

# What's new in Gradualizer?

## Gradually typing Erlang and Elixir

Radek Szymczyszyn

@erszcz

Tech Lead at Erlang Solutions

[radek.szymczyszyn@erlang-solutions.com](mailto:radek.szymczyszyn@erlang-solutions.com)



# Agenda

- Who am I?
- Statically typing Erlang: state of the art
- Quantitative comparison
- Demo: TypedServer in Elixir with Gradient
- Qualitative comparison on selected examples
- Conclusions
- What's new? What's left to do?
- Contributing

# Who am I?



- Radek Szymczyszyn
  - @erszcz on GitHub / Medium / Twitter
- Tech Lead at Erlang Solutions Kraków
- Static type system enthusiast
  - Gradualizer core team member
  - Co-creator of Gradient
- Believer in sleek and good quality docs
  - Co-author of Erlang Enhancement Proposal 48 aka EEP-48
  - EEP-48 implementer in EDoc - merged into Erlang/OTP 24.0
    - This was the groundwork for using ExDoc with Erlang
  - Author of docsh, a prototype of a doc system for the Erlang shell
- Instant messaging expert
  - XMPP engineer
  - MongooselM core team member (recently not very active)

# **Statically typing Erlang: State of the Art**

# Statically typing Erlang: state of the art

## S. Marlow and P. Wadler

- Solving unification constraints of the subtyping form  $U \subseteq V$ , unlike Hindley-Milner systems solving constraints of form  $U = V$
- Type inference was slow, types large and complex, sometimes wrong results due to pattern matching
- In general, the results were not satisfactory. Not used in practice

# Statically typing Erlang: state of the art

## Dialyzer, T. Lindahl and K. Sagonas

- Success typings, not a Hindley-Milner type system
- *Dialyzer is never wrong*
  - It assumes the implementation is right
  - It does not depend on extra annotations (specs), but it compares its inference results with them
- The most mature Erlang type checker, maintained by the OTP team, widely adopted by the community
- Popularised the use of `-spec` and `-type` attributes

# Statically typing Erlang: state of the art

## WhatsApp/erlt

- Introduces new syntax, so it's actually a new language, not Erlang
- Optional static typing, uses specs, allows for unchecked code
- Discontinued by WhatsApp
  - But WhatsApp is not done with type checking Erlang - stay tuned!

# Statically typing Erlang: state of the art

ETC, N. Valliappan and J. Hughes

- An experiment in applying a Hindley-Milner type system to Erlang
- Enabling overloaded constructors
- Partial evaluation to improve accuracy
- Requires code modifications to existing programs,  
but does not require specs as it relies on type inference



# Statically typing Erlang: state of the art

ETC, N. Rajendrakumar and A. Bieniusa

- Bidirectional typing
- Requires some specs to support higher-ranked polymorphism
- Uses theory by J. Dunfield and N. Krishnaswami successfully applied in languages like PureScript, Hackett, Discus

# Statically typing Erlang: state of the art

## Gradualizer and Gradient

- Created by Josef Svenningsson (now at Meta / Facebook, not working on Gradualizer)
- Bidirectional typing
- Opt-in: no specs, no checks
- Gradual - any type information is better than no type information
- No type checking of message passing
- Still experimental, e.g. no constraint solver yet, so no polymorphism support
- Gradient is an Elixir frontend to Gradualizer

# **Quantitative comparison**

# Statically typing Erlang: quantitative comparison

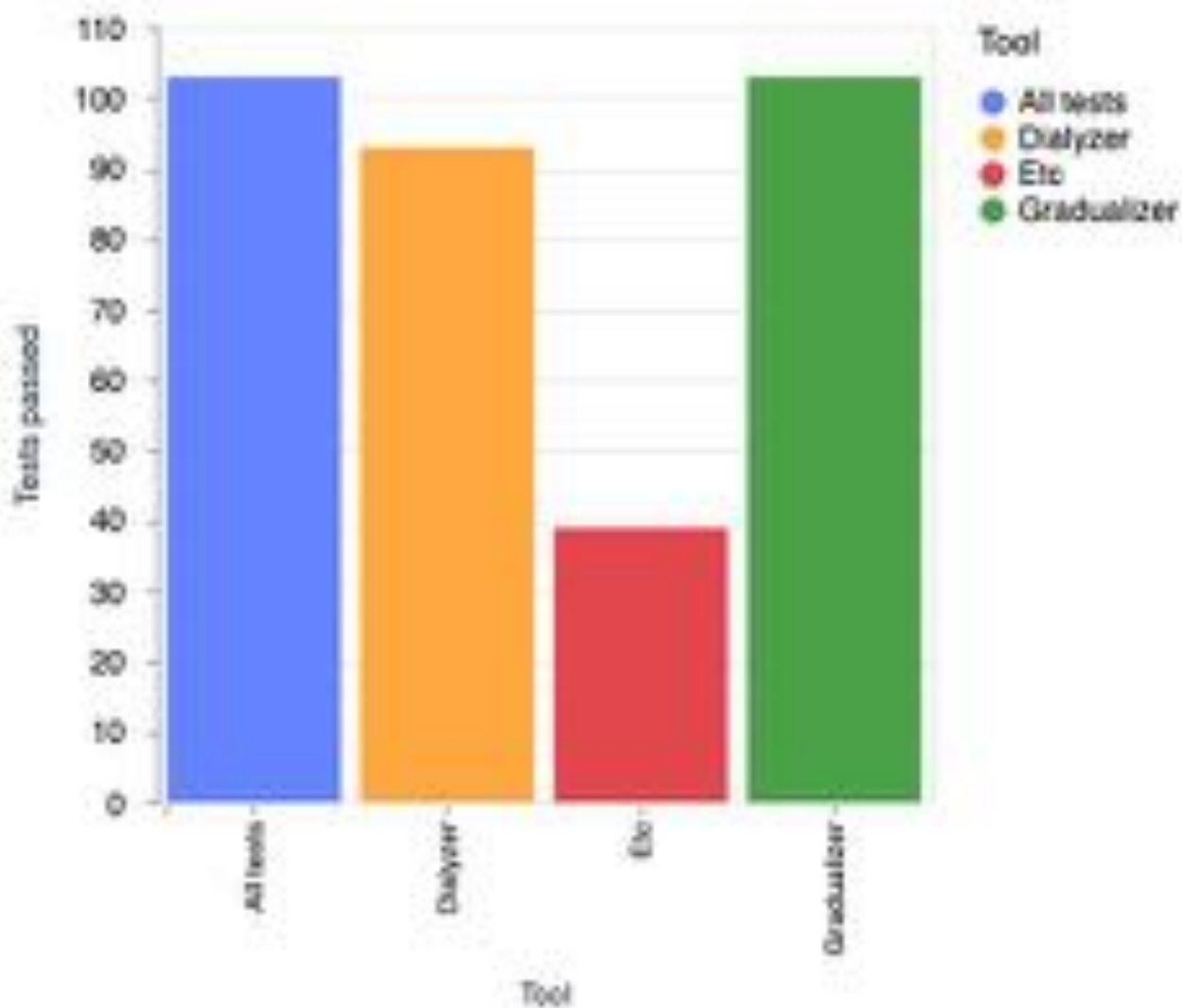
- Selected for comparison:
  - Dialyzer
  - ETC (by N. Rajendrakumar and A. Bieniusa)
  - Gradualizer
- Gradualizer has a suite of tests divided into categories:
  - `test/should_pass`
  - `test/known_problems/should_pass`
  - `test/should_fail`
  - `test/known_problems/should_fail`
- Comparison data and more details available at <https://github.com/erszcz/erlang-type-checker-comparison>

# Should pass

This chart depicts the number of tests which should pass, i.e. should type check. After all, **we do not want our type checkers to raise warnings about valid code**. None of the type checkers should report warnings here.

Dialyzer reports only a few errors. Gradualizer, as expected, reports none. ETC fares quite poorly, passing only 1/3 of the tests, which suggests it might not cover the complete Erlang syntax.

Higher is better.

