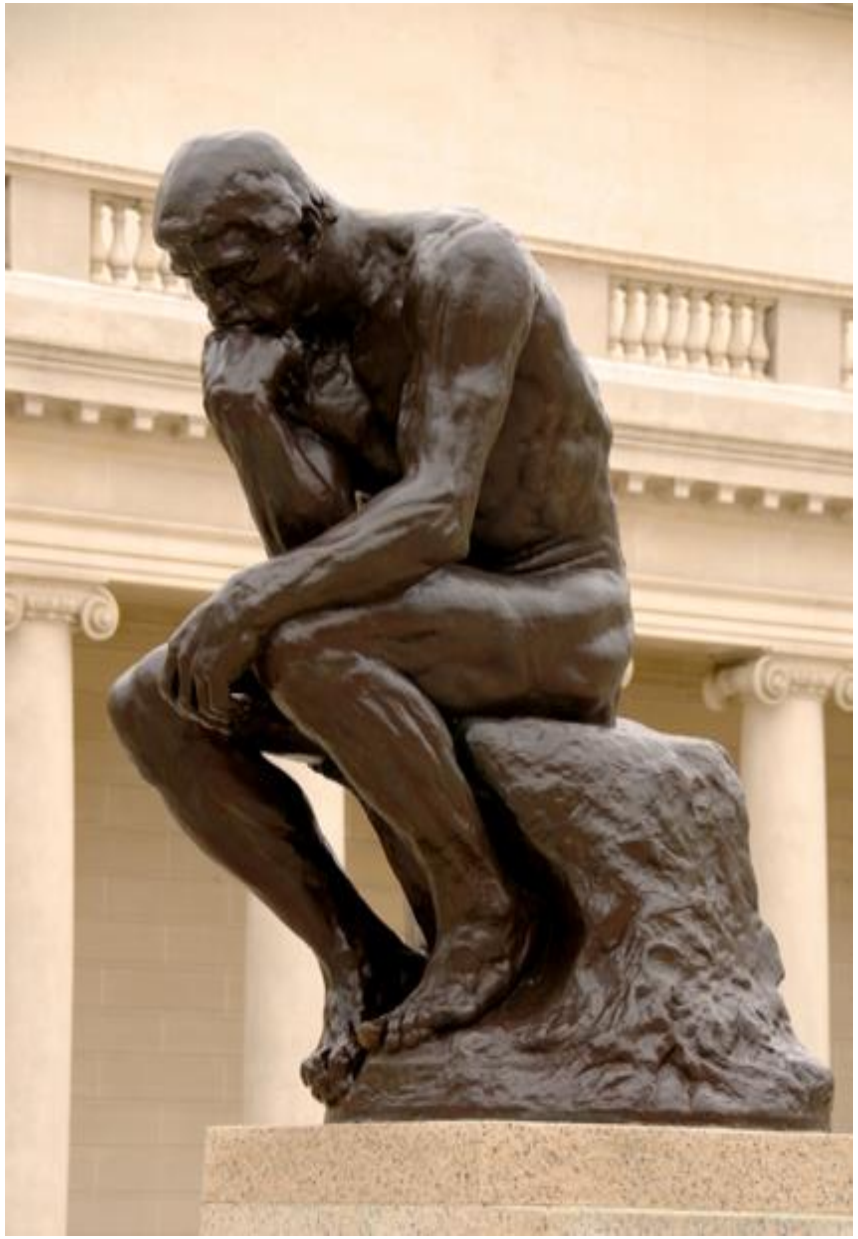
CS3121 - Introduction to Data Science

Big Data

Dr. Nisansa de Silva,
Department of Computer Science & Engineering
http://nisansads.staff.uom.lk/

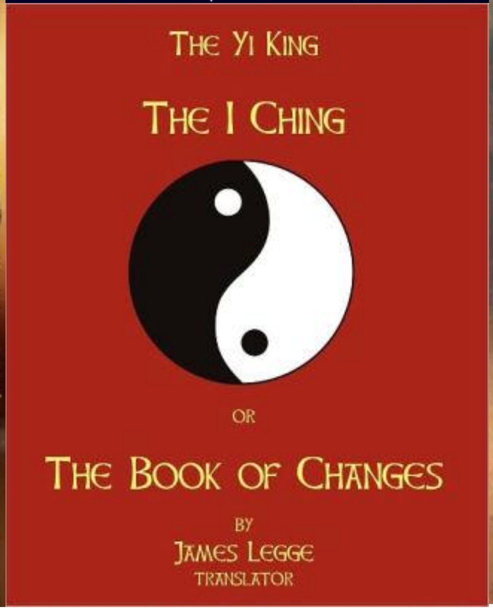
People Wanted to Through Ages

- To Know (what happened?)
- To Explain (why it happened)
- To Predict (what will happen?)



Many Cultures had ***Claims for Omniscience***

- Oracles
- Astrology
- Book of Changes
- Tarot Cards
- Crystal balls
- Others



To know, explain and predict!!

- Grand challenge of our time
- We have been trying to do this in many other means
- Now we trying to do this via science

To know, explain and predict!!

- Grand challenge of our time
- We have been trying to do this in many other means
- Now we trying to do this via science

Any sufficiently advanced technology is indistinguishable from magic.

--Arthur C. Clarke.

- We see a possibilities though “lot of data”

Data, the wealth of our time

"Data is a precious thing because they last longer than systems"
-***Tim Barnes Lee***

- Access to data is becoming ultimate competitive advantage
  – E.g. Google+ vs. Facebook
  – Why many organizations try hard to give us free things and keep us always logged in (e.g. Gmail, facebook, search engine tool bars)

Big Data: What?

No single definition; here is from Wikipedia:

- **Big data** is the term for a collection of data sets so large and complex that it becomes difficult to process using on-hand database management tools or traditional data processing applications.
- The challenges include capture, curation, storage, search, sharing, transfer, analysis, and visualization.

Big Data: How?

The Model of Generating/Consuming Data has Changed

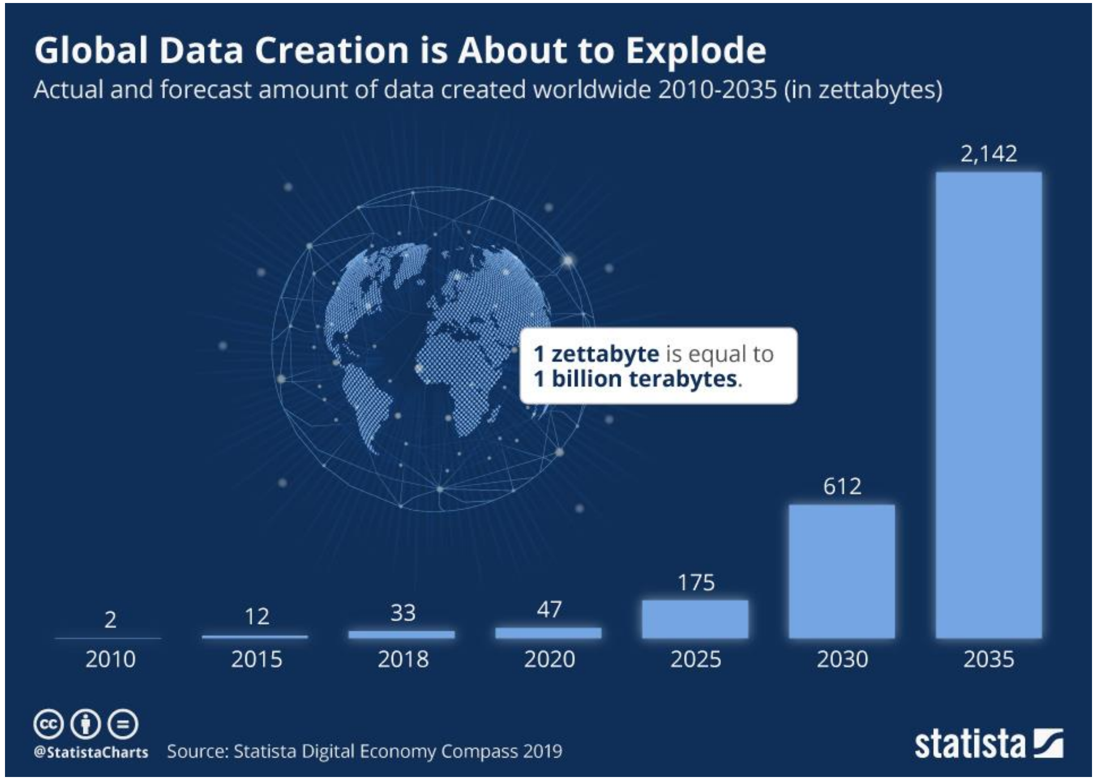
- **Old Model:** Few companies are generating data, all others are consuming data

- **New Model:** all of us are generating data, and all of us are consuming data

Big Data: When?

Data Avalanche/ Moore's law of data

- We are now collecting and converting large amount of data to digital forms
- 90% of the data in the world today was created within the past two years.
- Amount of data we have doubles very fast



Graph Source: https://www.statista.com/chart/17727/global-data-creation-forecasts/

Big Data: Why?

The trend to larger data sets is due to the additional information derivable from analysis of a single large set of related data, as compared to separate smaller sets with the same total amount of data, allowing correlations to be found to:

- Spot business trends
- Determine quality of research
- Prevent diseases
- Link legal citations
- Combat crime
- Determine real-time roadway traffic conditions

Big Data: Who?

Image Source: https://medium.com/@wtrsld/big-data-made-me-do-it-5bfc3f46871c

And while We are on the Subject

"My total linear computational speed has been rated at 60 trillion operations per second".

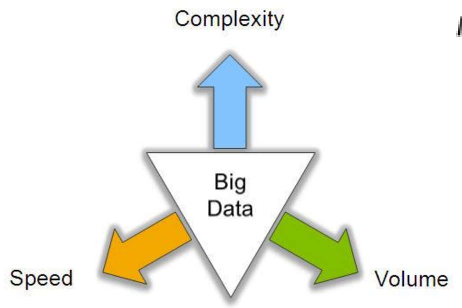
- The Measure of a Man, TNG S02E09 (aired 1989)

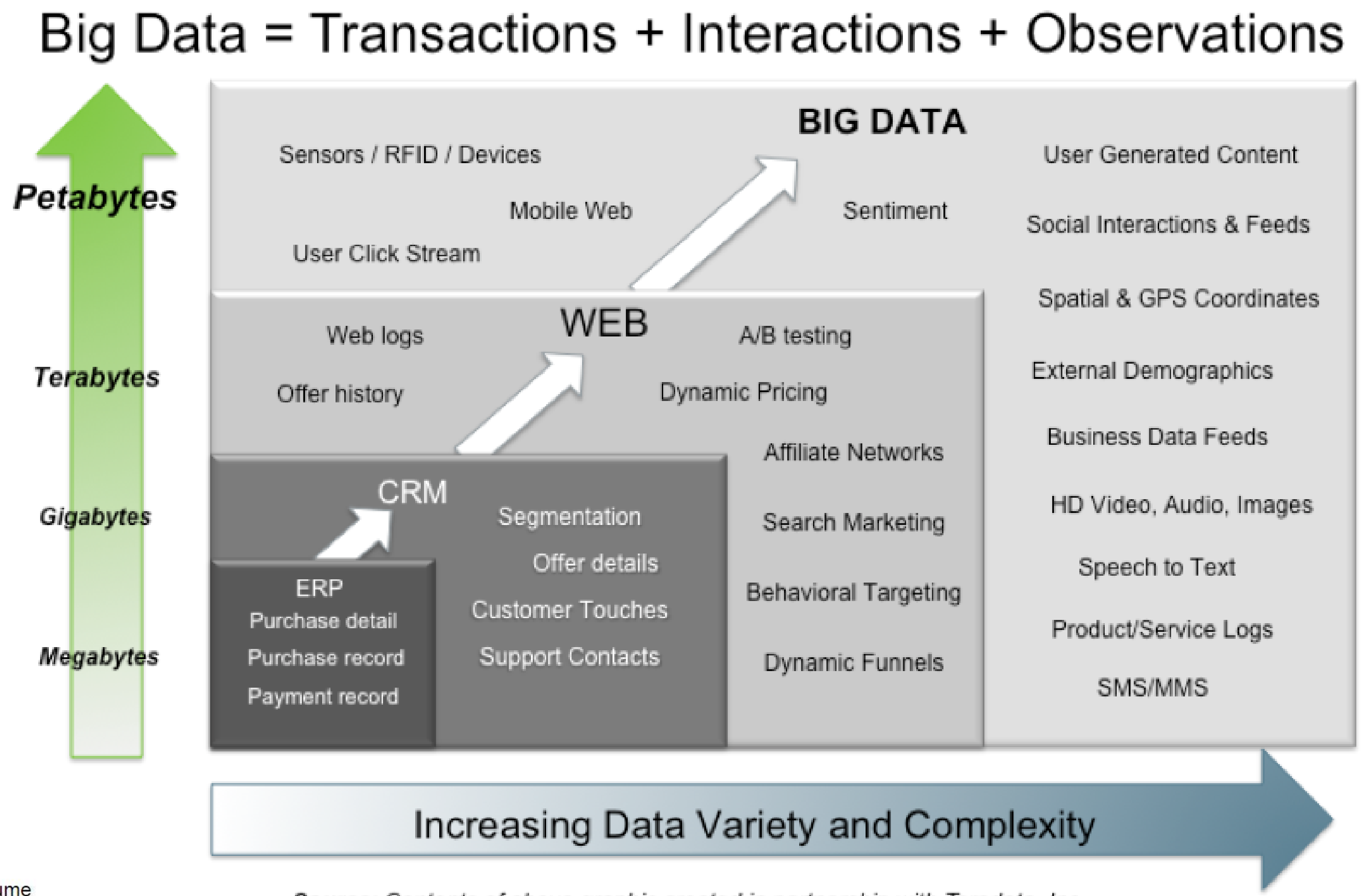
RTX3090 is rated at between 29389 and 35686 gigaflops

- Wikipedia (Released 2020)

Big Data: 3Vs

- Volume
  - **Large** amount of data.
- Velocity
  - Needs to be analyzed **quickly**.
- Variety
  - **Different types** of structured and unstructured data.





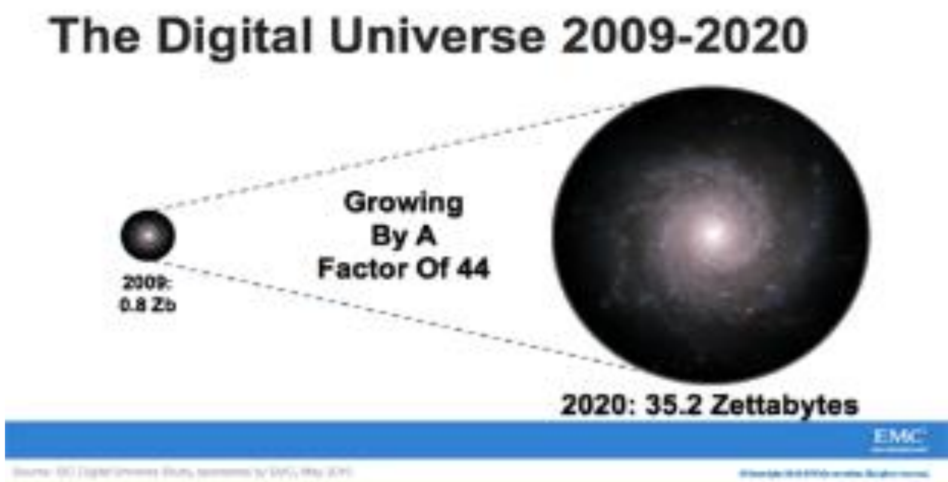
***Source**: Contents of above graphic created in partnership with Teradata, Inc.*

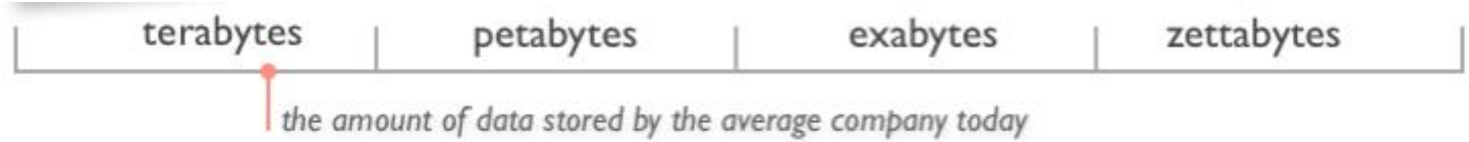
Volume (Scale)

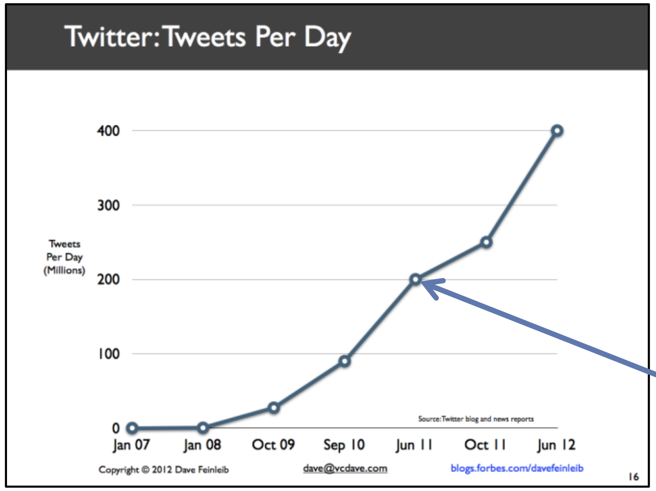
Class	Size	Manage With	How it Fits	Examples
Small	< 10 GB	Excel, R	Fits in one machine’s memory	Thousands of sales figures.
Medium	10 GB – 1 TB	Indexed files, monolithic DB	Fits on one machine’s disk	Millions of web pages
Big	> 1 TB	Hadoop, Distributed DBs	Stored across many machines	Billions of web clicks

