Big Data & Hadoop: Google Cloud & Apache Spark

Prateek Chandra Jha, Tanmey Rawal, Ashray Anand, Saikrishna Ranganathan

Chennai Mathematical Institute

09-03-2020, 9:10 AM - 9:55 AM

Introduction: Cloud Business & Google Cloud

	2018	2019	2020	2021	2022
Cloud Business Process Services (BPaaS)	41.7	43.7	46.9	50.2	53.8
Cloud Application Infrastructure Services (PaaS)	26.4	32.2	39.7	48.3	58.0
Cloud Application Services (SaaS)	85.7	99.5	116.0	133.0	151.1
Cloud Management and Security Services	10.5	12.0	13.8	15.7	17.6
Cloud System Infrastructure Services (IaaS)	32.4	40.3	50.0	61.3	74.1
Total Market	**196.7**	**227.8**	**266.4**	**308.5**	**354.6**

BPaaS = business process as a service; IaaS = infrastructure as a service; PaaS = platform as a service; SaaS = software as a service

Figure 1: Worldwide Public Cloud Service Revenue Forecast (Billions of U.S. Dollars), Source: Gartner (2019)

Big Data Challenges and Where Does this Presentation Fit?

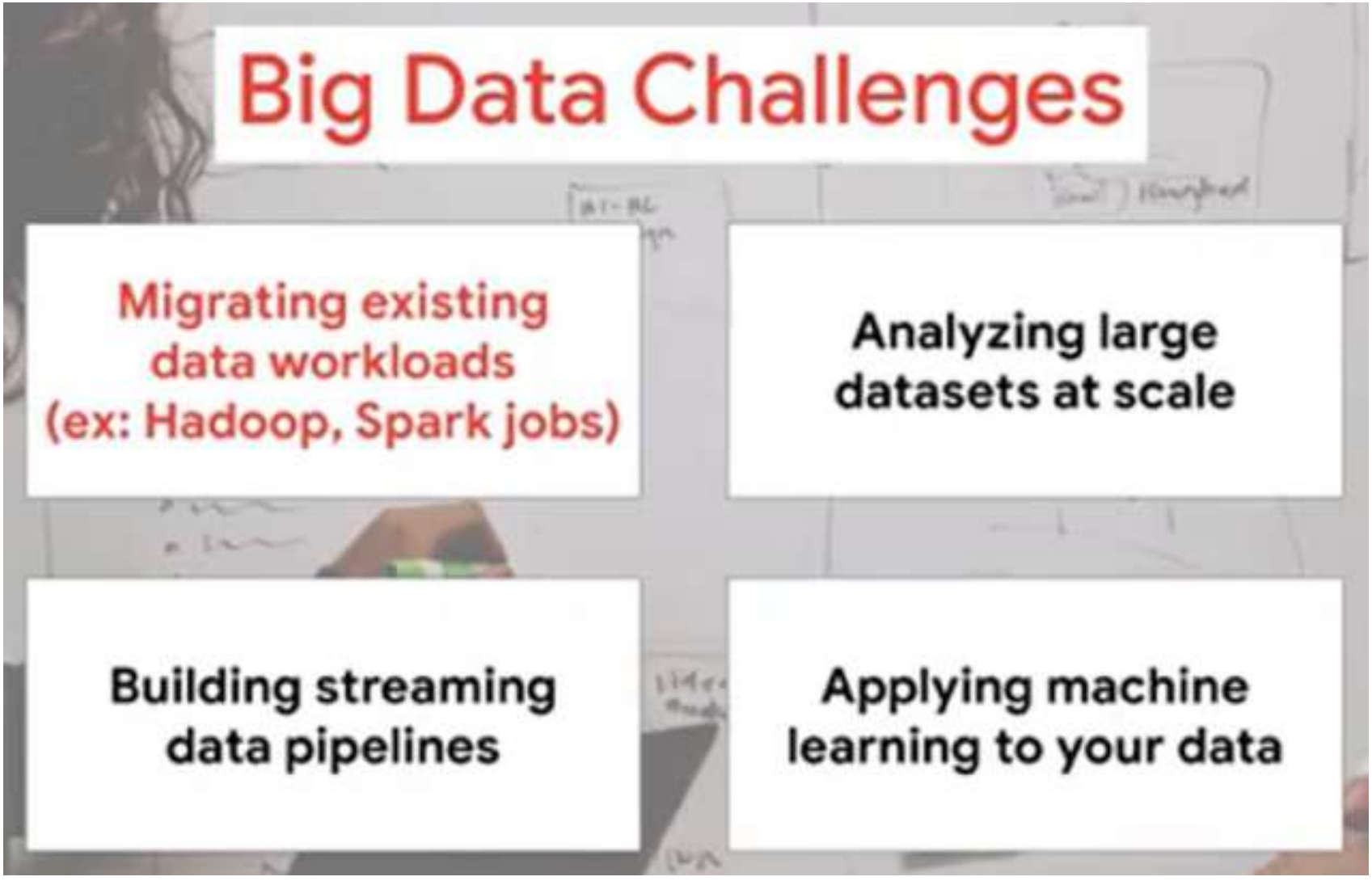

Figure 2: This presentation will serve you with two demos: One using Google Cloud Vision and another using PySpark

Google Cloud Platform Infrastructure

- Cloud Infrastructure originally built to power Google's own applications
- But now it's Public! : Compute, Storage, Networking, Security
- Big Data and ML Products are built over that infrastructure for companies to work with
- 7 Cloud Products with over a Billion Users: Google Search, Android, Maps, Gmail, Playstore, Chrome, YouTube
- End-users served by Google Cloud Customers are even more than that: Home Depot, Spotify, Twitter, New York Times, Colgate-Palmolive etc.

Layers of Google Cloud

Figure 3: We'll focus on the top layer as that's the abstraction built by Google for scalability of infrastructure required for processing large datasets and running ML/AI algorithms over them

What's the Scale of Google Cloud Compute?

