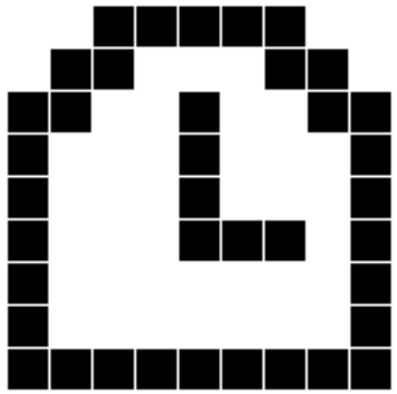
Adventures with go.tools/ssa

A talk by Elliott Stoneham at the Go London User Group of 21st November 2013

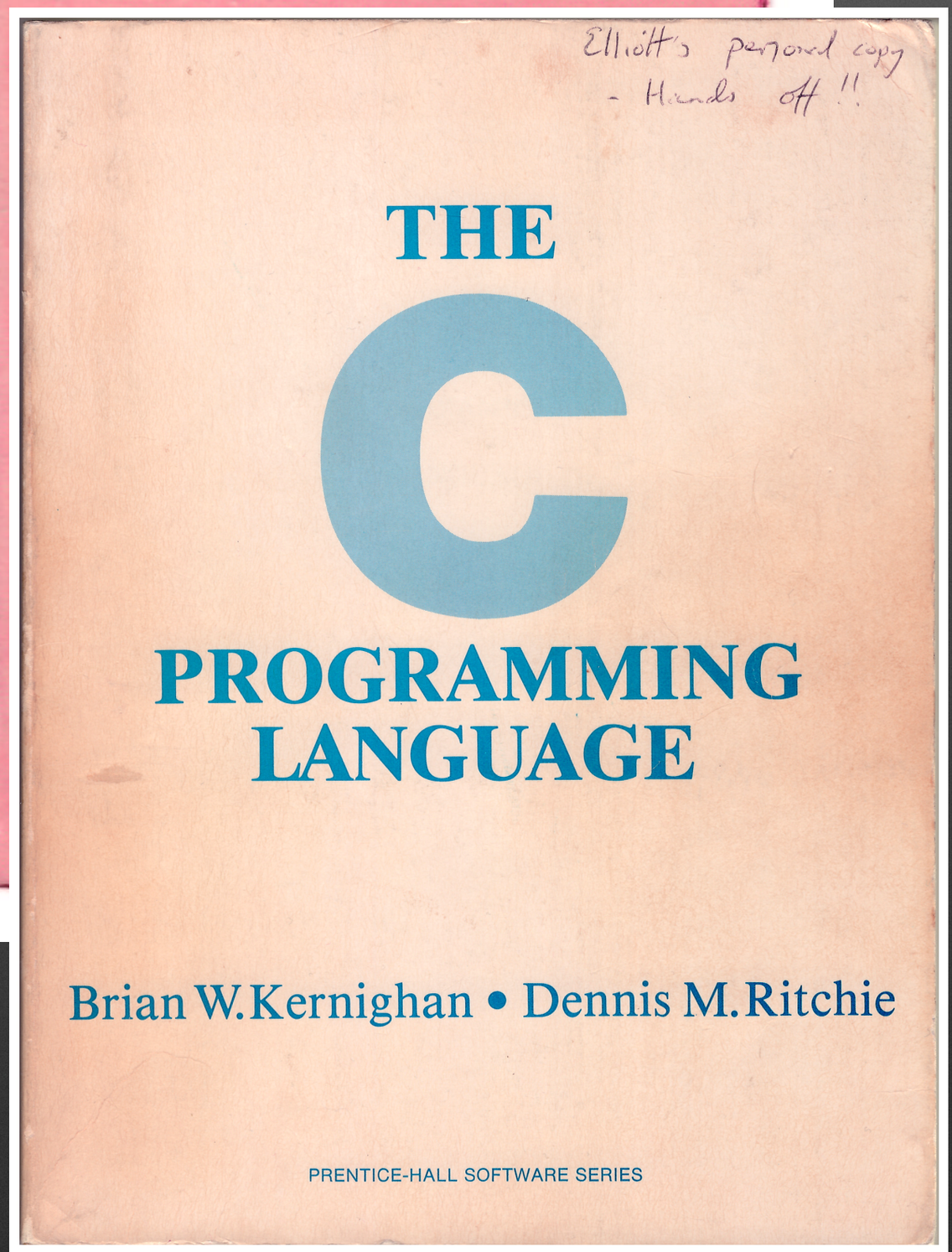


The countdown timer is running

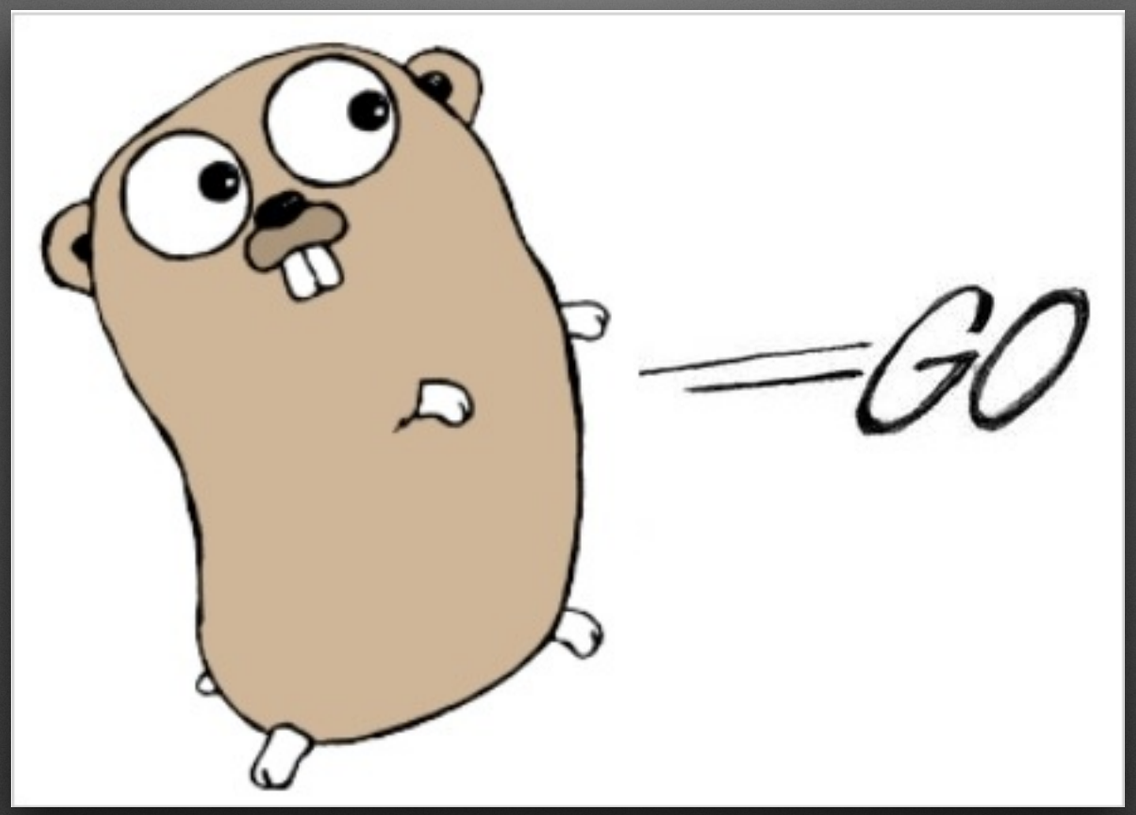
Questions at the end please

To deliver over 30 slides in under 20 minutes I need to speak fast…

The last time I was paid to be a programmer

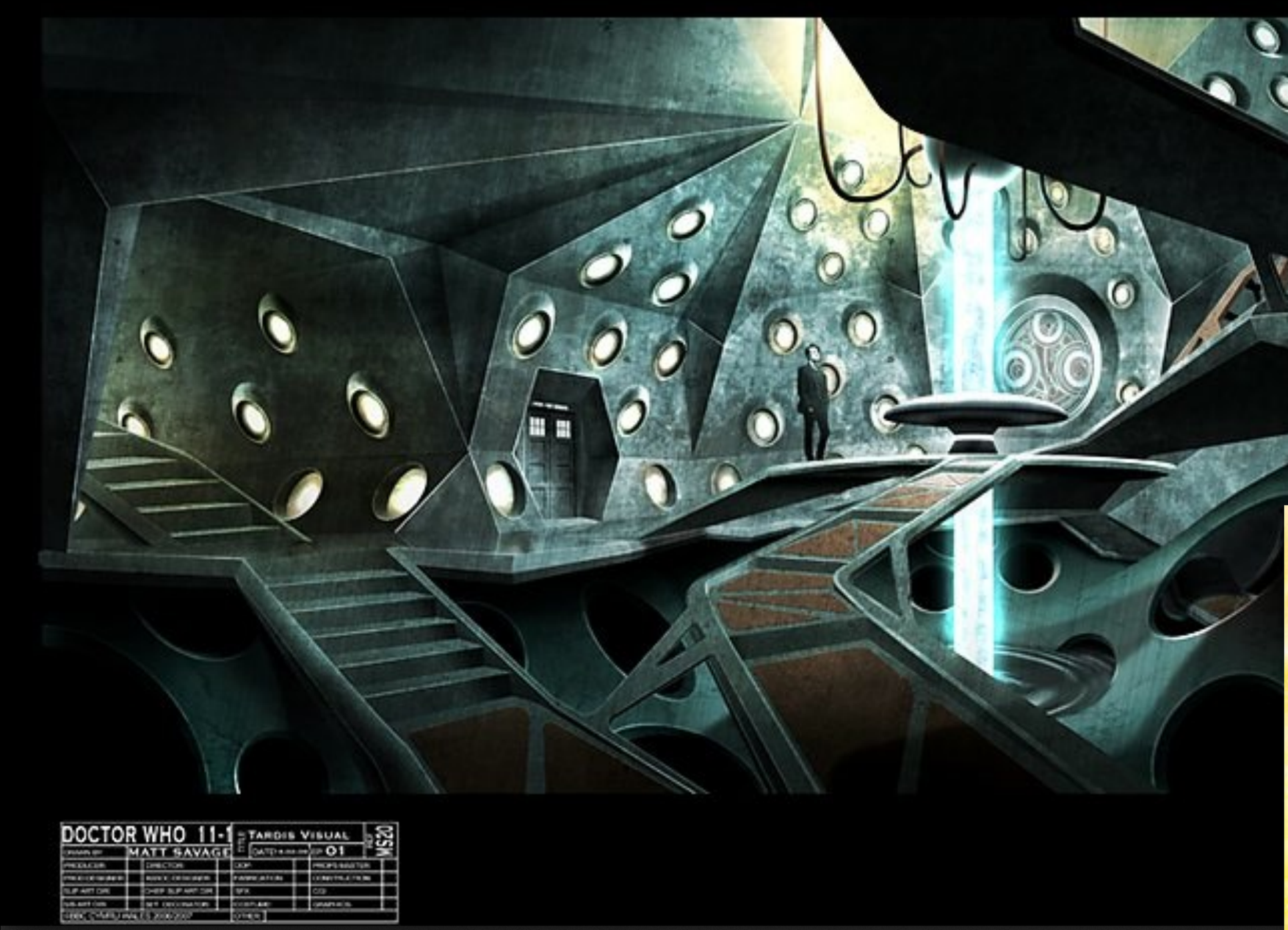


After 20+ years away,
Go has tempted me back



**“Go is like a better C,
from the guys that didn’t bring you C++”**

— Ikai Lan



Go is like the TARDIS

A small idiosyncratic exterior,
conceals large-scale quality engineering.

Everything that follows is a

PROTOTYPE

Consulting the Oracle

go.tools/cmd/oracle

The oracle answers questions about Go code

“Questions such as:

- What is the type of this expression? What are its methods?
- What’s the value of this constant expression?
- Where is the definition of this identifier?
- What are the exported members of this imported package?
- What are the free variables of the selected block of code?
- What interfaces does this type satisfy?
- Which concrete types implement this interface?”

“And [looking across the whole application]:

- What are the possible concrete types of this interface value?
- What are the possible callees of this dynamic call?
- What are the possible callers of this function?
- What objects might this pointer point to?
- Where will a value sent on this channel be received?
- Which statements could update this field/local/global/map/array/etc?
- Which functions might be called indirectly from this one?”

