Basho Riak

A Dynamo-inspired key/value store with a distributed database network platform.

History

- Developed by Basho
- Sales Force Automation business
- Riak more relevant.
- Build a business around riak.

The Team

- Erlang REST framework Webmachine.
- Akamai
- Eric Brewer (CAP theorem)

What is riak?

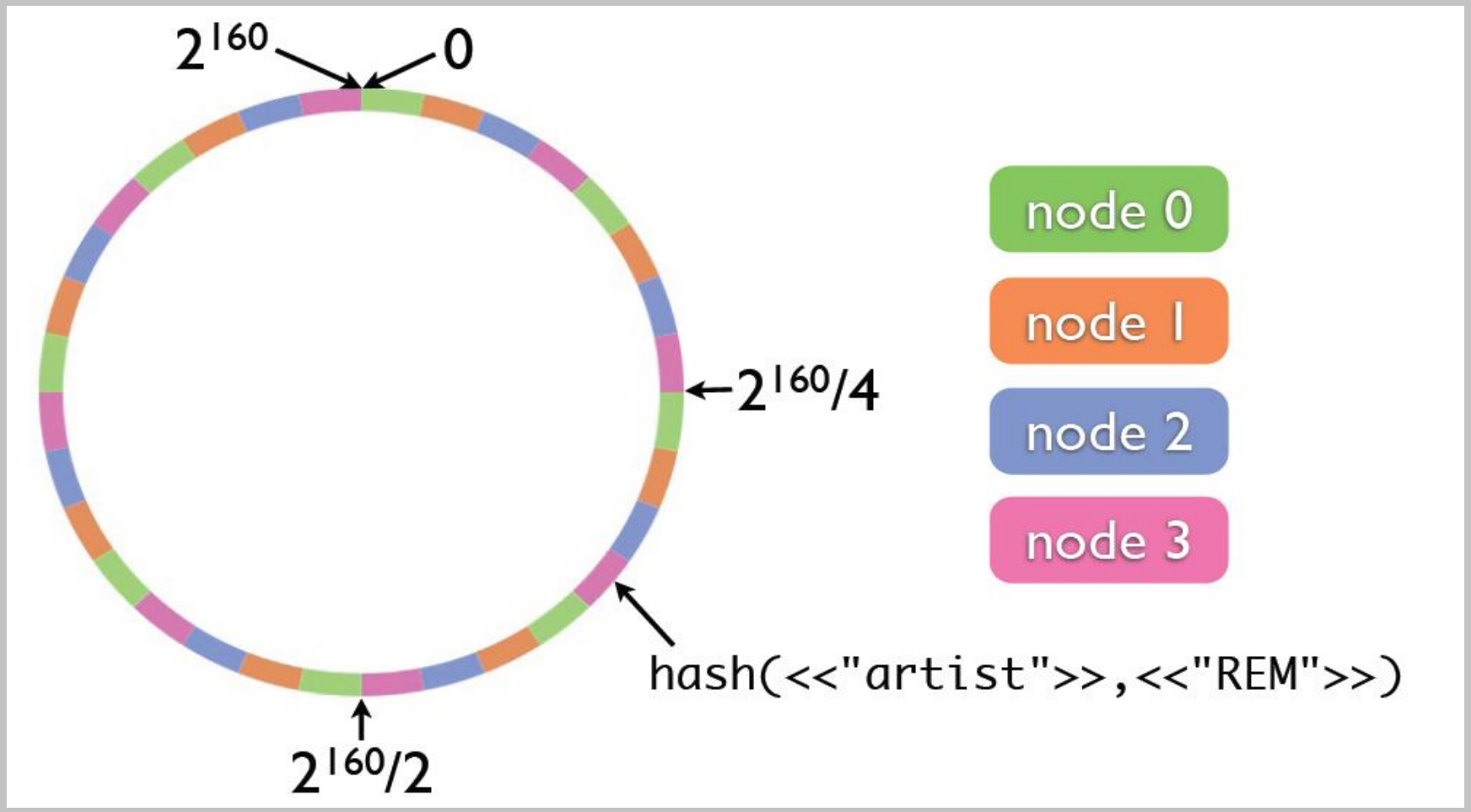
- A document-oriented database
- A decentralized datastore
- A fault-tolerant storage solution
- nosql, http, scalable, distributed, reliable

What is riak?

