Distributed Computing for BigData

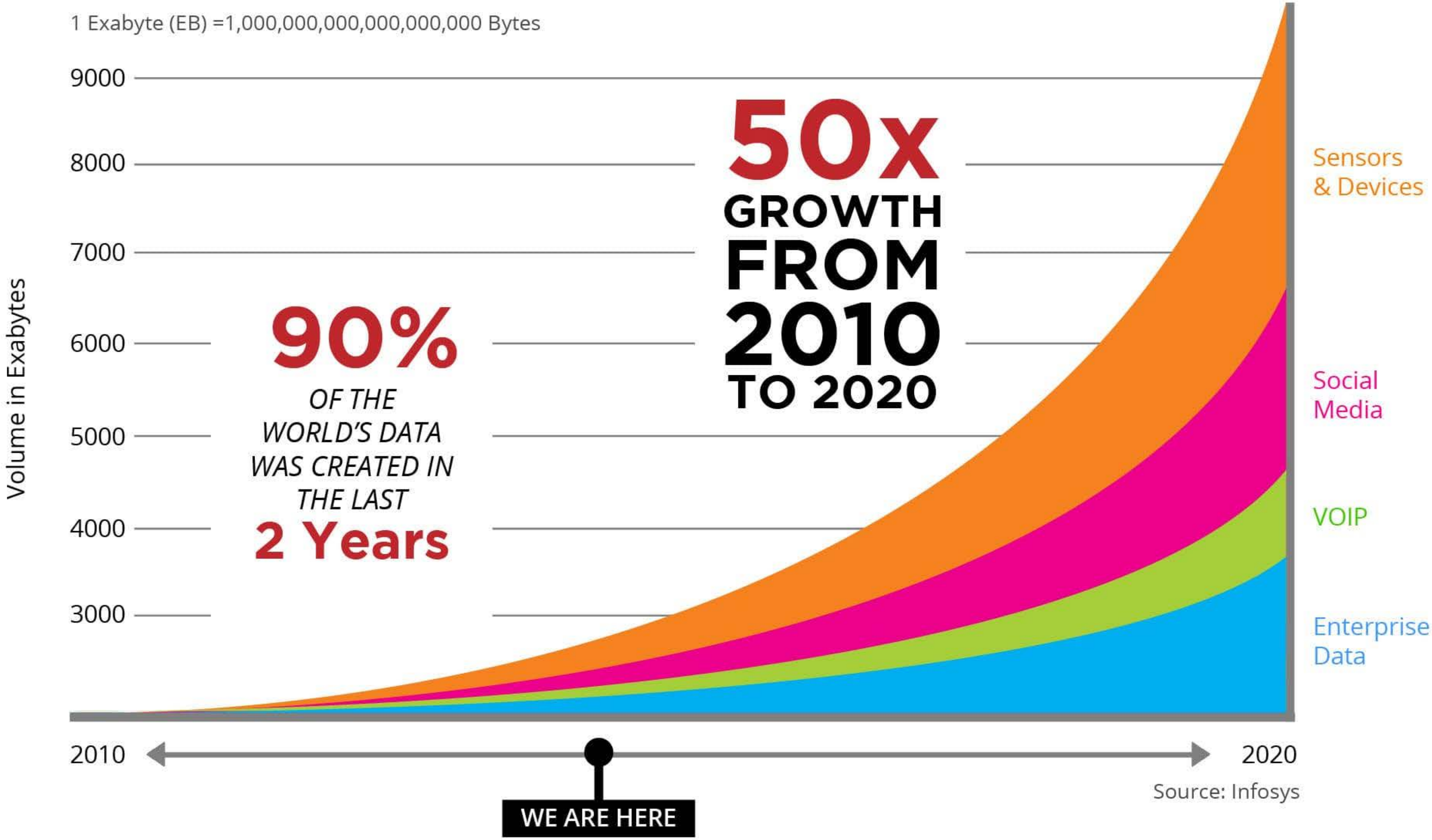
National Tsing Hua University

2019, Fall Semester

Outline

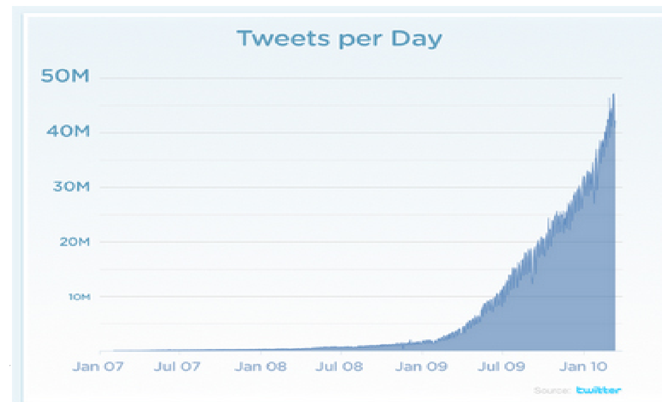
- Introduction of BigData
- Hadoop Eco-system & Current Trends
- HDFS & MapReduce
- MapReduce Applications: Information retrieval
- Hive/Pig & Spark

Data Growth


1 Exabyte (EB) =1,000,000,000,000,000,000 Bytes
9000
8000
7000
6000
5000
4000
3000
Volume in Exabytes
90%
OF THE
WORLD'S DATA
WAS CREATED IN
THE LAST
2 Years
50x
GROWTH
FROM
2010
TO 2020
Sensors
& Devices
Social
Media
VOIP
Enterprise
Data
2010
2020
WE ARE HERE
Source: Infosys

The Explosion of Data

- A increased number and variety of **data sources** that generate large quantities of data
  - Sensors(e.g. measurements)
  - Mobile devices(e.g. phone)
  - Social Network (e.g. twitter, wikis)
  - OLTP (e.g. bank transactions)




Mobile device


Sensors

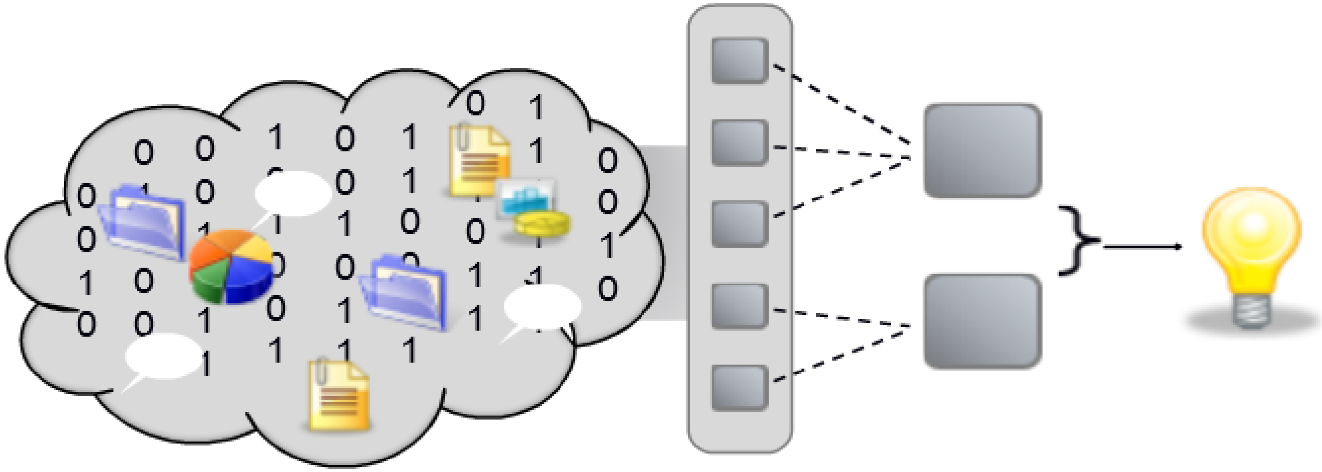

OLTP

Social Networks


Scientific Devices

The Explosion of Data

- Realize data is "too valuable" to delete
  - Diagnose system
  - Understand user behavior
  - Evaluate merchandise & products
  - Make business decision



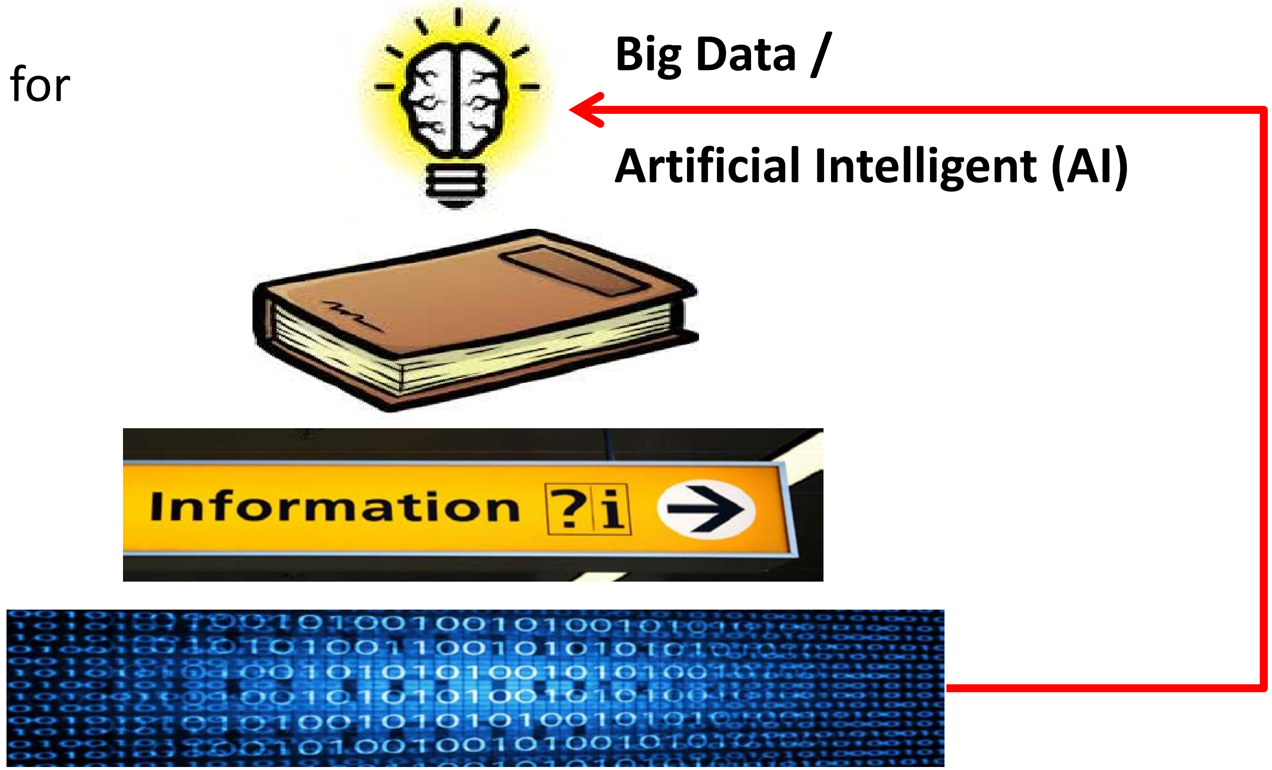
Data to Wisdom (DIKW Pyramid)

Wisdom:
Intelligent decision for creating values

Knowledge:
Analyzed info
(How & Why)

Information:
Data description
(What is it?)

Data:
Symbols or Signs



行車影像

行車車距離、號誌

駕駛方式、路徑

智慧車輛

Big Data Paradigm

Tapping into diverse data sets

Finding and monetizing unknown relationships

Data driven

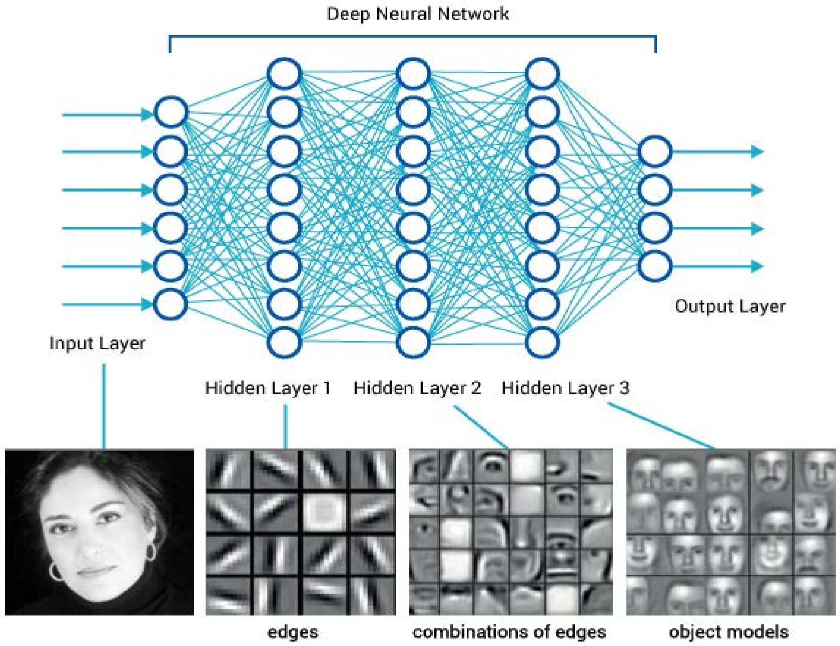
business decisions/intelligence

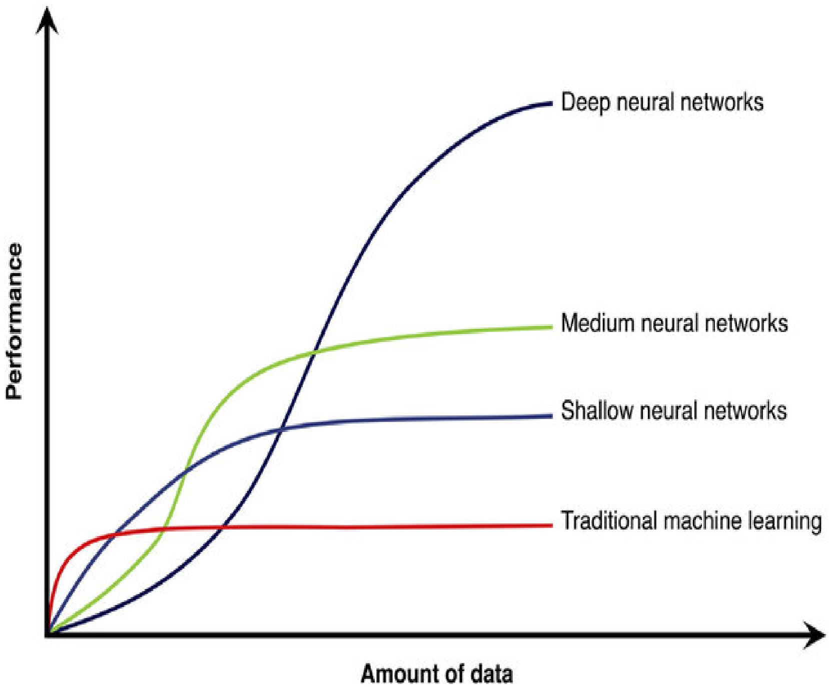
The Tales of Beers and Diapers

- A large supermarket chain, Wal-Mart, did an analysis of customers' buying habits and found a **statistically significant correlation between purchases of beer and purchases of diapers**

From BigData to Deep Learning

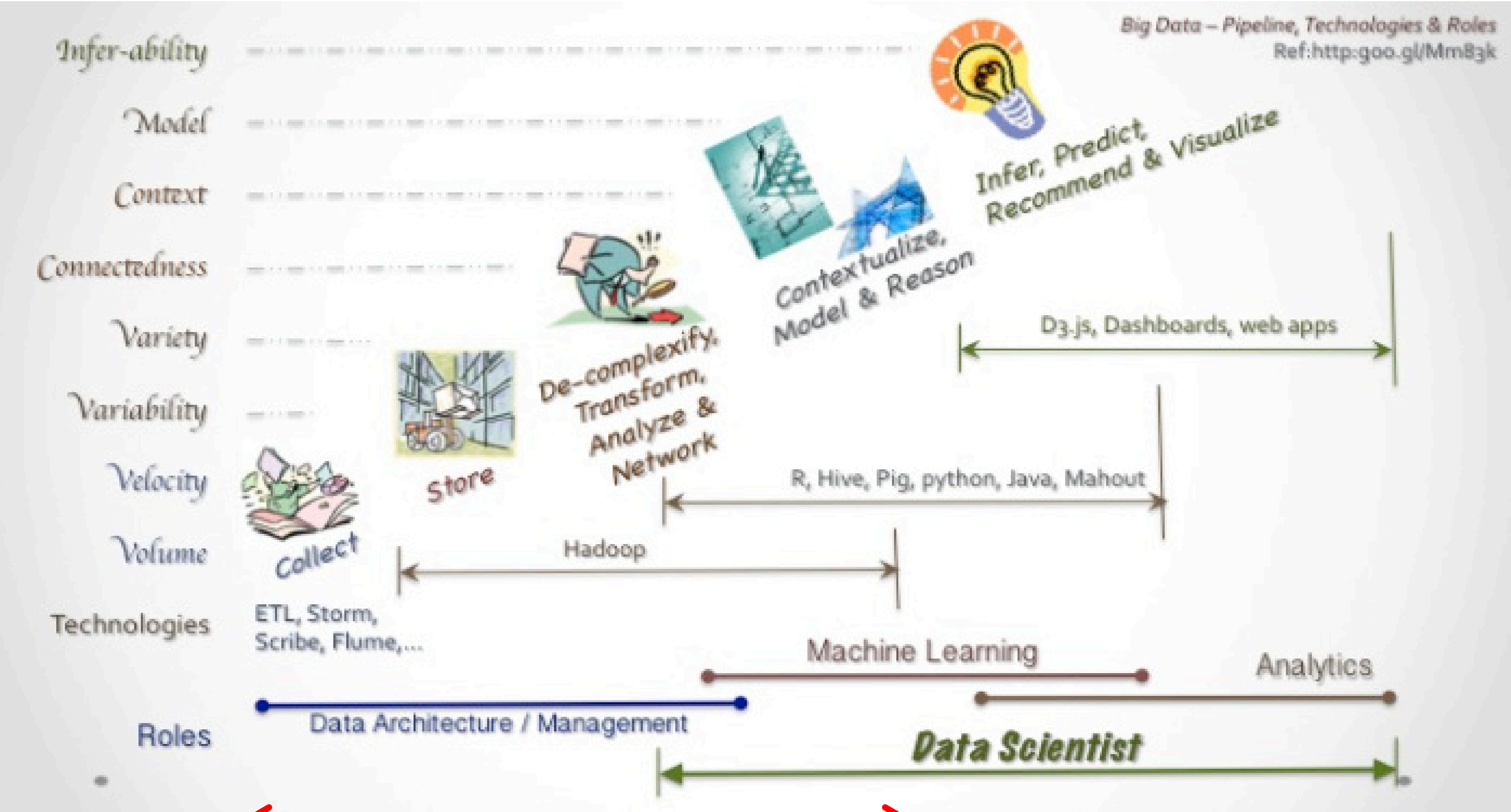
- Based on universal approximation theorem
  - A model constructed with a **greedy layer-by-layer method**, such as the artificial neural network





