Big Data Analytics

Manos Papagelis

Overview

- Data Driven Organizations (DDOs)
- Evaluating DDOs solutions
- Big Data Architectures
- Processing Platforms
  - Distributed File System
  - The Map-Reduce Programming Model
- Summary

Data Driven Organizations (DDOs)

How non-DDOs make decisions?

- Intuition
- Ad-hoc or based on few customers feedback
- Look at competition
- Try to be different
- Based on assumptions (that may be wrong)
- No way to validate if it was the right decision

What do DDO's do?

- Make decisions based on data not intuition
- More precise on what they want to achieve
- Measure and validate with data

Example 1: Email Marketing

Pre DDO

- Did not measure campaign effectiveness
- Did not cluster customers
- Did not have tailored campaigns

Post DDO

- Behavioral clustering
- Predictive analytics
- Life-time Value Analysis
- Targeted campaigns
- Measure effectiveness

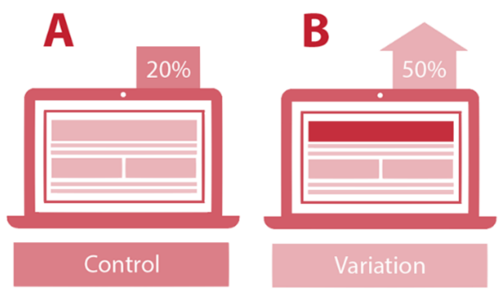
Result

- Cannibalized own market
- Offered discounts when not needed
- Significant loss revenue

Result

- Increased revenue

Example 2: Application Feature



Pre DDO

- Introduced features on intuition
- No measurable goals

Post DDO

- A/B testing, measures
- Do not launch unless measurable benefit

Result

- Sometimes features decreased engagement
- Many features, unknown value
- Occasional lost revenue

Result

- More successful feature introductions (increased engagement)
- Remove features that do not contribute to metrics

Summary

DDOs

- collect data
- make decisions based on data, not intuition
- use data to drive applications

To be a DDO, you need an efficient way of storing and retrieving data

Evaluating DDO Solutions

Challenge

- A variety of solutions/technologies available
- There is no one solution/technology that solves all possible data analytics problems
- Most solutions solve a range of problems, but are outstanding on a specific type

How to map problems to DDO solutions?

How to compare alternative DDO solutions?

To be able to evaluate DDO solutions you need to understand your needs

DDOs Evaluation

Data dimension

What characteristics should be considered with respect to **data**?

- Structure
- Size
- Sink Rate
- Source Rate
- Quality
- Completeness

Processing dimension

What characteristics should be considered with respect to **processing**?

- Query Selectivity
- Query Execution Time
- Aggregation
- Processing Time
- Join
- Precision

Other dimensions: cost, implementation complexity, …

Example DDO Solutions

RDBMS: Relational model with powerful querying capabilities

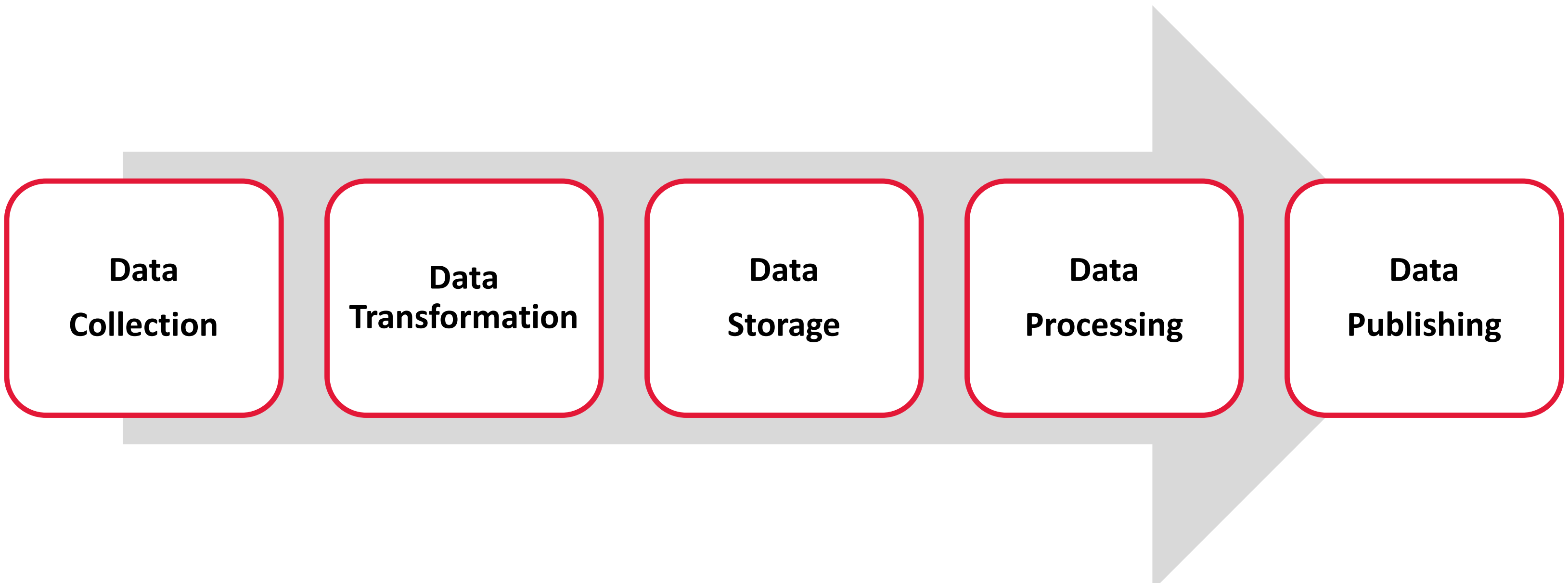
HDFS+M/R: Batch oriented system for processing and storing large data sets

Storm: A stream processing system that computes in real-time over large streams

BlinkDB: Experimental system for approximate query answering over large data that trade error over response time

Big Data Architectures

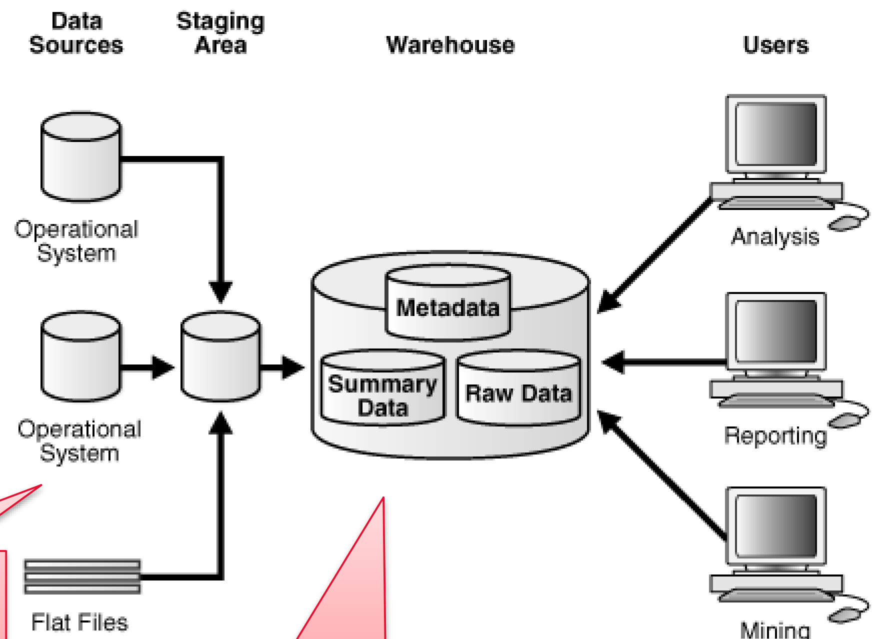
Data Analytics Pipeline



Traditional Approach

- Handles **certain type of data** well
- Handles **certain ranges of data size** well
- Performs **certain types of queries and computations** well

