Introducing Riak

Tim.Tang

Riak Features

Fault-tolerant
Highly available
Low-latency
Key-Value
Eventual Consistency

Querying Riak

MapReduce
Secondary Index
Key Filters
Link Walking
Full Text Search

Keys To MapReduce Success

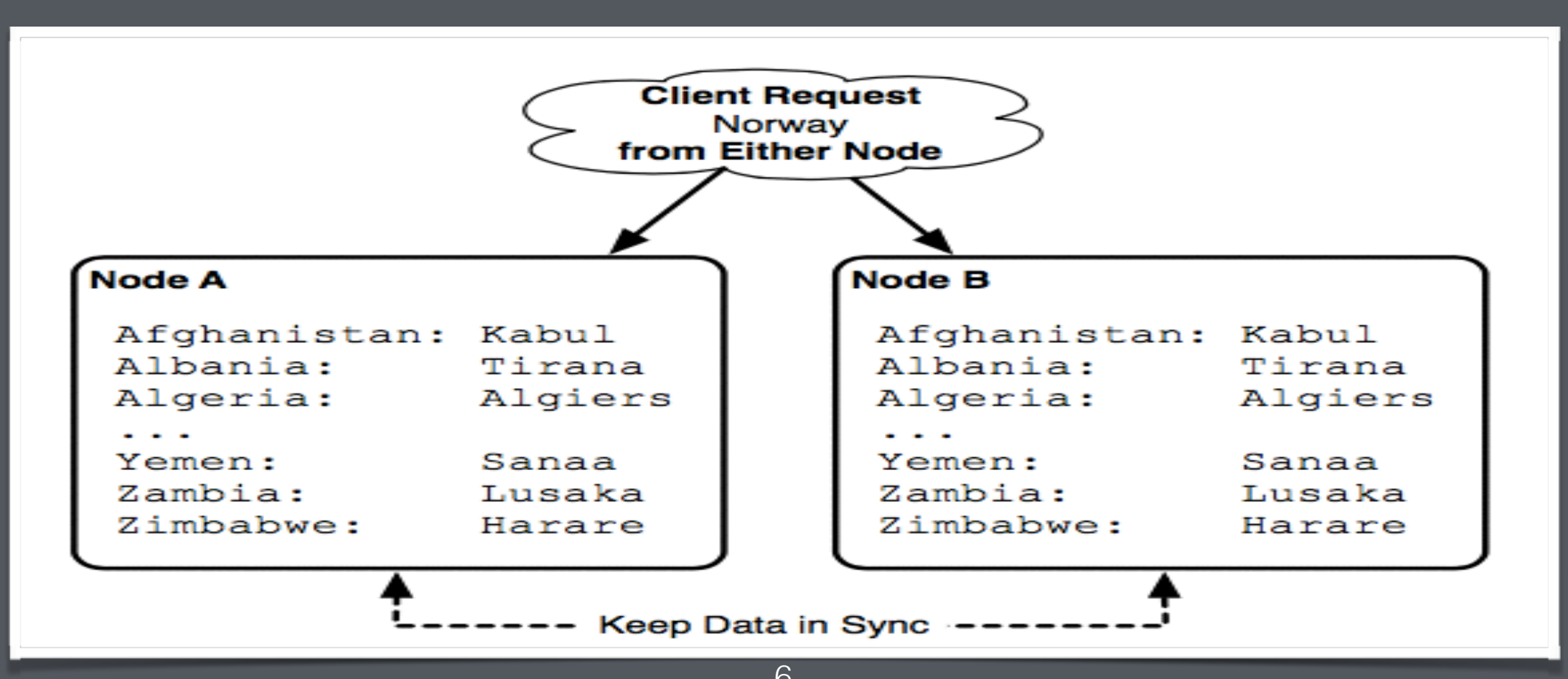
Avoid full-bucket Map
Convert slow queries to Erlang
Build a library of common functions
Structure keyspace well