Source: Benoit Gallix

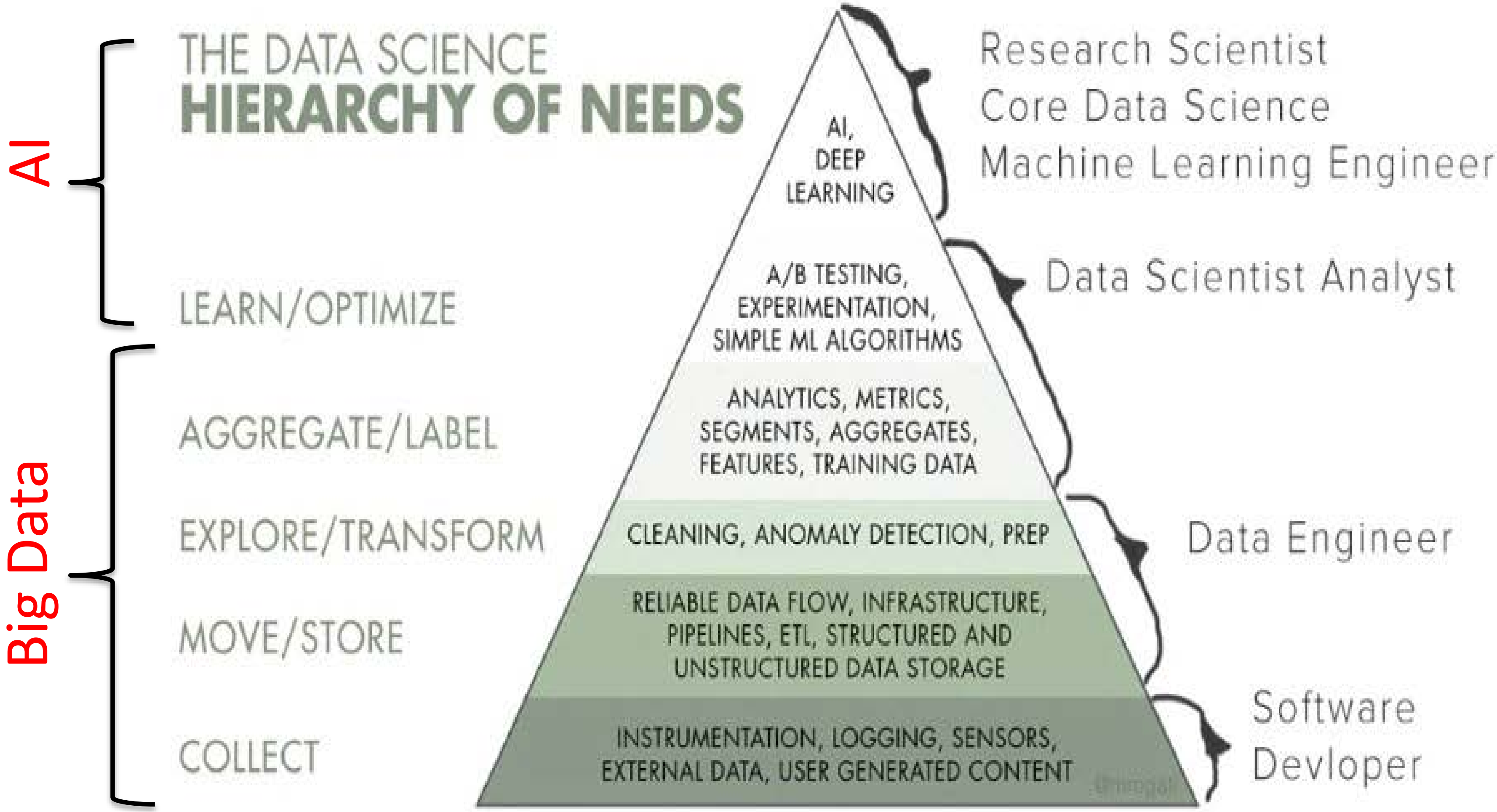
Moving from Big Data to Smart Data



Data Engineer

Source: http://tinyurl.com/atcanjw

Hierarchy of the Data Process



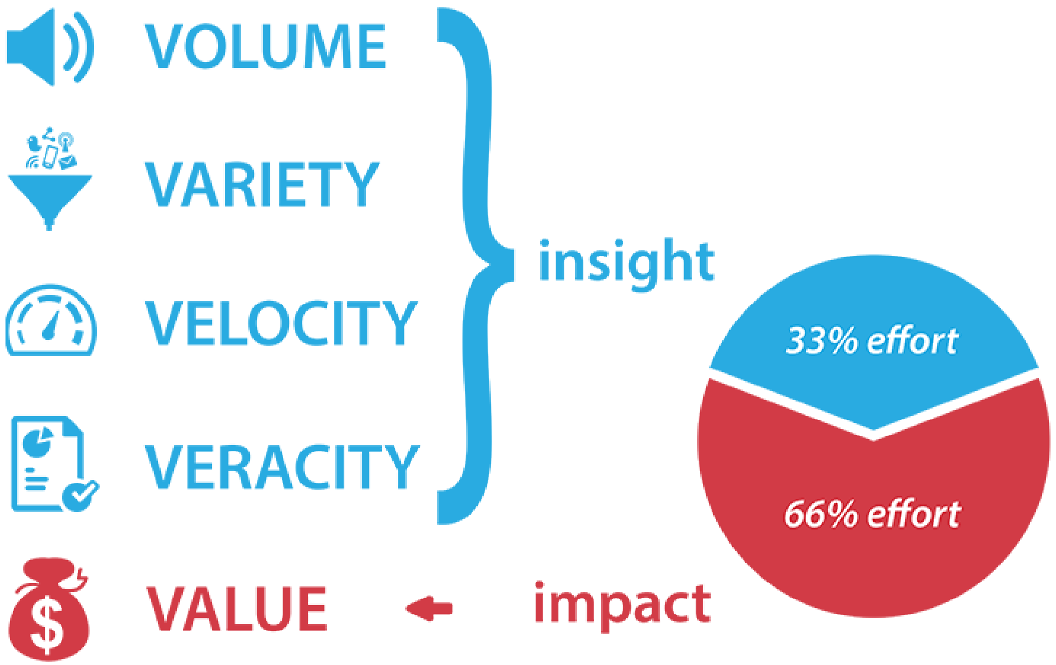
Big Data Problem

Fastest way to transmit 5MB of data in 1956

Big Data Problem

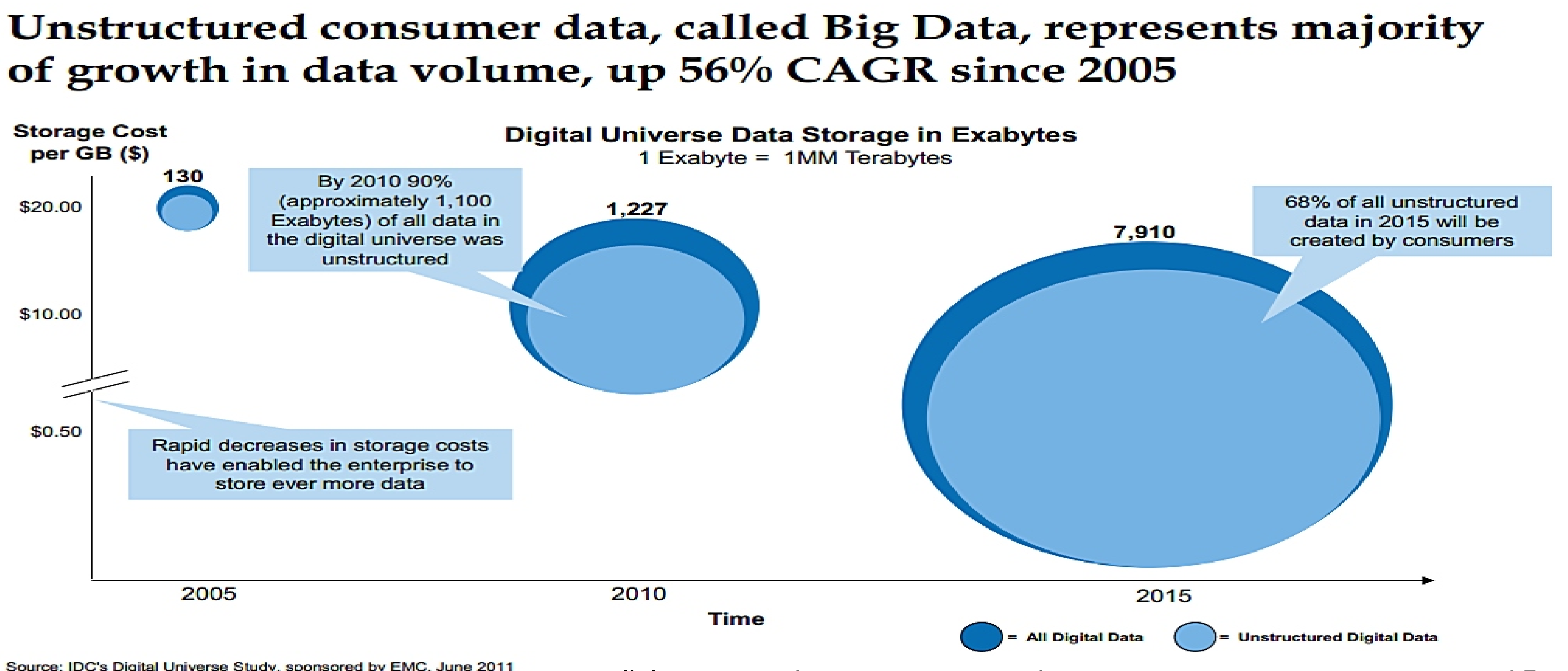
What Makes Big Data Different?

- *Extracting **values**(insight) from an immense volume, variety and velocity of data, in context, beyond what was previously possible*
- Defined by the Vs of Big Data



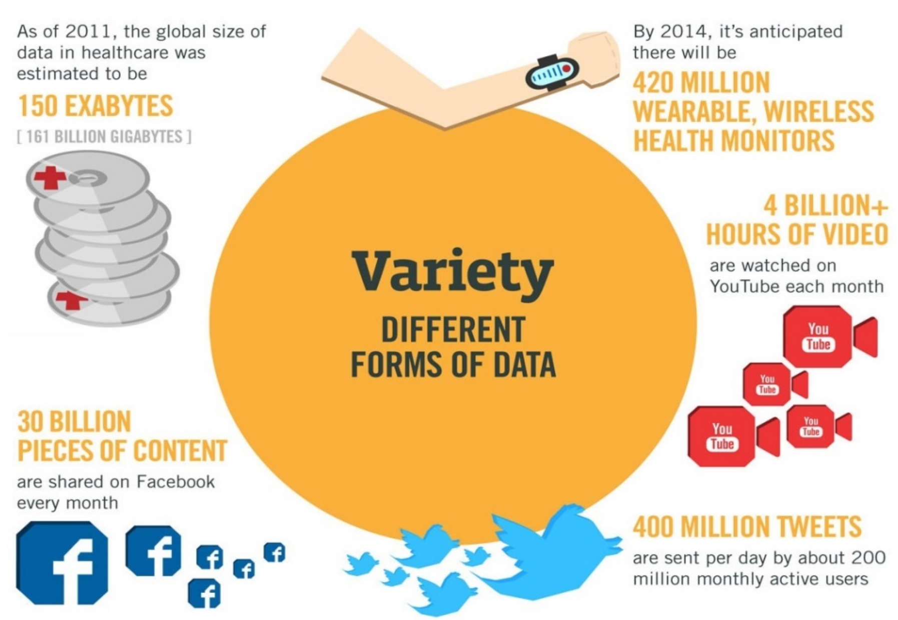
Volume: Size (大)

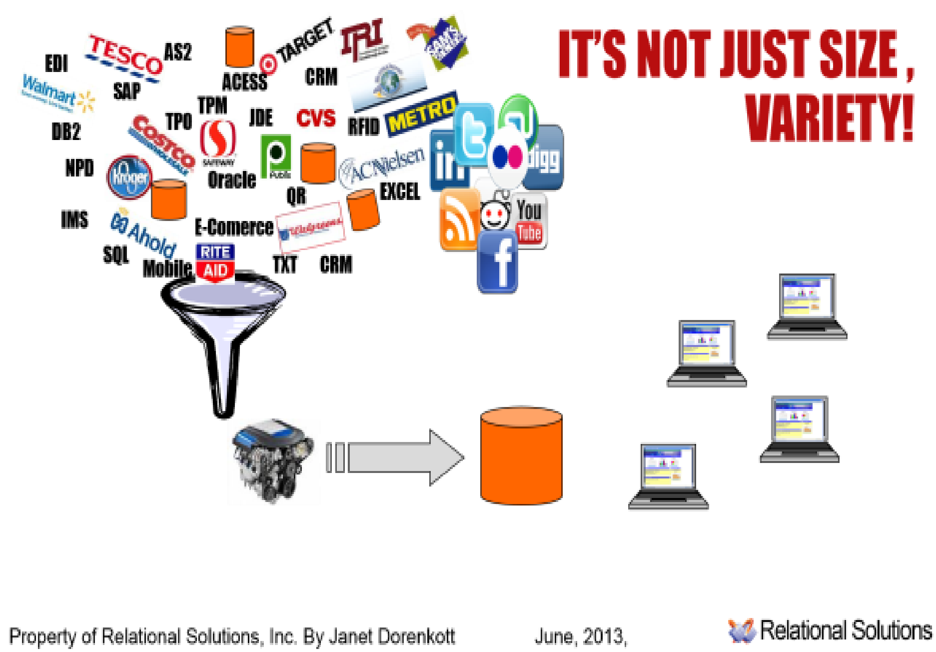
- Scale of data size grows from Terabytes to Petabytes (1K TBs) to Zetabytes (1B TBs)
  - Require more computing resource and time to process it



Variety: Complexity (雜)

- Variety defines the nature of data that exists within big data. This includes different data formats, data semantics and data structures types.

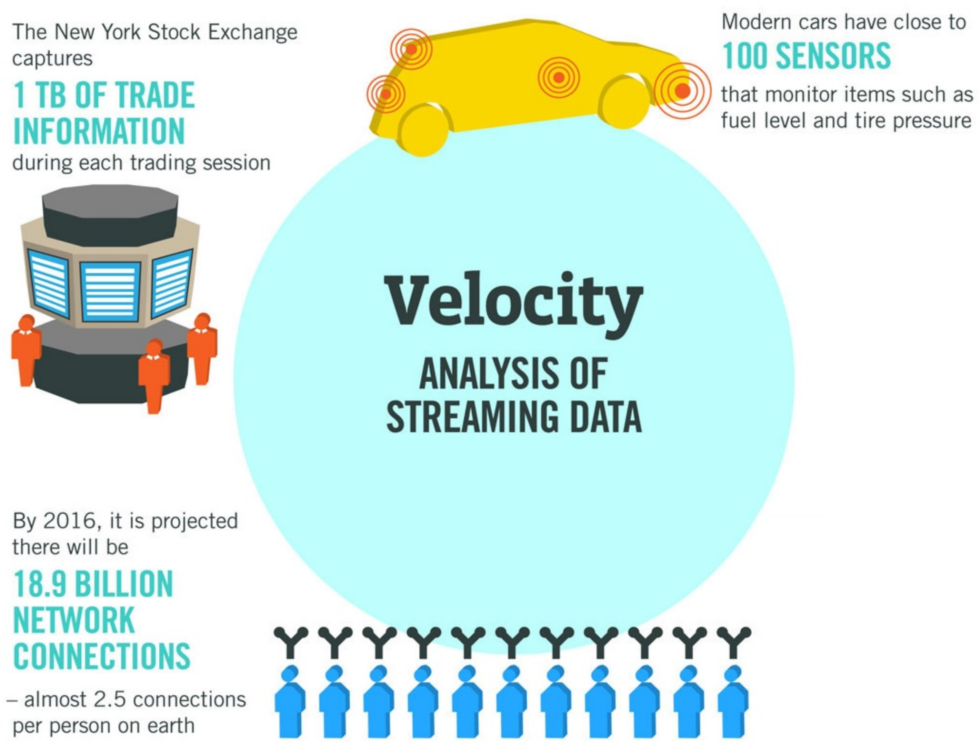


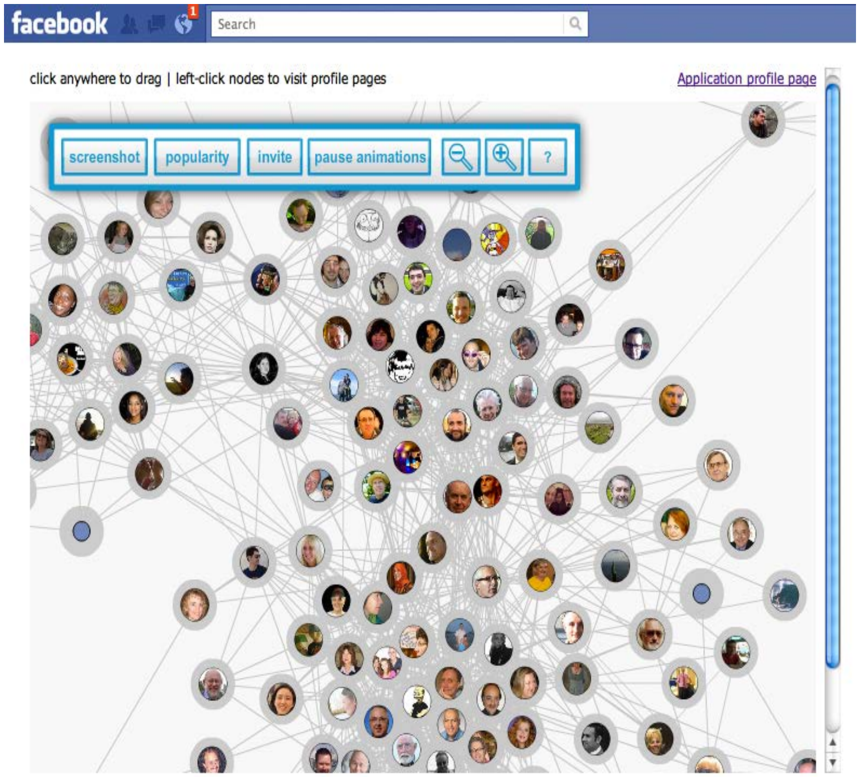


Source: https://www.mytechlogy.com/IT-blogs/7151/the-four-vs-of-big-data/

Velocity: Speed (快)

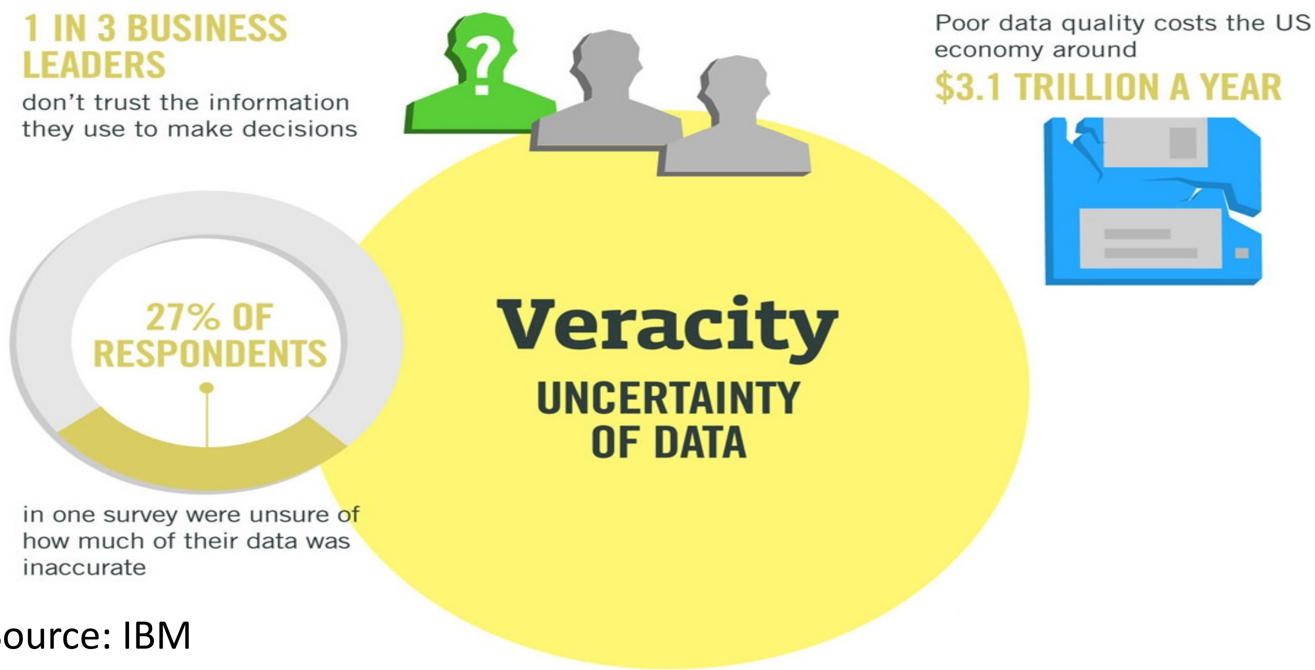
- Applications(Streaming data): Control Systems, Finance Trading, Smart Devices, Social Networks





