Unleashing the Core Value

Torben Hoffmann (@lehoff), architect @ basho.com


Through a ring of fire...

Grease Monkey... erh, Lion

I loose data!!!!

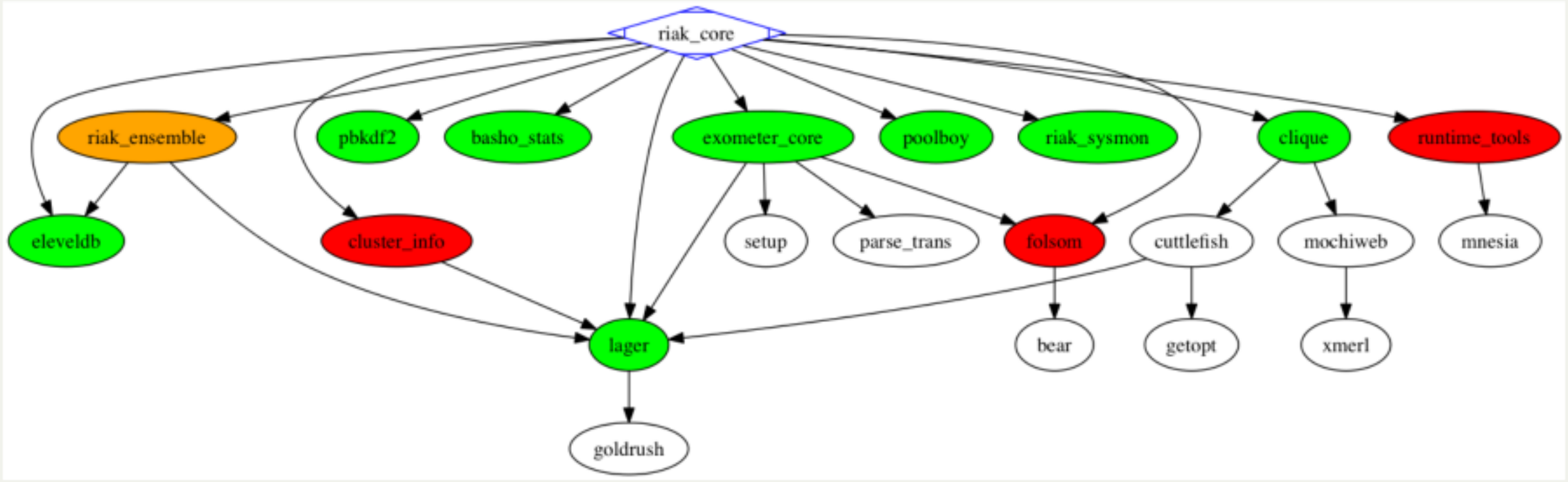
The Dream

Drop this into your rebar.config or erlang.mk

```
{deps,
      [{riak_core, "(2.0|2.1|2.2).*",
                   {git, "git://github.com/basho/riak_core.git",
                   {tag, "2.2.1"}}}]}.
```

and riak_core "just works":

