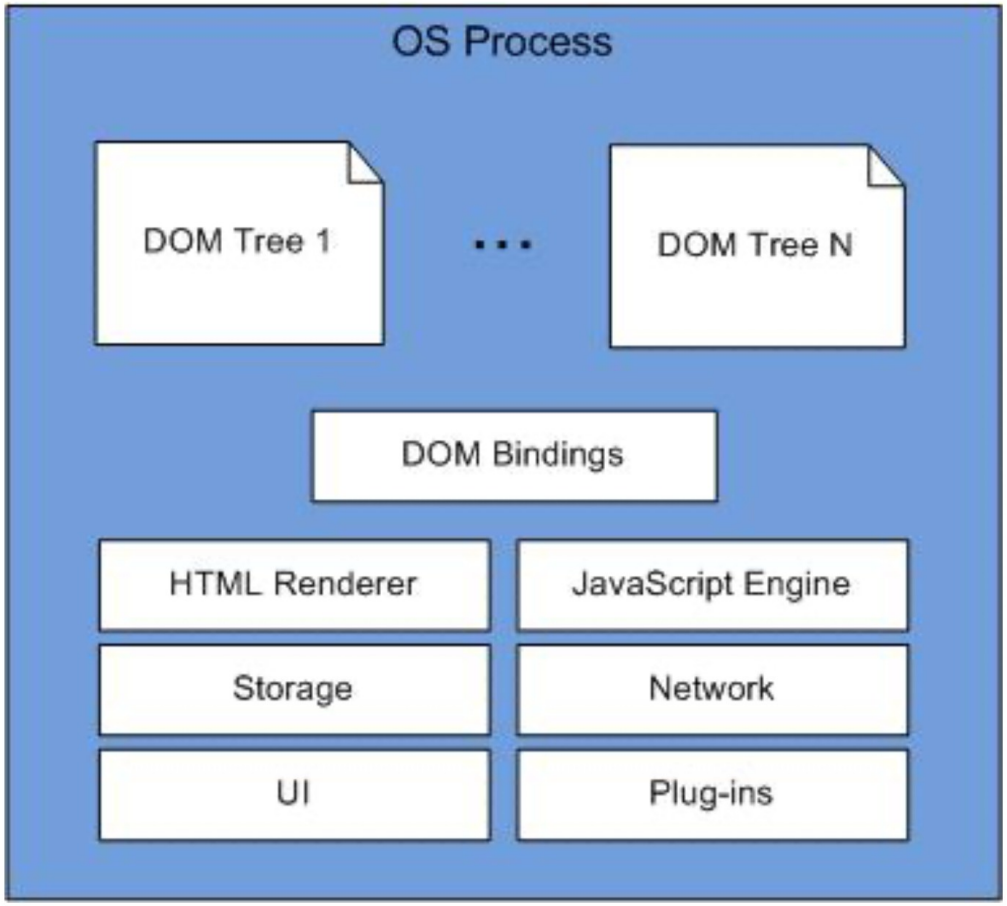
Secure Software Design and Engineering
(CY-321)

