Hadoop

Historic view

- As of the rise of the Web 2.0:
  - Very large volumes of data being collected
    - Web logs first, then social media, web apps, etc.
    - Data from all type of sensors (phone, cars, …)
    - Metadata from communication networks
  - Analytics on this data, of great value

- *Big Data*: different from earlier DBs by…
  - **Volume**: much larger amounts of data stored
  - **Velocity**: much higher data ingest rate
  - **Variety**: many data types, beyond relational data

Introduction

- Large increase in the size of disks
- Access-to-disk time has not decreased proportionally

- Obvious solution: use multiple disks and work in parallel!

- Problems:
  - Hardware failures

    HDFS!
  - Many analysis tasks are not completely parallelizable

    MapReduce!

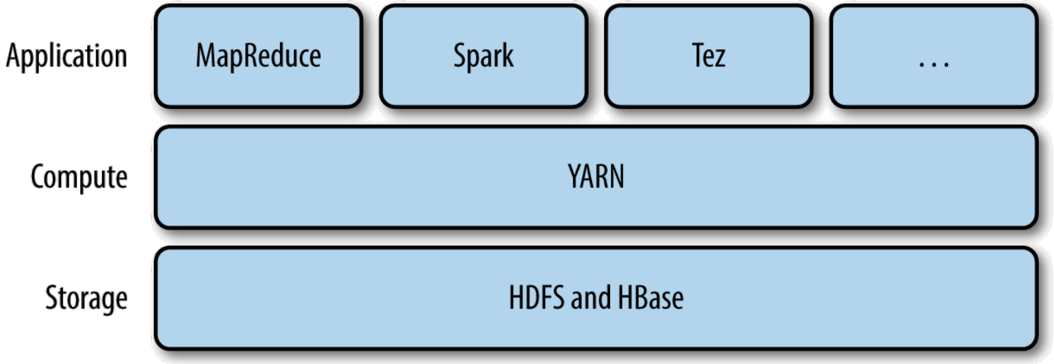
- **Hadoop**: General purpose storage and analysis platform for *big data*

HADOOP

Based on Google's work from early 2000's

Hadoop

- 3 building blocks
  - HDFS: distributed file system
  - YARN: Resource (cluster) manager
  - Map-reduce: parallel computing paradigm



- Splits data and computation across (thousands of) hosts
  - A Hadoop cluster scales computation capacity, storage capacity and I/O bandwidth by simply adding nodes.

Distributed File Systems

- Store files across many nodes while giving a single-system view
  - Need to address the distribution of files across nodes
  - Unified file system, with file names and directories
    - Users do not bother where the file actually is stored
  - Designed to store very large files
    10s MB to 100s GB
  - Highly scalable for large data-intensive applications
    - E.g., 10K nodes, 100 million files, 10 PB

  - E.g., Google File System (GFS), Hadoop File System (HDFS), CODA, Google Colossus

Distributed File Systems

- Files, divided into **blocks**

- Remember the concept of blocks in the context of disks:
  - Minimum amount of data that a disk can read/write.
  - Usually, disk blocks: 512 bytes; regular filesystem blocks: few KBs

- Blocks, file's chunks, are stored as independent units
  - Spread through nodes, replicated for availability
  - Much larger unit: 128 MB by default.
  - Why so large?
    - To minimize the (relative) cost of seeks
  - Unlike a regular filesystem, if the file is smaller than block-size, it doesn't occupy a full block's storage.

Distributed File Systems

- **Blocks** are good for:
  - Make easy to handle files larger than any disk in the cluster
    - Blocks from a file can be stored on different disks
  - Simplifying storage management (e.g., blocks have fix size) and other concerns
  - Replication, to provide fault tolerance and availability.
    - Usually 3 replicas (replication factor, user defined per file)
    - If a block becomes unavailable, a copy can be read from another location in a transparent way
    - Inaccessible blocks can be replicated from its alternative locations to other live machines to ensure replication factor
    - Multiply data transfer bandwidth

Hadoop Distributed File Systems (HDFS)

- **HDFS**: filesystem designed for storing *very large files* with *streaming data access* patterns, running on clusters of *commodity hardware*.

  - *Very large files*: 100+ MB, GB, TB

  Data coherence

  - *Streaming data access*: **write-once, read-many** pattern
    - Various analyses can be performed on the data.
    - Time to read the whole dataset, more important than the latency in reading the first record.

  - *Commodity hardware*: expensive/highly-reliable hardware not required
    - The chance of node failure across a cluster is high
    - Designed to carry on working without a noticeable interruption to the user when this happens.

