Ch. 5

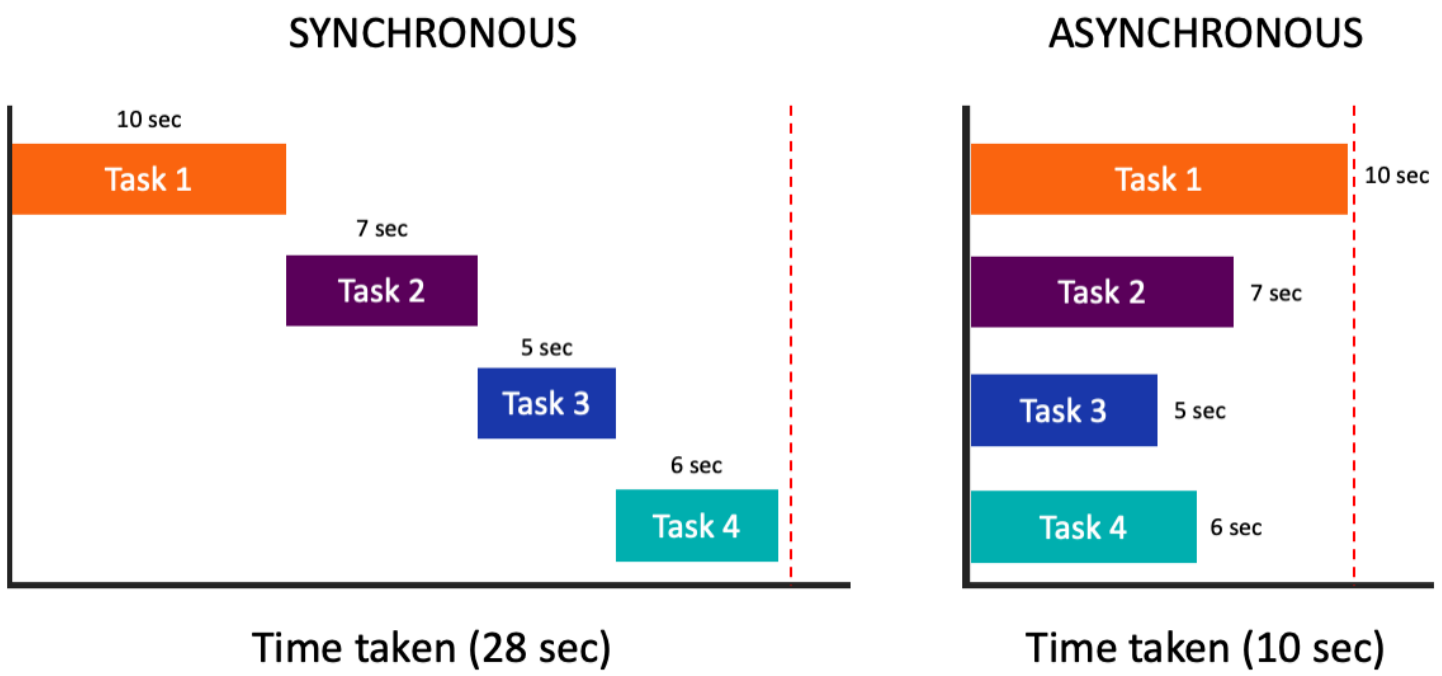
자바스크립트에서 비동기 처리하기

Table of contents

Asynchronous JavaScript

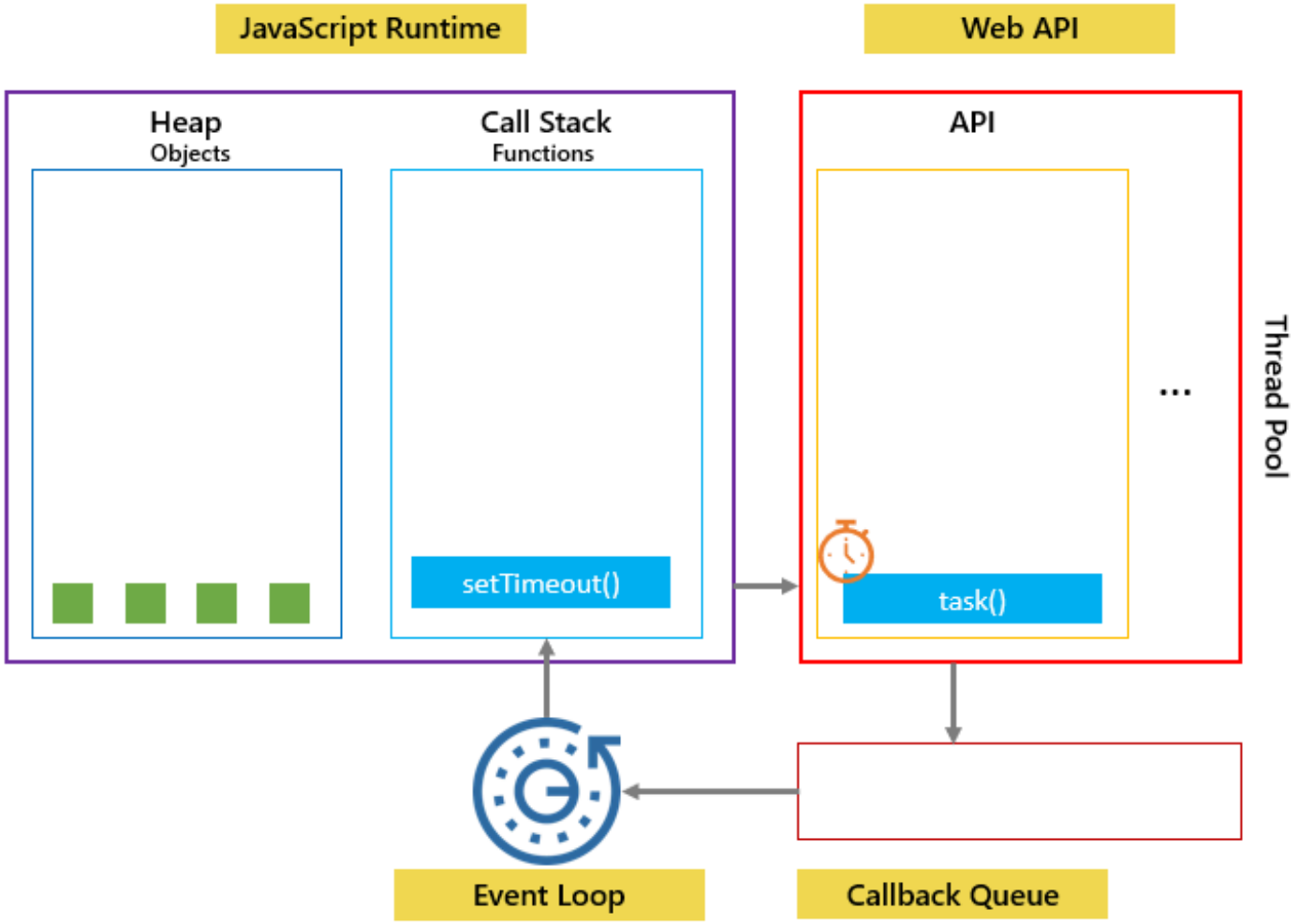
Asynchronous JavaScript

