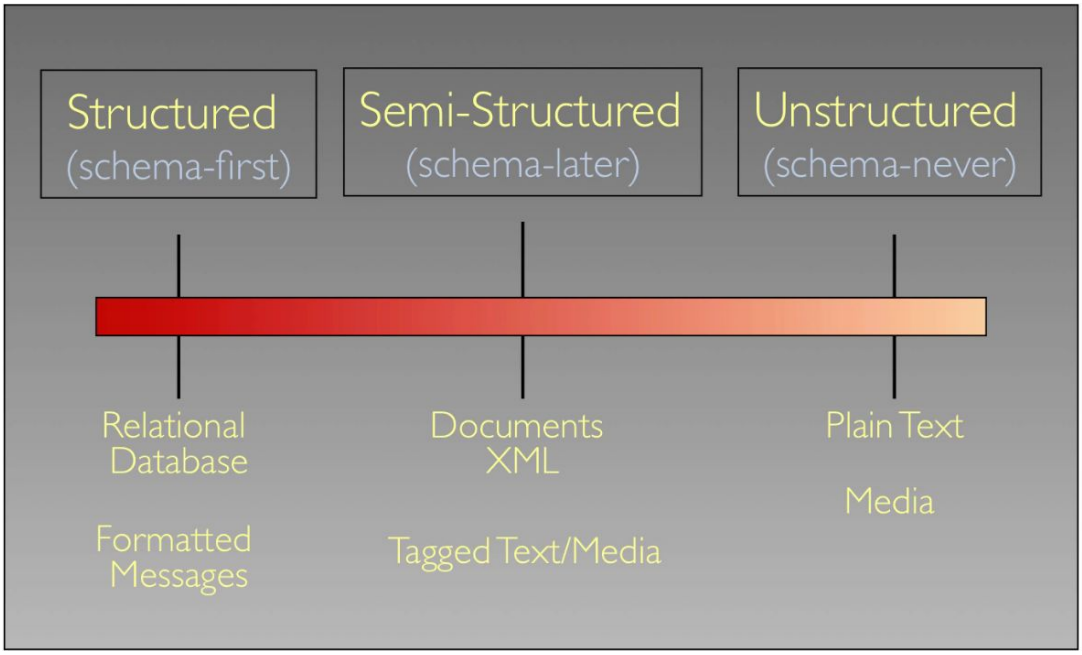
Spark
Data management concepts
A data model is a collection of concepts for describing data
A schema is a description of a particular collection of data using a given data model.

RFID tags

The structure spectrum
Each column has a type (string, integer,..)
Together, the column types are the schema for the data



Spark can infer the schema while reading each row.

What to do with Big data?
Crowdsourcing + Physical modeling + Sensing + Data Assimilation = Real-time traffic info

Big Data Problem
Growing data sources, storage getting cheap but stalling CPU speeds and storage bottlenecks

Examples:
Facebook daily logs: 60TB
1000 genome project: 200 TB
Google web index: 10+ PB
Cost of 1 TB of disk: $35
Time to read 1 TB from disk: 3 hours (100MB/s)

