The Concrete Architecture of Chrome

Thick Glitches

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

Introduction

- The Concrete Architecture was developed using Understand.
- Chrome is made up of 5 distinct systems that interact together and uses an object-oriented style.
- We refined our original Conceptual Architecture, developed an alternative Concrete Architecture, and then produced our final Concrete Architecture.

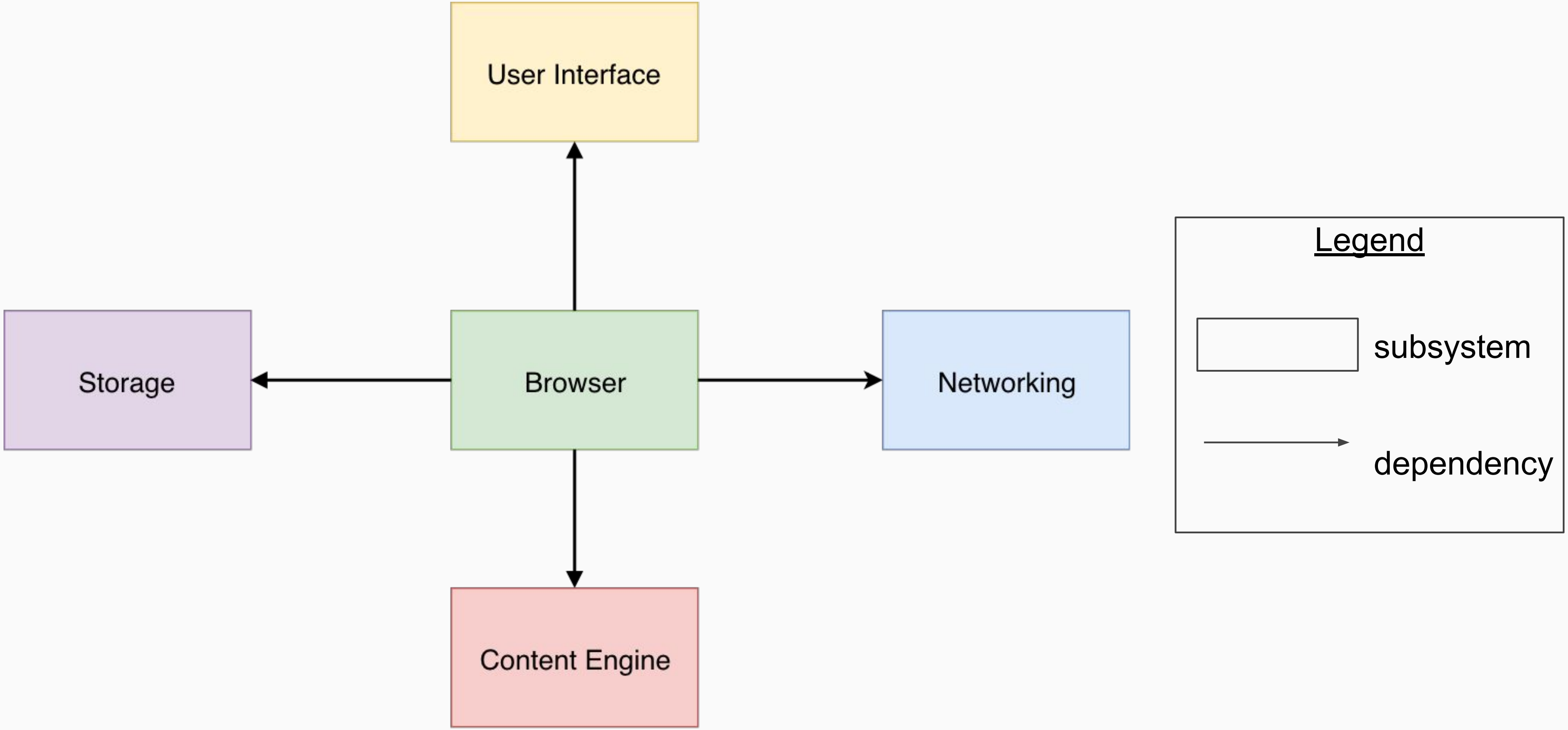
scitools™ Understand

Google

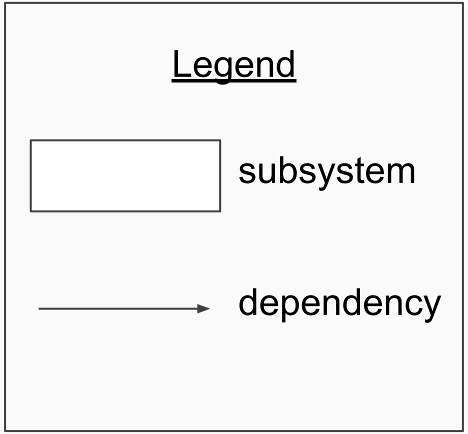
Derivation Process

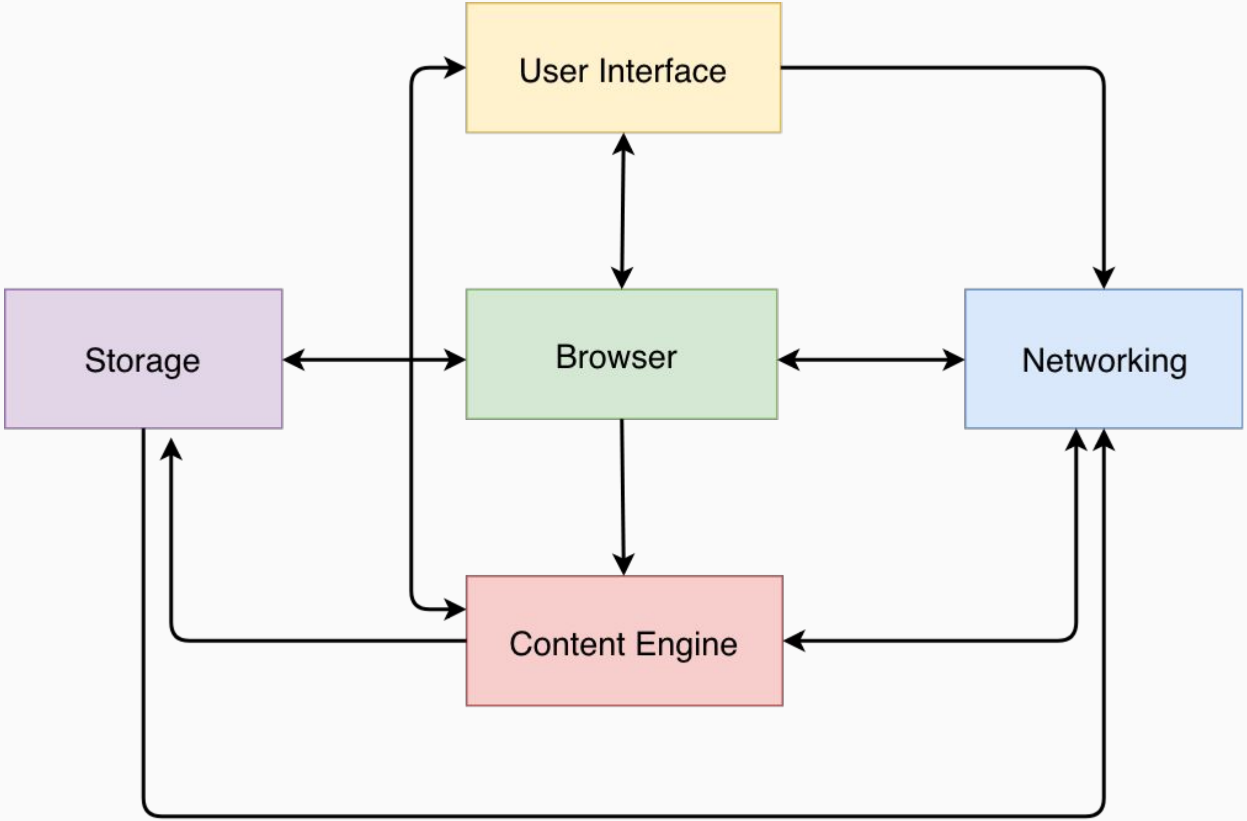
Part 1	Part 2	Part 3
Revised our conceptual architecture (subsystems and dependencies)	Observed the metrics tree map on Understand to determine the major subsystems and their dependencies	Came up with a possible concrete architecture and applied the reflexion model to derive the final version

