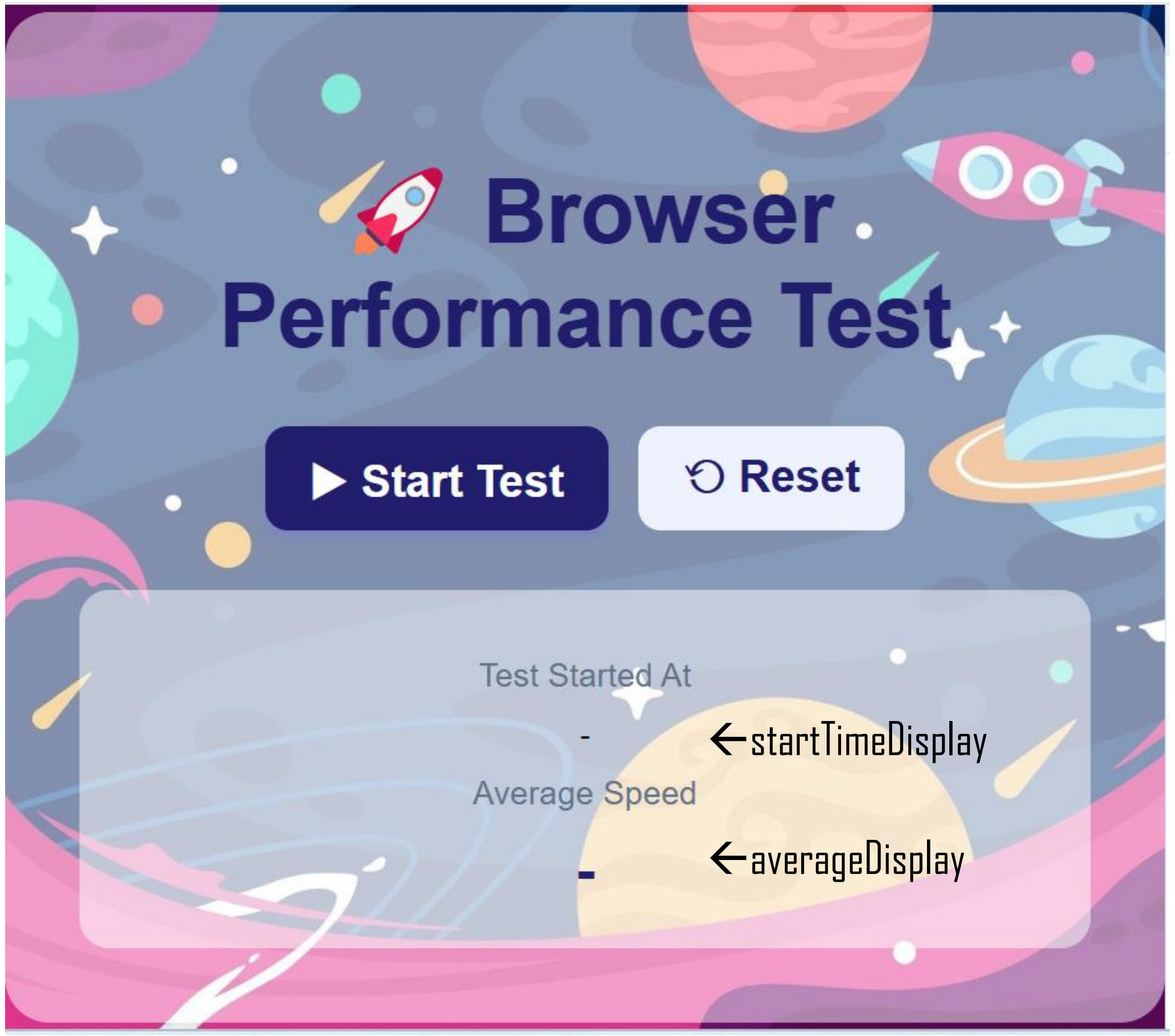
EVENT LOOP IN JS

Presented by : Yousuf And Usman

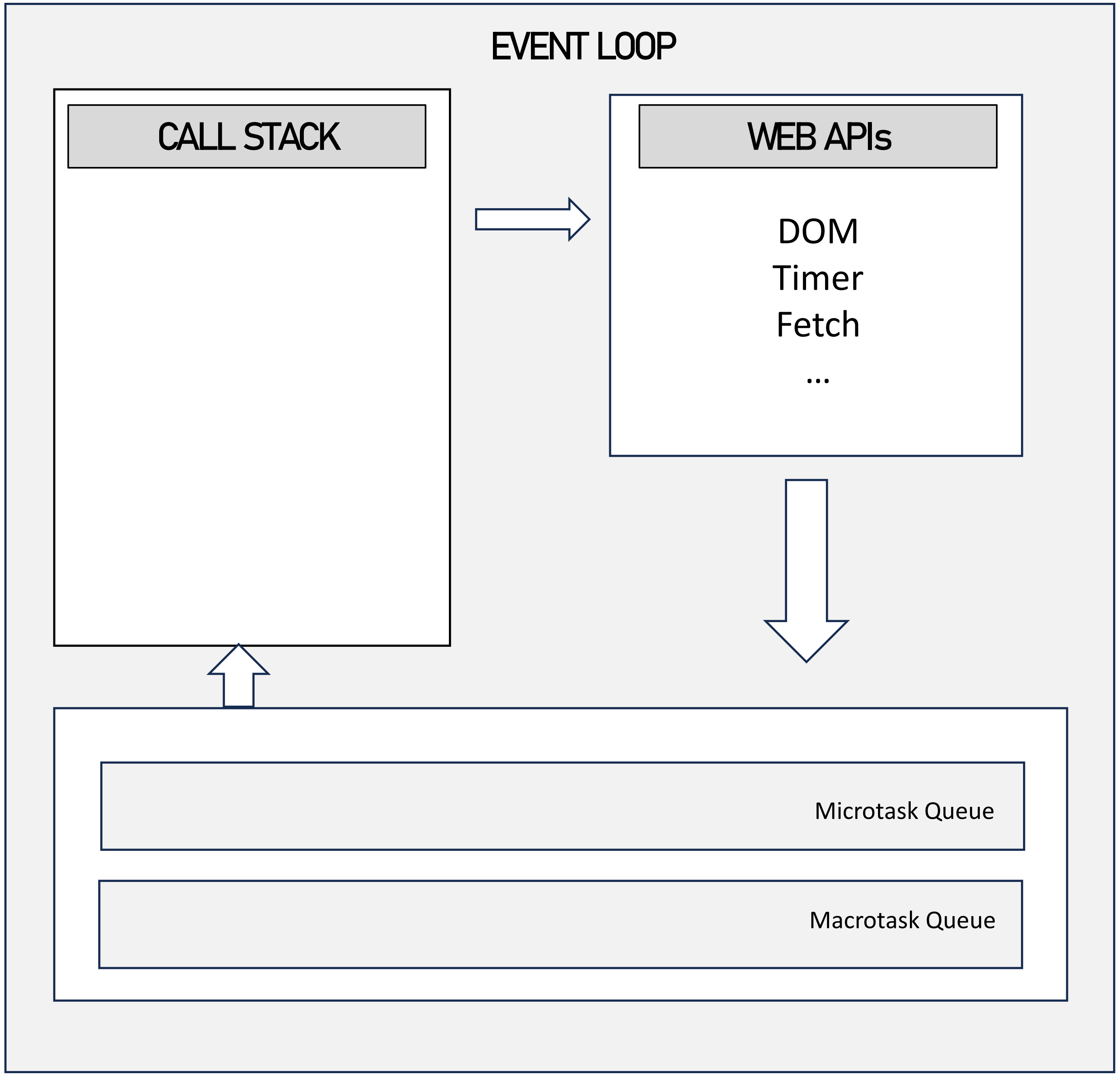
PROBLEM



```
function startTest() {
  if (testActive) return;
  testActive = true;
  resetTest();

  startTime = Date.now();
  clickCount = 0;

  while (Date.now() - startTime < 10000) {
    alert(`Alert ${clickCount + 1}`);
    clickCount++;
  }

  const average = (clickCount / 10).toFixed(2);

  startTimeDisplay.textContent = new
         Date(startTime).toLocaleTimeString();
  averageDisplay.textContent = average;

  const resultsStart = Date.now();
  while (Date.now() - resultsStart < 2000) {
    alert(`Average speed: ${average} alerts/sec`);
  }

  testActive = false;
}
```

