Inside look at modern web browser (part 1)

Published on Wednesday, September 5, 2018 • Updated on Friday, September 21, 2018

Mariko Kosaka
Mariko is a drawsplainer

Table of contents

CPU, GPU, Memory, and multi-process architecture

In this 4-part blog series, we'll look inside the Chrome browser from high-level architecture to the specifics of the rendering pipeline. If you ever wondered how the browser turns your code into a functional website, or you are unsure why a specific technique is suggested for performance improvements, this series is for you.

As part 1 of this series, we'll take a look at core computing terminology and Chrome's multi-process architecture.

> If you are familiar with the idea of CPU/GPU and process/thread you may skip to Browser Architecture.

At the core of the computer are the CPU and GPU

In order to understand the environment that the browser is running, we need to understand a few computer parts and what they do.

CPU

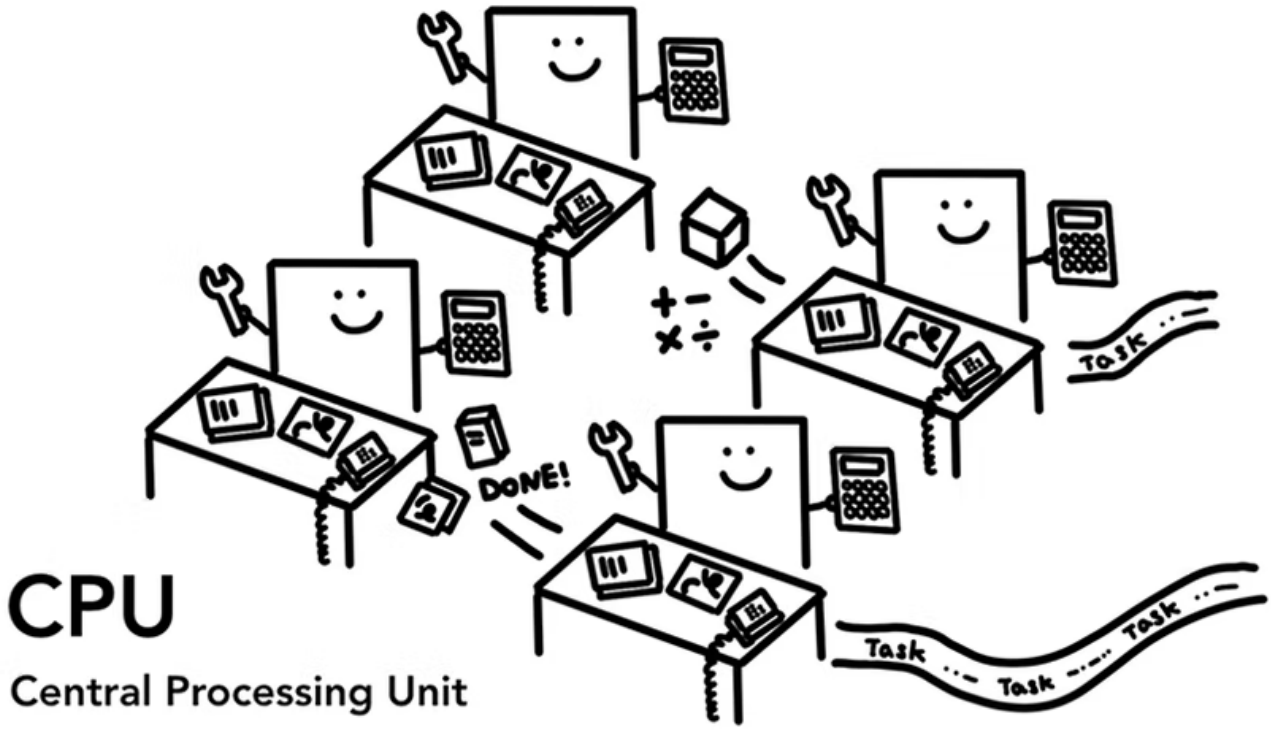

Figure 1: 4 CPU cores as office workers sitting at each desk handling tasks as they come in

First is the **C**entral **P**rocessing **U**nit - or **CPU**. The CPU can be considered your computer's brain. A CPU core, pictured here as an office worker, can handle many different tasks one by one as they come in. It can handle everything from math to art while knowing how to reply to a customer call. In the past most CPUs were a single chip. A core is like another CPU living in the same chip. In modern hardware, you often get more than one core, giving more computing power to your phones and laptops.

