COESIONE ITALIA 21-27 PIEMONTE
Cofinanziato dall'Unione europea
REGIONE PIEMONTE

Web Developer

Programmazione - Javascript e Typescript

Docente: Shadi Lahham

Intervento realizzato da ITS TECNOLOGIE DELL'INFORMAZIONE E DELLA COMUNICAZIONE

come avvengono gli eventi!

Event loop

Concurrency model

Shadi Lahham - Web development

Stack, queue & heap

Queue

Queue (Coda): È una struttura FIFO (First In, First Out), dove il primo elemento che entra è il primo ad uscire.

La coda è usata per gestire eventi e operazioni asincrone, come i timer o le risposte di rete.

First in first out - FIFO

Queue
Items are used or processed in the order they arrive

© Shadi Lahham

FIFO

First In, First Out

- a method for managing inventory or data
- items or data that enter first are used or processed first
- commonly used in manufacturing, retail, and computing
- prevents newer items from being used before older ones

Stack

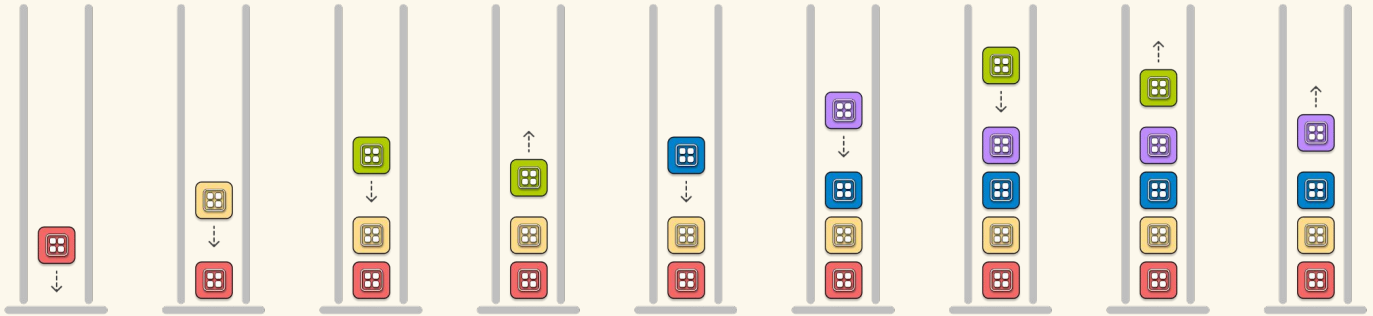
È una struttura di dati che segue il principio LIFO (Last In, First Out).

In JavaScript, lo stack gestisce le chiamate di funzione:
01. quando viene chiamata una funzione, questa viene inserita in cima allo stack
02. quando finisce, viene rimossa.

Questo permette di tenere traccia dell'ordine di esecuzione delle funzioni.

Last In, First Out - LIFO

Stack
Items are added to and removed from the top of the stack



© Shadi Lahham

LIFO

Last In, First Out

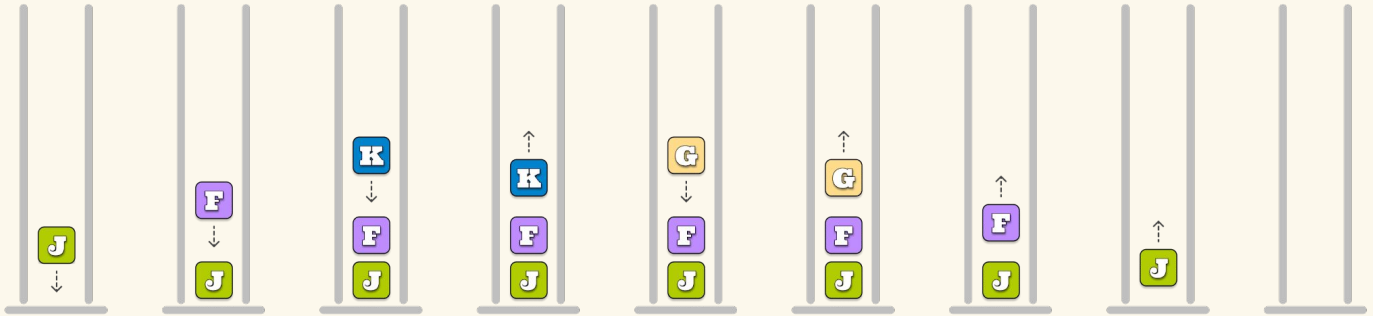
- a method for managing inventory or data
- newest items are used or processed first
- commonly used in accounting, storage systems, and computing
- ensures older items are used after newer ones

Javascript call stack

LA CALL STACK DI JS: serve a tenere traccia dell'ordine di esecuzione e traccia delle funzioni ancora attive

Call stack

JavaScript call stack
Manages function calls enabling
execution order and tracking of active functions



© Shadi Lahham

JavaScript call stack

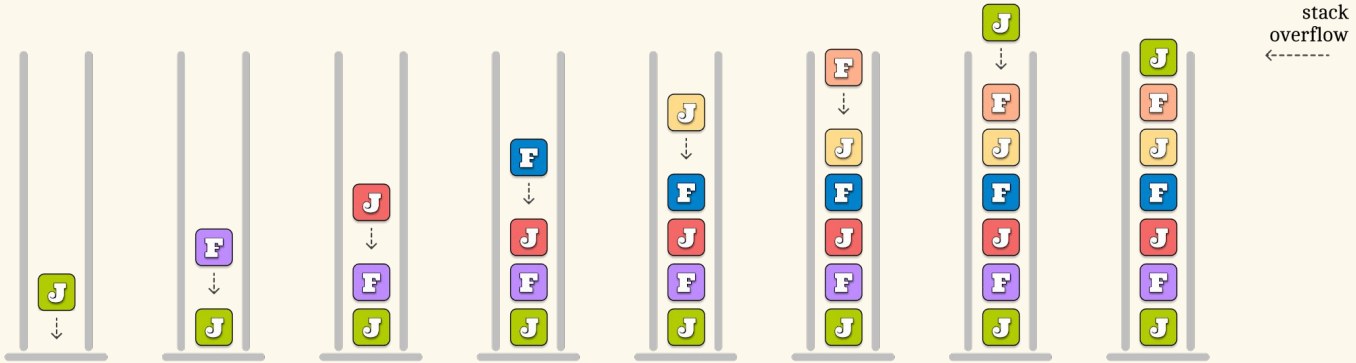
JS segue il modello del LIFO!!!

