Big Data Analytics

Alex Rudniy, Ph.D.

Fairleigh Dickinson University

"Your recent Amazon purchases, Tweet score and location history makes you 23.5% welcome here."

Who's Generating Big Data

Social media and networks
(all of us are generating data)

Scientific instruments
(collecting all sorts of data)

Mobile devices
(tracking all objects all the time)

Sensor technology and networks
(measuring all kinds of data)

- The progress and innovation is no longer hindered by the ability to collect data
- But, by the ability to manage, analyze, summarize, visualize, and discover knowledge from the collected data in a timely manner and in a scalable fashion

Bits to Megabytes

- 1 **Bit**
  - Binary Digit, 0 or 1
- 1 **Byte**
  - = 8 Bits
  - A single character
- 1 **Kilobyte**
  - = 1024 Bytes
  - A very short story
- 1 **Megabyte**
  - = 1024 Kilobytes ≈1,000,000 bytes
  - A small novell

Terabyte to Exabyte

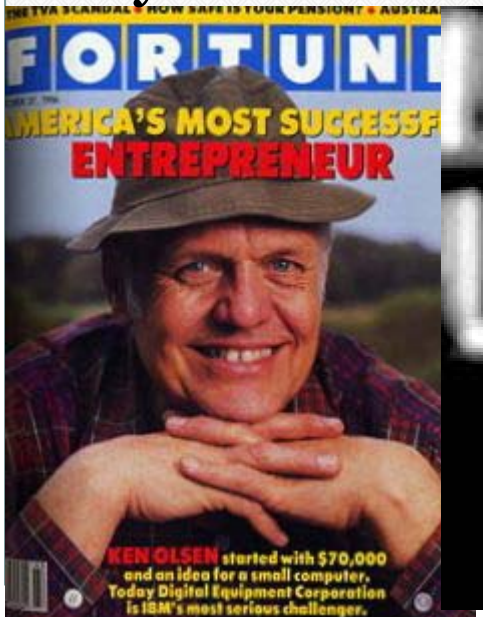
- 1 **Terabyte**
  - = 1024 Gigabytes
  - All X-ray images at a large hospital
- 1 **Petabyte**
  - =1024 Terabytes
  - All US academic libraries
- 1 **Exabyte**
  - =1024 Petabytes
  - All words ever spoken by people

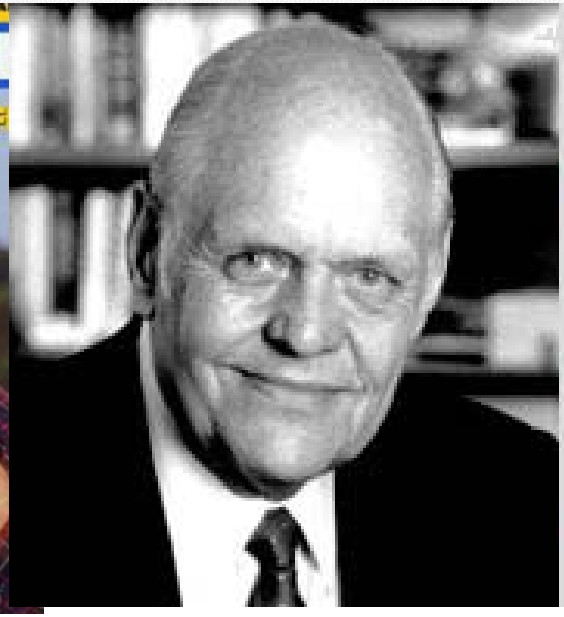
Wait, there is more!

- 1 **Zettabyte**
  - = 1024 Exabytes
  - 1,000,000,000,000,000,000,000 Bytes (21 zeroes)
  - World internet traffic
- 1 **Yottabyte**
  - =1024 Zettabytes
  - 1,000,000,000,000,000,000,000,000 Bytes (24 zeroes)
- 1 **Xenottabyte**
  - 1 000 000 000 000 000 000 000 000 000 Bytes (27 zeroes)
  - = 1024 Yottabytes
- **Shilentnobyte**
  - 1 000 000 000 000 000 000 000 000 000 000 Bytes (30 zeroes)
- **Domegemegrottebyte**
  - 1 000 000 000 000 000 000 000 000 000 000 000 Bytes (33 zeroes)

How much data?

- Google processes 20 PB a day
- Wayback Machine has 3 PB + 100 TB/month
- Facebook has 2.5 PB of user data + 15 TB/day
- eBay has 6.5 PB of user data + 50 TB/day
- CERN's Large Hydron Collider (LHC) generates 15 PB a year

