Electron

A Frontend Masters workshop.

Steve Kinney

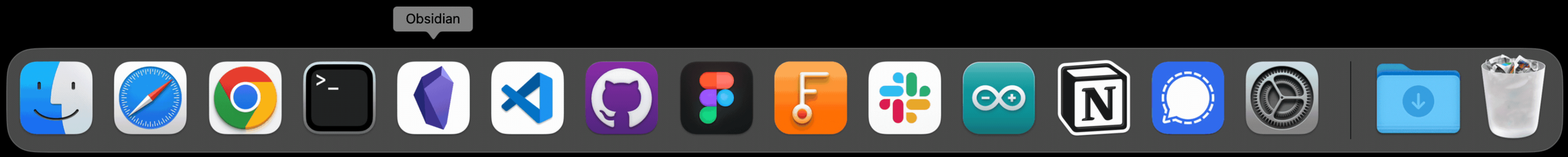
Electron

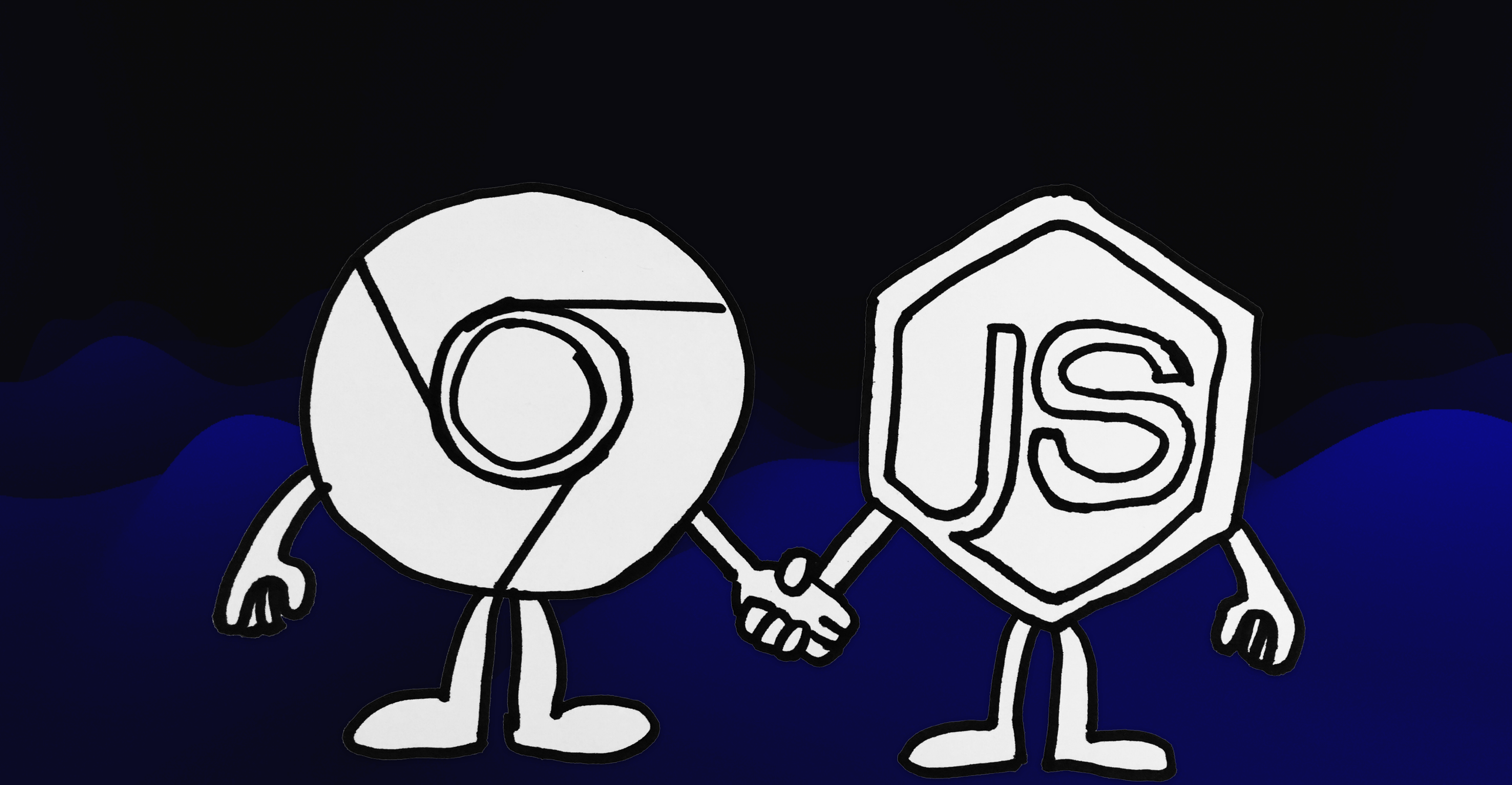
Get set up: bit.ly/electron-v3

Steve Kinney

