CSP554 BIG DATA TECHNOLOGIES

Module 2a

Hadoop Overview

Big Data Usage is Enabled via New Data Platforms

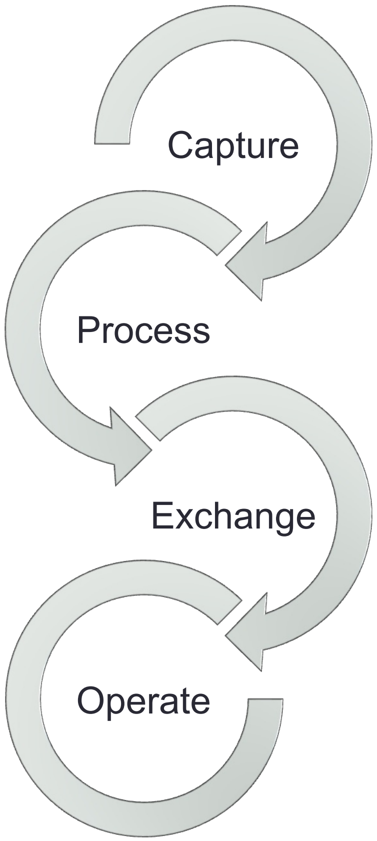
- Facilitate data capture
  - Collect data from all sources - structured and unstructured data
  - At all speeds batch, asynchronous, streaming, real-time
- Comprehensive data processing
  - Transform, refine, aggregate, analyze, report
- Simplified data exchange
  - Interoperate with enterprise data systems
  - Share data with enterprise analytic applications
- Easy to operate platform
  - Provision, monitor, diagnose, manage at scale
  - Reliability, availability, affordability, scalability, interoperability

Big Data Platform: Basic Concept

- A single computational job is too large =>
  - Distribute the computation across multiple commodity servers servers or datacenter or cloud virtual machines
- A single dataset (file) is too large =>
  - Divide the dataset into sections and store them across multiple storage systems

Big Data Platform: Requirements

A big data platform is an integrated set of components that allows you to capture, process and share data in any format at scale



- Store and integrate data sources in real time and batch
- Process and manage data of any size and include tools to sort, filter summarize and apply functions to data
- `Open and promote the exchange of data with new and existing external applications
- Include tools to manage and operate the platform

Big Data Platform: Categories

- Parallel data processing platforms and ecosystems
  - ElasticSearch
    - An open source search engine built on top of Apache Lucene
    - It is Java-based and can search and index document files in diverse formats
  - Apache Hadoop
    - An open source, Java-based programming framework that supports the processing and storage of very large data sets in a distributed computing environment
- Non-relational (NoSQL) distributed database systems
  - Cloud Services
    - Amazon DynamoDB, Azure DocumentDB, Google BigTable
  - Commercial Products
    - MongoDB, Cassandra, Neo4j, Others
- Column-oriented SQL distributed database systems
  - HP Vertica
  - Amazon Redshift

Big Data Platform: Categories

- Column-oriented SQL distributed database systems
  - Cloud Services
    - Amazon Redshift
  - Commercial Products
    - HP Vertica

Our Emphasis

- Apache Hadoop and a selection of its ecosystem of tools
  - Now
- NoSQL databases generally with focus on representative examples
  - Hadoop HBASE now
  - Others later in the course

Hadoop

What is Apache Hadoop?

It is an open source software framework that enables reliable and scalable storage and processing of large datasets in a distributed environment

What is Apache Hadoop?

Definition provided on the official Apache Hadoop's website:

The Apache Hadoop software library is (1) a framework that allows for the distributed processing of large data sets (2) across clusters of computers (3) using simple programming models

It is (4) designed to scale up from single servers to thousands of machines, each offering local computation and storage

Rather than rely on hardware to deliver high-availability, the library itself is (5) designed to detect and handle failures at the application layer

Delivering (6) a highly-available service on top of a cluster of computers, each of which may be prone to failures

Hadoop Concepts

- Hadoop takes a different approach to distributed computing compared to traditional systems
- There are two key notions
  - Distribute data across cluster when it is loaded into the system
  - Run computations across the cluster where this data is stored
- Data is stored on industry-standard commodity hardware
- Add capacity by scaling *out* (more nodes), not scaling *up* (more powerful nodes)
- The Hadoop ecosystem simplifies distributed computing so programmers can focus on the application

What is Apache Hadoop?

The name Apache Hadoop, sometimes, is also used for the family of projects related to distributed computing and Big Data processing, most of which are hosted by Apache Software Foundation.

The Hadoop ecosystem, as it has come to be called, is growing with many projects coming up that complement Hadoop or use Hadoop.

Apache Hadoop Characteristics

- **Scalable**
  - Efficiently store and process petabytes of data
  - Linear scale driven by additional processing and storage
- **Reliable**
  - Redundant storage
  - Failover across nodes and racks
- **Flexible**
  - Store all types of data in any format
  - Apply schema on analysis and sharing of the data
- **Economical**
  - Use commodity hardware
  - Open source software guards against vendor lock-in

The Origins of Hadoop

- Hadoop is based on work done at Google in the late 1990s/early 2000s
- Google's problem:
  - Indexing the entire web requires massive amounts of storage
  - A new approach was required to process such large amounts of data
- Google's solution:
  - GFS, the Google File System
  - Distributed MapReduce

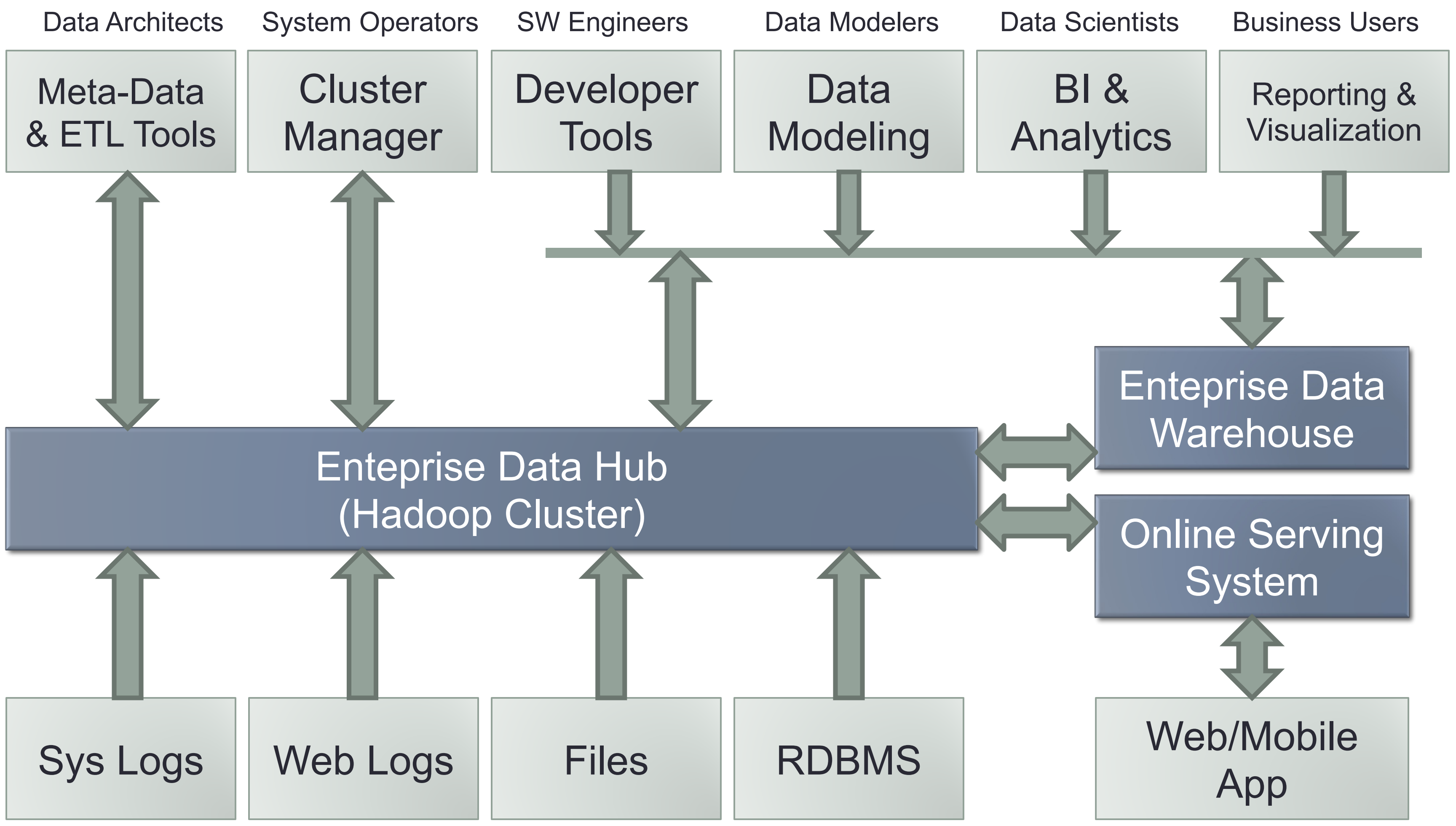
The Origins of Hadoop

- Work described in two papers from Google
  - Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. 2003. The Google file system. In *Proceedings of the nineteenth ACM symposium on Operating systems principles* (SOSP '03). ACM, New York, NY, USA, 29-43.
  - Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: simplified data processing on large clusters. *Commun. ACM* 51, 1 (January 2008), 107-113.
- Doug Cutting read these papers and implemented a similar, open-source solution
  - Initially designed to solve the need to store and compute large numbers of files generated from an open source search engine
- This is what would become Hadoop
  - So Hadoop arose from a need to store and process massive amounts of data in an efficient and cost-effective way