“using the oracle is as simple as selecting a region of source code, pressing a button, and receiving a precise answer”

(but I’ve not been able to get EMACS working)

github.com/fzipp/pythia

“Pythia is a web front-end for the Go source code oracle, which is a source code comprehension tool for Go programs.”

go.tools/ssa was created for go.tools/cmd/oracle

As a prerequisite to pointer analysis, the program must first be converted from typed syntax trees into a simpler, more explicit intermediate representation (IR), as used by a compiler. We use a high-level static single-assignment (SSA) form IR in which the elements of all Go programs can be expressed using only about 30 basic instructions.

(Oracle design document, Alan Donovan, Google)

The 36 SSA Instructions

	Value?	Instruction?
*Alloc	✔	✔
*BinOp	✔	✔
*Builtin	✔	✔
*Call	✔	✔
*ChangeInterface	✔	✔
*ChangeType	✔	✔
*Convert	✔	✔
*DebugRef		✔
*Defer		✔
*Extract	✔	✔
*Field	✔	✔
*FieldAddr	✔	✔
*Go		✔
*If		✔
*Index	✔	✔
*IndexAddr	✔	✔
*Jump		✔
*Lookup	✔	✔
*MakeChan	✔	✔
*MakeClosure	✔	✔
*MakeInterface	✔	✔
*MakeMap	✔	✔
*MakeSlice	✔	✔
*MapUpdate		✔
*Next	✔	✔
*Panic		✔
*Phi	✔	✔
*Range	✔	✔
*Return		✔
*RunDefers		✔
*Select	✔	✔
*Send		✔
*Slice	✔	✔
*Store		✔
*TypeAssert	✔	✔
*UnOp	✔	✔

Static Single Assignment

“In compiler design, static single assignment form (often abbreviated as SSA form or simply SSA) is a property of an intermediate representation (IR), which says that **each variable is assigned exactly once**. Existing variables in the original IR are split into versions, new variables typically indicated by the original name with a subscript in textbooks, so that every definition gets its own version.”

