한입 크기로 잘라먹는

Truthy & Falsy 🔁

JavaScript 에서는 참, 거짓이 아닌 값도 참, 거짓으로 평가한다

```
if (123) {
  console.log("123 is true");
} else {
  console.log("123 is false");
}

if (undefined) {
  console.log("undefined is true");
} else {
  console.log("undefined is false");
}
```

```
123 is true                    chapter01.js:2
undefined is false             chapter01.js:10
```

실행 결과

Truthy & Falsy란?

- 참이나 거짓을 의미하지 않는 값도, 조건문 내에서 참이나 거짓으로 평가하는 특징

```
if (123) {
  console.log("123 is true");
} else {
  console.log("123 is false");
}

if (undefined) {
  console.log("undefined is true");
} else {
  console.log("undefined is false");
}
```

Truthy 한 값
(참 같은 값)

Falsy 한 값
(거짓 같은 값)

JavaScript의 모든 값은 Truthy 하거나 Falsy 합니다

- 이를 이용하면 조건문을 간결하게 만들 수 있음

한입 크기로 잘라먹는

단락 평가 🧪

@한 입 크기로 잘라먹는 리액트

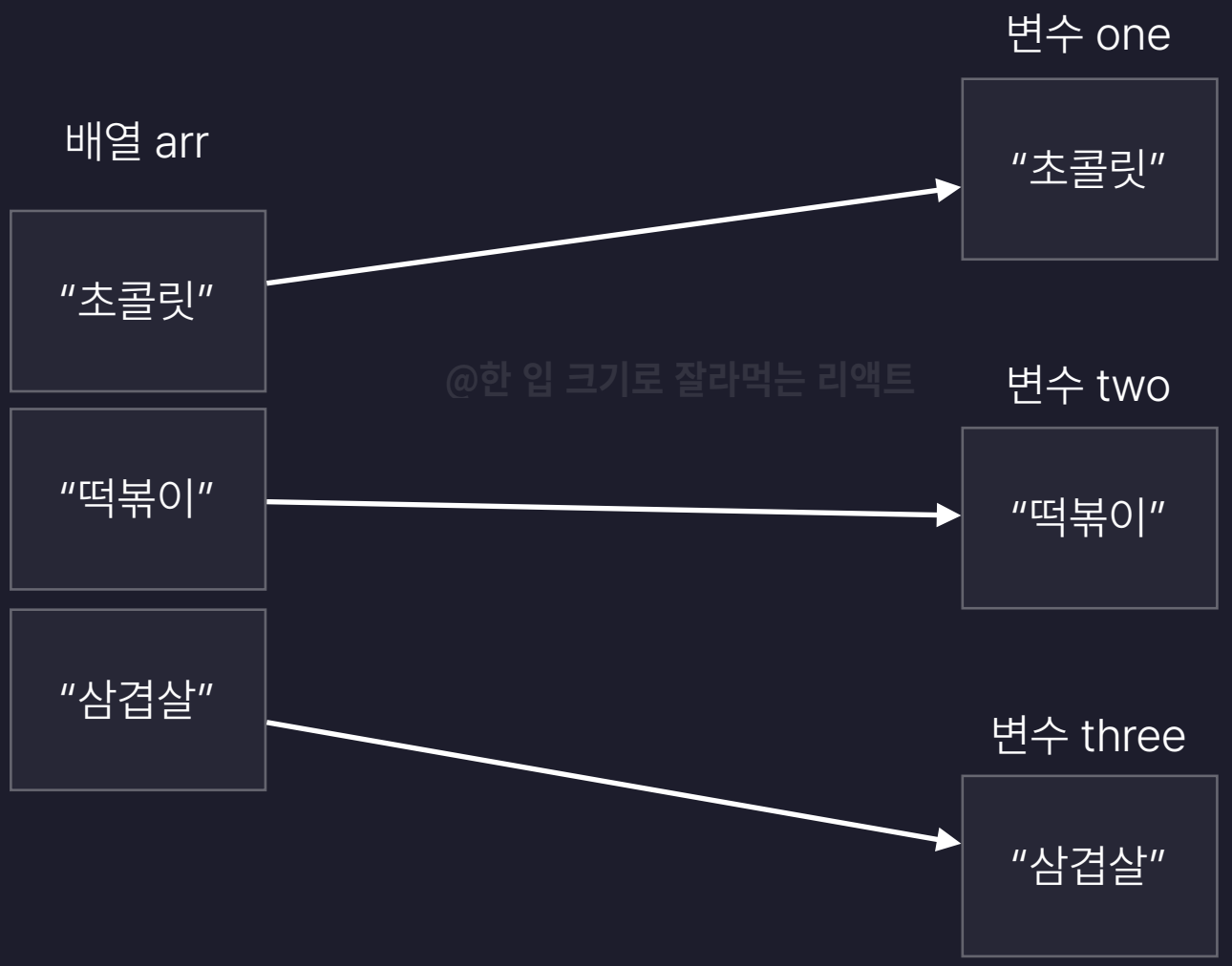
단락 평가(Short-circuit Evaluation)이란?

```
let varA = false;

let varB = true;

// AND 연산자
console.log(varA && varB);

// OR 연산자
console.log(varB || varA);
```

한입 크기로 잘라먹는

구조 분해 할당 🔩

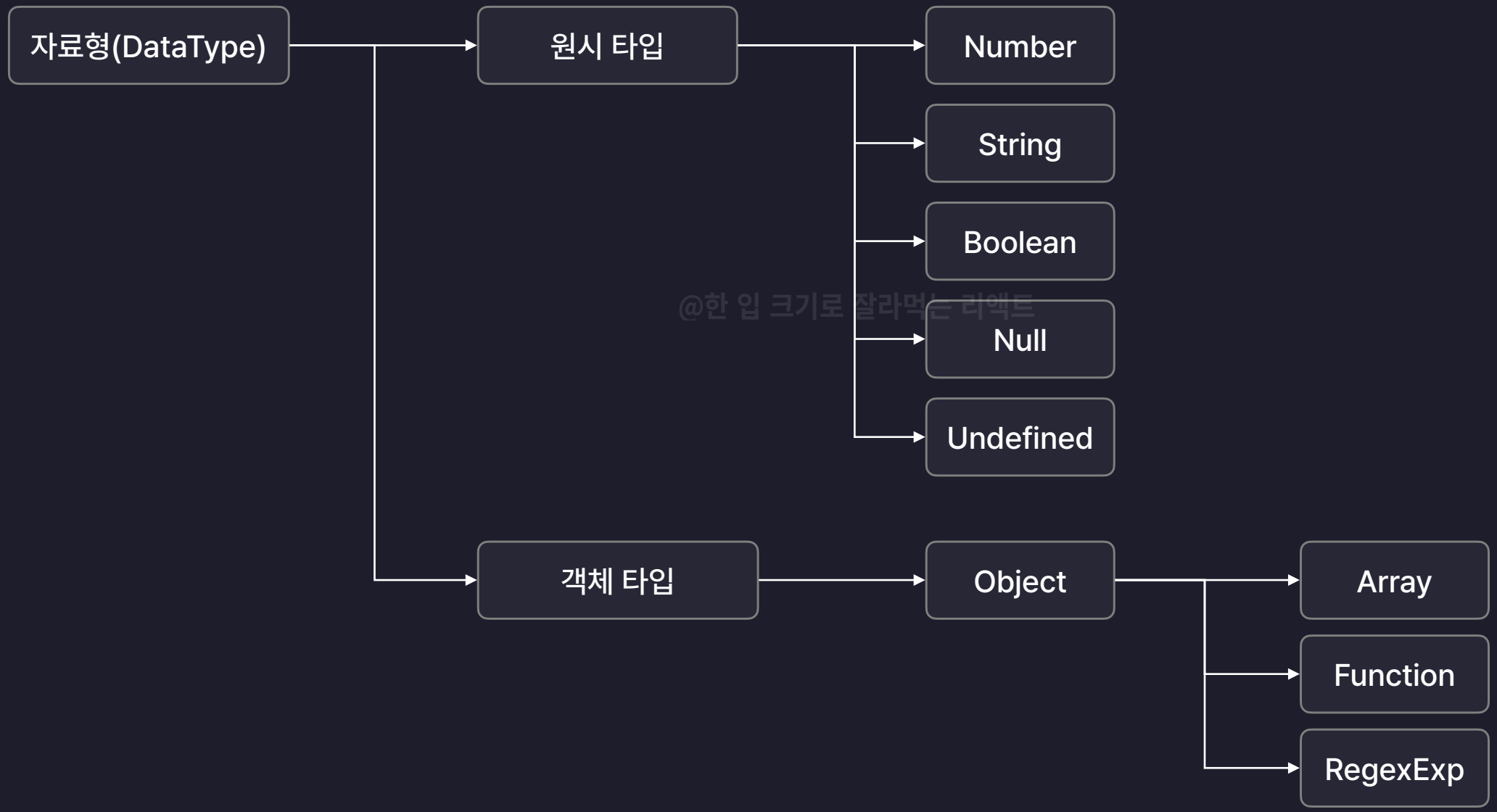
구조 분해 할당이란?


변수 one
배열 arr
"초콜릿"
"초콜릿"
@한 입 크기로 잘라먹는 리액트
변수 two
"떡볶이"
"떡볶이"
"삼겹살"
변수 three
"삼겹살"

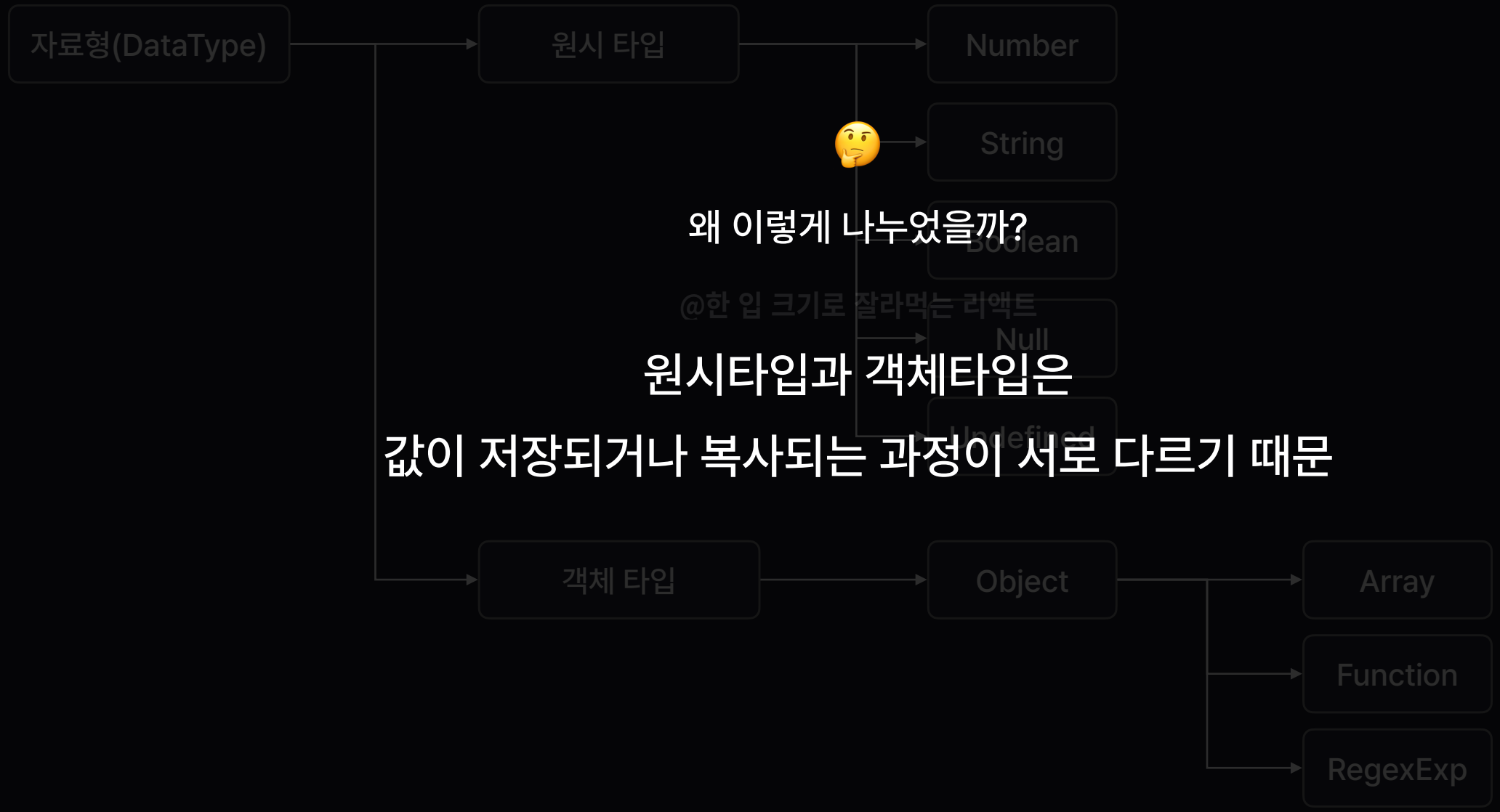
한입 크기로 잘라먹는

원시타입 VS 객체타입 🤺

자바스크립트의 자료형(타입)



자바스크립트의 자료형(타입)



왜 이렇게 나누었을까?

원시타입과 객체타입은

값이 저장되거나 복사되는 과정이 서로 다르기 때문

원시타입 VS 객체타입

원시 타입

Number, String, Boolean 등 ...

값 자체로써 변수에 저장되고 복사 된다

VS

객체 타입

Object, Array, Function 등 ...

참조값을 통해 변수에 저장되고 복사된다

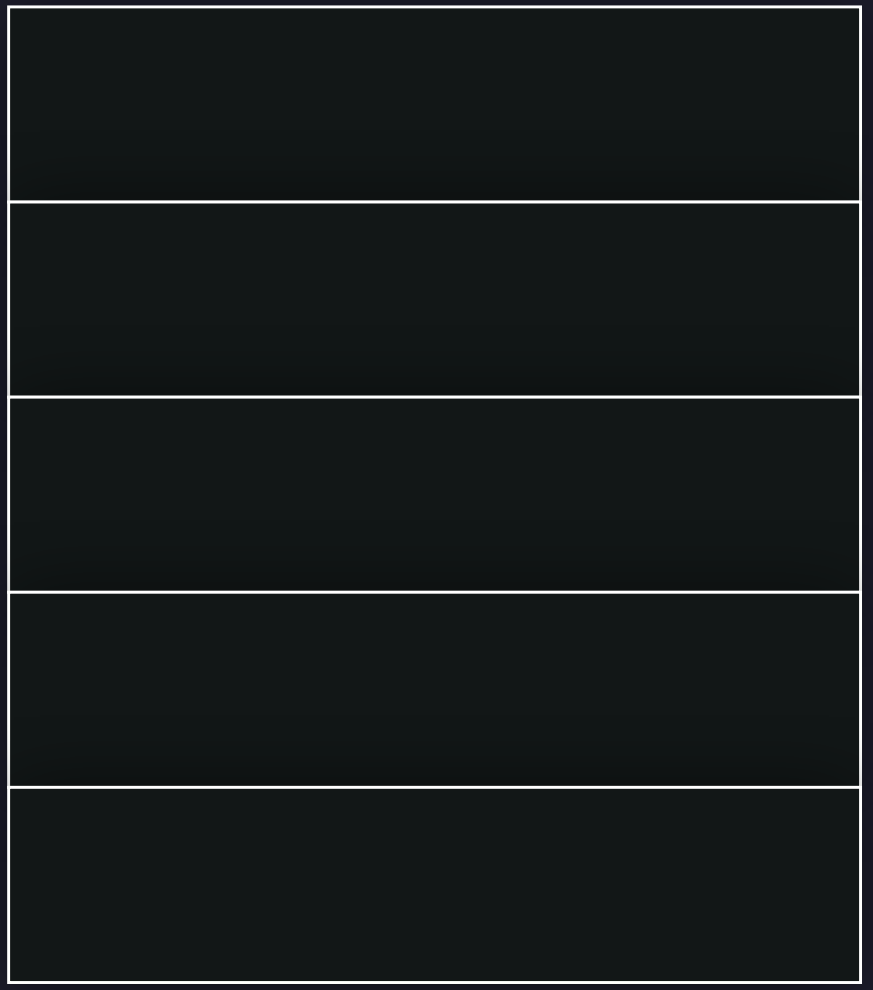
원시 타입

[Code]

```
let p1 = 1;
let p2 = p1;
```

[Name]

[Memory]



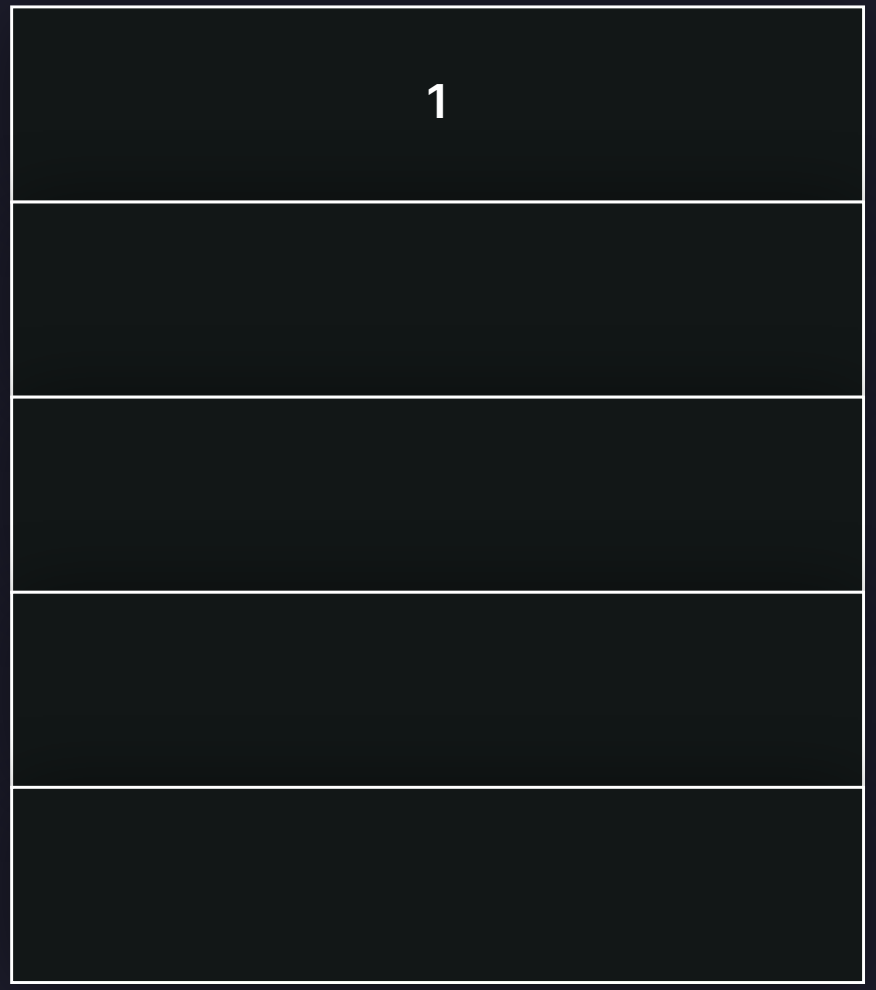
원시 타입

[Code]

```
let p1 = 1;
let p2 = p1;
```

[Name]

[Memory]

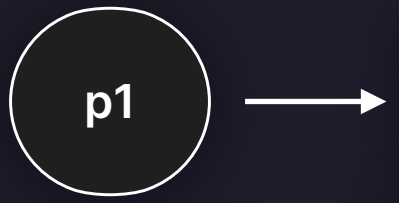


원시 타입

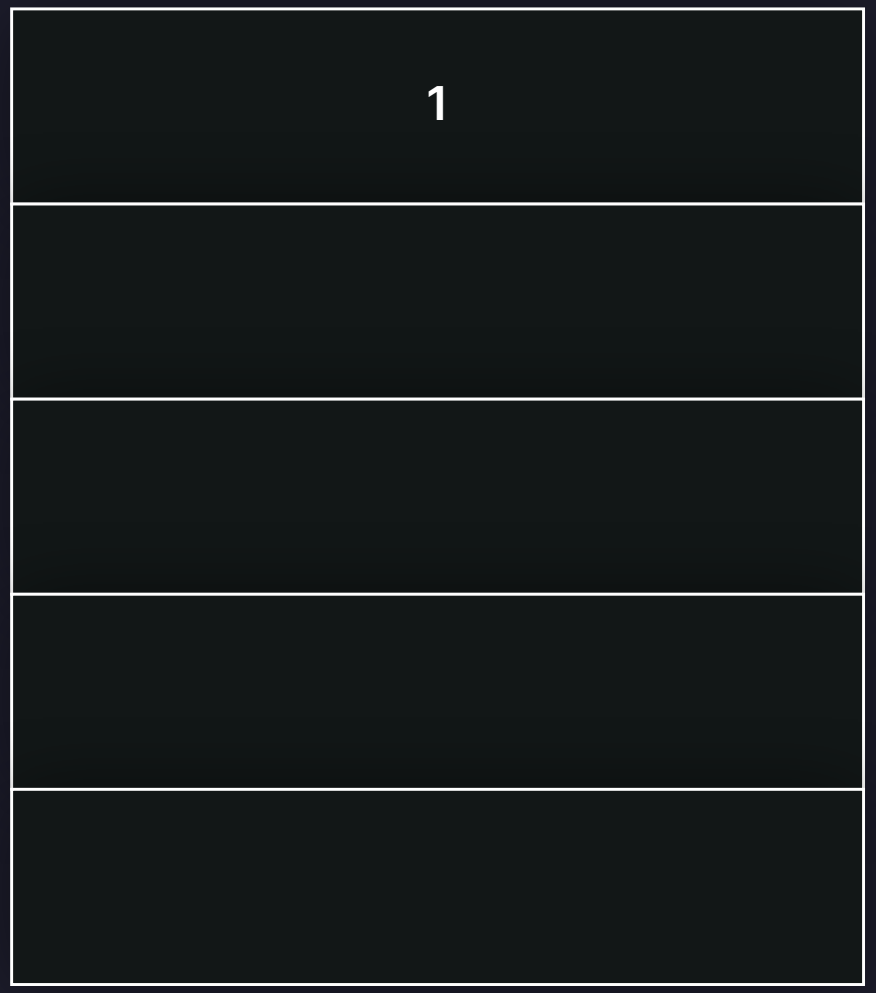
[Code]

```
let p1 = 1;
let p2 = p1;
```

[Name]



[Memory]

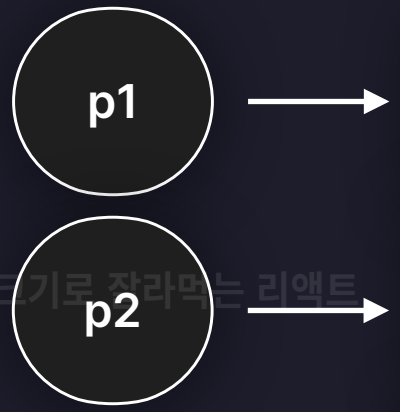


원시 타입

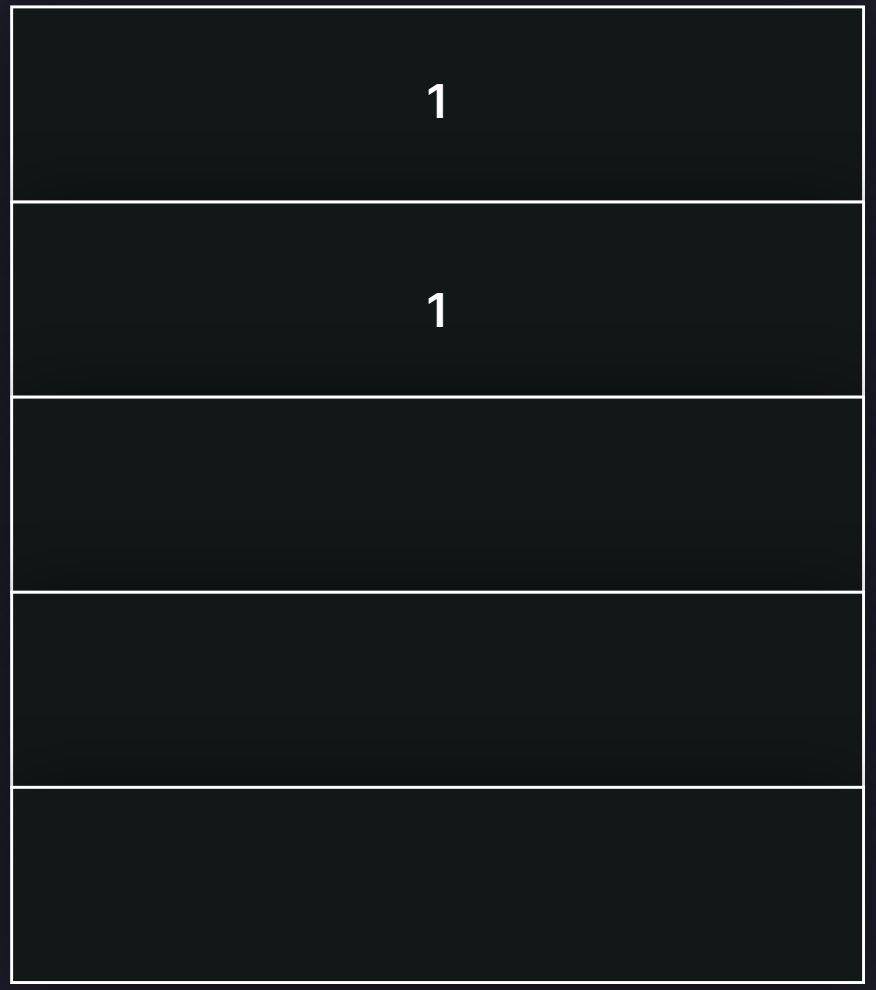
[Code]

```
let p1 = 1;
let p2 = p1;
```

[Name]



[Memory]

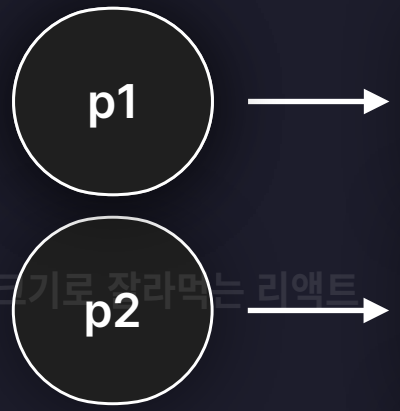


원시 타입

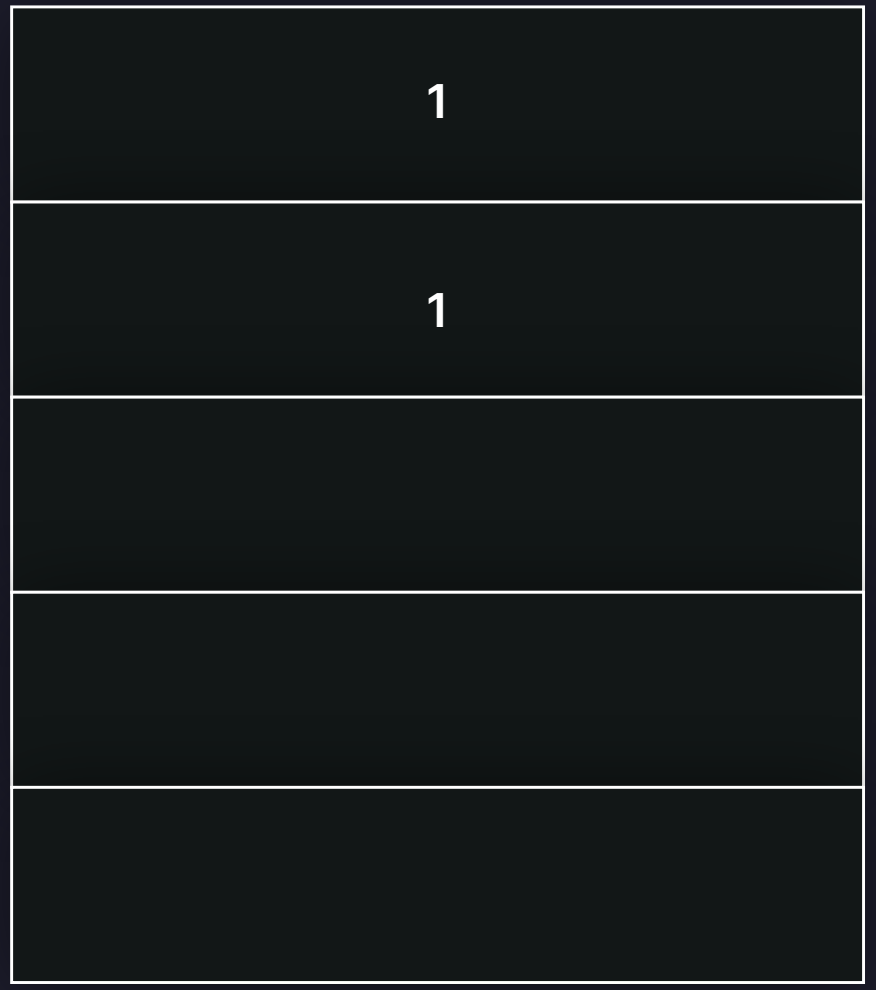
[Code]

```
let p1 = 1;
let p2 = p1;

p2 = 2;
```