Traditional Business Warehouse

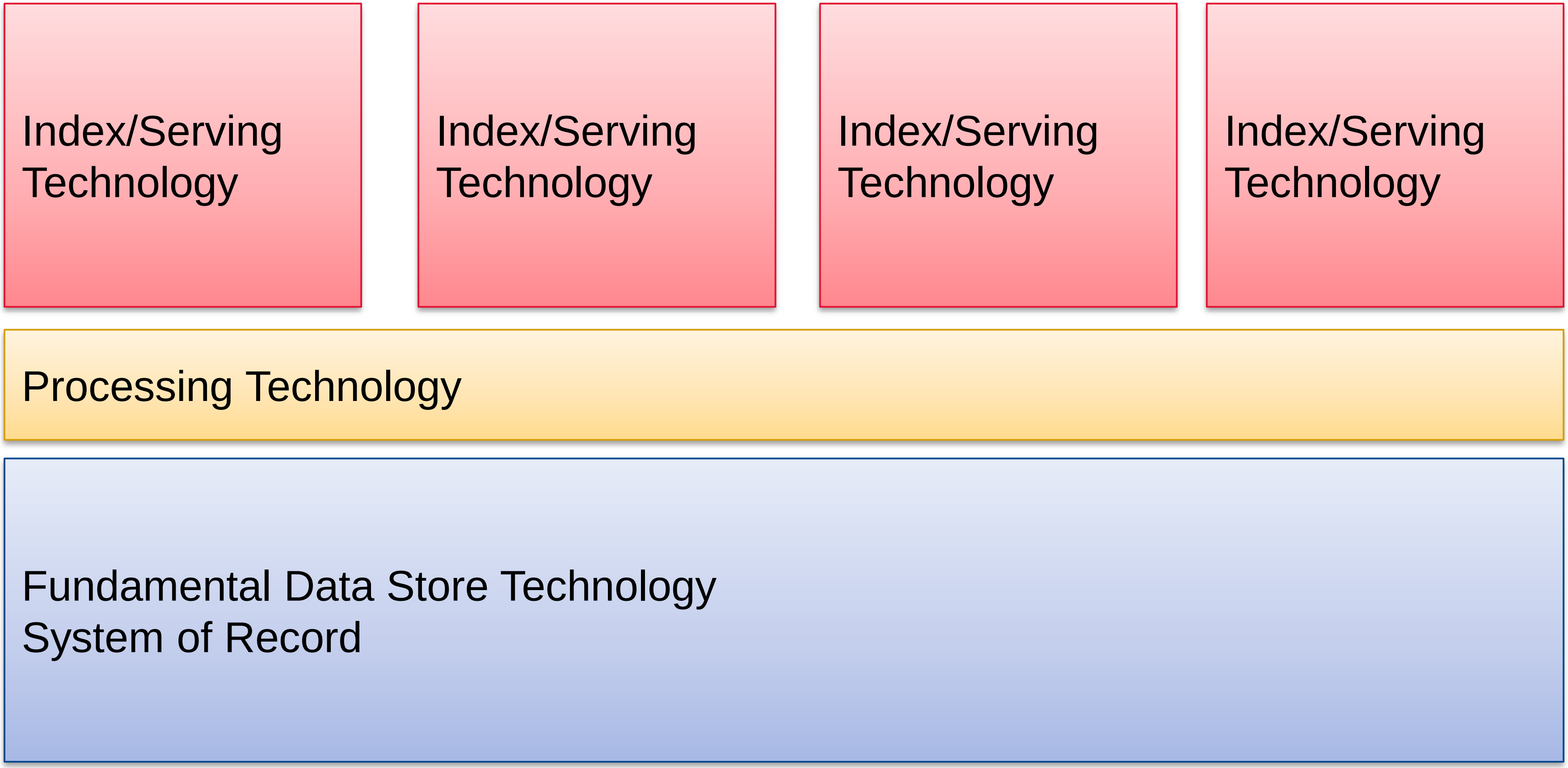


Extract
Transform
Load

Analytics

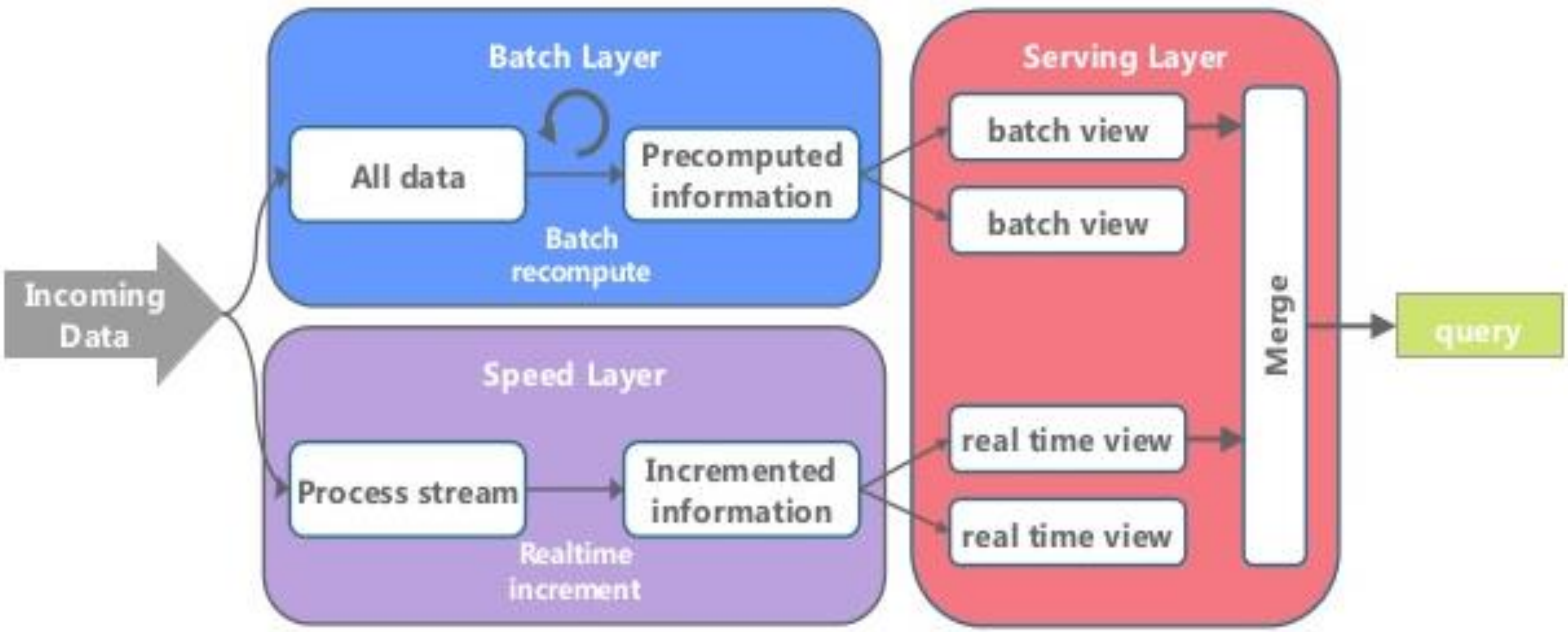
Reporting

A Big Data Approach



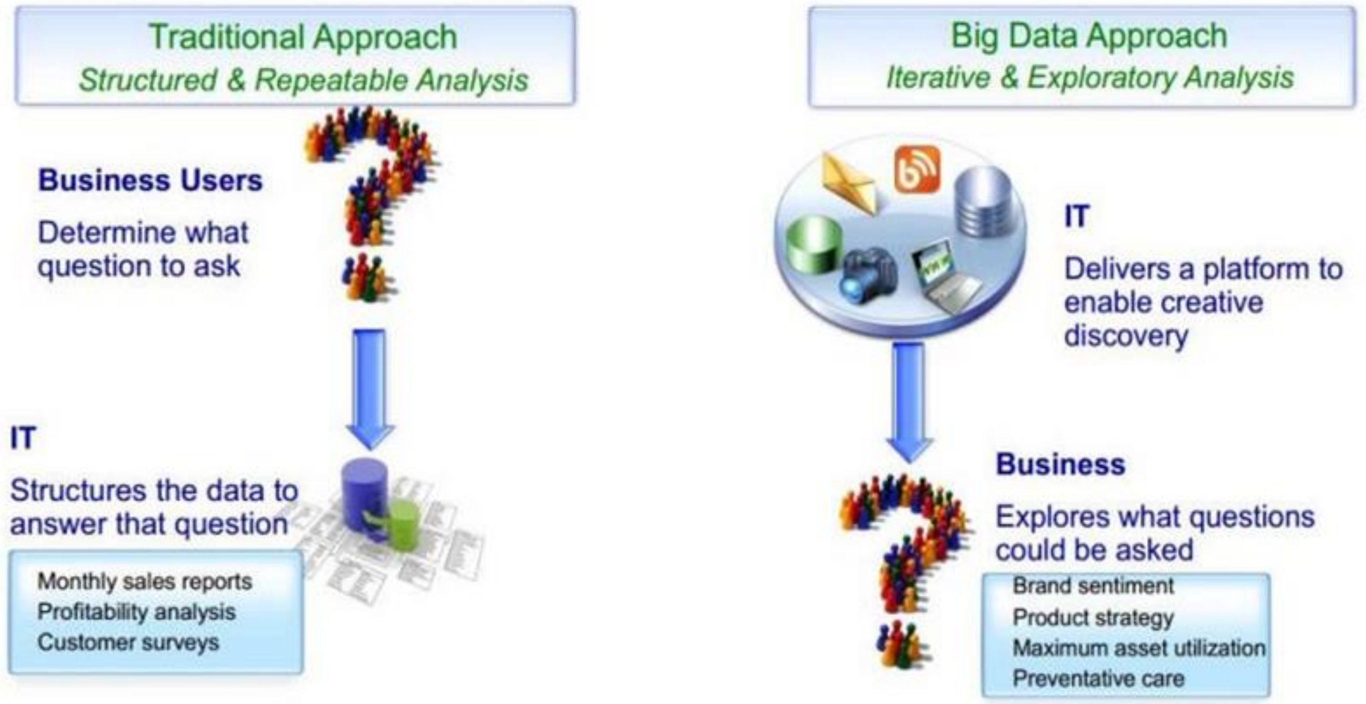
Big Data Analytics Architecture

Example: Lambda Architecture



Other examples: Kappa Architecture, Netflix Architecture

Difference in Approach



Notice the difference!

Processing Platforms

Big Data Technology & Analytics

Data Ingestion
ETL, Distcp, Kafka, OpenRefine, …

Query & Exploration
SQL, BigQuery, Hive, SparkSQL, Search, …

Stream Processing Platforms
Storm, Spark, ..

Batch Processing Platforms
