The Conceptual Architecture of Chrome

Thick Glitches

Tyler Mainguy, Liam Walsh, Andrea Perera-Ortega, Jessica Dassanayake, Alistair Lewis, Brendan Kolisnik

Introduction

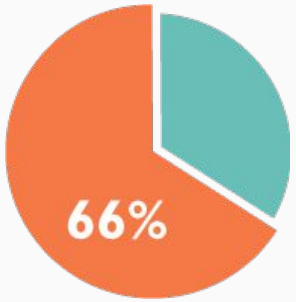
Web Browser
developed by

Google

for Windows, Linux, macOS, iOS and Android
Released in 2008

World's Most Popular Browser

Accounts for approximately



of usage share

One of the
most stable
and
high performance
browsers due to its
multi-processing implementation

Derivation Process

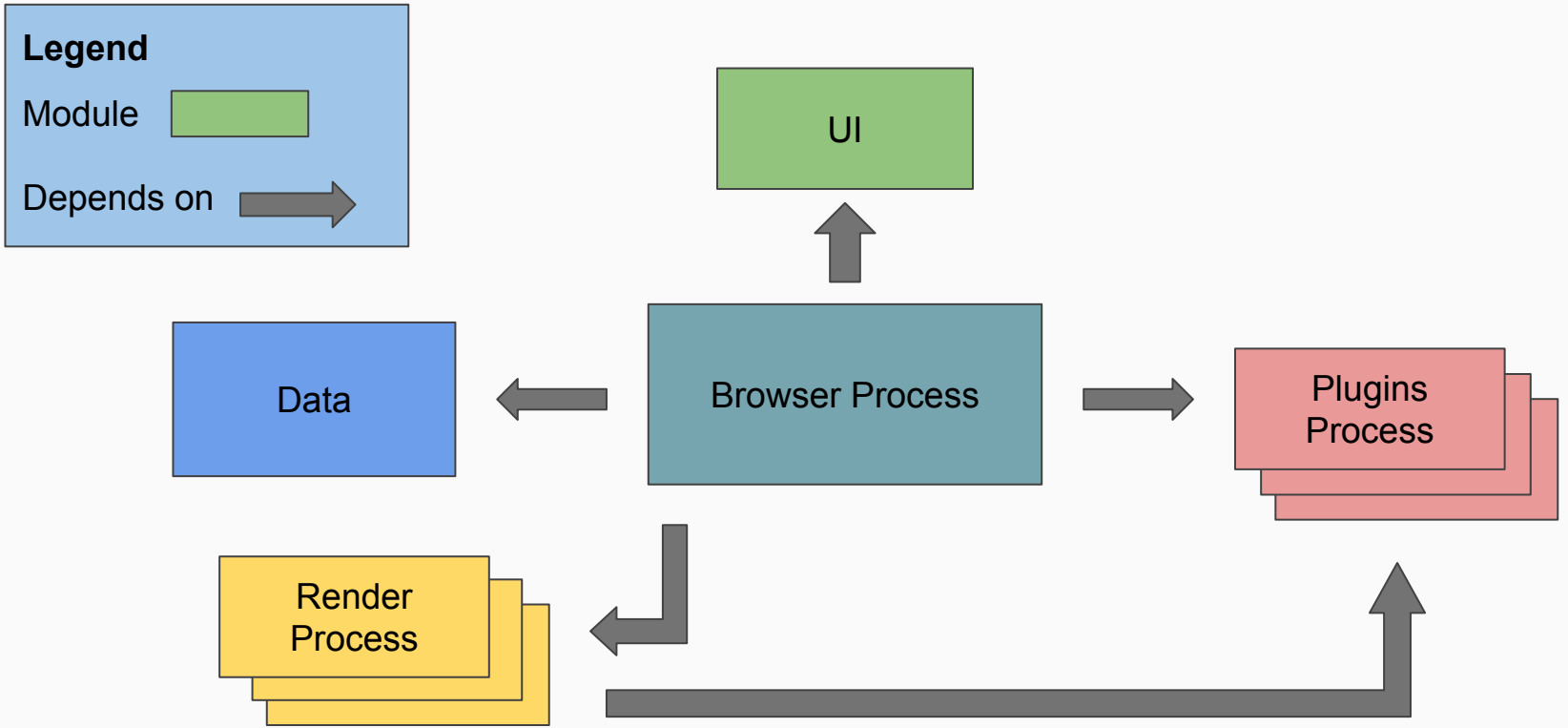
- Multiple conceptual architectures were made and compared.
- Some factors considered to derive our final architecture:

Architecture Style(s)

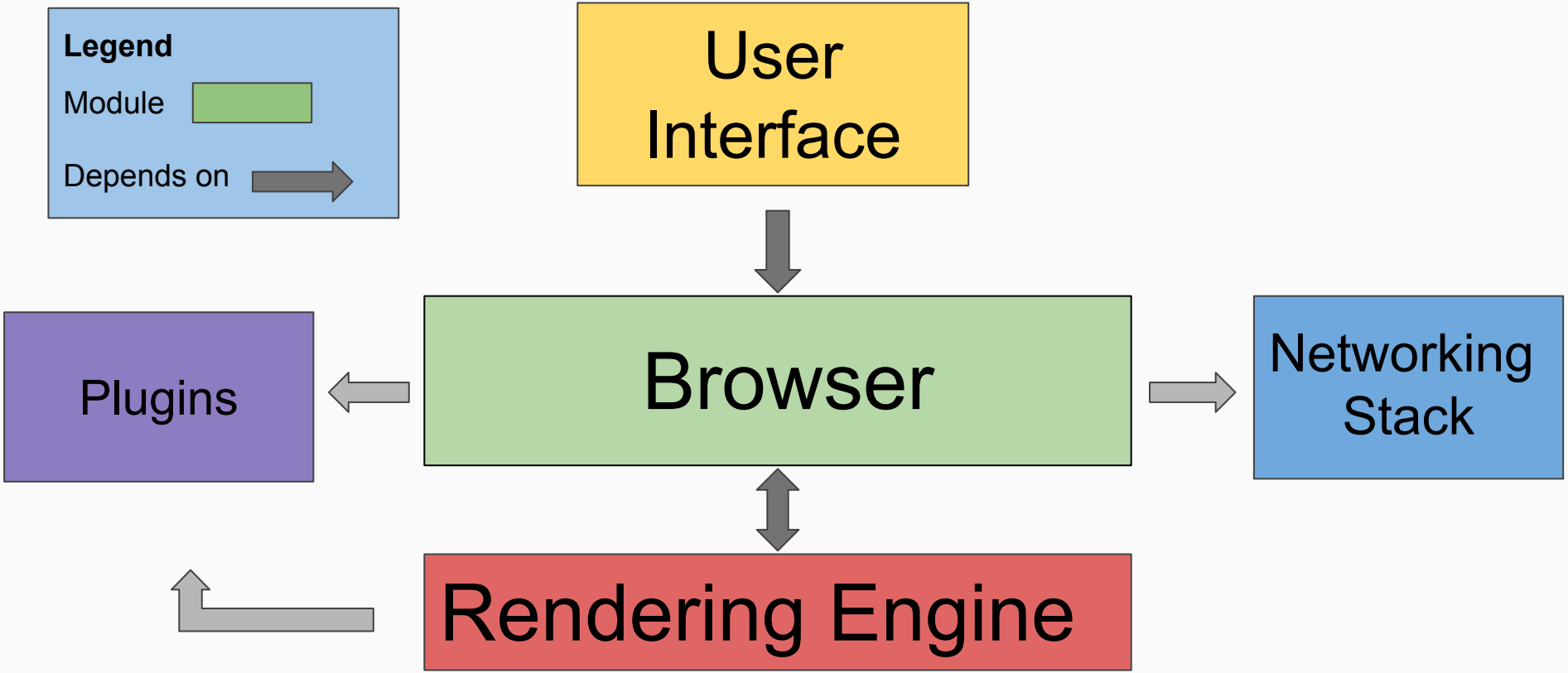
Multi-Process Architecture

Coupling and Cohesion

Alternative Conceptual Architecture



The Architecture



In-Depth Look at the Architecture

- **Layered** architecture to increase security between systems and increase modularity. Each system is a service to the layer above it.
- **Object-oriented** architecture to abstract systems. Change an implementation of an object without affecting its clients.

<table>
<tr><td rowspan="2">Browser
manages components of the application, interacts with the operating system and contains persistent storage.</td><td>Rendering System
Takes an HTTP response and renders a bitmap. Implements everything that renders content in a tab.</td><td>Plugins
Plugins run in a separate process to add additional third-party code.</td></tr>
<tr><td>Networking Stack
The network receives and resolves all network protocols.</td><td>User Interface
The link between the user and the browser.</td></tr>
</table>

Browser

- The Browser is responsible for the overall execution of the Chrome application
- It manages all of the components of the architecture and provides an interface for data to be exchanged across components
- Provides segregation between browser components which increases security and cohesion, while decreasing coupling
- Facilitates multiple processes occurring simultaneously (1 process per tab)
- Interacts with the operating system and contains persistent storage (bookmarks, history etc.)

