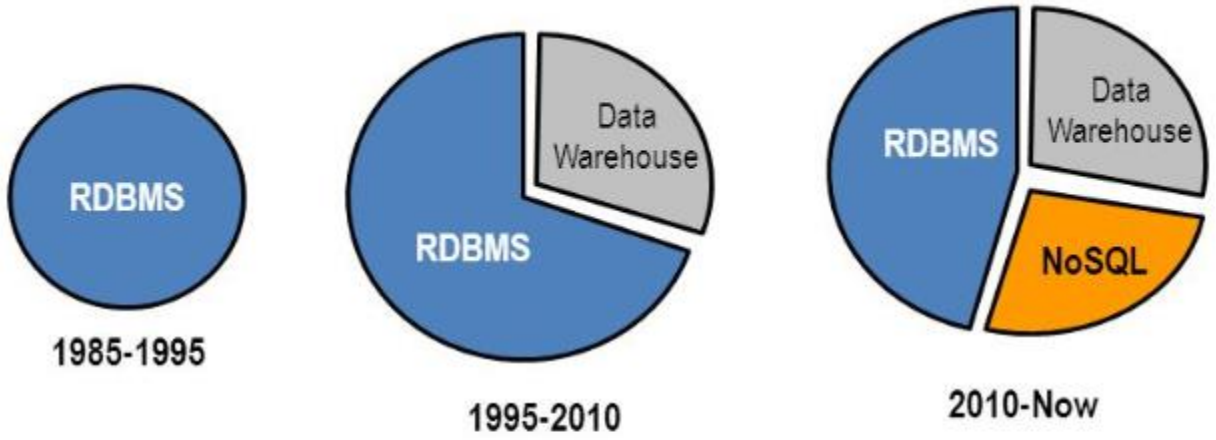
Big Data & NOSQL

Database Evolution

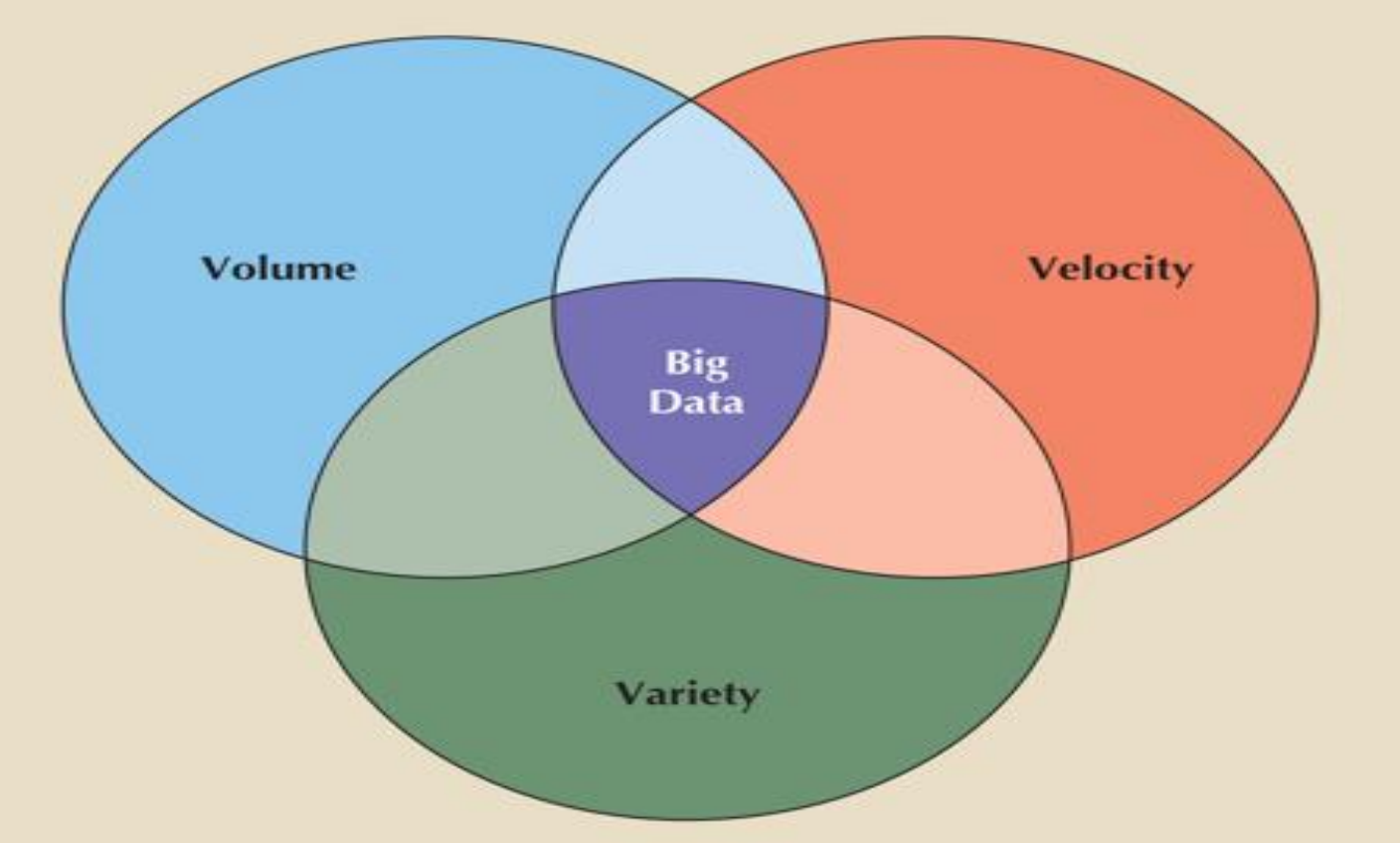


- RDBMS for transactions, Data Warehouse for analytics and NoSQL for scalability

Big Data V's

- Volume: quantity of data to be stored
  - Scaling up: keeping the same number of systems but migrating each one to a larger system
  - Scaling out: when the workload exceeds server capacity, it is spread out across a number of servers
- Velocity: speed at which data is entered into system and must be processed
  - Stream processing: focuses on input processing and requires analysis of data stream as it enters the system
  - Feedback loop processing: analysis of data to produce actionable results
- Variety: variations in the structure of data to be stored
  - Structured data: fits into a predefined data model
  - Unstructured data: does not fit into a predefined model

Big Data V's (con't)



Big Data Other Characteristics

- Other characteristics
  - Variability: changes in meaning of data based on context
  - Sentimental analysis: attempts to determine if a statement conveys a positive, negative, or neutral attitude about a topic
  - Veracity: trustworthiness of data
  - Value: degree data can be analyzed for meaningful insight
  - Visualization: ability to graphically resent data to make it understandable
- Relational databases are not necessarily the best for storing and managing all organizational data
  - Polyglot persistence: coexistence of a variety of data storage and management technologies within an organization's infrastructure

NoSQL
[Not Only SQL]

- Highly flexible and hugely scalable, NoSQL databases offer a range of data models and consistency options
- The modern sense of NoSQL, which dates from 2009, refers to databases that are not built on relational tables, unlike SQL databases
- Often, NoSQL databases boast better design flexibility, horizontal scalability, and higher availability than traditional SQL databases, sometimes at the cost of weaker consistency

NoSQL (con't)

- NoSQL databases can take a number of forms
- They can be cloud services or install on-premises
- They can support one or more data models: key-value, document, column, graph, and sometimes even relational—which is one reason that NoSQL is sometimes parsed as "Not Only SQL"
- They can also support a range of consistency models, from strong consistency to eventual consistency

NoSQL (con't)

- Key-value is the most basic of the four non-relational data models; sometimes other database models are implemented on top of a key-value foundation layer
- Column databases have keys, values, and timestamps; the timestamp is used for determining the valid content; Cassandra is a prominent example of a column database
- Document stores, such as MongoDB, have a query language or API for finding documents by content; they also have key lookups, like key-value stores
- Graph databases, such as Neo4j, explicitly express the connections between nodes; this makes them more efficient at the analysis of networks (computer, human, geographic, etc)

NoSQL (con't)

- A few databases expose multiple data models
- Some, such as Azure Cosmos DB, isolate the data models from each other
- Others, such as FaunaDB, combine the data models
- Some of these databases support globally distributed data and possibly automatic sharding; for example
  - Amazon DocumentDB replicates six copies of your data across three AWS Availability Zones and allows for up to 15 read replicas
  - Amazon DynamoDB supports multiple regions and global tables
- Azure Cosmos DB is globally distributed and horizontally partitioned
- YugaByte DB not only was designed for planet-scale applications, but also supports multi-cloud clusters, automatic sharding and rebalancing, and distributed ACID transactions

Sharding

- A database shard, or simply a shard, is a horizontal partition of data in a database or search engine
- Each shard is held on a separate database server instance, to spread load
- Some data within a database remains present in all shards, but some appears only in a single shard
- By spreading across multiple nodes one achieves horizontal scalability and improved performance
  - With massively parallel processing, you can take advantage of all the compute resources across your cluster for every query
  - Because the individual shards are smaller than the logical table as a whole, each machine has to scan fewer rows when responding to a query

Sharding (con’t)

Original Table

CUSTOMER ID	FIRST NAME	LAST NAME	CITY
1	Alice	Anderson	Austin
2	Bob	Best	Boston
3	Carrie	Conway	Chicago
4	David	Doe	Denver

Vertical Shards

VS1

CUSTOMER ID	FIRST NAME	LAST NAME
1	Alice	Anderson
2	Bob	Best
3	Carrie	Conway
4	David	Doe

VS2

CUSTOMER ID	CITY
1	Austin
2	Boston
3	Chicago
4	Denver

Horizontal Shards

HS1

CUSTOMER ID	FIRST NAME	LAST NAME	CITY
1	Alice	Anderson	Austin
2	Bob	Best	Boston

HS2

CUSTOMER ID	FIRST NAME	LAST NAME	CITY
3	Carrie	Conway	Chicago
4	David	Doe	Denver

The CAP Theorem

- Consistency - All the database partition replicas see the same data at any given point in time
- Availability - Unless a node has failed, it will respond to a request
- Partition tolerance - the distributed database is doing its job, and therefore if we can't get to the data on one partition, it will be available on another

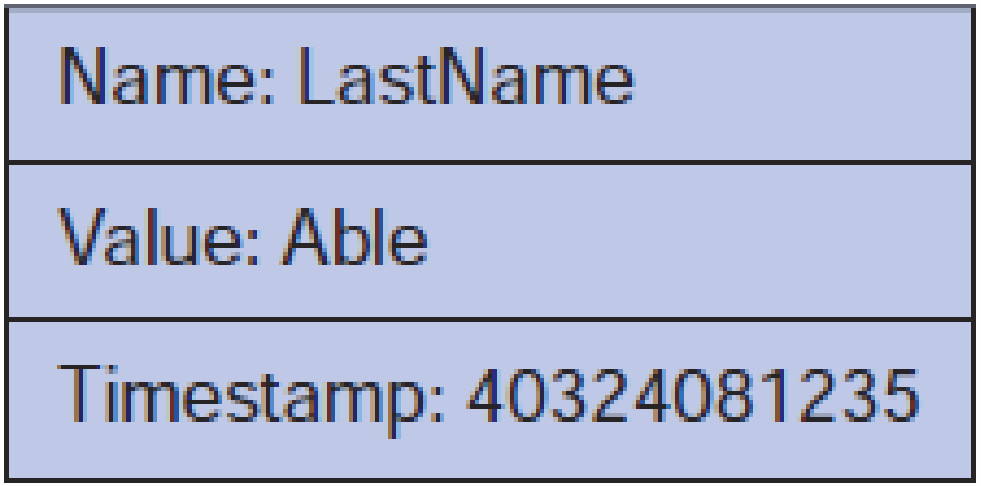
NoSQL Databases

Database	Companies Using	Website
CouchDB	Grubhub	couchdb.apache.org
Redis	Twitter, Craigslist	redis.io
HBase	Facebook, LinkedIn	hbase.apache.org
RocksDB	Facebook, LinkedIn	rocksdb.org
Cassandra	Netflix	Cassandra.apache.org
DynamoDB	Apple, Netflix	Aws.amazon.com/dynamodb

The CAP Theorem (con't)

- The CAP theorem basically states that *all three of these properties cannot be guaranteed at the same time*—at least one of the properties will be at least somewhat compromised
- An early, but now acknowledged as incorrect, statement was that "2 of the 3" was the best possible outcome
- Today -> "eventual consistency"

