Parallel Programming Principle and Practice

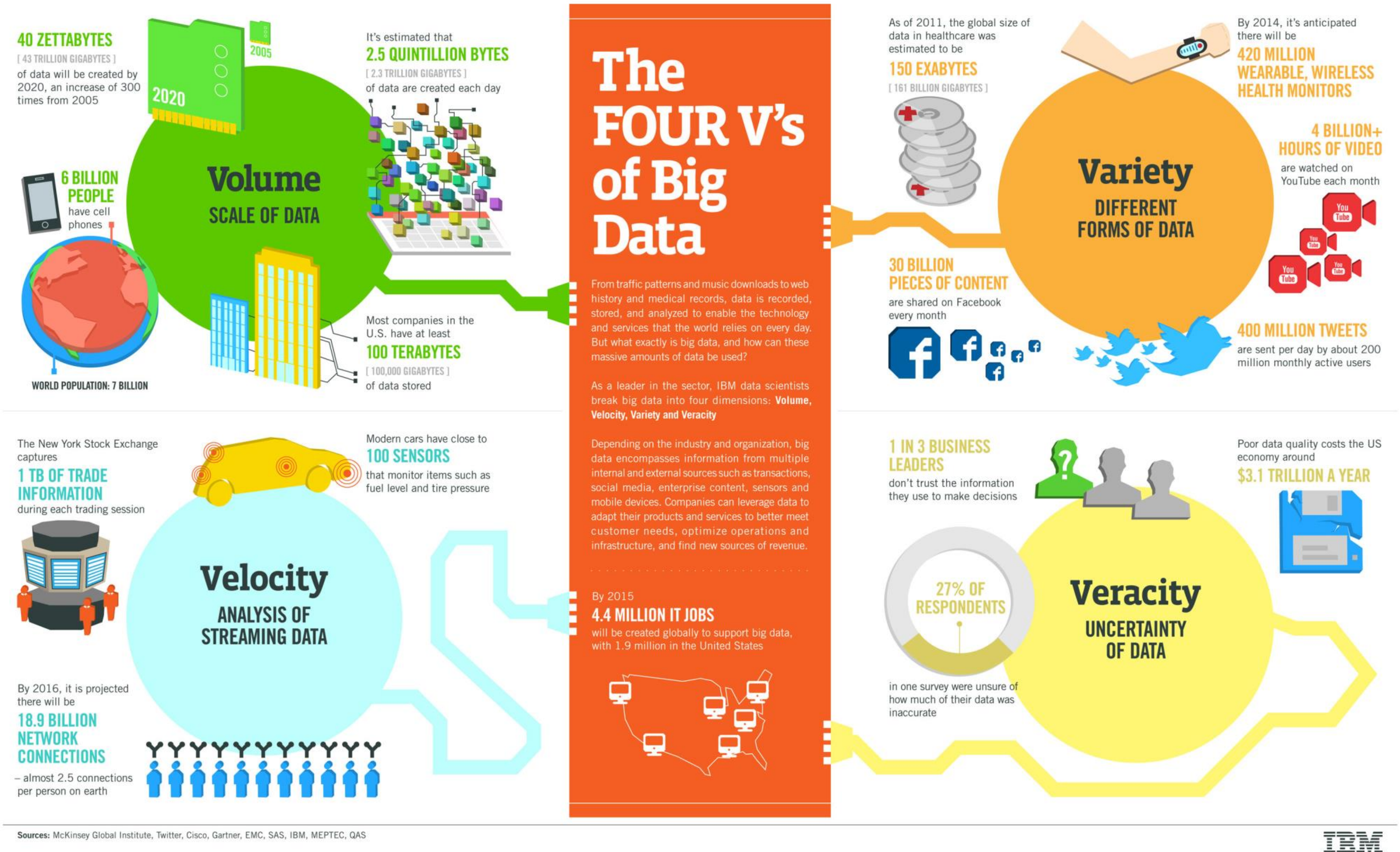
# Lecture 10 — Big Data Processing with MapReduce

