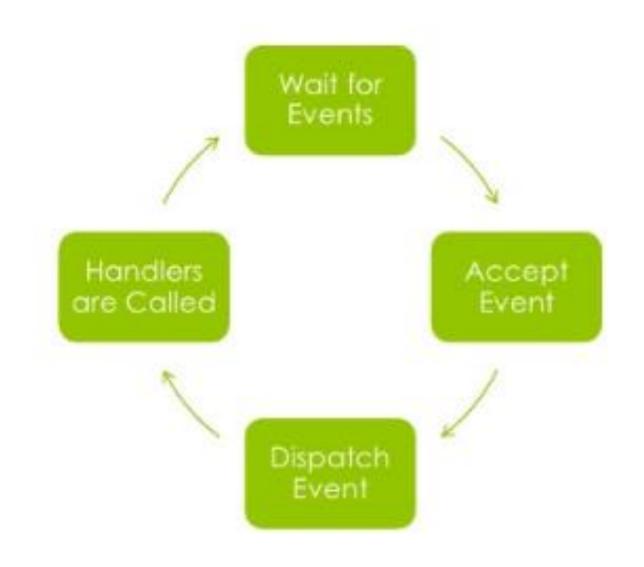


ASYNCHRONOUS EVENT PROGRAMMING WITH NODE

Andrew Sheehan MET CS602

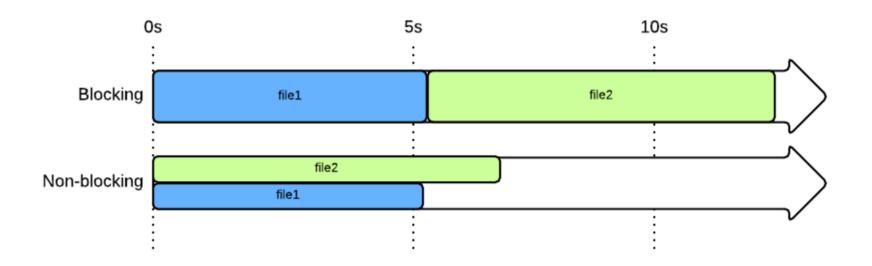
EVENT-DRIVEN

One of Node's Core Strengths: Faster Processing.

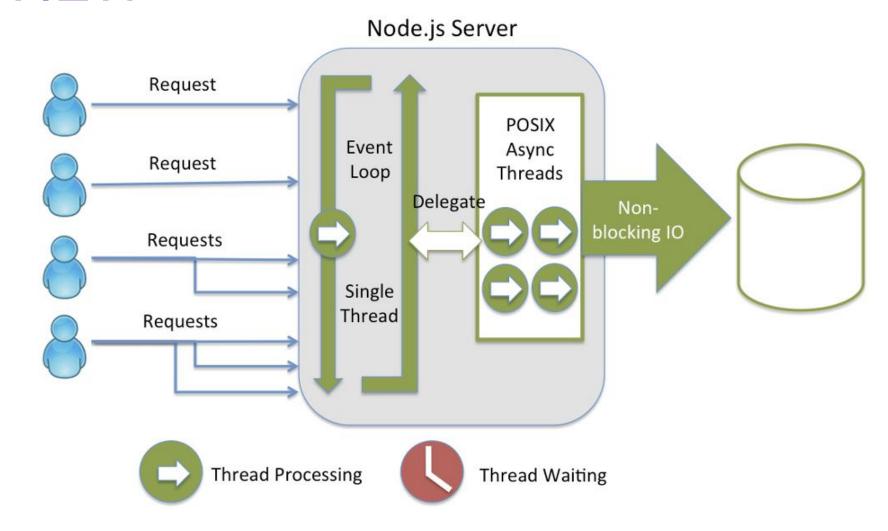


BLOCKING VS NON BLOCKING

Rather than waiting for an operation to finish, create a callback that will be invoked when the operation ends.



