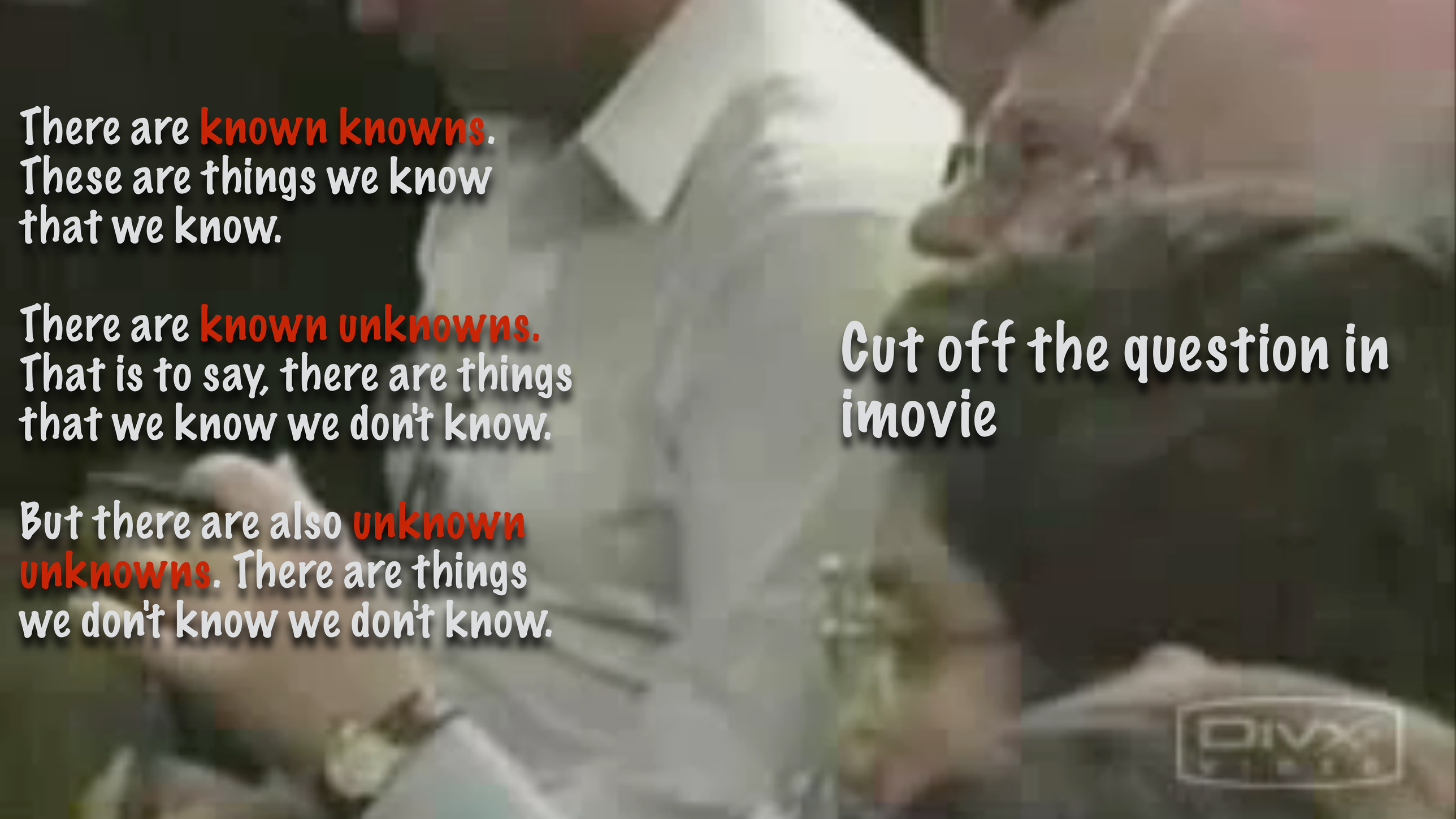
SPARK INTRO

US Secretary of Defense

Donald Rumsfeld

was once asked a tricky question


There are known knowns.
These are things we know
that we know.
There are known unknowns.
That is to say, there are things
that we know we don't know.
But there are also unknown
unknowns. There are things
we don't know we don't know.
Cut off the question in
imovie

There are **known knowns**. These are things we know that we know.

There are **known unknowns**. That is to say, there are things that we know we don't know.

But there are also **unknown unknowns**. There are things we don't know we don't know.

-Donald Rumsfeld

What seemed to be a clever evasion

There are **known knowns**. These are things we know that we know.

There are **known unknowns**. That is to say, there are things that we know we don't know.

But there are also **unknown unknowns**. There are things we don't know we don't know.

-Donald Rumsfeld

..went on to be recognized as a profound truth

Known knowns

Known unknowns

Unknown unknowns

This is a framework that can be applied to many things in life

Known knowns

Known unknowns

Unknown unknowns

In Behavioral psychology,

it represents a famous model about personal awareness

Personal Awareness

Known knowns	You know what you like and dislike
Known unknowns	You know that you don't understand rocket science and Greek
Unknown unknowns	You have an ability or weakness that you are unaware of

Personal Awareness

Known knowns	You know what you like and dislike
Known unknowns	You know that you don't understand rocket science and Greek
Unknown unknowns	You have an ability or weakness that you are unaware of

Things you need to seek feedback about

Known knowns

Known unknowns

Unknown unknowns

In Project Management

it represents a way to classify risks that need to be managed

Known knowns

Known unknowns

Unknown unknowns

In Data Analysis

it represents a way to classify types of insights we can get from the data

Known knowns

Data Analysis

Facts that you know and can confirm

Business working hours and holidays

Planned marketing events

Available production capacity

Known unknowns

Data Analysis

Questions that need to be answered on a regular basis

How much revenue did we earn yesterday?

How many customers visited our store?

What were our sales by product type?

Unknown unknowns

Data Analysis

Relationships and drivers that
you are unaware of

Searches for vacation
destinations are accompanied
by searches for diet tips

(Folks want to look good
when they go to the beach)

Unknown unknowns

Data Analysis

Relationships and drivers that
you are unaware of

Phones take much longer to be
packed and shipped compared
to Books

(Packaging instructions for
Phones are too complicated)

Unknown unknowns in data can only be identified through exploration and investigation

Unknown unknowns

Data Analysis

But they represent a huge opportunity

Unknown unknowns

Data Analysis

They can help

Shape Marketing plans

Build recommendation systems

Ex: Promote healthy recipe cook books during summer vacations

Unknown unknowns

Data Analysis

They can help

Identify process bottlenecks

Ex: Make the packaging instructions as simple as possible for all product types

Unknown unknowns

Data Analysis

They can help

Develop intelligent systems

Ex: Spam detection,
Fraud detection

Data Analysis

Unknown unknowns

Let's understand how these are identified and utilized traditionally

Data Analysis (Traditionally)

If your data is small enough

Excel is an excellent tool for data analysis

Data Analysis (Traditionally)

You can explore the data visually

You can interact with the data and immediately see the results of your operations

Data Analysis (Traditionally)

Very often though, datasets become too large and unwieldy to be managed in excel

Data Analysis (Traditionally)

In such cases you move to a programmatic environment

Data Analysis (Traditionally)

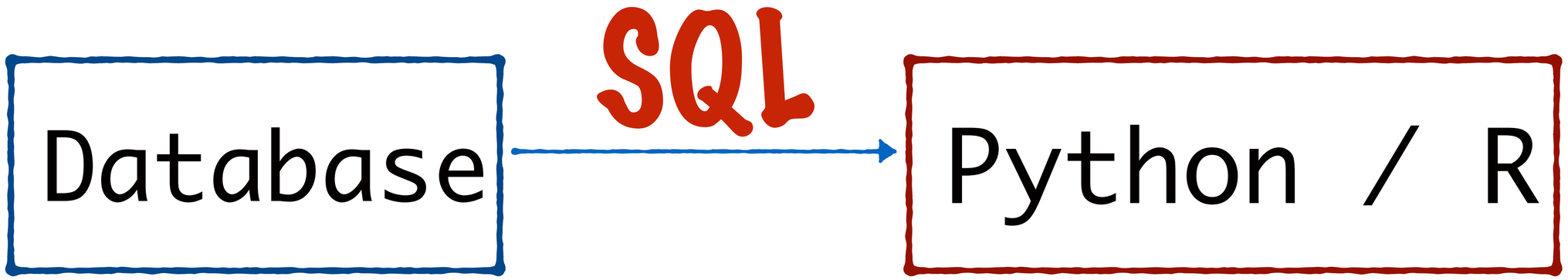
Data usually resides in a

Database

The data is accessed using

SQL

Data Analysis (Traditionally)



The data is explored in an interactive environment

Data Analysis (Traditionally)

an interactive environment → Python / R

Programming languages like Python, R, Scala etc provide a Read-Evaluate-Print Loop environment

Python / R

REPL

Read-Evaluate-Print Loop

An REPL is any shell/environment which

Takes in a single expression from the user

Evaluates it

Prints a result

Python / R

REPL

Read-Evaluate-Print Loop

An REPL environment for Python

```
>python
Python 2.7.11 (default, Jan 22 2016, 08:29:18)
[GCC 4.2.1 Compatible Apple LLVM 7.0.2 (clang-700.1.81)] on darwin
Type "help", "copyright", "credits" or "license" for more informatio
>>> len([1,2,5])
3
```