Column Family Databases:
A Column



Column Family Databases:
A Super Column

Super Column Name:	CustomerName	
Super Column Values:	Name: FirstName	Name: LastName
	Value: Ralph	Value: Able
	Timestamp: 40324081235	Timestamp: 40324081235

Column Family Databases:
A Column Family

Column Family Name:	Customer			
RowKey001	Name: FirstName	Name: LastName		
	Value: Ralph	Value: Able		
	Timestamp: 40324081235	Timestamp: 40324081235		
RowKey002	Name: FirstName	Name: LastName	Name: Phone	Name: City
	Value: Nancy	Value: Jacobs	Value: 817-871-8123	Value: Fort Worth
	Timestamp: 40335091055	Timestamp: 40335091055	Timestamp: 40335091055	Timestamp: 40335091055
RowKey003	Name: LastName	Name: EmailAddress		
	Value: Baker	Value: Susan.Baker@elswhere.com		
	Timestamp: 40340103518	Timestamp: 40340103518		

Column Family Databases:
A Super Column Family

Super Column Family Name:	Customer			
Rowkey001	Customer Name		CustomerPhone	
	Name: FirstName	Name: LastName	Name: AreaCode	Name: PhoneNumber
	Value: Ralph	Value: Able	Value: 210	Value: 281–7987
	Timestamp: 40324081235	Timestamp: 40324081235	Timestamp: 40335091055	Timestamp: 40335091055
Rowkey002	Customer Name		CustomerPhone	
	Name: FirstName	Name: LastName	Name: AreaCode	Name: PhoneNumber
	Value: Nancy	Value: Jacobs	Value: 817	Value: 871–8123
	Timestamp: 40335091055	Timestamp: 40335091055	Timestamp: 40335091055	Timestamp: 40335091055
Rowkey003	Customer Name		CustomerPhone	
	Name: FirstName	Name: LastName	Name: AreaCode	Name: PhoneNumber
	Value: Susan	Value: Baker	Value: 210	Value: 281–7876
	Timestamp: 40340103518	Timestamp: 40340103518	Timestamp: 40340103518	Timestamp: 40340103518

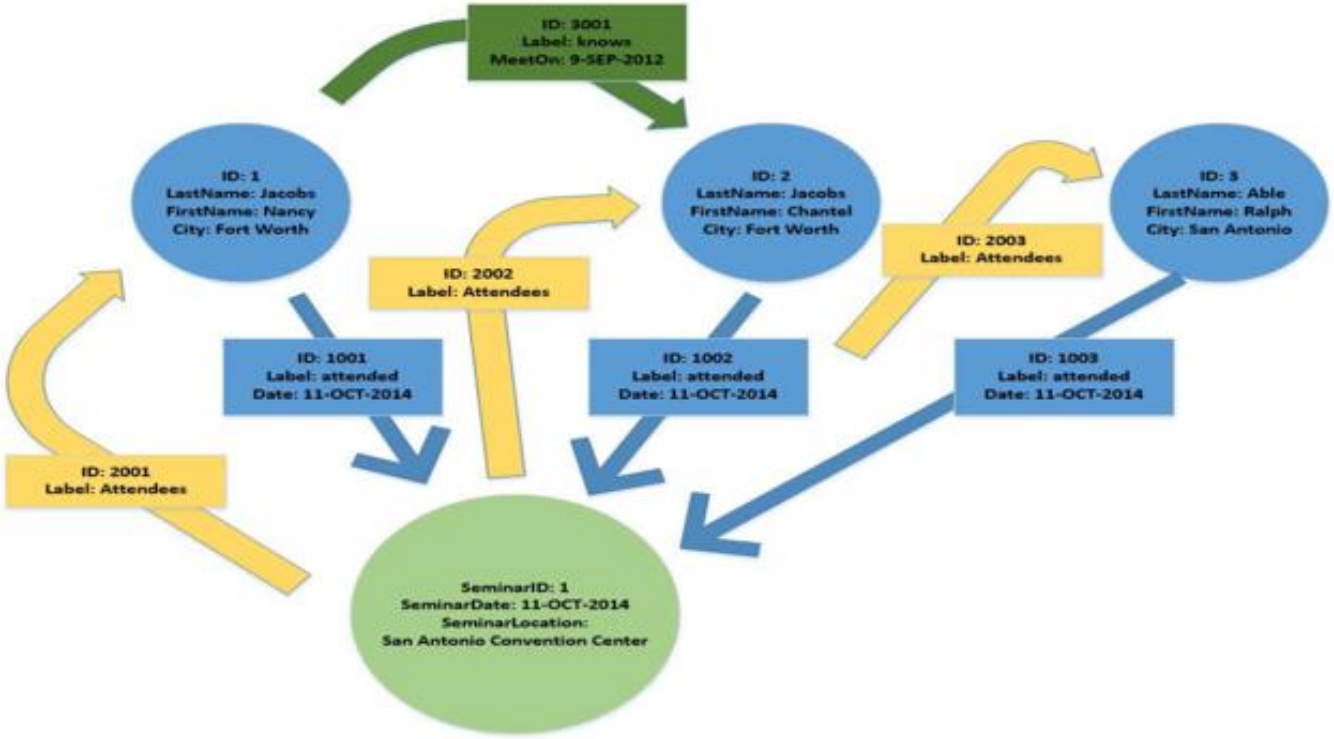
Graph Databases

- Based on mathematical graph theory, graph databases are composed of three elements:
  - Nodes
  - Properties
  - Edges
- Example:
  - **Neo4j**.

Graph Databases (con't)

- **Nodes**
  - Equivalent to entities in E-R data modeling and tables in database design.
  - They represent the things that we want to keep track of or about which we want to store data.
- **Properties**
  - Equivalent to attributes in E-R data modeling and columns (or fields) in database design.
  - They represent the data items that we want to store for each node.
- **Edges**
  - Similar to, but not identical to, the relationships in E-R data models and database designs.
  - They are similar because they connect nodes as relationships connect entities, but they are different because they also store data.
  - Edges can be one-way or two-way.

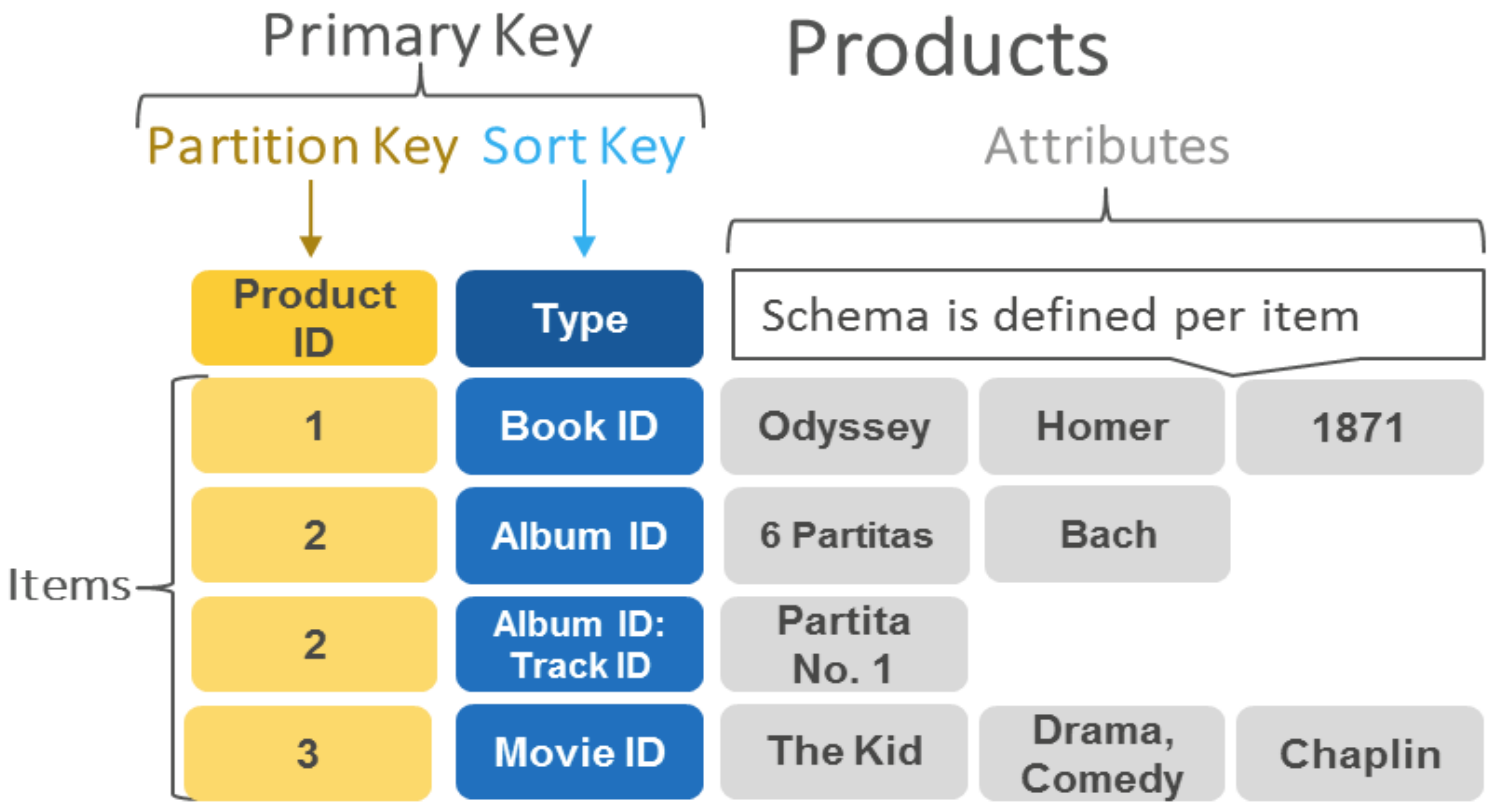
Graph Databases (con't)



Key Value Databases

- Key value databases uses a simple key-value pairing
- Each key (similar to a relational DBMS) appears only once in each database
- However, the “value” may be of different types or sizes for each key type
- Examples:
  - **Dynamo** - Amazon.com

Key Value Databases (con't)



Document Databases

- Data storage in a document-oriented format
  - XML (Extensible Markup Language)
  - JSON (Java Script Object Notation)
- Examples:
  - **Couchbase Server –** uses JSON

JSON

- **JavaScript Object Notation**, is an open standard format that uses human-readable text to transmit data objects consisting of attribute–value pairs
- It is used primarily to transmit data between a server and web application, as an alternative to XML

```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 25,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    }
  ],
  "children": [],
  "spouse": null
}
```

NoSql Implementation -> Hadoop and MapReduce

Scalability (petabytes of data, thousands of machines)

Flexibility in accepting all data formats (no schema)

Efficient and simple fault-tolerant mechanism

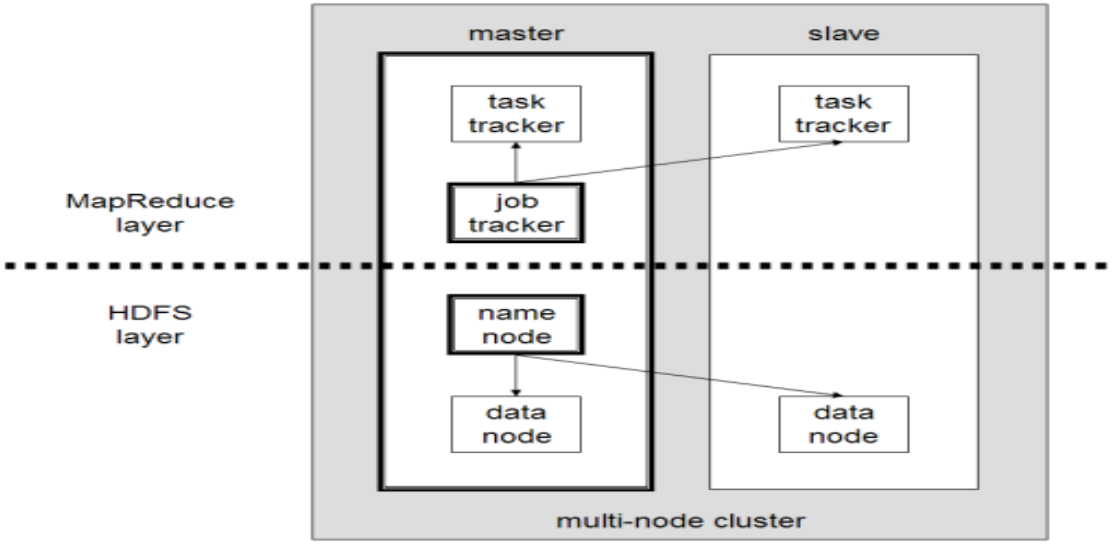
Commodity inexpensive hardware

What is Hadoop

- Hadoop is a software framework for *distributed processing* of *large datasets* across *large clusters* of computers
  - ***Large datasets*** → Terabytes or petabytes of data
  - ***Large clusters*** → hundreds or thousands of nodes
- Hadoop is open-source implementation of Google's ***MapReduce***
- Hadoop is based on a simple data model, *any data will fit and most commonly used is one of the NoSQL models*