Web Browser Security

Dr. Zubair Ahmad

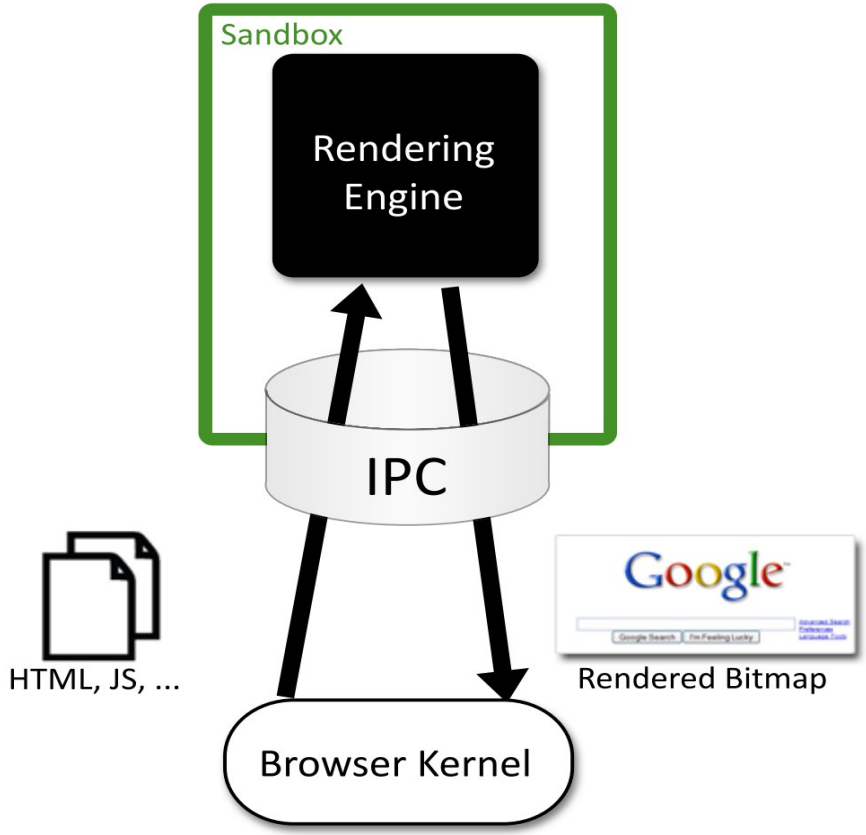
Web Browser Security

- Browsers are complicated
  - Large and complex codebase
  - Network-visible interface
  - Written in memory-unsafe languages
  - Lots of new features added regularly
- Without a solid architecture, this is a recipe for disaster!

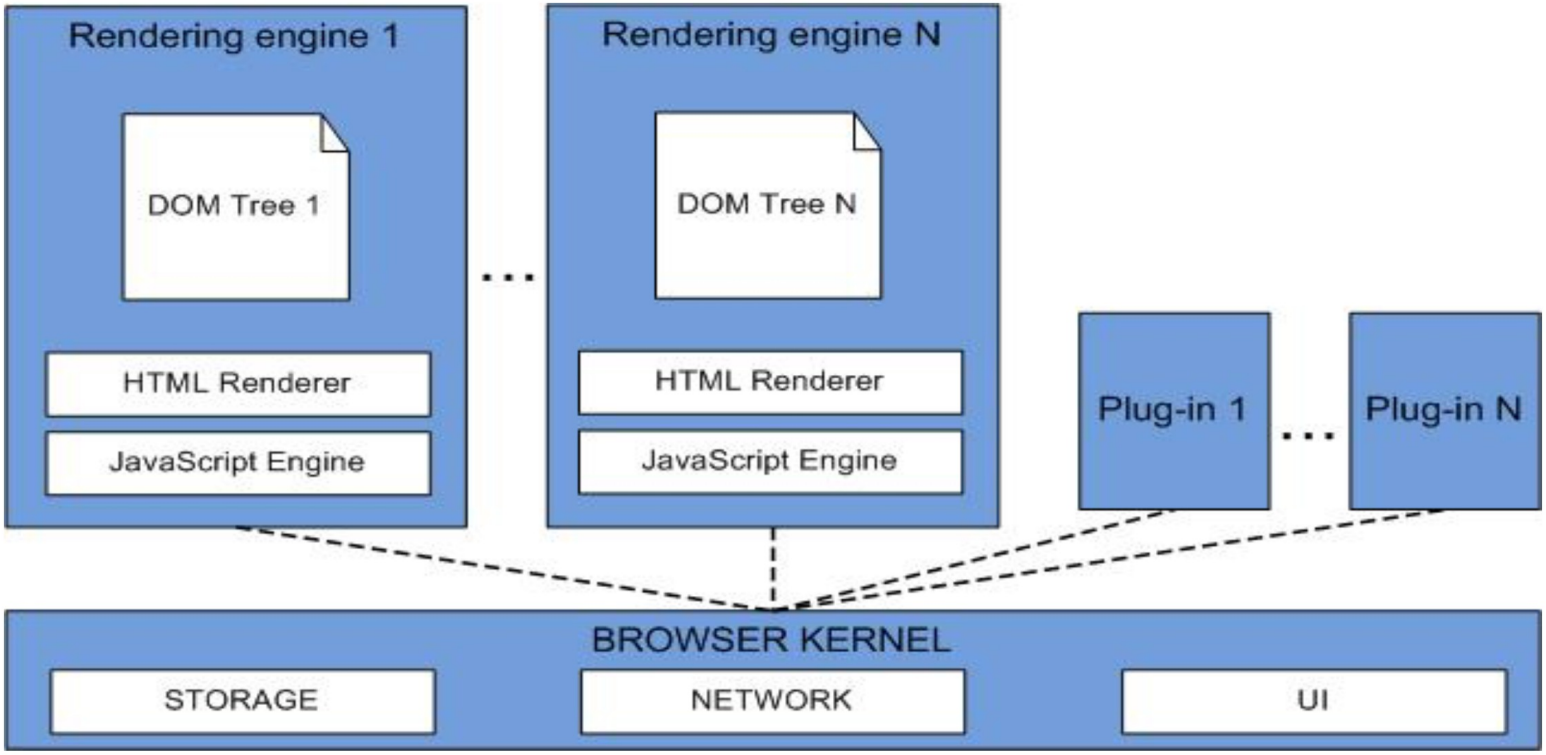

OS Process
DOM Tree 1
...
DOM Tree N
DOM Bindings
HTML Renderer
JavaScript Engine
Storage
Network
UI
Plug-ins