Not useful if: many small files, need low-latency data access, multiple writers and file modifications

Hadoop Distributed File Systems (HDFS)

Master–worker pattern

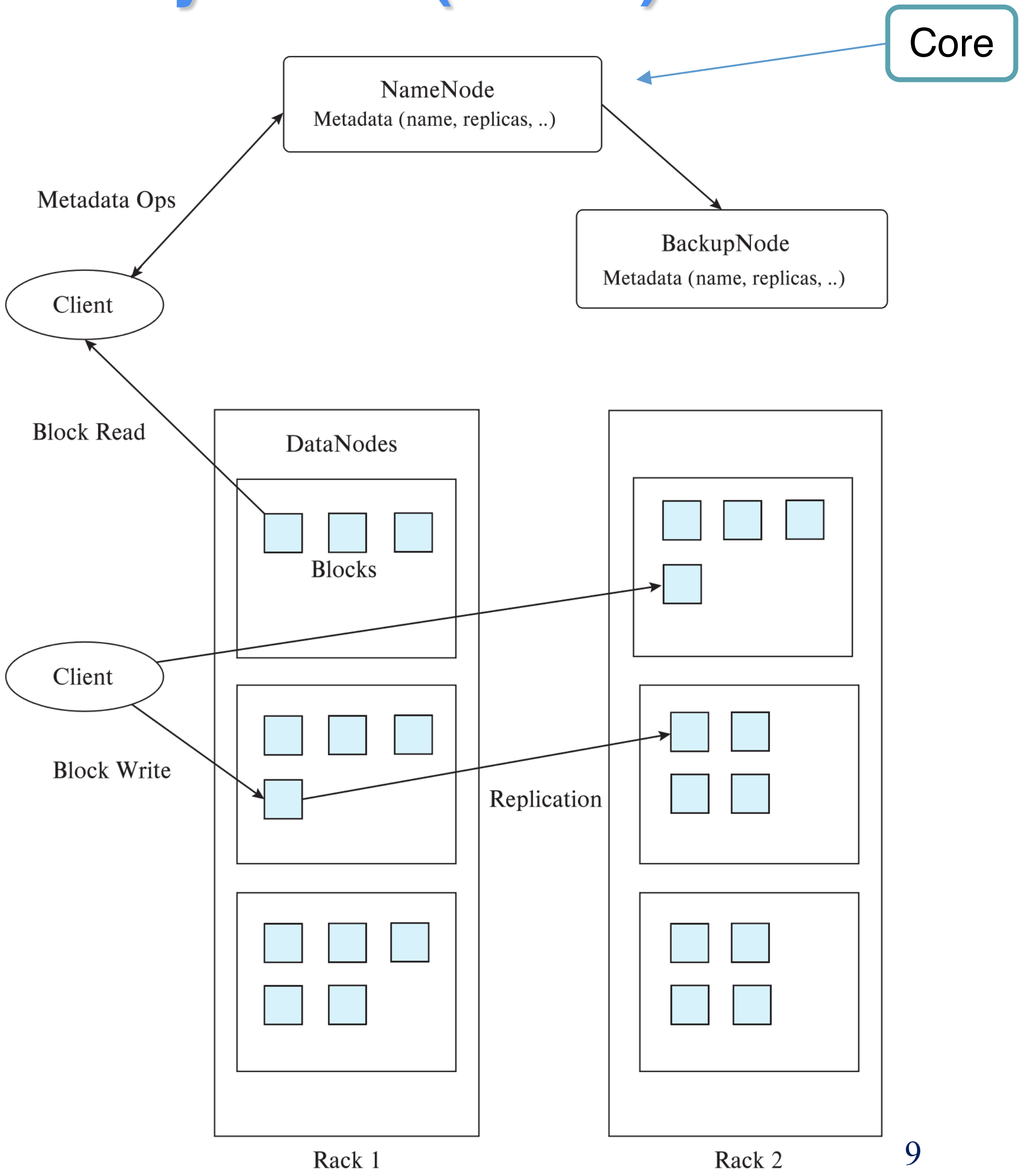
- A HDFS cluster is composed of:
  - A single **NameNode** (master)
    - Manages the filesystem namespace
    - Maintains the directory system for hierarchical file organization (directories/subdirectories)
      - Stored persistently on the local disk
    - Maps filename to sequence of node-block ids
      - Not stored persistently: reconstructed from DataNodes when system starts.
  - Many **DataNodes** (workers)
    - Store and retrieve blocks when instructed so
      - Each **block** replica is represented by **two files**:

        1) One containing the data itself

        2) Another one with block's metadata (checksums, generation stamp, …).
    - Report to the NameNode periodically with lists of blocks that they are storing.

Hadoop Distributed File Systems (HDFS)

