



Modern Web Browsers and JS Engine

Rujuan Xing

Maharishi International University - Fairfield, Iowa

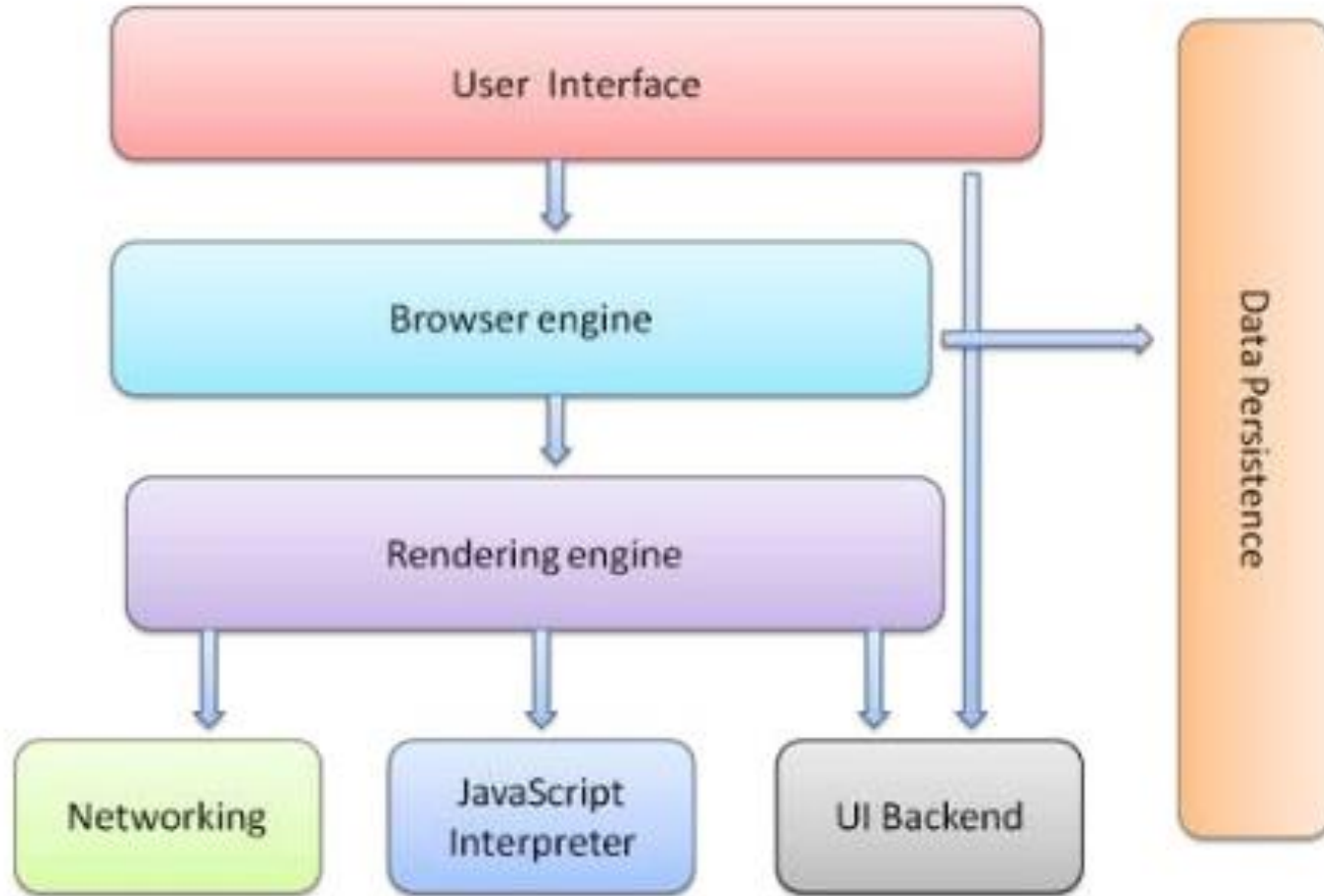


All rights reserved. No part of this slide presentation may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying or recording, or by any information storage and retrieval system, without permission in writing from Maharishi University of Management.