Veracity: Accuracy (疑)

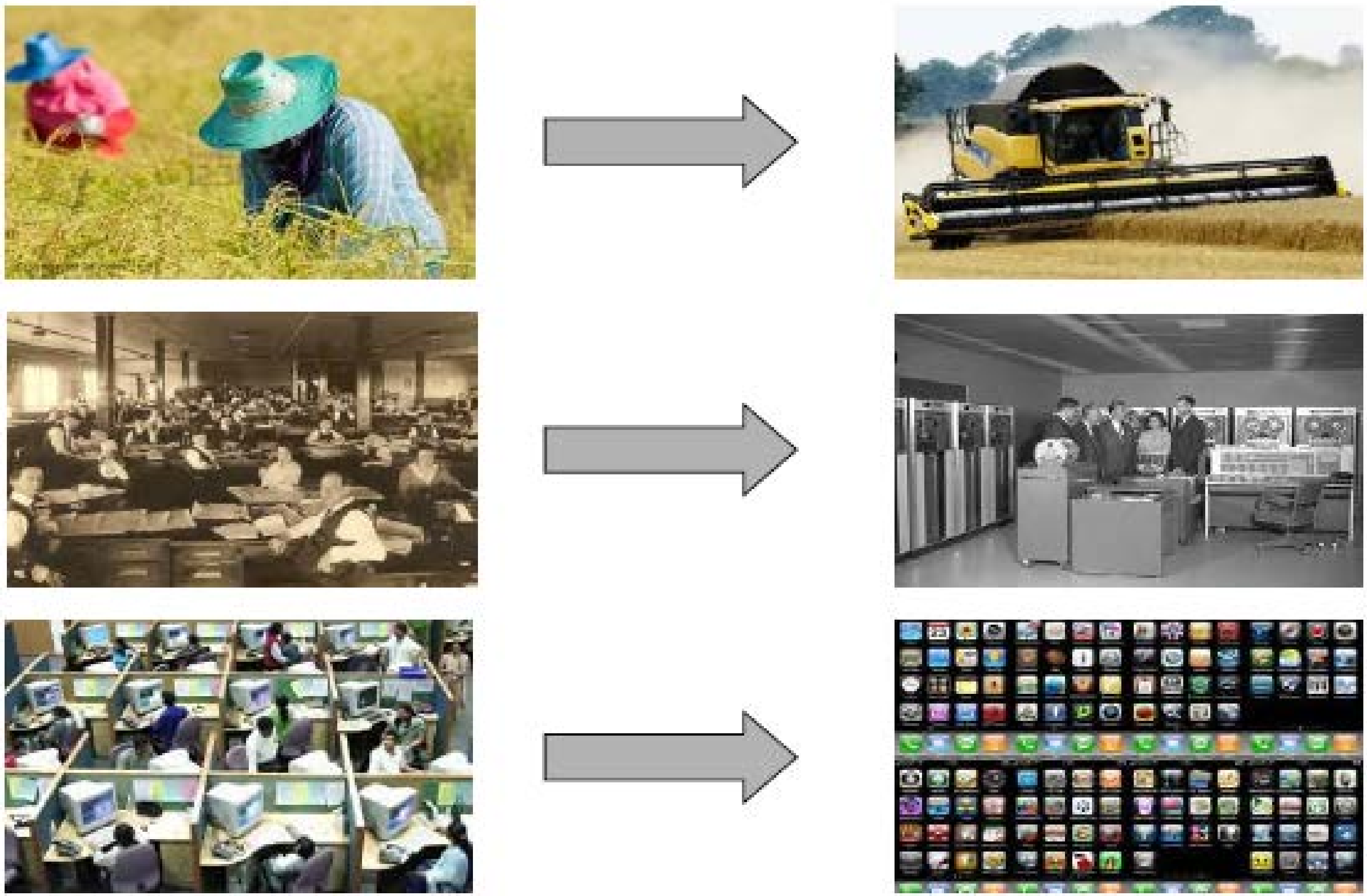
- Refer to the biases, noise and abnormality in data. It requires tools that handle uncertain or imprecise data, and even algorithms that can explore the dark data.



Source: IBM

John Wanamaker once said, “I know that half of my advertising doesn’t work. The problem is I don’t know which half.”

Technology Changes the World...



Outline

- Introduction of BigData
- Hadoop Eco-system & Current Trends
- HDFS & MapReduce
- MapReduce Applications: Information retrieval
- Hive/Pig & Spark

Hadoop Comes to Rescue

- Apache top level project, open-source implementation of frameworks for reliable, scalable, distributed computing and data storage

- Designed to answer the question: **"How to process big data with reasonable cost and time?"**

Google Origins

2003

The Google File System

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung

Google*

2004

MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat

jeff@google.com, sanjay@google.com

Google, Inc.

2006

Bigtable: A Distributed Storage System for Structured Data

Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach
Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber

{fay,jeff,sanjay,wilsonh,kerr,m3b,tushar,fikes,gruber}@google.com

Google, Inc.

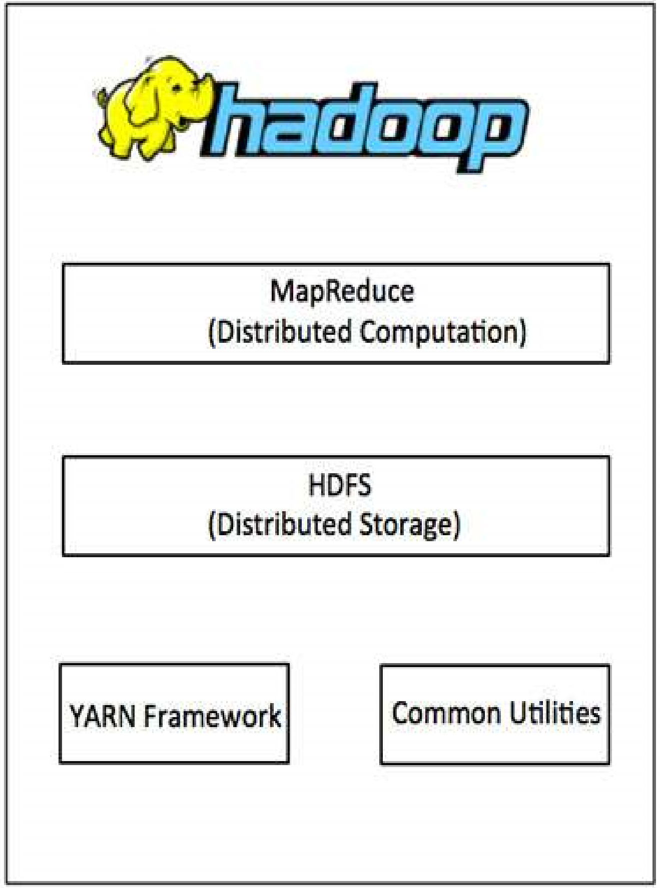
Abstract

igtable is a distributed storage system for managing tured data that is designed to scale to a very large petabytes of data across thousands of commodity ers. Many projects at Google store data in Bigtable, iding web indexing, Google Earth, and Google Fi- e. These applications place very different demands ligtable, both in terms of data size (from URLs to

achieved scalability and high performance, but Big provides a different interface than such systems. Big does not support a full relational data model; inste provides clients with a simple data model that sup dynamic control over data layout and format, an lows clients to reason about the locality properties c data represented in the underlying storage. Data i dexed using row and column names that can be arbi strings. Bigtable also treats data as uninterpreted str

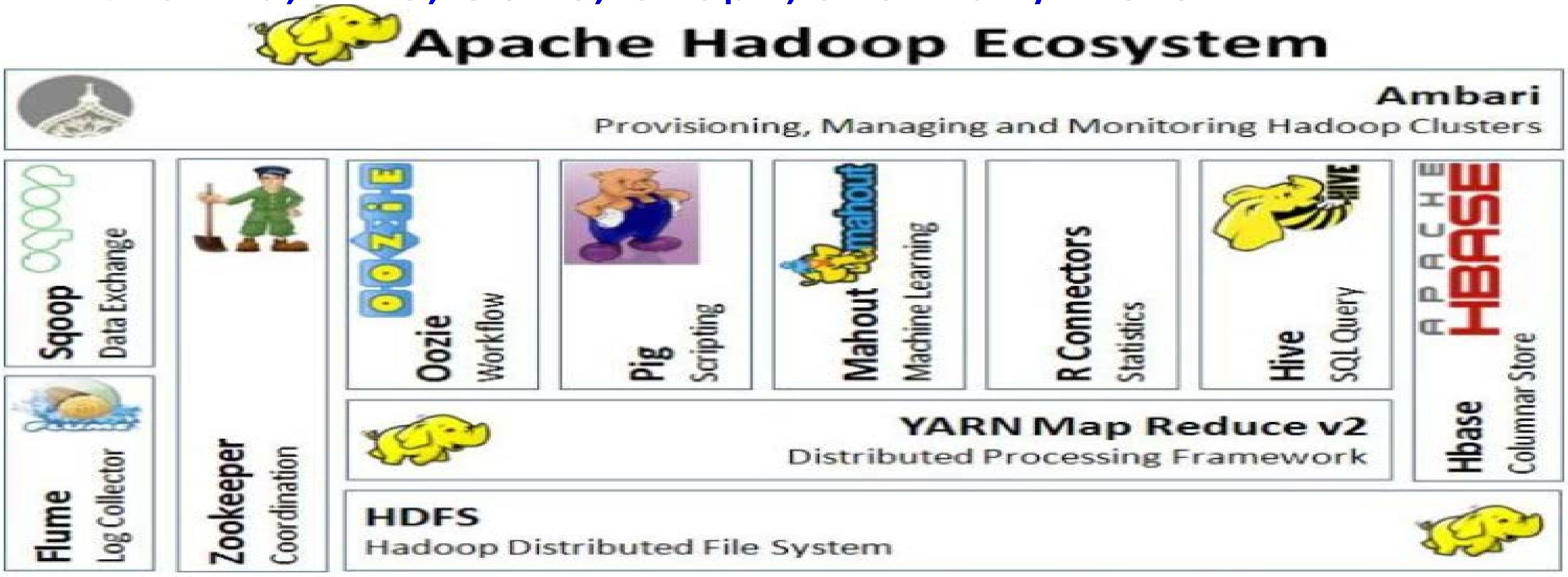
"Core" Hadoop

- Hadoop MapReduce
  - A programming model for large scale distributed data processing
- Hadoop Distributed File System (HDFS)
  - A distributed storage system for big files
- Hadoop YARN (MapReduce 2.0)
  - Resource negotiator for computing tasks
- Common Utilities
  - Contains Libraries and other modules



Hadoop Data Analytic Eco-system

- Ambari, Zookeeper (managing & monitoring)
- HBase, Cassandra (database)
- Hive, Pig (data warehouse and query language)
- Mahout (machine learning)
- Chukwa, Avro, Oozie, Giraph, and many more



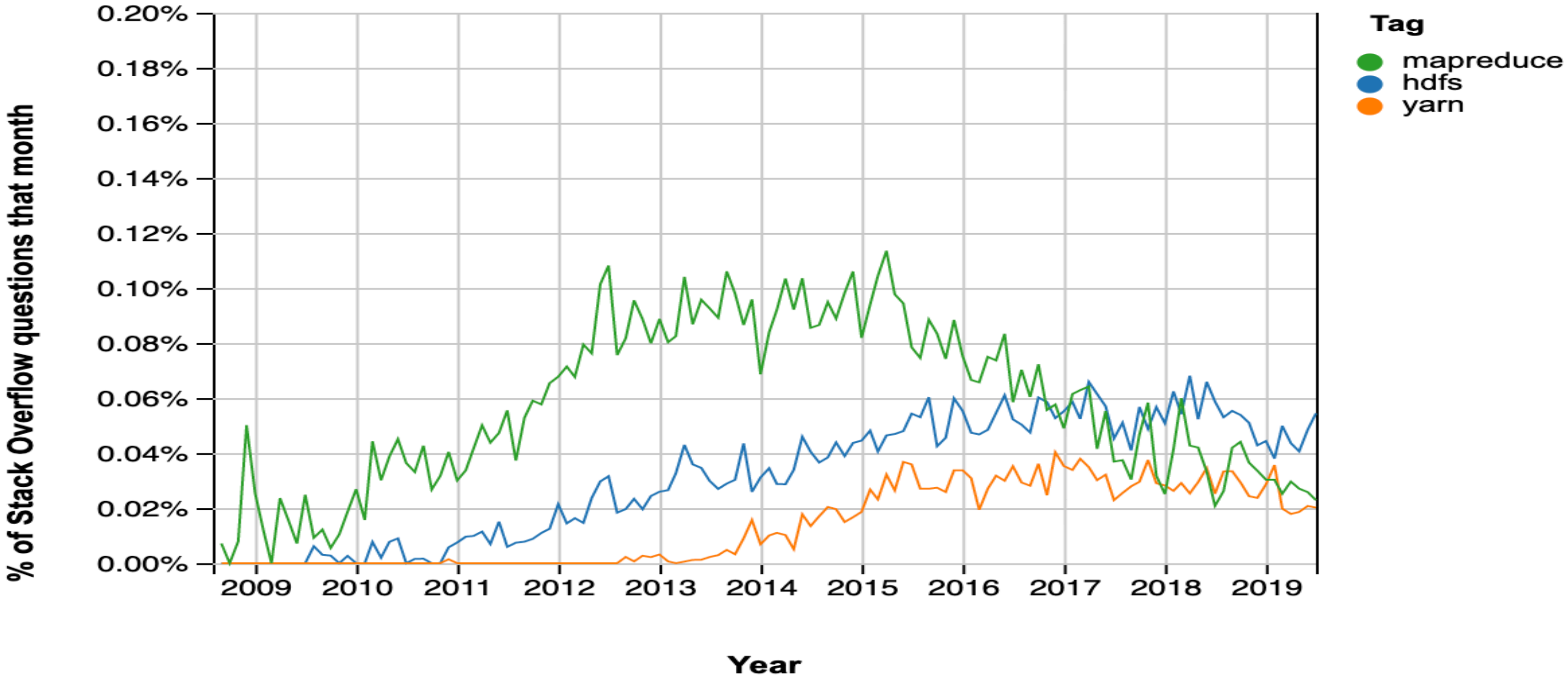
Hadoop Distributions

- A number of vendors have taken advantage of Hadoop open-ended framework and tweaked its codes to change or enhance its functionalities.
- Major companies in the completion:

Cloudera: the first & largest company to develop and distribute Apache Hadoop-based software

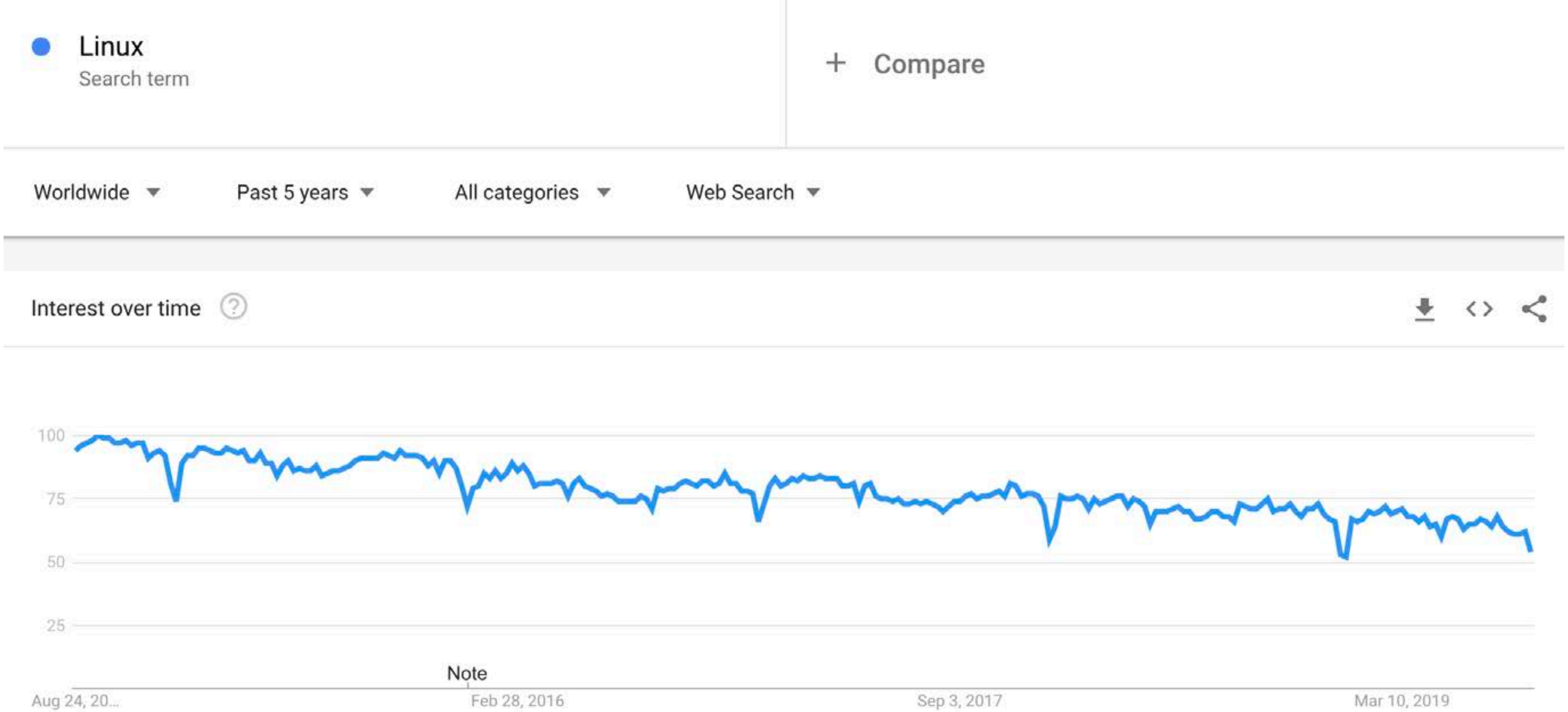
Trend of Big Data Eco-system

- Mapreduce is dead;
  long live hdfs and yarn
  - Stack Overflow Trends
  - HDFS and YARN are mature



