Part 2:
The MapReduce Paradigm

Ajay Deshpande, CTO.
Rakya Technologies Pvt Ltd
www.rakya.com

Recap...

- OLTP and OLAP Systems
- Rise of Bigdata
- Indexing documents – the Inverted Index
- Indexing Web Pages – rise of Hadoop
- Assignments: SOLR and NUTCH
- Map Reduce: Distributed Computing

How would you solve

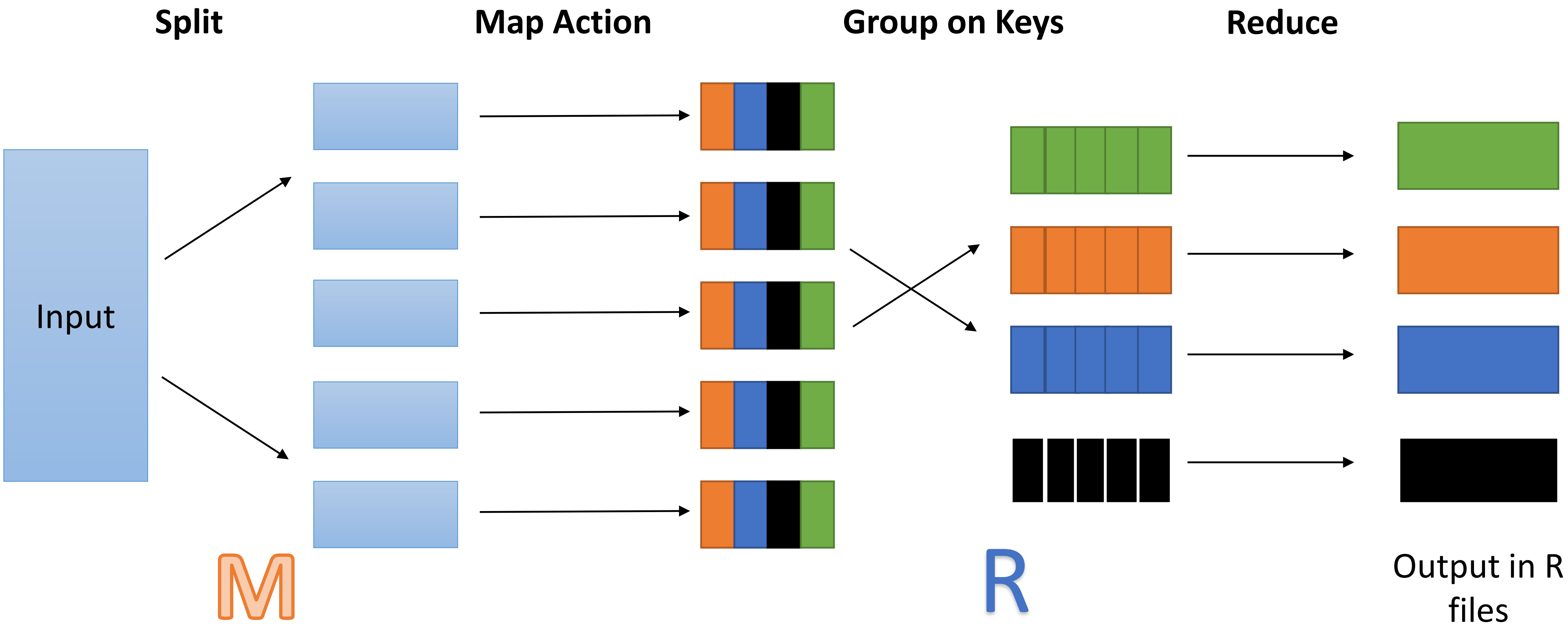
a LARGE problem by Brute Force

Map Reduce

The Map Reduce Paradigm

- A (functional) Programming Model to process large data sets
- Users specify
  - Map: processes a key/value pair -> intermediate key/value pairs
  - Reduce: merges all intermediate values associated with a key
- Such Programs can be easily parallelized and executed
- User need not be worried about
  - Data partitioning, Scheduling across multiple machines
  - Failure Handling, Managing inter-machine communication
- Power of parallel and distributed systems available to everyone

The MapReduce Algorithm



Example: Word Count

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

```
reduce(String key, List values):
  // key: a word
  // values: a list of counts
  int result = 0;
  for each count in values:
    sum += count;
  Emit(sum);
```

