# Session 5.1

## Node Fundamentals - Event Loop

# Topics

- **Node Fundamentals - II**
  - Asynchronous Code, Libuv and Event Loop

# Asynchronous Code, Libuv and Event Loop

# The Event Loop

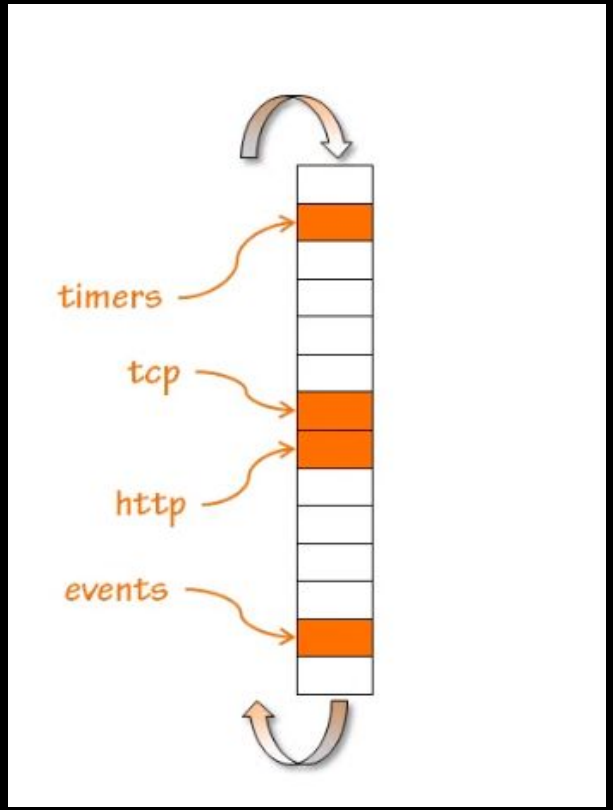
The event loop is what allows Node.js to perform non-blocking I/O operations.

Since most modern kernels are multi-threaded they can handle multiple operations in the background.

** In the Browser, the event loop is constantly listening to DOM events, such as key presses and mouse clicks.

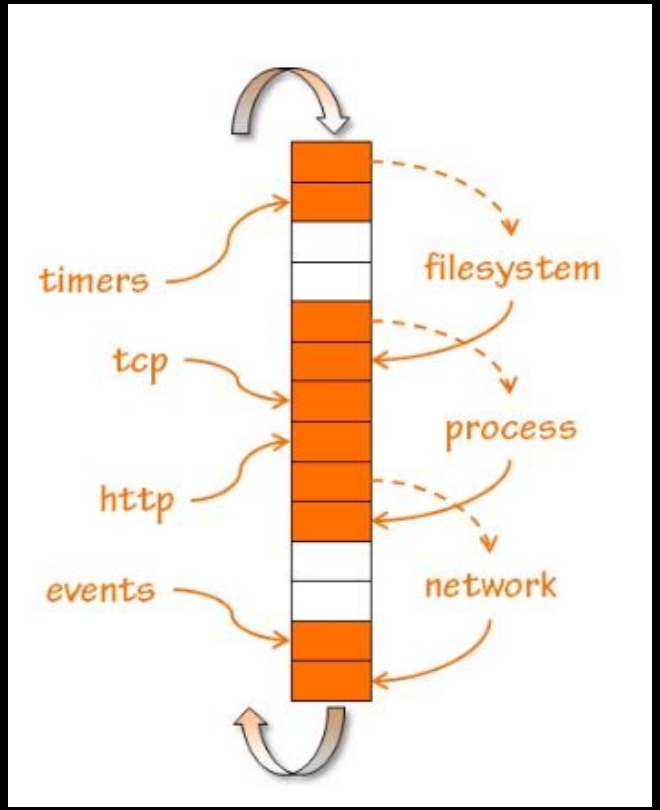
# Nodes Event Loop

- Nodes Event Loop is constantly listening for events on the server side.
- These events can be externally generated such as http requests or tcp connections.
- They can be internal events generated by your application such as Timers



