



# Go is not just on your server, it's in your browser

The story of and introduction to Vugu (Go+WebAssembly UI library)

# In This Talk

- The origins of Vugu
- How it works
- “Reactive” UIs
- .vugu files
- Features
- The Road Ahead




# The Origins of Vugu

# Modern Web Development (Front End)

- Most web development is done in JS
- But not by choice
- WebAssembly changes the game
- I've written my fair share of C, C++, Java, Python, PHP, JS and a few others, and in my experience/opinion none of it comes close to Go's productivity.
- So why can't we use it to write a UI
- We can!
- But it also means learning some new patterns (more on this in a bit)



# Languages

- Many people don't write in C or C++ because they want to, it's simply the viable choice for the situation (some things require manual memory management, for example)
  - Rust is moving in on some C/C++ territory, but it's still low level.
  - Go is moving in on Java/Python/etc territory. The application-level-i-need-to-get-this-done-no-messing-around space.
  - Disclaimer: There will always be language wars. But this is my guess about how this will play out.
  - I think we're poised to see many decades of large applications written and supported in Go.
  - And I think it can be (will be/is!) viable on the front-end too.
- 

# A Quick Trip Down Memory Lane

Today, if you want to look up some information, you probably use something like...



# A Quick Trip Down Memory Lane (2020)

Today, if you want to look up some information, you probably use something like:



(Or maybe you're talking to your car or refrigerator or something.)



# A Quick Trip Down Memory Lane (2010)

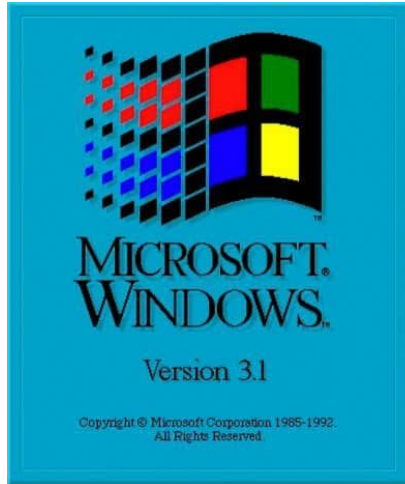
And before that, probably an Internet-connected computer was the next best thing, maybe one of these things:



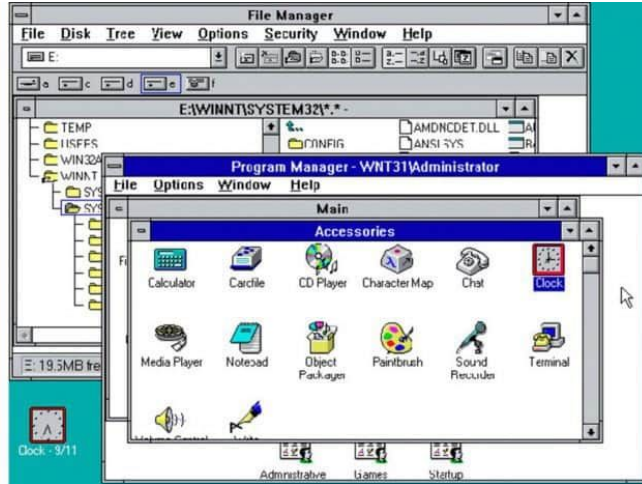


# A Quick Trip Down Memory Lane (1990-2000)

And before the Internet was everywhere, we had stuff like LANs workstations and corporate servers.



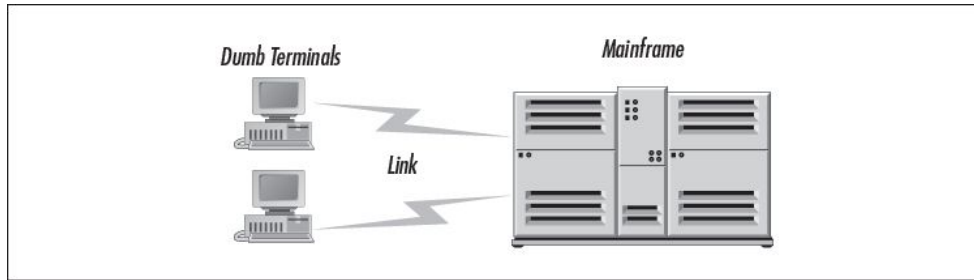
Old school Windows



Beige tower servers

# A Quick Trip Down Memory Lane (1960-1980)

And if we go back even more, we get things like mainframes and “dumb terminals”:



Old stuff only people with awesome beards remember.



