Inter-service messaging

Homogenous landscape

Homogenous landscape

Examine two out of the box tools

Peek into their limitations

Outlook on scaling out

Homogenous landscape

Examine two out of the box tools

Peek into their limitations

Outlook on scaling out

Homogenous landscape

Examine two out of the box tools

Peek into their limitations

Outlook on scaling out

What is in the box?

WEIRDEST UNBOXING EVER

378,319 views

7.6K 3.3K SHARE

Dexbonus
Published on Sep 6, 2014

SUBSCRIBE 264K

remote procedure call

:rpc

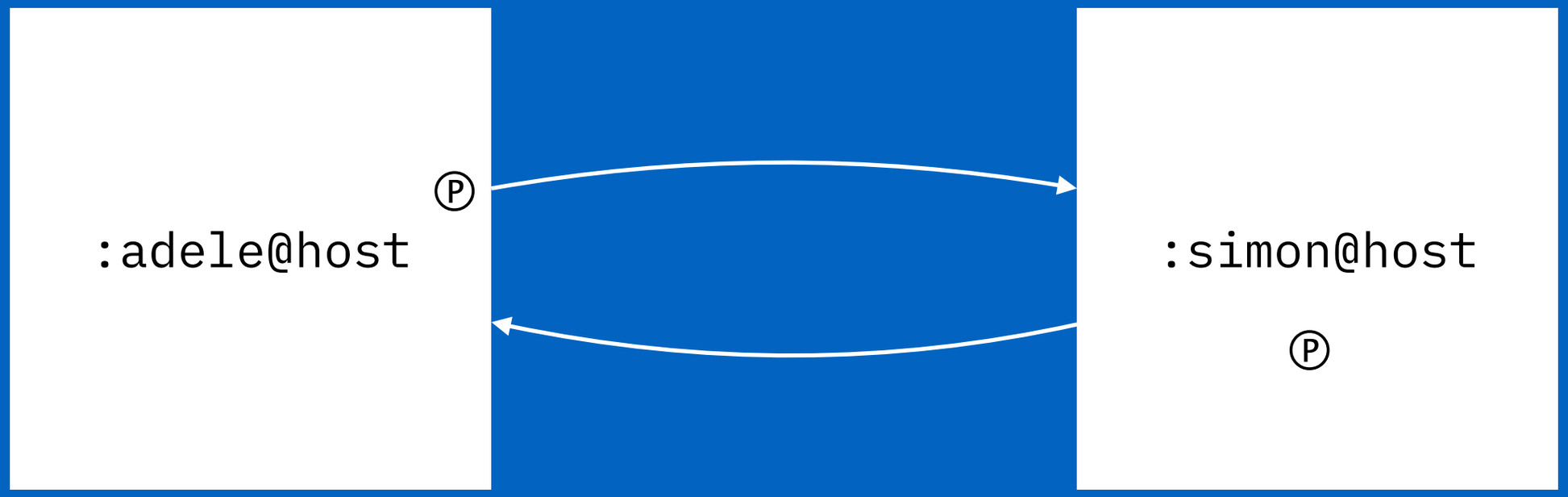
Ships with Erlang/OTP

`:rpc`

Sort of like `apply/3`

but remotely

`:rpc`



`spawn_monitor` *a middle-man process*

`:gen_server.call` *to* `{:rex, :simon@host}`

on `:simon@host` `spawn_monitor`*'s another process*

return the result to rex

:rpc

It can do very nice things for you

:rpc

Promises

`:rpc.async_call/4`

`:rpc.yield/1`

`:rpc.nb_yield/1`

:rpc

M-m-m-multicall

:rpc.multicall/4

:rpc

Fire and forget

`:rpc.cast/4`

`:rpc`

You can implement romantic-stupid-things

:rpc

```elixir
defmodule Adele do

  def hello(module, fun, args) do
    useable_node = Node.list()
                   |> Enum.filter(fn
                     n -> :rpc.call(n, Code, :ensure_compiled?, [module])
                   end)
                   |> Enum.find(fn n ->
                     call_args = [module, fun, Enum.count(args)]
                     :rpc.call(n, Kernel, :function_exported?, call_args)
                   end)
    case useable_node && :rpc.call(useable_node, module, fun, args) do
      nil -> nil
      {:badrpc, _} -> nil
      result -> result
    end
  end

end
```

```elixir
defmodule Adele do

  def hello(module, fun, args) do
    useable_node = Node.list()
                   |> Enum.filter(fn
