Shaping Electron

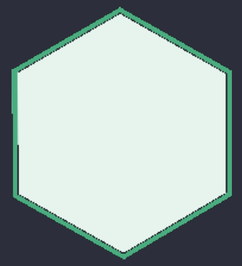
We Get By With a Little Help From Our Friends

Shelley Vohr
@codebytere

How do Electron's dependencies work together, and how do we use them to enable you to write the applications you do?

Electron

Node.js for filesystems and networks



Chromium for making web pages

Native APIs for three systems

The Main Players

V8

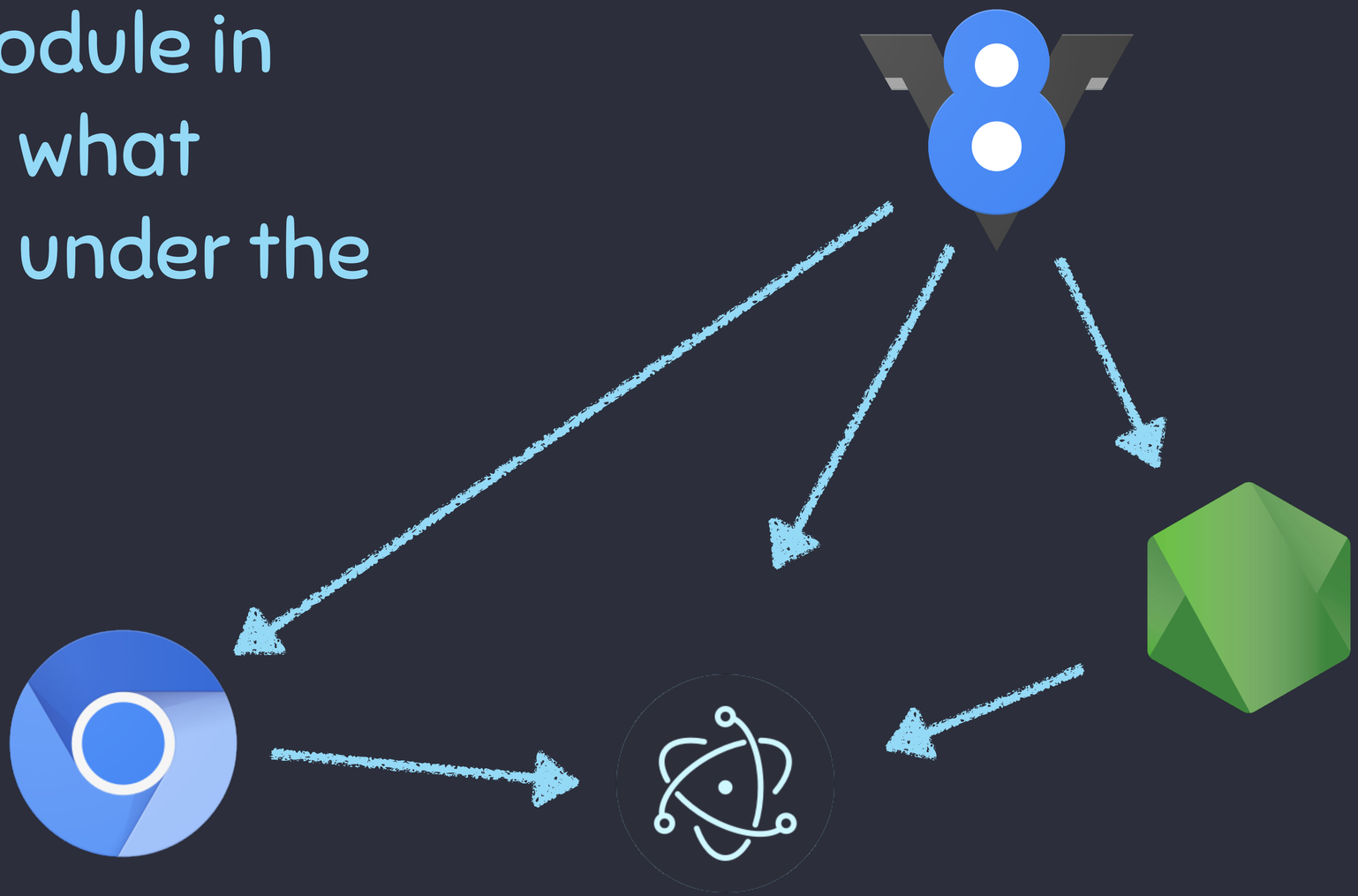
Node.js

Chromium

gin

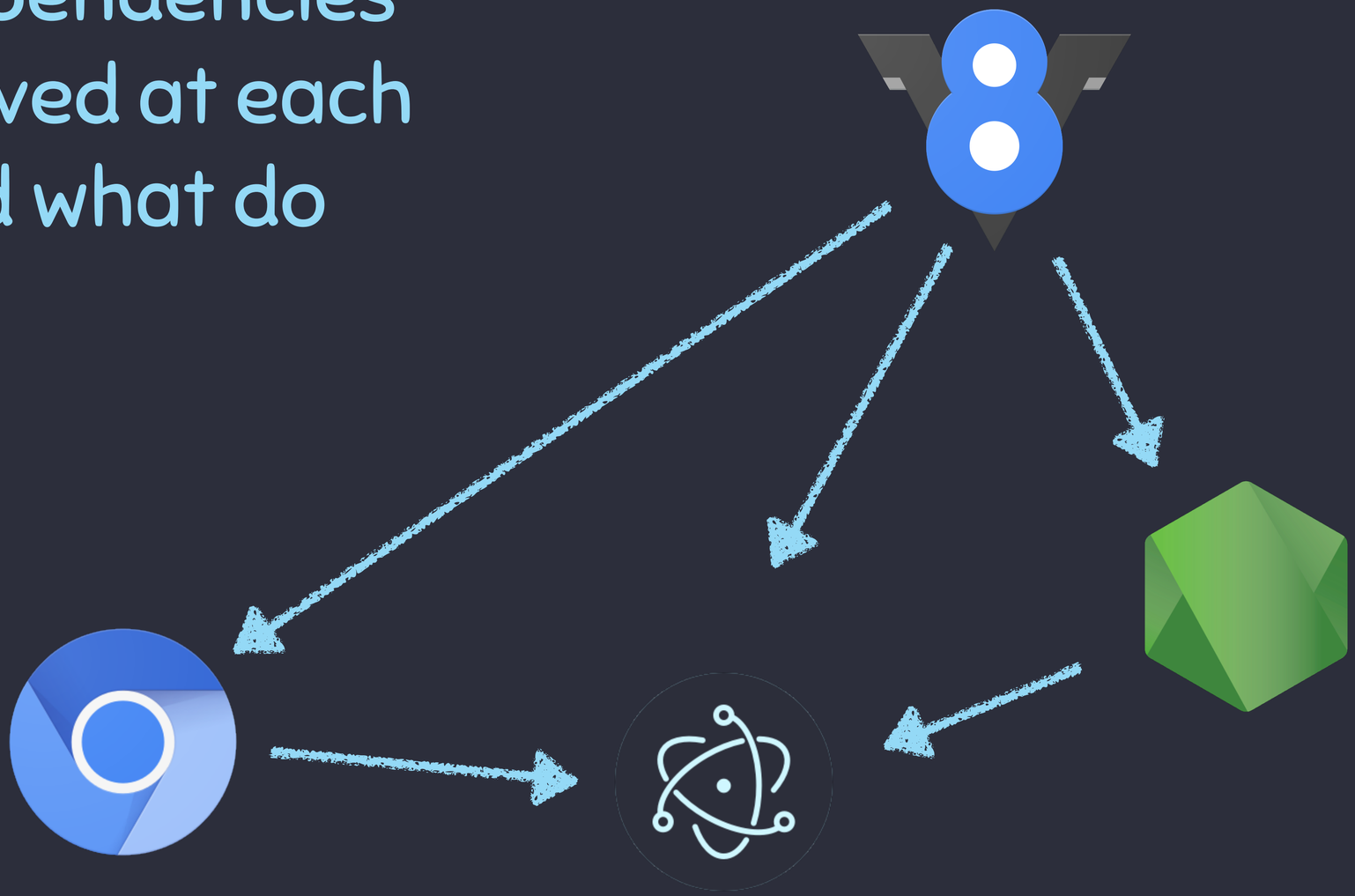
Today's Journey

When you write code with a module in Electron, what happens under the hood?



Today's Journey

What dependencies are involved at each step, and what do they do?

