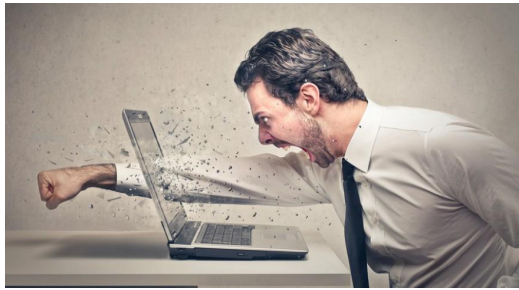
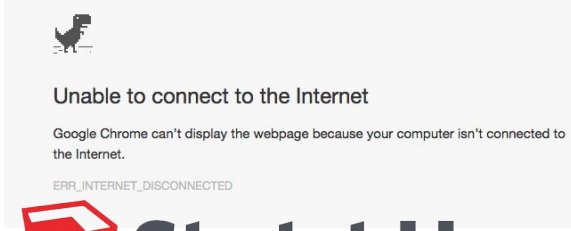

ELECTRON

Build cross platform desktop apps with JavaScript, HTML, and CSS

Why we need ‘Desktop Applications’? [1/2]

- Secure
- To work ‘Offline’
  - On premises
  - We are ‘NOT CONSTANTLY CONNECTED’
- For professional level applications
  - Photoshop, AutoCAD etc.
- Hosting?





SketchUp

AutoCAD

Ps

v-ray

MAX

Why we need ‘Desktop Applications’? [2/2]

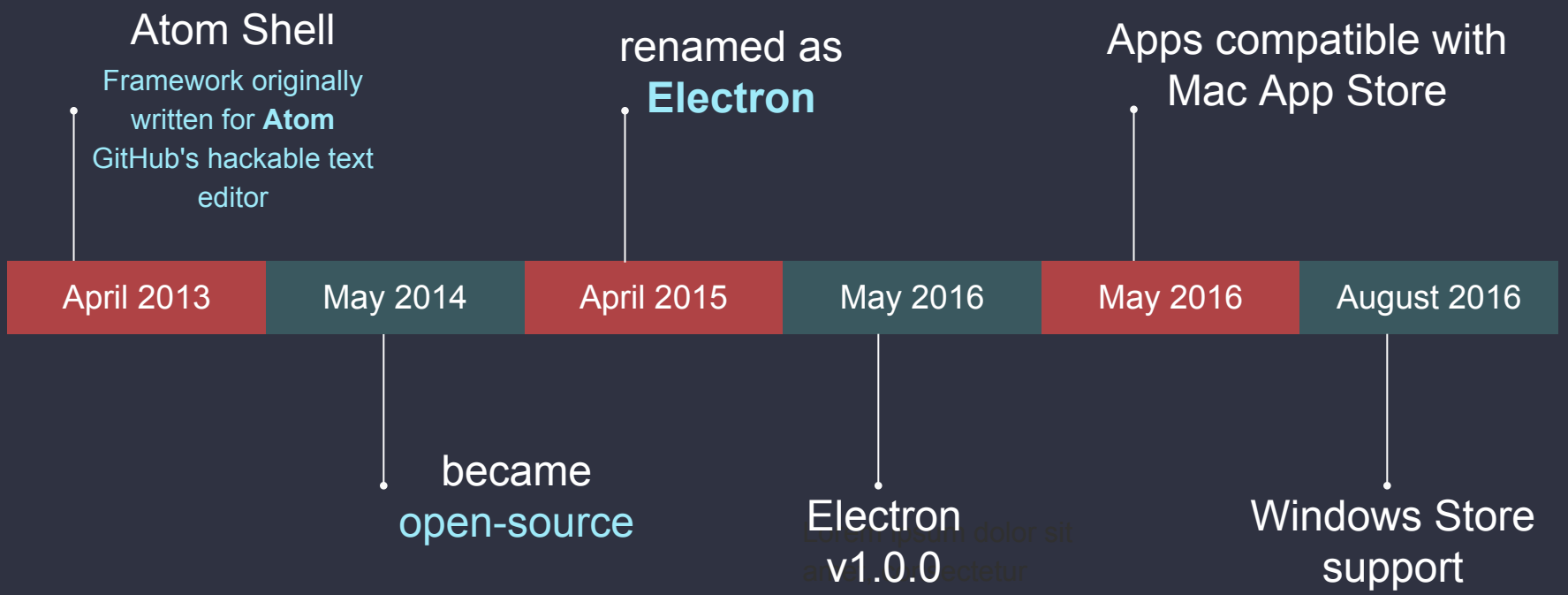
- Connect with peripheral devices & other local hardware
- Edit any local files
- Novice users - non tech savvy


ELECTRON

If you can build a website,
you can build a desktop app.