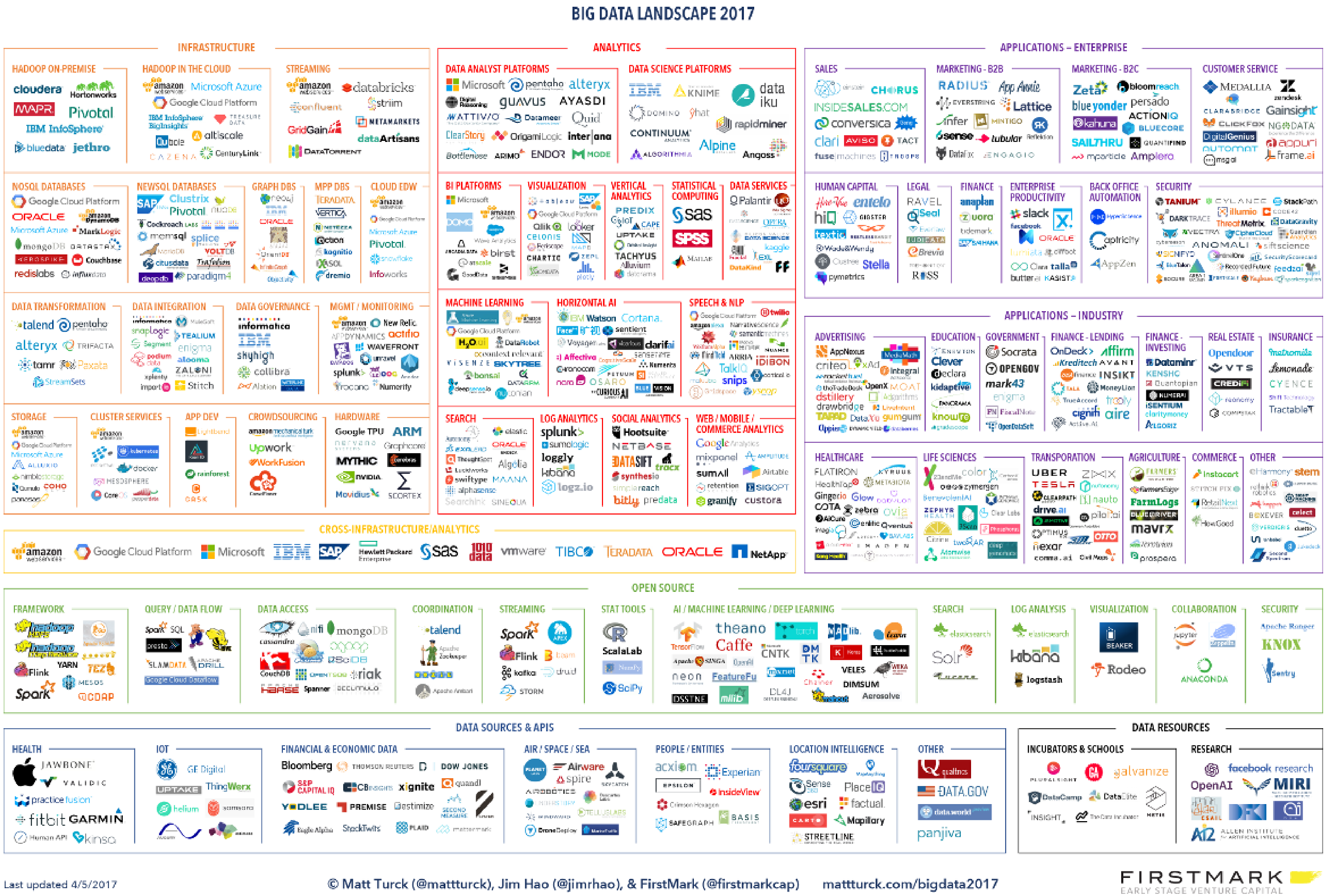
# Outline

- MapReduce Programming Model
- MapReduce Examples
- Hadoop

# Incredible Things That Happen Every Minute On The Internet

# The Four V's of Big Data




BIG DATA LANDSCAPE 2017
INFRASTRUCTURE
HADOOP ON-PREMISE
HADOOP IN THE CLOUD
STREAMING
NOSQL DATABASES
NEWSQL DATABASES
GRAPH DBS
MPP DBS
CLOUD EDW
DATA TRANSFORMATION
DATA INTEGRATION
DATA GOVERNANCE
MGMT / MONITORING
STORAGE
CLUSTER SERVICES
APP DEV
CROWDSOURCING
HARDWARE
ANALYTICS
DATA ANALYST PLATFORMS
DATA SCIENCE PLATFORMS
BI PLATFORMS
VISUALIZATION
VERTICAL ANALYTICS
STATISTICAL COMPUTING
DATA SERVICES
MACHINE LEARNING
HORIZONTAL AI
SPEECH & NLP
SEARCH
LOG ANALYTICS
SOCIAL ANALYTICS
WEB / MOBILE / COMMERCE ANALYTICS
APPLICATIONS – ENTERPRISE
SALES
MARKETING - B2B
MARKETING - B2C
CUSTOMER SERVICE
HUMAN CAPITAL
LEGAL
FINANCE
ENTERPRISE PRODUCTIVITY
BACK OFFICE AUTOMATION
SECURITY
APPLICATIONS – INDUSTRY
ADVERTISING
EDUCATION
GOVERNMENT
FINANCE - LENDING
FINANCE - INVESTING
REAL ESTATE
INSURANCE
HEALTHCARE
LIFE SCIENCES
TRANSPORTATION
AGRICULTURE
COMMERCE
OTHER
CROSS-INFRASTRUCTURE/ANALYTICS
OPEN SOURCE
FRAMEWORK
QUERY / DATA FLOW
DATA ACCESS
COORDINATION
STREAMING
STAT TOOLS
AI / MACHINE LEARNING / DEEP LEARNING
SEARCH
LOG ANALYSIS
VISUALIZATION
COLLABORATION
SECURITY
DATA SOURCES & APIS
HEALTH
IOT
FINANCIAL & ECONOMIC DATA
AIR / SPACE / SEA
PEOPLE / ENTITIES
LOCATION INTELLIGENCE
OTHER
DATA RESOURCES
INCUBATORS & SCHOOLS
RESEARCH
Last updated 4/5/2017
© Matt Turck (@mattturck), Jim Hao (@jimrhao), & FirstMark (@firstmarkcap)
mattturck.com/bigdata2017
FIRSTMARK
EARLY STAGE VENTURE CAPITAL

# Motivation: Large Scale Data Processing

- Want to process lots of data ( >1TB)
- Want to parallelize across hundreds/thousands of CPUs
- … Want to make this easy

# MapReduce

- "A simple and powerful interface that enables automatic parallelization and distribution of large-scale computations, combined with an implementation of this interface that achieves high performance on large clusters of commodity PCs."

- More simply, MapReduce is
  - A parallel programming model and associated implementation

*Dean and Ghermawat, "MapReduce: Simplified Data Processing on Large Clusters", Google Inc.*

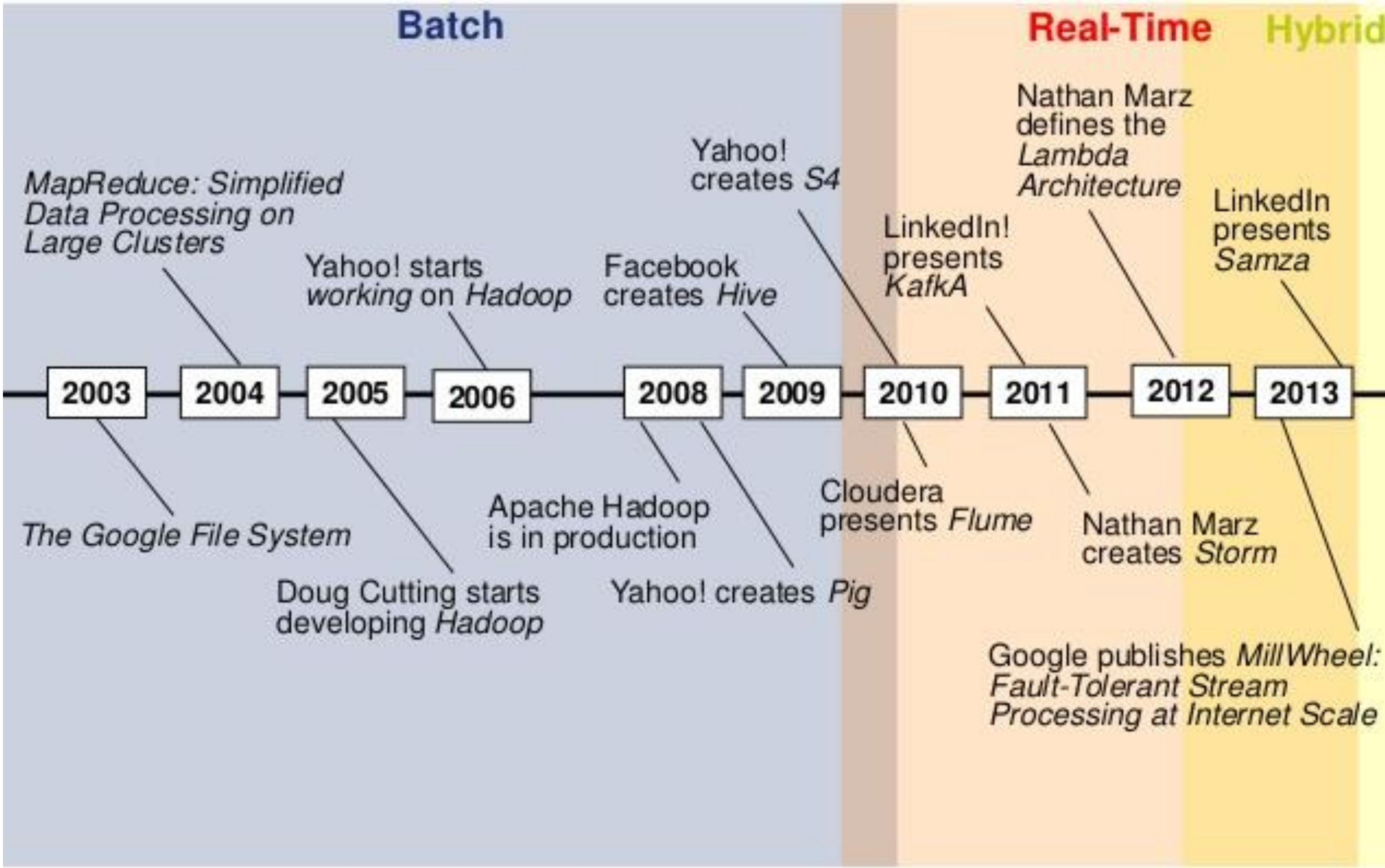
# A Brief History

MapReduce is a new use of an old idea in Computer Science

- Map:  Apply a function to every object in a list
  - Each object is independent
    - Order is unimportant
    - Maps can be done in parallel
  - The function produces a result
- Reduce:  Combine the results to produce a final result

You may have seen this in a Lisp or functional programming course

# 10 Years of Big Data Processing Technologies



# Some MapReduce Terminology

- *Job* – A “full program” - an execution of a Mapper and Reducer across a data set
- *Task* – An execution of a Mapper or a Reducer on a slice of data
  - a.k.a. Task-In-Progress (TIP)
- Task Attempt – A particular instance of an attempt to execute a task on a machine

# Terminology Example

- Running "Word Count" across 20 files is one *job*
- 20 files to be mapped imply 20 *map tasks* + some number of *reduce tasks*
- At least 20 map *task attempts* will be performed… more if a machine crashes, etc.

# Task Attempts

- A particular task will be attempted at least once, possibly more times if it crashes
  - If the same input causes crashes over and over, that input will eventually be abandoned
- Multiple attempts at one task may occur in parallel with speculative execution turned on
  - Task ID from *TaskInProgress* is not a unique identifier

# MapReduce Programming Model

- Process data using special map() and reduce() functions
  - The map() function is called on every item in the input and emits a series of intermediate key/value pairs
  - All values associated with a given key are grouped together
  - The reduce() function is called on every unique key, and its value list, and emits a value that is added to the output

# map

- Records from the data source (lines out of files, rows of a database, etc) are fed into the map function as key*value pairs: e.g., (filename, line)
- map() produces one or more *intermediate* values along with an output key from the input

```
– map  (in_key, in_value) ->
    (out_key, intermediate_value) list
```

# reduce

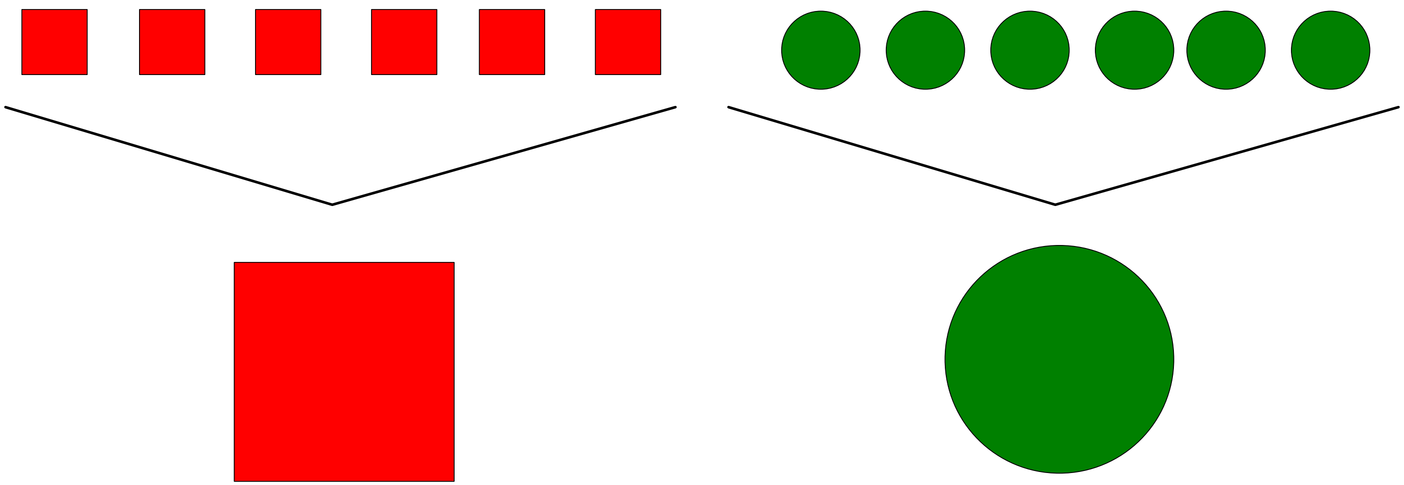
- After the map phase is over, all the intermediate values for a given output key are combined together into a list
- reduce() combines those intermediate values into one or more *final values* for that same output key