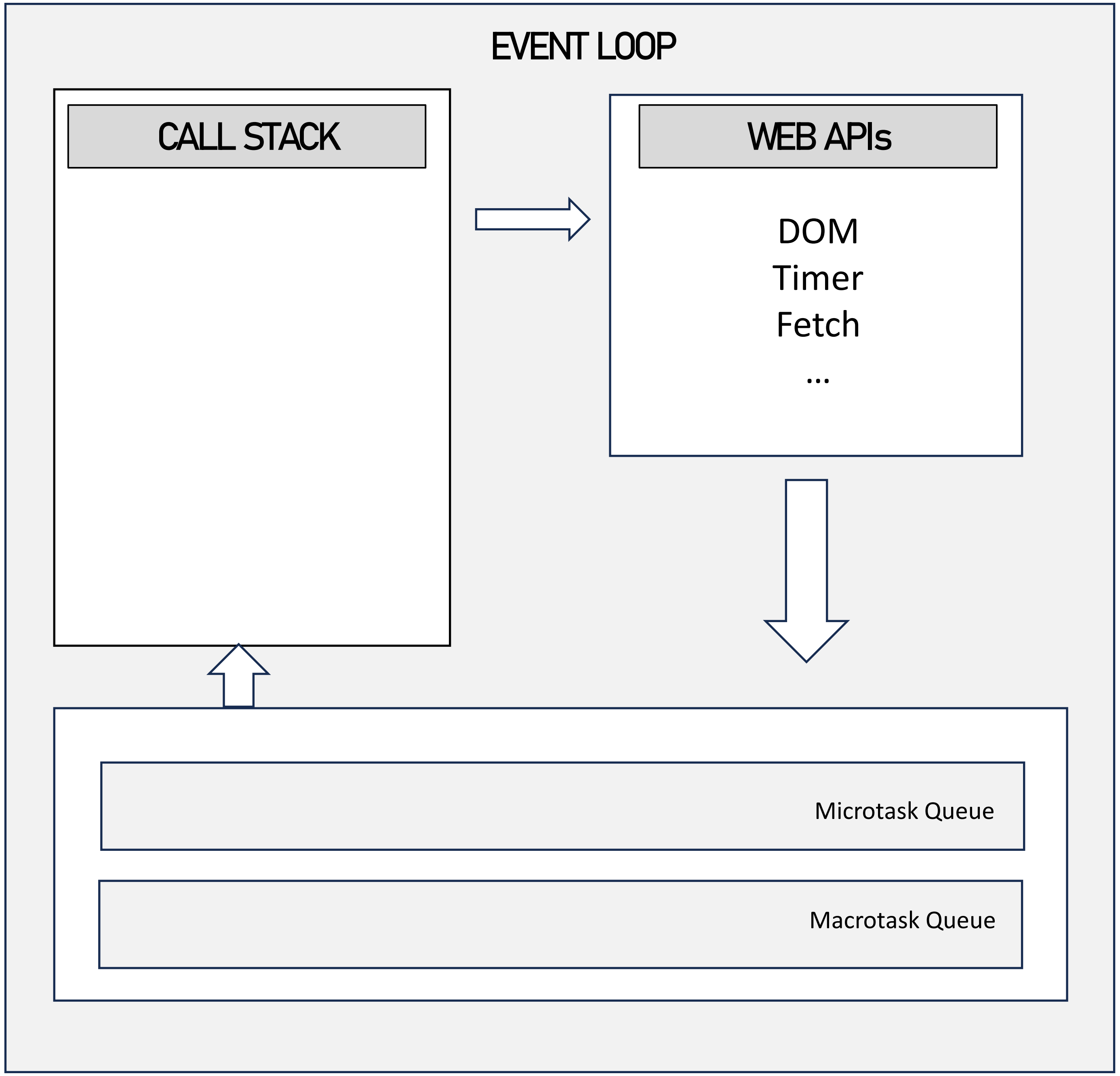
Event Loop in JS

- Javascript is an single threaded language which means it can be able to handle only one task at a time (synchronous task).
- But if the task takes more time to compile and execute then it would block the user to make any interaction in the web page like scrolling, clicking etc.
- So to avoid this block of execution, Javascript has to handle tasks simultaneously on some other thread.
- Javascript uses web api with the support of browser to handle asynchronous tasks on other thread provided by browser
- Tasks handled by web api's are classified into Microtasks and Macrotasks.
- The process of deciding the order of the execution of the task is called Event Loop.
- Event Loop has different groups
  - Call Stack
  - Web Api
  - Microtask Queue
  - Macrotask Queue (Callback Queue)