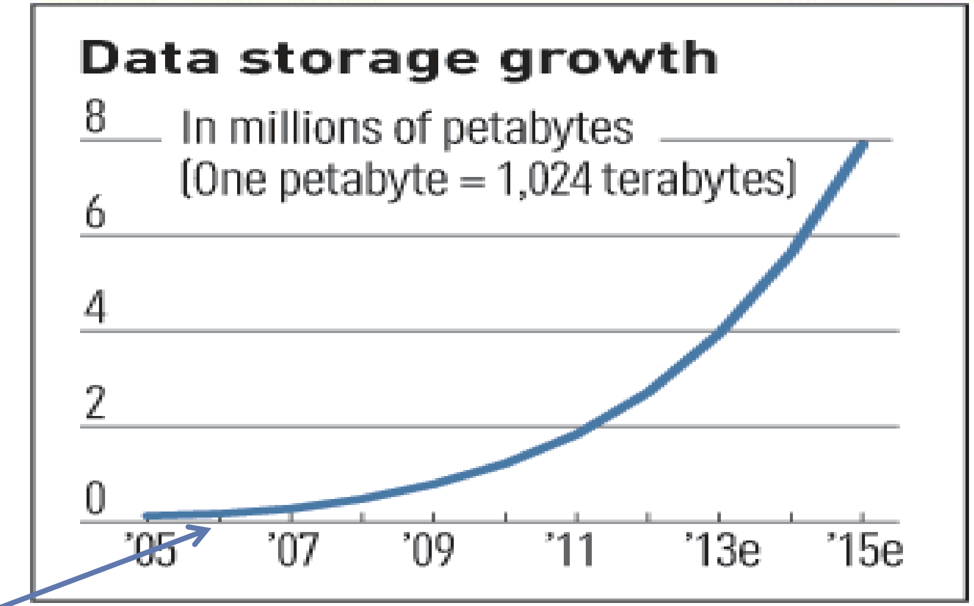
Volume (Scale)

- Data volume is increasing exponentially
  - 44x increase from 2009 2020
  - From 0.8 zettabytes to 35zb









Exponential increase in collected/generated data

Volume (Scale)

12+ TBs **of tweet data every day**

? TBs of data every day

25+ TBs **of log data every day**

30 billion RFID tags today (1.3B in 2005)

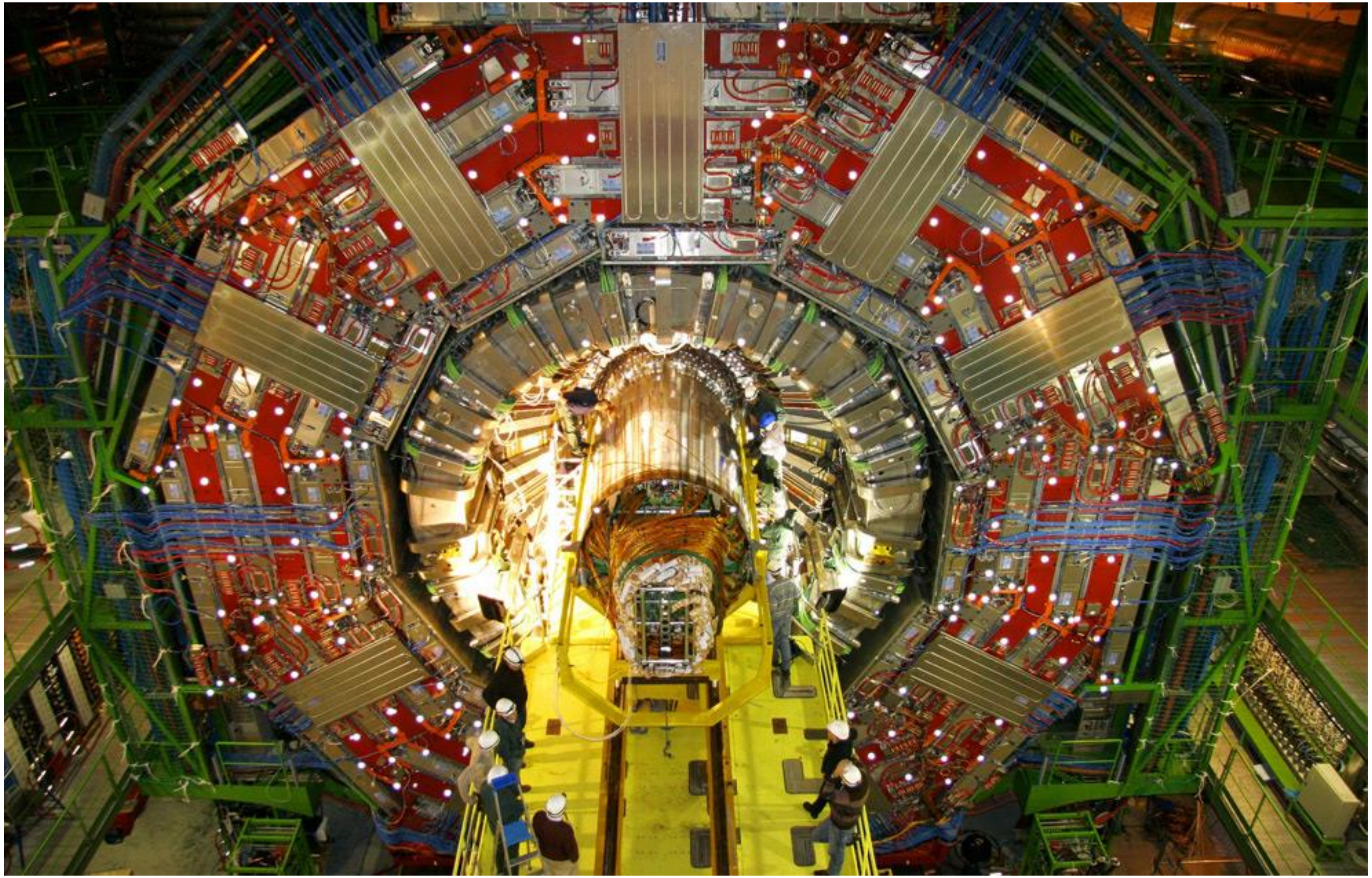
4.6 billion camera phones world wide

100s of millions of GPS enabled devices sold annually

76 million smart meters in 2009… 200M by 2014

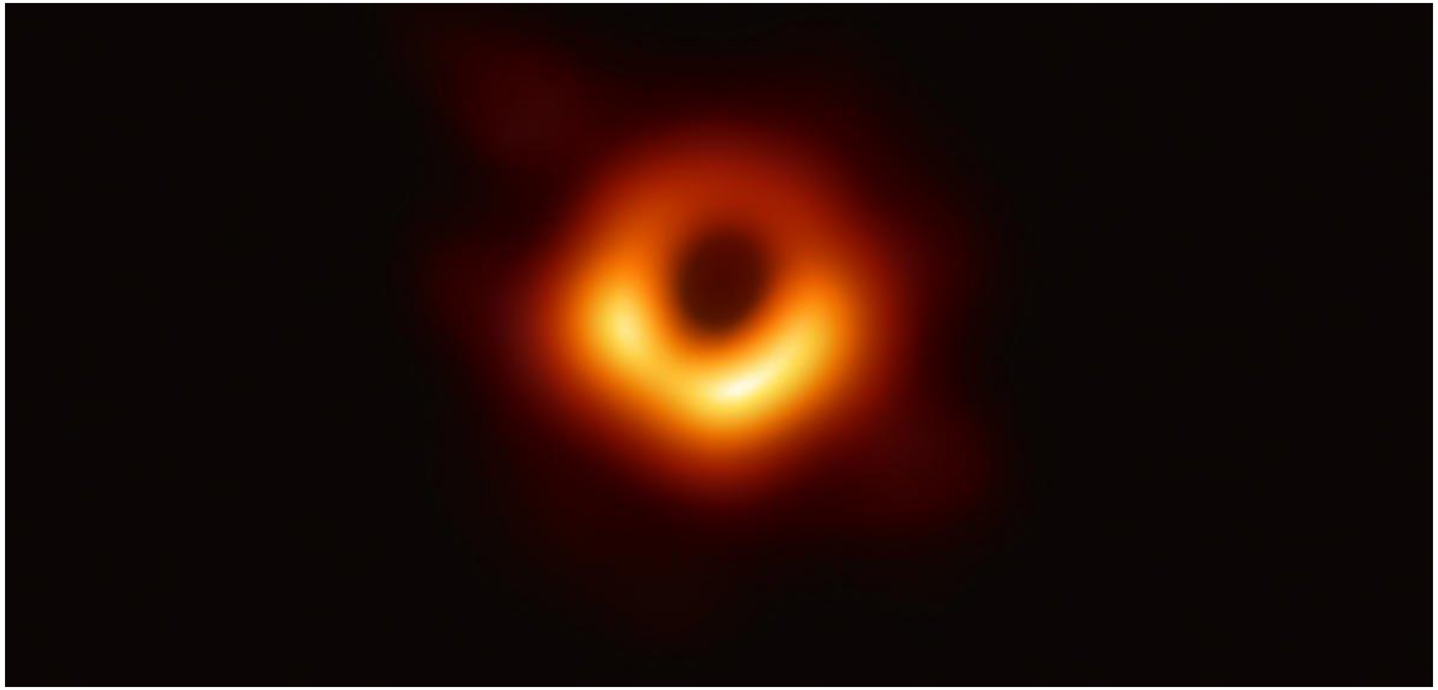
2+ billion people on the Web by end 2011

Volume (Scale)



Maximilien Brice, © CERN

Volume (Scale)



Velocity (Speed)

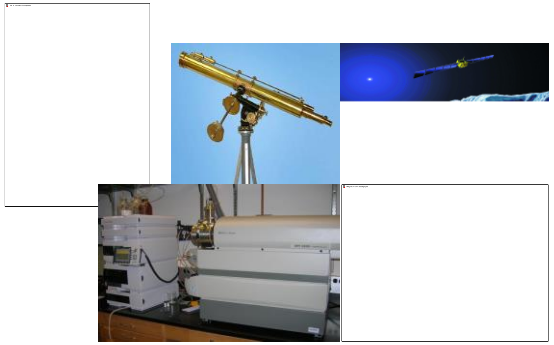
- Data is begin generated fast and need to be processed fast
- Online Data Analytics
- Late decisions ➔ missing opportunities

- Examples:
  – **E-Promotions**: Based on your current location, your purchase history, what you like ➔ send promotions right now for store next to you
  – **Healthcare monitoring:** sensors monitoring your activities and body ➔ any abnormal measurements require immediate reaction
  – **Investments:** Stock/News trends. Tweets by certain people.

Velocity (Speed): Real-time/Fast Data

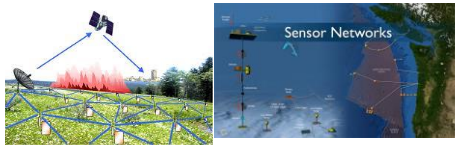
- The progress and innovation is no longer hindered by the ability to collect data
- But, by the ability to manage, analyze, summarize, visualize, and discover knowledge from the collected data in a timely manner and in a scalable fashion

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



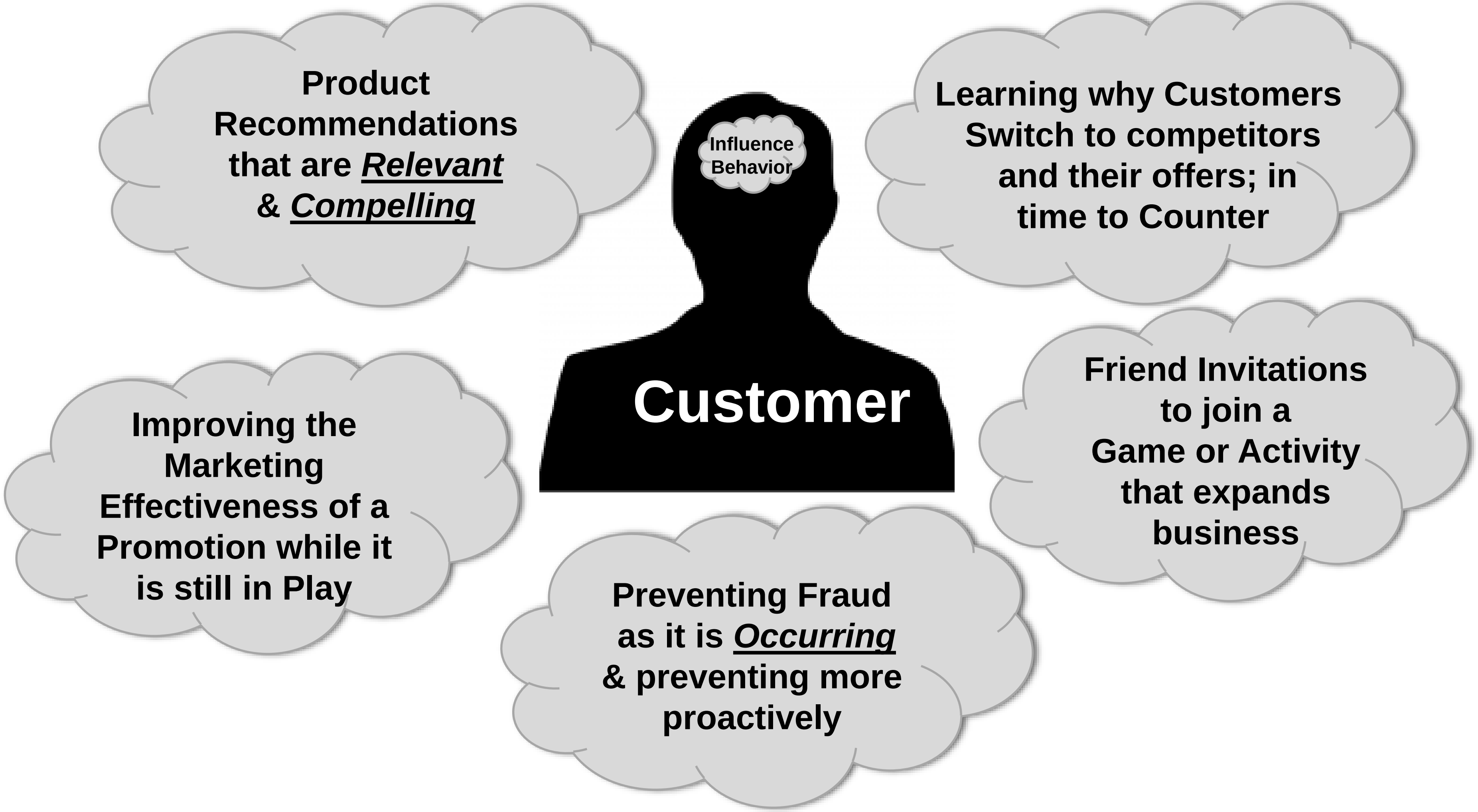
Scientific instruments
(collecting all sorts of data)

Mobile devices
(tracking all objects all the time)

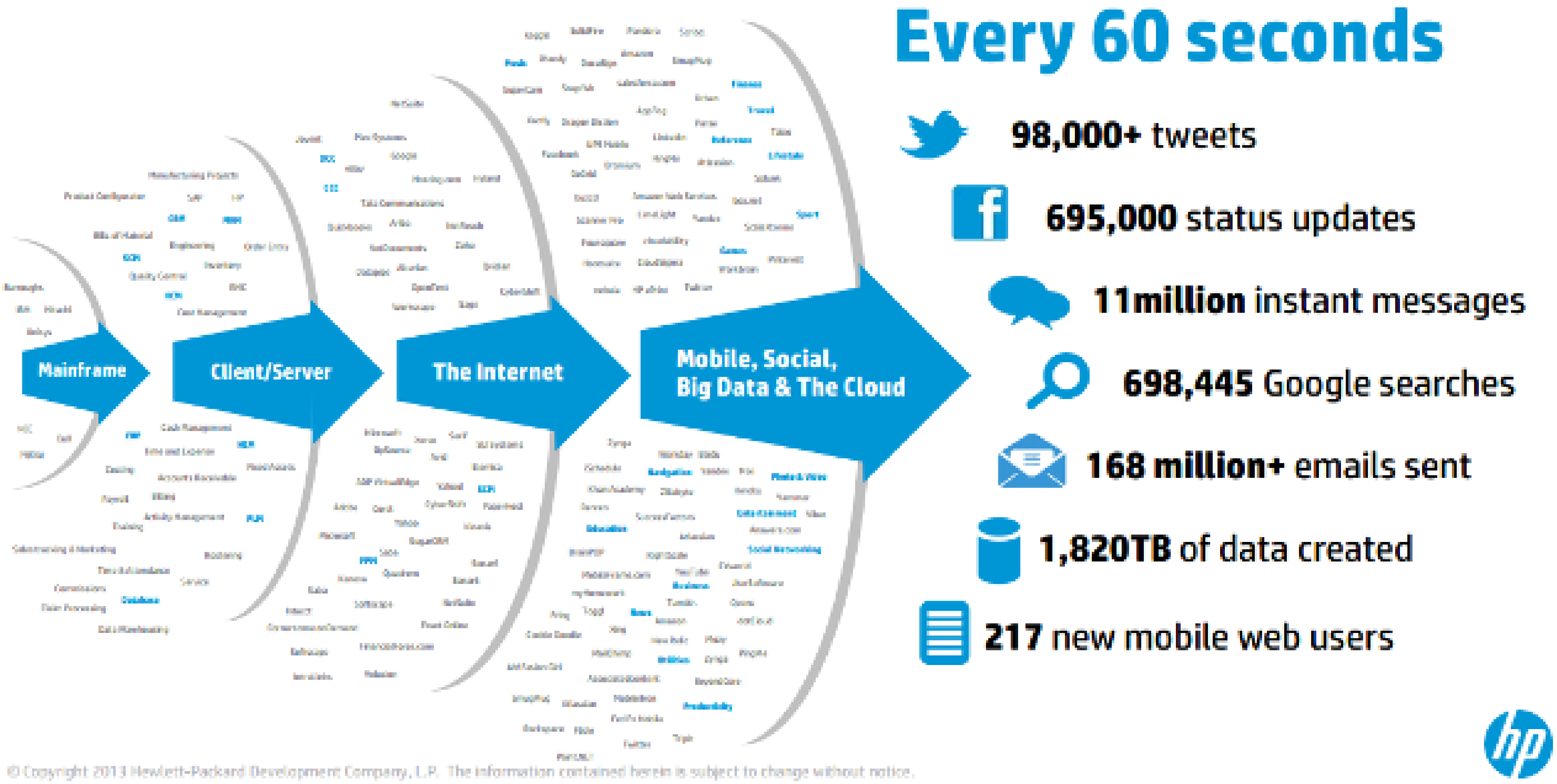


Sensor technology and networks
(measuring all kinds of data)

Velocity (Speed): Real-Time Analytics/Decision Requirement



Velocity (Speed):



Velocity (Speed):

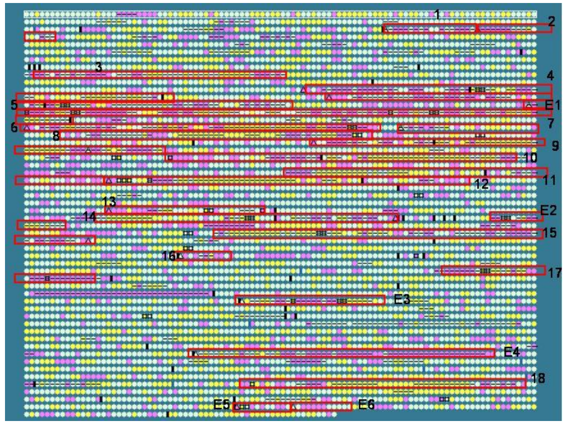
Latency Numbers Every Programmer Should Know?