  - `reduce (out_key, intermediate_value list) -> out_value list`

# reduce

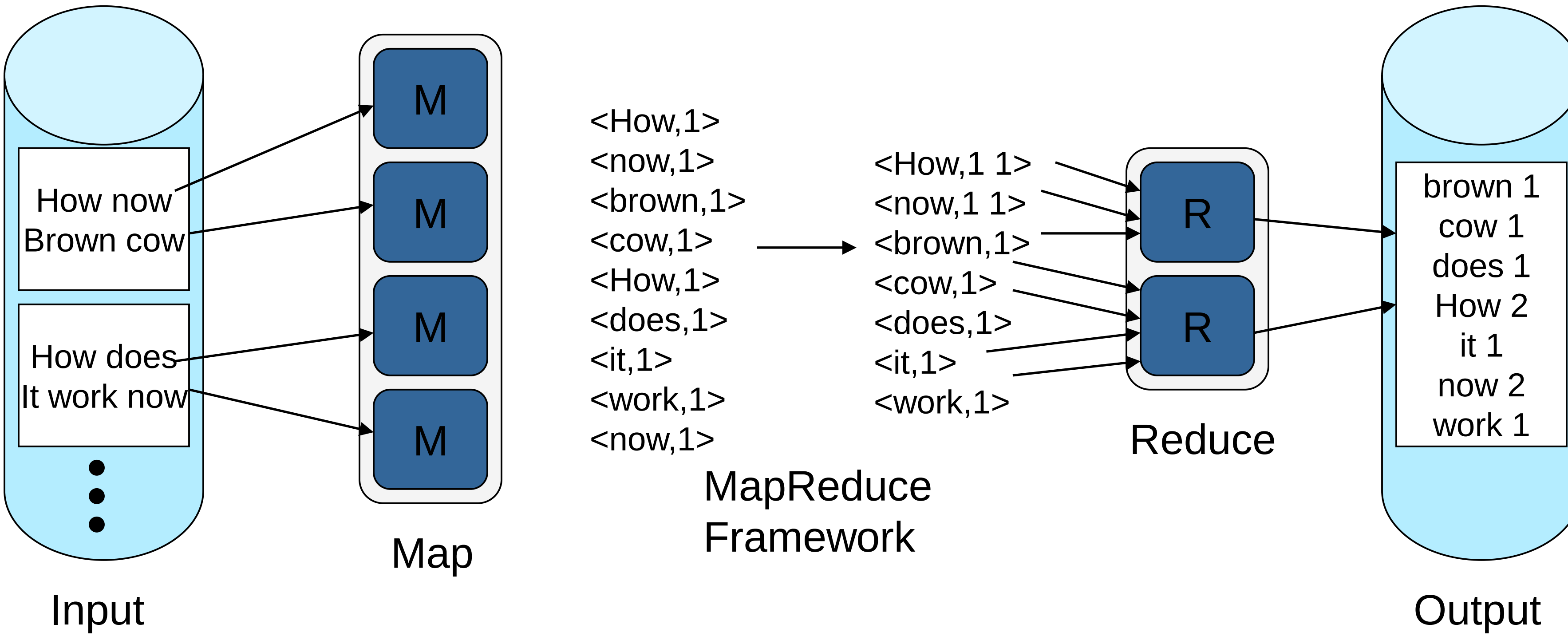
```
reduce (out_key, intermediate_value list) ->
        out_value list
```

