CLOUD COMPUTING APPLICATIONS

MapReduce Introduction

Roy Campbell & Reza Farivar

Motivation: Large Scale Data Processing

- Many tasks composed of processing lots of data to produce lots of other data
- Large-Scale Data Processing
  - Want to use 1000s of CPUs
    - But don’t want hassle of managing things

Motivation: Large Scale Data Processing

- MapReduce provides
  - User-defined functions
  - Automatic parallelization and distribution
  - Fault tolerance
  - I/O scheduling
  - Status and monitoring

CLOUD COMPUTING APPLICATIONS

MapReduce Motivation

Roy Campbell & Reza Farivar

Lesson Outline

- Motivation: Why MapReduce model
- Programming Model
- Examples
  - Word Count
  - Pi Estimation
  - Image Smoothing
  - PageRank
- MapReduce execution

Challenges with Traditional Programming Models (MPI)

- MPI gives you MPI_Send, MPI_Recieve
- Deadlock is possible…
  - Blocking communication can cause deadlock
    - "crossed" calls when trading information
    - example:
      - Proc1: MPI_Receive(Proc2, A); MPI_Send(Proc2, B);
      - Proc2: MPI_Receive(Proc1, B); MPI_Send(Proc1, A);
    - There are some solutions - MPI_SendRecv()
- Large overhead from comm. mismanagement
  - Time spent blocking is wasted cycles
  - Can overlap computation with non-blocking comm.
- Load imbalance is possible! Dead machines?
- Things are starting to look hard to code!

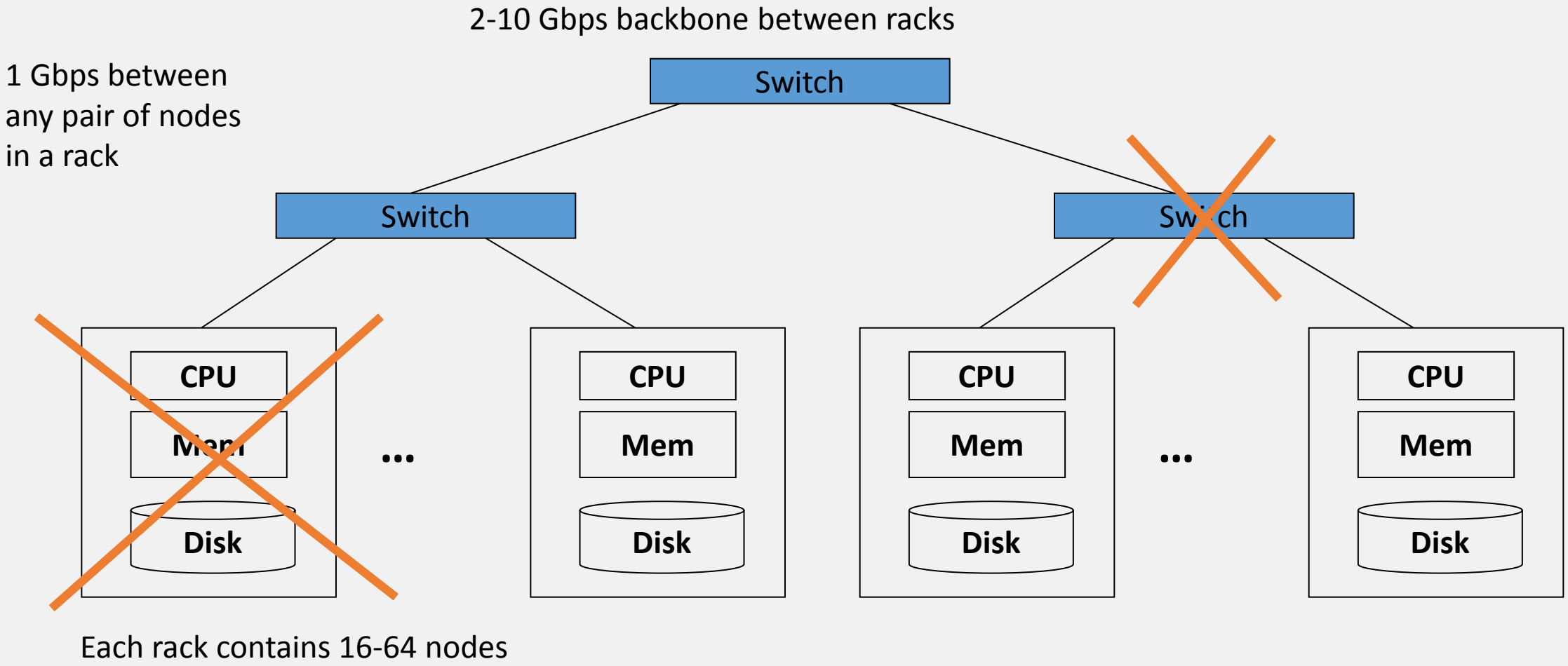
Commodity Clusters

- Web data sets can be very large
  - Tens to hundreds of terabytes
  - Cannot mine on a single server
- Standard architecture emerging:
  - Cluster of commodity Linux nodes
  - Gigabit Ethernet interconnect
- How to organize computations on this architecture?
  - Mask issues such as hardware failure

Solution

- Use distributed storage
  - 6-24 disks attached to a blade
  - 32-64 blades in a rack connected by Ethernet
- Push computations down to storage
  - Computations process contents of disks
  - Data on disks read sequentially from beginning to end
  - Rate limited by speed of disks (speed can get at data)

Cluster Architecture



Stable Storage

- First-order problem: if nodes can fail, how can we 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 is rarely updated in place
  - Reads and appends are common

CLOUD COMPUTING APPLICATIONS

MapReduce Programming Model

Roy Campbell & Reza Farivar

What is MapReduce?

- MapReduce
  - Programming model from LISP
- Many problems can be phrased this way
- **Easy** to distribute
  - Hides difficulty of writing parallel code
  - System takes care of load balancing, dead machines, etc.
- Nice retry & failure semantics

Programming Concept

- Map
  - **Perform** a function on **individual values** in a data set to create a **new list** of values
- Reduce
  - **Combine** values in a data set to create a new **value**

SQUARE X = X * X **MAP** SQUARE [1,2,3,4,5] RETURNS [1,4,8,16,25]	SUM= [All Elements in Array, Total+=] REDUCE [1,2,3,4,5] RETURNS 15

MapReduce Programming Model

Input & Output: Each a set of key/value pairs

Programmer specifies two functions:

MAP (IN_KEY, IN_VALUE)

LIST(OUT_KEY, INTERMEDIATE_VALUE)

- Processes input key/value pair
- Produces intermediate pairs

MapReduce Programming Model

Input & Output: Each a set of key/value pairs

Programmer specifies two functions:

REDUCE (IN_KEY, IN_VALUE)

LIST(OUT_KEY, INTERMEDIATE_VALUE)

- Combines all intermediate values for a particular key
- Produces a set of merged output values (usually just one)

CLOUD COMPUTING APPLICATIONS

MapReduce Example: Word Count

Roy Campbell & Reza Farivar

Word Count

- Have a large file of words, many words in each line
- Count the number of times each distinct word appears in the file

Word Count Program

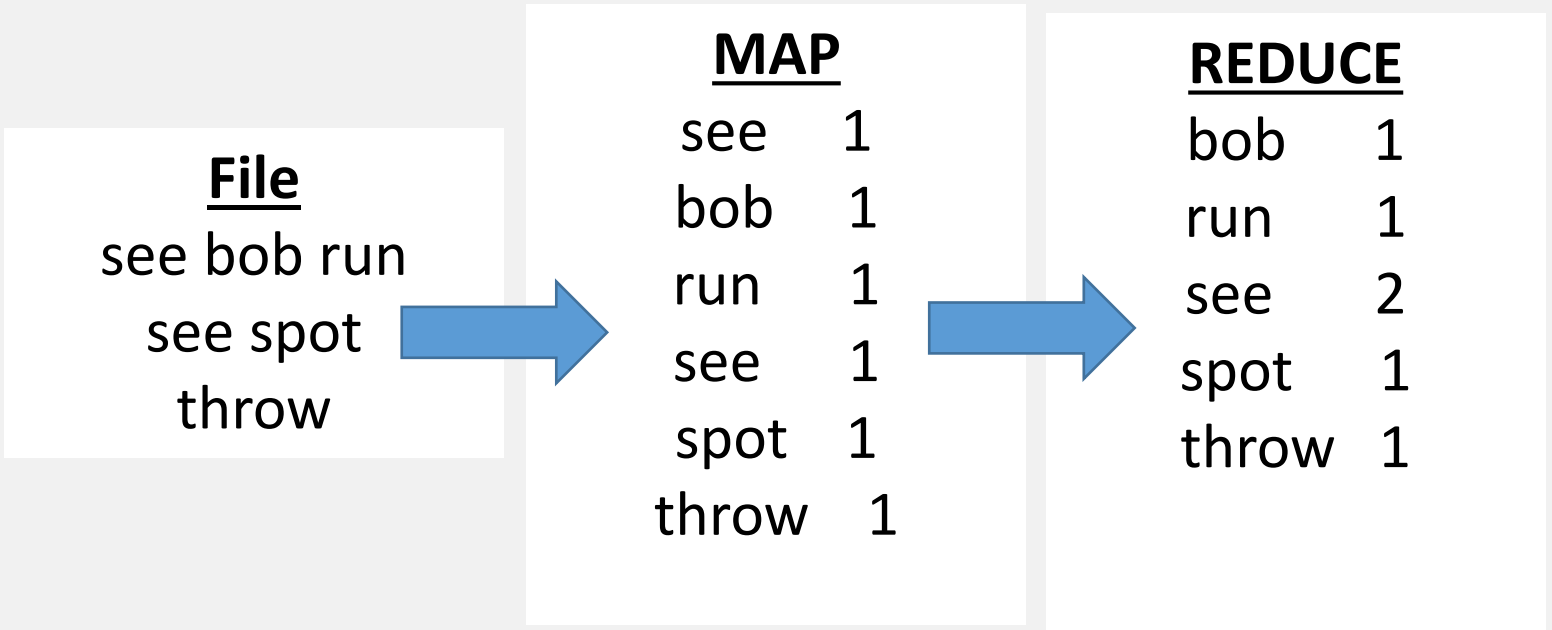
MAP(KEY = LINE, **VALUE** = **CONTENTS**):
    FOR EACH WORD W IN VALUE:
        EMIT INTERMEDIATE(w,1)

REDUCE(KEY, VALUES):
 //key: a word; values: an iterator
//over counts
    RESULT = 0
    FOR EACH V IN **INTERMEDIATE VALUES**:
        RESULT += V
     EMIT(KEY, RESULT)

Word Count Illustrated

```
map(key=line, values=contents):
   for each word w in contents:
    emit(w,"1")
```

```
reduce(key=line, values=uniq_counts):
   sum all 1's in values list
     emit result (word, sum)
```



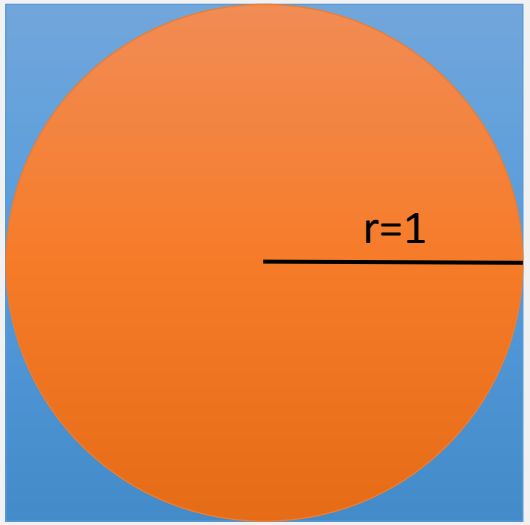
CLOUD COMPUTING APPLICATIONS

MapReduce Example: Pi Estimation & Image Smoothing

Roy Campbell & Reza Farivar

Pi Estimation

- Using Monte Carlo simulation, estimate the value of π
- Throw darts
- Compute the ratio of the darts landed within the square vs. the darts landed within the circle
- Evaluating whether a particular dart landed within the circle is easy



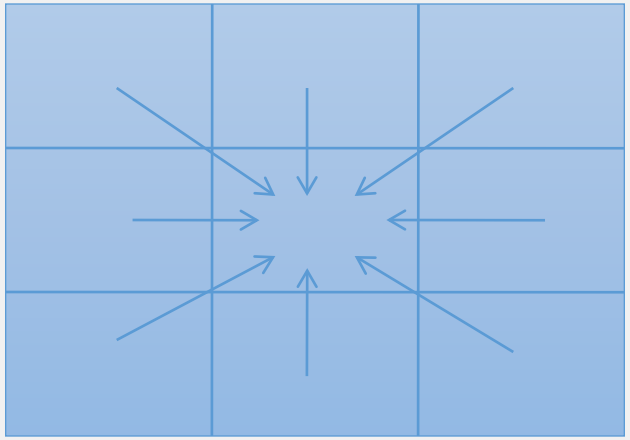
Circle Area = π
Square Area = 4
π=4.C/S

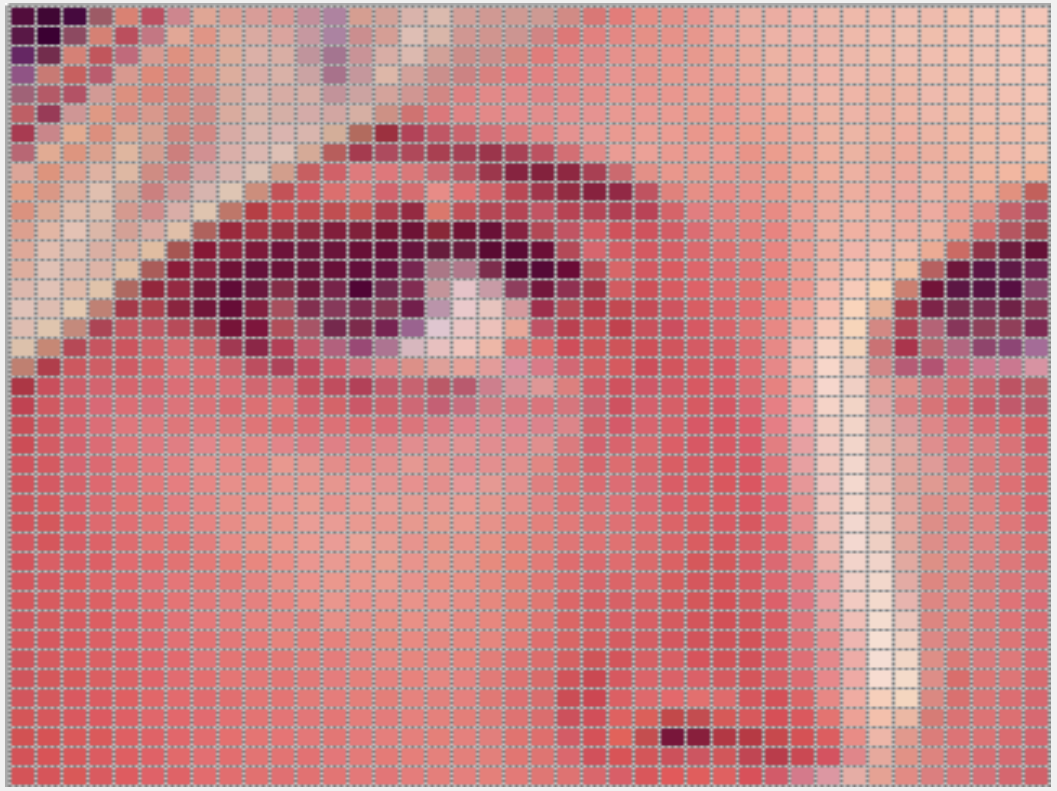
Pi Estimation

- **Mapper:** Generate points in a unit square and then count points inside/outside of the inscribed circle of the square
- **Reducer:** Accumulate points inside/outside results from the mappers
- After the MapReduce job, estimate Pi
  - The fraction **NumInside/NumTotal** is an approximation of the value
    - (Area of the circle) / (Area of the square)
  - Then, Pi estimated value to be **(NumInside/NumTotal)**

