Multicore Computing
Lecture24 – Big Data Platforms 1

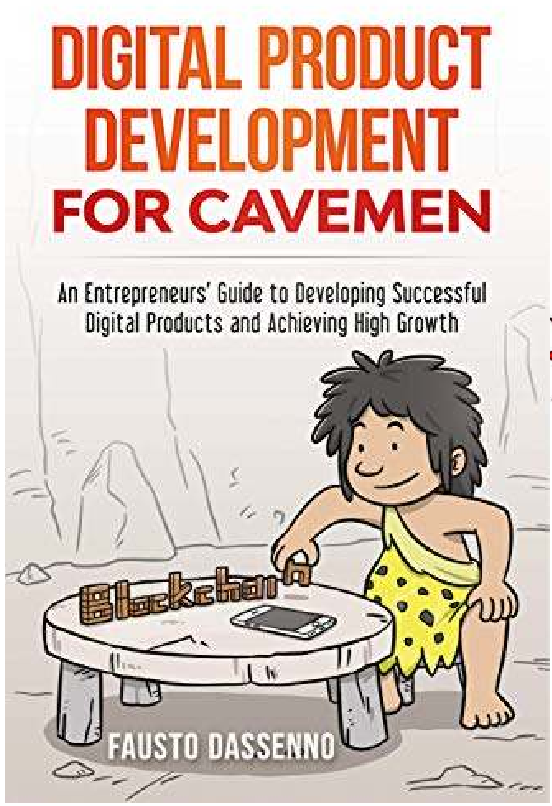
SUNG KYUN KWAN UNIVERSITY

남 범 석

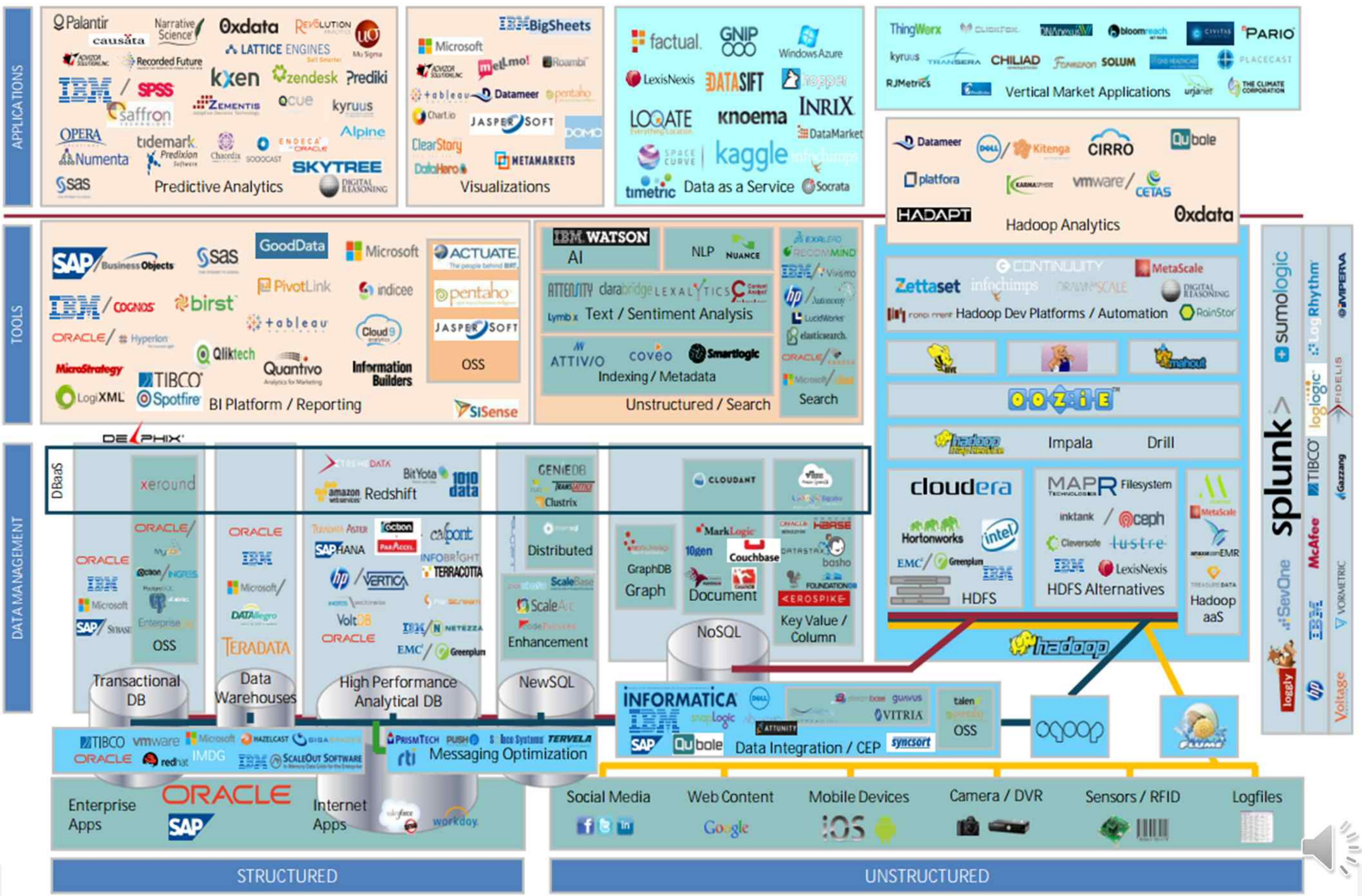
bnam@skku.edu

How to build a scalable system?

- Computer Science is a Science of Abstraction
  - creating the right model for a problem and devising the appropriate mechanizable techniques to solve it. – Alfred Aho



Which solution to choose?



Big Data

- Transaction processing systems that need very high scalability
  - Many applications willing to sacrifice ACID properties and other database features, if they can get very high scalability
- Query processing systems that
  - Need very high scalability, and
  - Need to support non-relation data

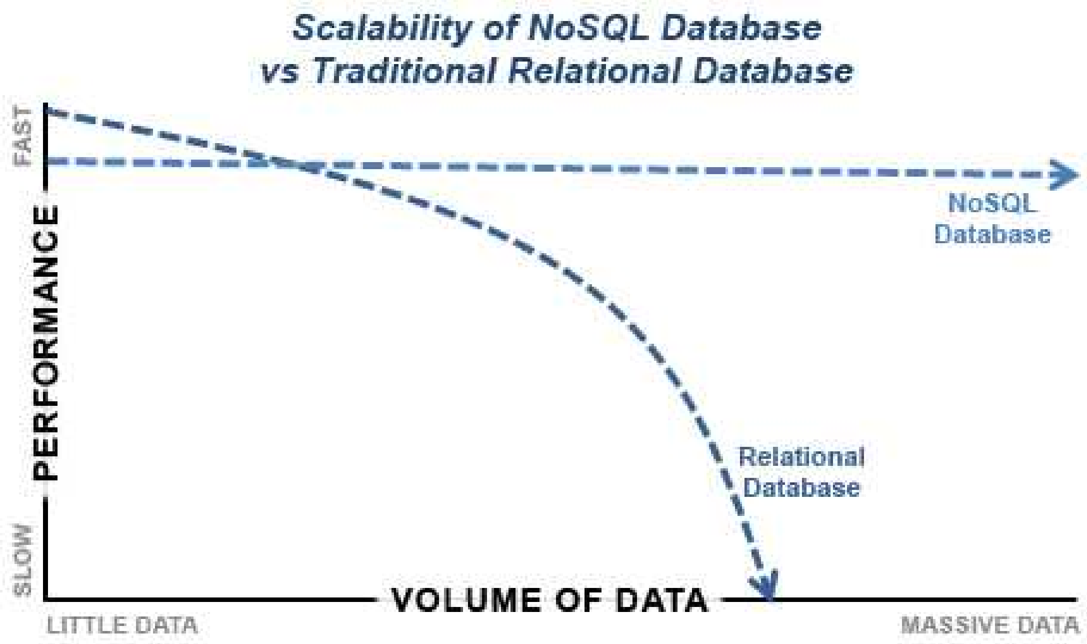
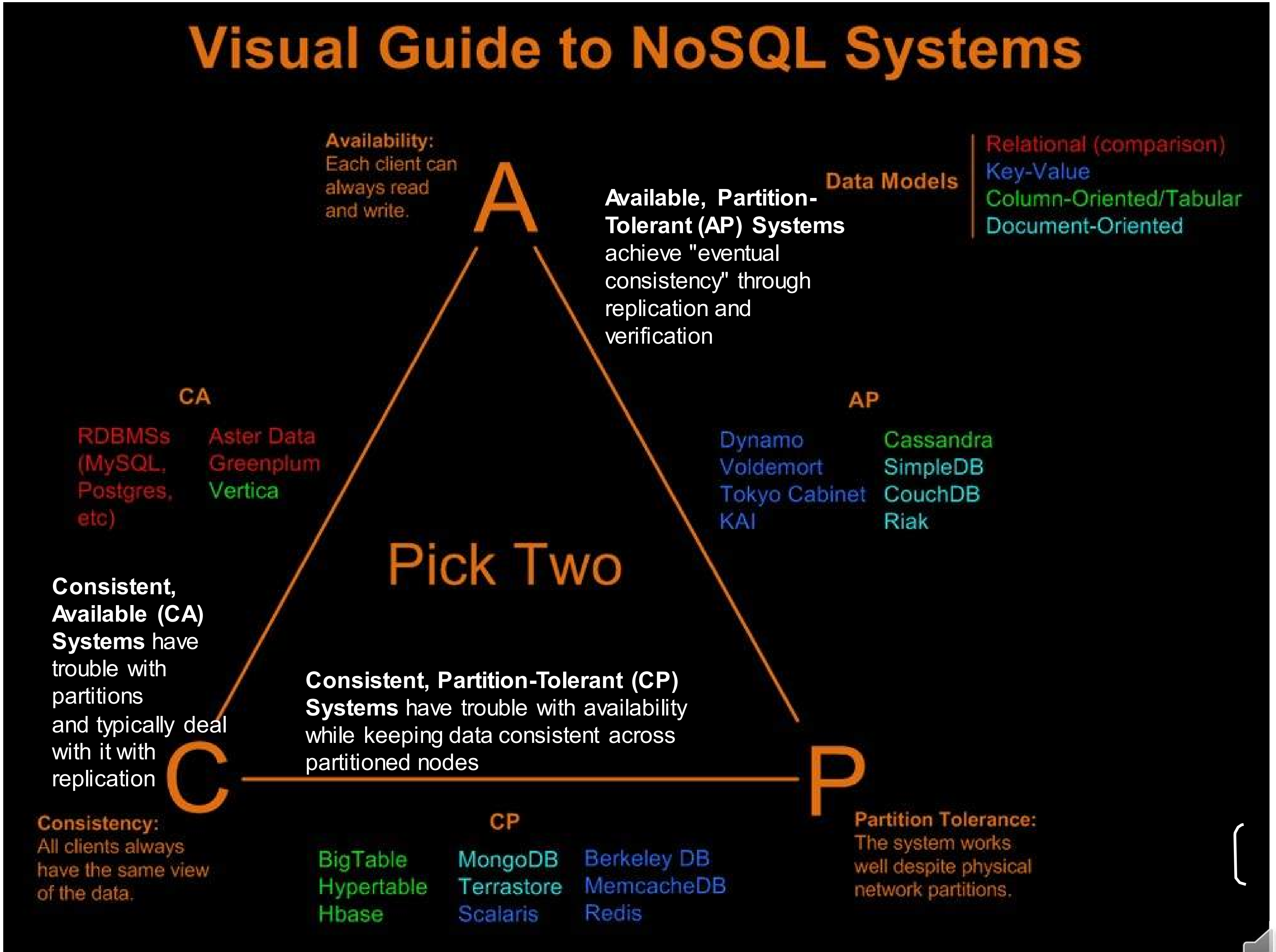


