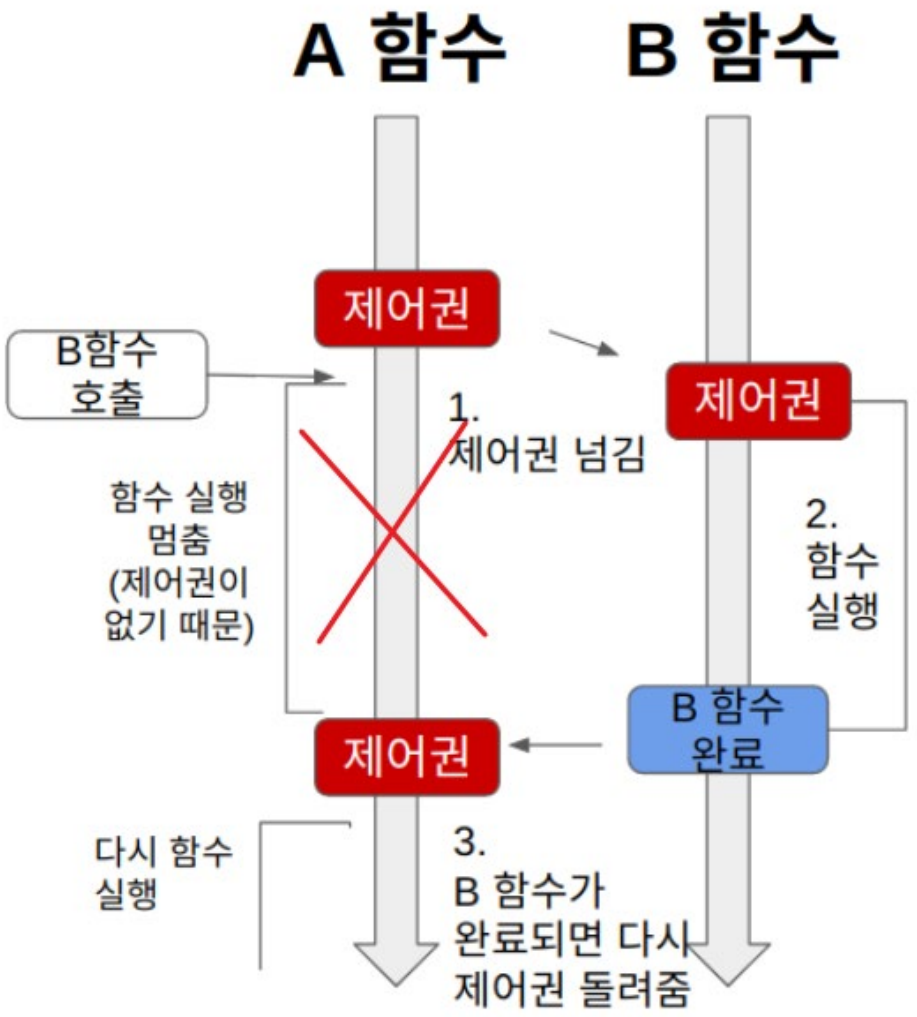
Synchronous vs Asynchronous
Blocking vs non-Blocking

1 Blocking VS Non-Blocking

블록킹 (Blocking)



