The Scalable Stateful Web

with Phoenix and Riak Core

Ben Tyler
Booking.com

ElixirConf.EU 2016

What?

What?

"joy of cool tech"

What?

"joy of cool tech"

Talk notes and code will be available w/ the slides.

What?

"joy of cool tech"

Talk notes and code will be available w/ the slides.

Let’s chat after the conf!

Buzzword Bingo

Stateful

Buzzword Bingo

Stateful

Distributed

Buzzword Bingo

Stateful

Distributed

Fault-tolerant

Buzzword Bingo

Stateful

Distributed

Fault-tolerant

Real-time

Buzzword Bingo

Stateful

Distributed

Fault-tolerant

Real-time

Impress your cat

Buzzword Bingo

Stateful

Distributed

Fault-tolerant

Real-time

Impress your cat

(application)

Buzzword Bingo

Stateful

Distributed

Fault-tolerant

Real-time

Impress your cat

(application)

Stateful

Memory that lasts for more than one request

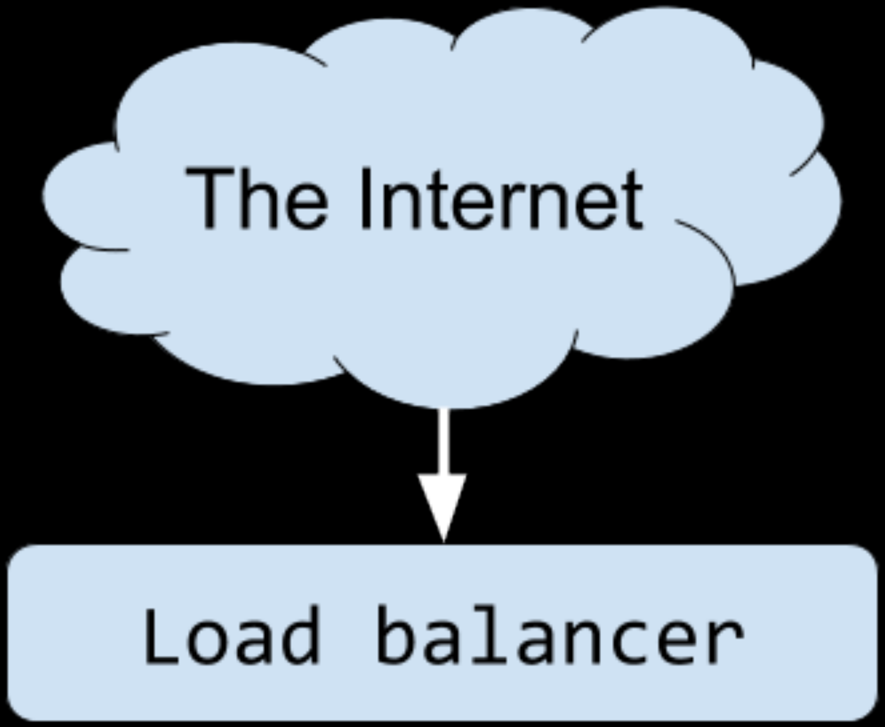
Stateful

(stateless?)

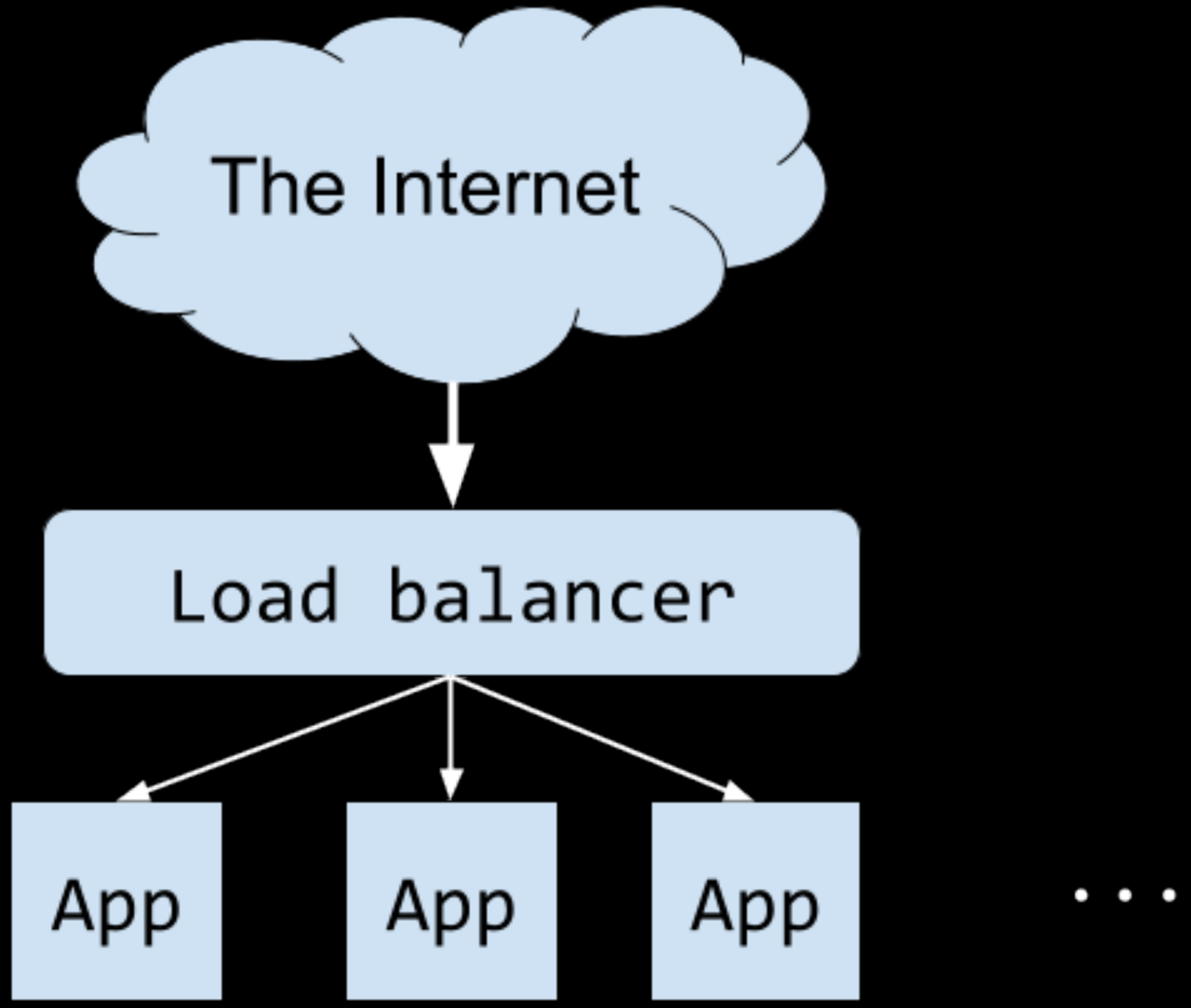
State*less*

State*less*

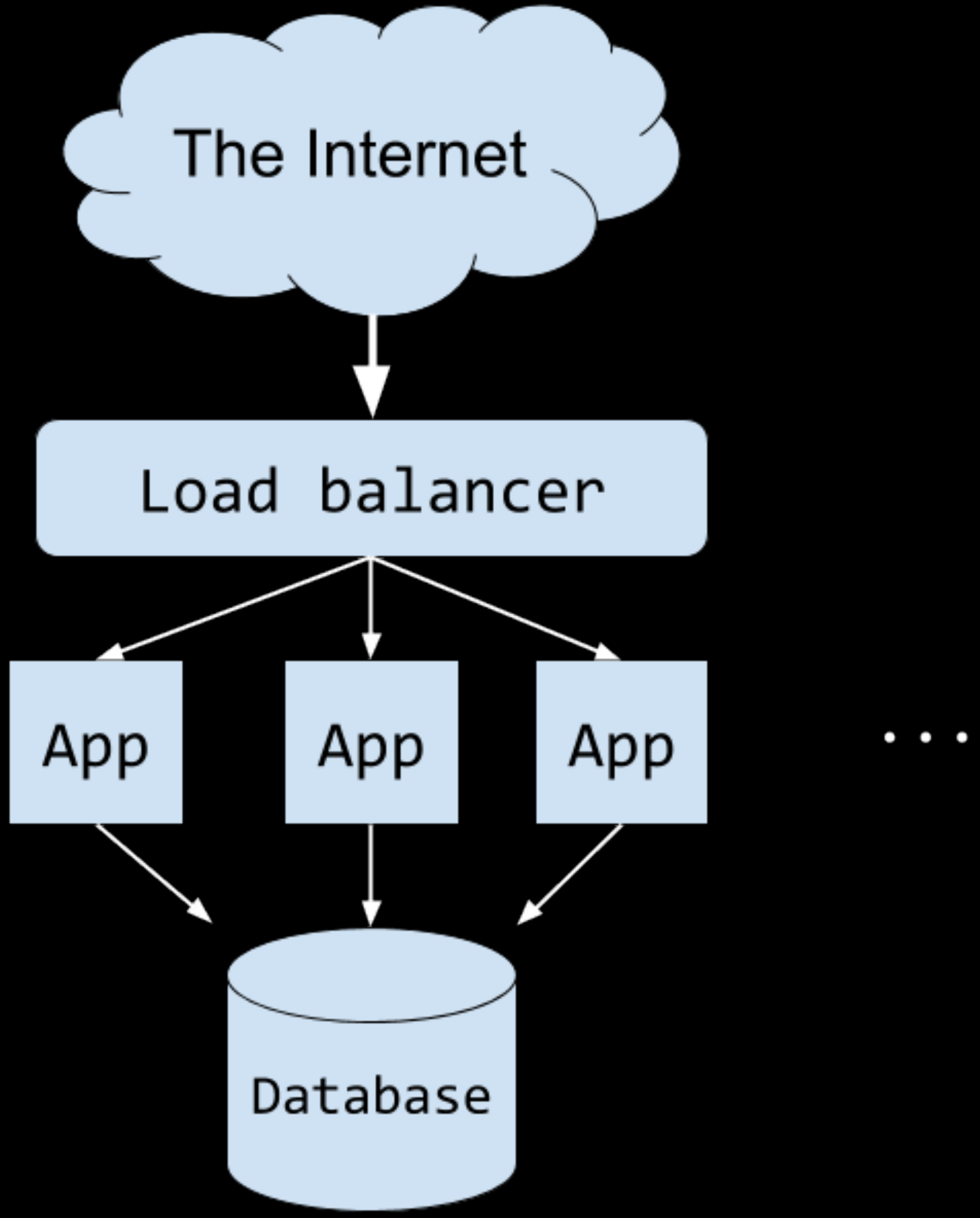
State*less*



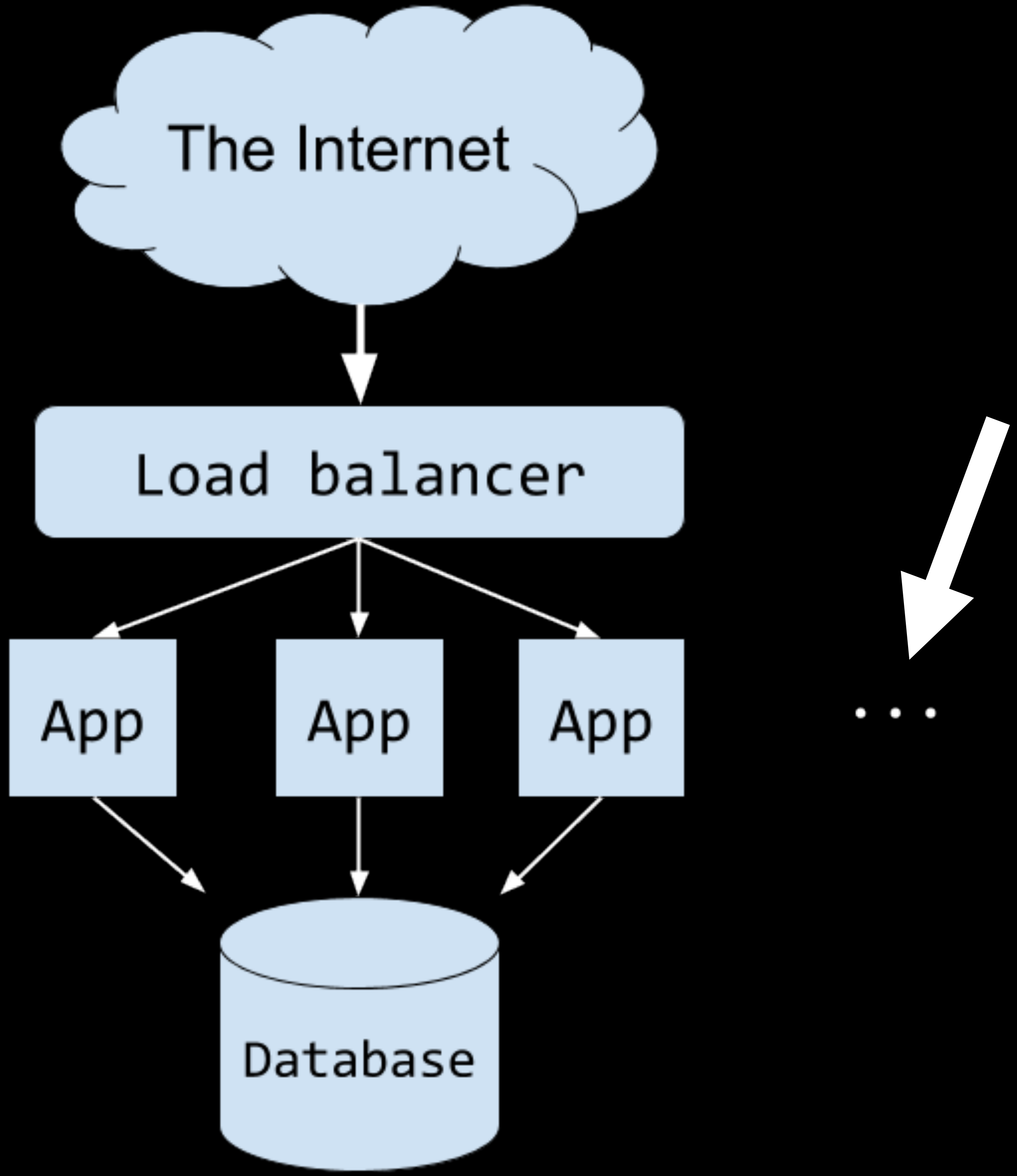
State*less*



State*less*

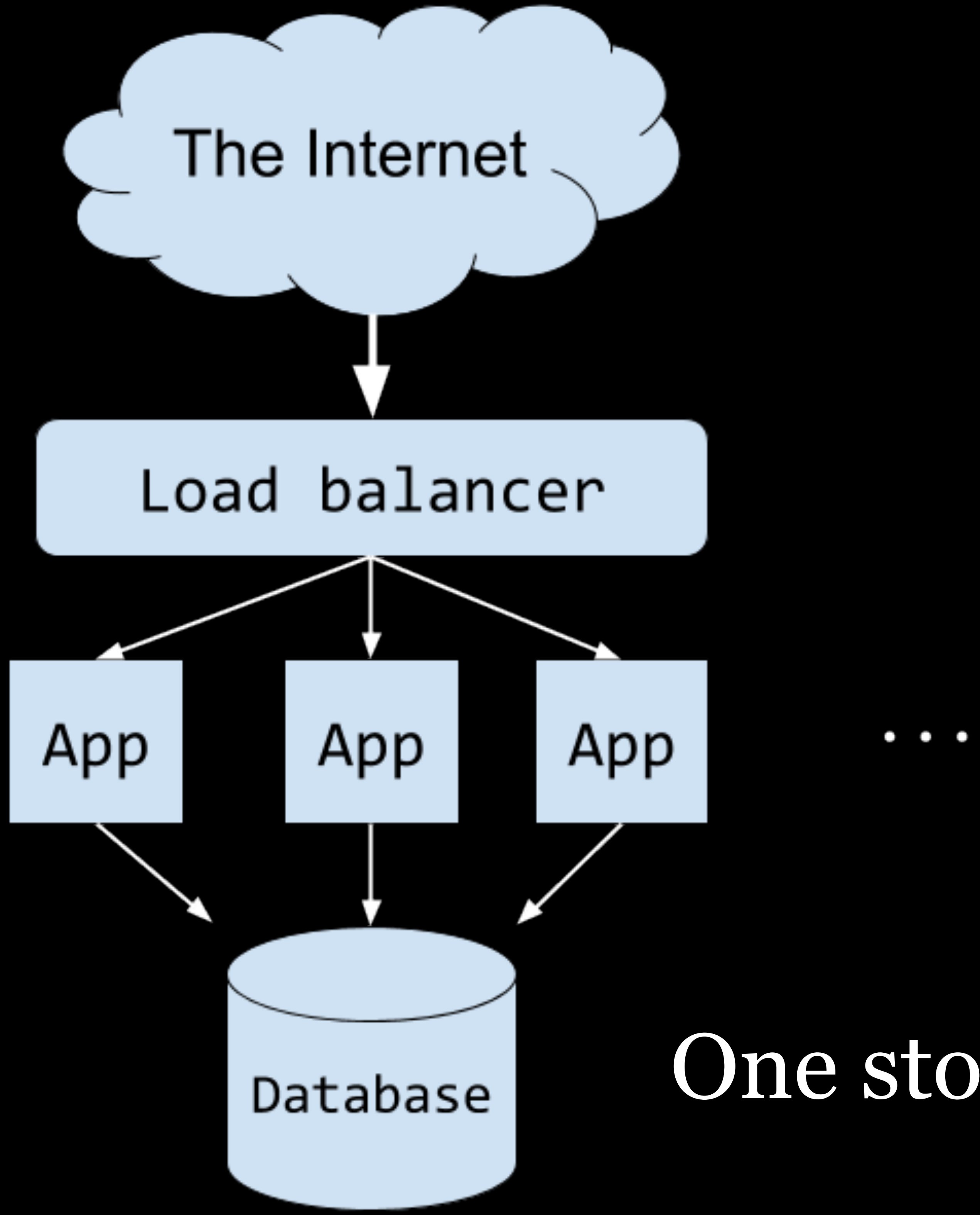


State*less*



Horizontally scalable

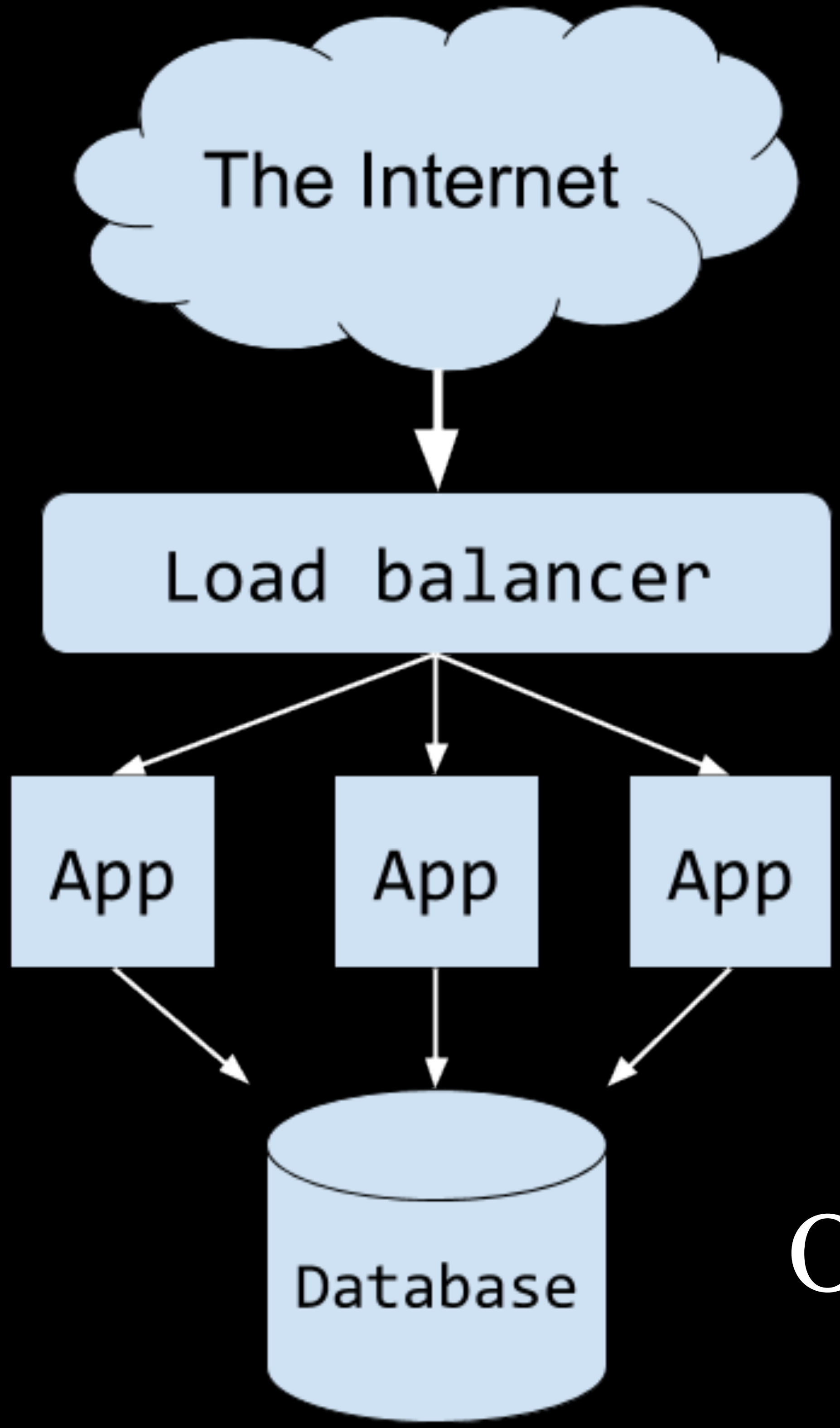
State*less*



Horizontally scalable

One stop shop

State*less*



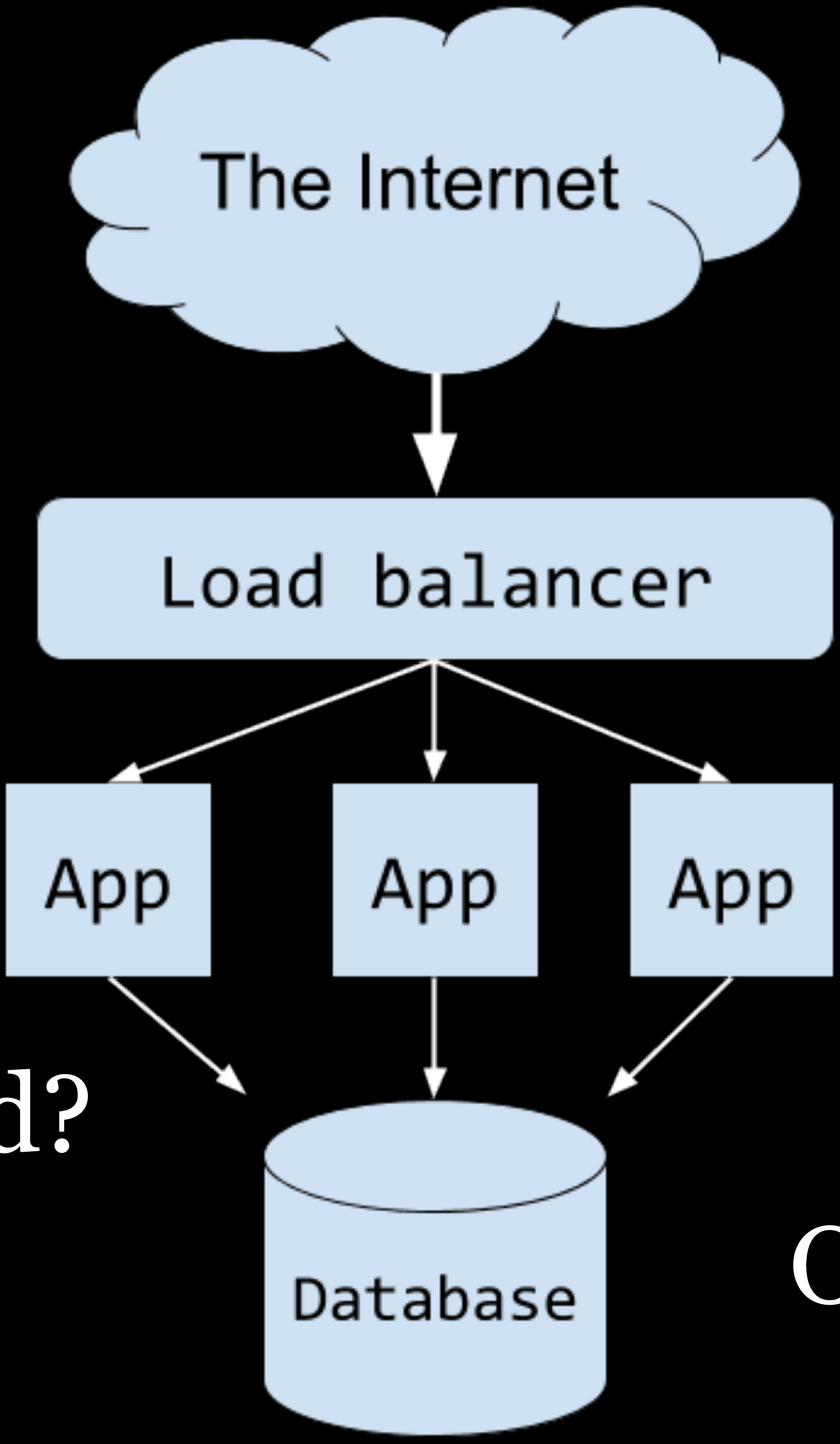
Simple load balancing

Horizontally scalable

. . .

