Promise, Async, Await

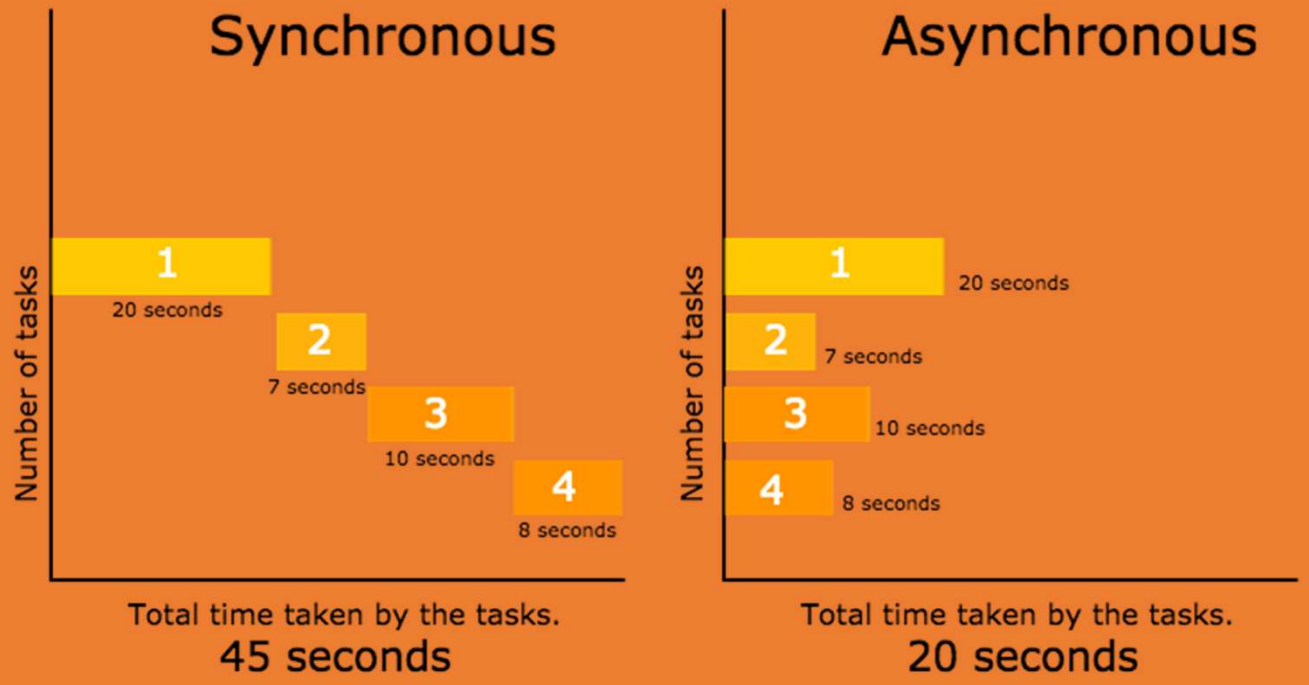
เนื้อหา

- Asynchronous คืออะไร
- Callback Hell คืออะไร
- Promise คืออะไร
- Async, Await คืออะไร