Riak Metadata With Keys

Content-Type
Last-Modified
Link
X-Riak-Meta-*

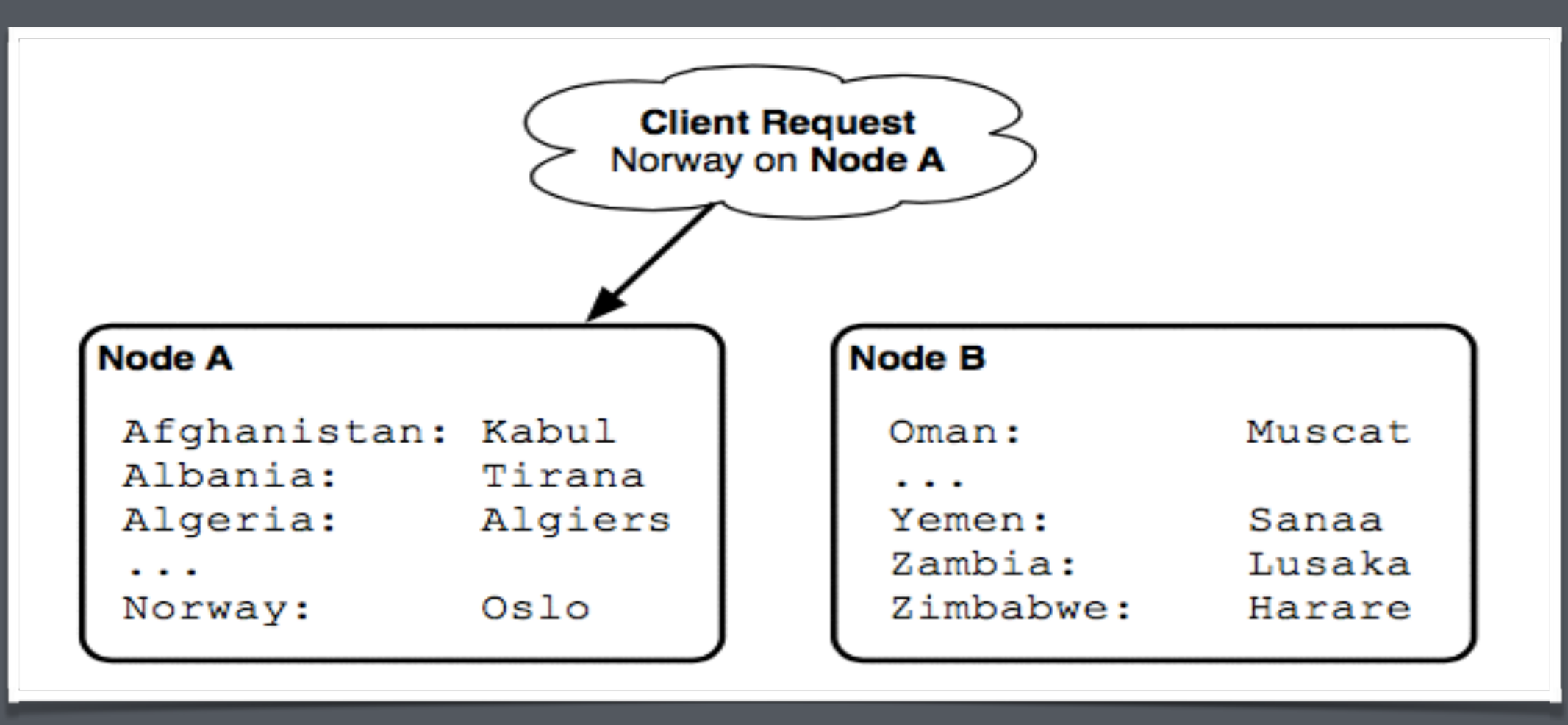
Replication

- Benefit side is that if one node goes down, nodes that contain replicated data remain available to serve requests.
- Downside with replication is that you are multiplying the amount of storage required for every duplicate.



