Map-reduce Framework

BITS Pilani
Pilani Campus

K Hari Babu
Department of Computer Science & Information Systems

innovate achieve lead

BITS Pilani

Pilani | Dubai | Goa | Hyderabad

MapReduce Paradigm

MapReduce Paradigm

- MapReduce is a parallel programming model and an associated implementation introduced by Google. In the programming model, a user specifies the computation by two functions, Map and Reduce.
- MapReduce programming model derives from the map and reduce combinators from a functional language like Lisp.
  - In Lisp, a map takes as input a function and a sequence of values. It then applies the function to each value in the sequence.
  - A reduce combines all the elements of a sequence using a binary operation. For example, it can use "+" to add up all the elements in the sequence.
- MapReduce is inspired by these concepts

MapReduce Framework

- MapReduce framework provides the means to break a large task into smaller tasks, run the tasks in parallel, and consolidate the outputs of the individual tasks into the final output
- As its name implies, MapReduce consists of two basic parts
  - a map step and a reduce step
- Map:
  - Applies an operation to a piece of data
  - Provides some intermediate output
- Reduce:
  - Consolidates the intermediate outputs from the map steps
  - Provides the final output
- Each step uses key/value pairs, denoted as <key, value>, as input and output

MapReduce Framework

- Map, written by a user, takes an input pair and produces a set of intermediate key/value pairs
- The MapReduce library groups together all intermediate values associated with the same intermediate key I and passes them to the reduce function.
- The reduce function, also written by the user, accepts an intermediate key I and a set of values for that key. It merges together these values to form a possibly smaller set of values

WordCount Example

<1234, “For each word in each string”>

Map

<For, 1> <each, 1> <word, 1> <in, 1> <each, 1> <string, 1>

Reduce

<For, 1>
<each, 2>
<word, 1>
<in, 1>
<string, 1>

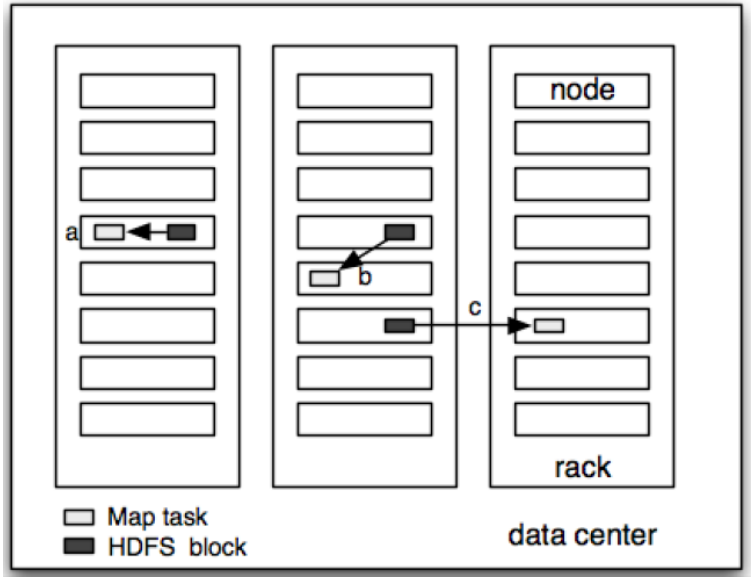
- For map()
  - input is <line no, line>
  - Output is <word, 1>
- For reduce()
  - Input is collection of key,value pairs.
  - Output is <word, final count>

MapReduce in Hadoop

- Hadoop divides the input to a MapReduce job into fixed-size pieces called input-splits, or just splits
- Hadoop creates one map task for each split, which runs the user-defined map function for each record in the split
- For most jobs, a good split size tends to be the size of an HDFS block, 64 MB by default.
  - although this can be changed for the cluster (for all newly created files), or specified when each file is created

Scheduling Map task- Data Locality

- Hadoop does its best to run the map task on a node where the input data resides in HDFS
  - This is called the data locality optimization since it doesn't use valuable cluster bandwidth
- Sometimes all three nodes hosting the HDFS block replicas are busy, job scheduler will look for a free map slot on a node in the same rack as one of the blocks
  - If this is not possible, so an off-rack node is used, which results in an inter-rack network transfer

