IMPROVING RPC CALLS IN ERLANG AND ELIXIR

Erlang Factory SF 2016

WHO AM I?

Panagiotis "PJ" Papadomitsos

Distributed Systems Architect

Splunk Inc.

AGENDA

- The Problem
- Experimenting
- The Fix
- Features
- Architecture
- Performance
- Shortcomings
- Coding in Elixir for fun and fun
- Features
- The Good
- The Bad
- Performance Considerations
- The Future
- Demo

THE PROBLEM

- Erlang cluster with >100 nodes
- Each node uses **rpc:call** to ship analytics data to other nodes
- RPC payload between 10KB to 2MB
- Extremely high traffic, 24/7, 150.000 calls/sec/node

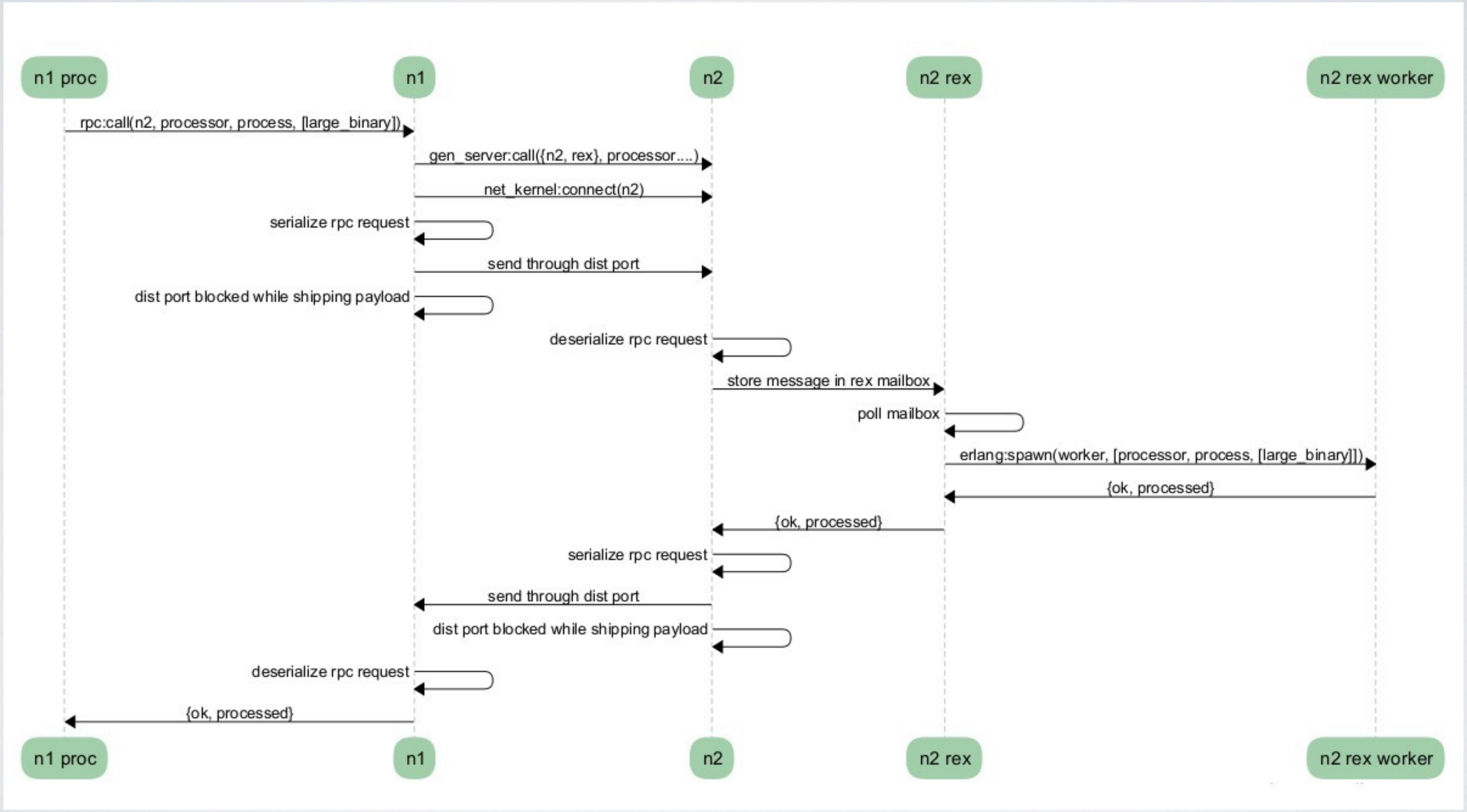
THE PROBLEM (CONT'D)

- Nodes would crash after a while due to mailbox issues
- Scaling up didn't solve the problem enough to make sense financially
- We needed to start tracing/introspecting the system

WHAT WE FOUND

- The **rpc** library uses a single **rex** gen_server to **receive** messages from **any** node
- A single mailbox per node is responsible for receiving messages from **every** node in the cluster
- Those were the mailboxes we were looking for

