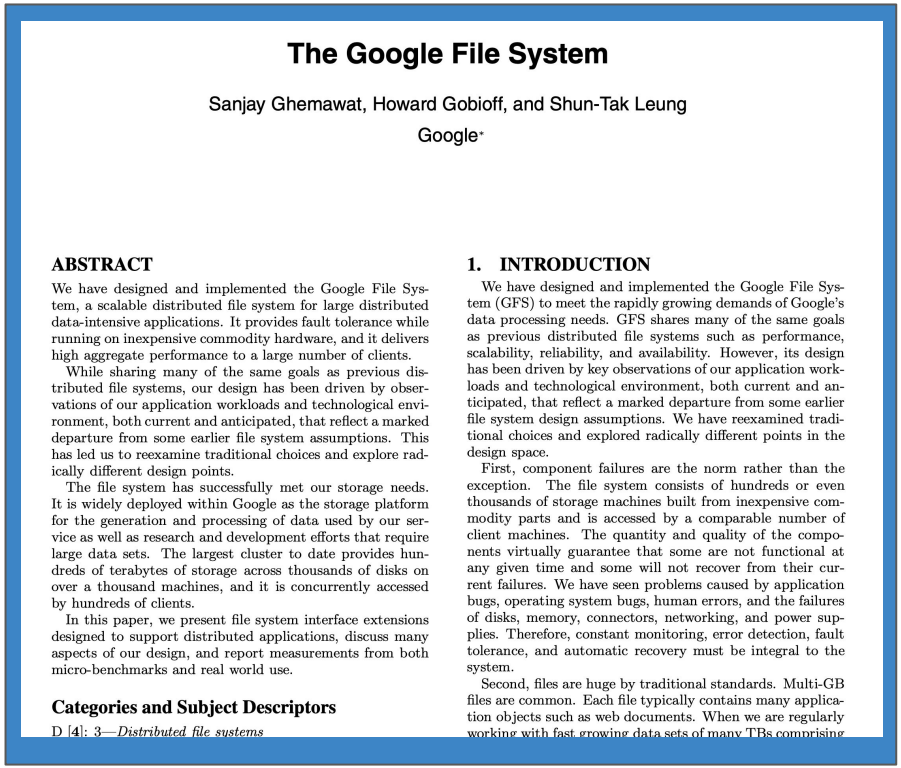
# Smarter Agent TechTalks

Apache Spark Pipelines

# The Problem: How do we handle big data?

- 2003 Google File System Paper
  - Outlined the theory and algorithms behind the building blocks of Google Bigtable (2006) and map-reduce computation (2004)
  - Basis for several current Google technologies including cloud, maps, earth, gmail, youtube...



**The Google File System**

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung

Google*

**ABSTRACT**

We have designed and implemented the Google File System, a scalable distributed file system for large distributed data-intensive applications. It provides fault tolerance while running on inexpensive commodity hardware, and it delivers high aggregate performance to a large number of clients.

While sharing many of the same goals as previous distributed file systems, our design has been driven by observations of our application workloads and technological environment, both current and anticipated, that reflect a marked departure from some earlier file system assumptions. This has led us to reexamine traditional choices and explore radically different design points.

The file system has successfully met our storage needs. It is widely deployed within Google as the storage platform for the generation and processing of data used by our service as well as research and development efforts that require large data sets. The largest cluster to date provides hundreds of terabytes of storage across thousands of disks on over a thousand machines, and it is concurrently accessed by hundreds of clients.

In this paper, we present file system interface extensions designed to support distributed applications, discuss many aspects of our design, and report measurements from both micro-benchmarks and real world use.

**Categories and Subject Descriptors**

D [4]: 3—*Distributed file systems*

**1. INTRODUCTION**

We have designed and implemented the Google File System (GFS) to meet the rapidly growing demands of Google's data processing needs. GFS shares many of the same goals as previous distributed file systems such as performance, scalability, reliability, and availability. However, its design has been driven by key observations of our application workloads and technological environment, both current and anticipated, that reflect a marked departure from some earlier file system design assumptions. We have reexamined traditional choices and explored radically different points in the design space.

