whoami

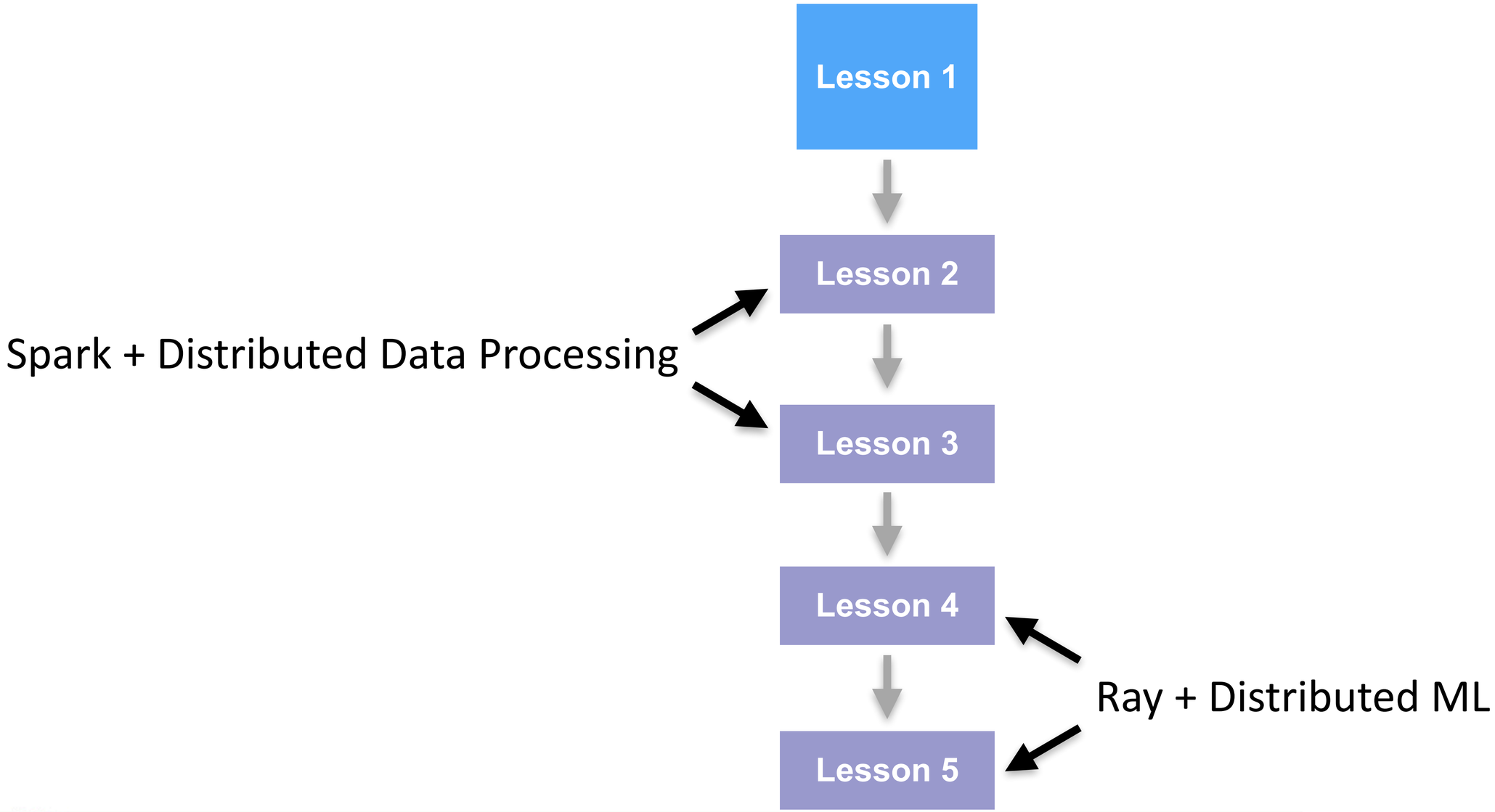
Researcher

- Human-in-the-loop machine learning + interpretability

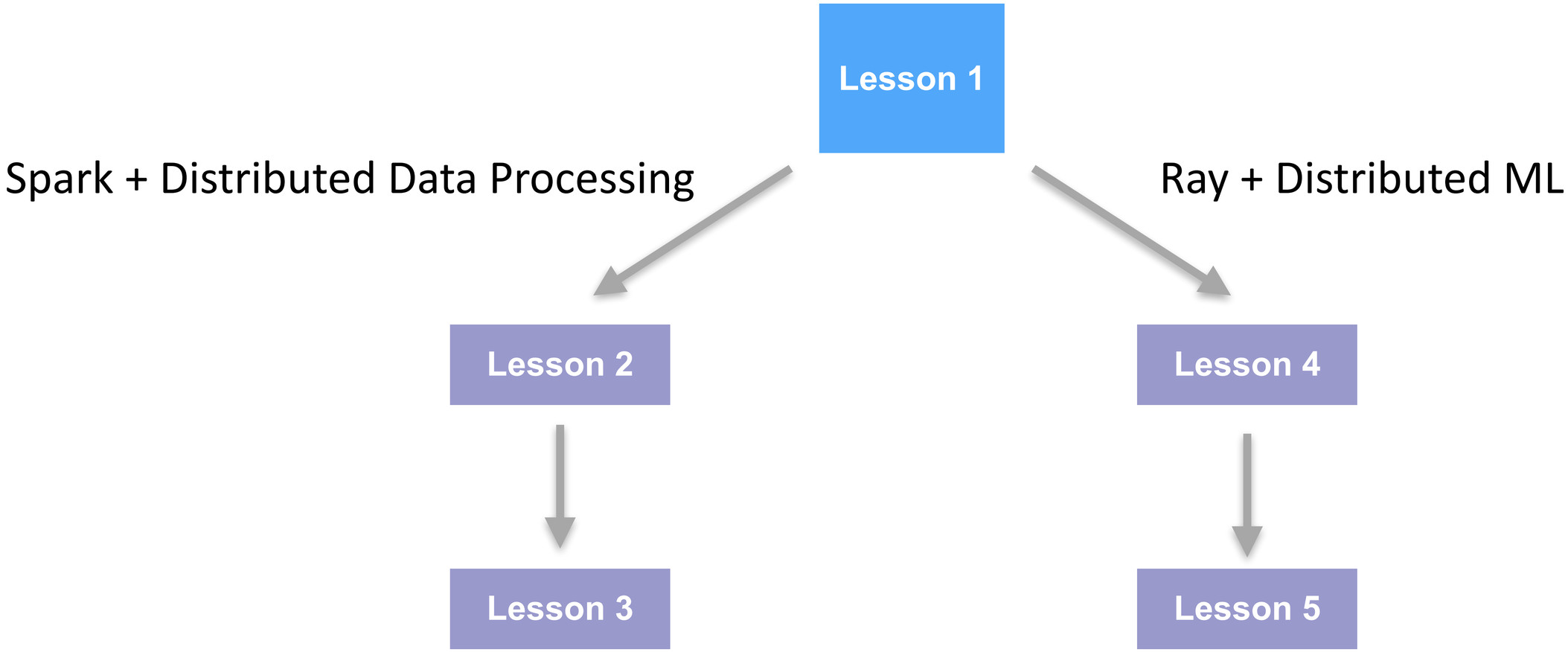
YouTuber — https://jonathans.estate/youtube

- Data science and deep learning

Overview


Lesson 1
Lesson 2
Lesson 3
Lesson 4
Lesson 5
Spark + Distributed Data Processing
Ray + Distributed ML

Overview


Lesson 1
Spark + Distributed Data Processing
Ray + Distributed ML
Lesson 2
Lesson 4
Lesson 3
Lesson 5