Electron

- Open source library
- Developed by GitHub - Led by Cheng Zhao
- Cross-platform desktop applications
- Use web technologies - HTML, CSS & JavaScript.
- Combines Chromium and Node.js into a single runtime
- Mac, Windows, and Linux


Atom Shell
Framework originally written for **Atom** GitHub's hackable text editor
April 2013
May 2014
became open-source
renamed as **Electron**
April 2015
Electron v1.0.0
May 2016
Apps compatible with Mac App Store
May 2016
Windows Store support
August 2016

Why Electron?

Web Technologies

Electron uses Chromium and Node.js so you can build your app with HTML, CSS, and JavaScript.

Open Source

Electron is an open source project maintained by GitHub and an active community of contributors.

Cross Platform

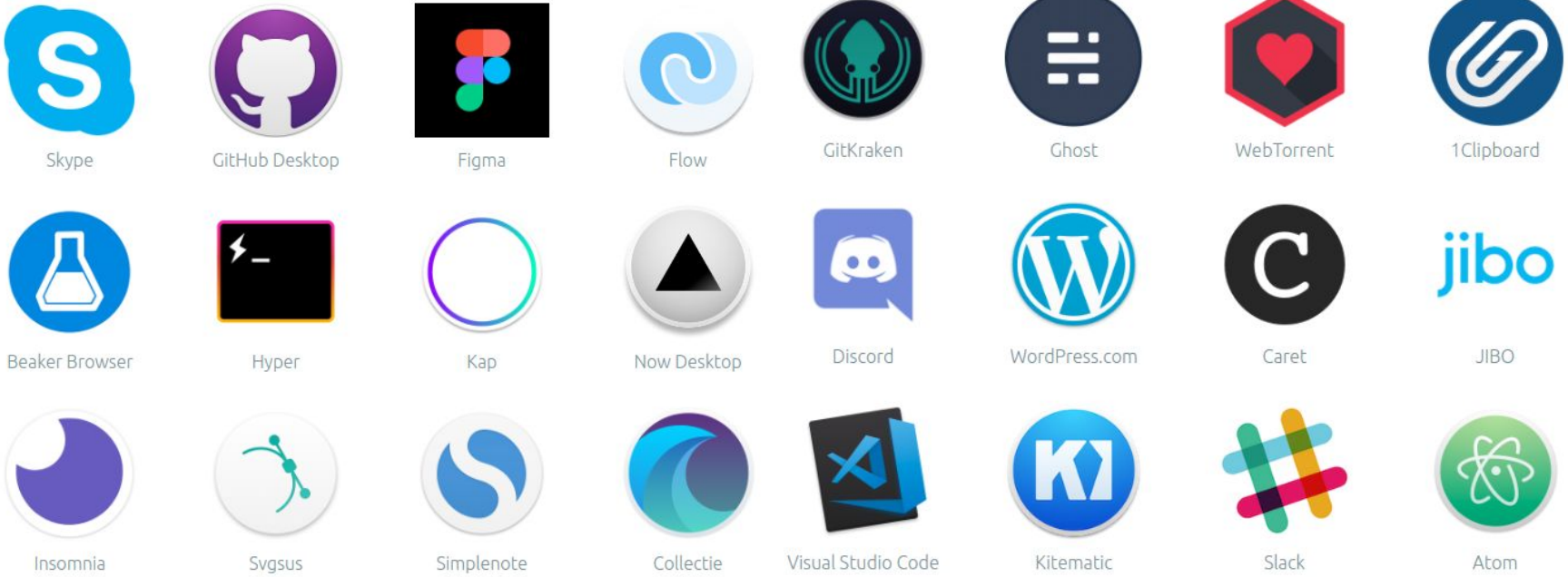
Compatible with Mac, Windows, and Linux, Electron apps build and run on three platforms.

