

Building desktop apps with JS

by
Chris Rickard

@chrisrickard
<http://inoutput.io>

So why the hell would you do it?

(build desktop apps with webtech)

So compared to a “classic” desktop app...



Great frameworks...



YEOMAN



d3



BACKBONE.JS



Sass.

style up the world

THREE.JS



Haml



handlebars



RIVETS.JS



ANGULARJS
by Google

Derby

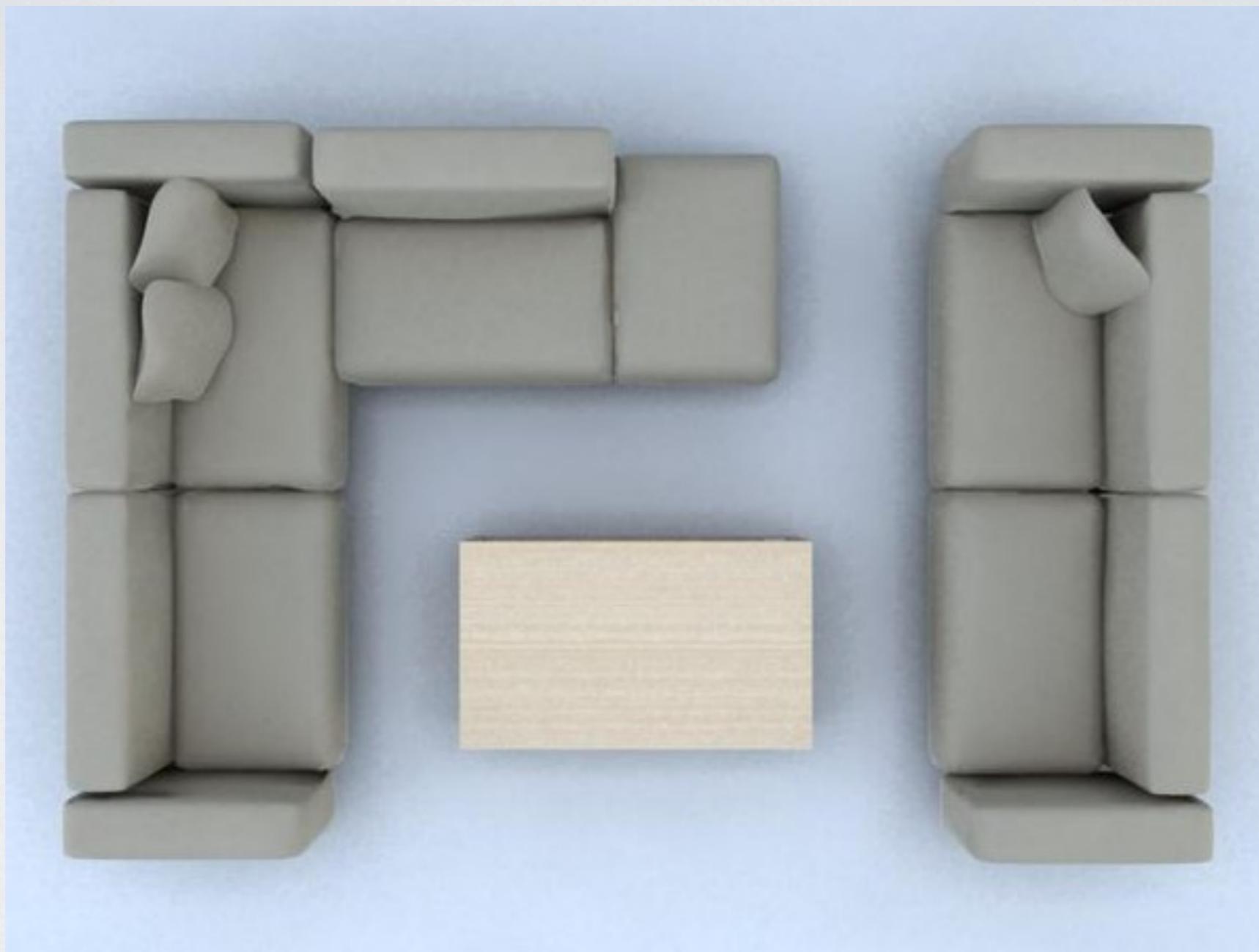


dōjō



Meteor JS

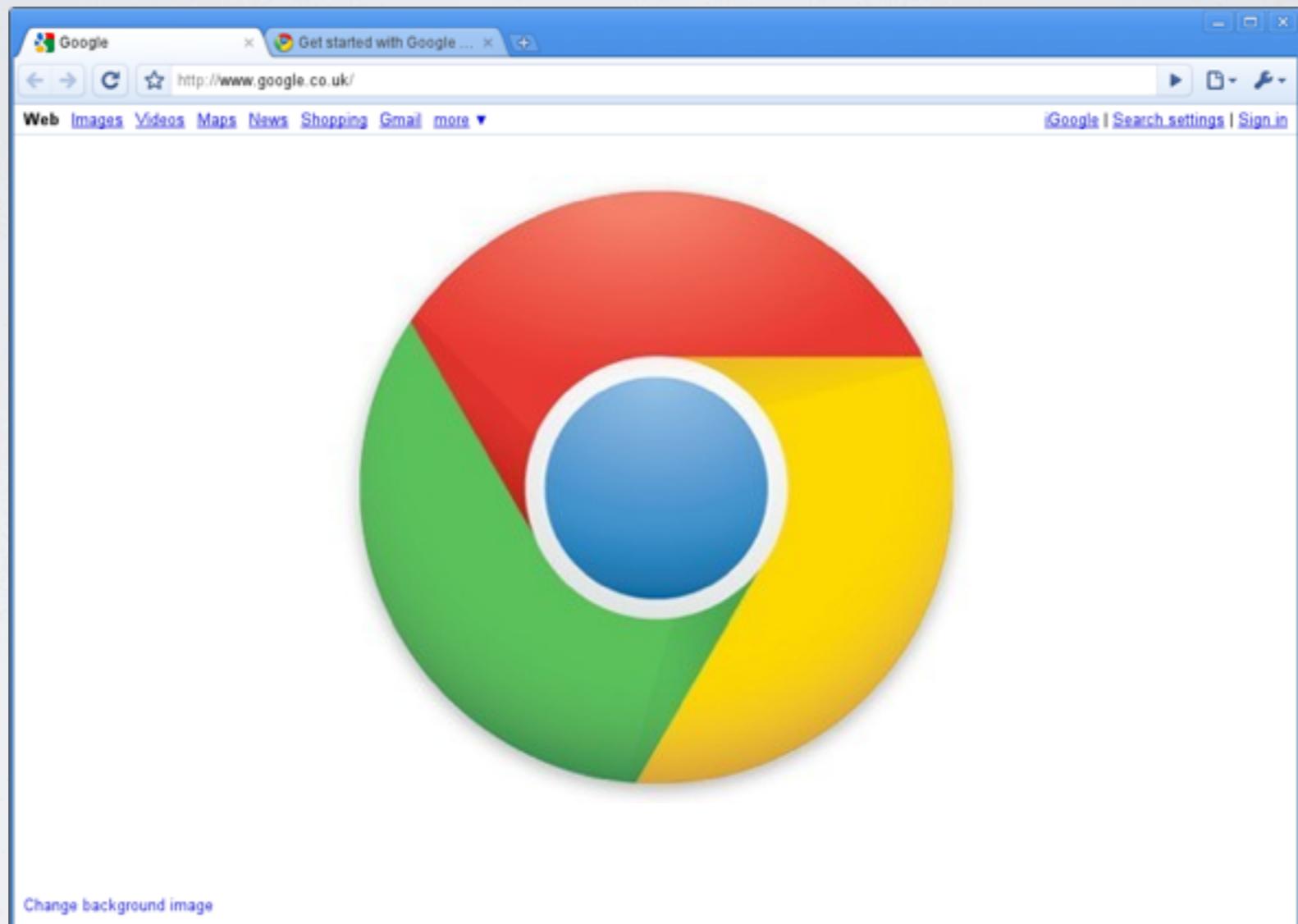
Modularity...