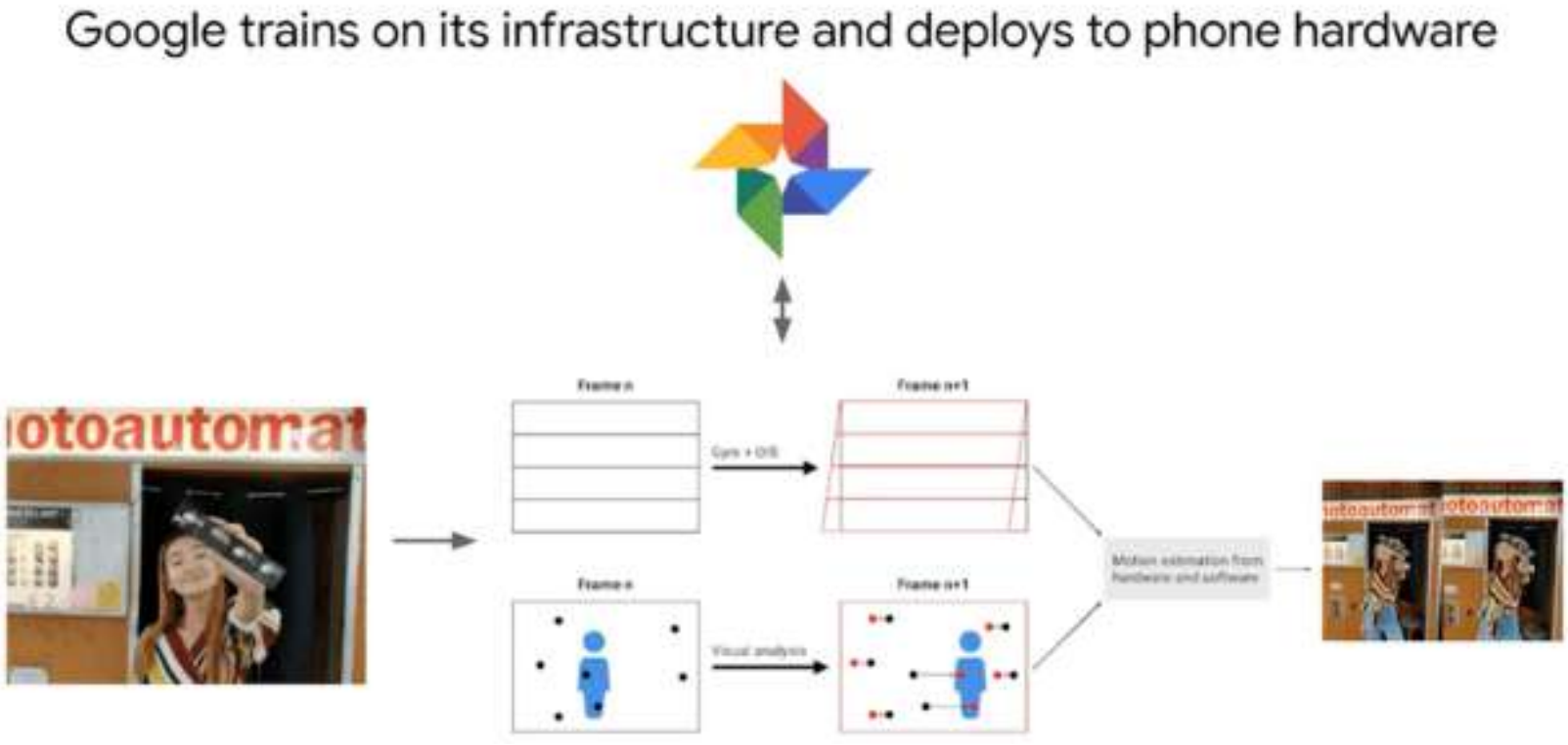

Figure 4: Computation Power required for ML/AI workloads is immense. Google trains production ML models over vast data centers and then deploys smaller trained models on our phones/products

Coming Up: Google Cloud Vision Demo

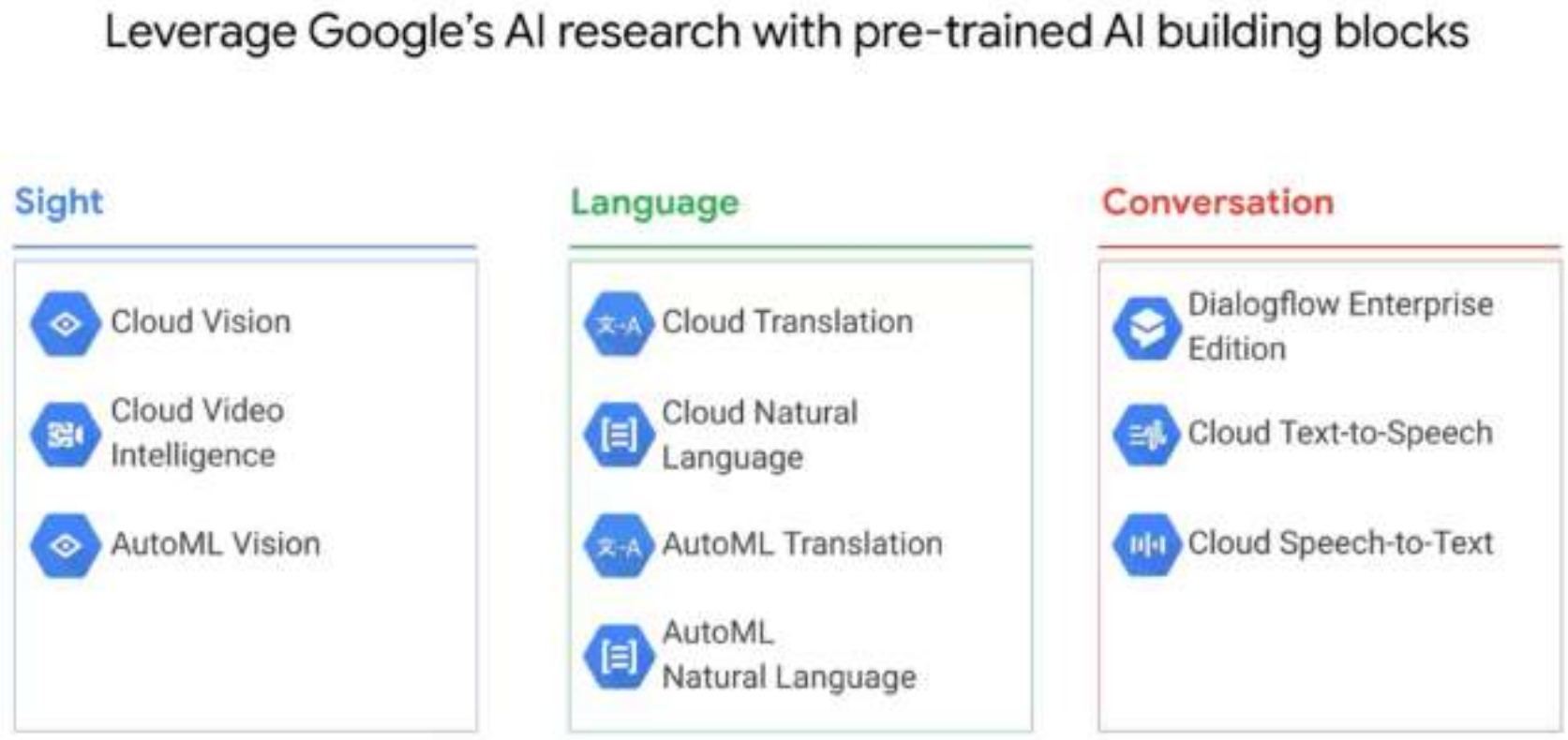

Figure 5: One can use Google's many APIs instead of building and training one's own custom ML/AI model. There are other fully trained models for language and videos. Running that many sophisticated ML models on large structured and unstructured datasets for Google's own products, required a massive investment in computing power. Wired called it: "The Mother of All Clouds".
Source: `https://cloud.google.com/video-intelligence`

How is Cloud Computing different from Desktop Computing?

- **Virtual Machine Instance:** Google's Infra & Compute Engine delivers configurable virtual machines running in Google's data centers with access to high-performance networking infrastructure and block storage
- 1 vCPU to 64vCPUs with about 250GB RAM available for free computing ($300 one-year free signups available just as we did)
- Customizable to more or less CPUs or RAM requirements
- Different OS available with customizable disk sizes
- Access to Cloud APIs so that I can write to cloud storage via VMs
- These VMs are technically called **"bare bones virtual machines"**[1] as they don't even have basic services like git installed

[1] Once you do your data analysis on the VMs, you generally delete those machines and copy the transformed data and insights into the cloud storage. These iterations are practically impossible to do by desktop computing.