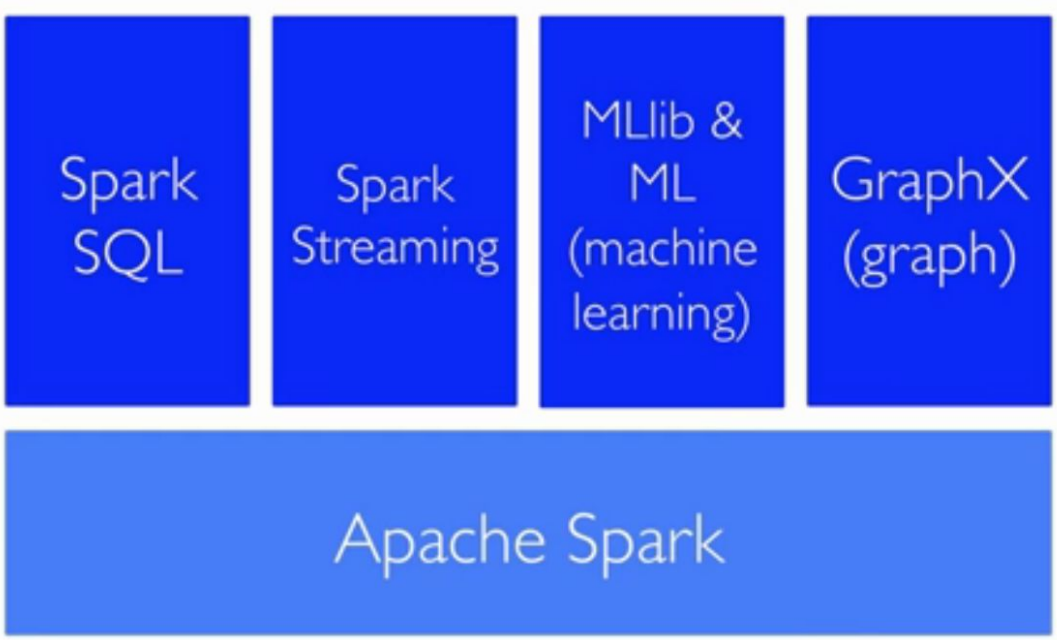
Solution is to distribute data over cluster of machines:

Lots of hard drives, CPUs and memory.

Spark DataFrame
We take our big data and partition it up across all of the machines in the cluster. Each machines gets some of the rows from the big data that we want to store and analyze.

Spark Computing Framework
Provides programming abstraction and parallel runtime to hide complexities of fault-tolerance and slow machines.

A Spark program has two programs:
A driver program and a workers program
Worker programs run on cluster nodes or in local threads
DataFrames are distributed across workers.



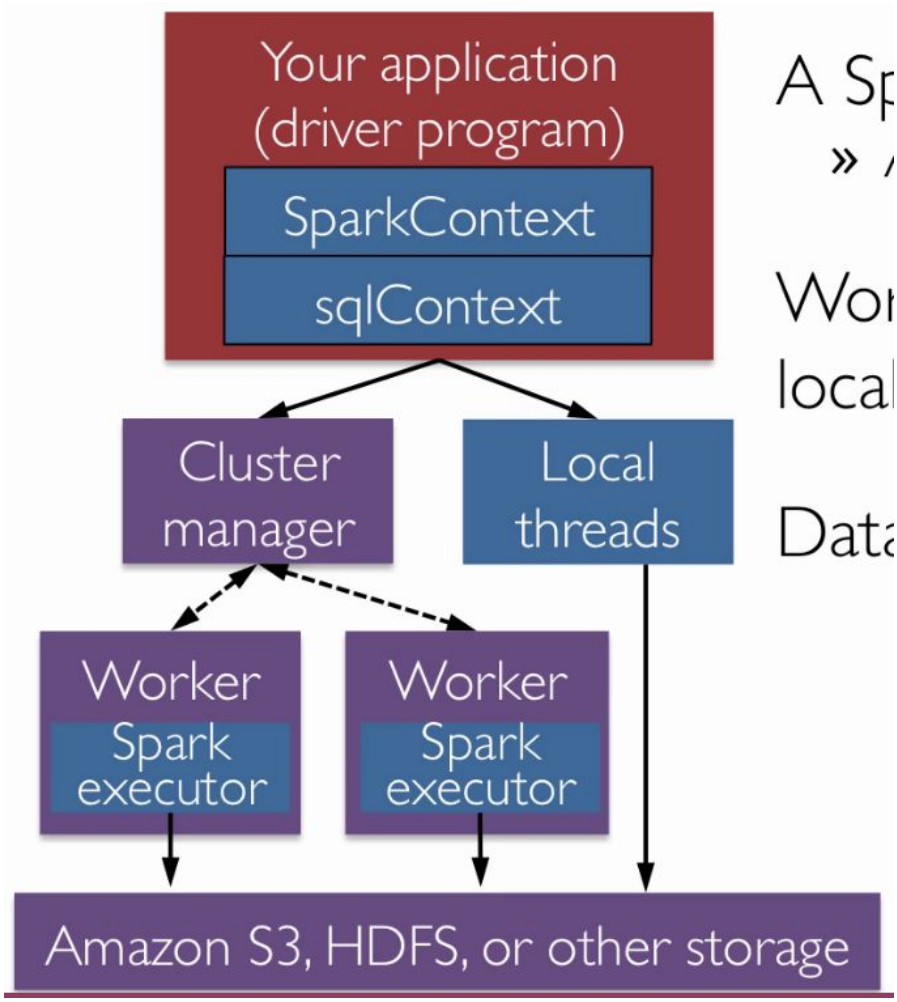
Spark and SQL Contexts
A spark program first creates SparkContext object
SparkContext tells Spark how and where to access a cluster
Then, the program creates a sqlContext object
Use sqlContext to create DataFrames