비동기 프로그래밍의 필요성



Asynchronous JavaScript

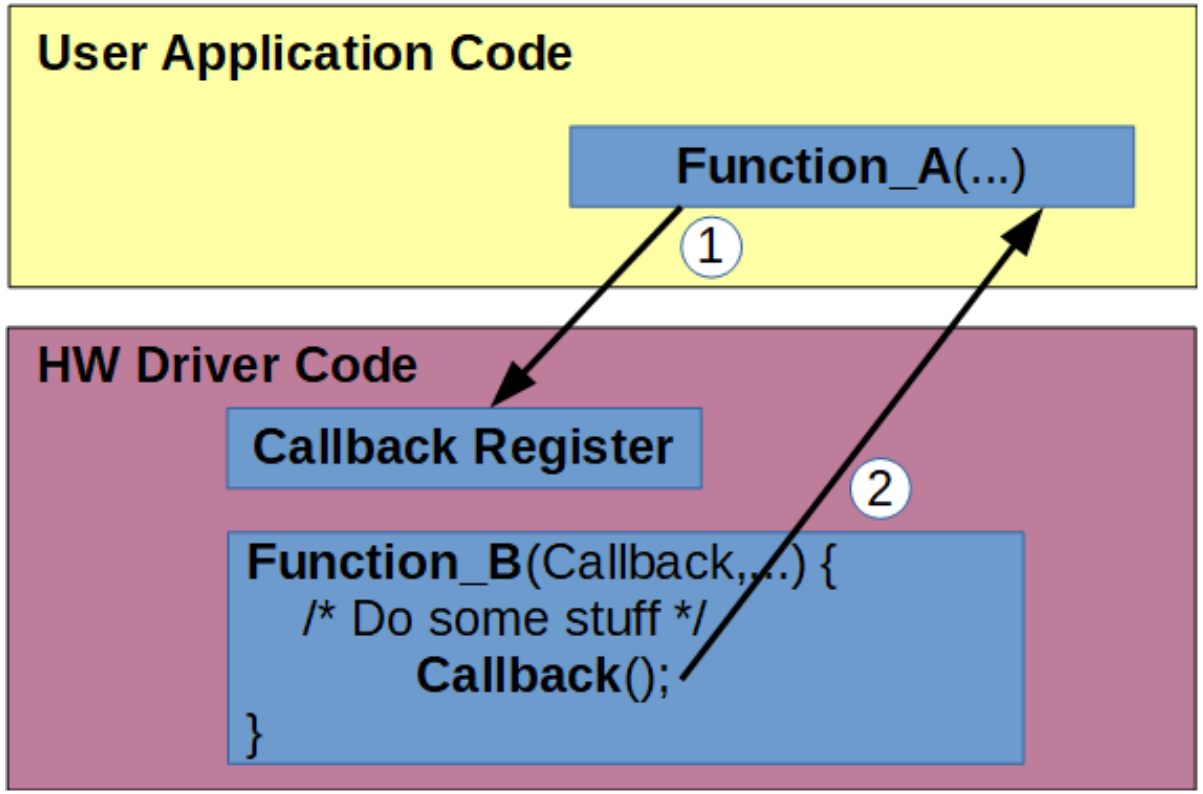
NodeJS 에서



Callback

Callback

태초에 콜백이 있었다



Callback

C 시절 - function pointer

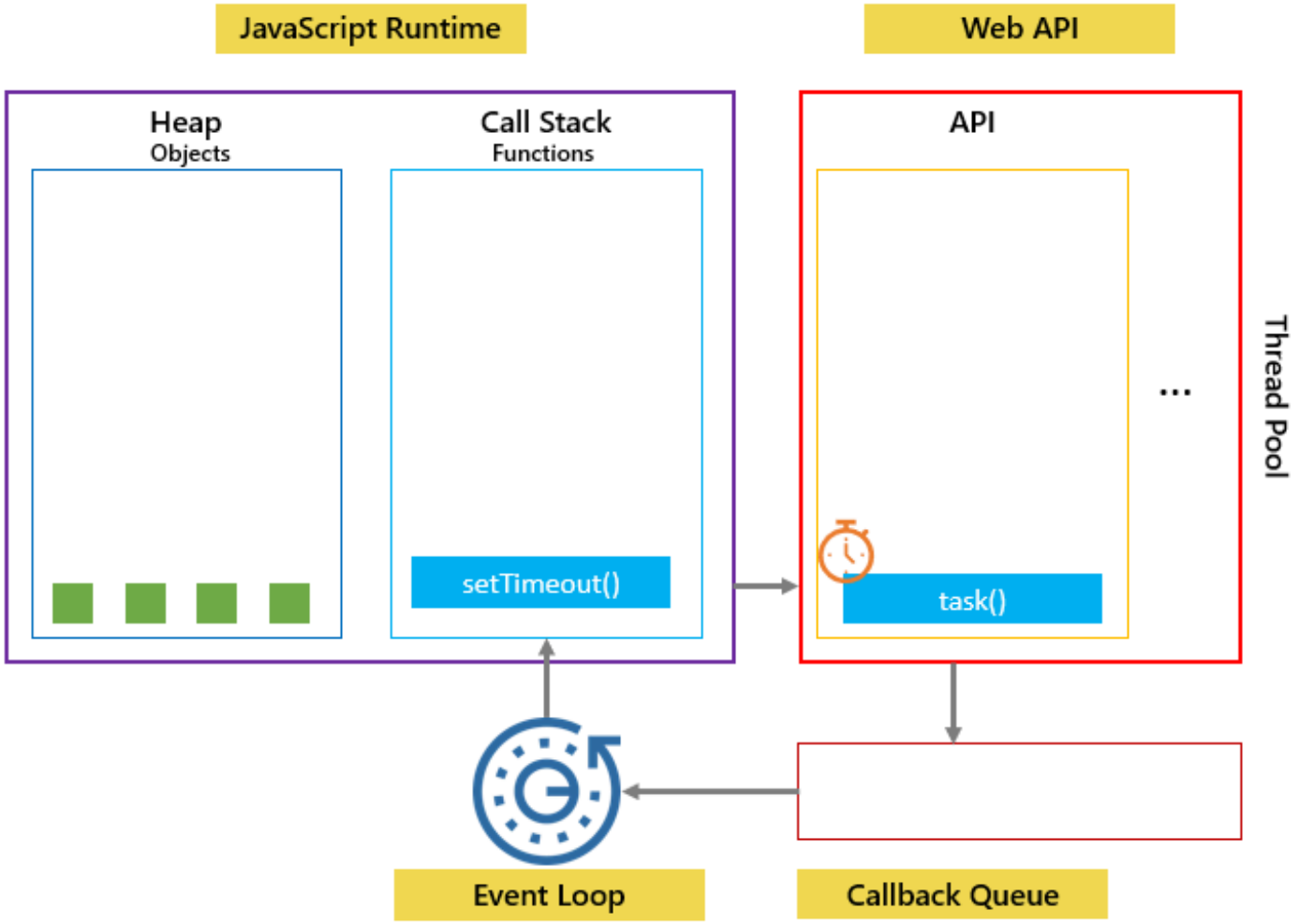
```
uint16_t sum(uint8_t a, uint8_t b) { return a + b; }
uint16_t mul(uint8_t a, uint8_t b) { return a * b; }

uint16_t arithm_op (uint16_t (*callback_func)(uint8_t, uint8_t),
                                    uint8_t a, uint8_t b)
{ return callback_func(a,b); }

void main() {
    arithm_op(mul,4,10); /* 4x10*/
    arithm_op(add,9,5); /* 9+5 */
}
```