(Wikipedia)

```
func fact(n int) int {
    if n == 0 {
        return 1
    }
    return n * fact(n-1)
}
```

go -> ssa example

```
# Name: main.fact
# Package: main
# Location: glug.go:9:6
func fact(n int) int:
.0.entry:                                                  P:0 S:2
   t0 = n == 0:int                                        bool
   if t0 goto 1.if.then else 2.if.done
.1.if.then:                                                P:1 S:0
   return 1:int
.2.if.done:                                                P:1 S:0
   t1 = n - 1:int                                          int
   t2 = fact(t1)                                           int
   t3 = n * t2                                             int
   return t3
```

go.tools/cmd/ssadump

```
$ ssadump -help
Usage of ssadump:
  -build="": Options controlling the SSA builder.
The value is a sequence of zero or more of these letters:
C  perform sanity [C]hecking of the SSA form.
D  include [D]ebug info for every function.
P  log [P]ackage inventory.
F  log [F]unction SSA code.
S  log [S]ource locations as SSA builder progresses.
G  use binary object files from gc to provide imports (no code).
L  build distinct packages seria[L]ly instead of in parallel.
N  build [N]aive SSA form: don't replace local loads/stores with registers.

  -cpuprofile="": write cpu profile to file
  -interp="": Options controlling the SSA test interpreter.
The value is a sequence of zero or more more of these letters:
R  disable [R]ecover() from panic; show interpreter crash instead.
T  [T]race execution of the program.  Best for single-threaded programs!

  -run=false: Invokes the SSA interpreter on the program.
```

Go SSA viewer

https://github.com/tmc/ssaview (https://github.com/tmc/ssaview)

Shows the SSA (Static Single Assignment) (http://en.wikipedia.org/wiki/Static_single_assignment_form) representation of input code.
Uses the wonderful go.tools/ssa (http://godoc.org/code.google.com/p/go.tools/ssa) package.

status: done

Input

```
1  package main            // example inspired by gobyexample.c
2  import  _ "runtime"     // required by go.tools/ssa/interp
3  func fact(n int) int {
4      if n == 0 {
5          return 1
6      }
7      return n * fact(n-1)
8  }
9  func main() { println("ten factorial is ",fact(10)) }
```

Result

```
1  package main:
2    func  fact       func(n int) int
3    func  init       func()
4    var   init$guard bool
5    func  main       func()
6
7  # Name: main.init
8  # Package: main
9  # Synthetic: package initializer
10 func init():
11 .0.entry:
12     t0 = *init$guard
13     if t0 goto 2.init.done else 1.init.start
14 .1.init.start:
```

http://golang-ssaview.herokuapp.com/

To see this application running live

Go Interpreter (35 lines)

```
package main
import (
    "code.google.com/p/go.tools/importer"
    "code.google.com/p/go.tools/ssa"
    "code.google.com/p/go.tools/ssa/interp"
    "fmt"
    "go/build"
    "go/parser"
)
const code = `
package main            // example inspired by gobyexample.com/recursion
import  _ "runtime"     // required by go.tools/ssa/interp
func fact(n int) int {
    if n == 0 {
        return 1
    }
    return n * fact(n-1)
}
func main() { println("ten factorial is ",fact(10)) }
`
func main() {
    imp := importer.New(&importer.Config{Build: &build.Default}) // Imports will be loaded as if by 'go build'.
    file, err := parser.ParseFile(imp.Fset, "main.go", code, 0)  // Parse the input file.
    if err != nil {
        fmt.Print(err) // parse error
        return
    }
    mainInfo := imp.CreatePackage("main", file) // Create single-file main package and import its dependencies.
    var mode ssa.BuilderMode
    prog := ssa.NewProgram(imp.Fset, mode) // Create SSA-form program representation.
    if err := prog.CreatePackages(imp); err != nil {
        fmt.Print(err) // type error in some package
        return
    }
    mainPkg := prog.Package(mainInfo.Pkg)
    packages := prog.AllPackages() // Build SSA code for bodies of functions in all packages
    for p := range packages {
        packages[p].Build()
    }
    rv := interp.Interpret(mainPkg, 0, "", nil)
    if rv != 0 {
        fmt.Printf("Interpreter error: %d\n", rv) // show any error
    }
}
```

```
$ go run interp.go
ten factorial is  3628800
```

“It is not, and will never be, a production-quality Go interpreter.”

github.com/rocky/ssa-interp

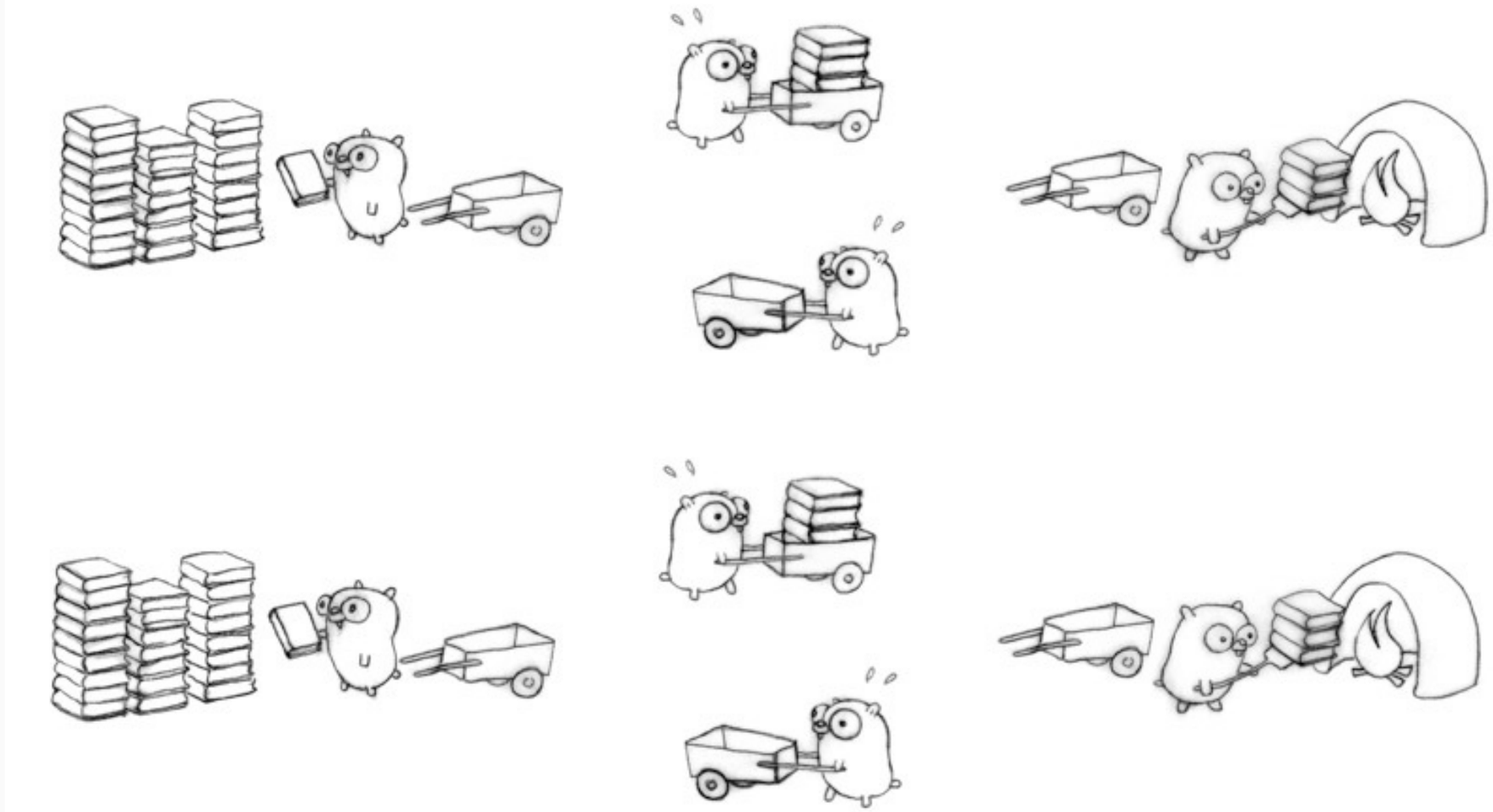
“gub and tortoise - A Go SSA Debugger and Interpreter”

(not currently compiling)

Highlights the problems of working with go.tools/ssa:

- Frequent changes to the API, but stabilising now
- Requires working at tip.golang.org
- Few peer projects to learn from

…but excellent code quality and documentation!



