Riak Pipe:
Distributed Processing System

Bryan Fink
Principal Software Engineer,
Basho Technologies

EUC2013
14.Jun.2013
Stockholm, Sweden

What is Riak Pipe?

- Abstraction of Riak Core
- Stages consuming inputs and producing outputs

Conceptual Example

Fetch

"comment/12345"

"{"author":"bryan",
"time":20120905040000
"text":"What a great slide!"}"

Render

*" *
bryan said:
* What a great slide! "*

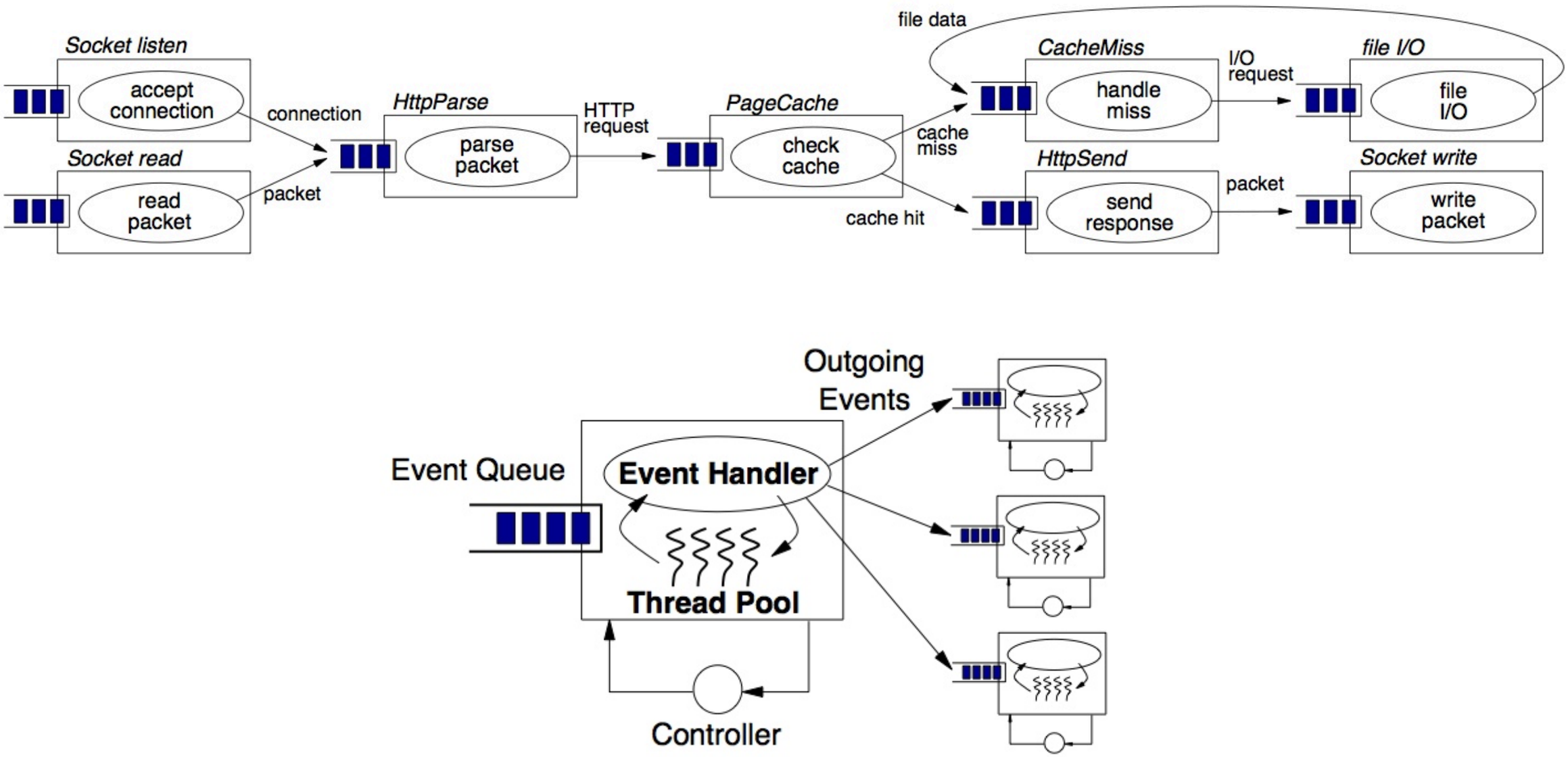
Why is Riak Pipe?

- Needed a new system to run Riak's MapReduce
- Needed debugging & monitoring
- Needed backpressure

Conceptual Problem

- Uncontrolled parallelism
- Uncontrolled memory consumption
- Uncontrolled worst-case latency