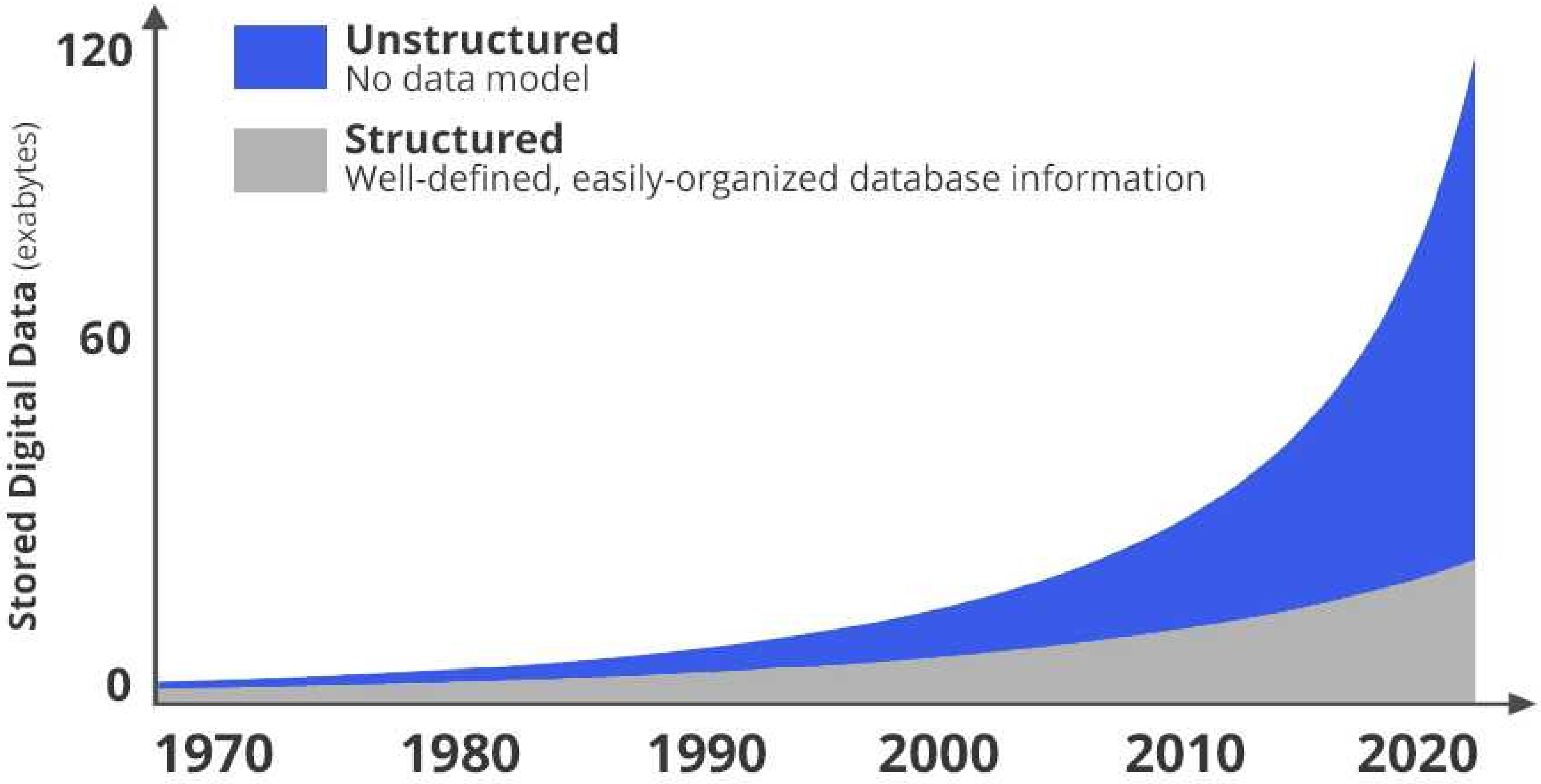
Image Credit: DataJobs.com



http://blog.nahurst.com/visual-guide-to-nosql-systems

Explosion of Unstructured/Semi-Structured Data

- A poor fit for the legacy RDBMS



Graph Source: IDC

Semi-Structured Data

- JSON: Textual representation widely used for data exchange
  - Example 1:

```
{
        "ID": ”1111",
        "name": {
                "firstname: "Albert",
                "lastname: "Einstein"
        },
        "deptname": "Physics",
        "children": [
                {"firstname": "Hans", "lastname": "Einstein" },
                {"firstname": "Eduard", "lastname": "Einstein" }
        ]
}
```

  - Example 2:

```
{
        "ID": "22222",
        "name": {
                ”Beomseok Nam"
        },
        "deptname": ”Computer Science",
        “e-mail”: “bnam@skku.edu”
}
```

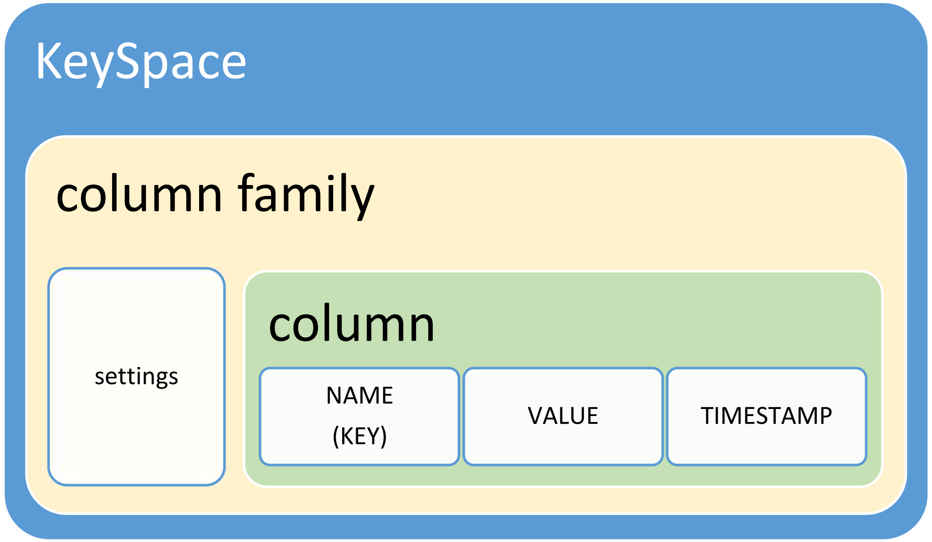
Data Model for Semi-Structured Data

- Structured Table for Semi-Structured Data?
  - Does not allow schema changes
  - Too many columns
  - Sparse tables

Structured Table (Schema)

ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	[illegible]
10101	Srinivasan	Comp. Sci.	[illegible]
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

Key-Value Stores (Schema-less)



Key-Value Storage (KVStore)

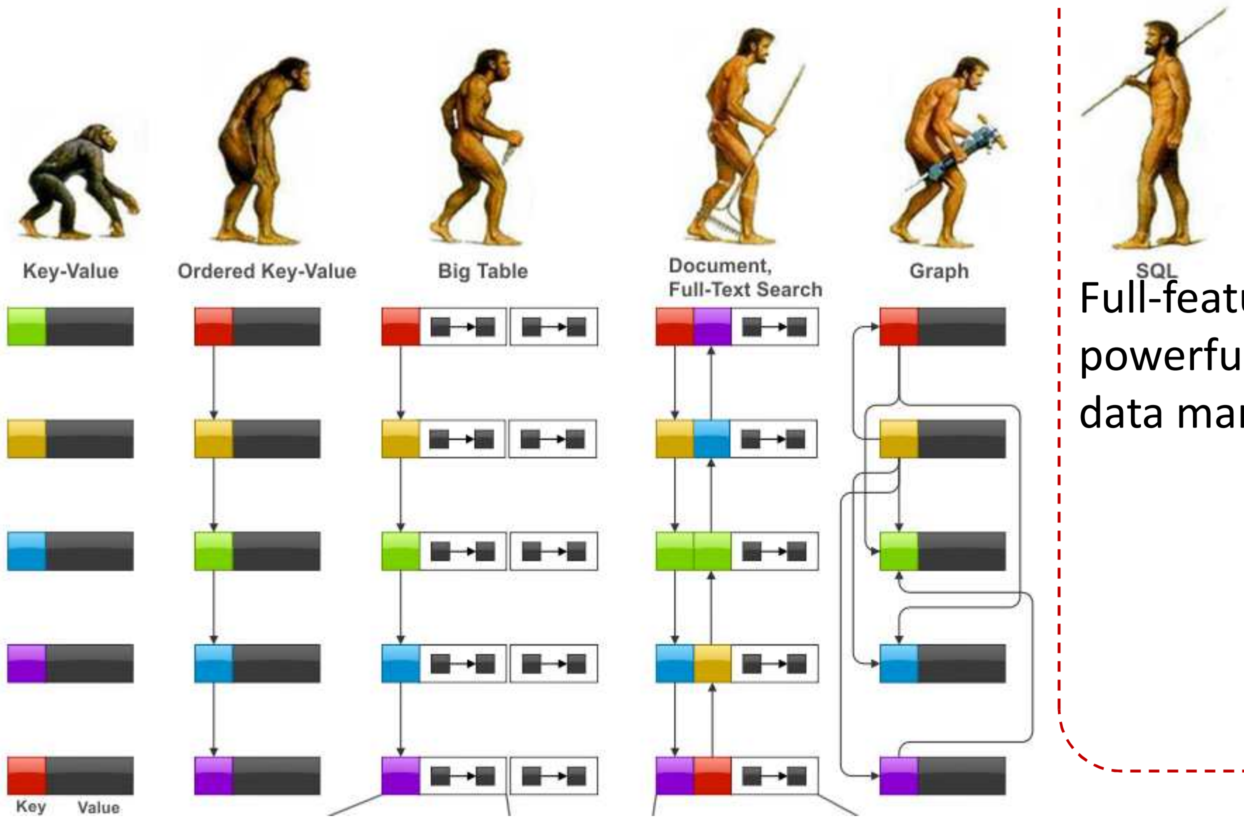
- KV-Stores seem very simple indeed
  - They are nothing but indexing structures
  - A simpler and more scalable "database"

- Interface
  - `put(key, value);` // insert/write "value" associated with "key"
  - `value = get(key);` // get/read data associated with "key"

- Examples
  - Google BigTable & internal codes:
    - Key: hangoutID
    - Value: Hangout conversations
  - Facebook, Twitter:
    - Key: UserID
    - Value: user profile (e.g., posting history, photos, friends, …)

When to use Key-Value Stores over RDBMS?