The Problem

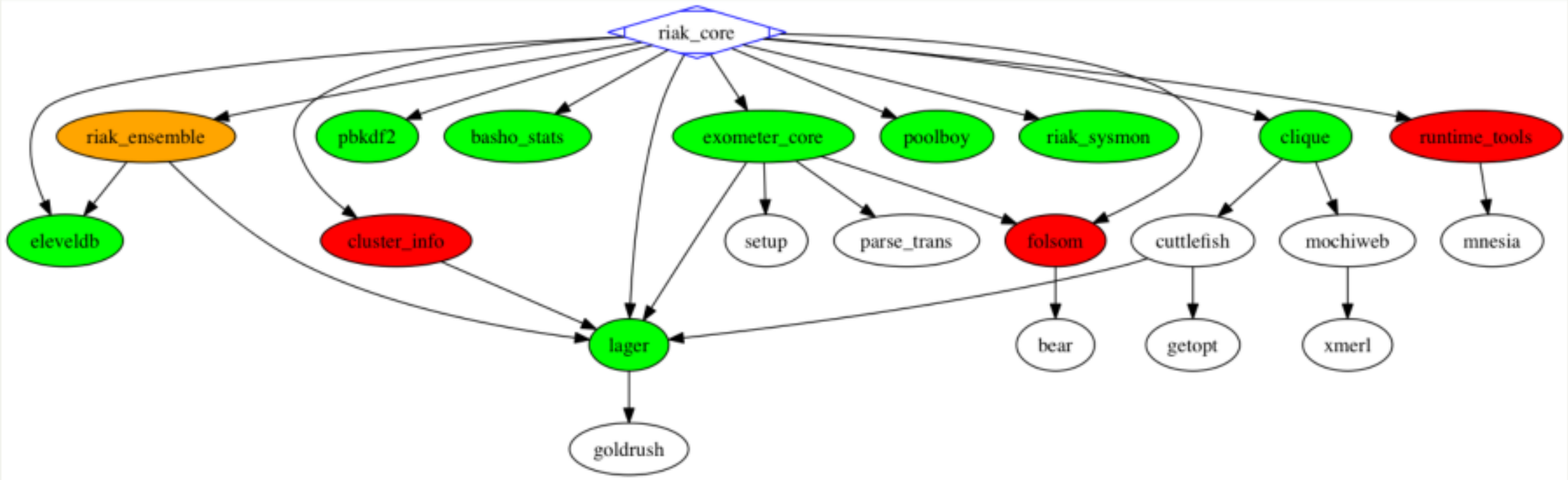


folsom is called, but not in rebar.config or .app.src :-(

Effect on releases

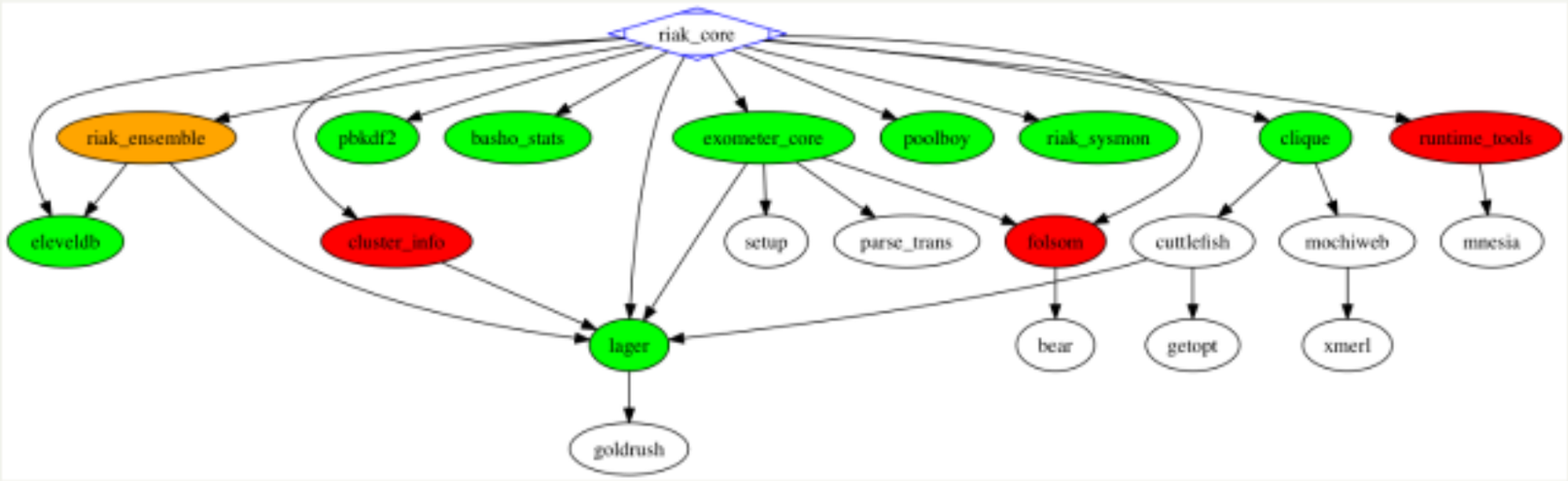
lib dir: non-OTP apps are without version numbers...

```
ogma:csi th$ ls lib
10-riak.schema              cuttlefish              meck
riak_pipe                   11-erlang_vm.schema     ebloom
merge_index                 riak_repl               12-riak_core.schema
edown                       mnesia-4.10             riak_repl_pb_api
13-riak_api.schema          eleveldb                mochiweb
riak_search                 14-riak_kv.schema       env.sh
neotoma                     riak_snmp               15-riak_sysmon.schema
eper                        node_package            riak_sysmon
16-bitcask.schema           erlang_js               observer-1.3.1.1
riaknostic                  17-bitcask_multi.schema  erlydtl
os_mon-2.2.13               runtime_tools-1.8.12
...
```


riak_core
riak_ensemble
pbkdf2
basho_stats
exometer_core
poolboy
riak_sysmon
clique
runtime_tools
eleveldb
cluster_info
setup
parse_trans
folsom
cuttlefish
mochiweb
mnesia
lager
bear
getopt
xmerl
goldrush

Simple fixes

runtime_tools and folsom just have to be added to rebar.config and .app.src.



Moderate fix

riak_ensemble is not started as an application.

riak_core_sup starts riak_ensemble_sup if strong_consistency is on.

This makes it harder to turn the whole thing into a proper OTP release.

Fix: make the starting of riak_ensemble_sup's children dynamic.

This will allow on-the-fly enabling of riak_ensemble and thereby also strong_consistency.

Starting children on-the fly

Move the children into a separate function

```
children() ->
     [?CHILD(riak_ensemble_router_sup, supervisor),
      ?CHILD(riak_ensemble_storage, worker),
      ?CHILD(riak_ensemble_peer_sup, supervisor),
      ?CHILD(riak_ensemble_manager, worker)].
```

Start each child intelligently

```
ensure_child({Id, _, _, _, _, _}=ChildSpec) ->
  case supervisor:start_child(?MODULE, ChildSpec) of
       {ok, _} ->
           ok;
       {ok, _, _} ->
           ok;
       {error, {already_started, _}} ->
           ok;
       {error, already_present} ->
           case supervisor:restart_child(?MODULE, Id) of
                {ok, _} ->
                    ok;
                {ok, _, _} ->
                    ok;
                {error, Okay} when Okay==running orelse Okay==restarting ->
                    ok;
                {error, Error} ->
                    exit({riak_ensemble_ensure_child_restart, Error})
           end;
      {error, Error} ->
           exit({riak_ensemble_ensure_child, Error})
end.
```

Start/stop when requested

```
engage() ->
    lists:foreach(fun ensure_child/1, children()).

disengage() ->
    lists:foreach(fun(Id) ->
                          supervisor:terminate_child(?MODULE, Id)
                  end,
                  children_ids()).
```

And in riak_ensemble_manager we find:

```
start_ensemble() ->
    riak_ensemble_sup:engage().

stop_ensemble() ->
    riak_ensemble_sup:disengage().
```

Dependency Nightmare

When you have a shared dependency rebar can run into severe problems...

E.g., edown updated in one place can cause other dependencies to barf.

This is **very painful**.

Getting a build with erlang.mk

Required two things...

COMPILE_FIRST for two files
update the clique version

But...

It doesn't ensure that we get all the versions we want.

But it is predictable!

Resolving version problems

First you see it...

Detecting dependency conflics by hand is painful.

Enter...