- manages function calls
- follows the Last In, First Out principle
- when a function is called, it's added to the top of the stack
- when a function returns, it's removed from the stack
- the stack helps track the order of function execution

Stack overflow

Call stack

Circular dependency
Functions continuously invoke each other without reaching an endpoint potentially resulting in a stack overflow error



© Shadi Lahham

Stack overflow

- occurs when the call stack exceeds its maximum size
- can happen due to infinite recursion or deeply nested function calls
- results in a runtime error and can crash the program
- can be prevented by optimizing code and avoiding
  - infinite recursion
  - circular dependency
  - deeply nested function calls

Recursion stack overflow limits

- Google Chrome: Around 10,000 to 17,000 function calls
- Mozilla Firefox: Around 10,000 to 20,000 function calls
- Safari: Around 10,000 to 15,000 function calls
- Microsoft Edge: Around 10,000 to 20,000 function calls

note

numbers valid for browser versions in 2022

The heap

Heap: È un'area di MEMORIA utilizzata per l'allocazione dinamica e intelligente degli oggetti.

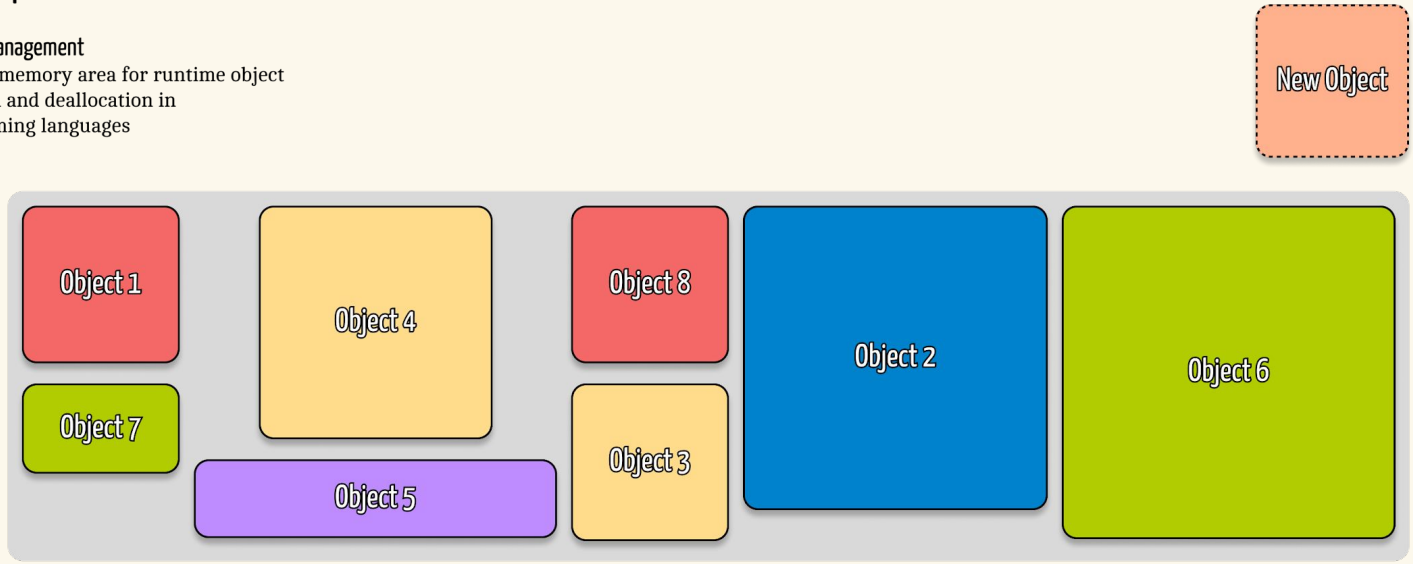
la memoria viene gestita tramite il GARBAGE COLLECTOR, che si occupa di liberare la memoria quando non è più necessaria.

- region of memory for runtime object allocation
- managed by the memory management system
- objects stored on the heap
- memory allocation and deallocation handled by the garbage collector
- enables dynamic memory management in managed languages
- different from the call stack used for function call management

The heap

The heap

Memory management
Dynamic memory area for runtime object allocation and deallocation in programming languages