The master parameter for a SparkContext determines which type and size of cluster to use

Master Parameter	Description
`local`	run Spark locally with one worker thread (no parallelism)
`local[K]`	run Spark locally with K worker threads (ideally set to number of cores)
`spark://HOST:PORT`	connect to a Spark standalone cluster; PORT depends on config (7077 by default)
`mesos://HOST:PORT`	connect to a Mesos cluster; PORT depends on config (5050 by default)

DataFrames
The primary abstraction in Spark
- Immutable once constructed
- Track lineage information to efficiently recompute lost data
- Enable operations on collection of elements in parallel

There are two types of operations: transformations and actions
Transformations are lazy (not computed immediately)
Transformed DF is executed when action runs on it
Persist (cache) DFs in memory or disk



Spark Transformations
Use lazy evaluation: results not computed right away- Spark remembers set of transformations applied to base DataFrame
- Spark uses Catalyst to optimize the required calculations
- Spark recovers from failures and slow workers

Spark Program lifecycle
- Create DataFrame from external data or createDataFrame from a collection in driver program
- Lazily transform them into new DataFrames
- cache() some DataFrames for re-use
- Perform actions to execute parallel computation and produce results (print)

Where code runs
Most python code runs in the driver
Transformations run at executors
Actions run at executors and driver

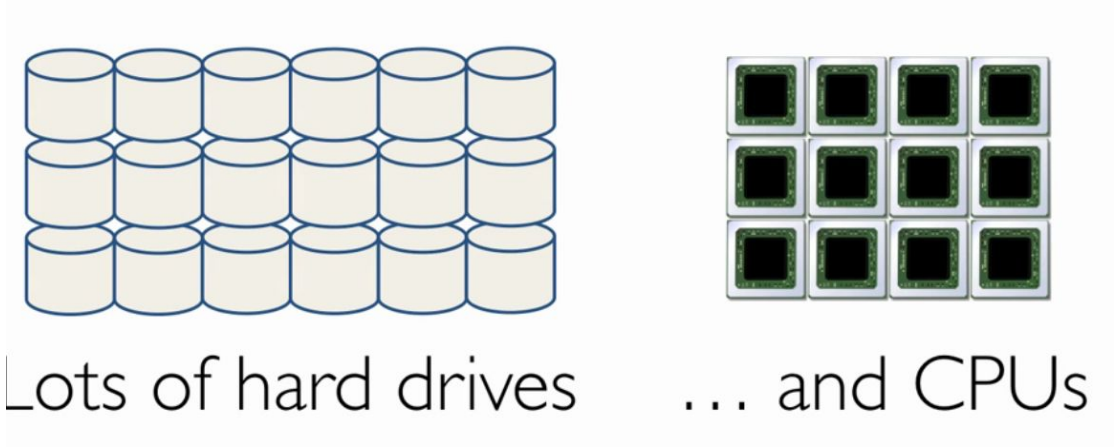
The structured query language and Spark SQL

The Big Data problem
Apache Spark
Scalable, efficient analysis of Big Data

Data growing faster than CPU speeds
Data growing faster than per-machine storage
Cannot process or store all data on a single machine

