Week- 4

Backend Engineering Launchpad

Pawan Panjwani

Airtribe

Advanced Node Js Event Loop and Clustering

Airtribe

Agenda

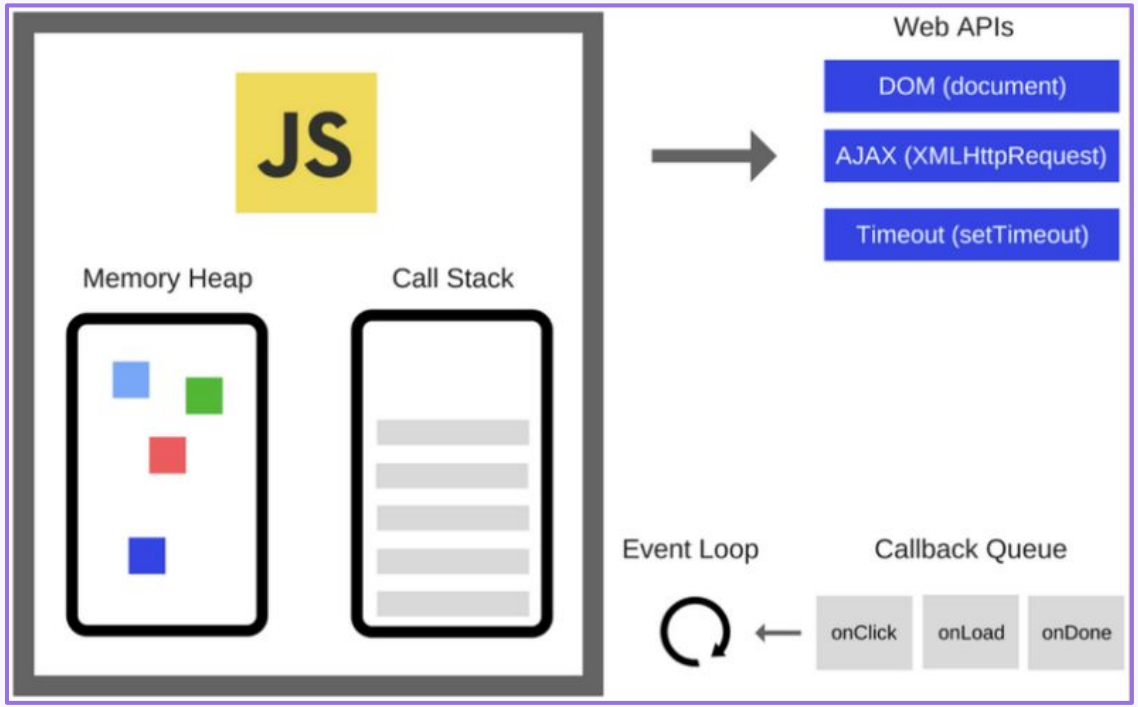
- Understanding event loop
- Understanding call stack and callback queue
- Understanding web API's
- Understanding microtasks and macrotasks
- Understanding order of execution in node js event loop
- Node js event loop blocking
- Clustering
- Question and answers

Airtribe

Event loop demystified

- The event loop is what allows Node.js to perform non-blocking I/O operations — despite the fact that JavaScript is single-threaded — by offloading operations to the system kernel whenever possible.
- Since most modern kernels are multi-threaded, they can handle multiple operations executing in the background. When one of these operations completes, the kernel tells Node.js so that the appropriate callback may be added to the poll queue to eventually be executed
- Node Js is single threaded
- Node Js makes use of event loop for being fast and asynchronous

Event loop mystified



Airtribe

Callstack & callback queue

Callstack queue

The call stack keeps track of the function currently being executed and where it is run from. A function is added to the call stack when it is about to be executed. This helps JavaScript retrace its steps after executing a function.

Callback queue

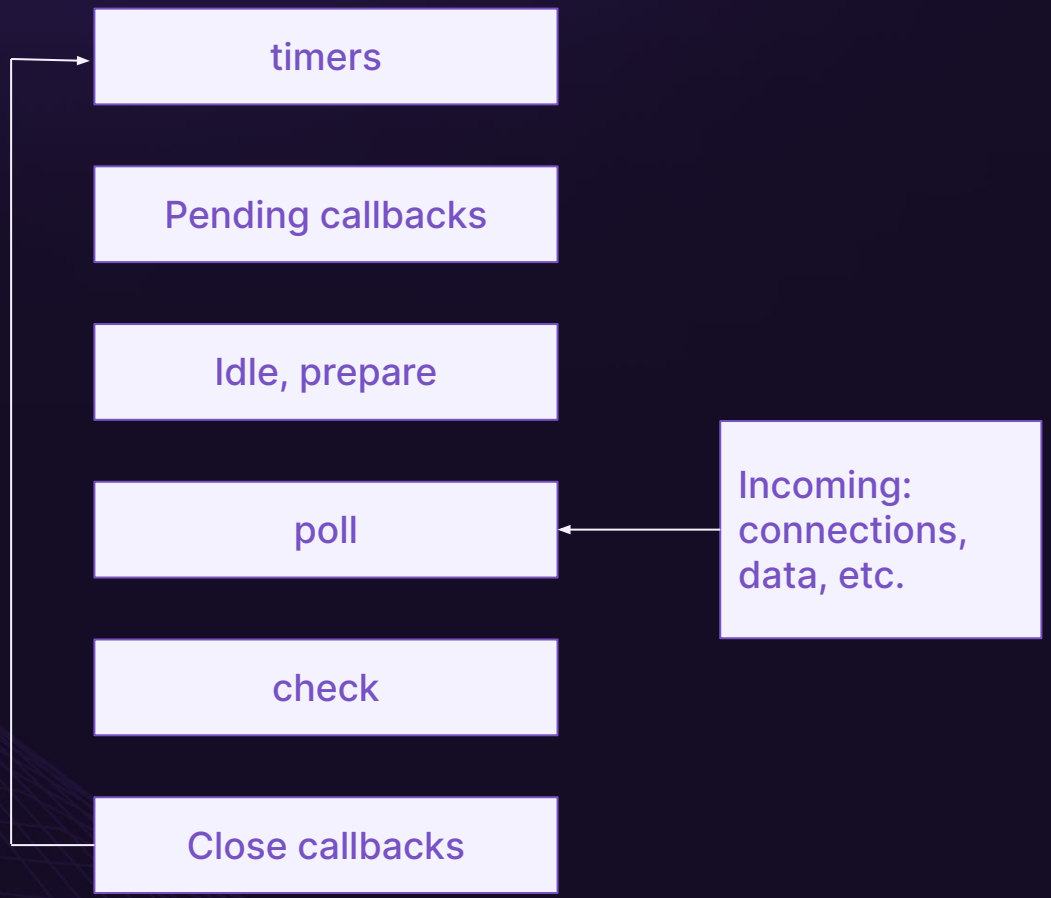
The callback queues are queues that hold callback functions to asynchronous operations when they have been completed in the background.

