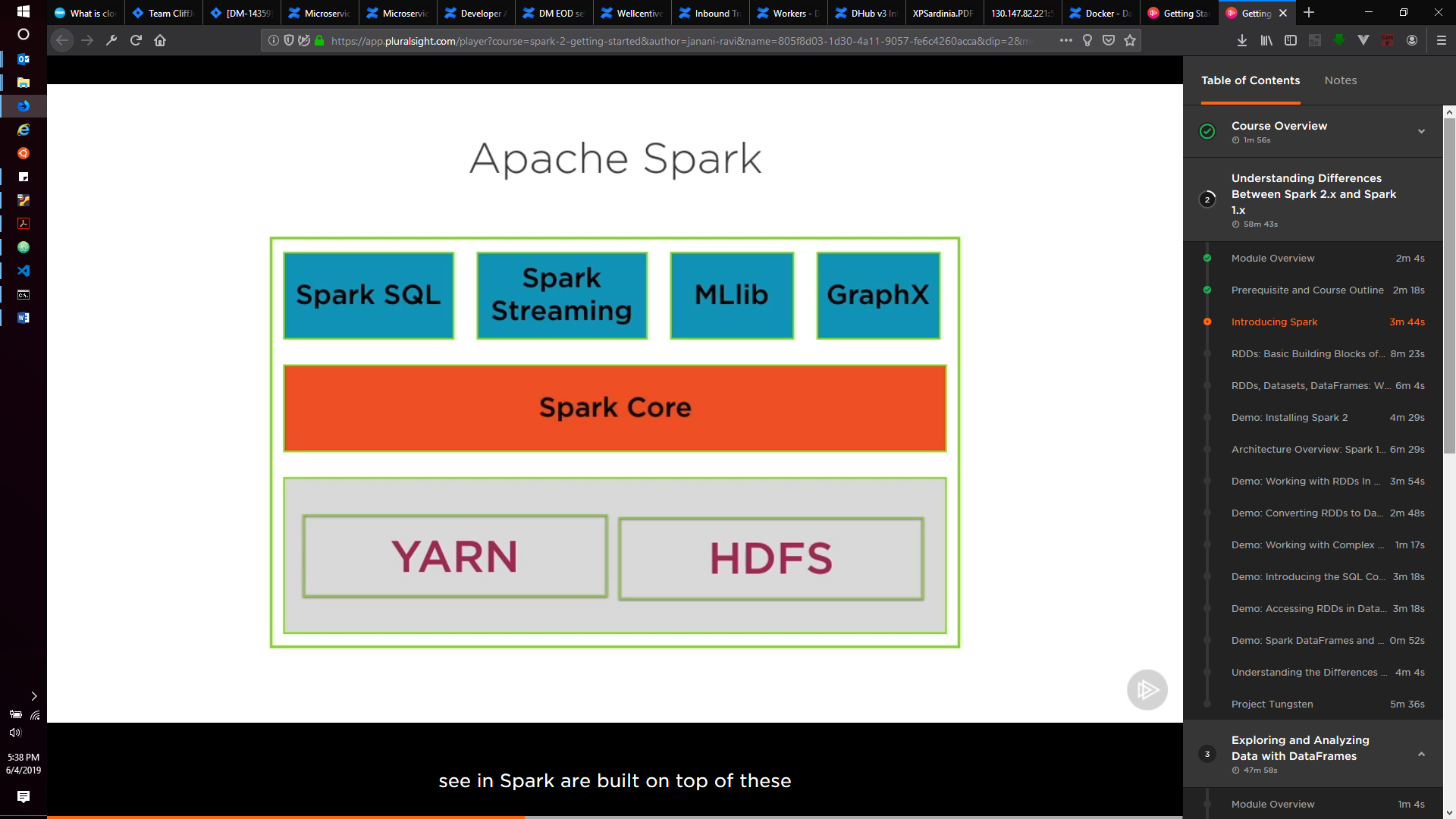
# Apache Spark

Apache Spark is a distributed computing engine. It can also be described as a unified analytics engine for largescale data processing. It is a part of Hadoop.



Spark can act as a replacement to map-reduce layer in a big data ecosystem. Spark API creates the job binaries that are fed into YARN for execution and then persisted in HDFS.

Spark has many libraries running on top of spark core. The key power of spark is the abstraction that it provides for distributed processing.

**What does spark offer?**

* Real-time as well as batch processing is possible
* Interactive REPL environment (Read Evaluate Print Loop), this can be used to run and prototype spark jobs.
* Support for many languages.