Asynchronous คืออะไร

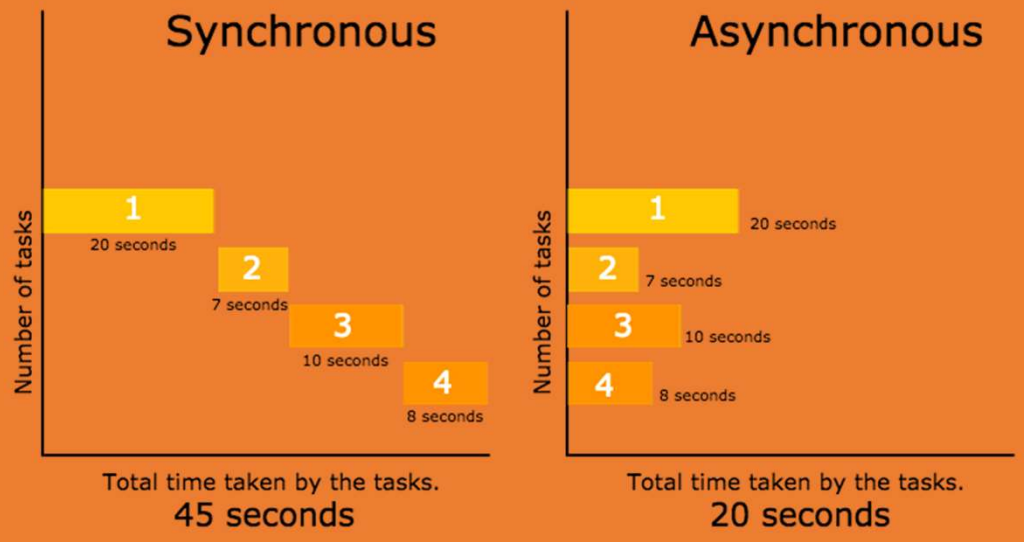
Asynchronous คืออะไร

- ฟังก์ชันที่รันคู่ขนานไปกับ Function อื่น ๆ จะถูกเรียกว่า Asynchronous



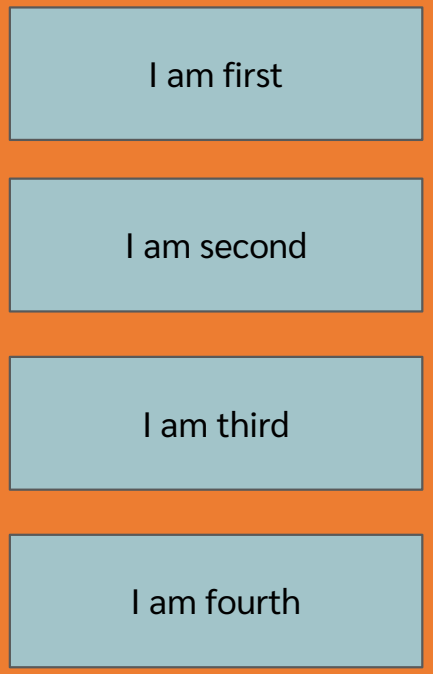
Asynchronous คืออะไร

- ประโยชน์ของ Asynchronous Function
  - ช่วยให้การทำงานของฟังก์ชันที่ต้องรอนาน ๆ ไม่ไปบล็อคฟังก์ชันอื่น



Asynchronous คืออะไร

- ตัวอย่างของ Asynchronous Function



Asynchronous คืออะไร

```
1 console.log('I am first')
2
3 setTimeout(() => {
4   console.log('I am second')
5 }, 2000)
6
7 setTimeout(() => {
8   console.log('I am third')
9 }, 1000)
10
11 console.log('I am fourth')
12
13 //Expected output:
14 // I am first
15 // I am fourth
16 // I am third
17 // I am second
18
```

Callback จะถูกเรียกตามเวลาที่ set Timeout ไว้

Asynchronous คืออะไร

Built-in โค้ดใน JS ที่ทำงานแบบ Asynchronous

- setTimeout(callback, time) - **callback** จะถูกเรียกหลังจากที่ได้ตั้งเวลา **time** (หน่วยเป็น millisecond)

```
console.log('processing ' + new Date());
setTimeout(function () {
    console.log('done ' + new Date());
}, 2000);
```

```
/opt/homebrew/opt/node@14/bin/node ./test.js
processing Mon May 16 2022 16:50:01 GMT+0700 (Indochina Time)
done Mon May 16 2022 16:50:03 GMT+0700 (Indochina Time)
```