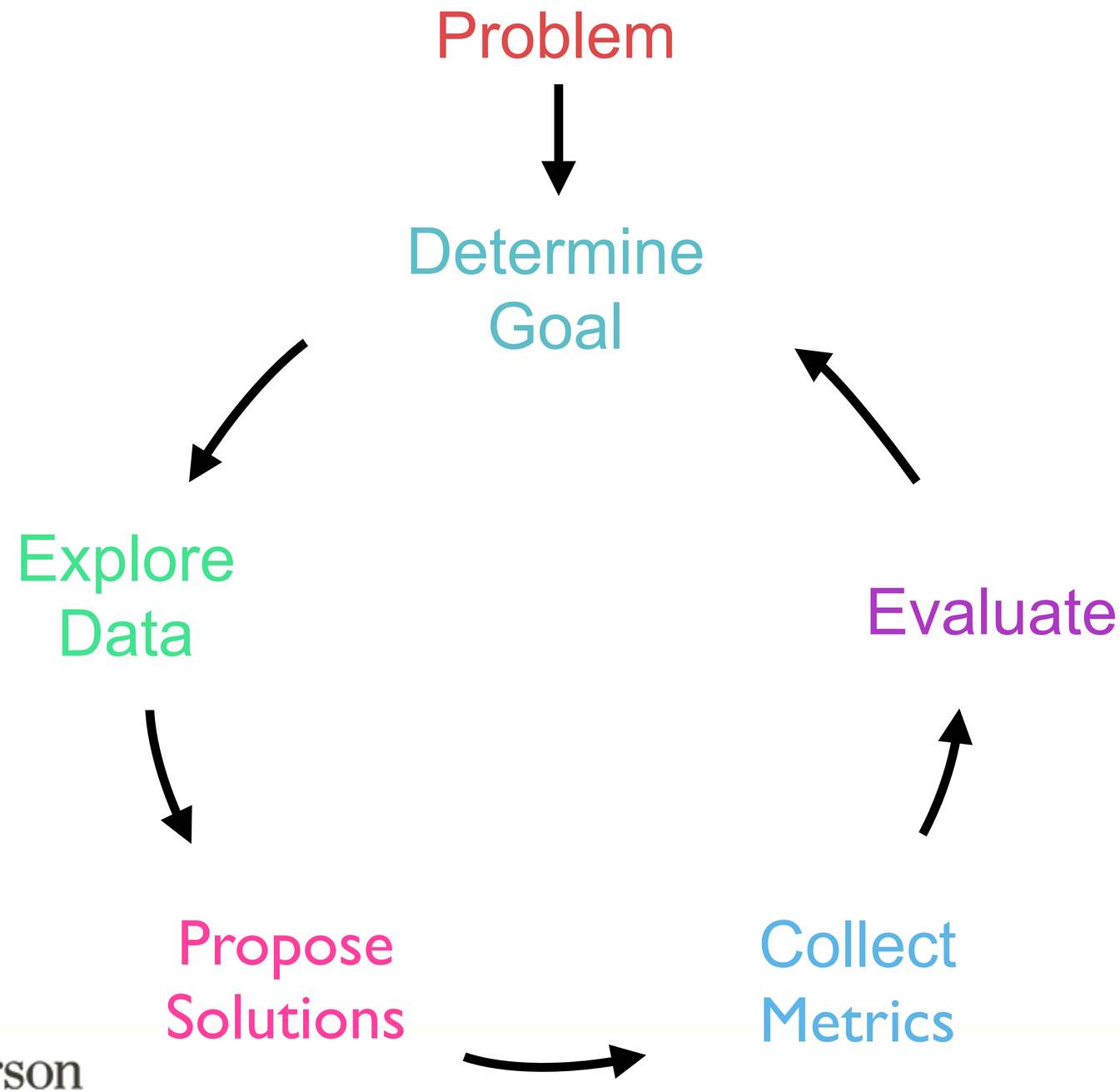
Getting the Materials

https://jonathans.estate/scaling-data-science

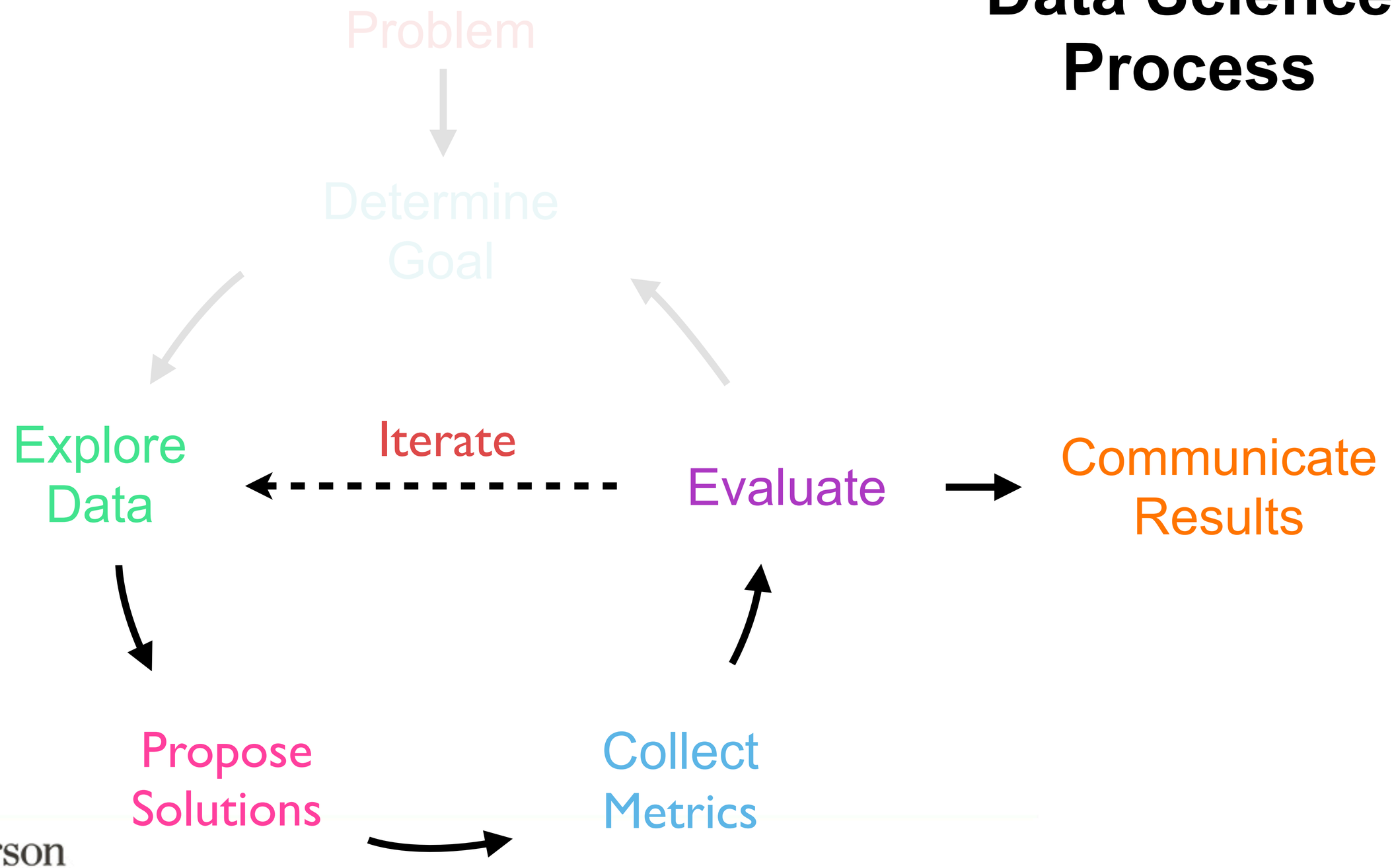
Getting Help

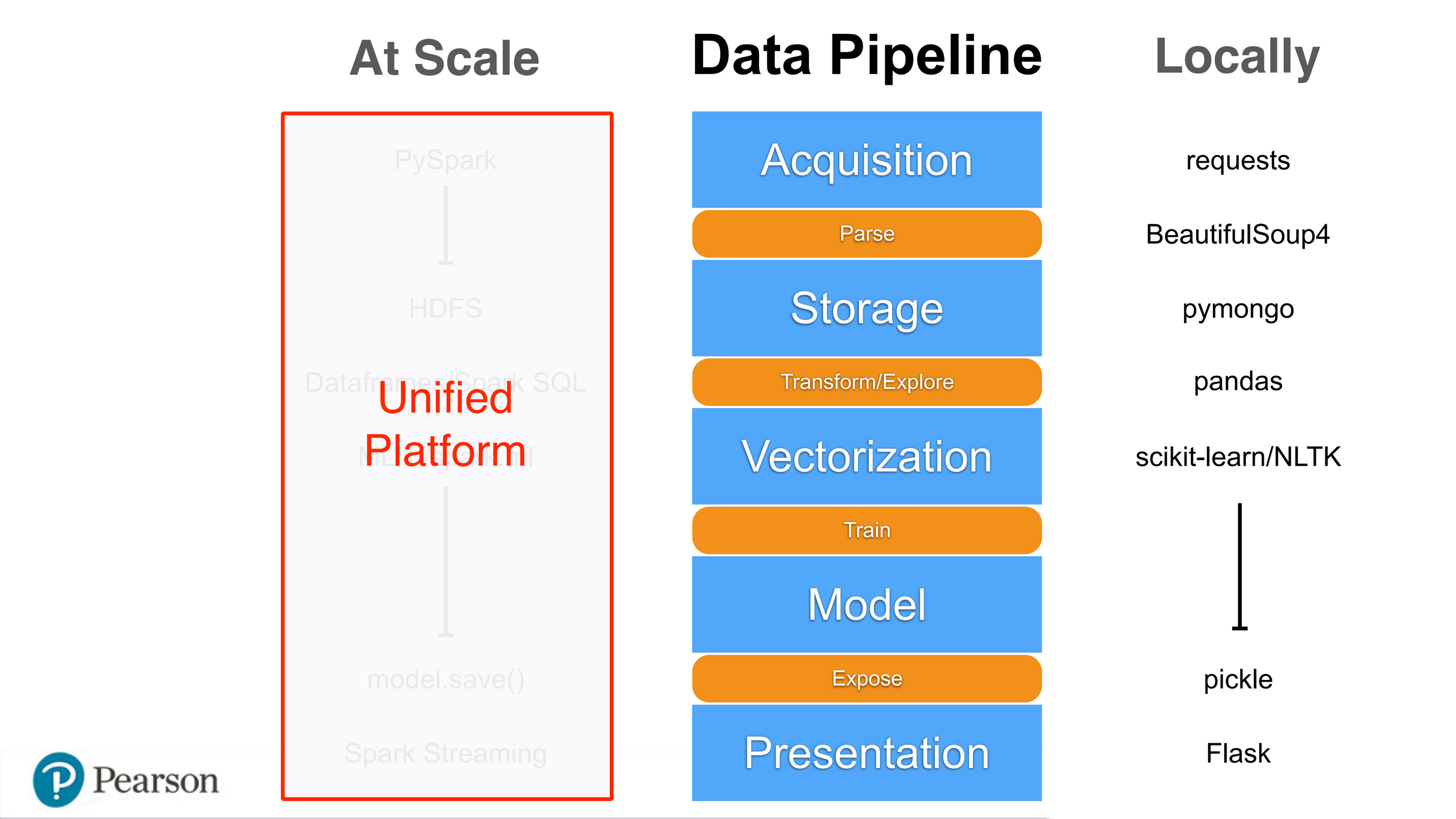
https://jonathans.estate/scaling-data-science

Data Science Process



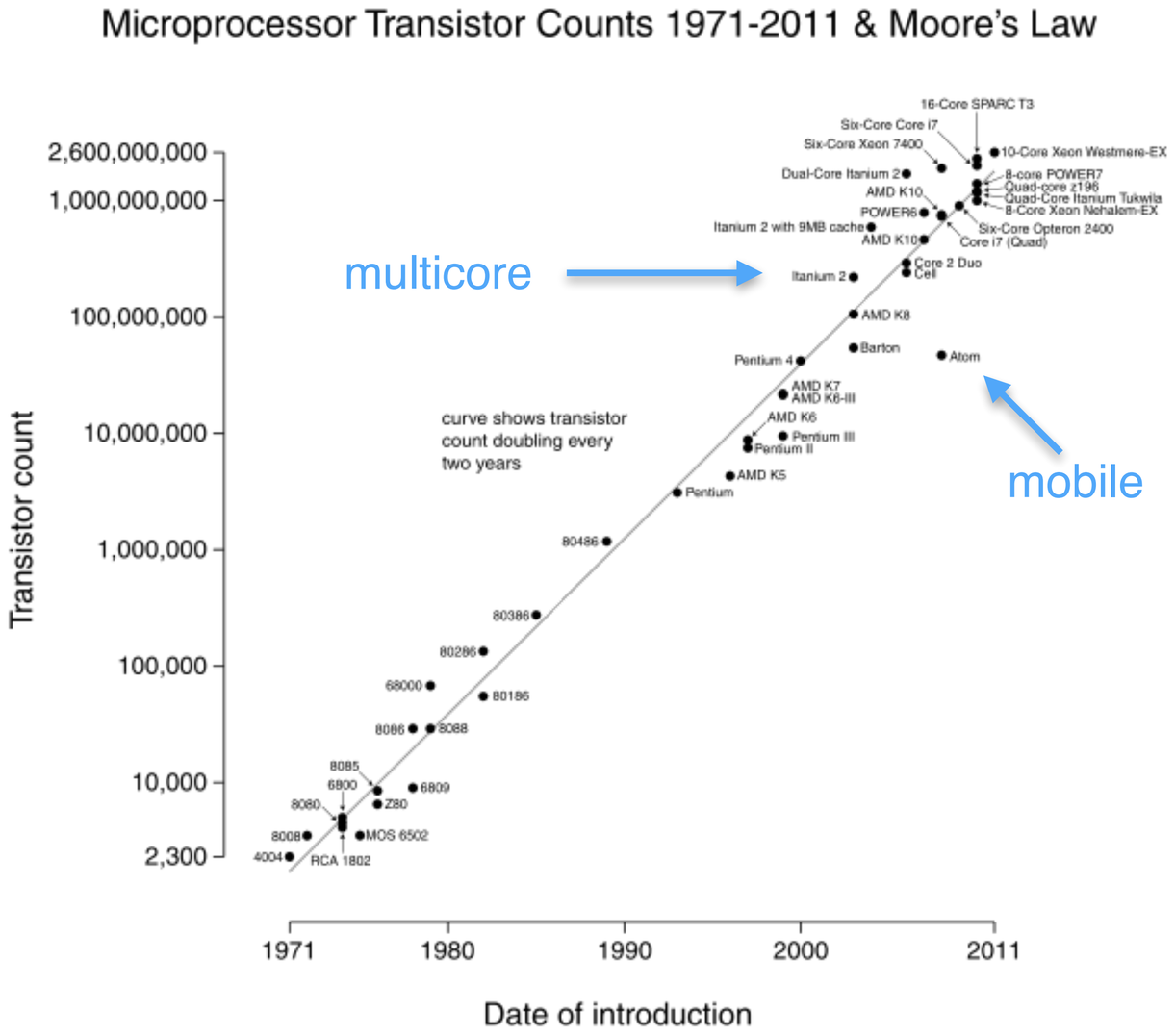
Data Science Process




At Scale
Data Pipeline
Locally
PySpark
HDFS
Dataframes/Spark SQL
MLlib/Spark ML
model.save()
Spark Streaming
Unified Platform
Acquisition
Parse
Storage
Transform/Explore
Vectorization
Train
Model
Expose
Presentation
requests
BeautifulSoup4
pymongo
pandas
scikit-learn/NLTK
pickle
Flask
Pearson

