

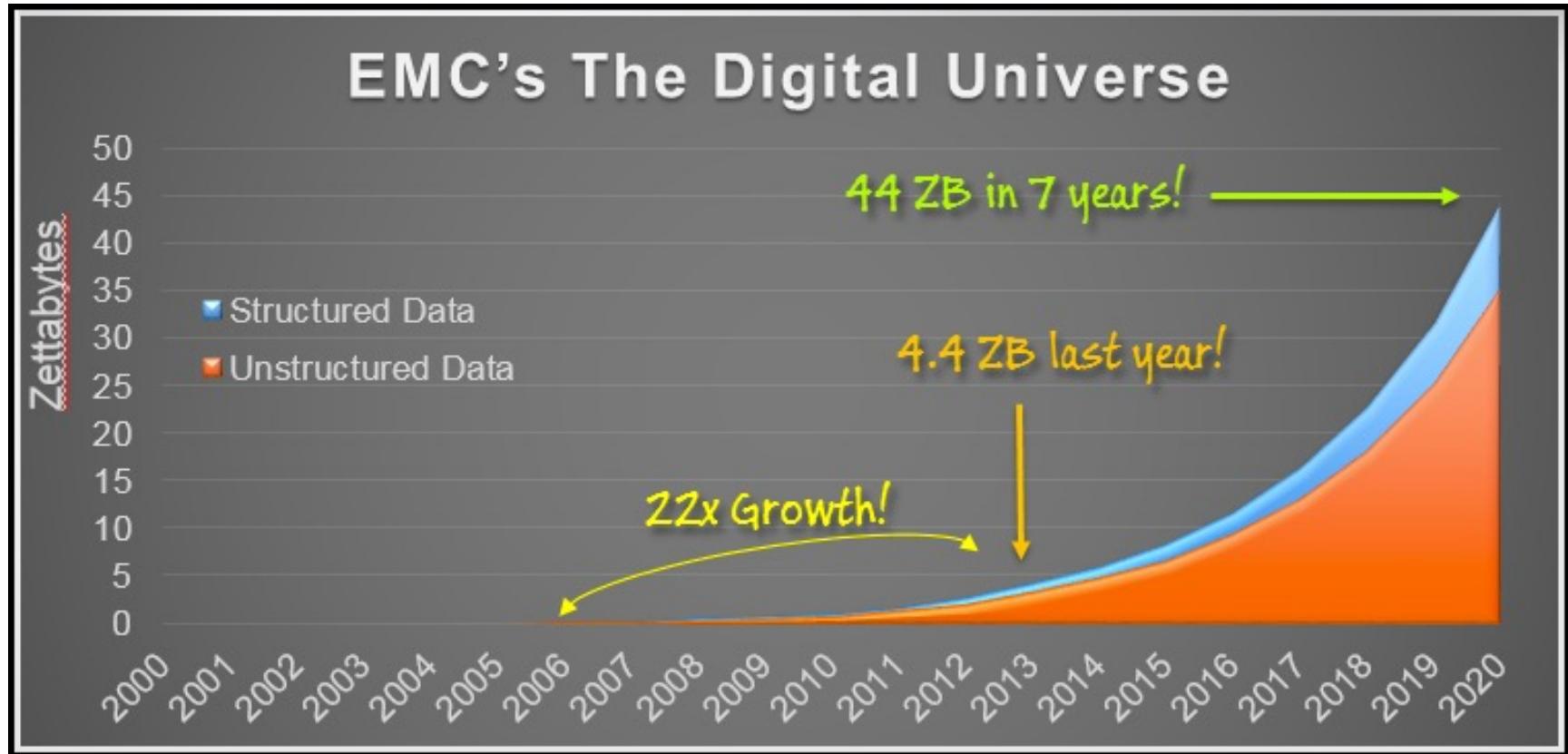


ĐẠI HỌC BÁCH KHOA HÀ NỘI  
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

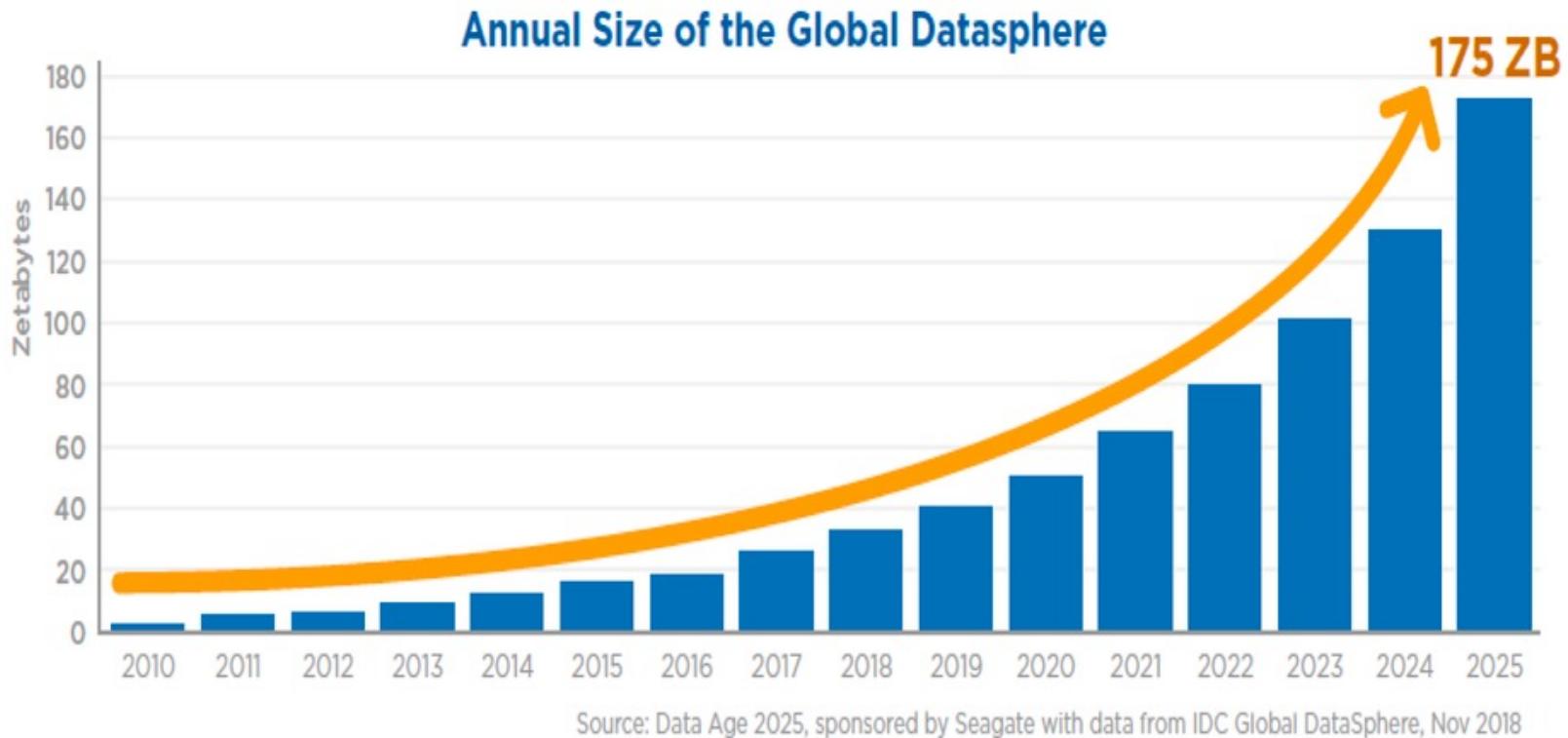
# Chương 1

# Tổng quan về lưu trữ và xử lý dữ liệu lớn

# Tổng dung lượng dữ liệu 2020



# Tổng dung lượng dữ liệu 2025



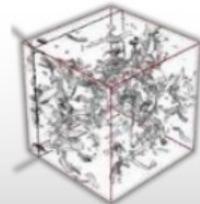
# Hình dung về độ lớn của dữ liệu



# Khoa học dữ liệu: Bước phát triển thứ 4 của khoa học khám phá



$$\left(\frac{\dot{a}}{a}\right)^2 = \frac{4\pi G\rho}{3} - K \frac{c^2}{a^2}$$



Experimental	Theoretical	Computational	The Fourth Paradigm
Thousand years ago <i>Description of natural phenomena</i>	Last few hundred years <i>Newton's laws, Maxwell's equations...</i>	Last few decades <i>Simulation of complex phenomena</i>	Today and the Future <i>Unify theory, experiment and simulation with large multidisciplinary Data</i> <i>Using data exploration and data mining (from instruments, sensors, humans...)</i> <i>Distributed Communities</i>

# Nói về dữ liệu lớn năm 2008

<http://www.wired.com/wired/issue/16-07>

September 2008



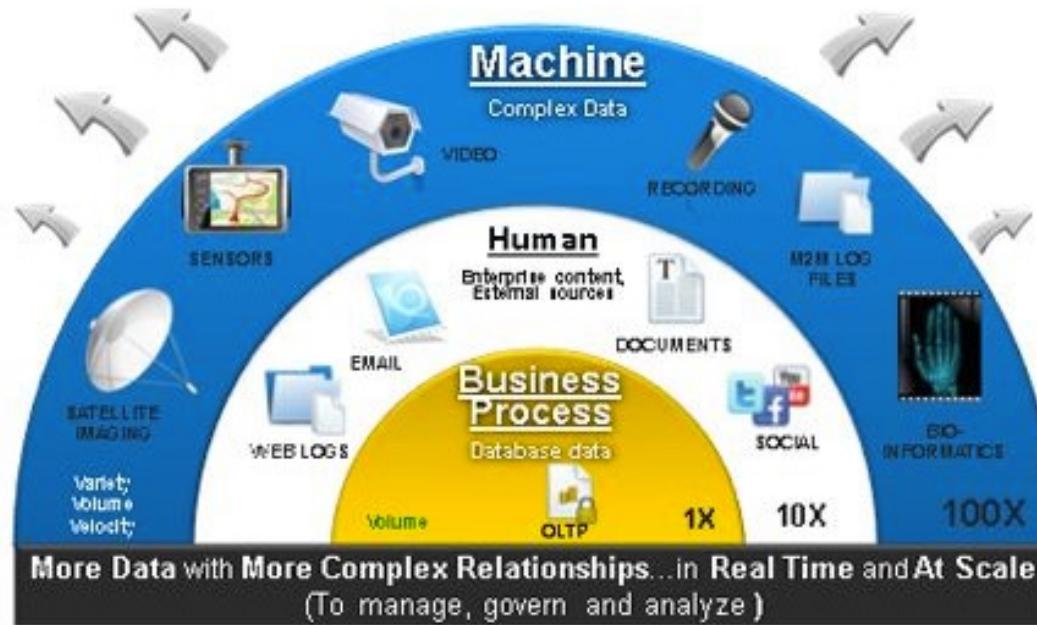
# Những con số về tốc độ sinh dữ liệu

## 2020 *This Is What Happens In An Internet Minute*

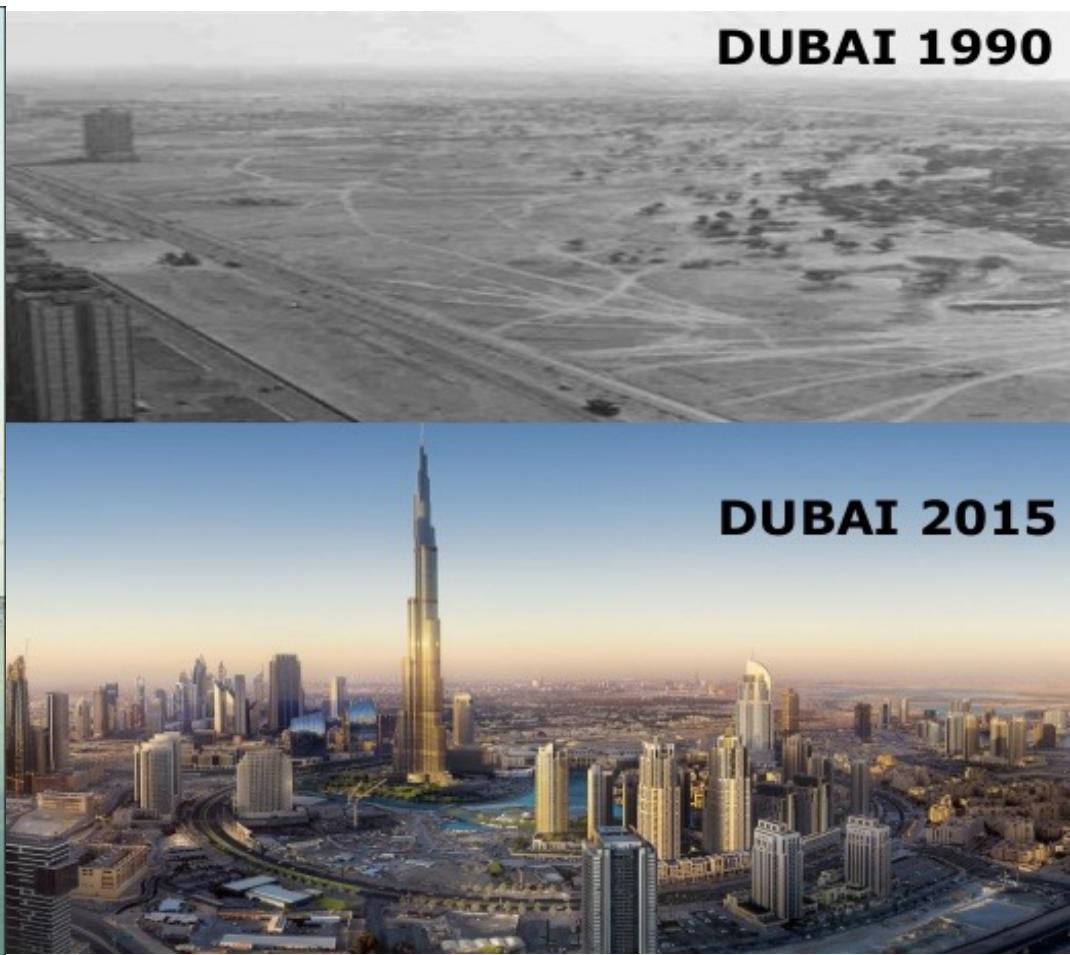


# Các nguồn tạo ra dữ liệu lớn

- Thương mại điện tử
- Mạng xã hội
- Internet vạn vật (IoT)
- Các thử nghiệm dữ liệu lớn (tin sinh học, vật lý lượng tử, vvv)



# Dữ liệu được ví như nguồn tài nguyên dầu mỏ mới

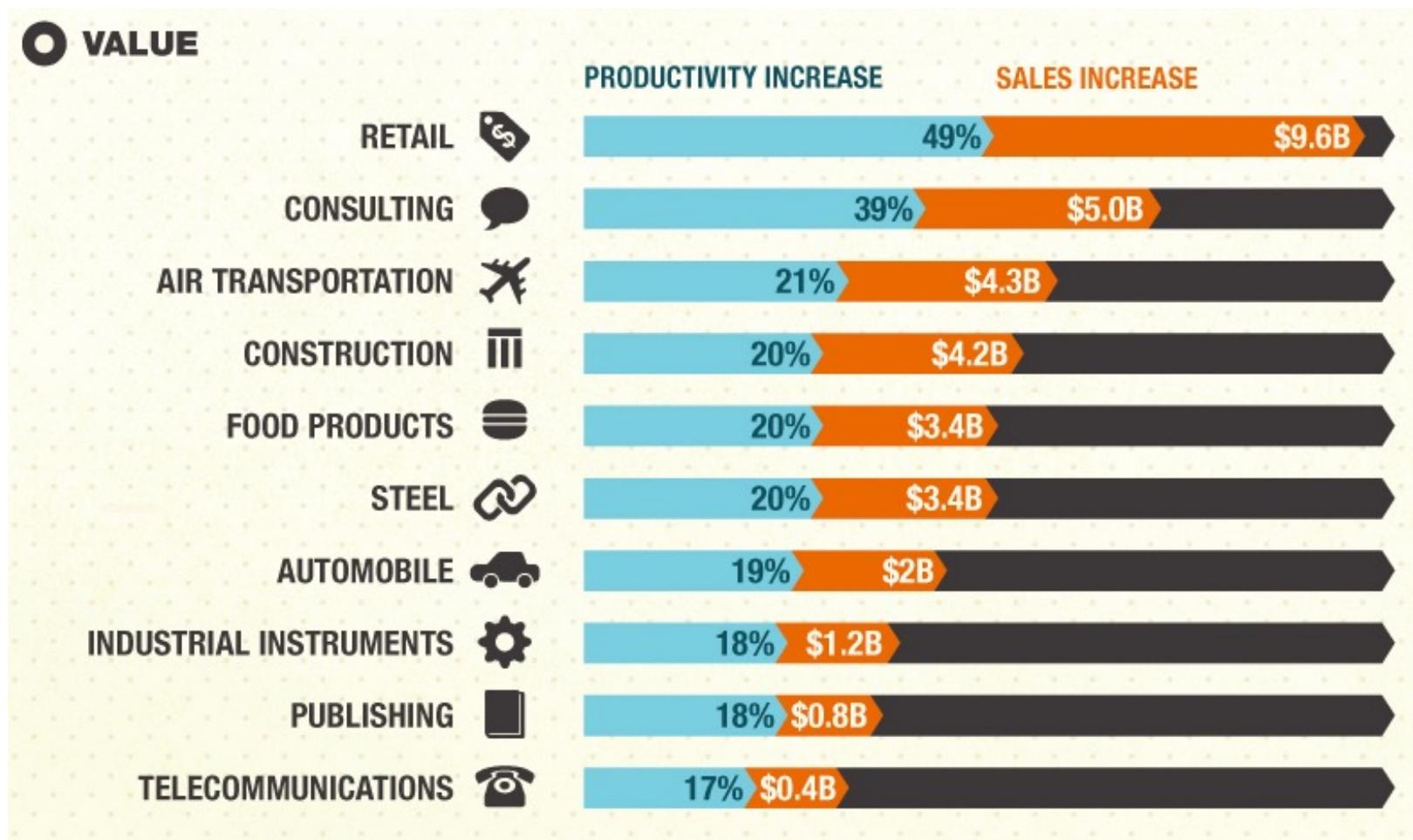


# Đặc điểm 5'V của dữ liệu lớn



Dữ liệu lớn là tập dữ liệu quá lớn hoặc là quá phức tạp mà các nền tảng lưu trữ và xử lý dữ liệu truyền thống không đáp ứng được.