Python / R

REPL

Read-Evaluate-Print Loop

When you are looking for unknown unknowns in data, you want fast feedback

Python / R

REPL

Read-Evaluate-Print Loop

You want to quickly try out ideas and get a sense of whether they will lead anywhere

Python / R

REPL

Read-Evaluate-Print Loop

Because of their interactive nature,
REPL environments are excellent for
exploration and iteration

Python / R

REPL

Read-Evaluate-Print Loop

Languages like Java and C++ lack this interactive nature

Python / R

REPL

Read-Evaluate-Print Loop

You need to write complete programs, compile and execute them each and every time

Python / R

REPL

Read-Evaluate-Print Loop

With an REPL, you could

- Represent the data in a form suitable for analysis
- Explore and identify hidden patterns
- Confirm and quantify using Statistical models, machine learning
- Iterate

Python / R

REPL

Read-Evaluate-Print Loop

W

orm

n

ng

arning

Iterate

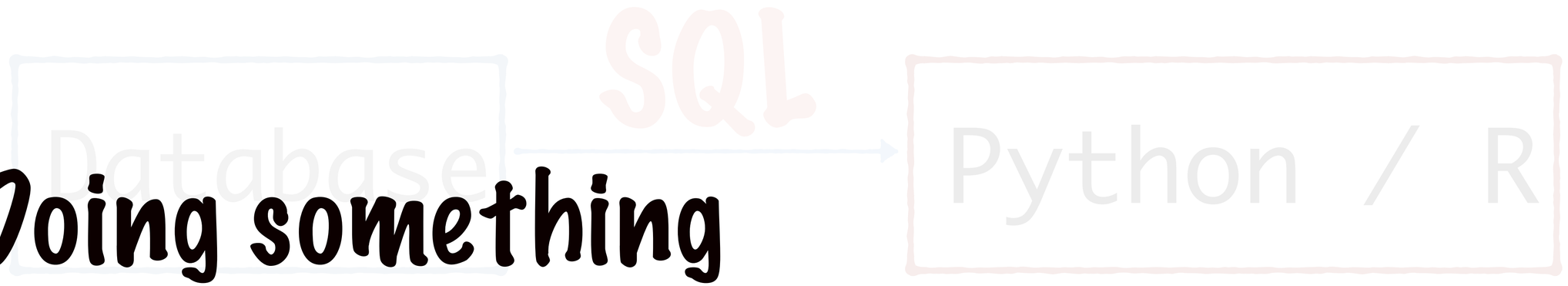
Do all of this with fast feedback at every step!

Data Analysis (Traditionally)

Once an interesting and useful insight/ model is found

You want to do something with it

Data Analysis (Traditionally)



Doing something useful usually means

Taking your models and building systems like recommendation/ fraud detection etc

Data Analysis (Traditionally)

Recommendation/fraud detection etc

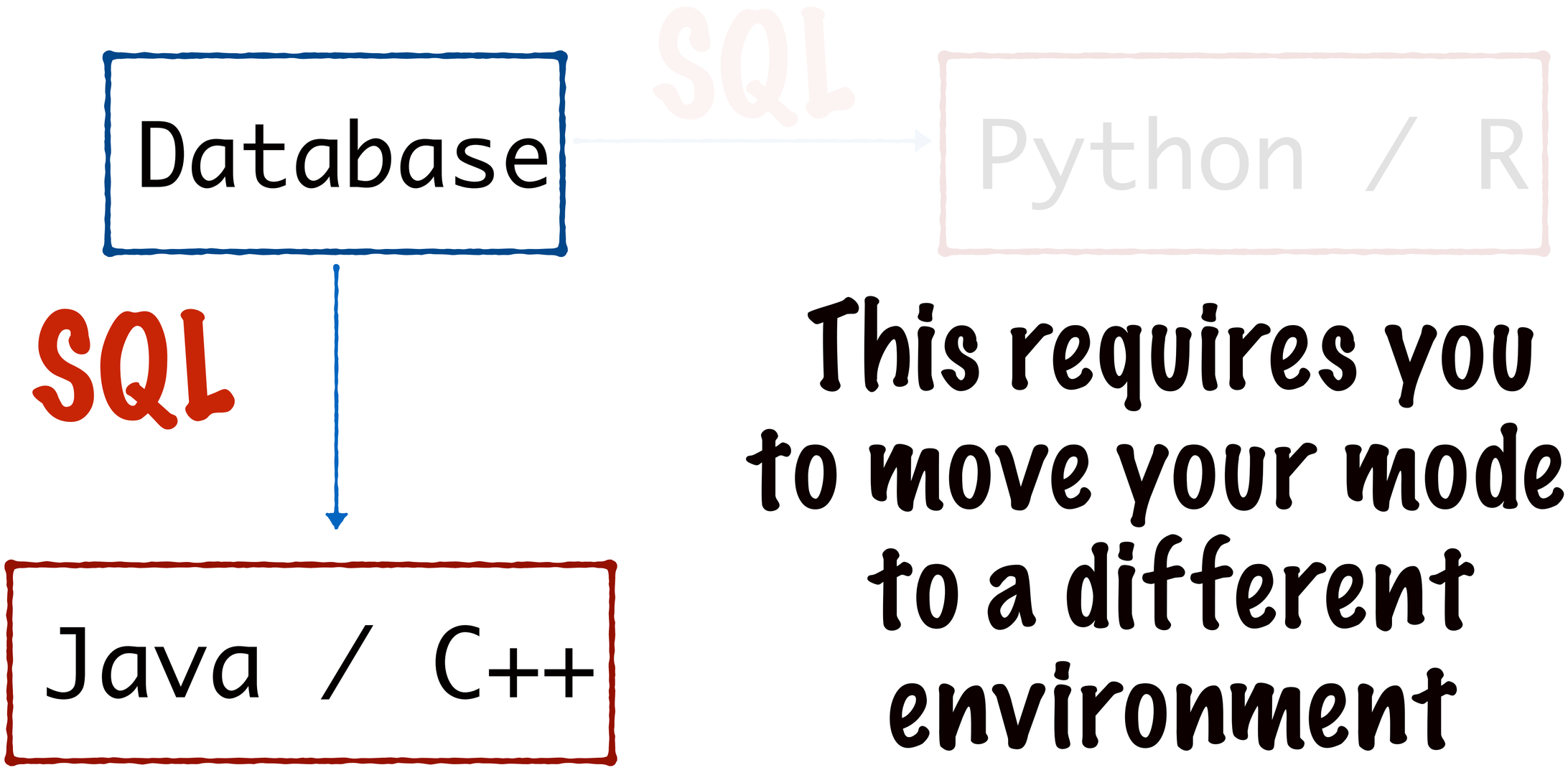
These systems will be used for taking important decisions that impact the business

Data Analysis (Traditionally)

Recommendation/fraud detection etc

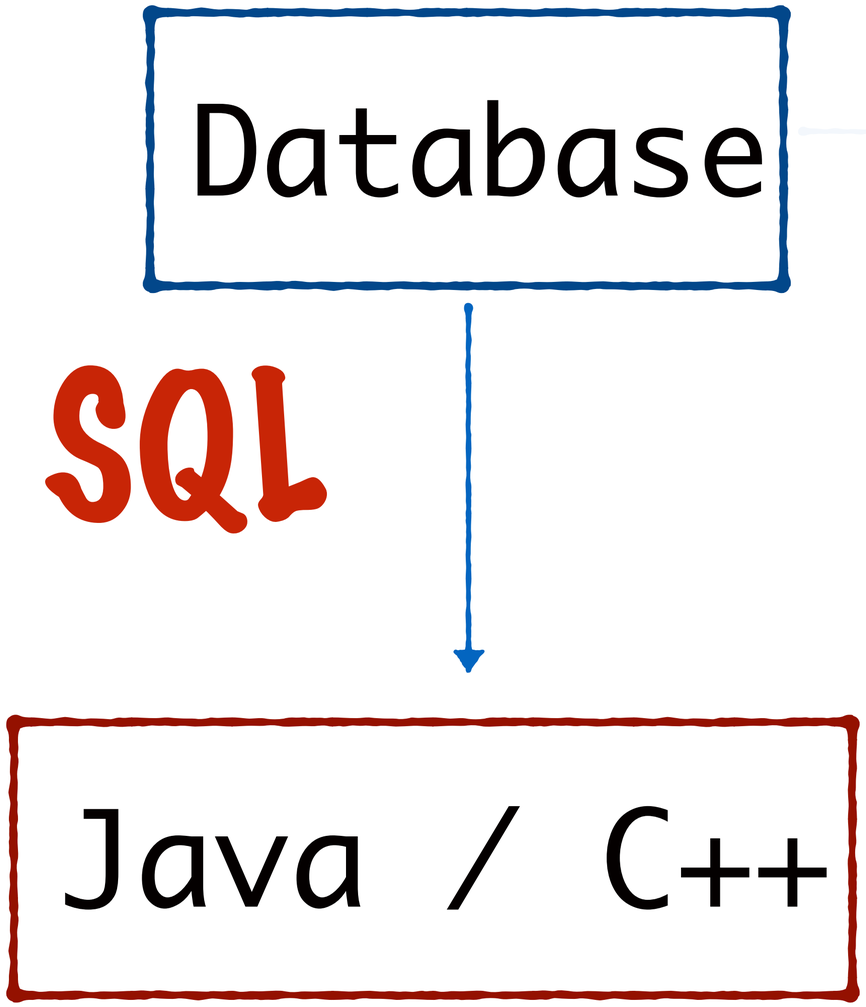
They have strict requirements around SLAs, performance etc

