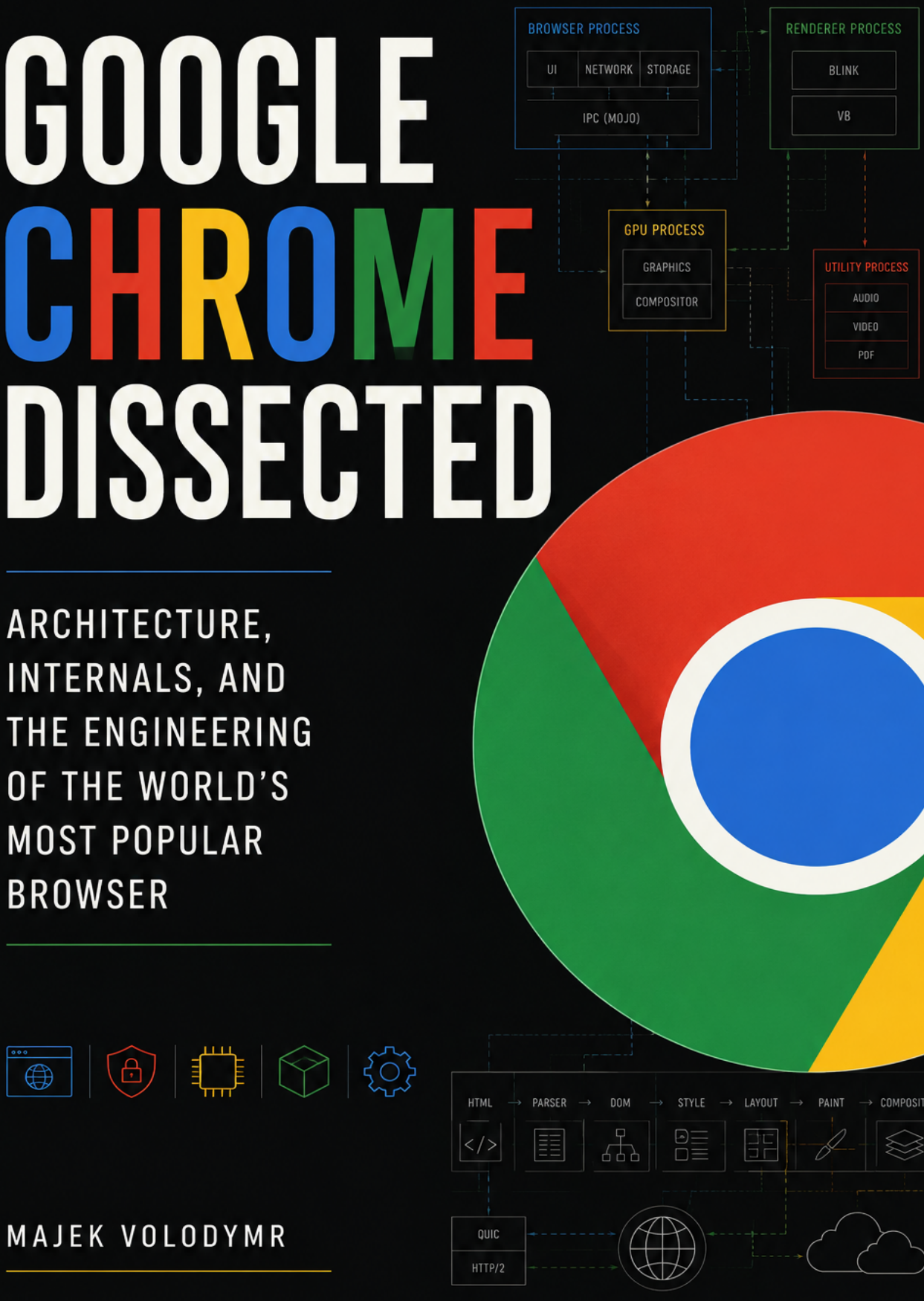
GOOGLE CHROME DISSECTED

ARCHITECTURE, INTERNALS, AND THE ENGINEERING OF THE WORLD'S MOST POPULAR BROWSER



MAJEK VOLODYMR

Google Chrome Dissected

Architecture, Internals, and the Engineering of the World's Most Popular Browser

Steve T. Publications

This book is available at https://leanpub.com/googlechromedissected

This version was published on 2026-07-13

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

Contents

Architecture, Internals, and the Engineering of the World's Most Popular Browser

Google Chrome is the most widely used web browser on Earth, serving billions of users across desktops, phones, tablets, and embedded systems. Behind its minimalist interface lies one of the most sophisticated pieces of software ever built: a multi-process application that parses HTML, executes JavaScript at near-native speed, renders graphics on the GPU, manages network connections over QUIC and HTTP/2, enforces security policies against hostile web content, and does all of this while consuming memory and CPU with deliberate trade-offs between performance, stability, and security. This book takes you inside every major subsystem of Chrome and its open-source foundation, the Chromium project. You will learn how the browser process coordinates dozens of child processes, how Blink parses and renders web pages through a carefully pipelined architecture, how V8 compiles JavaScript to optimized machine code in real time, how the GPU process accelerates graphics and compositing, how Mojo enables inter-process communication across privilege boundaries, and how sandboxing mechanisms on every major platform confine potentially malicious web content. Along the way, we examine real vulnerability case studies, compare Chrome's design choices with Firefox, Safari, and Edge, and explore where browser architecture is heading next. This book is written for software engineers, security researchers, systems programmers, and advanced web developers who want to understand not just how Chrome works, but why it was designed the way it was.

Introduction: The Browser as Operating System

In 2008, when Google released the first beta of Chrome, the browser landscape looked nothing like it does today. Internet Explorer held roughly seventy percent of the market, Firefox was a scrappy challenger with about twenty percent, and Safari was confined to Apple's ecosystem [46]. Nobody expected a newcomer to disrupt this order, let alone come to dominate it within five years. But Chrome did exactly that, and the reason was not marketing or distribution, though both certainly helped. The reason was architecture.