Exercise 2: Image Smoothing

- To smooth an image, use a sliding mask and replace the value of each pixel





Exercise 2: Image Smoothing

- Map: input key = x, y input value = R, G, B
  - Emit 9 points
    - (x-1, y-1, R, G, B)
    - (x, y-1, R, G, B)
    - (x+1, y-1, R, G, B)
    - Etc.
- Reduce: input key = x, y input value: list of R, G, B
  - Compute average R, G, B
  - Emit key = x, y value = average R, G, B

CLOUD COMPUTING APPLICATIONS

MapReduce Example: Page Rank

Roy Campbell & Reza Farivar

PageRank Algorithm

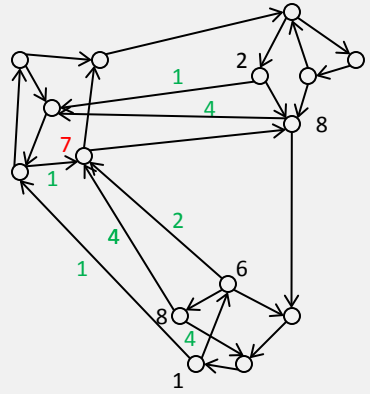
- Program implemented by Google to rank any type of recursive “documents” using MapReduce
- Initially developed at Stanford University by Google founders, Larry Page and Sergey Brin, in 1995
- Led to a functional prototype named Google in 1998

- PageRank value for a page *u* is dependent on the PageRank values for each page *v* out of the set B_u (all pages linking to page *u*), divided by the number *L(v)* of links from page *v*

$$PR(u) = \sum_{v \in B_u} \frac{PR(v)}{L(v)}$$

PageRank

- Phase 1: Propagation
- Phase 2: Aggregation



- Input: A pool of objects, including both vertices and edges

PageRank: Propagation

- Map: for each object
  - If object is vertex, emit key=URL, value=object
  - If object is edge, emit key=source URL, value=object
- Reduce: (input is a web page and all the outgoing links)
  - Find the number of edge objects→outgoing links
  - Read the PageRank value from the vertex object
  - Assign PR(edges)=PR(vertex)/num_outgoing

PageRank: Aggregation

- Map: for each object
  - If object is vertex, emit key=URL, value=object
  - If object is edge, emit key=destination URL, value=object
- Reduce: (input is a web page and all the incoming links)
  - Add the PR value of all incoming links
  - Assign PR(vertex)=$\sum$PR(incoming links)

CLOUD COMPUTING APPLICATIONS

Summary

Roy Campbell & Reza Farivar

MapReduce Advantages/Disadvantages

- Now it's easy to program for many CPUs
  - Communication management effectively gone
    - I/O scheduling done for us
  - Fault tolerance, monitoring
    - Machine failures, suddenly slow machines, etc., are handled
  - Can be much easier to design and program
  - Can cascade several MapReduce tasks
- But … it further restricts solvable problems
  - Might be hard to express problem in MapReduce
  - Data parallelism is key
    - Need to be able to break up a problem by data chunks

MapReduce Conclusions

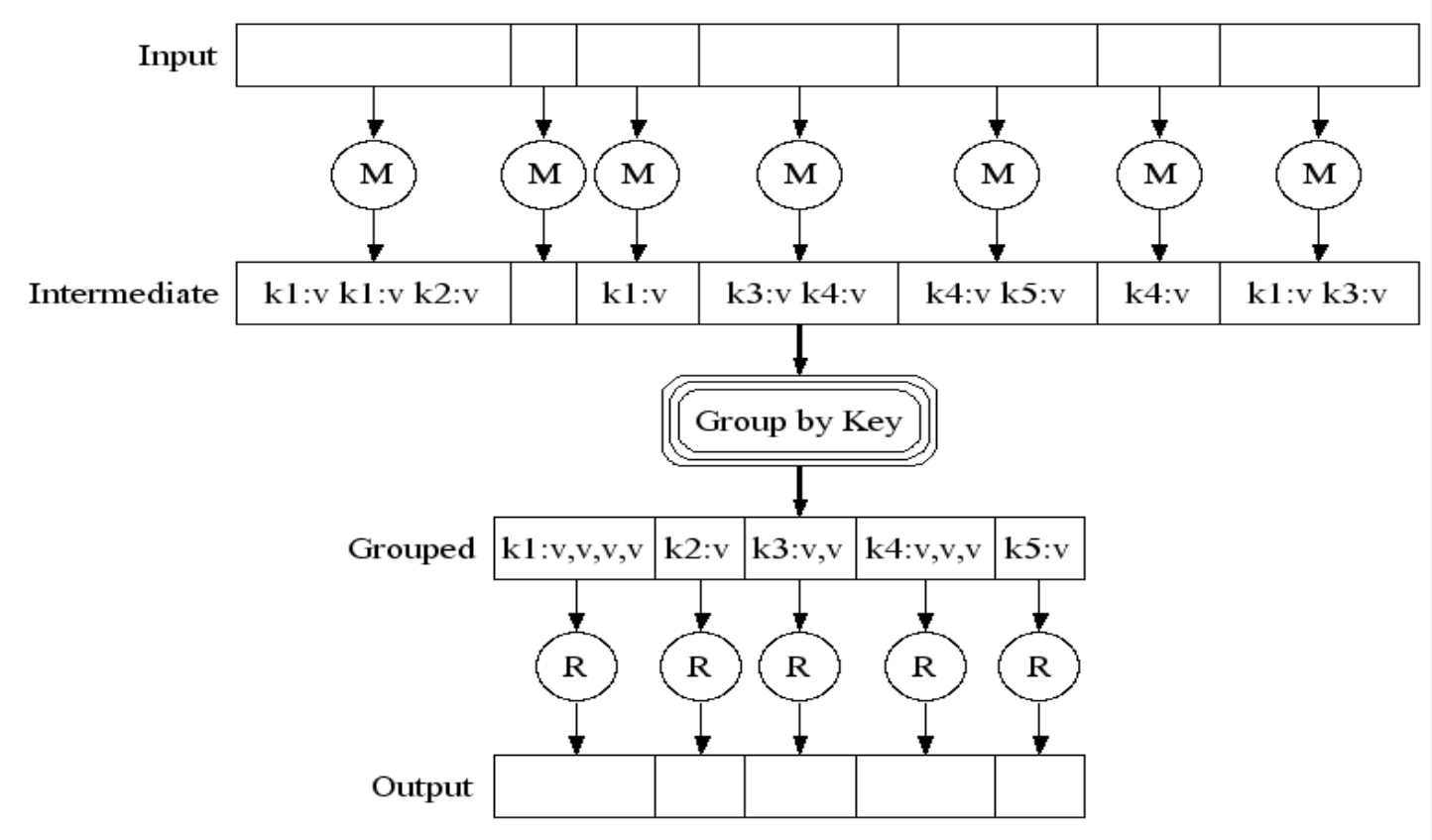
- MapReduce has proven to be a useful abstraction
- Greatly simplifies large-scale computations
- Functional programming paradigm can be applied to large-scale applications
- Fun to use: focus on problem, let the middleware deal with messy details

CLOUD COMPUTING APPLICATIONS

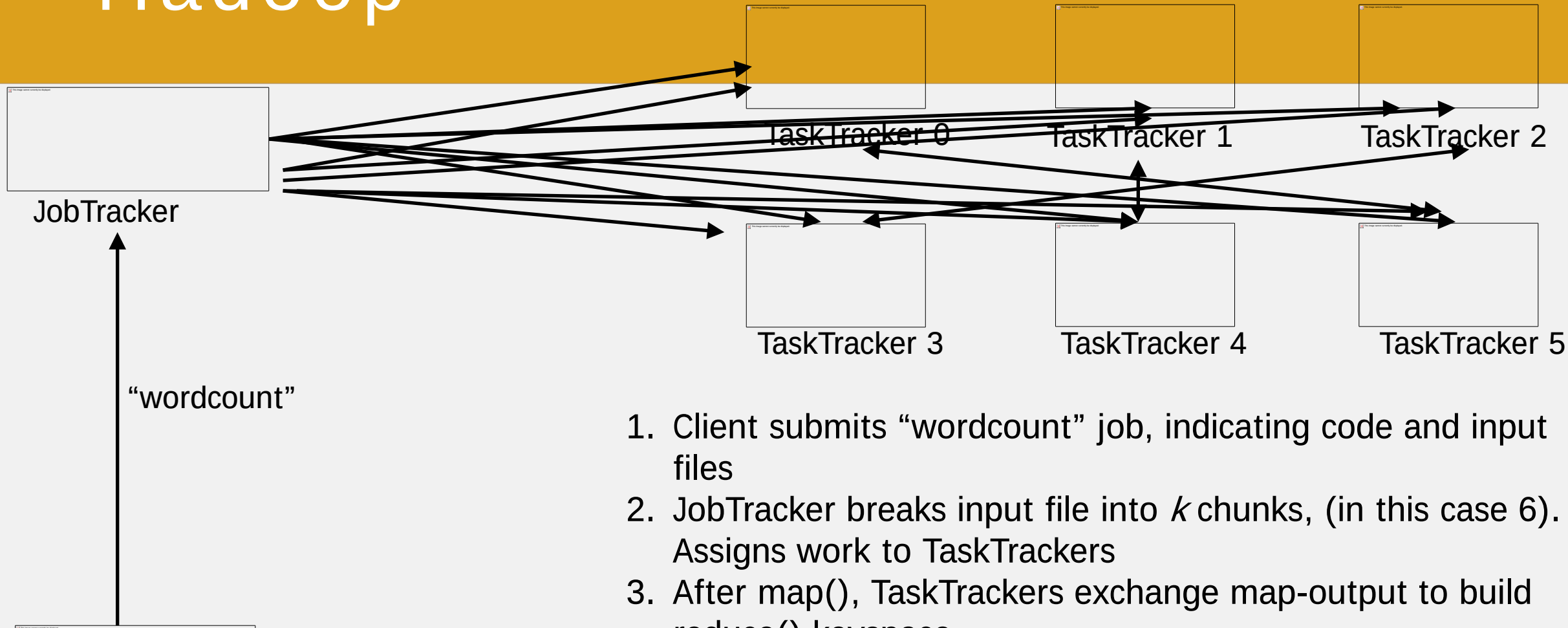
Hadoop Introduction

Roy Campbell & Reza Farivar

Execution


Input
M
M
M
M
M
M
M
Intermediate
k1:v k1:v k2:v
k1:v
k3:v k4:v
k4:v k5:v
k4:v
k1:v k3:v
Group by Key
Grouped
k1:v,v,v,v
k2:v
k3:v,v
k4:v,v,v
k5:v
R
R
R
R
R
Output

Hadoop



1. Client submits “wordcount” job, indicating code and input files
2. JobTracker breaks input file into *k* chunks, (in this case 6). Assigns work to TaskTrackers
3. After map(), TaskTrackers exchange map-output to build reduce() keyspace
4. JobTracker breaks reduce() keyspace into *m* chunks (in this case 6). Assigns work
5. reduce() output may go to HDFS

Execution Initialization

- Split input file into 64MB sections (GFS)
  - Read in parallel by multiple machines
- Fork off program onto multiple machines
- One machine is Master
- Master assigns idle machines to either Map or Reduce tasks
- Master coordinates data communication between map and reduce machines

Partition Function

- Inputs to map tasks are created by contiguous splits of input file
- For reduce, we need to ensure that records with the same intermediate key end up at the same worker
- System uses a default partition function e.g., hash(key) mod R
- Sometimes useful to override

  e.g., hash(hostname(URL)) mod R ensures URLs from a host end up in the same output file

Map-Machine

- Reads contents of assigned portion of input file
- Parses and prepares data for input to map function (e.g., read <a /> from HTML)
  - Classes implementing InputFormat
- Passes data into map function and saves result in memory (e.g., <target, source>)
- Periodically writes completed work to local disk
- Notifies Master of this partially completed work (intermediate data)

Reduce-Machine

- Receives notification from Master of partially completed work
- Retrieves intermediate data from Map-Machine via remote-read
- Sorts intermediate data by key (e.g., by target page)
- Iterates over intermediate data
  - For each unique key, sends corresponding set through reduce function
- Appends result of reduce function to final output file (GFS)

Data Flow

- Input, final output are stored on a distributed file system
  - 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 map reduce task

CLOUD COMPUTING APPLICATIONS

BIG DATA PIPELINES: THE MOVE TO HADOOP

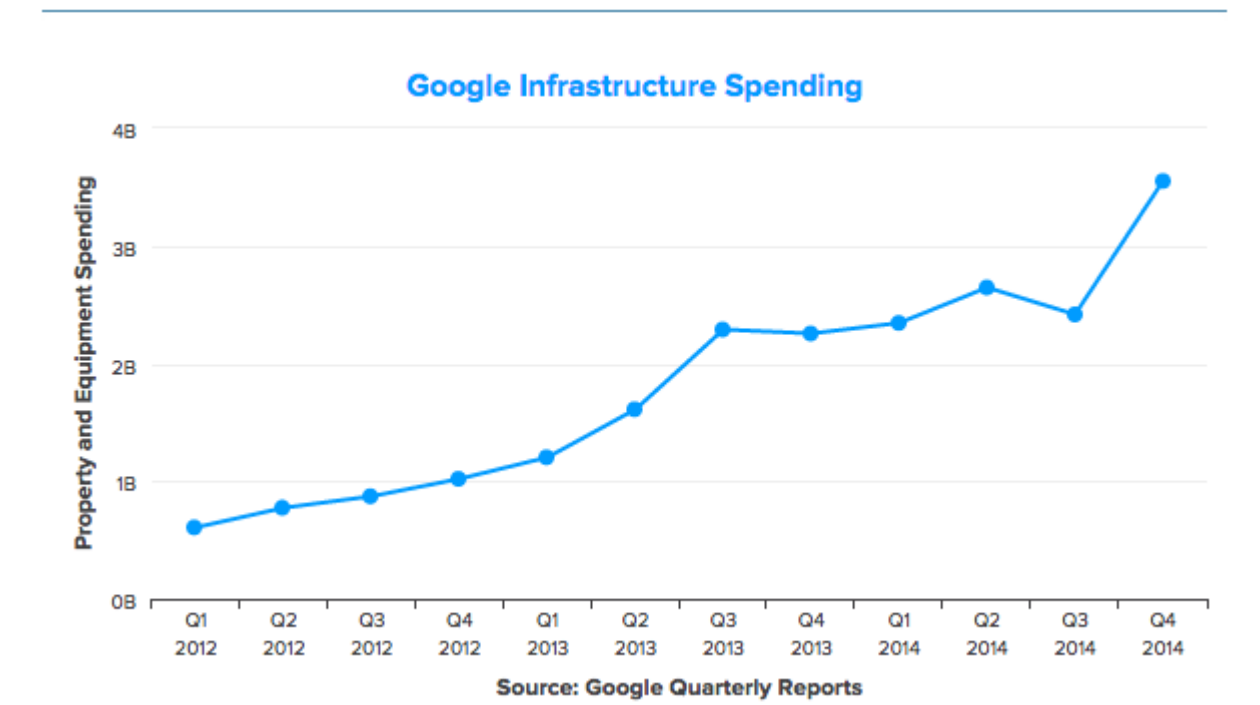
Matt Ahrens – Yahoo

Why Pipelines Are Behind Everything

- With the rise of large data sets, there needs to be a system that can reliably and quickly organize the data
- “Big data” is the trend in the industry, but how do you actually obtain data that is useful on a regular basis?
- Use cases
  - Relevant content tailored to users
  - Programmatic digital advertising
  - Data analytics for research and sciences