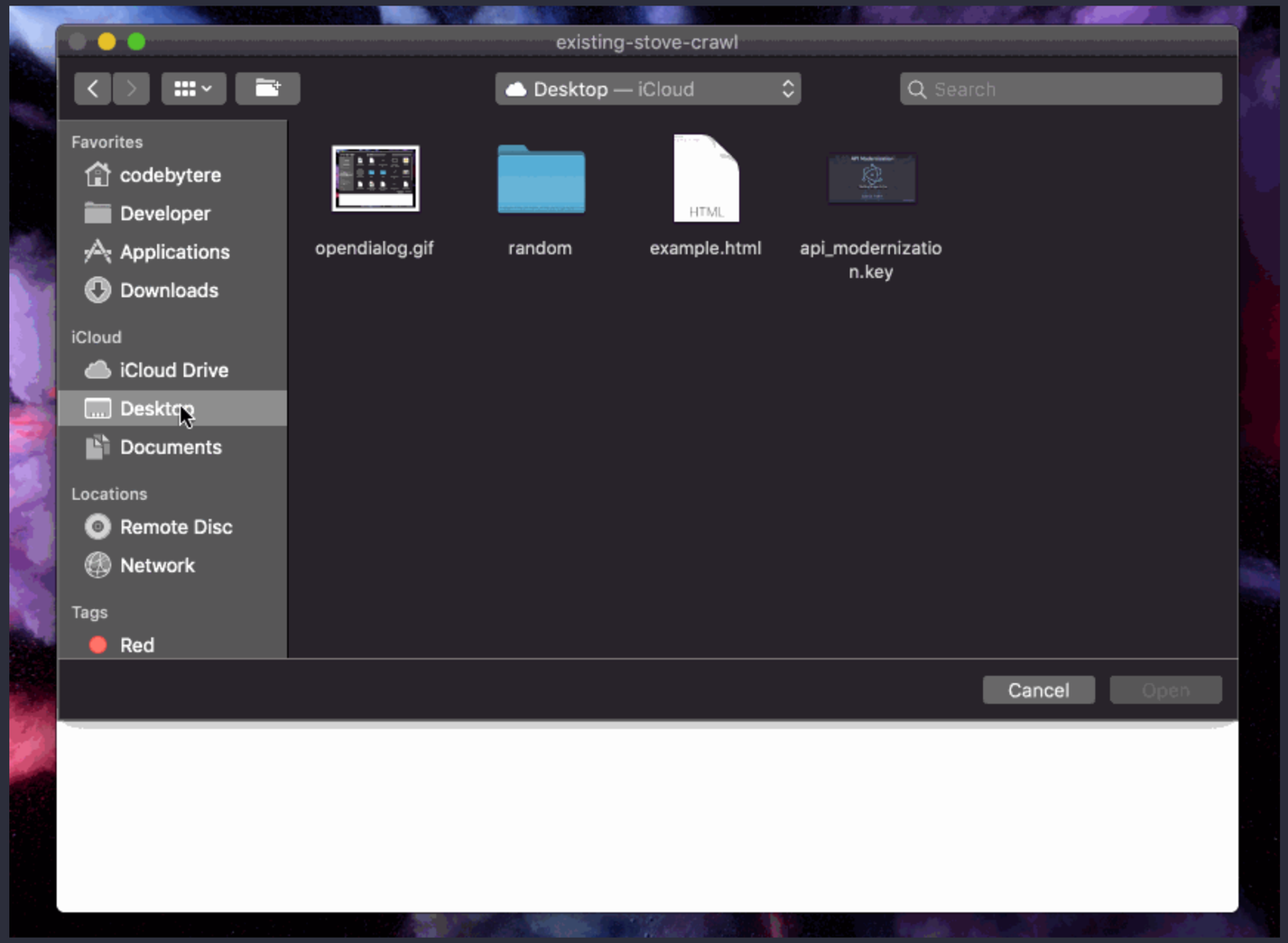


The dialog Module

Allows you to open **native dialogs** across **different platforms**

```
const { app, BrowserWindow, dialog } = require('electron')

let window = null

app.on('ready', async () => {
  window = new BrowserWindow()

  const { canceled, filePaths } = await dialog.showOpenDialog({
    title: 'Hello!',
    properties: ['openFile']
  })

  if (canceled) {
    console.log('Dialog was canceled')
    return
  }
  window.loadURL(`file://${filePaths[0]}`)
})
```