Hey web developers, we're going to build some desktop applications.

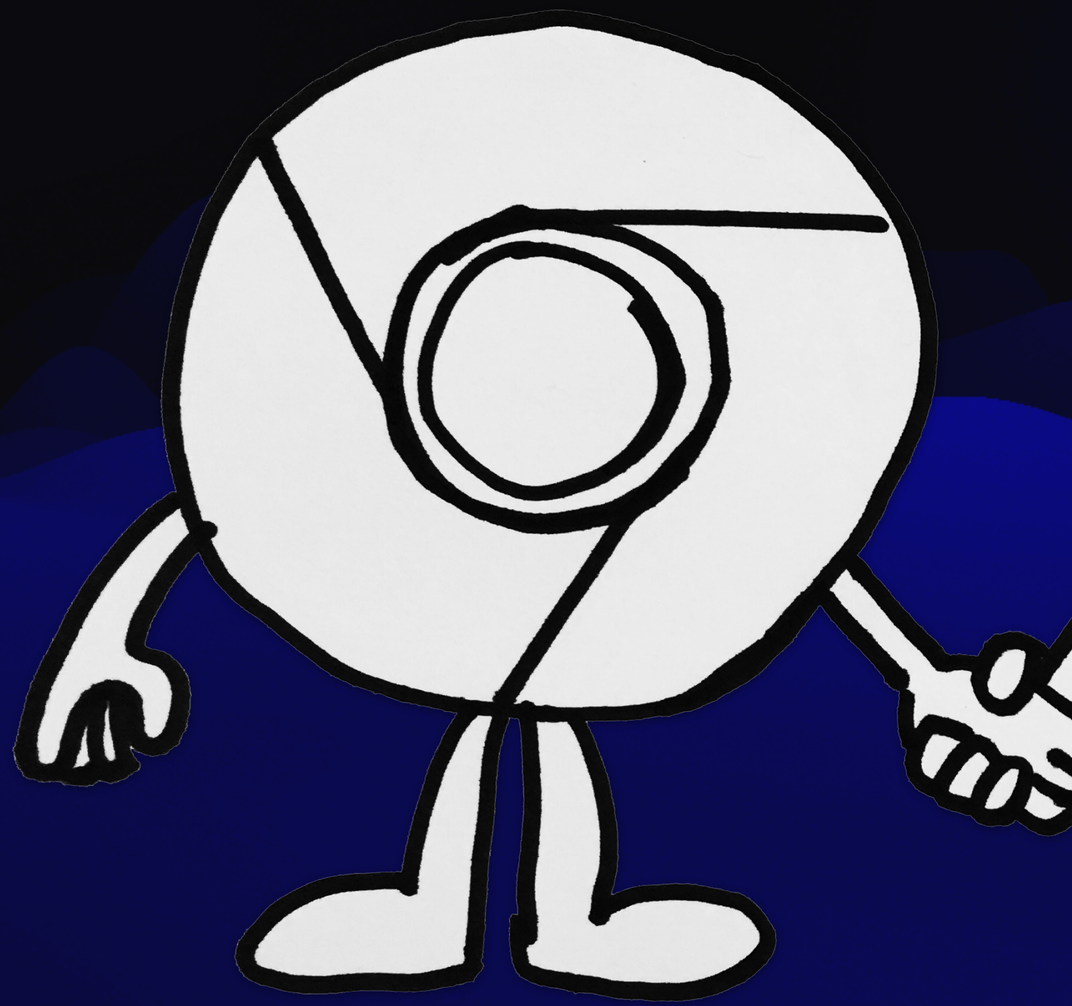
Electron is a framework for building desktop applications using web technologies.