[Name]



[Memory]

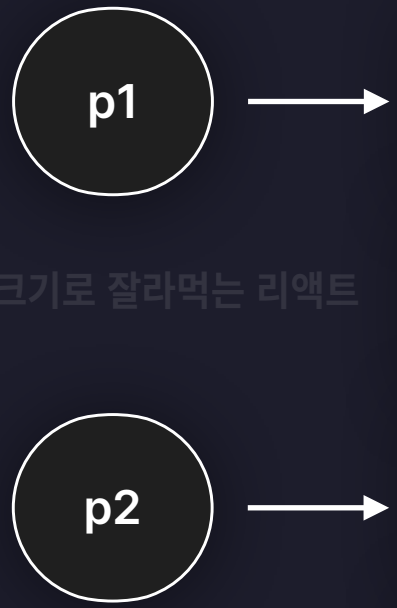


원시 타입

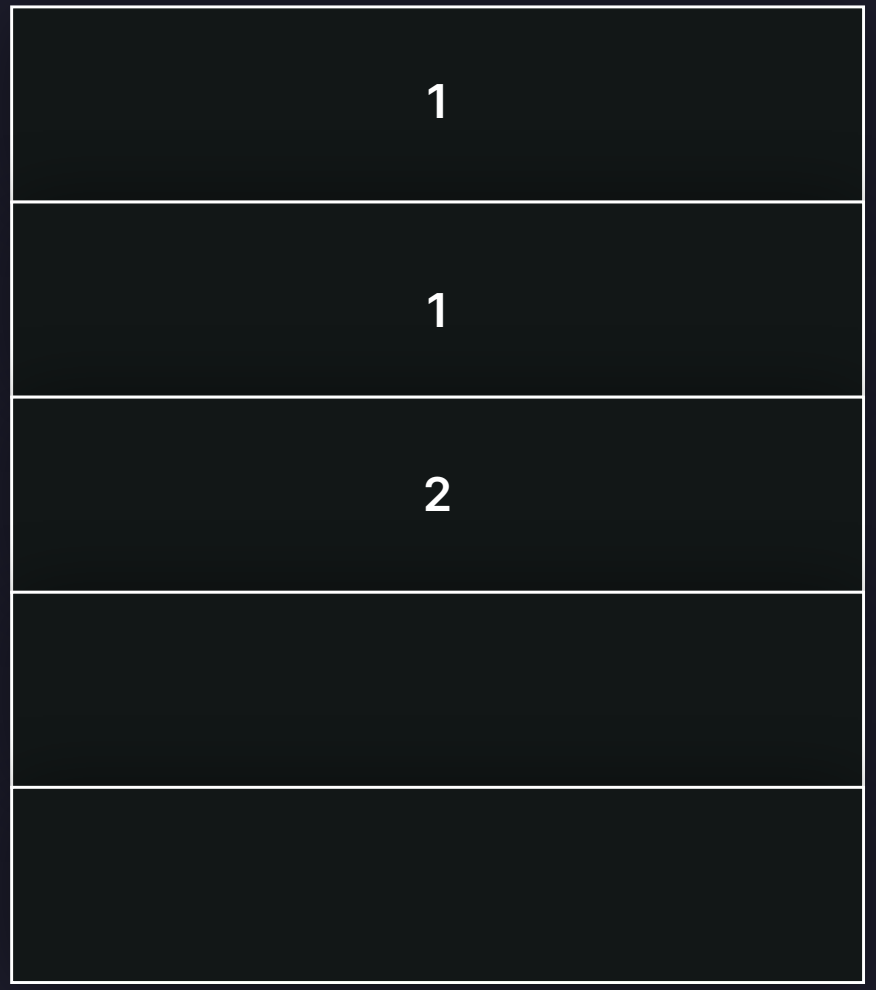
[Code]

```
let p1 = 1;
let p2 = p1;

p2 = 2;
```

[Name]



[Memory]



원시 타입

[Code]

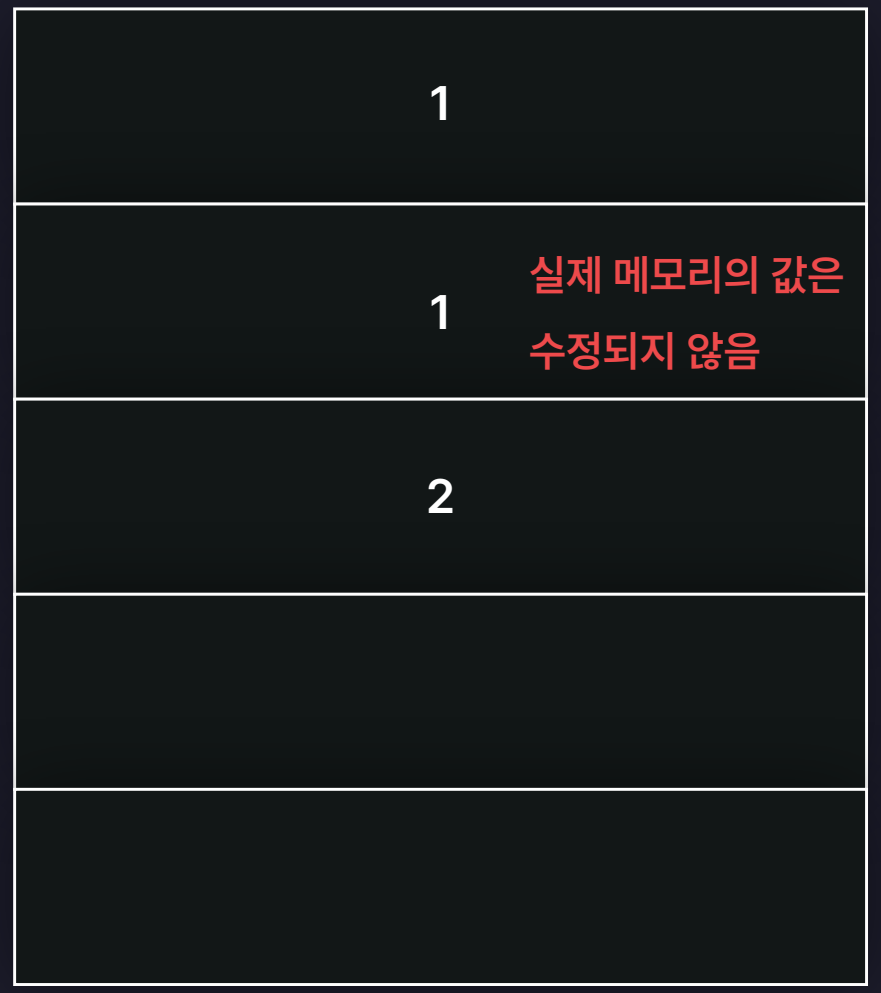
```
let p1 = 1;
let p2 = p1;

p2 = 2;
```

[Name]

p1

p2

[Memory]

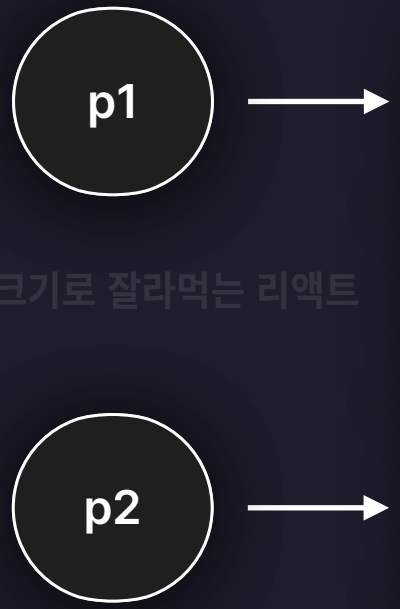


원시 타입 = 불변값

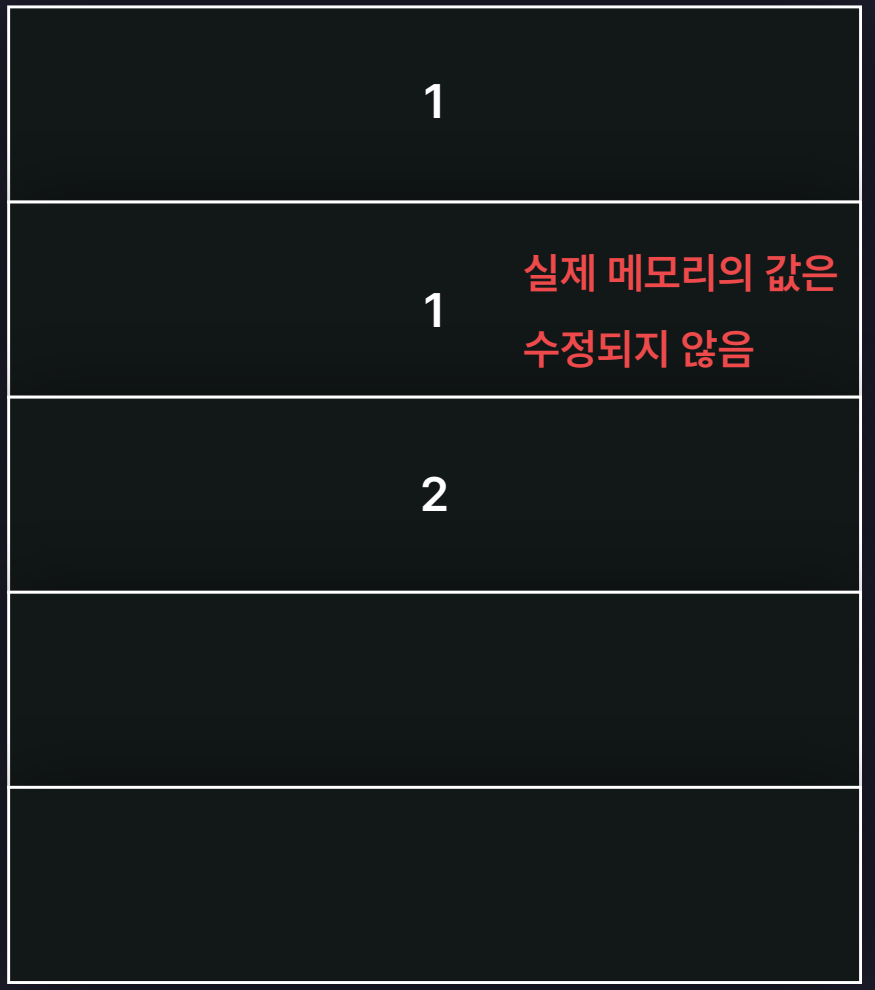
[Code]

```
let p1 = 1;
let p2 = p1;

p2 = 2;
```

[Name]



[Memory]



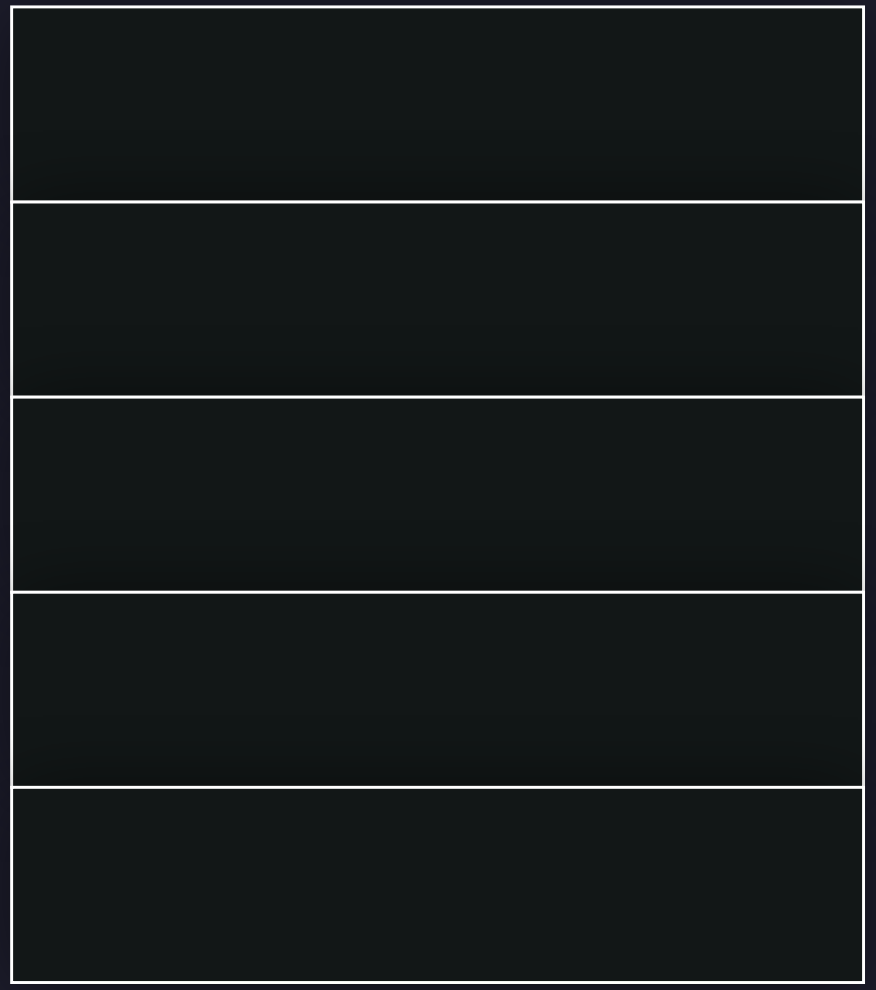
객체타입

[Code]

```
let o1 = { name: "이정환" };
```

[Name]

[Memory]



객체타입

[Code]

```
let o1 = { name: "이정환" };
```

[Name]

o1

[Memory]

{ name : “이정환” }

@한 입 크기로 잘라먹는 리액트

객체타입

[Code]

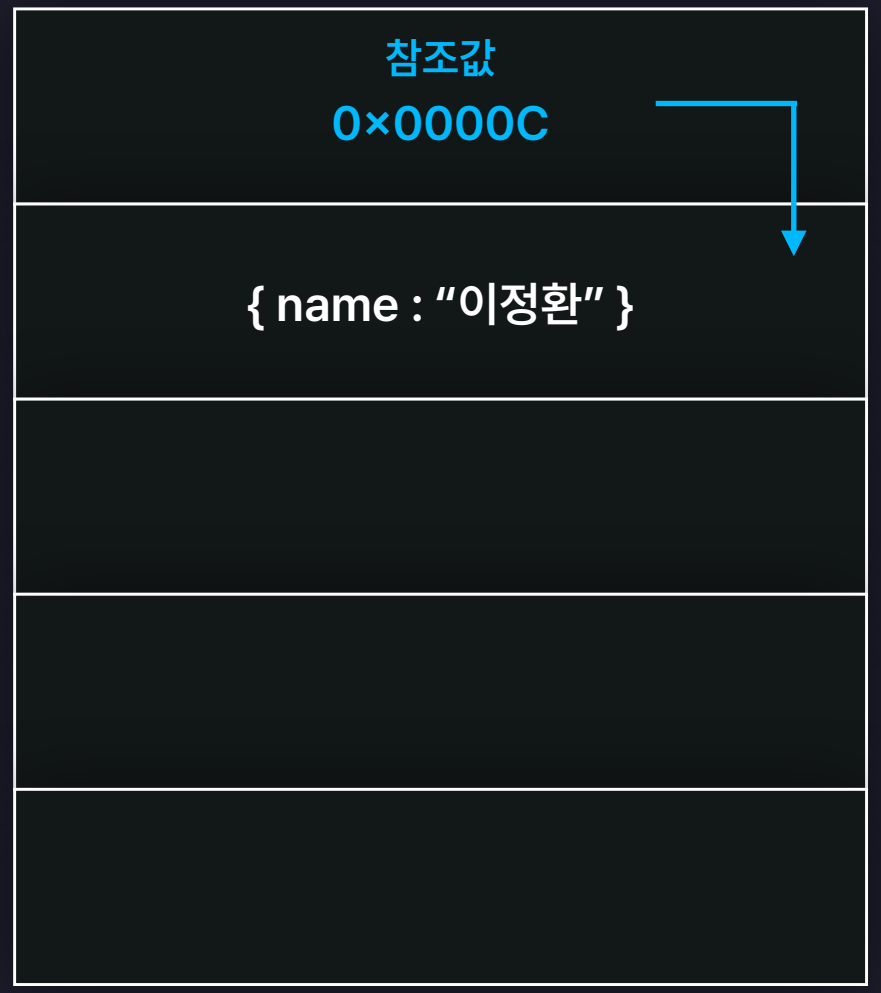
```
let o1 = { name: "이정환" };
```

[Name]

o1

@한 입 크기로 잘라먹는 리액트

[Memory]

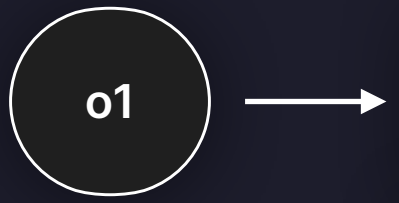


객체타입

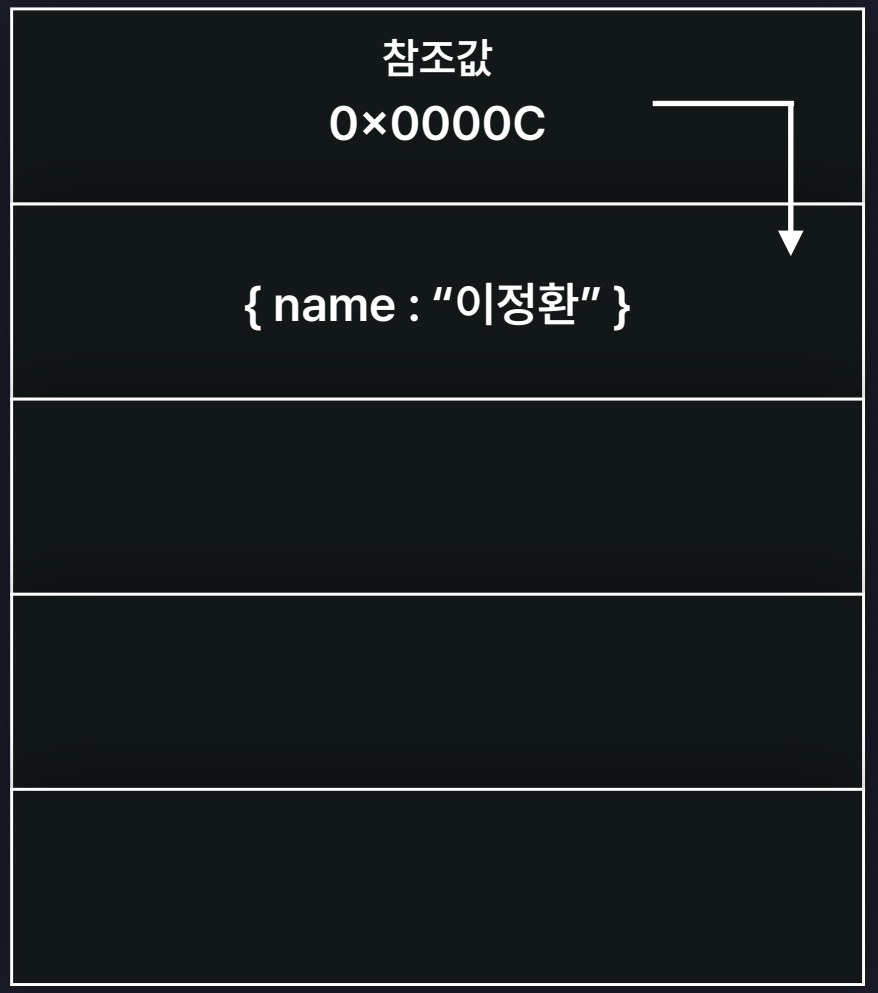
[Code]

```
let o1 = { name: "이정환" };
let o2 = o1;
```

[Name]



[Memory]

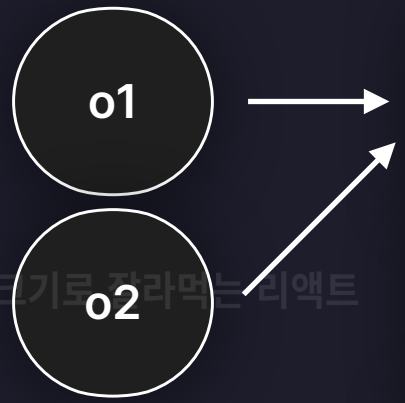


객체타입

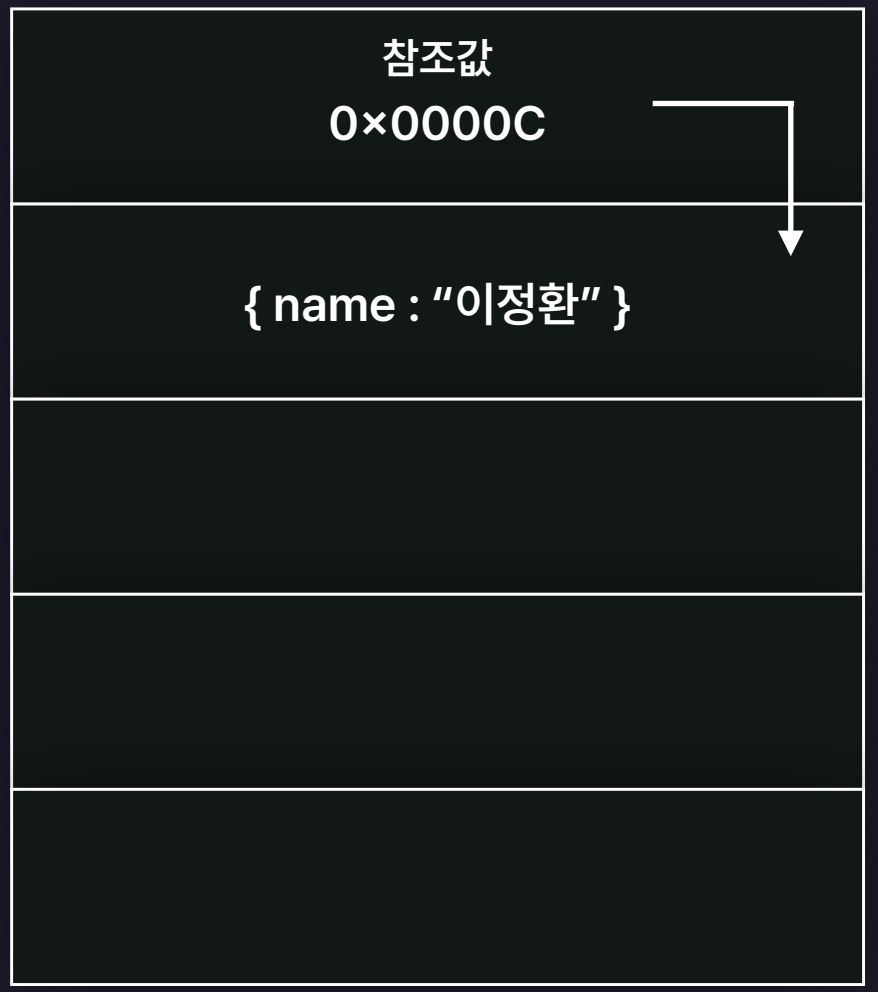
[Code]

```
let o1 = { name: "이정환" };
let o2 = o1;
```

[Name]



[Memory]

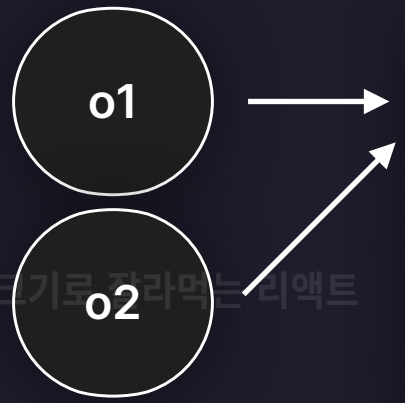


객체타입

[Code]

```
let o1 = { name: "이정환" };
let o2 = o1;

o2.name = "홍길동";
```

[Name]



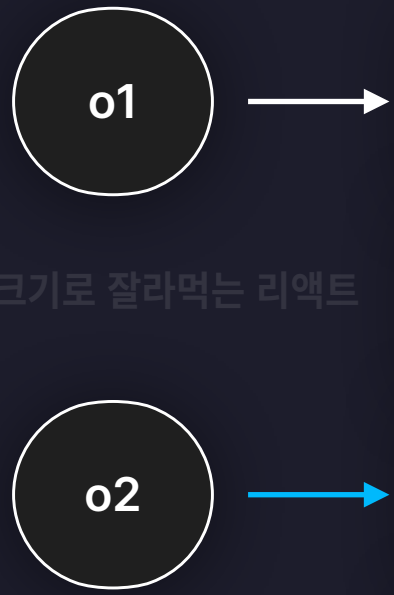
[Memory]

객체타입

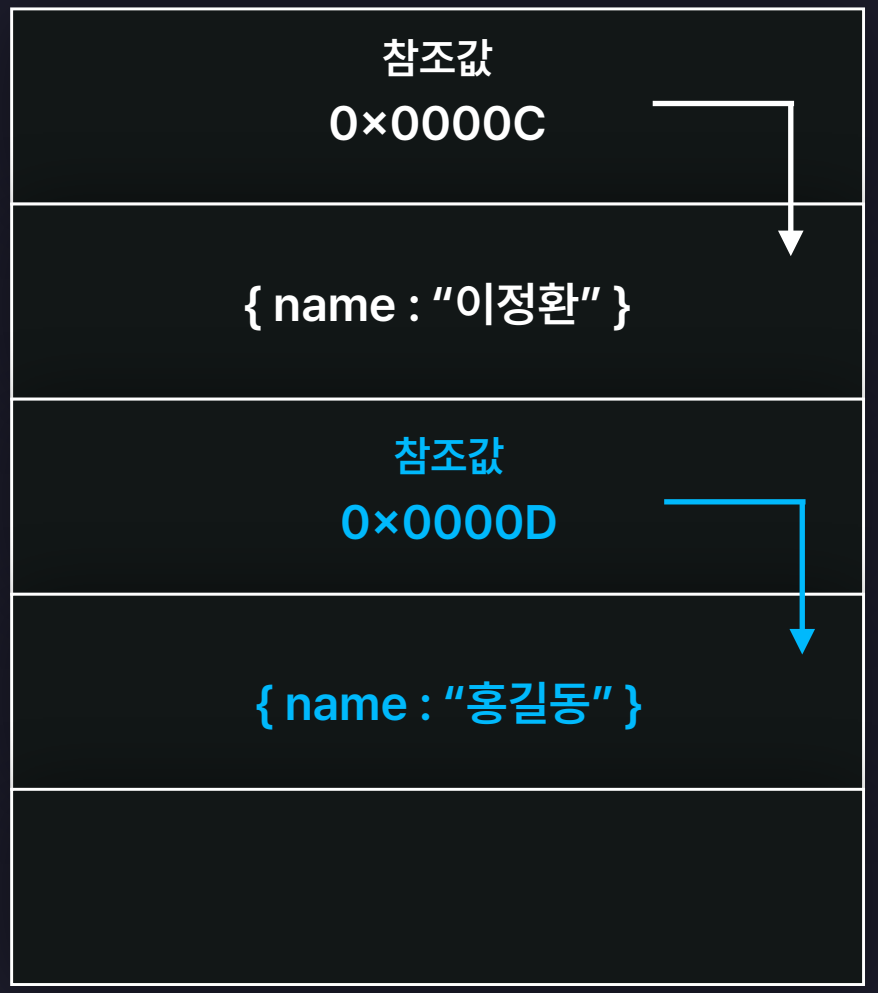
[Code]

```
let o1 = { name: "이정환" };
let o2 = o1;

o2.name = "홍길동";
```