# Dữ liệu lớn – giá trị mang lại lớn



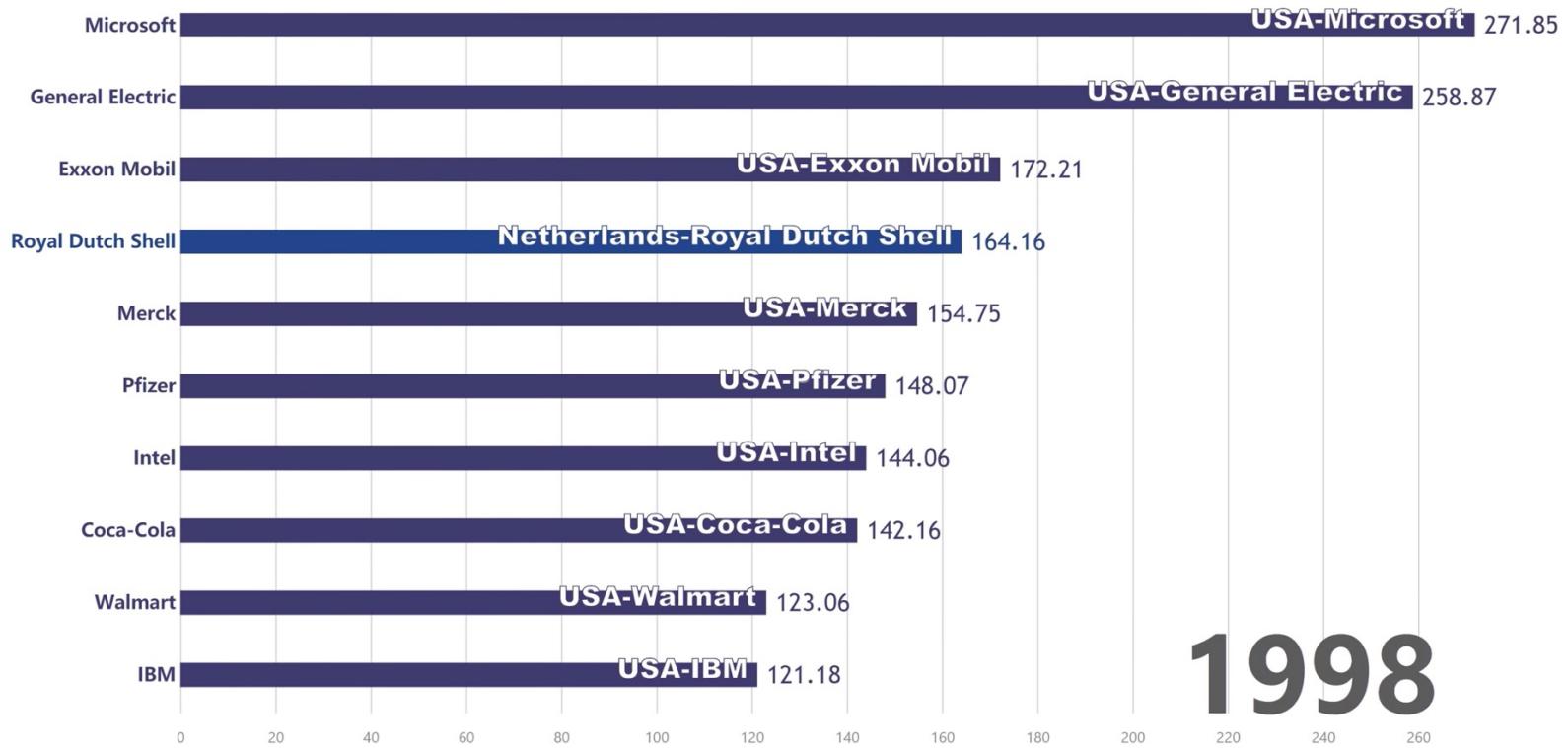
# Khai thác dữ liệu lớn trong khoa học khám phá



CERN's Large Hydron Collider (LHC) generates 15 PB a year

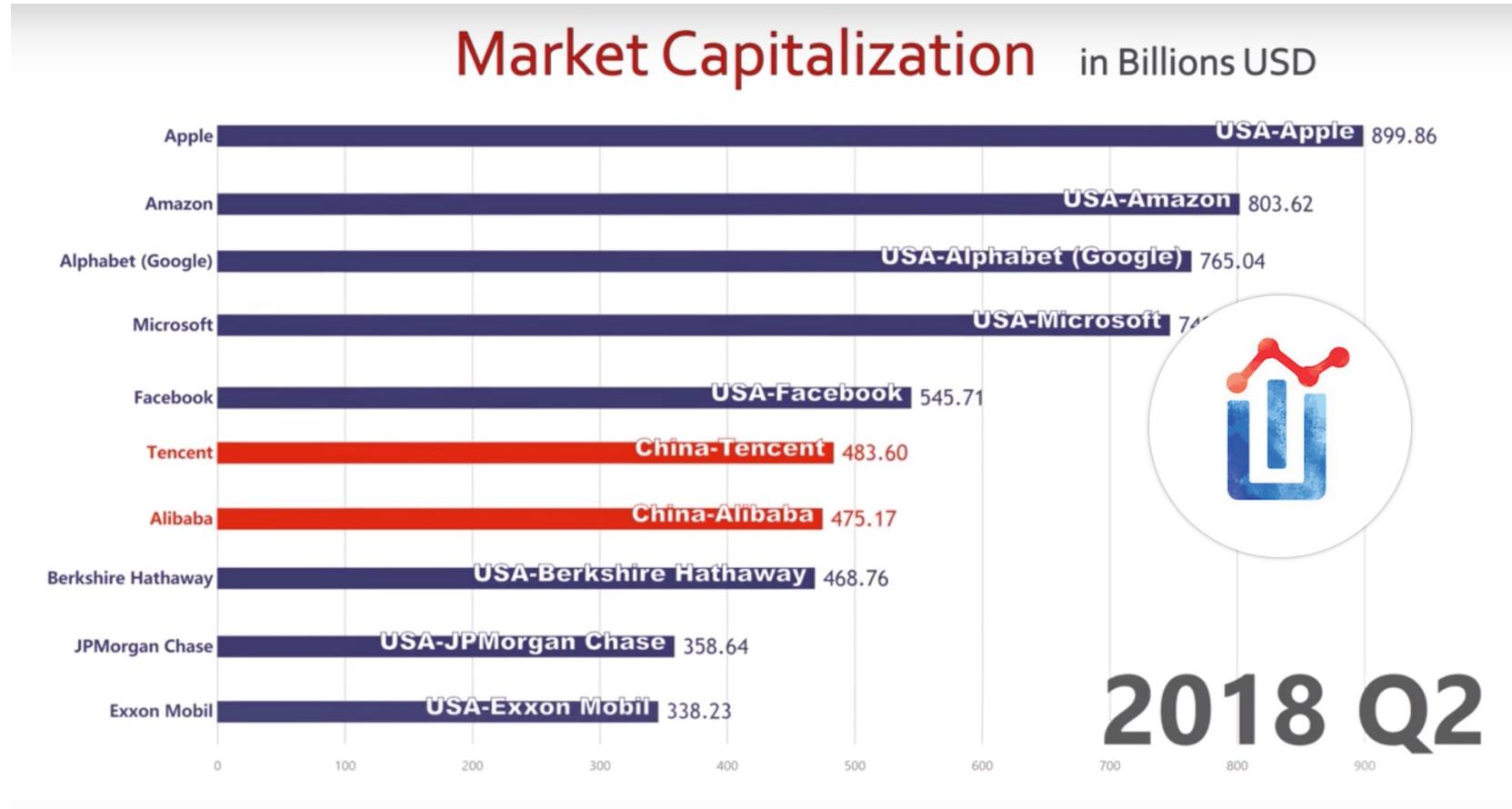
# 10 công ty lớn nhất (1998-2018)

## Market Capitalization in Billions USD

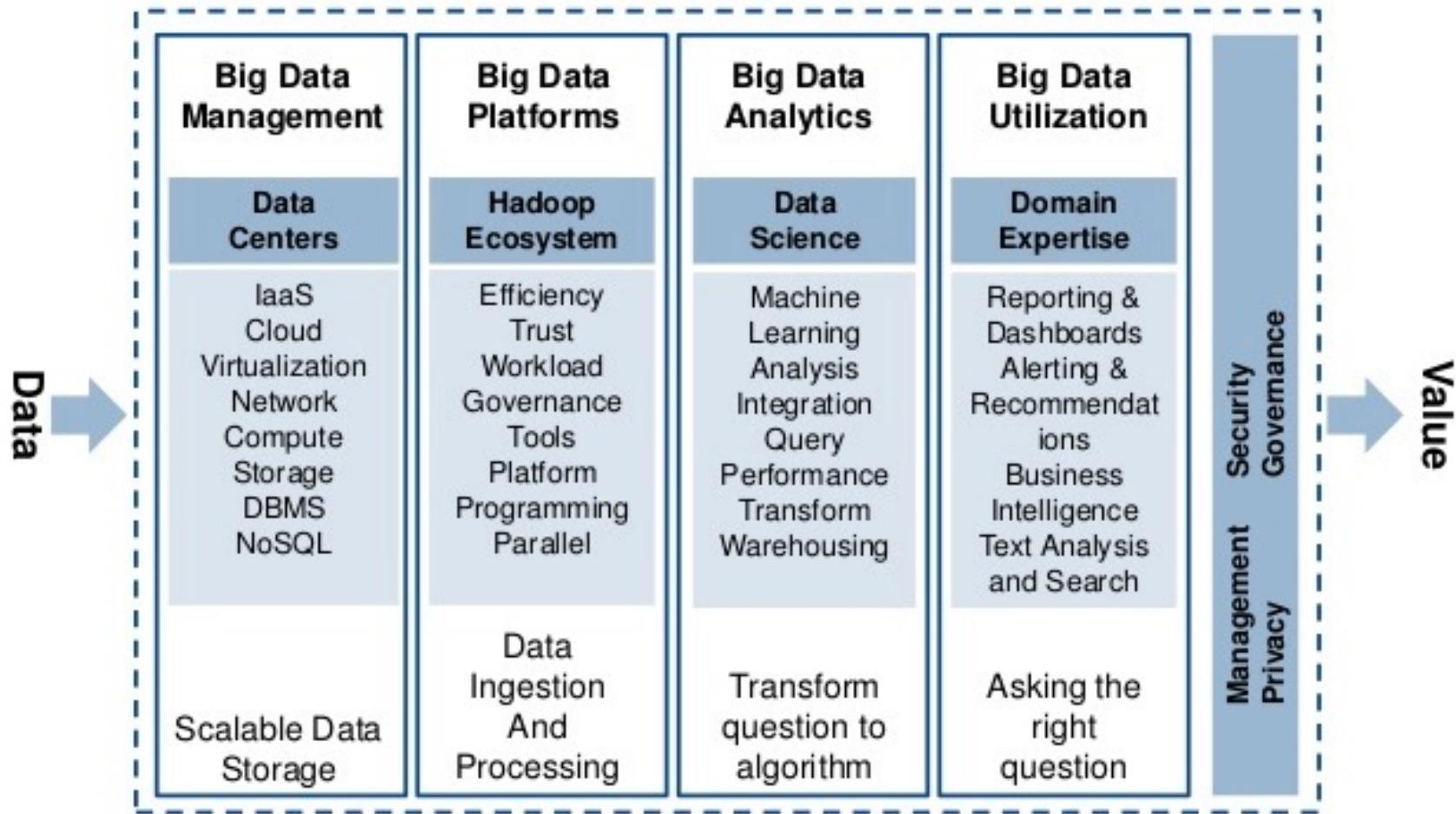


1998

# 10 công ty lớn nhất (1998-2018)



# Các tầng công nghệ cho dữ liệu lớn



# Quản lý dữ liệu phải khả mở

- Scalability
  - Khả năng quản lý lượng dữ liệu lớn không ngừng tăng lên theo thời gian.
- Accessibility
  - Cho phép đọc ghi I/O dữ liệu hiệu quả.
- Transparency
  - Truy cập dữ liệu dễ dàng, vị trí lưu trữ dữ liệu trên hệ thống là trong suốt với người dùng cuối.
- Availability
  - Khả năng chống chịu lỗi, khi tăng số lượng người dùng, khi hỏng hóc.

# Xử lý và tích hợp dữ liệu phải khả mở

- Tích hợp dữ liệu

- Dữ liệu có định dạng khác nhau
- Dữ liệu tồn tại ở các mô hình và lược đồ dữ liệu khác nhau
- Các vấn đề liên quan đến an toàn an ninh thông tin, quyền riêng tư

- Xử lý dữ liệu

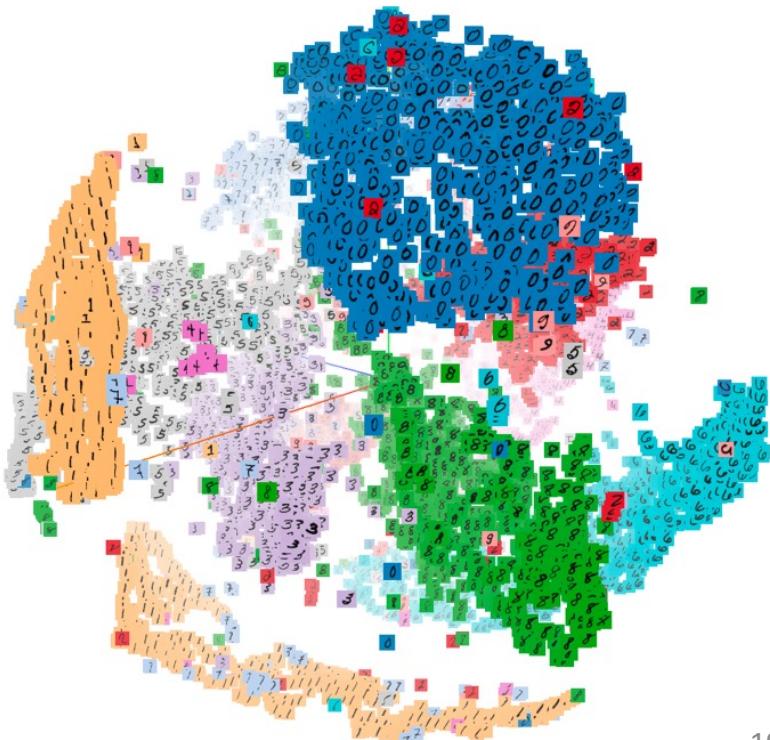
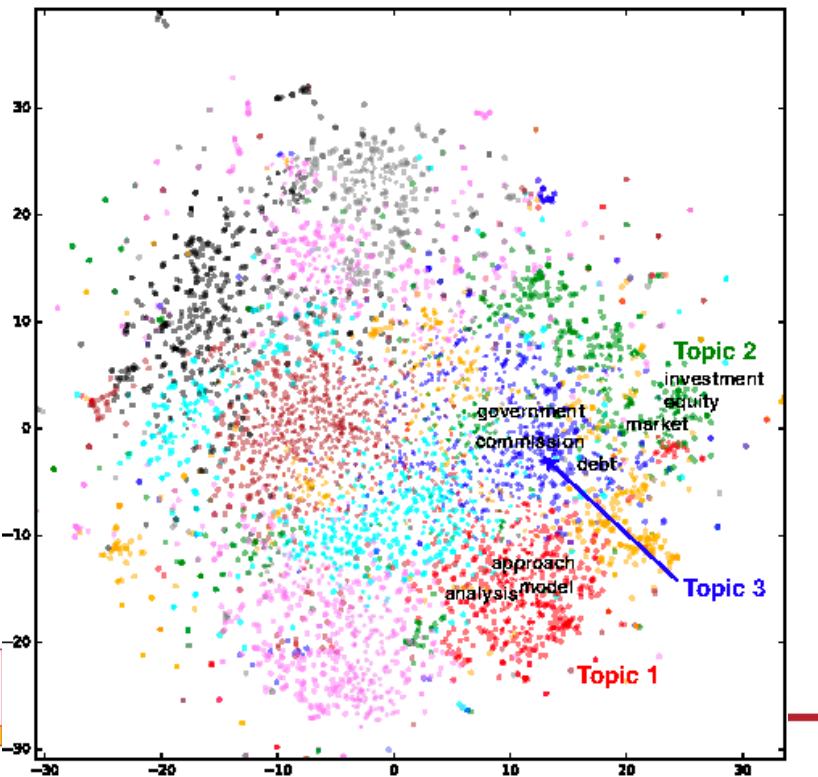
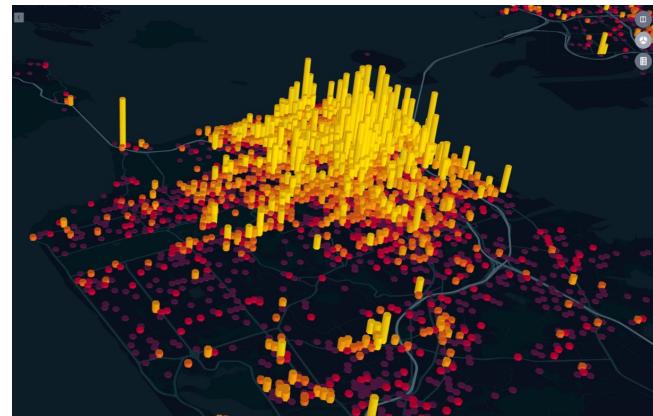
- Xử lý khối lượng dữ liệu rất lớn
- Xử lý luồng dữ liệu lớn
- Xử lý dữ liệu song song, phân tán truyền thống (OpenMP, MPI)
  - Phức tạp, khó học
  - Khả năng khả mở có giới hạn
  - Cơ chế chịu lỗi kém
  - Chi phí hạ tầng đắt đỏ
- Kiến trúc xử lý dữ liệu luồng dữ liệu lớn
  - Spark mini-batch
  - Apache Flink

# Các giải thuật phân tích dữ liệu khả mở

- Làm nhỏ lại dữ liệu cho phù hợp với các giải thuật truyền thống
  - Eg. Sub-sampling
  - Eg. Principal component analysis
  - Eg. Feature extraction and feature selection
- Song song hoá các giải thuật học máy
  - Eg. k-nn classification based on MapReduce
  - Eg. scaling-up support vector machines (SVM) by a divide and-conquer approach

# Sử dụng và trực quan hóa dữ liệu lớn

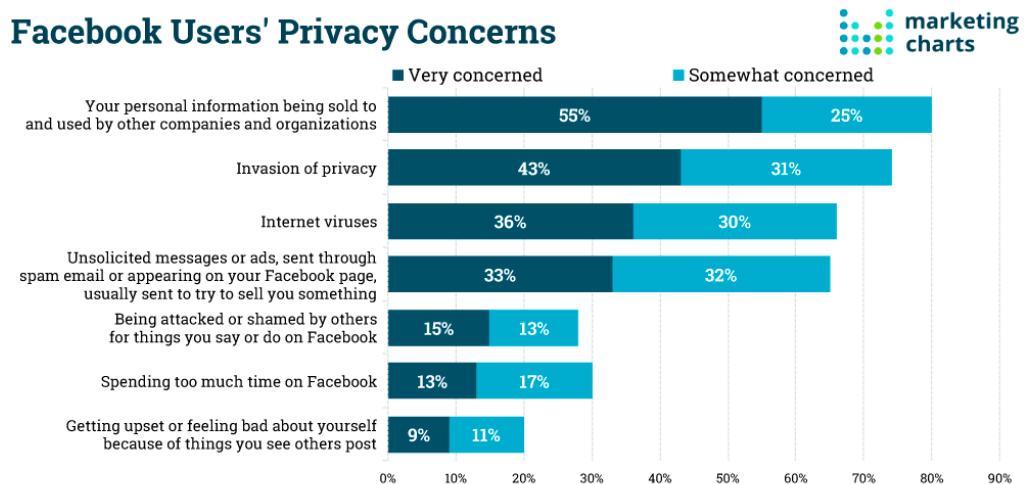
- Cần kiến thức chuyên gia
- Cần kỹ thuật và công cụ để hỗ trợ hiệu quả việc trình diễn và hiểu về dữ liệu lớn



# Bảo mật và quyền riêng tư



## Facebook Users' Privacy Concerns



Published on MarketingCharts.com in April 2018 | Data Source: Gallup

Based on telephone interviews conducted April 2-8, 2018 among 1,509 US adults ages 18 and older, of whom 785 are Facebook users.  
The remaining respondents answered "Not too concerned" or "Not concerned at all."

## How was Facebook users' data misused?

- In 2014 a Facebook quiz invited users to find out their personality type
- The app collected the data of those taking the quiz, but also recorded the public data of their friends
- About 305,000 people installed the app, but it gathered information on up to 87 million people, according to Facebook
- It is claimed at least some of the data was sold to Cambridge Analytica (CA) which used it to psychologically profile voters in the US
- CA denies it broke any laws and says it did not use the data in the US presidential election
- Facebook sends notices to users telling them whether their data was breached

CA denies any wrongdoing. Facebook has apologised to users and says a "breach of trust" has occurred.



ĐẠI HỌC BÁCH KHOA HÀ NỘI  
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

## Chương 2

# Hệ sinh thái Hadoop

# Nội dung

- Apache Hadoop
- Hệ thống tệp tin Hadoop (HDFS)
- Mô thức xử lý dữ liệu MapReduce
- Các thành phần khác trong hệ sinh thái Hadoop

# Mục tiêu của Hadoop

- Mục tiêu chính
  - Lưu trữ dữ liệu khả mỏ, tin cậy
  - Powerful data processing
  - Efficient visualization
- Với thách thức
  - Thiết bị lưu trữ tốc độ chậm, máy tính thiếu tin cậy, lập trình song song phân tán không dễ dàng



# Giới thiệu về Apache Hadoop

- **Lưu trữ và xử lý dữ liệu khả mở, tiết kiệm chi phí**
  - Xử lý dữ liệu phân tán với mô hình lập trình đơn giản, thân thiện hơn như MapReduce
  - Hadoop thiết kế để mở rộng thông qua kỹ thuật scale-out, tăng số lượng máy chủ
  - Thiết kế để vận hành trên phần cứng phổ thông, có khả năng chống chịu lỗi phần cứng
- Lấy cảm hứng từ kiến trúc dữ liệu của Google

# Các thành phần chính của Hadoop

- Lưu trữ dữ liệu: Hệ thống tệp tin phân tán Hadoop (HDFS)
- Xử lý dữ liệu: MapReduce framework
- Các tiện ích hệ thống:
  - Hadoop Common: Các tiện ích chung hỗ trợ các thành phần của Hadoop.
  - Hadoop YARN: Một framework quản lý tài nguyên và lập lịch trong cụm Hadoop.