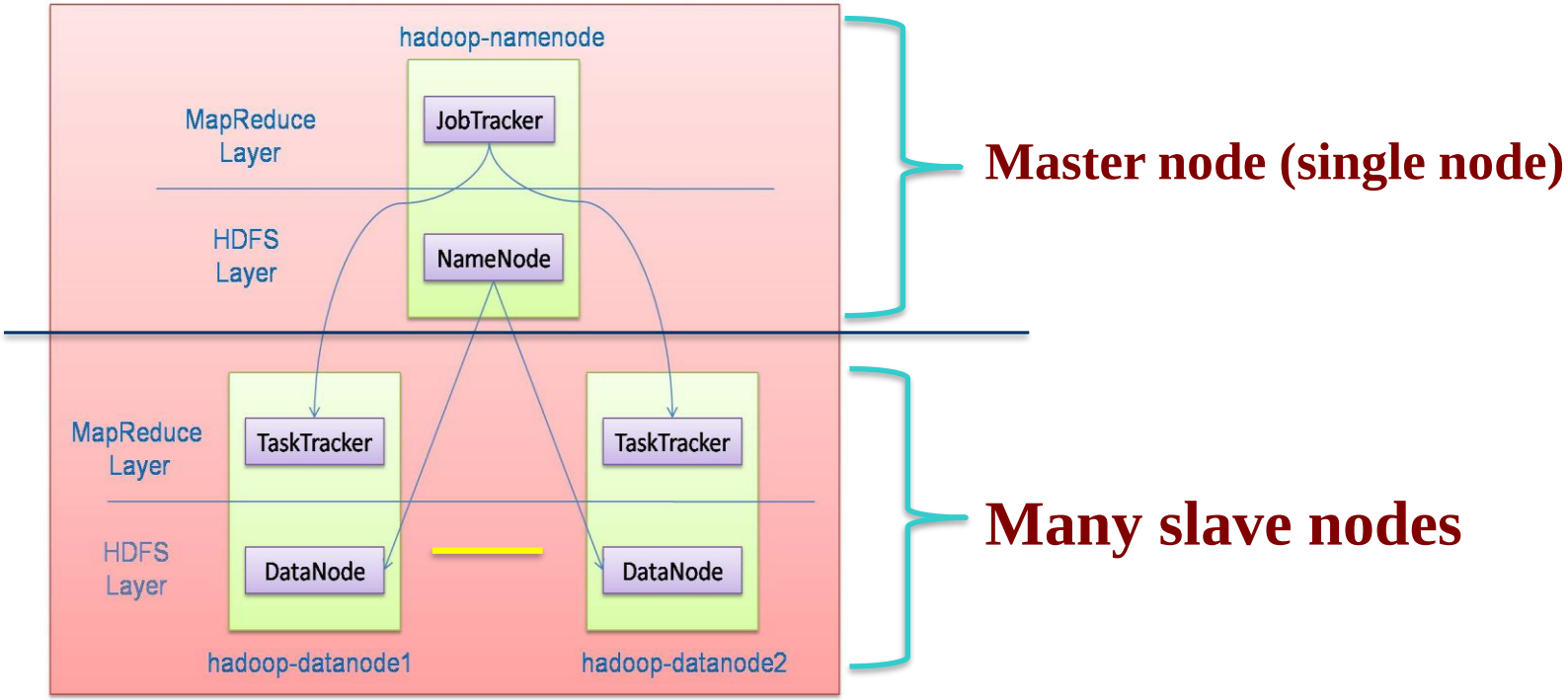
What is Hadoop (Con't)

- **Hadoop framework consists on two main layers**
  - Distributed file system (HDFS)
  - Execution engine (MapReduce)



Hadoop Master/Slave Architecture

- Hadoop is designed as a *master-slave* *shared-nothing* architecture



Design Principles of Hadoop

- Need to process big data
- Need to parallelize computation across thousands of nodes
- **Commodity hardware**
  - Large number of low-end cheap machines working in parallel to solve a computing problem
- This is in contrast to **Parallel DBs**
  - Small number of high-end expensive machines

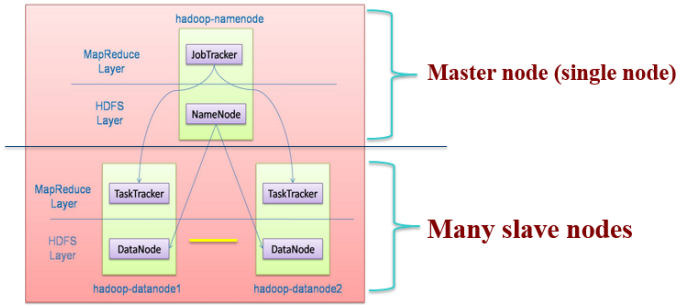
Design Principles of Hadoop

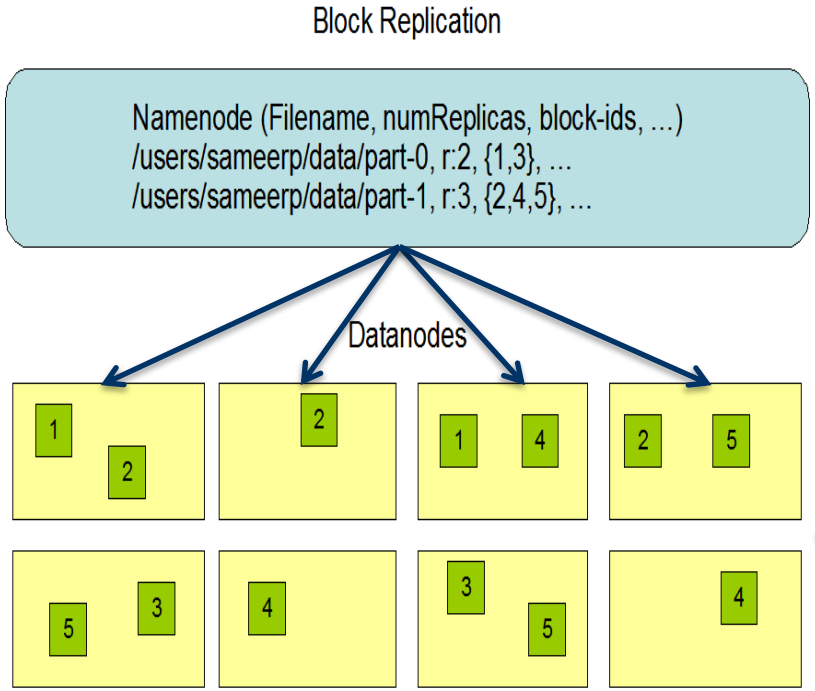
- **Automatic parallelization & distribution**
  - Hidden from the end-user
- **Fault tolerance and automatic recovery**
  - Nodes/tasks will fail and will recover automatically
- **Clean and simple programming abstraction**
  - Users only provide two functions “map” and “reduce”

Who Uses MapReduce/Hadoop

- Google: Inventors of MapReduce computing paradigm
- Yahoo: Developing Hadoop open-source of MapReduce
- IBM, Microsoft, Oracle
- Facebook, Amazon, AOL, NetFlex
- Many others + universities and research labs

Hadoop Distributed File System (HDFS)





Centralized namenode

- Maintains metadata info about files

File *F* | 1 | 2 | 3 | 4 | 5 |

Blocks (64 MB)

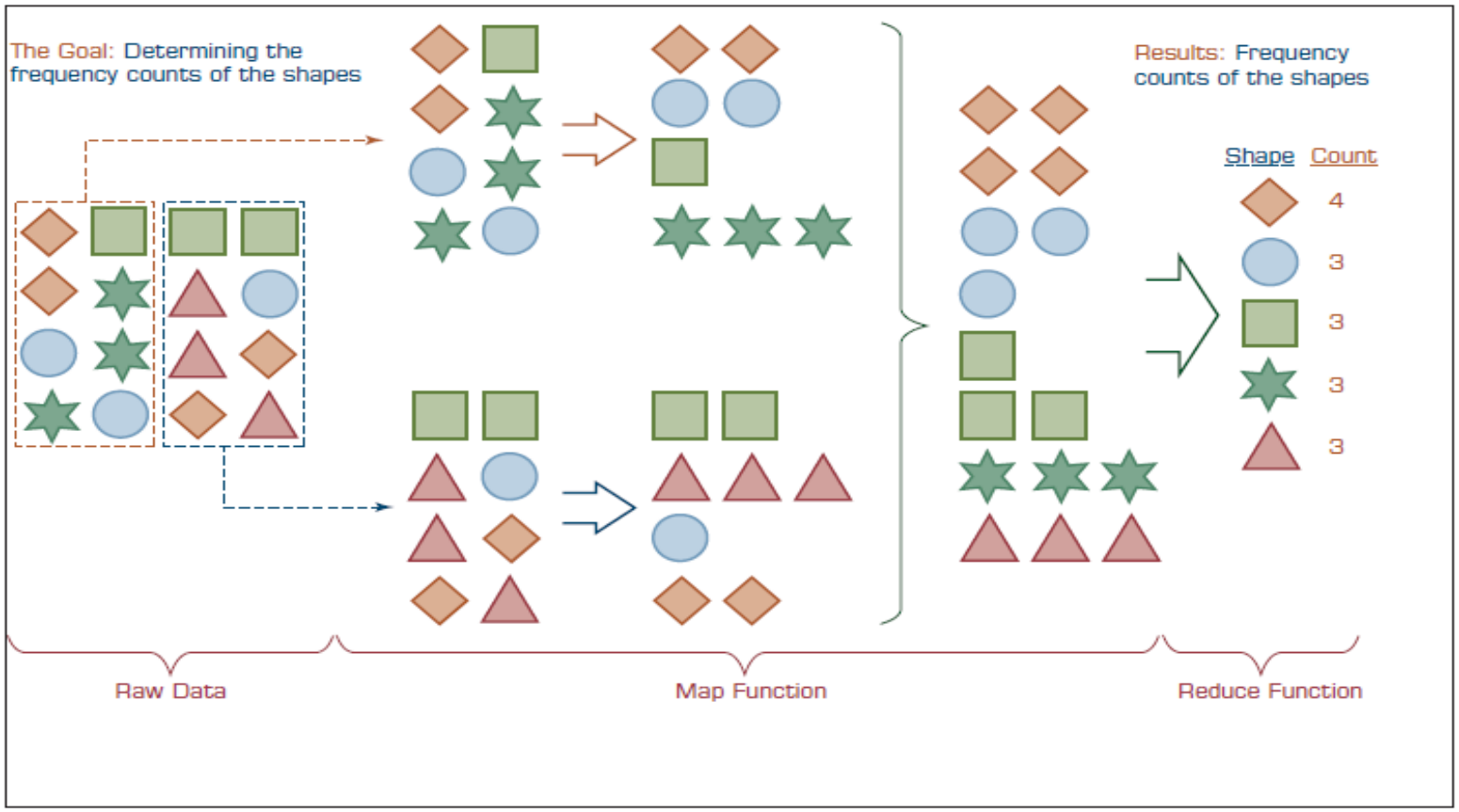
Many datanode (1000s)

- Store the actual data
- Files are divided into blocks
- Each block is replicated *N* times (Default = 3)