Two Go compiler projects based on go.tools/ssa

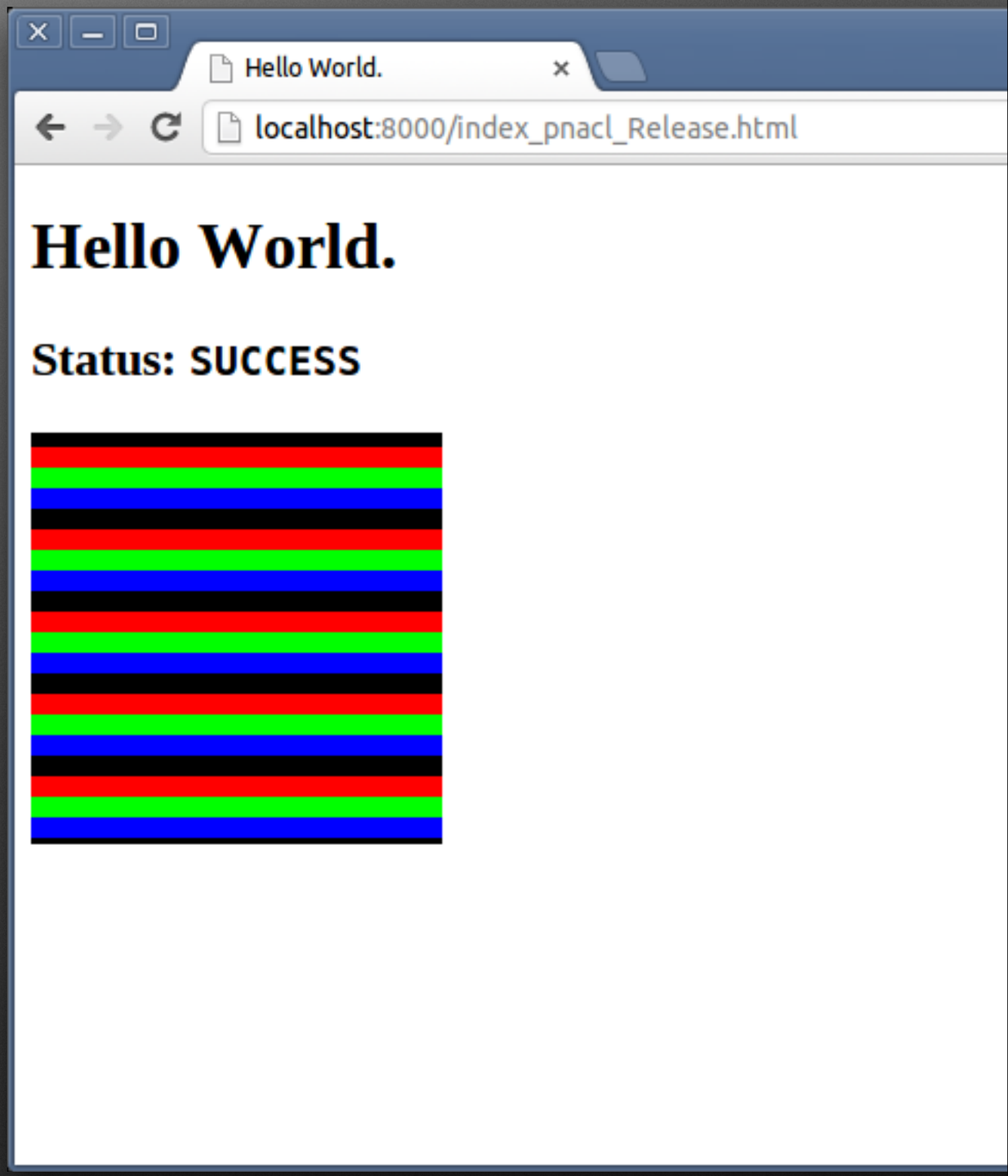
LLVM web applications in Go

llgo

github.com/axw/llgo

"llgo is a compiler for Go, written in Go, and using the LLVM compiler infrastructure."

The image on the right shows an llgo prototype running in the Chrome/Chromium browser using PNaCl



Andrew Wilkins, author of llgo, has written:

- “I’m committed to doing [the rewrite to use go.tools/ssa].”
- “I … was was invited to lunch/discuss llgo with some members of the core Go team. It was fairly informal, with no specific outcomes. It's highly likely that the llgo runtime will be replaced with libgo, sooner rather than later.”
- “I really wish I had more time to play with [PNaCl].”