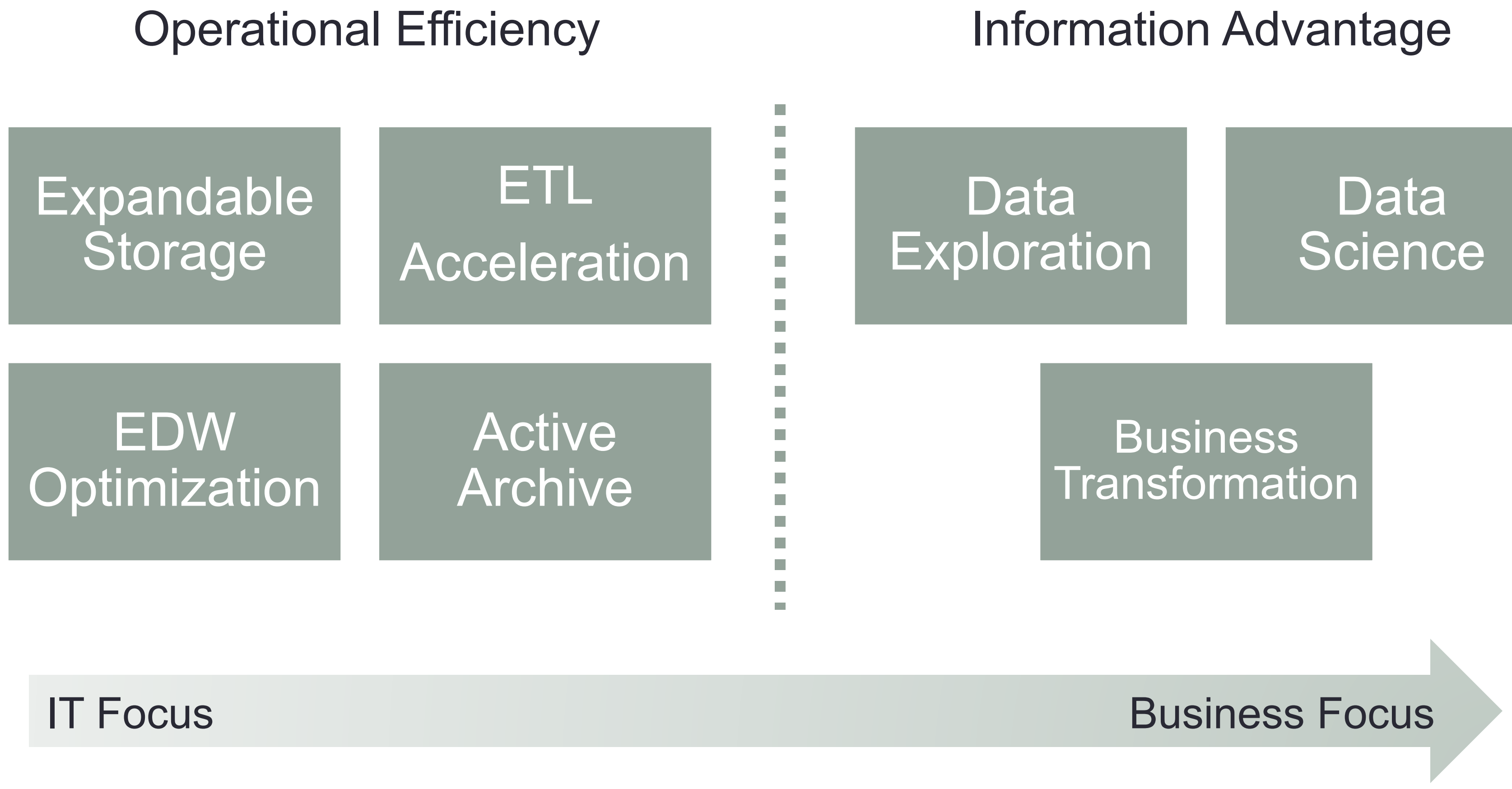
Why Do You Need Hadoop?

- More data is coming
  - Internet of things
  - Sensor data
  - Streaming
- More data means we can ask bigger questions
  - Answer questions that you previously could not ask
- More data means pursue even better answers
- Hadoop scales to store and handle all available data
  - Hadoop is cost-effective
  - Typically provides a significant cost-per-terabyte saving over traditional systems
  - Lets you exploit data you have been throwing away
- Hadoop integrates with existing datacenter components

Integrated Information Architecture



Evolution of Hadoop Use



The Hadoop Ecosystem

- Many tools have been developed around “Core Hadoop”
  - Known as the Hadoop ecosystem
- Designed to make Hadoop easier to use or to extend its functionality
- The majority are open source but supported by various vendors
- The ecosystem is growing all the time, adding extended capabilities
  - Processing frameworks
  - Support for new file types
  - Performance enhancements
  - Self-service visualization, reporting and analytics tools

General Use Cases Supported by Hadoop

- Iterative Exploration
  - Traditional data storage and management systems such as data warehouses, data models, and reporting and analytical tools
  - However, this works well only for business data that be structured, managed, and processed in a relational database
  - Data files are loaded into Hadoop, processed by one or more queries and the output is consumed by the chosen reporting and visualization tools.
  - The cycle repeats using the same data until useful insights have been found, or it becomes clear that there is no useful information available from the data—in which case you might choose to restart the process with a different source dataset.

General Use Cases Supported by Hadoop

- Data warehouse or data mart on demand
  - Use Hadoop to quickly create a basic or ad-hoc data warehouse or mart
  - Create a data repository for very large quantities of data that is relatively low cost to maintain compared to traditional relational database systems, where you do not need the additional capabilities of these types of systems.
  - Consume the results directly in business applications through interactive analytical tools such as Excel or Tableau, or in reporting platforms such as COGNOS
  - Use the Hive tools to support SQL queries on a row and column view of stored data
  - Use HBase to organize data for efficient reading, writing and updating of stored data
  - Use Pig (scripting tool) and Oozie (workflow tool) to organize data transformations

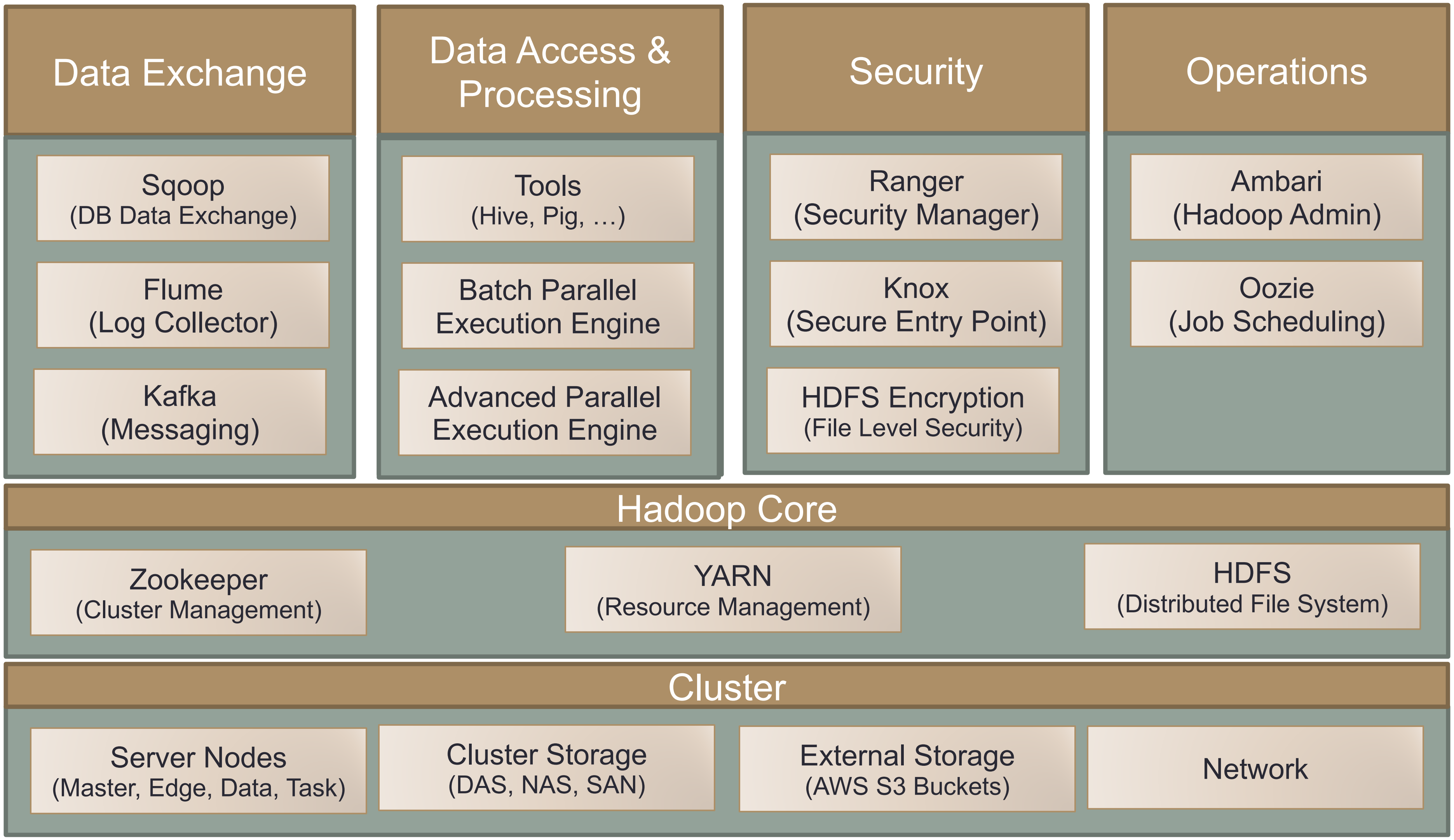
General Use Cases Supported by Hadoop

- ETL Acceleration
  - Use parallel processing to speed up ETL of input data into a relational or NoSQL (Hbase) database
  - Use less costly storage and processing for transforming staging and pre-staging data such as NDW DCV rather than require expensive DB2 licenses
- Flexible Business Intelligence (Reporting/Visualization)
  - Enable analysis and reporting directly from Hadoop supporting advanced users such as business analysts and data scientists to create their own personal analytical data models and reports
- Active Historical Data Archiving
  - Place historical data into HDFS
  - Use compression to reduce space requirements
  - Less costly than storage in DB2 or Oracle
  - Allows queries to the stored data