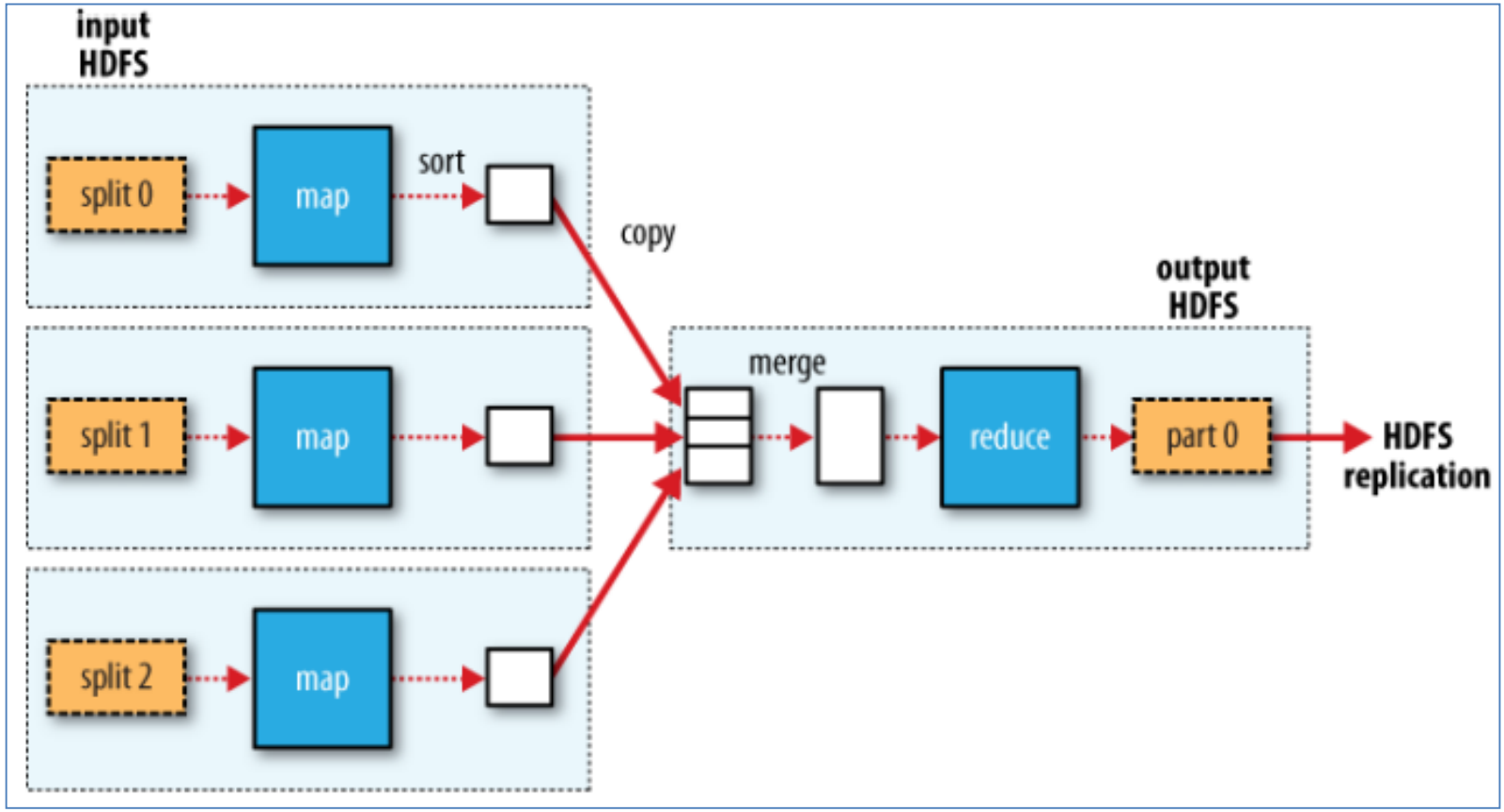


Scheduling Reduce Jobs

- Reduce tasks don't have the advantage of data locality
  - the input to a single reduce task is normally the output from all mappers
- In the wordcount example, there is a single reduce() job
  - Therefore, the sorted map outputs have to be transferred across the network to the node where the reduce task is running
  - Where they are merged and then passed to the user-defined reduce function

Scheduling Reduce Jobs

- In single reduce case, the sorted map outputs have to be transferred across the network to the node where the reduce task is running
- Where they are merged and then passed to the user-defined reduce function

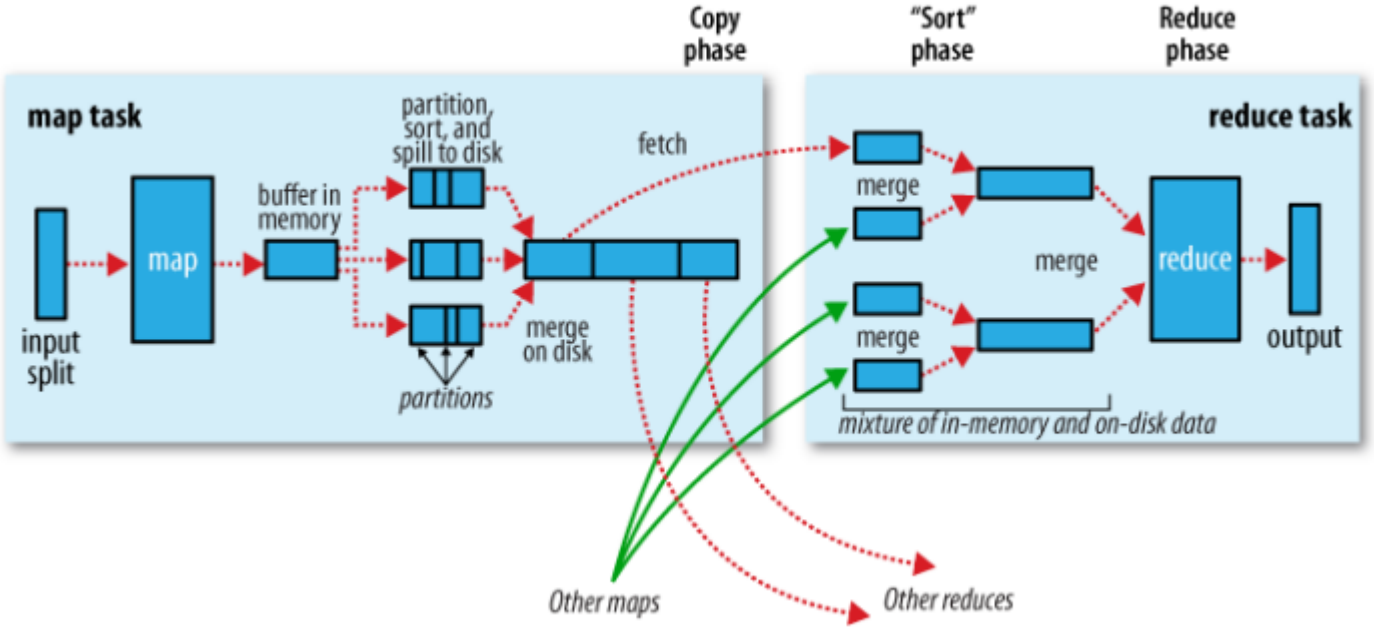


Multiple Reducers

- When there are multiple reducers, the map tasks partition their output, each creating one partition for each reduce task.
  - There can be many keys (and their associated values) in each partition, but the records for any given key are all in a single partition.
- The partitioning can be controlled by a user-defined partitioning function, but normally the default partitioner—which uses a hash function works very well