The dialog Module

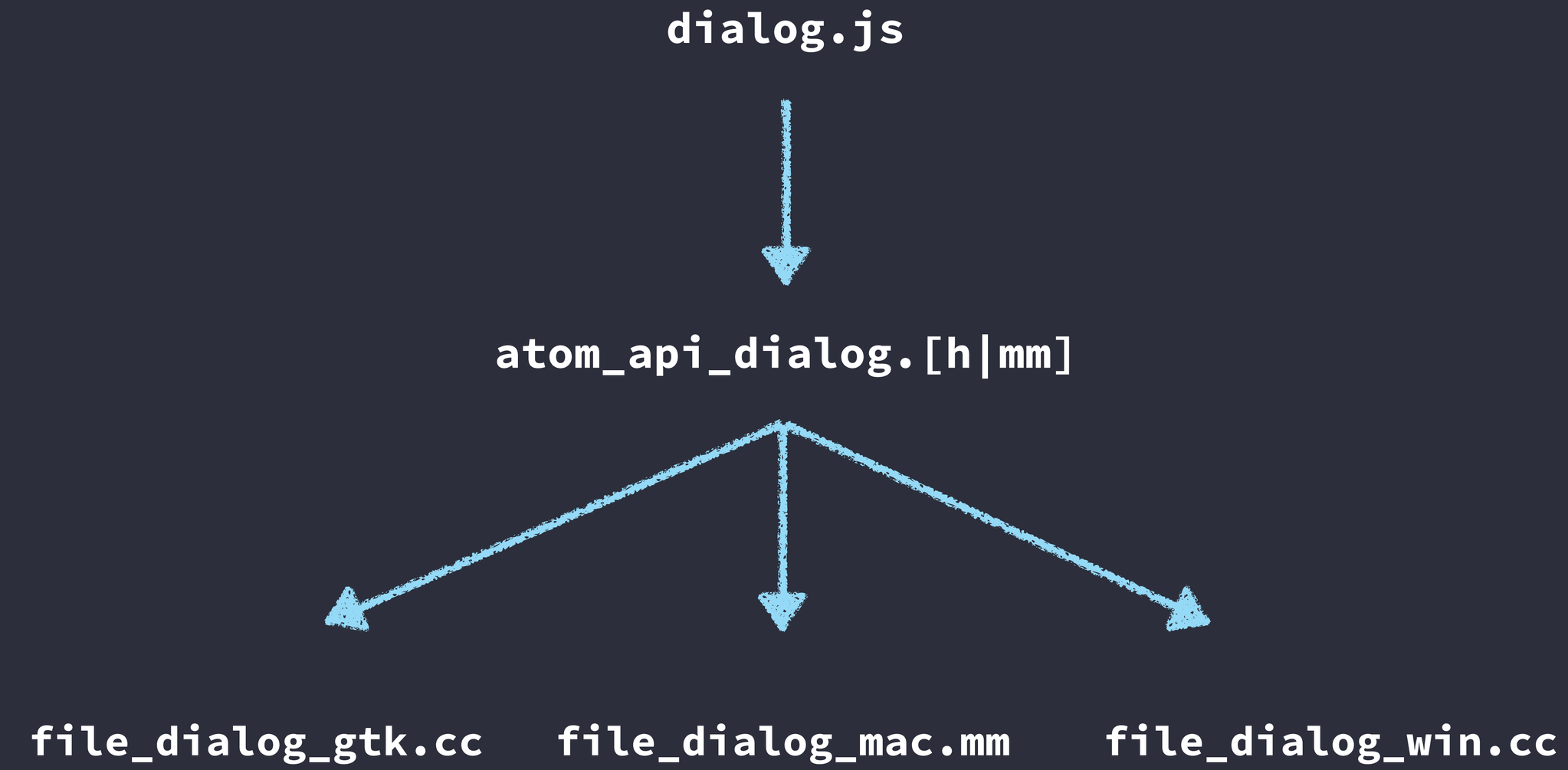


The dialog Module

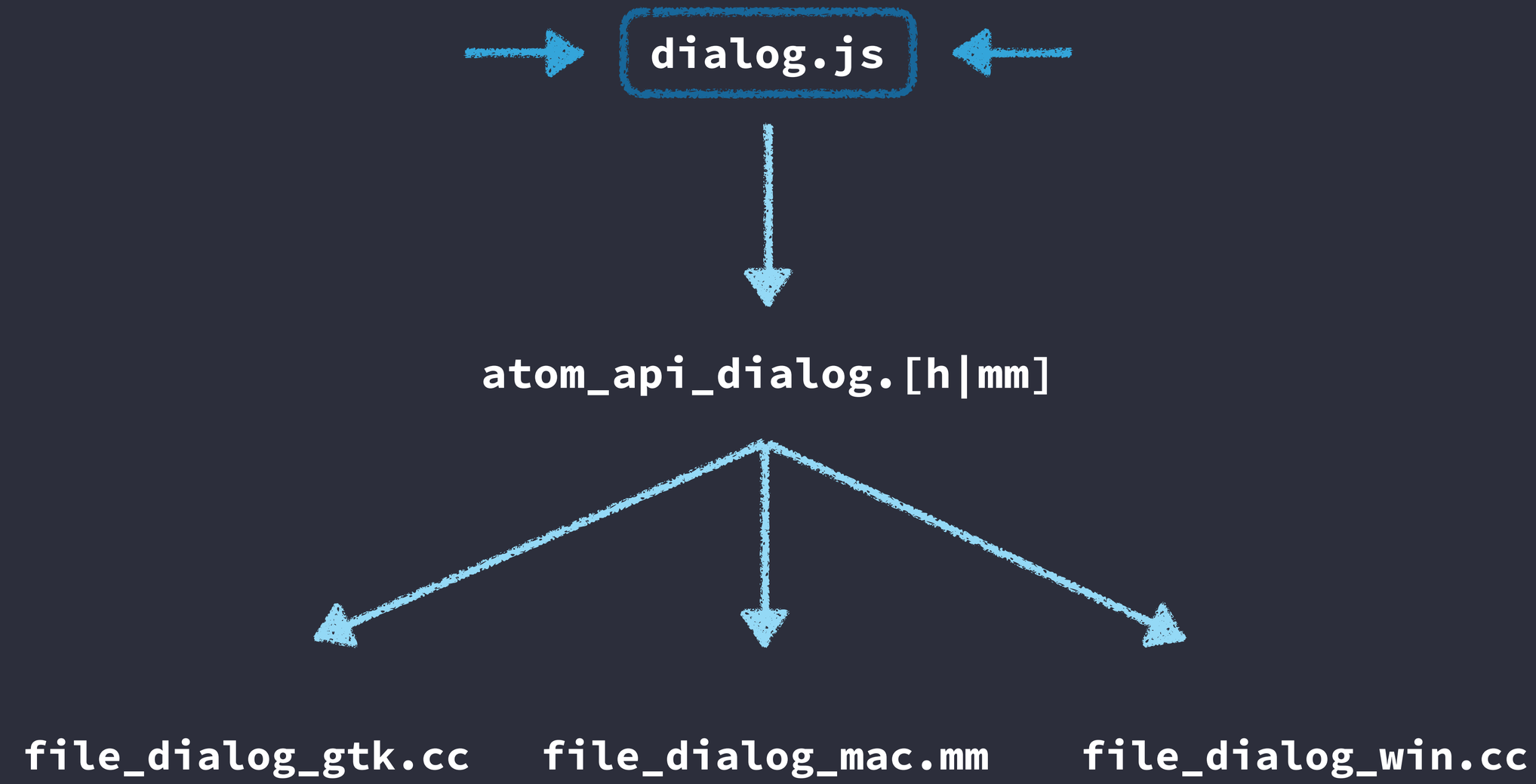
There's a **lot happening here**, so let's **dig in further** and discuss what's happening in **each different part** of this snippet

```
const { app, BrowserWindow, dialog } = require('electron')

let window = null

app.on('ready', async () => {
  window = new BrowserWindow()

  const { canceled, filePaths } = await dialog.showOpenDialog({
    title: 'Hello!',
    properties: ['openFile']
  })

  if (canceled) {
    console.log('Dialog was canceled')
    return
  }
  window.loadURL(`file://${filePaths[0]}`)
})
```

The dialog Module



The dialog Module



Accessing Electron

Require the
Electron module

```
const { app, BrowserWindow, dialog } = require('electron')

let window = null

app.on('ready', async () => {
  window = new BrowserWindow()

  const { canceled, filePaths } = await dialog.showOpenDialog({
    title: 'Hello!',
    properties: ['openFile']
  })

  if (canceled) {
    console.log('Dialog was canceled')
    return
  }
  window.loadURL(`file://${filePaths[0]}`)
})
```

Node.js

An open-source, cross-platform, **JavaScript runtime environment** that executes JavaScript code outside of a browser

Node.js

Helps Electron **create and export modules** to users

Allows Electron users to access **filesystem** and **networking** capabilities

require()

require – takes in an **id string** representing a **module name or path**, and returns the **exported module content**

Builtin Modules

```
const fs = require('fs')
```

Community Modules

```
const electron = require('electron')
```

Local Paths

```
const localMod = require('/path/to/local_mod')
```

The dialog Module

Import the exported **dialog** module from Electron

```
const { app, BrowserWindow, dialog } = require('electron')

let window = null

app.on('ready', async () => {
  window = new BrowserWindow()

  const { canceled, filePaths } = await dialog.showOpenDialog({
    title: 'Hello!',
    properties: ['openFile']
  })

  if (canceled) {
    console.log('Dialog was canceled')
    return
  }
  window.loadURL(`file://${filePaths[0]}`)
})
```

module.exports

```
module.exports = {
  showOpenDialog: function (window, options) {
    return openDialog(false, window, options)
  },
  /* more exports here */
}
```

