Masterless Distributed Computing with Riak Core

Erlang User Conference
Stockholm, Sweden · November 2010

Rusty Klophaus (@rklophaus)
Basho Technologies

basho

riak

Basho Technologies

About Basho Technologies

- Offices in:
  - Cambridge, Massachusetts
  - San Francisco, California
- Distributed company
- ~20 people
- Riak KV and Riak Search (both Open Source)
- SLA-based Support and Enterprise Software ($)
- Other Open Source Erlang developed by Basho:
  - Webmachine, Rebar, Bitcask, Erlang_JS, Basho Bench

basho

riak

Riak KV and Riak Search

Riak KV

Key/Value Datastore

Map/Reduce, Lightweight Data Relations, Client APIs

Riak Search

Full-text search and indexing engine

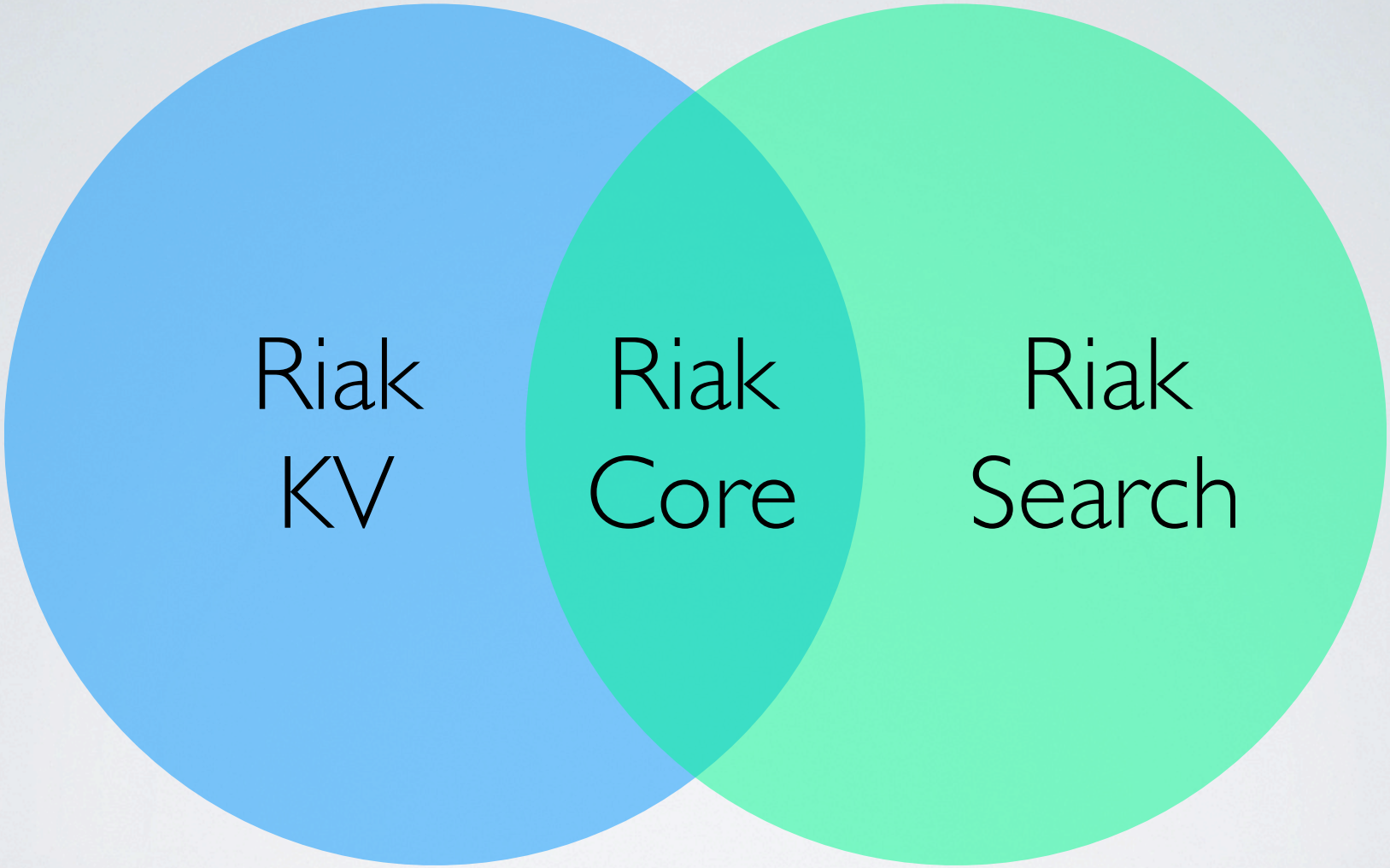
Near Realtime Indexing, Riak KV Integration, Solr Support.

Common Properties

Both are distributed, scalable, failure-tolerant applications.

Both based on Amazon's Dynamo Architecture.

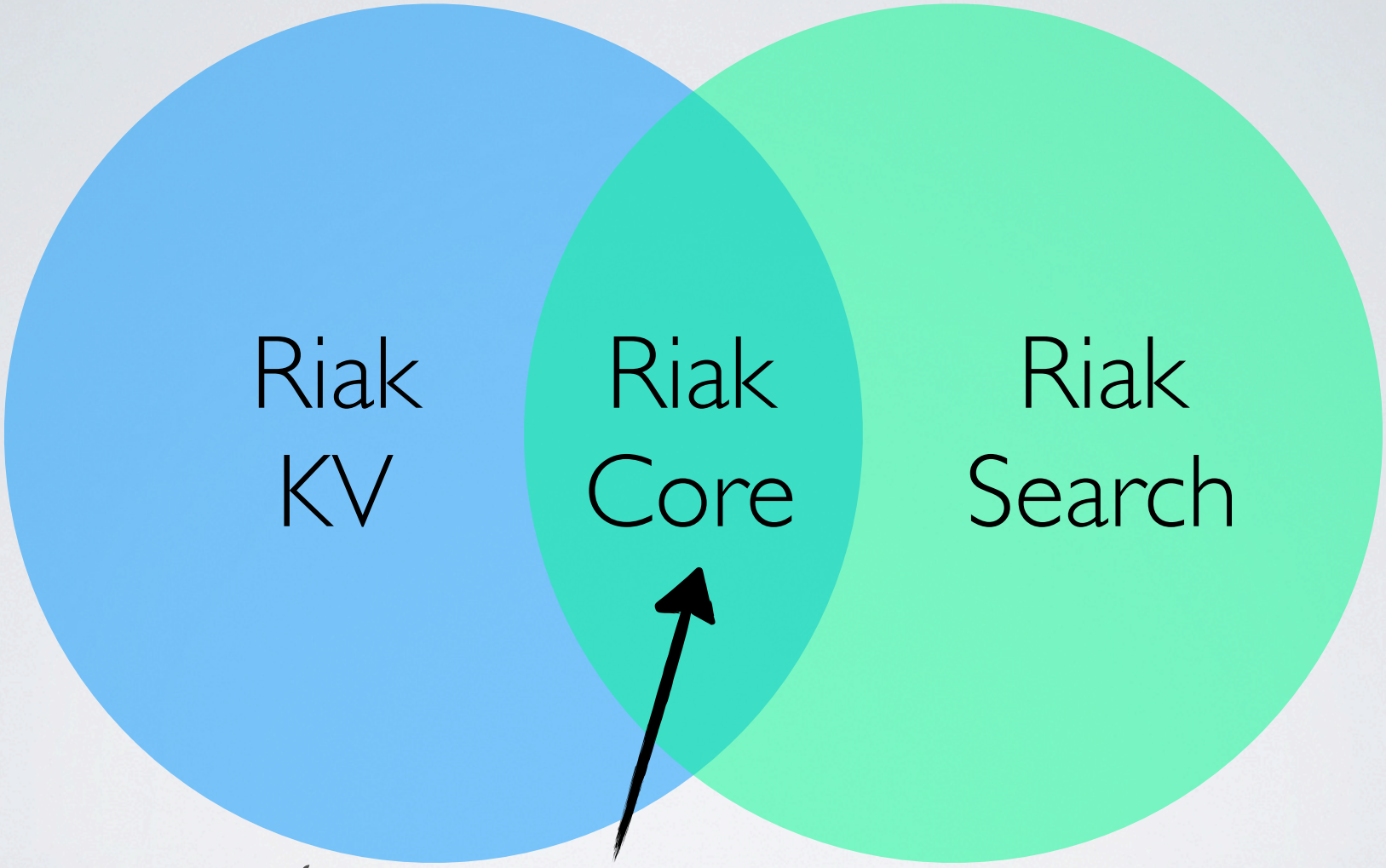
The Common Parts are called Riak Core



basho

riak

The Common Parts are called Riak Core



basho

riak

Riak Core is an
Open Source Erlang library
that helps you build
distributed, scalable, failure-tolerant
applications using a
Dynamo-style architecture.