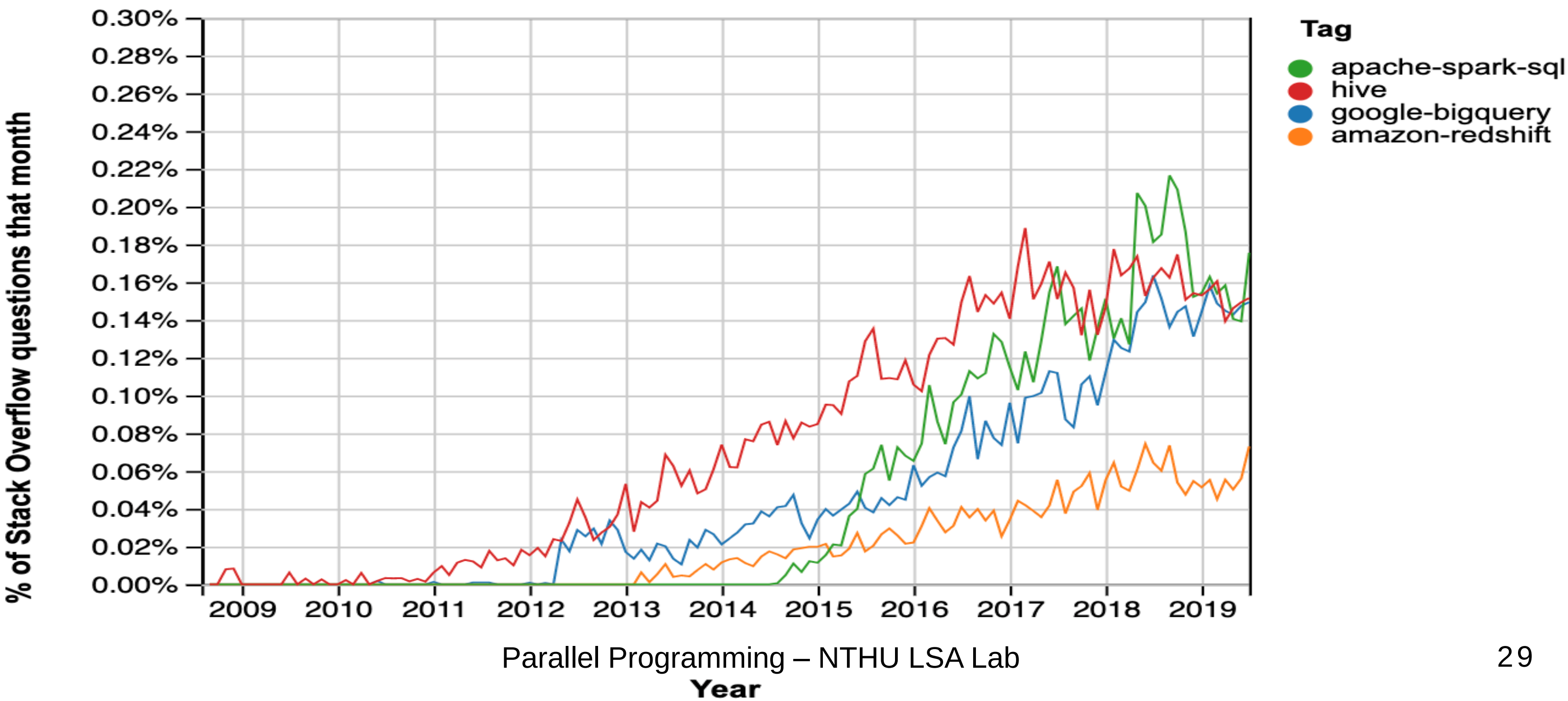
Trend of Big Data Eco-system

- Hadoop is the on-prem platform for Big Data Analytics.
  - Like Linux. Boring, but it's the foundation.



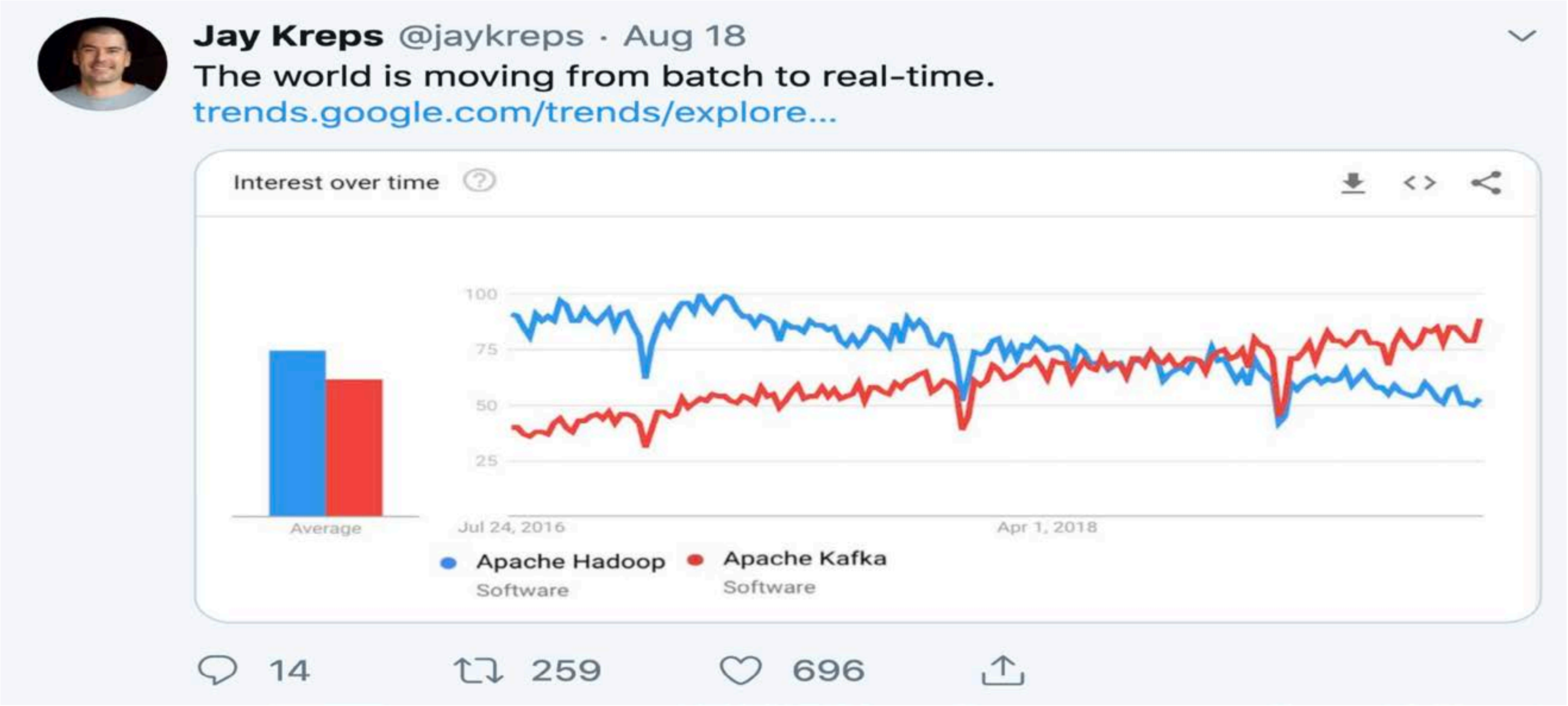
Trend of Big Data Eco-system

- big data: your name is sql
  - Hive was the most popular until 2018.
  - SparkSQL grew fastest until 2018.
  - Cloud: BigQuery is more popular than Redshift



Trend of Big Data Eco-system

- Real-time streaming over Batch
  - Due to emerging technologies like IoT, mobile computing, edge computing, …



Outline

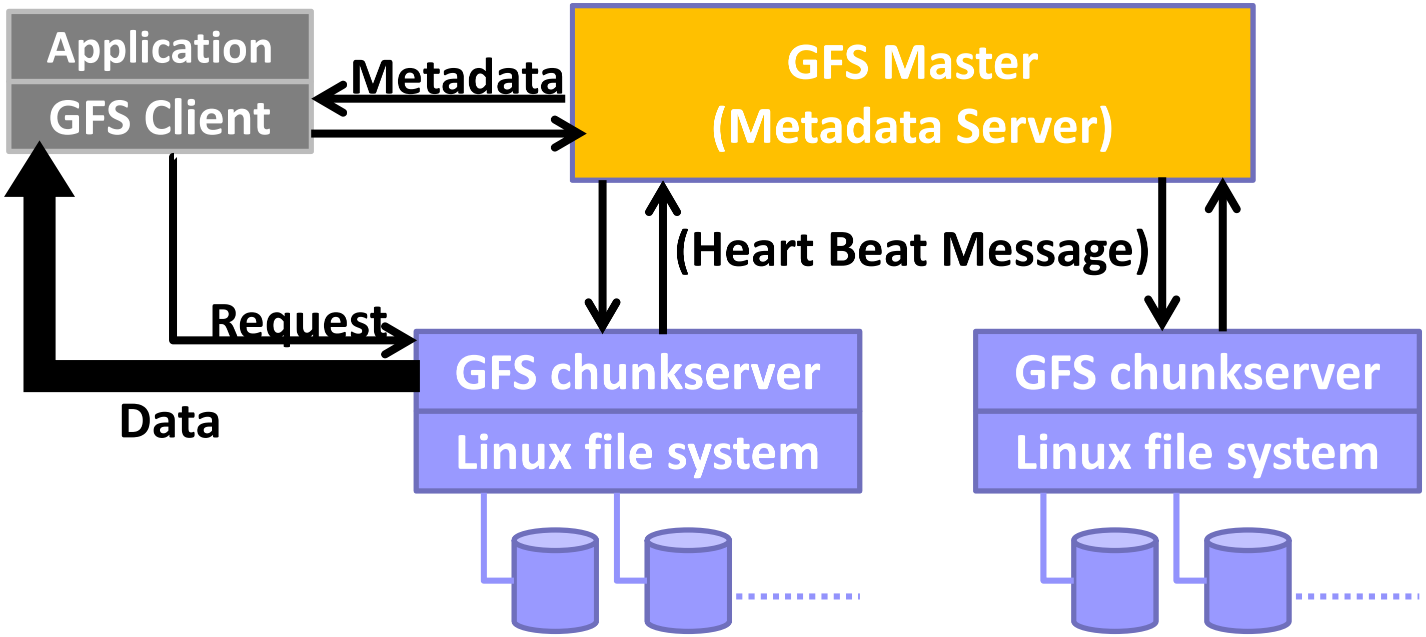
- Introduction of BigData
- Hadoop Eco-system & Current Trends
- HDFS & MapReduce
- MapReduce Applications: Information retrieval
- Hive/Pig & Spark

HDFS: Hadoop Distributed File System

- Very large distributed **file system**
  - 10K nodes, 100 million files, 10PB data
  - **Files are partitioned** and stored across nodes
  - Enabling **parallel I/O**
- Assume commodity hardware
  - **Files are replicated** to handle hardware failure
  - Detect failures and recover from them
- Optimized for batch(large & sequential) processing
  - Primary consist of **large streaming reads** & small random reads
  - Multiple clients concurrent **append data** instead of write
  - Seldom or **No random write**

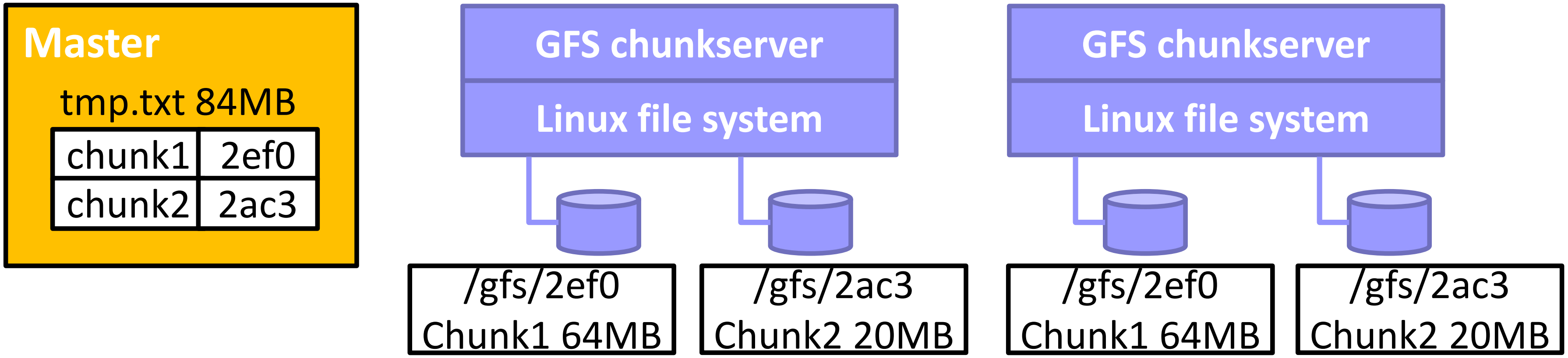
HDFS Architecture

- Components:
  - Master holds metadata (location & access permission of files)
  - Chunkservers hold data
  - Client produces/consumes data



HDFS Data Management

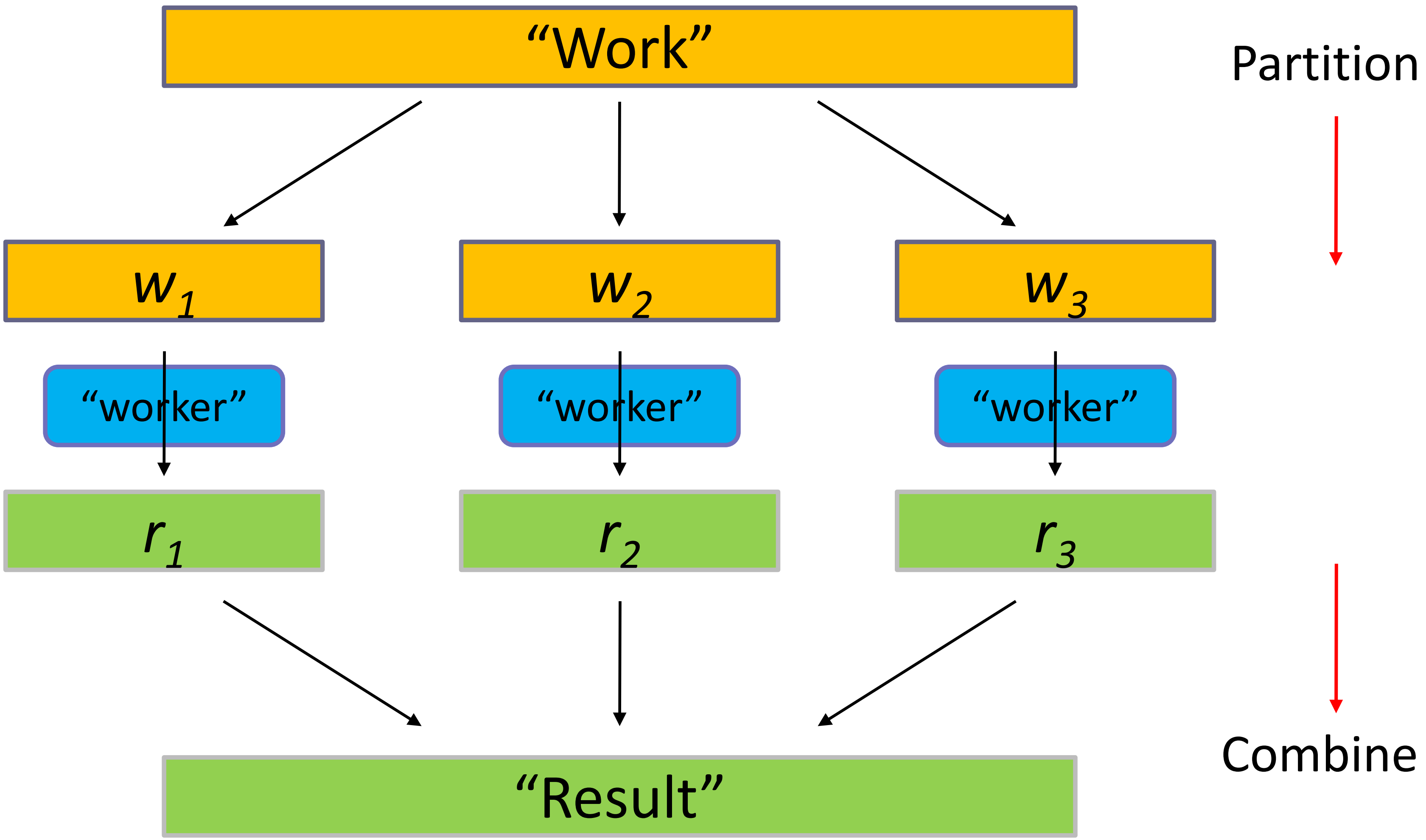
- Files stored in chunks (c.f. "blocks" in disk file systems)
  - A file is partitioned into chunks
  - A chunk is a Linux file on local disk of a chunckserver
  - Unique chunk handles assigned by master at creation time
  - Read/write by (chunk handle, byte range)
  - Fixed large chunk size (i.e. 64MB)
  - Each chunk is replicated across 3 + chunkservers



MapReduce

- Developed by Google to process PB of data per data using datacenters (published in OSDI'04)
  - Program written in this functional style are automatically parallelized and executed on machines
- Hadoop is the open source (JAVA) implemented by Yahoo
- MapReduce has several meanings
  - A programming model
  - A implementation
  - A system architecture

Basic Concept: Divide and Conquer



Typical Large-Data Problem

- Iterate over a large number of records
- Extract something of interest from each record
- Shuffle and sort intermediate results
- Aggregate intermediate results
- Generate final output

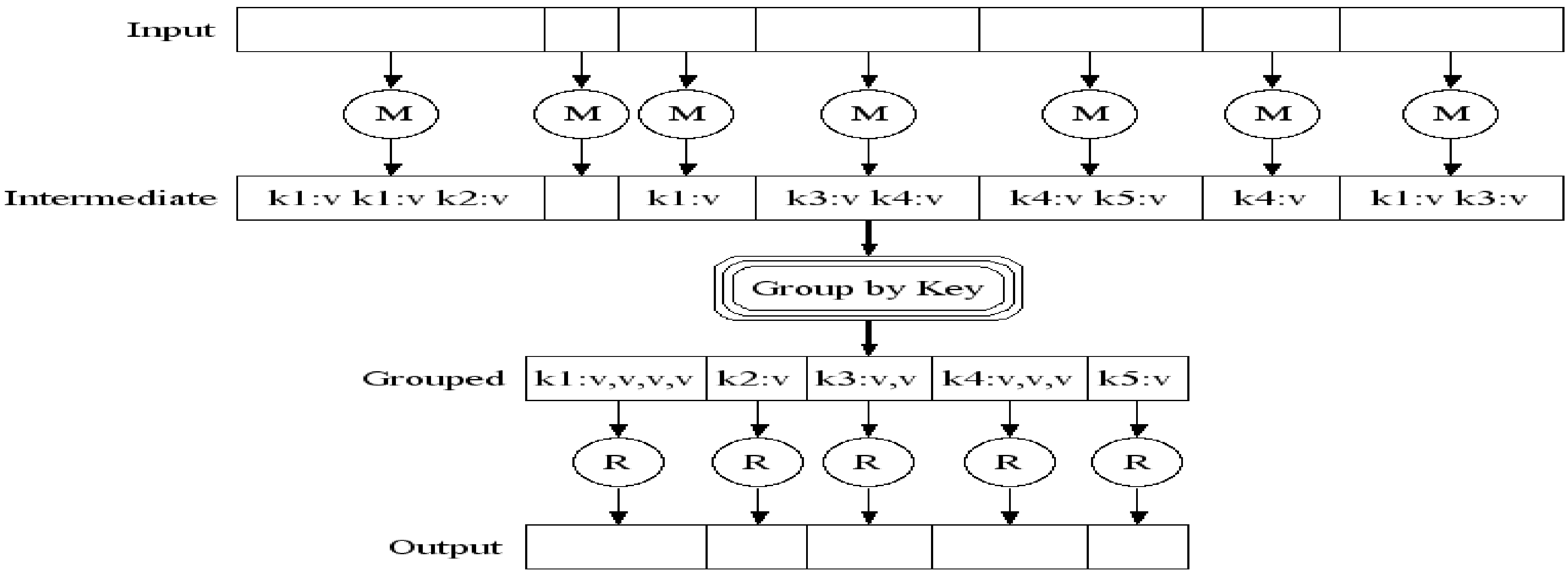
Map

Reduce

Key idea: provide **a functional abstraction** for these two operations

MapReduce Programming Model

- A parallel programming model (divide-conquer)
  - Map: processes a key/value pair to generate a set of **intermediate key/value pairs**
  - Reduce: merges all **intermediate values** associated with