Better the devil you know...



... and compared to a Webapp?



Power

- Access to underlying OS
- File menu, system tray etc..
- No security restrictions
- Local file access
- No prompts for camera, audio, location etc
- No cross domain restrictions



Compatibility

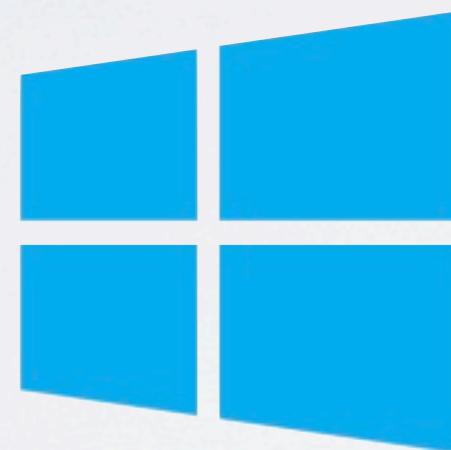
Just this



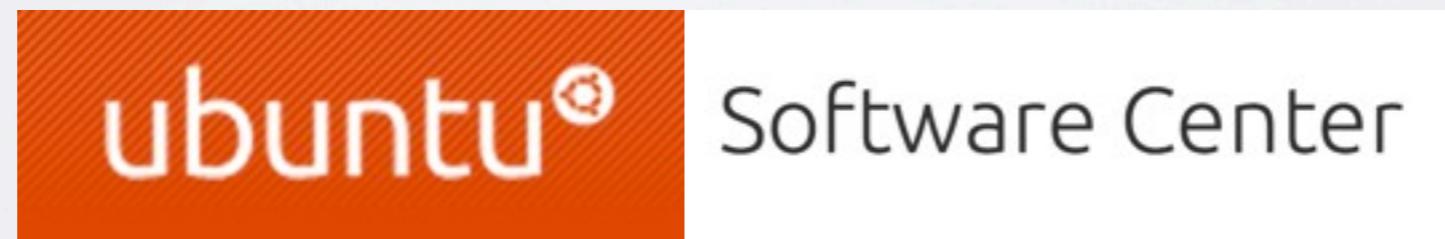
and not all of these suckers:



Distribution



**Windows 8
Store**



blah blah blah....

So who are building their apps in JS?

Wunderlist

The screenshot shows the Wunderlist application window. The main area displays a list titled "Birthday Party" with the following tasks:

- Invite friends (exclude Jan) - Yesterday
- Buy Balloons at Simon's - Tomorrow
- Bake the cake
- Hire the clown (the funny one)
- Send Invitation - Today

A red callout bubble indicates "1 task is overdue". The sidebar on the right lists other task categories:

- Inbox (0)
- Homework (5)
- Work (13)
- Birthday Party** (5)
- Shopping List (8)

At the bottom, there are navigation buttons for Today (1), Tomorrow, Next 7 Days, Later, Without Date, Add List, and Help.

