Distributed Computing Platform

Presented By

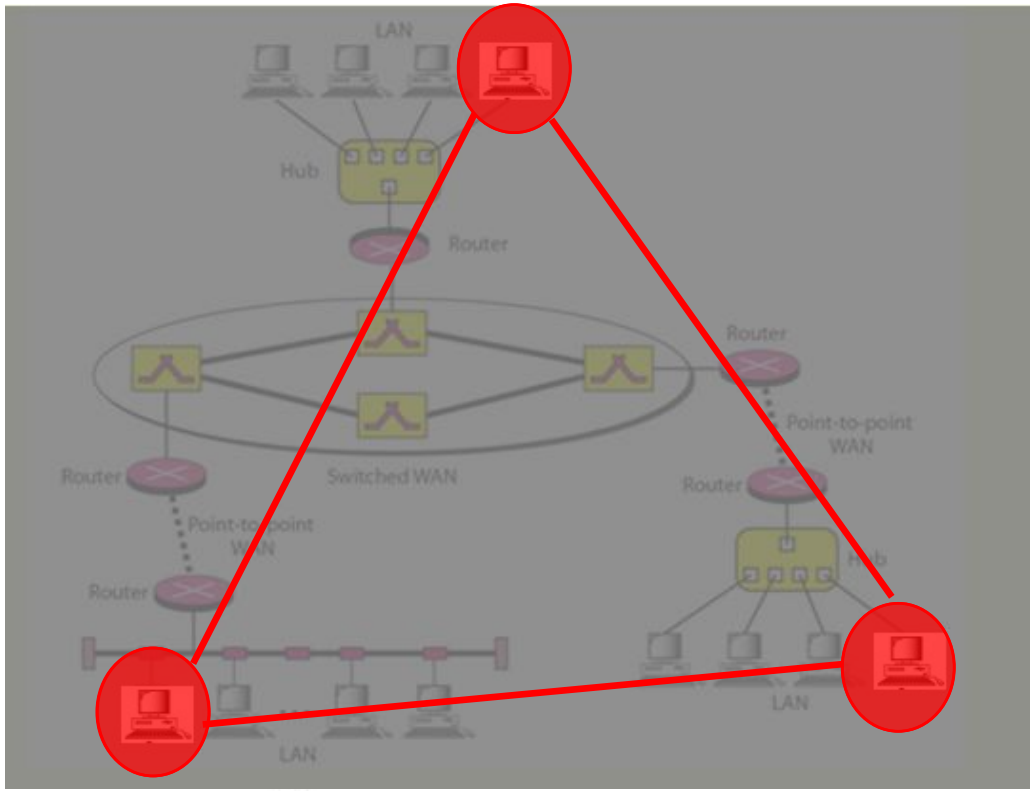
Dr. Sunirmal Khatua

Visvesvaraya Young Faculty Fellow, Govt. of India
Assistant Professor, University of Calcutta

What is a Distributed Computing Platform?

- A Distributed System is a Collection of SMALL independent computers that appears to its users as a single BIG coherent system.
  - *Multiple Processes*
  - *Interprocess Communication*
  - *Disjoint Address Spaces*
  - *Collective Goal*

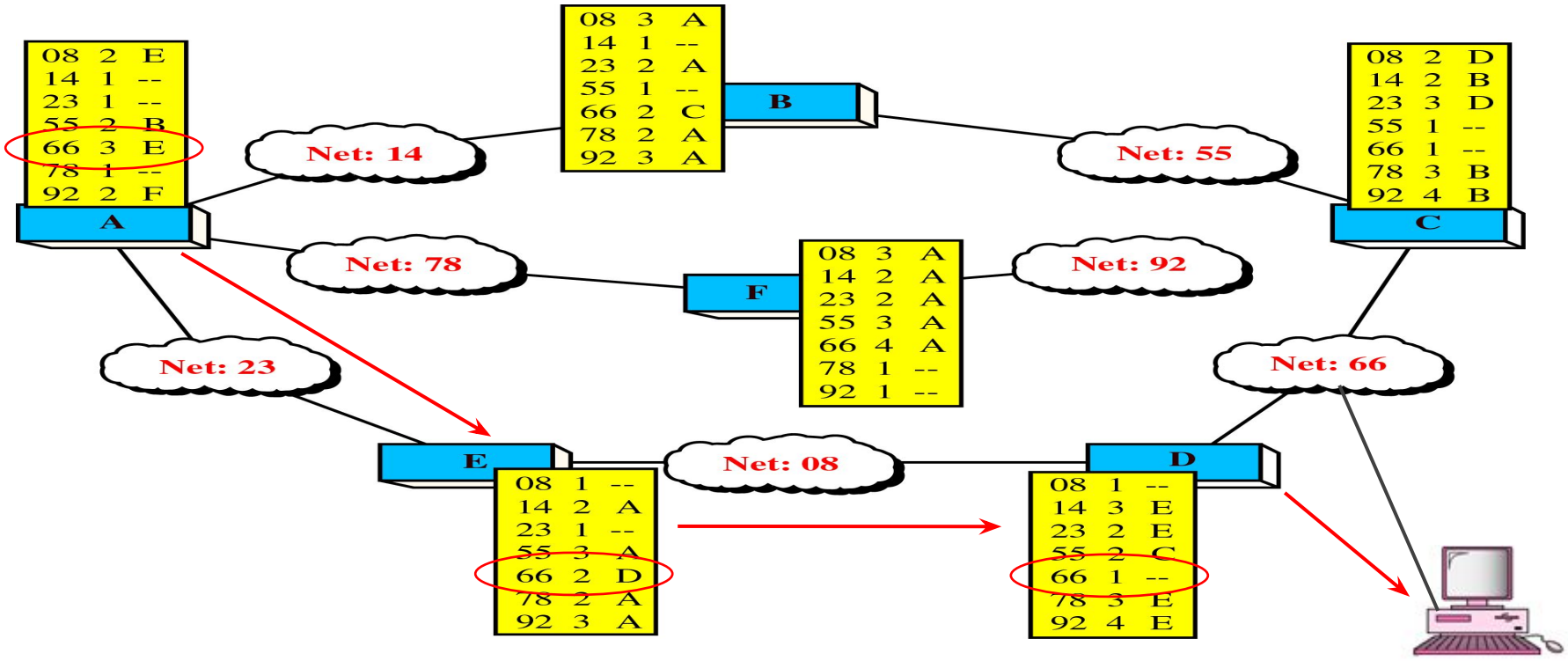
The Gulliver's Travel



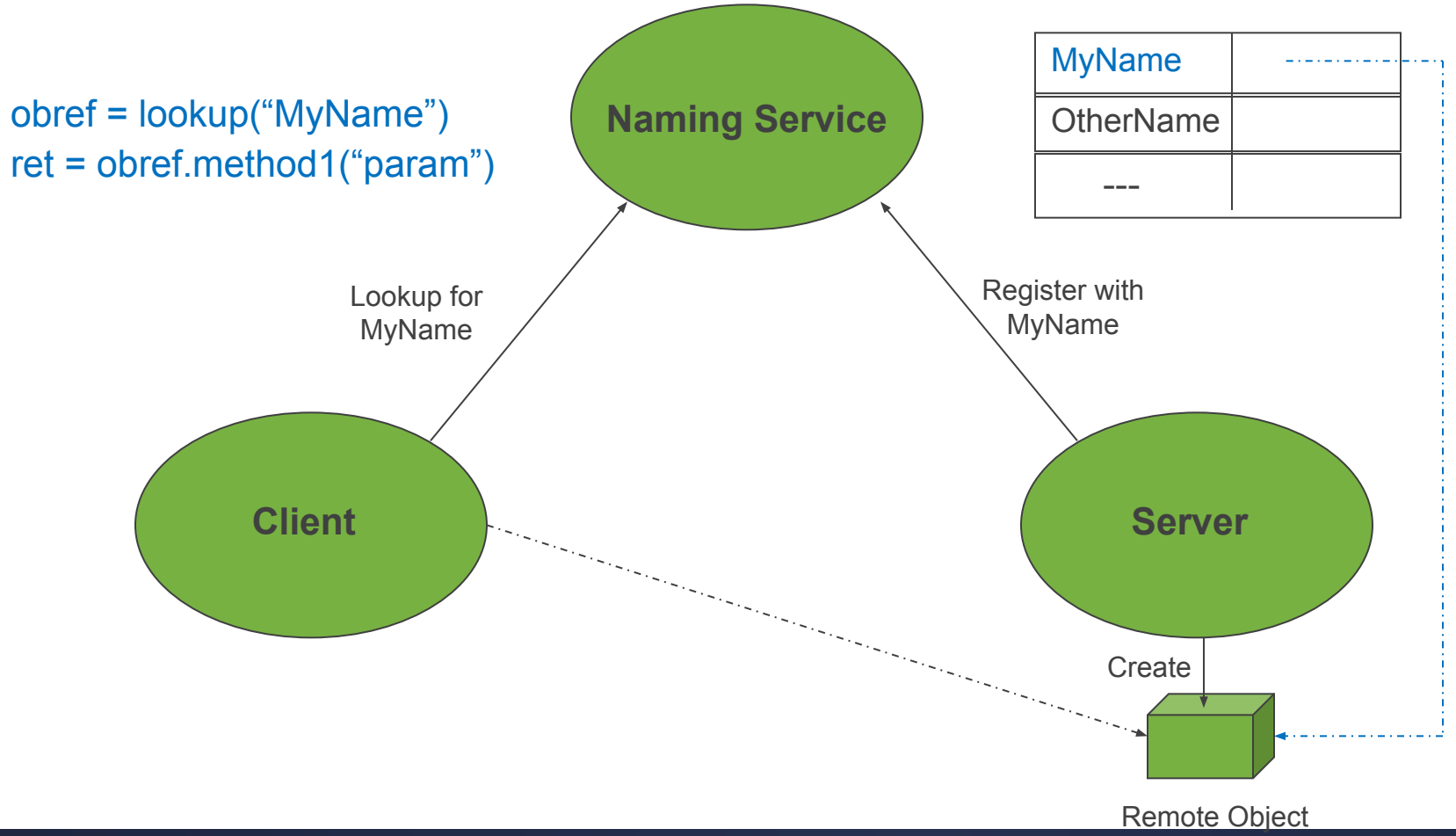
What is a Distributed Computing Platform?