One stop shop

State*less*



Simple load balancing

Horizontally scalable

. . .

Memory wasted?

One stop shop

Latency Numbers

CPU L1 cache reference	0:00:01

- github.com/kofemann

Latency Numbers

CPU L1 cache reference	0:00:01
Main memory reference	0:03:20

- github.com/kofemann

Latency Numbers

CPU L1 cache reference	0:00:01
Main memory reference	0:03:20
Read 1MB sequentially from memory	6 days

- github.com/kofemann

Latency Numbers

CPU L1 cache reference	0:00:01
Main memory reference	0:03:20
Read 1MB sequentially from memory	6 days
Network round trip, same datacenter	11.5 days

- github.com/kofemann

Latency Numbers

CPU L1 cache reference	0:00:01
Main memory reference	0:03:20
Read 1MB sequentially from memory	6 days
Network round trip, same datacenter	11.5 days
Read 1MB sequentially from disk	463 days

- *github.com/kofemann*

Latency Numbers

CPU L1 cache reference	0:00:01
Main memory reference	0:03:20
Read 1MB sequentially from memory	6 days
Network round trip, same datacenter	11.5 days
Read 1MB sequentially from disk	463 days

- *github.com/kofemann*

Latency Numbers

CPU L1 cache reference	0:00:01
Main memory reference	0:03:20
Read 1MB sequentially from memory	6 days
Network round trip, same datacenter	11.5 days
Read 1MB sequentially from disk	463 days

- github.com/kofemann

Latency Numbers

CPU L1 cache reference	0:00:01
Main memory reference	0:03:20
Read 1MB sequentially from memory	6 days <- do this!
Network round trip, same datacenter	11.5 days
Read 1MB sequentially from disk	463 days

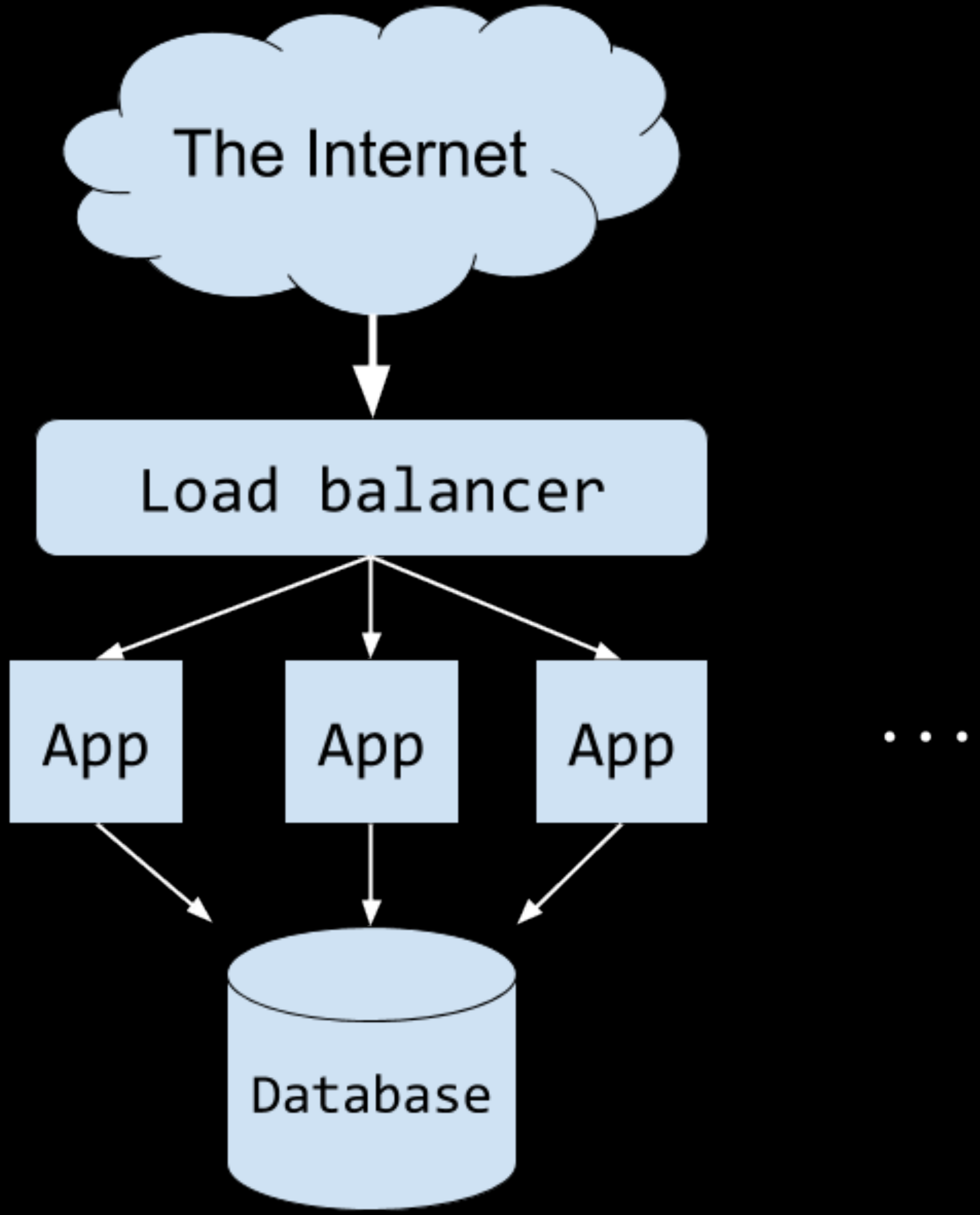
- github.com/kofemann

Latency Numbers

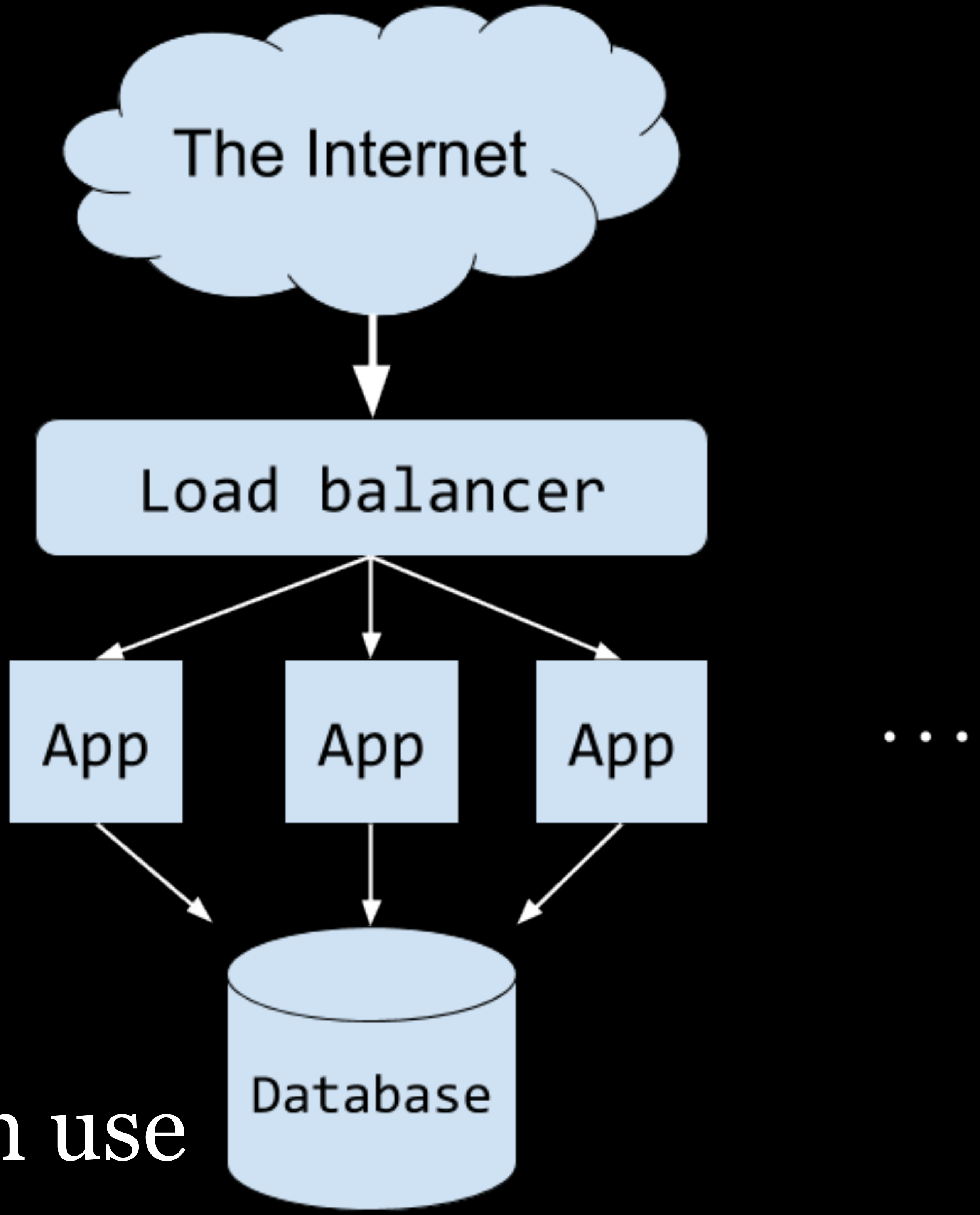
CPU L1 cache reference	0:00:01
Main memory reference	0:03:20
Read 1MB sequentially from memory	6 days <- do this!
Network round trip, same datacenter	11.5 days <- and this!
Read 1MB sequentially from disk	463 days

- github.com/kofemann

State*less*

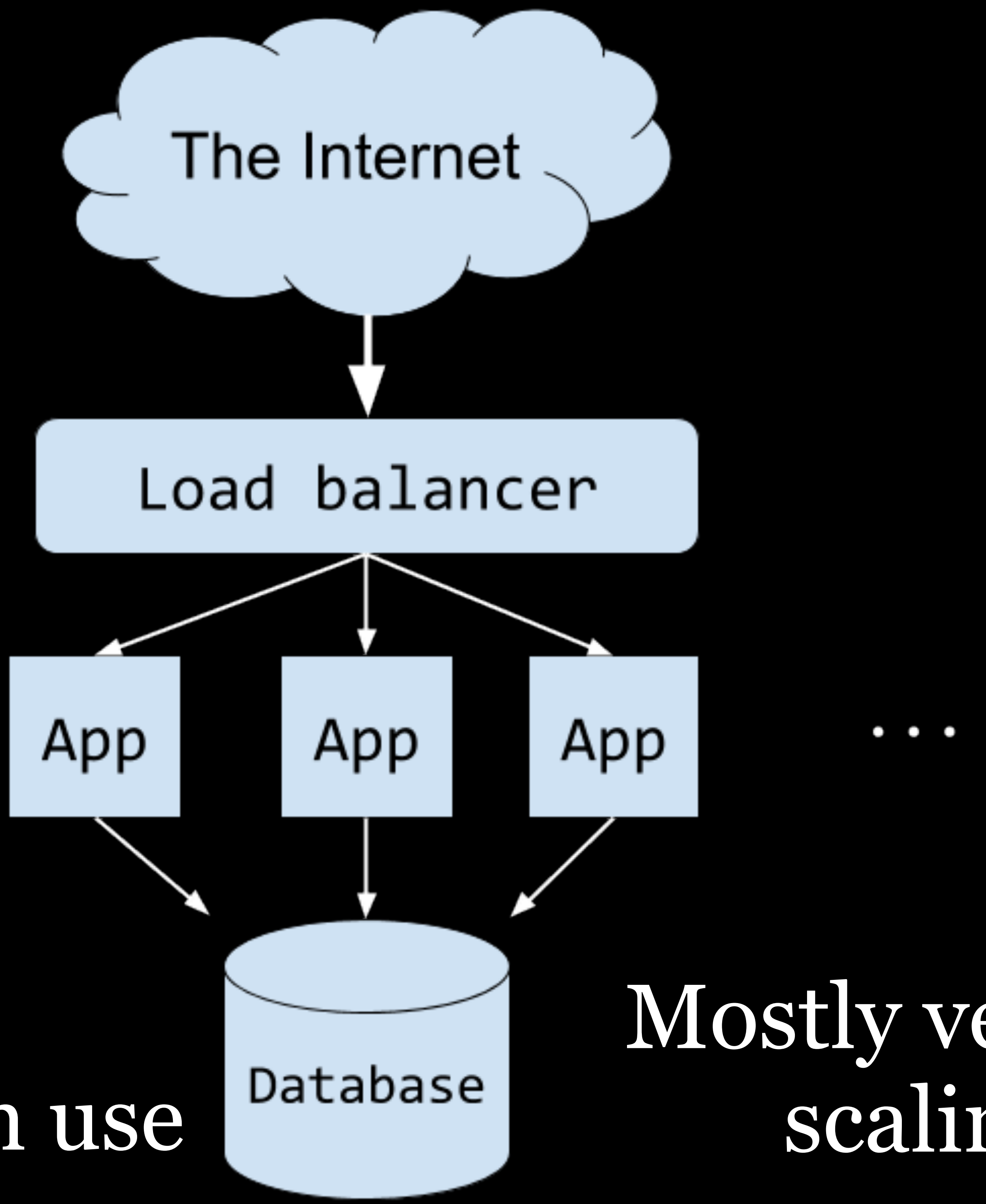


State*less*



Memory we can use

State*less*



Memory we can use

Mostly vertical
scaling

Why state*less*?

Why state*less*?

- Workload (HTTP)

Why state*less*?

- Workload (HTTP)
- Hard to reuse memory

Why state*less*?

- Workload (HTTP)
- Hard to reuse memory
  - Short lived programs

Why state*less*?

- Workload (HTTP)
- Hard to reuse memory
    - Short lived programs
    - Single threaded programs

Why state*less*?

- Workload (HTTP)
- Hard to reuse memory
    - Short lived programs
    - Single threaded programs
    - Tricky to coordinate servers

And for Elixir/Phoenix?

And for Elixir/Phoenix?

- Workload (HTTP)

And for Elixir/Phoenix?

- Workload (HTTP) —> Channels!

And for Elixir/Phoenix?

- Workload (HTTP) —> Channels!
- Hard to reuse memory

And for Elixir/Phoenix?

- Workload (HTTP) —> Channels!
- Hard to reuse memory
    - Short lived programs

And for Elixir/Phoenix?

- Workload (HTTP) —> Channels!
- Hard to reuse memory
  - Short lived programs —> BEAM!

And for Elixir/Phoenix?

- Workload (HTTP) —> Channels!
- Hard to reuse memory
  - Short lived programs —> BEAM!
  - Single threaded programs

And for Elixir/Phoenix?

- Workload (HTTP) —> Channels!
- Hard to reuse memory
  - Short lived programs —> BEAM!
  - Single threaded programs —> BEAM!

And for Elixir/Phoenix?

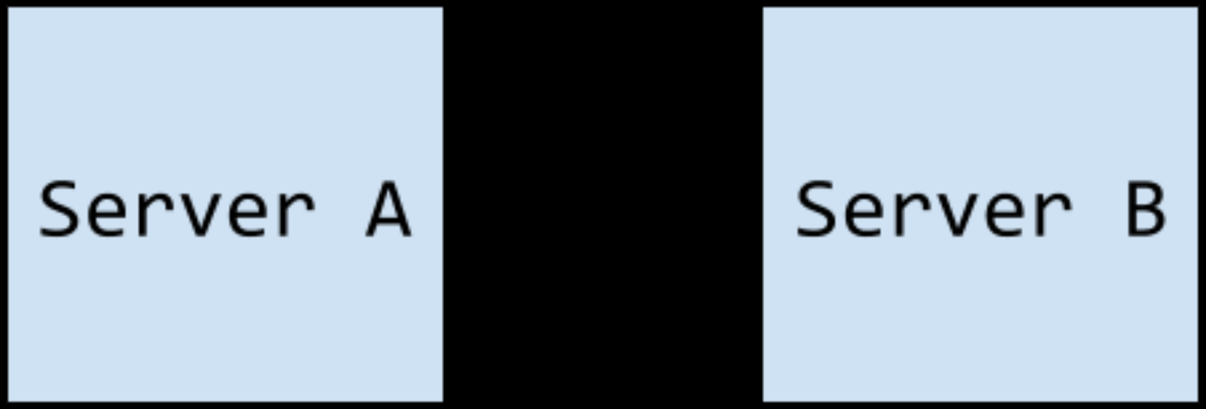
- Workload (HTTP) —> Channels!
- Hard to reuse memory
  - Short lived programs —> BEAM!
  - Single threaded programs —> BEAM!
  - Tricky to coordinate servers

And for Elixir/Phoenix?

- Workload (HTTP) —> Channels!
- Hard to reuse memory
  - Short lived programs —> BEAM!
  - Single threaded programs —> BEAM!
  - Tricky to coordinate servers —> BEAM!

Awesome! Let’s build
a stateful web app!