- CAP Theorem
- Dynamo
- The web
- Easy ops experience

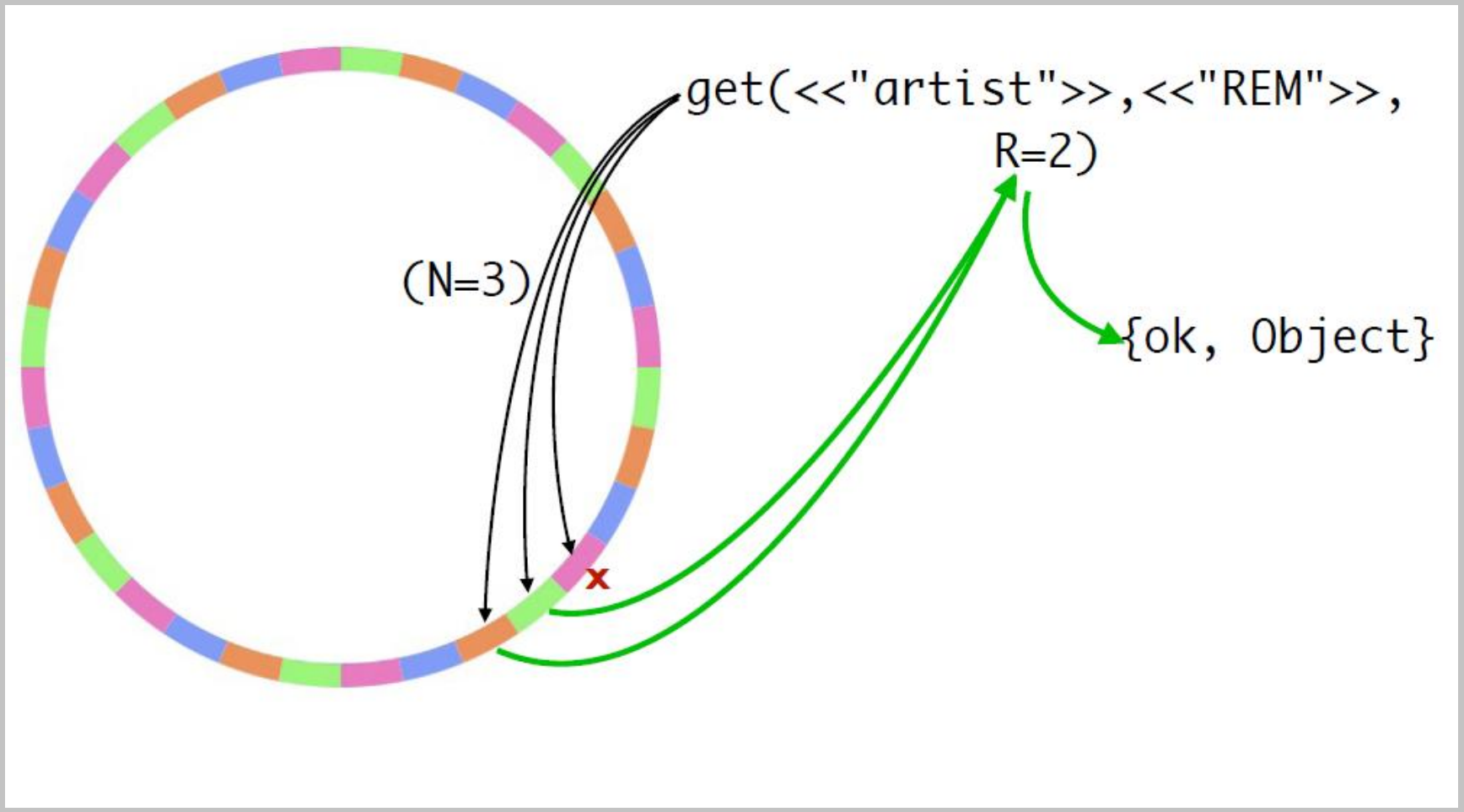
N Value

```
$ curl -v -X PUT -H "Content-Type: application/json"
-d '{"props":{"n_val":2}}' \
http://127.0.0.1:8091/riak/REM
```