Example: URL Access Frequency

```
map(String name, String logs):
  // name: log file name
  // logs: log contents
  for each line w in logs:
    Parse Line into constituents
    EmitIntermediate(URL, 1);
```

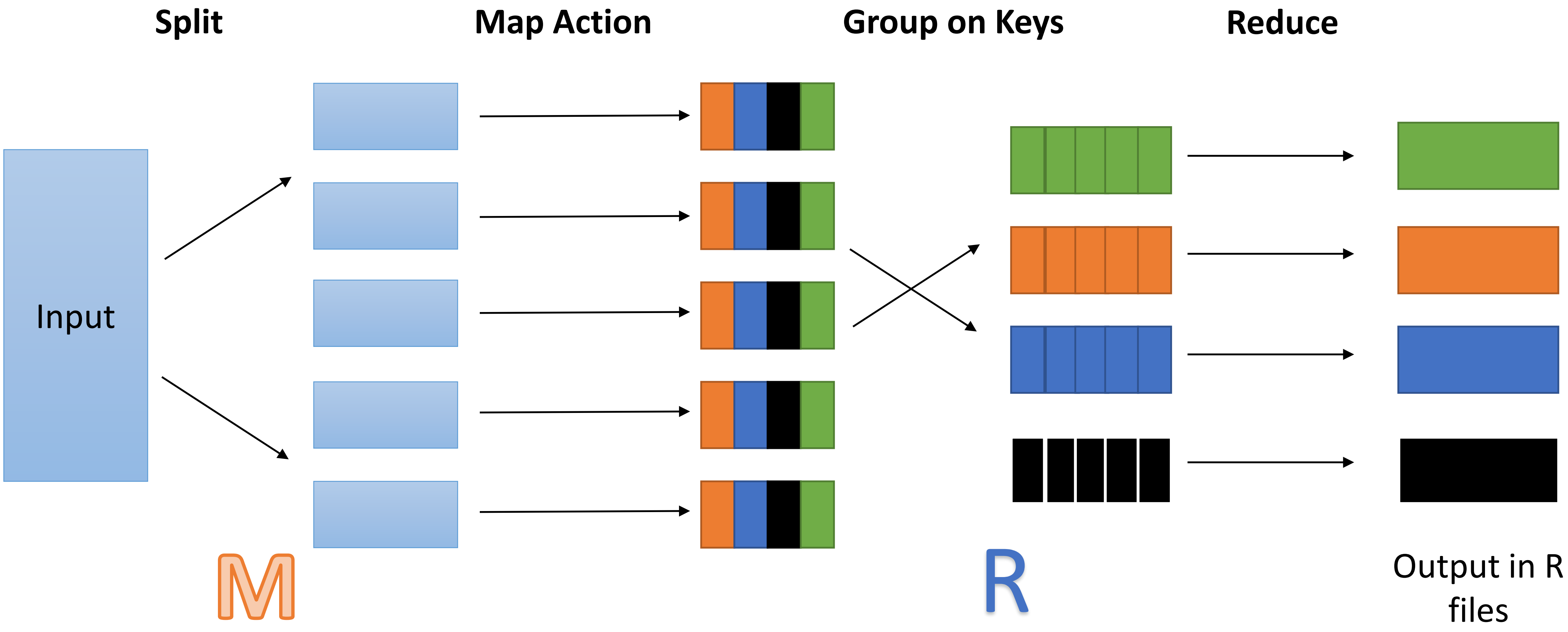
```
reduce(String URL, List values):
  // URL: a url
  // values: a list of counts
  int result = 0;
  for each count in values:
    sum += count;
  Emit(sum);
```

Example: “Who Refers to me” List

```
map(String SRC, String content):
  // SRC: url of a source page
  // content: HTML contents
  Parse HTML content
  for each link A in content:
    EmitIntermediate(A, SRC);
```

```
reduce(String URL, List refs):
  // URL: a url
  // refs: list who refers to me
  myRefs = empty
  for each r in refs:
    concat r to myRefs;
  Emit(URL, myRefs);
```

The MapReduce Algorithm



The MapReduce Algorithm…

- Input is split into M pieces (typically each of 64 MB)
  - The Master manages data for work progress across the network
- The Master starts up (and tracks) M Map Tasks
- Mapper Worker: execute Map on its input piece
  - Writes output to local file partitioning into R sets based on the intermediate keys
- Master starts Reduce Tasks as data becomes available
- Reduce Worker: Reads data from multiple sources
  - Sorts data on keys and executes reduce for each key
- Output data available in R files

