# Stressing the Maintenance System of HDFS

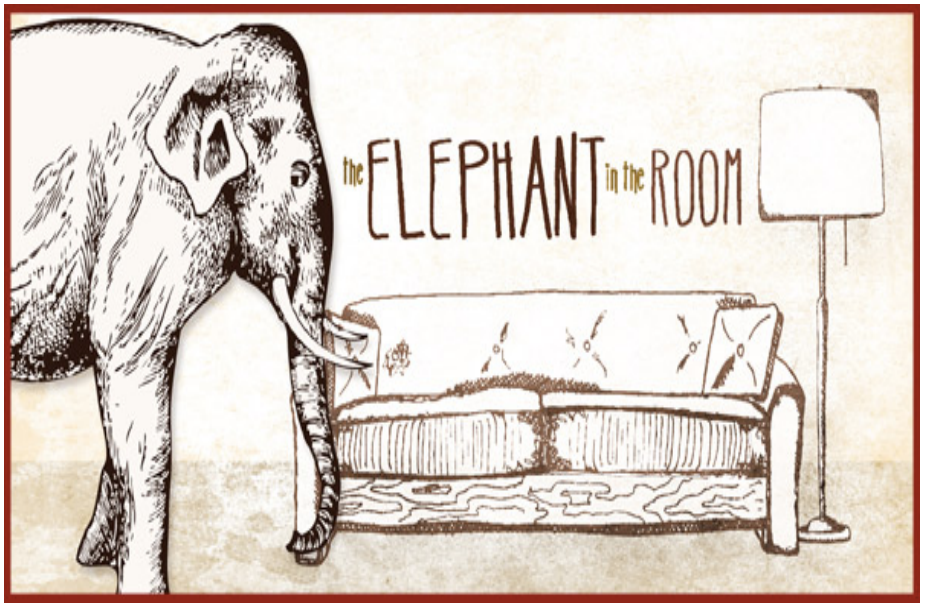
Sejal Chauhan, Arman Shanjani, Mihir Shete

# Stressing the Maintenance System of HDFS

- Zero to Hadoop
- Fail nodes in different scenarios
- Naïve ways of failure handling
- Model to analyze rate of failures
- Optimizing parameters:
  - Time to detect failure
  - Time to replicate data on failed nodes

# Stressing the Maintenance System of HDFS

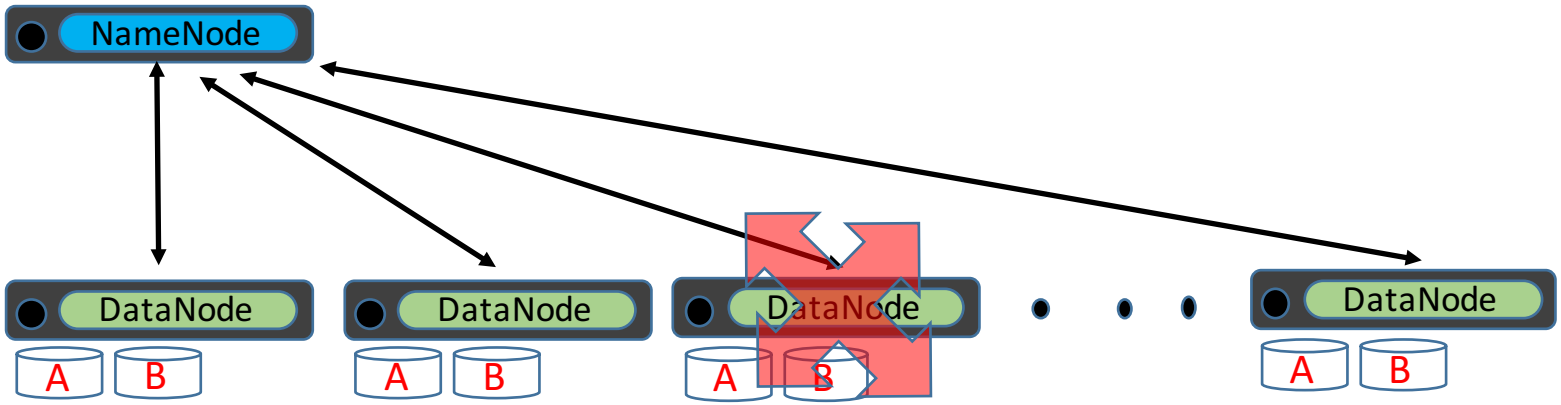
- **Zero to Hadoop**
- Fail nodes in different scenarios
- Naïve ways of failure handling
- Model to analyze rate of failures
- Optimizing parameters:
  - Time to detect failure
  - Time to replicate data on failed nodes



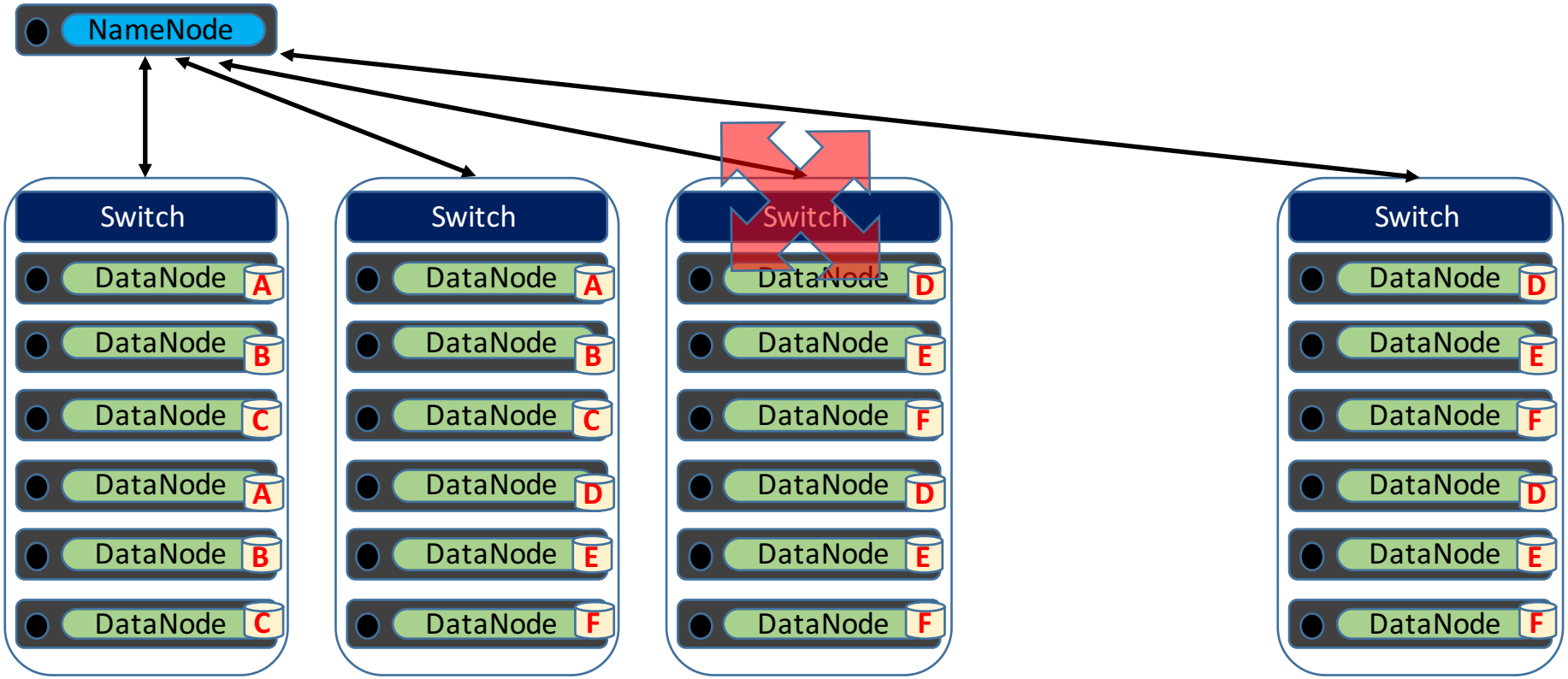
# Stressing the Maintenance System of HDFS

