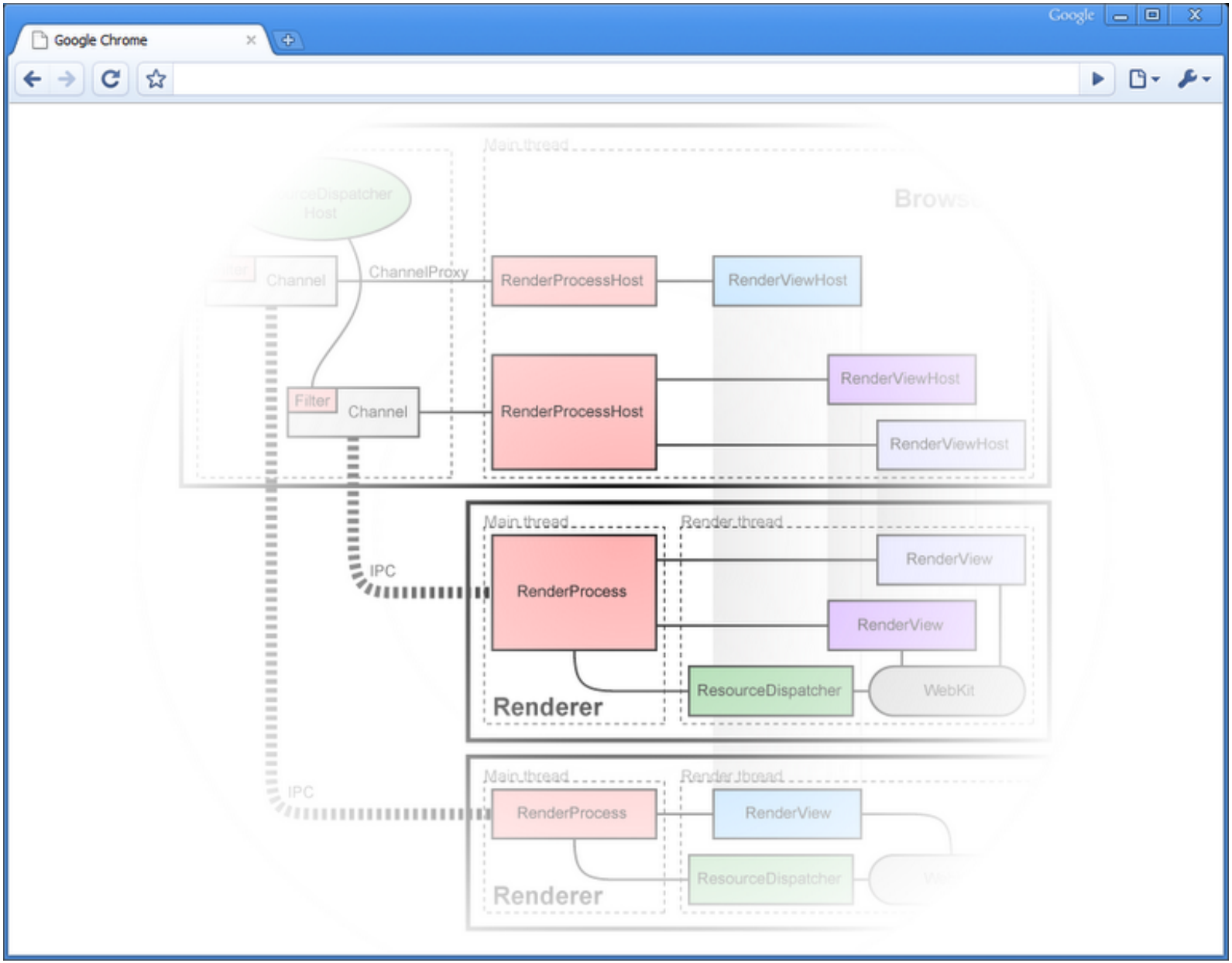
Google™ 09 I/O

Exploring Chrome Internals

Darin Fisher
May 28, 2009

Google 09 IO

Simple interface, powerful core


Google Chrome
Google
Main thread
Channel
ChannelProxy
RenderProcessHost
RenderViewHost
Filter
Channel
RenderProcessHost
RenderViewHost
RenderViewHost
IPC
Main thread
Render thread
RenderProcess
RenderView
RenderView
ResourceDispatcher
WebKit
Renderer
IPC
Main thread
Render thread
RenderProcess
RenderView
ResourceDispatcher
Renderer