Blocking : 자신의 작업을 진행하다가 다른 주체의 작업이 시작되면 다른 작업이 끝날 때까지 기다렸다가 자신의 작업을 시작하는 것

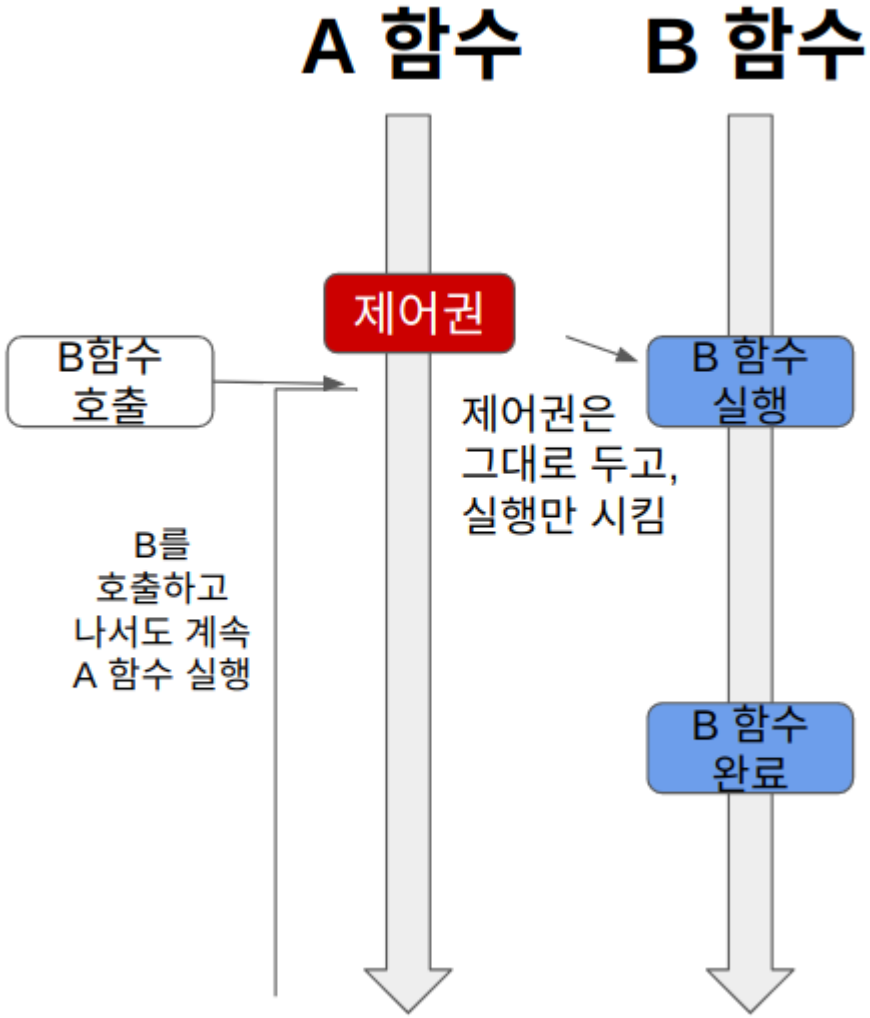
1. A함수가 B함수를 호출하면 B에게 제어권을 넘긴다.

2. **제어권**을 넘겨받은 B는 열심히 함수를 실행한다. A는 B에게 **제어권**을 넘겨주었기 때문에 함수 실행을 잠시 멈춘다.

3. B함수는 실행이 끝나면 자신을 호출한 A에게 **제어권**을 돌려준다.

1 Blocking VS Non-Blocking

논블록킹 (Non-Blocking)



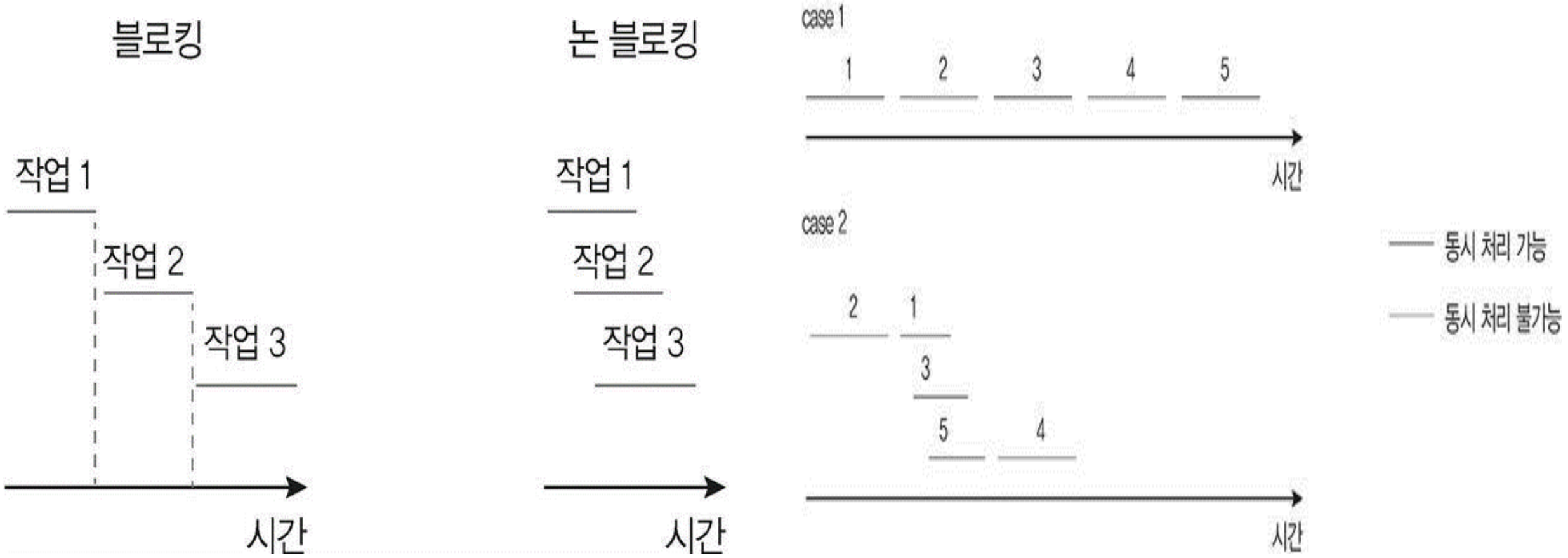
Non-Blocking: 다른 주체의 작업에 관련없이 자신의 작업을 하는 것