initial



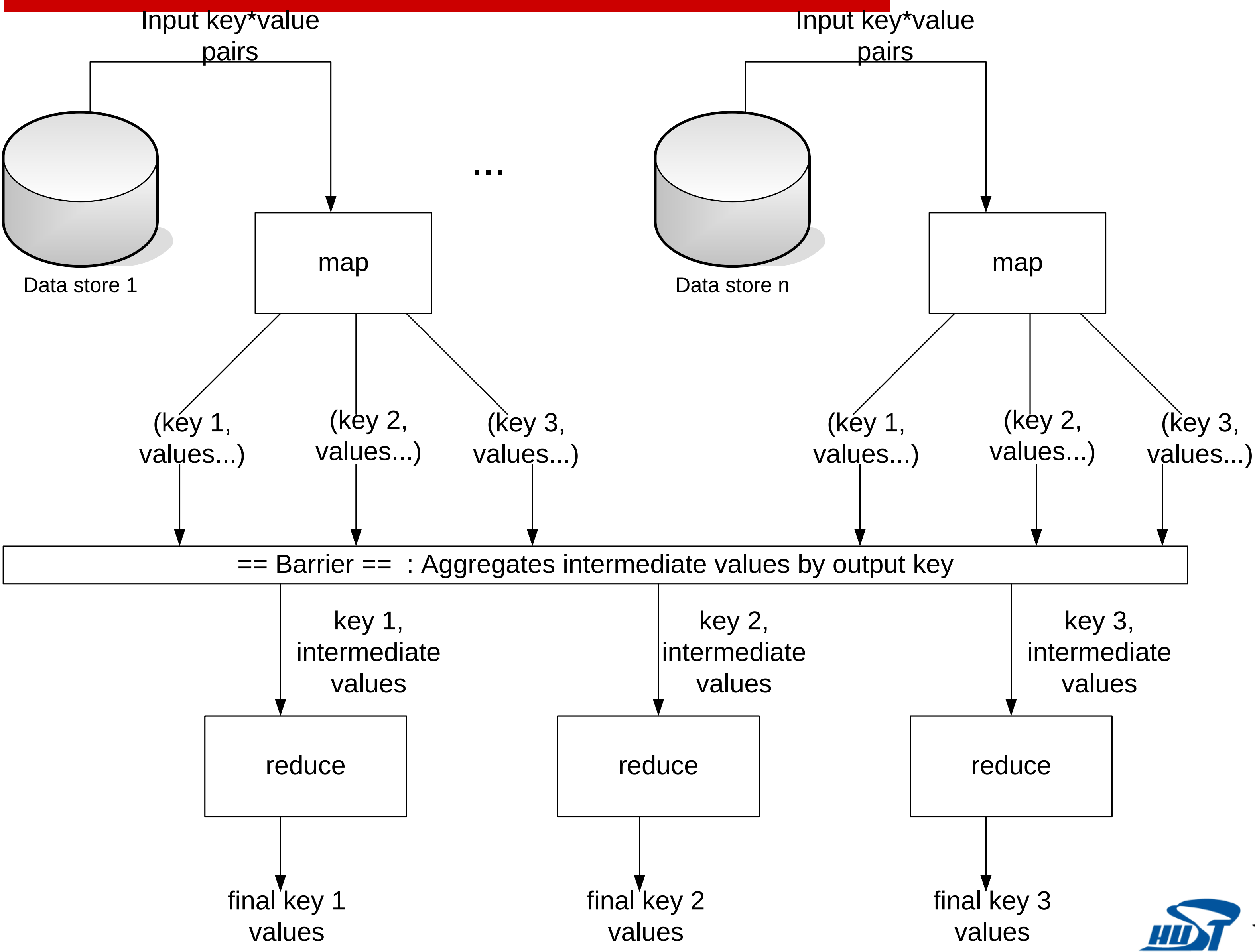
returned

# MapReduce Programming Model

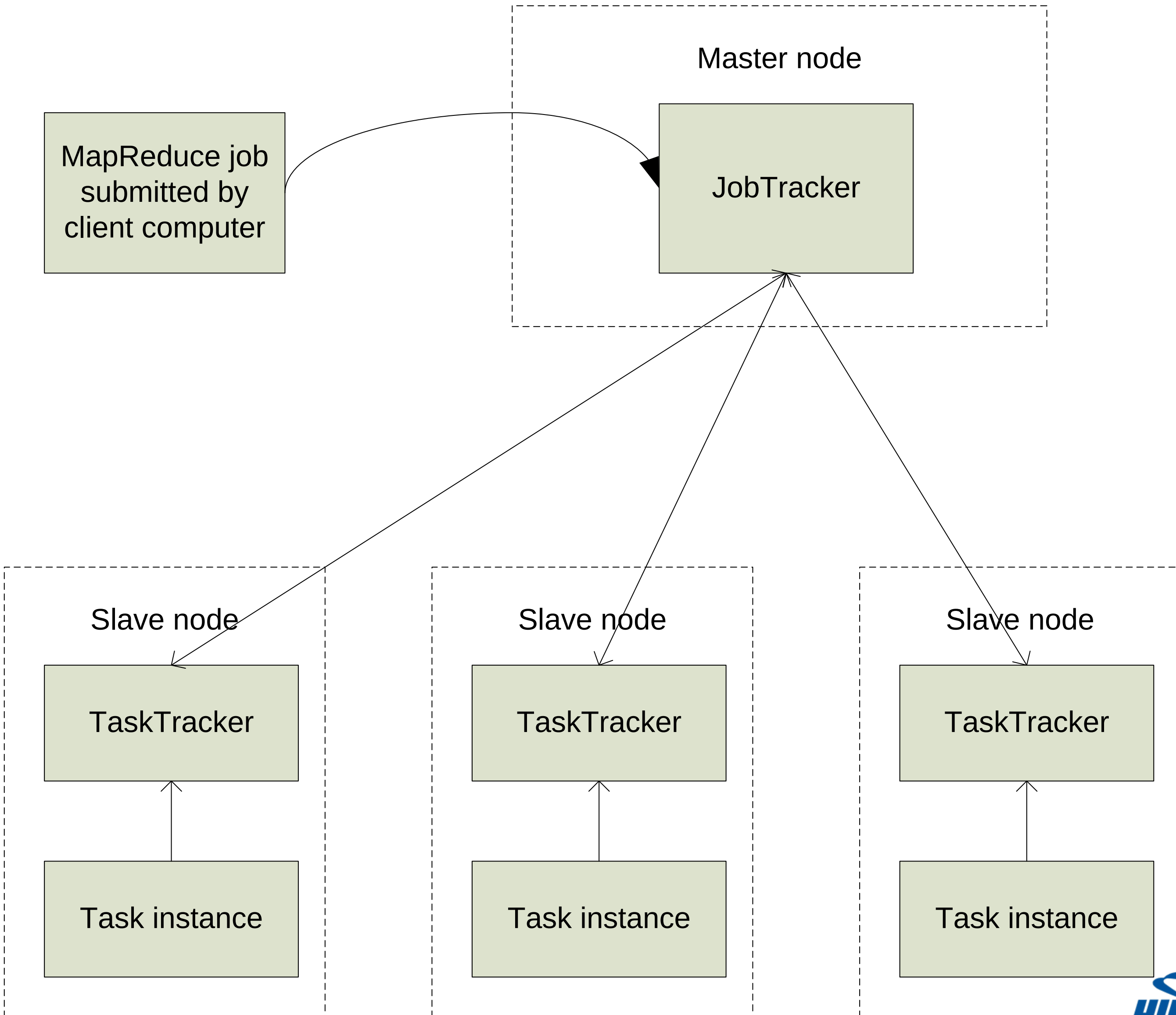



- More formally,
  - Map(k1,v1) --> list(k2,v2)
  - Reduce(k2, list(v2)) --> list(v2)