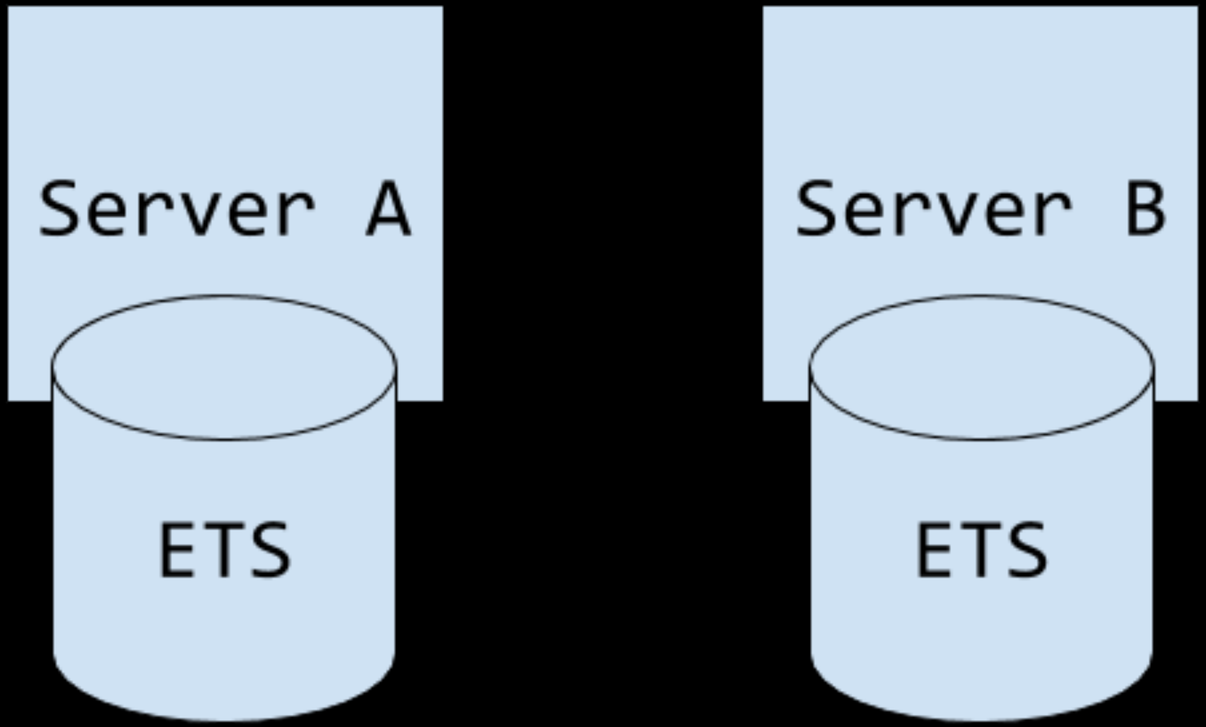
A stateful web app



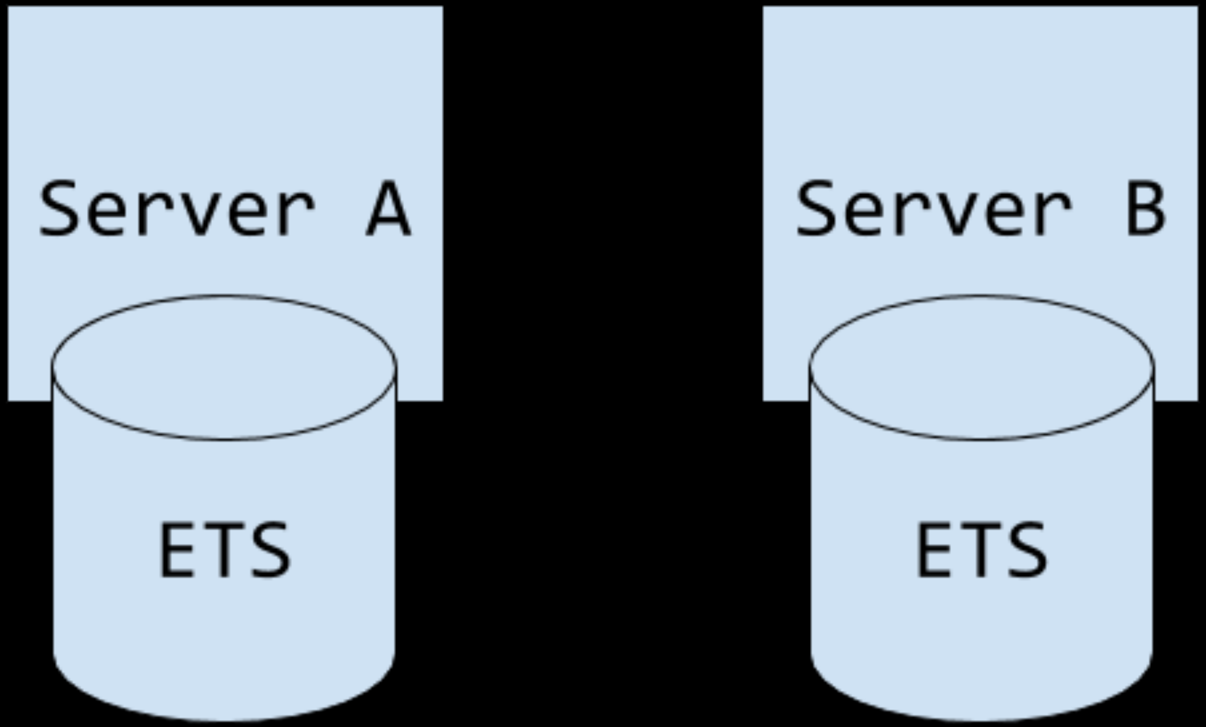
Server A

Server B

A stateful web app



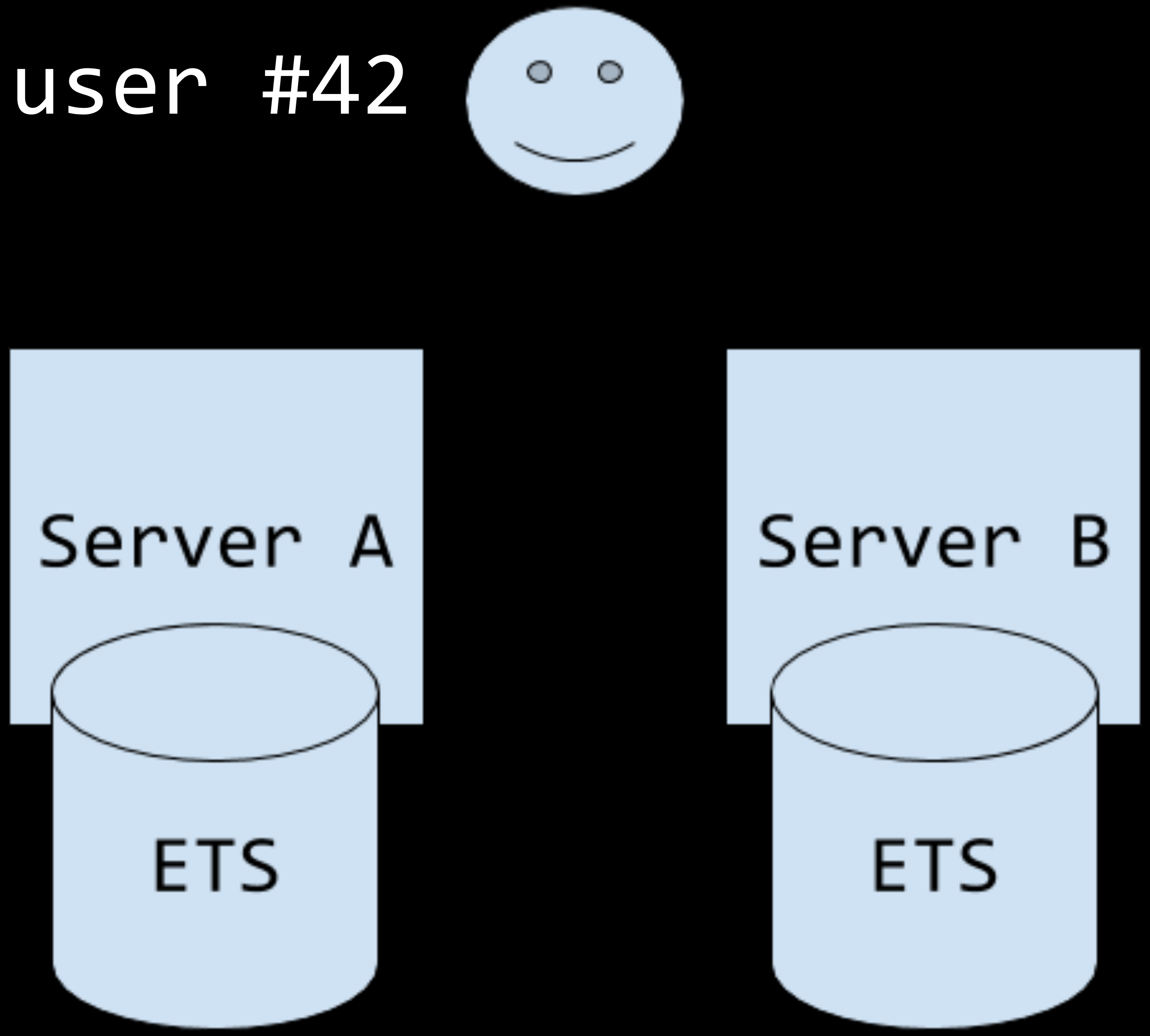
A stateful web app



Data for user #42

A stateful web app

user #42



Server A

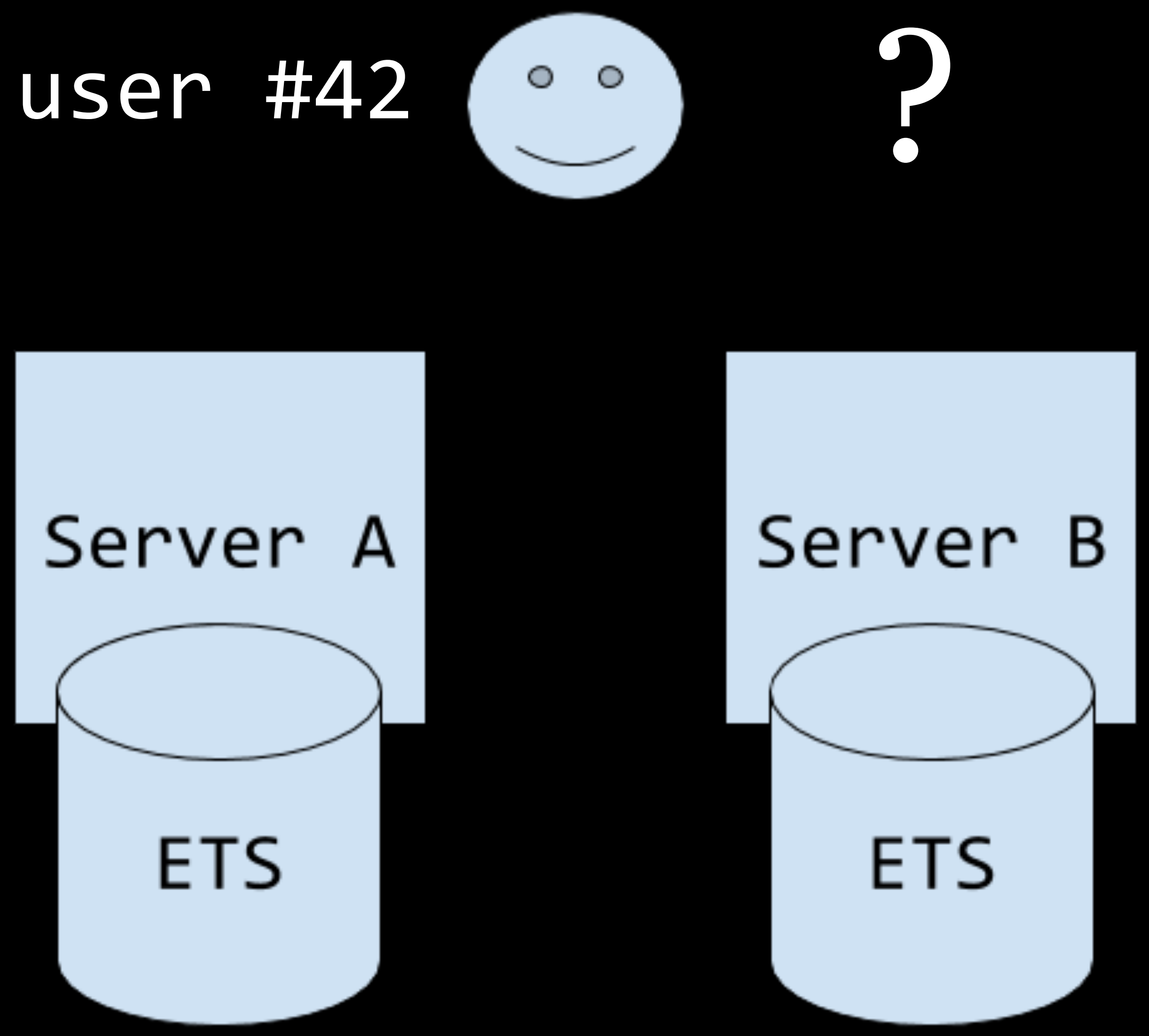
ETS

Server B

ETS

Data for user #42

A stateful web app



user #42

?

Server A

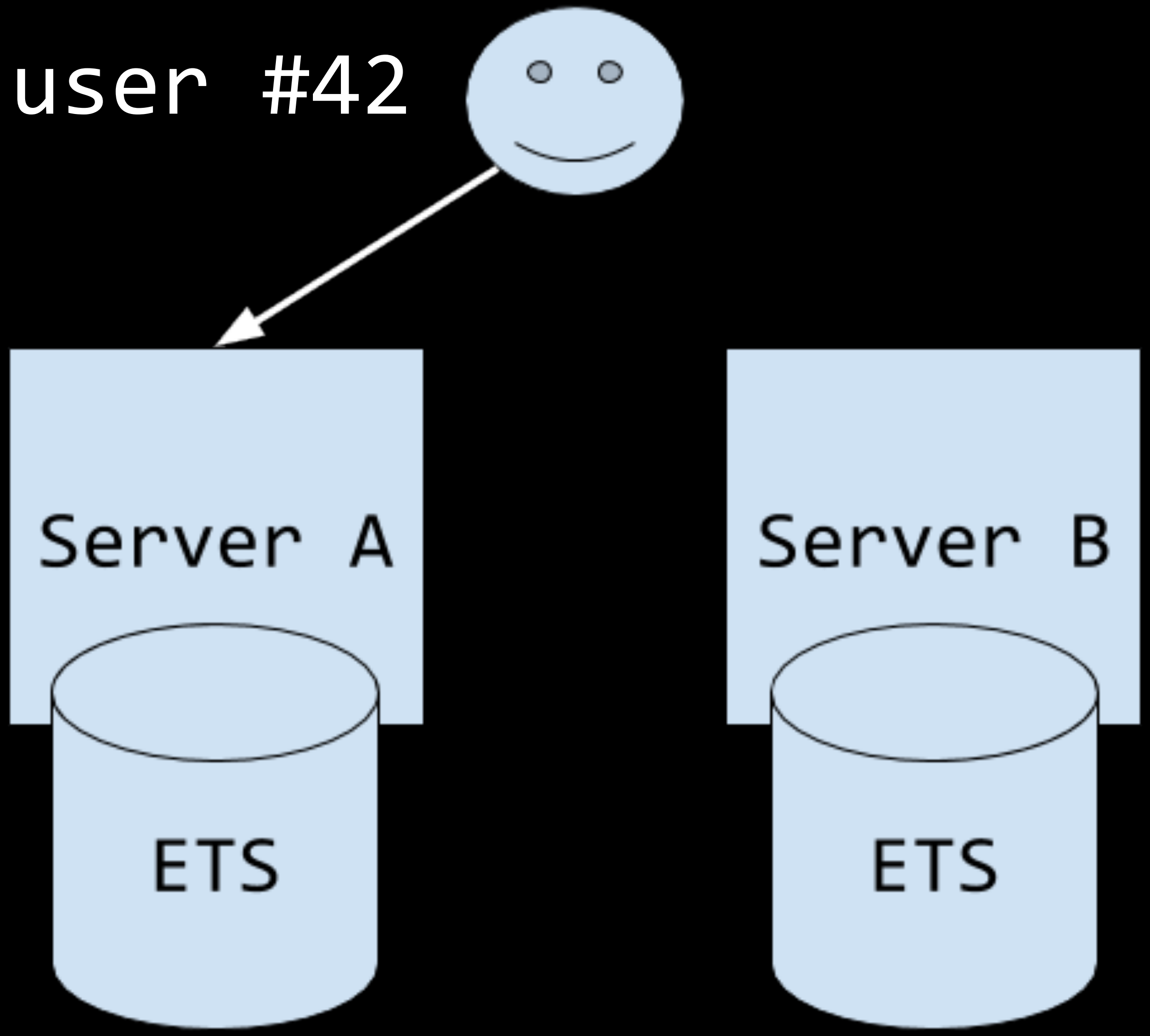
ETS

Server B

ETS

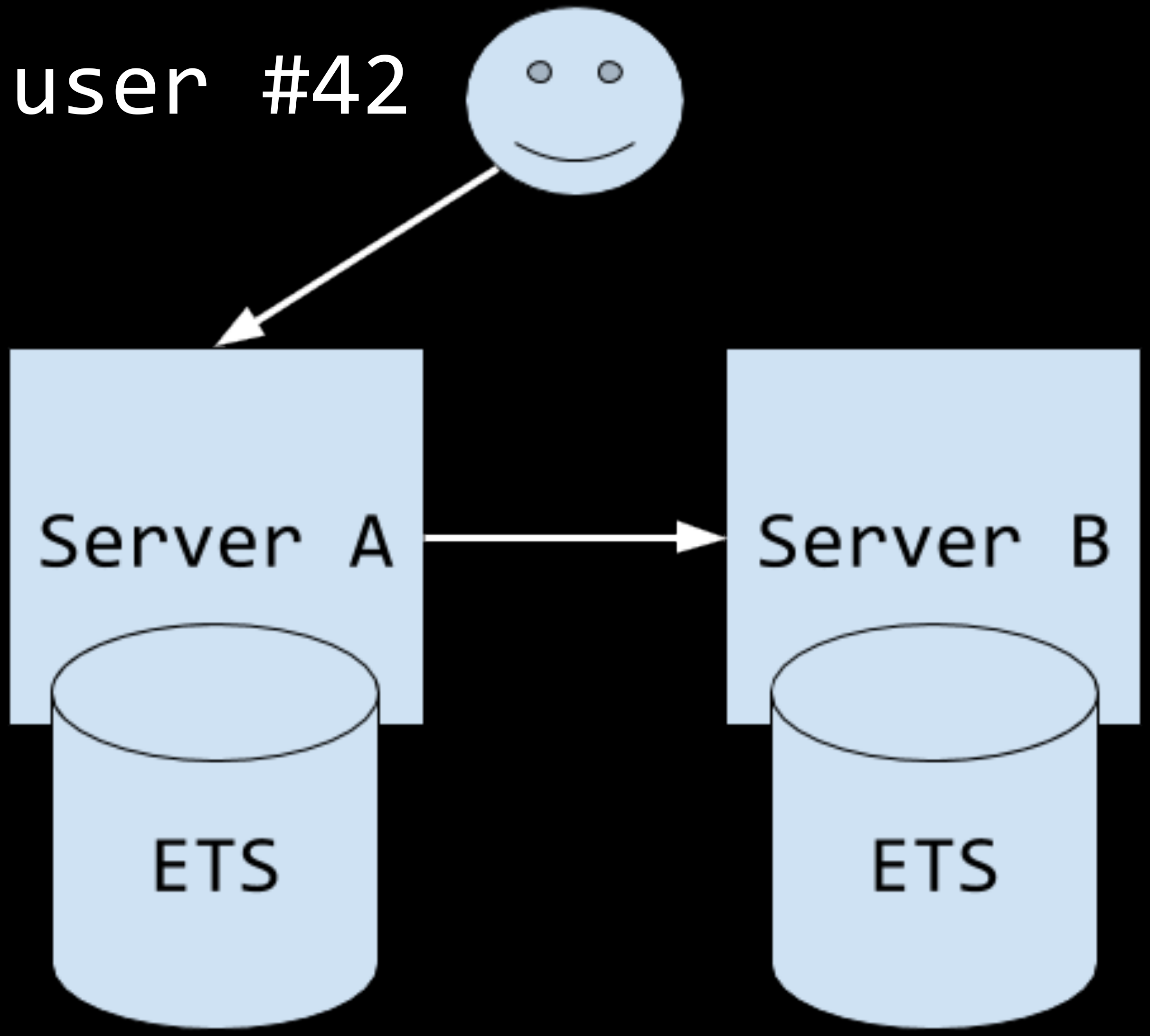
Data for user #42

A stateful web app



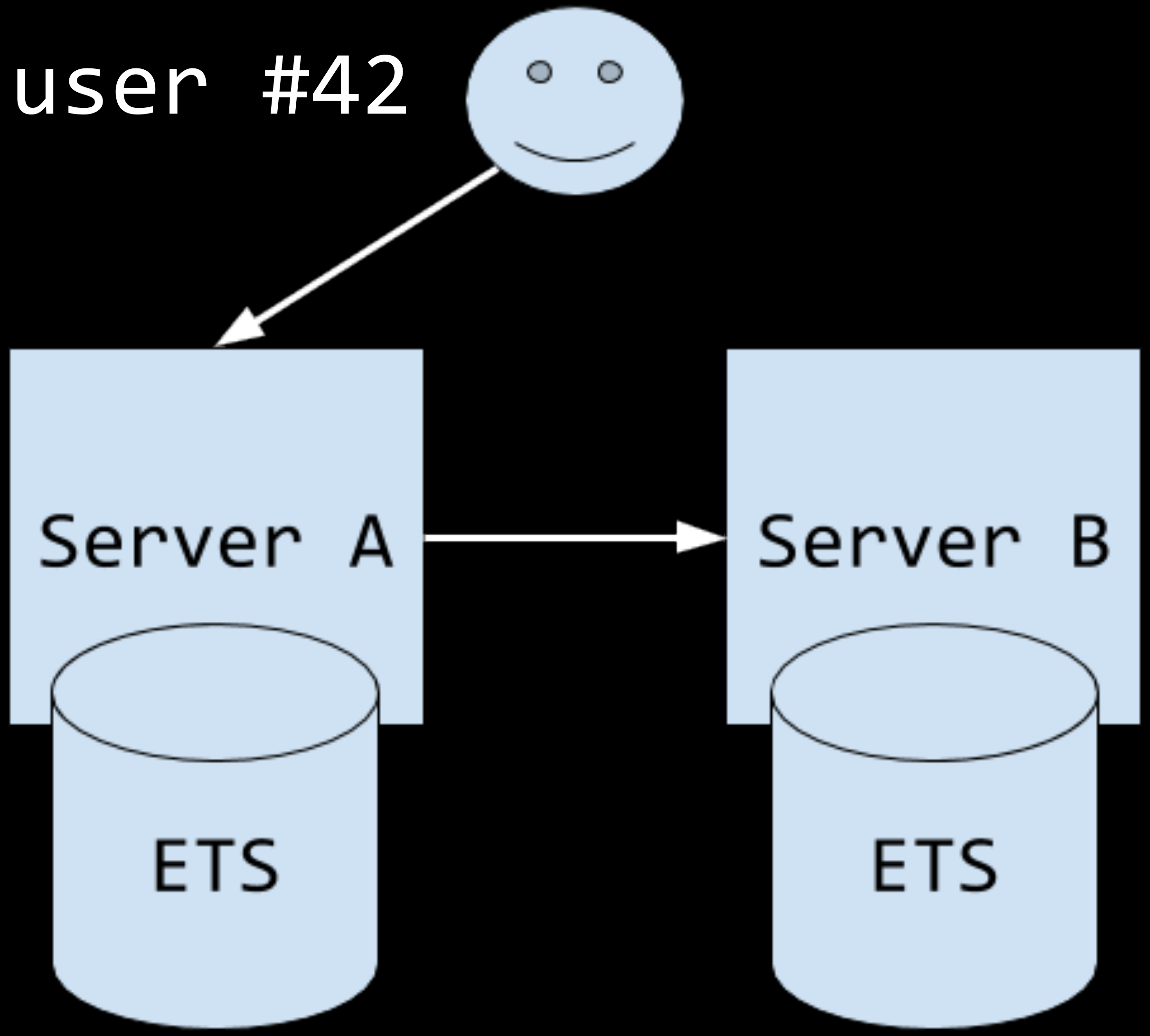
Data for user #42

A stateful web app



Data for user #42

A stateful web app

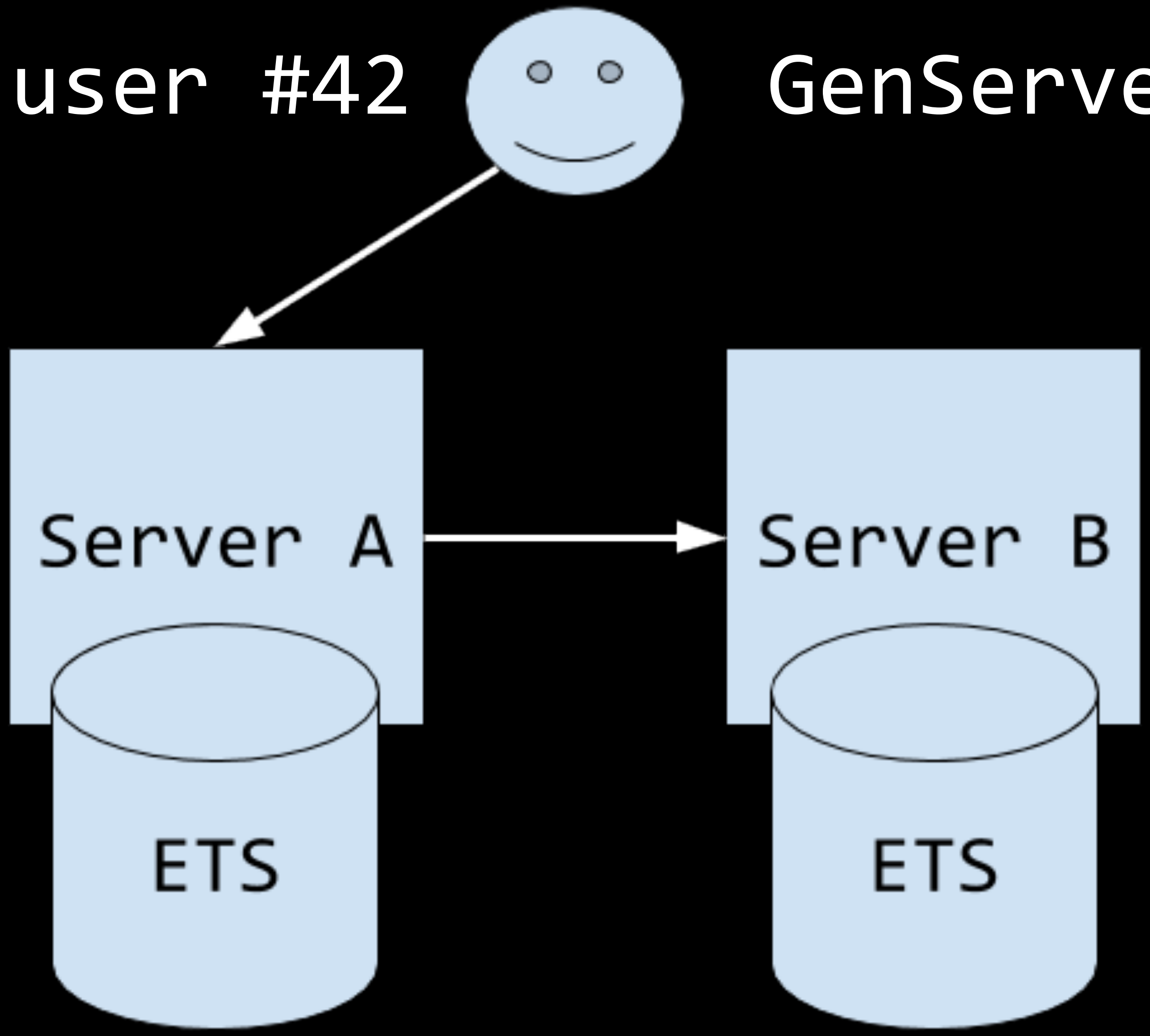


```
send remote_pid,
   {:get_user_data, 42}
```

