NodeJS Advance 2

1. Closure
2. EventEmitter
3. Callback
4. Promise
5. Async/Await
6. Eventloop

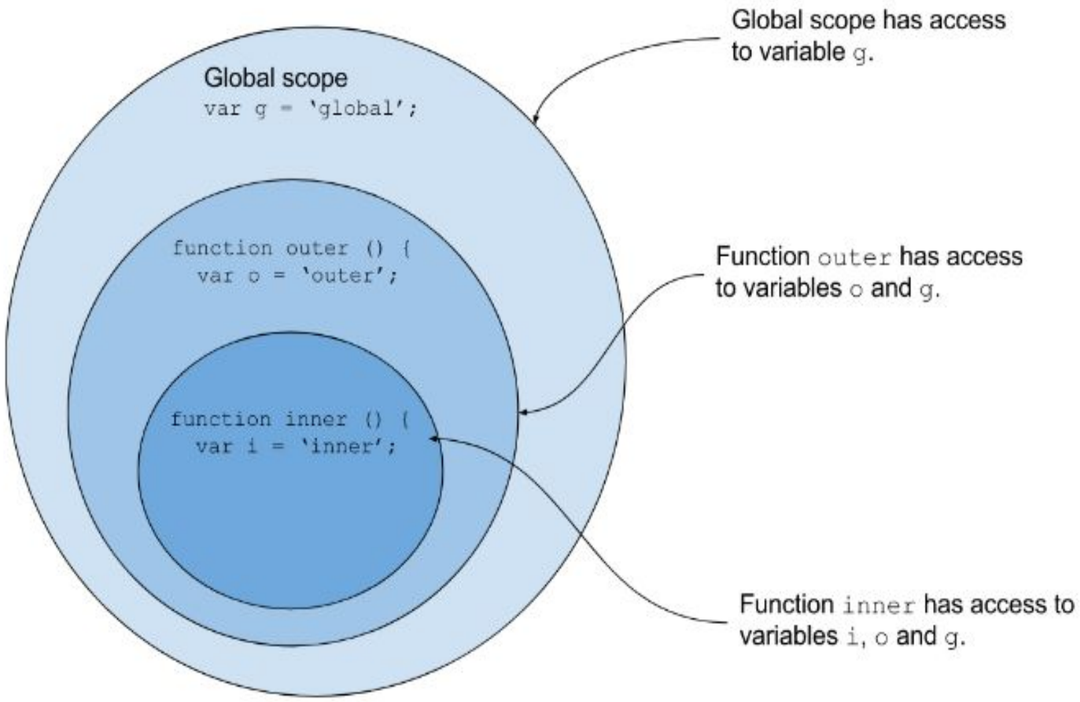
1. Closure

- Closure là hàm được khai báo trong hàm.
- Closure có thể tự động nhớ **outer variables** và có thể truy cập chúng.

```
1  // //Closure
2  function outer() {
3    let index = 0;
4    return function inner() {
5      index++;
6      console.log({ index });
7    };
8  }
9
10 const innerFunc = outer();
11 const innerFunc_2 = outer();
12 innerFunc();
13 innerFunc();
14 innerFunc();
15
```

1. Closure

- Lexical scope: function trong có quyền truy cập tới scope của function bên ngoài.



2. EventEmitter

- The EventEmitter is a module that facilitates communication/interaction between objects in Node
- It allows you to listen for events and assign actions to run when those events occur

Class: EventEmitter

The `EventEmitter` class is defined and exposed by the `events` module:

```
const EventEmitter = require('events');
```

```
var events = require('events');
var eventEmitter = new events.EventEmitter();
```

2. EventEmitter methods

EventEmitter Methods	Description
emitter.addListener(event, listener)	Adds a listener to the end of the listeners array for the specified event. No checks are made to see if the listener has already been added.
emitter.on(event, listener)	Adds a listener to the end of the listeners array for the specified event. No checks are made to see if the listener has already been added. It can also be called as an alias of emitter.addListener()
emitter.once(event, listener)	Adds a one time listener for the event. This listener is invoked only the next time the event is fired, after which it is removed.
emitter.removeListener(event, listener)	Removes a listener from the listener array for the specified event. Caution: changes array indices in the listener array behind the listener.
emitter.removeAllListeners([event])	Removes all listeners, or those of the specified event.
emitter.setMaxListeners(n)	By default EventEmitters will print a warning if more than 10 listeners are added for a particular event.
emitter.getMaxListeners()	Returns the current maximum listener value for the emitter which is either set by emitter.setMaxListeners(n) or defaults to EventEmitter.defaultMaxListeners.
emitter.listeners(event)	Returns a copy of the array of listeners for the specified event.
emitter.emit(event[, arg1][, arg2][, ...])	Raise the specified events with the supplied arguments.
emitter.listenerCount(type)	Returns the number of listeners listening to the type of event.