Issues in MapReduce Implementations

- Handling Stragglers (Machines that have slowed down)
- Partitioning and Ordering output data in user determined
- Reducing network load using combiner function
  - Executed by machines immediately after map for partial merges
- Handling Process / Machine failures
  - Restart of the failed tasks

Advantages of MapReduce

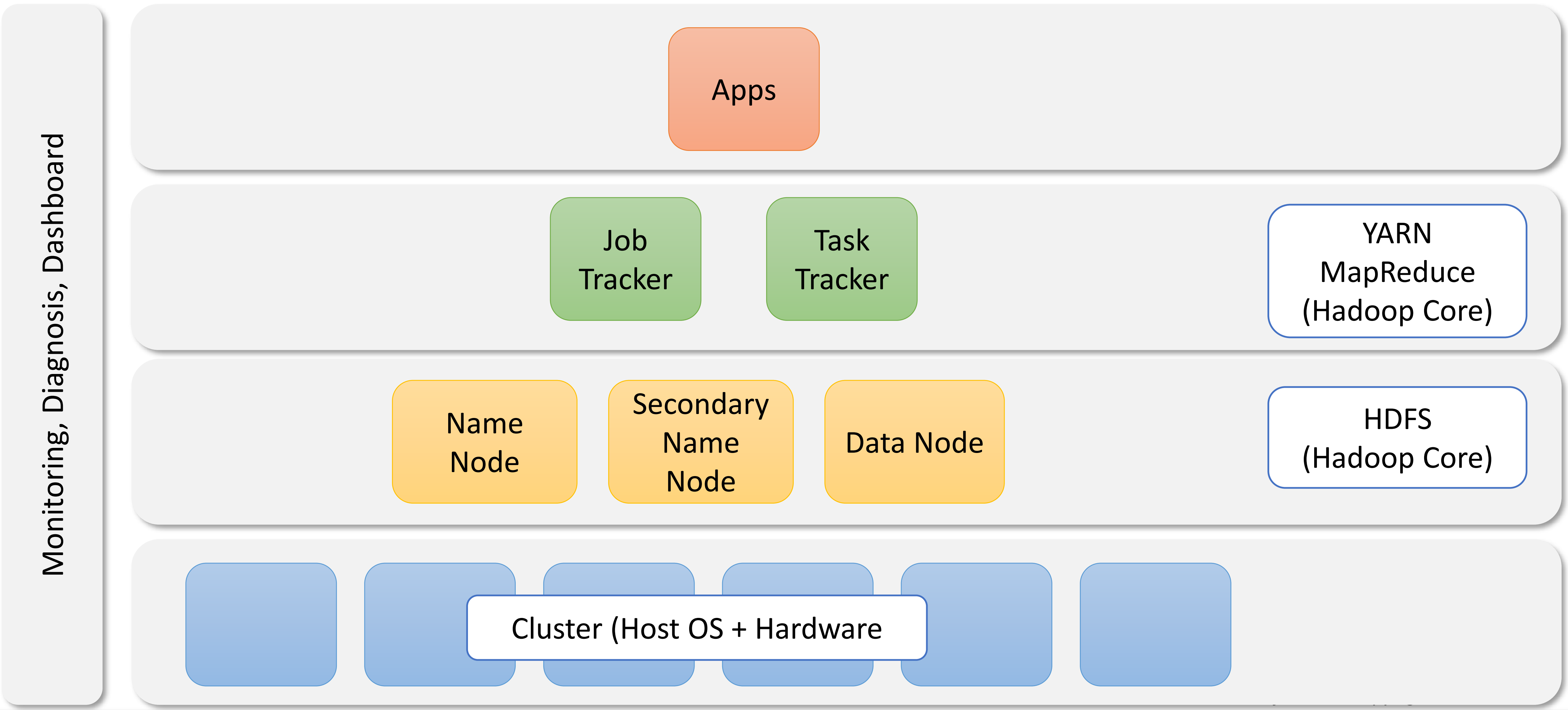
- Makes Distributed Computing available to the ***common*** programmer
- Inherent Simplicity
  - Primarily because the ***infrastructure*** code does not clutter the algorithm
- Code is more maintainable
- Increases Resilience – failures can be gracefully handled