                     n -> :rpc.call(n, Code, :ensure_compiled?, [module])
                   end)
                   |> Enum.find(fn n ->
                     call_args = [module, fun, Enum.count(args)]
                     :rpc.call(n, Kernel, :function_exported?, call_args)
                   end)
    case useable_node && :rpc.call(useable_node, module, fun, args) do
      nil -> nil
      {:badrpc, _} -> nil
      result -> result
    end
  end

end
```

```elixir
defmodule Adele do

  def hello(module, fun, args) do
    useable_node = Node.list()
                   |> Enum.filter(fn
                     n -> :rpc.call(n, Code, :ensure_compiled?, [module])
                   end)
                   |> Enum.find(fn n ->
                     call_args = [module, fun, Enum.count(args)]
                     :rpc.call(n, Kernel, :function_exported?, call_args)
                   end)
    case useable_node && :rpc.call(useable_node, module, fun, args) do
      nil -> nil
      {:badrpc, _} -> nil
      result -> result
    end
  end

end
```

```elixir
defmodule Adele do

  def hello(module, fun, args) do
    useable_node = Node.list()
                   |> Enum.filter(fn
                     n -> :rpc.call(n, Code, :ensure_compiled?, [module])
                   end)
                   |> Enum.find(fn n ->
                     call_args = [module, fun, Enum.count(args)]
                     :rpc.call(n, Kernel, :function_exported?, call_args)
                   end)
    case useable_node && :rpc.call(useable_node, module, fun, args) do
      nil -> nil
      {:badrpc, _} -> nil
      result -> result
    end
  end

end
```

Spawn a new process on node.

`:rpc`

Can be linked using `Node.spawn_link/4`

`:rpc`

Limitations

Security?

All of this assumes a private network!

There is only one "rex" RPC server per node!