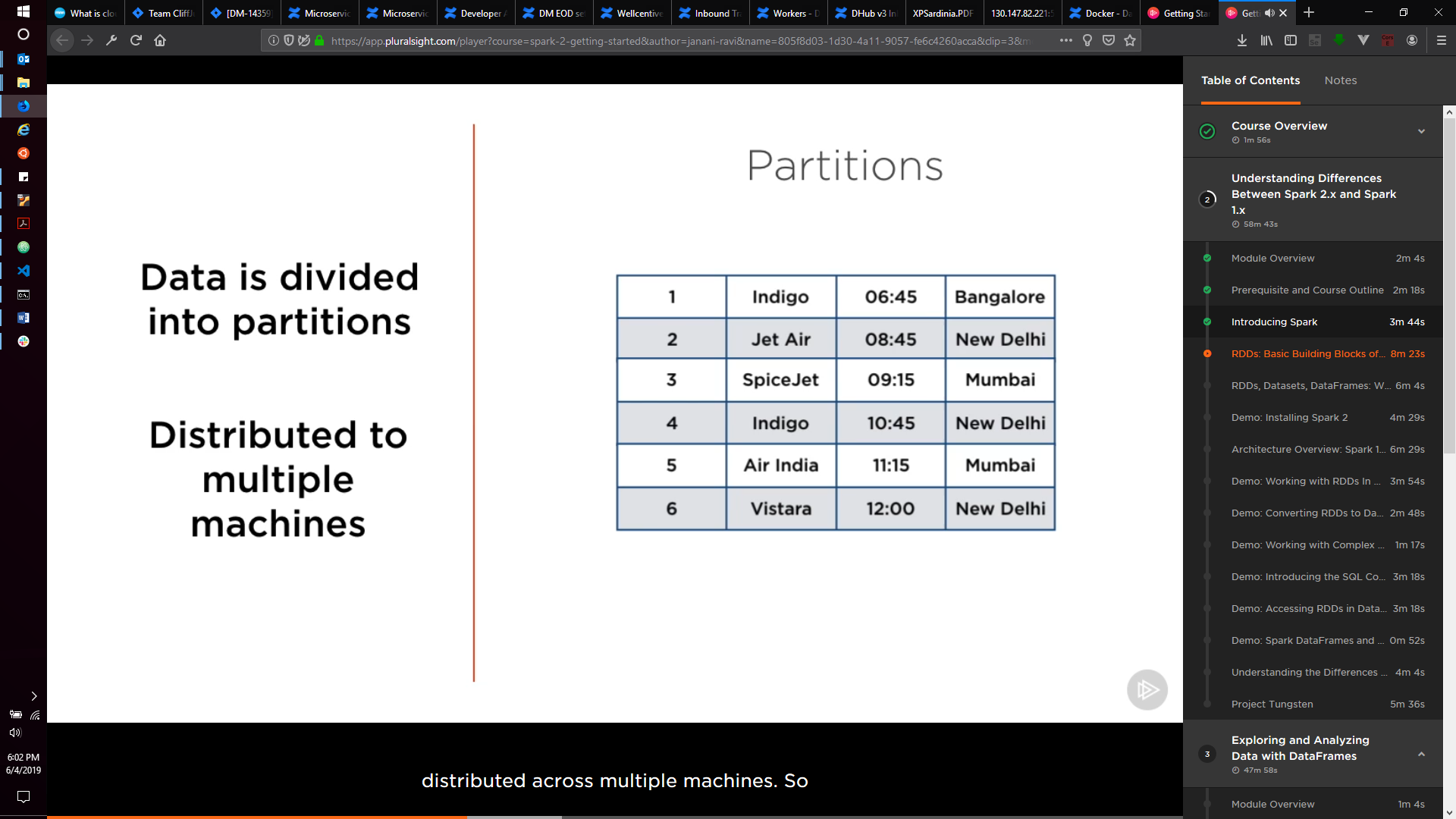
**RDDs (Resilient Distributed Datasets)**

All operations in spark are performed on in-memory objects called RDDs. There is no back and forth communication with the file system, this is the most significant difference between Hadoop and Spark. An RDD is a collection of entities (Rows, records, consisting of integers, strings, etc.). RDD is like a collection in Java. It can be assigned to variables and methods can be invoked on it. Methods on these variables can return values or make create transformed RDDs.

**Characteristics of RDDs that make them different from collections:**

* Partitioned: They can split across data nodes in a spark cluster. This partitioning is **completely abstracted** from the developer.
* Immutable: Once they are created, they cannot be changed.
* Resilient: Can be recreated even if a node crashes. RDDs can be reconstructed from their raw form.