- If data model is not complex nor hierarchical
- If workload is write-intensive
- If strong consistency is not required



Full-featured
powerful
data management

Image Source: From RDBMS to Key-Value Store: Data Modeling Techniques- Wishmitha S. Mendis

Hadoop Distributed File System

Distributed File Systems

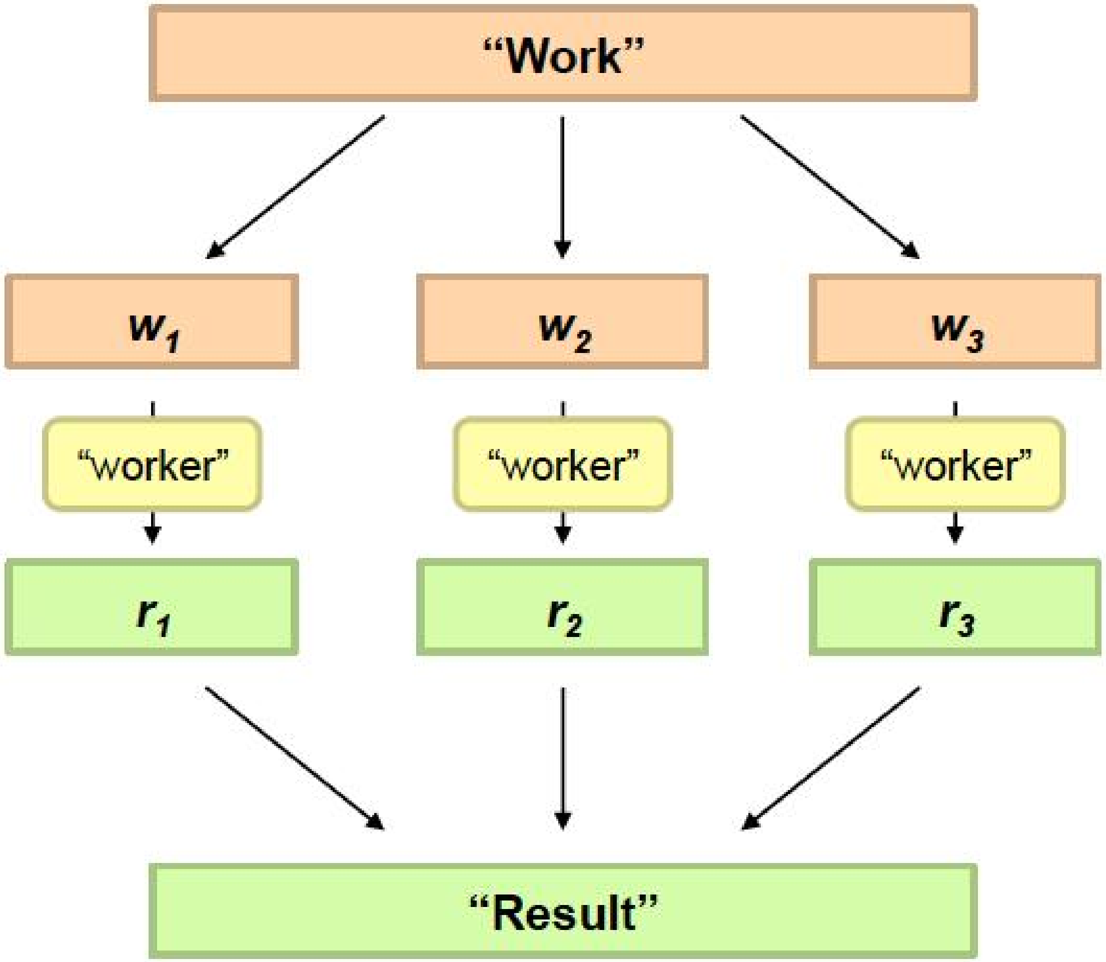
- A distributed file system stores data across a large collection of machines, but provides single file-system view
- Highly scalable distributed file system for large data-intensive applications.
  - E.g., 10K nodes, 100 million files, 10 PB
- Provides redundant storage of massive amounts of data on cheap and unreliable computers
  - Files are replicated to handle hardware failure
  - Detect failures and recovers from them
- Examples:
  - Google File System (GFS)
  - Hadoop File System (HDFS)

How to process a large data?

- Q: How do you sort a trillion rows of data of integers (a few terabytes) in a file with only 16 GB of main memory.?

- Hint: Divide and Conquer

Divide and Conquer



Divide Work

Combine Results

Distributed processing is non-trivial

- How to assign tasks to different workers in an efficient way?
- What happens if tasks fail?
- How do workers exchange results?
- How to synchronize distributed tasks allocated to different workers?
- Data Volumes are massive
- Reliability of Storing PBs of data is challenging
- All kinds of failures: Disk/Hardware/Network Failures
- Probability of failures simply increase with the number of machines …
- …

Hadoop offers

- Redundant, Fault-tolerant data storage
- Parallel computation framework
- Job coordination

Hadoop offers

- Redundant, Fault-tolerant data storage
- Parallel computation framework
- Job coordination

Programmers

No longer need to worry about

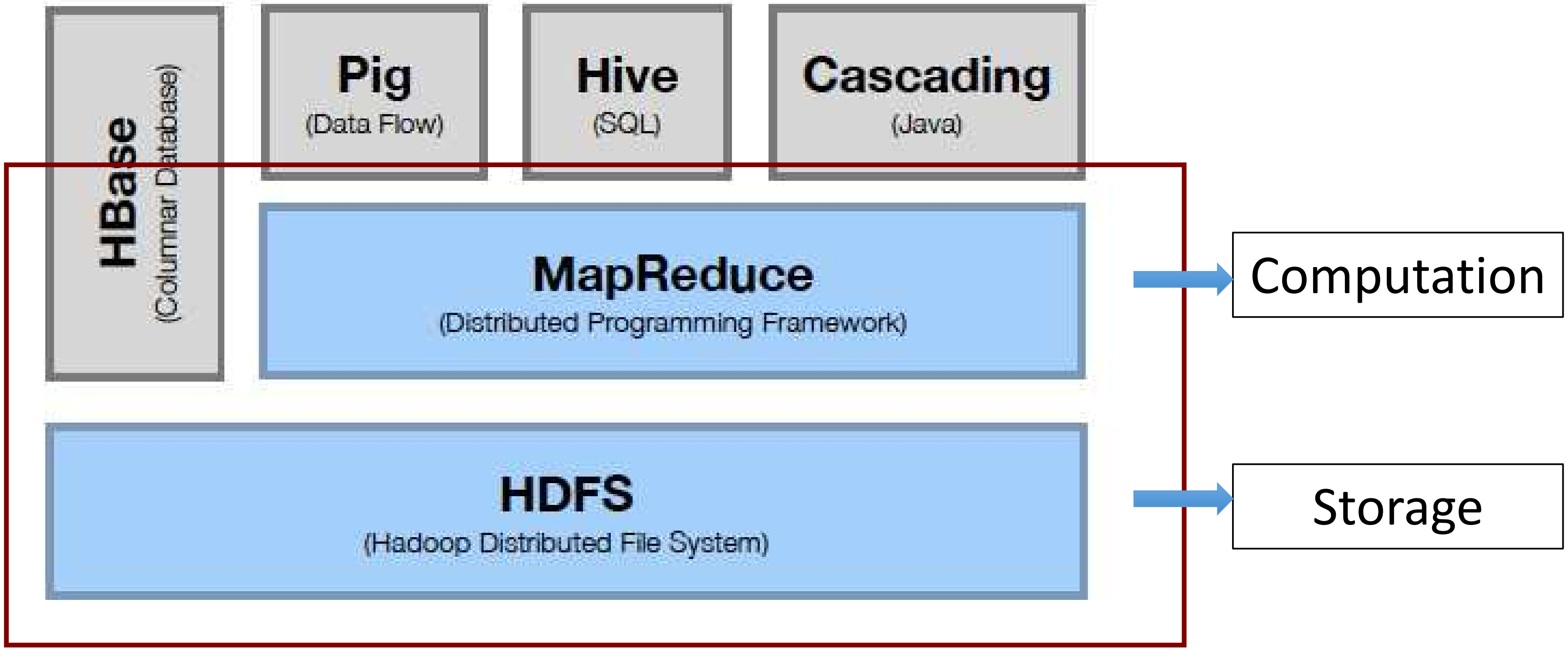
Q: Where file is located?

Q: How to handle failures & data lost?

Q: How to divide computation?

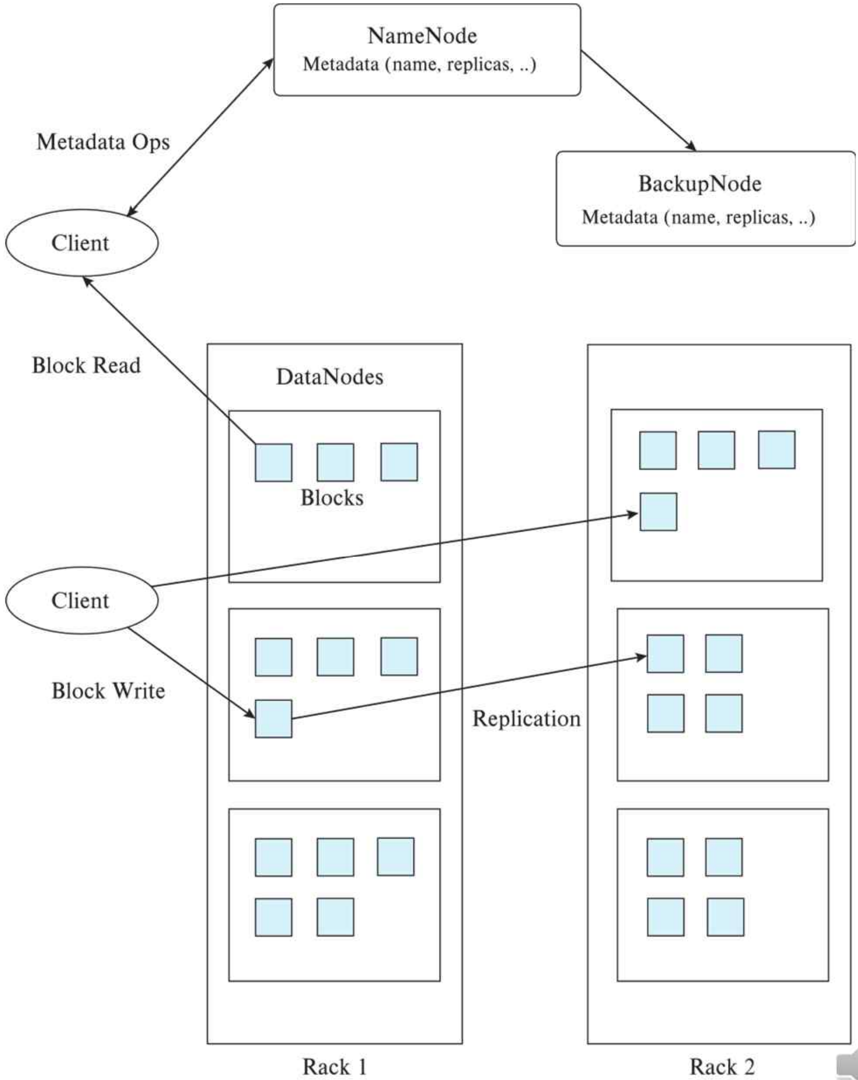
Q: How to program for scaling?