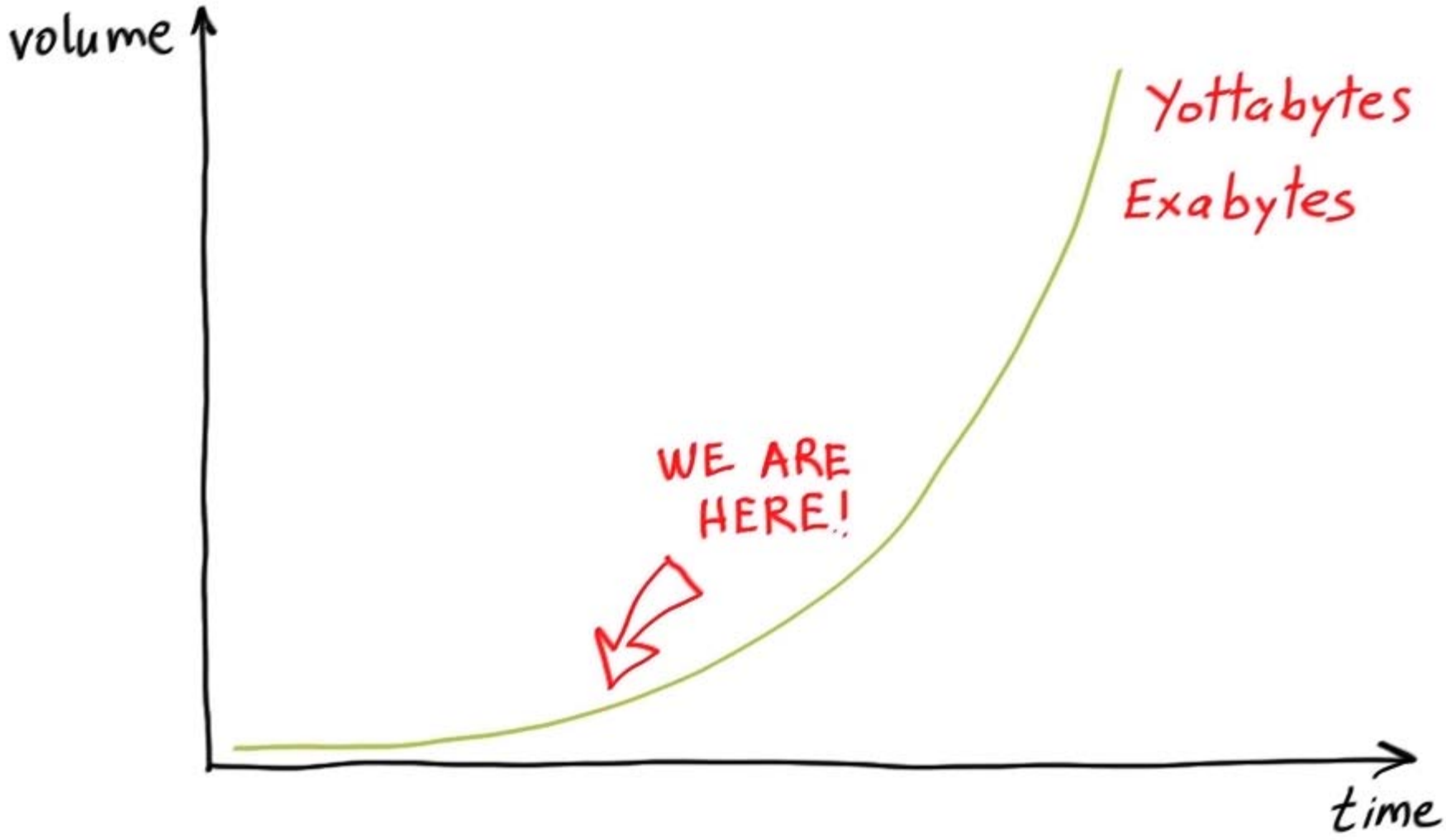




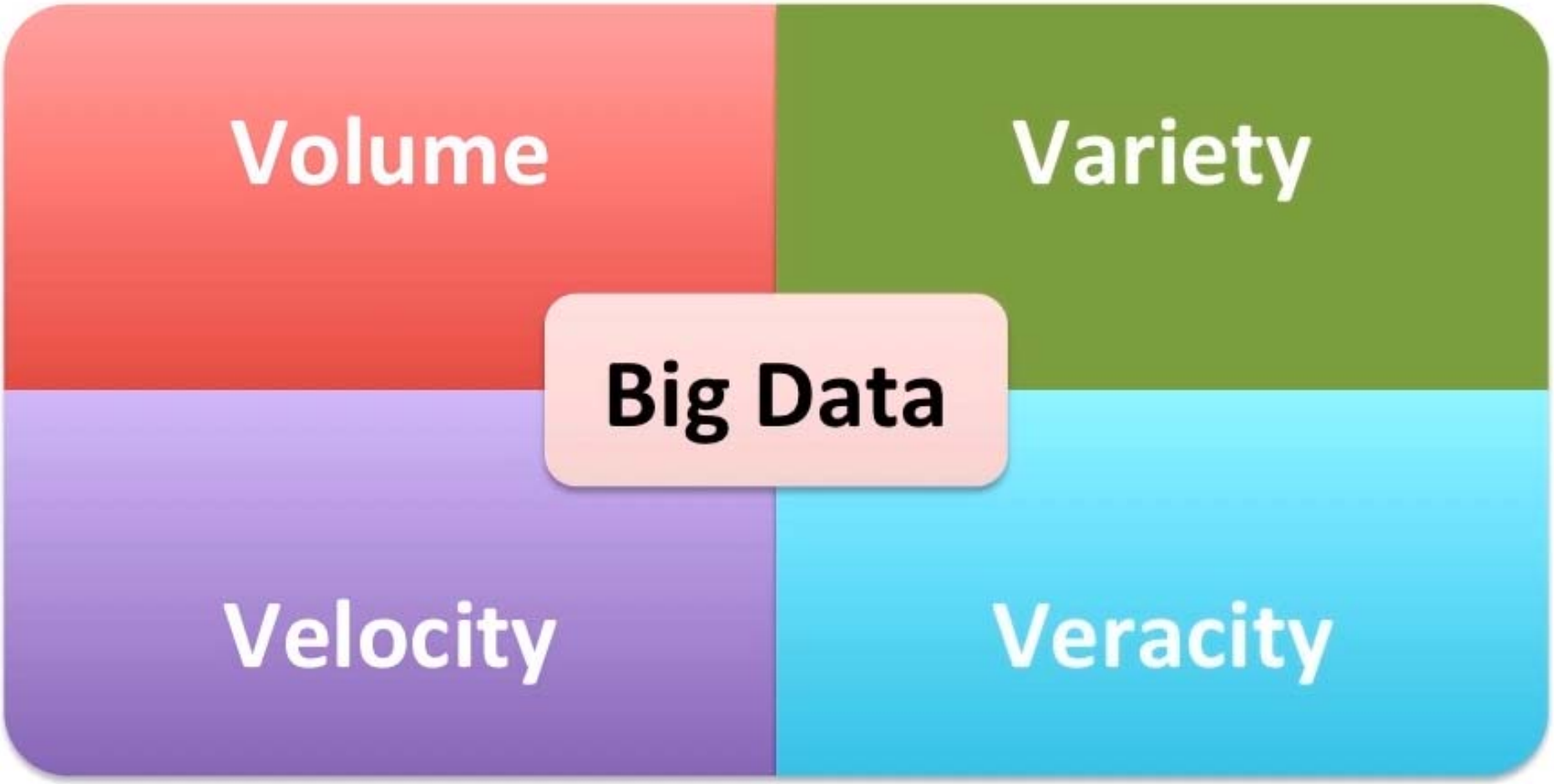
"There is no reason for any individual to have a computer in his home."