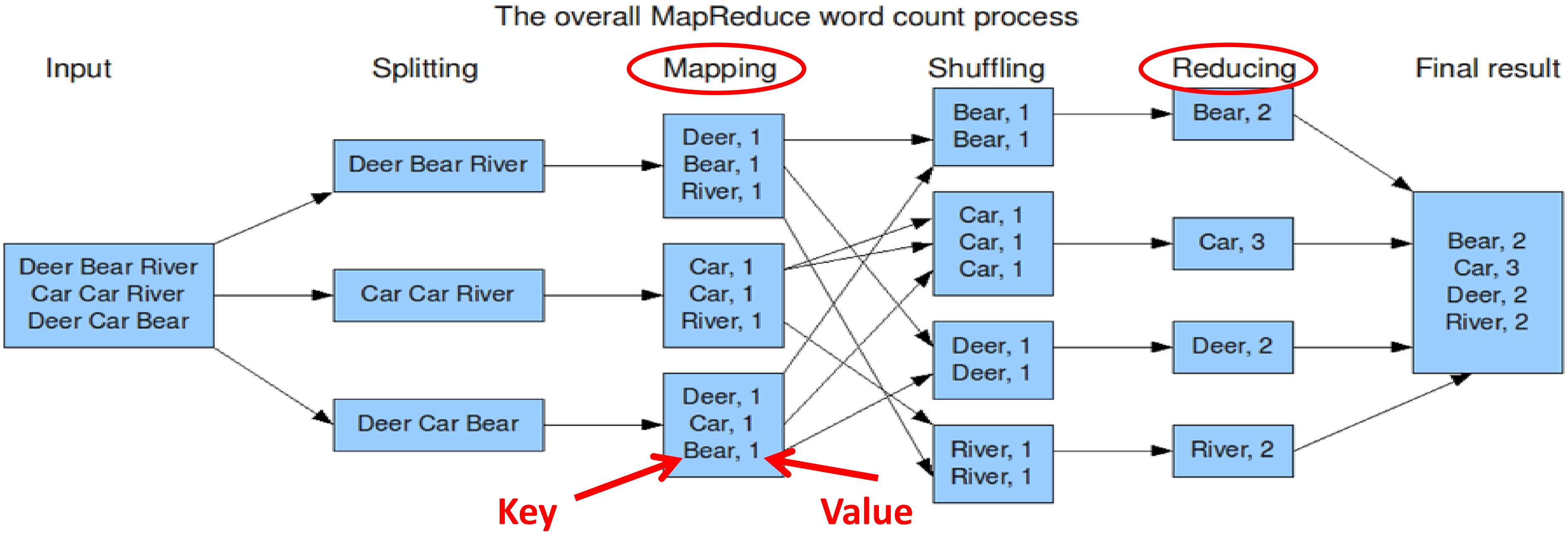


MapReduce Word Count Example

- User specify the map and reduce functions

```
Map(String docid, String text):
    for each word w in text:
        Emit(w, 1);
```

```
Reduce(String term, Iterator<Int> values):
    int sum = 0;
    for each v in values:
        sum += v;
        Emit(term, value);
```



- The execution framework handles everything else…

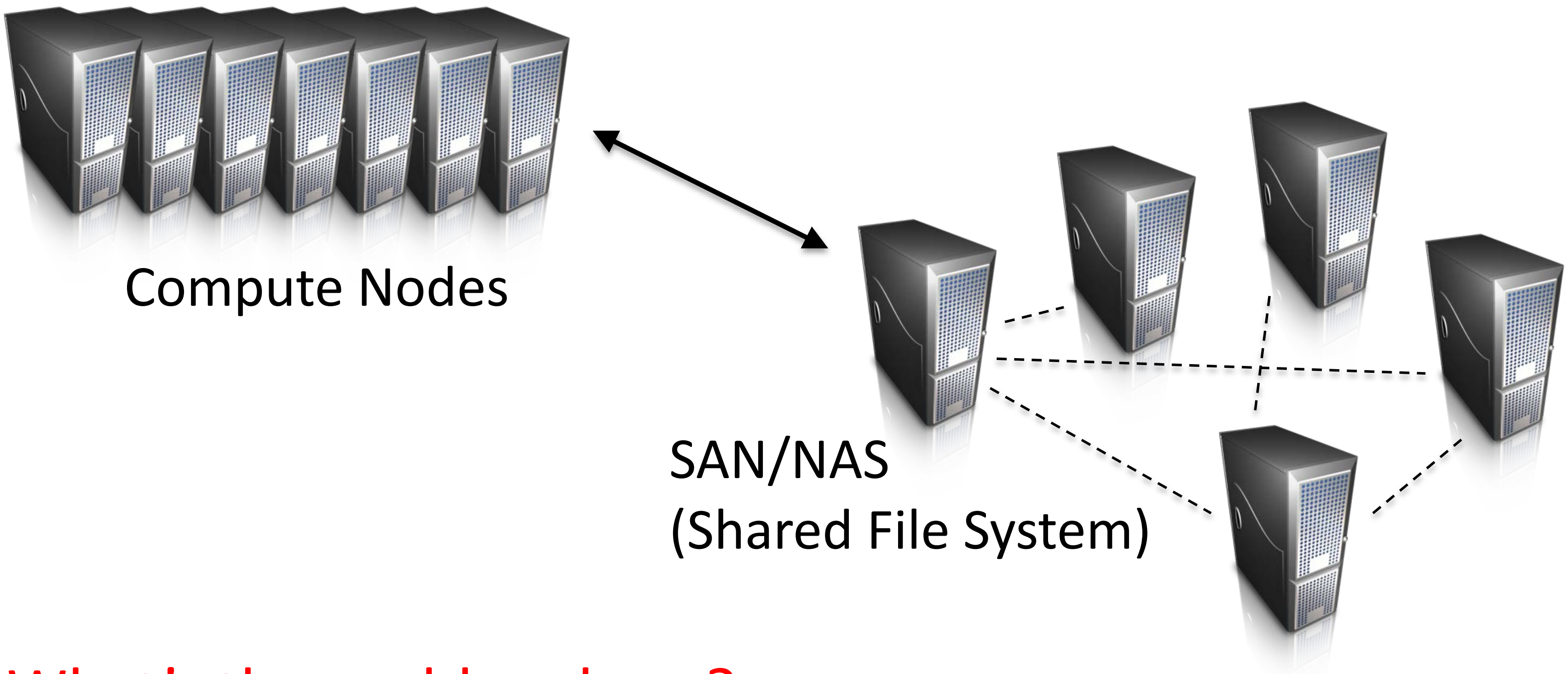
What's "everything else"?

MapReduce “Runtime”

- Handles scheduling
  - Assigns workers to map and reduce tasks
- Handles “data distribution”
  - Moves processes to data
- Handles synchronization
  - Gathers, sorts, and shuffles intermediate data
- Handles errors and faults
  - Detects worker failures and restarts
- Everything happens on top of a distributed FS

Problem of Data Access Bottleneck

- Traditional Solution:



What's the problem here?

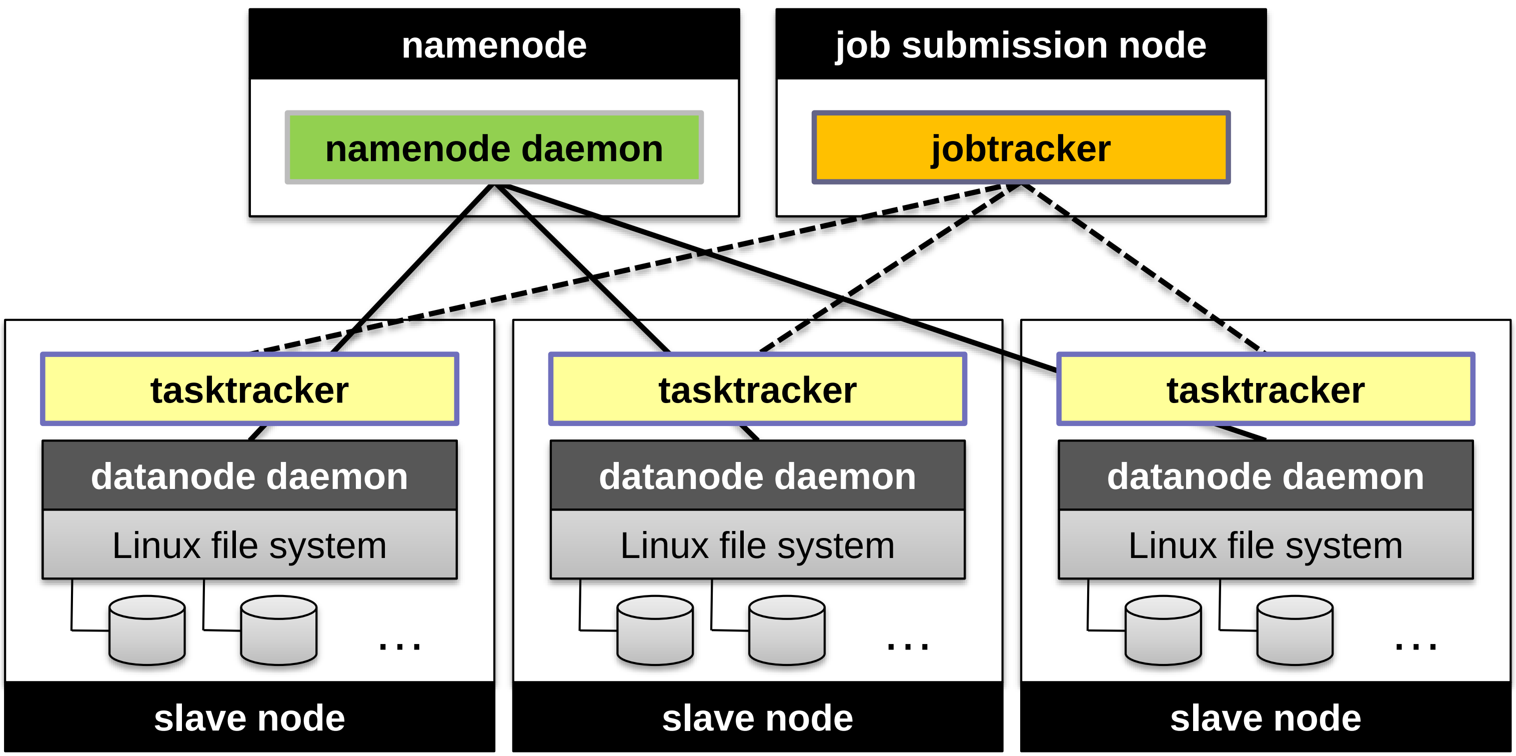
Get Workers Closer to Data

- Don’t move data to workers…
  move workers to the data!
  - A node act as both compute and storage node
  - Store data on the local disks of nodes in the cluster
  - Start up the workers on the node that has the data local

- A distributed file system is the answer
  - GFS (Google File System) for Google’s MapReduce
  - HDFS (Hadoop Distributed File System) for Hadoop

Putting everything together…

- Hadoop:
  - Namenode (Master in GFS): file metadata server
  - Job/Task tracker: MapReduce engine



MapReduce in Action

User program

File (3*64MB)