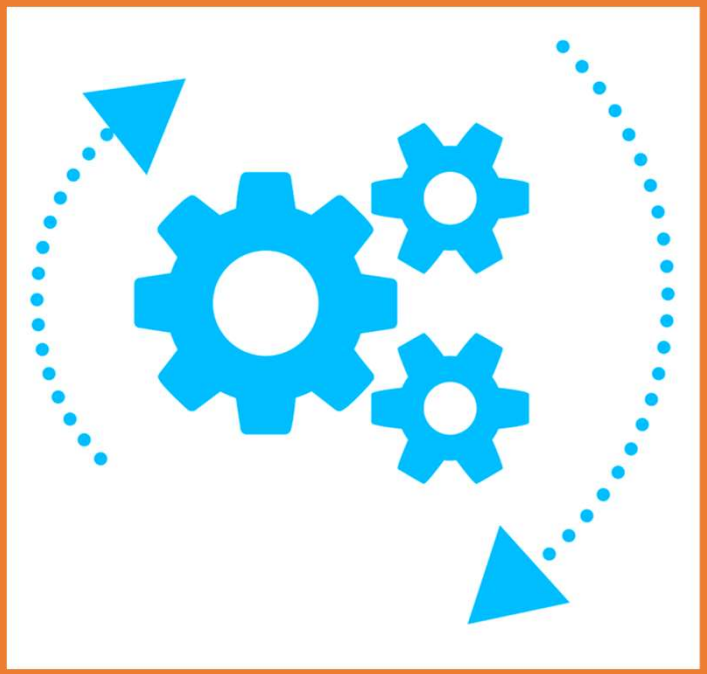
- setInterval(callback,time) - **callback** จะถูกเรียกอย่างไม่รู้จบ (loop) ทุกๆ เวลา **time** (หน่วยเป็น millisecond)

```
setInterval(() => {
 console.log(new Date());
}, 1000);
```

```
/opt/homebrew/opt/node@14/bin/node ./test.js
Mon May 16 2022 16:47:57 GMT+0700 (Indochina Time)
Mon May 16 2022 16:47:58 GMT+0700 (Indochina Time)
Mon May 16 2022 16:47:59 GMT+0700 (Indochina Time)
Mon May 16 2022 16:48:00 GMT+0700 (Indochina Time)
```

Callback Function

What is Callback Function



Why we use callback?

- ใช้สั่งให้ทำงานที่ยังไม่เริ่มทำทันที
- รอจนกว่าจะมีเหตุการณ์เกิดขึ้นถึงจะทำงาน
- เตรียมชุดคำสั่งไว้ล่วงหน้า เกิดเหตุการณ์จะได้ทำทันที

Callback Function เมื่อได้รับ Data แล้ว

Callback Function

Data

ตัวอย่าง Callback

```
customerOrderFood(function waitForGrabDriver() {
  getGrabDriver(function waitForReachRestaurant() {
    grabOrderFood(function waitForFood() {
      getFood(function waitToSendToYou() {
        ReceiveFood()
      })
    })
  })
}

```

หยุด

ตัวอย่าง Callback

```
const fs = require("fs");

fs.readFile("first.txt", { encoding: "utf-8" }, (err, data1) => {
    console.log(data1);
    fs.readFile(data1, { encoding: "utf-8" }, (err, data2) => {
        console.log(data2);
        fs.readFile(data2, { encoding: "utf-8" }, (err, data3) => {
            console.log(data3);
        });
    });
});
```

หยุด

Callback Hell คืออะไร

Callback Hell คืออะไร

- การทำงานแบบ Asynchronous แบบต่อเนื่อง แต่อ่านยาก

```
// Callback Hell


a(function (resultsFromA) {
    b(resultsFromA, function (resultsFromB) {
        c(resultsFromB, function (resultsFromC) {
            d(resultsFromC, function (resultsFromD) {
                e(resultsFromD, function (resultsFromE) {
                    f(resultsFromE, function (resultsFromF) {
                        console.log(resultsFromF);
                    })
                })
            })
        })
    })
});

```

Callback Hell คืออะไร

Promise คืออะไร

Promise คืออะไร

- การจัดลำดับรูปแบบการทำงานของ Asynchronous Pattern เพื่อให้
- เป็นการจำลองค่าในรูปของกล่องใส่ค่าชั่วคราวให้ function ที่ยังไม่รู้ค่าผลลัพธ์ เพราะยังทำงานไม่เสร็จตามกระบวนการ Asynchronous เพราะไม่รู้ว่าจะเสร็จเมื่อไร (แต่ Promise ว่าจะมีค่าในอนาคตอันใกล้) เป็นการพยายามให้ทำงานใกล้เคียงการทำงานแบบ Synchronous คืออ่านจากบนลงล่าง
- ใช้การจัดลำดับการทำงานว่าจะให้ทำงานใดก่อน งานใดหลัง ในกลุ่ม Promise ด้วยกัน
- Promise ใช้เพื่อแก้ปัญหา Callback Hell