- Zero to Hadoop
- **Fail nodes in different scenarios**
- Naïve ways of failure handling
- Model to analyze rate of failures
- Optimizing parameters:
  - Time to detect failure
  - Time to replicate data on failed nodes

# Dealing with Node Failures

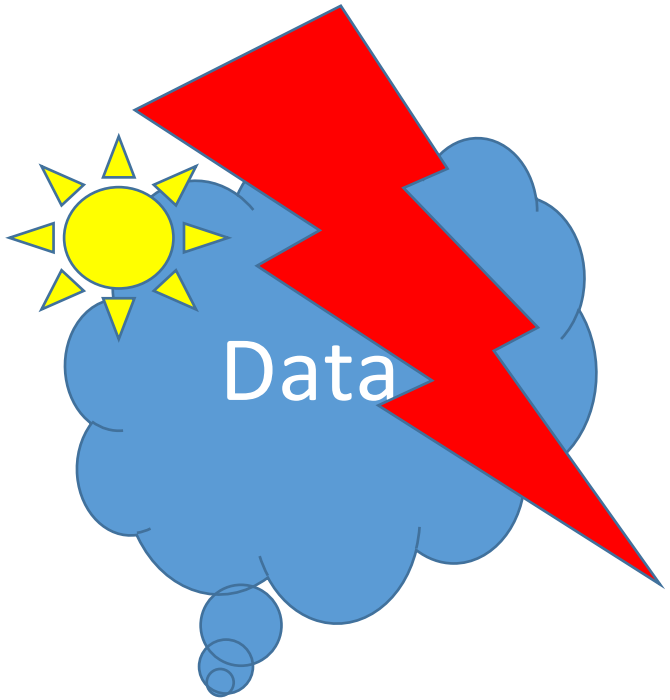


# Transient Failures



# Stressing the Maintenance System of HDFS

- Zero to Hadoop
- Fail nodes in different scenarios
- **Naïve ways of failure handling**
- Model to analyze rate of failures
- Optimizing parameters:
  - Time to detect failure
  - Time to replicate data on failed nodes

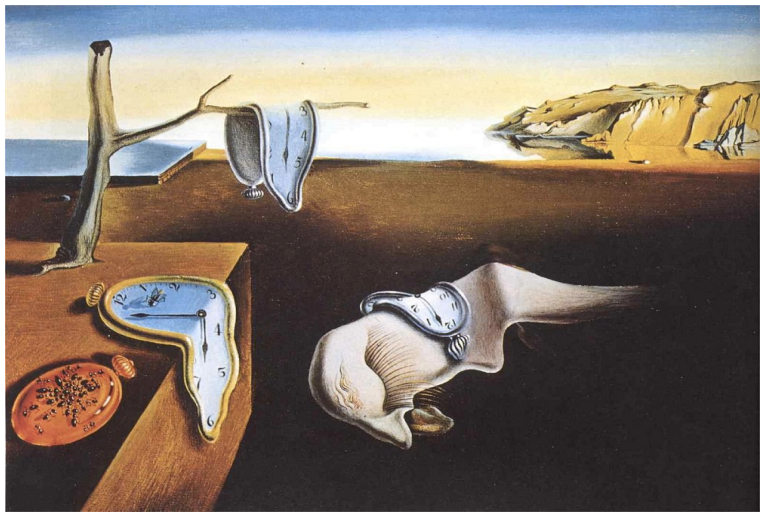


# Stressing the Maintenance System of HDFS

- Zero to Hadoop
- Fail nodes in different scenarios
- Naïve ways of failure handling
- **Model to analyze rate of failures**
- Optimizing parameters:
  - Time to detect failure
  - Time to replicate data on failed nodes

# Stressing the Maintenance System of HDFS

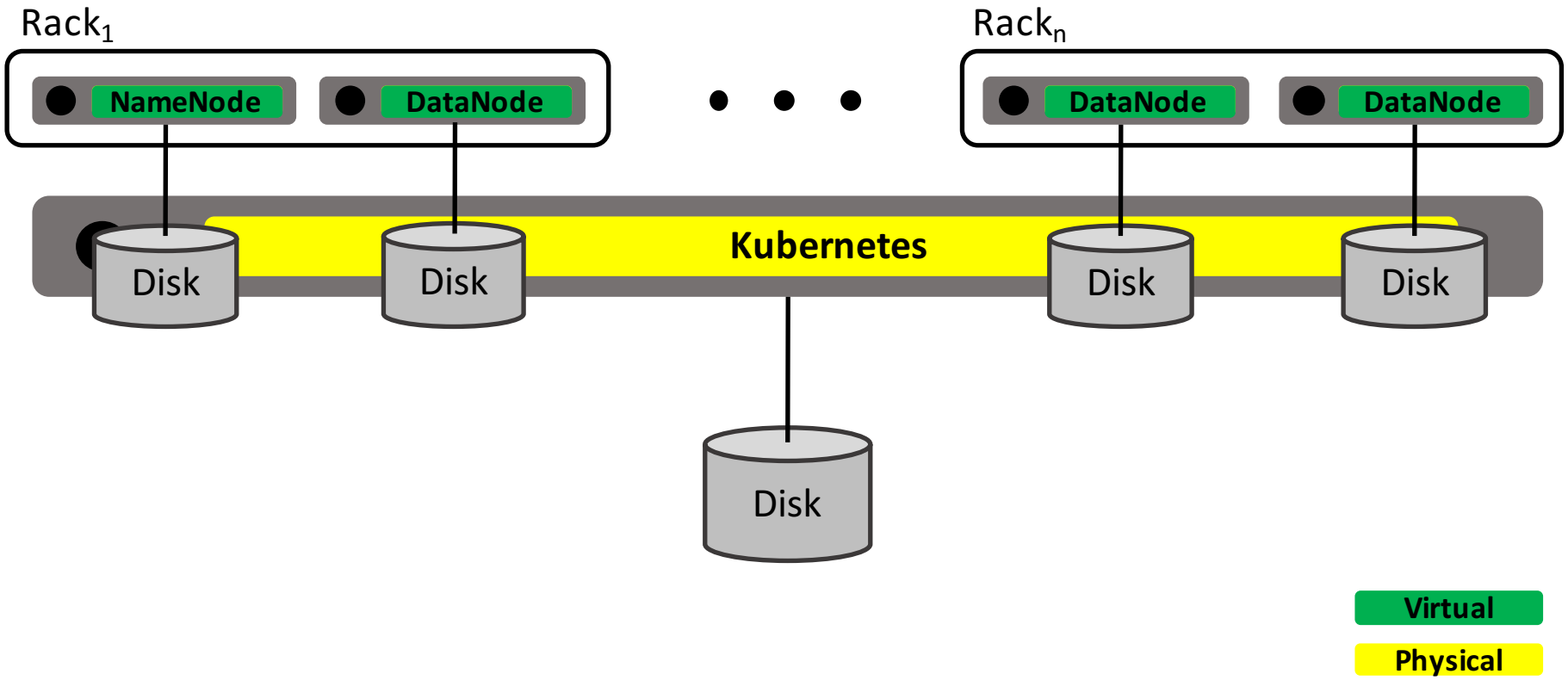
- Zero to Hadoop
- Fail nodes in different scenarios
- Naïve ways of failure handling
- Model to analyze rate of failures
- **Optimizing parameters:**
  - Time to detect failure
  - Time to replicate data on failed nodes



# Agenda

- Our automated scalable framework
- Experiments and results
- Model for rate of failures
- Improvements
- Summary
- Conclusion