[Name]



[Memory]

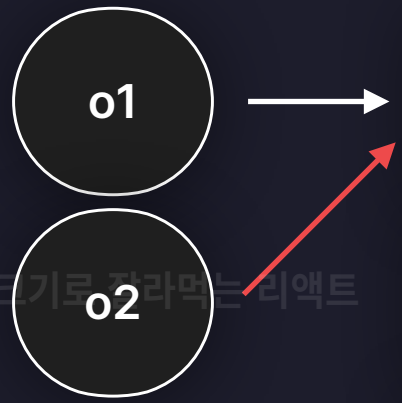


객체타입 = 가변값

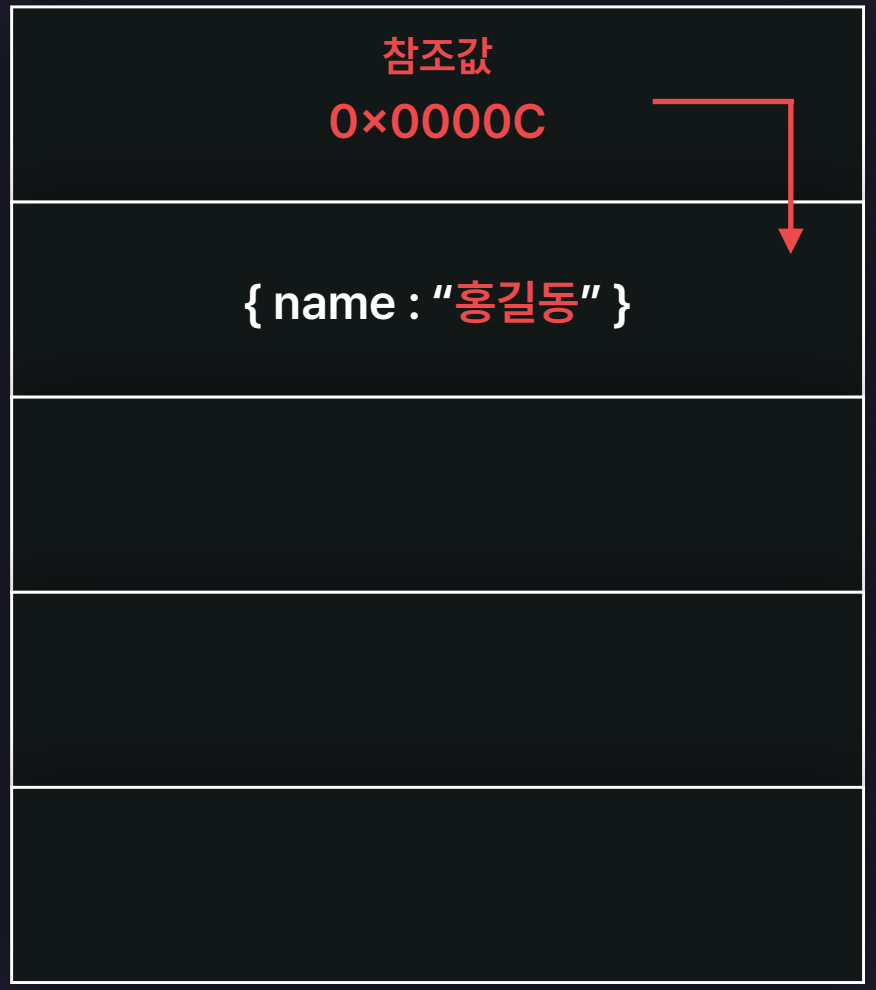
[Code]

```
let o1 = { name: "이정환" };
let o2 = o1;

o2.name = "홍길동";
```

[Name]



[Memory]



원시타입 VS 객체타입

원시 타입

Number, String, Boolean 등 ...

값 자체로써 변수에 저장되고 복사 된다

불변값이다 (메모리 값 수정 X)

VS

객체 타입

Object, Array, Function 등 ...

참조값을 통해 변수에 저장되고 복사된다

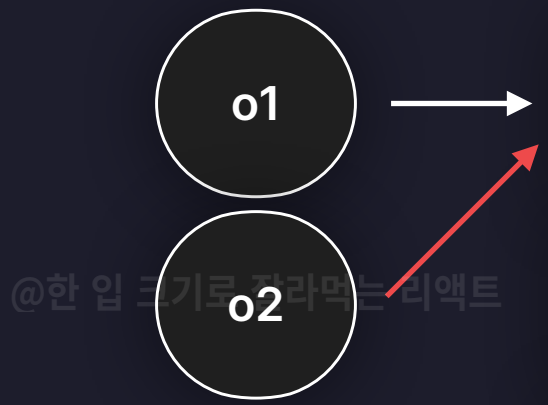
가변값이다 (메모리 값 수정 O)

객체 타입 주의사항 1. 의도치 않게 값이 수정될 수 있다.

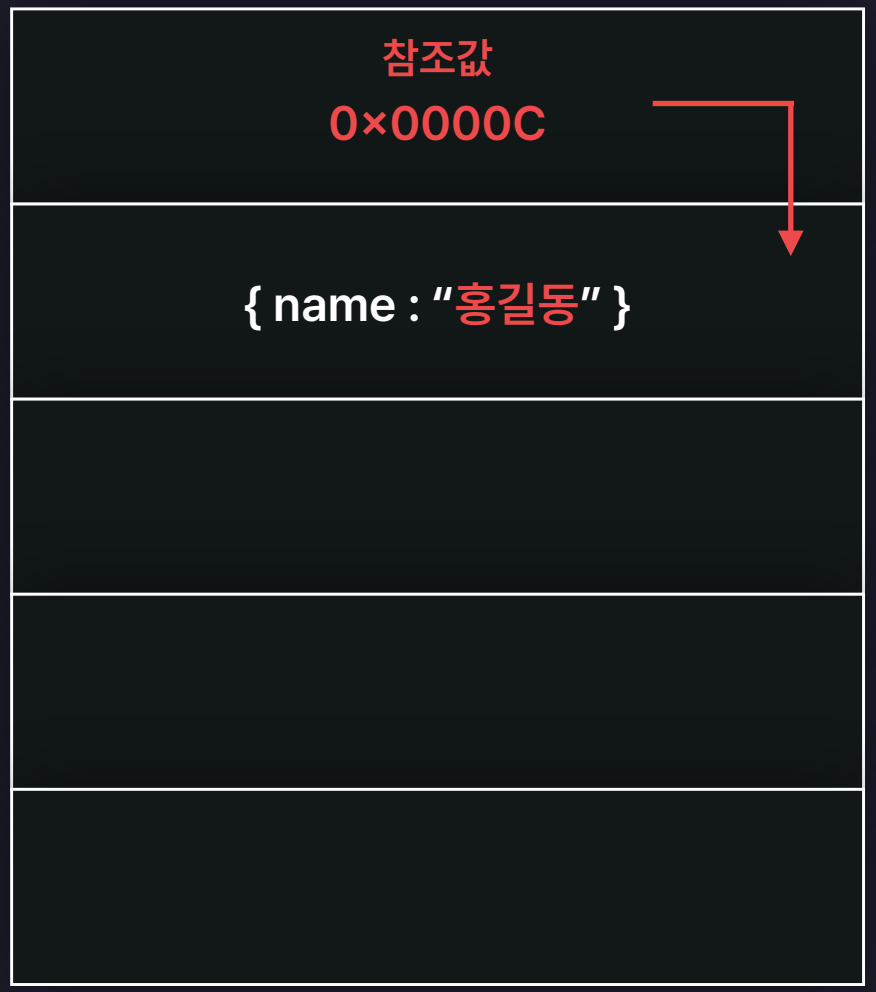
[Code]

```
let o1 = { name: "이정환" };
let o2 = o1;

o2.name = "홍길동";
```

[Name]



[Memory]



객체 타입 주의사항 1. 의도치 않게 값이 수정될 수 있다.

[Code]

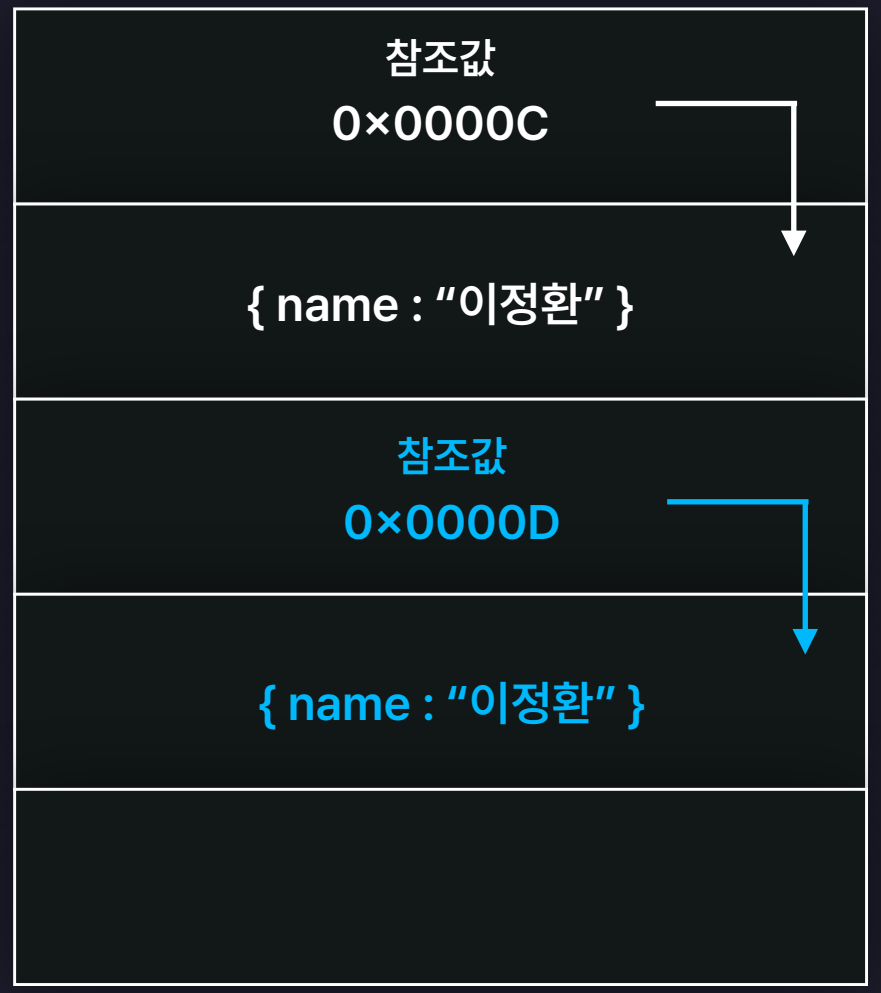
```
let o1 = { name: "이정환" };
let o2 = { ...o1 };
```

[Name]

o1

o2

[Memory]



객체 타입 주의사항 1. 의도치 않게 값이 수정될 수 있다.

[Code]

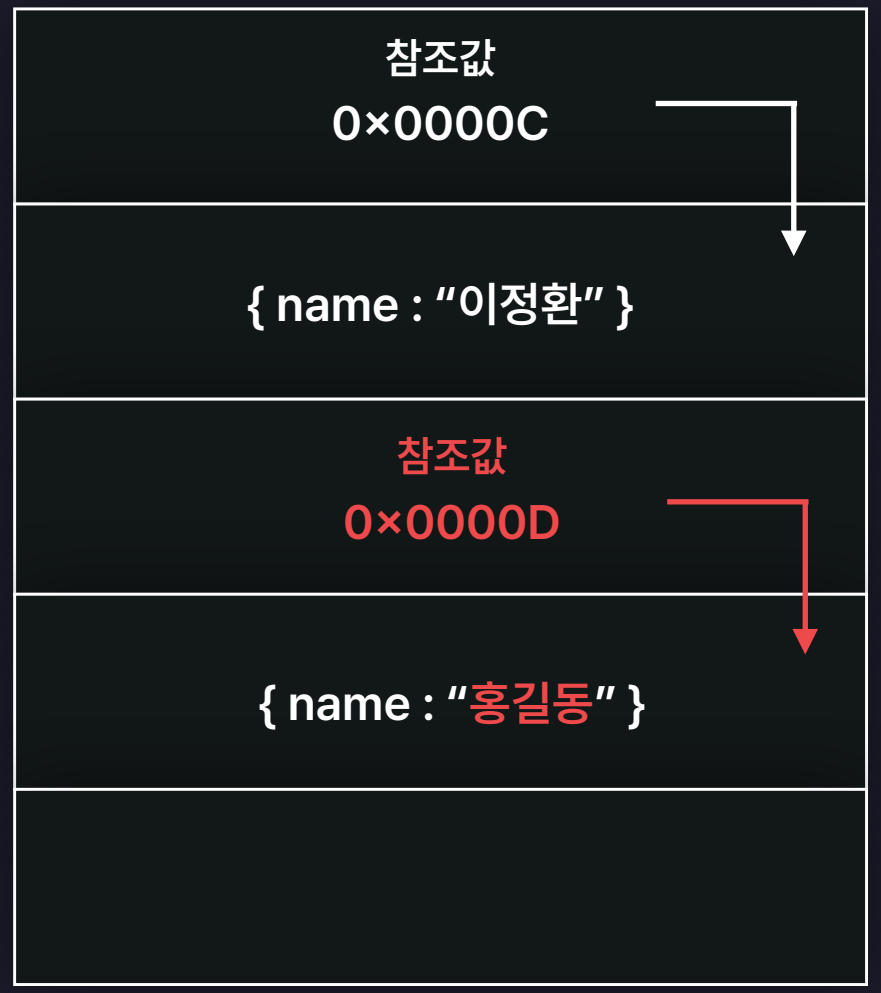
```
let o1 = { name: "이정환" };
let o2 = { ...o1 };

o2.name = "홍길동";
```

[Name]

o1

o2

[Memory]



객체 타입 주의사항 1. 의도치 않게 값이 수정될 수 있다.

얕은 복사

```
let o1 = { name: "이정환" };
let o2 = o1;
```

객체의 참조값을 복사함

원본 객체가 수정될 수 있어 위험함

깊은 복사

```
let o1 = { name: "이정환" };
let o2 = { ...o1 };
```

새로운 객체를 생성하면서

프로퍼티만 따로 복사 함

원본 객체가 수정될 일이 없어 안전함

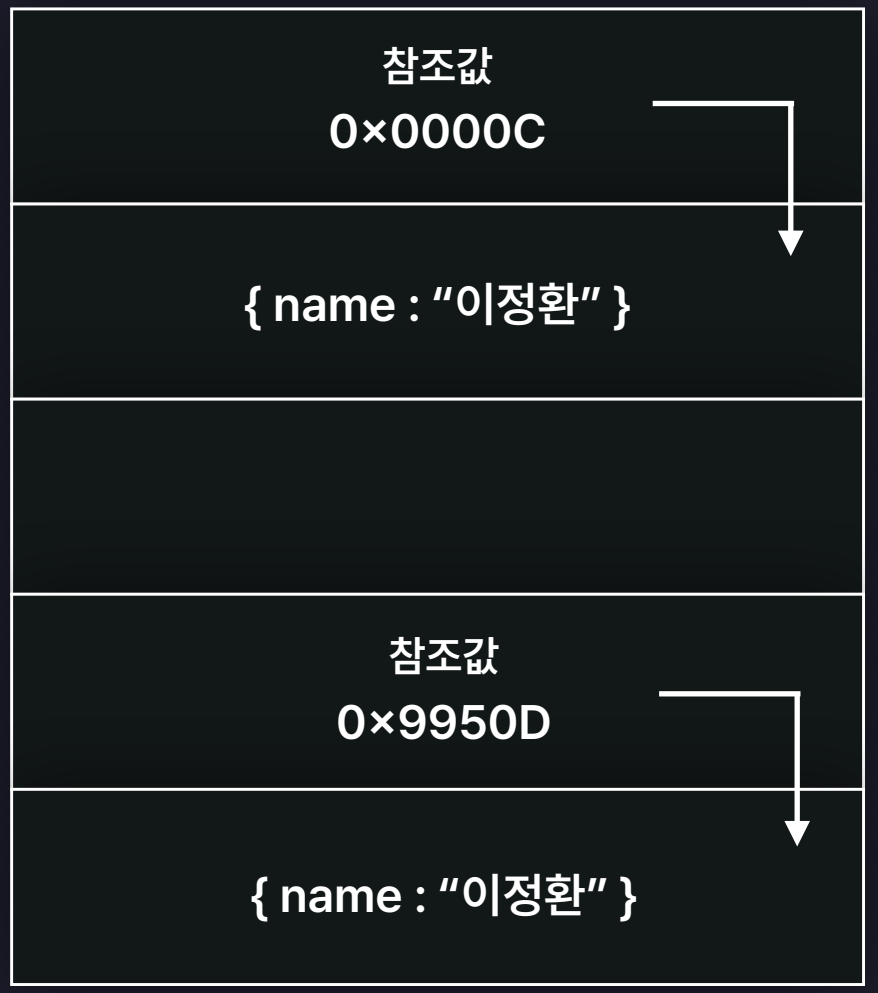
객체 타입 주의사항 2. 객체간의 비교는 기본적으로 참조값을 기준으로 이루어진다

[Code]

```
let o1 = { name: "이정환" };
let o2 = o1;
let o3 = { ...o1 };
```

[Name]

[Memory]



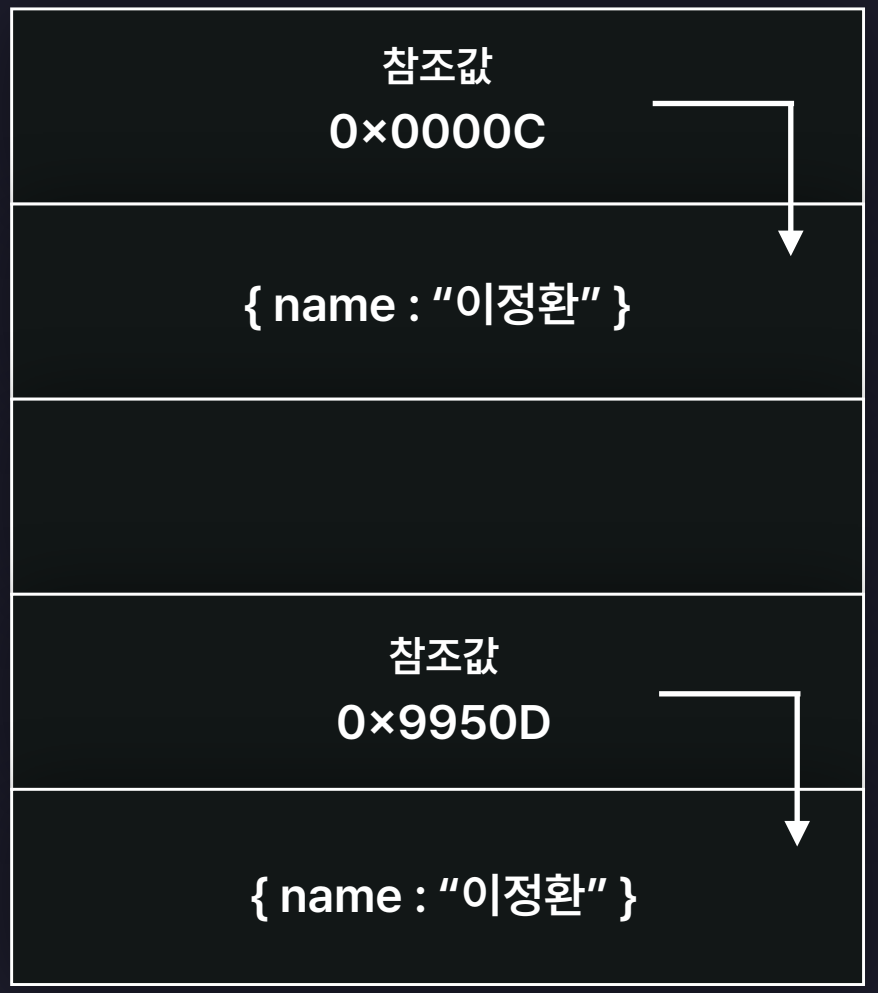
객체 타입 주의사항 2. 객체간의 비교는 기본적으로 참조값을 기준으로 이루어진다

[Code]

```
let o1 = { name: "이정환" };
let o2 = o1;
let o3 = { ...o1 };

console.log(o1 === o2);
```

[Name]

[Memory]



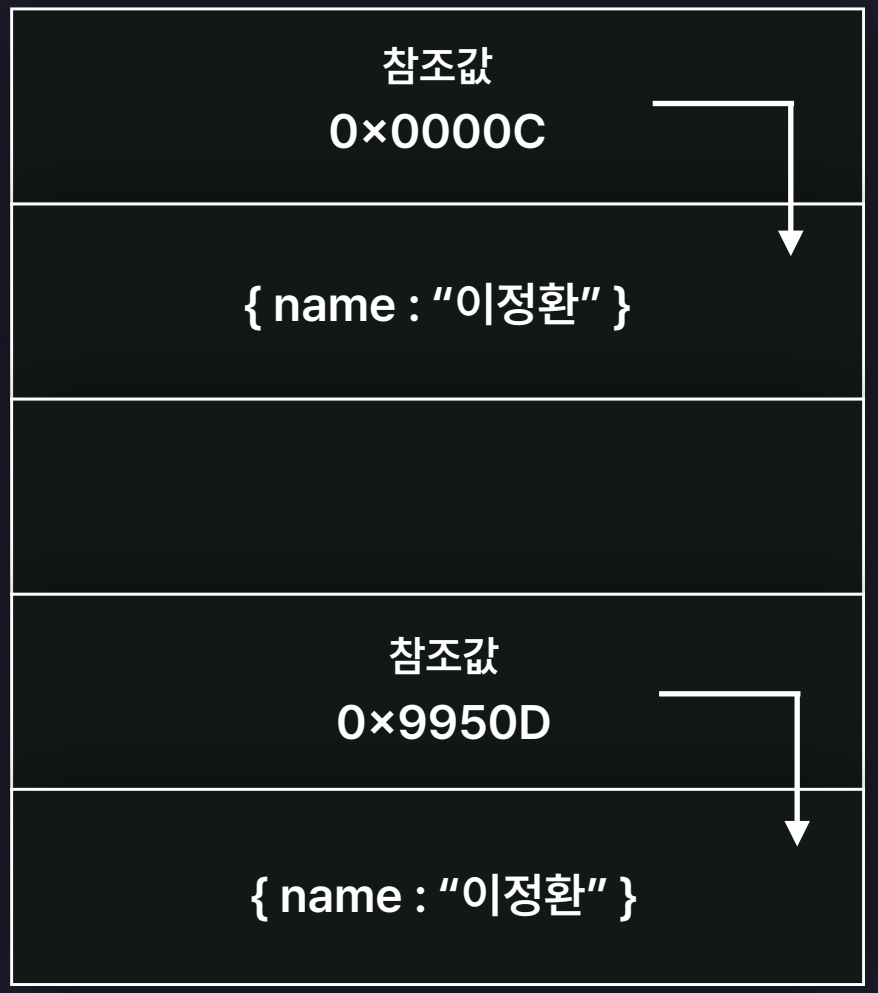
객체 타입 주의사항 2. 객체간의 비교는 기본적으로 참조값을 기준으로 이루어진다

[Code]

```
let o1 = { name: "이정환" };
let o2 = o1;
let o3 = { ...o1 };

console.log(o1 === o2);
console.log(o1 === o3);
```

[Name]

[Memory]



객체 타입 주의사항 2. 객체간의 비교는 기본적으로 참조값을 기준으로 이루어진다

[Code]

```
let o1 = { name: "이정환" };
let o2 = o1;
let o3 = { ...o1 };

console.log(o1 === o2);
console.log(o1 === o3);

console.log(
  JSON.stringify(o1) === JSON.stringify(o3)
);

// 결과 : True
```


JSON.stringify()

자바스크립트 내장 함수

객체를 문자열로 변환하는 기능

객체 타입 주의사항 2. 객체간의 비교는 기본적으로 참조값을 기준으로 이루어진다

얕은 비교

```
o1 === o2
```

참조값을 기준으로 비교

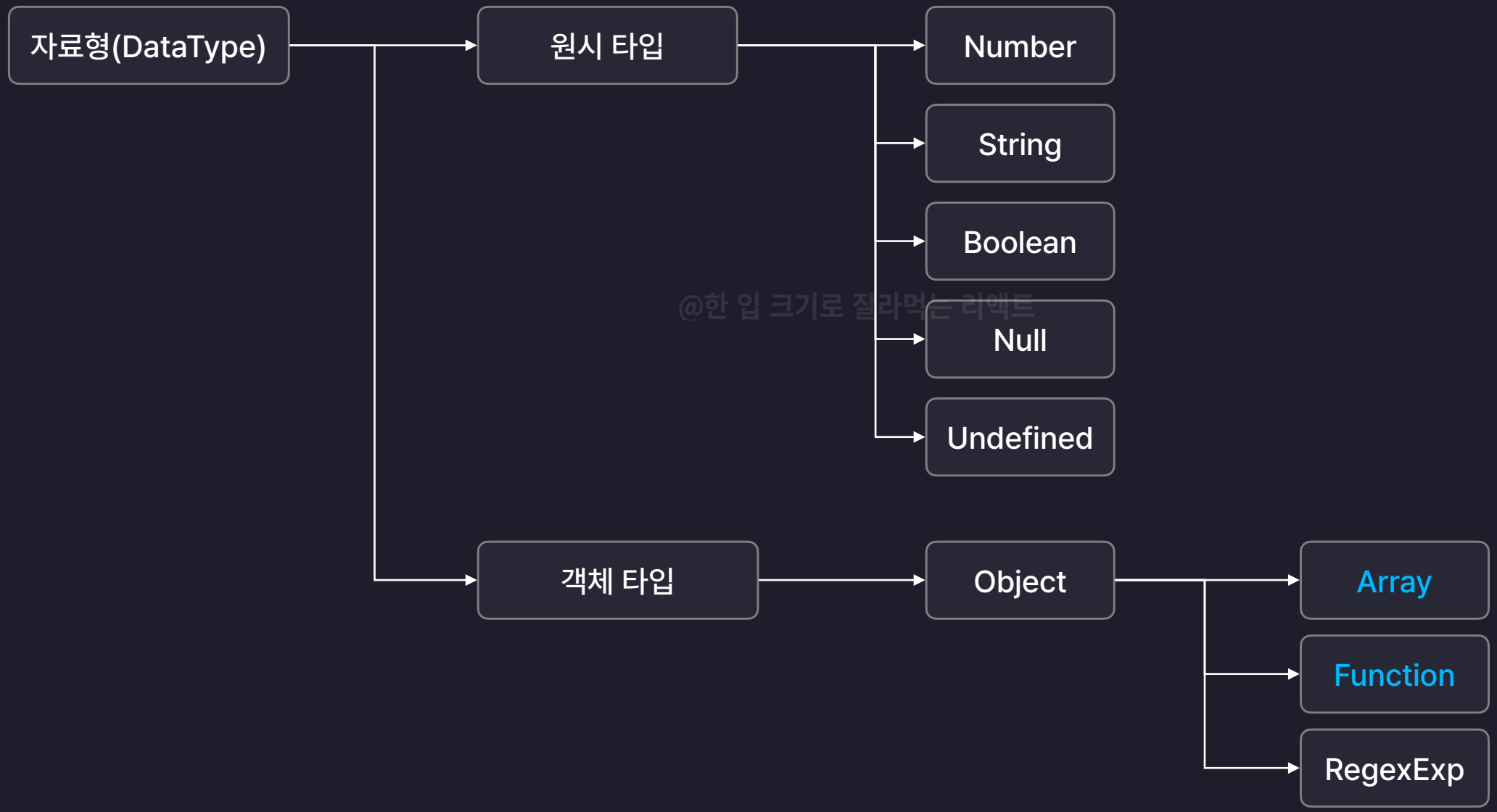
깊은 비교

```
JSON.stringify(o1) === JSON.stringify(o2)
```