Cloud computing provides access to low-cost computing and storage
Costs decreasing every year, but the challenge is programming the resources

Cluster Computing Challenges and the Map Reduce Programming Paradigm
Big Data Processing

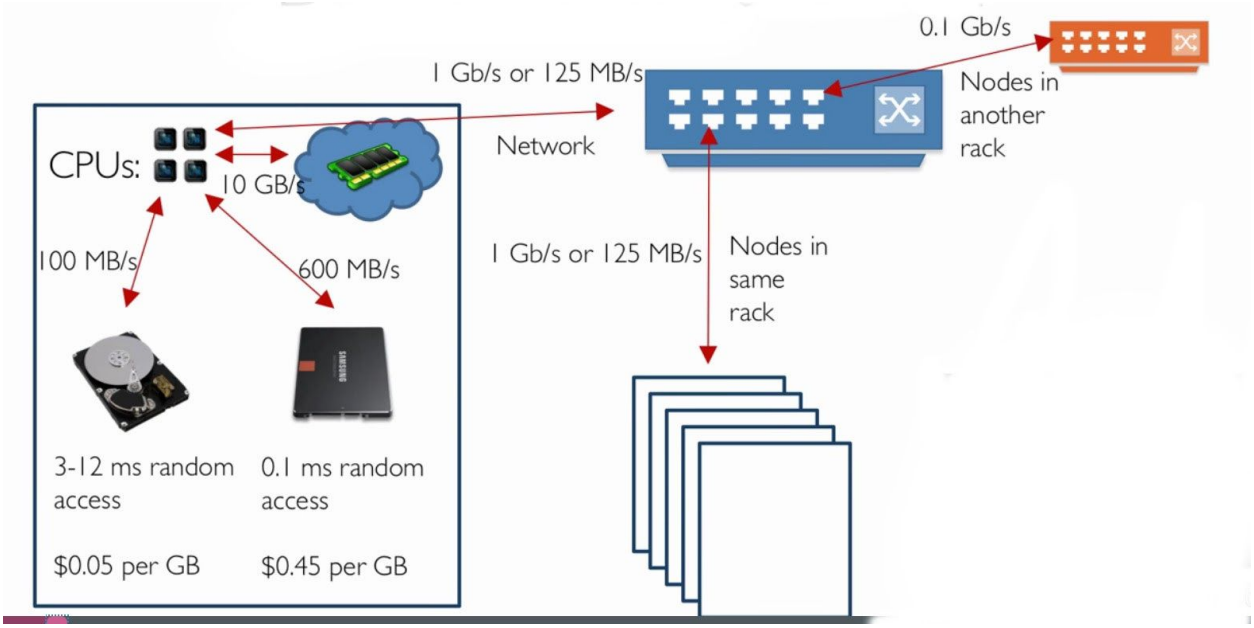


A big box: very expensive, low volume, all premium and not big enough

We need consumer-grade hardware
Many desktop-like servers: easy to add capacity and cheaper per CPU/disk
But increase the complexity in software

Problems with cheap hardware
Failures (Google's number): 1-5% hard drives/year 0.2% DIMMs/year

Datacenter organization



Network speeds: Much more latency, Network slower than storage
Uneven performance

Example: Word counting
Hashing (dictionary), Divide and conquer, several layers of aggregation
Map: distribute the words (data)
Reduce: partition the counting results by words

Sorting:
partition by occurrences

Challenges in cluster computing
How to divide work across machines
Must consider network, data locality
Moving data may be very expensive