Why Pipelines Are Behind Everything

- Data keeps growing



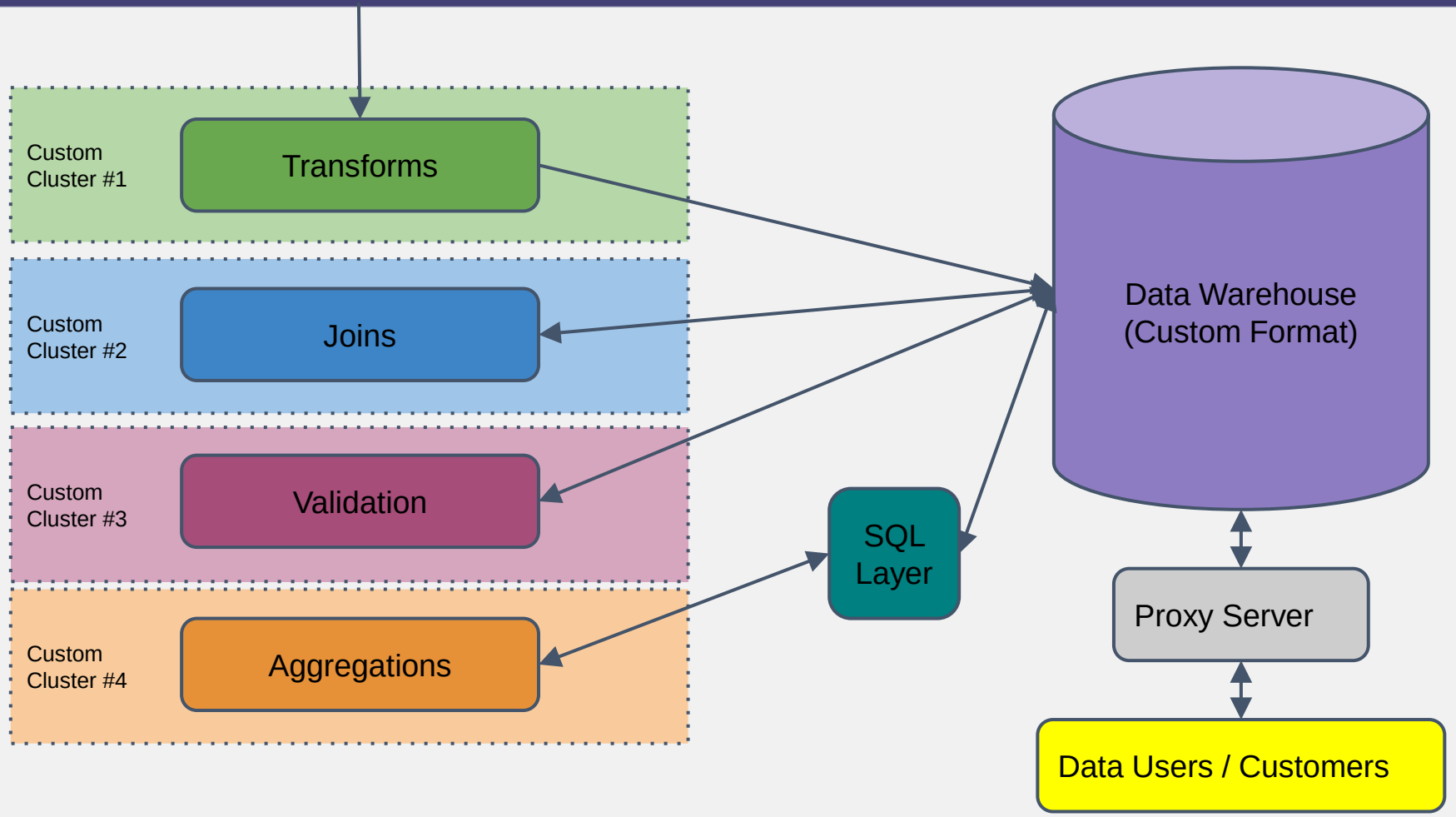
What Is a Data Pipeline

- Simple definition: system that transforms events into a usable format
- Input: raw logs, interactions, activities
- Output: data sets for specific users (filtered, aggregated, joined, etc.)
- Data size scale
  - Billions of transactions per day (millions / minute)
  - TBs of data per day (GBs / minute)

Where We Came From

- Customized mini-clusters of hardware
  - Tailored to specific type of job: transformation, joins, aggregation
  - Pro: mix of memory/cpu config specific for job type
  - Con: scale issues, overhead of HW setup/maintenance
- Lack of well-defined interfaces and APIs
  - No standard schema format or data model
- Data access limitations
  - Access was limited to core developers with advanced data and programming knowledge

The Past Architecture

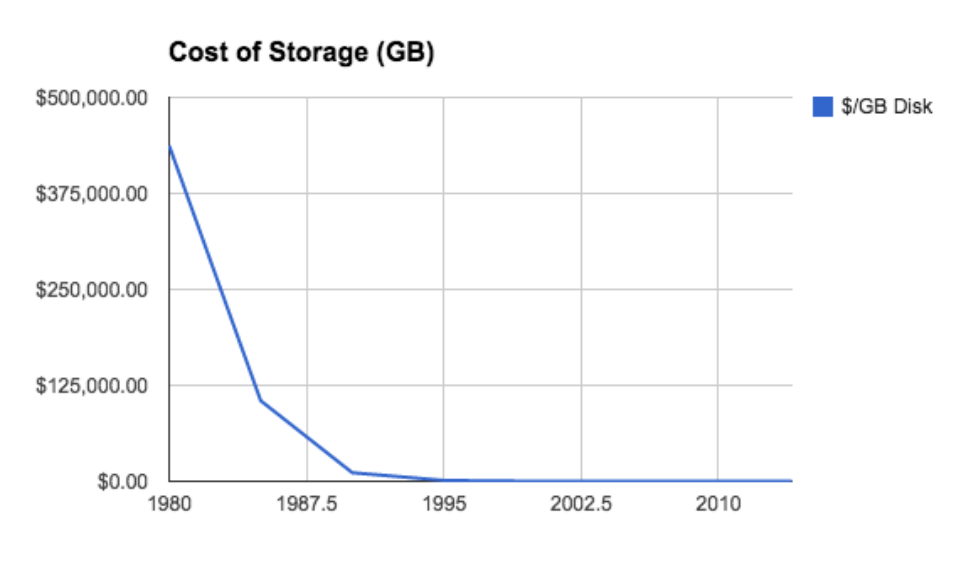


The Elephant Comes Into The Room

Why Move To Hadoop?

- Legacy systems were not performing well (< 1 TB / day)
- We had customers who wanted access to raw feeds (TB / day per customer)
- The advertising roadmap called for a 3-5x increase in traffic (new features, new customers onboarding)

Year	$/GB Disk
1980	$437,500.00
1985	$105,000.00
1990	$11,200.00
1995	$1,120.00
2000	$11.00
2005	$1.24
2010	$0.09
2013	$0.05
2014	$0.03

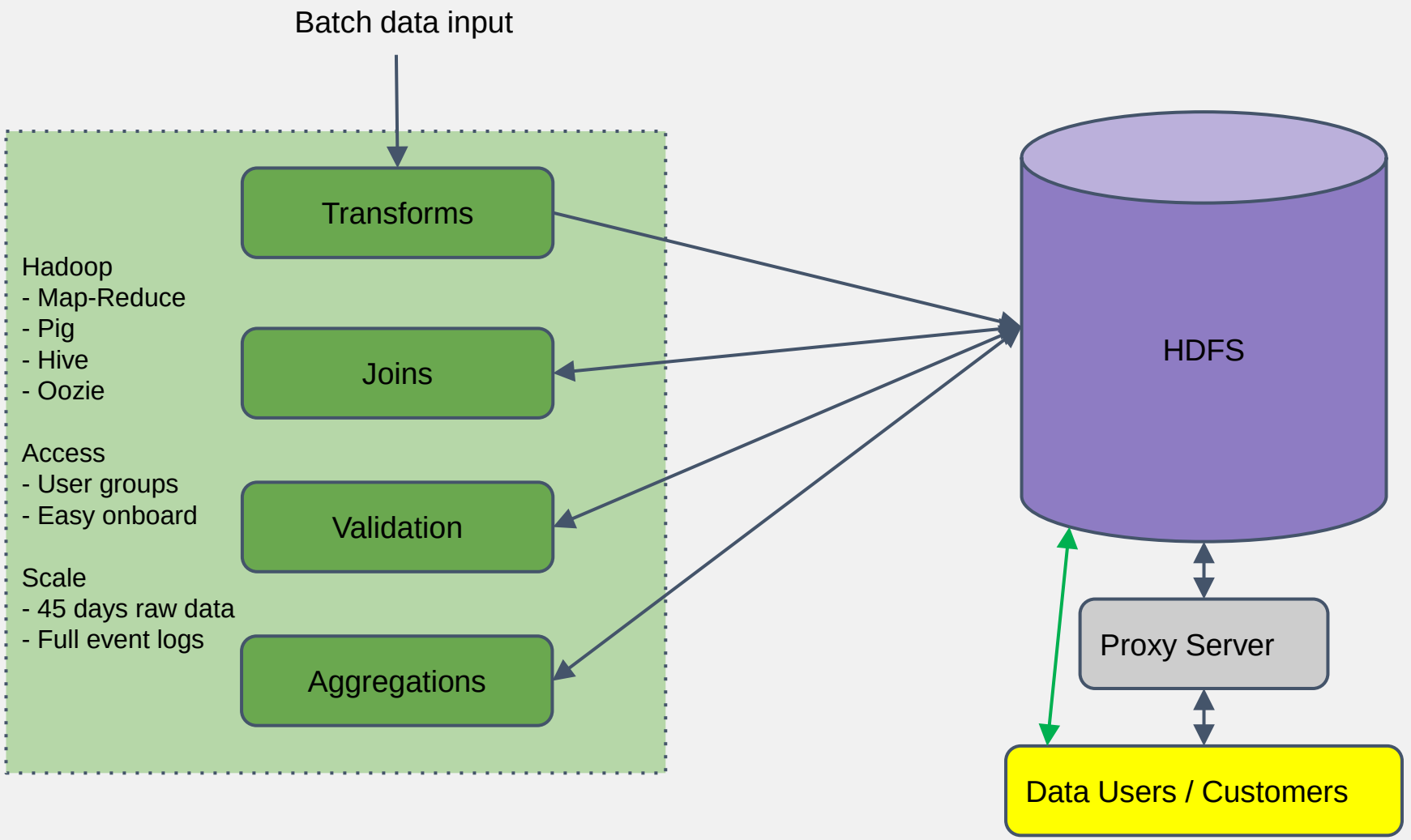


The Promise of Hadoop

- PB+ storage capabilities
  - Multi-tenant internal clusters made up of 1000s of nodes could handle TB/day easily
  - Storage was fault-tolerant with default 3x replication
  - Easy to scale up as new growth occurred

- Hosted service for job execution and data storage
  - No more need for separate clusters as Map/Reduce could handle all types of jobs
  - ETL operations easily handled using Pig Latin interface
  - New innovative frameworks were starting up (HBase, Hive, Oozie) promising more platform adoption

The Architecture on Hadoop



Life on Hadoop

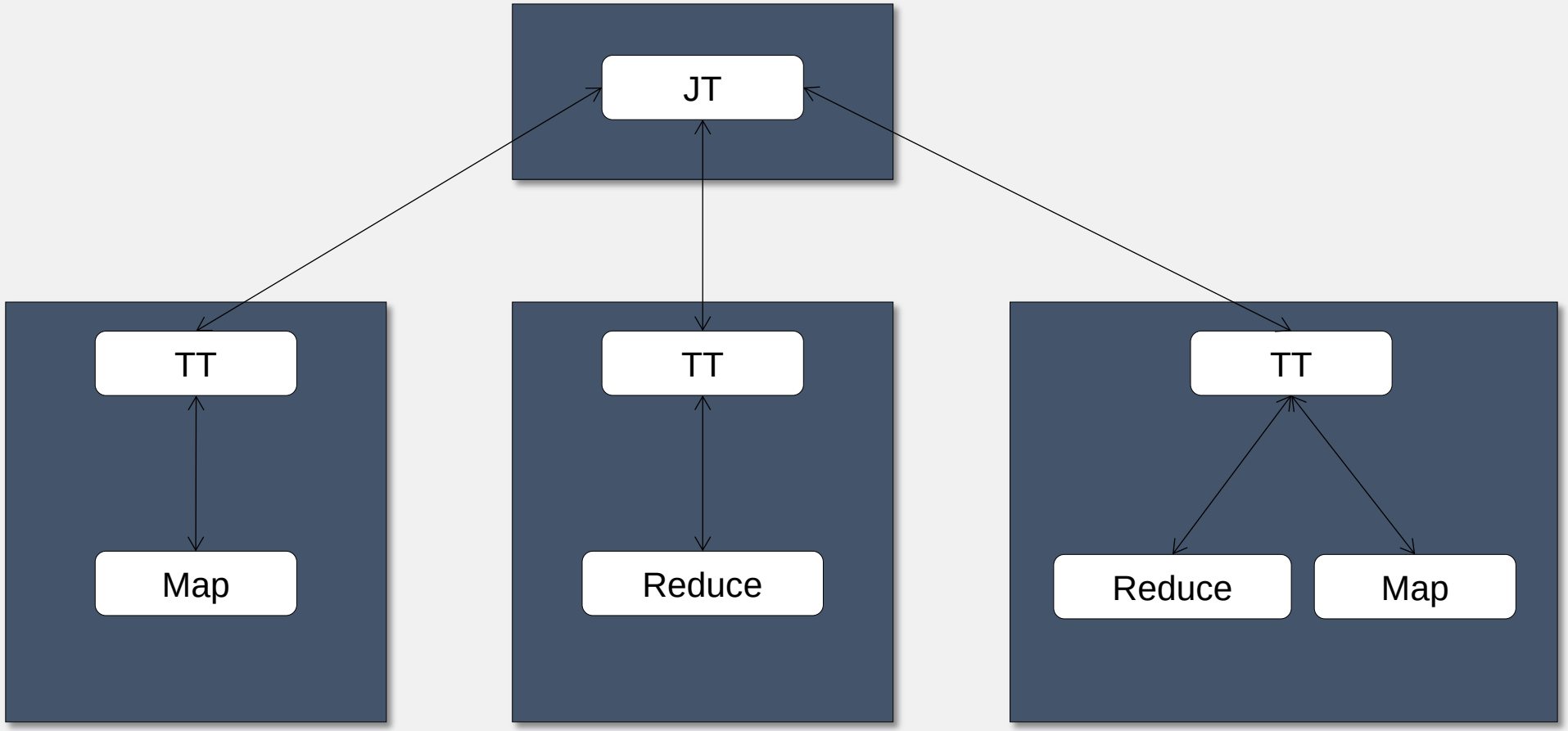
- Platform hardening had its consequences
  - Migrating data users and customers to the new system took longer than expected
  - Running large-scale data pipelines on multi-tenant clusters caused customer issues
- Data for everyone (who is permitted)
  - Number of data users increased dramatically on Hadoop
- Scaled better than expected (over past 5 years)
  - As data size has continued to grow, job runtime and data latency has continued to shrink