Data for user #42

A stateful web app

user #42

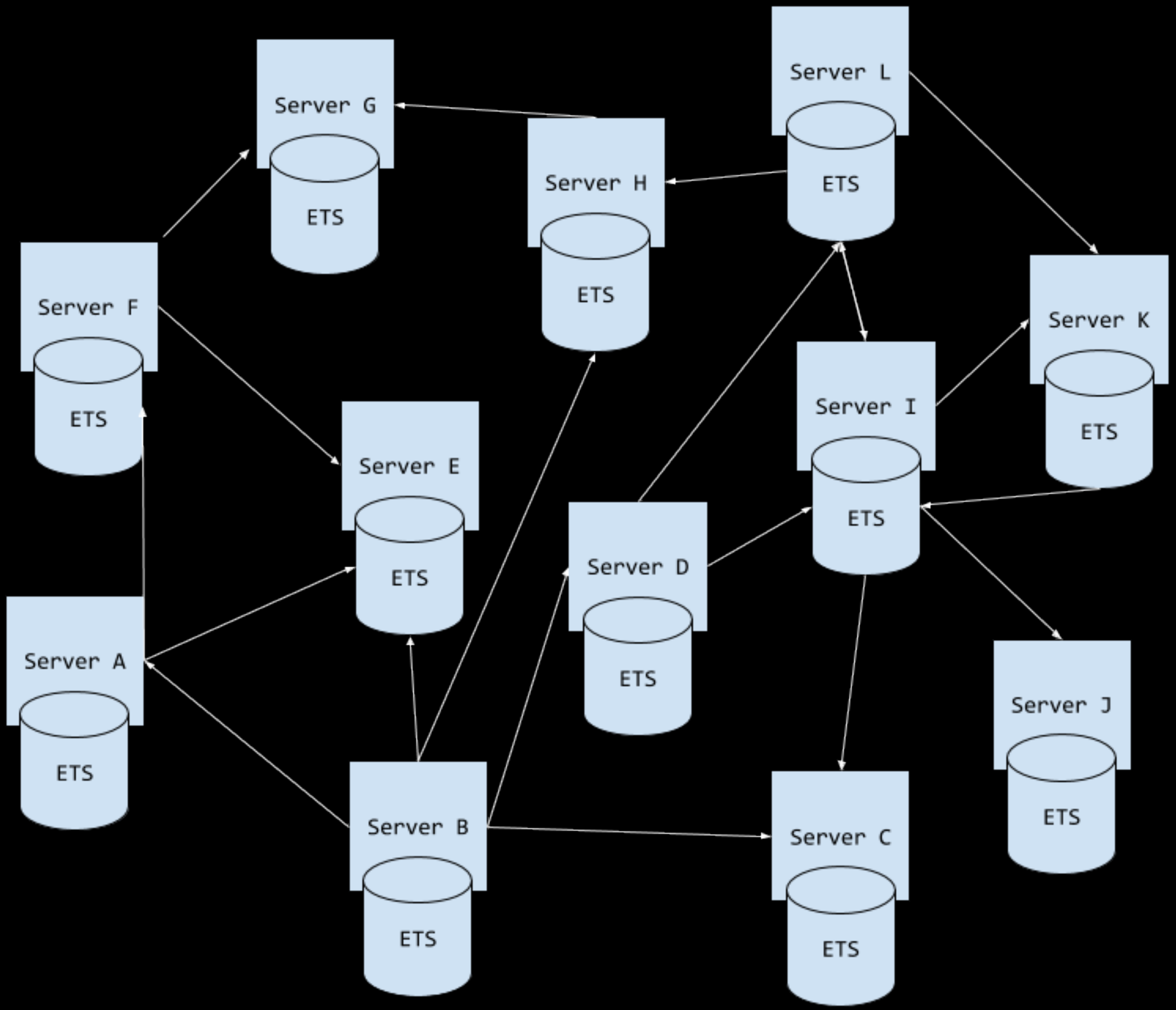


GenServer.call(…, …)

```
send remote_pid,
   {:get_user_data, 42}
```

Data for user #42

ARE YOU SURE?


Server G
ETS
Server L
ETS
Server H
ETS
Server F
ETS
Server K
ETS
Server I
ETS
Server E
ETS
Server D
ETS
Server A
ETS
Server J
ETS
Server B
ETS
Server C
ETS

DISTRIBUTED SYSTEMS

Buzzword Bingo

Stateful

Distributed

Fault-tolerant

Real-time

Impress your cat

(application)

Buzzword Bingo

Stateful - ✓

Distributed

Fault-tolerant

Real-time

Impress your cat

(application)

Distributed Primitives and Patterns

Distributed Primitives and Patterns

```
send(remote_pid, …), GenServer.call/cast
```

distributed systems

Distributed Primitives and Patterns

```
send(remote_pid, …), GenServer.call/cast
```

distributed systems

≈

Distributed Primitives and Patterns

```
send(remote_pid, …), GenServer.call/cast
```

distributed systems

≈

```
print("<div>")
```

web development

Distributed Primitives and Patterns

distributed systems

≈

Distributed Primitives and Patterns

ad hoc message passing or RPC

distributed systems

≈

Distributed Primitives and Patterns

ad hoc message passing or RPC

distributed systems

≈

```
<?php
```

Distributed Primitives and Patterns

ad hoc message passing or RPC

distributed systems

≈

```
<?php
    $foo = mysql_query($my_cool_query);
```

Distributed Primitives and Patterns

ad hoc message passing or RPC

distributed systems

≈

```
<?php
    $foo = mysql_query($my_cool_query);
    echo "<div>$foo</div>";
```

Distributed Primitives and Patterns

ad hoc message passing or RPC distributed systems

≈

```
<?php
    $foo = mysql_query($my_cool_query);
    echo "<div>$foo</div>";
?>
```

Distributed Primitives and Patterns

ad hoc message passing or RPC

distributed systems

≈

```
<?php
    $foo = mysql_query($my_cool_query);
    echo "<div>$foo</div>";
?>
```

web development

Distributed Primitives and Patterns

```php
<?php
    $foo = mysql_query($my_cool_query);
    echo "<div>$foo</div>";
?>
```

Distributed Primitives and Patterns

$$$$$$$

```
<?php
    $foo = mysql_query($my_cool_query);
    echo "<div>$foo</div>";
?>
```

Distributed
Primitives and Patterns

I want my MVC

Distributed Primitives and Patterns

I want my MVC

what is the MVC of distributed systems?

Distributed
CAP

Distributed
CAP

Consistency

Distributed
CAP

Consistency

Availability

Distributed
CAP

Consistency

Availability

Partition tolerance

Distributed
CAP

Consistency

"Pick two"

Availability

Partition tolerance

Distributed CAP

Distributed
CAP

<u>**CP**</u>

<u>**AP**</u>

<u>**CA**</u>

Distributed
CAP

<u>**CP**</u> <u>**AP**</u>

<u>**CA**</u>

Distributed
CAP

CP

AP

CA

Distributed
CAP

AP

Distributed
CAP

<u>AP</u>

gossip protocols

Distributed CAP

AP

gossip protocols

CRDTs

Distributed
CAP

AP

gossip protocols

CRDTs

distributed hash tables

Distributed
CAP

AP

gossip protocols

CRDTs

distributed hash tables

Awesome! Let’s build a stateful, distributed web app using a distributed hash table!

A stateful, distributed web app with a DHT

A stateful, distributed web app with a DHT

Use a framework, don’t write one

Riak Core

Riak Core

"a toolkit for building distributed, scalable, fault-tolerant applications"

- riak_core README

Riak Core

mind blowing, advanced technology

Riak Core

mind blowing, advanced technology

without peer in other platforms

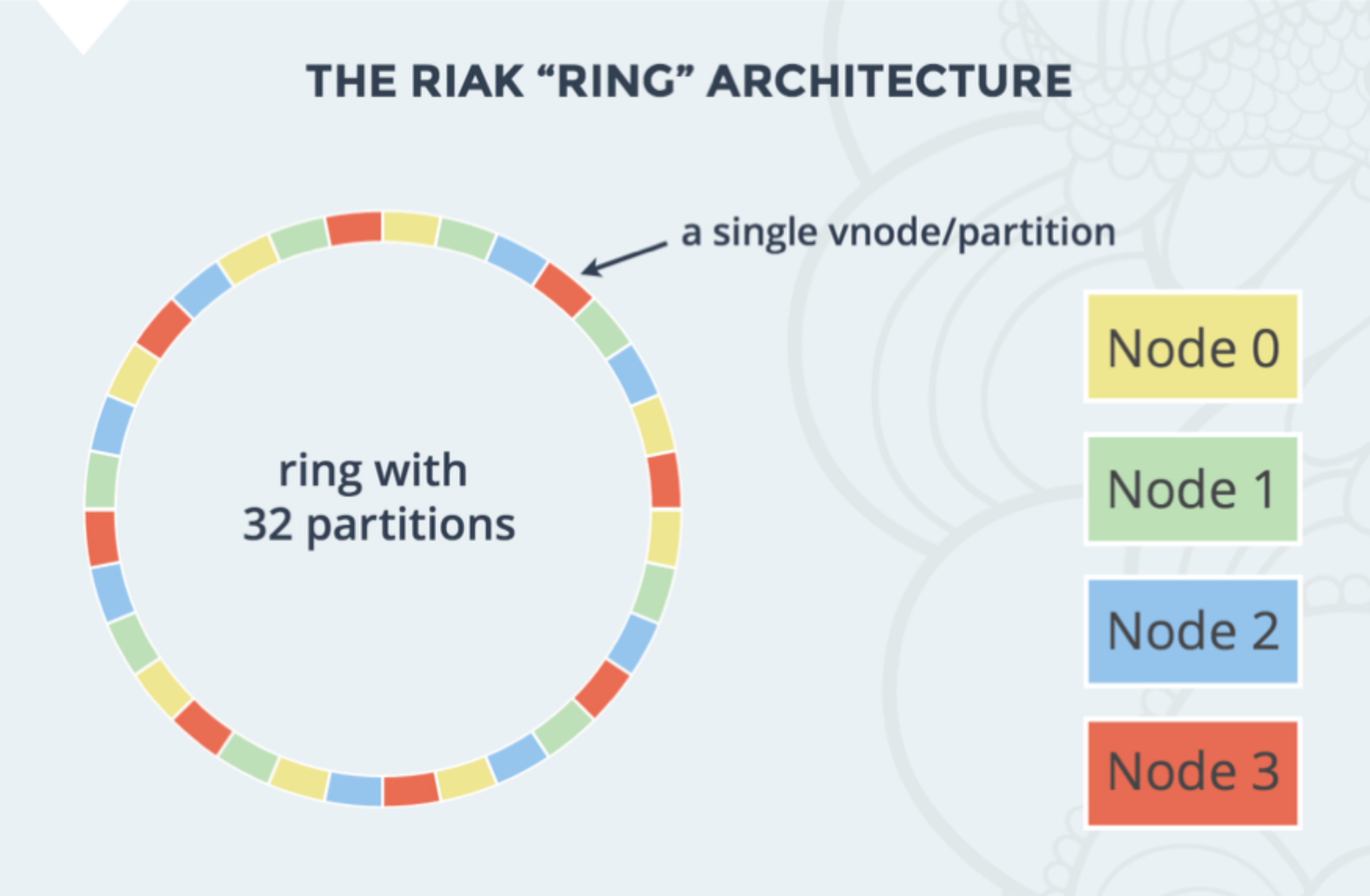
Riak Core

mind blowing, advanced technology

without peer in other platforms

source of magic for Basho + others

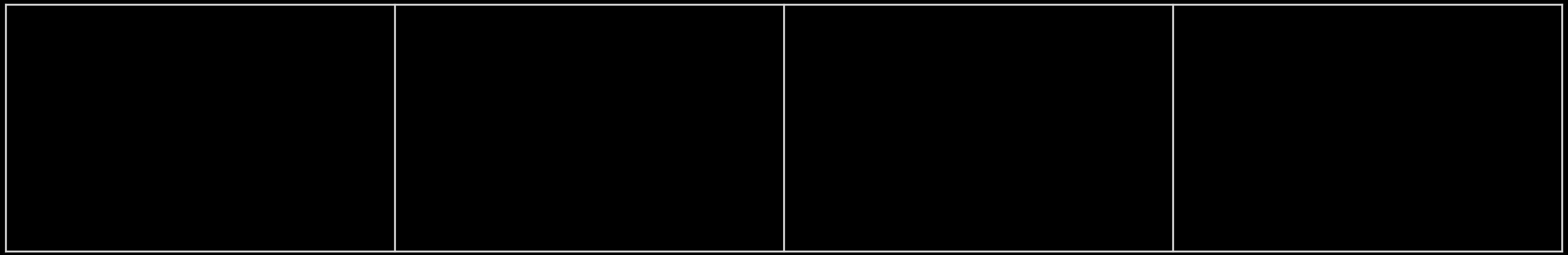
Riak Core



THE RIAK “RING” ARCHITECTURE

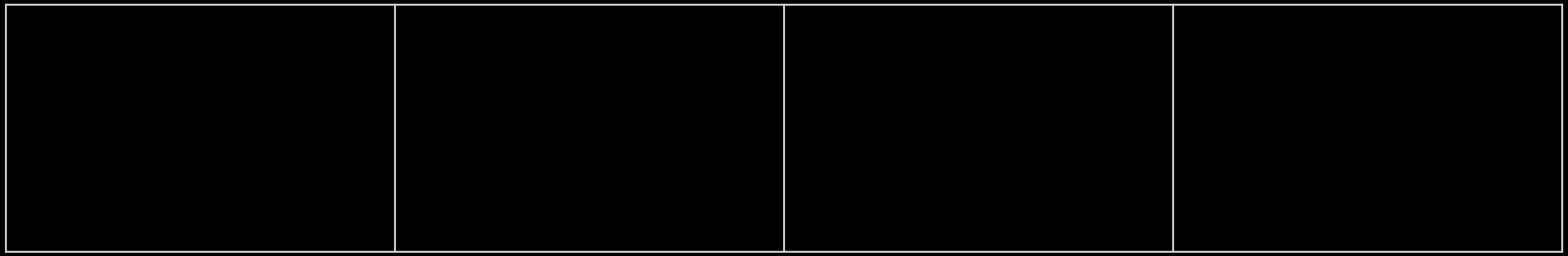
Riak Core
Hash Ring

```
my_hash = {}
```



Riak Core
Hash Ring

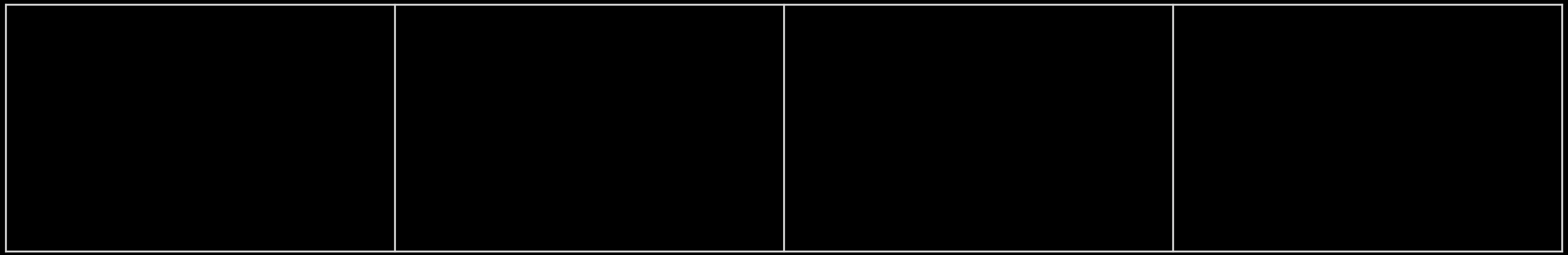
```
my_hash = {}
```



```
my_hash["answer"] = 42
```

Riak Core
Hash Ring

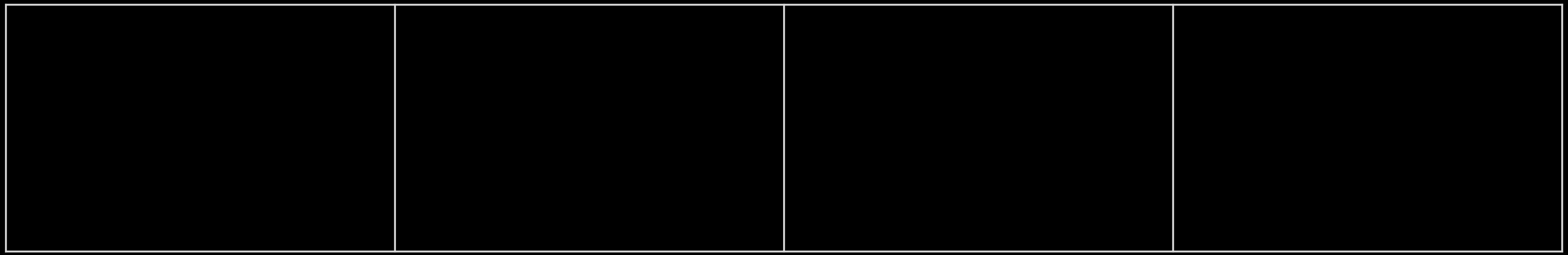
```
my_hash = {}
```



```
my_hash["answer"] = 42
hash("answer") -> 10403
```

Riak Core
Hash Ring

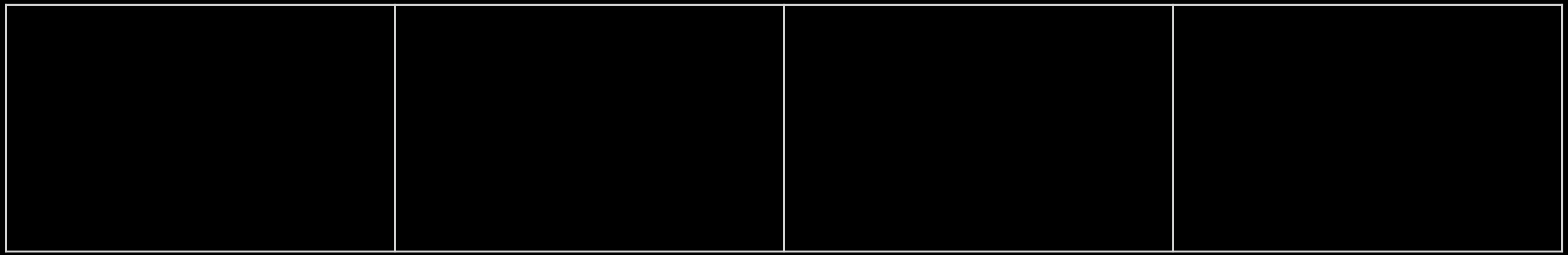
```
my_hash = {}
```



```
my_hash["answer"] = 42
hash("answer") –> 10403
index = 10403 % 4
```

Riak Core
Hash Ring

```
my_hash = {}
```



```
my_hash["answer"] = 42
hash("answer") -> 10403
index = 10403 % 4
# put 42 in index 3!
```

Riak Core
Hash Ring

```
my_hash = {"answer"=>42}
```

			"answer", 42

```
my_hash["answer"] = 42
hash("answer") -> 10403
index = 10403 % 4
# put 42 in index 3!
```

Riak Core
Hash Ring

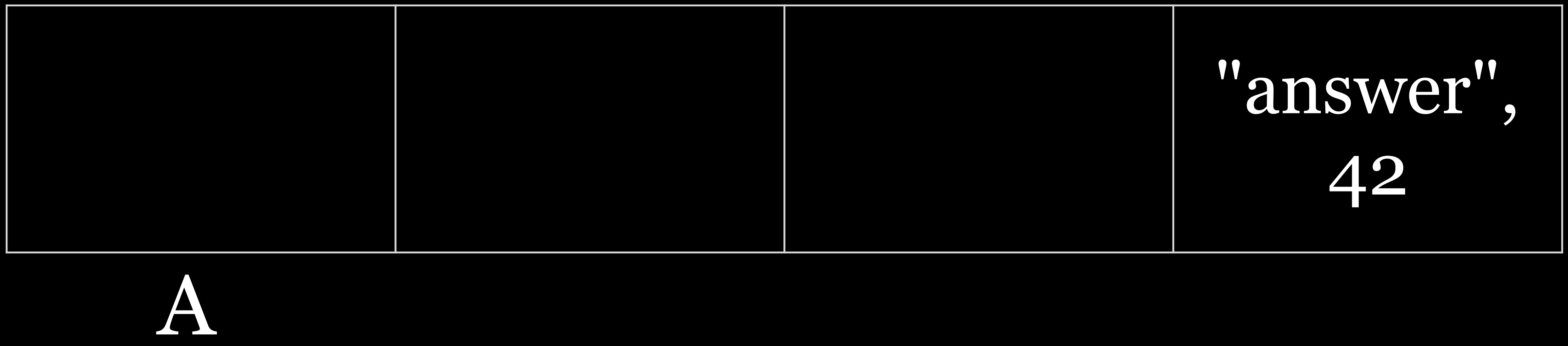
```
my_hash = {"answer"=>42}
```

			"answer", 42

Which server owns which bucket?