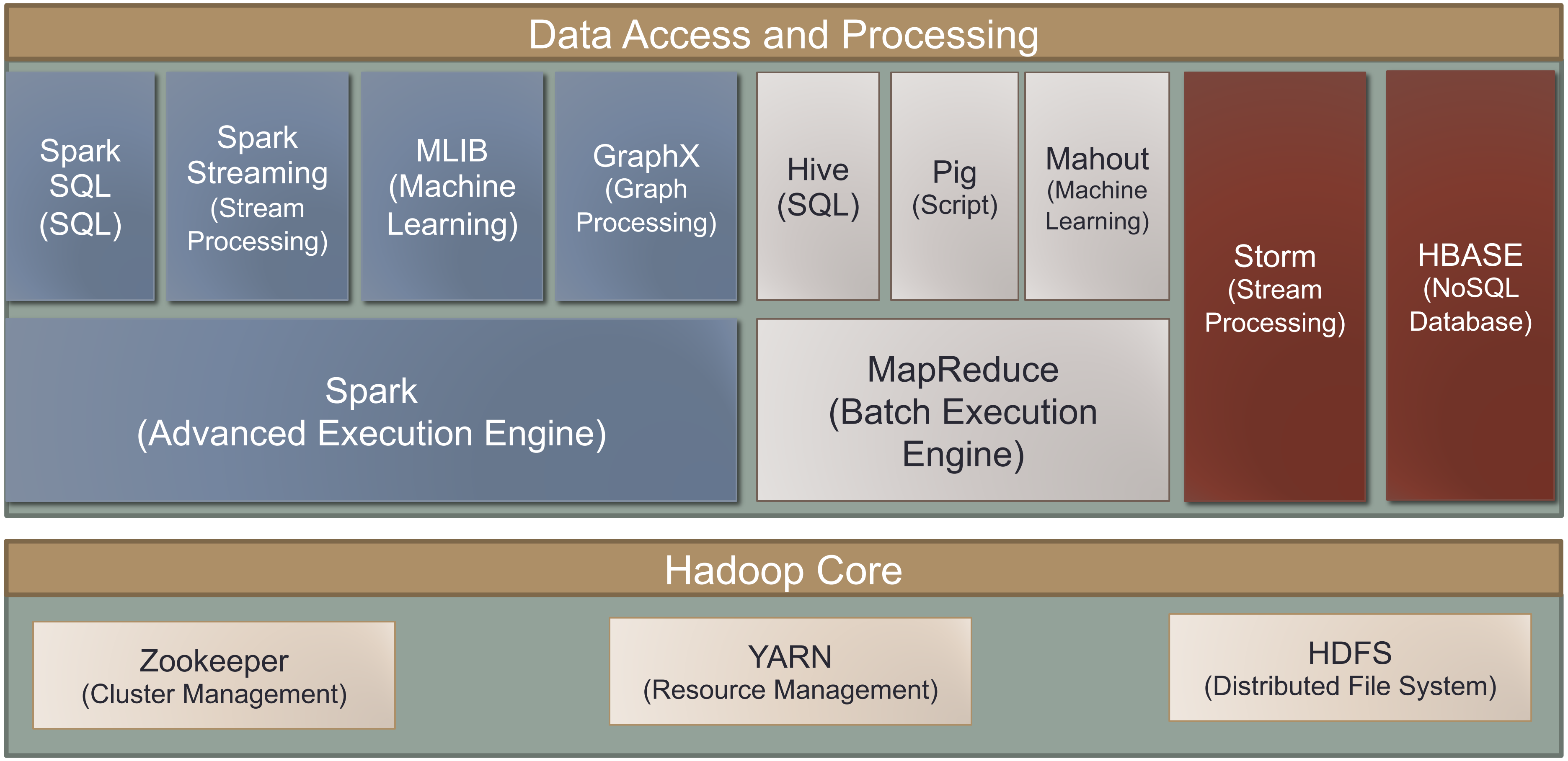
Core Hadoop

- Hadoop Common
  - The common utilities that provide basic support to other Hadoop modules.
- Hadoop Distributed File System
  - Enables storage of large amounts of data in redundancy over a cluster of commodity machines,
- Zookeeper
  - A centralized service for maintaining Hadoop cluster configuration information, naming and providing distributed synchronization
- Hadoop YARN:
  - A framework that takes care of cluster resource management and job scheduling tasks

Hadoop Architecture Landscape


Data Exchange
Sqoop (DB Data Exchange)
Flume (Log Collector)
Kafka (Messaging)
Data Access & Processing
Tools (Hive, Pig, …)
Batch Parallel Execution Engine
Advanced Parallel Execution Engine
Security
Ranger (Security Manager)
Knox (Secure Entry Point)
HDFS Encryption (File Level Security)
Operations
Ambari (Hadoop Admin)
Oozie (Job Scheduling)
Hadoop Core
Zookeeper (Cluster Management)
YARN (Resource Management)
HDFS (Distributed File System)
Cluster
Server Nodes (Master, Edge, Data, Task)
Cluster Storage (DAS, NAS, SAN)
External Storage (AWS S3 Buckets)
Network

Hadoop Architecture Landscape
Data Access Detail


Data Access and Processing
Spark SQL (SQL)
Spark Streaming (Stream Processing)
MLIB (Machine Learning)
GraphX (Graph Processing)
Hive (SQL)
Pig (Script)
Mahout (Machine Learning)
Spark (Advanced Execution Engine)
MapReduce (Batch Execution Engine)
Storm (Stream Processing)
HBASE (NoSQL Database)
Hadoop Core
Zookeeper (Cluster Management)
YARN (Resource Management)
HDFS (Distributed File System)

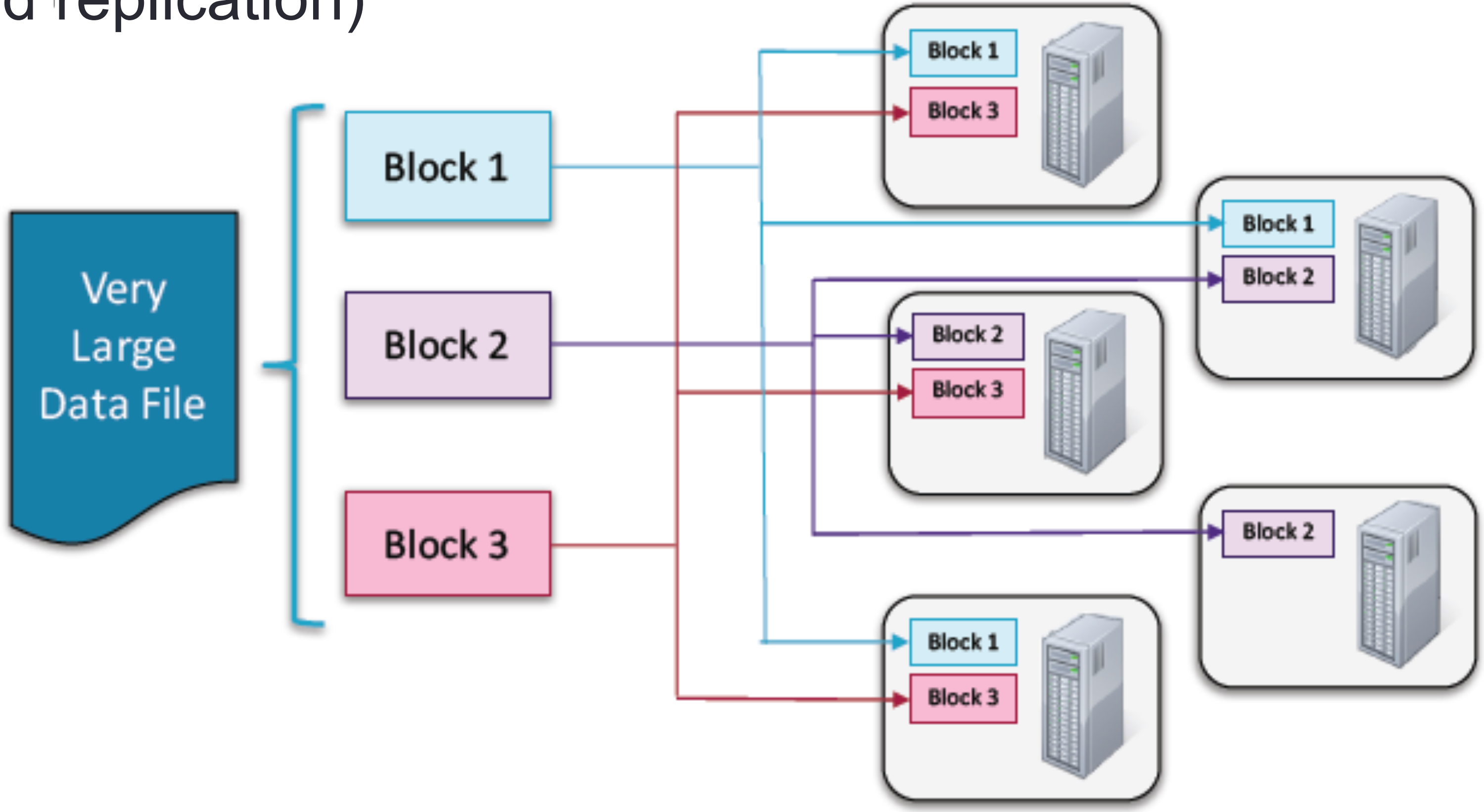
Hadoop Core

The Hadoop Distributed File System (HDFS)