CLOUD COMPUTING APPLICATIONS

YARN Introduction

Roy Campbell & Reza Farivar

Hadoop 1.x



Issues with Hadoop

- Hadoop JobTracker was a barrier for scaling
  - Primary reason Hadoop 1.x is recommended for clusters no larger than 4000 nodes
  - Thousands of applications each running tens of thousands of tasks
  - JobTracker not able to schedule resources as fast as they became available
  - Distinct map and reduce slots led to artificial bottlenecks and low cluster utilization

Issues with Hadoop

- MapReduce was being abused by other application frameworks
  - Frameworks trying to work around sort and shuffle
  - Iterative algorithms were suboptimal
- YARN strives to be application framework agnostic
- Different application types can share the same cluster
- Runs MapReduce “out of the box” as part of Apache Hadoop

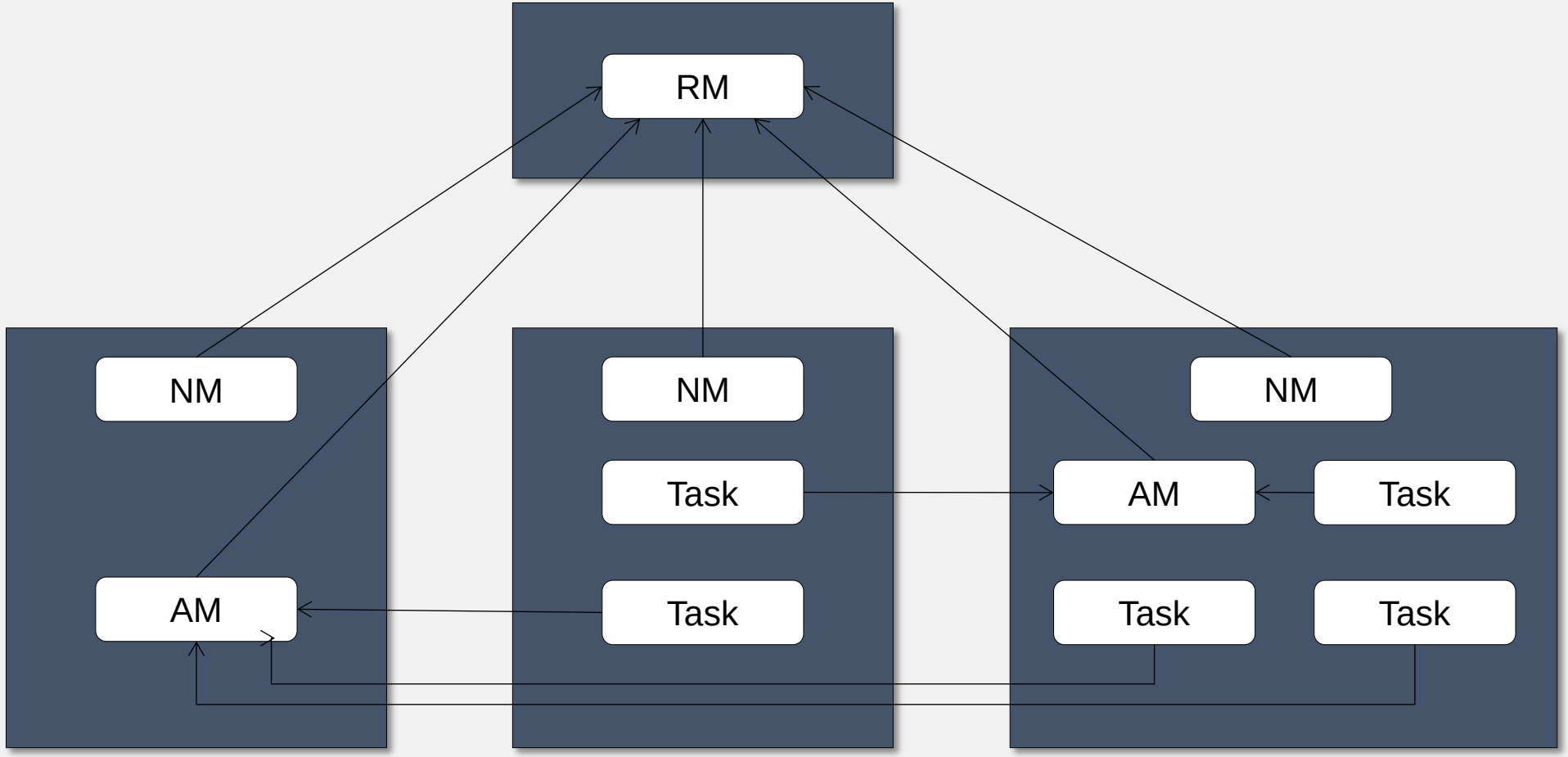
What is YARN?

- Yet Another Resource Negotiator
- Provides resource management services
  - Scheduling
  - Monitoring
  - Control
- Replaces the resource management services of the JobTracker
- Bundled with Hadoop 0.23 and Hadoop 2.x

YARN High-Level Architecture

- ResourceManager
  - Single, centralized daemon for scheduling containers
  - Monitors nodes and applications
- NodeManager
  - Daemon running on each worker node in the cluster
  - Launches, monitors, and controls containers
- ApplicationMaster
  - Provides scheduling, monitor, control for an application instance
  - RM launches an AM for each application submitted to the cluster
  - AM requests containers via RM; launches containers via NM
- Containers
  - Unit of allocation and control for YARN
  - ApplicationMaster and application-specific tasks run within containers

YARN High-Level Architecture

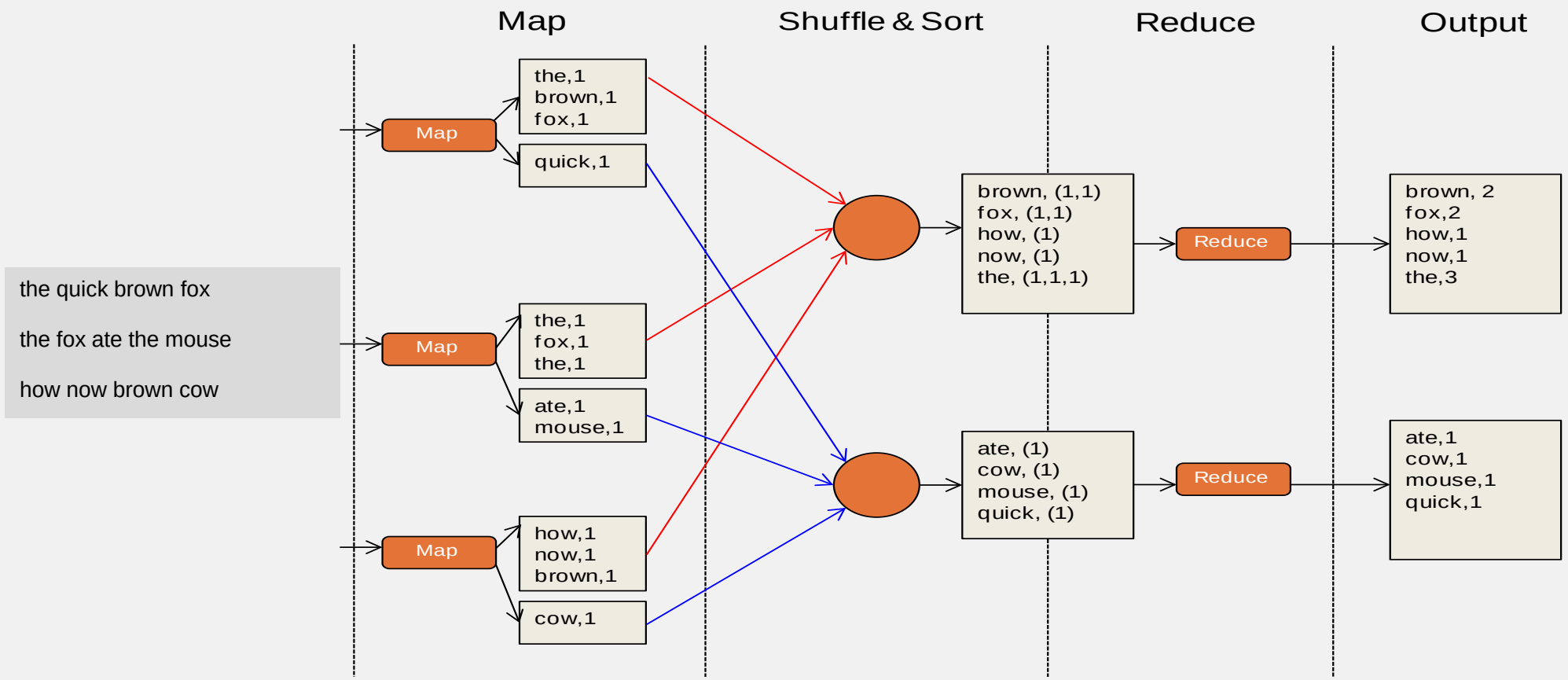


CLOUD COMPUTING APPLICATIONS

YARN: MapReduce on YARN

Roy Campbell & Reza Farivar

MapReduce


Map
Shuffle & Sort
Reduce
Output
the quick brown fox
the fox ate the mouse
how now brown cow
Map
Map
Map
the,1
brown,1
fox,1
quick,1
the,1
fox,1
the,1
ate,1
mouse,1
how,1
now,1
brown,1
cow,1
brown, (1,1)
fox, (1,1)
how, (1)
now, (1)
the, (1,1,1)
ate, (1)
cow, (1)
mouse, (1)
quick, (1)
Reduce
Reduce
brown, 2
fox,2
how,1
now,1
the,3
ate,1
cow,1
mouse,1
quick,1

MapReduce on YARN

- MapReduce AM determines number of map and reduce tasks
  - Split metainfo file indicates number of map tasks based on number of splits
  - Job config determines number of reducers
- AM schedules when to request containers for map and reduce tasks
  - Split metainfo file has data locality for each map task
  - Reducers have no locality
  - Uses headroom provided by RM to avoid livelocks where reducers consume all available resources but more maps need to run

MapReduce on YARN

- Tasks connect back to AM upon startup via TaskUmbilicalProtocol
  - Report progress, liveliness
  - AM kills tasks that do not report progress in a timely manner
  - AM provides reducers with shuffle data locations
  - Reducers notify AM of shuffle fetch failures; AM relaunches map tasks if necessary

MapReduce on YARN

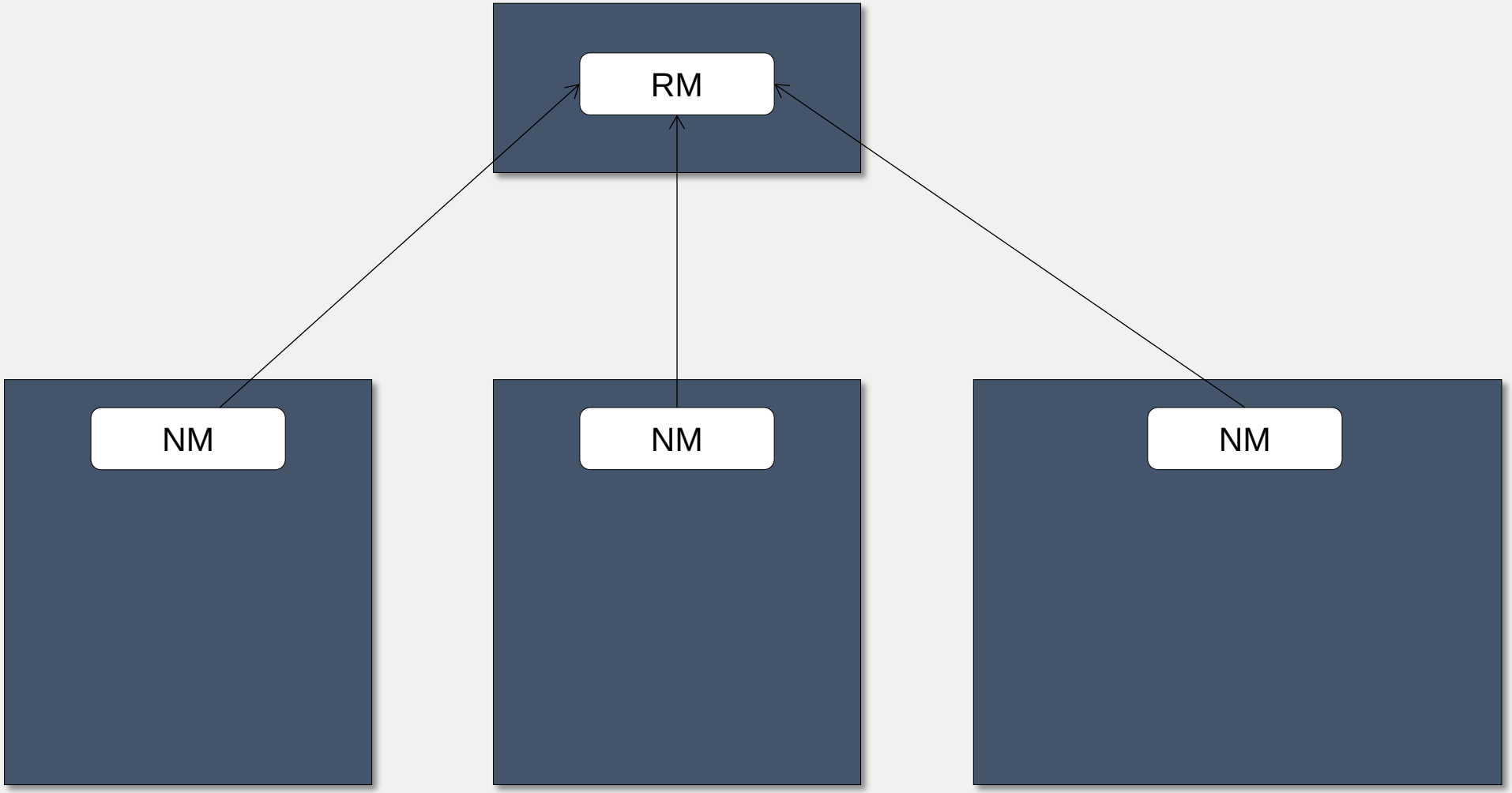
- Shuffle provided as a plugin service to NodeManagers
  - Shuffle port configurable, passed to reducers via AM
- AM responsible for job history
  - Job history events written to a file as job progresses
  - Copied to a drop location in HDFS when job completes
  - Used to provide recovery when AM crashes and is retried by RM
- MapReduce AM provides client interface
  - Report job and tasks status
  - Kill job or task attempts
  - Web app and services
  - Client can redirect to job history server if application has completed

