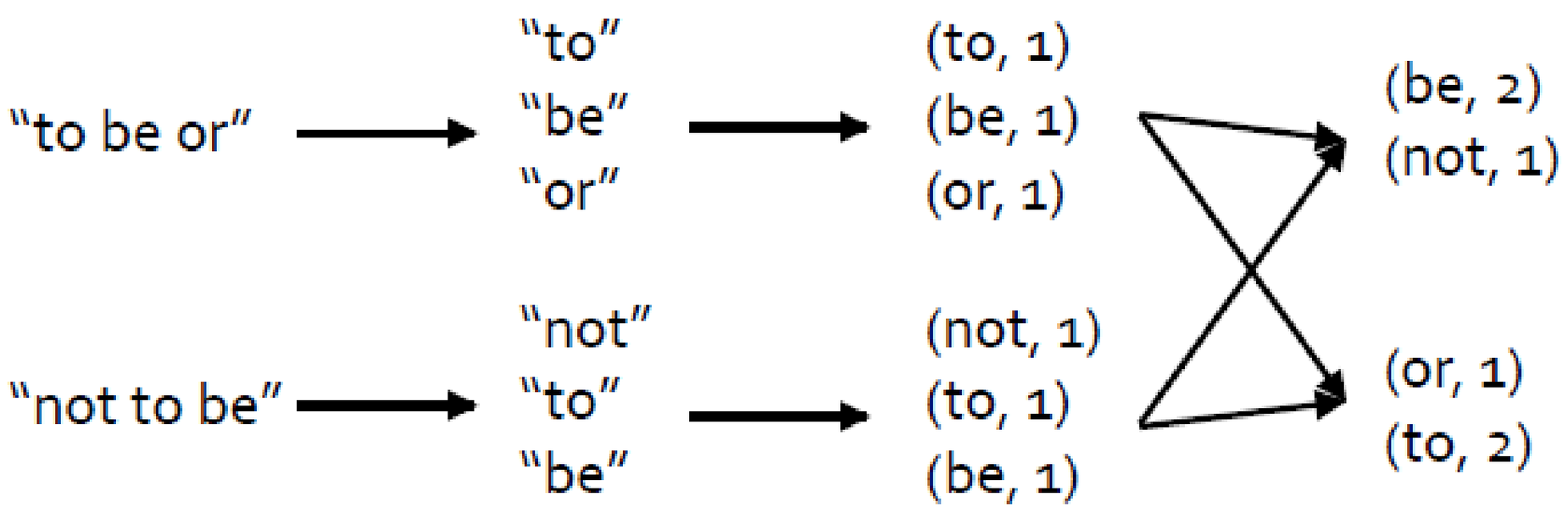
Scala Closure Syntax

```
(x: Int) => x + 2   // full version

x => x + 2    // type inferred

_ + 2         // when each argument is used exactly once

x => {        // when body is a block of code
  val numberToAdd = 2
  x + numberToAdd
}

// If closure is too long, can always pass a function
def addTwo(x: Int): Int = x + 2

list.map(addTwo)
```

Scala allows defining a "local function" inside another function

Example: Word Count

```
val lines = sc.textFile(“hamlet.txt”)

val counts = lines.flatMap(line => line.split(“ ”))
                  .map(word => (word, 1))
                  .reduceByKey(_ + _)
```

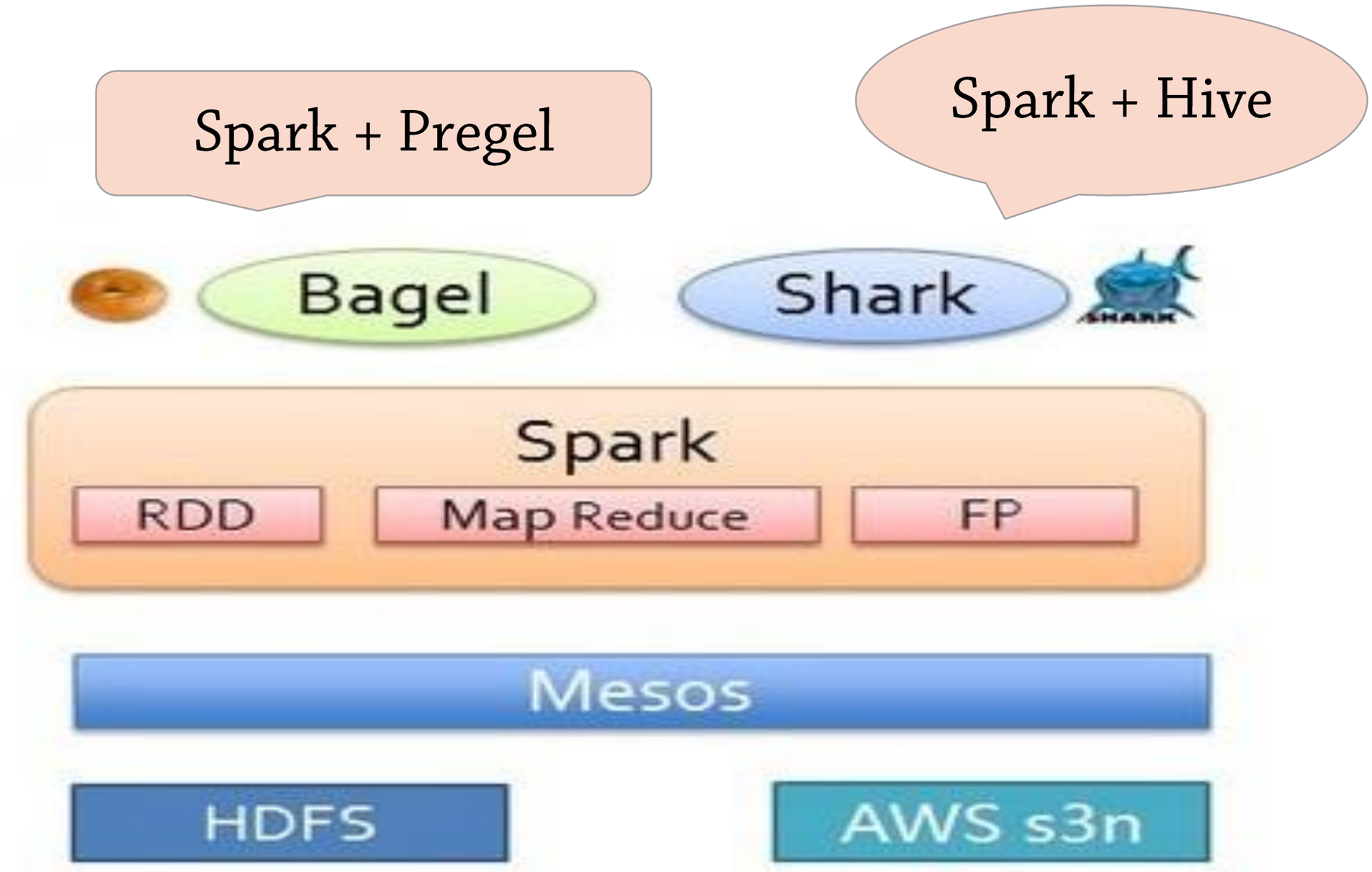


Other Collection Methods

Scala collections provide many other functional methods; for example, Google for "Scala Seq"