Data Analysis (Traditionally)



This requires you to move your model to a different environment

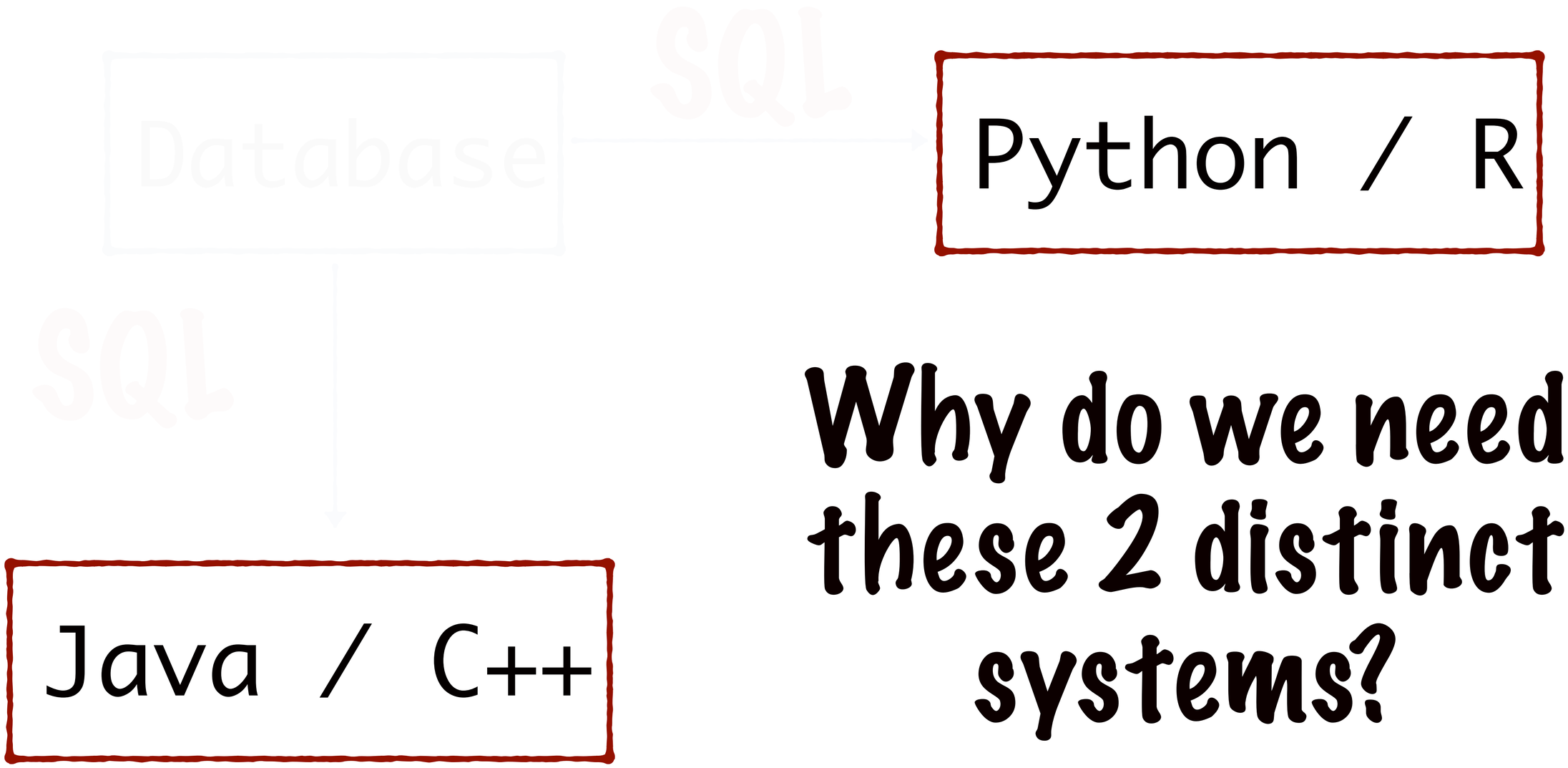
Data Analysis (Traditionally)



It means that you (or another engineer)

will end up re-implementing all your code in Java/C++

Data Analysis (Traditionally)



Why do we need these 2 distinct systems?

Data Analysis (Traditionally)

Python / R

Python and R have REPL environments that are useful for exploration and iteration

Java / C++

But they cannot meet the performance and SLA requirements of a production system

Data Analysis (Traditionally)

Python / R

Python and R have REPL environments that are useful for exploration and iteration

But they don't have REPL environments that provide fast feedback

Java / C++

Java and C++ are great for building stable and performant systems with strict SLAs

Data Analysis (Traditionally)

Python / R

Java / C++

Read-Evaluate-Print Loop

Python and R have REPL environments that are useful for exploration and iteration

Java and C++ are great for building stable and performant systems with strict SLAs

Traditionally

Python / R

Java / C++

APACHE SPARK

REPL environments
Exploration and iteration

Stable and performant
Strict SLAs

APACHE SPARK

Spark is a general-purpose engine
for data processing and analysis

APACHE SPARK

Spark is a general-purpose engine
for data processing and analysis

One engine that does the jobs of
SQL, Python/R and Java/C++

APACHE SPARK

Spark was built using Scala

But you don't need to know
Scala to use it

APACHE SPARK

Spark provides APIs in
Scala, Python and Java

It also provides interactive REPL
environments for Python and Scala

APACHE SPARK

Spark is a part of the
Hadoop ecosystem

APACHE SPARK

It's engine is capable of
Distributed Computing

i.e processing data stored
across a cluster of machines

APACHE SPARK

Spark was built to overcome some of the limitations of Hadoop's MapReduce engine

Limitations of Hadoop's MapReduce engine

Everything has to be expressed as a chain of map and reduce tasks

Limitations of Hadoop's MapReduce engine

Everything has to be expressed as a chain of map and reduce tasks

No Interactive environment

Limitations of Hadoop's MapReduce engine

Everything has to be expressed as a chain of map and reduce tasks

No Interactive environment

Disk writes occur at the end of each intermediate map-reduce task

Limitations of Hadoop's MapReduce engine

Everything has to be expressed as a chain of map and reduce tasks

No Interactive environment

Disk writes occur at the end of each intermediate map-reduce task

Can only do batch processing

ie. files stored on disk

MapReduce

Everything has to be expressed as a chain of map and reduce tasks

No Interactive environment

Disk writes occur at the end of each intermediate map-reduce task

Can only do batch processing

SPARK

Operations can be expressed in a very intuitive way

MapReduce

Everything has to be expressed as a chain of map and reduce tasks

No Interactive environment

Disk writes occur at the end of each intermediate map-reduce task

Can only do batch processing

SPARK

very intuitive

Interactive shells are available for Python and Scala

MapReduce

Everything has to be expressed as a chain of map and reduce tasks

No Interactive environment

Disk writes occur at the end of each intermediate map-reduce task

Can only do batch processing

SPARK

very intuitive
Interactive

Data is kept in-memory and can be passed directly to the next step

MapReduce

Everything has to be expressed as a chain of map and reduce tasks

No Interactive environment

Disk writes occur at the end of each intermediate map-reduce task

Can only do batch processing

SPARK

very intuitive
Interactive
in-memory

Can do stream processing

ex: a stream of status messages from a web service

SPARK

very intuitive

Interactive

in-memory

Stream Processing

How does Spark do all of this?

SPARK

Using Resilient Distributed Datasets

RDD

Resilient Distributed Datasets

RDDs are the main programming abstraction in Spark

Resilient Distributed Datasets

RDDs are in-memory objects and all data is processed using them

Resilient Distributed Datasets

in-memory

In-memory, yet resilient!

RDDs are designed to be fault-tolerant while dealing with vast amounts of data

Resilient Distributed Datasets

There are a lot of complexities involved in dealing with vast amounts of data

Resilient Distributed Datasets

Lot of complexities

1. Distributing data across a cluster
2. Fault tolerance (if in-memory data is lost)
3. Efficiently processing billions of rows of data

Resilient Distributed Datasets

1. Distributing data across a cluster
2. Fault tolerance
3. Efficiently processing billions of rows of data

Think of other environments
like Java or Excel

It would be the responsibility of the
user to deal with most of these issues

Resilient Distributed Datasets

1. Distributing data across a cluster
2. Fault tolerance
3. Efficiently processing billions of rows of data

With RDDs, you can interact and play with billions of rows of data

...without caring about any of the complexities

Resilient Distributed Datasets

1. Distributing data across a cluster
2. Fault tolerance
3. Efficiently processing billions of rows of data

Spark takes care of all the complexities under the hood

The user is completely abstracted!