- A Distributed System is a Collection of SMALL independent computers that appears to its users as a single BIG coherent system.
  - *Multiple Processes*
  - *Interprocess Communication*
  - *Disjoint Address Spaces*
  - *Collective Goal*

Routing Algorithm : An Example

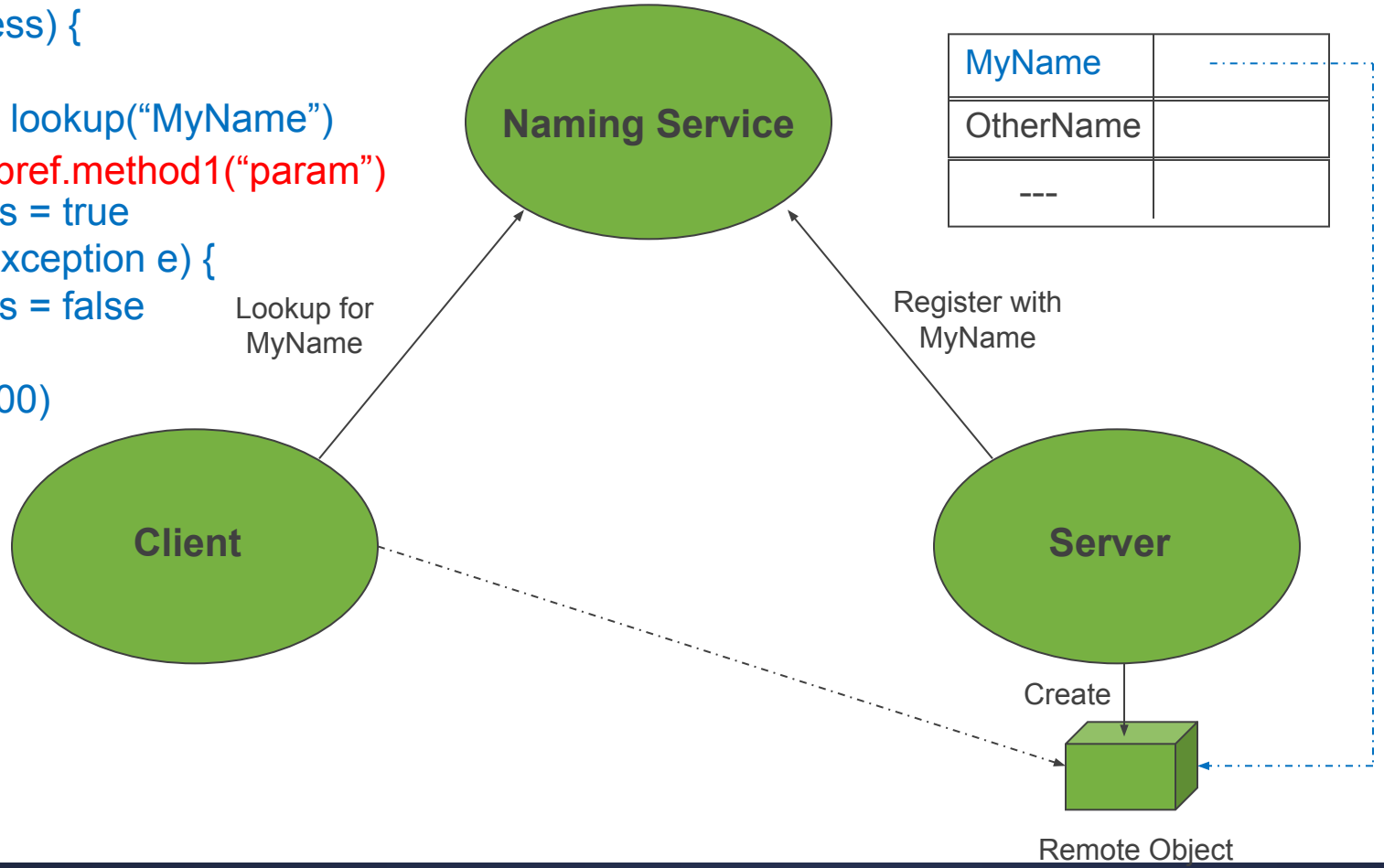


Transparency and Fault Tolerance in Distributed Systems



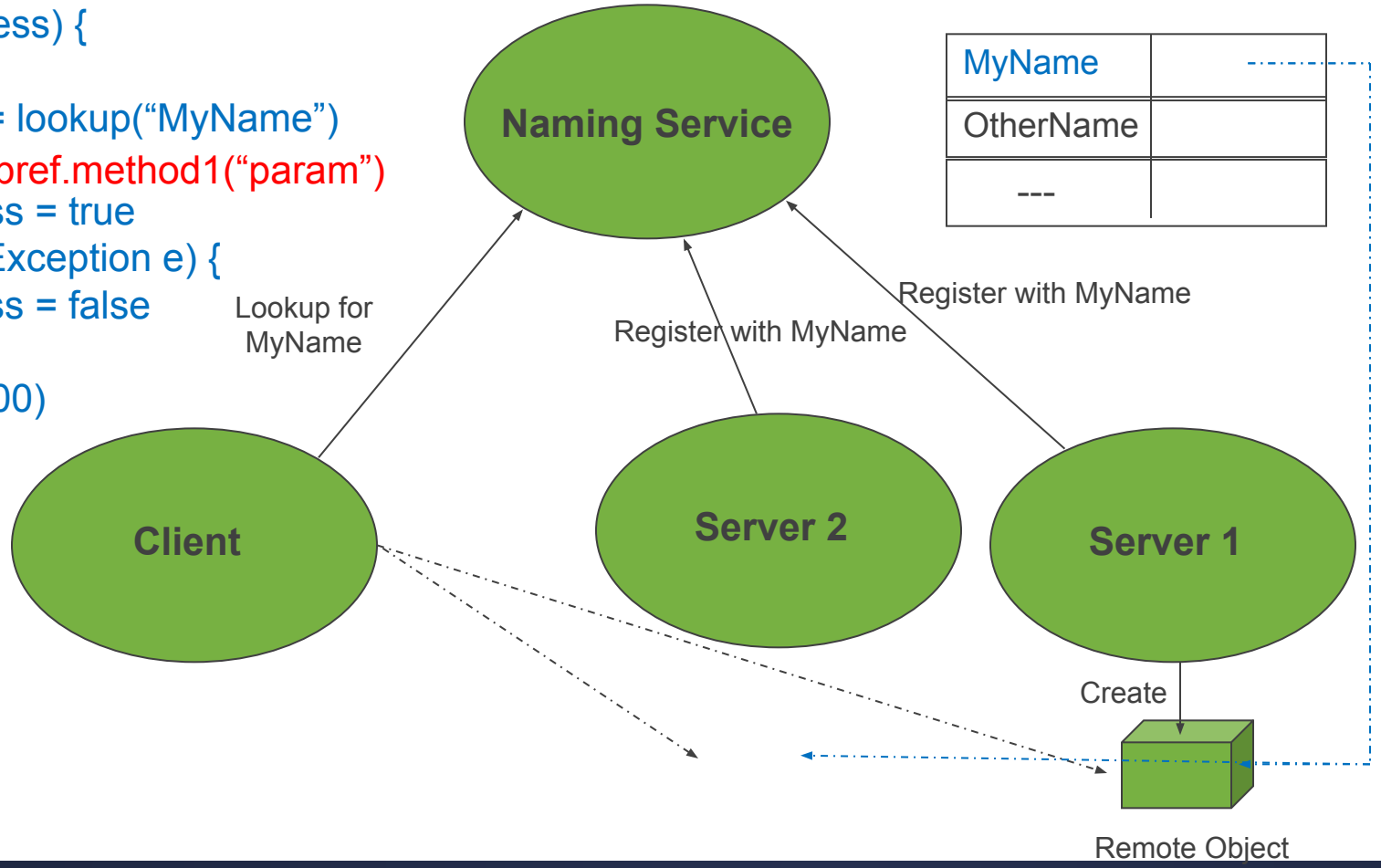
Transparency and Fault Tolerance in Distributed Systems

```
success = false
while(!success) {
    try {
       obref = lookup("MyName")
        ret = obref.method1("param")
       success = true
    } catch(Exception e) {
       success = false
    }
    delay(2000)
}
```

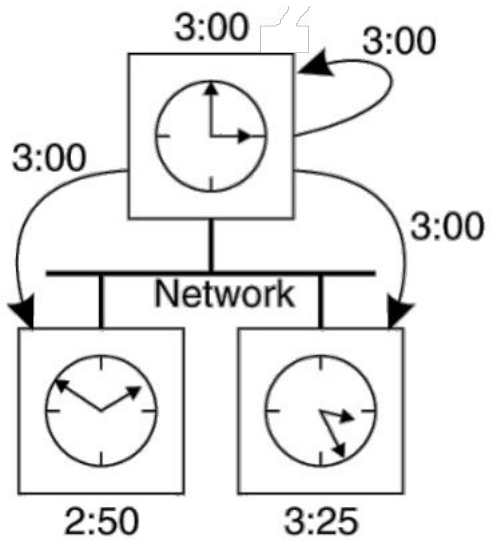


Transparency and Fault Tolerance in Distributed Systems

```
success = false
while(!success) {
    try {
       obref = lookup("MyName")
       ret = obref.method1("param")
       success = true
    } catch(Exception e) {
       success = false
    }
   delay(2000)
}
```



Transparency and Fault Tolerance in Distributed Systems



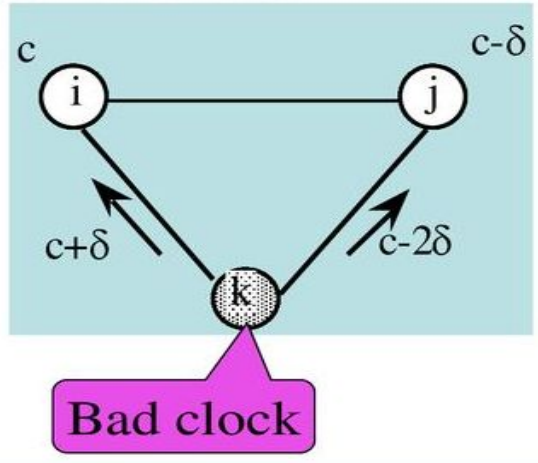
Algorithm : Berkley for Node i

Step 1 : Read all the Clock Values

```
for(j=1; j<=n; j++)
    Read C(i, j)
```

Step 2 : Discard Bad Clocks

```
for(j=1; j<=n; j++)
    if(|C(i,j) – C(i, i)| > δ)  then C(i, j) = C
```

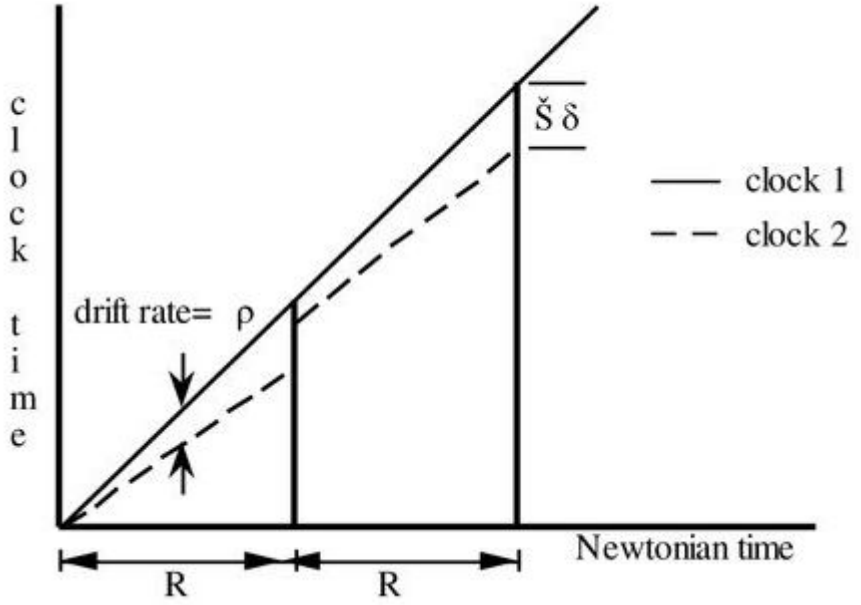


Byzantine Clocks

Step 3 : Set the Clock as Average Clock Values

$$C(i,i) = \frac{\sum_{j=1}^{n} C(i,j)}{n}$$

$$\frac{3\delta t}{n} < \delta$$



Berkley Algorithm can tolerate *t* byzantine clocks if $n > 3t$

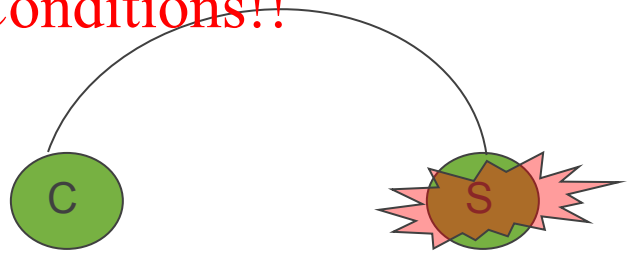
$$R = \frac{\delta - \frac{3\delta t}{n}}{\rho}$$

$$R = \frac{\delta}{\rho(3t+1)}$$

- *Multiple Processes*
- *Interprocess Communication*
- *Disjoint Address Spaces*
- *Collective Goal*

Transparency and Fault Tolerance in Distributed Systems

Impossibility Conditions!!