First, component failures are the norm rather than the exception. The file system consists of hundreds or even thousands of storage machines built from inexpensive commodity parts and is accessed by a comparable number of client machines. The quantity and quality of the components virtually guarantee that some are not functional at any given time and some will not recover from their current failures. We have seen problems caused by application bugs, operating system bugs, human errors, and the failures of disks, memory, connectors, networking, and power supplies. Therefore, constant monitoring, error detection, fault tolerance, and automatic recovery must be integral to the system.

Second, files are huge by traditional standards. Multi-GB files are common. Each file typically contains many application objects such as web documents. When we are regularly working with fast growing data sets of many TBs comprising

# The Problem: How do we handle big data?

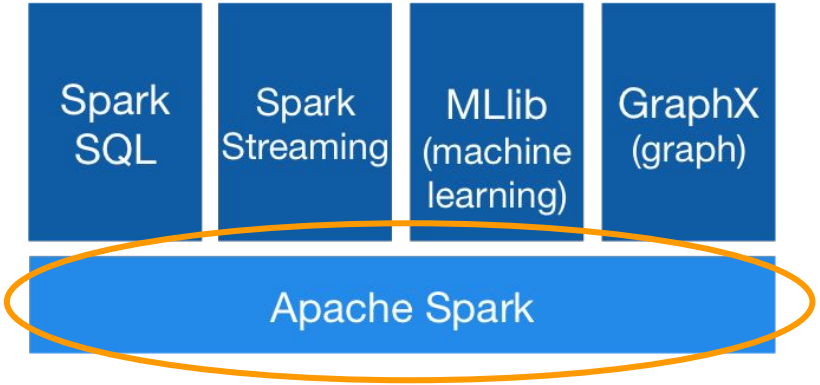
- 2006 Open Source Hadoop
  - Hadoop Distributed File System (HDFS) datastore
  - Map-reduce functionality
  - No in-memory computation, lots of disc i/o
- 2010 Open Source Spark
  - Builds on Hadoop functionality
  - Map-reduce functionality in memory when possible

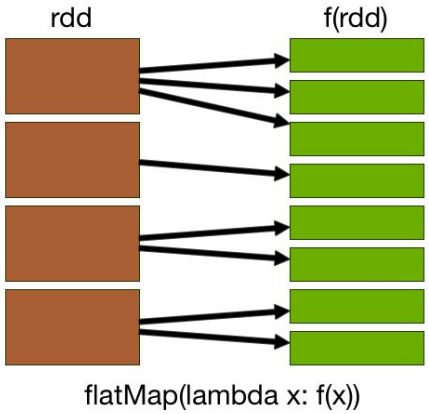
2014 big data sort

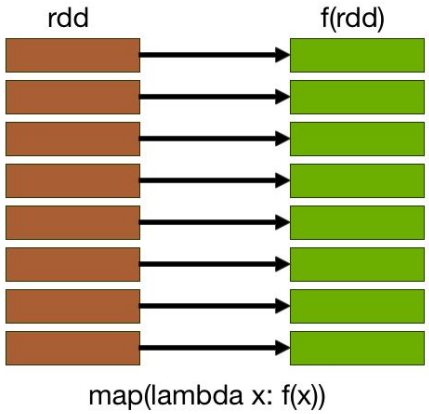
| | Hadoop MR Record | Spark Record | Spark 1 PB |
|---|---|---|---|
| Data Size | 102.5 TB | 100 TB | 1000 TB |
| Elapsed Time | 72 mins | 23 mins | 234 mins |
| # Nodes | 2100 | 206 | 190 |
| # Cores | 50400 physical | 6592 virtualized | 6080 virtualized |
| Cluster disk throughput | 3150 GB/s (est.) | 618 GB/s | 570 GB/s |
| Sort Benchmark Daytona Rules | Yes | Yes | No |
| Network | dedicated data center, 10Gbps | virtualized (EC2) 10Gbps network | virtualized (EC2) 10Gbps network |
| **Sort rate** | **1.42 TB/min** | **4.27 TB/min** | **4.27 TB/min** |
| **Sort rate/node** | **0.67 GB/min** | **20.7 GB/min** | **22.5 GB/min** |