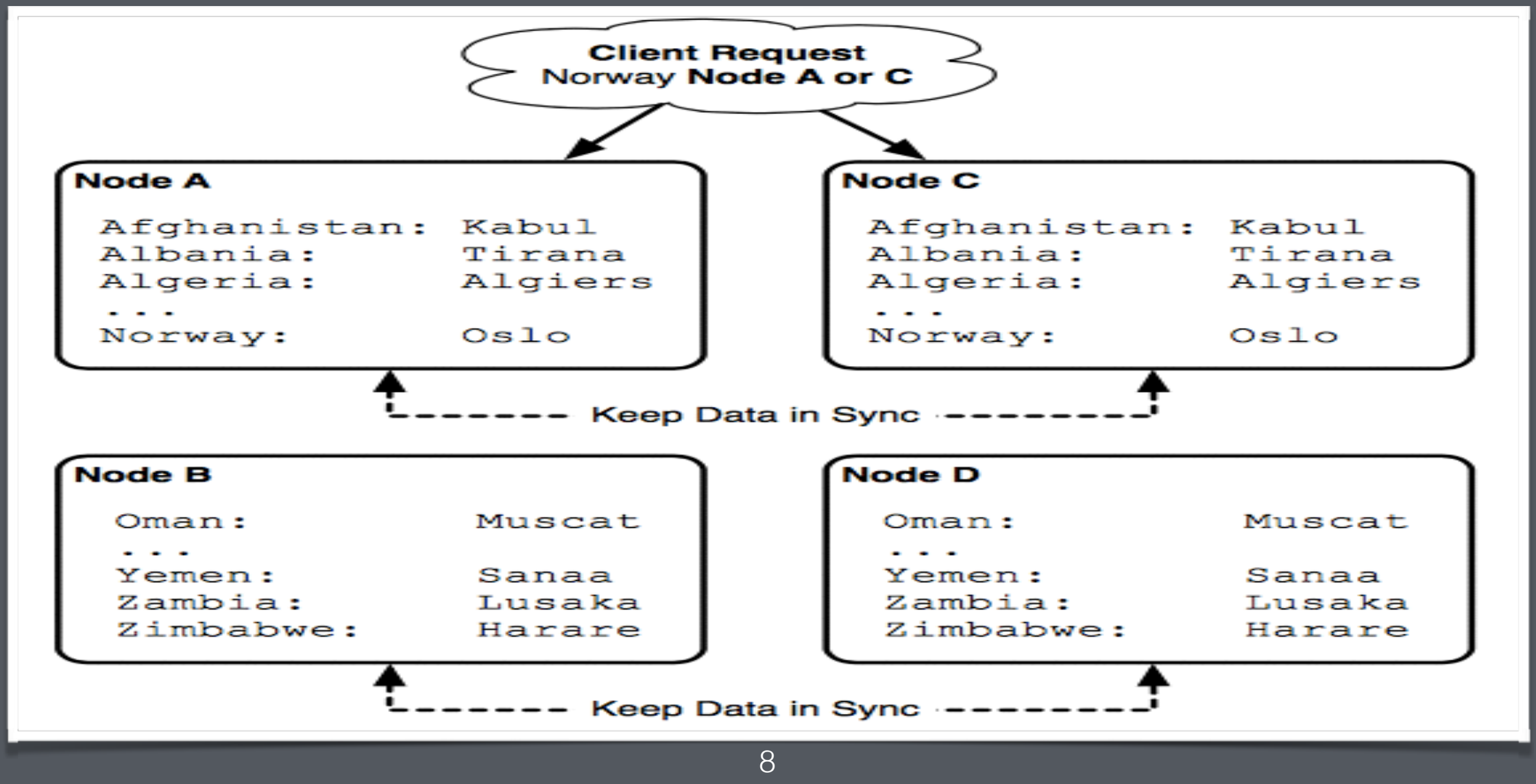
Partitions

- Partition is how we divide a set of keys onto separate physical servers
- Down side with one node goes down, that entire partition of data is unavailable.