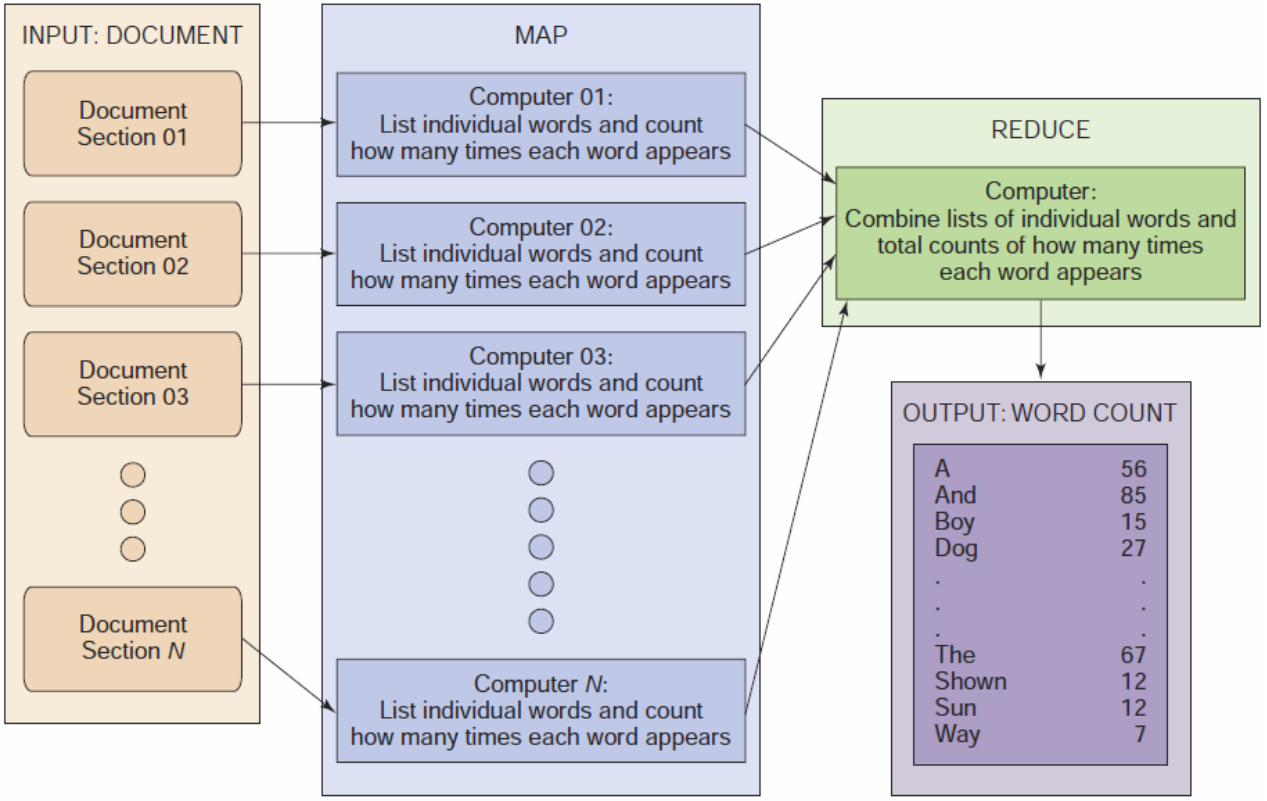
Main Properties of HDFS

- ***Large:*** A HDFS instance may consist of thousands of server machines, each storing part of the file system's data
- ***Replication:*** Each data block is replicated many times (default is 3)
- ***Failure:*** Failure is the norm rather than exception
- ***Fault Tolerance:*** Detection of faults and quick, automatic recovery from them is a core architectural goal of HDFS
  - Namenode is consistently checking Datanodes

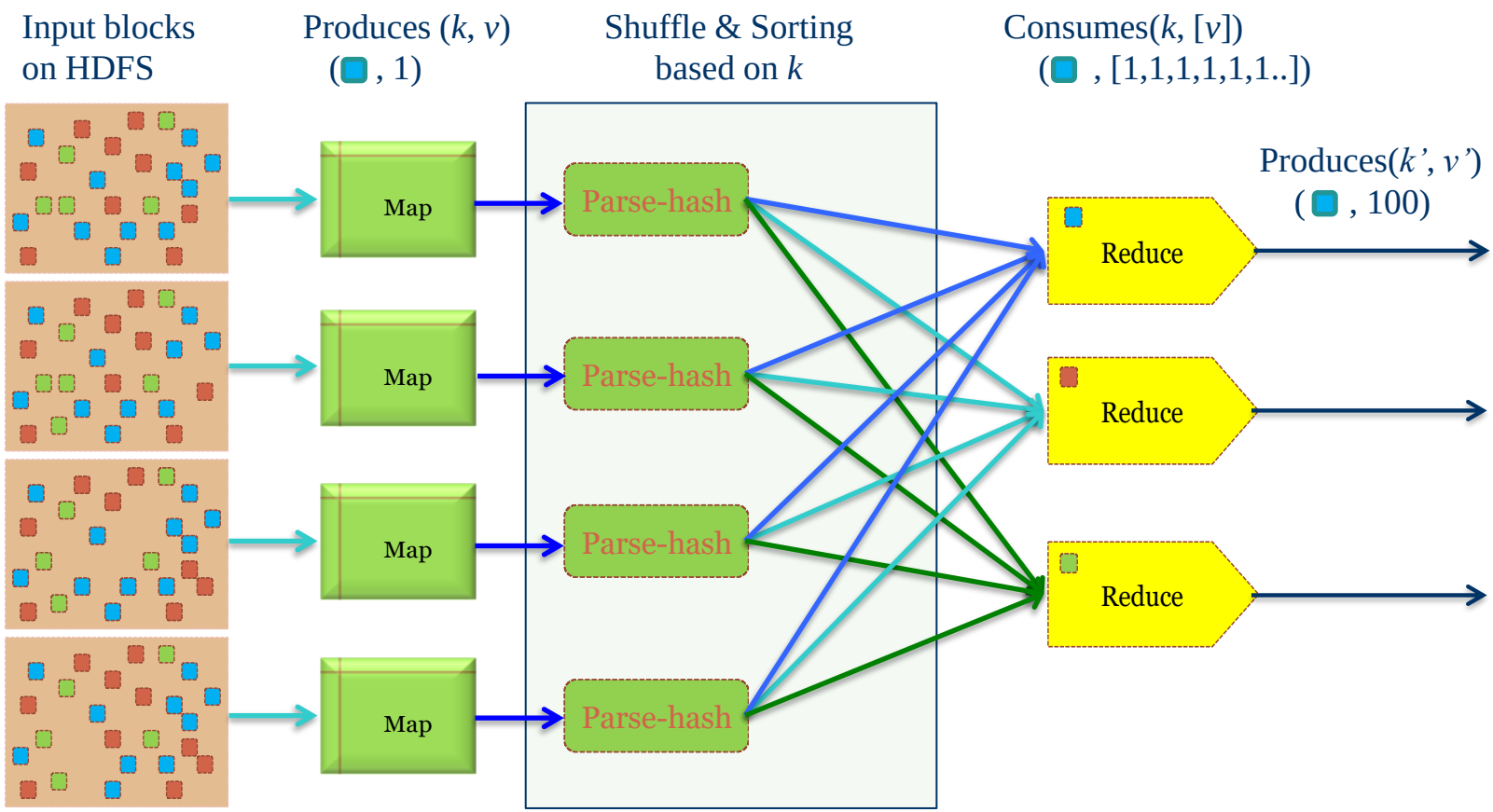
Map/Reduce Process



The MapReduce Process (con't)



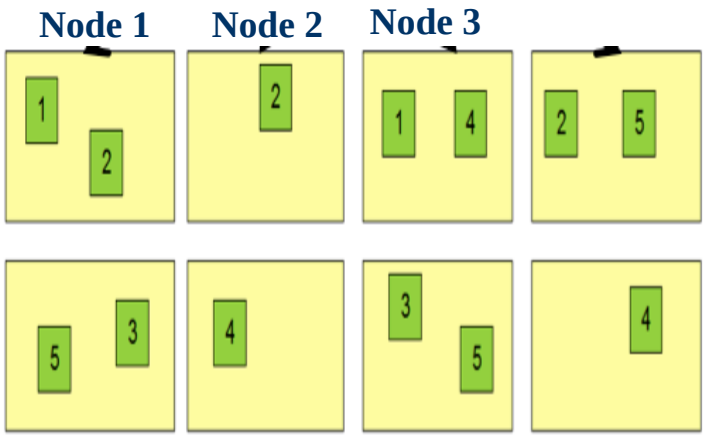
Map-Reduce Execution Engine (Example: Color Count)



Users only provide the "Map" and "Reduce" functions

Properties of MapReduce Engine

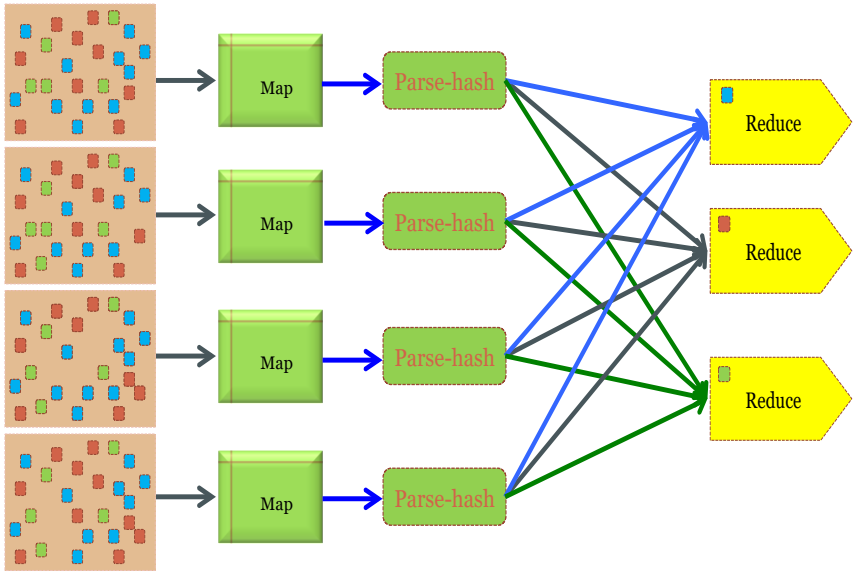
- **Job Tracker is the master node (runs with the namenode)**
  - Receives the user's job
  - Decides on how many tasks will run (number of mappers)
  - Decides on where to run each mapper (concept of locality)



- This file has 5 Blocks → run 5 map tasks
- Where to run a task needing block "1"
  - *Try to run it on Node 1 or Node 3*

Properties of MapReduce Engine (Con't)

- **Task Tracker is the slave node (runs on each datanode)**
  - Receives the task from Job Tracker
  - Runs the task until completion (either map or reduce task)
  - Always in communication with the Job Tracker reporting progress

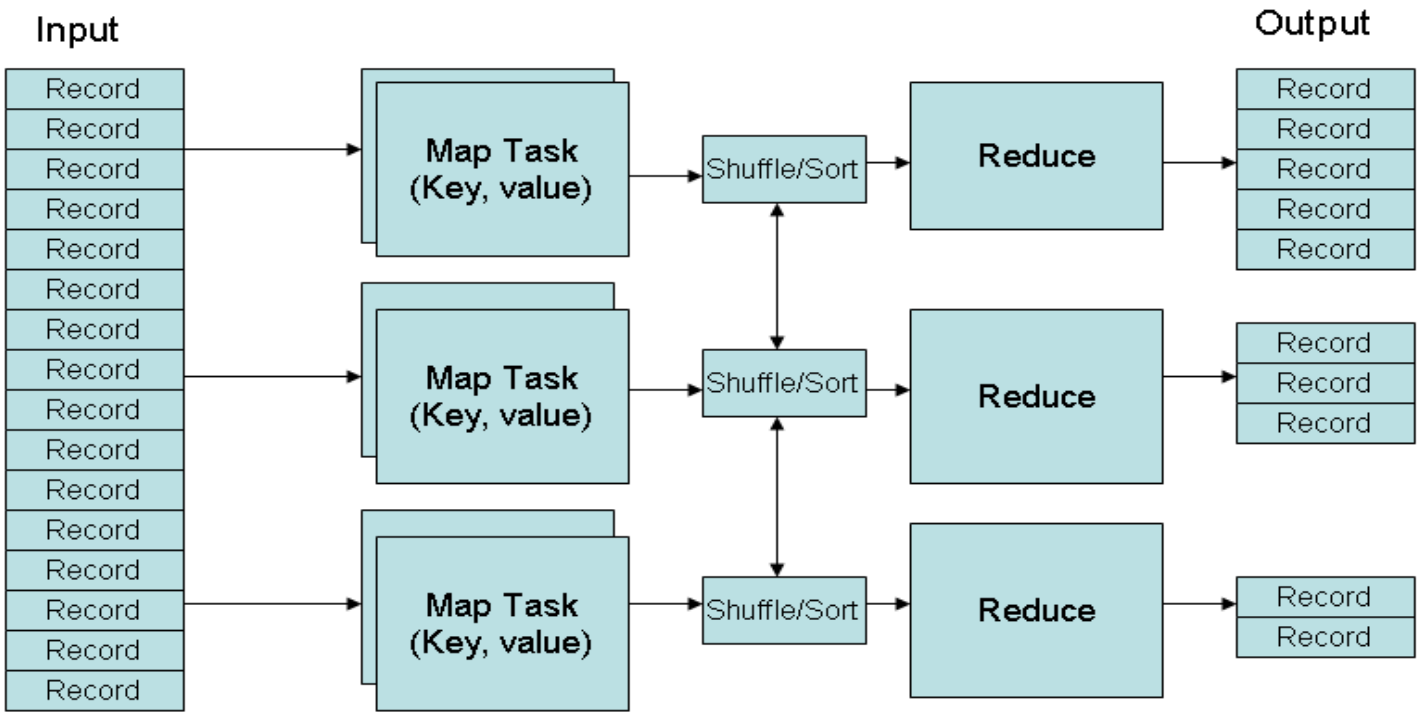


In this example, 1 map-reduce job consists of 4 map tasks and 3 reduce tasks

Key-Value Pairs

- Mappers and Reducers are users' code (provided functions)
- Just need to obey the Key-Value pairs interface
- **Mappers:**
  - Consume <key, value> pairs
  - Produce <key, value> pairs
- **Reducers:**
  - Consume <key, <list of values>>
  - Produce <key, value>
- **Shuffling and Sorting:**
  - Hidden phase between mappers and reducers
  - Groups all similar keys from all mappers, sorts and passes them to a certain reducer in the form of <key, <list of values>>