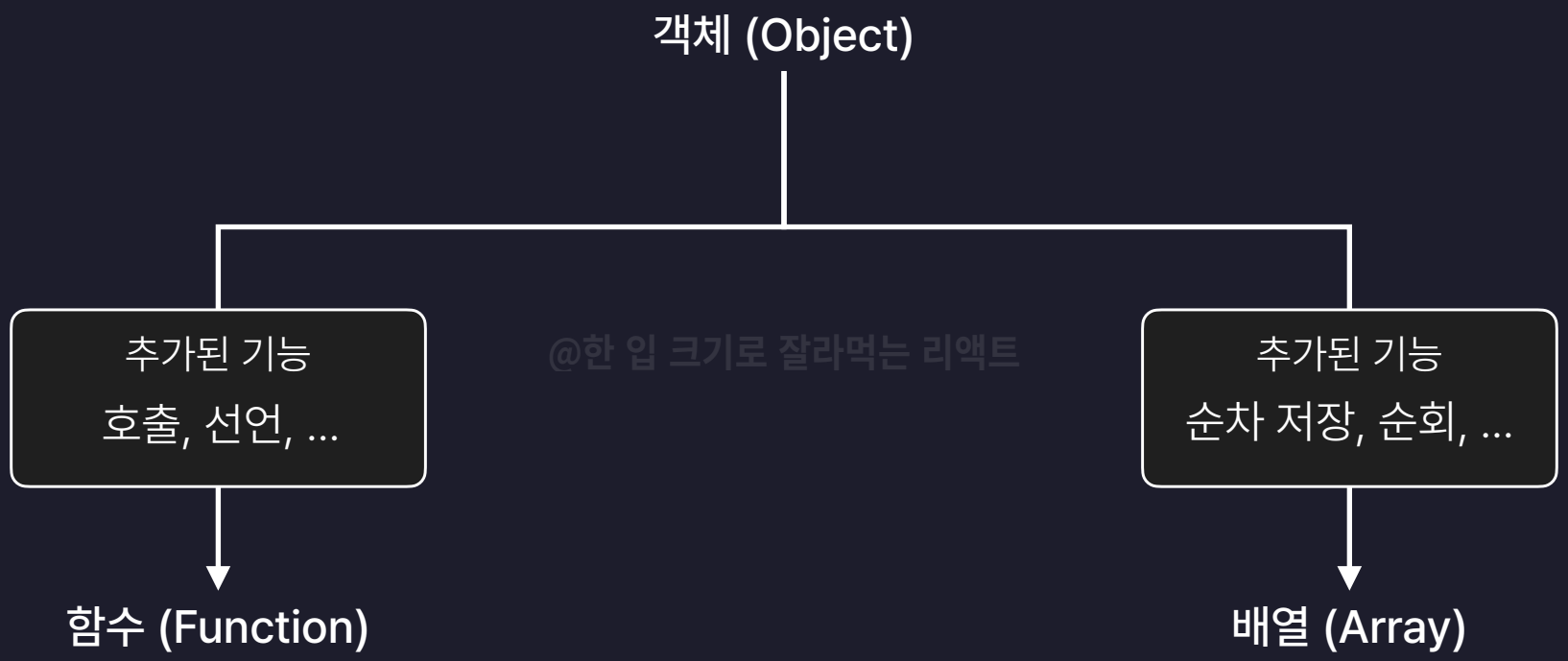
객체를 문자열로 변환하여 비교

JSON.stringify 등의 내장 함수를 이용해야 함

객체 타입 주의사항 3. 배열과 함수도 사실 객체이다



객체 타입 주의사항 3. 배열과 함수도 사실 객체이다



한입 크기로 잘라먹는

반복문으로 배열과 객체 순회하기 🔁

순회(Iteration)이란?

- 배열, 객체에 저장된 여러개의 값에 순서대로 하나씩 접근하는 것을 말함

ex) 배열 순회

```
let numbers = [1, 2, 3];
```


ex) 객체 순회

```
let person = {
  name: "이정환",
  age: 27,
  hobby: "테니스",
};
```

순회(Iteration)이란?

- 배열, 객체에 저장된 여러개의 값에 순서대로 하나씩 접근하는 것을 말함

반복문을 이용한 배열, 객체 순회

```
for (let value of numbers) {
  console.log(valule);
}

for (let key in Object.keys(person)) {
  console.log(key);
}
```

한입 크기로 잘라먹는

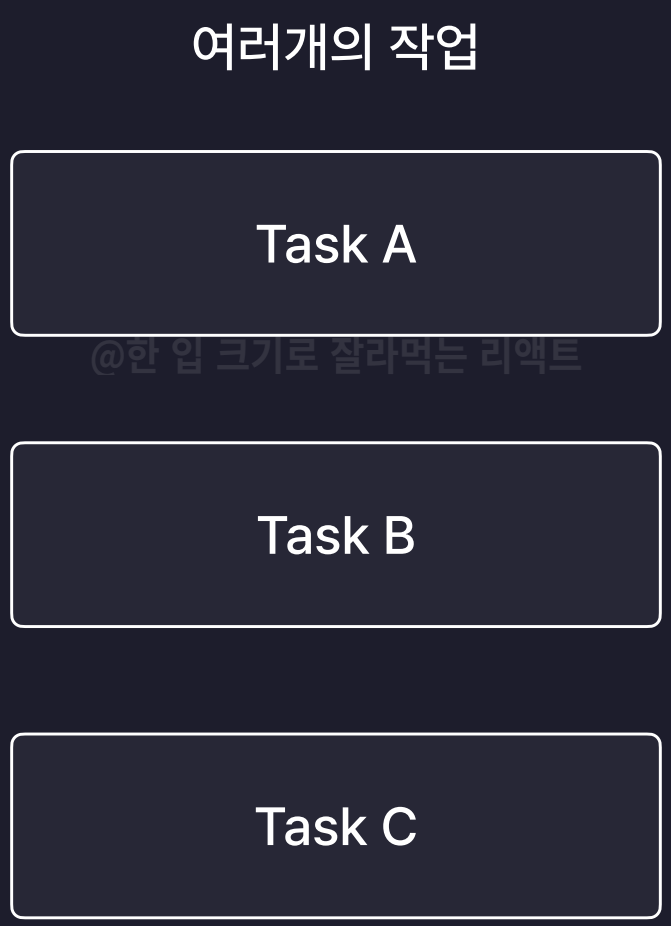
동기와 비동기 🤖

한입 크기로 잘라먹는 리액트

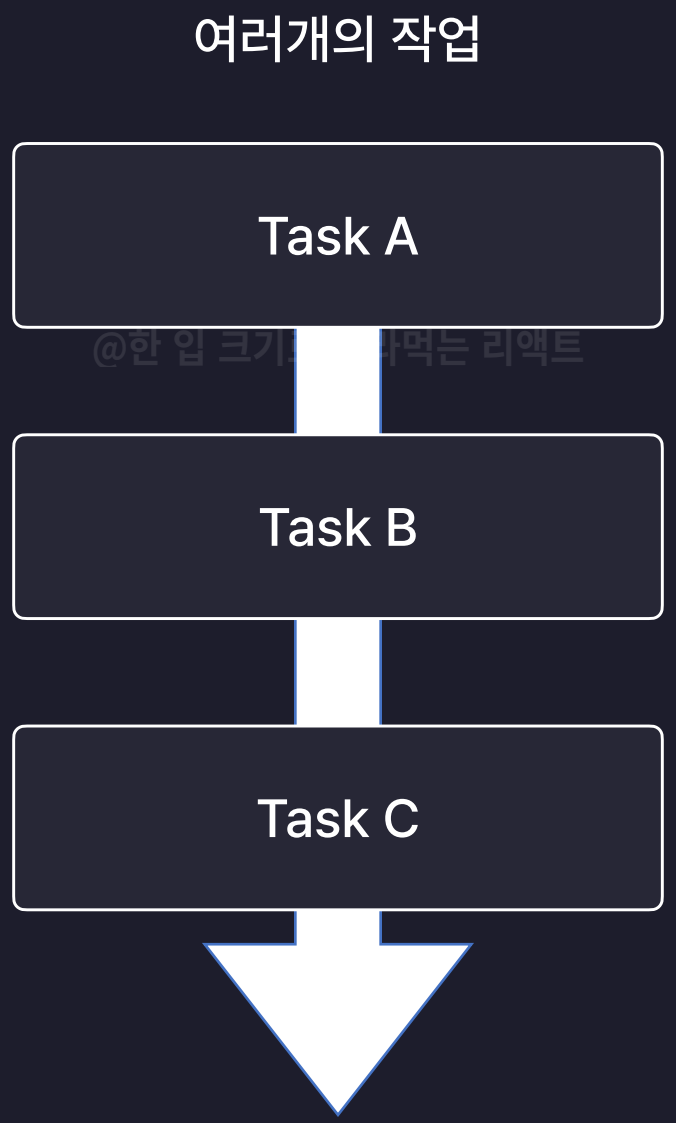
"동기"란 무엇일까?

대학 동기?, 회사 동기?

동기란 무엇일까?

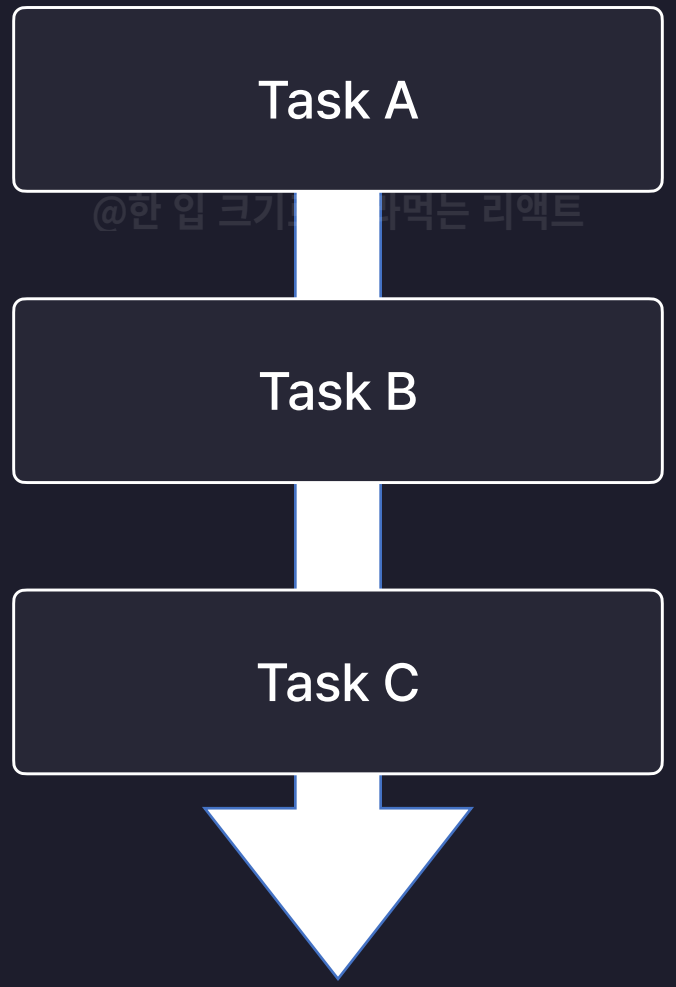


동기란 무엇일까?



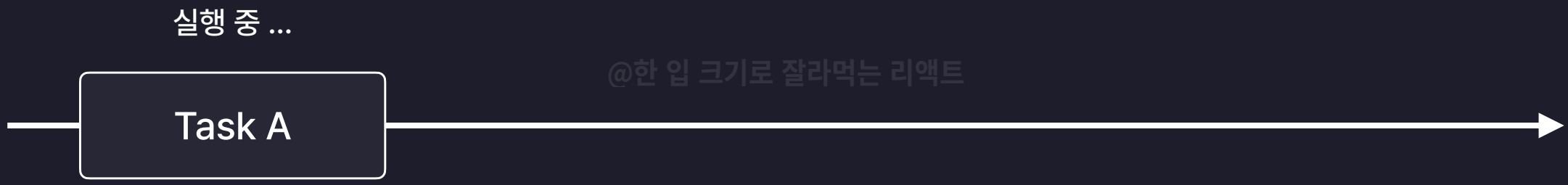
동기란 무엇일까?

"동기적으로 처리한다"

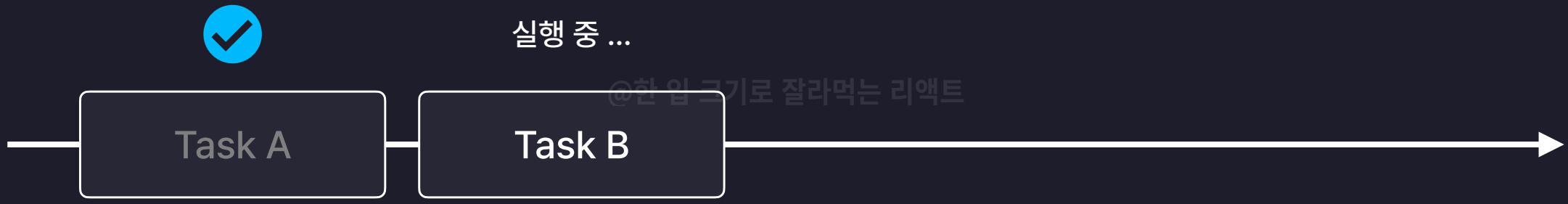


동기란 무엇일까?

동기란 무엇일까?



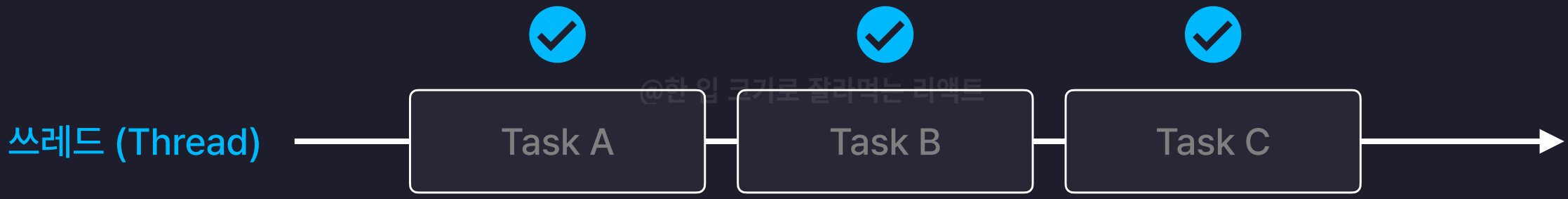
동기란 무엇일까?



동기란 무엇일까?

동기란 무엇일까?

동기란 무엇일까?



"동기"란 무엇일까?

여러개의 작업을 순서대로, 하나씩 처리하는 방식

JavaScript는 “동기”적으로 코드를 실행한다

JavaScript 코드

```
console.log("1");
console.log("2");
console.log("3");
```


@한 입 크기로 잘라먹는 리액트

실행 결과

```
1
2
3
```

JavaScript는 “동기”적으로 코드를 실행한다

JavaScript 코드

```
function taskA() {
  console.log("2");
}

console.log("1");
taskA();
console.log("3");
```

실행 결과

```
1
2
3
```

JavaScript는 “동기”적으로 코드를 실행한다

JavaScript 코드

```
function taskA() {
  console.log("2");
}

console.log("1");
taskA();
console.log("3");
```

실행 결과

```
1
2
3
```

동기 방식에는 치명적인 단점이 존재한다

JavaScript 코드

```
function taskA() {...}
function taskB() {...}
function taskC() {...}

taskA();
taskB();
taskC();
```

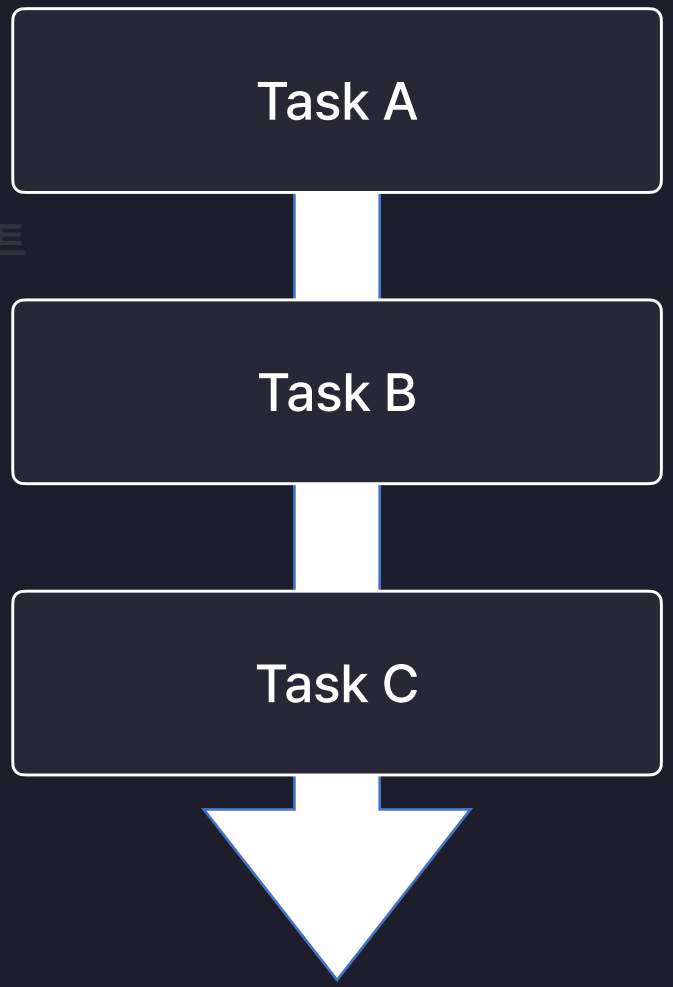
동기 방식에는 치명적인 단점이 존재한다

JavaScript 코드

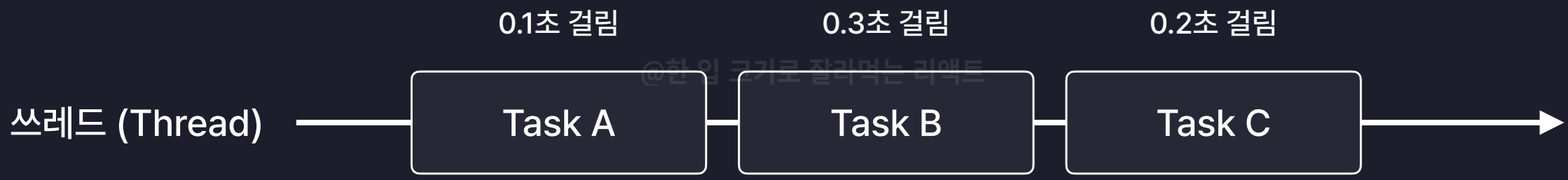
```
function taskA() {...}
function taskB() {...}
function taskC() {...}

taskA();
taskB();
taskC();
```

@한 입 크기로 잘라먹는 리액트

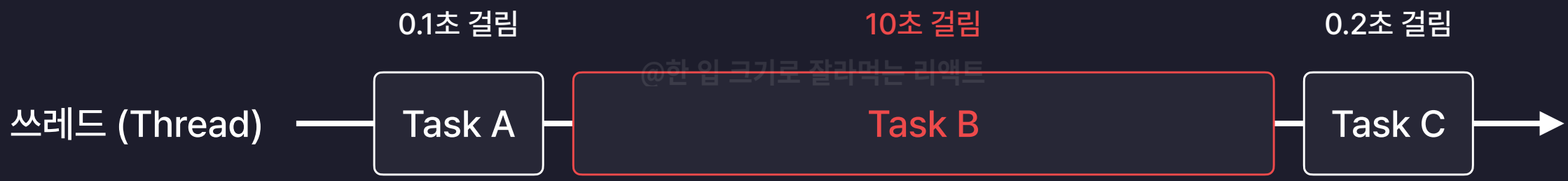
실행



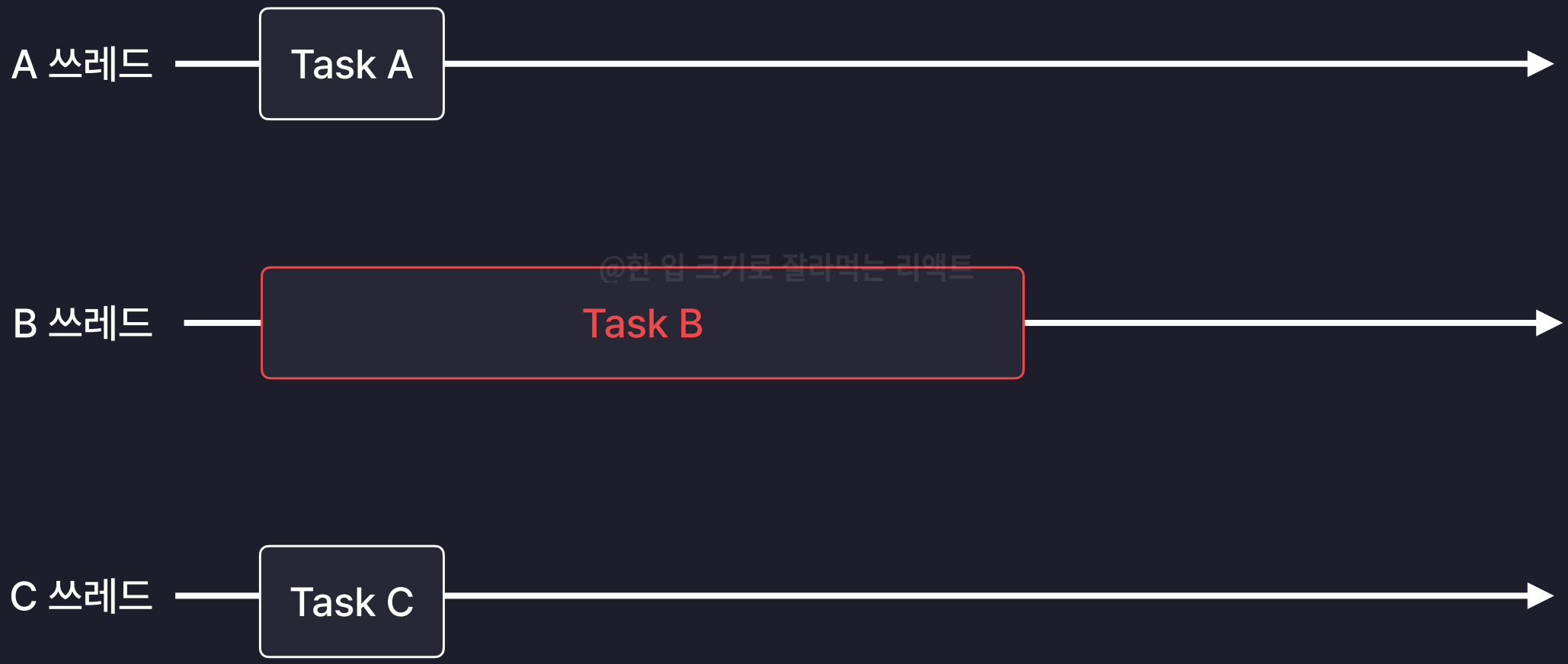
동기 방식에는 치명적인 단점이 존재한다



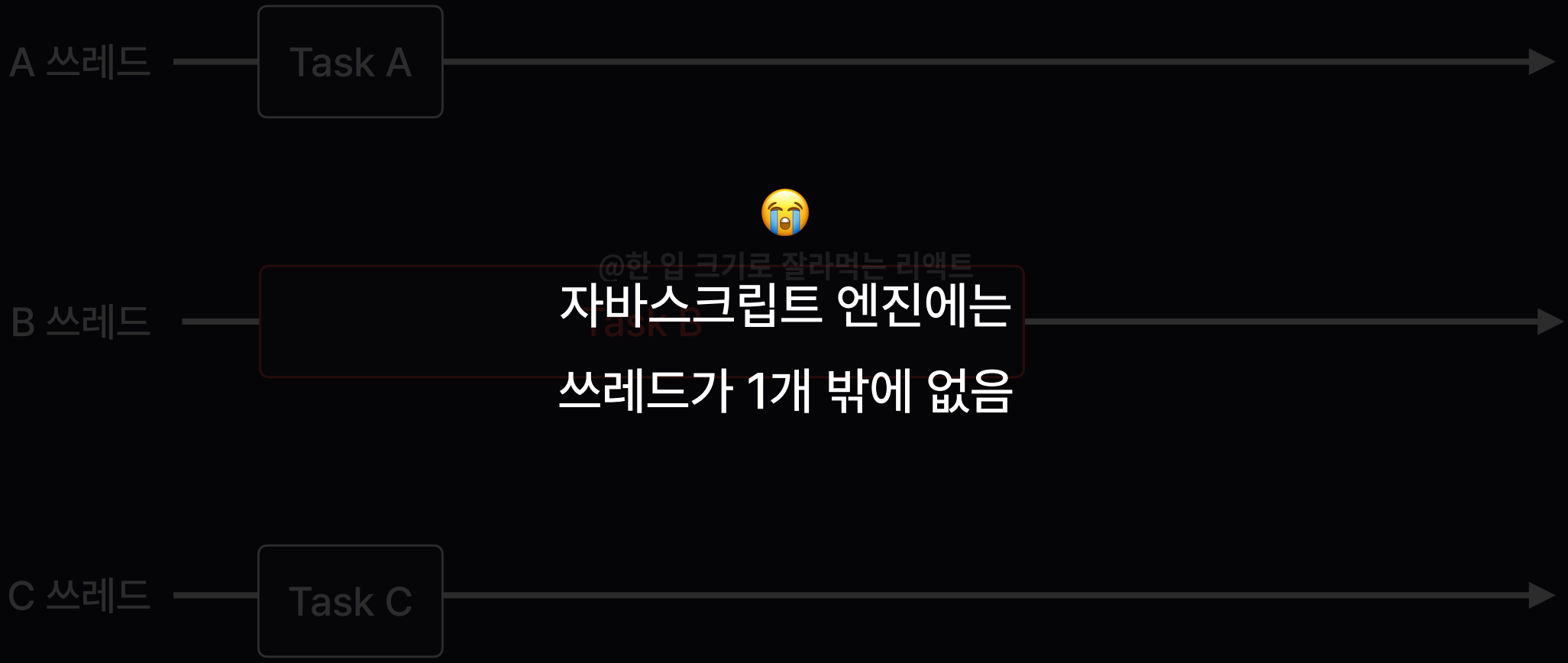
동기 방식에는 치명적인 단점이 존재한다



동기 방식에는 치명적인 단점이 존재한다



동기 방식에는 치명적인 단점이 존재한다



😭

@한 입 크기로 잘라먹는 리액트

자바스크립트 엔진에는

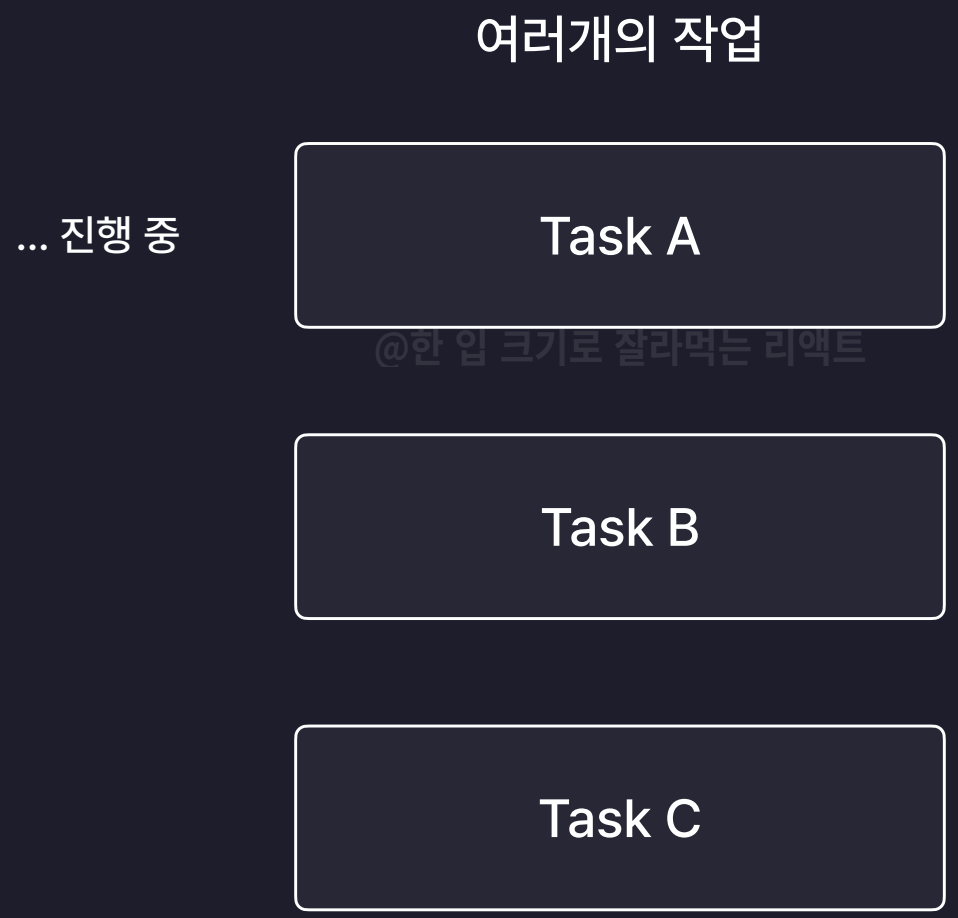
쓰레드가 1개 밖에 없음

“비동기”란 무엇일까?

