

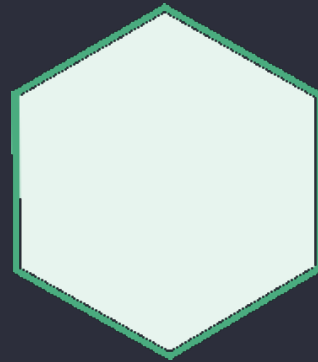


# Electron: The Language Circus

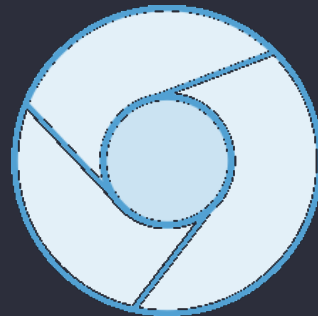
Shelley Vohr  
@codebytere

# Electron

**Node.js** for  
filesystems  
and networks



**Chromium**  
for making  
web pages

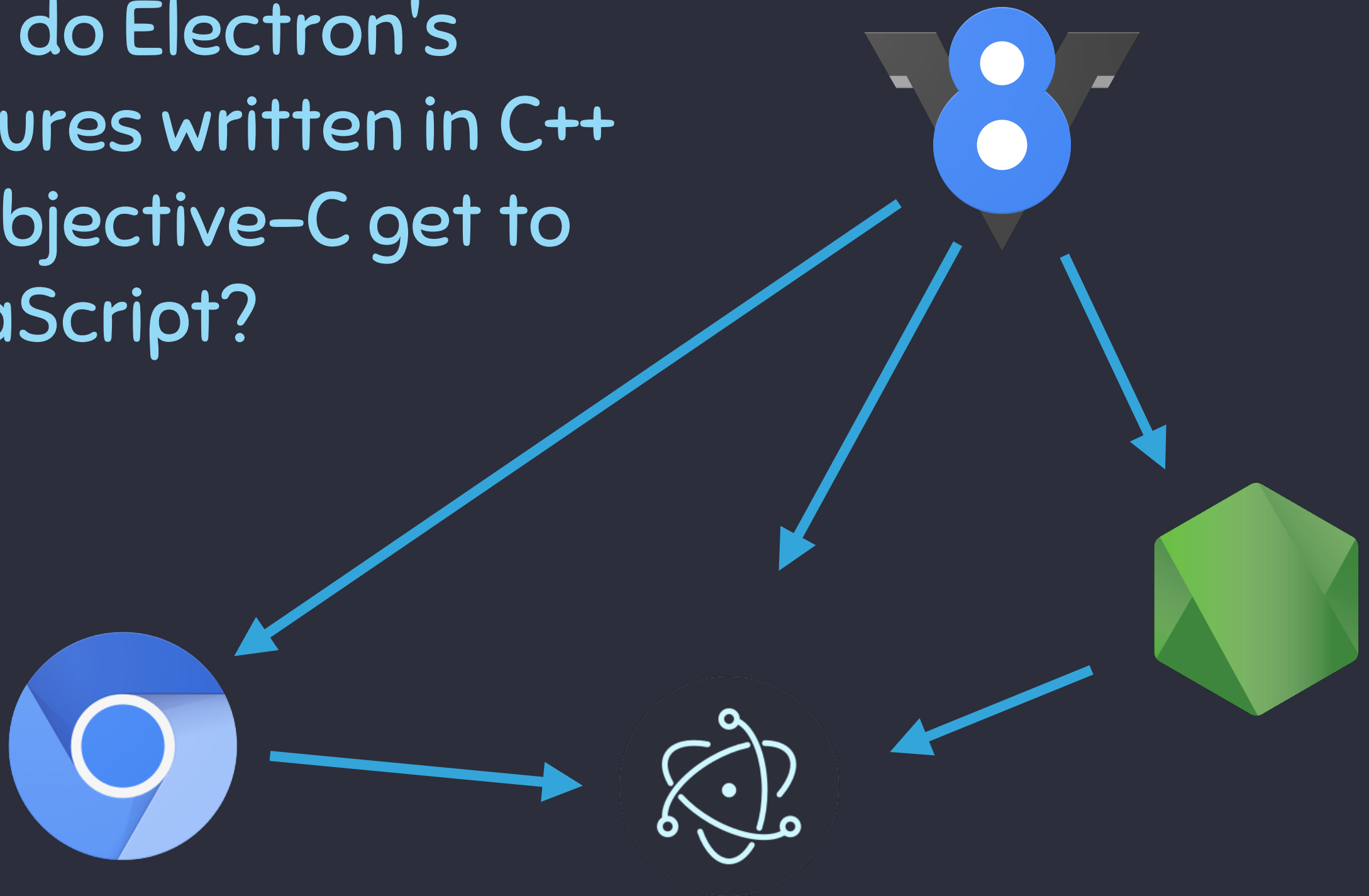


**Native APIs**  
for three  
systems



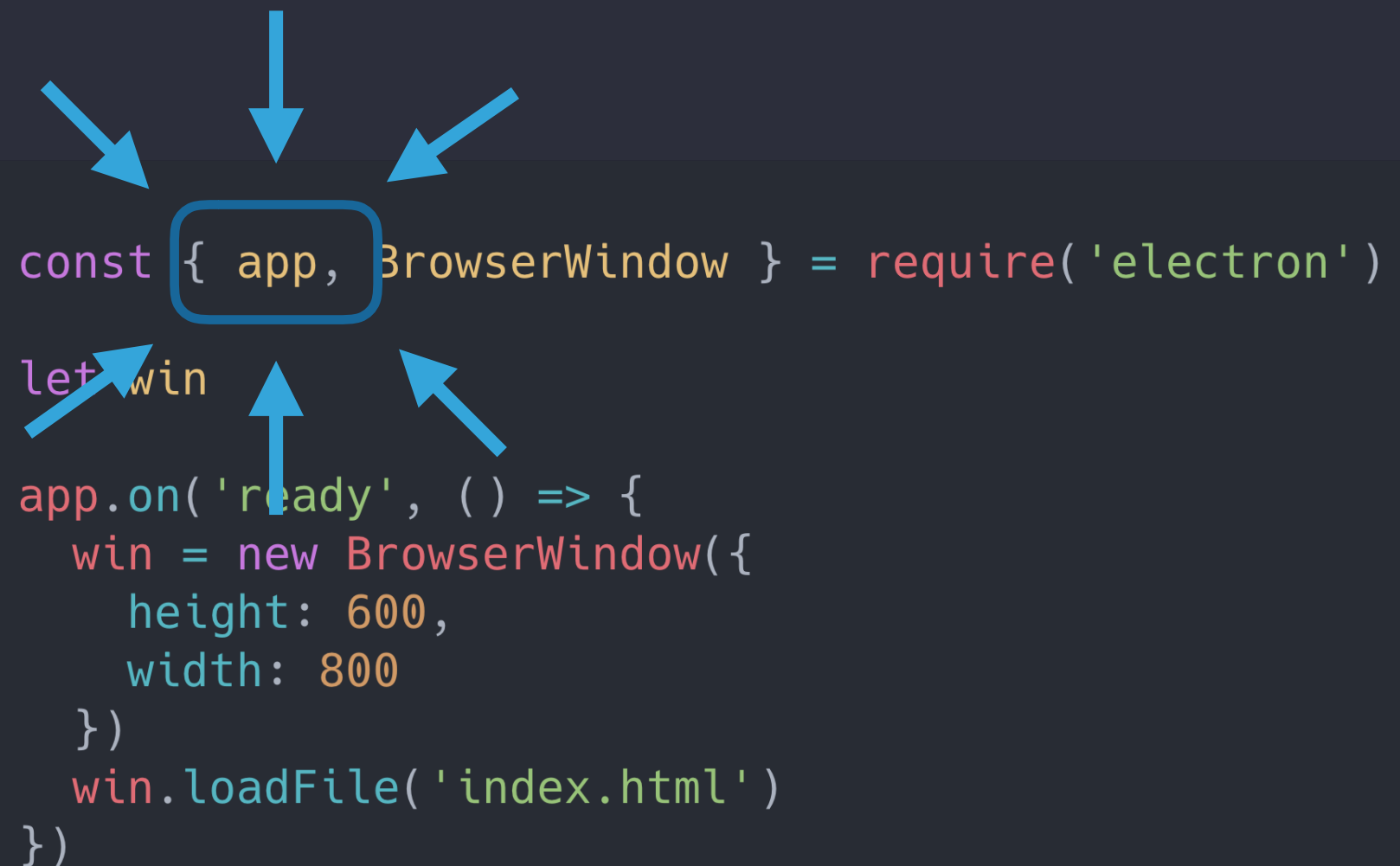
# Today's Journey

How do Electron's  
features written in C++  
or Objective-C get to  
JavaScript?