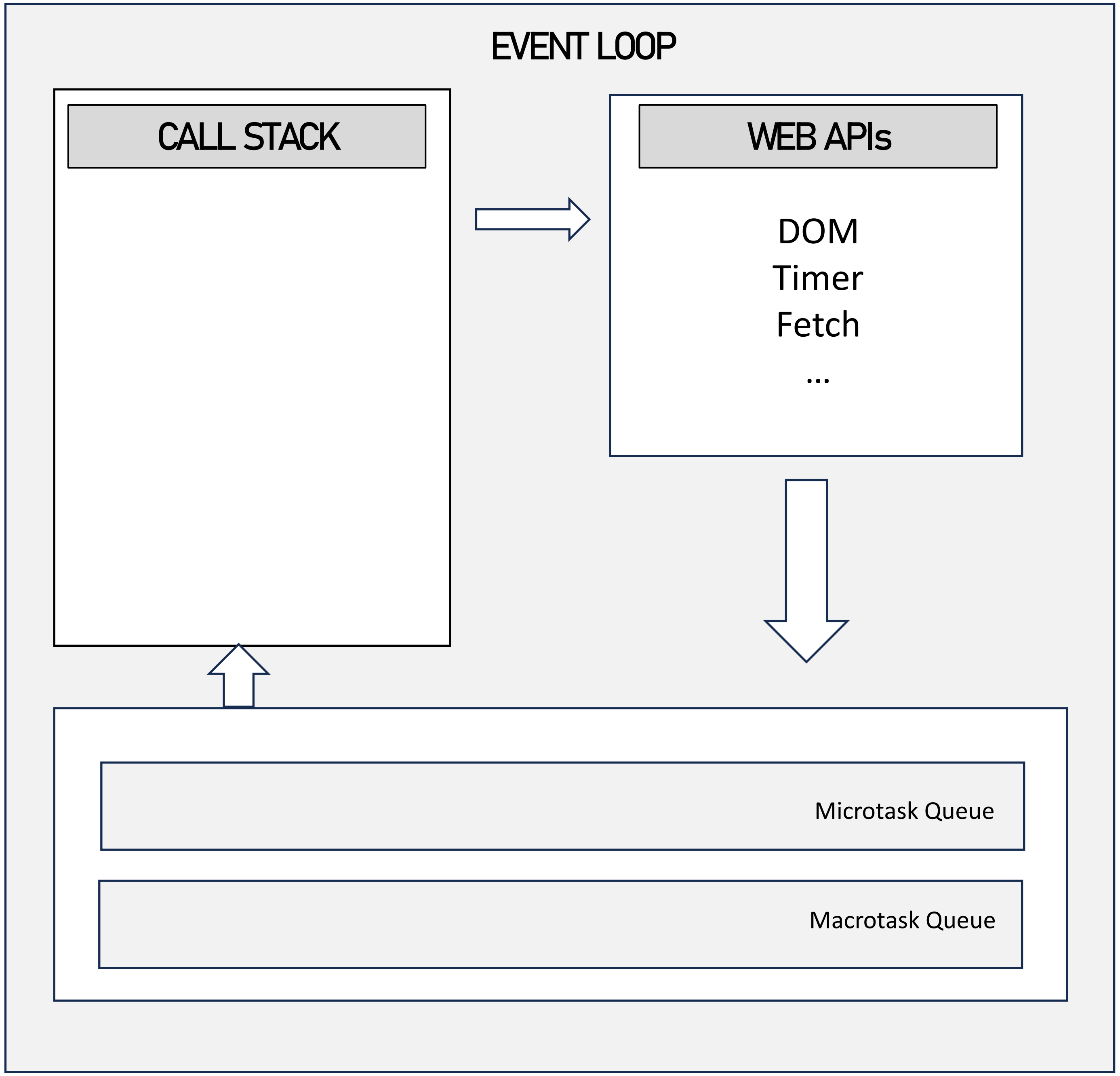
CALL STACK

- This is the single thread where the Javascript runs the program.
- It is an stack data structure so it is LIFO (Last In First Out).
- But only the sync task is handled here, if any async found it will be moved to web api
- Once that async task is processed, it will be moved to either Microtask or Macrotask.
- **Examples Async tasks are:** setTimeout, UI updates, setInteval, promise etc.



