Building Masterless Distributed Applications

Glue Conference
May 2011

Joseph Blomstedt (@jtuple)
Basho Technologies

basho

riak

Riak is
a scalable, highly-available, distributed
open-source key/value store.

basho

riak

Riak is
a scalable, highly-available, distributed
open-source key/value store.

...but that's not what I'm here to tell you about.

Some features of Riak KV & other dynamo-style DBs:

- simple interface (get, put, delete, list, etc)
- extremely high write-availability
- linear scaling of both capacity and throughput

basho

riak

Riak KV
& other dynamo-style DBs
achieve this by standing on the shoulders of:

- consistent hashing
- vector clocks
- sloppy quorums
- gossip protocols
- virtual nodes
- hinted handoff

basho

riak

None of this has much to do with k/v data.

- consistent hashing
- vector clocks
- sloppy quorums
- gossip protocols
- virtual nodes
- hinted handoff

basho

riak

This is riak core.

- consistent hashing
- vector clocks
- sloppy quorums
- gossip protocols
- virtual nodes
- hinted handoff

basho

riak

Riak KV

Riak Core

Riak Search

Distribution / Scaling / Failure-Tolerance Code

basho

riak

Riak Core is an

Open Source Erlang library

that helps you build

distributed, scalable, failure-tolerant

applications using a

Dynamo-style architecture.

basho

riak

Distributed, scalable, failure-tolerant.

Distributed, scalable, failure-tolerant.

No central coordinator.

Easy to setup/operate.

basho

riak

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.

basho

riak

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

The Command Set

Command = ObjectName, Payload

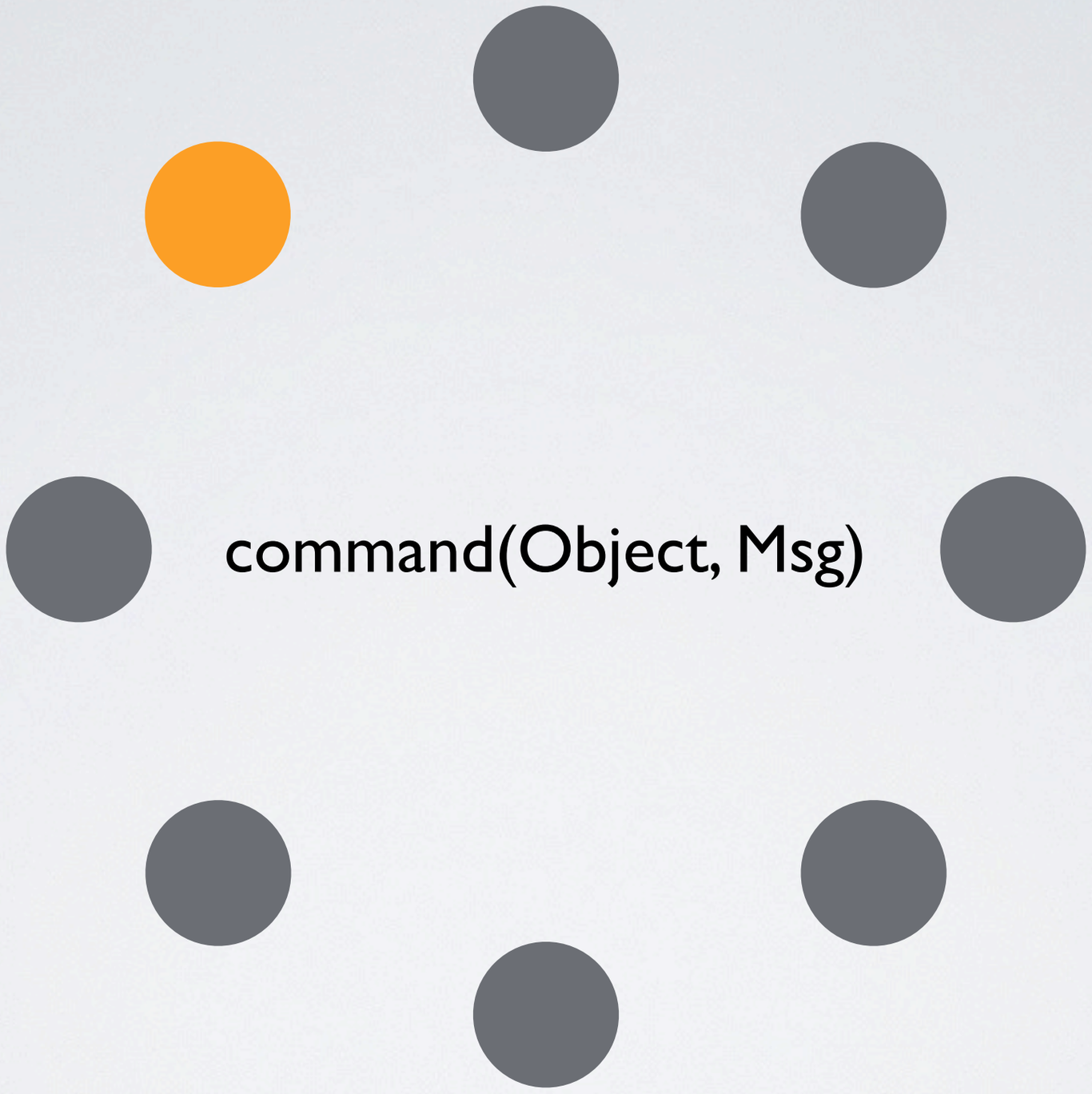
The commands/requests/operations that you will send through the system.

The VNode Module

The callback module that will receive the commands.