MapReduce, Spark, …

Data Definition
SQL DDL, Avro, Protobuf, CSV

Storage Systems
HDFS, RDBMS, Column Stores, Graph Databases

Data Serving
BI, Cubes, RDBMS, Key-value Stores, Tableau, …

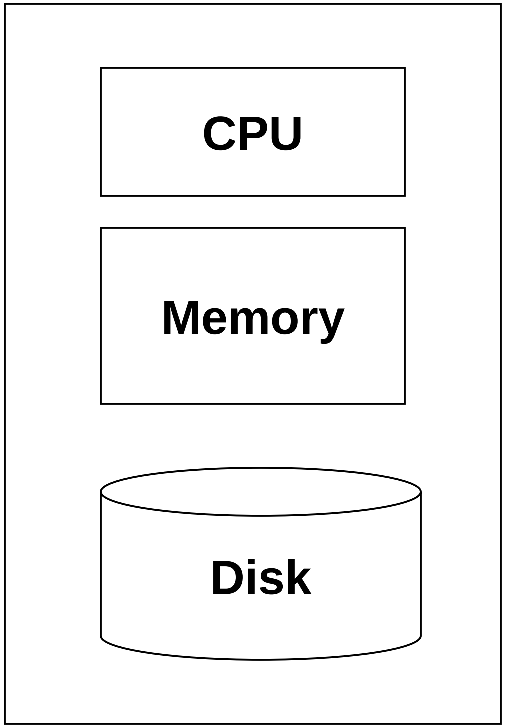
Computing Platforms
Distributed Commodity, Clustered High-Performance, Single Node

Processing Platforms

- Batch Processing
  - Google GFS/MapReduce (2003)
  - Apache Hadoop HDFS/MapReduce (2004)
- SQL
  - BigQuery (based on Google Dremel, 2010)
  - Apache Hive (HiveQL) (2012)
- Streaming Data
  - Apache Storm (2011) / Twitter Huron (2015)
- Unified Engine (Streaming, SQL, Batch, ML)
  - Apache Spark (2012)