GPU

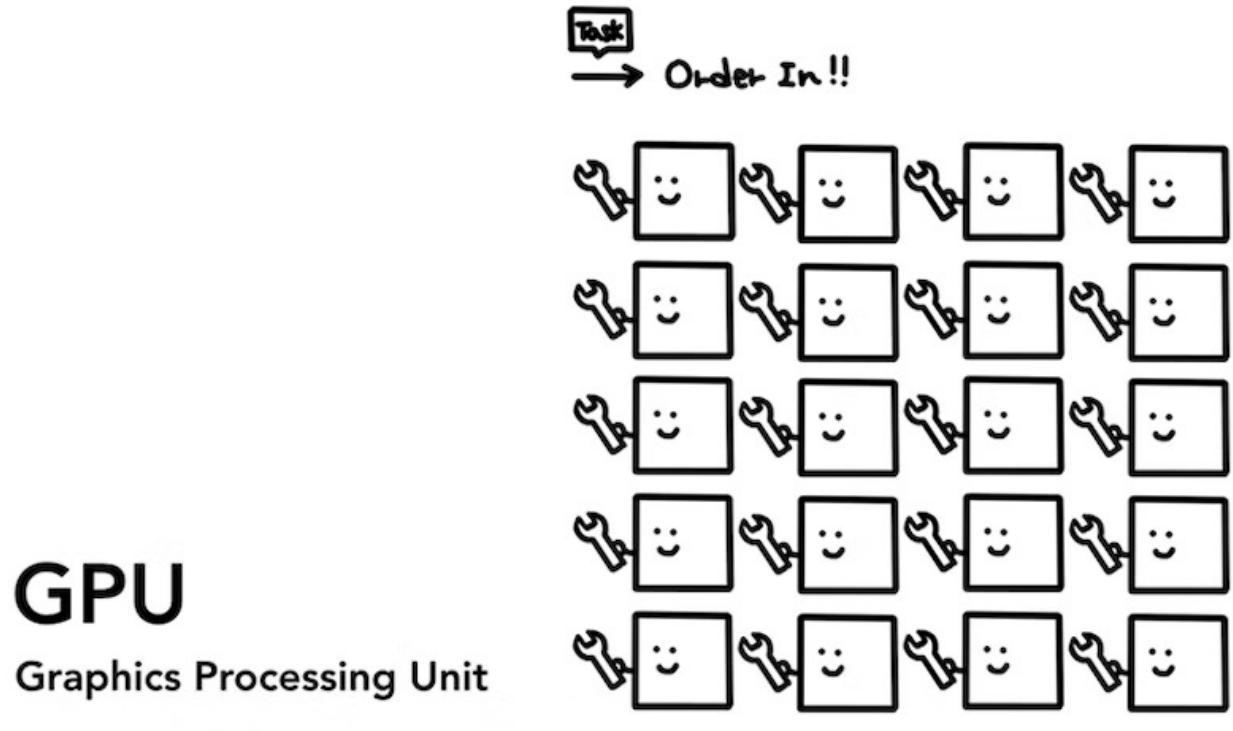

Figure 2: Many GPU cores with wrench suggesting they handle a limited task

Graphics **P**rocessing **U**nit - or **GPU** is another part of the computer. Unlike CPU, GPU is good at handling simple tasks but across multiple cores at the same time. As the name suggests, it was first developed to handle graphics. This is why in the context of graphics "using GPU" or "GPU-backed" is associated with fast rendering and smooth interaction. In recent years, with GPU-accelerated computing, more and more computation is becoming possible on GPU alone.

When you start an application on your computer or phone, the CPU and GPU are the ones powering the application. Usually, applications run on the CPU and GPU using mechanisms provided by the Operating System.