1.A함수가 B함수를 호출하면, B 함수는 실행되지만, **제어권은 A 함수가 그대로 가지고 있는다.**

2.A함수는 계속 **제어권**을 가지고 있기 때문에 B함수를 호출한 이후에도 자신의 코드를 계속 실행한다.

1 Blocking VS Non-Blocking

Blocking and Non-Blocking의 시간적 이점



1 Bloking VS Non-Bloking

예시

```
const longRunningTask = () => {
  // 오래 걸리는 작업
  console.log('작업 끝');
}

console.log('시작');
longRunningTask();
console.log('다음 작업');
```

```
시작
작업 끝
다음 작업
```

```
const longRunningTask = () => {
  // 오래 걸리는 작업
  console.log('작업 끝');
}

console.log('시작');
setTimeout(longRunningTask, 0);
console.log('다음 작업');
```

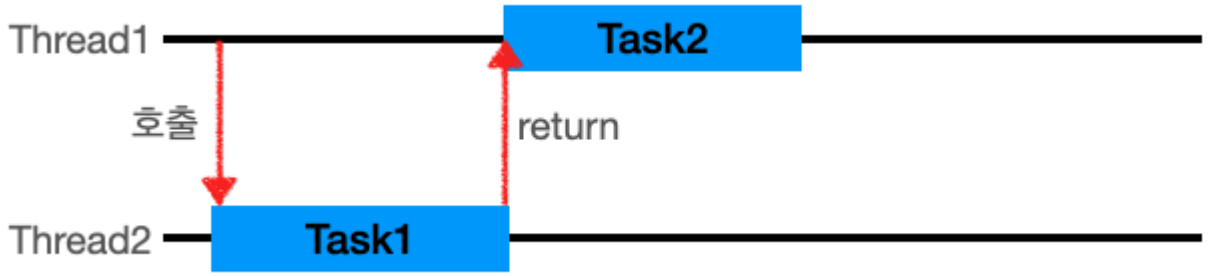
```
시작
다음 작업
작업 끝
```

2 Synchronous VS Asynchronous

동기 (Synchronous)

Synchronous: 번역을 해보면 동기라는 뜻을 가진다. 작업을 동시에 수행하거나, 동시에 끝나거나, 동시에 시작함을 의미함

- 작업 요청을 했을 때 요청의 **결과값(return)**을 직접 받는 것이다.
- 요청의 결과값이 **return**값과 동일하다.
- **호출한 함수가 작업 완료를 신경 쓴다.**
- **모든 요청, 응답이 일련의 순서를 따른다**