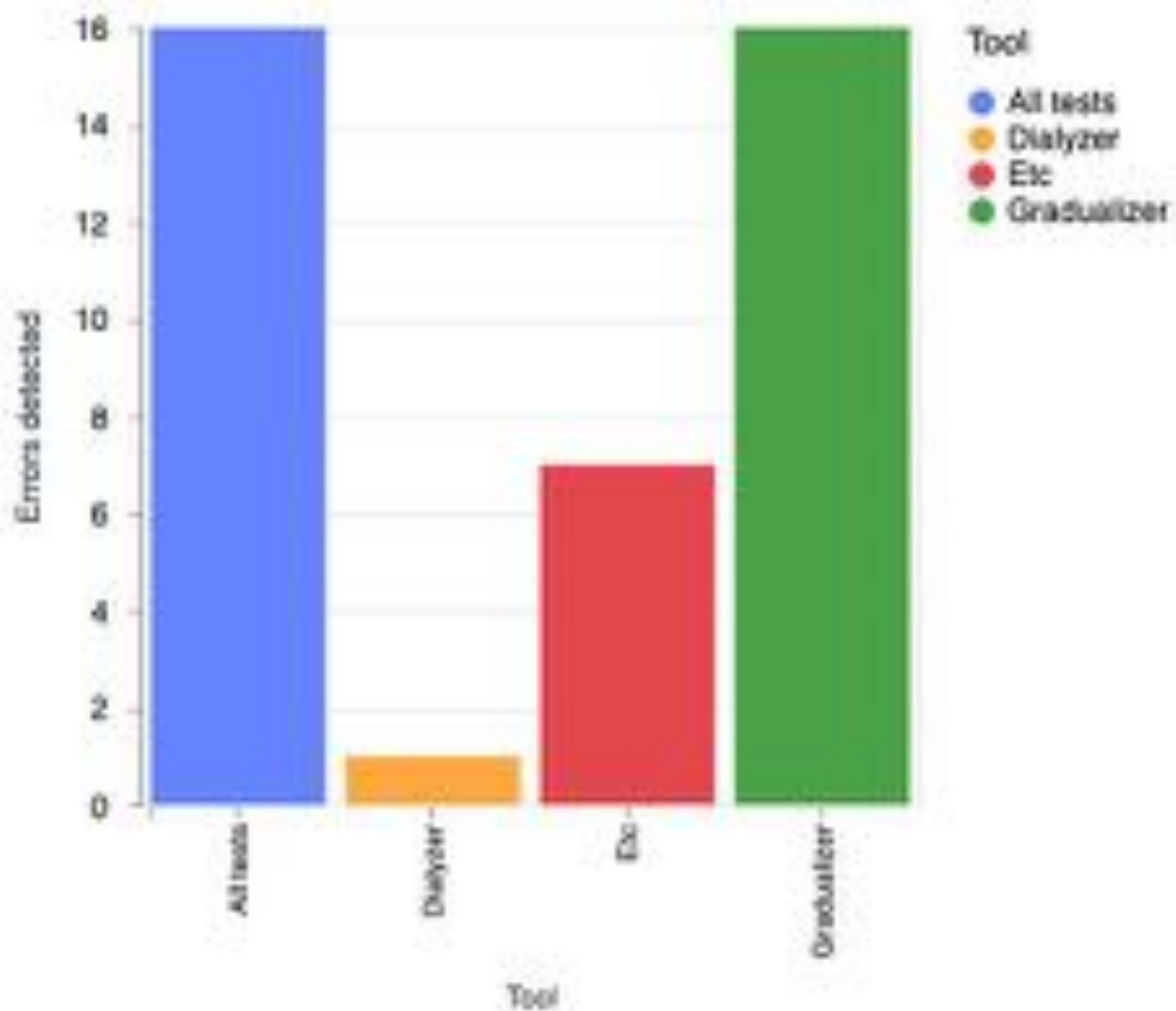


## Known problems: should pass

This chart depicts the number of tests which should pass, i.e. should type check, yet raise Gradualizer errors. **These are the false positives - invalid and misleading reports about non-issues.**

Dialyzer reports only a single false positive and is a clear winner here! This confirms the slogan that *Dialyzer is never wrong*. ETC reports some of the errors, but not all of them. Gradualizer, as expected, reports errors in all of the tests.

Lower is better.

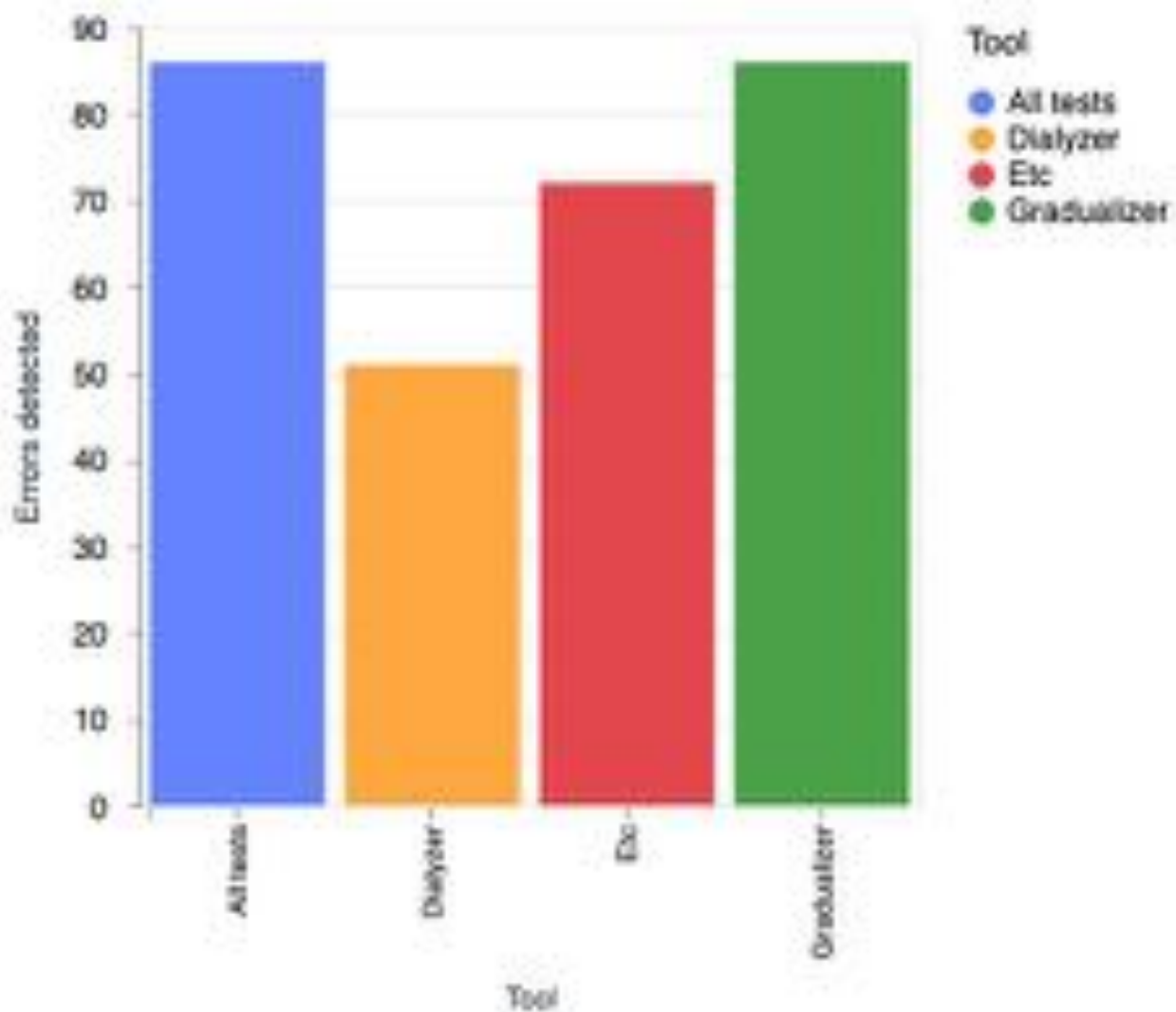


# Should fail

This chart depicts the number of tests which should fail, i.e. should not type check. **These tests check that the warnings we want to see in our buggy code are actually generated.**

Dialyzer seems to be somewhat permissive. ETC reports errors in the majority of tests. Gradualizer reports errors in all the tests.

Higher is better.