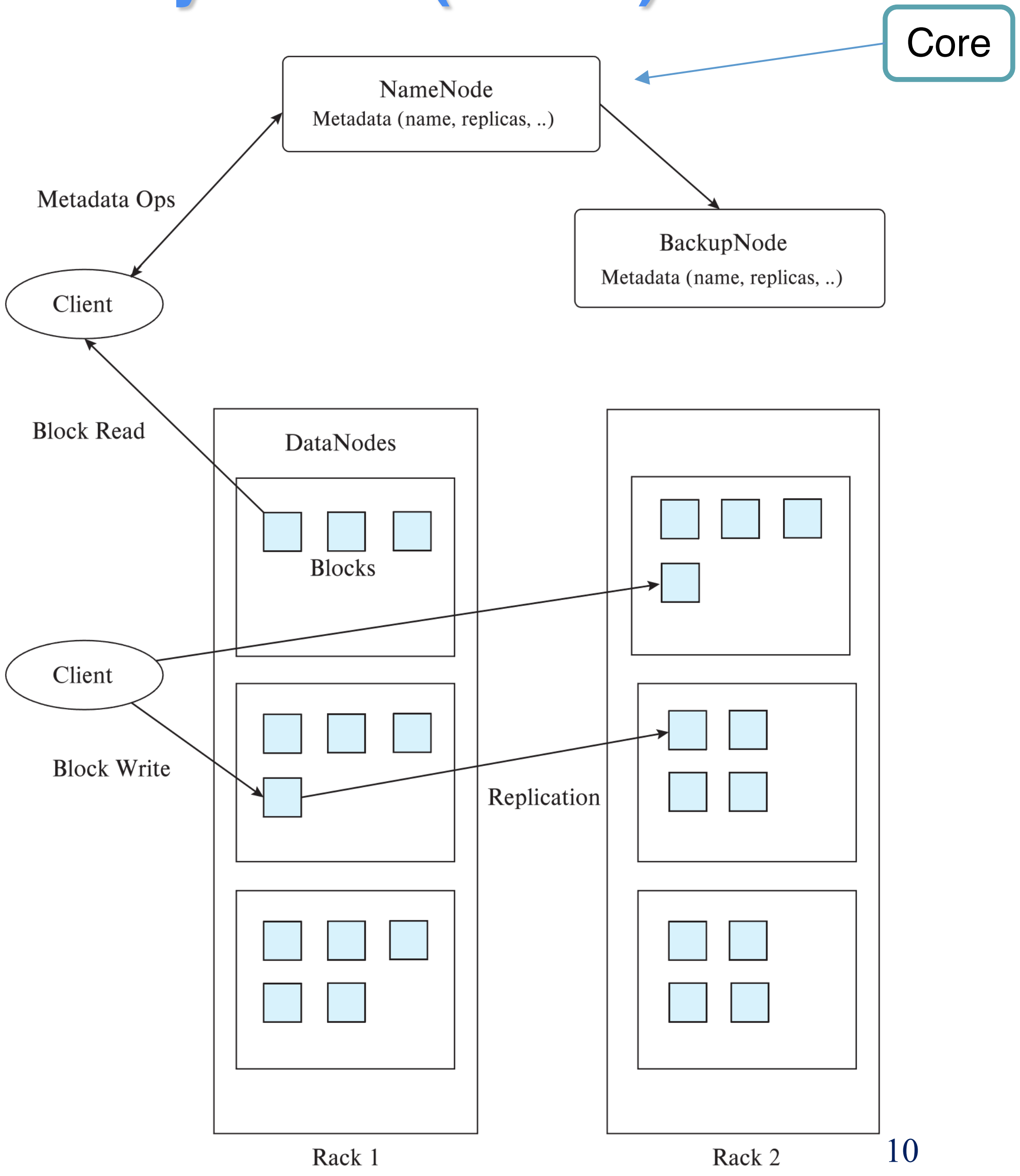
- Single Namespace
- Files are broken up into blocks
  block size = 128MB
  - Each block, replicated on multiple DataNodes
    3 times
- NameNode stores:
  - List of blocks of a file
  - List of nodes with a copy of each block
- Access:
  - API
  - As a subdirectory of the local file system
    Connected to HDFS server



Hadoop Distributed File Systems (HDFS)

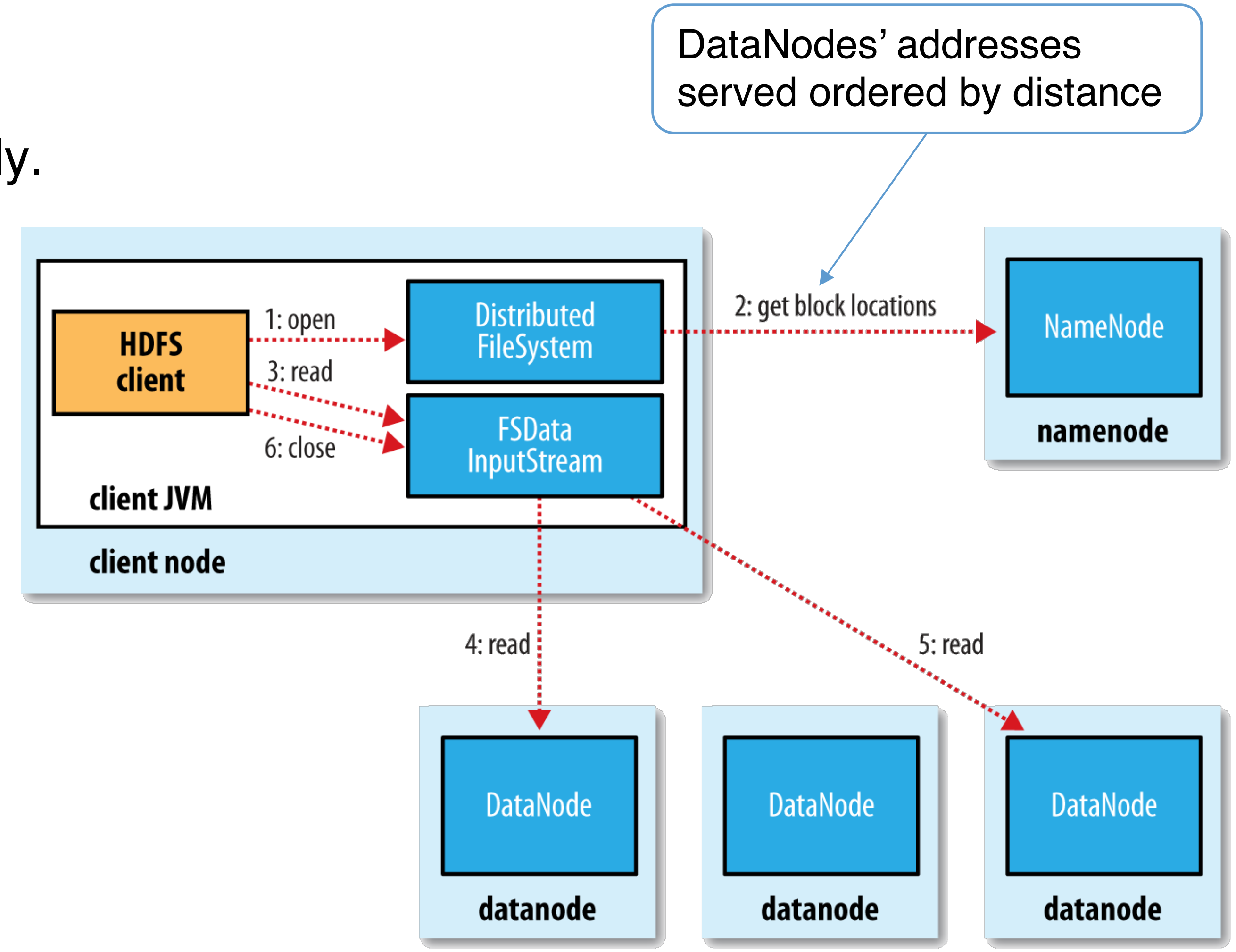
- Read from a client:
  - Retrieves from NameNode block and node ids
  - Selects a DataNode for each block and retrieve data

