Spark – Day 1

Introduction to Spark

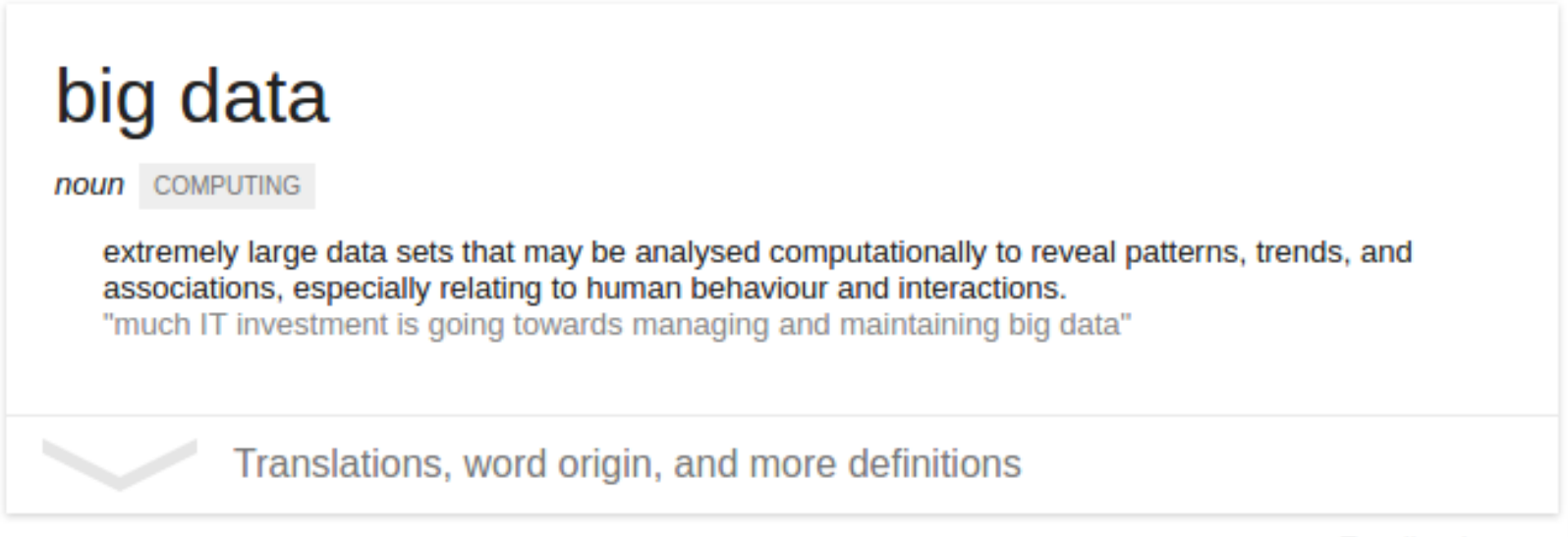
Agenda

- Big Data
- Spark Introduction
- RDDs

Agenda

- Big Data
- Spark Introduction
- RDDs

What is Big Data?



big data

noun COMPUTING

extremely large data sets that may be analysed computationally to reveal patterns, trends, and associations, especially relating to human behaviour and interactions.
"much IT investment is going towards managing and maintaining big data"

Translations, word origin, and more definitions

What is Big Data?

- Big data is a term for data sets that are so large or complex that traditional data processing applications are inadequate.
- Challenges include analysis, capture, data curation, search, sharing, storage, transfer, visualization, querying, updating and information privacy.
- The term often refers simply to the use of predictive analytics or certain other advanced methods to extract value from data.
- Accuracy in big data may lead to more confident decision making.

Where?

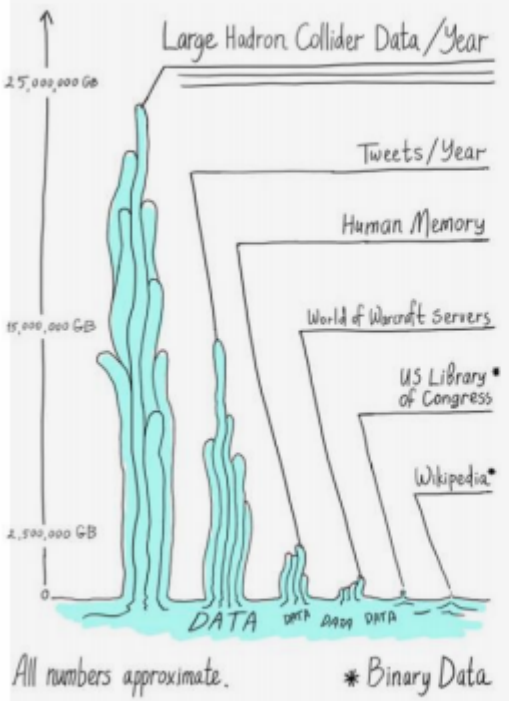
- It's all happening online – could record every:
  - » Click
  - » Ad impression
  - » Billing event
  - » Fast Forward, pause,…
  - » Server request
  - » Transaction
  - » Network message
  - » Fault
  - » …

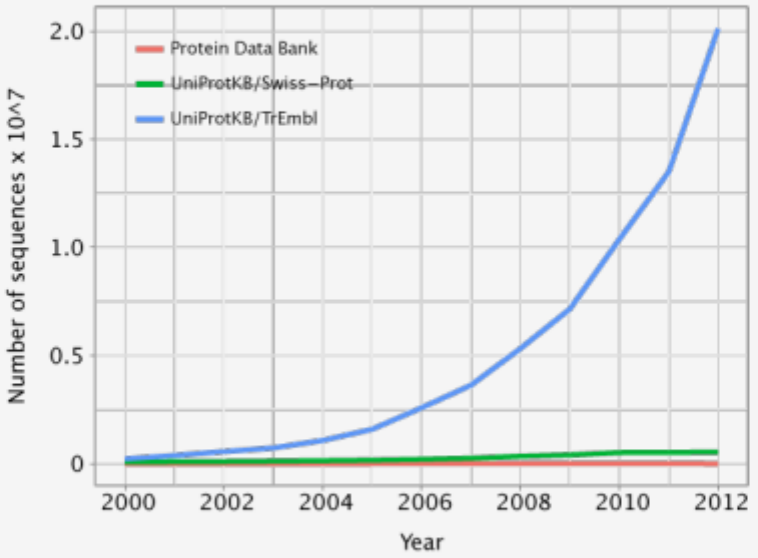
Where?

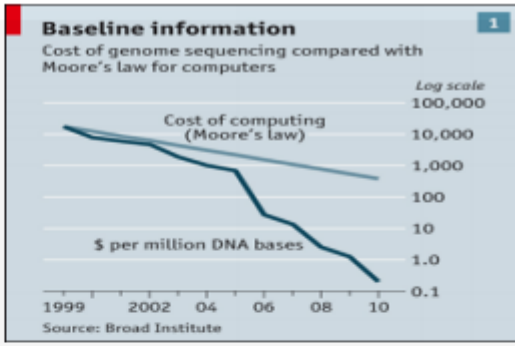
- User Generated Content (Web & Mobile)
  - » Facebook
  - » Instagram
  - » Yelp
  - » TripAdvisor
  - » Twitter
  - » YouTube
  - » …

Where?

- Health and Scientific Computing



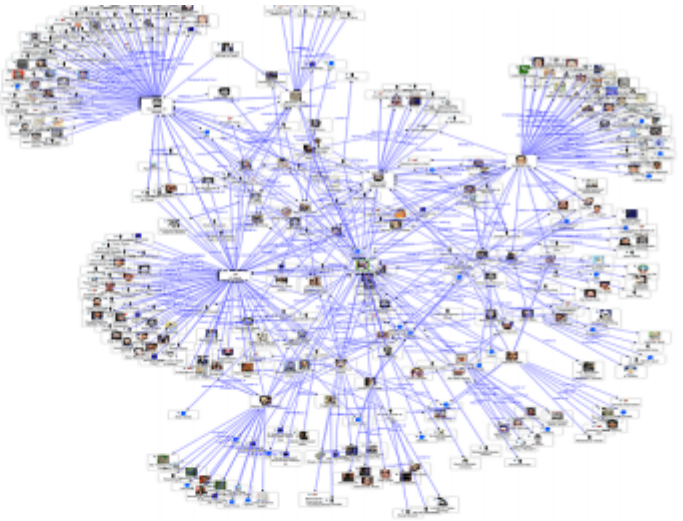




Images: http://www.economist.com/node/16349358
http://gorbi.irb.hr/en/method/growth-of-sequence-databases/
http://www.symmetrymagazine.org/article/august-2012/particle-physics-tames-big-data

Where?

Graph Data



Lots of interesting data has a graph structure:

- Social networks
- Telecommunication Networks
- Computer Networks
- Road networks
- Collaborations/Relationships
- …

Some of these graphs can get quite large
(e.g., Facebook user graph)

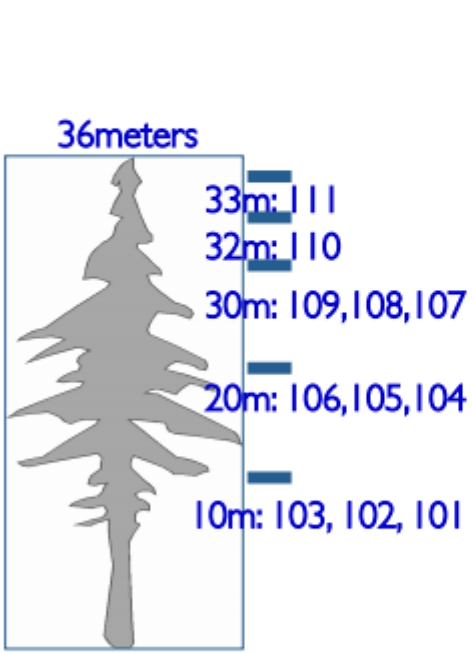
Where?