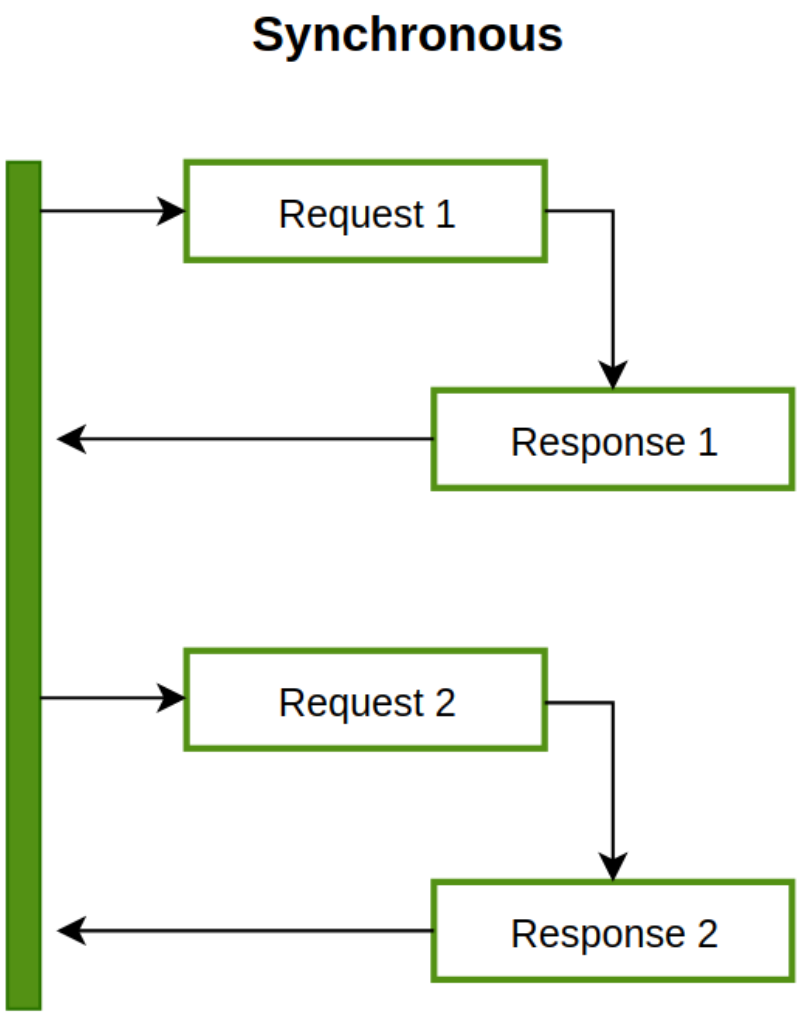


Thread1이 작업을 시작 시키고, **Task1**이 끝날때까지 기다렸다 **Task2**를 시작한다.

A 함수가 B 함수를 호출 할 때, B 함수의 결과를 A 함수가 처리하는 것.

2 Synchronous VS Asynchronous

동기 (Synchronous)



호출하는 함수 A가 **호출되는 함수 B**의 작업 완료 후 리턴을 기다리거나, 바로 리턴 받더라도 미완료 상태이라면 작업 완료 여부를 스스로 계속 확인하며 **신경 씀**

(* 함수 A가 함수 B를 호출한 뒤, **함수 B의 리턴 값을 계속 확인하면서 신경 쓰는 것**)

2 Synchronous VS Asynchronous

비동기 (Asynchronous)