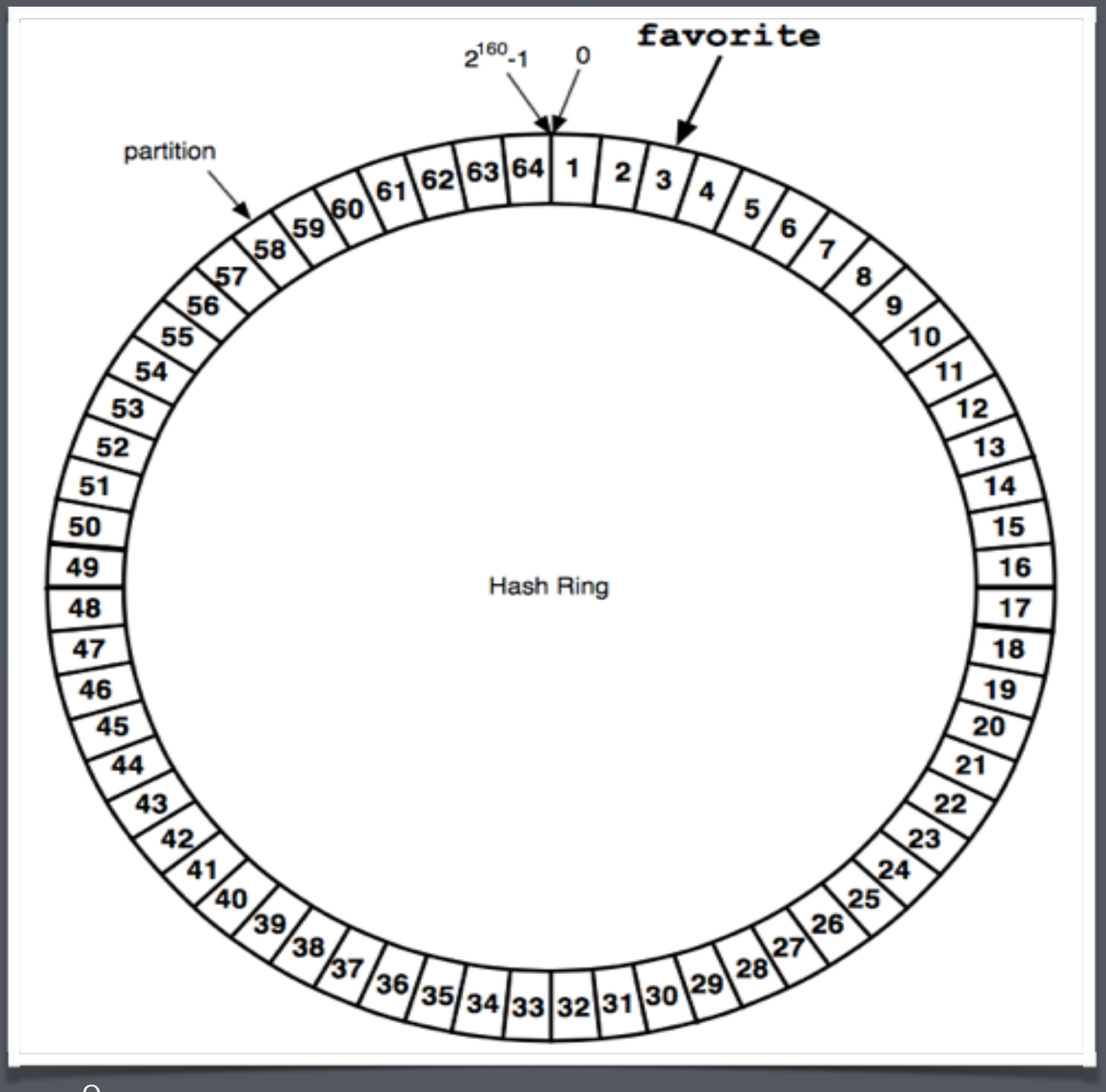
What Riak Do?