- Electron enables you to create desktop applications with pure JavaScript by providing a runtime with rich native (operating system) APIs.
- Electron is NOT a JavaScript binding to GUI libraries.
- Electron uses web pages as its GUI
  - “Chromium browser, controlled by JavaScript”
- Electron App: Web pages running in a browser that can leverage your OS APIs.

It takes care of the hard parts.
So you can focus on the core of your application

- Enable auto update
  - Squirrel/Squirrel.Windows - installation/update framework
- Native Components
  - Create native application menus and context menus, dialogs, system tray & notifications etc.
- Crash reporting
  - Submit crash reports to a remote server.
- Debugging & Profiling
  - Collect tracing data from Chromium's content module for finding performance bottlenecks and slow operations.

Apps built on Electron



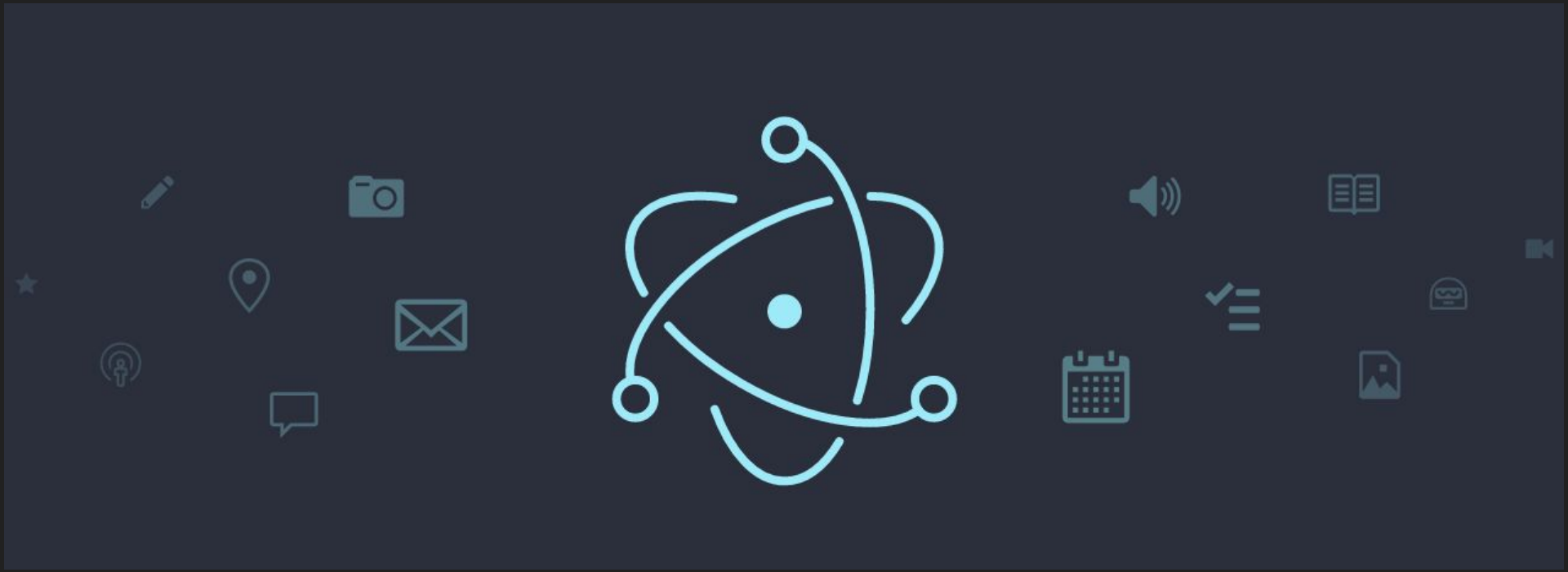
Install Electron?

- Recommended: Install as a development dependency
  - Allows you to work on multiple apps with different Electron versions
- Global Installation
- Proxies
- Custom Mirrors & Caches
- Customized

Install Electron as a development dependency

- npm
  - Package manager/ software registry
  - Open-source building blocks of code (third party packages)
  - Install, update, share & distribute