**Partitions:**



Data might look like it is together, but it distributed among multiple in-memory nodes. All these nodes make up the RDD. This enables parallel processing on the data.

**Immutability:**

Only permitted operations on RDDs are Transformation and Actions:



**Transformations:**

A Transformation is any task requested to the RDD which generates a transformed RDD. Loading data into an RDD and define a chain of transformations that generates new RDDs. For example: Load Data set -> Select Row 25 -> Sort Values. ON APPLYING THE TRANSFORMATION, THEY ARE NOT ACTUALLY EXECUTED. The actual execution of a transformation is done when a result is requested, which is an ACTION.

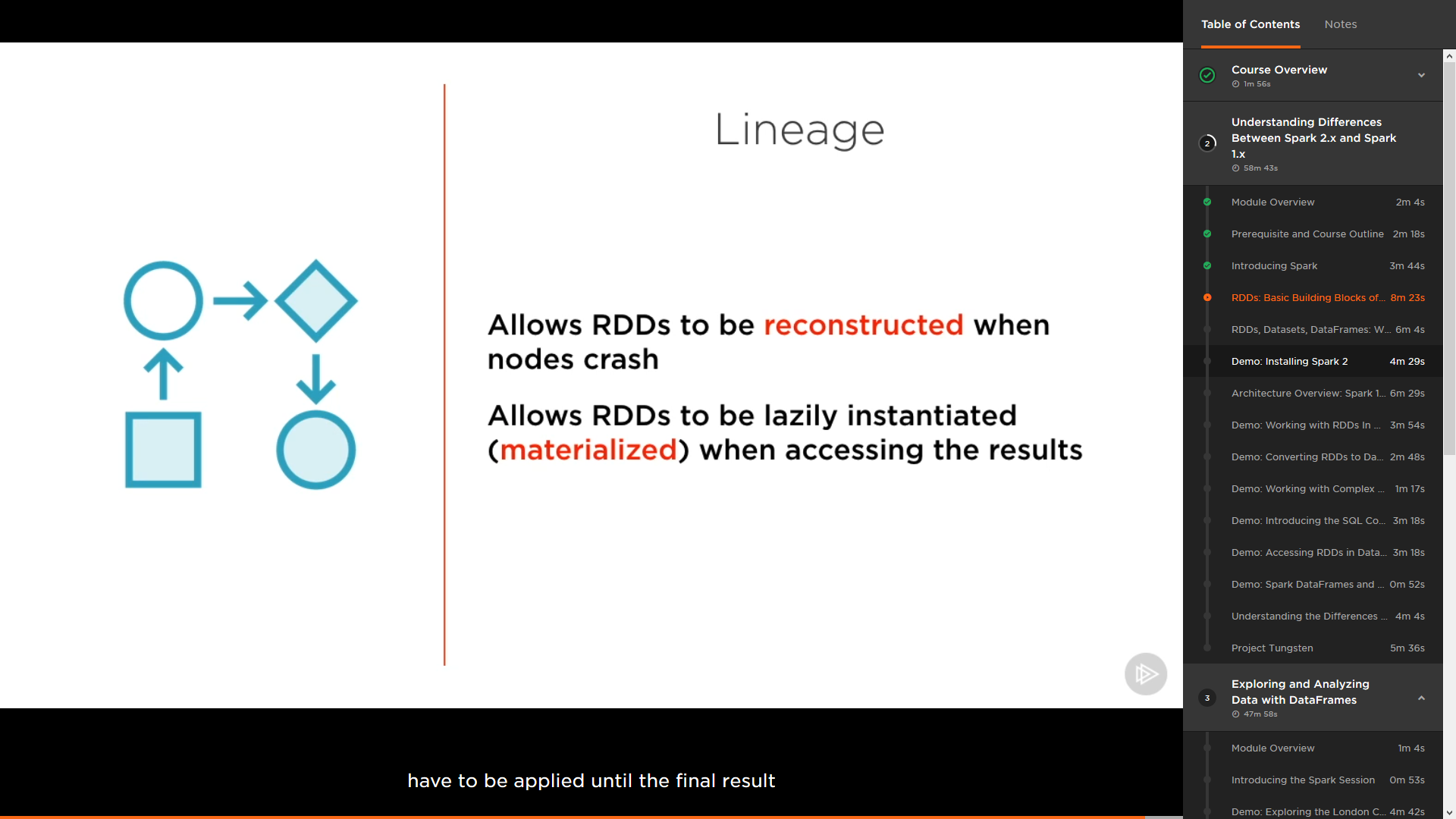
**Actions:**

Some stats/info that needs to come out of the cluster. Example: Give first 10 rows. Give max of all these values.