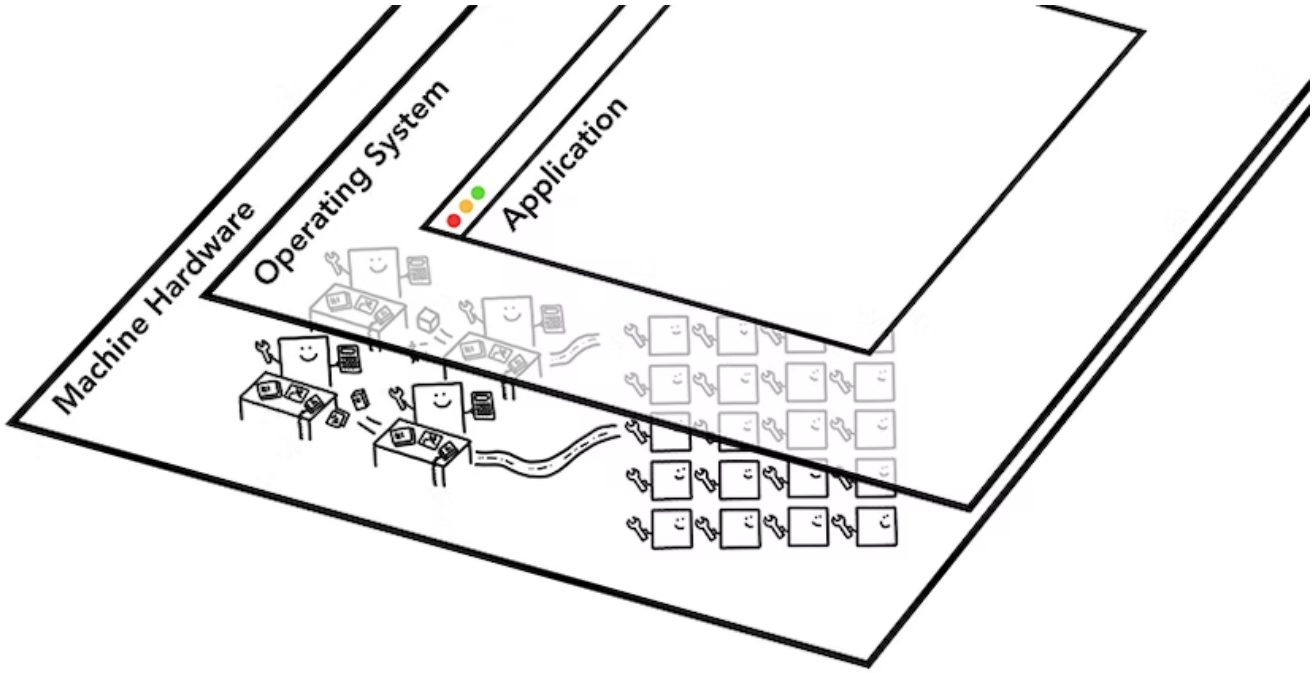

Figure 3: Three layers of computer architecture. Machine Hardware at the bottom, Operating System in the middle, and Application on top.

