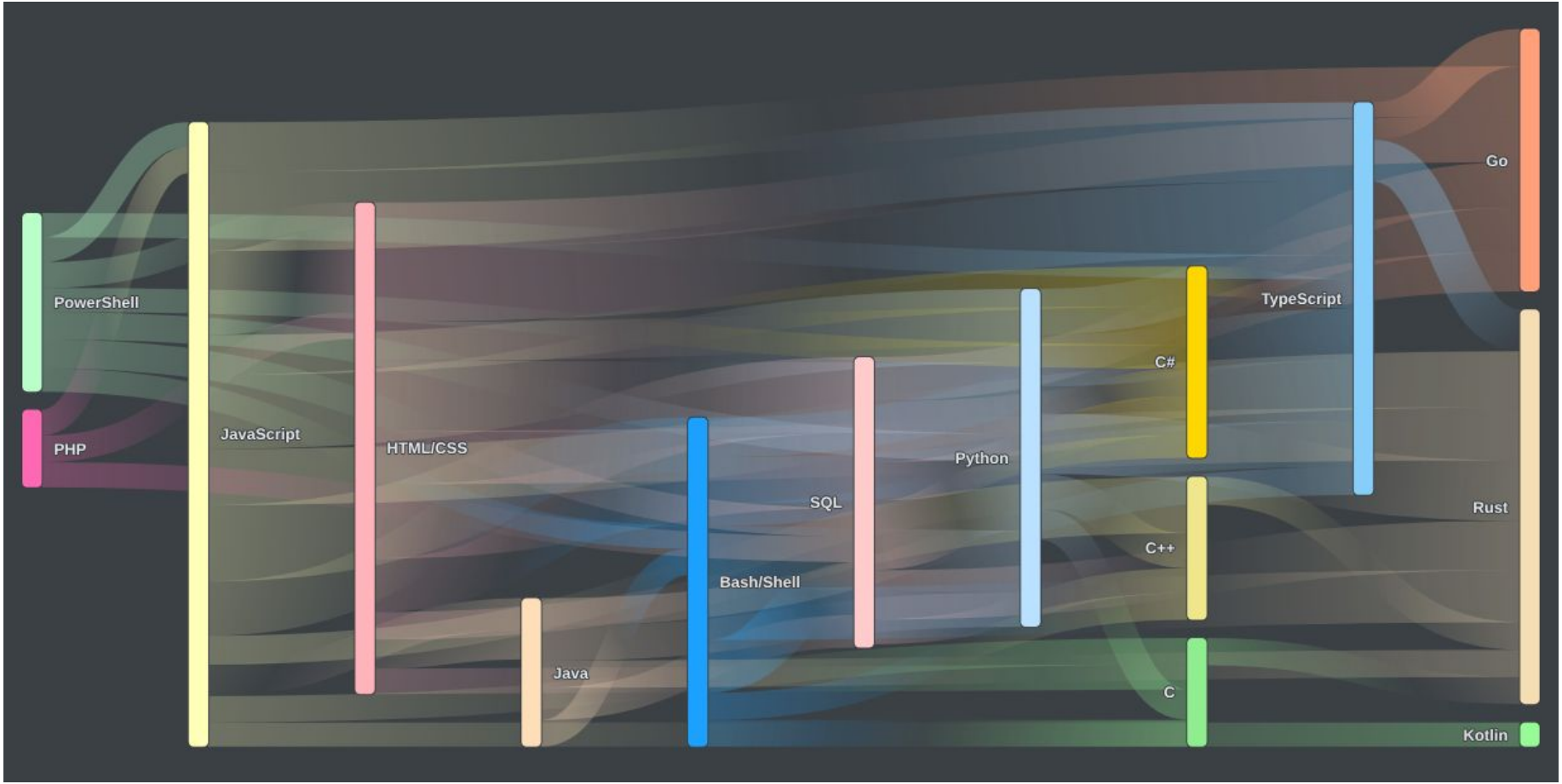
Making GStreamer Go!

Wilhelm Bartel (RSWilli)
Maintainer of the GStreamer Go bindings



https://survey.stackoverflow.co/2025/technology#3-programming-scripting-and-markup-languages

What is Go?