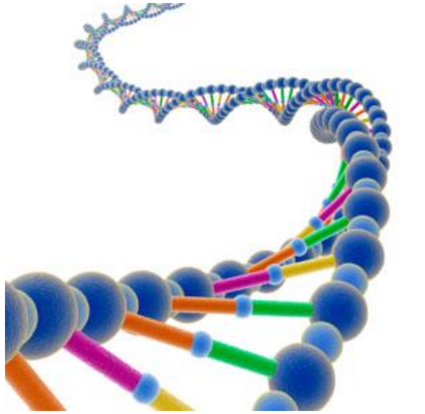
```
Latency Comparison Numbers
--------------------------
L1 cache reference                           0.5 ns
Branch mispredict                            5   ns
L2 cache reference                           7   ns                      14x L1 cache
Mutex lock/unlock                           25   ns
Main memory reference                      100   ns                      20x L2 cache, 200x L1 cache
Read 4K randomly from memory             1,000   ns      0.001 ms
Compress 1K bytes with Zippy             3,000   ns
Send 1K bytes over 1 Gbps network       10,000   ns      0.01 ms
Read 4K randomly from SSD*             150,000   ns      0.15 ms
Read 1 MB sequentially from memory     250,000   ns      0.25 ms
Round trip within same datacenter      500,000   ns      0.5  ms
Read 1 MB sequentially from SSD*     1,000,000   ns      1    ms     4X memory
Disk seek                           10,000,000   ns     10    ms     20x datacenter roundtrip
Read 1 MB sequentially from disk    20,000,000   ns     20    ms     80x memory, 20X SSD
Send packet CA->Netherlands->CA    150,000,000   ns    150    ms
```

Source: Jeff Dean and Peter Norvig (Google), with some additions

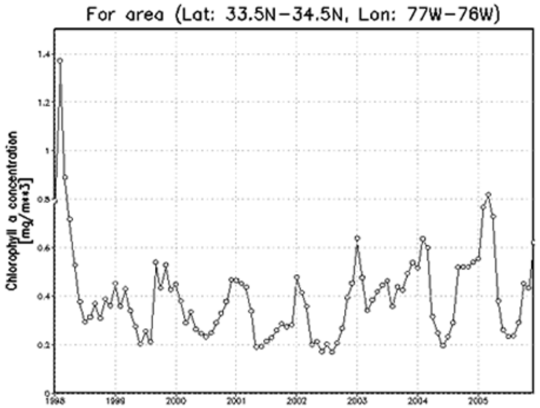
https://gist.github.com/hellerbarde/2843375

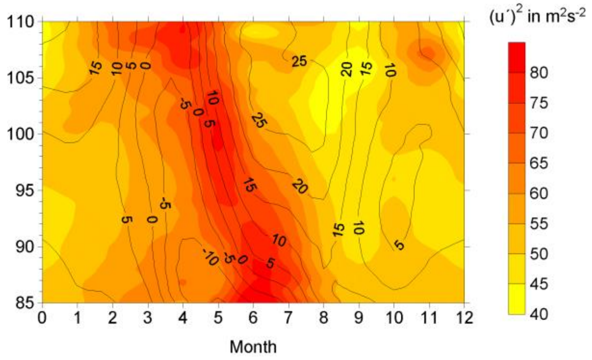
Variety (Complexity)





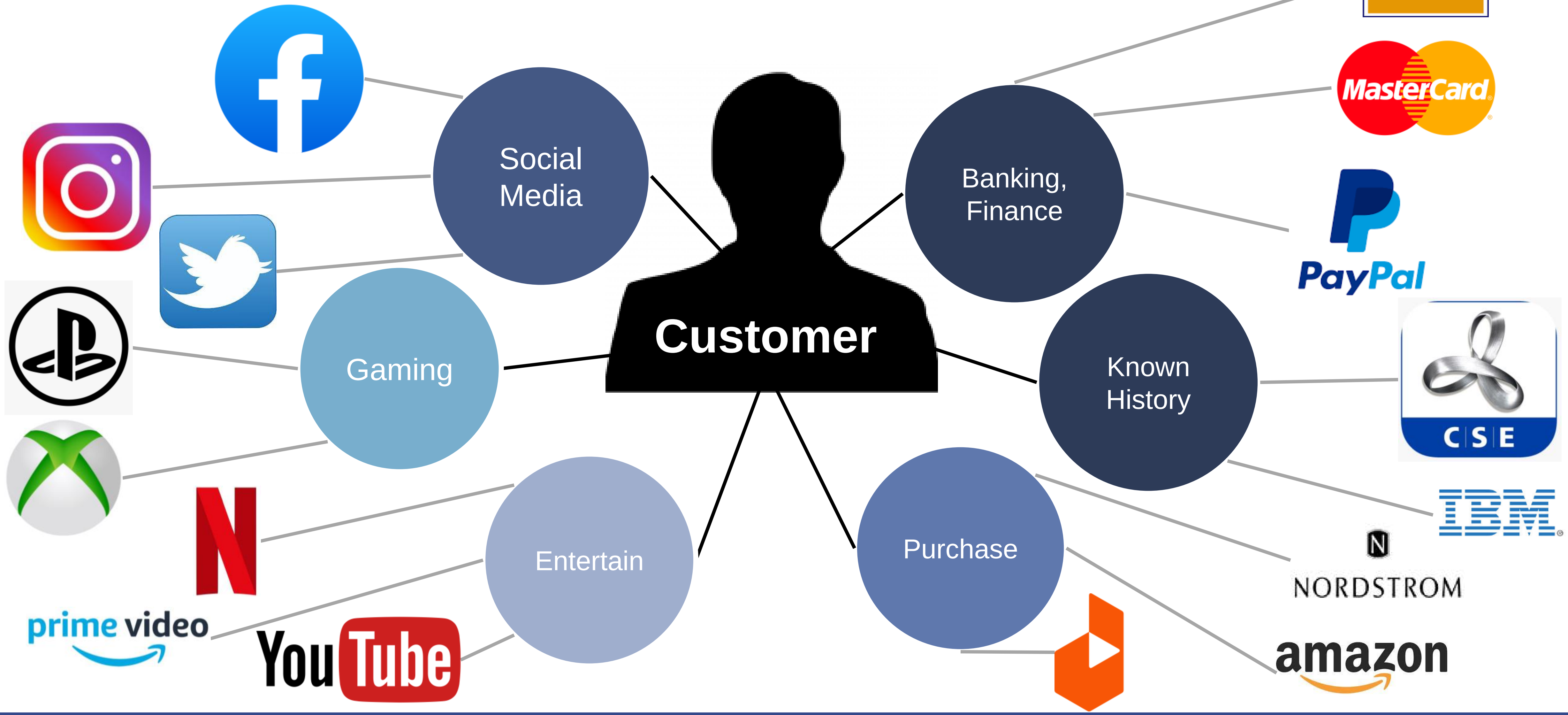
- Relational Data (Tables/Transaction/Legacy Data)
- Text Data (Web)
- Semi-structured Data (XML)
- Graph Data
  - Social Network, Semantic Web (RDF), …
- Streaming Data
  - You can only scan the data once
- A single application can be generating/collecting many types of data
- Big Public Data (online, weather, finance, etc)



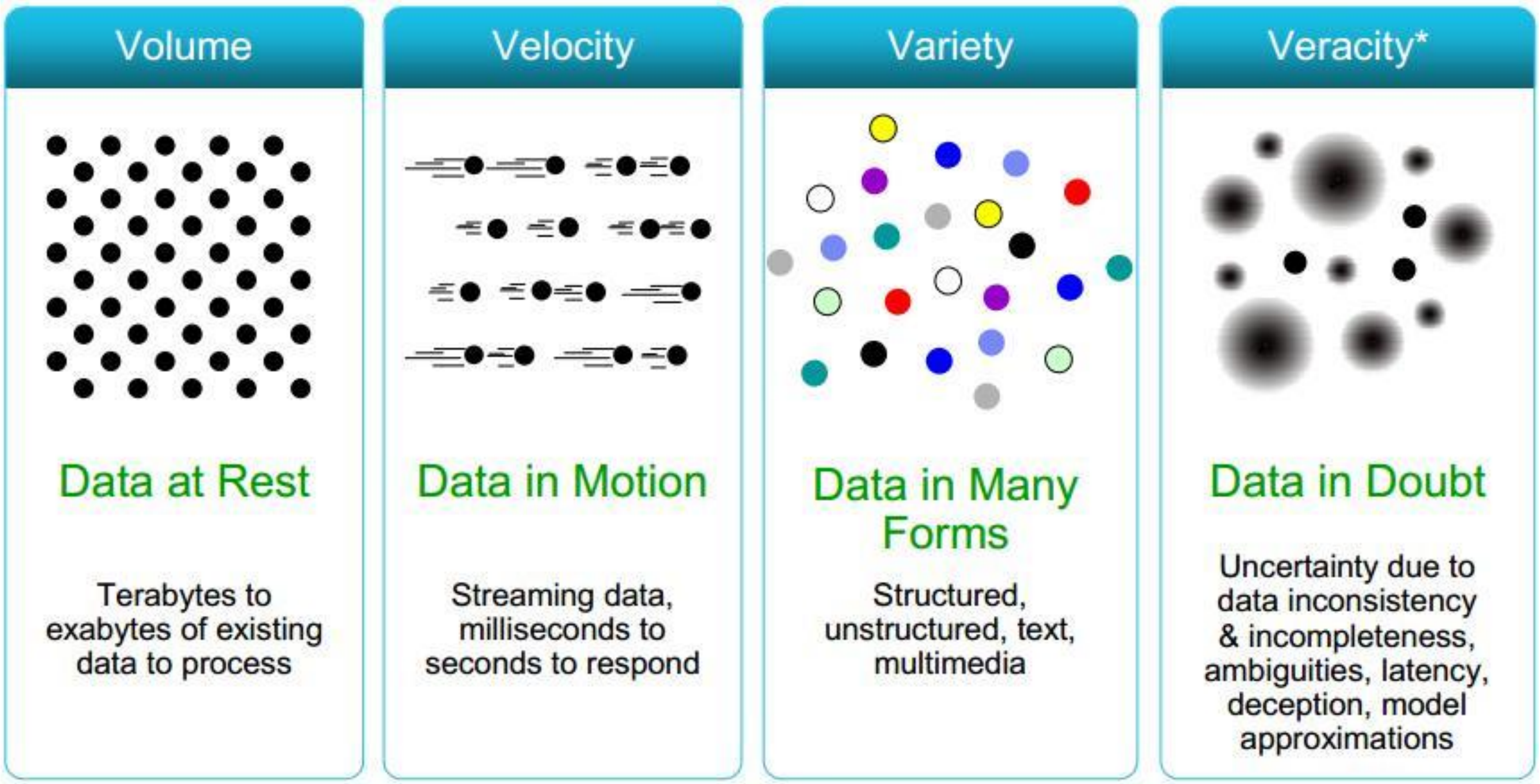


To extract knowledge➔ all these types of data need to linked together

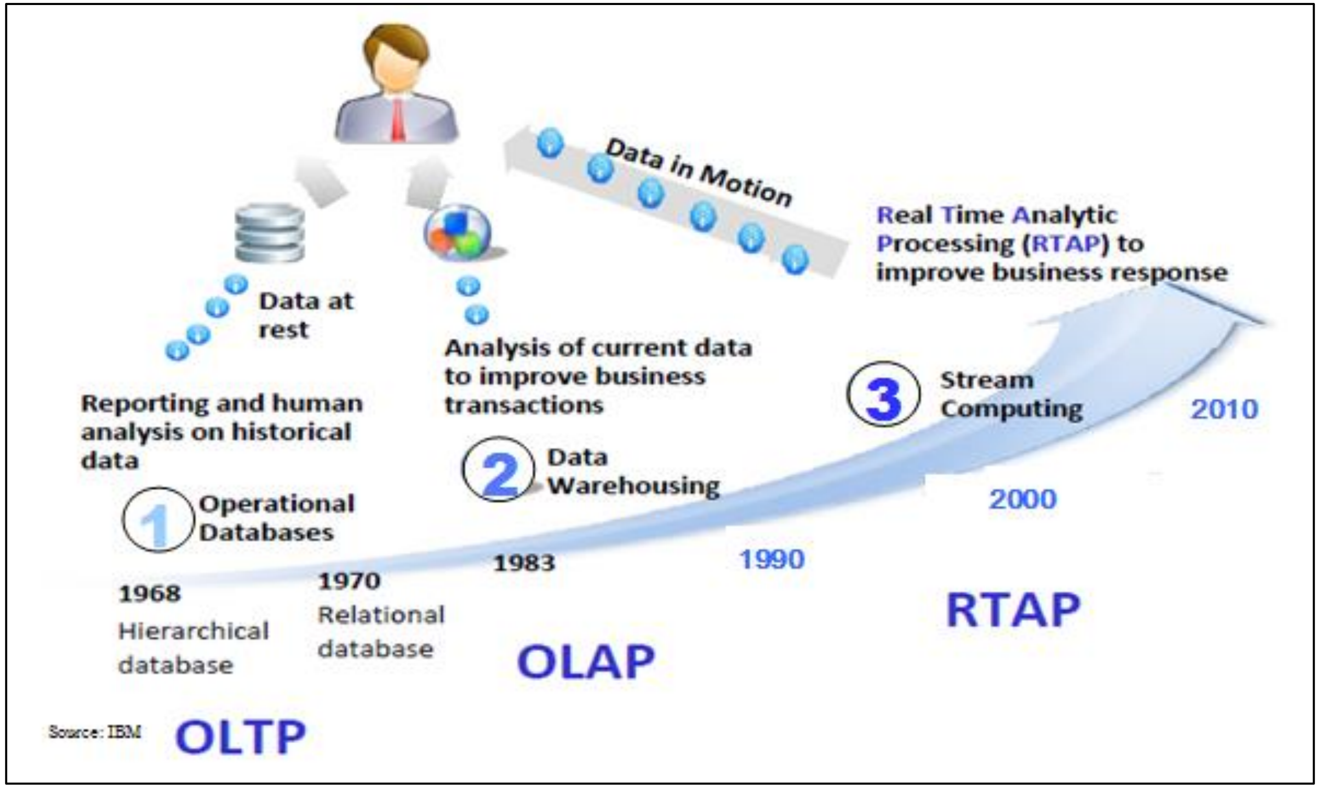
Variety (Complexity): A Single View to the Customer



Some Make it 4V's

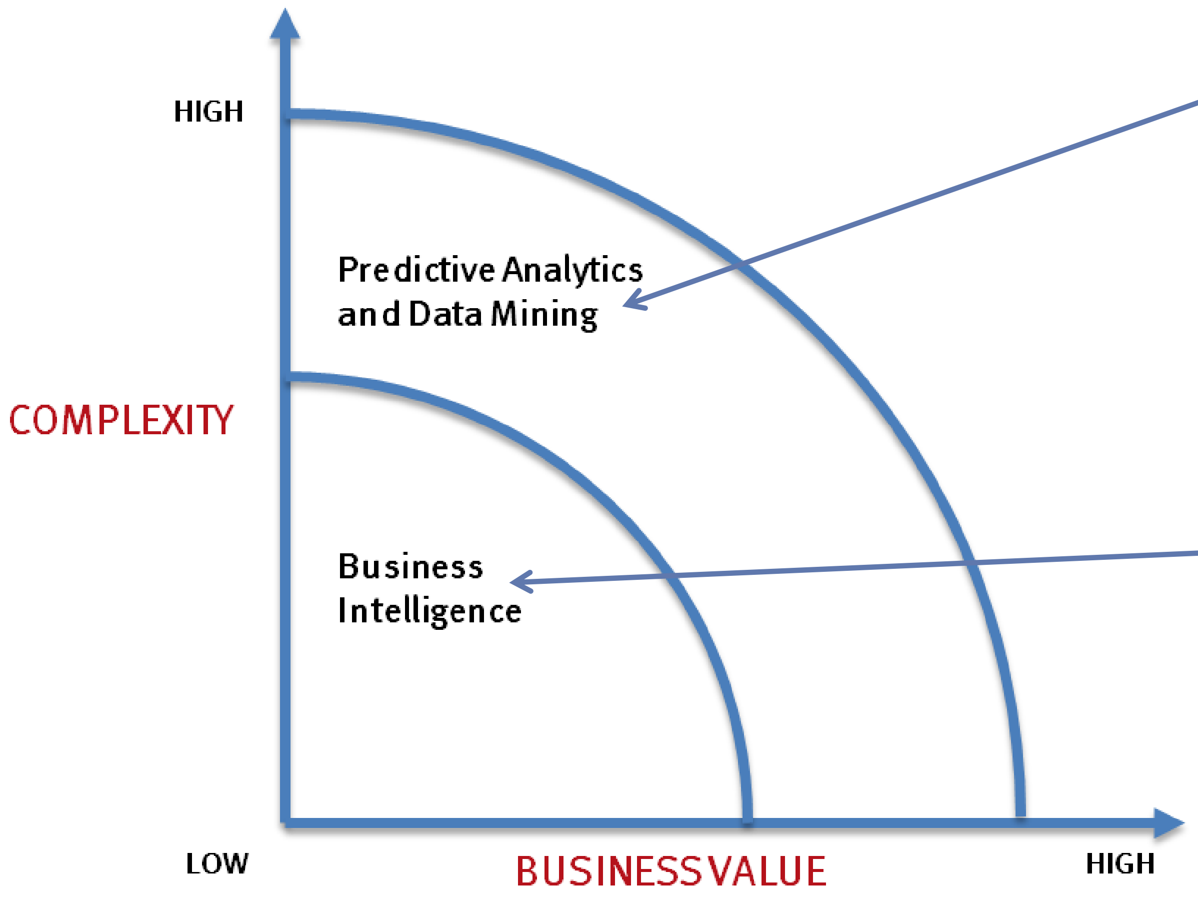


Harnessing Big Data



- **OLTP:** Online Transaction Processing (DBMSs)
- **OLAP:** Online Analytical Processing (Data Warehousing)
- **RTAP:** Real-Time Analytics Processing (Big Data Architecture & technology)