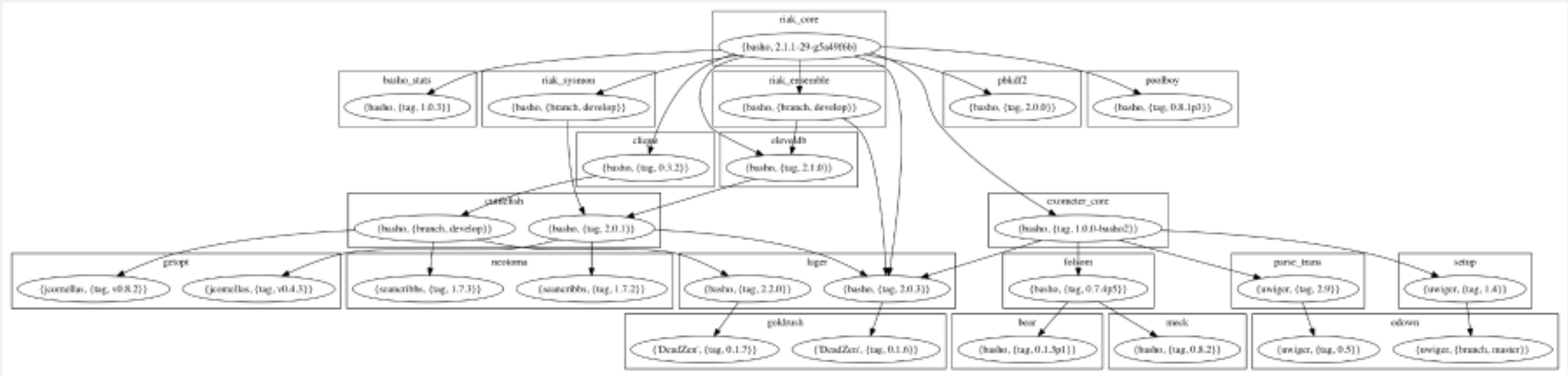
First you see it...

Detecting dependency conflics by hand is painful.

Enter...

Columbo

Columbo traverses the entire dependency tree and builds a graph of it.



Columbo source code

https://github.com/basho/columbo

Very WIP

Only works with rebar based projects for now.

Building a release with relx

Now that we have a proper OTP structure to our rebar.config and .app.src files we can use relx to build a release.

```
{lib_dirs, ["./deps"]}.

{release, {riak_core, "2.1.2.1"},
          [riak_core]}.
```

How about that lib dir?

```
asn1-3.0.4
basho_stats-1.0.3
clique-0.3.3-1-gb971cef
compiler-5.0.4
crypto-3.5
cuttlefish-2.0.4
eleveldb-2.1.2-7-gc72c498
exometer_core-1.0.0-basho2-3-g3e15168
folsom-0.7.4p5
goldrush-0.1.7
inets-5.10.6
kernel-3.2
lager-2.1.1
mochiweb-2.9.0
os_mon-2.3.1
pbkdf2-2.0.0
poolboy-0.8.1p3
public_key-0.23
riak_core-2.1.1-31-g28c4b9a
riak_ensemble-2.1.0-27-g7a43867
riak_sysmon-2.0.0
runtime_tools-1.8.16
sasl-2.4.1
ssl-6.0
stdlib-2.4
syntax_tools-1.6.18
webmachine-1.10.8
xmerl-1.3.7
```