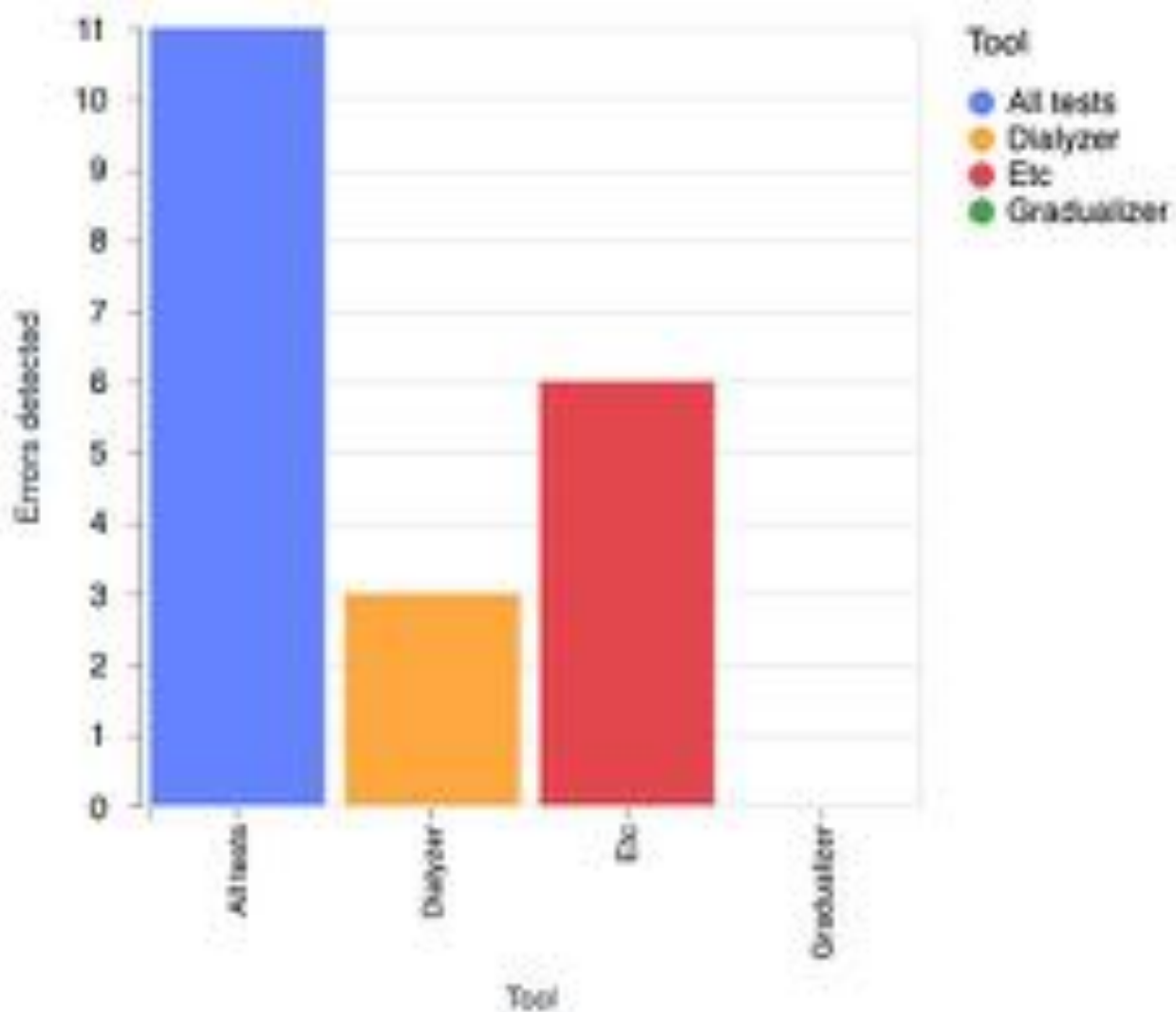


## Known problems: should fail

This chart depicts the number of tests which should not type check, but are known to type check with Gradualizer. In other words, **these are the errors that type checkers cannot find**. We have to rely on other techniques to find them.

Dialyzer detects some of the errors in these examples, ETC seems to detect even more. Gradualizer doesn't detect any of them, but the examples are crafted against this type checker, so it's expected.

Higher is better.

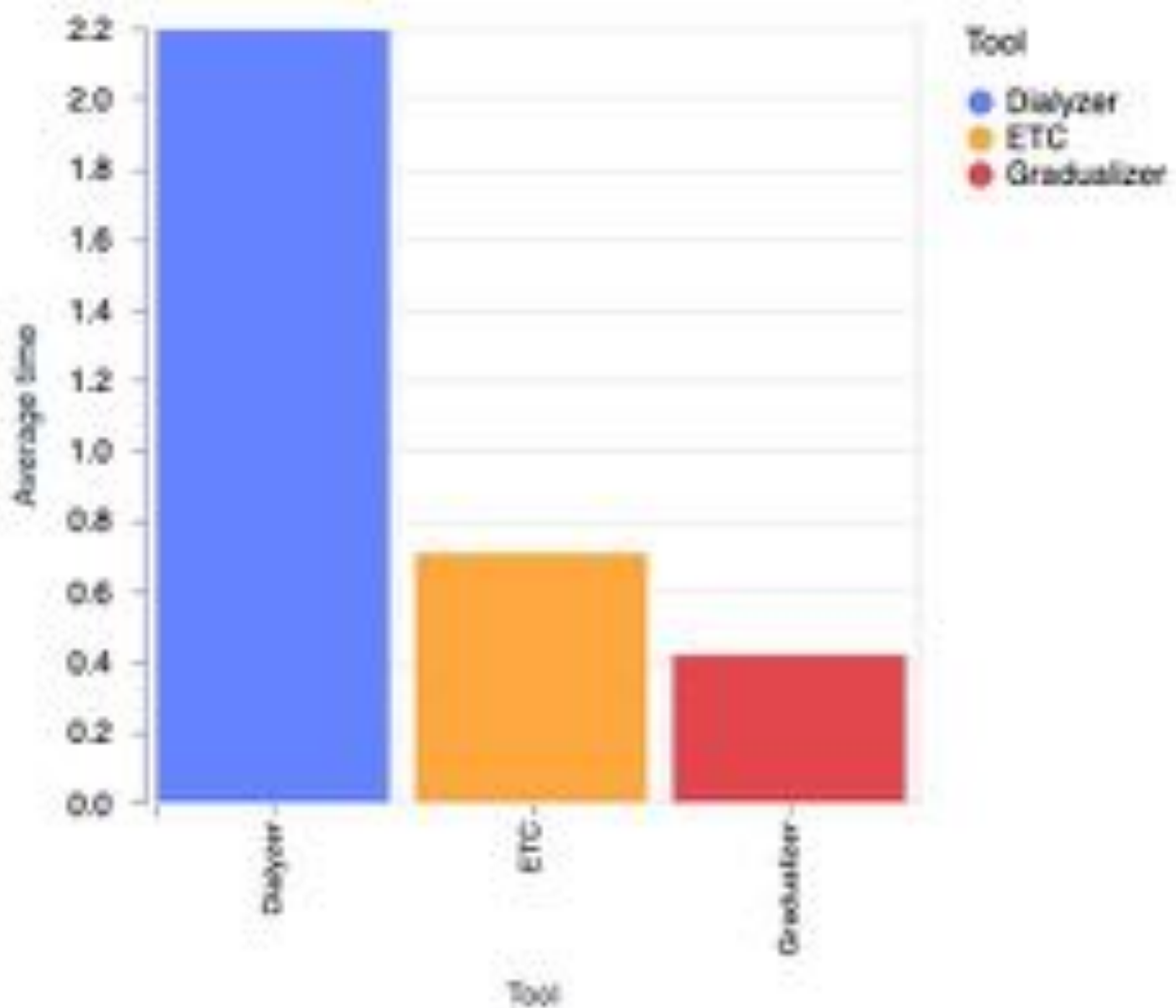




# Runtime performance

Runtime performance averaged over all the `should_pass` / `should_fail` / `known_problems` tests.

Lower is better. Time is in seconds.



**Demo:**  
**TypedServer in Elixir with**  
**Gradient**

# **Qualitative comparison on selected examples**

# Some issues are serious - type soundness

```
%% file: should_pass/andalso_any.erl
```

```
-spec f1() -> boolean().
```

```
f1() ->
```

```
    true andalso g1().
```

```
-spec g1() -> any().
```

```
g1() -> 3.
```

Gradualizer is fine with the above code due to `any()` being the dynamic type. Dialyzer reports that `f1/0` returns 3, which is not `boolean()`. And it's right!

## Type soundness issue in Gradualizer!

Example:

```
> andalso_any:f1() andalso 5.
```

```
** exception error: bad argument: 3
```

## Some issues are trivial - unexported function

```
% file: should_pass/binary_exhaustiveness_checking.erl
```

```
--export([f/1, g/1, h/1, k/1]).  
+-export([f/1, g/1, h/1, k/1, l/1]).
```

Dialyzer reported `l/1` would never be called. That was the case as it was not exported. **Exporting `l/1` fixes the issue.**

# Dialyzer is better at type inference...

```
%% file: should_pass/return_fun.erl
```

```
-spec return_fun_no_spec() -> integer().  
return_fun_no_spec() -> fun no_spec/0.
```

```
no_spec() -> ok.
```

Dialyzer:

The success typing is  
    () -> fun(() -> 'ok')

Gradualizer sees no evil, unless we pass use the --infer flag:

```
return_fun.erl:29:25: The fun expression is expected to have type integer()  
but it has type fun(() -> any())
```

ETC - crashes!