"We Generalized the Dynamo Architecture and Open-Sourced the Bits."

What Areas are Covered?

Amazon's Dynamo Paper highlighted to show parts covered in Riak Core.

basho

riak

Distributed, scalable, failure-tolerant.

Distributed, scalable, failure-tolerant.

No central coordinator.

Easy to setup/operate.

Distributed, scalable, failure-tolerant.

Horizontally scalable;
add commodity hardware
to get more X.

basho

riak

Distributed, scalable, failure-tolerant.

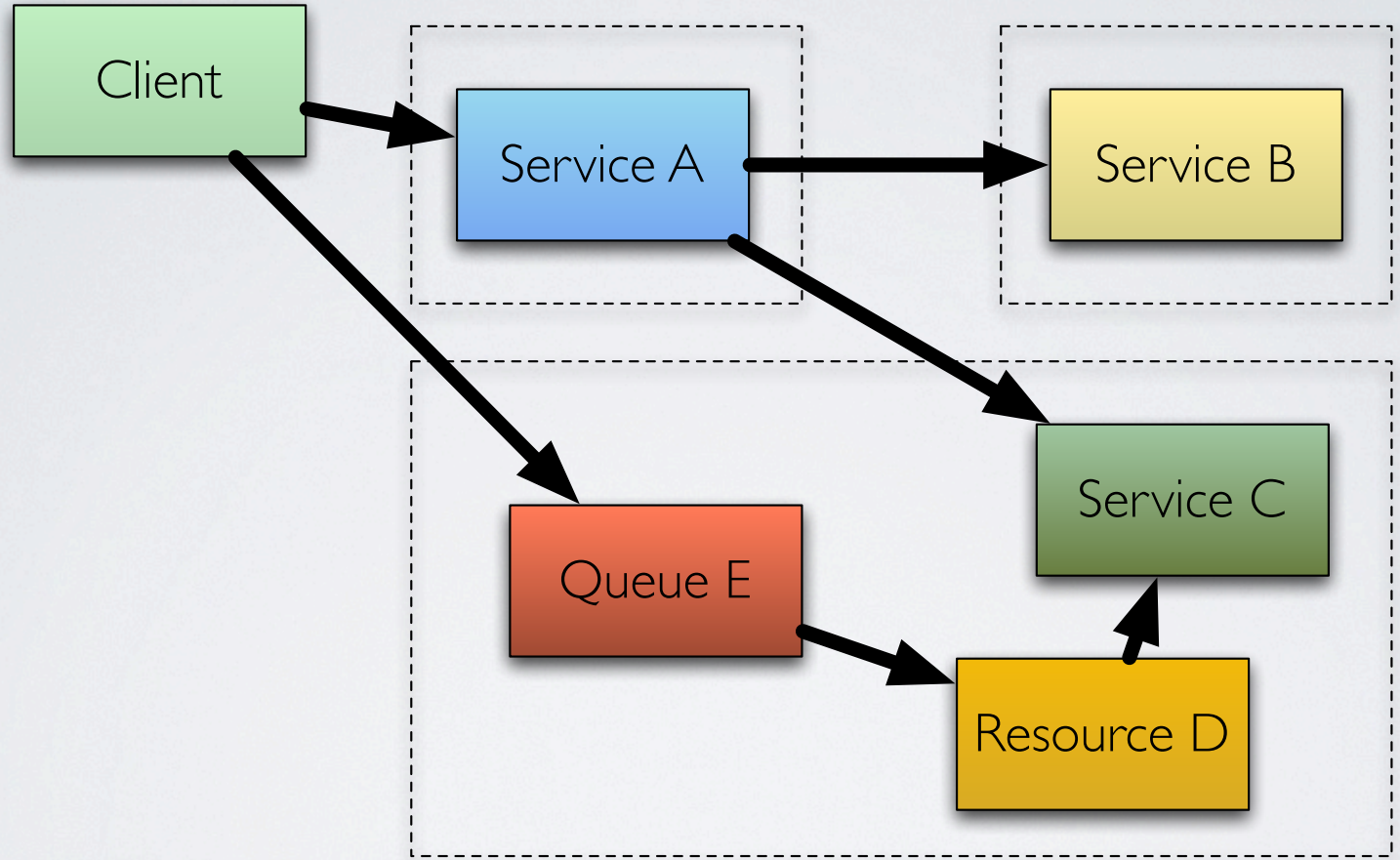
Always available.

No single point of failure.

Self-healing.

Wait, doesn't *Erlang* let you build distributed, scalable, failure-tolerant applications?

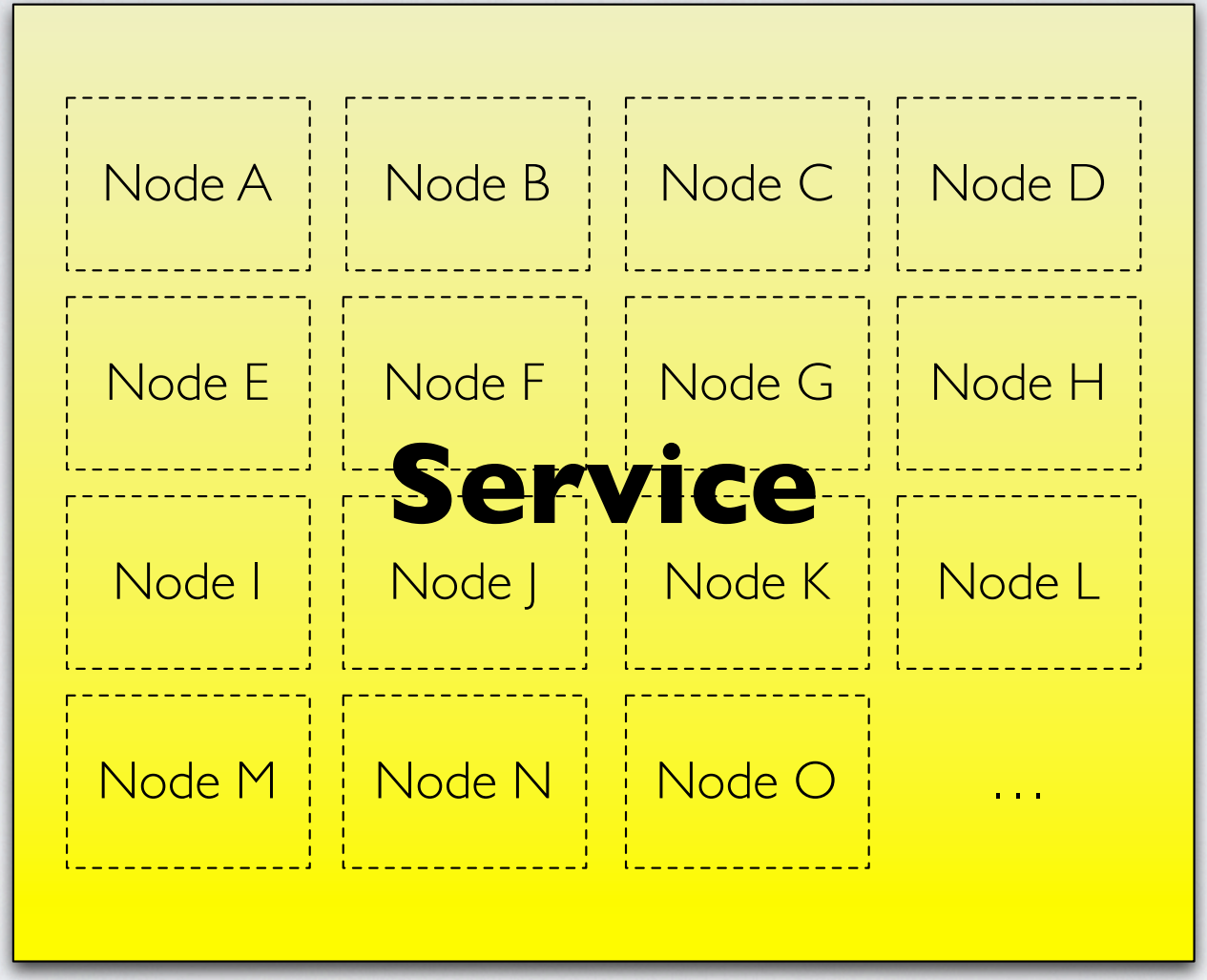
Erlang makes it easy to connect the components of your application.



basho

riak

Riak Core helps you build a service that harnesses the power of many nodes.



basho

riak

How does Riak Core work?

A Simple Interface...

Send commands, get responses.