Callstack & callback queue

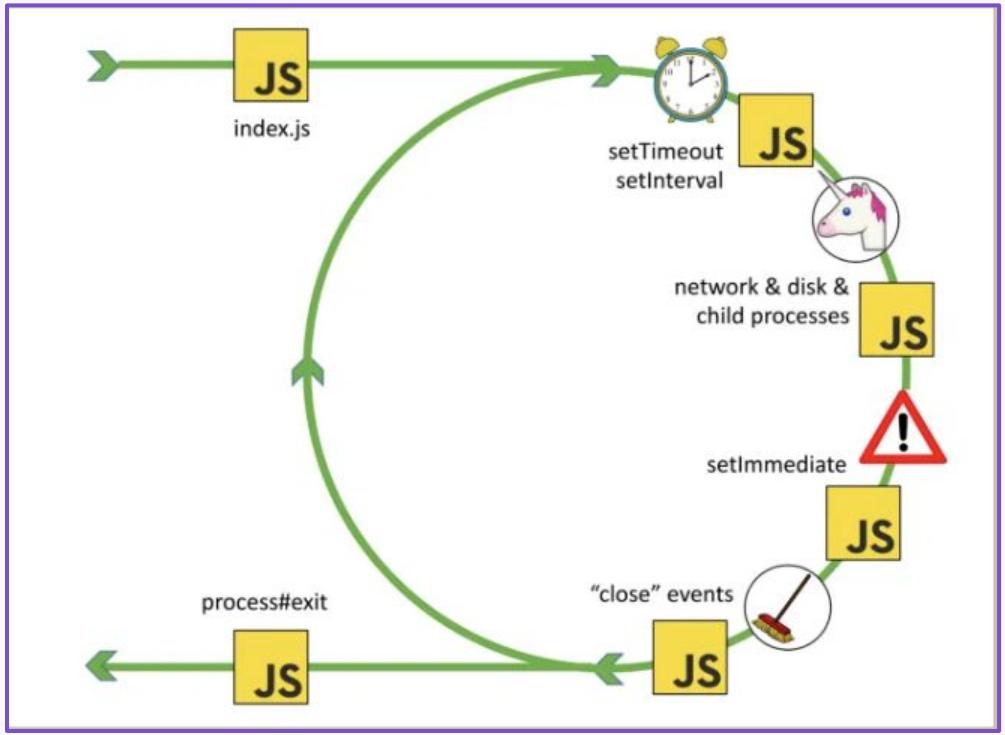
- They work in a first-in-first-out (FIFO) manner.
- There are multiple types of callback queues
- Callback queue types - IO Callback queue, Timer queue, Microtask queue, Check Queue, Close Queue

Order of queue

- Microtask Queue
- Timer Queue
- IO Queue
- Check Queue
- Close Queue


timers
Pending callbacks
Idle, prepare
poll
Incoming:
connections,
data, etc.
check
Close callbacks