Riak Core
Hash Ring

```
my_hash = {"answer"=>42}
```



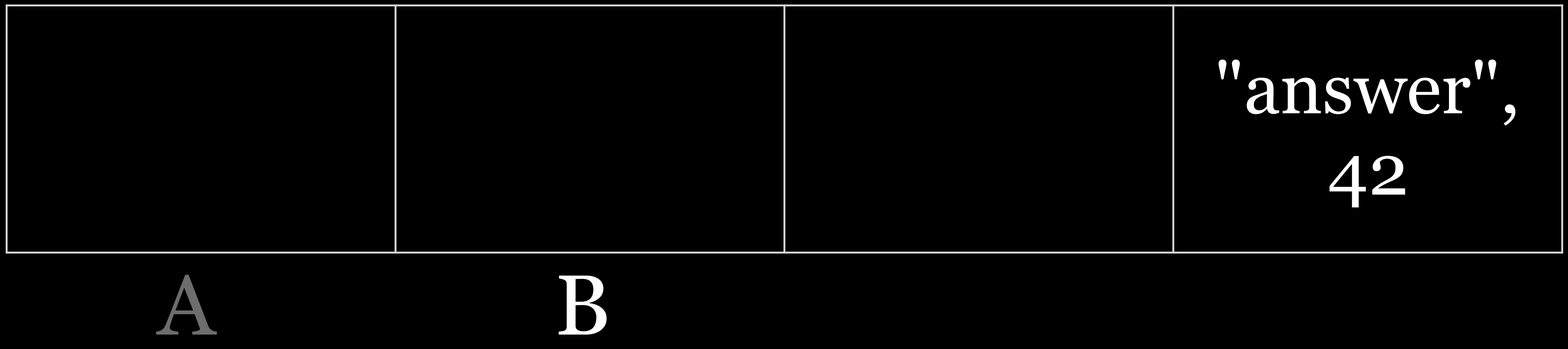
"answer", 42

A

Which server owns which bucket?

Riak Core
Hash Ring

```
my_hash = {"answer"=>42}
```



"answer", 42

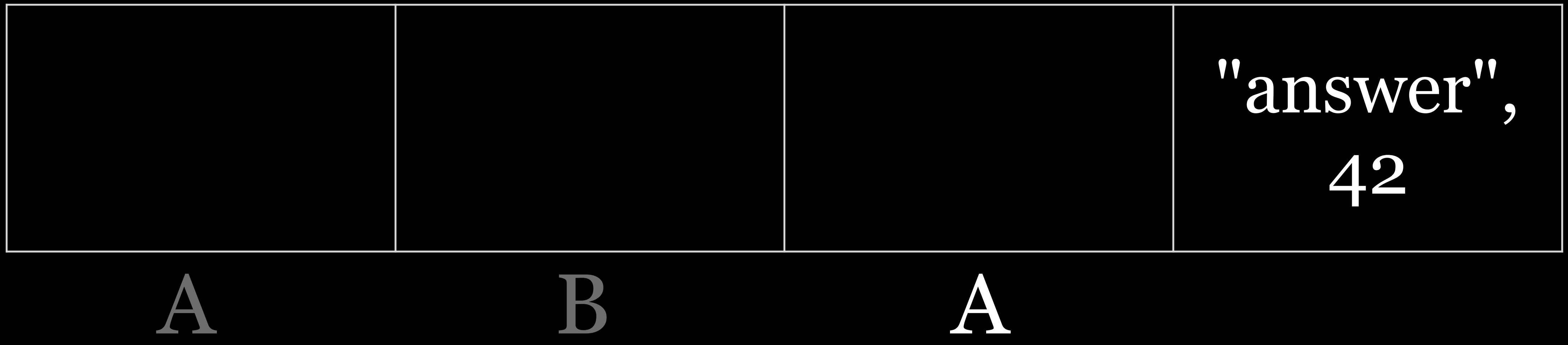
A

B

Which server owns which bucket?

Riak Core
Hash Ring

```
my_hash = {"answer"=>42}
```

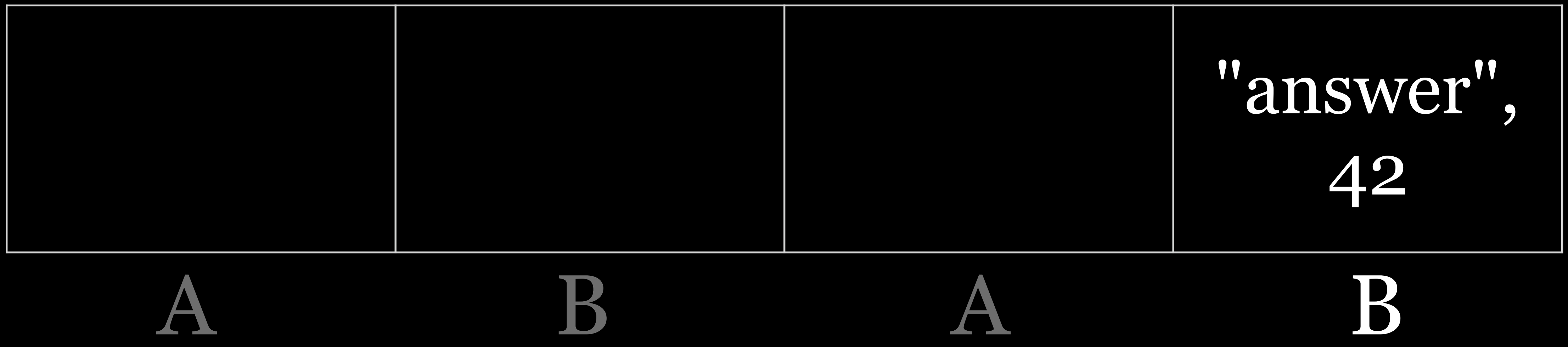


			"answer", 42
A	B	A	

Which server owns which bucket?

Riak Core
Hash Ring

```
my_hash = {"answer"=>42}
```



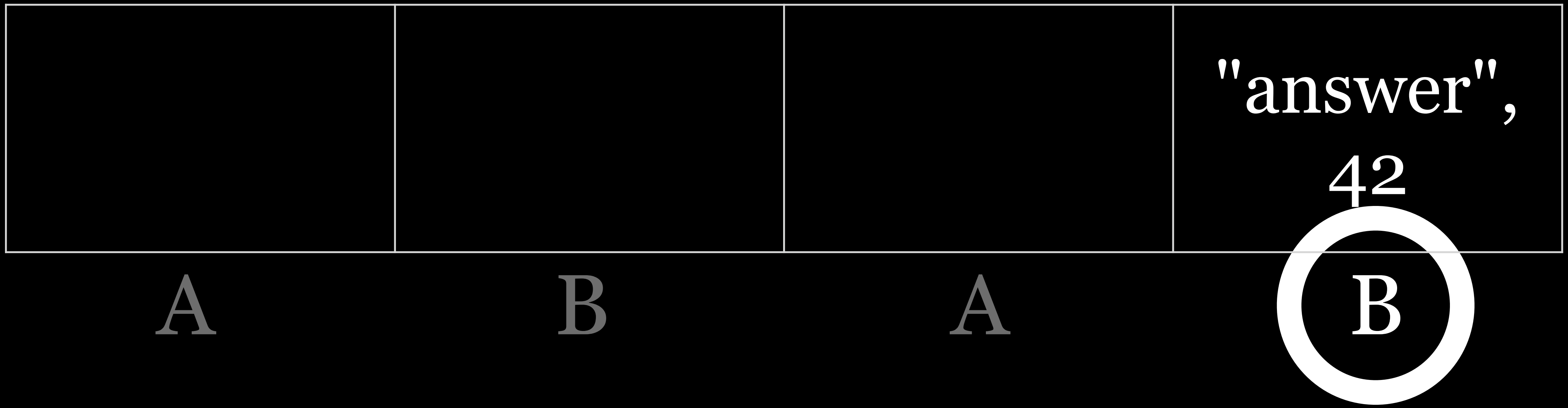
"answer", 42

A B A B

Which server owns which bucket?

Riak Core
Hash Ring

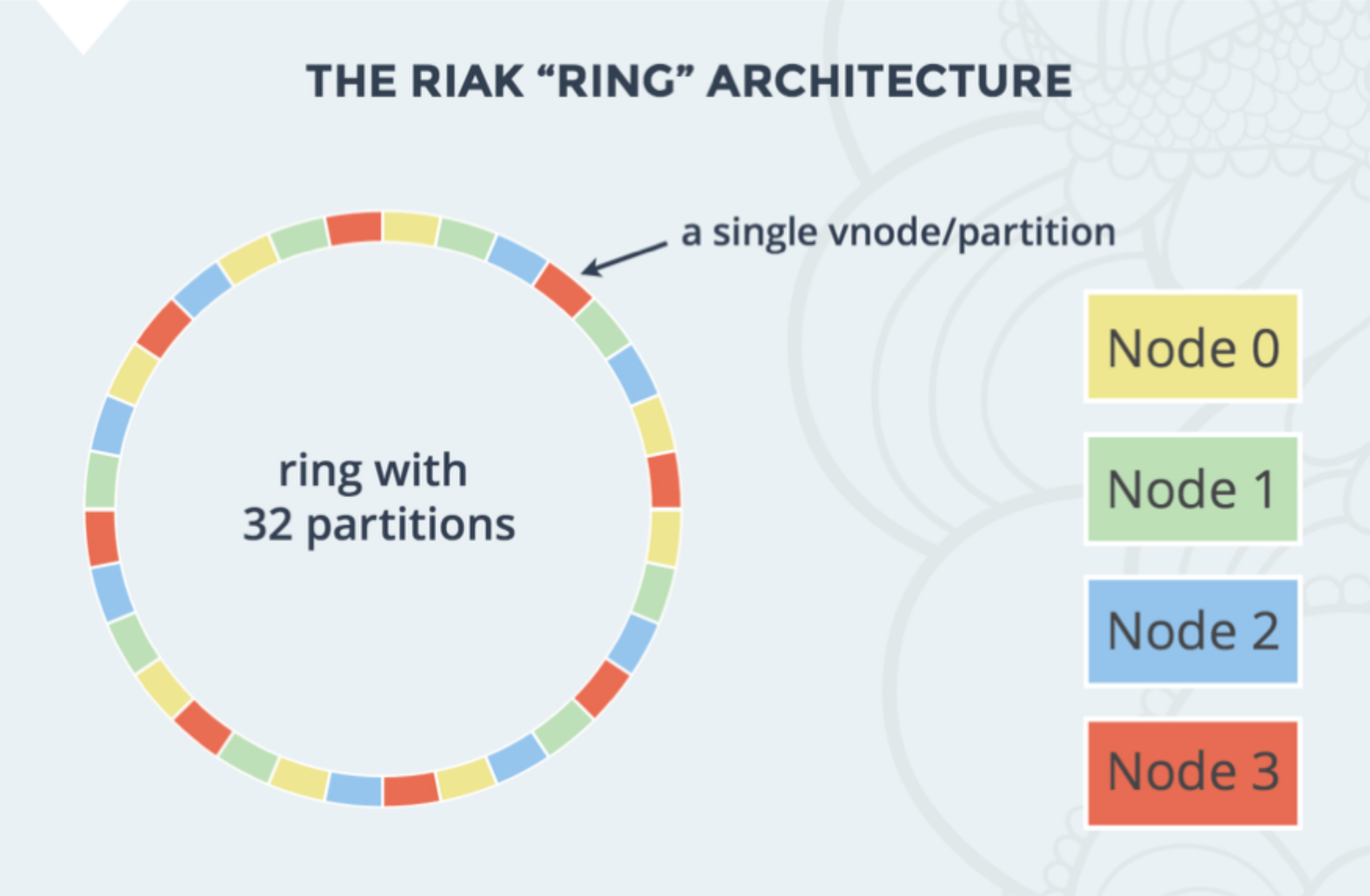
```
my_hash = {"answer"=>42}
```



			"answer", 42
A	B	A	B

Which server owns which bucket?

Riak Core



THE RIAK "RING" ARCHITECTURE

a single vnode/partition

ring with
32 partitions

Node 0

Node 1

Node 2

Node 3

A Riak Core App:
pingring - the app

A Riak Core App:
pingring - the app

ping -> pong

A Riak Core App: pingring - the app

ping -> pong

hash(timestamp) -> vnode (bucket)

A Riak Core App:
pingring - the app

ping -> pong

hash(timestamp) -> vnode (bucket)

example of distributing CPU work

A Riak Core App: pingring - the parts

A Riak Core App: pingring - the parts

service (high level API)

A Riak Core App: pingring - the parts

service (high level API)

vnode (business logic)

A Riak Core App: pingring - the parts

service (high level API)

vnode (business logic)

(supervisor)

A Riak Core App: pingring - the parts

service (high level API)

vnode (business logic)

(supervisor)

(application)

A Riak Core App: pingring - the parts

service (high level API)

vnode (business logic)

(supervisor)

(application)

A Riak Core App:
pingring - service

```
defmodule Pingring.Service do
  def ping do










  end
end
```

A Riak Core App: pingring - service

```
defmodule Pingring.Service do
  def ping do
    doc_idx =                  hash_key(
                                                     )








  end
end
```

A Riak Core App: pingring - service

```
defmodule Pingring.Service do
  def ping do
    doc_idx =                  hash_key(
                                      :os.timestamp  )








  end
end
```

A Riak Core App: pingring - service

```
defmodule Pingring.Service do
  def ping do
    doc_idx = :riak_core_util.chash_key(
                                      :os.timestamp  )









  end
end
```

A Riak Core App: pingring - service

```
defmodule Pingring.Service do
  def ping do
    doc_idx = :riak_core_util.chash_key(
      {"ping", :erlang.term_to_binary(:os.timestamp)})









  end
end
```

A Riak Core App: pingring - service

```
defmodule Pingring.Service do
  def ping do
    doc_idx = :riak_core_util.chash_key(
      {"ping", :erlang.term_to_binary(:os.timestamp)})

    pref_list =







  end
end
```

A Riak Core App: pingring - service

```
defmodule Pingring.Service do
  def ping do
    doc_idx = :riak_core_util.chash_key(
      {"ping", :erlang.term_to_binary(:os.timestamp)})

    pref_list =                get_primary_apl(
      doc_idx,                    )





  end
end
```

A Riak Core App: pingring - service

```
defmodule Pingring.Service do
  def ping do
    doc_idx = :riak_core_util.chash_key(
      {"ping", :erlang.term_to_binary(:os.timestamp)})

    pref_list =                get_primary_apl(
      doc_idx,    Pingring.Service)





  end
end
```

A Riak Core App: pingring - service

```
defmodule Pingring.Service do
  def ping do
    doc_idx = :riak_core_util.chash_key(
      {"ping", :erlang.term_to_binary(:os.timestamp)})

    pref_list =                get_primary_apl(
      doc_idx, 1, Pingring.Service)





  end
end
```

A Riak Core App:
pingring - service

```
defmodule Pingring.Service do
  def ping do
    doc_idx = :riak_core_util.chash_key(
      {"ping", :erlang.term_to_binary(:os.timestamp)})

    pref_list = :riak_core_apl.get_primary_apl(
      doc_idx, 1, Pingring.Service)





  end
end
```

A Riak Core App:
pingring - service

```
defmodule Pingring.Service do
  def ping do
    doc_idx = :riak_core_util.chash_key(
      {"ping", :erlang.term_to_binary(:os.timestamp)})

    pref_list = :riak_core_apl.get_primary_apl(
      doc_idx, 1, Pingring.Service)

    [{index_node, _type}] = pref_list



  end
end
```

A Riak Core App: pingring - service

```
defmodule Pingring.Service do
  def ping do
    doc_idx = :riak_core_util.chash_key(
      {"ping", :erlang.term_to_binary(:os.timestamp)})

    pref_list = :riak_core_apl.get_primary_apl(
      doc_idx, 1, Pingring.Service)

    [{index_node, _type}] = pref_list

                            sync_spawn_command(
      index_node,                             )
  end
end
```

A Riak Core App: pingring - service

```
defmodule Pingring.Service do
  def ping do
    doc_idx = :riak_core_util.chash_key(
      {"ping", :erlang.term_to_binary(:os.timestamp)})

    pref_list = :riak_core_apl.get_primary_apl(
      doc_idx, 1, Pingring.Service)

    [{index_node, _type}] = pref_list

                            sync_spawn_command(
      index_node, :ping,                      )
  end
end
```

A Riak Core App: pingring - service

```
defmodule Pingring.Service do
  def ping do
    doc_idx = :riak_core_util.chash_key(
      {"ping", :erlang.term_to_binary(:os.timestamp)})

    pref_list = :riak_core_apl.get_primary_apl(
      doc_idx, 1, Pingring.Service)

    [{index_node, _type}] = pref_list

                            sync_spawn_command(
      index_node, :ping, Pingring.Vnode_master)
  end
end
```

A Riak Core App:
pingring - service

```
defmodule Pingring.Service do
  def ping do
    doc_idx = :riak_core_util.chash_key(
      {"ping", :erlang.term_to_binary(:os.timestamp)})

    pref_list = :riak_core_apl.get_primary_apl(
      doc_idx, 1, Pingring.Service)

    [{index_node, _type}] = pref_list
    # riak core appends "_master" to Pingring.Vnode.
                            sync_spawn_command(
      index_node, :ping, Pingring.Vnode_master)
  end
end
```

A Riak Core App: pingring - service

```
defmodule Pingring.Service do
  def ping do
    doc_idx = :riak_core_util.chash_key(
      {"ping", :erlang.term_to_binary(:os.timestamp)})

    pref_list = :riak_core_apl.get_primary_apl(
      doc_idx, 1, Pingring.Service)

    [{index_node, _type}] = pref_list
    # riak core appends "_master" to Pingring.Vnode.
    :riak_core_vnode_master.sync_spawn_command(
      index_node, :ping, Pingring.Vnode_master)
  end
end
```

A Riak Core App: pingring - service

```
defmodule Pingring.Service do
  def ping do
    doc_idx = :riak_core_util.chash_key(
      {"ping", :erlang.term_to_binary(:os.timestamp)})

    pref_list = :riak_core_apl.get_primary_apl(
      doc_idx, 1, Pingring.Service)

    [{index_node, _type}] = pref_list
    # riak core appends "_master" to Pingring.Vnode.
    :riak_core_vnode_master.sync_spawn_command(
      index_node, :ping, Pingring.Vnode_master)
  end
end
```

A Riak Core App: pingring - the parts

service (high level API)

vnode (business logic)

(supervisor)

(application)

A Riak Core App:
pingring - vnode

```
defmodule Pingring.Vnode do









end
```

A Riak Core App: pingring - vnode

```
defmodule Pingring.Vnode do
  @behaviour :riak_core_vnode

end
```

A Riak Core App: pingring - vnode

```
defmodule Pingring.Vnode do
  @behaviour :riak_core_vnode
  # … some boilerplate for startup









end
```

A Riak Core App: pingring - vnode

```
defmodule Pingring.Vnode do
  @behaviour :riak_core_vnode
  # … some boilerplate for startup
  def init(      ), do: {:ok, %{          }}








end
```

A Riak Core App: pingring - vnode

```
defmodule Pingring.Vnode do
  @behaviour :riak_core_vnode
  # … some boilerplate for startup
  def init([part]), do: {:ok, %{part: part}}







end
```

A Riak Core App:
pingring - vnode

```
defmodule Pingring.Vnode do
  @behaviour :riak_core_vnode
  # … some boilerplate for startup
  def init([part]), do: {:ok, %{part: part}}

  def handle_command(

  ) do

  end

end
```

A Riak Core App: pingring - vnode

```
defmodule Pingring.Vnode do
  @behaviour :riak_core_vnode
  # … some boilerplate for startup
  def init([part]), do: {:ok, %{part: part}}

  def handle_command(
    :ping, _sender, %{part: part} = state
  ) do

  end

end
```

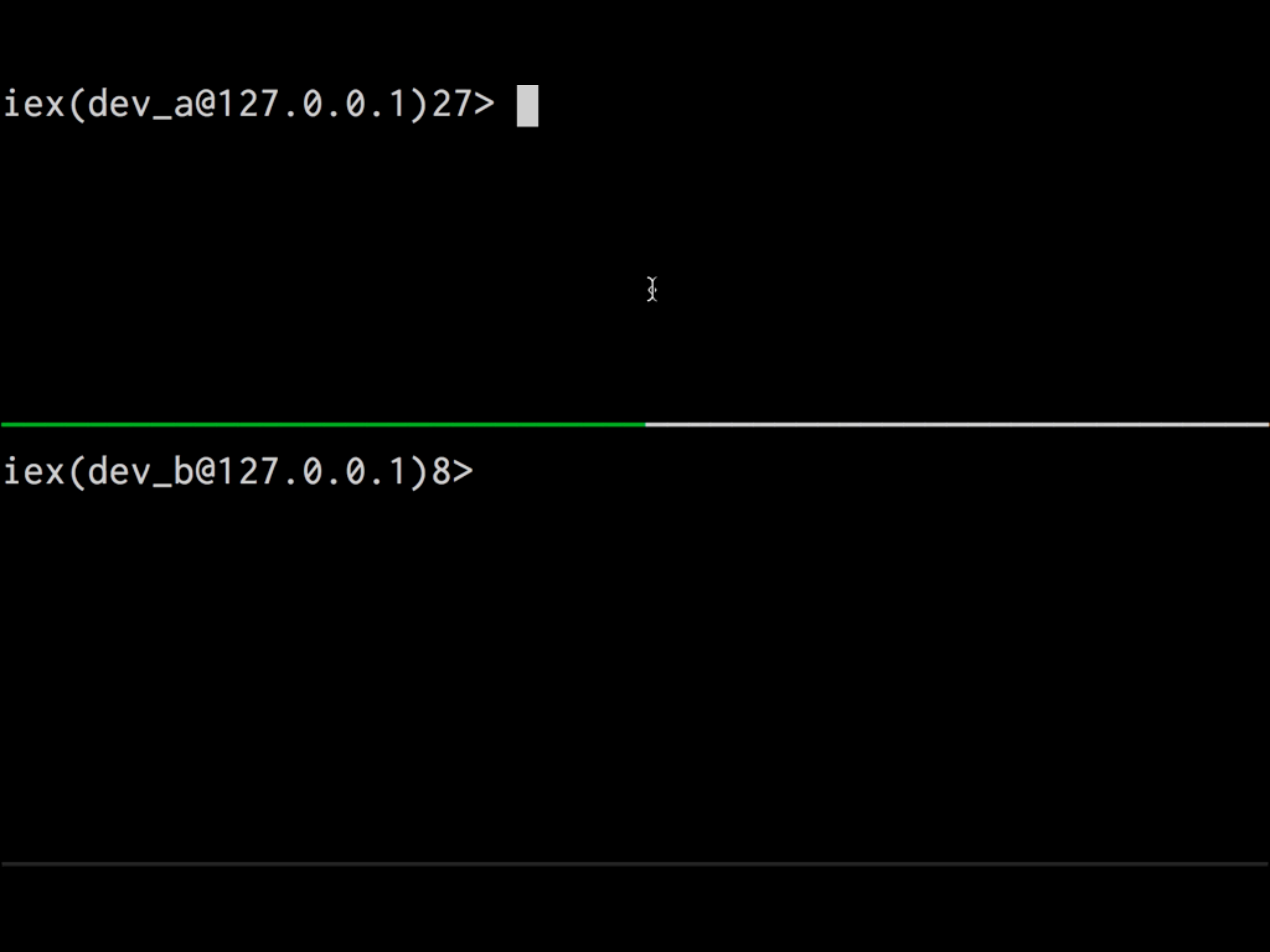
A Riak Core App:
pingring - vnode

```
defmodule Pingring.Vnode do
  @behaviour :riak_core_vnode
  # … some boilerplate for startup
  def init([part]), do: {:ok, %{part: part}}

  def handle_command(
    :ping, _sender, %{part: part} = state
  ) do
    {:reply, {:pong, part}, state}
  end

end
```

A Riak Core App: pingring - vnode

```
defmodule Pingring.Vnode do
  @behaviour :riak_core_vnode
  # … some boilerplate for startup
  def init([part]), do: {:ok, %{part: part}}

  def handle_command(
    :ping, _sender, %{part: part} = state
  ) do
    {:reply, {:pong, part}, state}
  end
  # … other callbacks for :riak_core_vnode
end
```

Demo



```
iex(dev_a@127.0.0.1)27>
```

```
iex(dev_b@127.0.0.1)8>
```

What about…state?