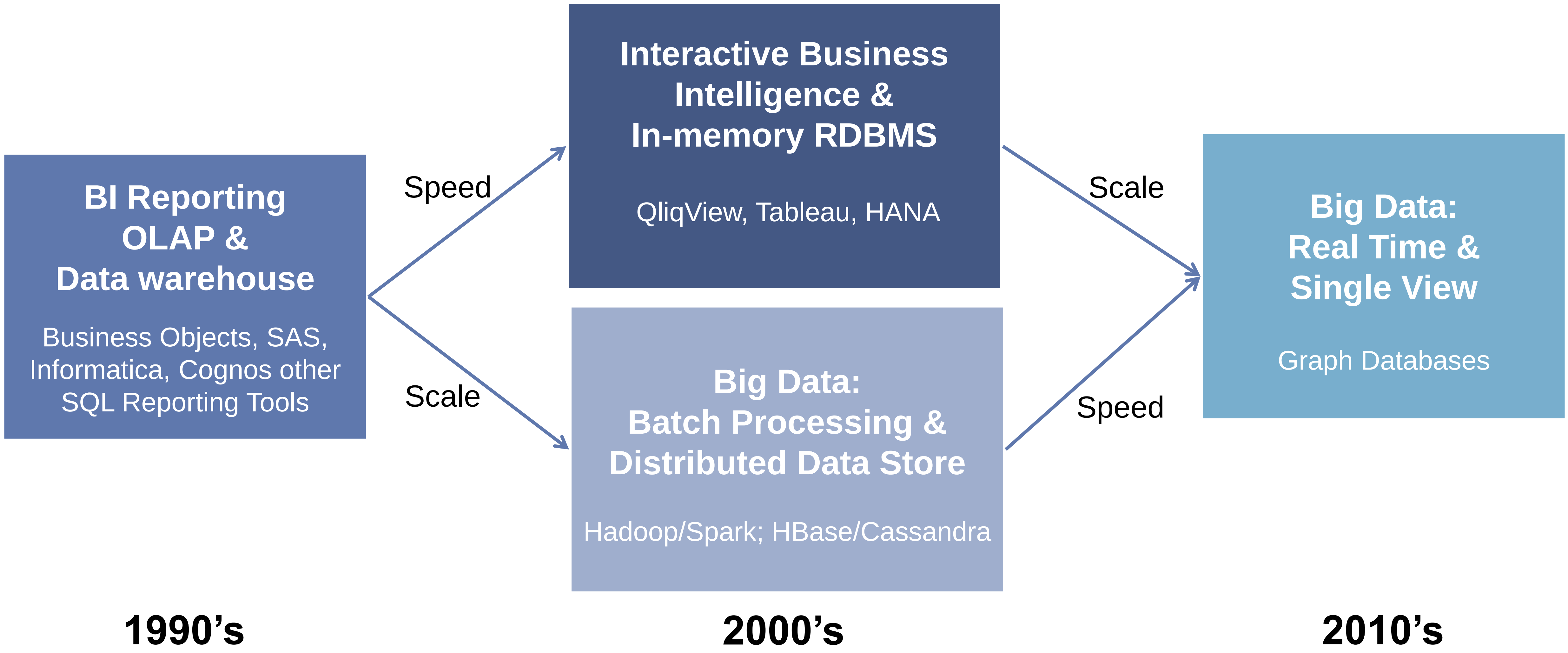
What’s driving Big Data



- Optimizations and predictive analytics
- Complex statistical analysis
- All types of data, and many sources
- Very large datasets
- More of a real-time

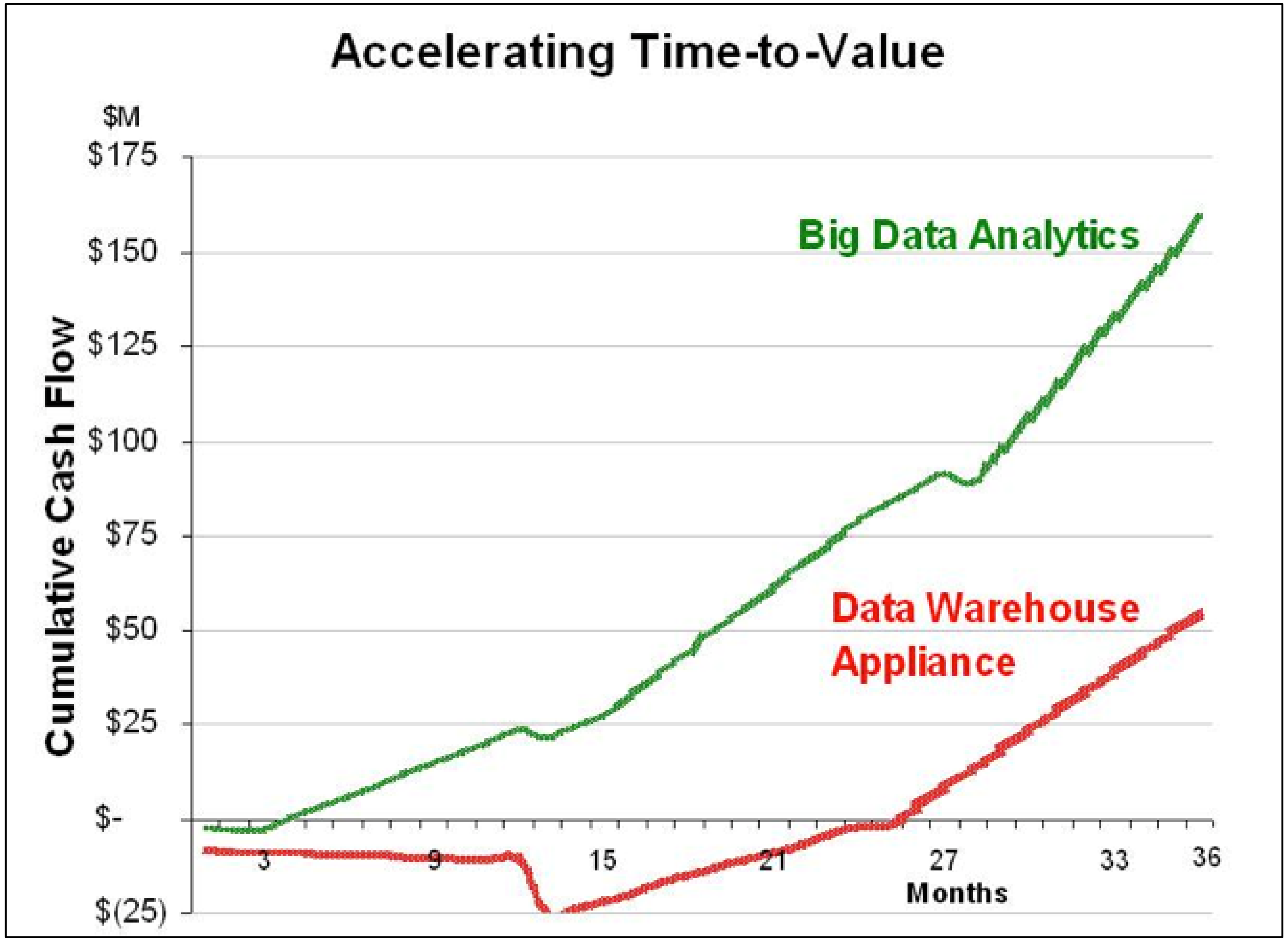
- Ad-hoc querying and reporting
- Data mining techniques
- Structured data, typical sources
- Small to mid-size datasets

The Evolution of Business Intelligence

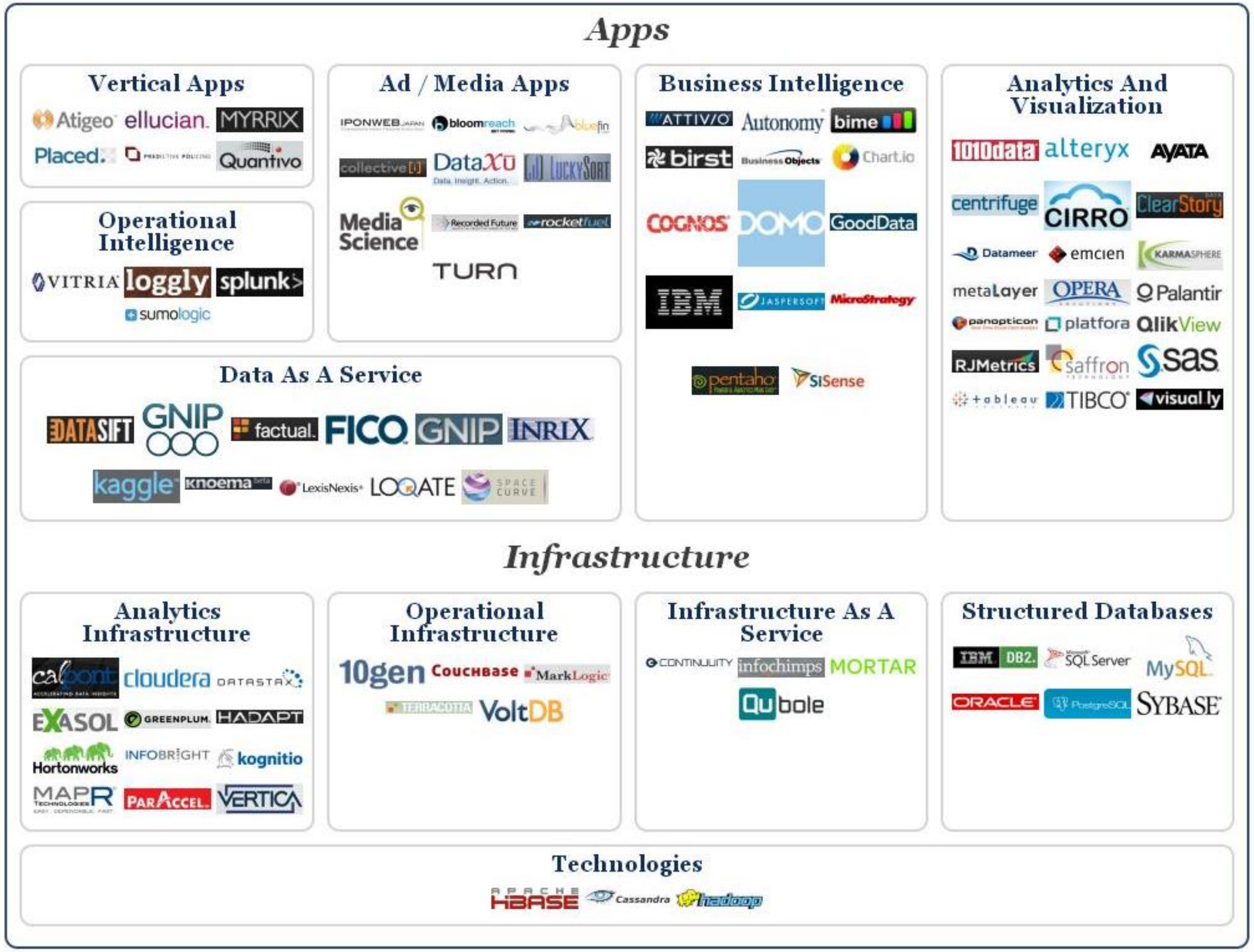


Big Data Analytics

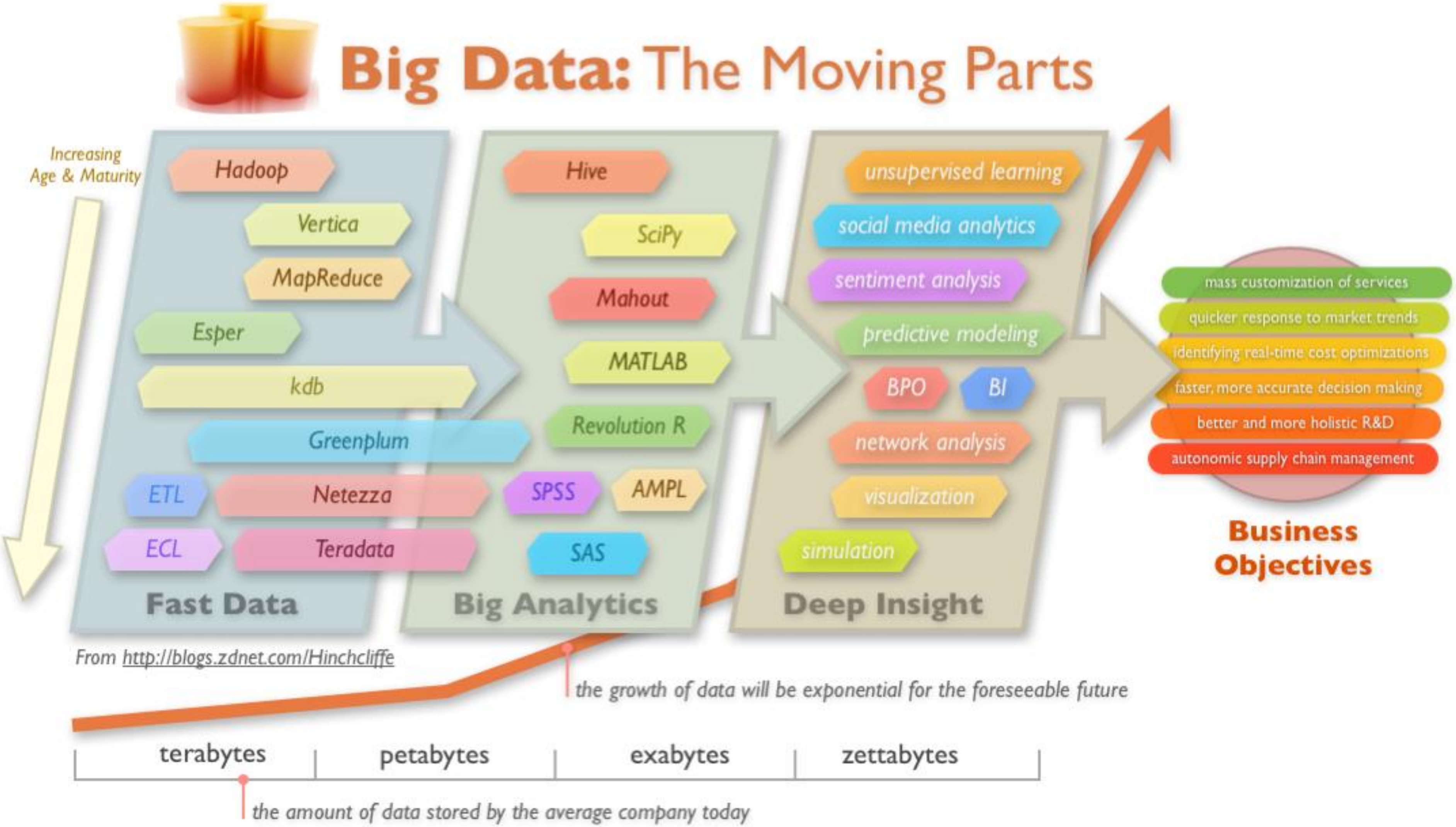
- Big data is more real-time in nature than traditional Data Warehouse (DW) applications
- Traditional DW architectures (e.g. Exadata, Teradata) are not well-suited for big data apps
- Shared nothing, massively parallel processing, scale out architectures are well-suited for big data apps



The Big Data Landscape

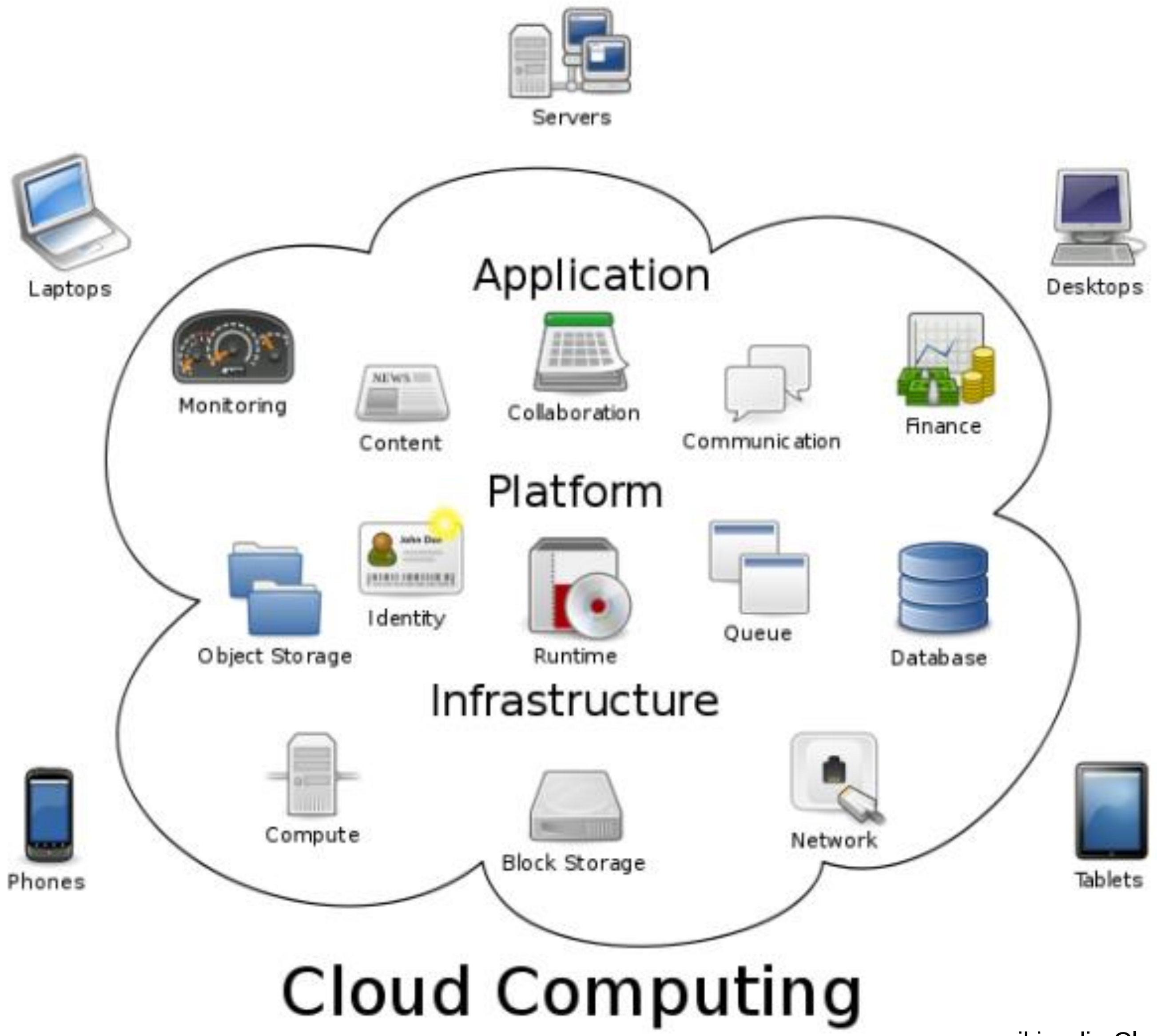


Big Data Technology



Cloud Computing

- IT resources provided as a service
  - Compute, storage, databases, queues
- Clouds leverage economies of scale of commodity hardware
  - Cheap storage, high bandwidth networks & multicore processors
  - Geographically distributed data centers
- Offerings from Microsoft, Amazon, Google, …