Ken Olsen

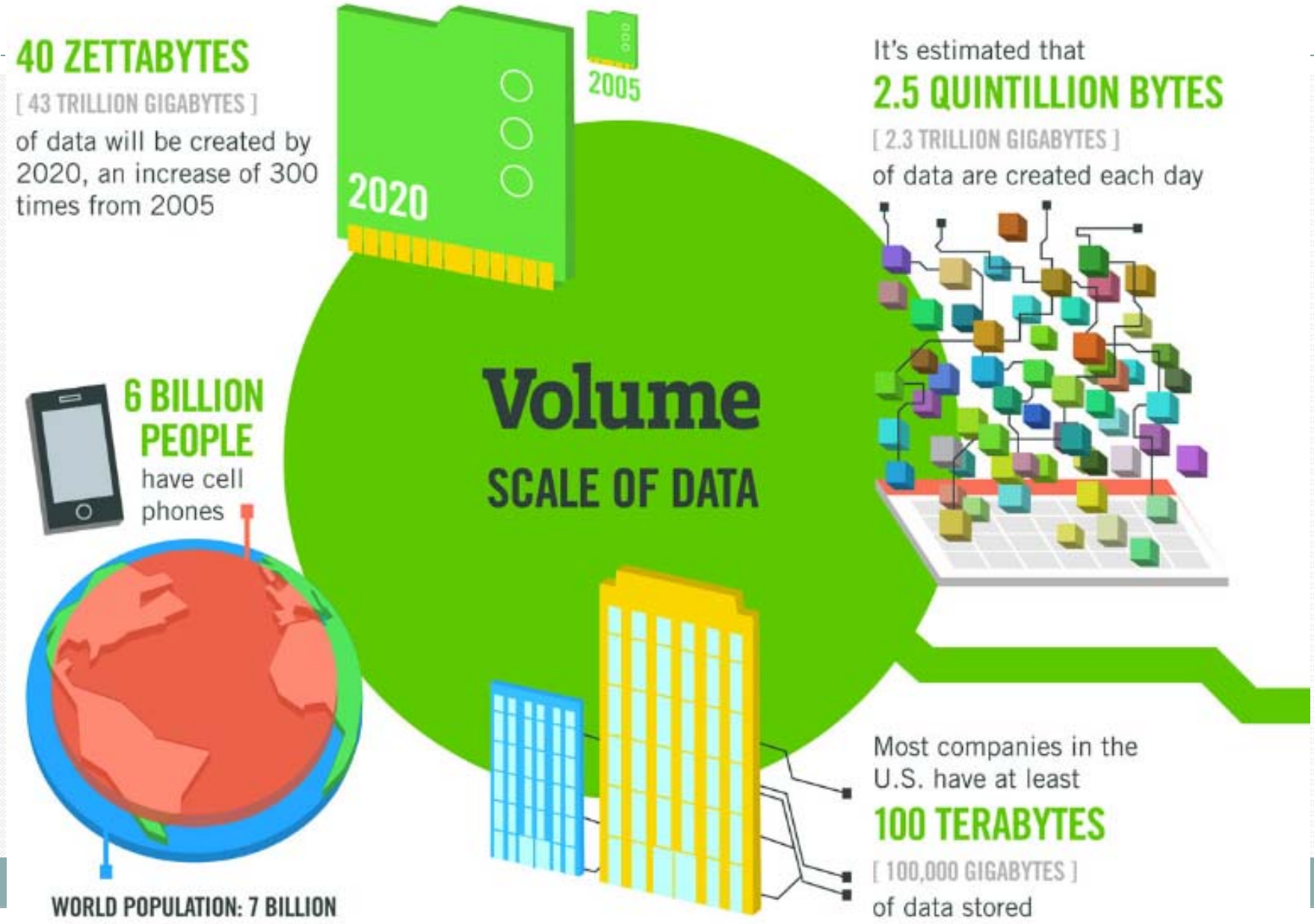
Big Data Present Moment



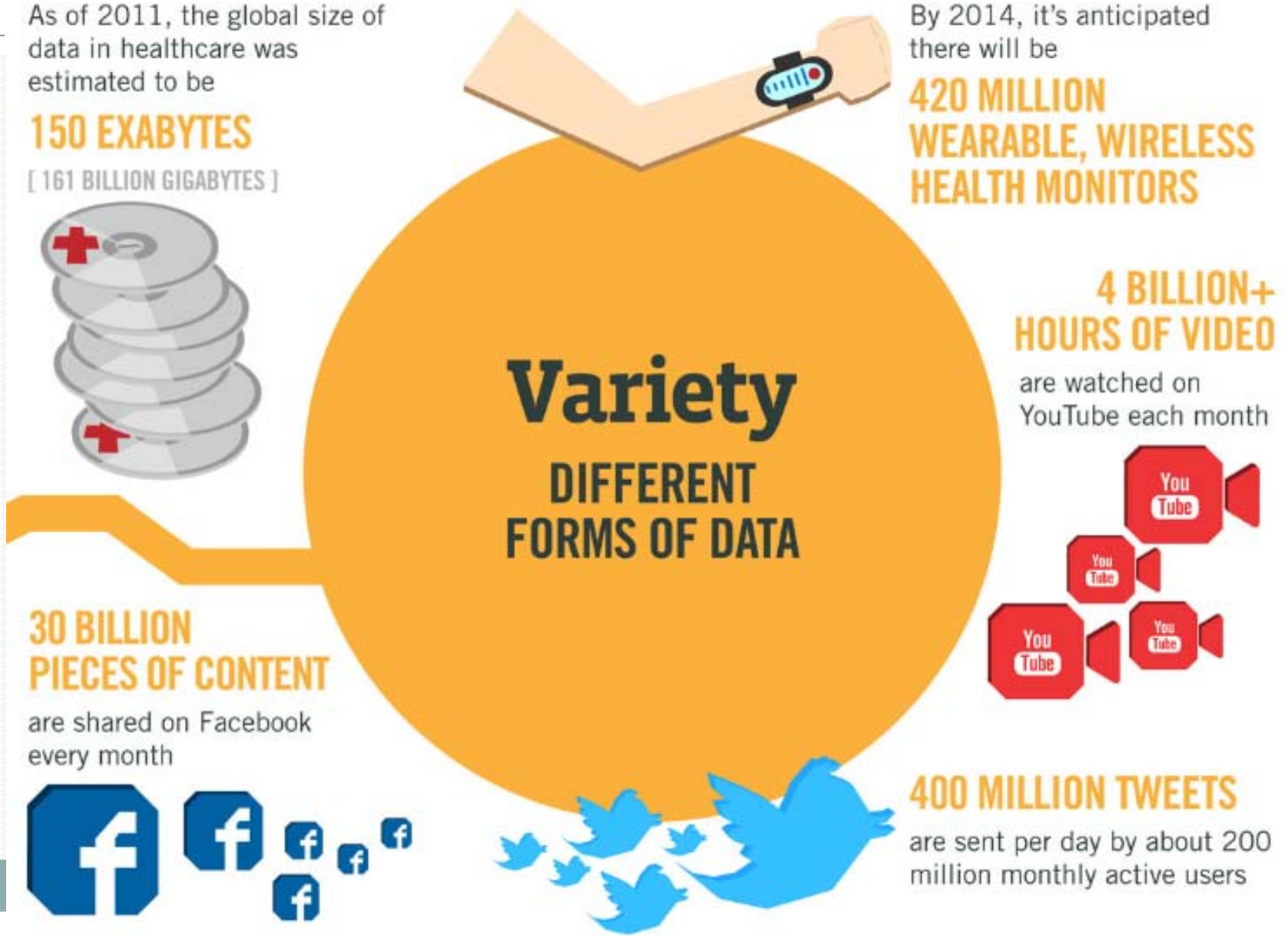
Big Data Characteristics



Characteristics of Big Data: **1 Volume**



Characteristics of Big Data: **2 Variety**



Characteristics of Big Data: **3 Velocity**

The New York Stock Exchange captures

1 TB OF TRADE INFORMATION

during each trading session

Modern cars have close to

100 SENSORS

that monitor items such as fuel level and tire pressure