Order of queue visualised



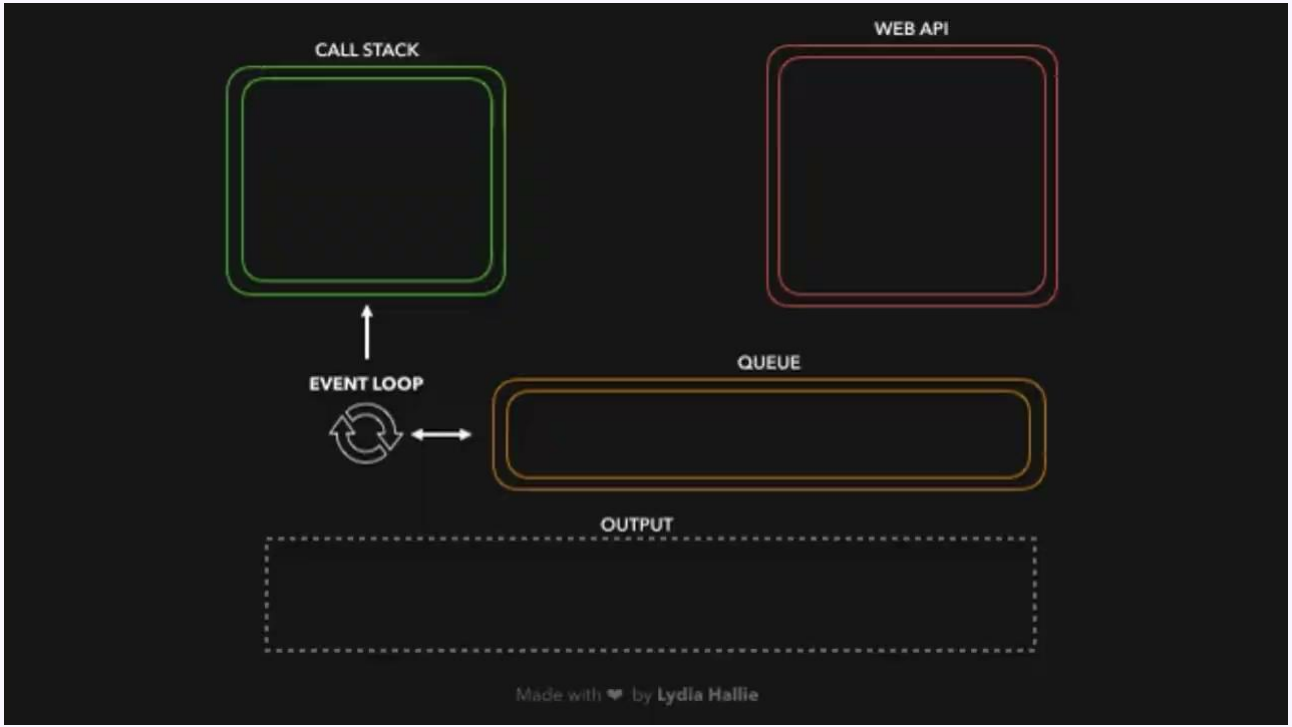
Airtribe

Code snippet-1

```
const foo = () => console.log("First");
const bar = () => setTimeout(() => console.log("Second"), 500);
const baz = () => console.log("Third");

bar();
foo();
baz();
Output:
First
Third
Second
```

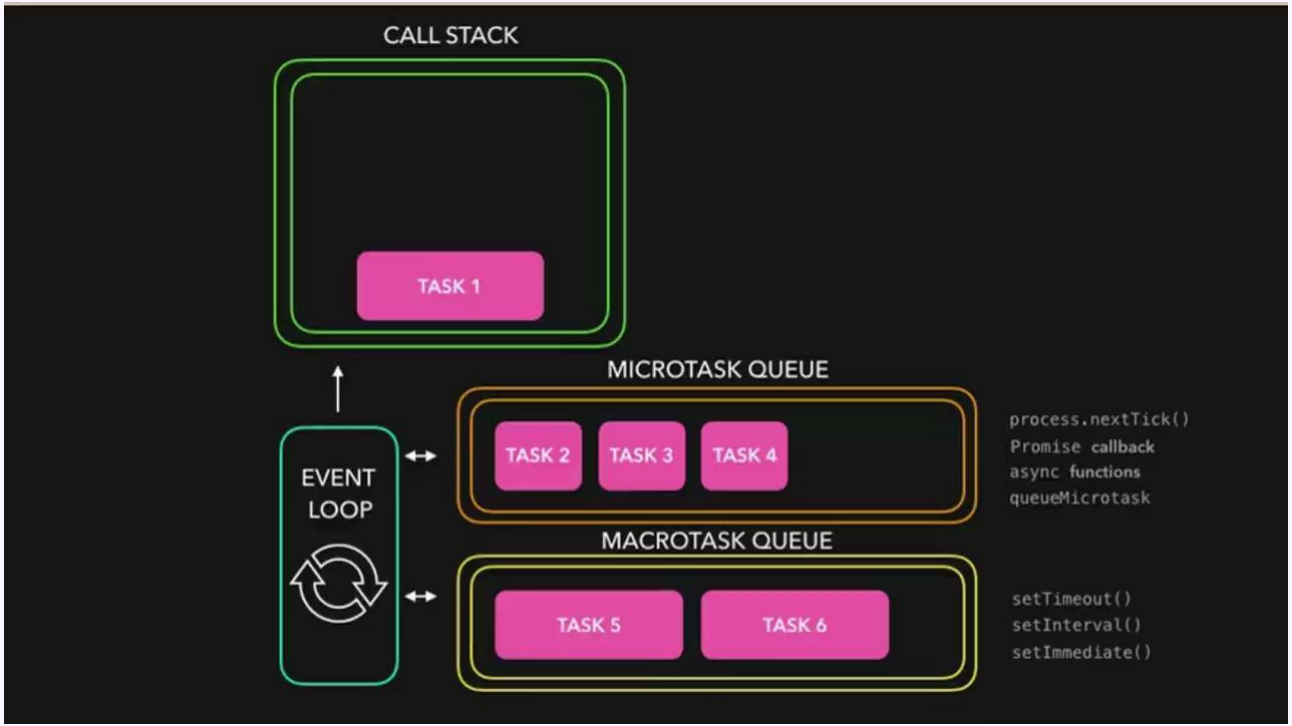
Airtribe

Code snippet–1 visualization



Airtribe

Microtasks/callstacks/macrotasks



Airtribe

Microtasks/macrotasks

- Macrotasks:

  setTimeout, setInterval, setImmediate, requestAnimationFrame, I/O, UI rendering

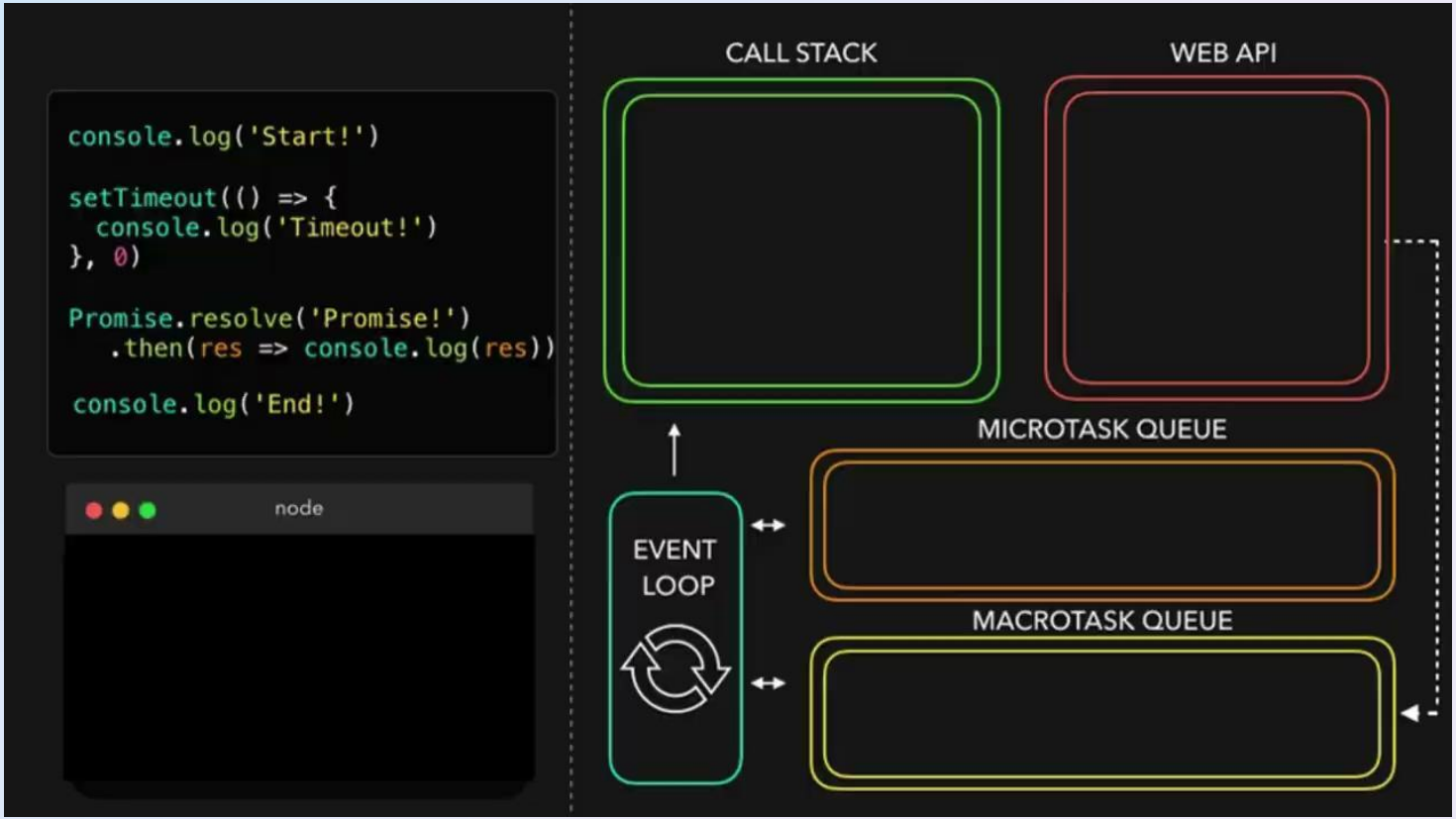
- Microtasks:

  process.nextTick, Promises, queueMicrotask, MutationObserver

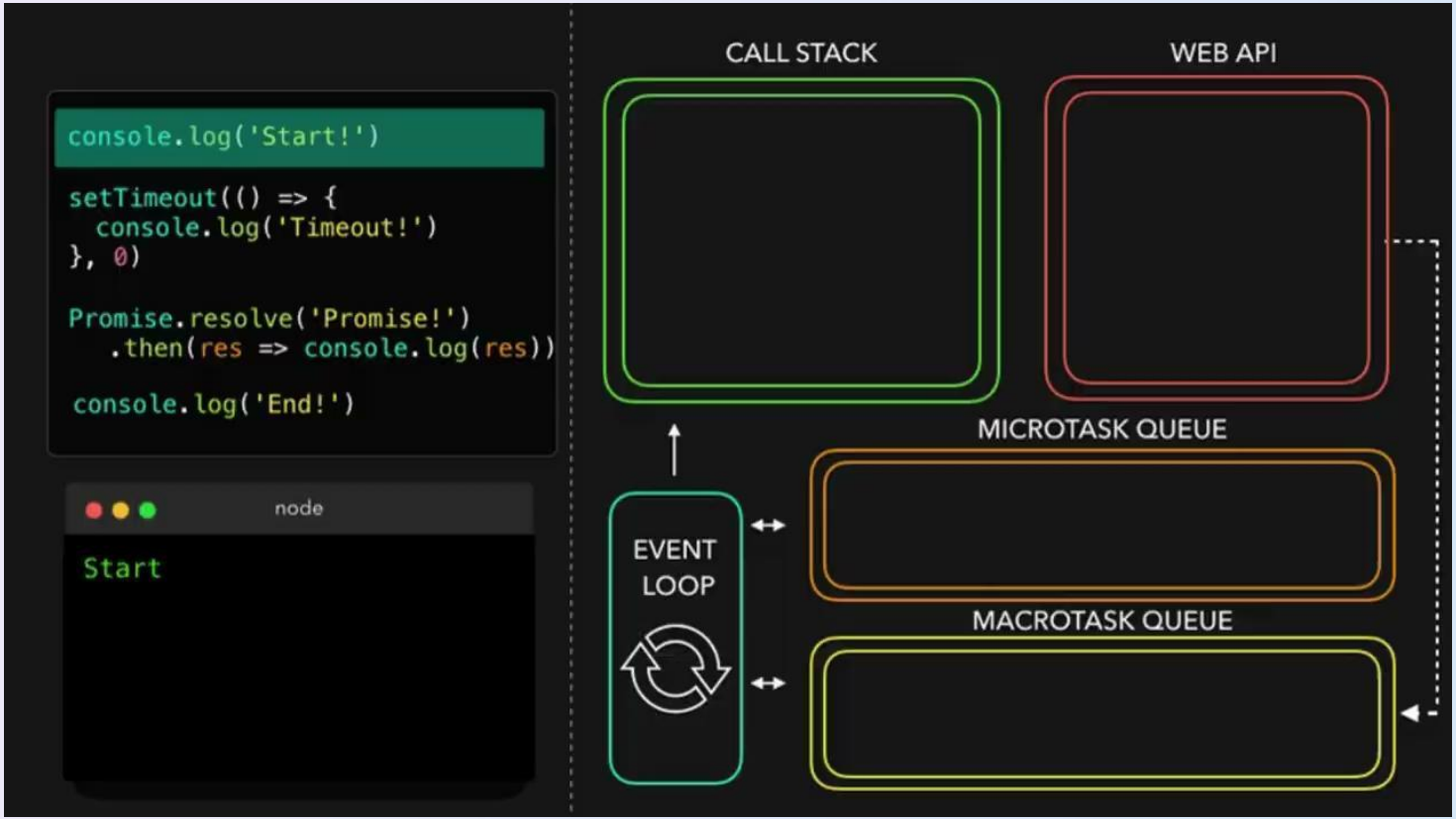
Code snippet–2

```
console.log('Start!')

setTimeout(() => {
  console.log('Timeout!')
}, 0)

Promise.resolve('Promise!')
    .then(res => console.log(res))

console.log('End!')
```

