GUIDE TO DX

event loop

IN ECMASCRIPT

version 1.0
last updated 12/28/2022

Author: Milan Adamovsky

milan.adamovsky@gmail.com

CLEAN | DX

event loop

The event loop is the runtime model in ECMAScript

To understand the event loop
is to understand how ECMAScript
code is executed.

- code execution
- event handling
- memory management

Author: Milan Adamovsky

milan.adamovsky@gmail.com

CLEAN | DX
event loop

The event loop is an infinite loop
that runs for as long as there is
is an active execution context

Everything happens automatically ,
the developer needs to do nothing other
than understand the mechanism.

fundamental concept

low-level

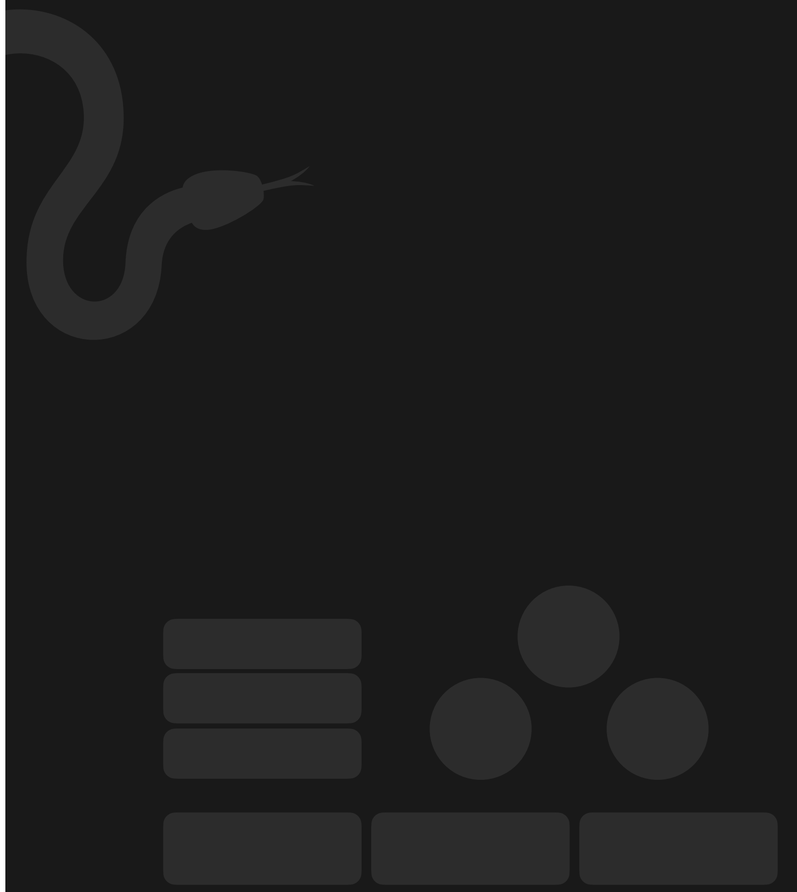
Author: Milan Adamovsky

milan.adamovsky@gmail.com

CLEAN | DX
event loop

Three parts make up the event loop:

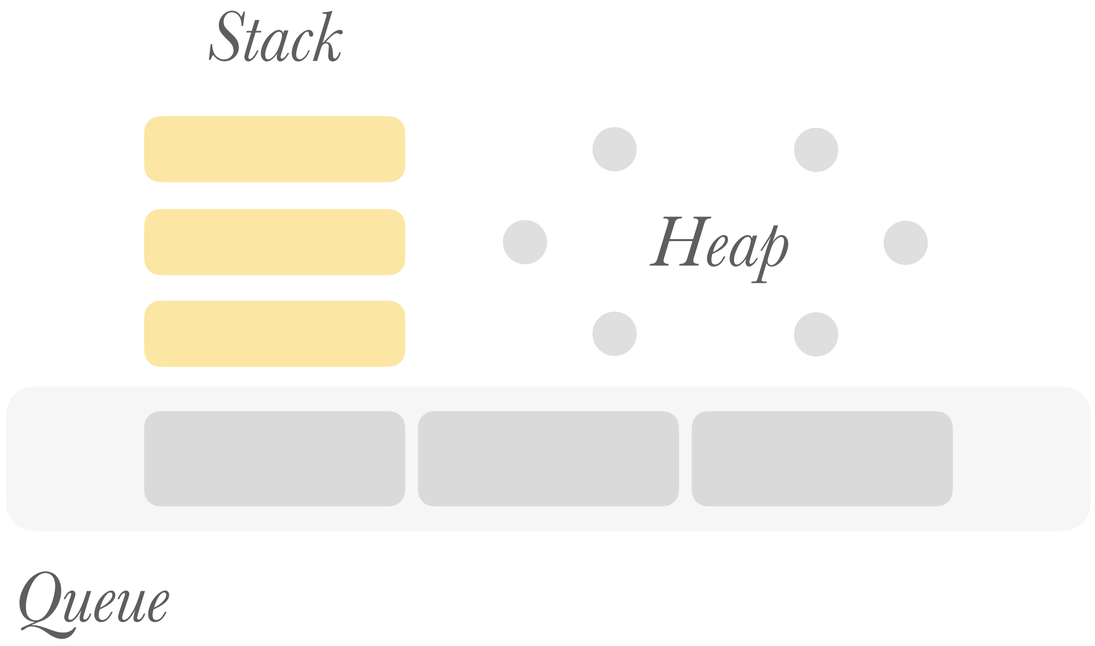
- Stack: contains all function calls
- Heap: contains all variables
- Queue: contains all messages