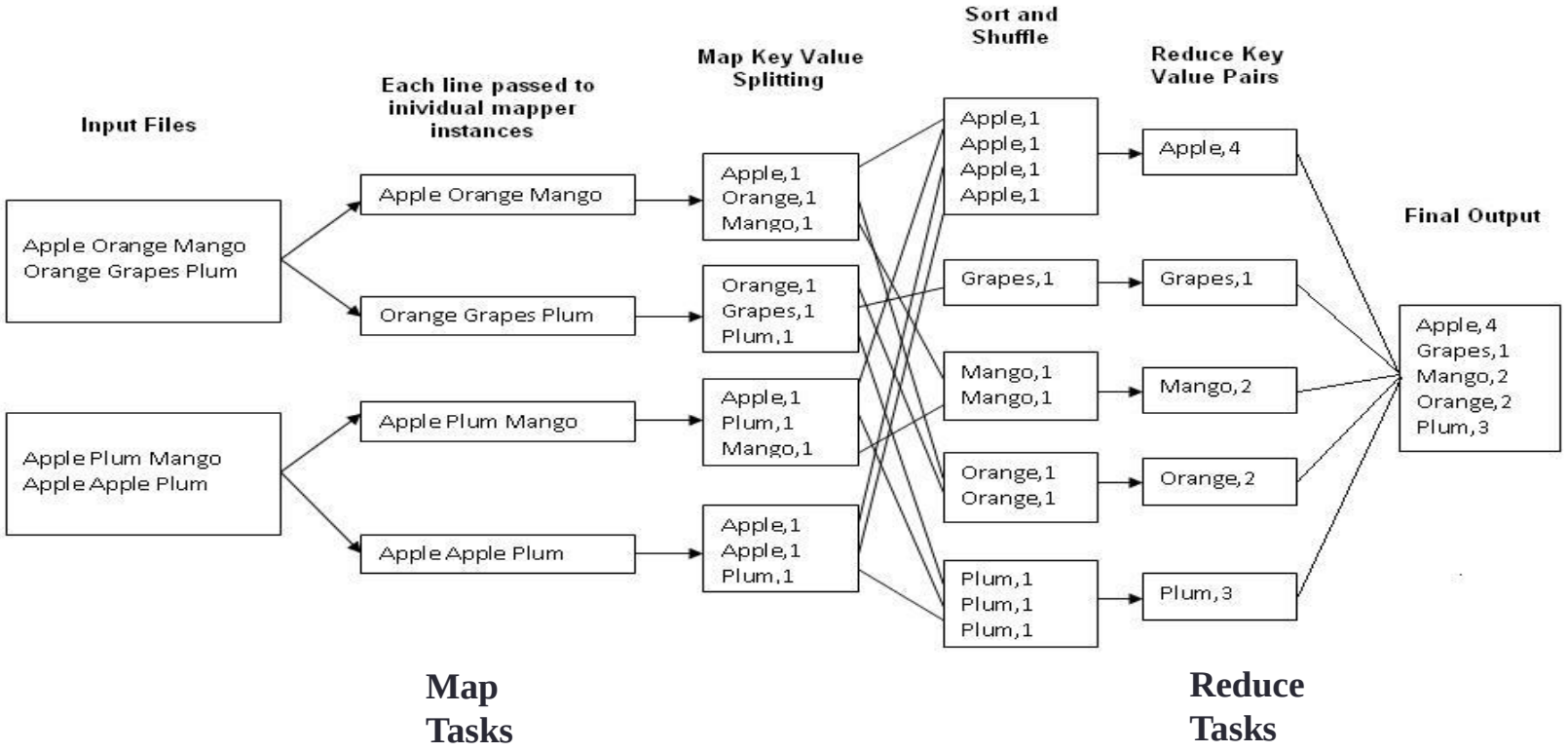
MapReduce Phases



Deciding on what will be the key and what will be the value is the developer's responsibility !