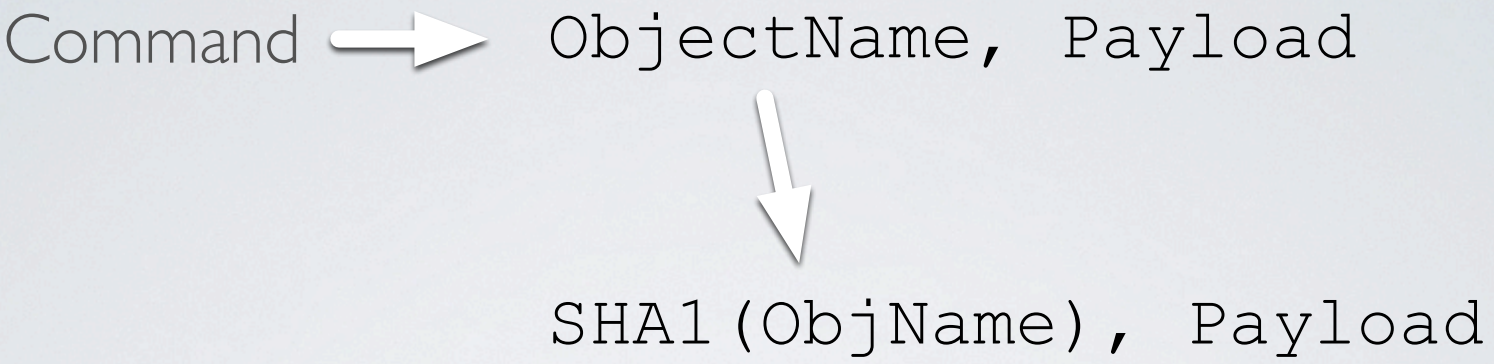
Command  `ObjectName, Payload`

How do we route the commands
to physical machines?

basho

riak

Hash the Object Name

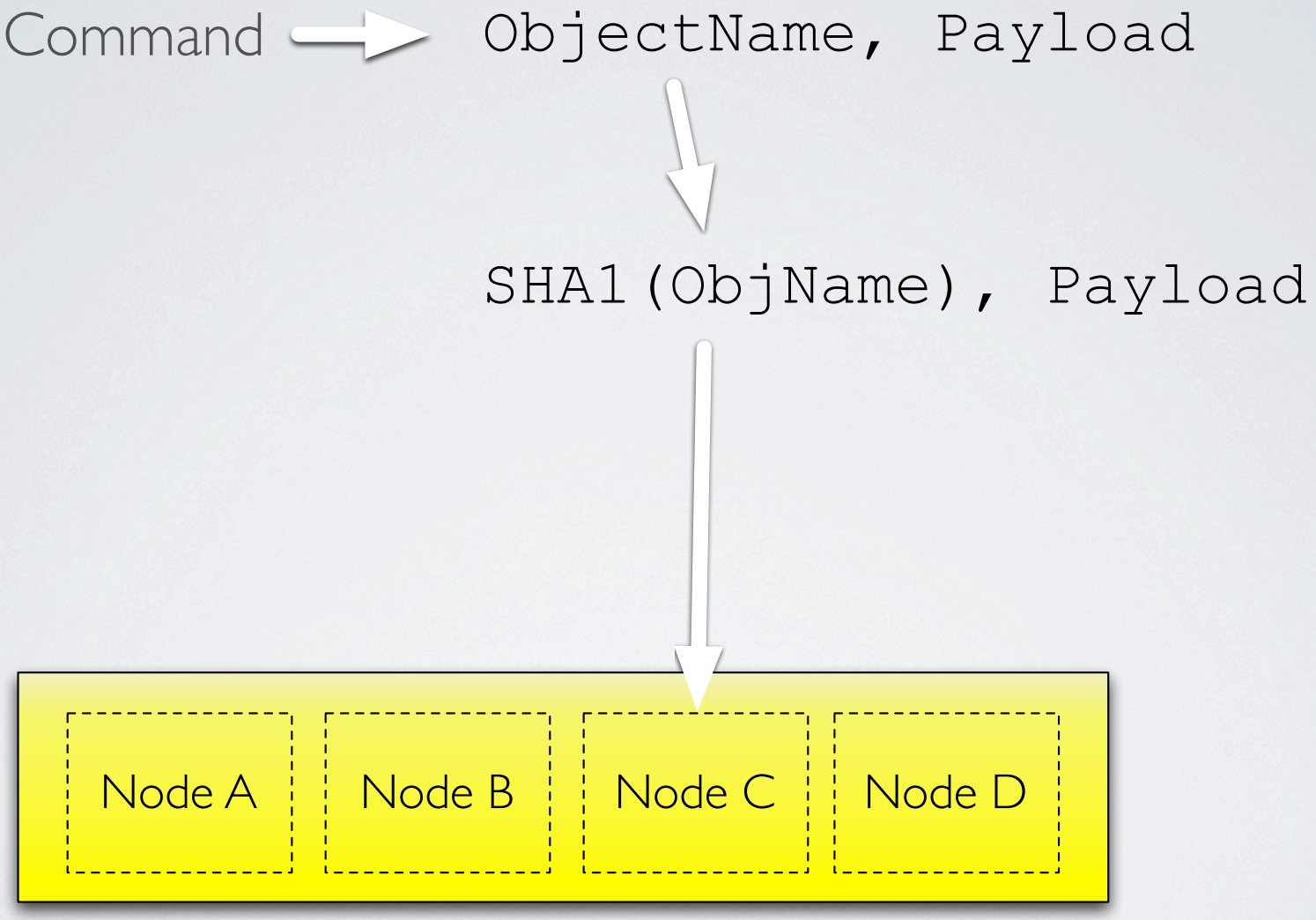


0 to 2^160

basho

riak

A Naive Approach



basho

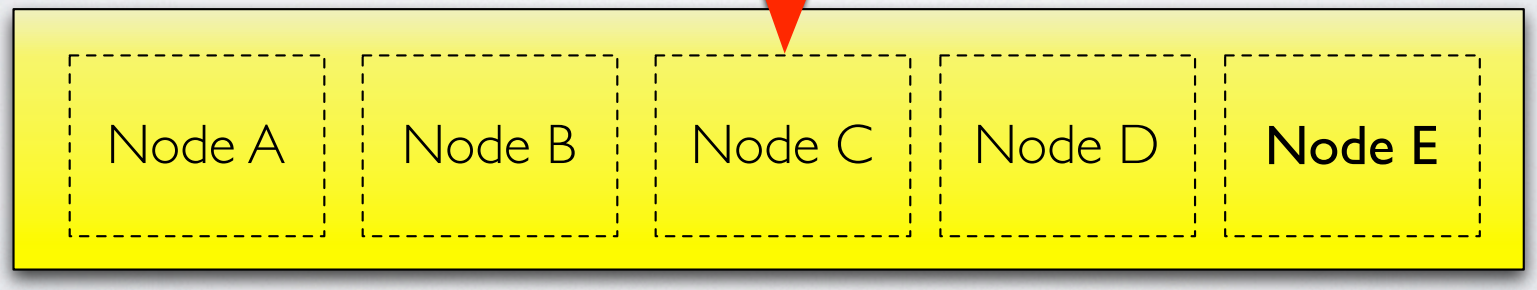
riak

A Naive Approach

Command → `ObjectName, Payload`

`SHA1(ObjName), Payload`

Existing routes become invalid when you add/remove nodes.