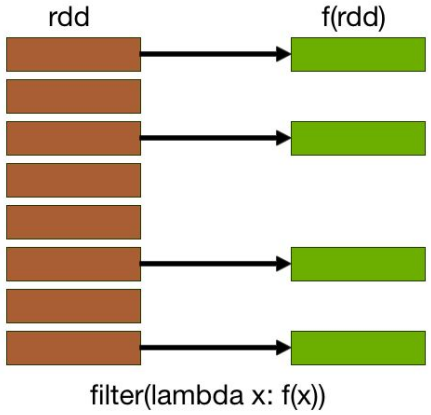
# Core Spark Basics

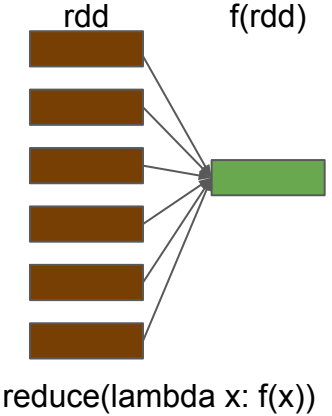
- Basic data type - resilient distributed dataset (rdd)
- Array like distributed data structure



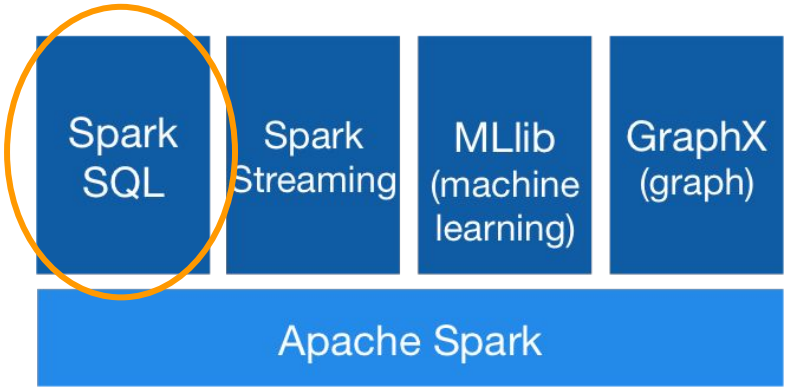



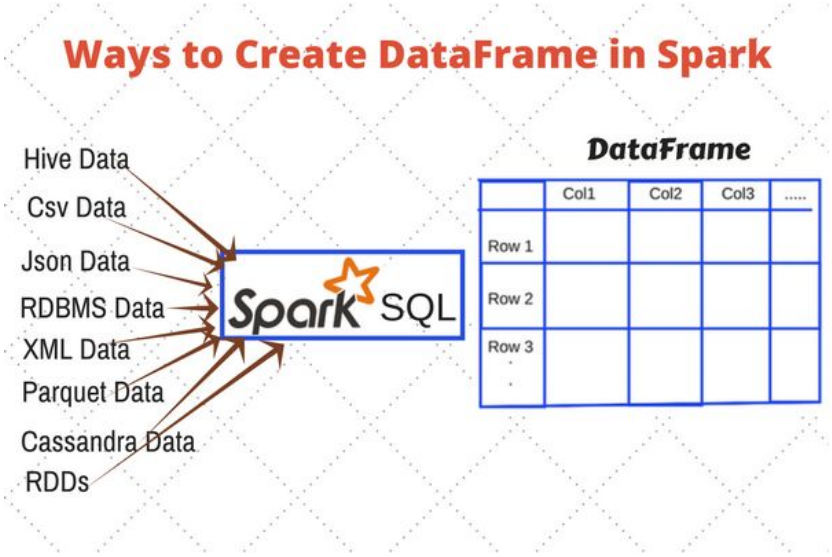



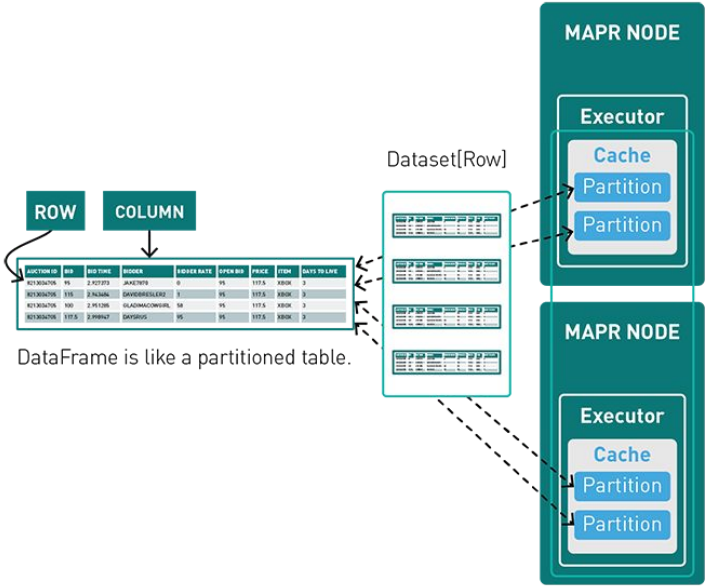



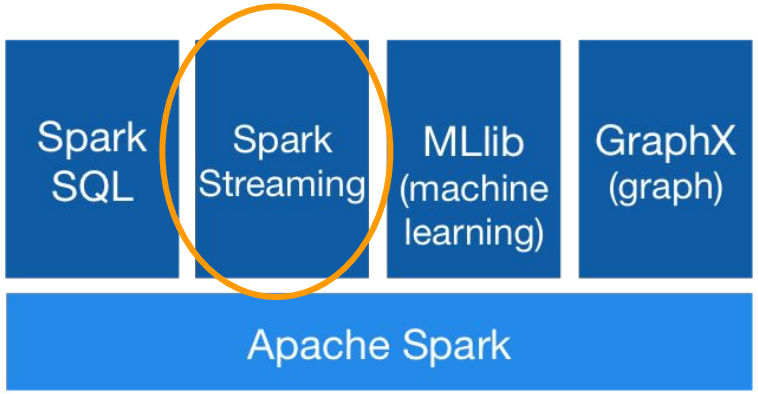
# Spark SQL Basics

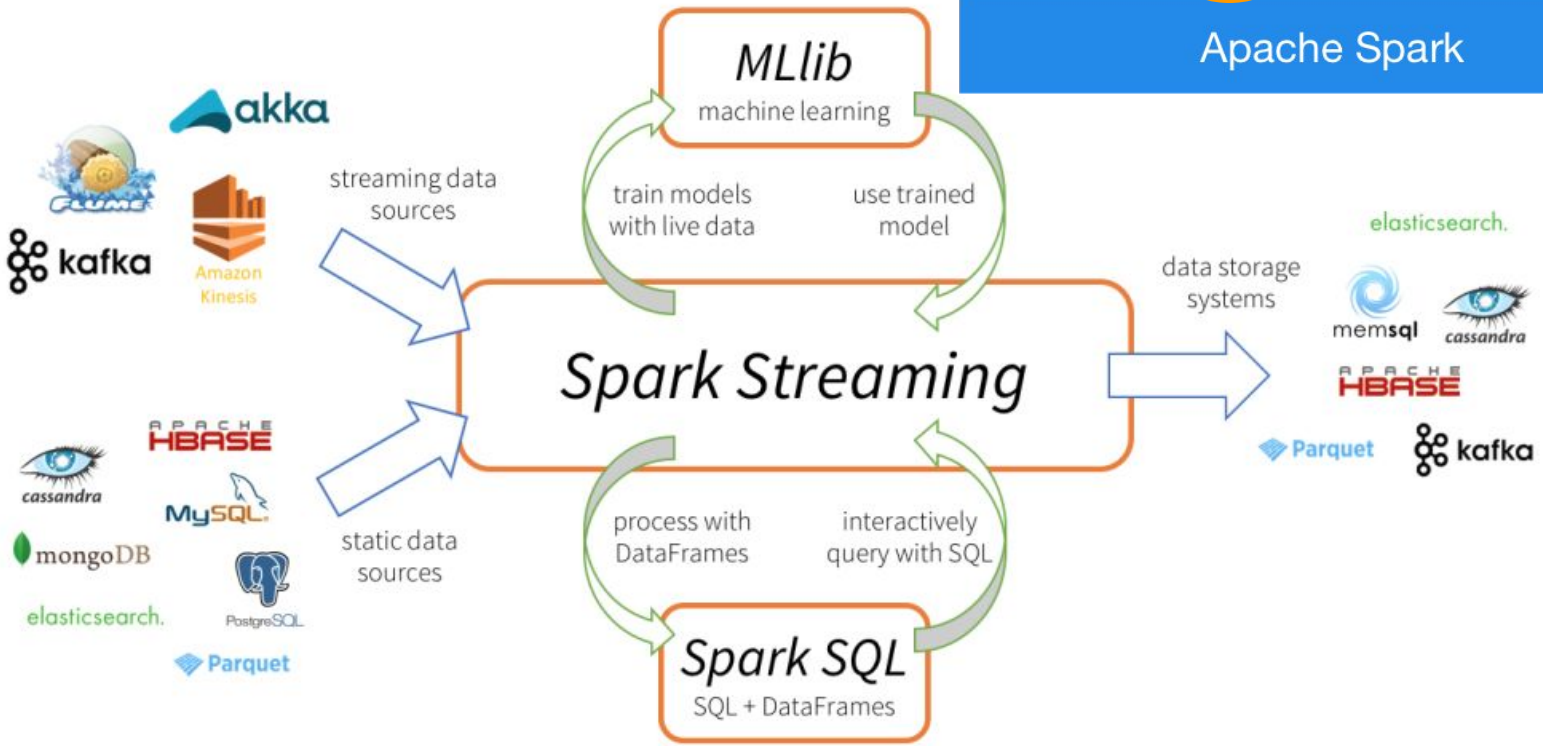