Diving Into Hadoop

Revisiting: Assumptions / Goals of Hadoop

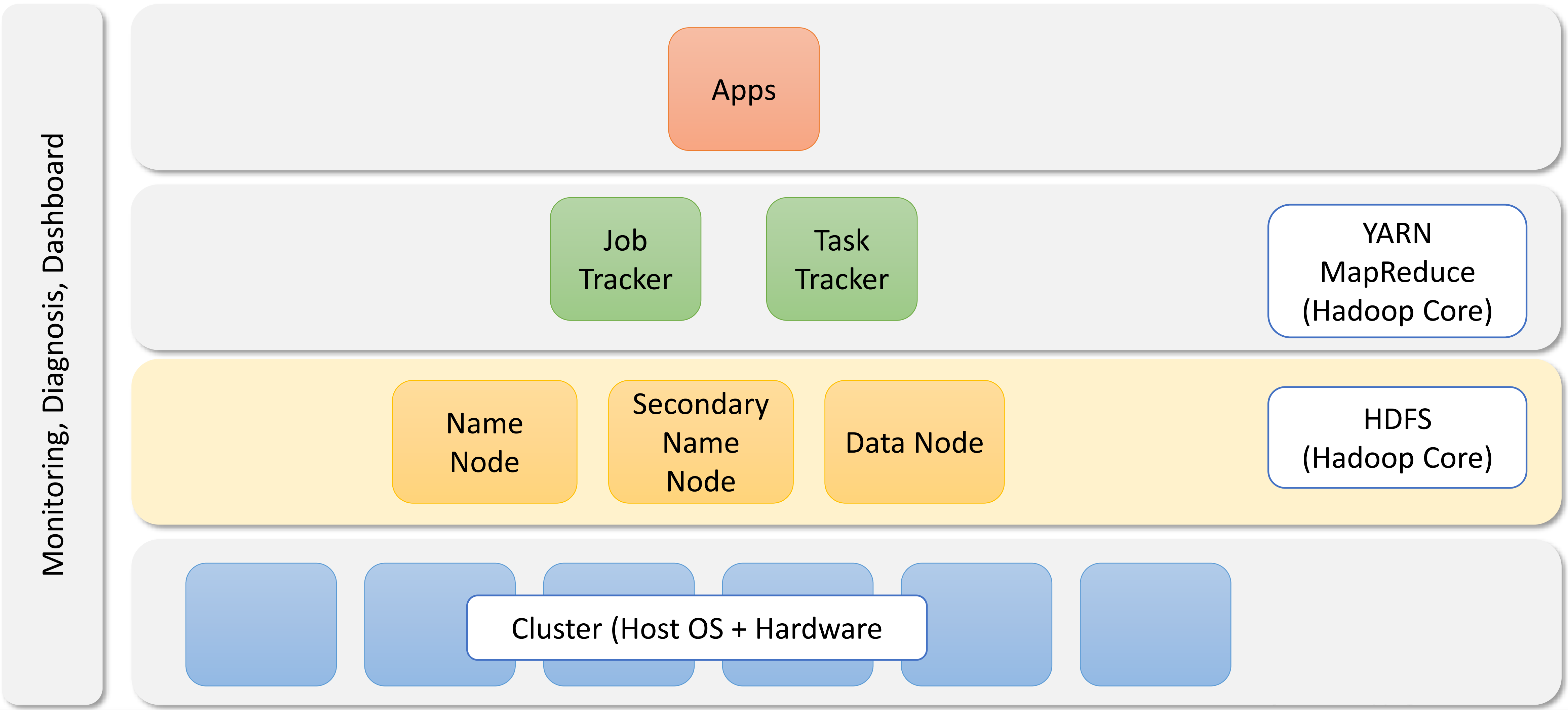
- Hardware will fail
- Tuned for batch processing / streaming data
  - Does not work well for interactive applications
- Works with huge data sets
  - Moving computation is cheaper
- Portability across platforms
- Does not allow random changes to files
  - Only Append / Truncate available
  - Enables simple concurrency control semantics
- No specialized hardware

Hadoop Layers



Implementation: HDFS

Hadoop Layers



Understanding HDFS

- One Master (NameNode) and many Slaves (DataNodes) Architecture
- NameNode manages Files Namespace and access
  - Manages the metadata of the file system
  - Open, Close, Rename files
  - Instructs DataNodes on file operations
- DataNodes store the data blocks (on their host OS)
  - Participate with client application in read and write
  - DataNodes check in with the NameNode via a Heartbeat
  - NameNode always works with DataNodes in responses to Heartbeat
- Capacity can be increased by adding new DataNodes

