Node JS

Banuprakash

banuprakashc@yahoo.co.in

banu@lucidatechnologies.com

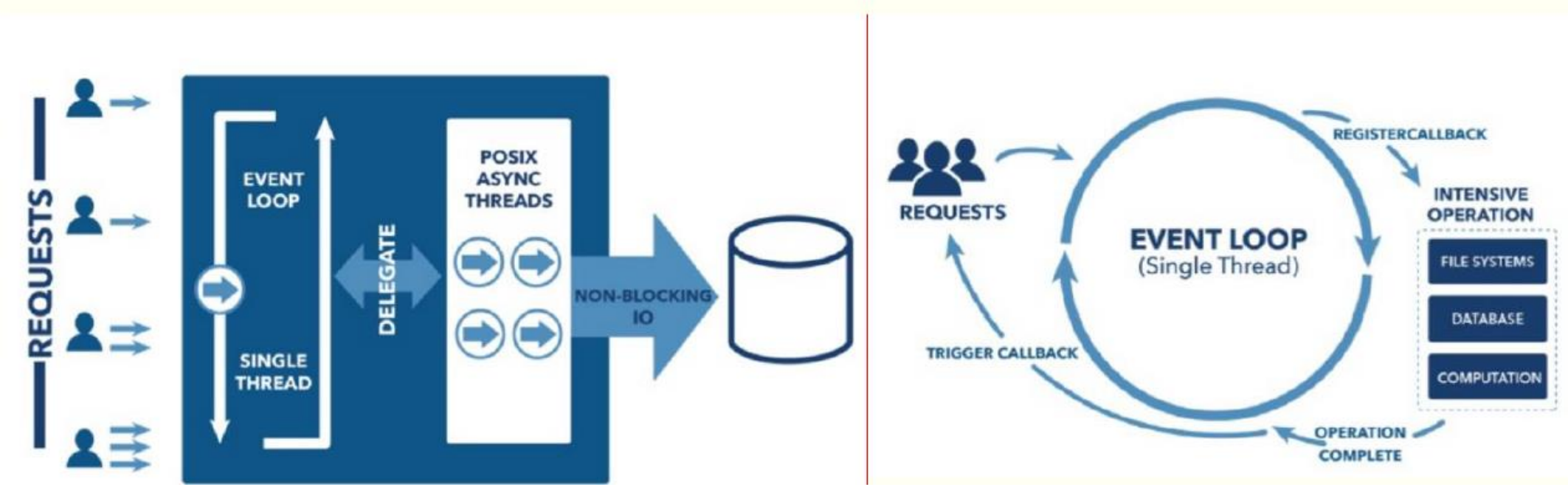
Node JS

- Node.js is a platform built on V8 engine runtime for easily building fast, scalable network applications.

- Node.js uses an **event-driven**, **non-blocking I/O model** that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.