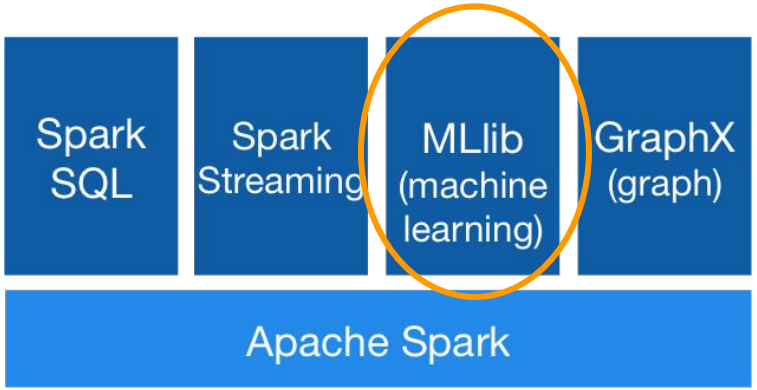



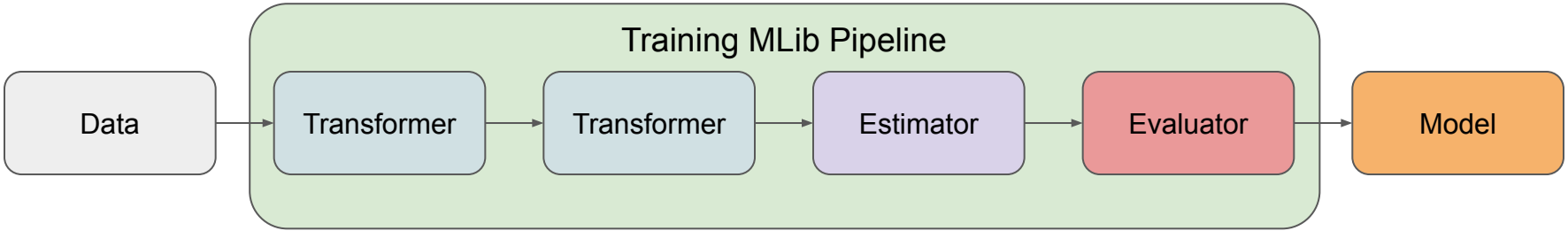
# Spark Streaming Basics

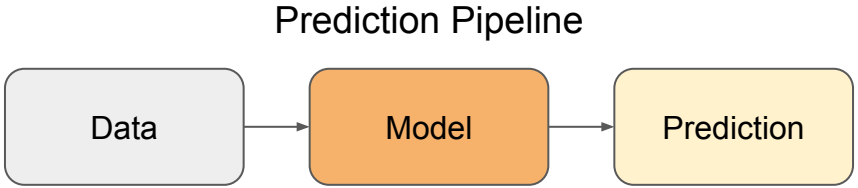


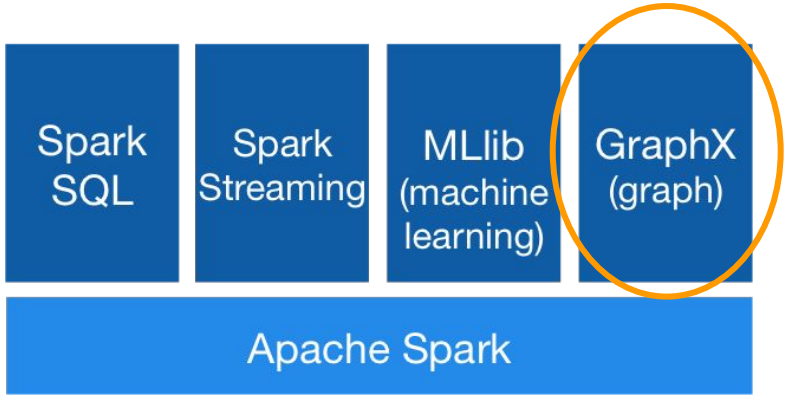