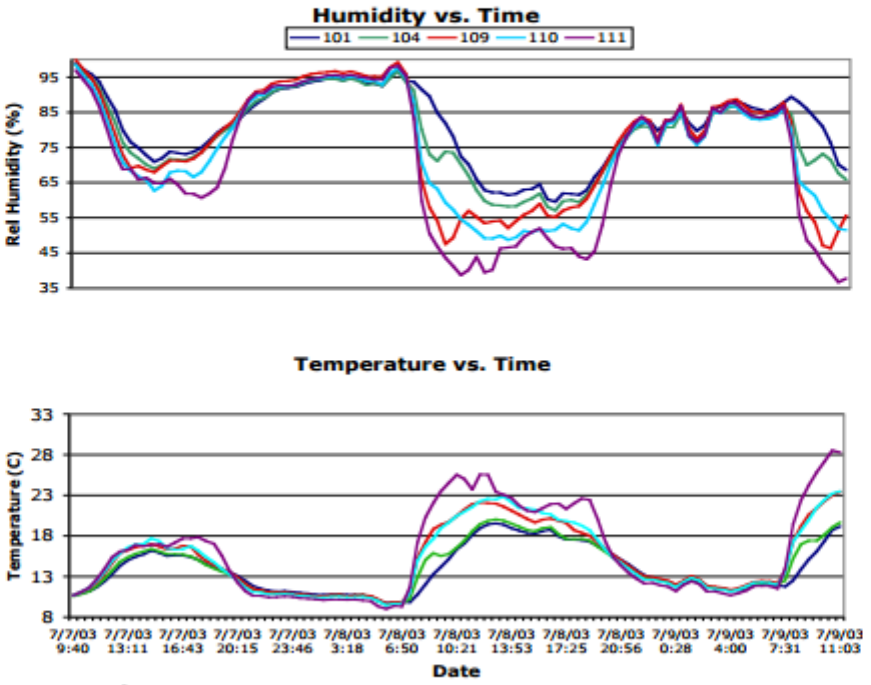
Log Files – Apache Web Server Log

```
uplherc.upl.com - - [01/Aug/1995:00:00:07 -0400] "GET / HTTP/1.0" 304 0
uplherc.upl.com - - [01/Aug/1995:00:00:08 -0400] "GET /images/ksclogo-medium.gif
HTTP/1.0" 304 0
uplherc.upl.com - - [01/Aug/1995:00:00:08 -0400] "GET /images/MOSAIC-logosmall.gif
HTTP/1.0" 304 0
uplherc.upl.com - - [01/Aug/1995:00:00:08 -0400] "GET /images/USA-logosmall.gif HTTP/
1.0" 304 0
ix-esc-ca2-07.ix.netcom.com - - [01/Aug/1995:00:00:09 -0400] "GET /images/launch-
logo.gif HTTP/1.0" 200 1713
uplherc.upl.com - - [01/Aug/1995:00:00:10 -0400] "GET /images/WORLD-logosmall.gif
HTTP/1.0" 304 0
slppp6.intermind.net - - [01/Aug/1995:00:00:10 -0400] "GET /history/skylab/
skylab.html HTTP/1.0" 200 1687
piweba4y.prodigy.com - - [01/Aug/1995:00:00:10 -0400] "GET /images/launchmedium.gif
HTTP/1.0" 200 11853
tampico.usc.edu - - [14/Aug/1995:22:57:13 -0400] "GET /welcome.html HTTP/1.0" 200 790
```

Where?

Internet of Things: Example Measurements





Redwood tree humidity and temperature at various heights

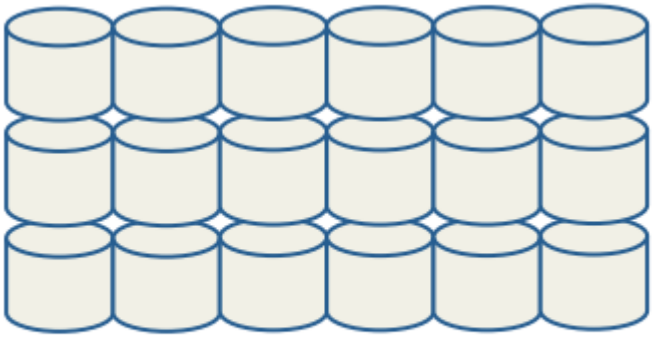
Truth

- Facebook's daily logs: 60 TB
- 1,000 genomes project: 200 TB
- Google web index: 10+ PB
- Cost of 1 TB of disk: ~$35
- Time to read 1 TB from disk: 3 hours (100 MB/s)

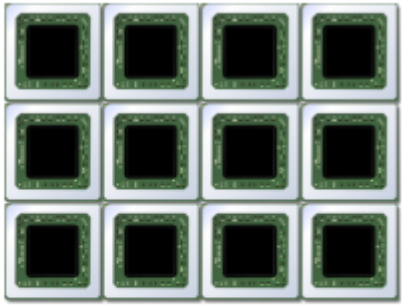
All this data cannot fit into a single machine

We need ways to distribute data over large clusters

Hardware for Big Data



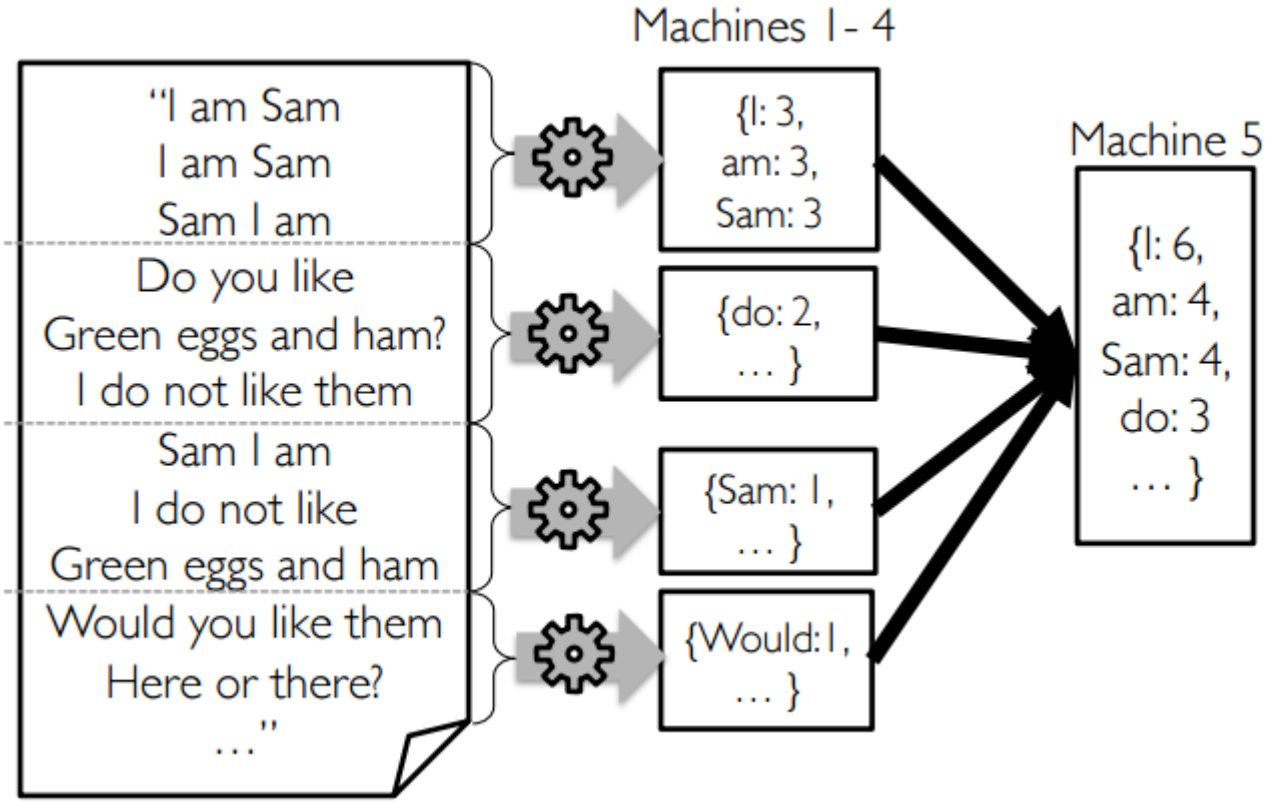
Lots of hard drives



… and CPUs

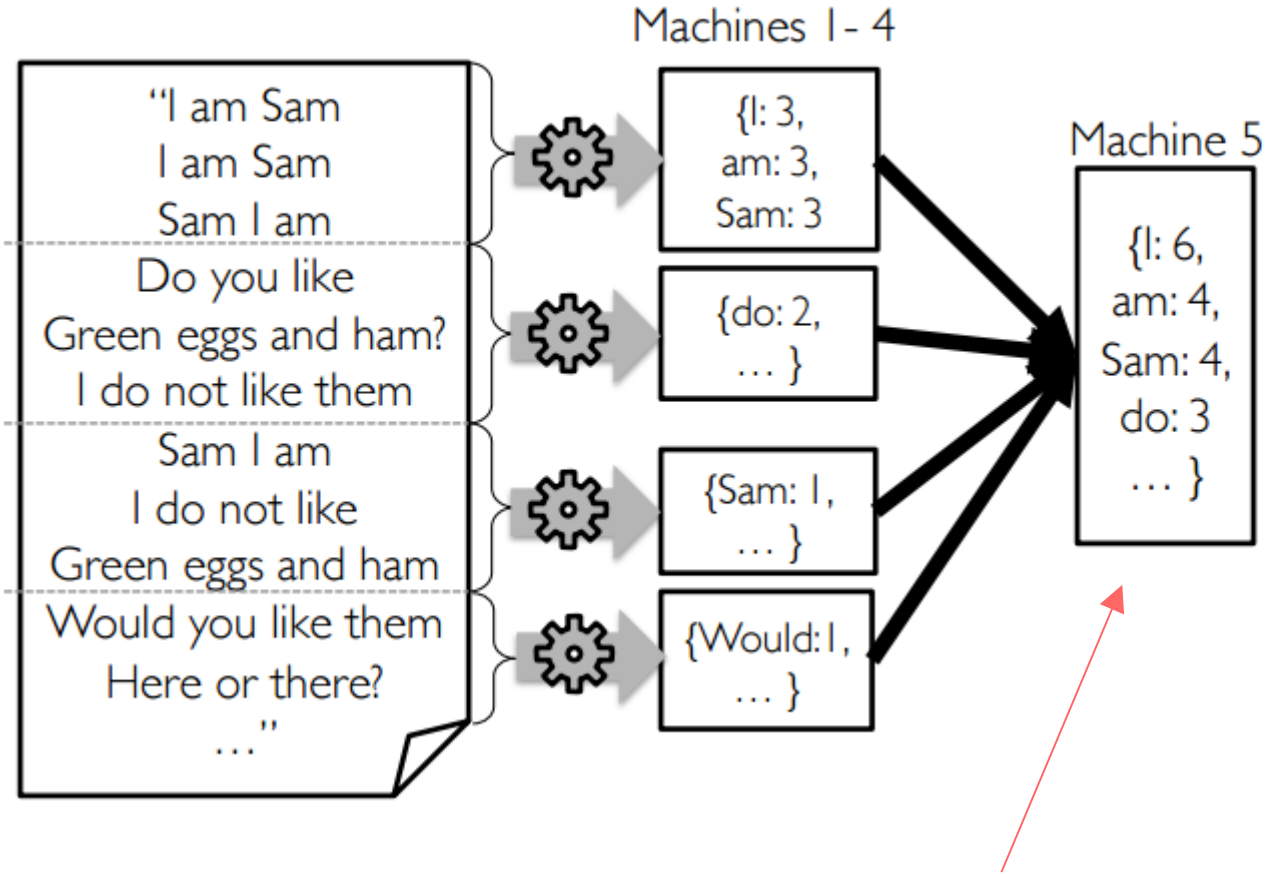
Consumer grade hardware

Solving for Big Data



What's the problem with this approach?