CLOUD COMPUTING APPLICATIONS

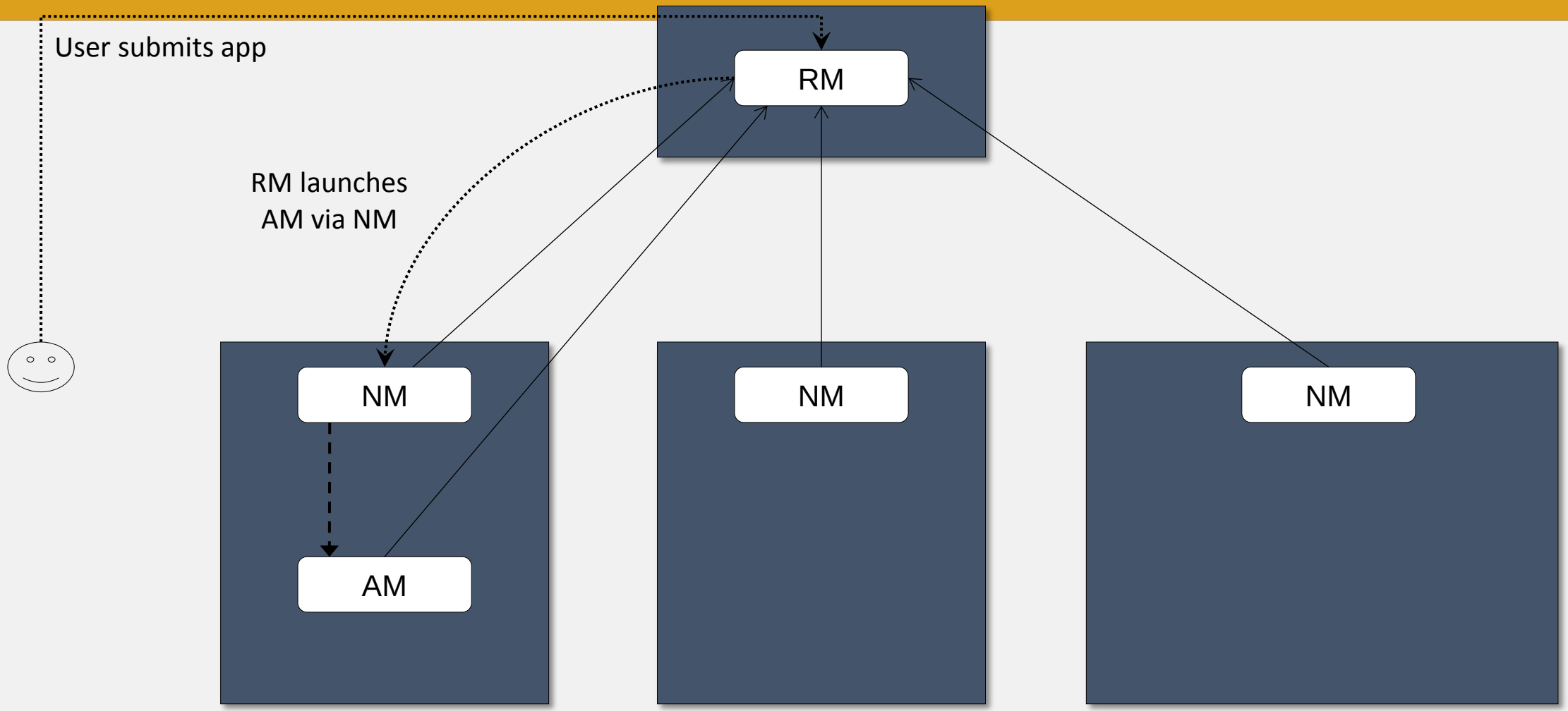
YARN: MapReduce on YARN in Diagram

Roy Campbell & Reza Farivar

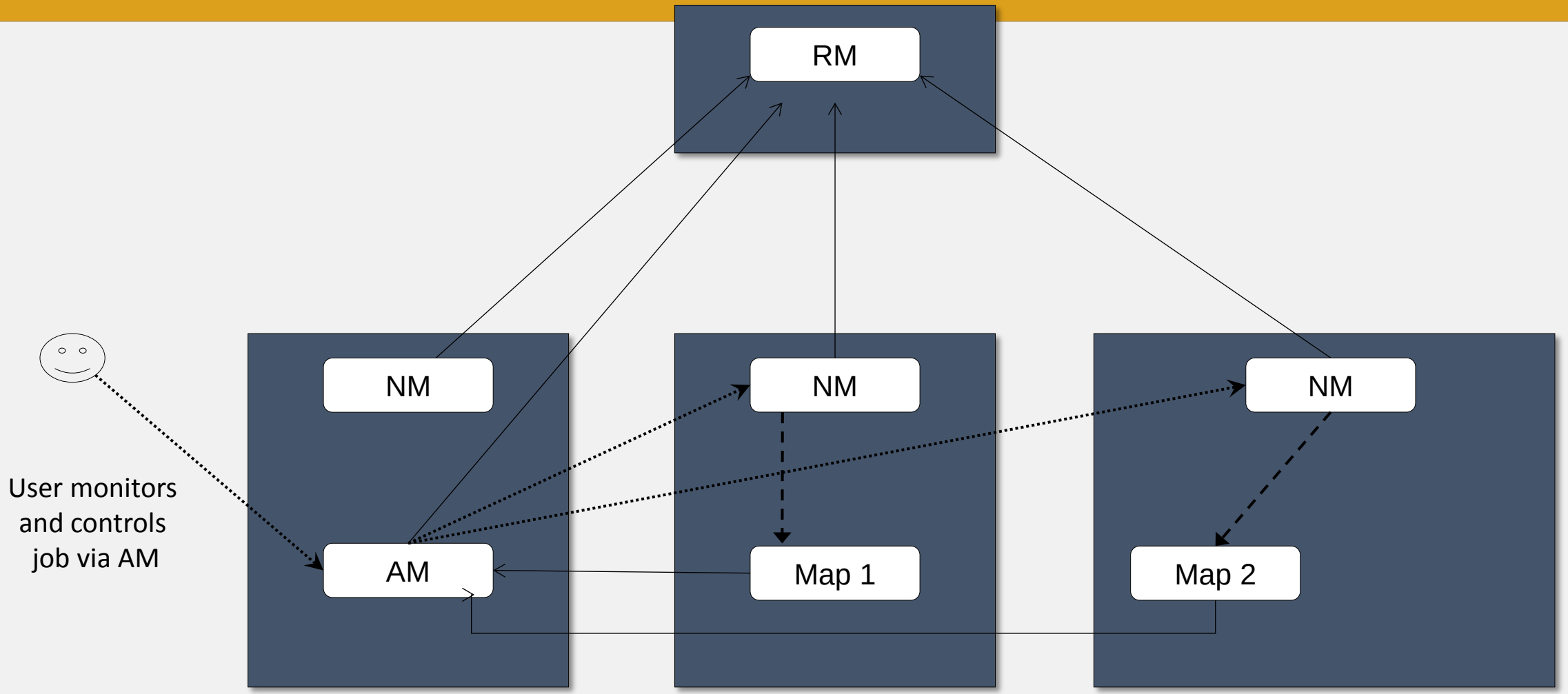
MapReduce on YARN



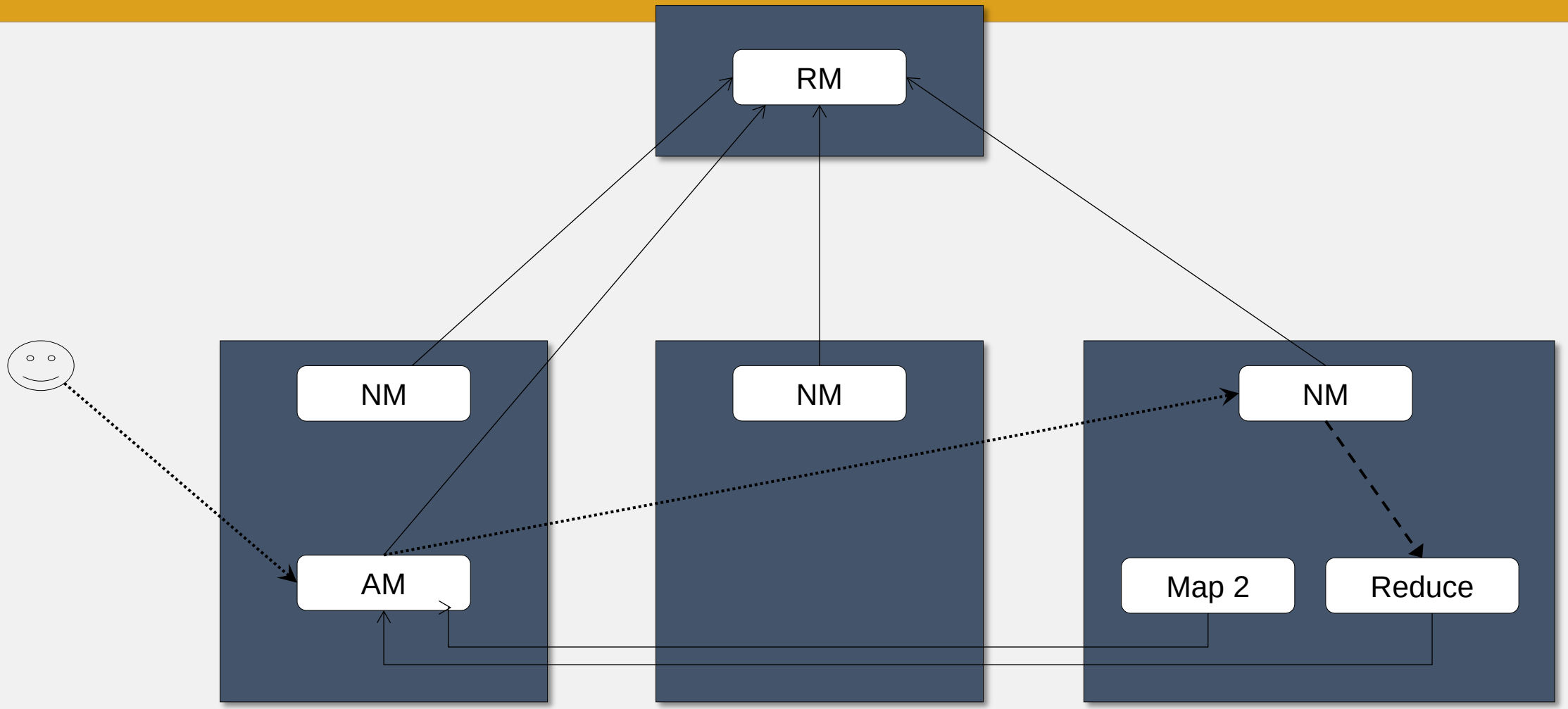
MapReduce on YARN



MapReduce on YARN

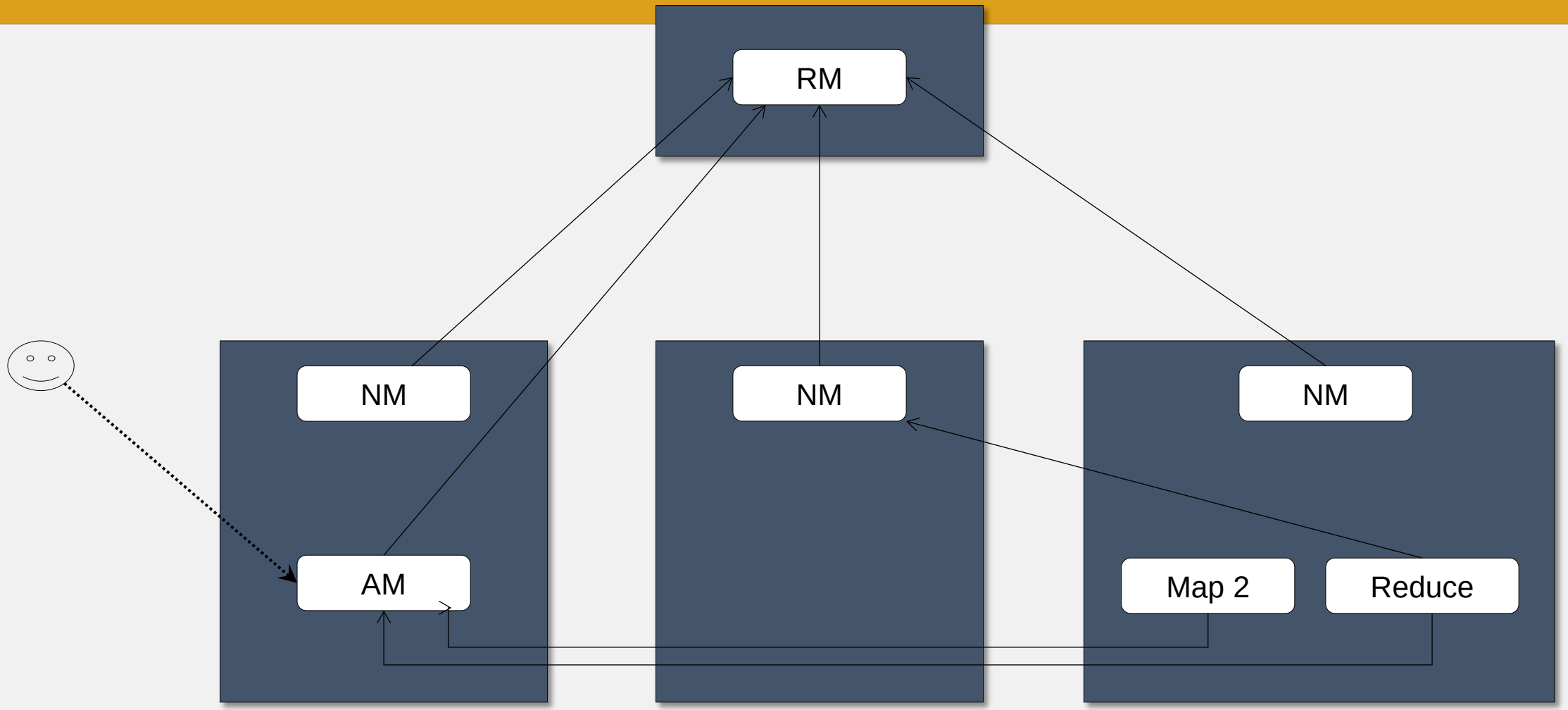


MapReduce on YARN



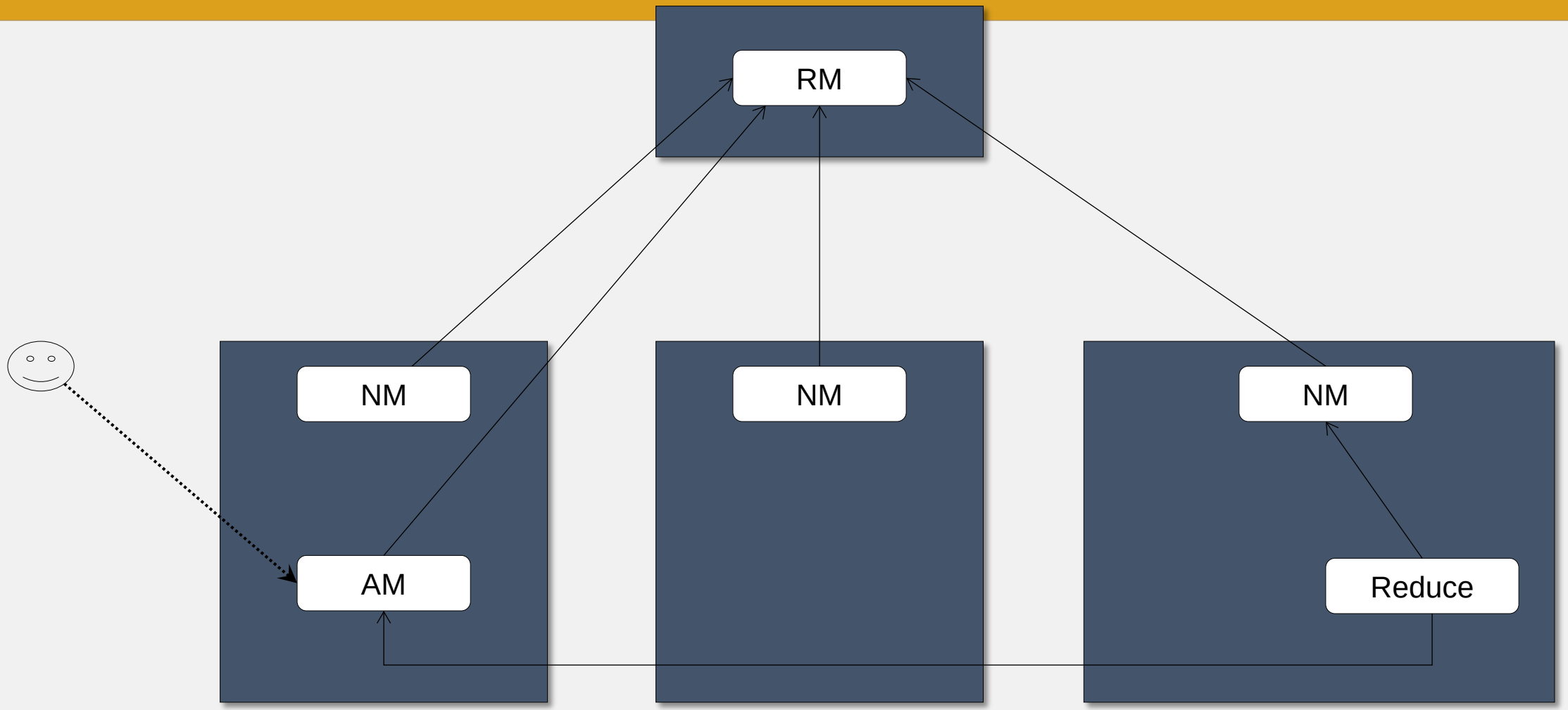
Map task 1 completes; AM requests container for reduce task and launches via NM

