JS Asynchronous

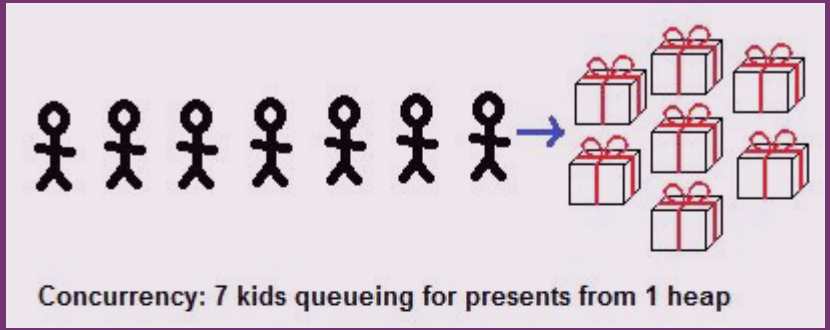
Isumi
Batam, Jan 2020

GOALS

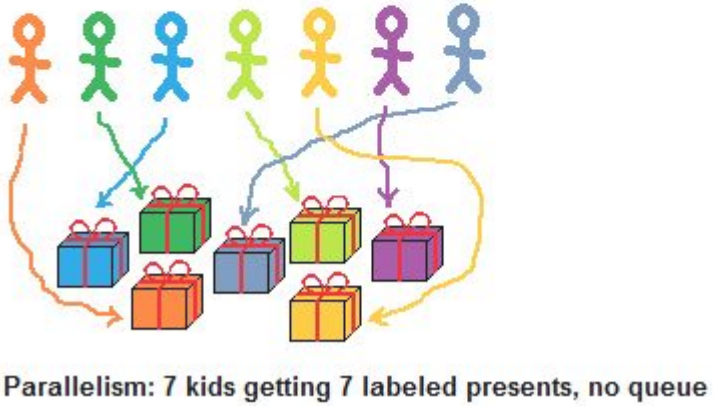
1. Asynchronous Concept
2. Callback
3. Promise
4. Async await

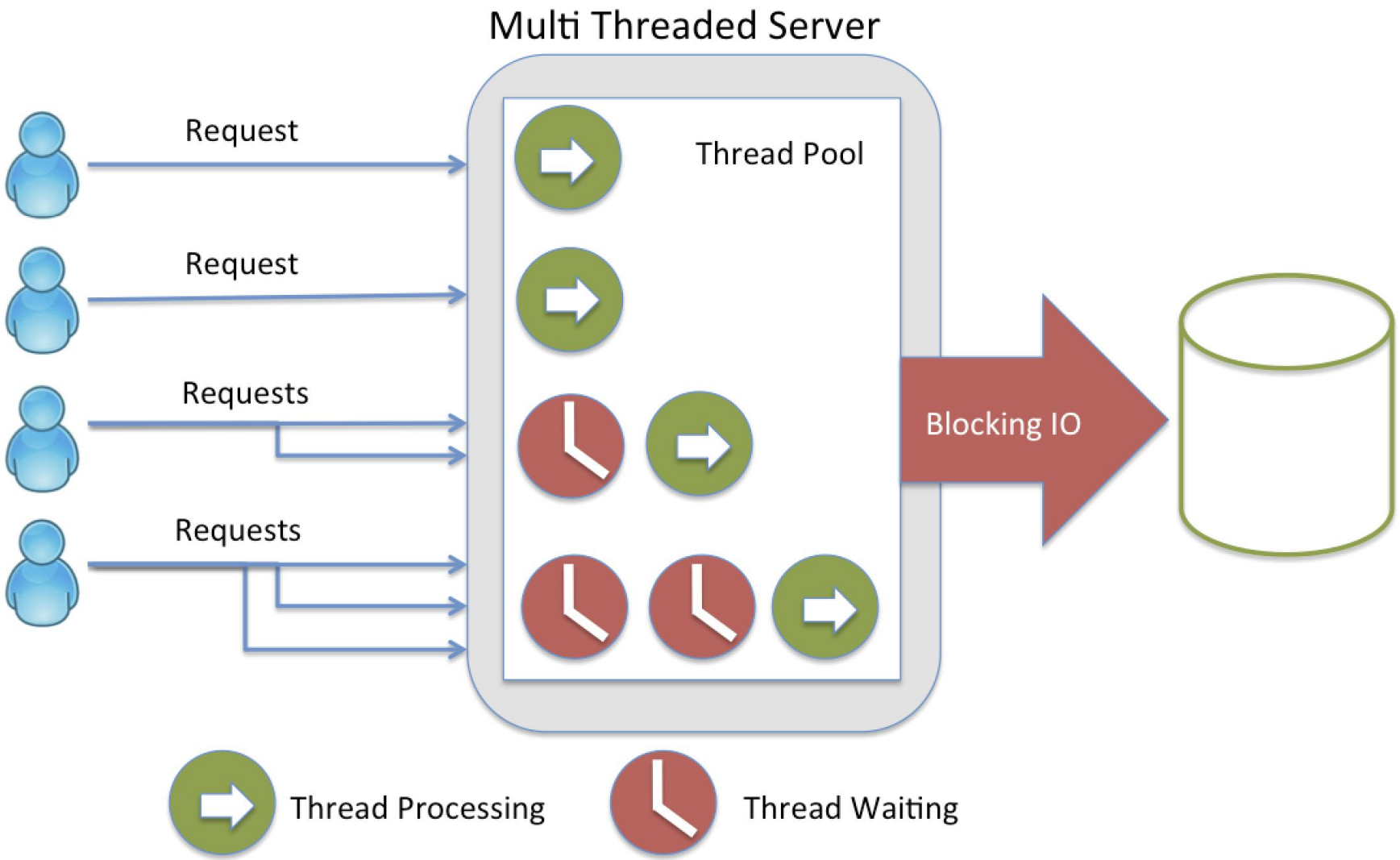
Oops wait...
take a look...