Callback

NodeJS 에서



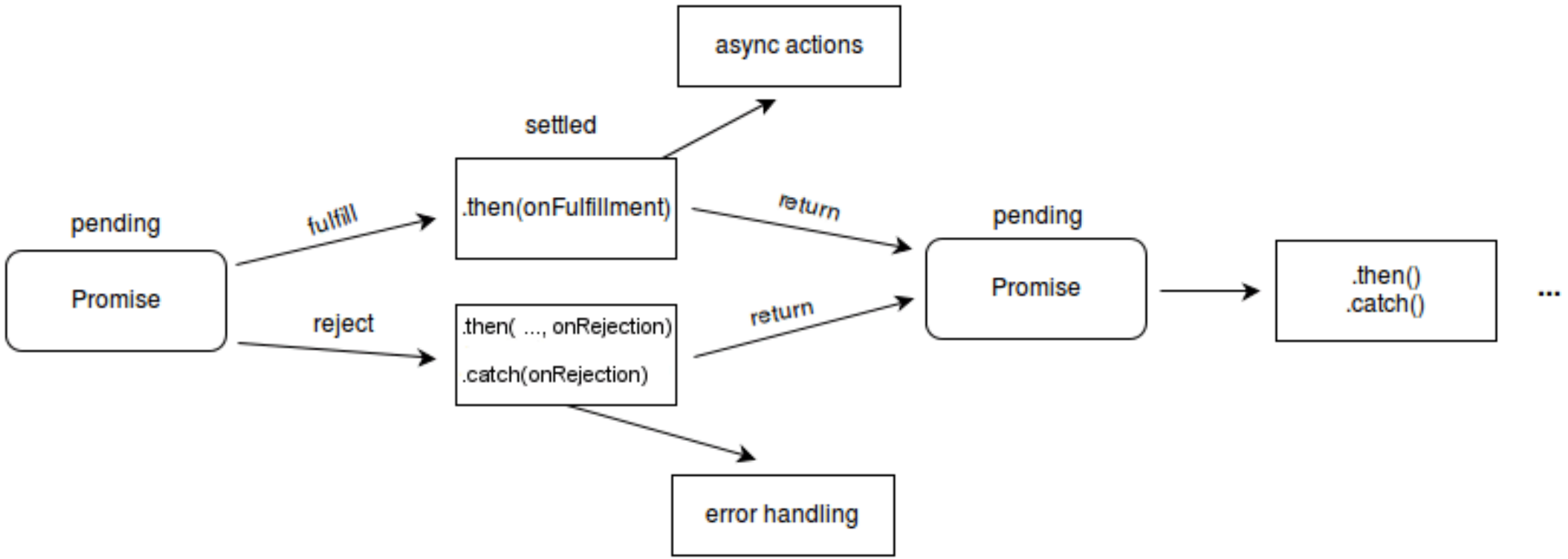
Callback

Callback Hell / Callback Pyramid

```
btn1.addEventListener("click", function () {
        fadeOut(this, 500, function () {
                fadeIn(btn2, 500, function () {
                        btn2.addEventListener("click", function () {
                                fadeOut(this, 500, function () {
                                        fadeIn(btn3, 500, function () {
                                                btn3.addEventListener("click", function () {
                                                        fadeOut(this, 500, function () {
                                                                fadeIn(btn4, 500, function () {
                                                                        btn4.addEventListener("click", function () {
                                                                                fadeOut(this, 500, function () {
                                                                                        fadeIn(btn5, 500, function () {
                                                                                                btn5.addEventListener("click", function () {
                                                                                                        fadeOut(this, 500, function () {
                                                                                                                fadeIn(btn6, 500, function () {
                                                                                                                        btn6.addEventListener("click", function () {
                                                                                                                                fadeOut(this, 500, function () {
                                                                                                                                        fadeIn(document.body, 500, function () {
                                                                                                                                                alert("all callbacks completed");
                                                                                                                                        });
                                                                                                                                });
                                                                                                                        });
                                                                                                                });
                                                                                                        });
                                                                                                });
                                                                                        });
                                                                                });
                                                                        });
                                                                });
                                                        });
                                                });
                                        });
                                });
                        });
                });
        });
});
```

Promise

Promise


async actions
settled
pending
fulfill
.then(onFulfillment)
return
pending
Promise
.then()
.catch()
...
Promise
reject
.then(..., onRejection)
.catch(onRejection)
return
error handling

