Async development introduction

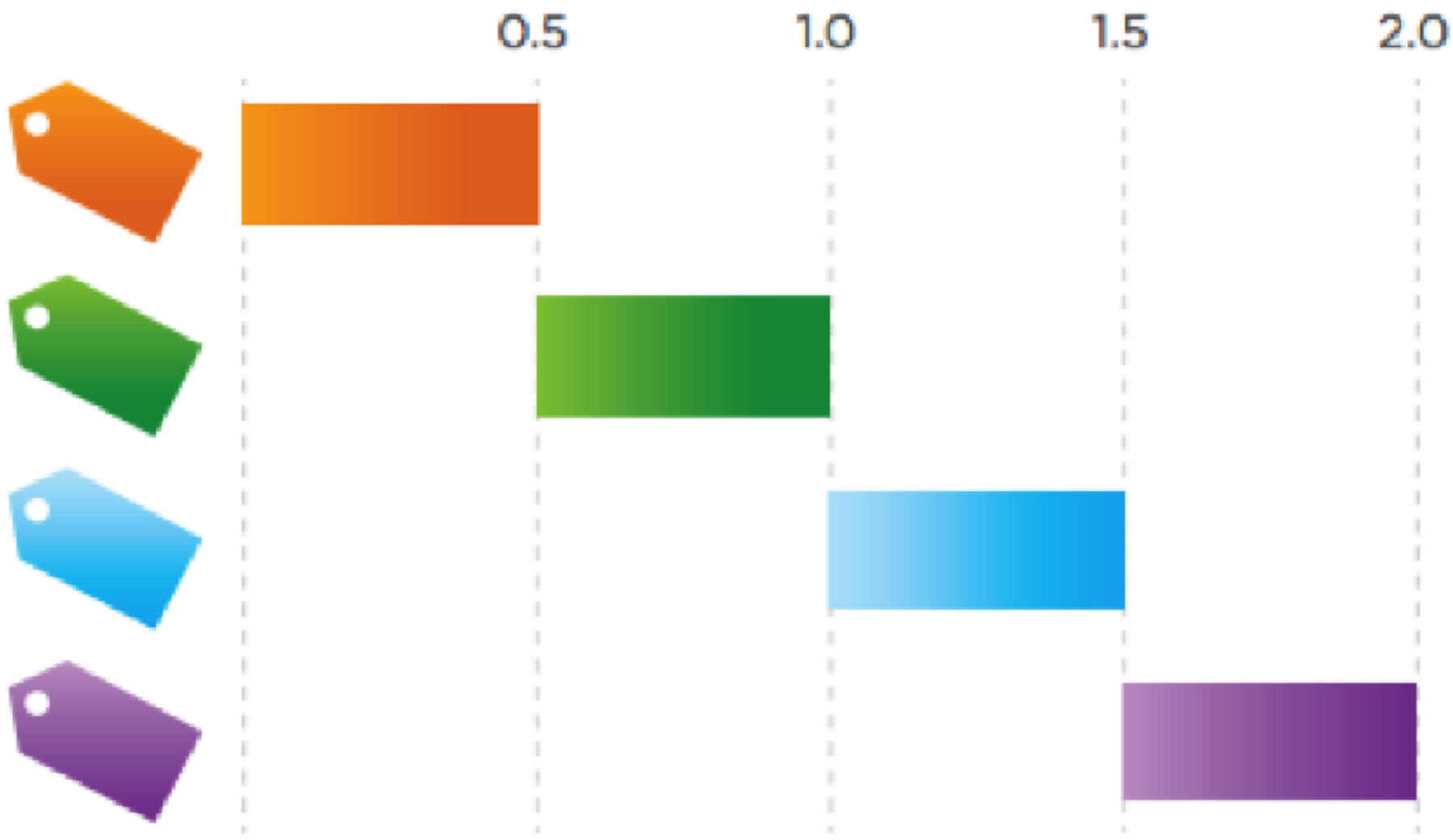
Agenda

- Async Functions
- Timers
- EventLoop and async operations
- process.nextTick()
- Async.js
- new in Node v10