Distributed File System & the Map-Reduce Programming Model

Single Node Architecture



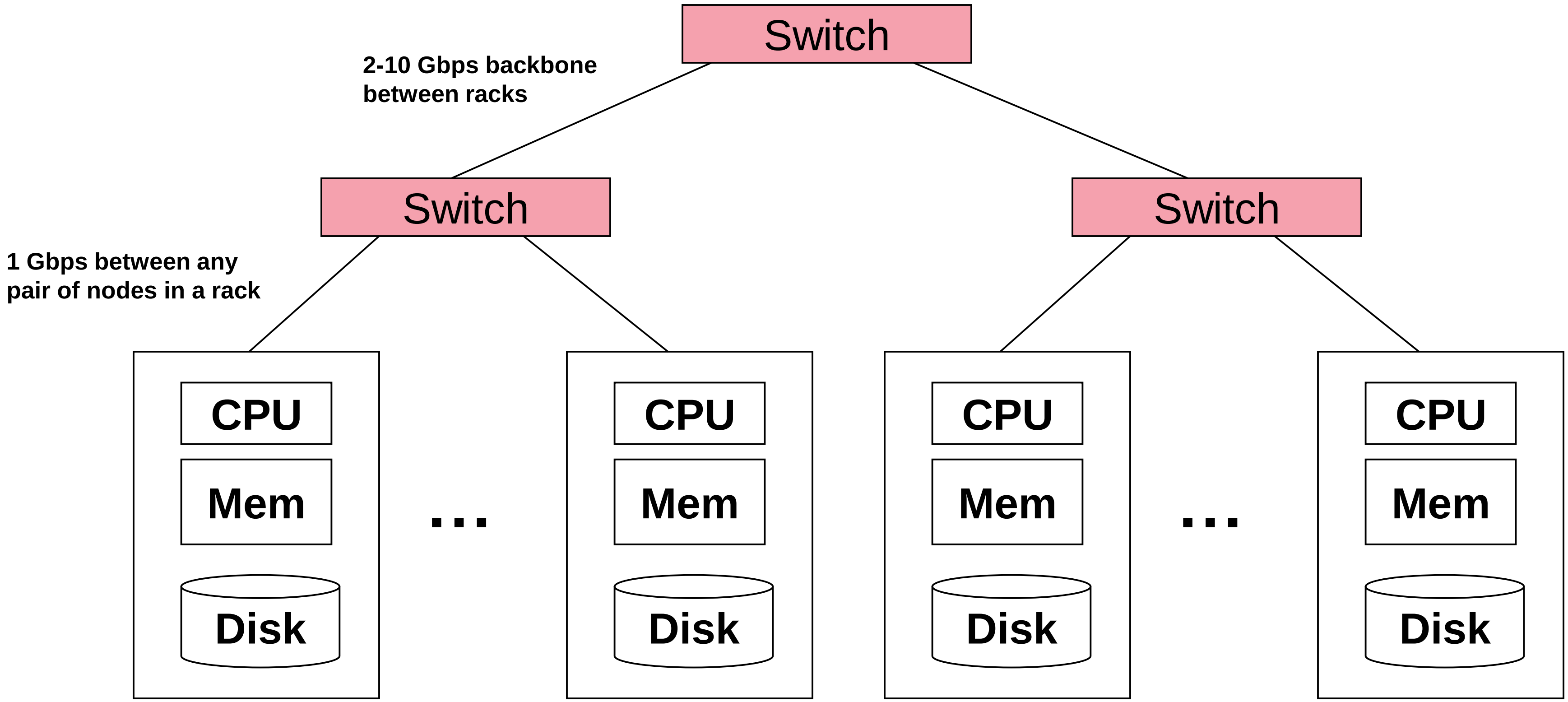
Machine Learning, Statistics

“Classical” Data Analytics

Motivation: Google Example

- 20+ billion web pages x 20KB = 400+ TB
- 1 computer reads 30-35 MB/sec from disk
  - ~4 months to read the web
- ~1,000 hard drives to store the web
- Takes even more to **do** something useful with the data!
- **Today, a standard architecture for such problems is emerging:**
  - Cluster of commodity Linux nodes
  - Commodity network (ethernet) to connect them

Cluster Architecture



Each rack contains 16-64 nodes

In 2011 it was guestimated that Google had 1M machines, http://bit.ly/Shh0RO

Large-scale Computing Challenges

- How do you distribute computation?
- How can we make it easy to write distributed programs?
- Machines fail:
  - One server may stay up 3 years (1,000 days)
  - If you have 1,000 servers, expect to loose 1/day
  - People estimated Google had ~1M machines in 2011
    - 1,000 machines fail every day!

Idea and Solution

- **Issue: Copying data over a network takes time**
- **Idea:**
  - Store files multiple times for reliability
  - Bring computation close to the data
- **Storage Infrastructure: Distributed File system**
  - Google: GFS. Hadoop: HDFS
- **Programming Model: Map-Reduce**
  - Google’s computational/data manipulation model
  - Elegant way to work with big data

Storage Infrastructure

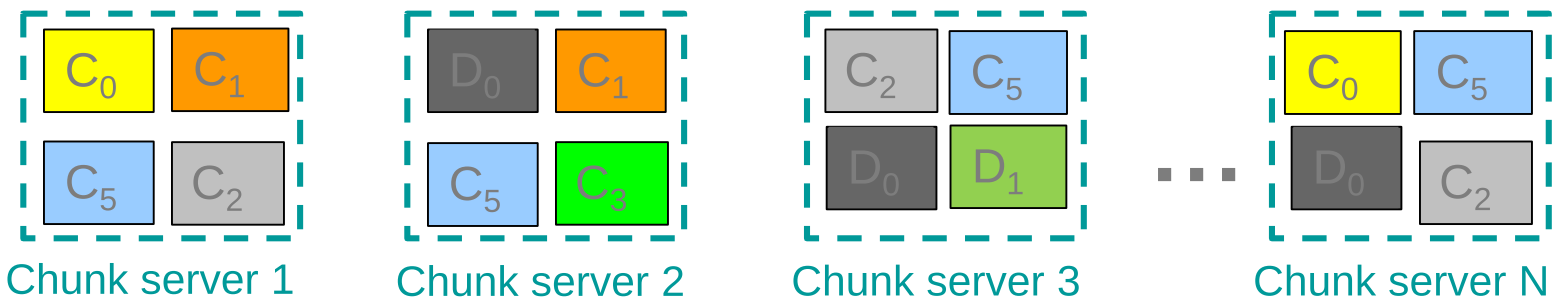
- Problem:
  - If nodes fail, how to store data persistently?
- Answer:
  - Distributed File System:
    - Provides global file namespace
    - Google GFS; Hadoop HDFS
- Typical usage pattern
  - Huge files (100s of GB to TB)
  - Data *reads* and *appends* are common
  - Data is rarely *updated* in place

Distributed File System

- Chunk servers
  - File is split into contiguous chunks
  - Typically each chunk is 16-64MB
  - Each chunk replicated (usually 3x)
  - Try to keep replicas in different racks
- Master node
  - a.k.a. Name Node in Hadoop's HDFS
  - Stores metadata about where files are stored
  - Might be replicated
- Client library for file access
  - Talks to master to find chunk servers
  - Connects directly to chunk servers to access data

Distributed File System

- **Reliable distributed file system**
- Data kept in “chunks” spread across machines
- Each chunk replicated on different machines
  - Seamless recovery from disk or machine failure



Chunk servers also serve as compute servers

Bring computation directly to the data!