# Spark MLib Basics

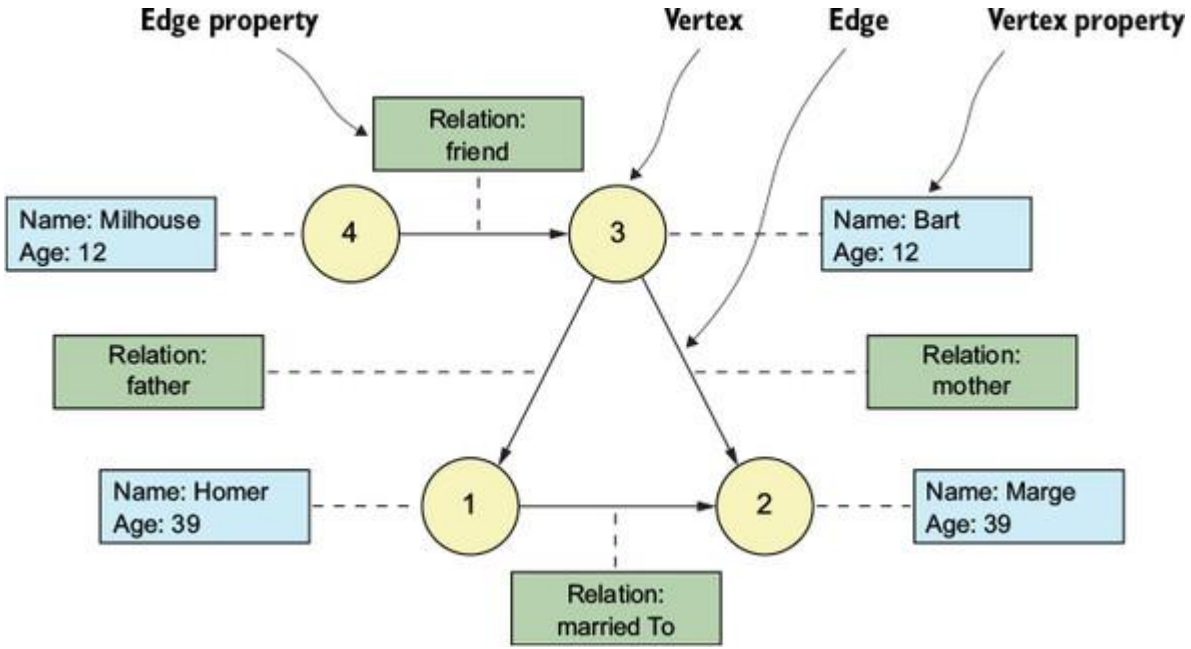


# Spark GraphX Basics




- Analyze relationships between elements of a network in parallel
- Pagerank algorithms for connections between vertices, edge analytics, etc