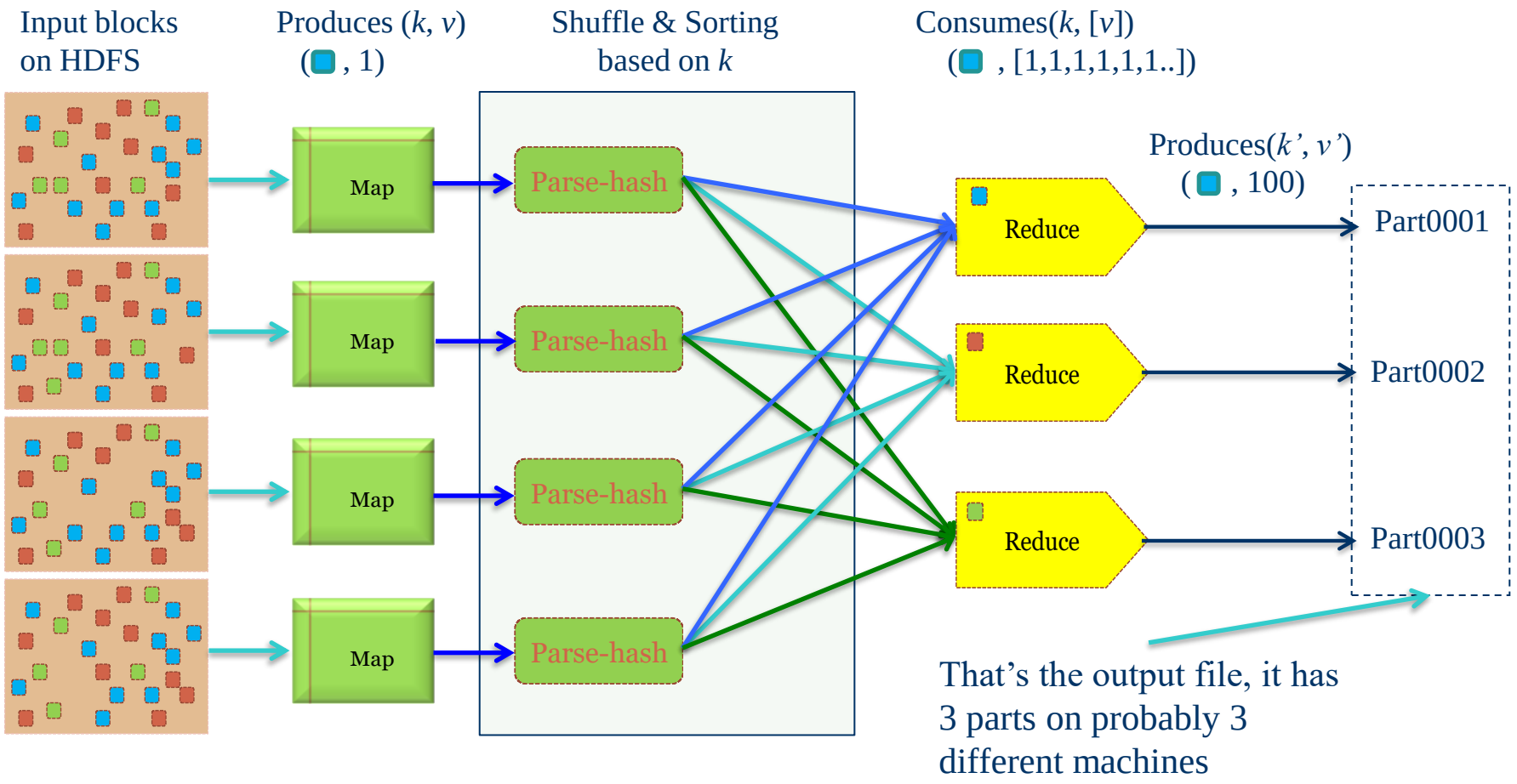
Example 1: Word Count

- **Job: Count the occurrences of each word in a data set**



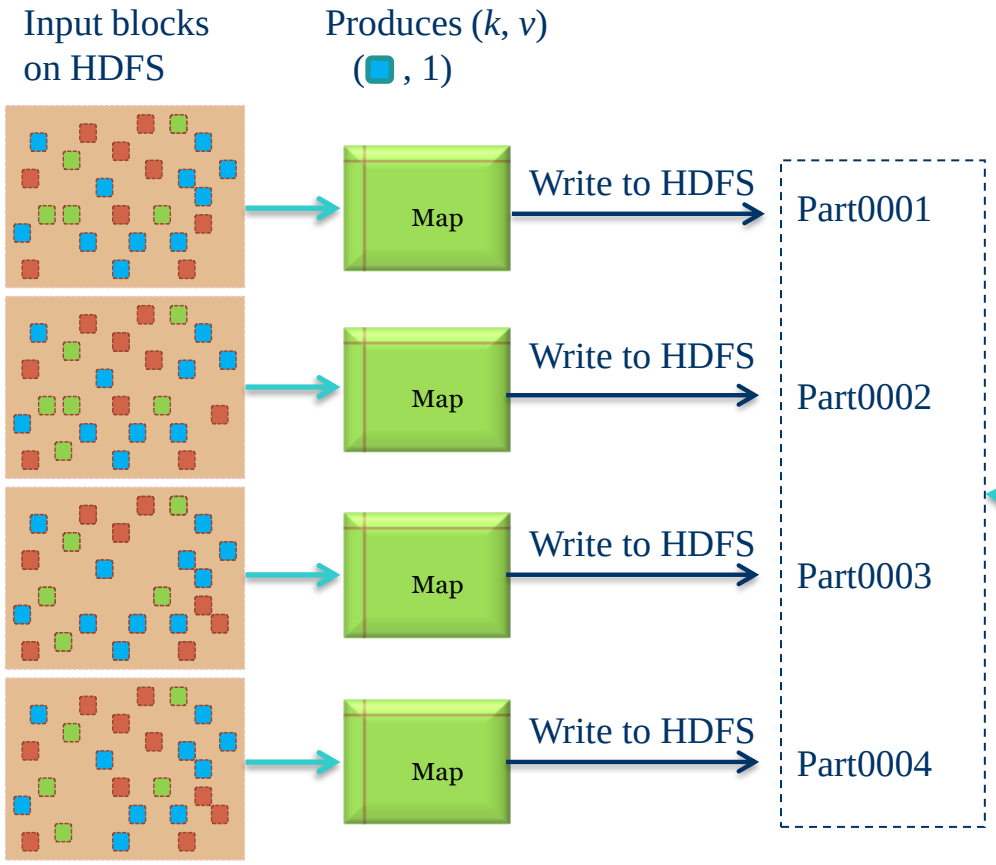
Example 2: Color Count

Job: Count the number of each color in a data set



Example 3: Color Filter

Job: Select only the blue and the green colors



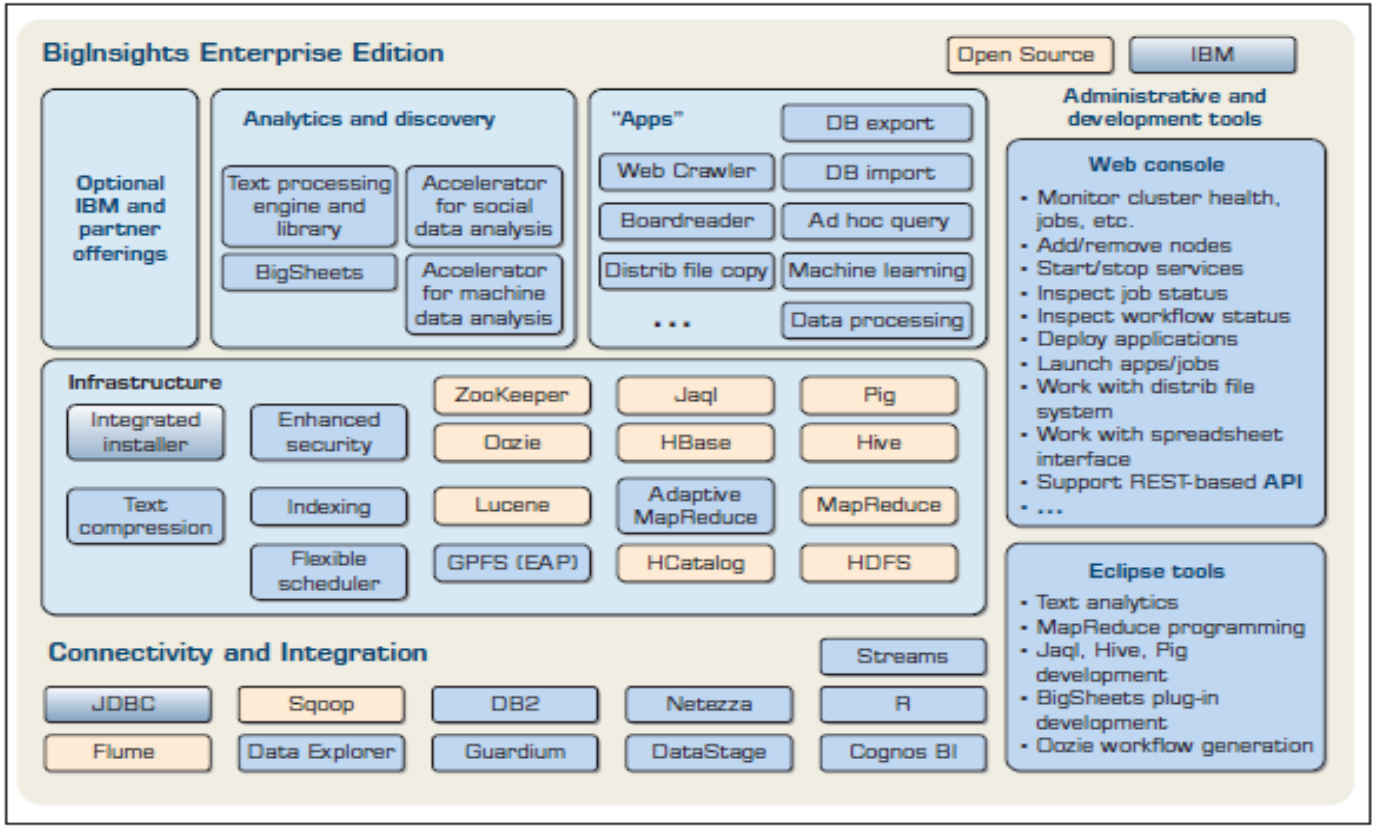
- Each map task will select only the blue or green colors
- No need for reduce phase

That's the output file, it has 4 parts on probably 4 different machines

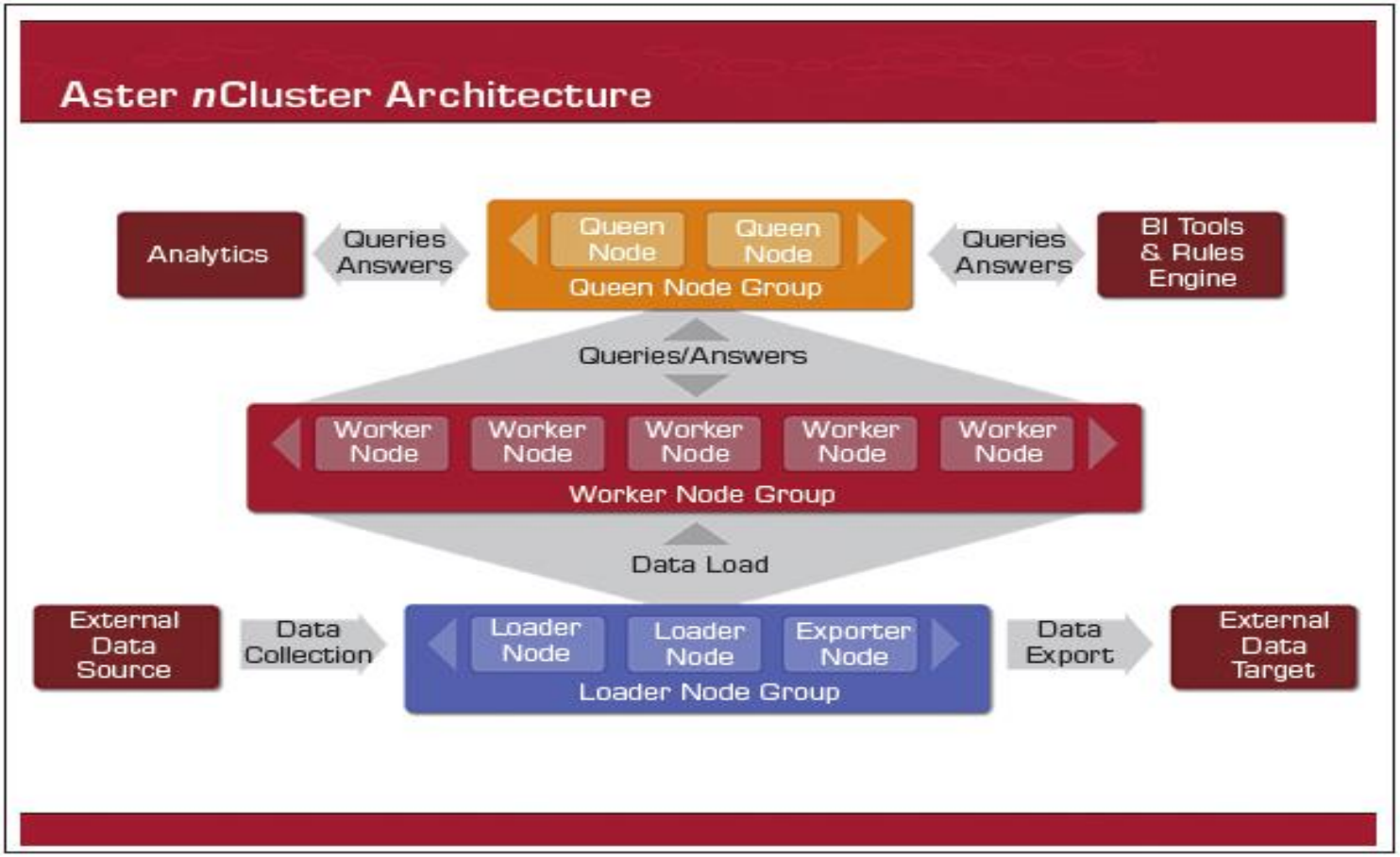
Hadoop vs SQL Distributed Database

	Distributed Databases	Hadoop
Computing Model	- Notion of transactions - Transaction is the unit of work - ACID properties, Concurrency control	- Notion of jobs - Job is the unit of work - No concurrency control
Data Model	- Structured data with known schema - Read/Write mode	- Any data will fit in any format - (un)(semi)structured - ReadOnly mode
Cost Model	- Expensive servers	- Cheap commodity machines
Fault Tolerance	- Failures are rare - Recovery mechanisms	- Failures are common over thousands of machines - Simple yet efficient fault tolerance
Key Characteristics	- Efficiency, optimizations, fine-tuning	- Scalability, flexibility, fault tolerance

IBM InfoSphere BigInsights



Teradata Aster



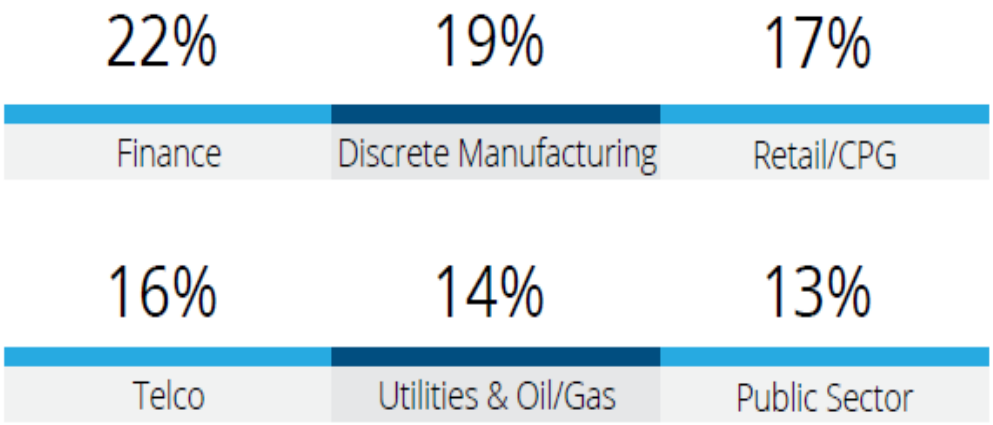
IDC 2015 Study

IDC recently conducted a survey sponsored by SAP and Intel to discover how organizations are successfully using Big Data and analytics. The survey evaluated the relevance of Big Data and analytics to business transformation and was used to identify best practices from the most mature organizations.

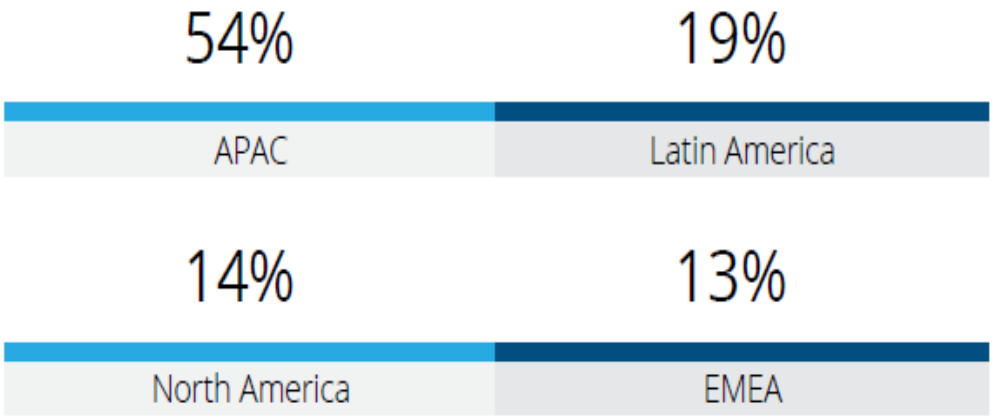
1,810 interviews were conducted in three phases between July 2014 and February 2015. The survey took place in Brazil, China, France, Germany, Italy, Kenya, Kuwait, Oman, Mexico, the Netherlands, Nigeria, the Nordics, Portugal, Qatar, Saudi Arabia, Spain, South Africa, UAE, the U.K., and the U.S.

IDC evaluated the results using IDC's Big Data Maturity model, a methodology that benchmarks organizations' competence across five dimensions: people, process, technology, data, and intent. Based on this model, IDC looked at the most mature organizations and identified what benefits they achieved from Big Data and analytics. We called these organizations the "Big Data Innovators" and looked at what they do differently to the other respondents. These differences become useful lessons around increasing maturity in Big Data and analytics and driving further success.