`:rpc`

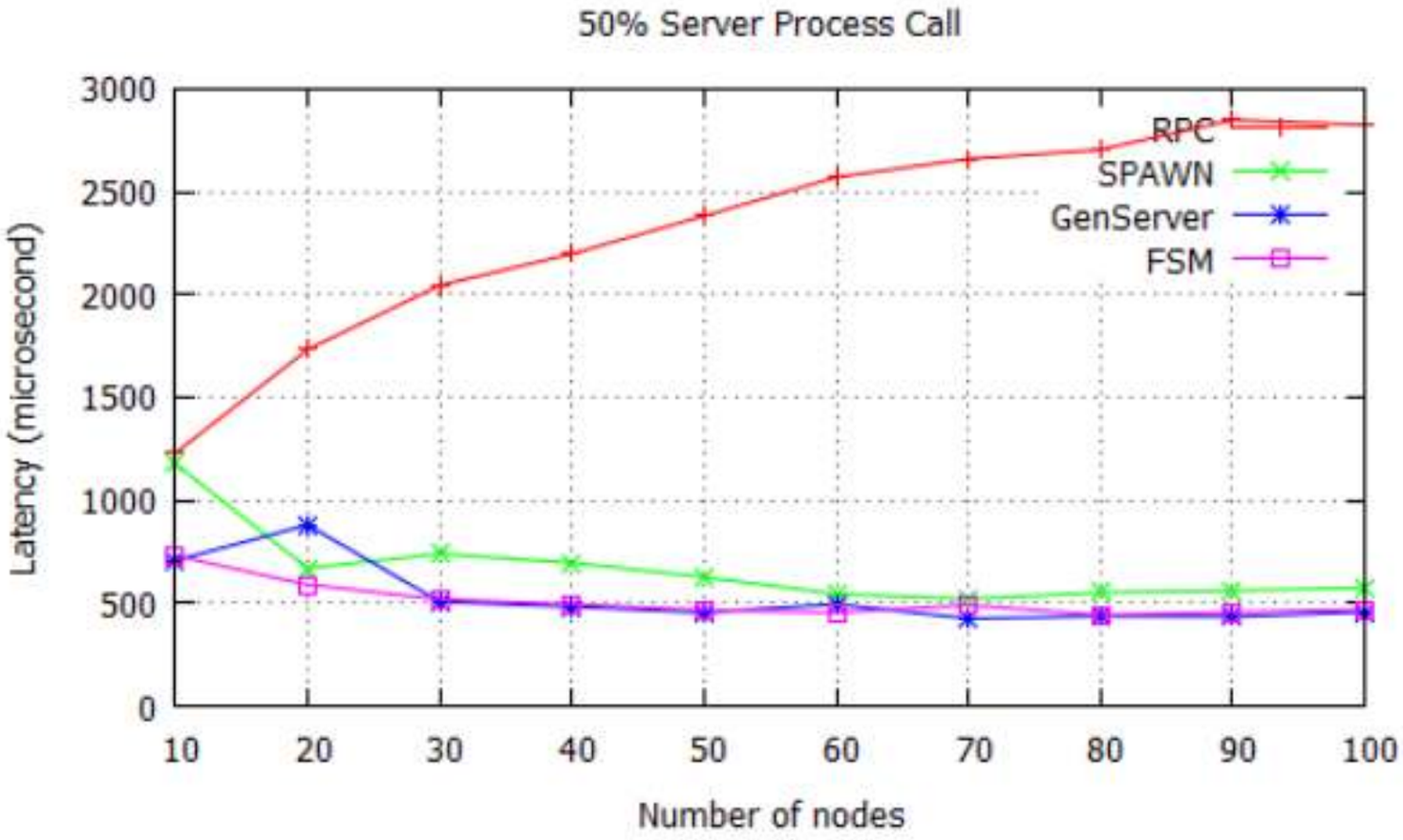


Figure 13: Latency of Commands for 50% Server Process Call

Make `:rpc`-calls obvious to other developers!

`:rpc`

Size matters!

n^2 *chatty connections*

Size matters!

You are fine < 50 nodes

>= 50 nodes

You will have to tame the fully meshed network

Large messages can clog the BeamVM, and thus mess with Mnesia and :global synchronisation.

Much of the problems can be attributed to the fact that the fully meshed network scales badly.

Outlook

`Node.spawn/4` *can replace* `:rpc.async_call/4` *solving the rex mailbox flooding issue for small scale networks.*