3. Callback

Là hàm (function) được truyền qua đối số khi gọi hàm khác.
1, Là hàm
2, Được truyền qua đối số
3, Được gọi lại (trong hàm nhận đối số)

```
function myFunction(param) {
  param("Team 2")
}

function callbackFunc(value) {
  console.log('value: ', value)
}

myFunction(callbackFunc)
```

3. Callback hell

- Đây là hiện tượng một chuỗi các callback được lồng vào nhau liên tiếp, dẫn đến việc khó theo dõi và debug code.
- Chúng ta có thể khắc phục điều này bằng Promise

```
asyncFunc('something', function (err, data) {
    asyncFunc('something', function (err, data) {
        asyncFunc('something', function (err, data) {
            asyncFunc('something', function (err, data) {
                asyncFunc('something', function (err, data) {
                    asyncFunc('something', function (err, data) {
                        asyncFunc('something', function (err, data) {
                            asyncFunc('something', function (err, data) {
                                // do the final action
                            });
                        });
                    });
                });
            });
        });
    });
});
```

4. Promise

- Đây là hiện tượng một chuỗi các callback được lồng vào nhau liên tiếp, dẫn đến việc khó theo dõi và debug code.
- Chúng ta có thể khắc phục điều này bằng Promise

```
asyncFunc('something', function (err, data) {
    asyncFunc('something', function (err, data) {
        asyncFunc('something', function (err, data) {
            asyncFunc('something', function (err, data) {
                asyncFunc('something', function (err, data) {
                    asyncFunc('something', function (err, data) {
                        asyncFunc('something', function (err, data) {
                            asyncFunc('something', function (err, data) {
                                // do the final action
                            });
                        });
                    });
                });
            });
        });
    });
});
```

4. Promise

Promise là một đối tượng được sử dụng cho tính toán bất đồng bộ. Một promise đại diện cho một tiến trình hay một tác vụ chưa thể hoàn thành ngay được.

Trong tương lai, promise sẽ trả về giá trị hoặc là đã được giải quyết (resolve) hoặc là không (reject).

Promise được sử dụng để xử lý các hoạt động bất đồng bộ trong JavaScript. Chúng dễ quản lý khi xử lý nhiều hoạt động bất đồng bộ so với các lệnh callback