A Riak Core App:
store_fetch - the app

A Riak Core App: store_fetch - the app

store(key, data)

A Riak Core App: store_fetch - the app

store(key, data)

store in ETS

A Riak Core App: store_fetch - the app

store(key, data)

store in ETS

hash(key) -> vnode

A Riak Core App:
store_fetch - service

```
def store(key, data) do








end
```

A Riak Core App: store_fetch - service

```
def store(key, data) do
  doc_idx =                     hash_key(
                                      )









end
```

A Riak Core App: store_fetch - service

```
def store(key, data) do
  doc_idx =                        hash_key(
                                           key  )









end
```

A Riak Core App: store_fetch - service

```
def store(key, data) do
  doc_idx = :riak_core_util.chash_key(
                                   key  )










end
```

A Riak Core App: store_fetch - service

```
def store(key, data) do
  doc_idx = :riak_core_util.chash_key(
    {"store", :erlang.term_to_binary(key)})









end
```

A Riak Core App: store_fetch - service

```
def store(key, data) do
  doc_idx = :riak_core_util.chash_key(
    {"store", :erlang.term_to_binary(key)})

  pref_list =                get_primary_apl(
    doc_idx,                      )





end
```

A Riak Core App:
store_fetch - service

```
def store(key, data) do
  doc_idx = :riak_core_util.chash_key(
    {"store", :erlang.term_to_binary(key)})

  pref_list =                get_primary_apl(
    doc_idx,    StoreFetch.Service)





end
```

A Riak Core App:
store_fetch - service

```
def store(key, data) do
  doc_idx = :riak_core_util.chash_key(
    {"store", :erlang.term_to_binary(key)})

  pref_list =                get_primary_apl(
    doc_idx, 1, StoreFetch.Service)






end
```

A Riak Core App:
store_fetch - service

```
def store(key, data) do
  doc_idx = :riak_core_util.chash_key(
    {"store", :erlang.term_to_binary(key)})

  pref_list = :riak_core_apl.get_primary_apl(
    doc_idx, 1, StoreFetch.Service)







end
```

A Riak Core App: store_fetch - service

```
def store(key, data) do
  doc_idx = :riak_core_util.chash_key(
    {"store", :erlang.term_to_binary(key)})

  pref_list = :riak_core_apl.get_primary_apl(
    doc_idx, 1, StoreFetch.Service)

  [{index_node, _type}] = pref_list




end
```

A Riak Core App: store_fetch - service

```
def store(key, data) do
  doc_idx = :riak_core_util.chash_key(
    {"store", :erlang.term_to_binary(key)})

  pref_list = :riak_core_apl.get_primary_apl(
    doc_idx, 1, StoreFetch.Service)

  [{index_node, _type}] = pref_list
                          sync_spawn_command(
    index_node,
                           )
end
```

A Riak Core App: store_fetch - service

```
def store(key, data) do
  doc_idx = :riak_core_util.chash_key(
    {"store", :erlang.term_to_binary(key)})

  pref_list = :riak_core_apl.get_primary_apl(
    doc_idx, 1, StoreFetch.Service)

  [{index_node, _type}] = pref_list
                          sync_spawn_command(
    index_node, {:store, key, data},
                           )
end
```

A Riak Core App: store_fetch - service

```
def store(key, data) do
  doc_idx = :riak_core_util.chash_key(
    {"store", :erlang.term_to_binary(key)})

  pref_list = :riak_core_apl.get_primary_apl(
    doc_idx, 1, StoreFetch.Service)

  [{index_node, _type}] = pref_list
                          sync_spawn_command(
    index_node, {:store, key, data},
    StoreFetch.Vnode_master)
end
```

A Riak Core App: store_fetch - service

```
def store(key, data) do
  doc_idx = :riak_core_util.chash_key(
    {"store", :erlang.term_to_binary(key)})

  pref_list = :riak_core_apl.get_primary_apl(
    doc_idx, 1, StoreFetch.Service)

  [{index_node, _type}] = pref_list
  :riak_core_vnode_master.sync_spawn_command(
    index_node, {:store, key, data},
    StoreFetch.Vnode_master)
end
```

A Riak Core App: store_fetch - vnode

```
defmodule StoreFetch.Vnode do

end
```

A Riak Core App: store_fetch - vnode

```
defmodule StoreFetch.Vnode do
  @behaviour :riak_core_vnode
  # … some boilerplate for startup















end
```

A Riak Core App:
store_fetch - vnode

```
defmodule StoreFetch.Vnode do
  @behaviour :riak_core_vnode
  # … some boilerplate for startup
  def init([_part]) do


  end







end
```

A Riak Core App: store_fetch - vnode

```
defmodule StoreFetch.Vnode do
  @behaviour :riak_core_vnode
  # … some boilerplate for startup
  def init([_part]) do
    ets_handle = :ets.new(nil, [])
    {:ok, %{db: ets_handle}}
  end







end
```

A Riak Core App:
store_fetch - vnode

```
defmodule StoreFetch.Vnode do
  @behaviour :riak_core_vnode
  # … some boilerplate for startup
  def init([_part]) do
    ets_handle = :ets.new(nil, [])
    {:ok, %{db: ets_handle}}
  end
  def handle_command(

  ) do


  end

end
```

A Riak Core App: store_fetch - vnode

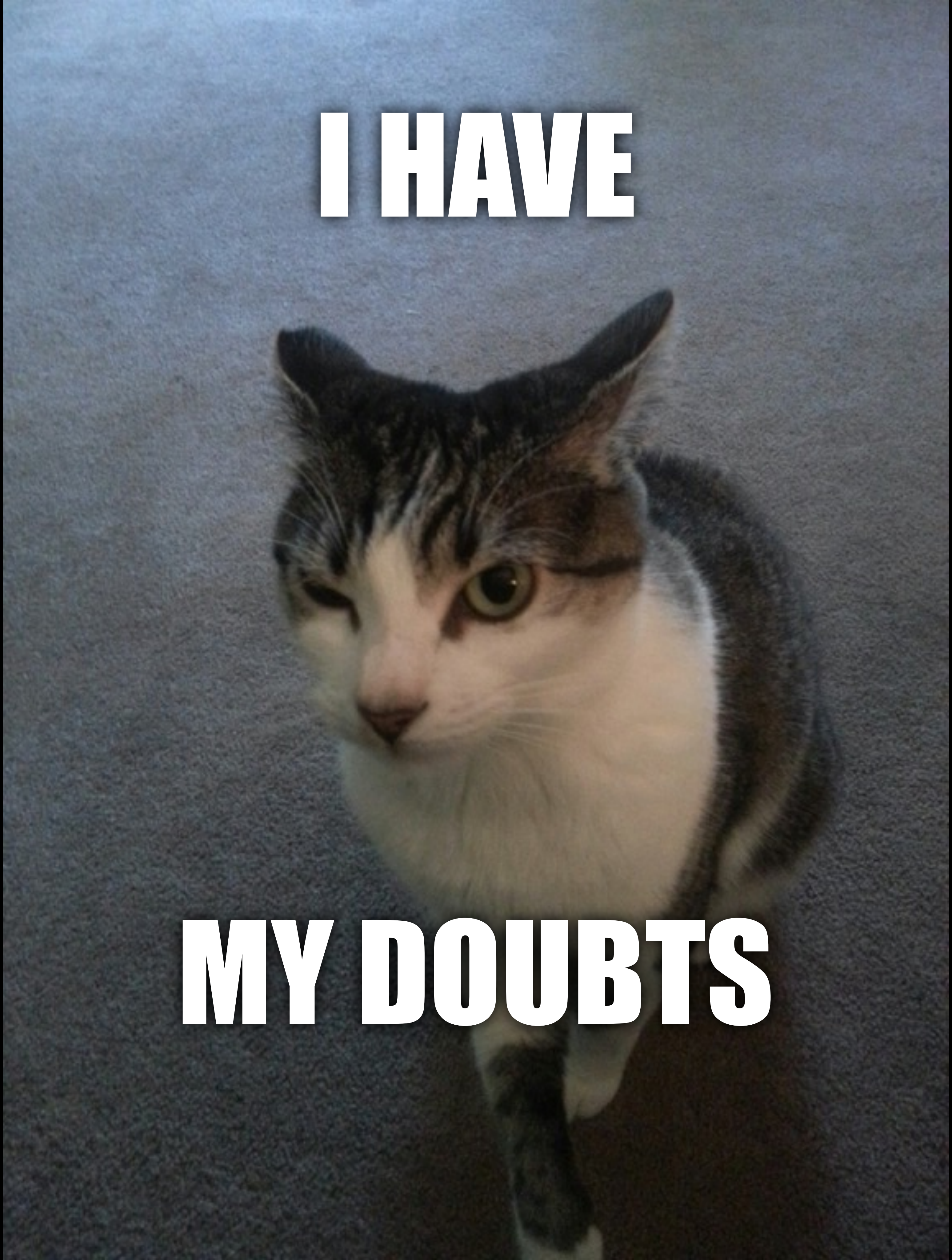
```
defmodule StoreFetch.Vnode do
  @behaviour :riak_core_vnode
  # … some boilerplate for startup
  def init([_part]) do
    ets_handle = :ets.new(nil, [])
    {:ok, %{db: ets_handle}}
  end
  def handle_command(
    {:store, key, data}, _sender, %{db: db} = state
  ) do


  end

end
```

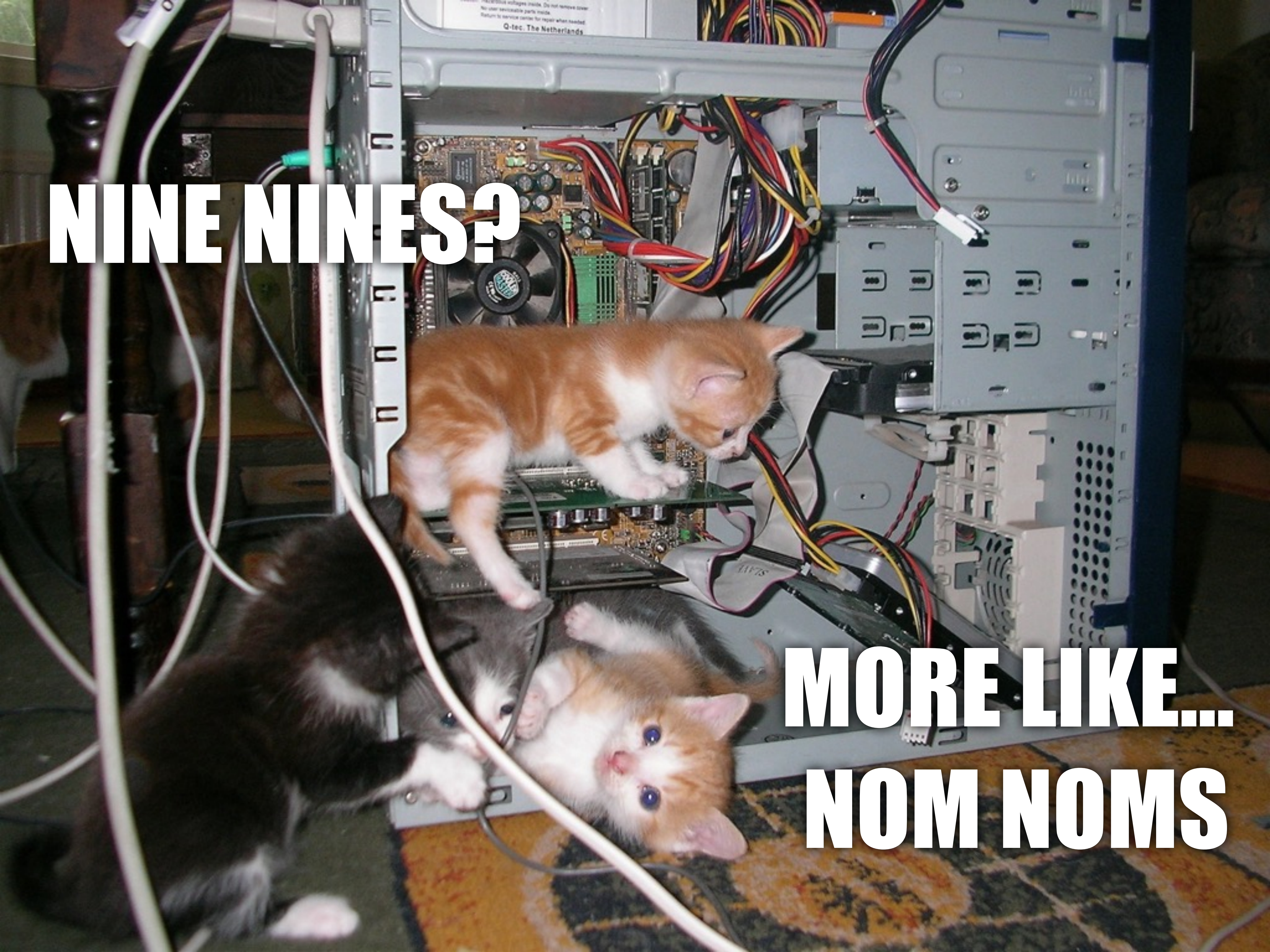
A Riak Core App: store_fetch - vnode

```
defmodule StoreFetch.Vnode do
  @behaviour :riak_core_vnode
  # … some boilerplate for startup
  def init([_part]) do
    ets_handle = :ets.new(nil, [])
    {:ok, %{db: ets_handle}}
  end
  def handle_command(
    {:store, key, data}, _sender, %{db: db} = state
  ) do
    result = :ets.insert(db, {key, data})
    {:reply, result, state}
  end

end
```

A Riak Core App: store_fetch - vnode

```
defmodule StoreFetch.Vnode do
  @behaviour :riak_core_vnode
  # … some boilerplate for startup
  def init([_part]) do
    ets_handle = :ets.new(nil, [])
    {:ok, %{db: ets_handle}}
  end
  def handle_command(
    {:store, key, data}, _sender, %{db: db} = state
  ) do
    result = :ets.insert(db, {key, data})
    {:reply, result, state}
  end
  # … same for fetch, but :ets.lookup instead
end
```


I HAVE
MY DOUBTS



NINE NINES?

MORE LIKE...
NOM NOMS

Buzzword Bingo

Stateful - ✓

Distributed

Fault-tolerant

Real-time

Impress your cat

(application)

Buzzword Bingo

Stateful - ✓

Distributed - ✓

Fault-tolerant

Real-time

Impress your cat

(application)

Fault Tolerance

Computers needed for fault tolerance?

Fault Tolerance

Computers needed for fault tolerance?

> 1

A Riak Core App: store_fetch - the parts

service

vnode

(supervisor)

(application)

A Riak Core App: store_fetch - the parts

service

vnode

(supervisor)

(application)

write coordinator (plus supervisor)

The Write Coordinator

The Write Coordinator

executes commands on multiple vnodes

The Write Coordinator

executes commands on multiple vnodes

riak_core takes care of spreading vnodes across servers

The Write Coordinator

executes commands on multiple vnodes

riak_core takes care of spreading vnodes across servers

(as much as possible)

Using the Write Coordinator

```
defmodule StoreFetch.Service do



end
```

Using the Write Coordinator

```
defmodule StoreFetch.Service do
  def store(key, data      ) do










  end

end
```

Using the Write Coordinator

```
defmodule StoreFetch.Service do
  def store(key, data, n, w) do










  end

end
```

Using the Write Coordinator

```
defmodule StoreFetch.Service do
  def store(key, data, n, w) do
    {:ok, req_id} = StoreFetch.WCoord.do(
                                    )








  end

end
```

Using the Write Coordinator

```
defmodule StoreFetch.Service do
  def store(key, data, n, w) do
    {:ok, req_id} = StoreFetch.WCoord.do(
      key, {:store, key, data}, n, w)









  end

end
```

Using the Write Coordinator

```
defmodule StoreFetch.Service do
  def store(key, data, n, w) do
    {:ok, req_id} = StoreFetch.WCoord.do(
      key, {:store, key, data}, n, w)

    receive do


    after


    end
  end

end
```

Using the Write Coordinator

```
defmodule StoreFetch.Service do
  def store(key, data, n, w) do
    {:ok, req_id} = StoreFetch.WCoord.do(
      key, {:store, key, data}, n, w)

    receive do


    after
      5000 ->
        {:error, :timeout}
    end
  end

end
```

Using the Write Coordinator

```
defmodule StoreFetch.Service do
  def store(key, data, n, w) do
    {:ok, req_id} = StoreFetch.WCoord.do(
      key, {:store, key, data}, n, w)

    receive do
      {^req_id, value} ->
        {:ok, value}
    after
      5000 ->
        {:error, :timeout}
    end
  end

end
```

Using the Write Coordinator

```
defmodule StoreFetch.Service do
  def store(key, data, n, w) do
    {:ok, req_id} = StoreFetch.WCoord.do(
      key, {:store, key, data}, n, w)

    receive do
      {^req_id, value} ->
        {:ok, value}
    after
      5000 ->
        {:error, :timeout}
    end
  end
  # … fetch implementation
end
```