“Modern browsers resemble the co-operatively multi-tasked operating systems of the past.”

Guiding sentiment, 2006

Goals

- Speed
- Stability
- Security

Use multiple processes!

- Speed: *Separate threads for separate web apps*
- Stability: *Separate address spaces for separate web apps*
- Security: *Sandbox the web app's process*

Moar speed please

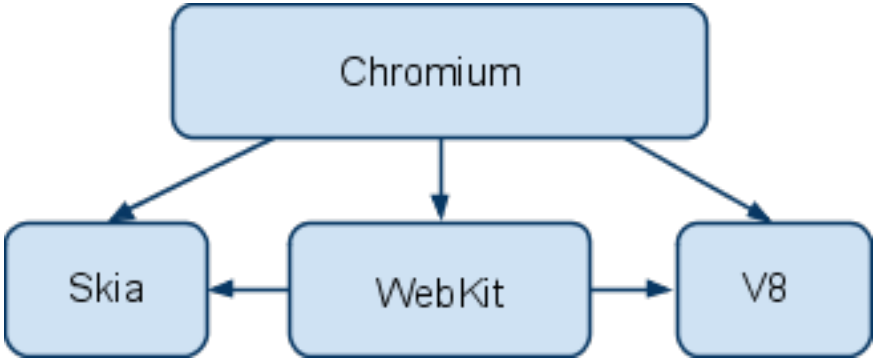
- WebKit
  - Super fast, opensource rendering engine
  - Small footprint (witness: mobile browsers)

- V8
  - Optimized JavaScript engine
  - Opportunity for web apps to do way more

Under the hood...

The major components

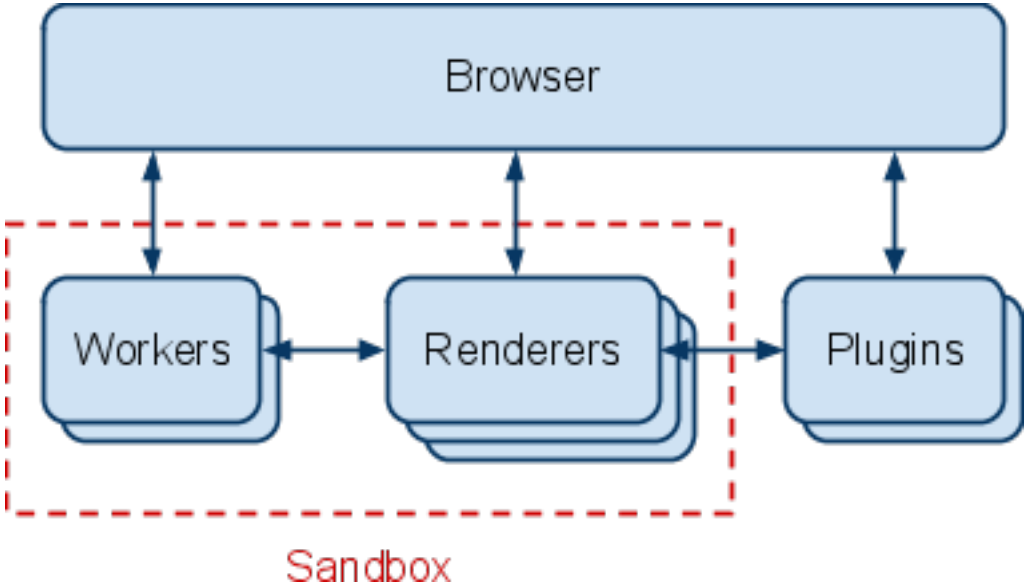
- Chromium
  - UI: tab strip, omnibox, new tab page, ...
  - Multi-process architecture
  - History system
  - Network stack
  - Sandbox
  - *etc...*
- Skia
- WebKit
- V8