:gen_rpc

Solves security by providing SSL connections

:gen_rpc

Solves connection overflow by providing its own

:gen_rpc

Solves mailbox-flooding

:gen_rpc

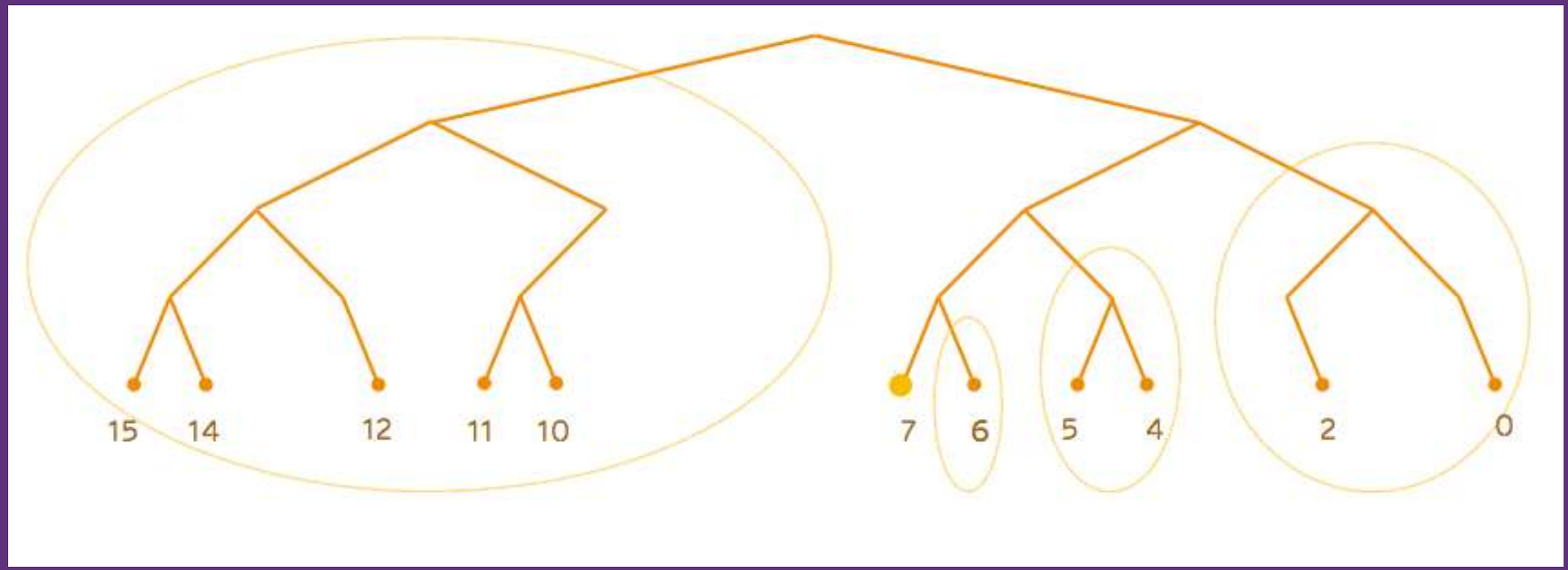
https://github.com/priestjim/gen_rpc

:gen_rpc

Kademlia routing tables

Restructures the cluster connections into a tree

Kademlia



Kademlia

IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL. 28, NO. 8, AUGUST 2017

Evaluating Scalable Distributed Erlang for Scalability and Reliability

Natalia Chechina, Kenneth MacKenzie, Simon Thompson, Phil Trinder, Olivier Bo
Viktória Fördős, Csaba Hoch, Amir Ghaffa

Abstract—Large scale servers with hundreds of hosts and tens of thousand
platforms softwa must be both scalable and reliable, and distributed actor l
While distribut conceptually supports the engineering of large scale
that force dev the standard language mechanisms at s
limitations, a a Scalable Distributed (SD) Erla
Machines (This paper presents the firs
associated mprehensive ev
three typi oups in
applicati
but scal
Erlang
Erlang

Ind

P. Trinder, N. Chechina, N. Papaspyrou, K. Sagonas, S. Thompson, *et al.* • Scaling Reliably • 2017

Scaling Reliably: Improving the Scalability of the Erlang Distributed Actor Platform

CHINA, NIKOLAOS PAPASPYROU,
IPSON, STEPHEN ADAMS, STAVROS ARONIS,
BOUDEVILLE, FRANCESCO CESARINI,
ON, VIKTORIA FORDOS, AMIR GHAFFARI,
CSABA HOCH, DAVID KLAFTENEGGER,
H MACKENZIE, KATERINA ROUKOUNAKI,
KJELL WINBLAD

EP (287510) project
project.eu/

2017

ct

tructing scalable reliable systems, and the Erlang
tial model. While the Erlang model conceptually
limits and these force developers to depart from
mits of Erlang systems, and reports the work of
derstandability of the Erlang reliable distributed

nd then address the issues at the virtual machine,
olved the Erlang virtual machine so that it can
LIMA architectures. We have made important
Erlang/OTP release. (2) We have designed and
address language-level scalability issues, and
e constructs. (3) To make large Erlang systems
ade open source releases of five complemen

te the capabil

Investigating the Scalability Limits of Distributed Erlang

Amir Ghaffari
The University of Glasgow
School of Computing Science, G12 8QQ, UK
Amir.Ghaffari@glasgow.ac.uk
http://www.release-project.eu

Abstract

With the advent of many-core architectures, scalability is a key property for programming languages. Actor-based frameworks like Erlang are fundamentally scalable, but in practice they have some scalability limitations.