1. *At least once :* Try until a reply received.
2. *At most once :* Report failure without retry.
3. *Exactly once :* No way to guaranty.

Client
Reissue Strategy

Always
Never
Only when ACKed
Only when not ACKed

Strategy R -> E

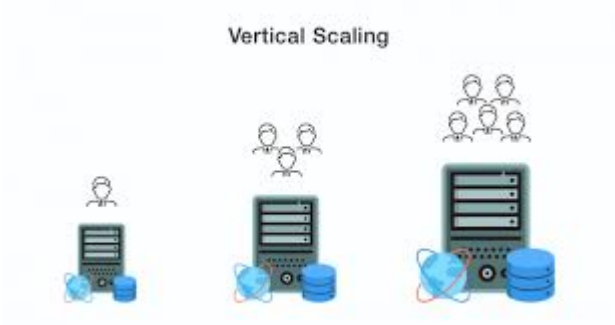
REC	RC(E)	C(RE)
DUP	ONE	ONE
ONE	ZERO	ZERO
DUP	ONE	ZERO
ONE	ZERO	ONE

Server
Strategy E -> R

ERC	EC(R)	C(ER)
DUP	DUP	ONE
ONE	ONE	ZERO
DUP	ONE	ZERO
ONE	DUP	ONE

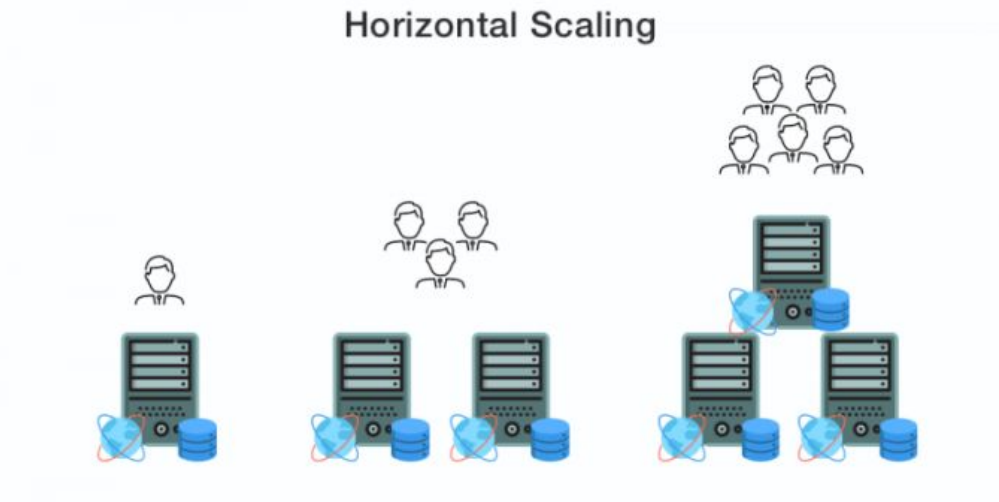
Scalability in Distributed Systems

A scalable distributed system can adapt to changes in demand by expanding its resources or redistributing tasks effectively.



1. **Vertical Scalability (Scale-Up)**:
   Increasing the capacity of existing nodes by adding more resources

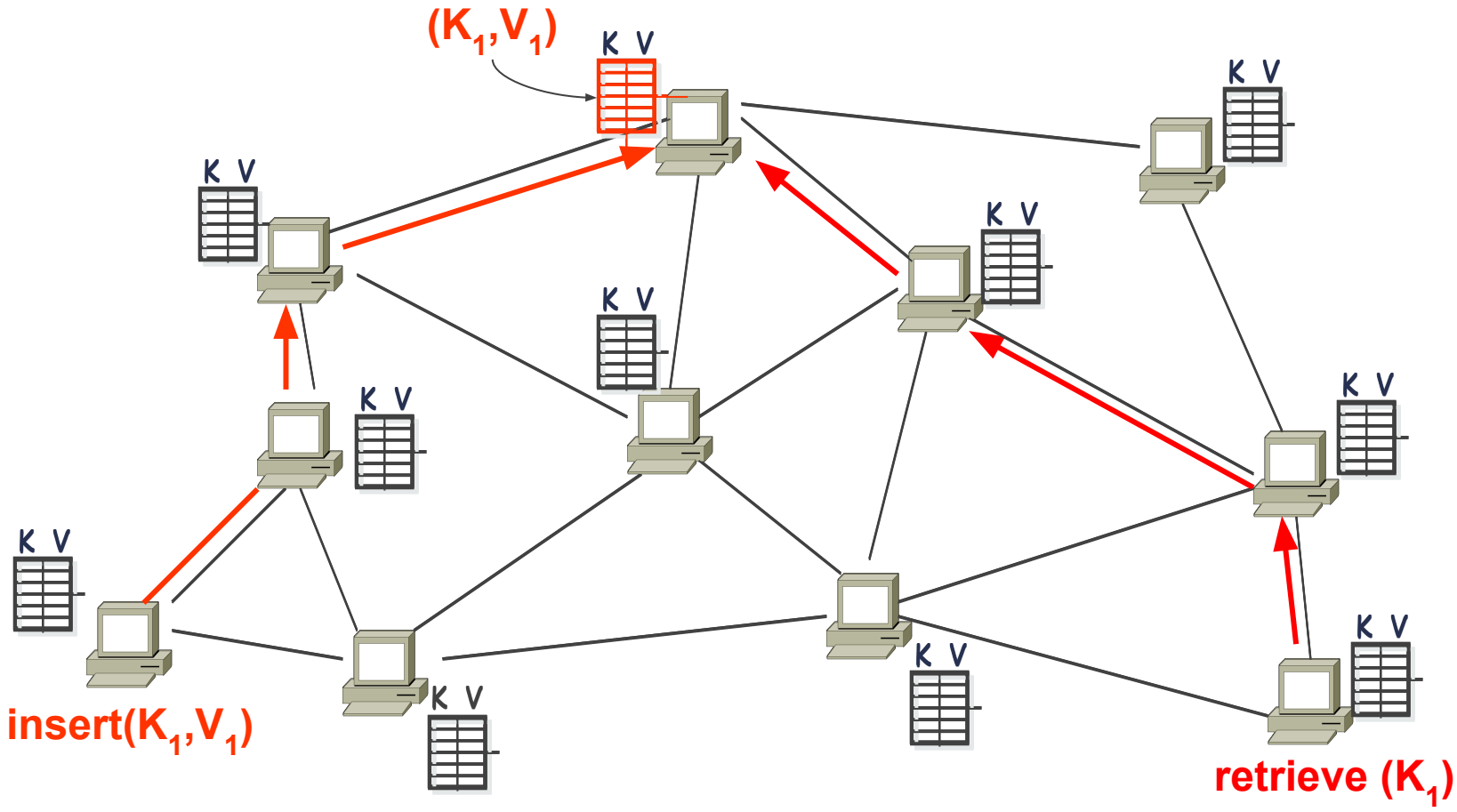
2. **Horizontal Scalability (Scale-Out):**
   Adding more nodes to the system to handle increased demand.