MapReduce on YARN



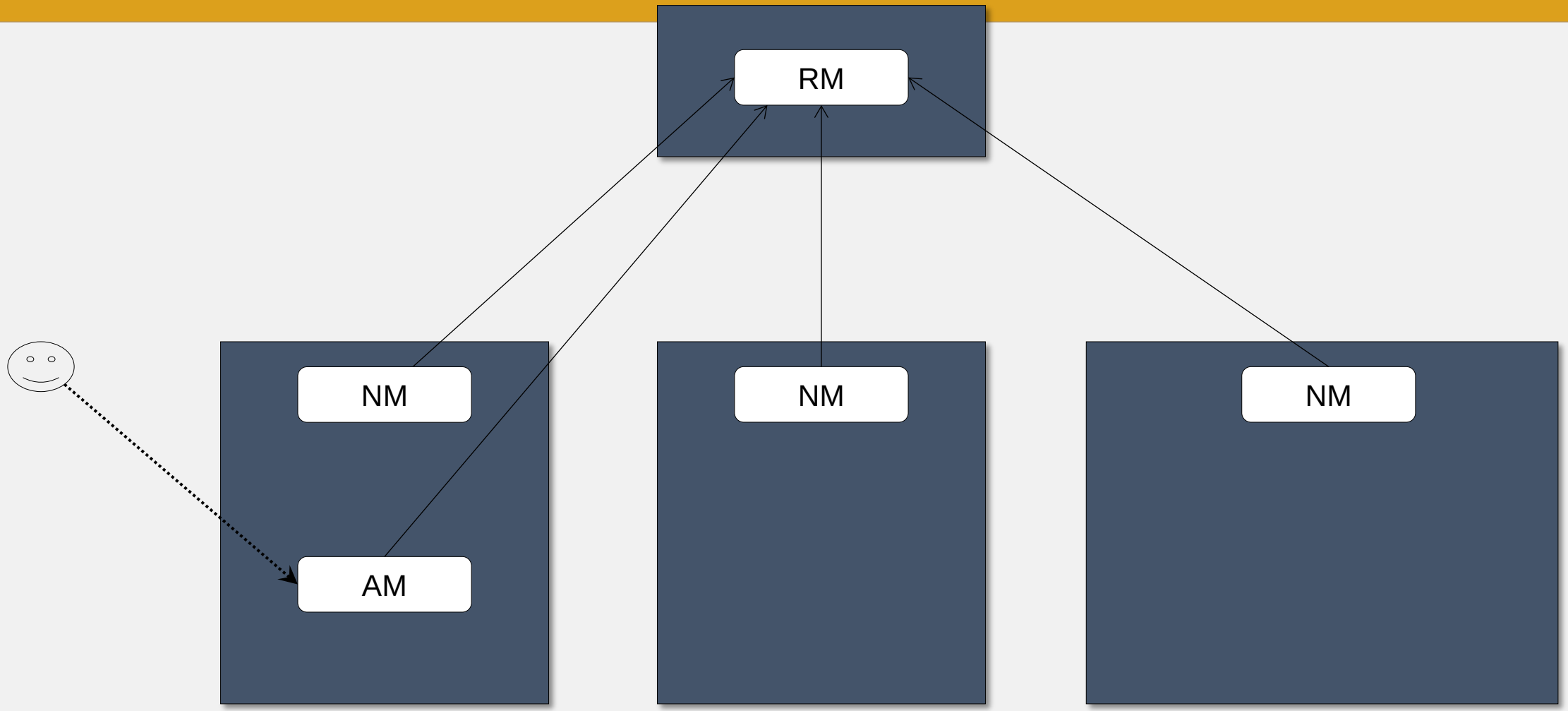
AM informs reducer of shuffle data location for map 1;
reduce task connects to NM and retrieves

MapReduce on YARN



Map 2 completes. AM informs reducer of shuffle data location for map 2 and reducer retrieves

MapReduce on YARN



Reducer completes. AM commits final output, copies job history, cleans staging area, and unregisters from RM before exiting

CLOUD COMPUTING APPLICATIONS

YARN: Node Manager

Roy Campbell & Reza Farivar

Node Manager Responsibilities

- Launch containers
- Localize files for containers
- Report container status to RM
- Kill containers and clean up local storage
- Monitor container resource usage and kill misbehaving containers
- Run health check and notify RM if unhealthy
- Log aggregation
- Web app for reporting node status and applications/containers
- Web services to avoid screen-scraping
- Runs auxiliary services for application frameworks as plugins
  - MapReduce ShuffleHandler serves map outputs to reducers

Node Manager Container Localization

- Download files/directories to local filesystem for the container
  - .jar file or executable for container
  - Supporting files, libraries
  - Known as the Distributed Cache in MapReduce
- Container launch context lists entries needing to be localized
- Localization requests have three privilege modes
  - Public: all users can access
  - Private: only the user can access
  - Application: only the application can access
- Downloaded entries are symlinked into the container working directory

Node Manager Log Aggregation

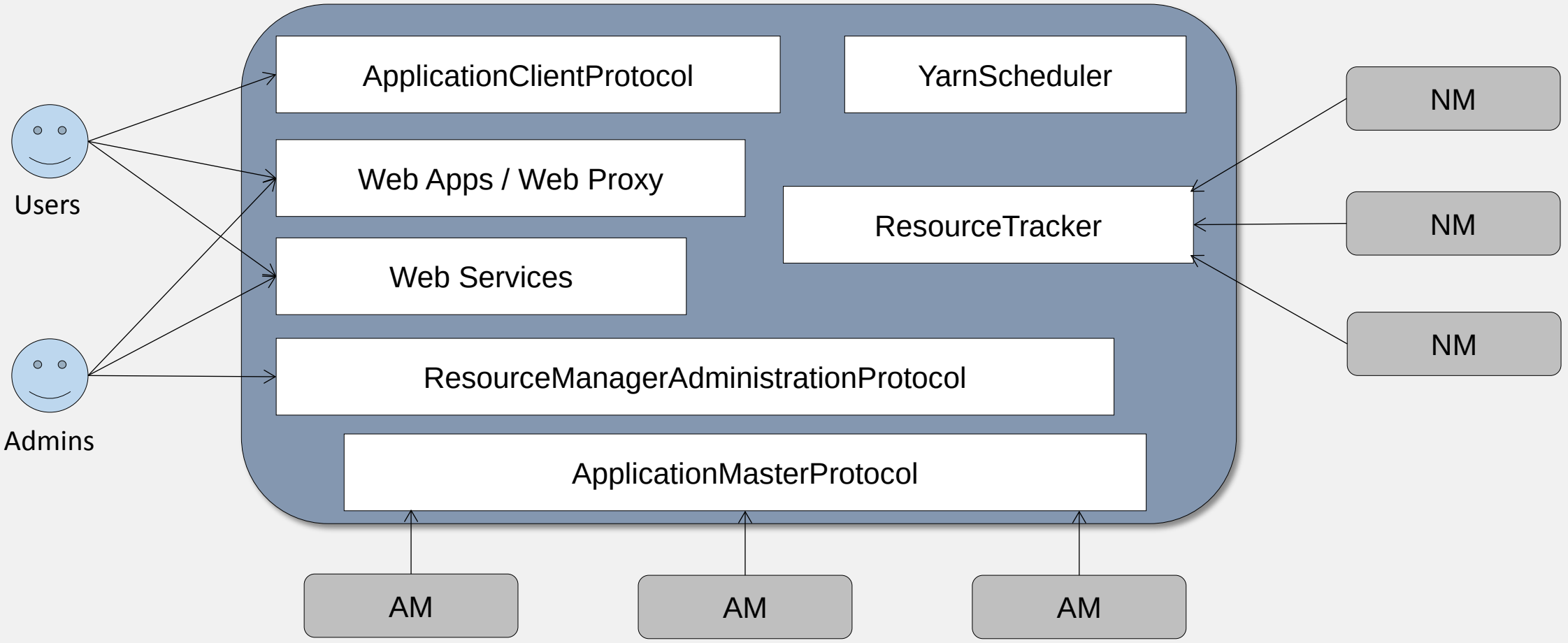
- NM collects logs from all containers for an application
- Collated into a file per node per application and uploaded to HDFS
- Benefits:
  - Allows long-term application log retention
  - Increases log availability
  - Easier to post-process logs via MapReduce jobs

CLOUD COMPUTING APPLICATIONS

YARN: Resource Manager

Roy Campbell & Reza Farivar

ResourceManager Interfaces



Resource Manager Interfaces

- ApplicationClientProtocol
  - Submit an application
  - Kill an application
  - Query application, scheduler, cluster status
- Web Apps
  - UI for users to track status of cluster, scheduler, and applications

User-Facing Interfaces

- Web Proxy
  - Users provided with a proxy URL when application submitted
  - AM can update tracking URL to an application-specific UI
  - Proxy redirects users to tracking URL
- Web Services
  - Avoid programmatic screen-scraping of web apps

User-Facing Interfaces

- ResourceManagerAdministrationProtocol
  - Control access of users
  - Decommission or restrict nodes
  - Reconfigure scheduler

Node Manager-Facing Interface

- ResourceTracker
  - Node registration and liveliness monitoring
  - Process container status updates and notify scheduler

Application Master-Facing Interface

- ApplicationMasterProtocol
  - AM registration and liveliness monitoring
  - Notify scheduler of container request updates from AMs
  - Notify AMs of granted containers and completed containers
  - Notify AMs of capacity remaining in their queue
  - Tracking URL management

Scheduling Interfaces

- YarnScheduler
  - Pluggable interface
  - FIFOScheduler, CapacityScheduler, and FairScheduler bundled with YARN
  - Active area of research in YARN

CLOUD COMPUTING APPLICATIONS

YARN: Application Master

Roy Campbell & Reza Farivar

Application Master Responsibilities

- Task scheduling and monitoring for an application instance
  - Only AM knows if a task has hung as RM and NM only see an opaque container

Application Master Responsibilities

- Heartbeats to RM
  - Indicates liveliness
  - Request new containers
  - Receive allocated containers

Application Master Responsibilities

- Provides optional tracking URL when registering with RM
  - Proxy URL provided by RM on app submission is an easy way to find app instance
  - URL can be updated when unregistering to point to a history server

CLOUD COMPUTING APPLICATIONS

Motivation for Spark

Roy Campbell & Reza Farivar

Motivation

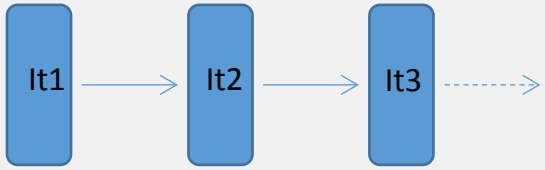
- Iterative algorithms and interactive data exploration are commonly used in many domains
- Traditional MapReduce and classical parallel runtimes cannot solve iterative algorithms efficiently
  - Hadoop: Repeated data access to HDFS, no optimization to data caching and data transfers
  - MPI: no natural support of fault tolerance and programming interface is complicated

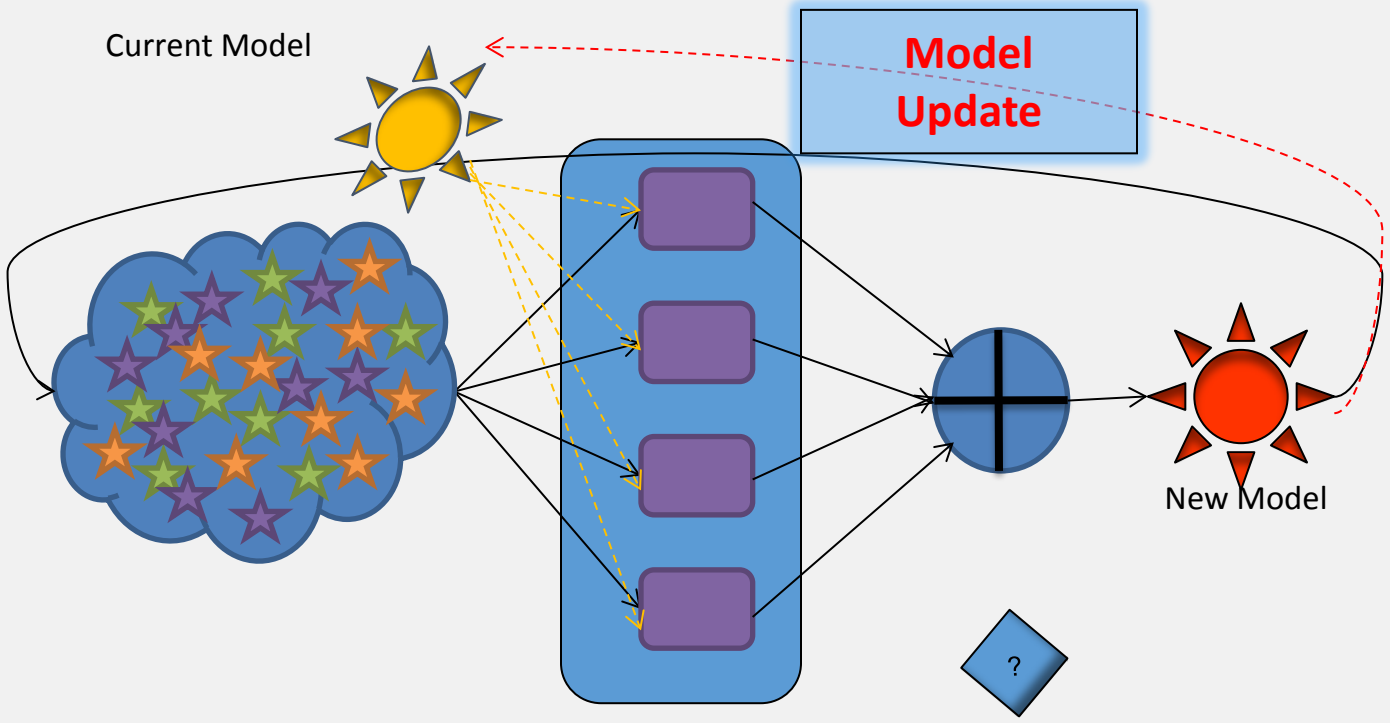
Retrofitting Iterations on MR

- MR does not support iteration out of the box
- But we still want Page Rank, clustering etc.
  - Mahout
  - Nutch
- The “obvious” solution: Split iteration into multiple MapReduce jobs. Write a driver program for orchestration.