Revised Conceptual Architecture

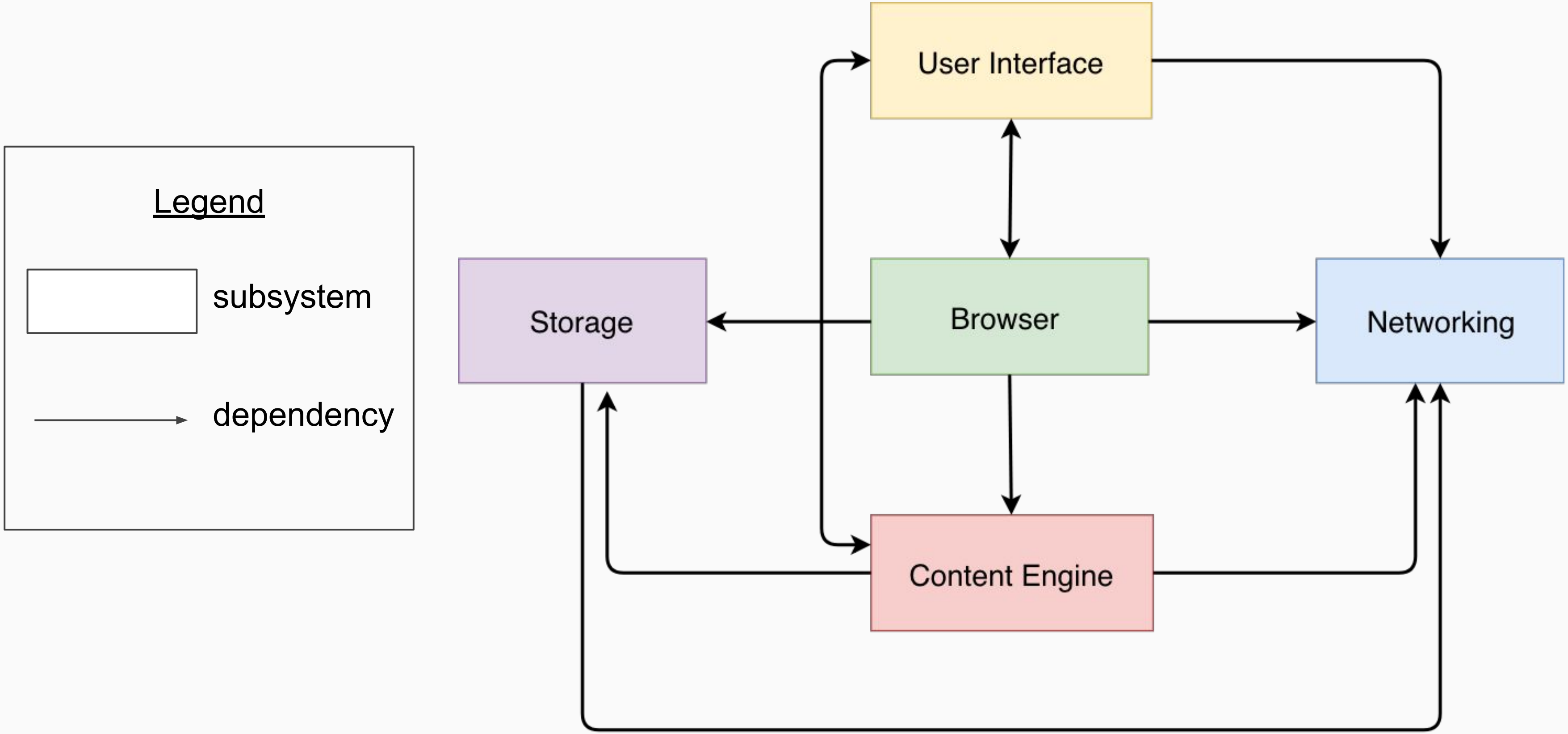


Alternative Concrete Architecture





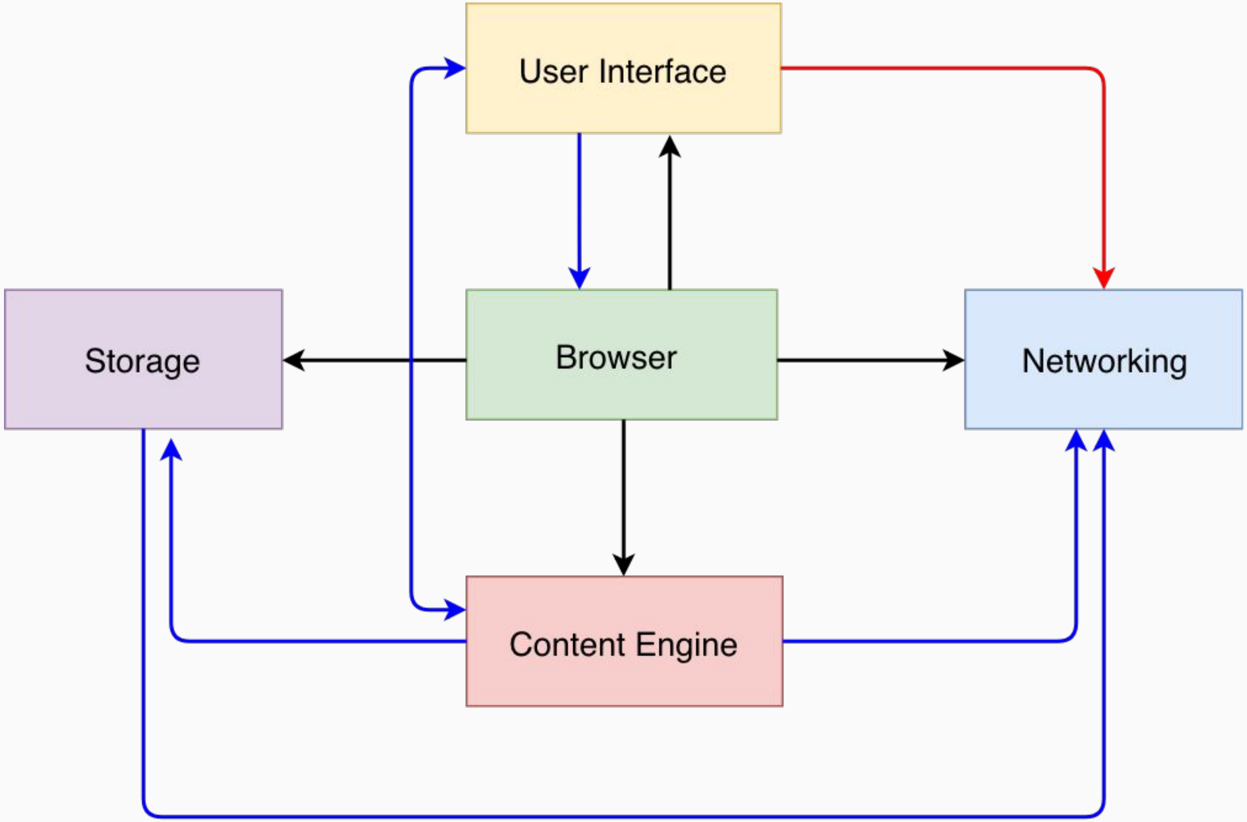
Concrete Architecture



Reflexion Analysis

Legend

- □ subsystem
- → expected & justified dependency
- → unexpected & justified dependency
- → unexpected & unjustified dependency

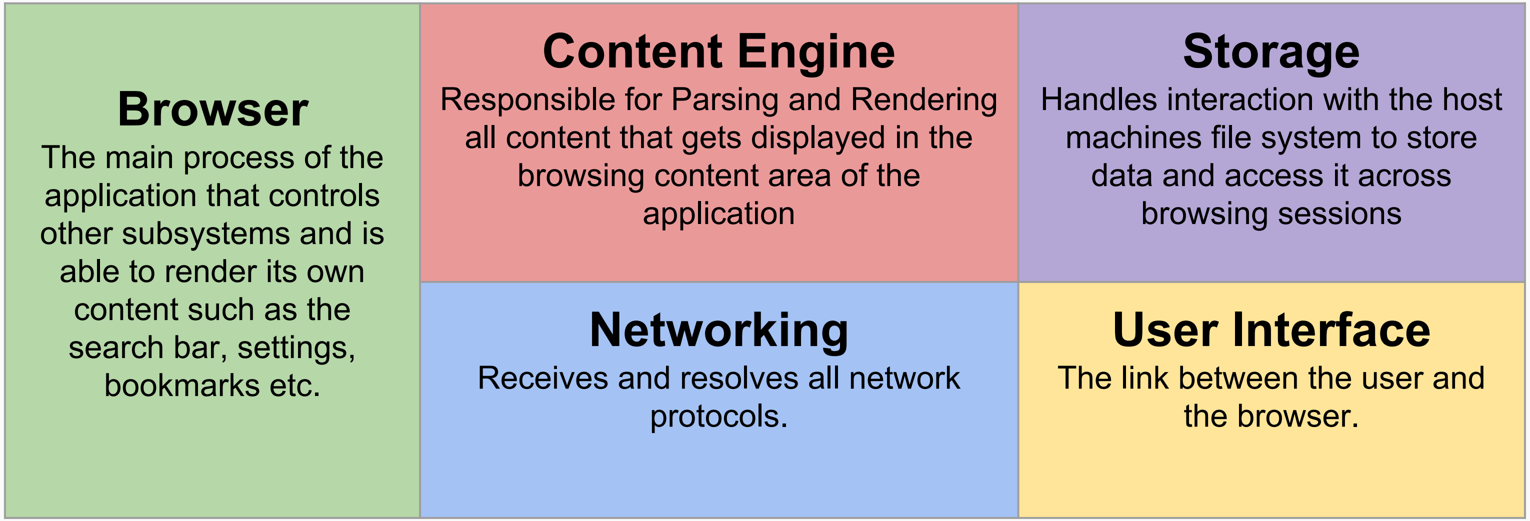


Reflexion Analysis Dependencies

Justified/Unjustified	Dependency From	Dependency To	Rationale
Unjustified	UI	Networking	Reuses networking's platform independent code for resolving local path names
Justified	Storage	Networking	Blob downloading/uploading. Blobs are not likely to be malicious. A lot of data to route through multiple subsystems, so a direct link is beneficial
Justified	UI	Browser	Using apple's framework to fill in the task bar. Chrome is responsible for drawing anything that isn't tab content
Justified	UI	Content Engine	Developer Tools. Chrome shell console, inspect element, etc
Justified	Content Engine	Storage	File API, blob storage, quota manager
Justified	Content Engine	Network	Web sockets, Hyperlinks, CDN, disk caching (unconfirmed downloads)
Justified	Content Engine	UI	Every time UI needs something painted/rendered, it communicates directly with UI. All event objects in UI are depended on by content engine

In-Depth Look at the Architecture

- **Object-oriented** architecture to abstract systems. Change an implementation of an object without affecting its clients.



Browser
The main process of the application that controls other subsystems and is able to render its own content such as the search bar, settings, bookmarks etc.

Content Engine
Responsible for Parsing and Rendering all content that gets displayed in the browsing content area of the application

Storage
Handles interaction with the host machines file system to store data and access it across browsing sessions

Networking
Receives and resolves all network protocols.