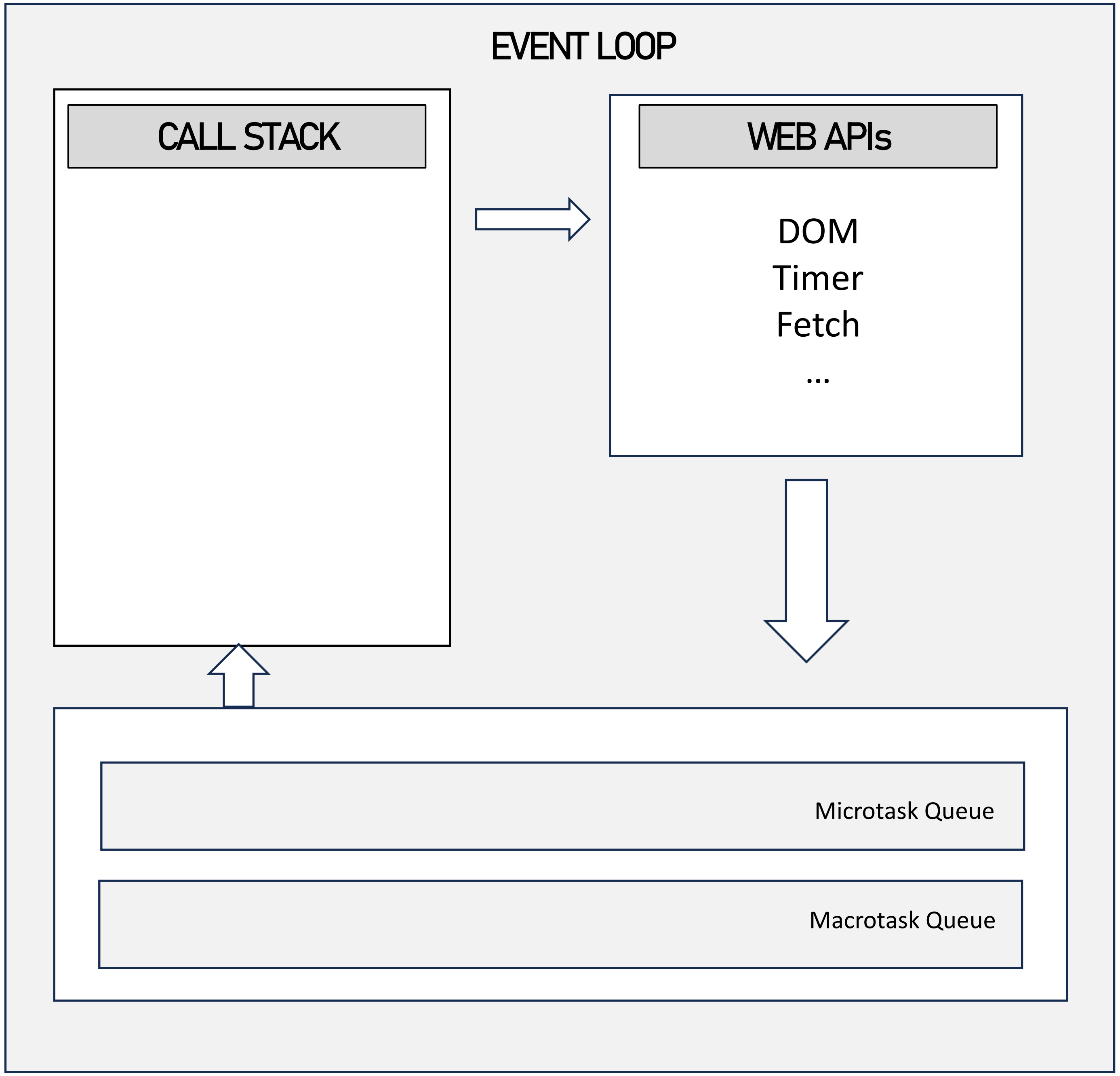
MICROTASK QUEUE

- Microtasks are the high priority tasks that are generally need to be executed as soon as possible.
- As these tasks all are more based on logic and functionality.
- So once the call stack is empty, tasks in Microtasks queue will be moved to call stack and tasks are executed in the same order it was added to the Microtask queue.
- **Example**: Promise, MutationObserver, queueMicrotask



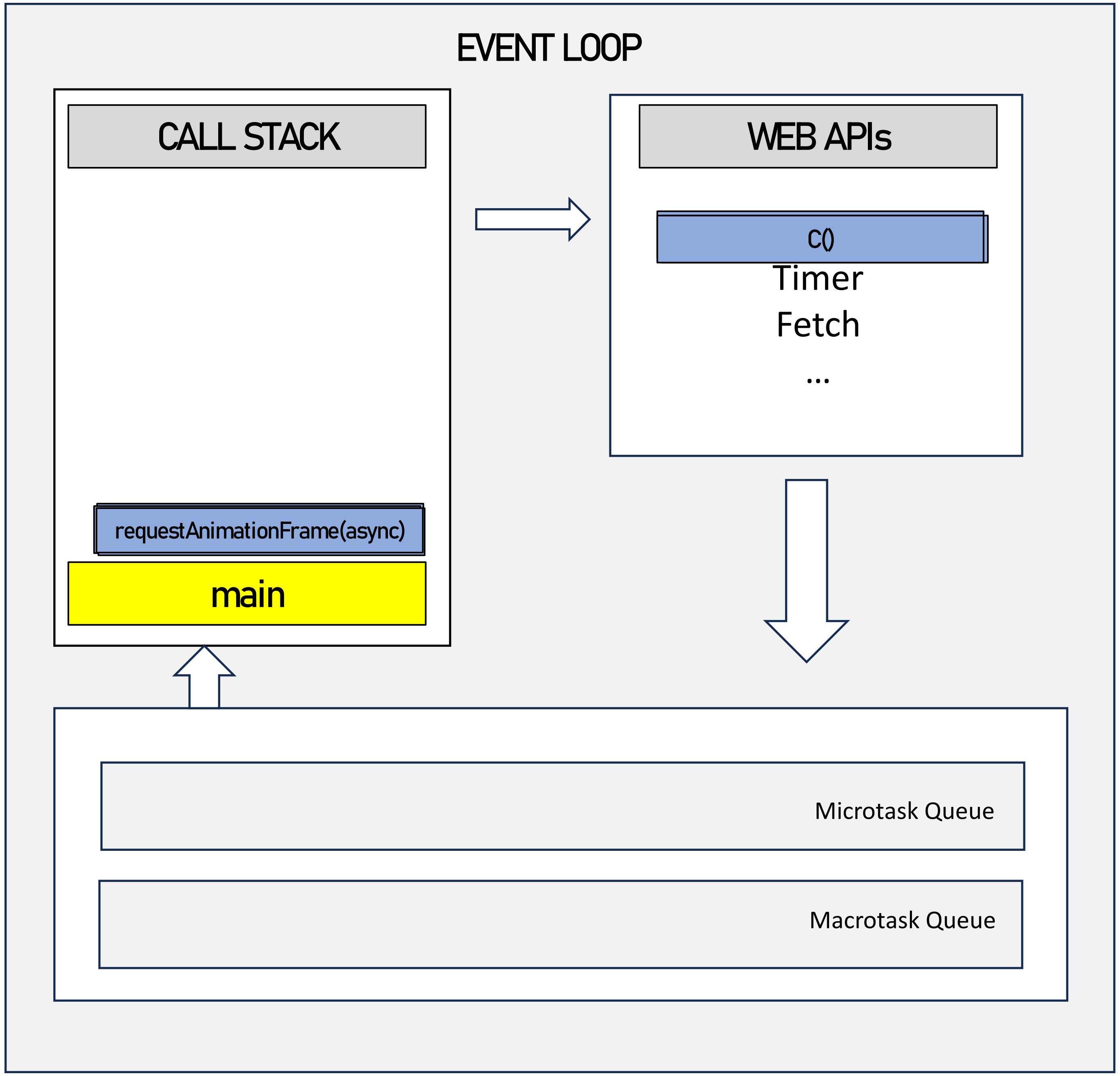
MACROTASK QUEUE

- Macrotasks are lower priority and large scale operations (which takes more time to process).
- **Example:** setTimeout, setInterval, fetch API, DOM events, I/O operations, UI Events