SEDA

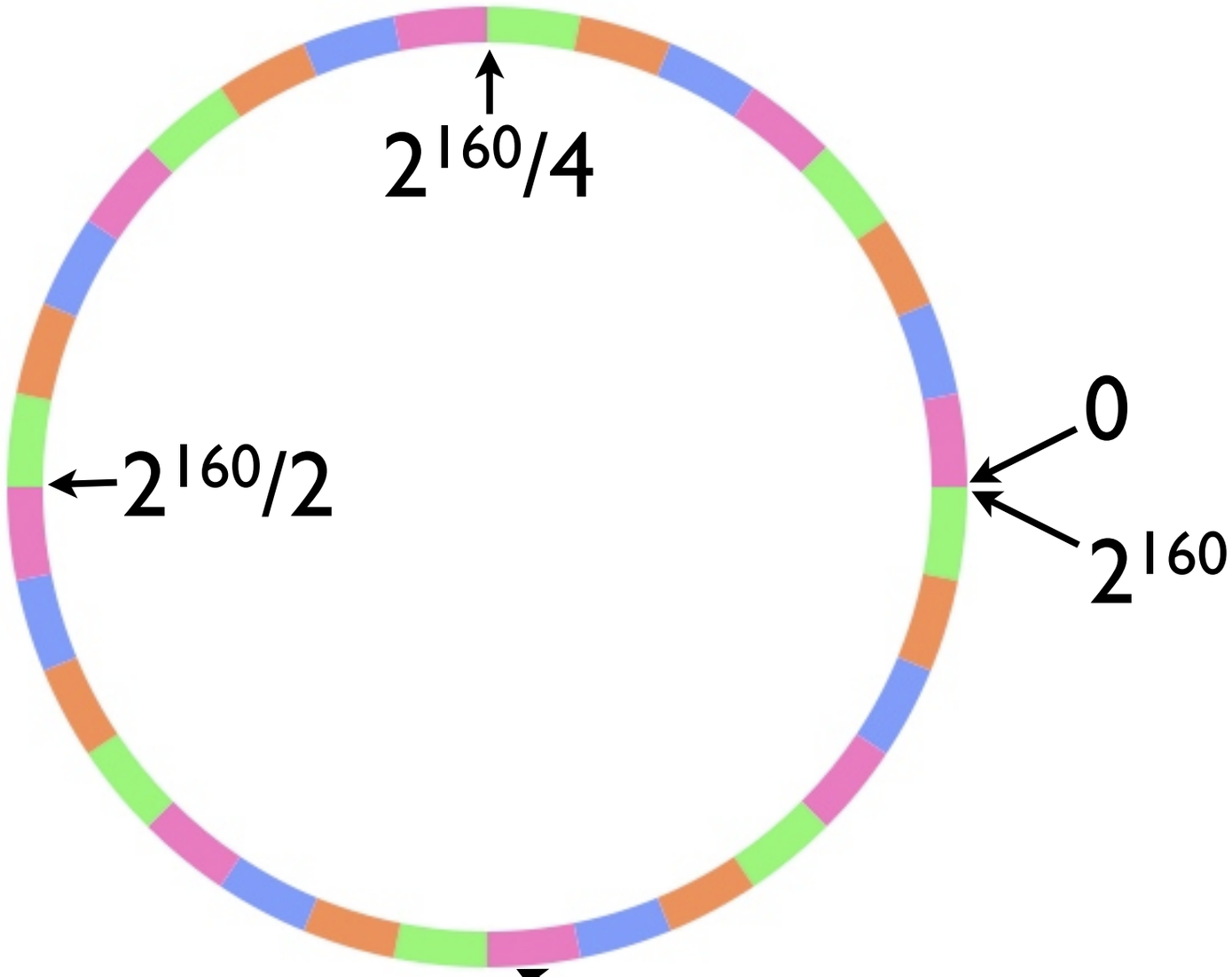


Figures 5 & 6, M.Welsh, D. Culler, E. Brewer. SEDA:An Architecture for Well-Conditioned, Scalable Internet Services. SOSP 2001, October 21-24, 2001, Chateau Lake Louise, Canada.

SEDA Advantages

- Queues can be size-capped to limit backlog
- Size of worker pool can be managed to limit resource usage