- Write from a client:
  - Retrieves new block ids and a set of machines for each, assigned by NameNode
  - Client sends block ids and data to the assigned DataNodes



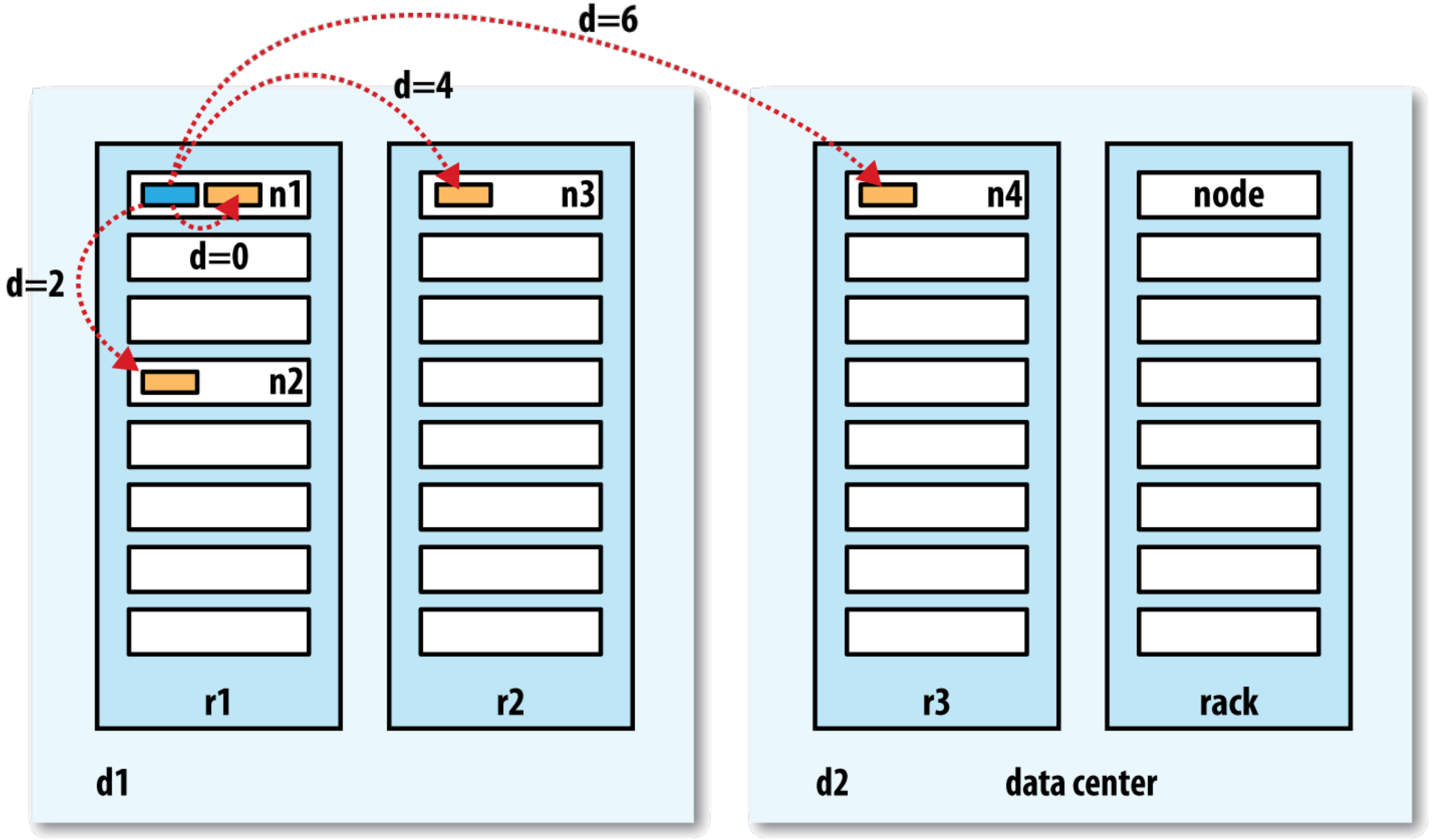
HDFS

- Reading a file from HDFS
  - Client contacts DataNodes directly.
  - This design allows HDFS to scale to many concurrent clients.
  - Data traffic is spread across all the cluster.