Method on Seq[T]	Explanation
`map(f: T => U): Seq[U]`	Pass each element through f
`flatMap(f: T => Seq[U]): Seq[U]`	One-to-many map
`filter(f: T => Boolean): Seq[T]`	Keep elements passing f
`exists(f: T => Boolean): Boolean`	True if one element passes
`forall(f: T => Boolean): Boolean`	True if all elements pass
`reduce(f: (T, T) => T): T`	Merge elements using f
`groupBy(f: T => K): Map[K,List[T]]`	Group elements by f(element)
`sortBy(f: T => K): Seq[T]`	Sort elements by f(element)
. . .	

Spark framework



Run Spark

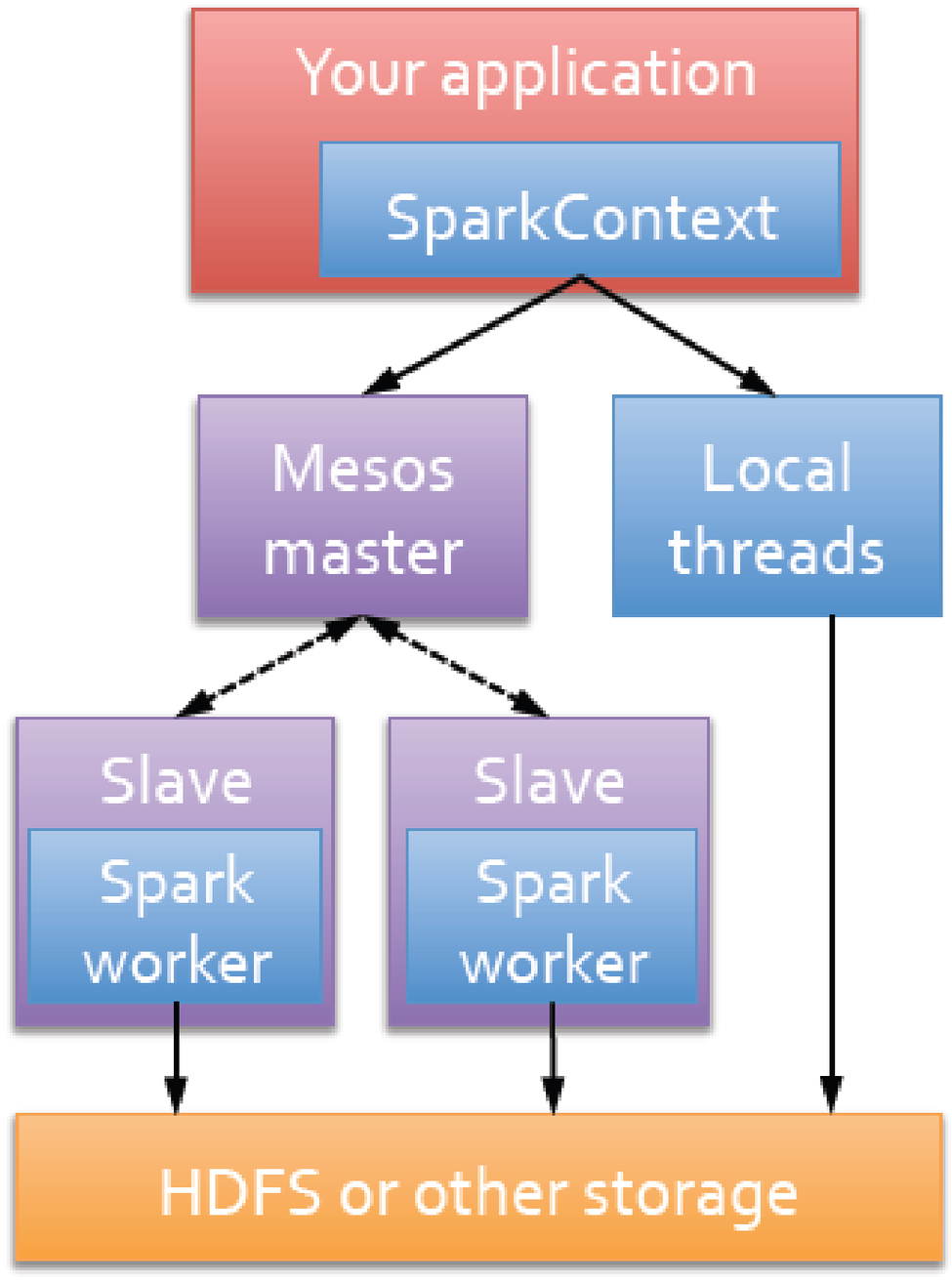
Spark runs as a library in your program
(1 instance per app)

Runs tasks locally or on Mesos

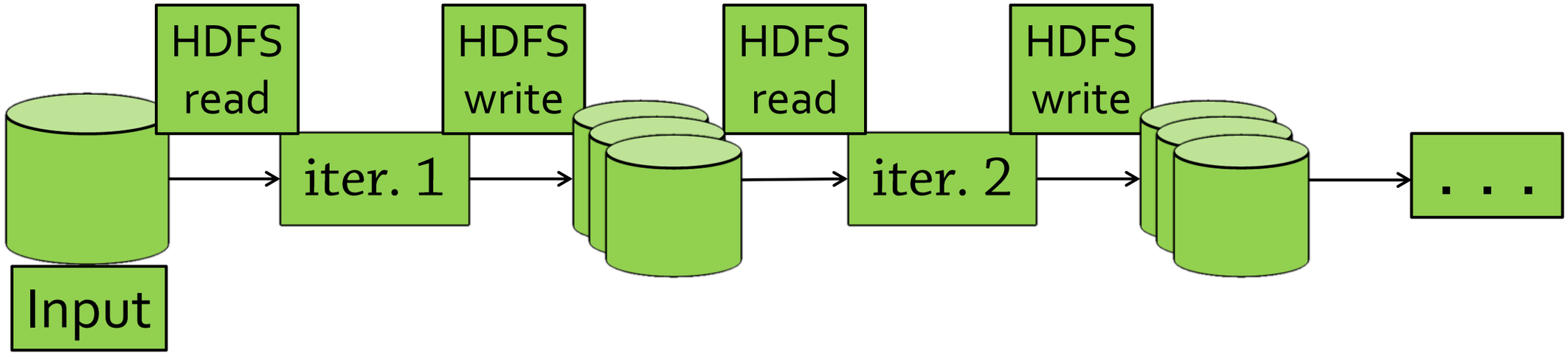
>> **new SparkContext** (masterUrl, jobname, [sparkhome], [jars])

