THREAD SAFETY FIRST !

Emily Stolfo

Ruby Engineer at mongoDB, Adjunct Faculty at Columbia

@EmStolfo

There is no such thing as thread-safe Ruby code.

Emily Stolfo

Ruby Engineer at mongoDB

Adjunct Faculty at Columbia

~~Expert~~

THREAD SAFETY FIRST

1. Concurrency in Ruby
2. Writing thread-safe code
3. Testing concurrency

Demo code

```
require 'set'

members = Set.new
threads = []

200.times do |n|
  threads << Thread.new do
    if n % 2 == 0
      members << n
    else
      members.first.nil?
    end
  end
end

threads.each(&:join)
```

Inconsistent bug?

- ✓ 2.0 with 200 threads
- ✓ JRuby with 10 threads
- ✗ JRuby with 200 threads

There are different Ruby implementations.

Different Ruby implementations have their own semantics.

Threads are like music.

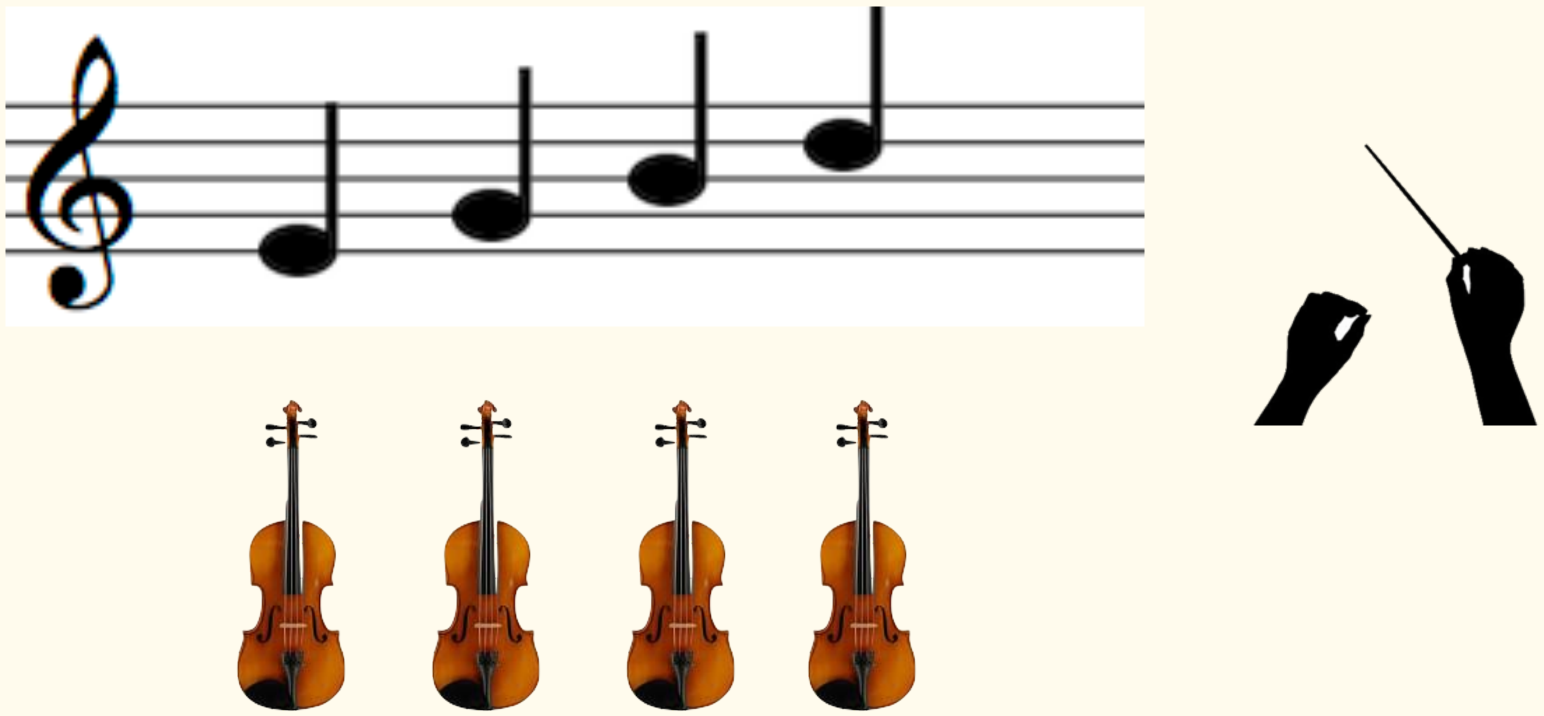
GIL

♩ green thread

native (OS) thread

Threads and Ruby

ruby 1.8



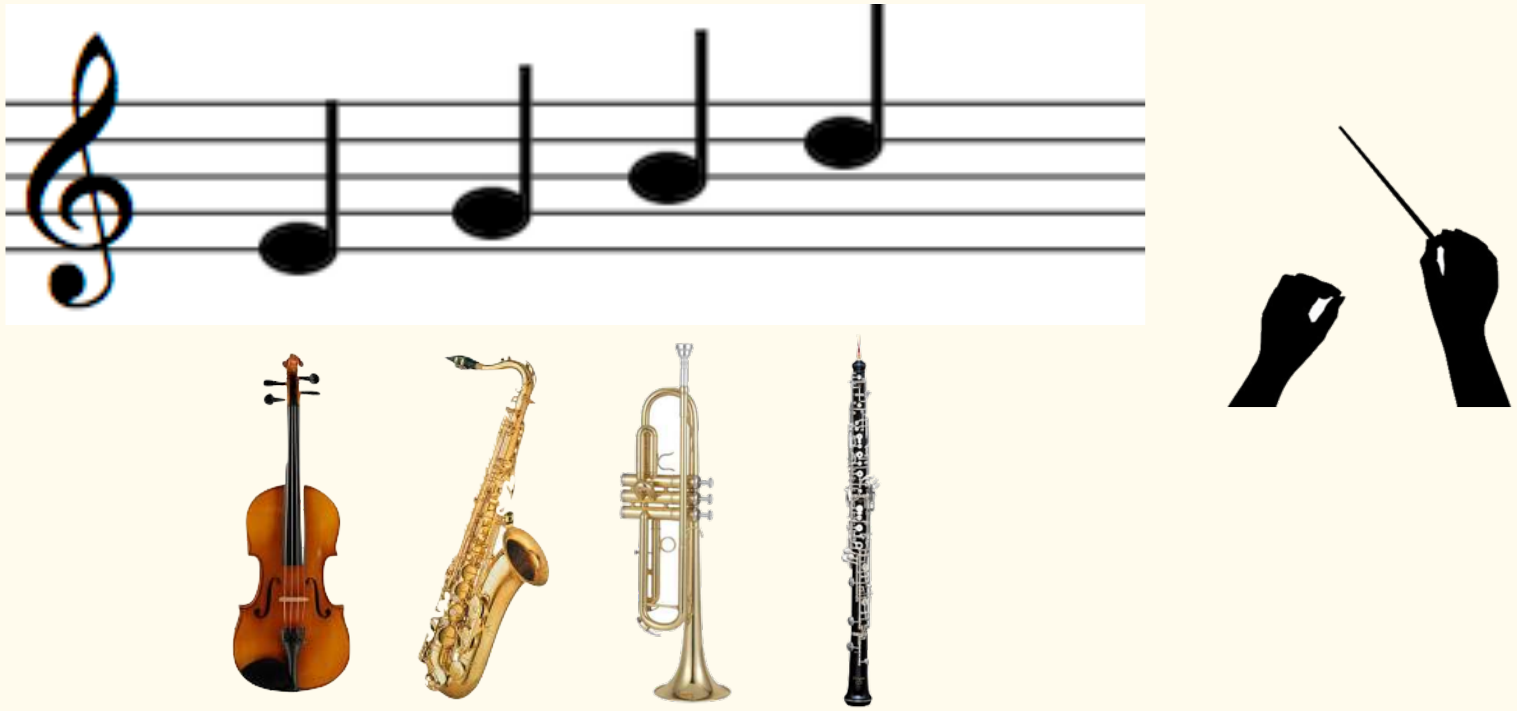