- HDFS is the storage layer for Hadoop
  - A file system which can store any type of data
- Provides inexpensive and reliable storage for massive amounts of data
  - Data is replicated across computers
- HDFS performs best with a “modest” number of large files
  - Millions, rather than billions, of files
  - Each file typically 100MB or more
- File in HDFS are “write once”
  - Appends are permitted
  - But no random writes are allowed

How Files are Stored

- Data files are split into blocks and distributed to data nodes
- Each block is replicated on multiple nodes (default: three-fold replication)



Data Exchange

Sqoop

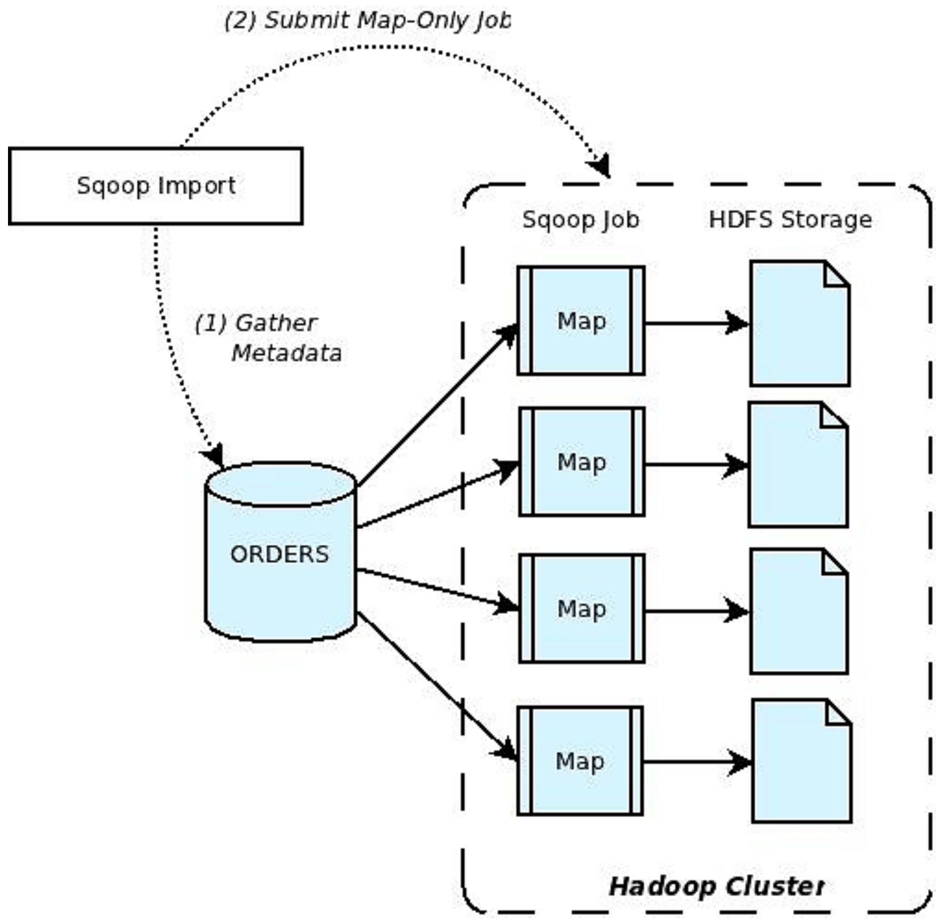
- Allows easy import and export of data from structured data stores
  - Relational databases, NoSQL systems.
- A tool for automating the import and export of data between structured data stores and Hadoop
  - Relational databases (MySQL, Oracle, SQL Server, …)
  - NoSQL Databases (Mongo DB, Cassandra, …)
- Can move data from external systems to HDFS and also populate tables in Hive and HBASE
  - Allows structure definitions to be provisioned into the Hive metastore
- Uses the MapReduce framework to move data in parallel

Sqoop

- SQOOP import
  - Divide table into ranges using primary key max/min
  - Create mappers for each range
  - Mappers write to multiple HDFS nodes
  - Creates text or sequence files
  - Generates Java class for resulting HDFS file
  - Generates Hive definition and auto-loads into HIVE
- SQOOP export
  - Read files in HDFS directory via MapReduce
  - Bulk parallel insert into database table

Sqoop in Action

```
sqoop import connect jdbc:mysql://localhost/db table ORDERS  \
username test password somePassword
```



Data Processing

How Do We Process Big Data?

- Start with multiple nodes each having or controlling access to some amount of persistent (disk) storage
- Divide the file into blocks, then distribute these blocks across each node
- Distribute software to process each block separately to each node which holds a block of the file
- Distribute software to summarize, group and otherwise further process the results from processing each block
- Collect the final result of the separate and the grouped processing

How Do We Process Big Data?

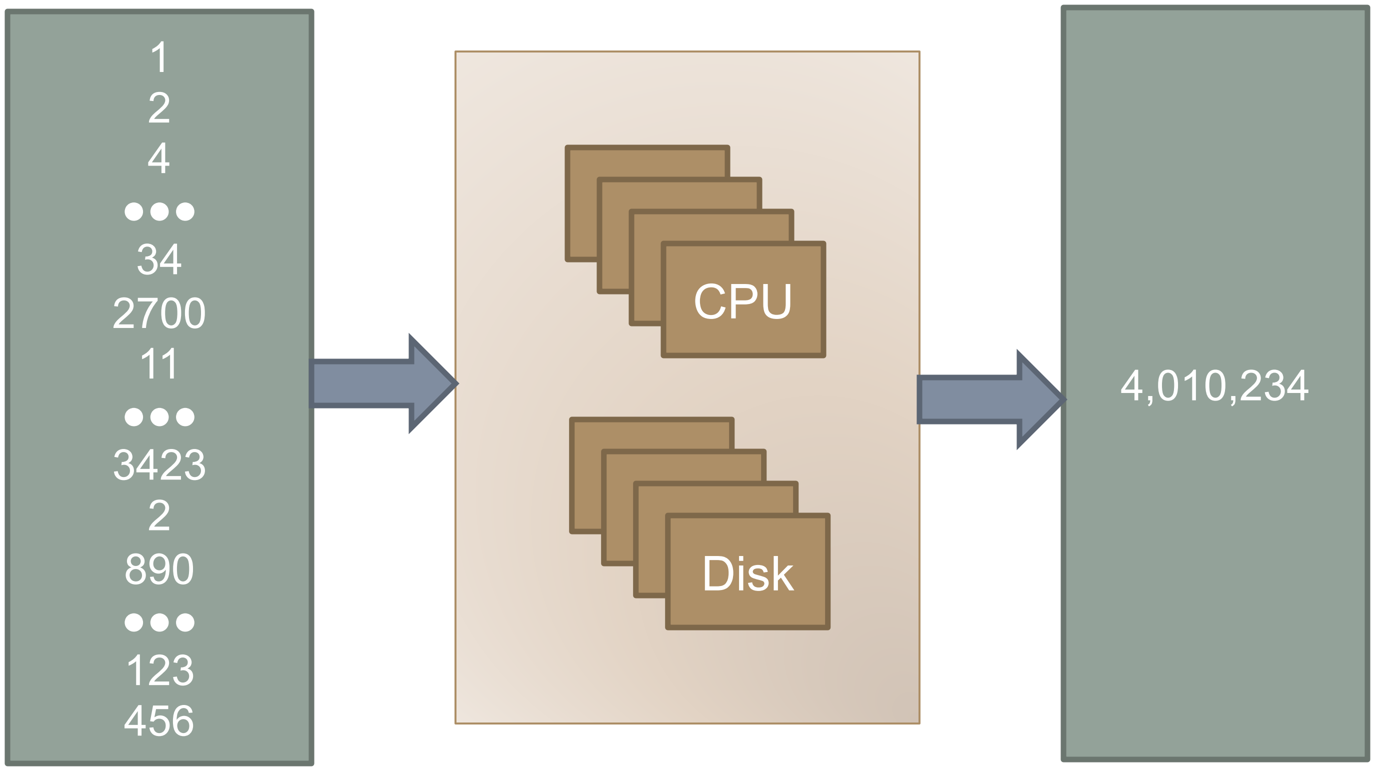
- Think of a file holding a long list of numbers to total..

```
1
2
4
●●●
34
2700
11
●●●
3423
2
890
●●●
123
456
```

- Imagine that adding numbers together is computationally expensive

How Do We Process Big Data?