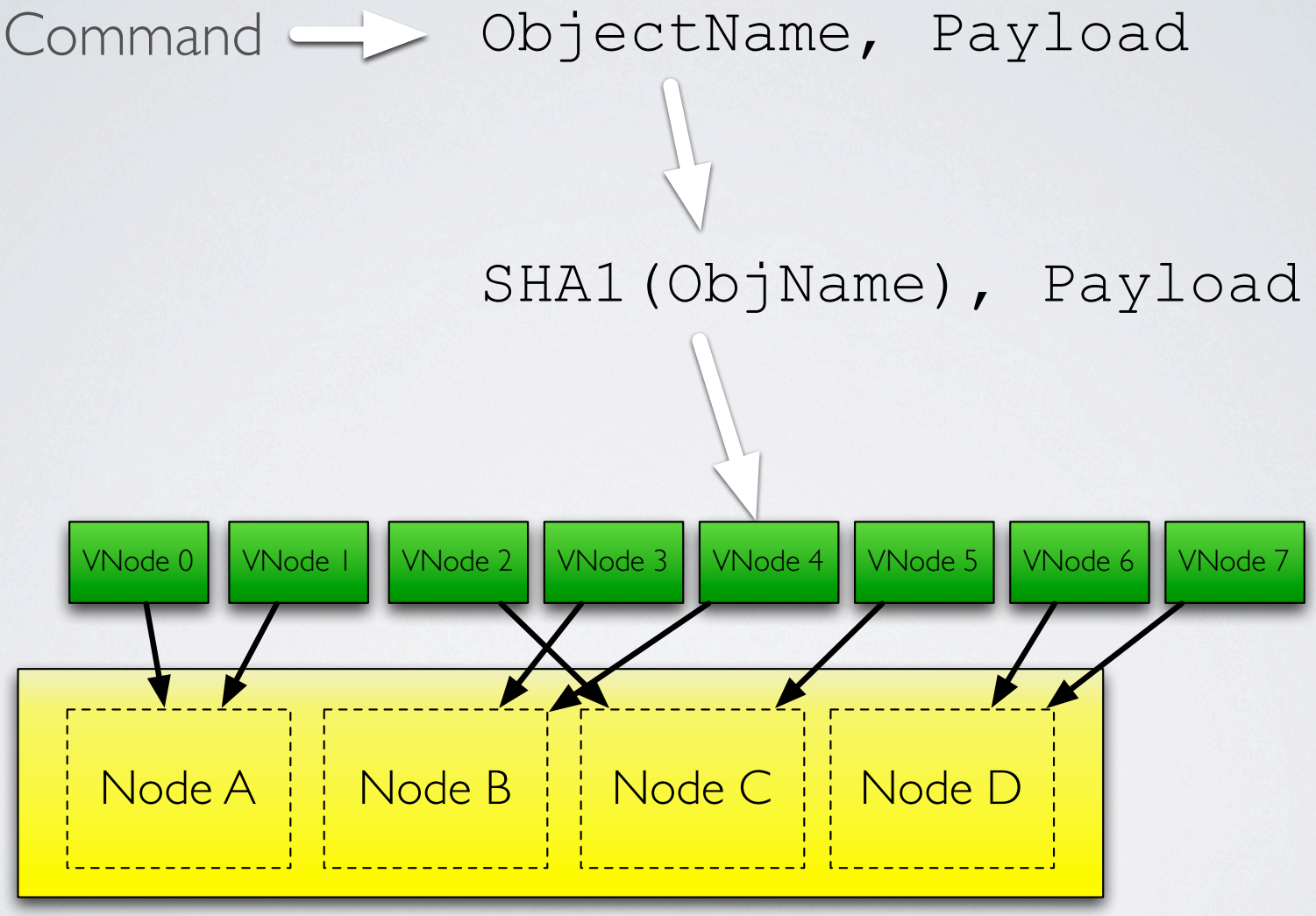
basho

riak

"All problems in computer science can be solved by another level of indirection."