Distributed Hash Table

- **insert(key, value)**
- **value = lookup(key)**

Scalability in Distributed Systems

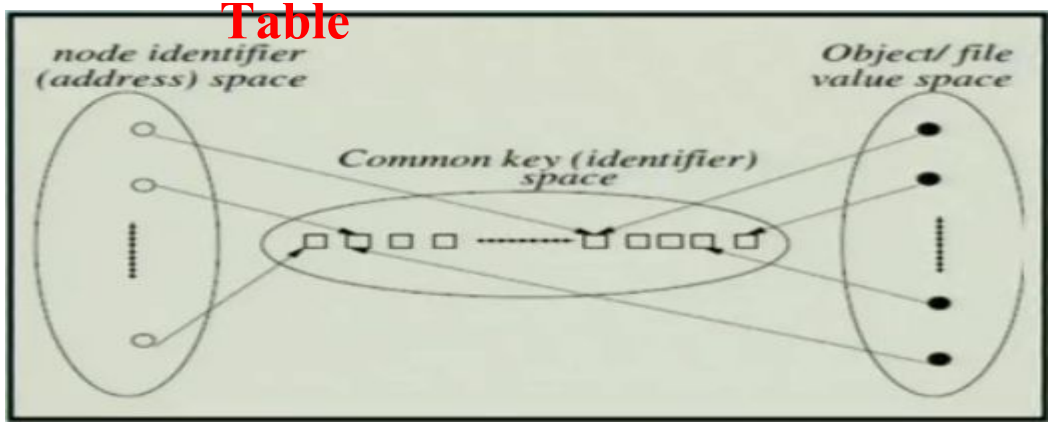


Distributed Hash Table

- **insert(key, value)**
- **value = lookup(key)**

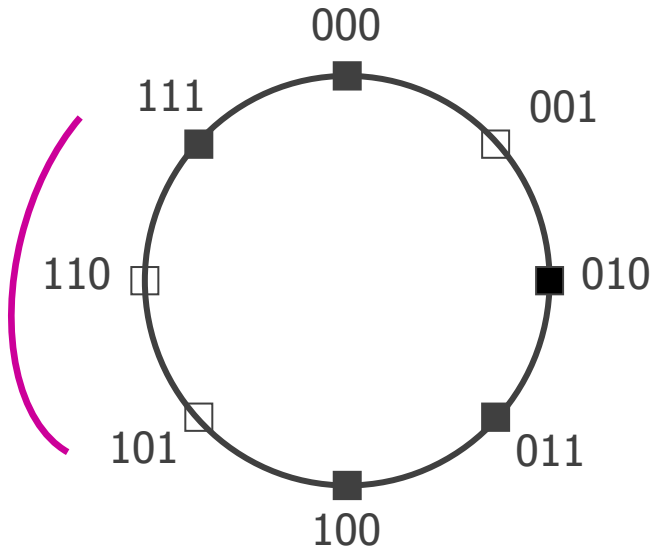
Scalability in Distributed Systems

Chord Distributed Hash Table

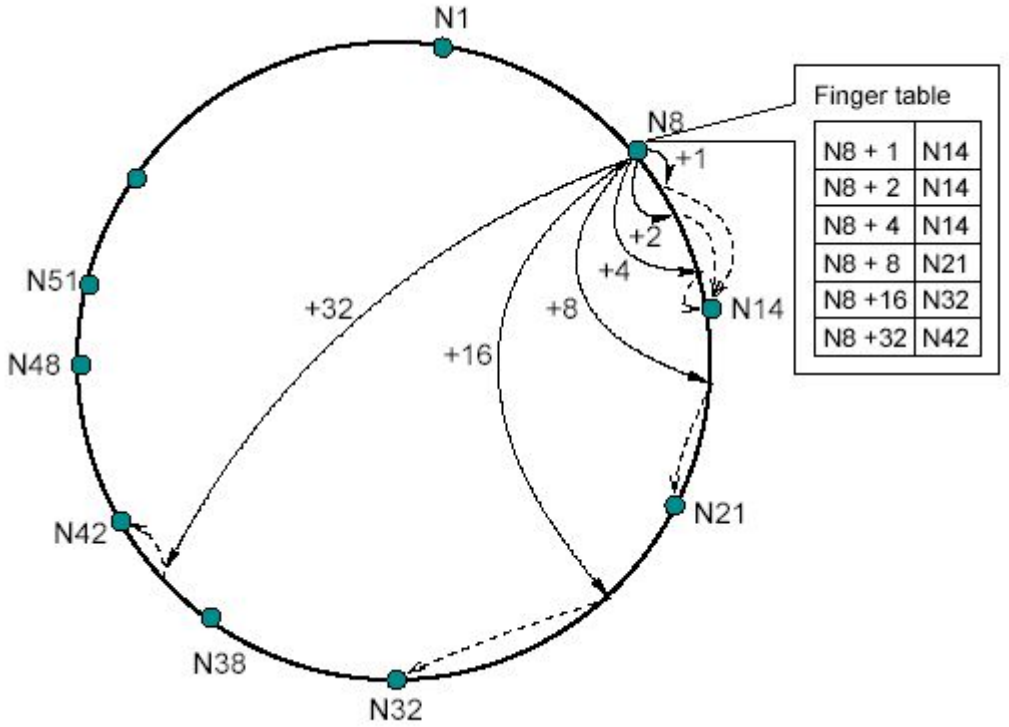


h(IP) → m-bit id

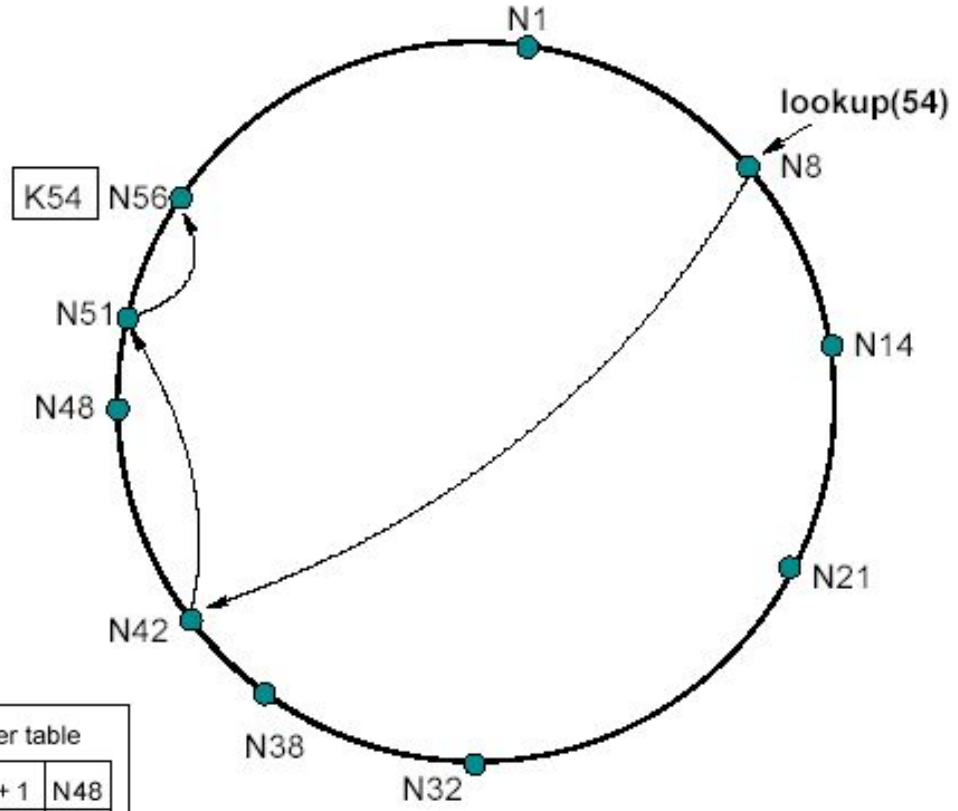
h(file name) → m-bit id



Finger table: *finger[i] = successor (n +*



Scalability in Distributed Systems



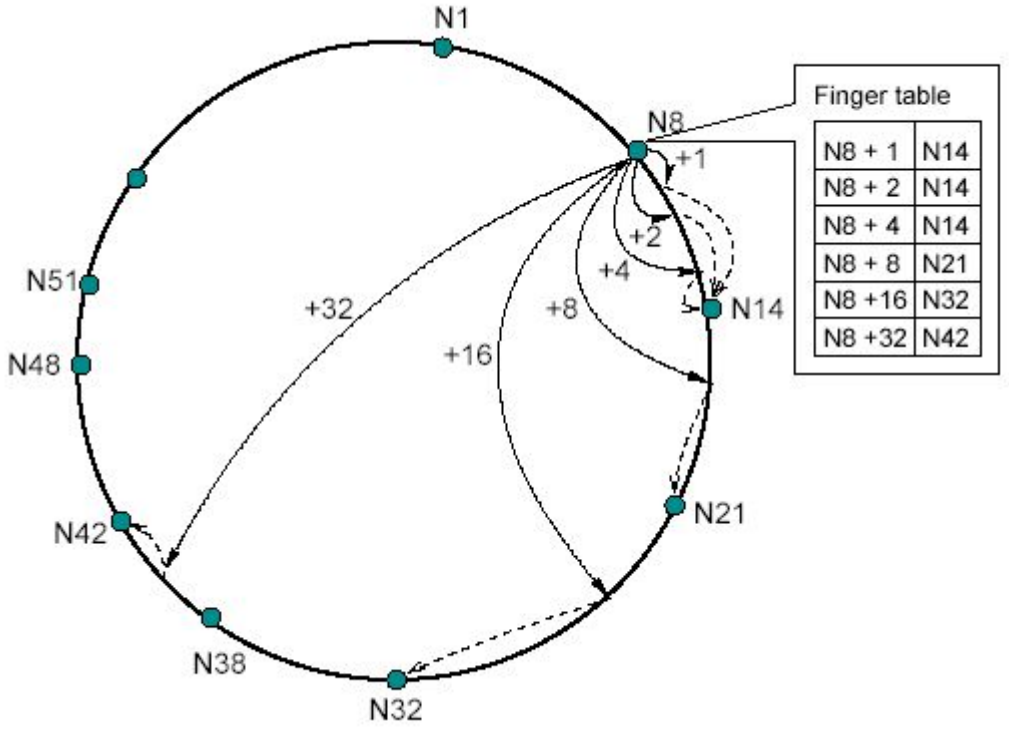
Finger table

N8 + 1	N14
N8 + 2	N14
N8 + 4	N14
N8 + 8	N21
N8 +16	N32
N8 +32	N42

Finger table

N42+ 1	N48
N42+ 2	N48
N42+ 4	N48
N42+ 8	N51
N42+16	N1
N42+32	N14

Finger table: *finger[i] = successor (n +*



Finger table

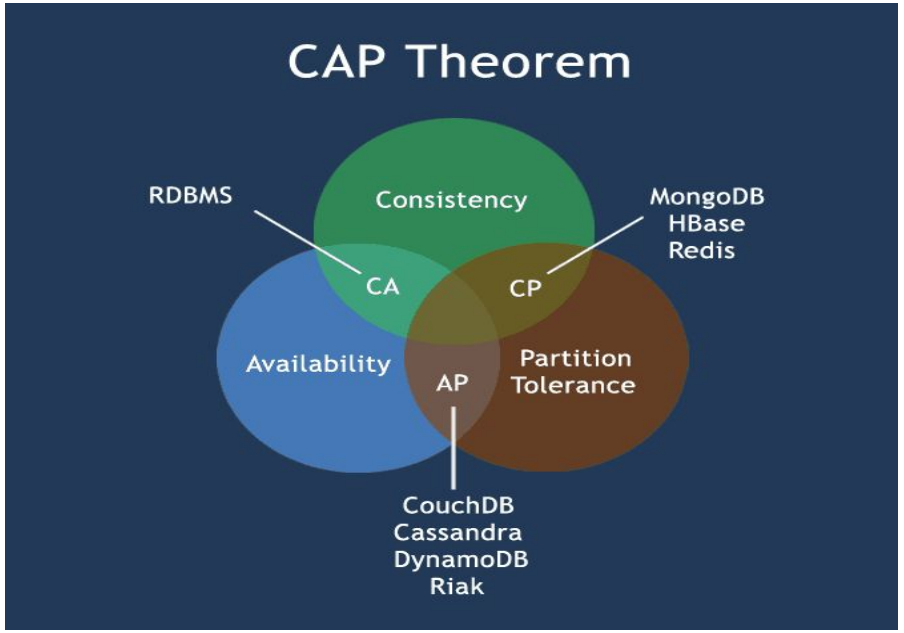
N8 + 1	N14
N8 + 2	N14
N8 + 4	N14
N8 + 8	N21
N8 +16	N32
N8 +32	N42

High Availability in Distributed Systems

Replicate Data Items across multiple Nodes.

Data Items needs to be Locked for multiple Transactions.

There are 3 different approached to Lock.



Primary Copy Approach:

- ✔One of the replica is chosen as the Primary Copy.
- ✔All lock/unlock requests goes to the Node containing Primary Copy.
- ✔Disadvantage – If the primary Site fails, the data becomes inaccessible.

Majority Protocol Approach

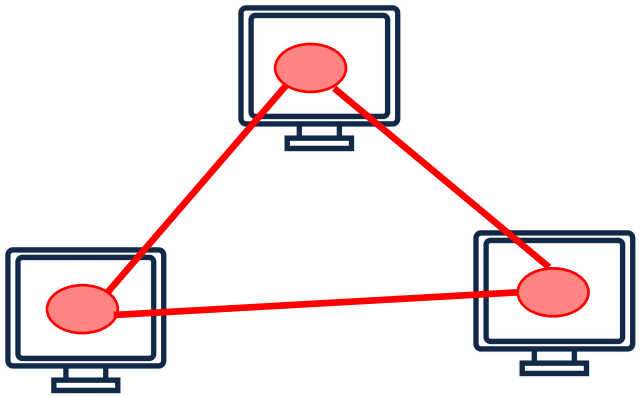
- ✔If the data item is replicated in n no. of sites, then the lock/unlock request must be sent to more than n/2 sites containing a replica of the data item.
- ✔It requires 2(n/2 + 1) messages for handling Lock request & (n/2 + 1) messages for handling Unlock request.