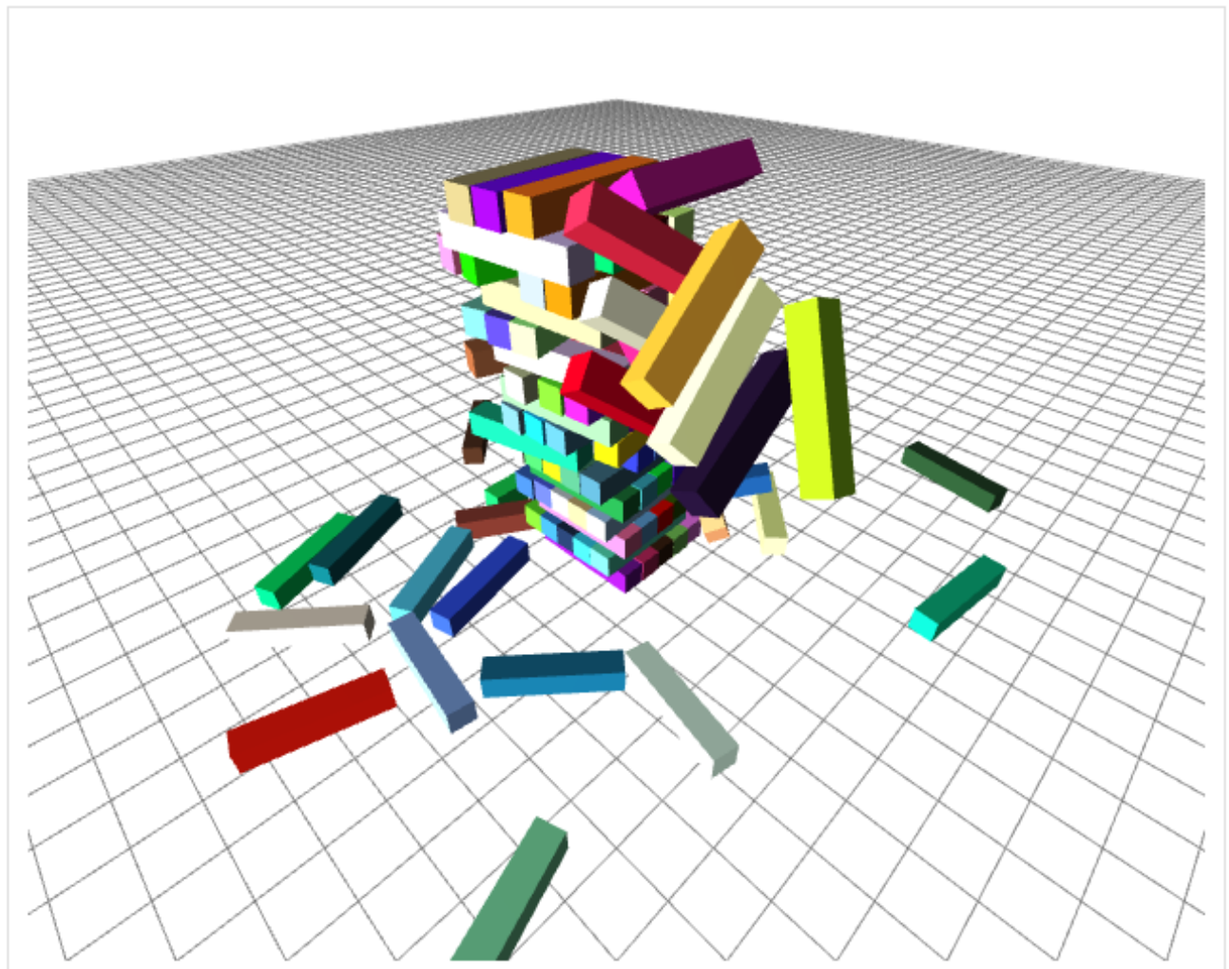
PNaCl Demos

- Lua Editor
- Bullet Physics
- Raycasted Earth
- Game of Life
- Voronoi Simulation
- SmoothLife
- Rotating Cube

Bullet Physics

An example of the Bullet Physics SDK ported to Native Client, using WebGL for rendering.

Example courtesy John McCutchan. Read his description of the demo or browse the source. Simulation time: 3550µs.



Controls

- **Click and drag** an object to move it
- **Click and drag** elsewhere to rotate the camera
- Use the **mousewheel** to zoom in/out

Add Objects

Block Tower (10 blocks) | Block Tower (20 blocks)
Random Cubes (250) | Random Cubes (1000)
Random Cubes (2000) | Random Cylinders (500)
Random Shapes | Reload Scene

developers.google.com/native-client/

PNaCl is a software framework that allows developers to compile native C and C++ code so that it can be embedded in a web page and run in a Chrome browser on any platform.