Compute and Storage are Independent and Separate

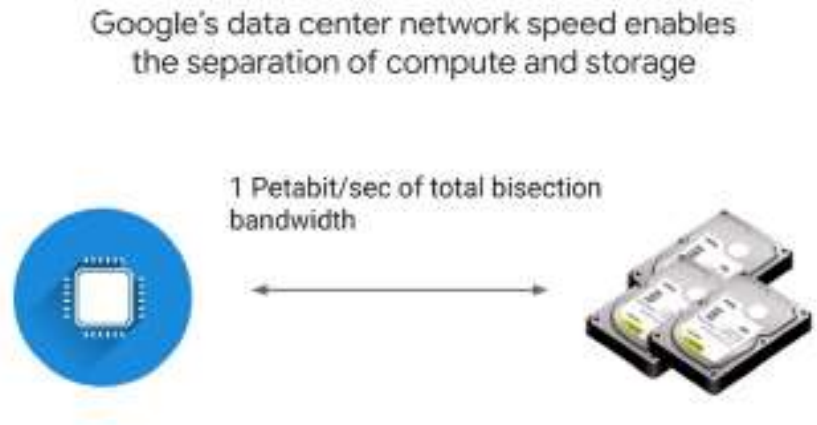

Figure 6: Disks attached to the compute VM instances as the limit of how much data you can process and store? Even a single cluster of machines with their own dedicated storage useless. Why? Perform computations on data located elsewhere like many distributed servers. Google's Jupiter Network can deliver enough bandwidth to allow 100,000 machines to communicate amongst each other. Locality within the cluster is not important. If every machine can talk to every other machine at 10 Gbps, racks don't matter for data analytics and machine learning.

What are Edge Points of Presence in the Google Cloud Network?

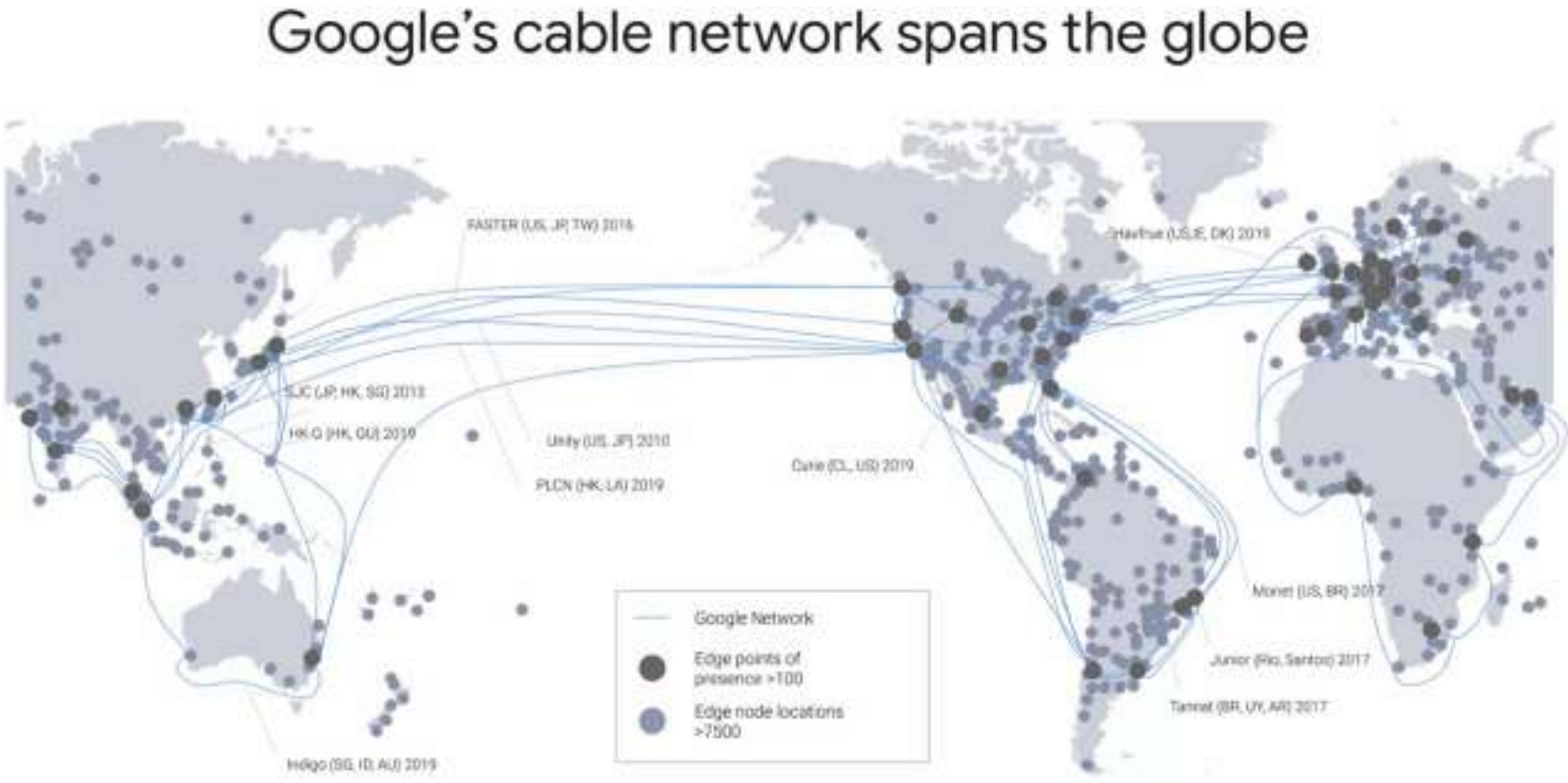

Figure 7: Google's Network, interconnects with the public Internet at more than 90 internet exchanges and more than 100 points of presence worldwide. Google responds to the user's request from an Edge network location that will provide the lowest delay or latency. Google's Edge caching network places content close toend-users to minimize latency.

BigQuery: Google Cloud's Petabyte Scale Analytics Data Warehouse + Visualization map of where to store your data in GCP

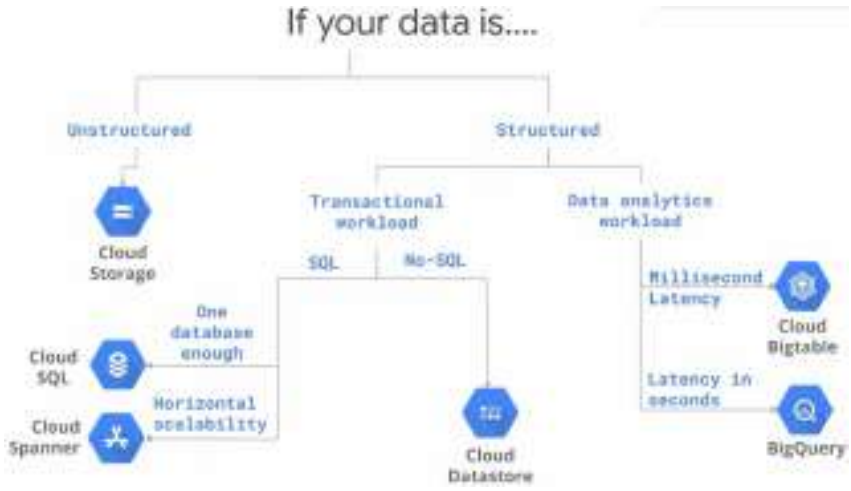

Figure 8: If your data is unstructured, like images or audio, use Cloud Storage. Cloud SQL generally plateaus out at a few gigabytes. Transactional database that is horizontally scalable, multiple databases, lots of data: Cloud Spanner; Structured Data and Analytics: BigQuery/Bigtable; Real-Time high throughput applications: Bigtable; Analytics over petabyte-scale datasets: BigQuery; Structured Data & Transactions: Cloud SQL or Cloud Datastore (single key search NoSQL); But if one DB is enough: Always CloudSQL.