A MapReduce Implementation of I.C.

```
Iterate {
   Map: for (each) i = 1 to M
      Compute();
   Reduce();
} Until converged();
```





- Repeated reads of constant (input) data in each iteration
- Run-time overheads in each iteration
- Intermediate Communication resulting from model updates
- Model Update Traffic
- Granularity of parallelism limited by iteration

CLOUD COMPUTING APPLICATIONS

Apache Spark

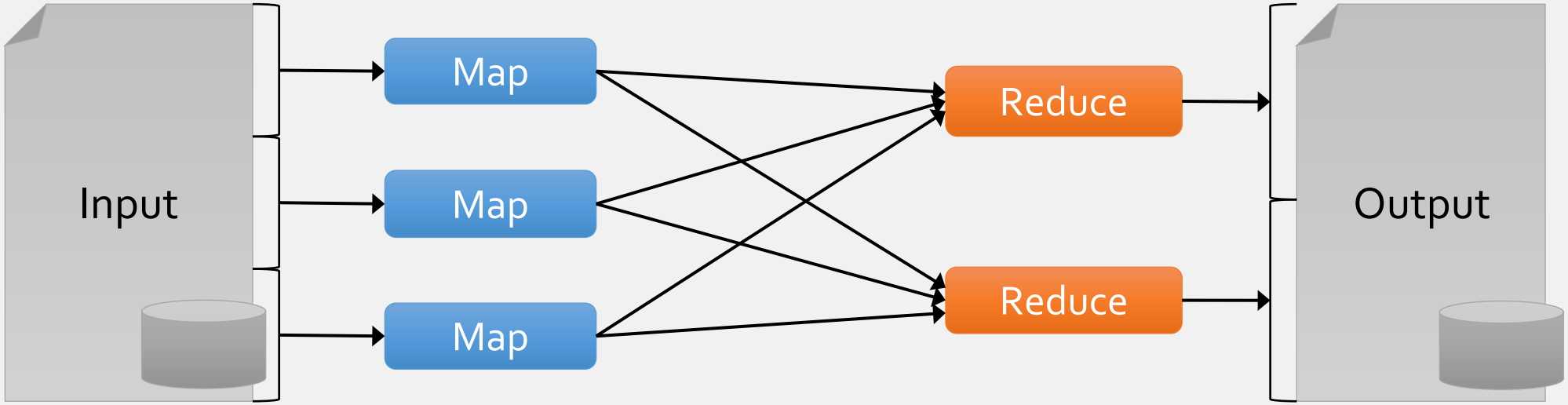
Roy Campbell & Reza Farivar

Apache Spark

- Extend the MapReduce model to better support two common classes of analytics apps:
  - **Iterative** algorithms (machine learning, graphs)
  - **Interactive** data mining
- Enhance programmability:
  - Integrate into Scala programming language
  - Allow interactive use from Scala interpreter

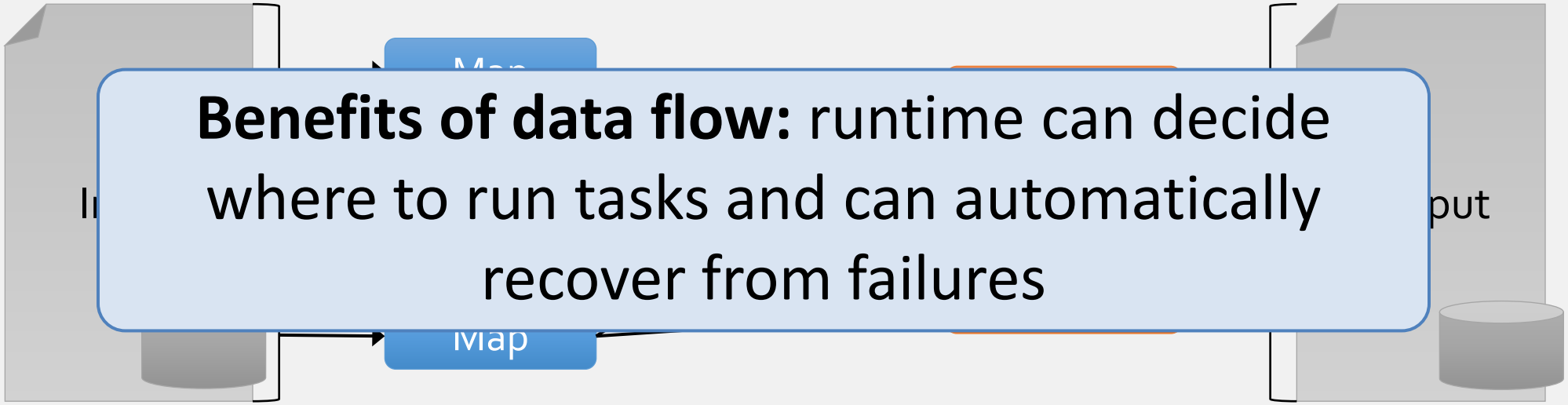
Motivation

Most current cluster programming models are based on *acyclic data flow* from stable storage to stable storage



Motivation

Most current cluster programming models are based on *acyclic data flow* from stable storage to stable storage



Benefits of data flow: runtime can decide where to run tasks and can automatically recover from failures

Motivation

- Acyclic data flow is inefficient for applications that repeatedly reuse a *working set* of data:
  - **Iterative** algorithms (machine learning, graphs)
  - **Interactive** data mining tools (R, Excel, Python)
- With current frameworks, apps reload data from stable storage on each query

Solution: Resilient Distributed Datasets (RDDs)

- Allow apps to keep working sets in memory for efficient reuse
- Retain the attractive properties of MapReduce
  - Fault tolerance, data locality, scalability
- Support a wide range of applications

Programming Model

- Resilient distributed datasets (RDDs)
  - Immutable, partitioned collections of objects
  - Created through parallel *transformations* (map, filter, groupBy, join, …) on data in stable storage
  - Can be *cached* for efficient reuse
- Actions on RDDs
  - Count, reduce, collect, save, …

CLOUD COMPUTING APPLICATIONS

Spark Example: Log Mining

Roy Campbell & Reza Farivar

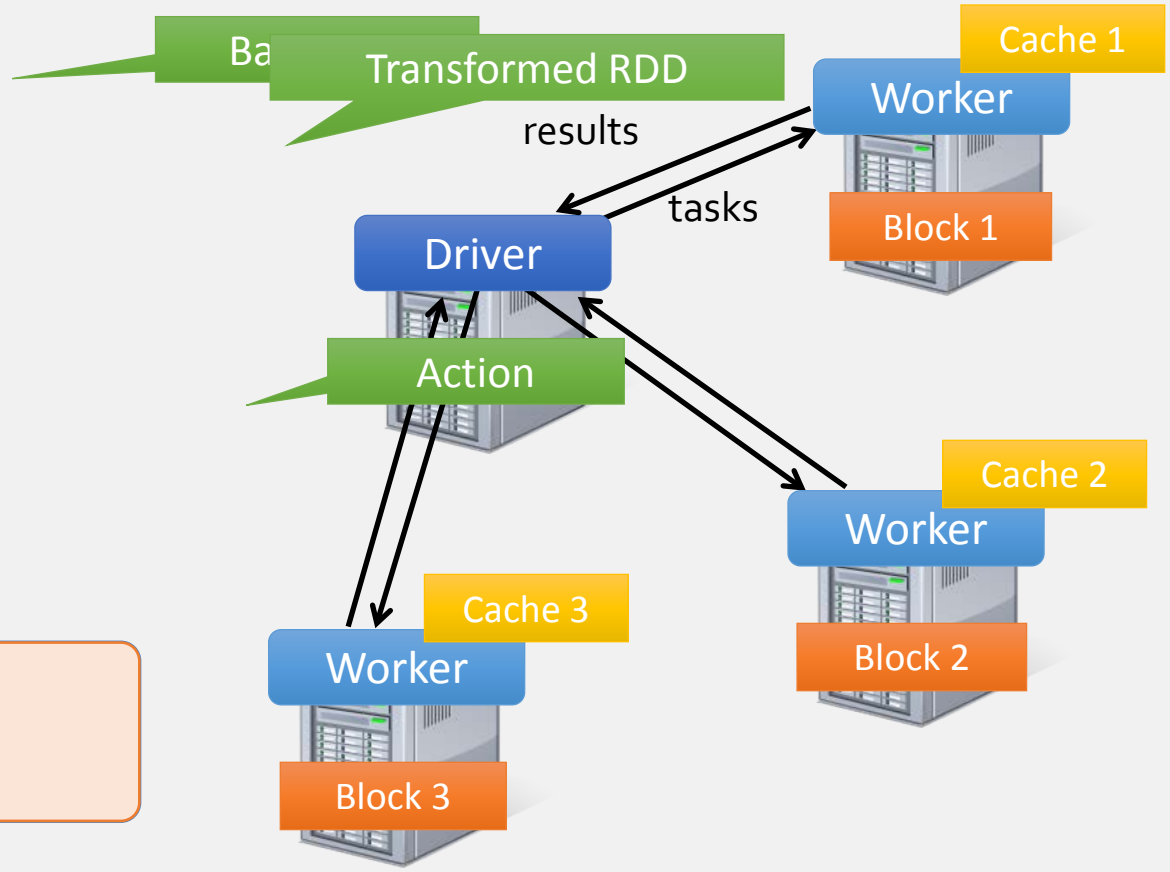
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile(“hdfs://...”)
errors = lines.filter(_.startsWith(“ERROR”))
messages = errors.map(_.split(‘\t’)(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains(“foo”)).count
cachedMsgs.filter(_.contains(“bar”)).count
. . .
```

Result: scaled to 1 TB data in 5-7 sec
(vs 170 sec for on-disk data)



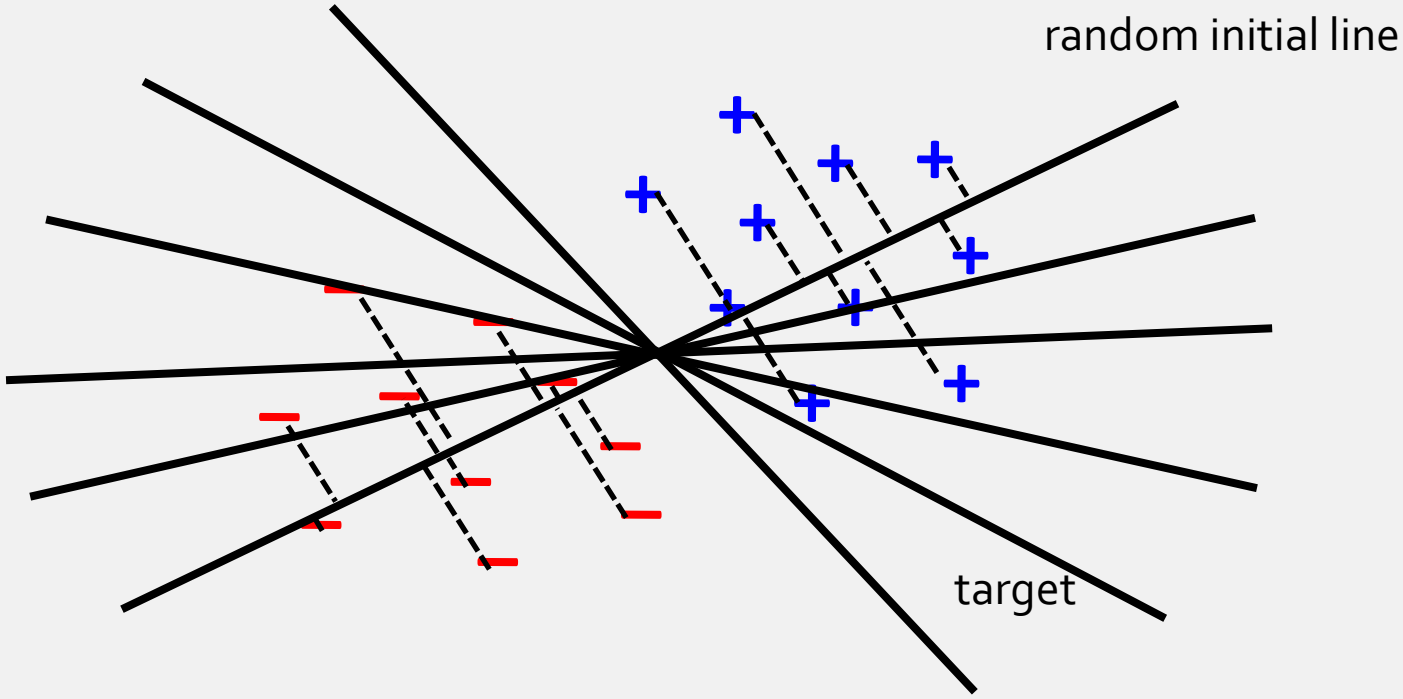
CLOUD COMPUTING APPLICATIONS

Spark Example: Logistic Regression

Roy Campbell & Reza Farivar

Example: Logistic Regression

Goal: find best line separating two sets of points



Example: Logistic Regression

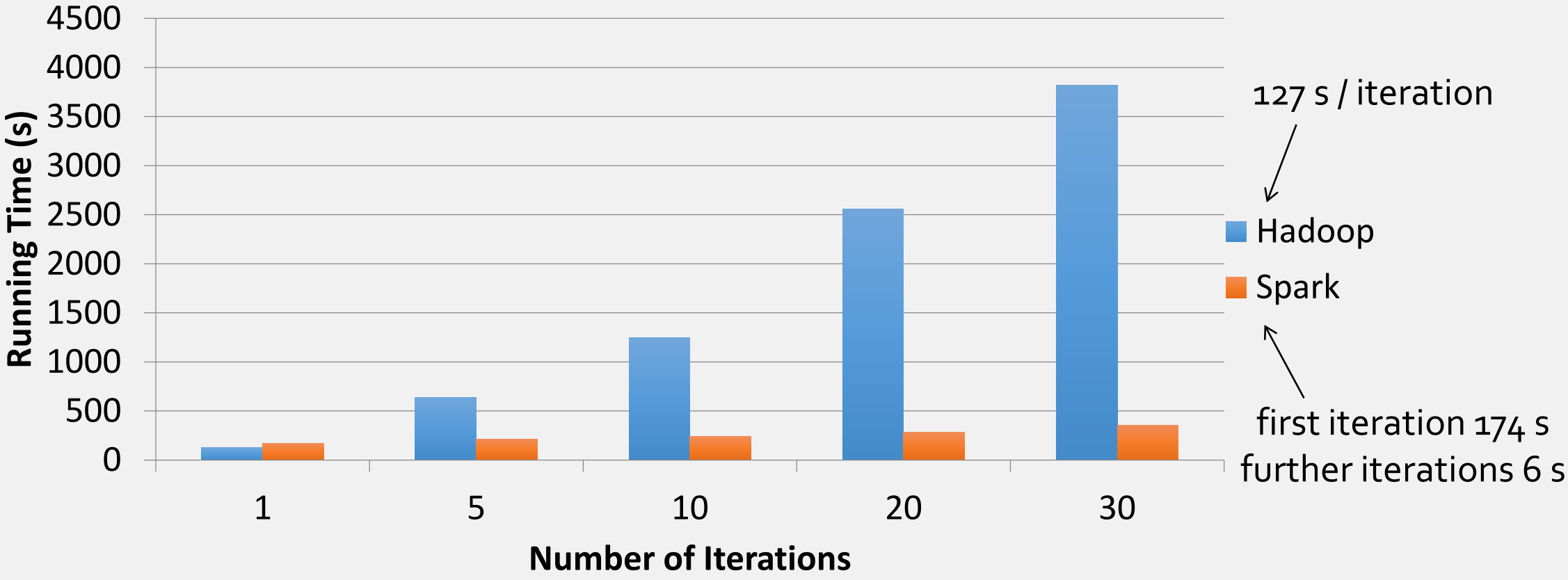
```
val data = spark.textFile(...).map(readPoint).cache()