  - NameNode merely serves block location requests.



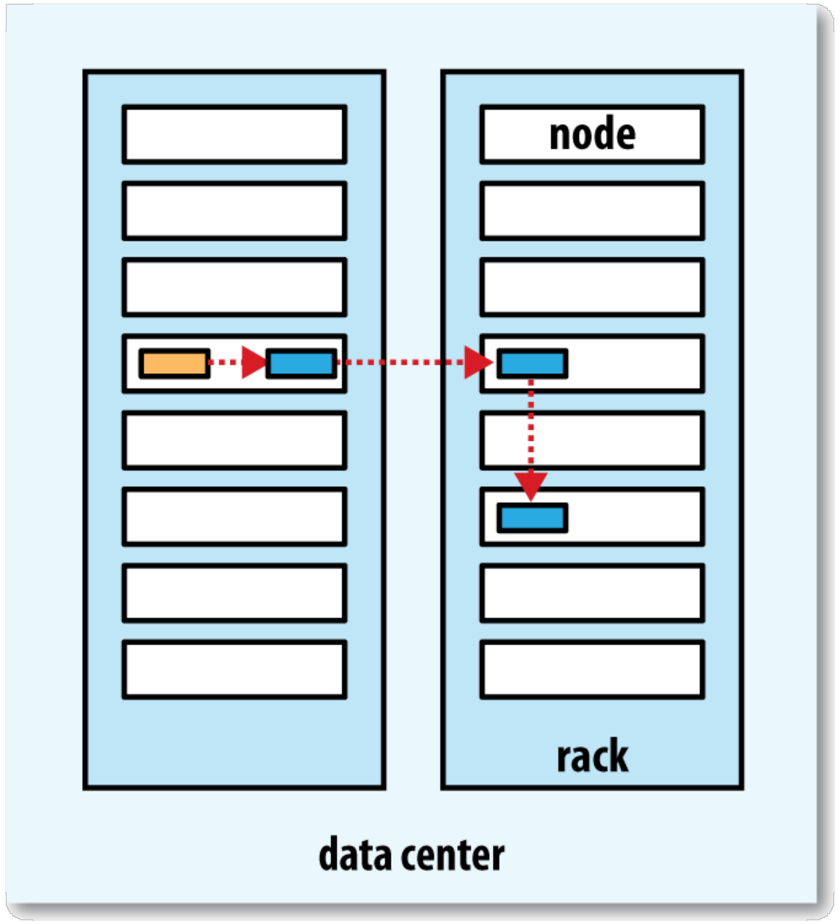
HDFS

- Distance in Hadoop between nodes
  - Replication usually within a single data center



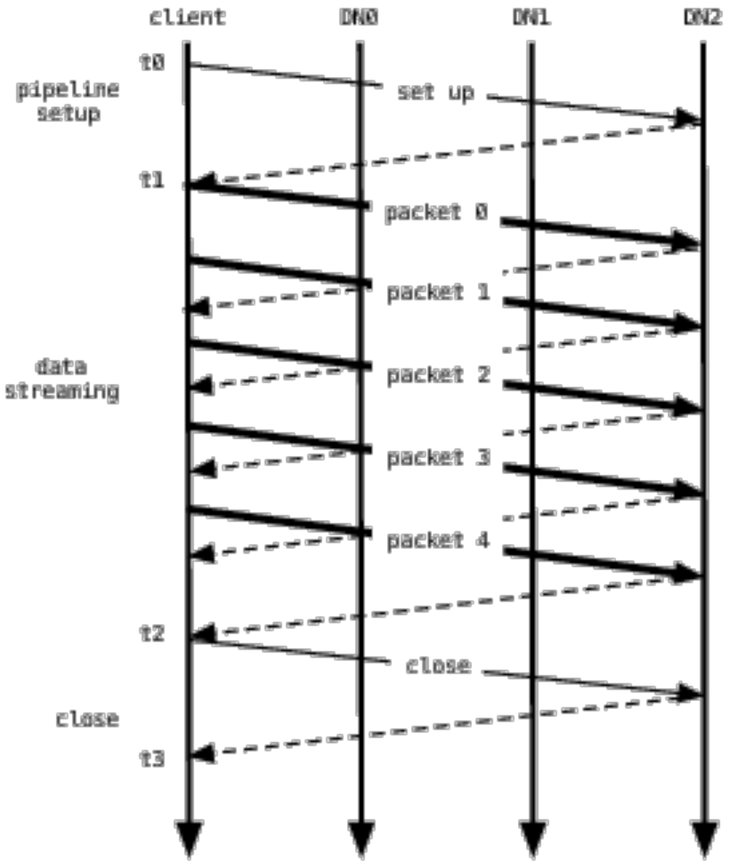
HDFS

- Hadoop's replica location policy
  1. Client's node
     - Or randomly, if it hasn't
  2. Another rack
  3. Another node in 2nd rack