Author: Milan Adamovsky

milan.adamovsky@gmail.com

CLEAN | DX
event loop



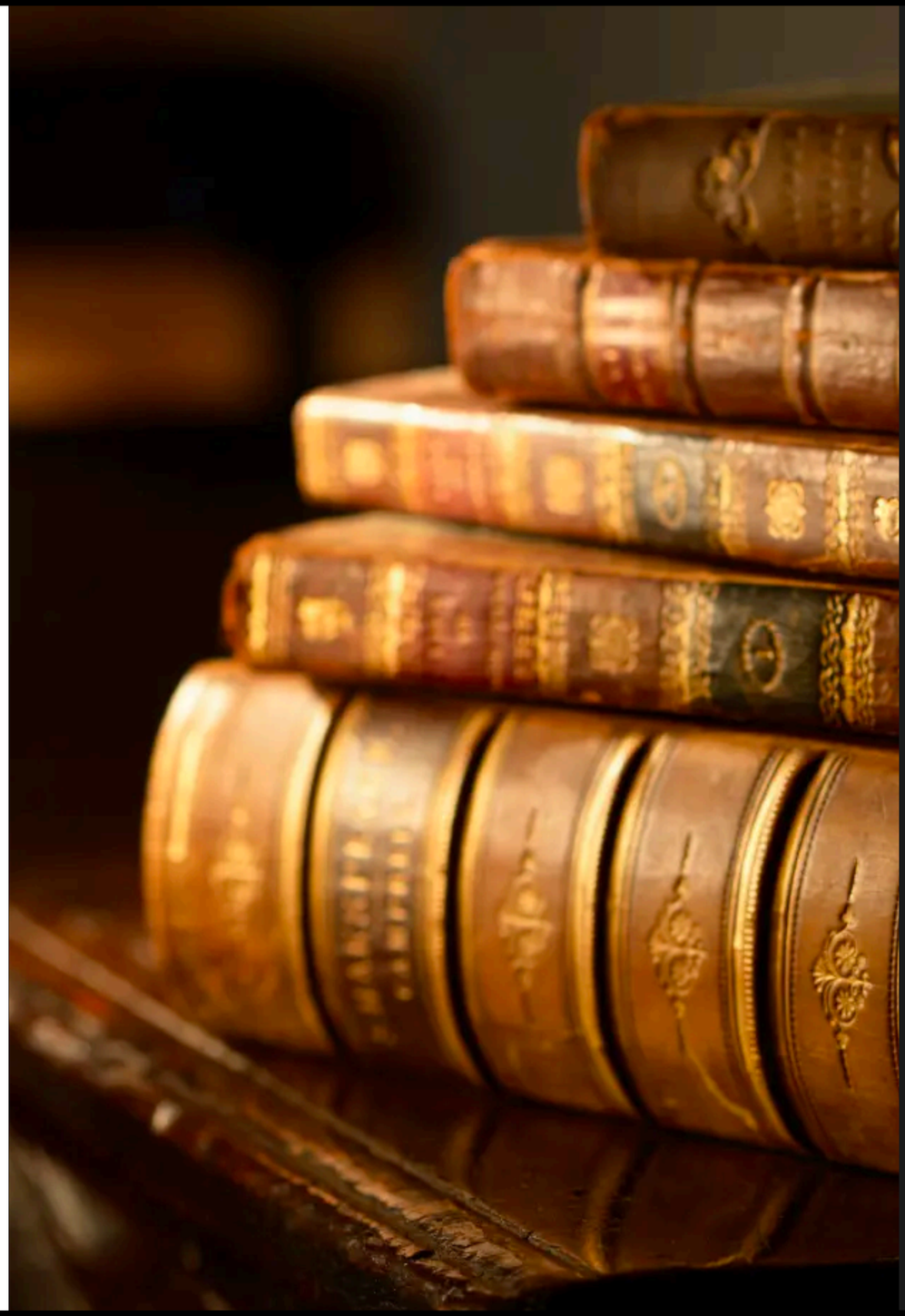
visualized

Author: Milan Adamovsky

milan.adamovsky@gmail.com

event loop

> The *stack* contains all the commands in the order of execution and is what we see in a stack trace



Author: Milan Adamovsky

milan.adamovsky@gmail.com

CLEAN | DX
event loop

We begin with the stack because it is what decides where to place what, and what order to execute code

It is managed by the Operating System that runs the ECMAScript engine.

FILO

very fast

call stack

—stack_size

Author: Milan Adamovsky

milan.adamovsky@gmail.com

event loop

There is only one stack per process,
since it is single-threaded

To achieve multi-threaded-like setup
we need to run multiple processes,
each with its own stack.

blocking

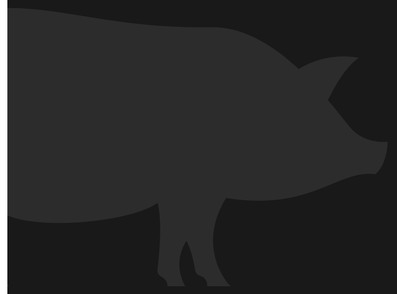
messaging

nodemon

Author: Milan Adamovsky

milan.adamovsky@gmail.com

event loop



The stack records function call sequence, thus call stack

It also serves as the memory store for primitive values such as booleans, strings, and numbers.

call sequence

memory lookup

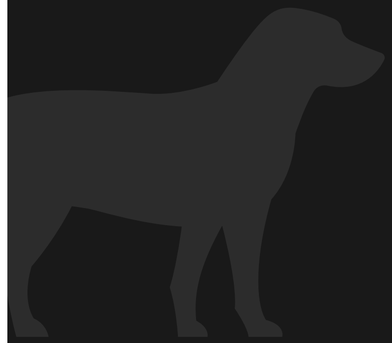
heap references

simple storage

Author: Milan Adamovsky

milan.adamovsky@gmail.com

CLEAN | DX
event loop



Pointers are recorded onto the stack
to non-primitive values

All pointers are by reference.

references

fast lookup

Author: Milan Adamovsky

milan.adamovsky@gmail.com

Since a pointer lives on the stack, we can conclude that the declaration keyword *const* protects the identifier's value that is registered on the stack

This also helps explain why a primitive value declared a *const* cannot be modified but an object's (referenced) value can.