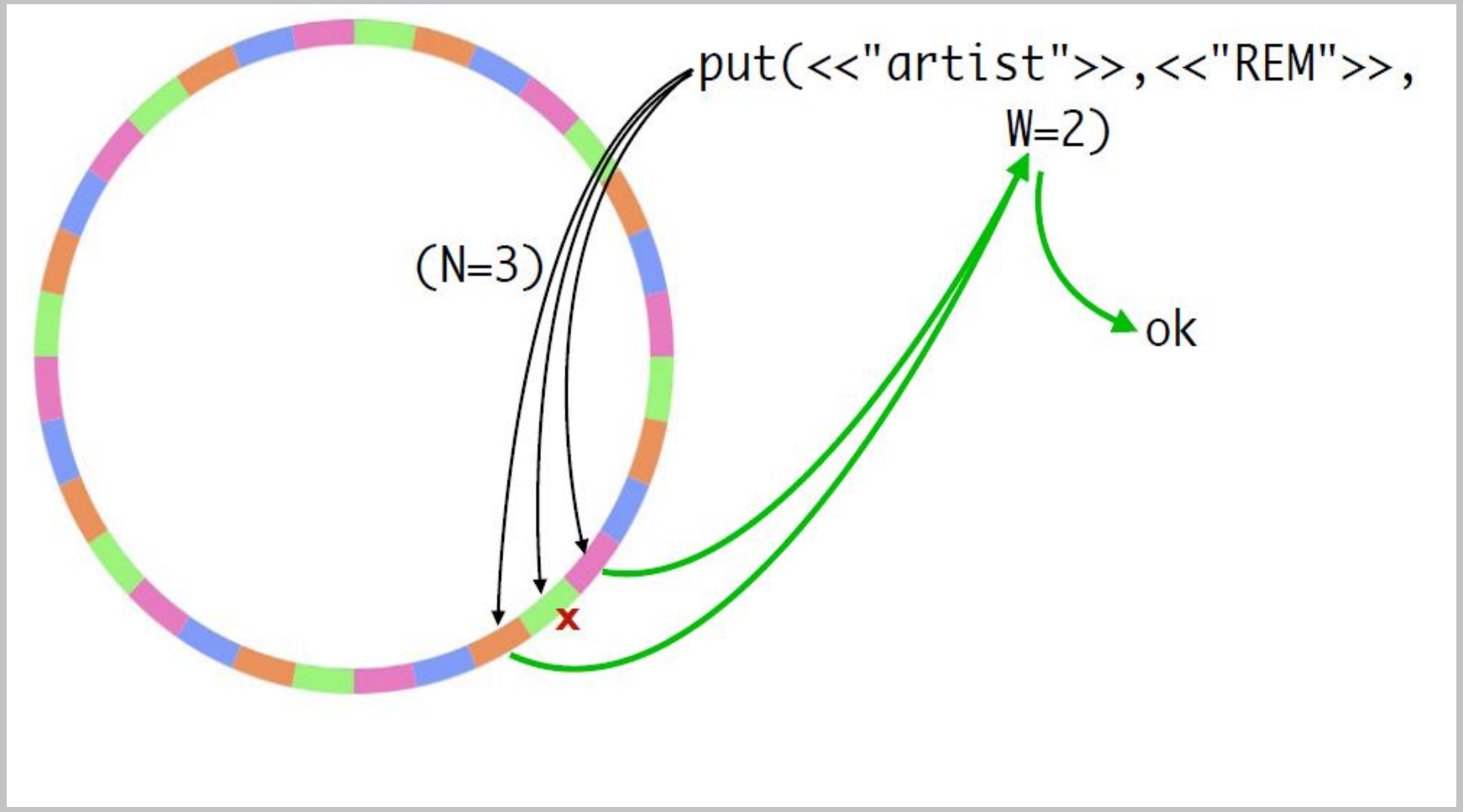
R Value

`http://`**`127.0.0.1:8091/riak/REM/artist?r=2`**



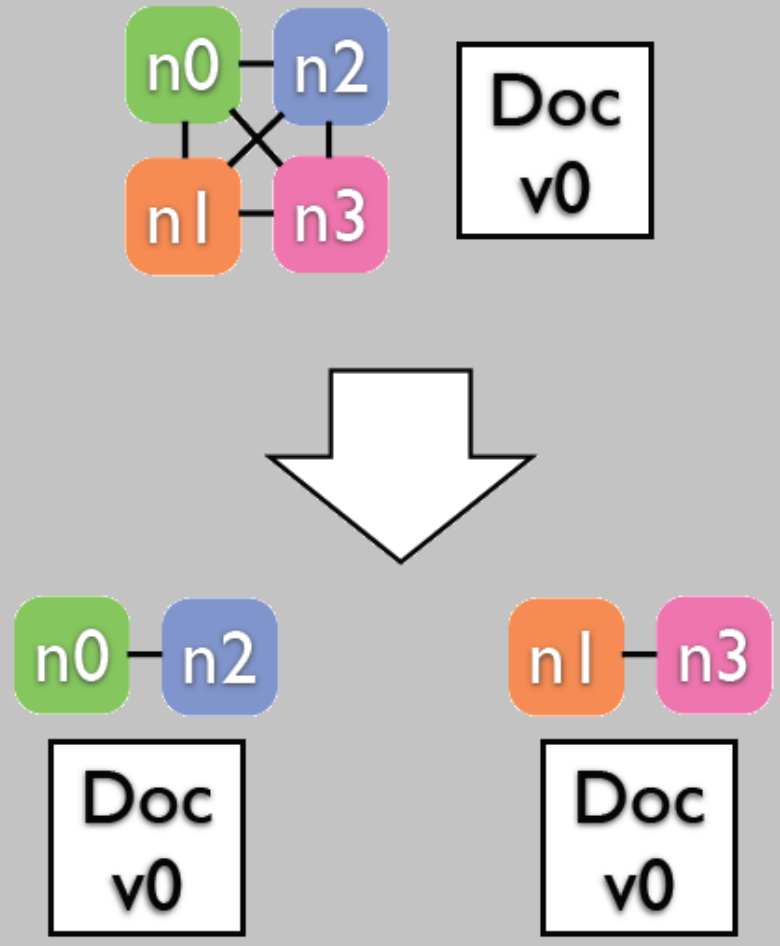
W Value

```
$ curl -v -X PUT http://127.0.0.1:8091/riak/docs/story.txt?w=2 \
-H "Content-type: text/plain" --data-binary @story.txt
```