PRIORITIES OF TASKs IN EVENT LOOP

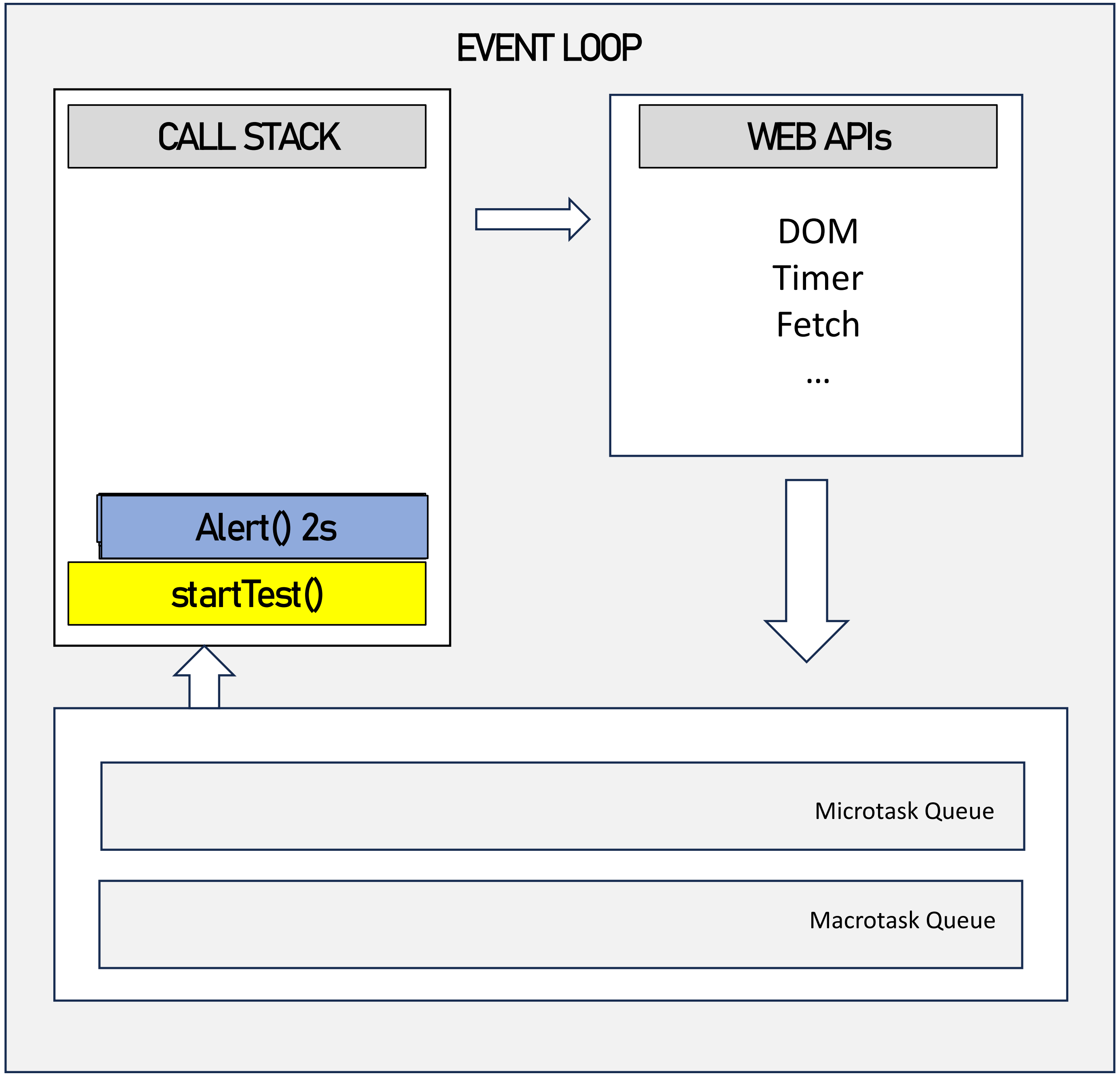
Priority	Task Type	Examples
1	Call Stack (Synchronous Code)	Normal JS execution (console.log(), variable assignments, function calls)
2	Microtasks (Higher Priority)	Promise.then(), queueMicrotask(), MutationObserver
3	UI Updates	Recalculate styles, layout, paint
4	Macrotasks (Lower Priority)	setTimeout(), setInterval() etc.
5	Rendering	requestAnimationFrame
6	I/O Callbacks	File system operations, network requests (AJAX, fetch API, WebSockets)
7	UI Events (Low Priority)	Click, scroll, keypress, mousemove, input events