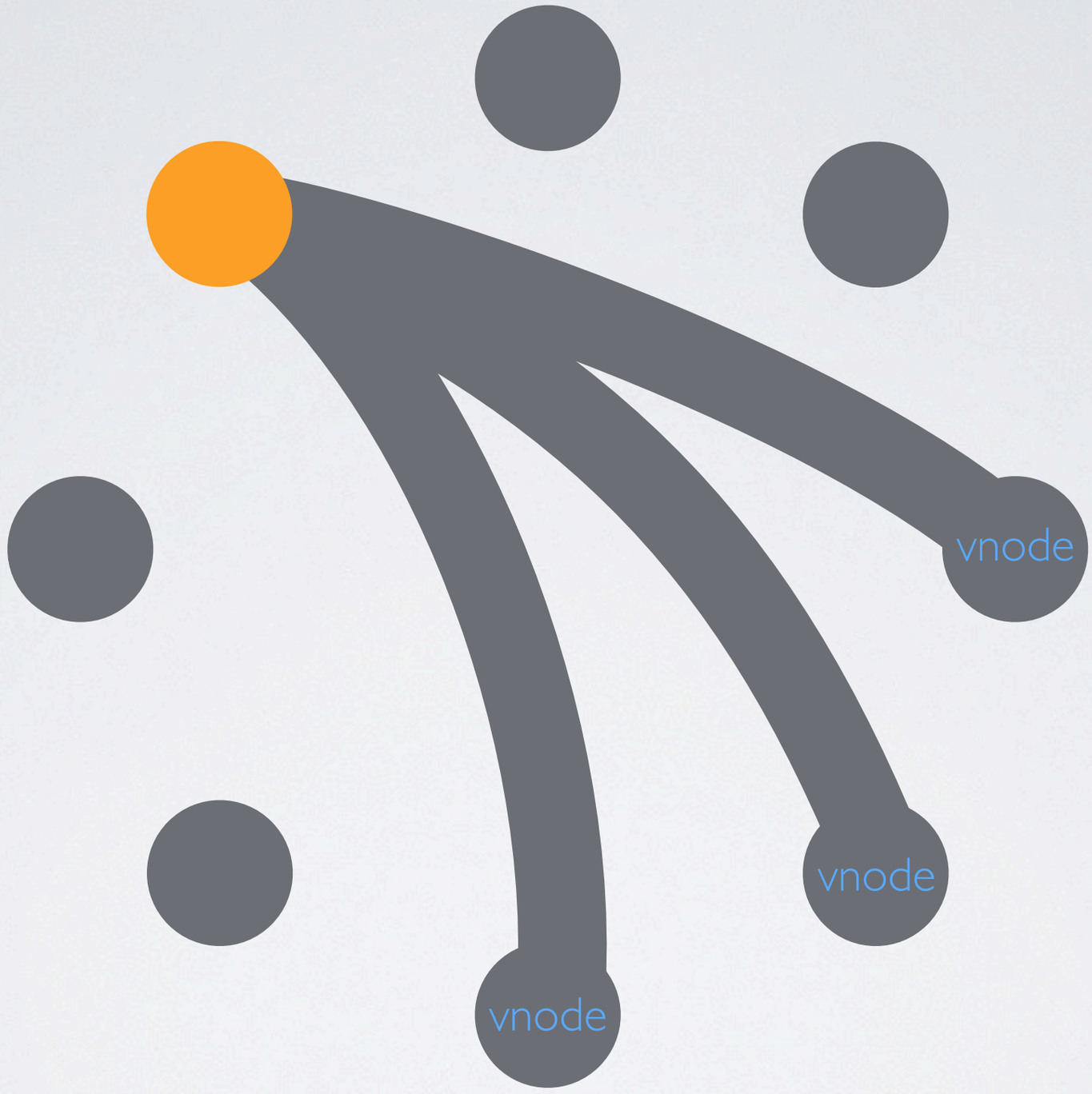
basho

riak



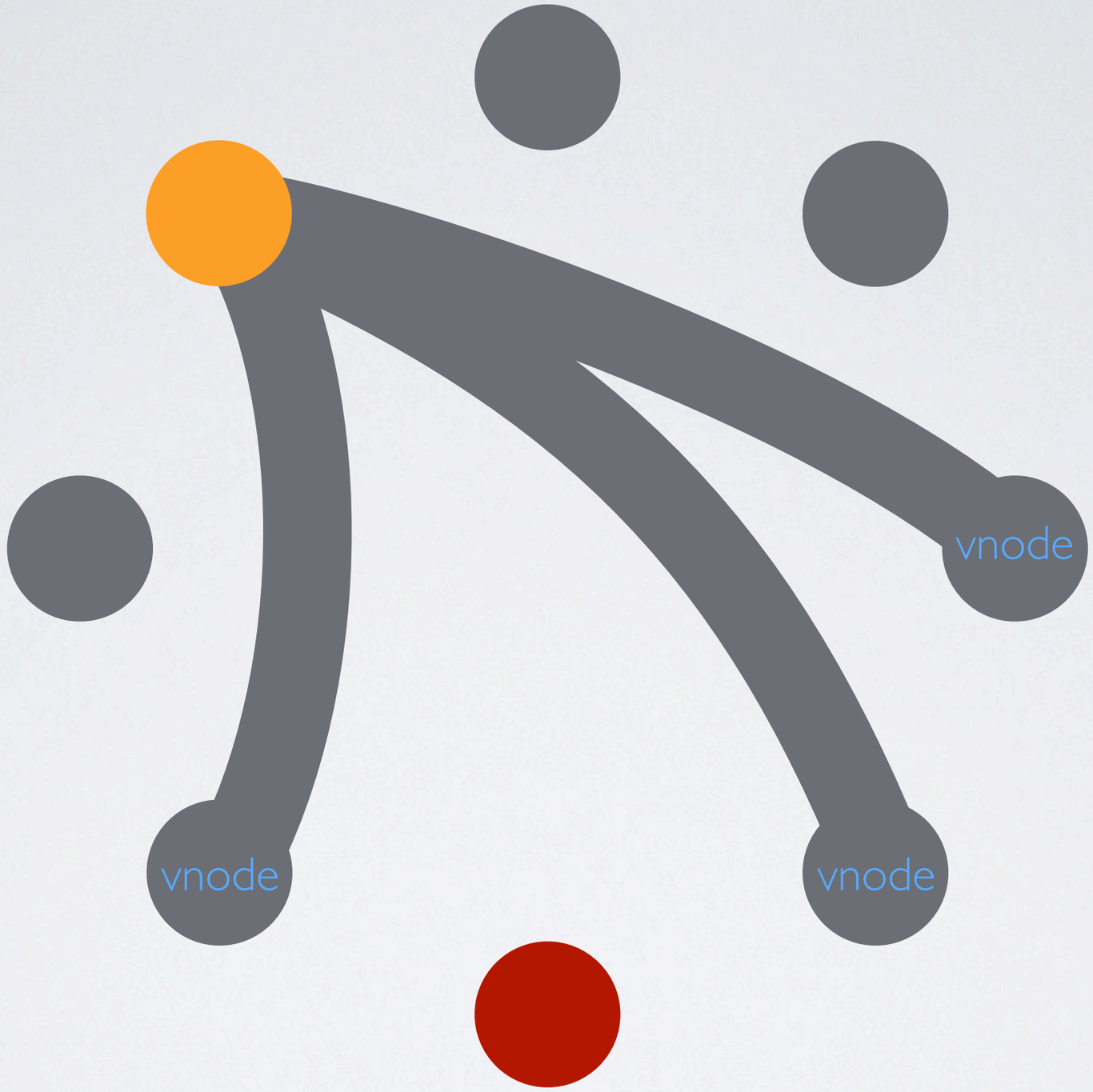
basho

riak



basho

riak



basho

riak

basho

riak

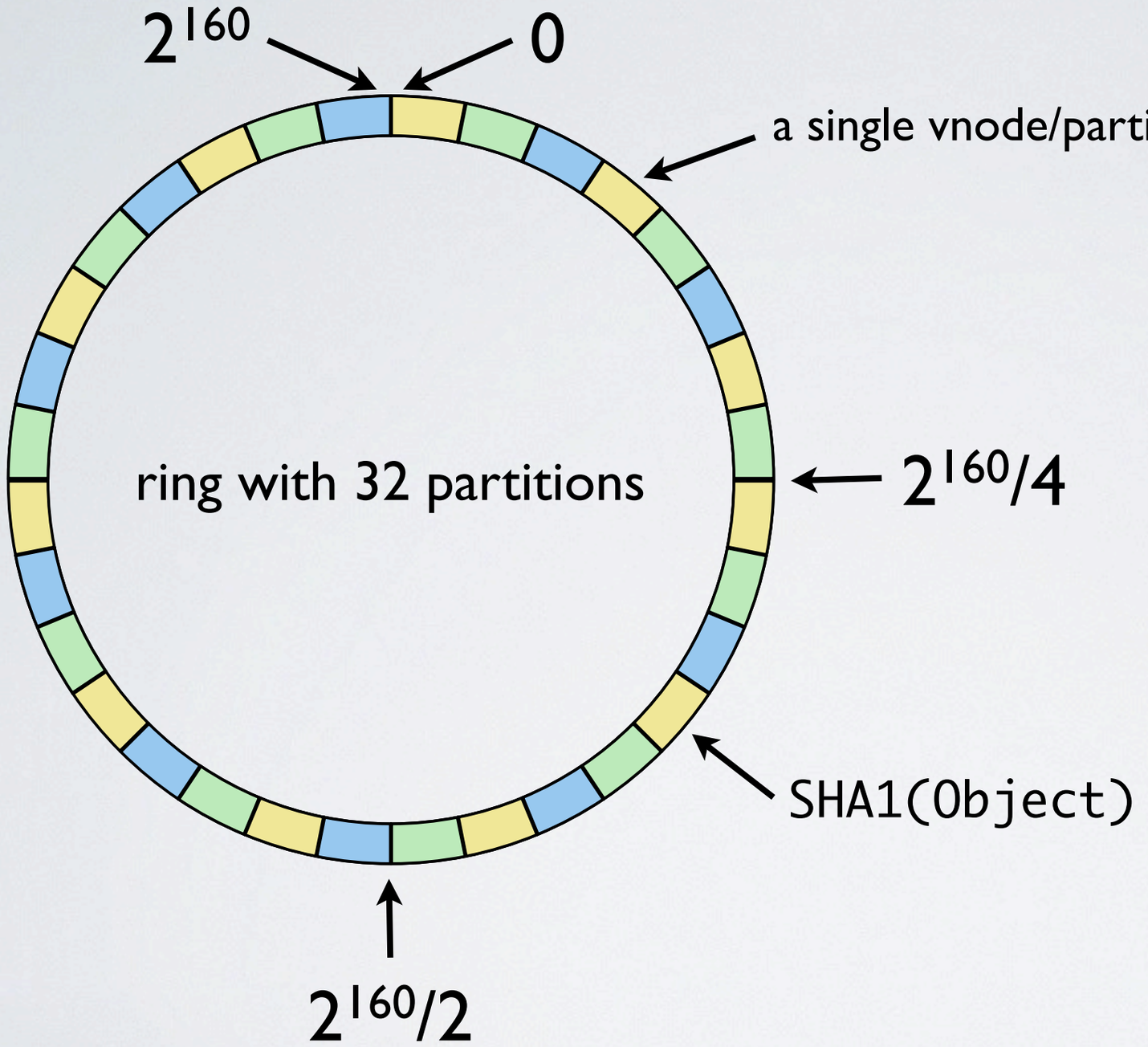
Core Dynamo Principals

- Scalable
  - Consistent hashing
  - Virtual nodes
- Highly Available + Self Healing
  - Always writeable
  - Sloppy quorums
  - Hinted handoff
- All Nodes Equal / Masterless
  - Gossip protocol

