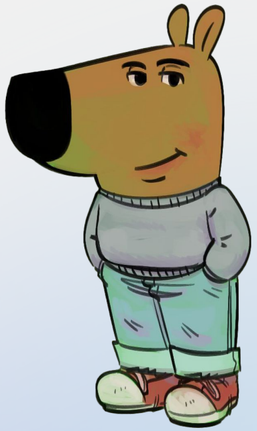
Abdullah S.



CSC309 Week 5

ASYNC & MIDTERM REVIEW

HTTPS://WWW.CS.TORONTO.EDU/~KIANOOSH/COURSES/CSC309H5/

6 FEBRUARY, 2025

Please join the

Zoom for polls

Meeting Code:

555 377 5792

Password:

n/a

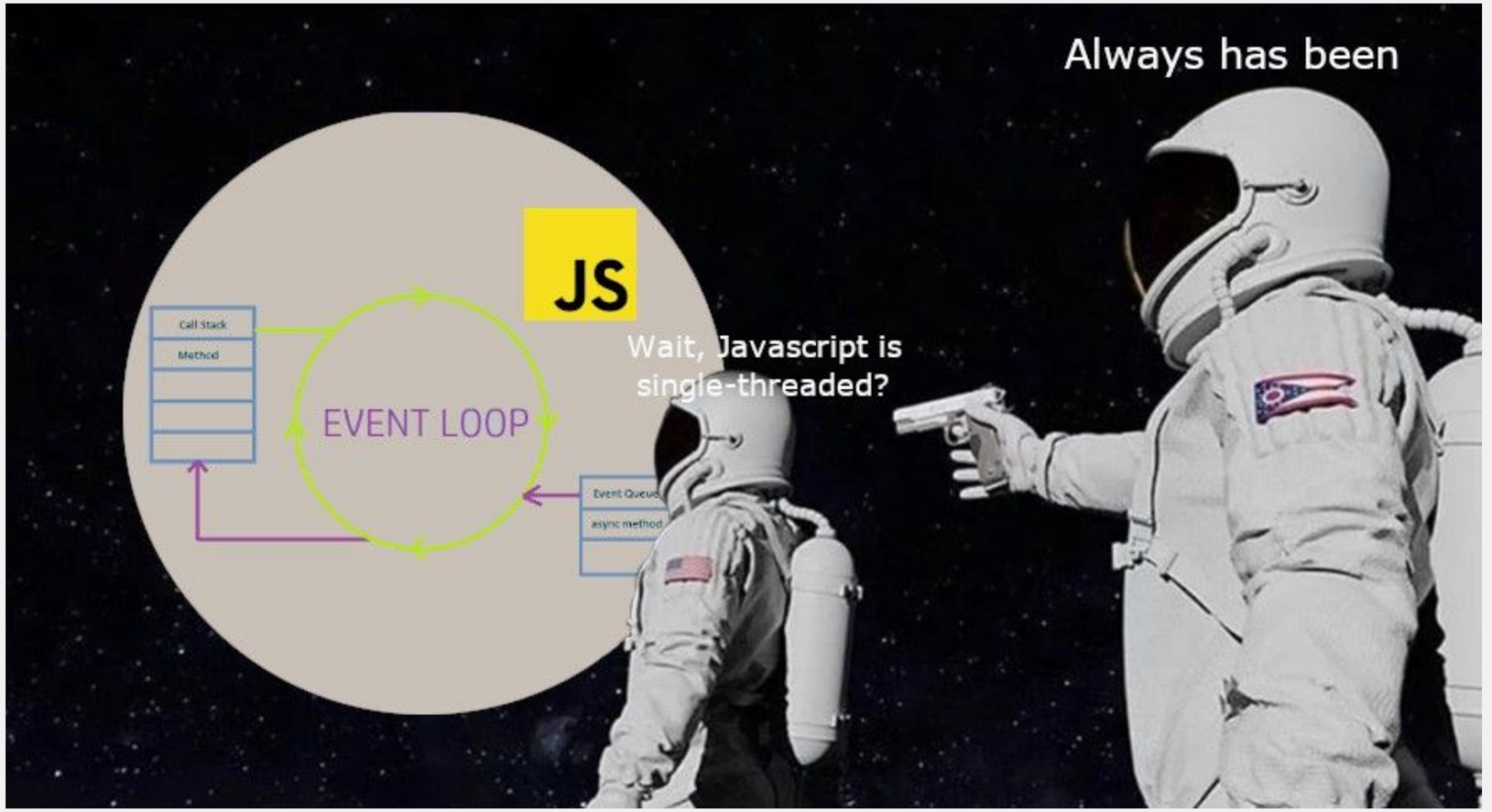
HTTPS://WWW.CS.TORONTO.EDU/~KIANOOSH/COURSES/CSC309H5/

Breakdown

Welcome to fifth tutorial of CSC309!