immutability

copy by reference

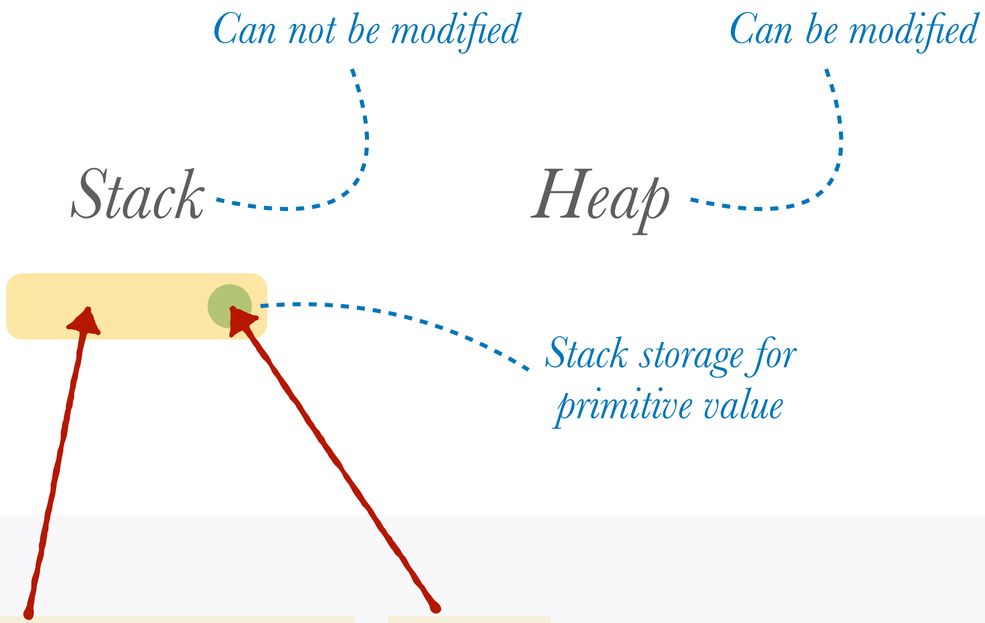
copy by value

Author: Milan Adamovsky

milan.adamovsky@gmail.com

CLEAN | DX

event loop



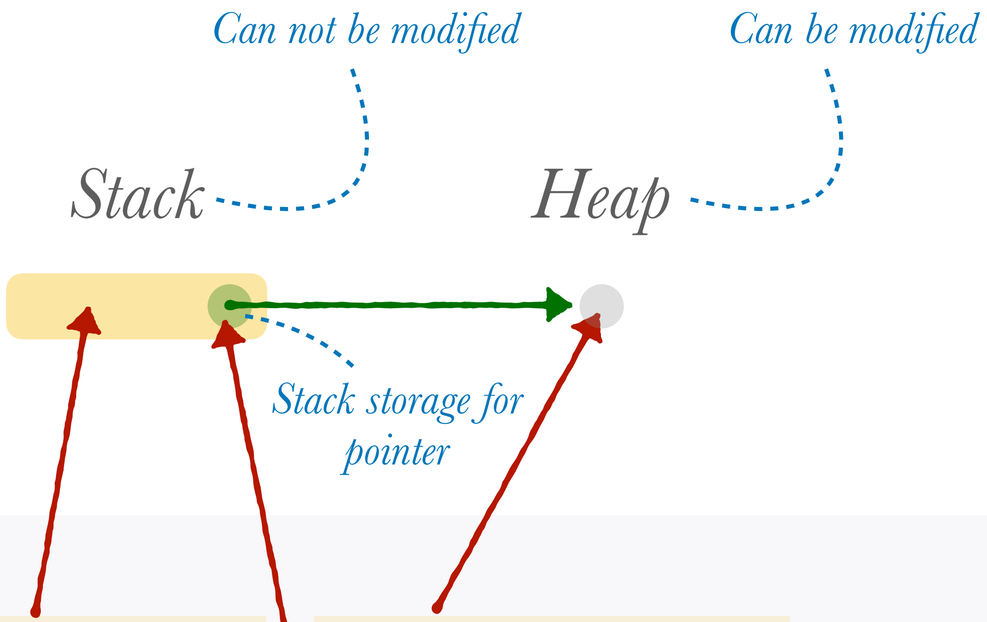
visualized

Author: Milan Adamovsky

milan.adamovsky@gmail.com

CLEAN | DX

event loop



visualized

Author: Milan Adamovsky

milan.adamovsky@gmail.com

event loop

Noteworthy is that anything stored in the global scope is kept in a global frame on the stack.

This explains why keeping things in the global scope is bad, since it is referenced for the code's lifetime.

memory footprint

system requirements

Author: Milan Adamovsky

milan.adamovsky@gmail.com

event loop

In summary, a function call gets added to the stack as a frame, and once it is done executing it gets popped off from the stack, moving on to the next frame in the stack.

