Understanding Spark

Bellevue/Seattle, July 29, 2017

To Do

- WiFi Password: lemon...7
- Get on Seattle Spark Slack channel: goo.gl/LV3Irv
- Sign up to out mailing list: eepurl.com/cXm00r
- Check out Seattle Spark Meetup page
- To get started with Databricks: goo.gl/oLa8qx
- Tweet pictures with hashtag *#SparkSaturday*

Today's Agenda

09:00 Registration, breakfast, social

09:30 Intro to Big Data, Hadoop and Spark

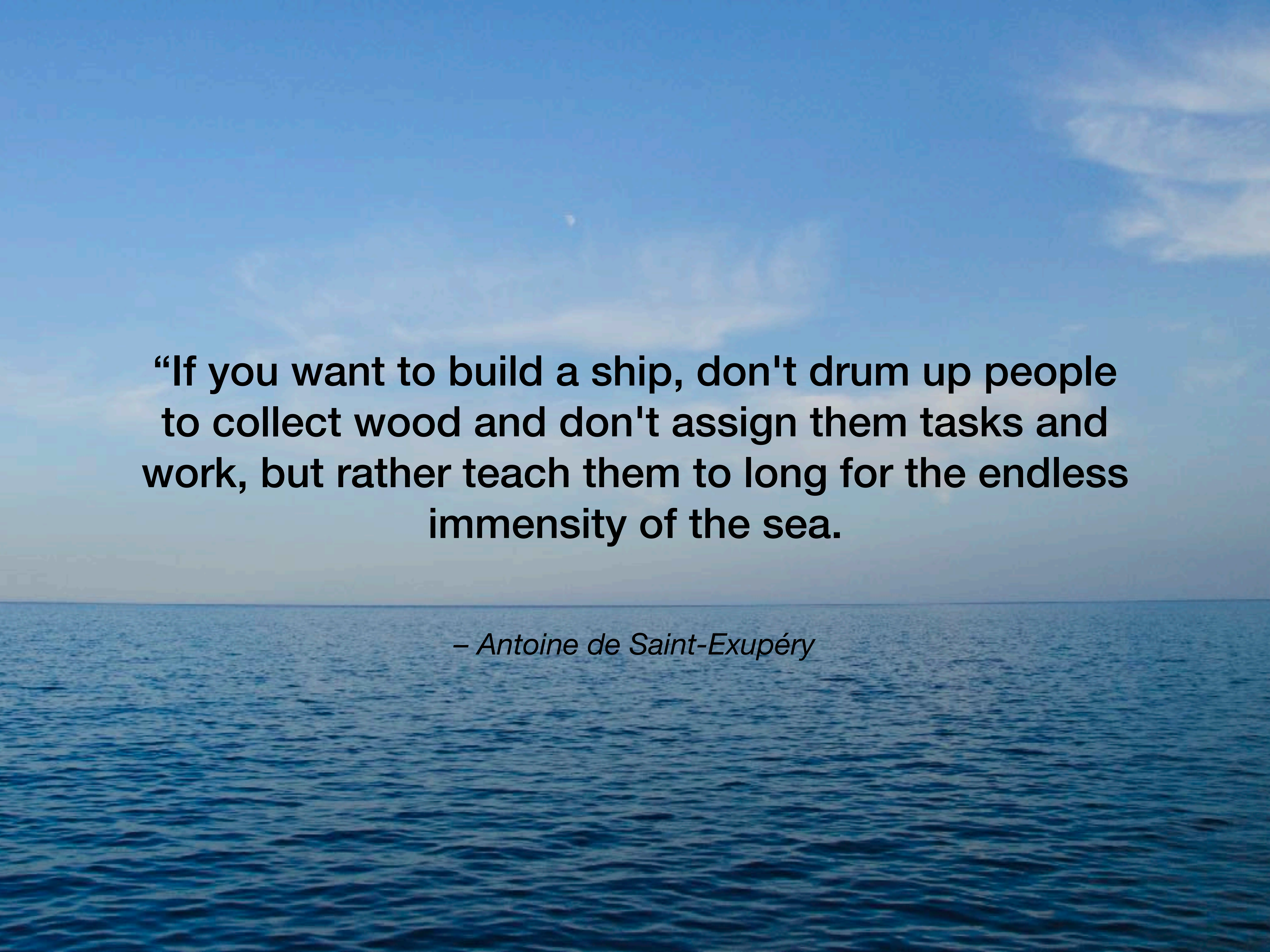
10:00 Datasets, Dataframes, Spark SQL

10:50 Break

11:00 Spark ML and ML Pipelines

11:30 Practical ETL with Databricks

12:00 End


“If you want to build a ship, don't drum up people to collect wood and don't assign them tasks and work, but rather teach them to long for the endless immensity of the sea.
– Antoine de Saint-Exupéry

databricks™

Spark

Blueprint™

CONSULTING SERVICES

Thanks to our sponsors!