APACHE SPARK

Spark is made up of a few different components

APACHE SPARK

Spark Core

Spark Core contains the basic functionality of Spark

APACHE SPARK

Spark Core

It provides an API for working with RDDs

APACHE SPARK

Spark Core

Spark Core is just a computing engine

APACHE SPARK

Spark Core

It needs two additional components

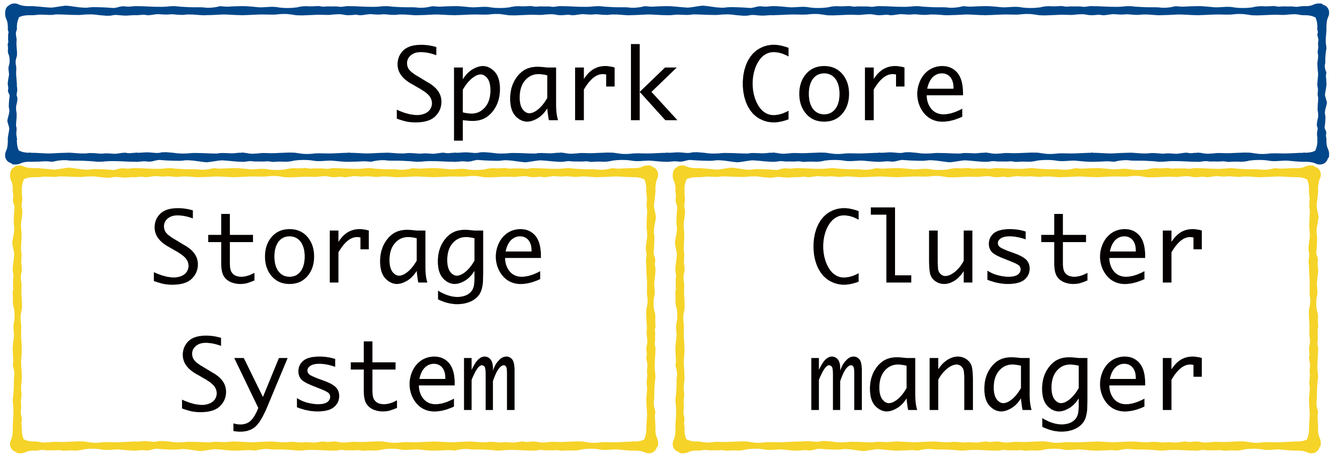
APACHE SPARK

Spark Core

A **Storage System** that stores the data to be processed

A **Cluster manager** to help Spark run tasks across a cluster of machines

APACHE SPARK



Both of these are plug and play components

APACHE SPARK

Spark Core

Storage System

Cluster manager

Could be a Local file system, HDFS, Hive, HBase, Cassandra etc

APACHE SPARK

The cluster manager schedules tasks and manage resources across the cluster

APACHE SPARK

Spark comes with a built-in Cluster manager

APACHE SPARK

But you can plug and play other options

APACHE SPARK

Apache Mesos was the original Cluster Manager used when Spark was developed

APACHE SPARK

Hadoop is a very popular choice for distributed computing and already widely deployed

APACHE SPARK

Spark Core

Storage System

Cluster manager

If you already have a Hadoop cluster, you can plug in Hadoop's YARN

Hadoop cluster

For storage

For managing the cluster

For computing

Hadoop cluster

HDFS

YARN

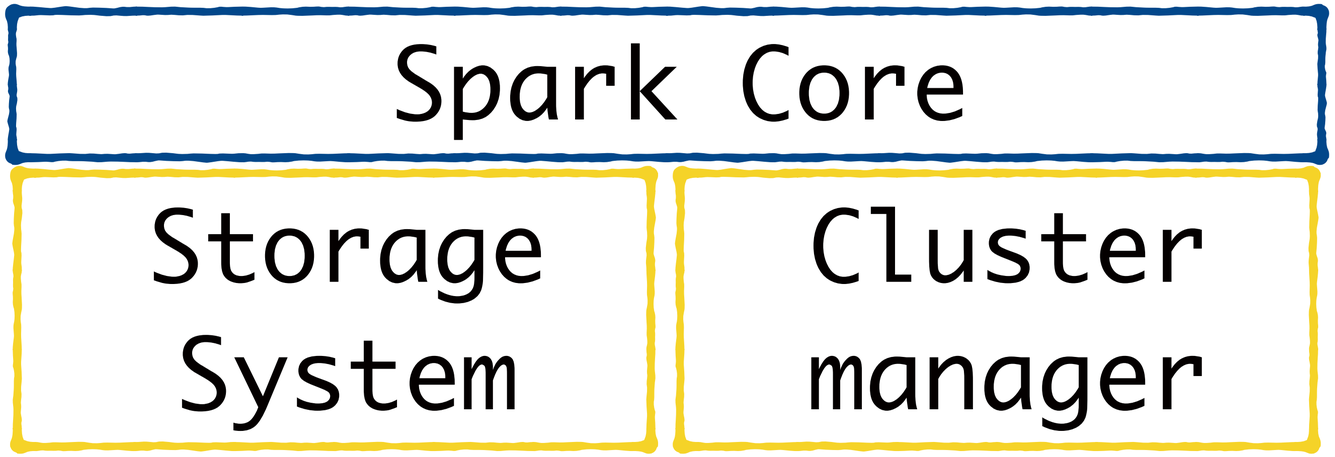
MapReduce

Spark Core

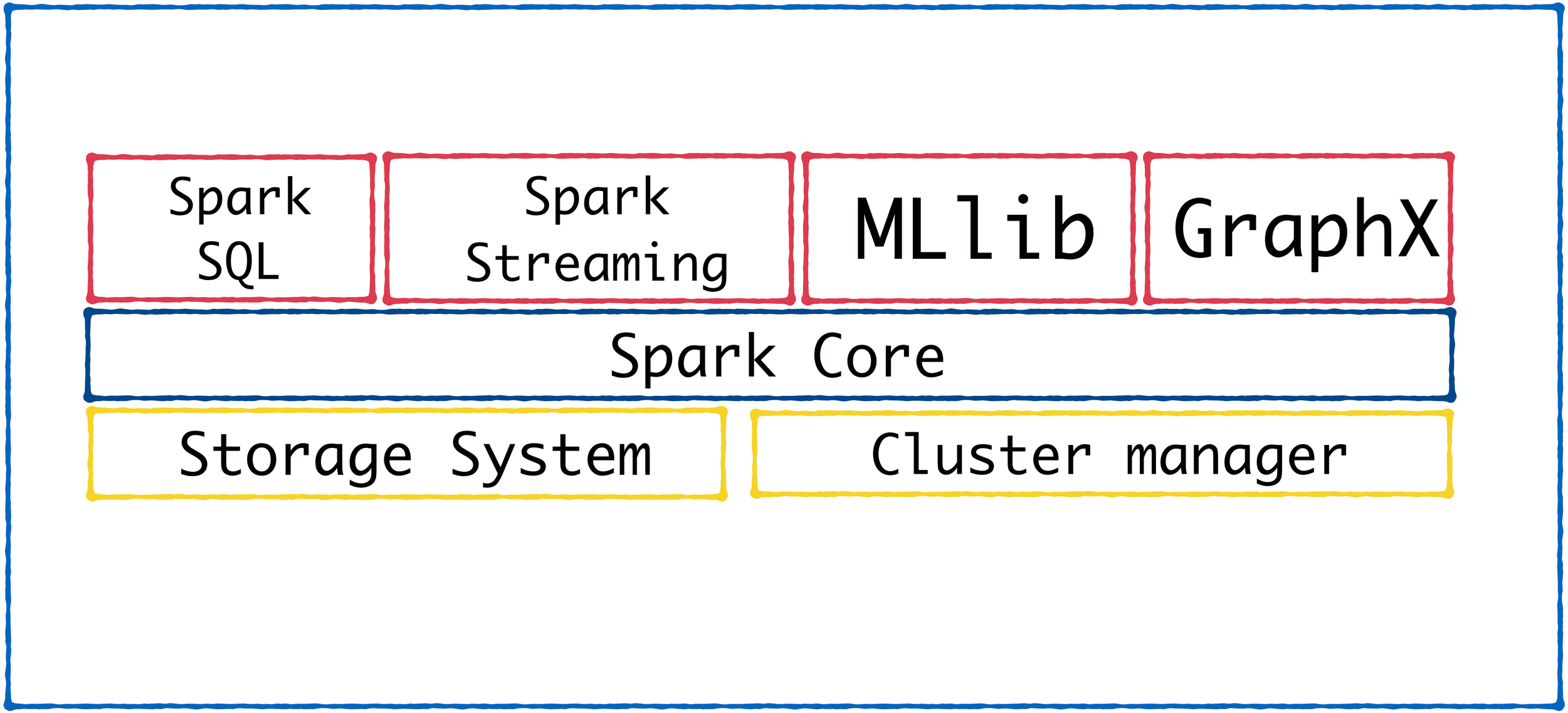
Use Spark as an alternate/ additional compute engine

APACHE SPARK

Spark comes with some additional packages that make it truly general - purpose



APACHE SPARK



APACHE SPARK

Spark SQL

Spark SQL provides an SQL interface for Spark

APACHE SPARK

Spark SQL

Lot's of folks are familiar with SQL and like to use to express data manipulation

APACHE SPARK

Spark SQL

With Spark SQL, folks can use SQL to work with large amounts of data stored in a cluster