Modern Web Browser

The Browser Architecture



Browser Components

- **User Interface** is the top bar in the browser, where control lies. It includes space where we type the URL, have back/forward buttons, space where tabs and setting options are.



- **Browser engine** is the bridge between User Interface and the Rendering engine. Based on the inputs from a user it queries and manipulates the rendering engine.
- **Rendering engine** is responsible for parsing HTML/CSS/XML and displaying it into the empty space below User-Interface.
 - Parses HTML by looking for `</>` and related symbols and attributes, then it parses CSS and finally scripts.
 - HTML parsing makes DOM while CSS parsing results in CSSOM. Both of which combine to make a render tree. This render tree has 'document' as its head and is exposed to JS as JS cannot directly use the DOM tree made by HTML parsers. Then the layout of elements is decided based on the CSS box model and position defined in CSS properties. This is finally painted.

Browser Components

- **Networking** is fetching resources and handling everything related to the internet.
- **Javascript interpreter** is JS engine which reads and executes the js code and handles the result to the rendering engine.
- **UI backend** is used for drawing basic widgets like combo boxes and windows. This backend exposes a generic interface that is not platform specific. It underneath uses operating system user interface methods.
- **Data Persistence** is support for storage mechanisms such as localStorage, IndexedDB, WebSQL, and FileSystem. It is a small database created on the local drive of the computer where the browser is installed. It manages user data such as cache, cookies, bookmarks, and preferences.

JS Engine

The background features a series of overlapping, wavy, paper-like layers in a color palette of teal, green, yellow, and orange. The layers create a sense of depth and movement, with the text 'JS Engine' centered over the middle section.