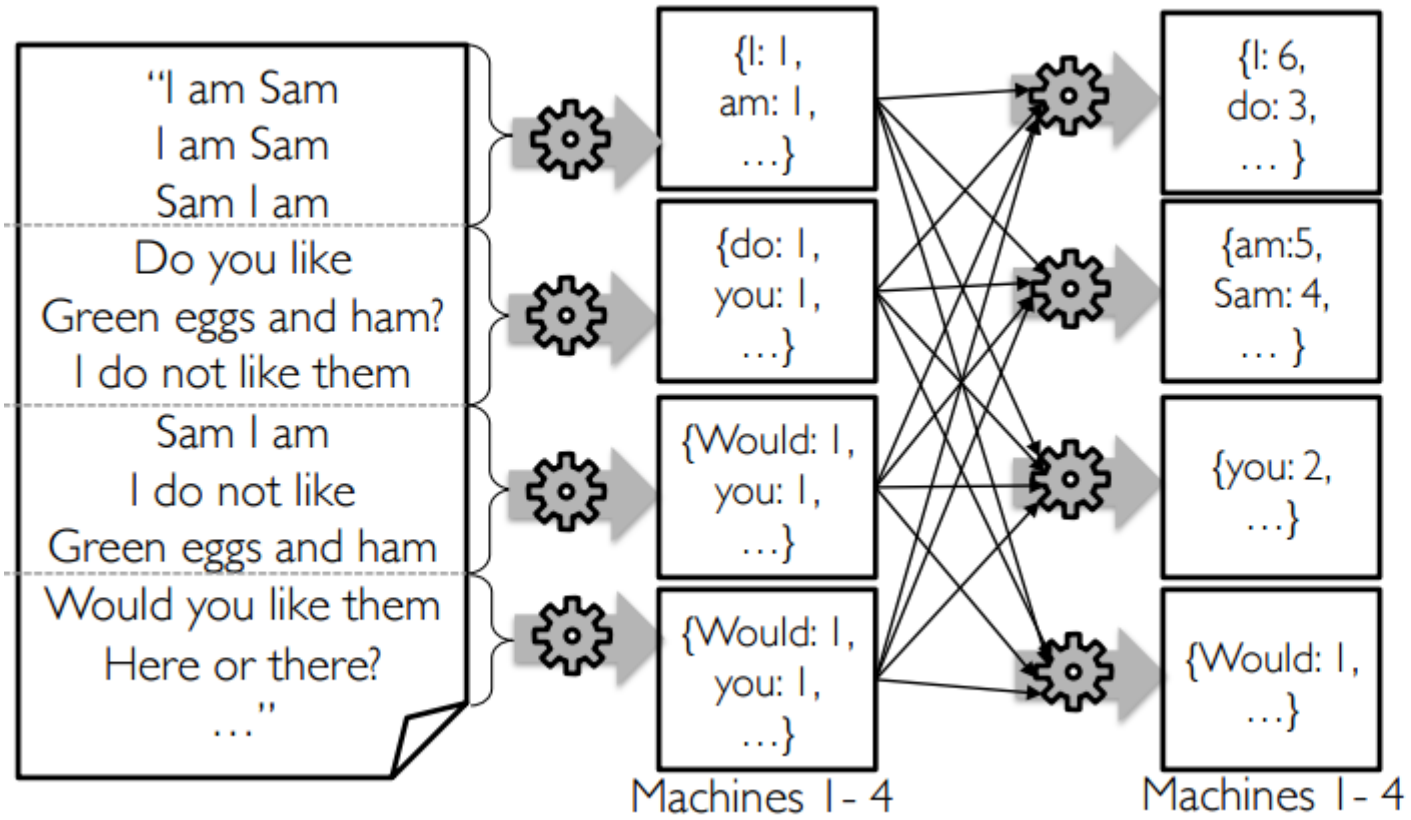
Solving for Big Data



What's the problem with this approach?

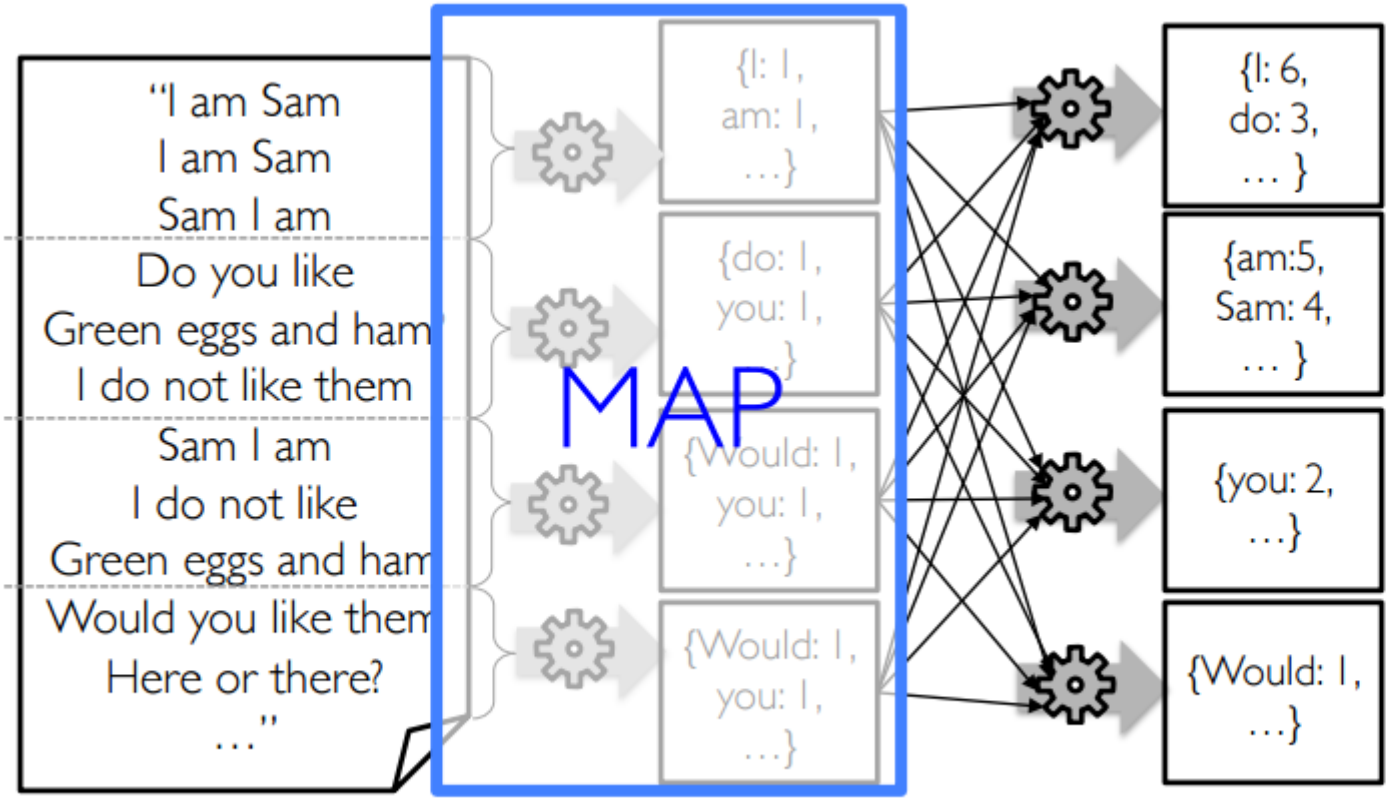
Results need to fit in a single machine

Solving for Big Data



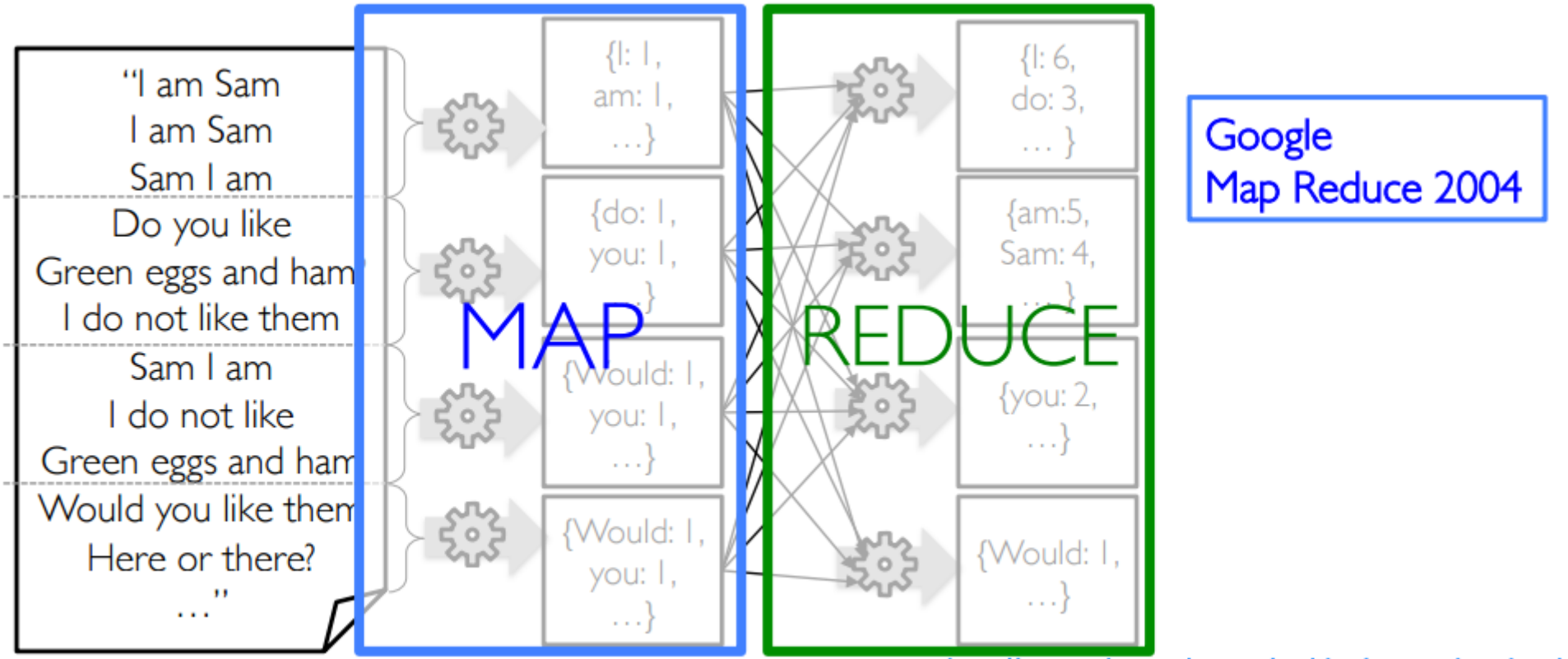
Use Divide and Conquer!!