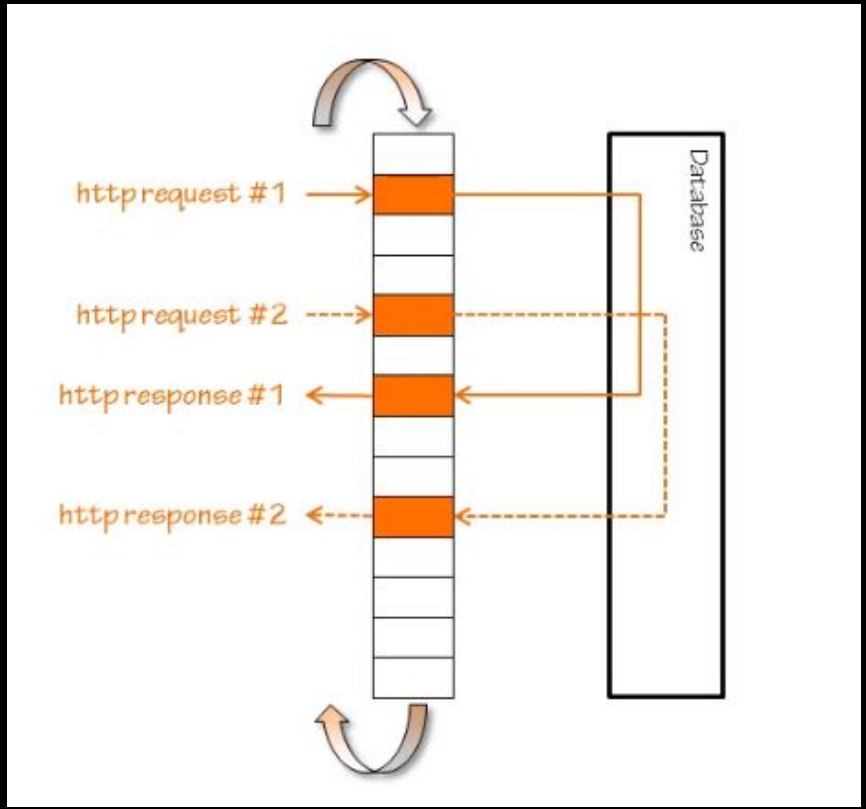
# Nodes Event Loop

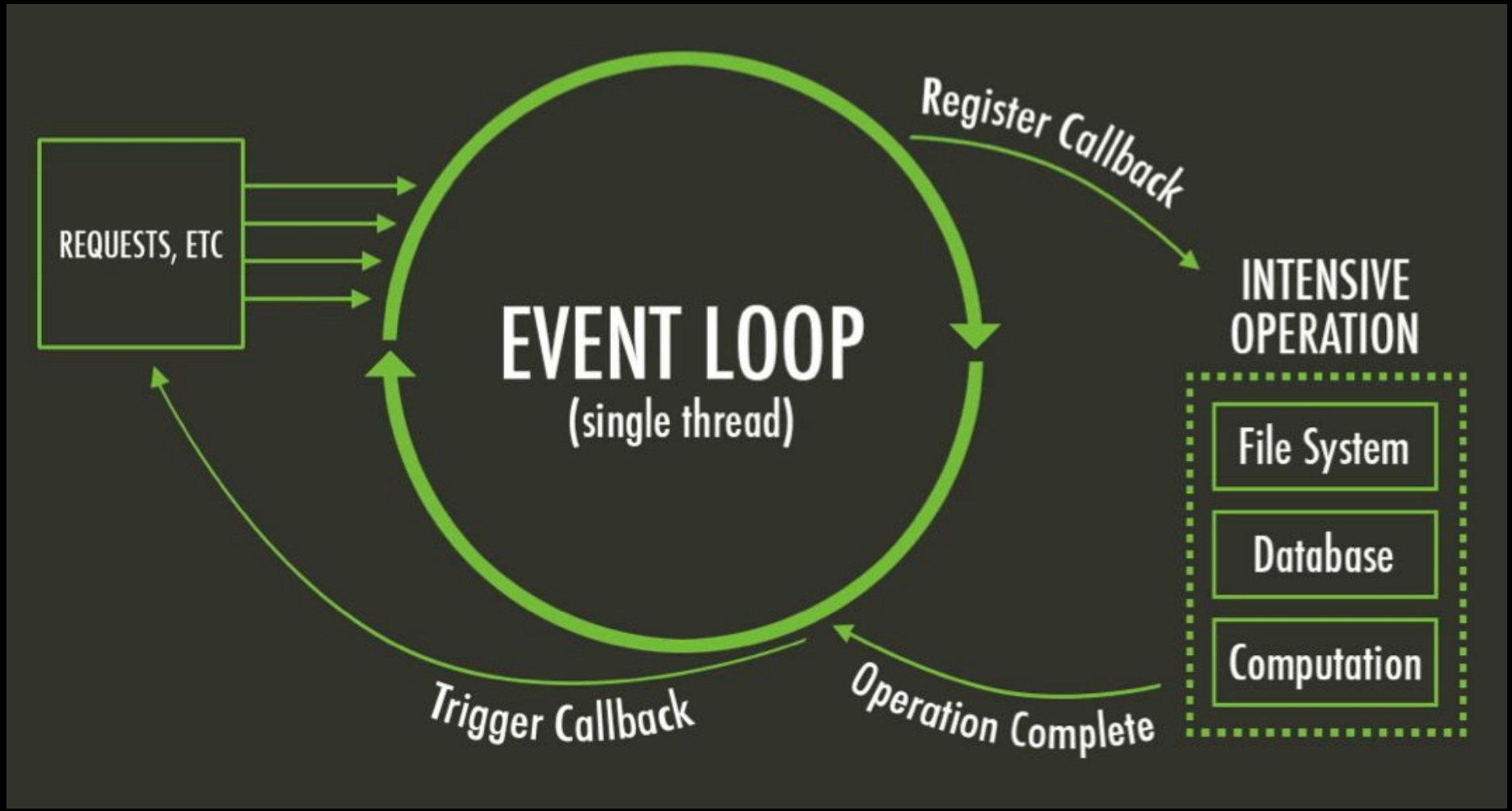
- Other events may triggered as a result of a response to external event.
- ie. Requesting Node to open file for reading will trigger event when completed.
- All these events will be handled asynchronously.



