



elm



Phoenix
Framework

Making the Web Functional

@chris_mccord / @czaplic

The dysfunctional Web

on the server

- There is no more free lunch
- The world is moving from mostly stateless connections to increasingly stateful ones
- This is at odds with most languages' concurrency models
- OO solutions are becoming increasingly more complex to handle the demands of the modern web

The dysfunctional Web

on the client

- Fragile tooling
- Competing async models
- Framework churn and framework fatigue as the community seeks ideal architectures

The
Pragmatic
Programmers

Programming Phoenix

Productive |> Reliable |> Fast



Chris McCord,
Bruce Tate,
and José Valim

edited by Jacquelyn Carter

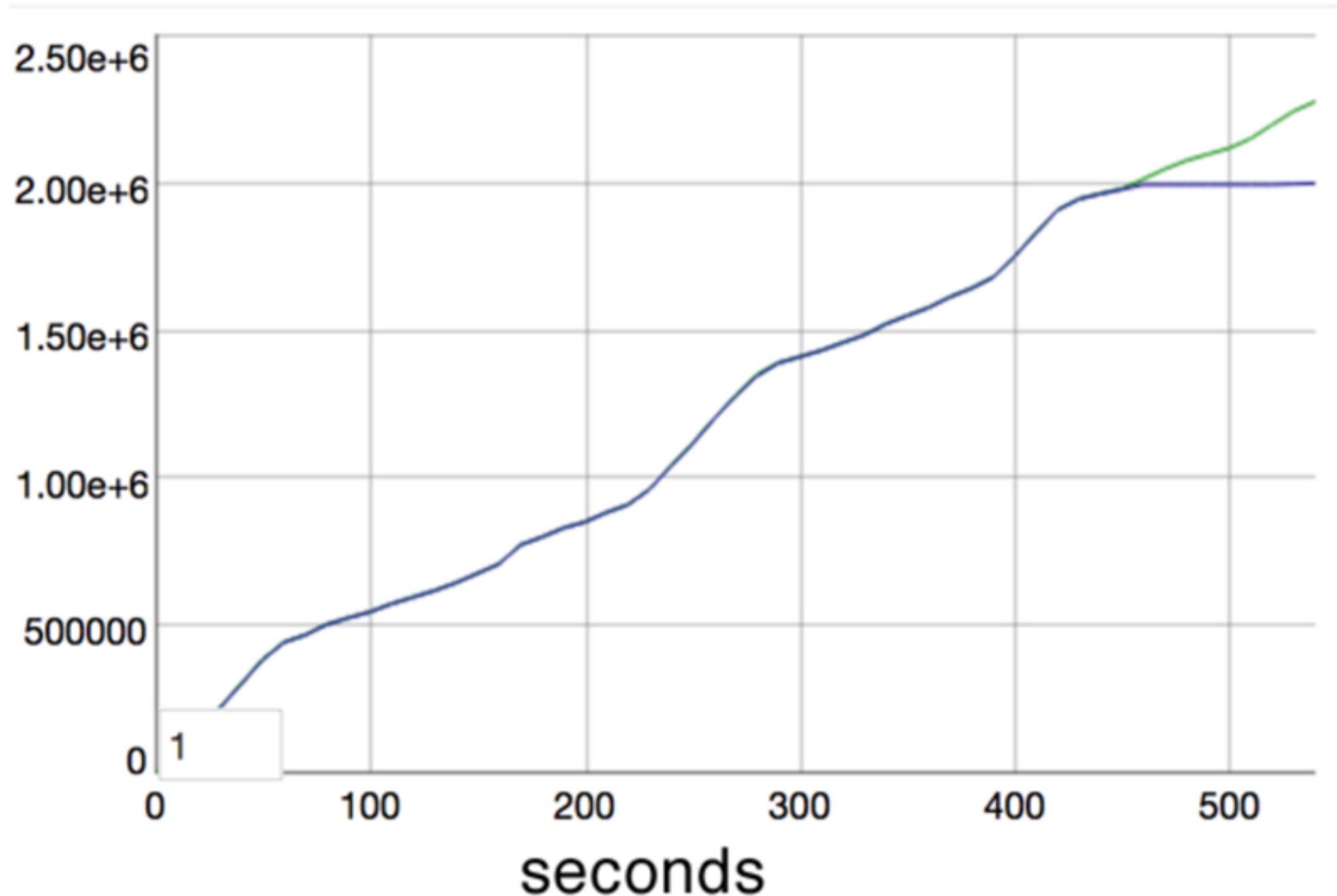


DOCKYARD

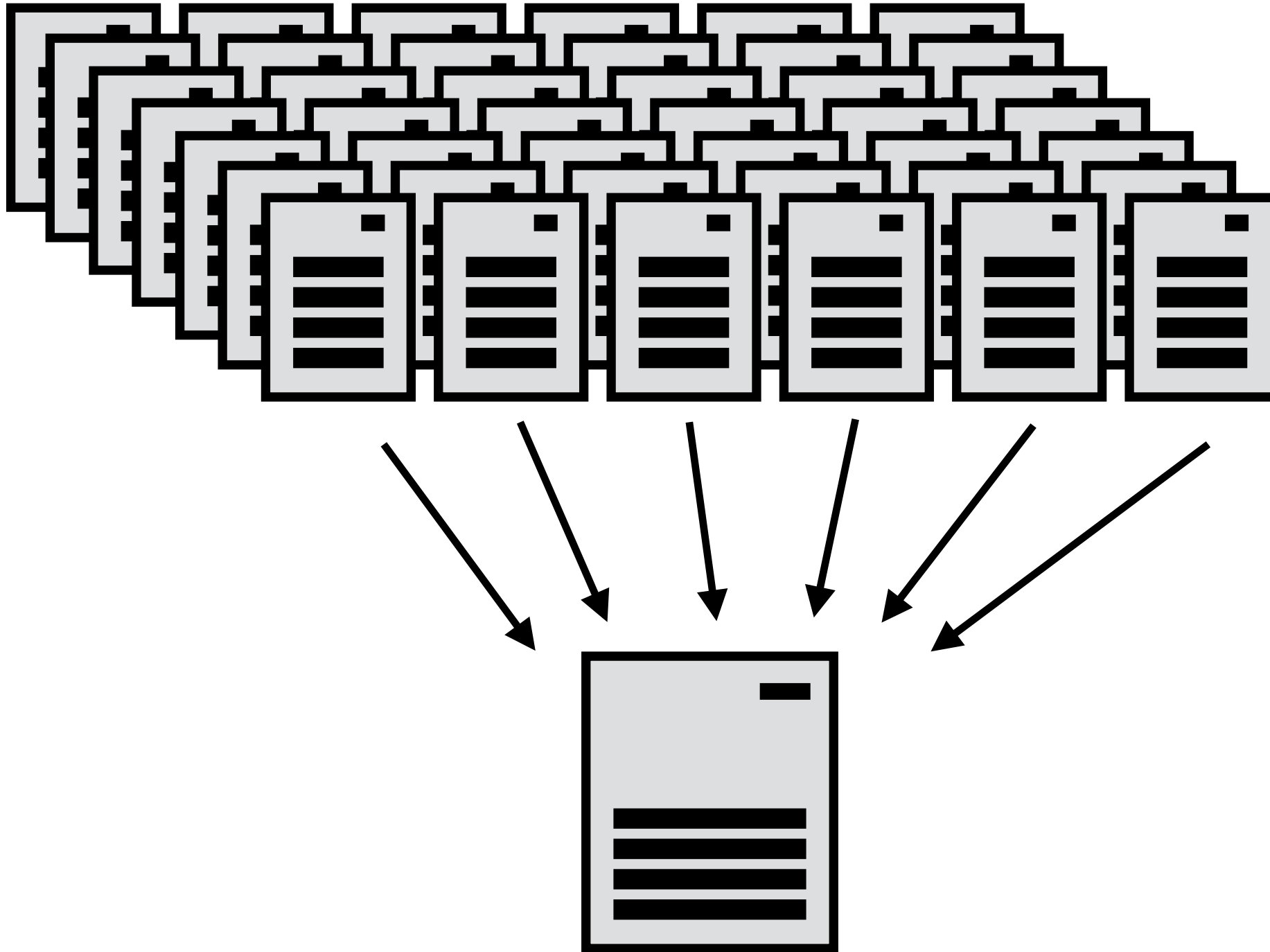
www.dockyard.com

2M channel clients on one server

Simultaneous Users




45 Rackspace 4GB 2vCPU tsung fleet



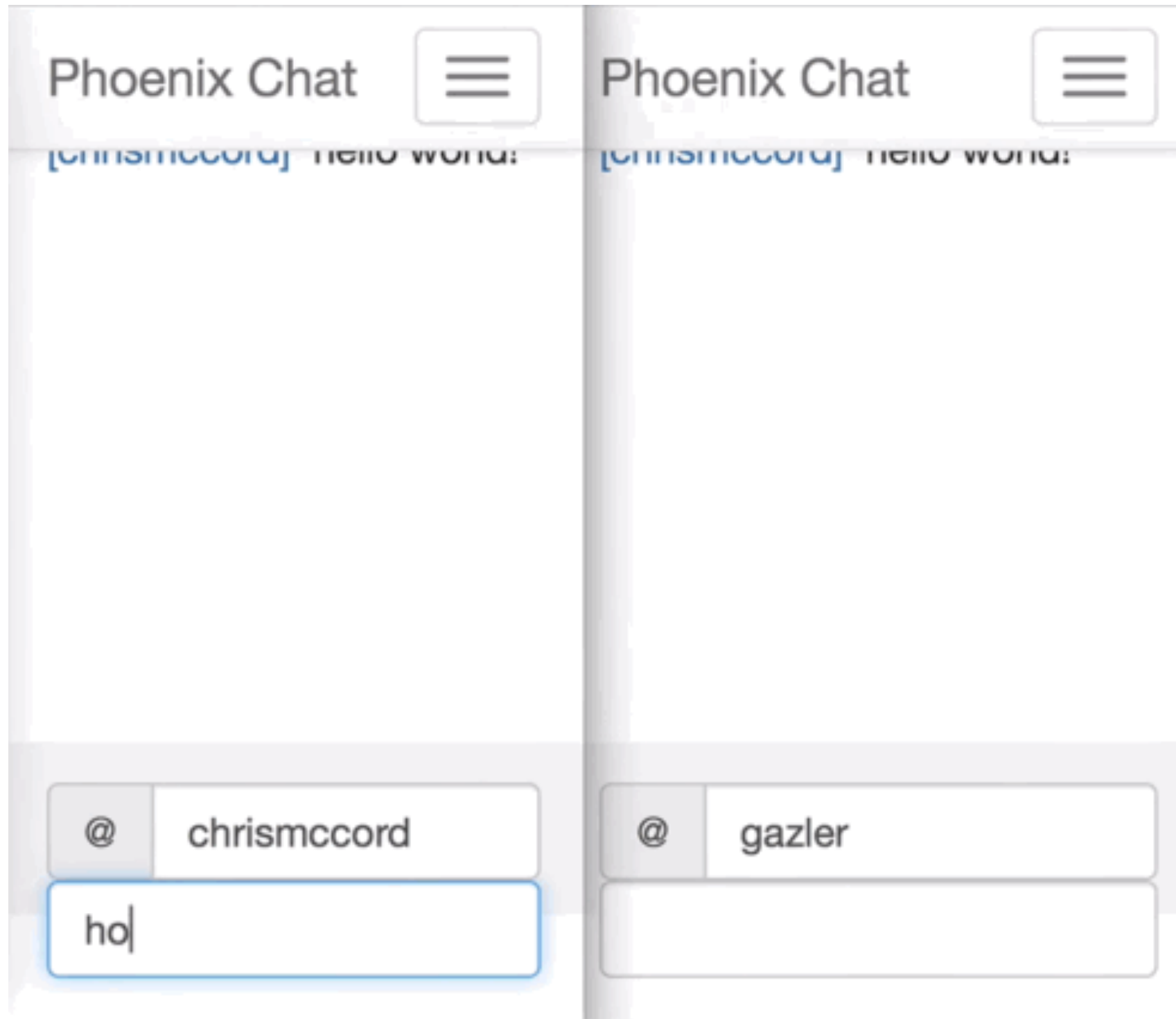
1 Rackspace 40core 2.8Ghz Xeon 128GB Phoenix server