ASYNC	**Review + Deep dive into JS Runtime Model (15 minutes)**
MIDTERM REVIEW	**“Exam jam” via polling (25 minutes)**
Q/A	**Ask questions! (5-10 minutes)**

Async Programming in JS



Key point(s)

Javascript is a single threaded language.

- Libraries like NodeJS may provide additional libraries to provide real multithreaded capabilities but JS itself is single-threaded.

- So how does JS provide concurrency via a single thread?
  - …. the event loop

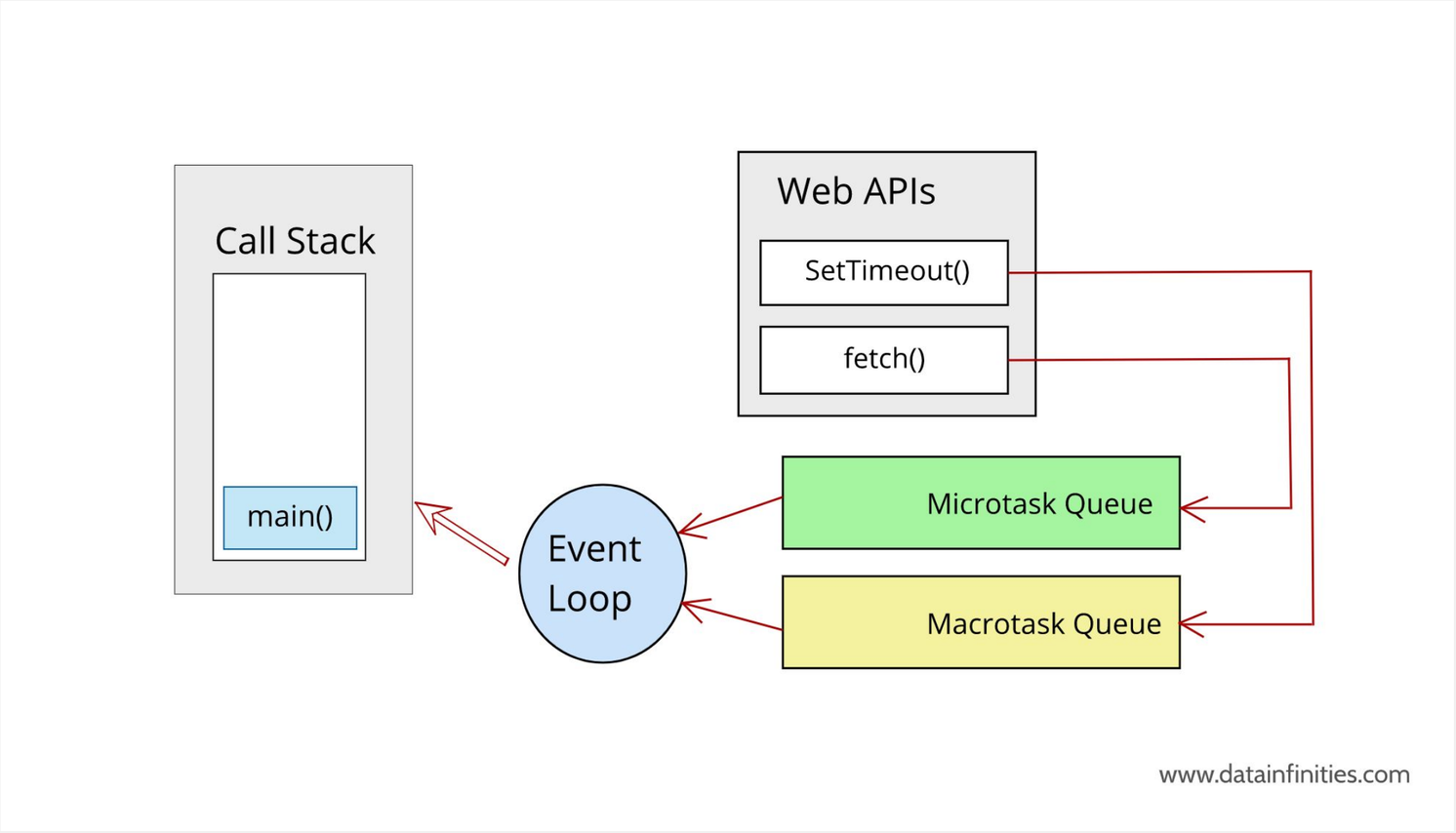
Async Programming in JS

- A lot of the work an API or a web application does is I/O bound. For example:
  - Waiting for user requests
  - Fetching data from a database
  - Writing logs to disk
  - Requesting data from other microservices

- So… there is a lot of waiting involved, if we were using a synchronous programming model, then we would be waiting for these task to complete.
  - And the entire app will have to wait for that…
  - Instead, we use the event loop to “wait” for events and come back to them when they’re finished. Meanwhile, our app can work on other tasks.