basho

riak

Consistent Hashing



Node 0

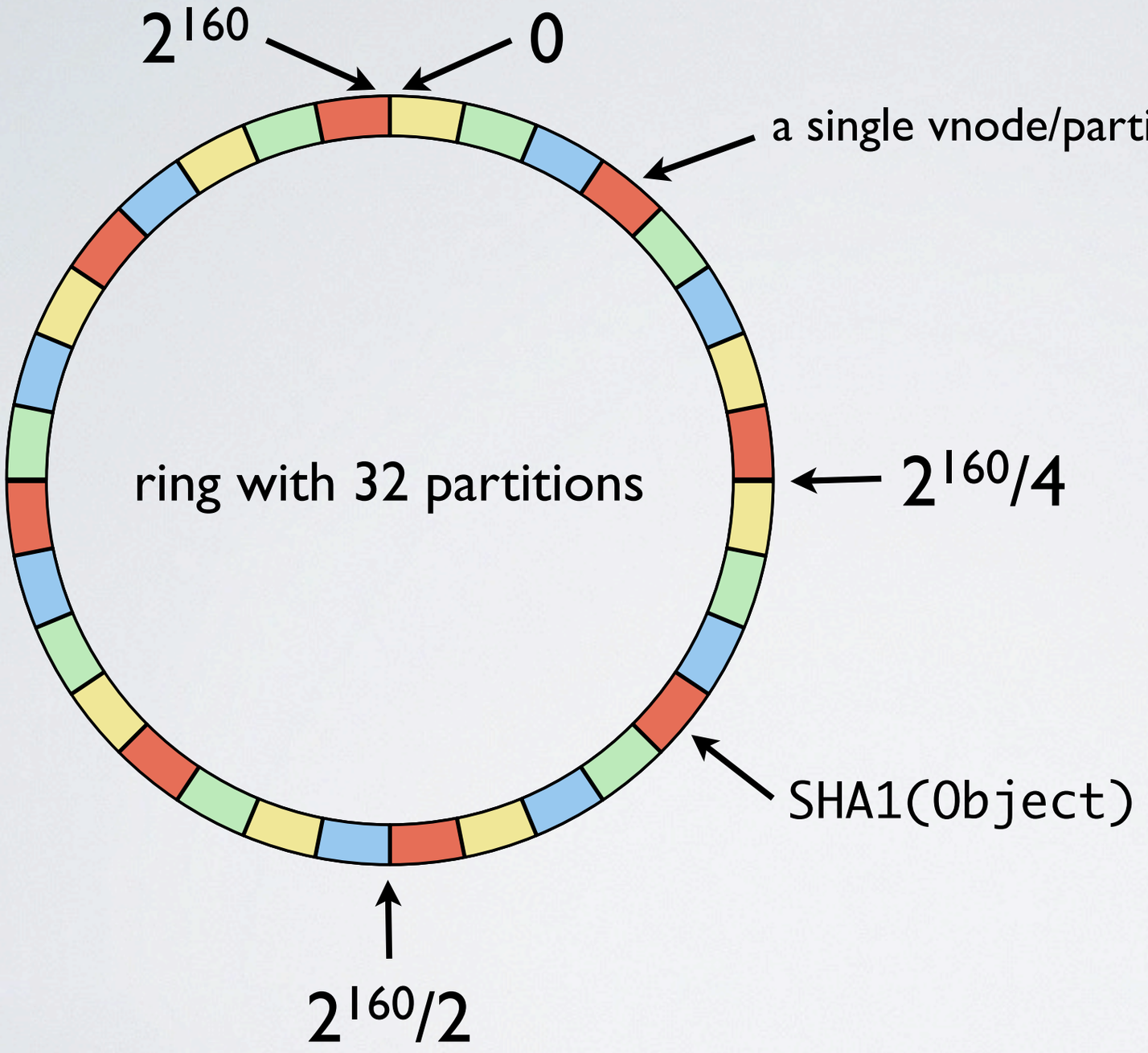
Node 1

Node 2

basho

riak

Adding a node



Node 0

Node 1

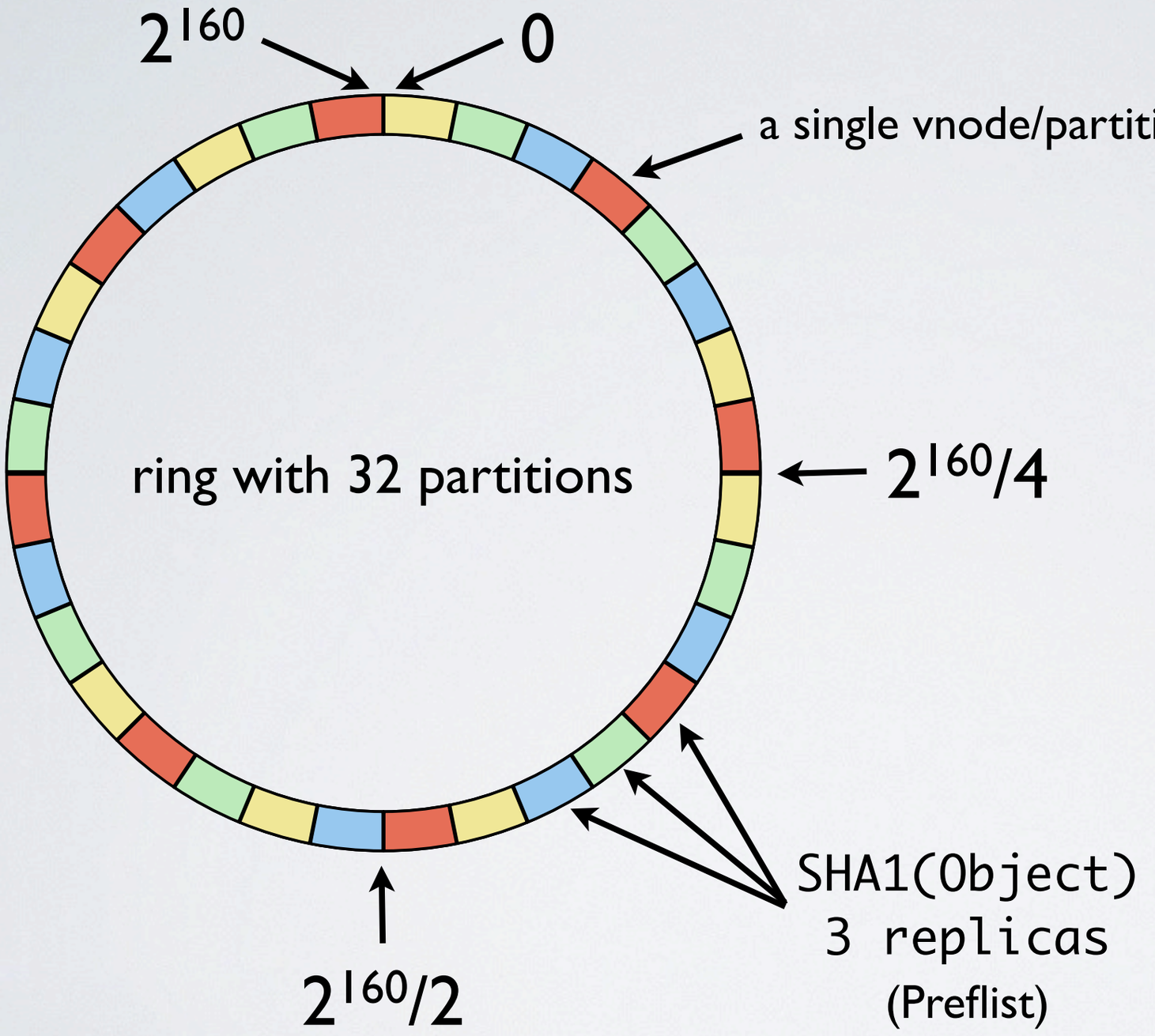
Node 2

Node 3

basho

riak

Replication



Node 0