RDDs perform lazy evaluation, i.e. evaluating a result only when it is specifically requested. All the transformation on the data are setup as a directed acyclic graph. The nodes on the graphs are the actual operations performed on the data. When an action is requested, Spark can traverse this graph and efficiently compute the best possible way to process the data, execute operations in parallel and return the results.

**Resilience:**

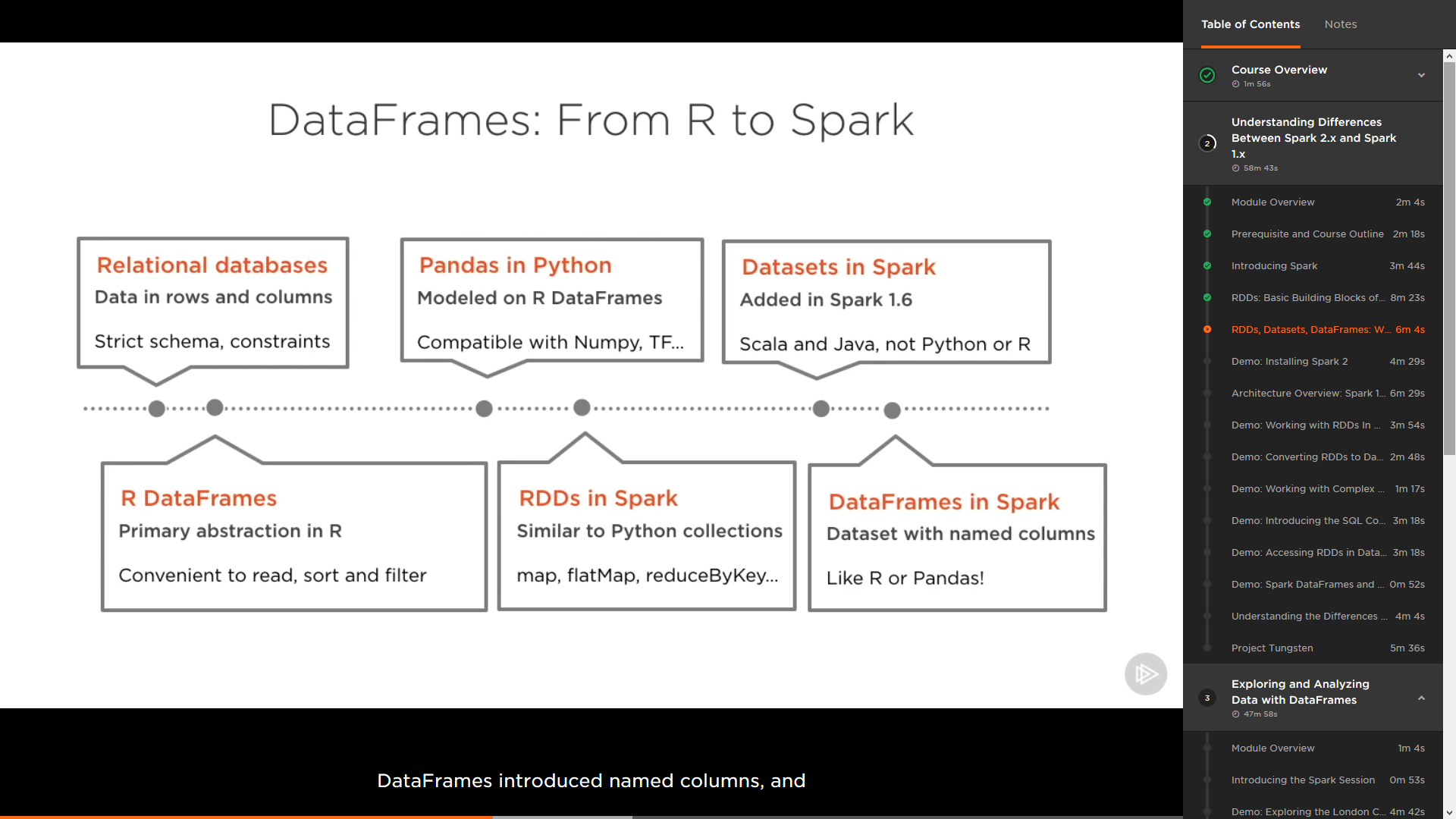
An RDD can be created by only two methods: Either reading a file, or applying a transformation to the existing RDD. Both of these actions can be tracked and it can be found where the RDD came from. All the transformations happening on RDD are tracked in meta data. This is called the **lineage of an RDD**. Every RDD in Spark is aware of its own lineage, back to the original source of data. This lineage allows reconstruction of RDD if a node crashes. Lineage also allows lazy instantiation and materialization upon accessing the result.