# MapReduce Architecture



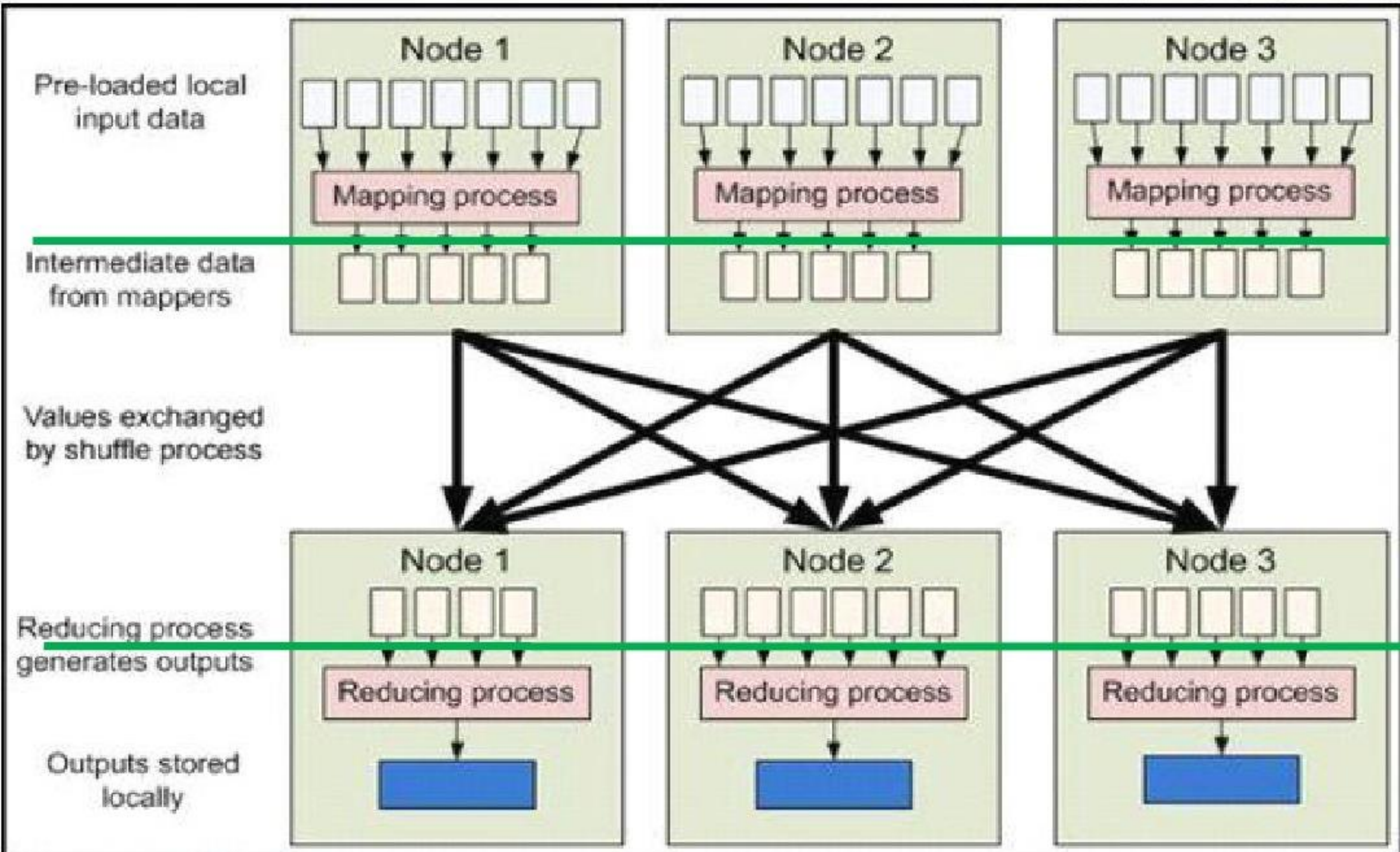
# MapReduce: High Level



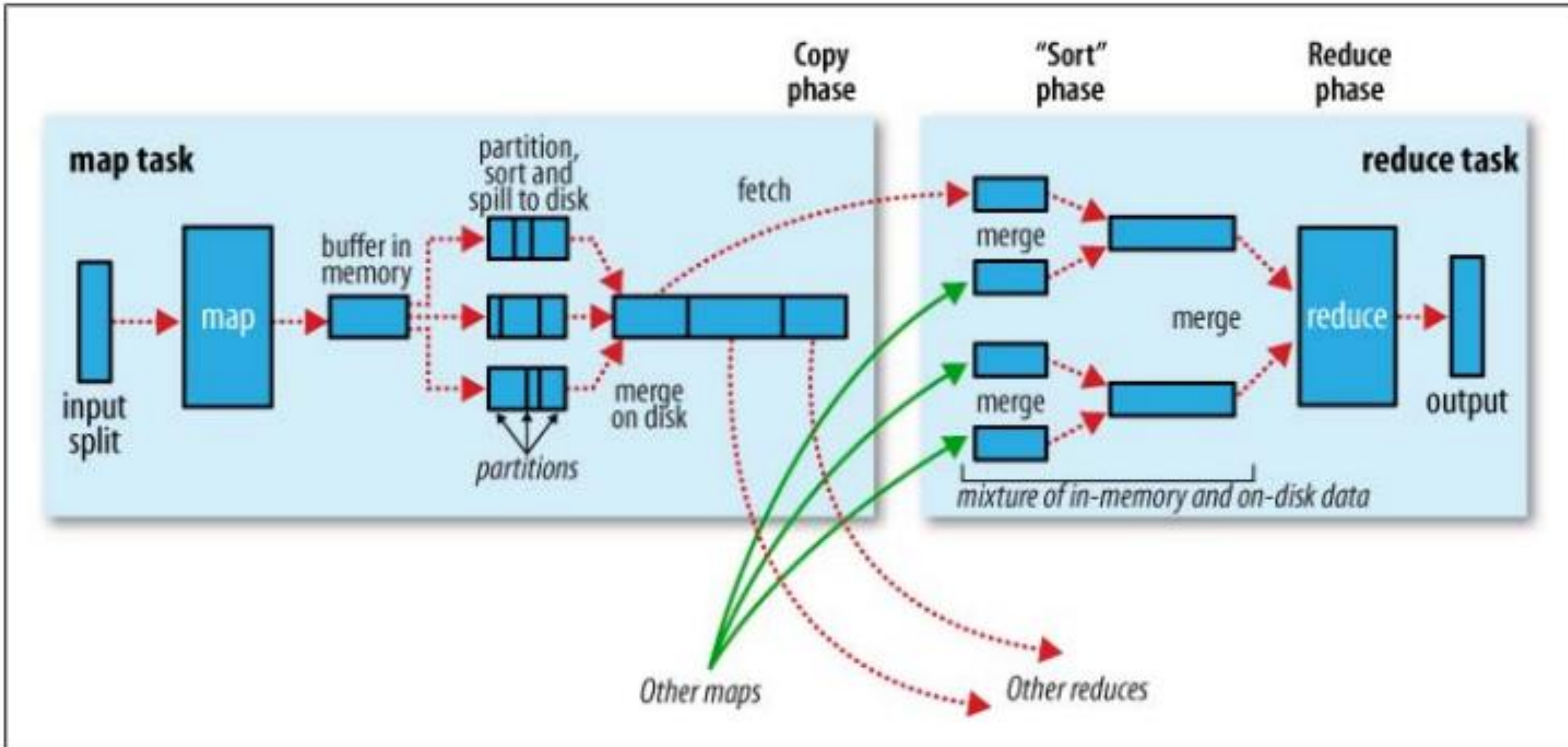
# Nodes, Trackers, Tasks

- Master node runs *JobTracker* instance, which accepts *Job* requests from clients
- *TaskTracker* instances run on slave nodes
- *TaskTracker* forks separate Java process for task instances

# MapReduce Workflow



# MapReduce in One Picture




Tom White, *Hadoop: The Definitive Guide*

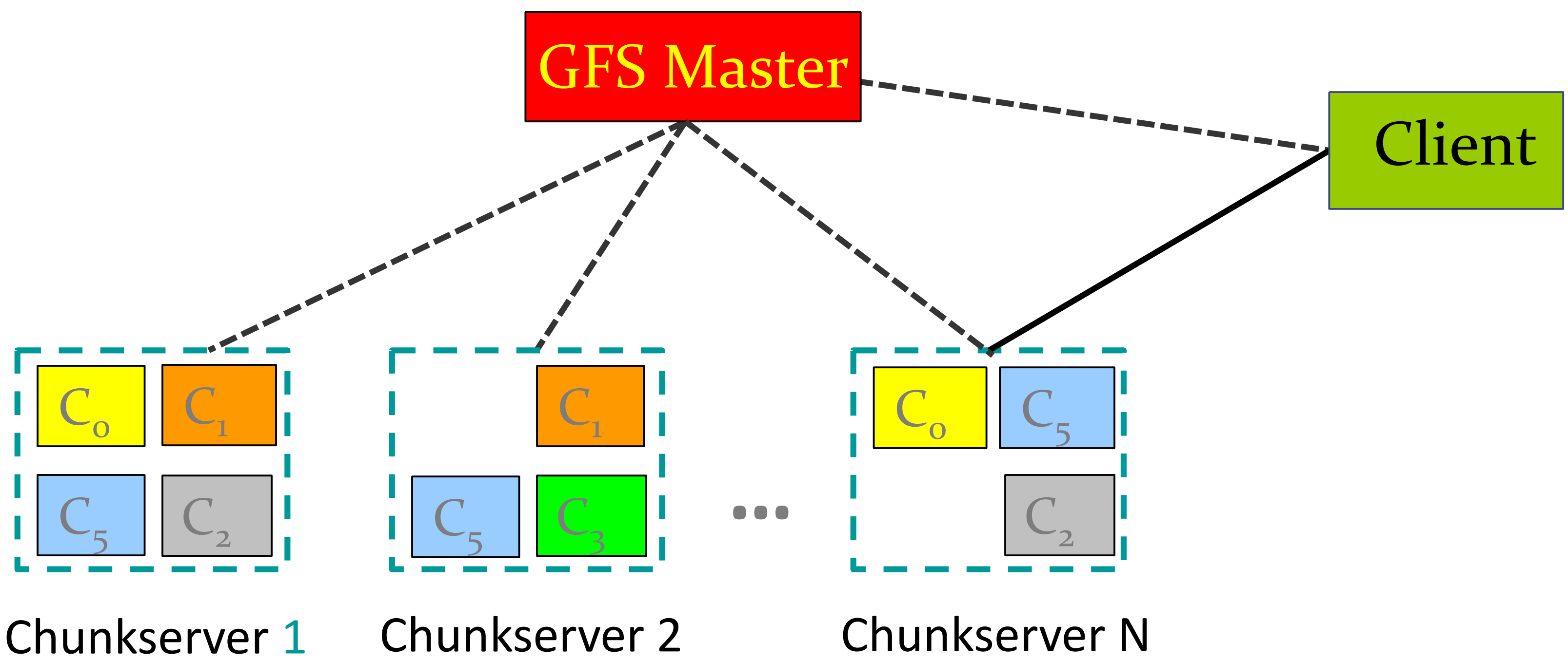
# MapReduce Runtime System

1. Partitions input data
2. Schedules execution across a set of machines
3. Handles machine failure
4. Manages interprocess communication

# GFS: Underlying Storage System

- Goal
  - global view
  - make huge files available in the face of node failures
- Master Node (meta server)
  - Centralized, index all chunks on data servers
- Chunk server (data server)
  - File is split into contiguous chunks, typically 16-64MB
  - Each chunk replicated (usually 2x or 3x)
    - Try to keep replicas in different racks

# GFS Architecture



# Parallelism

- map() functions run in parallel, creating different intermediate values from different input data sets
- reduce() functions also run in parallel, each working on a different output key
- All values are processed *independently*
- Bottleneck: reduce phase can't start until map phase is completely finished

# Locality

- Master program divides up tasks based on location of data: tries to have map() tasks on same machine as physical file data, or at least same rack
- map() task inputs are divided into 64 MB blocks: same size as Google File System chunks