- David Wheeler

Routing with Consistent Hashing



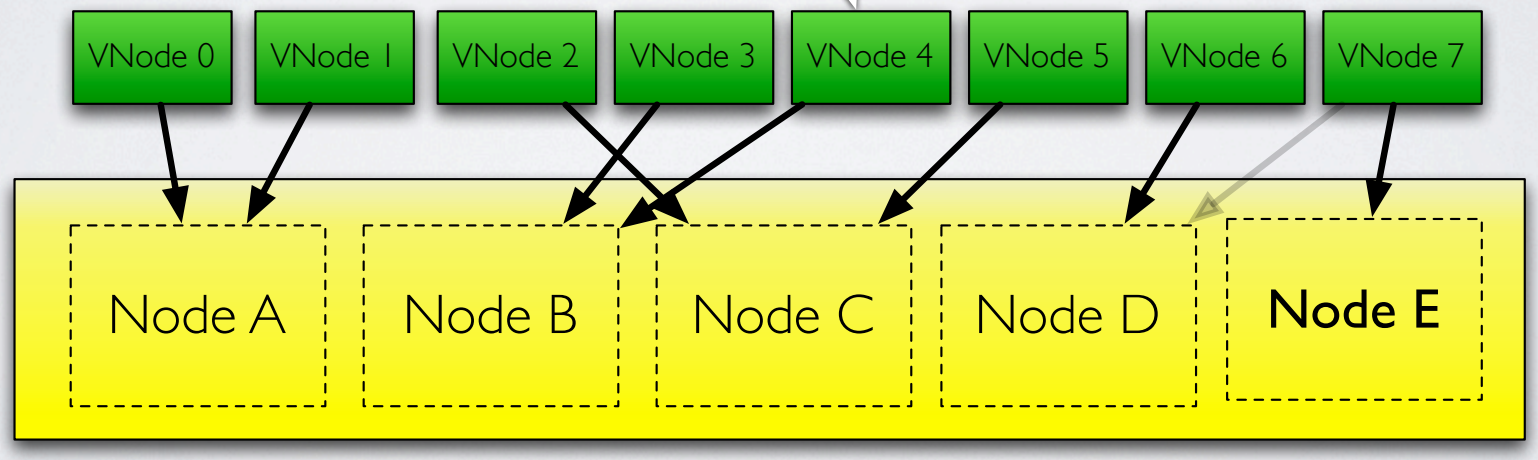
basho

riak

Adding a Node

Command → `ObjectName, Payload`

`SHA1(ObjName), Payload`



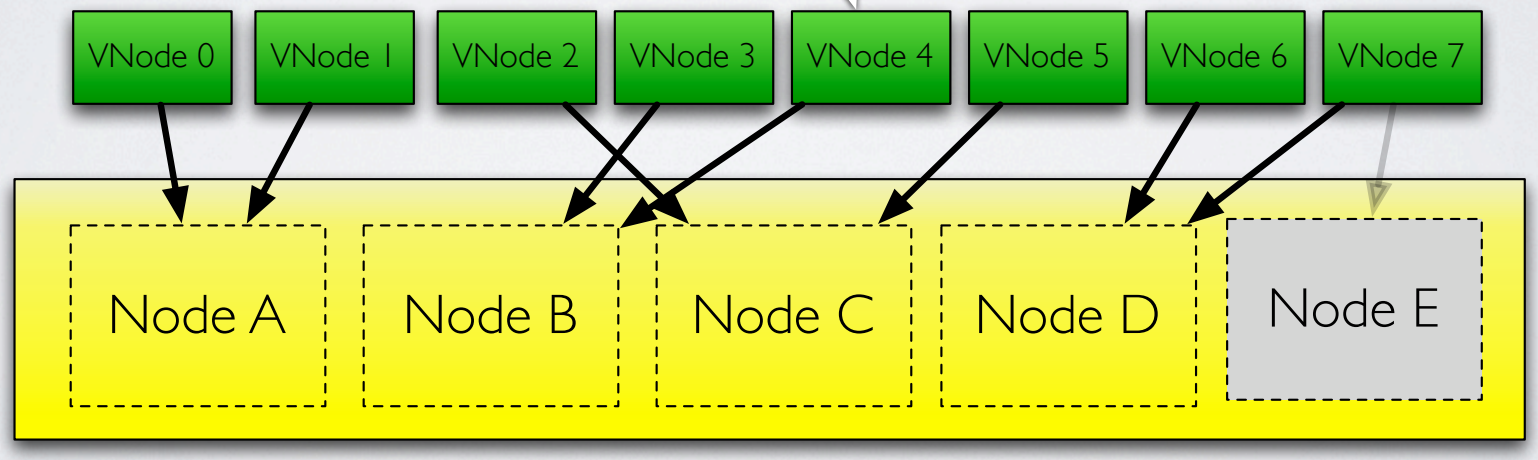
basho

riak

Removing a Node

Command → `ObjectName, Payload`

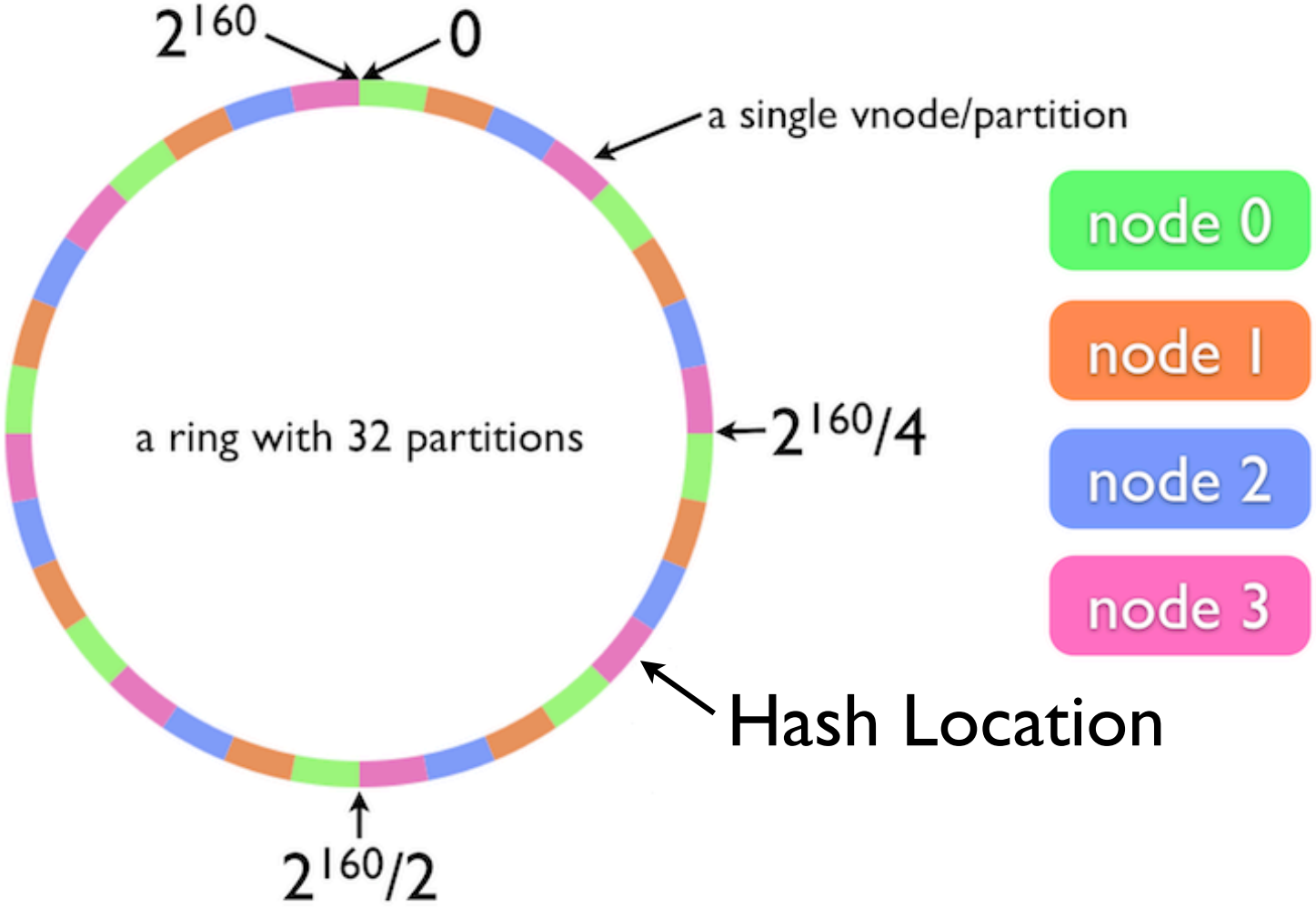
`SHA1(ObjName), Payload`



basho

riak

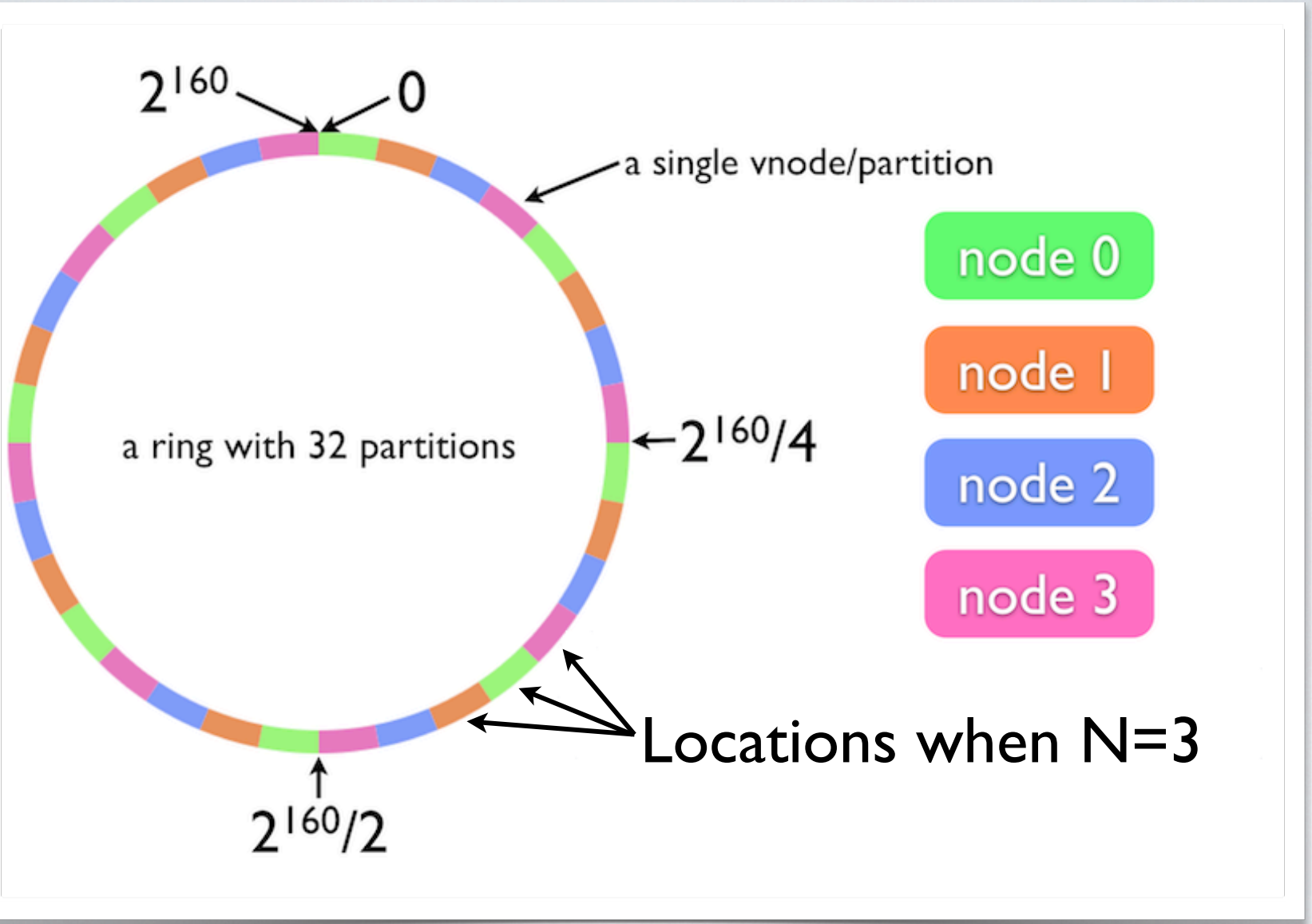
The Ring



basho

riak

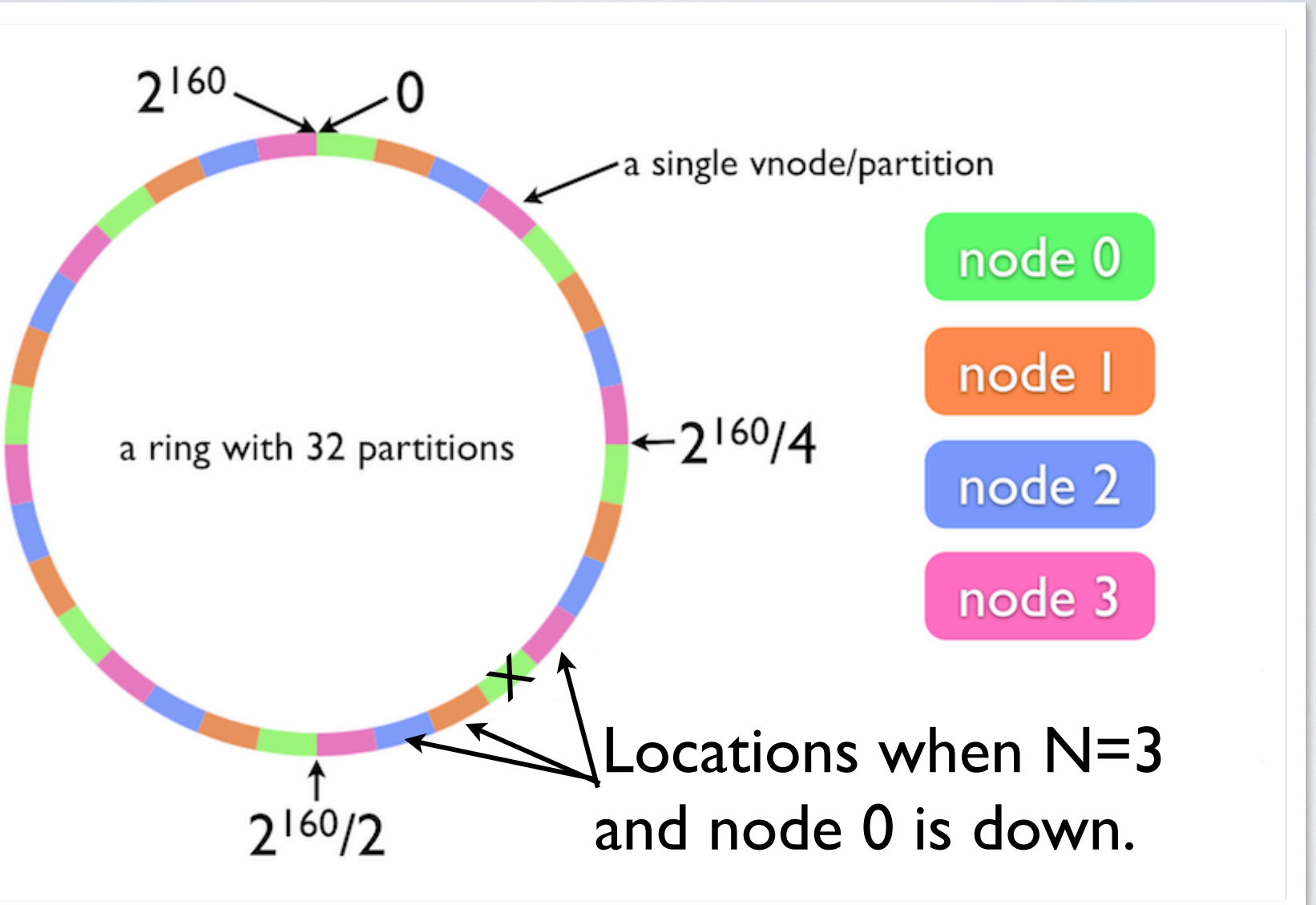
Writing Replicas (N Value)