Solving for Big Data

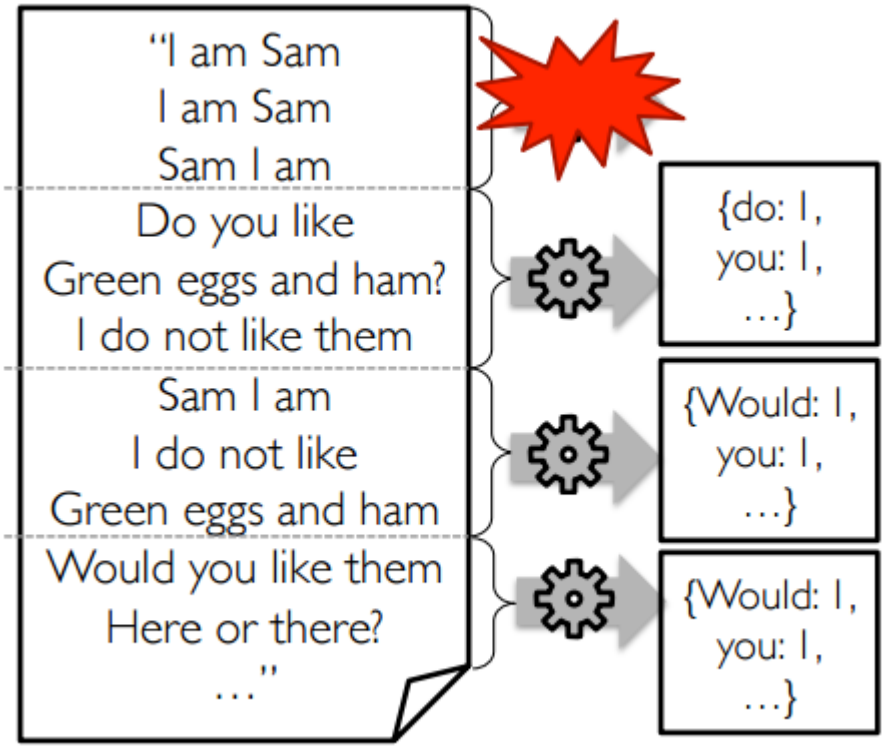


Use Divide and Conquer!!