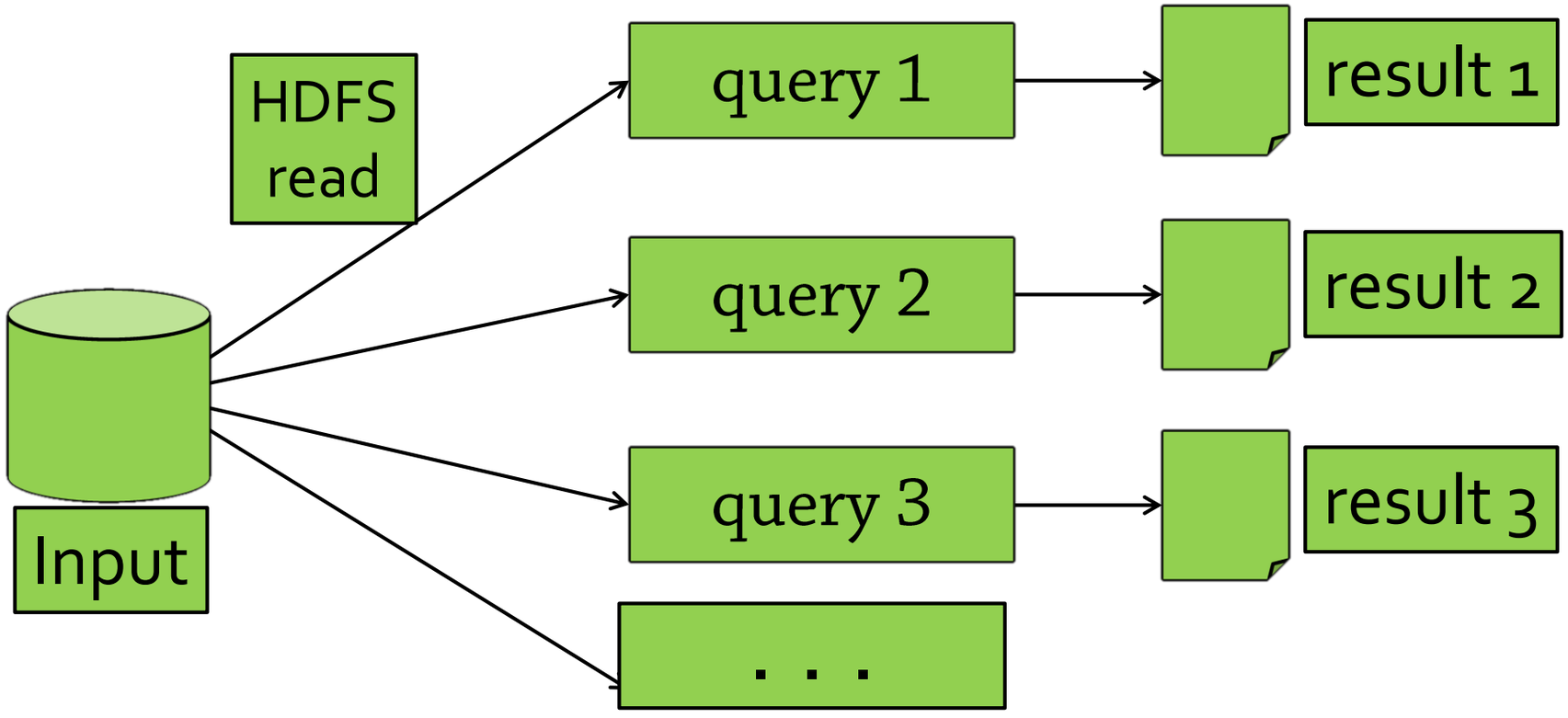
>> MASTER=local[n] ./spark-shell

>> MASTER=HOST:PORT ./spark-shell



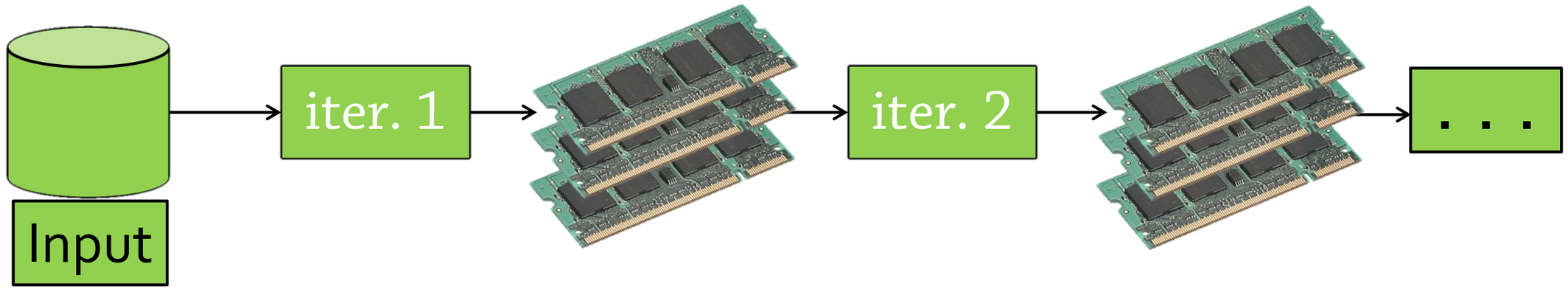
DATA SHARING IN MAPREDUCE

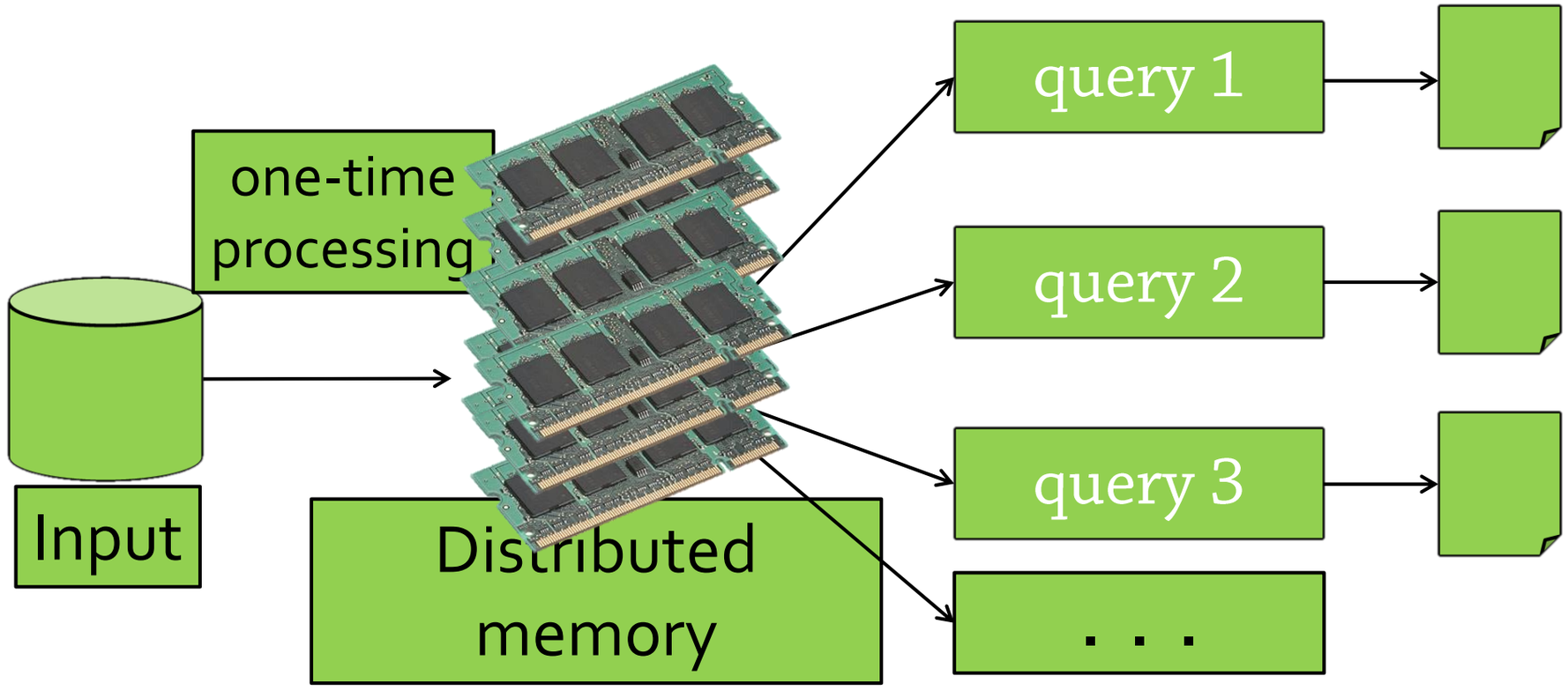




Slow due to replication, serialization, and disk IO

DATA SHARING IN SPARK





10-100× faster than network and disk

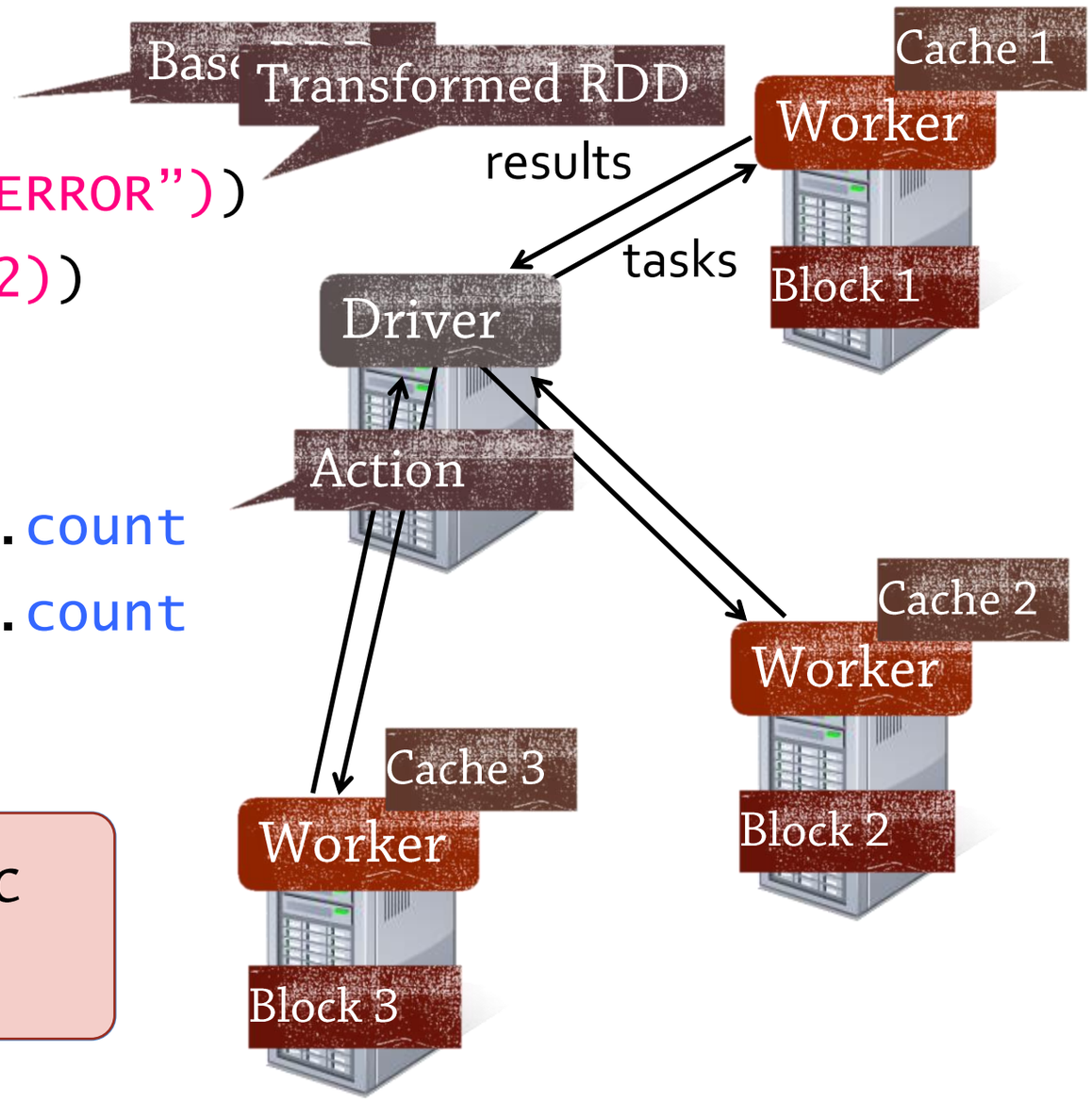
Spark Programming Model

- Key idea: *resilient distributed datasets (RDDs)*
  - Distributed collections of objects that can be cached in memory across cluster nodes
  - Manipulated through various parallel operators
  - Automatically rebuilt on failure
- Interface
  - Clean language-integrated API in Scala
  - Can be used *interactively* from Scala console