Browser Dependencies

Component	Dependency
Rendering Engine	• The browser passes the Rendering Engine HTML, CSS, JavaScript and other data from the Network Stack so that it can be compiled together into a visual webpage that can be displayed by the User Interface
Network Stack	• The browser sends data requests to the network stack, which is responsible for communicating with the external internet to retrieve the data requested by the browser. • The Network Stack then passes the data back to the Browser, which will, in turn, send it to the Rendering Engine to be processed.
Plugins	• The browser relies on Plugins in order to incorporate third party software in order to add custom functionality to the browser. • If a plugin crashes, the entire browser will not crash as it is a separate subsystem.

Rendering Engine

- Implements everything that renders content inside a browser tab (includes parsing).
- Each tab typically runs its own renderer process.
- Runs in a sandbox with limited access to the operating system (no direct disk or network access).

Rendering Engine Dependencies

Component	Dependency
Browser	• rendering engine depends on the browser as it does not have direct access to persistent storage or network requests • increasing security in case the rendering engine is compromised • this decreases coupling and increases the cohesion of the networking and rendering engine system
Plugins	• one plugin for the entire instance of a web browser (not one per render process) • direct access for drawing and the presentation of the site with plugins • plugins are sandboxed now, less of a security threat (still some) • this dependency increases coupling for performance

Multi-Process Arch. & Concurrency

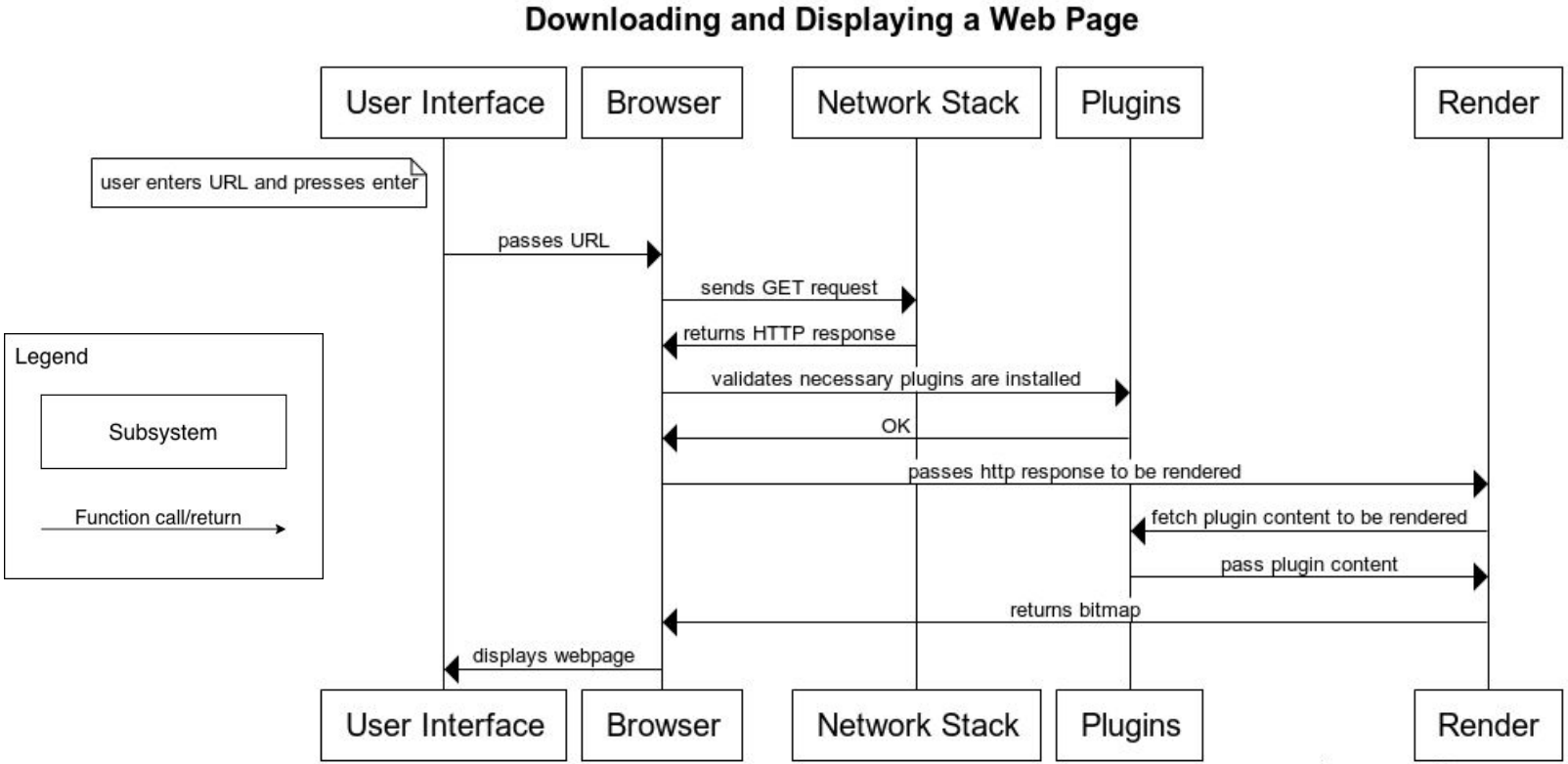
- Concurrency means multiple processes can run at the same time
- Multi-process architecture takes advantage of multi-core CPUs (which are now commonplace)