Velocity

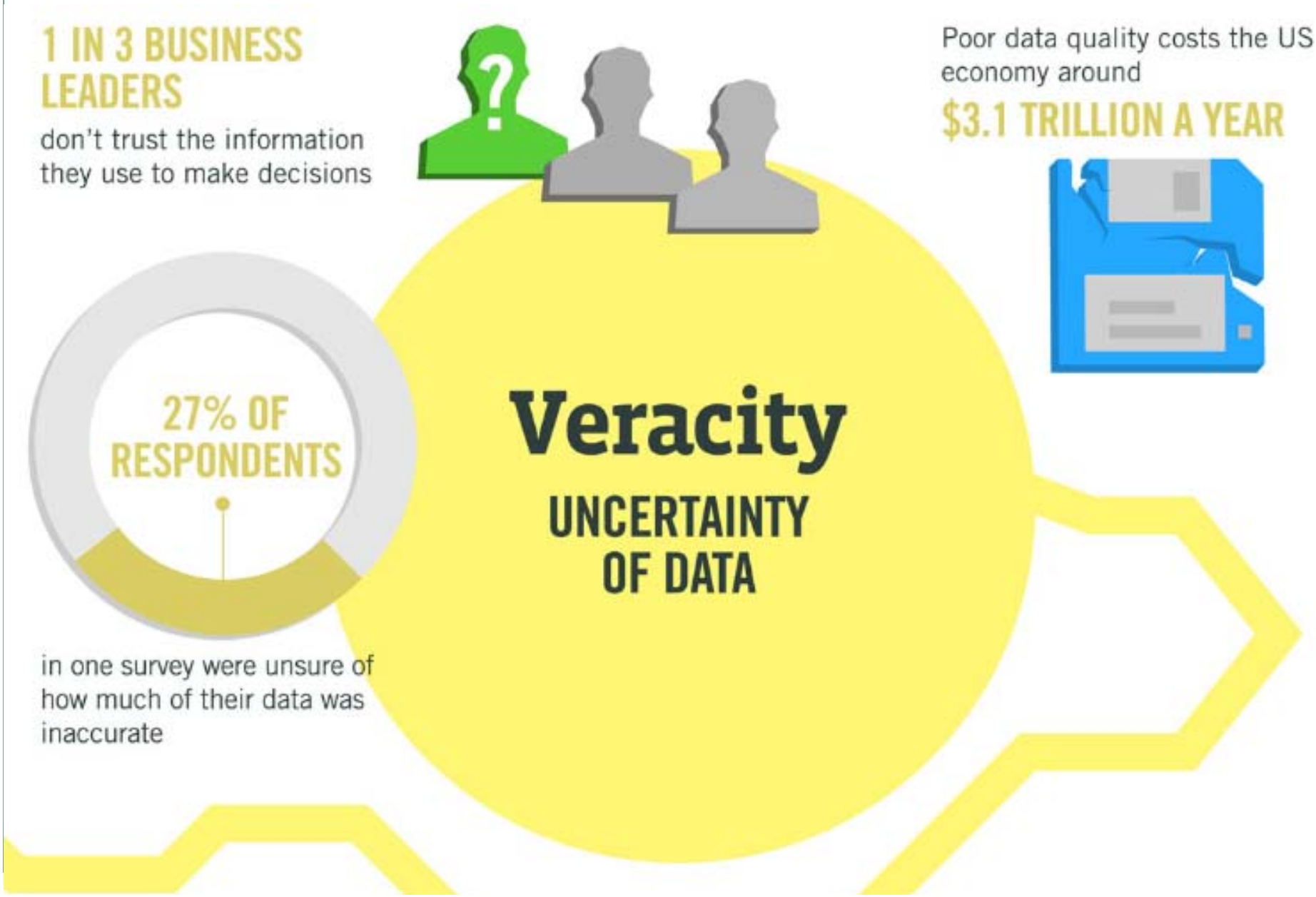
ANALYSIS OF STREAMING DATA

By 2016, it is projected there will be

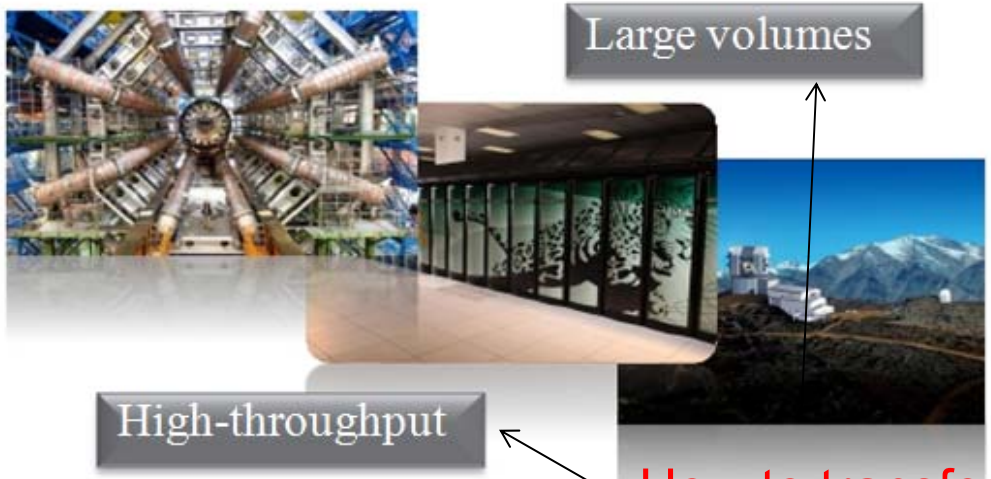
18.9 BILLION NETWORK CONNECTIONS

– almost 2.5 connections per person on earth

Characteristics of Big Data: **4 Veracity**



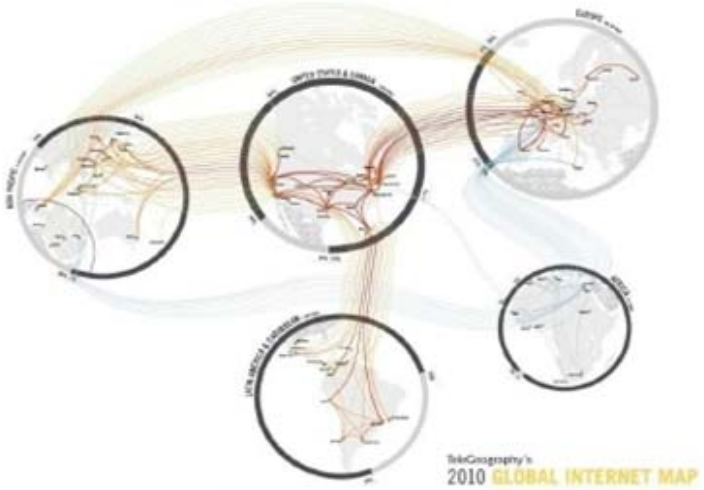
Challenges



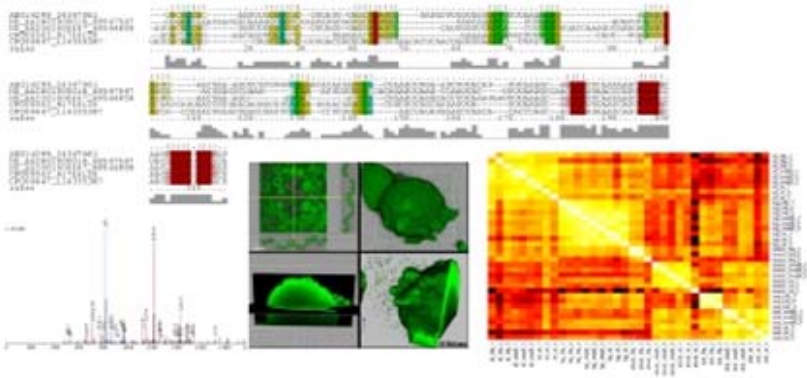
Large volumes

High-throughput

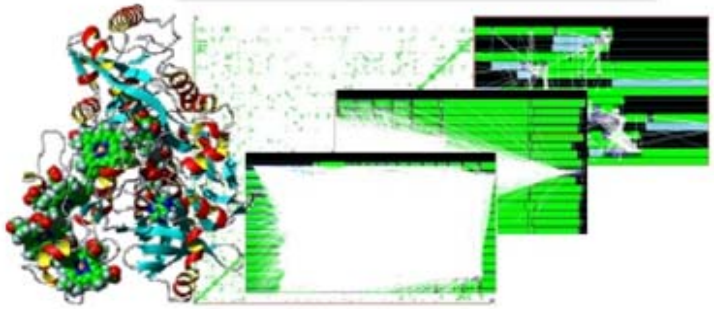
How to transfer Big Data?