4. Promise

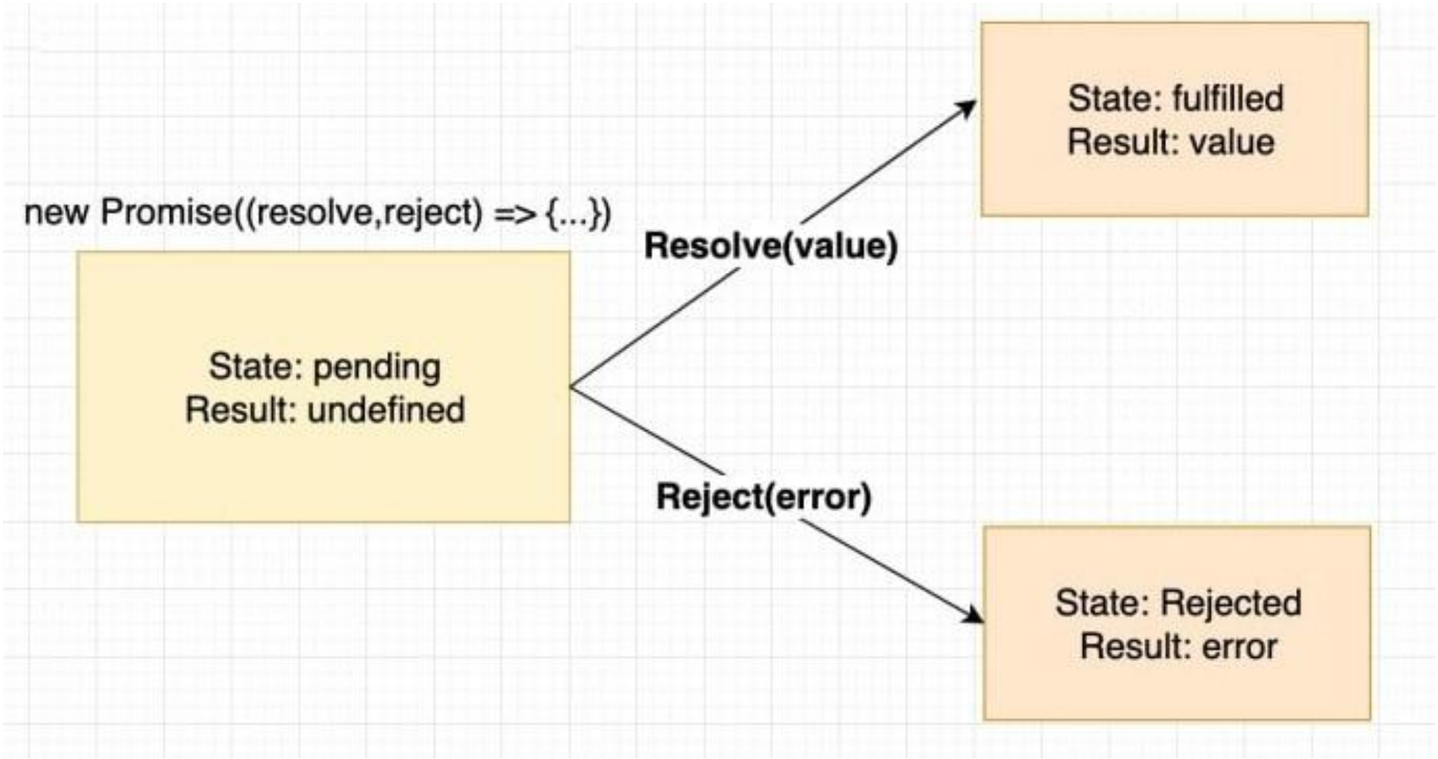
```
var promise = new Promise (
  function(resolve,reject){
    var randomNumber = Math.random();
    setTimeout(() => {
    if(randomNumber < .7) {
          resolve('success');
          } else {
          reject('error');
        }
      }, 2000);
    console.log(randomNumber);
  }
);

promise
.then(function(message){
  console.log(message);
})
.catch(function(error){
  console.log(error);
})
.finally(function(){
  console.log("Done");
})
```

Parameters

- Hàm tạo Promise chỉ nhận một đối số là một callback function.
- Callback function nhận hai đối số, resolve và reject.
- Thực hiện các thao tác bên trong callback function và nếu mọi thứ suôn sẻ thì hãy gọi resolve.
- Nếu các hoạt động mong muốn xảy ra lỗi thì hãy gọi reject.

4. Promise’s states

- Pending - `Promise` đang chờ xử lý nếu kết quả chưa sẵn sàng. Khi đó, nó đang chờ một thứ gì đó kết thúc (Ví dụ hoạt động bất đồng bộ).
- Resolved or Fulfilled - `Promise` được giải quyết nếu có kết quả.
- Rejected - `Promise` bị từ chối nếu xảy ra lỗi



4. Promise’s states

Promise Chain

Các phương thức `then()` và `catch()` cũng có thể trả về một `promise` mới có thể được xử lý bằng cách xâu chuỗi then () khác vào cuối phương thức `then()` trước đó.

Promise.all ()

Phương thức này nhận một mảng các lời hứa làm đầu vào và trả về một lời hứa mới thực hiện khi tất cả các lời hứa bên trong mảng đầu vào đã hoàn thành hoặc từ chối ngay khi một trong các lời hứa trong mảng từ chối

Promise.race ()

Phương thức này nhận một mảng các `promise` làm đầu vào và trả về một `promise` mới thực hiện ngay khi một trong các `promise` trong mảng đầu vào thực hiện hoặc từ chối ngay khi một trong các `promise` trong mảng đầu vào từ chối.

5. Async/await

- Là một tính năng của JavaScript giúp làm việc với các hàm bất đồng bộ một cách dễ dàng hơn. Nó được xây dựng trên Promise và tương thích với tất cả API dựa trên Promise.
- Async - khai báo một hàm bất đồng bộ

```
async function nameFunction() {
    // code
}
```

- Await - tạm dừng việc thực hiện các hàm async

```
async function nameFunction() {
    await func()
}
```