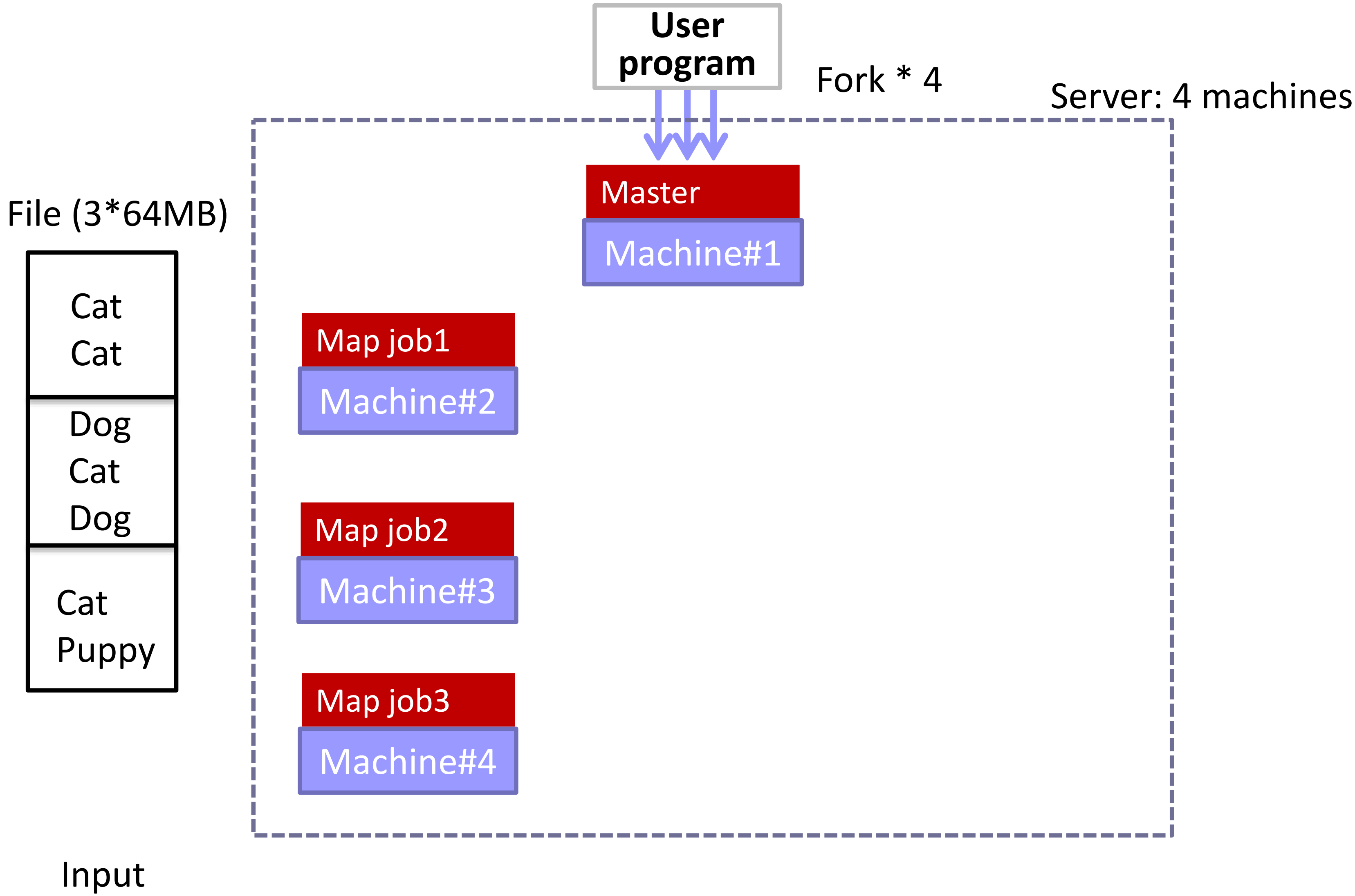
Cat
Cat

Dog
Cat
Dog

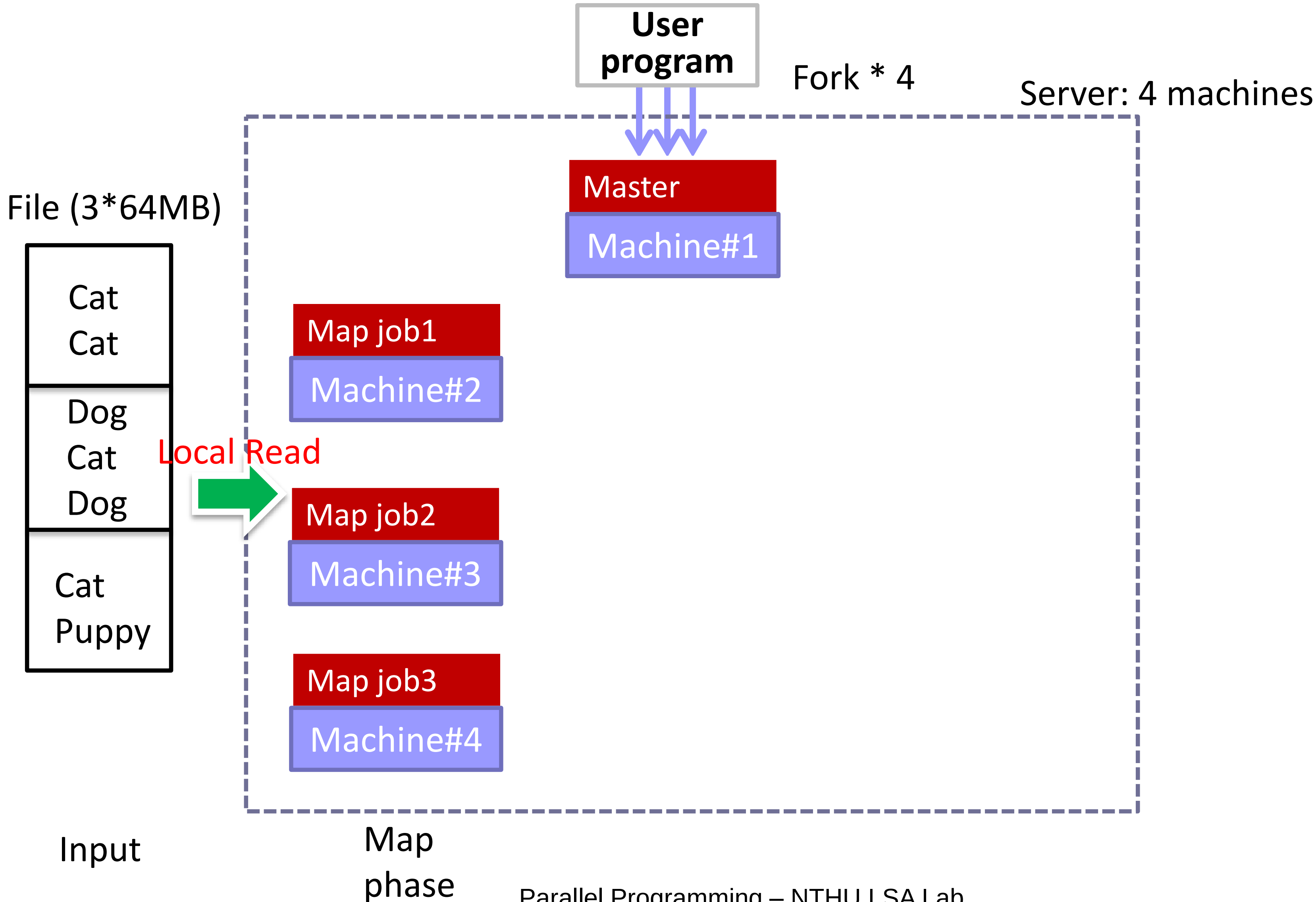
Cat
Puppy

Input

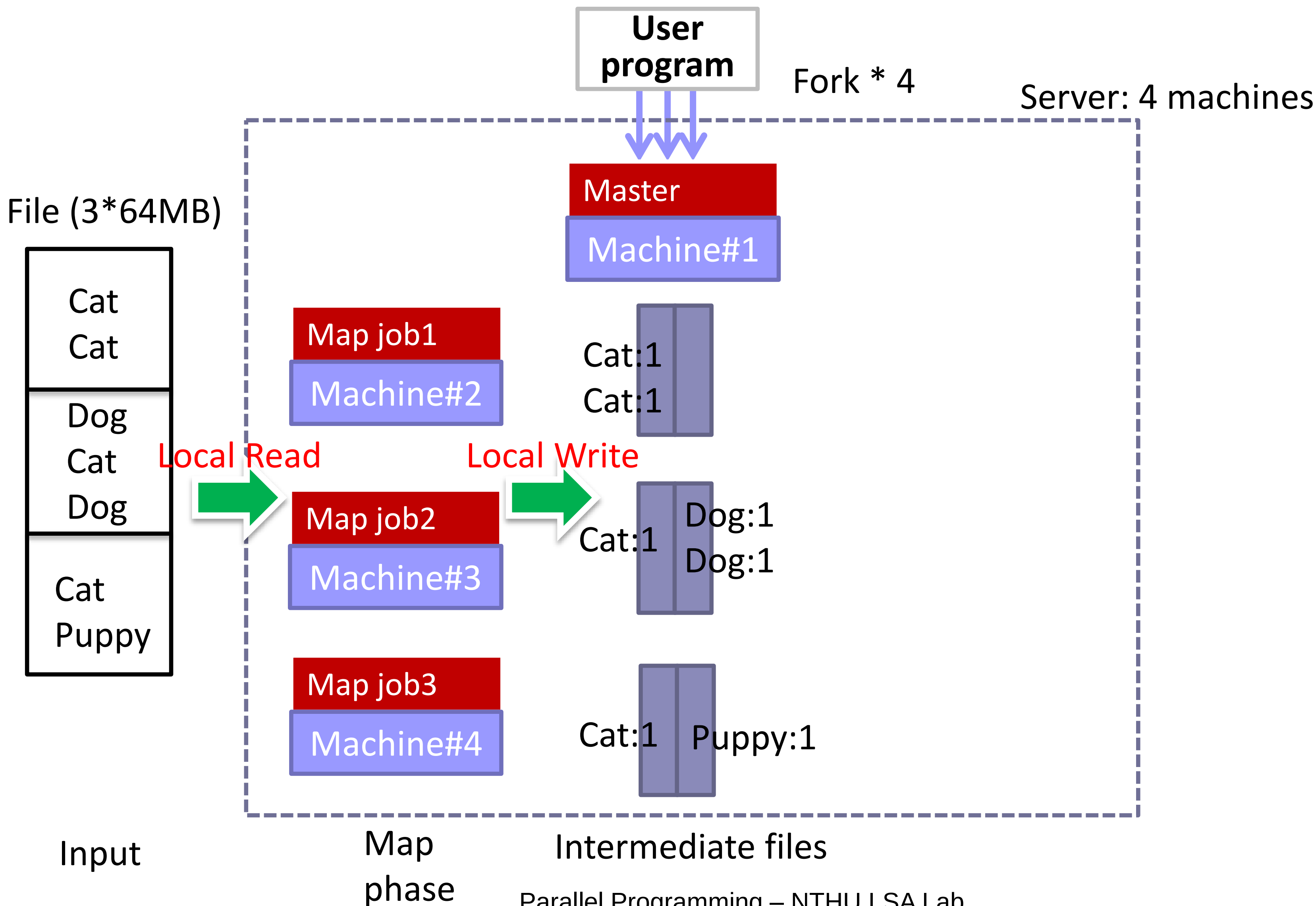
MapReduce in Action



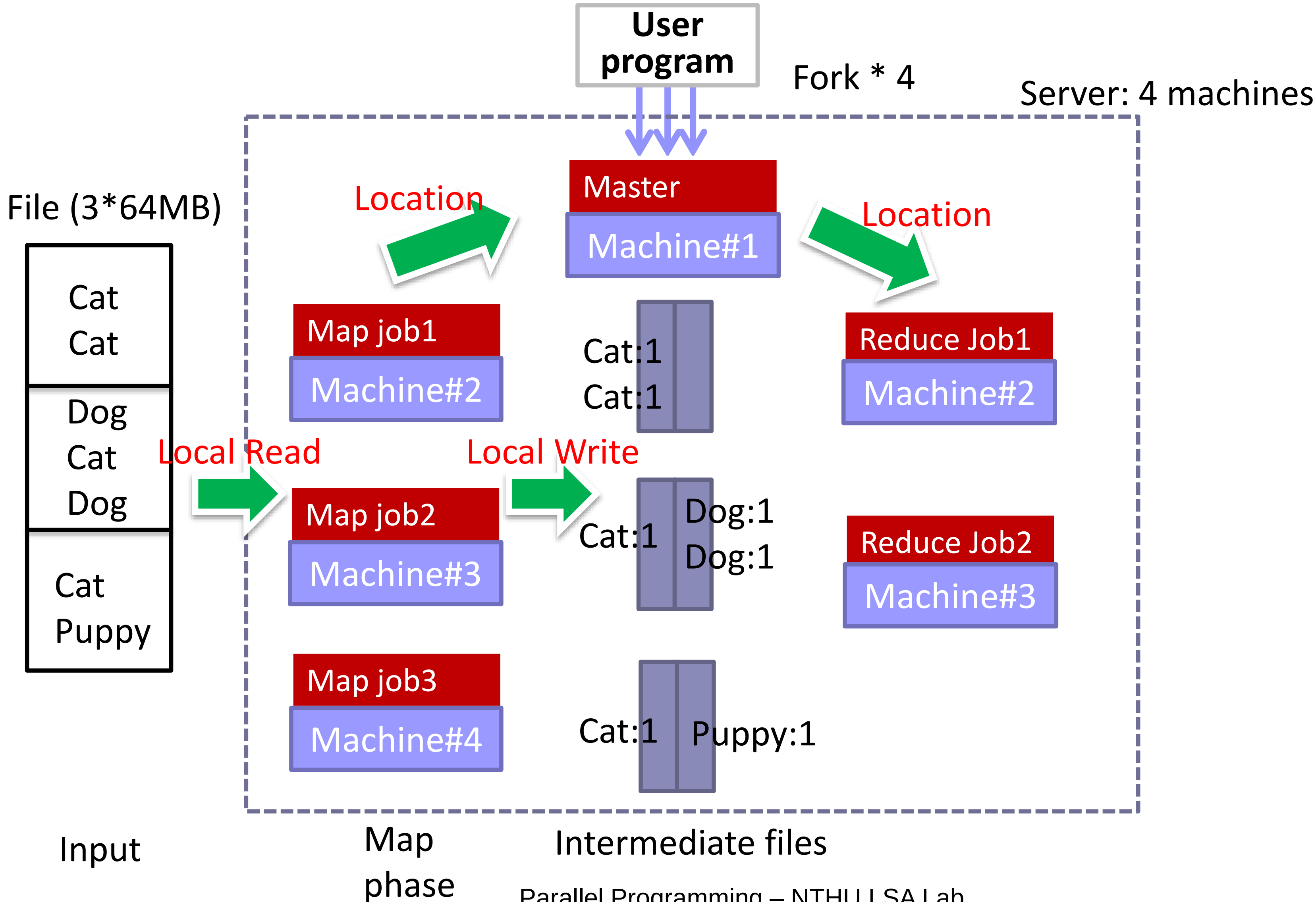
MapReduce in Action



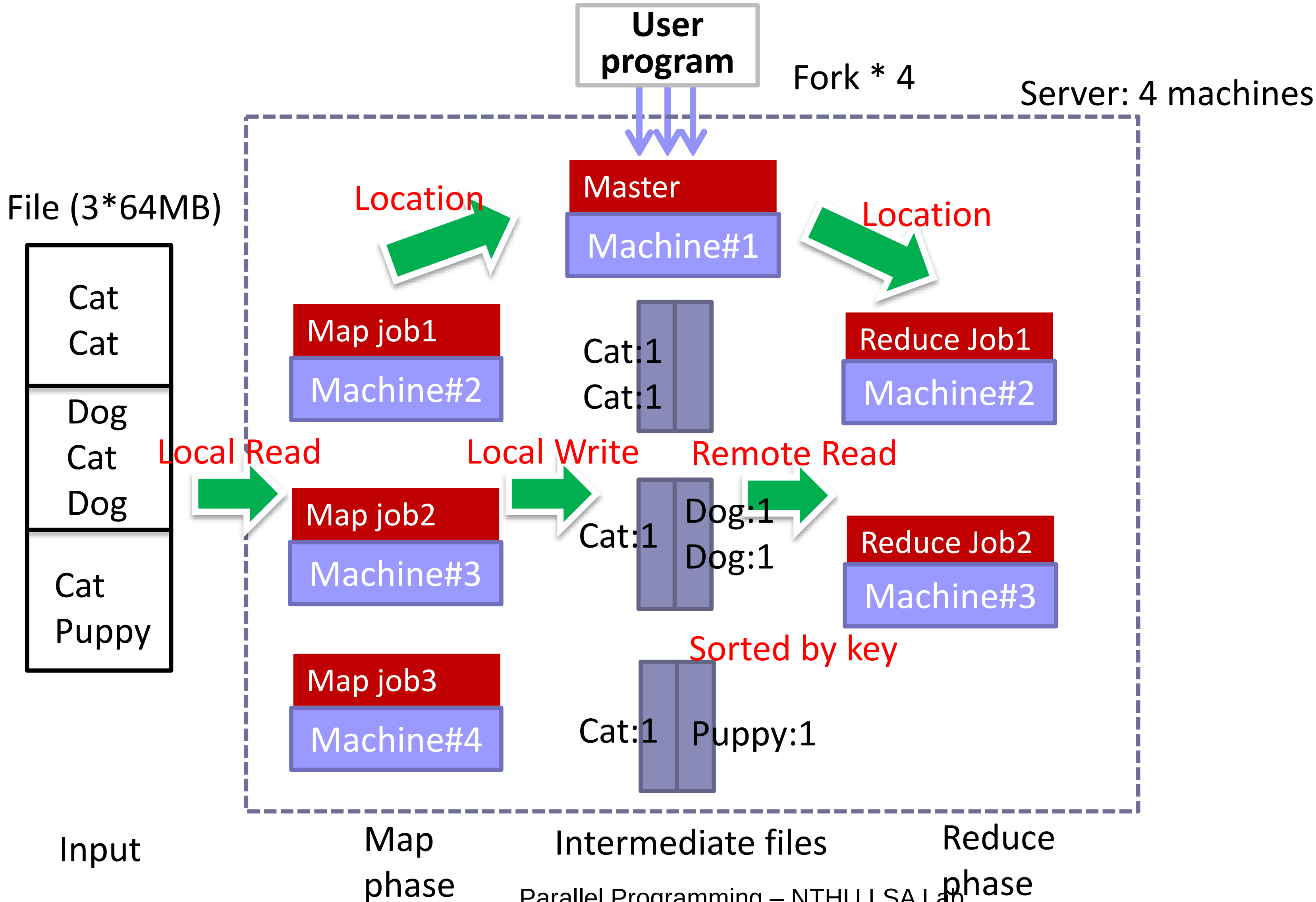
MapReduce in Action



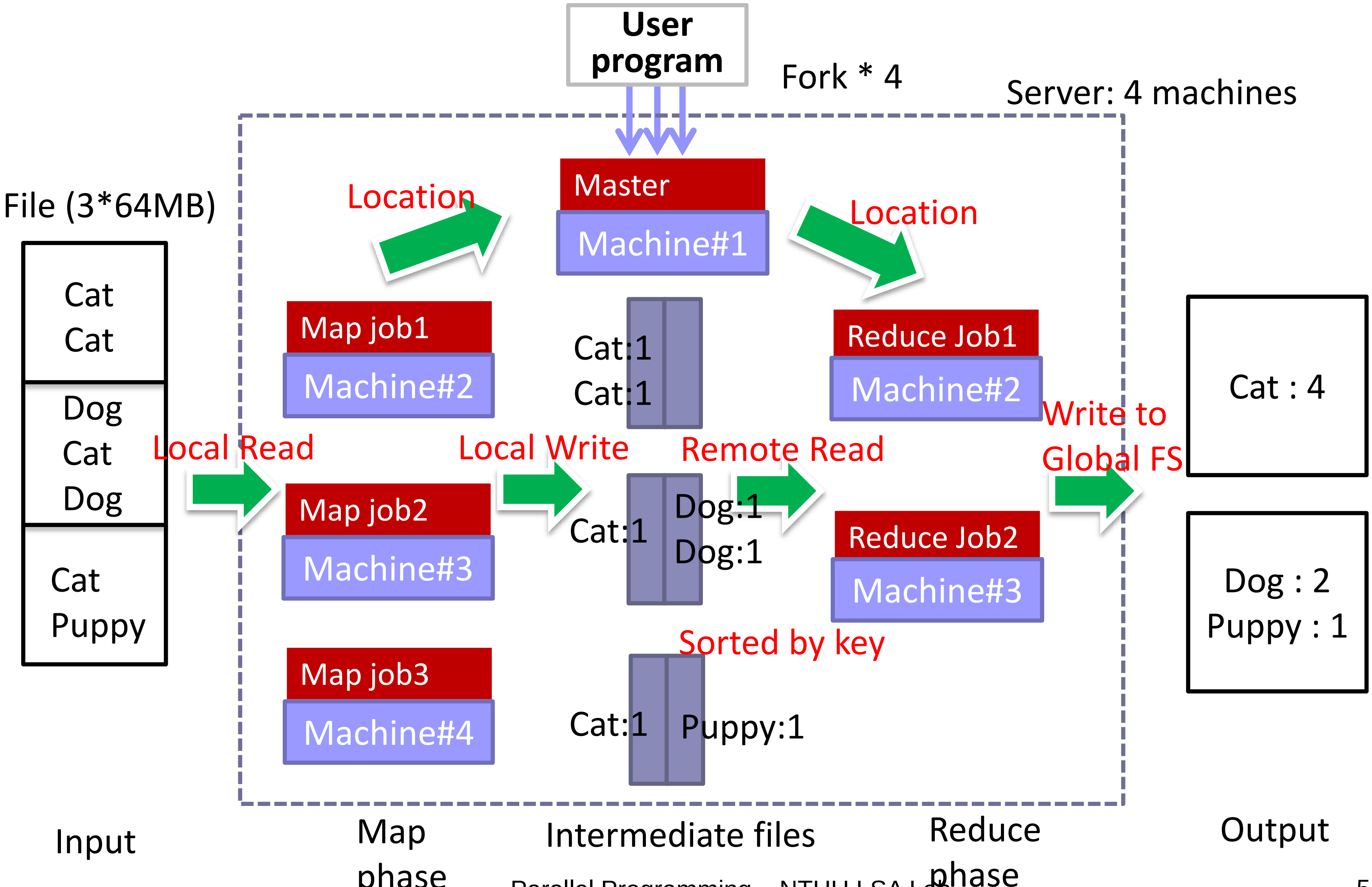
MapReduce in Action



MapReduce in Action



MapReduce in Action



Outline

- Introduction of BigData
- Hadoop Eco-system & Current Trends
- HDFS & MapReduce
- MapReduce Applications: Information retrieval
- Hive/Pig & Spark

Information retrieval (IR)

- Information retrieval (IR)
  - the activity of obtaining information resources relevant to an information need from a collection of information resources
  - Focus on textual information (= text/document retrieval)
  - Other possibilities include image, video, music, …
- What do we find?
  - Generically, “documents”
  - Even though we may be referring to web pages, PDFs, PowerPoint slides, paragraphs, etc.

The Central Problem in Search

Author

Concepts

Query Terms

"tragic love story"

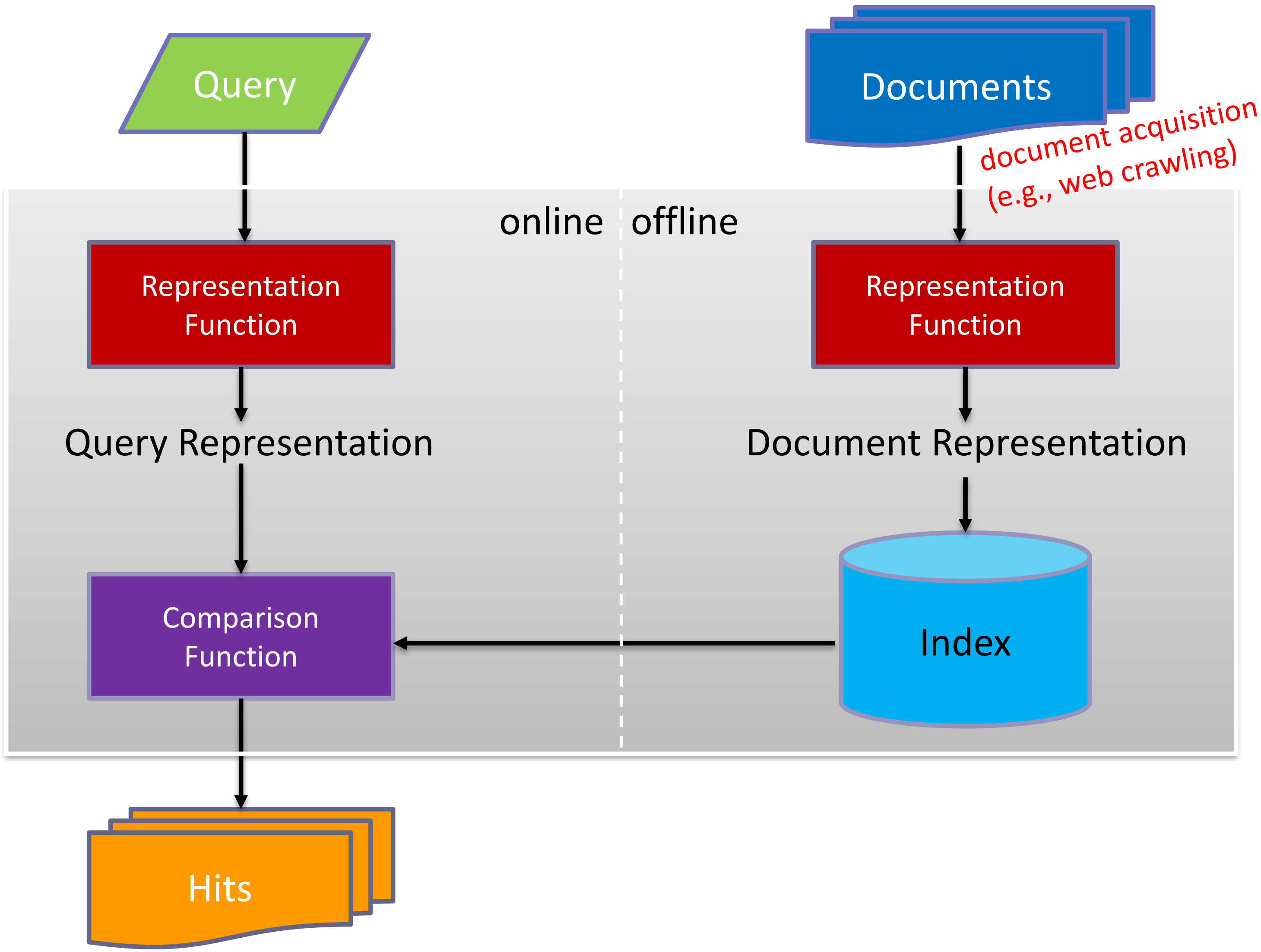
Concepts

Document Terms

"fateful star-crossed romance"

Do these represent the same concepts?