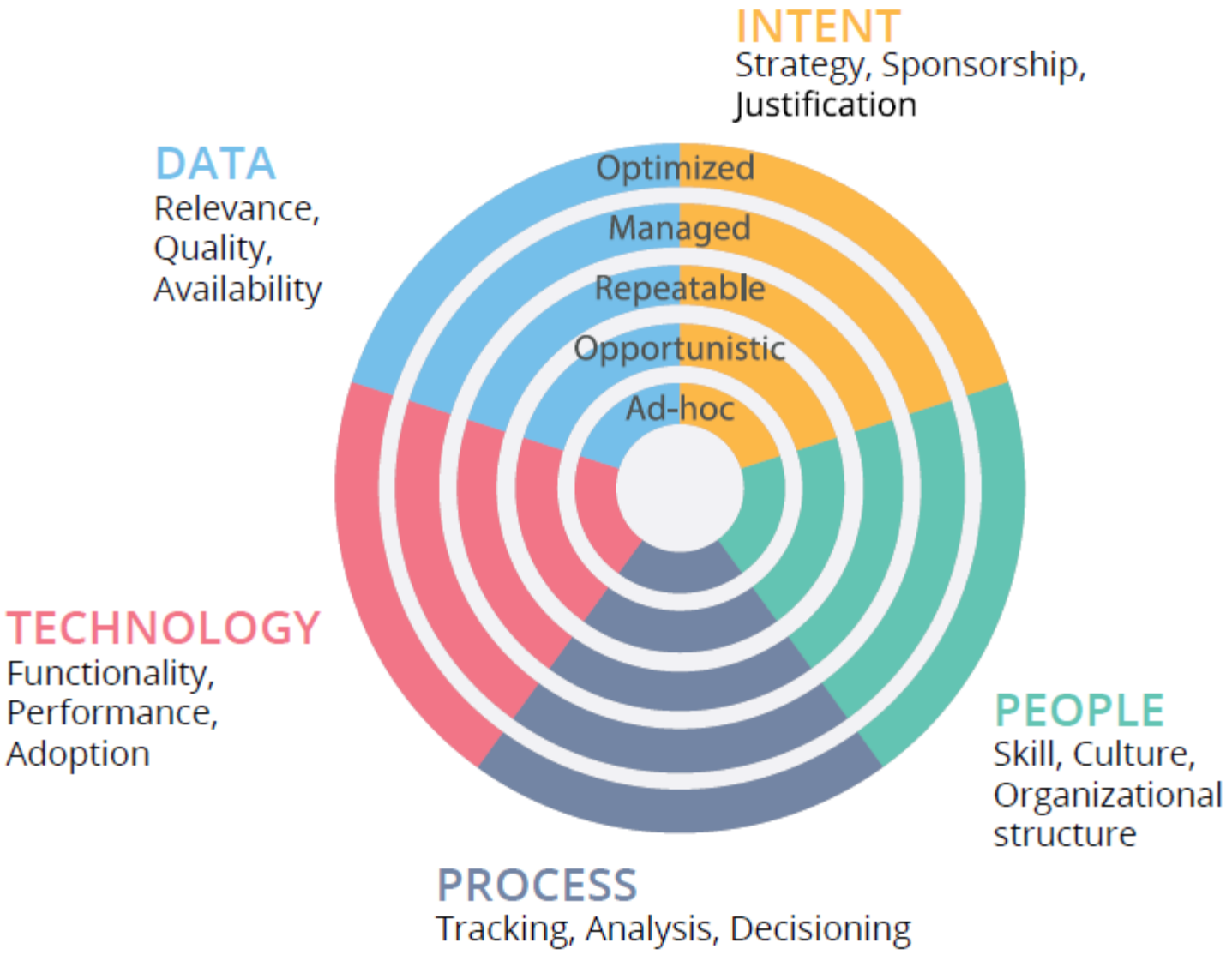
Industry

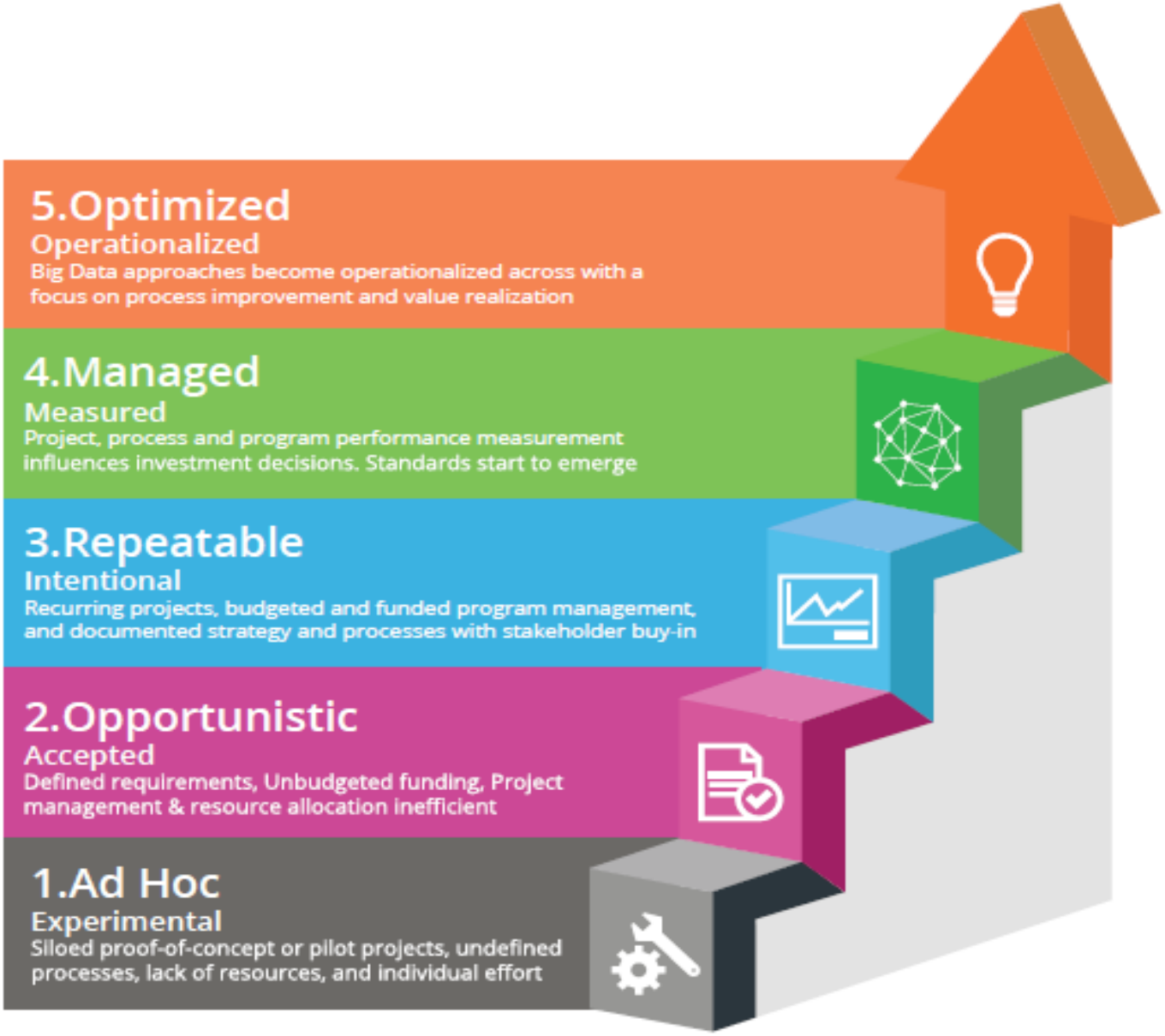


Region



Big Data Maturity Model

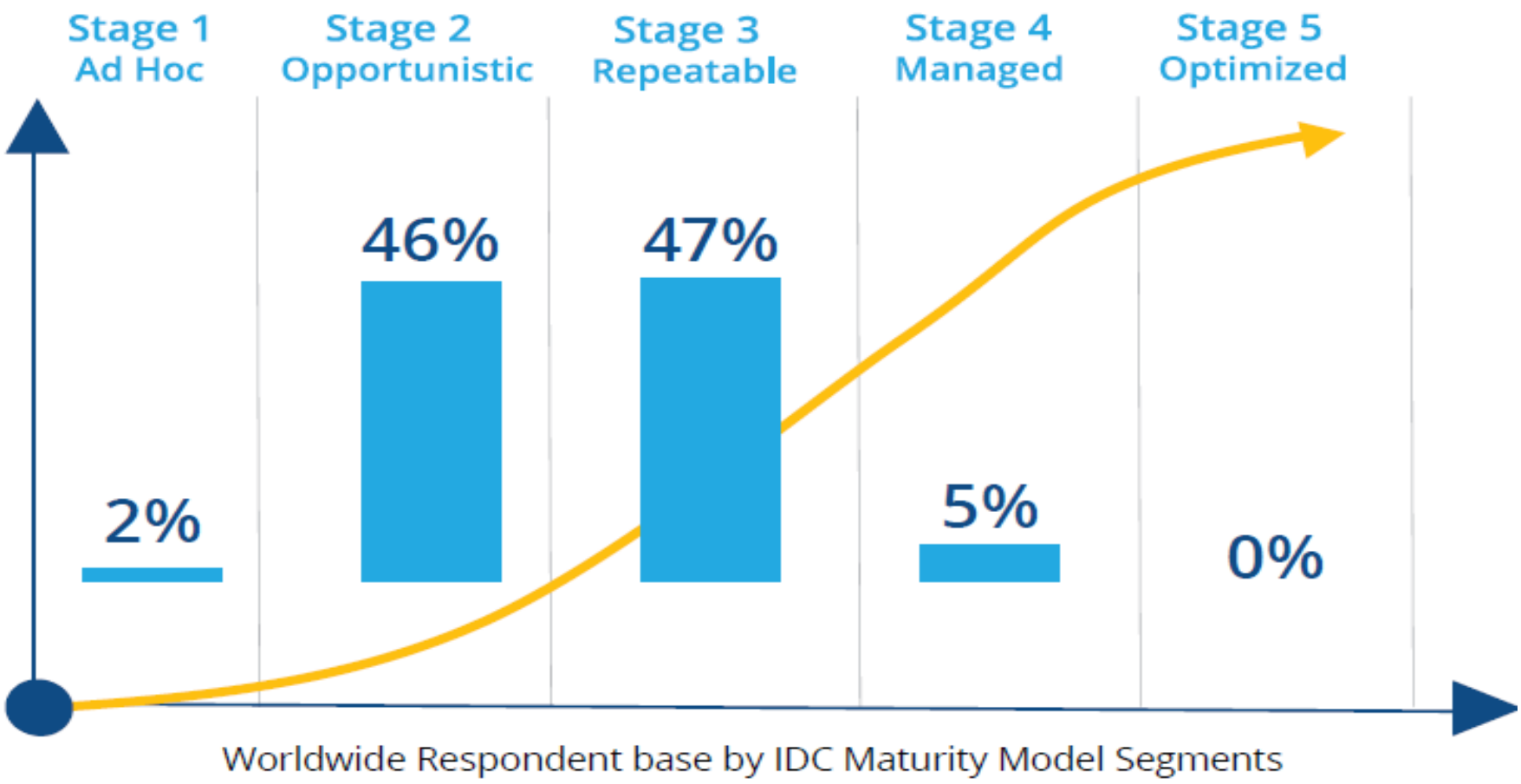




Source: IDC, Big Data MaturityScape, 2014

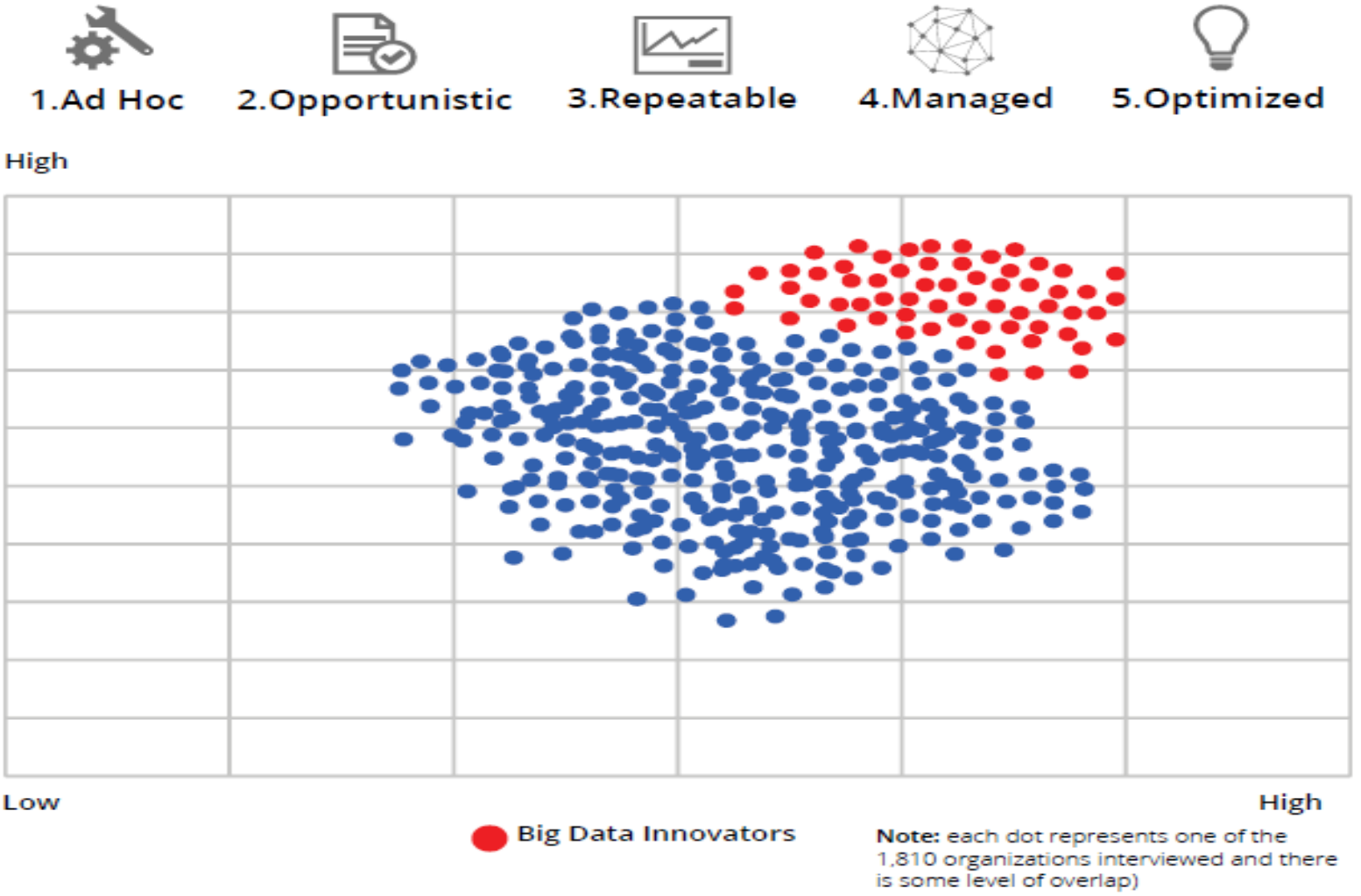
Big Data Maturity Worldwide

Worldwide Respondent base by IDC Maturity Model Segments

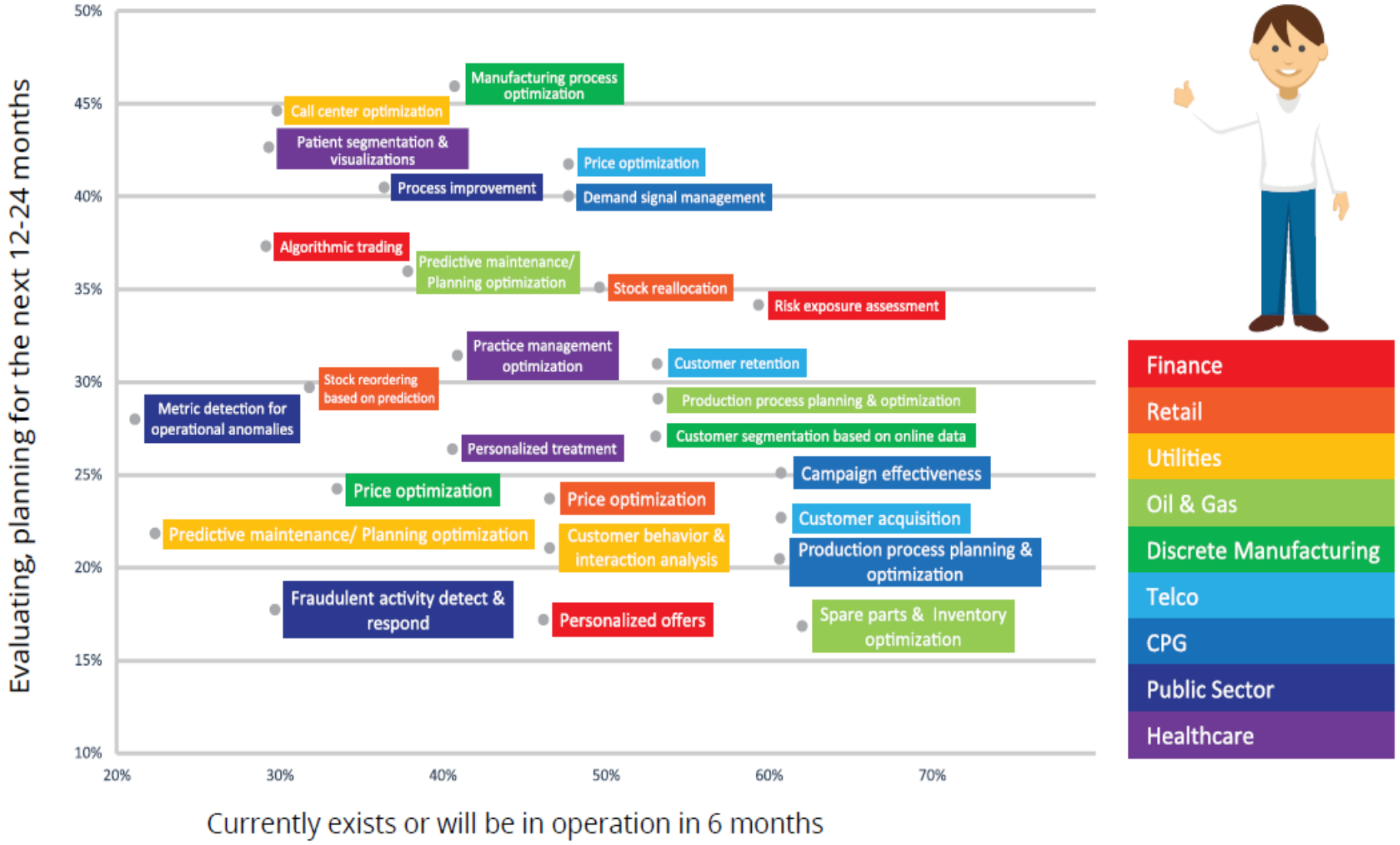


Source: IDC Big Data End-User Survey for SAP and Intel 2015 , n=1810

Big Data Maturity



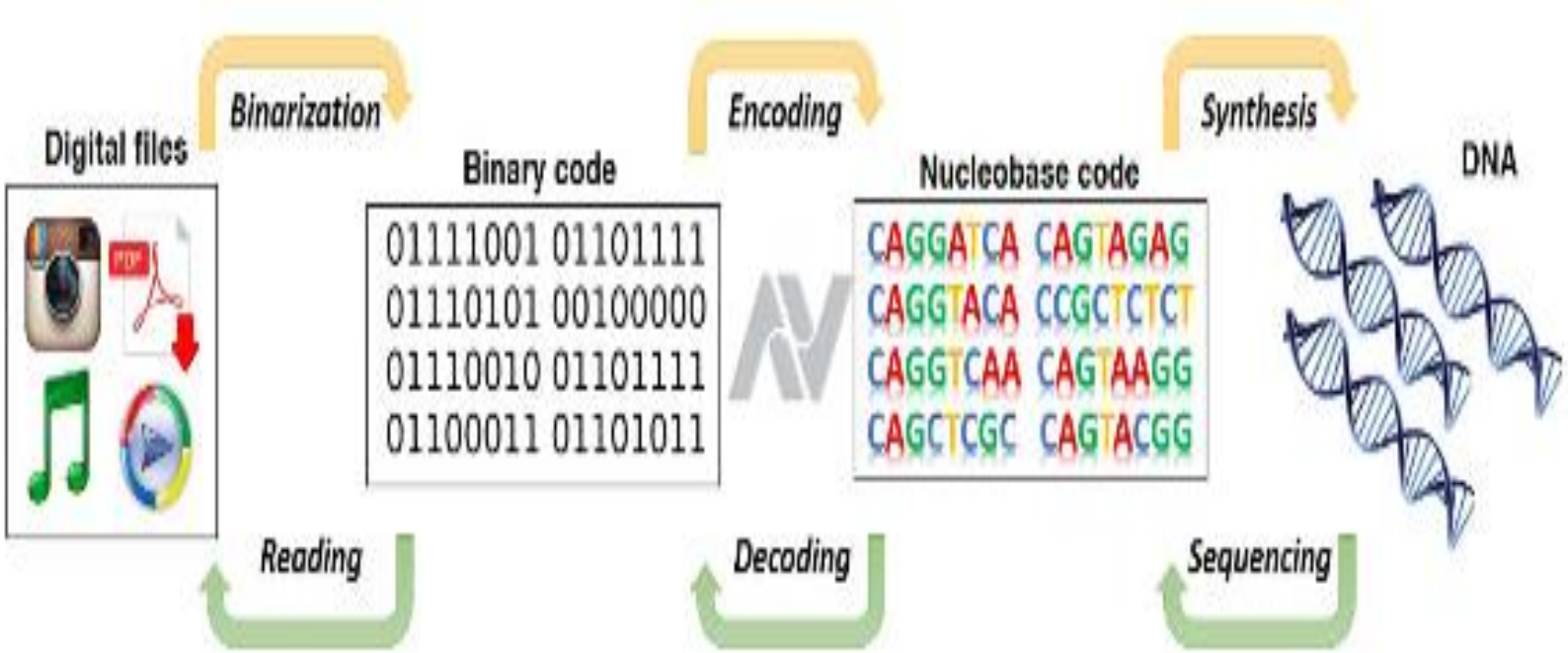
Key Big Data Use Cases



DNA Data Storage

- Scientists have long touted DNA's potential as an ideal storage medium; it's dense, easy to replicate, and stable over millennia
- In the past few years, researchers have encoded all kinds of things in those strings of As, Ts, Cs, and Gs
- But in order to replace existing silicon-chip or magnetic-tape storage technologies, DNA will have to get a lot cheaper to predictably read, write, and package
- Catalog, an MIT DNA-storage spinoff emerging out of stealth on Tuesday, has come a long way since encoding their first poetic kilobyte by hand a year and a half ago
- Now they're building a machine that will write a terabyte of data a day, using 500 trillion molecules of DNA

DNA Data Storage (con't)



DNA Data Storage (con’t)

- DNA storage could be the answer to a uniquely 21st-century problem: information overload
- Five years ago humans had produced 4.4 zettabytes of data; that's set to explode to 160 zettabytes (each year!) by 2025
- Current infrastructure can handle only a fraction of the coming data deluge, which is expected to consume all the world's microchip-grade silicon by 2040