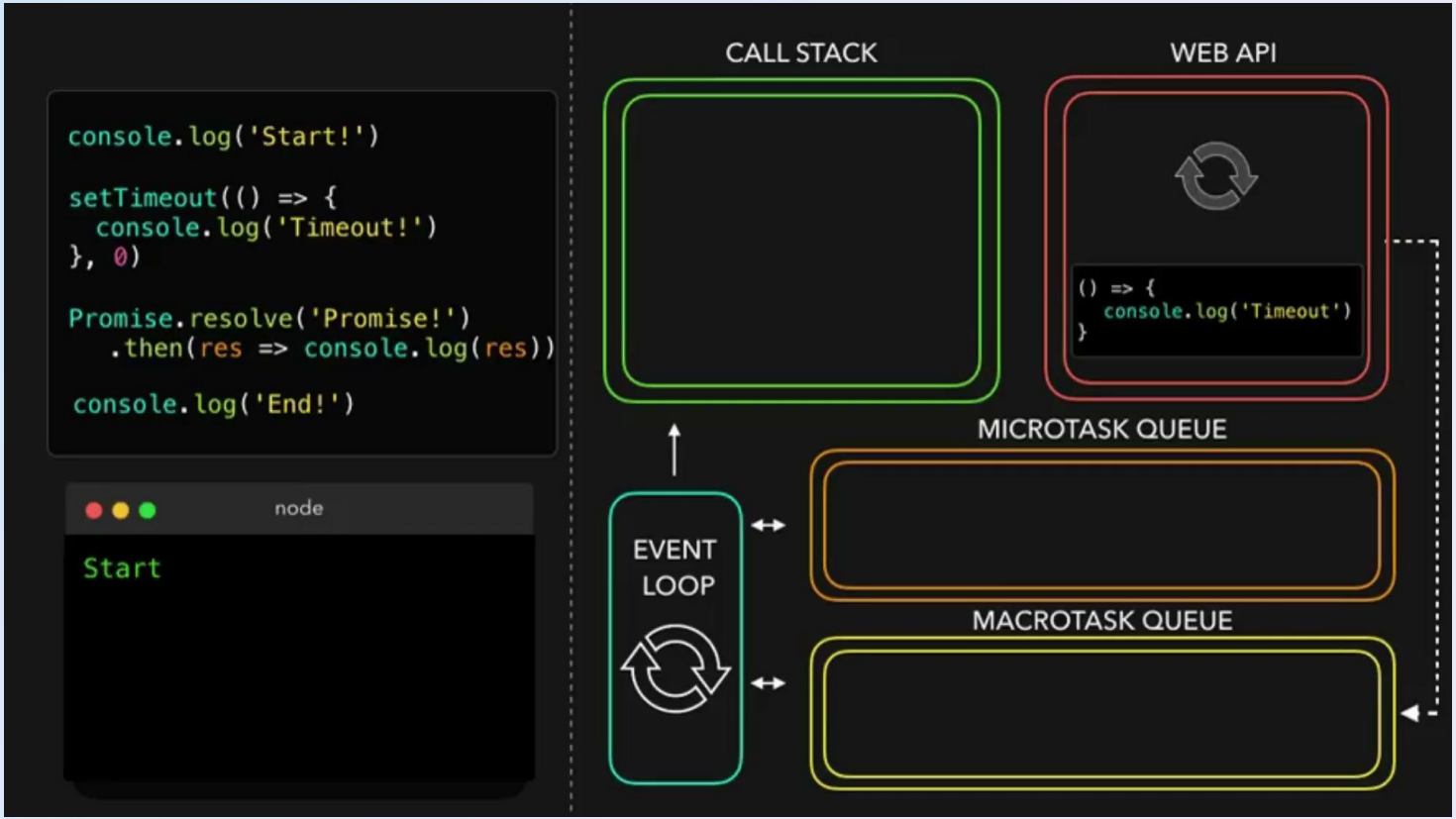
Airtribe


console.log('Start!')
setTimeout(() => {
console.log('Timeout!')
}, 0)
Promise.resolve('Promise!')
.then(res => console.log(res))
console.log('End!')
node
CALL STACK
WEB API
MICROTASK QUEUE
EVENT LOOP
MACROTASK QUEUE

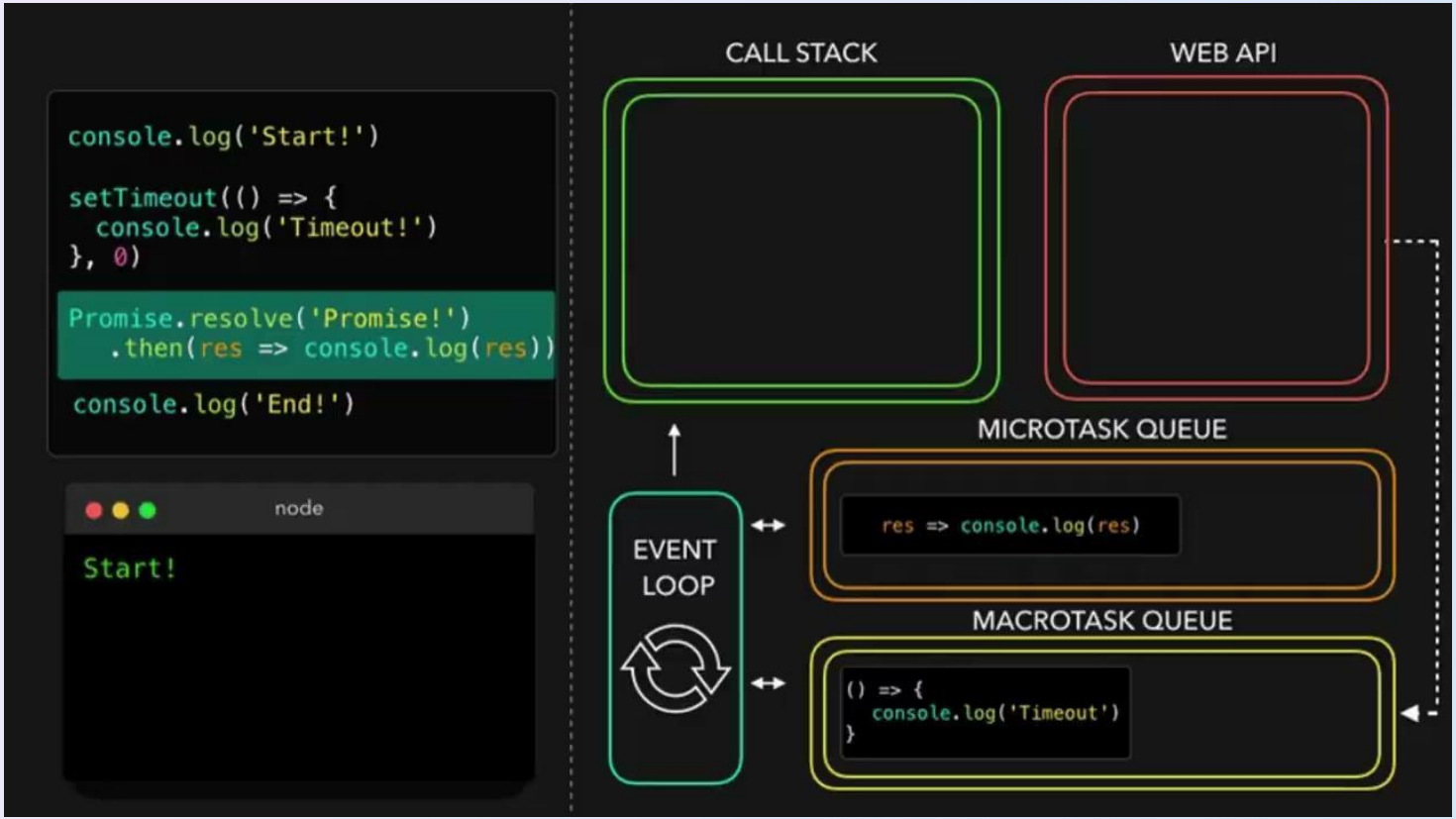
Airtribe


console.log('Start!')
setTimeout(() => {
console.log('Timeout!')
}, 0)
Promise.resolve('Promise!')
.then(res => console.log(res))
console.log('End!')
node
Start
CALL STACK
WEB API
MICROTASK QUEUE
EVENT LOOP
MACROTASK QUEUE