A General BigQuery Data Warehouse Architecture: Cloud Data Pipelines

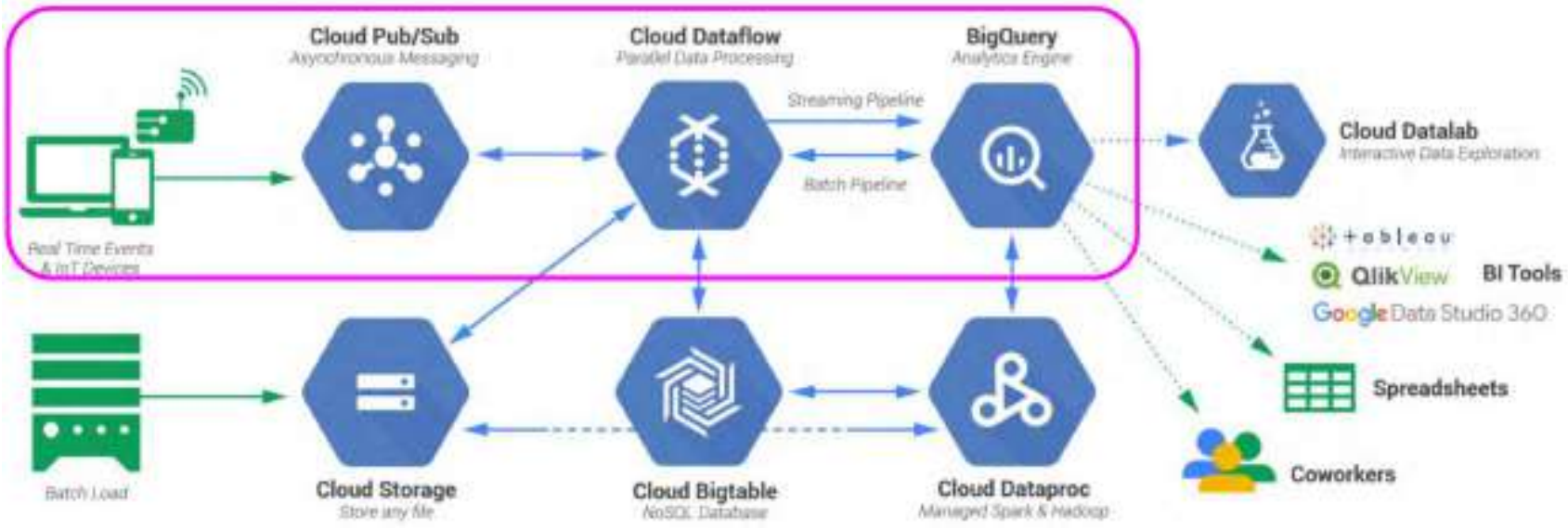

Figure 9: BigQuery is designed to be an easy-to-use data warehouse. We can focus on writing SQL statements on small or large datasets without worrying about infrastructure. Use BigQuery to store, transform, and then feed those large datasets directly into your ML models. This is a huge leap over training ML models on just a few small samples of your data locally on your laptop or desktop. You can now train on all the data that you have available. BigQuery is a common sink or staging area for your data analytics workloads.

It can take from days to a few Months to create a ML Model. But Now you can build ML Models in minutes with SQL. Yes, you heard it right!

- **Label** = alias a column as 'label' or specify column in OPTIONS using input_label_cols
- **Feature** = passed through to the model as part of your SQL SELECT statement
  `SELECT * FROM ML.FEATURE_INFO(MODEL `mydataset.mymodel`)`
- **Model** = an object created in BigQuery that resides in your BigQuery dataset
- **Model Types** = Linear Regression, Logistic Regression
  ```
  CREATE OR REPLACE MODEL <dataset>.<name>
  OPTIONS(model_type='<type>') AS
  <training dataset>
  ```
- **Training Progress** = `SELECT * FROM ML.TRAINING_INFO(MODEL `mydataset.mymodel`)`
- **Inspect Weights** = `SELECT * FROM ML.WEIGHTS(MODEL `mydataset.mymodel`, (<query>))`
- **Evaluation** = `SELECT * FROM ML.EVALUATE(MODEL `mydataset.mymodel`)`
- **Prediction** = `SELECT * FROM ML.PREDICT(MODEL `mydataset.mymodel`, (<query>))`

Figure 10: Step one, you create the model with just a SQL statement. Step number two, write a SQL prediction query and invoke ML.predict. Step number three, profit, that's it. You've got a model and then you can review the results. Well, in reality, there are more than three steps. You have to evaluate the model, but the main point is that, if you know basic SQL, you can now do machine learning which is pretty cool. BigQuery ML was designed with simplicity in mind.

Google Cloud Big Data Universe!

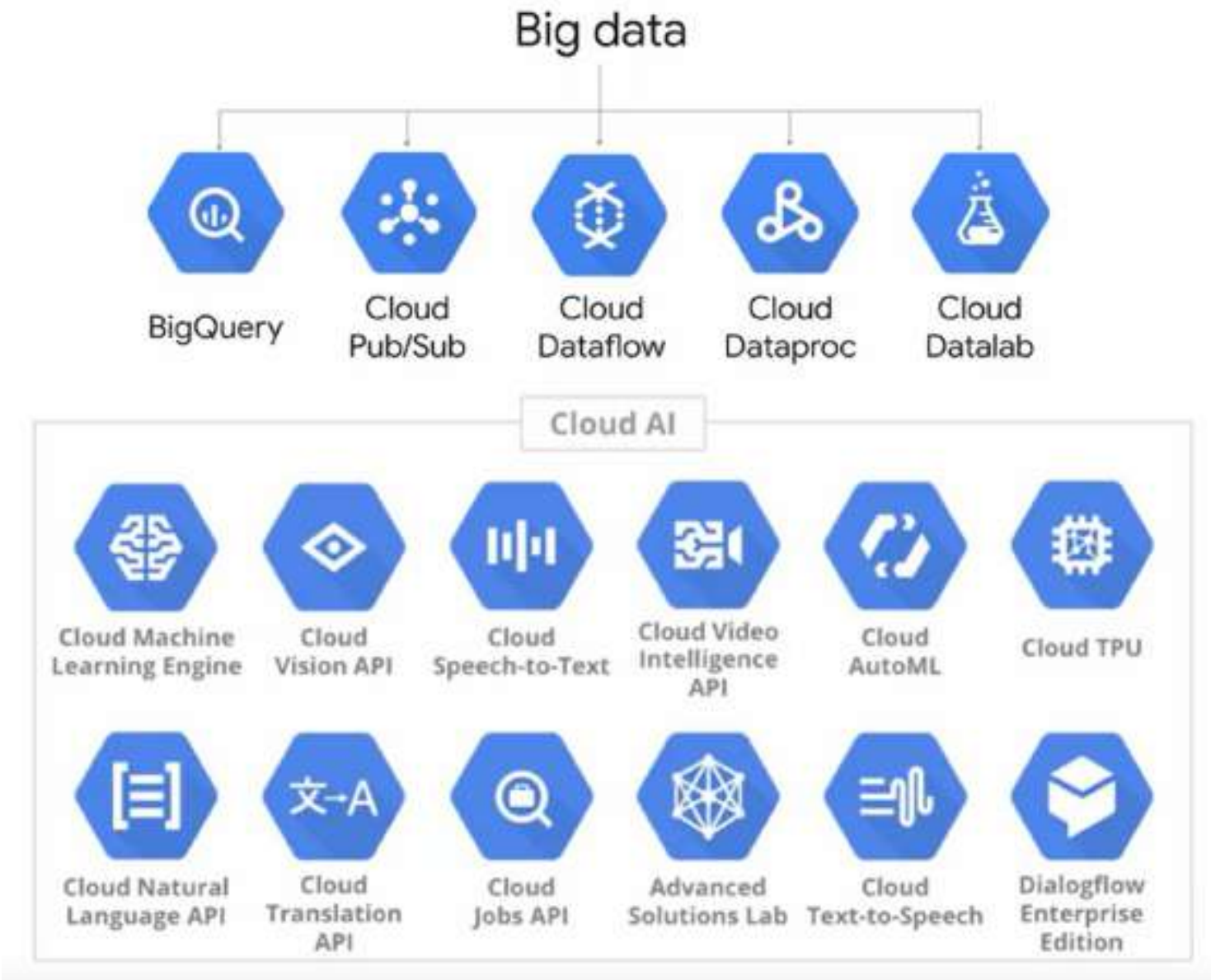

Figure 11: We use Cloud Vision API today to analyze a few photos.