REX ARCHITECTURE



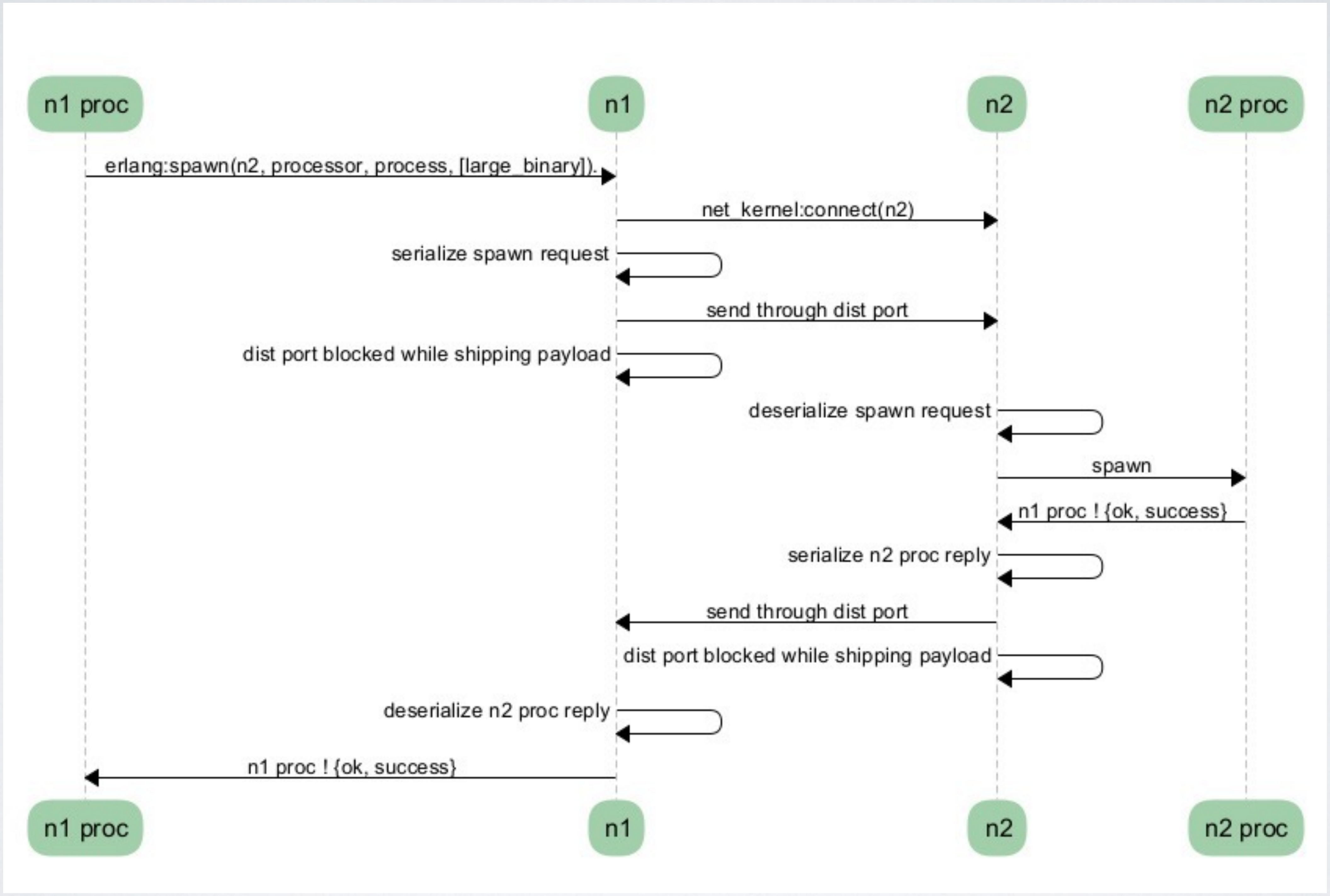
EXPERIMENT #1

- Switch from **rpc:call** to **rpc:cast**
- Does not return error if the node on the other side has gone down
- Still limited by the single mailbox **rex** server

EXPERIMENT #2

- Switched to a hybrid solution with **erlang:spawn** to remote nodes
- Not limited to a **single** mailbox
- But had to implement **naive** connection state logic
- {monitor,<4685.187.0>,busy_dist_port,#Port<4685.41652>} (thanks Wombat!)
- Mnesia was crashing more often than the Chinese stock market