Interlude: How to Scale

A Brief History

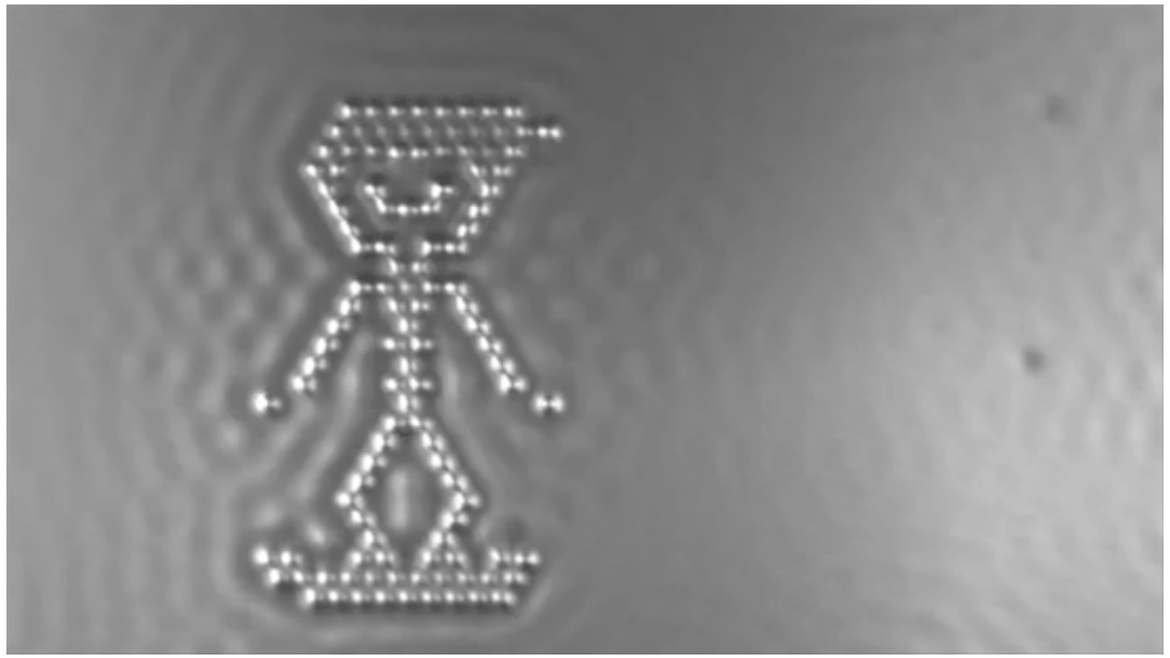


A Brief History

> *In terms of size [of transistors] you can see that we're approaching the size of atoms which is a fundamental barrier, but it'll be two or three generations before we get that far—but that's as far out as we've ever been able to see.*

- Gordon Moore (2006)

A Brief History



https://www.youtube.com/watch?v=oSCX78-8-q0

A Brief History

Power Wall

- As transistor density increases, so too does power consumption (and dissipation)
- This limits the speed (and density) of transistors on a chip

Atomic Limits

- Thinner transistors can switch faster
- But you cannot make a transistor smaller than a single atom
- This limits the speed of an individual transistor

A Brief History

> *We have another 10 to 20 years before we reach a fundamental limit. By then they'll be able to make bigger chips and have transistor budgets in the billions.*

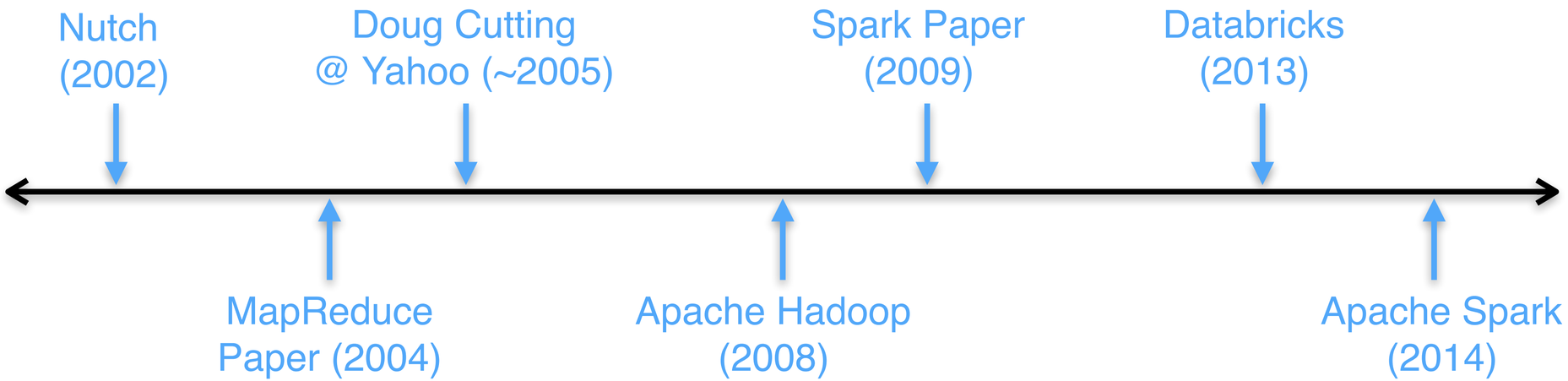
- Gordon Moore (2006)

The Recent Past

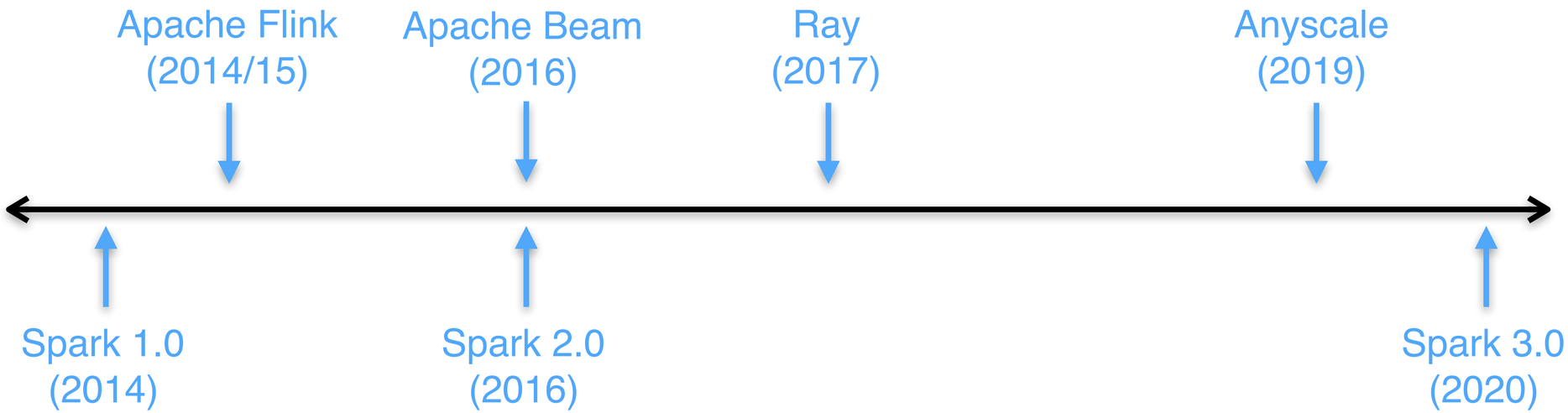
- Found by AMPLab, UC Berkeley (circa 2009)
  - Created by Matei Zaharia (PhD Thesis)
  - Maintained by Apache Software Foundation
  - Commercial Support by Databricks

The Recent Past

- Found by AMPLab, UC Berkeley (circa 2009)
  - Created by Matei Zaharia (PhD Thesis)
  - Maintained by Apache Software Foundation
  - Commercial Support by Databricks