Example: Log Mining

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

```
lines = spark.textFile(“hdfs://...”)
errors = lines.filter(_.startsWith(“ERROR”))
messages = errors.map(_.split(‘\t’)(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains(“foo”)).count
cachedMsgs.filter(_.contains(“bar”)).count
. . .
```



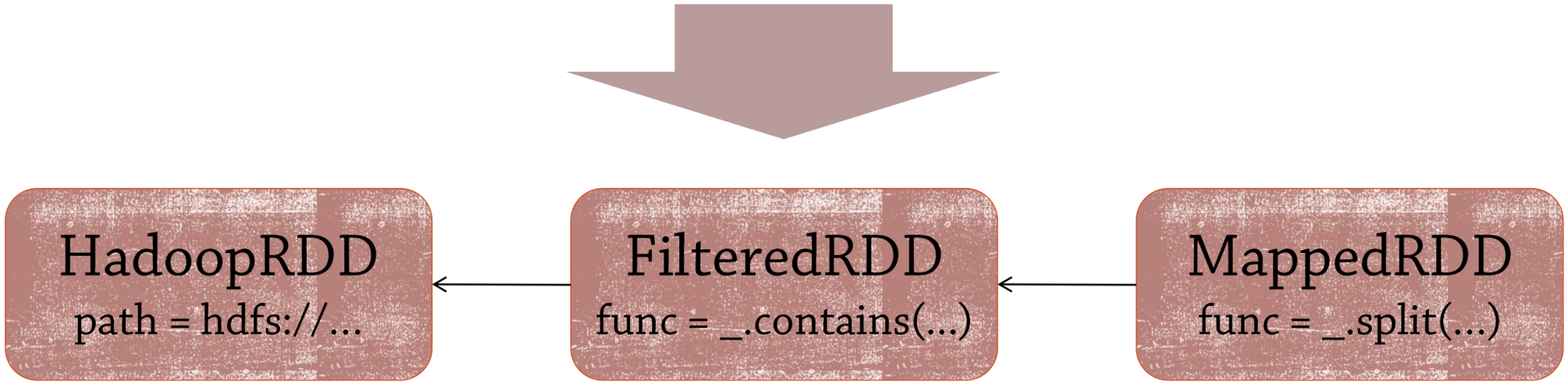
Result: scaled to 1 TB data in 5-7 sec (vs 170 sec for on-disk data)

FAULT TOLERANCE

RDDs track the series of transformations used to build them (their *lineage*) to recompute lost data

E.g:

```
messages = textFile(...).filter(_.contains("error"))
           .map(_.split('\t')(2))
```



Example: Logistic Regression

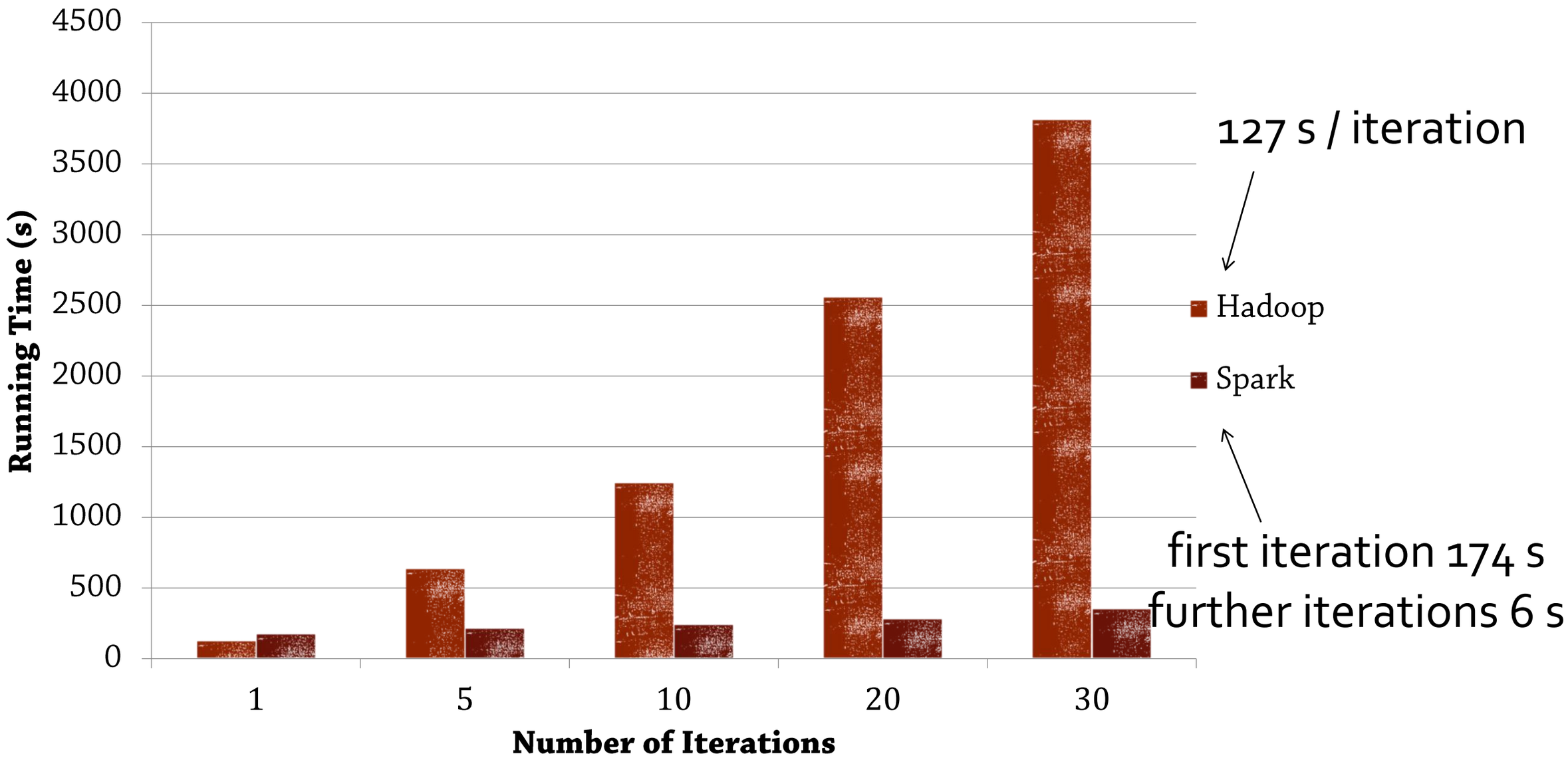
```
val data = spark.textFile(...).map(readPoint).cache()

var w = Vector.random(D)

for (i <- 1 to ITERATIONS) {
  val gradient = data.map(p =>
    (1 / (1 + exp(-p.y*(w dot p.x)))) * p.x
  ).reduce(_ + _)
  w -= gradient
}

println("Final w: " + w)
```

Load data in memory once

Initial parameter vector

Repeated MapReduce steps to do gradient descent

LOGISTIC REGRESSION PERFORMANCE



SUPPORTED OPERATORS

- map
- filter
- groupBy
- sort
- join
- leftOuterJoin
- rightOuterJoin
- reduce
- count
- reduceByKey
- groupByKey
- first
- union
- cross

sample

cogroup

take

partitionBy

pipe

save

...