Consistent Hashing



node 0

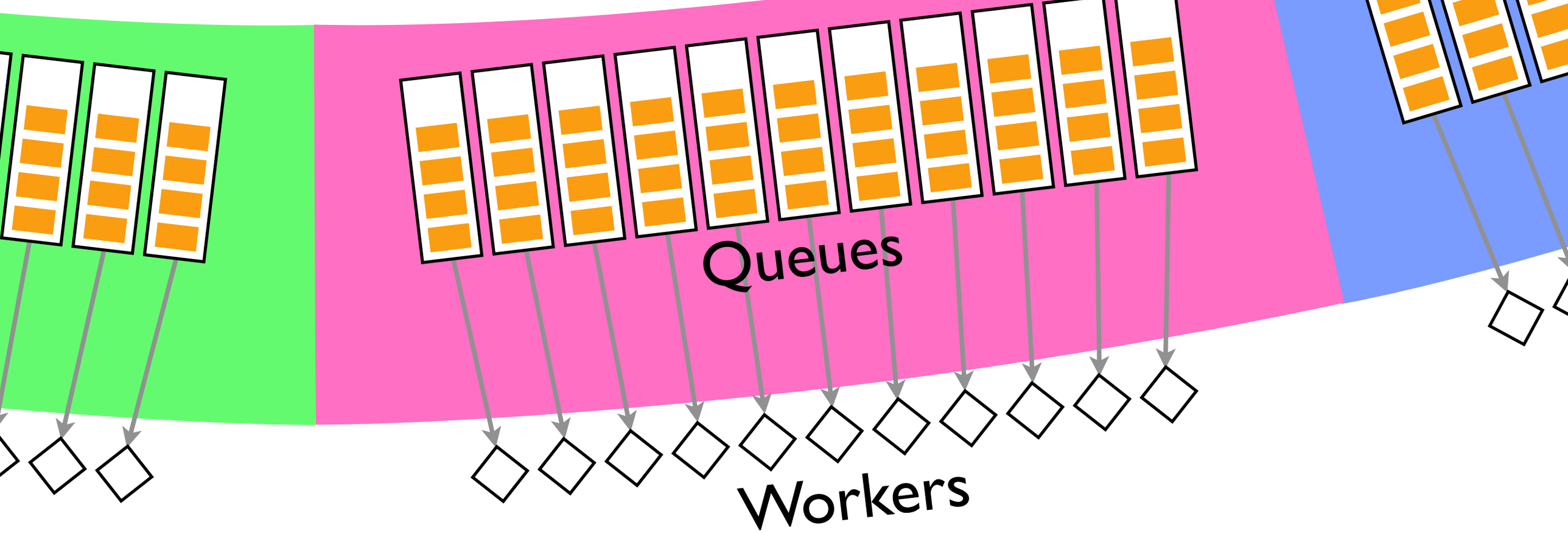
node 1

node 2

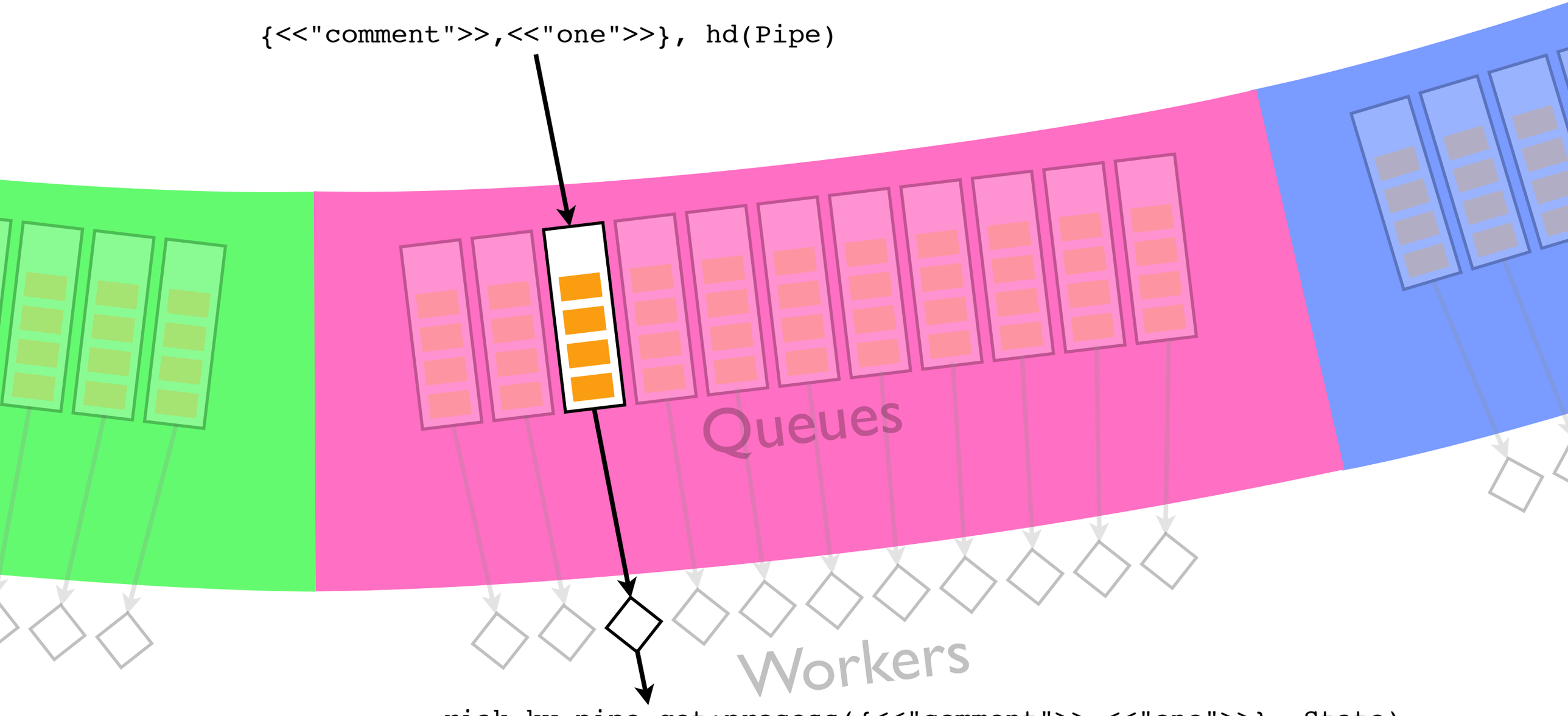
node 3

```
riak_kv_pipe_get:bkey_chash({<<"comment">>,<<"one">>})
```

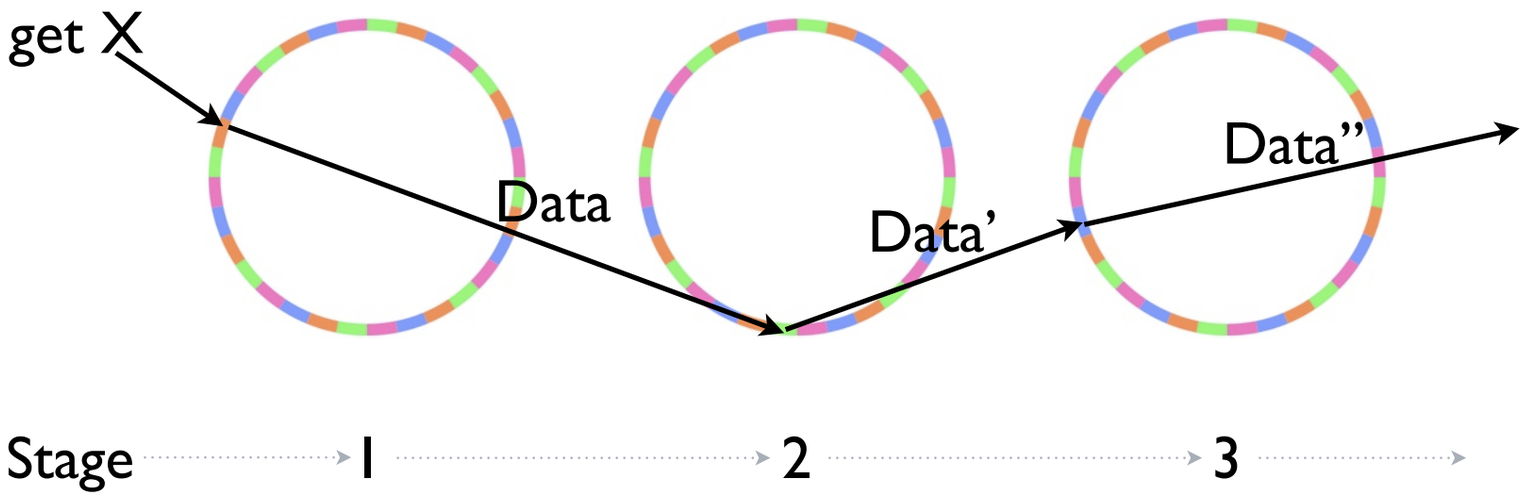
Riak Pipe “Vnode”



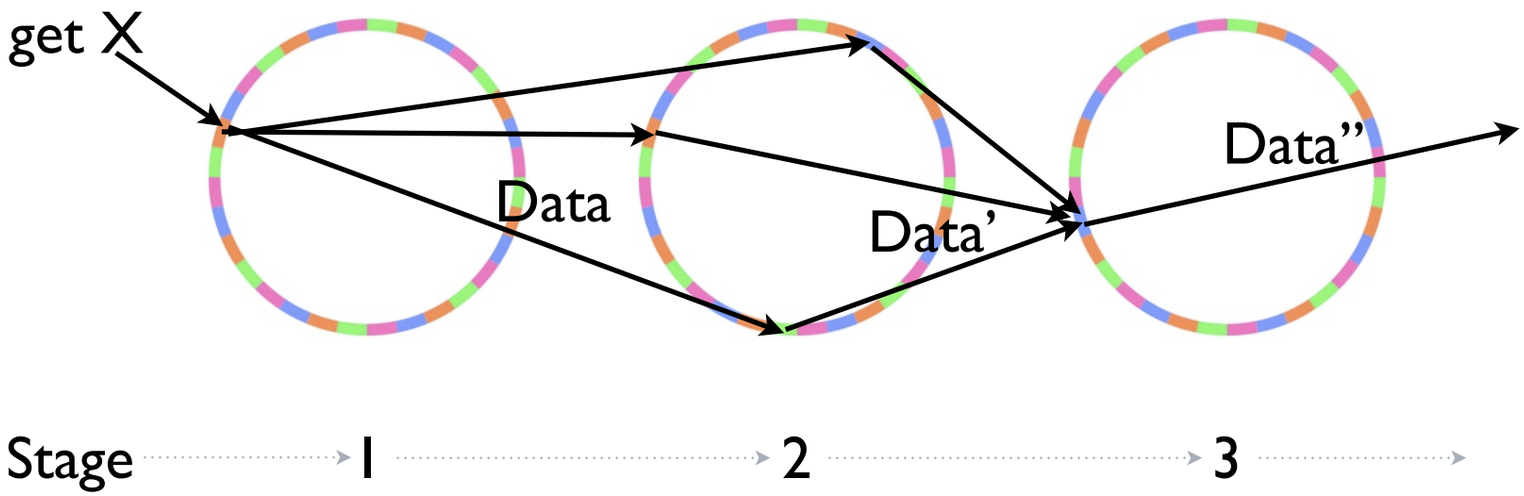
Riak Pipe “Vnode”