# Web Application example for non-blocking approach

- Application raises an http request, which triggers an event to read a query from database.
- Once Node receives event from database it is complete it sends back response.
- While waiting, Node is <u>not</u> blocked and free to handle additional requests.



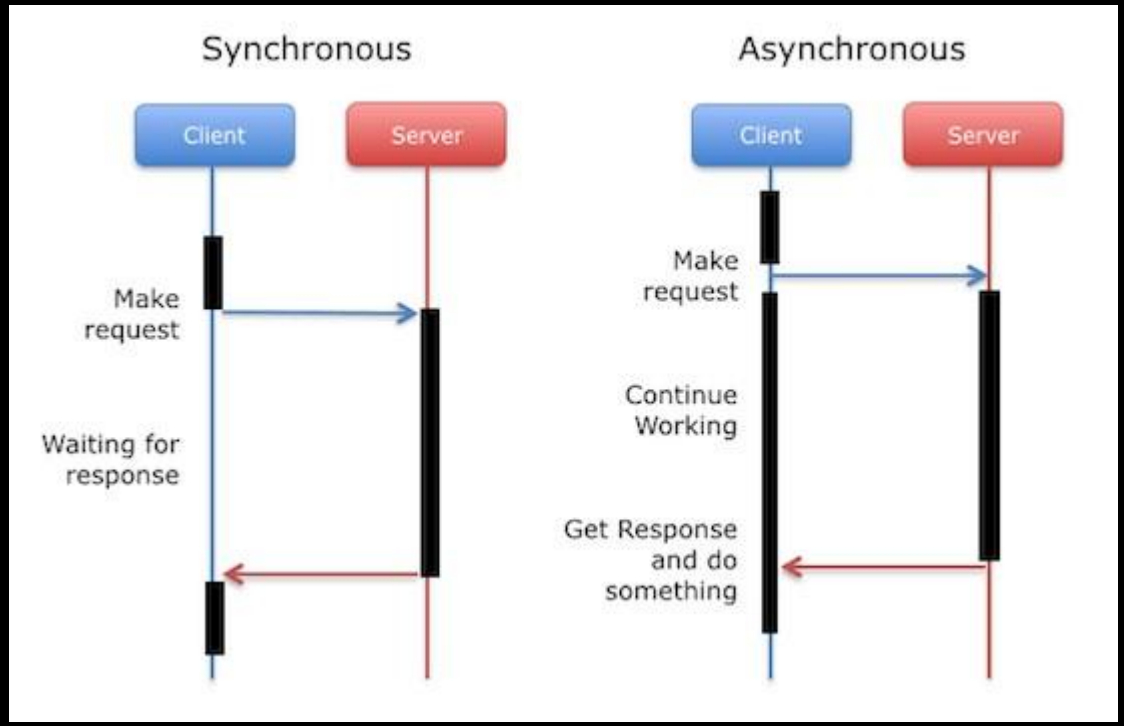

REQUESTS, ETC
EVENT LOOP
(single thread)
Register Callback
INTENSIVE OPERATION
File System
Database
Computation
Operation Complete
Trigger Callback