var w = Vector.random(D)

for (i <- 1 to ITERATIONS) {
  val gradient = data.map(p =>
    (1 / (1 + exp(-p.y*(w dot p.x))) - 1) * p.y * p.x
  ).reduce(_ + _)
  w -= gradient
}

println("Final w: " + w)
```

Logistic Regression Performance



CLOUD COMPUTING APPLICATIONS

RDD Fault Tolerance

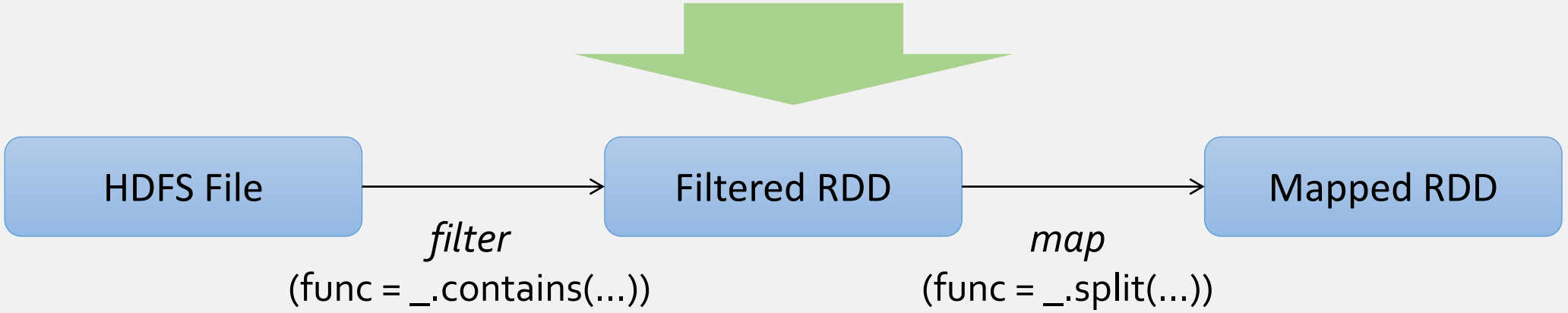
Roy Campbell & Reza Farivar

RDD Fault Tolerance

RDDs maintain *lineage* information that can be used to reconstruct lost partitions

Ex:

```
messages = textFile(...).filter(_.startsWith("ERROR"))
                        .map(_.split('\t')(2))
```



CLOUD COMPUTING APPLICATIONS

Interactive Spark

Roy Campbell & Reza Farivar

Interactive Spark

- Modified Scala interpreter to allow Spark to be used interactively from the command line
- Required two changes:
  - Modified wrapper code generation so that each line typed has references to objects for its dependencies
  - Distribute generated classes over the network

Frameworks Built on Spark

- Pregel on Spark (GraphX)
  - Google message passing model for graph computation
- Hive on Spark (SparkSQL)
  - Compatible with Apache Hive
  - ML operators in Scala
- Mllib
  - Scalable Machine Learning Library

CLOUD COMPUTING APPLICATIONS

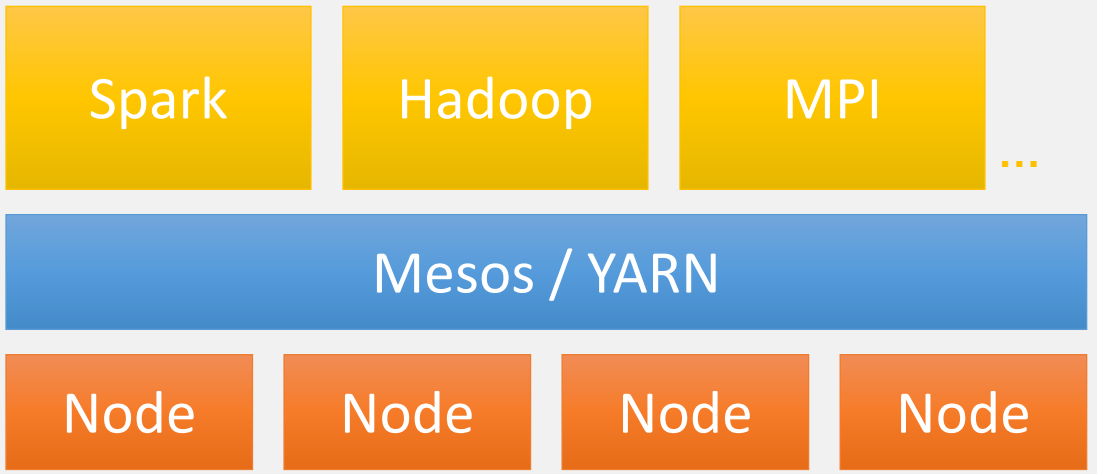
Spark Implementation

Roy Campbell & Reza Farivar

Implementation

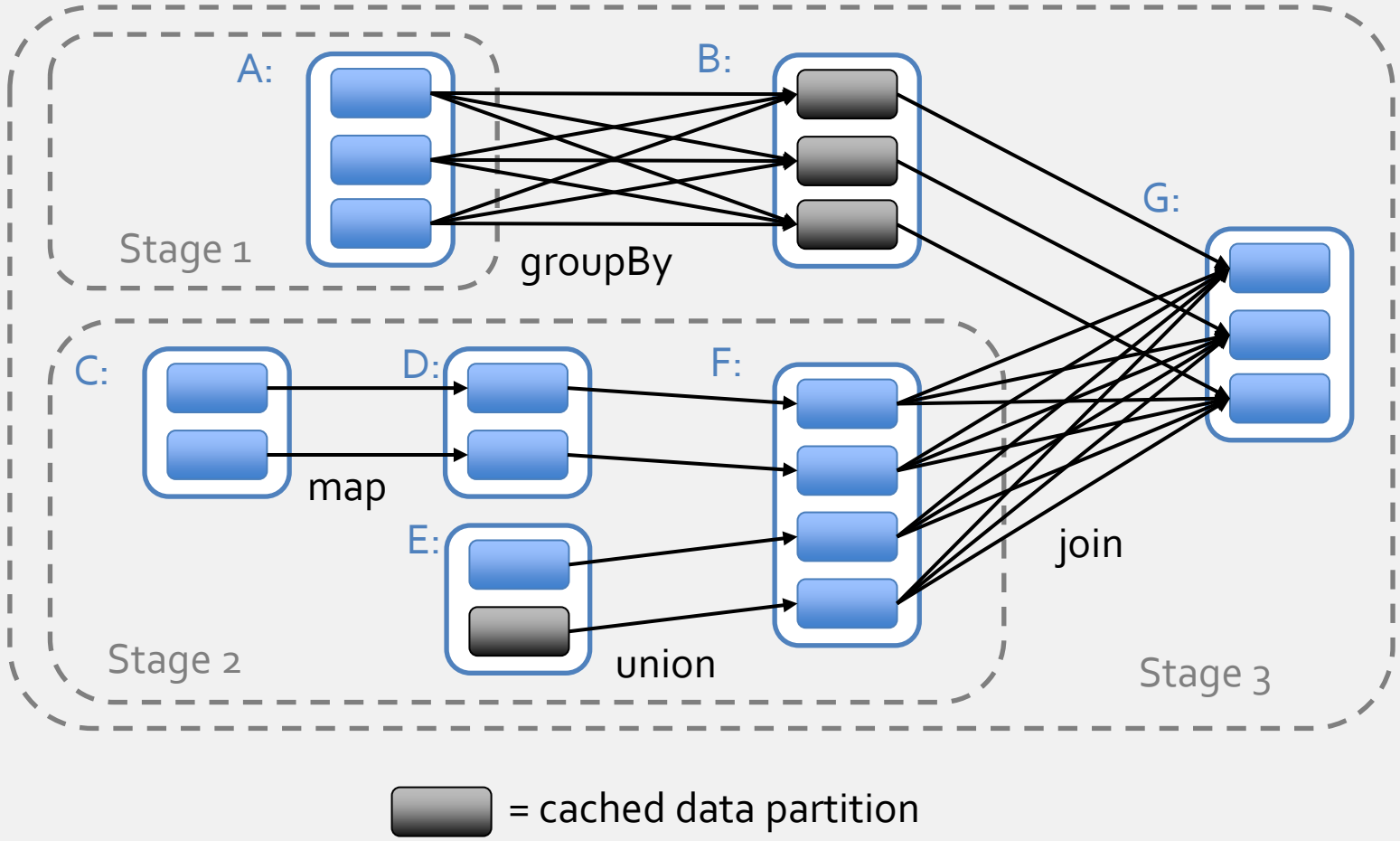
Runs on Apache Mesos or YARN to share resources with Hadoop & other apps

Can read from any Hadoop input source (e.g. HDFS)



Spark Scheduler

- Dryad-like DAGs
- Pipelines functions within a stage
- Cache-aware work reuse & locality
- Partitioning-aware to avoid shuffles



CLOUD COMPUTING APPLICATIONS

HDFS Introduction

Roy Campbell & Reza Farivar

Large-Scale Data Processing

- Many tasks
  - Processing lots of data in parallel to produce lots of other data
- Large-scale data processing
  - Want to use 1000s of CPUs
    - But don't want overhead of **managing** storage
    - Storage devices fail 1.7% year 1 – 8.6% year 3 (Google, 2007)
    - 10,000 nodes, 7 disks per node, year 1, 1190 failures/yr or 3.3 failures/day
- MapReduce provides:
  - User-defined functions
  - Automatic parallelization and distribution
  - Fault tolerance
  - I/O scheduling
  - Status and monitoring

HDFS

- Synergistic with Hadoop
  - Apache Project inspired by Google Map/Reduce and GFS
- Massive throughput
- Throughput scales with attached HDs
- Have seen VERY LARGE production clusters
  - Facebook, Yahoo… Nebraska
- Doesn’t even pretend to be POSIX compliant
- Optimized for reads, sequential writes and appends

References

- The Google File System
  - SOSP 2003
- The Hadoop Distributed File System
  - 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)

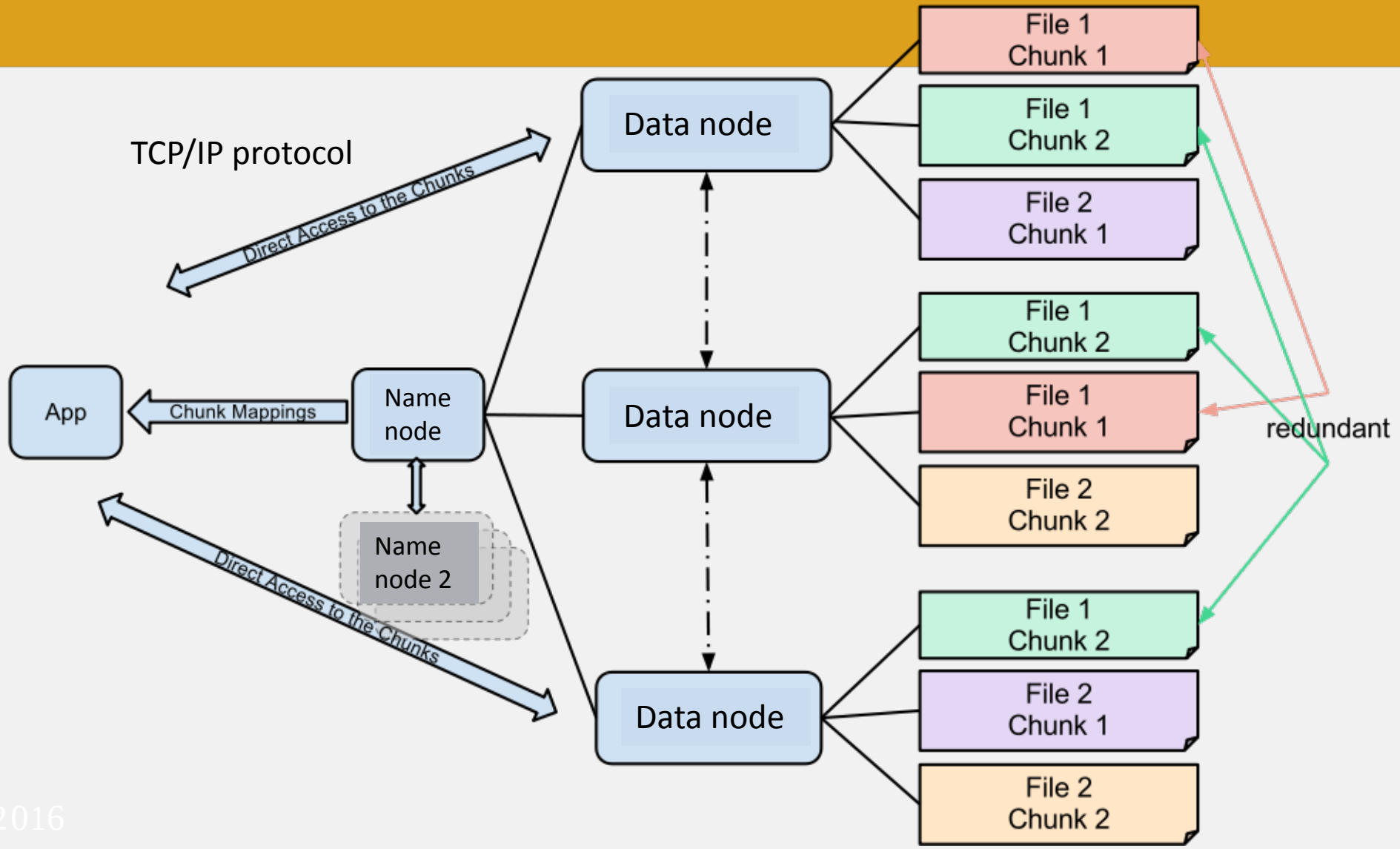
Stable Storage

- If nodes failure is the norm and not the exception, how can we store data persistently?
- **Answer:** Distributed File System replicates files
  - Provides global file namespace
  - Google GFS, Hadoop HDFS
  - Kosmix KFS
- Typical usage pattern
  - Huge files (100s of GB to TB)
  - Data is rarely updated in place
  - Multiple copies improves availability
  - Reads and appends are common

Distributed File System

- Datanode Servers
  - A file is split into contiguous chunks
  - Typically each chunk is 16-64MB
  - Each chunk replicated (usually 2x or 3x)
  - Sends heartbeat and BlockReport to namenode
- Replicas are placed: one on a node in a local rack, one on a different node in the local rack, and one on a node in a different rack

HDFS Architecture



Distributed File System

- Master node
  - a.k.a. NameNode in HDFS
  - Stores metadata
  - Might be replicated
- Client library for file access
  - Talks to master to find data node chunk
  - Connects directly to data node servers to access data

Replication Pipelining

- When the client receives response from NameNode, it flushes its block in small pieces (4K) to the first replica, that in turn copies it to the next replica and so on
- Thus data is pipelined from data node to the next

Staging

- A client request to create a file does not reach NameNode immediately
- HDFS client caches the data into a temporary file. When the data reaches an HDFS block size, the client contacts the NameNode
- NameNode inserts the filename into its hierarchy and allocates a data block for it
- The NameNode responds to the client with the identity of the data node and the destination of the replicas (data nodes) for the block
- Then the client flushes it from its local memory

Application Programming Interface

- HDFS provides Java API for application to use
- Python access is also used in many applications
- A C language wrapper for Java API is also available
- A command line interface provided in Hadoop
- The syntax of the commands is similar to bash
- Example: to create a directory /foodir
- /bin/hadoop dfs –mkdir /foodir
- An HTTP browser can be used to browse the files of an HDFS instance