The Write Coordinator

The Write Coordinator

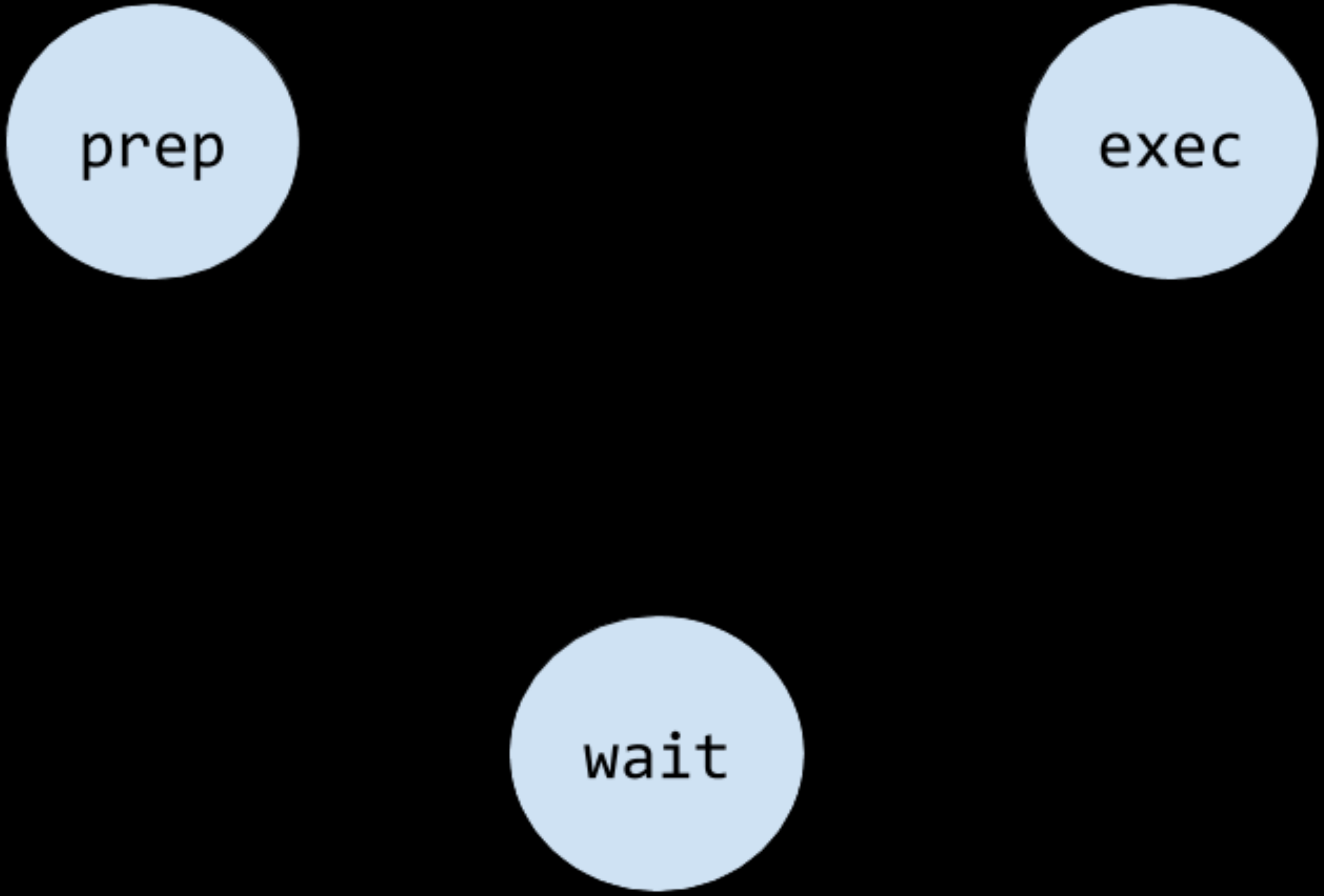
:gen_fsm

The Write Coordinator

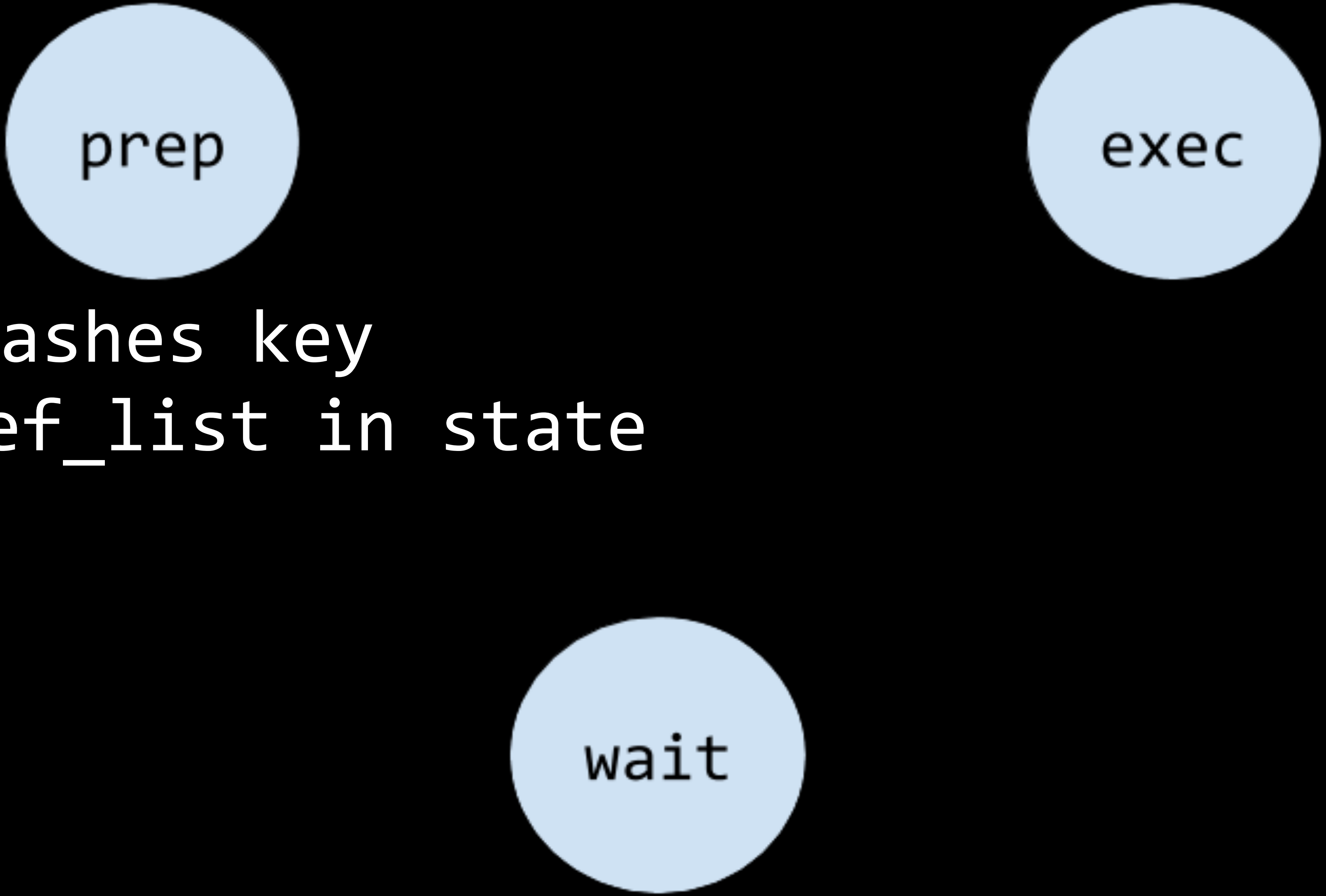
:gen_fsm

spawned on demand (:simple_one_for_one)

The Write Coordinator



The Write Coordinator



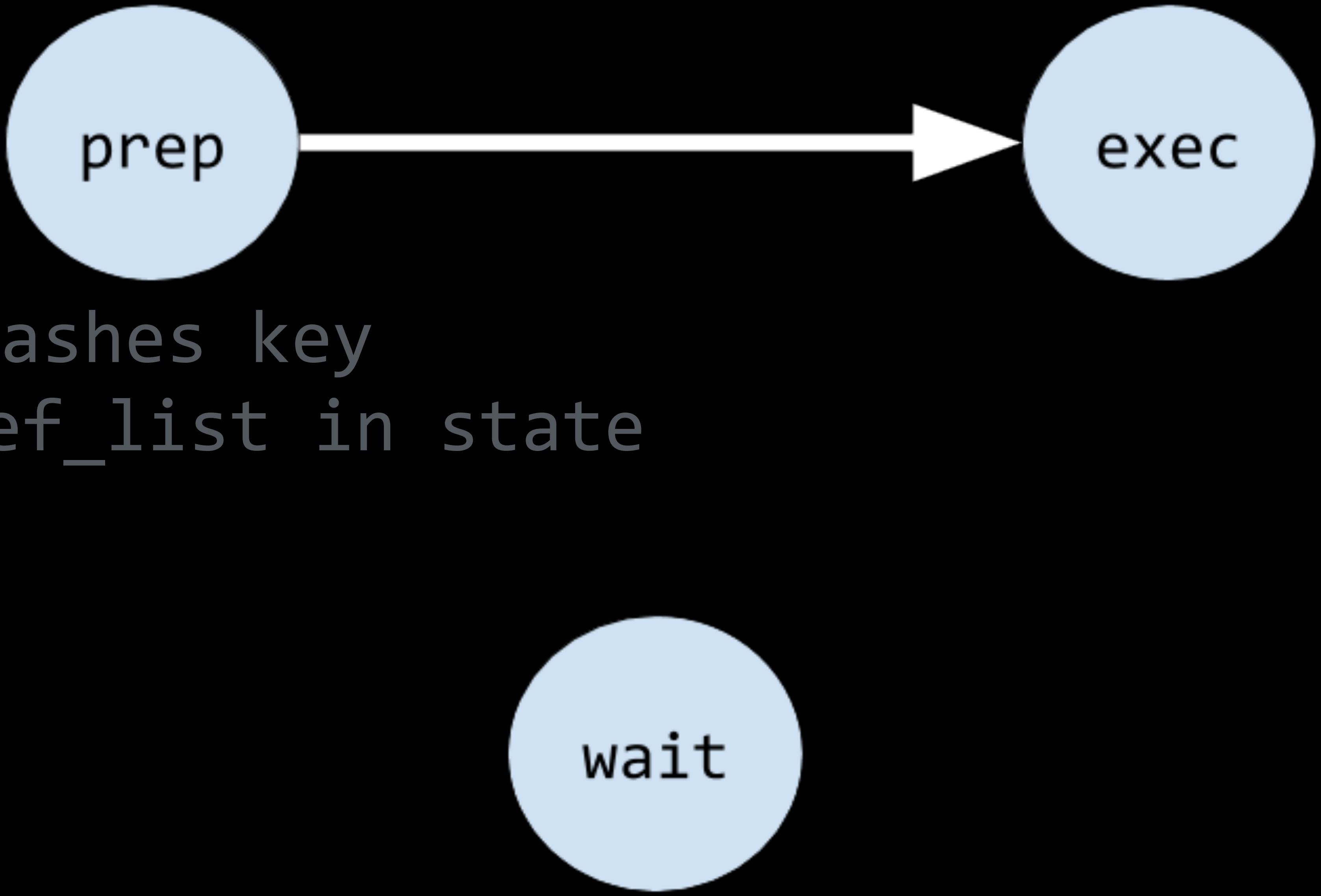
prep

exec

wait

hashes key
puts pref_list in state

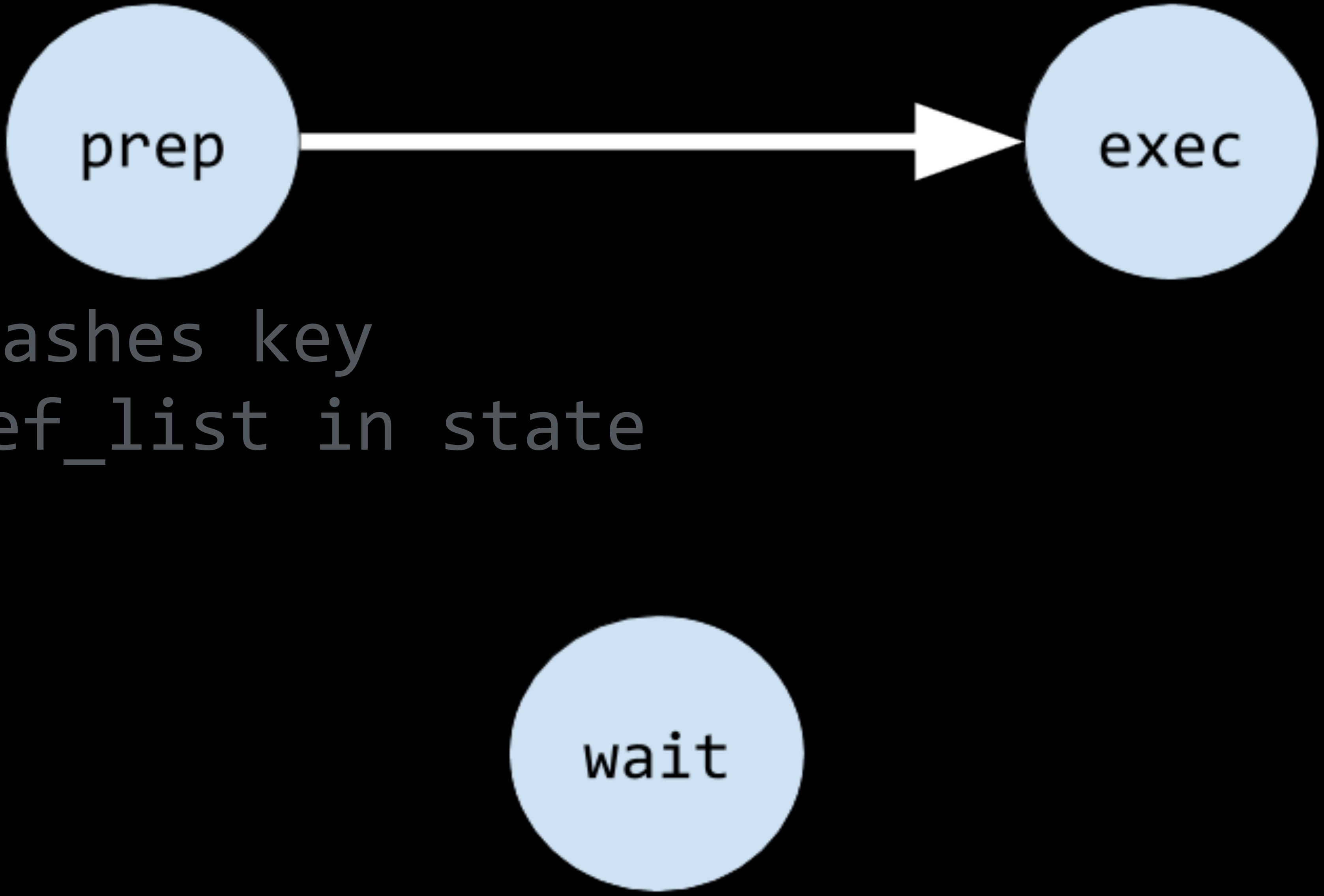
The Write Coordinator



hashes key
puts pref_list in state

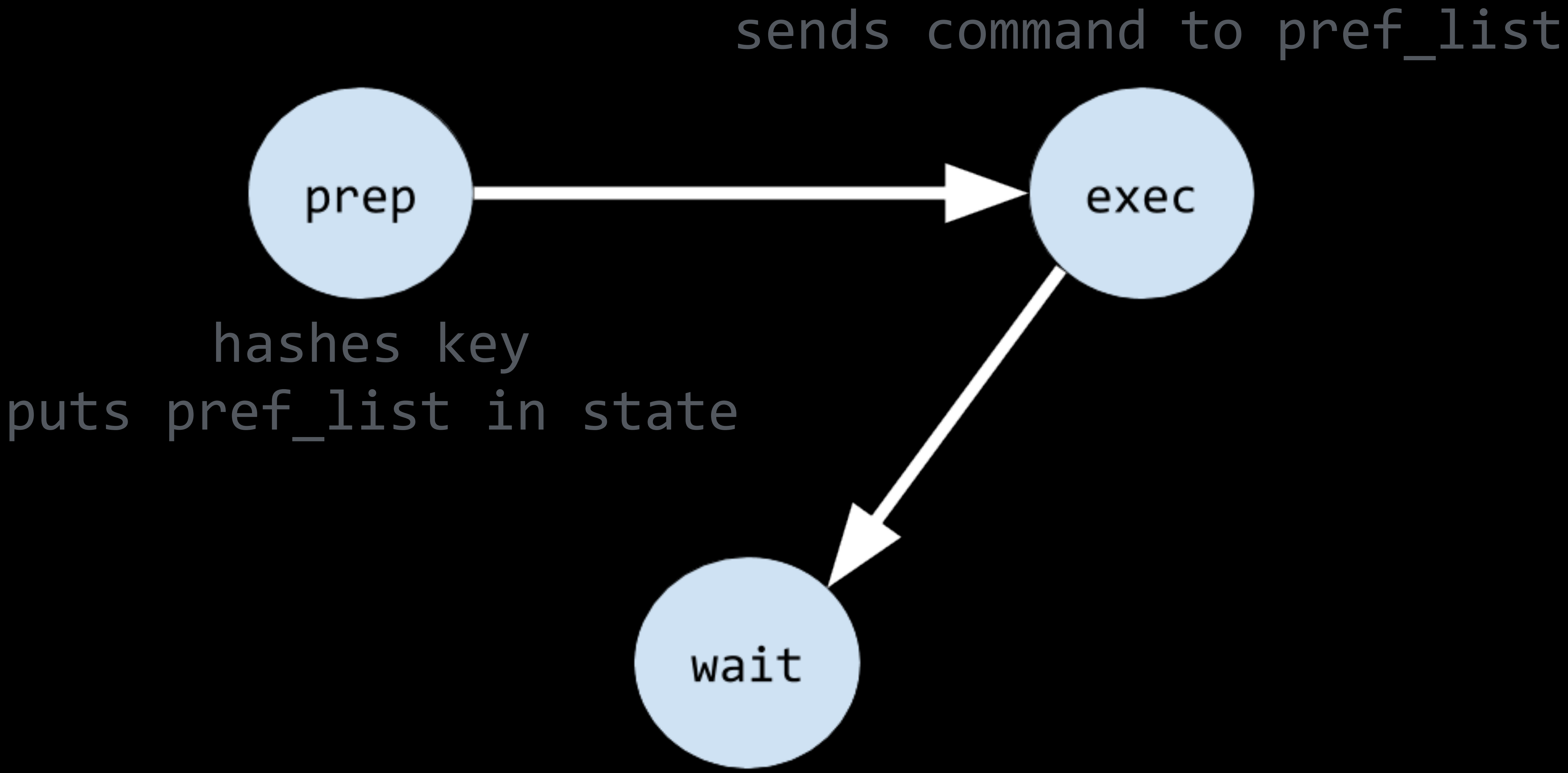
The Write Coordinator

sends command to pref_list



hashes key
puts pref_list in state

The Write Coordinator

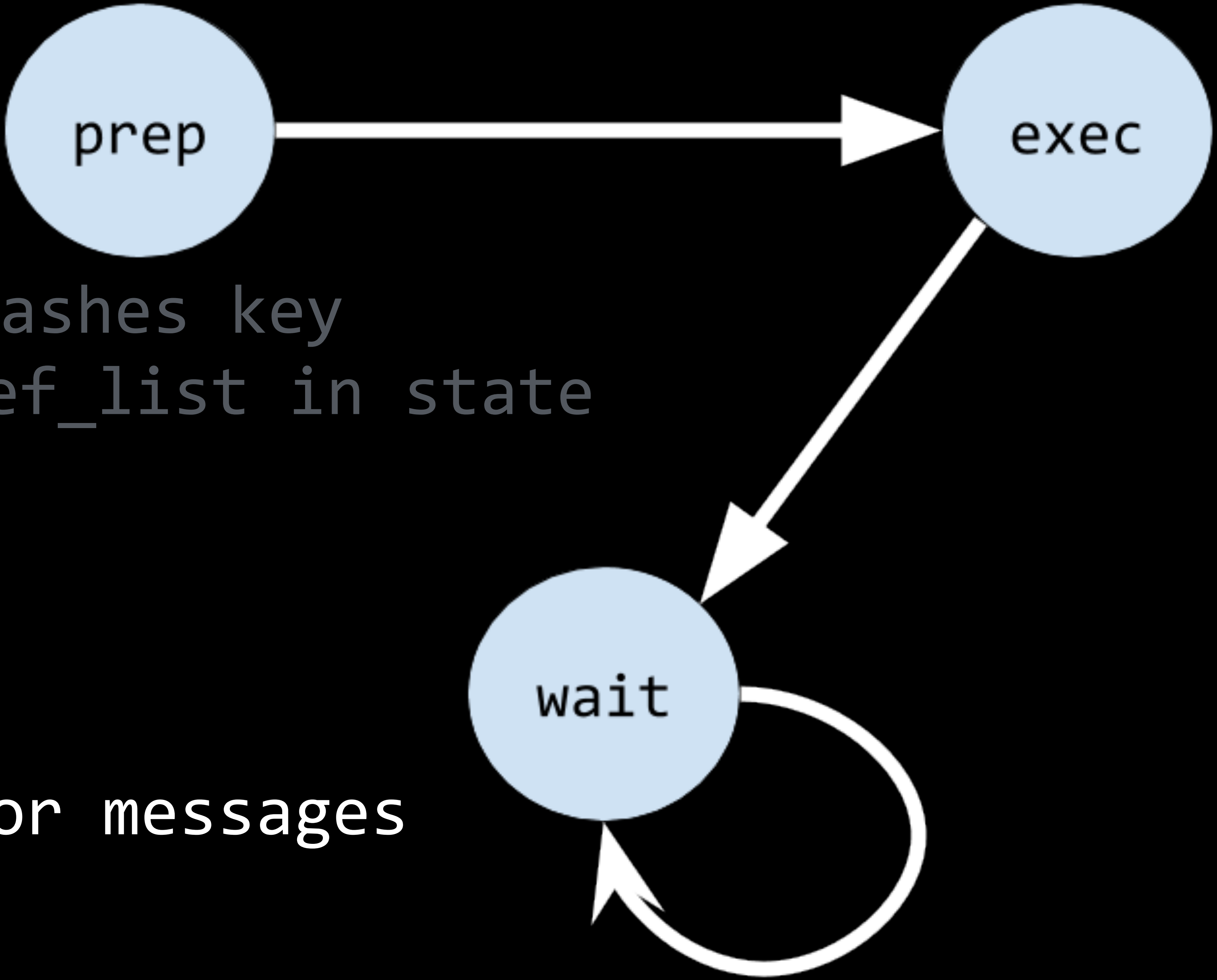


sends command to pref_list

hashes key
puts pref_list in state

The Write Coordinator

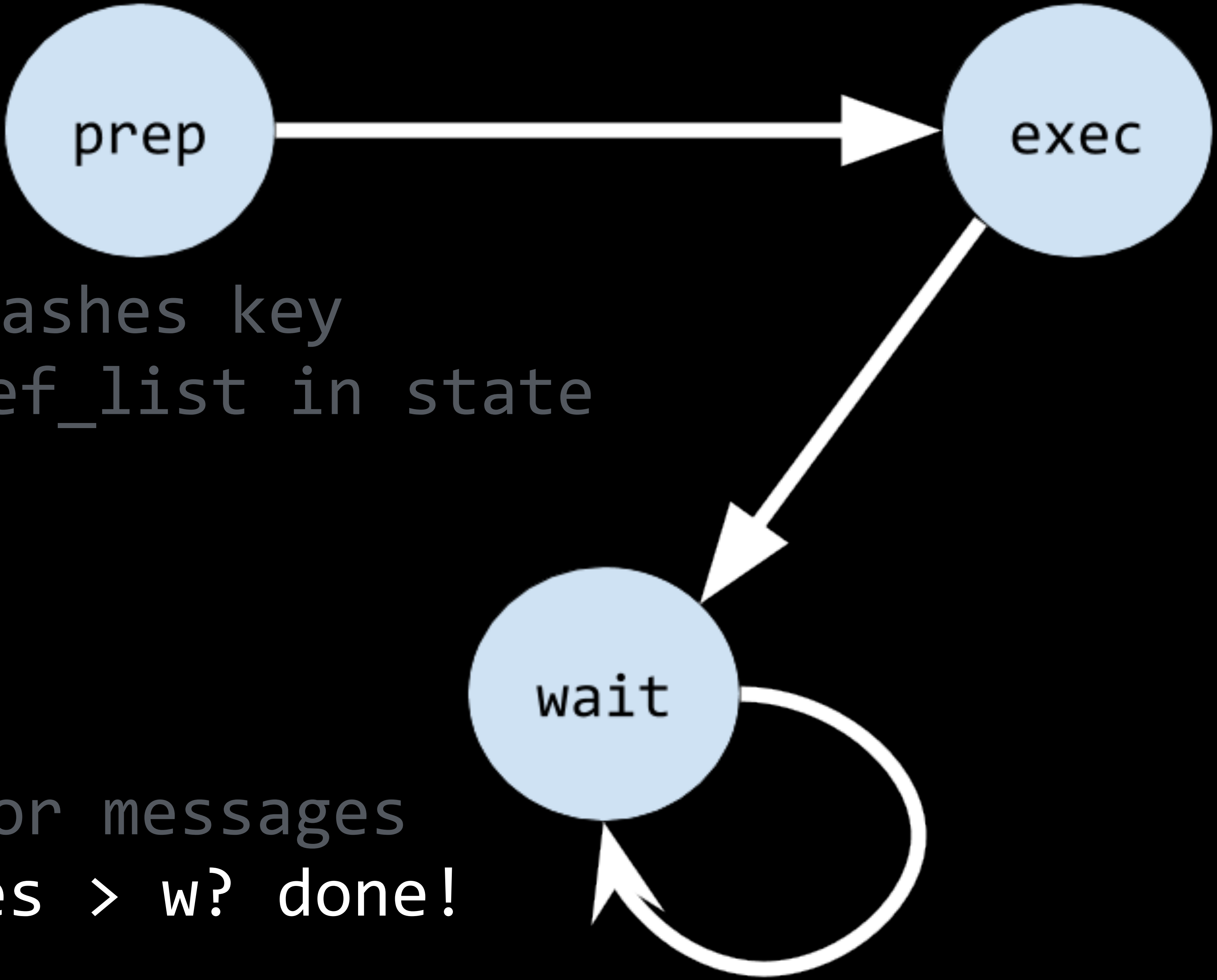
sends command to pref_list



hashes key
puts pref_list in state

waits for messages

The Write Coordinator



sends command to pref_list

hashes key
puts pref_list in state

waits for messages
#responses > w? done!