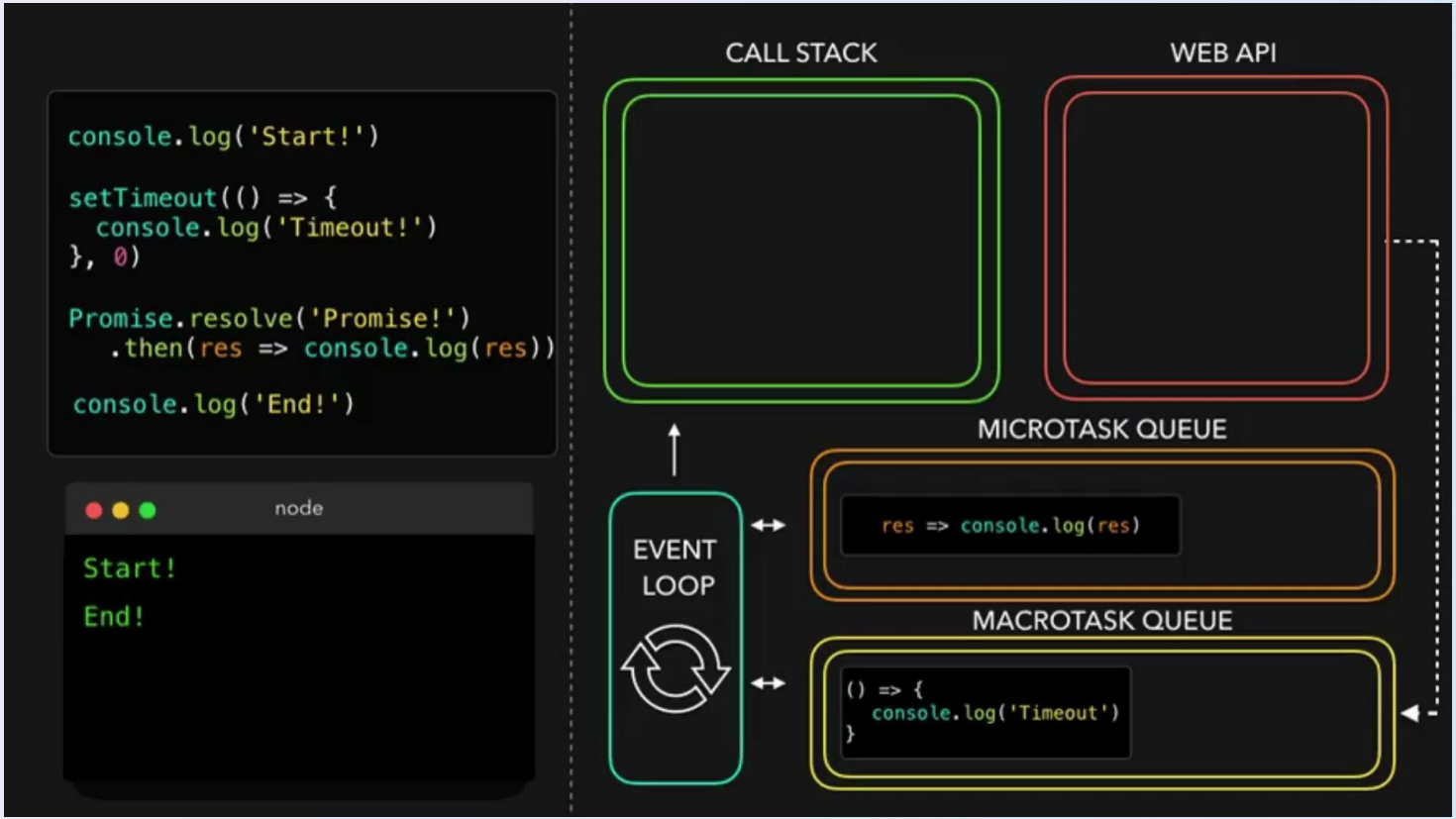
Airtribe


console.log('Start!')
setTimeout(() => {
console.log('Timeout!')
}, 0)
Promise.resolve('Promise!')
.then(res => console.log(res))
console.log('End!')
node
Start
CALL STACK
WEB API
() => {
console.log('Timeout')
}
MICROTASK QUEUE
EVENT LOOP
MACROTASK QUEUE