Executing program on Process and Thread

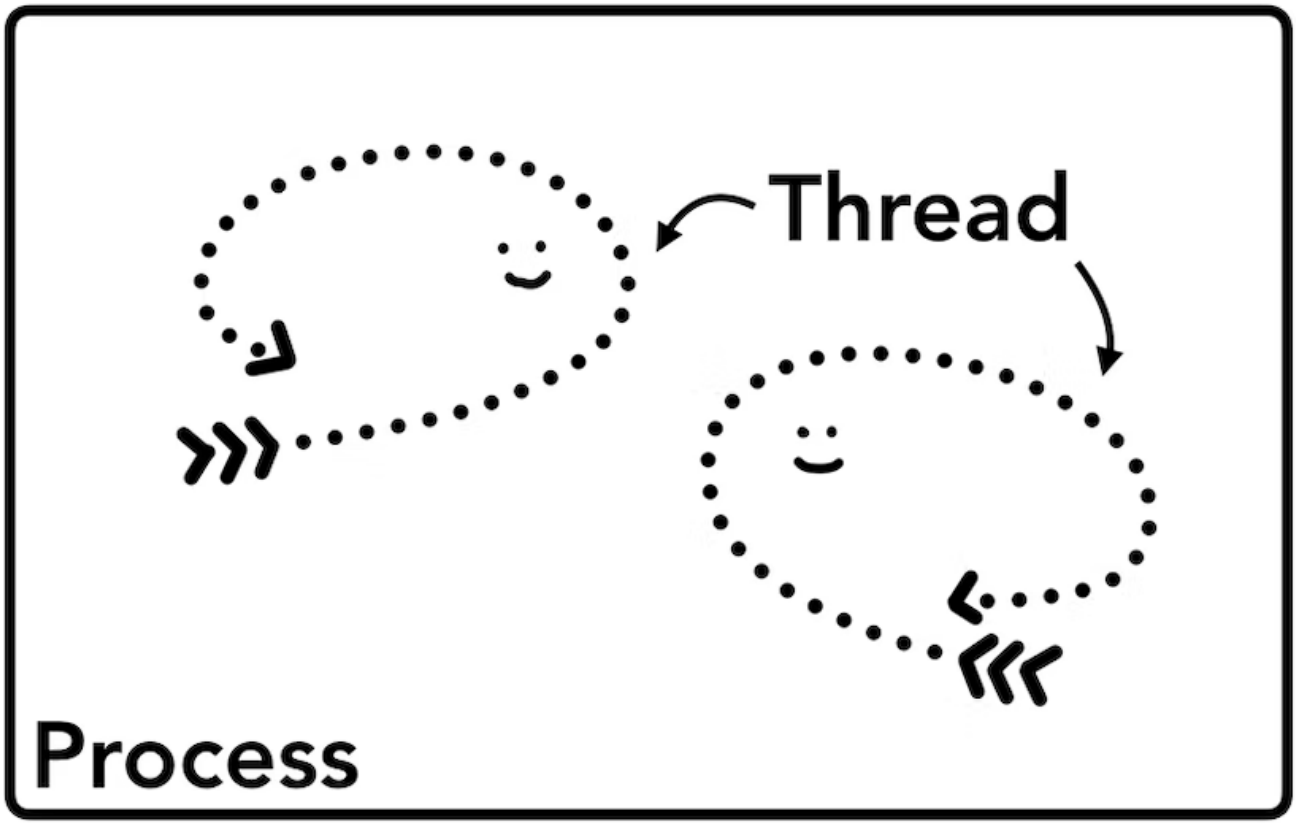

Figure 4: Process as a bounding box, threads as abstract fish swimming inside of a process

Another concept to grasp before diving into browser architecture is Process and Thread. A process can be described as an application's executing program. A thread is the one that lives inside of process and executes any part of its process's program.

When you start an application, a process is created. The program might create thread(s) to help it do work, but that's optional. The Operating System gives the process a "slab" of memory to work with and all application state is kept in that private memory space. When you close the application, the process also goes away and the Operating System frees up the memory.

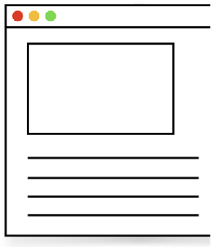

Figure 5: Diagram of a process using memory space and storing application data

A process can ask the Operating System to spin up another process to run different tasks. When this happens, different parts of the memory are allocated for the new process. If two processes need to talk, they can do so by using **I**nter **P**rocess **C**ommunication (**IPC**). Many applications are designed to work this way so that if a worker process get unresponsive, it can be restarted without stopping other processes which are running different parts of the application.

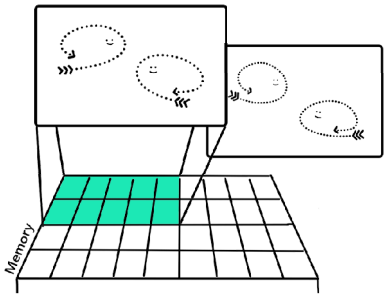

Figure 6: Diagram of separate processes communicating over IPC

Browser Architecture

So how is a web browser built using processes and threads? Well, it could be one process with many different threads or many different processes with a few threads communicating over IPC.

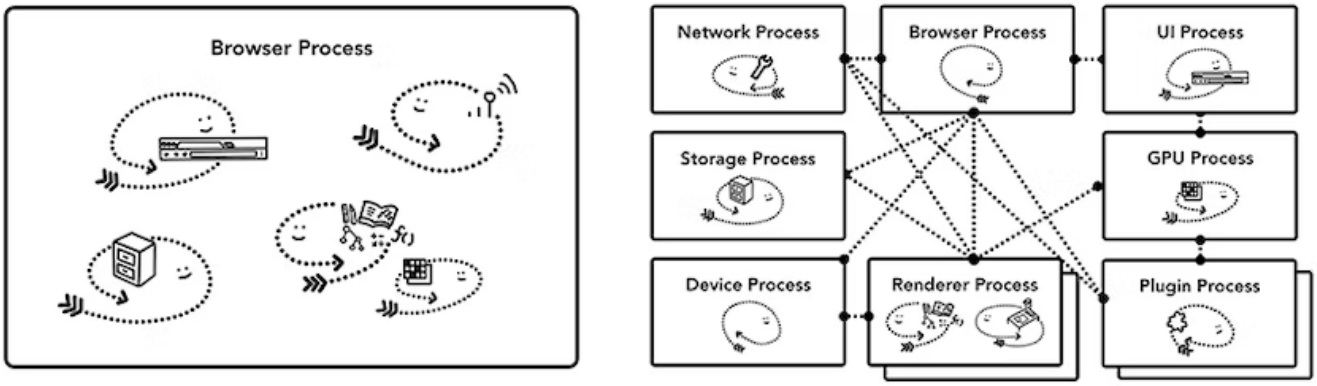

Figure 7: Different browser architectures in process/thread diagram

The important thing to note here is that these different architectures are implementation details. There is no standard specification on how one might build a web browser. One browser's approach may be completely different from another.