83 of 128GB used

```
1  [ 0.0%] 11 [ 0.5%] 21 [ 0.0%] 31 [ 0.0%]
2  [ 0.0%] 12 [ 0.5%] 22 [ 0.0%] 32 [ 0.0%]
3  [ 0.0%] 13 [ 0.0%] 23 [ 0.0%] 33 [ 0.0%]
4  [ 1.0%] 14 [ 0.0%] 24 [ 0.5%] 34 [ 0.0%]
5  [ 0.5%] 15 [ 0.0%] 25 [ 0.0%] 35 [ 0.0%]
6  [ 0.5%] 16 [ 0.0%] 26 [ 0.0%] 36 [ 0.0%]
7  [ 0.0%] 17 [ 0.0%] 27 [ 0.0%] 37 [ 0.0%]
8  [ 1.0%] 18 [ 0.0%] 28 [ 0.5%] 38 [ 0.0%]
9  [ 0.0%] 19 [ 0.0%] 29 [ 0.0%] 39 [ 0.0%]
10 [ 0.0%] 20 [ 0.0%] 30 [ 0.0%] 40 [ 0.0%]
Mem[|||||||83765/128906MB] Tasks: 22, 150 thr; 2 running
Swp[ 0/0MB] Load average: 5.98 5.45 3.98
Uptime: 5 days, 11:17:13
```



Broadcasting to 2M subscribers



Optimizations

- **30k → 60k subscribers**
 - 14 additions and 69 deletions
- **60k → 330k subscribers**
 - 5 additions and 38 deletions
- **330k → 450k subscribers (+10x arrival rate)**
 - 1 addition and 1 deletion

observer

nonode@nohost

System

Load Charts

Memory Allocators

Applications

Processes

Table Viewer

Trace Overview

Pid	Name or Initial Func	Reds	Memory	MsgQ	Current Function
<0.61.0>	timer_server	1202342	24776	120	gen_server:loop/6
<0.105.0>	gen:init_it/6	0	8920	0	wx_object:loop/6
<0.97.0>	erlang:apply/2	0	2752	0	observer_backend:flag_holder_proc/1
<0.95.0>	erlang:apply/2	21645	142808	0	observer_pro_wx:table_holder/1
<0.94.0>	gen:init_it/6	889	24808	0	wx_object:loop/6
<0.93.0>	gen:init_it/6	0	8920	0	wx_object:loop/6
<0.92.0>	gen:init_it/6	0	7048	0	wx_object:loop/6
<0.91.0>	gen:init_it/6	0	8920	0	wx_object:loop/6
<0.90.0>	gen:init_it/6	0	88728	0	wx_object:loop/6
<0.89.0>	erlang:apply/2	572	8744	0	timer:sleep/1
<0.88.0>	wxe_master	0	21680	0	gen_server:loop/6
<0.87.0>	wxe_server:init/1	72924	27664	0	gen_server:loop/6
<0.55.0>	Elixir.Logger.Watcher:init/1	0	2864	0	gen_server:loop/6
<0.54.0>	Elixir.Logger.Watcher:init/1	0	2968	0	gen_server:loop/6
<0.53.0>	Elixir.Logger.Watcher	0	7016	0	gen_server:loop/6
<0.52.0>	Elixir.Logger.Watcher:init/1	0	2968	0	gen_server:loop/6
<0.51.0>	Elixir.Logger	0	7264	0	Elixir.GenEvent:fetch_msg/5
<0.50.0>	Elixir.Logger.Supervisor	0	10880	0	gen_server:loop/6
<0.49.0>	application_master:start_it/4	0	6912	0	application_master:loop_it/4
<0.48.0>	application_master:init/4	0	2864	0	application_master:main_loop/2
<0.45.0>	Elixir.IEx.Config	0	2824	0	gen_server:loop/6
<0.44.0>	Elixir.IEx.Supervisor	0	5872	0	gen_server:loop/6
<0.43.0>	application_master:start_it/4	0	2760	0	application_master:loop_it/4



Adam Kittelson

@adamkittelson

Was playing around with :observer and spotted a memory leak. Took about 30 seconds to see it was monitor refs. [#myelixirstatus](#)



This repository Search

Pull requests Issues Gist

📄 phoenixframework / phoenix_pubsub

👁 Unwatch ▾ 9

<> Code

🔔 Issues 6

🔗 Pull requests 0

📖 Wiki

📶 Pulse

📊 Graphs

⚙ Settings

memory leak (monitor refs) in Phoenix.PubSub.Local #23

🔔 Open

adamkittelson opened this issue 32 minutes ago · 1 comment



adamkittelson commented 32 minutes ago

I think I found a memory leak in Phoenix.PubSub.Local with monitor refs and long running subscribers that join topics frequently.