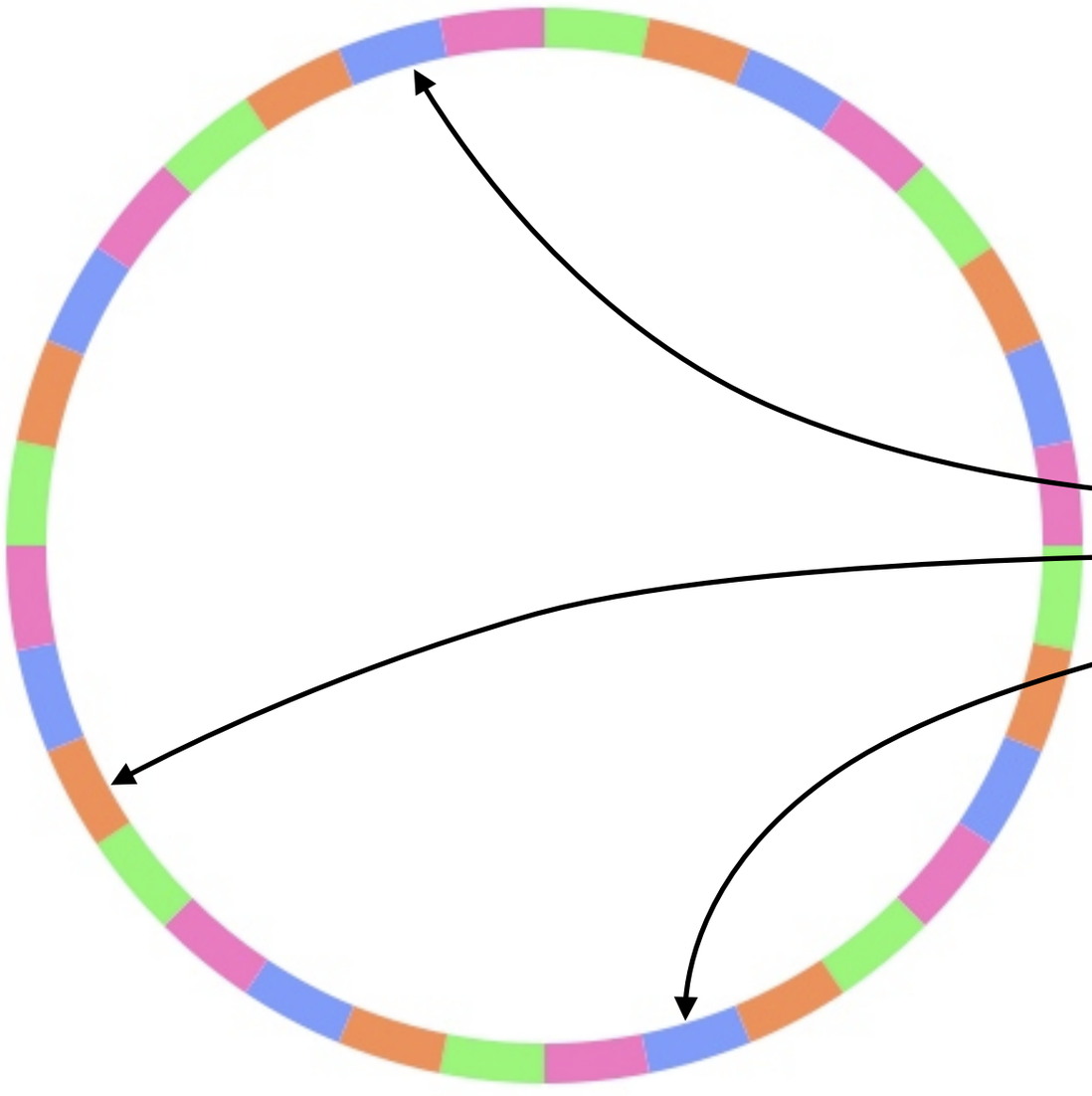
Core Abstraction



Core Abstraction



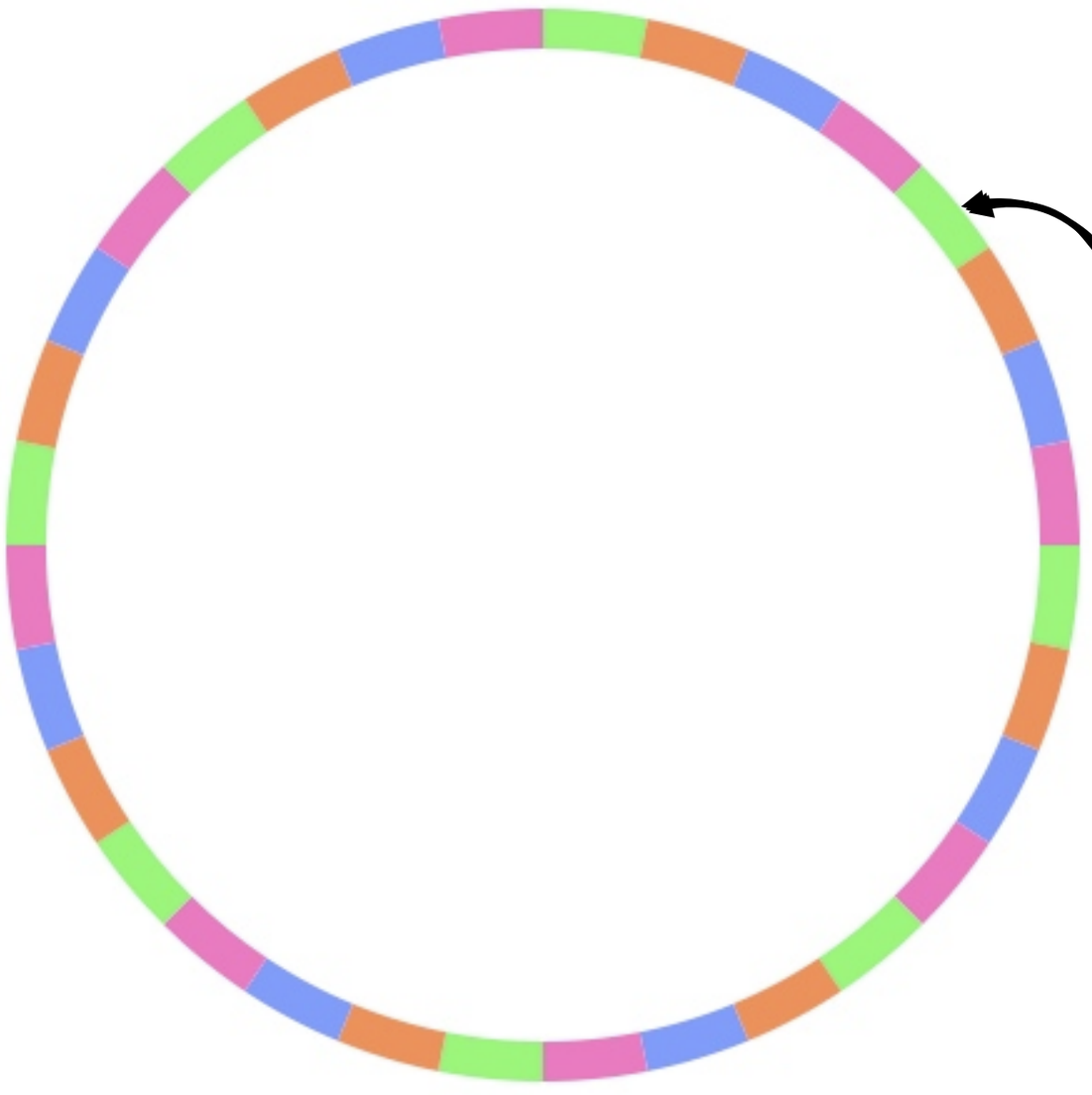
Spread Work



```
hash(X) ->
    crypto:sha(X).
```

```
{“comment”, “one”}
{“comment”, “two”}
{“comment”, “three”}
```