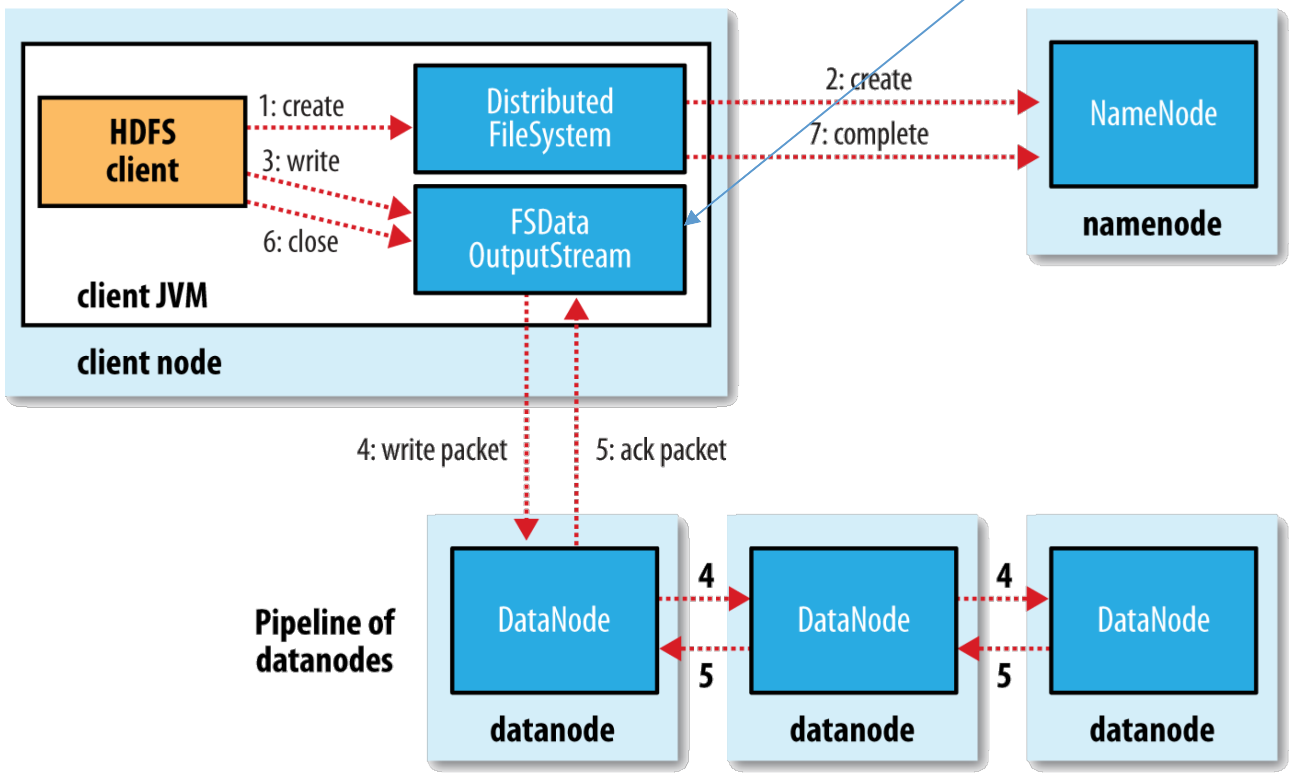


HDFS

- Writing a file in HDFS
  - Pipeline of DataNodes



DataNodes' addresses ser-ved ordered by NameNode

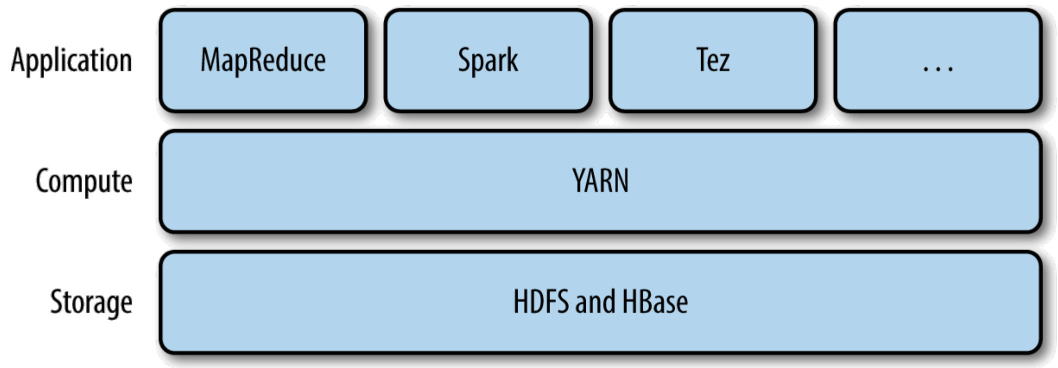


Limitations of HDFS

- Central master becomes bottleneck
  - Keep metadata in memory to avoid IO
    - Memory size limits no. files
  - Distributed master supported by Colossus file system
    - With smaller (1MB) block size
- File system directory overheads per file
  - Not appropriate for storing too many files
- Without consistency guarantees
  - File systems cache blocks locally
  - Ideal for write-once and append-only data

Hadoop

- 3 building blocks
  - HDFS: distributed file system
  - YARN: Resource (cluster) manager
  - Map-reduce: parallel computing paradigm



- Splits data and computation across (thousands of) hosts
  - A Hadoop cluster scales computation capacity, storage capacity and I/O bandwidth by simply adding nodes.