- Riak uses both replication and partitioning



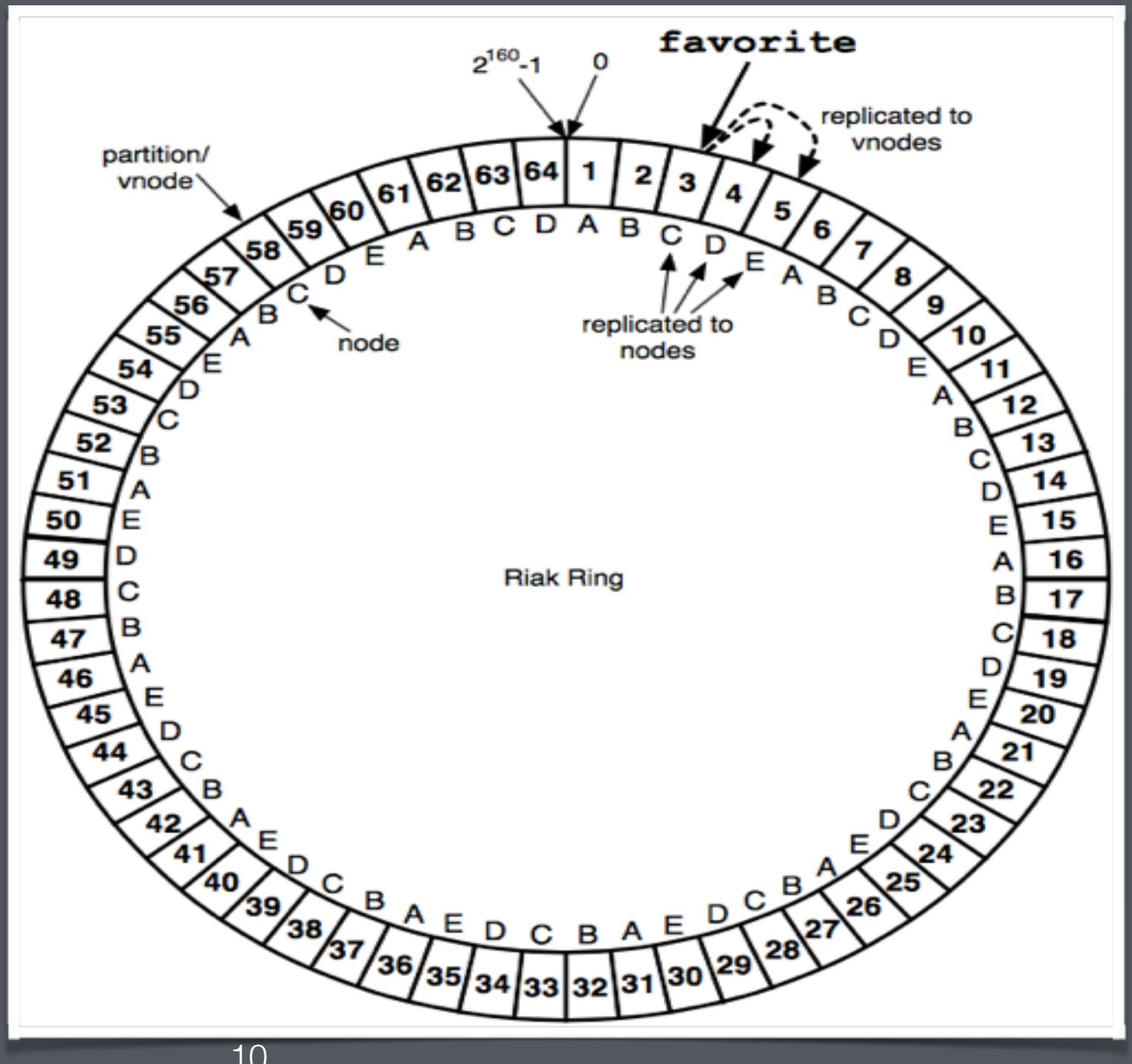
The Ring

- A = [1,6,11,16,21,26,31,36,41,46,51,56,61]
- B = [2,7,12,17,22,27,32,37,42,47,52,57,62]
- C = [3,8,13,18,23,28,33,38,43,48,53,58,63]
- D = [4,9,14,19,24,29,34,39,44,49,54,59,64]
- E = [5,10,15,20,25,30,35,40,45,50,55,60]