## ...but at other times is quite vague

```
%% file: should_fail/branch2.erl
```

```
-spec c(boolean()) -> integer().
```

```
c(true) ->
```

```
    1;
```

```
c(false) ->
```

```
    apa.
```

Dialyzer, with no options, does not detect the type mismatch. With `-Wspecdiffs` it reports:

```
branch2.erl:5:2: The success typing for branch2:c/1 implies that the function might  
also return
```

```
    'apa' but the specification return is  
    integer()
```

Which is expected, but a bit vague. It might be hard to use in CI - should a build fail or not when a warning is emitted?

Gradualizer by default reports an error - the atom `apa` is not an `integer()`.

# Exhaustiveness checking

%% Example from “Bidirectional Typing for Erlang”

```
-spec foo2(integer()) -> {}.
foo2(1) -> {};
foo2(42) -> {}.
```

Dialyzer with -Wunderspecs or with -Wspecdiffs:

```
bdtfe3_should_fail.erl:8:2: Type specification bdtfe3_should_fail:foo2
    (integer()) -> {} is a supertype of the success typing:
bdtfe3_should_fail:foo2
    (1 | 42) -> {}
```

Gradualizer reports:

```
bdtfe3_should_fail.erl:9:1: Nonexhaustive patterns: 0
```



# Exhaustiveness checking

```
%% file: should_fail/exhaustive_float.erl
```

```
-type t() :: {int, integer()}  
           | {float, float()}.
```

```
-spec ef(t()) -> ok.
```

```
ef(T) ->  
    case T of  
        {int, _} -> ok  
    end.
```

Dialyzer does not detect a missing case clause if run with no options, with `-Wunderspecs`, or with `-Woverspecs`. With `-Wspecdiffs` it reports:

```
exhaustive_float.erl:8:2: Type specification exhaustive_float:ef  
    (t()) -> 'ok' is not equal to the success typing: exhaustive_float:ef  
    ({'int', _}) -> 'ok'
```

Gradualizer reports: Nonexhaustive patterns: {float, -1.0}

# Overspecs

`% Example from “Bidirectional Typing for Erlang”`

```
-spec lookup(T1, [{T1, T2}]) -> (none | T2).  
lookup(_, []) -> none;  
lookup(K, [{K, V}|_]) -> V;  
lookup(K, [_|KVs]) -> lookup(K, KVs).
```

```
find() ->  
    "s" = lookup(0, [{0, 1}]).
```

Dialyzer with `-Wspecdiffs` or with `-Woverspecs`:

```
bdtfe1.erl:9:2: Type specification bdtfe1:lookup  
    (T1, [{T1, T2}]) -> 'none' | T2 is a subtype of the success typing:  
bdtfe1:lookup  
    (_, maybe_improper_list()) -> any()
```

Gradualizer still lacks a constraint solver, so it cannot find the polymorphic type error.  
ETC properly reports a unification error.

# Conclusions

## Dialyzer

- `-Wspecdiffs`  $\approx$  `-Wunderspecs` `-Woverspecs` - good, we should be able to detect a lot (all? more?) stuff than Gradualizer does by default
- `-Wunderspecs` - usually, but not always, detects the same things as Gradualizer's exhaustiveness checking, likely real bugs, so it should fail CI
- `-Woverspecs` - reports useful warnings, though we often see that Dialyzer ignores the extra information we pass in the spec (it assumes the implementation is correct, not the spec) - we want to see them, but not fail CI
- Run Dialyzer twice with both options respectively?  
But it's slow already...
- Check out Code BEAM Stockholm 2022 talks about Dialyzer by Jesper Eskilson and Thomas Davies - lots of good stuff!

# Conclusions

## Gradualizer

- Good coverage of Erlang syntax and constructs
- Already useful in practice
- Fast
- There are still known problems to solve
  - False positives are the most annoying
  - But some problems are low hanging fruit (e.g. `known_problems/should_fail/arith_op.erl`)
- No constraint solver - no warnings if polymorphism in play

# Conclusions

## ETC

- Uses a theory successfully tried and proved, e.g. in PureScript
- Crashes/fails on a lot of valid, ordinary code
- Does find bugs in polymorphic code the other checkers cannot
- Not fit for purpose yet :(

# Conclusions

More examples are described at

<https://github.com/erszcz/erlang-type-checker-comparison>

# What's new, in-progress, on the roadmap?

- Done \o/
  - Most (all?) exhaustiveness checking bugs squashed!
- InProgress:
  - Make it self-gradualize to prove a moderate size project can be Gradualizer-clean (now down to 9 warnings)
    - The rest are limitations or bugs in the type checker - more work ahead!
  - Fix the issues found when cross-checking tests with Dialyzer and ETC
  - Find new `known_problems`, fix existing `known_problems`
- ToDo:
  - Constraint solver to properly handle type variables!

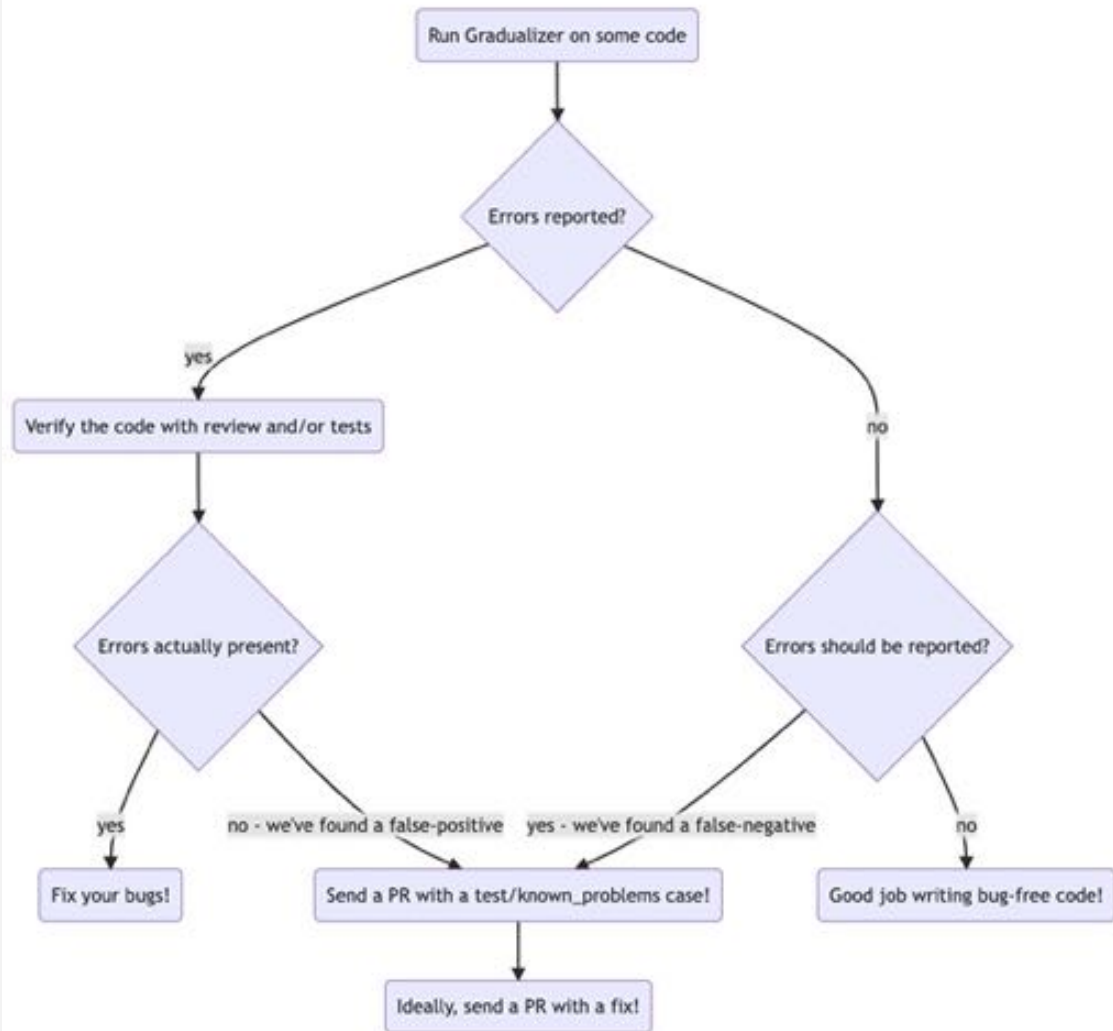
# What's new, in-progress, on the roadmap?

- Done \o/
  - Complete support for inline type annotations/assertions (builtin, user, and remote types)
- InProgress:
  - Ignore selected warnings (thanks Victor Rodrigues!)
  - Fix (ignore?) warnings related to Elixir constructs not present in Erlang
    - Dot-access
    - `with` keyword
- ToDo:
  - Escript to run as a standalone tool



# Contributing

There's still low hanging fruit in both projects, so if you have some spare time...