# The **app** Module

Manages your  
application's  
**Lifecycle**



The diagram illustrates the usage of the `app` module from the `electron` package. It features a central code block with several blue arrows pointing to specific parts of the code:

- Three arrows point to the `app` property in the `require('electron')` destructuring assignment.
- One arrow points to the `win` variable in the `let win` declaration.
- Two arrows point to the `app` and `win` objects in the `app.on('ready', ...)` event listener.

```
const { app, BrowserWindow } = require('electron')  
  
let win  
  
app.on('ready', () => {  
  win = new BrowserWindow({  
    height: 600,  
    width: 800  
  })  
  win.loadFile('index.html')  
})
```

# The **app** Module

```
import * as path from 'path'

import * as electron from 'electron'
import { EventEmitter } from 'events'

const commandLine = process.electronBinding('command_line')
const { app, App } = process.electronBinding('app')
```

The diagram illustrates the dependencies of the code. Arrows point from the following elements to the `process.electronBinding` calls in the last two lines of code:

- From `process` in `process.electronBinding('command_line')` to `process` in `process.electronBinding('app')`.
- From `electron` in `process.electronBinding('command_line')` to `electron` in `process.electronBinding('app')`.
- From `process` in `process.electronBinding('app')` to `process` in `process.electronBinding('command_line')`.
- From `app` in `process.electronBinding('app')` to `app` in `process.electronBinding('command_line')`.
- From `App` in `process.electronBinding('app')` to `App` in `process.electronBinding('command_line')`.

# process.binding()

```
process.binding('native_module')
```



A binding type for accessing internal native modules; **not intended to be used by embedders** but we did it anyway!

< Node.js **v12.0.0**

# process.internalBinding()

```
process.internalBinding( 'native_module' )
```



A binding type for accessing internal native modules that are **not intended** to be used by user code.

It has a **separate cache object** and **modlist** in order to avoid collisions.

≥ Node.js **v8.8.0**

# process.\_linkedBinding()

```
process._linkedBinding( 'native_module' )
```



Embedders (that's us!)  
can use this type of  
binding for **statically  
linked native bindings!**

≥ Node.js **v8.8.0**



# ElectronBindings

```
process.electronBinding( 'app' )
```



```
process.electronBinding( 'native_electron_module' )
```



Google's open source high-performance JavaScript and WebAssembly engine, written in C++

Implements **ECMAScript** and **WebAssembly**

# ObjectTemplateBuilder

JavaScript objects without a  
**dedicated constructor  
function** and **prototype**

Uses **Object[.prototype]**

```
function fruits () {  
    this.name = 'fruit one'  
}  
  
function apple () {  
    fruits.call(this)  
}  
  
apple.prototype = Object.create(fruits.prototype)  
const app = new apple()
```

# native\_mate



A fork of Chromium's **gin** library

Makes it easier to **marshal types** between C++ and JavaScript

# ObjectTemplateBuilder

```
mate::ObjectTemplateBuilder(isolate, prototype->PrototypeTemplate())  
    .SetMethod("getGPUInfo", &App::GetGPUInfo)
```

# Setting Methods

```
.SetMethod( "getGPUInfo", &App::GetGPUInfo )
```



```
.SetMethod( "methodName", &Namespace::NativeImpl )
```

# Setting Methods

```
function App {}  
App.prototype.methodName = function () {  
    // implementation here  
}
```

# Setting Properties

```
. SetProperty( "webContents", &BrowserWindow::GetWebContents )
```



```
. SetProperty( "propertyName", &Namespace::GetterFn )
```



# Setting Properties

```
function App {}  
Object.defineProperty(App.prototype, 'propertyName', {  
  get() {  
    return _propertyName  
  }  
})
```

# Setting Properties

```
. SetProperty( "body", &Notification::GetBody, &Notification::SetBody )
```



```
. SetProperty( "propertyName", &Namespace::GetterFn, &Namespace::SetterFn )
```

# Setting Properties

```
function App {}  
Object.defineProperty(App.prototype, 'propertyName', {  
  get() {  
    return _propertyName  
  }  
  set(newPropertyValue) {  
    _propertyName = newPropertyValue  
  }  
})
```

# THANK YOU!