Understanding HDFS…

- The file system name space is a tree (as in Unix)
- Files are write once and read many times
  - Only one writer simplifies concurrency implementation
- Data maintained in many places: replication factor
  - Replication happens at block level (64MB default size)
  - Same blocks are stored on multiple nodes
- If a DataNode crashes, its blocks are replicated on to other nodes

Creating a new file in HDFS

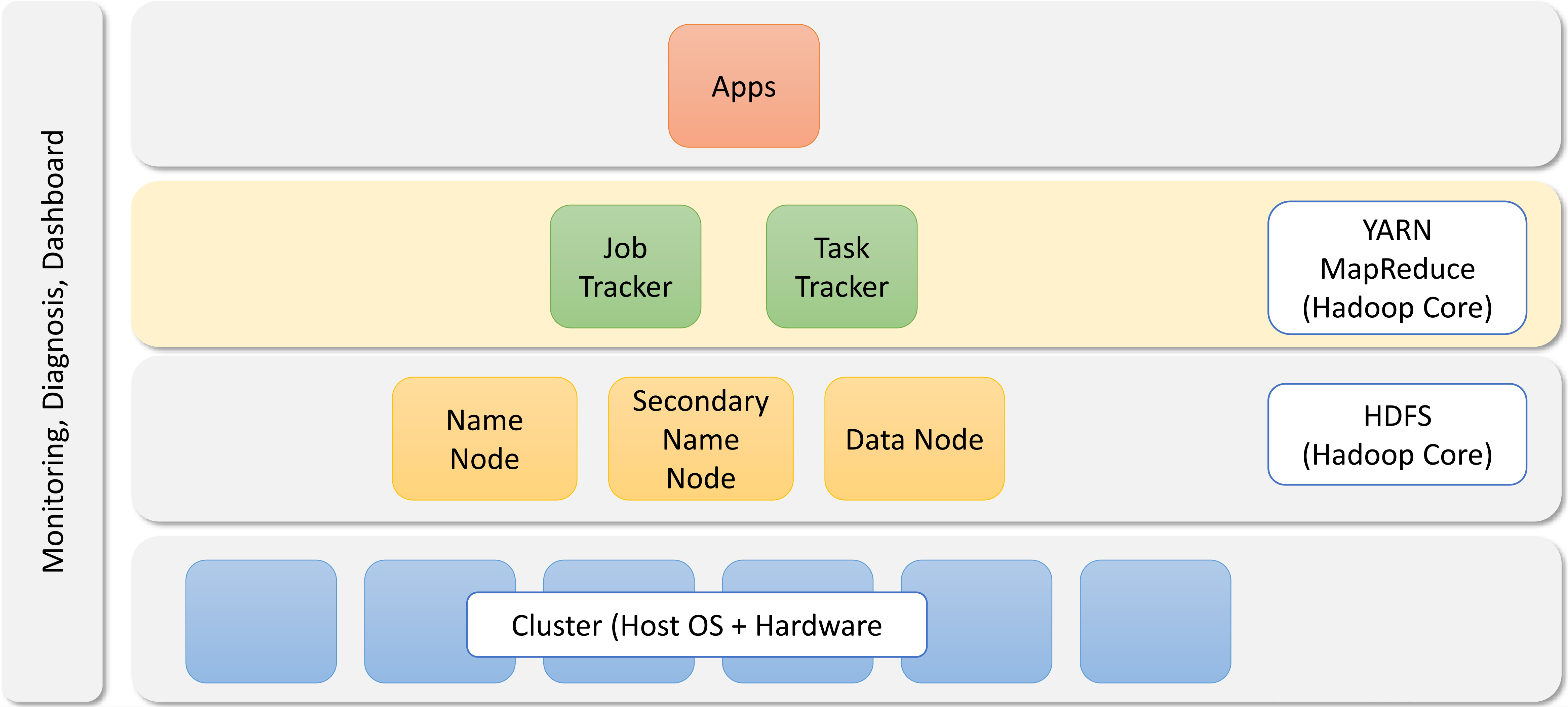
- Client sends CreateFile to NameNode
  - Response has a list of DataNodes for the first block
- Do Until No More Blocks to be written
  - Client writes the block to the first DataNode
    - The first DataNode forwards it to the next in the list and so on
    - ACK back to the client from the first DataNode after all written
  - Client asks NameNode for a list of DataNodes for the next Block
- Client closes file with NameNode