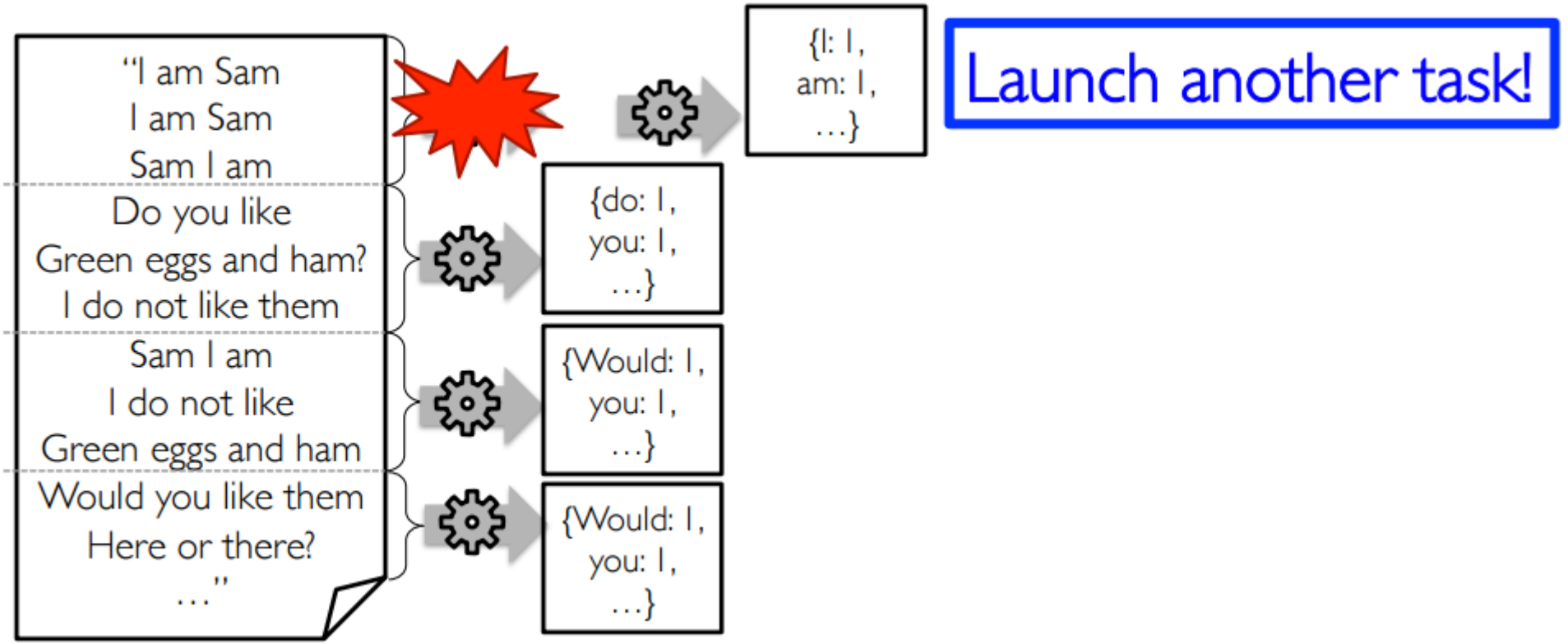
Solving for Big Data



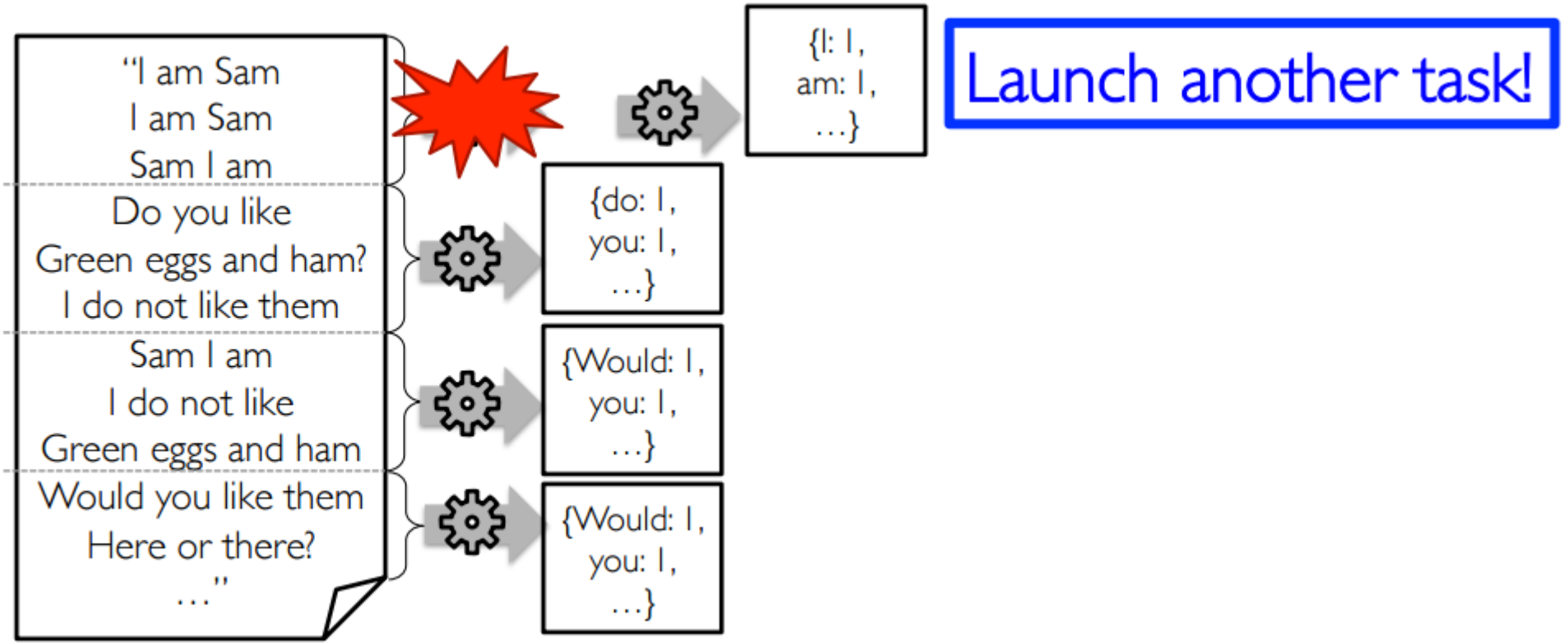
Dealing with Failures



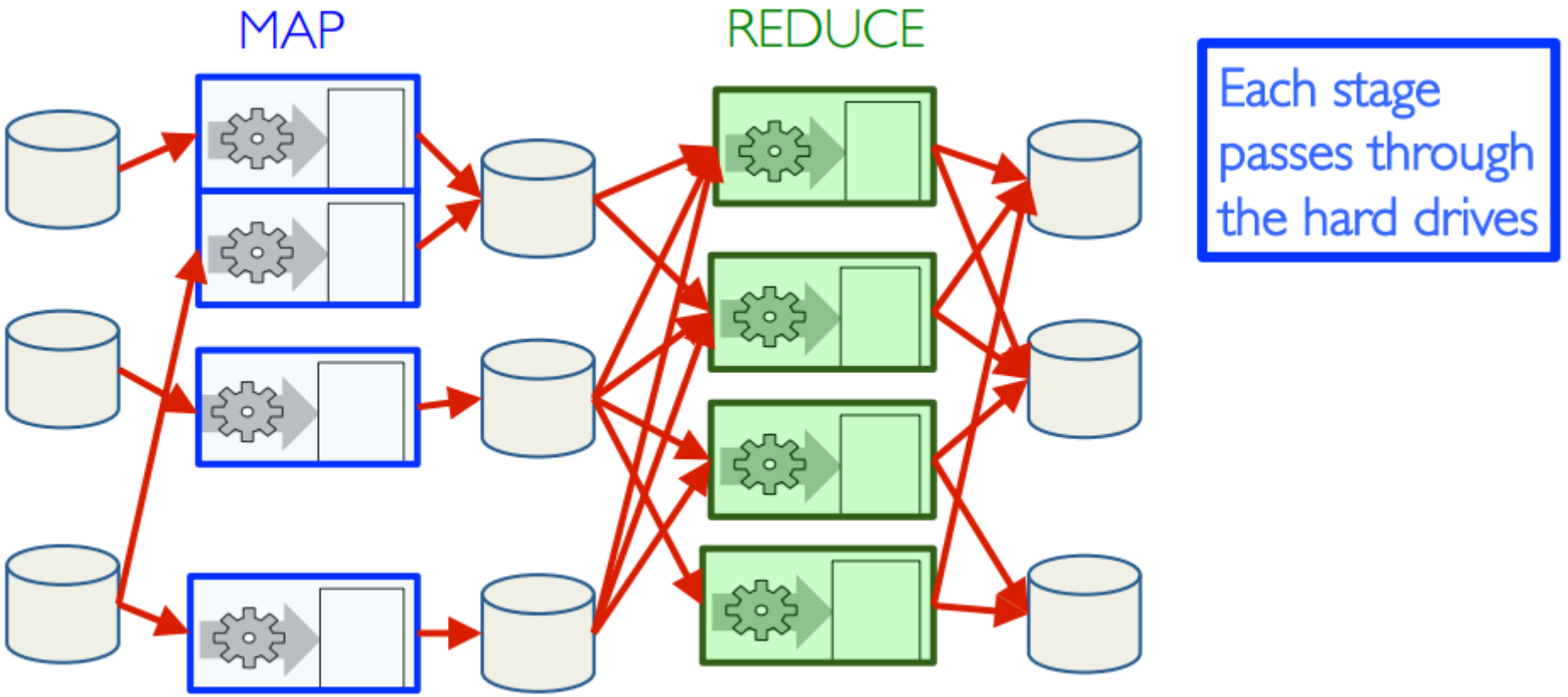
Dealing with Failures



Dealing with Failures

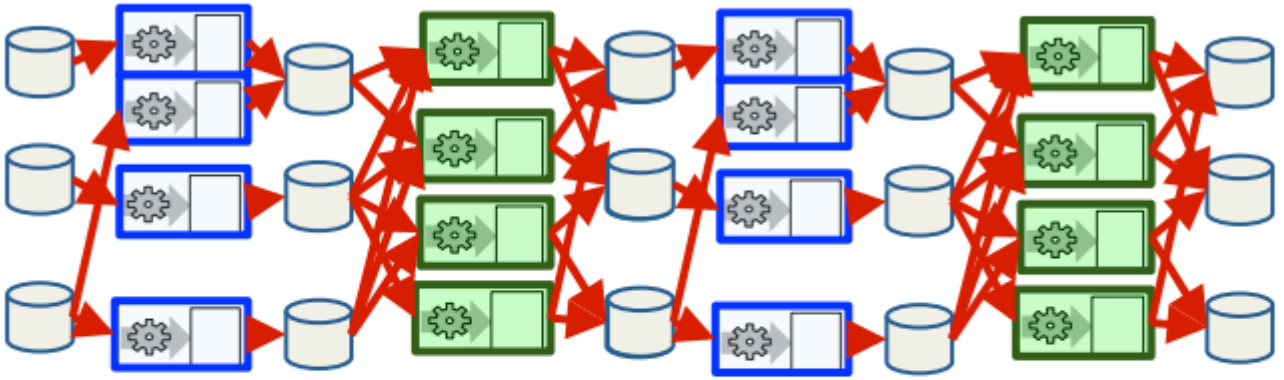


Problem with Map Reduce

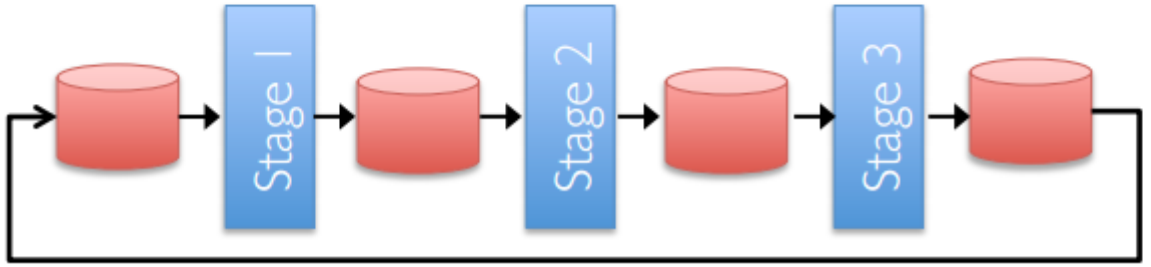


Problem with Map Reduce

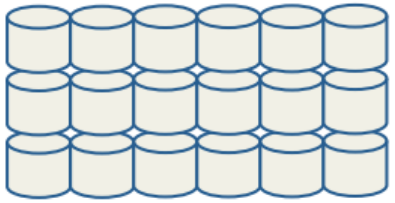
- Iterative jobs involve a lot of disk I/O for each repetition



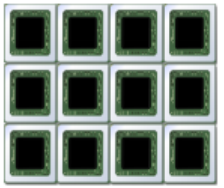
Disk I/O is very slow!



Spark introduces another player

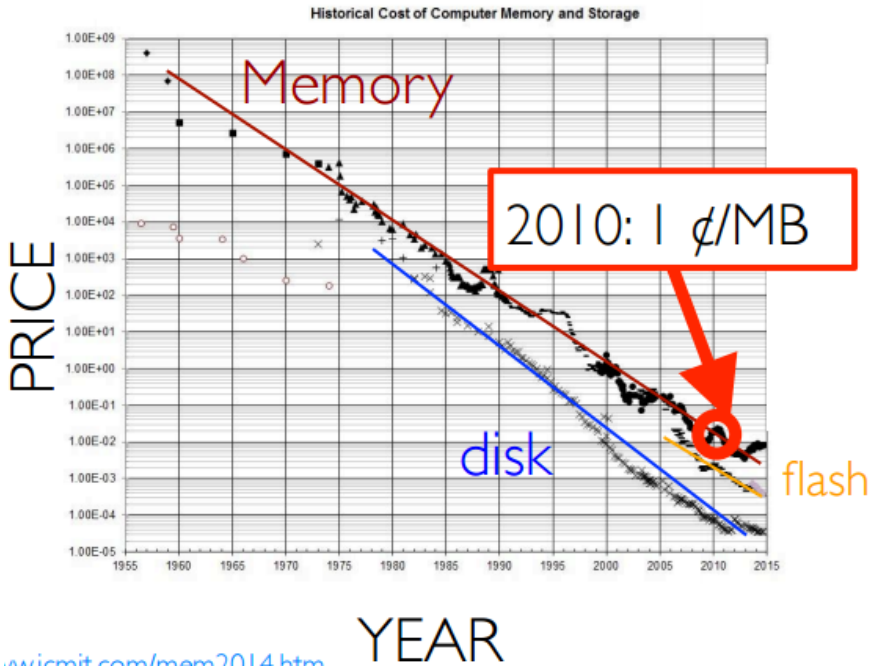


Lots of hard drives



… and CPUs

… and memory!



Spark Opportunity

- Keep more data *in-memory*

- Create new distributed execution engine:

Agenda

- Big Data
- Spark Introduction
- RDDs

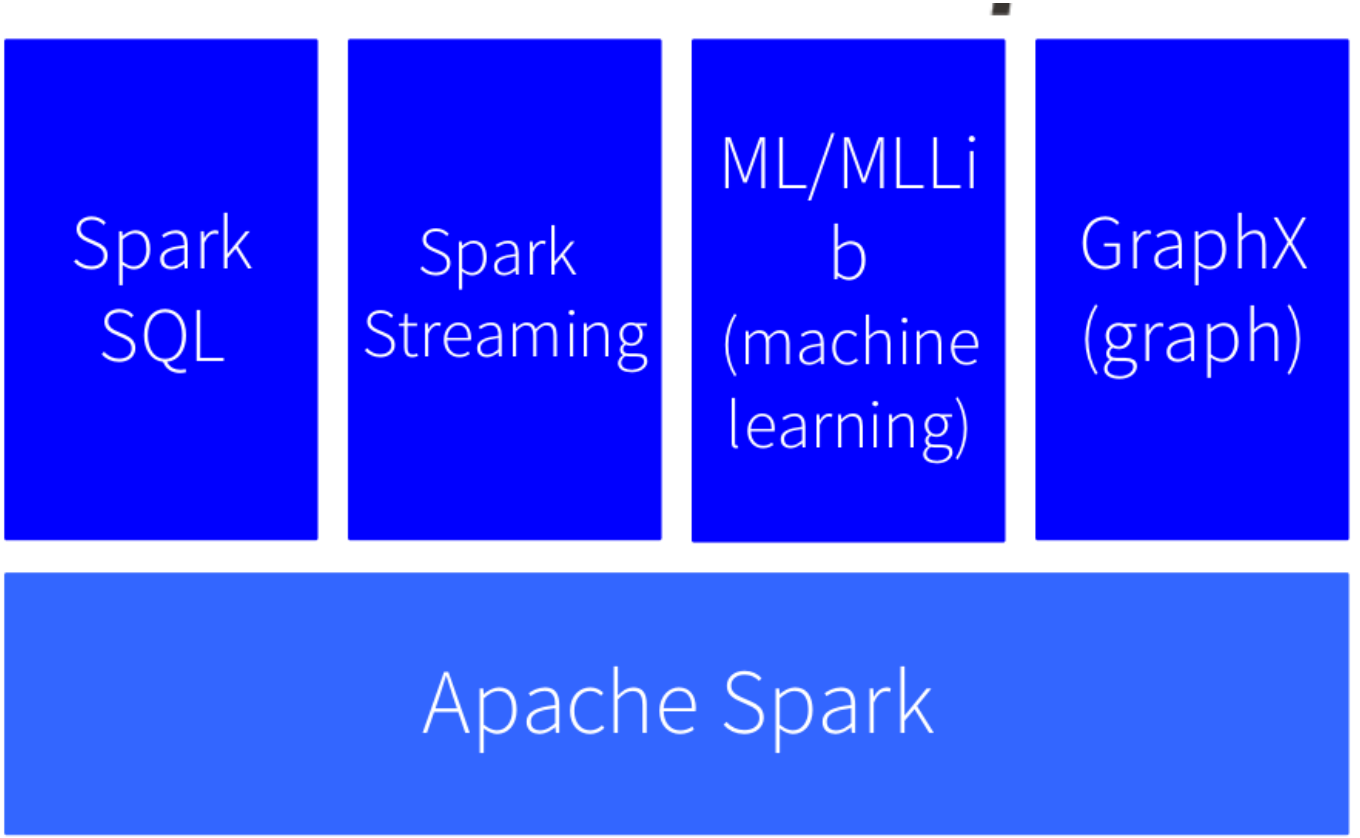
Spark Introduction

Apache Spark is a fast and general-purpose cluster computing system

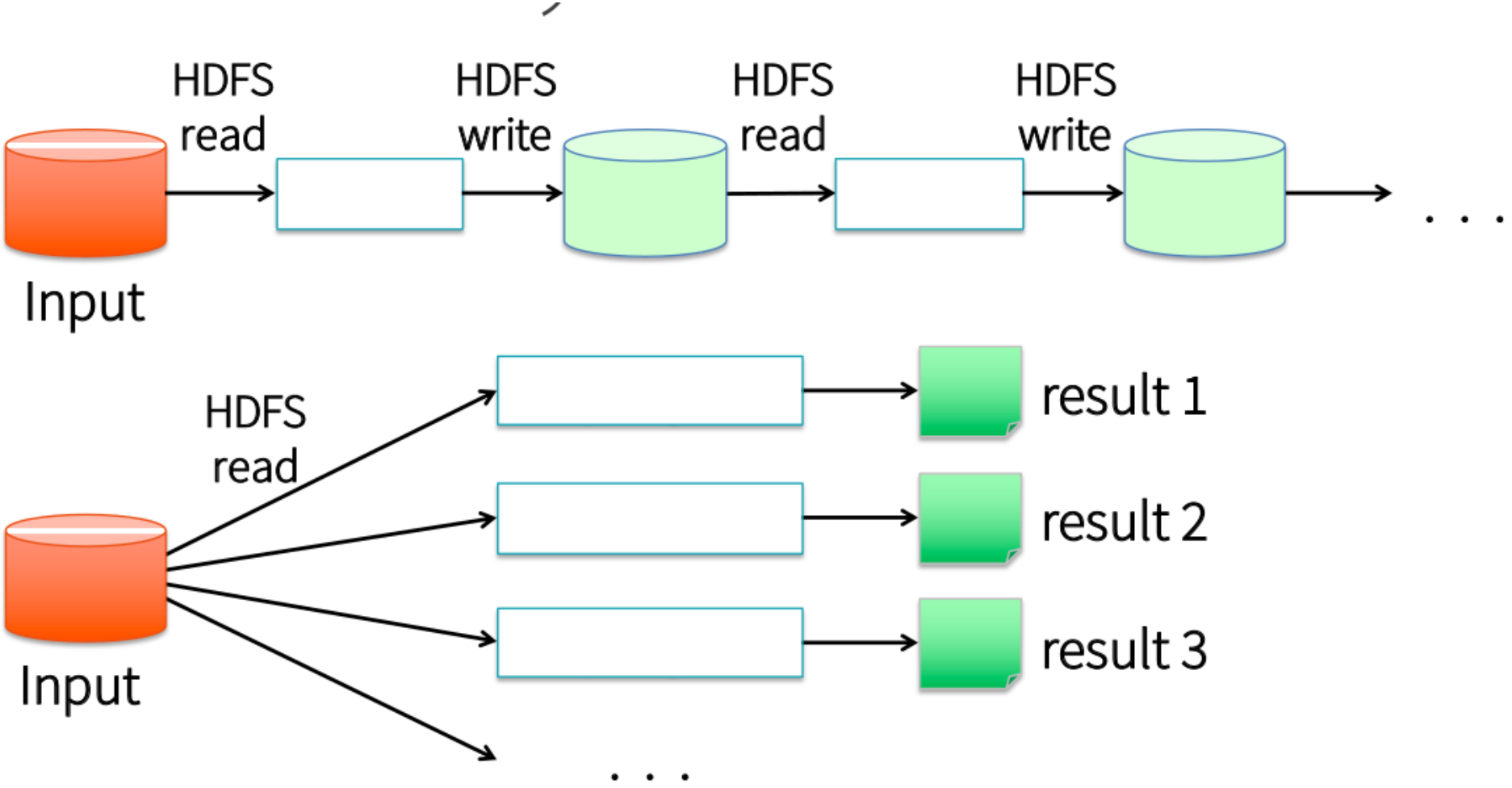
Spark Introduction

- Started as a research project at UC Berkeley in 2009
- Open Source License (Apache 2.0)
- Latest Stable Release: v2.1.1
- 500,000 lines of code (77% Scala)
- Built by 1100+ developers

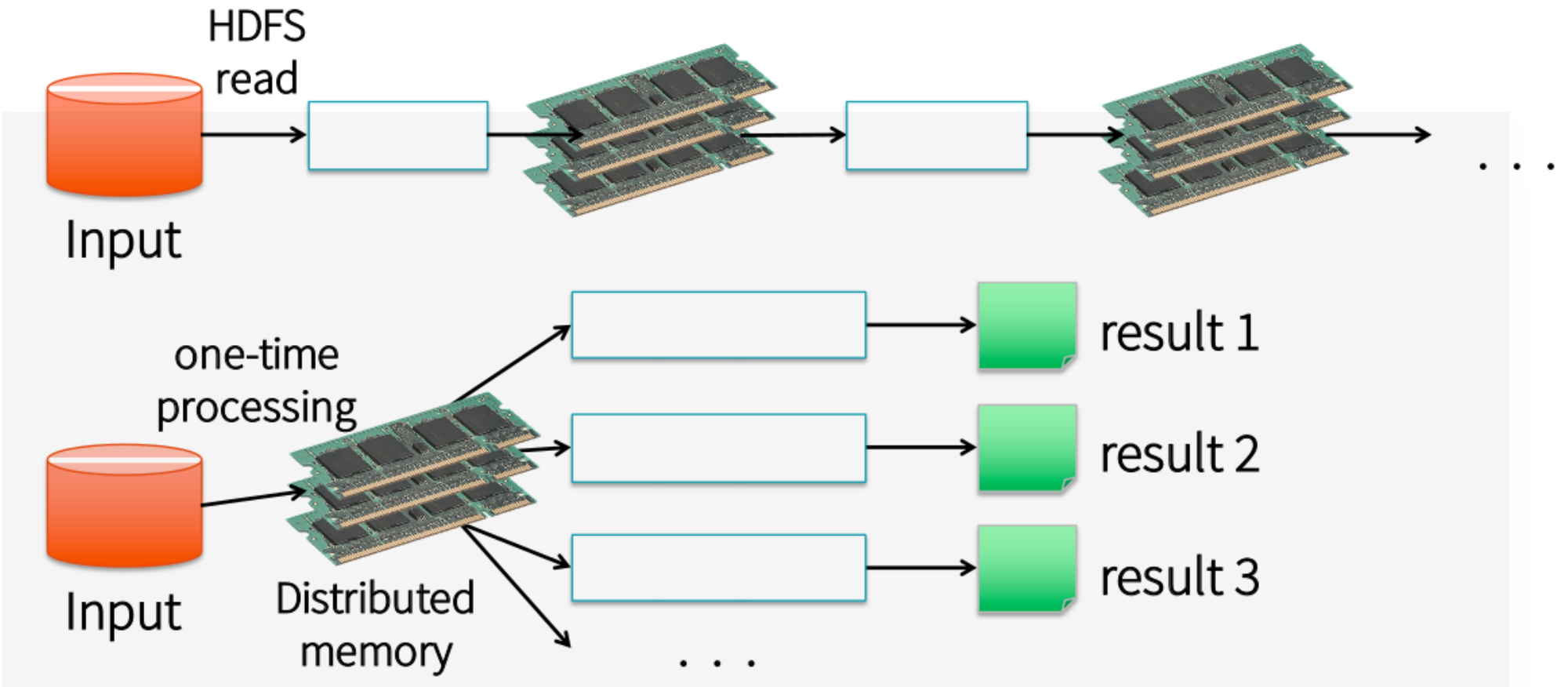
- Keep more data *in-memory*
- New distributed execution environment
- Bindings for:
  - Python, Java, Scala, R



Use memory instead of disk



In memory data sharing



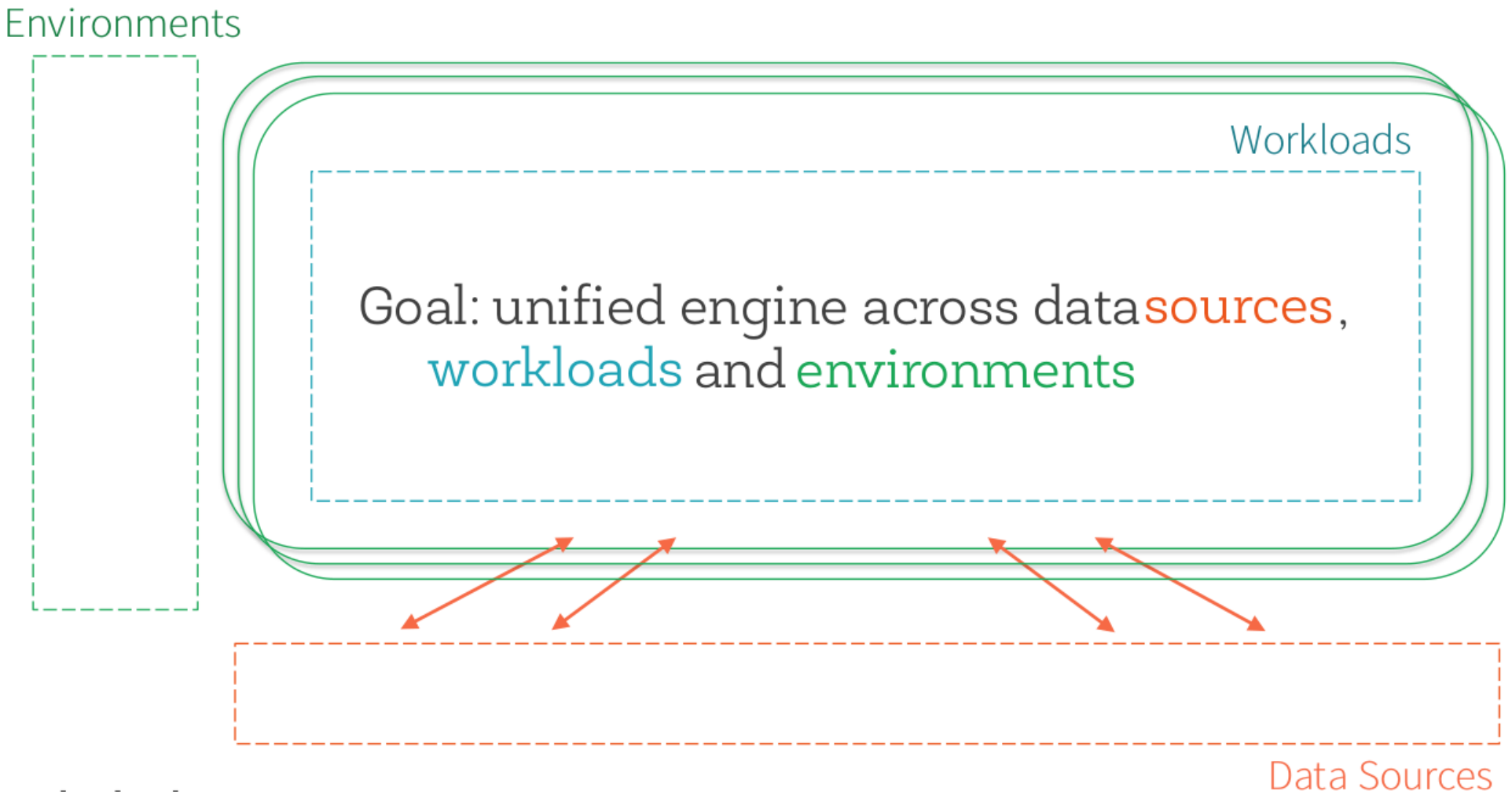
In-Memory huge difference

- Two iterative Machine Learning algorithms:

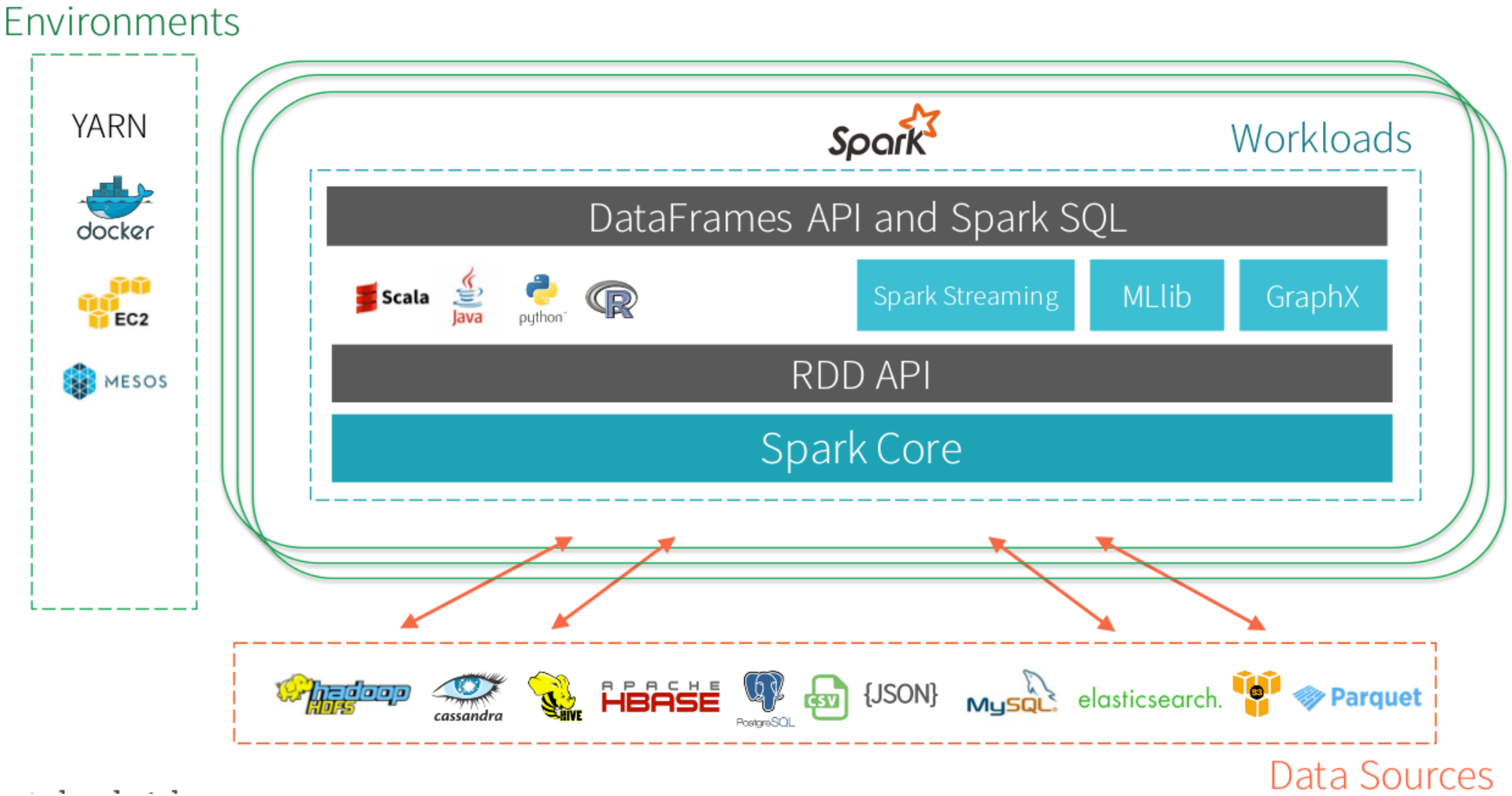
In memory data sharing

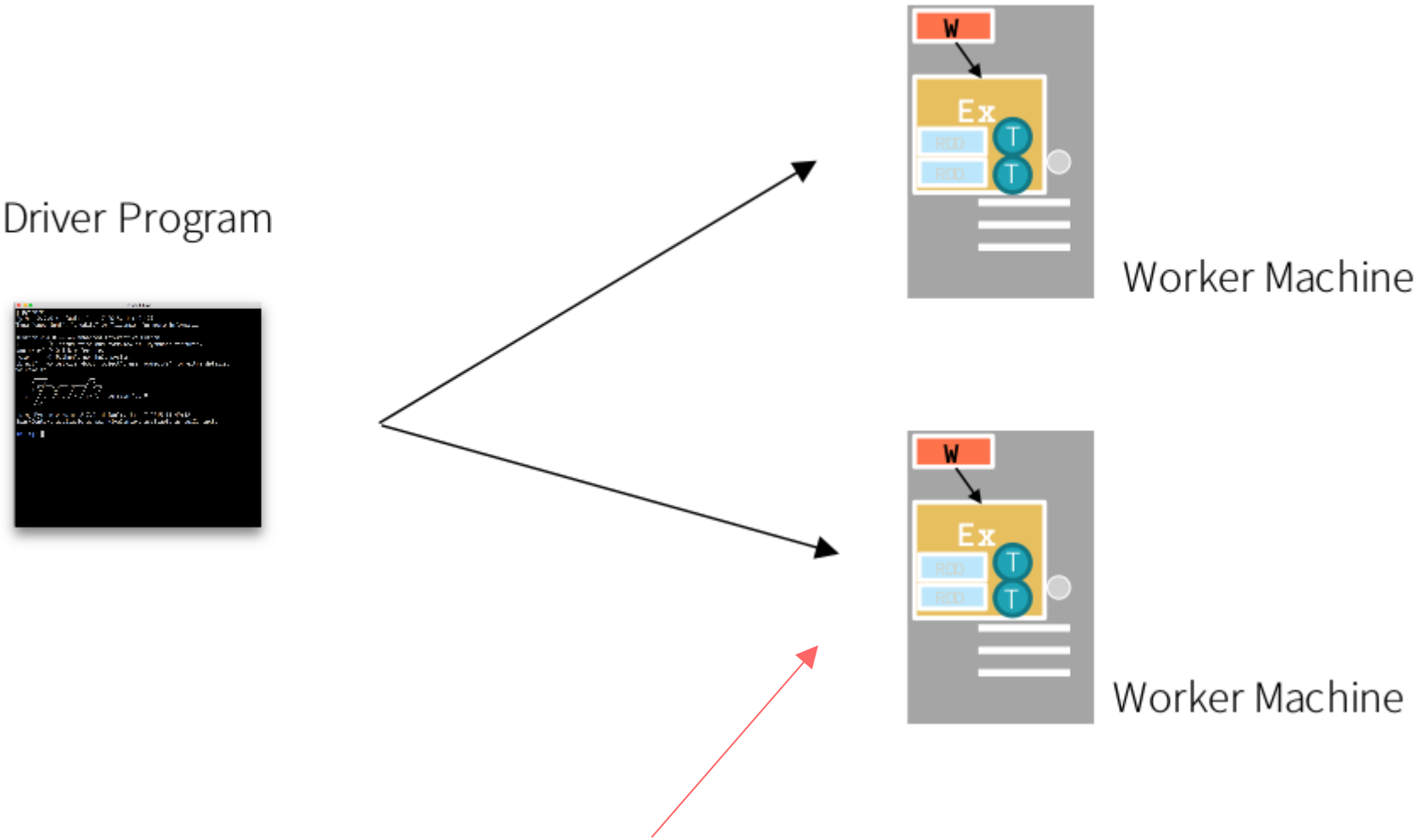
10-100x faster than network and disk

Goal

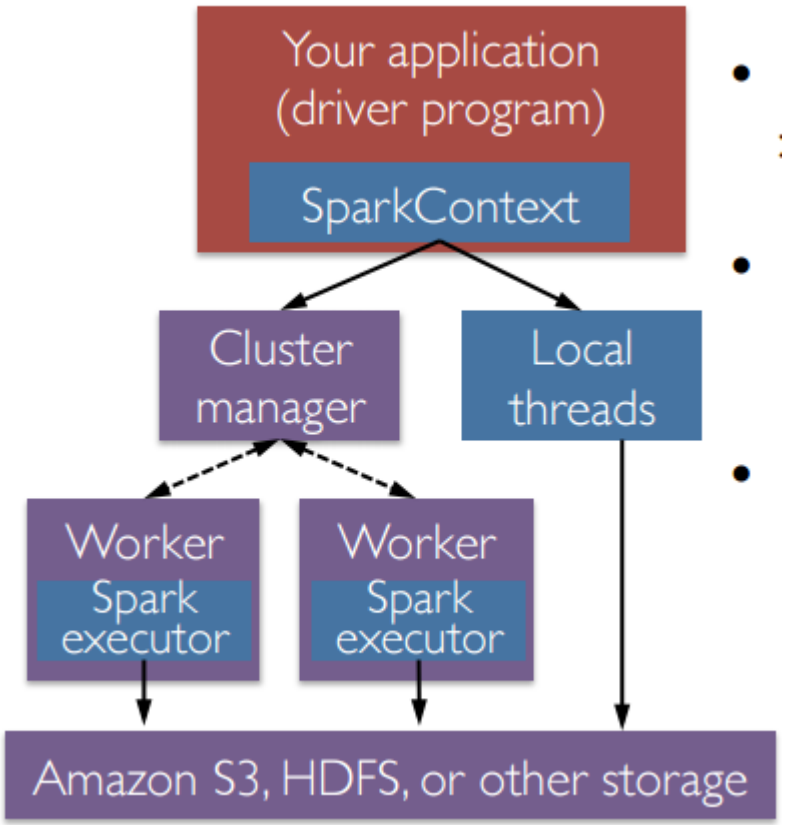

Environments
Workloads
Goal: unified engine across data sources, workloads and environments
Data Sources