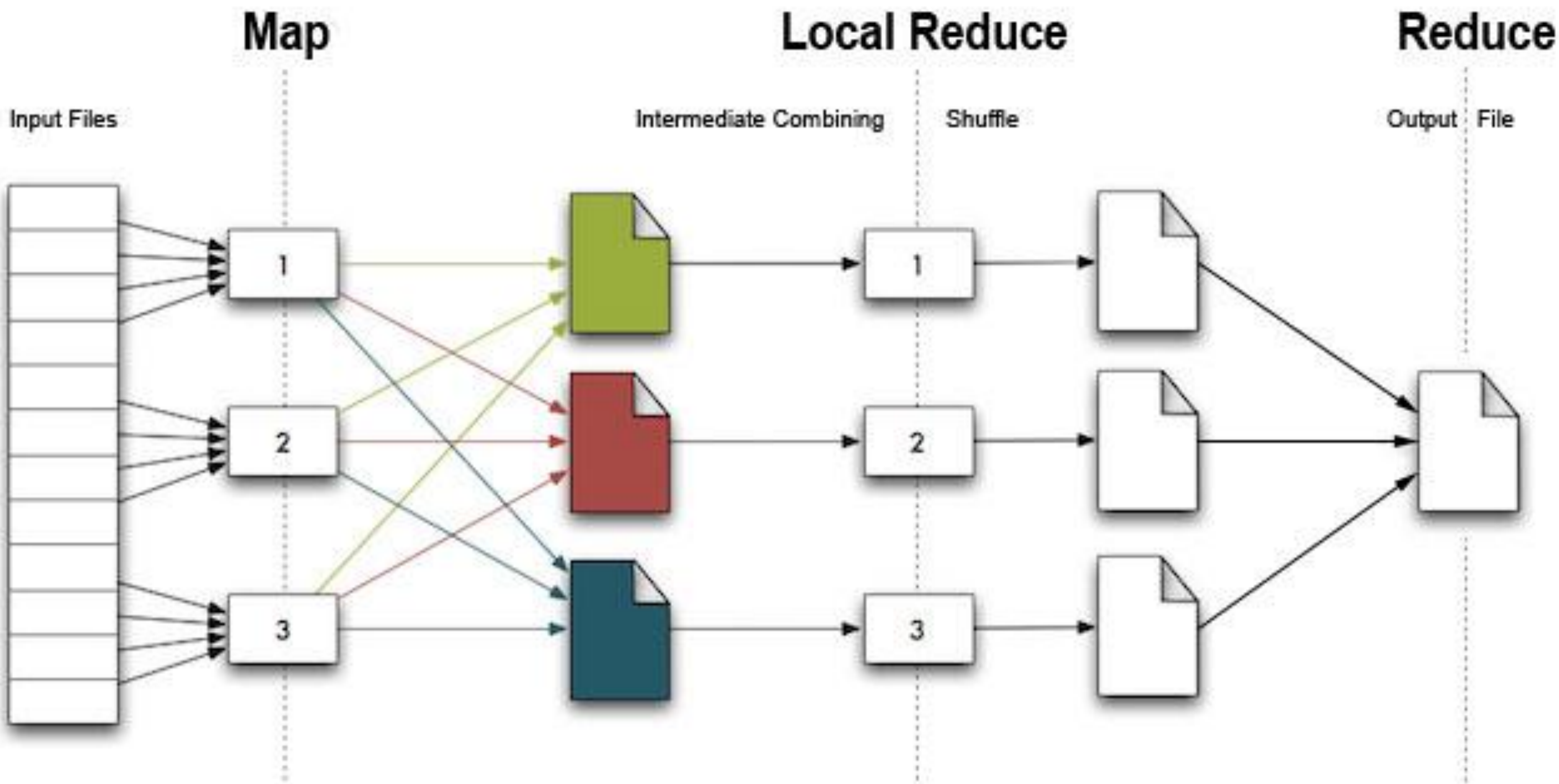
# Fault Tolerance

- Master detects worker failures
  - Re-executes completed & in-progress map() tasks
    - Re-execute
      - All output was stored locally
  - Re-executes in-progress reduce() tasks
    - Only re-execute partially completed tasks
      - All output stored in the global file system
- Master notices particular input key/values cause crashes in map(), and skips those values on re-execution
  - Effect: Can work around bugs in third-party libraries!

# Optimizations

- No reduce can start until map is complete
  - A single slow disk controller can rate-limit the whole process
- Master redundantly executes “slow-moving” map tasks; uses results of first copy to finish
- “Combiner” functions can run on same machine as a mapper
- Causes a mini-reduce phase to occur before the real reduce phase, to save bandwidth

# Optimizations



# MapReduce Benefits

- Greatly reduces parallel programming complexity
  - Reduces synchronization complexity
  - Automatically partitions data
  - Provides failure transparency
  - Handles load balancing

# Outline

- MapReduce Programming Model
- MapReduce Examples
- Hadoop

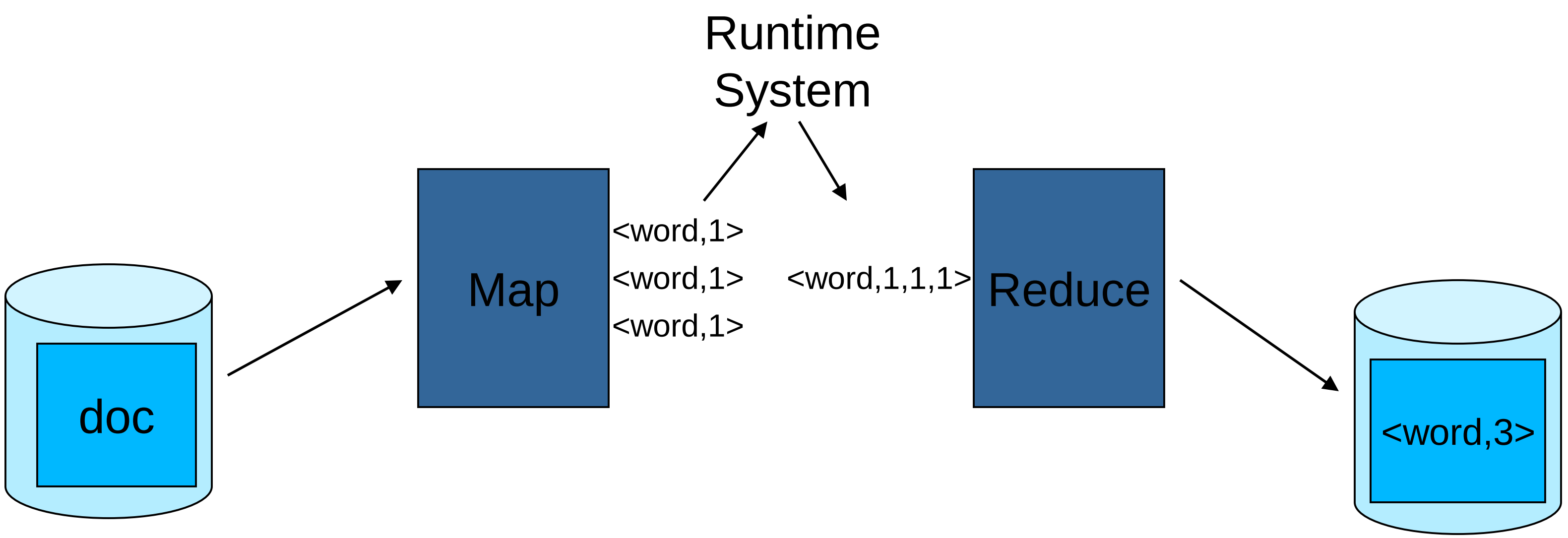
# Typical Problems Solved by MapReduce

- Read a lot of data
- Map: extract something you care about from each record
- Shuffle and Sort
- Reduce: aggregate, summarize, filter, or transform
- Write the results

- Outline stays the same, but **map** and **reduce** change to fit the problem

# MapReduce Examples

- Word frequency

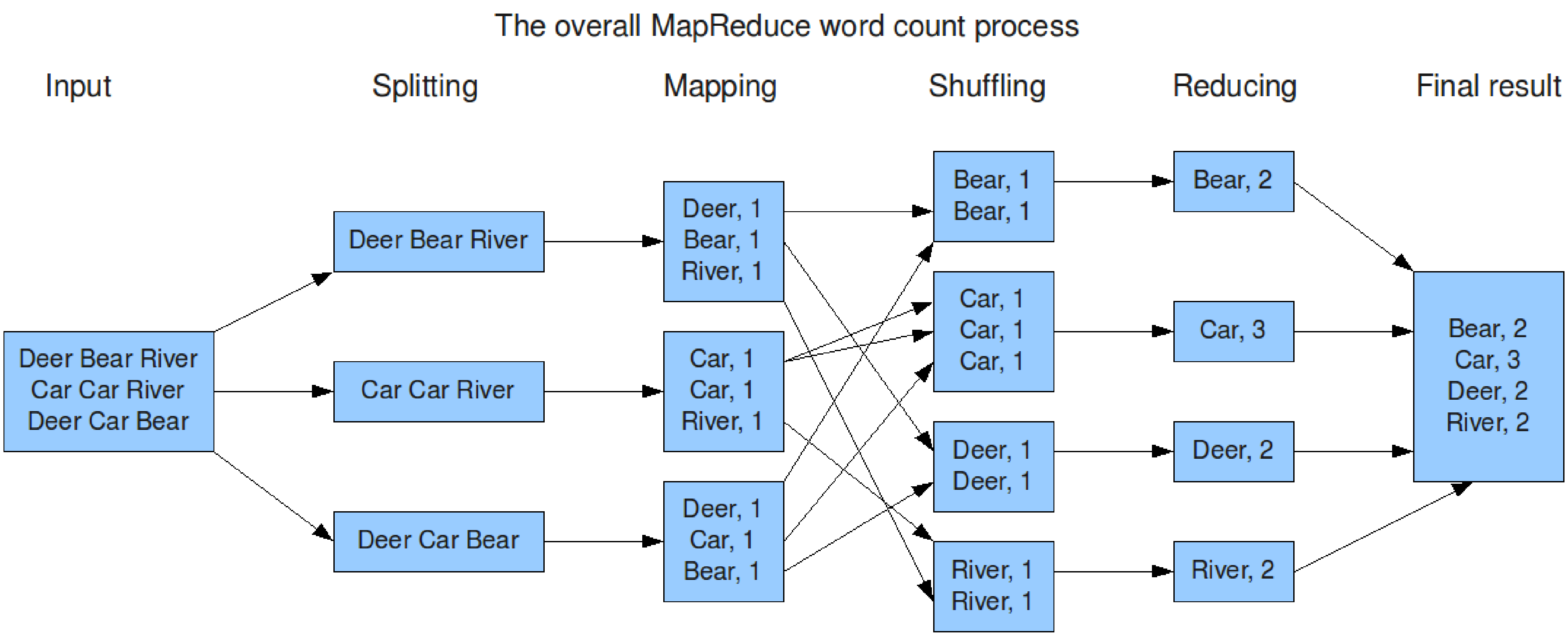


# Example: Count Word Occurrences

```
map(String input_key, String input_value):
  // input_key: document name
  // input_value: document contents
  for each word w in input_value:
    EmitIntermediate(w, "1");


reduce(String output_key, Iterator
   intermediate_values):
  // output_key: a word
  // output_values: a list of counts
  int result = 0;
  for each v in intermediate_values:
    result += ParseInt(v);
 Emit(AsString(result));
```