Node 1

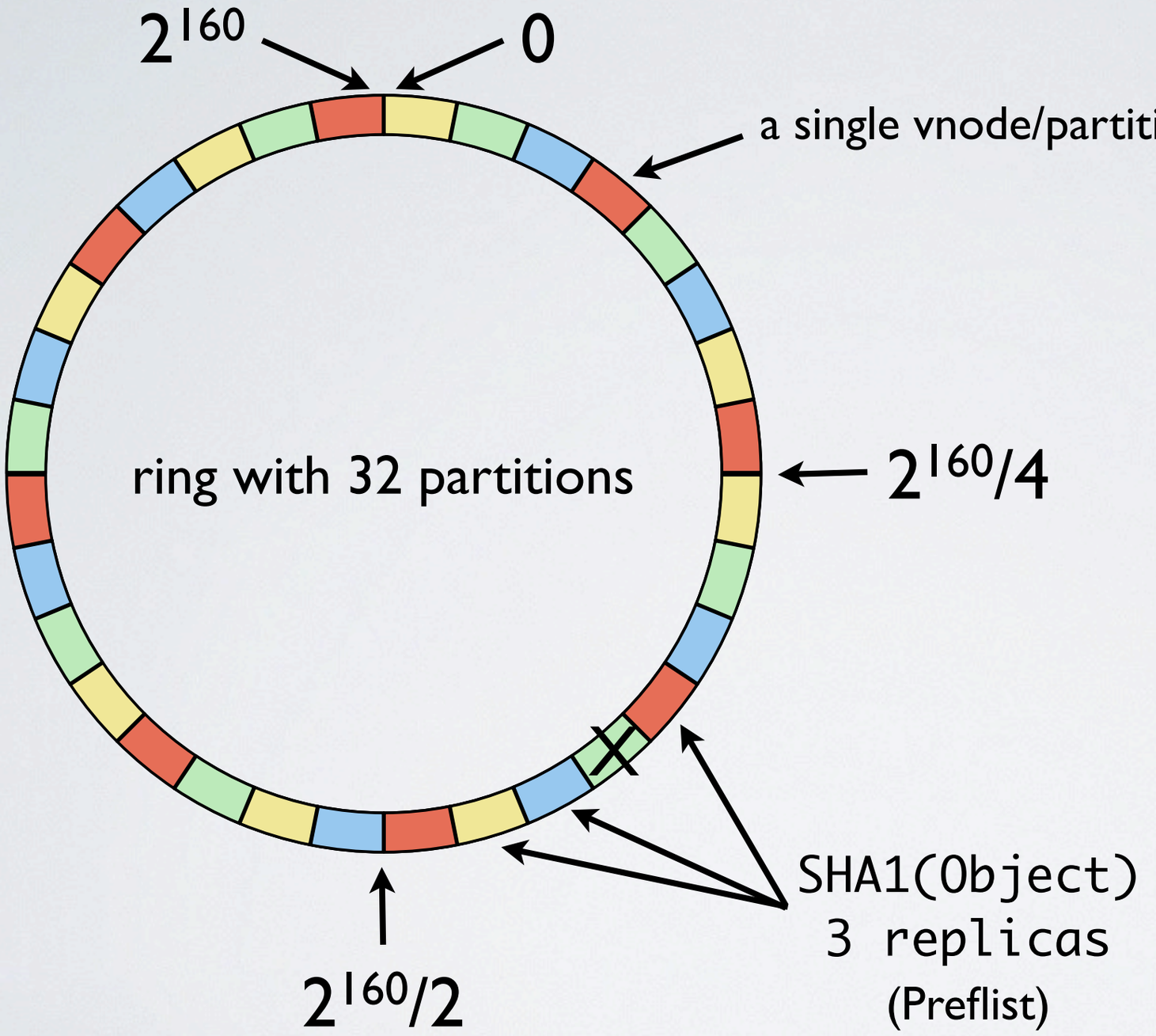
Node 2

Node 3

basho

riak

Routing around failures



Node 0

Node 1

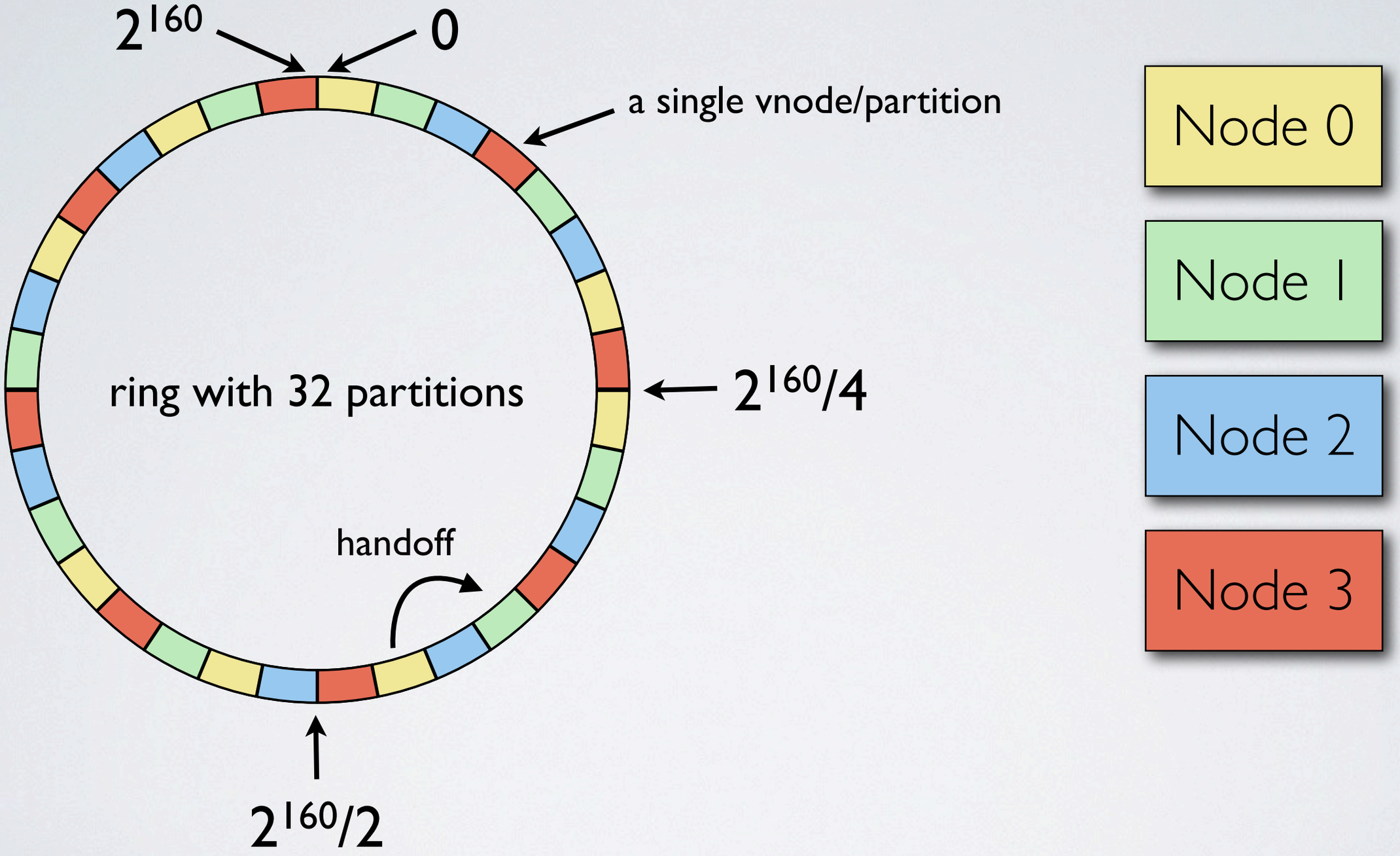
Node 2

Node 3

basho

riak

Hinted Handoff



basho

riak

Let's build something!