Hadoop Stack



Hadoop Distributed File System Architecture

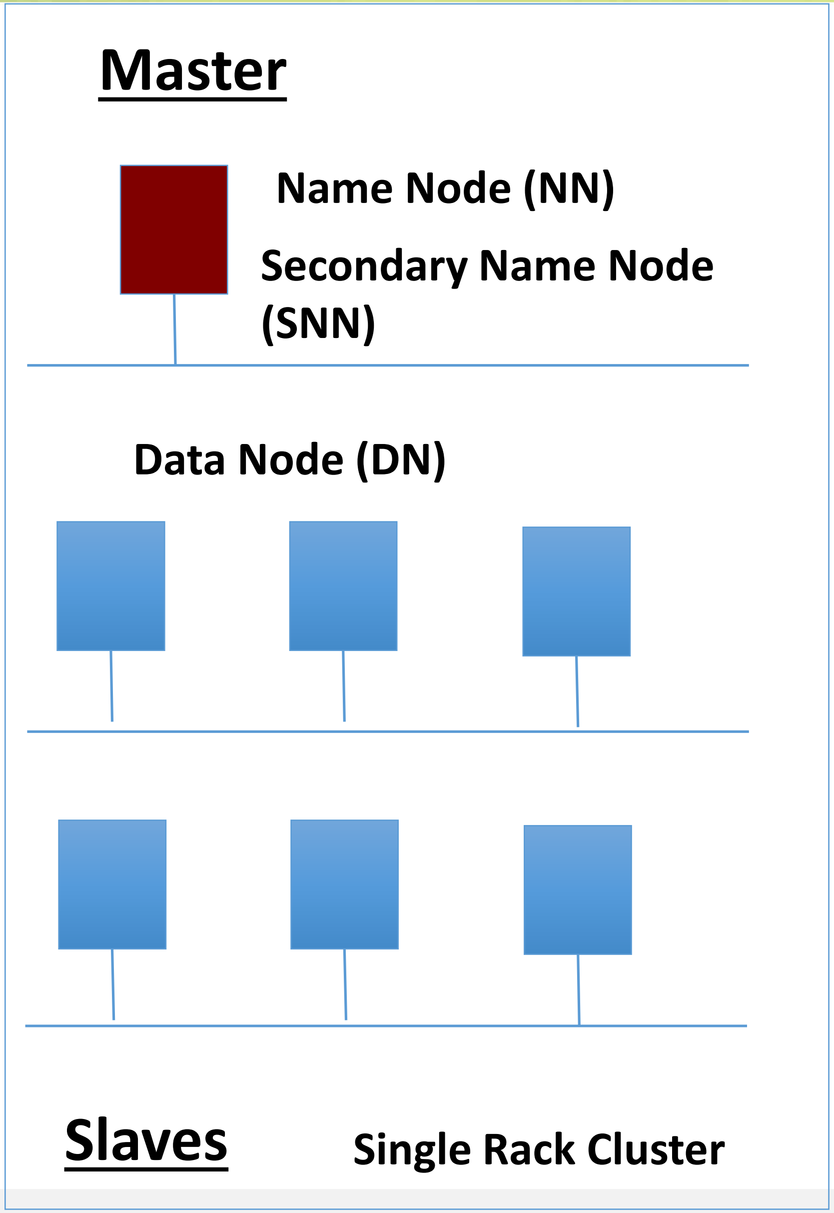
- Single Namespace for entire cluster
- Files are broken up into blocks
  - Typically 128 MB block size
  - Each block replicated on multiple DataNodes
- Client
  - Finds location of blocks from NameNode
  - Accesses data directly from DataNode



Hadoop Distributed File System (HDFS)

- **NameNode**
  - Maps a filename to list of Block IDs
  - Maps each Block ID to DataNodes containing a replica of the block
- **DataNode**: Maps a Block ID to a physical location on disk
- Data Coherency
  - Write-once-read-many access model
  - Client can only append to existing files
- Distributed file systems good for millions of large files
  - But have very high overheads and poor performance with billions of smaller tuples

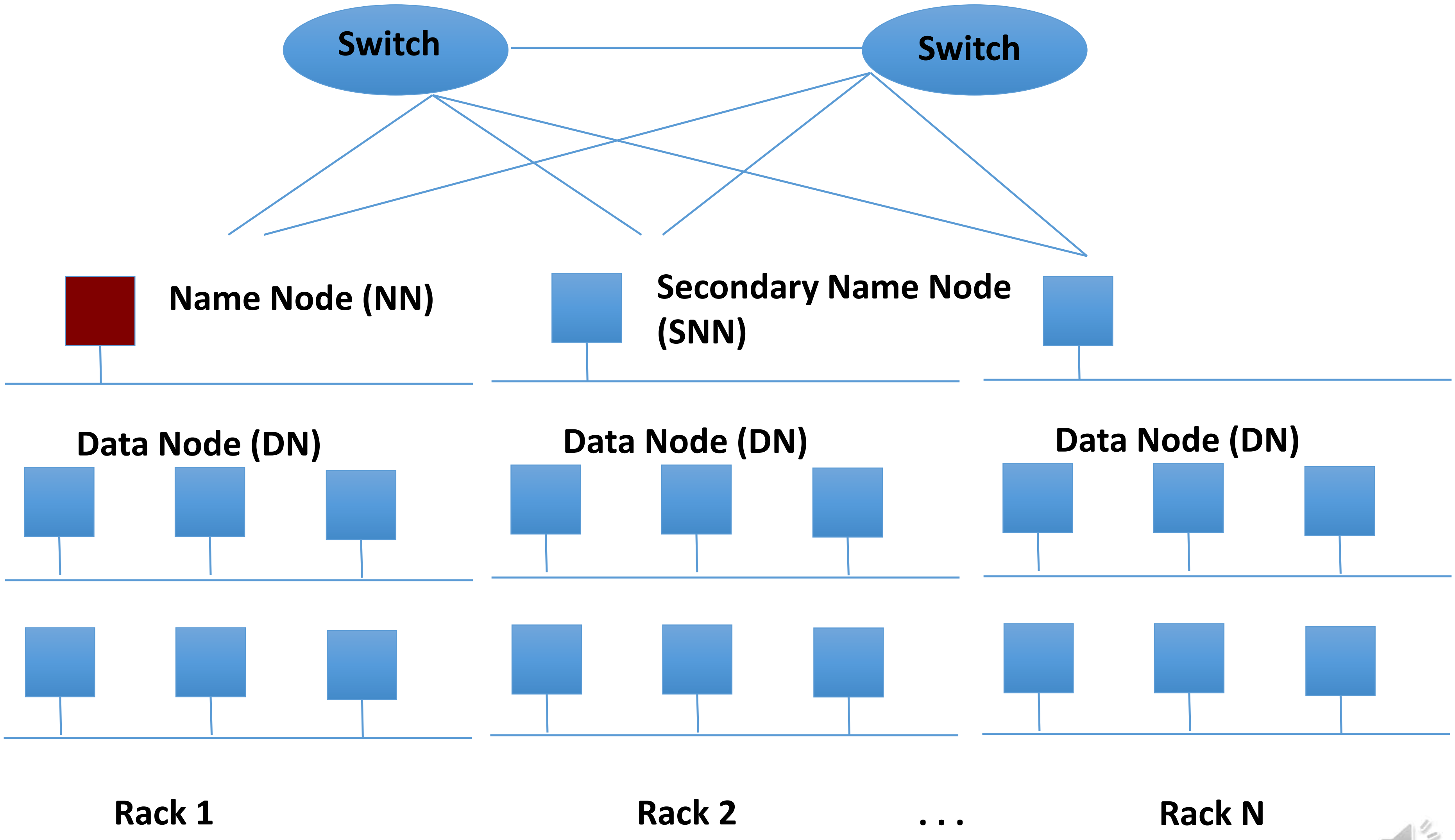
HDFS Architecture: Master-Slave



- Name Node: Controller
  - File System Name Space Management
  - Block Mappings
- Data Node: Work Horses
  - Block Operations
  - Replication
- Secondary Name Node:
  - Checkpoint node