ECMAScript

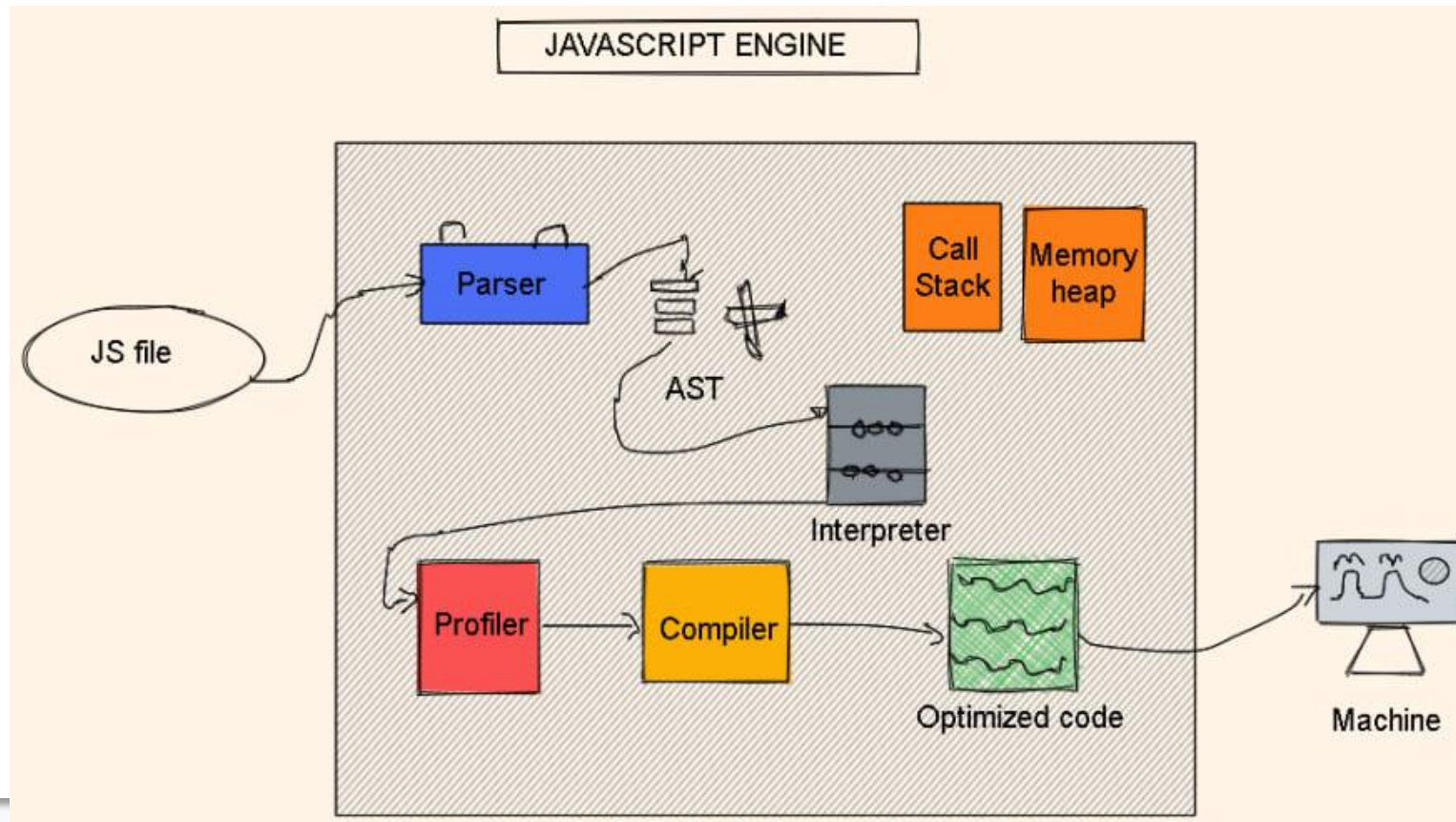
- **ECMAScript** is the scripting language that forms the basis of [JavaScript](#). ECMAScript standardized by the [ECMA International](#) standards organization in the **ECMA-262** and **ECMA-402 specifications**.
- ECMAScript Specification
 - ES5: <https://262.ecma-international.org/5.1/>
 - The latest version ES2020: <https://262.ecma-international.org/11.0/>
- Based on the specification, you can create your own JavaScript Engine.

JavaScript Engine

- A **JavaScript engine** is a [computer program](#) that executes [JavaScript](#) code.
- It should follow the ECMAScript standard and how the language should work and what features it should have.
- Notable Engines:
 - [V8](#) from [Google](#) is the most used JavaScript engine. [Google Chrome](#) and the many other [Chromium](#)-based browsers use it, or any other [framework](#) that embeds Chromium. Other uses include the [Node.js](#) and [Deno runtime systems](#).
 - [SpiderMonkey](#) is developed by [Mozilla](#) for use in [Firefox](#) and its [forks](#).
 - [JavaScriptCore](#) is [Apple](#)'s engine for its [Safari](#) browser. Other [WebKit](#)-based browsers also use it.
 - [Chakra](#) is the engine of the [Internet Explorer](#) browser. It was also forked by [Microsoft](#) for the original [Edge](#) browser, but Edge was later rebuilt as a Chromium-based browser and thus now uses V8.

The JavaScript Engine Pipeline

- JS is a higher level dynamic language and it has no way to directly interact with our machines lower level logic. So JavaScript engine can be implemented as a standard interpreter, or just-in-time compiler that compiles JavaScript to bytecode in some form.



Details about JS Engine Pipeline

- Parser
 - The Html Parser will fetch all scripts loaded via `<script>` tag.
 - The source code inside this script gets loaded as a UTF-16 byte stream to a byte stream decoder.
 - This byte stream decoder then decodes the bytes into token and then its sent to parser.
- AST
 - The parser creates nodes based on the tokens it receives. With these nodes, it creates an Abstract Syntax Tree (AST).
 - <https://astexplorer.net/>
- Interpreter
 - Walks through the AST and generates byte code.
 - Reads the code line by line. When the byte code has been generated, the AST is deleted for clearing up memory space.
 - The problem with interpreters is that running the same code multiple times can get really slow.