MapReduce paradigm

- Old paradigm in parallel and functional programming
  - `map()`: solve a task for each element of a large list
  - `reduce()`: aggregates the solutions of the different elements

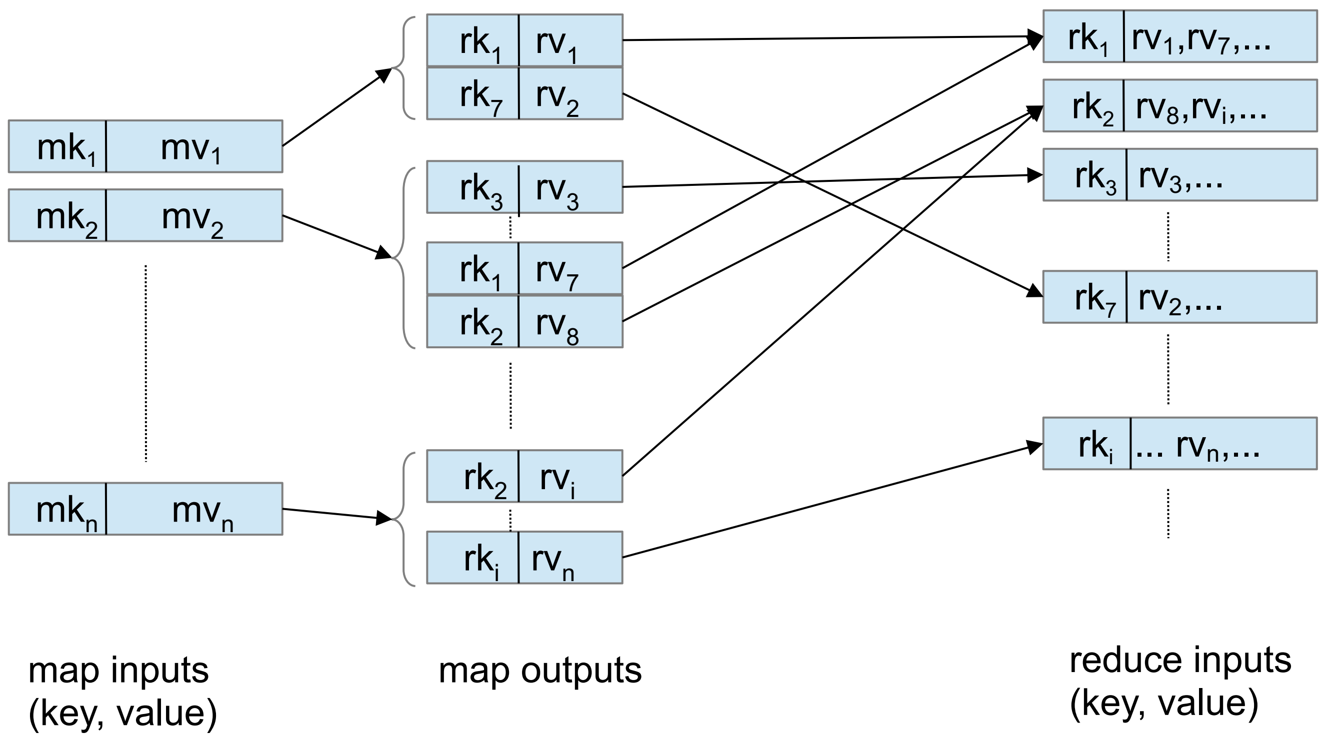
- E.g., word counts in a large collection of books
  - Solution:
    - Divide documents among workers
    - Each worker finds all words, and outputs (word, count) pairs
    - Partition (word, count) pairs across workers based on word
    - For each word at a worker, add up counts

```
map(String document):
   for word in document:
     emit(word, 1);
```

```
reduce(String key, List value_list):
   int count = 0;
   for value in value_list:
     count += value;
   return(key, count);
```

MapReduce paradigm

- Flow of **keys** and **values** in a map-reduce task



Programming Hadoop

- **Workflow**:
  - Put data in HDFS, run code, bring results back from HDFS

```
% hdfs dfs -mkdir /wordcount

% hdfs dfs -copyFromLocal /path/to/your/data /wordcount/input

% hadoop jar ${HADOOP_PATH}/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.3.0.jar \
       wordcount /wordcount/input /wordcount/output

% hdfs dfs -cat /wordcount/output/part-r-00000

% hdfs dfs -copyToLocal /wordcount/output/part-r-00000 .
```