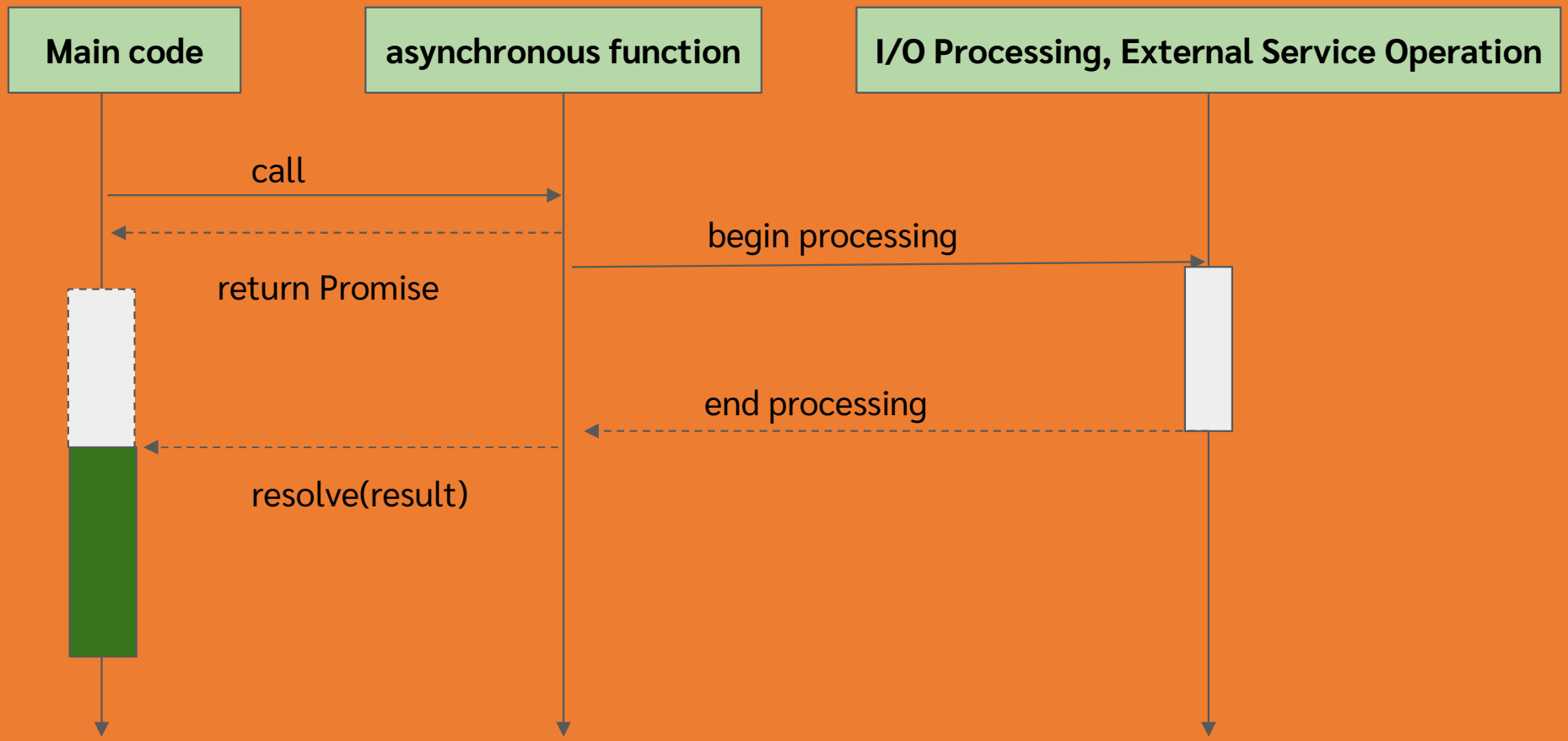
Promise คืออะไร

Function หลักของ Promise

- **Resolve** -> เรียกเมื่อการทำงานร้องขอค่า**เสร็จสมบูรณ์แล้วเท่านั้น**และส่งผลลัพธ์ของ Function ไป Resolve จะมี parameter กี่ตัวก็ได้ แต่ตัว callback function ที่นำมารับค่าใน Promise ตัวถัดไปจะต้องมีจำนวน parameter เท่ากัน
- **Reject** -> เรียกเมื่อมีบางอย่างผิดพลาดจึงจะยกเลิกการทำงานที่เหลือทั้งหมด

- **ทั้ง Resolve และ Reject ของ Promise จะถูกเรียกได้เพียงครั้งเดียวเท่านั้น! หากต้องการเรียกซ้ำจะต้องประกาศ new Promise Object ขึ้นมาใหม่**