Goal


Environments
YARN
docker
EC2
MESOS
Spark
Workloads
DataFrames API and Spark SQL
Scala
Java
python
Spark Streaming
MLlib
GraphX
RDD API
Spark Core
hadoop HDFS
cassandra
HIVE
APACHE HBASE
PostgreSQL
CSV
{JSON}
MySQL
elasticsearch.
Parquet
Data Sources



Worker programs run on cluster nodes or in local threads
RDDs are distributed across workers



- SparkContext tells Spark how & where to access the cluster
- Master determines the type and size of cluster

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)

Agenda

- Big Data
- Spark Introduction
- RDDs

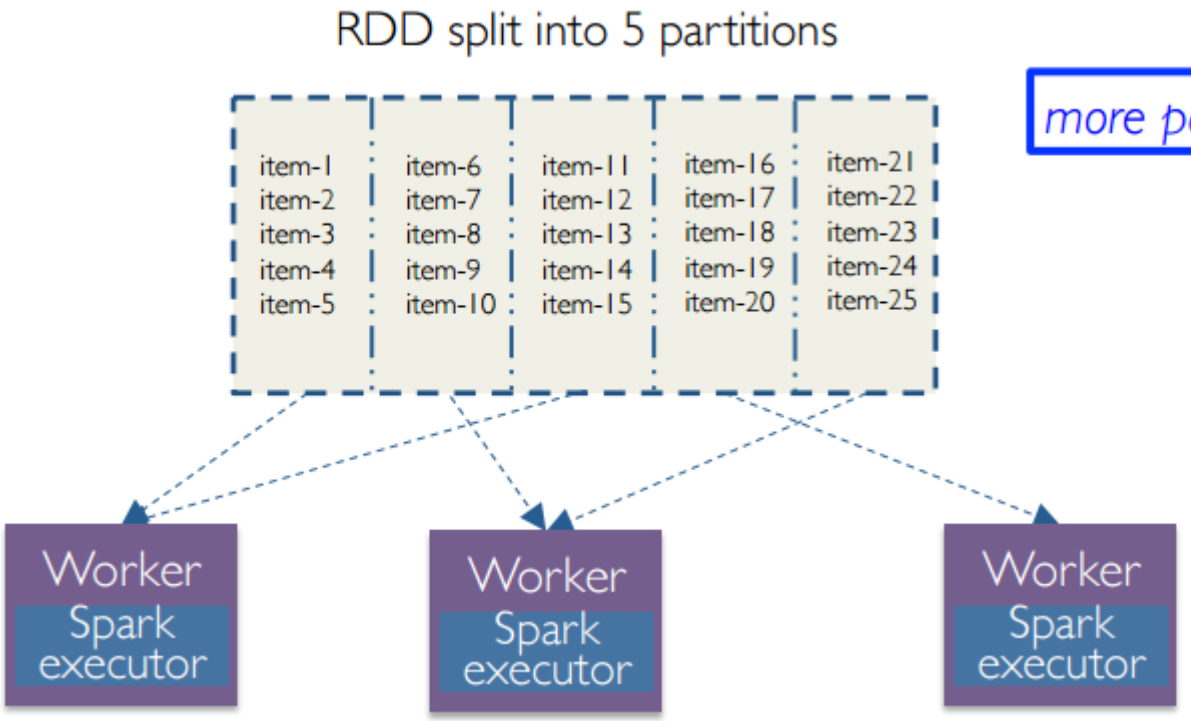
RDDs

- Resilient Distributed Dataset
- Immutable once created
- Collection of elements partitioned across nodes of a cluster which can be operated upon in parallel
- Can be persisted in memory thus allowing reuse across parallel operations
- Automatically recover from node failures

2 Ways to create RDDs

- Starting with a file in Hadoop File System, or any Hadoop supported file system
- Using existing Scala Collection in driver program and transforming it

Partitions



more partitions = more parallelism

Shared Variables

- Another abstraction after RDDs
- Sometimes variables need to be shared between tasks or between tasks and driver program
  - Broadcast – Used to cache a value in memory of all nodes
  - Accumulator – Variables which get added to like counters and sums

First Steps

Tells Spark how to access the cluster

```
import org.apache.spark.SparkContext
import org.apache.spark.SparkConf

val conf = new SparkConf().setAppName("BigApple").setMaster("local")
new SparkContext(conf)
```

SC needs conf object that has info about the project

Name of the app

Spark, Mesos or Yarn cluster URL
Or “local” for special mode

Using the Shell

- In the Spark shell, a special interpreter-aware SparkContext is already created for you, in the variable called sc. Making your own SparkContext will not work.

```
$ ./bin/spark-shell --master local[4]
```

Use exactly 4 cores

RDDs

- Fault tolerant collection of elements that can be operated on in parallel

Creating RDD

- Parallelized Collections

```
val conf = new SparkConf().setAppName("BigApple").setMaster("local")
val sc = new SparkContext(conf)

val data = Array(1, 2, 3, 4, 5)
val distData = sc.parallelize(data)

println(distData.reduce((a,b)=>a+b))
```

Once created RDD can
Be operated in parallel

Calling SparkContext's parallelize
method on a collection

Partitions

```
val distData = sc.parallelize(data,10)
```

- We can define # of partitions to cut the dataset into
- Spark would run one task per partition
- Typically 2-4 partitions per CPU in the cluster is a good idea
- Without the additional parameter Spark would automatically set the # of partitions based on the cluster

Creating RDD

- External Datasets – Any storage source supported by Hadoop

Reading the file

```
val distFile = sc.textFile("src/main/resources/ReadMe.txt")
val addFunc = (a:Int,b:Int)=>a+b
val sumOfLengthOfAllLines = distFile.map(x=>x.length).reduce(addFunc)
println(sumOfLengthOfAllLines)
```

Operating on the RDD

Working with files

- Files should be accessible to all nodes on the cluster at the same location. How?
- File input methods like textFile work on files, directories, compressed files and wildcards
  - textFile("/my/directory"), textFile("/my/directory/*.txt"), and textFile("/my/directory/*.gz")
- Like collections, we can give # of partitions of the file. By default there is one partition for each block. Block default is 64MB

Other Data formats

- Apart from text files, Spark's Scala/Java API also supports several other data formats:
  - SparkContext.wholeTextFiles lets you read a directory containing multiple small text files, and returns each of them as (filename, content) pairs. This is in contrast with textFile, which would return one record per line in each file.
  - For other Hadoop InputFormats, you can use the SparkContext.hadoopRDD method, which takes an arbitrary JobConf and input format class, key class and value class. Set these the same way you would for a Hadoop job with your input source. You can also use SparkContext.newAPIHadoopRDD for InputFormats based on the "new" MapReduce API (org.apache.hadoop.mapreduce).
  - RDD.saveAsObjectFile and SparkContext.objectFile support saving an RDD in a simple format consisting of serialized Java objects. While this is not as efficient as specialized formats like Avro, it offers an easy way to save any RDD.

RDD Operations

- Transformations – create a new dataset from existing
- Actions – return value to driver after running computation on the dataset

What is map? What is reduce?

RDD Operations

- All transformations are lazy. They do not compute results but just remember the base dataset on which they are applied.
- Compute happens only when result needs to be returned to driver program
- Transformations are recomputed every time action is called on it
- RDDs can be persisted in memory. Spark keeps elements in cluster for faster access next time.
- RDD can be persisted on disk or replicated across nodes

RDD Basics

```
val lines = sc.textFile("data.txt")
val lineLengths = lines.map(s => s.length)
val totalLength = lineLengths.reduce((a, b) => a + b)
```

Is the dataset loaded to memory?

RDD Basics

```
val lines = sc.textFile("data.txt")
val lineLengths = lines.map(s => s.length)
val totalLength = lineLengths.reduce((a, b) => a + b)
```

No, it is not

Is the compute done?

RDD Basics

```
val lines = sc.textFile("data.txt")
val lineLengths = lines.map(s => s.length)
val totalLength = lineLengths.reduce((a, b) => a + b)
```

No, it is not

No, it is lazy

What happens here?

RDD Basics

```
val lines = sc.textFile("data.txt")
val lineLengths = lines.map(s => s.length)
val totalLength = lineLengths.reduce((a, b) => a + b)
```

No, it is not

No, it is lazy

Reduce, is an action

Spark breaks the computation into tasks to run on separate machines, and each machine runs both its part of the map and a local reduction, returning only its answer to the driver program.

RDD Basics

```
val lines = sc.textFile("data.txt")
val lineLengths = lines.map(s => s.length)
val totalLength = lineLengths.reduce((a, b) => a + b)
```

No, it is not

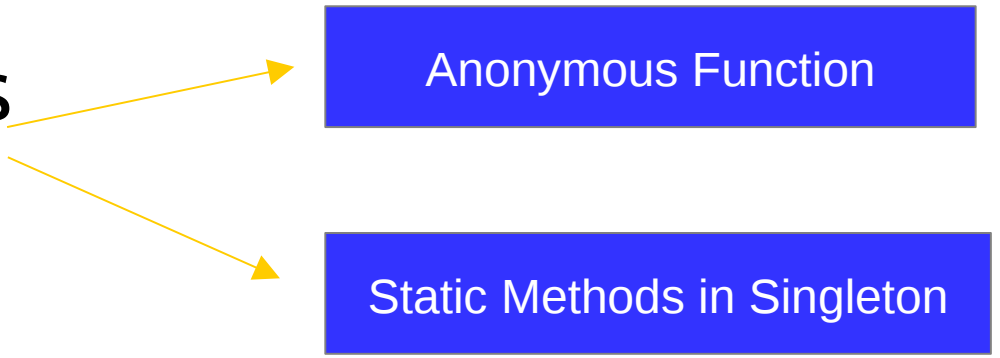
No, it is lazy

Where would you put lineLengths.persist() ?

Passing Functions to Spark

- Functions are passed in the driver program to run on the cluster

- 2 ways



Passing Functions to Spark

```
val distFile = sc.textFile("src/main/resources/compressed.gz")
val linesRDD = distFile.map(x => x.length)
val yaSumOfLines = linesRDD.reduce((a,b)=>a+b)
val sumOfLines = linesRDD.reduce(FunctionHolder.addFunc)
println(sumOfLines,yaSumOfLines)

object FunctionHolder {
  val addFunc = (a: Int, b: Int) => a + b
}
```

Anonymous function

Static method in a singleton

Exercise

- Write a function to multiply all the lengths of the lines