For the sake of this blog series, we are going to use Chrome's recent architecture described in the diagram below.

At the top is the browser process coordinating with other processes that take care of different parts of the application. For the renderer process, multiple processes are created and assigned to each tab. Until very recently, Chrome gave each tab a process when it could; now it tries to give each site its own process, including iframes (see Site Isolation).

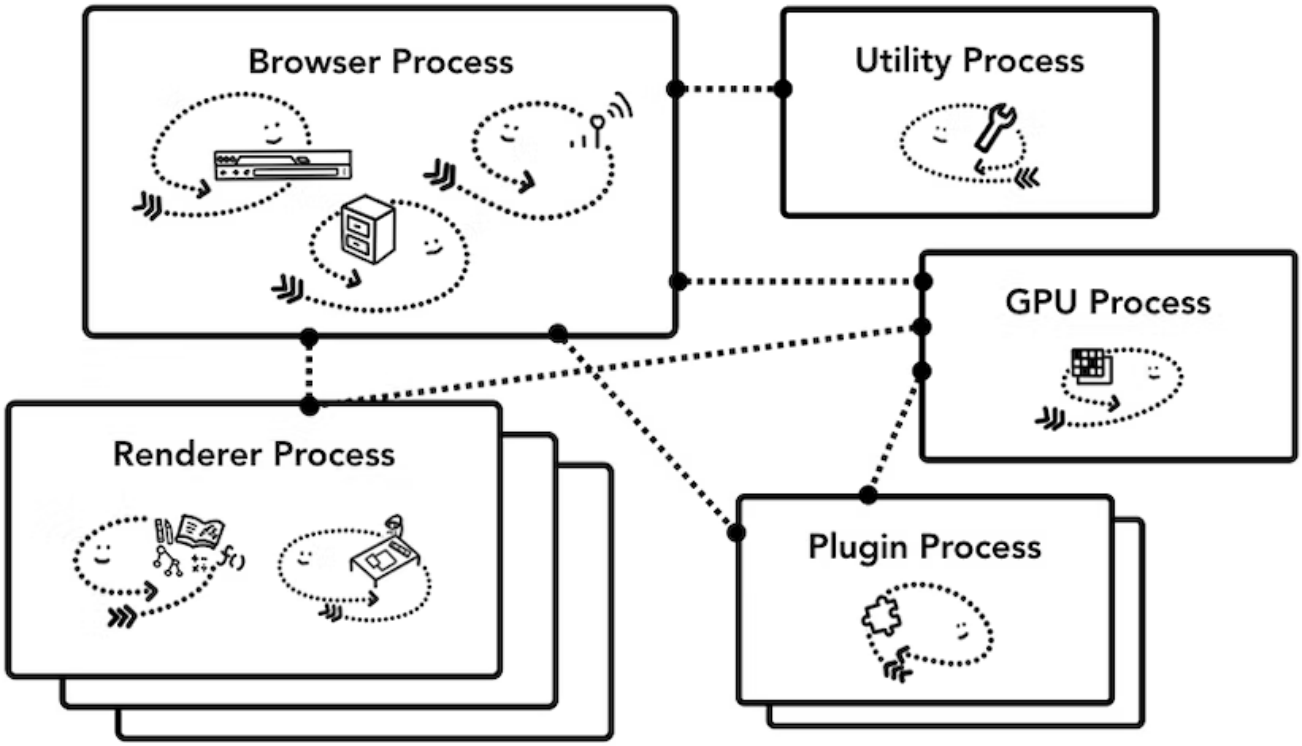

Figure 8: Diagram of Chrome's multi-process architecture. Multiple layers are shown under Renderer Process to represent Chrome running multiple Renderer Processes for each tab.

Which process controls what?

The following table describes each Chrome process and what it controls:

Process and What it controls	
Browser	Controls "chrome" part of the application including address bar, bookmarks, back and forward buttons. Also handles the invisible, privileged parts of a web browser such as network requests and file access.
Renderer	Controls anything inside of the tab where a website is displayed.
Plugin	Controls any plugins used by the website, for example, flash.
GPU	Handles GPU tasks in isolation from other processes. It is separated into different process because GPUs handles requests from multiple apps and

draw them in the same surface.