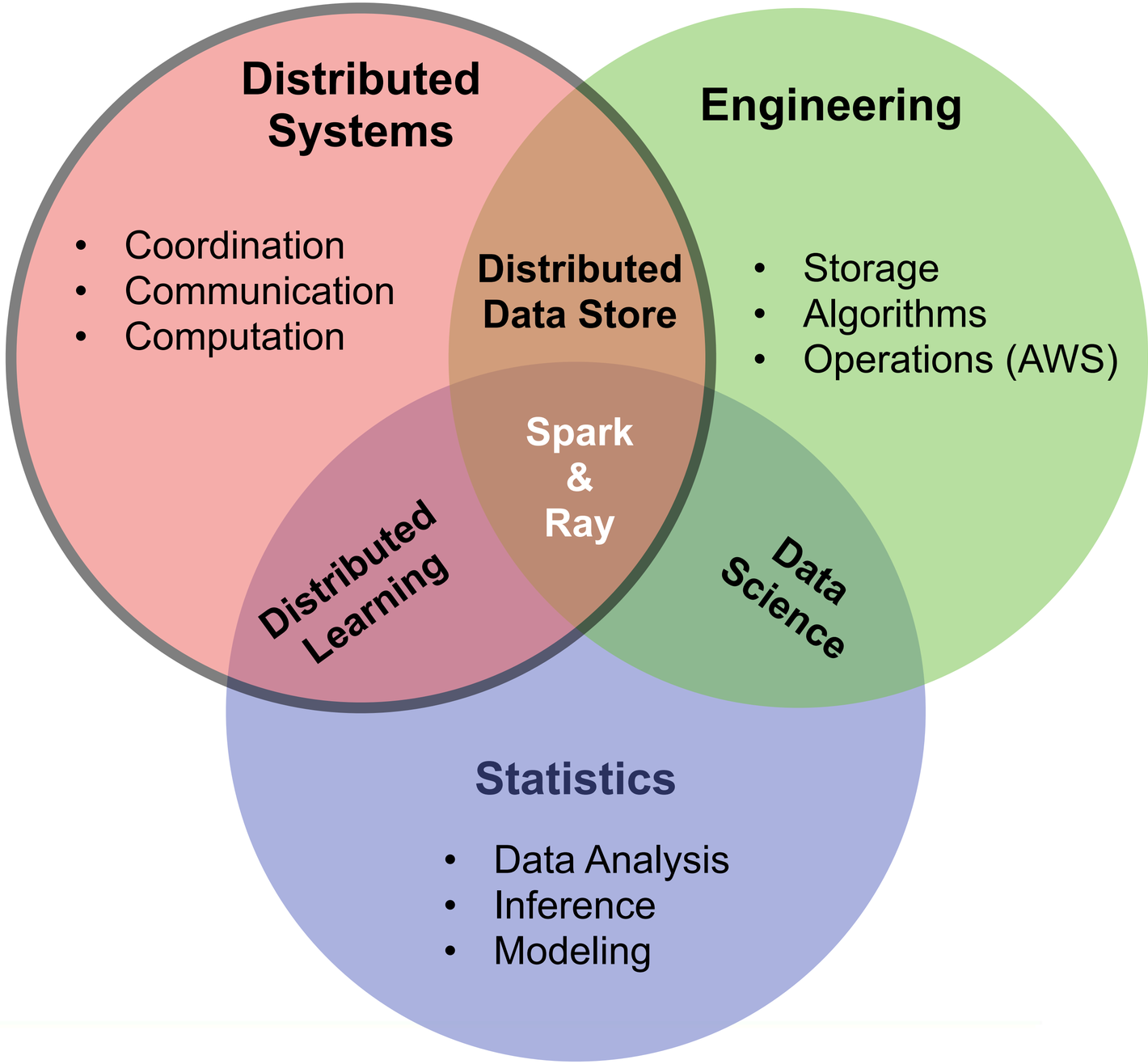
The Recent Future: ML Systems



> *The Data Engineer lives in the liminal space between distributed systems (CS theory), engineering (operations/infrastructure), and statistics. While it is not crucial for them to understand each of these domains wholly, being able to synthesize concepts from each is essential for success…*

- Jonathan Dinu

We are Here


Distributed Systems
• Coordination
• Communication
• Computation
Engineering
• Storage
• Algorithms
• Operations (AWS)
Distributed Data Store
Spark & Ray
Distributed Learning
Data Science
Statistics
• Data Analysis
• Inference
• Modeling

So why do I need to know this?

- **Logic:** if you understand the system you are programming, you can reason about your code much more effectively.
- **Debugging:** if you understand the system you are programming, you can fix your code much faster.
- **Performance:** if you understand the system you are programming, you can write smarter (and more optimal) code.

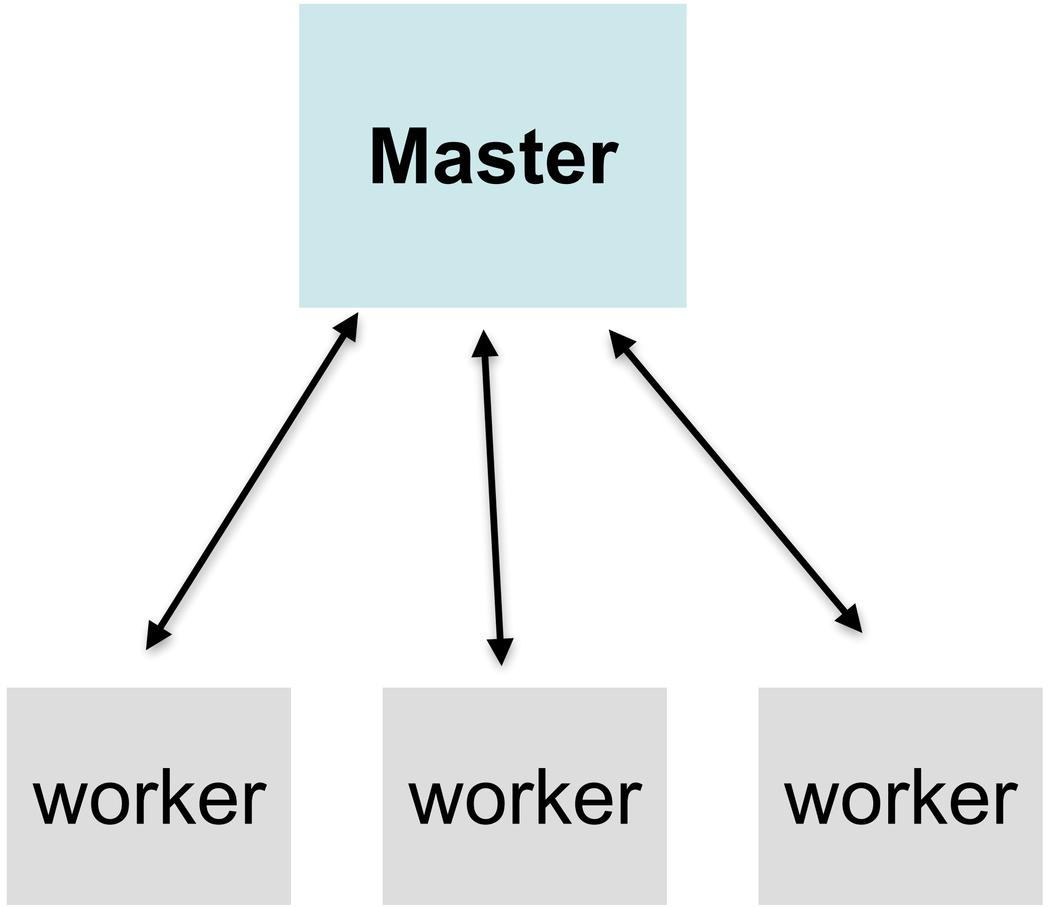
> *Distributed programming is the art of solving the same problem that you can solve on a single computer using multiple computers.*

- Distributed Systems for Fun and Profit, Mikito Takada

Local

Single Machine

Distributed



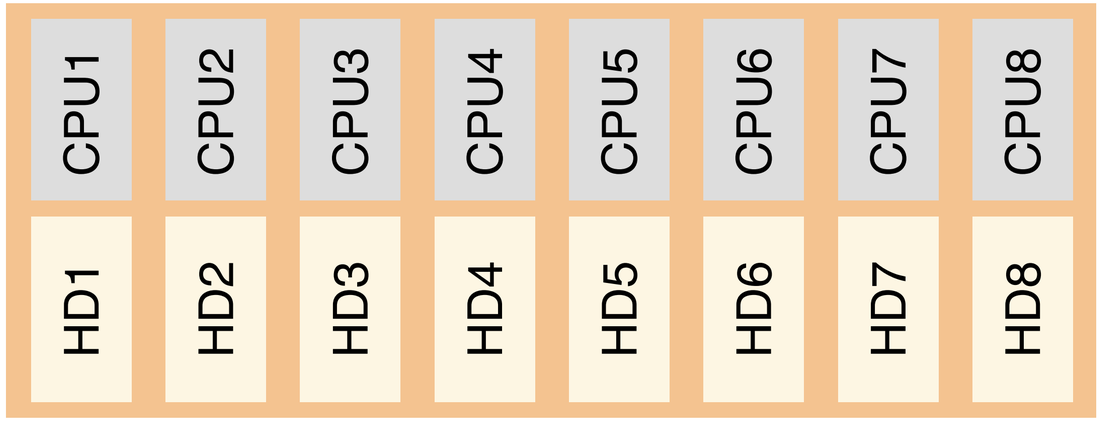
Why Distribute?