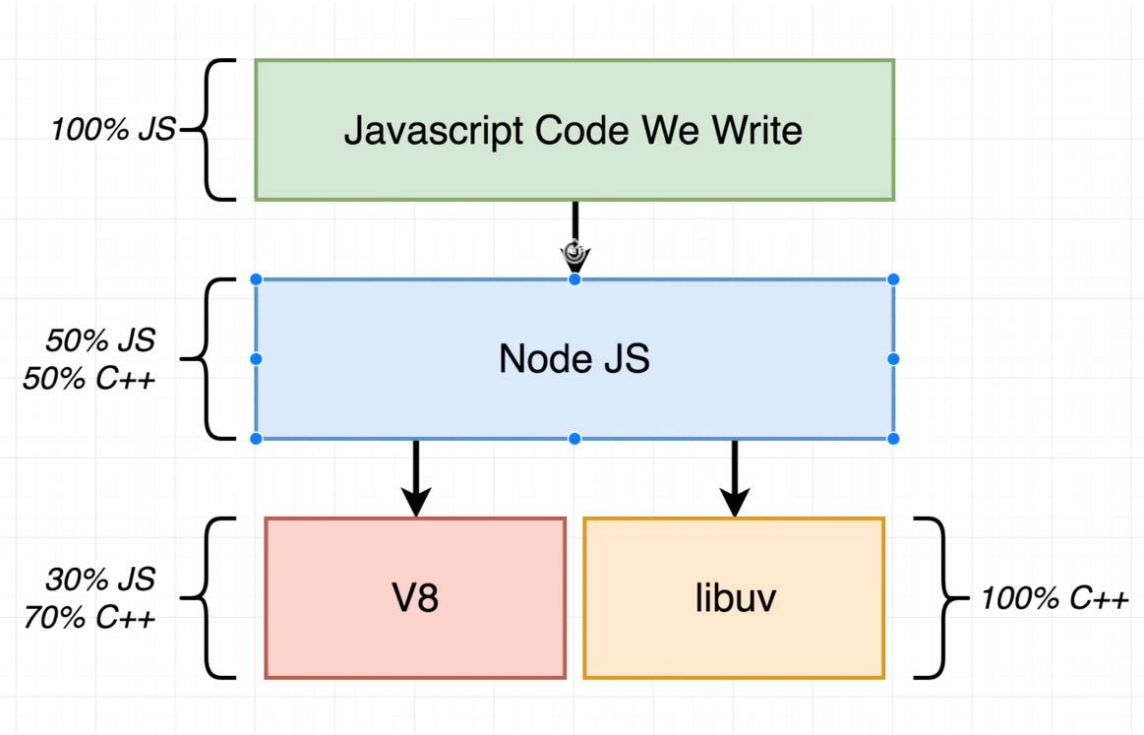
- Created by **Ray Daul** in 2009

Event loop


REQUESTS
EVENT LOOP
SINGLE THREAD
DELEGATE
POSIX ASYNC THREADS
NON-BLOCKING IO
REQUESTS
EVENT LOOP
(Single Thread)
REGISTERCALLBACK
INTENSIVE OPERATION
FILE SYSTEMS
DATABASE
COMPUTATION
TRIGGER CALLBACK
OPERATION COMPLETE