Invention of New Data Processing Methods as Google Grew

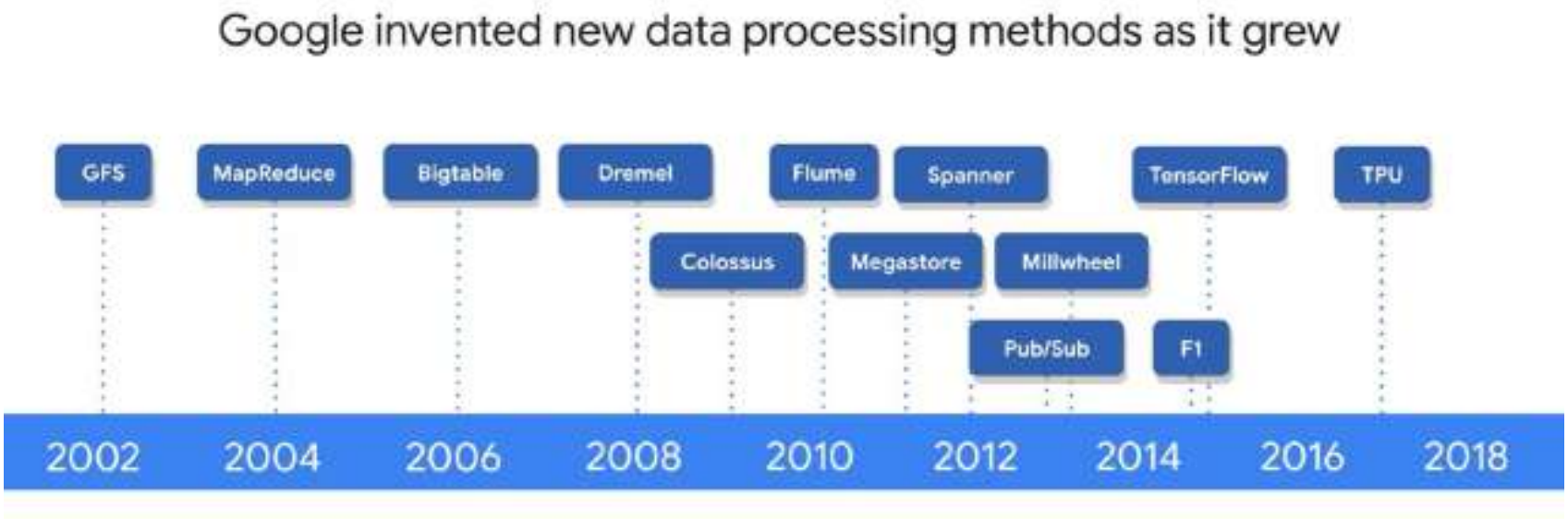

Figure 12: One issue with MapReduce is that developers have to write code to manage all of the infrastructure of commodity servers. Developers couldn't just focus on their application logic. So between 2008 and 2010, Google started to move away from MapReduce to process and query large data sets, and instead they started moving towards new tools. Tools like Dremel.

Open Source Ecosystem for Big Data: What have we seen so far in this Course?

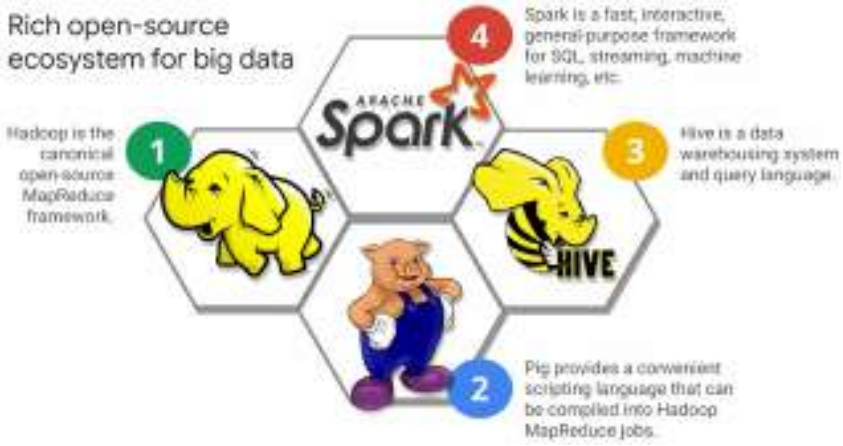

Figure 13: Around 2010, BigQuery was released which was the first of many big data services developed by Google. Around 2015, Google launched Cloud Dataproc which provides a managed service for creating Hadoop and Spark clusters, and managing data processing workloads. Apache Spark is an open source software project that provides a high-performance analytics engine for processing batch and streaming data. Spark can be up to 100 times faster than equivalent Hadoop jobs because it leverages in-memory processing. Spark also provides a couple of abstractions for dealing with data including what are called RDDs, or Resilient Distributed Datasets and DataFrames.

Why the need for Spark? What's Wrong with Hadoop?

- Hadoop made it possible to analyze large data sets, but relied heavily on disk storage (rather than memory) for computation.
- While it's inexpensive to store large volumes of data this way, it makes accessing and processing it much slower.
- Hadoop wasn't a great solution for calculations requiring multiple passes over the same data or many intermediate steps, due to the need to write to and read from the disk between each step.[2]
- Once the cost of RAM (computer memory) started to drop significantly, augmenting or replacing Hadoop by storing data in-memory quickly emerged as an appealing alternative.[3]

---

[2] This drawback also made Hadoop difficult to use for interactive data analysis, the main task data scientists need to do.

[3] Hadoop also suffered and still suffers from suboptimal support for the additional libraries many data scientists needed, such as SQL and machine learning implementations.

Who Developed Spark? How is Spark Better than Hadoop?

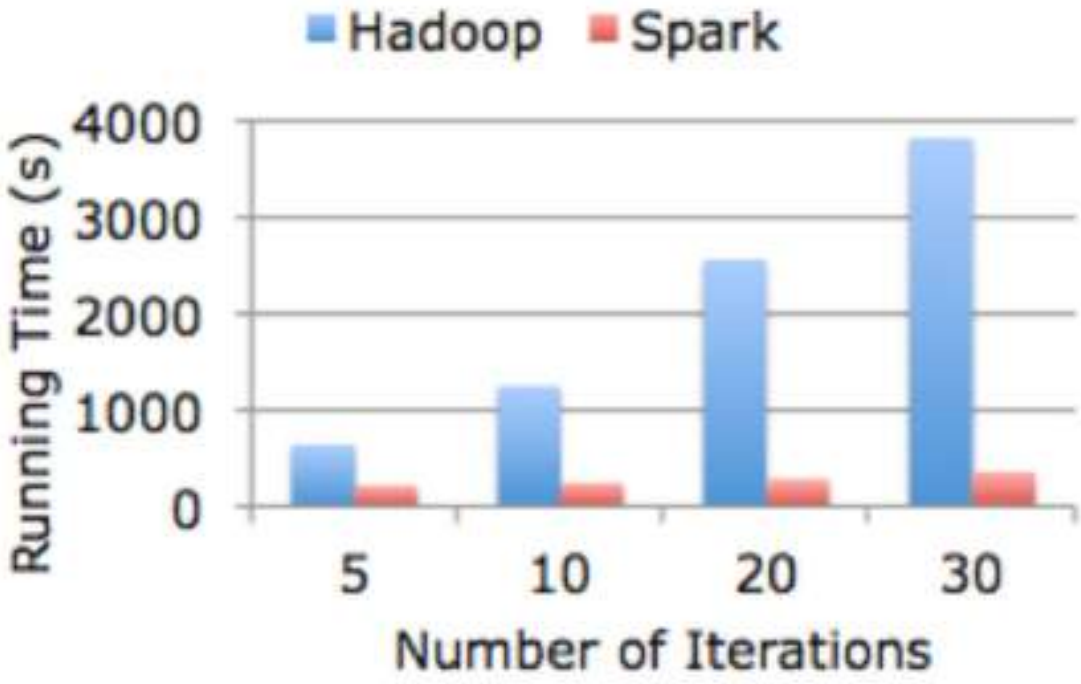

Figure 14: MapReduce is notoriously low-level and difficult to express complex computations in. The nature of MapReduce makes it inefficient for iterative computations, and most machine learning algorithms have an iterative component. So, The UC Berkeley AMP Lab spearheaded groundbreaking work to develop Spark, which uses distributed, in-memory data structures to improve speeds for many data processing workloads by several orders of magnitude.