extraSlice

The Place for Tech

METIS®

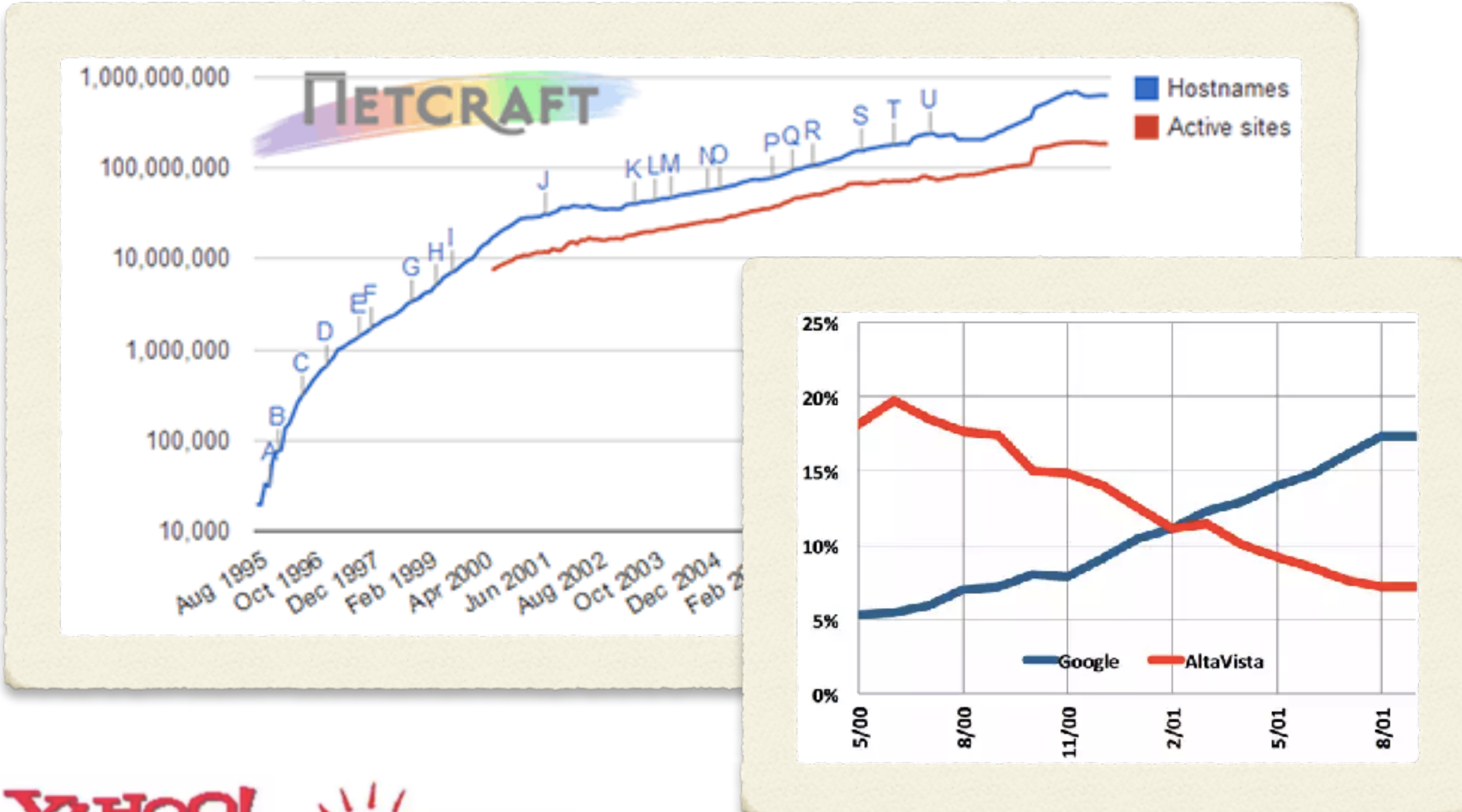
E8

Big Data, Hadoop and Spark