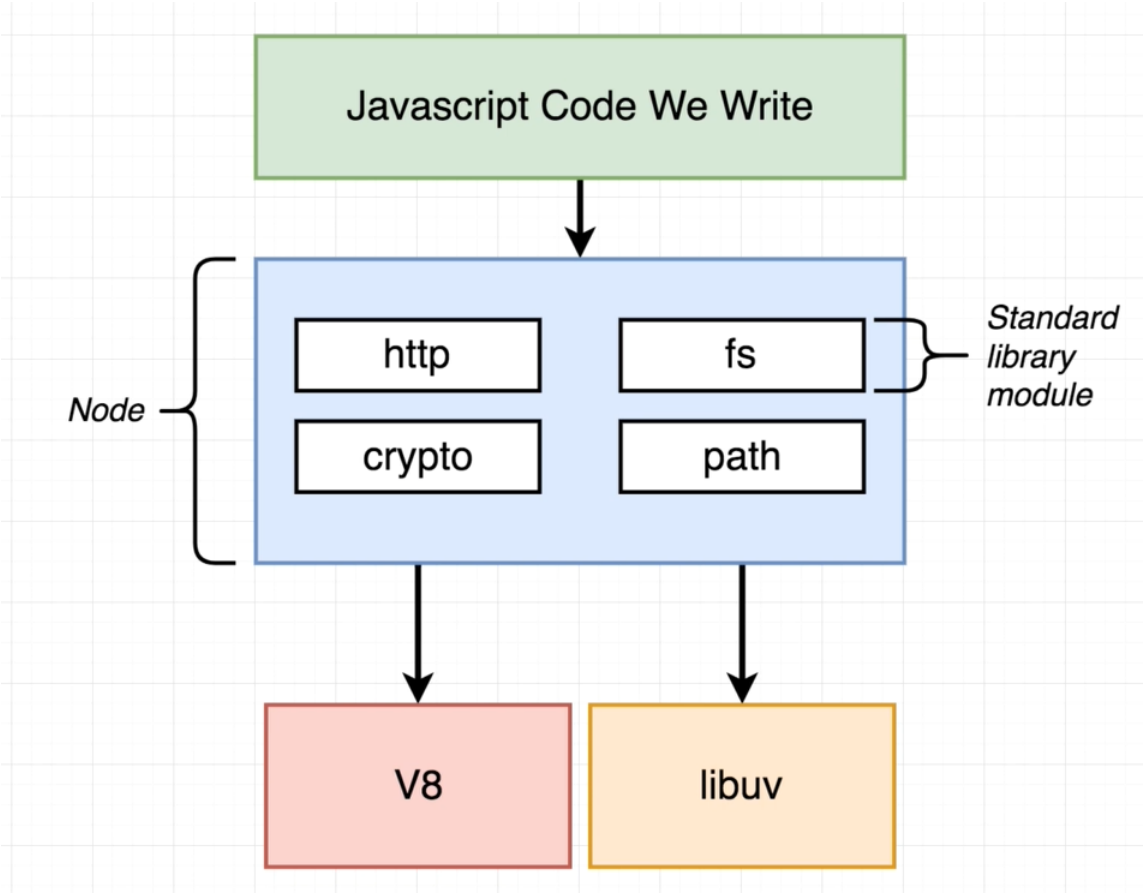
Internals of Node JS

- Node JS internally has a collection of dependencies that it uses to execute JS code
- Two of the most important dependencies are
  - V8 – open source JavaScript engine created by google
  - libuv – open source C++ project, which gives access to underlying filesystem, networking, threads and concurrency



Internals of Node JS

- Node JS provides a wrappers with very consistent APIs

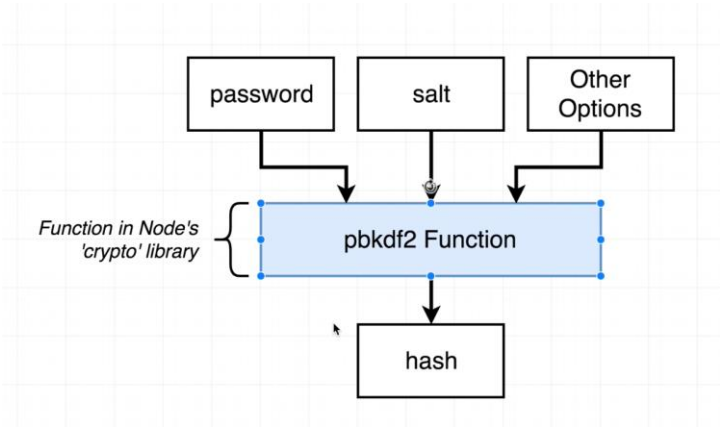


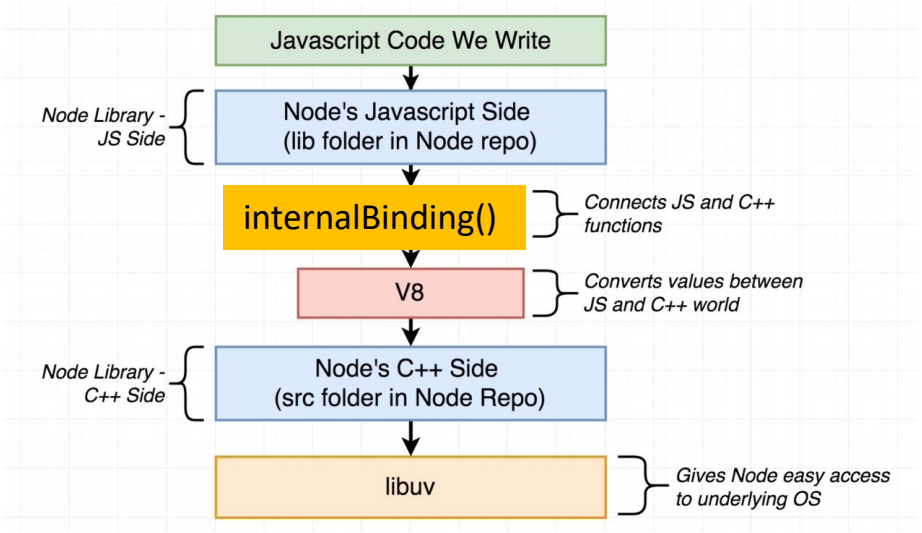
Internals of Node JS

- See how V8 and libuv are used to implement Node JS function
- https://github.com/nodejs/node/blob/master/lib/internal/crypto/pbkdf2.js

```
const {
  PBKDF2Job,
  kCryptoJobAsync,
  kCryptoJobSync,
} = internalBinding('crypto');
```