Asynchronous: 번역을 해보면 비동기라는 뜻을 가진다. 시작,종료가 일치하지 않으며, 끝나지않음을 의미함

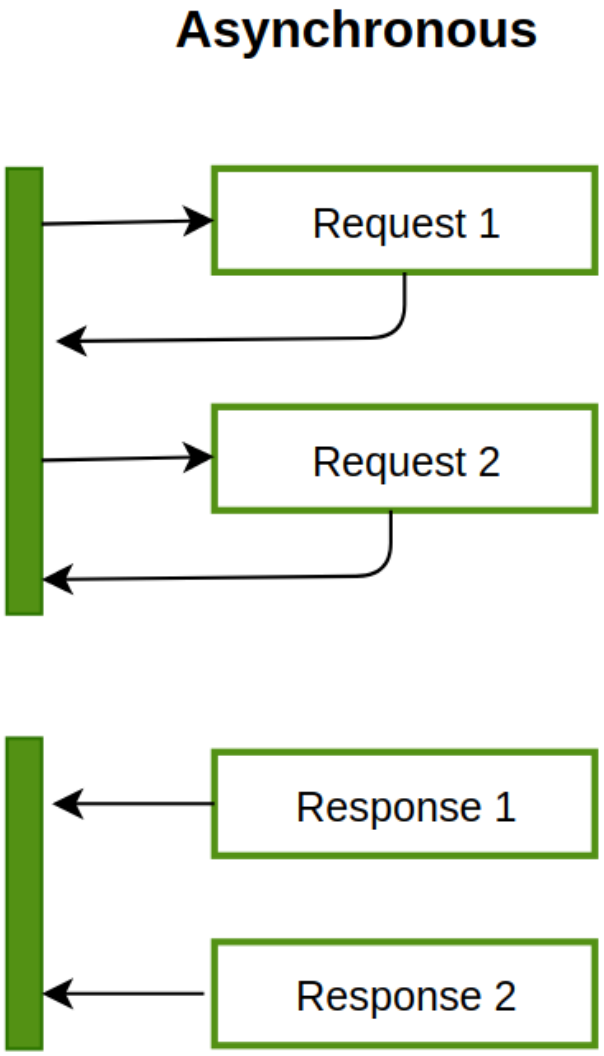
- 작업 요청을 했을 때 요청의 **결과값(return)을 간접적으로 받는 것**이다.
- 요청의 결과값이 return값과 다를 수 있다.
- 해당 요청 작업은 별도의 스레드에서 실행하게 된다.
- **콜백**을 통한 처리가 비동기 처리라고 할 수 있다.
- **호출된 함수(callback 함수)가 작업 완료를 신경 쓴다.**
- 요청을 보냈을 때 응답 상태와 상관없이 다음 동작을 수행한다. 따라서 작업의 순서가 보장되지 않는다.