# Hadoop giải quyết bài toán khả mở

- Thiết kế hướng “phân tán” ngay từ đầu
  - Hadoop mặc định thiết kế để triển khai trên cụm máy chủ
- Các máy chủ tham gia vào cụm được gọi là các Nodes
  - Mỗi node tham gia vào cả 2 vai trò lưu trữ và tính toán
- **Hadoop mở rộng bằng kỹ thuật scale-out**
  - Có thể tăng cụm Hadoop lên hàng chục ngàn nodes

# Hadoop giải quyết bài toán chịu lỗi

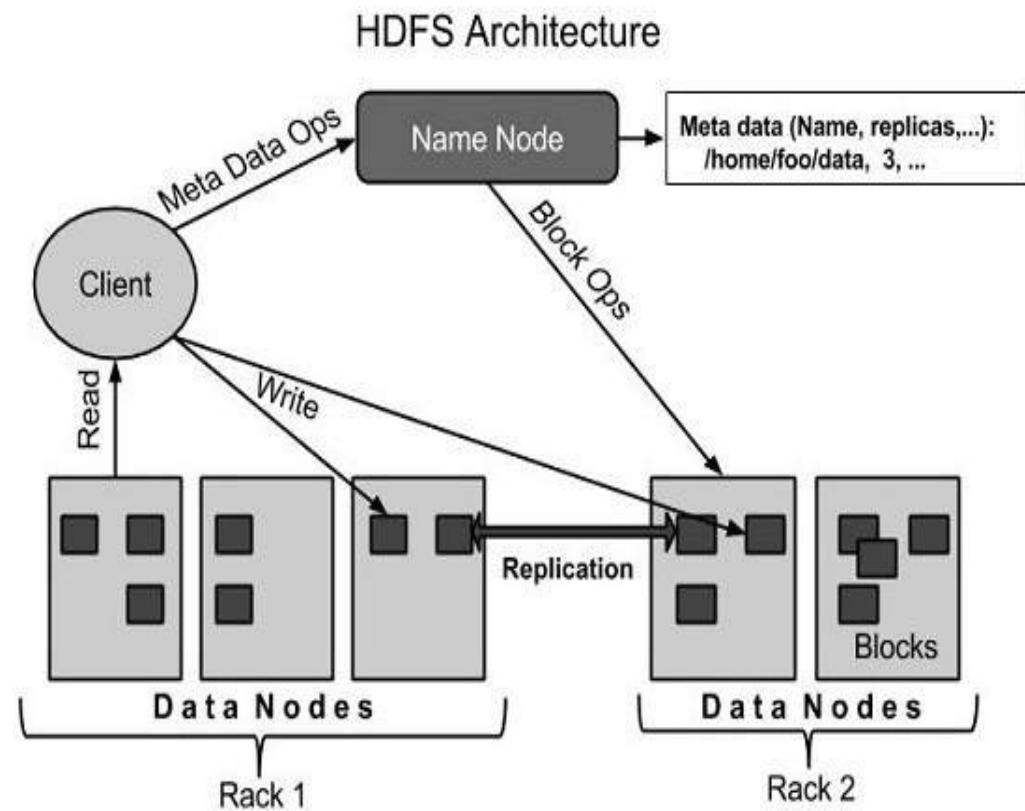
- Với việc triển khai trên cụm máy chủ phổ thông
  - Hỗn hối phần cứng là chuyện thường ngày, không phải là ngoại lệ
  - **Hadoop chịu lỗi thông qua kỹ thuật “dữ thừa”**
- Các tệp tin trong HDFS được phân mảnh, nhân bản ra các nodes trong cụm
  - Nếu một node gặp lỗi, dữ liệu ứng với nodes đó được tái nhân bản qua các nodes khác
- Công việc xử lý dữ liệu được phân mảnh thành các tác vụ độc lập
  - Mỗi tác vụ xử lý một phần dữ liệu đầu vào
  - Các tác vụ được thực thi song song với các tác vụ khác
  - Tác vụ lỗi sẽ được tái lập lịch thực thi trên node khác
- **Hệ thống Hadoop thiết kế sao cho các lỗi xảy ra trong hệ thống được xử lý tự động, không ảnh hưởng tới các ứng dụng phía trên**

# Tổng quan về HDFS

- HDFS cung cấp khả năng lưu trữ tin cậy và chi phí hợp lý cho khối lượng dữ liệu lớn
- Tối ưu cho các tập tin kích thước lớn (từ vài trăm MB tới vài TB)
- HDFS có không gian cây thư mục phân cấp như UNIX (vd., /hust/soict/hello.txt)
  - Hỗ trợ cơ chế phân quyền và kiểm soát người dùng như của UNIX
- Khác biệt so với hệ thống tập tin trên UNIX
  - Chỉ hỗ trợ thao tác ghi thêm dữ liệu vào cuối tệp (APPEND)
  - Ghi một lần và đọc nhiều lần

# Kiến trúc của HDFS

- Kiến trúc Master/Slave
- HDFS master: name node
  - Quản lý không gian tên và siêu dữ liệu ánh xạ tệp tin tới vị trí các chunks
  - Giám sát các data node
- HDFS slave: data node
  - Trực tiếp thao tác I/O các chunks



# Nguyên lý thiết kế cốt lõi của HDFS

- I/O pattern
  - Chỉ ghi thêm (Append) → giảm chi phí điều khiển tương tranh
- Phân tán dữ liệu
  - Tệp được chia thành các chunks lớn (64 MB)  
→ Giảm kích thước metadata  
→ Giảm chi phí truyền dữ liệu
- Nhân bản dữ liệu
  - Mỗi chunk thông thường được sao làm 3 nhân bản
- Cơ chế chịu lỗi
  - Data node: sử dụng cơ chế tái nhân bản
  - Name node
    - Sử dụng Secondary Name Node
    - SNN hỏi data nodes khi khởi động thay vì phải thực hiện cơ chế đồng bộ phức tạp với primary NN

# Mô thức xử lý dữ liệu MapReduce

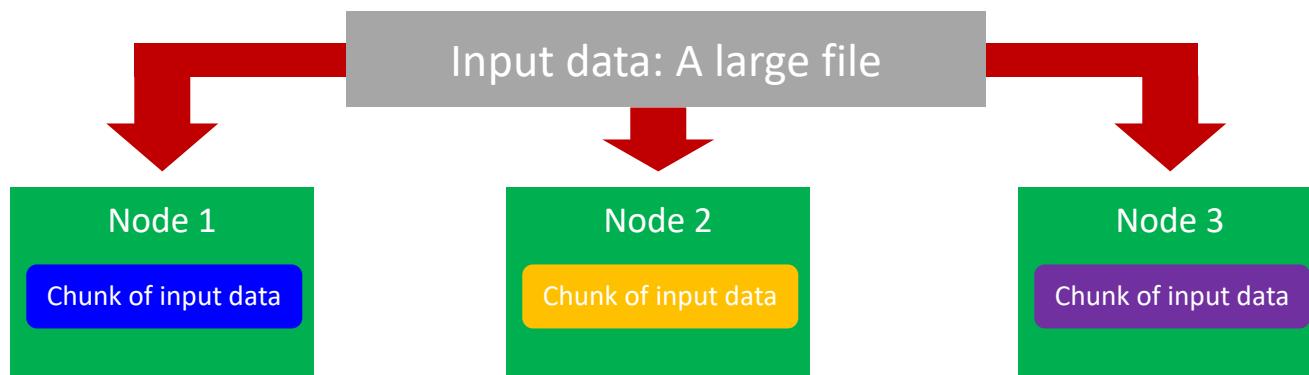
- MapReduce là mô thức xử lý dữ liệu mặc định trong Hadoop
- MapReduce không phải là ngôn ngữ lập trình, được đề xuất bởi Google
- Đặc điểm của MapReduce
  - Đơn giản (Simplicity)
  - Linh hoạt (Flexibility)
  - Khả mở (Scalability)

# A MR job = {Isolated Tasks}n

- Mỗi chương trình MapReduce là một công việc (job) được phân rã làm nhiều tác vụ độc lập (task) và các tác vụ này được phân tán trên các nodes khác nhau của cụm để thực thi
- Mỗi tác vụ được thực thi độc lập với các tác vụ khác để đạt được tính khả mở
  - Giảm truyền thông giữa các node máy chủ
  - Tránh phải thực hiện cơ chế đồng bộ giữa các tác vụ

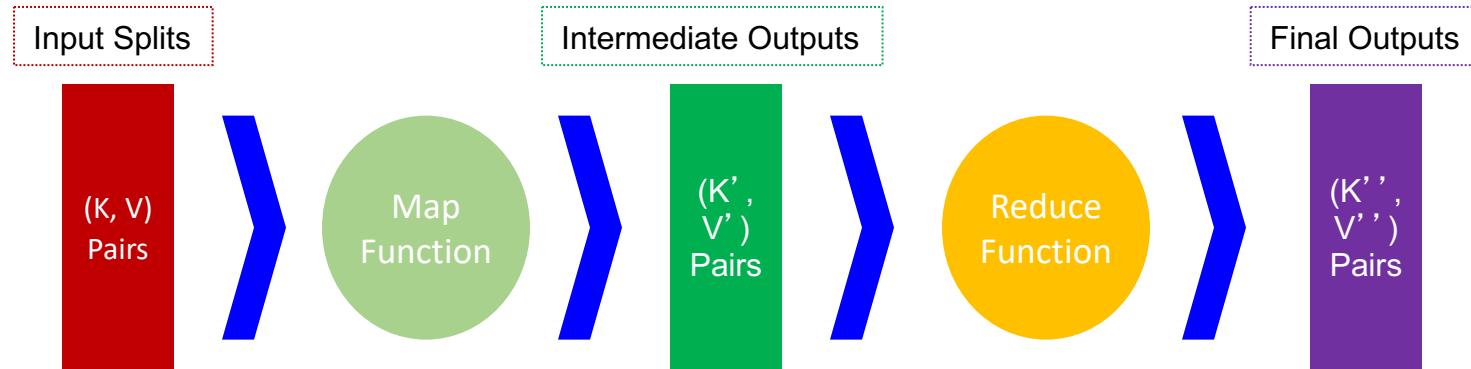
# Dữ liệu cho MapReduce

- MapReduce trong môi trường Hadoop thường làm việc với dữ liệu đa có sẵn trên HDFS
- Khi thực thi, mã chương trình MapReduce được gửi tới các node đã có dữ liệu tương ứng



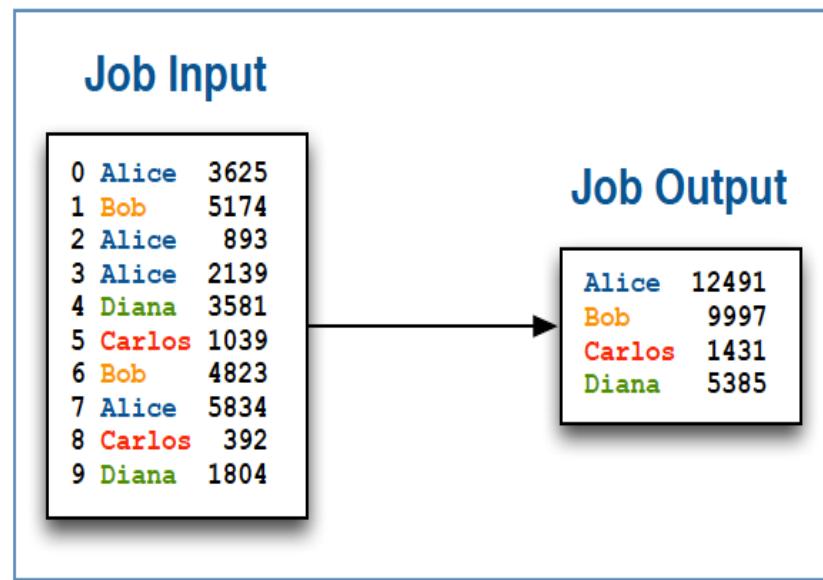
# Chương trình MapReduce

- Lập trình với MapReduce cần cài đặt 2 hàm Map và Reduce
  - 2 hàm này được thực thi bởi các tiến trình Mapper và Reducer tương ứng.
- Trong chương trình MapReduce, dữ liệu được nhìn nhận như là các cặp khóa – giá trị (key – value)
- Các hàm Map và Reduce nhận đầu vào và trả về đầu ra các cặp (key – value)



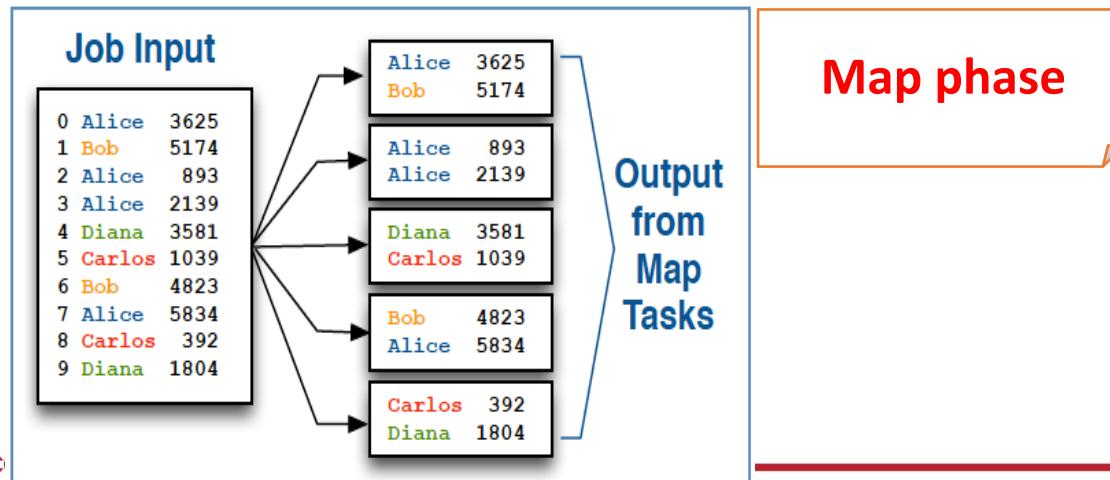
# Ví dụ về MapReduce

- Đầu vào: tệp văn bản chứa thông tin về order ID, employee name, and sale amount
- Đầu ra : Doanh số bán (sales) theo từng nhân viên (employee)



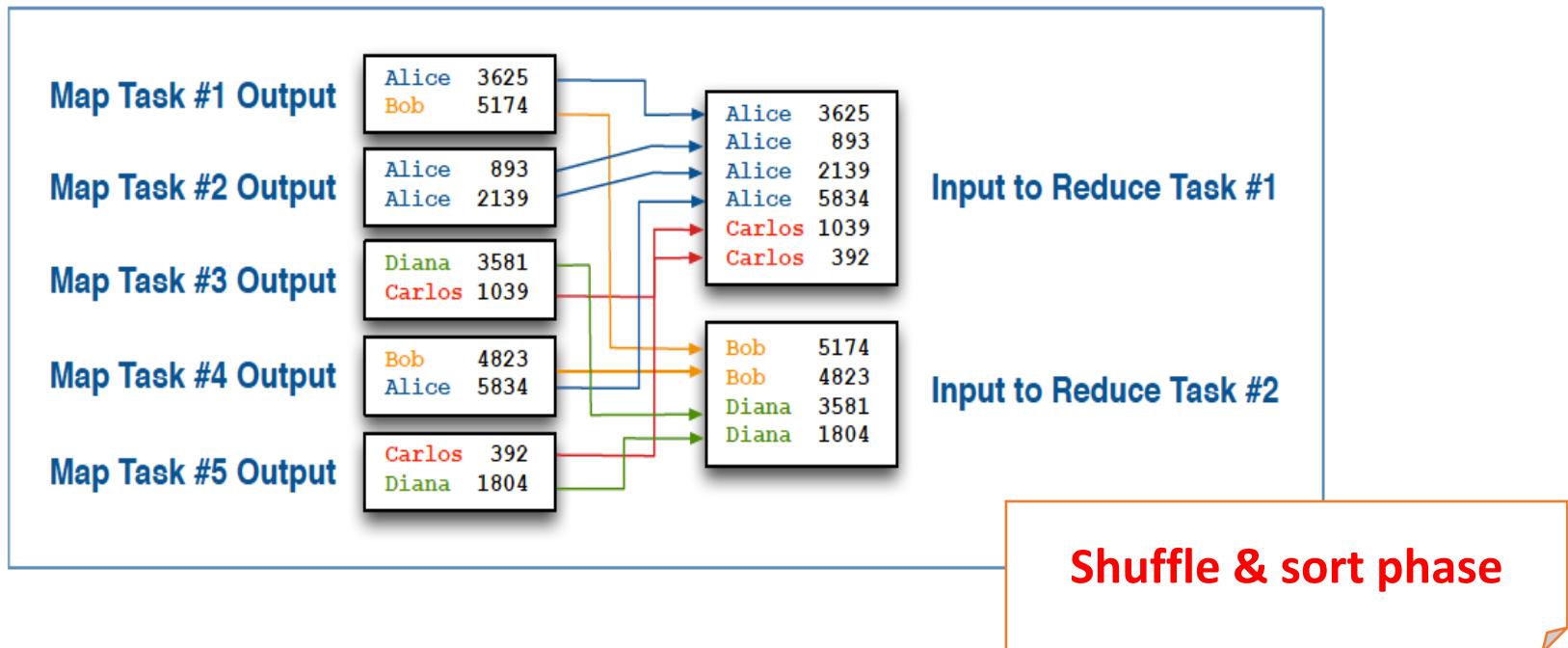
# Bước Map

- Dữ liệu đầu vào được xử lý bởi nhiều tác vụ Mapping độc lập
  - Số lượng các tác vụ Mapping được xác định theo lượng dữ liệu đầu vào (~ số chunks)
  - Mỗi tác vụ Mapping xử lý một phần dữ liệu (chunk) của khối dữ liệu ban đầu
- Với mỗi tác vụ Mapping, Mapper xử lý lần lượt từng bản ghi đầu vào
  - Với mỗi bản ghi đầu vào (key-value), Mapper đưa ra 0 hoặc nhiều bản ghi đầu ra (key – value trung gian)
- Trong ví dụ này, tác vụ Mapping đơn giản đọc từng dòng văn bản và đưa ra tên nhân viên và doanh số tương ứng



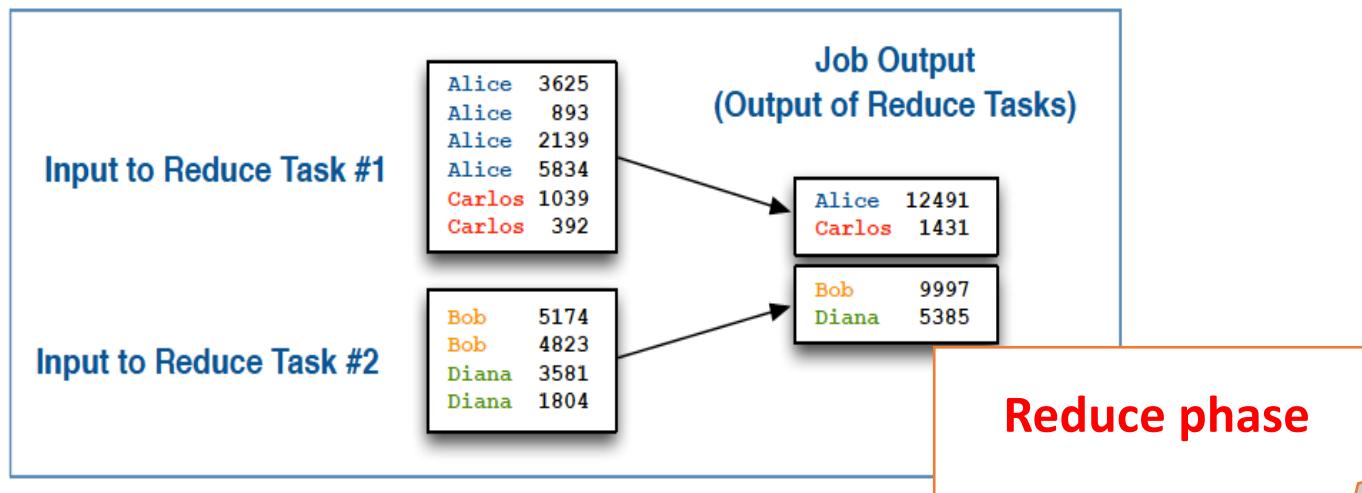
# Bước shuffle & sort

- Hadoop tự động sắp xếp và gộp đầu ra của các Mappers theo các partitions
  - Mỗi partitions là đầu vào cho một Reducer