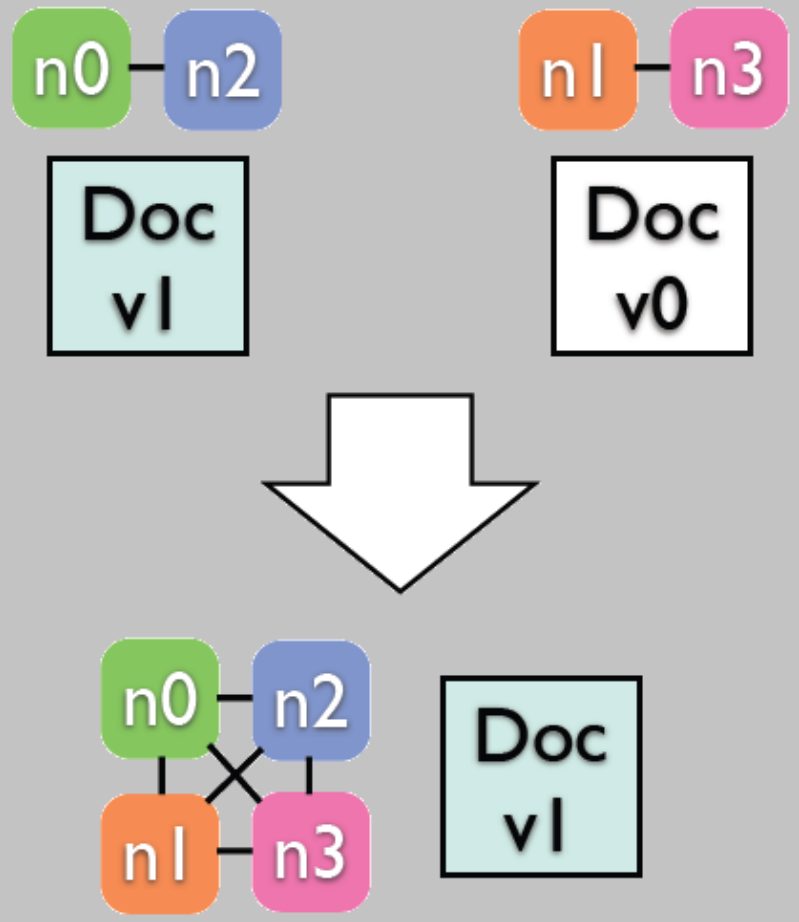
Partition Tolerance

All replicas of the object have the same Vector clock, and therefore the same copy.