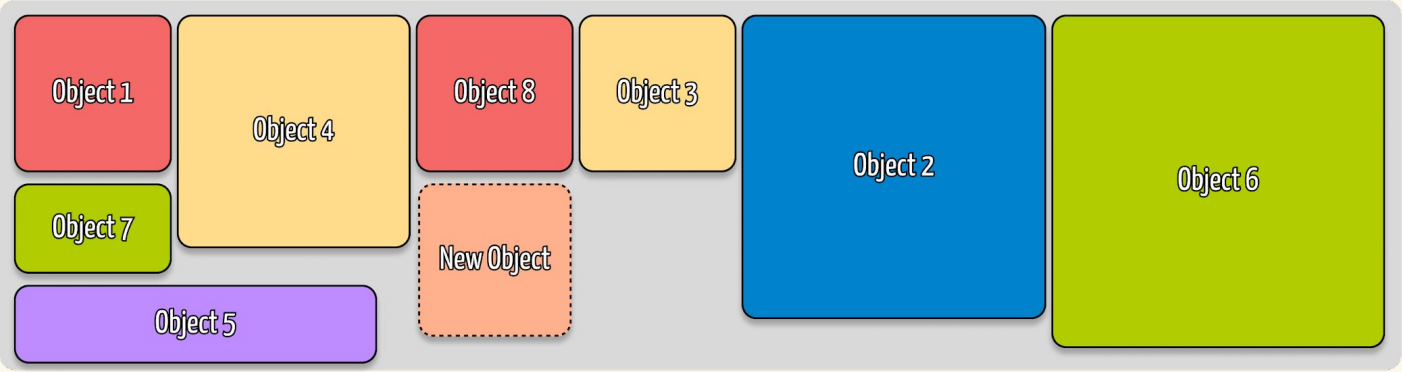


© Shadi Lahham

The heap

The heap

Heap fragmentation
Dispersed memory blocks hinder allocation despite sufficient total space availability



© Shadi Lahham

Heap management

Options when there's no contiguous space on the heap

- trigger garbage collection to free up memory
- expand heap by requesting additional memory from the operating system
- move already existing allocated memory to create contiguous space

Stack vs heap

1. Stack

- structured memory space for function calls and local variables
- fixed size determined by the operating system
- faster access due to its LIFO structure
- managed automatically by the compiler or runtime system
- suitable for managing predictable lifetimes of variables and function calls

Stack vs heap

2. Heap

- memory space for dynamic memory allocation and storage of objects
- dynamically grows and shrinks during program execution
- access speed might be slower due to dynamic allocation
- requires explicit allocation and deallocation
- suitable for dynamic size requirements and longer-lived objects

Event loop

Single and multi-threaded

Single-Threaded: JavaScript esegue il codice su un singolo thread, il che significa che può fare solo un'operazione alla volta. Tuttavia, grazie al modello di concorrenza, può simulare un comportamento simile al multithreading gestendo in modo asincrono operazioni come il caricamento dei dati o l'interazione con l'utente.

Single-threaded

- a process that executes tasks sequentially
- can only perform one operation at a time
- limited in utilizing system resources efficiently

Multithreaded

- a process can execute multiple tasks concurrently
- utilizes multiple threads for parallel execution
- allows for efficient resource utilization
- potential for faster execution of tasks

Multi-Threaded: Diversi thread possono eseguire più operazioni in parallelo, utilizzando al meglio le risorse di sistema. Anche se JavaScript è single-threaded, può comunque utilizzare il browser per gestire alcune operazioni in parallelo.

Event loop

L'Event Loop permette a JavaScript di gestire operazioni asincrone come:
- i timer (setTimeout, setInterval), le chiamate API, e le Promises.

Questo evita che JavaScript blocchi l'interfaccia utente mentre aspetta il completamento di un'operazione.

The event loop enables JavaScript to handle asynchronous operations and provide the illusion of being multithreaded despite being single-threaded

- being single-threaded, JavaScript can only perform one operation at a time
- examples of asynchronous operations that are handled by the event loop
  - DOM events for user input
  - fetching data with fetch()
  - promises, and async/await
  - timer events such as setTimeout() and setInterval()
- with asynchronous code, statements are not necessarily executed in the order they appear

Event loop and timer events

- timer events are triggered by setTimeout() and setInterval(), which schedule code execution after a specified time
- execution of Javascript continues with other tasks instead of waiting while the browser Web API keeps track of the timer and the callback function
- when a timer expires, the callback function is added to the "task queue"
- the event loop constantly monitors the queue
- when the main thread is idle and the call stack empty, the event loop gets tasks from the queue and runs their callback functions

Timer events example

La schedulazione delle funzioni nell'event loop:
è il meccanismo che gestisce l'ordine e il momento in cui il codice asincrono viene eseguito.

```
// define functions
let f = () => console.log('first');
let h = () => console.log('hi');
let b = () => console.log('bye');

// schedule calls
setTimeout(h, 0);
setTimeout(b, 400);
setTimeout(b, 1000);

// main
f();
```

SPIEGAZIONE DEL PROCESSO:

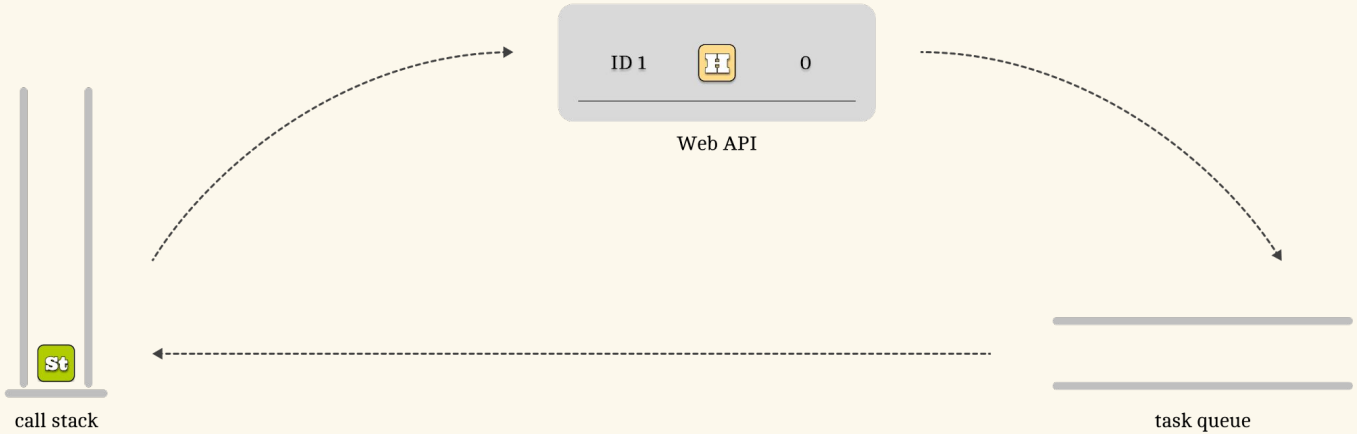
01. console.log('first') viene eseguito immediatamente.