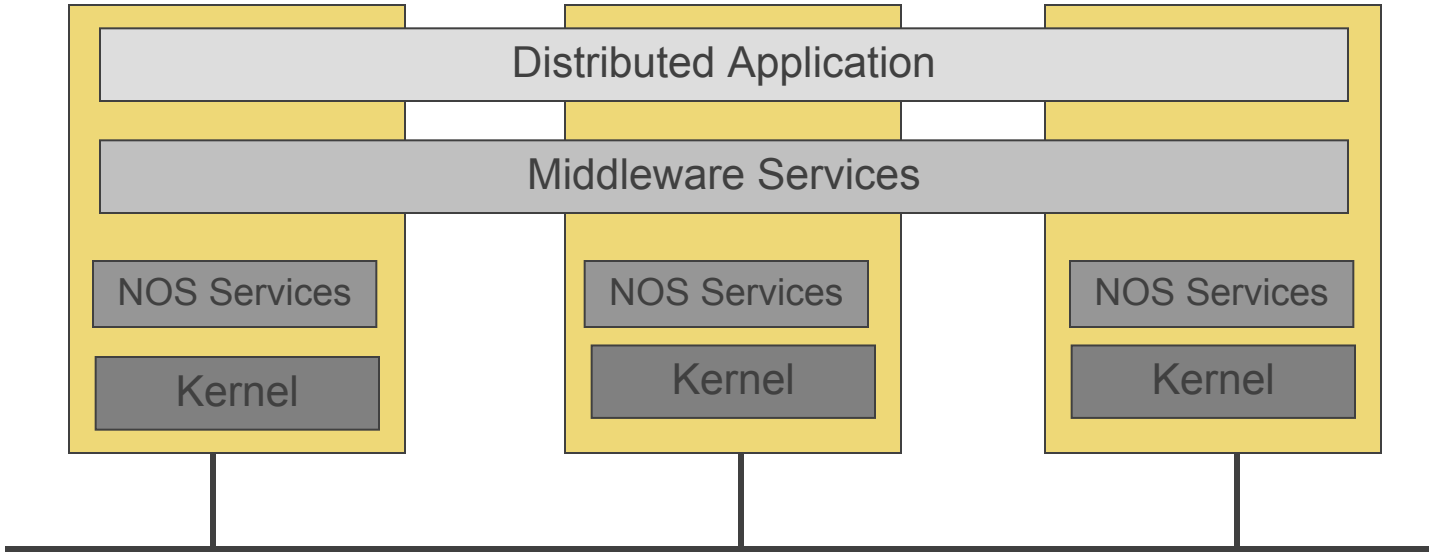
Biased Protocol Approach

- ✔Here shared locks are given more preference over exclusive locks.
- ✔For shared lock it sends lock/unlock request to any one site containing a replica.
- ✔For exclusive lock, it has to sent lock/unlock request to all the replica sites.

What is Distributed Computing Platform?

- A Distributed System is a Collection of SMALL independent computers that appears to its users as a single BIG coherent system.
  - *Multiple Processes*
  - *Interprocess Communication*
  - *Disjoint Address Spaces*
  - *Collective Goal*





What is a Big Data?

Data Volume

- 44x increase from 2009 2020
- From 0.8 zettabytes to 35zb

Data volume is increasing exponentially

Data Variety

- Semi structure Data
- Unstructured Data

Data Velocity

- Data is generated very fast and need to be processed very fast (Stream Computing)
- Late decisions missing opportunities

Data Veracity

- Erroneous Data
- Missing Data

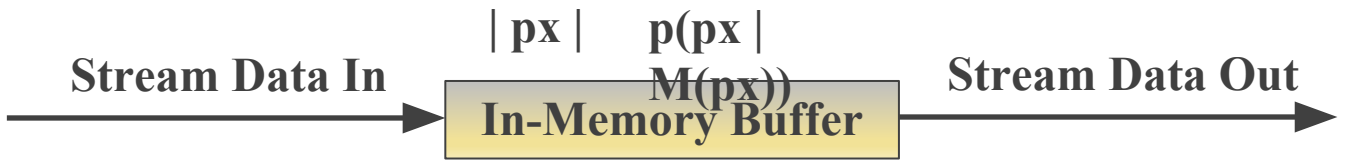
Data Value

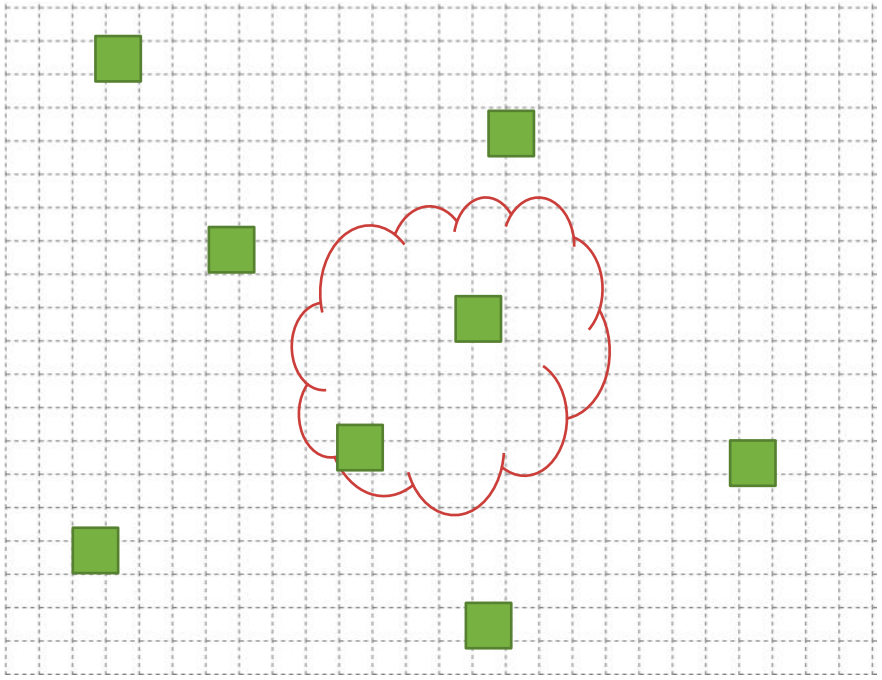
- Business Insight
- Inherent Pattern

Considering Velocity : Stream Computing

Data Velocity

- Data is generated very fast and need to be processed very fast (Stream Computing)
- Late decisions missing opportunities

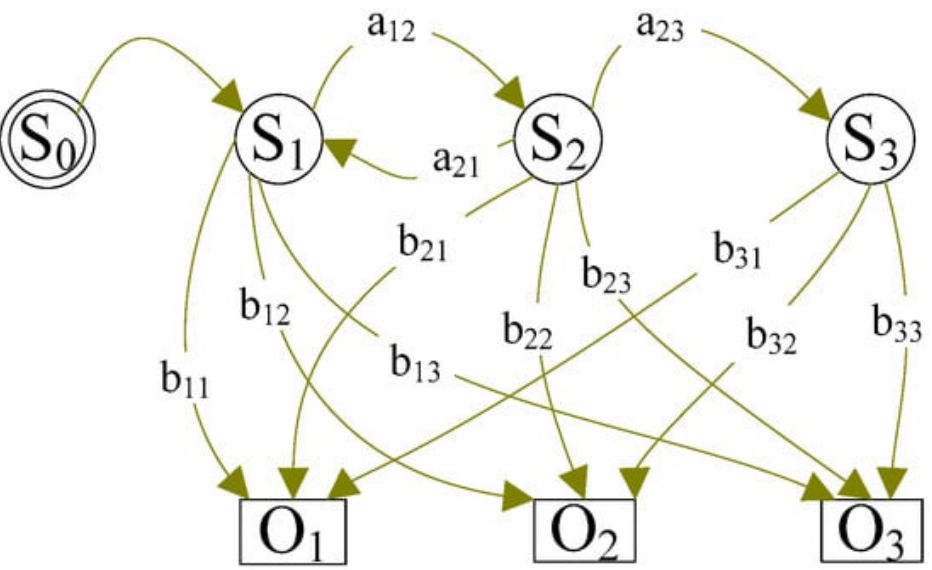




Given the Membership M(px) = 1 if px is inside the region

```
area = 0;
for(i=1 to #rows)
    for(j=1 to #cols)
        if(M(p(i,j) ==
1)
            area ++
```

Considering Veracity : Error Correction



Hidden Markov Model

Maximum Likelihood Estimation Algorithm

Algorithm:

function VITERBI(O, S, π, A, T,B) : X

for each state s *from* 1 *to* N *do*

Viterbi[s,1] ← π_s * $B_{s,o1}$

Backpointer[s,1] ← 0

for each time step t *from* 2 *to* T *do*

for each state s *from* 1 *to* N *do*

Viterbi[s, t] ← $\max_{k=1}^{N}$ (Viterbi [k , t-1] *$A_{k,s}$ *B_{s,o_t})

Backpointer[s, t] ← $\operatorname{argmax}_{k=1}^{N}$ (Viterbi [k , t-1] * $A_{k,s}$ * B_{s,o_t})

End *for*

End *for*

Challenges with Big Data

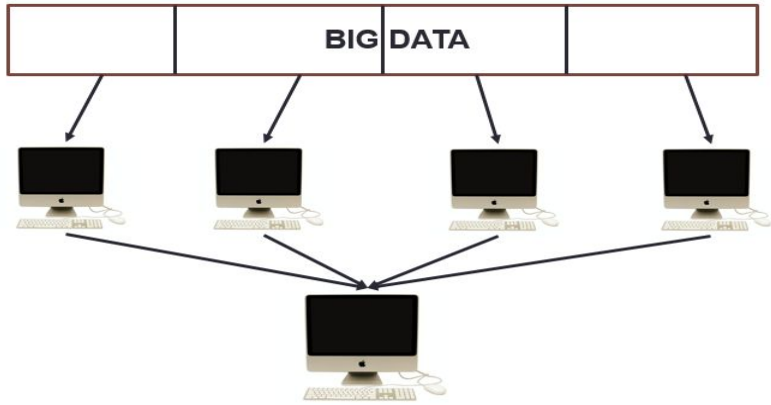
How to access Big Data?

Size = 200 TB with Disk access speed = 50 MB/s

Time to just read = 4 million seconds = 46+ days

Solution?

Distribute Data



How to handle Node Failure?

Let Node lifetime = 1000 days

1000 nodes cluster □ 1 failure/day

1M nodes cluster □ 1000 failures/day

Solution

Data Replication to 3 or more Nodes

Network Bottleneck

Let Network bandwidth = 1gpps

Moving 10 TB of data required 1 day

Solution

Move Code to Data source

Challenges with Big Data

How to access Big Data?

Size = 200 TB with Disk access speed = 50 MB/s
Time to just read = 4 million seconds = 46+ days

Solution?

Distribute Data

How to handle Node Failure?