The RELEASE project aims to improve the scalability of Erlang on emergent commodity architectures with 10^5 cores. This paper investigates the scalability limits of distributed Erlang on up to 150 nodes by using *DE-Bench*. We discuss the design and implementation of DE-Bench, a scalable peer-to-peer benchmarking tool for distributed Erlang.

Our benchmarking results demonstrate that the frequency of global commands limits the scalability of distributed Erlang. There is also a common belief that distributed Erlang does not scale on large architectures with hundreds of nodes and thousands of cores. We provide evidence against this belief and show that distributed Erlang scales linearly up to 150 nodes and 1200 cores with relatively heavy data and computation loads when no global command is made.

Measuring the

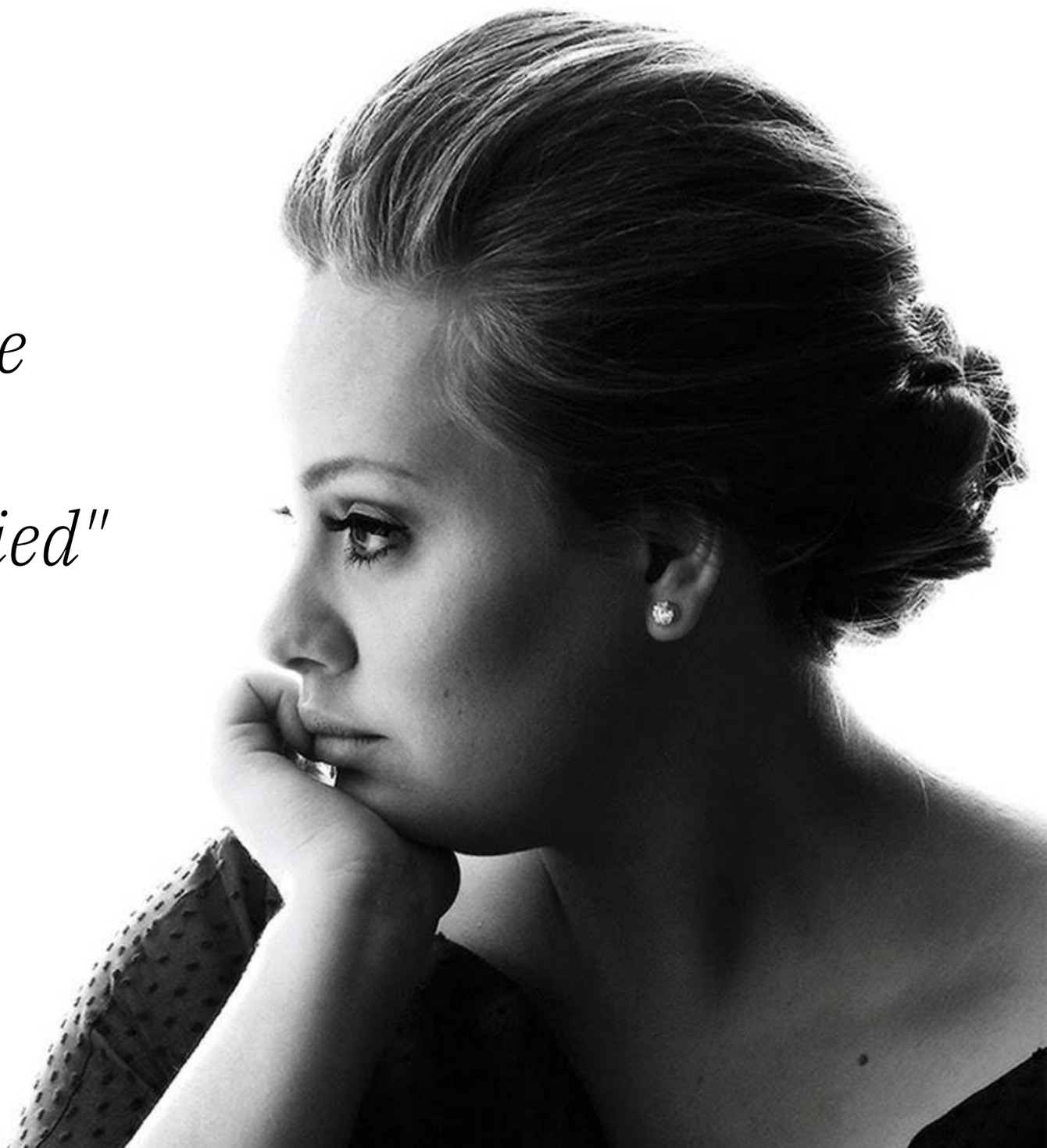
Recently, Erlang has become a popular platform to develop large-scale distributed applications, e.g. *Facebook chat backend, T-Mobile advanced call control services, Ericsson AXD301 ATM switch,* and *Riak DBMS* [2], [3]. This popularity is due to a combination of factors, including data immutability, share-nothing concurrency, asynchronous message passing based on the actor model, process's location transparency, and fault tolerance [4].