02. setTimeout pianifica la callback, ma il programma continua senza aspettare eseguendo codice sincrono di "f()".

03. console.log('hi') viene eseguito subito dopo della callback function "h()".

04. Dopo 400ms / 1000 ms, la callback function "b()" di setTimeout viene inserita nella Task Queue e sarà eseguita solo quando il Call Stack sarà VUOTA.

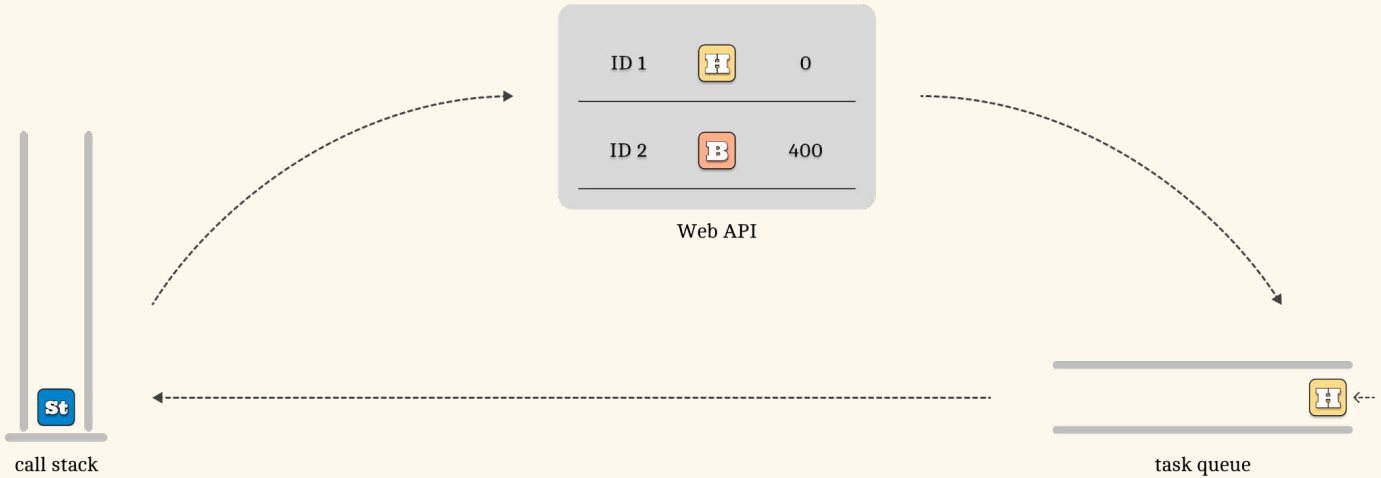