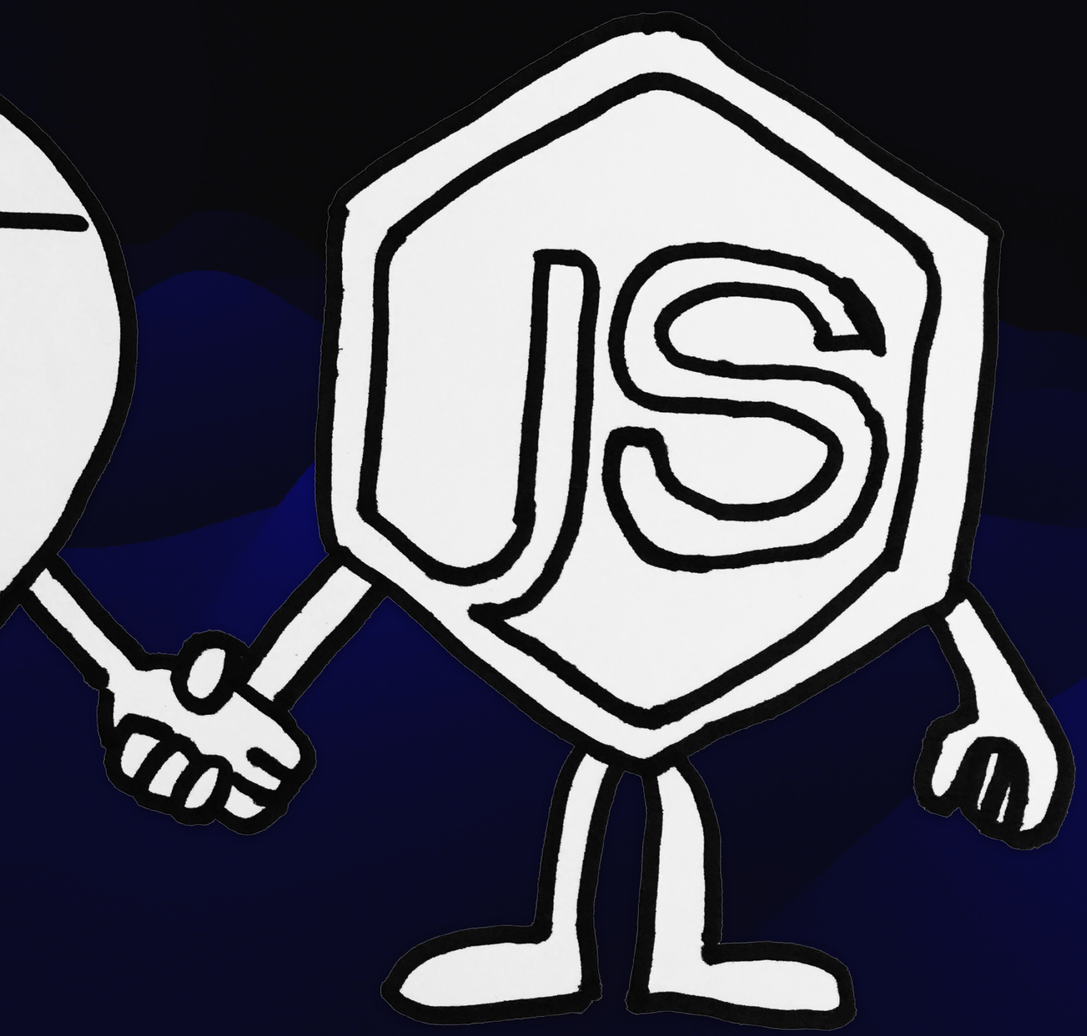



JS

Chromium
The Renderer Process

- HTML5 Support
- GPU Acceleration
- Blink
- V8
- Web APIs

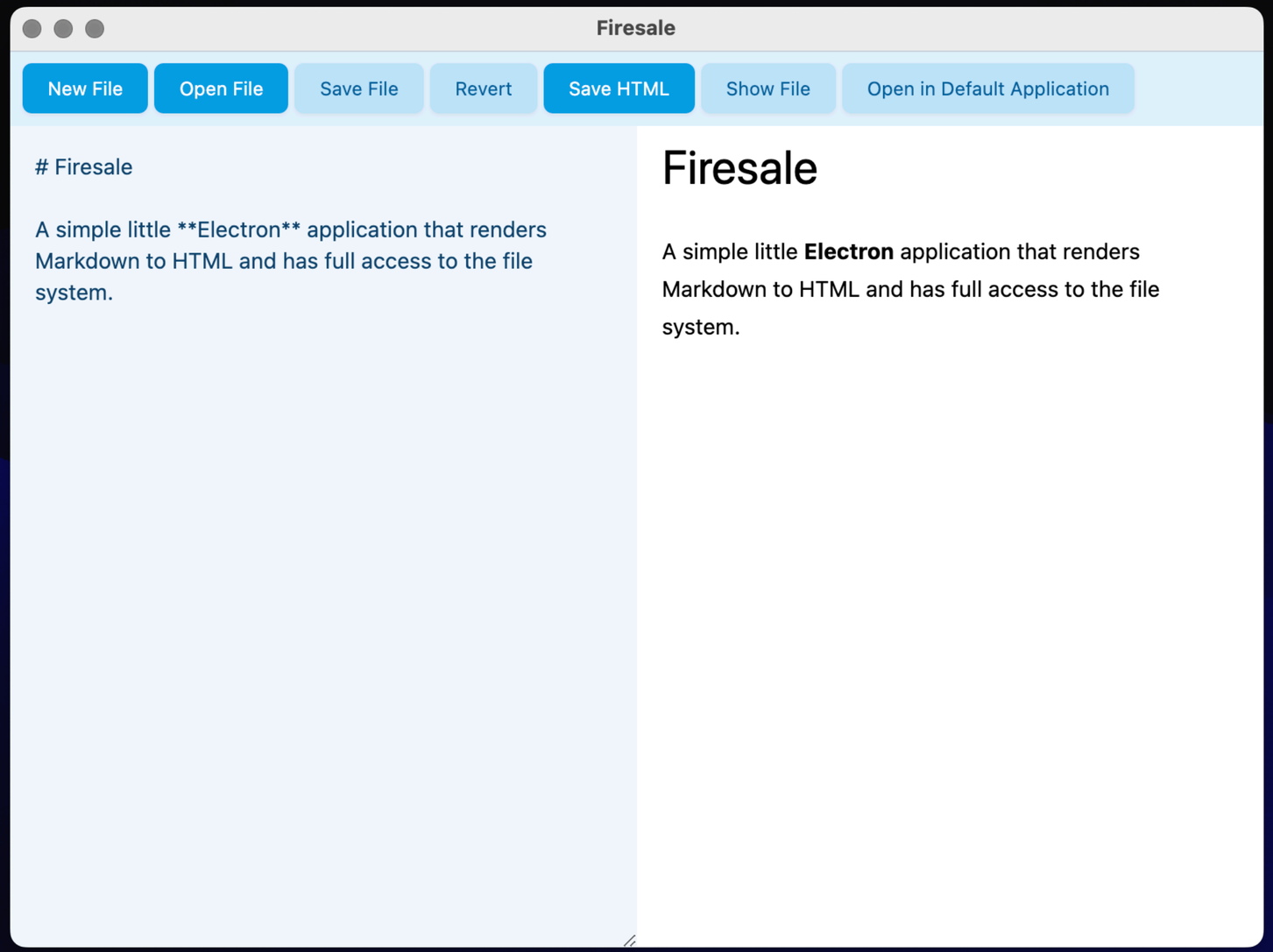




Node

The Main Process

- Native Modules
- Filesystem Access
- Full Server-side Capabilities


Firesale
New File
Open File
Save File
Revert
Save HTML
Show File
Open in Default Application
Firesale
A simple little **Electron** application that renders Markdown to HTML and has full access to the file system.
Firesale
A simple little Electron application that renders Markdown to HTML and has full access to the file system.

Electron File Edit View Window Help

7:25 56°

Not Done (Yet)

Add Task

New thing that I should totally do...

+ Add

Figure out what to do

Do the thing

Clipmaster

Type a new clipping and press "Enter"

Save to Clipboard

Something from my clipboard

Delete Copy

Copy from Clipboard

What are we going to cover?

- Native file system dialogs.
- Operating system-level integration.
- TypeScript support.
- Reading and writing from the filesystem.
- Using a frontend framework and build tools.
- Clipboard access.
- Notifications.
- Global shortcuts.
- Making system tray applications.
- Using native modules in Node.