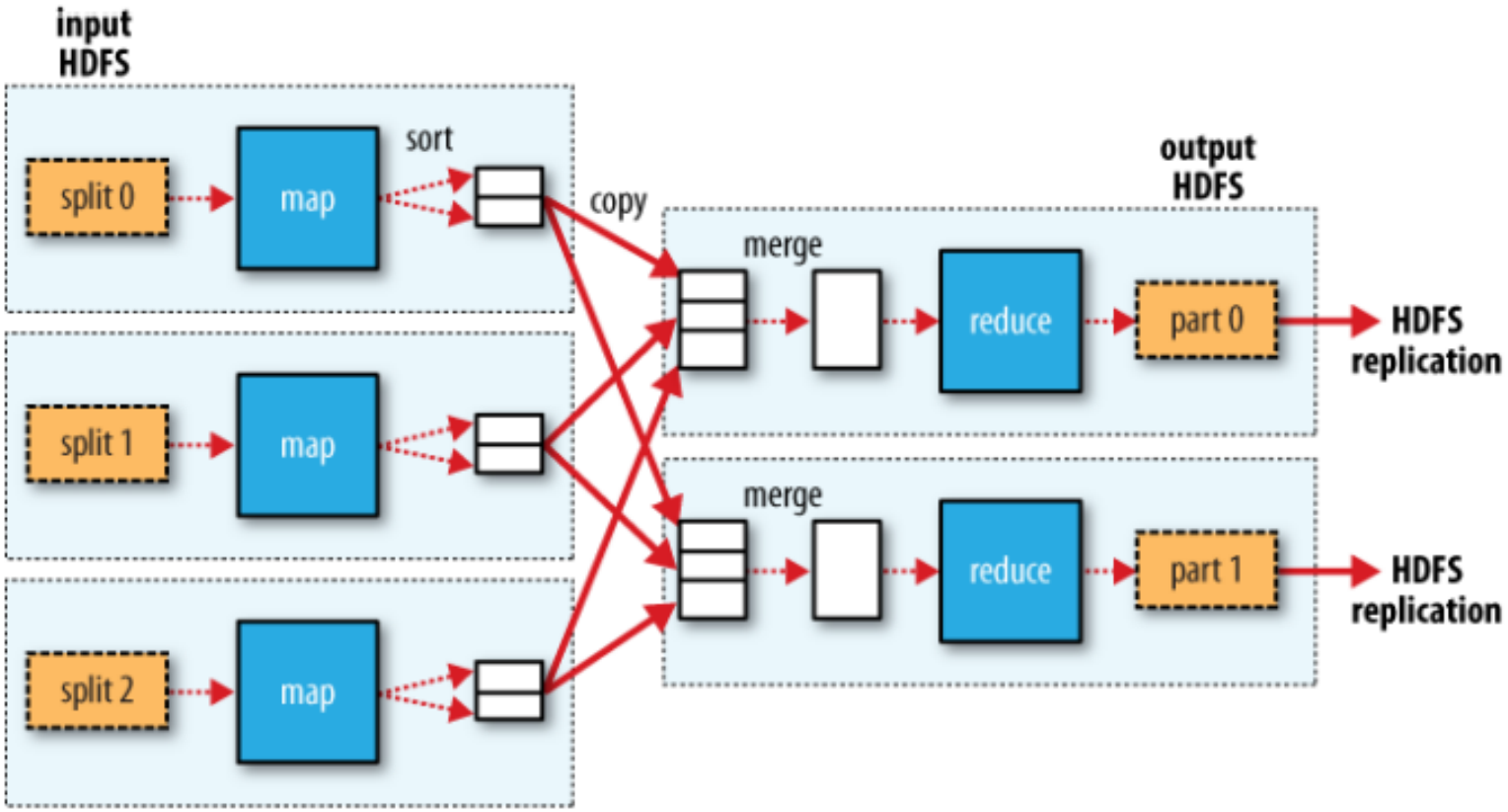
Shuffle and Sort

- MapReduce makes the guarantee that the input to every reducer is sorted by key
- The process by which the system performs the sort—and transfers the map outputs to the reducers as inputs—is known as the shuffle