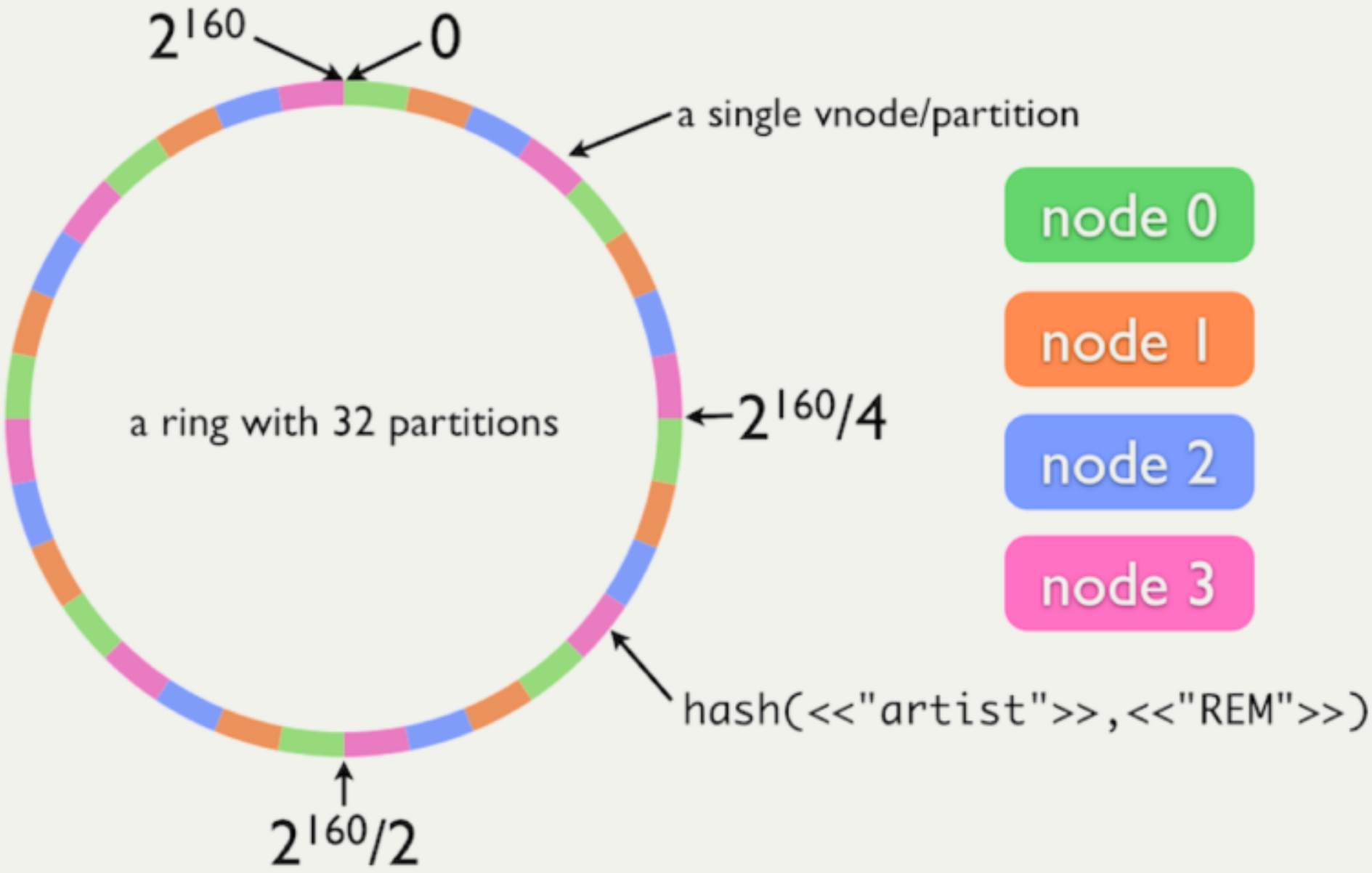
Next Steps

1. Get overlays in place
2. Proper configuration files
3. Figure out if cluster_info should be a utility

Using Riak Core

Do **not** stare into the ring!

The Ring


2^{160}
0
a single vnode/partition
node 0
node 1
node 2
node 3
a ring with 32 partitions
$2^{160}/4$
hash(<<"artist">>,<<"REM">>)
$2^{160}/2$

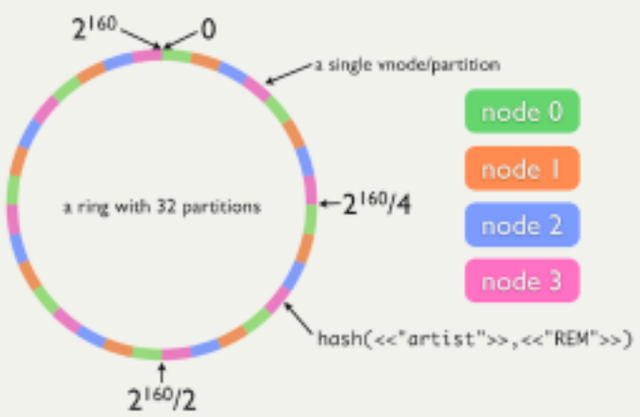
Nodes and Vnodes

Node: physical machine.

Vnode: Erlang Process responsible for a partition.

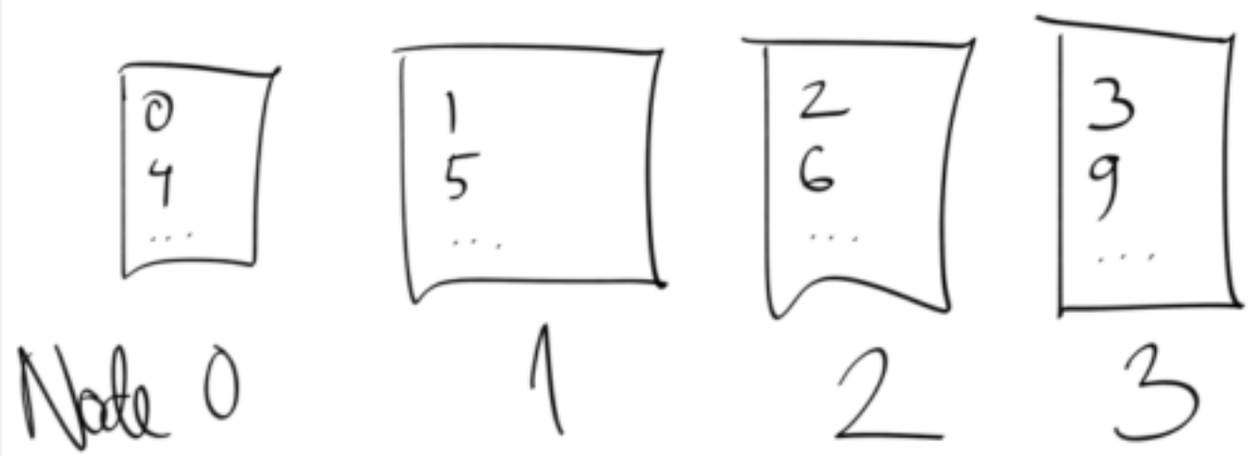
Not 1:1 ratio between node:vnode!

One node handles several vnodes.