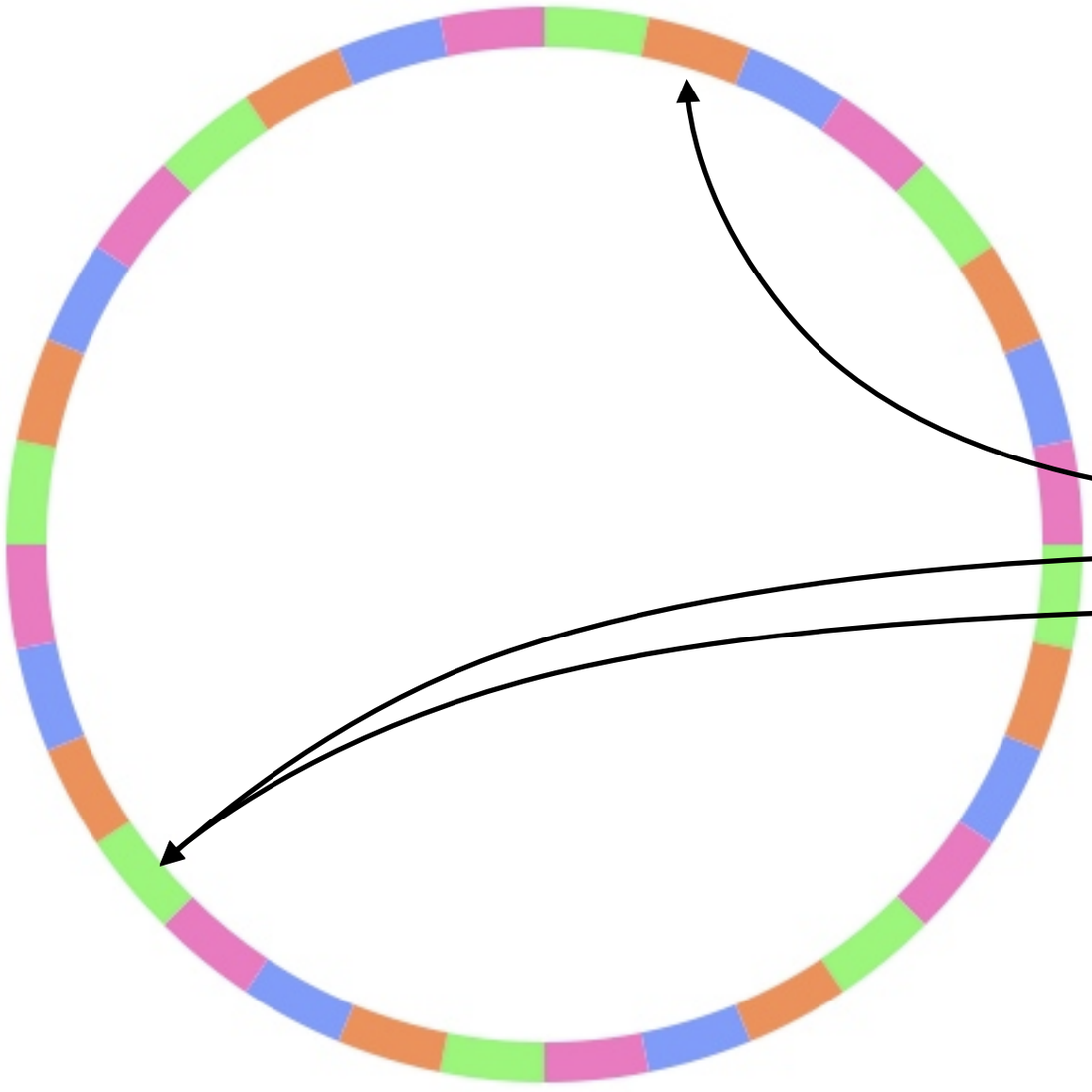
Funnel Results



```
hash(X) ->
    crypto:sha(element(1, X)).
```

```
{“comment”, “one”}
{“comment”, “two”}
{“comment”, “three”}
```