Recent history, briefly

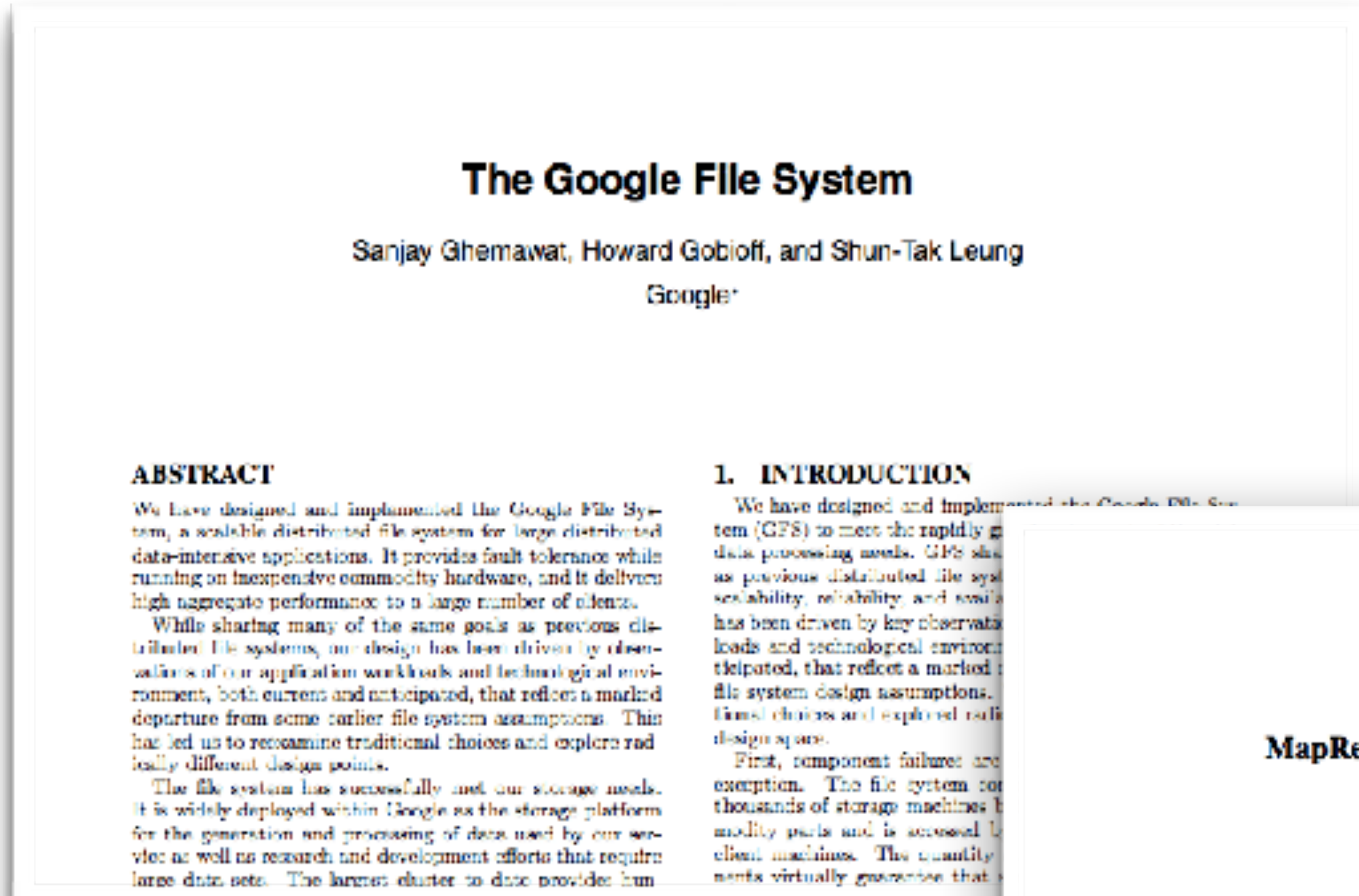
Understanding Spark, Bellevue/Seattle, July 29, 2017

