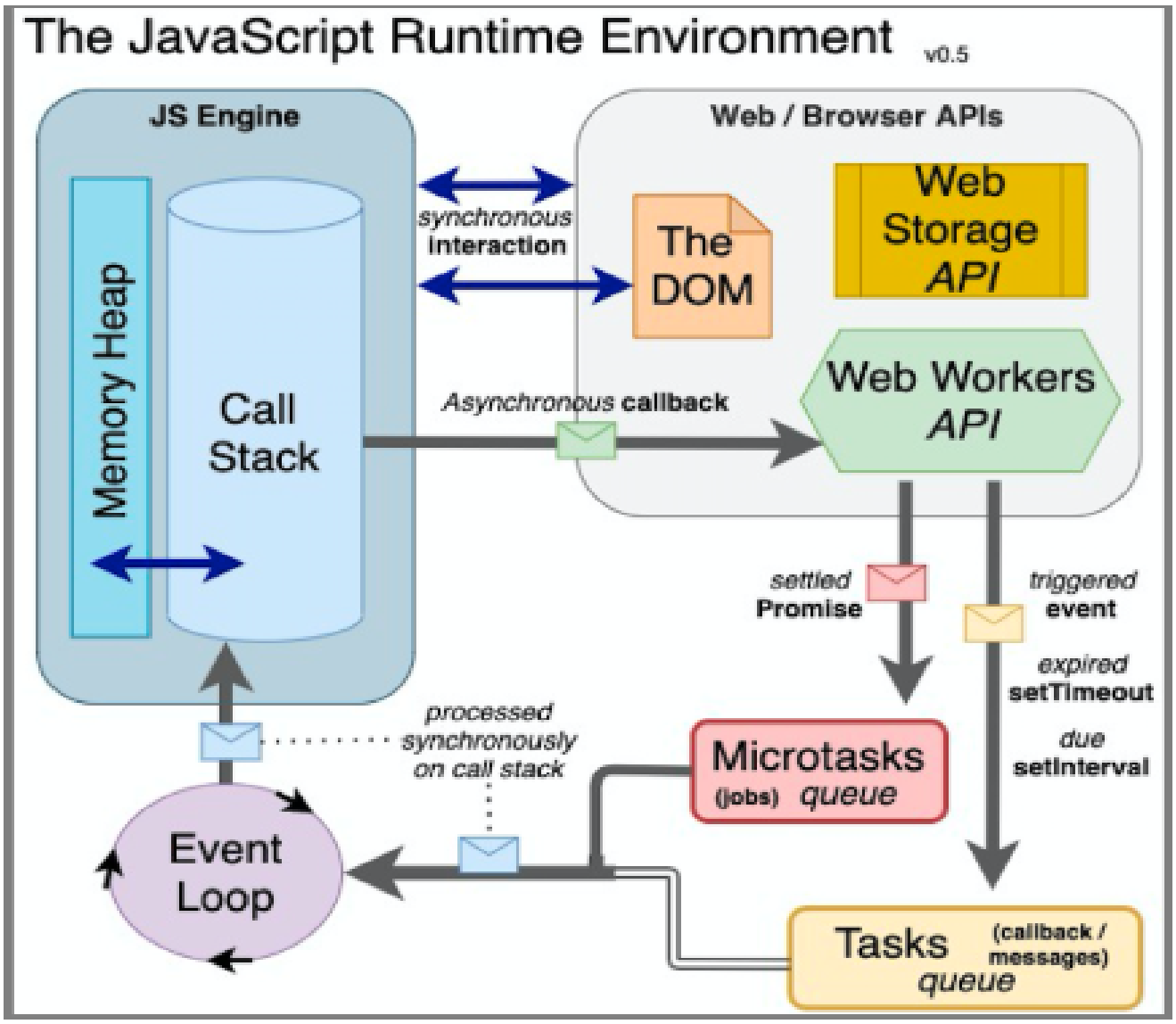
**17/12/2024 : Documentation on V8 Engine**




- What is heap?
  The memory Heap is the place where all the variables and functions are assigned to the memory. It also has a garbage collector that used to free up memory space whenever possible.

- What is stack?
  The Job of the call stack is to execute the code. It has Global execution context that invokes in the call stack during run time and executes the code line by line

- What is web api?
  A Web API is an Application Programming Interface (API) that enables communication and data exchange between applications over the web, typically using HTTP protocols.

- What is a callback queue?
  The callback queue in JavaScript is a queue data structure that stores callback functions waiting to be executed. It operates on the FIFO (First-In, First-Out) principle. These callbacks are typically associated with asynchronous operations, such as:

  1. **Timers:** setTimeout and setInterval
  2. **Event handlers:** User interactions like clicks or key presses.
  3. **I/O operations:** Network requests or file system access.

- What is an event loop?
  The event loop ensures that tasks are executed in the correct order, enabling asynchronous programming. **or**
  The event loop is a crucial mechanism in JavaScript that enables it to handle asynchronous operations efficiently, despite being single-threaded. It continuously monitors the call stack and the task queue, executing tasks in a non-blocking manner. When the call stack is empty, the event loop takes the first event from the task queue and pushes it onto the call stack for execution. This process allows JavaScript to manage multiple tasks concurrently, such as handling user interactions, fetching data, and setting timers, without blocking the main thread.

- Synchronous vs Asynchronous

| Aspect | Synchronous | Asynchronous |
|---|---|---|
| Execution Order | Executes **line by line**, in the exact order. | Does **not wait** for tasks to complete before moving to the next one. |
| Blocking | **Blocks** further execution until the current task is finished. | **Non-blocking**, allows next tasks to run without waiting. |
| Performance | Slower for tasks like network requests or timers. | Faster for API calls, Timers (`setTimeout`), File loading, Waiting for user input (clicks, scrolls, etc.) tasks as they run in the background. |
| Use Case | Simple operations like math, DOM manipulation. | Time-consuming tasks: API calls, timers, file loading. |
| Example | console.log("A");<br>console.log("B"); | setTimeout(() =><br>console.log("A"), 1000);<br>console.log("B"); |
| Main Tools | Plain JavaScript execution. | Web APIs (e.g., `setTimeout`, `fetch`), Promises, `async/await`, Events. |
| Waiting Behavior | Next line **waits** for the current one to finish. | Next line **runs immediately**, async results are handled later. |
| Thread | Runs in a **single thread** sequentially. | Uses the **event loop** to manage background tasks and callbacks. |