# Thank you!

[Viktor Söderqvist](#)

[Przemek Wojtasik](#)

[Victor Rodrigues](#)

[Eksperimental](#)

[Pedro Miguel Pereira Serrano Martins](#) aka FI4m3Ph03n1x

And all the people who reported issues, contributed ideas, examples, or code, and made effort to make any of the Erlang type checkers better!

# Thanks for joining!

## Questions?

Radek Szymczyszyn

@erszcz

Tech Lead at Erlang Solutions

[radek.szymczyszyn@erlang-solutions.com](mailto:radek.szymczyszyn@erlang-solutions.com)



## We are hiring!

### Elixir Developers

- Technology stack: Elixir, Phoenix, OTP, SQL, Linux, Git

### Erlang Developers

- Technology stack: Erlang, RabbitMQ, OTP, Kubernetes, SQL, Linux, Git

CHECK out open positions on  
[www.erlang-solutions.com/careers](http://www.erlang-solutions.com/careers)

Are you an expert in a different  
technology but would like to start a  
career  
in Erlang or Elixir?  
We are open to talk! 😊

**100% remote /  
Cracow office**

Various sectors,  
challenging projects:

- Fintech & Blockchain
- Telecommunications
- Healthcare
- Gaming



# INSTRUCTIONS FOR USE

Before starting to edit this document, go to File and select Make a copy.

## EDIT IN GOOGLE SLIDES

You will get a copy of this document on your Google Drive and will be able to name it, edit, add or delete slides.

You can hit the 'share' button to make it available to selected contacts for them to edit, comment or view only.

## CUSTOMISE YOUR OWN PRESENTATION

We have created a range of slides with various layouts, so browse and select the ones that suit your content.

If you have any questions or troubles, contact Magda.

# INSTRUCTIONS FOR USE

Please use the agreed  
fonts and colours:

## FONTS

Proxima is used as a substitute for the  
brand font Aktiv Grotesk in Slides.

## COLOURS

Use white for most text.  
One of the following colours can be used  
to highlight a key word or a headline.  
The colours are linked to our three service  
areas, to try to pick the most appropriate.

Space Grey		#1A1A1A
Purple	Community	#822FEB
Blue	Consultancy	#3668EB
Green	Capability	#20E89F

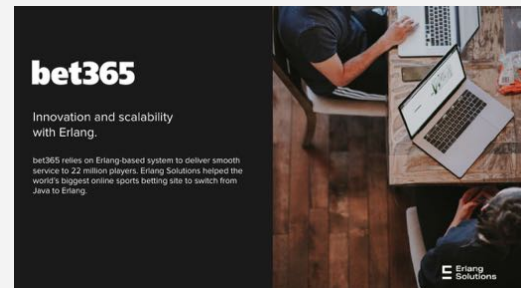
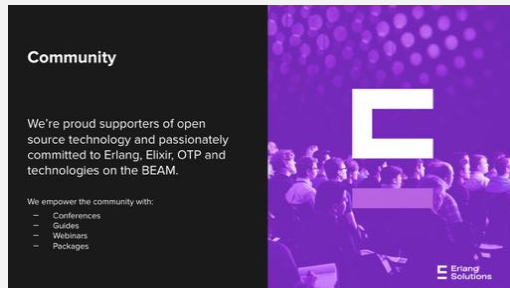
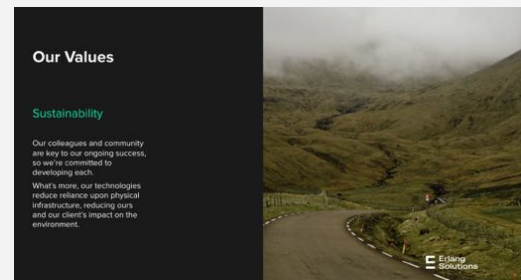
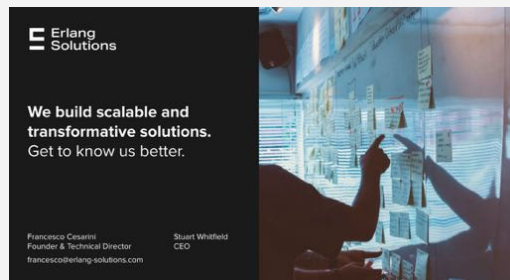
# INSTRUCTIONS FOR USE

## Images

### PHOTO LIBRARY

Marketing have built a library of stock images to use in presentations, which can be found here: (insert Drive link)

Images should be used as per the example layouts in this document. Right click an image and click 'replace image' to insert a new one into an existing layout.





# INSTRUCTIONS FOR USE

## How to reuse, add, delete slides and change slide layouts

### REUSING SLIDES

We have prepared a set of slides with various layouts for you to choose from. Many of the slides have a background or logo built in, and so you may want to copy an existing slide to base a new one on these elements.

### ADDING NEW & DELETING SLIDES

To add a new slide you can right click and duplicate one of the existing slides or on the left panel with slide thumbnails right click and choose 'Add slide'.

Once you are finished delete any unwanted template slides.

# INSTRUCTIONS FOR USE

Use only the agreed font sizes.

A specific selection of font sizes (shown on the following page) are used throughout the template to give consistency to the document and create a clear hierarchy of information on each page.

The examples throughout this document show how these should be used to adapt to different content.

**24pt Bold** for normal page headlines.

**36pt Bold** for short statement slides.

36pt Normal for longer statements.

18pt for subheadings or short body copy.

12pt for body copy/ long paragraphs.

14pt can be used instead of 12pt and 18pt where content requires. Be as consistent as possible across neighbouring slides.

# INSTRUCTIONS FOR USE

## Bullet Points

We use dashes rather than the standard dot bullet points. To the right are two styles to copy and paste to wherever you might need them.


- 90+ experts working across the Americas and Europe.
  - 20 yrs and counting working with startups to Fortune 100s.
- 
- Consulting
  - Development
  - Support
  - Code/Architecture Reviews
  - Training - F2F and remote

# INSTRUCTIONS FOR USE


## Icons

The following pages contain icons that can be used to illustrate an idea or key point.

These should be used sparingly for emphasis. To the right is one example icon layout that can be found in this template.



**Why Erlang?**



**Less effort**

- 4 to 20 times less code
- vs C / C++ / Java
- Suitable for rapid prototyping
- Powerful middleware and libraries

 Erlang Solutions

## MAIN ICON SET



Global Community



Notification



Settings



Filter



Cloud Upload



Cloud Download



Shopping Cart



Contact



Full Screen



Clock



Refresh



Brightness



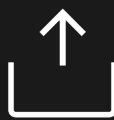
Exit



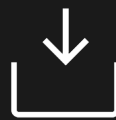
Location



Map



Upload



Download



Search

MAIN ICON SET



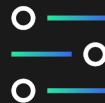
Global Community



Notification



Settings



Filter



Cloud Upload



Cloud Download



Shopping Cart



Contact



Full Screen



Clock



Refresh



Brightness



Exit



Location



Map



Upload



Download



Search

SOCIAL MEDIA SET



Products



Solutions



Industries



Training



Events



Team



Community



Ask Us



Blog



SOCIAL MEDIA SET



Products



Solutions



Industries



Training



Events



Team



Community



Ask Us



Blog

# INSTRUCTIONS FOR USE

## Tables and Graphs

This template contains some simple examples of tables and graphs using the brand colours and style. Simple graphics can be built in Slides but more complex graphs can also be pulled from Sheets.

Tables, boxes, and grids should be filled with Space Grey so the background does not show. Where possible, the shapes should align to the E grid so each edge just covers an E.

Colour can be used for emphasis or information. Try to use the three brand colours, but other colours can be used if required for more complex diagrams.

**This is a slide title**

This is an example of table styling:

	A	B	C
Yellow	10	20	7
Blue	30	15	10
Yellow	5	24	16

Erlang Solutions

**Use charts to explain your ideas**

This is an example of chart styling:

Erlang Solutions

**You can copy and paste graphs from Google Sheets**

This is an example of chart styling:

Erlang Solutions

**Show a process in simple steps**

**Step One**  
Our first stage is to research and fully understand the existing systems.

→

**Step Two**  
Then we look for ways in which to resolve any problems that arise.

→

**Step Three**  
We solve the problem with the most relevant tools and knowledge.

Erlang Solutions

**The presentation template starts  
from the following slide:**

# This is a Presentation Headline in Proxima Bold

## This is a Subheading.

Name Surname  
Title, Role, and/or Expertise  
email@erlang-solutions.com

Name Surname  
Title, Role, and/or Expertise  
email@erlang-solutions.com

# This is a Presentation Headline in Proxima Bold

## This is a Subheading.

Name Surname  
Title, Role, and/or Expertise  
email@erlang-solutions.com

Name Surname  
Title, Role, and/or Expertise  
email@erlang-solutions.com

# This is a Presentation Headline in Proxima Bold

## This is a Subheading.

Name Surname  
Title, Role, and/or Expertise  
email@erlang-solutions.com

Name Surname  
Title, Role, and/or Expertise  
email@erlang-solutions.com

# This is a Presentation Headline in Proxima Bold

## This is a Subheading.

Name Surname  
Title, Role, and/or Expertise  
email@erlang-solutions.com

Name Surname  
Title, Role, and/or Expertise  
email@erlang-solutions.com

# This is a Presentation Headline in Proxima Bold

## This is a Subheading.

Name Surname  
Title, Role, and/or Expertise  
email@erlang-solutions.com

Name Surname  
Title, Role, and/or Expertise  
email@erlang-solutions.com



**We build scalable  
and transformative  
solutions.**

Name Surname

Title, Role, and/or Expertise

[email@erlang-solutions.com](mailto:email@erlang-solutions.com)



## A page title

A short, impactful statement,  
no more than a sentence  
or two at the very most.  
Keep it to the point.