- We could use one very powerful server to calculate the total…



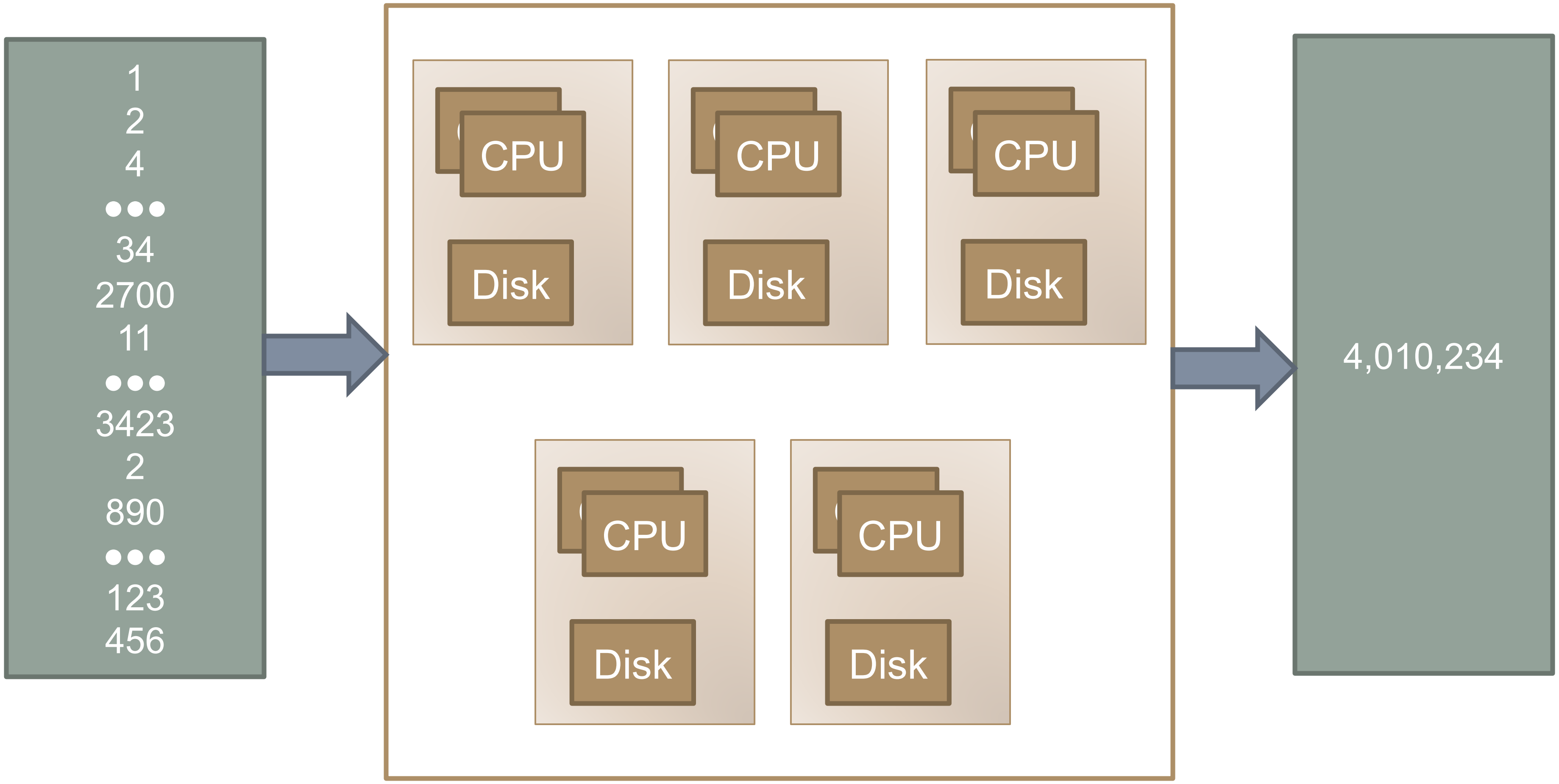
Pseudocode for calculating the total using one very large server

```
Function driver()  {
    Call init() to set the total to 0
    Open some file
    For each record in the file {
            next_record =
                    read next record
                    from the file
            Call map (next_record)
    }
    Call final() to output the total
}
```

```
total = 0

Function init () {
    total = 0
}

Function map (record) {
    value = parse value from record
    total = total + value
}

Function final () {
    output the total
}
```

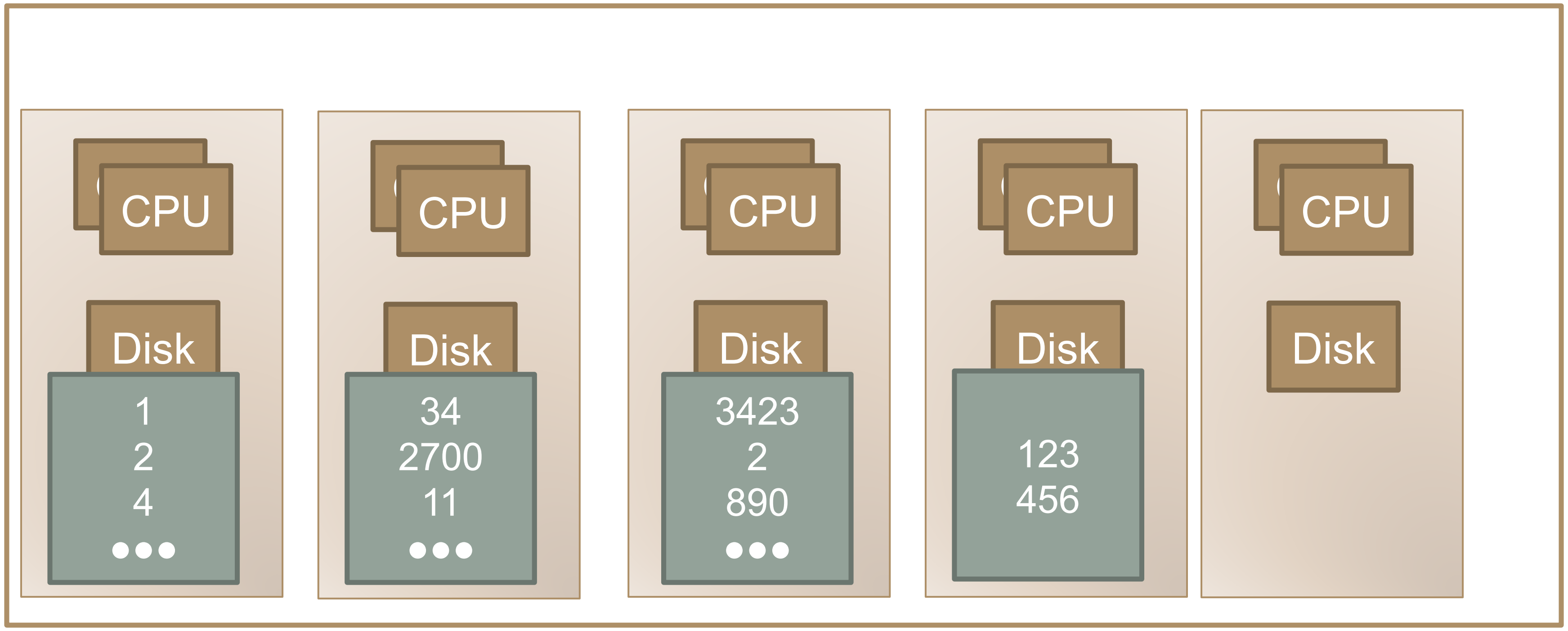
How Do We Process Big Data?

- Or we could use a cluster of less costly but less powerful servers…



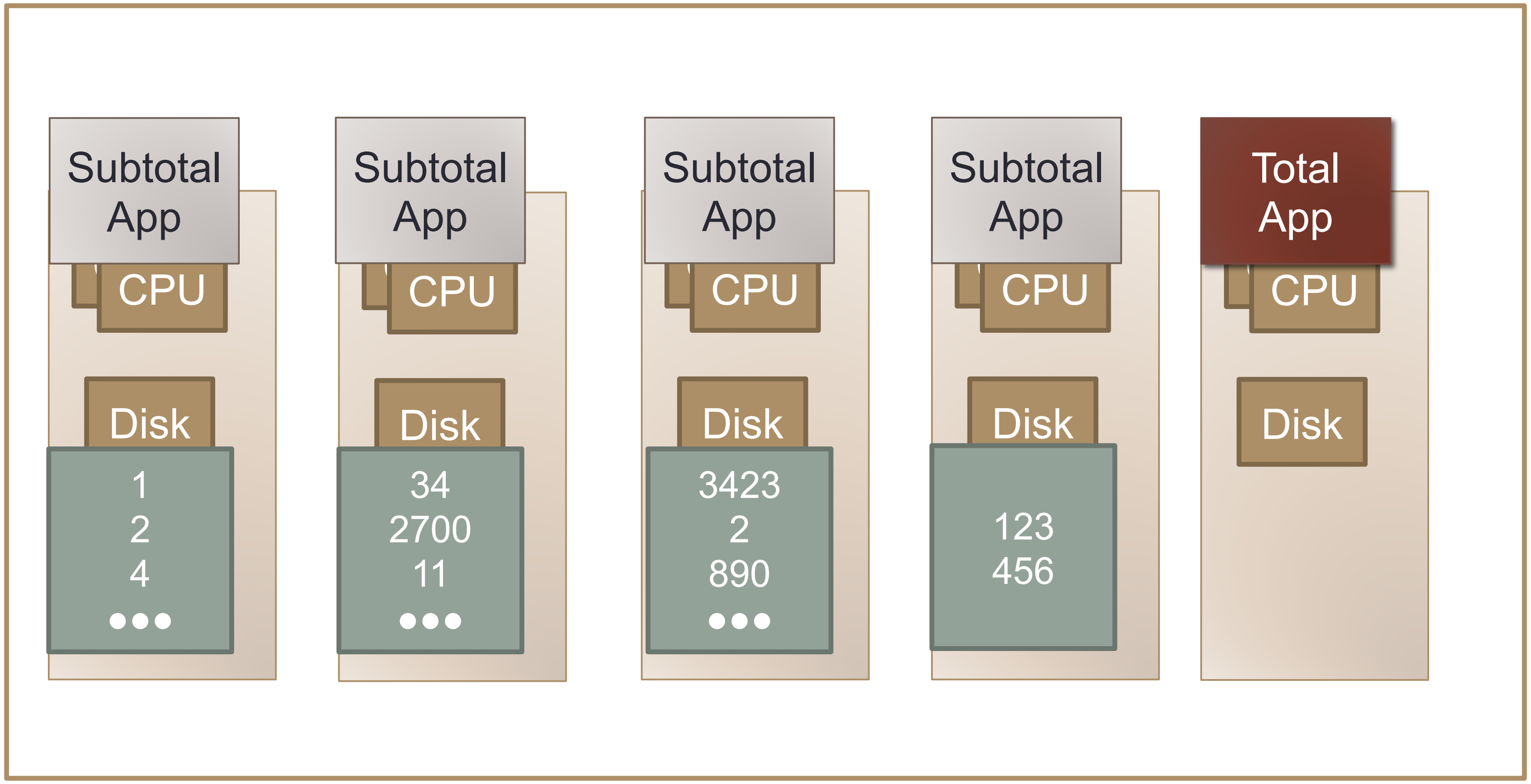
How Do We Process Big Data?