APACHE SPARK

Spark SQL

You can use the familiar SQL and still get all the in-memory performance benefits of Spark

There's another use for Spark SQL

APACHE SPARK

Spark SQL

Let's say you were performing some complex data manipulations in a program

APACHE SPARK

Spark SQL

Load data in table form into memory

Manipulate it using Spark SQL

ie. with select, group by , joins etc

APACHE SPARK

Spark SQL

Load data in table form into memory

Manipulate it using Spark SQL

Convert it to Python/Java objects and work on them

APACHE SPARK

Spark SQL

Spark allows programmers to mix SQL manipulations with Python/Java data manipulations on the same dataset and within a single program

APACHE SPARK

Spark SQL

Spark Streaming

MLlib

GraphX

Spark Core

Storage System

Let's say you have a live stream of data

APACHE SPARK

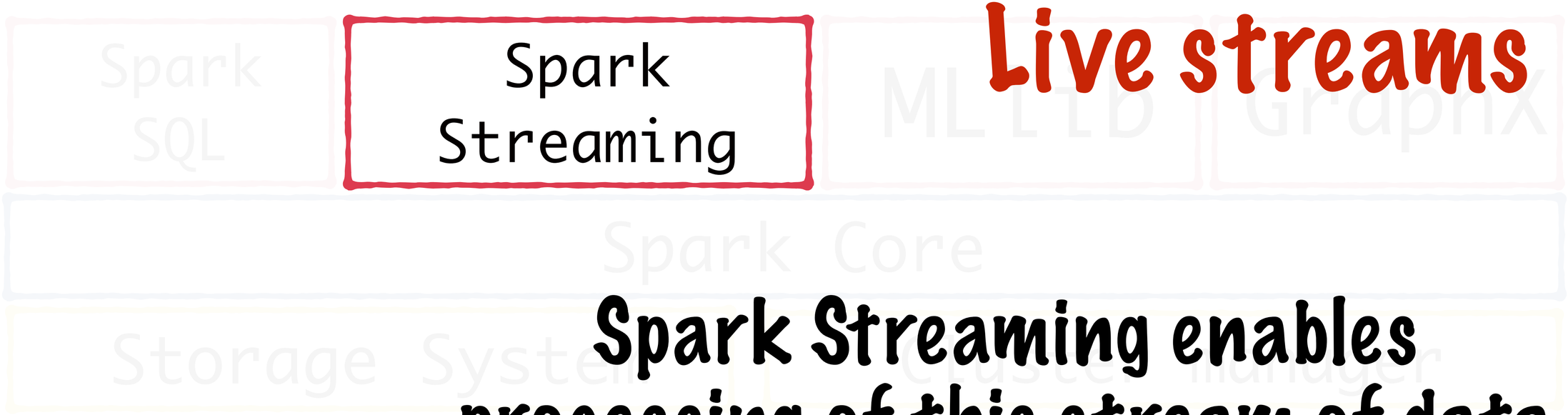
Spark Streaming

Live streams

Logs generated by a web server

Status updates posted by users

APACHE SPARK



Live streams

Spark Streaming enables processing of this stream of data in (near) real time

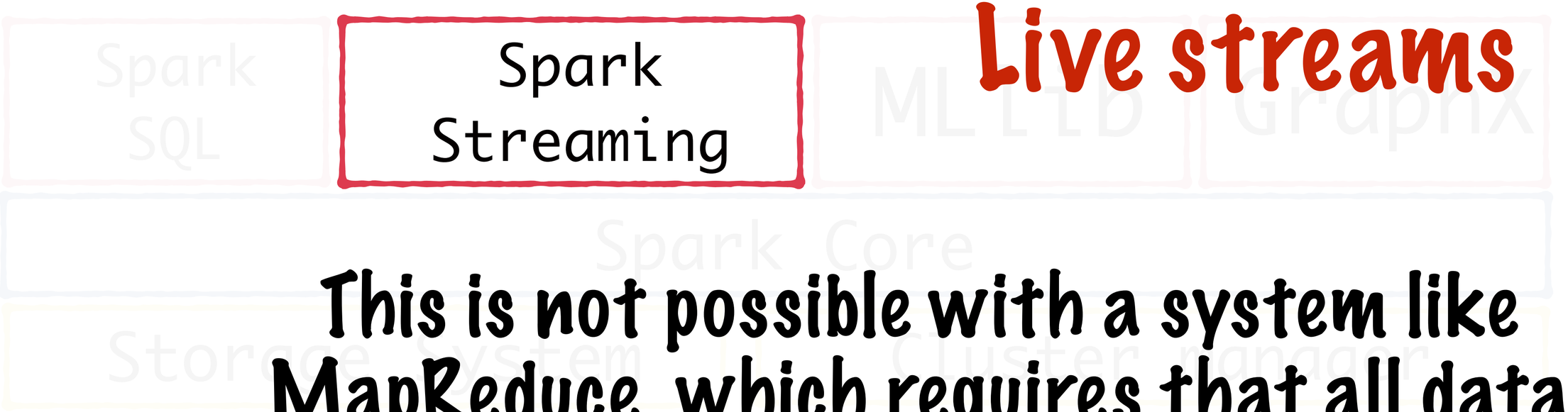
APACHE SPARK

Spark Streaming

Live streams

You can process logs for reporting, monitoring and react to them in real time

APACHE SPARK



This is not possible with a system like MapReduce, which requires that all data be written to disk before it's processed

APACHE SPARK

Whenever you have a large amount of data, **Machine Learning** is a great way to identify patterns within it

APACHE SPARK

Spark SQL | Spark Streaming | MLlib | GraphX

Spark Core

Storage Systems | Cluster Managers

MLlib provides built-in Machine Learning functionality in Spark

APACHE SPARK

This basically solves 2 problems with previous computing frameworks

MLlib

Abstraction

Performance

APACHE SPARK

Abstraction

MLlib

Machine Learning algorithms are pretty complicated

APACHE SPARK

Abstraction

Python and R have libraries which allow you to plug and play ML algorithms with minimal effort

MLlib

These libraries are not suited for distributed computing

APACHE SPARK

Abstraction

MLlib

Hadoop MapReduce is great for distributed computing, but it requires you to implement the algorithms yourself as map and reduce tasks

APACHE SPARK

Abstraction

MLlib

MLlib has Built-in methods for Classification, regression, clustering etc algorithms are provided

APACHE SPARK

Spark SQL | Spark Streaming | MLlib | GraphX

Spark Core

Storage System | Cluster manager

Abstraction

Under the hood the library takes care of running these algorithms across a cluster

APACHE SPARK

Abstraction

MLlib

This completely abstracts the programmer from

Implementing the ML algorithm

Intricacies of running it across a cluster

APACHE SPARK

This basically solves 2 problems with previous computing frameworks

MLlib

Abstraction

Performance

APACHE SPARK

Performance

Machine Learning algorithms are iterative, which means you need to make multiple passes over the same data

Hadoop MapReduce is heavy on disk writes which is not efficient for Machine Learning

APACHE SPARK

Performance

Since Spark's RDDs are in-memory
It can make multiple passes over the same data without doing disk writes

APACHE SPARK

Spark SQL | Spark Streaming | MLlib | GraphX

Spark Core

Storage System | Cluster Manager

GraphX is a library for graph algorithms

APACHE SPARK

Spark SQL | Spark Streaming | MLlib | GraphX

Spark Core

Storage Systems | Cluster Managers

Many interesting datasets can be represented as graphs

APACHE SPARK

Spark SQL | Spark Streaming | MLlib | GraphX

Spark Core

Storage System | Cluster Manager

Social networks, linked webpages etc

APACHE SPARK

Spark SQL | Spark Streaming | MLlib | GraphX

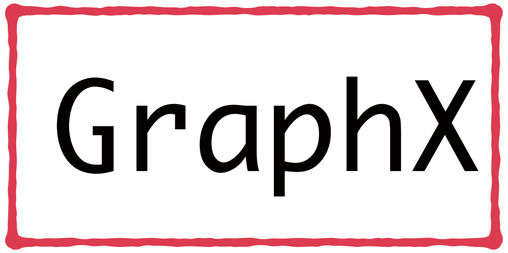
Spark Core

Storage System | Cluster manager

With GraphX you can represent and then perform computations across these datasets

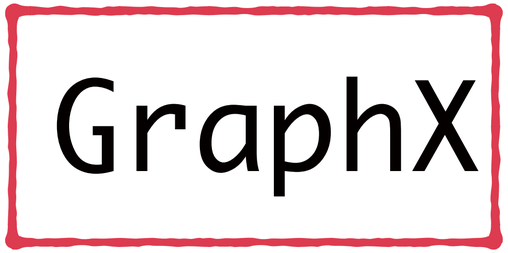
APACHE SPARK

Just like with MLlib, GraphX abstracts the programmer from having to implement Graph algorithms