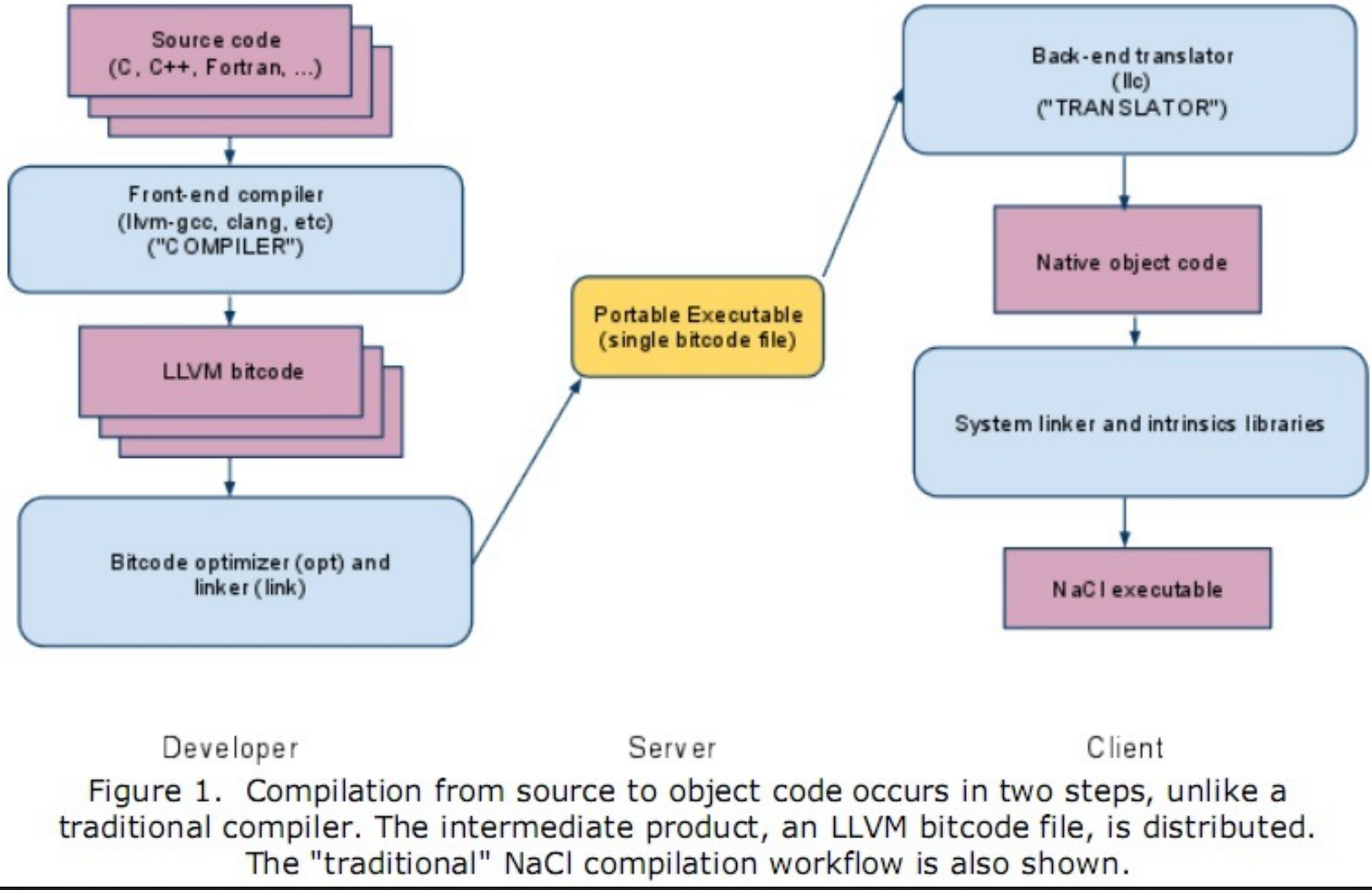


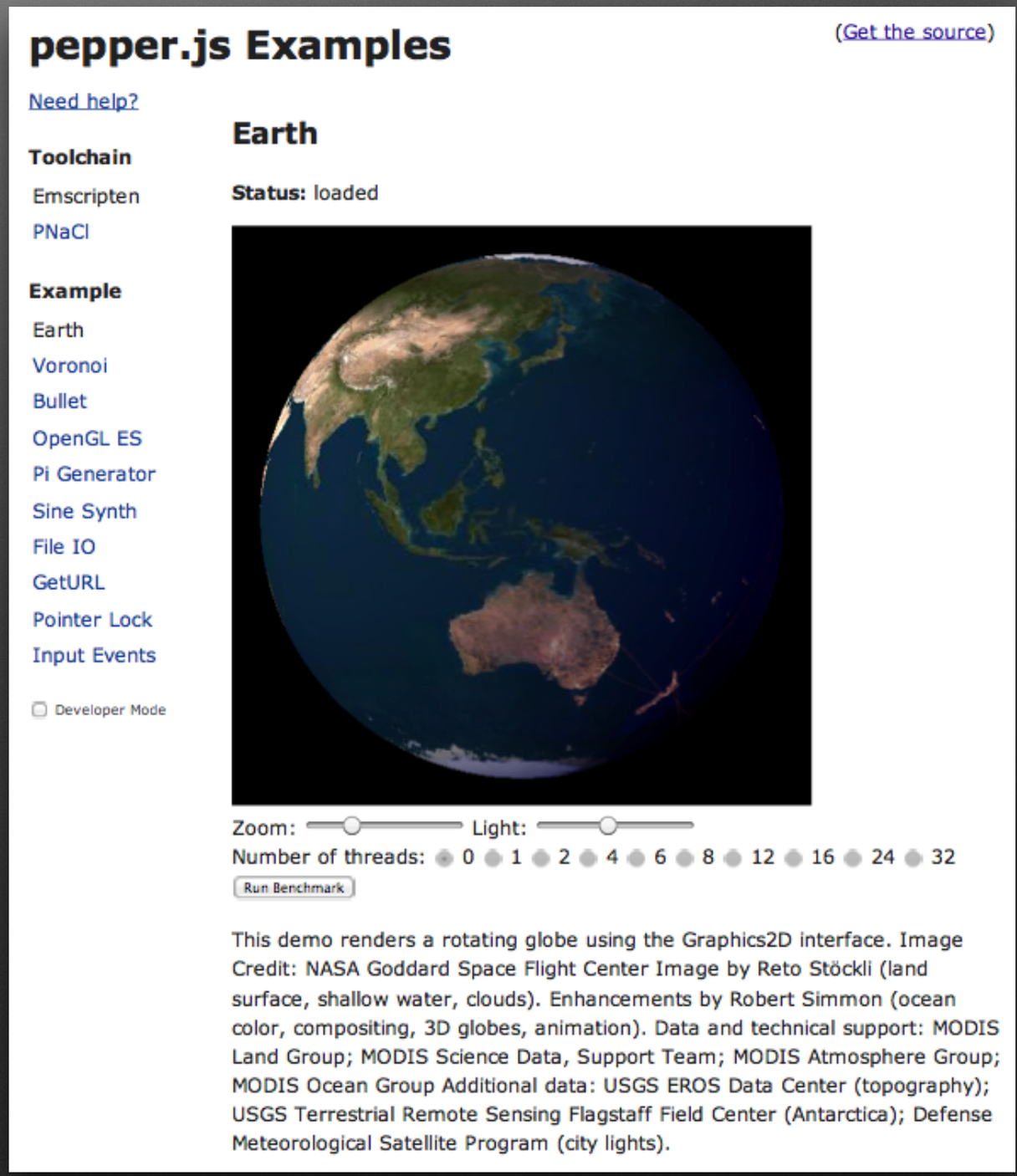
Figure 1. Compilation from source to object code occurs in two steps, unlike a traditional compiler. The intermediate product, an LLVM bitcode file, is distributed. The "traditional" NaCl compilation workflow is also shown.

How source code is compiled to run on PNaCl

(From a document written by Googler Alan Donovan, who also wrote go.tools/ssa)

github.com/google/pepper.js

“pepper.js is a JavaScript library that enables the compilation of native Pepper applications into JavaScript using Emscripten. This allows the simultaneous deployment of native code on the web both as a Portable Native Client (PNaCl) executable and as JavaScript. Native Pepper applications can now be run in Chrome, Firefox, Internet Explorer, Safari, and more.”

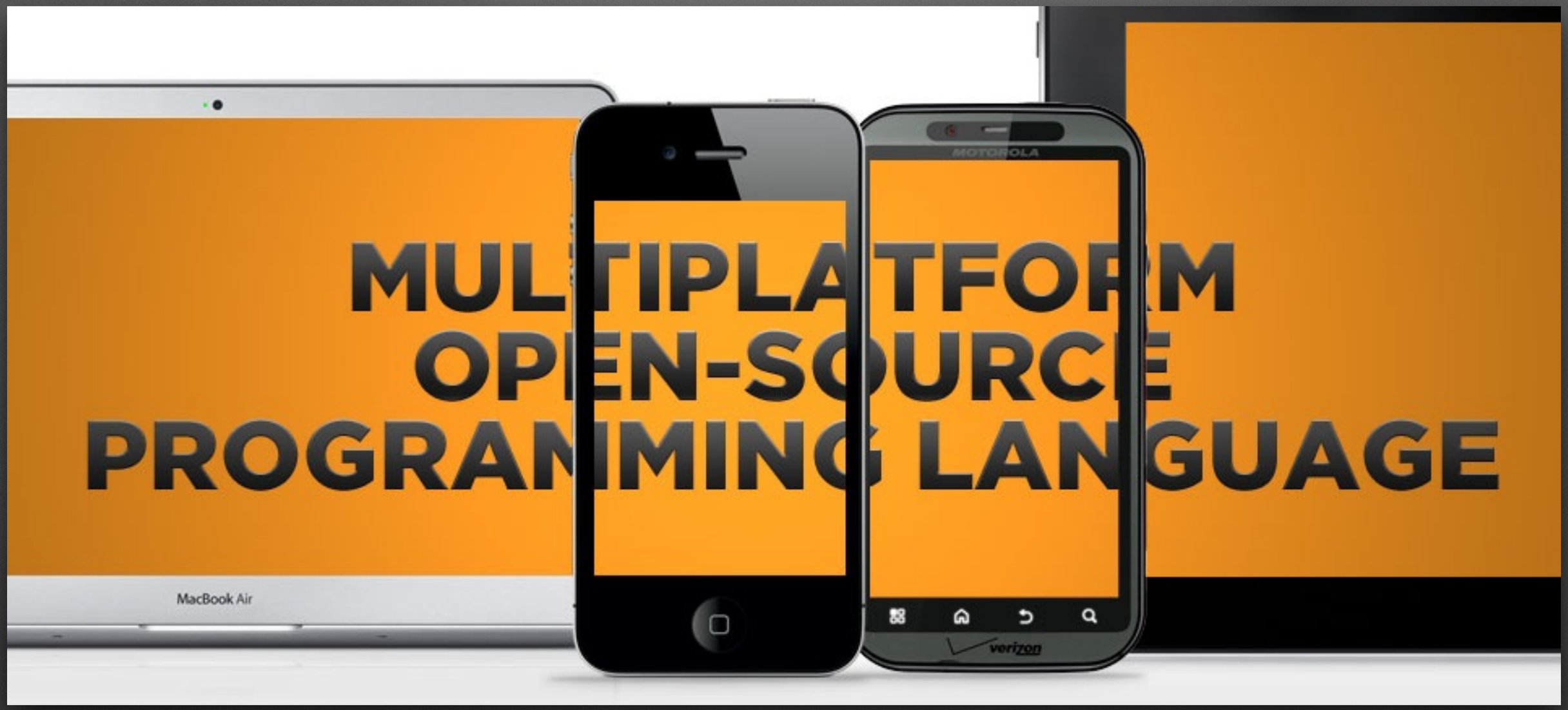


github.com/kripken/emscripten/wiki

Emscripten is an open source LLVM to JavaScript compiler.
It lets you take code written in C or C++ and run it on the web.