- Most digital archives—from music to satellite images to research files—are currently saved on magnetic tape. Tape is cheap. But it takes up space. And it has to be replaced roughly every 10 years
- Today’s technology is already close to the physical limits of scaling
- DNA has an information-storage density several orders of magnitude higher than any other known storage technology
- How dense exactly? Imagine formatting every movie ever made into DNA; it would be smaller than the size of a sugar cube. And it would last for 10,000 years

DNA Data Storage (con't)

- The trouble of course, is cost. Sequencing—or reading—DNA has gotten far less expensive in the last few years; but the economics of writing DNA remain problematic if it's going to become a standard archiving technology
- Catalog thinks it can rewrite those cost curves by decoupling the process of writing DNA from the process of encoding it
- Traditional methods map the sequence of bits—0s and 1s—onto a sequence of DNA's four base pairs
- In 2016, when Microsoft set a record by storing 200 megabytes of data in nucleotide strands, the company used 13,448,372 unique pieces of DNA
- What Catalog does, instead, is cheaply generate large quantities of just a few different DNA molecules, none longer than 30 base pairs; then it uses billions of enzymatic reactions to encode information into the recombination patterns

References

- Big Data: A Revolution That Will Transform How We Live, Work, and Think by Viktor Mayer-Schönberger and Kenneth Cukier
- Big Data For Dummies by Judith S. Hurwitz, Alan Nugent , et al.
- Big Data in Practice: How 45 Successful Companies Used Big Data Analytics to Deliver Extraordinary Results by Bernard Marr, Piers Hampton, et al.

Homework

- Textbook Chapter 14
- Review Questions 1, 2, 8, 9, 10, 11