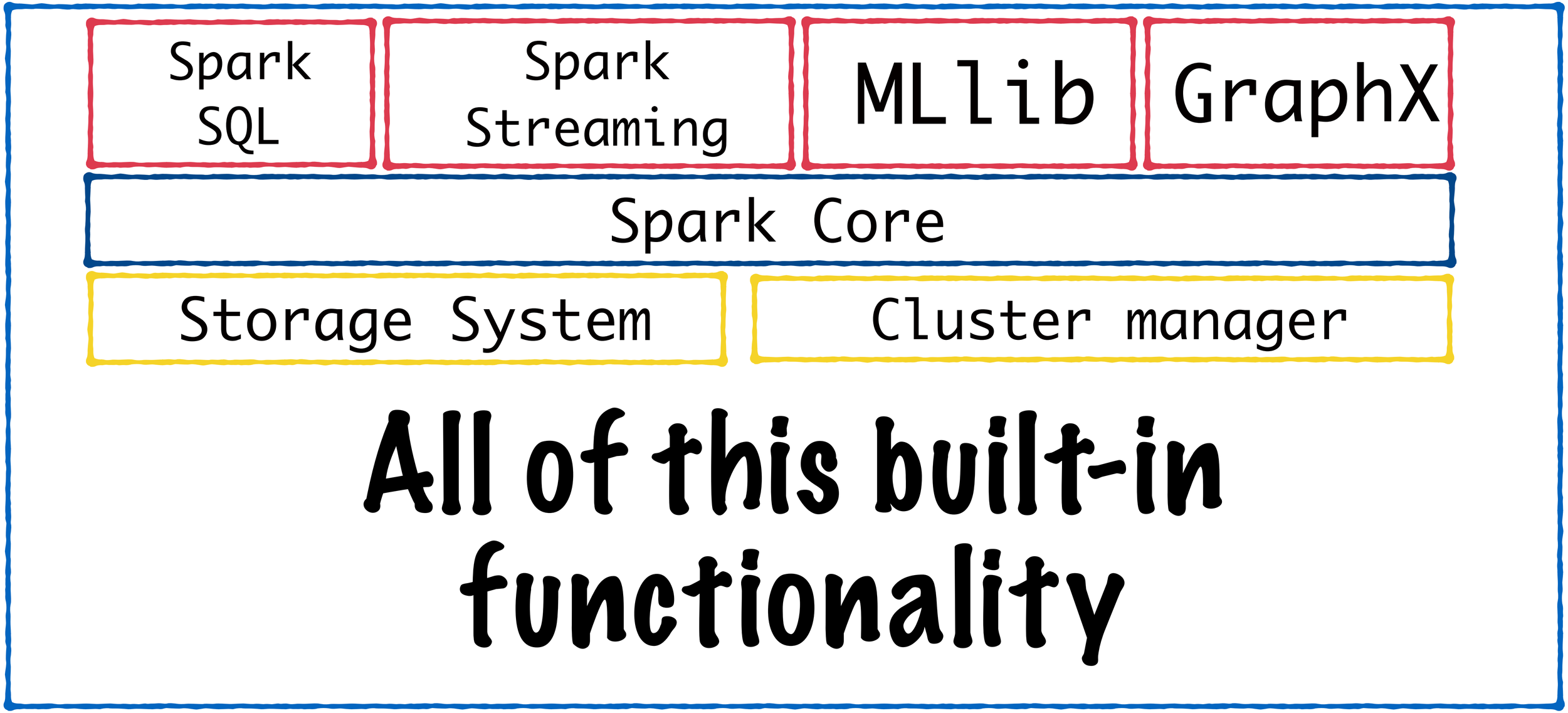


APACHE SPARK

Because of RDDs and in-memory computation, GraphX on Spark gives you **better performance** than other computing frameworks (like MapReduce)

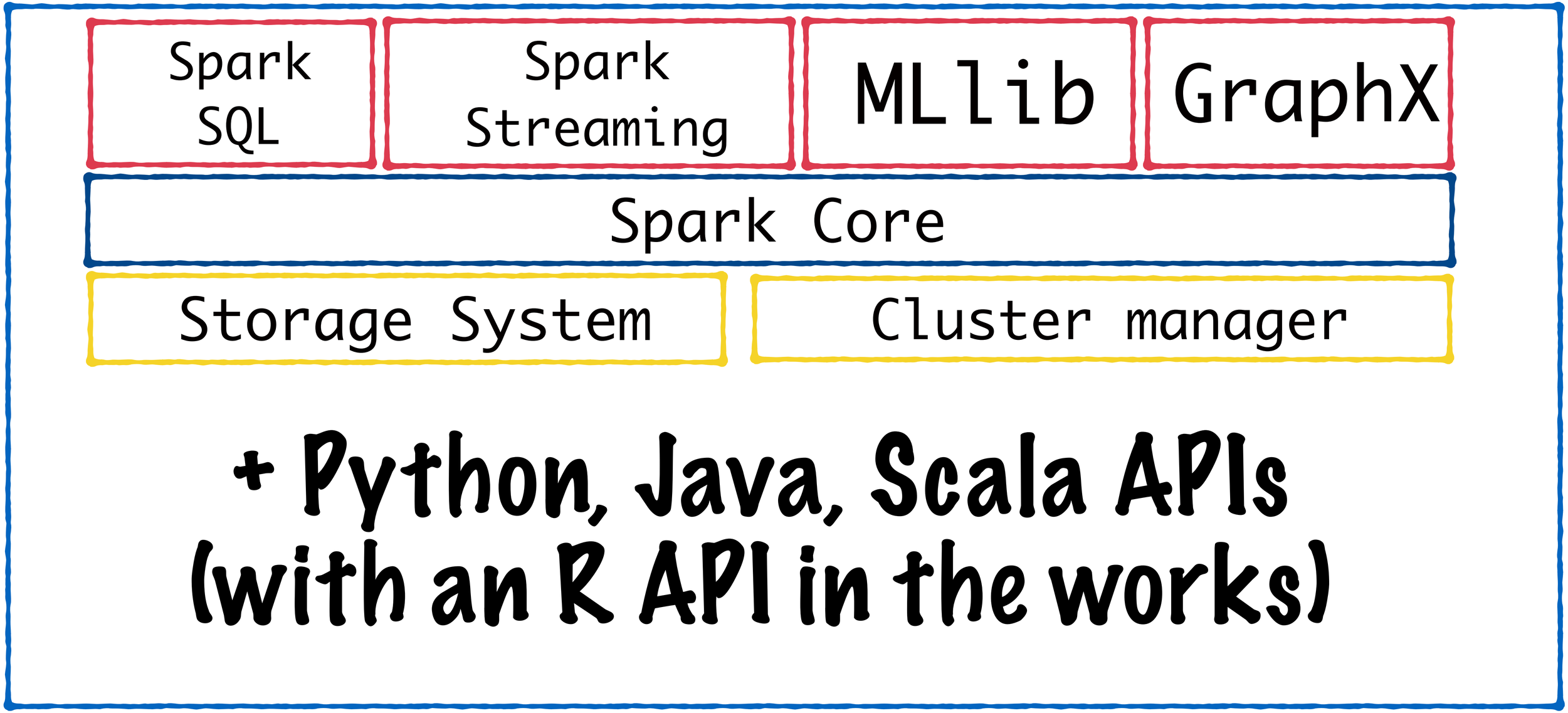


APACHE SPARK



All of this built-in functionality

APACHE SPARK



+ Python, Java, Scala APIs
(with an R API in the works)

APACHE SPARK

Installing Spark CodeAlong

Data Analysis using The PySpark Shell

Once you have installed Spark, you can launch the PySpark Shell from the commandline

```
> pyspark
```

```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 1.6.1
      /_/

Using Python version 2.7.11 (default, Jan 22 2016 08:29:18)
SparkContext available as sc, HiveContext available as sqlContext.
>>>
```

> pyspark

```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 1.6.1
      /_/

Using Python version 2.7.11 (default, Jan 22 2016 08:29:18)
SparkContext available as sc, HiveContext available as sqlContext.
>>>
```

This is just like a Python shell

> pyspark

```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 1.6.1
      /_/

Using Python version 2.7.11 (default, Jan 22 2016 08:29:18)
SparkContext available as sc, HiveContext available as sqlContext.
>>>
```

You can use Python functions, dicts, lists etc

> pyspark

```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 1.6.1
      /_/

Using Python version 2.7.11 (default, Jan 22 2016 08:29:18)
SparkContext available as sc, HiveContext available as sqlContext.
>>>
```

You can import and use any installed Python modules

> pyspark

```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 1.6.1
      /_/

Using Python version 2.7.11 (default, Jan 22 2016 08:29:18)
SparkContext available as sc, HiveContext available as sqlContext.
>>>
```

You can even configure Pyspark to launch in an ipython notebook

> pyspark

```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 1.6.1
      /_/

Using Python version 2.7.11 (default, Jan 22 2016 08:29:18)
SparkContext available as sc, HiveContext available as sqlContext.
>>>
```

By default, this launches Spark in a local non-distributed mode

```
> pyspark --master local[2]
```

By using the master option you can launch Spark in distributed mode and plug in different cluster managers

```
> pyspark --master local[2]
```

This tells Spark to launch the shell with multiple threads on the local machine

```
> pyspark --master local[2]
```

This is the number of threads (ideal is the number of cores on your machine)

```
> pyspark --master yarn-client
```

If you have a Hadoop cluster running, you can plug in YARN as the cluster manager

```
> pyspark --master yarn-client
```

```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 1.6.1
      /_/

Using Python version 2.7.11 (default, Jan 22 2016 08:29:18)
SparkContext available as sc, HiveContext available as sqlContext.
>>>
```