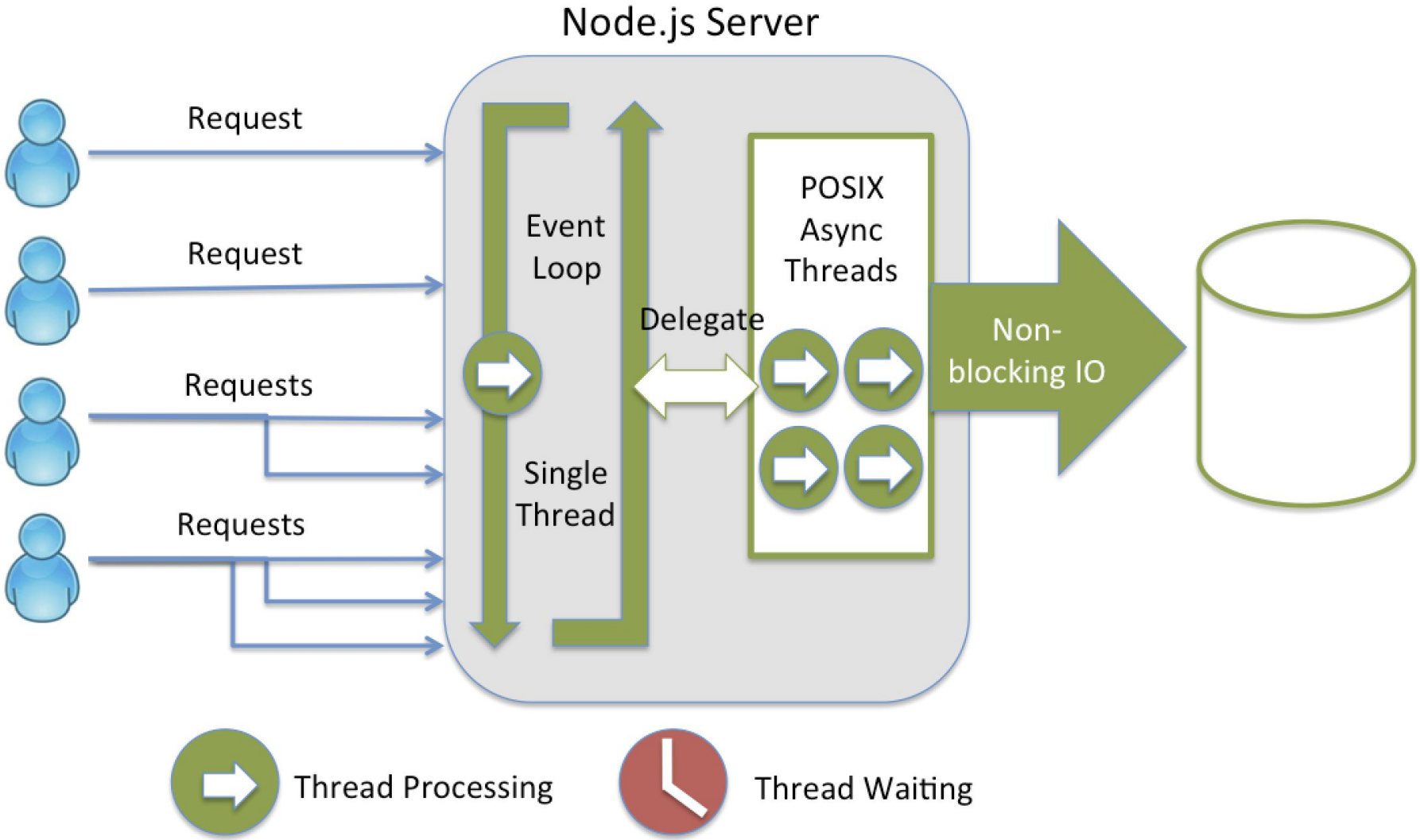
Concurrent



Parallel




Multi Threaded Server
Request
Request
Requests
Requests
Thread Pool
Blocking IO
Thread Processing
Thread Waiting


Node.js Server
Request
Request
Requests
Requests
Event
Loop
Single
Thread
Delegate
POSIX
Async
Threads
Non-
blocking IO
Thread Processing
Thread Waiting

```
// Stack In Action

function multiply(a, b) {
    return a * b
}
function square(num) {
    return multiply(num, num)
}
function printSquare(num) {
    let squared = square(num)
    console.log(squared)
}

printSquare(4)
```

Stack

- multiply(4, 4)
- console.log(16)
- printSquare(4)
- main()

Elements Console Sources Network Timeline

top Preserve log

```
> function baz() {
      throw new Error("Oops! I did it again!!")
  }

  function zinga() {
    baz()
  }

  function sheldon() {
    zinga()
  }

  sheldon()
```

```
Uncaught Error: Oops! I did it again!!(…)
    baz                 @ VM1294:2
    zinga               @ VM1294:6