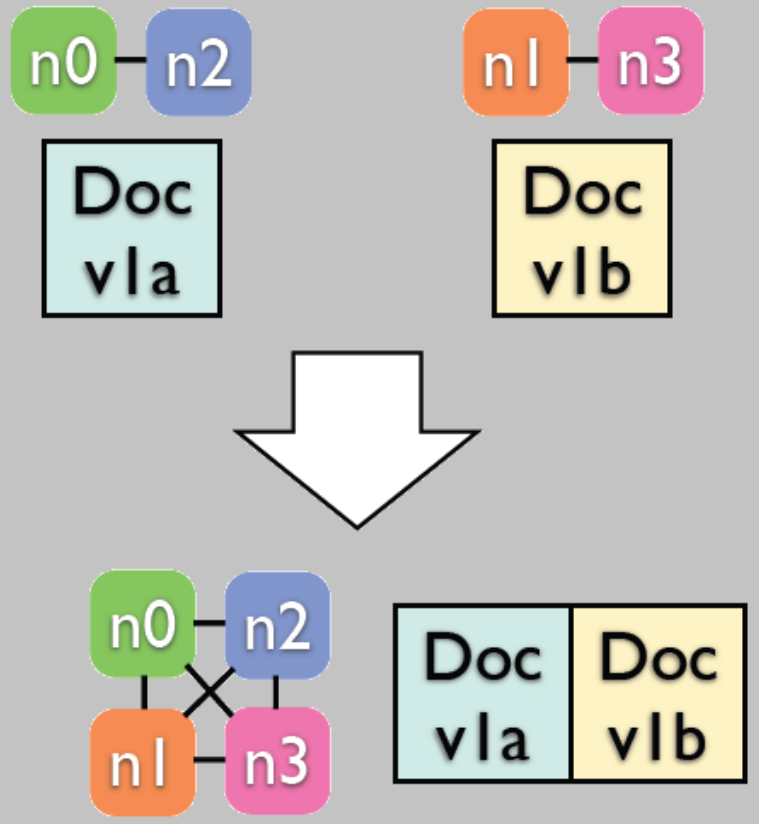
Riak Handling Inconsistency

- There is a network partition.
- The left partition is updated.
- Partition is removed, Riak will the causally older object.



Handling Inconsistency

- Both partitions modified the object.
- Neither document is more recent.
- From riak's point of view, the wall clock time is uninteresting.
- Can not disambiguate between versions. Defer to application.



Officially Supported Languages

- Erlang
- JavaScript
- Java
- PHP
- Python
- Ruby
- Community contributed projects for .NET, JavaScript, Python (and Twisted), Griffon, Perl, and Scala.

REST API

- Allows users to manipulate data using standard HTTP methods.
  - GET
  - PUT (POST)
  - DELETE

Bucket Operations

- List buckets
  - GET /riak?buckets=true
- Read bucket properties and keys
  - GET /riak/bucket
- Set bucket properties like “n_val” or “allow_mult”
  - PUT /riak/bucket

```
$ curl -v http://127.0.0.1:8098/riak/test
* About to connect() to 127.0.0.1 port 8098 (#0)
* Trying 127.0.0.1... Connected
* Connected to 127.0.0.1 (127.0.0.1) port 8098 (#0)
> GET /riak/test HTTP/1.1
> User-Agent: curl/7.19.7 (universal-apple-darwin10.0) libcurl/7.19.7 OpenSSL/0.9.8l zlib/1.2.3 > Host:
127.0.0.1:809
> Accept: */*
>
< HTTP/1.1 200 OK
< Vary: Accept-Encoding < Server: MochiWeb/1.1 WebMachine/1.7.1 (participate in the frantic)
< Date: Wed, 14 Jul 2010 18:23:14 GMT
< Content-Type: application/json
< Content-Length: 368
< * Connection #0 to host 127.0.0.1 left intact * Closing connection #0
{"props":{"name":"test","n_val":3,"allow_mult":false,"last_write_wins":false,"precommit":[],"postcommit":[],"
chash_keyfun":{"mod":"riak_core_util","fun":"chash_std_keyfun"},"linkfun":{"mod":"riak_kv_wm_link_walker","
fun":"mapreduce_linkfun"},"old_vclock":86400,"young_vclock":20,"big_vclock":50,"small_vclock":10,"r":"quor
um","w":"quorum","dw":"quorum","rw":"quorum"
```

Key Operations

- Read an object from a bucket
  - GET /riak/bucket/key
- Store new object in bucket
  - POST /riak/bucket/ (riak-assigned key)
  - POST /riak/bucket/key (user-defined key)
- Delete an object from a bucket
  - DELETE /riak/bucket/key