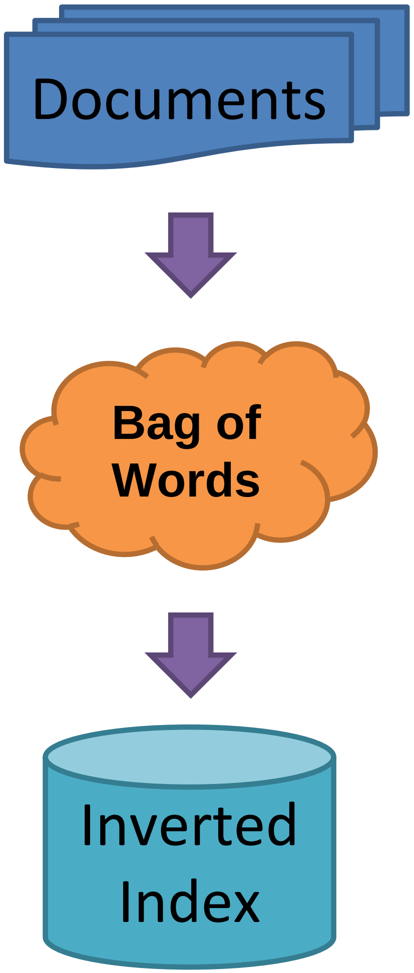
Abstract IR Architecture



How do we represent text?

- Remember: computers don’t “understand” anything!
- “Bag of words”
  - Treat all the words in a document as **index** terms
  - Assign a “**weight**” to each term based on “**importance**” (or, in simplest case, presence/absence of word)
  - Disregard order, structure, meaning, etc. of the words
  - Simple, yet effective!
- Assumptions
  - Term occurrence is independent
  - Document relevance is independent
  - “Words” are well-defined

Counting Words…



1. Tokenization: Aren't → are not
2. Stopword removal: a, an, is, it, etc.
3. Normalization (equivalence classing of terms):
   - Hello → hello, windows → window, 你好 → hello

~~syntax~~, ~~semantics~~, ~~word knowledge~~, etc.

Boolean Retrieval

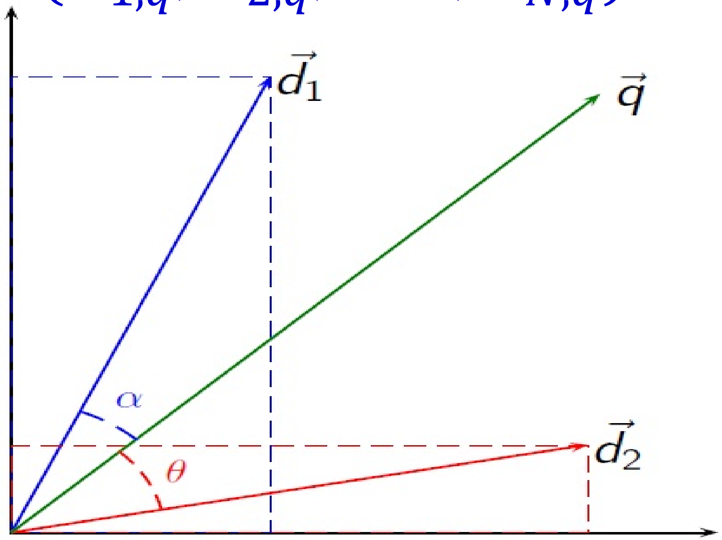
- Express queries as a **Boolean expression**
  - AND, OR, NOT
  - Can be arbitrarily nested
- Retrieval is based on the notion of sets
  - Any query divides the collection into TWO **sets**: retrieved, not-retrieved
  - Pure Boolean systems do NOT define an **ordering** of the results

Ranked Retrieval

- Order documents by how likely they are to be relevant to the information need
  - Sort documents by relevance
  - Display sorted results
- User model
  - Present hits one screen at a time, best results first
  - At any point, users can decide to stop looking
- How do we estimate relevance?
  - Assume document is relevant if it has a lot of query **terms**
  - Represent query and document in **vector**
  - Relevance means the **closeness of vectors**

Vector Space Model

- Documents and queries are represented as vectors: Each term is a dimension
  - Document: $d_j = \left(w_{1,j}, w_{2,j}, \dots \dots, w_{N,j}\right)$
  - Query: $q = (w_{1,q}, w_{2,q}, \dots \dots, w_{N,q})$



retrieve documents based on how close the document is to the query (i.e., similarity ~ “closeness”)

Similarity Metric

- Use "angle" between the vectors:

$$sim(d_j, q) = \cos(\theta) = \frac{\overrightarrow{d_j} \cdot \vec{q}}{\left|\overrightarrow{d_j}\right| \cdot |\vec{q}|}$$

$$sim(d_j, q) = \frac{\sum_{i=0}^{n} w_{i,j}\, w_{i,q}}{\sqrt{\sum_{i=0}^{n} \left(w_{i,j}\right)^2}\sqrt{\sum_{i=0}^{n} \left(w_{i,q}\right)^2}}$$

- Or, more generally, inner products:

$$sim(d_j, q) = \overrightarrow{d_j} \cdot \overrightarrow{q_k} = \sum_{i=0}^{n} w_{i,j}\, w_{i,q}$$

Term Weighting

- Term weights consist of two components
  - Local: how important is the term in this document?
  - Global: how important is the term in the collection?
- Here's the intuition:
  - Terms that appear often in a document should get high weights
  - Terms that appear in many documents should get low weights
- How do we capture this mathematically?
  - **Term frequency** (local)
  - Inverse **document frequency** (global)

TF.IDF Term Weighting

- Term Frequency-Inverse Document Frequency mode:
  - Terms appear often in a document should get high weights
  - Terms appear in many documents should get low weights

$$w_{i,j} = \mathrm{tf}_{i,j} \cdot \log \frac{N}{df_i}$$

$w_{i,j}$ weight assigned to term ***i*** in document ***j***

$\mathrm{tf}_{i,j}$ frequency of occurrence of term ***i*** in document ***j***

N number of documents in entire collection

df_i number of documents with term ***i***

Inverted Index: TF.IDF

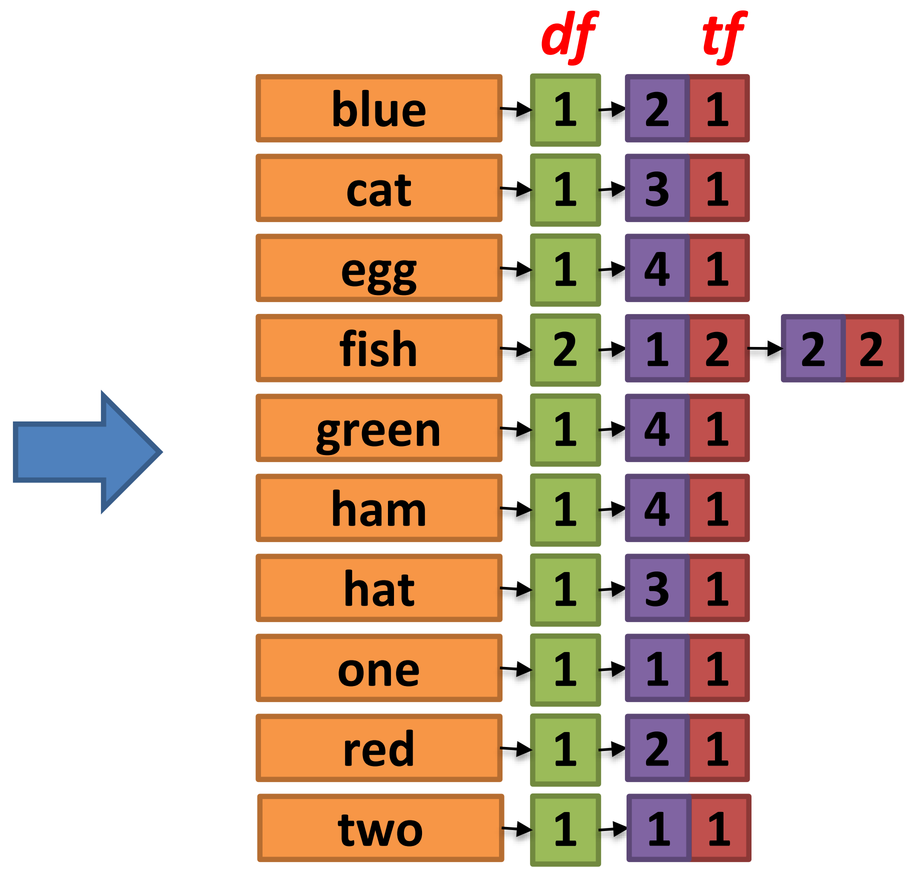
Doc 1
one fish, two fish

Doc 2
red fish, blue fish

Doc 3
cat in the hat

Doc 4
green eggs and ham

	tf				
	1	**2**	**3**	**4**	***df***
blue		1			1
cat			1		1
egg				1	1
fish	2	2			2
green				1	1
ham				1	1
hat			1		1
one	1				1
red		1			1
two	1				1



MapReduce: Index Construction

- Map over all documents
  - Emit *term* as key, (*docno, tf)* as value
  - Emit other information as necessary (e.g., term position)
- Sort/shuffle:
  - group postings by term
- Reduce
  - Gather and sort the postings (e.g., by *docno* or *tf*)
  - Write postings to disk
- MapReduce does all the heavy lifting!

Inverted Indexing with MapReduce

Map

Doc 1: one fish, two fish			Doc 2: red fish, blue fish			Doc 3: cat in the hat		
one	1	1	red	2	1	cat	3	1
two	1	1	blue	2	1	hat	3	1
fish	1	2	fish	2	2			

Shuffle and Sort: aggregate values by keys

Reduce

cat	3	1		
fish	1	2	2	2
one	1	1		
red	2	1		

blue	2	1
hat	3	1
two	1	1

Inverted Indexing: Pseudo-Code

class MAPPER
    procedure MAP(docid n, doc d)
        $H \leftarrow$ new ASSOCIATIVEARRAY
        for all term $t \in$ doc d **do**
            $H\{t\} \leftarrow H\{t\} + 1$
        for all term $t \in H$ **do**
            EMIT(term t, posting $\langle n, H\{t\} \rangle$)

H{t}: term frequency in a file

class REDUCER
    procedure REDUCE(term t, postings $[\langle a_1, f_1 \rangle, \langle a_2, f_2 \rangle \ldots]$)
        $P \leftarrow$ new LIST
        for all posting $\langle a, f \rangle \in$ postings $[\langle a_1, f_1 \rangle, \langle a_2, f_2 \rangle \ldots]$ **do**
            APPEND(P, $\langle a, f \rangle$)
        SORT(P)
        EMIT(term t, postings P)

Problem?

Another Try…

- Use <***term***><***docid***> as the key
  - Let the framework do the sorting
  - Term frequency implicitly stored
  - **Directly write postings to disk!**

(key)	(values)	
fish	1	[2,4]
	34	[23]
	21	[1,8,22]
	35	[8,41]
	80	[2,9,76]
	9	[9]

(keys)		(values)
fish	1	[2,4]
fish	9	[9]
fish	21	[1,8,22]
fish	34	[23]
fish	35	[8,41]
fish	80	[2,9,76]

Retrieval with MapReduce?

- MapReduce is fundamentally **batch-oriented**
  - Optimized for throughput, not latency
  - Startup of mappers and reducers is expensive
- MapReduce is not suitable for **real-time** queries!
  - Initial a job takes time
  - Use separate infrastructure (like database) for retrieval…

Outline

- Introduction of BigData
- Hadoop Eco-system & Current Trends
- HDFS & MapReduce
- MapReduce Applications: Information retrieval
- Hive/Pig & Spark

Hive and Pig

- Hive: data warehousing application in Hadoop
  - Query language is HQL, variant of **SQL**
  - Tables stored on HDFS as flat files
  - Developed by Facebook, now open source
- Pig: large-scale data processing system
  - Scripts are written in Pig Latin, a dataflow language
  - Developed by Yahoo!, now open source
  - Roughly 1/3 of all Yahoo! internal jobs
  - Similar to the role of SCALE for Spark

- Common idea:
  - **Provide higher-level language to facilitate large-data processing**
  - Higher-level language “compiles down” to Hadoop jobs

Hive: Behind the Scenes

```
SELECT s.word, s.freq, k.freq FROM shakespeare s
  JOIN bible k ON (s.word = k.word) WHERE s.freq >= 1 AND k.freq >= 1
  ORDER BY s.freq DESC LIMIT 10;
```

(Abstract Syntax Tree)

```
(TOK_QUERY (TOK_FROM (TOK_JOIN (TOK_TABREF shakespeare s) (TOK_TABREF bible k) (= (. (TOK_TABLE_OR_COL s)
word) (. (TOK_TABLE_OR_COL k) word)))) (TOK_INSERT (TOK_DESTINATION (TOK_DIR TOK_TMP_FILE)) (TOK_SELECT
(TOK_SELEXPR (. (TOK_TABLE_OR_COL s) word)) (TOK_SELEXPR (. (TOK_TABLE_OR_COL s) freq)) (TOK_SELEXPR (.
(TOK_TABLE_OR_COL k) freq))) (TOK_WHERE (AND (>= (. (TOK_TABLE_OR_COL s) freq) 1) (>= (. (TOK_TABLE_OR_COL k)
freq) 1))) (TOK_ORDERBY (TOK_TABSORTCOLNAMEDESC (. (TOK_TABLE_OR_COL s) freq))) (TOK_LIMIT 10)))
```

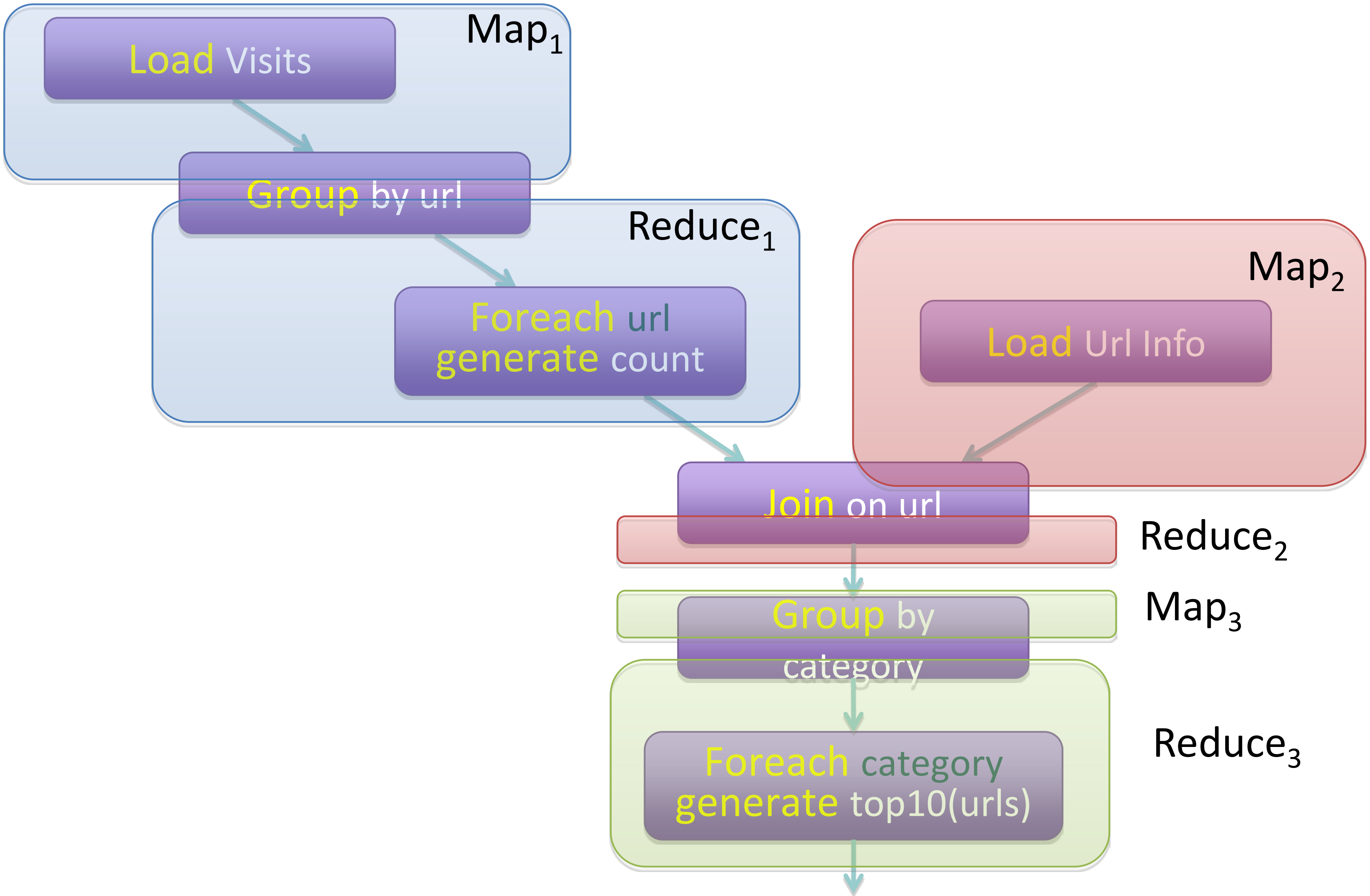
(one or more of MapReduce jobs)

Pig Script

```
visits = load '/data/visits' as (user, url, time);

gVisits = group visits by url;

visitCounts = foreach gVisits generate url, count(visits);

urlInfo = load '/data/urlInfo' as (url, category, pRank);

visitCounts = join visitCounts by url, urlInfo by url;

gCategories = group visitCounts by category;

topUrls = foreach gCategories generate
    top(visitCounts,10);

store topUrls into '/data/topUrls';
```


Pig Slides adapted from Olston et al. (SIGMOD 2008)

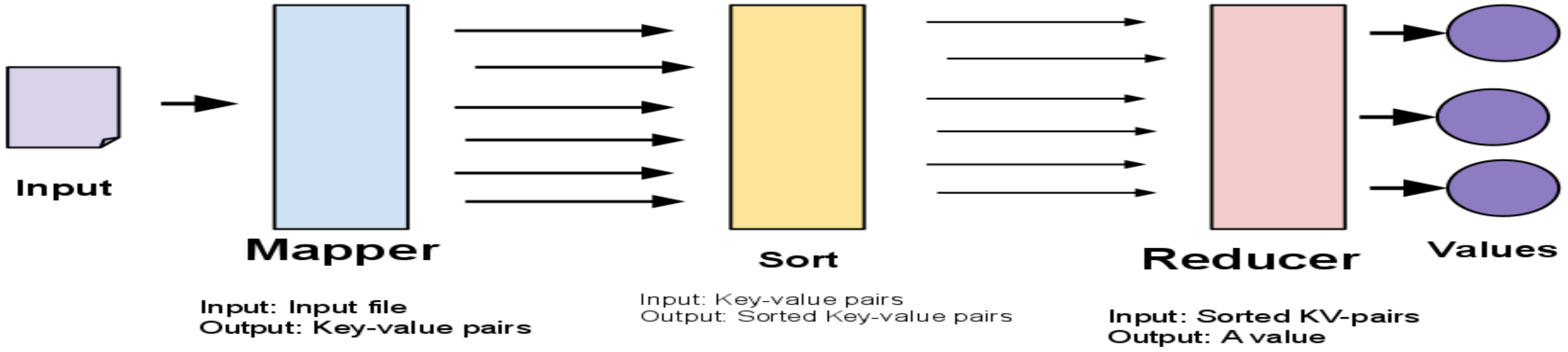
Pig Script in Hadoop



Pig Slides adapted from Olston et al. (SIGMOD 2008)