Growth of the Internet



YAHOO!
excite
Ask Jeeves
Google
HOTBOT
altavista
LYCOS

Google: Processing Large Amounts of Data



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.

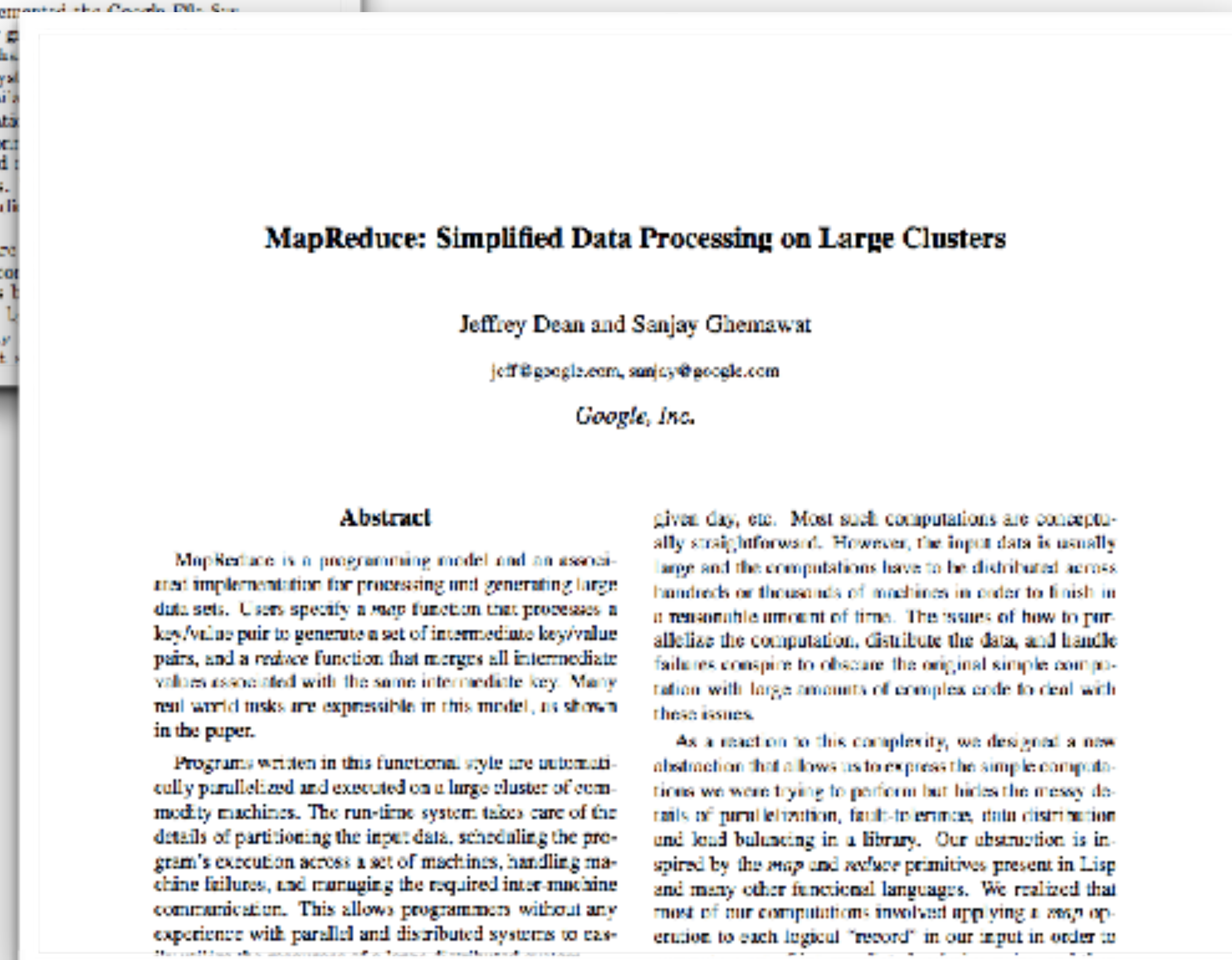
While sharing many of the same goals as previous distributed file systems, our design has been driven by observations of our application workloads and technological environment, both current and anticipated, that reflect a marked departure from some earlier file system assumptions. This has led us to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