Simple Message Server

- Send messages

  send_message(User, Msg)

- Retrieve messages

  get_messages(User)

Get Preflist, Including Fallback Nodes

```
get_preflist(User) ->
  Idx = chash:key_of(User),
  riak_core_apl:get_apl(Idx, 3, chat).
```

Public Chat Server API

```
-define(?CHAT, chat_vnode_master).

send_message(User, Msg) ->
  PL = get_preflist(User),
  Cmd = {send, User, Msg}
  riak_core_vnode_master:command(PL, Cmd, ?CHAT).

get_messages(User) ->
  PL = get_preflist(User),
  riak_core_vnode_master:sync_command(PL,
                                      {get, User},
                                      ?CHAT).
```

Chat Server VNode Module

Startup/Shutdown

```
init([Partition]) ->
  {ok, dict:new()}

terminate(State) ->
  ok
```

Chat Server VNode Module

Handle Send Commands

```
handle_command({send, User, Msg},
                    Sender, State) ->
  State2 = dict:append(User, Msg, State),
  {noreply, State2}.
```

Handle Get Commands

```
handle_command({get, User}, Sender, State) ->
  Msgs = dict:fetch(User, State),
  {reply, Msgs, State}.
```

Chat Server VNode Module

Handling Handoff

```
handle_handoff_command(?FOLD_REQ{foldfun=Fun,
                                 acc0=Acc0},
                       _Sender, State) ->
  Reply = dict:fold(Fun, Acc0, State),
  {reply, Reply, State};

handoff_starting(_Node, State) ->
  {true, State}.

handoff_cancelled(State) ->
  {ok, State}.

handoff_finished(_TargetNode, State) ->
  {ok, State}.
```

Chat Server VNode Module

```
handle_handoff_data(BinObj, State) ->
  {K, V} = binary_to_term(BinObj),
  State2 = dict:store(K, V, Logs),
  {reply, ok, State2}.

encode_handoff_item(K, V) ->
  term_to_binary({K,V}).

is_empty(State) ->
  Empty = (dict:size(Logs) == 0),
  {Empty, State}.

delete(State) ->
  {ok, dict:new()}.
```

Chat Server

- Fairly straight forward chat server

  Define commands

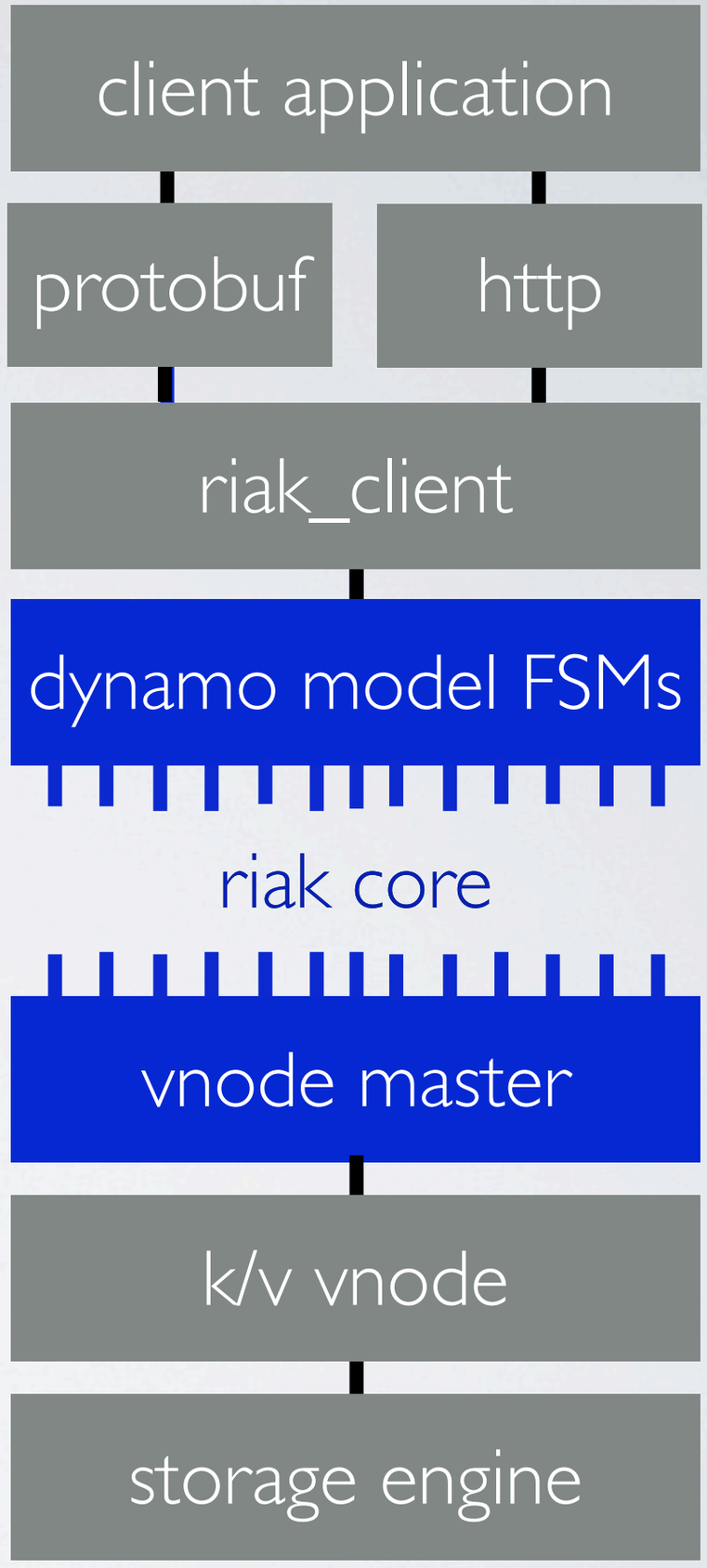
  Implement commands as a callback

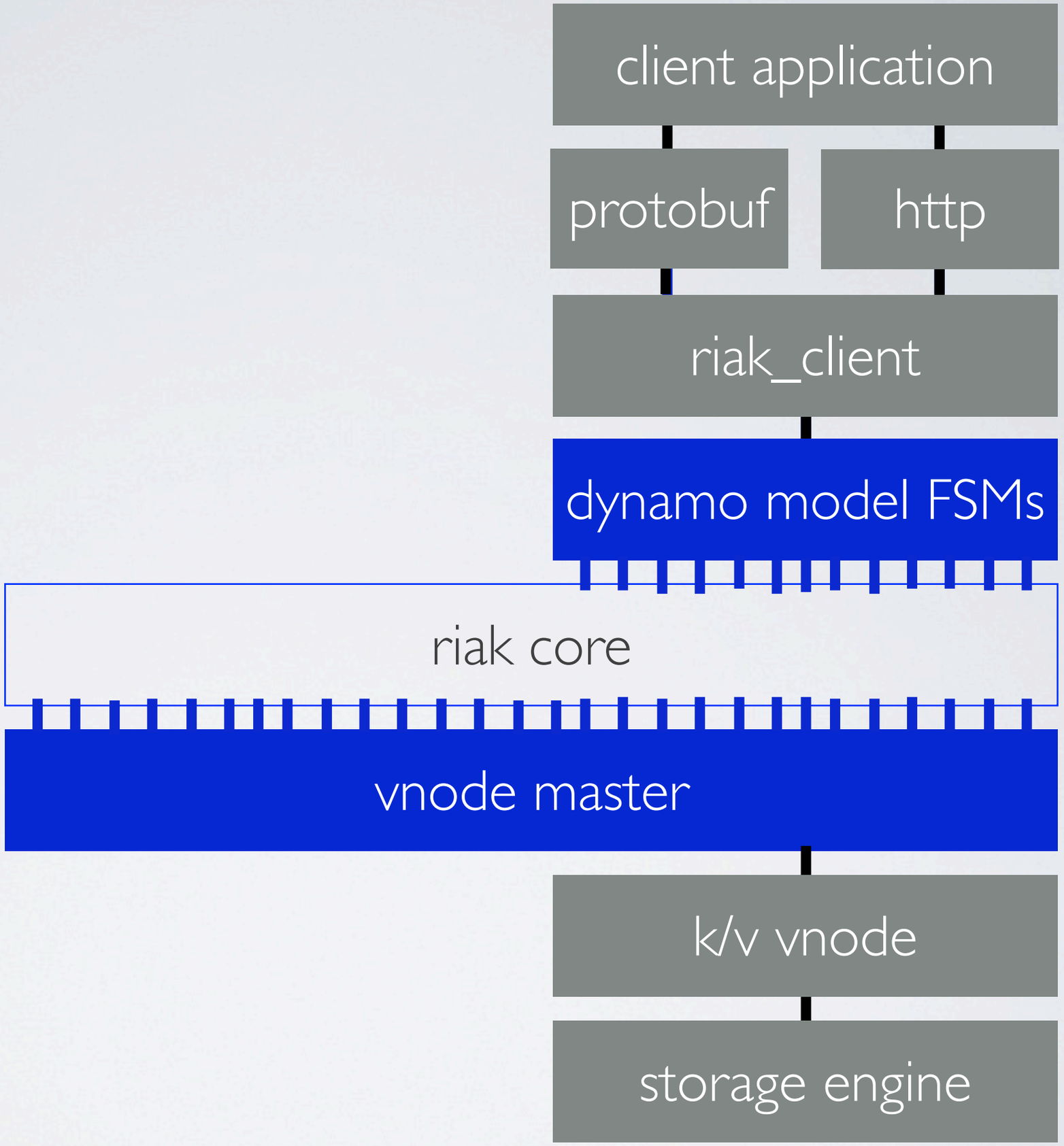
  Provide support for shuffling data between vnodes