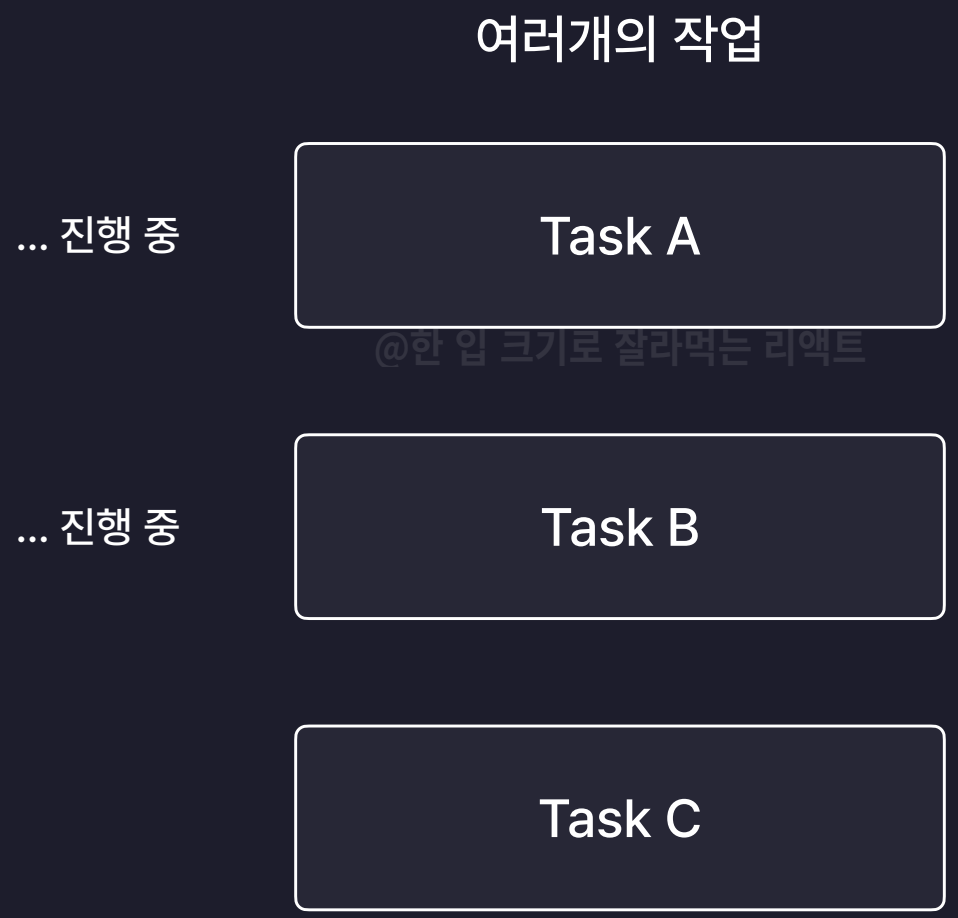
동기적이지 않다는 뜻

작업을 순서대로 처리하지 않음

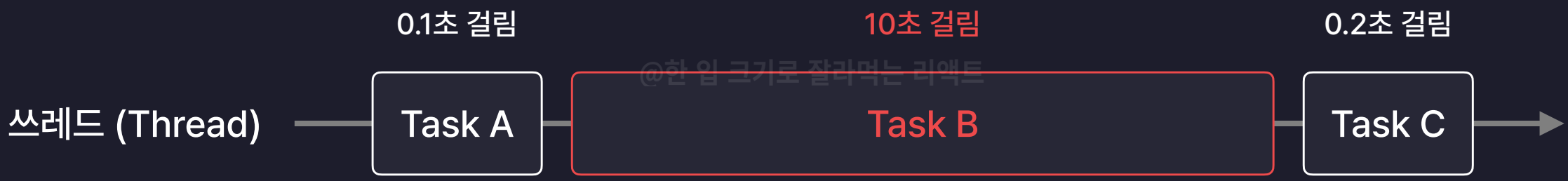
비동기란 무엇일까?



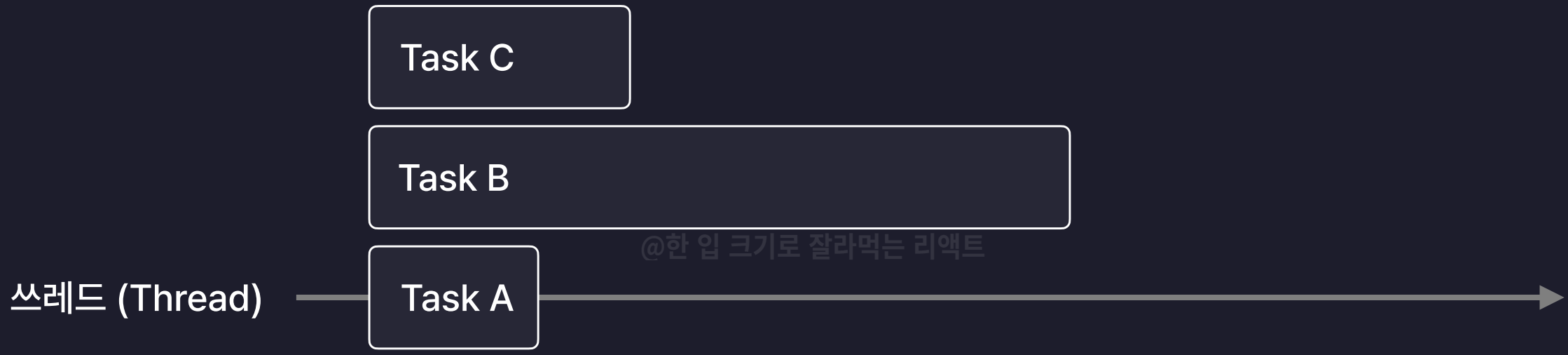
비동기란 무엇일까?



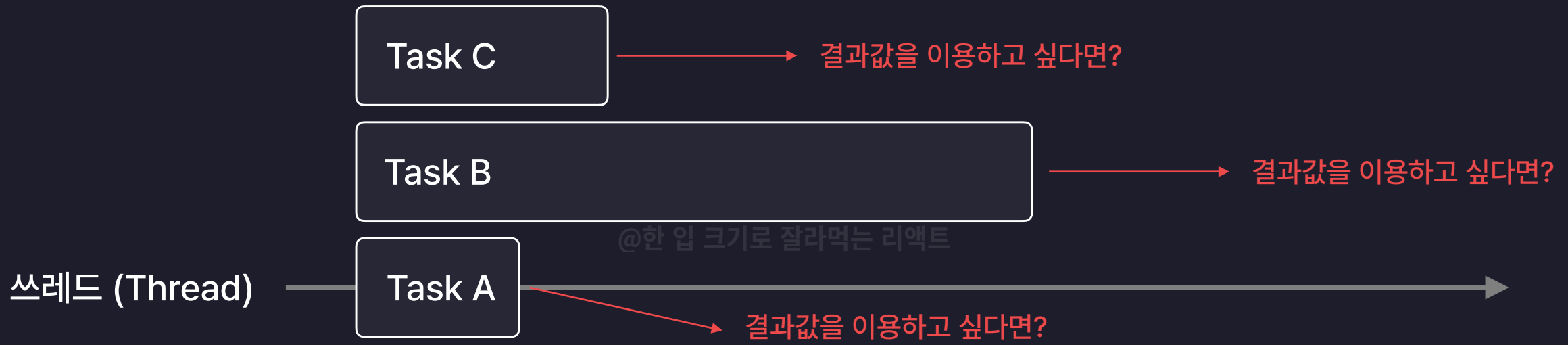
비동기란 무엇일까?



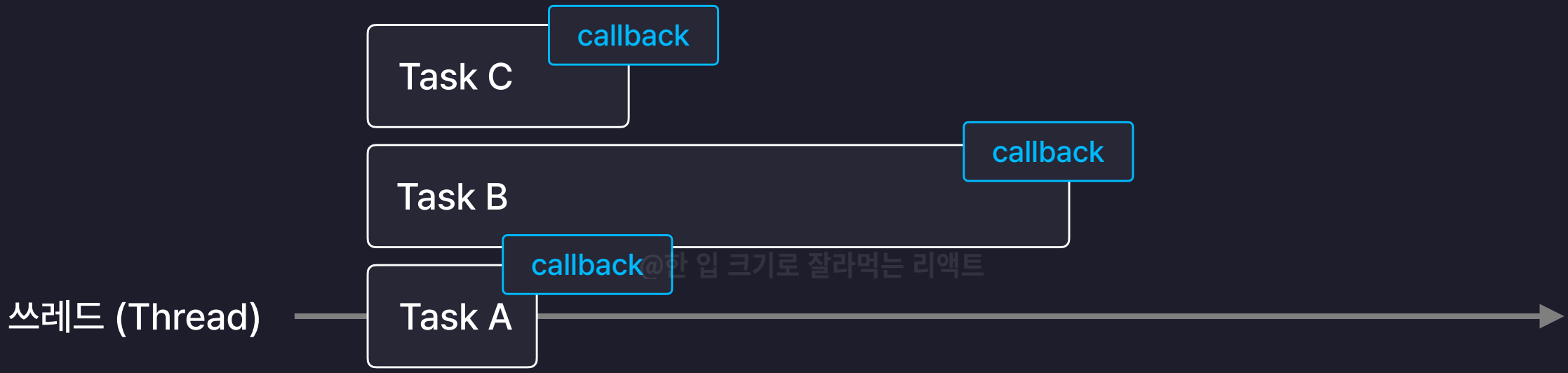
비동기란 무엇일까?



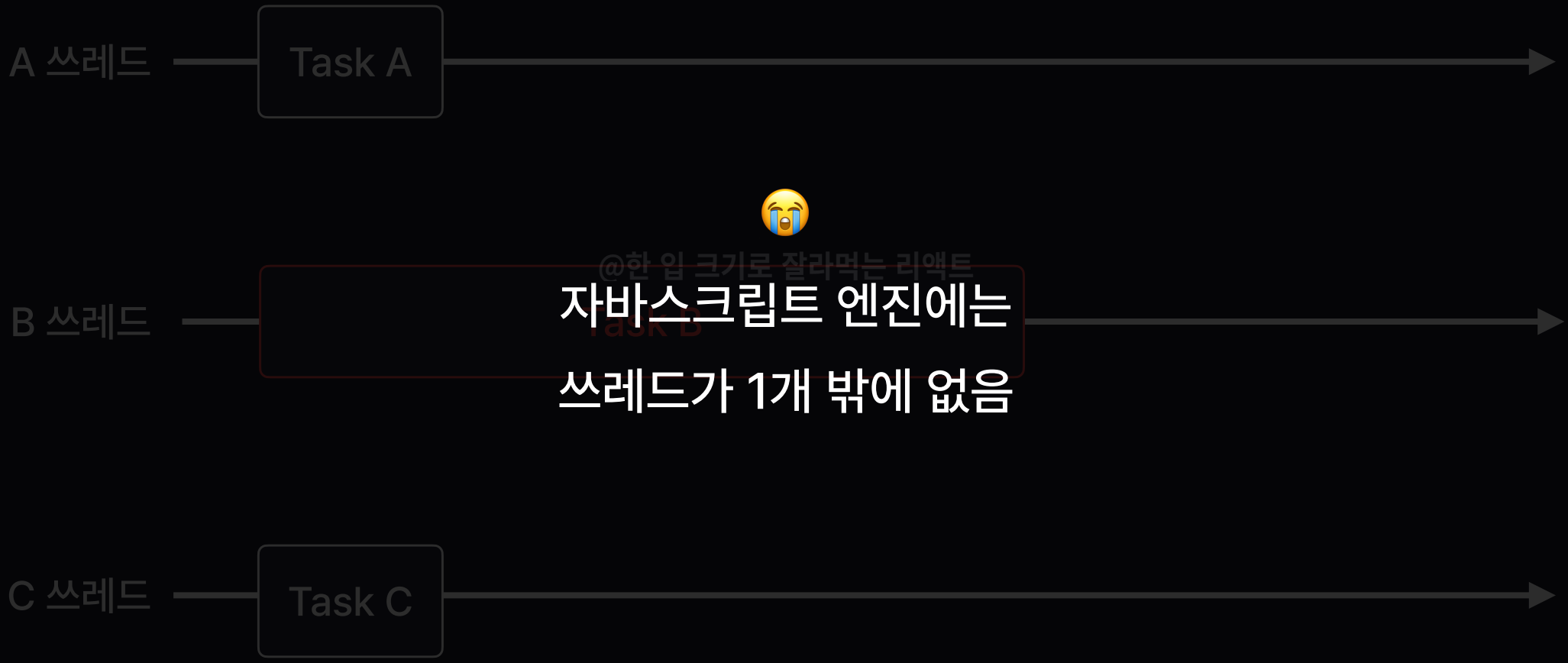
비동기란 무엇일까?


Task C
결과값을 이용하고 싶다면?
Task B
결과값을 이용하고 싶다면?
@한 입 크기로 잘라먹는 리액트
쓰레드 (Thread)
Task A
결과값을 이용하고 싶다면?

비동기란 무엇일까?



동기 방식에는 치명적인 단점이 존재한다



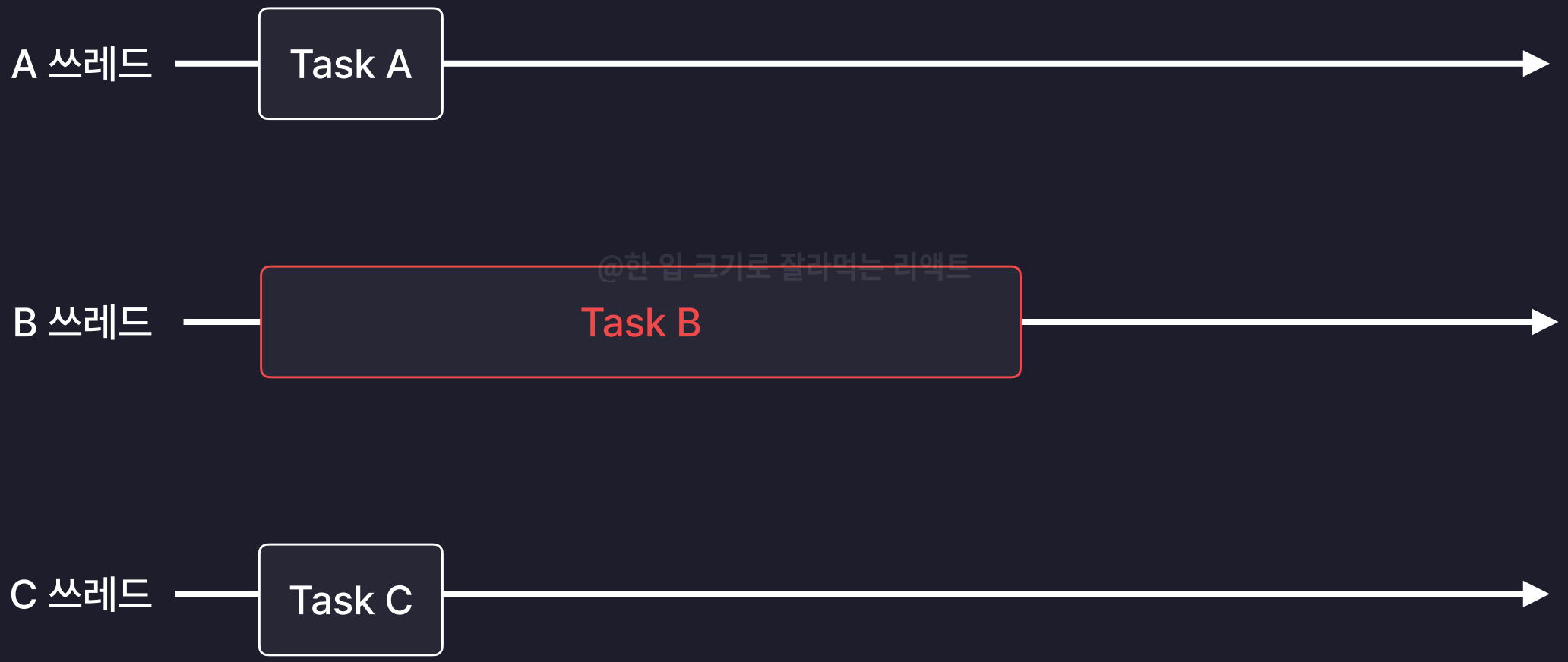
😭

@한 입 크기로 잘라먹는 리액트

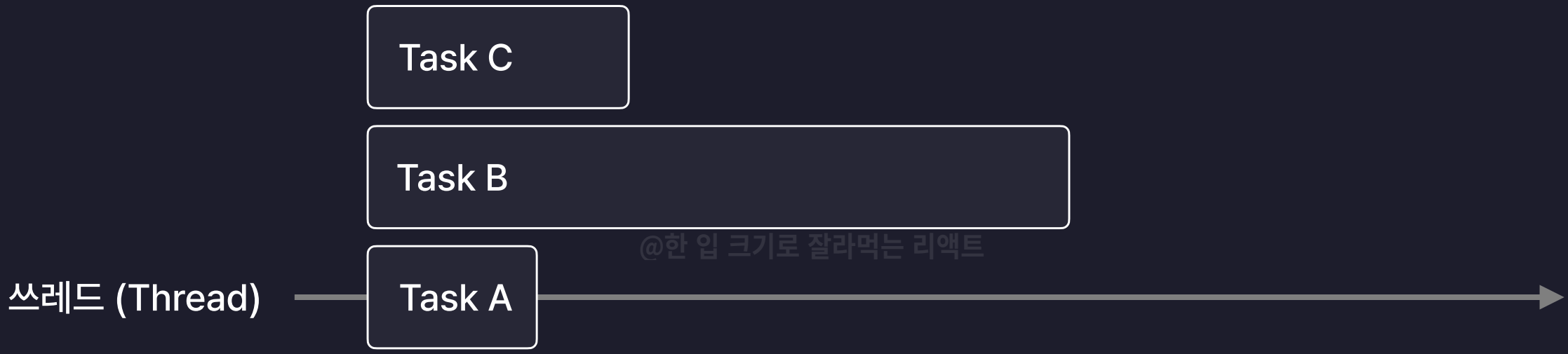
자바스크립트 엔진에는

쓰레드가 1개 밖에 없음

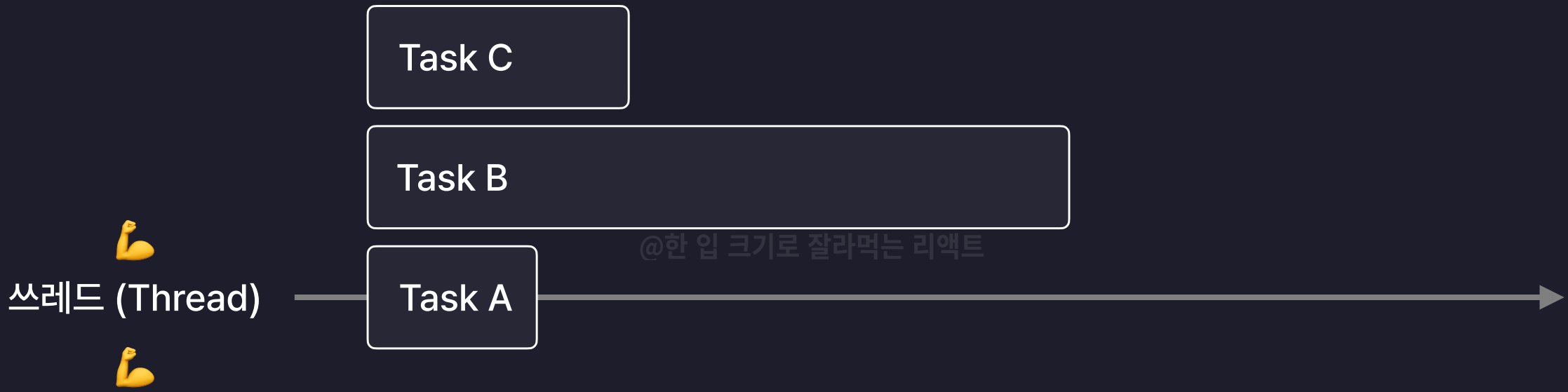
의문



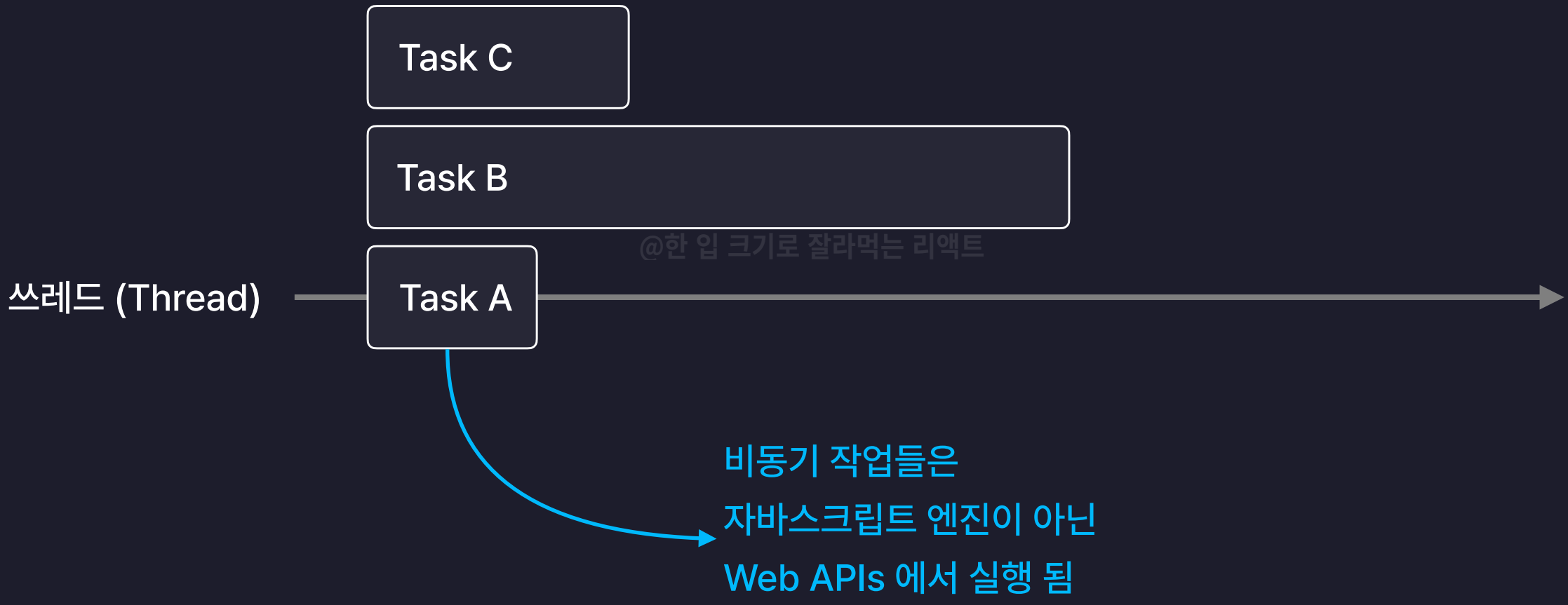
🤔 어떻게 동시에 작업을 처리하는거지?