Chrome introduced a radical idea to the browser world: every tab runs in its own operating-system process, isolated from other tabs and from the browser's core UI by memory protection boundaries and security sandboxes. If one tab crashed, the others kept working. If a web page contained malware, the sandbox limited what it could do to the user's machine. This multi-process model was not just an incremental improvement. It was a fundamental rethinking of what a browser should be, treating each web page not as a document but as an untrusted program that needed to be confined.

That architectural decision rippled through every subsystem in the browser. It drove the design of the inter-process communication system, now embodied in the Mojo framework. It shaped how rendering works, with Blink's pipeline designed to hand off work between the main thread, compositor thread, and GPU process. It influenced V8's compilation strategy, where type feedback collected during interpretation feeds into optimizing compilers that generate speculative machine code. It even affected Chrome's networking stack, which was extracted into a separate network service process to prevent compromised renderers from making arbitrary network connections.

Understanding Chrome means understanding this architecture and the trade-offs it entails. The multi-process model improves stability and security but adds complexity to inter-process communication, increases memory overhead, and requires careful coordination across process boundaries. Every subsystem in Chrome reflects these tensions.

This book is organized to take you from the broad strokes down to implementation detail. We begin with Chrome's history and the philosophical decisions that shaped it, then examine the high-level architecture and multi-process model in depth. From there, we dive into each major subsystem: the browser process that orchestrates everything, the renderer process and Blink engine that parse and render web pages, V8 the JavaScript virtual machine, the GPU process and graphics pipeline, the network service, the storage subsystem, and the IPC mechanisms that tie them all together. We then turn to security, examining sandboxing on every platform, site isolation, web security policies, and real vulnerability case studies. Finally, we look at Chrome's testing infrastructure, its update mechanism, and where browser architecture is heading next.