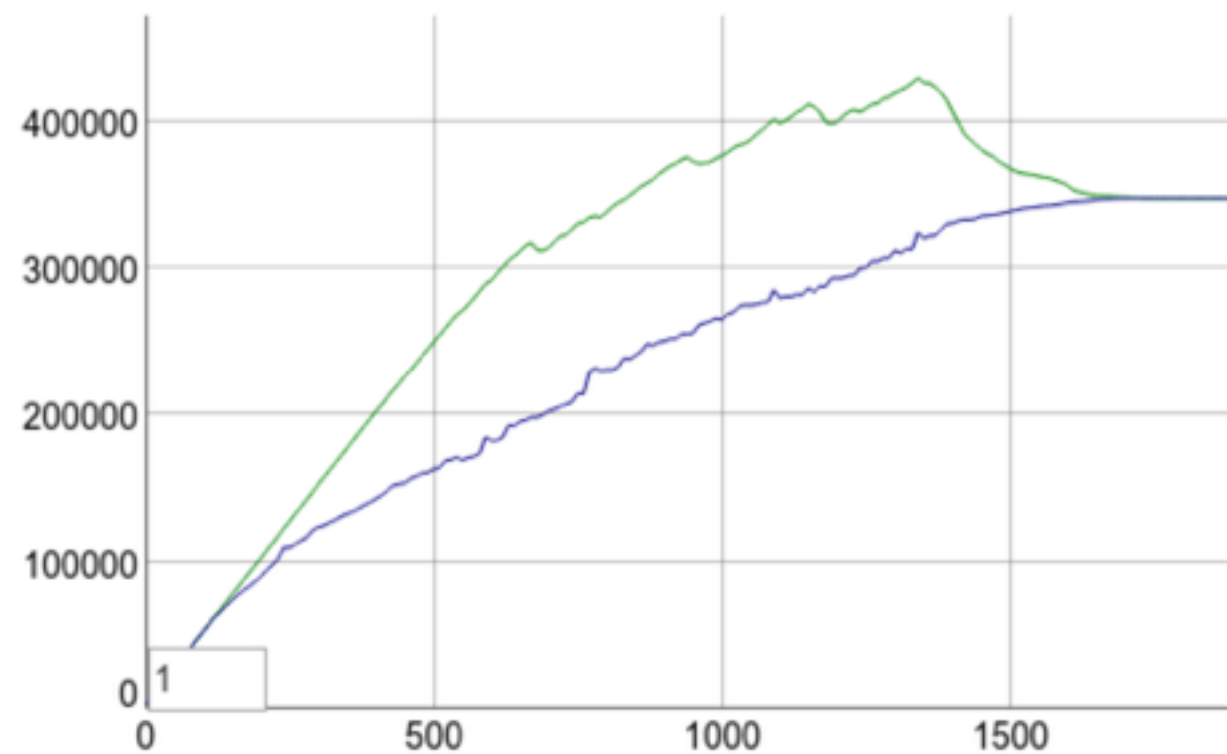
Each time a process subscribes to a topic `Process.monitor` is called at https://github.com/phoenixframework/phoenix_pubsub/blob/9e0e079bcb1a946a5c/lib/phoenix_pubsub/local.ex#L194 causing the Local GenServer to create subscriber pid regardless of whether it's already monitoring that pid as a result from another (or the same) topic.



10x increase in arrival rate

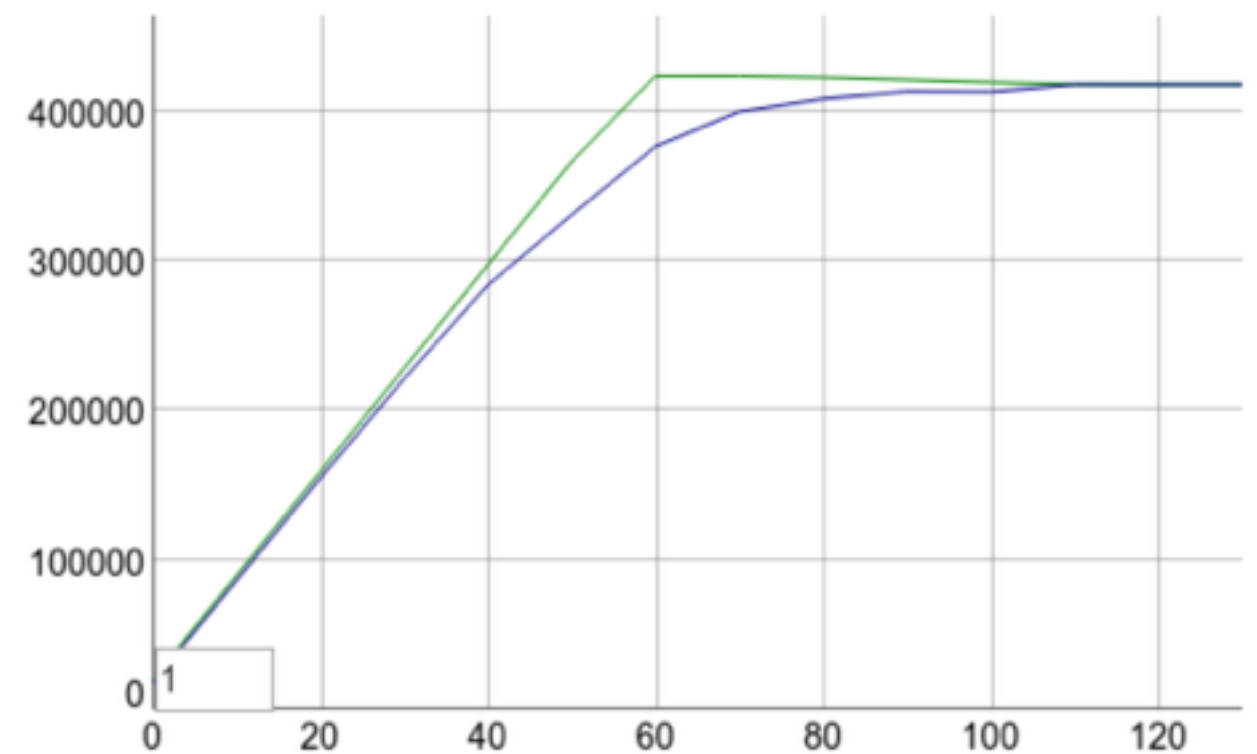
- ^local = :ets.new(local, [:bag, :named_table, :public,
- + ^local = :ets.new(local, [:duplicate_bag, :named_table, :public,