Ref: Hadoop Definitive Guide, Orielly

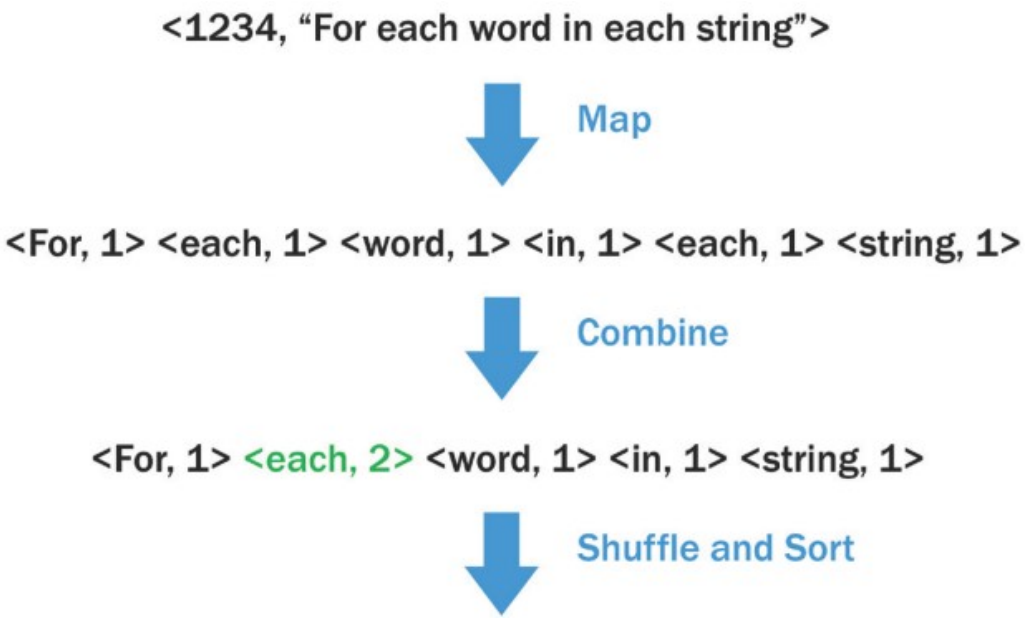
Data Flow with Multiple Reduce Tasks



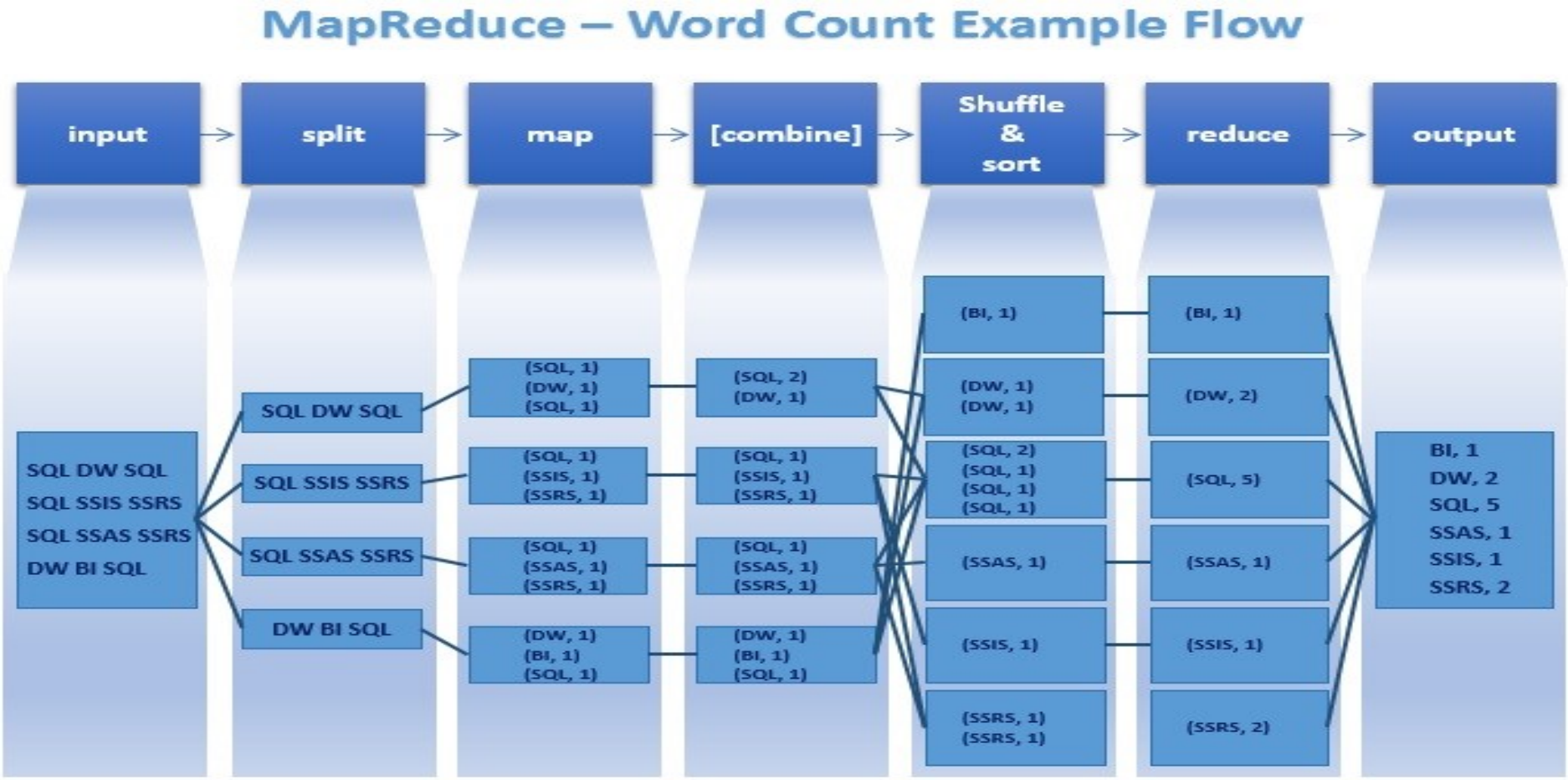
- It's also possible to have zero reduce tasks. This can be appropriate when you don't need the shuffle since the processing can be carried out entirely in parallel
  - E.g. data preparation such as replacing missing values

Combiner Functions

- Many MapReduce jobs are limited by the bandwidth available on the cluster, so it pays to minimize the data transferred between map and reduce tasks
- Hadoop allows the user to specify a combiner function to be run on the map output—the combiner function's output forms the input to the reduce function