FILO

Author: Milan Adamovsky

milan.adamovsky@gmail.com

CLEAN | DX
event loop

> The *heap* is the unstructured storage space where the event loop stores *most* of its data

Author: Milan Adamovsky

milan.adamovsky@gmail.com

event loop

The heap is unordered memory space where all the dynamic data such as variables and complex data structures live

It is managed by the engine that runs the ECMAScript code, such as V8.

unstructured

slower

garbage collection

Author: Milan Adamovsky

milan.adamovsky@gmail.com

CLEAN | DX
event loop

Garbage collection occurs in the heap

Each engine can implement its own garbage collection strategy, however they all implement a mark-and-sweep algorithm.

speed up

clean up

free up

optimizing

Author: Milan Adamovsky

milan.adamovsky@gmail.com

event loop

There are two memory regions in the heap memory space:

- Young generation
- Old generation

moving

performance

Author: Milan Adamovsky

milan.adamovsky@gmail.com

event loop

Think of the young generation memory space as the RAM space, therefore it is faster

Think of the old generation memory space as the ROM space, therefore it is slower.

lookup

Author: Milan Adamovsky

milan.adamovsky@gmail.com

CLEAN | DX

event loop

A minor garbage collection using a Scavenge algorithm occur in young generation memory region

A major garbage collection uses the better-known Mark-Sweep-Compact algorithm which replaces the more naïve reference-counting algorithm.

optimization

circular reference

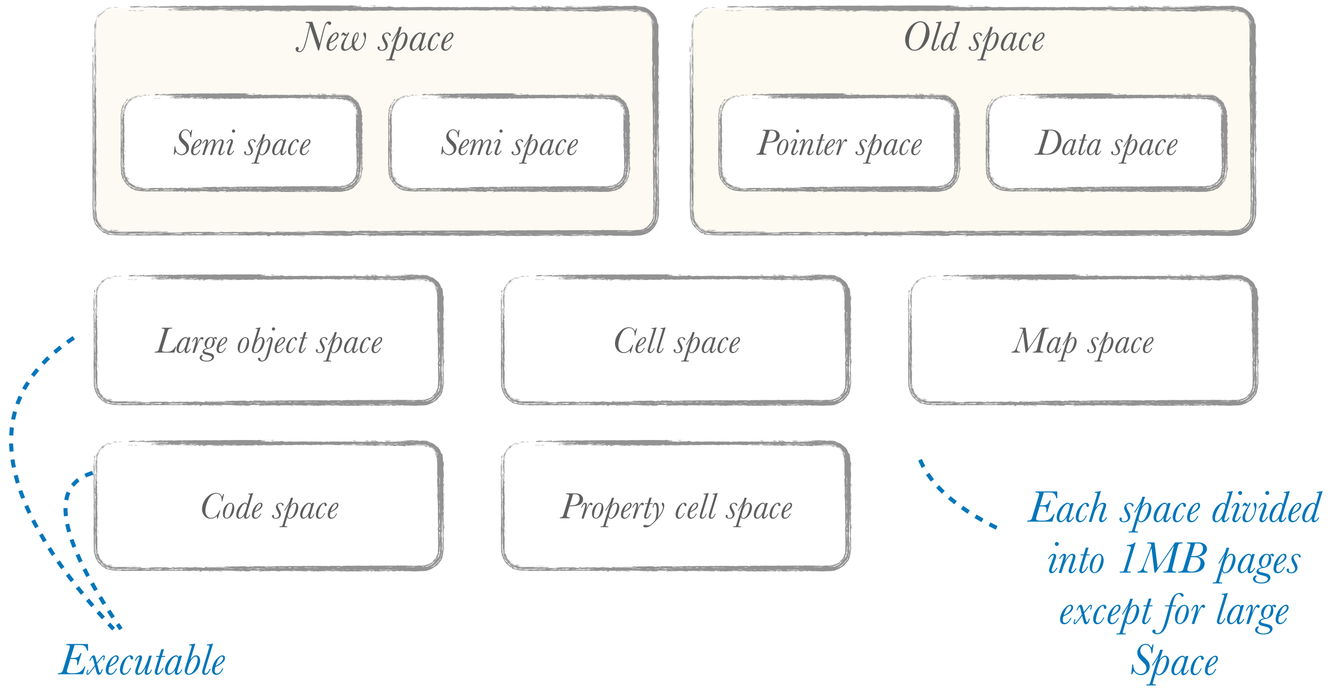
memory leak

Author: Milan Adamovsky

milan.adamovsky@gmail.com

event loop

The heap divides its data into spaces



JIT

—min_semi_space_size

—max_semi_space_size

—initial_old_space_size

—max_old_space_size

Author: Milan Adamovsky

milan.adamovsky@gmail.com

Most objects enter the new space of the heap, and anything that survives two garbage collection cycles gets moved to the old space.