basho

riak

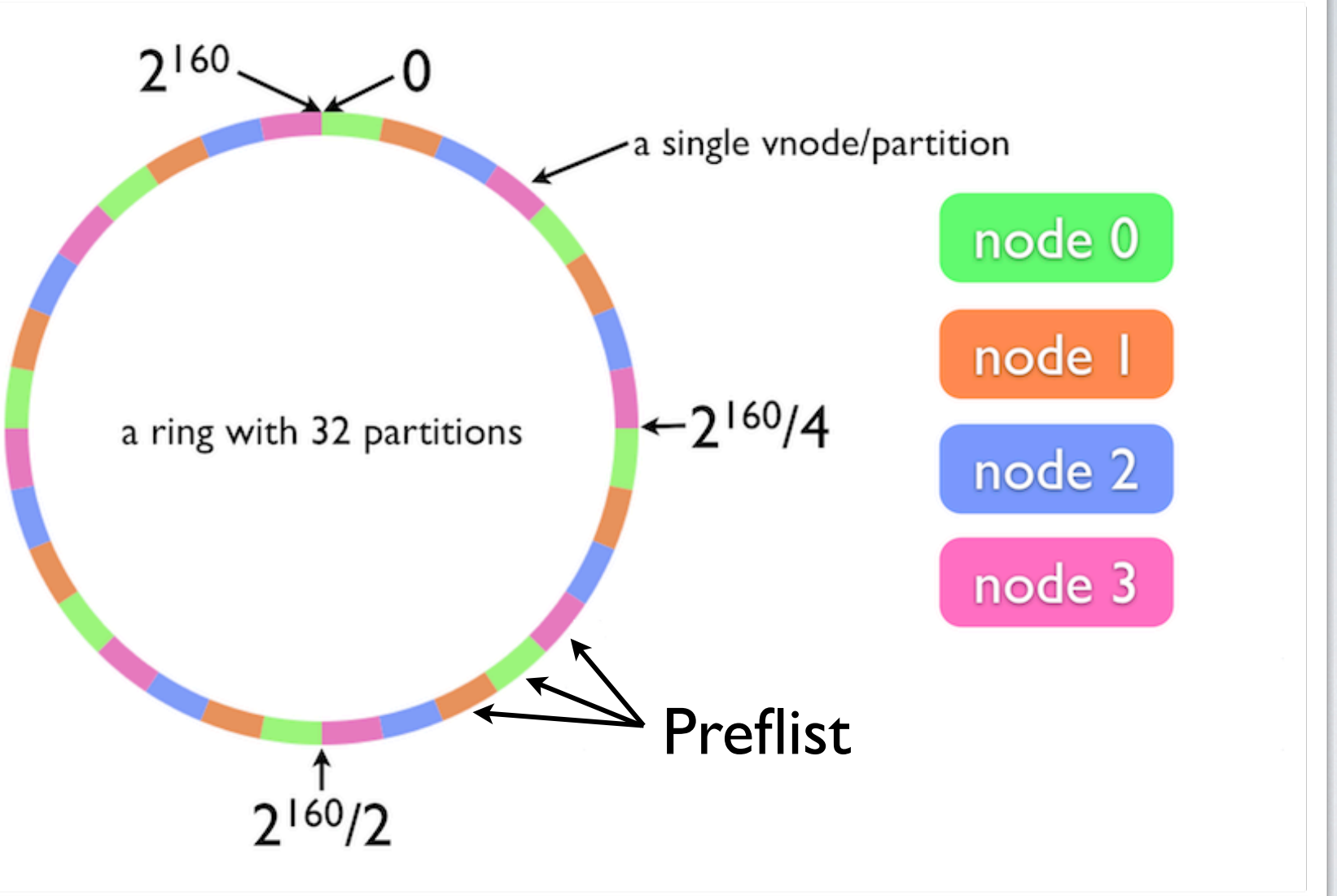
Routing Around Failures



basho

riak

The Preflist

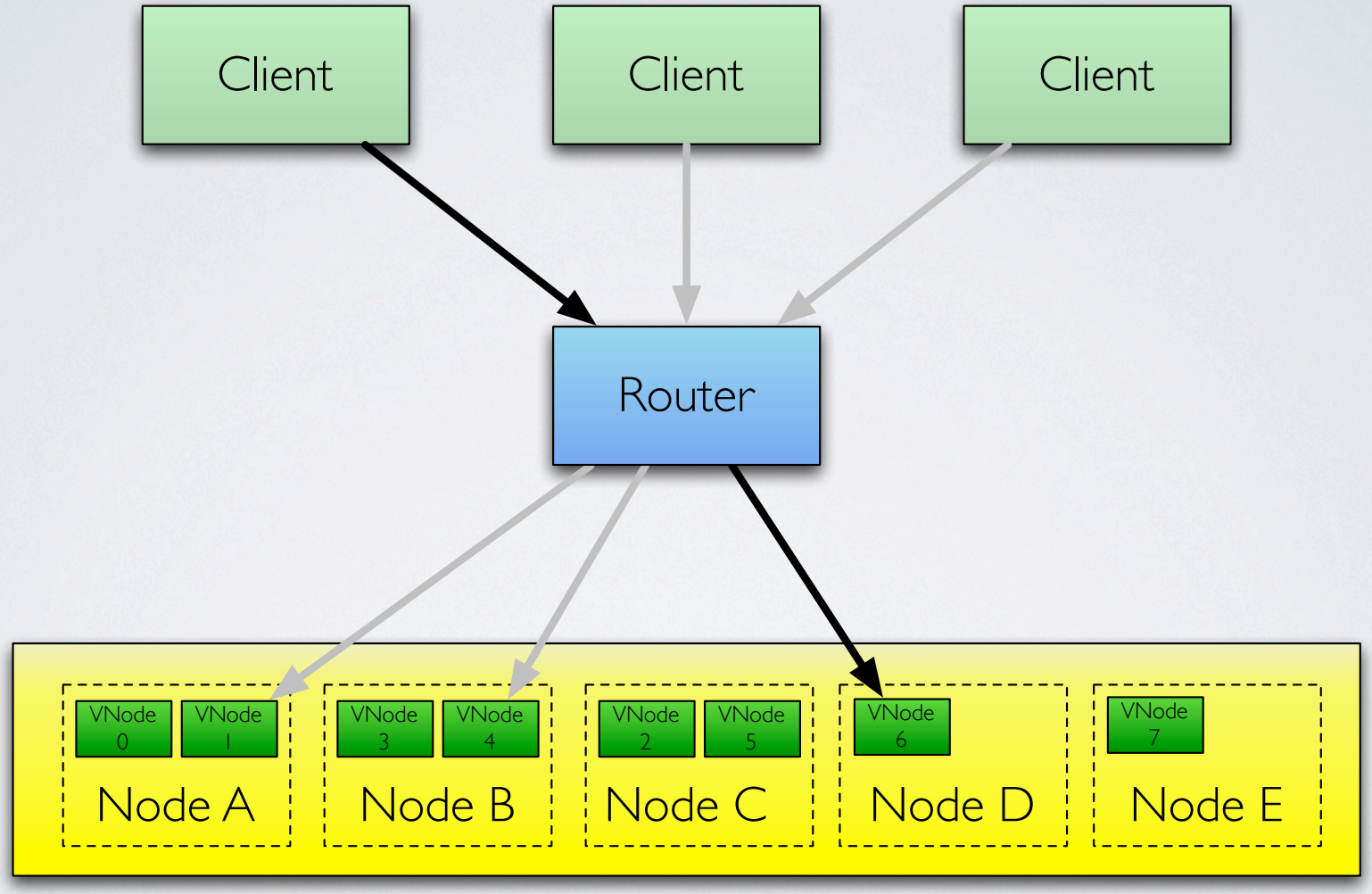


basho

riak

Location of the Routing Layer

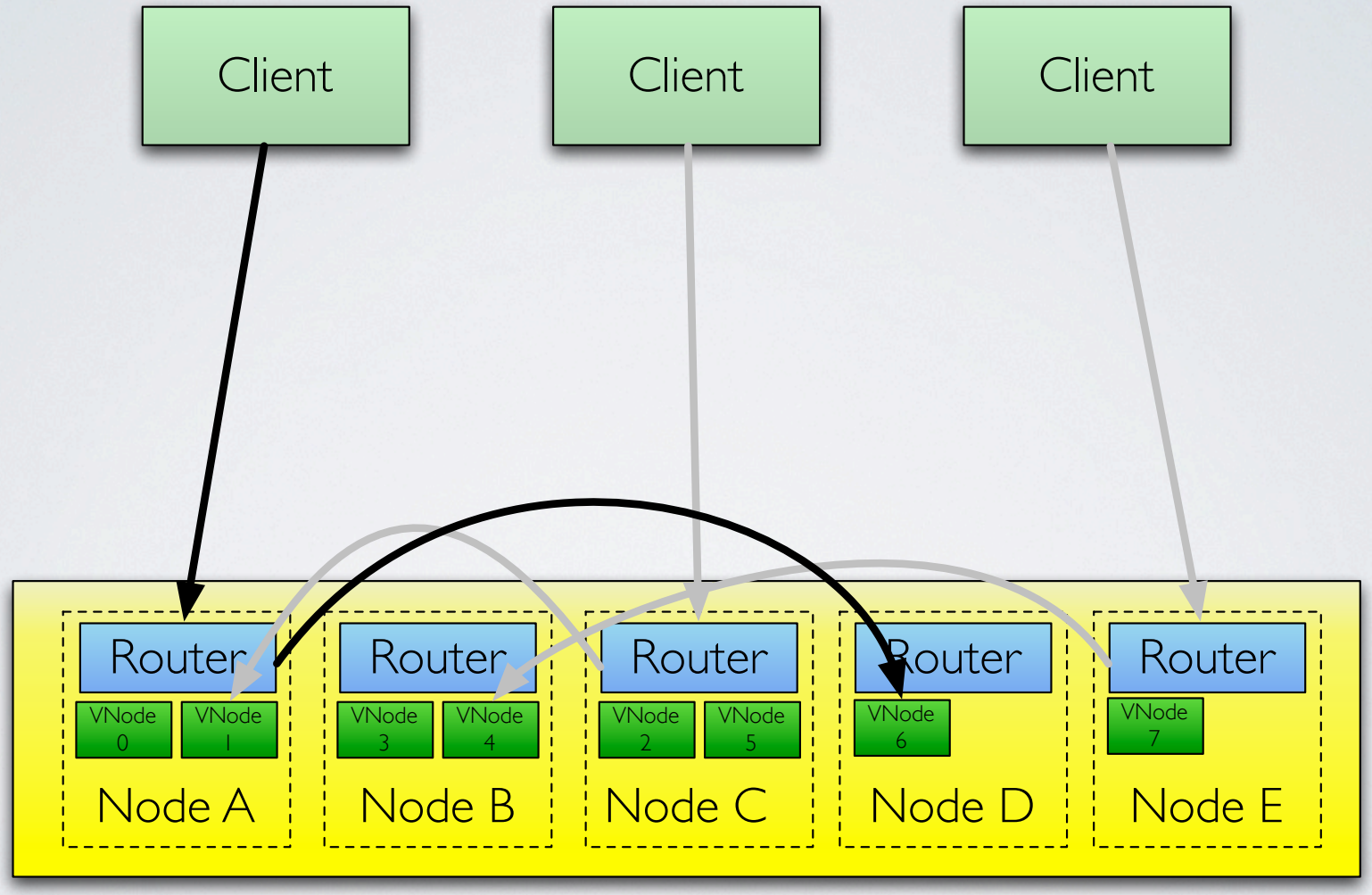