# Zero to Hadoop

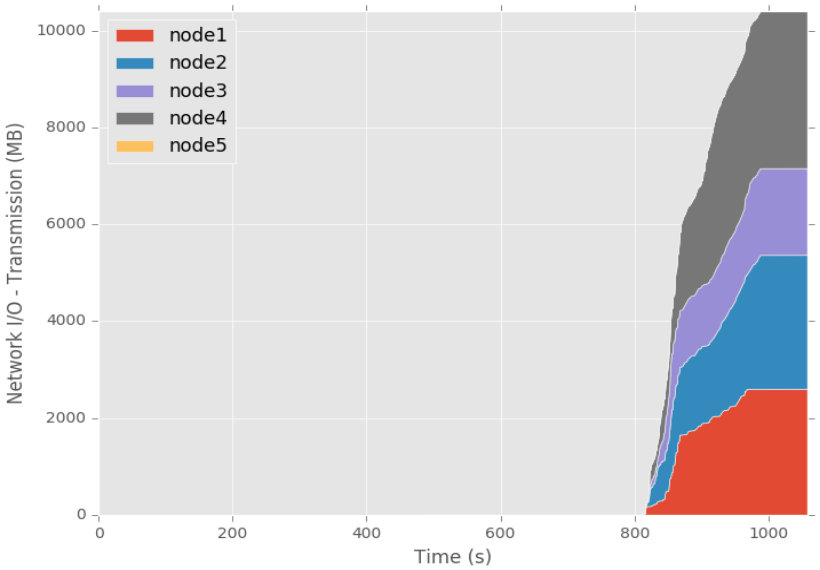


# Experiment #1
# Single DataNode Failure

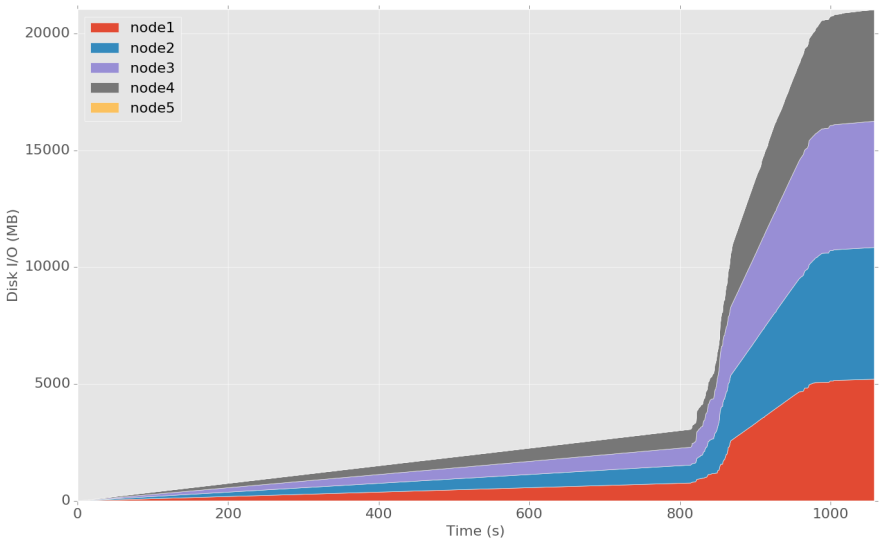
- Hadoop cluster of 5 VMs
- Replication factor: 3
- Size of data: 20GB (without replication)
  - ~12GB per VM
- Experiment: fail 1 DataNode
- Question: How much resources does the maintenance system consume?

# Experiment #1

Network Utilization vs. Time




Disk Utilization vs. Time




Network Traffic generated during maintenance: **10717MB**

Disk Reads during maintenance: **10717MB**

Disk Writes during maintenance: **10352MB**

# Experiment #2
# Effect of Single DataNode Failure on Workload

- Hadoop cluster of 5 VMs
- Replication factor: 3
- Size of data: 20GB (without replication)
  - ~12GB per VM
- Experiment:
  1. fail 1 DataNode
  2. **start a workload (TPC-DS Query #71) as soon as DataNode has failed**
- Question: What is the affect of failed node on workload?

# Experiment #2
# Effect of Single DataNode Failure on Workload

- Performance hit: takes **9%** more execution time
- Causes of performance degradation:
  - Less parallelism for executing MR jobs (execution and data access)
  - Accessing failed replicas first

# Experiment #2.5
# Effect of Re-replication of Failed Node on Workload

- Hadoop cluster of 5 VMs
- Replication factor: 3
- Size of data: 20GB (without replication)
  - ~12GB per VM
- Experiment:
  1. fail 1 DataNode
  2. **wait until maintenance begins**
  3. start a workload (TPC-DS Query #71)
- Question: What is the affect of maintenance work on workload?

# Experiment #2.5
# Effect of Re-replication of Failed Node on Workload

- Performance hit: takes **50%** more execution time
- Causes of performance degradation:
  - Less parallelism for executing MR jobs (execution and data access)
  - Accessing failed replicas first
  - **Resource contention**

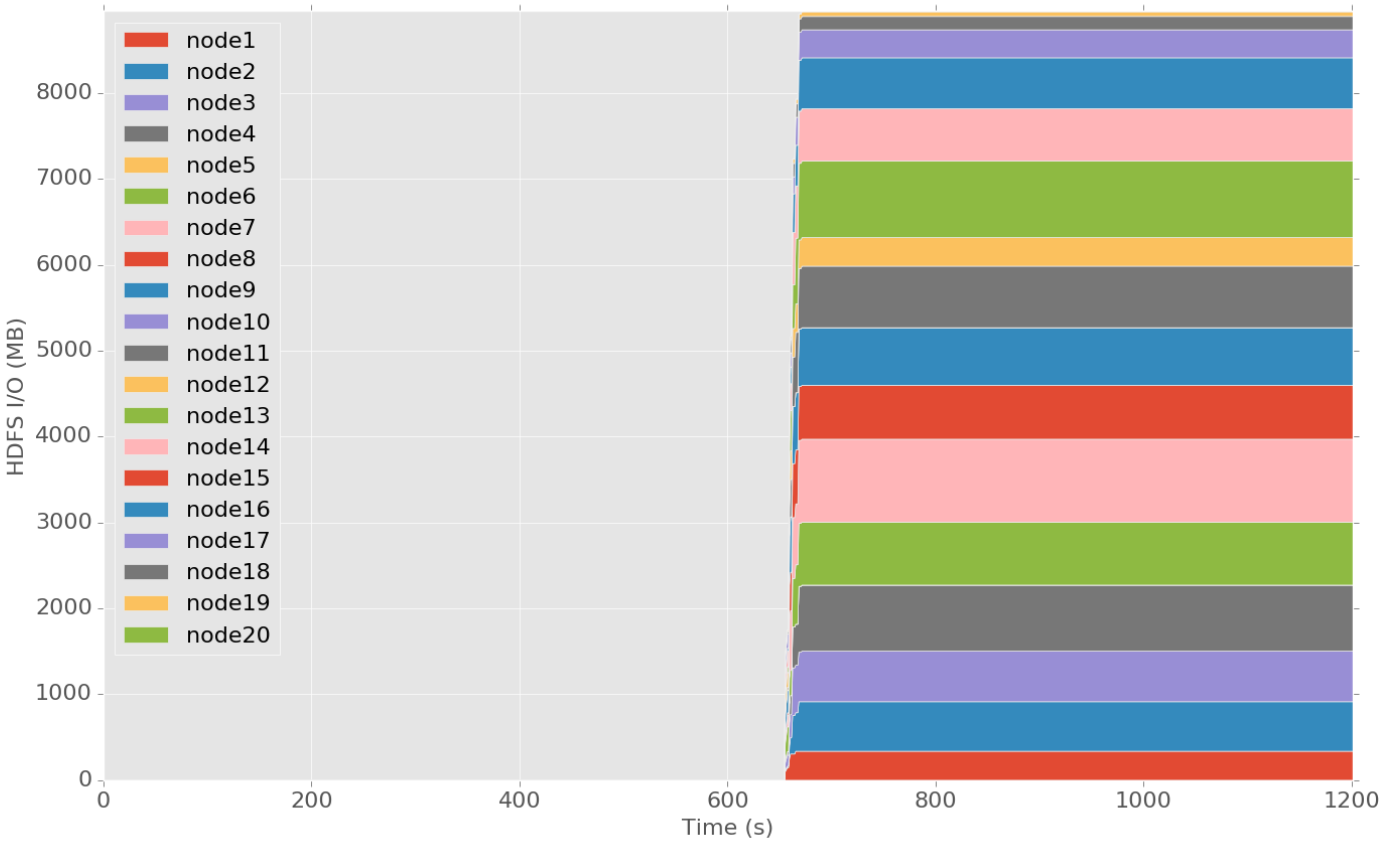
# Can the Maintenance Work be Avoided?