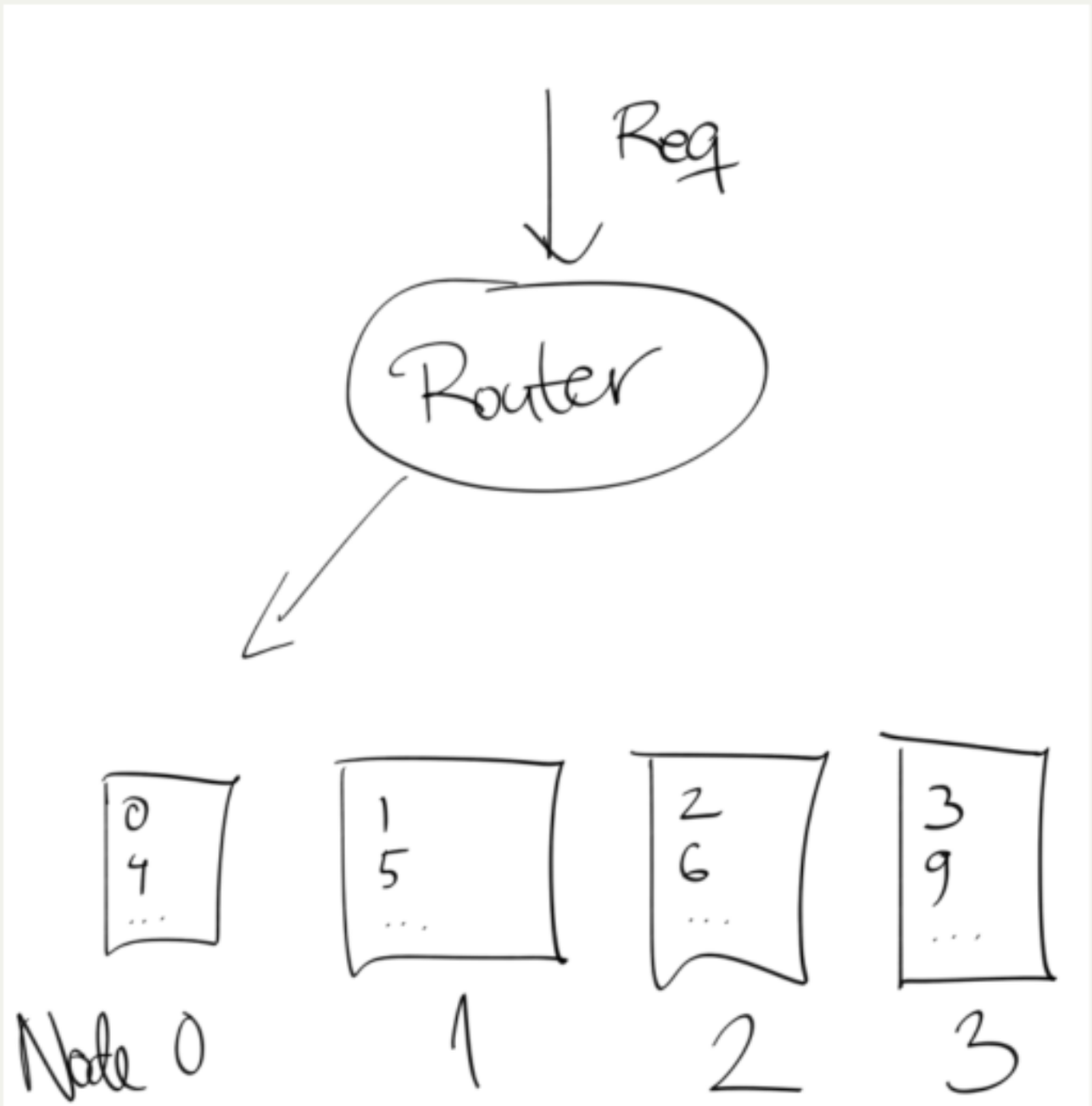
Node 0 handles partitions 0, 4, 8, 12, 16, . . ., 28