marking

sweeping

Author: Milan Adamovsky

milan.adamovsky@gmail.com

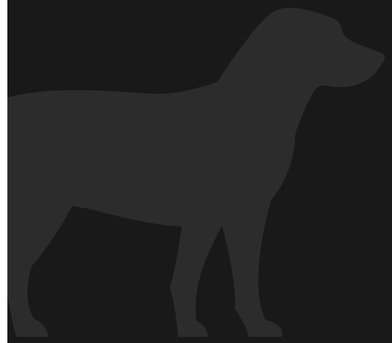
CLEAN | DX
event loop

> The *queue* is actually not a queue
> but a set since it grabs the first
> runnable task instead of
> *de-queuing* the first task

Author: Milan Adamovsky

milan.adamovsky@gmail.com

event loop



The queue is what tracks all asynchronously called functions such as callbacks and *events*

It gets called into action once the current stack is empty.

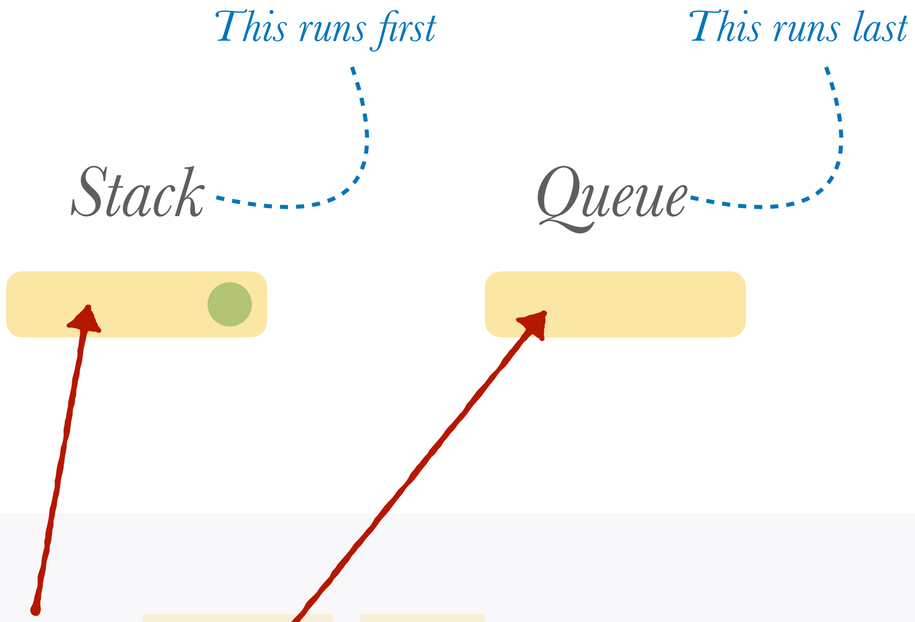
sequence

Author: Milan Adamovsky

milan.adamovsky@gmail.com

CLEAN | DX

event loop



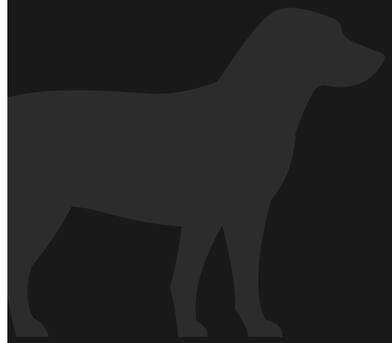

```
setTimeout(callback, 1000);
```

visualized

Author: Milan Adamovsky

milan.adamovsky@gmail.com

event loop



A function call with a callback gets added to the stack

The callback gets added to the queue

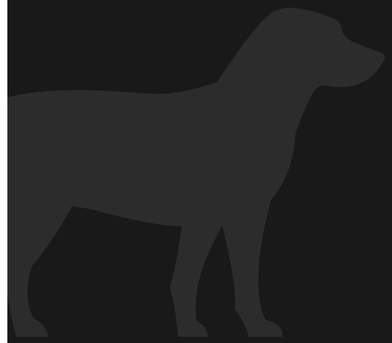
The callback only gets called after the stack is empty and any queue messages that are queued ahead.

sequence

Author: Milan Adamovsky

milan.adamovsky@gmail.com

event loop



Since anything on the queue relies on any preceding calls to complete, we are not guaranteed when our queue message will be processed