- Data block failures:
  - Permanent:
    - Node failed (fail-stop)
    - Block failed
      - Damaged/corrupted
      - Recognized by checksum
      - Immediately added to the replication queue
  - Transient:
    - Period of unavailability
    - Possibly stale when available again
    - Primarily caused by
      - Network partitions (e.g. ToR failures)
      - Node reboots

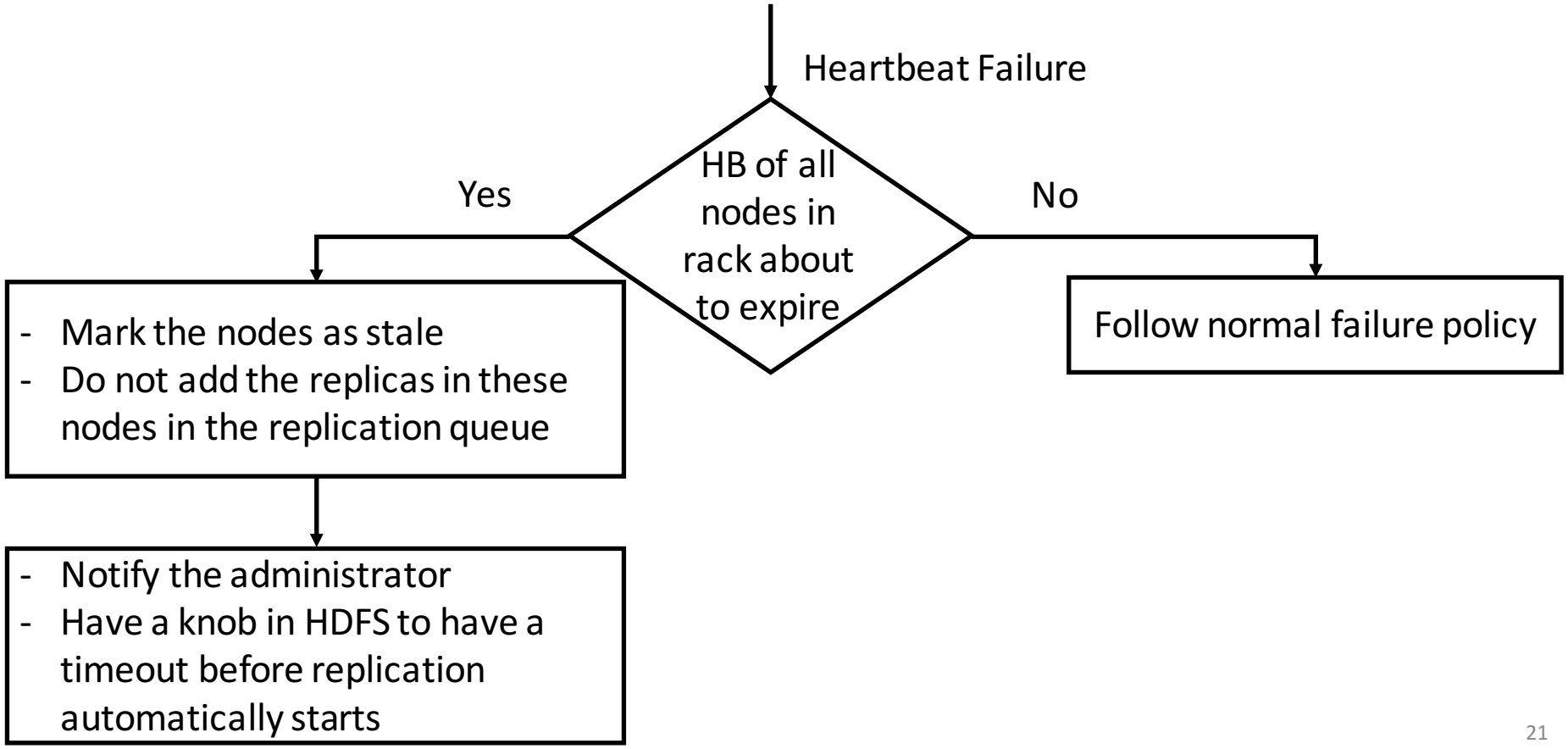
# Experiment #3
# Rack Failure

- Hadoop cluster of 20 containers (4 per rack)
- Replication factor: 3
- Size of data: 20GB (without replication)
  - ~12GB per rack
- Experiment:
  1. **fail 1 rack**
  2. Analyze Query 71 performance during maintenance
- Question: How much does maintenance affect performance?

# Experiment #3
# Rack Failure



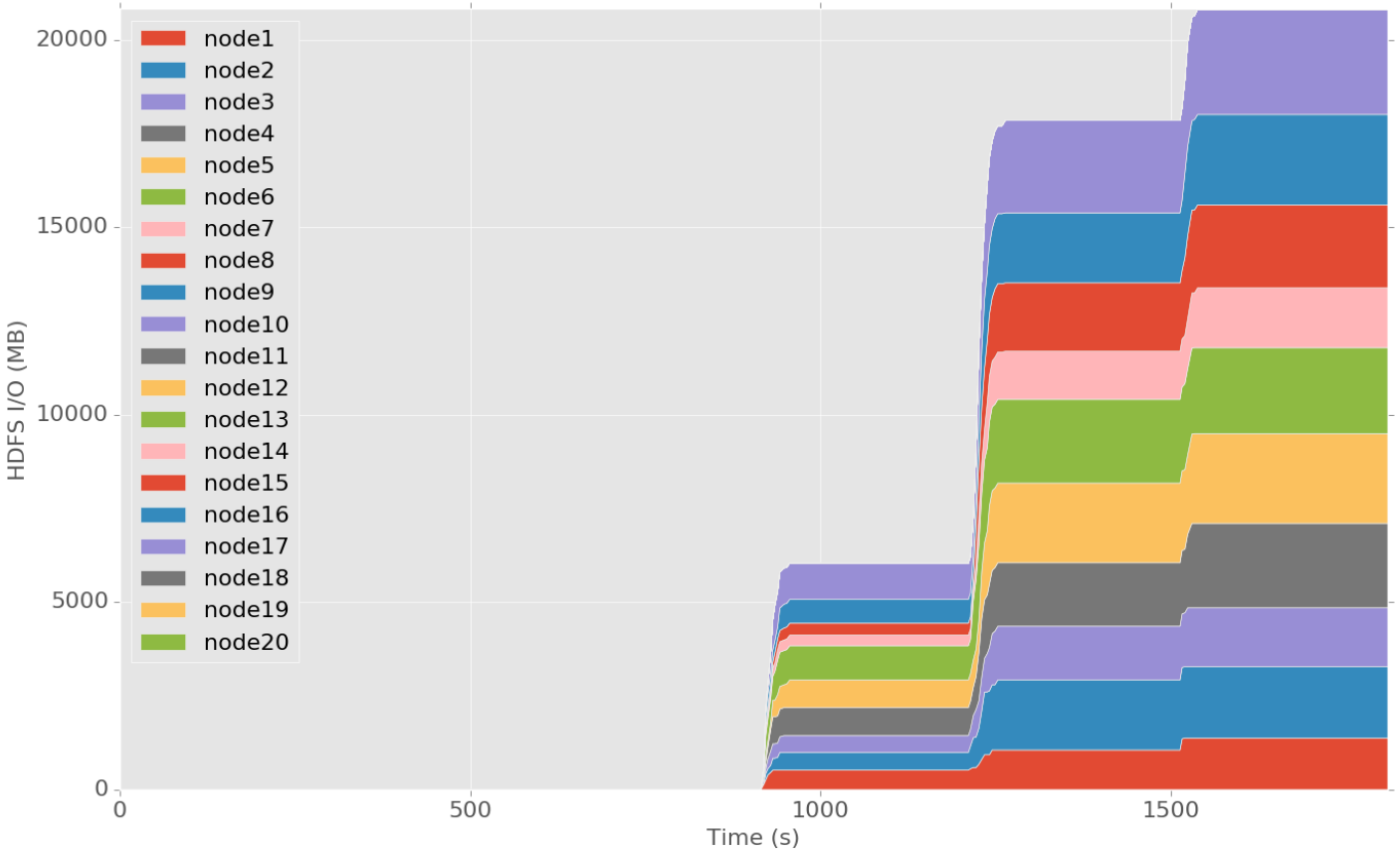
# An Adaptive Method to Handle ToR Failures