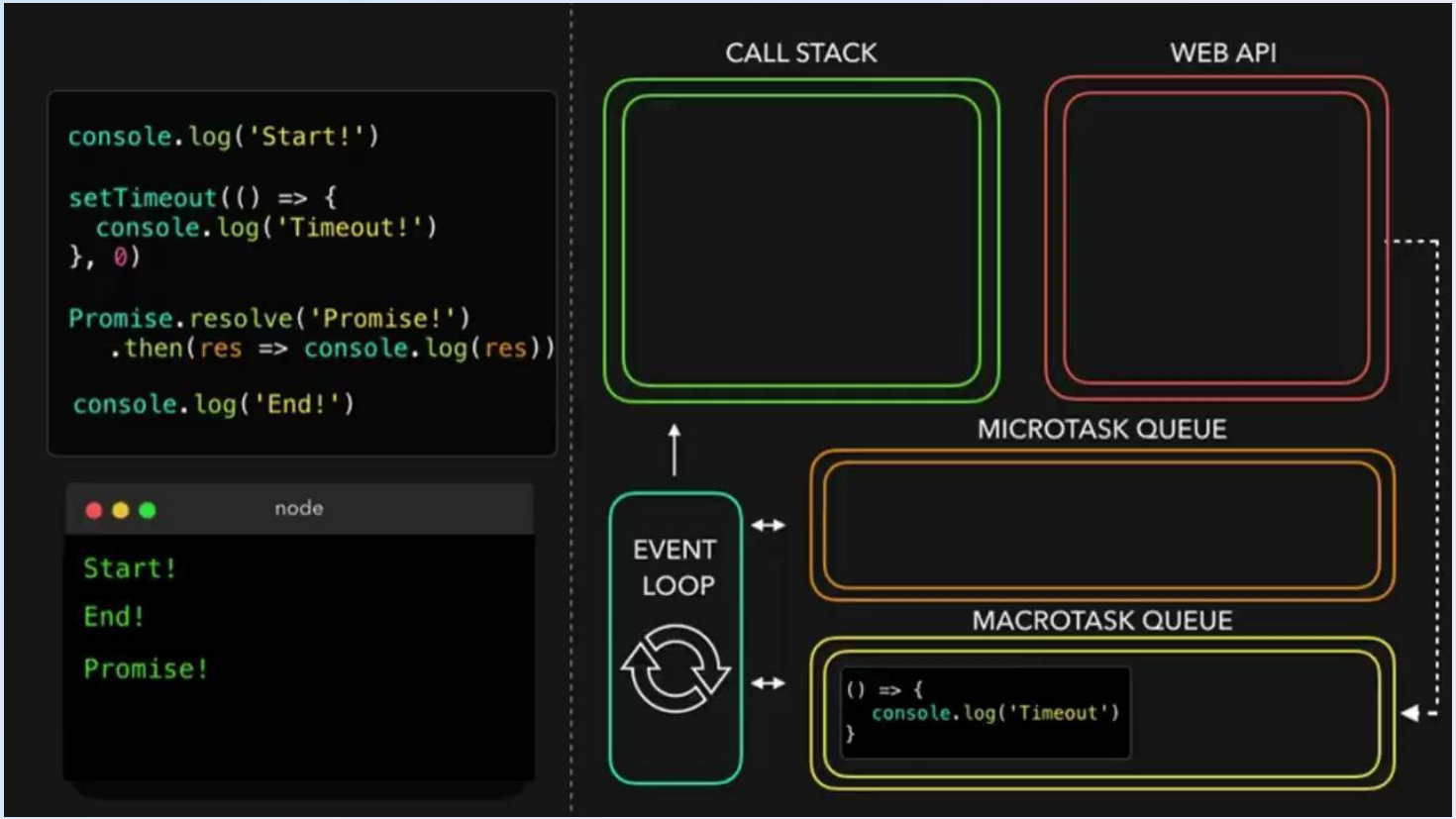
Airtribe


console.log('Start!')
setTimeout(() => {
console.log('Timeout!')
}, 0)
Promise.resolve('Promise!')
.then(res => console.log(res))
console.log('End!')
node
Start!
CALL STACK
WEB API
MICROTASK QUEUE
res => console.log(res)
EVENT LOOP
MACROTASK QUEUE
() => {
console.log('Timeout')
}

Airtribe


console.log('Start!')
setTimeout(() => {
console.log('Timeout!')
}, 0)
Promise.resolve('Promise!')
.then(res => console.log(res))
console.log('End!')
node
Start!
End!
CALL STACK
WEB API
MICROTASK QUEUE
res => console.log(res)
EVENT LOOP
MACROTASK QUEUE
() => {
console.log('Timeout')
}

Airtribe


console.log('Start!')
setTimeout(() => {
console.log('Timeout!')
}, 0)
Promise.resolve('Promise!')
.then(res => console.log(res))
console.log('End!')
node
Start!
End!
Promise!
CALL STACK
WEB API
MICROTASK QUEUE
EVENT LOOP
MACROTASK QUEUE
() => {
console.log('Timeout')
}

Airtribe

Clustering – Node Js

- Node.js is single-threaded by default, which means it can only use one CPU core.
- Clustering allows Node.js to create multiple processes, each running on a different CPU core.
- This can improve performance and scalability.
- Clustering is implemented using the built-in cluster module in Node.js.

Clustering – Node Js

- The cluster module allows you to create a master process that manages multiple worker processes.
- Each worker process runs on a separate CPU core and can handle incoming requests.
- By default, the cluster module uses round-robin load balancing.
- This means that incoming requests are distributed evenly across all worker processes.
- You can customize the load balancing algorithm using the cluster.schedulingPolicy property.

Inter process communication

- Inter-process communication (IPC) is the mechanism by which the master process communicates with the worker processes.
- The cluster module provides several methods for sending messages between the master and worker processes, including cluster.fork() and worker.send().
- IPC can be used to share state between processes or to send commands from the master process to the worker processes.

Thank you

Airtribe