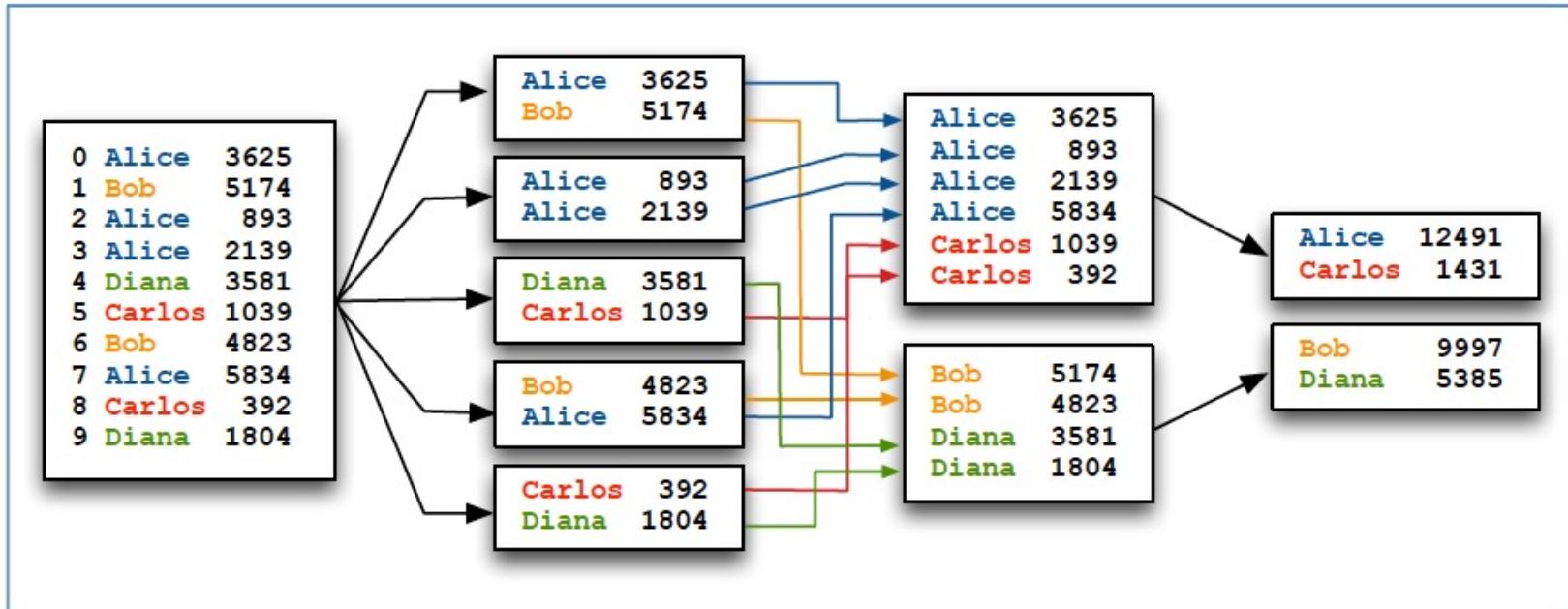


# Bước Reduce

- Reducer nhận dữ liệu đầu vào từ bước shuffle & sort
  - Tất cả các bản ghi key – value tương ứng với một key được xử lý bởi một Reducer duy nhất
  - Giống bước Map, Reducer xử lý lần lượt từng key, mỗi lần với toàn bộ các values tương ứng
- Trong ví dụ, hàm reduce đơn giản là tính tổng doanh số cho từng nhân viên, đầu ra là các cặp key – value tương ứng với tên nhân viên – doanh số tổng

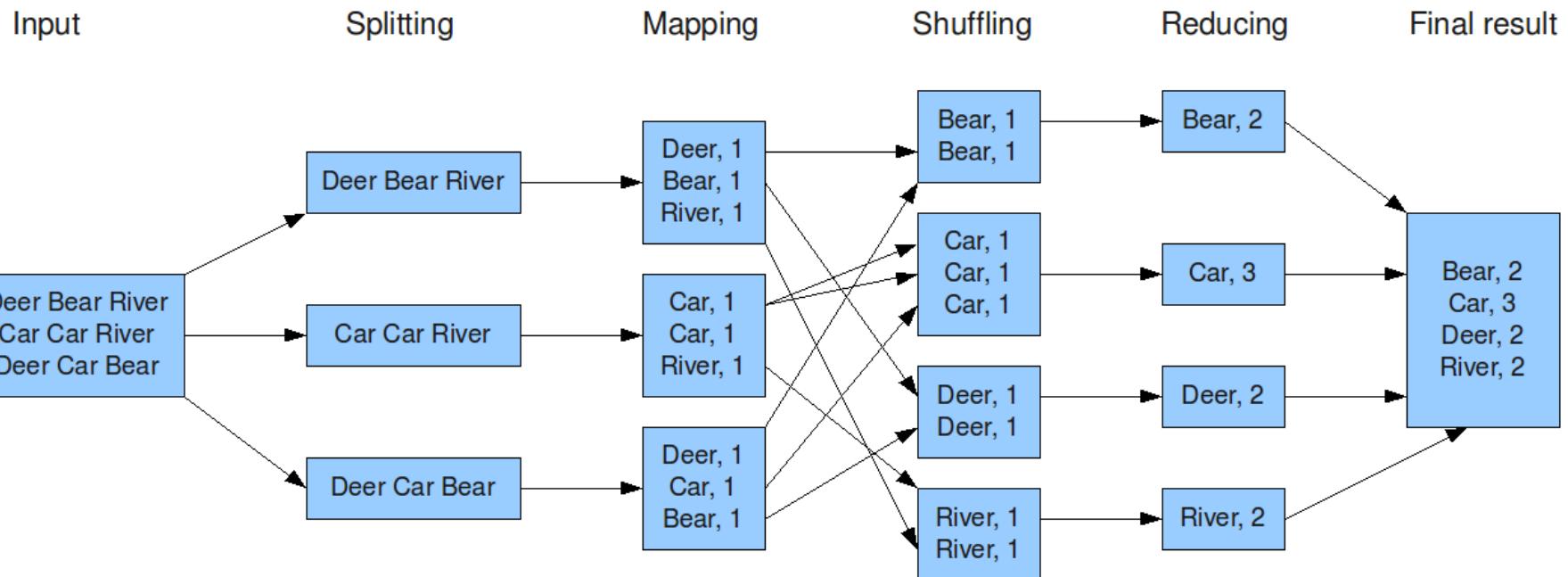


# Luồng dữ liệu



# Luồng dữ liệu với bài toán Word Count

The overall MapReduce word count process



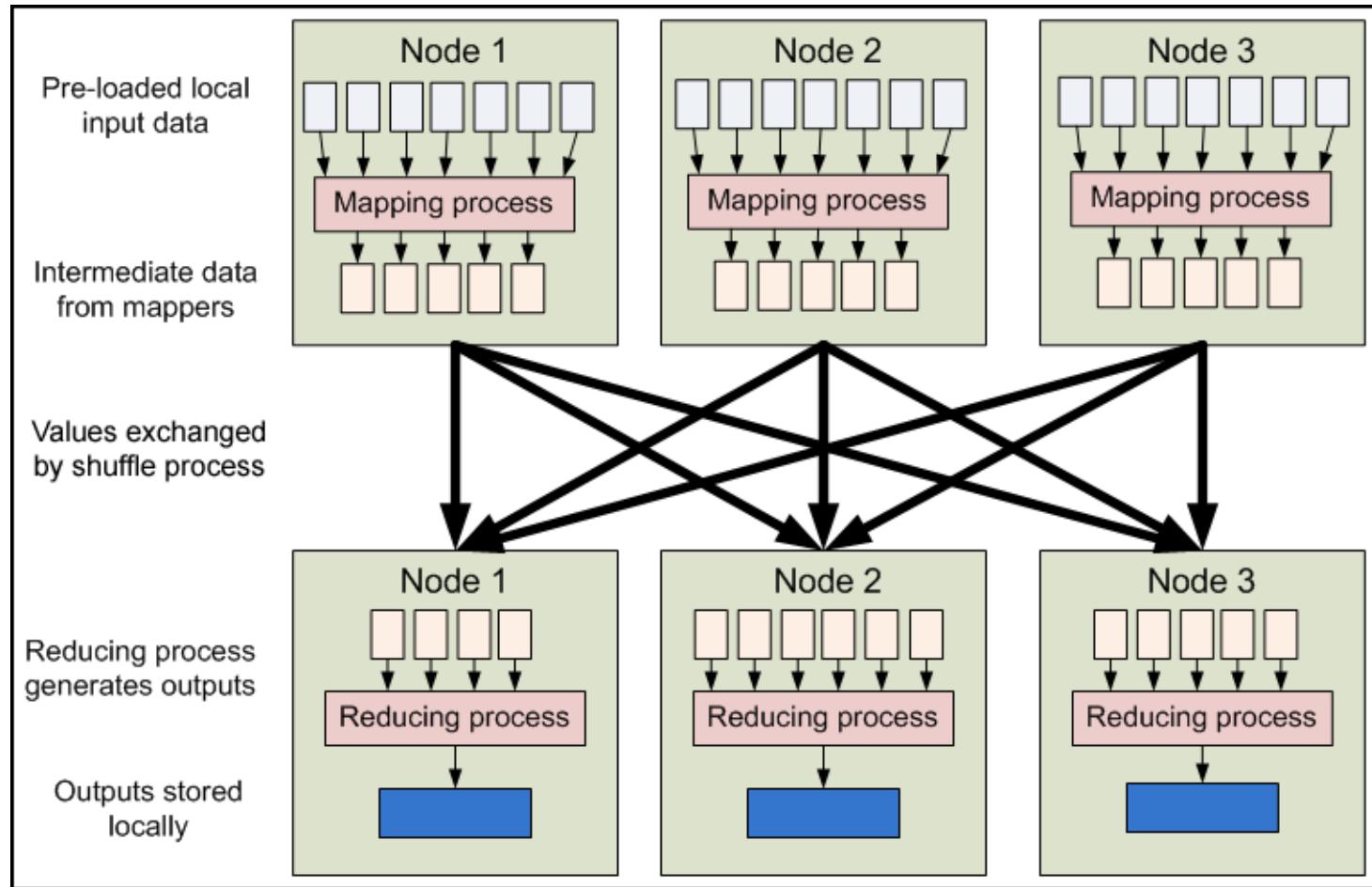
# Chương trình Word Count thực tế (1)

```
9 import org.apache.hadoop.mapreduce.Job;
10 import org.apache.hadoop.mapreduce.Mapper;
11 import org.apache.hadoop.mapreduce.Reducer;
12 import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
13 import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
14 import org.apache.hadoop.util.GenericOptionsParser;
15
16
17
18
19 public class WordCount {
20 public static void main(String [] args) throws Exception
21 {
22 Configuration c=new Configuration();
23 String[] files=new GenericOptionsParser(c,args).getRemainingArgs();
24 Path input=new Path(files[0]);
25 Path output=new Path(files[1]);
26 Job j=new Job(c,"wordcount");
27 j.setJarByClass(WordCount.class);
28 j.setMapperClass(MapForWordCount.class);
29 j.setReducerClass(ReduceForWordCount.class);
30 j.setOutputKeyClass(Text.class);
31 j.setOutputValueClass(IntWritable.class);
32 FileInputFormat.addInputPath(j, input);
33 FileOutputFormat.setOutputPath(j, output);
34 System.exit(j.waitForCompletion(true)?0:1);
35 }
```

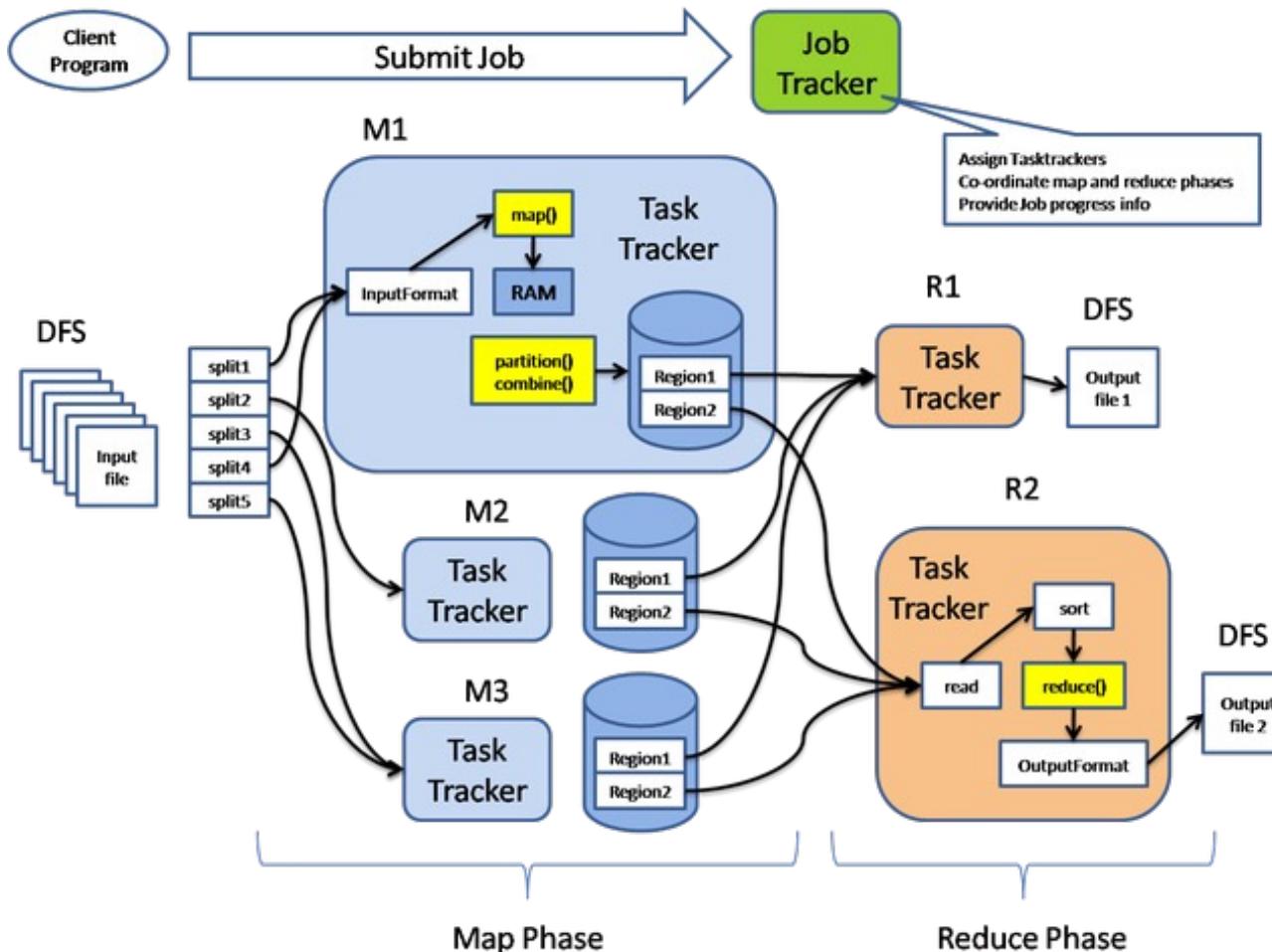
# Chương trình Word Count thực tế (2)

```
36 public static class MapForWordCount extends Mapper<LongWritable, Text, Text, IntWritable>{
37     public void map(LongWritable key, Text value, Context con) throws IOException, InterruptedException
38     {
39         String line = value.toString();
40         String[] words = line.split(",");
41         for(String word: words )
42         {
43             Text outputKey = new Text(word.toUpperCase().trim());
44             IntWritable outputValue = new IntWritable(1);
45             con.write(outputKey, outputValue);
46         }
47     }
48 }
49
50 public static class ReduceForWordCount extends Reducer<Text, IntWritable, Text, IntWritable>
51 {
52     public void reduce(Text word, Iterable<IntWritable> values, Context con) throws IOException, InterruptedException
53     {
54         int sum = 0;
55         for(IntWritable value : values)
56         {
57             sum += value.get();
58         }
59         con.write(word, new IntWritable(sum));
60     }
}
```

# MapReduce trên môi trường phân tán



# Vai trò của Job tracker và Task tracker



# Các thành phần khác trong hệ sinh thái Hadoop

- Ngoài HDFS và MapReduce, hệ sinh thái Hadoop còn nhiều hệ thống, thành phần khác phục vụ
  - Phân tích dữ liệu
  - Tích hợp dữ liệu
  - Quản lý luồng
  - Vvv
- Các thành phần này không phải là ‘core Hadoop’ nhưng là 1 phần của hệ sinh thái Hadoop
  - Hầu hết là mã nguồn mở trên Apache

# Apache Hive

- Cũng là một lớp trừu tượng mức cao của MapReduce
  - Giảm thời gian phát triển
  - Cung cấp ngôn ngữ HiveQL: SQL-like language
- Trình biên dịch Hive chạy trên máy client
  - Chuyển HiveQL script thành MapReduce jobs
  - Đệ trình các công việc này lên cụm tính toán



```
SELECT customers.cust_id, SUM(cost) AS total
      FROM customers
      JOIN orders
        ON customers.cust_id = orders.cust_id
 GROUP BY customers.cust_id
 ORDER BY total DESC;
```

# Apache Hbase



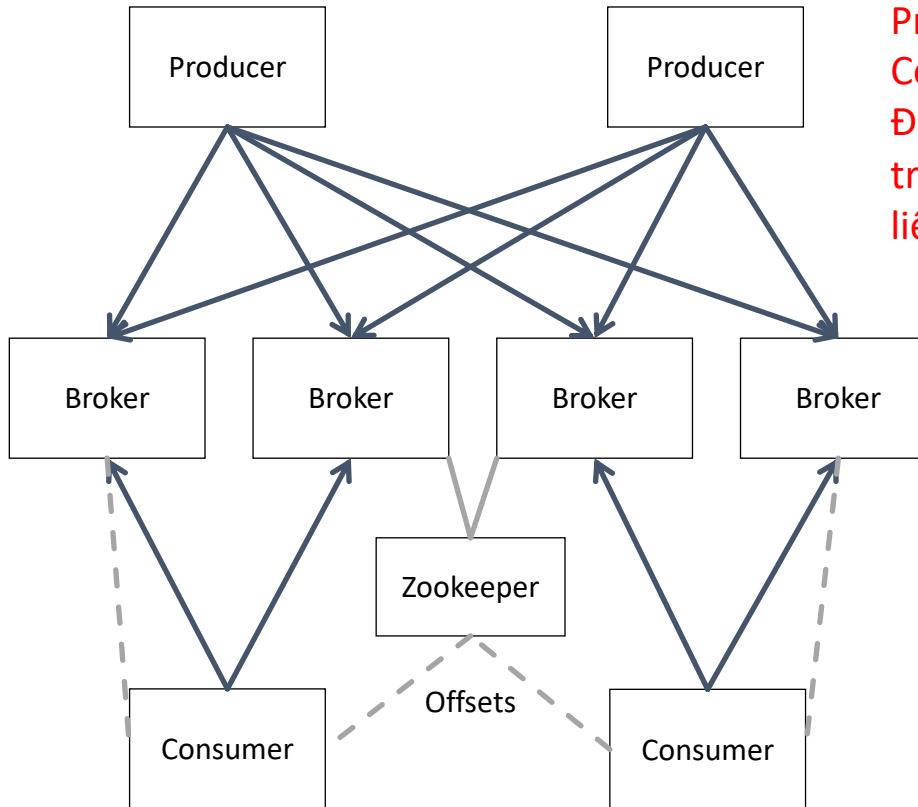
- HBase là một CSDL cột mở rộng phân tán, lưu trữ dữ liệu trên HDFS
  - Được xem như là hệ quản trị CSDL của Hadoop
- Dữ liệu được tổ chức về mặt logic là các bảng, bao gồm rất nhiều dòng và cột
  - Kích thước bảng có thể lên đến hàng Terabyte, Petabyte
  - Bảng có thể có hàng ngàn cột
- Có tính khả năng cao, đáp ứng băng thông ghi dữ liệu tốc độ cao
  - Hỗ trợ hàng trăm ngàn thao tác INSERT mỗi giây (/s)
- Tuy nhiên về các chức năng thì còn rất hạn chế khi so sánh với hệ QTCSDL truyền thống
  - Là NoSQL : không có ngôn ngữ truy vấn mức cao như SQL
  - Phải sử dụng API để scan/ put/ get/ dữ liệu theo khóa

# Apache Kafka

Producers

Kafka  
Cluster

Consumers



Producers không cần biết Consumers  
Đảm bảo sự linh hoạt và tin cậy  
trong quá trình trung chuyển dữ liệu  
giữa các bên