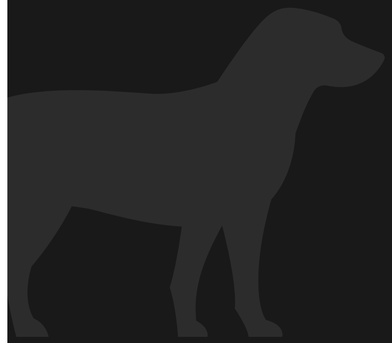
This is why the delay specified in functions like setTimeout is not a guaranteed but a minimum timing.

sequence

Author: Milan Adamovsky

milan.adamovsky@gmail.com

event loop



The queue as a whole is known as the message queue

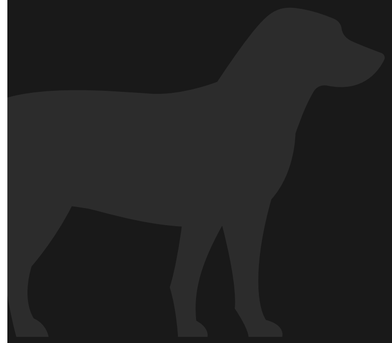
The message queue in turn is split up into multiple queues that bear different names and have different purposes depending on runtime context.

sequence

Author: Milan Adamovsky

milan.adamovsky@gmail.com

event loop



We have two purpose-specific queues:

- Macrotasks: used for *callbacks*
  aka *Callback queue*

- Microtasks: used for *thenables*
  aka *Job queue*

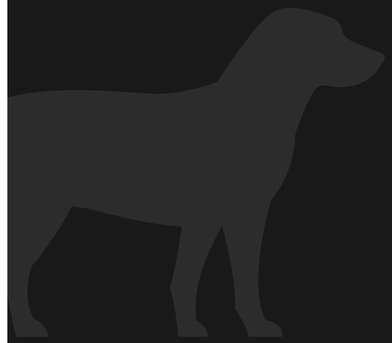
sequence

Author: Milan Adamovsky

milan.adamovsky@gmail.com

CLEAN | DX

event loop



Immediately after a macrotask is complete, the engine executes all microtasks before moving on to the next macro task

Event handling and rendering are suspended until all microtasks have completed.