5. Async/await

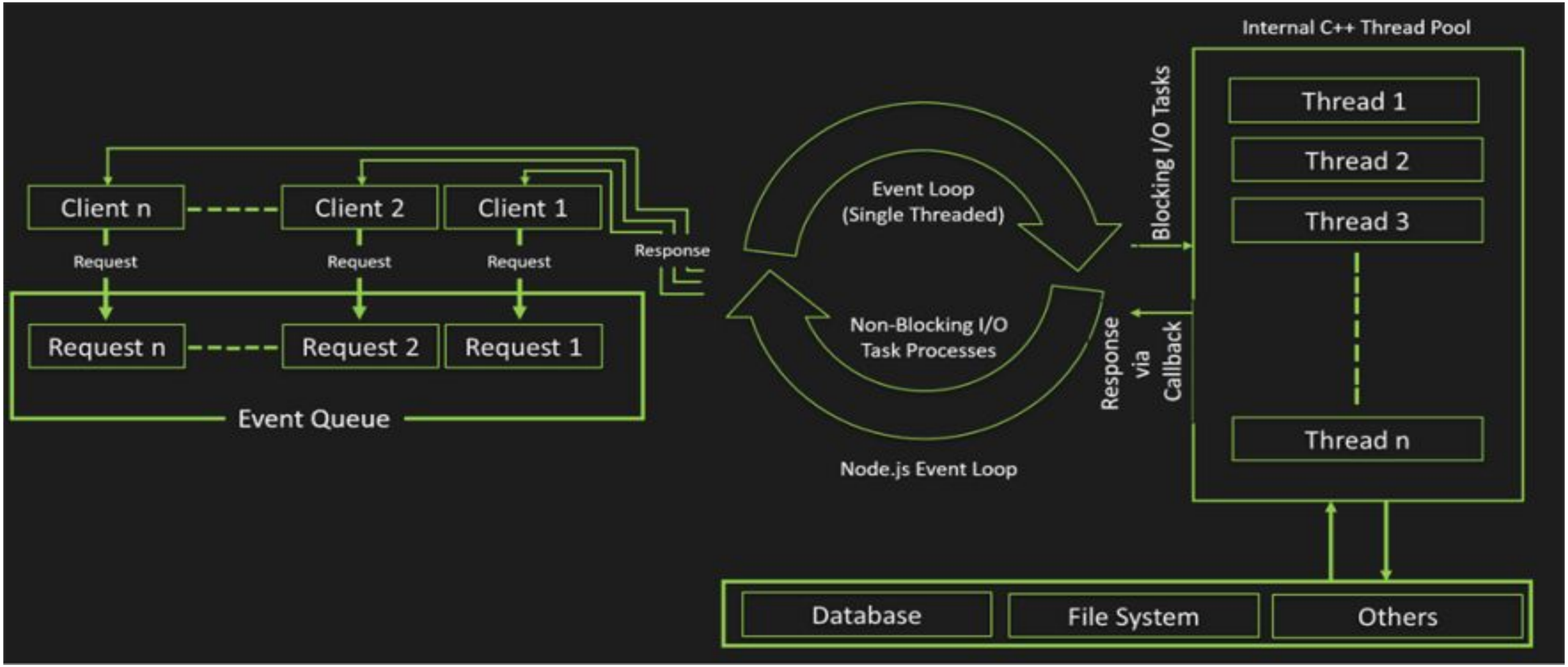
- **Async**
  - Tự động chuyển đổi func thông thường thành Promise
  - Async func cho phép sử dụng await

- **Await**
  - Await chỉ làm việc với Promise, không làm việc với Callback
  - Khi đặt trước một Promise, await sẽ đợi promise kết thúc và trả ra kết quả
  - Await chỉ có thể sử dụng khi đặt trong async func

6. Event loop

NodeJS là một ứng dụng đơn luồng (Single Thread), nó hoạt động phía trên một nền tảng được viết bởi C++, nền tảng này sử dụng đa luồng (Multi-Thread) để thực hiện đồng thời các nhiệm vụ.

6. Event loop


Client n
Client 2
Client 1
Request
Request
Request
Response
Request n
Request 2
Request 1
Event Queue
Event Loop
(Single Threaded)
Non-Blocking I/O
Task Processes
Node.js Event Loop
Blocking I/O Tasks
Response
via
Callback
Internal C++ Thread Pool
Thread 1
Thread 2
Thread 3
Thread n
Database
File System
Others

6. Event loop


PHP 1 thread
req1
validate req 1
db
res 1
req2
validate req 2
db
res 2
req3
validate req 3
db
res 3
req 1
validate req 1
db
res 1
req 2
validate req 2
db
res 2
req 3
validate req 3
db
res 3
event loop thread
validate req 1
db
validate req 2
db
res 1
validate req 3
res 2
db
res 3
db
db
db
db
db
db
db
db
db