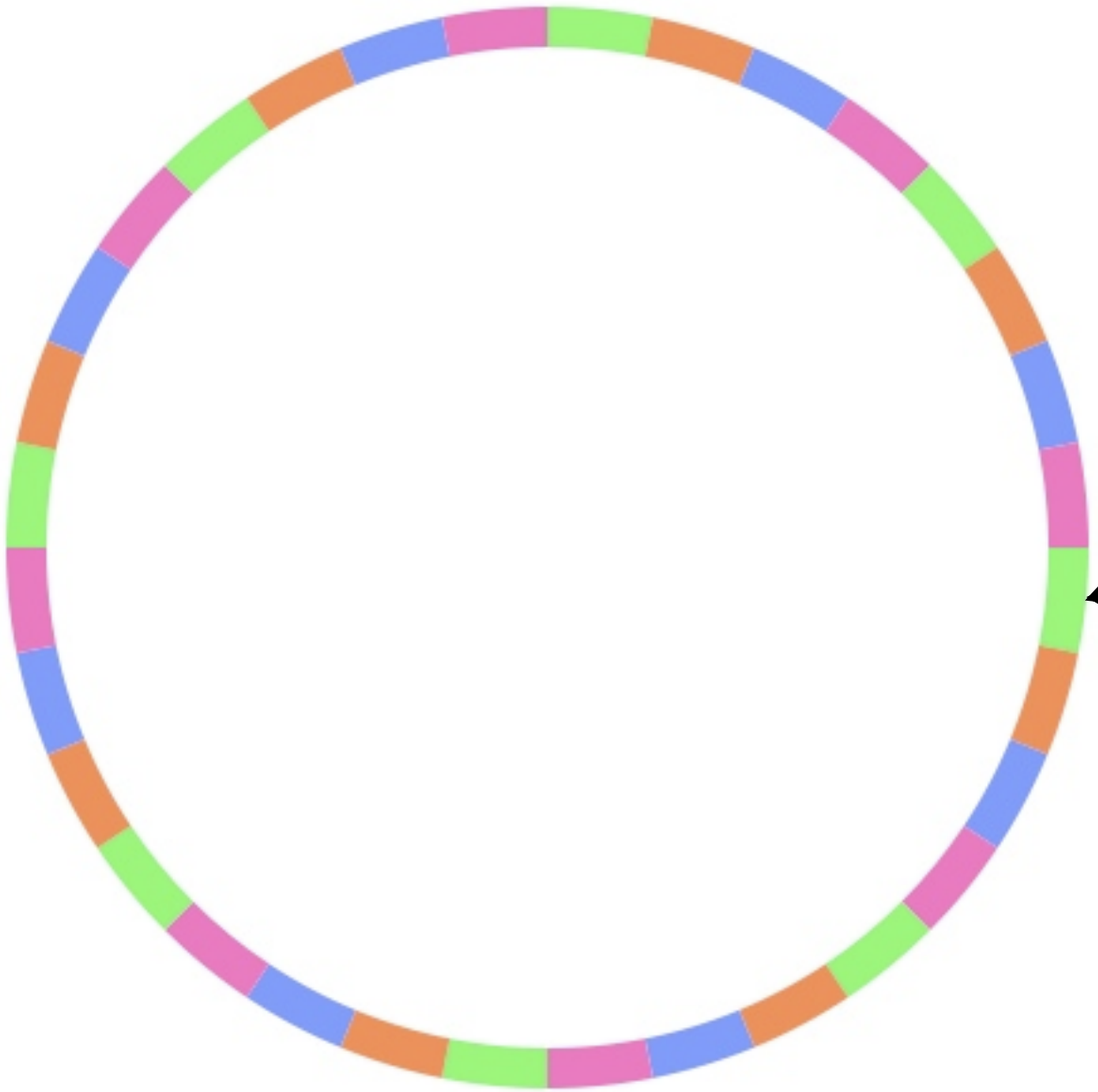
Bin Results



```
hash(X) ->
    crypto:sha(hd(element(2, X))).
```

```
{“comment”, “one”}
{“comment”, “two”}
{“comment”, “three”}
```

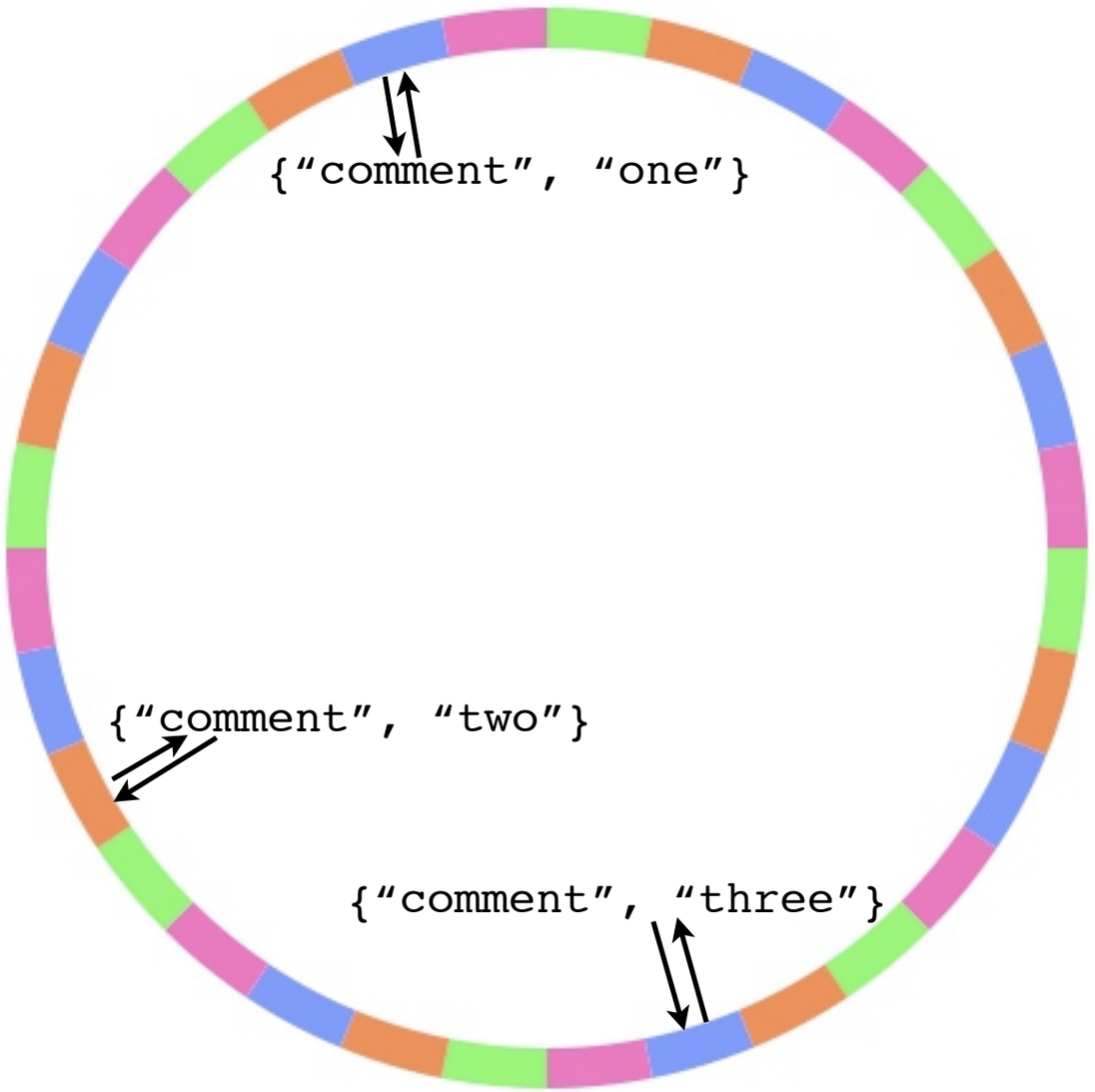
Constant



```
hash(_) ->
    <<123,321,234,0,0,...>>.
```

```
{“comment”, “one”}
{“comment”, “two”}
{“comment”, “three”}
```

Maintain Locality



‘follow’.