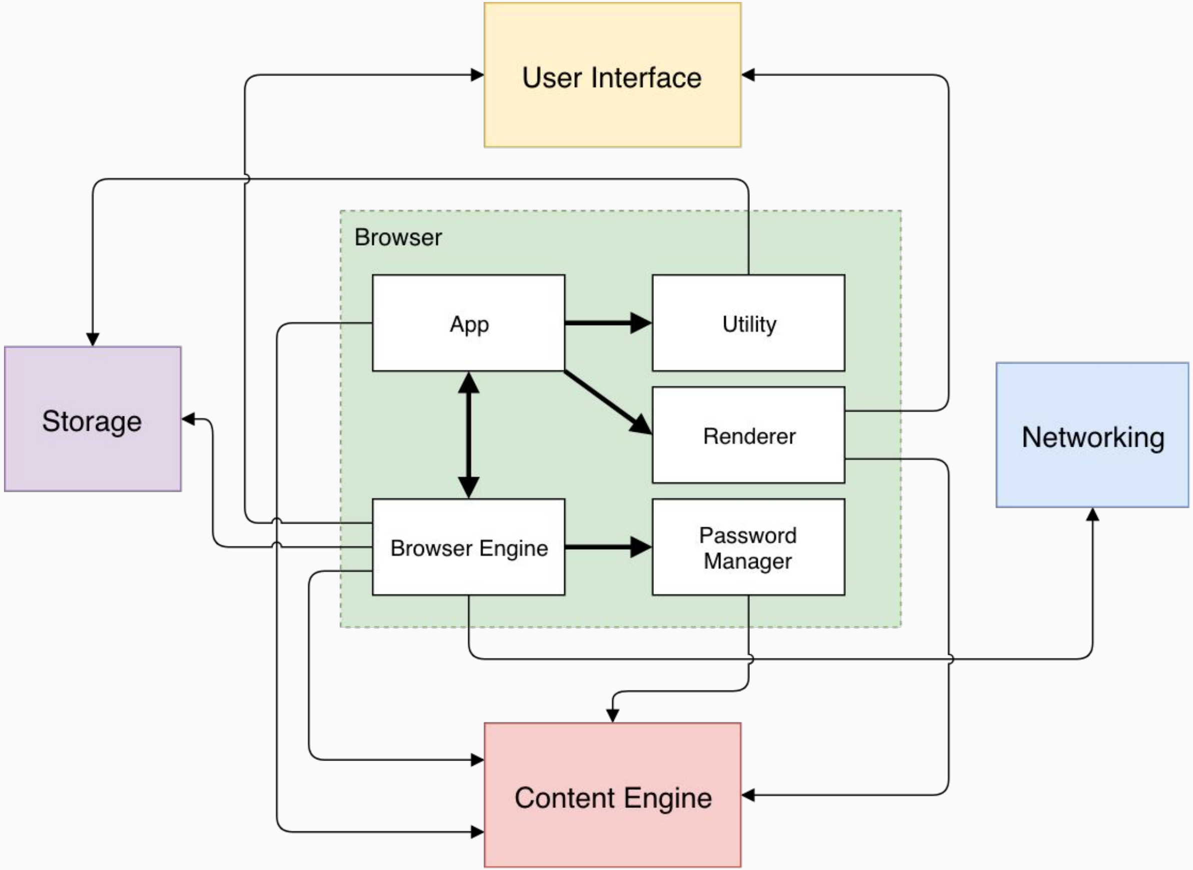
User Interface
The link between the user and the browser.

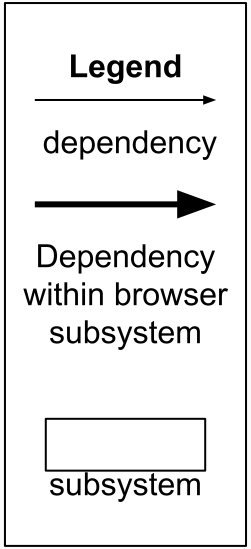
Browser in depth

- **App**
  - Lowest Level of the Chrome Application, runs on startup
  - Contains startup and shutdown files as well as crash reporters
- **Disk Utility**
  - Manages mounting and unmounting of file systems.
  - Contains functionality to import data from other browsers
- **Browser Engine**
  - Contains the code and files for all of Chrome's core functionality such as managing extensions, history, bookmarks, password manager, offline web pages, themes languages etc.
- **Renderer**
  - The browser has its own rendering process that draws all of the application other than the actual content being displayed
  - This includes the tabs, search bar, settings, and tools
- **Password Manager**
  - Facilitates the storage and retrieval of usernames and passwords
  - Interfaces with the storage module in order to access persistent storage on the host machine