Promise คืออะไร - Sequence



Promise คืออะไร - ตัวอย่าง

src > JS promise.js > MyPromise

```
1   import React, { useEffect } from 'react';
2
3   const MyPromise = () => {
4       let name = 'test'
5
6       const promise = new Promise((resolve, reject) => {
7           if (name === 'test') {
8               resolve("Promise resolved successfully");
9           }
10          else {
11              reject(Error("Promise rejected"));
12          }
13      });
14
15      useEffect(()=>{
16          promise.then(result => {
17              console.log('result',result);
18          }, function (error) {
19              console.log('error',error);
20          });
21      },[]);
22  }
23
24  export default MyPromise;
```

Promise คืออะไร - สถานะของ Promise

- **pending** - State เริ่มต้น
- **fulfilled** - State ที่บอกว่า Promise ทำงานสำเร็จแล้ว
- **rejected** - State ที่บอกว่า Promise ทำงานล้มเหลว (และจะเข้าส่วนของ catch ต่อไป)

Promise คืออะไร - then() , catch()

- **then(callback) :** callback จะถูกเรียก เมื่อ Promise ทำงานสำเร็จแล้ว

- **catch(callback) :** callback จะถูกเรียก เมื่อ Promise ทำงานล้มเหลว

Promise คืออะไร - catch() ตัวอย่าง

```
const resultPromise = getData()

resultPromise
  .then((data) => {
    // resolve callback จะถูกเรียกที่นี่
    console.log(data)
  })
  .catch((error) => {
    console.log(error)
  })
```

```
function getData() {
 return new Promise((resolve, reject) => {
   setTimeout(() => {
     // getting data from backend - ทำงาน 2 วินาที

     //...
     reject('get data fail');
   }, 2000);
 });
```