Instruments: OS threads

Notes: green threads

Conductor: GIL

Threads and Ruby

ruby 1.9+



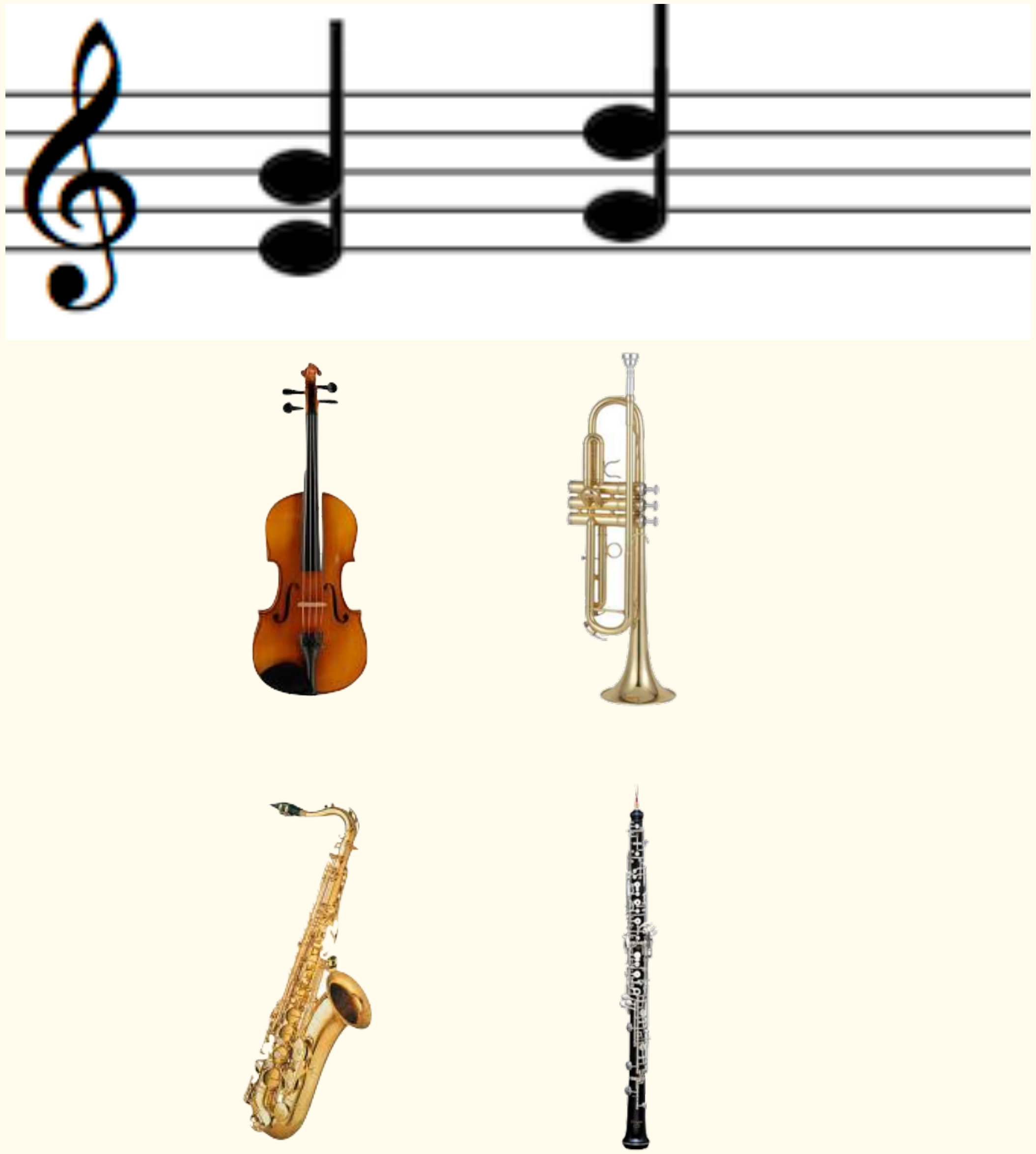
Instruments: OS threads

Notes: green threads

Conductor: GIL

Threads and Ruby

JRuby



Instruments: OS threads

Notes: green threads

Different Rubies?
Different semantics.

Our code sometimes works by **sheer luck.**

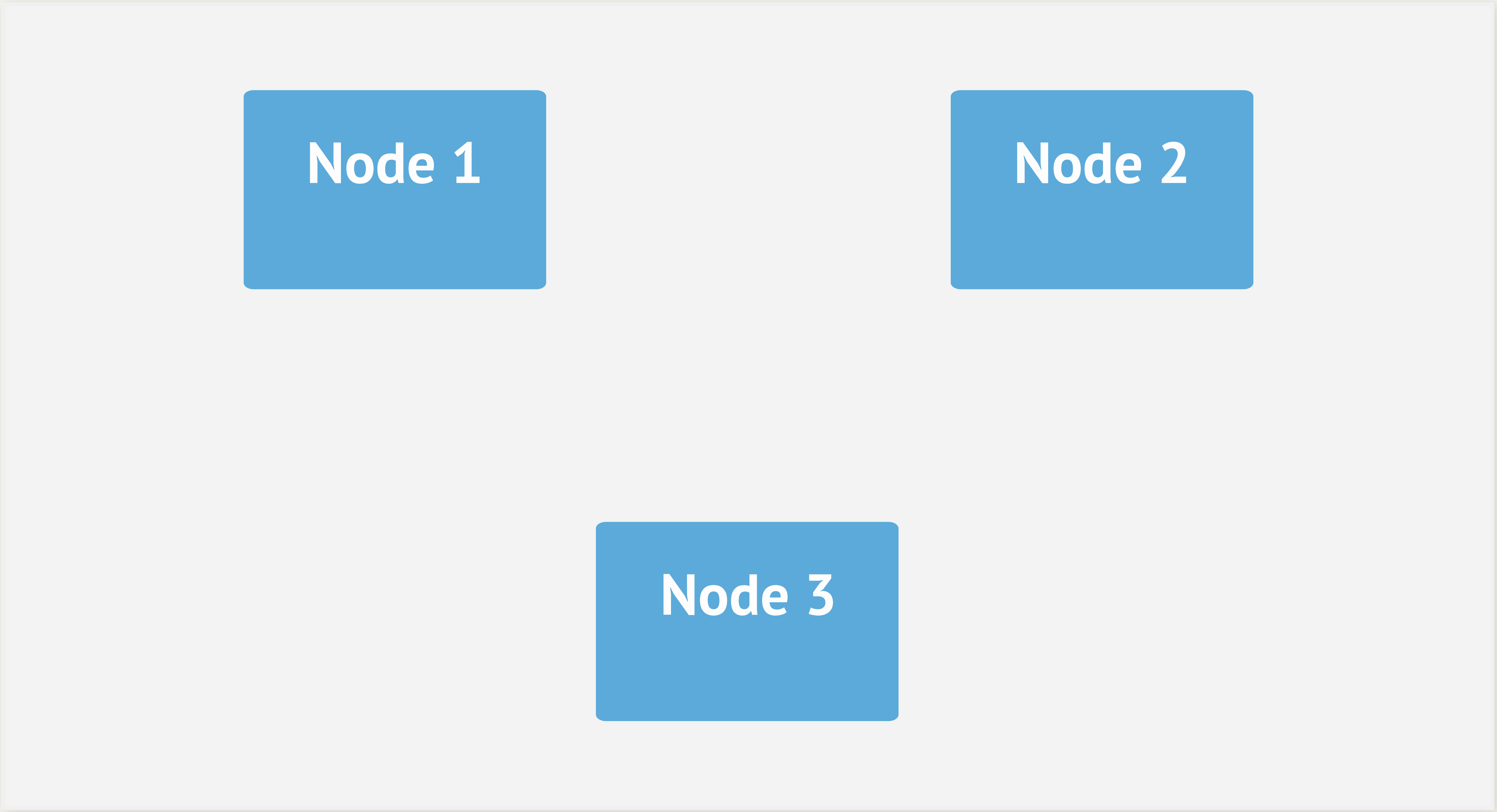
You’re lucky your code hasn’t run on JRuby.

You’re lucky
there is a GIL.