Multi-process architecture

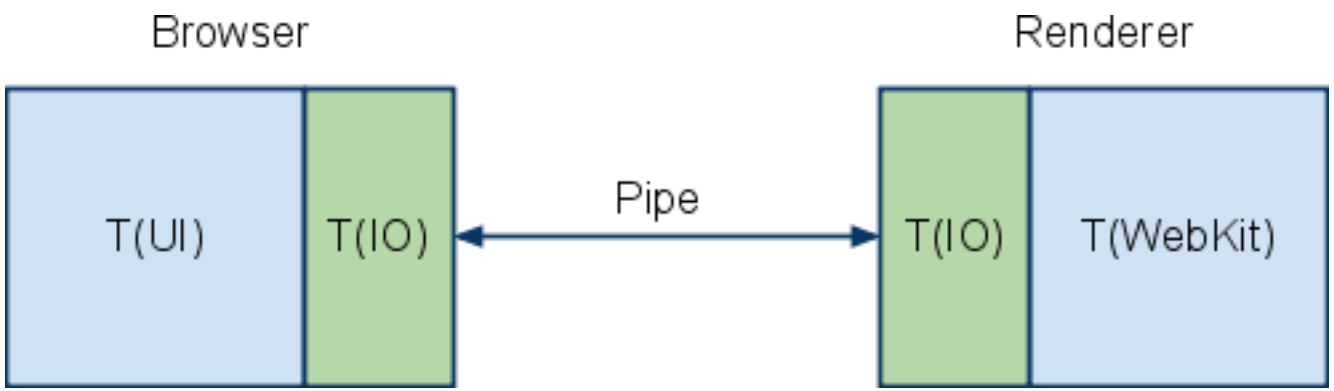
Process Types

- Browser
  - Main coordinator
  - IO proxy
  - Trusted
- Renderer, Worker
  - Embeds WebKit
  - **Untrusted**
- Plugin:
  - Embeds NPAPI (Flash, Java, Silverlight, etc.)
  - Trusted :-(



Inter-process Communication

- Apartment model
  - Primarily async communication over named pipes
  - Limited blocking calls and call nesting
- Some exchange of shared memory
- Each process has a thread dedicated to IPC:



Process assignments

- Approximating process per tab

- Tabs share processes when:
  - They have a (potential) script connection
  - Opened via link click: `<a target=_blank>`
  - The process limit is reached

- New process for Omnibox navigations when domain doesn't match. Tossing the old process -- ultimate GC!

- Process per domain would be nice, but...

The sandbox

- Primary goal:
  - Protect the user's system by blocking malware
- Restrictions:
  - Limit access to the file system and network
  - Limit access to the windowing system
  - Limit access to input devices
- Mechanism:
  - Strip the user's token
  - Use a job object to further restrict
  - Run on a separate desktop

The sandbox

- But, but... a browser needs to access the file system!
  - Supporting file uploads
  - Supporting file:// URLs

- What isn't protected?
  - Cookies
  - Passwords
  - HTML5 database, local/session storage
  - Cross-site attacks (user data in the cloud)

