# Browser Internals

Kameswari Chebrolu

Department of CSE, IIT Bombay

https://programmerhumor.io/programming-memes/the-best-browser-at/

# Background: Overall Outline

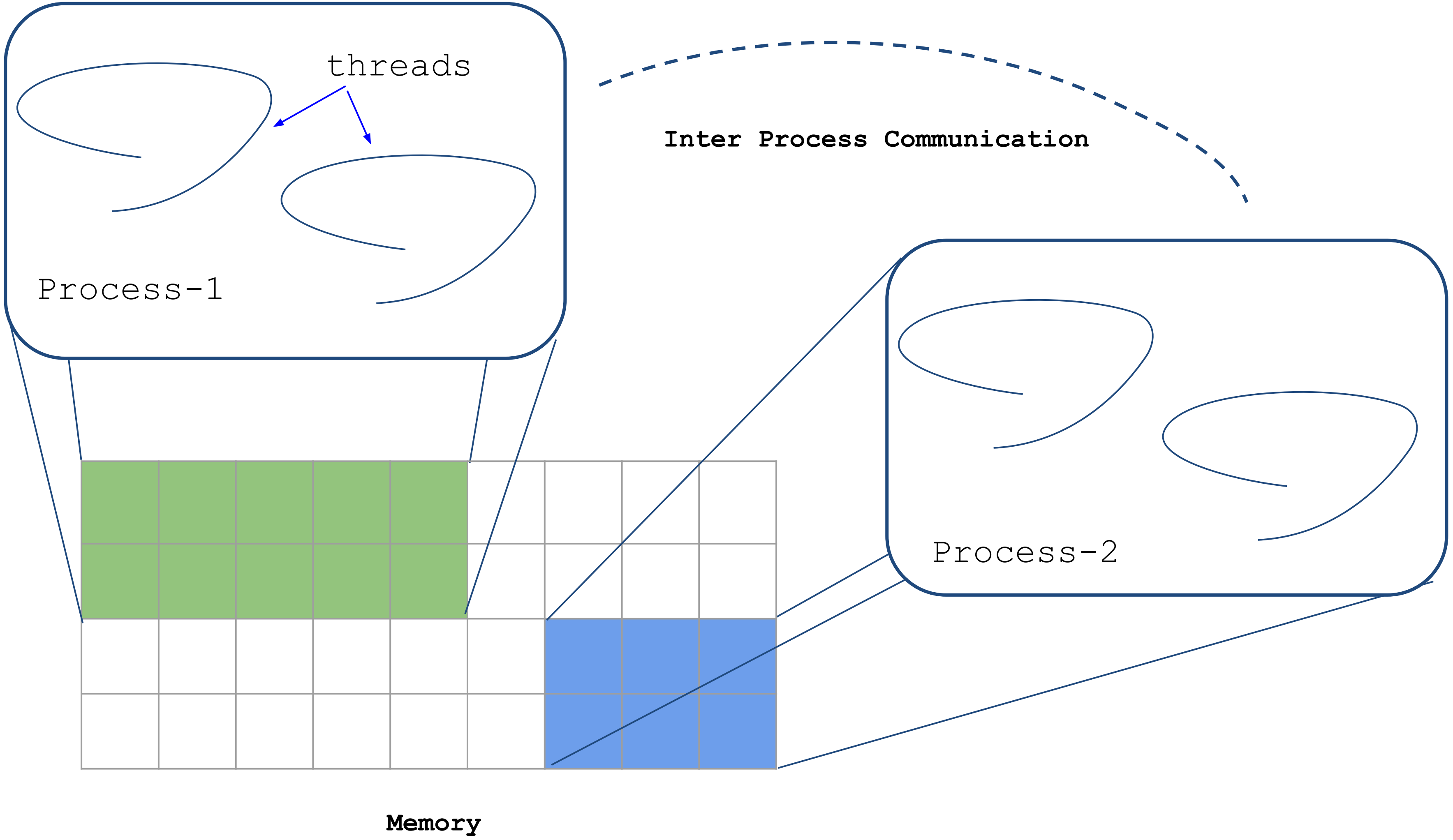
- ~~What constitutes a webpage?~~
- **What goes on inside a Browser?**
  - **What standard security mechanisms implemented?**
- How do client and server communicate?
  - HTTP/HTTPs protocol
  - Session Management via cookies and tokens
- How does a web server process requests and generate responses?
  - Static vs Dynamic content

## Browser

- A software application that helps users to “browse” the Web
  - Provide a rich user interface
    - address bar, navigation buttons, bookmarks etc
  - Support extensions or add-ons to enhance functionality
- Popular web browsers: Google Chrome, Mozilla Firefox, Microsoft Edge, Safari, and Opera
- Lot happens inside the Browser!