Starting Point

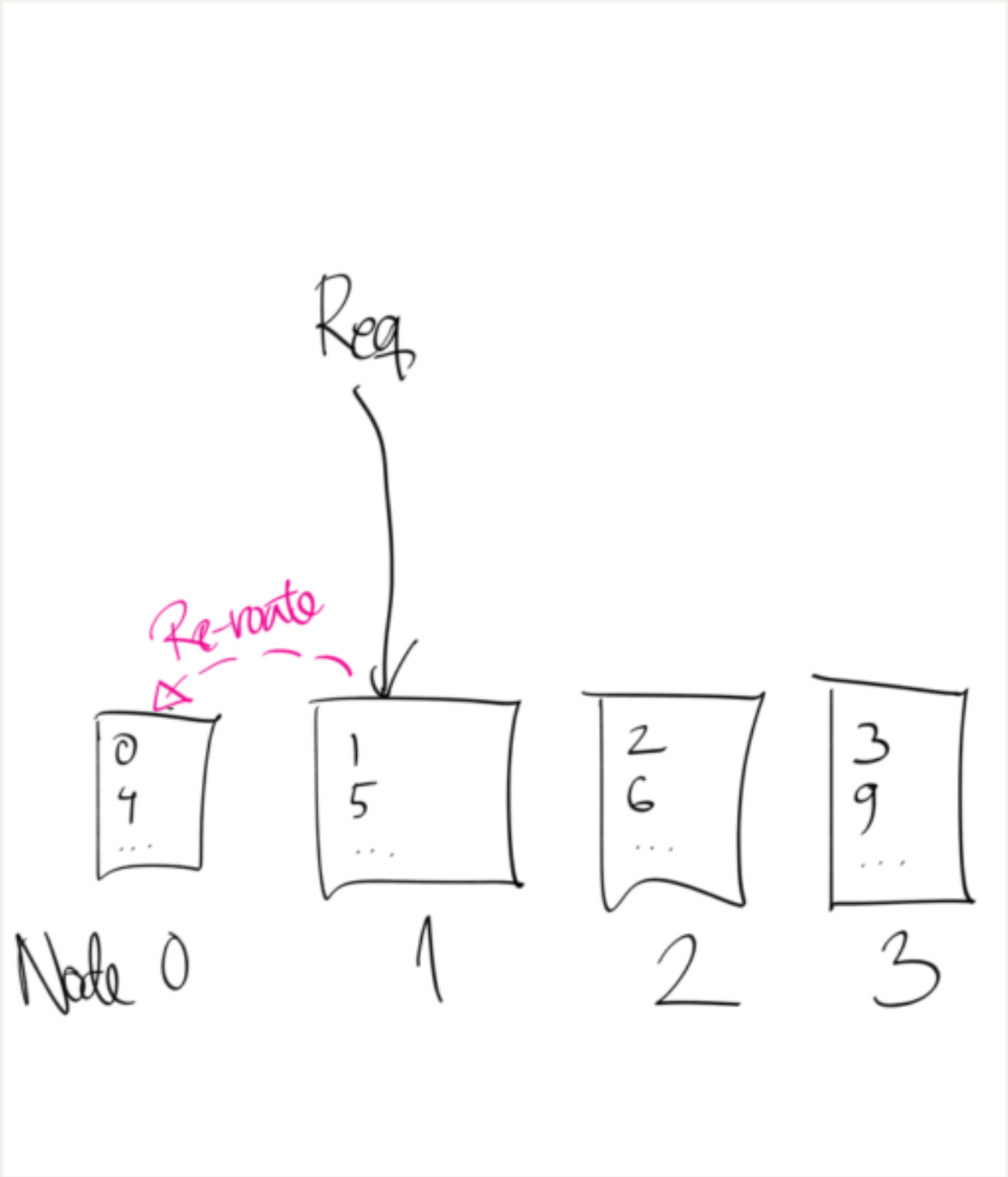


Naïve Routing

SPOF

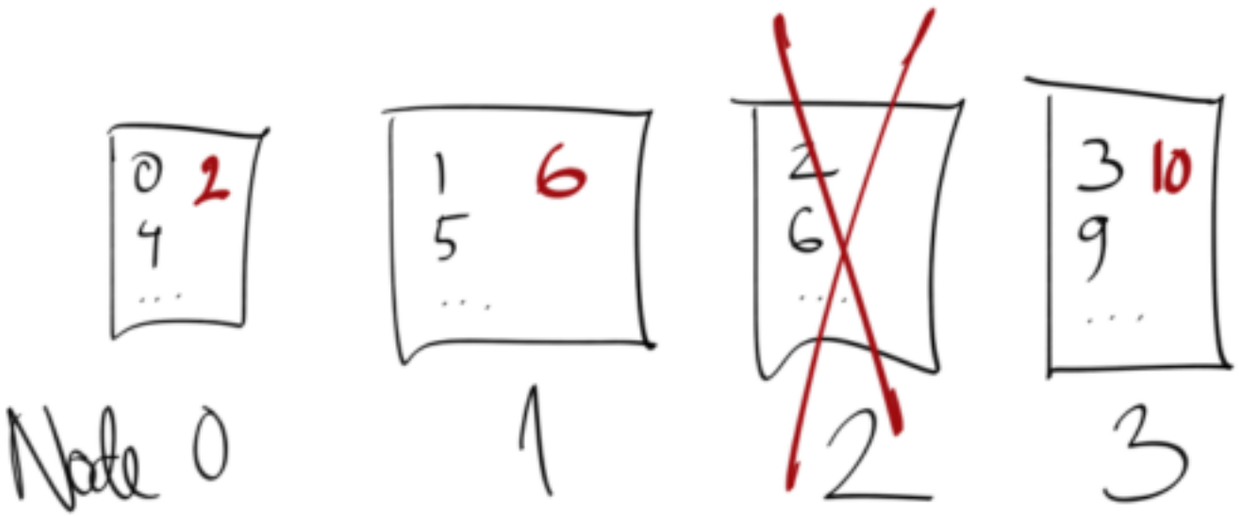


Actual Routing

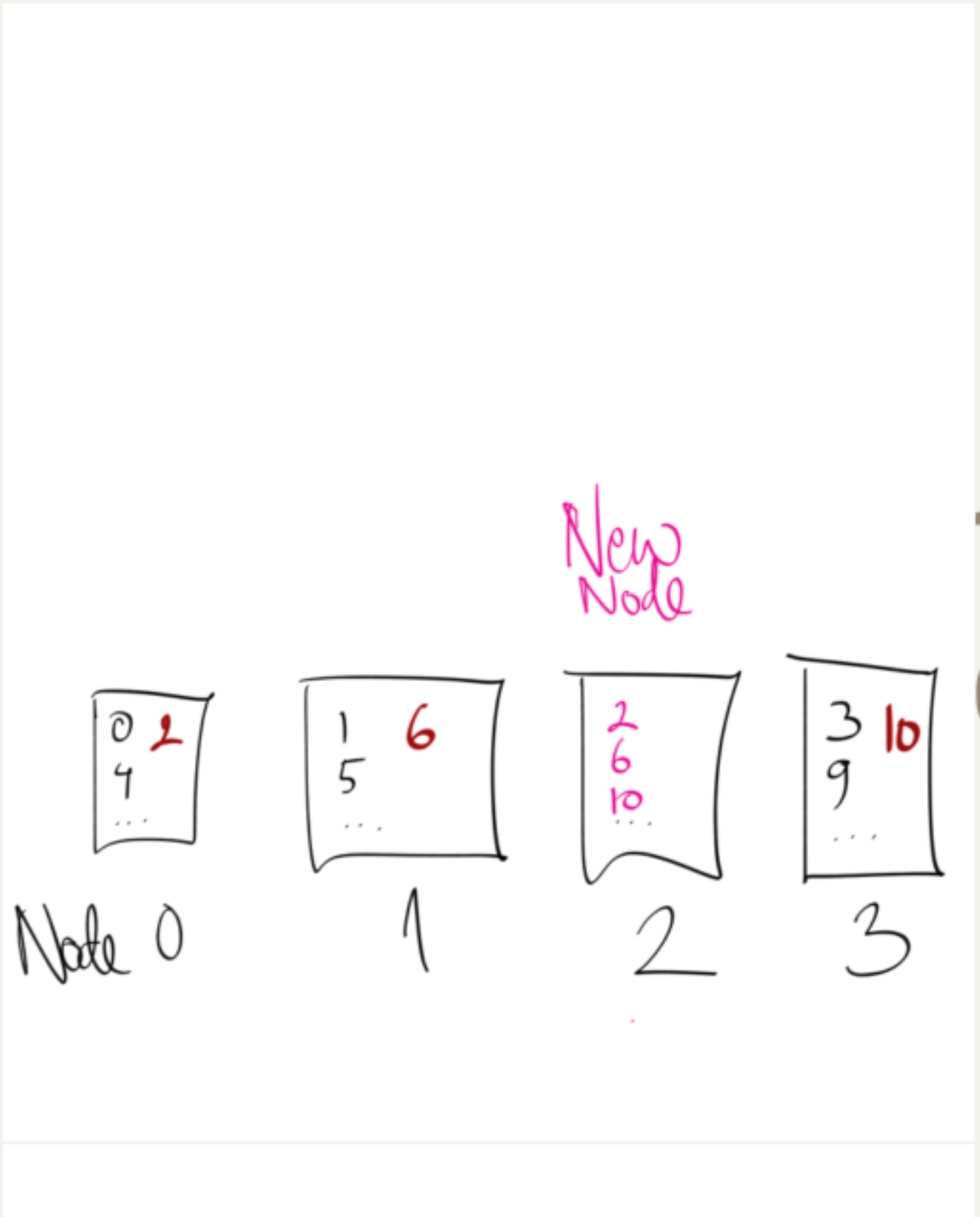


Node Death

New vnodes on the other nodes!