Throughout, we compare Chrome's design choices with those of Firefox (which uses a different multi-process model centered around the Gecko engine), Safari (which integrates tightly with Apple's platforms through WebKit and its own process model), and Microsoft Edge (which shares Chromium's foundation but diverges in areas like AI integration and enterprise features). These comparisons illuminate why there is no single "right" way to build a browser, only trade-offs shaped by different priorities, platform constraints, and engineering cultures.

By the end of this book, you will understand Chrome not as a black box but as a collection of interconnected systems, each solving hard problems with deliberate design choices. You will know how to read Chrome's source code, how to diagnose rendering and performance issues, how to evaluate security vulnerabilities, and how the browser's architecture will evolve as the web platform grows more powerful. You will, in short, understand one of the most important pieces of software in the modern world.

Chapter 1: The Origin Story

The Browser Wars Before Chrome

To understand why Chrome was designed the way it was, you need to understand what the web looked like in 2006. Microsoft's Internet Explorer dominated with roughly seventy percent of global market share, a position it had held since the late 1990s after defeating Netscape in the first Browser Wars. But IE's dominance came at a cost. Development had stagnated; the last major version, IE 6, was released in 2001 and remained the default on Windows XP for years. Standards support was patchy, security was an afterthought, and Microsoft had little incentive to innovate when its browser was essentially locked in by default installation on the world's most popular operating system.

Mozilla Firefox emerged as the principal alternative, launching in 2004 with a modern extension architecture, tabbed browsing, and strong standards compliance. By 2006, Firefox had captured about twenty percent of the market, growing steadily among power users and developers. Apple's Safari, built on the open-source WebKit rendering engine (itself a fork of KHTML from the Konqueror project), was gaining traction on macOS but remained largely absent from Windows. Opera, once an innovator with features like speed dial and tabbed browsing years before competitors, had dwindled to single-digit market share.

For Google, this landscape presented both a problem and an opportunity. As Larry Page and Sergey Brin reportedly told their engineers, the browser was critical to Google's business because "our entire business is people using the web." A slow or buggy browser meant fewer searches, fewer AdSense impressions, and a degraded experience for Google's growing suite of web applications including Gmail, Google Docs, and Google Maps. Firefox, while improving, still had performance problems with JavaScript-heavy applications. Internet Explorer was simply not fast enough, and its security vulnerabilities were well documented.

The Google Browser Team Assembles

Google began recruiting engineers for a secret browser project codenamed "Chrome" around 2006. The team included several notable figures who would shape the browser's architecture:

Ben Goodger, formerly of the Mozilla Firefox team, was brought in as the project lead. Goodger had been one of Firefox's principal architects and understood both the strengths and weaknesses of existing browser designs. He authored Chrome's original design document and drove much of the user interface philosophy that made Chrome distinctive: minimal chrome (the UI elements around the web content), a single omnibox combining URL bar and search field, and an emphasis on speed.

Dean Kaminsky, a security researcher who had co-discovered the BEAST attack against TLS, led the security architecture. Kaminsky's influence is visible in Chrome's sandboxing design and its threat model, which assumes that web content is inherently hostile and must be confined.

Lars Bak, another Mozilla alumnus who had worked on JavaScript engines including TraceMonkey, was tasked with building a new JavaScript engine from scratch. The result was V8, an engine designed around just-in-time compilation rather than interpretation, with aggressive optimizations like inline caching and hidden classes that would make Chrome's JavaScript performance dramatically faster than competitors.

Mark Larson led the graphics team, driving Chrome's early adoption of GPU-accelerated compositing, which would become a key differentiator for rendering performance.

The team also included Brian Ryner, who had worked on Firefox's networking stack, and several engineers from Google's search and advertising teams who understood the performance requirements of large-scale web applications.