    sheldon             @ VM1294:10
    (anonymous function) @ VM1294:13
```

The Stack Trace

```
// Synchronous Way

let sui = fs.readFileSync("sui.txt")
let lhx = fs.readFileSync("lhx.dat")
let sup = fs.readFileSync("sup.mpg")

console.log(sui)
console.log(lhx)
console.log(sup)
```

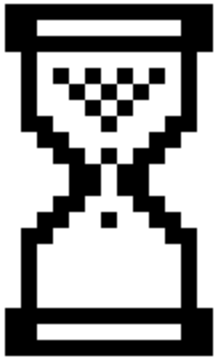
Stack

main()

```
// Synchronous Way

let sui = fs.readFileS
let lhx = fs.readFileS
let sup = fs.readFileS

console.log(sui)
console.log(lhx)
console.log(sup)
```



Stack

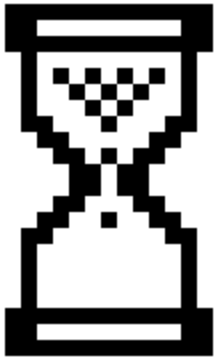
readFileSync(“sui.txt”)

main()

```
// Synchronous Way

let sui = fs.readFileS
let lhx = fs.readFileS
let sup = fs.readFileS

console.log(sui)
console.log(lhx)
console.log(sup)
```



Stack

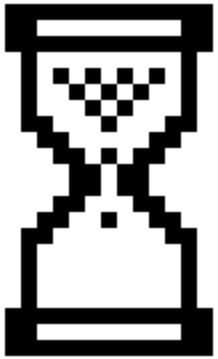
readFileSync("lhx.dat")

main()

```
// Synchronous Way

let sui = fs.readFileS
let lhx = fs.readFileS
let sup = fs.readFileS

console.log(sui)
console.log(lhx)
console.log(sup)
```