# A Quick Trip Down Memory Lane: What's the pattern...

- Yes, everything is getting smaller, faster and cheaper, Moore's law, blah, blah, etc.
- But there's another trend... One that is directly relevant to WebAssembly...



# A Quick Trip Down Memory Lane: What's the pattern...

*Computing tends to start centralized, and then as it becomes more cost-effective, moves out toward individual connected devices.*

*Or, put another way, as devices closer to the user become more powerful, people tend to run more stuff on them.*

- Terminal -> Workstation
- Desktop -> Mobile  
(Flip Phone -> Smartphone, Tablets)
- HTML -> Modern Browser




# WebAssembly

Wasm is not just about making web apps faster.

It's about moving the browser another major step closer to becoming a universal UI - a sort of VM for user interfaces. (Wasm will probably be a big deal on the backend too, we'll see.)

It is doubtful JavaScript will ever go away. But as support for Wasm improves and the ability to write user interfaces in other languages becomes more and more viable, it will likely unseat JS's current de-facto position as king of the web UI.



# How Vugu Works

# Vugu is...

- An attempt to “see if we can make something like Vue in Go”
- A code generation tool .vugu (HTML-like markup) -> .go
- A library to perform syncing with the browser DOM.



# Vugu Files

- Only used **during codegen**
- Translate directly to **Go structs**.
- Generated *Build(in) (out)* method **reconstructs virtual DOM** from .vugu file. HTML-like markup is useful.
- **Declarative attributes** like *vg-if* and *vg-for* describe markup branches and loops.
- All **expressions are Go**, not another template language.
- **Script block** for other Go code or just use separate file in package.

```
<div class="mt-5">
  <h1>Welcome to Vugu</h1>
  <p class="lead">This page is being rendered via
    <a @click='event.PreventDefault();c.ShowWasm=!c.ShowWasm'
      written in
    <a @click='event.PreventDefault();c.ShowGo=!c.ShowGo' href
        using
    <a @click='event.PreventDefault();c.ShowVugu=!c.ShowVugu'
      and built with
    <a @click='event.PreventDefault();c.ShowTinygo=!c.ShowTiny
  </p>
```



# Go Source Code

```
func (c *Root) Build(vgin *vugu.BuildIn) (vgout *vugu.BuildOut) {  
  
    vgout = &vugu.BuildOut{}  
  
    var vgiterkey interface{}  
    _ = vgiterkey  
    var vgn *vugu.VGNode  
    vgn = &vugu.VGNode{Type: vugu.VGNodeType(3), Namespace: "", Data: "div"  
    vgout.Out = append(vgout.Out, vgn) // root for output  
    {  
        // ...  
        vgparent.AppendChild(vgn)  
        vgn = &vugu.VGNode{Type: vugu.VGNodeType(3), Namespace: "",  
        vgparent.AppendChild(vgn)  
        vgn.SetInnerHTML(vugu.HTML("Welcome to Vugu"))  
        vgn = &vugu.VGNode{Type: vugu.VGNodeType(1), Data: "\n"  
        vgparent.AppendChild(vgn)  
        vgn = &vugu.VGNode{Type: vugu.VGNodeType(3), Namespace: "",  
        vgparent.AppendChild(vgn)
```