Read-Eval-Print-Loop (REPLs) and How Spark Follows the Same Principle? (Maybe We're Wrong!)

Figure 15: According to Stuart Halloway (Famous Clojure Developer), the absence of a REPL in Java is the most significant reason why schools started to move to other languages to teach programming. The REPL doesn't need to compile or deploy your code. This leads to a very short response time (<100ms). Thus, you are able to test your hypotheses without losing the flow. Remember the key term in Data Science - ITERATIONS!

Core Data Structures in Spark and What is PySpark?

- The core data structure in Spark is a resilient distributed data set (RDD)
- As the name suggests, an RDD is Spark's representation of a data set that's distributed across the RAM, or memory, of a cluster of many machines
- An RDD object is essentially a collection of elements we can use to hold lists of tuples, dictionaries, lists, etc
- Similar to a pandas DataFrame, we can load a data set into an RDD, and then run any of the methods accessible to that object

PySpark, Scala, JVM, Py4J - Too Much? Don't Worry!

While the Spark toolkit is in Scala, a language that compiles down to bytecode for the JVM, the open source community has developed a wonderful toolkit called PySpark that allows us to interface with RDDs in Python. Thanks to a library called Py4J, Python can interface with Java objects (in our case RDDs). Py4J is also one of the tools that makes PySpark work.

Spark Architecture and Spark Contexts

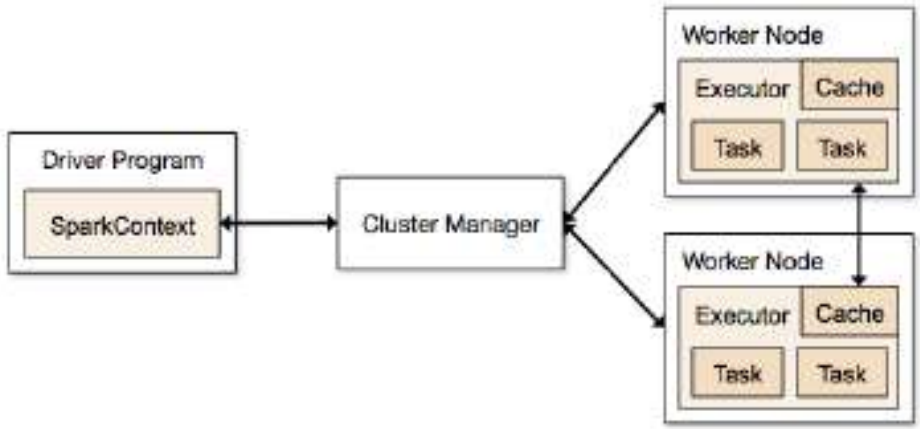

Figure 16: In Spark, the SparkContext object manages the connection to the clusters, and coordinates the running of processes on those clusters. We turn datasets into RDDs using our access to the SparkContext object. **RDD resembles a Python list BUT Spark distributes RDD objects across many partitions, and the RDD object is specifically designed to handle distributed data. We can't rely on the standard implementation of a list for these reasons.**

- Thanks to RDD abstraction, you can run Spark locally on your own computer. Spark will simulate distributing your calculations over many machines by automatically slicing your computer's memory into partitions

What's the thing with Lazy Evaluation? That's What makes Spark Tick!

- Spark's RDD implementation also lets us evaluate code "lazily," meaning we can postpone running a calculation until absolutely necessary
- Spark waits to load the dataset files into an RDD until commands like **raw_data.take(n)** execute.
- When our code calls **raw_data = sc.textFile("some_data_set.tsv")**, Spark creates a pointer to the file, but doesn't actually read it into **raw_data** until **raw_data.take(n)** needs that variable to run its logic
- The advantage of "lazy" evaluation is that we can build up a queue of tasks and let Spark optimize the overall workflow in the background.
- In regular Python, the interpreter can't do much workflow optimization.

Spark Application Flow

Figure 17: Usually Spark applications look like the following — we create RDD (for example, we set data source as file on HDFS), we transform it (map, reduce, join, groupBy, aggregate, reduce,...), do something with the result (for example, we throw it back into HDFS).

```
raw_data =
sc.textFile("daily_show.tsv");
raw_data.take(5);
daily_show =
raw_data.map(lambda line:
line.split(''));
daily_show.take(5);
tally =
daily_show.map(lambda x:
(x[0],1)).reduceByKey(lambda
x,y: x+y);
print(tally)
```

To see the results of these two steps, we'll use the take command, which forces lazy code to run immediately

PySpark: Separate Code Logic (which we write in Python) from the actual Data Transformation! Sweet, ain't it?!

Let's take a line of code such as: **raw_data.map(lambda line: line.split('t'))**: Now, even though the function is in Python, we take advantage of Scala when Spark actually runs the code over the RDD. This is the power of PySpark. Without learning any Scala, we get to harness the data processing performance gains from Spark's Scala architecture.

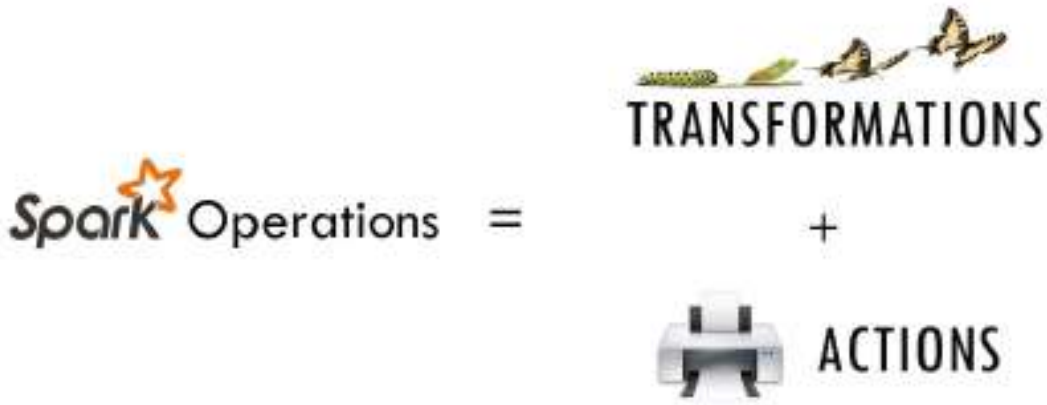

Figure 18: Source: Visual Guide to Spark APIs: `https://training.databricks.com/visualapi.pdf`

Tranformations and Actions: Two Deadly Methods in Spark!