- To do so let's store the file into HDFS to divide the file into blocks, then distribute these blocks across our cluster



How Do We Process Big Data?

Now distribute software to multiple nodes to calculate partial (per block) totals and other software to another node to calculate the overall total



Pseudocode sent to each node to calculate the per block partial totals

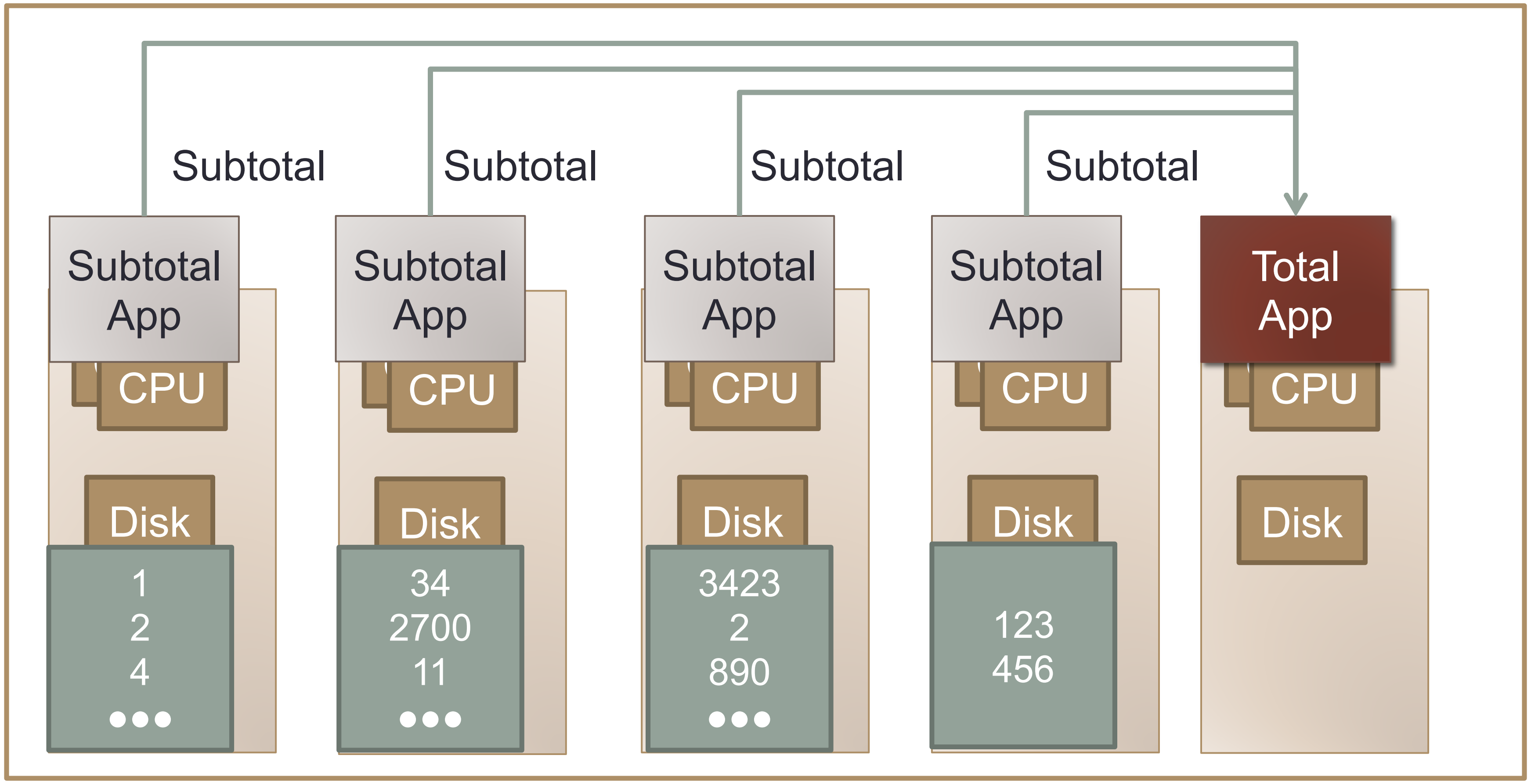
<In parallel on each node>

```
Function driver()  {
    Call init() to set the partial_total to 0

Open some block of an HDFS file
    For each record in the block {
                next_record =
                            read next record
                            from the block
                Call map (next_record)
    }

    Call final() to output the partial_total

    Send value of this partial total
                to the reducer

}
```

```
partial_total = 0

Function init () {
    partial_total = 0
}

Function map (record) {
    value = parse value from record
    partial_total =
            partial_total + value
}

Function final () {
    output the partial_total
}
```

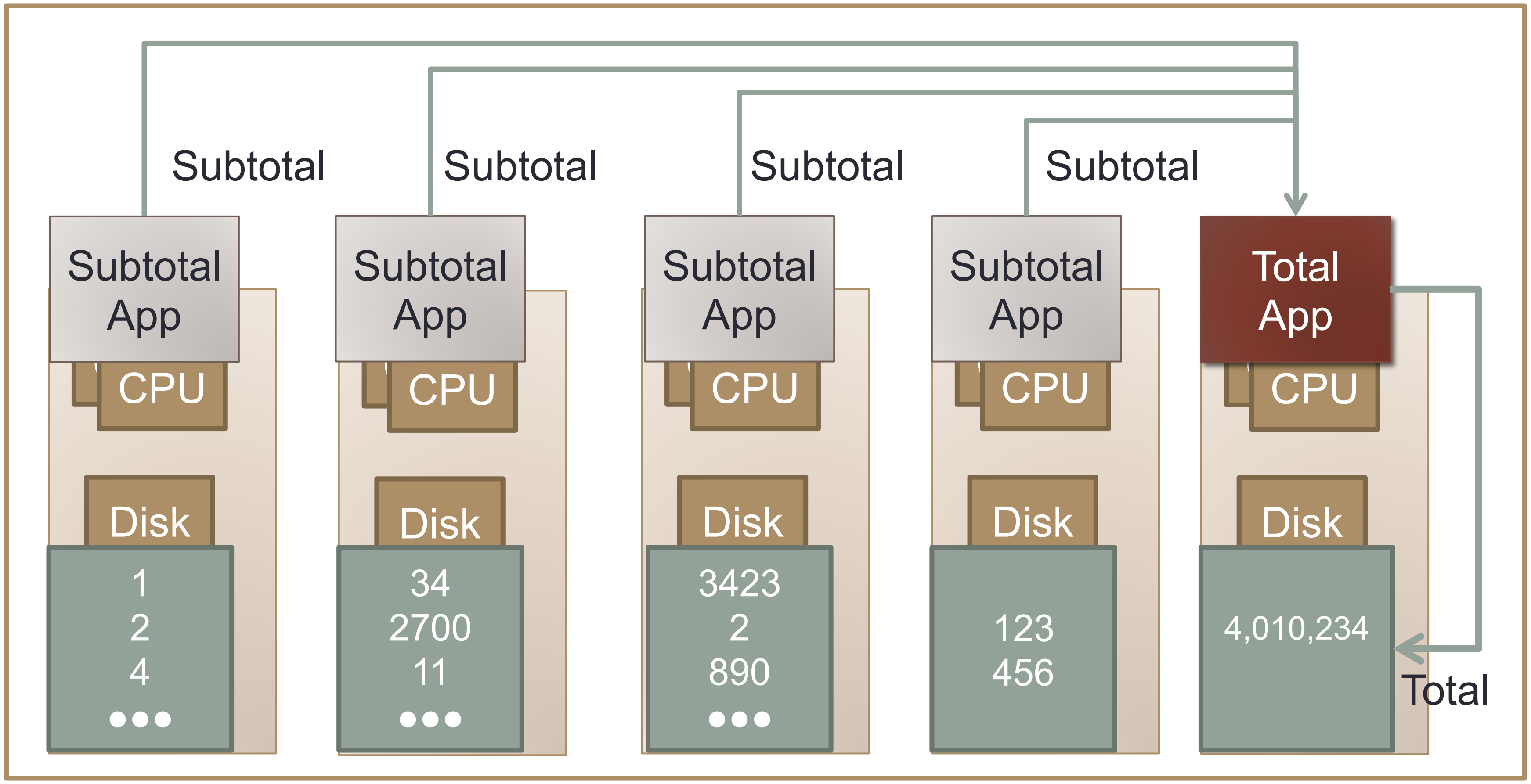
How Do We Process Big Data?