Relating and Linking

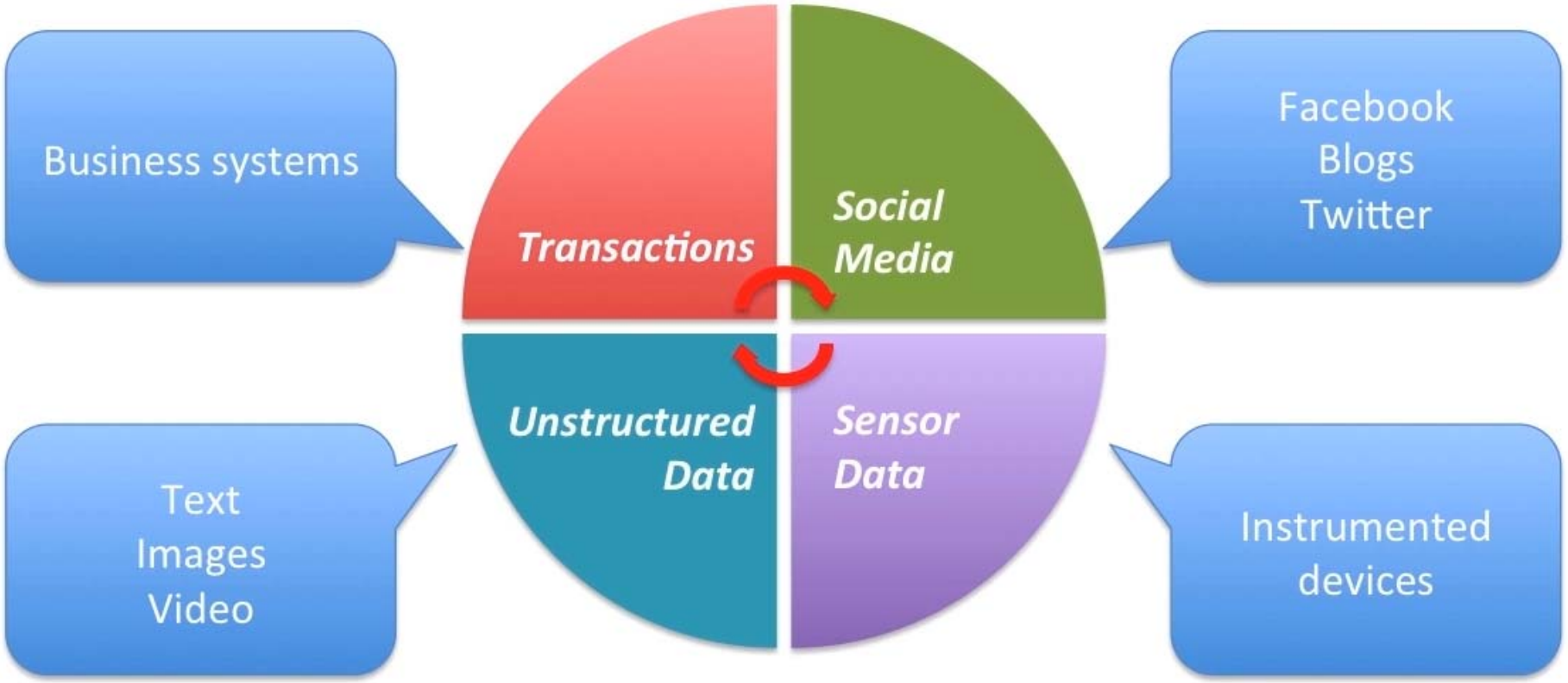


Heterogeneity

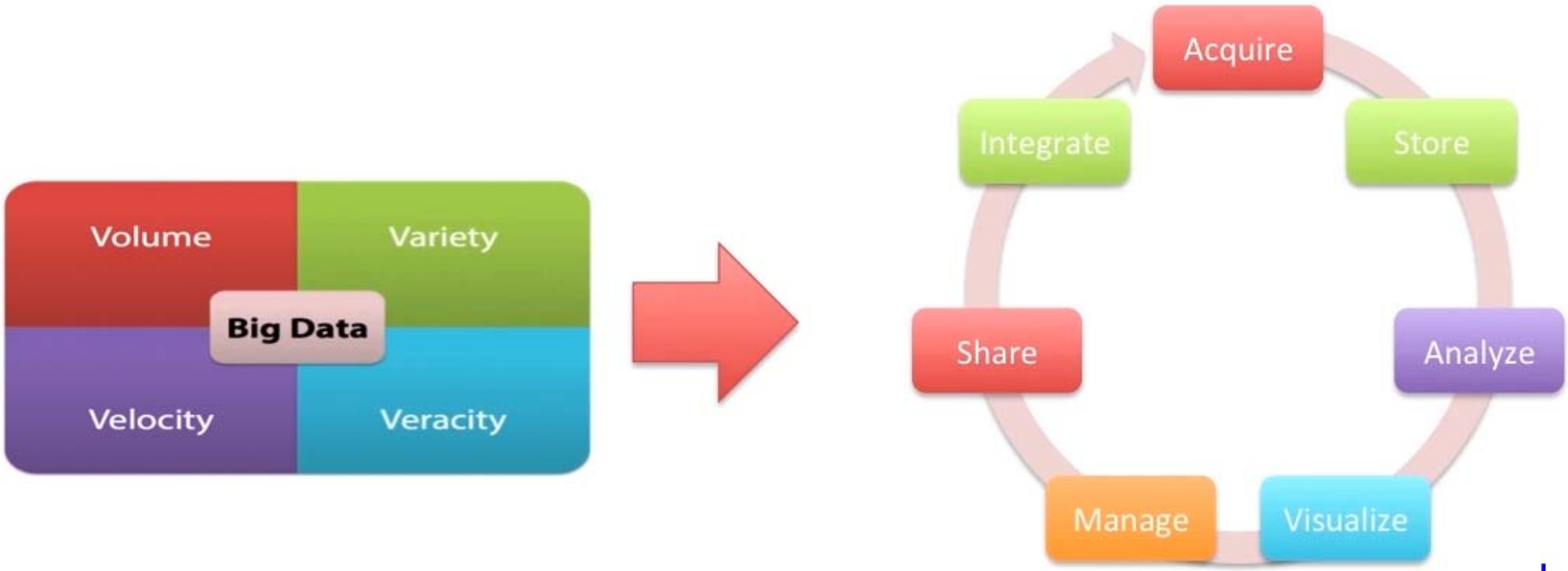


Complexity

Big Data Sources



Big Data Workflow



- Process workflow timely and cost-effectively

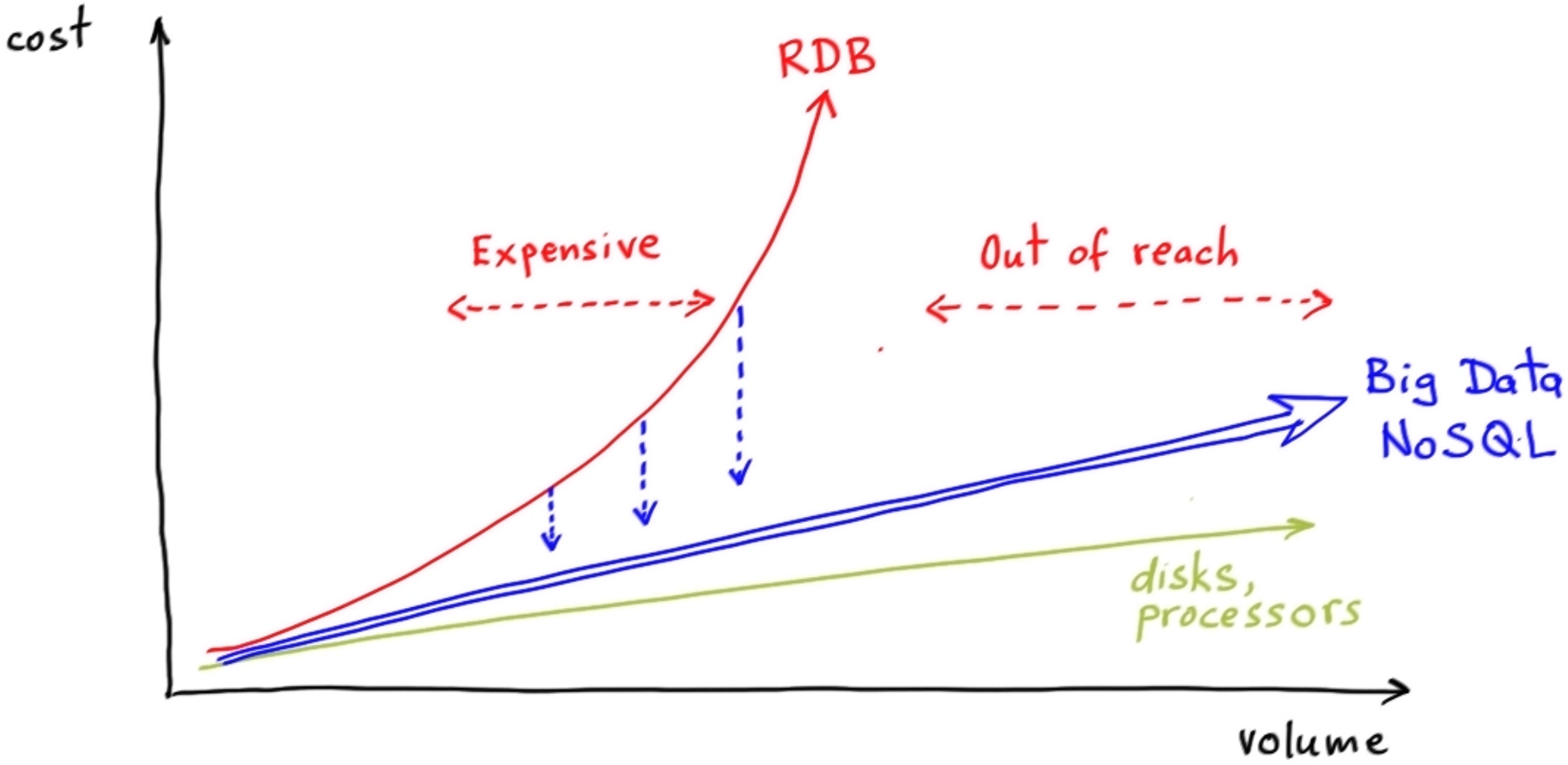
Big Data-Related Problems

- More data than traditional system can handle
  - Not always extremely high volumes
- Unstructured data
  - Data does not fit in conventional storage
- Data arrives quickly
  - May have very narrow usefulness window

More Big Data-Related Problems

- So much data – not clear what to analyze
- Integrate new data with conventional systems
- **Lack of experts to process, manage, and analyze data**
- Organizational silos

Challenges of Relational Databases



COURSE:

BIG DATA ANALYTICS WITH HADOOP AND R

Hive Data Warehouse

- Data warehousing infrastructure based on the Hadoop
- Massive scale out and fault tolerance
- Originated at Facebook in 2007, now Apache project
- HiveQL: Query language based on SQL
  - Tables are defined on top of HDFS files
  - Queries transformed into MapReduce tasks
- Good candidate when transitioning from Rdb to Hadoop
  - HiveQL will feel familiar

Scalding

- A library built on top of Scala
- Elegant model: programs look like manipulating in-memory data structures, get translated into MapReduce
- Very short programs
- Full programming environment
  - Full development ecosystem

Apache Pig

PLATFORM FOR ANALYZING LARGE DATA SETS

Ambari

adoop cluster

rks Hadoop

nance of hosts

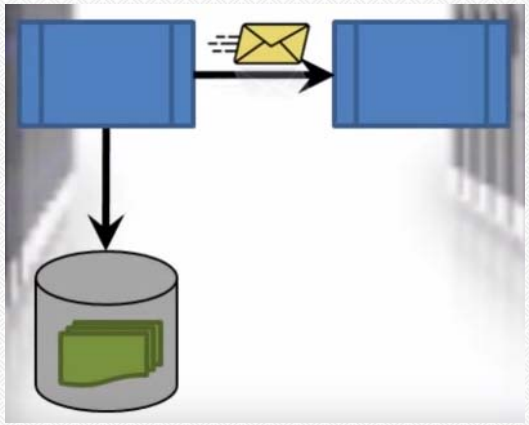
provision,
e Hadoop

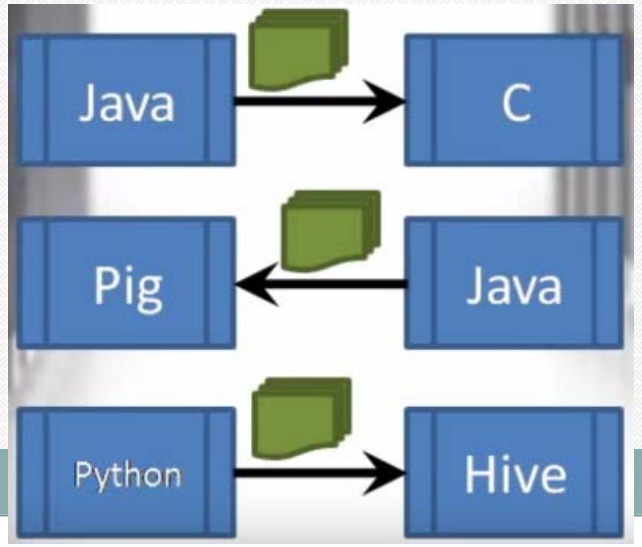
ng cluster
ion

nterface to:
components

Apache Avro

- Provides two important services for Hadoop
  - Data serialization
  - Data exchange
- Avro can serialize data into
  - Files
  - Messages
- Avro allows to exchange data between any language





Chukwa

- Large-scale log collection and analysis
- Logs are on remote machine
- Hard to collect/access logs from thousand of machines
- Hard to correlate information from different system
- Unable to extract useful information from terabytes of data
- No easy way to detect failures on thousand of machines

HBase

- HBase is a distributed column-oriented data store built on top of HDFS
- HBase is an Apache open source project whose goal is to provide storage for the Hadoop Distributed Computing
- Data is logically organized into tables, rows and columns

Apache Spark

FAST, IN-MEMORY DATA PROCESSING ENGINE

Apache Tez

APPLICATION FRAMEWORK FOR A COMPLEX DIRECTED-ACYCLIC-GRAPH OF TASKS FOR PROCESSING DATA.