Router in the Middle Leads to SPOF



basho

riak

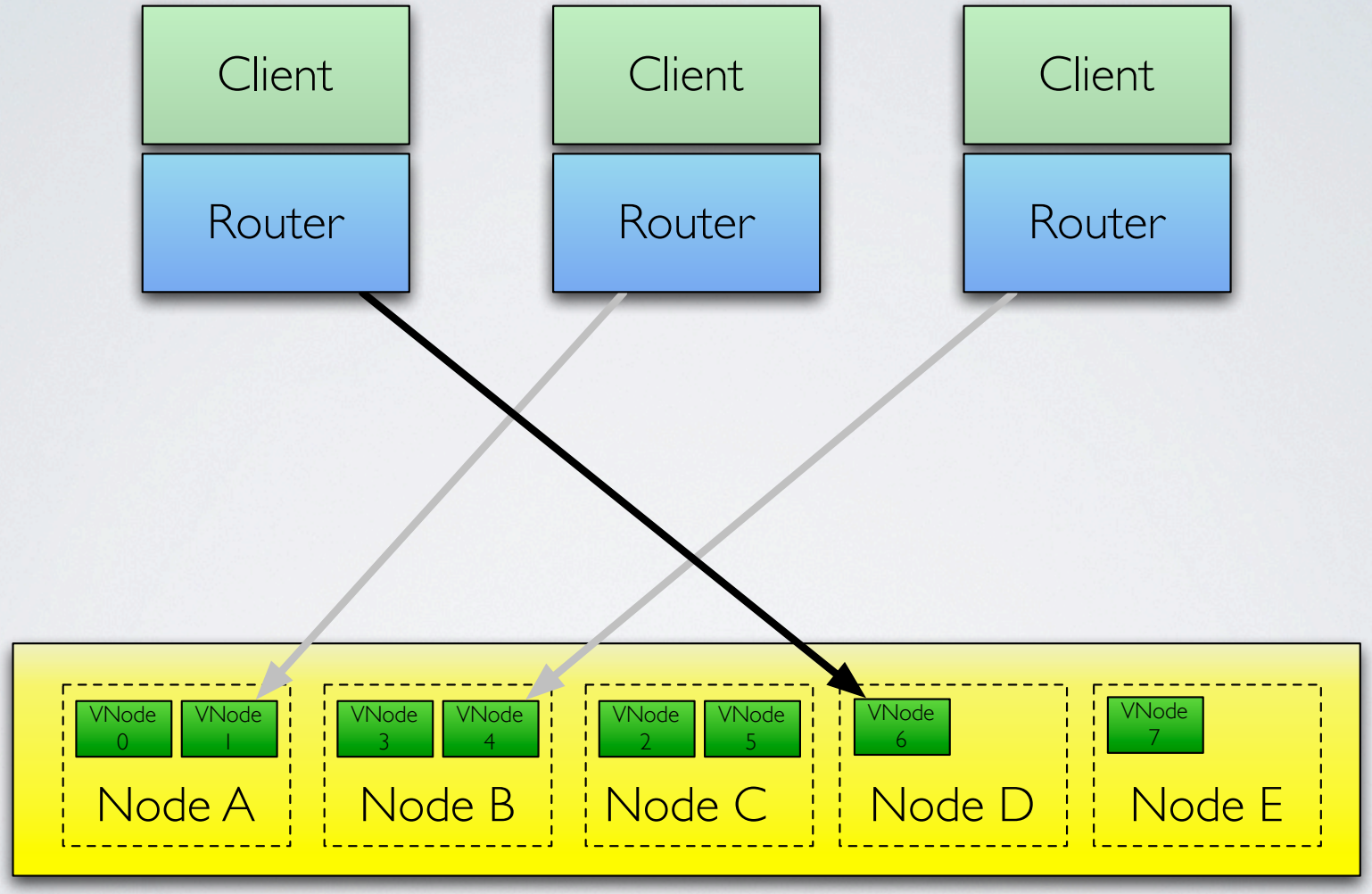
Riak Core - Router on Each Node



basho

riak

Eventually - Router in the Client

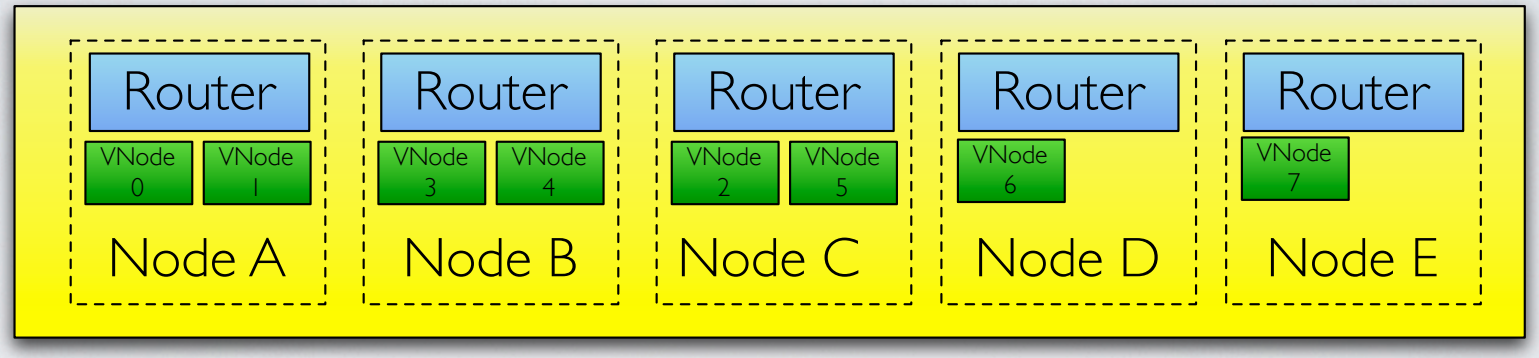


Why isn't this done yet?
Time and complexity.