Simultaneous Users



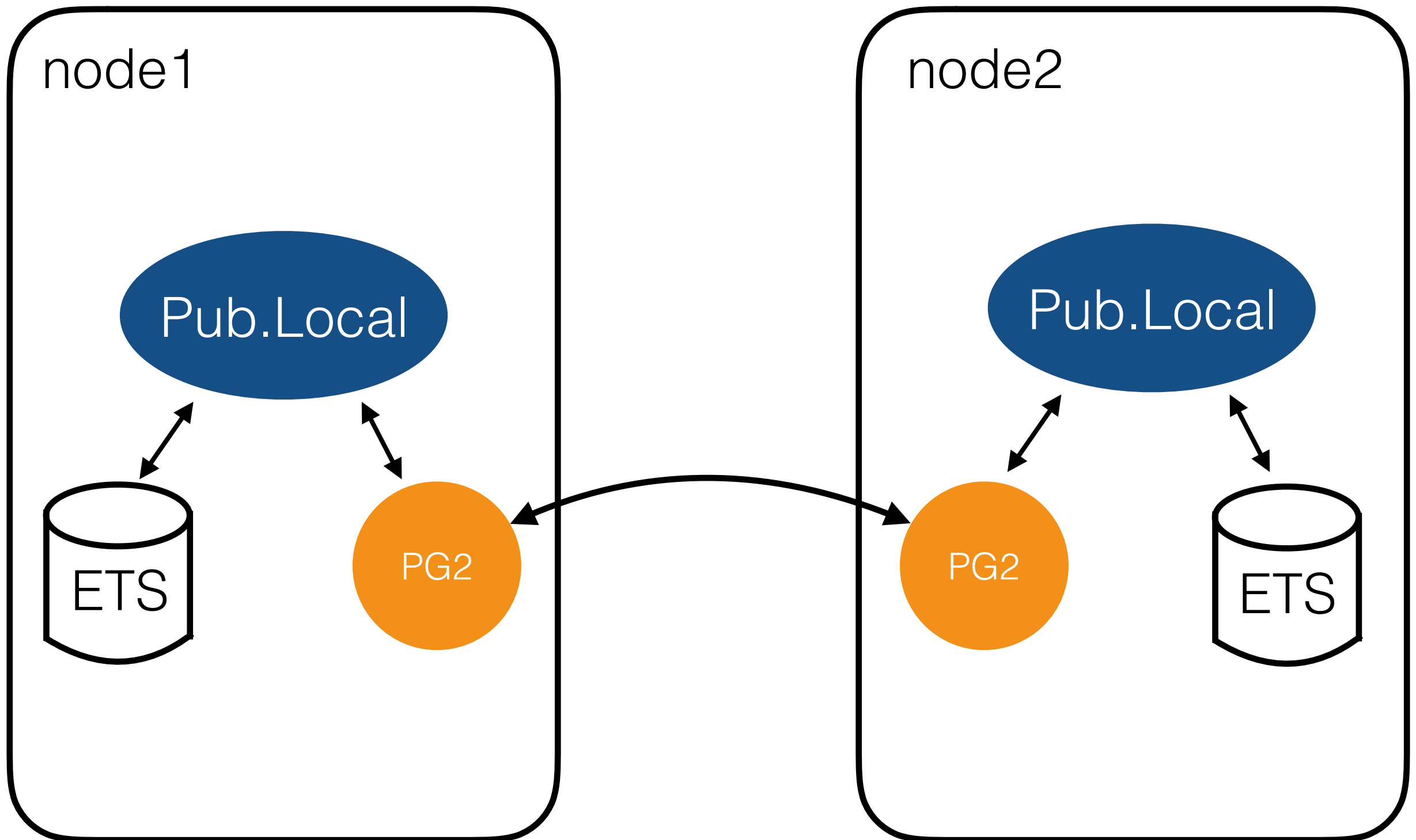
Info »