# Async I/O

The following tasks should be done asynchronously using the event loop:

- I/O operations
- Heavy computation
- Anything requiring blocking

# Asynchronous vs Synchronous programmimg



# What is a callback ?

- A callback is a function that is to be executed after another function has finished executing - hence the name ‘call back’

- An event is like a broadcast, while a callback is like a handshake.

- A component that raises an event knows nothing about its client, while a component that uses a callback knows a great deal.

# Asynchronous vs Synchronous code

```
// Synchronous code
var conn = getDbConnection(connectionString);
var stmt = conn.createStatement();
var result = stmt.executeQuery(sqlQuery);
```

# Asynchronous vs Synchronous code

```
// Synchronous code
var conn = getDbConnection(connectionString);
var stmt = conn.createStatement();
var result = stmt.executeQuery(sqlQuery);
```

```
// Asynchronous "non-blocking" code
getDbConnection(connectionString, function (err, conn) {  // callback #1
    conn.createStatement (function (err, stmt) {          // callback #2
        var result = stmt.executeQuery(sqlQuery);
    })
})
```

# Timers

- The timer module exposes a global API for scheduling function to be called at some point in the future.

- There is no need to use require('timers') because the timer functions are global.

- Node.js and Web Browser timer functions implement a similar API. However, Node.js uses the Event Loop to achieve it.

# setImmediate(callback[,...args])

```
setImmediate(() => {
  console.log('immediate');
});
```

- Schedules the “immediate” execution of the callback after I/) events callbacks

# setInterval(callback, delay[,...args])

```
setInterval((name) => {
  console.log(`Oh yeah, ${ name}`)
, 1000, "Macho man"})
```

- Schedules repeated execution of callback every delay millisecond.

# setTimeout(callback,delay[,...args])

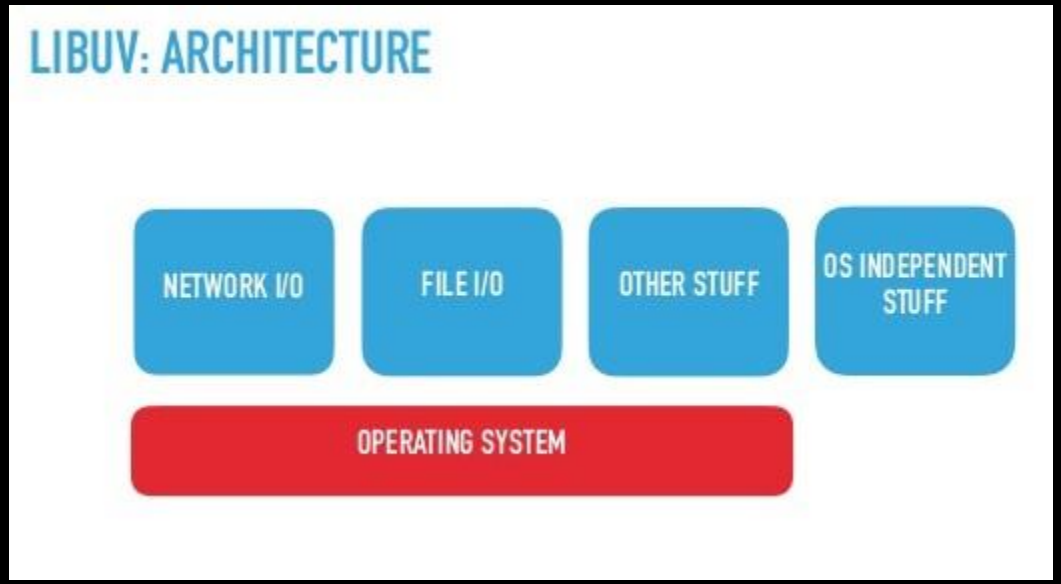
```
setTimeout(() => {
  console.log('timeout');
}, 0);
```

- Schedules execution of a one-time callback after delay of milliseconds.
- There is no guarantee that the callback will be triggered exactly after the specified time; instead, the callback will be called as close as possible to the time specified.