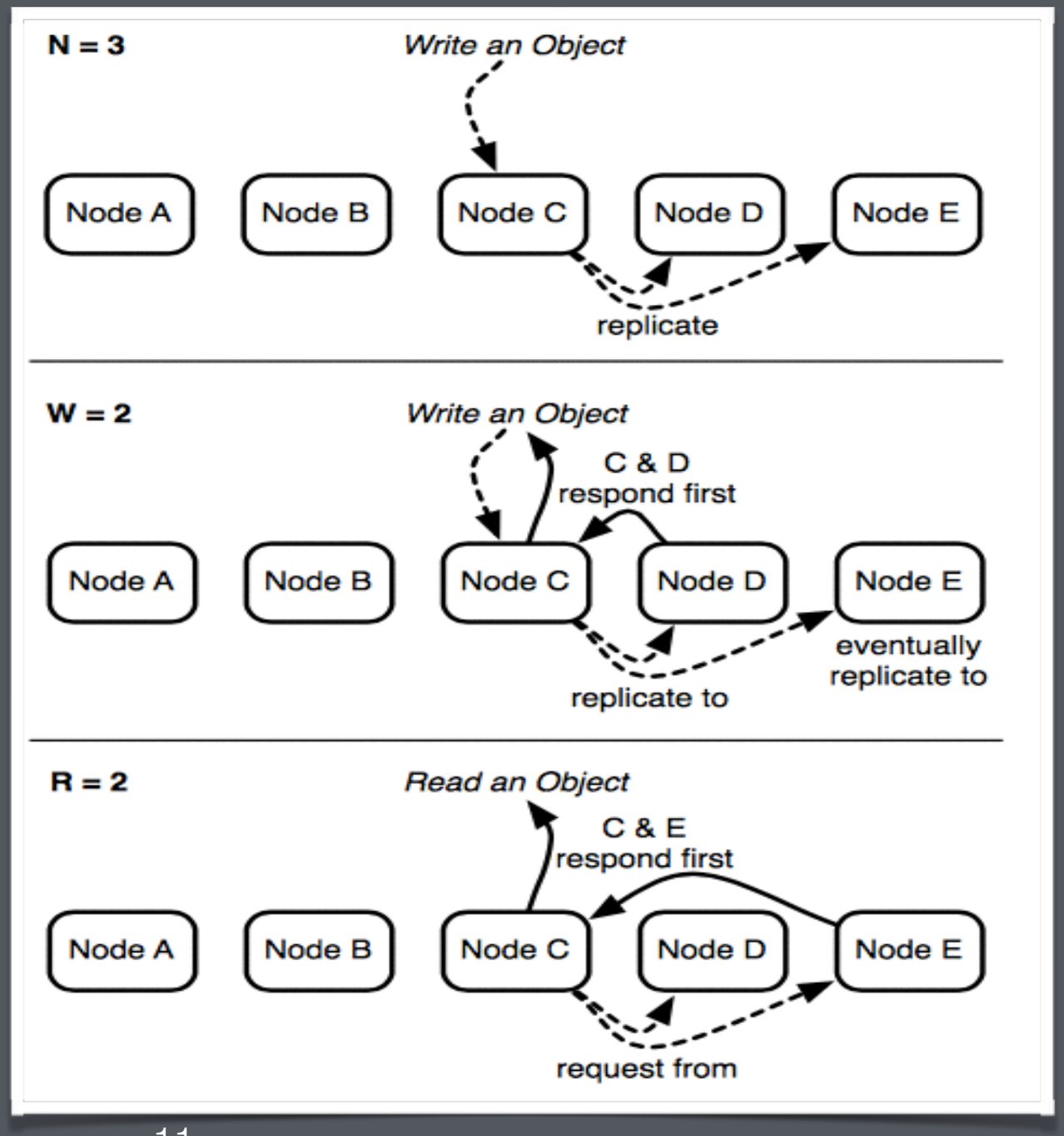
The Ring-Replication

- n_val=3
  - There are no guarantees that the three replicas will go to three separate physical nodes
- Favorite will be replicated to node C/D/E vnode 3/4/5
- Node C down, still available