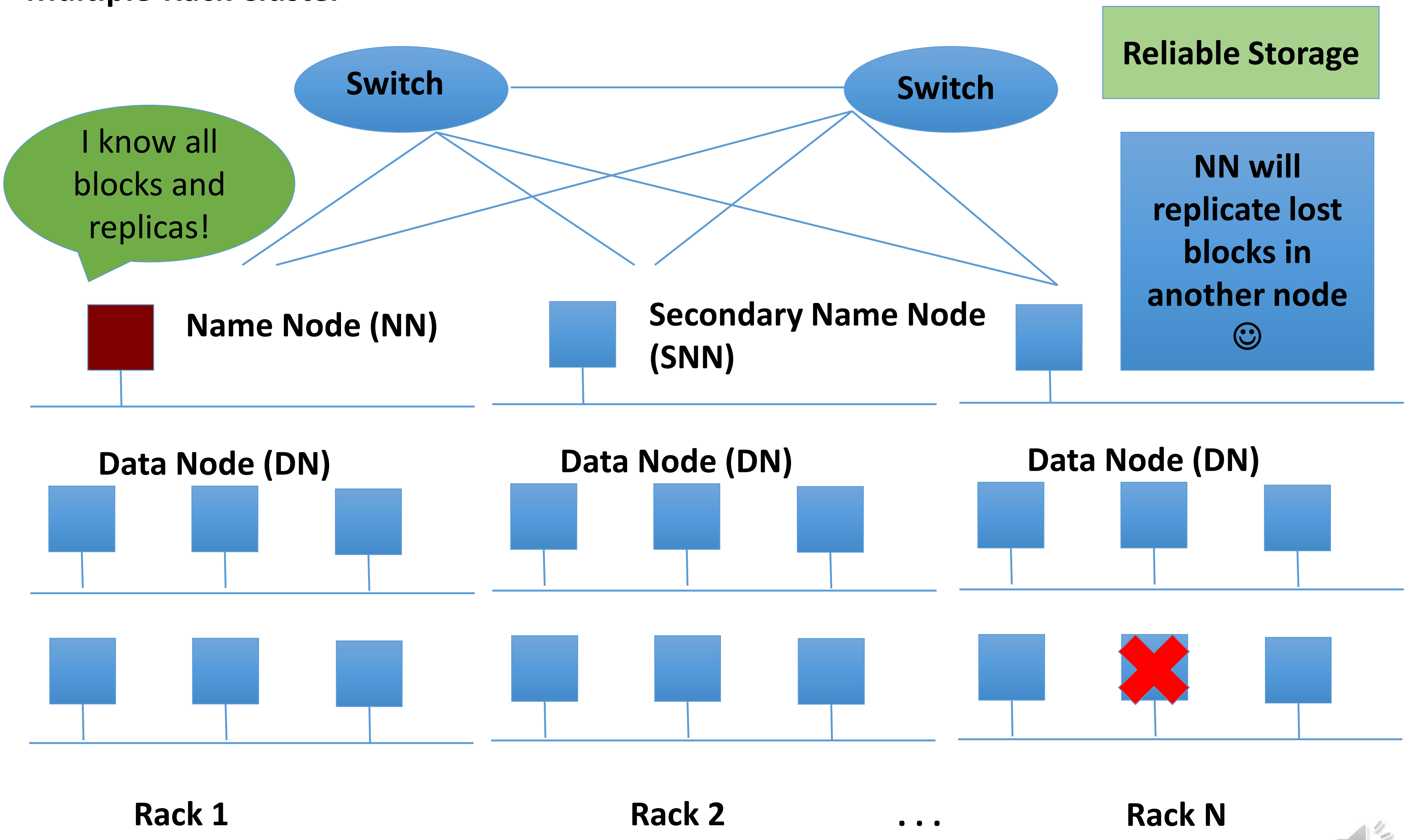
HDFS Architecture: Master-Slave

Multiple-Rack Cluster



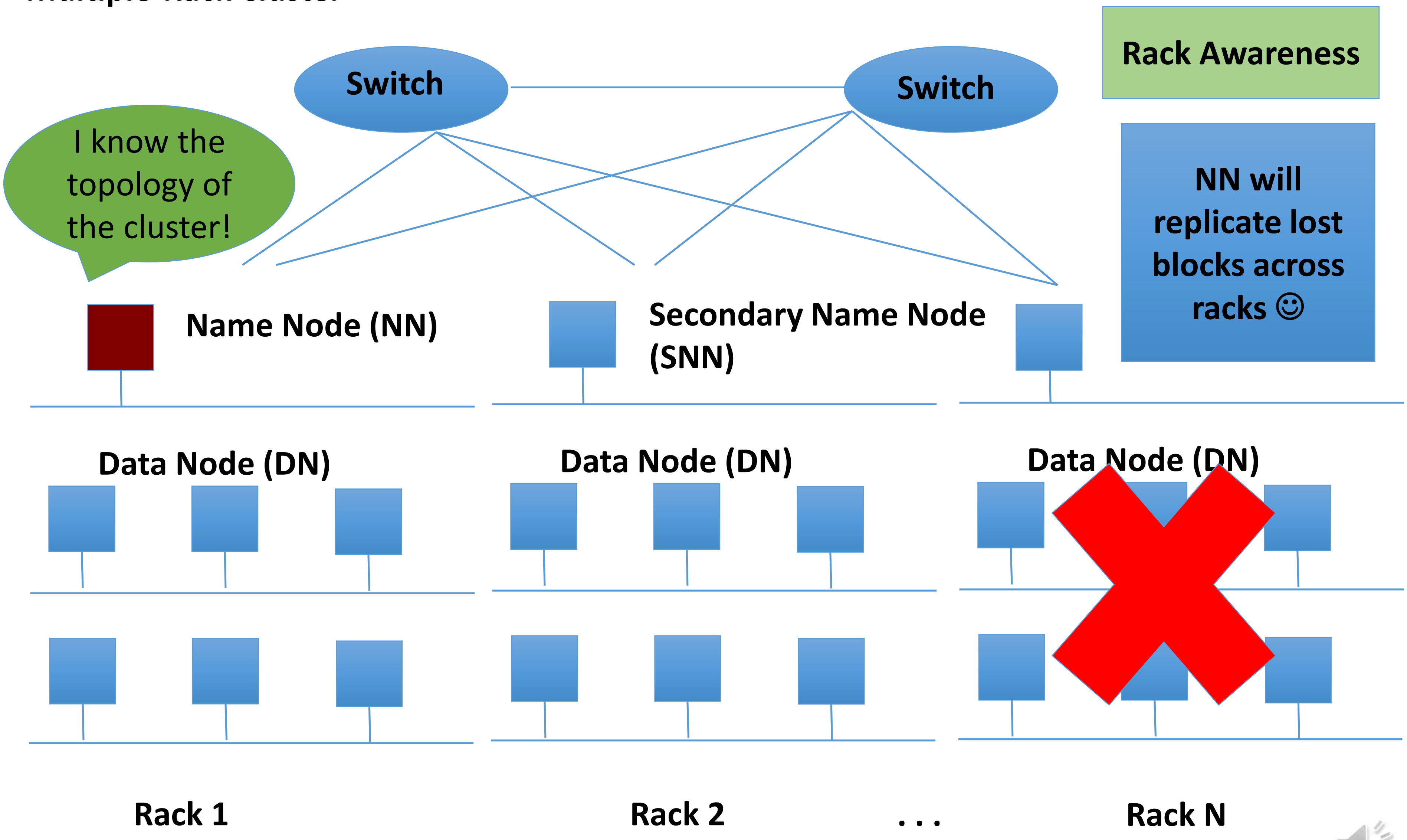
HDFS Architecture: Master-Slave

Multiple-Rack Cluster



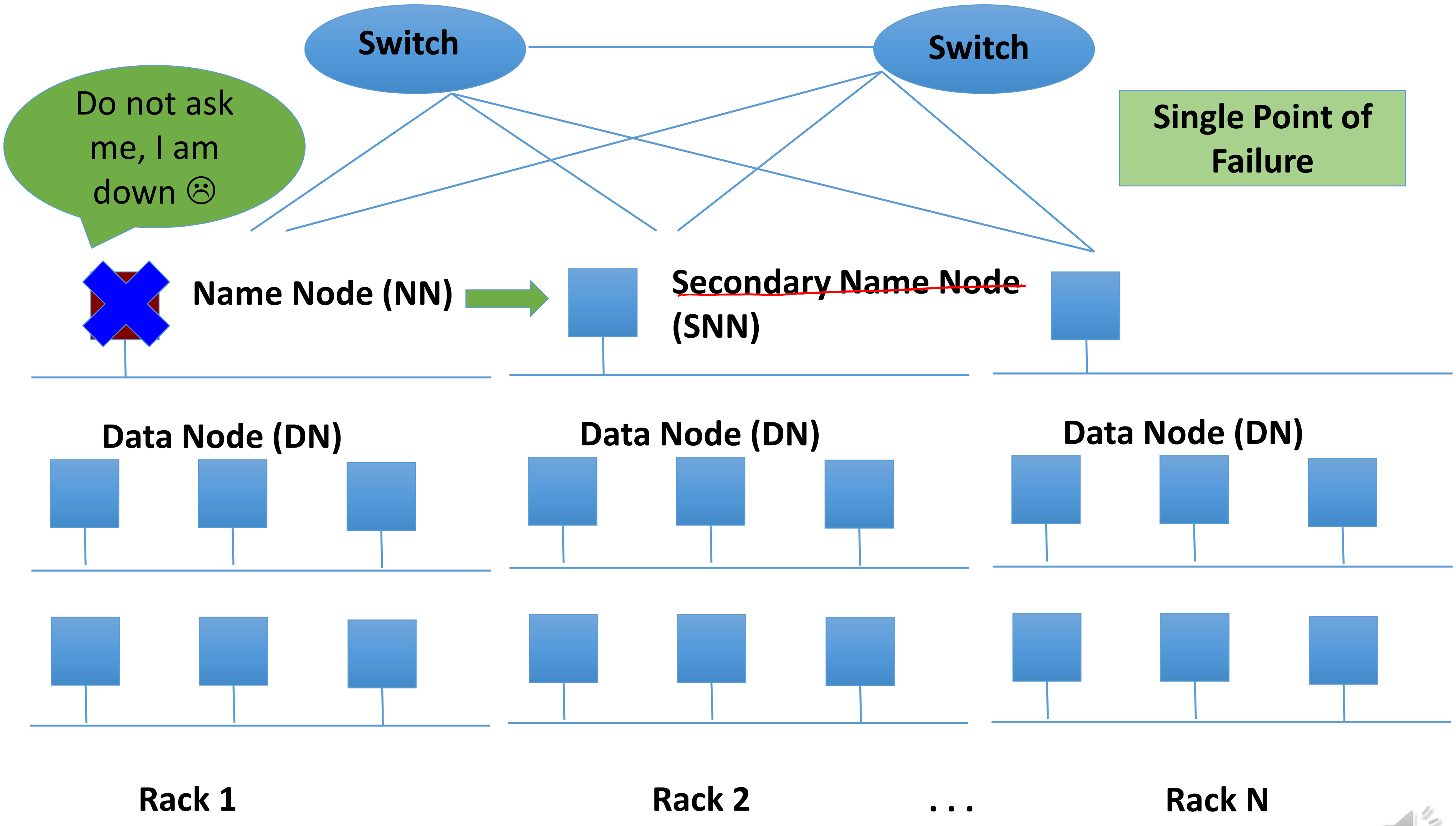
HDFS Architecture: Master-Slave

Multiple-Rack Cluster



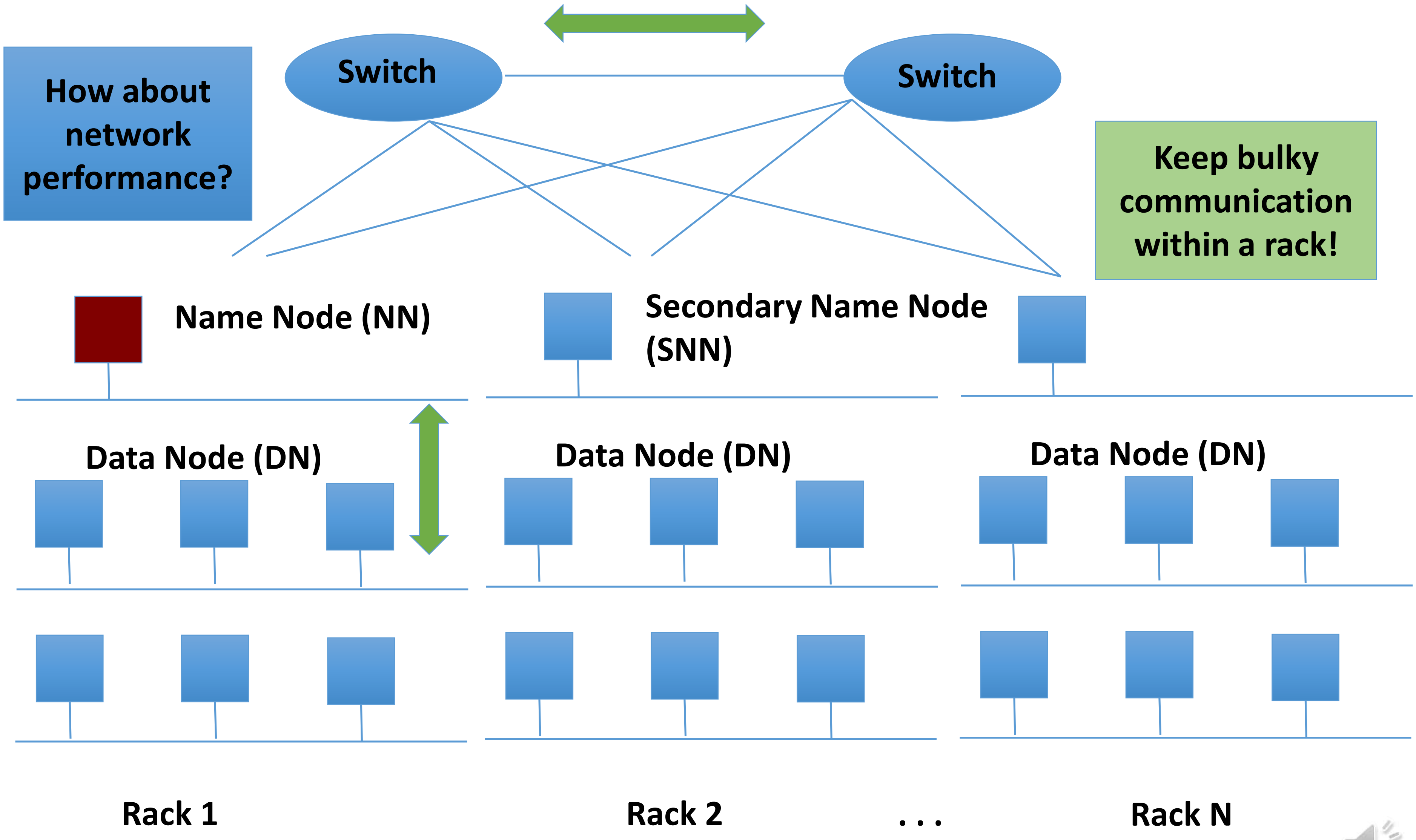
HDFS Architecture: Master-Slave

Multiple-Rack Cluster



HDFS Architecture: Master-Slave

Multiple-Rack Cluster



HDFS Inside: Name Node

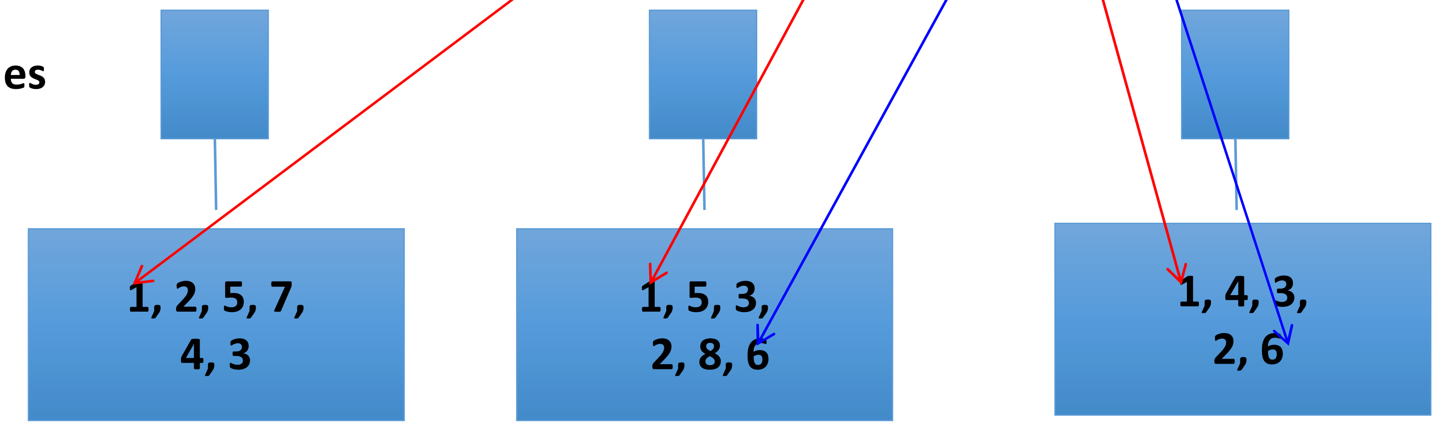
Name Node

Snapshot of FS

Edit log: record changes to FS

Filename	Replication factor	Block ID
File 1	3	[1, 2, 3]
File 2	2	[4, 5, 6]
File 3	1	[7,8]

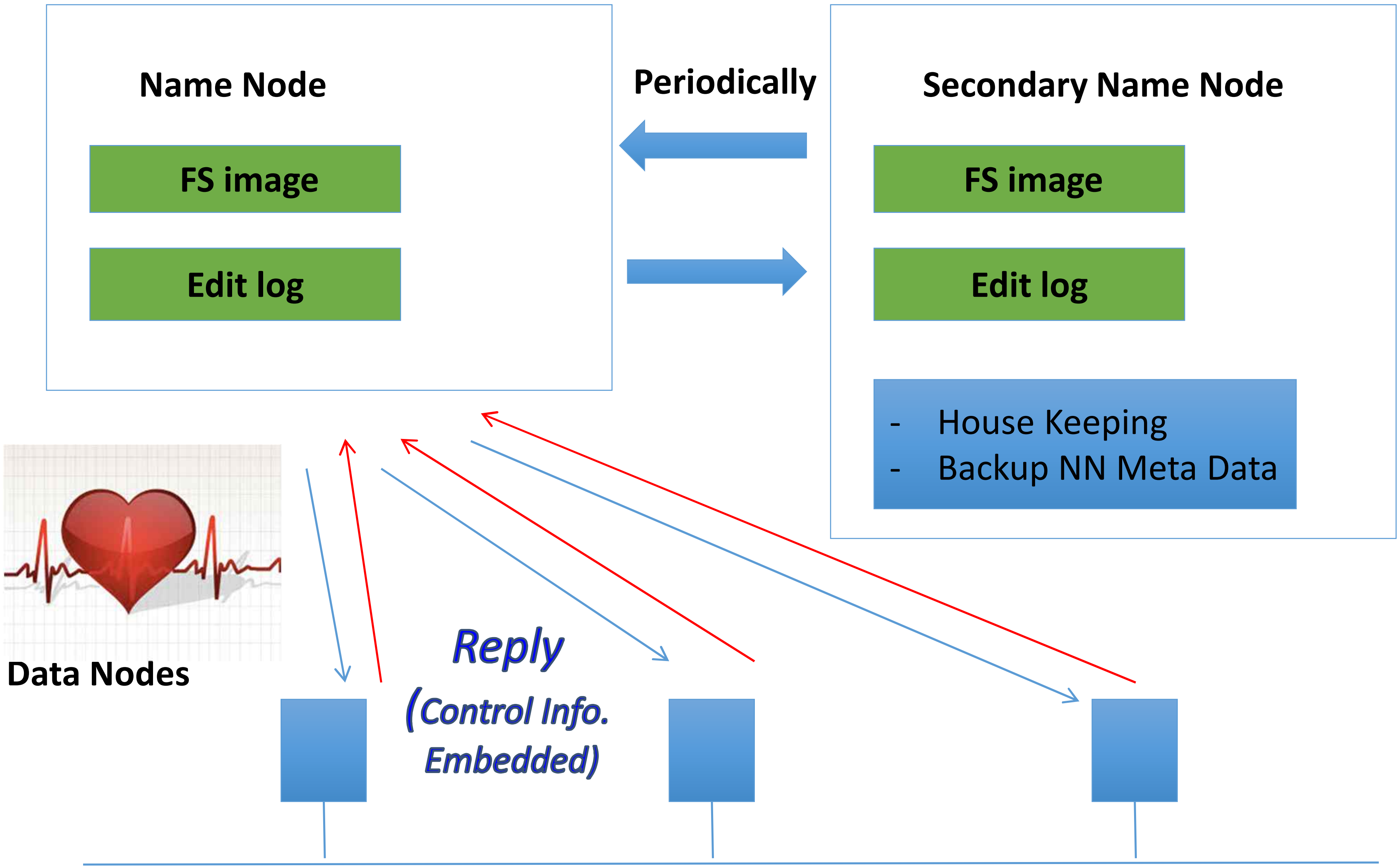