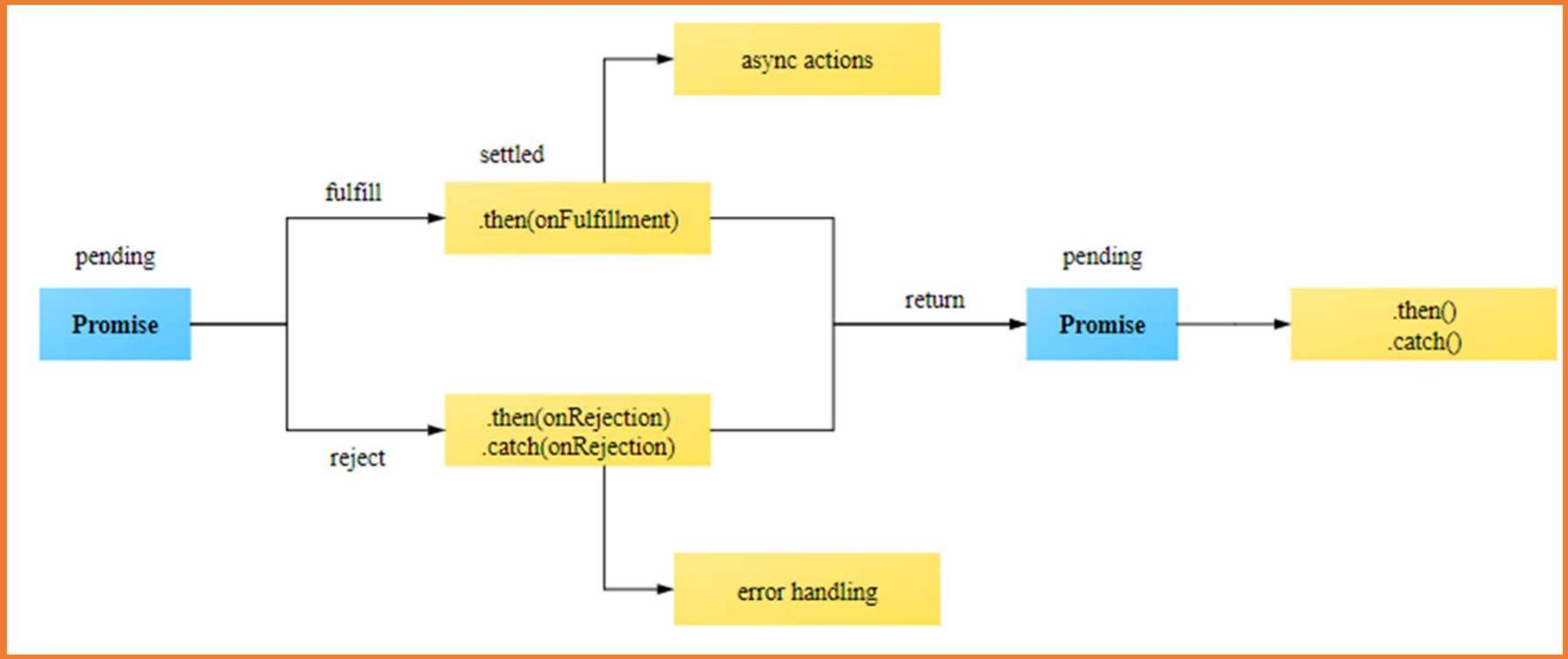
get data fail

Promise คืออะไร - catch() ตัวอย่าง

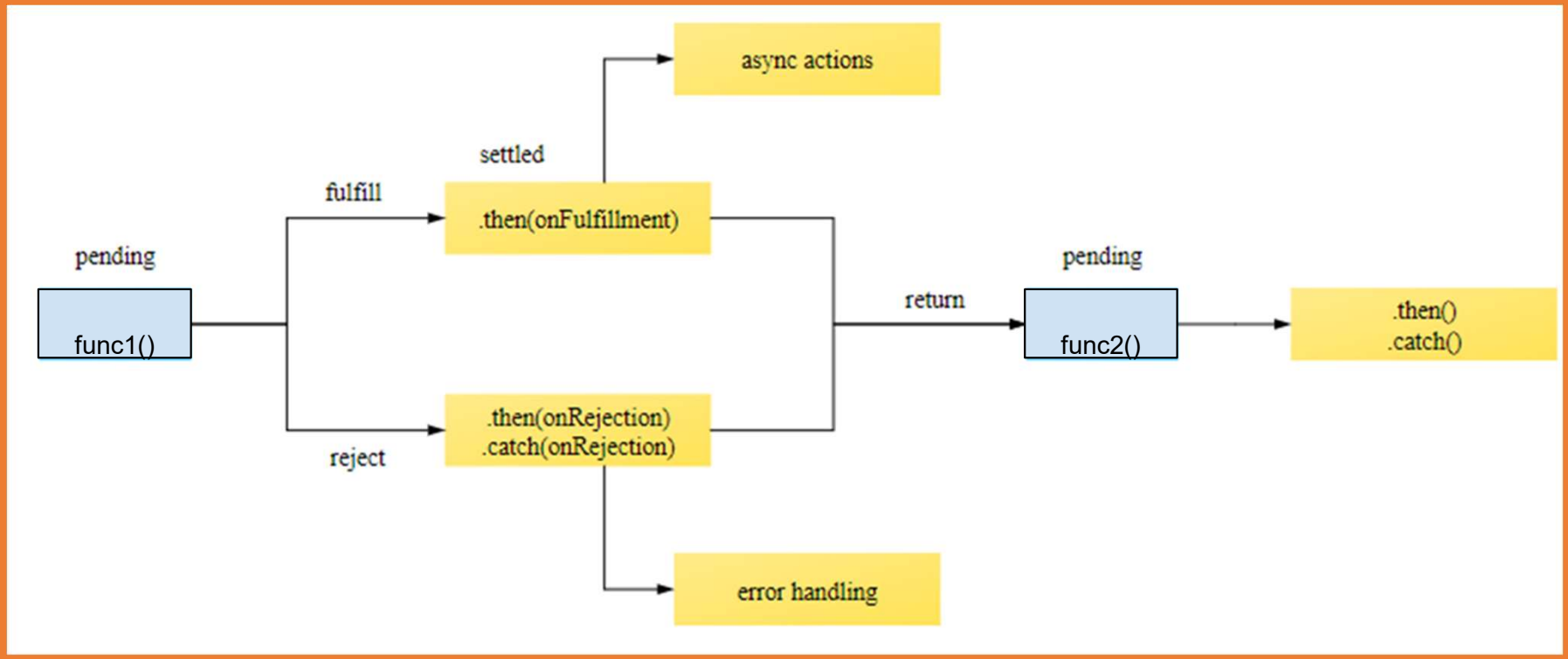
```
function getData() {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      resolve('My data')
    }, 2000)
  })
}

getData()
  .then((res) => {
    console.log('Success')
    console.log(res) // My data
  })
  .catch((err) => {
    console.log('Failed')
    console.log(err)
  })
```