However, in practice the scalability of Erlang is constrained by aspects of the language and virtual machine [5]. For instance, our measurement shows that Riak 1.1.1 does not scale beyond 60 nodes because of overloaded single supervisor processes [6].

The RELEASE project aims to scale the Erlang's radical concurrency-oriented programming paradigm to build reliable general-purpose software, such as server-based systems, on emergent commodity architectures with 10^5 cores [5]. The RELEASE consortium works to scale Erlang at the virtual machine level, infrastructure

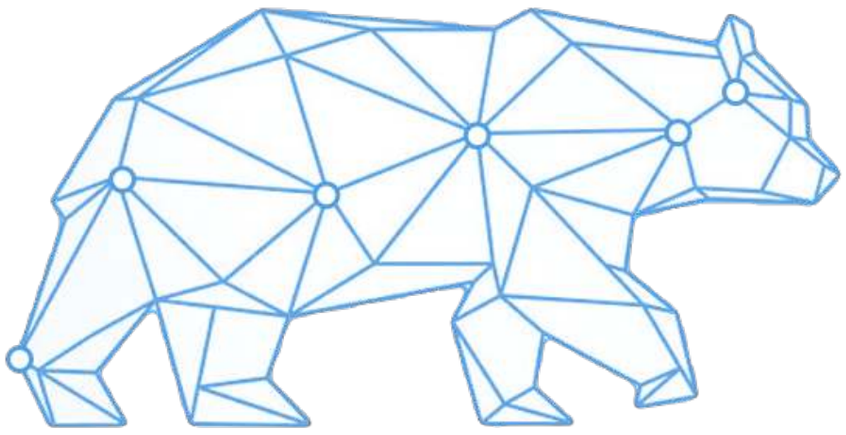
"Hello from the other side
I must have called a
thousand times"

"But when I call,
you never seem to be home
Hello from the outside
At least I can say that I've tried"



EOP

@Overbryd



May 8, 2018
Metabase UG Berlin

- http://www.dcs.gla.ac.uk/~amirg/publications/DE-Bench.pdf
- https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7820204
- http://www.dcs.gla.ac.uk/research/sd-erlang/release-summary-arxiv.pdf
- http://s3.amazonaws.com/erlang-conferences-production/media/files/000/000/074/original/Zandra_Norman_ScalingDistributedErlang.pdf
- http://learnyousomeerlang.com/distribunomicon
- "The fallacies of distributed computing" by Arnon Rotem-Gal-Oz
  http://www.rgoarchitects.com/Files/fallacies.pdf
- "Erlang Factory SF 2016 - Panagiotis Papadomitsos - Scaling RPC Calls In Erlang And Elixir"
  Talk: https://www.youtube.com/watch?v=xiPnLACtNeo
  Slides: http://www.erlang-factory.com/static/upload/media/145933737266372 8panagiotispapadomitsosimprovingrpccallsinerlangandelixir.pdf