Pipe Example

```
Spec = [#fitting_spec{name=fetch,
                      module=riak_kv_pipe_get,
                      chashfun={riak_kv_pipe_get, bkey_chash},
                      nval={riak_kv_pipe_get, bkey_nval}},
        #fitting_spec{name=render,
                      module=riak_pipe_w_xform,
                      arg={my_app, render_comment},
                      chashfun=follow}],
Options = [],
{ok, Pipe} = riak_pipe:exec(Spec, Options),

ok = riak_pipe:send_input({<<“comment”>>, <<“one”>>}, Pipe),
ok = riak_pipe:send_input({<<“comment”>>, <<“two”>>}, Pipe),
ok = riak_pipe:send_input({<<“comment”>>, <<“three”>>}, Pipe),
riak_pipe:eoi(Pipe),

{eoi, RenderedComments, _Log} = riak_pipe:collect_outputs(Pipe).
```

Fitting Example

```
-module(riak_pipe_w_xform).
-behaviour(riak_pipe_vnode_worker).
-export([init/2,process/3,done/1]).

-record(state, {p :: riak_pipe_vnode:partition(),
                fd :: riak_pipe_fitting:details()}).

init(Partition, FittingDetails) ->
    {ok, #state{p=Partition, fd=FittingDetails}}.

process(Input, _Last, #state{p=P, fd=FD}=State) ->
    {Mod,Fun} = FittingDetails#fitting_details.arg,
    Results = Mod:Fun(Input),
    [ riak_pipe_vnode_worker:send_output(R,P,FD) || R <- Results ],
    {ok, State}.

done(_State) ->
    ok.
```

Fitting Example

```
-module(my_app).
-behaviour(riak_pipe_vnode_worker).
-export([render_comment/1]).

render_comment({ok, RiakObject}) ->
    Value = riak_object:get_value(RiakObject),
    HTML = ...render Value to HTML...
    [HTML];
render_comment({error,Reason}) ->
    HTML = ...render error Reason...
    [HTML].
```

Practical Example

map

reduce

```
[#fitting_spec{name = {kvget_map,0},
               module = riak_kv_pipe_get,
               arg = undefined,
               chashfun = {riak_kv_pipe_get,bkey_chash},
               nval = {riak_kv_pipe_get,bkey_nval},
               q_limit = 64},
 #fitting_spec{name = {xform_map,0},
               module = riak_kv_mrc_map,
               arg = {{modfun,riak_kv_mapreduce,map_object_value},
                      none},
               chashfun = follow,
               nval = 1,
               q_limit = 64},
  #fitting_spec{name = 1,
               module = riak_kv_w_reduce,
               arg = {rct,#Fun<riak_kv_mapreduce.reduce_sum.2>,none},
               chashfun = <<252,254,56,192,68,143,70,78,255,139,154,26,
                            177,15,123,219,36,185,221,145>>,
               nval = 1,
               q_limit = 64}]
```