Programming Hadoop

- `Mapper` and `Reducer` interfaces take…
  - input key, input value, output key and output value

```
public static class myMap extends Mapper<LongWritable, Text,
                                        Text, IntWritable> {
   private final static IntWritable one = new IntWritable(1);
   private Text word = new Text();
   public void map(LongWritable key, Text value, Context context)
                       throws IOException, InterruptedException {
      String line = value.toString();
      StringTokenizer tokenizer = new StringTokenizer(line);
      while (tokenizer.hasMoreTokens()) {
          word.set(tokenizer.nextToken());
          context.write(word, one);
      }
   }
}
```

Programming Hadoop

- `Mapper` and `Reducer` interfaces take…
  - input key, input value, output key and output value

```
public static class myReduce extends Reducer<Text, IntWritable,
                                             Text, IntWritable> {
    public void reduce(Text key, Iterable<IntWritable> values,
                        Context context)
                        throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }
        context.write(key, new IntWritable(sum));
    }
}
```

Programming Hadoop

- General class

```
public class WordCount {
    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = new Job(conf, "wordcount");
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        job.setMapperClass(myMap.class);
        job.setReducerClass(myReduce.class);

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        System.exit(job.waitForCompletion(true)? 0:1);
    }
}
```

MapReduce for a Projection operator

- Explain the behavior of the mapper and the reducer

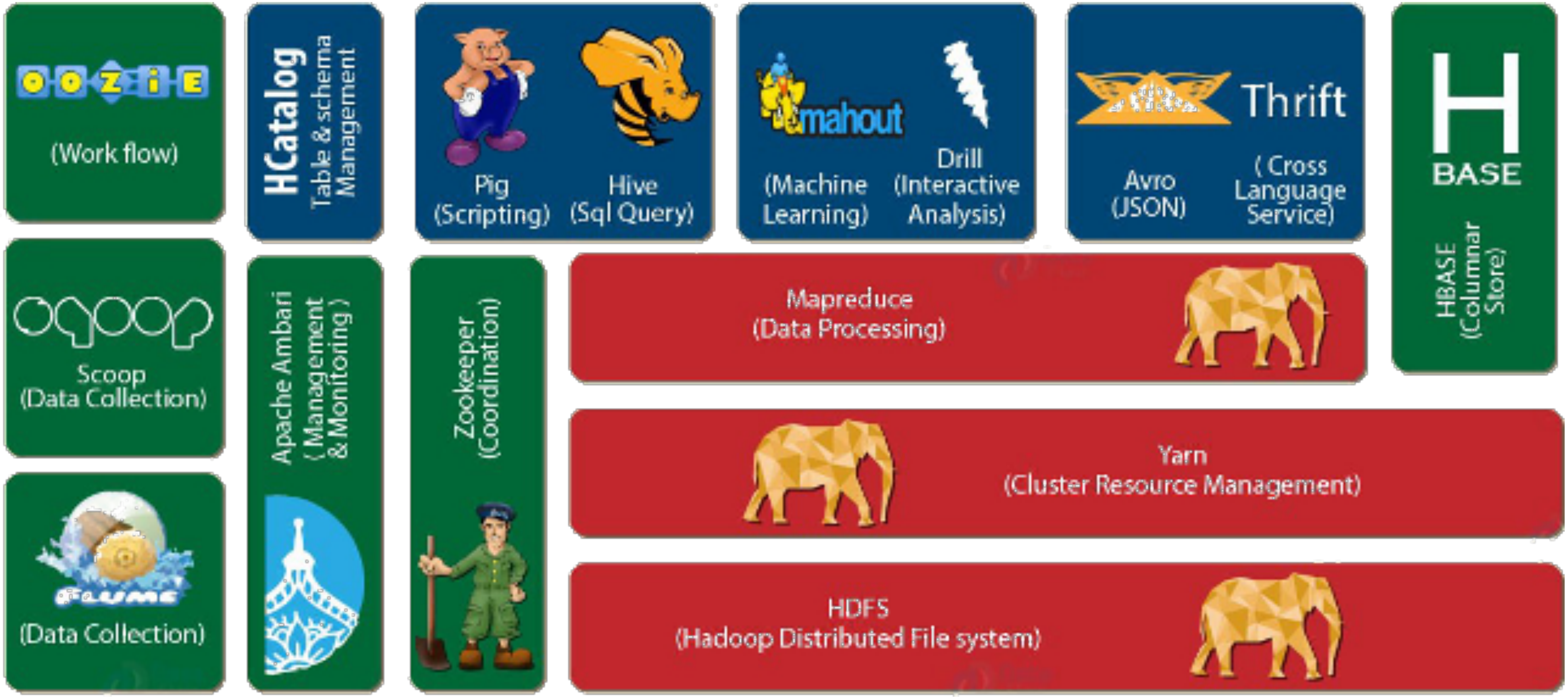
MapReduce for a Selection operator

- Explain the behavior of the mapper and the reducer

MapReduce vs. Relational Queries

- MapReduce widely used for parallel processing
  - Allows procedural code in map and reduce functions
  - Allows data of any type
- Many computations…
  - of MapReduce cannot be expressed in SQL
  - are much easier to express in SQL
  - of relational operations can be expressed using MapReduce
    - select, project, join, aggregation, etc.
- SQL queries have been translated into MapReduce
  - Apache Hive SQL, Apache Pig Latin, Microsoft SCOPE

Hadoop Ecosystem



Hadoop