Stack

readFileSync(“sup.mpg”)

main()

```
// Synchronous Way

let sui = fs.readFileSync("sui.txt")
let lhx = fs.readFileSync("lhx.dat")
let sup = fs.readFileSync("sup.mpg")

console.log(sui)
console.log(lhx)
console.log(sup)
```

Stack

console.log(sui)

main()

```
// Synchronous Way

let sui = fs.readFileSync("sui.txt")
let lhx = fs.readFileSync("lhx.dat")
let sup = fs.readFileSync("sup.mpg")

console.log(sui)
console.log(lhx)
console.log(sup)
```

Stack

console.log(lhx)

main()

```
// Synchronous Way

let sui = fs.readFileSync("sui.txt")
let lhx = fs.readFileSync("lhx.dat")
let sup = fs.readFileSync("sup.mpg")

console.log(sui)
console.log(lhx)
console.log(sup)
```

Stack

```
console.log(sup)
main()
```

```
// Synchronous Way

let sui = fs.readFileSync("sui.txt")
let lhx = fs.readFileSync("lhx.dat")
let sup = fs.readFileSync("sup.mpg")

console.log(sui)
console.log(lhx)
console.log(sup)
```

Stack

main()

This kind of program is soooo sulooooowwww....

JavaScript is concurrent, we should embrace it!

```
// Async Way

console.log("Hi Students!")

fs.readFile("lhx.csv", function cb(err, data) {
  console.log(data)
})

console.log("Hello World!")
```

Stack

main()

APIs

Console

event loop

task
queue

```
// Async Way

console.log("Hi Students!")

fs.readFile("lhx.csv", function cb(err, data) {
  console.log(data)
})

console.log("Hello World!")
```

Stack

console.log()

main()

APIs

Console

"Hi Students!"

event loop

task
queue

```
// Async Way

console.log("Hi Students!")

fs.readFile("lhx.csv", function cb(err, data) {
  console.log(data)
})

console.log("Hello World!")
```

Stack

main()

APIs

Console

"Hi Students!"

event loop

task queue

```
// Async Way

console.log("Hi Students!")

fs.readFile("lhx.csv", function cb(err, data) {
  console.log(data)
})

console.log("Hello World!")
```

Stack

```
readFile("l", cb)
main()
```

APIs

⏳Open The File `cb`

Console

"Hi Students!"

event loop

task queue

```
// Async Way

console.log("Hi Students!")

fs.readFile("lhx.csv", function cb(err, data) {
  console.log(data)
})

console.log("Hello World!")
```

Stack

console.log()

main()

APIs

⏳Open The File **cb**

Console

"Hi Students!"

"Hello World!"

event loop

task queue

```
// Async Way

console.log("Hi Students!")

fs.readFile("lhx.csv", function cb(err, data) {
  console.log(data)
})

console.log("Hello World!")
```

Stack

APIs

⏳ Open The File **cb**

Console

"Hi Students!"

"Hello World!"

event loop

task queue

cb

```
// Async Way

console.log("Hi Students!")

fs.readFile("lhx.csv", function cb(err, data) {
  console.log(data)
})

console.log("Hello World!")
```

Stack

cb

APIs

Console

"Hi Students!"

"Hello World!"

"Contents of data…"

event loop

task
queue

```
// Async Way

console.log("Hi Students!")

fs.readFile("lhx.csv", function cb(err, data) {
  console.log(data)
})

console.log("Hello World!")
```

Stack

APIs

Console

"Hi Students!"

"Hello World!"

"Contents of data…"

event loop

task
queue

JavaScript is Concurrent

Blocking Pseudocode

```
Read file from Filesystem, set equal to “contents”
Print contents
Do something else
```

Non-Blocking Pseudocode

This is a “callback”

```
Read file from Filesystem
    Whenever you’re complete, Print the contents
Do something else
```

JavaScript is Concurrent

Blocking Code

```
let contents = fs.readFileSync("/etc/hosts")
console.log(contents)
console.log("Doing something else")
```

This is a “callback”

Non-Blocking Pseudocode

```
fs.readFile("/etc/hosts/", (err, contents) => {
    console.log(contents)
});
console.log("Doing something else");
```