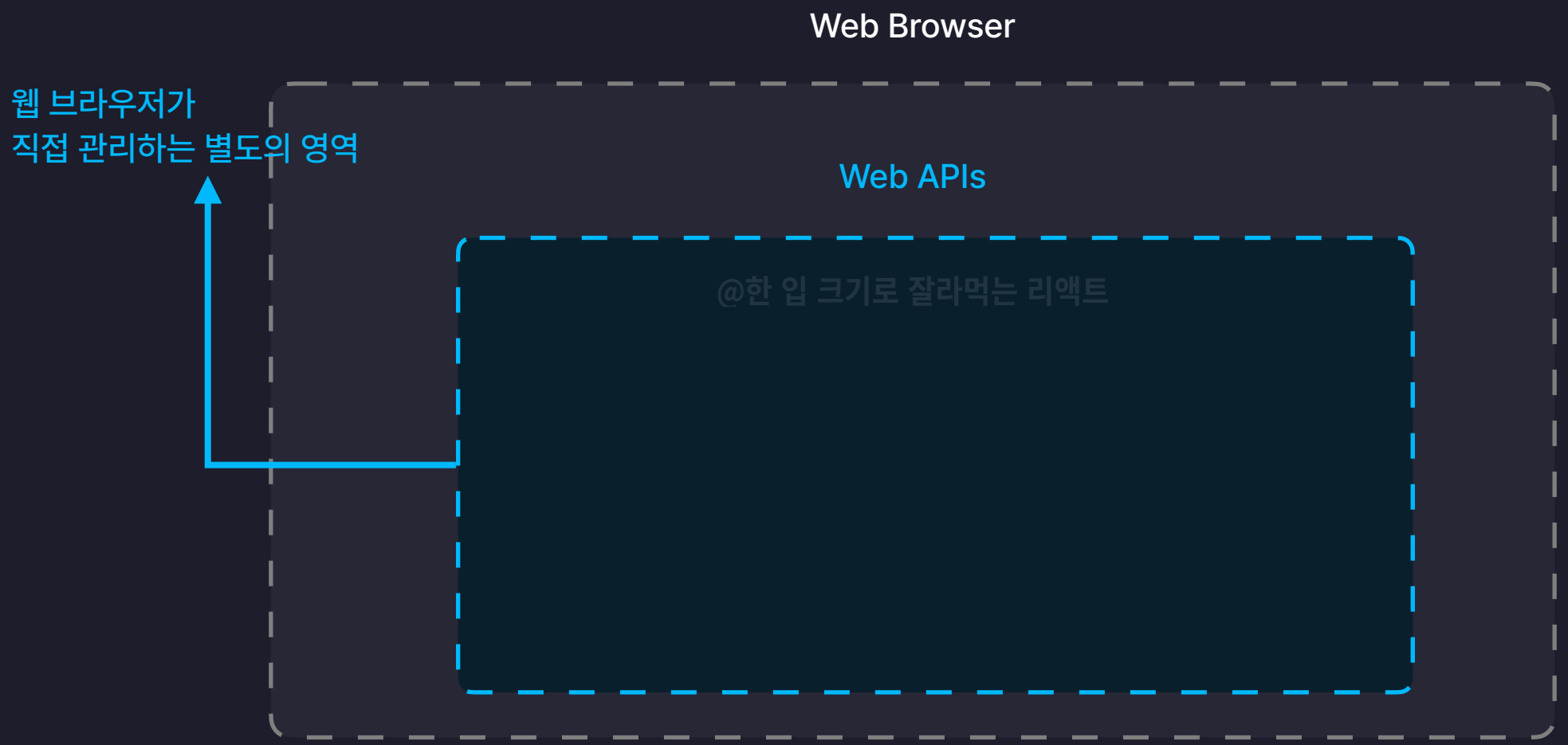
🤔 어떻게 동시에 작업을 처리하는거지?



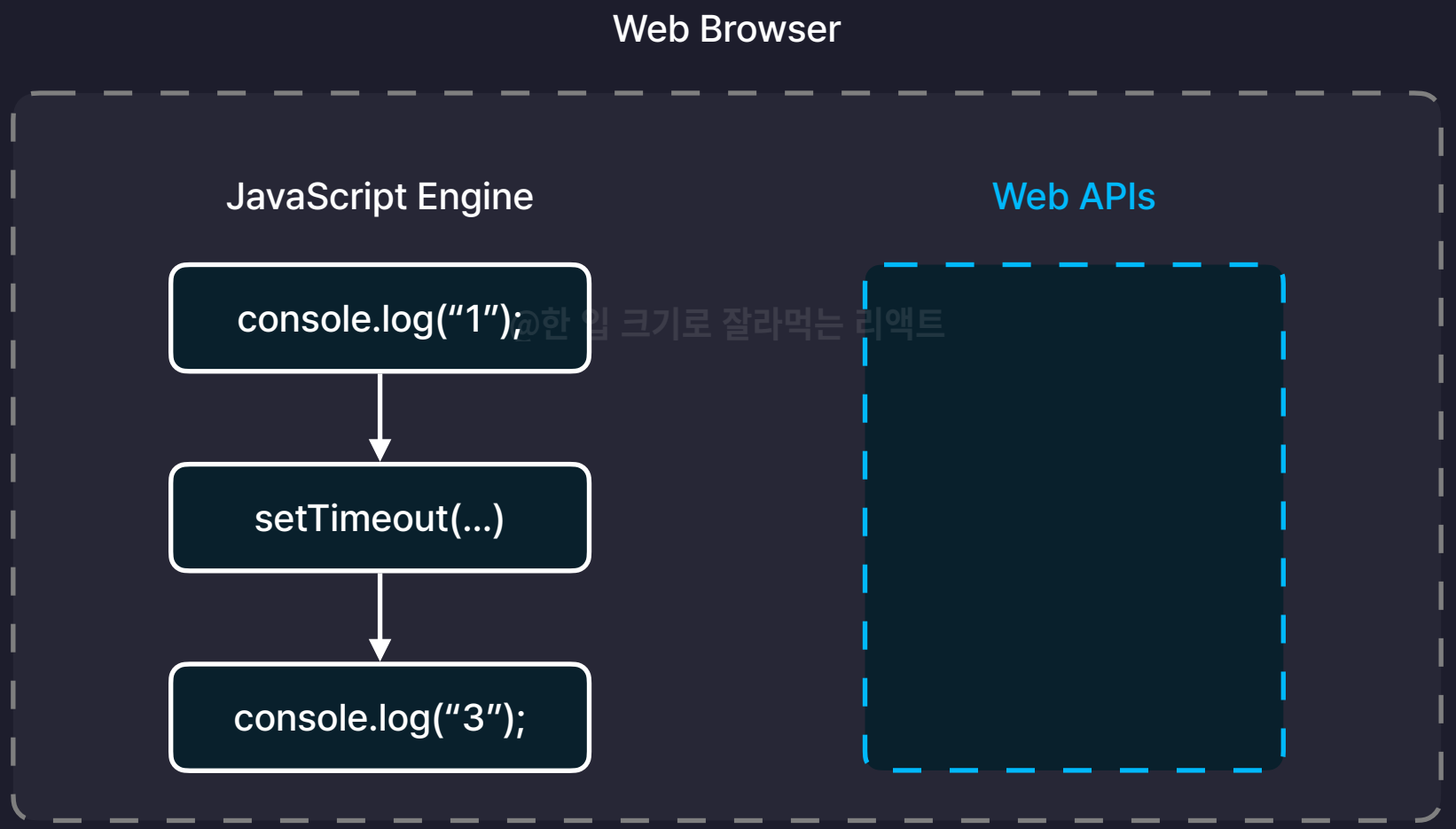
🤔 어떻게 동시에 작업을 처리하는거지?



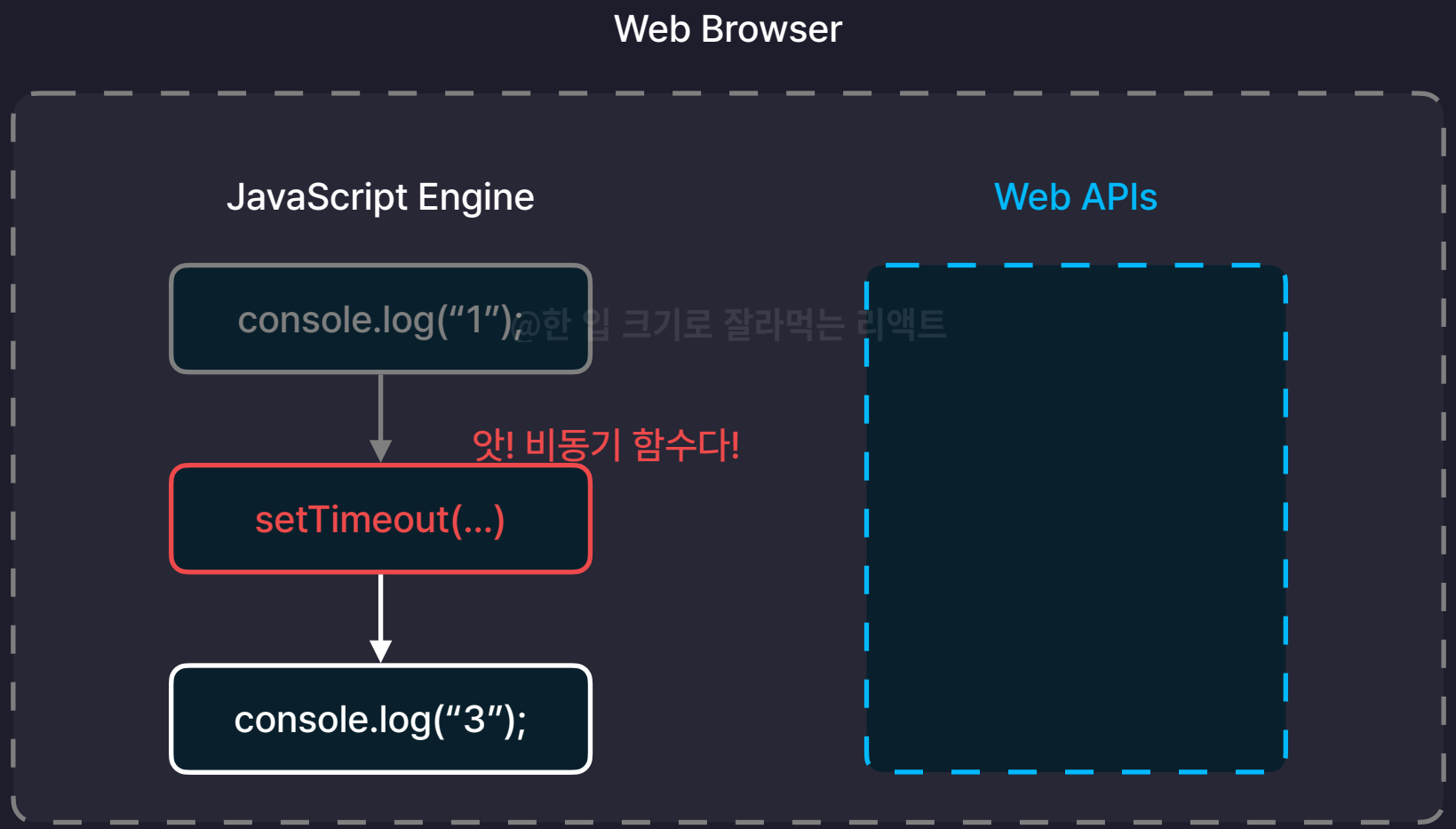
🤔 어떻게 동시에 작업을 처리하는거지?