JavaScript Runtime Model

- Components:
  - Call Stack
  - Event Loop
  - Microtask Queue
  - Macrotask Queue



JavaScript Runtime Model: Call Stack

- The Call Stack is a LIFO data structure that keeps track of function calls in JavaScript
- It manages execution context for synchronous code
- When a function is called, it is pushed onto the stack
- When a function returns, it is popped off the stack.

Call Stack: Live Demo

https://www.jsv9000.app/

JavaScript Runtime Model: Event Loop

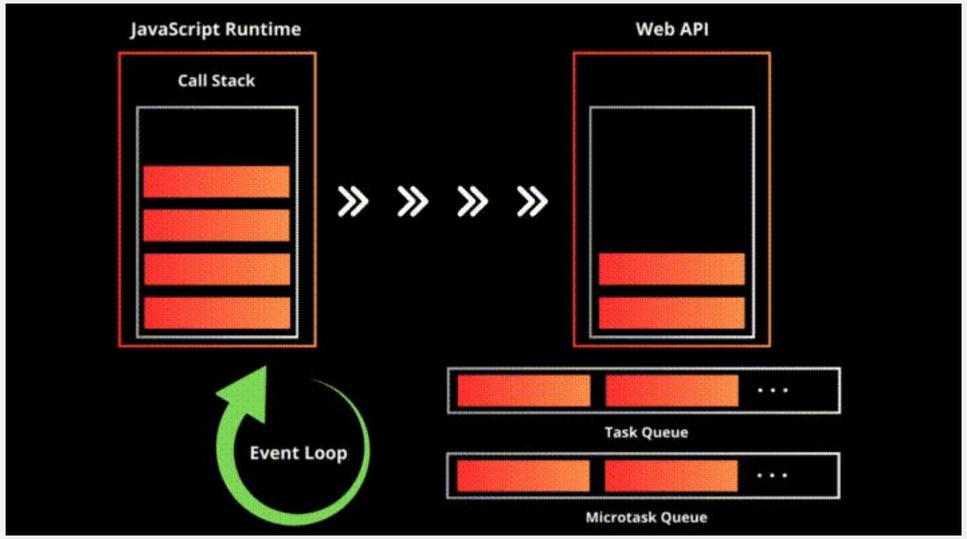
- JavaScript uses an event loop to manage asynchronous tasks.

- Tasks are handled in two separate queues:
  - Microtask Queue (Higher Priority)
  - Macrotask Queue (Lower Priority)

- Why should I care?
  - Understanding the difference helps optimize performance and give better insight into how your code works!

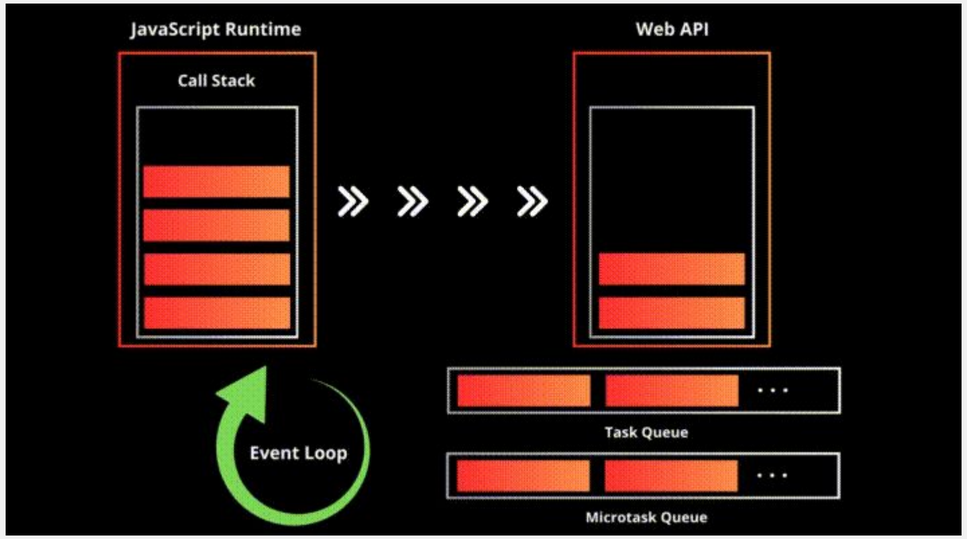
Microtask Queue

- Executes immediately after the current synchronous code.
- Higher priority than macrotasks.
- Runs before any macrotask is executed.
- Examples: Promises(.then(), .catch(), …), async/await, queueMicrotask()