Exported module
from dialog.js

The dialog Module

```
module.exports = {
  showOpenDialog: function (window, options) {
    return openDialog(false, window, options)
  },
  /* more exports here */
}
```

Exported function
in dialog.js

The dialog Module

```
module.exports = {
  showOpenDialog: function (window, options) {
    return openDialog(false, window, options)
  },
  /* more exports here */
}
```

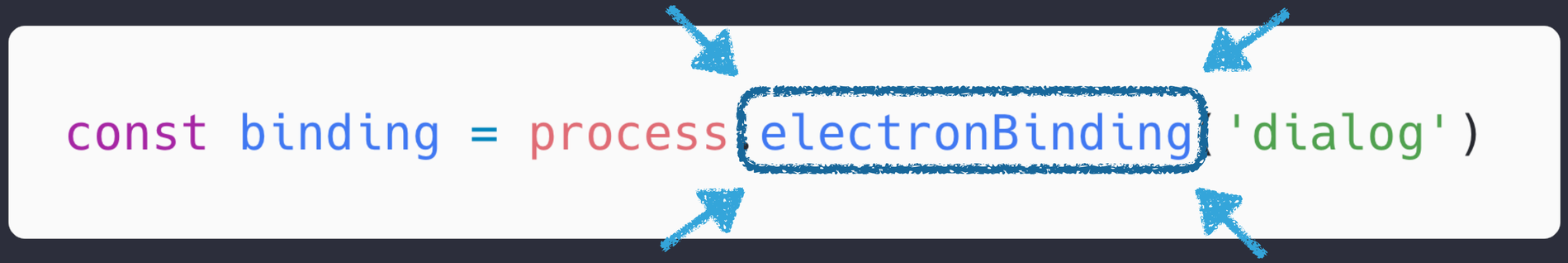
Exported function
calls out to another function

The dialog Module

```
const openDialog = (sync, window, options) => {
  checkAppInitialized()

  /* Some argument sanitization/setup here */

  if (sync) {
    return binding.showOpenDialogSync(settings)
  } else {
    binding.showOpenDialog(settings)
  }
}
```

Native bindings to
C++ code

The dialog Module



Native bindings to C++ code

How does this **custom electronBinding** work?

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.

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**!