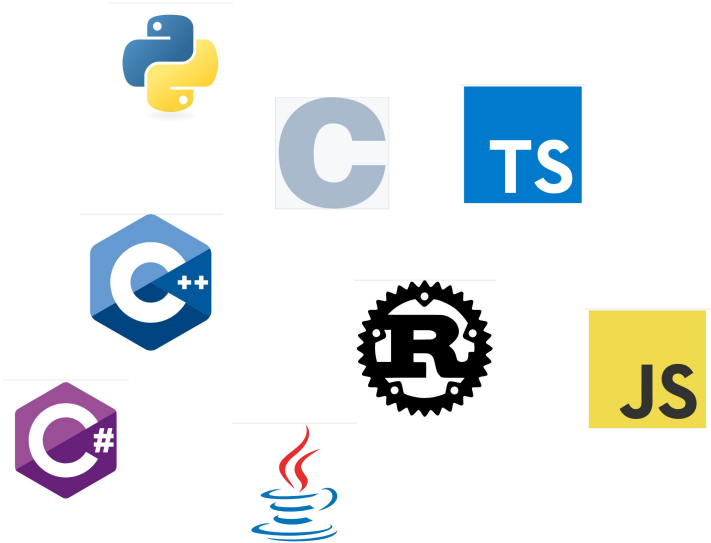
- Programming Language intended to solve development challenges at Google
- Designed in 2007 “while waiting for C++ to compile“
- Released in 2012

see Talk by Rob Pike: https://go.dev/talks/2012/splash.article

Where does GO fit in?

Compared to <*your favorite language*>, Go:

- is statically typed
- is compiled
  - multi platform, cross compilable
- is garbage collected
- uses errors as values (no exceptions)
- doesn't have inheritance
- doesn't have compile time macros or annotations
- doesn't have (safe) pointer arithmetics



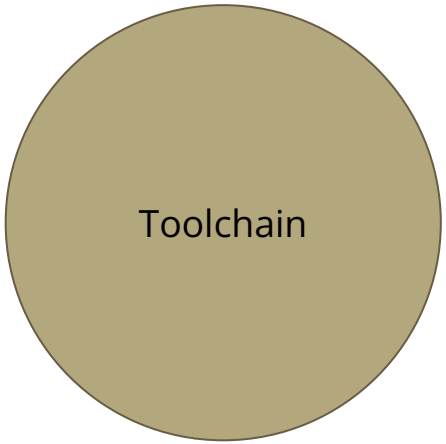
icons from https://devicon.dev/

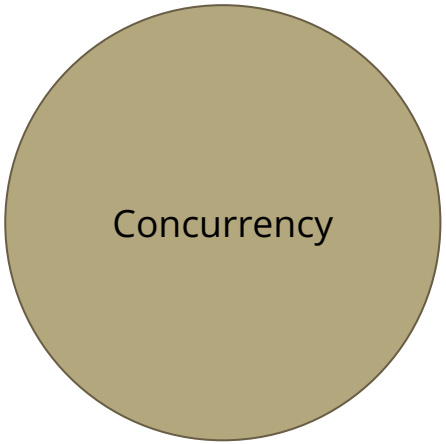
Go

```go
package main

import "fmt"

func printHelloWorld(nr int) {
    fmt.Printf("Hello World %d\n", nr)
}

func main() {
    printHelloWorld(1)
    printHelloWorld(2)
}
```

https://go.dev/play/p/AkPxGPtFDBM

Go's strong points





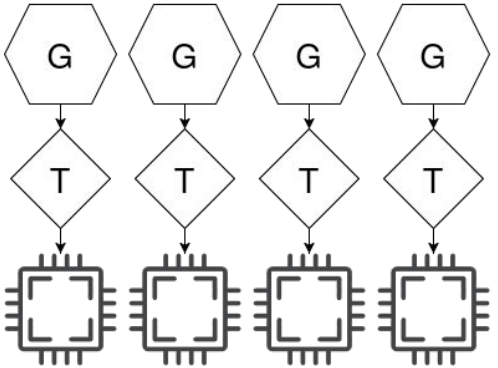
1. Go toolchain

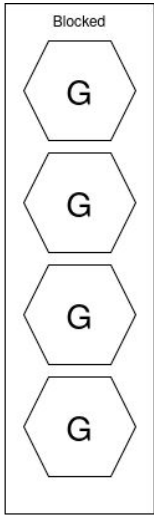
- How to install Go:
  - 1. Download and install Go
- OOTB Features in the go command:
  - Compiler (go build, go run)
  - Package manager (go mod, go get, go install)
  - Test runner (go test)
  - Code generator (go generate)
  - Tool runner:
    - Formatter (go fmt)
    - Profiler (go tool pprof)
    - Linter (go vet)
    - Race condition detector (go run -race, go test -race, ...)
    - ...