When the shell is launched it initializes a SparkContext (sc)

```
> pyspark --master yarn-client
```

```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 1.6.1
      /_/

Using Python version 2.7.11 (default, Jan 22 2016 08:29:18)
SparkContext available as sc, HiveContext available as sqlContext.
>>>
```

The SparkContext represents a connection to the Spark Cluster

```
> pyspark --master yarn-client
```

```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 1.6.1
      /_/

Using Python version 2.7.11 (default, Jan 22 2016 08:29:18)
SparkContext available as sc, HiveContext available as sqlContext.
>>>
```

This can be used to load data into memory from a specified source

> pyspark --master yarn-client

```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 1.6.1
      /_/

Using Python version 2.7.11 (default, Jan 22 2016 08:29:18)
SparkContext available as sc, HiveContext available as sqlContext.
>>>
```

The SparkContext object is required to create Resilient Distributed Datasets(RDDs)

> pyspark --master yarn-client

```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 1.6.1
      /_/

Using Python version 2.7.11 (default, Jan 22 2016 08:29:18)
SparkContext available as sc, HiveContext available as sqlContext.
>>>
```

All our data analysis will be through operations on RDDs

```
> pyspark --master yarn-client
```

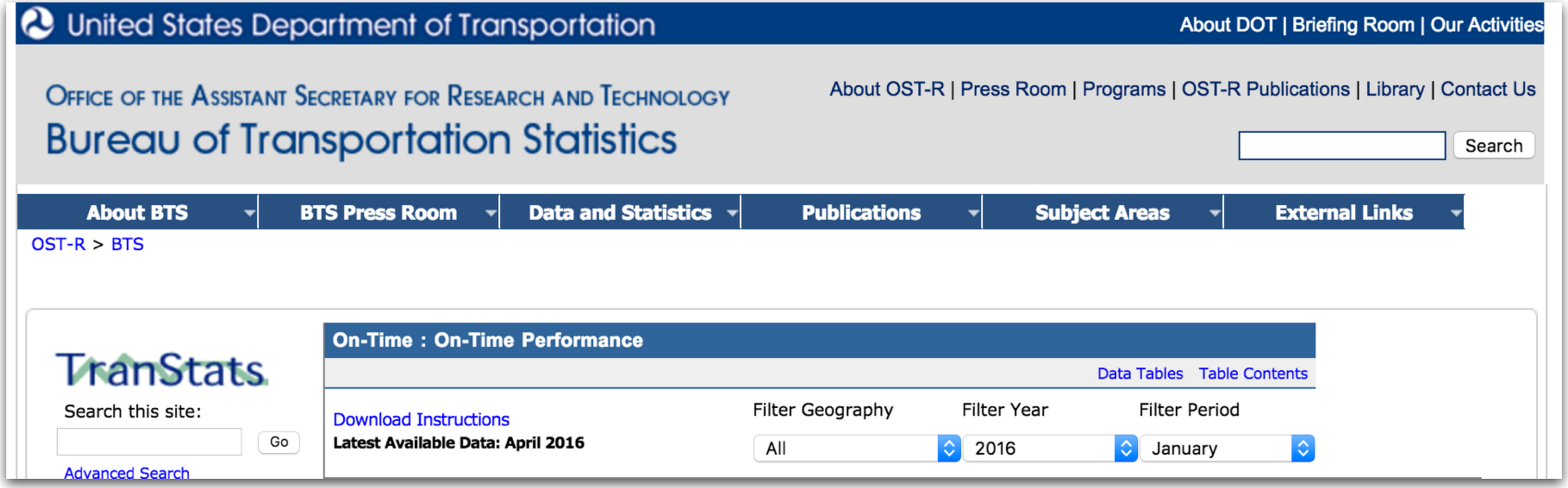
```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 1.6.1
      /_/

Using Python version 2.7.11 (default, Jan 22 2016 08:29:18)
SparkContext available as sc, HiveContext available as sqlContext.
>>>
```

Before we understand more about RDDs, let's see an example of data analysis using PySpark

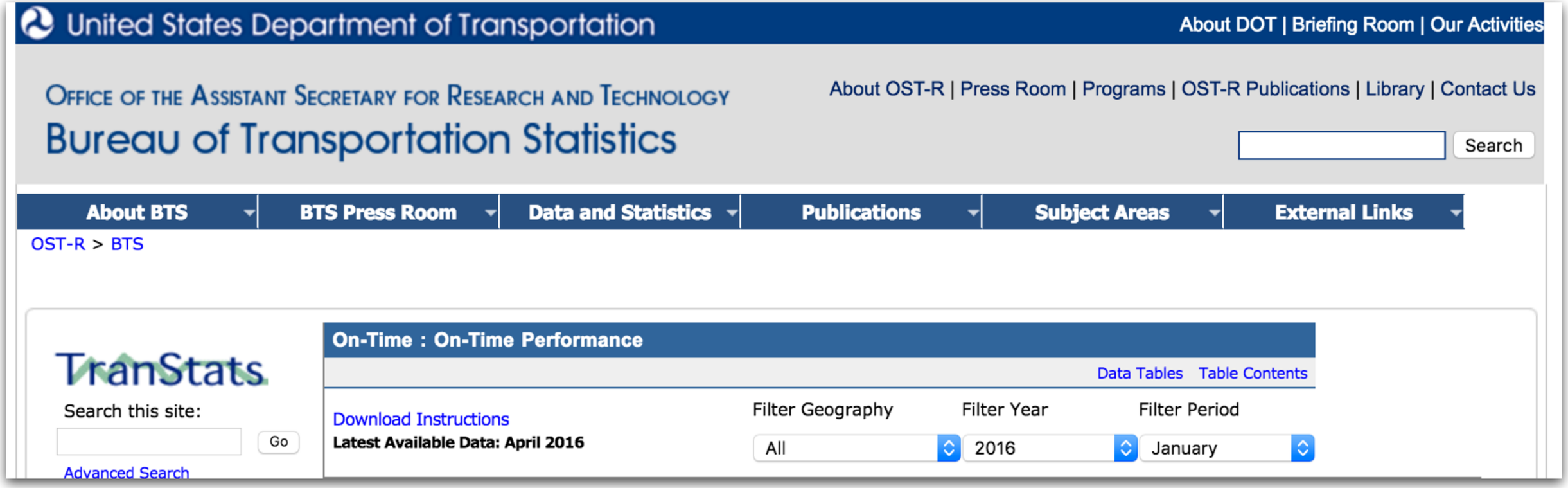
Exploring Airline delays data with PySpark

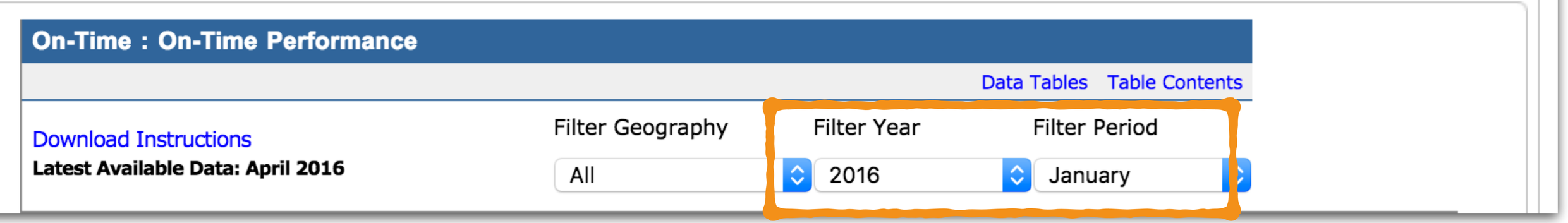
Part 1

The US department of transportation publishes Flight related information on their website



This data is free and available for anyone to analyze





From here we can get flight arrival and departure times, delays for all flights taking off in a certain period

There are 3 files

flights.csv	Flight id, airline, airport, departure, arrival , delay
airlines.csv	airline id, airline name
airports.csv	airport id, airport name

```
# Data location
airlinesPath="hdfs:///user/swethakolalapudi/flightDelayData/airlines.csv"
airportsPath="hdfs:///user/swethakolalapudi/flightDelayData/airports.csv"
flightsPath="hdfs:///user/swethakolalapudi/flightDelayData/flights.csv"
```

airlines.csv
airports.csv
flights.csv

Spark can read these files from the local file system or from HDFS

```
# Data location
airlinesPath="hdfs:///user/swethakolalapudi/flightDelayData/airlines.csv"
airportsPath="hdfs:///user/swethakolalapudi/flightDelayData/airports.csv"
flightsPath="hdfs:///user/swethakolalapudi/flightDelayData/flights.csv"
```

airlines.csv

airports.csv

flights.csv

If the files are in HDFS, you need to precede the file path with hdfs:///

```
# Data location
airlinesPath="hdfs:///user/swethakolalapudi/flightDelayData/airlines.csv"
airportsPath="hdfs:///user/swethakolalapudi/flightDelayData/airports.csv"
flightsPath="hdfs:///user/swethakolalapudi/flightDelayData/flights.csv"
```

airlines.csv

airports.csv

flights.csv

Now we can use the SparkContext initialized by PySpark to load the data

```
# Load one dataset
airlines=sc.textFile(airlinesPath)
```

```
# Load one dataset
airlines=sc.textFile(airlinesPath)
```