Kafka cho phép phân tách mạch lạc các thành phần  
tham gia vào luồng dữ liệu

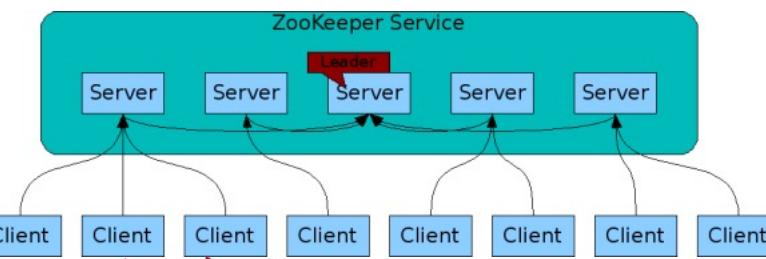
# Apache Oozie

- Oozie là một hệ thống lập lịch luồng công việc để quản lý các công việc thực thi trên cụm Hadoop
- Luồng workflow của Oozie là đồ thị vòng có hướng (Directed Acyclical Graphs (DAGs)) của các khối công việc
- Oozie hỗ trợ đa dạng các loại công việc
  - Thực thi MapReduce jobs
  - Thực thi Pig hay Hive scripts
  - Thực thi các chương trình Java hoặc Shell
  - Tương tác với dữ liệu trên HDFS
  - Chạy chương trình từ xa qua SSH
  - Gửi nhận email



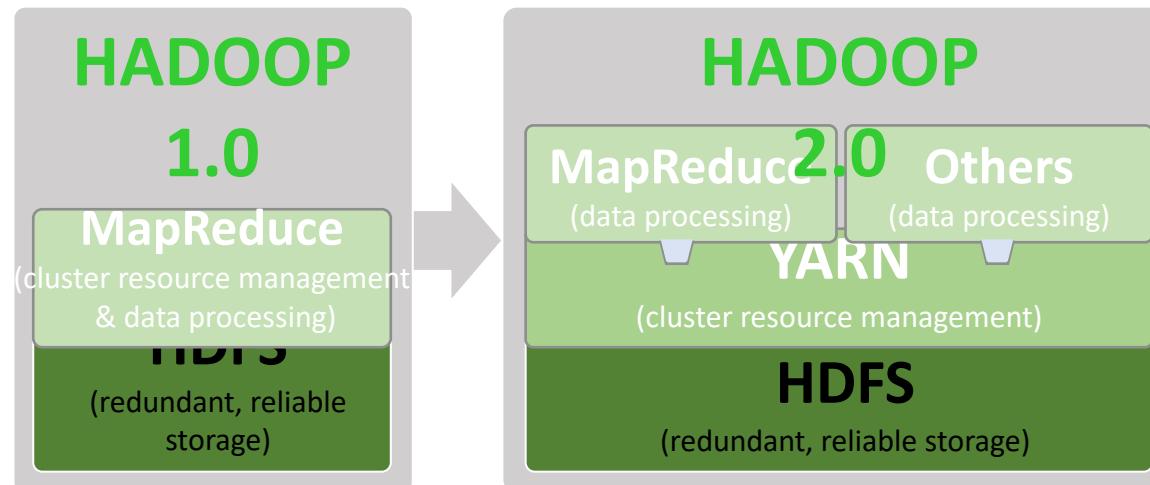
# Apache Zookeeper

- Apache ZooKeeper là một dịch vụ cung cấp các chức năng phối hợp phân tán độ tin cậy cao
  - Quản lý thành viên trong nhóm máy chủ
  - Bầu cử leader
  - Quản lý thông tin cấu hình động
  - Giám sát trạng thái hệ thống
- Đây là các service lõi, tối quan trọng trong hệ thống phân tán

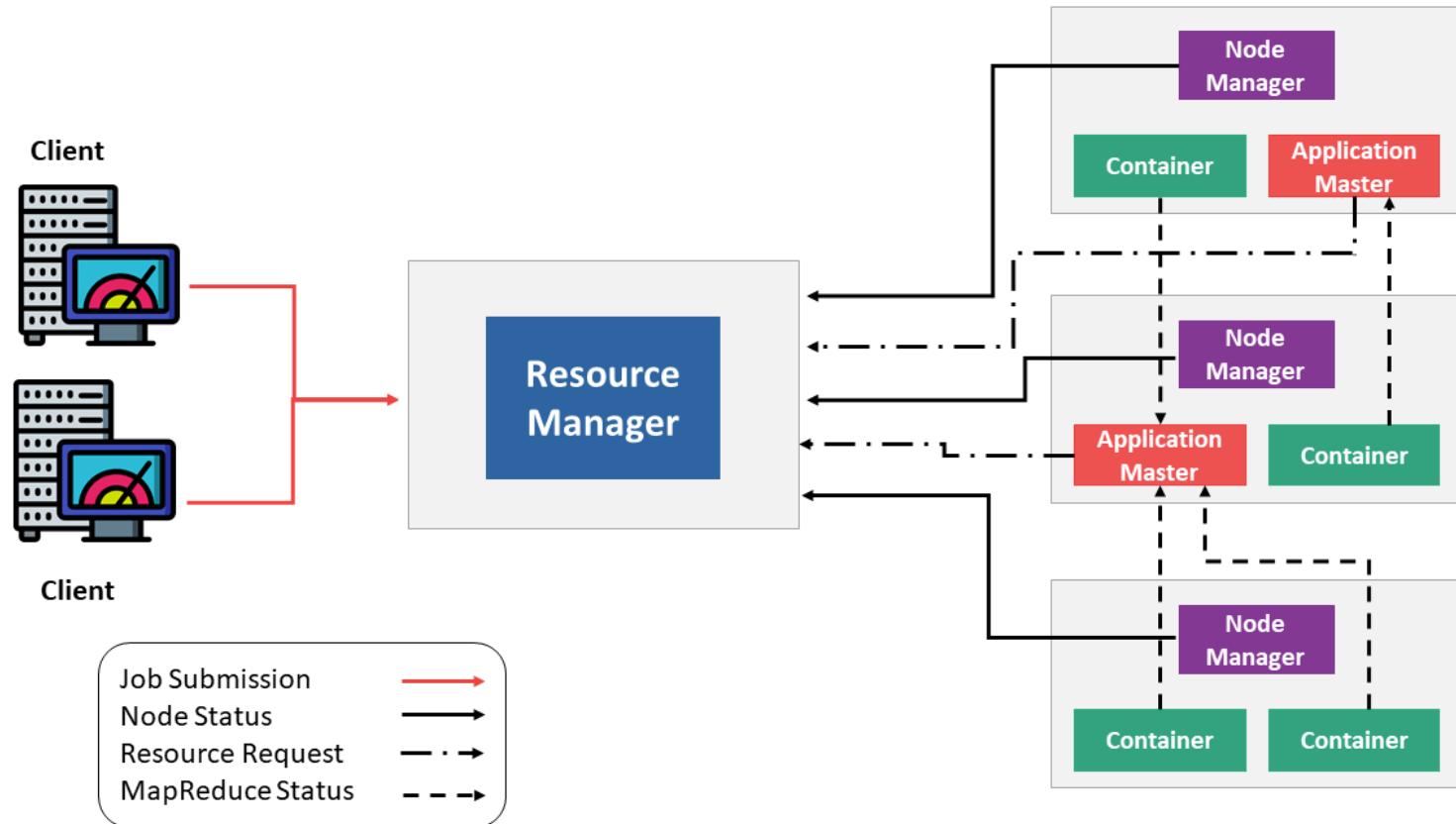


# YARN – Yet Another Resource Negotiator

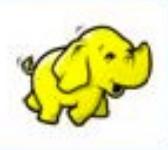
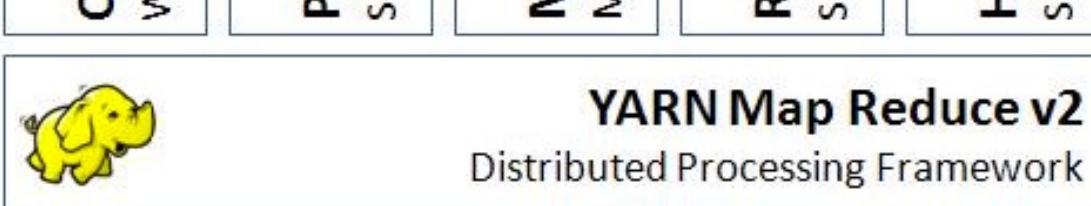
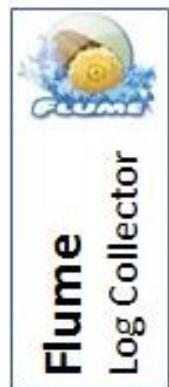
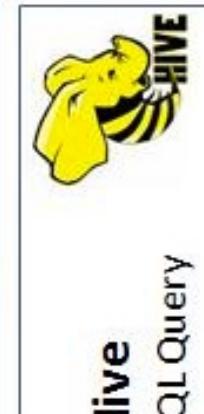
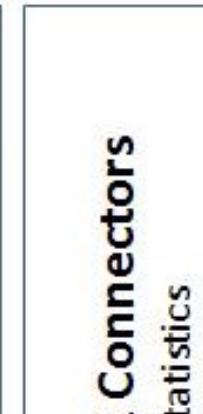
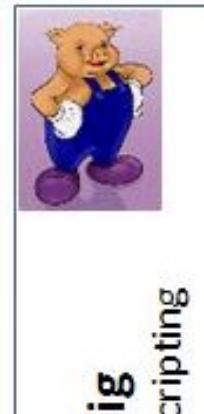
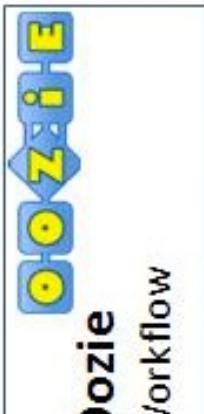
- Nodes có tài nguyên là – bộ nhớ và CPU cores
- YARN đóng vai trò cấp phát lượng tài nguyên phù hợp cho các ứng dụng khi có yêu cầu
- YARN được đưa ra từ Hadoop 2.0
  - Cho phép MapReduce và non MapReduce cùng chạy trên 1 cụm Hadoop
  - Với MapReduce job, vai trò của job tracker được thực hiện bởi application tracker



# Ví dụ về cấp phát trên YARN



# Bức tranh tổng thể hệ sinh thái Hadoop



# Chương 6

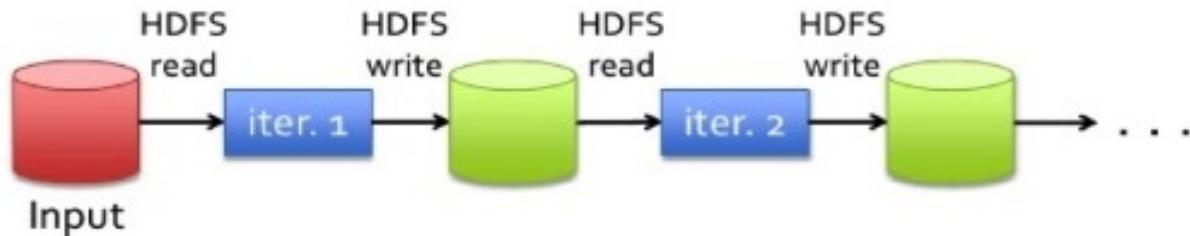
# Các kĩ thuật xử lý dữ liệu lớn theo khối - phần 2

## Apache Spark

Một nền tảng xử lý dữ liệu hợp nhất cho dữ liệu lớn

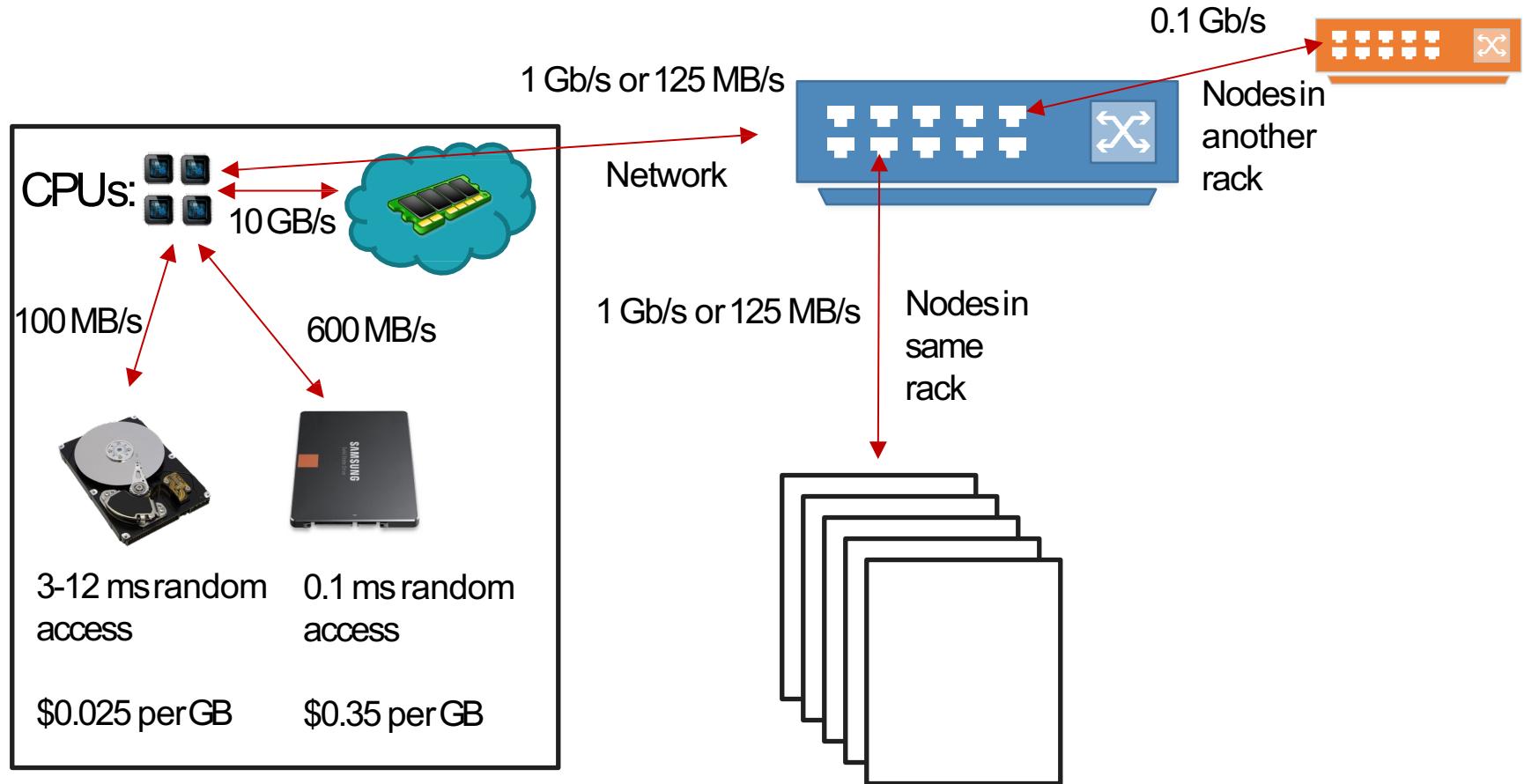
# MapReduce với chuỗi các jobs

- Iterative jobs với MapReduce đòi hỏi thao tác I/O với dữ liệu trên HDFS



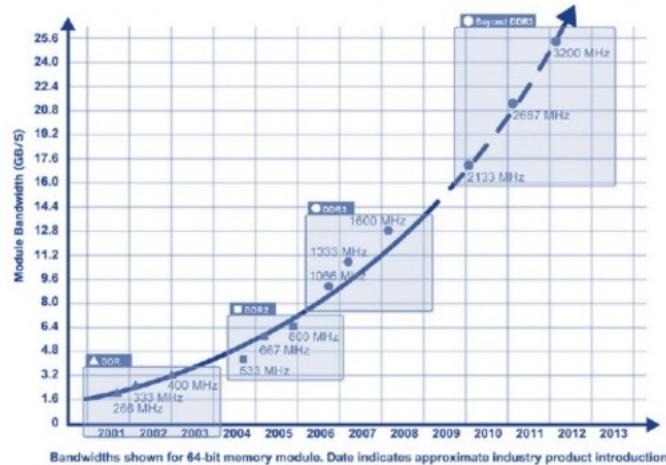
- Thực tế I/O trên ổ đĩa cứng rất chậm!

# Toàn cảnh về I/O dữ liệu

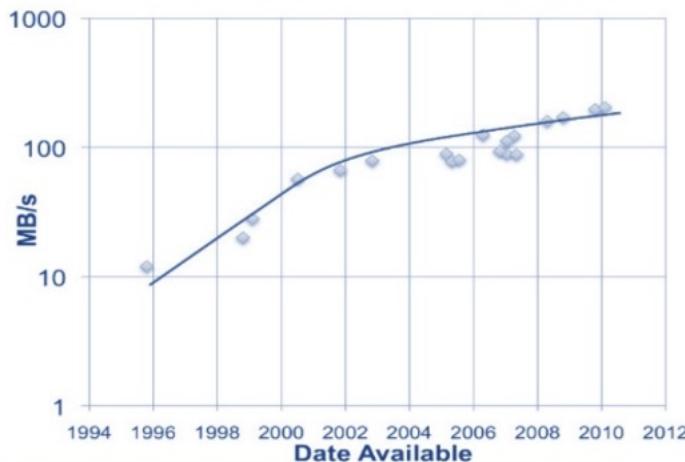


# RAM có khả năng thay thế ổ đĩa cứng

- RAM throughput increasing **exponentially**



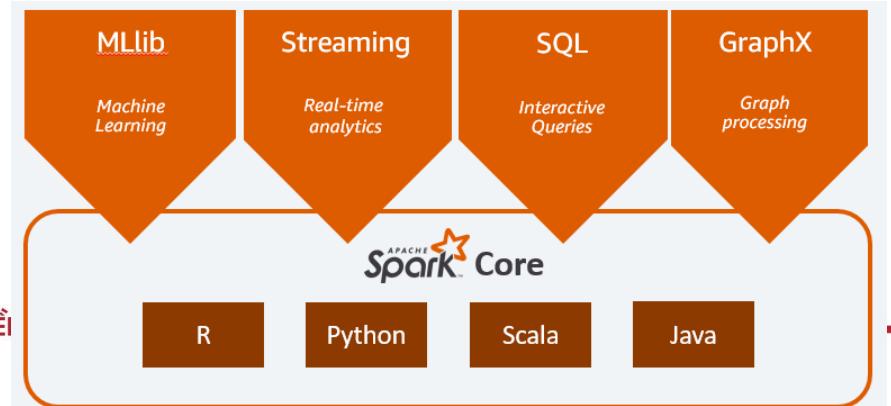
- Disk throughput increasing **slowly**



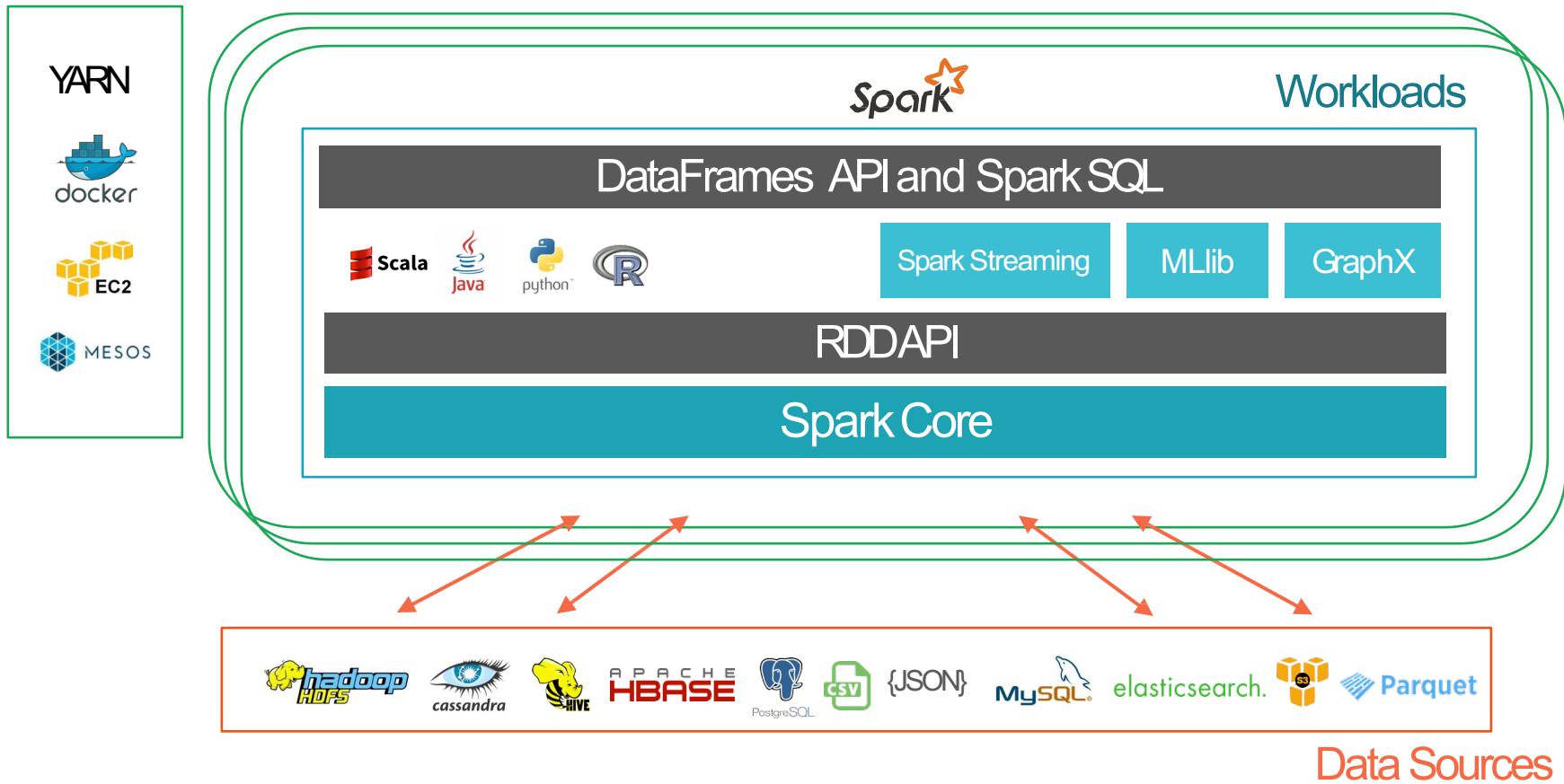
**Memory-locality** key to interactive response times