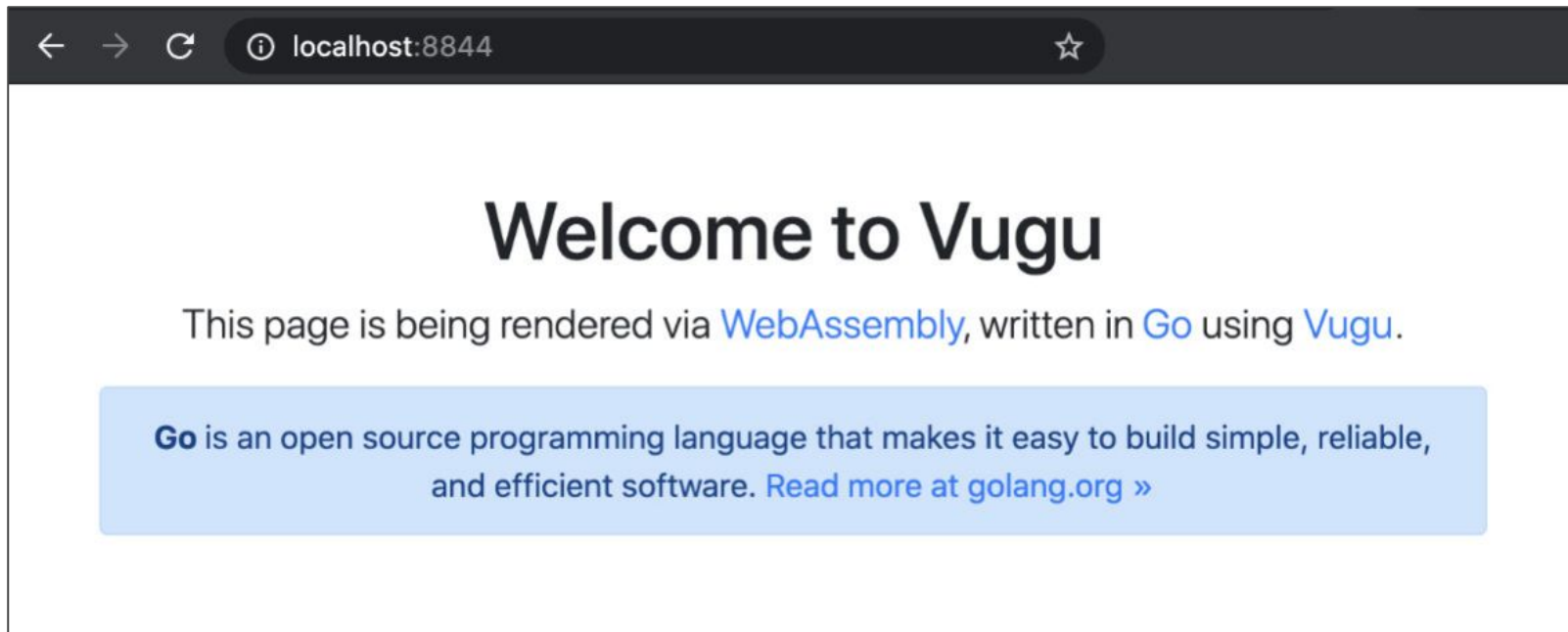
- You don't normally have to do anything with generated files.
- UI Components are **just structs**.
- **Vugu library does the heavy lifting** and provides a representation for virtual DOM.
- And otherwise it's **just a regular Go project**.
- Other than codegen (called from `go generate`) **no special build tools**.

# Wasm Binary Output

- The Go compiler produces a **WebAssembly binary** (.wasm file)
- **Disclaimer:** Binaries can be large, but this will improve with time. (TinyGo is also an option.)
- There are some **useful dev tools** to make the process smoother. But in general, “magic” is avoided.

```
00 61 73 6d 01 00 00 00 00 f2 80 80 80 00 0a 67 |.asm.....gl
6f 2e 62 75 69 6c 64 69 64 ff 20 47 6f 20 62 75 |o.buildid. Go bul
69 6c 64 20 49 44 3a 20 22 31 78 76 76 6e 51 61 |ild ID: "1xvvnQaI
31 46 67 67 68 46 32 49 62 5f 6d 74 61 2f 42 35 |lFgghF2Ib_mta/B5I
4a 5f 30 55 72 66 4e 32 6e 78 35 6a 6e 36 61 47 |lJ_0UrfN2nx5jn6aGI
6a 36 2f 4e 69 34 4e 67 33 35 33 68 75 75 66 5a |lj6/Ni4Ng353huufZI
7a 37 78 6d 47 34 78 2f 73 61 65 4c 6b 55 45 79 |lz7xmG4x/saeLkUEyl
79 73 41 41 53 76 59 5a 4d 54 48 51 22 0a 20 ff |lysAASvYZMTHQ". .l
61 68 6c 61 76 69 57 65 6c 63 6f 6d 65 20 74 6f |lahlaviWelcome toI
20 56 75 67 75 5d 0a 09 6d 6f 72 65 62 75 66 3d |l Vugu]..morebuf=I
7b 70 63 3a 61 73 79 6e 63 70 72 65 65 6d 70 74 |l{pc:asyncpreemptI
6f 66 66 62 61 64 20 64 65 62 75 67 43 61 6c 6c |loffbad debugCallI
56 31 65 6c 65 6d 20 73 69 7a 65 20 77 72 6f 6e |lVielem size wronI
67 66 6f 72 63 65 20 67 63 20 28 69 64 6c 65 29 |lgforce gc (idle)I
6e 74 69 6d 65 2e 72 65 73 65 74 4d 65 6d 6f 72 |lntime.resetMemorI
79 44 61 74 61 56 69 65 77 00 01 02 67 6f 10 72 |lyDataView...go.rI
75 6e 74 69 6d 65 2e 77 61 73 6d 45 78 69 74 00 |luntime.wasmExit.I
01 02 67 6f 11 72 75 6e 74 69 6d 65 2e 77 61 73 |l..go.runtime.wasI
```