- Now execute all the subtotal apps in parallel and forward the subtotals to total app



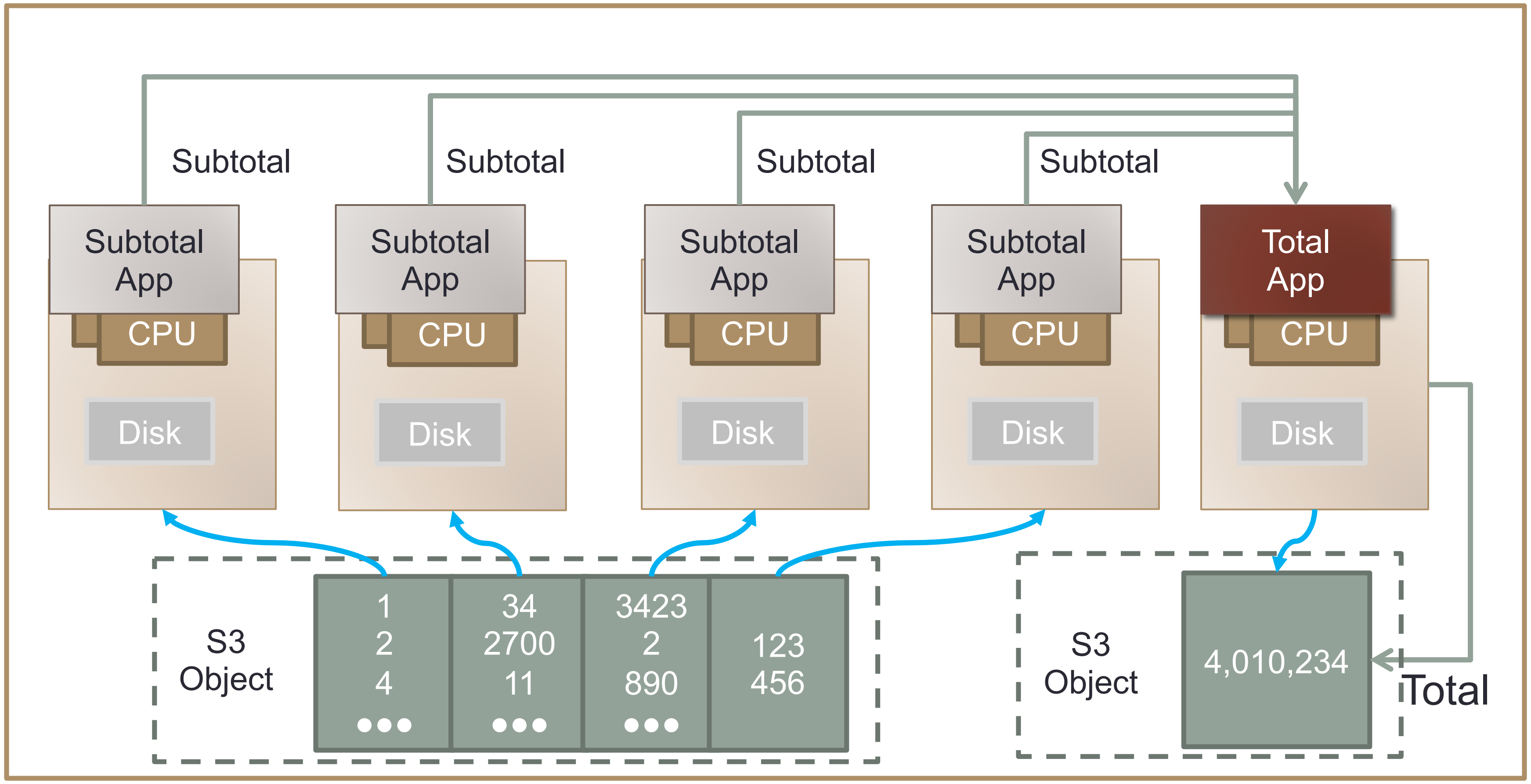
How Do We Process Big Data?

- Now execute the total app to sum the provided subtotals and place the result into another file in the cluster



One more thing… Using S3 Objects

- The S3 API allows you to read just a byte range of an S3 object, so each node is assigned the next 64MB range



Pseudocode for calculating the final total from all the partial totals

```
Function reduce (collection of partial sums) {
    final_total = sum up all partial totals in the collection
    output the final_total
}
```

Data Processing Frameworks

- One data has been loaded into Hadoop and stored in HDFS we want to access and work with that data
- With this, as with every other aspect of Hadoop, we need to know available processing options, before deciding on a specific one
- Such processing options include:
    - The well established MapReduce batch execution engine
    - The newer and much faster Spark advanced execution engine

Some Questions

- We know how to store the file on a large server and total it
- But how do we take advantage of our cluster and move all computation to the data
- How do we store the file to take advantage of clustered servers
  - Answer: Hadoop Distributed File System (HDFS)
- How do we then distribute computation among all these servers
  - Answer: Hadoop MapReduce: Map Phase (or Apache Spark)
- How do we collect the computations carried out on each server to compute a global total
  - Answer: Hadoop MapReduce: Reduce Phase (or Apache Spark)
- How do we perform the totaling operation on the cluster while simultaneously performing other jobs for other users
  - Answer: Apache YARN (Yet Another Resource Negotiator)

MapReduce

- A programming model for processing data sets stored in a distributed manner across a Hadoop cluster’s data nodes
- The key concept for processing large volumes of data using a cluster of nodes is divide and conquer
- You want to break a large data set into many small blocks and processes them in parallel with the same algorithm or pattern
- With HDFS files are already divided into blocks across a number of data nodes
- MapReduce is what you use to move processing code to the data nodes and execute it on each block in parallel

MapReduce

- All MapReduce programs must follow the same format of being organized into multiple phases the most important of which are the map and reduce phases
- The map phase is code that executes on each block of a file and must be distributed to each data node where one of the block is stored
- Mapping applies an algorithm which works on each block independent of all the others to generate some output
- Reducing takes the partial outputs and consolidates them into a combined result

MapReduce Abstractions

- One limitation of the MapReduce capability is that is requires its users to be programmers
- Another is that all problems need to be decomposed into one or more MapReduce steps
  - One needs to “think” in MapReduce terms
  - Often leads to lengthy map reduce programs
- Data analysts and business users could better leverage the power of Hadoop if there was a means to hide some of the complexities of MapReduce
- To address this need there are a growing number of tools such as Apache Hive and Apache Pig that hide the messy details of MapReduce
  - Using a range of abstractions much like compilers hide the details of machine language under high level language abstractions

MapReduce Challenges

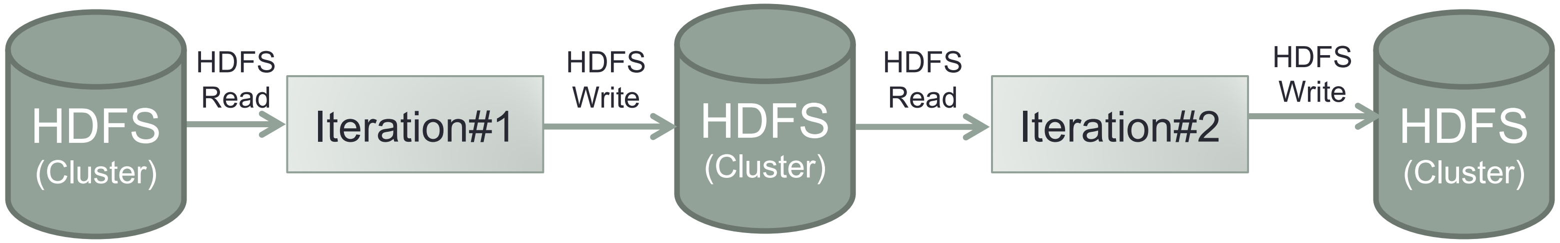
- Very rigid data flow model—map and then reduce
- Not all processing fits easily into this single pattern
- Limits each job to a single map and a single reduce step after which data is written back to HDFS
  - Makes machine learning algorithms which require iteration of several MapReduce jobs lower performing
  - Does take advantage of caching intermediate results to memory between iterations
- Makes MapReduce most appropriate for algorithms which run in a sequence of a few batch jobs
  - But MapReduce is not well suited to iterative jobs or any jobs requiring low latency real-time or interactive characteristics

Spark

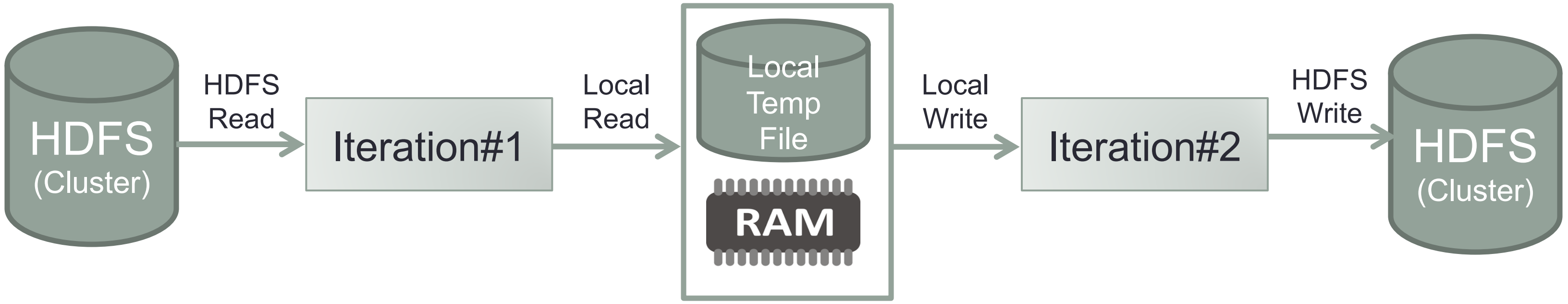
- Apache Spark is a framework extending to real time data analytics in a distributed computing environment.
- Spark is written in Scala and was originally developed at the University of California, Berkeley.
- It executes in-memory computations to increase speed of data processing over MapReduce.
- One of the Spark project goals was to deliver a platform that supports a very wide array of diverse workflows
  - Not only MapReduce batch jobs but also iterative computations like graph algorithms or Machine Learning.
  - And also different scales of workloads from sub-second interactive jobs to jobs that run for many hours.
  - Spark combines batch, interactive, and streaming workloads under one rich concise API.
- If you have large amounts of data that requires low latency processing Spark is a viable alternative