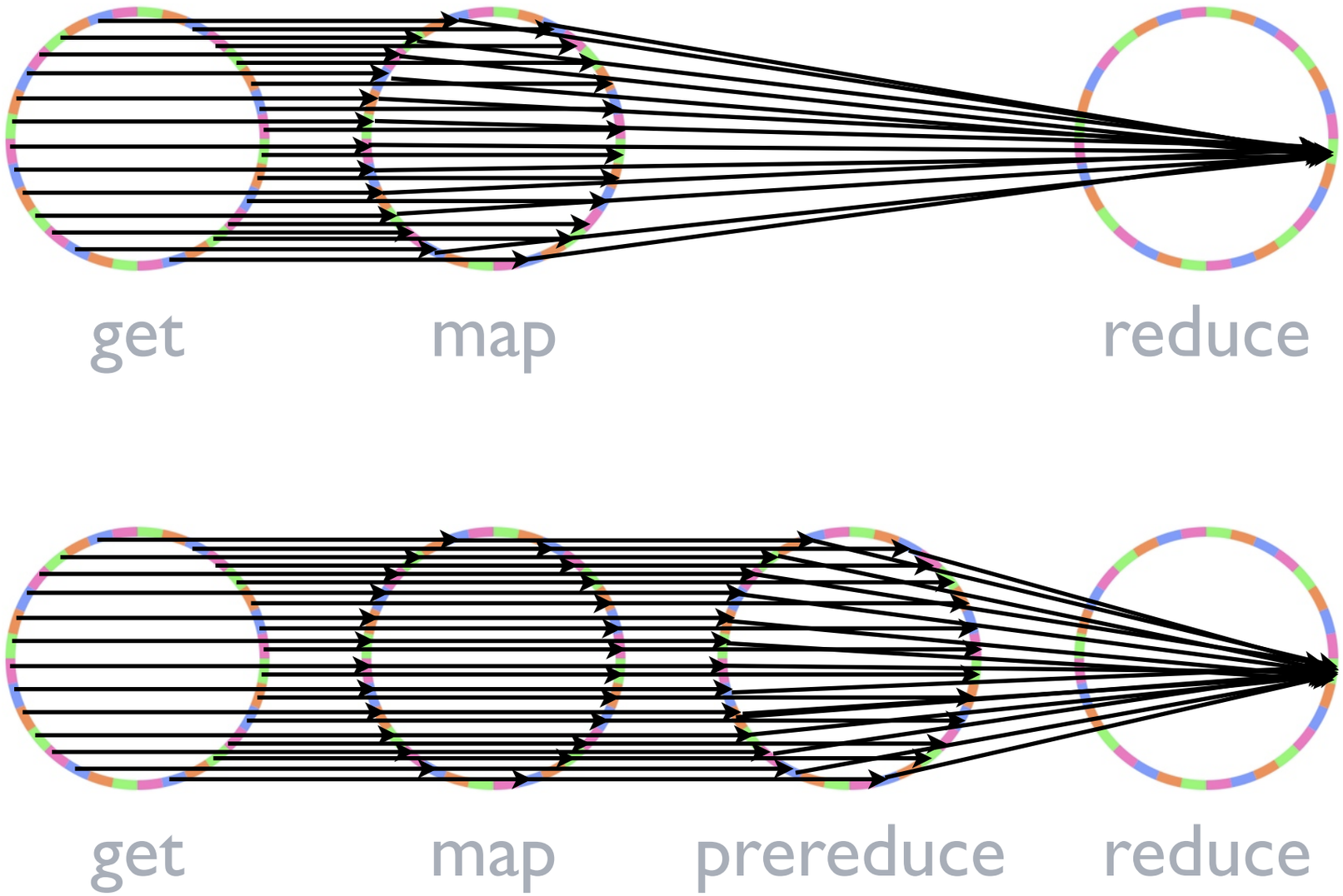
Practical Example

map

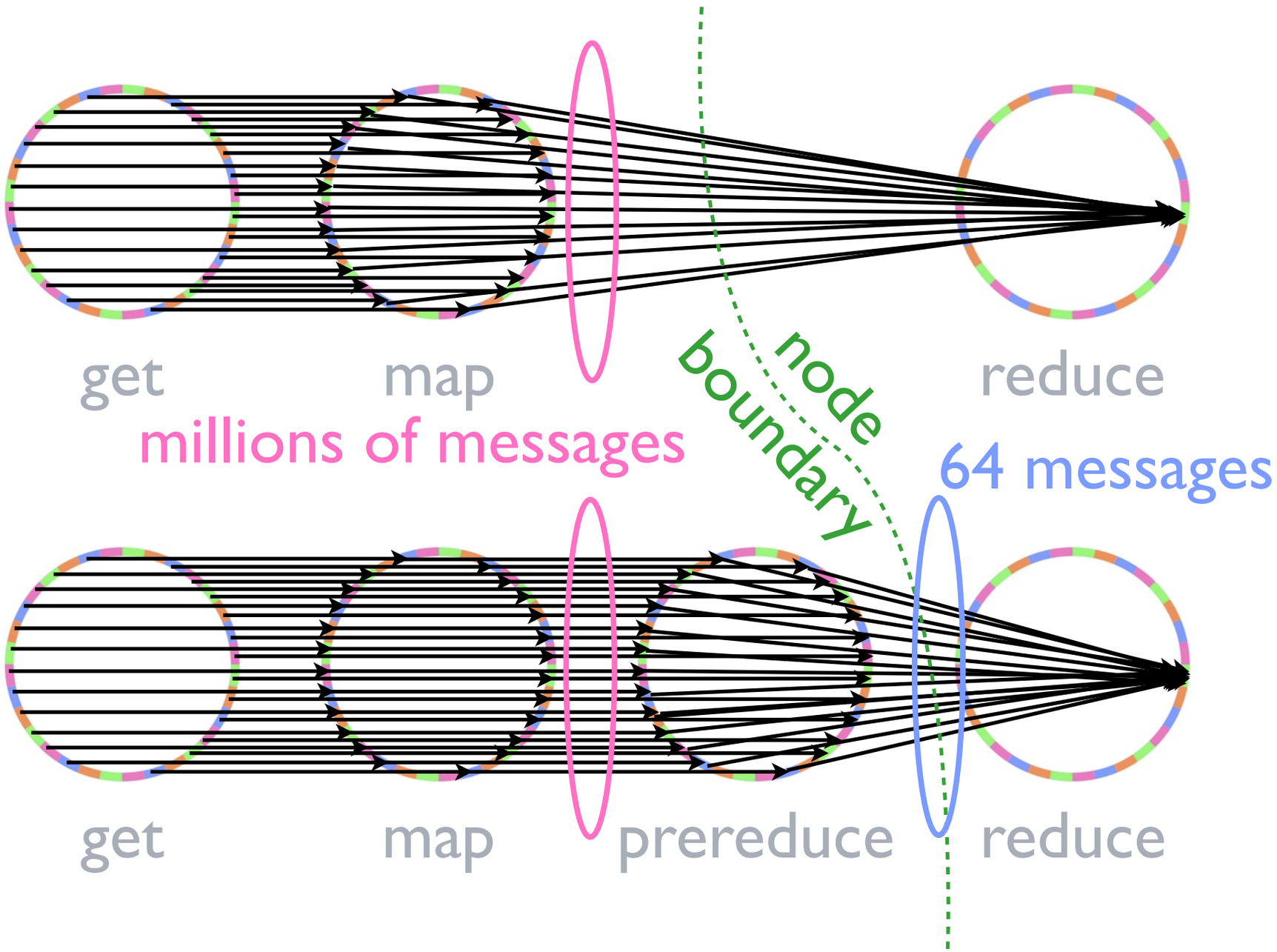
reduce

```
[#fitting_spec{name = {kvget_map,0},
               module = riak_kv_pipe_get,
               arg = undefined,
               chashfun = {riak_kv_pipe_get,bkey_chash},
               nval = {riak_kv_pipe_get,bkey_nval},
               q_limit = 64},
 #fitting_spec{name = {xform_map,0},
               module = riak_kv_mrc_map,
               arg = {{modfun,riak_kv_mapreduce,map_object_value},
                      [do_prereduce]},
               chashfun = follow,
               nval = 1,
               q_limit = 64},
 #fitting_spec{name = {prereduce,0},
               module = riak_kv_w_reduce,
               arg = {rct,#Fun<riak_kv_mapreduce.reduce_sum.2>,none},
               chashfun = follow,
               nval = 1,
               q_limit = 64},
 #fitting_spec{name = 1,
               module = riak_kv_w_reduce,
               arg = {rct,#Fun<riak_kv_mapreduce.reduce_sum.2>,none},
               chashfun = <<252,254,56,192,68,143,70,78,255,139,154,26,
                            177,15,123,219,36,185,221,145>>,
               nval = 1,
               q_limit = 64}]
```

Practical Example



Practical Example



Possible Apps?

- KV Could be redone as a Pipe app
- Get = fetch -> reconcile -> repair
- Put = pre-commit -> write -> post-commit

Current Work

- Inter-worker messaging efficiency
- Other-language Fittings
- External Interface
- Long-running Pipelines

Thanks!

- Source: github.com/basho/riak_pipe
- Paper: beerriot.com/bryan.html ☛ 
- Me: @hobbyist
- News: twitter.com/basho/team