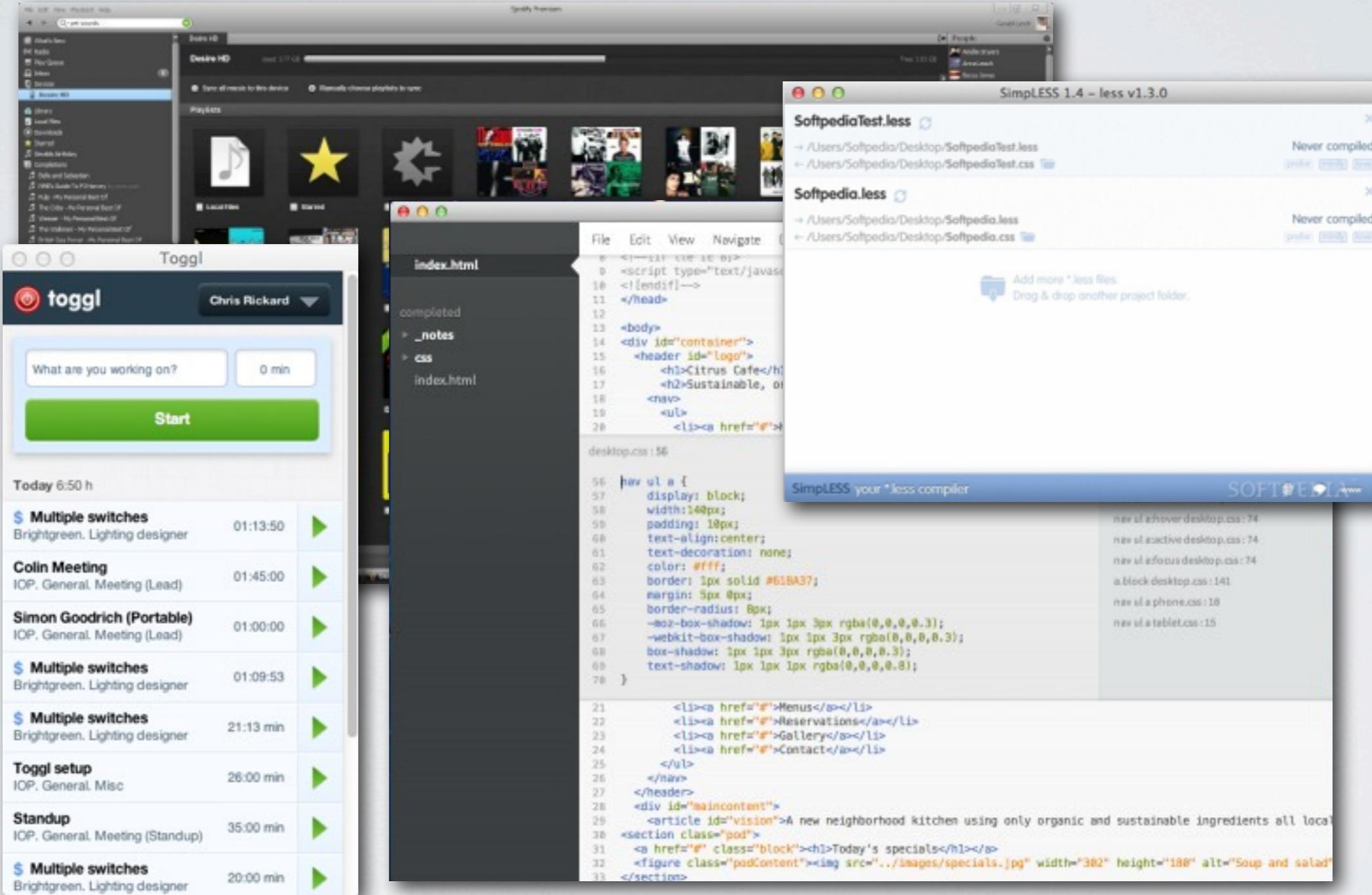
LightTable IDE

The screenshot displays the LightTable IDE interface with the following components:

- Code Editor:** On the left, a code editor window titled "Light Table" contains Clojure code. The code includes:
 - Server setup: `(ns asdf.server (:require [noir.server :as server]))` and `(server/load-views-ns 'asdf.views)`.
 - Main function: `(defn -main [& m] (let [mode (keyword (or (first m) :dev)) port (Integer. (get (System/getenv) "PORT" "8080"))] (server/start port {:mode mode :ns 'asdf})))`.
 - View configuration: `(ns asdf.views.common (:use [noir.core :only [defpartial]] [hiccup.page :only [include-css html5 include-js]]))`.
 - Defpartial: `(defpartial layout [& content] (html5 [:head [:title "asdf"] (include-js "https://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.min.js") (include-css "/css/reset.css") [:body [:div#wrapper content (include-js "/cljs/bootstrap.js")]]]))`.
 - Defpage: `(defpage "/1" [] "zoasdf")` and `(defpage "/4" [] "zoasdf")`.
- Sidebar:** A sidebar on the right lists namespaces and symbols. It shows:
 - Namespaces: asdf, asdf.server, asdf.views.common, asdf.views.welcome.
 - Symbols: /1, /2, /3, /4, cool, something.
- Results Panel:** At the bottom, a "results" panel shows the output of a command, likely related to the defpartial definition. The output is:

```
<!DOCTYPE html>\n<html><head><title>asdf</title>\nsrc='https://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.min.js'\ntype='text/javascript'></script><link href='/\nrel='stylesheet'\ntime='2013-09-17T14:45:22Z'>
```