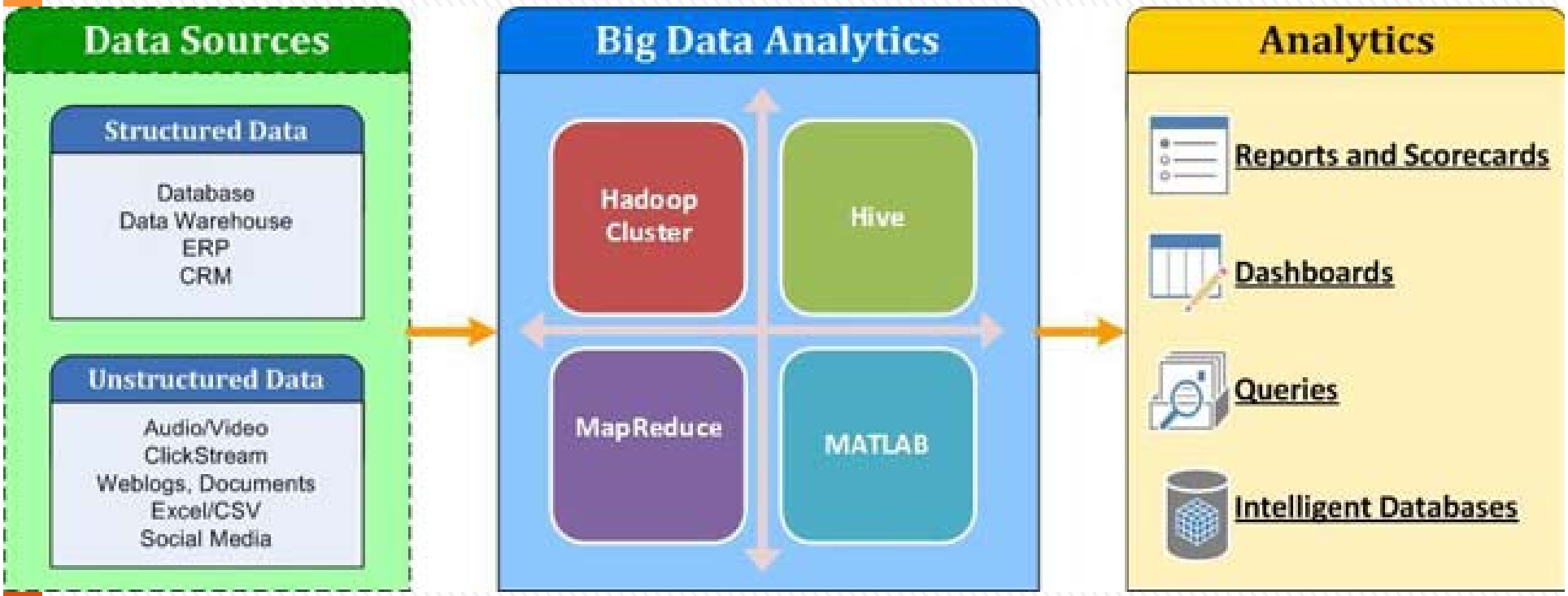
Apache Mahout

SCALABLE MACHINE LEARNING

Apache Zookeper

OPEN-SOURCE SERVER FOR HIGHLY RELIABLE DISTRIBUTED COORDINATION

Business Intelligence and Big Data

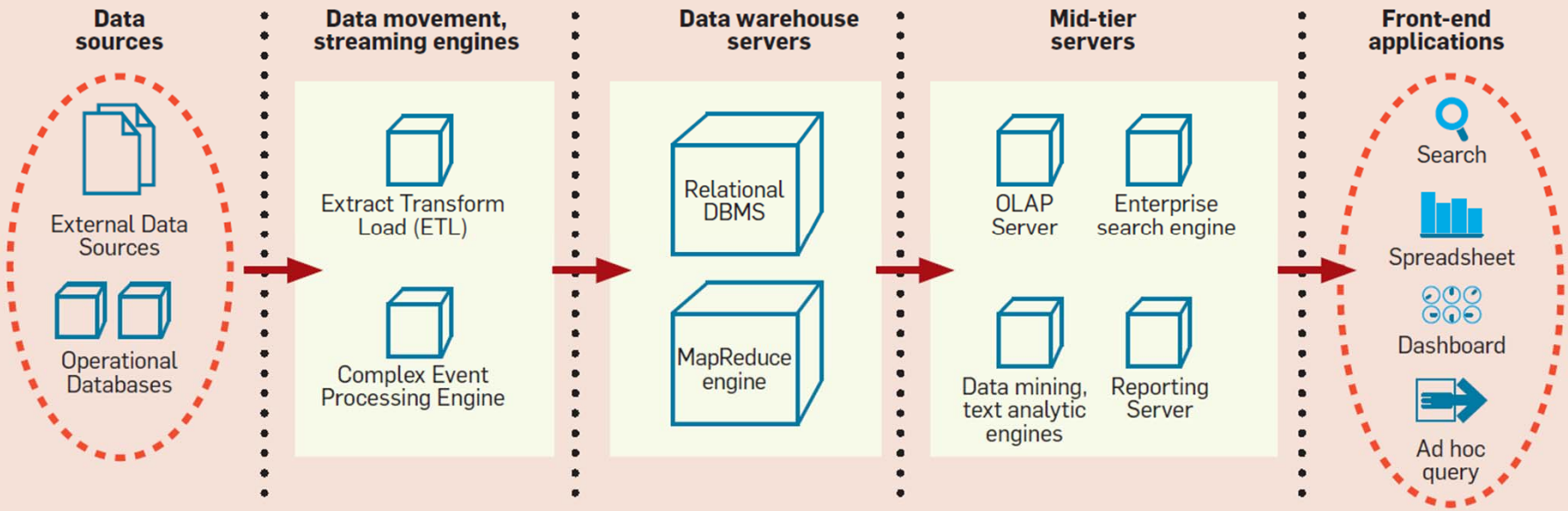


Where BI is applied? Examples:

- **Manufacturing**
  > for order shipment and customer support
- **Retail** →
  > for user profiling to target grocery coupons during checkout
- **Financial services** → for claims analysis and fraud detection
- **Transportation** → for fleet management
- **Telecommunications** → find reasons for customer churn
- **Utilities** → for power usage analysis
- **Health care** → outcomes analysis.

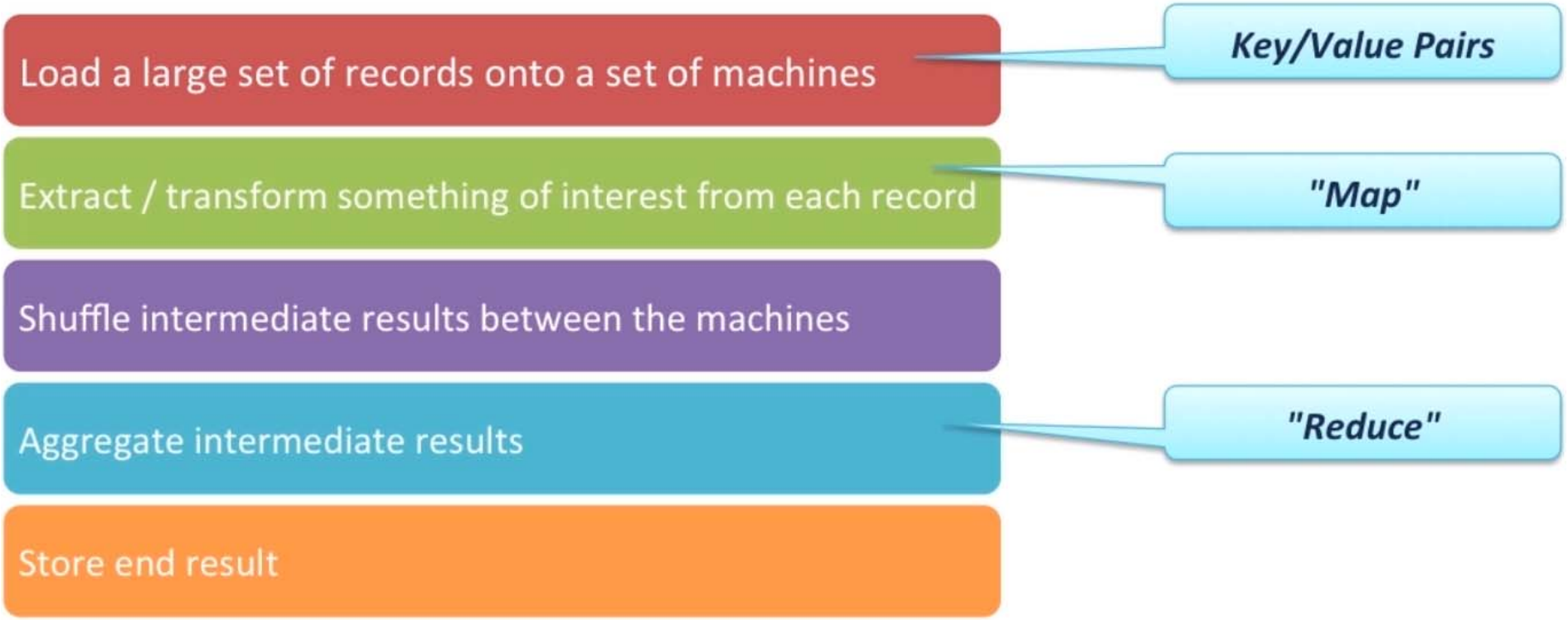
Typical BI Architecture

Figure 1. Typical business intelligence architecture.



MapReduce and Hadoop

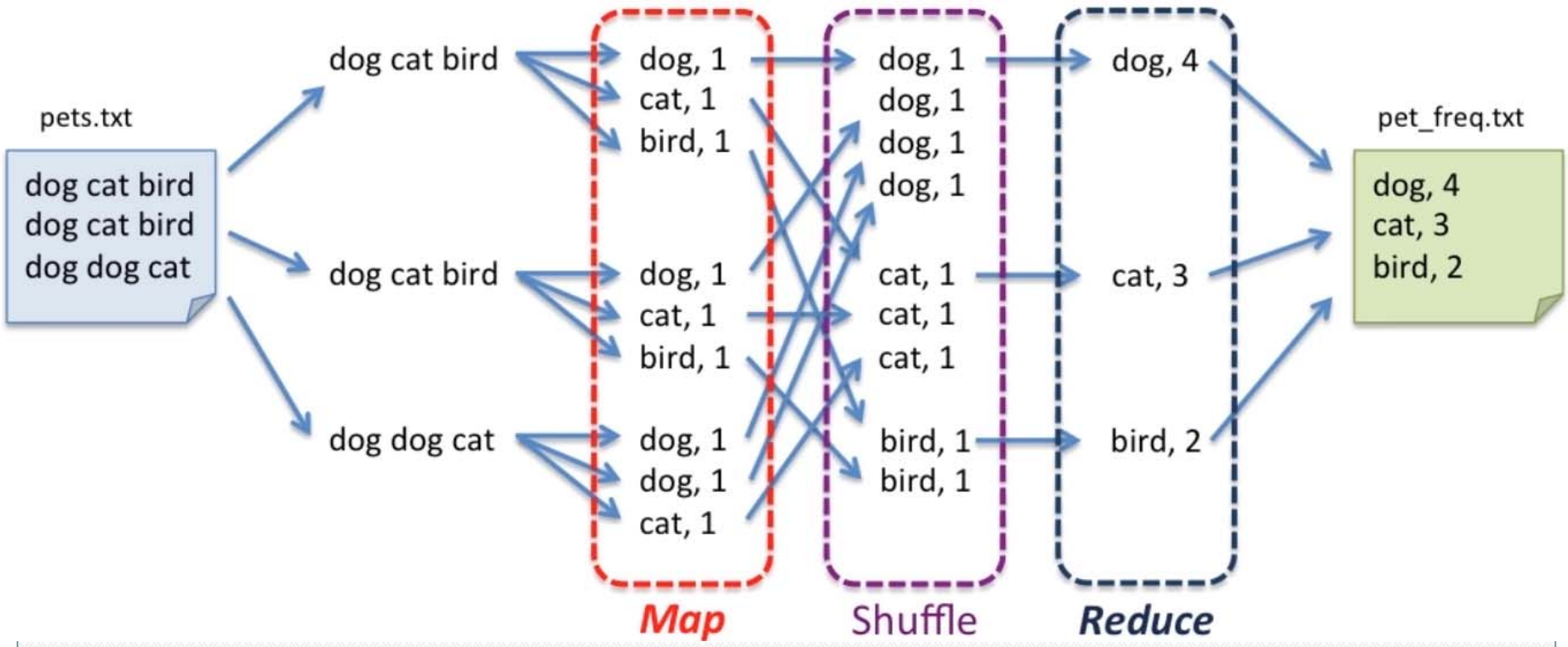
- Process large volumes of data with many machines