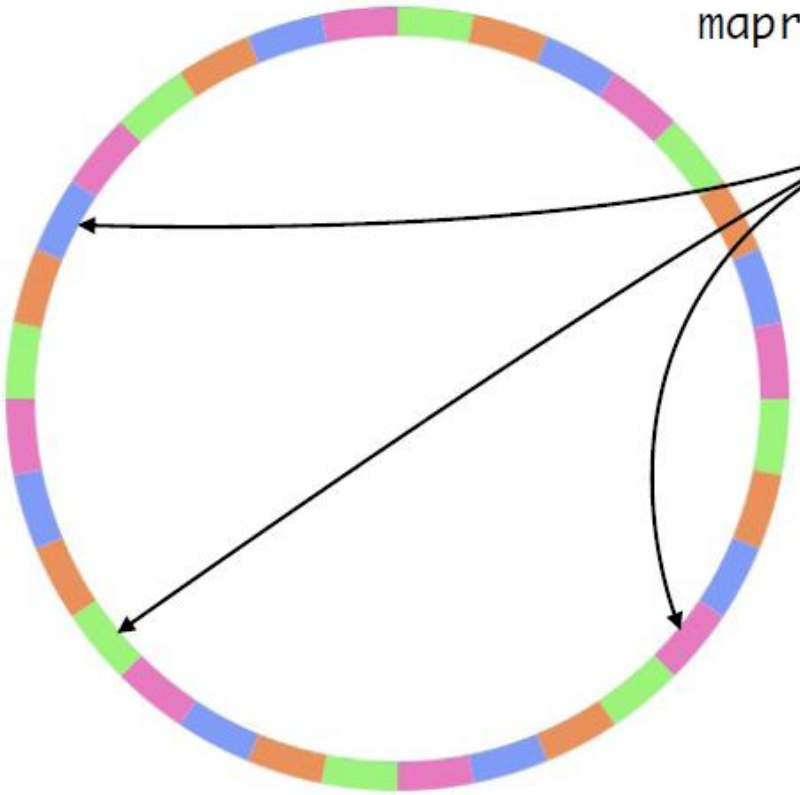
POST Example

```
$ curl -v -X PUT -d '{"bar":"baz"}' -H "Content-Type: application/json" -H
"X-Riak-Vclock: a85hYGBgzGDKBVIszMk55zKYEhnzWBlKIniO8mUBAA=="
http://127.0.0.1:8098/riak/test/doc?returnbody=true
* About to connect() to 127.0.0.1 port 8098 (#0)
* Trying 127.0.0.1... connected
* Connected to 127.0.0.1 (127.0.0.1) port 8098 (#0)
> PUT /riak/test/doc?returnbody=true HTTP/1.1
> User-Agent: curl/7.19.4 (universal-apple-darwin10.0) libcurl/7.19.4
OpenSSL/0.9.8l zlib/1.2.3
> Host: 127.0.0.1:8098
> Accept: */*
> Content-Type: application/json
> X-Riak-Vclock: a85hYGBgzGDKBVIszMk55zKYEhnzWBlKIniO8mUBAA==
> Content-Length: 13
>
< HTTP/1.1 200 OK
< X-Riak-Vclock: a85hYGBgymDKBVIszMk55zKYEhnzWBlKIniO8kGF2TyvHYIKfwcJZwEA
< Vary: Accept-Encoding
< Server: MochiWeb/1.1 WebMachine/1.6 (eat around the stinger)
< Link: </riak/test>; rel="up"
< Date: Wed, 10 Mar 2010 17:55:03 GMT
< Content-Type: application/json
< Content-Length: 13
< * Connection #0 to host 127.0.0.1 left intact * Closing connection #0
{"bar":"baz"}
```

Map Reduce

- Increased Data Locality
- Take the computation to the data
- Map-step
  - Run map-step functions on the node holding the data for the Map-step.
  - Sends results back to coordinating node
- Reduce-step
  - Run reduce-step functions on the node coordinating the Map Reduce query

Map Reduce



```
mapred([{<<"artist">>,<<"REM">>},
        {<<"artist">>,...},...],
       [{map,
         {modfun,artist,member_count},
         none,false},
        {reduce,
         {qfun,fun(L,_,_) ->
                   lists:unique(L)
               end},
         none, true}]).
```

Map Reduce

- POST operation to map reduce resource
- POST –H “content-type: application/json” http://localhost:8989/mapred --data @-
- Body application/json
  - {“inputs”: […inputs…], “query”: […query…]}
  - Inputs can be “bucket”, [bucket, key], [bucket, key, keydata]