Data Nodes



HDFS Inside: Blocks

- Q: Why do we need the abstraction "Blocks" in addition to "Files"?
- Reasons:
  - File can be larger than a single disk
  - Block is of fixed size, easy to manage and manipulate
  - Easy to replicate and do more fine grained load balancing

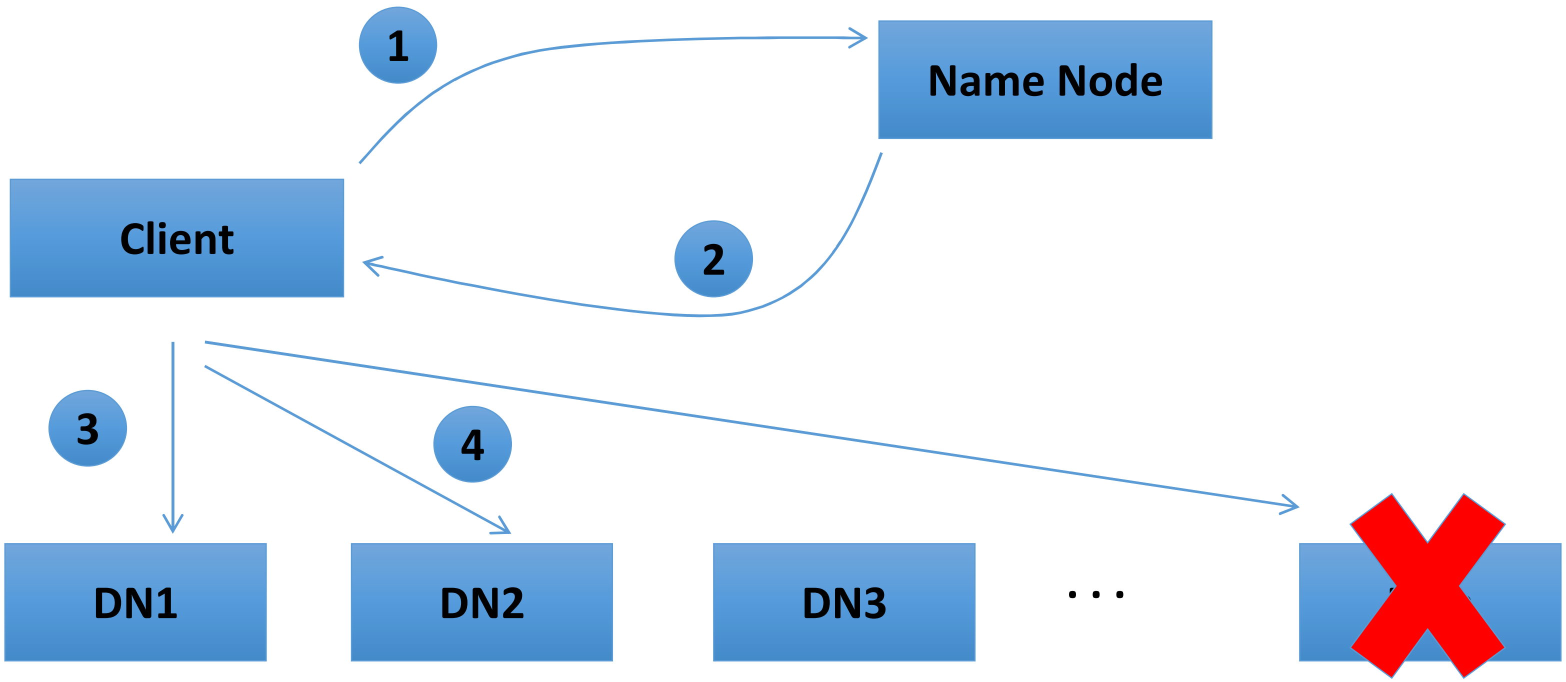
HDFS Inside: Name Node



HDFS Inside: Blocks

- HDFS Block size is by default **128 MB**, why it is much larger than regular file system block?
- Reasons:
  - Minimize overhead: disk seek time is almost constant
  - Example: seek time: 10 ms, file transfer rate: 100MB/s, overhead (seek time/a block transfer time) is 1%, what is the block size?
  - 100 MB (HDFS→ 128 MB)