Programming Model: MapReduce

Warm-up task

- We have a huge text document
- Count the number of times each distinct word appears in the file

Sample application

- Analyze web server logs to find popular URLs

Task: Word Count

Case 1:

- File too large for memory, but all **`<word, count>`** pairs fit in memory

Case 2:

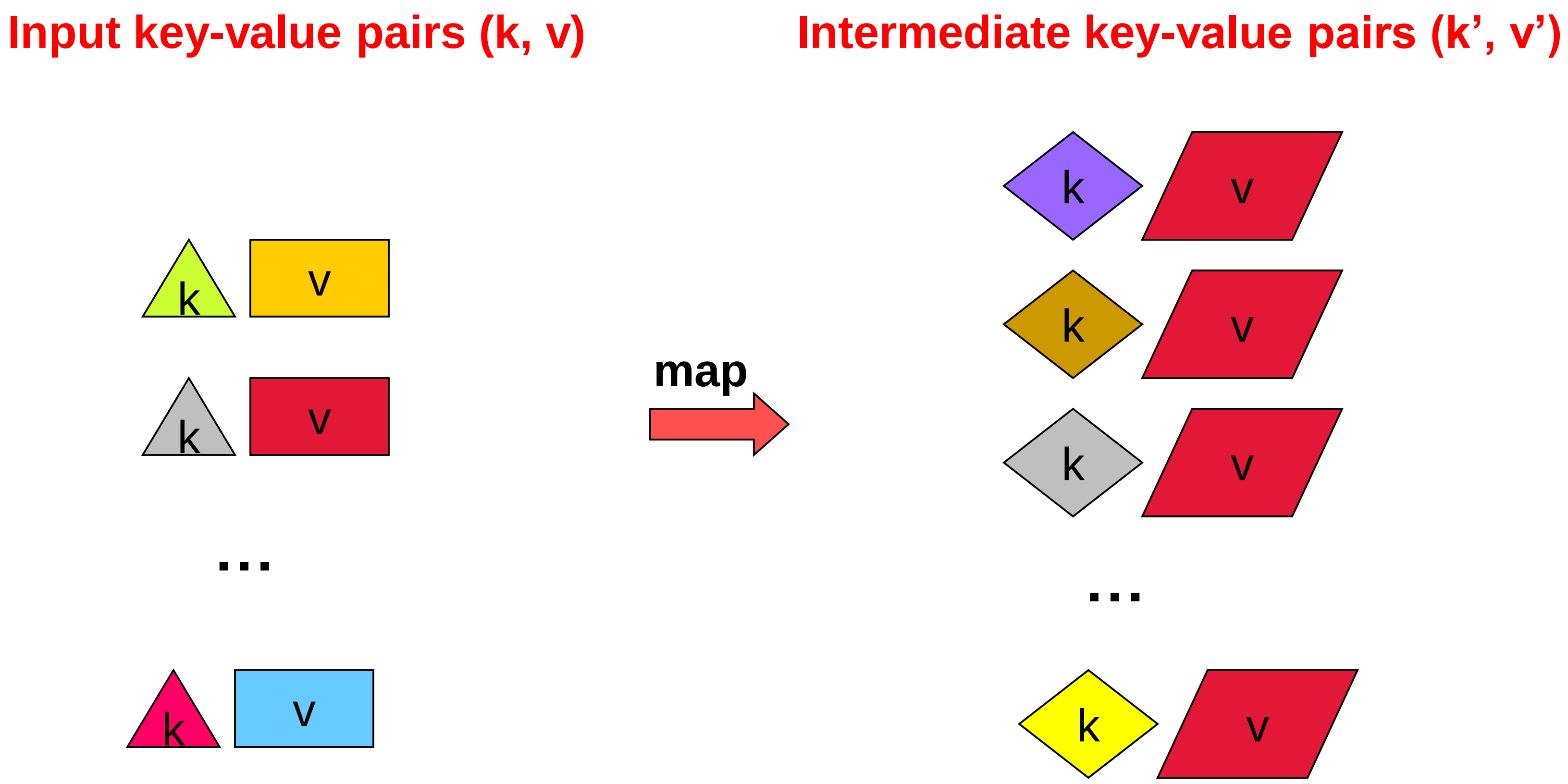
- Count occurrences of words:
  - **`words(doc.txt) | sort | uniq -c`**
    - where **`words`** takes a file and outputs the words in it, one per line
    - **uniq's –c option**, **--count** Prefix lines with a number representing how many times they occurred.
- Case 2 captures the essence of **MapReduce**
  - Great thing is that it is naturally parallelizable

MapReduce: Overview

- Sequentially read a lot of data
- **Map**: Extract something you care about
- **Group by key**: Sort and Shuffle
- **Reduce**: Aggregate, summarize, filter or transform
- Write the result

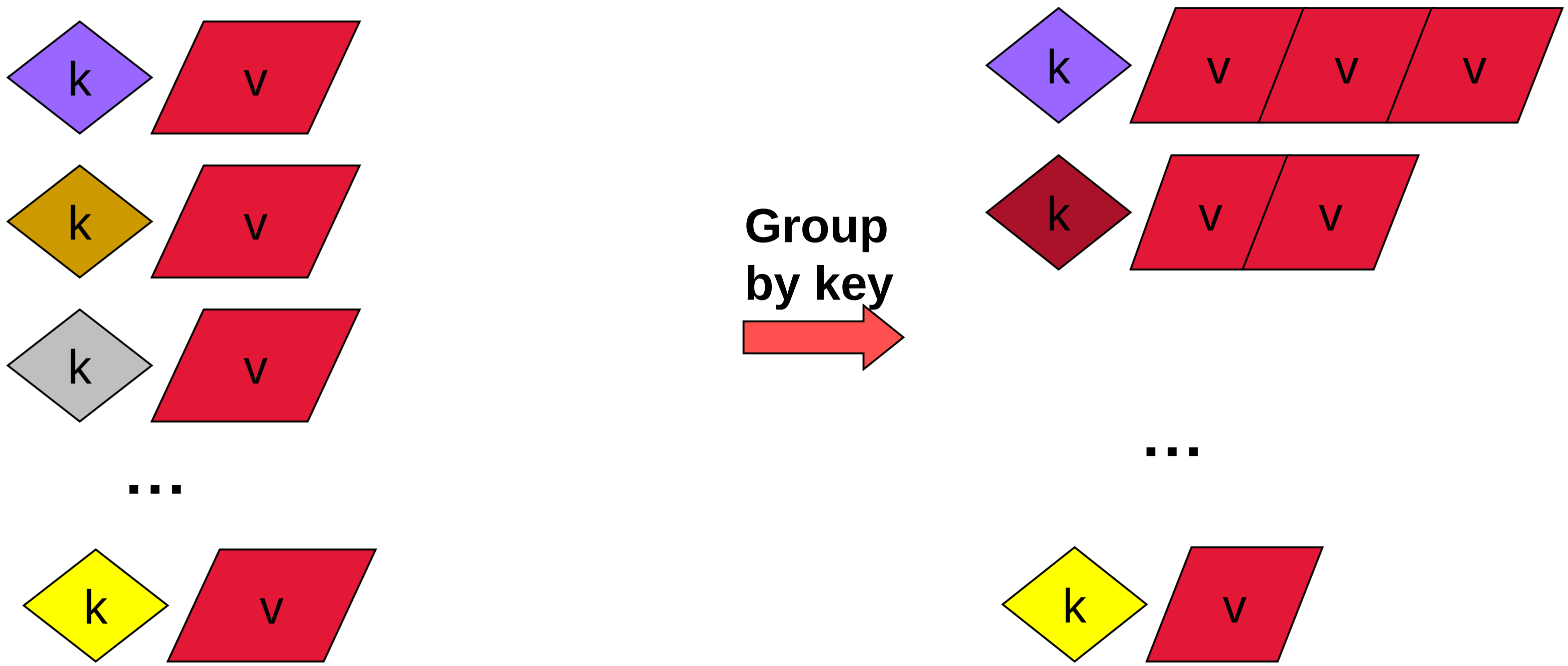
Outline stays the same, **Map** and **Reduce** steps change to fit the problem