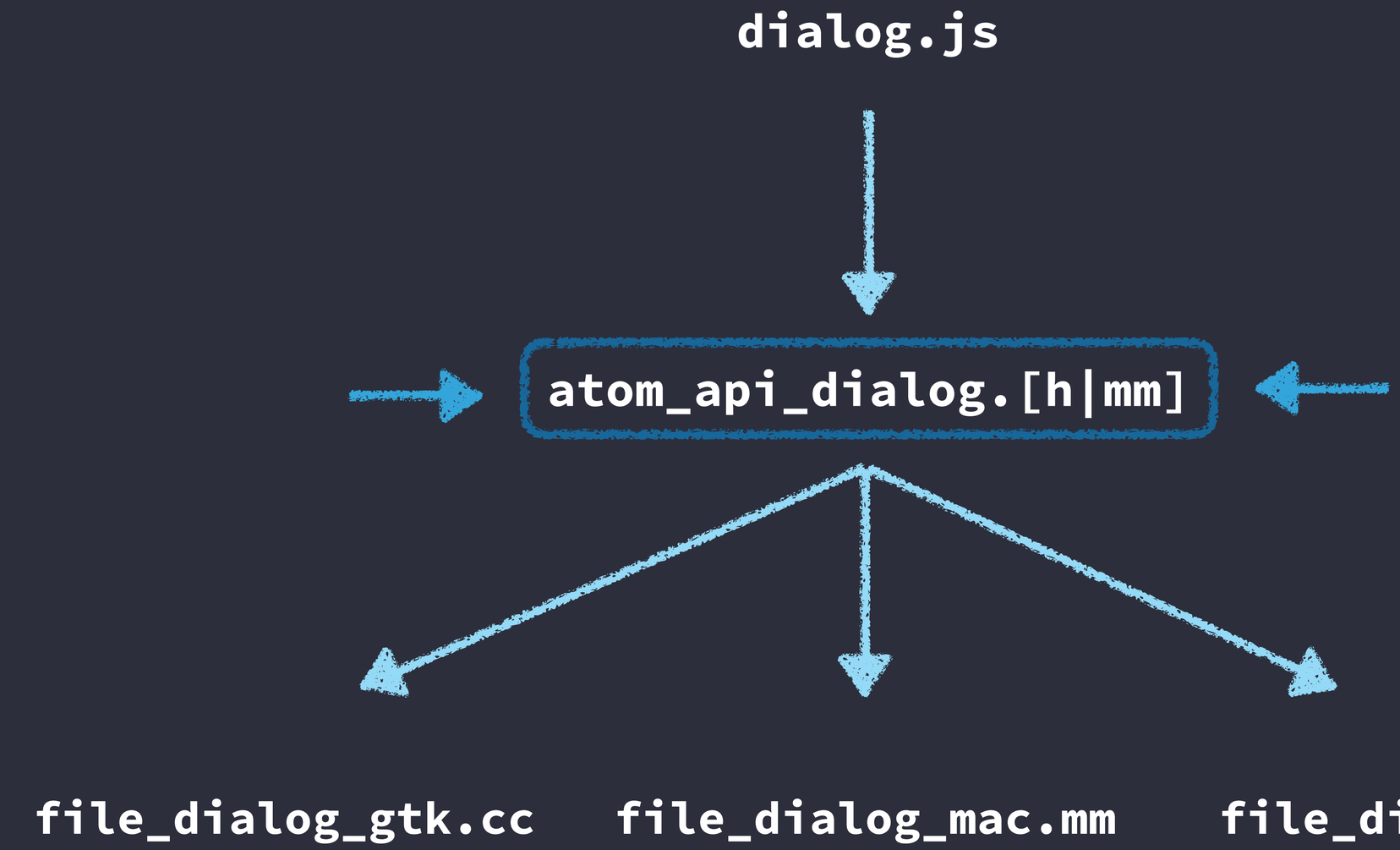
≥ Node.js **v8.8.0**

ElectronBindings

```
process.electronBinding('dialog')
```

Behaves like
process._linkedBinding
but loads native code from
Electron instead of from
Node.js

The dialog Module



V8

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

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

Allows Electron to **create Javascript objects** in C++!

```
void Initialize(v8::Local<v8::Object> exports,
                v8::Local<v8::Value> unused,
                v8::Local<v8::Context> context,
                void* priv) {
  v8::Isolate* isolate = context->GetIsolate();
  gin_helper::Dictionary dict(isolate, exports);
  dict.SetMethod("showOpenDialogSync", &ShowOpenDialogSync);
  dict.SetMethod("showOpenDialog", &ShowOpenDialog);
  /* other methods set here */
}

}  // namespace

NODE_LINKED_MODULE_CONTEXT_AWARE(atom_browser_dialog, Initialize)
```

Initialization of the
the **native binding**

```
void Initialize(v8::Local<v8::Object> exports,
                v8::Local<v8::Value> unused,
                v8::Local<v8::Context> context,
                void* priv) {
  v8::Isolate* isolate = context->GetIsolate();
  gin_helper::Dictionary dict(isolate, exports);
  dict.SetMethod("showOpenDialogSync", &ShowOpenDialogSync);
  dict.SetMethod("showOpenDialog", &ShowOpenDialog);
  /* other methods set here */
}

}  // namespace

NODE_LINKED_MODULE_CONTEXT_AWARE(atom_browser_dialog, Initialize)
```

Exports are held in a **V8 Object**

Node.js

```
void Initialize(v8::Local<v8::Object> exports,
                v8::Local<v8::Value> unused,
                v8::Local<v8::Context> context,
                void* priv) {
  v8::Isolate* isolate = context->GetIsolate();
  gin_helper::Dictionary dict(isolate, exports);
  dict.SetMethod("showOpenDialogSync", &ShowOpenDialogSync);
  dict.SetMethod("showOpenDialog", &ShowOpenDialog);
  /* other methods set here */
}

}  // namespace

NODE_LINKED_MODULE_CONTEXT_AWARE(atom_browser_dialog, Initialize)
```

Back to Node.js – passing this **initialization function** to **module registration**!

A top-level library in **Chromium.**

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

gin

```
void Initialize(v8::Local<v8::Object> exports,
                v8::Local<v8::Value> unused,
                v8::Local<v8::Context> context,
                void* priv) {
  v8::Isolate* isolate = context->GetIsolate();
  gin_helper::Dictionary dict(isolate, exports);
  dict.SetMethod("showOpenDialogSync", &ShowOpenDialogSync);
  dict.SetMethod("showOpenDialog", &ShowOpenDialog);
  /* other methods set here */
}

}  // namespace

NODE_LINKED_MODULE_CONTEXT_AWARE(atom_browser_dialog, Initialize)
```

receives an arbitrary JavaScript object as an argument

returns one as a result

We use a slightly modified **gin::Dictionary** when writing bindings for a function that either:

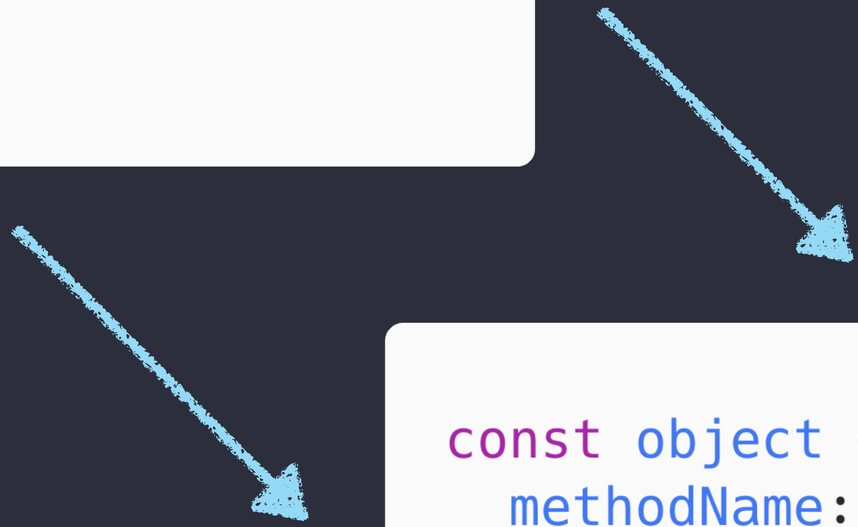
Setting Methods

```
dict.SetMethod("showOpenDialog", &ShowOpenDialog);
```

```
dict.SetMethod("methodName", &NativeImplementation);
```