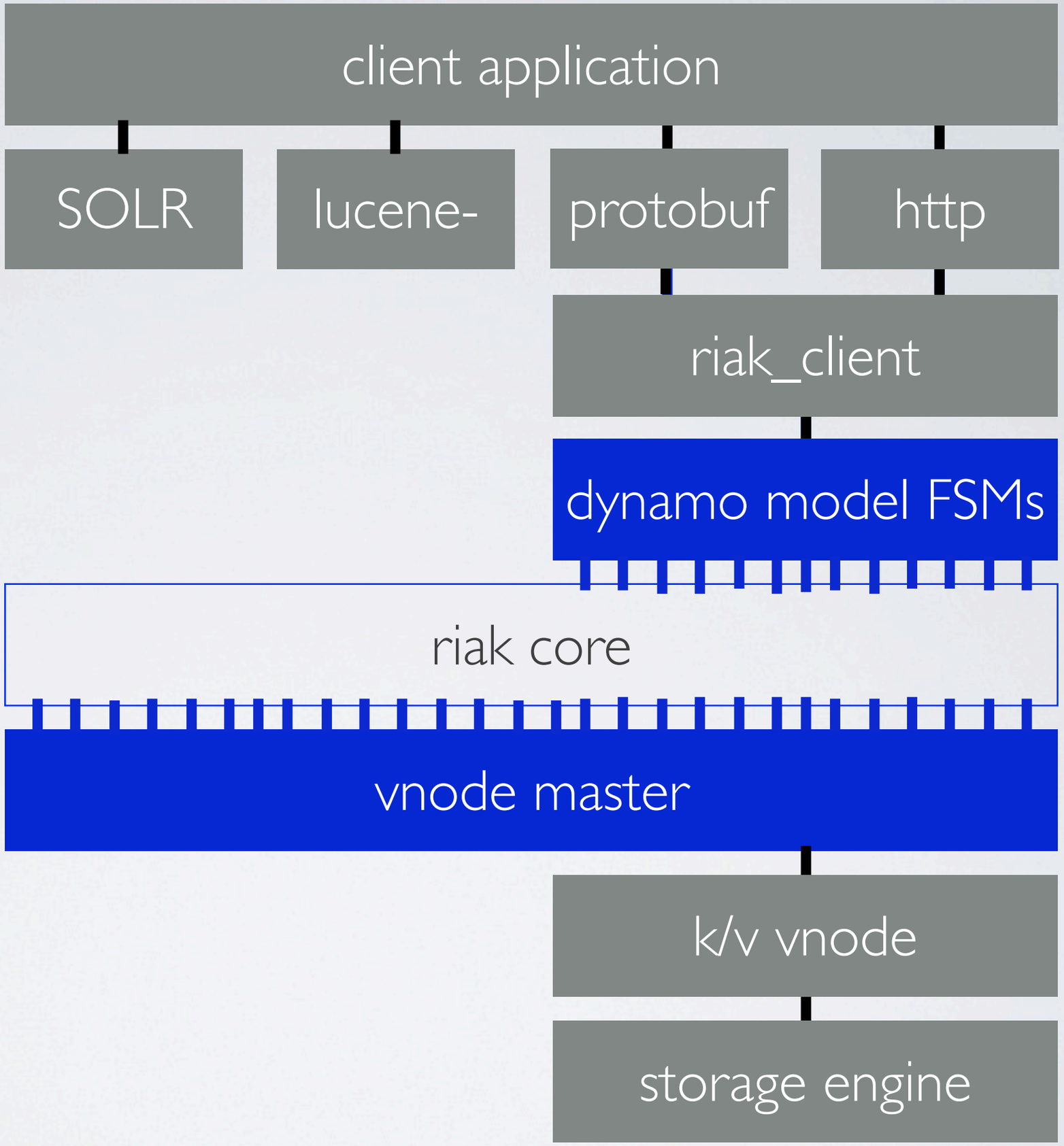
- Low complexity

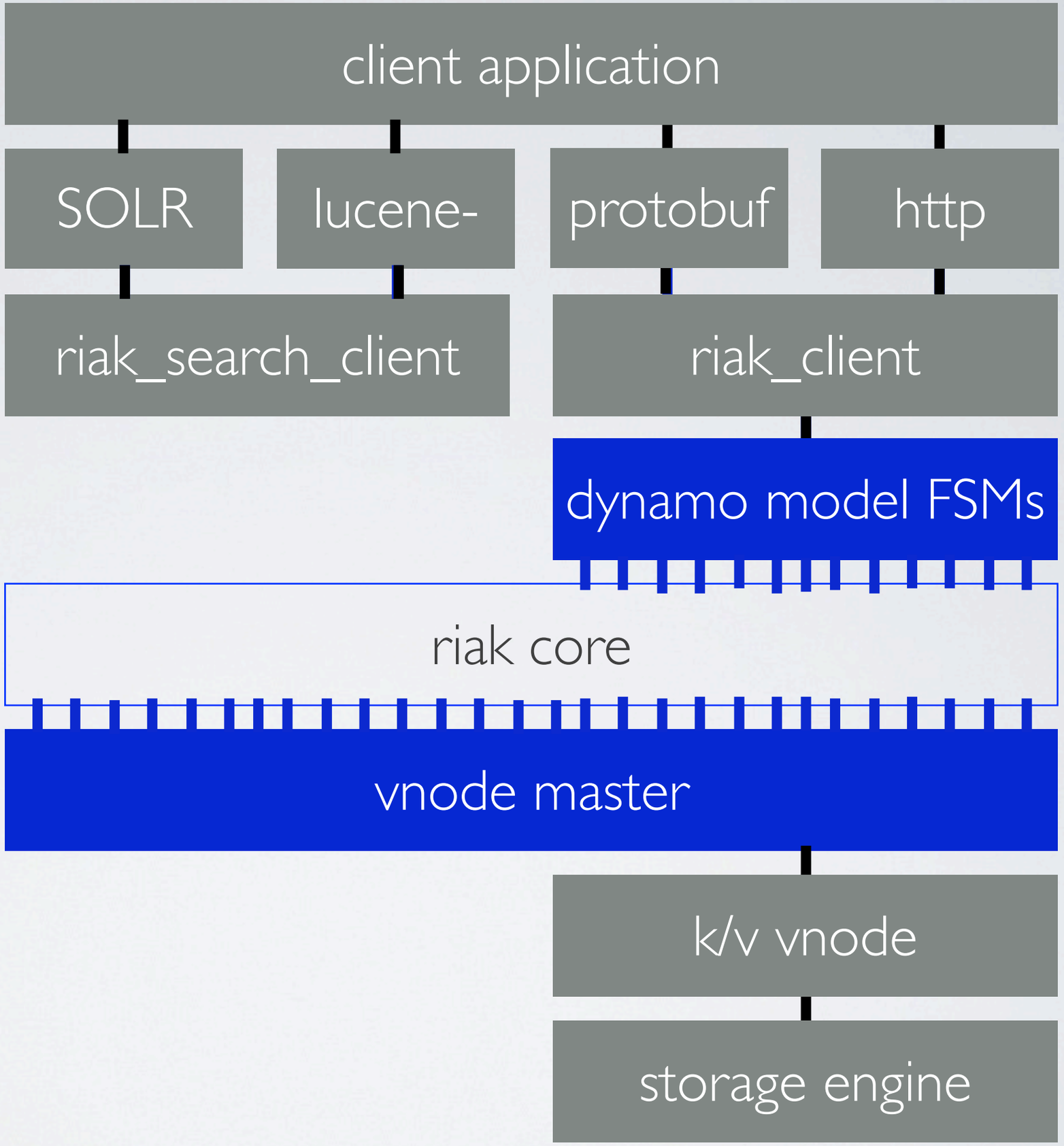
- Distributed, fault tolerance for "free"