Let Node lifetime = 1000 days
1000 nodes cluster □ 1 failure/day
1M nodes cluster □ 1000 failures/day

Solution

Data Replication to 2 or more Nodes

Network Bottleneck

Let Network bandwidth = 1gpps
Moving 10 TB of data required 1 day

Solution

Move Code to Data source

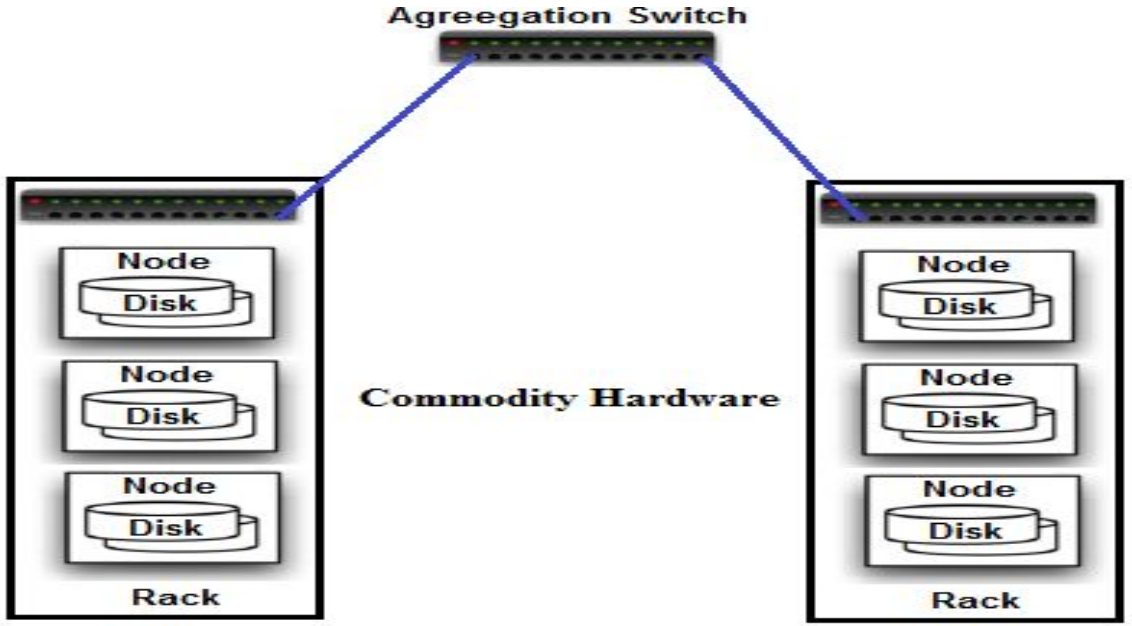
Hadoop and Spark are one stop Solution

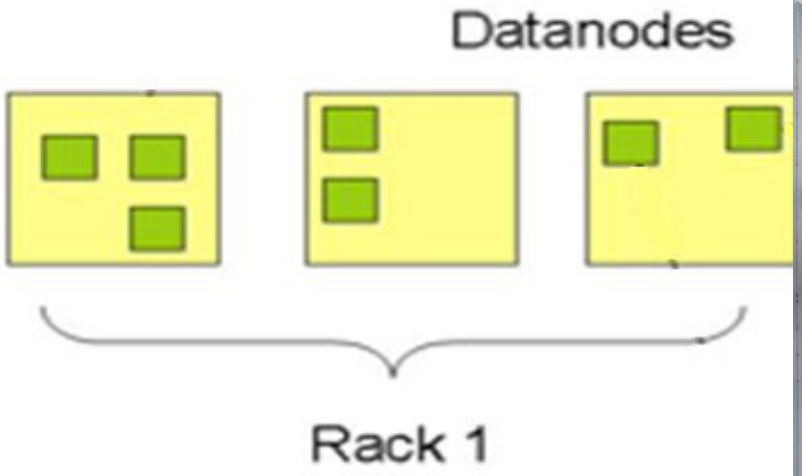
HDFS □ Distributed Storage

Map Reduce □ Parallel Processing

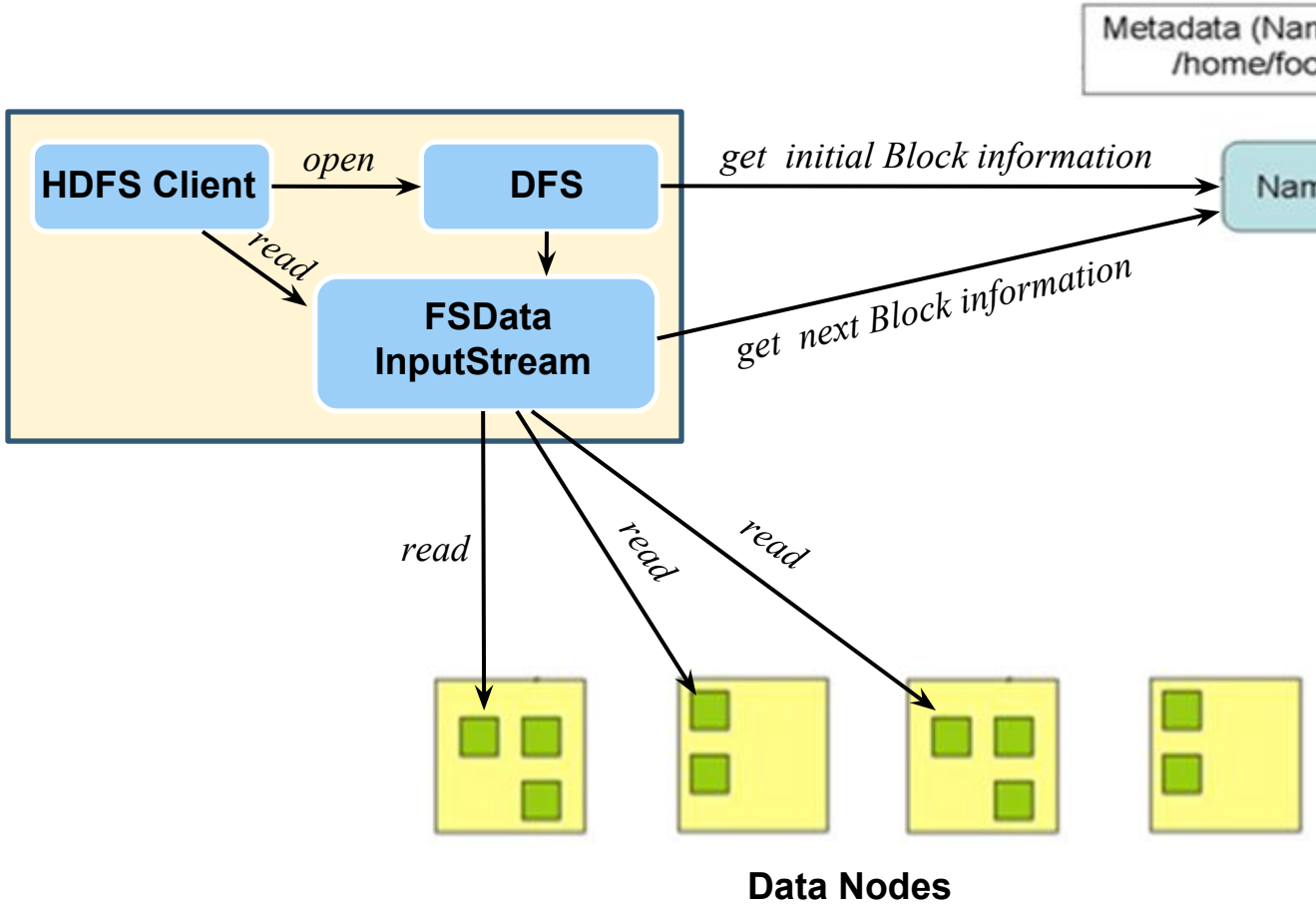
Stream Computing □ In-Memory and Stream

HDFS for Distributed Storage



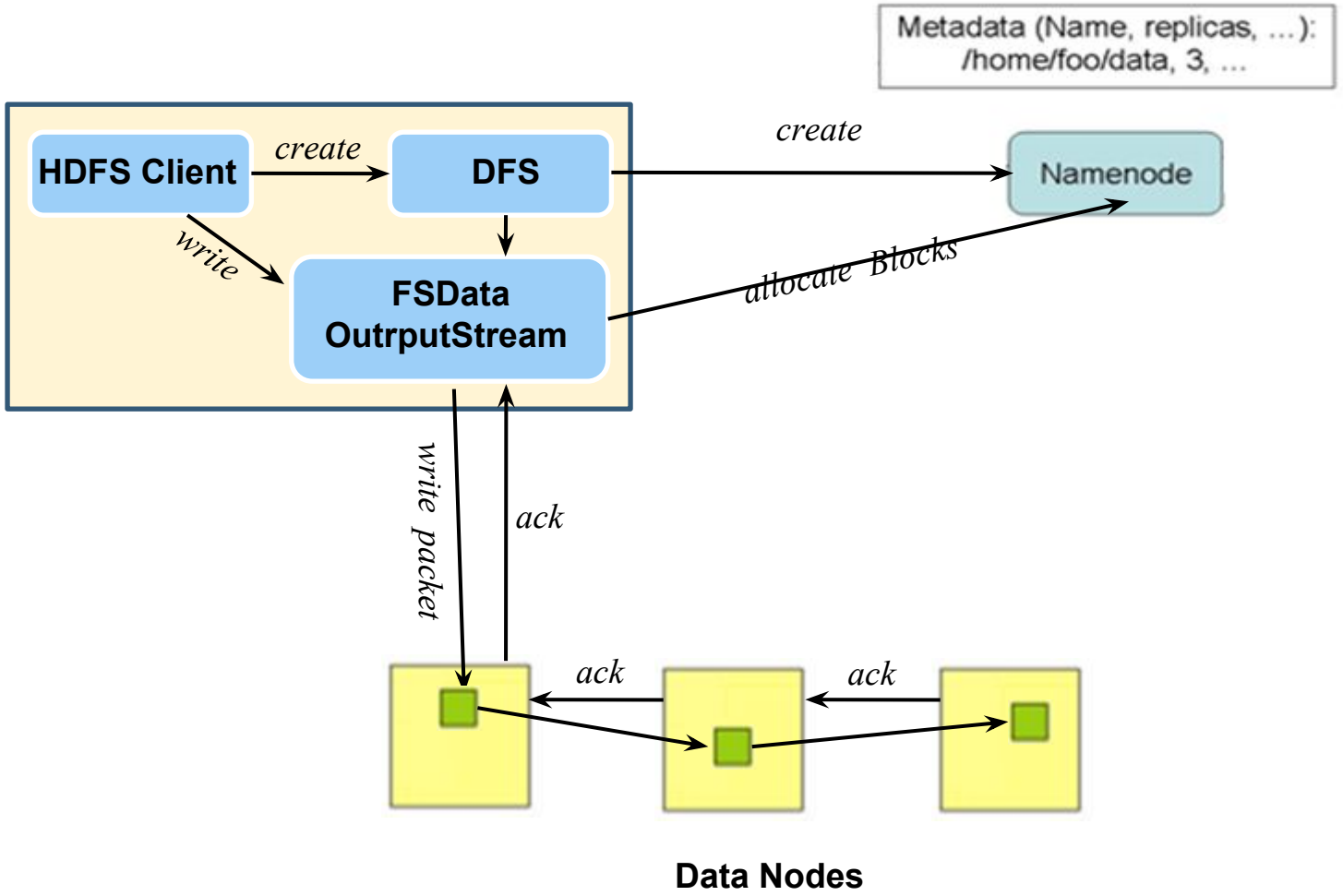


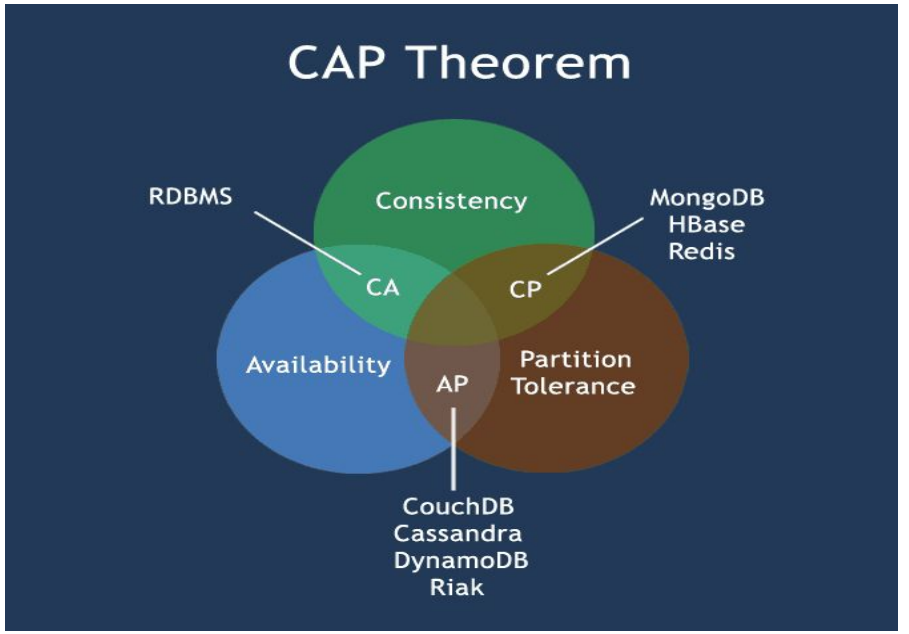
HDFS Read Operation




```
public class HDFSClient {
    public static void main(String args[]){
        Configuration conf = new Configuration();
        FileSystem dfs = FileSystem.get(conf);
        Path file = new Path("demo.txt");
            FSDataInputStream in = dfs.open(file)
        String data = in.readUTF();
        System.out.println(data);
        in.close();
        dfs.close();
    }
}
```