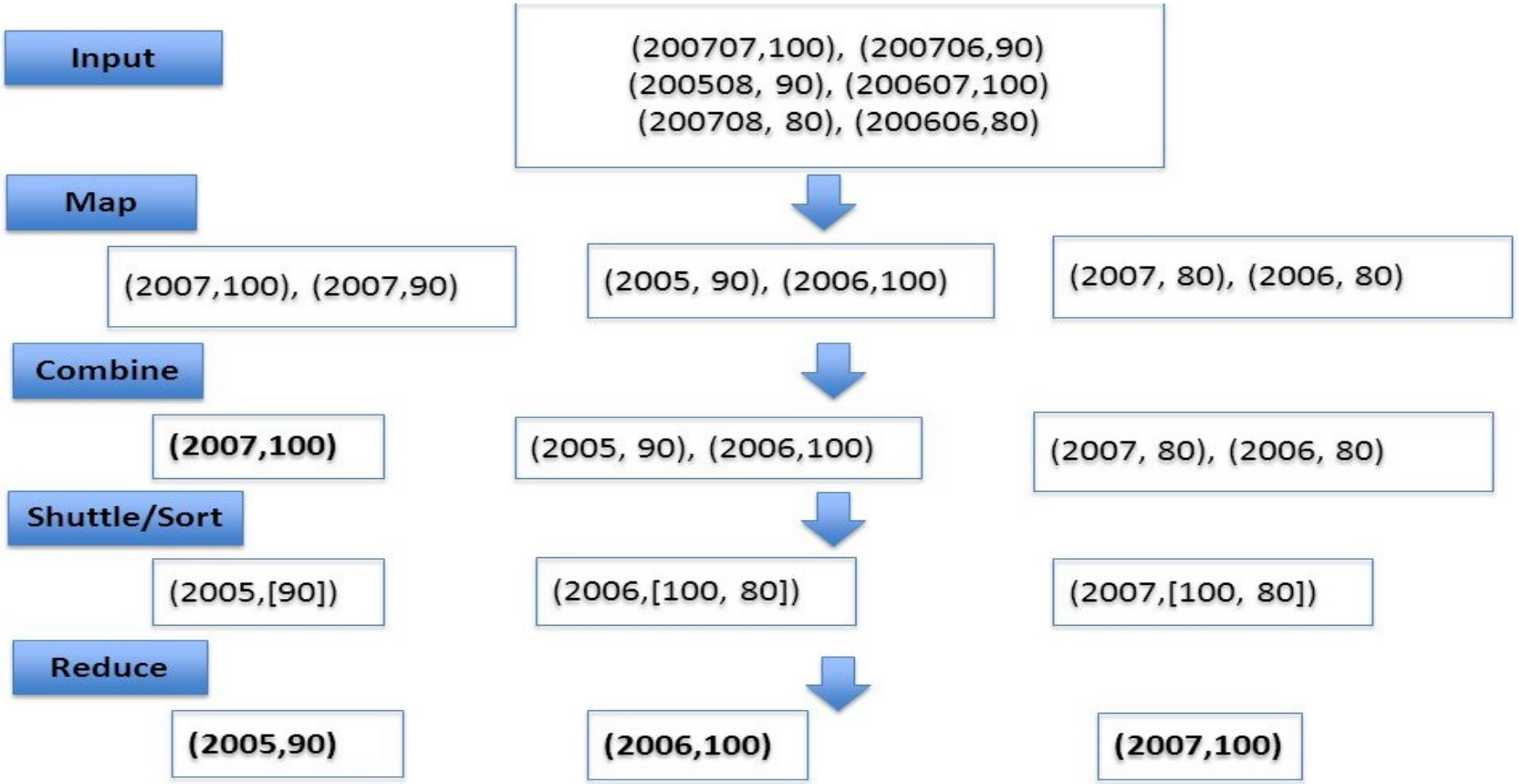
WordCount Example



Source: https://i.stack.imgur.com/IRpoU.jpg

Mapreduce Example

- Given set of tuples <yearmonth, temperature>, find maximum temperature per year



Source: https://slideplayer.com/slide/3271719/

Chaining MapReduce Jobs

- Many complex tasks need to be broken down into simpler subtasks, each accomplished by an individual MapReduce job
  - For example, given a <timestamp, temparature> records, finding a top 5 years with the highest average temperature a sequence of two MapReduce jobs.
  - The first one creates the <year, average_temp> and the second job finds the top 5 years with highest temperature.

Joining data from different sources

- It's inevitable that in data analyses, one needs to pull in data from different sources
- MapReduce can perform joins between large datasets, but writing the code to do joins from scratch is fairly involved
- Rather than writing MapReduce programs, one might consider using a higher-level framework such as Pig, Hive, or Cascading, in which join operations are a core part of the implementation
- If the join is performed by the mapper, it is called a map-side join, whereas if it is performed by the reducer it is called a reduce-side join.