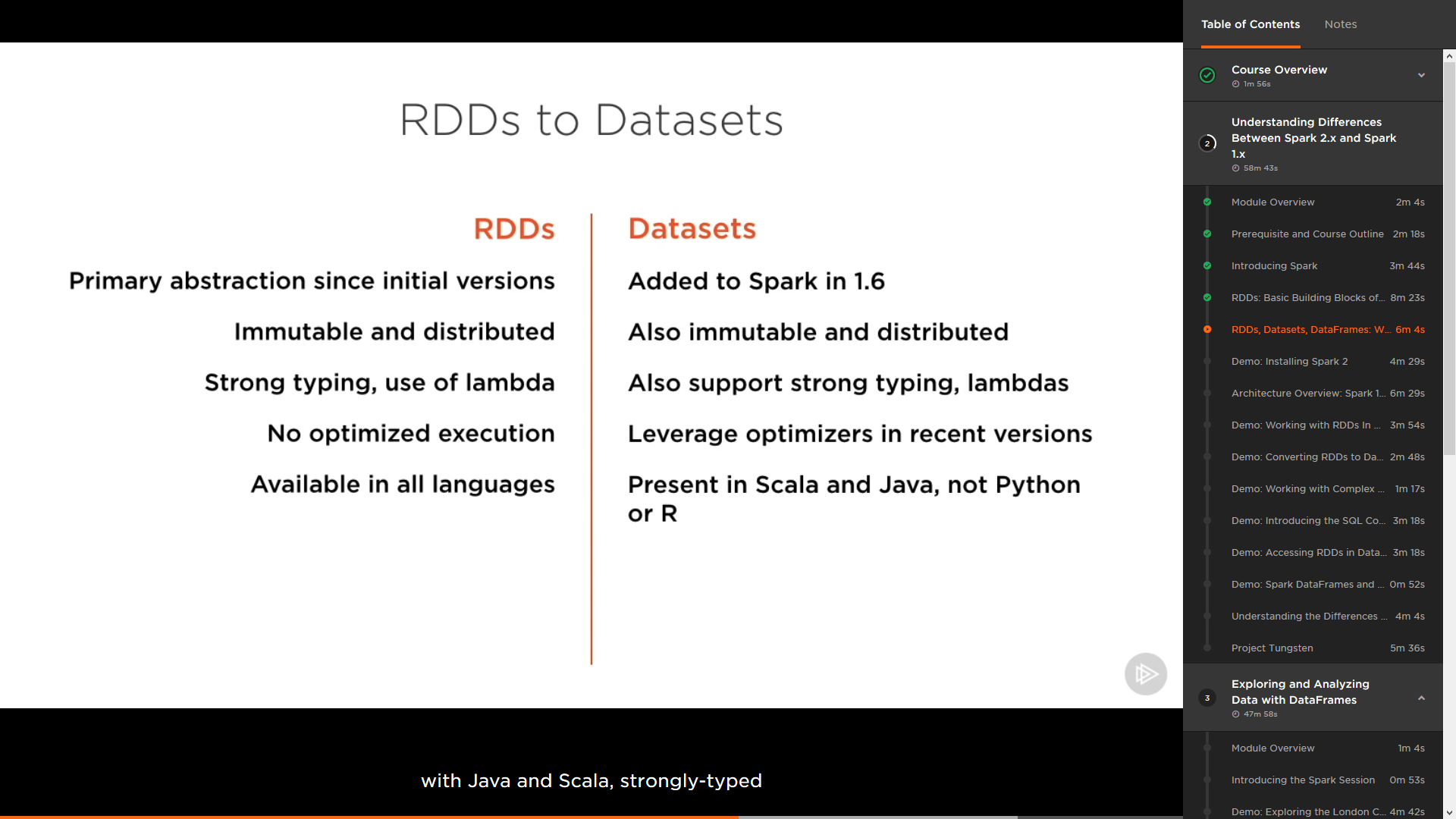
RDD were to be worked on directly in Spark 1, but in Spark 2 these RDDs are packages as Data Frames or Data Sets (Java/Scala).