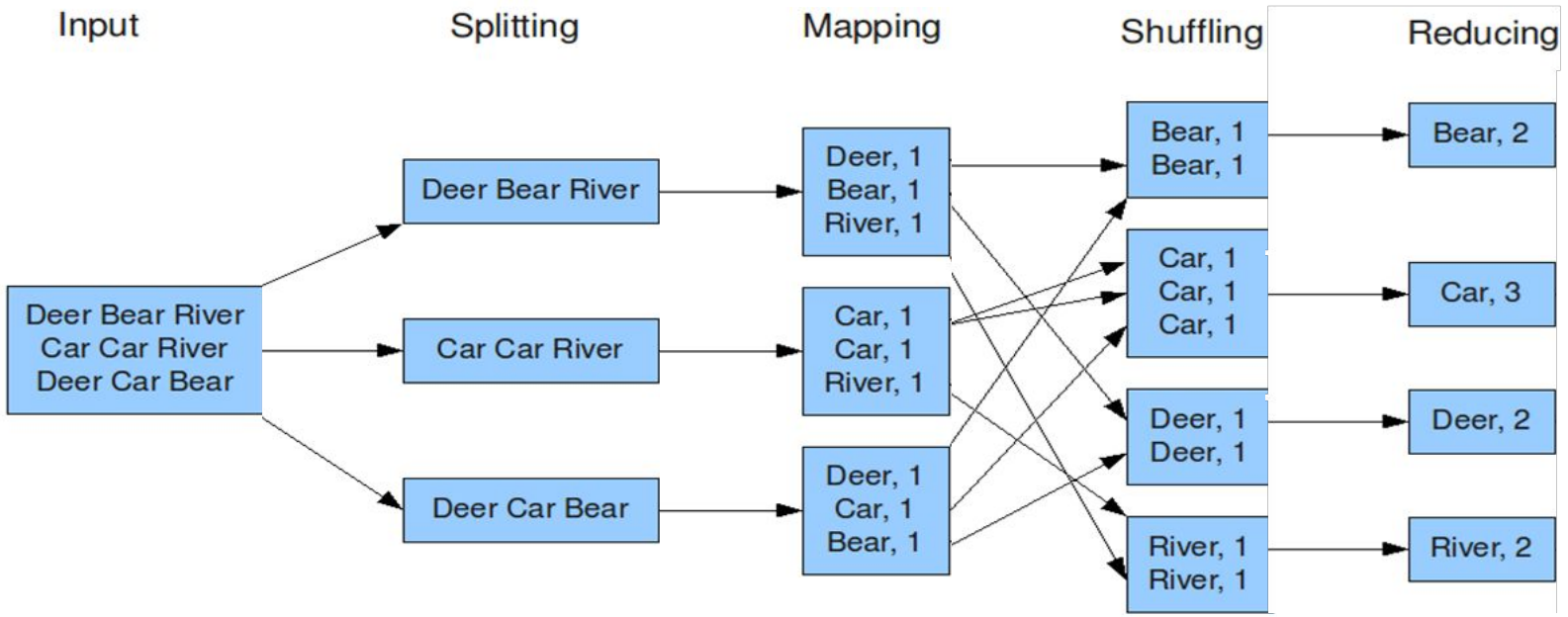
HDFS Write Operation





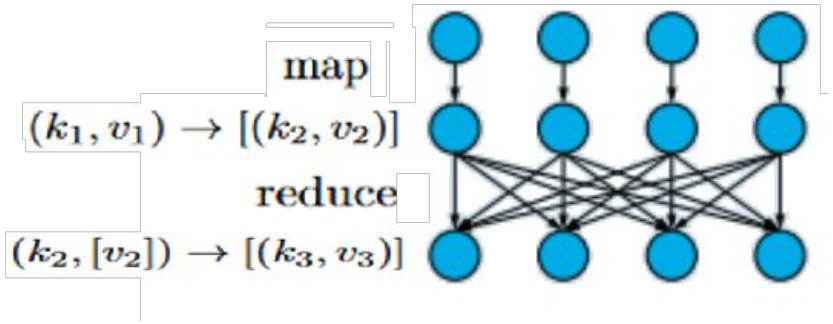

```
public class HDFSClient {
    public static void main(String args[]) {
        Configuration conf = new Configuration();
        FileSystem dfs = FileSystem.get(conf);
        Path file = new Path("demo.txt");
            FSDataOutputStream out = dfs.create(file);
        out.writeUTF("Welcome to HDFS using Java");
        out.close();
        dfs.close();
    }
}
```

Map Reduce for Distributed Processing



Overall complexity
= O(n/k)
= O(1) if k = O(n)

O(n/k) **O(n/k)** **O(n/k)**



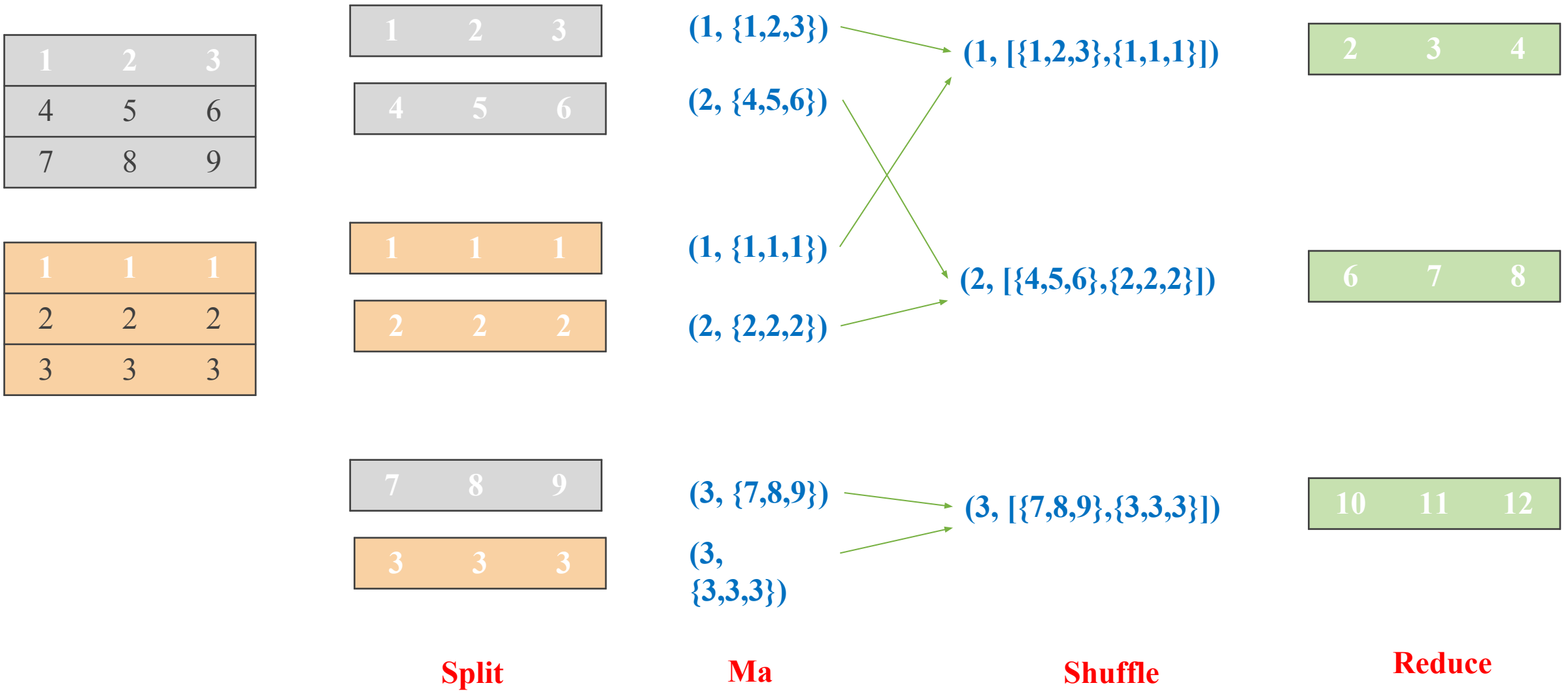
map: $(k_1, v_1) \rightarrow [(k_2, v_2)]$

```
class Mapper
    method Map(docid a, doc d)
        for all term t ∈ doc d do
            Emit(term t, count 1)
```

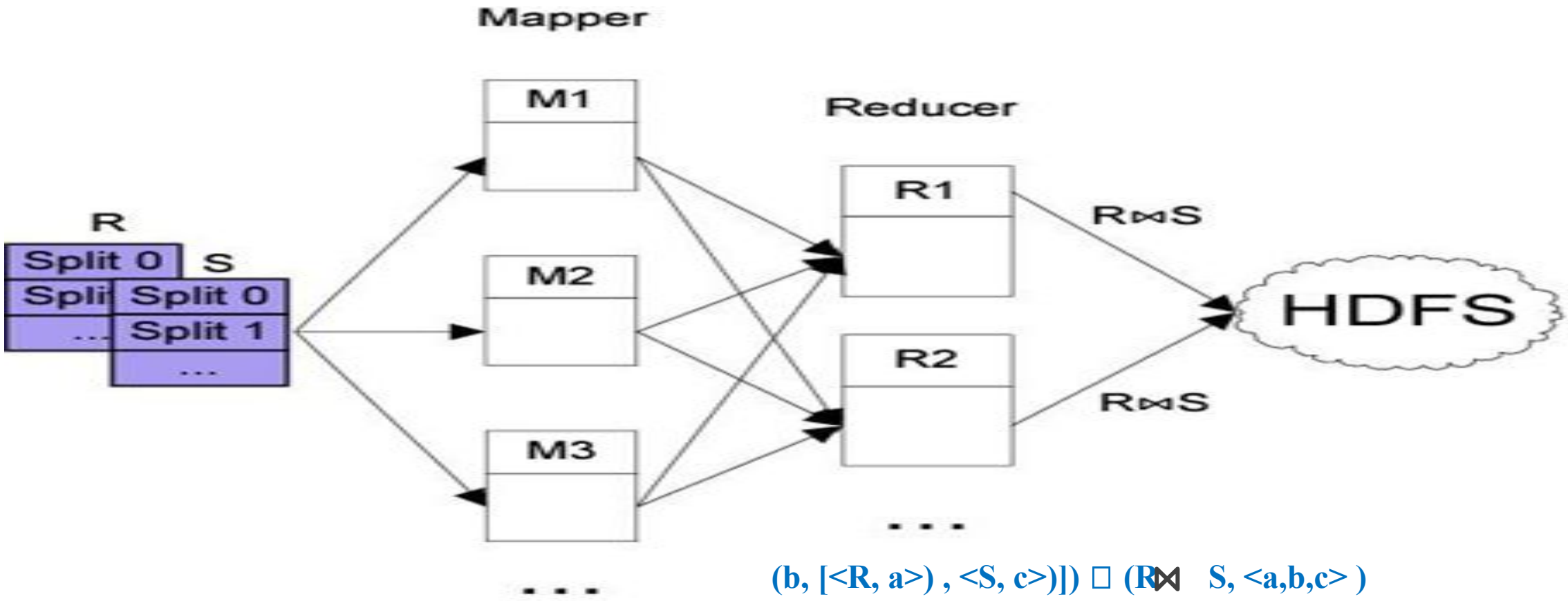
reduce: $(k_2, [v_2]) \rightarrow [(k_3, v_3)]$

```
class Reducer
    method Reduce(term t, counts [c1, c2, ...])
        sum ← 0
        for all count c ∈ counts [c1, c2, ...] do
            sum ← sum + c
        Emit(term t, count sum)
```

Data Analytics using Map Reduce : Matrix Addition



Data Analytics using Map Reduce : R(A,B)⋈ S(B,C)?