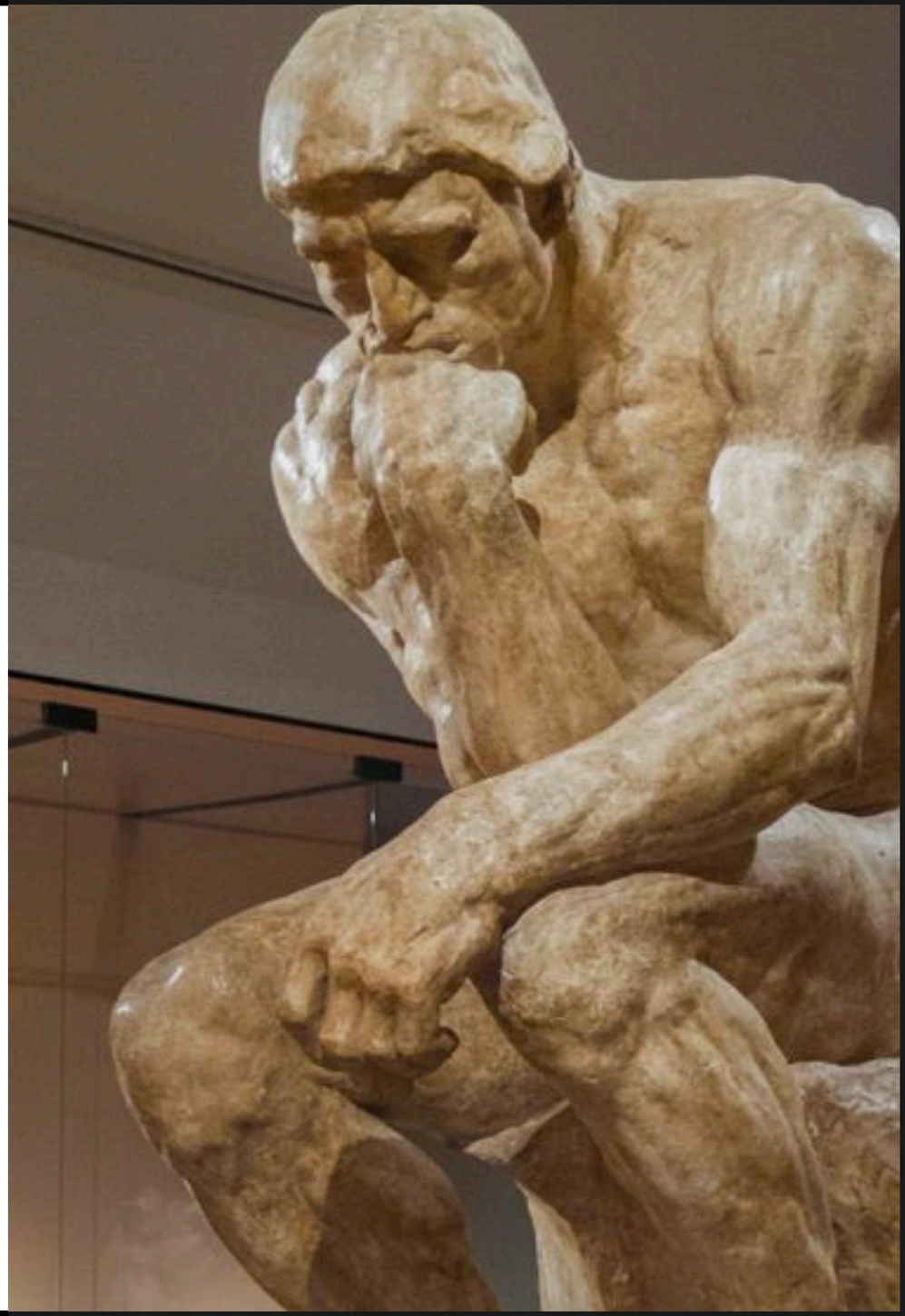
prioritization

Author: Milan Adamovsky

milan.adamovsky@gmail.com

CLEAN | DX
event loop

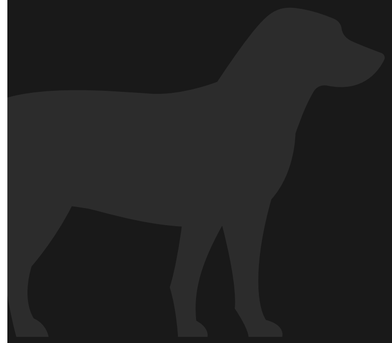
> While we do not need to actively think about the event loop, we can write better code with a basic understanding.



Author: Milan Adamovsky

milan.adamovsky@gmail.com

event loop



The event loop is non-blocking except for few exceptions like alert() and synchronous XHR

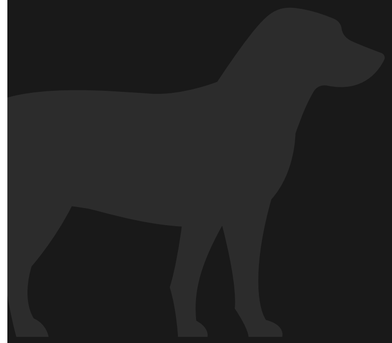
However, keep in mind that the garbage collector, when running, *is* blocking.

continuous

Author: Milan Adamovsky

milan.adamovsky@gmail.com

CLEAN | DX
event loop



The non-blocking aspect relates only to the execution of the code

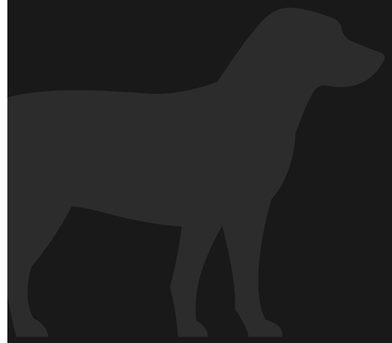
Rendering *is* blocked however while the queue is processed for example.

continuous

Author: Milan Adamovsky

milan.adamovsky@gmail.com

CLEAN | DX
event loop



A web-worker and cross-origin iframe have their own event stacks, heaps, and message queues

These separate runtimes can use the postMessage API to communicate with each other

isolation

Author: Milan Adamovsky

milan.adamovsky@gmail.com

CLEAN | DX
event loop

Milan has over 28 years of professional experience

Presently he focuses on front-end technologies, especially React and W3C Web Components

Contact him with any questions.

Author: Milan Adamovsky

milan.adamovsky@gmail.com

CLEAN | DX
event loop

Website coming soon…

www.cleandx.com

Author: Milan Adamovsky

milan.adamovsky@gmail.com