## A page title

A short, impactful statement,  
no more than a sentence  
or two at the very most.  
Keep it to the point.

## A page title

A short, impactful statement,  
no more than a sentence  
or two at the very most.  
Keep it to the point.

## A title for a bulleted list.

- 90+ experts working across the Americas and Europe.
- Over 20 yrs and counting working with startups to Fortune 500s.
- Part of the Trifork Group of tech companies, we added value to over 300+ clients.
- We also power communities with the Code Sync family of global tech events, and trainings.



## A title for a bulleted list.

- 90+ experts working across the Americas and Europe.
- Over 20 yrs and counting working with startups to Fortune 500s.
- Part of the Trifork Group of tech companies, we added value to over 300+ clients.
- We also power communities with the Code Sync family of global tech events, and trainings.



## A title for a bulleted list.

- We build trusted, fault-tolerant systems that can scale to billions.
- Our team is passionate about developing powerful systems using Elixir, Erlang and other scalable technologies.
- Limitations get redefined, we prefer they be shattered.
- Expertise across various sectors, markets and technologies.

Fintech & Blockchain

Telecommunications

Healthcare

Gaming

## A title for a bulleted list.

- We build trusted, fault-tolerant systems that can scale to billions.
- Our team is passionate about developing powerful systems using Elixir, Erlang and other scalable technologies.
- Limitations get redefined, we prefer they be shattered.
- Expertise across various sectors, markets and technologies.

Fintech & Blockchain

Telecommunications

Healthcare

Gaming





“ Quotations are commonly printed as a means of **inspiration** and to invoke philosophical thoughts from the reader.

Quotee

Title or role



“ Quotations are commonly printed as a means of **inspiration** and to invoke philosophical thoughts from the reader.

Quotee

Title or role



“ Quotations are commonly printed as a means of **inspiration** and to invoke philosophical thoughts from the reader.

Quotee

Title or role

We need additional slide that will allow developers to insert coding lines.

# A page/ section title

## A subheading

Lorem ipsum dolor sit amet, consectetur adipiscing elit.

Mauris vulputate libero eget erat dictum, quis scelerisque sem elementum. Aliquam erat volutpat. Etiam mollis lacus nulla, in euismod ipsum bibendum eget.



“ Quotations are commonly printed as a means of inspiration and to invoke philosophical thoughts from the reader.

Quotee

Title or role



# A page/ section title

## A subheading

Lorem ipsum dolor sit amet, consectetur adipiscing elit.

Mauris vulputate libero eget erat dictum, quis scelerisque sem elementum. Aliquam erat volutpat. Etiam mollis lacus nulla, in euismod ipsum bibendum eget.



# A page/ section title

## A subheading

Lorem ipsum dolor sit amet, consectetur adipiscing elit.

Mauris vulputate libero eget erat dictum, quis scelerisque sem elementum. Aliquam erat volutpat. Etiam mollis lacus nulla, in euismod ipsum bibendum eget.





# A page/ section title

A short paragraph, supported by a bulleted list. Duis sit amet metus vitae ipsum ultrices.

- Consulting
- Development
- Support
- Code/Architecture Reviews
- Training - F2F and remote



# A page/ section title

A short paragraph, supported by a bulleted list. Duis sit amet metus vitae ipsum ultrices.

- Elixir
- Erlang
- RabbitMQ
- Kubernetes
- MongooselM
- WombatOAM
- EMQ
- Humio

# Three columns with images.



## Yellow

Is the color of gold, butter and ripe lemons. In the spectrum of visible light, yellow is found between green and orange.



## Blue

Is the colour of the clear sky and the deep sea. It is located between violet and green on the optical spectrum.



## Red

Is the color of blood, and because of this it has historically been associated with sacrifice, danger and courage.

Six image grid with a small caption to accompany



Four images with a small  
caption to accompany





Image with a small  
caption to accompany



Image with a small  
caption to accompany



## A page/ product title



Erlang is a programming language used to build massively scalable soft real-time systems with requirements on high availability.

Some of its uses are in telecoms, banking, e-commerce, computer telephony and instant messaging.



## A page/ product title



### High Availability

- Built-in fault tolerance
- Software upgrade during runtime
- Suitable for server-side apps

# A page/ product title

The best Enterprise Instant Messaging Solution  
is the one built to scale your business.

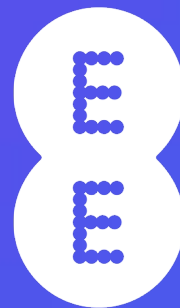
The MongooselM platform using XMPP is designed to build high-performance instant messaging systems. It is aimed at complex enterprise level projects where real-time communication is critical for business success.

Built around proven technologies XMPP/Jabber, MongooselM platform is reliable, fault-tolerant, can utilise the resources of multiple clustered machines and can scale easily when more capacity is needed.

MongooselM allows the addition of innovative real-time, social features to existing apps.



## Logo page example:



## Another logo page example...





Probably one of the most impactful systems built in Erlang.

The most widely known Erlang based messaging system built in part by Erlang Solutions.

It supported in February 2019 over 1.5 billion active users and 65 billion messages daily (29 million per minute).

It is an iconic example of a reliable and scalable messaging solution.



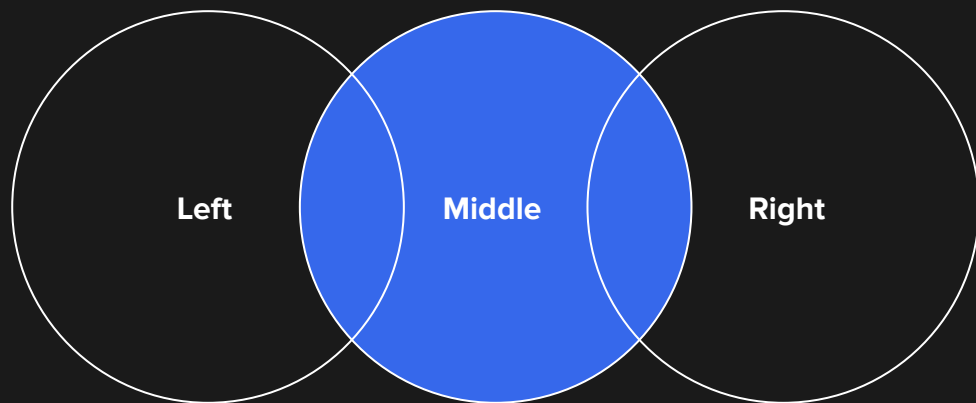
## This is a table example.

Lorem scalable, asynchronous messaging that delivers every time.

	A	B	C
Yellow	10	20	7
Blue	30	15	10
Yellow	5	24	16

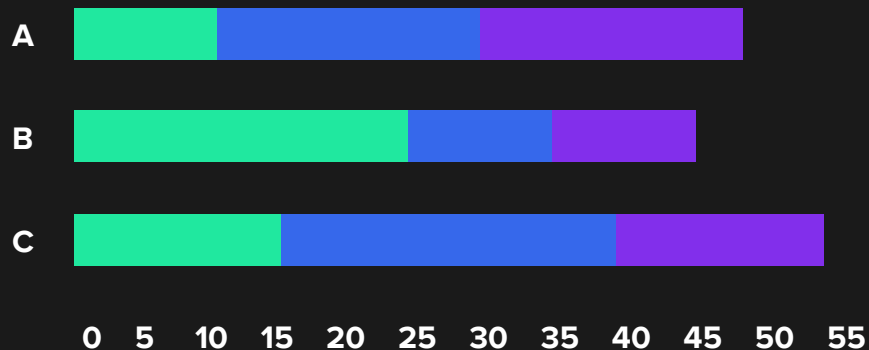
# This is a diagram example.

Lorem scalable, asynchronous messaging that delivers every time.



# This is a graph example.

Lorem scalable, asynchronous messaging that delivers every time.