Conceptual Arch. w/ In-Depth View of Browser

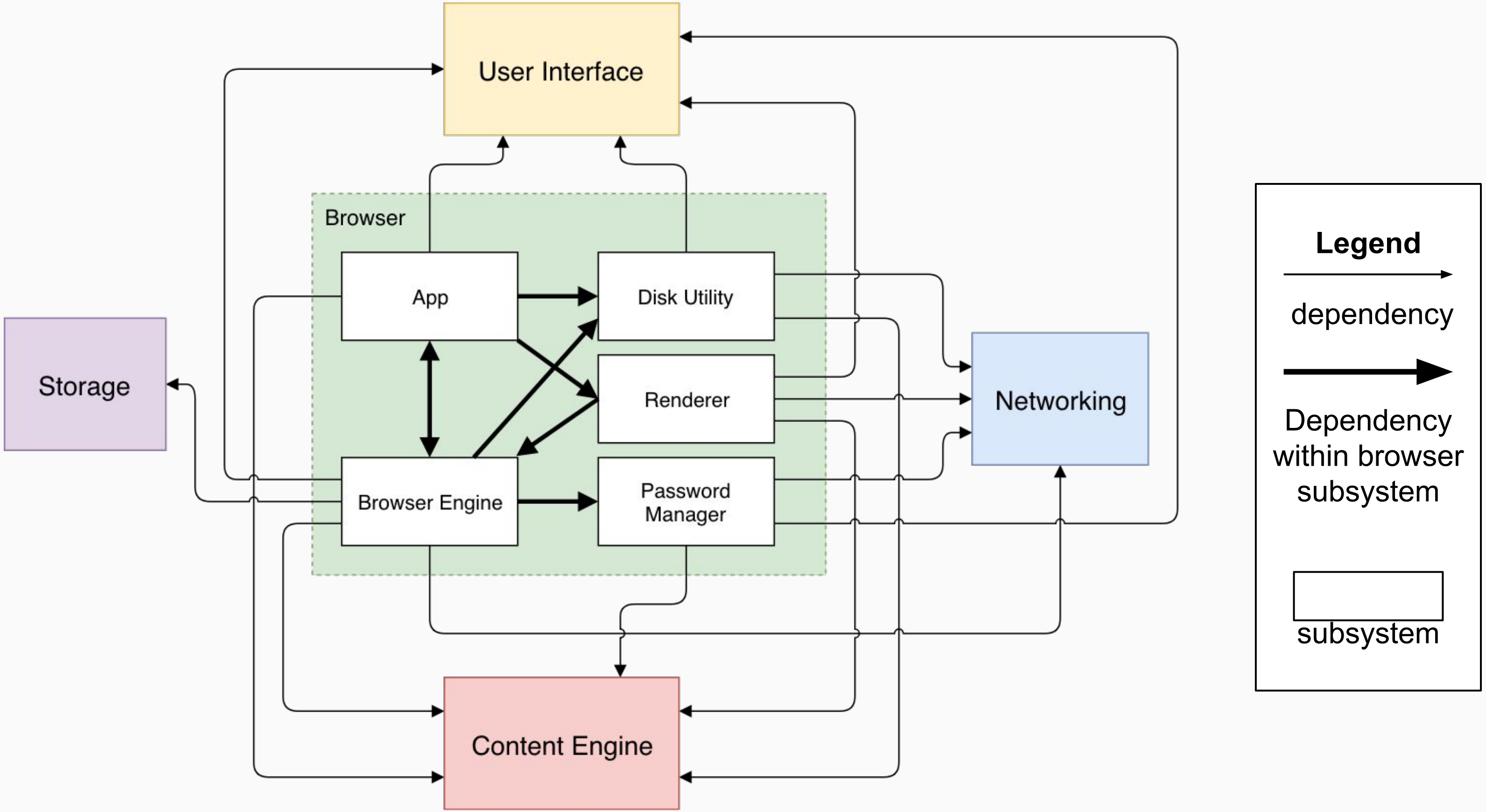
***Conceptual* Browser subsystem architecture**





Concrete Arch. w/ In-Depth View of Browser

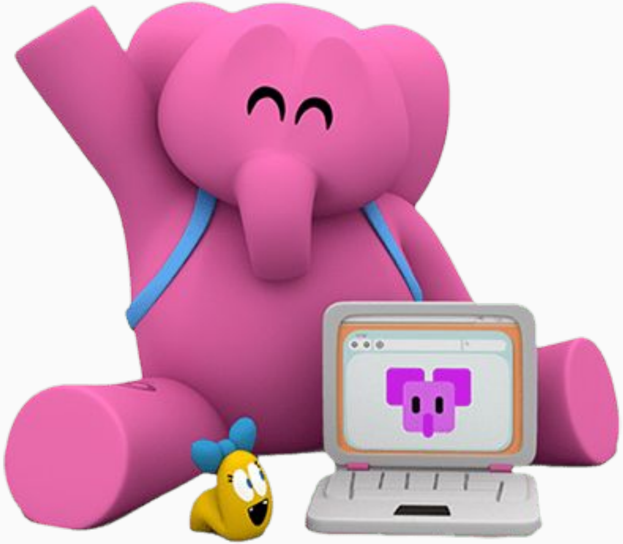
***Concrete* Browser subsystem architecture**