Thread1이 작업을 시작 시키고, 완료를 기다리지 않고, **Thread1**은 다른 일을 처리할 수 있다.

A 함수가 B 함수를 호출 할 때, B 함수의 결과를 B 함수가 처리하는 것. (callback)

2 Synchronous VS Asynchronous

비동기 (Asynchronous)



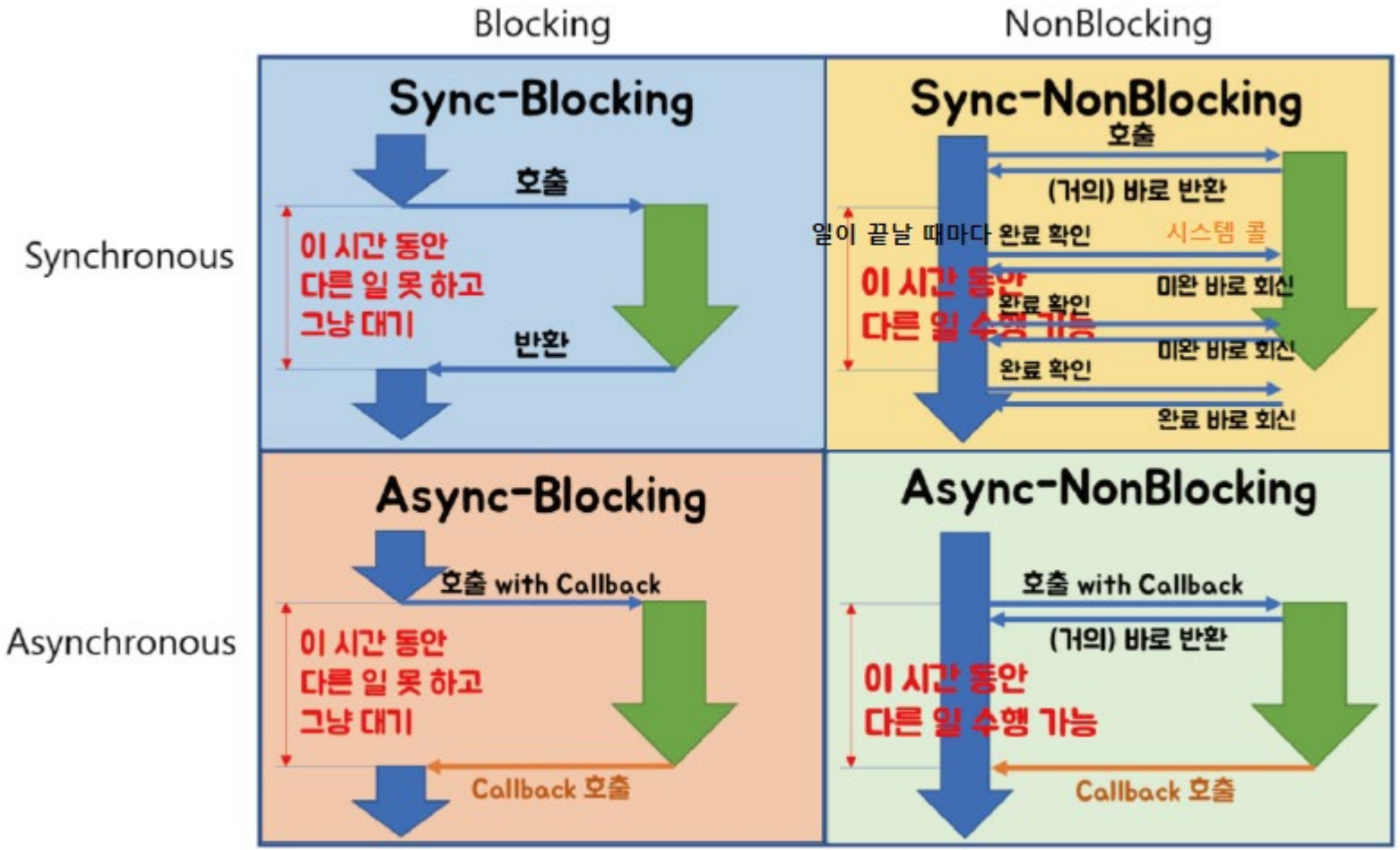
함수 A가 **함수 B**를 호출할 때 **콜백 함수를 함께 전달**해서, 함수 B의 작업이 완료되면 함께 보낸 콜백 함수를 실행한다.