Timer events example



© Shadi Lahham

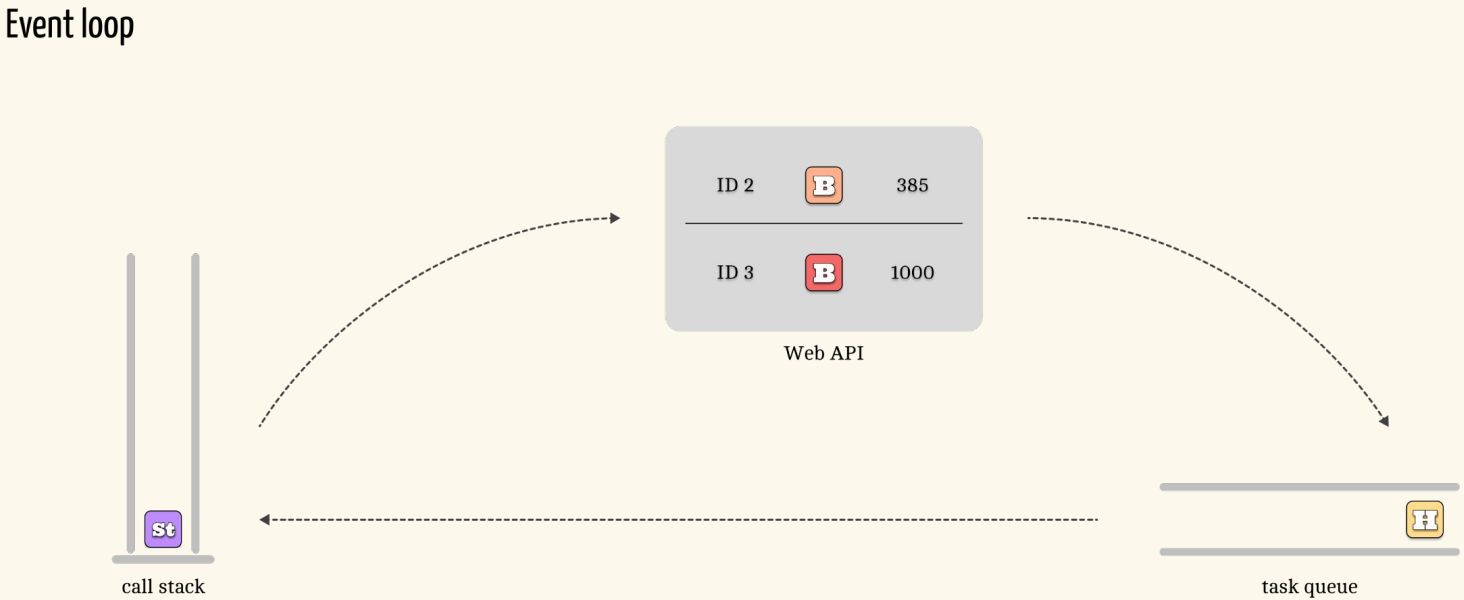
Timer events example

Event loop



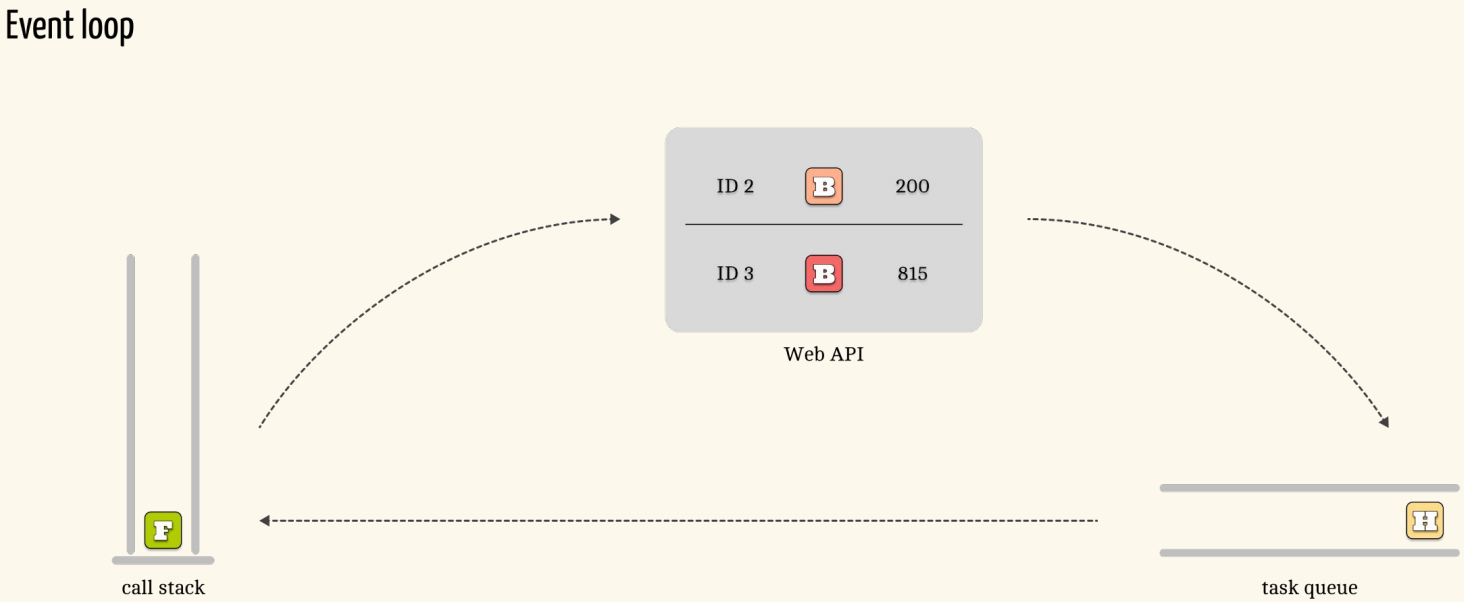
© Shadi Lahham

Timer events example



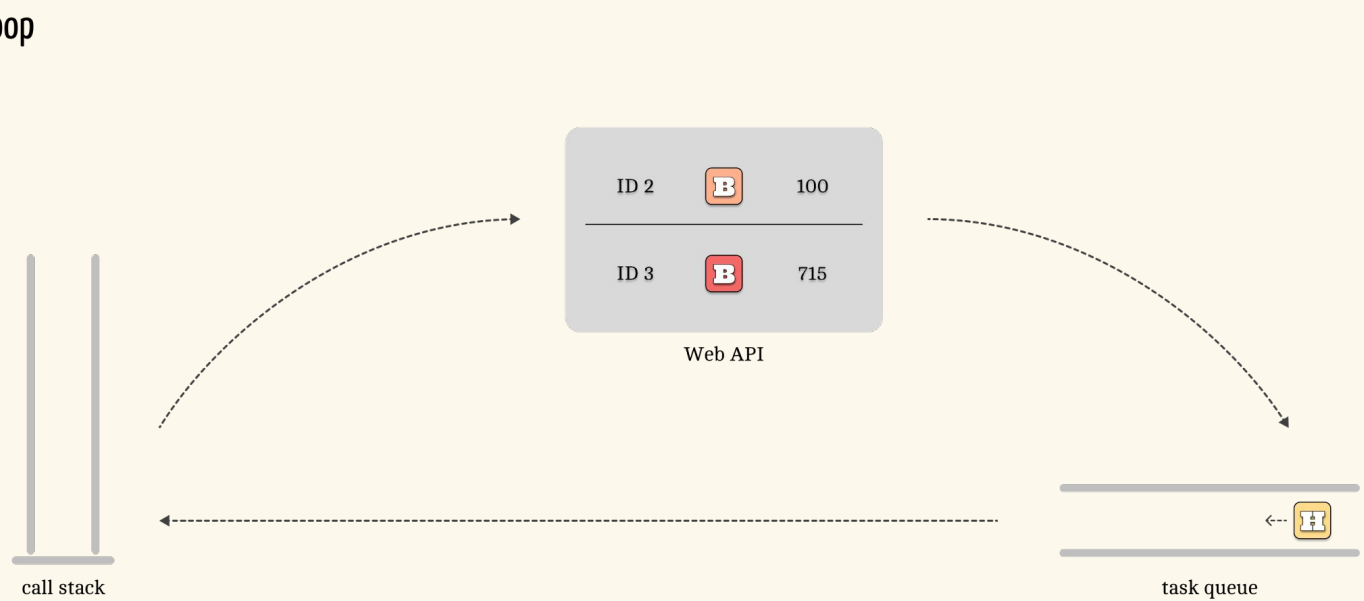
© Shadi Lahham

Timer events example



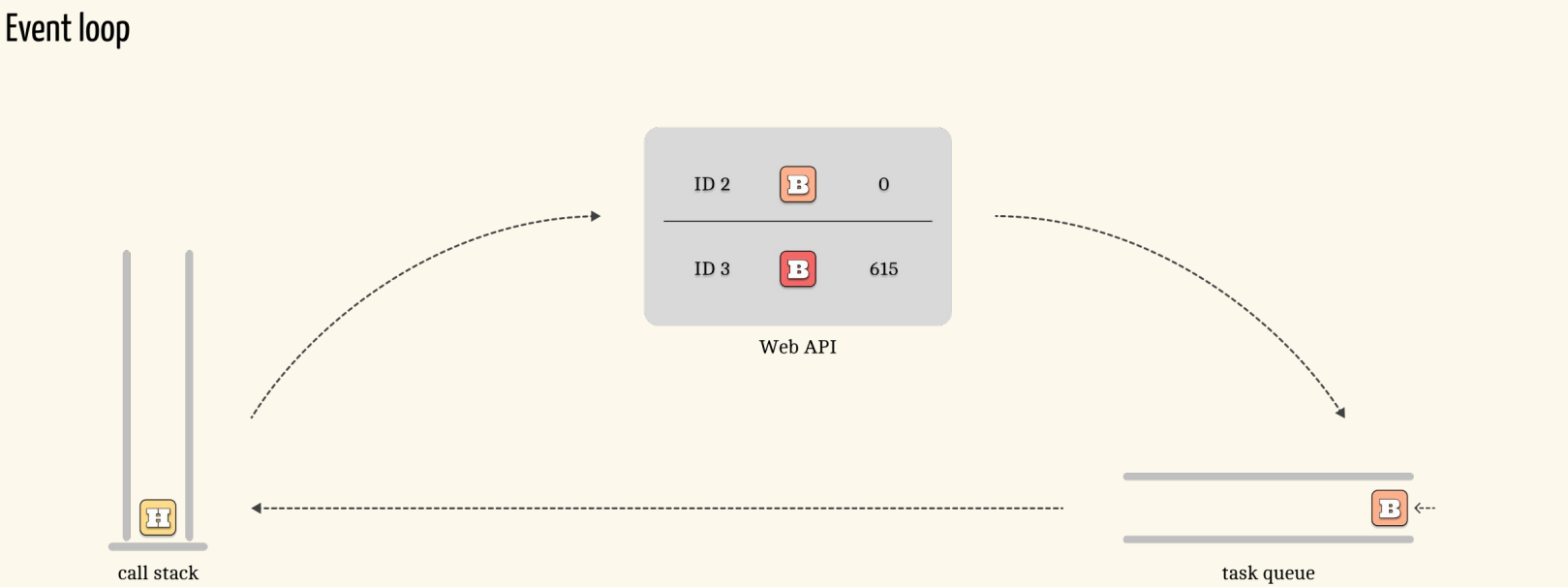
© Shadi Lahham

Timer events example



© Shadi Lahham

Timer events example