Simultaneous Users

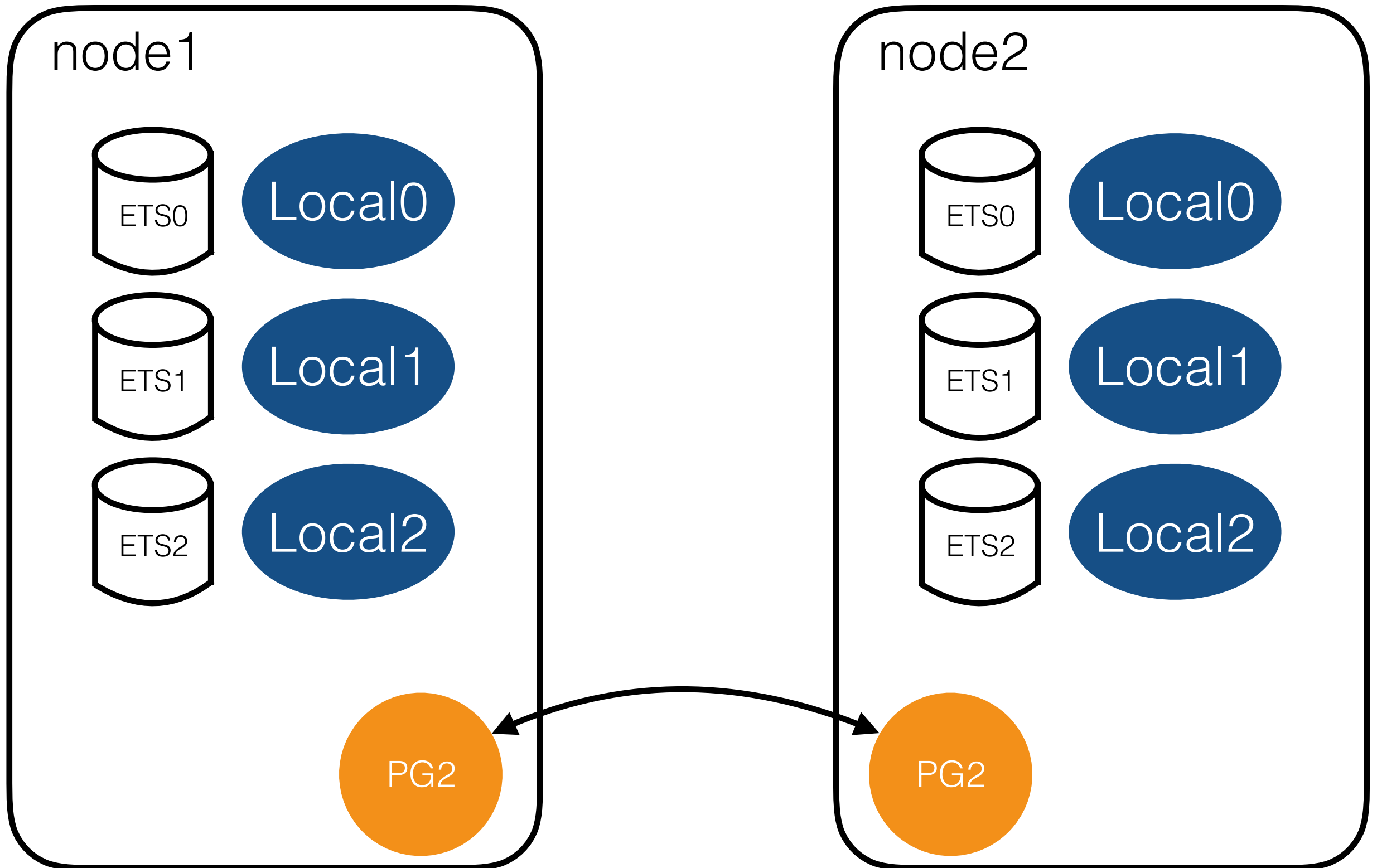


Info »

Sharding Subscriptions



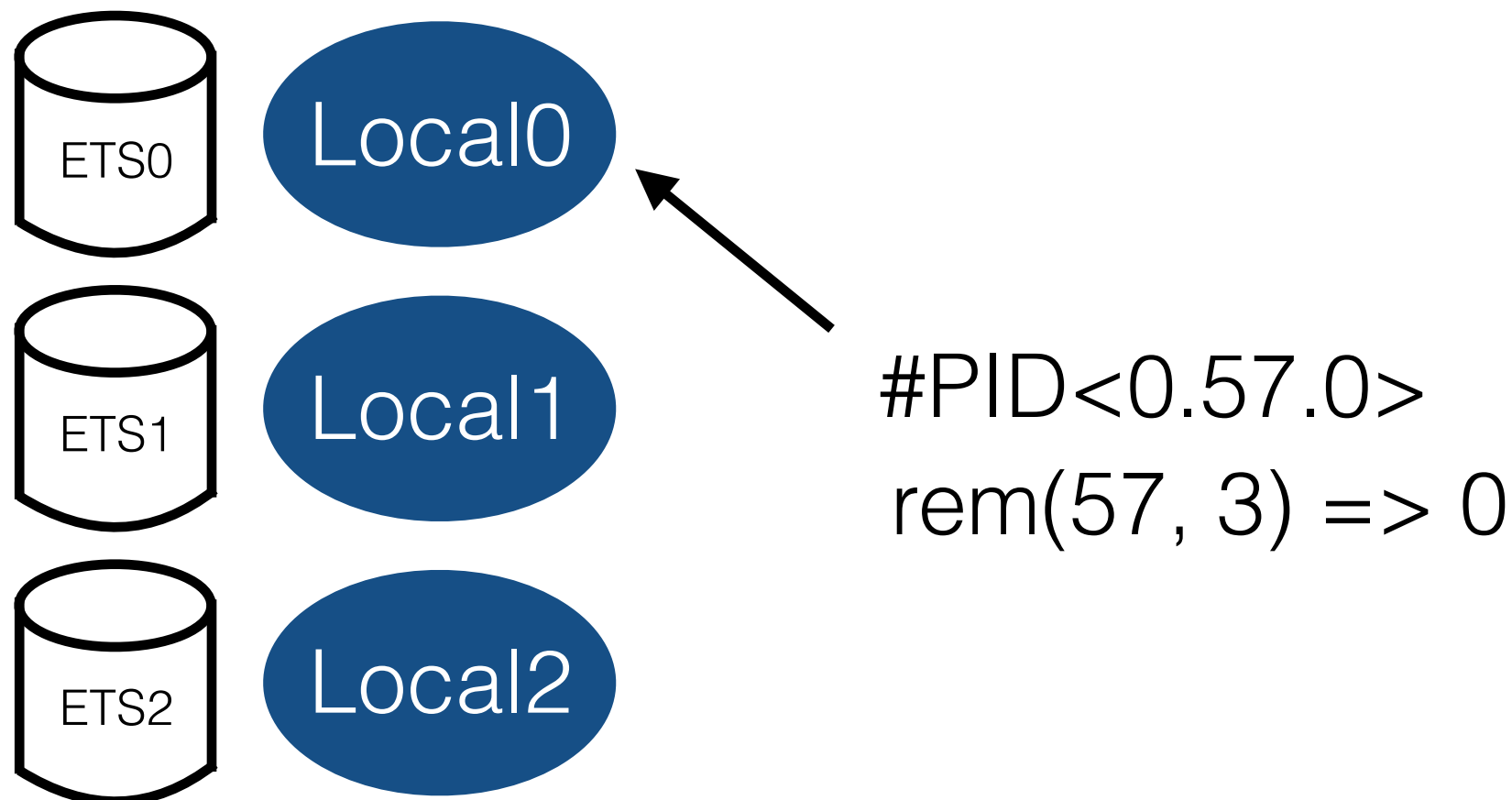
Sharding Subscriptions




```

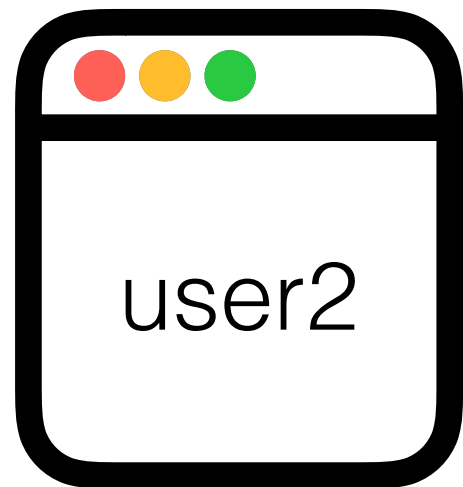
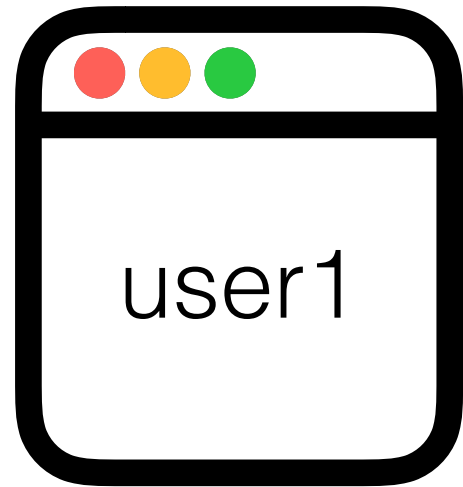
defp pid_to_shard(pid, shard_size) do
  pid
  |> pid_id()
  |> rem(shard_size)
end
defp pid_id(pid) do
  bin = :erlang.term_to_binary(pid)
  pre = (byte_size(binary) - 9) * 8
  <<_::size(pre), id::size(32), _::size(40)>> = bin
  id
end