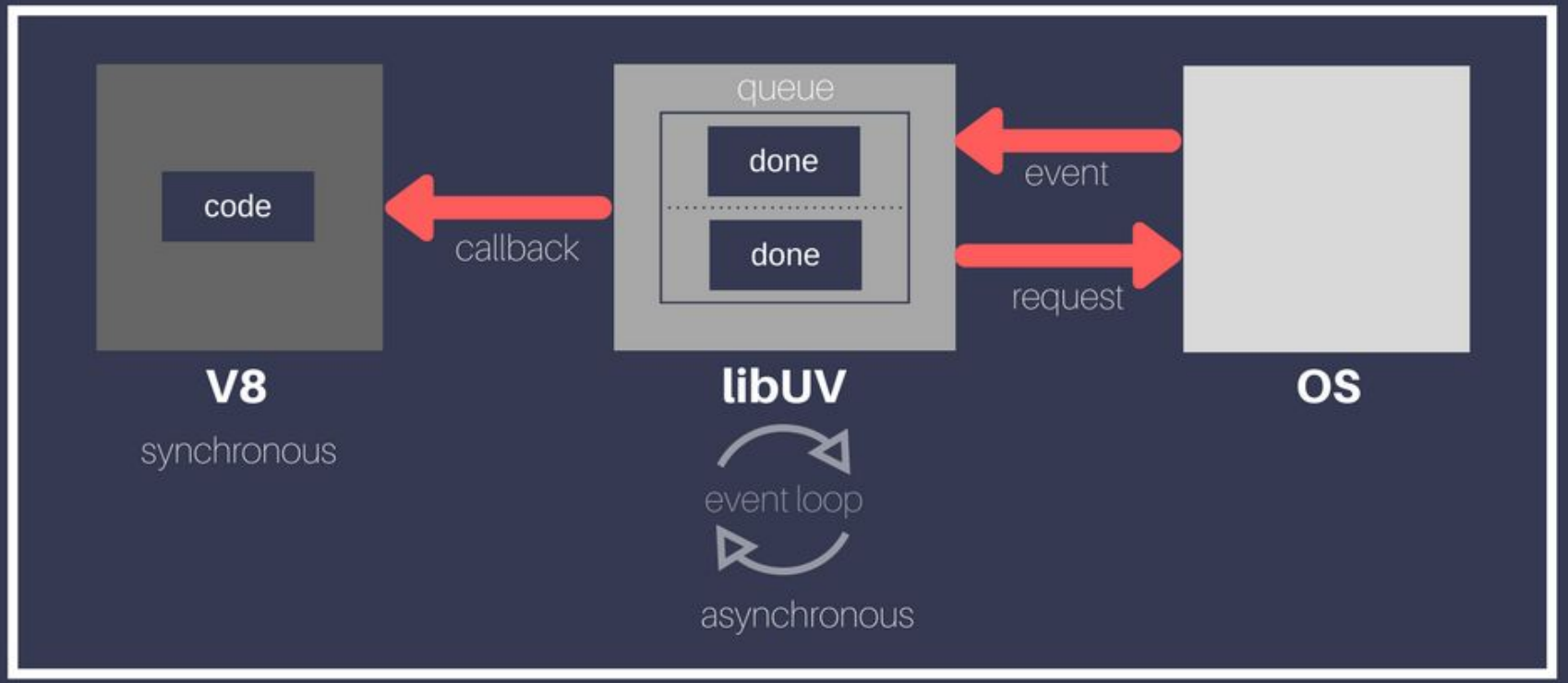
# libuv - (Unicorn Velociraptor Library)

- A multi-platform C library that provides support for asynchronous I/O based on event loops.

# libuv


LIBUV: ARCHITECTURE
NETWORK I/O
FILE I/O
OTHER STUFF
OS INDEPENDENT STUFF
OPERATING SYSTEM

# NodeJS


code
callback
queue
done
done
event
request
V8
synchronous
libUV
event loop
asynchronous
OS