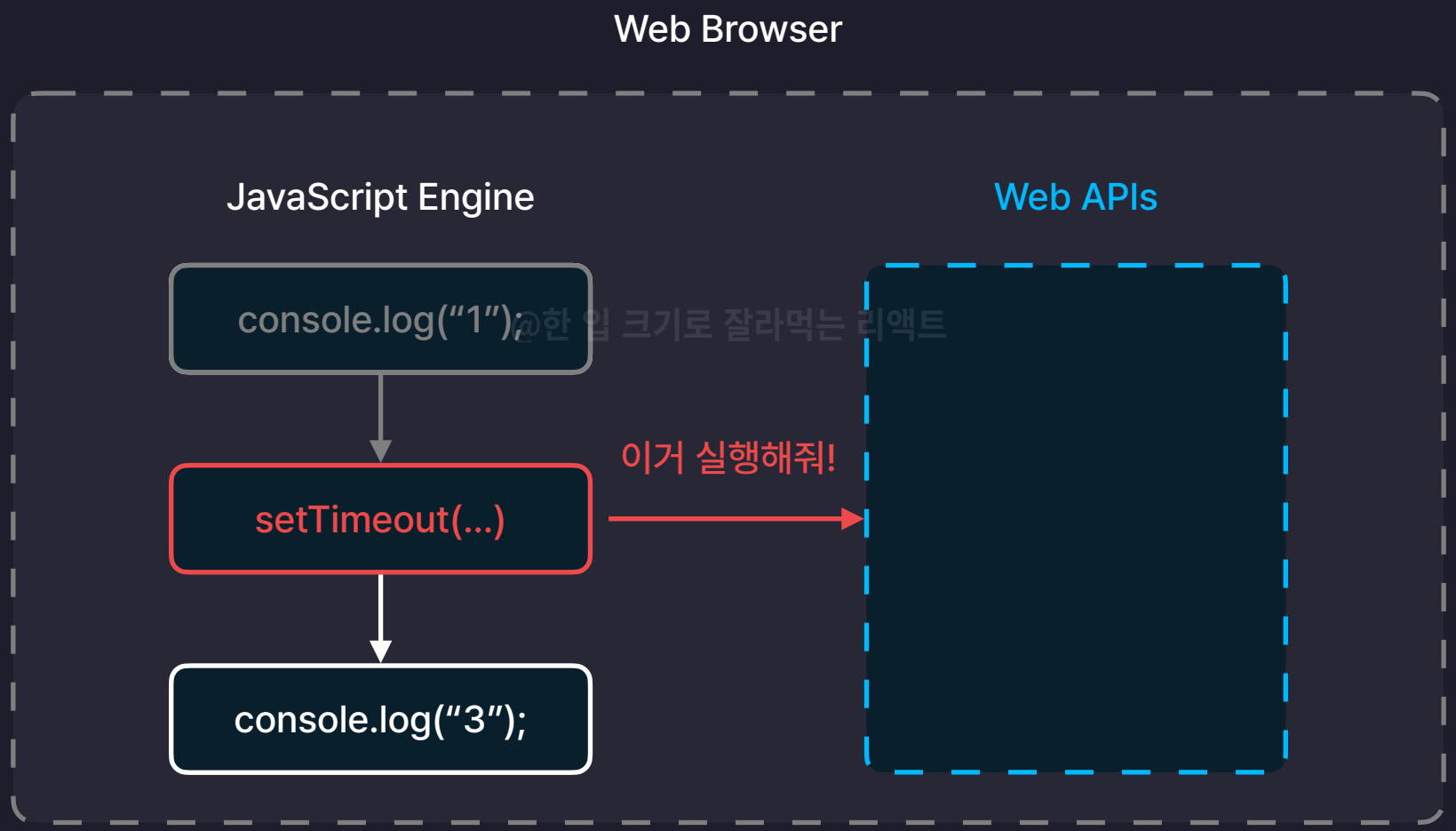
JavaScript의 비동기 처리



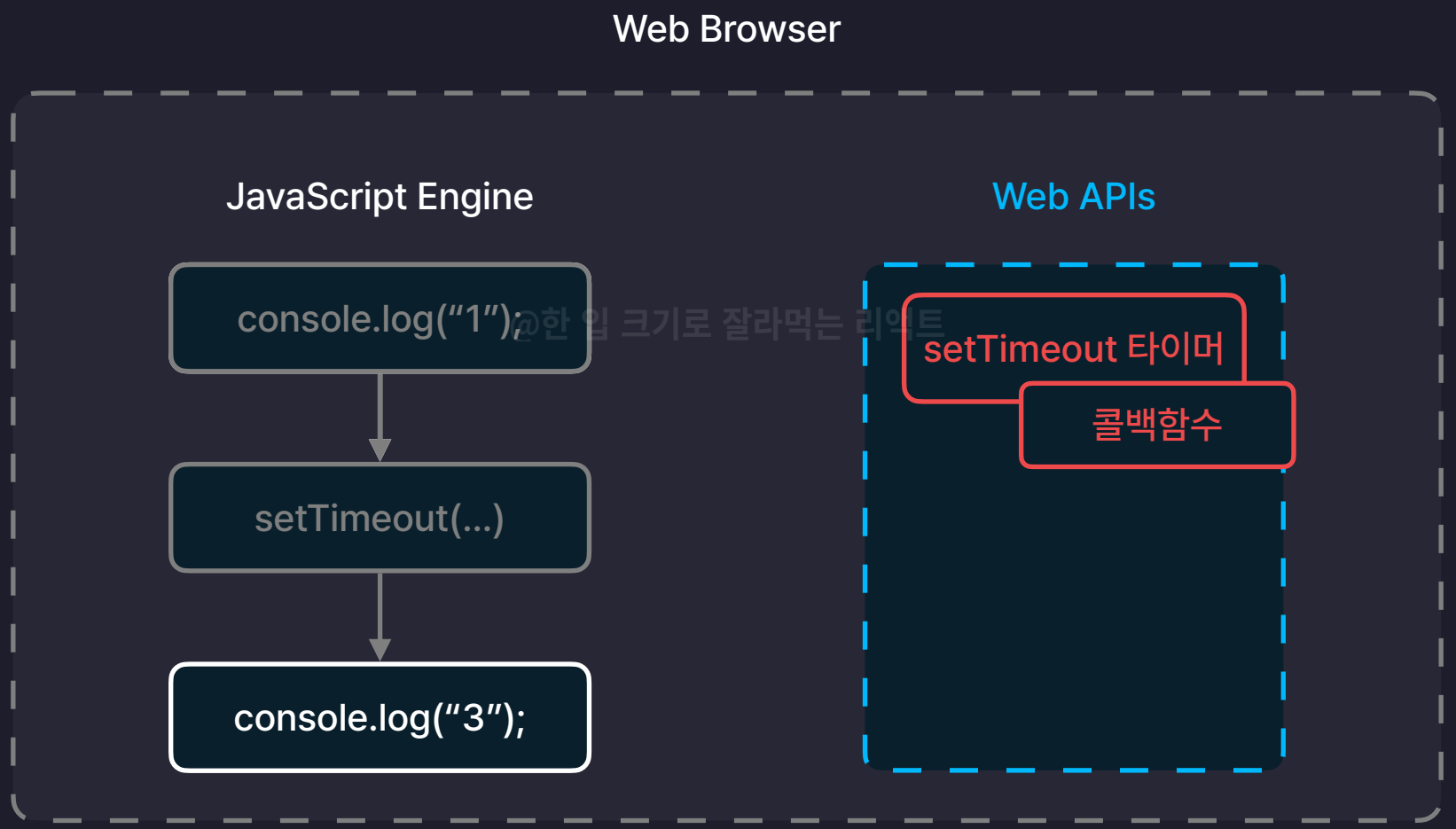
JavaScript의 비동기 처리



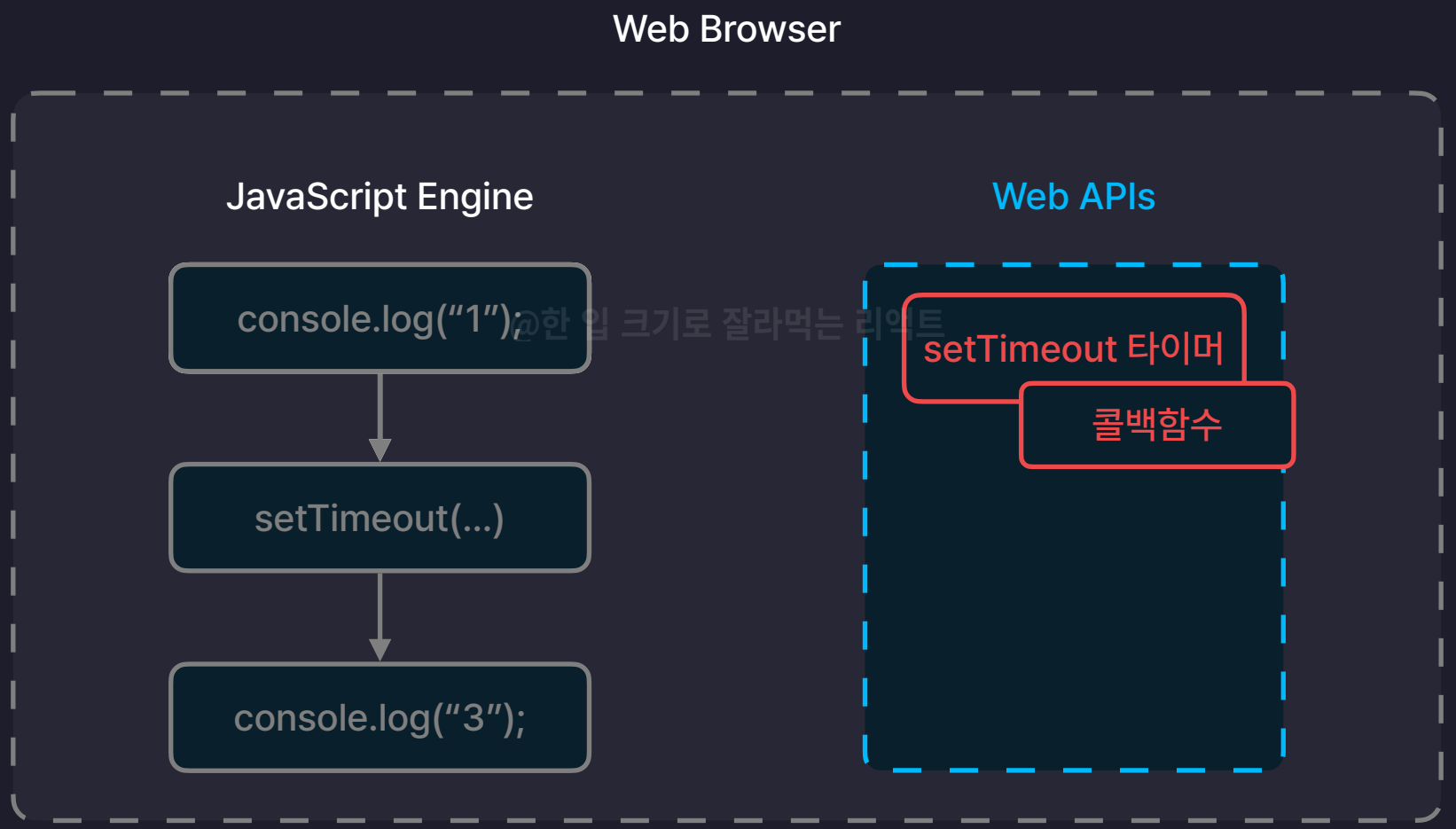
JavaScript의 비동기 처리



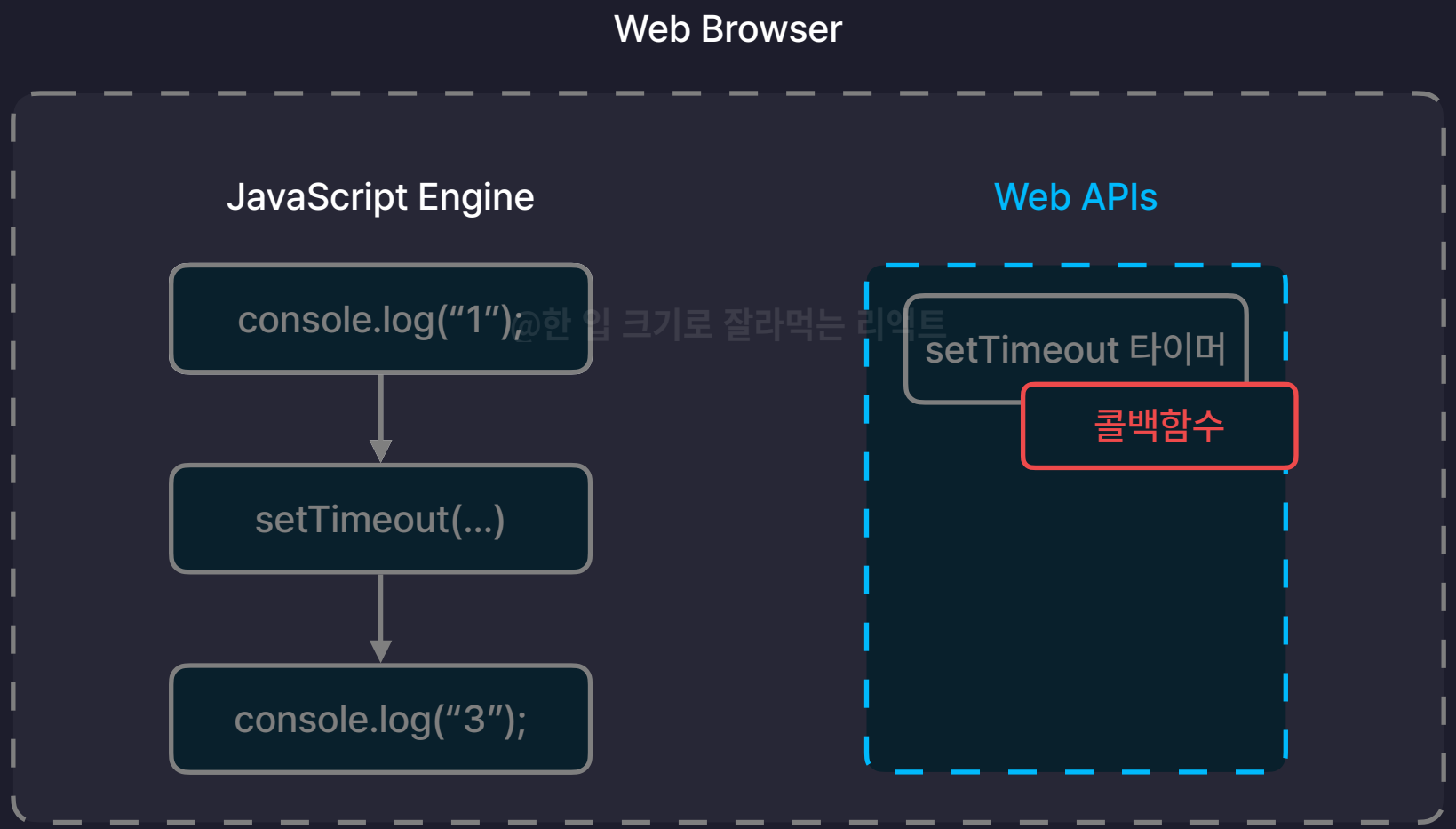
JavaScript의 비동기 처리



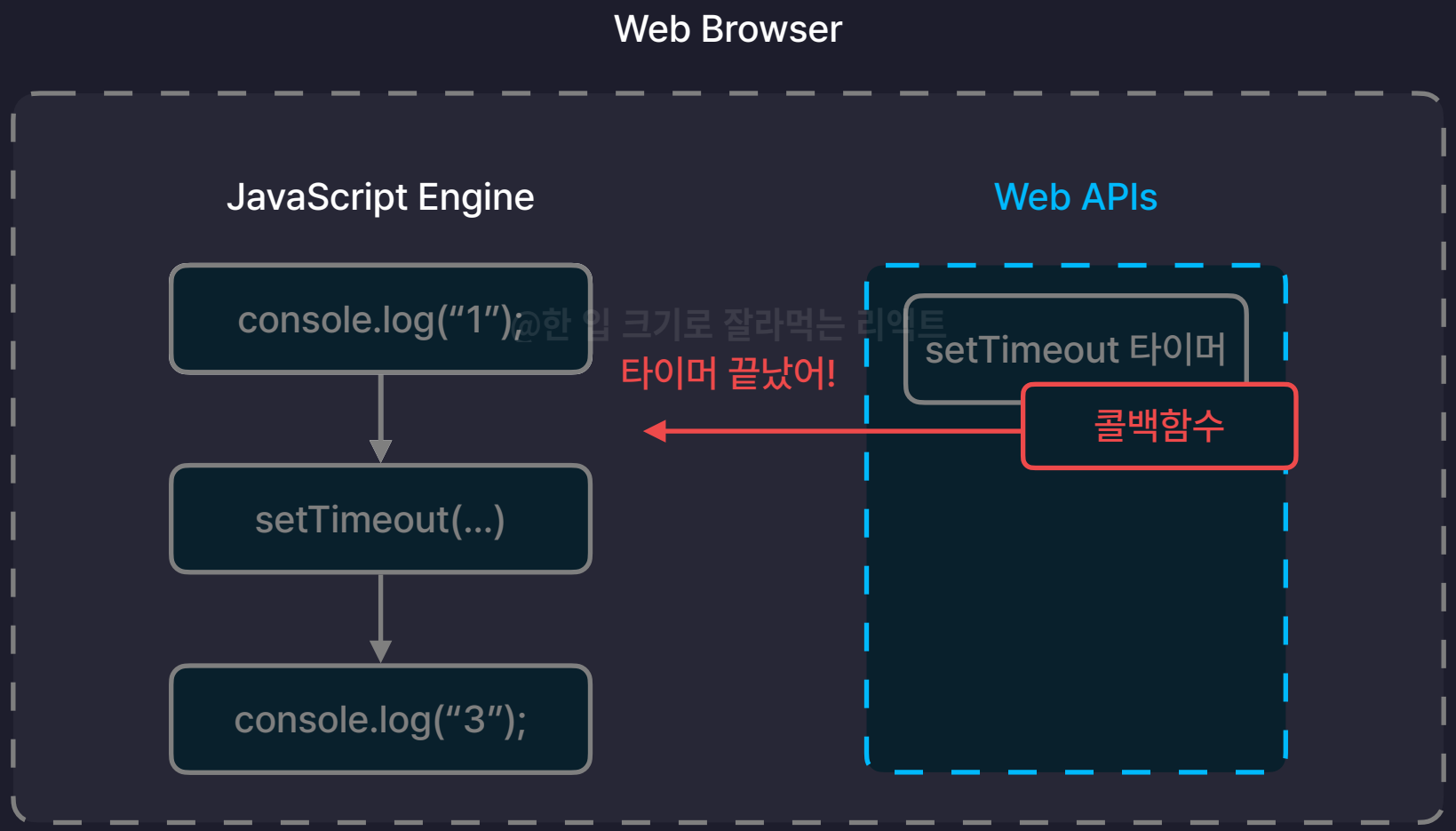
JavaScript의 비동기 처리



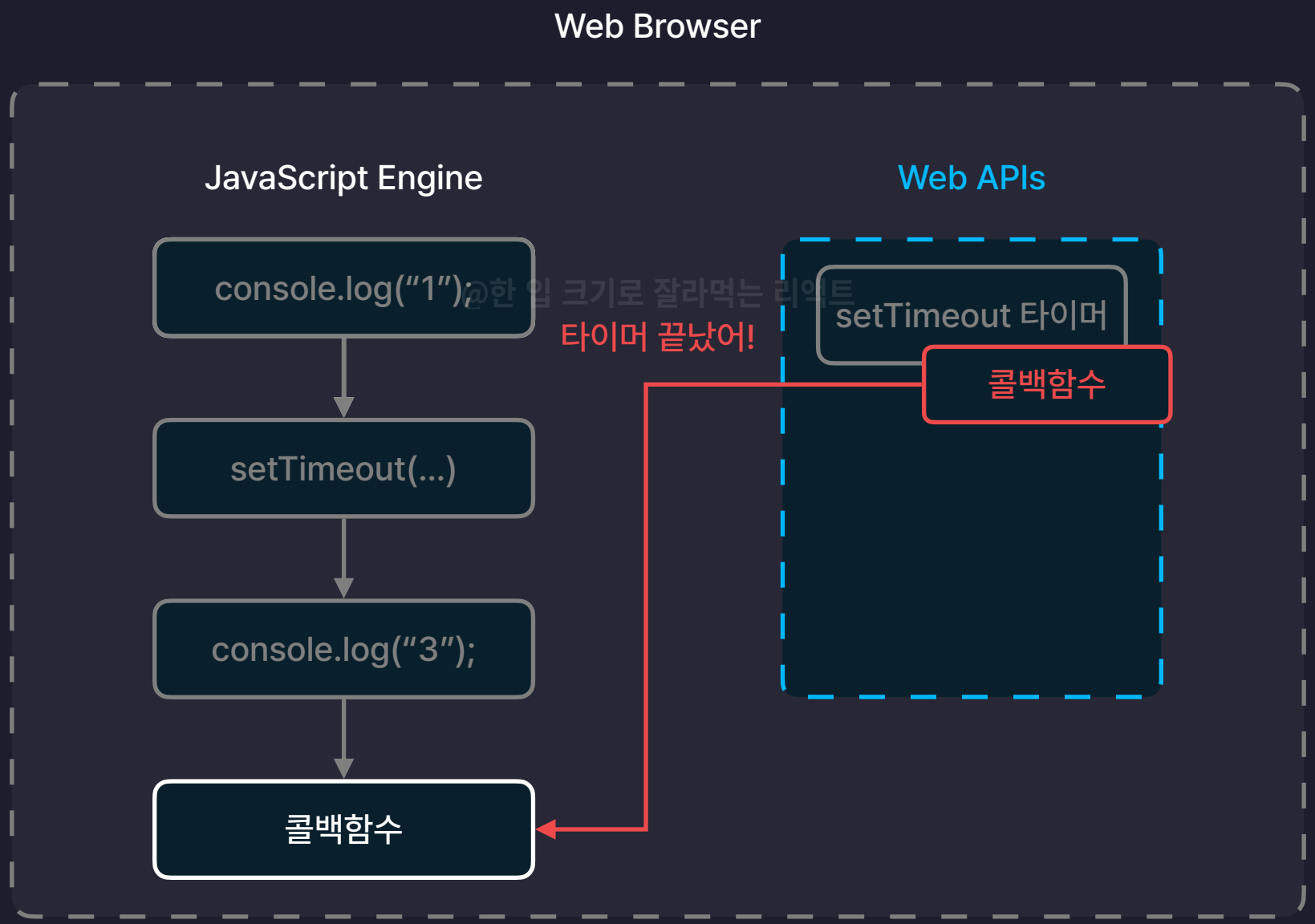
JavaScript의 비동기 처리



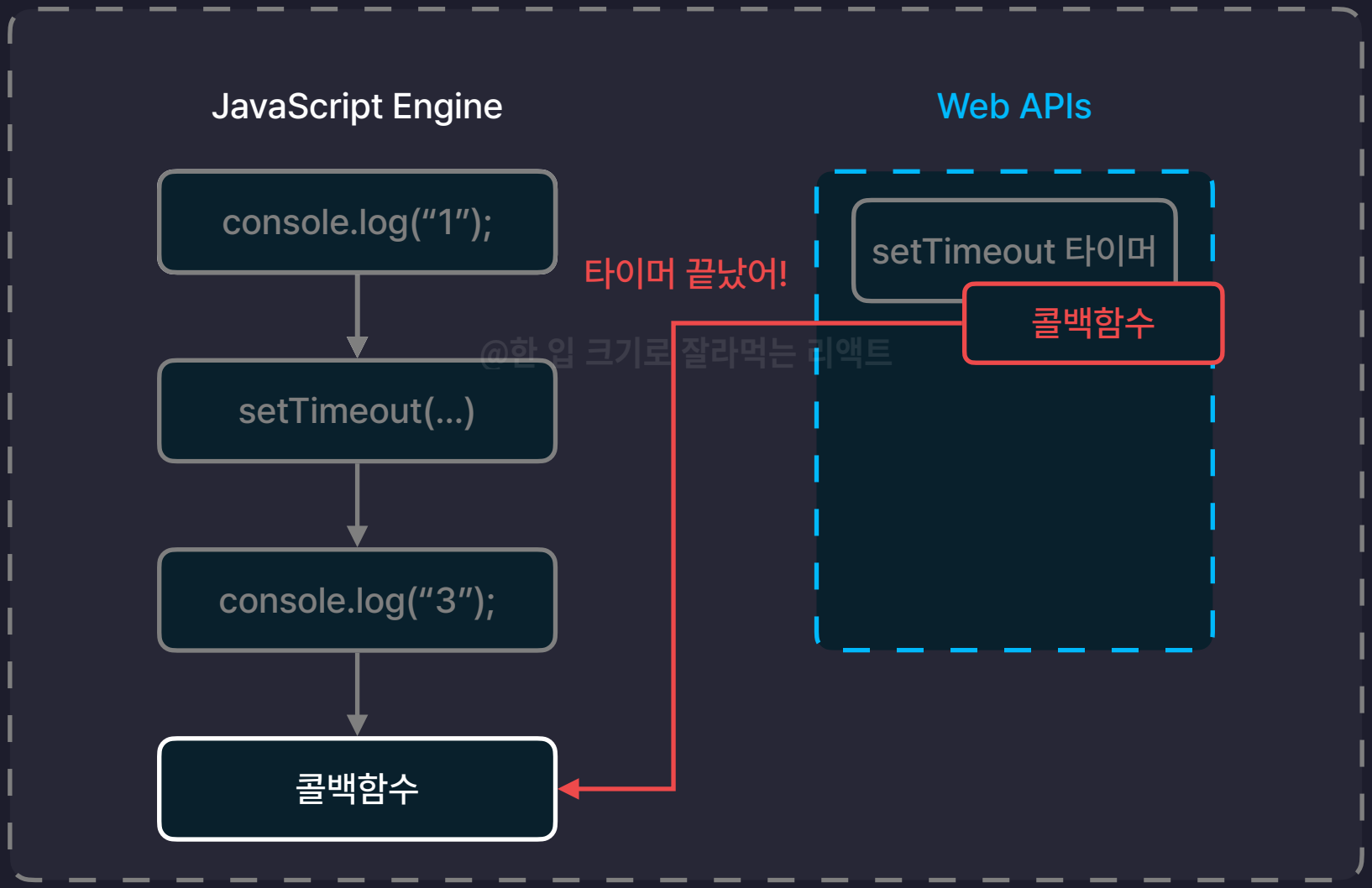
JavaScript의 비동기 처리



JavaScript의 비동기 처리



JavaScript의 비동기 처리



JavaScript의 비동기 처리

JavaScript 코드

```
setTimeout(() => {
  console.log("1");
}, 3000);

console.log("2");
```

실행 결과

3초 대기 후 실행

```
2
```

JavaScript의 비동기 처리

JavaScript 코드

```
setTimeout(() => {
  console.log("1");
}, 3000);

console.log("2");
```

JavaScript의 비동기 처리

JavaScript 코드

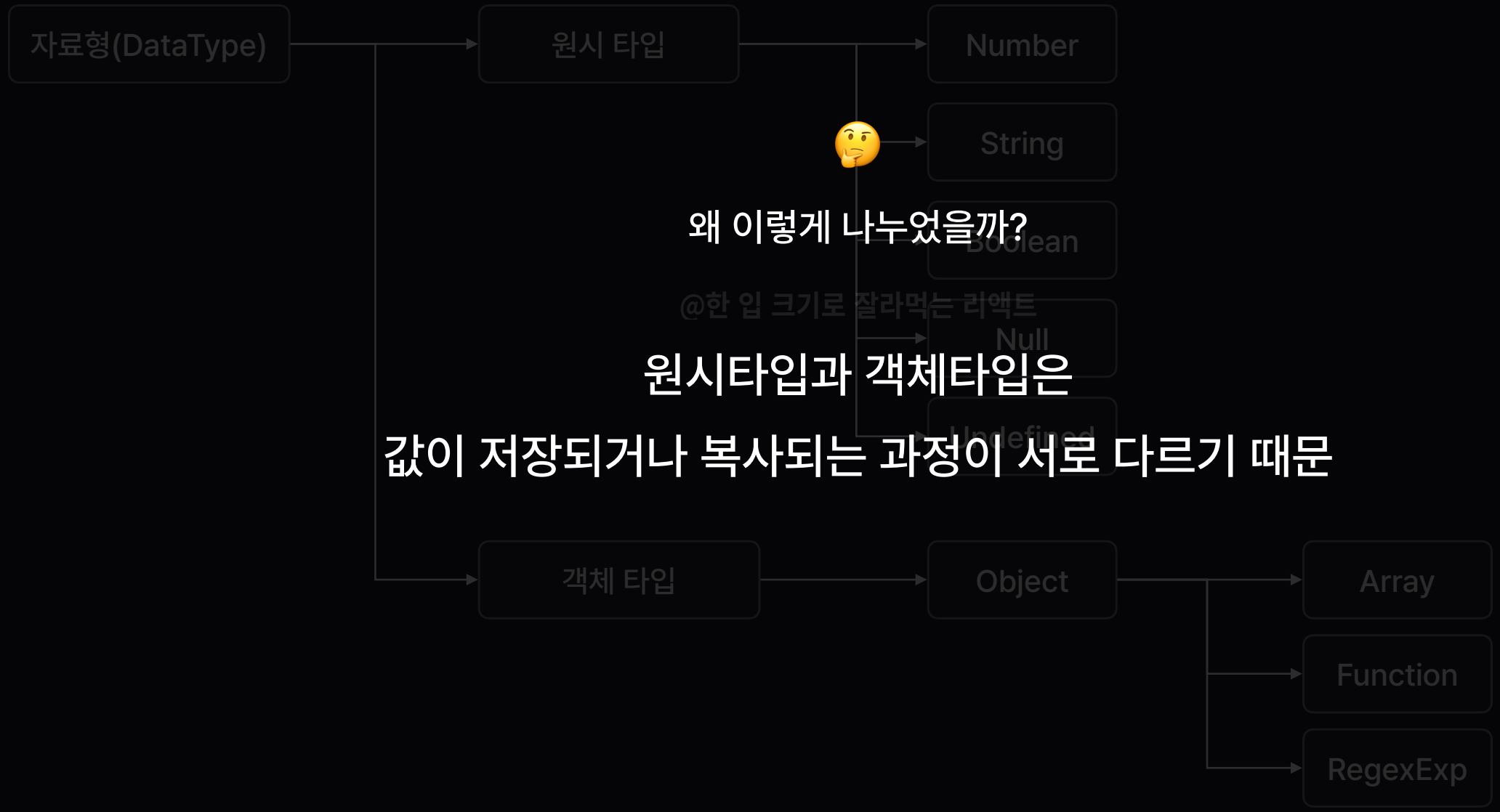
```
setTimeout(() => {
  console.log("1");
}, 3000);

console.log("2");
```


@한 입 크기로 잘라먹는 리액트

실행 결과

```
2
1
```

자바스크립트의 자료형(타입)



왜 이렇게 나누었을까?

원시타입과 객체타입은

값이 저장되거나 복사되는 과정이 서로 다르기 때문

한입 크기로 잘라먹는

비동기 작업 처리하기 2. Promise

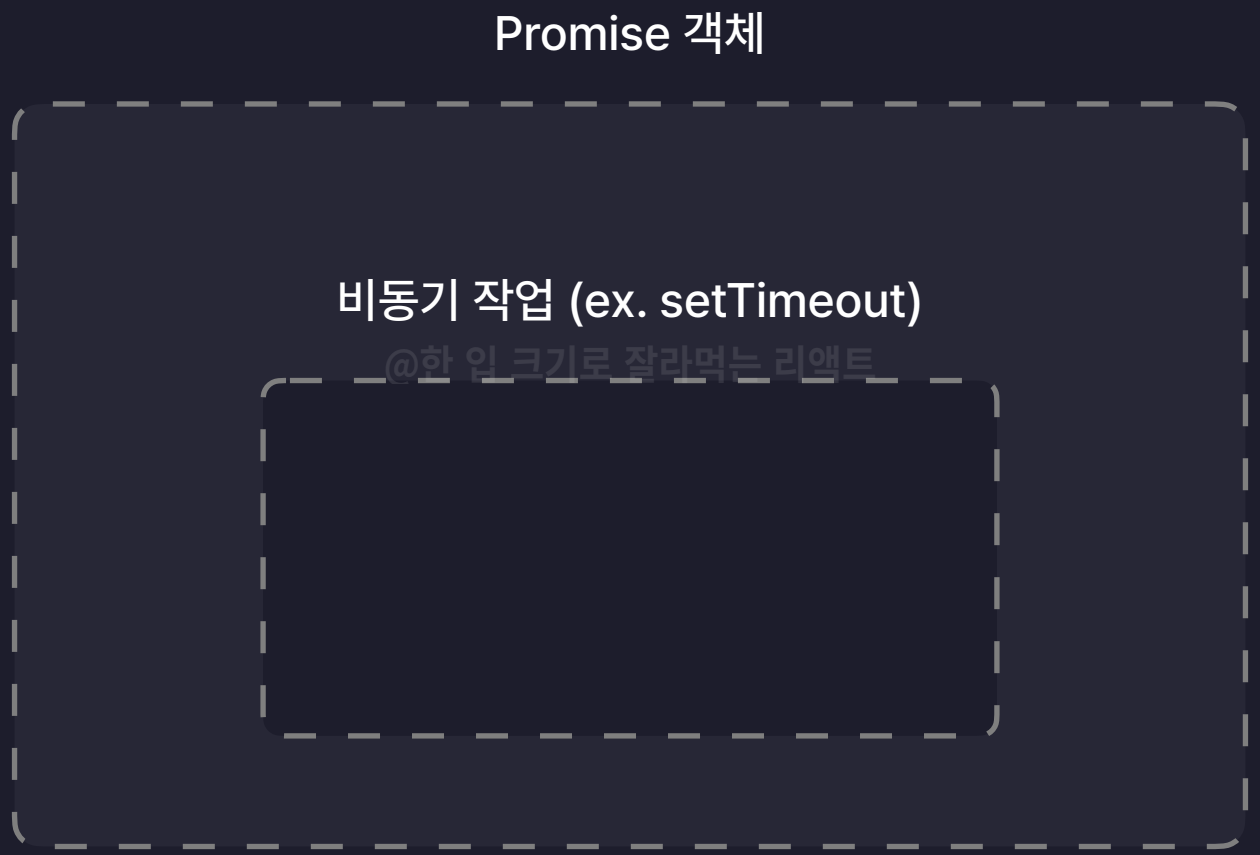
Promise 란?

비동기 작업을

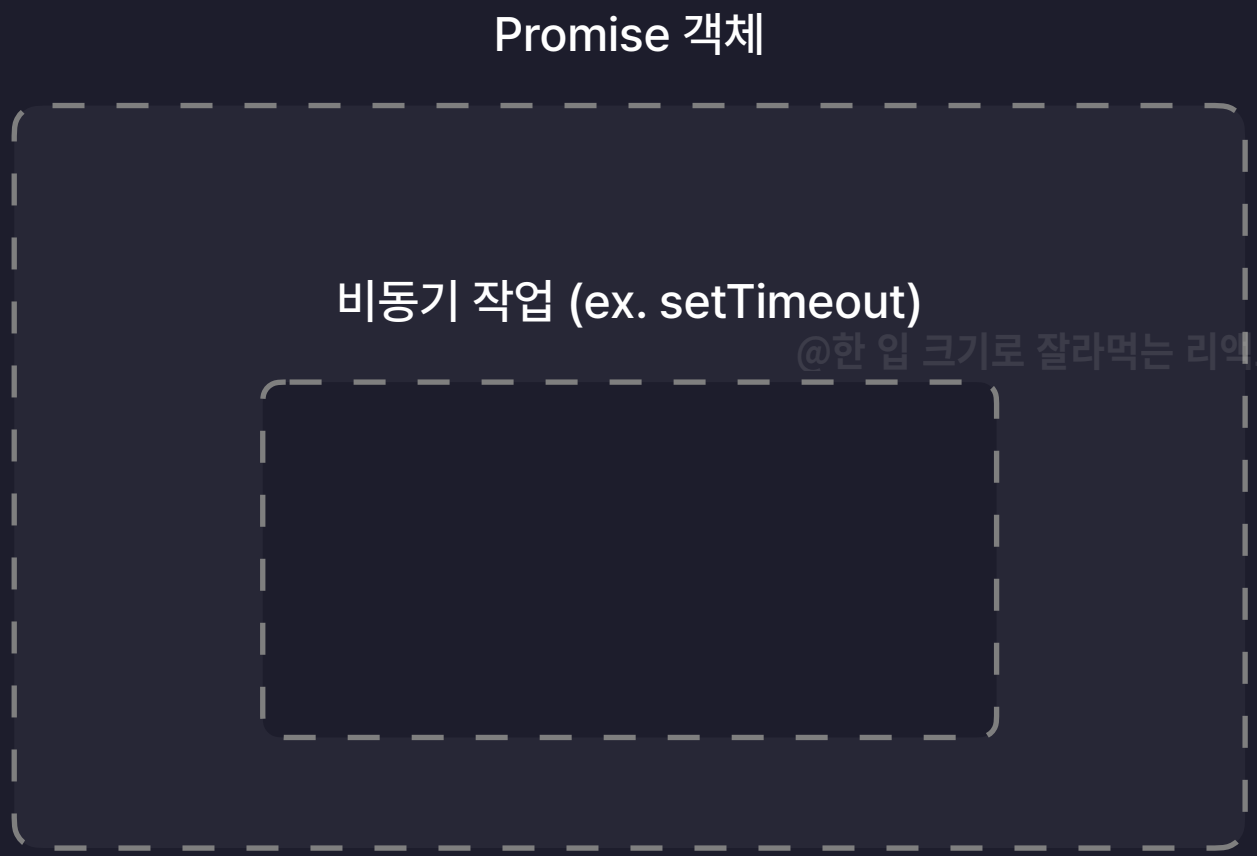
효율적으로 처리할 수 있도록 도와주는

자바스크립트의 내장 객체

Promise는 비동기 작업을 감싸는 객체이다



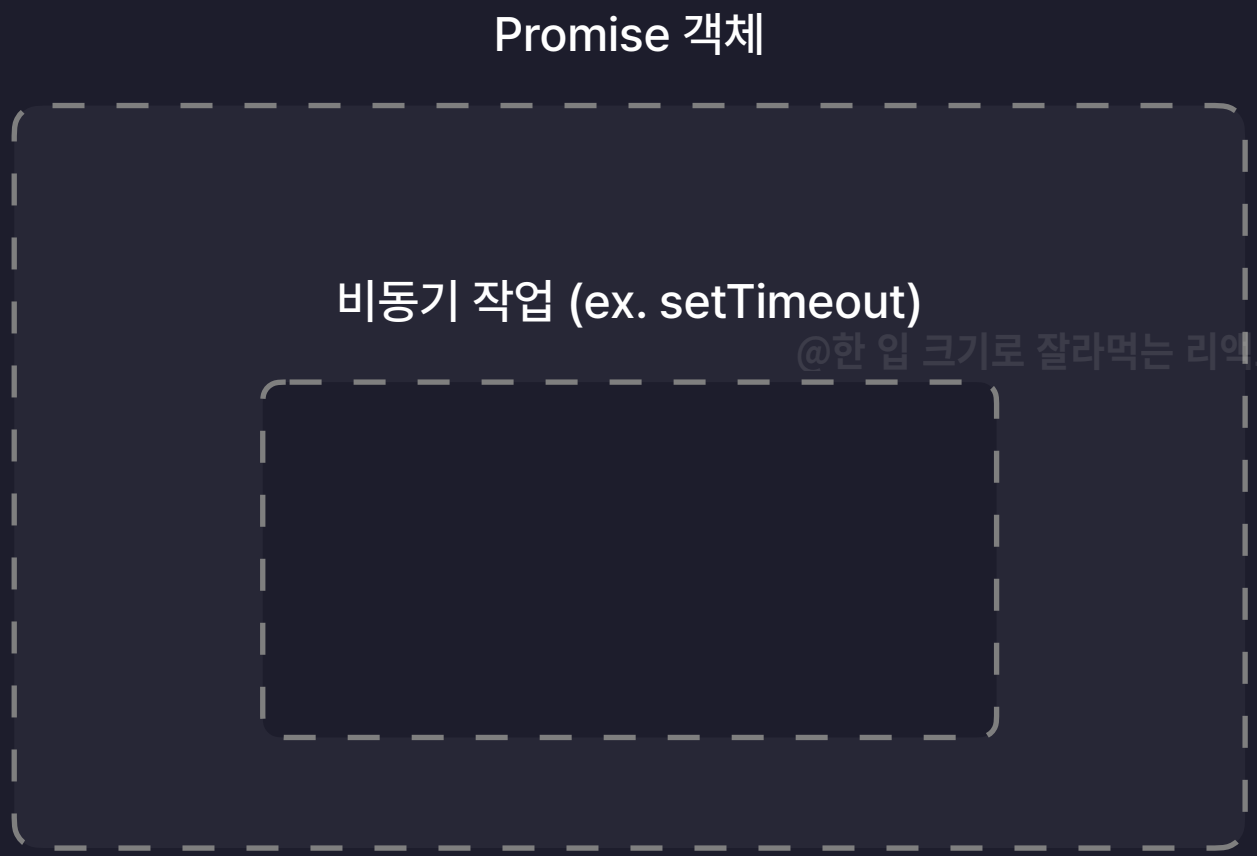
Promise는 비동기 작업을 감싸는 객체이다



Promise의 효능

- 비동기 작업 실행
- 비동기 작업 상태 관리
- 비동기 작업 결과 저장
- 비동기 작업 병렬 실행
- 비동기 작업 다시 실행
- 기타 등등 ...

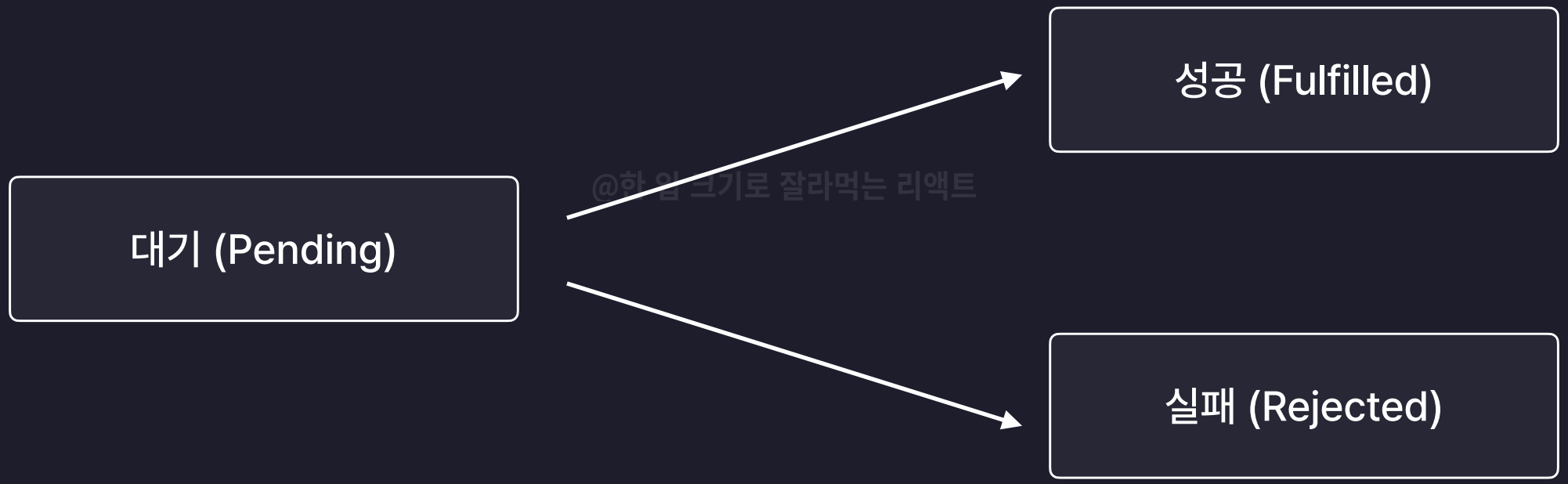
Promise는 비동기 작업을 감싸는 객체이다



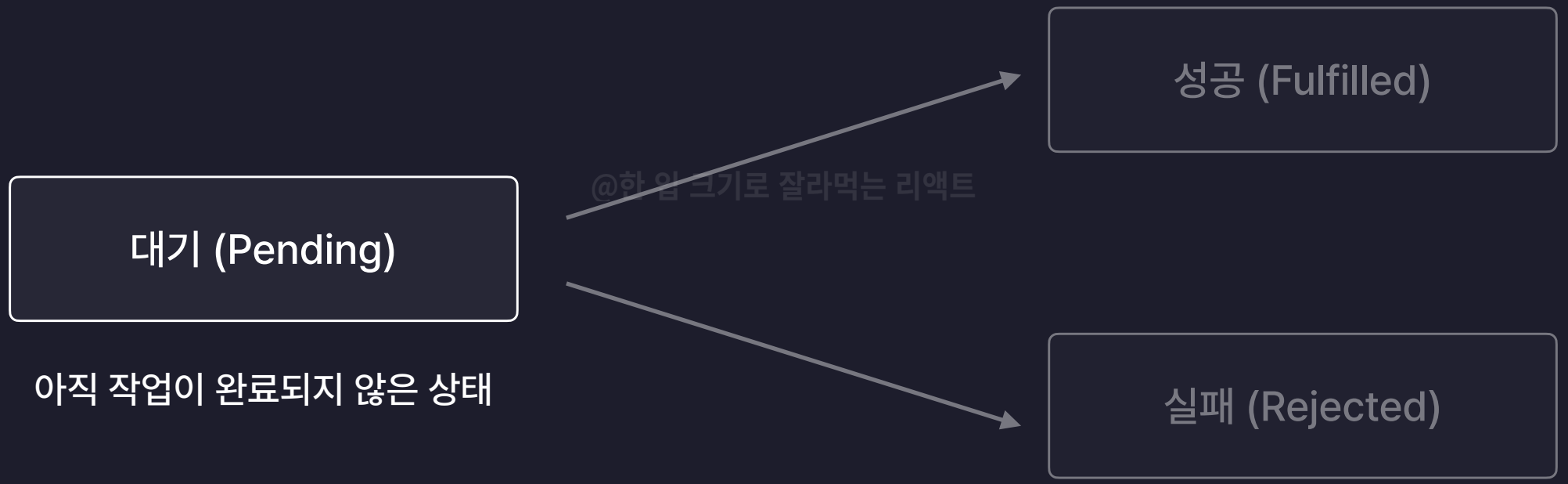
Promise의 효능

- 비동기 작업 실행
- 비동기 작업 상태 관리
- 비동기 작업 결과 저장
- 비동기 작업 병렬 실행
- 비동기 작업 다시 실행
- 기타 등등 ...