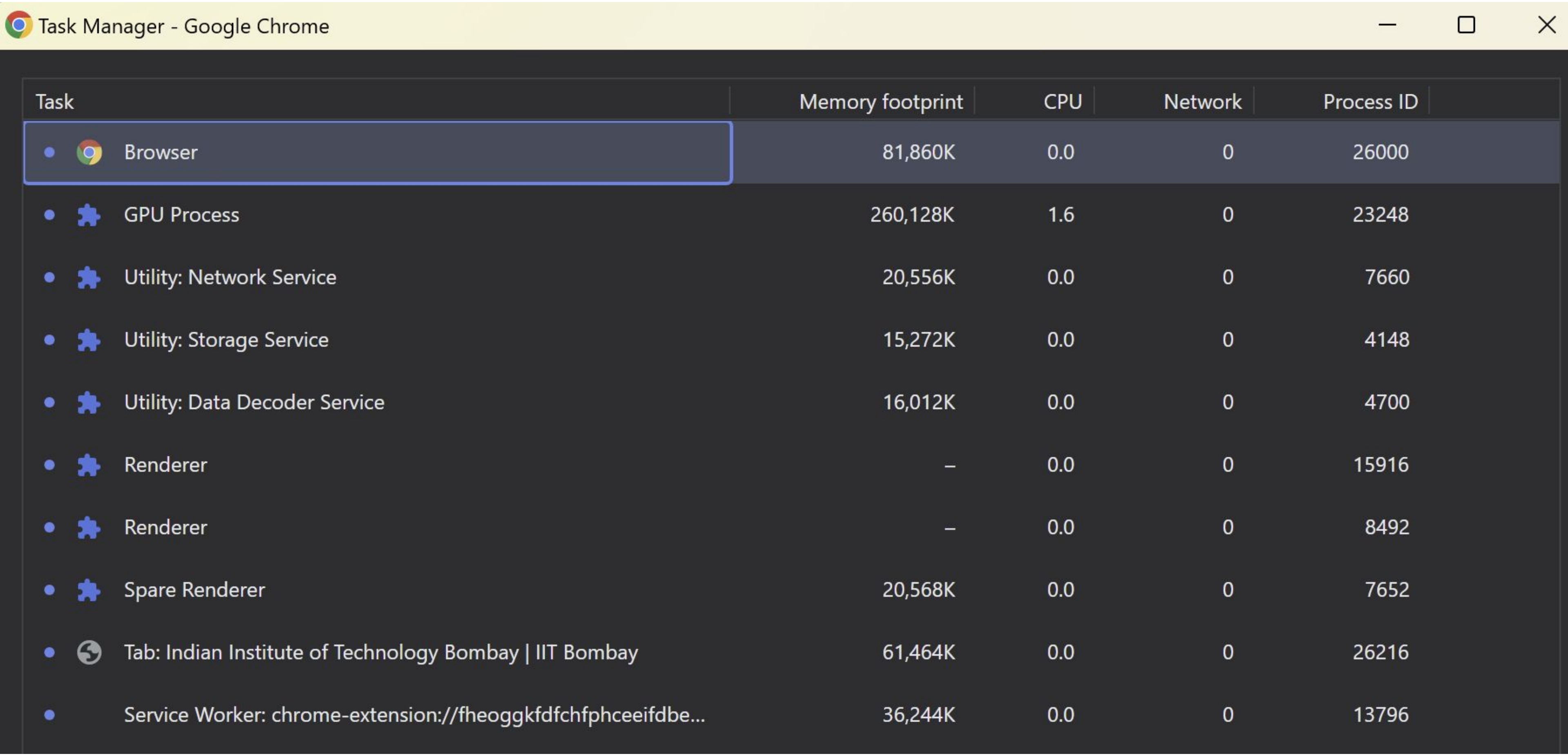
## Background: Processes and Threads

- Process: a running program
- Thread: lives inside a process and executes part of the program
- When a process starts, OS gives it a chunk of memory
  - Close application, processes stopped and memory freed by OS
- Process can fork other processes
  - These worker processes run different tasks of an application
    - They talk via Inter Process Communication (IPC)
    - Each process has its own "disjoint" chunk of memory
  - One worker unresponsive, can be restarted independent of others $\longrightarrow$ better fault tolerance