1. INTRODUCTION

We have designed and implemented the Google File System (GFS) to meet the rapidly g… data processing needs. GFS sha… as previous distributed file syst… scalability, reliability, and avail… has been driven by key observati… loads and technological environ… ticipated, that reflect a marked … file system design assumptions. … tional choices and explored radi… design space.

First, component failures are … exception. The file system co… thousands of storage machines b… modity parts and is accessed b… client machines. The quantity … nents virtually guarantee that …



MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat
jeff@google.com, sanjay@google.com
Google, Inc.

Abstract

MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a *map* function that processes a key/value pair to generate a set of intermediate key/value pairs, and a *reduce* function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model, as shown in the paper.

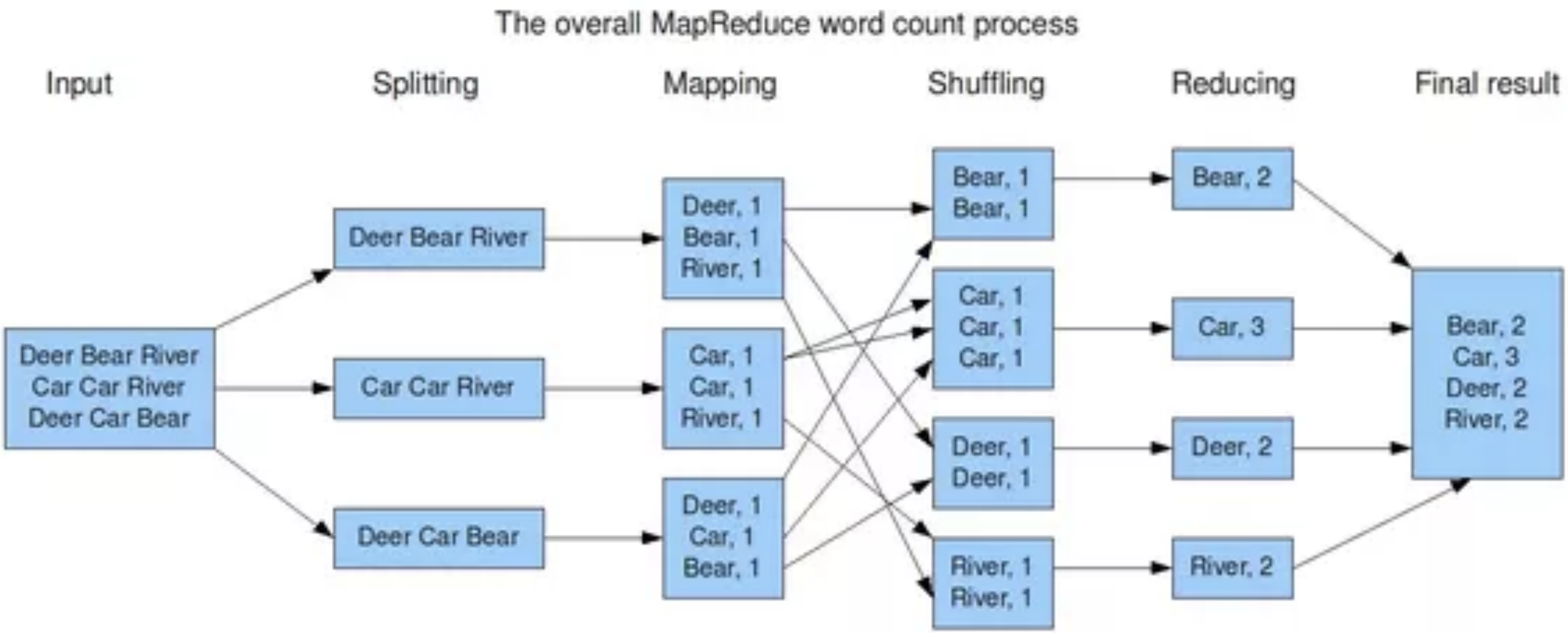
Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The run-time system takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling machine failures, and managing the required inter-machine communication. This allows programmers without any experience with parallel and distributed systems to eas…