AI Image

2. Go and concurrency

- Go has goroutines
- essentially userland threads
- 2 Kb growable stack size
- scheduled by the runtime onto kernel threads
  - similar to threadshare by François
- a goroutine can block
  - does not block the thread
  - wait for network, file system, mutex, timers, ...

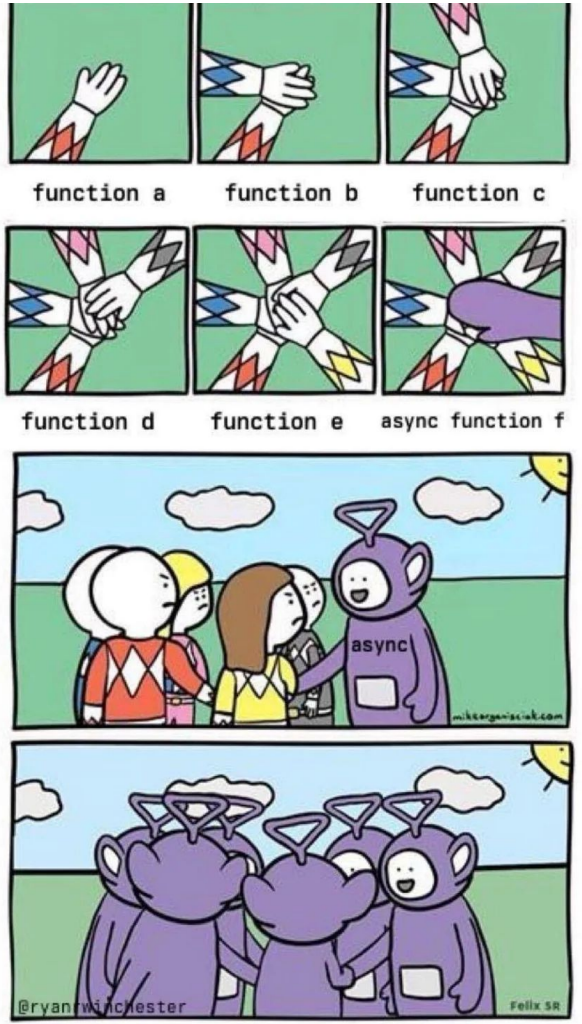




2. Go and concurrency

Go doesn't have coloured functions

```
async function doFoo() {
    return fetch("/foobar")
}

function func1() {
    const foo = doFoo()

    console.log(foo)
}
```



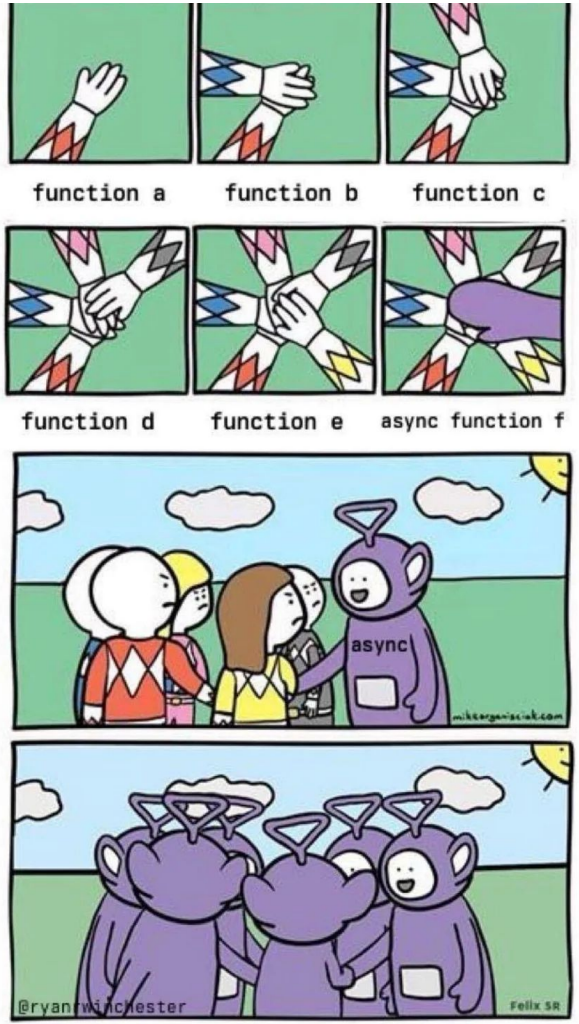
https://www.reddit.com/r/ProgrammerHumor/comments/1h4v9i3/dontyouhateitwhenthathappens/