MapReduce: The **Map** Step

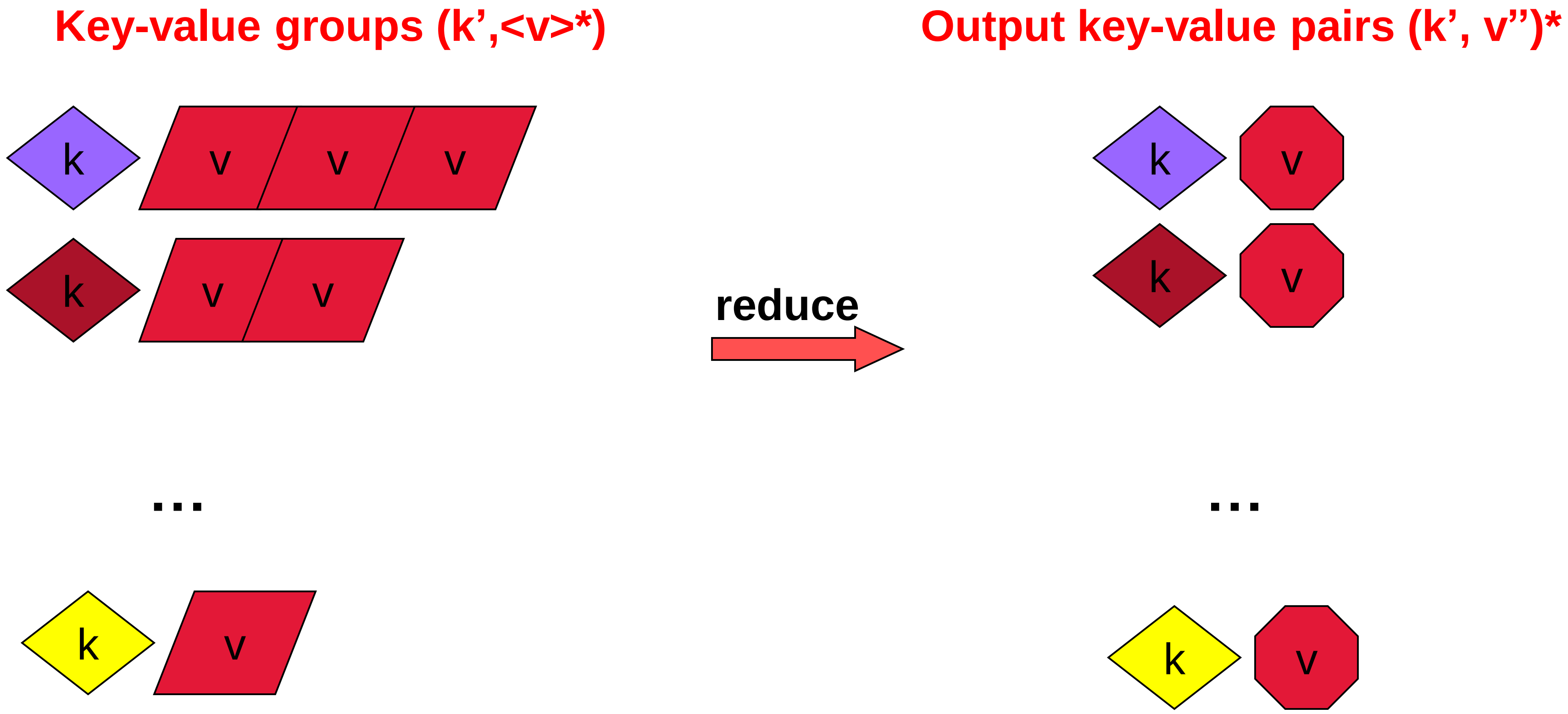


MapReduce: The **Group by key** Step

Intermediate key-value pairs (k’, v’) **Key-value groups (k’,<v>*)**



MapReduce: The **Reduce** Step



More Specifically

- **Input:** a set of key-value pairs
- Programmer specifies two methods:
  - **Map(k, v) → <k’, v’>***
    - Takes a key-value pair and outputs a set of key-value pairs
      - E.g., key is the filename, value is a single line in the file
    - There is one Map call for every *(k,v)* pair
  - **Reduce(k’, <v’>*) → <k’, v’’>***
    - All values *v’* with same key *k’* are reduced together and processed in *v’* order
    - There is one Reduce function call per unique key *k’*

MapReduce: Word Counting

Provided by the programmer

MAP
Read input and produces a set of **key-value** pairs

GROUP BY KEY
Collect all pairs with same key

Provided by the programmer

REDUCE
Collect all values belonging to the key and output

The crew of the space shuttle Endeavor recently returned to Earth as ambassadors, harbingers of a new era of space exploration. Scientists at NASA are saying that the recent assembly of the Dextre bot is the first step in a long-term space-based man/mache partnership. '"The work we're doing now -- the robotics we're doing -- is what we're going to need ……………………..

Big document

(The, 1)
(crew, 1)
(of, 1)
(the, 1)
(space, 1)
(shuttle, 1)
(Endeavor, 1)
(recently, 1)
….