given day, etc. Most such computations are conceptually straightforward. However, the input data is usually large and the computations have to be distributed across hundreds or thousands of machines in order to finish in a reasonable amount of time. The issues of how to parallelize the computation, distribute the data, and handle failures conspire to obscure the original simple computation with large amounts of complex code to deal with these issues.

As a reaction to this complexity, we designed a new abstraction that allows us to express the simple computations we were trying to perform but hides the messy details of parallelization, fault-tolerance, data distribution and load balancing in a library. Our abstraction is inspired by the *map* and *reduce* primitives present in Lisp and many other functional languages. We realized that most of our computations involved applying a *map* operation to each logical "record" in our input in order to

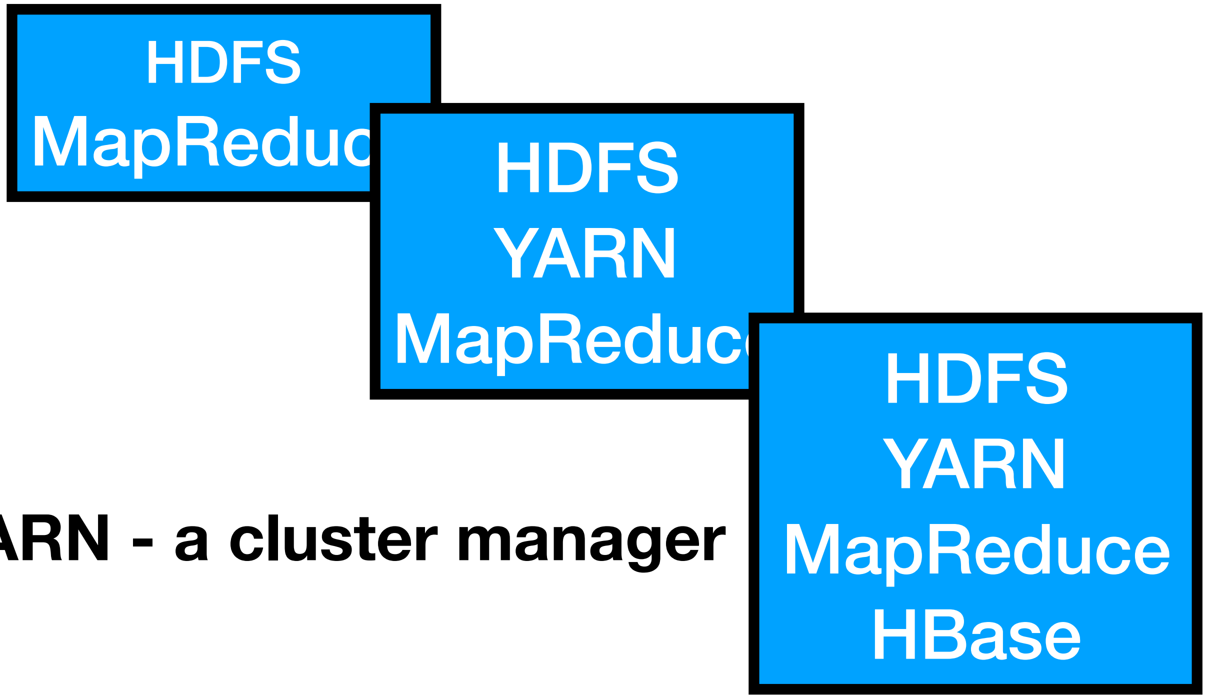
2003: Google File System
2004: MapReduce