Linking

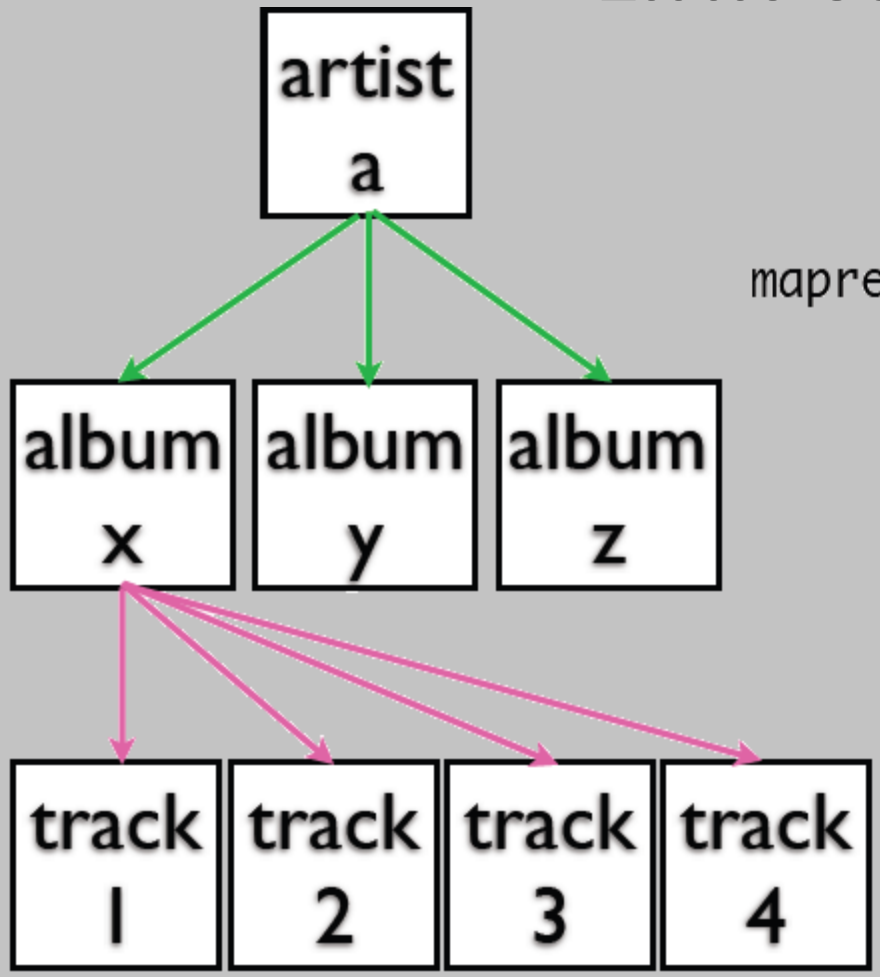
- Link is a HTTP header.
- Link: </riak/genre/bluegrass>; riaktag=“listens“

PUT –H Link: </riak/genre/bluegrass>; riaktag=“listens“ \

-H “content-type: text/plain” http://localhost:8989/riak/people/sbz \

-d “bluegrass music”

Link Walking




```
mapred([{<<"artist">>,<<"REM">>}],
       [{link,<<"album">>,'_',false},
        {link,<<"track">>,'_',false},
        {map,{modfun,track,name},
         none, true}]).
```

http://host/jiak/artist/REM/album,_,_/track,_,_

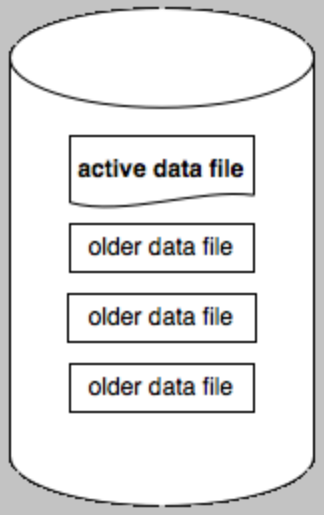
Data Storage

- Bucket/key pairs
- Links and Metadata
- Pluggable Backends
  - Uses API to interact with storage system.
  - Any thing k/v-shaped works.
  - Default backend: Bitcast

Bitcast Goals

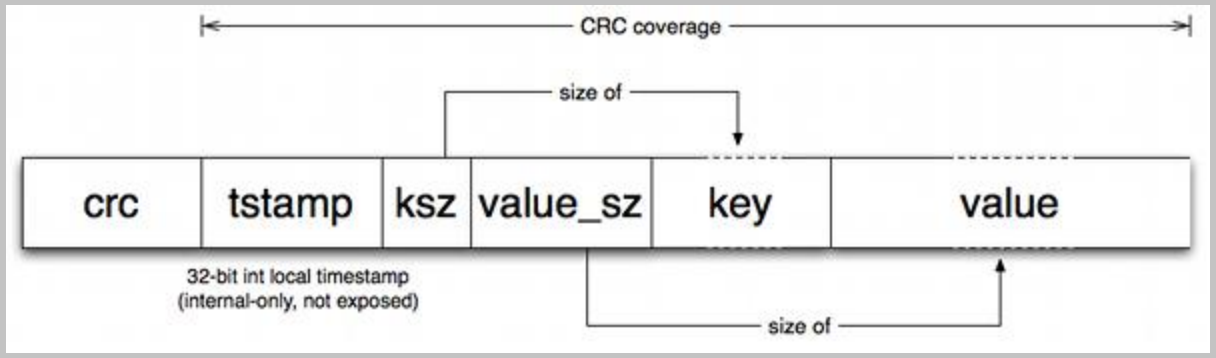
- Low latency per read/write
- High throughput
- Large Data w/o Degradation
- Crash Friendliness

Brewer proposes hash table log merging.



A Bitcast Instance: is basically a directory. Only one file is “active” for writing. All other files closed and immutable.

The active file is written by appending a new entry (below). Sequential writes do not require disk seeking.