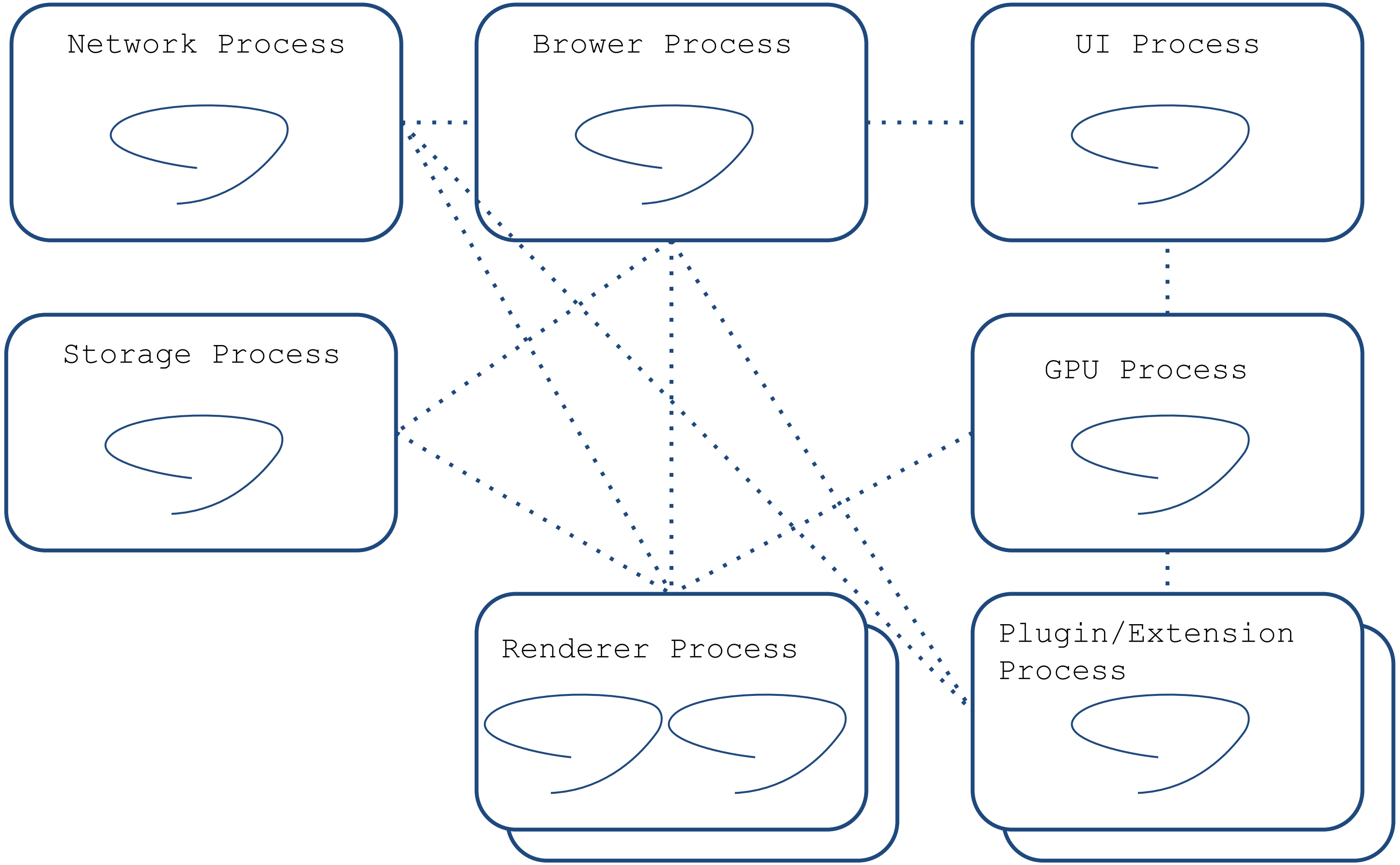

threads
Process-1
Inter Process Communication
Process-2
Memory

## Case Study: Chrome browser

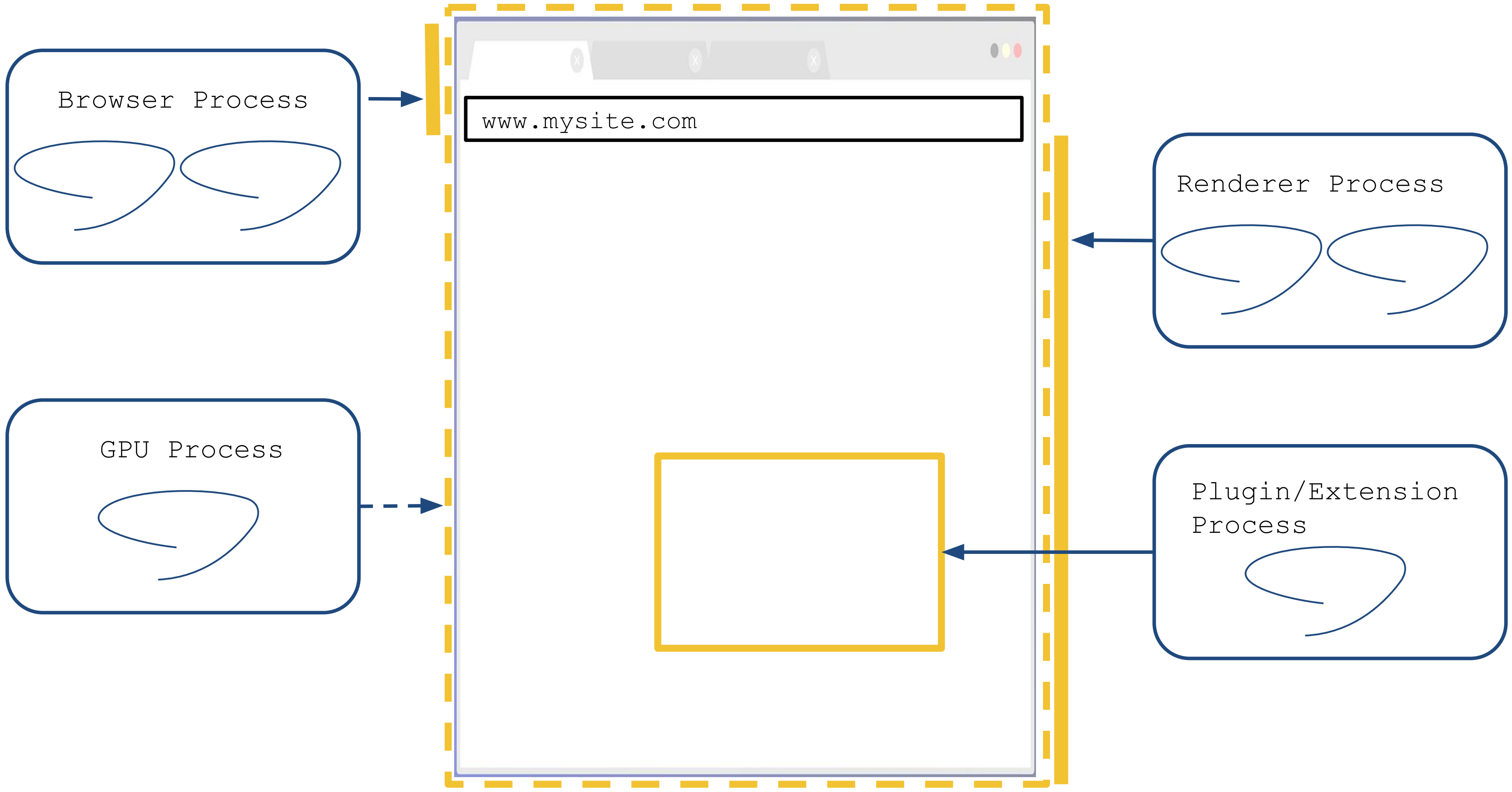
- Start a browser (application)
  - One process with many different threads or many different processes with a few threads?
  - Latter more common
- How many processes are running?
  - Open Chrome, click three dots $\longrightarrow$More Tools (top right corner)
  - Select Task Manager
  - Shows a list of processes currently running and their CPU/Memory usage


Task Manager - Google Chrome

| Task | Memory footprint | CPU | Network | Process ID |
|---|---|---|---|---|
| Browser | 81,860K | 0.0 | 0 | 26000 |
| GPU Process | 260,128K | 1.6 | 0 | 23248 |
| Utility: Network Service | 20,556K | 0.0 | 0 | 7660 |
| Utility: Storage Service | 15,272K | 0.0 | 0 | 4148 |
| Utility: Data Decoder Service | 16,012K | 0.0 | 0 | 4700 |
| Renderer | – | 0.0 | 0 | 15916 |
| Renderer | – | 0.0 | 0 | 8492 |
| Spare Renderer | 20,568K | 0.0 | 0 | 7652 |
| Tab: Indian Institute of Technology Bombay \| IIT Bombay | 61,464K | 0.0 | 0 | 26216 |
| Service Worker: chrome-extension://fheoggkfdfchfphceeifdbe... | 36,244K | 0.0 | 0 | 13796 |


Network Process
Brower Process
UI Process
Storage Process
GPU Process
Renderer Process
Plugin/Extension Process

| Process | Role |
|---|---|
| Browser | ● Controls address bar, bookmarks, back and forward buttons etc<br>● Underhood, handles privileged stuff such as network requests and file access as threads |
| Renderer | Controls what happens inside a tab (one tab per website) |
| Plugin/Extensions | Controls any plugins/extensions (one process per plugin or extension) |
| GPU | ● Handles GPU tasks in isolation from other processes<br>● Responsible for rendering and displaying graphics, images, and multimedia content<br>● Also handles compositing (final image that you see on your screen); ensure smoothness and responsiveness of web page interaction |
| Storage | Manages different types of storage (cookies, local storage, indexed dB etc) are managed efficiently, securely, and isolated from each other where appropriate |