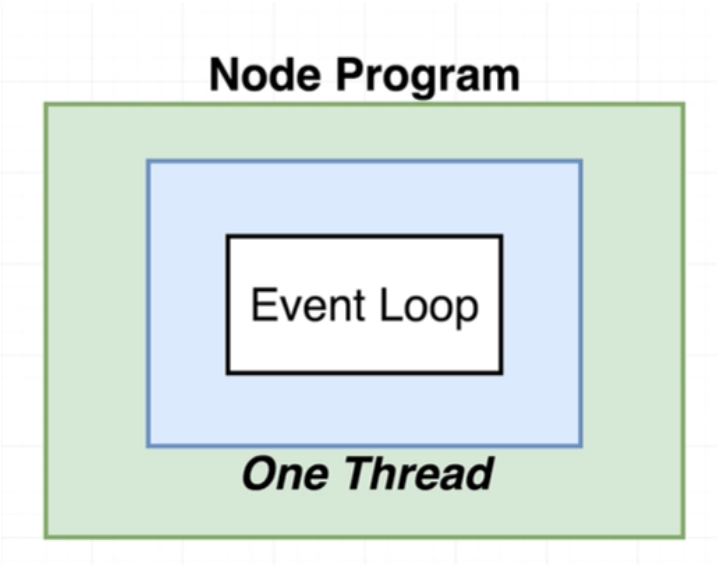
- https://github.com/nodejs/node/blob/master/src/node_crypto.cc





Node Event Loop

- Whenever Node process starts it creates a thread, inside that thread is a event loop.
- Think of event loop as a control structure where one thread should be doing at any given point of time
- Event loop is the core of how Node program runs.



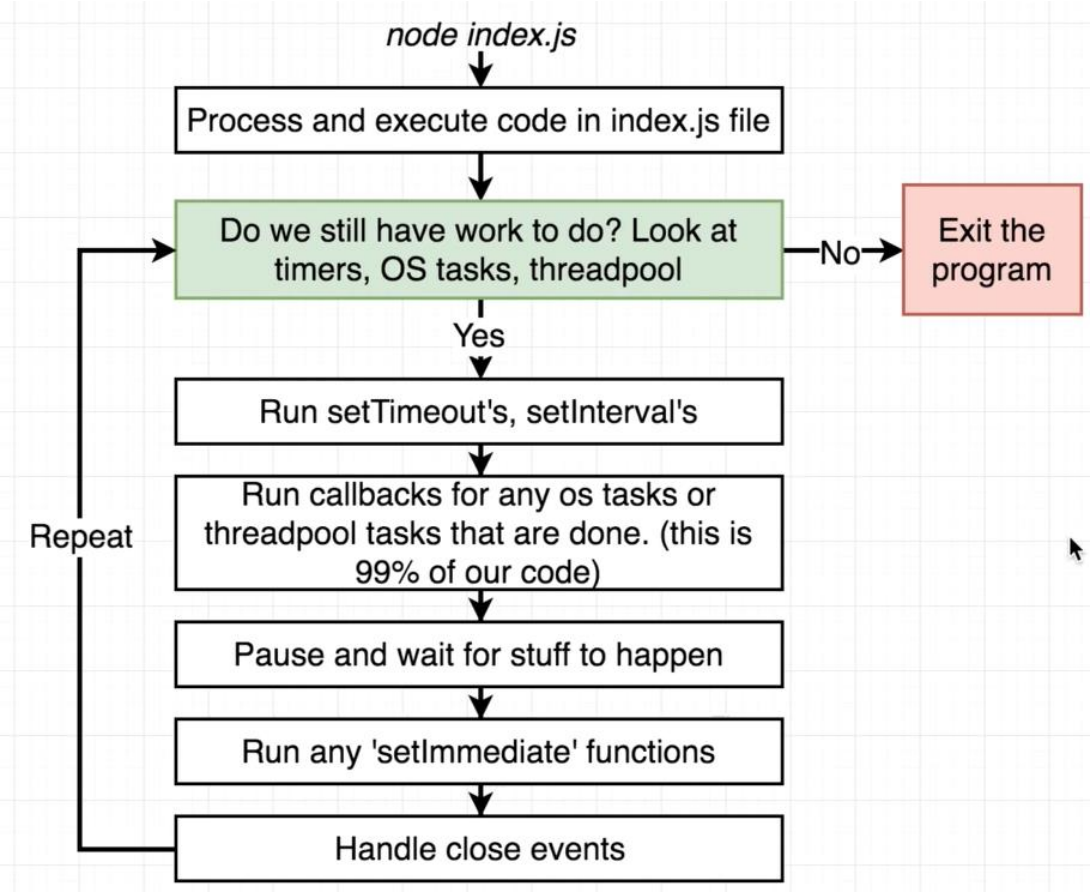
Event Loop pseudo code

```
// myFile.js

const pendingTimers = [];
const pendingOSTasks = [];
const pendingOperations = [];

// New timers, tasks, operations are recorded from myFile running
myFile.runContents();

function shouldContinue() {
    // Check One: Any Pending setTimeout, setInterval, setImmediate?
    // Check two: Any Pending OS task? (Listening to port)
    // Check three: Any pending long running operations? [ like fs module]
    return pendingTimers.length || pendingOSTasks.length || pendingOperations.length;
}
```

```
// entire body executes in one 'tick'
while(shouldContinue()) {
    // 1) Node looks at pendingTimers and sees if any functions
    // are ready to be called. setTimeout, setInterval

    // 2) Node looks at pendingOSTasks and pendingOperations
    // and calls relevant callbacks

    // 3) Pause execution. Continue when ..
    // - a new pendingOSTask is done
    // - a new pendingOperation is done
    // - a timer is about to complete

    // 4) Look at pendingTimers. setImmediate

    // 5) Handle any 'close' events
}
```