SPARK USERS


CONVIVA


foursquare


quantiFind


KLOUT


YAHOO!
RESEARCH


University of California
Berkeley


PRINCETON
UNIVERSITY


UCSF

SHARK: HIVE ON SPARK

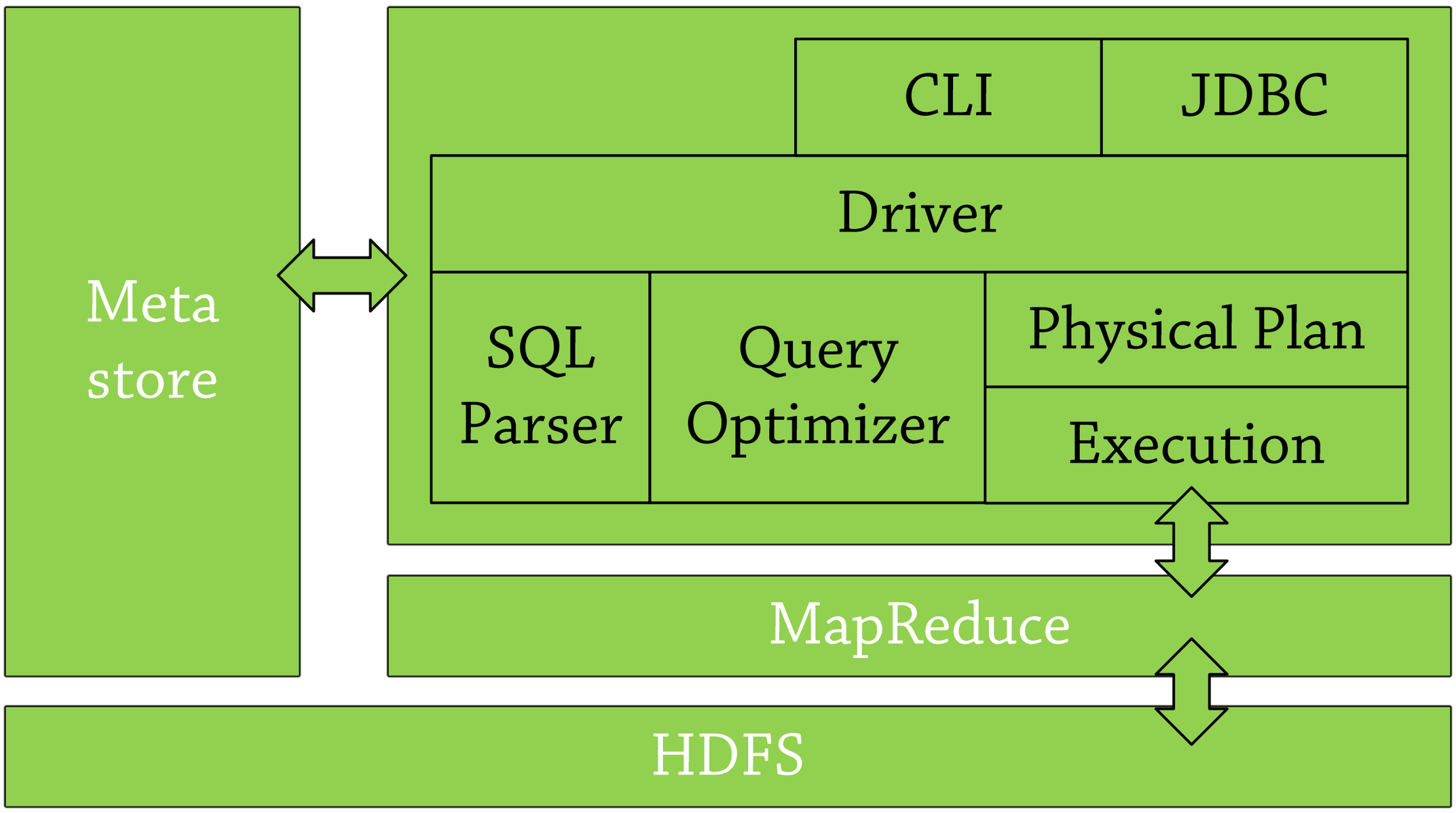
What is Shark?

- Port of Apache Hive to run on Spark
- Compatible with existing Hive data, metastores, and queries (HiveQL, UDFs, etc)
- Similar speedups of up to 40x