# Showing a process in simple steps

## Step One

Our first stage is to research and fully understand the existing systems.



## Step Two

Then we look for ways in which to resolve any problems that arise.



## Step Three

We solve the problem with the most relevant tools and knowledge.

# Showing a process in simple steps

## Step One

Our first stage is to research and fully understand the existing systems.



## Step Two

Then we look for ways in which to resolve any problems that arise.



## Step Three

We solve the problem with the most relevant tools and knowledge.

# Showing a process in simple steps

## Step One

Our first stage is to research and fully understand the existing systems.



## Step Two

Then we look for ways in which to resolve any problems that arise.



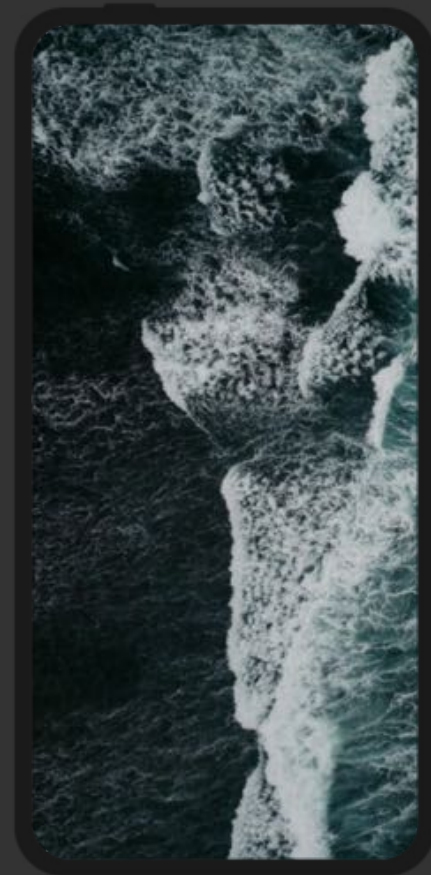
## Step Three

We solve the problem with the most relevant tools and knowledge.

# Android project visual

Lorem scalable, asynchronous  
messaging that delivers every time.  
Perfect for use in mission-critical  
applications.

- Lorem ipsum
- Dolor sit amet
- Lorem ipsum
- Lorem ipsum
- Dolor sit amet
- Lorem ipsum



# Android project visual

Lorem scalable, asynchronous messaging that delivers every time. Perfect for use in mission-critical applications.

- Lorem ipsum
- Dolor sit amet
- Lorem ipsum
- Lorem ipsum
- Dolor sit amet
- Lorem ipsum

Size your image to fit  
this screen then:

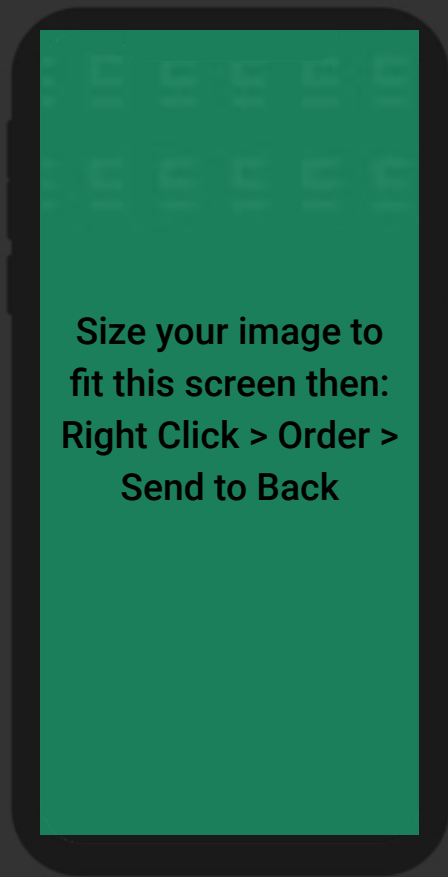
Right Click > Order >  
Send to Back.

Then Delete this.

# Iphone project visual

Lorem scalable, asynchronous  
messaging that delivers every time.  
Perfect for use in mission-critical  
applications.

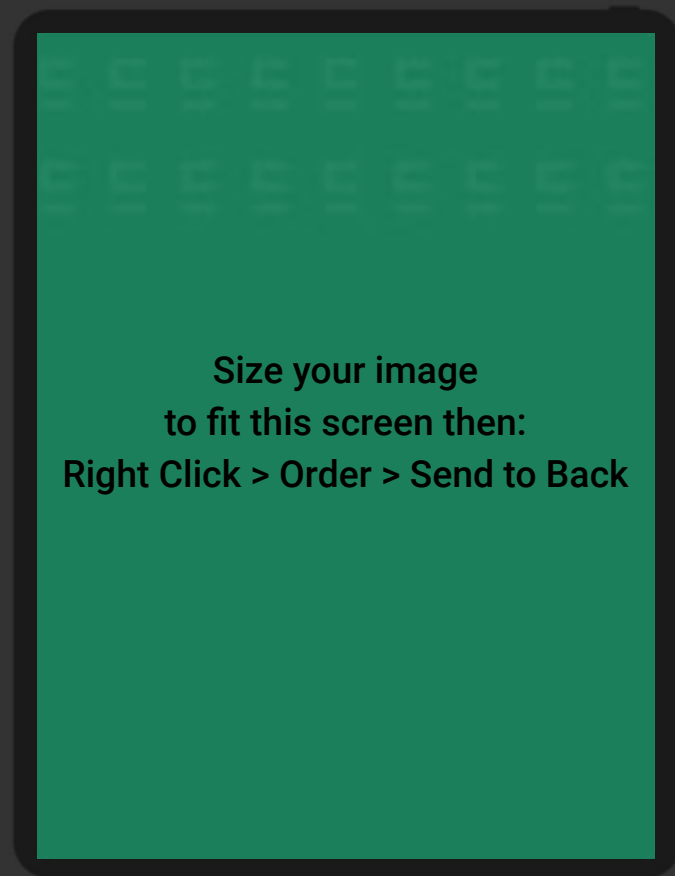
- Lorem ipsum
- Dolor sit amet
- Lorem ipsum
- Lorem ipsum
- Dolor sit amet
- Lorem ipsum



# Tablet project visual

Lorem scalable, asynchronous messaging that delivers every time. Perfect for use in mission-critical applications.

- Lorem ipsum
- Dolor sit amet
- Lorem ipsum
- Lorem ipsum
- Dolor sit amet
- Lorem ipsum



# Last slide

## Social media

## Contact details

Name Surname  
Title, Role, and/or Expertise  
[email@erlang-solutions.com](mailto:email@erlang-solutions.com)

Name Surname  
Title, Role, and/or Expertise  
[email@erlang-solutions.com](mailto:email@erlang-solutions.com)





## Contact us

London | Stockholm | Krakow | Budapest | US Remote



**[www.erlang-solutions.com](http://www.erlang-solutions.com)**

[general@erlang-solutions.com](mailto:general@erlang-solutions.com)

# Erlang Code Example

Code set in Monokai colour theme.

Currently set to shrink to fit.

Text set in Roboto Mono.

- Raw code coloured for slides by running through <http://hilite.me/>
- This code is taken from [http://www.rosettacode.org/wiki/Sorting\\_algorithms/Quicksort](http://www.rosettacode.org/wiki/Sorting_algorithms/Quicksort)
- Numbers are added using the numbered list settings within slides.

```
01 quick_sort(L) -> qs(L,  
02 trunc(math:log2(erlang:system_info(schedulers)))).  
03  
04 qs([],_) -> [];  
05 qs([H|T], N) when N > 0 ->  
06     {Parent, Ref} = {self(), make_ref()},  
07     spawn(fun()-> Parent ! {l1, Ref, qs([E||E<-T, E<H],  
N-1)} end),  
08     spawn(fun()-> Parent ! {l2, Ref, qs([E||E<-T, H ==  
E], N-1)} end),  
09     {L1, L2} = receive_results(Ref, undefined,  
undefined),  
10     L1 ++ [H] ++ L2;  
11 qs([H|T],_) ->  
12     qs([E||E<-T, E<H],0) ++ [H] ++ qs([E||E<-T, H ==  
E],0).  
13  
14 receive_results(Ref, L1, L2) ->  
15     receive  
16         {l1, Ref, L1R} when L2 == undefined ->  
17         receive_results(Ref, L1R, L2);  
18         {l2, Ref, L2R} when L1 == undefined ->  
19         receive_results(Ref, L1, L2R);  
20         {l1, Ref, L1R} -> {L1R, L2};  
21         {l2, Ref, L2R} -> {L1, L2R}  
22     after 5000 -> receive_results(Ref, L1, L2)  
23     end.
```

# Python Code Example

Code set in Monokai colour theme.

Currently set to shrink to fit.

Text set in Roboto Mono.