- npm install --save-dev electron

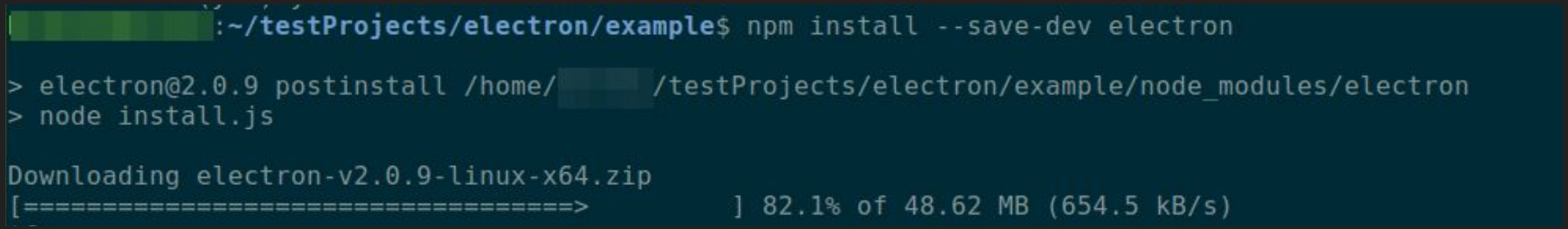


Quick Start

- Create a new empty folder for your new Electron application.
- Run npm init
- npm will guide you to create a basic package.json file

```
[redacted]:~/testProjects/electron/example$ npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help json` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg> --save` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
name: (example) electron-example
version: (1.0.0)
description:
entry point: (index.js) main.js
test command:
git repository:
keywords:
author:
license: (ISC)
About to write to /home/[redacted]/testProjects/electron/example/package.json:

{
  "name": "electron-example",
  "version": "1.0.0",
  "description": "",
  "main": "main.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}
```

- Install electron:
  - npm install --save-dev electron


:~/testProjects/electron/example$ npm install --save-dev electron
> electron@2.0.9 postinstall /home/ /testProjects/electron/example/node_modules/electron
> node install.js
Downloading electron-v2.0.9-linux-x64.zip
[=================================>   ] 82.1% of 48.62 MB (654.5 kB/s)

- Add a start script that instructs node to execute the current package with electron runtime.
  - "start": "electron ."

Folder Structure

```
your-app/
    |__packagage.json
    |__main.js
    |__index.html
```

1. package.json

```
{
    "name": "your-app",
    "version": "0.1.0",
    "main": "main.js",
    "scripts": {
      "start": "electron ."
    }
  }
```

2. main.js

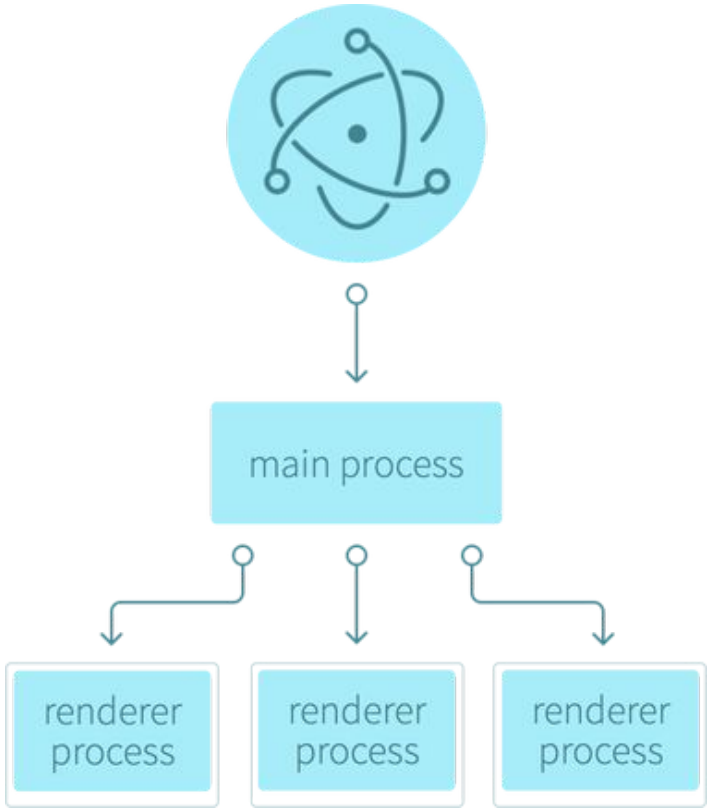
```
const {app, BrowserWindow} = require('electron')

function createWindow () {
  // Create the browser window.
  win = new BrowserWindow({width: 800, height: 600})

  // and load the index.html of the app.
  win.loadFile('index.html')
}

app.on('ready', createWindow)
```

3. index.html

```
<!DOCTYPE html>
  <html>
    <head>
      <meta charset="UTF-8">
      <title>Hello World!</title>
    </head>
    <body>
      <h1>Hello World!</h1>
    </body>
  </html>
```

npm start

Architecture



Main Process

- Runs package.json's main script.
- The script that runs in the main process can display a GUI by creating web pages.
- An Electron app always has one main process, but never more.