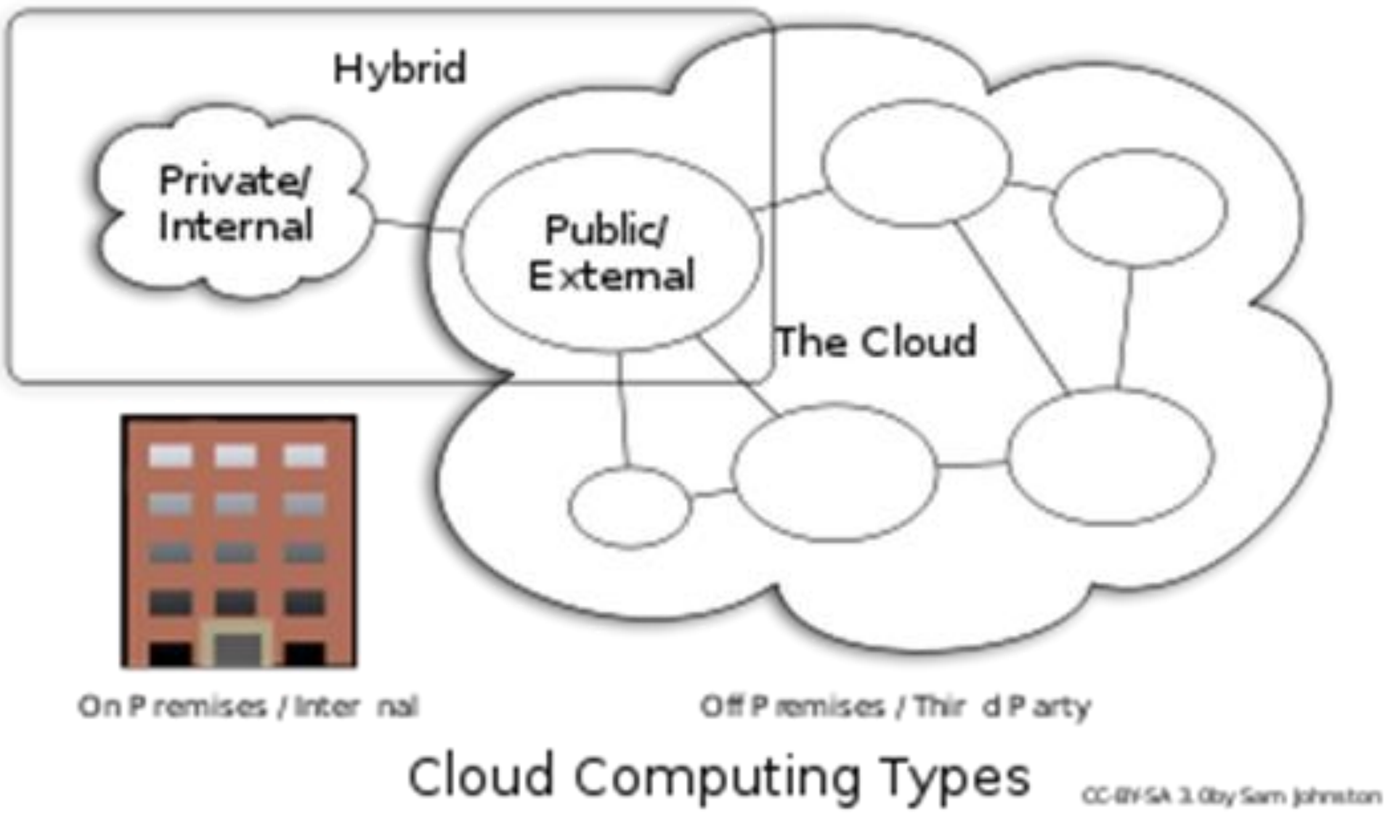
wikipedia:Cloud Computing

Benefits

- Cost & management
  – Economies of scale, “out-sourced” resource management
- Reduced Time to deployment
  – Ease of assembly, works “out of the box”
- Scaling
  – On demand provisioning, co-locate data and compute
- Reliability
  – Massive, redundant, shared resources
- Sustainability
  – Hardware not owned

Types of Cloud Computing

- **Public Cloud**: Computing infrastructure is hosted at the vendor's premises.
- **Private Cloud**: Computing architecture is dedicated to the customer and is not shared with other organisations.
- **Hybrid Cloud**: Organisations host some critical, secure applications in private clouds. The not so critical applications are hosted in the public cloud
  - **Cloud bursting**: the organisation uses its own infrastructure for normal usage, but cloud is used for peak loads.
- **Community Cloud:** The task/ burden of hosting the cloud is shared.



Classification of Cloud Computing based on Service Provided

- Infrastructure as a service (IaaS)
  - Why buy machines when you can rent cycles?
  - Offering hardware related services using the principles of cloud computing. These could include storage services (database or disk storage) or virtual servers.
  - Amazon EC2, Amazon S3, Rackspace Cloud Servers and Flexiscale.
- Platform as a Service (PaaS)
  - Give me nice API and take care of the maintenance, upgrades, …
  - Offering a development platform on the cloud.
  - Google's Application Engine, Microsofts Azure, Salesforce.com's force.com.
- Software as a service (SaaS)
  - Just run it for me!
  - Including a complete software offering on the cloud. Users can access a software application hosted by the cloud vendor on pay-per-use basis. This is a well-established sector.
  - Salesforce.coms' offering in the online Customer Relationship Management (CRM) space, Googles gmail and Microsofts hotmail, Google docs.

More Refined Categorization

- Storage-as-a-service
- Database-as-a-service
- Information-as-a-service
- Process-as-a-service
- Application-as-a-service
- Platform-as-a-service
- Integration-as-a-service
- Security-as-a-service
- Management/Governance-as-a-service
- Testing-as-a-service
- Infrastructure-as-a-service

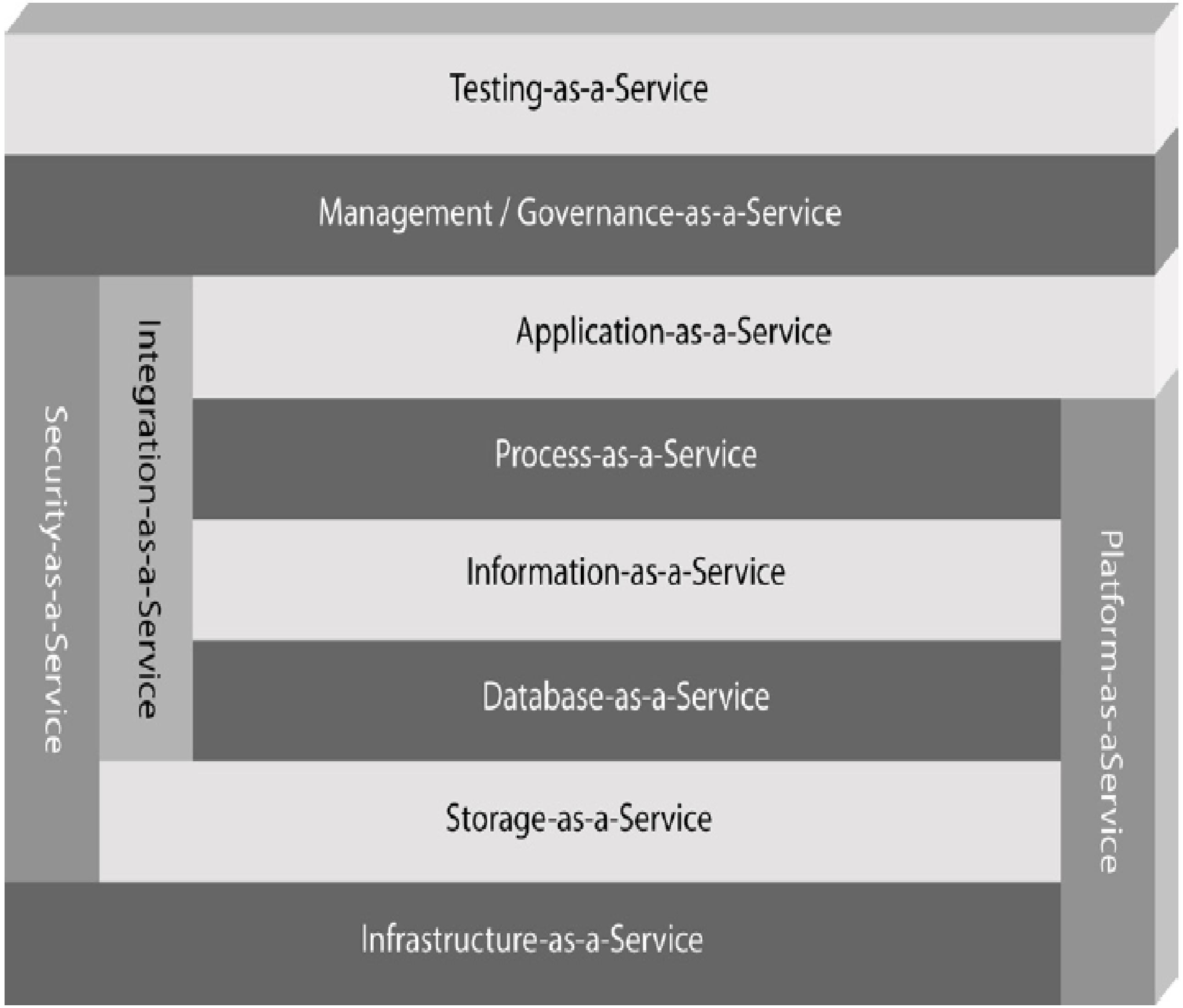


Figure 1: The patterns or categories of cloud computing providers allow you to use a discrete set of services within your architecture.

InfoWorld Cloud Computing Deep Dive

Map-Reduce

Why MapReduce?

- Before MapReduce
  - Large Concurrent Systems
  - Grid Computing
  - Rolling Your Own Solution
- Considerations
  - Threading is hard!
  - How do you scale to more machines?
  - How do you handle machine failures?
  - How do you facilitate communication between nodes?
  - Does your solution scale?

- **Need more power? Scale out, not up!**
  - Large number of **commodity servers** as opposed to some high end specialized servers

Typical problem solved by MapReduce

- Read a lot of data
- Map: extract something you care about from each record
  - Mappers read in data from the filesystem, and output (typically) modified data
- Shuffle and Sort
- Reduce: aggregate, summarize, filter, or transform
  - Reducers collect all of the mappers output on the keys, and output (typically) reduced data
- Write the results
  - The outputted data is written to disk

All data is in terms of key value pairs

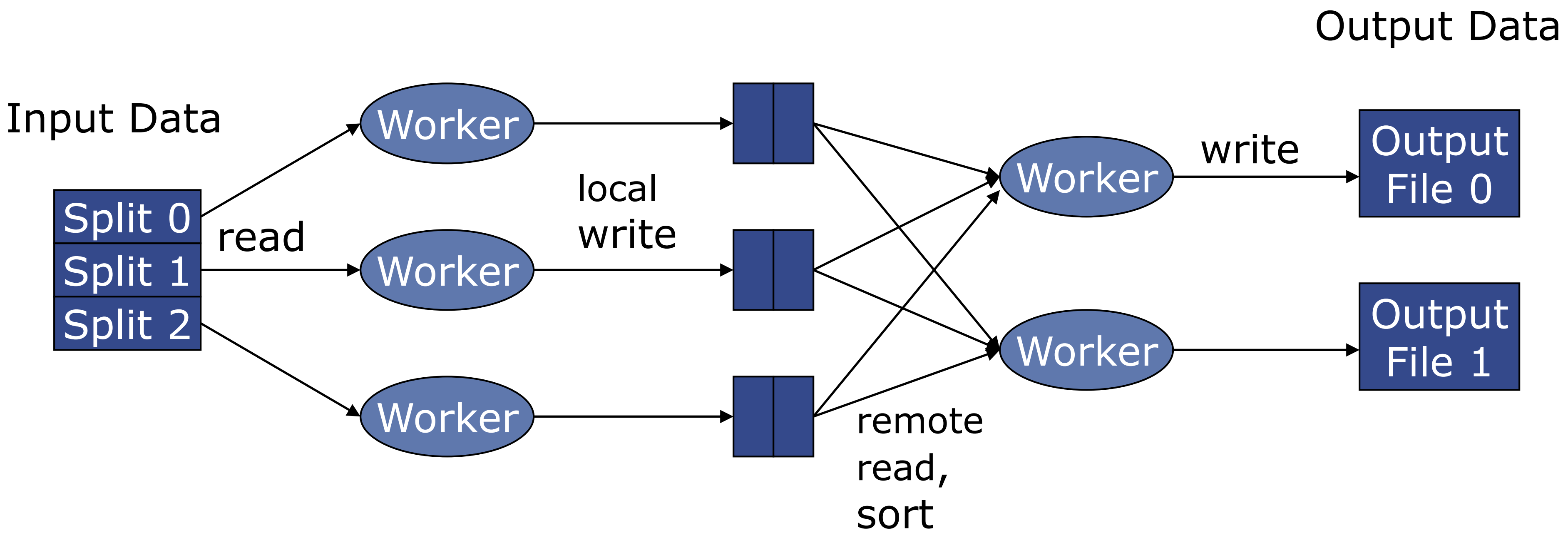
An Example Program

- I will present the concepts of MapReduce using the “typical example” of MR, Word Count
- The input of this program is a volume of raw text, of unspecified size (could be KB, MB, TB, it doesn’t matter!)
- The output is a list of words, and their occurrence count. Assume that words are split correctly, ignoring capitalization and punctuation.
- Example
  - The doctor went to the store. =>
    - the, 2
    - doctor, 1
    - went, 1
    - to, 1
    - store, 1

MapReduce vs Hadoop

- The paper is written by two researchers at Google, and describes their programming paradigm
  - Automatic parallelization, distribution
  - I/O scheduling
    - Load balancing
    - Network and data transfer optimization
  - Fault tolerance
    - Handling of machine failures
- Unless you work at Google, or use Google App Engine, you won’t use it!
- Open Source implementation is Hadoop MapReduce
  - Not developed by Google
  - Started by Yahoo
- Google’s implementation (at least the one described) is written in C++
- Hadoop is written in Java

MapReduce Workflow



Map

extract something you care about from each record

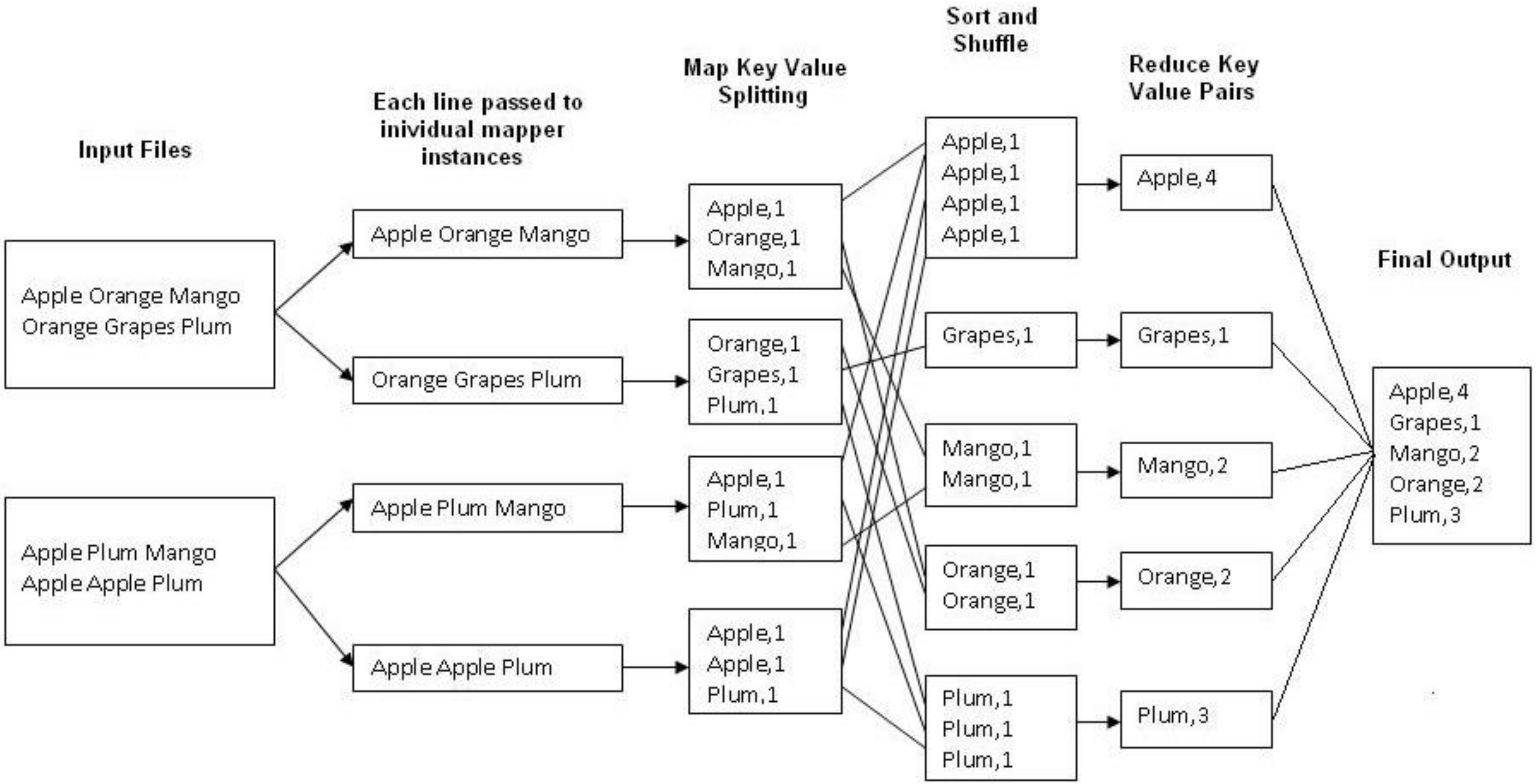
Reduce

aggregate, summarize, filter, or transform

Mappers and Reducers

- Mappers and Reducers are typically single threaded and deterministic
  - Determinism allows for restarting of failed jobs, or speculative execution
- Need to handle more data? Just add more Mappers/Reducers!
  - No need to handle multithreaded code ☺
  - Since they're all independent of each other, you can run (almost) arbitrary number of nodes
- Mappers/Reducers run on arbitrary machines. A machine typically multiple map and reduce slots available to it, typically one per processor core
- Mappers/Reducers run entirely independent of each other
  - In Hadoop, they run in separate JVMs