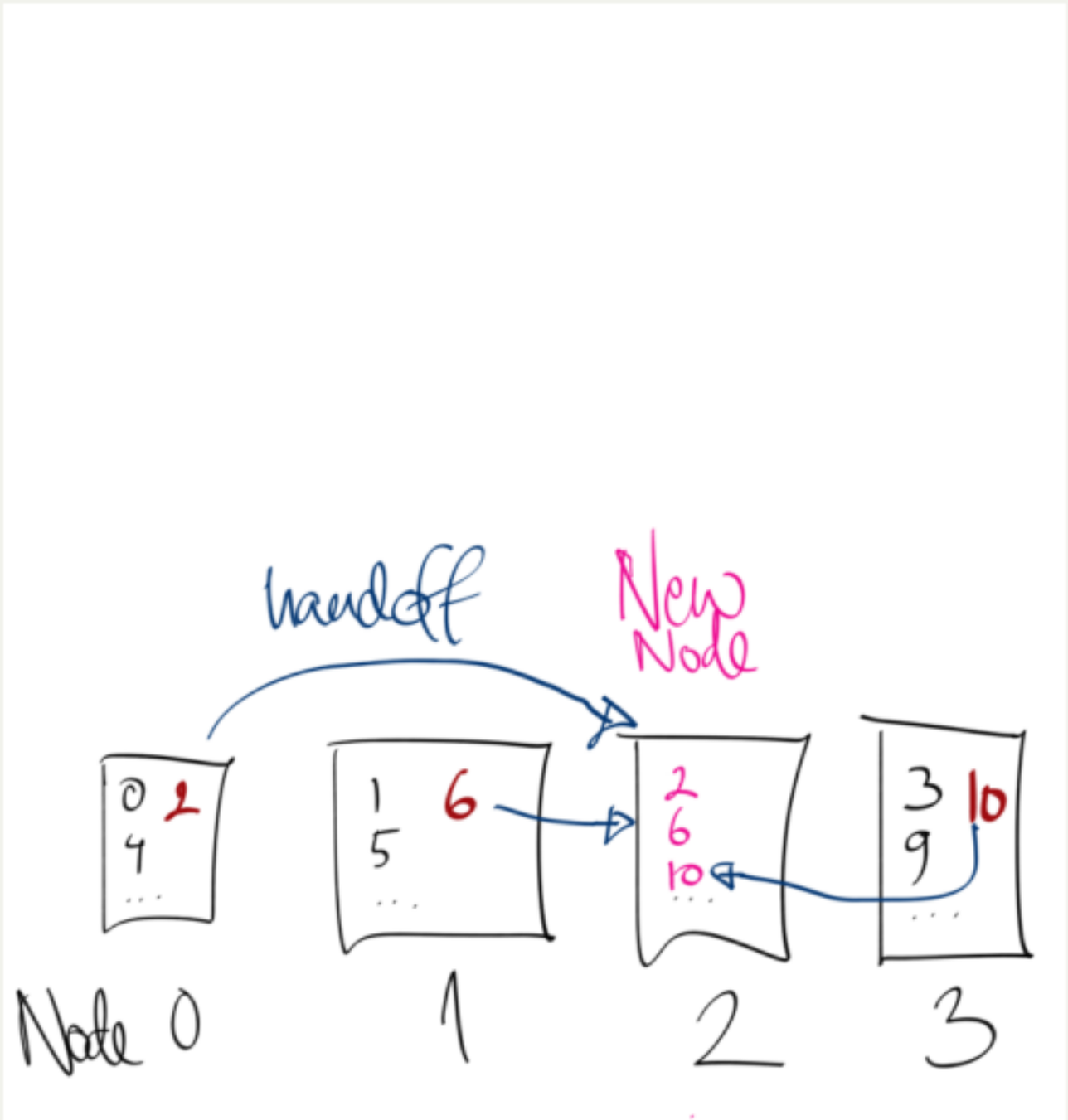


Return of the King



There can be only one!!

Handoff



handle_handoff

```
handle_handoff_command(?FOLD_REQ{foldfun=Fun, acc0=Acc0},
                       _Sender, State) ->
    Acc = dict:fold(Fun, Acc0, State#state.stats),
    {reply, Acc, State}.
```

You must fold all the data at your vnode into one blob.

The blob is returned to its rightful owner.

(from https://github.com/rzezeski/try-try-try/tree/master/2011/riak-core-the-vnode)