Details about JS Engine Pipeline (Cont.)

- Profiler (JiT)
 - The Profiler monitors and watches code to optimize it.
 - While the code is executed by the interpreter, a profiler will keep track of how many times the different statements get hit. The moment that number starts growing, it'll mark it as **Warm** and if it grows enough, it'll mark it as **Hot**.
 - In other words, it'll detect which parts of your code are being used the most, and then it'll send them over to be compiled and stored.
- Compiler
 - The compiler works ahead of time and creates a translation of the code that has been written and compiles down to a lower level language that machines can read.
- Optimized code
 - highly-optimized machine code which can run faster


Example of JS Optimize Code

- Function Inlining
- Browser will sometimes essentially rewrite your JavaScript.
- As you can see from the right, V8 is essentially removing the step where we call `func` and instead inlining the body of `square`. This is very useful as it will improve the performance of our code.

```
const square = (x) => { return x * x };

const callFunction100Times = (func) => {
  for(let i = 0; i < 100; i++) {
    // the func param will be called 100 times
    func(2);
  }
}

callFunction100Times(square);
```



```
const square = (x) => { return x * x };

const callFunction100Times = (func) => {
  for (let i = 100; i < 100; i++) {
    // the function is inlined so we don't
    // have to keep calling func
    return x * x;
  }
}

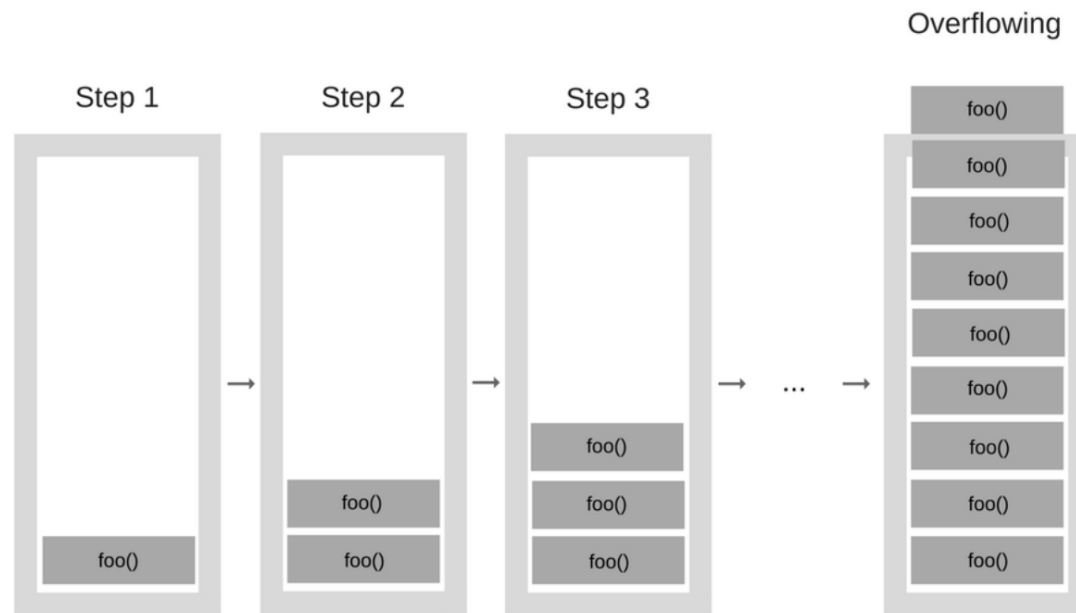
callFunction100Times(square);
```

The Memory Heap

- The memory heap is a place to store and write information, where we allocate, use and remove memory.
- Whenever you define a variable, constant, object, etc in your JavaScript program, it is stored in the memory heap. This memory is limited, and it is essential to make wise use of the available memory.
- Unlike languages like C, where we need to explicitly allocate and free memory, JavaScript provides the feature of automatic garbage collection. Once an object/variable is not used by any execution context, its memory is reclaimed and returned to the free memory pool. In V8 the garbage collector is named as **Orinoco**.