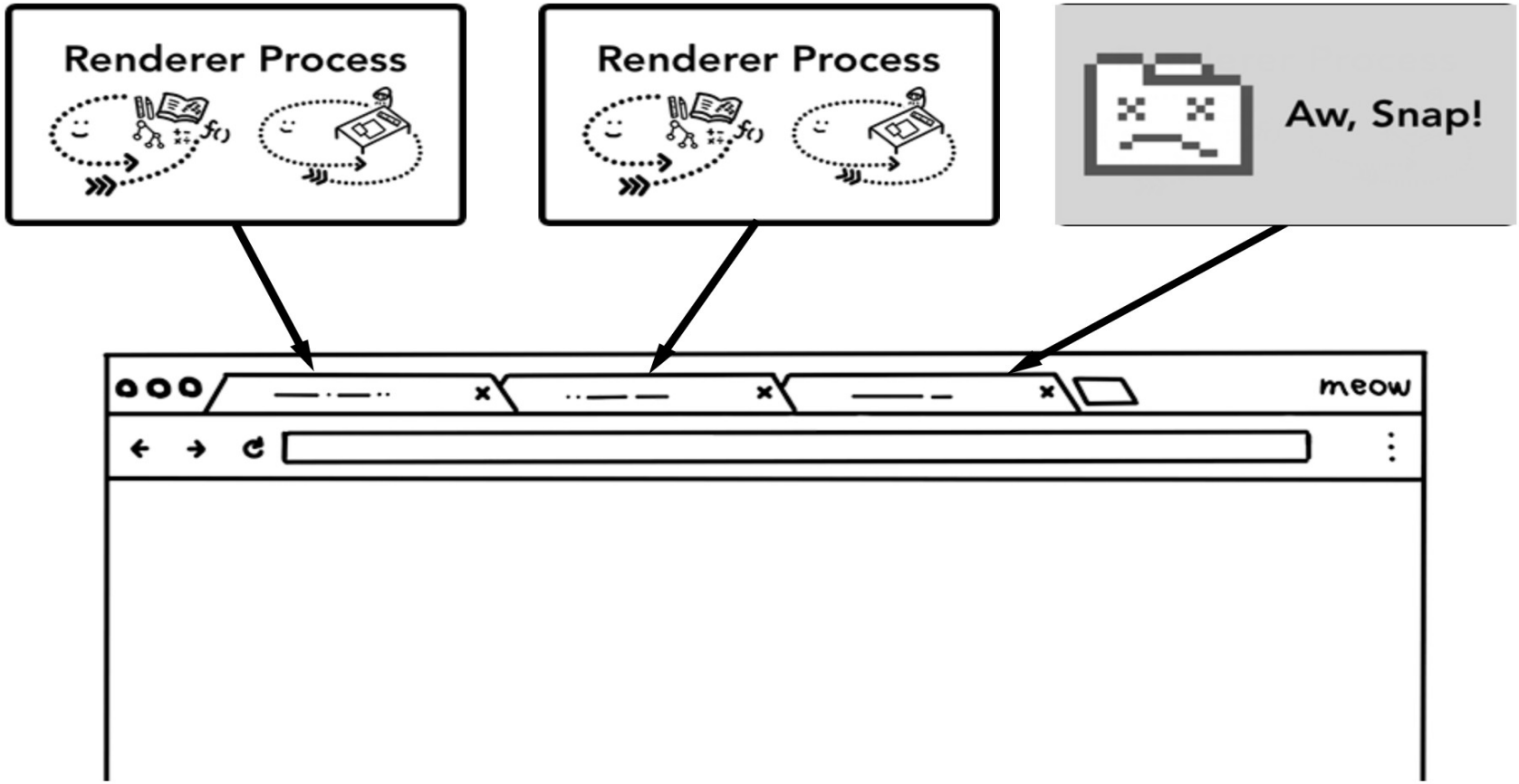
Rendering Engine	Browser Kernel
HTML parsing	Cookie database
CSS parsing	History database
Image decoding	Password database
JavaScript interpreter	Window management
Regular expressions	Location bar
Layout	Safe Browsing blacklist
Document Object Model	Network stack
Rendering	SSL/TLS
SVG	Disk cache
XML parsing	Download manager
XSLT	Clipboard