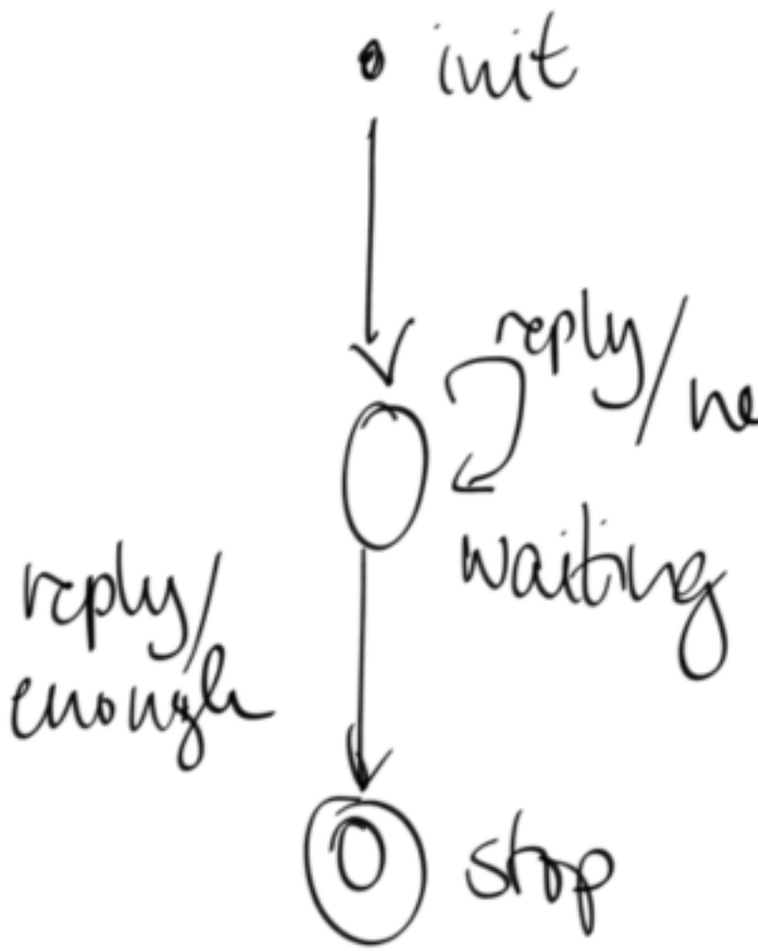
Handling Errors

Say you want to write some data to N nodes... if W of them succeeds you're happy ;-)

Enter coordination.

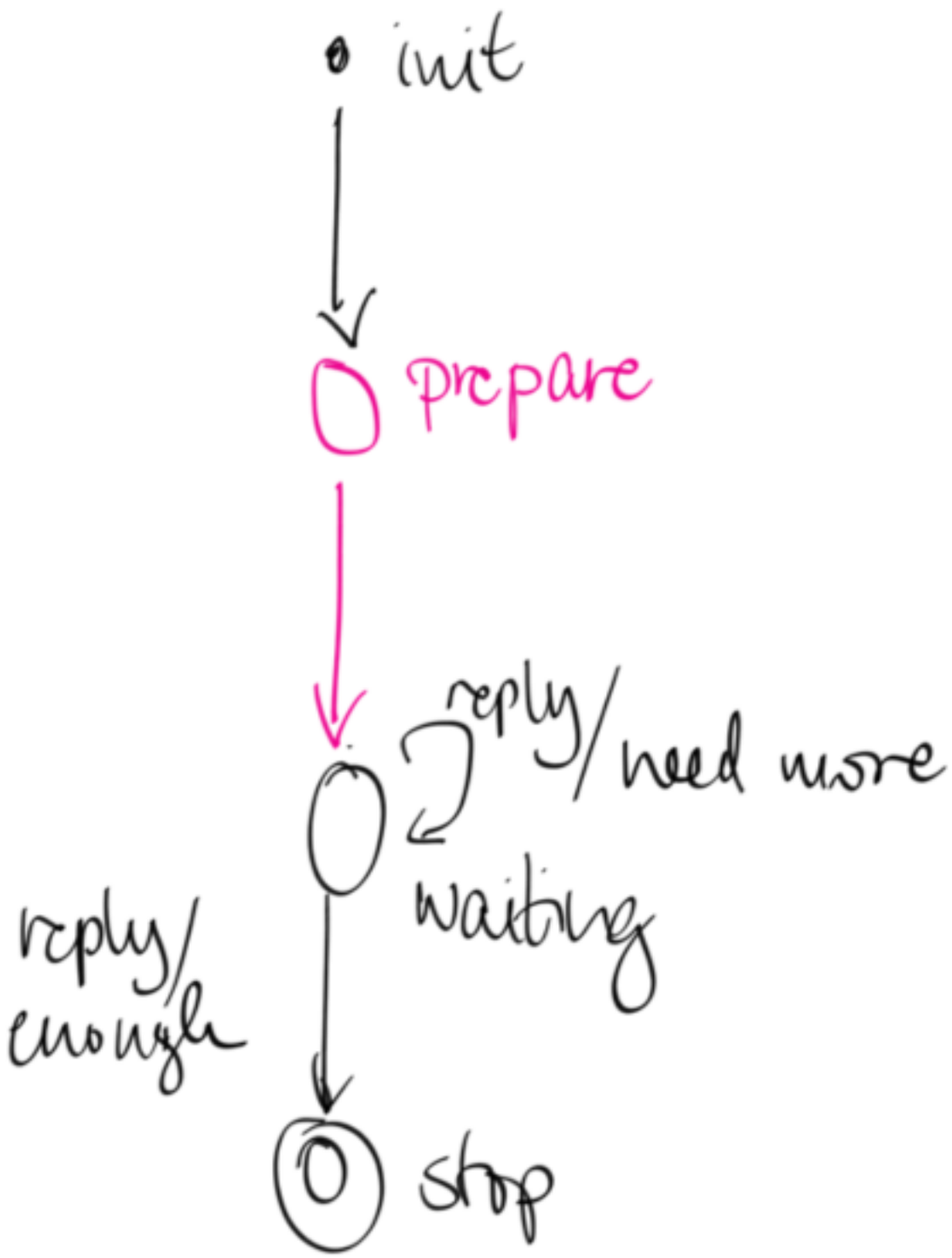
Create a gen_fsm that sends to N nodes and waits for the replies to come back.

Simple FSM



Send out all the requests to the other vnodes from the init state.

Complex Init



If the set-up for sending requests to the other vnodes takes a long time you may want to have a separate state for that.

Do not in a million years use a zero timeout to transition between states!

gen_fsm self-transitioning

```
init(...) ->
    ...
    gen_fsm:send_event(self(), kickoff),
    ...
    {ok, prepare, StateData}.
```

And the `prepare` state:

```
prepare(kickoff, StateData) ->
    ...
    {next_state, waiting, StateData}.
```

Is Riak Core a Good Fit(tm)?

Assumptions:

It expects you to have a "key" that links to a blob of data.

The key (or rather its chash) determines its primary vnode and adjacent replicas.

The data itself is opaque and has application context.

Presumably you want behavior semantics that are different than riak_kv with your data.

Resources

Little Riak Core Book:

https://marianoguerra.github.io/little-riak-core-book/

Ryan Zezeski blog posts:

https://github.com/rzezeski/try-try-try/tree/master/2011/riak-core-the-vnode

Mark Allens udon demo projec:

https://github.com/mrallen1/udon