함수 A는 함수 B를 호출한 후로 **함수 B의 작업 완료 여부에는 신경 쓰지 않는다.**

3 동기/비동기, Blocking/Non-Blocking의 차이는?

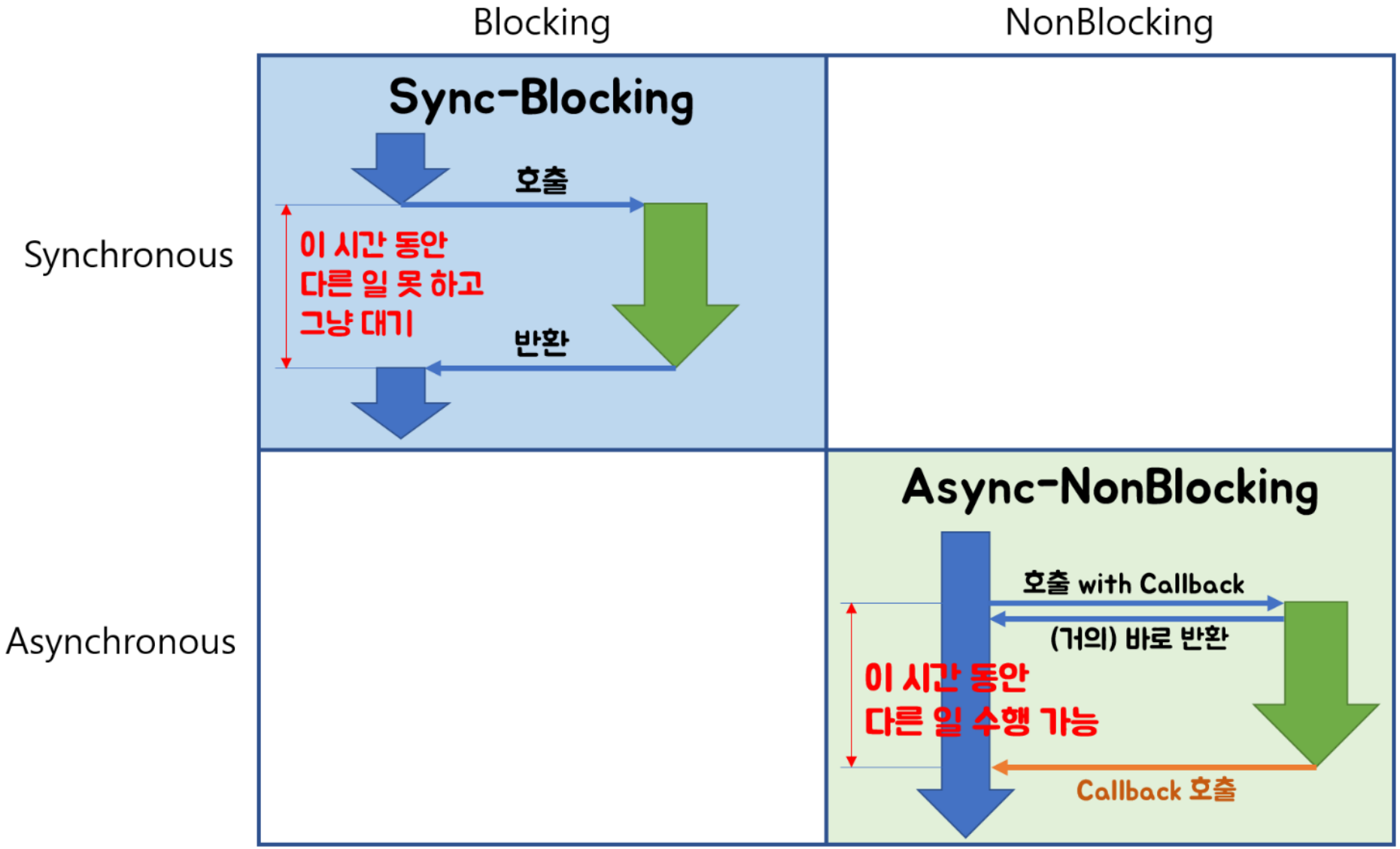
Blocking/Non-Blocking	호출되는 함수가 바로 **return**하느냐 마느냐가 관심사(함수가 바로 **return**되는지 여부)
Blocking	호출된 함수가 **자신의 작업을 모두 마칠 때까지** 호출한 함수 에게 **제어권을 넘겨주지 않고 대기**
Non-Blocking	호출된 함수가 **바로 return해서 호출한 함수에게 제어권을 넘겨주고 호출한 함수가 다른 일을 할 수 있는 기회**를 줄 수 있으면 Non Blocking
동기/비동기	호출되는 함수의 작업 완료 여부를 누가 신경쓰느냐가 관심사이다. (백그라운드 작업 완료 확인 여부)
동기	호출하는 함수가 호출되는 함수의 작업 완료 후 **return을 기다리거나** 호출되는 함수로부터 바로 return 받더라도 **작업 완료 여부를 호출한 함수 스스로 확인**
비동기	호출되는 함수에게 **callback**을 전달해서 호출되는 함수의 작업이 완료되면 호출되는 함수가 전달받은 callback을 실행하고, **호출한 함수는 작업 완료 여부를 신경쓰지 않는다**