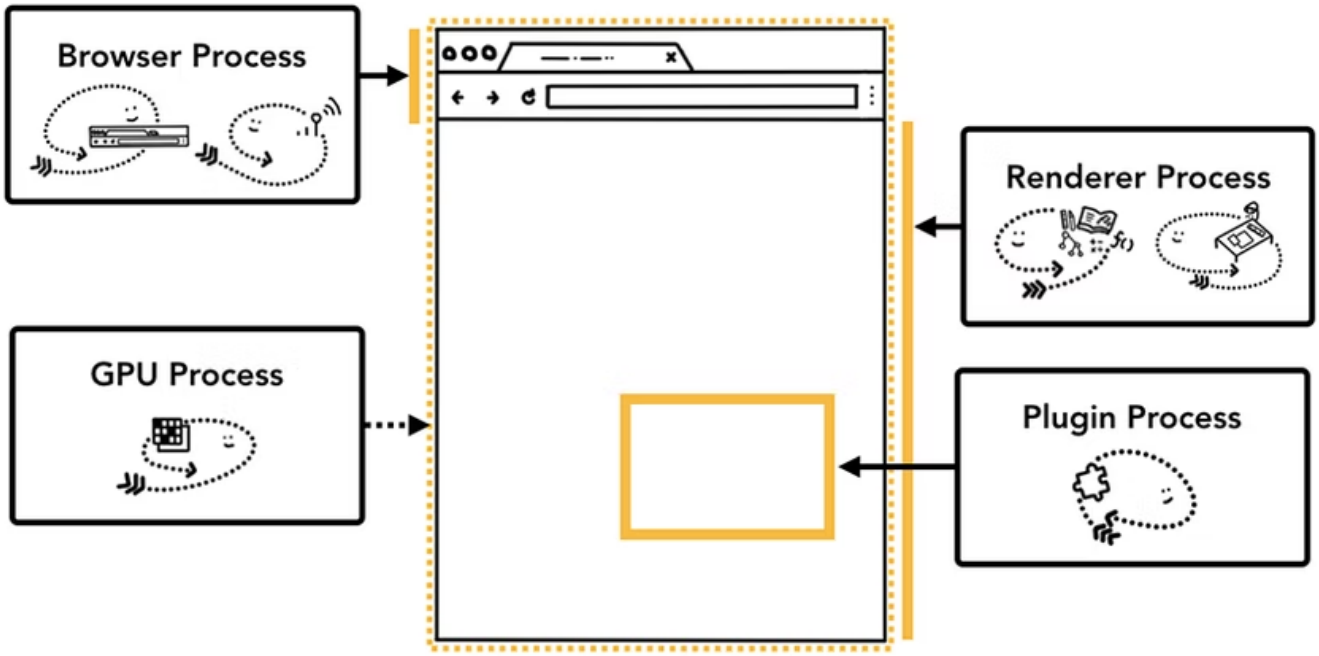

Figure 9: Different processes pointing to different parts of browser UI

There are even more processes like the Extension process and utility processes. If you want to see how many processes are running in your Chrome, click the options menu icon more_vert at the top right corner, select More Tools, then select Task Manager. This opens up a window with a list of processes that are currently running and how much CPU/Memory they are using.

The benefit of multi-process architecture in Chrome

Earlier, I mentioned Chrome uses multiple renderer process. In the most simple case, you can imagine each tab has its own renderer process. Let's say you have 3 tabs open and each tab is run by an independent renderer process. If one tab becomes unresponsive, then you can close the unresponsive tab and move on while keeping other tabs alive. If all tabs are running on one process, when one tab becomes unresponsive, all the tabs are unresponsive. That's sad.