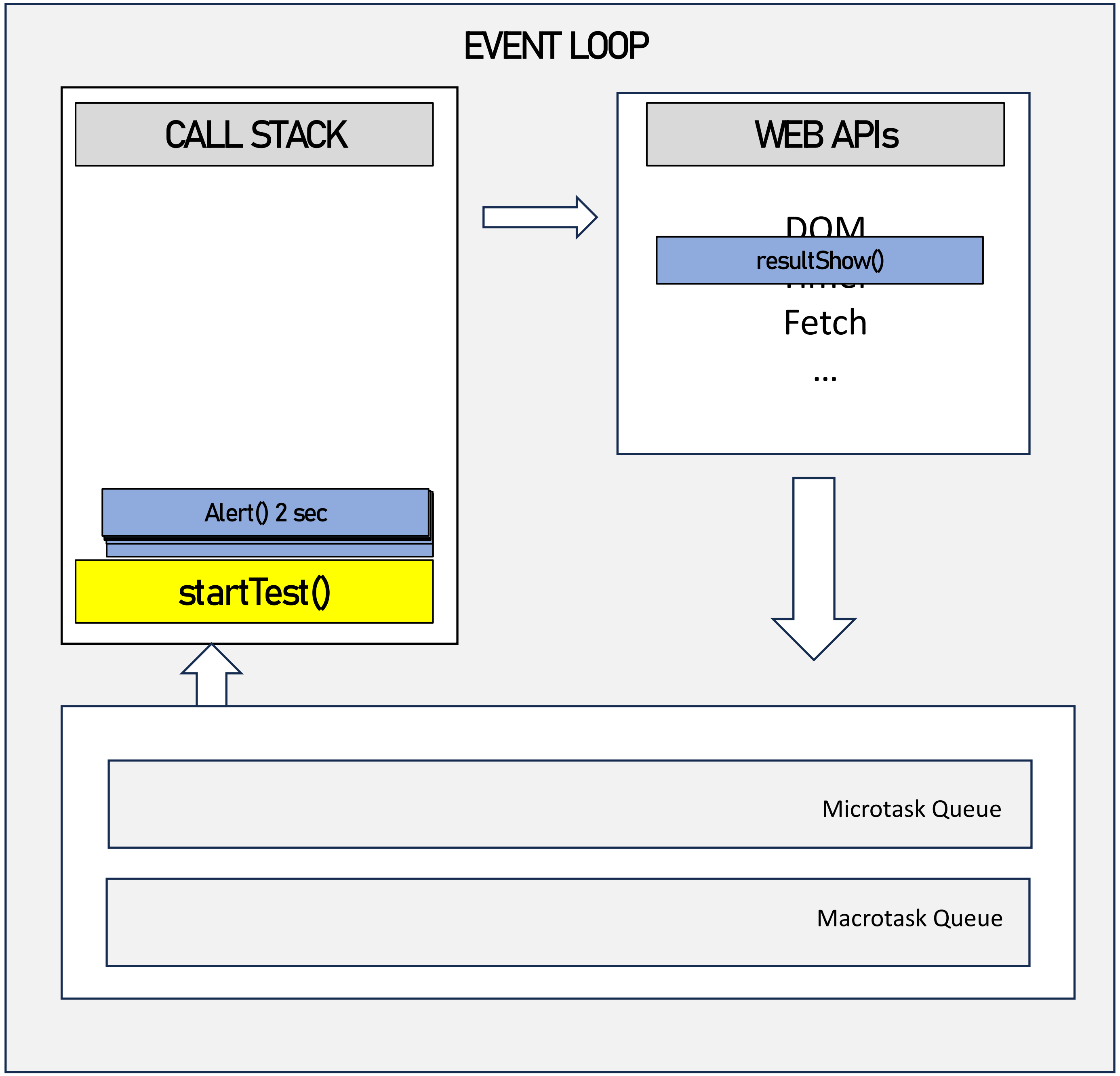
```
console.log("Start");

setTimeout(function a() {
  console.log("Macrotask: Timeout");
}, 0);

Promise.resolve().then(function b() {
  console.log("Microtask: Promise");
});

requestAnimationFrame(function c() {
  console.log("Rendering Task: Animation Frame");
});

console.log("End");
```

Console:

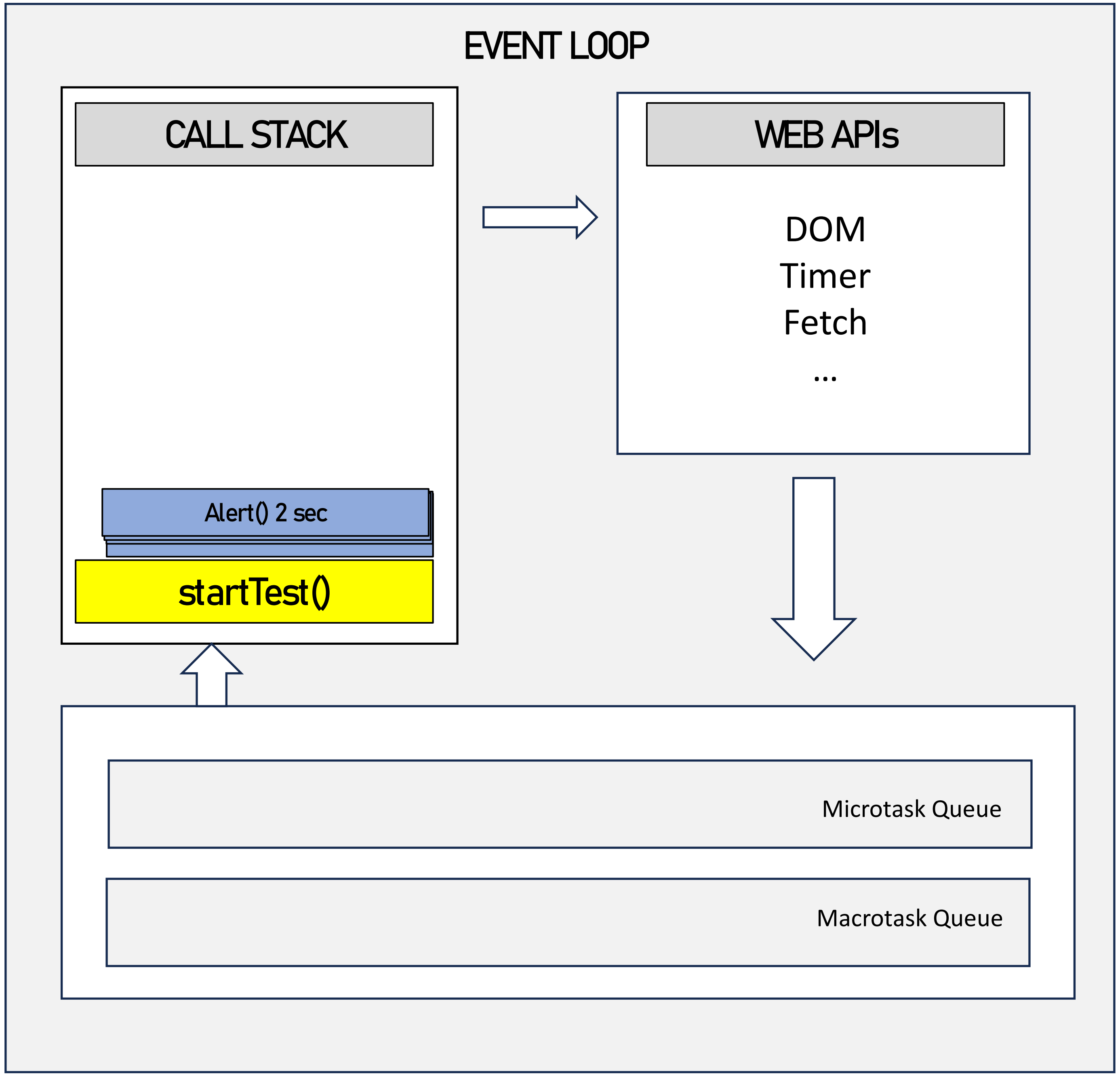
```
Start
End
Microtask: Promise
Macrotask: Timeout
Rendering Task: Animation Frame
```



```
● function startTest() {
    if (testActive) return;
    testActive = true;
    startTime = Date.now();
    clickCount = 0;

    while (Date.now() - startTime < 10000) {
      alert(`Alert ${clickCount + 1}`);
      clickCount++;
    }

    const average = (clickCount / 10).toFixed(2);

    startTimeDisplay.textContent = new
           Date(startTime).toLocaleTimeString();
    averageDisplay.textContent = average;

    const resultsStart = Date.now();
    while (Date.now() - resultsStart < 2000) {
      alert(`Average speed: ${average} alerts/sec`);
    }

    testActive = false;
  }
```



```
● function startTest() {
    if (testActive) return;
    testActive = true;

    startTime = Date.now();
    clickCount = 0;

    while (Date.now() - startTime < 10000) {
      alert(`Alert ${clickCount + 1}`);
      clickCount++;
    }

    const average = (clickCount / 10).toFixed(2);

    startTimeDisplay.textContent = new
  Date(startTime).toLocaleTimeString();
    averageDisplay.textContent = average;

    setTimeout(function resultShow() {
      const resultsStart = Date.now();
      while (Date.now() - resultsStart < 2000) {
        alert(`Average speed: ${average}
            alerts/sec`);
      }
      testActive = false;
    }, 0);
  }
```



```
● async function startTest() {
    if (testActive) return;
    testActive = true;

    startTime = Date.now();
    clickCount = 0;

    while (Date.now() - startTime < 2000) {
      alert(`Alert ${clickCount + 1}`);
      clickCount++;
    }

    const average = (clickCount / 10).toFixed(2);

    startTimeDisplay.textContent = new
  Date(startTime).toLocaleTimeString();
    averageDisplay.textContent = average;

    await new Promise((resolve) =>
  requestIdleCallback(resolve));

    const resultsStart = Date.now();
    while (Date.now() - resultsStart < 2000) {
      alert(`Average speed: ${average} alerts/sec`);
    }

    testActive = false;
  }
```

Something To remember

event loop works this way:

- execute everything from bottom to top from the stack, and ONLY when the stack is empty, check what is going on in queues above
- check micro stack and execute everything there (if required) with help of stack, one micro-task after another until the microtask queue is empty or don't require any execution and ONLY then check the macro stack
- check macro stack and execute everything there (if required) with help of the stack

Micro stack won't be touched if the stack isn't empty. The macro stack won't be touched if the micro stack isn't empty OR does not require any execution.

https://stackoverflow.com/questions/25915634/difference-between-microtask-and-macrotask-within-an-event-loop-context

Guess The output

```
console.log('stack [1]');
setTimeout(() => console.log("macro [2]"), 0);
setTimeout(() => console.log("macro [3]"), 1);

const p = Promise.resolve();
p.then(() => {
    setTimeout(() => {
        console.log('stack [4]')
        setTimeout(() => console.log("macro [5]"), 0);
        p.then(() => console.log('micro [6]'));
    }, 0);
    console.log("stack [7]");
});

console.log("macro [8]");
```

https://stackoverflow.com/questions/25915634/difference-between-microtask-and-macrotask-within-an-event-loop-context

Any
Question?