- Problem no longer fits on a single machine (storage)
- Problem need to be solved faster (computation)
- Often a combination of the two.

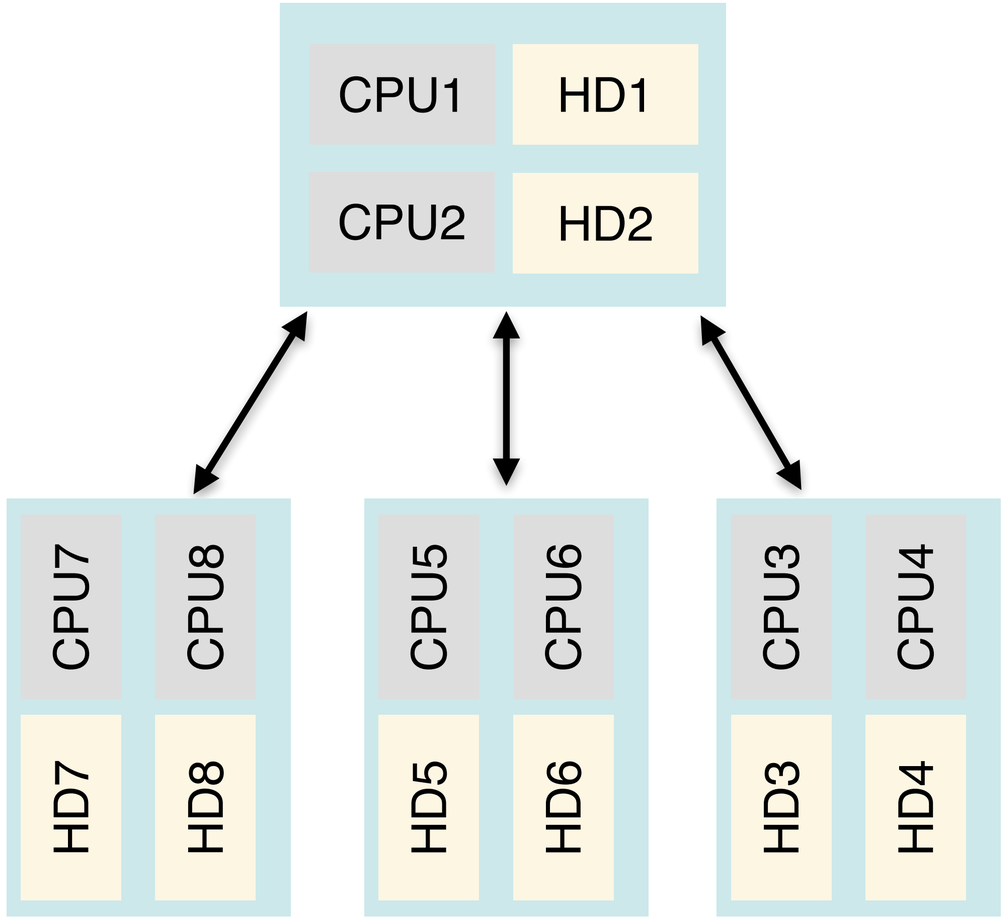
Availability is also often sought after and achieved by distributing a system but in the context of Spark we will concern ourselves with storage and computation since its programing model can be thought to be always available.

How to Distribute

Local



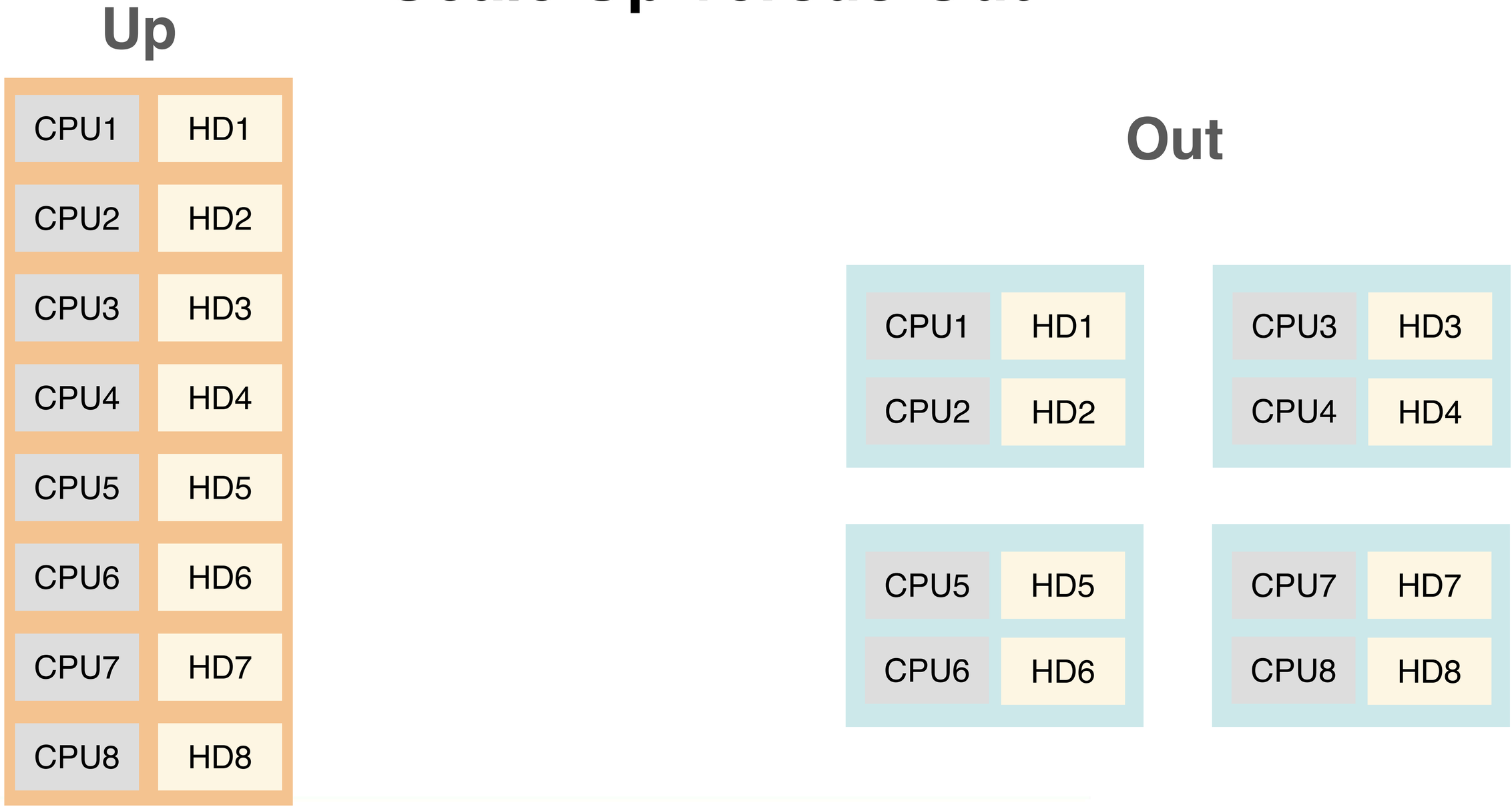
Distributed



Storage: Hard drives

Computation: CPUs

Scale Up versus Out



Problem Solved?

- **Coordination:** what happens when and to whom? And how do you synchronize these events
- **Communication:** how do the different nodes in your system talk to one another? And how do you talk to your nodes?
- **Fault Tolerance:** is your system robust to network and machine failures (because they will happen)?

CAP Theorem

- **Consistency:** all nodes see the same data at the same time
- **Availability:** every request receives a response about whether it succeeded or failed
- **Partition tolerance:** the system continues to operate despite arbitrary partitioning due to network failures

Choose two: except you can't really sacrifice P!

Review

- Different distributed architectures have different tradeoffs
- Scaling up is logically easier for programmers, and easier to reason about.
- Scaling out is theoretically unbounded, but coordinating and reasoning in a distributed system is hard
- You have to make compromises to ensure fault tolerance in a distributed system, though there are ways to mitigate these.