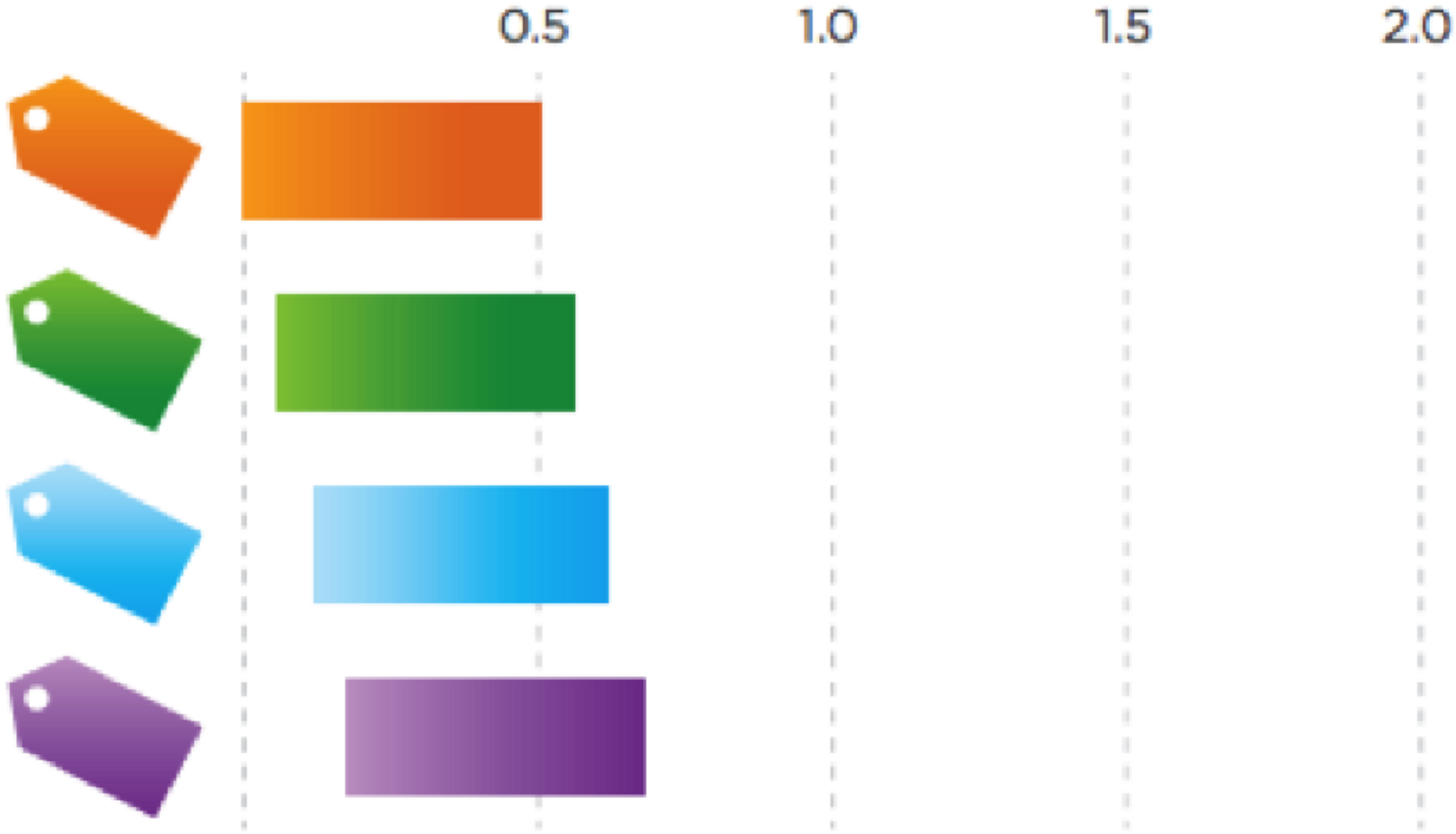
Async Functions

In computer science, **asynchronous I/O**, or "Non-sequential I/O" is a form of input/output processing that permits other processing to continue before the transmissionhas finished.

Sync



Async



When to use async functions

- FS tasks
- Events
- Streams
- DB operations

Async is not Multithread !

- Synchronous: you cook the eggs, then you cook the toast.
- Asynchronous, single threaded: you start the eggs cooking and set a timer. You start the toast cooking, and set a timer. While they are both cooking, you clean the kitchen. When the timers go off you take the eggs off the heat and the toast out of the toaster and serve them.
- Asynchronous, multithreaded: you hire two more cooks, one to cook eggs and one to cook toast. Now you have the problem of coordinating the cooks ...

Timers

setTimeout()	Executes a given function after a given time (in milliseconds).
setInterval()	Executes a given function at every given milliseconds.
setImmediate()	Executes a given function immediately.

setTimeout

setTimeout(*function, milliseconds, param1, param2, ...*)

Return Value:

A Number, representing the ID value of the timer that is set. Use this value with the clearTimeout() method to cancel the timer

setInterval

setInterval(*function, milliseconds, param1, param2, ...*)

Return Value:

A Number, representing the ID value of the timer that is set. Use this value with the clearInterval() method to cancel the timer

setImmediate

setImmediate(callback)

Returns an immediateObject for possible use with clearImmediate. Additional optional arguments may be passed to the callback.

ref/unref

unref()