(key, value)

(crew, 1)
(crew, 1)
(space, 1)
(the, 1)
(the, 1)
(the, 1)
(shuttle, 1)
(recently, 1)
…

(key, value)

(crew, 2)
(space, 1)
(the, 3)
(shuttle, 1)
(recently, 1)
…

(key, value)

only sequential reads

Word Count Using MapReduce

```
map(key, value):
// key: document name
// value: text of the document
  for each word w in value:
      emit(w, 1)

reduce(key, values):
// key: a word
// value: an iterator over counts
      result = 0
      for each count v in values:
            result += v
      emit(key, result)
```

Map-Reduce: Environment

Map-Reduce environment takes care of:

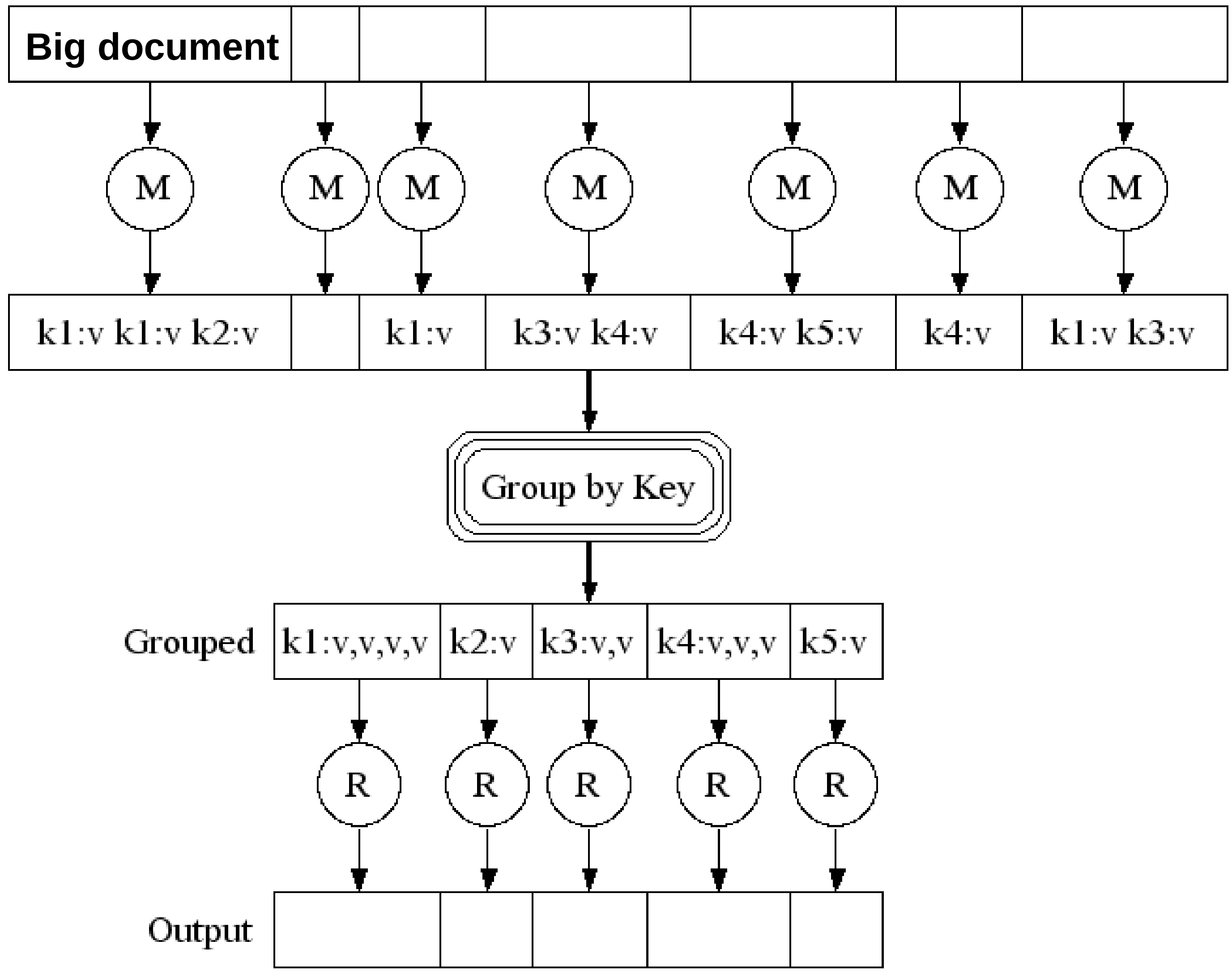
- **Partitioning** the input data
- **Scheduling** the program's execution across a set of machines
- Performing the **group by key** step
- Handling machine **failures**
- Managing required inter-machine **communication**

Map-Reduce: A diagram

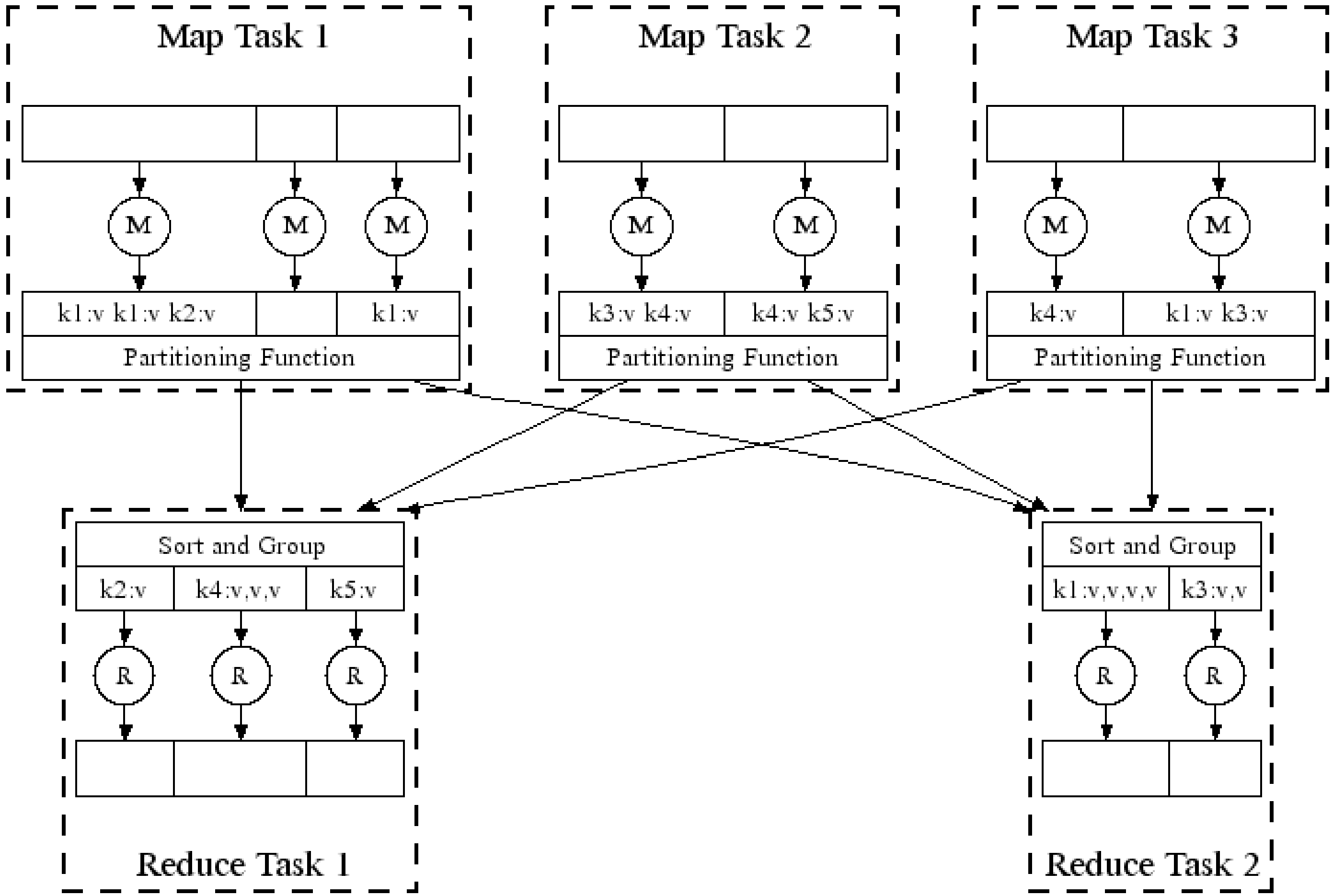
MAP
Read input and produces a set of key-value pairs