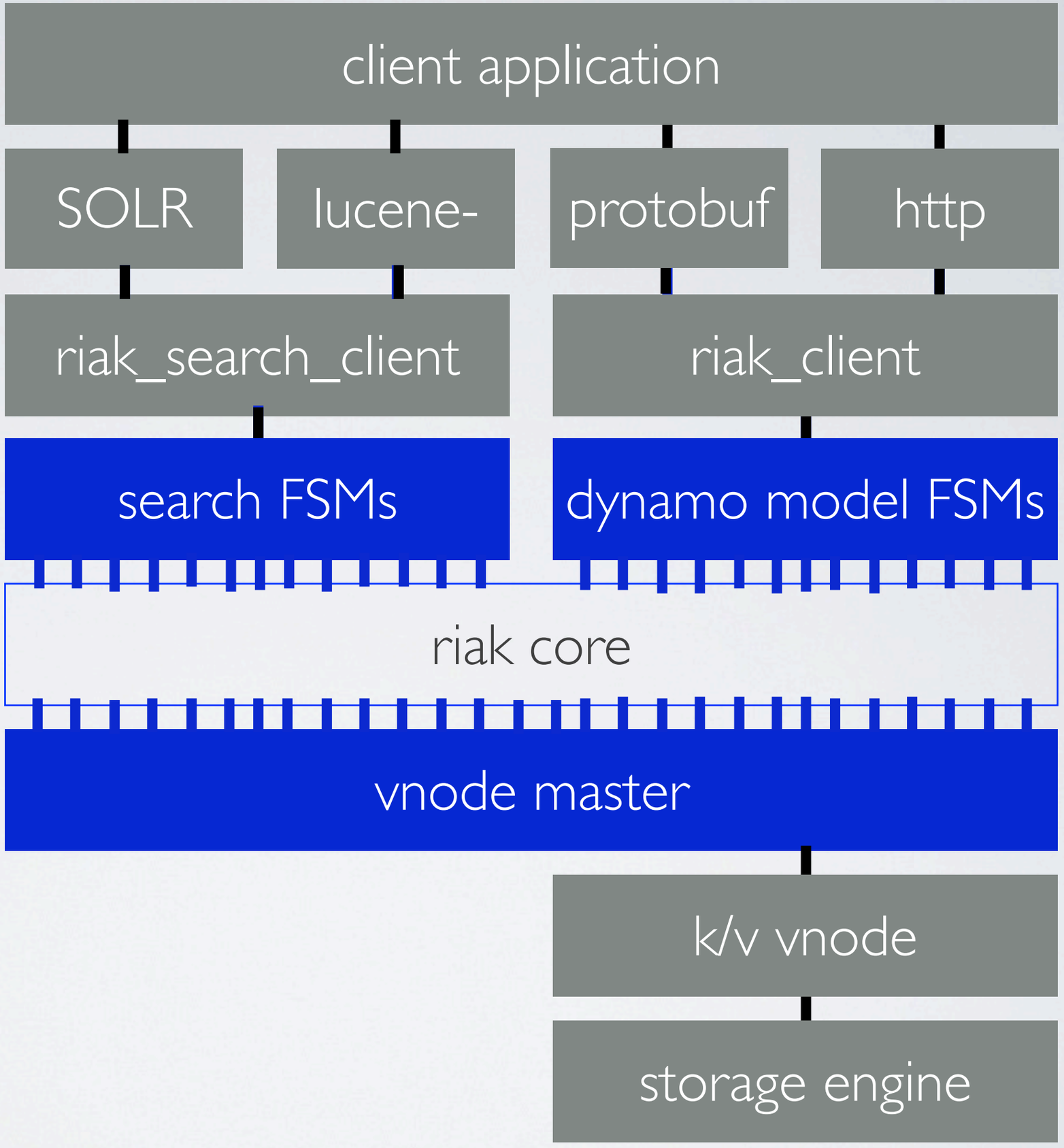
  Complexity handled by riak_core

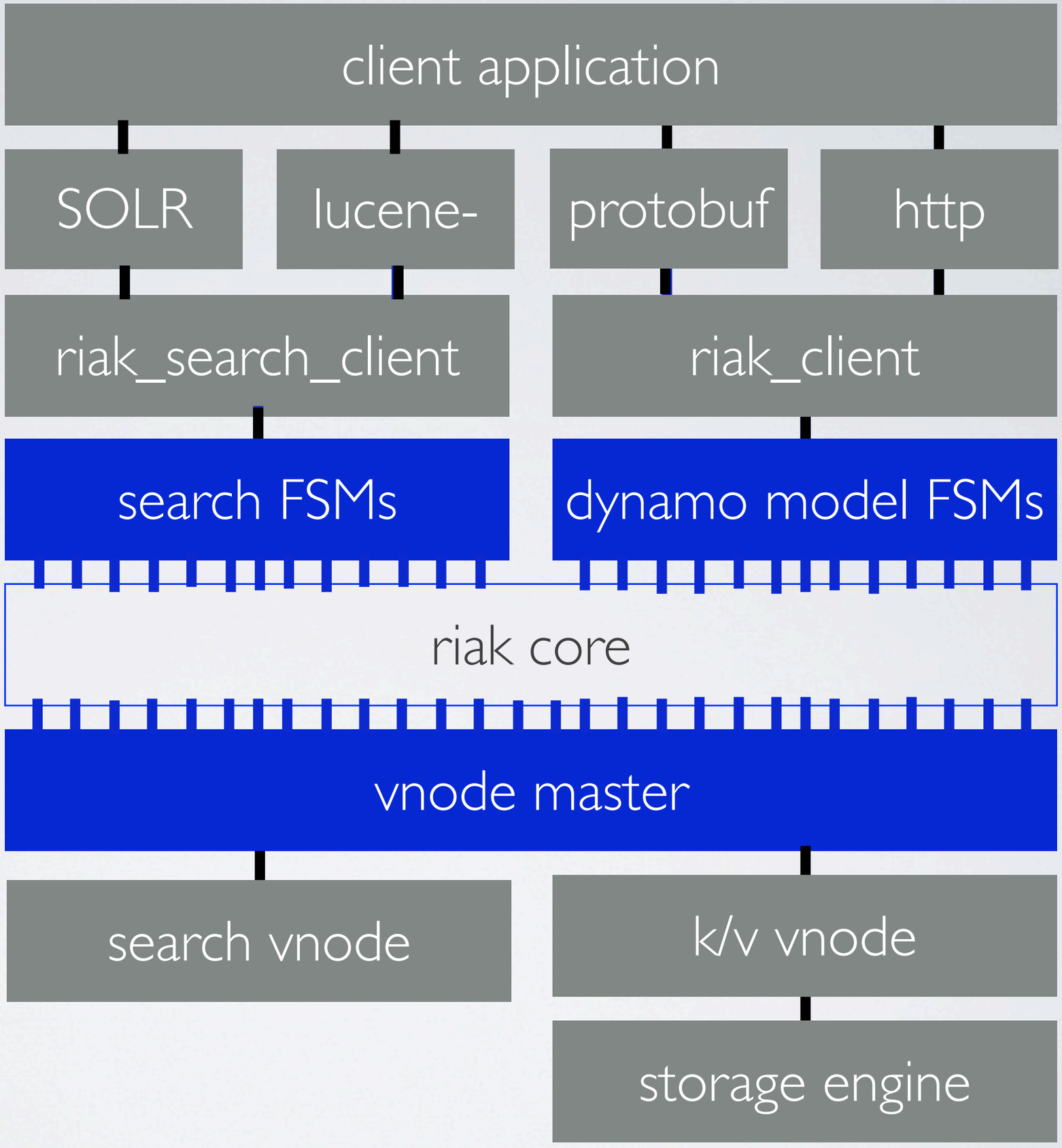

client application
protobuf
http
riak_client
dynamo model FSMs
riak core
vnode master
k/v vnode
storage engine