Example: Word Count



http://kickstarthadoop.blogspot.ca/2011/04/word-count-hadoop-map-reduce-example.html

Input Splitter

- Is responsible for splitting your input into multiple chunks
- These chunks are then used as input for your mappers
- Splits on logical boundaries. The default is 64MB per chunk
  - Depending on what you’re doing, 64MB might be a LOT of data! You can change it
- Typically, you can just use one of the built in splitters, unless you are reading in a specially formatted file

Mapper

- Reads in input pair <Key,Value>
- Outputs a pair <K', V'>
  - Let's count number of each word in user queries (or Tweets/Blogs)
  - The input to the mapper will be <queryID, QueryText>:

```
<Q1,"The teacher went to the store. The store was closed; the store opens
in the morning. The store opens at 9am." >
```

  - The output would be:

```
<The, 1> <teacher, 1> <went, 1> <to, 1> <the, 1> <store,1> <the, 1>
<store, 1> <was, 1> <closed, 1> <the, 1> <store,1> <opens, 1> <in, 1>
<the, 1> <morning, 1> <the 1> <store, 1> <opens, 1> <at, 1> <9am, 1>
```

Reducer

- Accepts the Mapper output, and aggregates values on the key
  – All inputs with the same key *must* go to the same reducer!
- Input is typically sorted, output is output exactly as is
- For our example, the reducer input would be:

  <the, 1> <teacher, 1> <went, 1> <to, 1> <the, 1> <**store**, 1> <the, 1> **<store, 1>** <was, 1> <closed, 1> <the, 1> <**store**, 1> <opens,1> <in, 1> <the, 1> <morning, 1> <the 1> <**store**, 1> <opens, 1> <at, 1> <9am, 1>

- The output would be:

  <the, 6> <teacher, 1> <went, 1> <to, 1> **<store, 4>** <was, 1> <closed, 1> <opens, 2> <morning, 1> <at, 1> <9am, 1>

Reducer

- Accepts the Mapper output, and aggregates values on the key
  – All inputs with the same key *must* go to the same reducer!
- Input is typically sorted, output is output exactly as is
- For our example, the reducer input would be:

  <the, 1> <teacher, 1> <went, 1> <to, 1> <the, 1> **<store**, 1> <the, 1> **<store, 1>** <was, 1> <closed, 1> <the, 1> <**store**, 1> <opens,1> <in, 1> <the, 1> <morning, 1> <the 1> <**store**, 1> <opens, 1> <at, 1> <9am, 1>

- The output would be:

  <the, 6> <teacher, 1> <went, 1> <to, 1> **<store, 4>** <was, 1> <closed, 1> <opens, 2> <morning, 1> <at, 1> <9am, 1>

Combiner

- Essentially an intermediate reducer
- Is optional
- Reduces output from each mapper, reducing bandwidth and sorting
- Cannot change the type of its input
  - Input types must be the same as output types

Output Committer

- Is responsible for taking the reduce output, and committing it to a file
- Typically, this committer needs a corresponding input splitter (so that another job can read the input)
- Again, usually built in splitters are good enough, unless you need to output a special kind of file

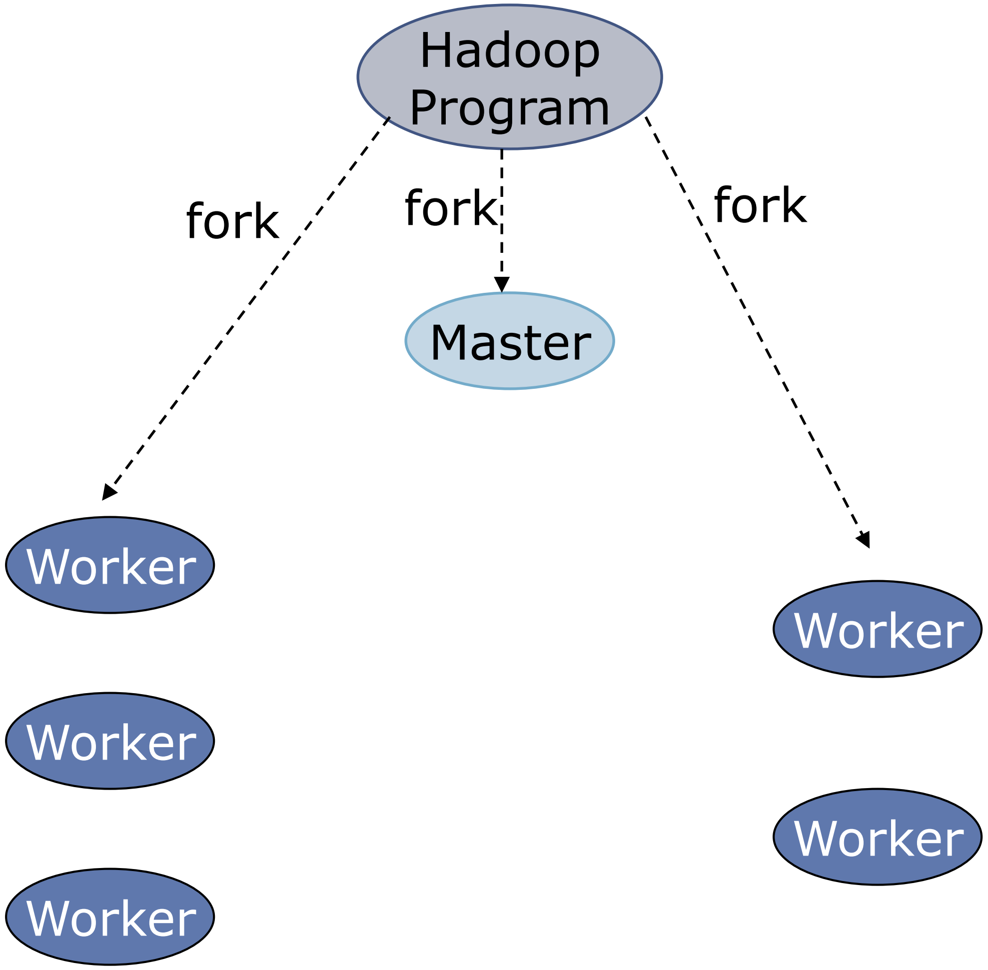
Partitioner (Shuffler)

- Decides which pairs are sent to which reducer
- Default is simply:
  - Key.hashCode() % numOfReducers
- User can override to:
  - Provide (more) uniform distribution of load between reducers
  - Some values might need to be sent to the same reducer
    - Ex. To compute the relative frequency of a pair of words <W1, W2> you would need to make sure all of word W1 are sent to the same reducer
  - Binning of results

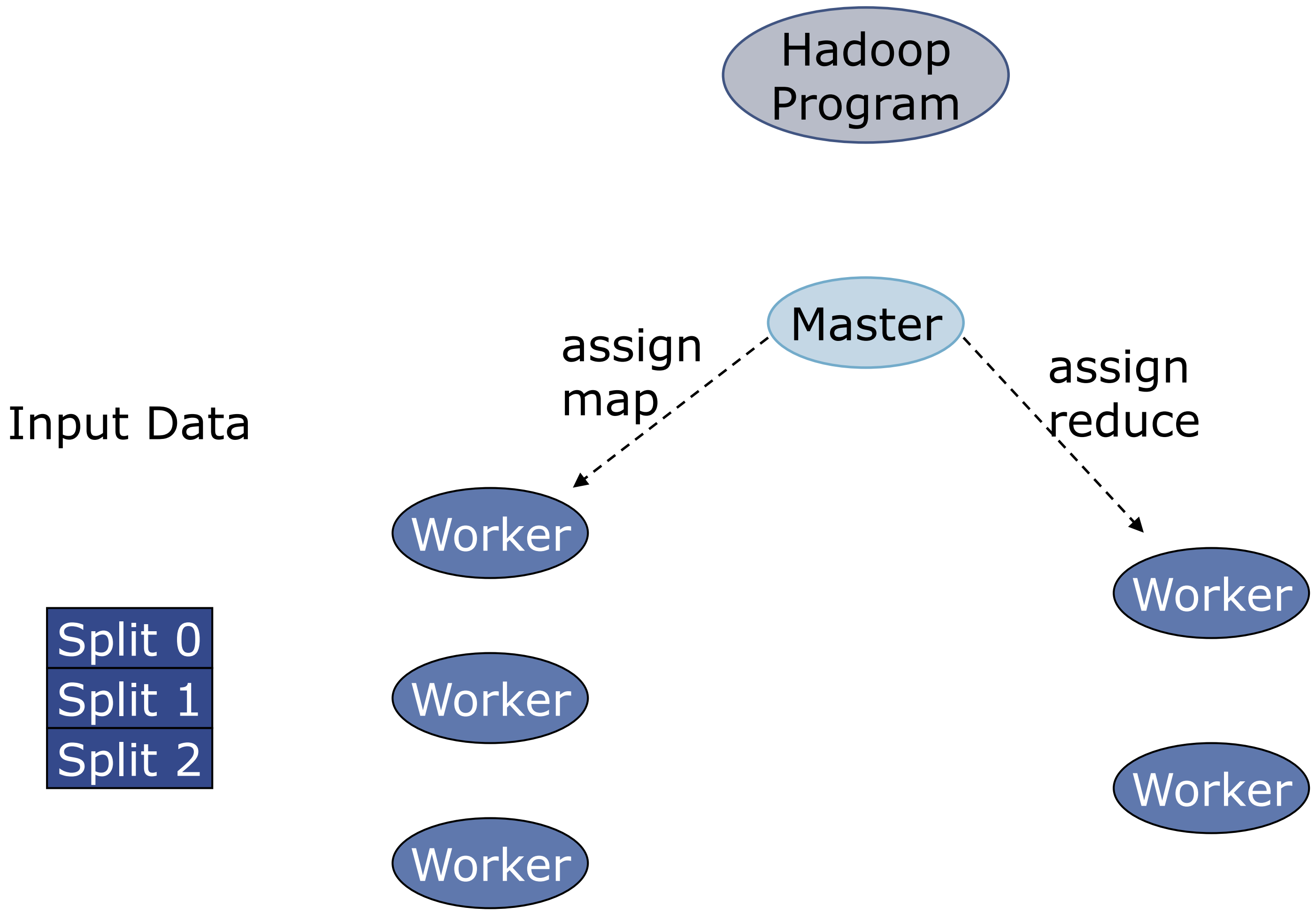
MapReduce

Hadoop Program

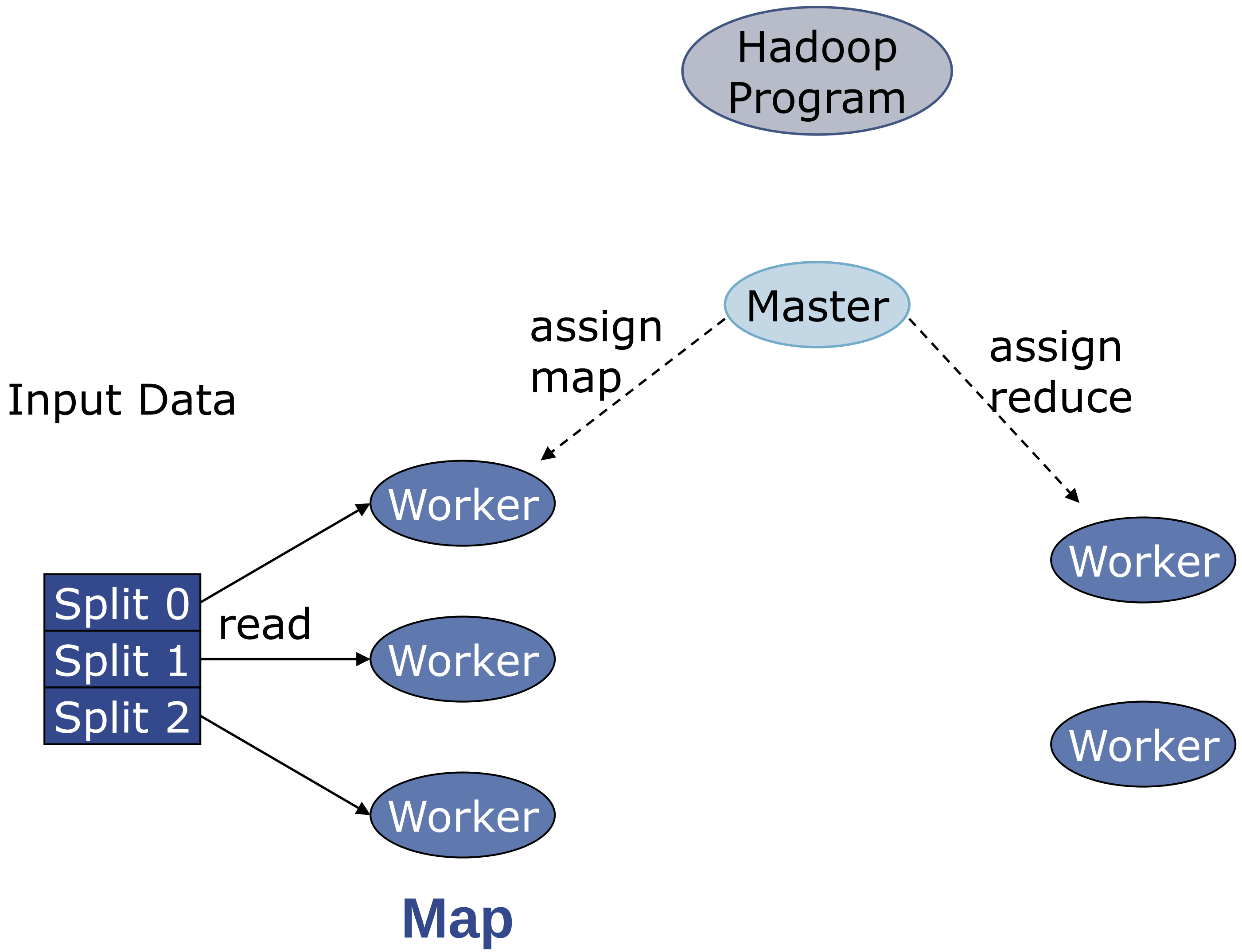
MapReduce


Hadoop Program
fork
fork
fork
Master
Worker
Worker
Worker
Worker
Worker

MapReduce

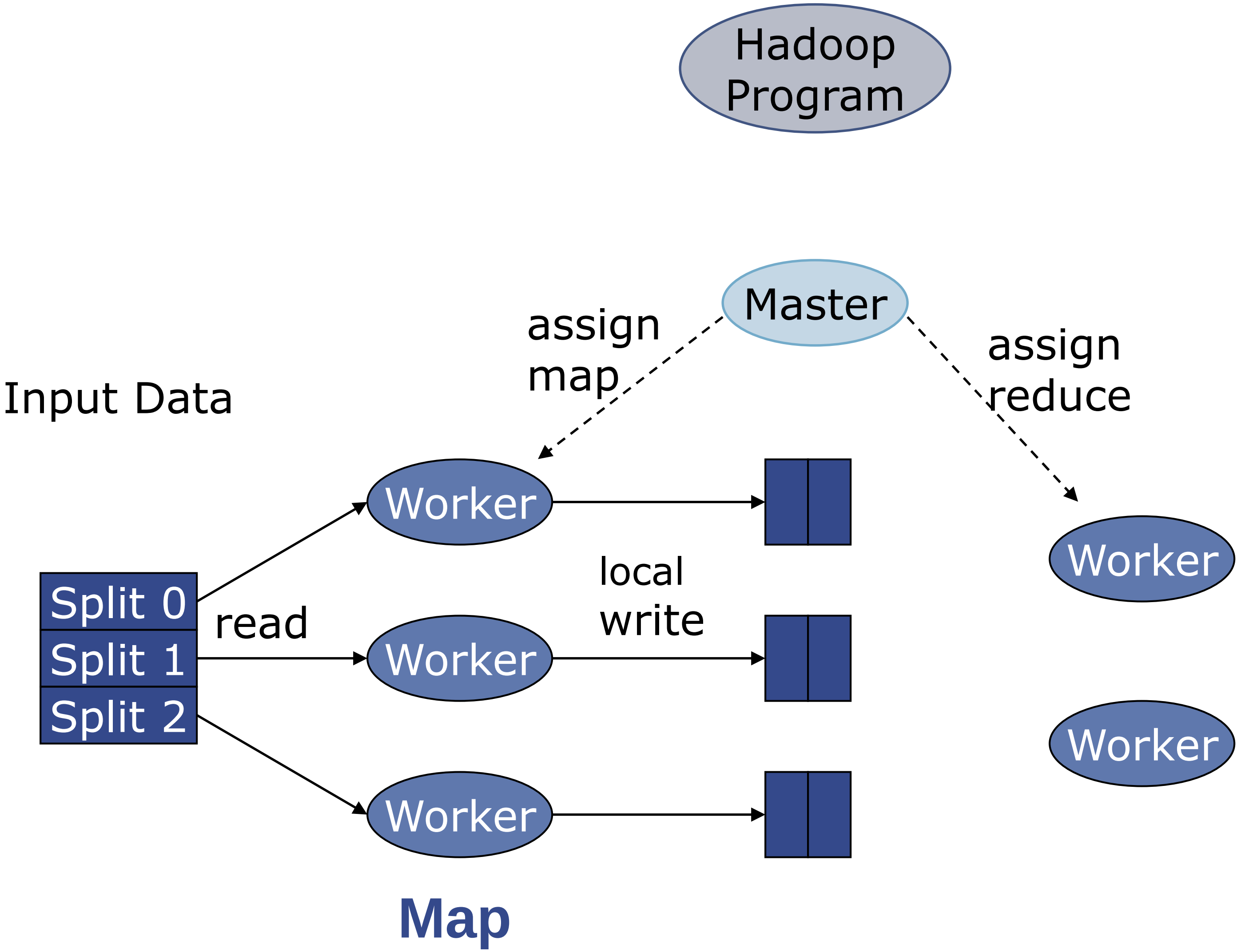

Hadoop Program
Master
assign map
assign reduce
Input Data
Split 0
Split 1
Split 2
Worker
Worker
Worker
Worker
Worker