GROUP BY
Collect all pairs with same key
(Hash merge, Shuffle, Sort, Partition)

REDUCE
Collect all values belonging to the key and output



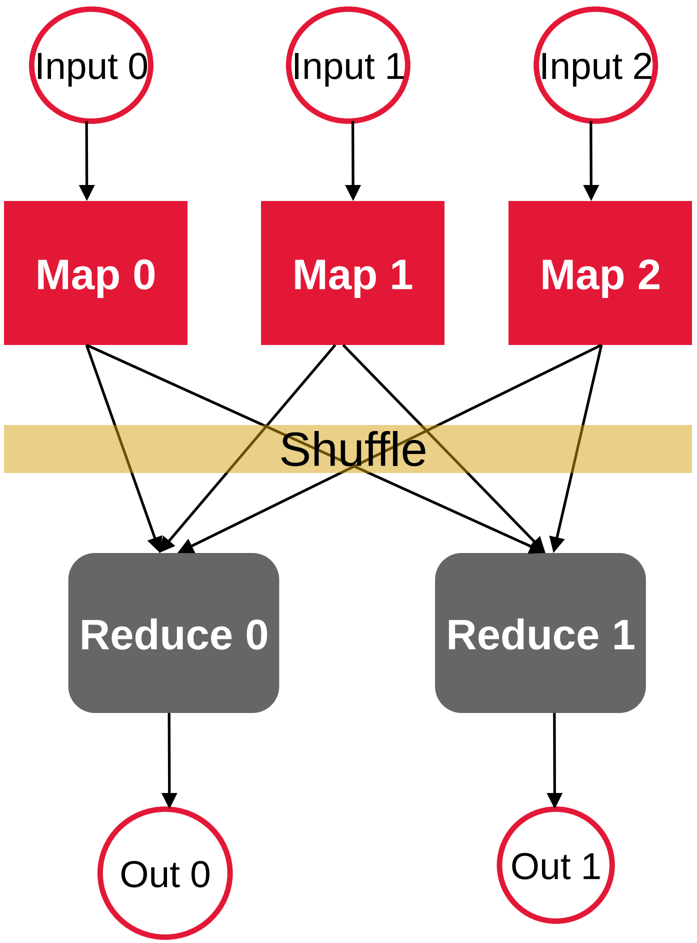
Map-Reduce: In Parallel



All phases are distributed with many tasks doing the work

Map-Reduce

- Programmer specifies:
  - Map and Reduce and input files
- Workflow:
  - Read inputs as a set of key-value-pairs
  - **Map** transforms input kv-pairs into a new set of k'v'-pairs
  - Sorts & Shuffles the k'v'-pairs to output nodes
  - All k’v’-pairs with a given k’ are sent to the same **reduce**
  - **Reduce** processes all k'v'-pairs grouped by key into new k''v''-pairs
  - Write the resulting pairs to files
- All phases are distributed with many tasks doing the work



Data Flow

- Input and final output are stored on a distributed file system (FS):
  - Scheduler tries to schedule map tasks “close” to physical storage location of input data
- Intermediate results are stored on local FS of Map and Reduce workers
- Output is often input to another MapReduce task

Summary

Summary: Processing Platforms

- Batch Processing
  - Google GFS/MapReduce (2003)
  - Apache Hadoop HDFS/MapReduce (2004)
- SQL
  - BigQuery (based on Google Dremel, 2010)
  - Apache Hive (HiveQL) (2012)
- Streaming Data
  - Apache Storm (2011) / Twitter Huron (2015)
- Unified Engine (Streaming, SQL, Batch, ML)
  - Apache Spark (2012)

Summary: Big Data Analytics

Data Ingestion
ETL, Distcp, Kafka, OpenRefine, …

Query & Exploration
SQL, Search, Cypher, …

Stream Processing Platforms
Storm, Spark, ..

Batch Processing Platforms
MapReduce, SparkSQL, BigQuery, Hive, Cypher, ...

Data Definition
SQL DDL, Avro, Protobuf, CSV

Storage Systems
HDFS, RDBMS, Column Stores, Graph Databases

Data Serving
BI, Cubes, RDBMS, Key-value Stores, Tableau, …

Computing Platforms
Distributed Commodity, Clustered High-Performance, Single Node

Pointers and Further Reading

Implementations

- Google
  - Not available outside Google
- Hadoop
  - An open-source implementation in Java
  - Uses HDFS for stable storage
  - Download: http://lucene.apache.org/hadoop/

Cloud Computing

- Ability to rent computing by the hour
  - Additional services e.g., persistent storage
- Amazon’s “Elastic Compute Cloud” (EC2)

Readings

- Jeffrey Dean and Sanjay Ghemawat: MapReduce: Simplified Data Processing on Large Clusters
  - http://labs.google.com/papers/mapreduce.html
- Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung: The Google File System
  - http://labs.google.com/papers/gfs.html

Resources

- Hadoop Wiki
  - Introduction
    - http://wiki.apache.org/lucene-hadoop/
  - Getting Started
    - http://wiki.apache.org/lucene-hadoop/GettingStartedWithHadoop
  - Map/Reduce Overview
    - http://wiki.apache.org/lucene-hadoop/HadoopMapReduce
    - http://wiki.apache.org/lucene-hadoop/HadoopMapRedClasses
  - Releases from Apache download mirrors
    - http://www.apache.org/dyn/closer.cgi/lucene/hadoop/
  - Eclipse Environment
    - http://wiki.apache.org/lucene-hadoop/EclipseEnvironment
- Javadoc
  - http://lucene.apache.org/hadoop/docs/api/

Further Reading

- Programming model inspired by functional language primitives
- Partitioning/shuffling similar to many large-scale sorting systems
  - NOW-Sort ['97]
- Re-execution for fault tolerance
  - BAD-FS ['04] and TACC ['97]
- Locality optimization has parallels with Active Disks/Diamond work
  - Active Disks ['01], Diamond ['04]
- Backup tasks similar to Eager Scheduling in Charlotte system
  - Charlotte ['96]
- Dynamic load balancing solves similar problem as River's distributed queues
  - River ['99]