© Shadi Lahham

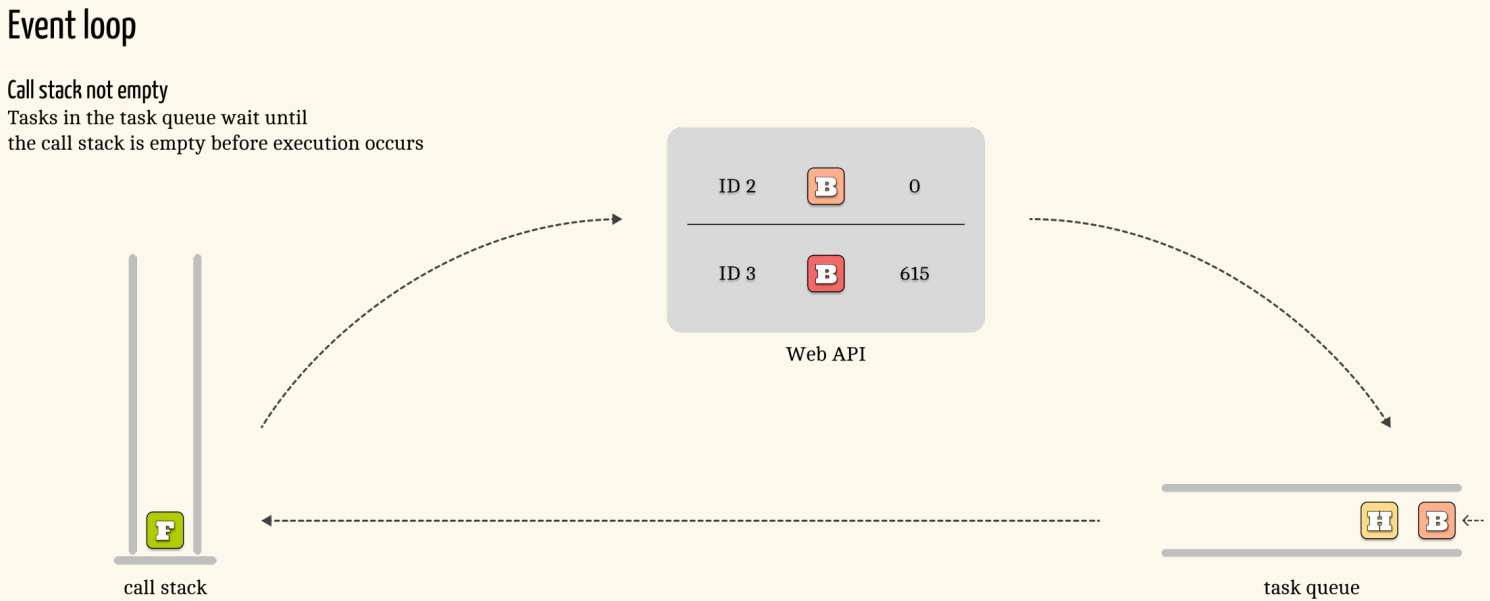
setTimeout minimum delay

setTimeout guarantees a minimum delay before executing the callback function, the actual delay may be longer due to factors such as the presence of other tasks in the task queue, the workload on the main thread, and system performance. This means that setTimeout does not provide an exact delay, but rather a minimum delay