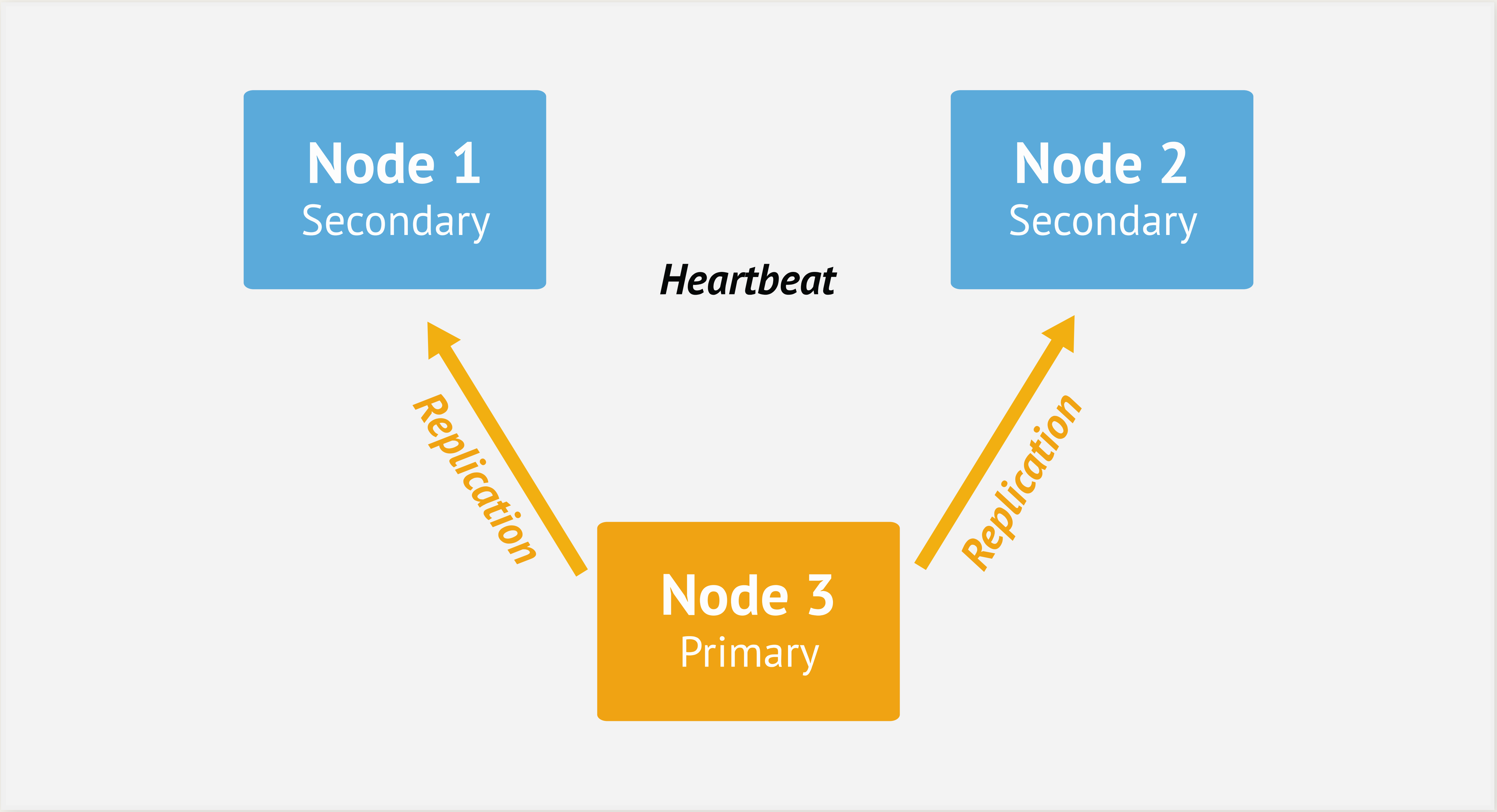
Example:

MongoDB Ruby driver and Replica Sets

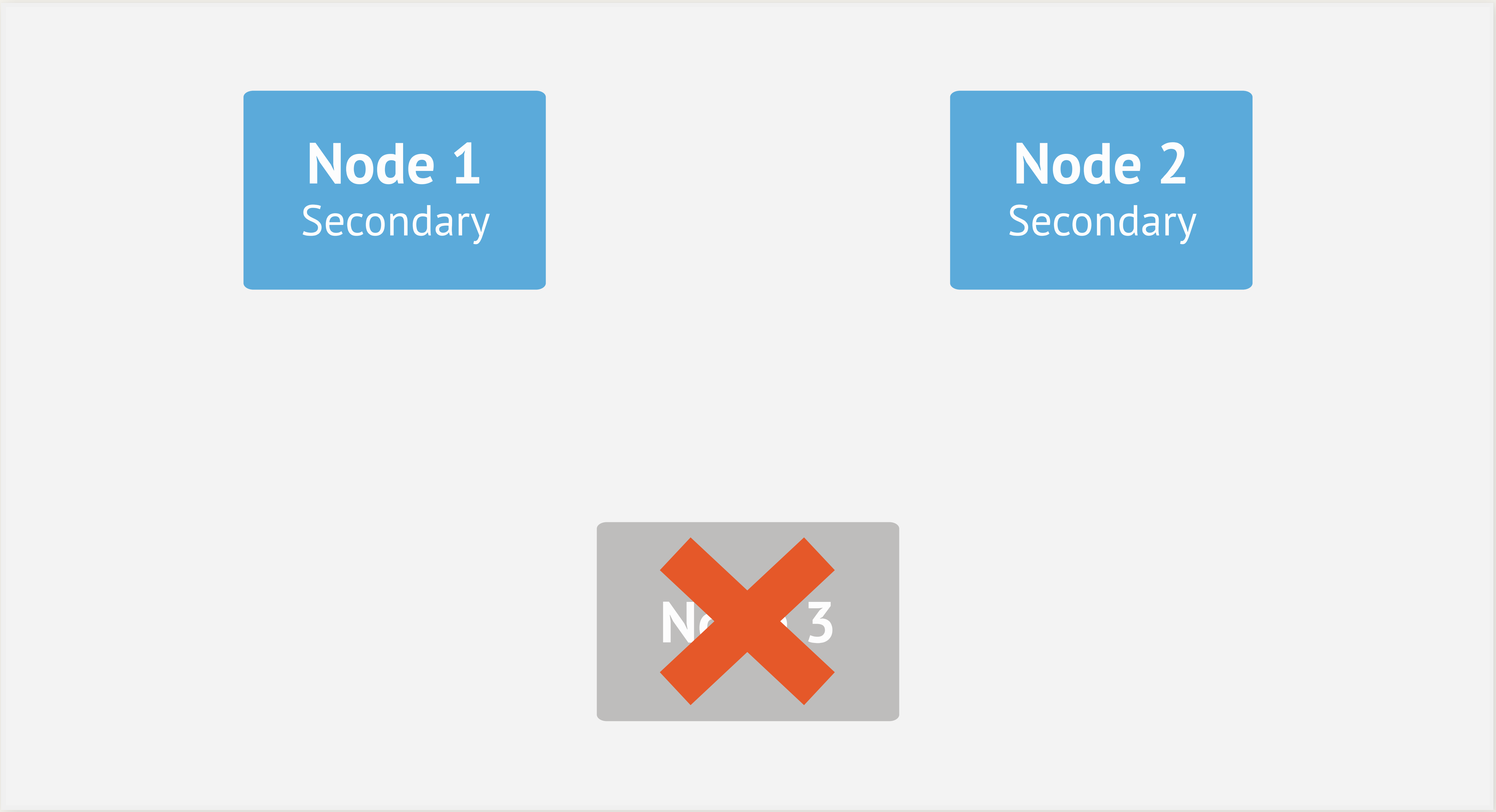
MongoDB Replica Set Creation



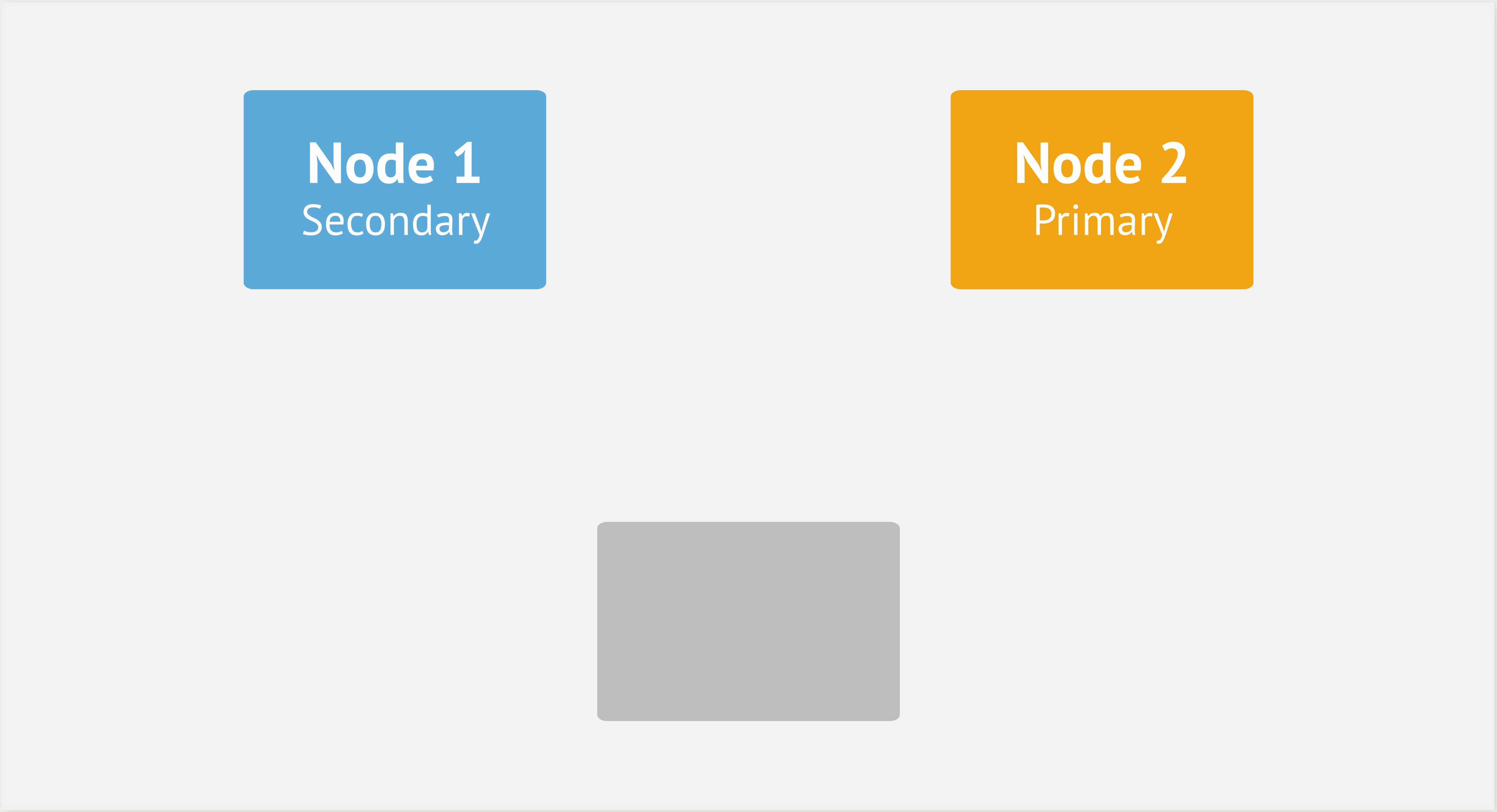
MongoDB Replica Set



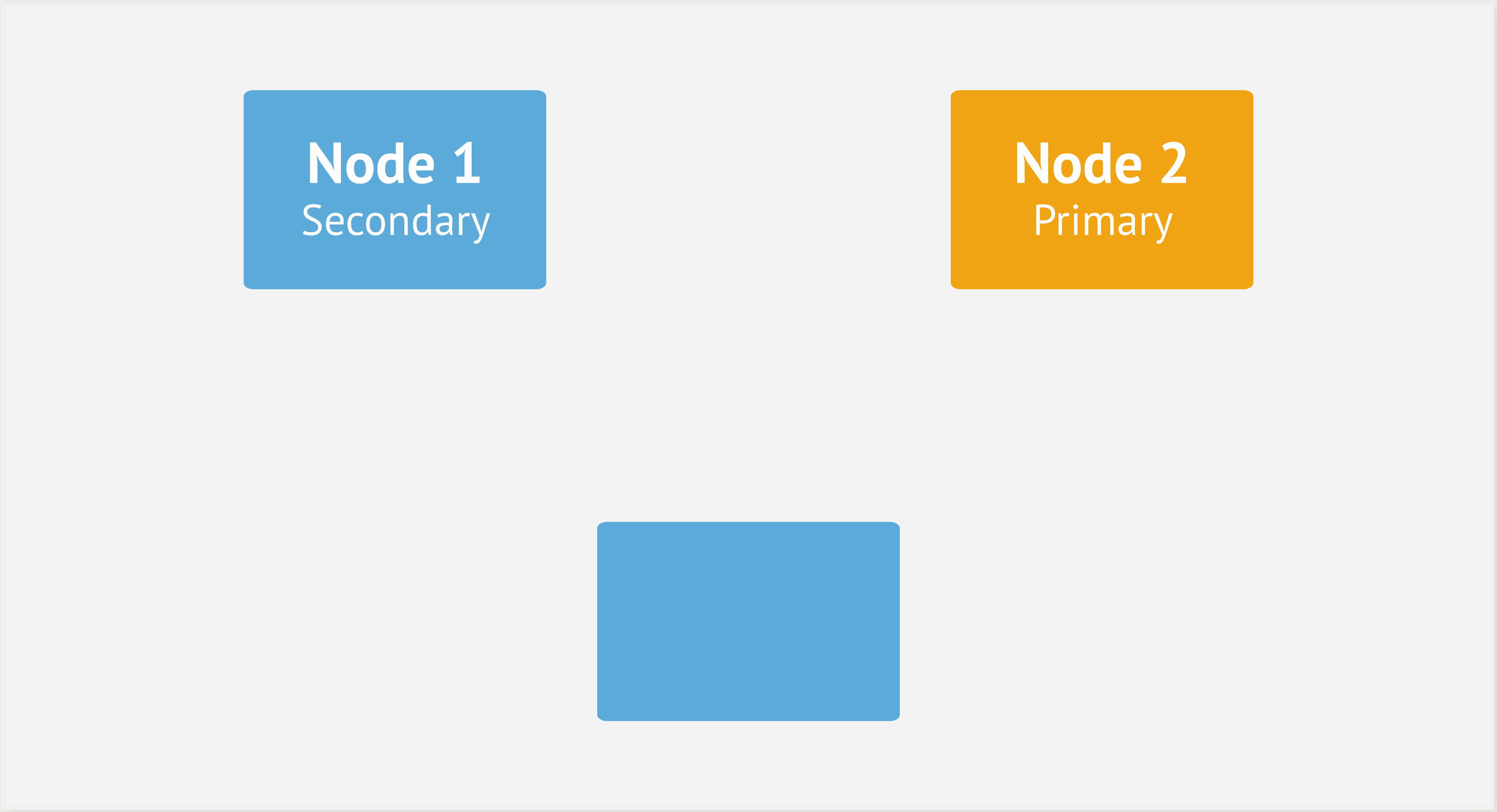
MongoDB Replica Set Failure



MongoDB Replica Set Recovery



MongoDB Replica Set Recovered



Mutable shared state.

Ruby driver

```ruby
class ReplicaSetClient
  def initialize
    @nodes = Set.new
  end

 ...

  def connect_to_nodes
    seed = valid_node
    seed.node_list.each do |host|
      node = Node.new(host)
      @nodes << node if node.connect
    end
  end

  def choose_node(type)
    @nodes.detect { |n| n.type == type }
  end
end
```

Potential concurrency bug

```
class ReplicaSetClient
  def initialize
    @nodes = Set.new
  end

 ...

  def connect_to_nodes
    seed = valid_node
    seed.node_list.each do |host|
      node = Node.new(host)
      @nodes << node if node.connect
    end
  end

  def choose_node(type)
    @nodes.detect { |n| n.type == type }
  end
end
```

```
(RuntimeError)
 "can't add a new key into
  hash during iteration"
```

We often use hashes as caches.

!

Hashes and their derivatives are not thread-safe in JRuby.

How do we write this code thread-safely?

WRITING THREAD-SAFE CODE

Shared data:

Avoid across threads.

If you can’t avoid shared data, at least avoid shared mutable data.

If you can’t avoid
shared mutable data,
use concurrency primitives.