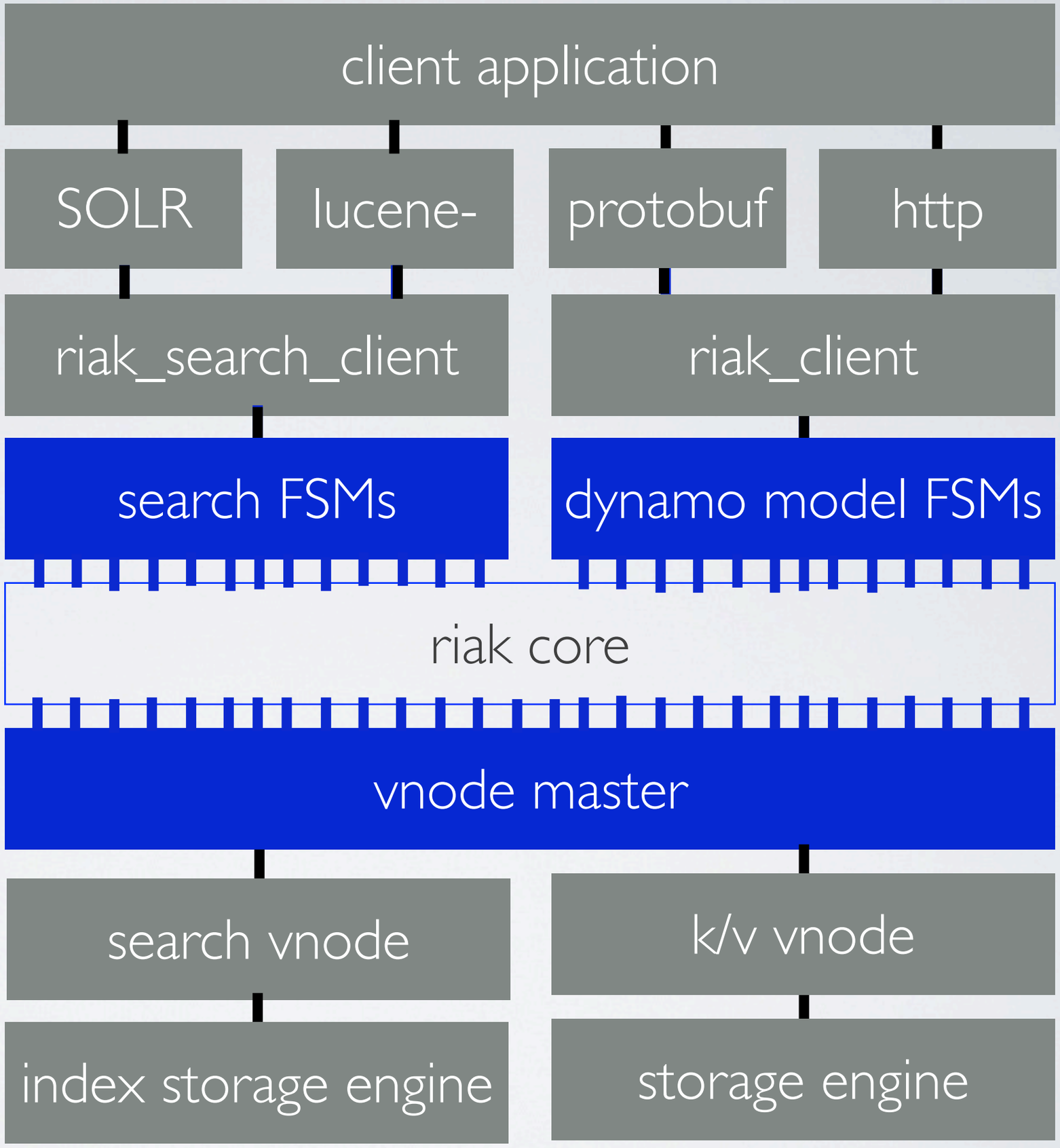

client application
protobuf
http
riak_client
dynamo model FSMs
riak core
vnode master
k/v vnode
storage engine


client application
SOLR
lucene-
protobuf
http
riak_client
dynamo model FSMs
riak core
vnode master
k/v vnode
storage engine

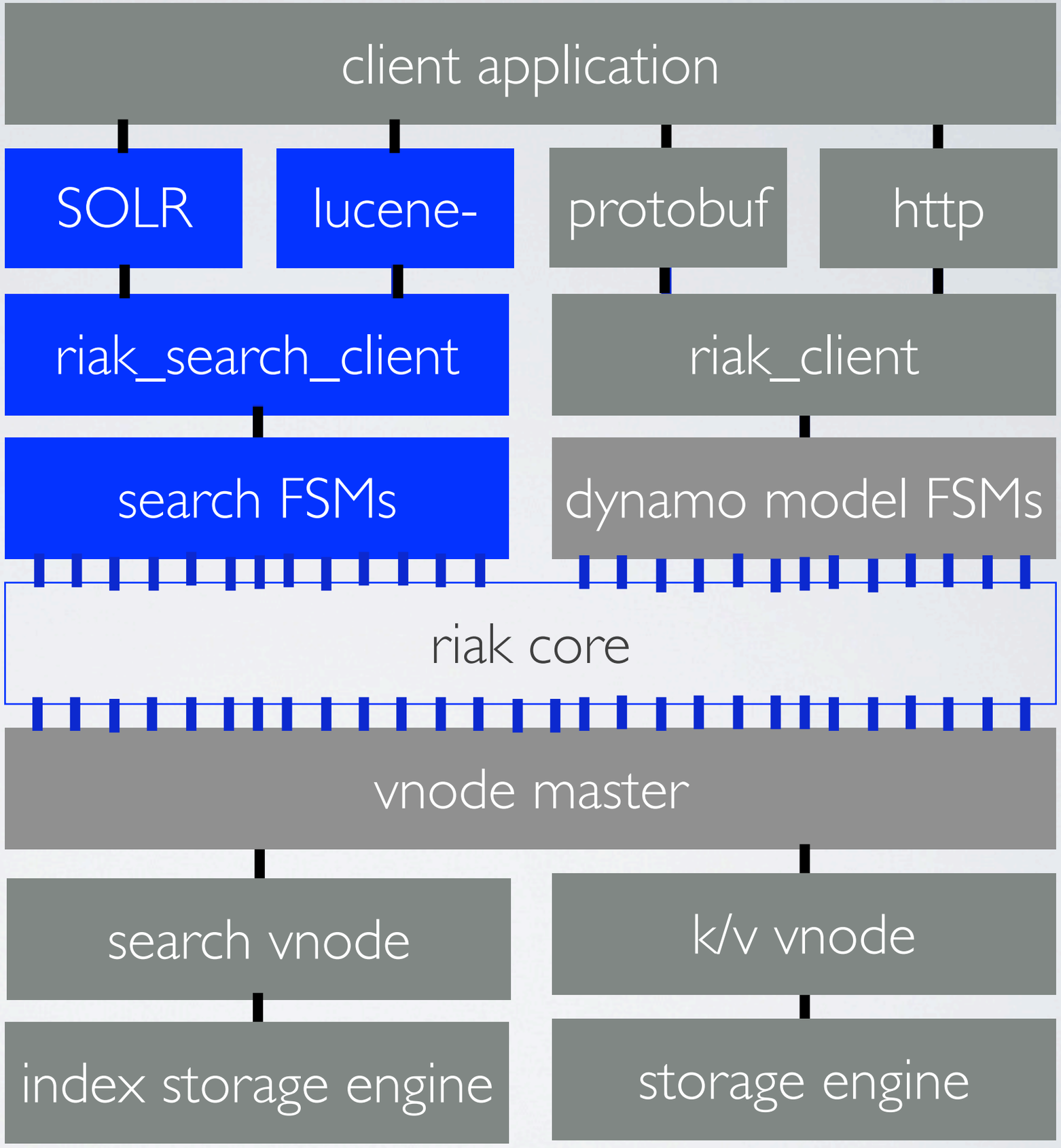

client application
SOLR
lucene-
protobuf
http
riak_search_client
riak_client
dynamo model FSMs
riak core
vnode master
k/v vnode
storage engine


client application
SOLR
lucene-
protobuf
http
riak_search_client
riak_client
search FSMs
dynamo model FSMs
riak core
vnode master
k/v vnode
storage engine


client application
SOLR
lucene-
protobuf
http
riak_search_client
riak_client
search FSMs
dynamo model FSMs
riak core
vnode master
search vnode
k/v vnode
storage engine


client application
SOLR
lucene-
protobuf
http
riak_search_client
riak_client
search FSMs
dynamo model FSMs
riak core
vnode master
search vnode
k/v vnode
index storage engine
storage engine

It's a search engine!



Made possible by Riak Core.

We're just getting started.

What will you build?

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

joe@basho.com on Email

@jtuple on Twitter

basho

riak

END