无名

Introducing my un-named, un-published and un-finished
Go cross-compiler project using go.tools/ssa



haxe.org

“Haxe can be compiled to all popular programming platforms with its fast compiler – JavaScript, Flash, NekoVM, PHP, C++, C# and Java – which means your apps will support all popular mobile devices, such as iOS, Android, BlackBerry and more.”

go -> ssa -> haxe

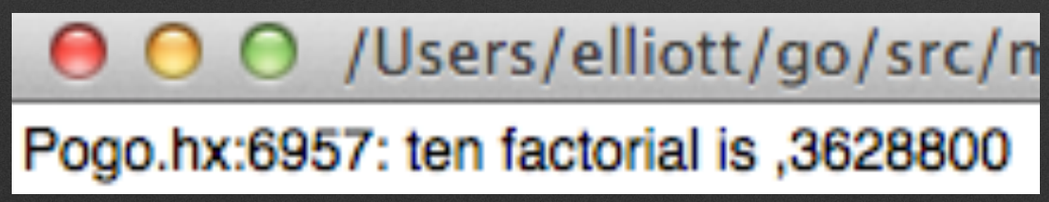
```
public function run():Pogo_main_fact {
while(true){
switch(_Next) {
case 0: // entry
this.setLatest(9,0);
this.SubFn0();
case 1: // if.then
this.setLatest(9,1);
_res= 1;
this._incomplete=false;
Scheduler.pop(this._goroutine);
return this; // return 1:int *ssa.Return @ glug.go:11:3
case 2: // if.done
this.setLatest(11,2);
this.SubFn1();
_SF1=Pogo_main_fact.call(this._goroutine,[],_t1);
_Next = -1;
return this;
case -1:
this.setLatest(13,-1);
_t2=_SF1.res();
 // _t2 = fact(t1) *ssa.Call @ glug.go:13:15
this.SubFn2();
_res= _t3;
this._incomplete=false;
Scheduler.pop(this._goroutine);
return this; // return t3 *ssa.Return @ glug.go:14:2
default: throw "Next?";}}}
private inline function SubFn0():Void {
var _t0:Bool;
_t0=(p_n==0); // _t0 = n == 0:int *ssa.BinOp @ glug.go:10:7
_Next=_t0 ? 1 : 2; // if t0 goto 1.if.then else 2.if.done *ssa.If near glug.go:10:7
}// end SubFn0
private inline function SubFn1():Void {
_t1=(p_n-1); // _t1 = n - 1:int *ssa.BinOp @ glug.go:13:17
}// end SubFn1
private inline function SubFn2():Void {
_t3=(p_n*_t2); // _t3 = n * t2 *ssa.BinOp @ glug.go:13:9
}// end SubFn2
```

```
func fact(n int) int {
    if n == 0 {
        return 1
    }
    return n * fact(n-1)
}
```

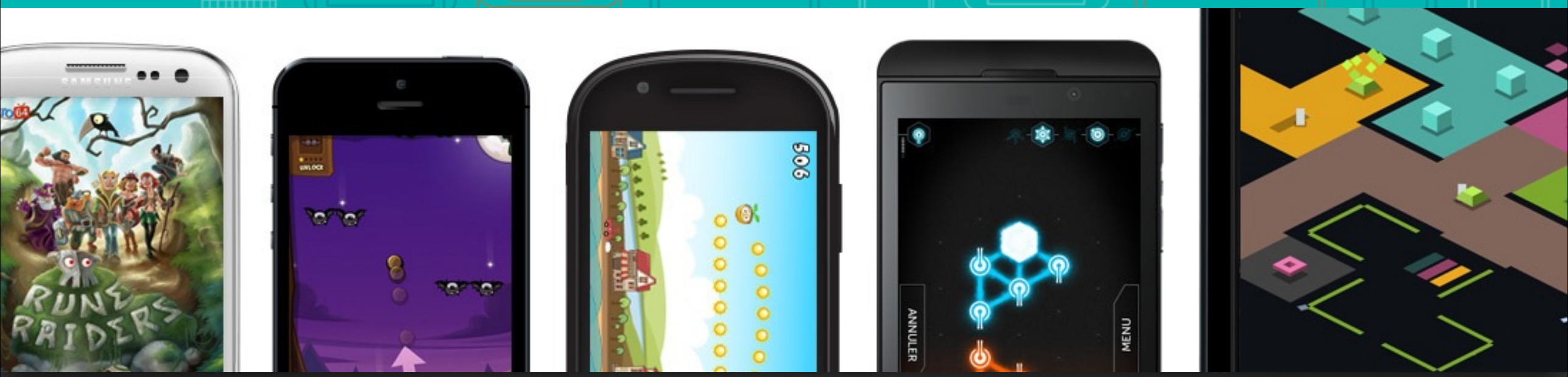
```
# Name: main.fact
# Package: main
# Location: glug.go:9:6
func fact(n int) int:
.0.entry:
   t0 = n == 0:int
   if t0 goto 1.if.then else 2.if.done
.1.if.then:
   return 1:int
.2.if.done:
   t1 = n - 1:int
   t2 = fact(t1)
   t3 = n * t2
   return t3
```

Cross-Compilation

```
Go native (go run glug.go):
ten factorial is  3628800

using go.tools/ssa/interp (go run interp.go):
ten factorial is  3628800

cross-compile Go->Haxe->Node/JS (node<pogo.js):
Pogo.hx:2716: ten factorial is ,3628800

cross-compile Go->Haxe->C++ (./cpp/Pogo):
Pogo.hx:2716: ten factorial is ,3628800

cross-compile Go->Haxe->Java (java -jar java/java.jar):
Pogo.hx:2716: ten factorial is ,3628800

cross-compile Go->Haxe->C# (mono ./cs/bin/cs.exe):
Pogo.hx:2716: ten factorial is ,3628800

cross-compile Go->Haxe->PHP (php php/index.php):
Pogo.hx:2716: ten factorial is ,3628800

cross-compile Go->Haxe->Flash (using flash player to test swf file):
```