Electron's Process Model

Dancing between processes is like 90% of the battle when building Electron applications.

MAIN PROCESS

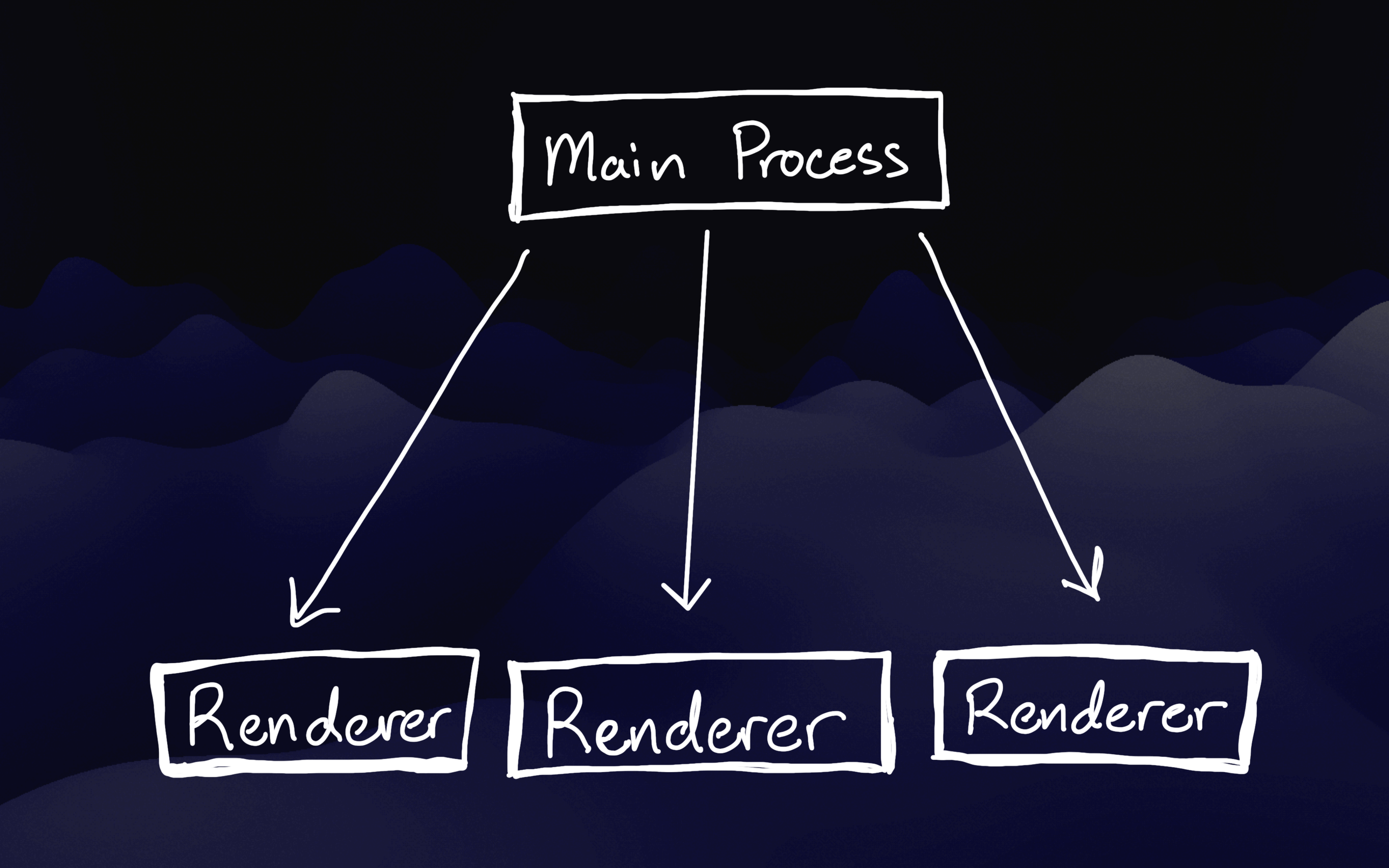
versus

RENDERER

HTML/JS/CSS

Main Process

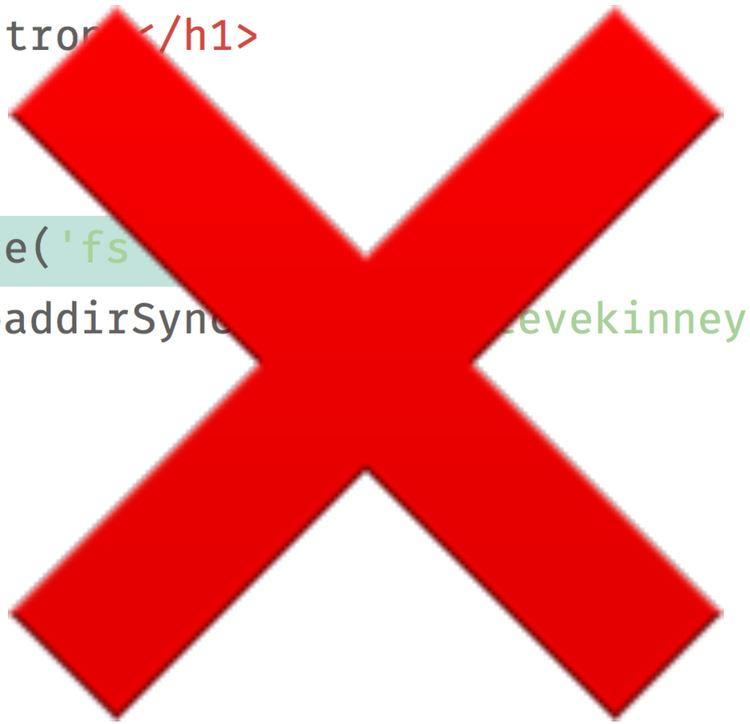
Renderer


Main Process
Renderer
Renderer
Renderer

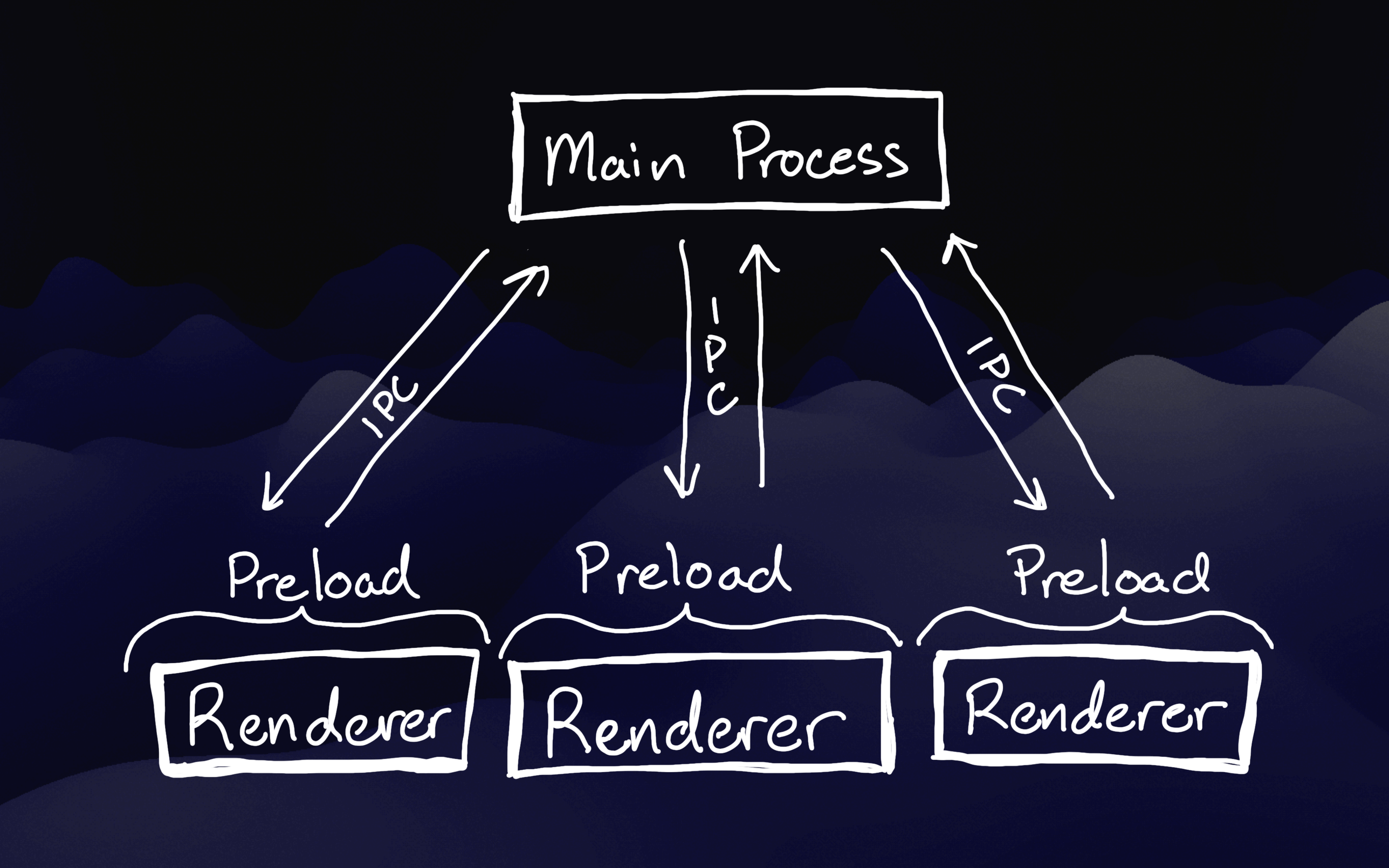
index.html — bartleby

main.js | index.html

```
1  <!DOCTYPE html>
2  <html>
3    <body>
4      <h1>Hello from Electron</h1>
5      <script>
6        'use strict';
7        const fs = require('fs
8        let files = fs.readdirSyn                    evekinney/Notes');
9      </script>
10   </body>
11 </html>
12
```



Line: 7:32-7:7 | HTML | Soft Tabs: 2


Main Process
IPC
IPC
IPC
Preload
Preload
Preload
Renderer
Renderer
Renderer

Getting Started

Quickly scaffold an Electron project with a full build pipeline

Overview

Electron Forge is an all-in-one tool for packaging and distributing Electron applications. It combines many single-purpose packages to create a full build pipeline that works out of the box, complete with code signing, installers, and artifact publishing. For advanced workflows, custom build logic can be added in the Forge lifecycle through its Plugin API. Custom build and storage targets can be handled by creating your own Makers and Publishers.