Scaling Data

Scaling Compute

TensorFlow

RAPIDS

DASK

PyTorch

> *The Data Engineer lives in the liminal space between distributed systems (CS theory), engineering (operations/infrastructure), and statistics. While it is not crucial for them to understand each of these domains wholly, being able to synthesize concepts from each is essential for success…*

- Jonathan Dinu

> *And Spark lives exactly at this nexus! Making it one of the most powerful frameworks for data scientists and engineers alike!*

- Jonathan Dinu

What is Spark

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

What is Spark

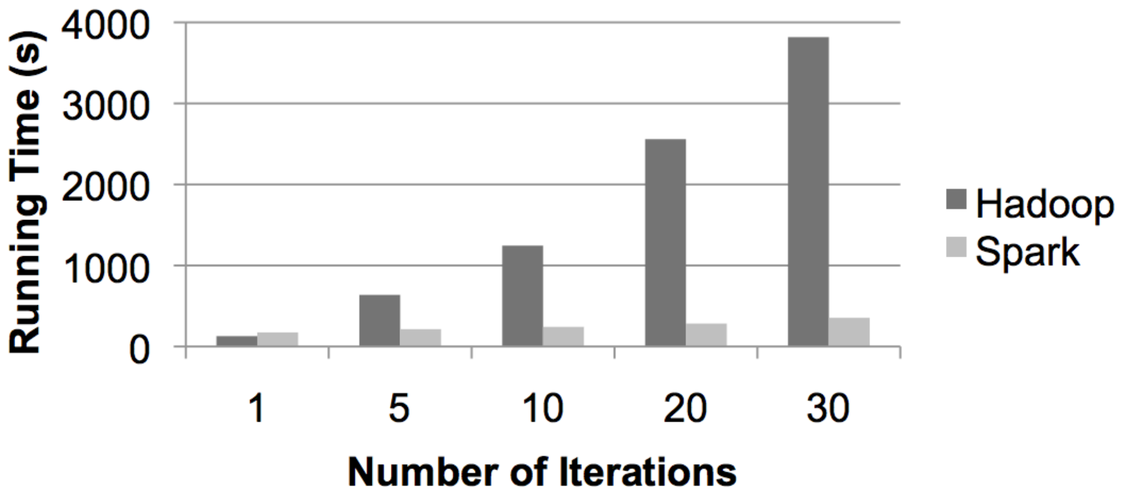
- Framework for distributed processing
- In-memory, fault tolerant data structures
- Flexible APIs in Scala, Java, Python, R, and SQL!
- Open Source

Why Spark?

- Handles Petabytes of data (and more!)
- Significantly faster than Hadoop Map-Reduce (for most jobs)
- Simple and intuitive APIs

- General Framework
  - Runs Anywhere
  - Handles (most) any I/O
  - Interoperable libraries for specific use-cases

Performance



Logistic Regression

- Very fast at iterative algorithms
- DAG scheduler supports cyclic flows (and graph computation)
- Intermediate results kept in memory when possible
- Bring computation to the data (data locality)

Rich API


```
map()          reduce()
filter()       sortBy()
join()         groupByKey()
first()        count()
```

… and more …

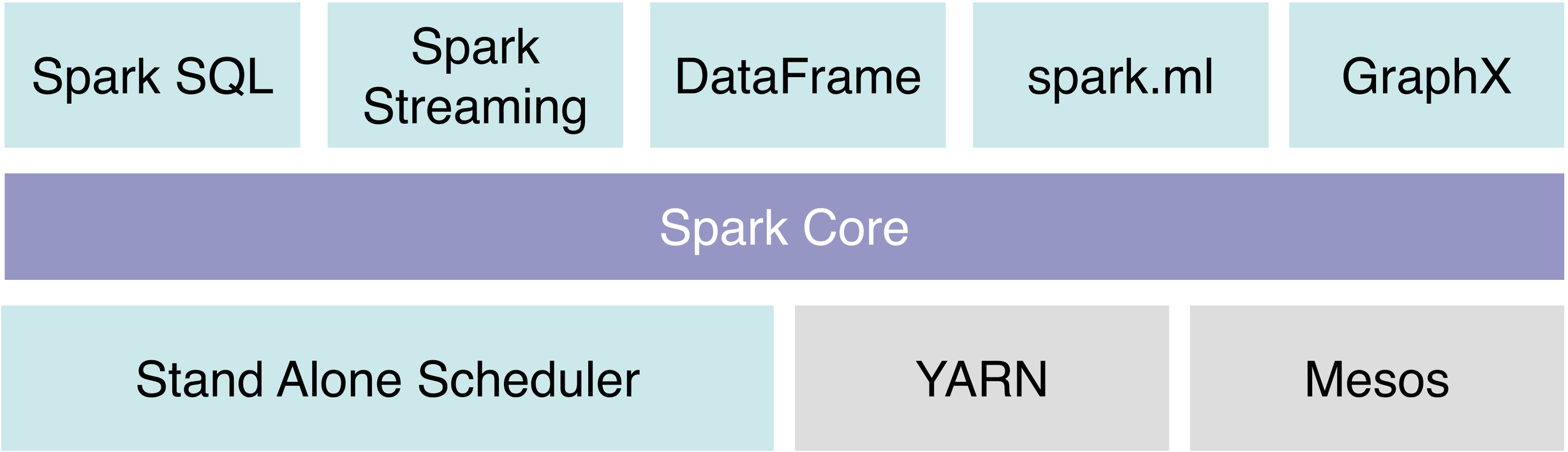

```
map()
reduce()
```

Rich API

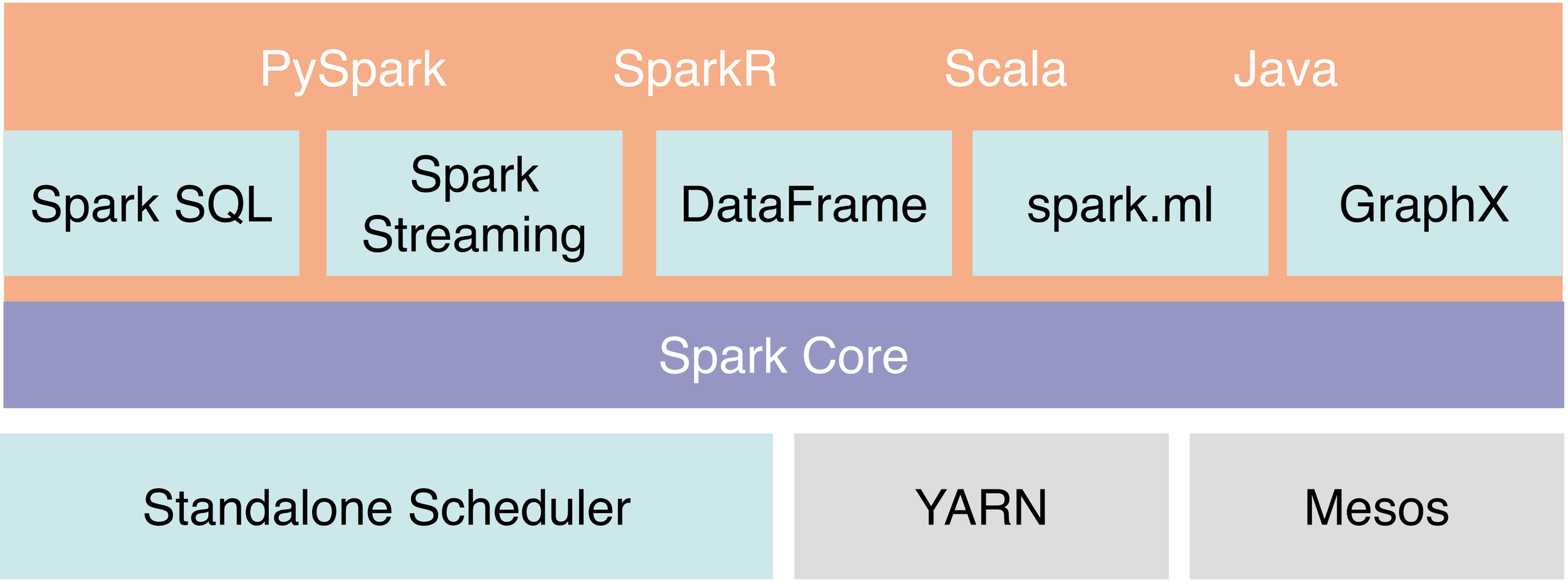

```
text_file = spark.textFile("hdfs://...")

text_file.flatMap(lambda line: line.split())
     .map(lambda word: (word, 1))
     .reduceByKey(lambda a, b: a+b)
```

```
package org.myorg;

import java.io.IOException;
import java.util.*;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class WordCount {

 public static class Map extends Mapper<LongWritable, Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(LongWritable key, Text value, Context context)
            throws IOException, InterruptedException {
        String line = value.toString();
        StringTokenizer tokenizer = new StringTokenizer(line);
        while (tokenizer.hasMoreTokens()) {
            word.set(tokenizer.nextToken());
            context.write(word, one);
        }
    }
 }

 public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {

    public void reduce(Text key, Iterable<IntWritable> values, Context context)
      throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }
        context.write(key, new IntWritable(sum));
    }
 }

 public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();

        Job job = new Job(conf, "wordcount");

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

    job.setMapperClass(Map.class);
    job.setReducerClass(Reduce.class);

    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.waitForCompletion(true);
 }

}
```