SPAWN/4 ARCHITECTURE

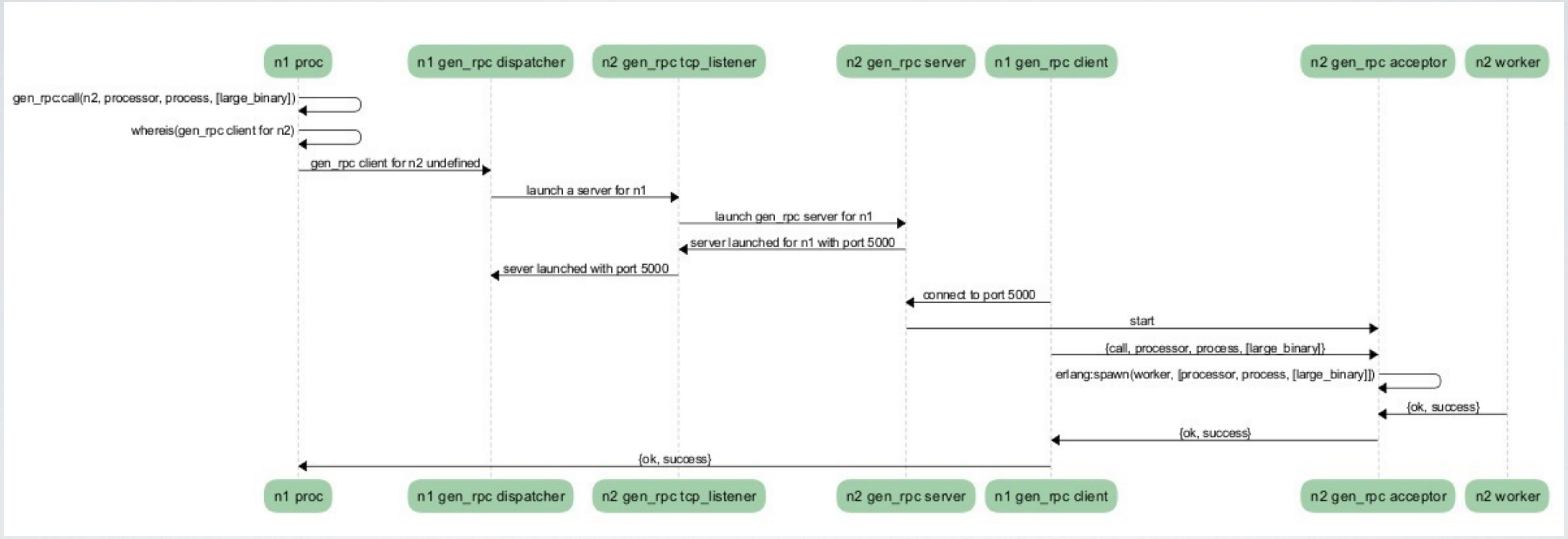


THE FIX: GEN_RPC

Features

- Uses separate TCP connections/mailboxes for each node for data transfer
- Does not block the VM's distributed port with large payloads
- No distributed Erlang dependency
- Offers connection state feedback

ARCHITECTURE



INTERNALS

- Named processes help track usage and mailbox issues
- Unidirectional connections support high performance communication
- Using TCP server for messaging instead of Distributed Erlang allows communication over insecure channels

INTERNALS

- Protected calls (a la RPC) shield socket owners from misbehaving workers
- Responses compatible with RPC
- Uses inet_async for acceptor handoff

PERFORMANCE

- Simple RPC scaled up to ~50K calls/sec/node before running into mailbox issues
- Remote spawn scaled to ~100K calls/sec/node before Mnesia started acting out
- gen_rpc currently handles > 150K calls/sec/node

SHORTCOMINGS

- Single client and acceptor mailbox per node pair
- Does not work with anonymous functions across nodes (VM limitation)
- Not as fast aware of TCP failures as Distributed Erlang
- That's it!

LEARNING ELIXIR FOR FUN AND FUN

GEN_RPC IN ELIXIR: EXRPC

- Elixir is fun!
- Wrote gen_rpc in Elixir to get acquainted with the language
- Engineering Erlang code =/= Engineering Elixir code

FEATURES

(Used to be) exactly the same as **gen_rpc**!

THE GOOD

- Elixir allows more concise code (:a **in** [:a, :b, :c])
- No boilerplate code for GenServer and friends
- Modern build and testing tools
- Interoperability with existing Erlang projects
- Documentation is a first class citizen
- All Erlang VM features (i.e slave nodes) are supported
- Transparently use ExRPC and gen_rpc between Erlang and Elixir nodes

THE BAD

- With great power sometimes comes not so great performance
- Elixir BEAMs need to run in the OTP release they've been compiled in
- Testing framework not as powerful as CT yet
- Running Dialyzer needs a Mix plugin (i.e dialyxir)

PERFORMANCE CONSIDERATIONS

Magic always comes with a price:

Erl:

```
1> timer:tc(fun() -> [ok || _ <- lists:seq(1, 5000000)] end).
{1399324, …}
```

IEx:

```
iex(1)> :timer.tc(fn() -> for _ <- :lists.seq(1, 5000000), do: :ok end)
{4765819, …}
```

But:

```
iex(1)> :timer.tc(fn() -> :lists.seq(1, 5000000) |> Enum.map(fn(_) -> :ok end) end)
{2034507, …}
```

(INCREDIBLY OVERENGINEERED)

DEMO

THE FUTURE

- SSL support (including CN verification)
- Client connection separation by node **and arbitrary ID** (thanks Erlang mailing list!)
- Blacklisting/whitelisting modules available for RPC
- Configurable server port allocation for strict firewalls

Thank you!

https://hex.pm/packages/**gen_rpc**
https://github.com/**priestjim/gen_rpc**
https://github.com/**priestjim/exrpc**

https://github.com/**priestjim**
https://linkedin.com/in/**priestjim**
@priestjim