basho

riak

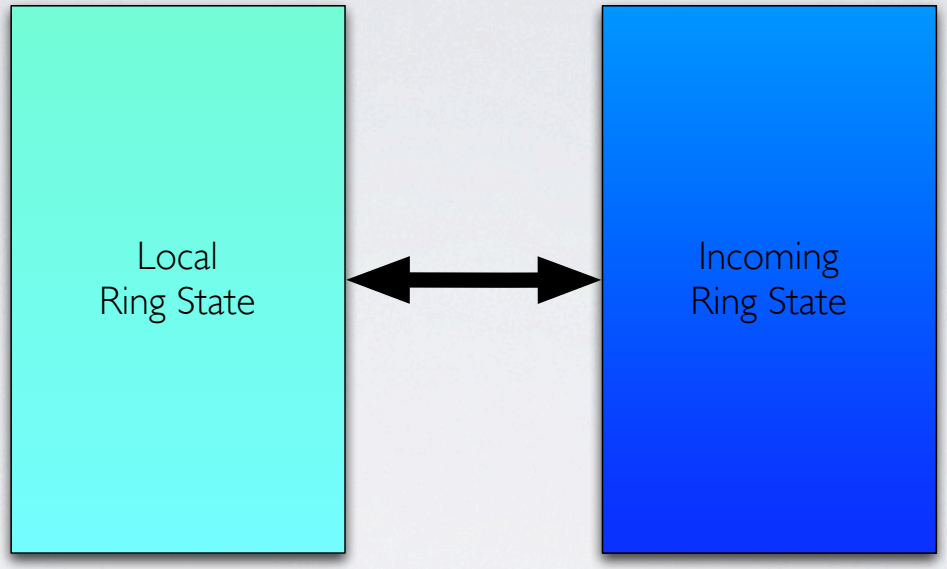
How Do The Routers Reach Agreement?



basho

riak

The Nodes Gossip Their World View



Are rings equivalent?

Strictly descendent?

Or different?

basho

riak

Not Mentioned

Vector Clocks

Merkle Trees

Bloom Filters

Building an Application with Riak Core

Building an Application on Riak Core? Two things to think about:

The Command Set

Command = ObjectName, Payload

The commands/requests/operations that you will send through the system.

The VNode Module

The callback module that will receive the commands.

Writing a VNode Module

Startup/Shutdown

```
init([Partition]) ->
  {ok, State}

terminate(State) ->
  ok
```

Receive Incoming Commands

```
handle_command(Cmd, Sender, State) ->
  {noreply, State1} | {reply, Reply, State1}

handle_handoff_command(Cmd, Sender, State) ->
  {noreply, State1} | {reply, ok, State1}
```

Writing a VNode Module

Send and Receive Handoff Data

```
handoff_starting(Node, State) ->
 {Bool, State1}

encode_handoff_data(Data, State) ->
 <<Binary>>.

handle_handoff_data(Data, Sender, State) ->
 {reply, ok, State1}

handoff_finished(Node, State) ->
 {ok, State1}
```

Start the riak_core application


```
application:start(riak_core).
```

Start the riak_core application

Supervise vnode processes.

Start the riak_core application

Start, coordinate, and supervise handoff.

Start the riak_core application

Maintain cluster membership information.

Start the riak_core application

Monitor node liveness,

broadcast to registered modules.

Start the riak_core application

Send ring information to other nodes.

Reconcile different views of the cluster.

Rebalance cluster when nodes join or leave.

In your application...

Start the vnodes for your application.

```
Master = {
    riak_X_vnode_master, {
      riak_core_vnode_master, start_link, [riak_X_vnode]
    },
    permanent, 5000, worker, [riak_core_vnode_master]
},
{ok, { {one_for_one, 5, 10}, [Master]} }.
```

basho

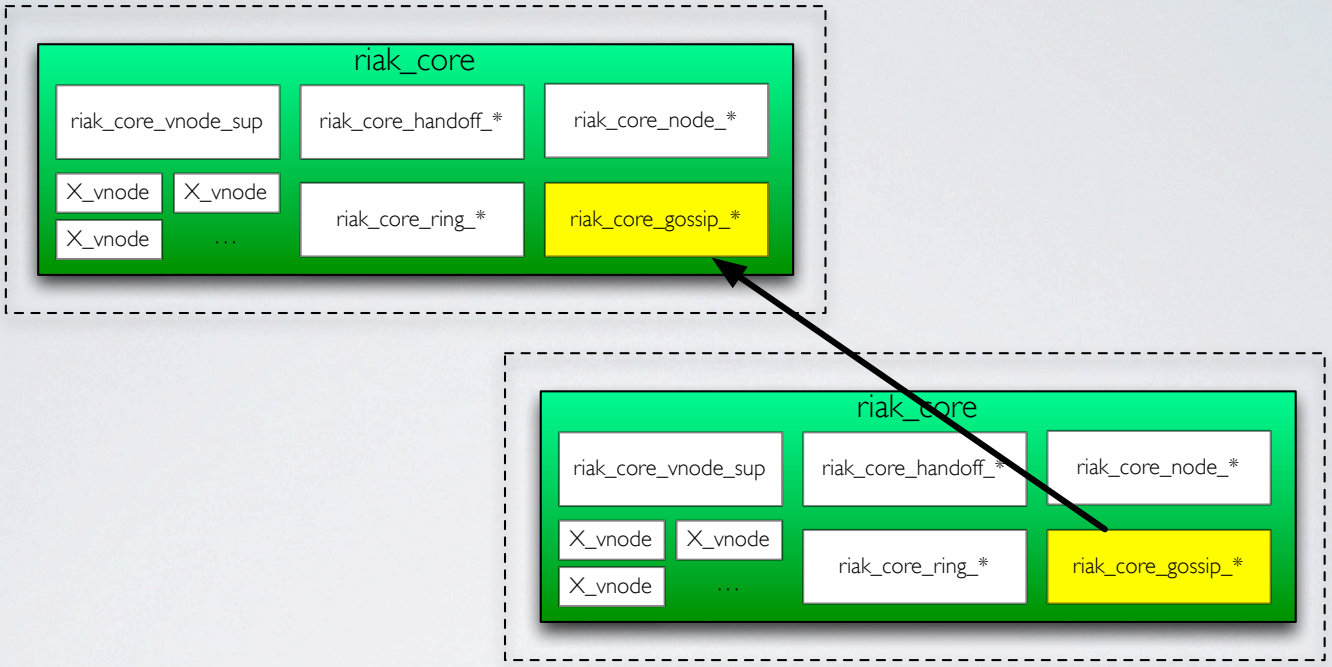
riak

In your application...

Tell riak_core that your application is ready to receive requests.

```
riak_core:register_vnode_module(riak_X_vnode),
riak_core_node_watcher:service_up(riak_X,
                                  self())
```

In your application...



Join to an existing node in the cluster.

```
riak_core_gossip:send_ring(ClusterNode,
                           node())
```

Start Sending Commands

```
# Figure out the preflist...
{_Verb, ObjName, _Payload} = Command,
PrefList = riak_core_apl:get_apl(ObjName,
                                 NVal,
                                 riak_X),

# Send the command...
riak_core_vnode_master:command(PrefList,
                               Command,
                               riak_X_vnode_master)
```

Review

Riak KV

Open Source Key/Value datastore.

Riak Search

Full-text, near real-time search engine based on Riak Core.

Riak Core

Open Source Erlang library that helps you build distributed, scalable, failure-tolerant applications using a Dynamo-style architecture.

Thanks! Questions?

Learn More

http://wiki.basho.com

Read Amazon's Dynamo Paper

Get the Code

http://github.com/basho/riak_core

Get in Touch

rusty@basho.com on Email

@rklophaus on Twitter

basho

riak

END