2. Go and concurrency

Go doesn't have coloured functions

```
async function doFoo() {
    return fetch("/foobar")
}

function func1() {
    const foo = await doFoo()

    console.log(foo)
}
```

JS ☹

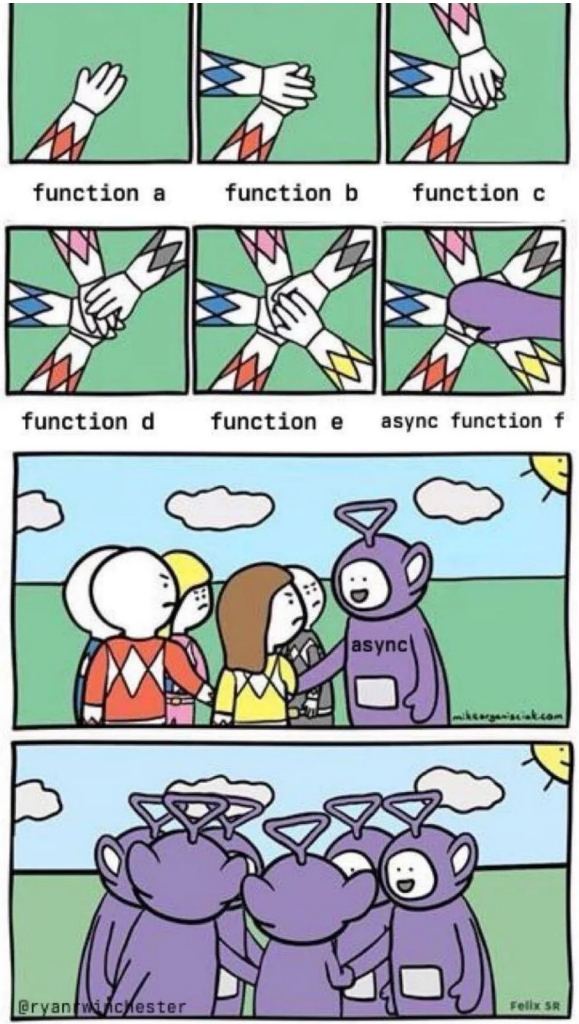


https://www.reddit.com/r/ProgrammerHumor/comments/1h4v9i3/dontyouhateitwhenthathappens/

2. Go and concurrency

Go doesn't have coloured functions

```
async function doFoo() {
    return fetch("/foobar")
}

async function func1() {
    const foo = await doFoo()

    console.log(foo)
}
```



https://www.reddit.com/r/ProgrammerHumor/comments/1h4v9i3/dontyouhateitwhenthathappens/

Concurrency with goroutines in action

```
package main

import (
    "fmt"
    "time"
)

func say(s string) {
    for i := 0; i < 5; i++ {
        time.Sleep(100 * time.Millisecond)
        fmt.Println(s)
    }
}

func main() {
    say("hello")
    say("world")
}
```

https://go.dev/play/p/NG4cVndn8lr

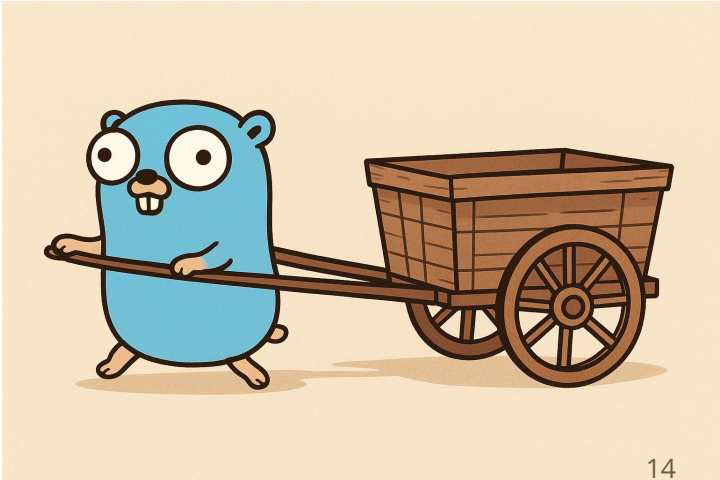
Concurrency with goroutines in action

```
package main

import (
    "fmt"
    "time"
)

func say(s string) {
    for i := 0; i < 5; i++ {
        time.Sleep(100 * time.Millisecond)
        fmt.Println(s)
    }
}

func main() {
    go say("Hello")
    say("World")
}
```