- Main purpose is to **increase execution speed**

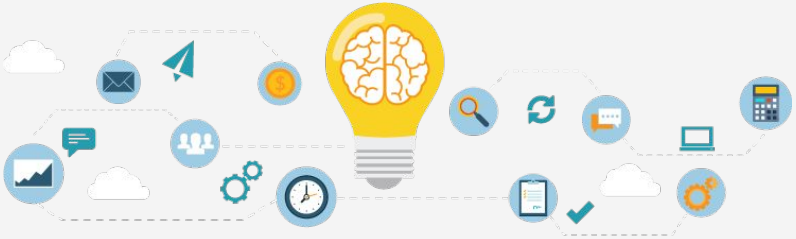
- **Reduces single points of failure**

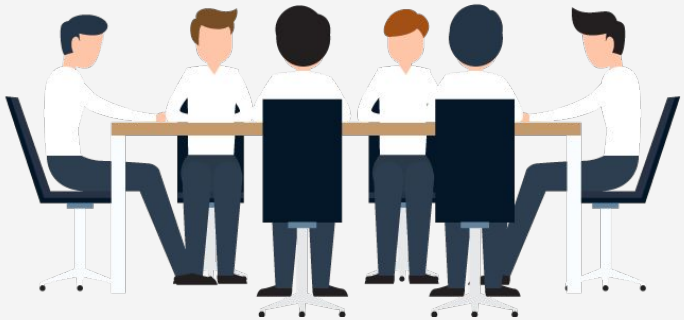
- **Increased memory usage**

Sequence Diagram



Team Issues





- **Chrome team decided to migrate initial IPC system to Mojo**
  - A lot of tight coupling to the IPC lead to a ripple effect when code was changed

- **Browser subsystem has multiple dependencies**
  - Team in charge of browser needs to interact more with other teams
  - Not as independent as other subsystem teams

Limitations & Lessons Learned

Limitations:

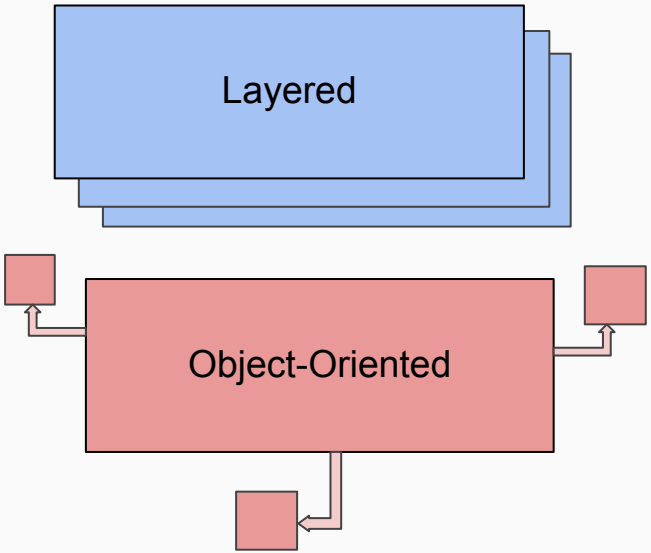
- A lot of new and old information available on the internet, which can make it overwhelming to research
- Chrome is a commercial product and not open source, so we had to look at Chromium, which is open source
- Most architecture documentation is from its release in 2008

Lessons:

- Large group sizes and busy schedules make it harder to organize meetings
    - However, group setting made it easier to discuss and develop ideas

Conclusion

Architecture Styles Used



Benefits

Security

Performance

Reliability

The multi-process architecture allows for concurrency.

Questions?