- Raw code coloured for slides by running through <http://hilite.me/>
- This code is taken from [http://www.rosettacode.org/wiki/Sorting\\_algorithms/Quicksort](http://www.rosettacode.org/wiki/Sorting_algorithms/Quicksort)
- Numbers are added using the numbered list settings within slides.

```
01 def quickSort(arr):
02     less = []
03     pivotList = []
04     more = []
05     if len(arr) <= 1:
06         return arr
07     else:
08         pivot = arr[0]
09         for i in arr:
10             if i < pivot:
11                 less.append(i)
12             elif i > pivot:
13                 more.append(i)
14             else:
15                 pivotList.append(i)
16         less = quickSort(less)
17         more = quickSort(more)
18         return less + pivotList + more
19
20 a = [4, 65, 2, -31, 0, 99, 83, 782, 1]
21 a = quickSort(a)
```

# C++ Code Example

Code set in Monokai colour theme.

Currently set to shrink to fit.

Text set in Roboto Mono.

- This shows some line breaks, which I imagine are probably not desirable.  
I imagine the following full width code screens might get more use.

```
01  #include <iterator>
02  #include <algorithm> // for std::partition
03  #include <functional> // for std::less
04
05  template<typename RandomAccessIterator,
06          typename Order>
07  void quicksort(RandomAccessIterator first,
08                RandomAccessIterator last, Order order)
09  {
10      if (last - first > 1)
11      {
12          RandomAccessIterator split =
13          std::partition(first+1, last,
14                        std::bind2nd(order, *first));
15          std::iter_swap(first, split-1);
16          quicksort(first, split-1, order);
17          quicksort(split, last, order);
18      }
19  }
20
21  template<typename RandomAccessIterator>
22  void quicksort(RandomAccessIterator first,
23                RandomAccessIterator last)
24  {
25      quicksort(first, last, std::less<typename
26                        std::iterator_traits<RandomAccessIterator>::va
27                        lue_type>());
28  }
```

```
01 def quickSort(arr):
02     less = []
03     pivotList = []
04     more = []
05     if len(arr) <= 1:
06         return arr
07     else:
08         pivot = arr[0]
09         for i in arr:
10             if i < pivot:
11                 less.append(i)
12             elif i > pivot:
13                 more.append(i)
14             else:
15                 pivotList.append(i)
16         less = quickSort(less)
17         more = quickSort(more)
18         return less + pivotList + more
19
20 a = [4, 65, 2, -31, 0, 99, 83, 782, 1]
21 a = quickSort(a)
```

```

01 quick_sort(L) -> qs(L, trunc(math:log2(erlang:system_info(schedulers)))).
02
03 qs([],_) -> [];
04 qs([H|T], N) when N > 0 ->
05     {Parent, Ref} = {self(), make_ref()},
06     spawn(fun()-> Parent ! {l1, Ref, qs([E||E<-T, E<H], N-1)} end),
07     spawn(fun()-> Parent ! {l2, Ref, qs([E||E<-T, H =< E], N-1)} end),
08     {L1, L2} = receive_results(Ref, undefined, undefined),
09     L1 ++ [H] ++ L2;
10 qs([H|T],_) ->
11     qs([E||E<-T, E<H],0) ++ [H] ++ qs([E||E<-T, H =< E],0).
12
13 receive_results(Ref, L1, L2) ->
14     receive
15         {l1, Ref, L1R} when L2 == undefined -> receive_results(Ref, L1R, L2);
16         {l2, Ref, L2R} when L1 == undefined -> receive_results(Ref, L1, L2R);
17         {l1, Ref, L1R} -> {L1R, L2};
18         {l2, Ref, L2R} -> {L1, L2R}
19     after 5000 -> receive_results(Ref, L1, L2)
20 end.

```

```

01  #include <iterator>
02  #include <algorithm> // for std::partition
03  #include <functional> // for std::less
04
05  template<typename RandomAccessIterator,
06          typename Order>
07  void quicksort(RandomAccessIterator first, RandomAccessIterator last, Order order)
08  {
09      if (last - first > 1)
10      {
11          RandomAccessIterator split = std::partition(first+1, last, std::bind2nd(order, *first));
12          std::iter_swap(first, split-1);
13          quicksort(first, split-1, order);
14          quicksort(split, last, order);
15      }
16  }
17
18  template<typename RandomAccessIterator>
19  void quicksort(RandomAccessIterator first, RandomAccessIterator last)
20  {
21      quicksort(first, last, std::less<typename
22  std::iterator_traits<RandomAccessIterator>::value_type>());
23  }

```

# Custom Theme Example

If we were to go for a custom theme- I'd suggest swapping the green, purple and blue of Monokai for the brand colours as shown.

- Switching the code to a darker background retains legibility.
- This would need a custom theme and a tool to deliver this styling as rich text.

```
01 quick_sort(L) -> qs(L,  
    trunc(math:log2(erlang:system_info(schedulers)))).  
02  
03 qs([],_) -> [];  
04 qs([H|T], N) when N > 0 ->  
05     {Parent, Ref} = {self(), make_ref()},  
06     spawn(fun()-> Parent ! {l1, Ref, qs([E||E<-T, E<H],  
N-1)} end),  
07     spawn(fun()-> Parent ! {l2, Ref, qs([E||E<-T, H ==  
E], N-1)} end),  
08     {L1, L2} = receive_results(Ref, undefined,  
    undefined),  
09     L1 ++ [H] ++ L2;  
10 qs([H|T],_) ->  
11     qs([E||E<-T, E<H],0) ++ [H] ++ qs([E||E<-T, H ==  
E],0).  
12  
13 receive_results(Ref, L1, L2) ->  
14     receive  
15         {l1, Ref, L1R} when L2 == undefined ->  
    receive_results(Ref, L1R, L2);  
16         {l2, Ref, L2R} when L1 == undefined ->  
    receive_results(Ref, L1, L2R);  
17         {l1, Ref, L1R} -> {L1R, L2};  
18         {l2, Ref, L2R} -> {L1, L2R}  
19     after 5000 -> receive_results(Ref, L1, L2)  
20     end.
```



# Custom Theme Example

If we were to go for a custom theme- I'd suggest swapping the green, purple and blue of Monokai for the brand colours as shown.

- Switching the code to a darker background retains legibility.
- This would need a custom theme and a tool to deliver this styling as rich text.

```
01 quick_sort(L) -> qs(L,  
    trunc(math:log2(erlang:system_info(schedulers)))).  
02  
03 qs([],_) -> [];  
04 qs([H|T], N) when N > 0 ->  
05     {Parent, Ref} = {self(), make_ref()},  
06     spawn(fun()-> Parent ! {l1, Ref, qs([E||E<-T, E<H],  
N-1)} end),  
07     spawn(fun()-> Parent ! {l2, Ref, qs([E||E<-T, H ==  
E], N-1)} end),  
08     {L1, L2} = receive_results(Ref, undefined,  
    undefined),  
09     L1 ++ [H] ++ L2;  
10 qs([H|T],_) ->  
11     qs([E||E<-T, E<H],0) ++ [H] ++ qs([E||E<-T, H ==  
E],0).  
12  
13 receive_results(Ref, L1, L2) ->  
14     receive  
15         {l1, Ref, L1R} when L2 == undefined ->  
receive_results(Ref, L1R, L2);  
16         {l2, Ref, L2R} when L1 == undefined ->  
receive_results(Ref, L1, L2R);  
17         {l1, Ref, L1R} -> {L1R, L2};  
18         {l2, Ref, L2R} -> {L1, L2R}  
19     after 5000 -> receive_results(Ref, L1, L2)  
20     end.
```

```

01 quick_sort(L) -> qs(L, trunc(math:log2(erlang:system_info(schedulers)))).
02
03 qs([],_) -> [];
04 qs([H|T], N) when N > 0 ->
05     {Parent, Ref} = {self(), make_ref()},
06     spawn(fun()-> Parent ! {l1, Ref, qs([E||E<-T, E<H], N-1)} end),
07     spawn(fun()-> Parent ! {l2, Ref, qs([E||E<-T, H =< E], N-1)} end),
08     {L1, L2} = receive_results(Ref, undefined, undefined),
09     L1 ++ [H] ++ L2;
10 qs([H|T],_) ->
11     qs([E||E<-T, E<H],0) ++ [H] ++ qs([E||E<-T, H =< E],0).
12
13 receive_results(Ref, L1, L2) ->
14     receive
15         {l1, Ref, L1R} when L2 == undefined -> receive_results(Ref, L1R, L2);
16         {l2, Ref, L2R} when L1 == undefined -> receive_results(Ref, L1, L2R);
17         {l1, Ref, L1R} -> {L1R, L2};
18         {l2, Ref, L2R} -> {L1, L2R}
19     after 5000 -> receive_results(Ref, L1, L2)
20 end.

```