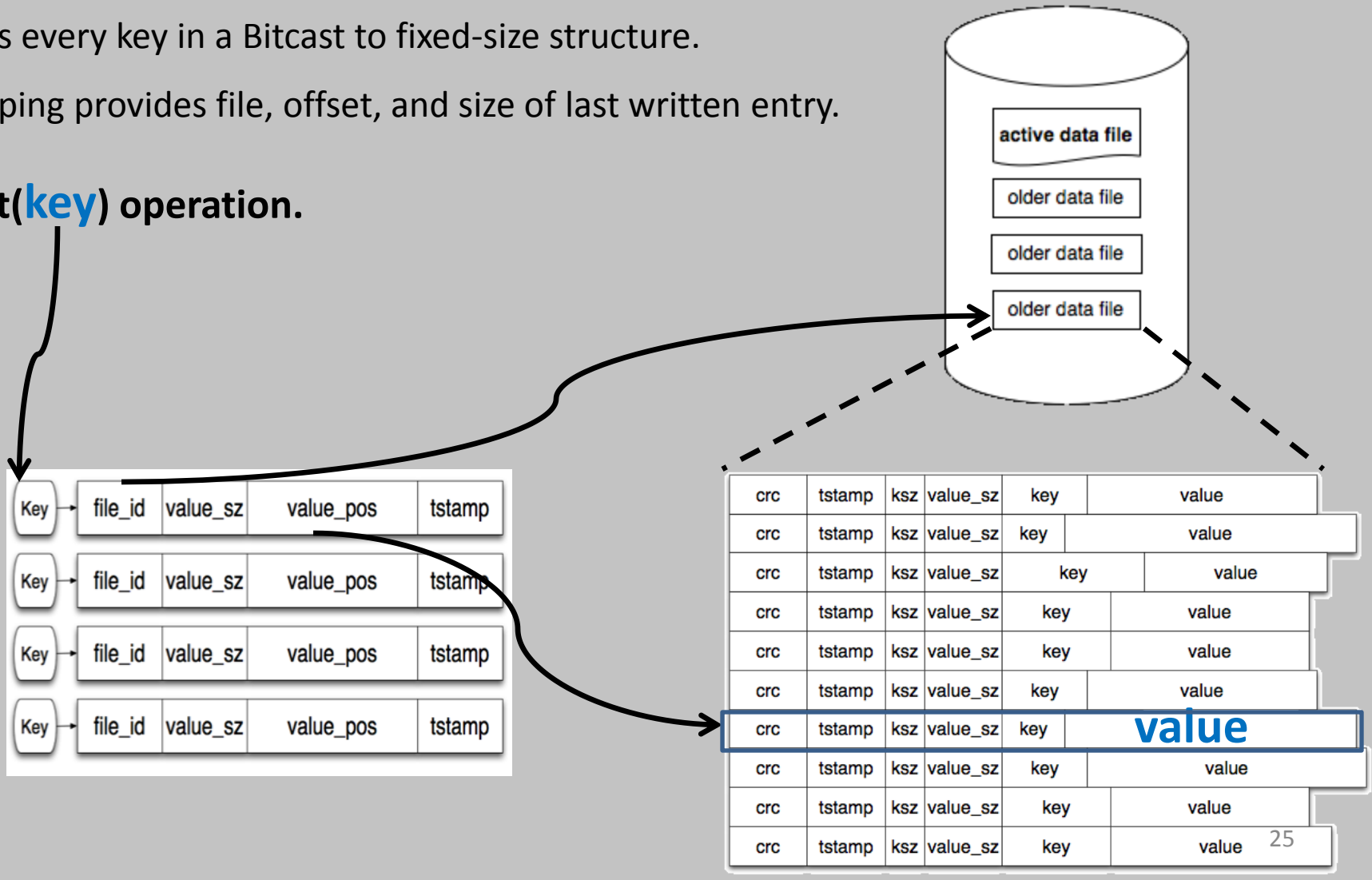


A delete is a simply a write of a tombstone value, which indicates an entry must be removed on the next merge.

After an append completes, the "keydir" is updated.

- keydir is a hash table.
- maps every key in a Bitcast to fixed-size structure.
- mapping provides file, offset, and size of last written entry.

A get(key) operation.



Applications on Riak

- Mozilla Test Pilot is using structured user feedback.
  - Running Multiple Riak Clusters to gather user data and perform large-scale analysis with MapReduce.
  - Chose Riak over Cassandra and Hbase because the extensibility; schema changes and bucket creation is completely dynamic.
  - API: The reliable and heavily tested REST server is built in to riak.
  - Cost : Light on memory requirements.

Decided to use Riak though mozilla is heavily invested in the similar product, HBase.

mozilla
FOUNDATION

Riak Search

- Inverted index of terms to document IDs.
- Enable buckets for search integration.
- Any objects stored in that bucket will be indexed seamlessly with Riak Search.