September 2, 2008: Launch Day

Chrome launched as a public beta on September 2, 2008, for Windows only [46]. The launch was accompanied by an unusual marketing piece: a web comic that explained Chrome's design philosophy in accessible terms, contrasting the

browser's multi-process architecture with the "monolithic" model of existing browsers. The comic showed how in traditional browsers, all tabs shared a single process, meaning a crash in one tab crashed the entire browser. In Chrome, each tab ran in its own sandboxed process, isolated from others [46].

Technically, Chrome 1.0 was impressive for its time:

- **V8 JavaScript engine**: Benchmarks showed V8 outperforming Firefox's SpiderMonkey and Safari's SquirrelFish by significant margins on JavaScript-heavy workloads. Google demonstrated this with a "Hello World" comparison showing Chrome rendering a complex JavaScript animation smoothly while Firefox struggled.
- **Multi-process architecture**: Each tab ran in a separate renderer process, sandboxed from the browser process and from each other. The browser process handled the UI, tab management, and coordination, while renderer processes handled HTML parsing, layout, and painting through the WebKit rendering engine.
- **Minimal UI**: Chrome's interface was radically simplified compared to Firefox and IE. There were no menus, no toolbars (beyond the omnibox), and no status bar. The design philosophy was "content first" – the browser's own UI should get out of the way of the web content.
- **Automatic updates**: Chrome included a background update mechanism that would silently download and install new versions, solving the problem of users running outdated, vulnerable browser versions.
- **Web Standards**: Chrome shipped with strong support for HTML5 features including `<canvas>`, geolocation, and application cache, positioning it as the browser for the modern web.

The initial version was built on WebKit, Apple's open-source rendering engine. Google had initially considered using Gecko (Firefox's engine) but chose WebKit because of its cleaner architecture and active development. However, Google's multi-process model required significant modifications to WebKit, creating tension with the upstream project that would eventually lead to the Blink fork in 2013.

Open-Sourcing Chromium

On December 1, 2008, just three months after Chrome's beta launch, Google released the source code for "Chromium," the open-source project from which Chrome is built [47,50]. This was a strategic decision with multiple motivations:

First, it invited community contribution and scrutiny. By making the browser open-source, Google could leverage external developers to find bugs, contribute features, and port the browser to new platforms. It also signaled good faith to the web standards community, showing that Google was committed to an open web rather than proprietary extensions.

Second, it created a foundation for browser innovation beyond Chrome itself. Chromium would eventually become the basis for Microsoft Edge (starting in 2018), Opera, Vivaldi, Brave, and dozens of other browsers, effectively establishing a de facto standard for browser architecture across the industry.

Third, the open-source model facilitated Google's business goals. By making the underlying engine available to anyone, Google encouraged web developers to optimize for Chromium-based browsers, which in turn benefited Google's web services and advertising platform.

The distinction between Chrome and Chromium is important and often confused. Chrome is the branded product that includes additional components: the automatic update client (Omaha), media codecs (including proprietary formats like H.264), the Flash plugin (historically), brand-specific UI elements, and telemetry reporting back to Google. Chromium is the open-source project without these additions. Most technical documentation, design decisions, and architectural discussions apply to both, since the core browser engine is identical.

From Niche to Dominance

Chrome's growth was astonishing. In its first year, it went from zero to roughly five percent of global market share. By 2012, it had surpassed Firefox to become the second most popular browser worldwide. By 2014, it had overtaken Internet Explorer as the most used browser on desktop platforms. Mobile adoption accelerated the trend: Chrome for Android, released in 2012, became the default browser on billions of Android devices, while Chrome for iOS (constrained by Apple's App Store policies to use WebKit rather than Blink) captured significant share among iPhone users who preferred Chrome's sync and extension ecosystem [46,49].

Several factors drove this growth:

Performance: V8's JavaScript speed gave Chrome a tangible advantage on increasingly complex web applications. The multi-process model meant

that heavy pages didn't slow down the entire browser. GPU-accelerated compositing made animations and transitions smooth.