HDFS Inside: Read

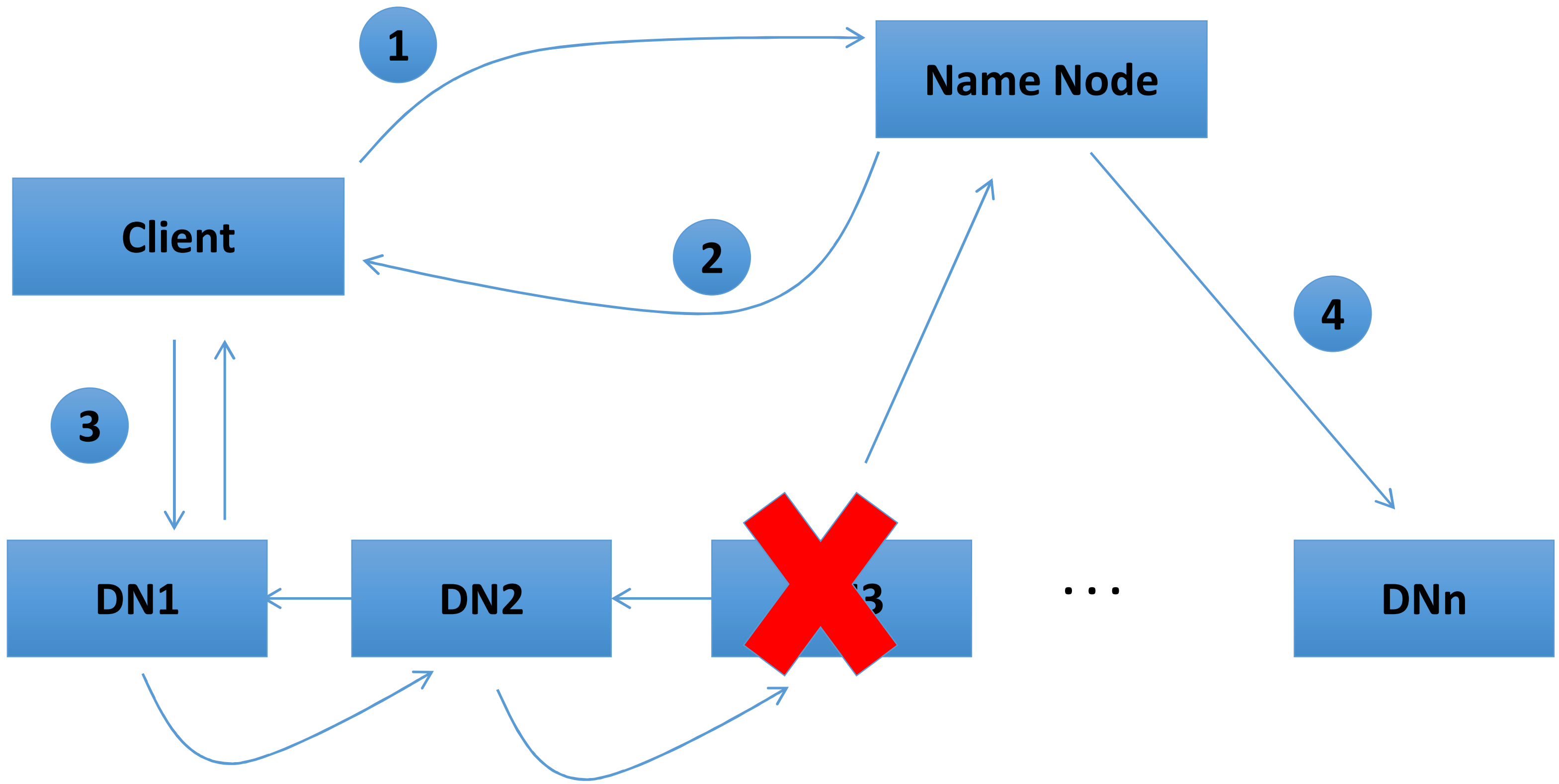


1. Client connects to NN to read data
2. NN tells client where to find the data blocks
3. Client reads blocks directly from data nodes (without going through NN)
4. In case of node failures, client connects to another node that serves the missing block

HDFS Inside: Read

- Q: Why does HDFS choose such a design for read? Why not ask client to read blocks through NN?
-
- Reasons:
  - Prevent NN from being the bottleneck of the cluster
  - Allow HDFS to scale to large number of concurrent clients
  - Spread the data traffic across the cluster

HDFS Inside: Write



1. Client connects to NN to write data
2. NN tells client write these data nodes
3. Client writes blocks directly to data nodes with desired replication factor
4. In case of node failures, NN will figure it out and replicate the missing blocks

HDFS Inside: Write

- Q: Where should HDFS put the three replicas of a block? What tradeoffs we need to consider?
- Tradeoffs:
  - Reliability
  - Write Bandwidth
  - Read Bandwidth
- Q: What are some possible strategies?

HDFS Inside: Write

- Replication Strategy vs Tradeoffs

	Reliability	Write Bandwidth	Read Bandwidth
Put all replicas on one node			
Put all replicas on different racks			

HDFS Inside: Write

- Replication Strategy vs Tradeoffs

	Reliability	Write Bandwidth	Read Bandwidth
Put all replicas on one node			
Put all replicas on different racks			
HDFS: 1→ same node as client 2→ a node on different rack 3→ a different node on the same rack as 2			

Hadoop MapReduce

What is MapReduce?

- Terms are borrowed from Functional Language (e.g., Lisp)
- Sum of squares:
- (map square '(1 2 3 4))
  - Output: (1 4 9 16)
  - [processes each record sequentially and independently]

- (reduce + '(1 4 9 16))
  - (+ 16 (+ 9 (+ 4 1)))
  - Output: 30
  - [processes set of all records in batches]

- Let's consider a sample application: Wordcount
  - You are given a huge dataset (e.g., Wikipedia dump or all of Shakespeare's works) and asked to list the count for each of the words in each of the documents therein

Map

- Process individual records to generate intermediate key/value pairs.

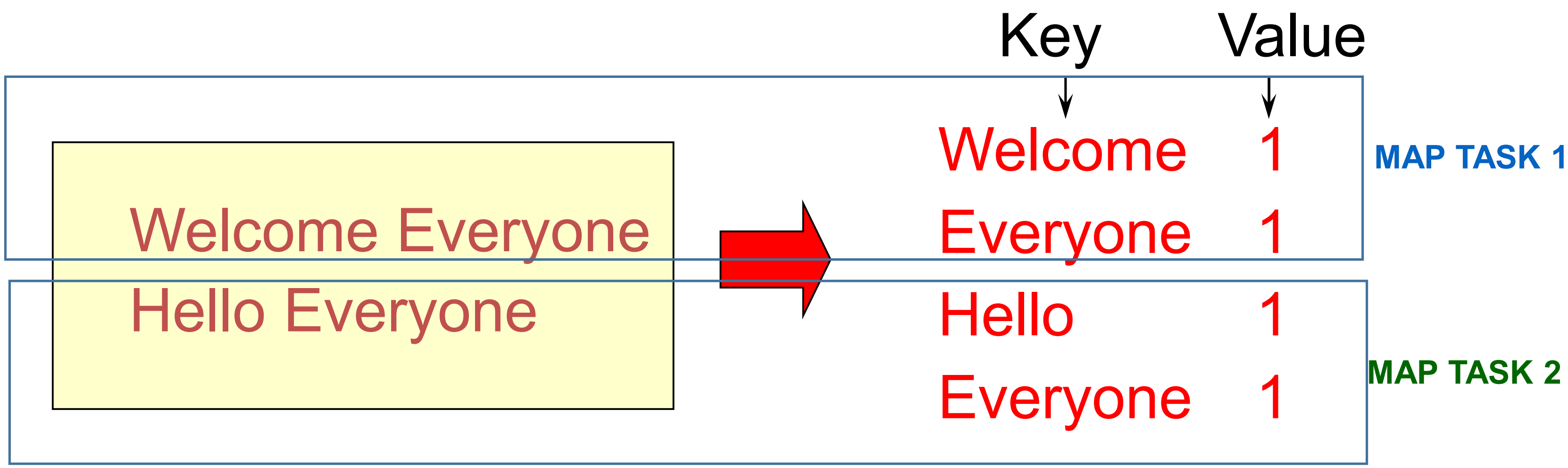
Welcome Everyone
Hello Everyone

Input <filename, file text>

Key	Value
Welcome	1
Everyone	1
Hello	1
Everyone	1

Map

- Parallelly Process individual records to generate intermediate key/value pairs.