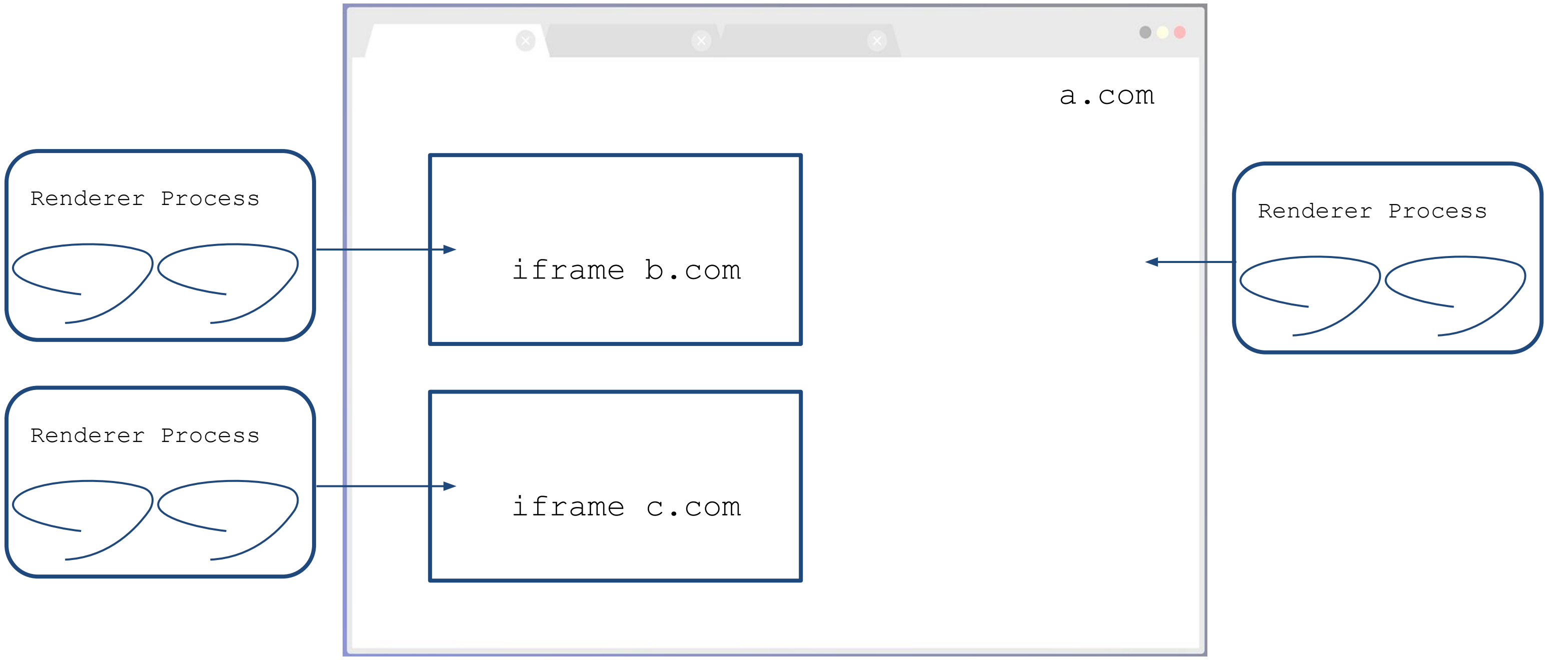

Browser Process
GPU Process
www.mysite.com
Renderer Process
Plugin/Extension Process

## A few points to note

- Browser Process:
  - Running on powerful hardware ⟶ Splits into different processes giving more stability (saw earlier under Chrome task manager)
  - Less powerful hardware ⟶ Runs as single process saving memory
- Renderer Process:
  - One process per tab. Why?
    - If one tab unresponsive, other tabs still active
    - More important reason: Security
      - OS protects one process from accessing another process memory

- iframe (inline frame) allows you to embed content from another website within your webpage
  - <iframe src="https://www.example.com" width="600" height="400" frameborder="0"></iframe>
  - src attribute specifies the URL of the external site to be displayed in the iframe

- Each cross-site iframe in the same tab gets a separate renderer process (security reasons, SOP policy)
  - Not easy to implement
  - E.g. Ctrl+F to find a word in a page ⟶ searching across different renderer processes



## Security Mechanisms

- Browsers implement many mechanisms to protect users from various threats
  - Many will be covered over time as we cover attacks and corresponding defenses!
- Some basic mechanisms important to know:
  - Sandboxing
  - Same Origin Policy
  - HTTPS (SSL/TLS) – covered under protocols

## SandBoxing

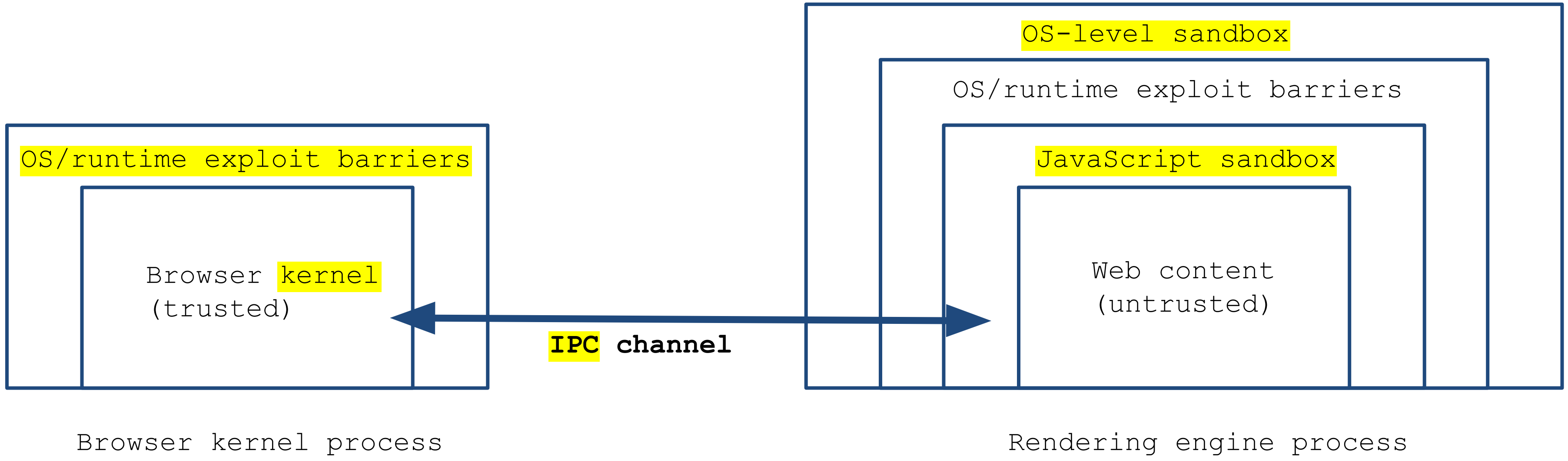
- What is a sandbox?
  - A safe and isolated environment for executing untrusted programs
    - Prevents a bad process from compromising the system
  - Achieved by restricting a process from accessing system resources
    - E.g. limit access to files outside of designated directory
- In Linux: chroot, Linux namespaces, seccomp etc establish a sandbox environment