Async Patterns in JavaScript

BINAR ACADEMY

1. Asynchronous Concept

```
console.log('First');
console.log('Second');
console.log('Third');
```

Synchronous

```
console.log('First');
setTimeout(function(){
  console.log('second');
},0);
console.log('Third');
```

Asynchronous

2. Callback

Function that is passed to another function as a parameter, and the callback function is executed inside the function. Using callback in jQuery:

```
$(".button").click(function() {
    console.log("I'm clicked");
});
```

```
var values = ['6 usd', '6 usd', '6 usd', '6 usd']
    .map(function(value) {
        return parseInt(value);
    });
console.log(values); //result ->  [6, 6, 6, 6]
```

Usually, callbacks are only used when doing Input/Output, like downloading things, reading files, talking to databases, etc.

Callback hell?


```
foo(() => {
    bar(() => {
        baz(() => {
            qux(() => {
                quux(() => {
                    quuz(() => {
                        corge(() => {
                            grault(() => {
                                run();
                            }).bind(this);
                        }).bind(this);
                    }).bind(this);
                }).bind(this);
            }).bind(this);
        }).bind(this);
    }).bind(this);
}).bind(this);
```

Callback hell is caused by poor coding practices. Luckily writing better code isn't that hard! You can get out of callback hell by doing some better practice like Modularizing code and handling errors can save you a lot.

BINAR ACADEMY

Practice

```javascript
//Callback
const UserProfile = function(id, callback) { // callback atau cb
    if (!id) {
        return callback(new Error('Invalid userId')) // syntax to create error
    }
    let result = {
        success: true,
        id: id,
        message: 'User Found'
    }
    return callback(null, result)
}

UserProfile(10, function(err, result) {
    if (err) {
        console.log(err.message)
    }
    console.log(result) // guard clause
})
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
(base) isumi@isumizumi:~/Documents/BinarAcademy/GA/KaryaSiswa$ node ExampleCb.js
{ success: true, id: 10, message: 'User Found' }
```