Fault Tolerance
handoff

Fault Tolerance
handoff

what if we add (or lose) servers?

Fault Tolerance
handoff

what if we add (or lose) servers?

"handing off" responsibility for vnode

Fault Tolerance
handoff

what if we add (or lose) servers?

"handing off" responsibility for vnode

series of callbacks in Vnode module

Fault Tolerance
handoff

what if we add (or lose) servers?

"handing off" responsibility for vnode

series of callbacks in Vnode module

mostly a matter of serialisation

Buzzword Bingo

Stateful - ✓

Distributed - ✓

Fault-tolerant

Real-time

Impress your cat

(application)

Buzzword Bingo

Stateful - ✓

Distributed - ✓

Fault-tolerant - ✓

Phoenix!

Real-time

Impress your cat

(application)

Phoenix + Riak Core

Phoenix + Riak Core

just use an umbrella!

Phoenix + Riak Core

just use an umbrella!

then use the Service API in your Phoenix app somewhere

Phoenix + Riak Core

just use an umbrella!

then use the Service API in your Phoenix app somewhere

```
scope "/api", MyApp do
    pipe_through :api
    put "/store/:key", StoreController, :store
    get "/store/:key", StoreController, :fetch
end
```

Phoenix + Riak Core

```
curl -XPUT -d '{"a":"b"}' localhost:4000/api/store/my_key
```

Phoenix + Riak Core

```
curl -XPUT -d '{"a":"b"}' localhost:4000/api/store/my_key
```

```
defmodule MyApp.StoreController do
  use Phoenix.Controller
  use MyApp.Web, :controller











end
```

Phoenix + Riak Core

```
curl -XPUT -d '{"a":"b"}' localhost:4000/api/store/my_key
```

```
defmodule MyApp.StoreController do
  use Phoenix.Controller
  use MyApp.Web, :controller
  def store(



  ) do




  end

end
```

Phoenix + Riak Core

```
curl -XPUT -d '{"a":"b"}' localhost:4000/api/store/my_key
```

```
defmodule MyApp.StoreController do
  use Phoenix.Controller
  use MyApp.Web, :controller
  def store(
    %Plug.Conn{body_params: data}=conn,

  ) do



  end

end
```

Phoenix + Riak Core

```
curl -XPUT -d '{"a":"b"}' localhost:4000/api/store/my_key
```

```
defmodule MyApp.StoreController do
  use Phoenix.Controller
  use MyApp.Web, :controller
  def store(
    %Plug.Conn{body_params: data}=conn,
    %{"key" => key}=params
  ) do



  end

end
```

Phoenix + Riak Core

```
curl -XPUT -d '{"a":"b"}' localhost:4000/api/store/my_key
  defmodule MyApp.StoreController do
    use Phoenix.Controller
    use MyApp.Web, :controller
    def store(
      %Plug.Conn{body_params: data}=conn,
      %{"key" => key}=params
    ) do
      n = 3


    end

  end
```

Phoenix + Riak Core

```
curl -XPUT -d '{"a":"b"}' localhost:4000/api/store/my_key
```

```
defmodule MyApp.StoreController do
  use Phoenix.Controller
  use MyApp.Web, :controller
  def store(
    %Plug.Conn{body_params: data}=conn,
    %{"key" => key}=params
  ) do
    n = 3
    result = StoreFetch.Service.store(key, data, n)

  end

end
```

Phoenix + Riak Core

```
curl -XPUT -d '{"a":"b"}' localhost:4000/api/store/my_key
```

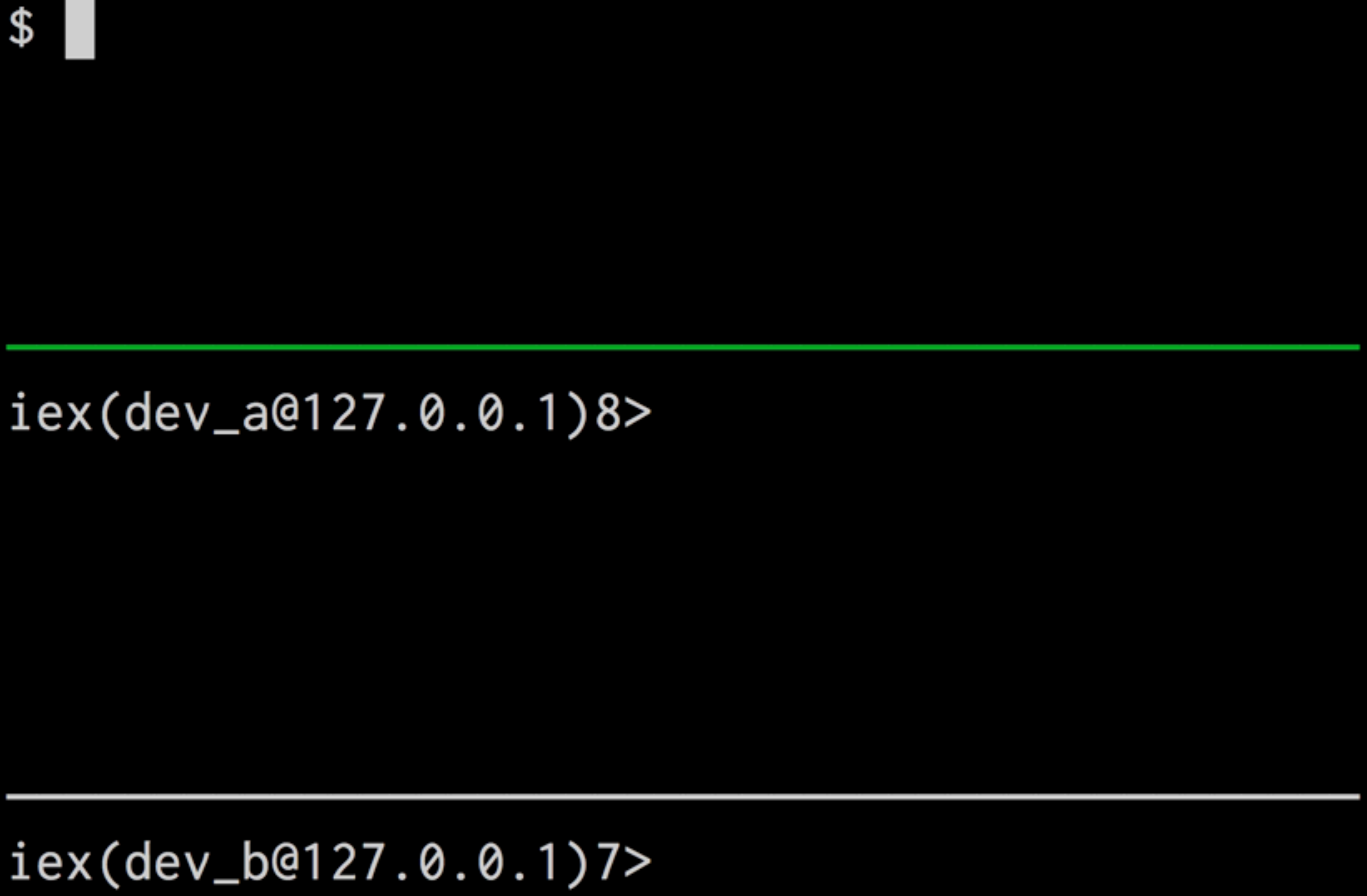
```
defmodule MyApp.StoreController do
  use Phoenix.Controller
  use MyApp.Web, :controller
  def store(
    %Plug.Conn{body_params: data}=conn,
    %{"key" => key}=params
  ) do
    n = 3
    result = StoreFetch.Service.store(key, data, n)
    render conn, store: result
  end

end
```

Phoenix + Riak Core

```
curl -XPUT -d '{"a":"b"}' localhost:4000/api/store/my_key
```

```
defmodule MyApp.StoreController do
  use Phoenix.Controller
  use MyApp.Web, :controller
  def store(
    %Plug.Conn{body_params: data}=conn,
    %{"key" => key}=params
  ) do
    n = 3
    result = StoreFetch.Service.store(key, data, n)
    render conn, store: result
  end
  # … similar for fetch/2
end
```

Demo



```
$ 

iex(dev_a@127.0.0.1)8>

iex(dev_b@127.0.0.1)7>
```

Buzzword Bingo

Stateful - ✓

Distributed - ✓

Fault-tolerant - ✓

Phoenix!

Real-time

Impress your cat

(application)

Buzzword Bingo

Stateful - ✓

Distributed - ✓

Fault-tolerant - ✓

Phoenix! - ✓

Real-time

Impress your cat

(application)

Real-time

Real-time

can we do something with channels?

Real-time

can we do something with channels?

need something to hash on (a key)

Real-time

can we do something with channels?

need something to hash on (a key)

```
%Phoenix.Socket.Broadcast{
    event: "new_msg",
    payload: %{body: "hey everyone!"},
    topic: "rooms:lobby"
}
```

Real-time

can we do something with channels?

need something to hash on (a key)

```
%Phoenix.Socket.Broadcast{
    event: "new_msg",
    payload: %{body: "hey everyone!"},
    topic: "rooms:lobby"
}
```

Phoenix.PubSub.Ricor

Phoenix.PubSub.Ricor

hack hack Phoenix.PubSub adapter

Phoenix.PubSub.Ricor

hack hack Phoenix.PubSub adapter

uses a Riak Core Service to direct messages to vnode (by topic)

Phoenix.PubSub.Ricor

hack hack Phoenix.PubSub adapter

uses a Riak Core Service to direct messages to vnode (by topic)

vnode manages subscriptions and broadcasts

Phoenix.PubSub.Ricor

```
def subscribe(pid, topic, opts) do

end

def unsubscribe(pid, topic) do

end

def broadcast(pid, topic, message) do

end
```

Phoenix.PubSub.Ricor

```
def subscribe(pid, topic, opts) do
  Pubring.Service.subscribe(pid, topic, opts)
end

def unsubscribe(pid, topic) do
  Pubring.Service.unsubscribe(pid, topic)
end

def broadcast(pid, topic, message) do
  Pubring.Service.broadcast(topic, message)
end
```

Pubring.Service

just like other services:

Pubring.Service

just like other services:

1) hash the key (topic)

Pubring.Service

just like other services:

1) hash the key (topic)

2) get preference list

Pubring.Service

just like other services:

1) hash the key (topic)

2) get preference list

3) send command to vnode from list

Pubring.Service

just like other services:

1) hash the key (topic)

2) get preference list

3) send command to vnode from list

let’s look at the vnode!

Pubring.Service

Pubring.Vnode

```
defmodule Pubring.Vnode do
  @behaviour :riak_core_vnode


end
```

Pubring.Vnode

```
defmodule Pubring.Vnode do
  @behaviour :riak_core_vnode
  def handle_command(

  ) do


  end
  def handle_command(

  ) do




  end

end
```

Pubring.Vnode

```
defmodule Pubring.Vnode do
  @behaviour :riak_core_vnode
  def handle_command(
     :subscribe
  ) do


  end
  def handle_command(
     :broadcast
  ) do




  end

end
```

Pubring.Vnode

```
defmodule Pubring.Vnode do
  @behaviour :riak_core_vnode
  def handle_command(
    {:subscribe, pid, topic}, _sender, %{db: db}=state
  ) do


  end
  def handle_command(
     :broadcast
  ) do




  end

end
```

Pubring.Vnode

```
defmodule Pubring.Vnode do
  @behaviour :riak_core_vnode
  def handle_command(
    {:subscribe, pid, topic}, _sender, %{db: db}=state
  ) do
    result = :ets.insert(db, {topic, pid})

  end
  def handle_command(
     :broadcast
  ) do




  end

end
```

Pubring.Vnode

```
defmodule Pubring.Vnode do
  @behaviour :riak_core_vnode
  def handle_command(
    {:subscribe, pid, topic}, _sender, %{db: db}=state
  ) do
    result = :ets.insert(db, {topic, pid})
    {:reply, result, state}
  end
  def handle_command(
     :broadcast
  ) do




  end

end
```

Pubring.Vnode

```
defmodule Pubring.Vnode do
  @behaviour :riak_core_vnode
  def handle_command(
    {:subscribe, pid, topic}, _sender, %{db: db}=state
  ) do
    result = :ets.insert(db, {topic, pid})
    {:reply, result, state}
  end
  def handle_command(
     :broadcast
  ) do




  end

end
```

Pubring.Vnode

```
defmodule Pubring.Vnode do
  @behaviour :riak_core_vnode
  def handle_command(
    {:subscribe, pid, topic}, _sender, %{db: db}=state
  ) do
    result = :ets.insert(db, {topic, pid})
    {:reply, result, state}
  end
  def handle_command(
    {:broadcast, topic, msg}, _sender, %{db: db}=state
  ) do




  end

end
```

Pubring.Vnode

```
defmodule Pubring.Vnode do
  @behaviour :riak_core_vnode
  def handle_command(
    {:subscribe, pid, topic}, _sender, %{db: db}=state
  ) do
    result = :ets.insert(db, {topic, pid})
    {:reply, result, state}
  end
  def handle_command(
    {:broadcast, topic, msg}, _sender, %{db: db}=state
  ) do
                 :ets.match(db, {topic, :"$1"})



  end

end
```

Pubring.Vnode

```
defmodule Pubring.Vnode do
  @behaviour :riak_core_vnode
  def handle_command(
    {:subscribe, pid, topic}, _sender, %{db: db}=state
  ) do
    result = :ets.insert(db, {topic, pid})
    {:reply, result, state}
  end
  def handle_command(
    {:broadcast, topic, msg}, _sender, %{db: db}=state
  ) do
    for [pid] <- :ets.match(db, {topic, :"$1"}) do

    end

  end

end
```

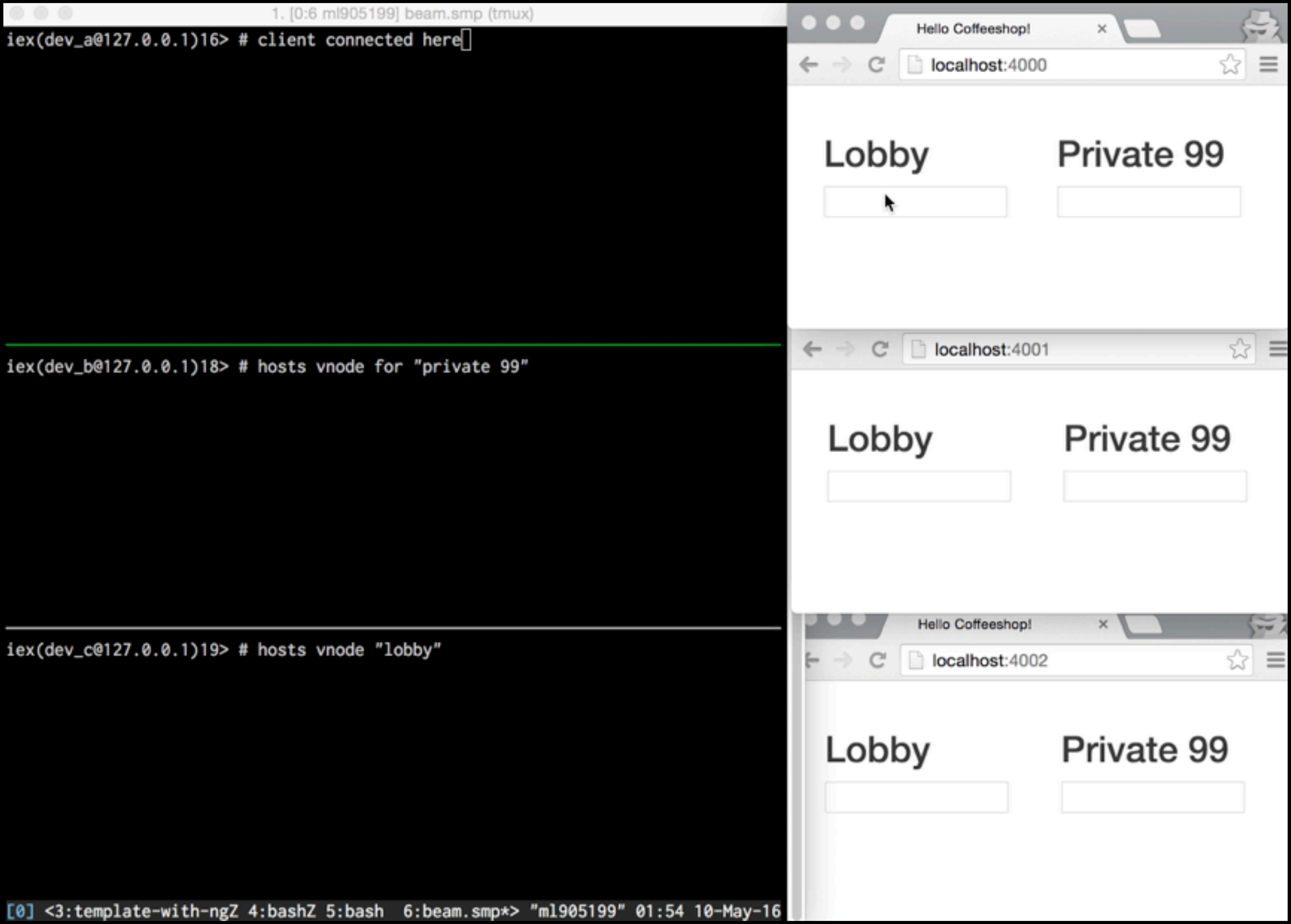
Pubring.Vnode

```
defmodule Pubring.Vnode do
  @behaviour :riak_core_vnode
  def handle_command(
    {:subscribe, pid, topic}, _sender, %{db: db}=state
  ) do
    result = :ets.insert(db, {topic, pid})
    {:reply, result, state}
  end
  def handle_command(
    {:broadcast, topic, msg}, _sender, %{db: db}=state
  ) do
    for [pid] <- :ets.match(db, {topic, :"$1"}) do
      send(pid, msg)
    end

  end

end
```

Pubring.Vnode

```
defmodule Pubring.Vnode do
  @behaviour :riak_core_vnode
  def handle_command(
    {:subscribe, pid, topic}, _sender, %{db: db}=state
  ) do
    result = :ets.insert(db, {topic, pid})
    {:reply, result, state}
  end
  def handle_command(
    {:broadcast, topic, msg}, _sender, %{db: db}=state
  ) do
    for [pid] <- :ets.match(db, {topic, :"$1"}) do
      send(pid, msg)
    end
    {:reply, :ok, state}
  end

end
```

Demo



```
1. [0:6 ml905199] beam.smp (tmux)
iex(dev_a@127.0.0.1)16> # client connected here

iex(dev_b@127.0.0.1)18> # hosts vnode for "private 99"

iex(dev_c@127.0.0.1)19> # hosts vnode "lobby"

[0] <3:template-with-ngZ 4:bashZ 5:bash  6:beam.smp*> "ml905199" 01:54 10-May-16
```

Phoenix.PubSub.Ricor

Phoenix.PubSub.Ricor

so far, just a really complicated Phoenix.PubSub.PG2

Phoenix.PubSub.Ricor

so far, just a really complicated
Phoenix.PubSub.PG2

but…state?

Phoenix.PubSub.Ricor

so far, just a really complicated
Phoenix.PubSub.PG2

but…state?

message history, game state, etc.

Phoenix.PubSub.Ricor

so far, just a really complicated
Phoenix.PubSub.PG2

but…state?

message history, game state, etc.

superpower!

Buzzword Bingo

Stateful - ✓

Distributed - ✓

Fault-tolerant - ✓

Phoenix! - ✓

Real-time

Impress your cat

(application)

Buzzword Bingo

Stateful - ✓

Distributed - ✓

Fault-tolerant - ✓

Phoenix! - ✓

Real-time - ✓

Impress your cat

(application)

{:ERROR, :NOPE}

Imgur

Buzzword Bingo

Stateful - ✓

Distributed - ✓

Fault-tolerant - ✓

Phoenix! - ✓

Real-time - ✓

Impress your cat

(application)

Buzzword Bingo

Stateful - ✓

Distributed - ✓

Fault-tolerant - ✓

Phoenix! - ✓

Real-time - ✓

Impress your cat - :(

(application)

Future

Future

```
mix ricor.new MyApp
```

Future

```
mix ricor.new MyApp

--phoenix
```

Future

```
mix ricor.new MyApp

--phoenix

--write-coord
```

Future

```
mix ricor.new MyApp

--phoenix

--write-coord
```

`GenVnode`

Future

```
mix ricor.new MyApp

--phoenix

--write-coord

GenVnode

Phoenix.PubSub.Ricor.ButSerious
```

Future

`mix ricor.new MyApp`

`--phoenix`

`--write-coord`

`GenVnode`

`Phoenix.PubSub.Ricor.ButSerious`

???

Future

(your app here)

Thanks!

- Mariano Guerra - Little Riak Core book, rebar3 Ricor template, talks
- Mark Allen - Udon, Ricor talk, Basho blog
- Ryan Zezeski - 'Try Try Try' blog
- Project FIFO - a Riak Core that compiles on Erlang 18
- All the comments in the riak_core source

{:exit, :talk_over}

Talk Materials

https://github.com/kanatohodets/scalable-stateful-web-phoenix-riak-core-talk <- not quite a transcript

https://github.com/kanatohodets/elixir_riak_core_ping

https://github.com/kanatohodets/phoenix-ricor-kv

https://github.com/kanatohodets/hashpub