Input <filename, file text>

Reduce

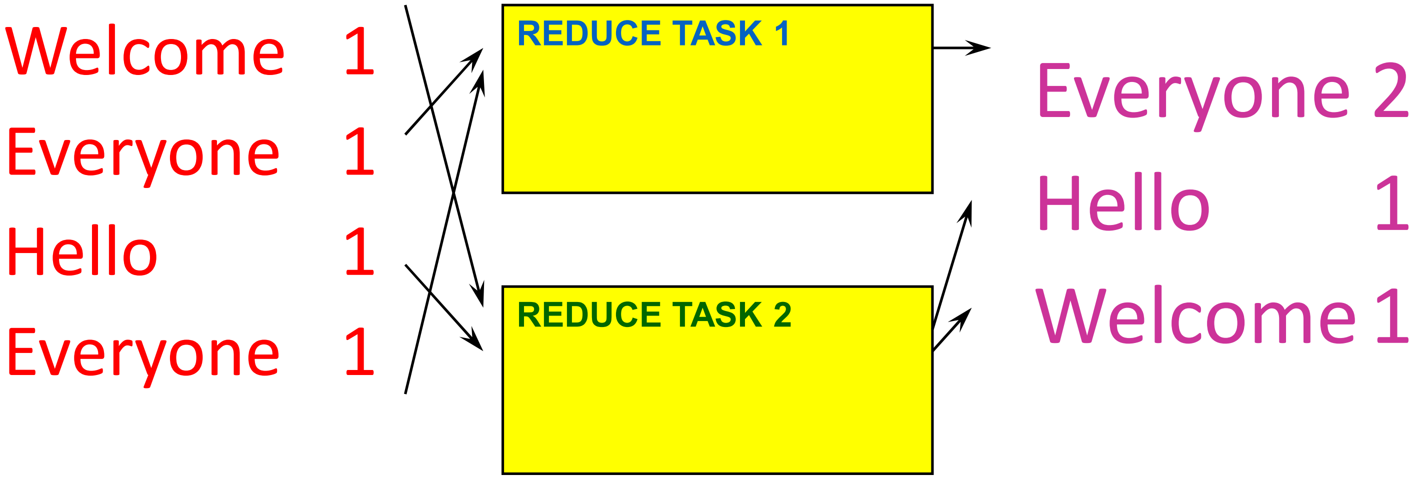
- Reduce processes and merges all intermediate values associated per key

Welcome	1
Everyone	1
Hello	1
Everyone	1

Key	Value
Everyone	2
Hello	1
Welcome	1

Reduce

- Each key assigned to one Reduce
- Parallelly Processes and merges all intermediate values by partitioning keys



- Popular: Hash partitioning, i.e., key is assigned to reduce # = hash(key)%number of reduce servers

Hadoop Code - Map

```
public static class MapClass extends MapReduceBase implements
                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, OutputCollector<Text,
IntWritable> output, Reporter reporter) throws IOException {

    String line = value.toString();

    StringTokenizer itr = new StringTokenizer(line);

    while (itr.hasMoreTokens()) {

              word.set(itr.nextToken());

              output.collect(word, one);

    }

}
```

Hadoop Code - Reduce

```
public static class ReduceClass extends MapReduceBase implements
Reducer<Text, IntWritable, Text, IntWritable> {
  public void reduce(Text key, Iterator<IntWritable> values,
OutputCollector<Text, IntWritable> output, Reporter reporter)
throws IOException {
        int sum = 0;
        while (values.hasNext()) {
          sum += values.next().get();
        }
        output.collect(key, new IntWritable(sum));
  }
}
```

Hadoop Code - Driver

```
// Tells Hadoop how to run your Map-Reduce job
public void run (String inputPath, String outputPath) throws Exception
{
  // The job. WordCount contains MapClass and Reduce.
  JobConf conf = new JobConf(WordCount.class);
  conf.setJobName("mywordcount");
  // The keys are words
  (strings) conf.setOutputKeyClass(Text.class);
  // The values are counts (ints)
  conf.setOutputValueClass(IntWritable.class);
  conf.setMapperClass(MapClass.class);
  conf.setReducerClass(ReduceClass.class);
  FileInputFormat.addInputPath(conf, newPath(inputPath));
  FileOutputFormat.setOutputPath(conf, new Path(outputPath));
  JobClient.runJob(conf);
}
```

Some Applications of MapReduce

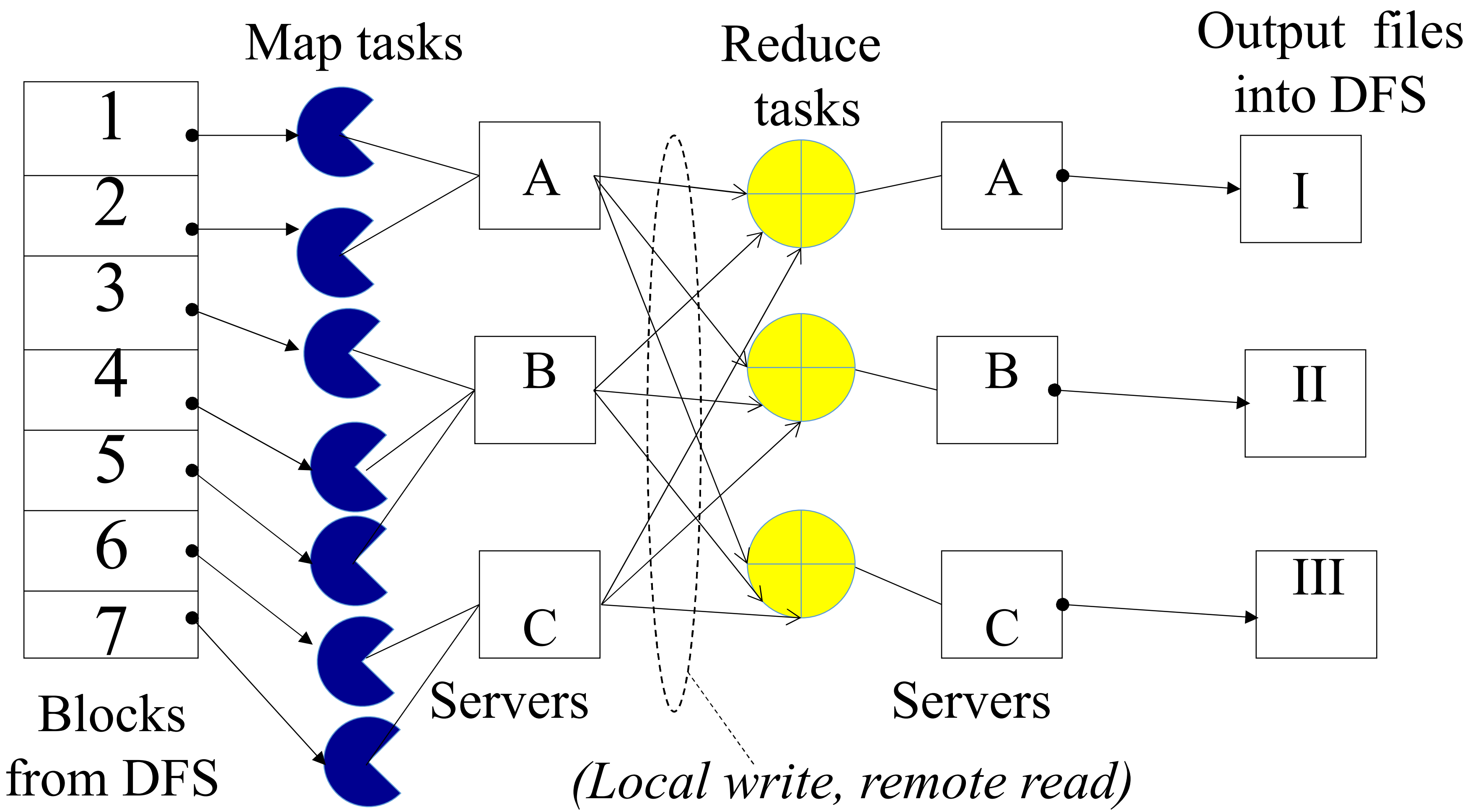
- Grep:
  - Input: large set of files
  - Output: lines that match pattern

  - Map – Emits a line if it matches the supplied pattern
  - Reduce – Copies the intermediate data to output

- WordCount:
  - Input: large set of files
  - Output: For each word, <word, word_count>

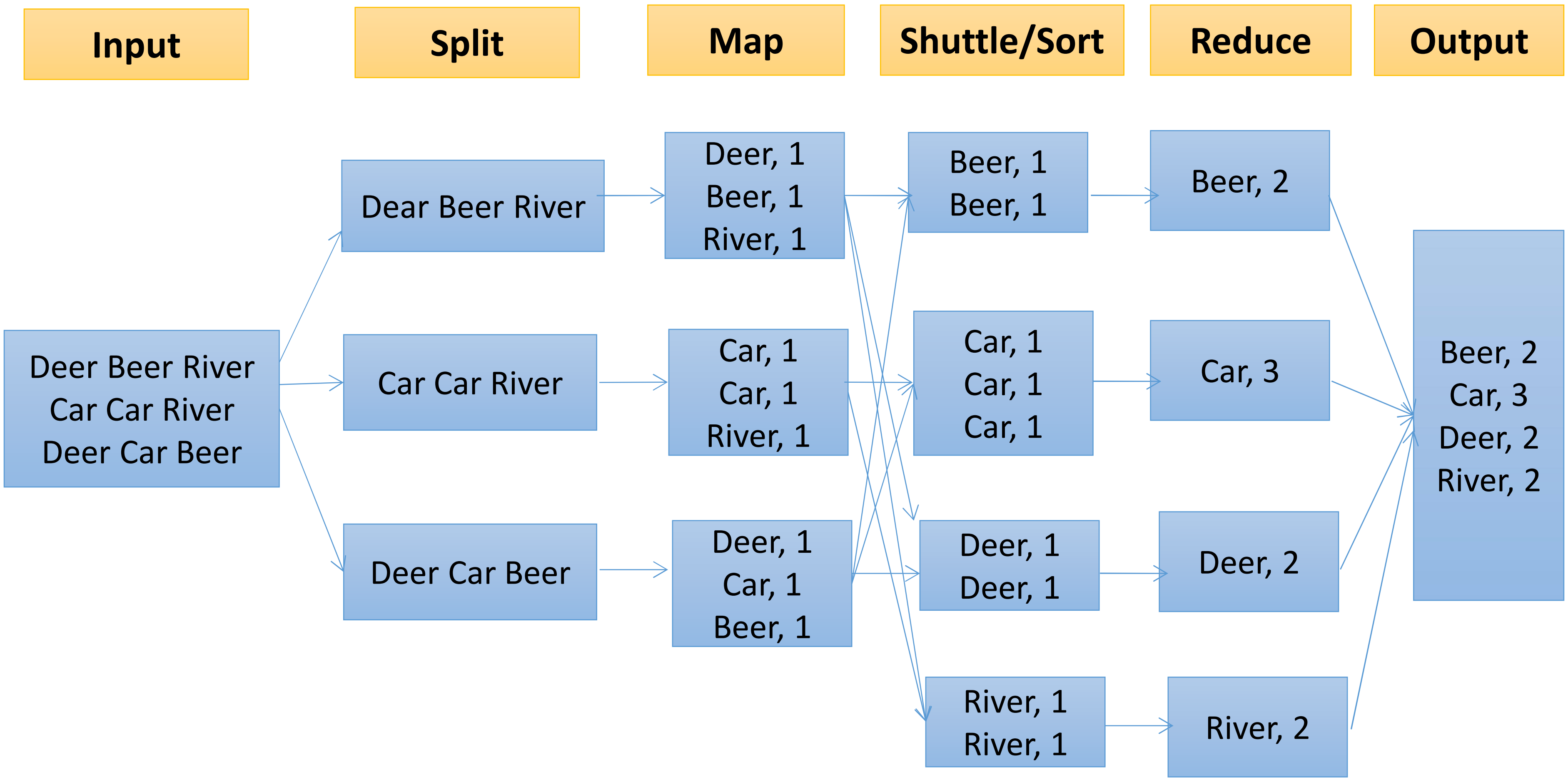
  - Map – Emits <word, 1>
  - Reduce – Sum up the intermediate count to output

MapReduce Data Flow



Resource Manager (assigns maps and reduces to servers)
Idea: Bring computations to data!

MapReduce Example: Word Count



MapReduce Example: Word Count