Both
URL parsing
Unicode parsing


Sandbox
Rendering Engine
IPC
HTML, JS, ...
Google
Rendered Bitmap
Browser Kernel

Chrome Architecture




Renderer Process
Renderer Process
Aw, Snap!
meow

DOM (Document Object Model)

a **tree-like, in-memory representation of an HTML**

Acts as an interface between web pages and scripts (e.g., JavaScript), allowing programs to read and manipulate the structure, style, and content of a document.

DOM Tree

```
Document
└──── html
    ├──── head
    │     └──── title
    │           └──── "My Page"
    └──── body
          ├──── h1
          │     └──── "Welcome!"
          └──── p
                └──── "This is a paragraph."
```

DOM (Document Object Model)

```
<!DOCTYPE html>
<html>
  <head>
    <title>Page Title</title>
  </head>
  <body>
    <h1>Hello</h1>
    <p>This is a
<strong>test</strong>.</p>
  </body>
</html>
```

Problem: == uses Abstract Equality Comparison Algorithm

```
function isZero(arg) {
  return arg == 0;
}
```

Possible Outcome?

Problem: == uses Abstract Equality Comparison Algorithm

```
function isZero(arg) {
  return arg == 0;
}



isZero(0);        // true

isZero('0');      // true

isZero(false);    // true
```

Solution: Always use ===

```
function isZero(arg) {
  return arg === 0;
}
```

```
isZero(0);        // true

isZero('0');      // false

isZero(false);    // false
```

Problem: Duplicate function arguments are allowed

```
function foo(a, b, a) {
  console.log(`value of the second
a is ${a}`)
}
```

Possible Outcome?

Solution: User strict mode

```
’   user strict’
function foo(a, b, a) {

// SyntaxError: Duplicate parameter name not allowed in this
context
  console.log(`value of the second a is ${a}`)
}
```

Solution: User strict mode

- Opt in to a restricted variant of JavaScript
- Not merely a subset. strict mode intentionally has different semantics from normal code
  - Eliminates some silent errors by changing them to throw errors
  - Fixes mistakes that make it difficult for JavaScript engines to perform optimizations

Problem: Duplicate keys in the objects are allowed

```
let foo = {
  bar: 'baz',
  bar: 'qux'
};
```

Possible Outcome?

Solution: Use a linter

```
let foo = {
  bar: 'baz',
  bar: 'qux'
};
```

```
// Standard: Use JavaScript Standard Style
// /Users/feross/websec/script.js:5:3: Duplicate key 'bar'.
```

Problem: Using Object.prototype

```
const obj = { foo: 1, bar: 2, hasOwnproperty: 3 }
obj.hasOwnproperty('bar')

const obj = Object.create(null)
obj.hasOwnproperty('bar')
```

Possible Outcome?

Problem: Use a linter

```
const obj = { foo: 1, bar: 2, hasOwnproperty: 3 }
obj.hasOwnproperty('bar') // TypeError: obj.hasOwnproperty is
not a function

const obj = Object.create(null)
obj.hasOwnproperty('bar') // TypeError: obj.hasOwnproperty is
not a function
```

Problem: Leading zero in numbers treated as octal

```
let num = 010

let num = 011

let num = 099
```

Possible Outcome?

Problem: Leading zero in numbers treated as octal

```
let num = 010 // 8
let num = 011 // 9
let num = 099 // 99
```

Solution: User strict mode

```
let num = 010
```

```
$ standard
standard: Use JavaScript Standard Style
/Users/feross/websec/script.js:1:11: Parsing
error: Invalid number
```

Problem: Variables default to global without var, let or const

```
function foo () {
x = 5
y = 10
return x + y
}

foo()
console.log(`${x} ${y}`)
// Prints '5 10'
```

Possible Issue?

Solution: User strict mode

```
'use strict'
function foo () {
x = 5
y = 10
return x + y
}

foo() // ReferenceError: x
is not defined
console.log(`${x} ${y}`)
```

```
undefined = true
console.log('hey') // prints 'hey'
```

Possible Issue?

Solution: User strict mode

```
"use strict";
undefined = true; // TypeError: Cannot assign to read only property 'undefined'
```

Questions??

zubair.ahmad@giki.edu.pk

Office: G14 FCSE lobby