This is the SparkContext variable
i.e. the connection to Spark

```
# Load one dataset
airlines=sc.textFile(airlinesPath)
```

We are telling SparkContext to load data from a text file into memory

```
# Load one dataset
airlines=sc.textFile(airlinesPath)
```

The path to the text file, either on local or in HDFS

```
# Load one dataset
airlines=sc.textFile(airlinesPath)
```

airlines is a Resilient Distributed Dataset (RDD)

```
print airlines
```

```
hdfs:///user/swethakolalapudi/flightDelayData/airlines.csv MapPartitionsRDD[4] at textFile at NativeMethodAccessorImpl.java:-2
```

Resilient Distributed Dataset (RDD)

olalapudi/flightDelayData/airlines.csv MapPartitionsRDD[4] at textFile at NativeMethodAccessorImp

We have created our first RDD!

```
print airlines
```

```
hdfs:///user/swethakolalapudi/flightDelayData/airlines.csv MapPartitionsRDD[4] at textFile at NativeMethodAccessorImpl.java:-2
```

Let's get a quick glimpse of the data from the RDD

```
# Get the first line
airlines.first()
```

```
u'Code,Description'
```

A quick glimpse of the data

```
# Get the first line
airlines.first()
```

```
u'Code,Description'
```

```
# View a few lines
airlines.take(10)
```

```
[u'Code,Description',
 u'"19031","Mackey International Inc.: MAC"',
 u'"19032","Munz Northern Airlines Inc.: XY"',
 u'"19033","Cochise Airlines Inc.: COC"',
 u'"19034","Golden Gate Airlines Inc.: GSA"',
 u'"19035","Aeromech Inc.: RZZ"',
 u'"19036","Golden West Airlines Co.: GLW"',
 u'"19037","Puerto Rico Intl Airlines: PRN"',
 u'"19038","Air America Inc.: STZ"',
 u'"19039","Swift Aire Lines Inc.: SWT"']
```

```
# Let's view the entire dataset
airlines.collect()
```

```
[u'Code,Description',
 u'"19031","Mackey International Inc.: MAC"',
 u'"19032","Munz Northern Airlines Inc.: XY"',
 u'"19033","Cochise Airlines Inc.: COC"',
 u'"19034","Golden Gate Airlines Inc.: GSA"',
 u'"19035","Aeromech Inc.: RZZ"',
 u'"19036","Golden West Airlines Co.: GLW"',
 u'"19037","Puerto Rico Intl Airlines: PRN"',
 u'"19038","Air America Inc.: STZ"',
 u'"19039","Swift Aire Lines Inc.: SWT"',
 u'"19040","American Central Airlines: TSF"',
 u'"19041","Valdez Airlines: VEZ"',
 u'"19042","Southeast Alaska Airlines: WEB"',
 u'"19043","Altair Airlines Inc.: AAR"',
 u'"19044","Chitina Air Service: CHI"',
 u'"19045","Marco Island Airways Inc.: MRC"',
 u'"19046","Caribbean Air Services Inc.: OHZ"',
```

```
[u'Code,Description',
 u'"19031","Mackey International Inc.: MAC"',
 u'"19032","Munz Northern Airlines Inc.: XY"',
 u'"19033","Cochise Airlines Inc.: COC"',
 u'"19034","Golden Gate Airlines Inc.: GSA"',
 u'"19035","Aeromech Inc.: RZZ"',
 u'"19036","Golden West Airlines Co.: GLW"',
 u'"19037","Puerto Rico Intl Airlines: PRN"',
 u'"19038","Air America Inc.: STZ"',
 u'"19039","Swift Aire Lines Inc.: SWT"']
```

Each line consists of an airline code and description

Here's how we can filter out the header row

```
airlines.filter(lambda x:'Description' not in x)
```

```
airlines.filter(lambda x:'Description' not in x)
```

The filter operation takes a Boolean function

It applies that function on each line in the RDD and only keeps lines that return True

```
airlines.filter(lambda x:'Description' not in x)
```

The function can be from any Python module, a user defined function, or a lambda function

```
airlines.filter(lambda x:'Description' not in x)
```

Lambda functions are super cool

They allow programmers to define functions on the fly

```
airlines.filter(lambda x:'Description' not in x)
```

```
PythonRDD[4] at RDD at PythonRDD.scala:43
```

Something interesting happens when we try to execute this operation

```
airlines.filter(lambda x:'Description' not in x)
```

```
PythonRDD[4] at RDD at PythonRDD.scala:43
```

Unlike the results of first(), take() or collect()

```
airlines.filter(lambda x:'Description' not in x)
```

```
PythonRDD[4] at RDD at PythonRDD.scala:43
```

Unlike the results of first(), take() or collect()

```
# View a few lines
airlines.take(10)
```

```
[u'Code,Description',
 u'"19031","Mackey International Inc.: MAC"',
 u'"19032","Munz Northern Airlines Inc.: XY"',
 u'"19033","Cochise Airlines Inc.: COC"',
 u'"19034","Golden Gate Airlines Inc.: GSA"',
 u'"19035","Aeromech Inc.: RZZ"',
 u'"19036","Golden West Airlines Co.: GLW"',
 u'"19037","Puerto Rico Intl Airlines: PRN"',
 u'"19038","Air America Inc.: STZ"',
 u'"19039","Swift Aire Lines Inc.: SWT"']
```

```
airlines.filter(lambda x:'Description' not in x)
```

```
PythonRDD[4] at RDD at PythonRDD.scala:43
```

The filter operation returns another RDD

```
airlines.filter(lambda x:'Description' not in x)
```

```
PythonRDD[4] at RDD at PythonRDD.scala:43
```

This is the key to understanding RDDs and Spark's core functionality

```
filter
```

The filter operation tells Spark to change the dataset in some way

Transformation

```
take(10)
```

The take operation actually asks for a result - ten rows from the dataset

Action

Transformation

```
filter
```

Action

```
take(10)
```

All operations on RDDs are either Transformations or Actions

Transformation

```
filter
```

Action

```
take(10)
```

Spark will not immediately execute a transformation

Transformation

Action

```
filter
```

```
take(10)
```

It will just make a record of this transformation request by creating a new RDD

Transformation

Action

```
filter
```

```
take(10)
```

The user may define a chain of transformations on the dataset

Transformation

Action

```
filter
```

```
take(10)
```

In the airline example:

1. Filter the header
2. Split by comma
3. Convert airline code to integer

Transformation

Action

```
filter
```

```
take(10)
```

Spark will wait until a result is requested before executing any of these transformations

Transformation

```
filter
```

Action

```
take(10)
```

When created, an RDD just holds metadata

1. A transformation
2. It's parent RDD

AirlineFiltered RDD

Transformation

filter

Action

take(10)

AirlineFiltered RDD

filter

airlines

This way, every RDD knows where it came from

ie. RDDs know their lineage

Transformation

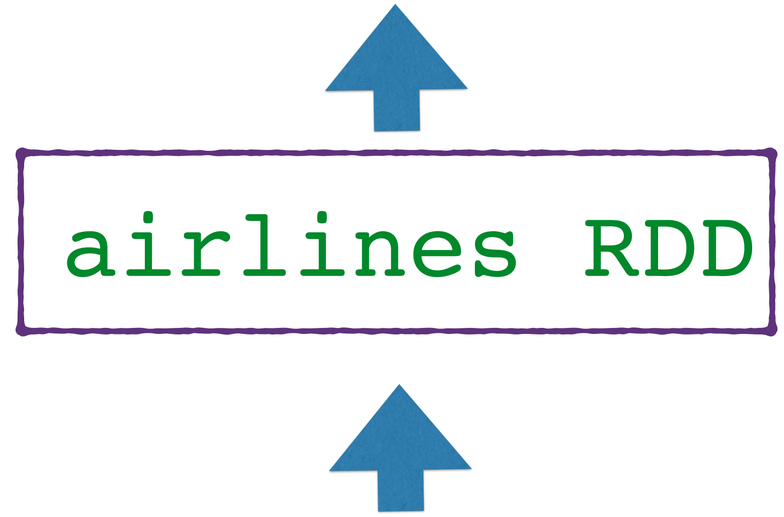
Action

```
take(10)
```

This lineage can be traced back to the original file that held the data

Recall: Reading a file created an RDD

AirlineFiltered RDD



Transformation

Action

filter

take(10)

File read is also like a transformation

RDDs are materialized only when a result is requested

Transformation

filter

Action

```
take(10)
```

A result is requested

using an action

Transformation

Action

```
filter
```

```
take(10)
```

Ex: the first 10 rows
a count
a sum
the entire dataset

Transformation

```
filter
```

Action

```
take(10)
```

To summarize, data is processed only when the user requests a result

Transformation

```
filter
```

Action

```
take(10)
```

This is known as

Lazy Evaluation

Lazy Evaluation

This is what allows Spark to perform **fast computations** on large amounts of data

Lazy Evaluation

Spark will keep a record of the series of transformations requested by the user

Lazy Evaluation

When called upon to execute,
Spark takes these operations
and groups them in
an efficient way

More on this later...

Let's now formally understand RDDs