Promise의 3가지 상태

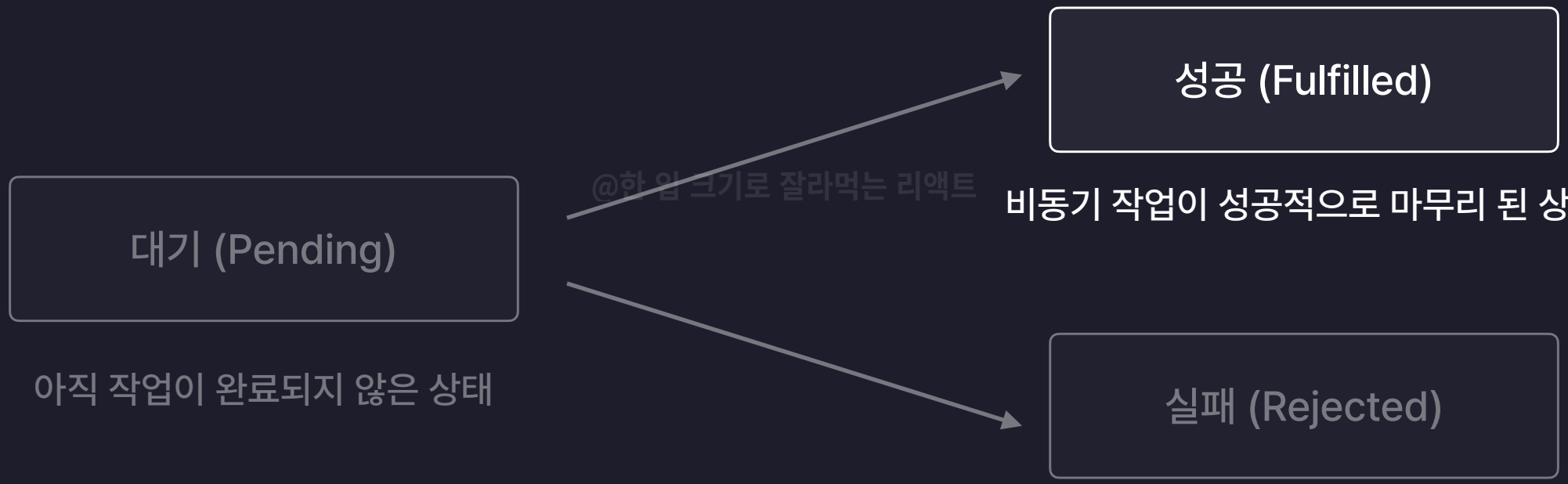


Promise의 3가지 상태

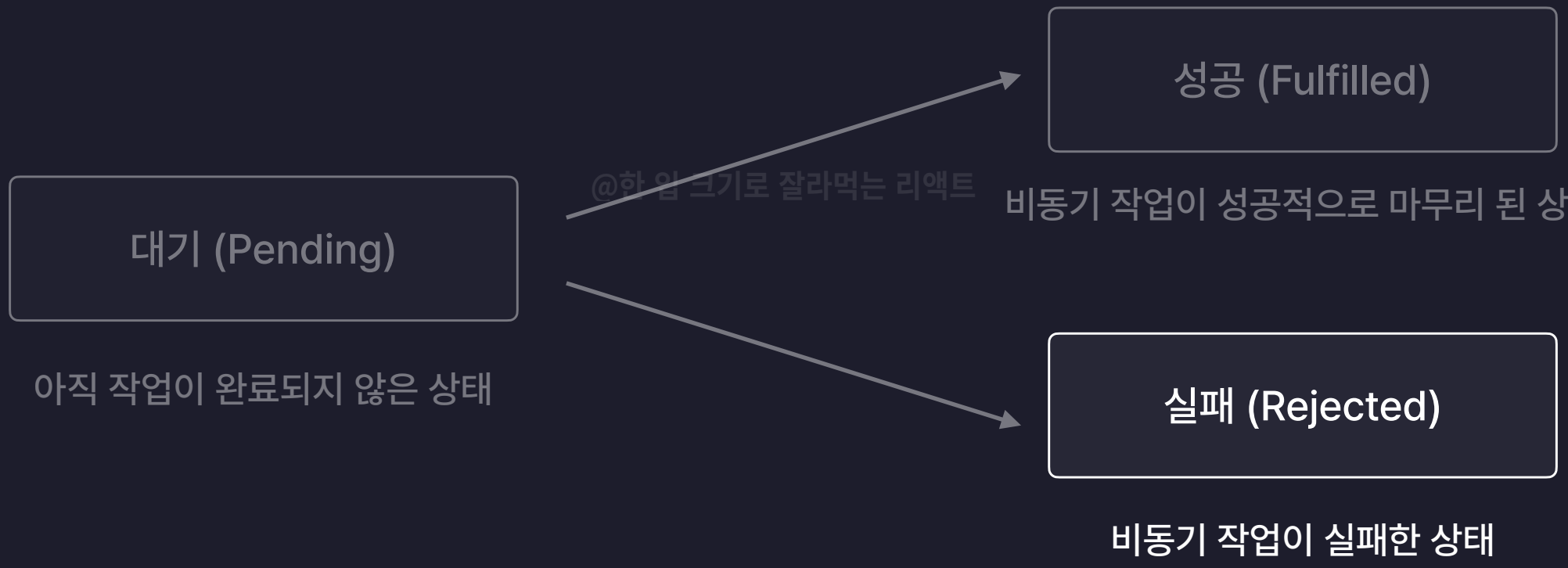


아직 작업이 완료되지 않은 상태

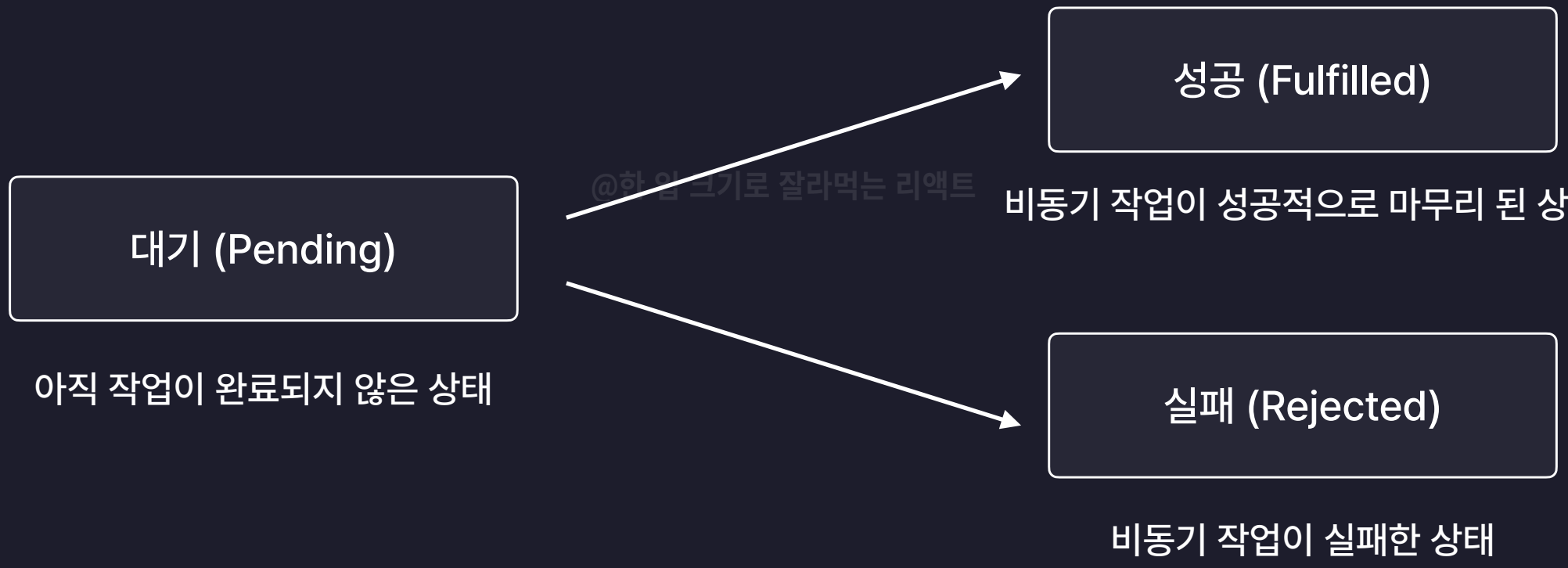
Promise의 3가지 상태



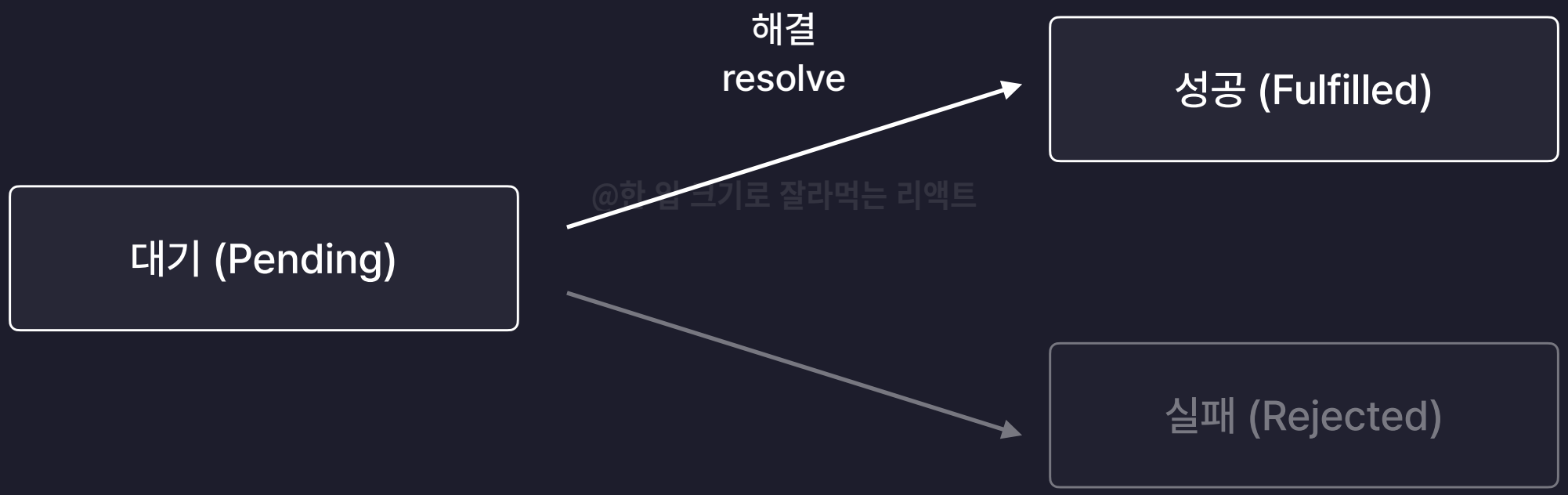
Promise의 3가지 상태



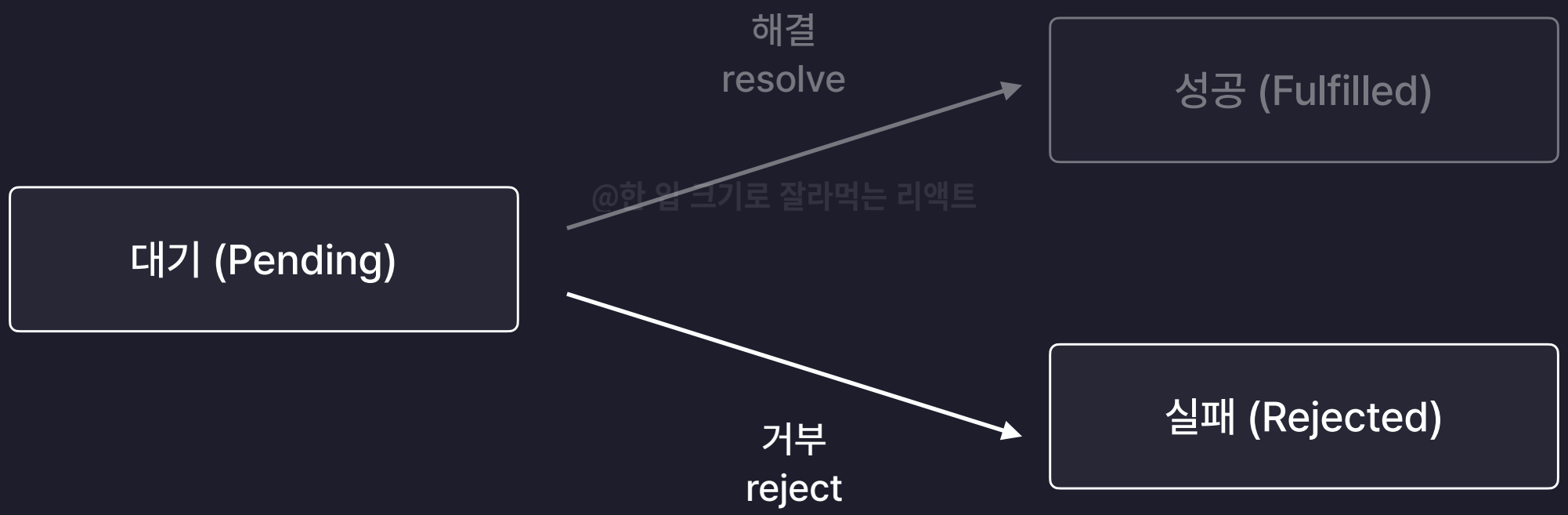
Promise의 3가지 상태



Promise의 3가지 상태



Promise의 3가지 상태



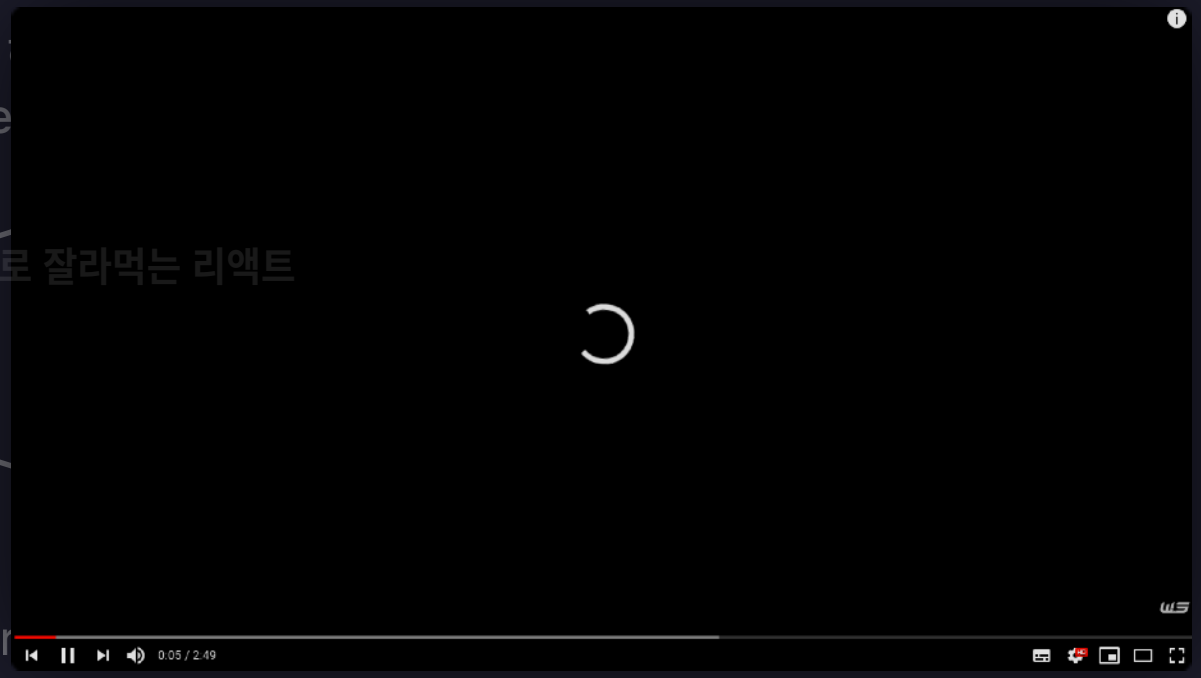
Promise의 3가지 상태

YouTube를 통해 살펴보는 Promise

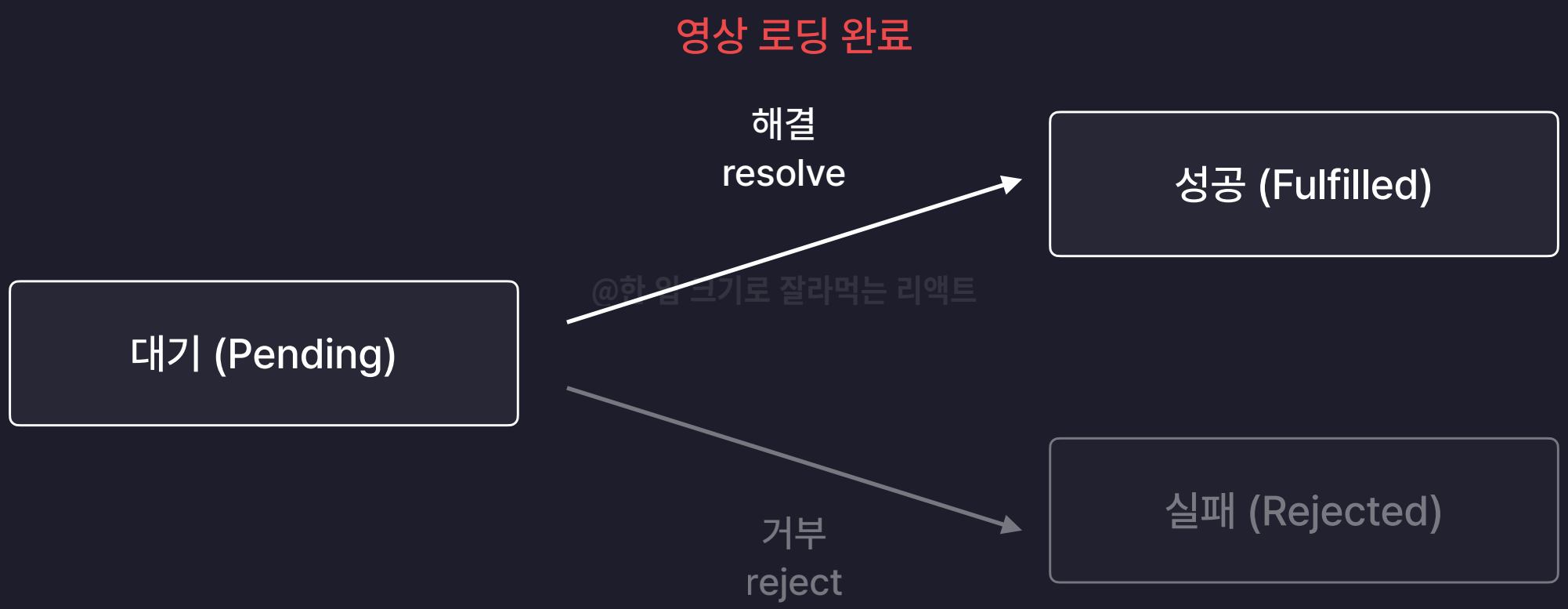
Promise의 3가지 상태

유튜브 영상 로딩 중

대기 (Pending)



Promise의 3가지 상태



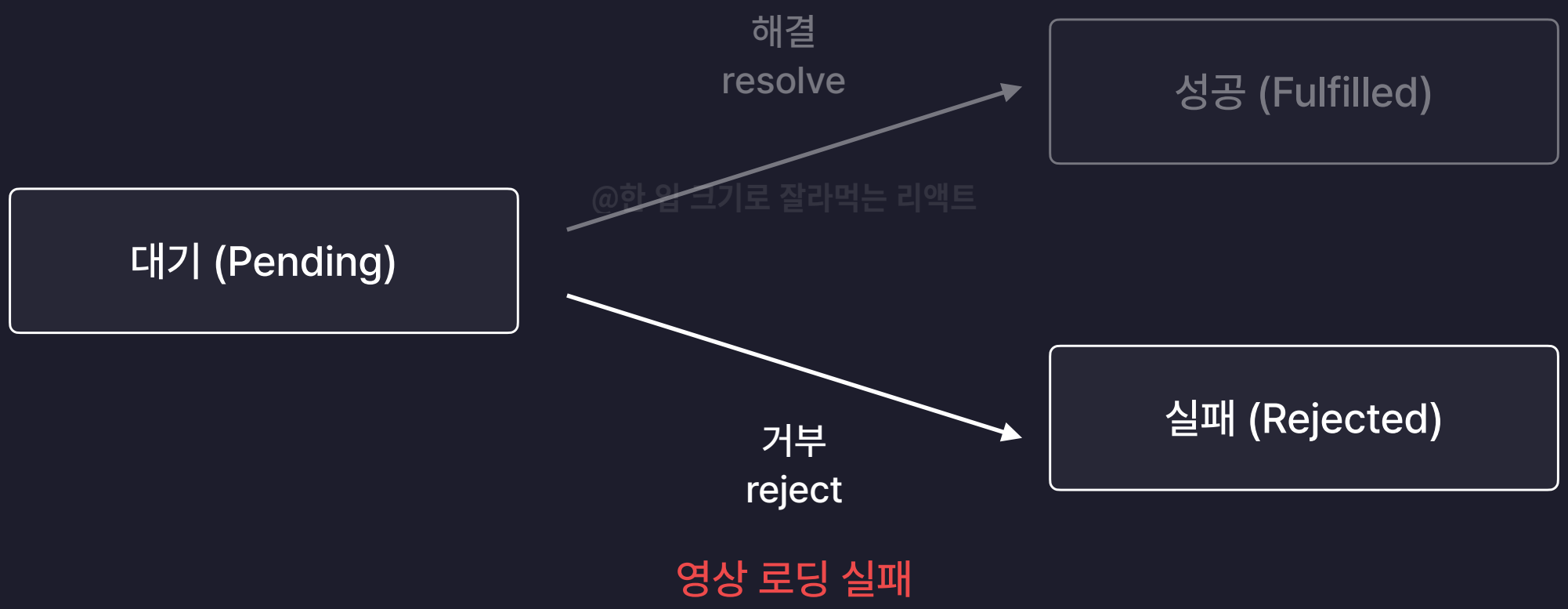
Promise의 3가지 상태

시청 가능

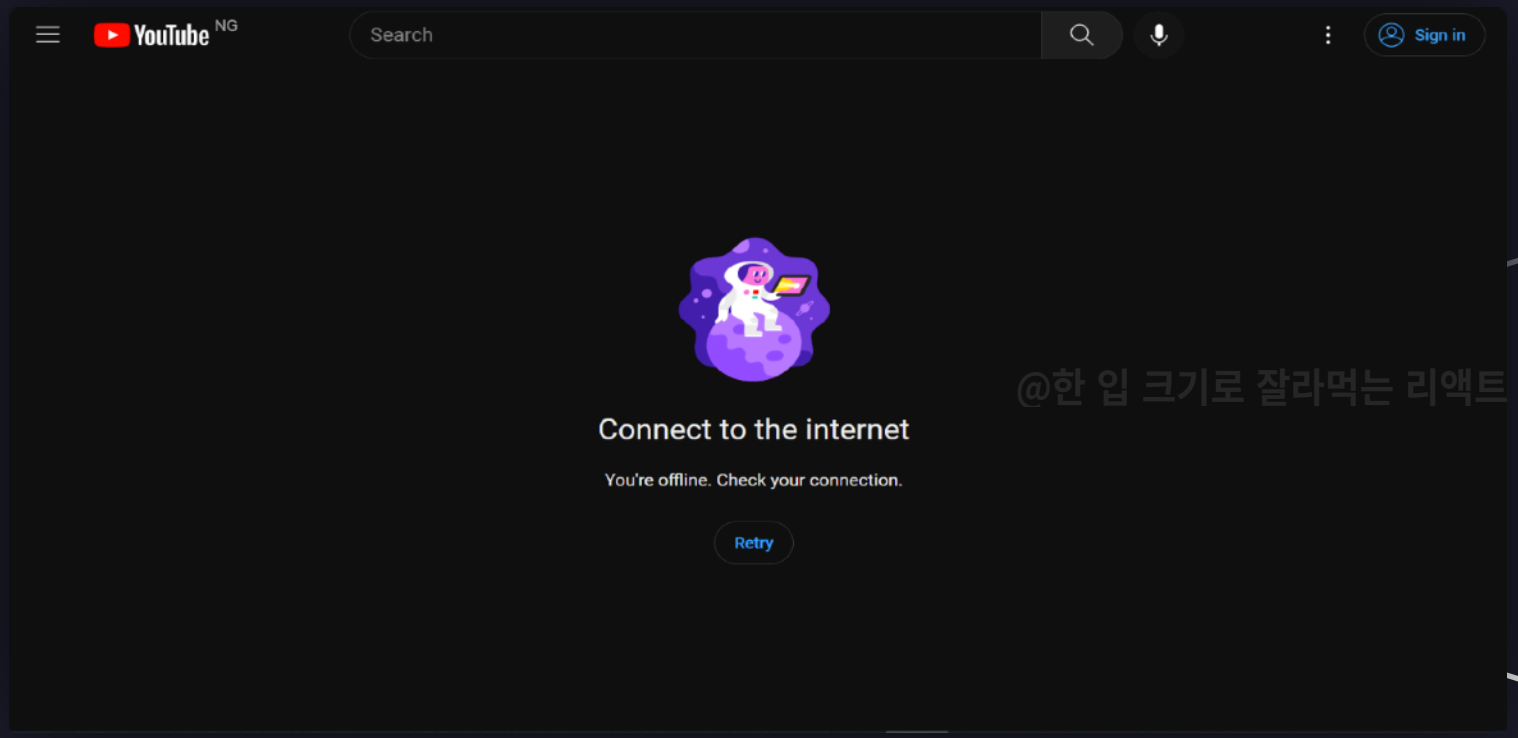
성공 (Fulfilled)

실패 (Rejected)

Promise의 3가지 상태



Promise의 3가지 상태



reject

성공 (Fulfilled)

시청 불가능 한 상태

실패 (Rejected)