Setting Methods

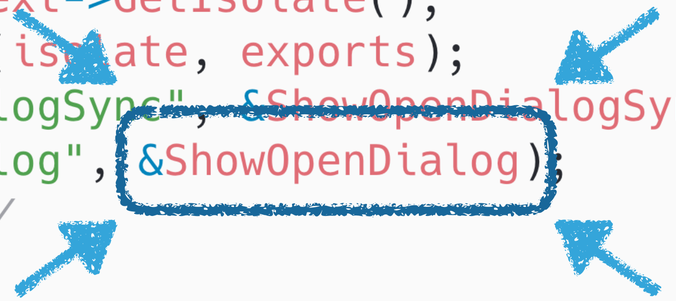
```
const dialog = {
  showOpenDialog: function () {
    // Implementation here
  }
}
```




```
const object = {
  methodName: function () {
    // Implementation here
  }
}
```

Native Implementations

```
void Initialize(v8::Local<v8::Object> exports,
                v8::Local<v8::Value> unused,
                v8::Local<v8::Context> context,
                void* priv) {
  v8::Isolate* isolate = context->GetIsolate();
  gin_helper::Dictionary dict(isolate, exports);
  dict.SetMethod("showOpenDialogSync", &ShowOpenDialogSync);
  dict.SetMethod("showOpenDialog", &ShowOpenDialog);
  /* other methods set here */
}

}  // namespace

NODE_LINKED_MODULE_CONTEXT_AWARE(atom_browser_dialog, Initialize)
```



Snippet

```
const { app, BrowserWindow, dialog } = require('electron')

let window = null

app.on('ready', async () => {
  window = new BrowserWindow()

  const { canceled, filePaths } = await dialog.showOpenDialog({
    title: 'Hello!',
    properties: ['openFile']
  })

  if (canceled) {
    console.log('Dialog was canceled')
    return
  }
  window.loadURL(`file://${filePaths[0]}`)
})
```

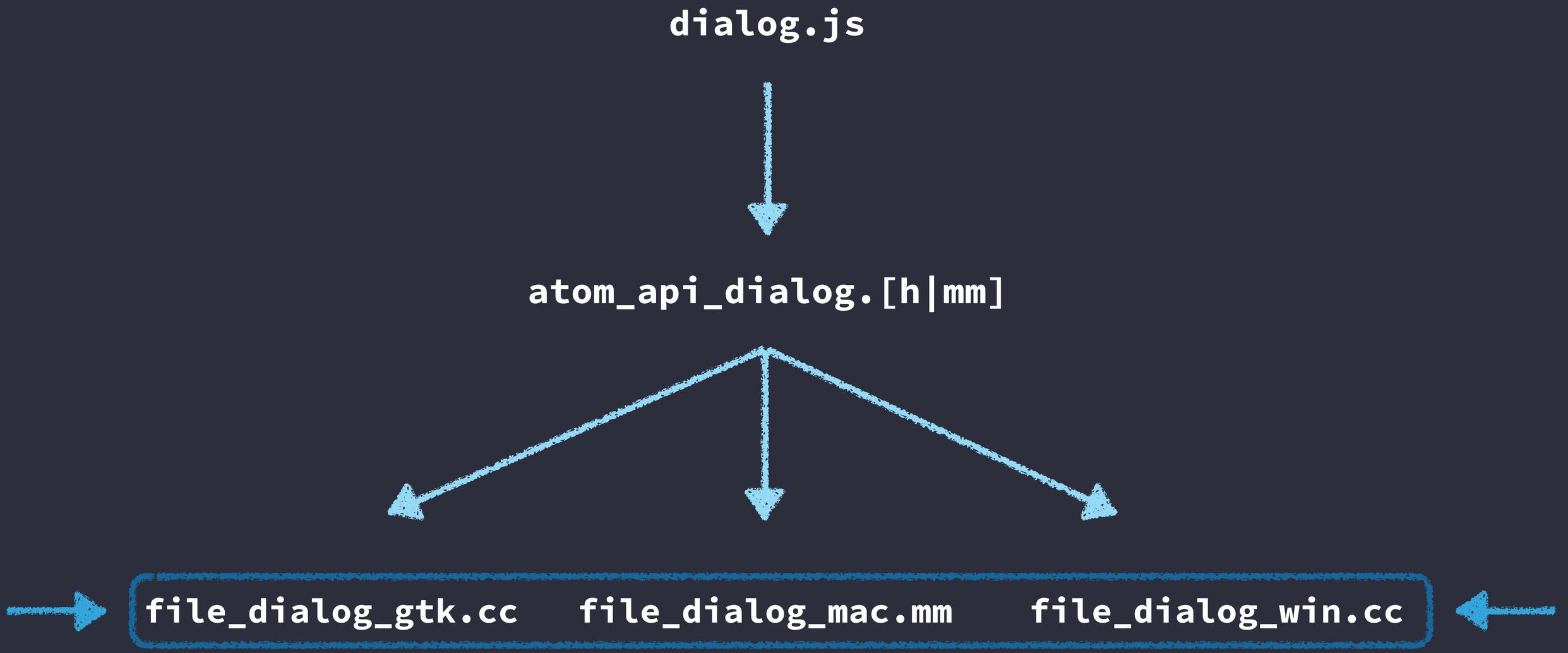
Native Implementations

```
v8::Local<v8::Promise> ShowOpenDialog(
    const file_dialog::DialogSettings& settings,
    gin::Arguments* args) {
  gin_helper::Promise<gin_helper::Dictionary> promise(args->isolate());
  v8::Local<v8::Promise> handle = promise.GetHandle();
  file_dialog::ShowOpenDialog(settings, std::move(promise));
  return handle;
}
```

showOpenDialog resolves an Object, so we're creating a V8::Promise resolving a **gin::Dictionary**