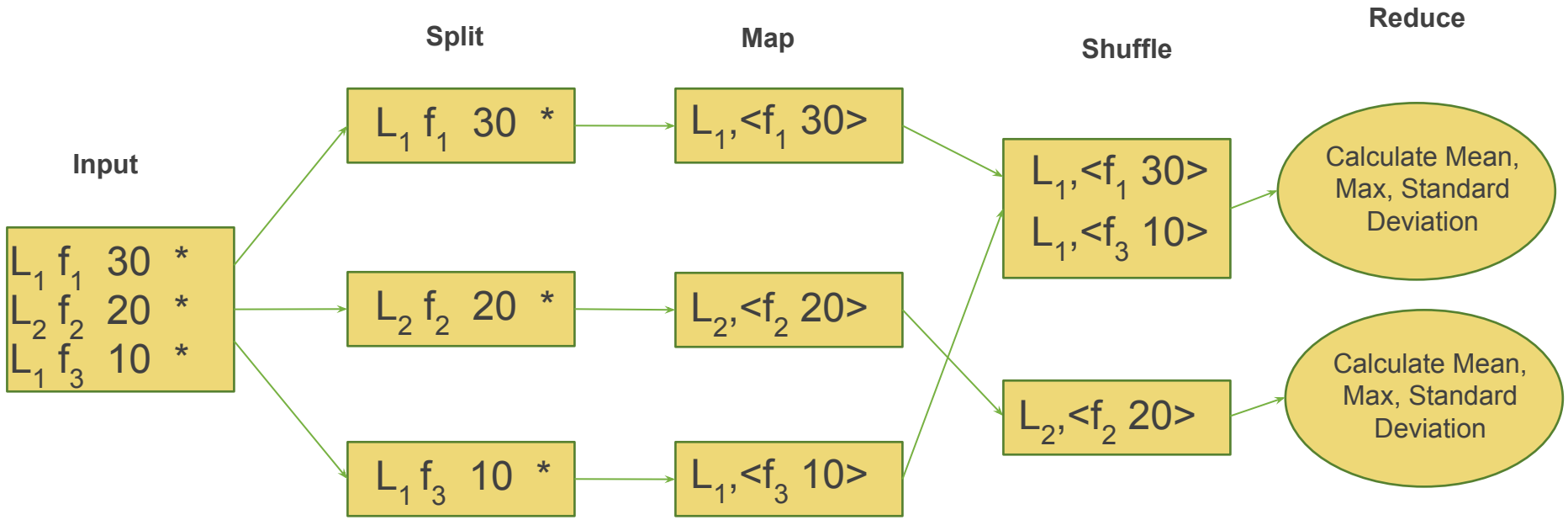


(b, [<R, a>) , <S, c>)]) □ (R⋈ S, <a,b,c>)

(R, <a,b>) □ (b, <R, a>)
(S, <b,c>) □ (b, <S, c>)

Data Analytics using Map Reduce : Group By Query

Find Mean, Max and Standard Deviation of the sizes of Files downloaded from NASA Server through each link.



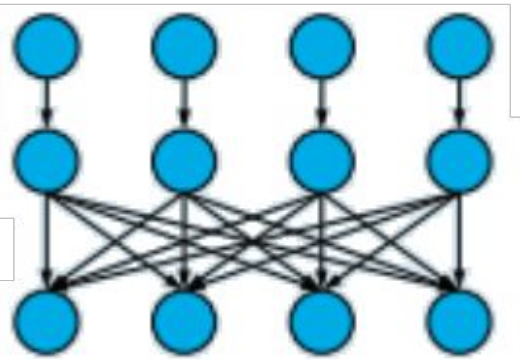
Data Analytics using Map Reduce : Single Pass vs Iterative

map

$(k_1, v_1) \rightarrow [(k_2, v_2)]$

reduce

$(k_2, [v_2]) \rightarrow [(k_3, v_3)]$

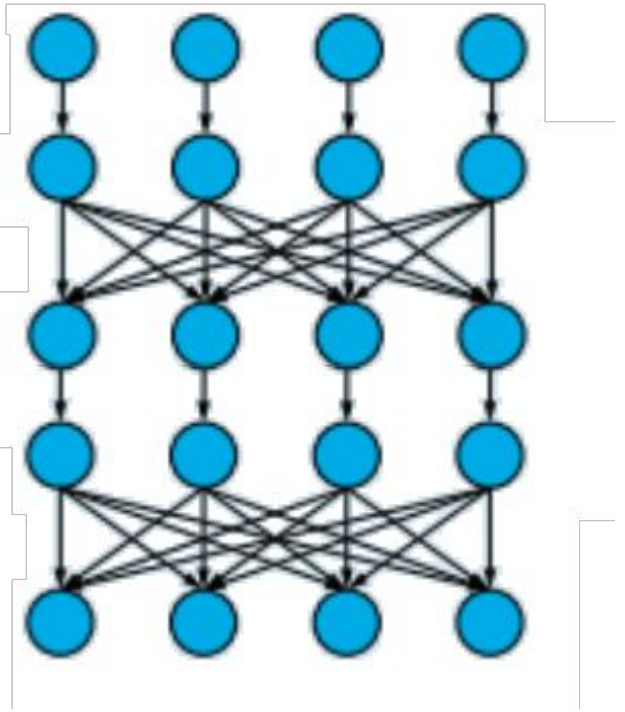


map

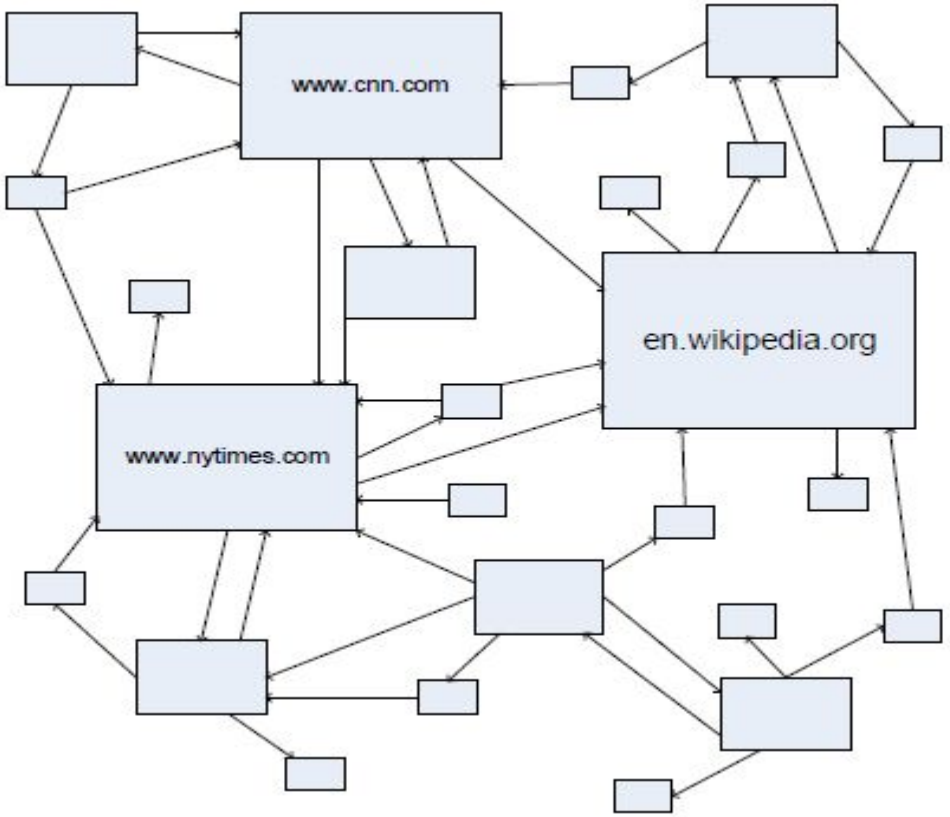
$(k_1, v_1) \rightarrow [(k_2, v_2)]$

reduce

$(k_2, [v_2]) \rightarrow [(k_3, v_3)]$

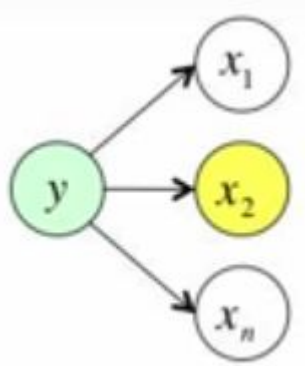


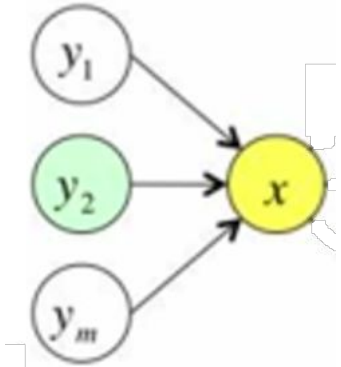
Data Analytics using Map Reduce : Page Rank Calculation



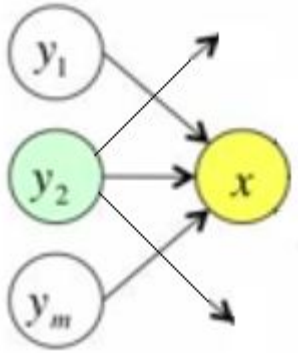
Mapper: $\langle y, \{x_1 \cdots x_n\} \rangle$

for j=1…n: emit $\left\langle x_j, \frac{PR(y)}{out(y)} \right\rangle$





$$PR(x) = \frac{1-\lambda}{N} + \lambda \sum_{y \to x} \frac{PR(y)}{out(y)}$$



Reducer: $\left\langle x, \left\{ \frac{PR(y_1)}{out(y_1)}, \cdots, \frac{PR(y_m)}{out(y_m)} \right\} \right\rangle$

$$PR(x) = \frac{1-\lambda}{N} + \lambda \sum_{y \to x} \frac{PR(y)}{out(y)}$$