- **Transformations:** map(), reduceByKey(), filter(), flatmap(), groupBy(), groupByKey(), reduceByKey(), mapPartitions(), mapPartitionsWithIndex(), sample(), union(), join(), distinct(), coalesce(), keyBy(), partitionBy(), zip(),
- **Actions:** take(), reduce(), saveAsTextFile(), collect(), getNumPartitions(), aggregate(), max(), sum(), mean(), stdev(), countByKey(), saveAsTextFile()

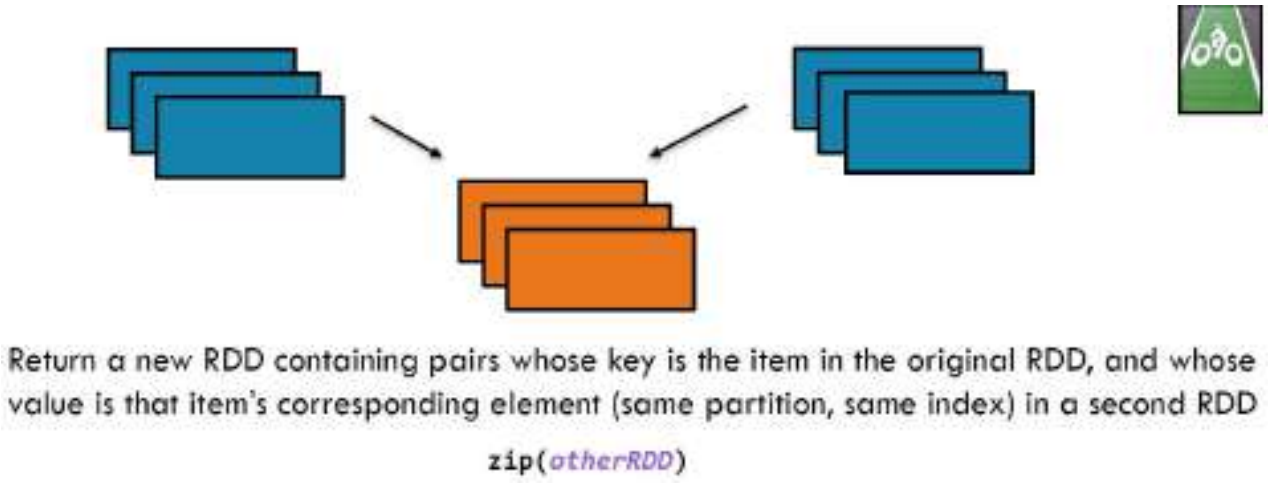

Figure 19: Visual representation of how an example transformation like ZIP fuses two RDDs together

High Level Visual Components of Running Spark on Your Machine

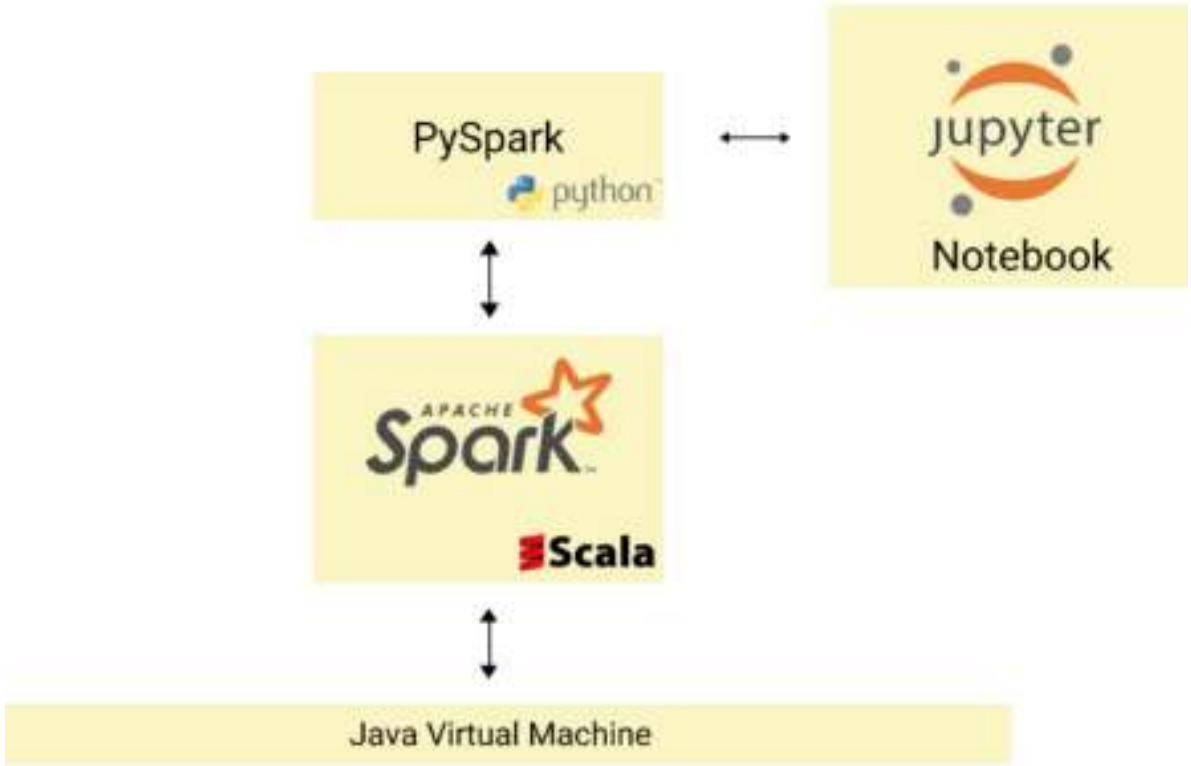

Figure 20: Installing Spark from `http://spark.apache.org/downloads.html` (211 MB) after installing recent version of JDK 8 (Java Development Kit) + Adding SPARK_HOME environment variables + Installing PySpark and findspark via pip + Start importing datasets into RDDs via SparkContext()

Advantages of DAG (Directed Acyclic Graph) execution of big data algorithms over MapReduce?

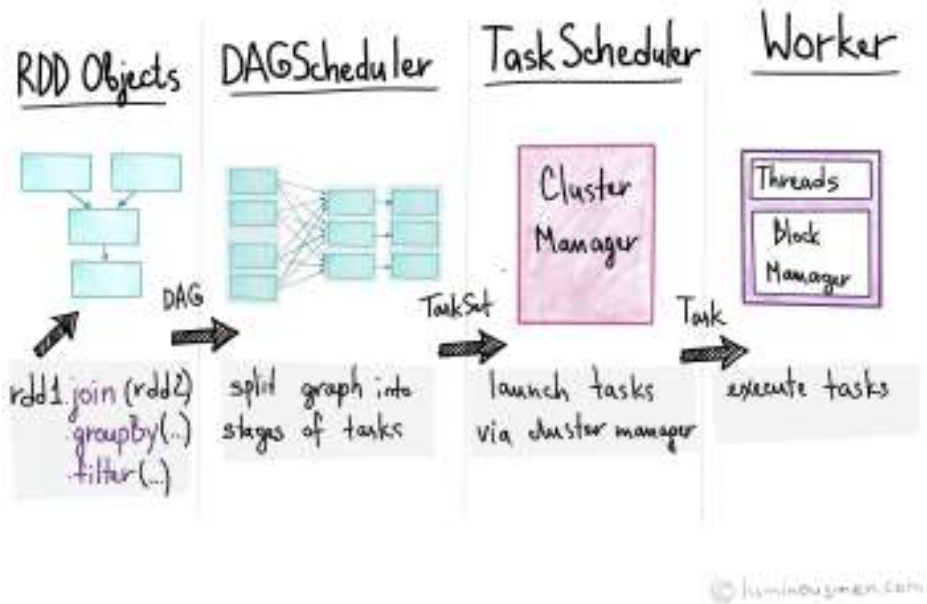

Figure 21: Unlike Hadoop, Spark identifies the tasks that can be computed in parallel with partitioned data on the cluster. With these identified tasks, Spark builds a logical flow of operations that can be represented in a graph that is a Directed Acyclic Graph, where a node is RDD partition and the edge is transformation on the data. Thus Spark builds its plan of executions implicitly from the provided spark application. Source: `https://bit.ly/2xk3wQe`

Immutability of RDD: A Deep Dive into Fault Tolerance!