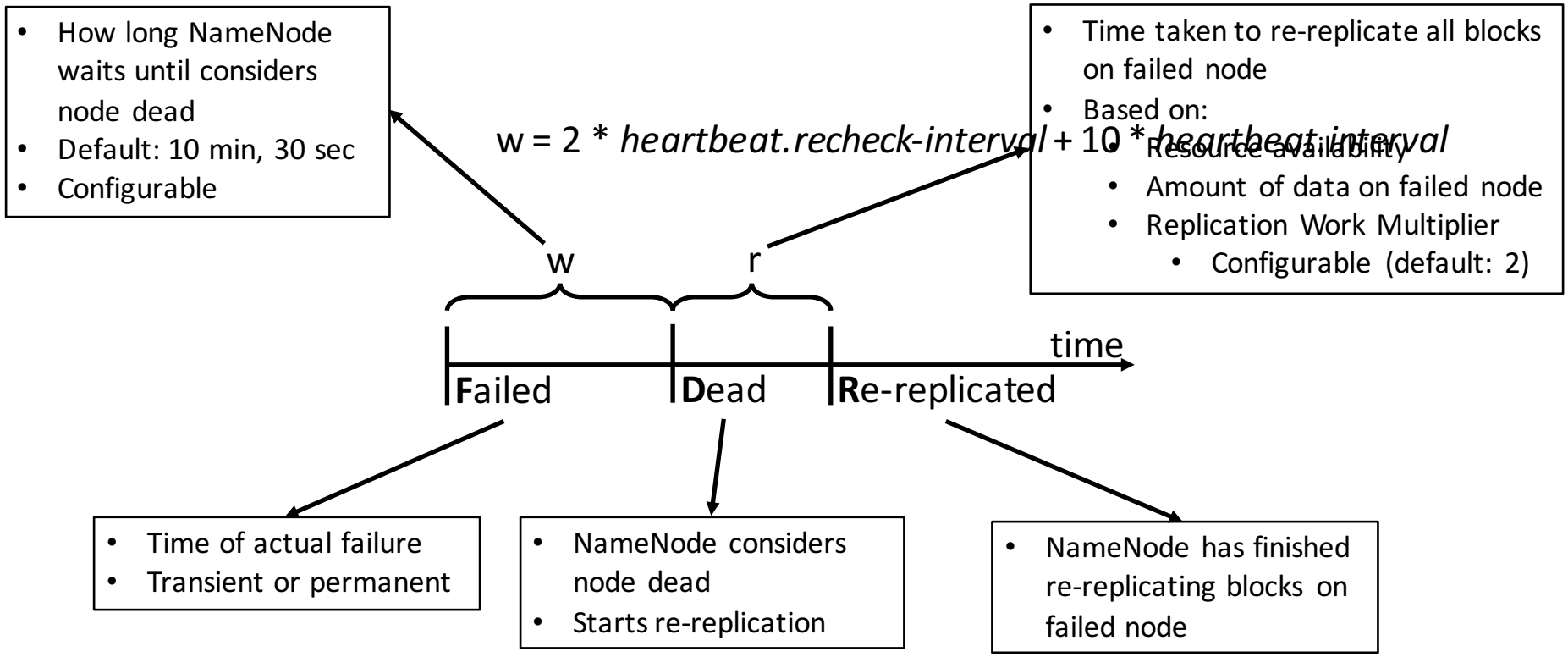
# Experiment #4
# Repeatedly Failing Nodes at a Rate

- Hadoop cluster of 20 containers
- Replication factor: 3
- Size of data: 20GB (without replication)
  - ~3GB per node
- Experiment:
  1. fail 1 node every 1 minute
- Question: What does HDFS do under higher rates of failure?

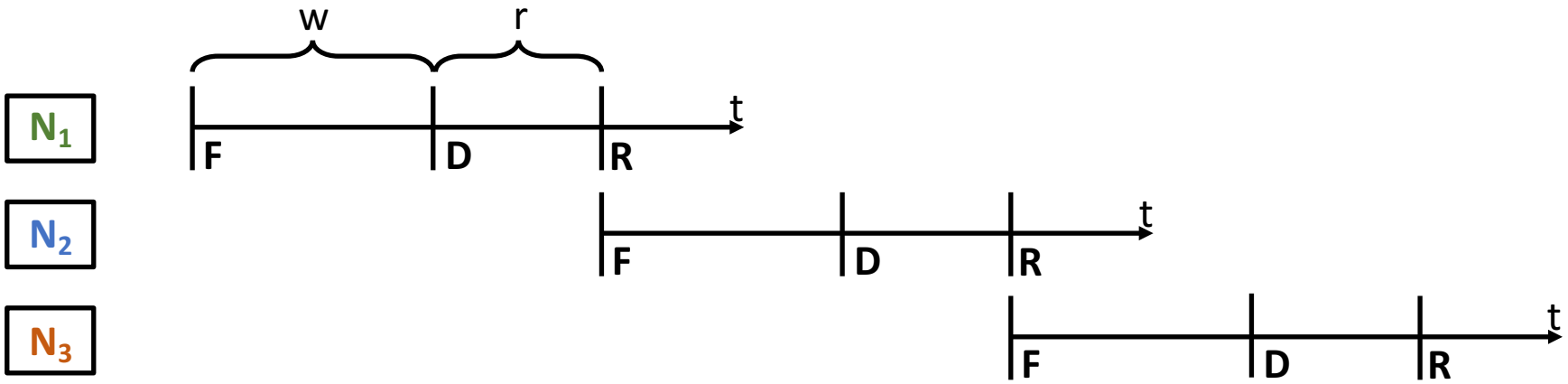
# Experiment #4
# Repeatedly Failing Nodes at a Rate