## Browser: Layers of defense

- Every large piece of software (including browser) contains bugs
- As users browse, websites can be malicious
- How to protect  users?

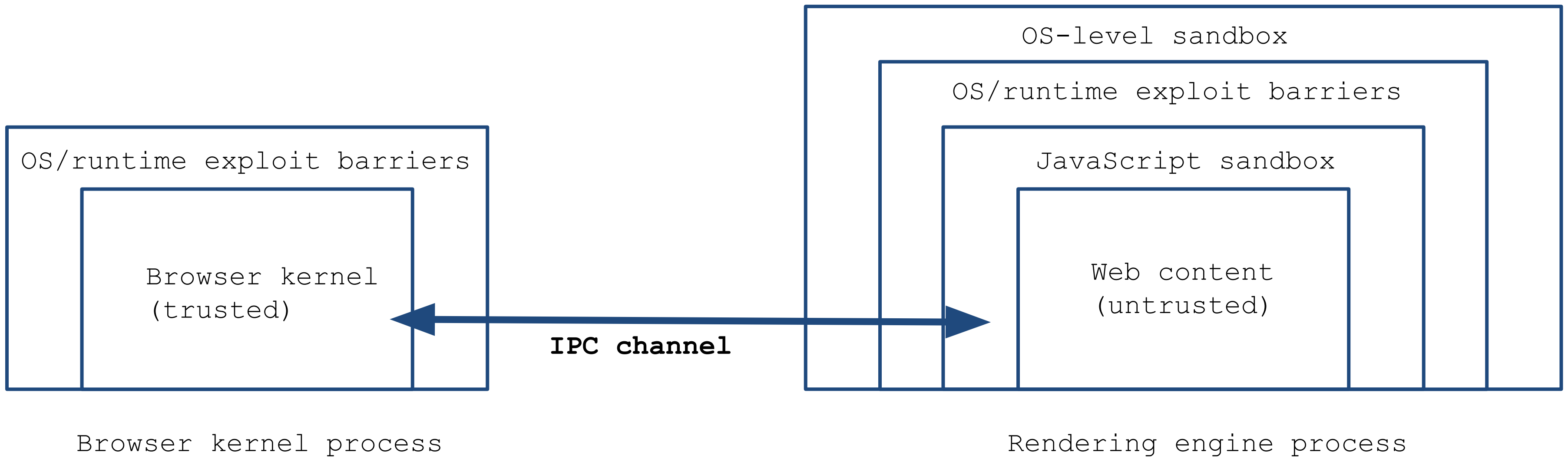
- Already saw process isolation (OS manages)!
  - Browser, Renderer (per tab), GPU etc are all different processes
- Browser process runs with high-privilege
  - Acts with user’s authority
  - Responsible for UI, storing cookies (of various websites), history and network access

# Layer of Defense around Google Chrome's Rendering Engine



- Rendering process runs at low privilege in a sandbox environment
  - JavaScript engine runs javascript in a sandbox
    - Prevents it from accessing sensitive resources on user's device
  - Each render process also runs in an OS-level sandbox
    - Prevents rendering process from interacting with other processes, file system etc
      - Can only exchange messages with browser process via IPC
      - Malicious code could still send messages to browser process via IPC but interface is simple and restricted to do damage!