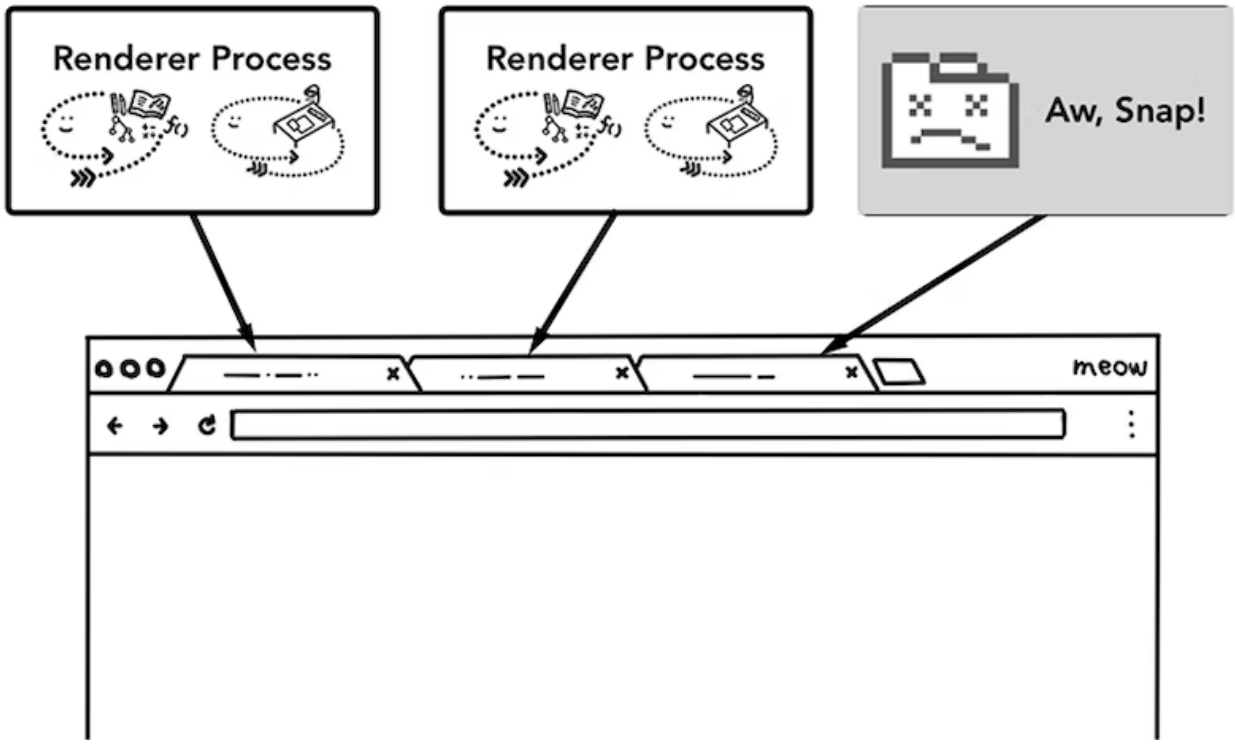

Figure 10: Diagram showing multiple processes running each tab

Another benefit of separating the browser's work into multiple processes is security and sandboxing. Since operating systems provide a way to restrict processes' privileges, the browser can sandbox certain processes from certain features. For example, the Chrome browser restricts arbitrary file access for processes that handle arbitrary user input like the renderer process.

Because processes have their own private memory space, they often contain copies of common infrastructure (like V8 which is a Chrome's JavaScript engine). This means more memory usage as they can't be shared the way they would be if they were threads inside the same process. In order to save memory, Chrome puts a limit on how many processes it can spin up. The limit varies depending on how much memory and CPU power your device has, but when Chrome hits the limit, it starts to run multiple tabs from the same site in one process.

Saving more memory - Servicification in Chrome

The same approach is applied to the browser process. Chrome is undergoing architecture changes to run each part of the browser program as a service allowing to easily split into different processes or aggregate into one.

General idea is that when Chrome is running on powerful hardware, it may split each service into different processes giving more stability, but if it is on a resource-constraint device, Chrome consolidates services into one process saving memory footprint. Similar approach of consolidating processes for less memory usage have been used on platform like Android before this change.

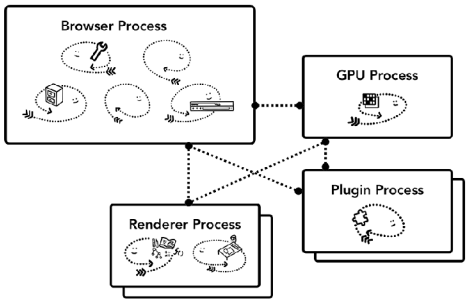

Figure 11: Diagram of Chrome's servicification moving different services into multiple processes and a single browser process

Per-frame renderer processes - Site Isolation

Site Isolation is a recently introduced feature in Chrome that runs a separate renderer process for each cross-site iframe. We've been talking about one renderer process per tab model which allowed cross-site iframes to run in a single renderer process with sharing memory space between different sites. Running a.com and b.com in the same renderer process might seem okay. The Same Origin Policy is the core security model of the web; it makes sure one site cannot access data from other sites without consent. Bypassing this policy is a primary goal of security attacks. Process isolation is the most effective way to separate sites. With Meltdown and Spectre, it became even more apparent that we need to separate sites using processes. With Site Isolation enabled on desktop by default since Chrome 67, each cross-site iframe in a tab gets a separate renderer process.