Long function execution example

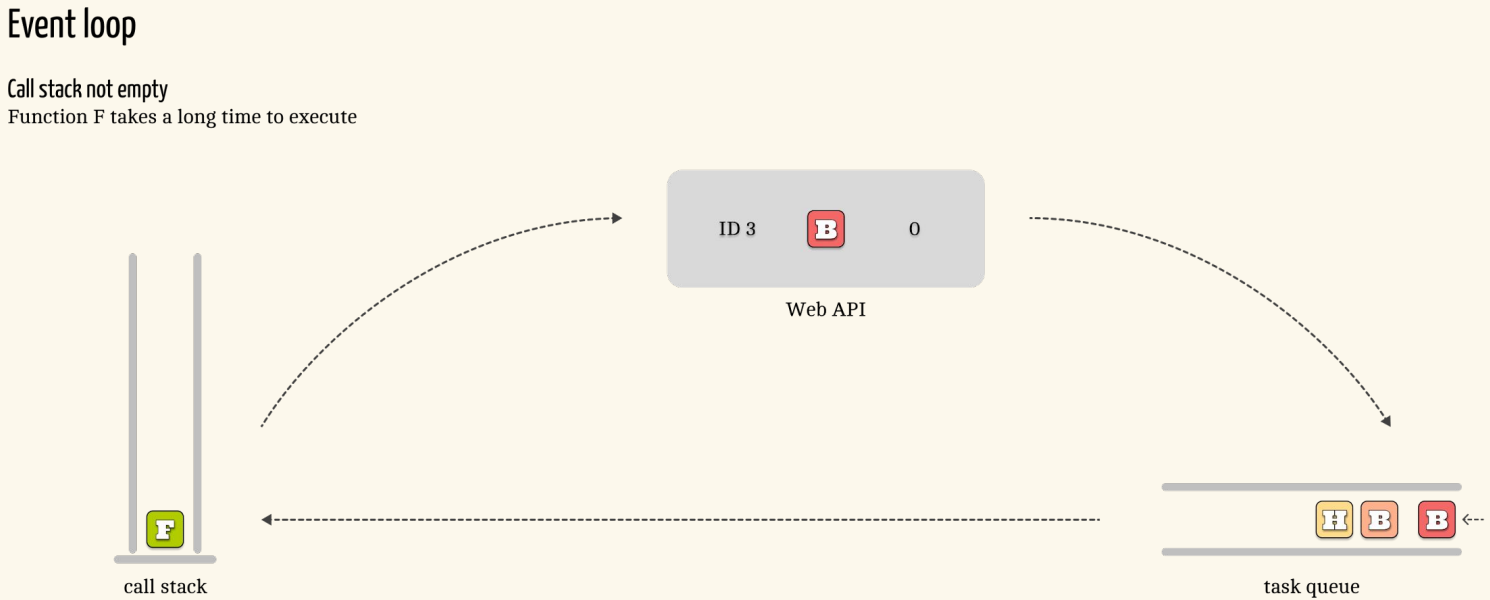
```
// function intentionally wastes time
let f = () => {
  let count = 0;
  // waste time
  while (count < 1e10) {
    count++;
  }
  console.log('first');
};
let h = () => console.log('hi');
let b = () => console.log('bye');

// schedule functions and invoke f
setTimeout(h, 0);
setTimeout(b, 400);
setTimeout(b, 1000);
f();
```