MapReduce Versus Spark

MapReduce
Each iteration (step) is independent of the previous one
Each MapReduce step writes its results back to HDFS



Spark
Each iteration (step) is in a pipeline with the previous one
Each Spark step writes its results to local RAM if possible, else a local temp file



Comparison: MapReduce and Spark

	MapReduce	**Spark**
Developed at	Google	UC Berkeley
Designed for	Batch processing	Batch processing Real-time processing Iterative & interactive operations
Witten in	Java	Scala
In memory processing support	No	Yes
Intermediate results are stored in	Storage	Memory where possible then disk
Fault tolerance	Data replication	Transformation log
Iterative operations organized as	Multiple MapReduce jobs	Single Spark job
Bottleneck	Frequent storage I/O	Large memory consumption

Spark Abstractions

- Spark SQL
  - Allows you to seamlessly mix SQL queries with Spark programs
- Spark Streaming
  - Allows you to build scalable fault-tolerant streaming applications
- Mllib
  - Implements common machine learning algorithms
- GraphX
  - For graph storage and graph-parallel computation.

What is Streaming Data?

- Data that is generated continuously by thousands of data sources (think Google Analytics, Health Trackers, IoT, …)
  - Which typically send in the data records simultaneously, and in small sizes (order of Kilobytes).
- Streaming data includes a wide variety of data
  - Log files generated by mobile or web applications
  - Ecommerce purchases
  - In-game player activity
  - Information from social networks
- Data needs to be processed sequentially and incrementally on a record-by-record basis or over sliding time windows
- Streaming data is used for a wide variety of analytics including correlations, aggregations, filtering, and sampling

Apache Storm

- A system for processing data streams in real time
- One of the biggest fundamental differences between Storm and Spark Streaming…
  - Storm works on individual events and Spark Streaming works on micro-batches.
- Whereas you run "MapReduce jobs", on Storm you run "topologies“
- "Jobs" and "topologies" themselves are very different
  - A MapReduce job eventually finishes, whereas a topology processes messages forever (until you kill it).

Data Access

Hive

- Hive provides a SQL-like language, called HiveQL, for easier analysis of data in Hadoop cluster.
- When using Hive our datasets in HDFS are represented as tables that have rows and columns.
- Hive is easy to learn and appealing to use for those who already know SQL, and have experience in working with relational databases.
- A Hive query is translated into a series of MapReduce jobs that are then executed on a Hadoop cluster

Hive Example

- Let’s process a dataset about songs listened to by users in a given time
- The input consists of a tab-separated file songs.txt

songs.txt
“Creep” Radiohead piotr 2014-07-20
“Desert Rose” Sting adam 2014-07-14
“Desert Rose” Sting piotr 2014-06-10
“Karma Police” Radiohead adam 2014-07-23
“Everybody” Madonna piotr 2014-07-01
“Stupid Car” Radiohead adam 2014-07-18
“All This Time” Sting adam 2014-07-13

Hive Example

Note: We assume that commands below are executed as user “training”

Put song.txt file on HDFS:

```
# hadoop fs -mkdir hdfs:///user/training/songs
# hadoop fs -put songs.txt hdfs:///user/training/songs
```

Enter the beeline hive interactive shell:

```
# beeline…
0: … >
```

Hive Example

Create an external table in Hive that gives a schema to our data on HDFS

```
0: …> CREATE TABLE songs(
title STRING,
artist STRING,
user STRING,
date DATE
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
LOCATION '/user/training/songs';
```

Hive Example

Run a query that finds the two most popular artists in July 2014

```
SELECT artist, COUNT(*) AS total
FROM songs
WHERE year(date) = 2014 AND month(date) = 7
GROUP BY artist
ORDER BY total DESC
LIMIT 2;
```

This query is translated into MapReduce jobs

Pig

- Apache Pig is another popular framework for large-scale computations on Hadoop.
- Similarly to Hive, Pig allows you to implement computations in an easier, faster and less-verbose way than using MapReduce.
- Pig supports data processing through a simple, yet powerful, scripting language called Pig Latin.
- Pig Latin supports many standard data manipulation operations like filtering, aggregating, sorting and joining
- Developers can also extend Pig Latin with their own specialized operations (or install operations from the community)
- Like Hive queries, Pig scripts are translated to MapReduce jobs scheduled to run on the Hadoop cluster

Pig Example

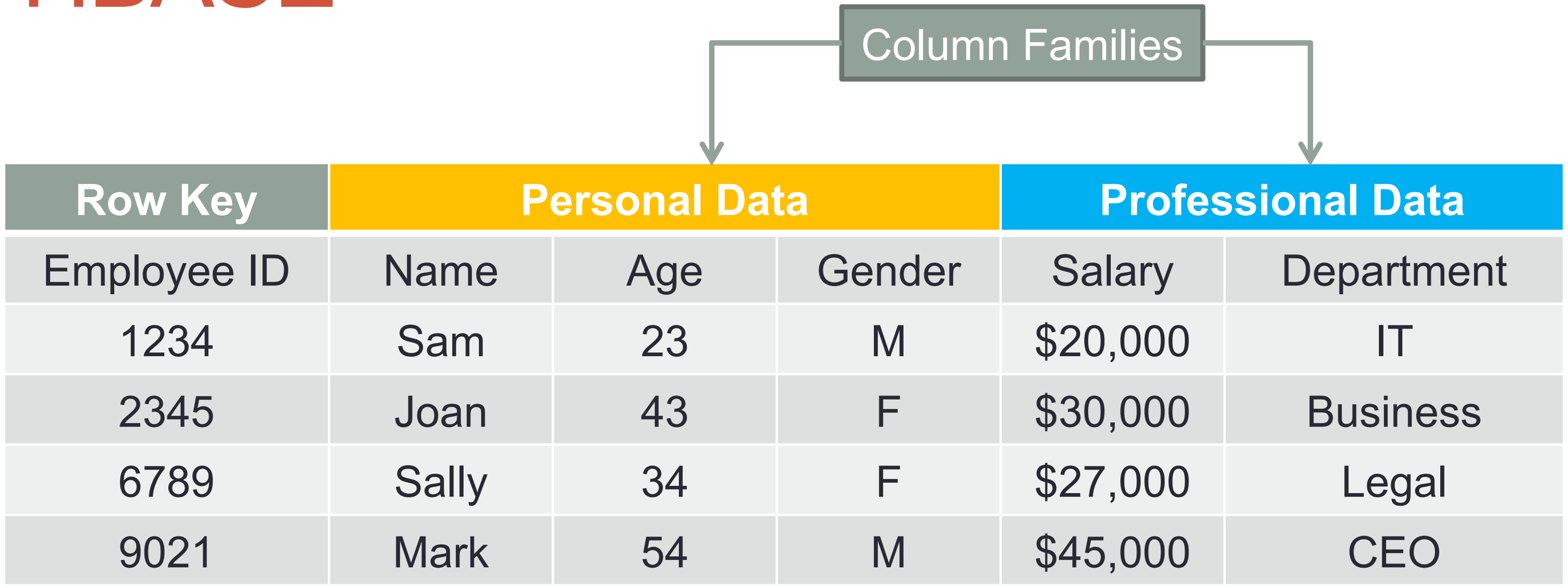
A script that finds the two most popular artists in July 2014

```
a = LOAD ‘songs/songs.txt’ as (title, artist, user, date);
b = FILTER a BY date MATCHES ‘2014-07-.*’;
c = GROUP b BY artist;
d = FOREACH c GENERATE group, COUNT(b) AS total;
e = ORDER d by total DESC;
f = LIMIT e 2;
STORE f INTO ‘top-artists-pig’;
```

HBASE

- A non-relational (NoSQL) database built on top of HDFS
- Adds ability to update data which HDFS & Hive do not
  - Does so on top of an append only file system which makes for an interesting architecture
- Organizes data into tables with each table storing records
  - Allows tables and individual records to be added or deleted
  - Each record is associated with a unique key and given a record's key HBASE can retrieve the data associated with that record
  - Allows fast random record writes and reads in an optimized way
- Reliably supports tables having billions or records
  - Is linearly scalable and also fault tolerant
- Does not support SQL directly; does not support foreign keys or table joins; limited support for indexing

HBASE



Row Key	Personal Data			Professional Data	
Employee ID	Name	Age	Gender	Salary	Department
1234	Sam	23	M	$20,000	IT
2345	Joan	43	F	$30,000	Business
6789	Sally	34	F	$27,000	Legal
9021	Mark	54	M	$45,000	CEO

- Column family-oriented data store
- Each row is indexed by a key you can use for lookup
- Each column family groups like data within rows
- Easy to add new columns and column families

Hadoop to Relational Database Comparison

Relational	Property	Hadoop
Gigabytes to Terabytes	◀ Data Size ▶	Terabytes to Petabytes
Read and Write Many Times	◀ Data Access ▶	Write Once, Read Many Times
Must Be Defined Before Data Is Written	◀ Schema ▶	Optionally Defined Before Data Is Read
Limited (Stored Procedures)	◀ Processing ▶	Parallel Processing Coupled With Data
Structured	◀ Data Types ▶	Structured, Semi-structured and Unstructured
ACID	◀ Transactions ▶	None to Limited