```



Phoenix.Presence

A look at the problem



Online Users

- user1
- user2

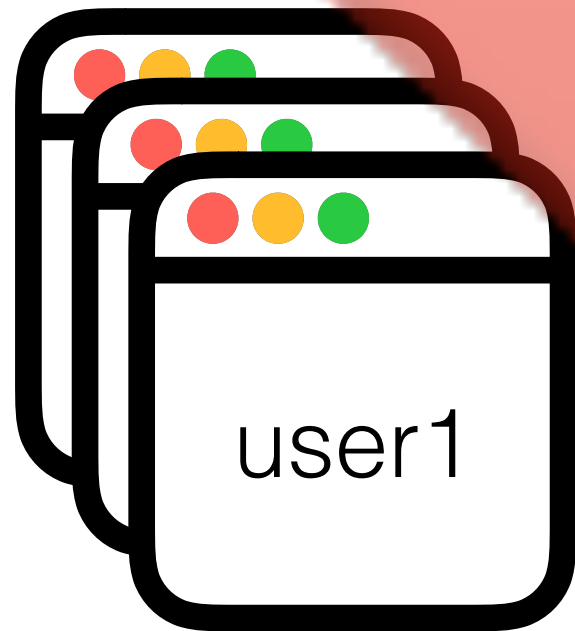
Solution?:

**Presence server that monitors and broadcasts joins/
leaves**

```
def join("lobby", _, socket) do
→ Presence.add(self(), socket.assigns.user)
→ {:ok, %{users: Presence.list(), socket}}
end
```

```
defmodule Presence do
  def handle_call(:list, _, users) do
    {:reply, users, users}
  end
  def handle_call({:add, pid, user}, _, users) do
    ref = Process.monitor(pid)
    broadcast("lobby", "join", user)
    {:reply, :ok, Map.put(users, ref, user)}
  end
  def handle_info({:DOWN, ref, :process, _, _}, users) do
    broadcast("lobby", "leave", Map.get(users, ref))
    {:noreply, Map.delete(users, ref)}
  end
end
```

A look at the problem



Online Users

- user2

`"join", user1`

`"join", user1`

`"join", user1`

`"leave", user1`

`"join", user2`

A look at the problem



`Presence.list()`

- user1

node1

node2



`Presence.list()`

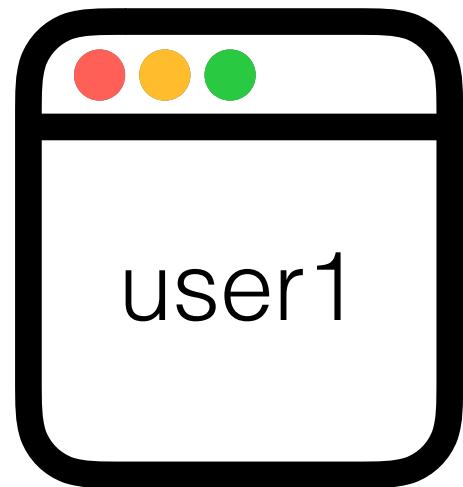
- user2

Solution?:

Presence backed by shared database!