Macrotask Queue

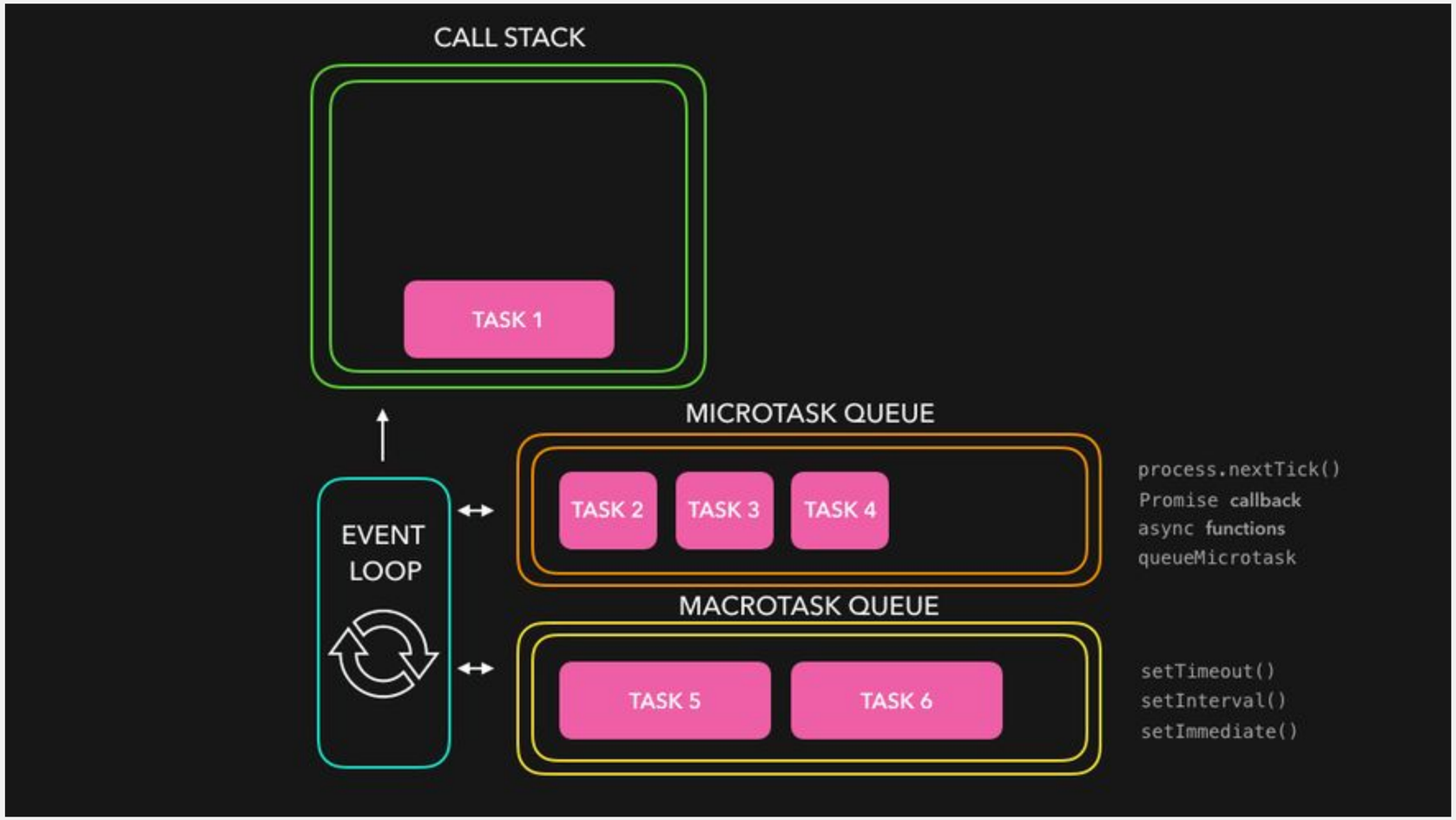
- Runs after the microtask queue is empty.
- Scheduled for the next event loop cycle.
- Used for lower-priority async operations.
- Examples: setTimeout(), setInterval(), fetch()



Summary: Microtasks vs Macrotasks

- Microtasks execute before macrotasks in the same event loop iteration.
- Microtasks are for promise resolutions and quick follow-up tasks.
- Macrotasks are for timers, I/O, and lower-priority async tasks.
- Understanding this helps in optimizing JavaScript performance.

Execution Order Visualized



Credit: @saravanaeswari22 (Medium)

Any questions?

Midterm Review (Polls)

ABDULLAH

Thank You

CSC309 Week 5

5 FEBRUARY, 2025