Extensions: Chrome Web Store, launched in 2011, provided a curated marketplace for extensions that ran in their own sandboxed processes. This gave Chrome a rich ecosystem of productivity tools, ad blockers, developer utilities, and entertainment extensions.

Sync: Chrome's cross-device synchronization of bookmarks, history, passwords, and open tabs created a sticky user experience that was difficult to replicate across browsers.

Automatic updates: Unlike Firefox (which required manual updates in its early days) or Internet Explorer (which depended on Windows Update), Chrome updated silently in the background, ensuring users always had the latest security patches and features.

Developer tools: Chrome DevTools, introduced in 2010, quickly became the industry standard for web debugging. Its Elements panel, Network tab, Performance profiler, and Console were more polished and feature-rich than competitors' developer tools, attracting web developers who then used Chrome as their primary browser.

By 2024, Chrome's global desktop market share hovered around sixty-five percent, with even higher shares on Android. Combined with Chromium-based browsers like Edge, Opera, and Brave, the Chromium ecosystem accounts for well over eighty percent of all browser usage worldwide. This dominance has significant implications for web standards development, as the Chromium team effectively controls the implementation of most new web platform features.

The Plugin Evolution: From NPAPI to WebAssembly

Chrome's approach to plugins represents one of the most consequential security architecture decisions in browser history. The evolution from NPAPI through PPAPI to modern web standards fundamentally reshaped Chrome's attack surface and influenced its multi-process design.

NPAPI (Netscape Plugin API) was the original plugin standard, inherited from Netscape Navigator and adopted by Firefox, Safari, and early Chrome. NPAPI plugins ran in the browser's main process with essentially unrestricted access to memory, the file system, and the operating system. This made them

incredibly powerful but also extraordinarily dangerous. A single vulnerability in an NPAPI plugin (such as Adobe Flash or Java) could compromise the entire browser and, by extension, the user's machine. In fact, NPAPI plugins were consistently the most exploited browser component, accounting for a majority of browser-based malware infections during the 2000s [46].

PPAPI (Pepper Plugin API) was Google's response. Introduced in Chrome 17 (2011), PPAPI was designed from the ground up for the multi-process model. Key architectural differences:

- PPAPI plugins ran in their own sandboxed processes, isolated from the browser and renderer processes.
- The plugin could not directly access the file system, network, or GPU. All privileged operations went through IPC to the browser process.
- PPAPI provided a restricted, well-defined API surface that plugins could use, rather than granting arbitrary system access.
- Memory management was more controlled, with explicit allocation and deallocation through the plugin host.

Flash was the most prominent PPAPI plugin, running as `pepperflash` in its own sandboxed process. Chrome's implementation of Flash (Pepper Flash) was significantly more secure than the NPAPI version used by Firefox and IE, as the sandbox limited what a compromised Flash instance could do.

Native Client (NaCl) was Google's ambitious attempt to provide a secure platform for executing native code in the browser. NaCl modules were compiled from C/C++ into a portable object format (`.nexe`) that could execute in a sandboxed environment without requiring platform-specific binaries. The architecture included:

- A simplified instruction set with restricted system calls.
- A "trusted" component written in Safe Language Subset (NaCl SSA) that mediated between the native code and the browser.
- Integration with PPAPI for browser interaction.

Native Client was eventually superseded by **Portable Native Client (PNaCl)**, which used LLVM bitcode as an intermediate representation, enabling true cross-platform compilation. However, PNaCl never achieved widespread adoption and was deprecated alongside NaCl in Chrome 89 (2021).

The NPAPI deprecation timeline was aggressive and definitive:

- **January 2014 (Chrome 32)**: NPAPI plugins blocked by default, requiring click-to-activate prompts [46].
- **April 2015 (Chrome 42)**: NPAPI support disabled entirely; extensions requiring NPAPI unpublished from Chrome Web Store.
- **September 2015 (Chrome 45)**: NPAPI code permanently removed from Chrome. The override mechanism was eliminated.