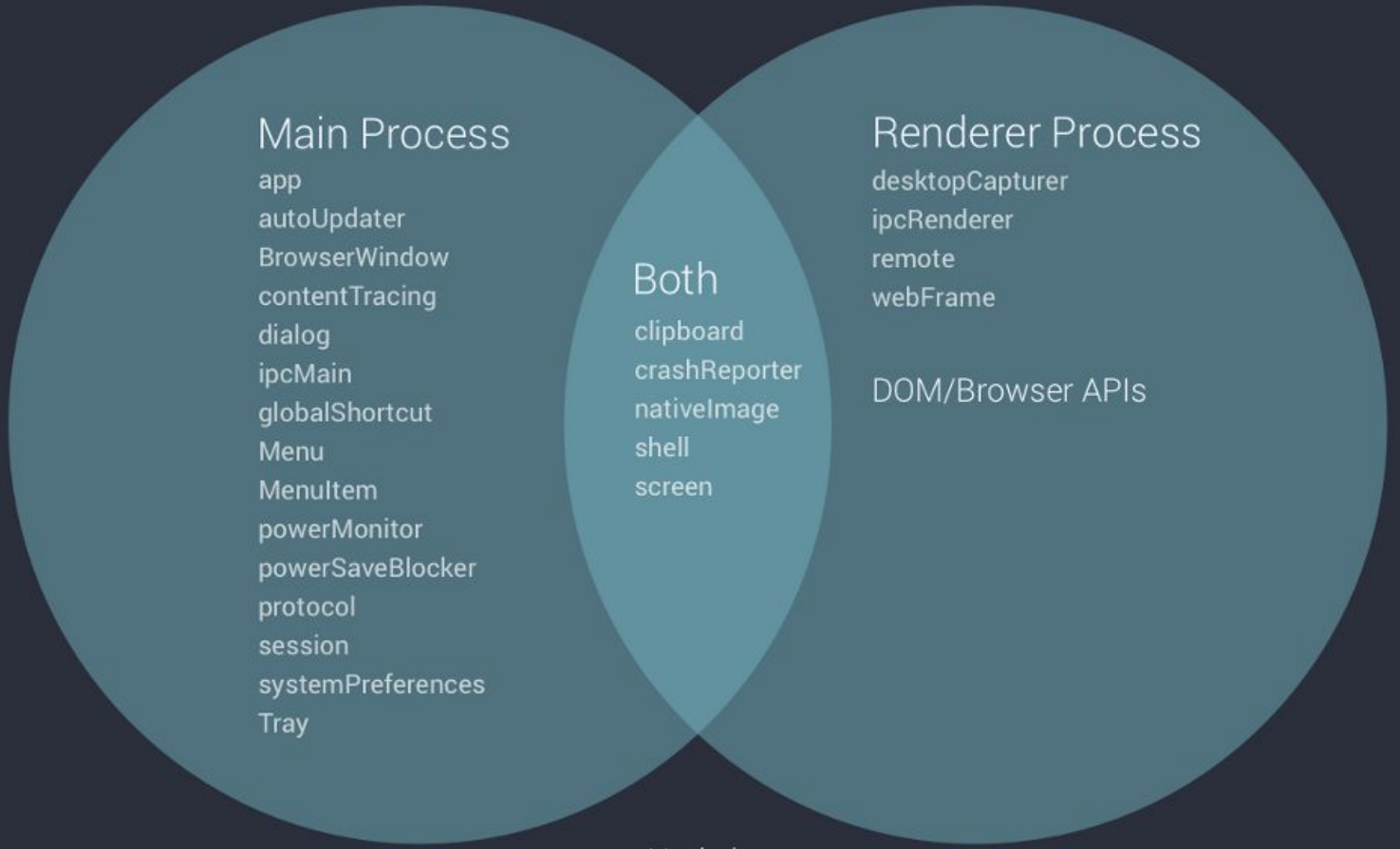
Renderer Process

- Based on Chromium's multi-process architecture
- Electron uses Chromium for displaying web pages
- Each web page in Electron runs in its own process