Creating a new app

To get started with Electron Forge, we first need to initialize a new project with `create-electron-app`. This script is a convenient wrapper around Forge's Init command.

Yarn | npm

```
npm init electron-app@latest my-app
```

If you used the `create-electron-app` script before during Forge `6.0.0-beta`, we recommend you uninstall the package globally before running the command again.

```
yarn global remove create-electron-app
```

IPC Methods
The Main Process

- `on`: Respond to a message we received from a renderer process.
- `handle`: Respond to a message and return a value—almost like it was a synchronous process.
- `webContents.send`: Send a message to a particular renderer.
- `webContents.postMessage`: Use the browser-based Message Channel API.

IPC Methods

Renderer Processes

- `on`: Respond to a message on a given channel.
- `send`: Send a message back to the main process.
- `invoke`: Call a function on the main process based on a message name and get the return value.

Open File Dialog Properties

showOpenDialog properties

- openFile
- openDirectory
- multiSelections
- showHiddenFiles
- createDirectory
- promptToCreate
- noResolveAliases
- treatPackageAsDirectory
- dontAddToRecent

Exporting the HTML

Exercise

Can you wire up the button that **exports the HTML**?

You can trigger a **save dialog** scoped to HTML files.

Reverting the Changes

Exercise

If we have unsaved changes, maybe our ideas weren't so good after all.

Can you communicate between the **main** and **renderer** processes to revert to the last saved copy of the file?

Our **documented edited** status should return to **false**.

Show File in File System

Exercise

Can you wire up the button that shows the **current file in the file system** (Finder, Windows Explorer, etc.)?

This command is your friend: `shell.showItemInFolder()`.

Open in Default Application

Exercise

Can you wire up the button that **opens the file in the default application**?

This command is your friend: `shell.openPath()`.

Add an Edit Menu

Exercise

We’ve lost the ability to copy and paste.

Can you bring back the edit menu?

Showing the Window with a Shortcut

Exercise

Can you show and focus the Clipmaster window on a global shortcut?

Add a Notification to the Global Shortcut

Exercise

Can you add a new notification when a new clipping is saved using a global shortcut?