**DataFrame:**

A data frame is like a table of rows and columns. Entries in a row can be values or collections (vectors).

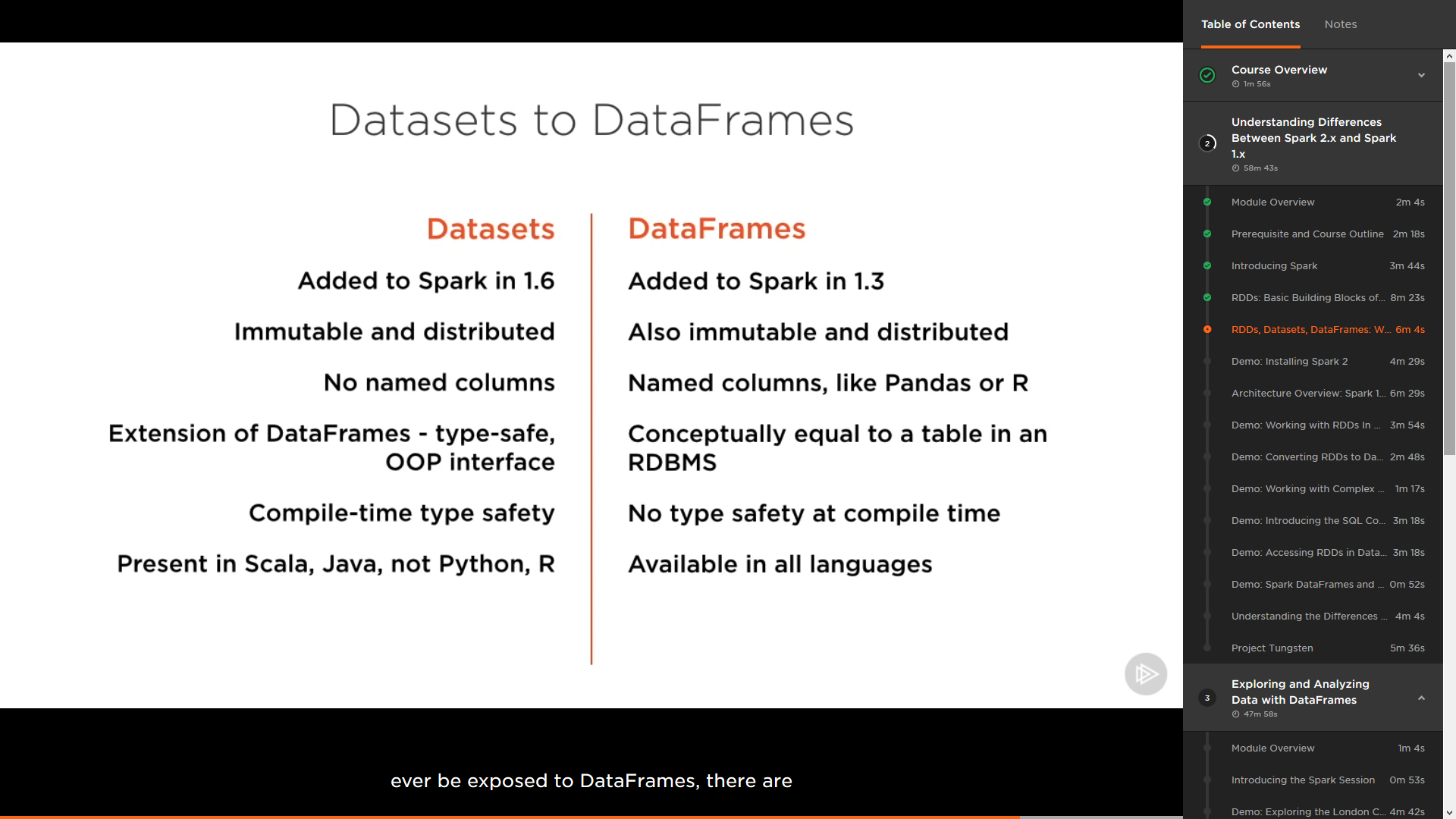


RDDs vs Datasets

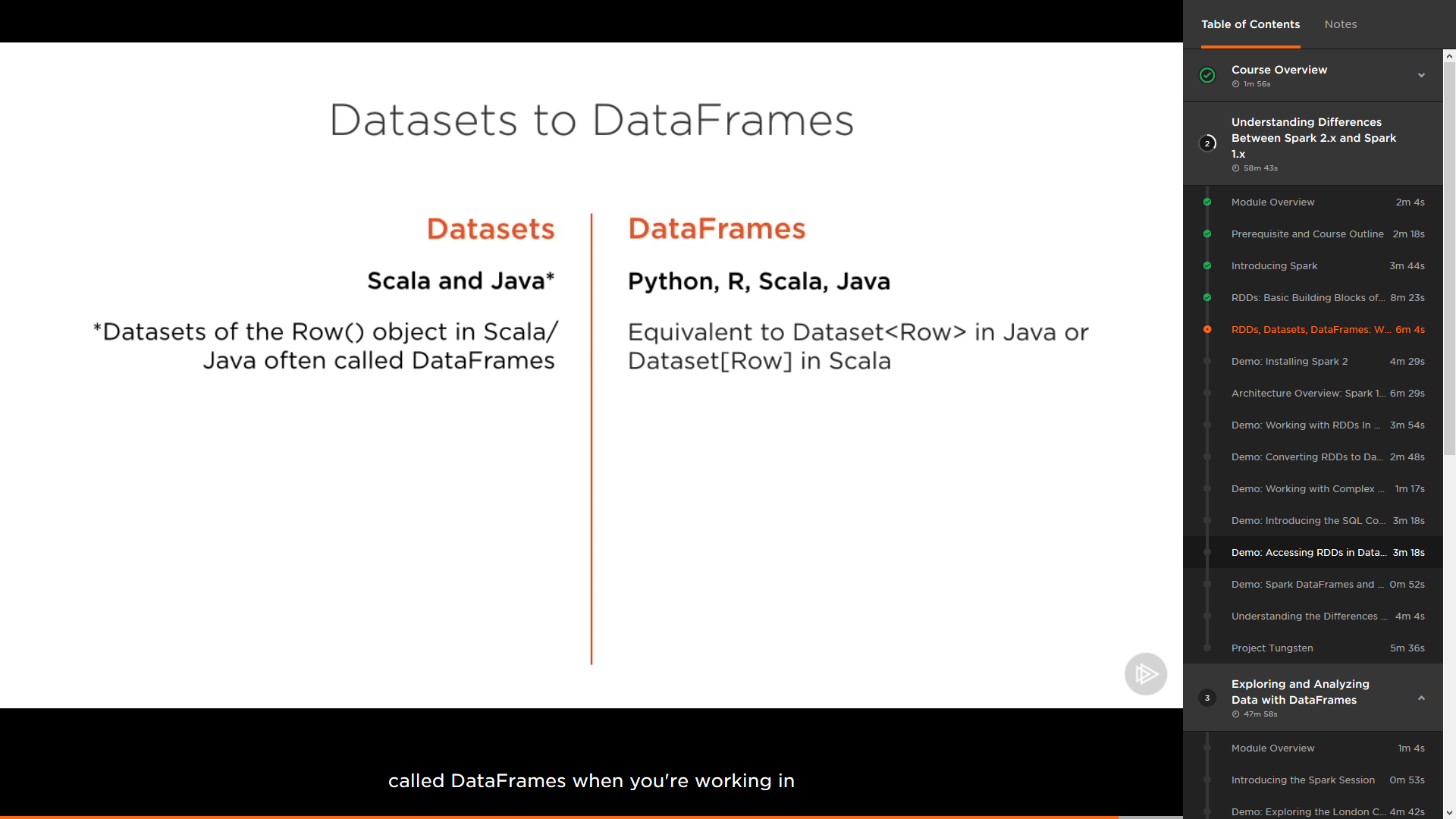


In addition, typically while working with the data, if very low-level granular control is required over the data, RDDs should be used; otherwise, if rich semantics and high-level abstraction is required, DataSets/DataFrames should be used.

DataFrames vs DataSets



APIs of DataSets and DataFrames have been **merged**. DataSets are just the strongly-types versions of DataFrames.