MapReduce

- Programming model for **parallel** data processing.
- Hadoop can run MapReduce programs written in **various languages**.
- MapReduce breaks processing into 2 phases:
  1. **Map** phase
  2. **Reduce** phase.
- Each phase has key-value pairs as input and output, the types of which may be chosen by the programmer.
- The programmer specifies two functions: the **map function** and the **reduce function**.

MapReduce Algorithm—**Simple Example**



MapReduce—**Map Step**

- "Map" Step:
  - Input split into pieces
  - Worker nodes process individual pieces in parallel
  - Each worker node stores its result in its local file system
  - Reducer accesses files produced by workers

MapReduce—**Reduce Step**

- "Reduce" step:
  - Data is aggregated by worker nodes
    - This is called "reduced" after the map steps
  - Multiple reduce tasks can be executed in parallel

MapReduce Example

- Task: Write program that mines data
- Given: Weather sensors collect data every hour at many locations across the globe (since 1940s)
- Use MapReduce:
  - We want to process all the data
  - Data is semi-structured and record-oriented

Data for MapReduce Example

- Each line is a record
- Contains several meteorological elements
- Some are optional
- Some are of variable data length
- Record example:

```
57332130999991950010130045131728783FM-
1217199999V0203201N7214501CN100001N9-
1281-1391102681
```

Formatted Record (part 1)

```
0057
332130   # USAF weather station identifier
99999    # WBAN weather station identifier
19500101 # observation date
0300     # observation time
4
+51317   # latitude (degrees x 1000)
+028783  # longitude (degrees x 1000)
FM-12
+0171    # elevation (meters)
99999
V020
320      # wind direction (degrees)
1        # quality code
N
0072
```

Formatted Record (part 2)

```
1
00450    # sky ceiling height (meters)
1        # quality code
C
N
010000   # visibility distance (meters)
1        # quality code
N
9
-0128    # air temperature (degrees Celsius x 10)
1        # quality code
-0139    # dew point temperature (degrees Celsius x 10)
1        # quality code
10268    # atmospheric pressure (hectopascals x 10)
1        # quality code
```

Example data continued …

- Datafiles are organized by date and weather station.
- Folder for each year from 1901 to 2001
- Folders contain files per weather station per year.

Some files in the 1990 folder

```
% ls raw/1990 | head
010010-99999-1990.gz
010014-99999-1990.gz
010015-99999-1990.gz
010016-99999-1990.gz
010017-99999-1990.gz
010030-99999-1990.gz
010040-99999-1990.gz
010080-99999-1990.gz
010100-99999-1990.gz
010150-99999-1990.gz
```

Question

- What's the highest recorded global temperature for each year in the dataset?

Map function (for this case)

- Pull out year and temperature
- Prepares data so that **reduce function** can find the max temperature for each year.
- In this example, **map function** also drops bad records
  - Remove temperatures that are missing, suspect, or wrong.

Map function input

- Lines in input files

```
0067011990999991950051507004...9999999N9+00001+99999999999...
0043011990999991950051512004...9999999N9+00221+99999999999...
0043011990999991950051518004...9999999N9-00111+99999999999...
0043012650999991949032412004...0500001N9+01111+99999999999...
0043012650999991949032418004...0500001N9+00781+99999999999...
```

- Transformed to input for **`map`**

```
(0, 0067011990999919500051507004...9999999N9+00001+99999999999...)
(106, 0043011990999919500051512004...9999999N9+00221+99999999999...)
(212, 0043011990999919500051518004...9999999N9-00111+99999999999...)
(318, 0043012650999919490032412004...0500001N9+01111+99999999999...)
(424, 0043012650999919490032418004...0500001N9+00781+99999999999...)
```

- Output

```
(1950, 0)
(1950, 22)
(1950, -11)
(1949, 111)
```

Reduce function input & output

- The output from **map function** is shuffled
  - Sorted, merged, grouped and distributed to reducers
- Afterwards, **reduce function input**:

```
(1949, [111, 78])
(1950, [0, 22, -11])
```

- **Reduce function output**:

```
(1949, 111)
(1950, 22)
```

Separation of Work

Programmers

- Map
- Reduce

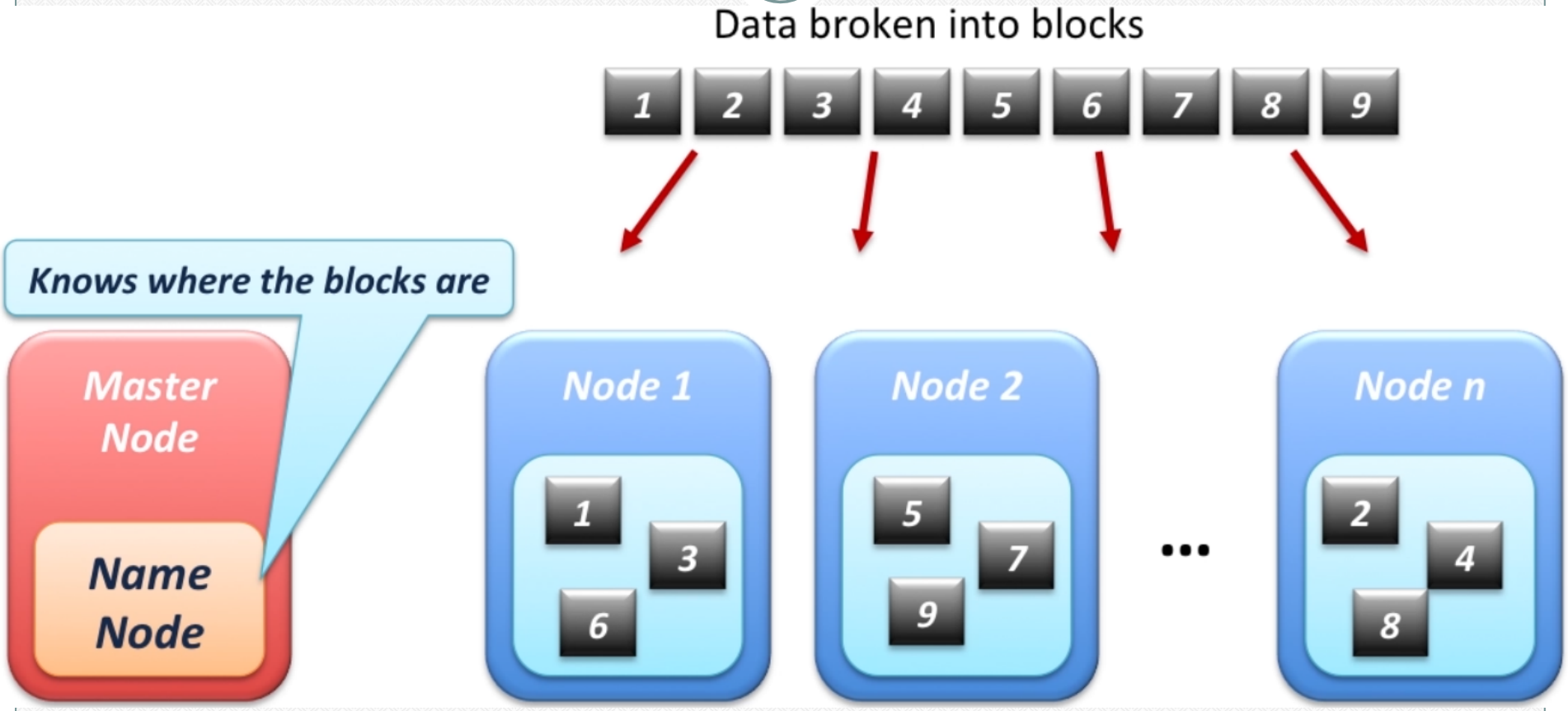
Framework

- Deals with fault tolerance
- Assign workers to map and reduce tasks
- Moves processes to data
- Shuffles and sorts intermediate data
- Deals with errors