# Example: Count Word Occurrences



# MapReduce Examples

- Distributed grep
  - Map function emits <word, line_number> if word matches search criteria
  - Reduce function is the identity function
- URL access frequency
  - Map function processes web logs, emits <url, 1>
  - Reduce function sums values and emits <url, total>

# Outline

- MapReduce Programming Model
- MapReduce Examples
- Hadoop

# Hadoop

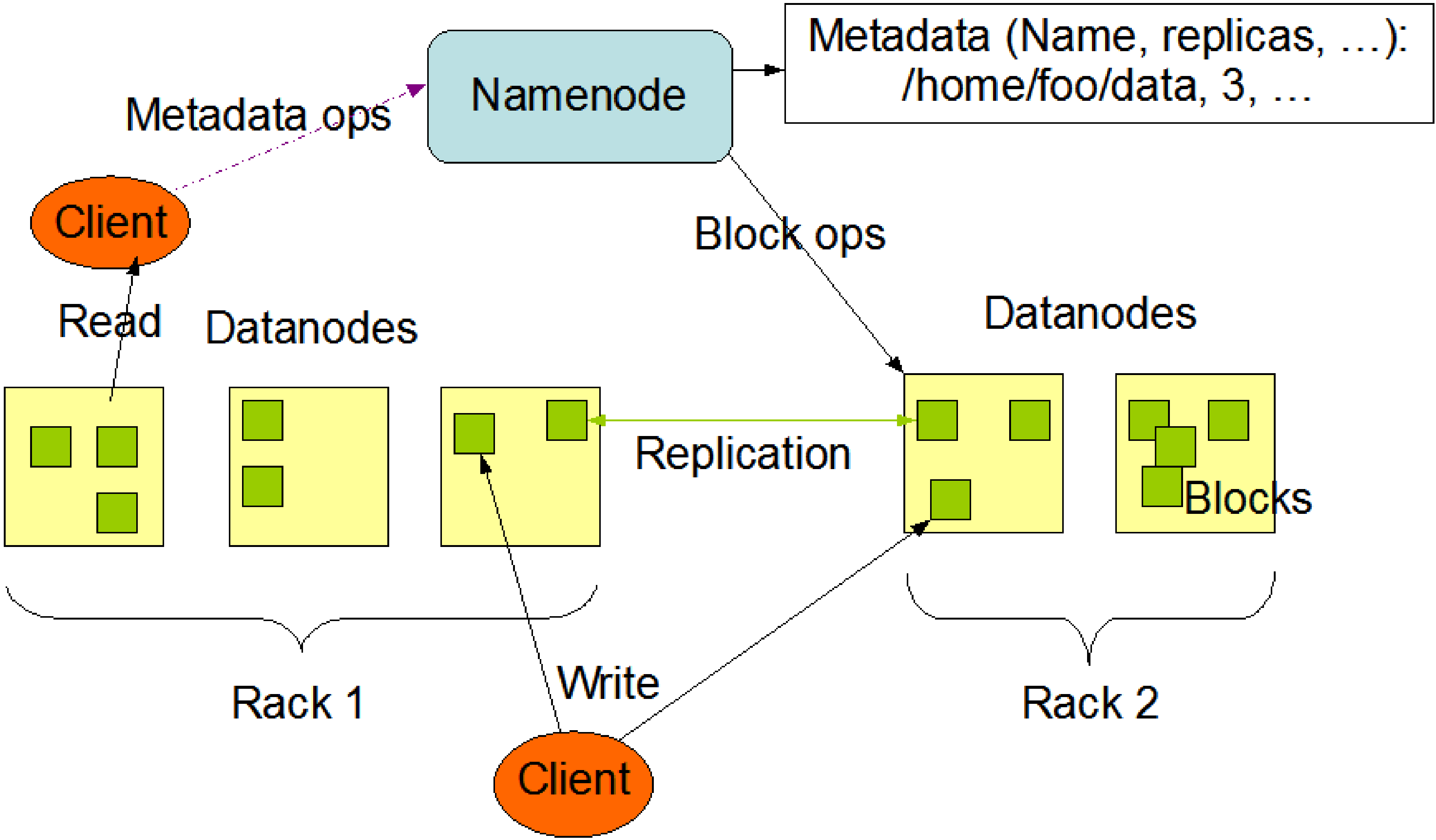
- Open source MapReduce implementation
  - http://hadoop.apache.org/core/index.html

| Google calls it | Hadoop equivalent |
|---|---|
| MapReduce | Hadoop |
| GFS | HDFS |
| Bigtable | HBase |

# Basic Features: HDFS

- Highly fault-tolerant
- High throughput
- Suitable for applications with large data sets
- Streaming access to file system data
- Can be built out of commodity hardware

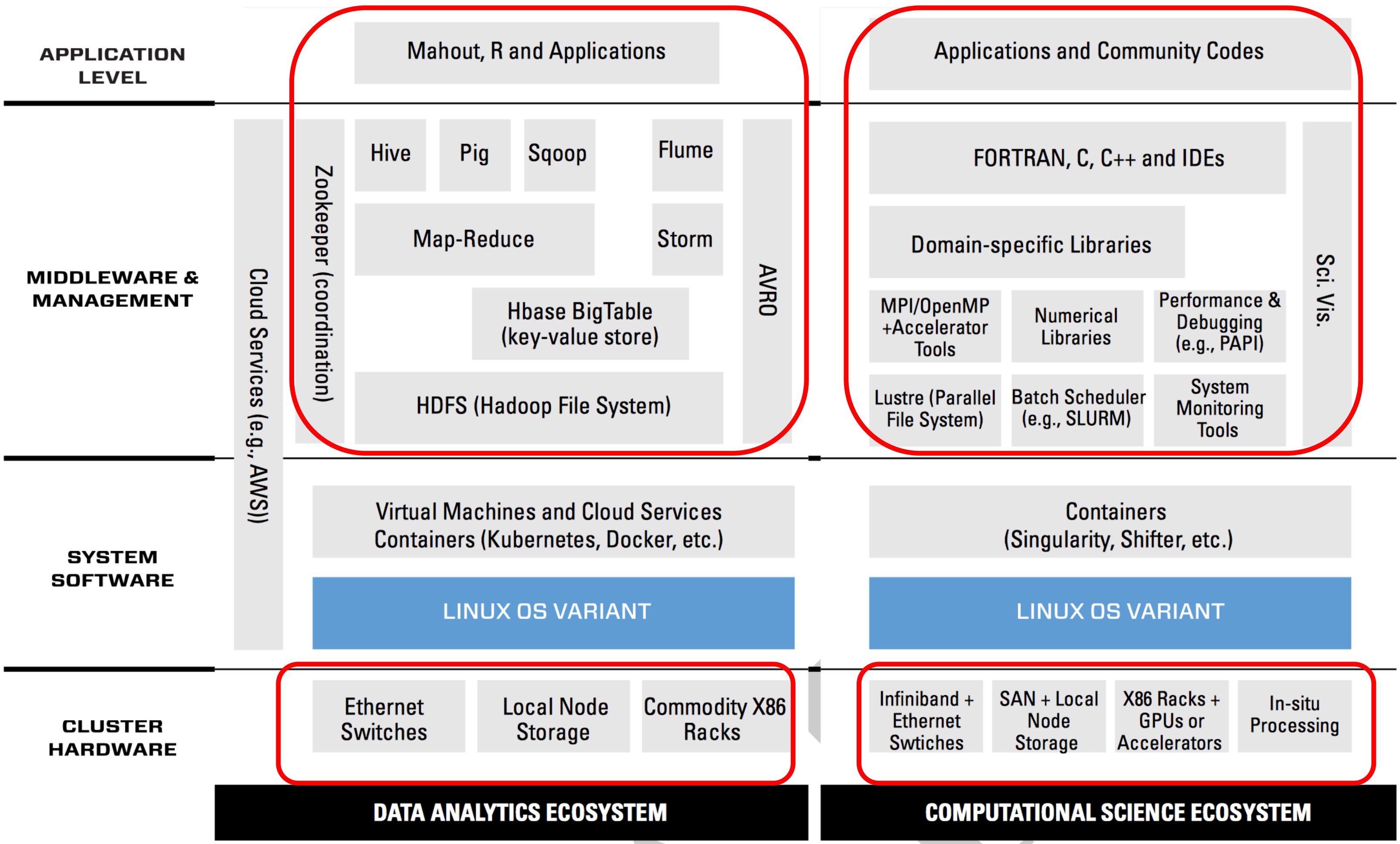
# HDFS Architecture



# Hadoop Related Projects

- Ambari: A web-based tool for provisioning, managing, and monitoring Apache Hadoop clusters which includes support for Hadoop HDFS, Hadoop MapReduce, Hive, HCatalog, HBase, ZooKeeper, Oozie, Pig and Sqoop. Ambari also provides a dashboard for viewing cluster health such as heat maps and ability to view MapReduce, Pig and Hive applications visually along with features to diagnose their performance characteristics in a user-friendly manner
- Avro: A data serialization system
- Cassandra: A scalable multi-master database with no single points of failure
- Chukwa: A data collection system for managing large distributed systems
- HBase: A scalable, distributed database that supports structured data storage for large tables (NoSQL)
- Hive: A data warehouse infrastructure that provides data summarization and ad hoc querying
- Mahout: A Scalable machine learning and data mining library
- Pig: A high-level data-flow language and execution framework for parallel computation
- ZooKeeper: A high-performance coordination service for distributed applications