The top 2 concurrency primitives

1. Mutex

```ruby
class ReplicaSetClient
  def initialize
    @nodes = Set.new
    @connect_mutex = Mutex.new
  end

...

  def connect_to_nodes
    seed = valid_node
    seed.node_list.each do |host|
      node = Node.new(host)
      @connect_mutex.synchronize do
        @nodes << node if node.connect
      end
     end
  end

  def choose_node(type)
    @connect_mutex.synchronize do
      @nodes.detect { |n| n.type == type }
    end
  end
end
```

1. Mutex

Use to isolate code that should be executed by at most one thread at a time.

```
class ReplicaSetClient
  def initialize
    @nodes = Set.new
    @connect_mutex = Mutex.new
  end

...

  def connect_to_nodes
    seed = valid_node
    seed.node_list.each do |host|
      node = Node.new(host)
      @connect_mutex.synchronize do
        @nodes << node if node.connect
      end
     end
  end

  def choose_node(type)
    @connect_mutex.synchronize do
      @nodes.detect { |n| n.type == type }
    end
  end
end
```

shared replica set state update

A mutex is not magic.

Avoid locking around I/O.

1. Mutex

```
class ReplicaSetClient
  def initialize
    @nodes = Set.new
    @connect_mutex = Mutex.new
  end

...

  def connect_to_nodes
    seed = valid_node
    seed.node_list.each do |host|
      node = Node.new(host)
      @connect_mutex.synchronize do
        @nodes << node if node.connect
      end
     end
  end

  def choose_node(type)
    @connect_mutex.synchronize do
      @nodes.detect { |n| n.type == type }
    end
  end
end
```

network I/O

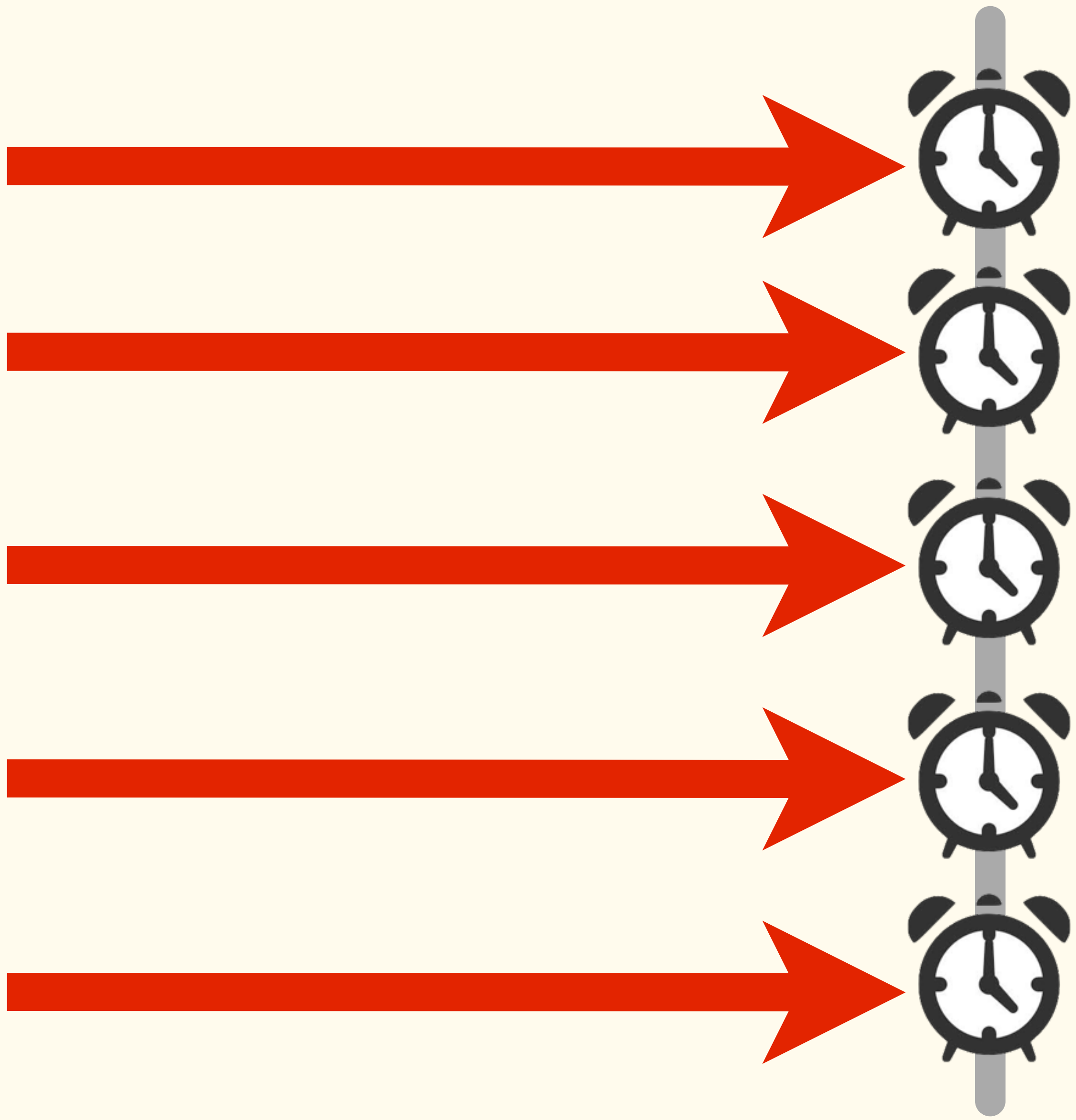
1. Mutex

```
class ReplicaSetClient
  def initialize
    @nodes = Set.new
    @connect_mutex = Mutex.new
  end

...

  def connect_to_nodes
    seed = valid_node
    seed.node_list.each do |host|
      node = Node.new(host)
      @connect_mutex.synchronize do
        @nodes << node
      end if node.connect
     end
  end

  def choose_node(type)
    @connect_mutex.synchronize do
      @nodes.detect { |n| n.type == type }
    end
  end
end
```

