# How I got tricked into writing a Clojure port

Christophe Grand @cgrand
(feat. @BaptisteDupuch the trickster)

reClojure 2021

# Who’s that guy?

- Part of the Clojure community since 2008
- Indepent dev since 2008 too
    - Lonesome Clojure coder until 2019
    - Now, Baptiste Dupuch ⋈ Christophe Grand = **Tensegritics**
    - Baptiste is another 🇫🇷 indie, working in Clojure since 2015

# Tensegritics

Our partnership is governed by two rules

1. I’m always right

2. When I’m not, Baptiste is to blame

# ClojureDart

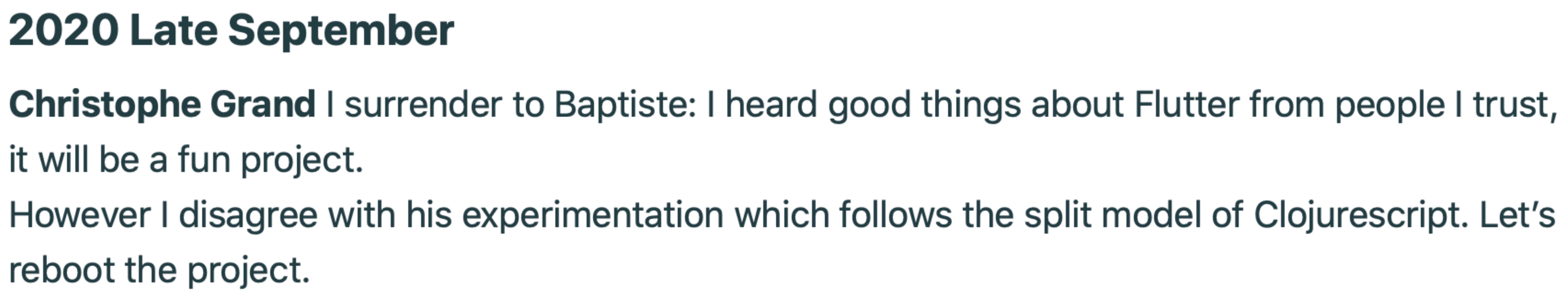
- Rule #2: It’s Baptiste’s fault
- He wanted to implement a toy lisp
- He wanted to target mobile
  - considered targeting Swift
  - settled on Dart+Flutter
    - not on my radar

# Contingency or frequency illusion

- Short after Baptiste getting started
- I heard/read positive mentions about Dart and Flutter
  - Notably from @swanodette and @coreload
- Maybe Baptiste was on something after all!

# First they laugh at you…

First entry of our dev diary:



## 2020 Late September

**Christophe Grand** I surrender to Baptiste: I heard good things about Flutter from people I trust, it will be a fun project.
However I disagree with his experimentation which follows the split model of Clojurescript. Let's reboot the project.

## So, what’s Dart + Flutter?

- Google products
- Dart: yet another statically typed object language
  - targets dartvm/native/js
  - with a conflicted attitude towards dynamism
  - message-passing concurrency only (≈ js + workers)
- Flutter: a cross-platform GUI toolkit for mobile, desktop and web

# Dart dynamism: it's complicated

- No dynamic code-loading
  - *but* hot-reload on dartvm
- Statically typed
  - *but* `dynamic` type with special method resolution semantics
  - *but* a `noSuchMethod` method
  - *but* a lying `runtimeType` property
  - *but* a treacherous objects-as-functions mechanism
  - *but* expandos

# It’s not the destination, it’s the journey!

- Spring 2020 lockdown Baptiste solo endeavour (CLJS-like compiler in CLJ, emits Dart)
- Late Sept. 2020 Reboot Let’s write a minimal compiler in dart which patch itself.
  - Too ambitious, too many headaches, too much hair-pulling
- Late Nov. 2020 Reboot Let’s write the compiler in CLJC and bootstrap from the JVM
  - 1:1 port to Clojure of the previous single-pass compiler, code-size halved
  - One month later, split the single pass in two stages with a sexp-based IR
- May 2021 Course correction Deliver the goods earlier
  - Let’s make the JVM-hosted compiler more useful
- September 2021 Course correction Lying little Dart forces us to go full-on on types

# What else we got so far

- ClojureDart is protocol-based and written exclusively in Clojure(Dart)
- Persistent Collections (except sorted ones and PersistentQueue)
- Interop with Dart
    - Clojure colls are Dart colls,
    - `nth`/`get`/`seq`/… work on Dart colls,
    - generics,
    - optional and named parameters
- Most of core.cljd (multimethods and defrecords are missing)
- `string.cljd`, `test.cljd`, `walk.cljd`

# Dart specifics: optionals

- Methods have no overload but may have optional arguments either positional or named.

-
  ```
  obj.meth(fix1, fix2, opt1) // Dart
  (.meth obj fix1 fix2 opt1) ; Cljd
  ```

-
  ```
  obj.meth(fix1, fix2, name1: opt1, name2: opt2) // Dart
  (.meth obj fix1 fix2 :name1 opt1 :name2 opt2) ; Cljd
  (.meth obj fix1 fix2 .& :name1 opt1 :name2 opt2) ; old
  ```

## Dart specifics: nullability

- Types are not nullable by default in Dart >= 2.12
- It means that the `namespace` fn which may return `nil` must be type-hinted `^String?` and not `^String`
  (we tried `^String!` but decided against it)

## Dart specifics: `^some`

- `^some` is a pseudo-type hint, it means “`Object?` but not `bool`”
- In Clojure anything is true but false and nil
  - by default boolean contexts must check both values
  - except when the type is inferred to be:
    - `bool` → only check for `false`
    - `some` or `T?` where `T` is not `Object`, `dynamic` or `bool` → only check for `nil`

# Dart specifics: generics

- Unlike Java, generics are not erased: you can’t pass a `List<Object>` where a `List<String>` is expected — even if it holds only `Strings`!

- Two problems:

  1. Expressing parametrized types:
     `List<String>?` becomes `#/(List? String)`
     Just a taglit producing `^{:type-params (String)} List?`

  2. Passing Clojure collections to Flutter or any Dart lib:
     `(.cast vector-of-widgets) ; where List<Widget> expected`
     WIP: doing it implicitly but with `*warn-on-magic-casts*`

## Dart specifics: `require`

- Like ClojureScript: strings for dart libs

```
(ns my.little.app
  (:require
   [clojure.string :as str] ; maps to cljd.string
   ["package:flutter/material.dart" :as flûte-alors]))
```

# Dart specifics: async

- Dart being single-threaded (no shared memory only message passing) it has builtin syntactic sugar for async/await
- Can't wait for/rely on `core.async` because of interop
- Need more lightweight async support
  - New special form `await`
  - `^:async` on functions (automatically inferred if `await` in the body)

We learnt Dart
so you don't have to! 😜

# Are we there yet?

- Blockers for a public alpha:
  - Finish type inference
  - Magic casts
  - Reconsider clj fns vs dart fns

# ClojureDart

public alpha

## Q1

2022

(🤞😊🤞)

# ClojureDart

Public alpha

Q1 2022

Coming to a repo near you

github.com/tensegritics/ClojureDart

(🤞😊🤞)

# 🙏 Thanks 🙏

- To our sponsors who made the leap of faith:
  - Tens of individuals
  - Nubank/Cognitect
  - Roam Research
- Sponsoring links:
  - http://github.com/sponsors/dupuchba
  - http://github.com/sponsors/cgrand