https://go.dev/play/p/6P7TehgokXB

Go is a workhorse

- Go is not flashy
- Designed to get the job done
- writing Go is very productive
- Many projects choose Go
  - Docker
  - Kubernetes
  - Prometheus
  - Grafana
  - Consul
  - Vault
  - ...



AI Image

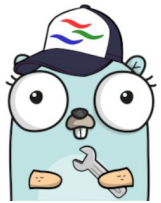
Go at streaMonkey

- Startup from Leipzig, Germany
- B2B in the radio sector
- IP Radio stream distribution to ~300.000 Listeners

- Backend code is entirely written in Go
- Heavily relies on ffmpeg

Go at streaMonkey #2

- Idea: Cloud playout system
- Multiple decks
  - File Sources
  - Live sources
- Live moderation via WebRTC from any Browser
- Adhoc mixing
- ...



gstreamer + GO

Existing bindings

- https://github.com/tinyzimmer/go-gst
  + https://github.com/tinyzimmer/go-glib fork of https://github.com/gotk3/gotk3

- Combined effort in new organisation: https://github.com/go-gst/go-gst
  + https://github.com/go-gst/go-glib

- Many features already implemented
- Handwritten, manually extended
- Complicated tangled licenses because of forks

The new approach

- generate Go code from GI docs
- manually implement some parts
- Result: Two PRs
  - https://github.com/go-gst/go-glib/pull/32 +334,905 −7,815
  - https://github.com/go-gst/go-gst/pull/170 +400,122 −30,365

```
<record name="Buffer" c:type="GstBuffer" glib:type-name="GstBuffer" glib:get-type="gst_buffer_get_type"
c:symbol-prefix="buffer">
 <method name="foreach_meta" c:identifier="gst_buffer_foreach_meta">
   <doc xml:space="preserve">Calls @func with @user_data for each meta in @buffer.

@func can modify the passed meta pointer or its contents. The return value
of @func defines if this function returns or if the remaining metadata items
in the buffer should be skipped.</doc>
   <return-value transfer-ownership="none">
     <doc xml:space="preserve">%FALSE when @func returned %FALSE for one of the metadata.</doc>
     <type name="gboolean" c:type="gboolean"/>
   </return-value>
   <parameters>
     <instance-parameter name="buffer" transfer-ownership="none">
       <doc xml:space="preserve">a #GstBuffer</doc>
       <type name="Buffer" c:type="GstBuffer*"/>
     </instance-parameter>
     <parameter name="func" transfer-ownership="none" scope="call" closure="1">
       <doc xml:space="preserve">a #GstBufferForeachMetaFunc to call</doc>
       <type name="BufferForeachMetaFunc" c:type="GstBufferForeachMetaFunc"/>
     </parameter>
     <parameter name="user_data" transfer-ownership="none" nullable="1" allow-none="1">
       <doc xml:space="preserve">user data passed to @func</doc>
       <type name="gpointer" c:type="gpointer"/>
     </parameter>
   </parameters>
 </method>
</record>
```

```go
// ForEachMeta wraps gst_buffer_foreach_meta
//
// The function takes the following parameters:
//
//  - fn BufferForeachMetaFunc: a #GstBufferForeachMetaFunc to call
//
// The function returns the following values:
//
//  - goret bool
//
// Calls @func with @user_data for each meta in @buffer.
//
// @func can modify the passed meta pointer or its contents. The return value
// of @func defines if this function returns or if the remaining metadata items
// in the buffer should be skipped.
func (buffer *Buffer) ForEachMeta(fn BufferForeachMetaFunc) bool {
   var carg0 *C.GstBuffer               // in, none, converted
   var carg1 C.GstBufferForeachMetaFunc// callback, scope: call, closure: carg2
   var carg2 C.gpointer                 // implicit
   var cret  C.gboolean                 // return

   carg0 = (*C.GstBuffer)(UnsafeBufferToGlibNone(buffer))
   carg1 = (*[0]byte)(C._goglib_gst1_BufferForeachMetaFunc)
   carg2 = C.gpointer(userdata.Register(fn))
   defer userdata.Delete(unsafe.Pointer(carg2))

   cret = C.gst_buffer_foreach_meta(carg0, carg1, carg2)
   runtime.KeepAlive(buffer)
   runtime.KeepAlive(fn)

   var goret bool

   if cret != 0 {
       goret = true
   }

   return goret
}
```