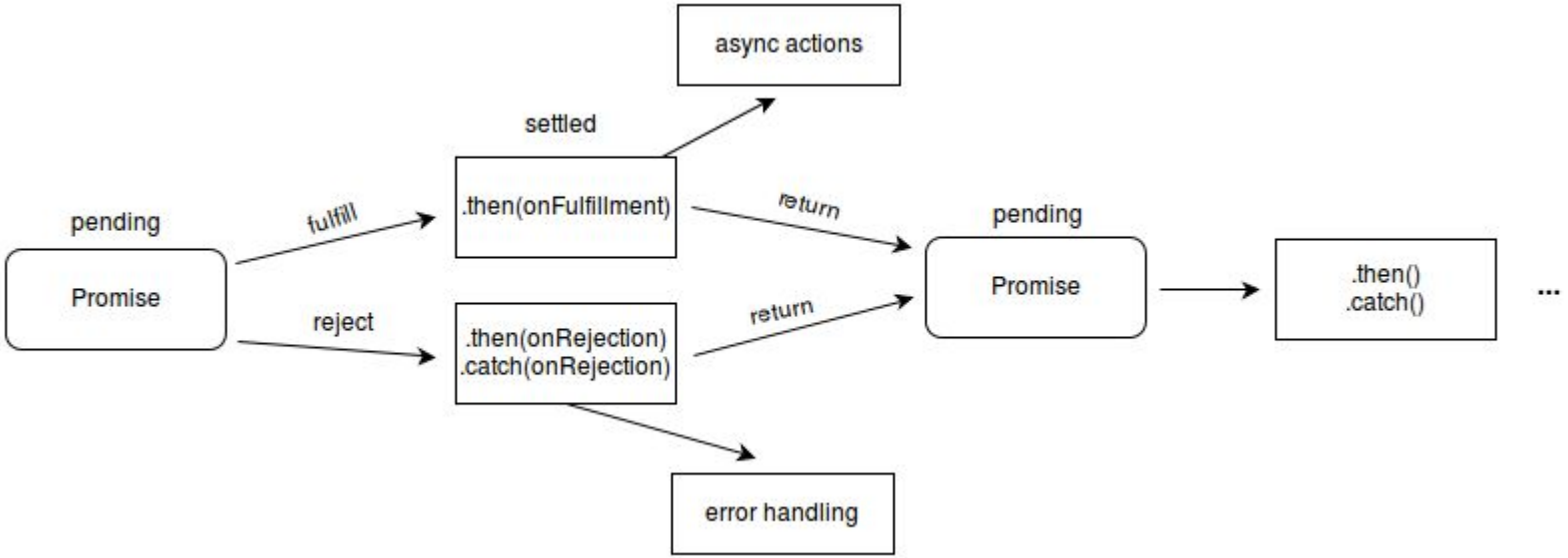
BINAR ACADEMY

3. Promise

Promises help you naturally handle errors, and write cleaner code by not having callback parameters, and without modifying the underlying architecture.

```
var myPromise = new Promise(function(resolve, reject) {
  // Do an async task async task and then...
 if(/* good condition */) {
   resolve('Success!');
 } else {
    reject('Failure!');
 }
});
myPromise.then(function() {
  /* do something with the result */
}).catch(function() {
  /* error */
})
```

How to Promise work



Let’s practice .then(‘we can know the output’)

```
let myPromise = new Promise(function(resolve, reject) {
        setTimeout(function() {
            resolve(11)
        }, 3000)
    })
    .then(function(num) {
        console.log('first', num)
        return num * 2
    })
    .then(function(num) {
        console.log('second', num)
        return num * 2
    })
    .then(function(num) {
        console.log('third', num)
        return num * 2
    })
```

BINAR ACADEMY

Practice

```javascript
// Promise
const UserProfile = function(id) { // parameter

    // statement
    return (new Promise(function(resolve, reject) {
        let response = {
            success: true,
            id: id,
            message: 'User Found'
        }
        if (id) {
            resolve(response)
        }
        reject(new Error('Invalid userId'))
    }))
}

UserProfile(1).then((result) => { // then when successed (resolve)
    console.log(result)
}).catch((err) => { // catch when failed (reject)
    console.log('User Not Found')
});
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
(base) isumi@isumizumi:~/Documents/BinarAcademy/GA/KaryaSiswa$ node ExamplePromise2.js
{ success: true, id: 1, message: 'User Found' }
```

BINAR ACADEMY

4. Async await

Async functions return a Promise.

If the function throws an error, the Promise will be rejected.

If the function returns a value, the Promise will be resolved.

```javascript
function doubleAfter2Seconds(x) {
    return new Promise(resolve => {
        setTimeout(() => {
            resolve(x * 2);
        }, 2000);
    });
}

async function addAsync(x) {
    const a = await doubleAfter2Seconds(10);
    const b = await doubleAfter2Seconds(20);
    const c = await doubleAfter2Seconds(30);
    return x + a + b + c;
}


addAsync(10).then((sum) => {
    console.log(sum);
});
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
(base) isumi@isumizumi:~/Documents/BinarAcademy/GA/KaryaSiswa$ node ExampleAsyncAwait3.js
130
```

BINAR ACADEMY

Practice

```javascript
const UserProfile = function(id) {
    return (new Promise(function(resolve, reject) {
        if (!id) {
            reject(new Error('Invalid userId'))
        }
        let response = {
            success: true,
            id: id,
            message: 'User Found'
        }
        resolve(response)
    }))
}

// menggunakan async await harus pada function yang sudah promise
// tidak bisa di root level line
async function main() {
    try {
        const result = await UserProfile(1)
        console.log(result)
    } catch (e) {
        console.log('User not found')
    }
}

main()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
(base) isumi@isumizumi:~/Documents/BinarAcademy/GA/KaryaSiswa$ node ExampleAsyncAwait.js
{ success: true, id: 1, message: 'User Found' }
```

BINAR ACADEMY

Exercise

Convert this Callback code to Promise and Async/await

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

const options = {
  url: 'https://api.github.com/repos/request/request',
  headers: {
    'User-Agent': 'request'
  }
};

function callback(error, response, body) {
  if (!error && response.statusCode == 200) {
    const info = JSON.parse(body);
    console.log(info.name)
    console.log(info.stargazers_count + " Stars");
    console.log(info.forks_count + " Forks");

  }
}

request(options, callback );
```

BINAR ACADEMY