Electron's custom Promise wrapper class allows us to **resolve custom types**

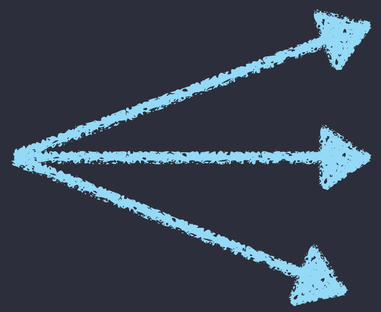
The dialog Module



Native Implementations

```
v8::Local<v8::Promise> ShowOpenDialog(
    const file_dialog::DialogSettings& settings,
    gin::Arguments* args) {
  gin_helper::Promise<gin_helper::Dictionary> promise(args->isolate());
  v8::Local<v8::Promise> handle = promise.GetHandle();
  file_dialog::ShowOpenDialog(settings, std::move(promise));
  return handle;
}
```

file_dialog namespace lets us invoke **platform-specific dialogs**



Windows

macOS

Linux

Native Implementations

```
void ShowOpenDialog(const DialogSettings& settings,
                    gin_helper::Promise<gin_helper::Dictionary> promise) {
  NSOpenPanel* dialog = [NSOpenPanel openPanel];

  SetupDialog(dialog, settings);
  SetupOpenDialogForProperties(dialog, settings.properties);

  // Other macOS-specific impl here
}
```

On macOS we use
NSOpenPanel to
create native dialogs

Native Implementations

```
void ShowOpenDialog(const DialogSettings& settings,
                    gin_helper::Promise<gin_helper::Dictionary> promise) {
  GtkFileChooserAction action = GTK_FILE_CHOOSER_ACTION_OPEN;
  if (settings.properties & OPEN_DIALOG_OPEN_DIRECTORY)
    action = GTK_FILE_CHOOSER_ACTION_SELECT_FOLDER;
  FileChooserDialog* open_dialog = new FileChooserDialog(action, settings);
  open_dialog->SetupOpenProperties(settings.properties);
  open_dialog->RunOpenAsynchronous(std::move(promise));
}
```

On Linux we use **GtkDialog** to create native dialogs

Native Implementations

```
void ShowOpenDialog(const DialogSettings& settings,
                    gin_helper::Promise<gin_helper::Dictionary> promise) {
  gin_helper::Dictionary dict = gin::Dictionary::CreateEmpty(promise.isolate());
  RunState run_state;

 // More Windows-specific implementation here
}
```

On Windows we use **IFileOpenDialog** to create native dialogs

Native Implementations

```
void OpenDialogCompletion(int chosen,
                          NSOpenPanel* dialog,
                          gin_helper::Promise<gin_helper::Dictionary> promise) {
  gin_helper::Dictionary dict = gin::Dictionary::CreateEmpty(promise.isolate());
  if (chosen == NSFileHandlingPanelCancelButton) {
    dict.Set("canceled", true);
    dict.Set("filePaths", std::vector<base::FilePath>());
    promise.Resolve(dict);
  } else {
    std::vector<base::FilePath> paths;
    dict.Set("canceled", false);
    ReadDialogPaths(dialog, &paths);
    dict.Set("filePaths", paths);

    promise.Resolve(dict);
  }
}
```

We **populate the object** with the fields we want to resolve

Chromium

An open source initiative developed by Google that powers the Google Chrome browser

Different parts of Electron hook into existing classes and functions in Chromium

Native Implementations

```
void OpenDialogCompletion(int chosen,
                          NSOpenPanel* dialog,
                          gin_helper::Promise<gin_helper::Dictionary> promise) {
  gin_helper::Dictionary dict = gin::Dictionary::CreateEmpty(promise.isolate());
  if (chosen == NSFileHandlingPanelCancelButton) {
    dict.Set("canceled", true);
    dict.Set("filePaths", std::vector<base::FilePath>());
    promise.Resolve(dict);
  } else {
    std::vector<base::FilePath> paths;
    dict.Set("canceled", false);
    ReadDialogPaths(dialog, &paths);
    dict.Set("filePaths", paths);

    promise.Resolve(dict);
  }
}
```

We leverage Chromium's **helpers** and **classes**

Chromium

```
#include "base/files/file_path.h"
#include "shell/common/gin_helper/dictionary.h"
#include "shell/common/gin_helper/promise.h"
```

base::FilePath – An abstraction to isolate users from the differences between native pathnames on different platforms

Resolving The Promise

```
void OpenDialogCompletion(int chosen,
                          NSOpenPanel* dialog,
                          gin_helper::Promise<gin_helper::Dictionary> promise) {
  gin_helper::Dictionary dict = gin::Dictionary::CreateEmpty(promise.isolate());
  if (chosen == NSFileHandlingPanelCancelButton) {
    dict.Set("canceled", true);
    dict.Set("filePaths", std::vector<base::FilePath>());
    promise.Resolve(dict);
  } else {
    std::vector<base::FilePath> paths;
    dict.Set("canceled", false);
    ReadDialogPaths(dialog, &paths);
    dict.Set("filePaths", paths);

    promise.Resolve(dict);
  }
}
```

Finally, we **resolve the Promise!**

Wrapping Up

This snippet **may be tiny**, but it **contains multitudes**

```
const { app, BrowserWindow, dialog } = require('electron')

let window = null

app.on('ready', async () => {
  window = new BrowserWindow()

  const { canceled, filePaths } = await dialog.showOpenDialog({
    title: 'Hello!',
    properties: ['openFile']
  })

  if (canceled) {
    console.log('Dialog was canceled')
    return
  }
  window.loadURL(`file://${filePaths[0]}`)
})
```

Wrapping Up

Without **these dependencies**, there is **no Electron**

THANK YOU!

@codebytere