Long function execution example

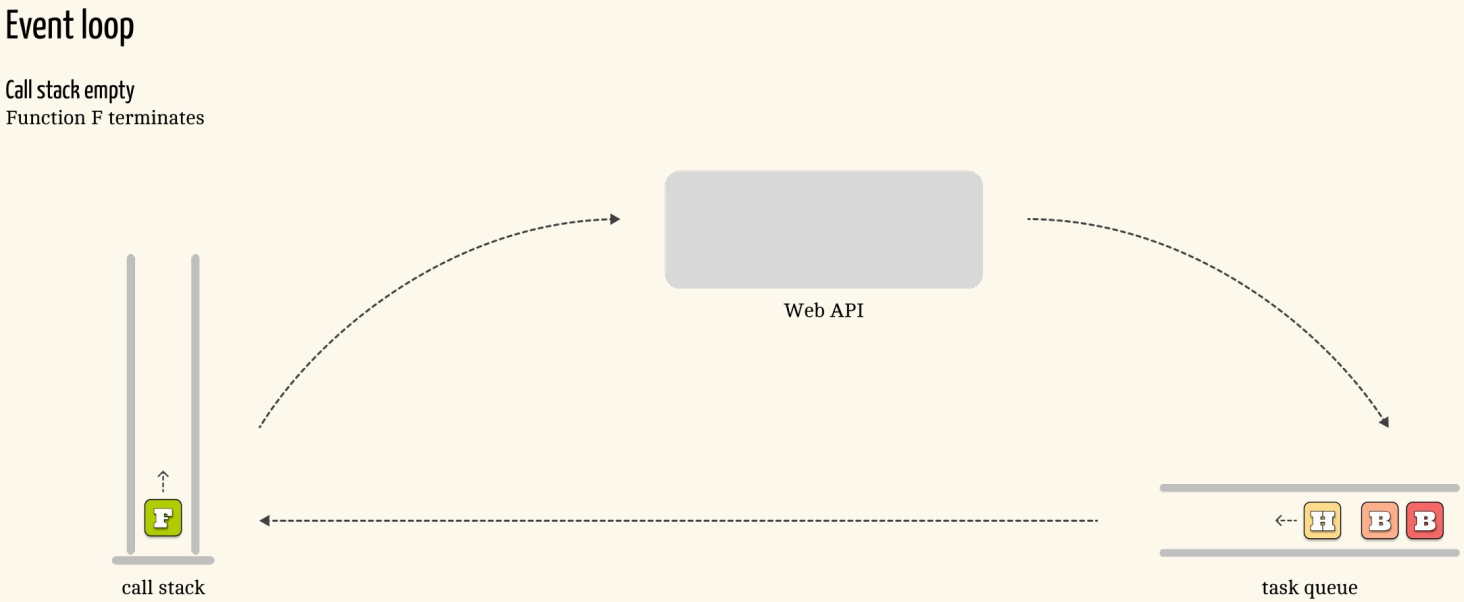


© Shadi Lahham

Long function execution example

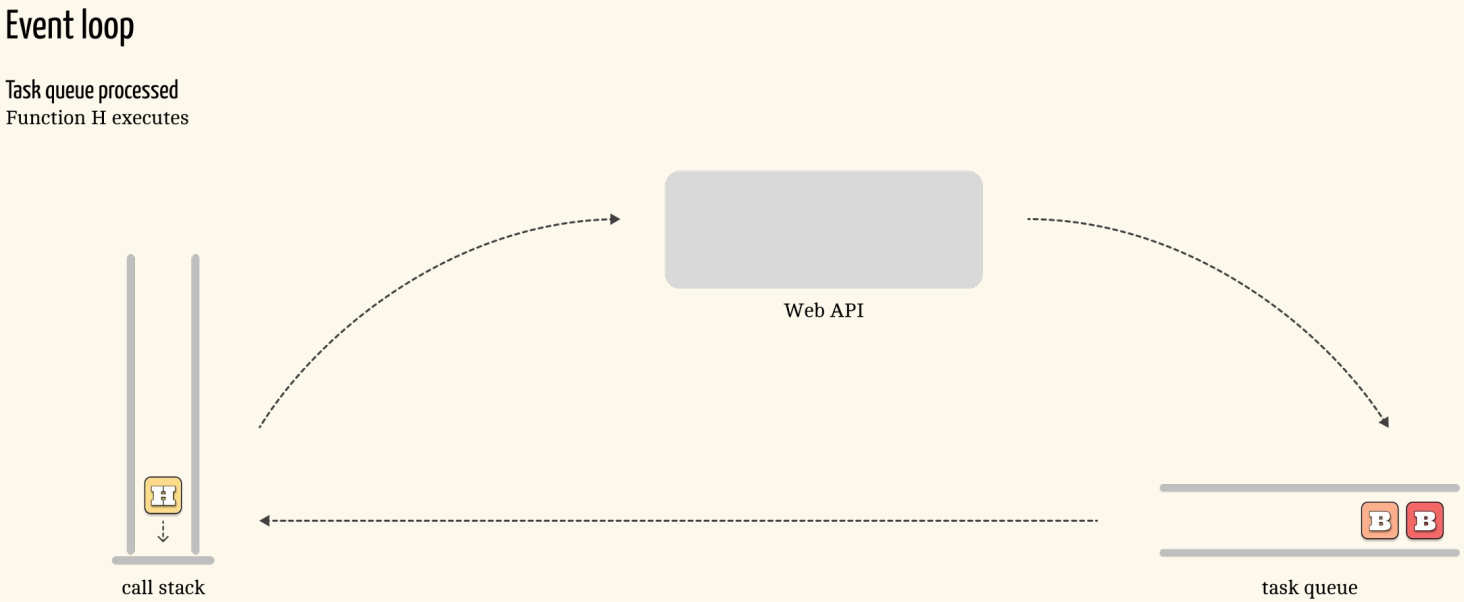


Long function execution example



© Shadi Lahham

Long function execution example



© Shadi Lahham

Your turn

1.The dishwasher

Create a simulation of a dishwasher system using two stacks of dishes

- one stack represents dirty dishes, and the other represents clean dishes
- the dirty stack has a random number of plates 10 - 50
- useful functions
  - washDish - moves a dish from the dirty stack to the clean stack
  - displayStacks - displays the current state of both stacks in the console
  - runSimulation - simulate washing all dirty dishes adding a random delay between steps

Bonus

1. have three stacks of dirty dishes and one clean stack
2. the dishwasher is able to wash two dishes at a time

Bonus

2.Double combo

Create a simulation of a turn-based combat system in a Dungeons and Dragons game using two queues of cards

- the card types are
  - characters, spells, or enemies
- create two queues: one for each player's cards
- fill each queue with N random cards of different types
- each card should be an object with appropriate
  - properties e.g. strength, defense, health
  - methods e.g. attack(enemy), buff(character), damage(character), duel(character)

2.Double combo

Turn-based Combat

- on each turn, draw one card from the start of each player's queue
- based on the type of cards drawn, certain combinations might trigger special actions or effects, such as:
  - Character vs. Enemy: Attack action, dealing damage to the enemy
  - Spell vs. Character: Buff action, boosting the character's attributes or Damage action, damaging or killing the character
  - Character vs. Character: Duel action, comparing attributes to determine the winner
  - Enemy vs. Enemy: No effect
- award points to each player based on the remaining health of their used character cards

References

Call stack - MDN

The event loop - MDN

Web API, Task Queue and Event Loop

Understanding event loop in JavaScript

Javascript Event Loop Explained