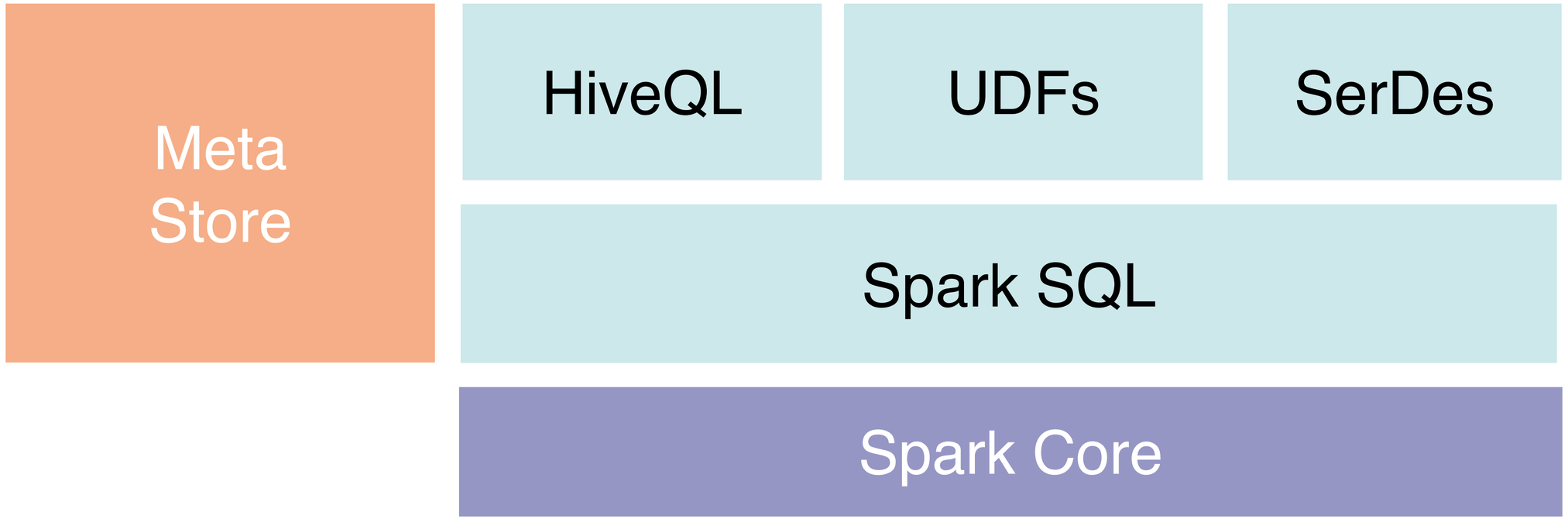
Unified Platform



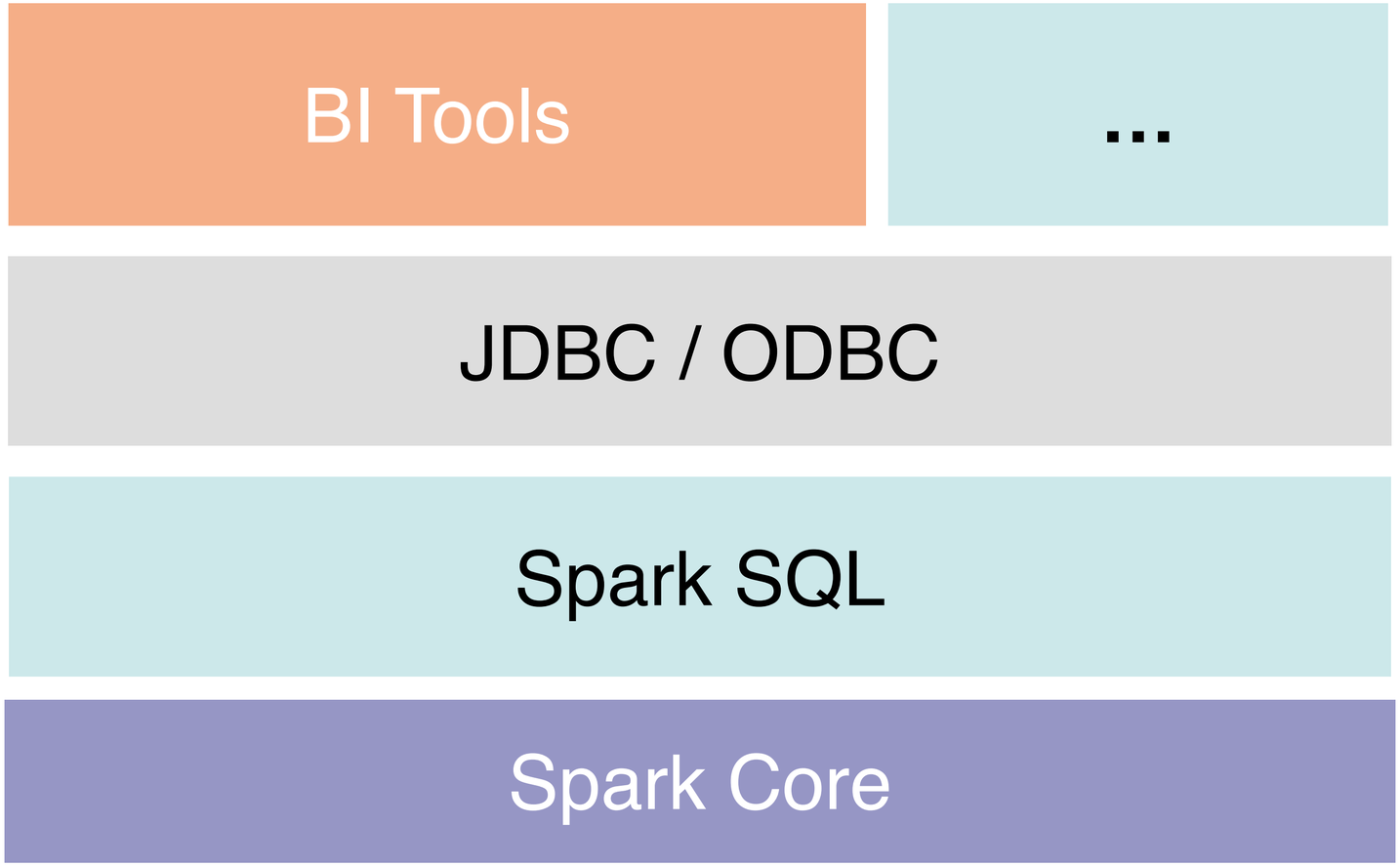
Unified Platform



Hive Integration



Standard RDBMS Integration



Review

- Framework for distributed processing
- In-memory, fault tolerant data structures
- Flexible APIs in Scala, Java, Python, SQL… and now R!
- Open Source

APIs

Spark

- Coarse grained
- SQL like abstractions (tables)
- Synchronous
- Functional

Ray

- Fine grained
- Actor based abstractions (agents)
- Asynchronous
- Object Oriented

Use Cases

Spark

- Data processing and transformation
- Data heavy workloads
- Iterative MapReduce
- Ex: Pagerank

Ray

- Distributed asynchronous algorithms
- Compute heavy workloads
- Ex: Reinforcement learning