MapReduce



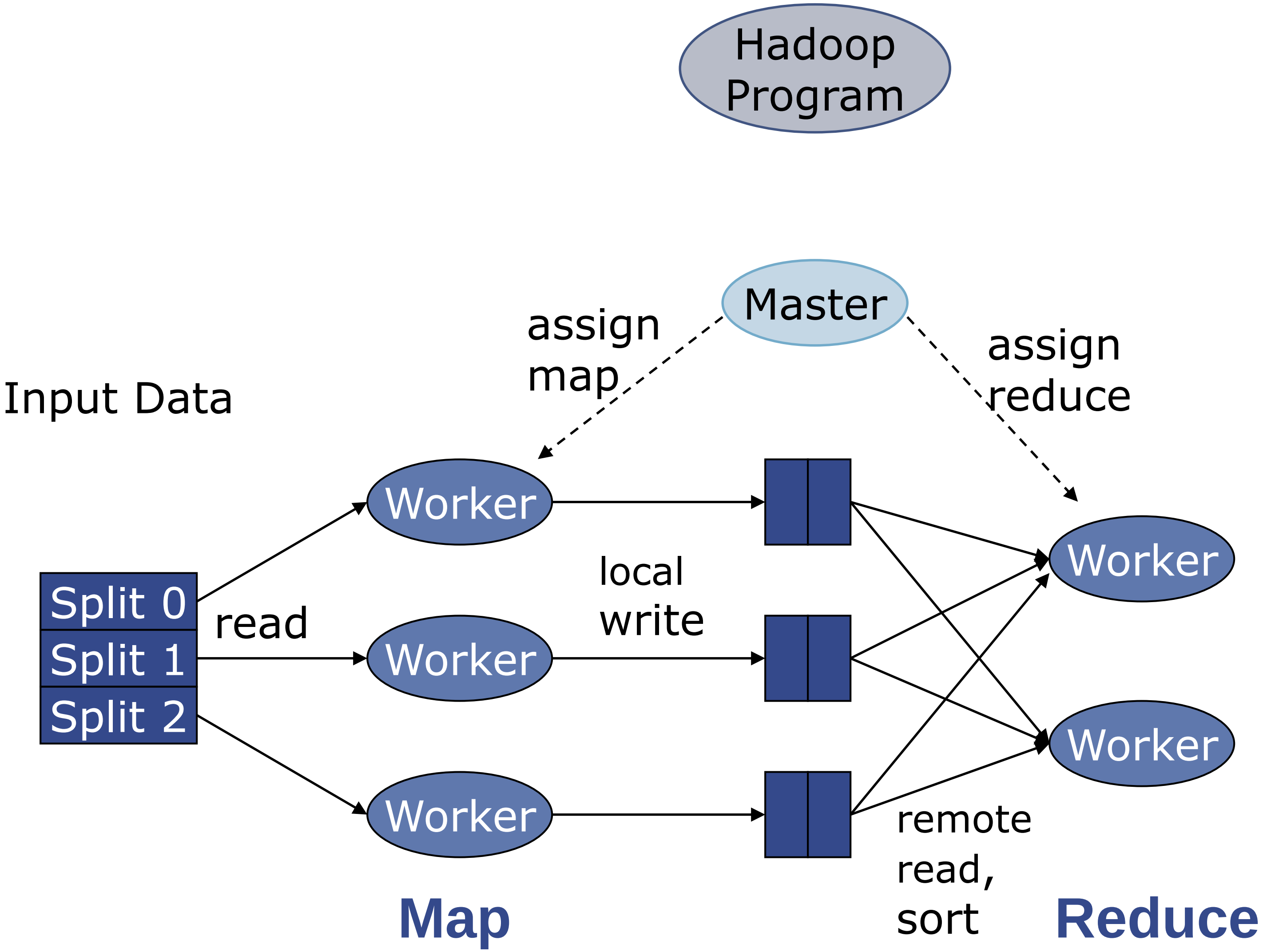
Hadoop Program
Master
assign map
assign reduce
Input Data
Split 0
Split 1
Split 2
read
Worker
Worker
Worker
Worker
Worker
Map

MapReduce

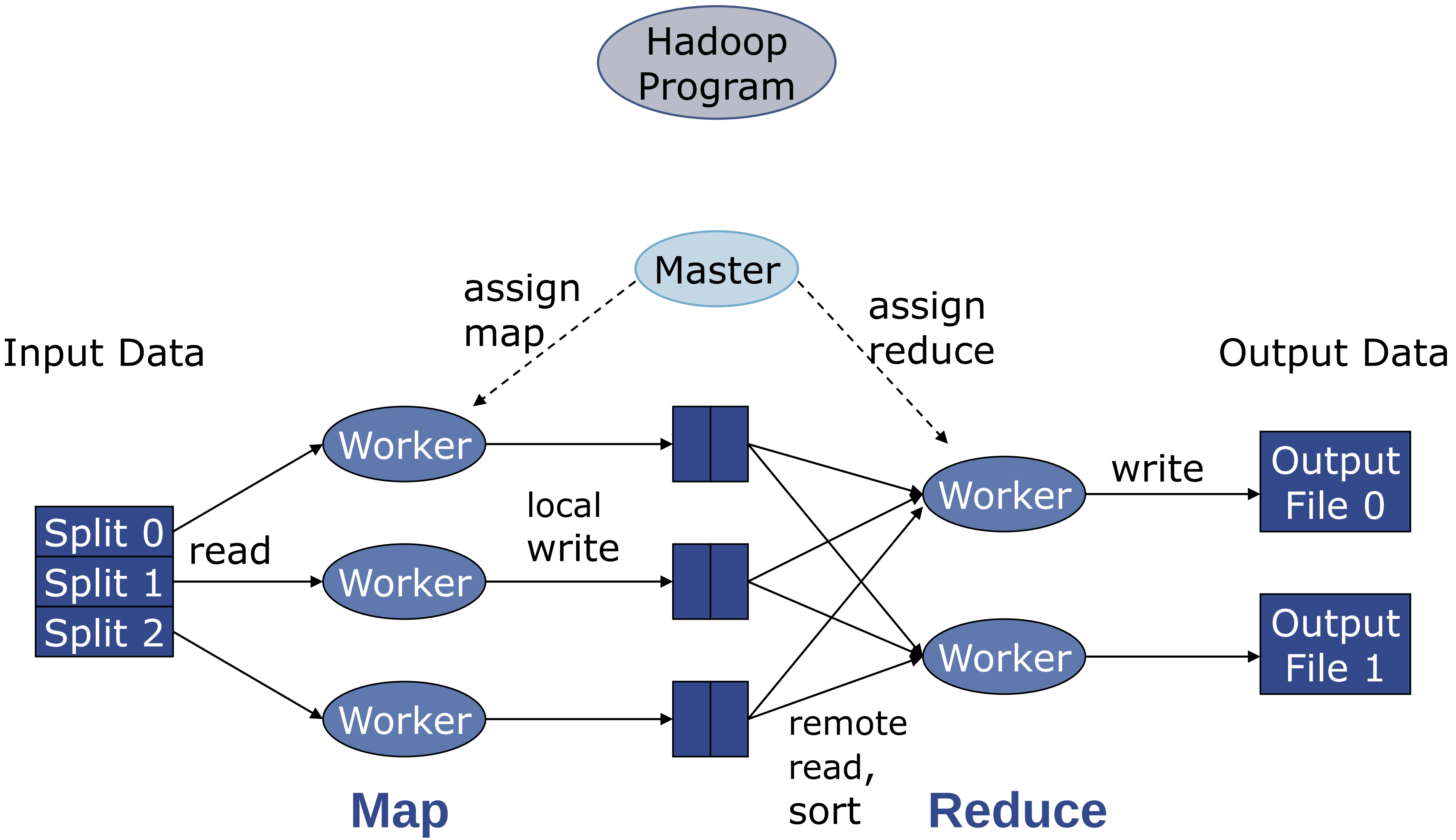


Hadoop Program
Master
assign map
assign reduce
Input Data
Split 0
Split 1
Split 2
read
Worker
Worker
Worker
local write
Worker
Worker
Map

MapReduce

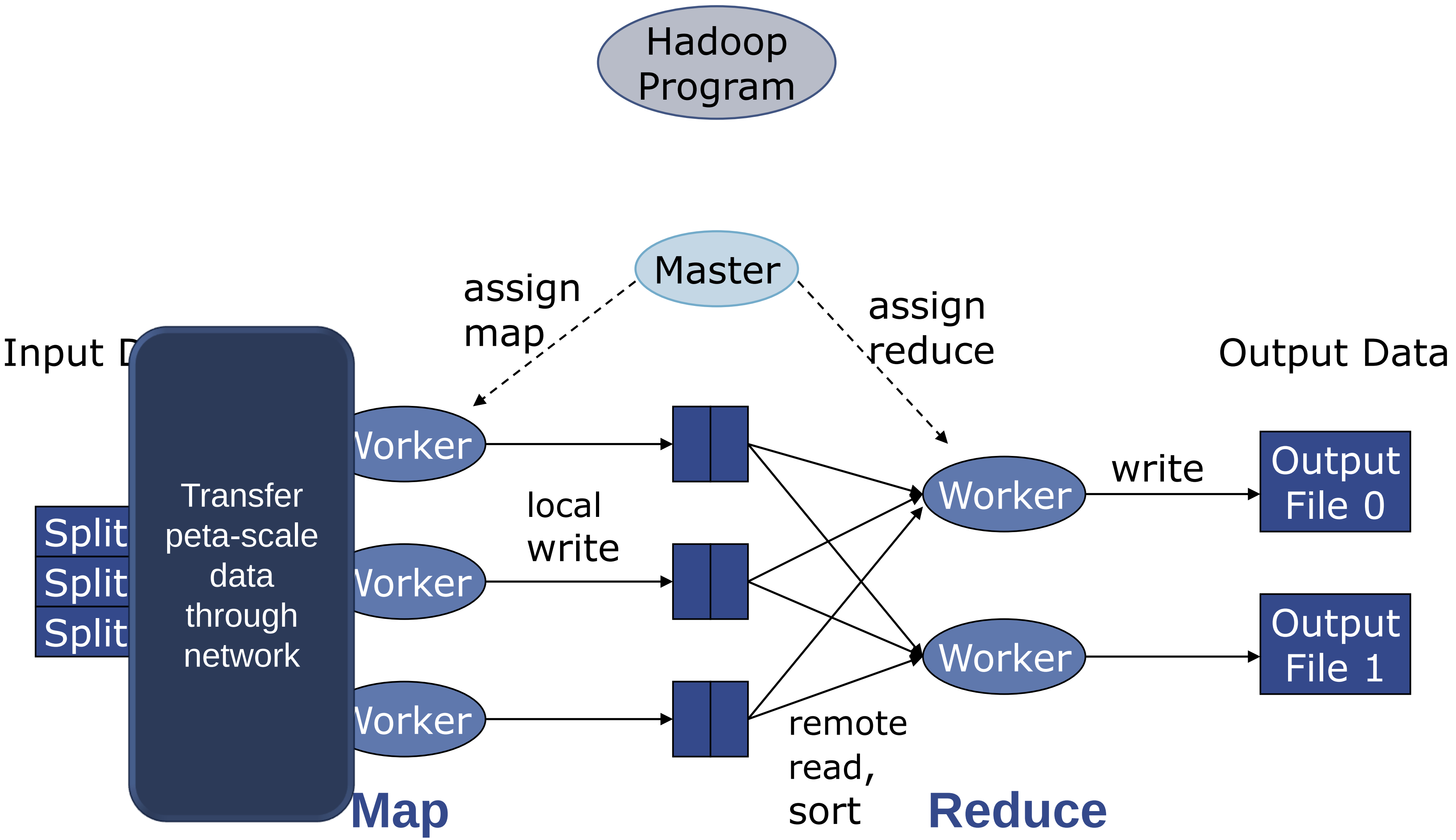

Hadoop
Program
Master
assign
map
assign
reduce
Input Data
Worker
Worker
Worker
Split 0
Split 1
Split 2
read
local
write
Worker
Worker
remote
read,
sort
Map
Reduce

MapReduce



Hadoop Program
Master
assign map
assign reduce
Input Data
Output Data
Split 0
Split 1
Split 2
read
Worker
Worker
Worker
local write
Worker
Worker
write
Output File 0
Output File 1
remote read, sort
Map
Reduce

MapReduce

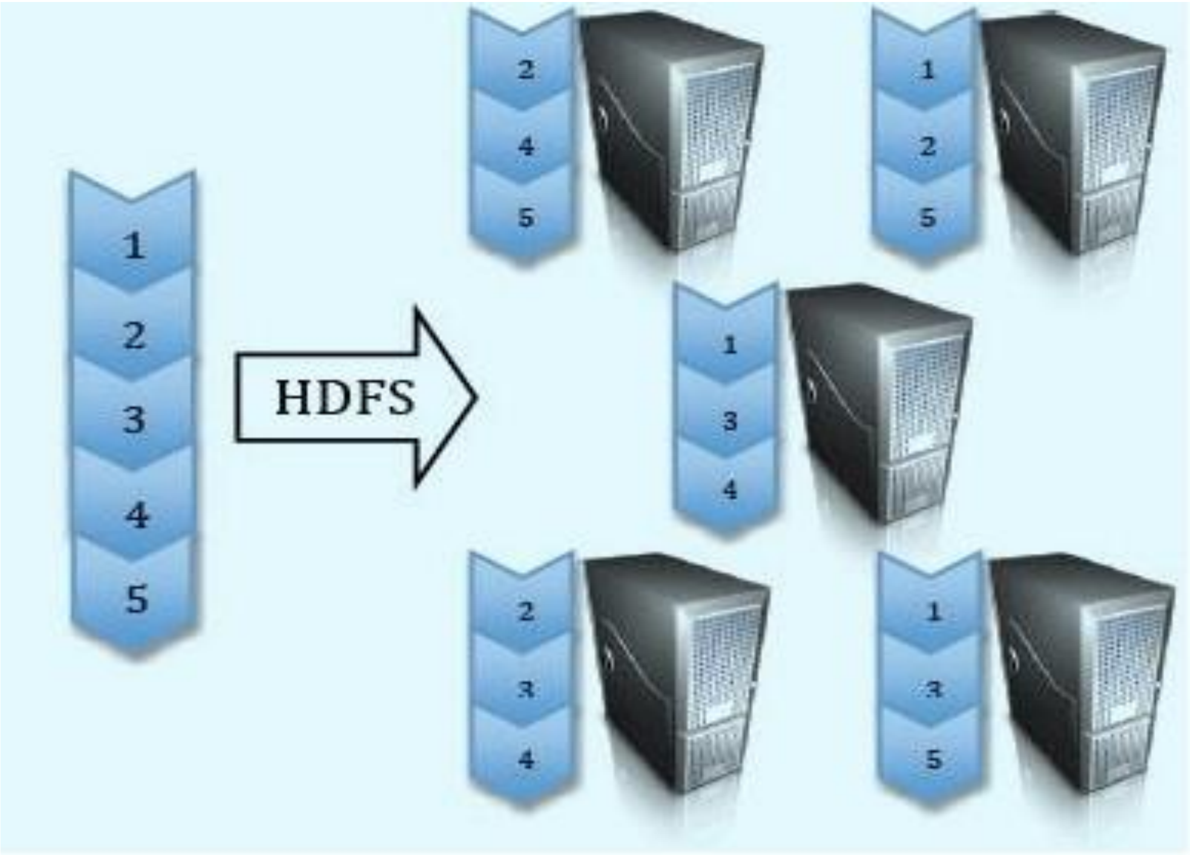


Hadoop Program
Master
assign map
assign reduce
Input D
Split
Split
Split
Transfer peta-scale data through network
Worker
Worker
Worker
local write
Worker
Worker
write
remote read, sort
Output Data
Output File 0
Output File 1
Map
Reduce

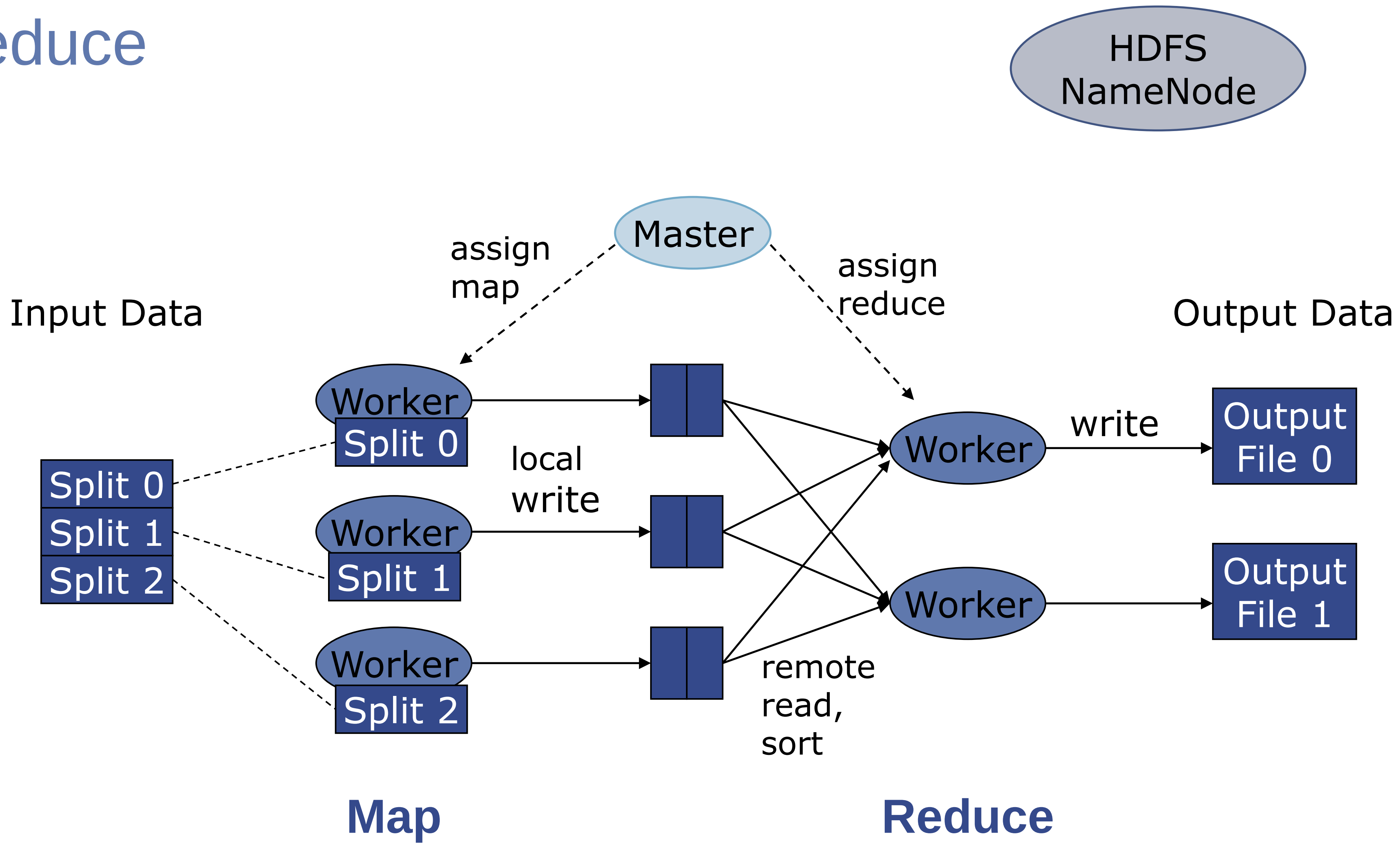
GFS/HDFS

- This is not a GFS/HDFS presentation!