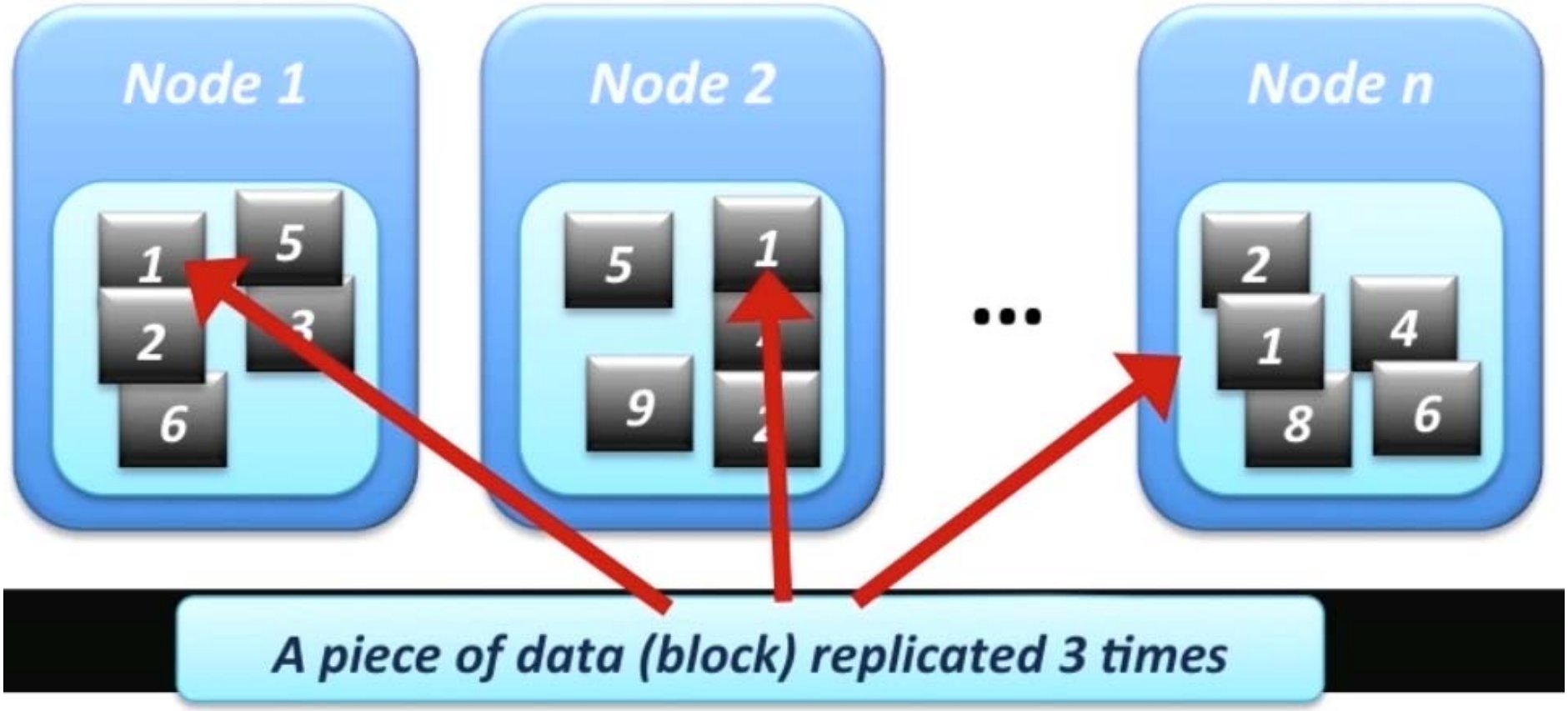
Apache Hadoop

- Open-source framework for processing large amounts of data, spread across multiple machines
- Designed to work on clusters of machines
  - Cheap and unreliable clusters
- Inspired by Google technologies
- Implemented in Java
- The system is designed to scale

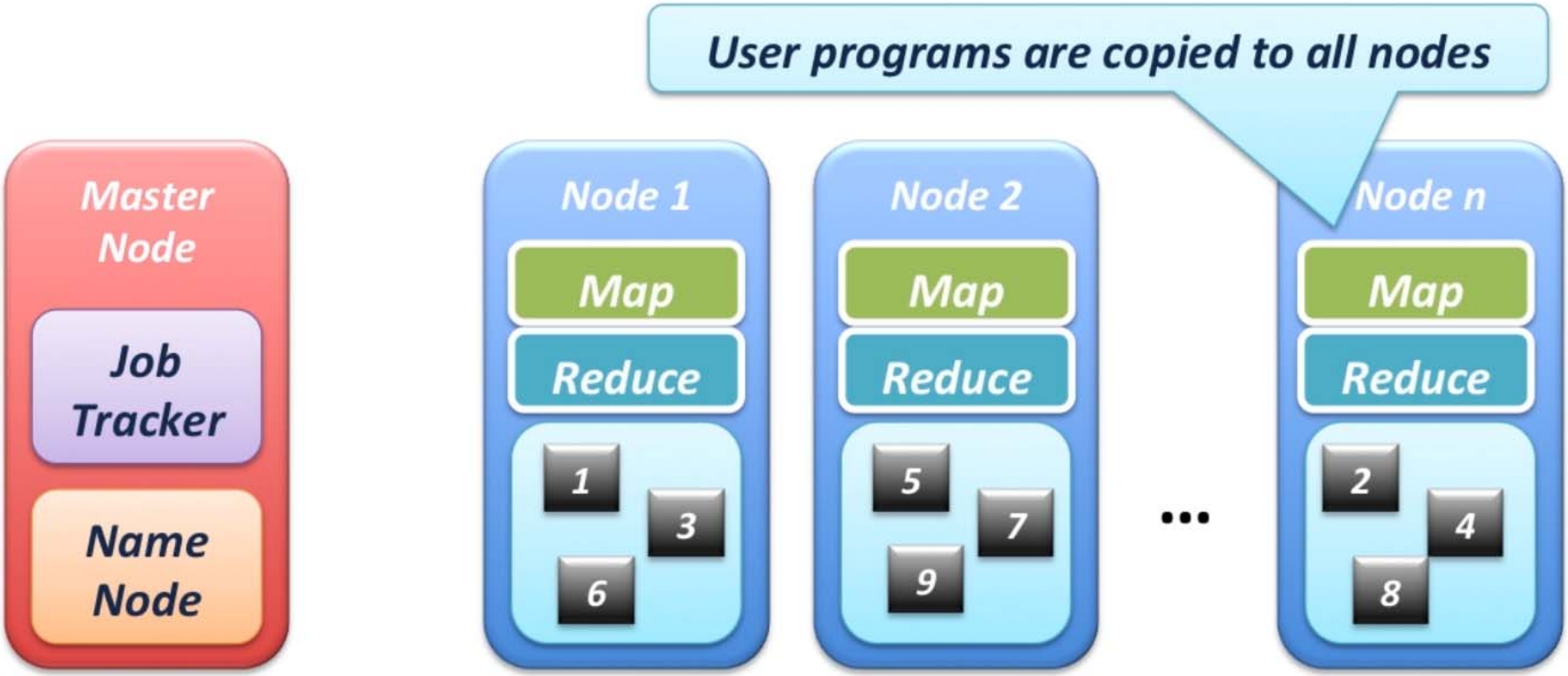
Hadoop Distributed File System



HDFS Replication

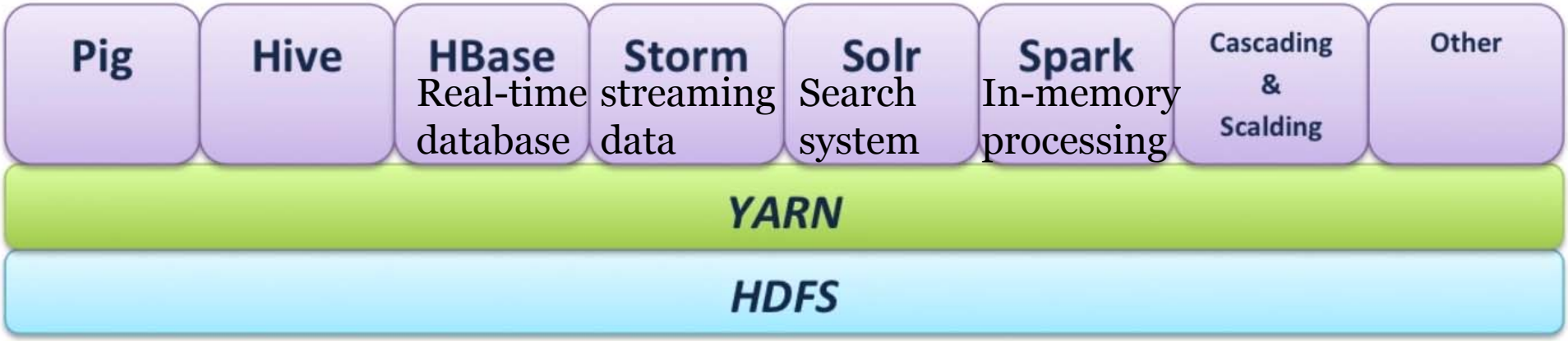


Map and Reduce Tasks on Nodes



Hadoop Architecture

Programming Languages



- Yarn: A common resource management for applications
  - Scalability
  - Improved cluster utilization
  - Workloads other than MapReduce