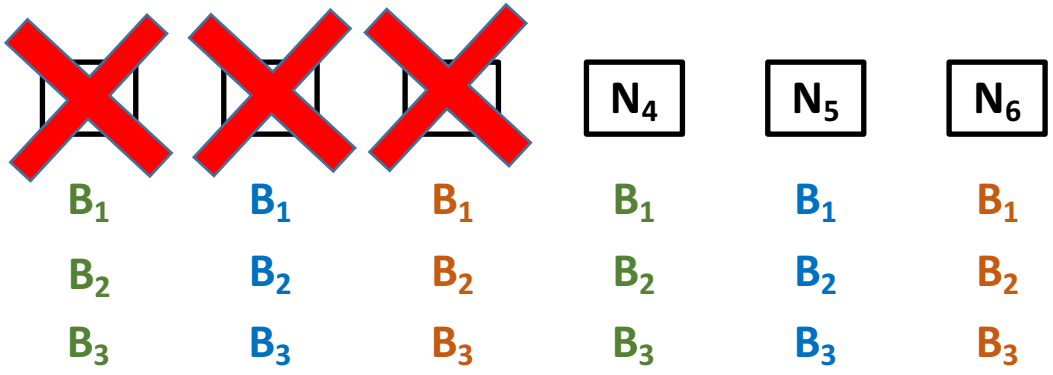


# F-D-R Model

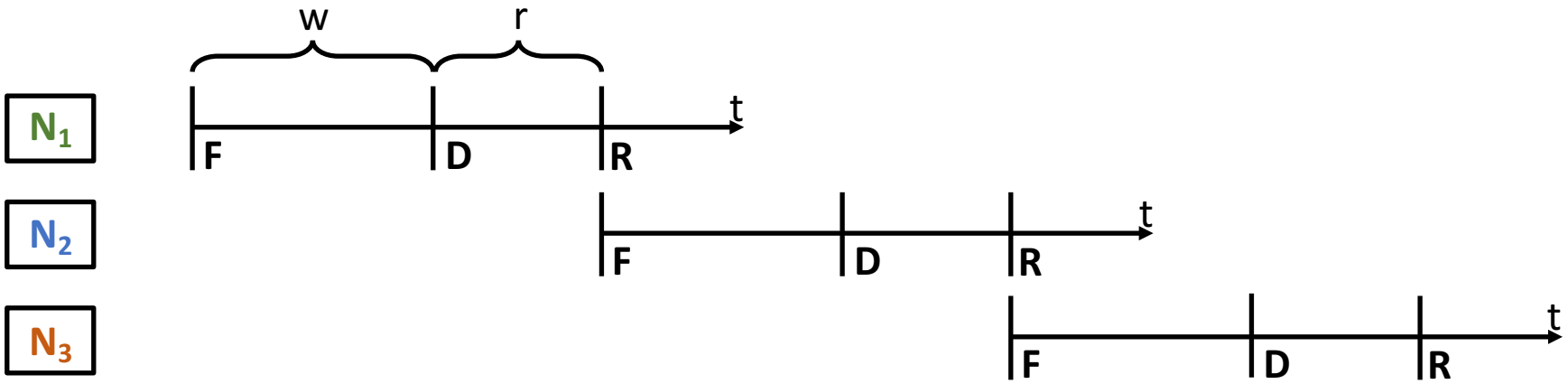

How long NameNode waits until considers node dead
Default: 10 min, 30 sec
Configurable
w = 2 * heartbeat.recheck-interval + 10 * heartbeat.interval
Time taken to re-replicate all blocks on failed node
Based on:
Resource availability
Amount of data on failed node
Replication Work Multiplier
Configurable (default: 2)
w
r
time
Failed
Dead
Re-replicated
Time of actual failure
Transient or permanent
NameNode considers node dead
Starts re-replication
NameNode has finished re-replicating blocks on failed node

# Acceptable Rate of Failure



# Acceptable Rate of Failure

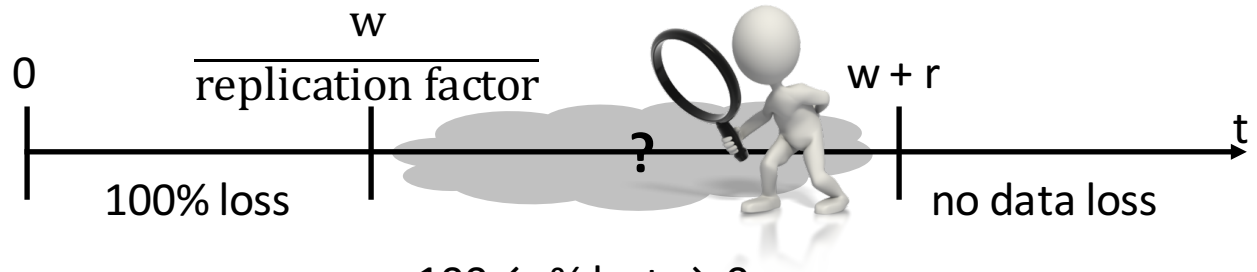


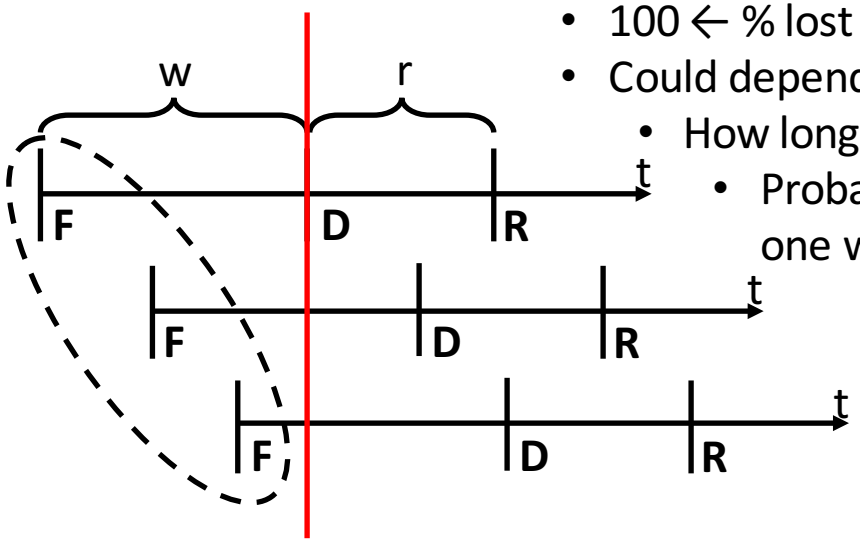

**Safe** if rate of failure is at least *every w + r*

$$\mathbf{w + r \leq f}$$

f is interval between each failure

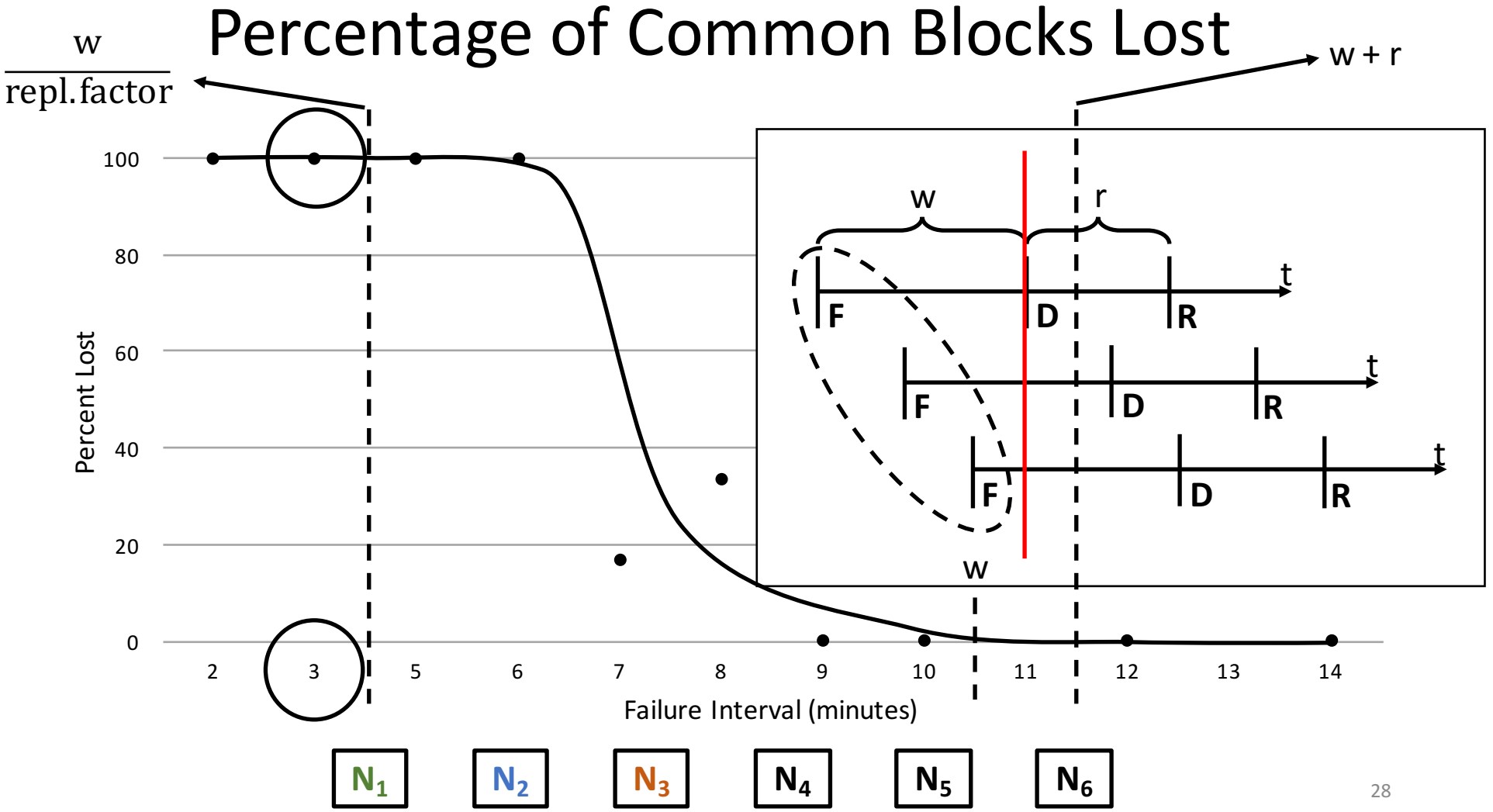
# Failure Interval Spectrum




- 100 ← % lost → 0
- Could depend on:
  - How long before the last replication is lost as well
  - Probability that the next node failing will be the one with the last replication