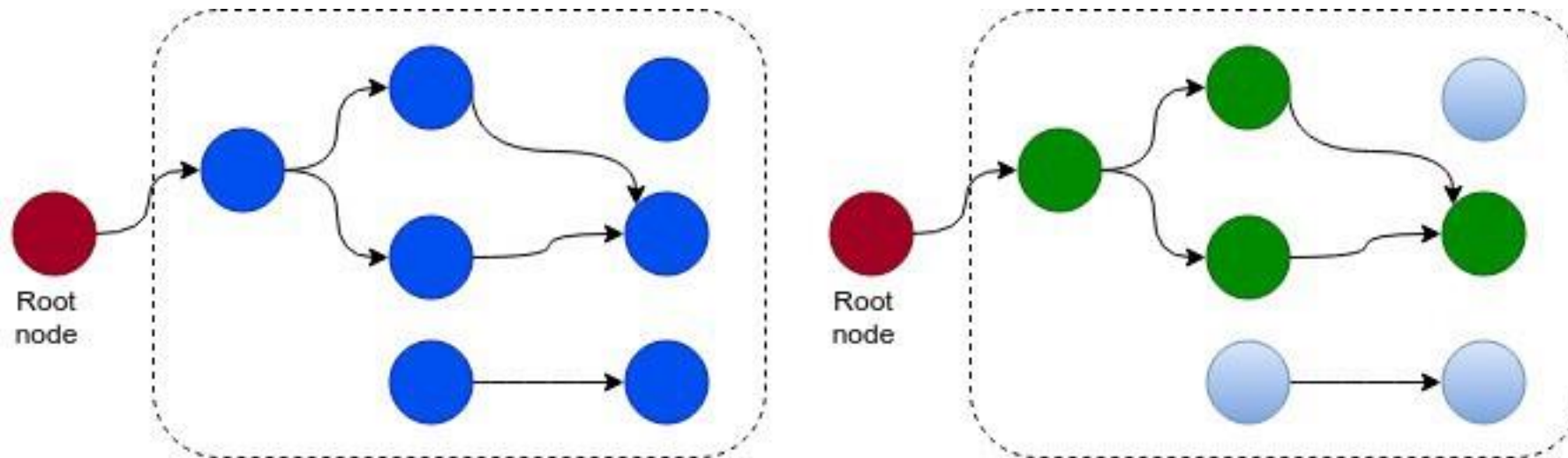
Call stack

- The call stack keeps track of where we are in the code. It uses first in and last out and stacks for example functions.
- The callstack calls a function from the memory heap and after executing removes it from the stack. When the maximum call stack has been reached, e.g. with an infinite loop, we call it a **stack overflow**.



Garbage Collection

- JS automatically frees up memory that no longer is used. It marks and sweeps it from the memory.
 - E.g. when a value gets reassigned and the original value is not used anymore.
- The garbage collector starts with the root or global objects periodically and moves to the objects referenced by them, and then to the objects referenced by these references and so on. All the unreachable objects are then cleared.



References

- <https://developers.google.com/web/updates/2018/09/inside-browser-part1>
- <https://cabulous.medium.com/how-browser-works-part-i-process-and-thread-f63a9111bae9>
- <https://cabulous.medium.com/how-does-browser-work-in-2019-part-ii-navigation-342b27e56d7b>
- <https://cabulous.medium.com/how-does-browser-work-in-2019-part-iii-rendering-phase-i-850c8935958f>
- <https://medium.com/@sanderdebr/a-brief-explanation-of-the-javascript-engine-and-runtime-a0c27cb1a397>
- <https://medium.com/web-god-mode/how-web-browsers-work-behind-the-scene-architecture-technologies-and-internal-working-fec601488bfa>