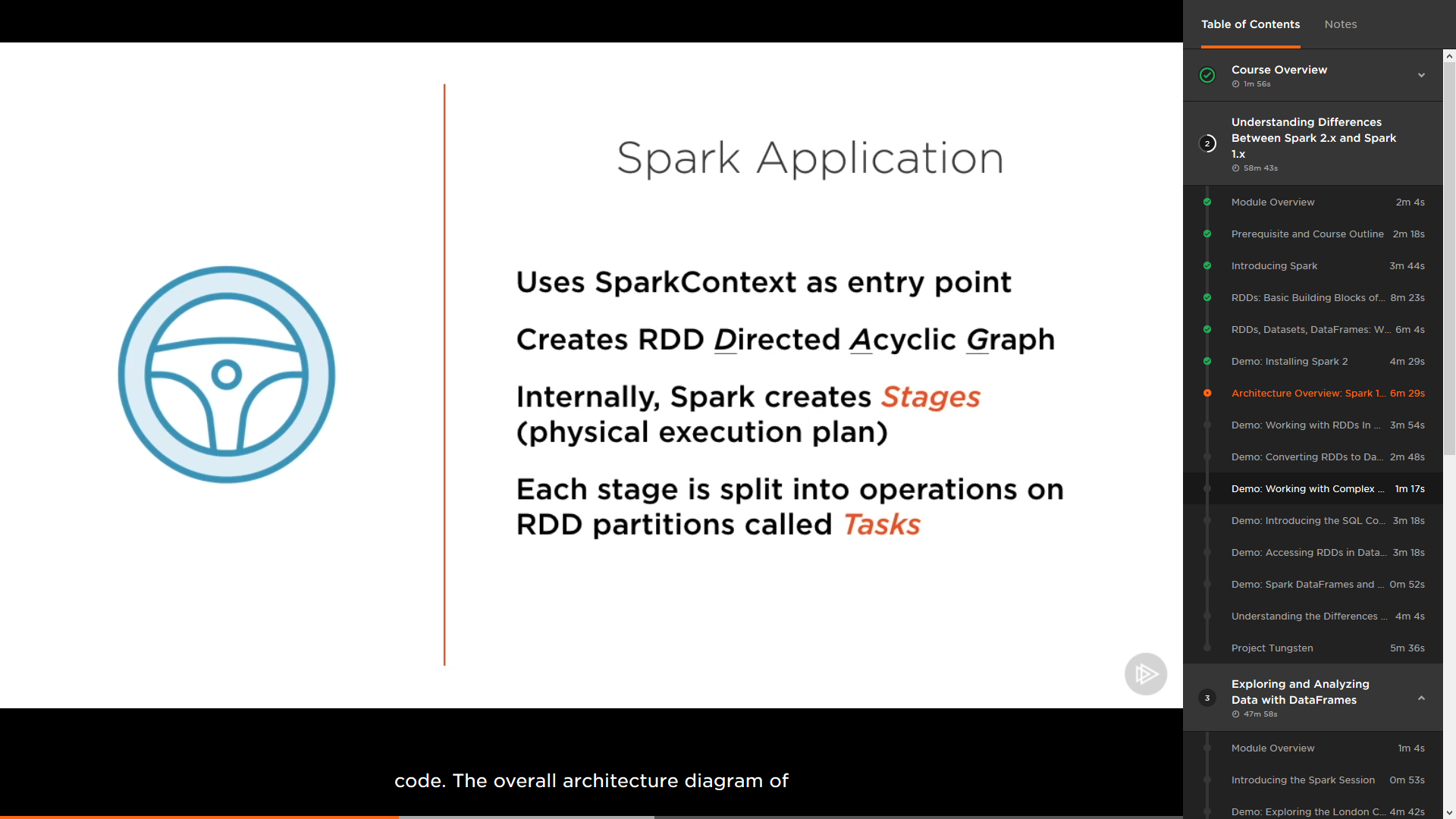


# Architecture of Apache Spark

Spark cluster is setup in a master/worker configuration. Master coordinates all the processes that run on worker nodes. The master node runs a driver program, which is a separate JVM process; this driver is responsible for launching tasks on individual nodes. These tasks are what operate on subsets of RDDs that are present on that node. This driver code also contains the SparkContext. This context is a gateway/bridge to the Spark environment and all that it has to offer from the running application. Driver also plays host to different services:

* SparkEnv
* DAGScheduler
* TaskScheduler
* SparkUI
* …

The spark application uses the SparkContext as entry point.



Spark 1.x (Volcano Iterator Model) -> Spark 2.x Tungsten Engine.

What Tungsten Engine offers?

* Eliminate virtual function calls
* Store data in registers and not RAM or Cache
* Compiler loop unrolling, pipelining
* 10-20x Volcano Iterator Model

What is SparkContext?

* Entry point to the Spark Application
* Create RDDs, Broadcast Variables, Accumulators, and even run the jobs.
* SparkSession encapsulates the SparkContext and gives additional capabilities like using any context (SQL Context or Hive Context), etc.

Execution of spark application happens over multiple compute nodes. These nodes are called workers. When an application starts, worker starts executers (Cluster manager worker). These executers then run the tasks, which are actually the basic units of execution. These tasks are logically grouped into stages and every stage is a physical unit of execution.