HIGH LEVEL OVERVIEW



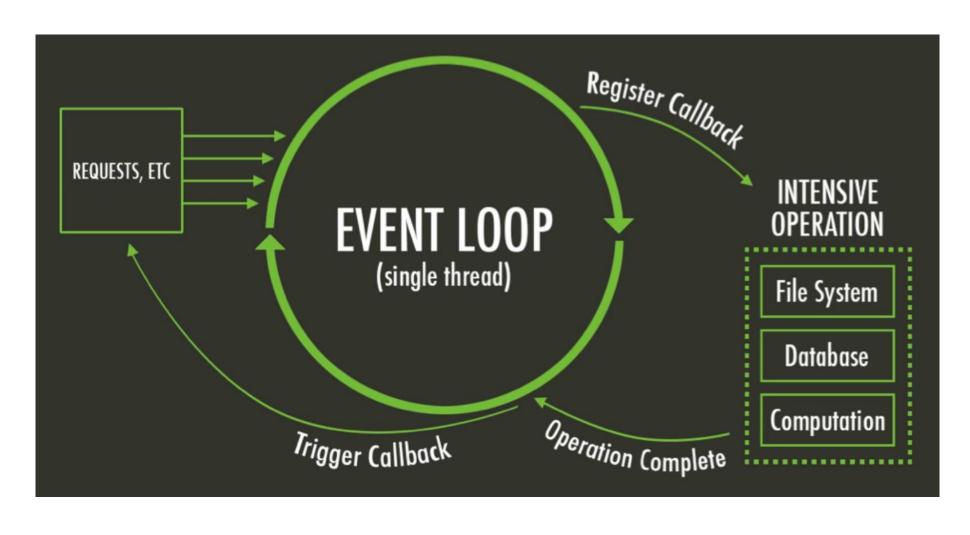
BLOCKING VS NON BLOCKING

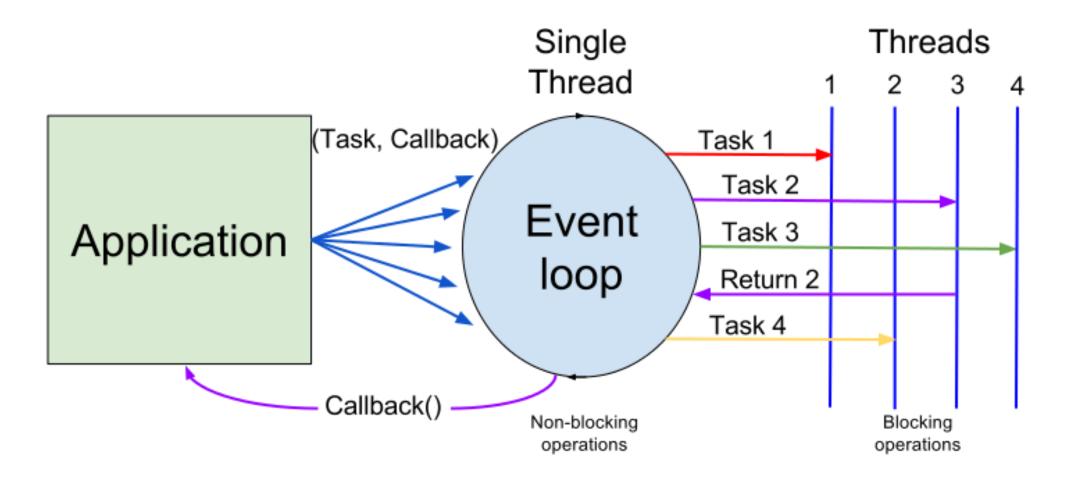
illiseconds	Blocking API
20	Begin Request 1
40	
60	waiting on database
80	
100	End Request 1
120	Begin Request 2
140	
160	waiting on database
180	
200	End Request 2
220	Begin Request 3
240	
260	waiting on database
280	
300	End Request 3
320	Begin Request 4
340	
360	waiting on database
380	
400	End Request 4
420	Begin Request 5
440	
460	waiting on database
480	
500	End Request 5

Non-Blocking API

Begin Request 1	
Begin Request 2	
Begin Request 3	
Begin Request 4	
End Request 1	
End Request 2	
End Request 3	
End Request 4	
Begin Request 5	
Begin Request 6	
Begin Request 7	
Begin Request 8	
End Request 5	
End Request 6	
End Request 7	
End Request 8	
Begin Request 9	
Begin Request 10	
Begin Request 11	
Begin Request 12	
End Request 9	
End Request 10	
End Request 11	
End Request 12	
Begin Request 13	

SINGLE THREADED

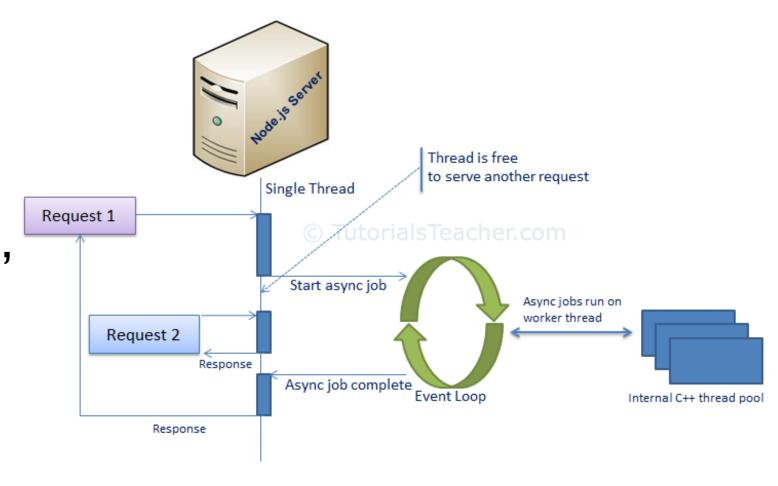




The main event loop is single-threaded but most of the I/O runs on separate threads