And viola, your mix of markup and Go runs in-browser!



# “Reactive” User Interfaces

# Reactive Web Apps

- **Modern apps** can be complicated
- With things like shared state management, data binding, the asynchronous nature of fetching data from a server, **sometimes the core idea gets lost.**
- What does **“reactive”** even mean?
- It means your **view is a function of state.**
- Or, more clearly, you **declare that your view should look a certain way** based on a set of variables, and **the library/framework makes it so.**

Old non-reactive way

```
clicked() {  
  $('#somediv').hide()  
}
```

Reactive way

```
func (c *Comp) clicked() {  
  c.showSomediv = false  
}
```

```
<div vg-if="c.showSomediv">
```


# Vugu is a “Reactive Web Framework”

- But it's otherwise **just a Go library**.
- We want to **keep magic to a minimum**, if any at all.
- There will be **more patterns that emerge**, stay tuned.



# Features

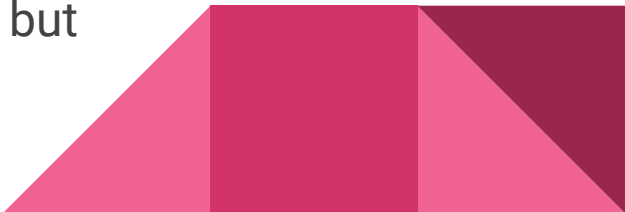
# What Vugu Does Right Now

- Single-file components
  - DOM Events
  - CSS
  - vg-if, vg-for, vg-content
  - Dynamic attributes
  - Static components
  - Dynamic components (events)
  - Lifecycle callbacks (Init, Compute, Rendered, Destroy)
  - Some TinyGo support (one order of magnitude smaller binaries)
  - Server-side Rendering
  - URL Router
  - Modification tracking for faster renders
  - Slots
  - Form support (wip)
  - Hot reloading
  - Some docs
  - Some examples
- 



# The Road Ahead

# Vugu Challenges

- **Component Library** - Vugu really needs a good component library and ideally without undue dependence upon a specific CSS framework.
  - **Performance** - we need to see what happens with very large and complicated apps. Rendering optimizations will likely be needed. But so far so good.
  - **File size** - TinyGo is awesome but there is still a lot of work to be done.
  - For me, the biggest challenge is **finding and documenting good patterns**. State management is one of the big ones that comes up. Look out for more updates on this soon.
  - For now all of this is just me and others contributing, but we're looking at ideas for official sponsorship to help solidify Vugu's future.
- 

# The Future is Bright

- **JS will likely become unseated** as the main choice for web UI development in the coming years.
- **Go is awesome.** And it gets **more awesome** all the time.
- People are going to want to write **web UIs in Go.**
- **Vugu gets us closer!**



# Credits

## Photos:

- Smartphone - @newkemail on Flickr (no mods)
- Desktop Computer @Rebecca Freedman on Flickr (cropped)
- Mainframe diagram - O'Reilly Publishing
- VAX/VMS photo - Wikipedia