innovate achieve lead

BITS Pilani

Pilani | Dubai | Goa | Hyderabad

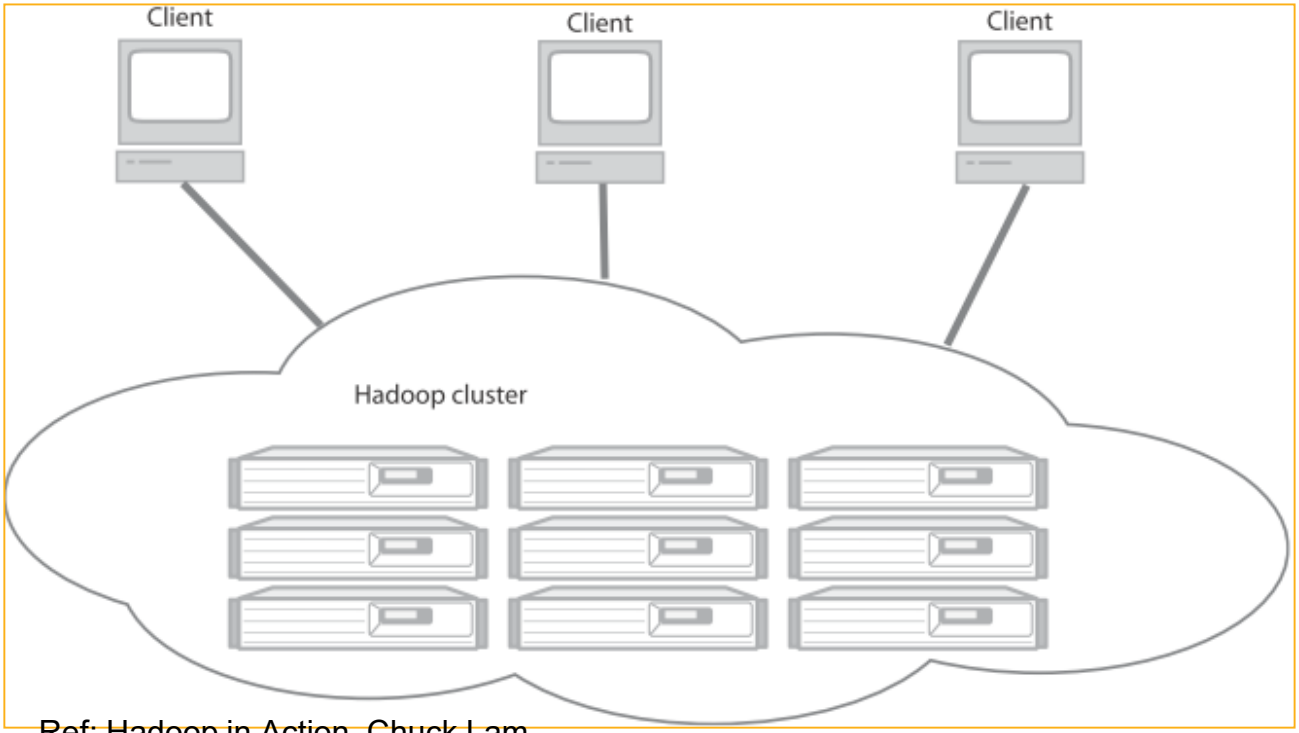
Overview of Hadoop

What is Hadoop?

- Hadoop is an open source framework for writing and running distributed applications that process large amounts of data
- Distributed computing is a wide and varied field, but the key distinctions of Hadoop are that it is
  - Accessible—Hadoop runs on large clusters of commodity machines or on cloud computing services such as Amazon's Elastic Compute Cloud (EC2)
  - Robust—Because it is intended to run on commodity hardware, Hadoop is architected with the assumption of frequent hardware malfunctions. It can gracefully handle most such failures
  - Scalable—Hadoop scales linearly to handle larger data by adding more nodes to the cluster
  - Simple—Hadoop allows users to quickly write efficient parallel code

What is Hadoop?

- Though writing large distributed programs is difficult, Hadoop's accessibility and simplicity give it an edge
- A Hadoop cluster has many commodity machines that store and process large data sets. Client computers send jobs into this computer cloud and obtain results.



Ref: Hadoop in Action, Chuck Lam

Hadoop

- Hadoop provides storage and computing
- Hadoop provides distributed storage through
  - HDFS – Hadoop distributed file system
- Hadoop provides distributed processing through
  - MapReduce – offline computing engine

Hadoop Distributed File System (HDFS)

- Hadoop is not only a data analysis platform but it also handles storage, because you need a place to store data before you can analyze it.
- HDFS is a file system. The data on HDFS can be a file or a directory, like the ordinary file systems
  - built for achieving high availability and scalability

Building Blocks of Hadoop

- Running Hadoop means running a set of daemons on the different servers in the cluster
  - These daemons have specific roles, some exist only on one server, some exist across multiple servers
- The daemons include
  - NameNode
  - DataNode
  - Secondary NameNode
  - JobTracker
  - TaskTracker