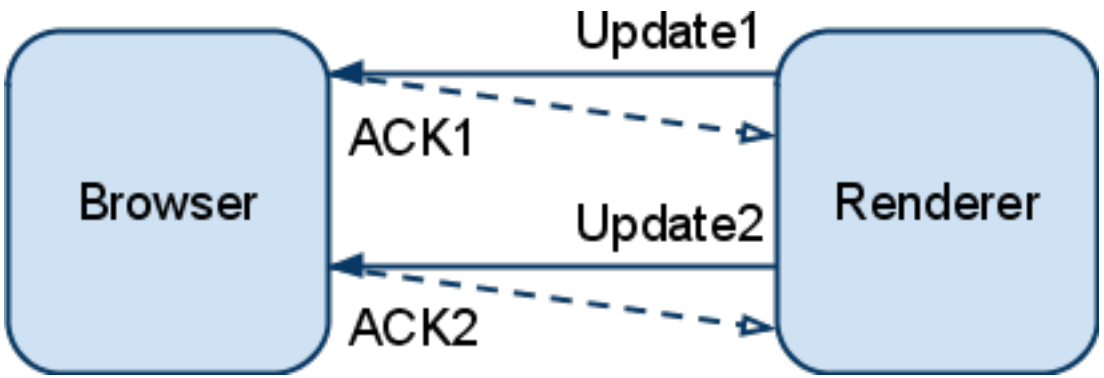
Rendering in a sandbox

- Short version:
  - Render to a bitmap
  - Send bitmap to the browser process
  - Browser copies the bitmap to the screen
- Complexities:
  - Limited access to OS APIs (fonts, etc.)
  - A hung renderer should not lock up the browser
  - Needs to be fast!

Painting and scrolling

- Lock free:
  - Browser maintains a backingstore
  - Renderer sends updates to the backingstore via SHM
  - Browser paints to the screen from the backingstore
  - Browser ACKs renderer to allow another update

- Scrolling is similar (includes a scroll delta)



Resource loading

- Browser serves as proxy for all IO
  - Restricts access to file:// and chrome://
  - Performs safe-browsing checks
  - Vends cookies

- Before WebKit sees any data, the browser...
  - Follows HTTP redirects
  - Handles HTTP auth
  - Detects MIME type (handles downloads browser-side)
  - Performs security checks for SSL

History system

- Lock free visited links system
  - Shared memory containing bitmap
  - Indexed by hash(URL)
  - Only the browser process can write
  - Grow map size by creating a copy

- After a page loads,
  - Text is extracted and fed into the FTS index (sqlite)
  - Thumbnail is generated and stored

Plugins

- Supports:
  - Netscape style plugins
  - Whitelist of ActiveX controls (only WMP now)

- One process per plugin type
  - Mimics the environment of a single-proc browser
  - Some plugins take a while to load :-/

- Challenge: NPAPI is a synchronous API
  - Cache rendering of windowless plugins
  - *Jump through hoops* for windowed plugins
  - Porting!

WebKit

WebKit overview

- Comprised of several modules:
  - JavaScriptCore: JS engine **(not used)**
  - WebCore: HTML+CSS rendering, DOM, etc.
  - WebKit: embedding API layer **(not used)**

- WebCore conditionals:
  - PLATFORM(CHROMIUM) ⇨ platform/chromium
  - PLATFORM(SKIA) ⇨ platform/graphics/skia
  - USE(V8) ⇨ bindings/v8

- WebKit versions:
  - Chrome 1 ~ Safari 3
  - Chrome 2 ~ Safari 4

WebKit development

- The Chromium devs on #webkit
  - 3 reviewers
  - Over a dozen contributors and counting
- Status: **Unforked!!**
- Focus going forward:
  - WebKit API for Chromium
  - Open web platform (HTML5, etc.)
  - Web compatibility improvements
  - Performance

Open web platform

- In progress:
  - Audio/video
  - Application caches
  - Database
  - Local storage
  - Session storage
  - Notifications
  - Web workers: dedicated, persistent, shared

- Multi-process arch and sandbox pose challenges

Network stack

Making a better wheel

- From Wininet to Winhttp to src/net/http/
- DNS prefetching
- In development:
  - Feature parity (client certs, socks, IPv6 literals, etc.)
  - Sparse caching
  - Pseudo-pipelining
  - Deferred connection binding
  - Parallel proxy auto config

Google™ 09 I/O