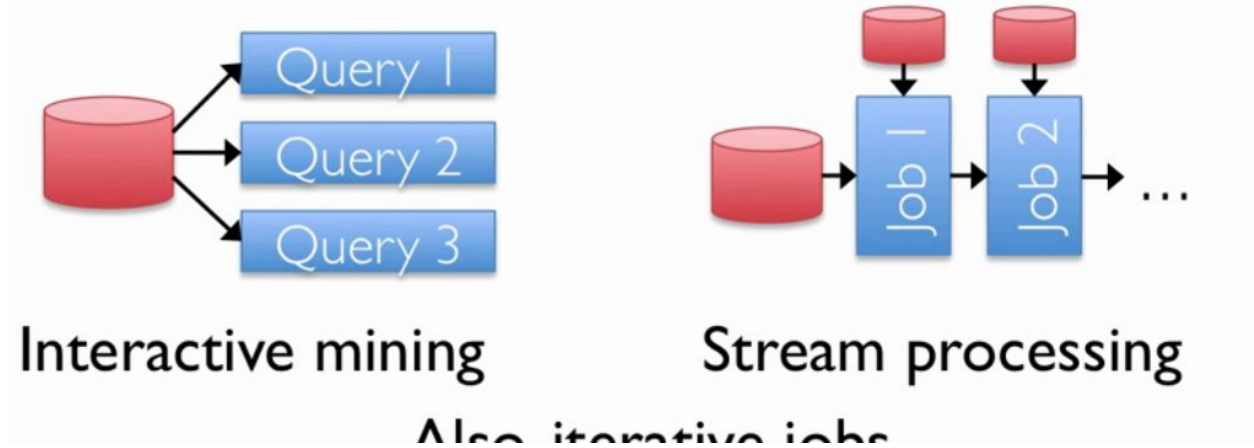
How to deal with failures
1 server fails every 3 year
Even worse: stragglers

Server failure: launch another task
Slow task: launch another task and terminate the slow task

Map reduce: distributed execution
Each stage passes through the hard drives

Motivation of Apache Spark

- Iterative jobs
It involves a lot of disk I/O for each repetition
Disk I/O is very slow.
- Using Map Reduce for complex jobs, interactive queries and online processing involves lots of disk I/O

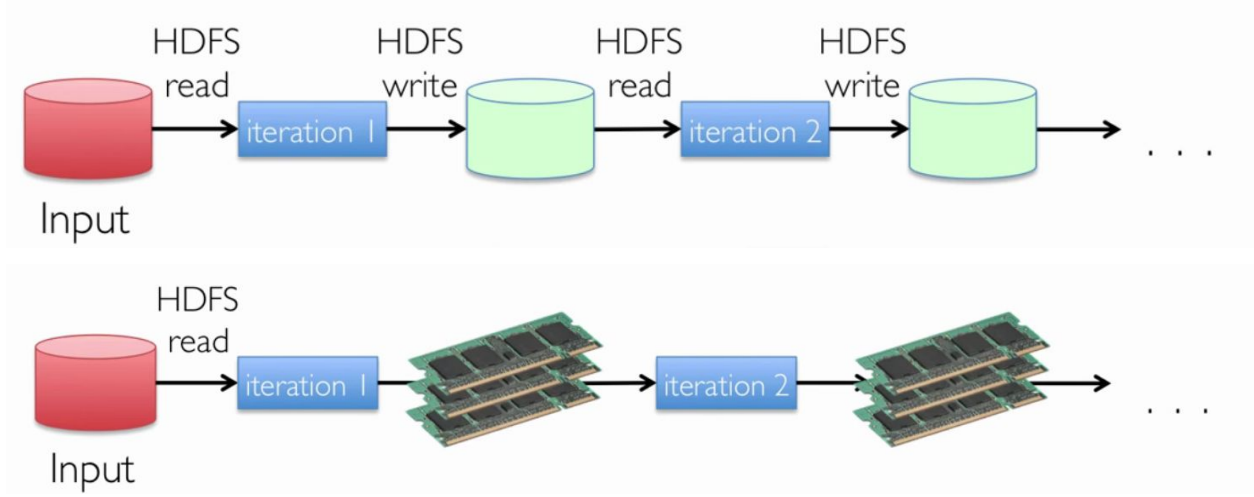


Cost of memory decreases annually
Lower cost means can put more memory in each server

Modern hardware for big data
lots of hard drives, CPUs and memory

Keep more data in-memory
comparison in in-memroy data sharing
10-100x faster than network and disk