NameNode

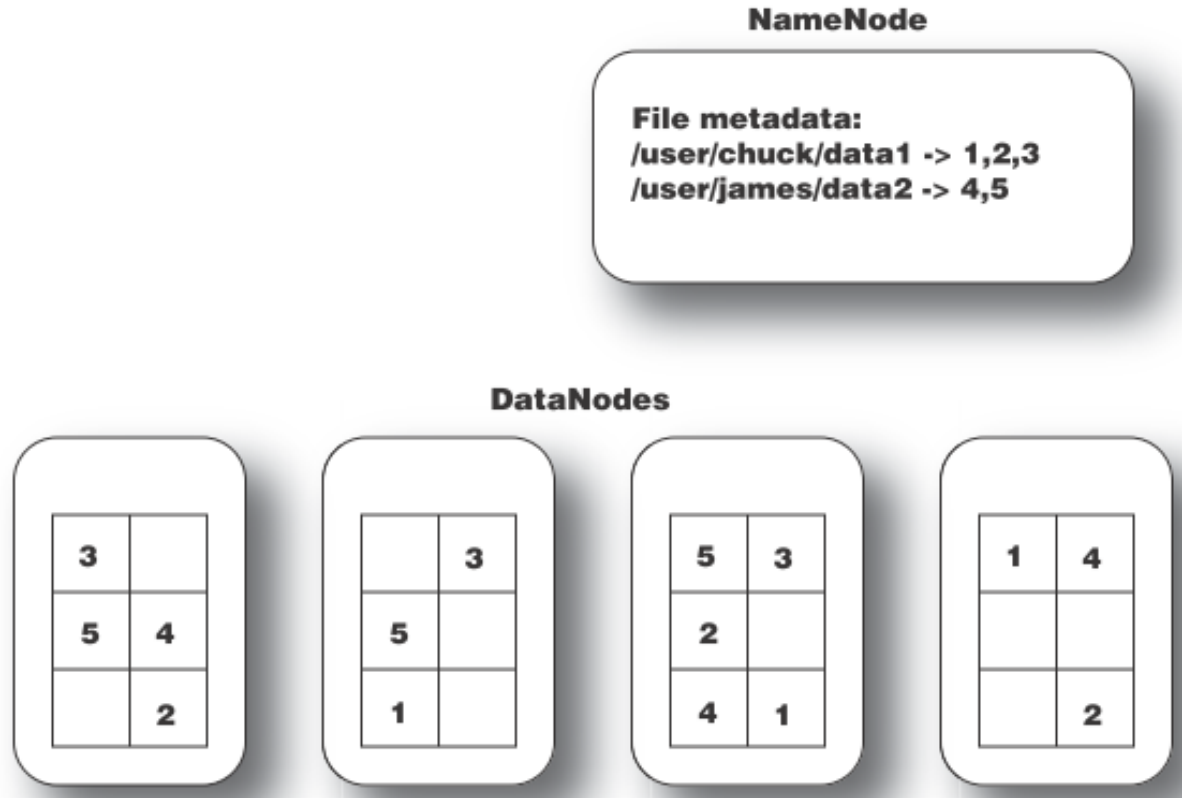
BITS Pilani

- Hadoop employs a master/slave architecture for both distributed storage and distributed computation
- The distributed storage system is called the Hadoop File System , or HDFS
- The NameNode is the master of HDFS that directs the slave DataNode daemons to perform the low-level I/O tasks
- The NameNode is the bookkeeper of HDFS
  - keeps track of how your files are broken down into file blocks, which nodes store those blocks, and the overall health of the distributed file system
- The function of the NameNode is memory and I/O intensive

DataNode

- Each slave machine in the cluster will host a DataNode daemon to perform the work of the distributed filesystem
  - reading and writing HDFS blocks to actual files on the local filesystem
- When HDFS client want to read a file,
  - the NameNode will tell HDFS client which DataNode each block resides in
  - client communicates directly with the DataNode daemons to process the local files corresponding to the blocks i.e. each block is a file in the local fie system
- Furthermore, a DataNode may communicate with other DataNodes to replicate its data blocks for redundancy.

NameNode



Each block is replicated in Two other Data nodes.

- The NameNode keeps track of the file metadata—which files are in the system and how each file is broken down into blocks
- The DataNodes provide backup store of the blocks and constantly report to the NameNode to keep the metadata current.

DataNode to NameNode

- DataNodes constantly repor to the NameNode
- Upon initialization, each of the DataNodes informs the NameNode of the blocks it's currently storing
- After this mapping is complete, the DataNodes continually poll the NameNode to provide information regarding local changes as well as receive instructions to create, move, or delete blocks from the local disk.

Secondary NameNode

- The NameNode is a single point of failure for a Hadoop cluster.
- Secondary NameNode runs in a separate system other than NameNode.
- Secondary NameNode doesn't receive or record any real-time changes to HDFS. Instead, it communicates with the NameNode to take snapshots of the HDFS metadata at intervals defined by the cluster configuration
- When NameNode fails, these snapshots help minimize the downtime and loss of data.

JobTracker

- The JobTracker daemon is the liaison between application and Hadoop
- Once job is submitted to cluster, the JobTracker determines the execution plan by determining which files to process, assigns nodes to different tasks, and monitors all tasks as they're running
  - Should a task fail, the JobTracker will automatically relaunch the task, possibly on a different node, up to a predefined limit of retries
- There is only one JobTracker daemon per Hadoop cluster. It's typically run on a server as a master node of the cluster.