Proposed Feature

Chrome Safe Mode

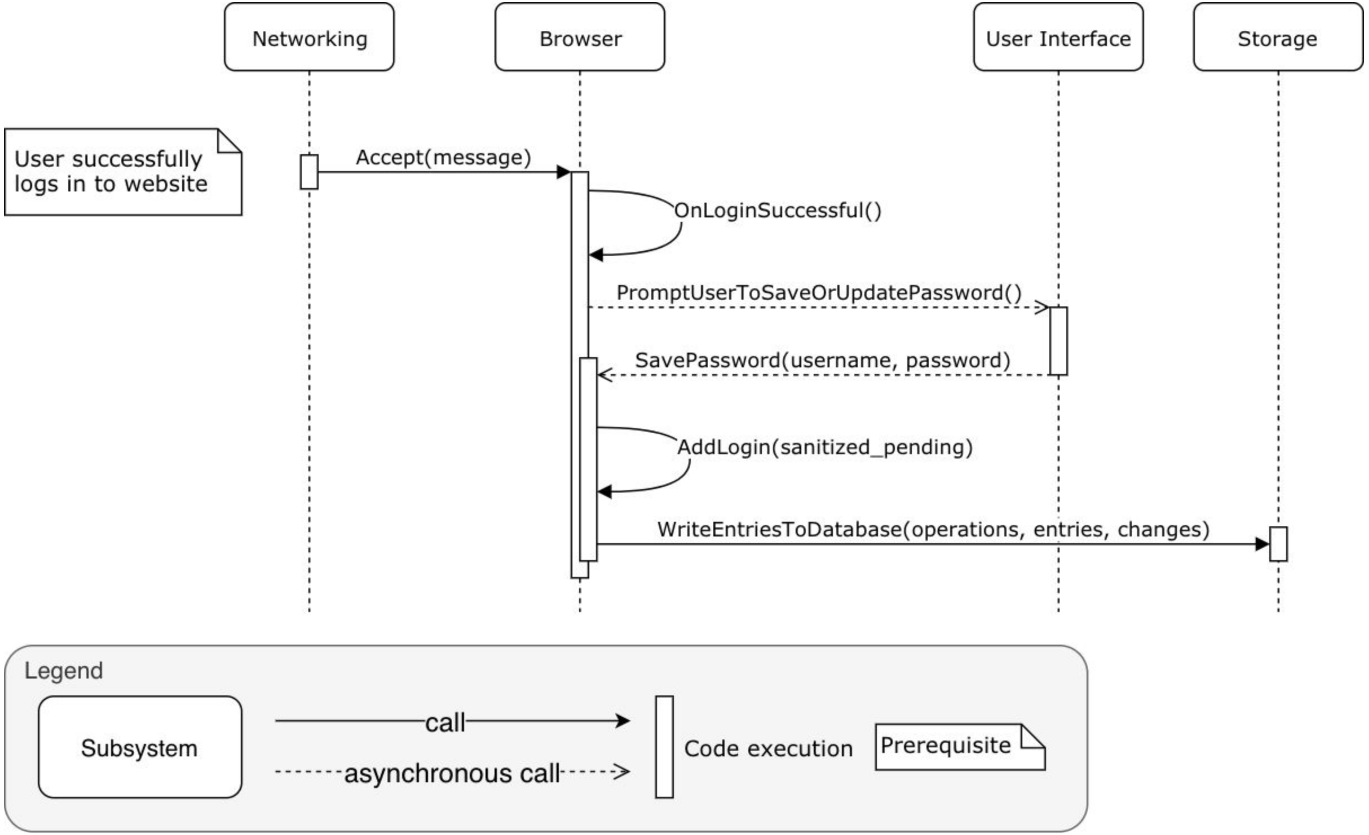
- Ideal for kids or people at work
- Allows user to censor instances of pre-set blacklisted words on a webpage or entire websites deemed as inappropriate
- Can be activated/deactivated with a user entered password
- Blocks out inappropriate content to be displayed
  - Ex. images
- Utilizes the Browser, UI, Storage and Content Engine systems



The Effects of Concurrency

- **What does it allow Chrome to do?**
  - **Sandboxing processes**
    - Async requests confirm failed processes don’t block browser I/O thread
    - Restrict processes network requests and system access by facilitating requests through single access point
  - **Execution speed increases**
    - Requests to access data made by processes independent of one another
- **How is concurrency achieved?**
  - Single process parent browser process manages child processes (render processes)
  - Facilitate communication of render processes to various subsystems (i.e. network) through IPC from child to parent
  - Non-blocking asynchronous requests made by render processes guarantees that concurrency achieved

Sequence Diagram



Networking
Browser
User Interface
Storage
User successfully logs in to website
Accept(message)
OnLoginSuccessful()
PromptUserToSaveOrUpdatePassword()
SavePassword(username, password)
AddLogin(sanitized_pending)
WriteEntriesToDatabase(operations, entries, changes)
Legend
Subsystem
call
asynchronous call
Code execution
Prerequisite