The removal of NPAPI dramatically reduced Chrome's attack surface. Plugins had been the number one vector for browser exploits, and their elimination forced developers to adopt web standards (HTML5, WebRTC, WebGL, WebAssembly) that run within the sandboxed renderer process.

WebAssembly as the modern successor: WebAssembly (Wasm), standardized in 2017, provides the high-performance, sandboxed execution environment that NPAPI and NaCl attempted to deliver. Unlike its predecessors, Wasm:

- Runs natively within the renderer process's sandbox, inheriting all of Chrome's memory protection and privilege restrictions.
- Provides near-native performance through efficient binary compilation from C, C++, Rust, and other languages.
- Supports SIMD, threading (via SharedArrayBuffer), garbage collection (WasmGC), and the Component Model for language interoperability.
- Can be compiled ahead-of-time (WasmToAOT) or just-in-time, with streaming compilation as modules download.

The transition from NPAPI to WebAssembly represents a fundamental shift in Chrome's security philosophy: rather than granting native code special privileges, all execution – whether JavaScript, Wasm, or native – is subject to the same sandbox boundaries. This principle of uniform confinement is central to Chrome's defense-in-depth strategy.

PPAPI deprecation: Following NPAPI's removal, PPAPI was also phased out. Chrome 89 (2021) removed PPAPI support entirely, with Flash being the last significant PPAPI plugin (itself deprecated in December 2020). The PPAPI codebase was removed from Chromium, further simplifying the browser's architecture and reducing the attack surface associated with plugin-host communication.

The plugin evolution timeline:

```
2008-2015: NPAPI era (unrestricted, single-process plugins)
     |-> Flash, Java, Silverlight, QuickTime
     |
2011-2021: PPAPI era (sandboxed, multi-process plugins)
     |-> Pepper Flash, Native Client, PNaCl
     |
2017-present: WebAssembly era (sandboxed execution within renderer)
     |-> Wasm modules, WASI experiments, Component Model
```

The journey from NPAPI to WebAssembly is not just a story of technological replacement. It is a story of architectural maturation: Chrome gradually moved from trusting native code with system-level access to confining all execution within well-defined security boundaries. This trajectory informs Chrome's current approach to every new feature, from WebGPU to WebNN, ensuring that power and performance never come at the cost of security isolation.

The journey from a secret Google project in 2006 to the world's dominant browser in 2024 is remarkable, but what makes Chrome truly interesting to engineers and researchers is not its market success but its architecture. The multi-process model, the V8 engine, the Blink rendering pipeline, the sandboxing mechanisms, and the inter-process communication framework represent a coherent engineering vision that has proven remarkably durable. Understanding this architecture is the subject of this book.

Chapter 2: Architecture at a Glance

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Design Philosophy: Stability, Security, Responsiveness

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

The Process Taxonomy

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Component Architecture Overview

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

How Chrome Compares: Firefox, Safari, Edge

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

The Chromium Project Structure

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Chapter 3: The Multi-Process Model

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Why Multiple Processes?

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Process Types and Their Roles

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Process Lifecycle Management

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

The Site Instance Model

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Resource Process and Specialized Workers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Chapter 4: The Browser Process

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

The Browser Process as Orchestrator

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

TabStripModel and Tab Management

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Navigation Controller and URL Handling

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Bookmarks, History, and Session State

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Download Manager and Background Services

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

RenderProcessHost Lifecycle: Source Code Walkthrough

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Chapter 5: The Renderer Process and Blink Engine

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Renderer Process Bootstrapping

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

HTML Parsing and DOM Construction

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

HTML Parser Source Code Walkthrough

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

CSS Parsing and Style Resolution

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Layout and the Critical Rendering Path