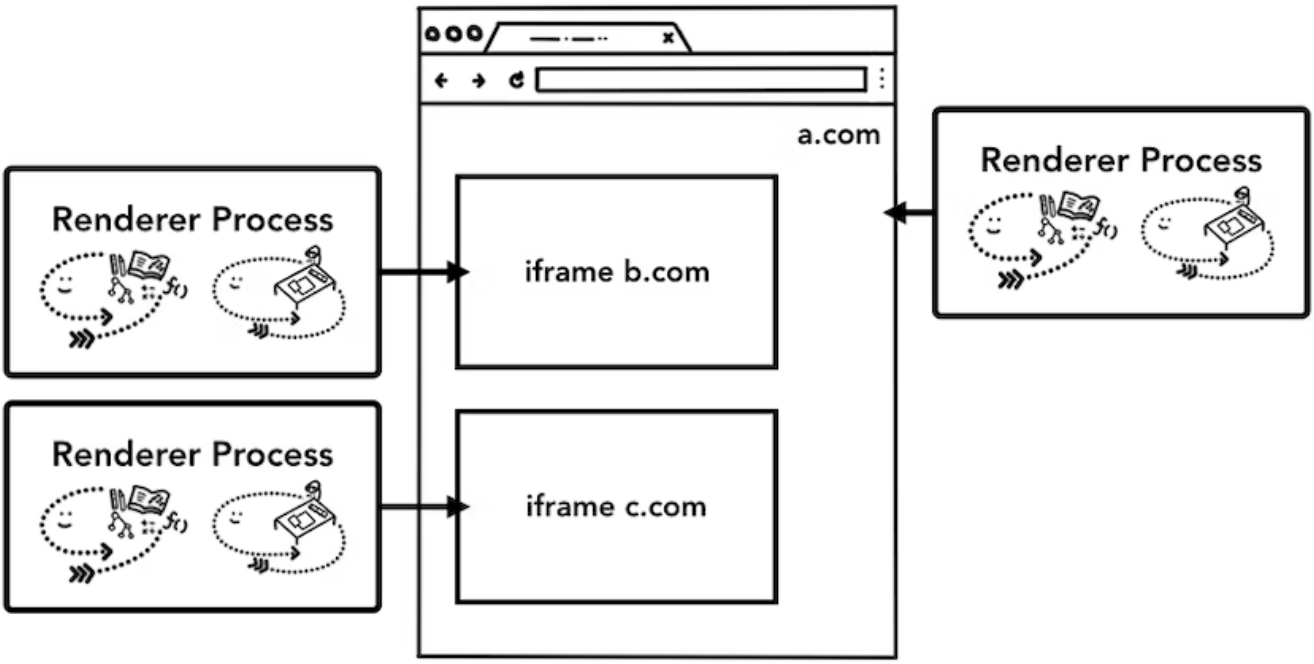

Figure 12: Diagram of site isolation; multiple renderer processes pointing to iframes within a site

Enabling Site Isolation has been a multi-year engineering effort. Site Isolation isn't as simple as assigning different renderer processes; it fundamentally changes the way iframes talk to each other. Opening devtools on a page with iframes running on different processes means devtools had to implement behind-the-scenes work to make it appear seamless. Even running a simple Ctrl+F to find a word in a page means searching across different renderer processes. You can see the reason why browser engineers talk about the release of Site Isolation as a major milestone!

Wrap-up

In this post, we've covered a high-level view of browser architecture and covered the benefits of a multi-process architecture. We also covered Servicification and Site Isolation in Chrome that is deeply related to multi-process architecture. In the next post, we'll start diving into what happens between these processes and threads in order to display a website.

Did you enjoy the post? If you have any questions or suggestions for the future post, I'd love to hear from you in the comment section below or @kosamari on Twitter.

Next: What happens in navigation

Last updated: Friday, September 21, 2018 Improve article

Follow us

Contribute

File a bug

View source

Related content

web.dev

Case studies

Podcasts

Connect

Twitter

YouTube

GitHub

Google Developers

Chrome Firebase All products Privacy Terms

Choose language

ENGLISH (en)