...and a bunch of others

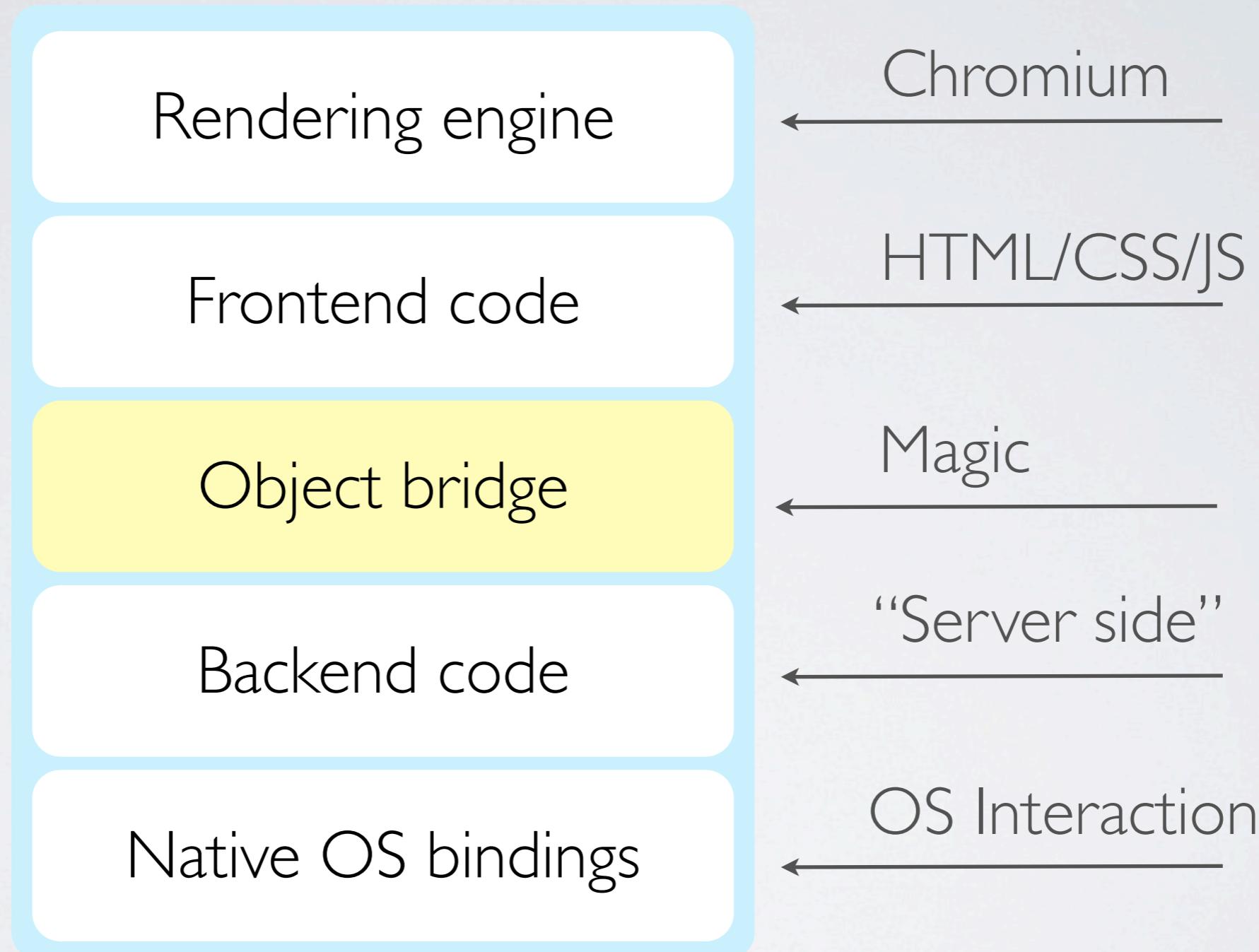


The current landscape

- AppJS
- node-webkit
- TideSDK
- Sencha Desktop Packager (osx & win)
- Adobe AIR (osx & win... and it's flash)

or Chromium Embedded Framework (CEF)

So how does it all work?





- Was previously “Titanium Desktop”
- Supports Ruby, Python & PHP
- Creates a bridge between chromium and libtide - marshalling, object mapping etc.
- After Titanium ditched it - picked up by OS
- Possibly needs a little bit more time...

<http://www.tidesdk.org>



- NodeJS
- Built on CEF (Chromium embedded framework)
- Bridges Chromium & node via IPC
- Does talk “across the wire” - but hooks into the custom appjs:// scheme to handle requests

<http://appjs.org>



nodewebkit

- NodeJS
- Rad. Merged Node & Chromiums Event-loops
- Direct access, same thread
- Sync's with newest Chromium build every few days
- Backed by Intel

<https://github.com/rogerwang/node-webkit>



nodewebkit

Let's take a look at a demo...

And the code?

The beginning: package.json

```
1 {
2   "name": "Example",
3   "main": "index.html",
4   "window": {
5     "name": "example",
6     "title": "Example",
7     "position": "center",
8     "width": 700,
9     "height": 500,
10    "toolbar": false,
11    "show": false
12  }
13 }
14 }
```

And the code?

Adding a menubar

```
1 var gui = require('nw.gui');
2
3 var main_menu = new gui.Menu({ type: 'menubar' });
4 var submenu = new gui.Menu();
5
6 submenu.append(new gui.MenuItem({ type: 'checkbox', label: 'You' }));
7 submenu.append(new gui.MenuItem({ type: 'checkbox', label: 'Can' }));
8 submenu.append(new gui.MenuItem({ type: 'checkbox', label: 'Tick' }));
9 submenu.append(new gui.MenuItem({ type: 'checkbox', label: 'These items' }));
10
11 main_menu.append(new gui.MenuItem({ label: 'Menu', submenu: submenu }));
12 main_menu.append(new gui.MenuItem({ icon: 'img/cut.png', label: 'Cut' }));
13 main_menu.append(new gui.MenuItem({ icon: 'img/apple.png', label: 'Play' }));
14 main_menu.append(new gui.MenuItem({ icon: 'img/tick.png', label: 'Tick' }));
15
16 gui.Window.get().menu = main_menu;
```

And the code?

External process interaction

```
1 | var child_process = require('child_process');
2 | var proc = process.exec('ping -c 5 reddit.com');
3 |
4 | proc.stdout.on('data', function (data) {
5 |   console.log(data);
6 | });
```

Packaging? Easy

- Zip app directory, and rename app.nw
- OSX - Copy node-webkit.app & place app.nw inside
- Windows - copy /b nw.exe+app.nw app.exe
- Linux - cat /usr/bin/nw app.nw > app && chmod +x app

but... apps are 24MB (compressed) for a hello world :(

And thats pretty much it...

1. If your building a desktop app: consider using JS
2. A bunch of neat frameworks already out there
3. But personally I think Webapps will eventually win

Thanks... Questions?!



@chrisrickard