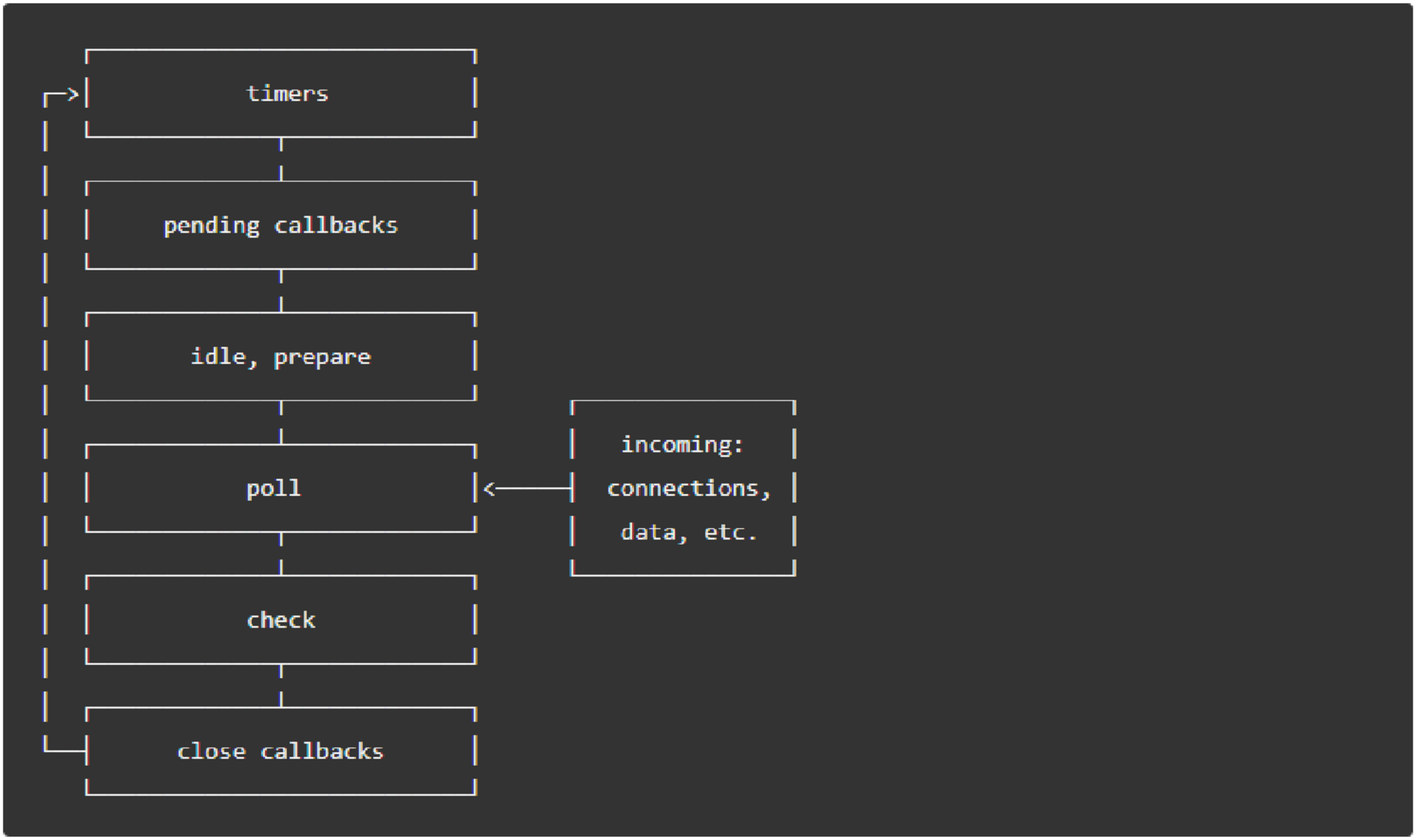
The opaque value returned by setTimeout and setInterval also has the method timer.unref() which allows the creation of a timer that is active but if it is the only item left in the event loop, it won't keep the program running. If the timer is already unrefd calling unref again will have no effect.

In the case of setTimeout, unref creates a separate timer that will wakeup the event loop, creating too many of these may adversely effect event loop performance -- use wisely.

Returns the timer.

Async operations and Event Loop

Phases


timers
pending callbacks
idle, prepare
poll
incoming:
connections,
data, etc.
check
close callbacks

Phases Overview

- **timers**: this phase executes callbacks scheduled by setTimeout() and setInterval().
- **pending** callbacks: executes I/O callbacks deferred to the next loop iteration.
- **idle, prepare**: only used internally.
- **poll**: retrieve new I/O events; execute I/O related callbacks (almost all with the exception of close callbacks, the ones scheduled by timers, and setImmediate()); node will block here when appropriate.
- **check**: setImmediate() callbacks are invoked here.
- **close** callbacks: some close callbacks, e.g. socket.on('close', ...).

Event-driven programming is a programming paradigm in which the flow of the program is determined by events such as user actions (mouse clicks, key presses), sensor outputs, or messages from other programs/threads.

Reasons to use nextTick()

- Allow users to handle errors, cleanup any then unneeded resources, or perhaps try the request again before the event loop continues.
- At times it's necessary to allow a callback to run after the call stack has unwound but before the event loop continues.


asynic

Whats new in Node.js 10

Q & A