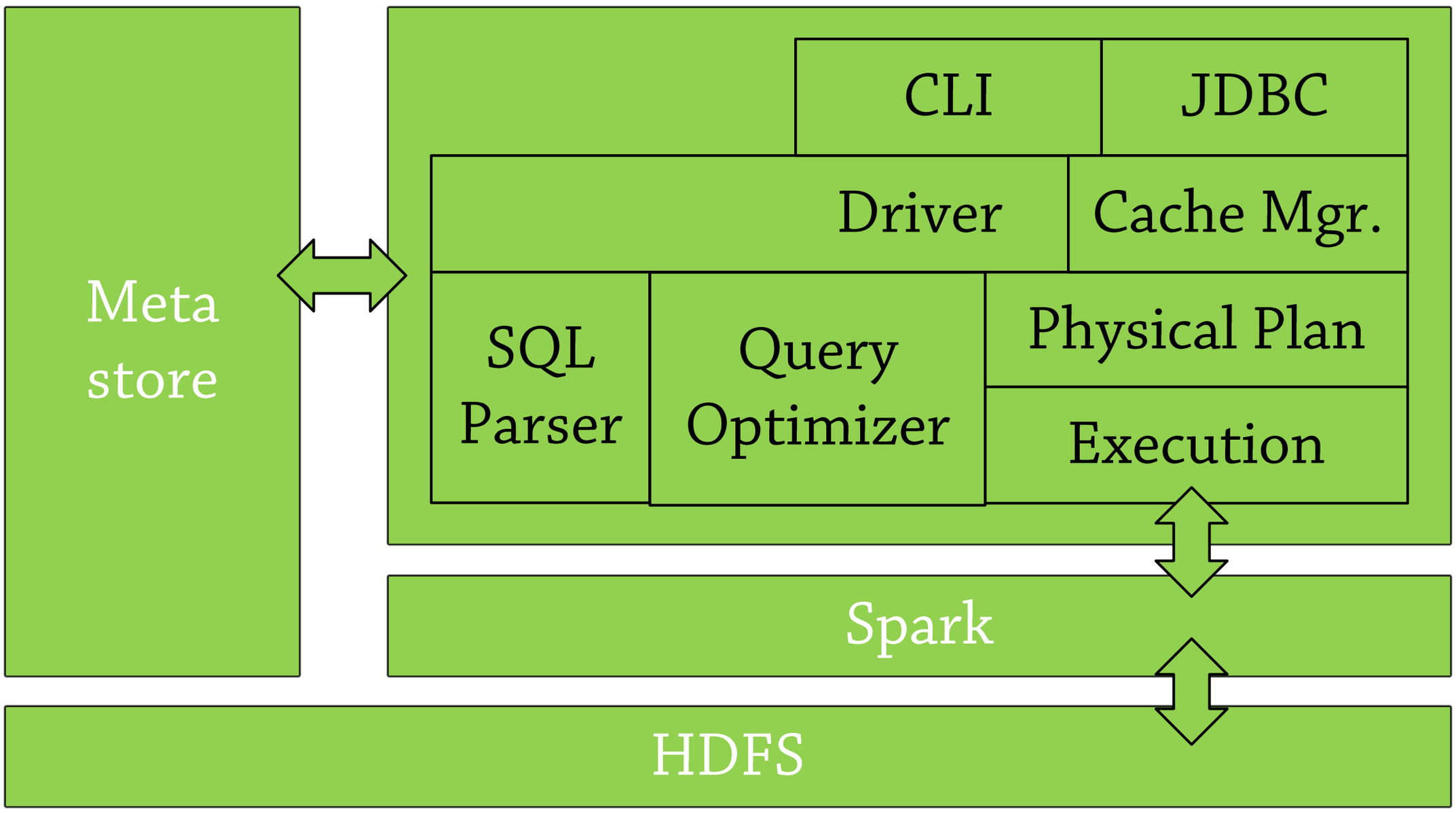
Motivation

- Hive is great, but Hadoop's execution engine makes even the smallest queries take minutes
- Scala is good for programmers, but many data users only know SQL
- **Can we extend Hive to run on Spark?**

Hive Architecture

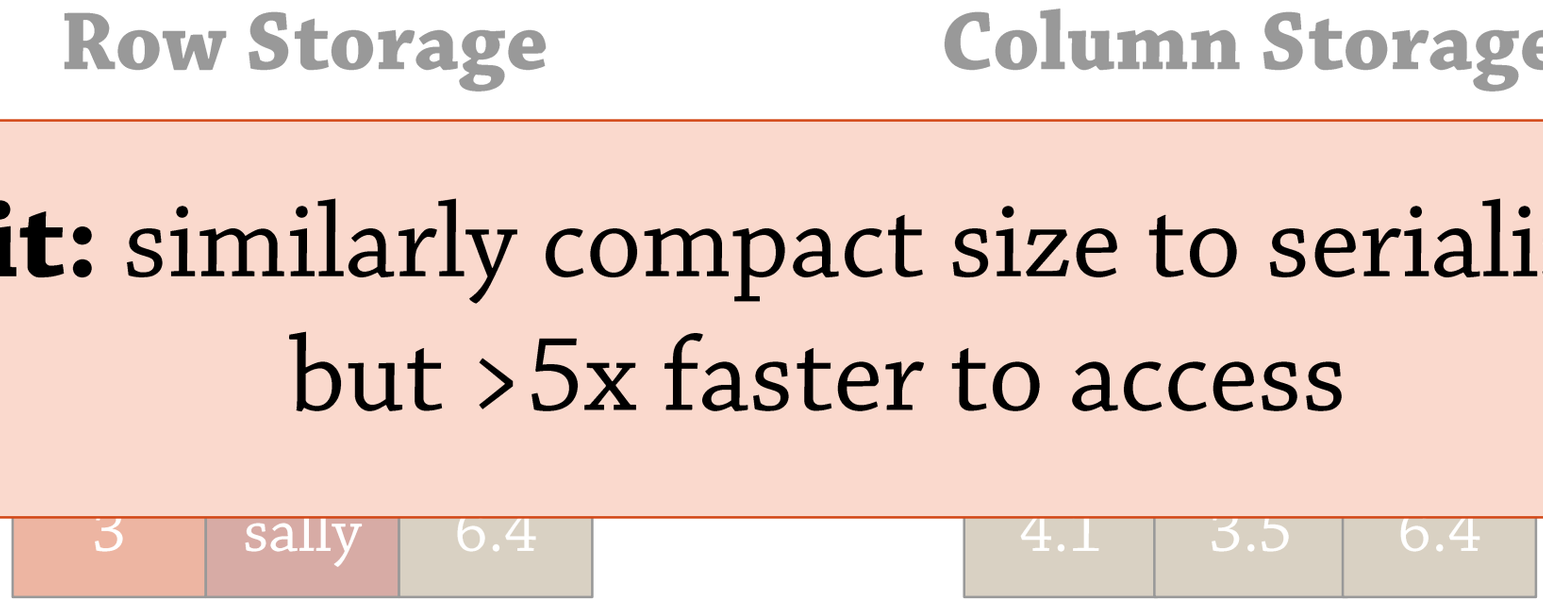


Shark Architecture



EFFICIENT IN-MEMORY STORAGE

- Simply caching Hive records as Java objects is inefficient due to high per-object overhead
- Instead, Shark employs column-oriented storage using **arrays of primitive types**