Event loop Review

- Each phase has a FIFO queue of callbacks to execute.
- When the event loop enters a given phase, it will perform any operations specific to that phase, then execute callbacks in that phase's queue until the queue has been exhausted or the maximum number of callbacks has executed.
- When the queue has been exhausted or the callback limit is reached, the event loop will move to the next phase, and so on.



process.nextTick() vs setImmediate

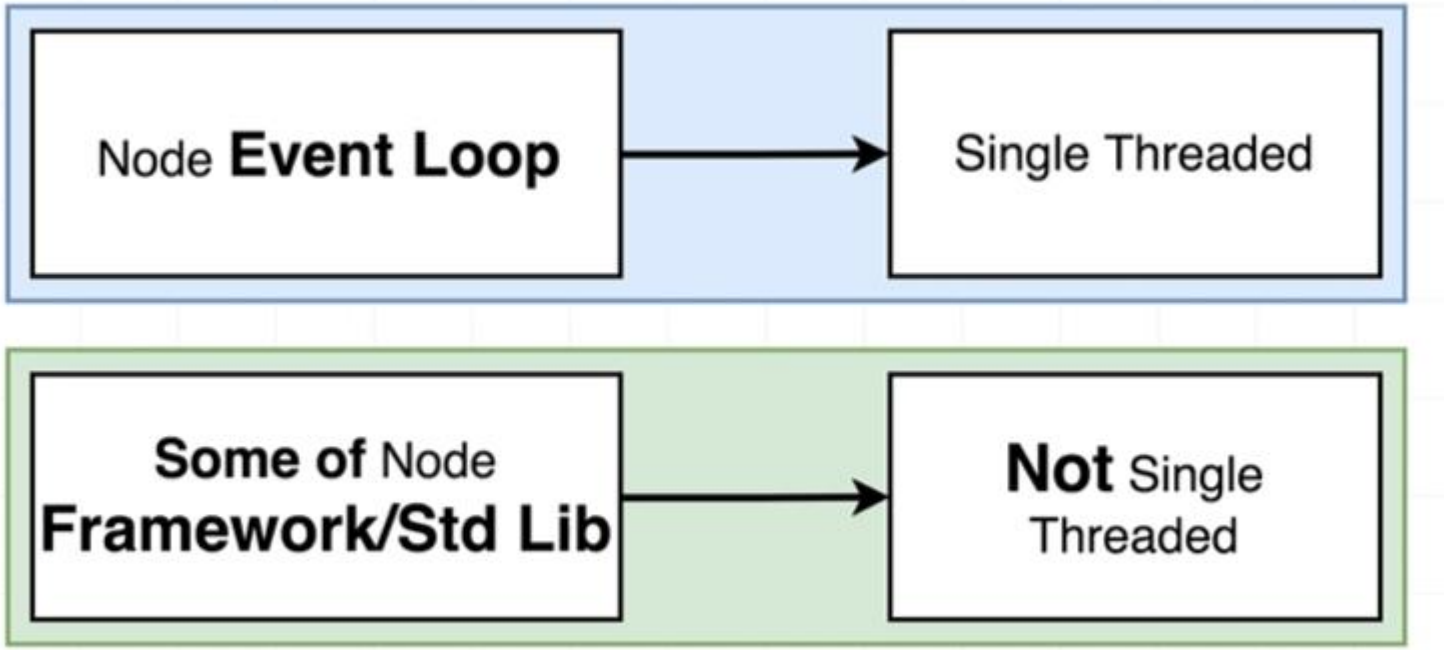
- process.nextTick() fires immediately on the same phase
- setImmediate() fires on the following iteration or 'tick' of the event loop

```
process.nextTick(()=>console.log("next 1"));

setImmediate(() => {
console.log("immediate 1")
});

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

process.nextTick(()=>console.log("next 2"));

console.log("hey");

setImmediate(() => {
    console.log("immediate 2")
});
```

```
hey
next 1
next 2
timeout
immediate 1
immediate 2
```