Promise คืออะไร - การทำงานของ Promise



Promise คืออะไร - การทำงานของ Promise



Promise คืออะไร - Promise.all()

- **then()** จะทำงานเมื่อ Promise ทุกตัวใน array ทำงานเสร็จสิ้นแล้วเท่านั้น
- หากมี Promise ใน array บางตัวสั่ง **reject()** จะทำให้การทำงานทั้งหมดหยุดทันที
- ได้ค่าสุดท้ายเป็น array ของผลลัพธ์จากทุก Promise มารวมกันโดยเรียงลำดับให้ในรูปของ array
- ใช้สั่ง Promise เพื่อทำงานแบบ Parallel ให้ทำงานพร้อมๆ กันและรอเก็บผลลัพธ์ในองค์รวมหลัง Promise ทุกตัวทำงานเสร็จสิ้นเรียงตามลำดับตาม array ที่ส่งไปในขั้นตอนแรก

Promise คืออะไร - ตัวอย่างของ Promise.all()

```
let p1 = new Promise(function (resolve, reject) {
  resolve('one')
})

let p2 = new Promise(function (resolve, reject) {
  resolve('two')
})

let p3 = new Promise(function (resolve, reject) {
  resolve('three')
})

Promise.all([p1, p2, p3])
  .then(function (result) {
    console.log('All completed!: ', result) // result = ['one','two','three']
  })
  .catch(function (error) {
    console.error("There's an error", error)
  })

```

ส่งค่าออกมาพร้อมกัน
แต่ไม่รู้ตัวไหนออกมา
ก่อน จะรอเสร็จทั้งหมด
ก่อนจะออกมาพร้อมกัน
เป็น Array

Async และ Await คืออะไร

Async, Await คืออะไร

- คำสั่งเปลี่ยน new Promise ให้กลายเป็นค่าที่ resolve() แล้ว เรียกภายใน () เพื่อให้ assign ค่าให้ตัวแปรที่ประกาศมารับทางซ้ายได้
- สั่งให้รอจนกว่าจะเสร็จถึงจะไปบรรทัดถัดไปได้
- ต้องอยู่ภายใน function ที่มี keyword ว่า async นำหน้าเท่านั้น
- การใส่ Async หน้าฟังก์ชันนั้นจะทำให้ Function นั้น Return ออกมาเป็น Promise

Async, Await คืออะไร

```
function fetchData() {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      // fetching data - ทำงาน 2 วินาที

      //...
      resolve('data')
    }, 2000)
  })
}

const resultPromise = fetchData()
resultPromise
  .then((data) => {
    // resolve callback จะถูกเรียกที่นี่
    console.log(data)
  })
  .catch((error) => {
    console.log(error)
  })
```

```
async function main() {
  async function fetchData() {
    return new Promise((resolve, reject) => {
      setTimeout(() => {
        // fetching data - ทำงาน 2 วินาที

        //...
        resolve('data')
      }, 2000)
    })
  }

  const data = await fetchData()
  console.log(data)
}

main()
```

Callback

```
1 function getCoffeeCallback(num, func) {
2   setTimeout(() => {
3     if (typeof num === 'number') {
4       return func(`Here are your ${num} coffees!`, null)
5     } else {
6       return func(null, `${num} is not a number!`)
7     }
8   }, 3000)
9 }
10
11 getCoffeeCallback(2, (error, result) => console.log(error ? error : result))
12
13 getCoffeeCallback('butterfly', (error, result) =>
14   console.log(error ? error : result)
15 )
16
```

Promise

```
1 function getCoffeePromise(num) {
2   return new Promise((resolve, reject) => {
3     setTimeout(() => {
4       if (typeof num === 'number') {
5         resolve(`Here are your ${num} coffees!`)
6       } else {
7         reject(`${num} is not a number!`)
8       }
9     }, 3000)
10  })
11 }
12
13 getCoffeePromise(2)
14   .then((result) => console.log(result))
15   .catch((error) => console.log(error))
16
17 getCoffeePromise('butterfly')
18   .then((result) => console.log(result))
19   .catch((error) => console.log(error))
20
```