# Một nền tảng xử lý dữ liệu hợp nhất cho dữ liệu lớn

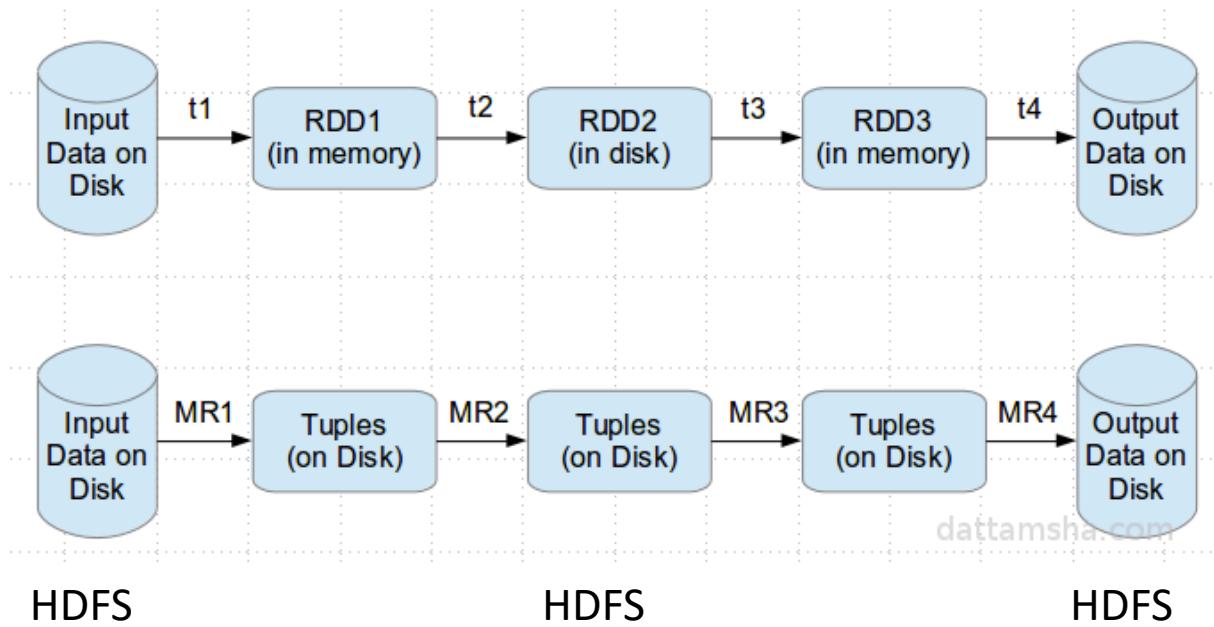
- Hỗ trợ tốt hơn MapReduce trong
  - Các giải thuật có tính lặp - Iterative algorithms
  - Khai phá dữ liệu trong môi trường tương tác - Interactive data mining
- Khả năng chịu lỗi, khai thác tính địa phương của dữ liệu, tính khả mở
- Âm đi sự phức tạp của môi trường phân tán khi lập trình



# Environments



# Khai thác bộ nhớ trong thay vì ổ đĩa HDD



# Sự khác nhau giữa Spark và MapReduce

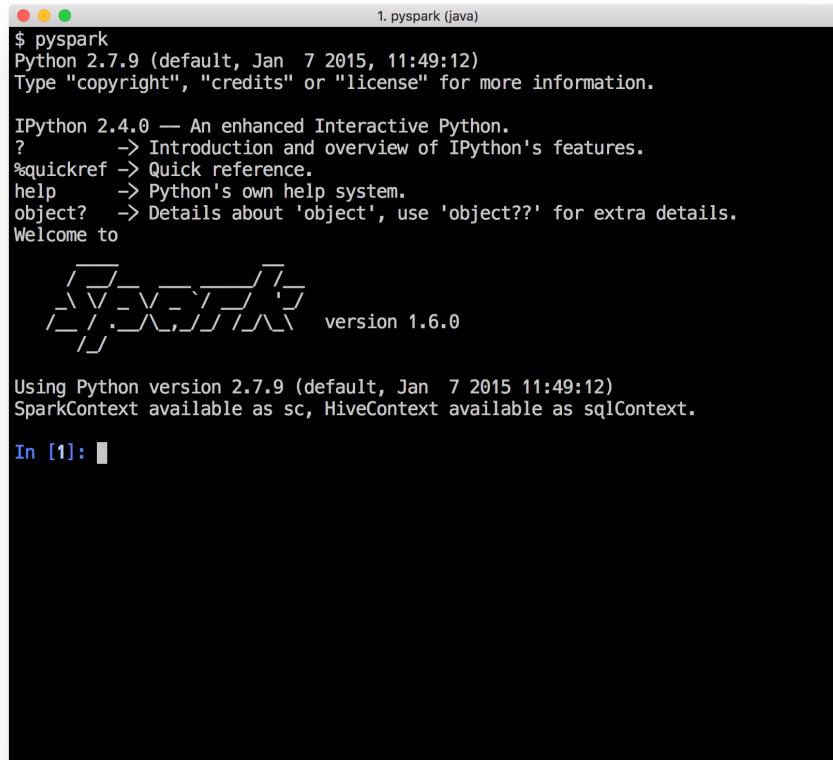
	Apache Hadoop MR	Apache Spark
Storage	Chỉ sử dụng HDD	Sử dụng cả bộ nhớ trong và HDD
Operations	Hai thao tác Map và Reduce	Bao gồm nhiều thao tác biến đổi (transformations) và hành động (actions) trên dữ liệu
Execution model	Xử lý theo khối – batch	Theo khối, tương tác , luồng
Languages	Java	Scala, Java, Python và R

# So sánh hiệu năng Spark và MapReduce

	Hadoop World Record	Spark 100 TB *	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	6592	6080
# Reducers	10,000	29,000	250,000
Rate	1.42 TB/min	4.27 TB/min	4.27 TB/min
Rate/node	0.67 GB/min	20.7 GB/min	22.5 GB/min
Sort Benchmark Daytona Rules	Yes	Yes	No
Environment	dedicated data center	EC2 (i2.8xlarge)	EC2 (i2.8xlarge)

<https://databricks.com/blog/2014/10/10/spark-petabyte-sort.html>

# Giao diện dòng lệnh tương tác



```
$ pyspark
Python 2.7.9 (default, Jan  7 2015, 11:49:12)
Type "copyright", "credits" or "license" for more information.

IPython 2.4.0 — An enhanced Interactive Python.
?           -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help        -> Python's own help system.
object?     -> Details about 'object', use 'object??' for extra details.
Welcome to

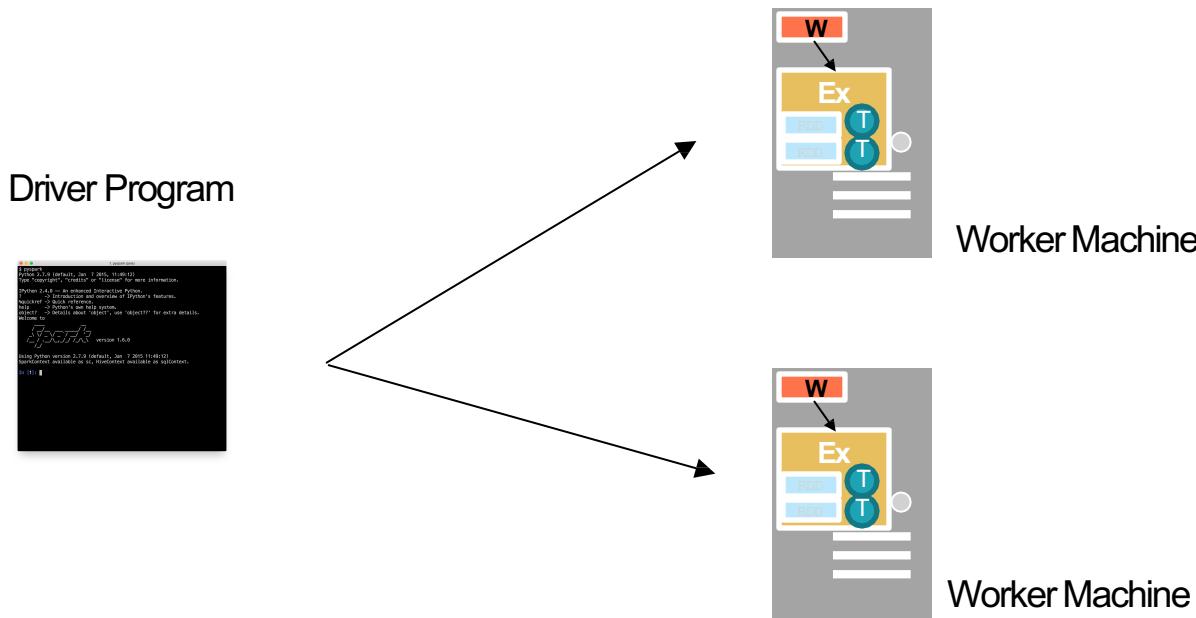
          _/\_ 
         / \ \_ 
        /   \ \_ 
       /     \ \_ 
      /       \ \_ 
     /         \ \_ 
    /           \ \_ 
   /             \ \_ 
  /               \ \_ 
 /                 \ \_ 
version 1.6.0

Using Python version 2.7.9 (default, Jan  7 2015 11:49:12)
SparkContext available as sc, HiveContext available as sqlContext.

In [1]:
```

(Scala, Python and Ronly)

# Thực thi chương trình Spark



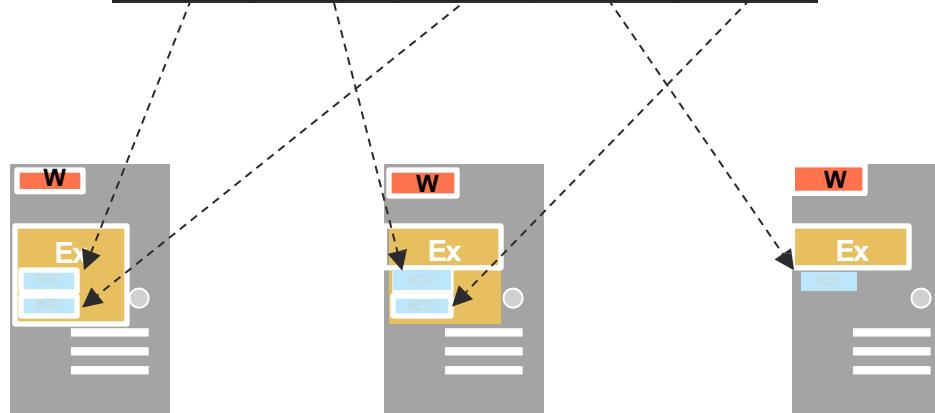
# Resilient Distributed Dataset (RDD)

- RDDs là cấu trúc dữ liệu song song, có khả năng chịu lỗi (***fault-tolerant, parallel data structures***) mà cho phép người dùng chỉ định lưu trữ dữ liệu trung gian trên bộ nhớ (***intermediate results in memory***), điều khiển sự phân chia để tối ưu hóa việc phân tán dữ liệu, và cũng có thể thay đổi, chỉnh sửa những dữ liệu này sử dụng một tập các thao tác rất đa dạng (***a rich set of operators***).
  - RDDs có khả năng tự động tái tạo lại khi bị lỗi
- RDD được thiết kế tối ưu cho các biến đổi thô, theo lô (coarse-grained transformations) thay vì hỗ trợ các thao tác cập nhật quá chi tiết (fine-grained updates)
  - Vd., map, filter và join mà tác động tới nhiều bản ghi dữ liệu đồng thời thay vì là các thao tác chỉ cập nhật lên một đối tượng dữ liệu riêng lẻ

# Sự phân vùng của RDD và khả năng song song hóa

RDD

item-1	item-6	item-11	item-16	item-21
item-2	item-7	item-12	item-17	item-22
item-3	item-8	item-13	item-18	item-23
item-4	item-9	item-14	item-19	item-24
item-5	item-10	item-15	item-20	item-25



# Khởi tạo RDD

- Một RDD cơ sở có thể được tạo theo 2 cách
  - Song song hóa một collection (ví dụ mảng trong Python)
  - Đọc dữ liệu từ một nguồn bên ngoài (S3, C\*, HDFS, etc)

logLinesRDD			
Error, ts, msg1 Warn, ts, msg2 Error, ts, msg1	Info, ts, msg8 Warn, ts, msg2 Info, ts, msg8	Error, ts, msg3 Info, ts, msg5 Info, ts, msg5	Error, ts, msg4 Warn, ts, msg9 Error, ts, msg1

# Phương thức Parallelize



```
// Parallelize in Scala  
val wordsRDD = sc.parallelize(List("fish", "cats", "dogs"))
```

---

- Một mảng làm thm số đầu vào cho phương thức parallelize của SparkContext



```
# Parallelize in Python  
wordsRDD = sc.parallelize(["fish", "cats", "dogs"])
```

---

- Thường được sử dụng để thử nghiệm vì đòi hỏi toàn bộ dữ liệu phải có sẵn trên bộ nhớ trong.



```
// Parallelize in Java  
JavaRDD<String> wordsRDD = sc.parallelize(Arrays.asList("fish", "cats", "dogs"));
```

# RDD khởi tạo từ tệp tin văn bản



```
// Read a local txt file in Scala  
val linesRDD = sc.textFile("/path/to/README.md")
```

---

Ngoài ra có các phương thức đọc từ HDFS, C\*, S3, HBase, etc.



```
# Read a local txt file in Python  
linesRDD = sc.textFile("/path/to/README.md")
```

---

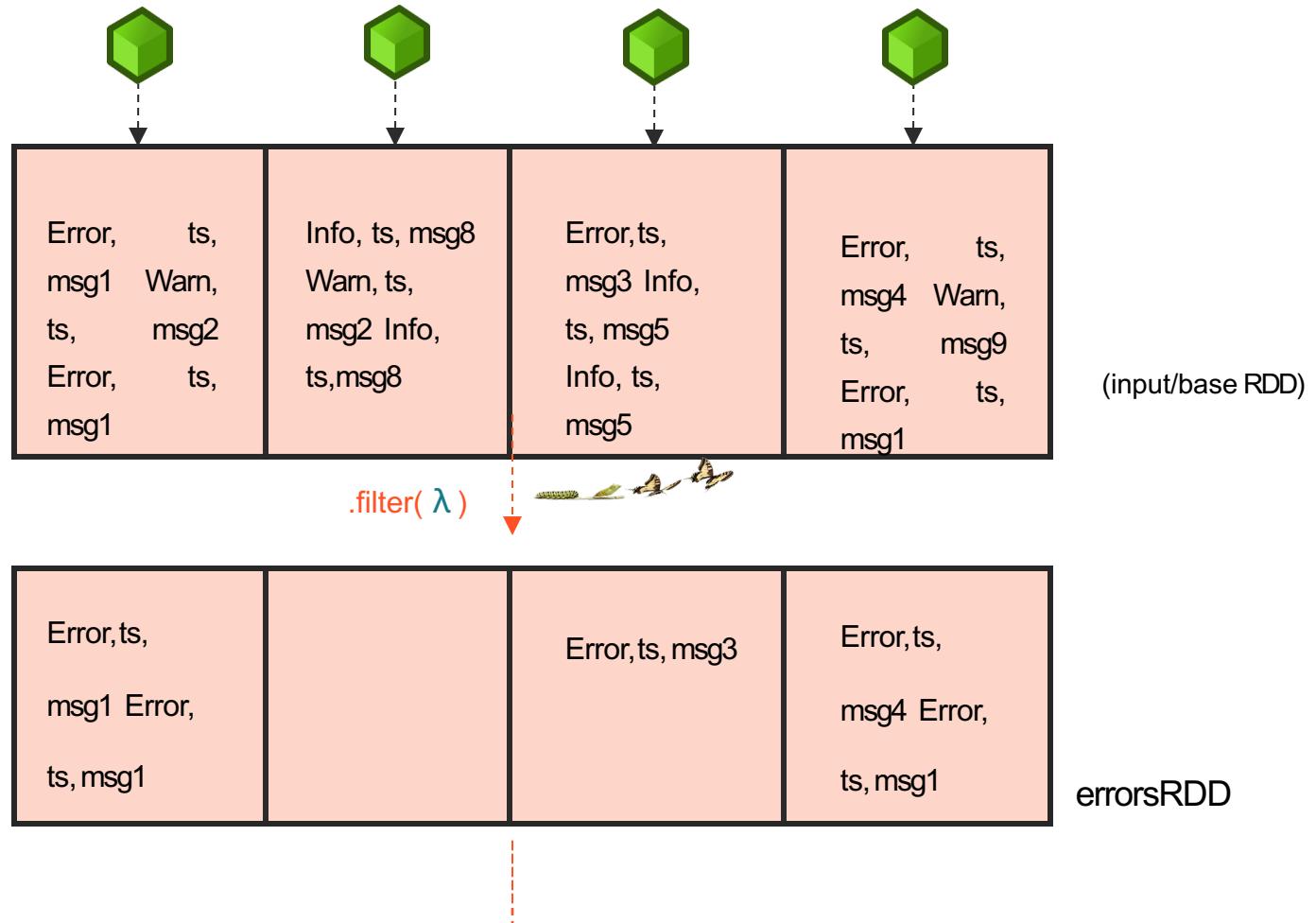


```
// Read a local txt file in Java  
JavaRDD<String> lines = sc.textFile("/path/to/README.md");
```

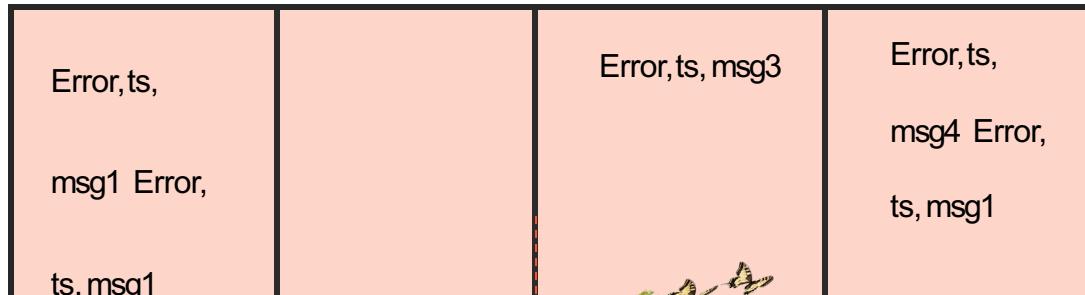
# Các thao tác trên RDD

- 2 dạng thao tác: **transformations and actions**
- Transformations là lazy (không được tính toán ngay lập tức)
- Transformations chỉ được thực thi khi một action được gọi
- Có thể lưu trữ lâu dài (hoặc tạm thời) dữ liệu của RDD trên RAM và cứng