Threads

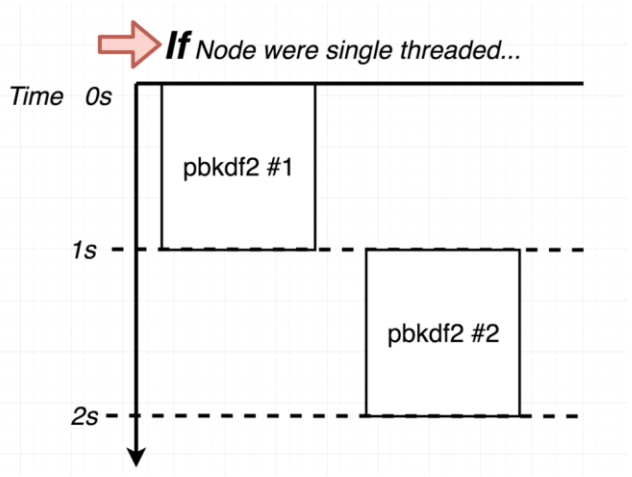
- Threads in Node JS

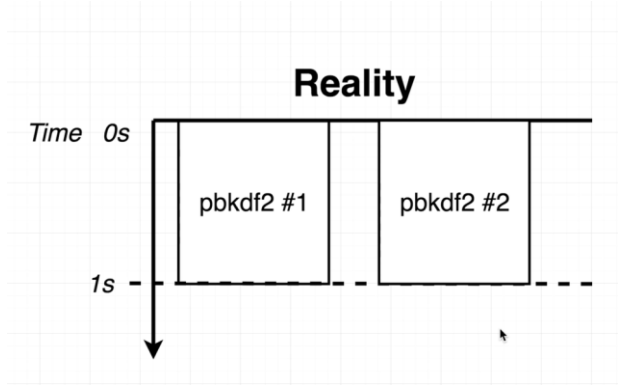


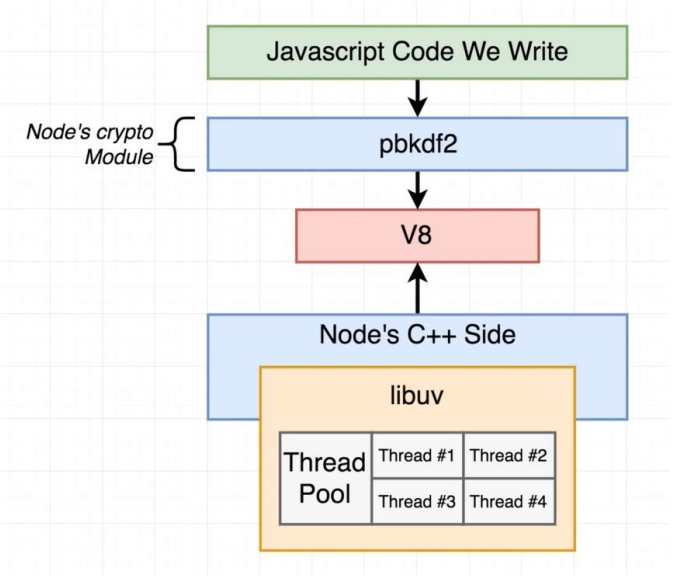
Threads

```
const crypto = require('crypto');

const start = Date.now();

crypto.pbkdf2('secret','saltvalue',100000, 512,'sha512', () => {
    console.log("1: ", Date.now() - start  );
});

crypto.pbkdf2('secret','saltvalue',100000, 512,'sha512', () => {
    console.log("2: ", Date.now() - start  );
});
```

```
node threads.js

1:  809
2:  819
```



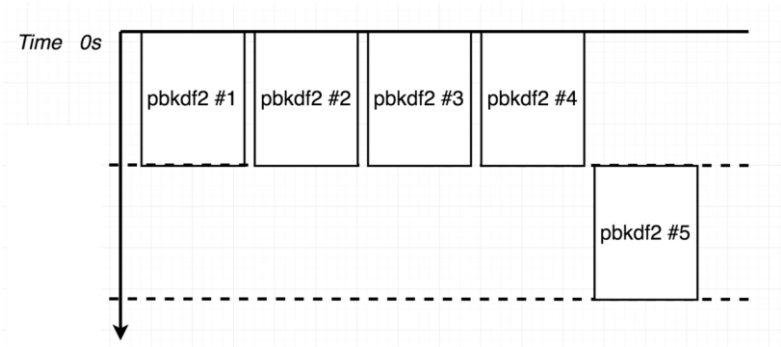




Thread Pool

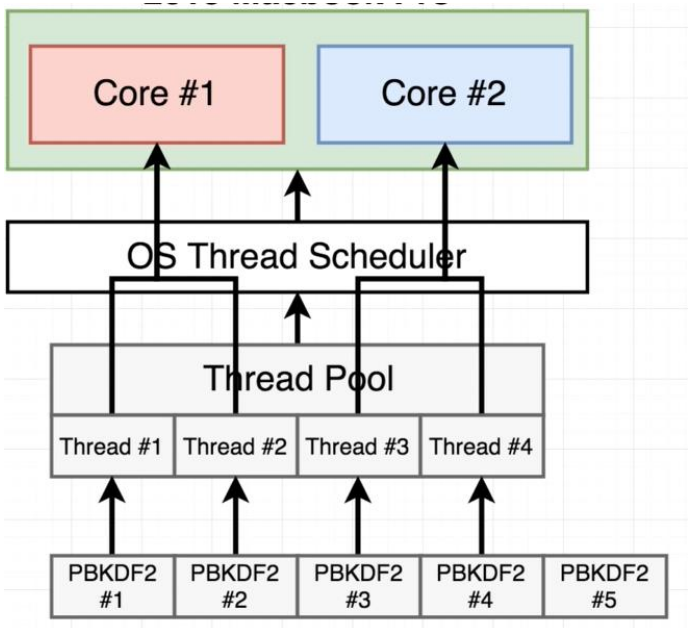
```
const crypto = require('crypto');

const start = Date.now();

crypto.pbkdf2('secret','saltvalue',100000, 512,'sha512', () => {
    console.log("1: ", Date.now() - start  );
});

crypto.pbkdf2('secret','saltvalue',100000, 512,'sha512', () => {
    console.log("2: ", Date.now() - start  );
});

crypto.pbkdf2('secret','saltvalue',100000, 512,'sha512', () => {
    console.log("3: ", Date.now() - start  );
});

crypto.pbkdf2('secret','saltvalue',100000, 512,'sha512', () => {
    console.log("4: ", Date.now() - start  );
});
crypto.pbkdf2('secret','saltvalue',100000, 512,'sha512', () => {
    console.log("5: ", Date.now() - start  );
});
```