Benefit: similarly compact size to serialized data, but >5x faster to access

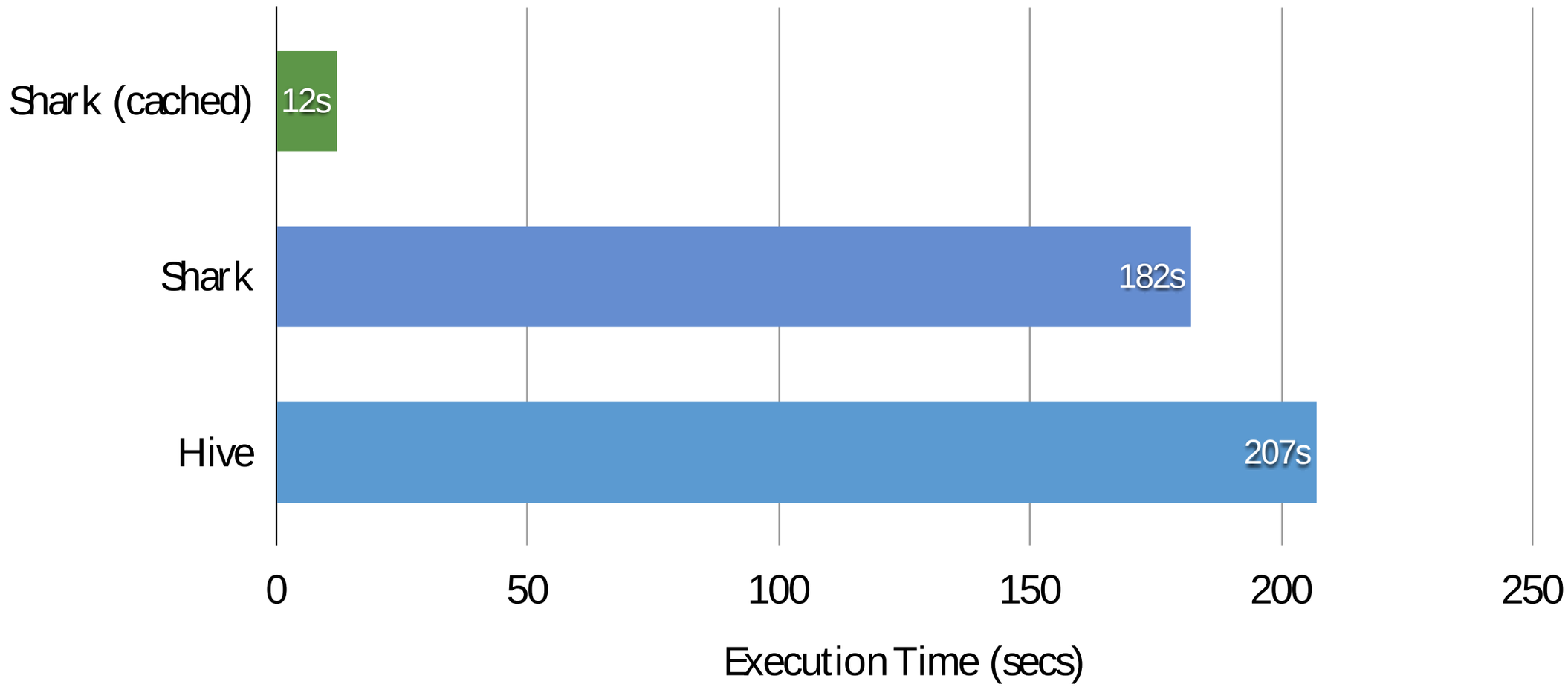
USING SHARK

- `CREATE TABLE mydata_cached AS SELECT …`
- Run standard HiveQL on it, including UDFs
  - A few esoteric features are not yet supported
- Can also call from Scala to mix with Spark

Early alpha release at shark.cs.berkeley.edu

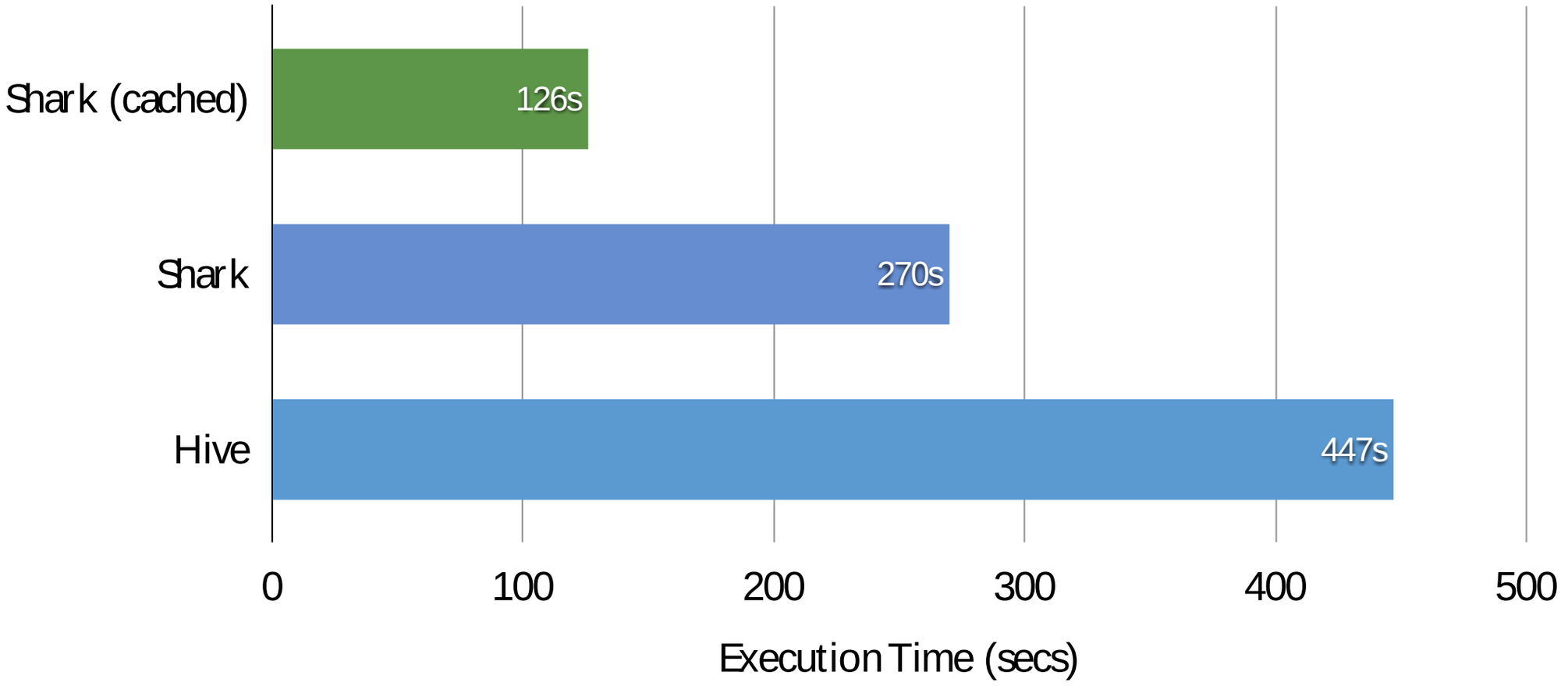
BENCHMARK QUERY 1

```
SELECT * FROM grep WHERE field LIKE '%XYZ%';
```



BENCHMARK QUERY 2

```
SELECT sourceIP, AVG(pageRank), SUM(adRevenue) AS earnings
FROM rankings AS R, userVisits AS V ON R.pageURL = V.destURL
WHERE V.visitDate BETWEEN ‘1999-01-01’ AND ‘2000-01-01’
GROUP BY V.sourceIP
ORDER BY earnings DESC
LIMIT 1;
```



REFERENCES

- **BDAS**: https://amplab.cs.berkeley.edu/software/
- **Spark:** http://spark.apache.org/