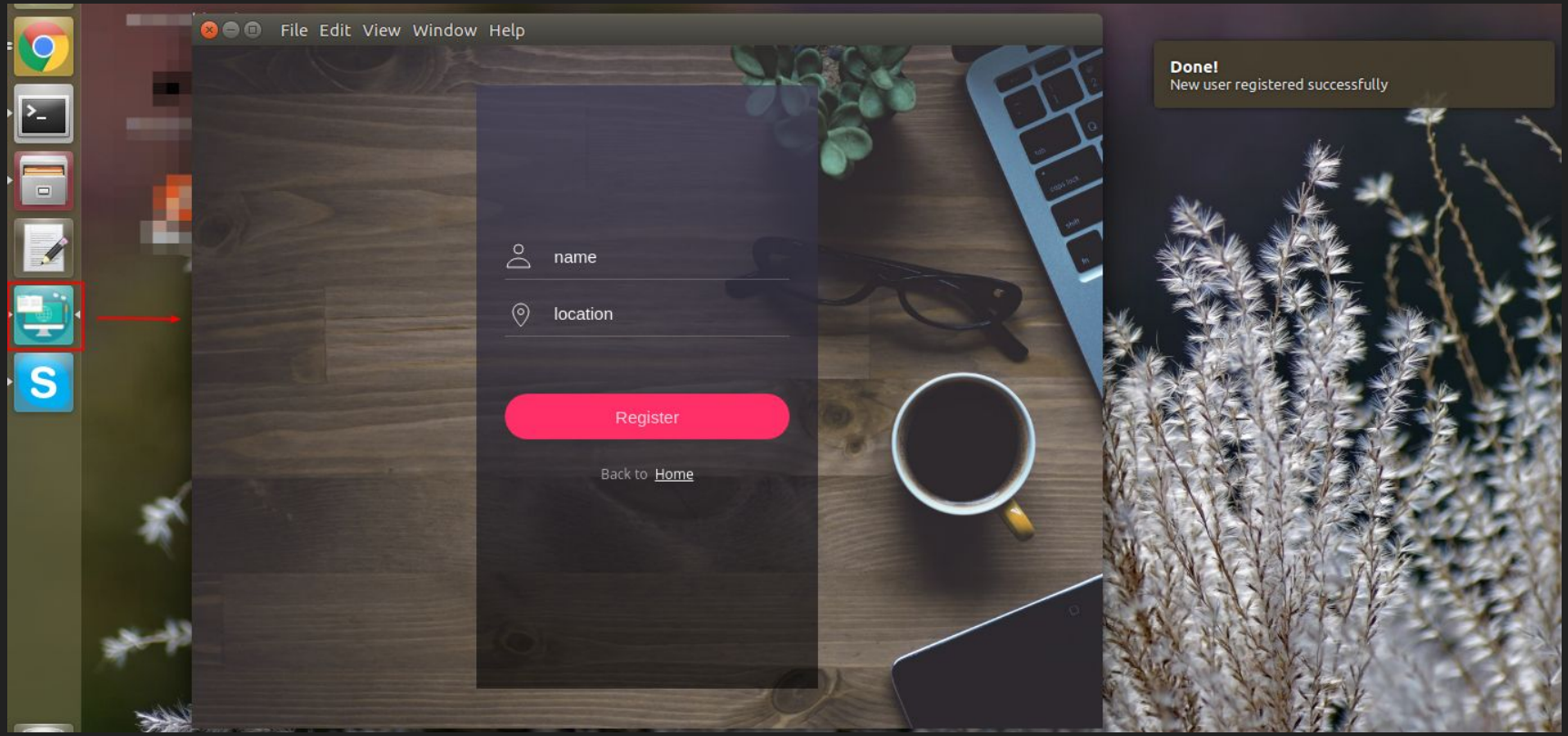
- The main process creates web pages by creating BrowserWindow instances.
- Each BrowserWindow instance runs the web page in its own renderer process.
- When a BrowserWindow instance is destroyed, the corresponding renderer process is also terminated.

- The main process manages all web pages and their corresponding renderer processes.
- Each renderer process is isolated and only cares about the web page running in it.