A MapReduce program



Evolution of Hadoop

HDFS - a distributed file system



YARN - a cluster manager

New cluster managers: Mesos, Kubernetes

New distributed applications running on clusters

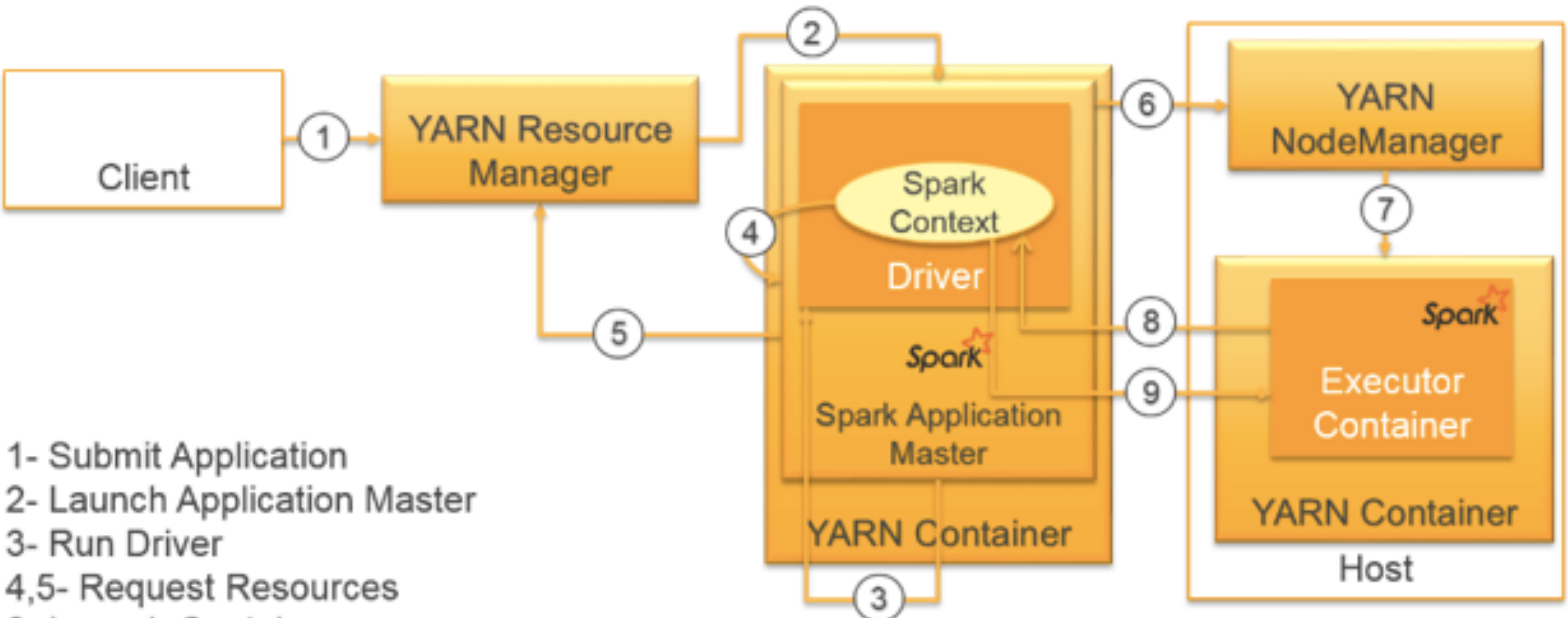
Distributed applications running on clusters

HDFS
YARN
Spark

HDFS - a distributed file system

YARN - a cluster manager

Spark - a distributed application running on a cluster



1- Submit Application
2- Launch Application Master
3- Run Driver
4,5- Request Resources
6- Launch Containers
7- Launch Spark Executors
8- Register with the Driver
9- Launch Tasks

Spark competes with MapReduce

Advantages of Spark: Faster execution, better API

- ✓ **Pipelining of operations**
- ✓ **In-memory caching** — Memory became relatively cheaper!
- ✓ **Long-running JVMs**
- ✓ **RDD abstraction**
- ✓ **Interactive (e.g. shell/notebook)**

Architecture of Spark