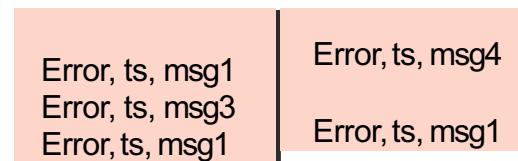
# logLinesRDD



# errorsRDD

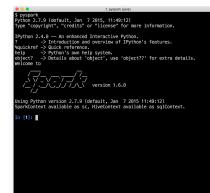


.coalesce( 2 )



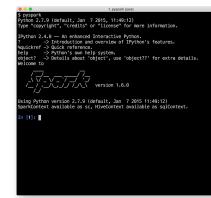
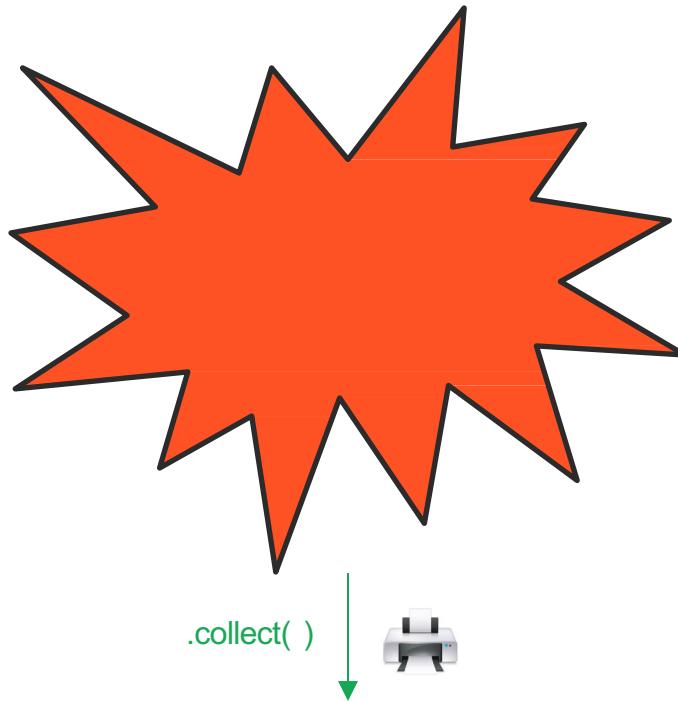
cleanedRDD

.collect( )



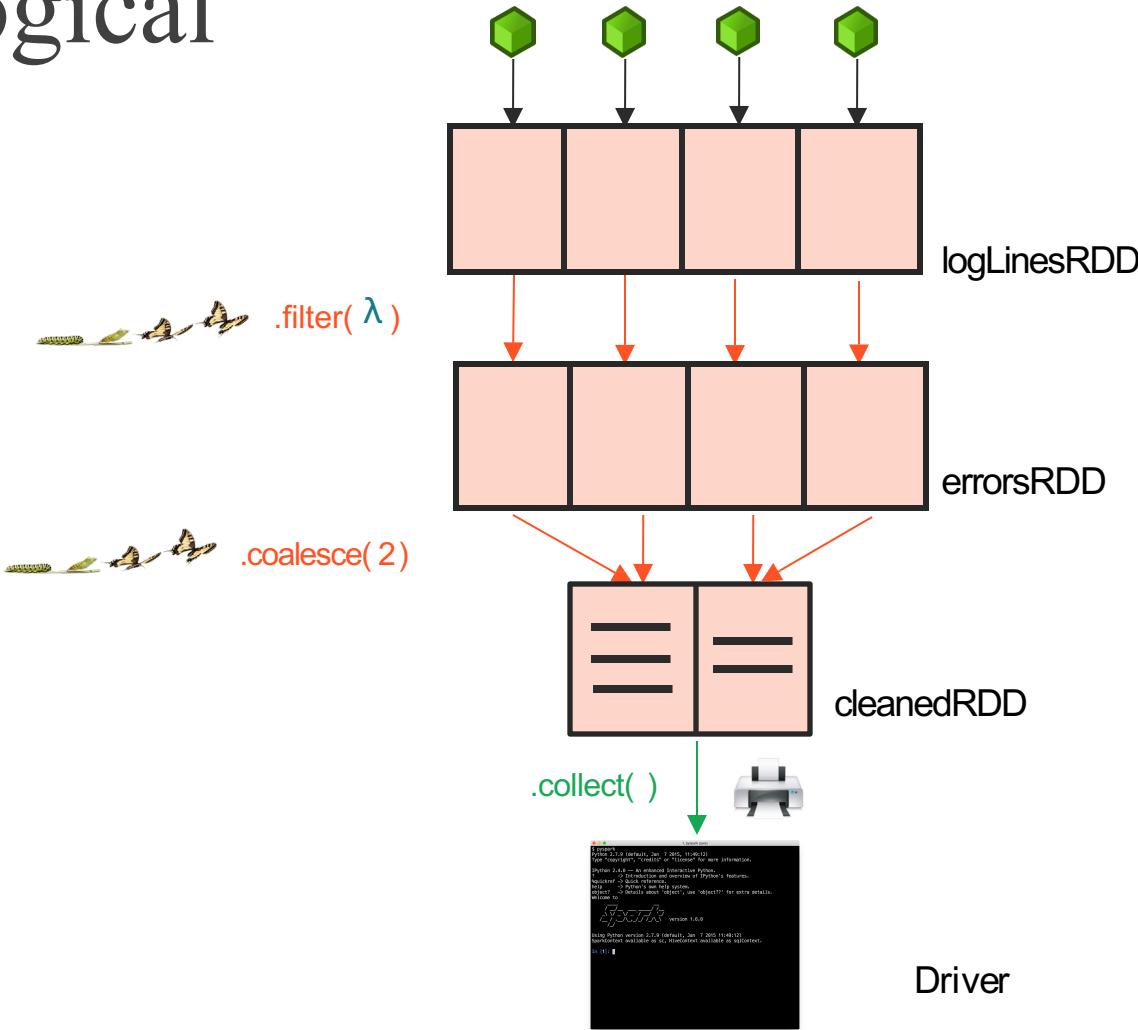
Driver

# Execute DAG!

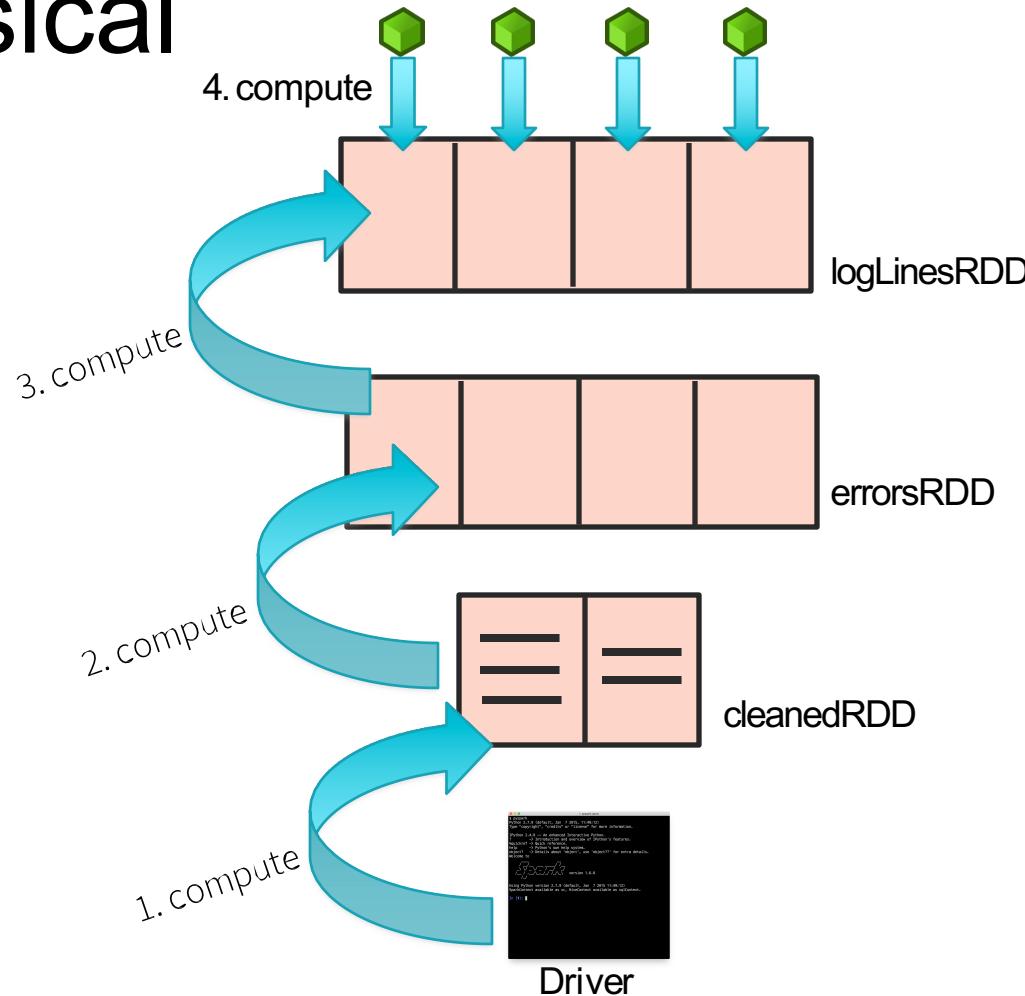


Driver

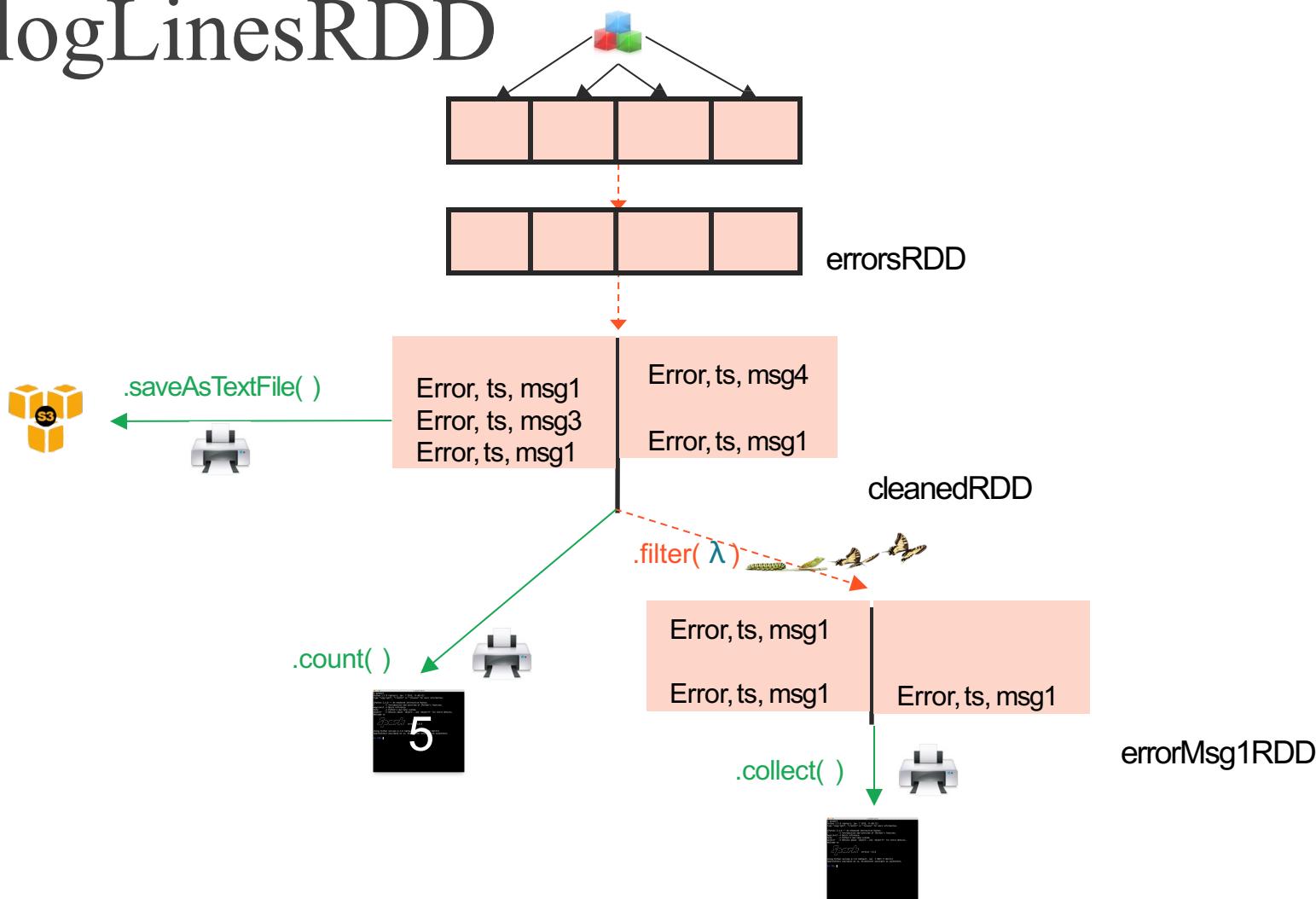
# Logical



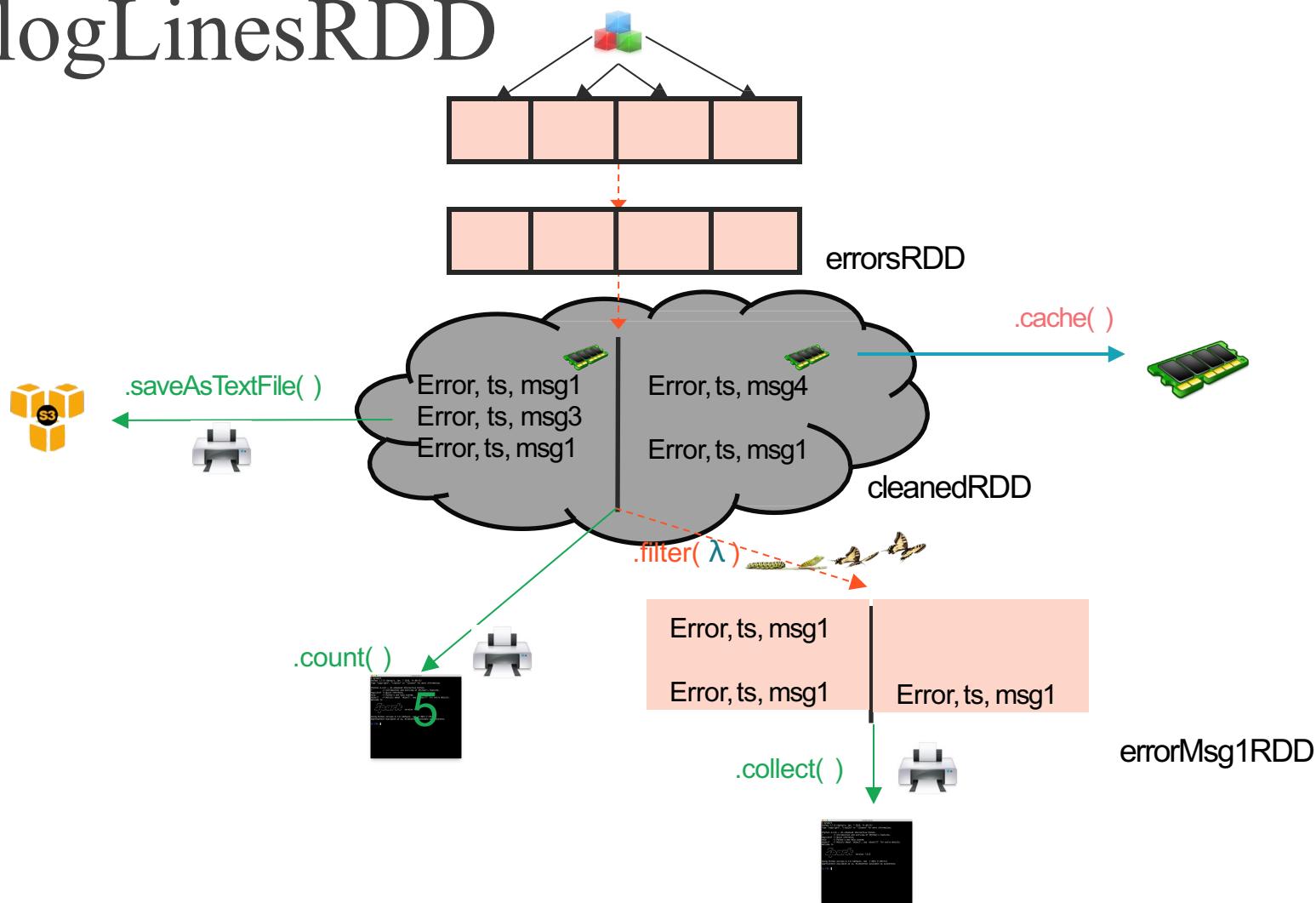
# Physical



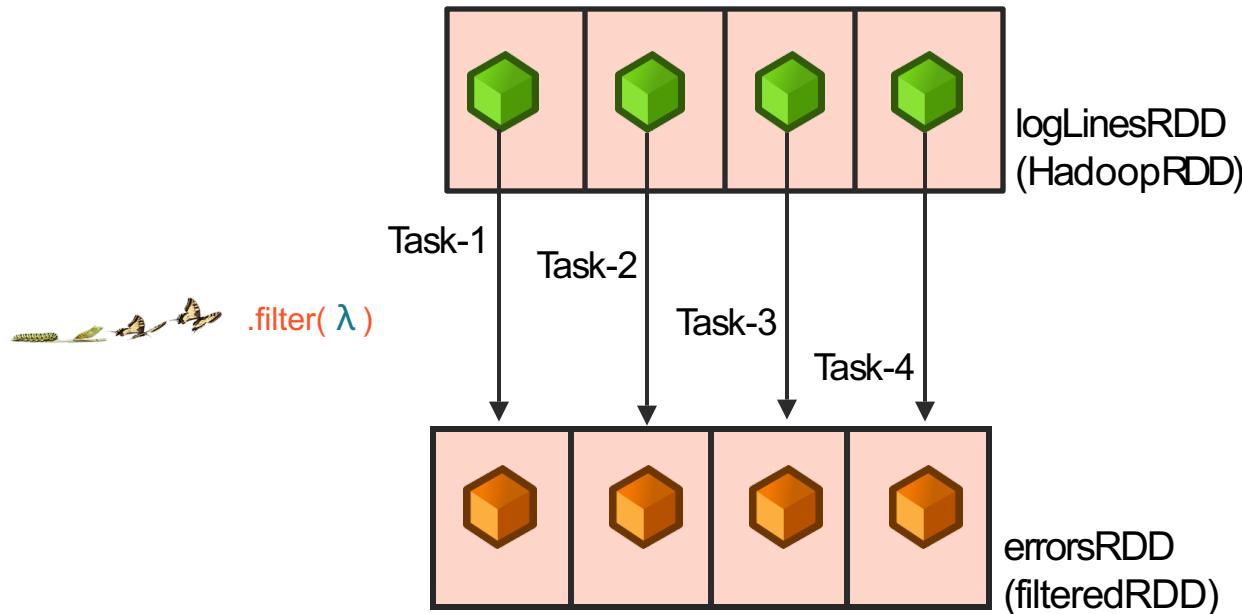
# logLinesRDD



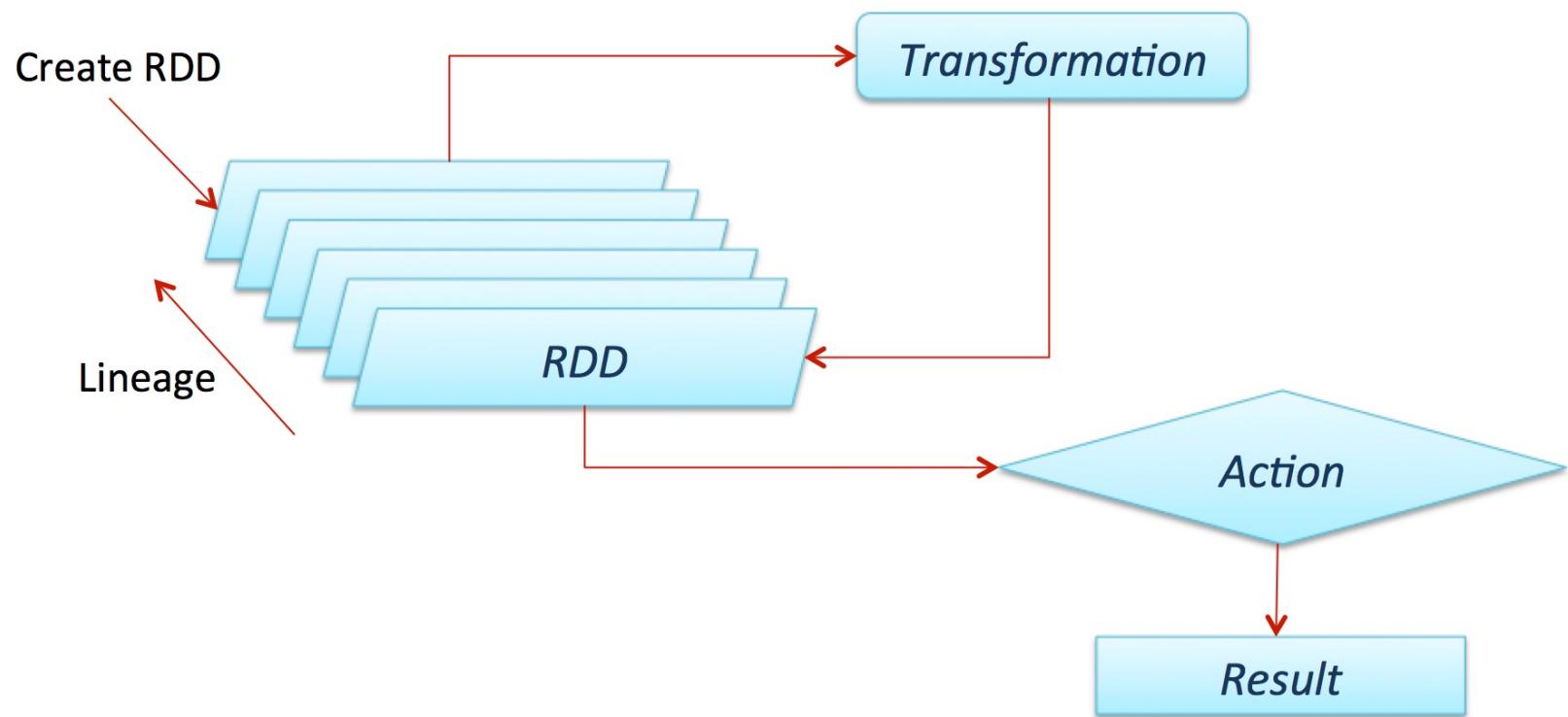
# logLinesRDD



# Partition >> Task >> Partition



# Chịu lối sử dụng kỹ thuật Lineage



# Một vài ghi nhớ về RDD

- Khởi tạo RDD từ disks (HDFS, etc)
- RDD trung gian nằm trên RAM
- Khôi phục lỗi sử dụng lineage
- Các thao tác trên RDD là phân tán