Spark and map reduce differences

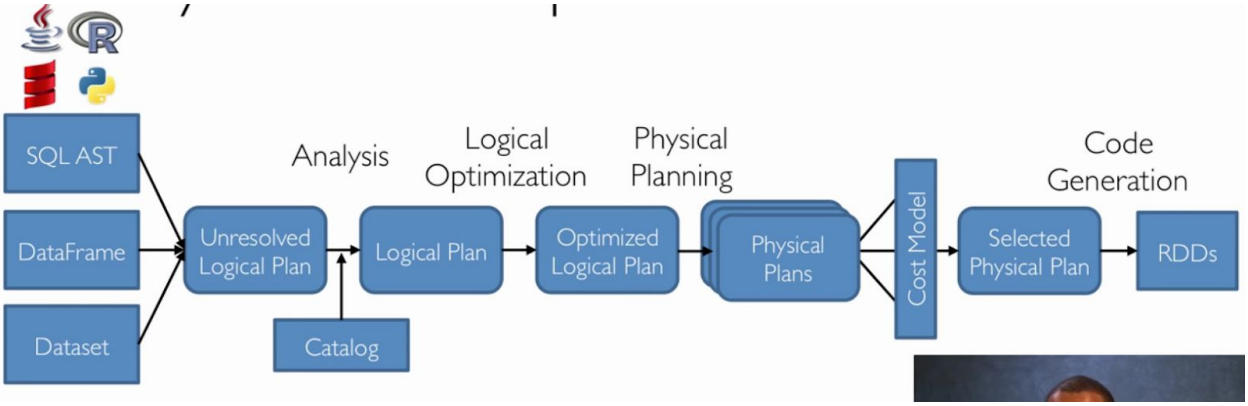


Advantage
- generalized patterns for computation

provide unified engine for man use cases
require 2-5x less code
- Lazy evaluation of the lineage graph
can optimize, reduce wait states, pipeline better
- Lower overhead for starting jobs
- Less expensive shuffles

	Apache Hadoop Map Reduce	Apache Spark
Storage	Disk only	In-memory or on disk
Operations	Map and Reduce	Many transformation and actions, including Map and Reduce
Execution model	Batch	Batch, interactive, streaming
Languages	Java	Scala, Java, R, and Python

Catalyst: shared optimization and execution



Java virtual machine object overhead
example
abcd: 4 bytes with UTF-8 encoding, 48 bytes in Java
Project Tungsten's compact encoding to compress Java objects

Apache Spark is often faster than a traditional Hadoop/MapReduce implementation because Apache Spark keeps results in memory so they do not need to be serialized (converted into a format that can be stored on disk) and they do not need to be written to disk.

Relational data model
A relational data model is the most used data model
- Relation, a table with rows and columns
- Every relation has a schema defining fields in columns

Two parts to a Relation:
Schema: specifies name of relation, plus each column's name and type
Instance: the actual data at a given time
#rows=cardinality #fields=degree

Database
A large organized collection of data

SQL Structured Query Language
Some functionality SQL provides:
- Create, modify, delete relations
- Add, modify, remove tuples
- Specify queries to find tuples matching criterion

Join
```
SELECT S.name, E.cid
FROM Students S, Enrolled E
WHERE S.sid=E.sid
```

Cross Join E x S

Explicit SQL Joins (preferred)
```
SELECT S.name, E.cid
FROM Student S INNER JOIN Enrolled E on S.sid = E.sid
```

Implicit join
```
SELECT S.name, E.cid
FROM Student S JOIN Enrolled E on S.sid = E.sid
```

Left outer join
```
SELECT S.name, E.cid
FROM Student S LEFT OUTER JOIN Enrolled E on S.sid = E.sid
```

Right outer join
```
SELECT S.name, E.cid
FROM Student S RIGHT OUTER JOIN Enrolled E on S.sid = E.sid
```

Spark supports join
join(other, on=None, how=None)
e.g. df.join(df2,!'name').select(df.name,!df2.height)
df.join(df2, df.name == df2.name, 'outer').select(df.name, df2.height).collect()