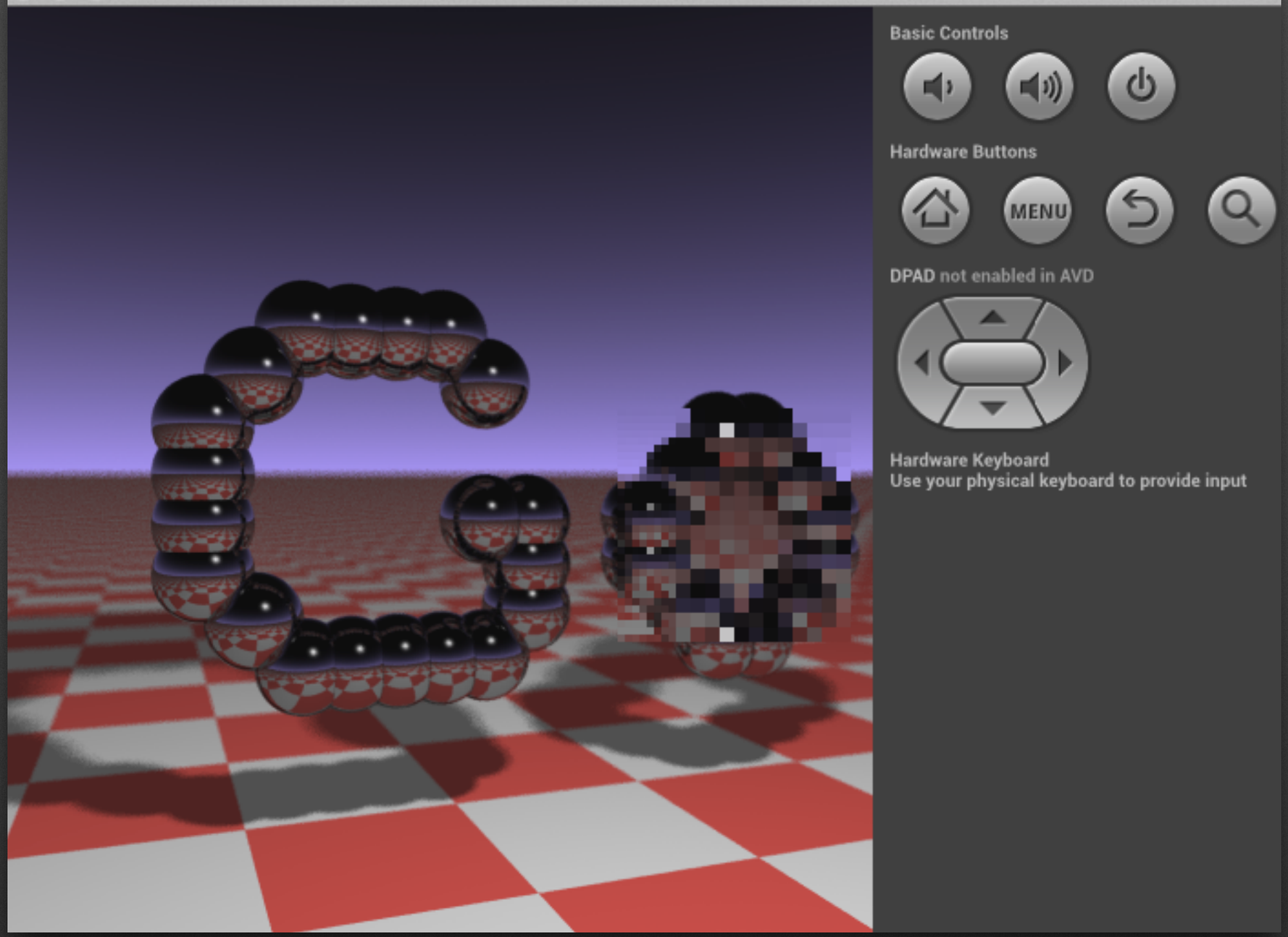


openFL



openfl.org

A Haxe based “cross-platform framework that targets Windows, Mac, Linux, iOS, Android, BlackBerry, Flash and HTML5” based on the Flash API

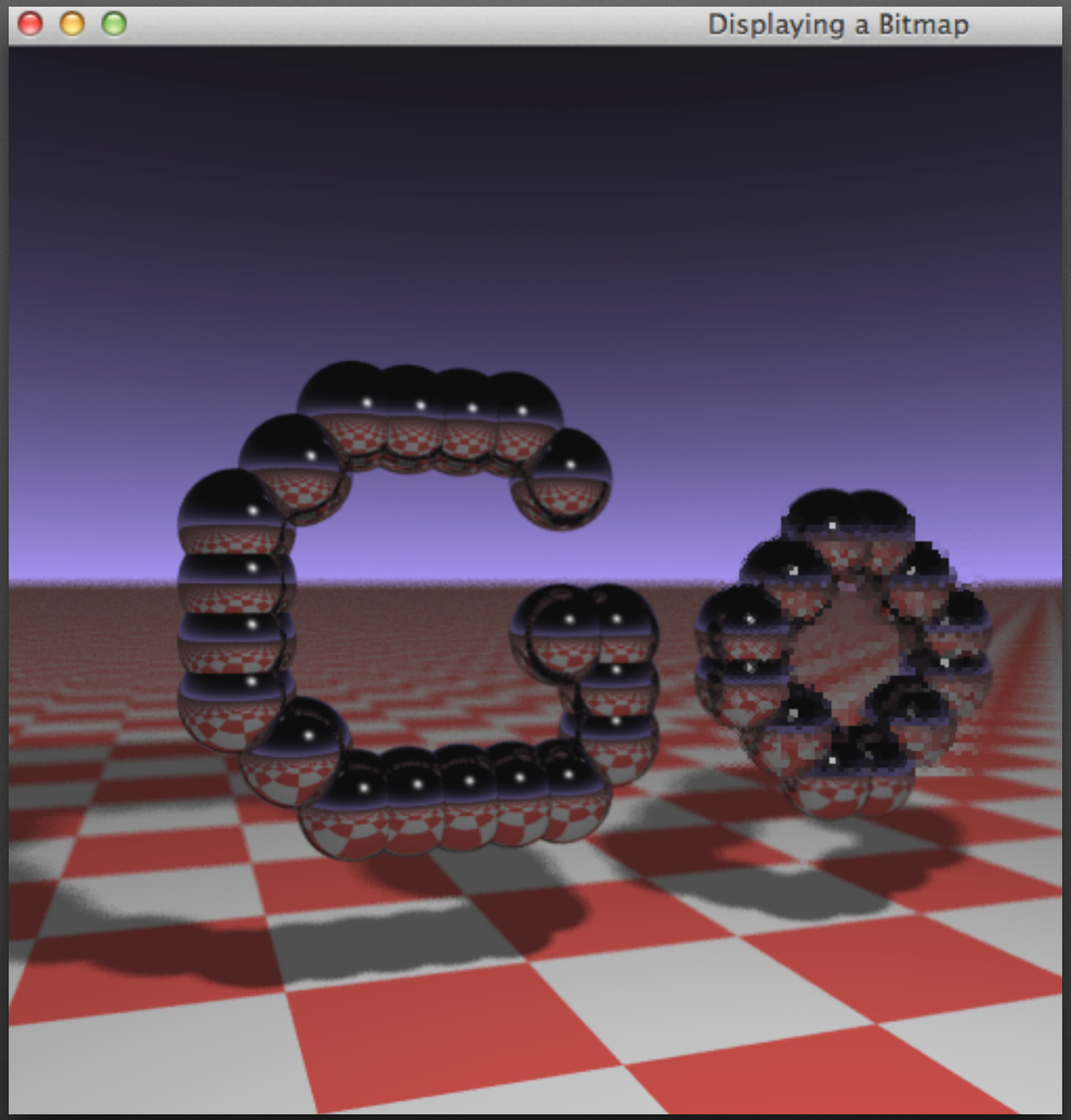


Example: Go -> Haxe + OpenFL -> Android

adapted from: github.com/kid0m4n/gorays “Ray tracing”

key project issues

- Speed vs Go is not too bad for simple mathematics (e.g. Java 125%; C++ 151%; C# 252%; JS/Node 270%), but some other operations are currently an order of magnitude slower
- Go library code size on the client side (e.g. unicode is huge and used widely by other libraries)
- Understanding how best to open-source the project



I hope to launch the project at a future Go London User Group meeting

Any Questions?

- Slides available at speakerdeck.com/elliott5
- interp.go available at gist.github.com/elliott5
- library code available at code.google.com/p/go.tools

Email me at elliott.stoneham@gmail.com

Image Sources

- http://commons.wikimedia.org/
- http://www.wikipedia.org/
- http://www.bbc.co.uk/
- Project related images from relevant project sites
- My picture by www.facebook.com/mstarsphotography
- Other images self-created