- Spark uses the immutability of RDDs to enhance calculation speeds.
- Rules out updates from multiple threads at once so immutable data is definitely safe to share across processes
- So What's the trade-off?[4]
- Adding memory is much easier than adding I/O bandwidth
- Of course, an RDD isn't really a collection of data, but just a recipe for making data from other data.

```
filtered_daily_show.filter(lambda line: line[1] != '') \
                   .map(lambda line: (line[1].lower(), 1)) \
                   .reduceByKey(lambda x,y: x+y) \
                   .take(5)
```

Figure 22: Hadoop had to write intermediate results to disk, and wasn't aware of the full pipeline. Spark improves on Hadoop's turnaround time significantly.

[4] gaining the fault tolerance and correctness with no developer effort is worth spending memory and CPU on, since the latter are cheap

Spark Dataframes and Spark SQL (Contd. Next Slide):The Real Beasts in the Room!

- Combines the scale and speed of Spark with the familiar query, filter, and analysis capabilities of pandas
- Unlike pandas, which can only run on one computer, Spark can use distributed memory (and disk when necessary) to handle larger data sets and run computations more quickly[5]
- We can even use a SQL interface to write **distributed SQL queries** that query large database systems and other data stores
- Spark DataFrames allow us to interface with different types of data:
  - JSON, CSV/TSV, XML Parquet, Amazon S3 (cloud storage service)
  - Big Data Systems Hive, Avro, HBase
  - SQL Database Systems MySQL, PostgreSQL

[5]Spark DataFrames allow us to modify and reuse our existing pandas code to scale up to much larger data sets. They also have better support for various data formats

Spark SQL: Combining the best aspects of both DataFrames and SQL

- We need to tell Spark to treat the DataFrame as a SQL table - SIMPLE! Can you do that in Pandas?
- Spark internally maintains a virtual database within the SQLContext object[6]
- Done! Now we can start writing and running SQL queries on that dataframe table
- Remember that because the results of SQL queries are DataFrame objects, we can combine the best aspects of both DataFrames and SQL to enhance our workflow!

Most Fantastic Feature:

One of the most powerful use cases in SQL is joining tables. Spark SQL takes this a step further by enabling you to run join queries across data from multiple file types.

---

[6] This object, which we enter as sqlCtx, has methods for registering temporary tables.

Some Code Snippets Illustrating Spark Dataframes and Spark SQL

```
from pyspark.sql import SQLContext
sqlCtx = SQLContext(sc)
df = sqlCtx.read.json("census_2010.json")
df.registerTempTable("census2010")
tables = sqlCtx.tableNames()
print(tables)

sqlCtx.sql("select age from census2010").show()

query = 'select males, females from census2010 where age>5
sqlCtx.sql(query).show()

query = 'select males,females from census2010'
sqlCtx.sql(query).describe().show()

df_2000 = sqlCtx.read.json("census_2000.json")
df_1990 = sqlCtx.read.json("census_1990.json")
df_1980 = sqlCtx.read.json("census_1980.json")
```

Figure 23:

```
df_2000.registerTempTable('census2000')
df_1990.registerTempTable('census1990')
df_1980.registerTempTable('census1980')
tables = sqlCtx.tableNames()
print(tables)

query = """
 select census2010.total, census2000.total
 from census2010
 inner join census2000
 on census2010.age=census2000.age
"""

sqlCtx.sql(query).show()
```

Figure 24:

We've served whatever the Nation wanted to Know! We're Done!

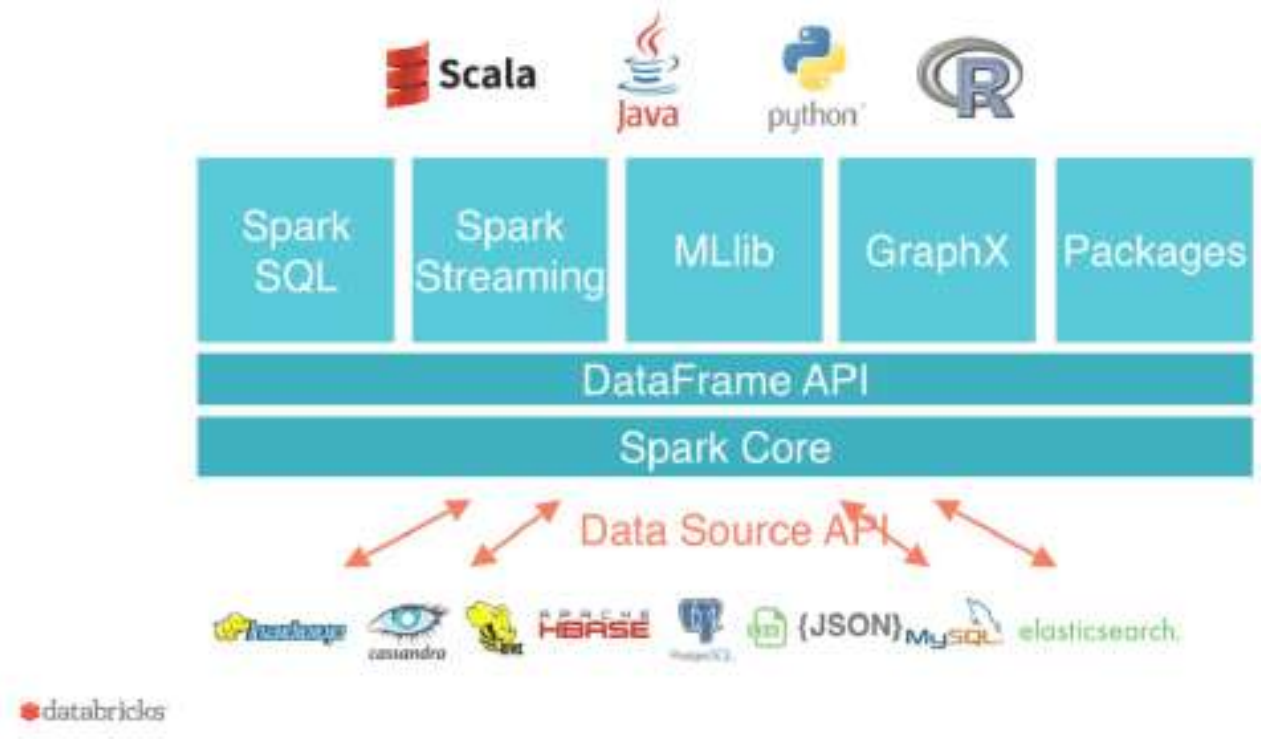

Figure 25: Spark Pipeline API is designed entirely around Dataframes as their sole data structure for parallel computations, model training and predictions. **Source**: Databricks Founder Patrick Wendell's presentation - "What the future of Spark is?" (2015)

But Forget about the Nation! What does the World Want to Know? Who can Sort Faster?

Figure 26: In 2014, Databricks (Original Makers of Spark at UC Berkeley) entered the competition using a distributed program built on top of Spark. That system sorted 100 TB of data in 23 minutes, using only 207 machines on EC2. **That is to say, Spark sorted the same data 3X faster using 10X fewer resources than the 2013 Hadoop entry.** In addition to winning the benchmark, they also sorted 1 PB of data in 4 hours using a similar setup and achieved near linear scalability.