# Layer of Defense around Google Chrome's Rendering Engine



- Several OS/Runtime level protection also!
  - DEP (data execution prevention)
    - Marks memory pages (e.g. stack, heap etc) as NX (non executable)
  - ASLR (address space layout randomization)
    - Prevents attackers from guessing memory address locations
    - Can upload malicious code and jump to those locations
  - SafeSEH (safe exception handlers)
  - Stack overrun detection (GS) via canary
    - Prevents overwriting of return address
    - Prevents buffer overflow type attacks

All  of above help prevent attackers from running malicious code

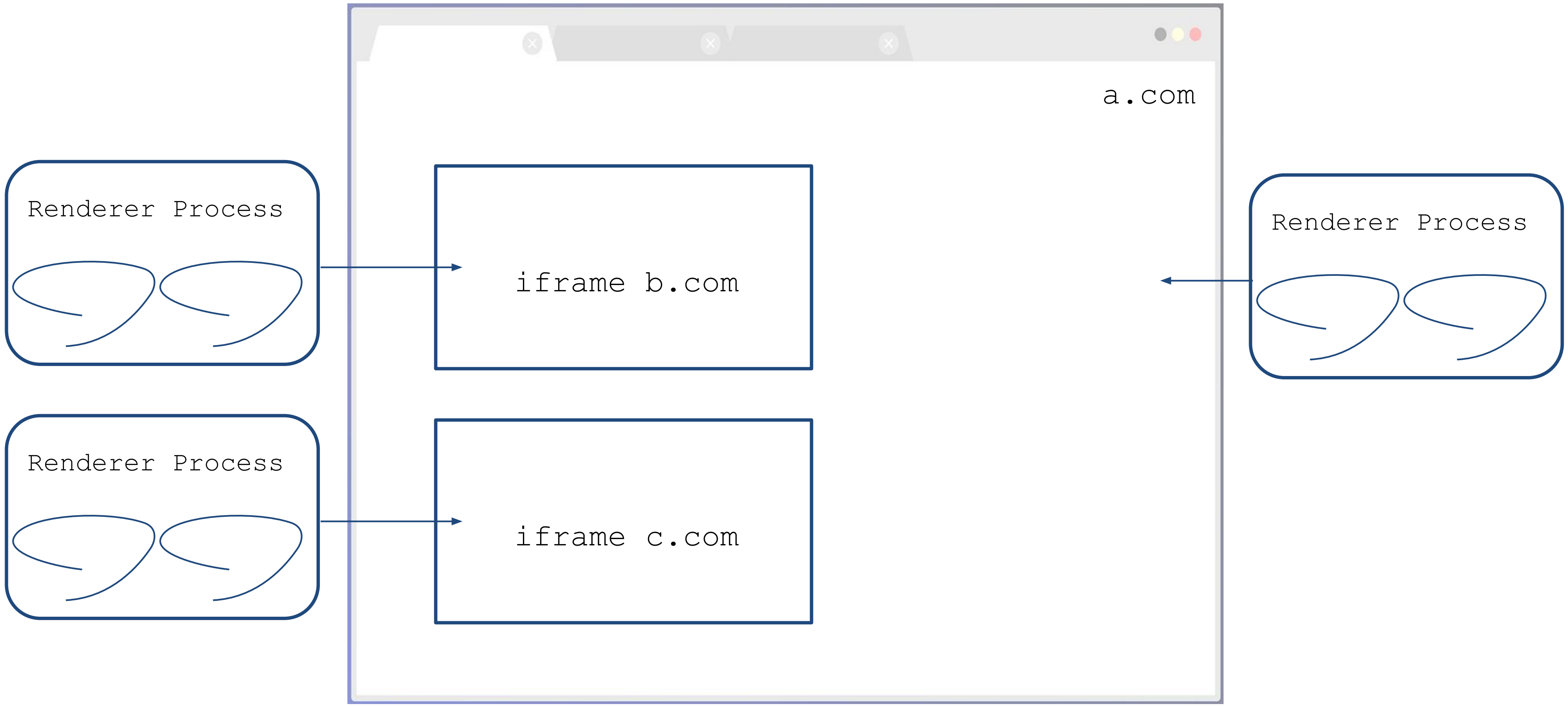
## Same Origin Policy (SOP)

- SOP introduced by Netscape in 1995
  - Implemented by all  browsers today
  - Result of introduction of JavaScript that manipulates DOM
- Restricts how a document or script loaded by one origin can interact with a resource from another origin
  - What is an origin?
  - Origin: protocol + hostname + port
    - (http) + (www.iitb.ac.in) + (80)