Reading an existing file from HDFS

- Client sends OpenFile to NameNode
  - Response has a list of blocks for that file
    - **B1**: DN5, DN3, DN1
    - **B2**: DN6, DN1, DN5
    - ……
- Do Until No More Blocks to be read
  - For this block send a read request to the first DataNode
    - If DataNode not available ask the next in the priority list
  - If all replicas for a block are unavailable then file read fails
- Client closes file

Implementation: Hadoop Framework for MapReduce

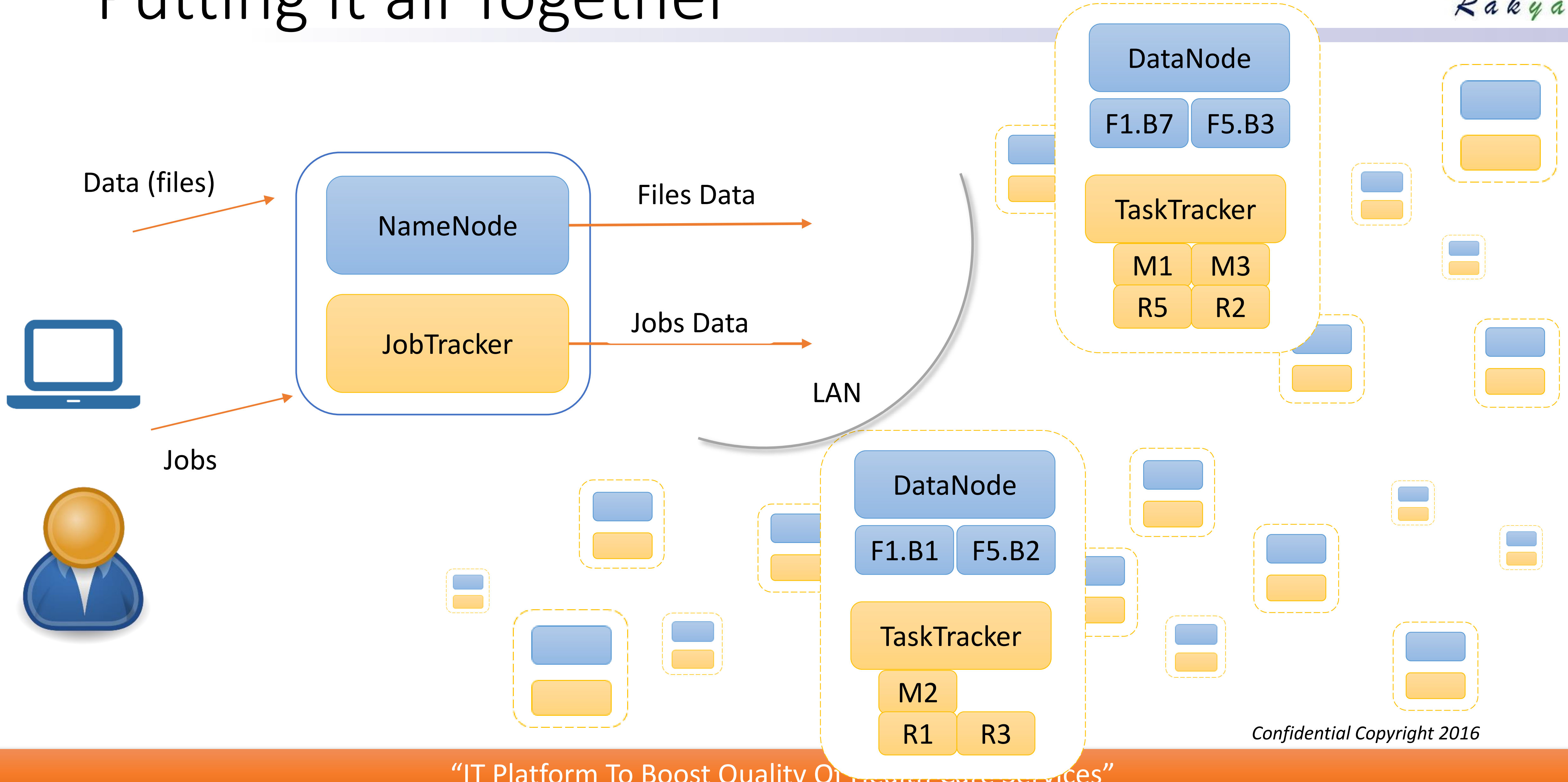
Hadoop Layers



The MapReduce Subsystem

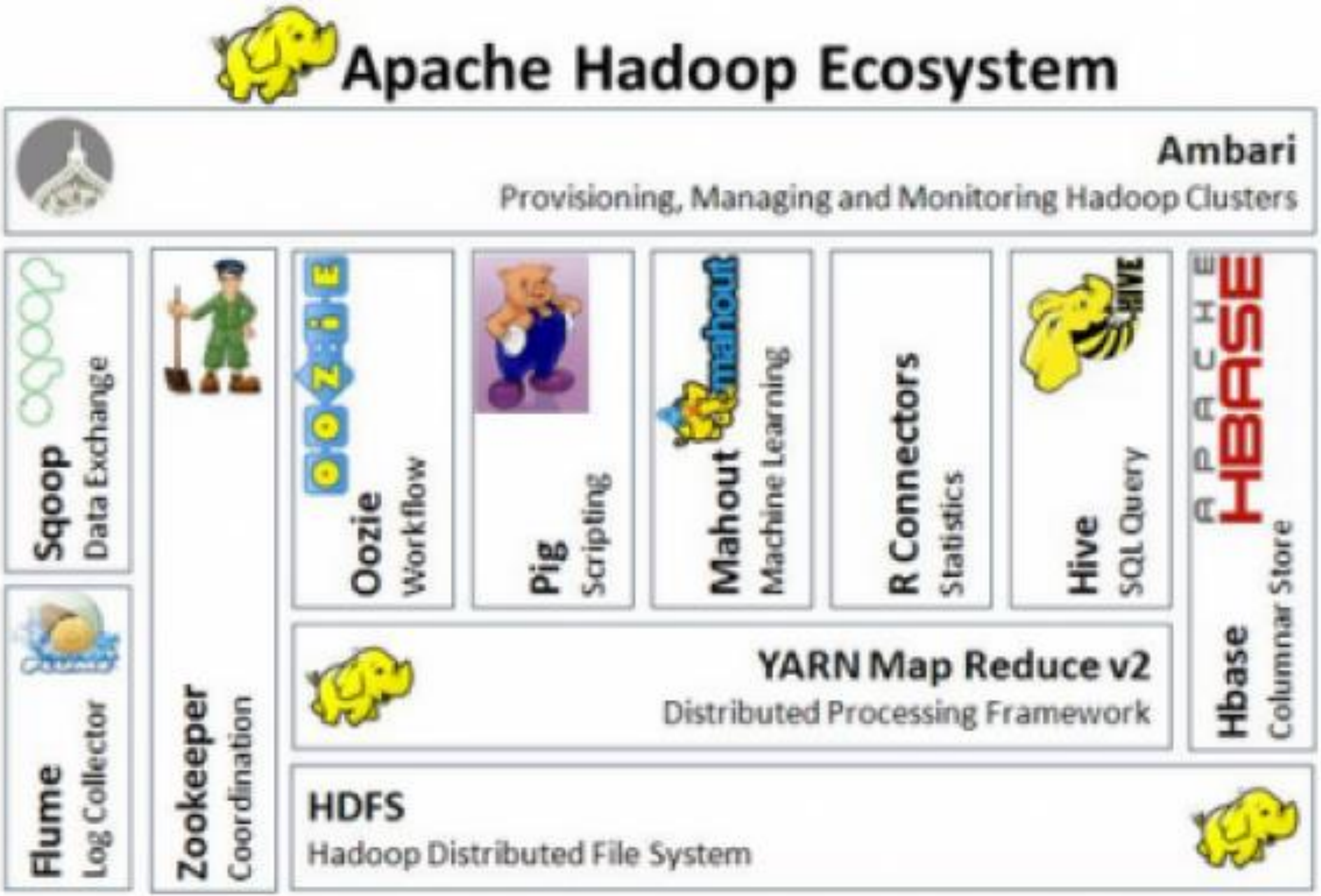
- Master (JobTracker) Slave (TaskTracker) setup again
- The JobTracker schedules and monitors the tasks
- User provides the Map and Reduce code as a JAR file
- The inputs and outputs of these functions are files in HDFS
- Job Execution Workflow
  - User submits a job via a JobClient to JobTracker
  - JobTracker prepares an execution plan
    - Input size determines number of Mappers and Reducers needed
  - JobTracker ships JAR file to all TaskTrackers to execute
  - Each TaskTracker will run its tasks and report status
  - JobTracker ensures all sub tasks complete