# Quy trình lập trình trên Spark

- Tạo RDD từ nguồn dữ liệu bên ngoài hoặc từ các collections trong bộ nhớ
- Lazily transform thành các RDD trung gian
- cache() RDD để tái sử dụng
- Gọi các actions để thực thi tính toán song song và nhận về các kết quả

# Transformations (lazy)

map()

intersection()

cartesian()

flatMap()

distinct()

pipe()

filter()

groupByKey()

coalesce()

mapPartitions()

reduceByKey()

repartition()

mapPartitionsWithIndex()

sortByKey()

partitionBy()

sample()

join()

...

union()

cogroup()

...

# Actions

reduce()

takeOrdered()

collect()

saveAsTextFile()

count()

saveAsSequenceFile()

first()

saveAsObjectFile()

take()

countByKey()

takeSample()

foreach()

saveToCassandra()

...

# Dataframe

- Là một lớp trừu tượng hóa dữ liệu chính của Spark 2.0
  - Có tính bất biến, không thay đổi được sau khi được khởi tạo
  - Lưu vết lịch sử các thay đổi theo cơ chế lineage để tái tạo lại khi bị lỗi
  - Cho phép thực thi các thao tác trên tập dữ liệu trên cơ chế song song hóa
- Để khởi tạo các Dataframe
  - Song song hóa các collections trong bộ nhớ (vd. List, set)
  - Biến đổi từ một dataframe đã có trên Spark hoặc từ pandas
  - Từ các tệp tin trên HDFS hoặc các hệ thống lưu trữ khác

	Job	U-Tot	U-Avg	D-Tot	D-Avg	M-Tot	M-Avg	S-Tot	S-Avg
	admin.	24.0	1.71	3398.0	2.65	13807.0	2.63	10113.0	2.61
	blue-collar	54.0	3.86	1893.0	2.6	17051.0	2.55	4678.0	2.56
	entrepreneur	11.0	3.67	485.0	2.71	2664.0	2.49	532.0	2.62
	housemaid	17.0	5.67	396.0	2.46	2089.0	2.69	296.0	2.49
	management	16.0	5.33	866.0	2.62	5192.0	2.49	1166.0	2.33
	retired	17.0	3.4	865.0	2.49	3110.0	2.44	268.0	2.88
	self-employed	11.0	2.2	341.0	2.56	2525.0	2.79	904.0	2.39
	services	33.0	5.5	1345.0	2.53	5903.0	2.57	2990.0	2.63
	student	1.0	1.0	19.0	2.11	112.0	2.73	1709.0	2.07
	technician	21.0	1.75	2113.0	2.73	9348.0	2.55	5897.0	2.58
	unemployed	11.0	2.2	303.0	2.44	1701.0	2.68	585.0	2.33
	unknown	39.0	4.33	29.0	2.23	633.0	2.71	173.0	2.34

# Machine Learning Library (MLlib)

- 2 packages
  - spark.mllib
  - spark.ml
- Các giải thuật học máy
  - Bao gồm các giải thuật phổ biến như phân lớp, hồi quy, phân nhóm, lọc cộng tác
- Xây dựng đặc trưng - Featurization
  - Trích rút, biến đổi, giảm chiều, lựa chọn các đặc trưng
- Các tiện ích
  - Đại số tuyến tính, thống kê, ...

# ML: Transformer

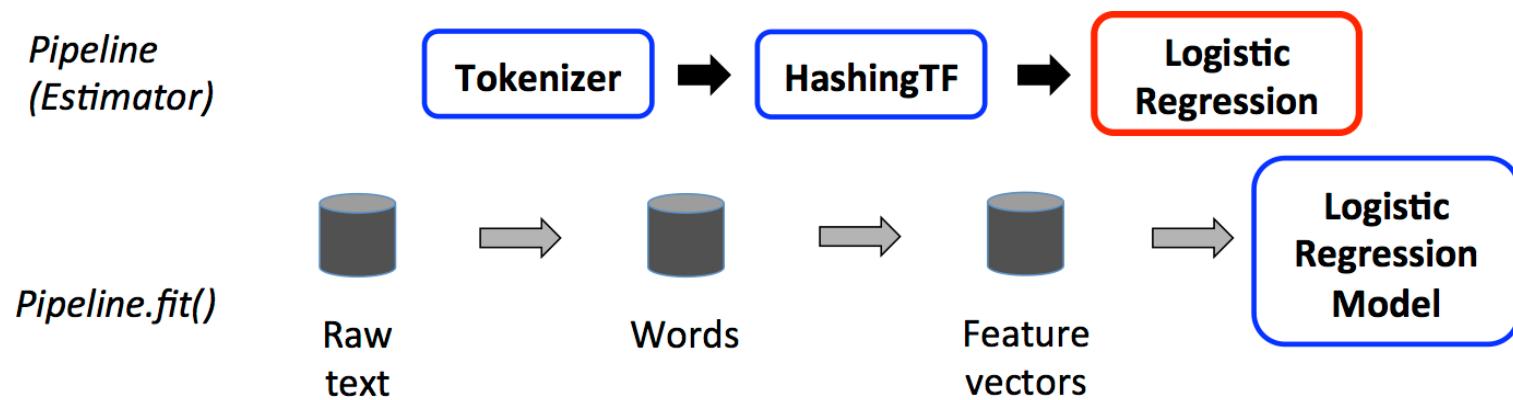
- Transformer là một lớp mà có thể tạo ra dataframe mới bằng cách biến đổi một dataframe ban đầu
- Transformer cài đặt phương thức transform()
- Ví dụ
  - HashingTF
  - LogisticRegressionModel
  - Binarizer

# ML: Estimator

- Estimator là lớp mà lấy đầu vào là 1 dataframe và trả về một transformer
- Estimator cài đặt phương thức fit()
- Ví dụ
  - LogisticRegression
  - StandardScaler
  - Pipeline

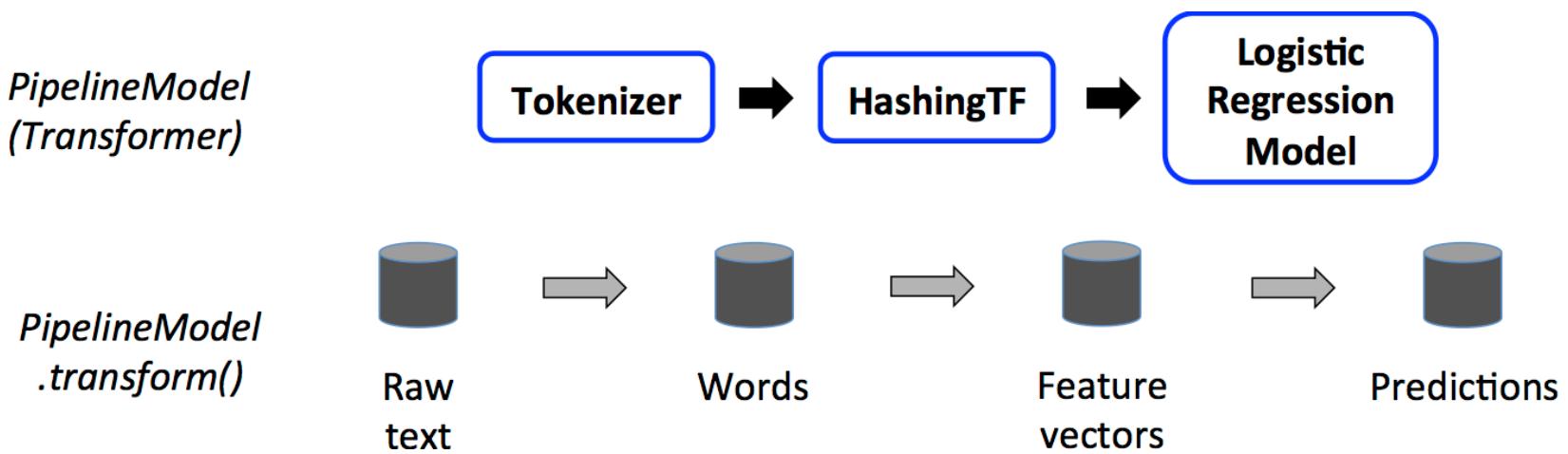
# ML: Pipeline

- Pipeline là một estimator mà bao gồm 1 chuỗi các màn (stage) mà mỗi stage có thể là estimator hoặc là transformer



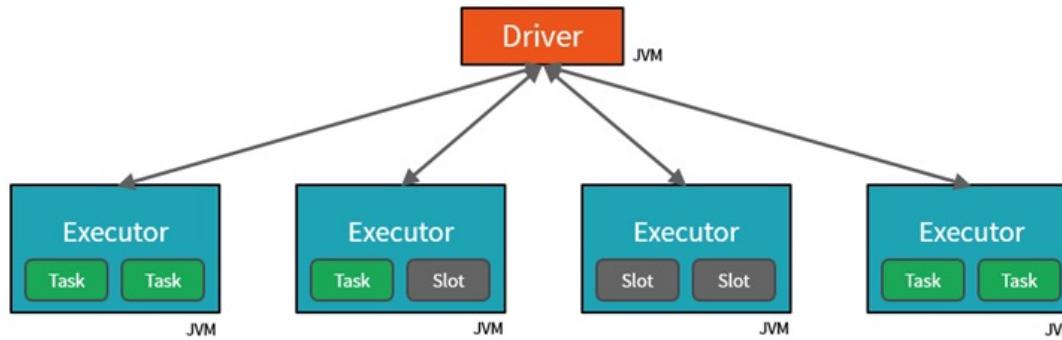
# ML: PipelineModel

- Kết quả sau khi gọi Pipeline.fit() trả về một PipelineModel. PipelineModel được sử dụng khi kiểm thử.



# Kiến trúc chung

- Kiến trúc master worker
  - 1 driver hay master node
  - Nhiều Worker nodes



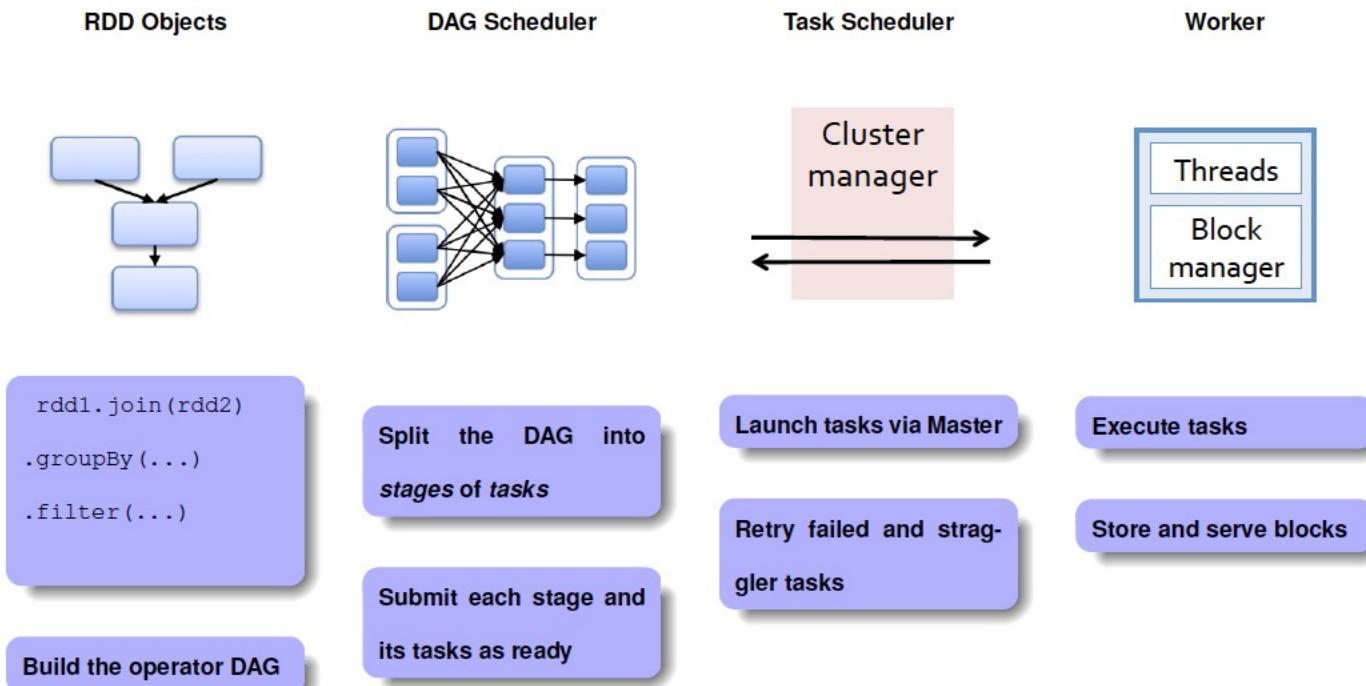
- Master gửi các công việc tới các workers và chỉ định lấy dữ liệu từ bộ nhớ trong hoặc bên ngoài (vd. S3 hoặc HDFS)

# Kiến trúc chung (2)

- Một chương trình Spark program đầu tiên cần tạo một `SparkContext` object
  - `SparkContext` chỉ định cách Spark sử dụng các tài nguyên tính toán được cấp phát
  - Tham số cho `SparkContext` là kiểu và kích thước của cụm tính toán được sử dụng

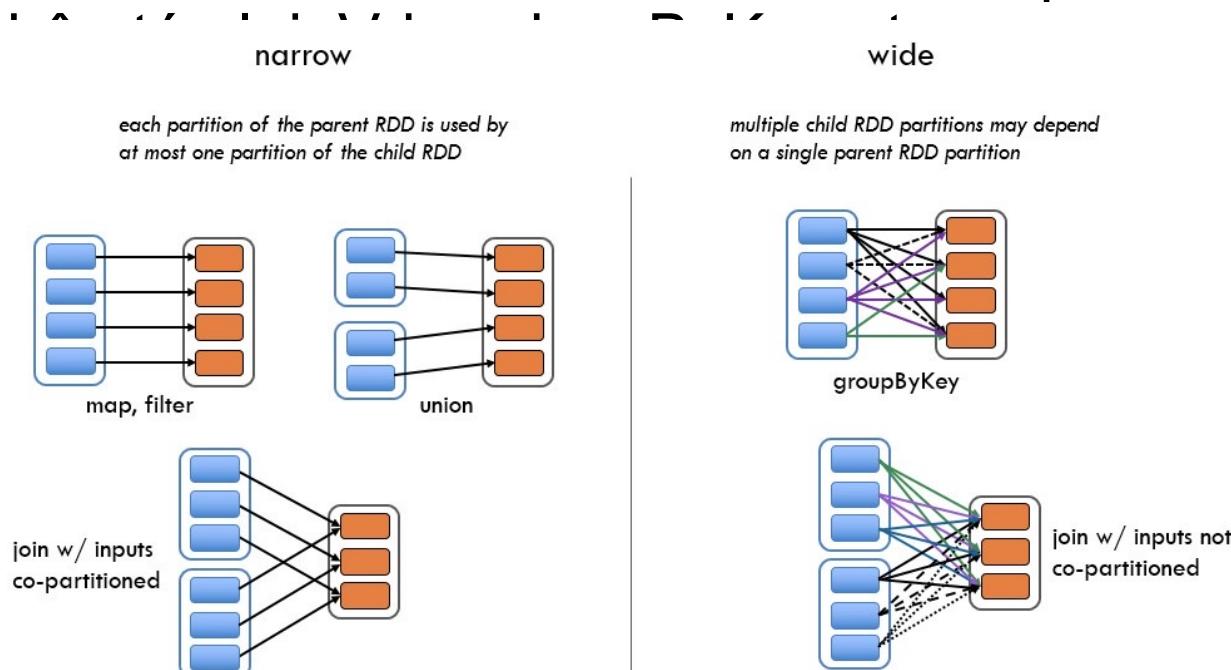
Master parameter	Description
local	Chạy cục bộ, sử dụng 1 luồng tính toán
local[K]	Chạy cục bộ sử dụng song song K luồng tính toán
spark://HOST:PORT	Kết nối tới một Spark standalone cluster
mesos://HOST:PORT	Kết nối tới một Mesos cluster
yarn	Kết nối tới một YARN cluster

# Chu trình một công việc trên Spark



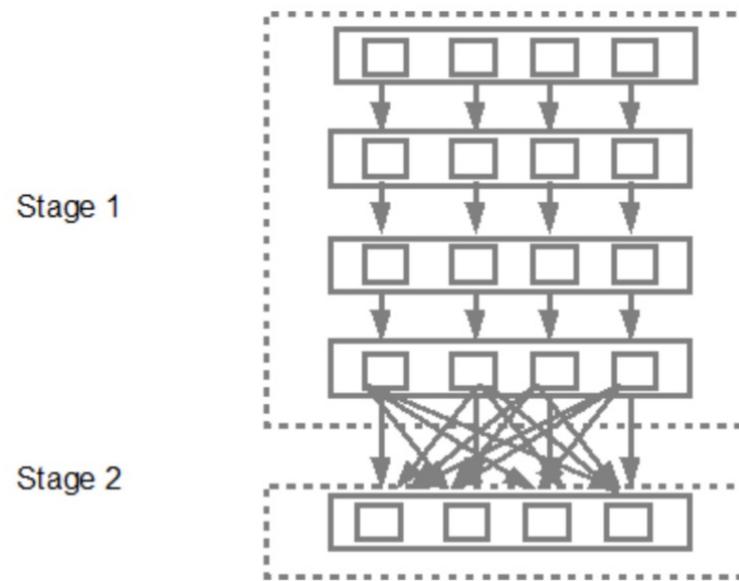
# Phụ thuộc dạng Narrow hoặc Wide

- Có 2 dạng biến đổi
  - Narrow transformation—Không đòi hỏi dữ liệu phải được phân tán lại giữa các phân vùng partitions. Vd. Map, filter etc..
  - Wide transformation—đòi hỏi dữ liệu phải buộc



# Mối quan hệ giữa Spark transformations và stages

- Các narrow transformations sẽ được nhóm lại trong cùng một stage



# Tổng kết

- Hadoop: Hệ sinh thái lưu trữ và xử lý dữ liệu lớn kinh tế và khả mở
- Spark: Nền tảng phân tích dữ liệu hợp nhất cho dữ liệu lớn