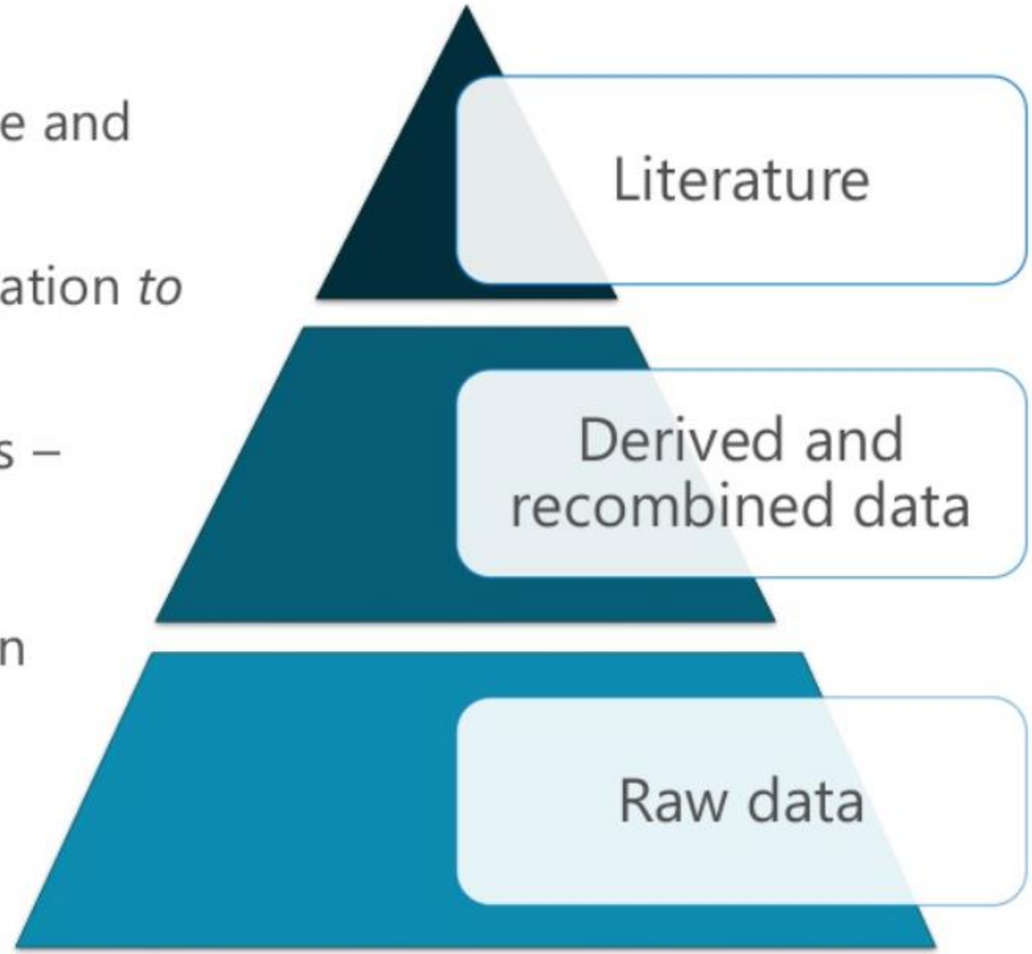
# Two Ecosystems




BDEC Committee, The BDEC "Pathways to Convergence" Report, 2017

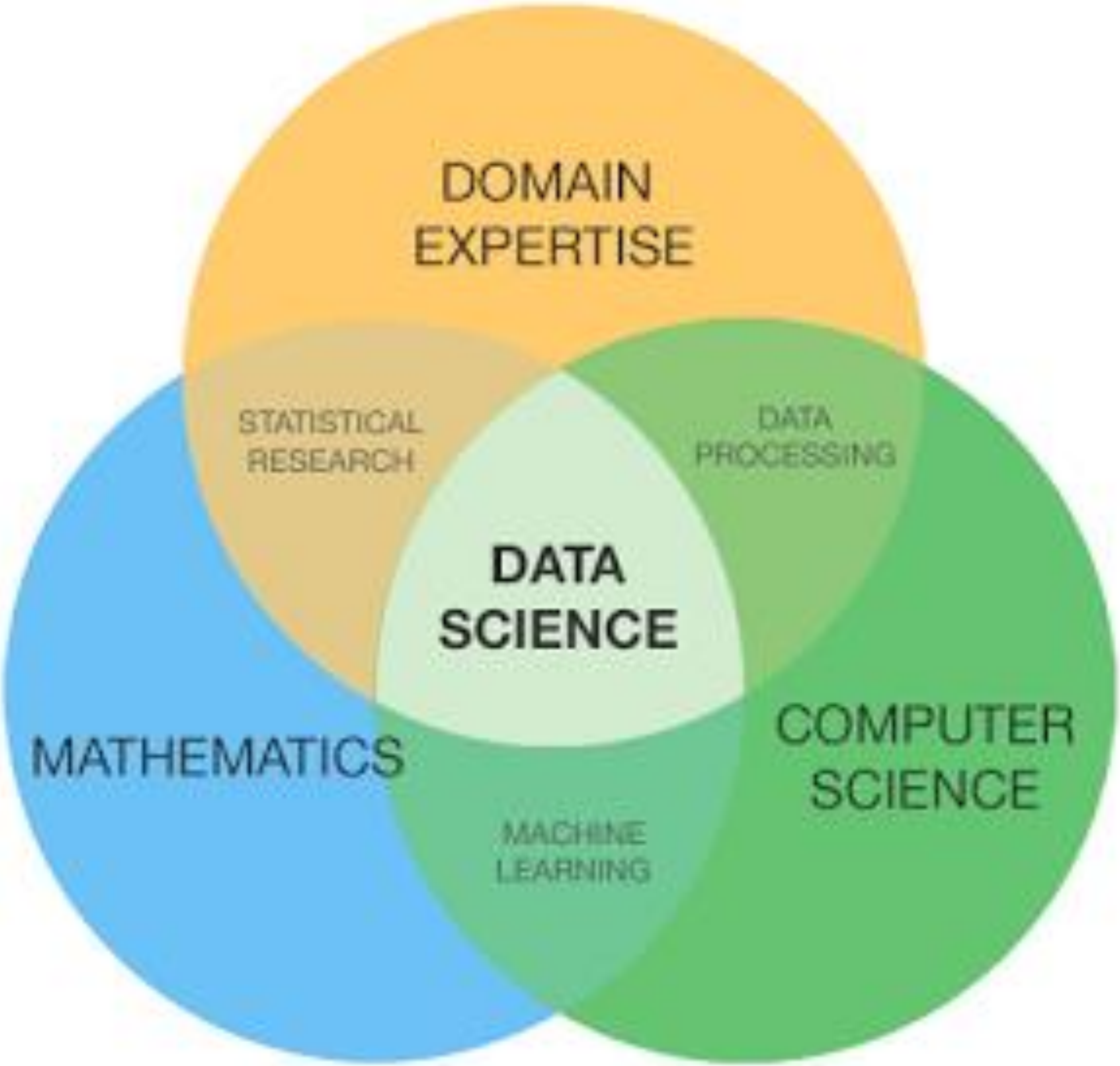
# All Scientific Data Online

- Many disciplines overlap and use data from other sciences.
- Internet can unify all literature and data
- Go from literature *to* computation *to* data *back to* literature.
- Information at your fingertips – For everyone, everywhere
- Increase Scientific Information Velocity
- Huge increase in Science Productivity




*(From Jim Gray's last talk)*

# Data Science



# Data Science in Wikipedia