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

LayoutNG Source Code Walkthrough

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Blink Architecture and Module Organization

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Chapter 6: V8 – The JavaScript Virtual Machine

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

V8 Architecture Overview

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

The Ignition Interpreter

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

TurboFan Optimizing Compiler

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Sparkplug Baseline Compiler

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Maglev Mid-Tier Optimizing Compiler

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Garbage Collection Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Hidden Classes and Inline Caches

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

V8 Map (Hidden Class) Source Code Walkthrough

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

WebAssembly Integration

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Chapter 7: GPU Process, Graphics, and Compositing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

GPU Process Architecture

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Command Buffer Protocol: Source Code Walkthrough

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Layer Tree and Compositor

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Rasterization and Paint

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Canvas and WebGL Acceleration

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

ANGLE and Cross-Platform Graphics

Chapter 8: Network Service and Networking Stack

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Network Service Process Architecture

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

DNS Resolution and Prefetching

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

HTTP/2 and Multiplexing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

QUIC Protocol Implementation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Socket Pool and Connection Management

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

QUIC Source Code Walkthrough

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Chapter 9: Storage Subsystem

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Cookie Management and SameSite

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

HTTP Cache Architecture

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

IndexedDB and SQLite Backend

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

LocalStorage and SessionStorage

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Service Worker Storage Model

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Chapter 10: IPC – Inter-Process Communication

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

The IPC Evolution: From NaCl to Mojo

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Mojo Framework Architecture

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Mojo IDL and Generated Bindings: Source Code Walkthrough

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Message Serialization and Channels

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

IPC Performance and Bottlenecks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Cross-Process API Calls

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Chapter 11: Sandboxing and Security Architecture

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

The Threat Model

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Linux Sandboxing: Namespaces and Seccomp

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Seccomp-bpf Policy Compilation: Source Code Walkthrough

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Windows Sandboxing: Job Objects and Integrity Levels

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

macOS Sandboxing: Seatbelt and Sandbox Profiles

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Exploit Mitigations: ASLR, DEP, CFG

Chapter 12: Web Security Policies – Origins, CORS, CSP, Site Isolation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

The Origin Model and Same-Origin Policy

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Cross-Origin Resource Sharing (CORS)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Content Security Policy (CSP)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Site Isolation Implementation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Certificate Validation Pipeline

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Permissions API and Feature Policies

Chapter 13: Privacy Features

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Tracking Protection and Shields

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

The Topics API (Post-FLoC)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Incognito Mode Implementation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Privacy Sandbox Initiatives

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Chapter 14: Extensions Architecture and Developer Tools

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Extension Architecture and Manifest V3

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Background Service Workers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Content Scripts and Isolated Worlds

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Chrome DevTools Protocol

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

DevTools Panel Architecture

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Chapter 15: Performance Optimizations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Resource Loading Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Speculative Parsing and Preload

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Paint Timing and Frame Budget

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Memory Management and Pressure

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Performance Comparison with Peers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Chapter 16: Security Case Studies and Vulnerabilities

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

The Chrome Security Team and Bug Rewards

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Famous Exploit Chains: From Zero-Day to Patch

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Use-After-Free in Blink

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Type Confusion in V8

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Password Encryption and Key Management

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Debugging and Reverse Engineering Methodologies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Sandboxing Escape Case Studies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Chapter 17: Testing, Telemetry, and Infrastructure

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Automated Testing Infrastructure

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Chrome for Testing and Canary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Telemetry and Usage Statistics

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Update Mechanism and Omaha

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Fuchsia and Browser Evolution

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Chapter 18: Future Directions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

WebGPU and the Future of Graphics

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

WebAssembly System Interface (WASI)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

AI and Machine Learning in the Browser

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Architectural Shifts Under Consideration

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

The Browser as Platform

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

Conclusion: The Architecture of a Billion-User System

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.

References

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/googlechromedissected.