- In web pages, calling native GUI related APIs is not allowed.
  - Managing native GUI resources in web pages is very dangerous
  - It is easy to leak resources
- If you want to perform GUI operations in a web page, the renderer process of the web page must communicate with the main process to request that the main process perform those operations.


Main Process
app
autoUpdater
BrowserWindow
contentTracing
dialog
ipcMain
globalShortcut
Menu
MenuItem
powerMonitor
powerSaveBlocker
protocol
session
systemPreferences
Tray
Both
clipboard
crashReporter
nativeImage
shell
screen
Renderer Process
desktopCapturer
ipcRenderer
remote
webFrame
DOM/Browser APIs
Node.js

Demo


File Edit View Window Help
name
location
Register
Back to Home
Done!
New user registered successfully

Application Distribution

1. Download Electron's prebuilt binaries
2. Packaging
    - can be done manually
    - Third party packaging tool
        - electron-forge
        - electron-builder
        - Electron-packager

Pros [1/5]

- Electron Apps Are Similar to Web Apps
  - Relies entirely on web standards
    - HTML, CSS, JS
- Can focus on the core functionality
  - Electron already handles the hard part in software development

Pros [2/5]

- Reuse of resources
  - Same team working on a web-application can easily implement a desktop application.
  - Several things from the web-application can be reused in the desktop app:
    - Business logic (services, helpers)
    - Design
    - General application structure
  - Saves time and money.

Pros [3/5]

- Chromium engine for rendering UI
  - Developer tools
    - Don’t have to think of an external debugger
    - easy to measure performance, conduct profiling or detect memory leaks
  - Hot / Live reload
    - See changes instantly
  - Storage access
    - LocalStorage, SessionStorage, IndexedDB to persist application state.

Pros [4/5]

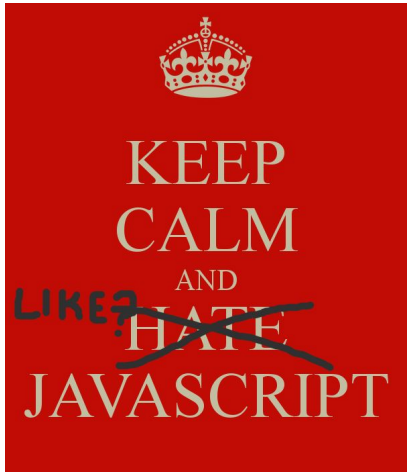
- Provides various core functionalities:
  - Auto-update
  - crash reporter
  - installer creator
  - system-specific/native-ish features
    - Menu, system dialogs, file selection dialogs, notifications, printer, etc
    - Shortcuts bindings
  - Open file manager & external links
  - Screenshots & PDF

Pros [5/5]

- Reuse of npm modules, out of the box
- No cross-browser compatibility issues
- No loading of remote assets (then no latency)
- Hardware Access
  - Keyboard shortcuts