Limitation of MapReduce

- Simple but limited programming model



- Can only apply two computation functions in a job: Map&Reduce

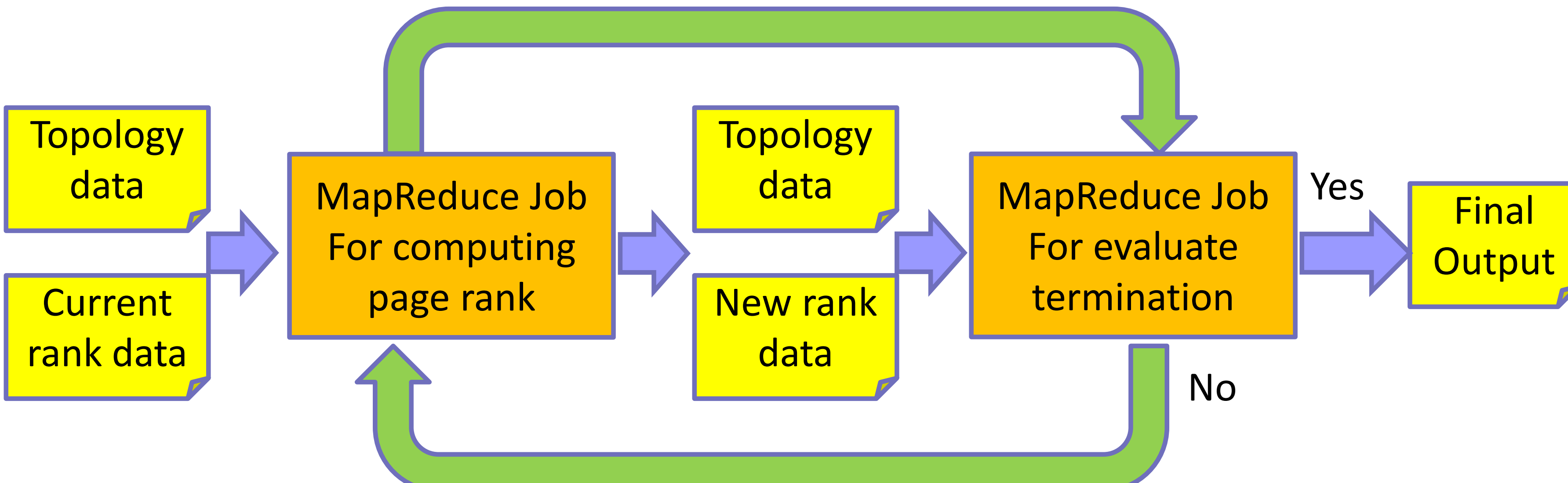
→More complex work must use **multiple** jobs

- The input and output of a job must store into a FS

→FS(disk) is the only device to provide **data persistency**

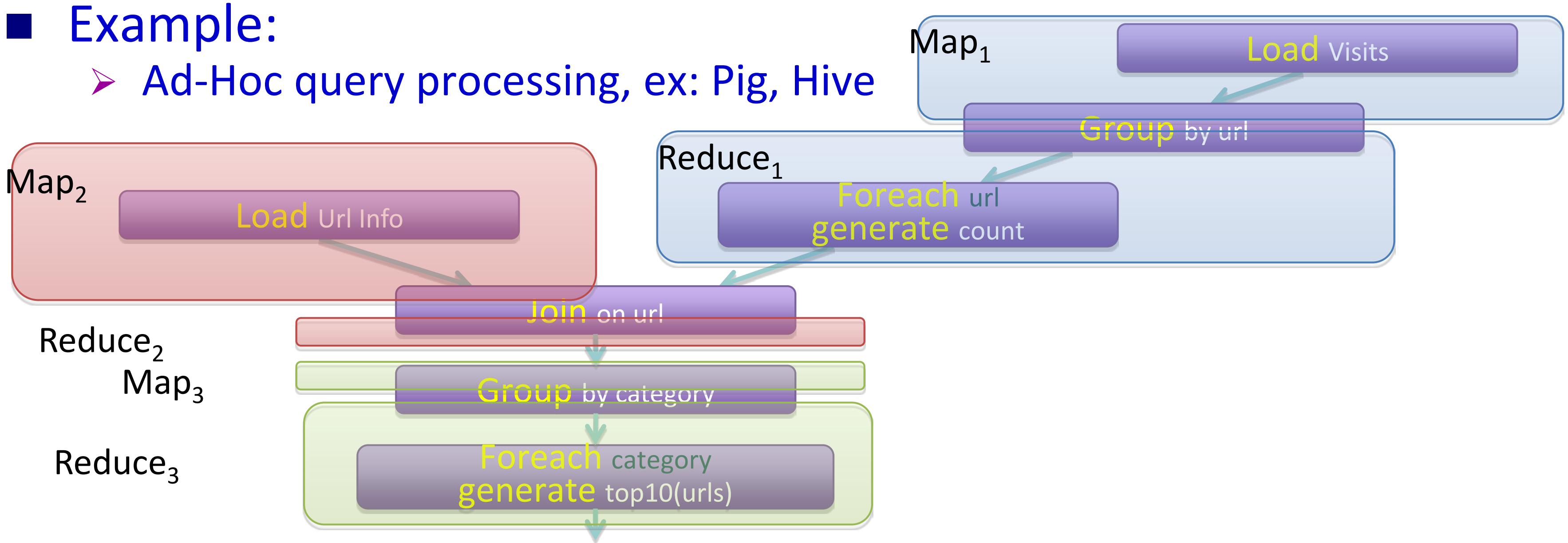
Limitation of MapReduce: Iterative

- How MapReduce handles iterative processing?
  - Each iteration is submitted as an independent job
  - Termination criteria is evaluated after each iteration
  - Data is written and read out disk after each iteration
  - The invariant data is repeatedly store/load/transfer



Limitation of MapReduce: Interactive

- What is interactive processing?
  - computation involving the exchange of information between a user and the computer
- Property:
  - Require short response time ➔ disk is too slow
  - Repeatedly process on the same set of data ➔ redundant I/O
  - Complex data-flow ➔ can be specified by one job
- Example:
  - Ad-Hoc query processing, ex: Pig, Hive



Spark Comes to Rescue

- Support efficient **iterative and interactive data processing**
  - Suitable for machine learning, data analytics, stream processing
- Adapt generalized functional programming language - **SCALA**
  - A scalable language combining **object-oriented** and **functional prog**.
- Utilize **DSM** (Distributed Shared Memory) in data processing to enable in-memory computing
  - Allow users to explicitly **cache dataset in memory across machines** and reuse it in multiple MapReduce-like parallel operations
- Retain the **scalability and fault tolerance** property like MapReduce
  - Overcome volatile memory challenge

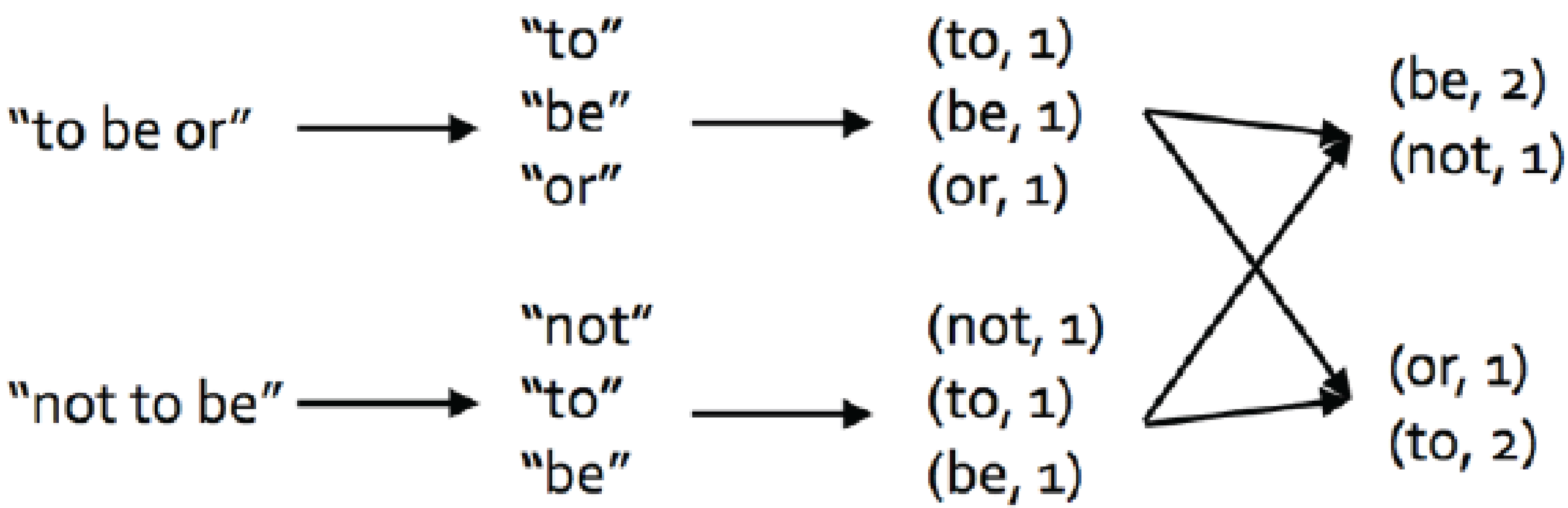
Functional methods on collections

- There are a lot of methods on Scala collections
  http://www.scala-lang.org/api/2.10.4/index.html#scala.collection.Seq

Method on Seq[T]	Explanation
map(f: T=>U): Seq[U]	Each element is result of f
flatMap(f: T=>Seq[U]): Seq[U]	One to many mapping
filter(f: T=>Boolean): Seq[T]	Keep elements passing f
exists(f: T=>Boolean): Boolean	True if one element passes f
forall(f: T=>Boolean): Boolean	True if all elements pass f
reduce(f: (T,T) => T): T	Merge elements using f
groupBy(f: T=>K): Map[K,List[T]]	Group elements by f
sortBy(f: T=>K): Seq[T]	Sort elements

Word Count Example

- val lines = sc.textFile(“hamlet.txt”)!
- val counts = lines.flatMap(line => line.split(“ “)).
  map(word => (word, 1)).
  reduceByKey(_ + _)



Spark: RDDs

Resilient distributed datasets (RDDs)

- **Immutable, partitioned collections** of objects
- **Created through parallel *transformations*** (map, filter, groupBy, join, …) on data in stable storage
- Can be ***cached*** for efficient reuse
- RDDs are lazy and ephemeral. That is, partitions of a dataset are **materialized (i.e. computed)** on demand when they are used in a parallel operation (**i.e. actions**)

Actions on RDDs

- Count, reduce, collect, save, …

Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

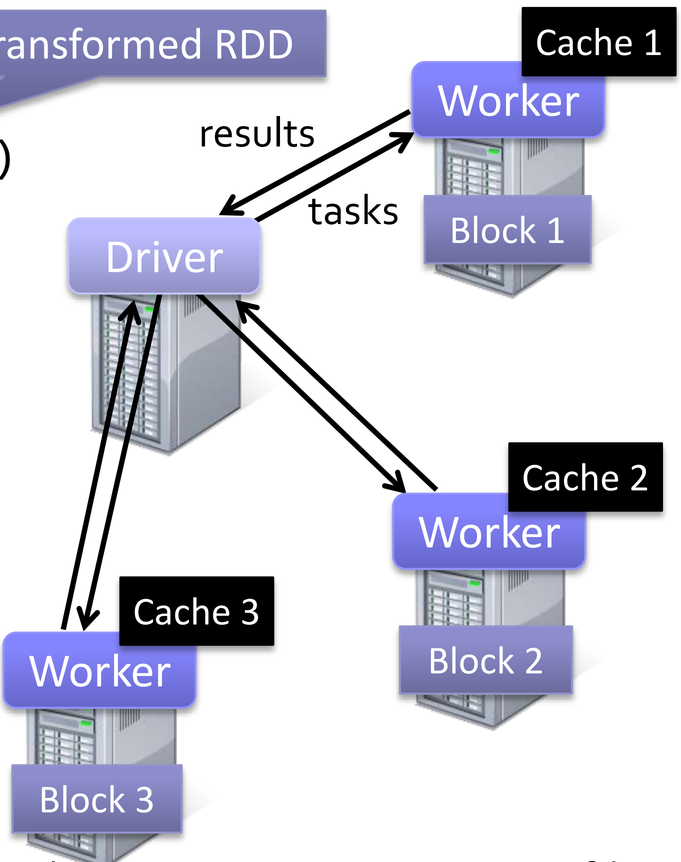
Base RDD

Transformed RDD

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('\t')(2))
cachedMsgs = messages.cache()
```

Action

```
cachedMsgs.filter(_.contains("foo")).count
cachedMsgs.filter(_.contains("bar")).count
. . .
```



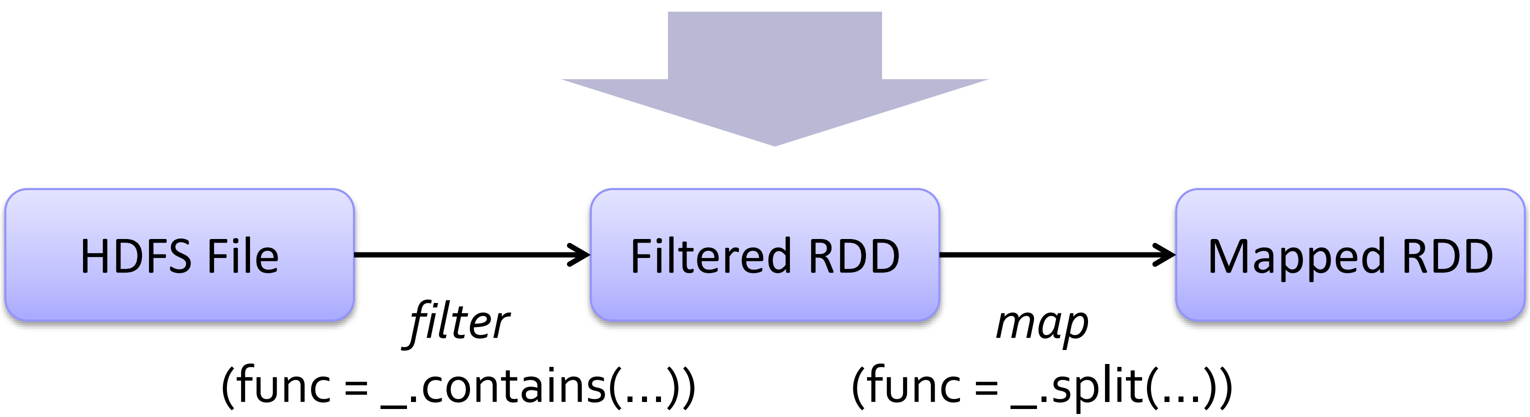
Result: scaled to 1 TB data in 5-7 sec
(vs 170 sec for on-disk data)

RDD Fault Tolerance

RDDs maintain *lineage* information that can be used to reconstruct lost partitions

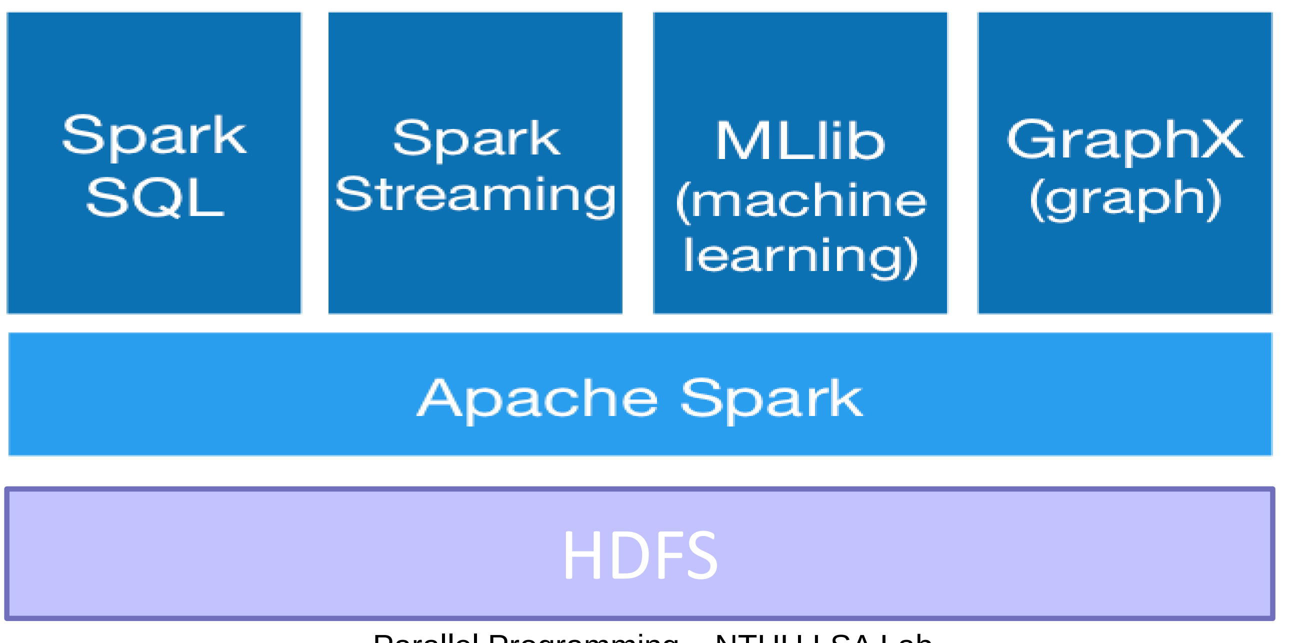
Ex:

```
messages = textFile(...).filter(_.startsWith(“ERROR”))
                        .map(_.split(‘\t’)(2))
```



Apache Spark Eco-system

- Spark SQL for relational data analytics
- Spark Streaming for streaming data processing
- MLlib for machine learning library
- GraphX for graph data processing



Reference (Papers)

- MapReduce: Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. Proceedings of the 6th Symposium on Operating System Design and Implementation (OSDI 2004), pages 137-150
- Google File System: GHEMAWAT, Sanjay; GOBIOFF, Howard; LEUNG, Shun-Tak. The Google file system. 2003.
- BigTable: Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. 2008. Bigtable: A Distributed Storage System for Structured Data. ACM Trans. Comput. Syst. 26, 2, Article 4 (June 2008), 26 pages.
- Spark: ZAHARIA, Matei, et al. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In: Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation. USENIX Association, 2012. p. 2-2.
- Hive: Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Suresh Anthony, Hao Liu, Pete Wyckoff, and Raghotham Murthy. 2009. Hive: a warehousing solution over a map-reduce framework. Proc. VLDB Endow. 2, 2 (August 2009), 1626-1629.