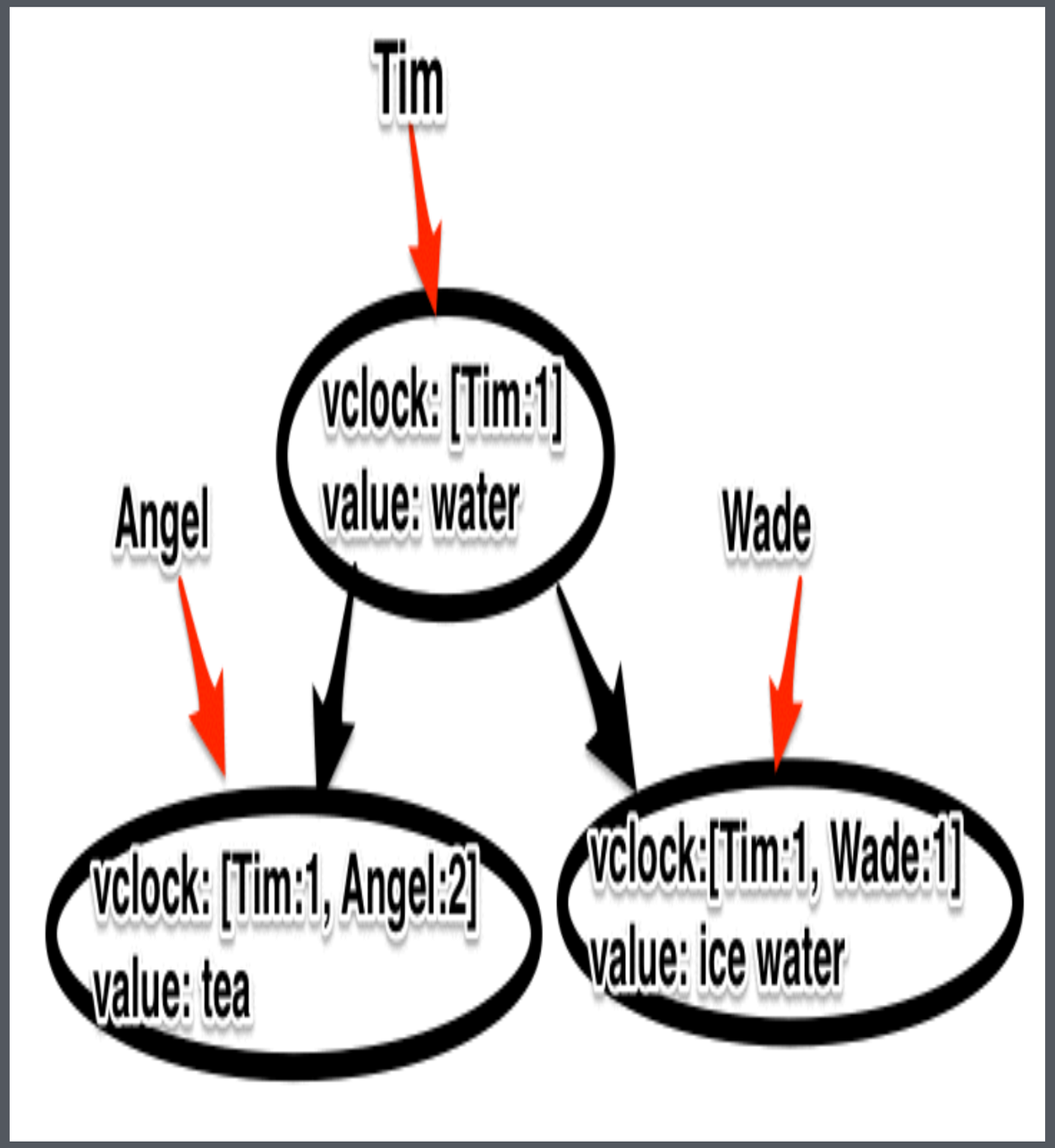
N/R/W

- N -> replicate n node
- W -> Write success node number
- DW -> Durable write
  - Write data to disk
- PW -> Write on primary node
- R -> Read success node number
- PR -> Read on primary node
- N - R = Read fault tolerance
- N - W = Write fault tolerance



Vector Clock

- Siblings
  - Concurrent writes
  - Missing vector clock
- Conflict Resolution
  - Keep change info while update
  - compare/merge siblings by timestamp
- Sibling Explosion
- Vector clock explosion
  - Large amount of updates performed on single object.
  - Resolve -> Vector Clock Pruning



How Eventual Consistency?

- Last Write Wins
  - Riak defualt consistency strategy
- Strong Consistency R+W>N
- Read Repair
  - Sync replicas while read to keep replicas up-to-date
- Hinted-Handoff
  - Node down -> other nodes accept writes to ensure availability
  - Node back -> other nodes send hints to recover data

What is Anti-Entropy?

- Ensure integrity of all data store in Riak
  - Continuous background process that compares and repairs any missing, or corrupted replicas
- Merkle Tree store on disk
  - Compare/validate data
  - Less memory usage
  - Avoid restart data loss
- Periodically Clear and regenerates Merkle Tree from K/V data
  - Detect slient data corruption on bad disk
  - Period config -> app.config

Upcoming Riak2.0 Features

- New Riak Data Types
  - Sets, maps, registers, and flags
  - Free to handle confliction
- Consistency preferences
  - Lets developers choose strongly or eventually consistent
  - There no need to calculate R/N/W
- Simplified Configuration Management
  - Remove Erlang specified syntax
  - Provide new automated deployment tool

References

- Basic Riak Cluster
- Riak Basic Usage
- Riak Concepts
- Riak Video
- Useful Riak Http Query API
- Riak Load Testing Sample Data
- Useful Riak Tools
- Riak Load Balancing and Proxy Configuration
- Riak Operating Riak FAQs
- Riak Configuration File
- Riak Log With Lager
- Riak Vector Clock
- Riak-CLI-Riak
- Riak-CLI-RiakAdmin
- Riak-CLI-SearchCMD
- Riak-BackUp
- Riak-Load-Balancing-With-HAProxy
- Riak-Eventual-Consistency-Video
- Riak FAQs
- Riak Control Cluster Nodes
- Riak-Cluster-On-Seperate-Machine
- Inspecting-Nodes
- Riak-Replication
- Riak Hand Book

Question&Answer

–Thank you!