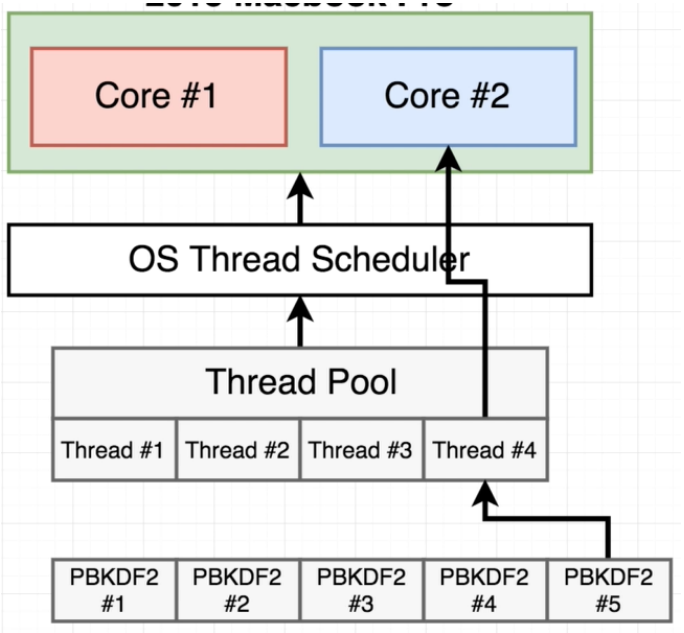
```
node threads.js

4:  1330
1:  1349
3:  1350
2:  1356
5:  2005
```



Thread Pool

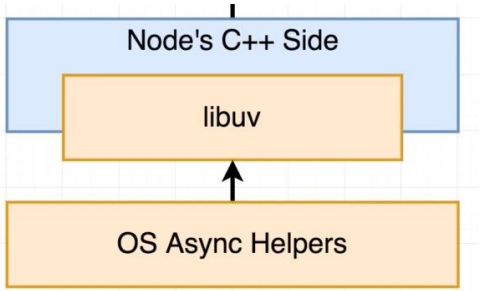
- Each core has more threads to handle, hence more time compared to previous execution with only 2 threads





Async operations

- Some function calls in NodeJS are delegated directly to the underlying OS outside of the thread pool



- Almost everything around networking uses OS's async features.
- Tasks using the underlying OS are reflected in our 'pendingOSTasks' array

```
const https = require('https');

const start = Date.now();

function doRequest() {
https.request('https://www.google.com', res => {
    res.on('data', () => {});
    res.on('end', () => {
        console.log(Date.now() - start);
    })
}).end();
}

doRequest();        220
doRequest();        227
doRequest();        232
doRequest();        237
doRequest();        242
doRequest();        244
doRequest();        245
doRequest();        246
```

Multi Tasking

```
const https = require('https');
const crypto = require('crypto');
const fs = require('fs');

const start = Date.now();

function doRequest() {
    https.request('https://www.google.com', res => {
        res.on('data', () => { });
        res.on('end', () => {
            console.log("HTTP: " , Date.now() - start);
        })
    }).end();
}

function doHash() {
    crypto.pbkdf2('secret', 'saltvalue', 100000, 512, 'sha512', () => {
        console.log("HASH: ", Date.now() - start);
    });
}
```

```
doRequest();

fs.readFile('multiTask.js', () => {
    console.log("FS: ", Date.now() - start);
});

doHash();
doHash();
doHash();
doHash();
```

```
HTTP:  254
HASH:  1454
FS:  1455
HASH:  1459
HASH:  1462
HASH:  1476
```

```
doRequest();

fs.readFile('multiTask.js', () => {
    console.log("FS: ", Date.now() - start);
});

// doHash();
// doHash();
// doHash();
// doHash();
```

```
FS:  84
HTTP:  387
```

Multi-tasking