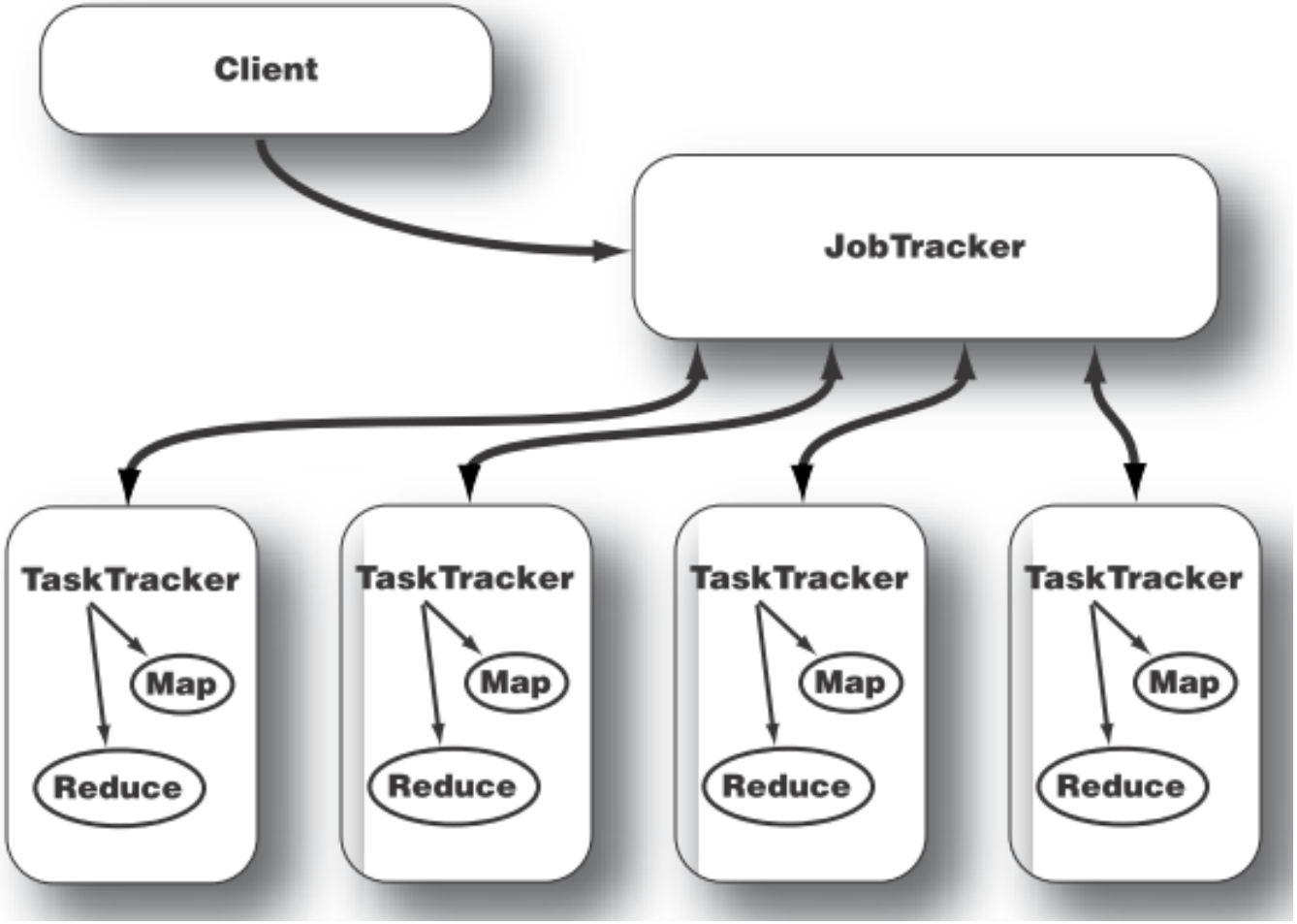
TaskTracker

- As with the storage daemons, the computing daemons also follow a master/slave architecture:
  - The JobTracker is the master overseeing the overall execution of a MapReduce job
  - TaskTrackers manage the execution of individual tasks on each slave node
- Each TaskTracker is responsible for executing the individual tasks that the JobTracker assigns

TaskTracker

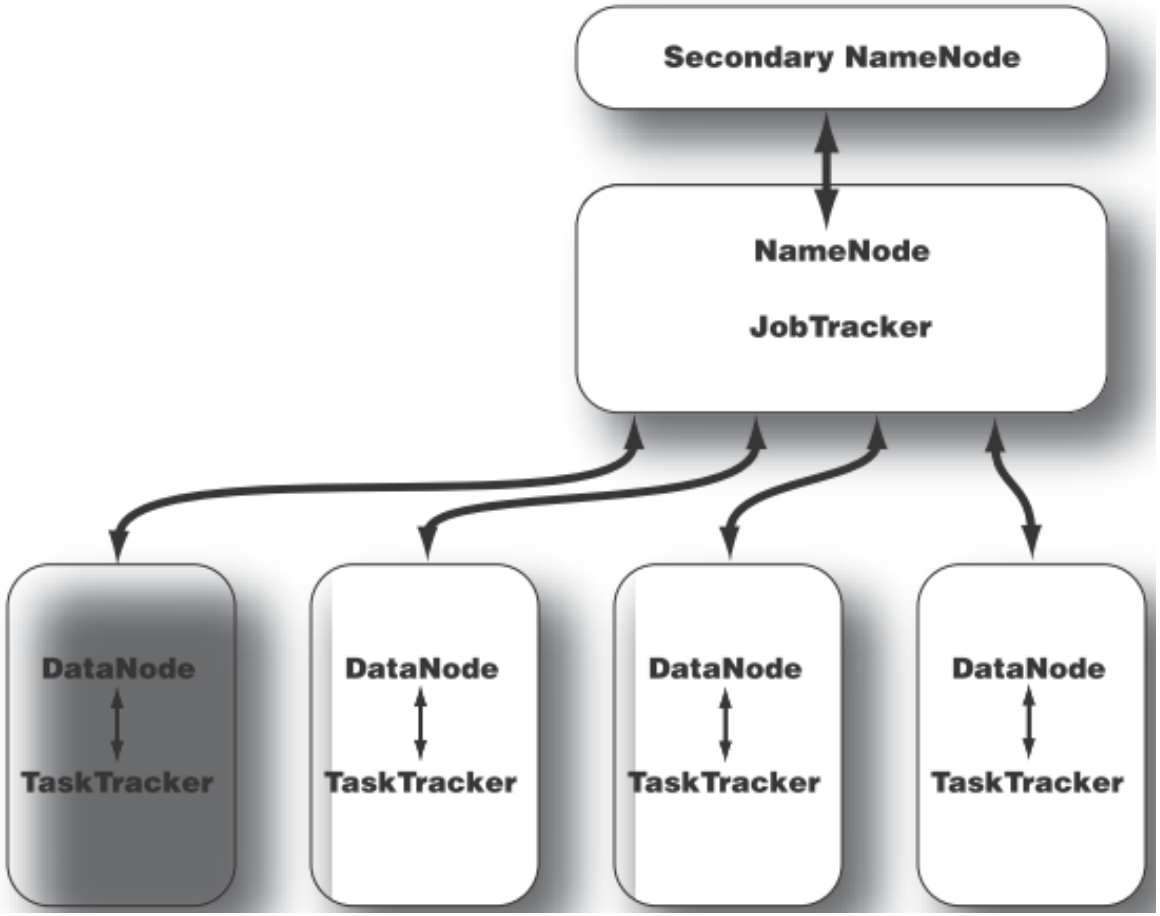
- Although there is a single TaskTracker per slave node, each TaskTracker can spawn multiple JVMs to handle many map or reduce tasks in parallel
- One responsibility of the TaskTracker is to constantly communicate with the JobTracker
  - If the JobTracker fails to receive a heartbeat from a TaskTracker within a specified amount of time, it will assume the TaskTracker has crashed and will resubmit the corresponding tasks to other nodes in the cluster.

JobTracker and TaskTracker



- After a client calls the JobTracker to begin a data processing job, the JobTracker partitions the work and assigns different map and reduce tasks to each TaskTracker in the cluster

NameNode and JobTracker



- NameNode and JobTracker are masters and the DataNodes and TaskTrackers are slaves.

References

- Hadoop in Action, Chuck Lam

https://www.manning.com/books/hadoop-in-action

References

- Hadoop: The Definitive Guide, 4th Edition [Book] - O'Reilly Media

Q&A

innovate achieve lead

BITS Pilani
Pilani Campus

Thank You