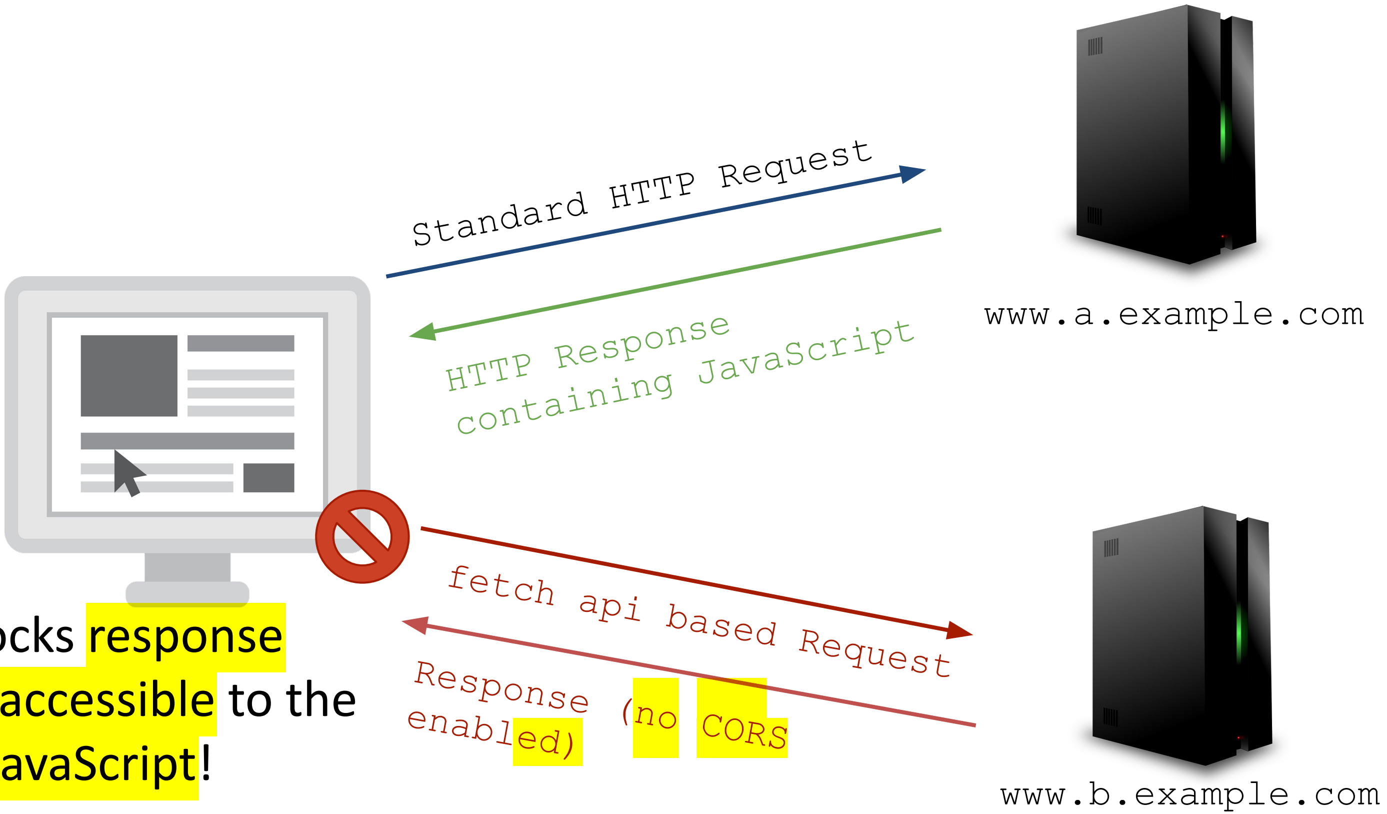
| URL1 | URL2 | Same origin? |
|---|---|---|
| http://abc.org/a | http://abc.org/b | |
| http://abc.org | http://www.abc.org | |
| http://abc.org | https://abc.org | |
| http://abc.org:81 | http://abc.org:82 | |

| URL1 | URL2 | Same origin? |
|---|---|---|
| http://abc.org/a | http://abc.org/b | Yes |
| http://abc.org | http://www.abc.org | No (hostname different) |
| http://abc.org | https://abc.org | No (protocol different) |
| http://abc.org:81 | http://abc.org:82 | No (port different) |

- Normally, one origin cannot access resources of another origin
  - A malicious script loaded from one origin cannot access **sensitive data** of a page loaded from different origin
    - Sensitive Data: Cookies or response to HTTP request or DOM object


a.com
Renderer Process
iframe b.com
Renderer Process
Renderer Process
iframe c.com

- Cross-origin reads via scripts are disallowed by default
  - Note: Web page can still freely embed cross-origin images, stylesheets, scripts, iframes, and videos etc
  - But “cross-origin” requests, notably Ajax requests via JavaScript are forbidden by default
    - Possible to allow via Cross-Origin Resource Sharing (CORS, will be covered later)



Browser blocks response from being accessible to the client-side JavaScript!

## Many more such mechanisms

- Automatic Security Updates
- Secure Credential Management
  - Securely store and manage user credentials
- Phishing and Malware Protection
  - Browsers talk with safe browsing services that maintain databases of known phishing sites and malicious URLs
  - Warn users before navigating to such sites
- Privacy Controls
  - Include features to block or limit third-party tracking cookies
- Many policies implemented via http headers, which browsers execute (more later)

# References

- https://developers.google.com/web/updates/2018/09/inside-browser-part1
- https://developer.chrome.com/blog/inside-browser-part2/
- https://developer.chrome.com/blog/inside-browser-part3/
- https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy
- Browser Security: Lessons from Google Chrome: https://queue.acm.org/detail.cfm?id=1556050

# Summary

- Lots of action inside a browser
  - We just touched the surface! (see references for depth)
- Basic Security Mechanisms employed by browsers
  - Sandboxing
  - Same Origin Policy (SOP)
  - (more, specifically HTTP headers covered over time)