# Percentage of Common Blocks Lost

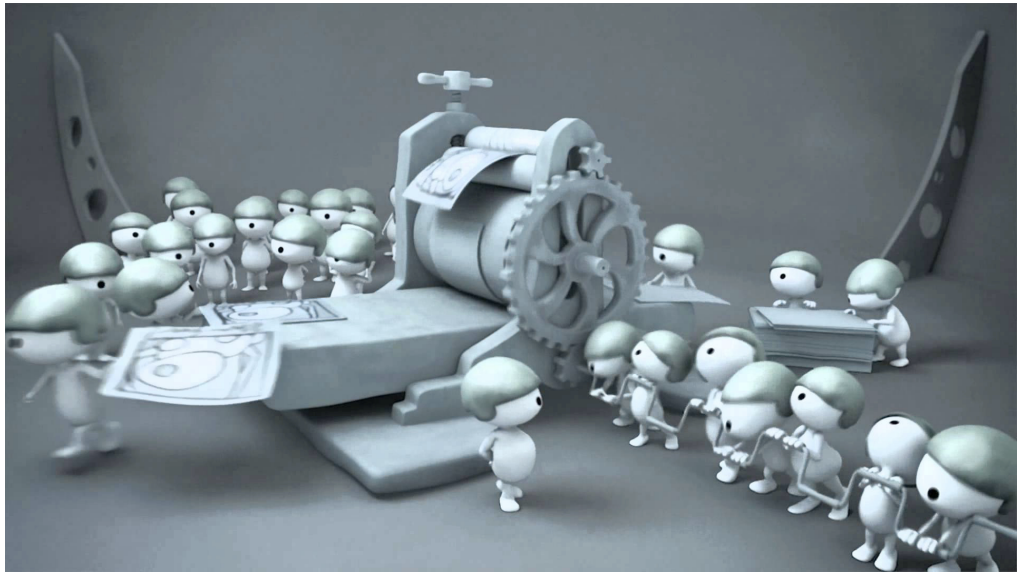


# Improvements:
# Optimizing Failure Detection Latency

1. w should be dynamically settable per node(s)
2. Hadoop to dynamically change it based on detecting failure behaviors (e.g. ToR failure)

# Improvements:
# Reducing Time Taken to Re-replicate

1. Leveraging balancer to perform periodic data distribution
2. Dynamically changing Replication Work Multiplier



# Balancer

- Configurable:
  - Cap the Network usage
  - Threshold
- Threshold: Tolerance between the % used blocks in
  - entire cluster
  - unbalanced node

| Threshold (%) | Time Taken to Balance (s) |
|---|---|
| 10 | 6300 |
| 15 | 3750 |
| 20 | 1040 |

# Replication Work Multiplier

- Configurable
- With each heartbeat:
  - NameNode asks the DataNode to replicate X number of blocks
  - X = replication work multiplier * live nodes

| Replication Multiplier | Time taken to replicate (secs) |
|---|---|
| 2 | 185 |
| 5 | 166 |

# Summary

- Built a scalable framework to stress HDFS
- Measured the failure handling mechanism with variable rates of failure in different scenarios
- Found that current replication approach is not a resilient
  - Developed FDR model to understand and reason about the system

# Conclusion

- 1 policy for all scenarios
- Should improve HDFS to make it more dynamic
- Active community work in this space

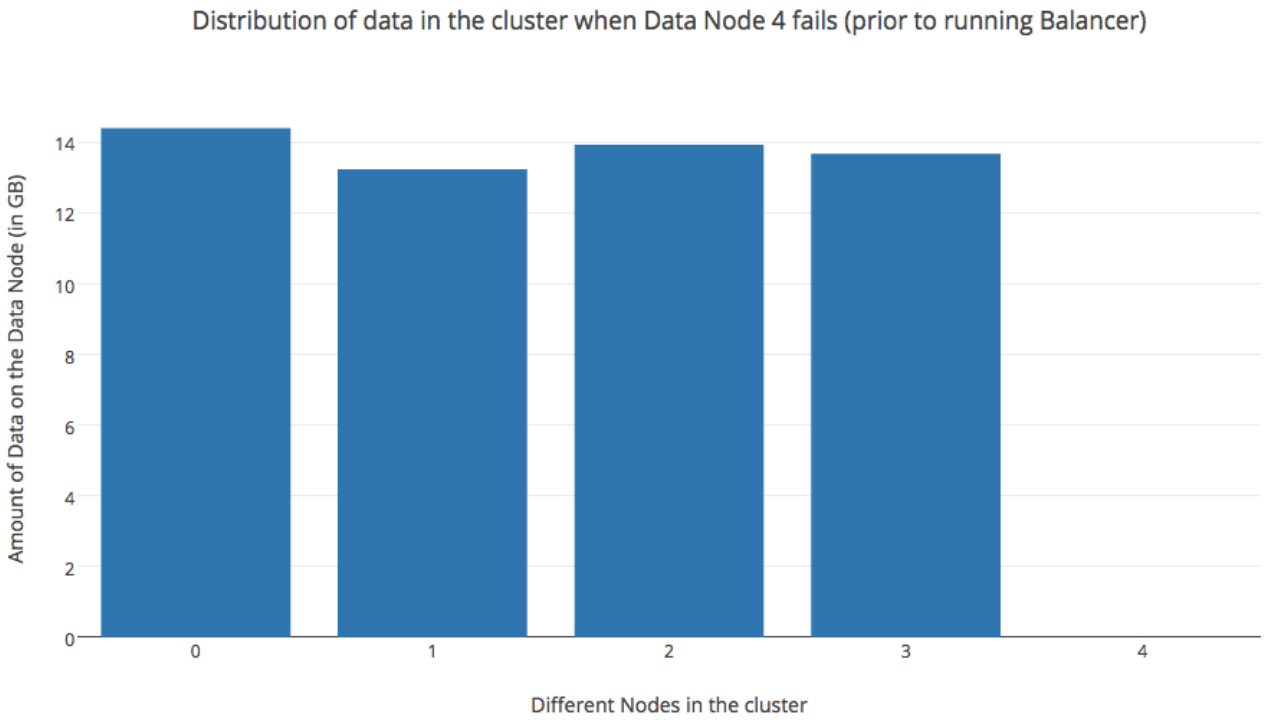
Questions?

Thoughts?

Wake up?