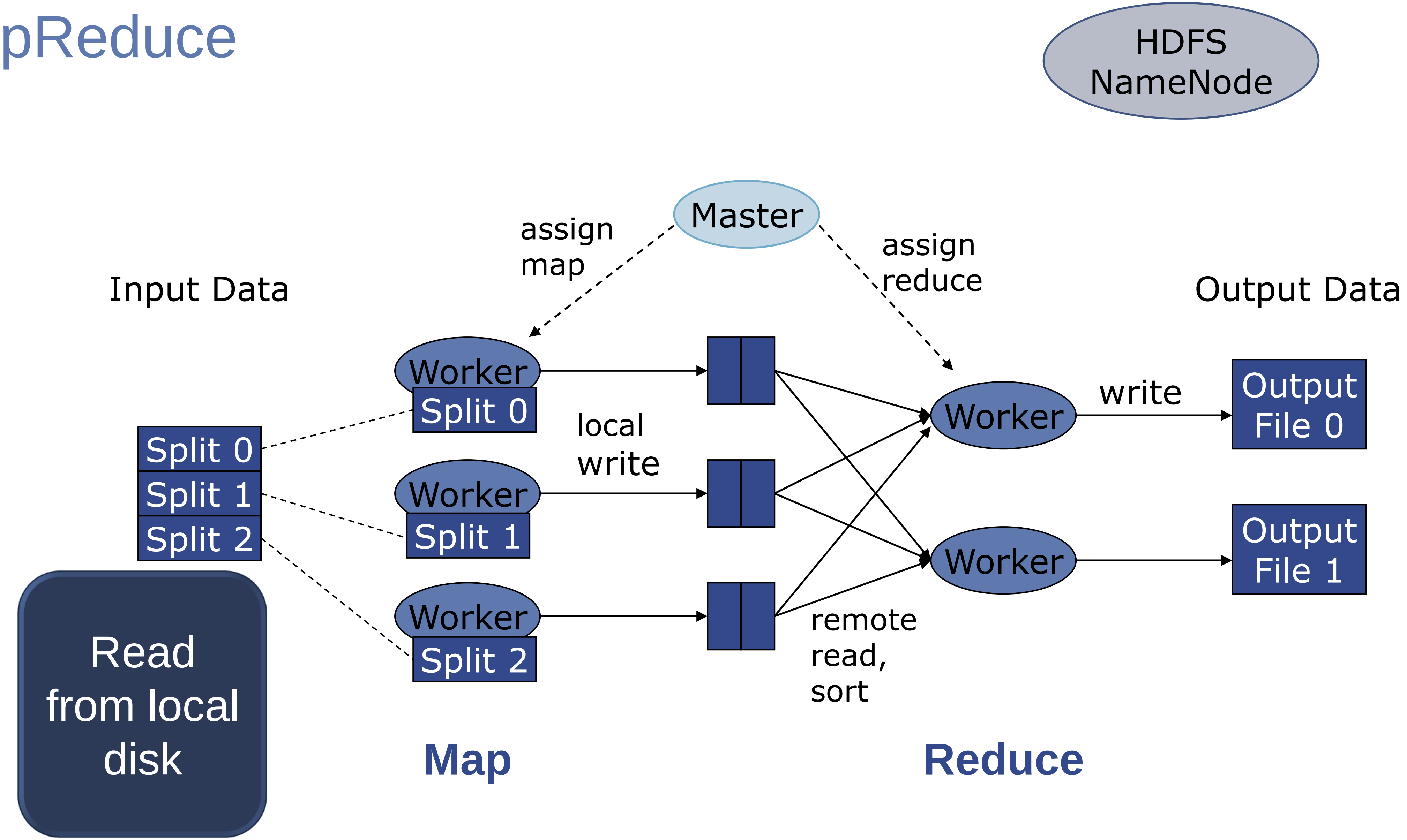
- A few concepts are key to MapReduce though:
  - Google File System (GFS) and Hadoop Distributed File System (HDFS) are essentially distributed filesystems
  - Are fault tolerant through replication
  - Allows data to be local to computation



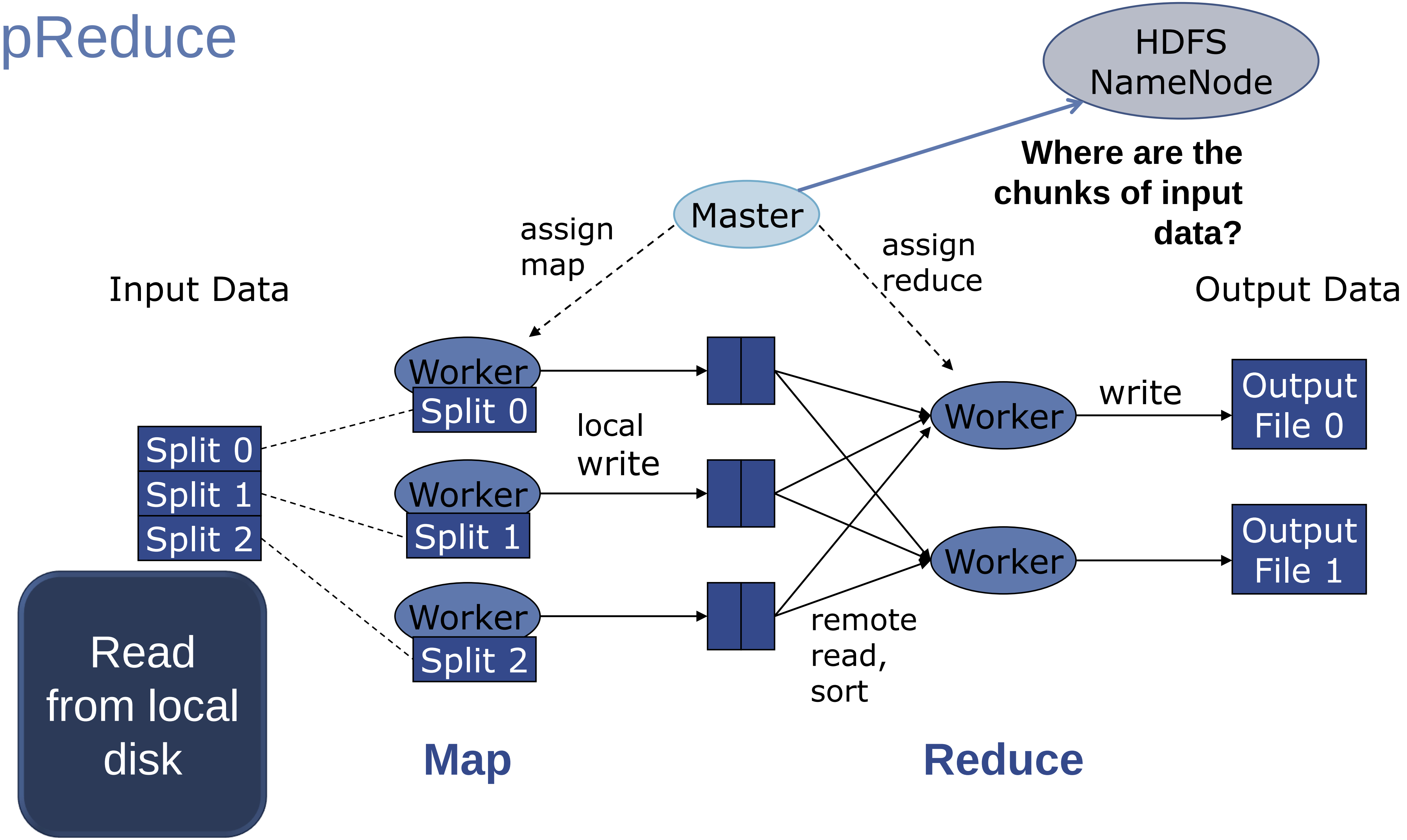
MapReduce


HDFS
NameNode
Master
assign
map
assign
reduce
Input Data
Output Data
Worker
Split 0
Split 0
Split 1
Split 2
local
write
Worker
Split 1
Worker
Split 2
Worker
write
Output
File 0
Worker
Output
File 1
remote
read,
sort
Map
Reduce

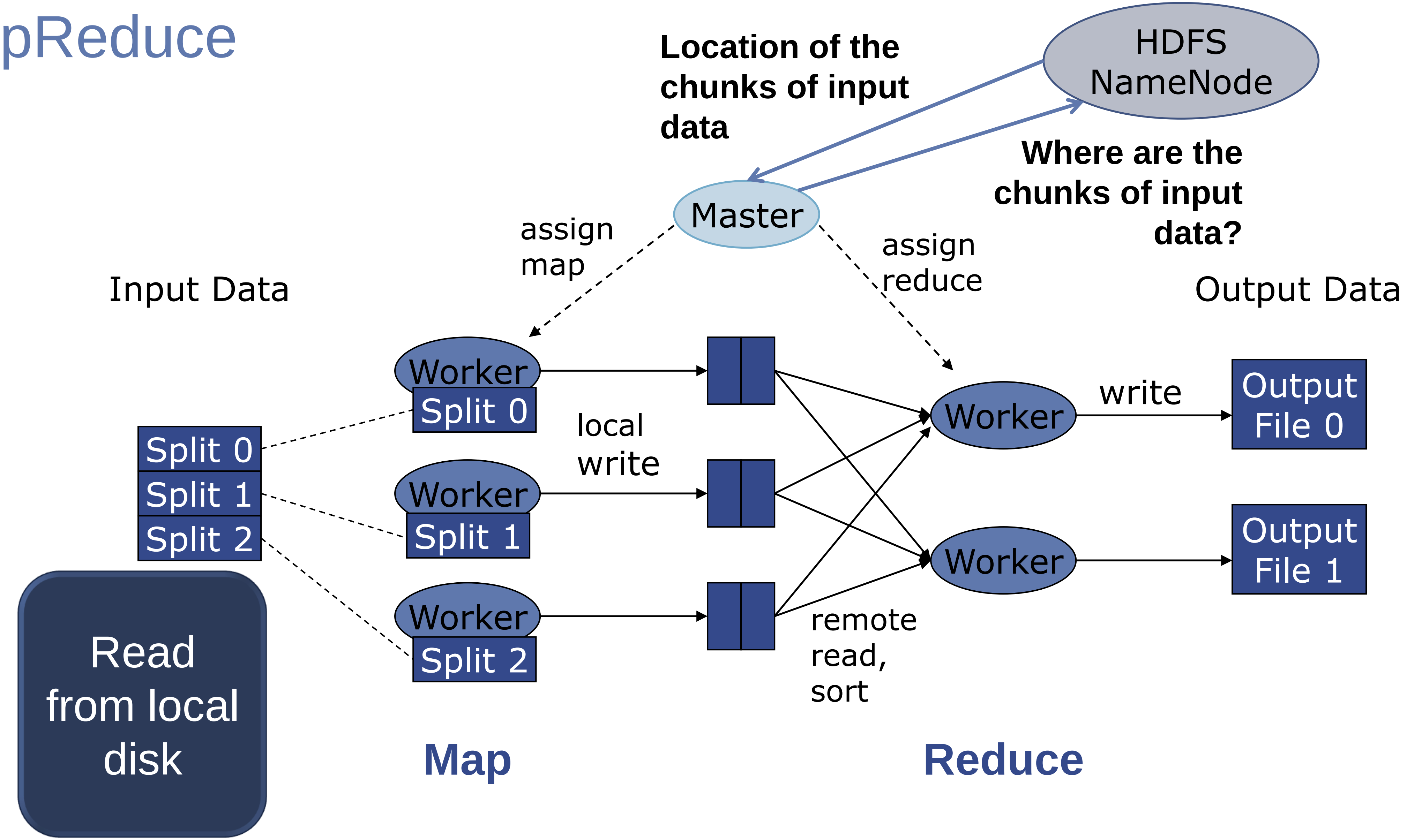
MapReduce


HDFS
NameNode
Master
assign
map
assign
reduce
Input Data
Output Data
Split 0
Split 1
Split 2
Worker
Split 0
Worker
Split 1
Worker
Split 2
local
write
Worker
Worker
write
Output
File 0
Output
File 1
remote
read,
sort
Read
from local
disk
Map
Reduce

MapReduce


HDFS NameNode
Where are the chunks of input data?
Master
assign map
assign reduce
Input Data
Output Data
Worker
Split 0
Split 0
Split 1
Split 2
Worker
Split 1
Worker
Split 2
local write
Worker
Worker
write
Output File 0
Output File 1
remote read, sort
Read from local disk
Map
Reduce

MapReduce


Location of the chunks of input data
HDFS NameNode
Where are the chunks of input data?
Master
assign map
assign reduce
Input Data
Output Data
Worker
Split 0
Worker
Split 1
Worker
Split 2
Split 0
Split 1
Split 2
local write
Worker
write
Output File 0
Worker
Output File 1
remote read, sort
Read from local disk
Map
Reduce

Master

- Responsible for scheduling & managing jobs
- Scheduled computation should be close to the data if possible
  - Bandwidth is expensive! (and slow)
  - This relies on a Distributed File System (GFS / HDFS)!
- If a task fails to report progress (such as reading input, writing output, etc), crashes, the machine goes down, etc, it is assumed to be stuck, and is killed, and the step is re-launched (with the same input)
- The Master is handled by the framework, no user code is necessary

Master Cont.

- HDFS can replicate data to be local if necessary for scheduling
- Because our nodes are (or at least should be) deterministic
  - The Master can restart failed nodes
    - Nodes should have no side effects!
  - If a node is the last step, and is completing slowly, the master can launch a second copy of that node
    - This can be due to hardware isuses, network issues, etc.
    - First one to complete wins, then any other runs are killed

Writables

- Are types that can be serialized / deserialized to a stream
- Are required to be input/output classes, as the framework will serialize your data before writing it to disk
- User can implement this interface, and use their own types for their input/output/intermediate values
- There are default for basic values, like Strings, Integers, Longs, etc.
- Can also handle store, such as arrays, maps, etc.
- Your application needs at least six writables
  - 2 for your input
  - 2 for your intermediate values (Map <-> Reduce)
  - 2 for your output

Basic Concepts

- All data is represented in key value pairs of an arbitrary type
- Data is read in from a file or list of files, from HDFS
- Data is chunked based on an input split
  - A typical chunk is 64MB (more or less can be configured depending on your use case)
- Mappers read in a **chunk** of data
- Mappers emit (write out) a set of data, typically derived from its input
- Intermediate data (the output of the mappers) is split to a number of reducers
- Reducers receive each key of data, along with **ALL** of the values associated with it (this means each key must always be sent to the same reducer)
  - Essentially, <key, set<value>>
- Reducers emit a set of data, typically reduced from its input which is written to disk

Locality Optimization

- **Master scheduling policy:**
  - Asks GFS for locations of replicas of input file blocks
  - Map tasks scheduled so GFS input block replica are on same machine or same rack
- Effect: Thousands of machines read input at local disk speed
  - Eliminate network bottleneck!

Fault tolerance: Handled via re-execution

- On worker failure:
  - Detect failure via periodic heartbeats
  - Re-execute completed and in-progress *map* tasks
  - Task completion committed through master
- On master failure:
  - Single point of failure; Resume from Execution Log
- **Robust:** [Google’s experience] lost 1600 of 1800 machines, but finished fine

Refinement: Redundant Execution

- **Slow workers** significantly lengthen completion time
  - Other jobs consuming resources on machine
  - Bad disks with soft errors transfer data very slowly
  - Weird things: processor caches disabled (!!)
- **Solution**: spawn backup copies of tasks
  - Whichever one finishes first "wins"

Refinement: Skipping Bad Records

Map/Reduce functions sometimes fail for particular inputs

- Best solution is to debug & fix, but not always possible
- If master sees two failures for the same record:
  - Next worker is told to skip the record

Mapper Code (Java)

- Our input to our mapper is <LongWritable, Text>
- The key (the LongWritable) can be assumed to be the position in the document our input is in. This doesn't matter for this example.
- Our output is a bunch of <Text, LongWritable>. The key is the token, and the value is the count. This is always 1.
- For the purpose of this demonstration, just assume Text is a fancy String, and LongWritable is a fancy Long. In reality, they're just the Writable equivalents.

```java
public void map(LongWritable key, Text value, Context context) {
    String line = value.toString();
    for(String part : tokenizeString(line)) {
        context.write(new Text(part), new LongWritable(1));
    }
}
```

Reducer Code (Java)

- Our input is the output of our Mapper, a <Text, LongWritable> pair
- Our output is still a <Text,LongWritable>, but it reduces N inputs for token T, into one output <T, N>

```java
    public void reduce(Text key, Iterable<LongWritable> values, Context context) {
        long sum = 0;
        for (LongWritable val : values) {
            sum += val.get();
        }
        context.write(key, new LongWritable(sum));
    }
```

Combiner Code

- Do we need a combiner?
  - No, but it reduces bandwidth.

- Our reducer can actually be our combiner in this case though!

That’s it!

- All that is needed to run the above code is an extremely simple runner class.
  - Simply specifies which components to use, and your input/output directories

Let’s try it in python

[Lecture 02 - Big Data.ipynb]

Conclusion

- MapReduce provides a simple way to scale your application
- Scales out to more machines, rather than scaling up
- Effortlessly scale from a single machine to thousands
- Fault tolerant & High performance
- If you can fit your use case to its paradigm, scaling is handled by the framework

References:

- Sides by Dr. R. T. Uthayasanker based on slides by Dr. Srinath Perera
- Slides by Prof. Ruoming Jin
- Slides by Prof. Cristiana Amza
- Slides by Jeffrey Dean and Sanjay Ghemawa