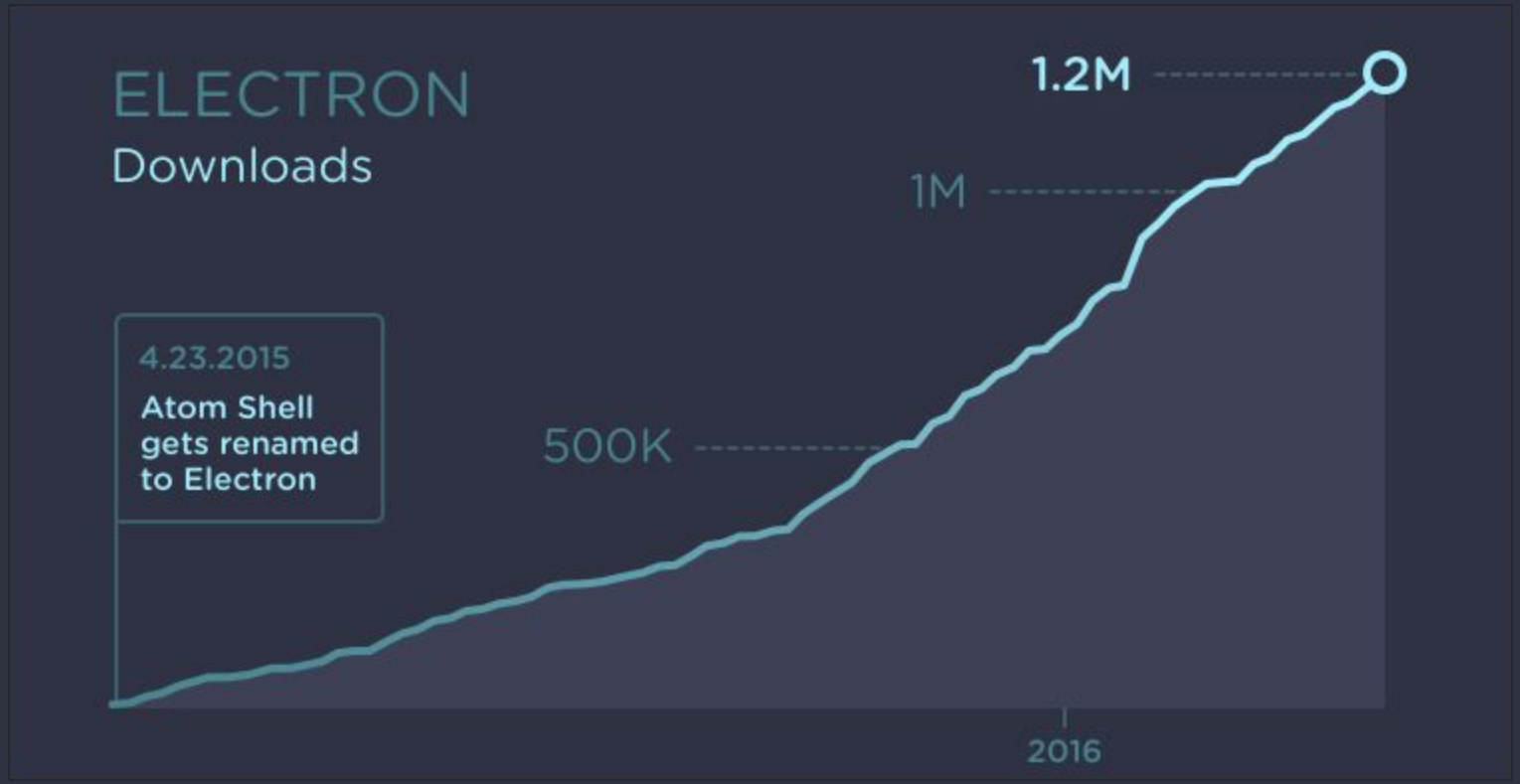
Cons [1/2]

- JavaScript
- No built-in MVC
- Not as feature-rich or mature as NW.js

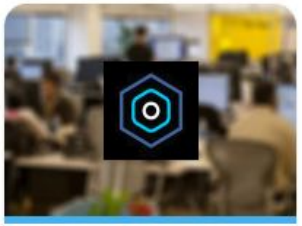


Cons [2/2]

- Size & Memory Usage!
  - Node JS + Chromium
  - Electrino - experimental featherweight alternative
- Cross Platform Builds?
  - If your app has native dependencies, it can be compiled only on the target platform.
    - macOS code Signing works only on MacOS.


ELECTRON
Downloads
1.2M
1M
500K
4.23.2015
Atom Shell gets renamed to Electron
2016

Sponsors



Facebook Open Source

Facebook Open Source Team

TeamSQL

TeamSQL

MyCard

Kid Talk

jamovi

open source, real-time statistical spreadsheet built on electron

Coherent Software

Lost Astronaut Studios

Pittsburgh, PA based Lost Astronaut Game Studios: Check us out ...

Resources for learning Electron

- electronjs.org/docs - all of Electron's documentation
- electron/electron-quick-start - a very basic starter Electron app
- electronjs.org/community#boilerplates - sample starter apps created by the community
- electron/simple-samples - small applications with ideas for taking them further
- electron/electron-api-demos - an Electron app that teaches you how to use Electron
- hokein/electron-sample-apps - small demo apps for the various Electron APIs

References

- https://electronjs.org/
- https://electronjs.org/docs/faq
- https://www.npmjs.com/package/electron
- https://slides.com/juliamaksimchik/electron-awesome
- https://www.tutorialspoint.com/electron/
- https://steemit.com/technology/@ryanbaer/getting-started-with-electron-pt-3-how-the-hell-does-this-thing-work
- https://www.arroyolabs.com/2016/08/electron-and-cross-platform-applications/
- https://ourcodeworld.com/articles/read/259/how-to-connect-to-a-mysql-database-in-electron-framework

Thank you!