A look at the problem



```
Presence.list()  
• user1  
• user2 (orphaned)
```

node1

node2



```
Presence.list()  
• user1  
• user2
```



redis

The Problem

- Local-node concerns
 - must account for unique presences for same user
- Multi-node concerns
 - must handle node-down events and clean up *local* state for presences belonging to downed node
 - must replicate data across cluster

The Solution

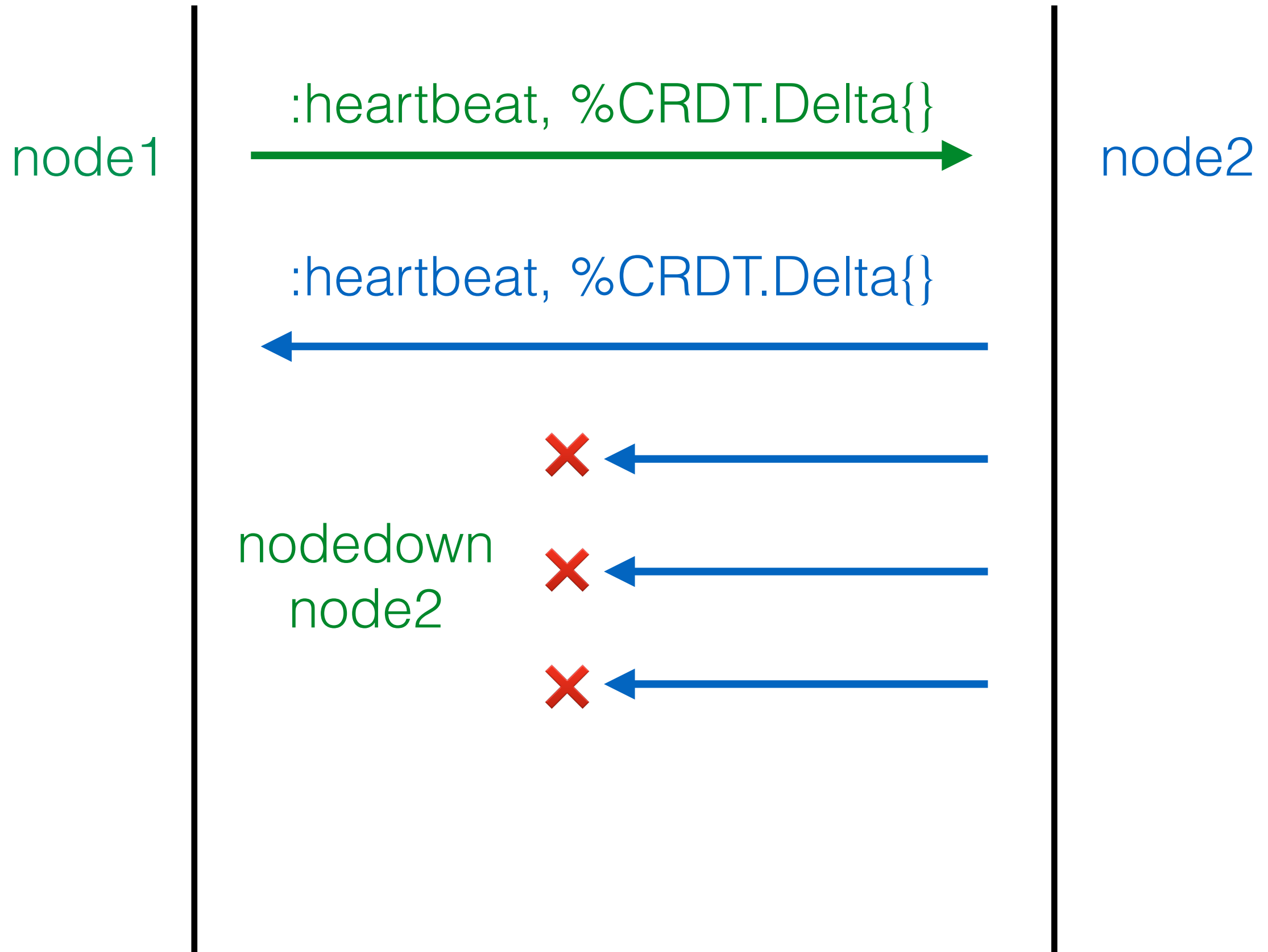
- Ideal Architecture
 - no single source of truth
 - no single point of failure
- CRDT (Conflict-free Replicated Data Type)
- Heartbeat/Gossip protocol

CRDT

conflict-free, replicated data type

- Strong eventual consistency
- Replicate presence join and leave events across the cluster without merge conflicts
- Conflicts are *mathematically impossible*
- Supports replication without remote synchronization

Heartbeat Protocol



Vector Clocks

Catching up on missed deltas

node1

n2: $1 \rightarrow 1$, $\% \{n3: 1, n4: 2\}$

n3: $1 \rightarrow 2$, $\% \{n2: 2, n4: 2\}$ *has updates*

n4: $2 \rightarrow 3$, $\% \{n2: 2, n3: 2\}$ *has updates*

only n4 is selected to request updates since n3 is contained in its future

Server API

```
defmodule RoomChannel do
  def join("rooms:" <> room_id, _, socket) do
    send self(), :after_join
    {:ok, socket}
  end

  def handle_info(:after_join, sock) do
    id = sock.assigns.user_id
    → Presence.track(sock, id, %{status: "avail"})
    → push sock, "presences", Presence.list(sock)
    {:noreply, sock}
  end
end
```

Client API

```
import {Socket, Presence} from "phoenix"
let socket = new Socket("/socket")
let room = socket.channel("rooms:" + id)
let presences = {}

room.on("presences", state => {
  Presence.syncState(presences, state)
})
room.on("presence_diff", diff => {
  Presence.syncDiff(presences, diff)
})
console.log("users", Presence.list(presences))
```


Demo

Making The Web Functional

- Good platforms drive you toward optimal solutions
- You can trust that following the principles laid out produces fast, maintainable programs
- Fast code does not have to equal dense code
- Productive code does not have to equal slow code
- Good platforms let you to focus on what matters – your application

“writing great code should be easy... now it is”

– elm-lang.org