Async Await

```
1 const getCoffeeAsync = async function (num) {
2   return new Promise((resolve, reject) => {
3     setTimeout(() => {
4       if (typeof num === 'number') {
5         resolve(`Here are your ${num} coffees!`)
6       } else {
7         reject(`${num} is not a number!`)
8       }
9     }, 3000)
10  })
11 }
12
13 const start = async function (num) {
14   try {
15     const result = await getCoffeeAsync(num)
16     console.log(result)
17   } catch (error) {
18     console.error(error)
19   }
20 }
21
22 start(2)
23 start('butterfly')
24
```

Async, Await คืออะไร - ตัวอย่าง

```
async function test() {
  return 1
}

async function main() {
  console.log(test())
  console.log(await test())
}

main()

```

Async, Await คืออะไร - ตัวอย่าง

```
async function serial() {
  try {
    let result1 = await func1()
    let result2 = await func2(result1)
    let result3 = await func3(result2)
    console.log(result3)
  } catch (error) {
    console.error(error)
  }
}
serial()
```

Async, Await คืออะไร - ลองเปลี่ยนเป็น Promise

```
1 func1()
2   .then(function (result1) {
3     return func2(result1)
4   })
5   .then(function (result2) {
6     return func3(result2)
7   })
8   .then(function (result3) {
9     console.log(result3)
10  })
11  .catch(function (error) {
12    console.error(error)
13  })
14
```

Async, Await คืออะไร - แบบใช้ Arrow Function

```
1 func1()
2   .then((result1) => {
3     return func2(result1)
4   })
5   .then((result2) => {
6     return func3(result2)
7   })
8   .then((result3) => {
9     console.log(result3)
10  })
11  .catch((error) => {
12    console.error(error)
13  })
14
```

Async, Await คืออะไร - ตัวอย่าง

```
function wait1() {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      console.log('wait1')
      resolve('wait1.1')
    }, 1000)
  })
}

function wait2() {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      console.log('wait2')
      resolve('wait2.2')
    }, 1000)
  })
}

```

Async, Await คืออะไร - ตัวอย่าง

```
1 async function waitAll() {
2   try {
3     let w1 = await wait1()
4     console.log(w1)
5     let w2 = await wait2()
6     console.log(w2)
7   } catch (error) {
8     console.error(error)
9   }
10 }
11
12 waitAll() // wait 2 seconds
13
```

```
wait1
wait1.1
wait2
wait2.2
```

Async, Await คืออะไร - ตัวอย่าง

```
async function waitAll2() {
  try {
    let w1 = wait1()
    console.log(w1)
    let w2 = await wait2()
    console.log(w2)
  } catch (error) {
    console.error(error)
  }
}
waitAll2() // wait 1 seconds

```

```
wait1
wait2
wait2.2
```

Async, Await คืออะไร - ตัวอย่าง

```
async function waitAll3() {
  try {
    let w1 = await wait1()
    console.log(w1)
    setTimeout(async () => {
      //ไม่มี async จะ error
      let w2 = await wait2()
      console.log(w2)
    }, 1000)
  } catch (error) {
    console.error(error)
  }
}

waitAll3() // wait 3 seconds

```

```
wait1
wait1.1
wait2
wait2.2
```

Example

```
async function runMyCode() {
  try {
    const res = await dbUpdate(2)
    console.log('Success')
    console.log(res)
  } catch (err) {
    console.log('Failed')
    console.log(err)
  }
}

runMyCode()

```

Callback vs Promise vs Await

Callback

```
fs.readFile("first.txt", { encoding: "utf-8" }, (err, data1) => {
    console.log(data1);
    fs.readFile(data1, { encoding: "utf-8" }, (err, data2) => {
        console.log(data2);
        fs.readFile(data2, { encoding: "utf-8" }, (err, data3) => {
            console.log(data3);
        });
    });
});
```

Callback vs Promise vs Await

Promise

```
readFilePromise("first.txt")
    .then((res) => {
        console.log(res);
        return readFilePromise(res);
    })
    .then((res) => {
        console.log(res);
        return readFilePromise(res);
    })
    .then((res) => {
        console.log(res);
    });
```

Callback vs Promise vs Await

Await

```
async function runCodeAsync() {
    const res1 = await readFilePromise("first.txt");
    console.log(res1);
    const res2 = await readFilePromise(res1);
    console.log(res2);
    const res3 = await readFilePromise(res2);
    console.log(res3);
}
runCodeAsync();
```