network
I/O

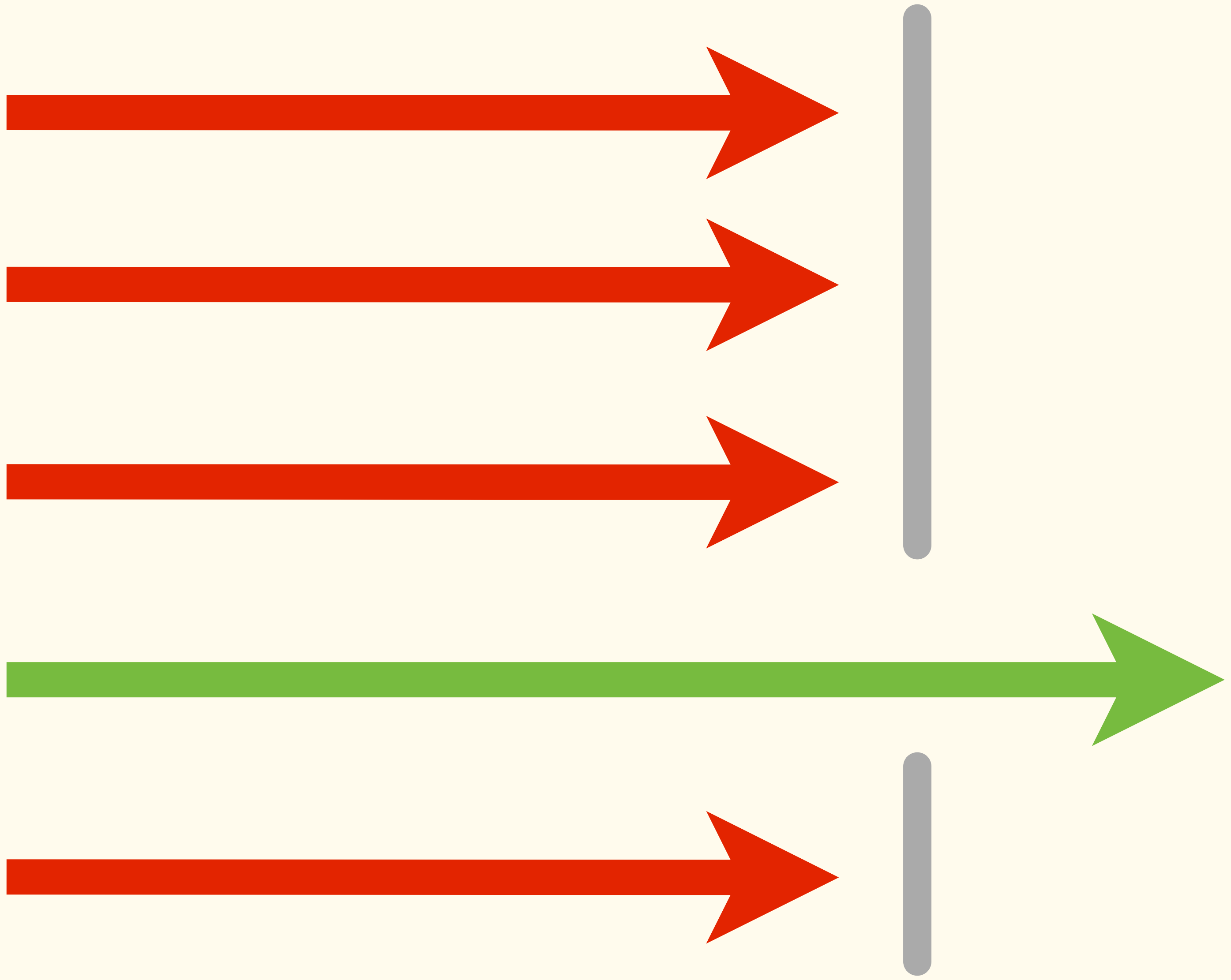
Consider resources.

Ex: Thundering herd



Thundering herd

n threads are woken up after waiting on something, but only 1 can continue. Waste of system resources to wake up n threads.



2. Condition Variable

Use to communicate between threads.

Condition Variable

```
class Pool
  def initialize
    @socket_available = ConditionVariable.new
    ...
  end

  def checkout
    loop do
      @lock.synchronize do
        if @available_sockets.size > 0
          socket = @available_sockets.next
          return socket
        end
        @socket_available.wait(@lock)
      end
    end
  end

  def checkin(socket)
    @lock.synchronize do
      @available_sockets << socket
      @socket_available.broadcast
    end
  end
end
```

Why is this a waste of system resources?

Condition Variable

```ruby
class Pool
  def initialize
    @socket_available = ConditionVariable.new
    ...
  end

  def checkout
    loop do
      @lock.synchronize do
        if @available_sockets.size > 0
          socket = @available_sockets.next
          return socket
        end
        @socket_available.wait(@lock)
      end
    end
  end

  def checkin(socket)
    @lock.synchronize do
      @available_sockets << socket
      @socket_available.signal
    end
  end
end
```

Only one thread can continue

TESTING CONCURRENCY

Testing

1. Test with different implementations.

2. Test with a ton of threads.

3. Use patterns for precision.

Use synchronization methods if you need more precision.

ex: rendezvous / barrier pattern

Concurrency in Ruby

!

Know your implementations.

!

Know your concurrency primitives.

!

Know your code.

“thread-safe JRuby 1.7.4 code“

Thanks

Emily Stolfo

@EmStolfo