Features of the Go bindings

- GLib:
  - GLib, GObject, GIO
- GStreamer:
  - Gst, GstAllocators, GstApp, GstAudio, GstBase, GstCheck, GstController, GstGL, GstMpegts, GstNet, GstPbutils, GstPlay, GstPlayer, GstRtp, GstRtsp, GstSdp, GstTag, GstVideo, GstWebRTC
- GTK: possible, but not currently (Help wanted!)

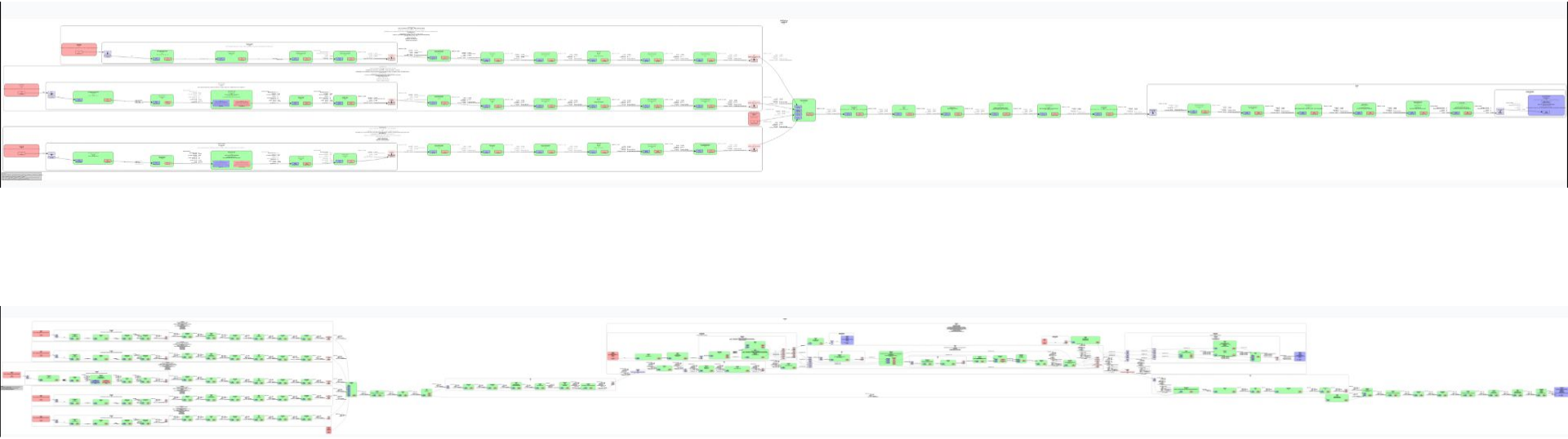
Features of the Go bindings

- Refcounting done via the GC
  - TODO: manual opt out to manual (but more unsafe) unreffing
- Standard GStreamer stuff:
  - Pipeline manipulation
  - Pad Probes
  - GObject Signal handling/emitting
  - ...
- Convenience functions
- Subclassing
- Writing Plugins (Thanks to tinyzimmer)
  - compiling Go in c-shared mode
  - requires a tiny bit of unsafe user code boilerplate
  - possibility of future gst-plugins-go ?

```go
package main

import "github.com/go-gst/go-gst/pkg/gst"

func main() {
    gst.Init()

    res, err := gst.ParseLaunch("audiotestsrc ! fakesink")
    if err != nil {
        fmt.Printf("Parse error: %v", err)
        return
    }

    pipeline := res.(gst.Pipeline)

    pipeline.SetState(gst.StatePlaying)
}
```

The Result

- It works!



State of the Go bindings

- Nothing too flashy
- Testing needed for many features (especially video)
- Gets the job done
- It works: tested at streaMonkey in production for >2y

Go try them youselves

Thank you