# Backup Slides

# When a Data Node fails…



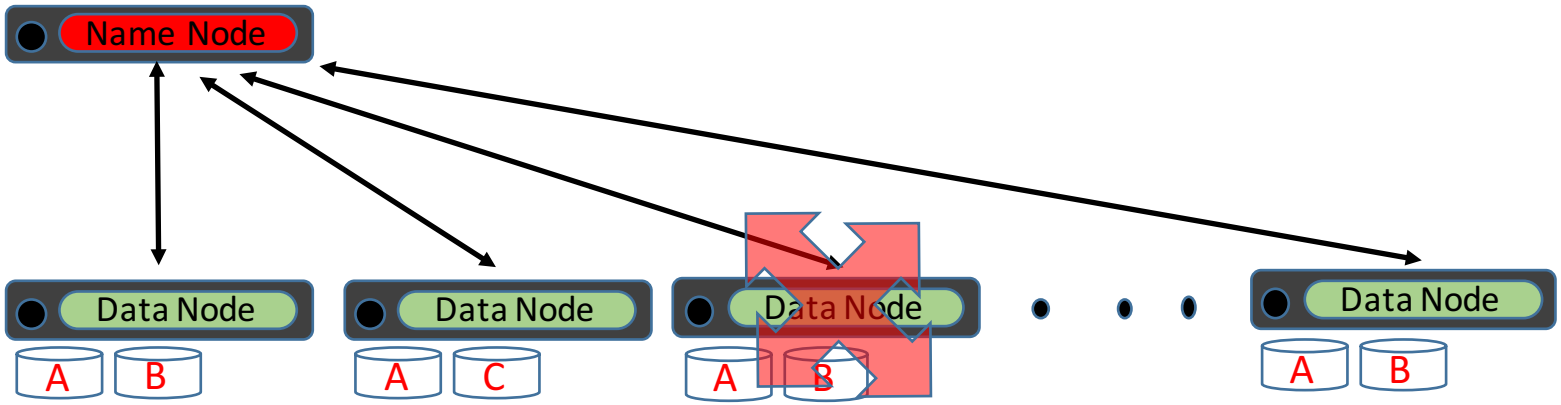
# When the Data Node is back on the Grid Balancer does its job

# HDFS Balancer: Balancing Data on Data Nodes

| Threshold % | Time taken | Iterations |
|---|---|---|
| 20 | 17.49m | 2 |
| 15 | 63.14m | 9 |
| 10 | 16.96m | 4 |
| 1 | | |

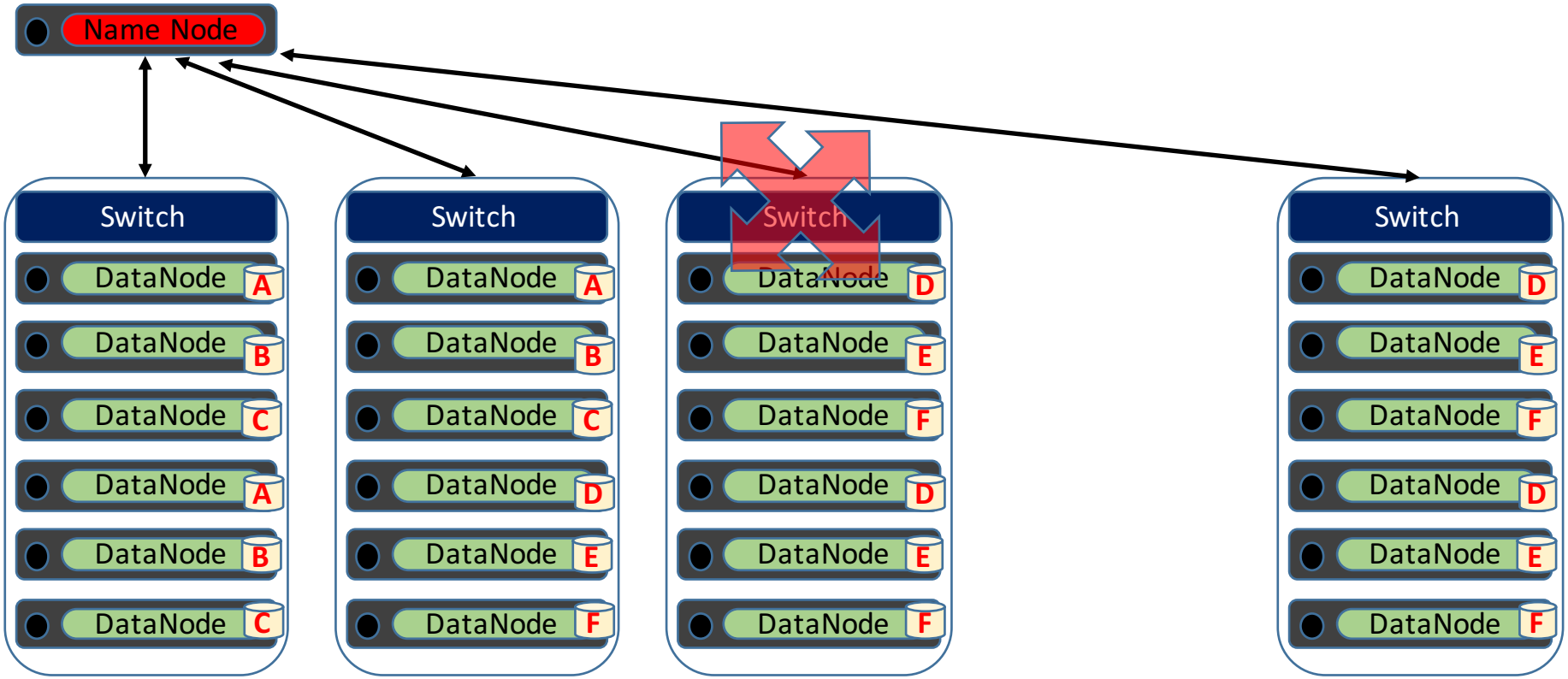
# Dealing with Node Failures

## Name Node and Data Nodes




- When a Data Node goes down
- Name Node detects failure after 10min
- Looks up the dead node's block report
- Requests other Data Nodes to replicate the affected blocks using heartbeat messages
- Over utilized

# Top of Rack Switch Failures

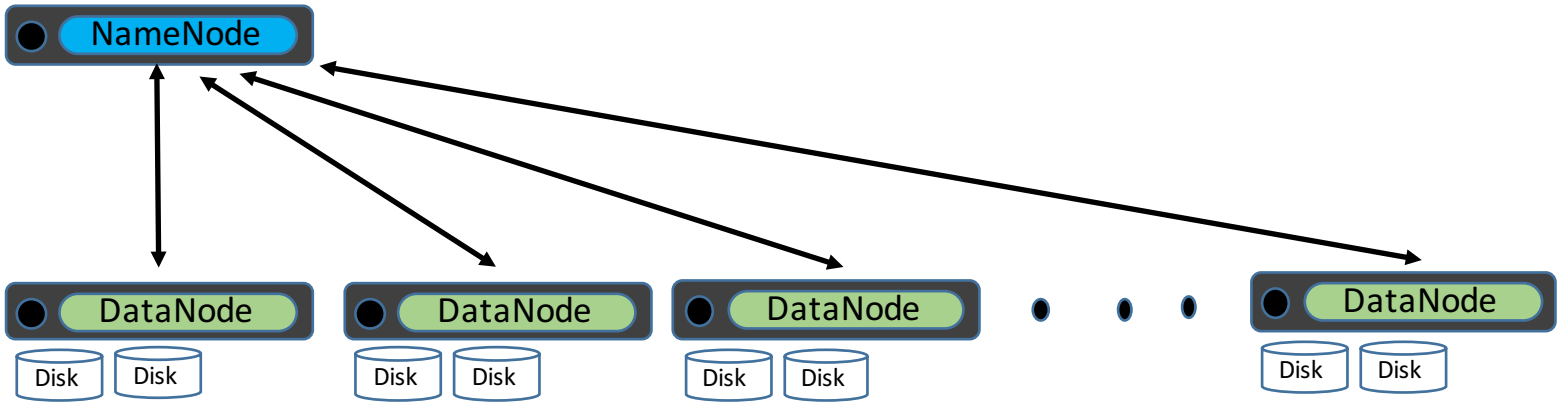


# Optimal Solution: Rate Adaptive

- Before
- Time to detect Failure + Time to replicate a Node * (Total Time/Min Time to Node failure) + Time to replicate a Rack * (Total Time/Min Time to Rack Failure)

After
Optimized Time to detect Failure + A*Time to replicate a Node * (Total Time/Min Time to Node failure) + B*Time to replicate a Rack * (Total Time/Min Time to Rack Failure) + (1-B) *Time to replicate a Rack * (Total Time/Min Time to Rack Failure) + (A)*Time to run Balancer for reduced replication time
Where B- probability of ToR failure recovery

# Brief Overview of HDFS




- DataNode sends **heartbeat every 3 seconds** to NameNode
- By default, **block report** is sent every **6hrs** to the NameNode
- NameNode builds **metadata** from **block reports**
- Most clusters have a **replication factor of 3**
- If **NameNode** is **down**, the cluster is **unresponsive**