Putting it all Together



Hadoop Ecosystem Today

http://www.ossmentor.com/2015/07/what-is-hadoop-ecosystem.html



Organizing the Web: The PageRank Algorithm

What Do you know about Google?

- Google comes from misspelling googol (1 followed by 100 zeros)
  - Originally called as Backrub
- Founders wanted to sell company for USD 1 mil to Excite in 1999 (Larry Sergey wanted to complete their PhDs)
  - Today its worth more than 300,000 times that value
- No part of any office is more than 150 feet away from food ☺
- Number of searches per second > 2 million
- They regularly rent goats to “mow” grass on their lawns ☺
- More than 100 popular products (Maps, YouTube, Waze, Android…)
- Maps has photographed more than 10 million KMs of roads
- Is acquiring one company a week since 2008!

How Would You Determine

- Relevance of a web page for your query?
  - Page should contain what you are looking for. Relevance.
  - Everyone else thinks the page is important. Ranked high.
- Generally: more links implies more importance
  - However, even if there is only one link from an "important" page that link becomes significant
- Birth of Google: original PageRank (PR) algorithm developed by Larry Page and Sergey Brin

Modeling the Web Surfer

- WWW is a directed graph: nodes are pages, links are edges
- Surfer visits a page if s/he deems it as important (PR is high)
- The probability that s/he clicks on a link on a page is inversely proportional to the number of links
  - Implies probability that s/he arrives at a page is the SUM of probabilities that s/he clicks on a link to this page
- Further s/he suddenly switches to a completely new page (not part of the current link structure)
  - So a page always has some probability that it will be selected independent of the number of links coming into it

The PageRank Algorithm

- A page's rank PR is proportional to the sum of the ranks of its back links
- In other words, PR is recursively defined in terms of PR of pages that link to it

$$PR(A) = 1 - d + d\ [\ PR(T1)/C(T1) + \ldots + PR(Tn)/C(Tn)\ ]$$

Where:

- PR(A) is the PageRank of page A
- PR(Ti) is the PageRank of pages Ti which link to page A,
- C(Ti) is the number of outbound links on page Ti
- d is a damping factor which can be set between 0 and 1

PageRank…

- Lets look at

  PR(A) = 1 - d + d [**PR(T1)/C(T1) + … + PR(Tn)/C(Tn)**]
- Implies: PR of pages Ti does not influence the PR of page A uniformly
- PR of page T is always weighted by the number of outbound links C(T) on page T.
- The more outbound links in page T has, the less will be the benefit for page A
- Summation => More Links to A increases A's Rank

PageRank…

- Lets look at **<u>d</u>**

$$PR(A) = 1 - \underline{\mathbf{d}} + \underline{\mathbf{d}} \, [\, PR(T1)/C(T1) + \ldots + PR(Tn)/C(Tn) \,]$$

- Summation => More Links to A increases A’s Rank
- d is damping factor (set between 0 and 1)
  - Higher value => the more likely the surfer continues in the same chain
  - Lower indicates s/he jumps to a completely new page
- Controls the extent of influence linking pages exert on page A
  - Setting d to 0 => no effect of linking pages

Implementing the PageRank

- Google uses an approximative, iterative computation of PageRank values
- Each page is assigned an initial starting value
- New values for all pages are calculated in iterations based on the equations
  - Eventually the PRs of all page converge (don’t change much with more iterations)
- Google’s magic recomputes these ranks regularly
- As of yesterday, # of web pages in Google’s PR algorithm > 4.5 Billion!!
- Today they use more than 200 factors to determine search results

How to Boost Your Website

- Search Engine Optimization (SEO)
- Remember PageRank is page level and not website level
- Start with Good / Relevant content
- Swap links with websites which have high PageRank™ value
- Raise the number of inbound links
  - For Eg. Advertise your website on other sites)
- Add new pages to your website (as many as you can)

THANK YOU !!

DAJAY0@YAHOO.COM