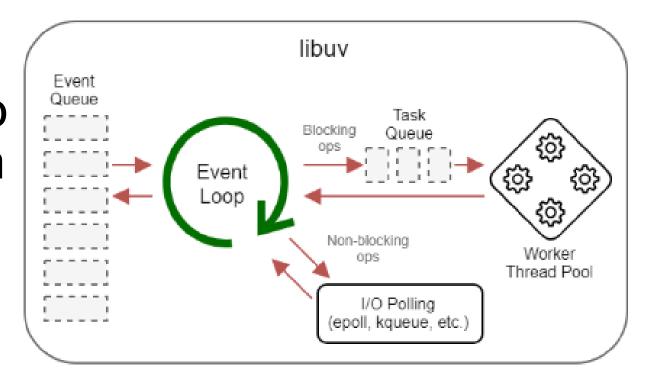
HIGH LEVEL THREADS

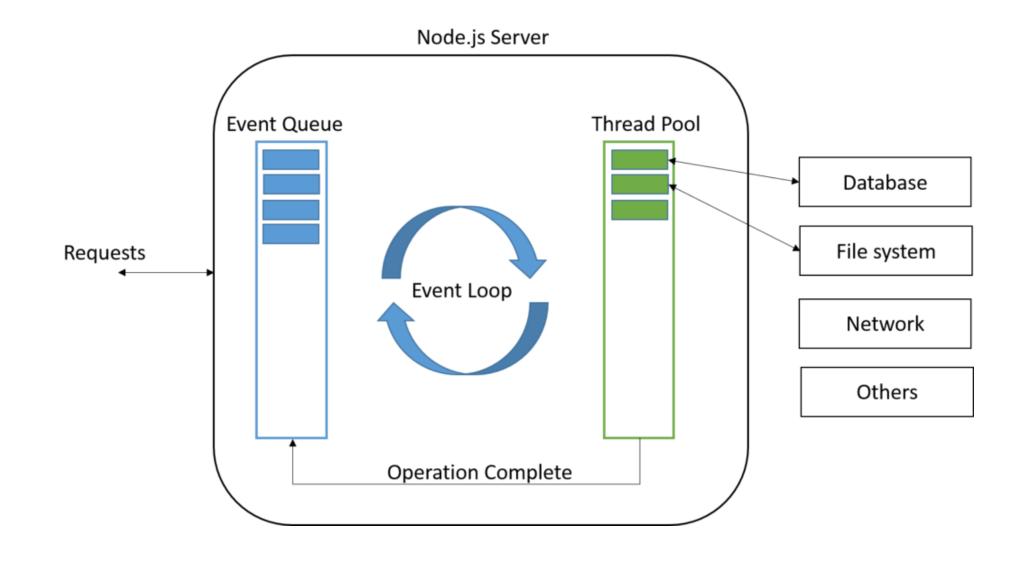
Threads run in processes; one process can have many threads in it, and as they are in same process, they share a memory space.



EVENT LOOP WORKER THREADS

Events are pushed to the main thread then worker threads process the request.





The event loop is implemented via **libuv** (C++).

EVENT EMITTER

Have custom events? Use the events module.

const emitter = require("events");

EVENT EMITTER

Who else uses EventEmitter class?



EVENT EMITTER NOTES & COMMENTS

The EventEmitter will call all listeners synchronously in the order in which they were registered.

This is important to ensure the proper sequencing of events and to avoid race conditions or logic errors

EVENT EMITTER

```
const EventEmitter = require('events');
class MyEmitter extends EventEmitter {}
const myEmitter = new MyEmitter();
myEmitter.on('event', () => {
  console.log('an event occurred!');
});
myEmitter.emit('event');
```

Used to raise and handle custom events

EXAMPLE: EMITTER

```
var emitter = require('events').EventEmitter;
function LoopProcessor(num) {
    var e = new emitter();
    setTimeout(function () {
        for (var i = 1; i <= num; i++) {
            e.emit('BeforeProcess', i);
            console.log('Processing number:' + i);
            e.emit('AfterProcess', i);
    , 2000)
    return e;
var lp = LoopProcessor(3);
lp.on('BeforeProcess', function (data) {
    console.log('About to start the process for ' + data);
});
lp.on('AfterProcess', function (data) {
    console.log('Completed processing ' + data);
});
```

EVENT EMITTER ONE TIME

To emit (send) a one-time event

myemitter.Once()

Calling it again will do nothing...

EVENT EMITTER UNREGISTER EVENTS

myemitter.off('event', listener)

myemitter.removeListener('event', listener)