Promise

사용법

```
const myPromise = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve("foo");
  }, 300);
});

myPromise
  .then((value) => `${value} and bar`)
  .then((value) => `${value} and bar again`)
  .then((value) => `${value} and again`)
  .then((value) => `${value} and again`)
  .then((value) => {
    console.log(value);
  })
  .catch((err) => {
    console.error(err);
  });
```

Promise

조금 나아졌지만 여전히 쓰기 어렵다

1. chaning 도중에 특정 로직/에러핸들링이 필요하면?
2. 여전한 가독성 문제
3. 중첩 promise

Async/Await

Async/Await

- Syntactic sugar - 아래 두 코드는 "거의 동일"하다 (참고: MDN)
- `async function` 은 다른 레퍼런스, `Promise.resolve` 은 같은 레퍼런스 반환
  - 동일한지 체크 시 주의 필요

```
async function foo() {
  return 1;
}
```

```
function foo() {
  return Promise.resolve(1);
}
```

Async/Await

Promise => async/await 으로 바꿔보기

값 받아오기 예제

```
$.get(url1)
.then((res1) => {
    console.log(res1)
    return $.get(doSomethingWithRes(res1, url2))
})
.then((res2) => {
    console.log(res2)
    return $.get(doSomethingWithRes(res2, url3))
})
.then((res3) => {
    console.log(res3)
})
```

Async/Await

Promise => async/await 으로 바꿔보기

값 받아오기 예제

```
async function myFunction (){
    let res1 = await $.get(url)
    console.log(res1)
    let res2 = await $.get(doSomethingWithRes(url, res1))
    console.log(res2)
    let res3 = await $.get(doSomethingWithRes(url, res2))
    console.log(res3)
}
```

Async/Await

Promise => async/await 으로 바꿔보기

에러 처리

```
try {
    $.get(url1)
    .then(() => $.get(url2))
    .then(function (res){ throw new Error("Hello error! ⚠️") })
    // .catch((err) => console.error('Handling error: ', err))
} catch(err){ // 소용없는 catch
    console.error(err)
}
```

Async/Await

Promise => async/await 으로 바꿔보기

에러 처리

```
async function myFunction () {
    try {
        await $.get(url1)
        await $.get(url2)
        throw new Error("Hello error! ⚠️")
    } catch(err){
        console.error(err)
    }
}
```

Practical Use Cases

Practical Use Cases

그래서 언제 쓸까?

- Frontend (반응형 UI)
- I/O Bound task (DB, FS, APIs, ...)
- 요청이 많은데 (DB등이) scale 가능할때

쓸 때 생각할 부분

- MSA 등의 "small cpu" 환경일 경우 (?)
- DB가 scale 안될때
- (속도가 느리더라도) 요청이 많지 않을때
- 이미 빠르게 작동하는 요청일 경우 - context switching 비용
- CPU-heavy 한 연산이 많고 굉장히 최적화되어 순차적 실행을 가정하는 monolithic 구조 시스템

References

1. https://scoutapm.com/blog/async-javascript
2. https://open4tech.com/using-callbacks-in-firmware-development
3. https://www.youtube.com/watch?v=8aGhZQkoFbQ&vl
4. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise
5. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/async_function
6. https://scoutapm.com/blog/async-javascript

References

7. https://www.linkedin.com/pulse/callback-hell-javascriptjquery-its-solution-ram-kumar-shrestha/
8. https://thecodebarbarian.com/using-promise-finally-in-node-js.html
9. https://dev.to/chinmaymhatre/crash-course-in-asynchronous-javascript-part-2-29jn
10. https://codeculturepro.medium.com/what-distinguishes-js-promises-from-async-await-syntax-in-javascript-7976be18c11a
11. https://medium.com/@rajatsikder/asynchronous-programming-use-cases-86727de31992