Team Issues

- Unjustified or unclear dependencies between systems
    - makes it difficult for developers working across systems
- More dependencies in the concrete architecture
    - leads to suboptimal coupling
    - teams need to communicate efficiently
- Using Mojo as an IPC system instead of the original proprietary system improved inter- and intra- process handling which made concurrently running systems easier for developer teams to work with
    - However, they needed to migrate the initial IPC system to Mojo

Limitations & Lessons Learned

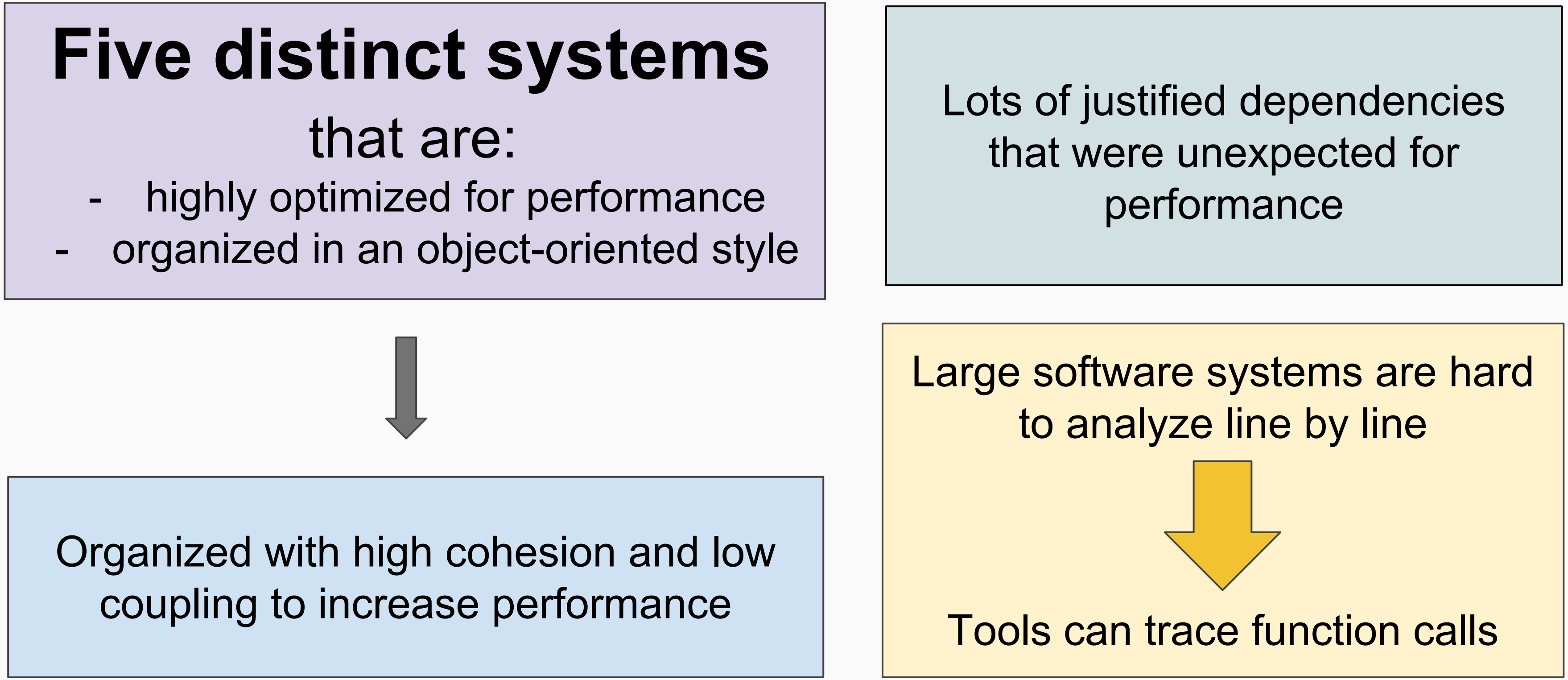
Limitations:

- Source code can be extremely overwhelming and confusing
- Hard to know scope, and when to stop going through function calls
- Source code is in C++, a language none of us were very experienced with
  - Limited comments in the source code
- Steep learning curve at the beginning with Understand
  - Would crash constantly before we sought out help to learn how to use Understand properly
- Had to meet twice the amount of times for A2 vs. A1 due to amount of information we had to go through and the more technical nature of the information

Lessons Learned:

- Understand was helpful once we got help from the TAs
- Learned some C++ syntax
- Patience and organization
- It’s ok to ask for help

Conclusion



Five distinct systems

that are:

- highly optimized for performance
- organized in an object-oriented style

Organized with high cohesion and low coupling to increase performance

Lots of justified dependencies that were unexpected for performance

Large software systems are hard to analyze line by line

Tools can trace function calls

Questions?