4 동기/비동기, blocking/non-blocking 조합



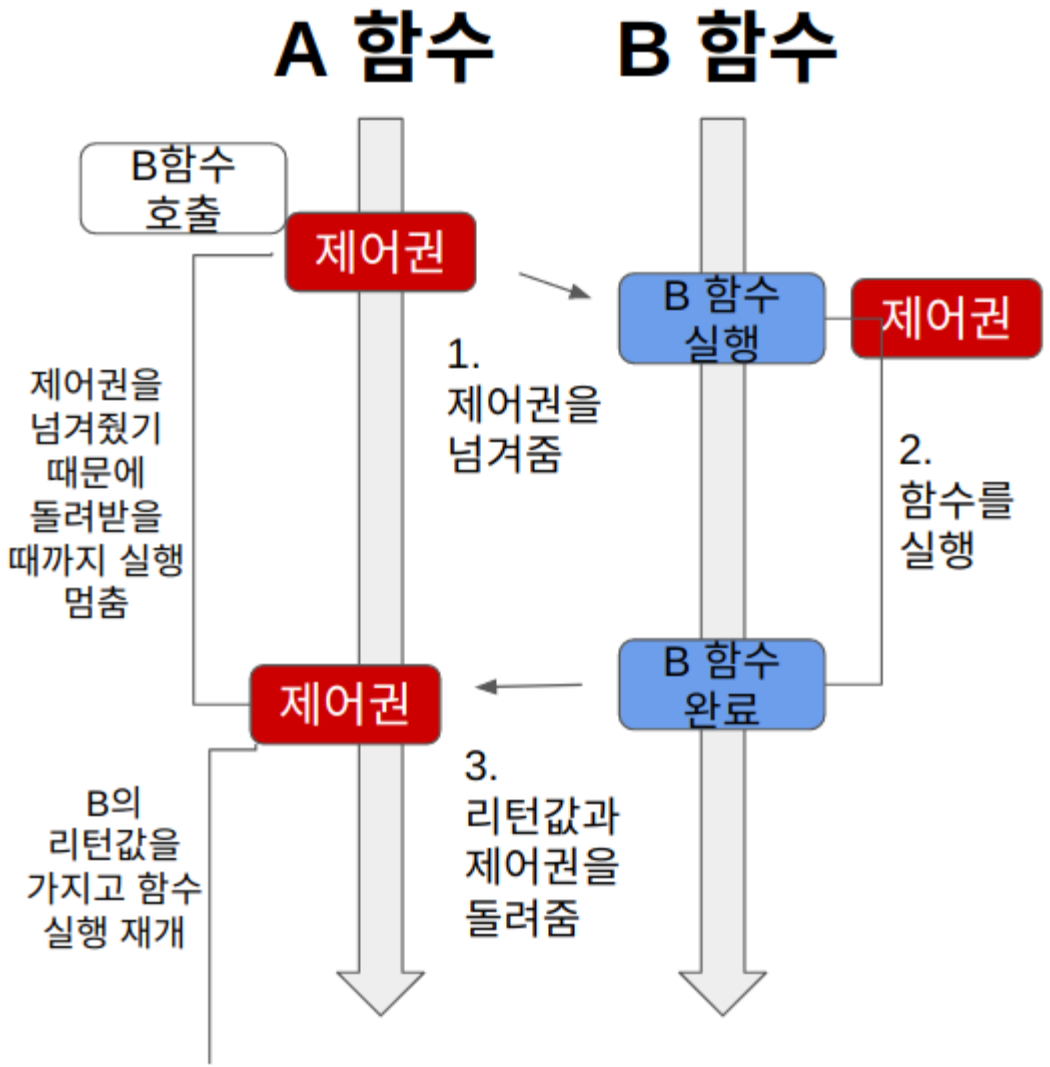
4 동기/비동기, blocking/non-blocking 조합

blocking + Synchronous, non-blocking + Asynchronous



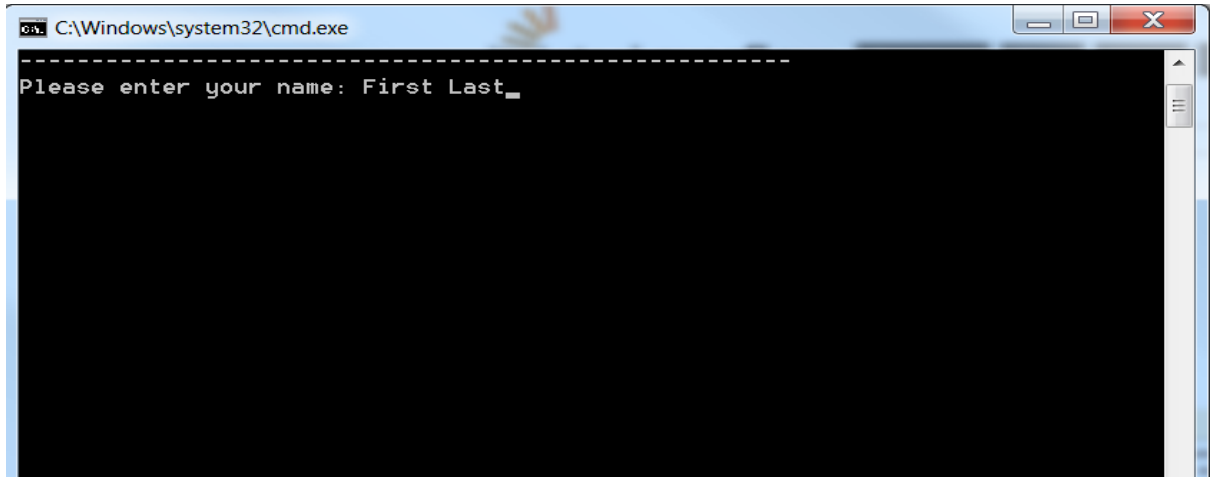
4 동기/비동기, blocking/non-blocking 조합

Sync-Blocking



함수 A는 함수 B의 리턴값을 필요로 한다 (**동기**)

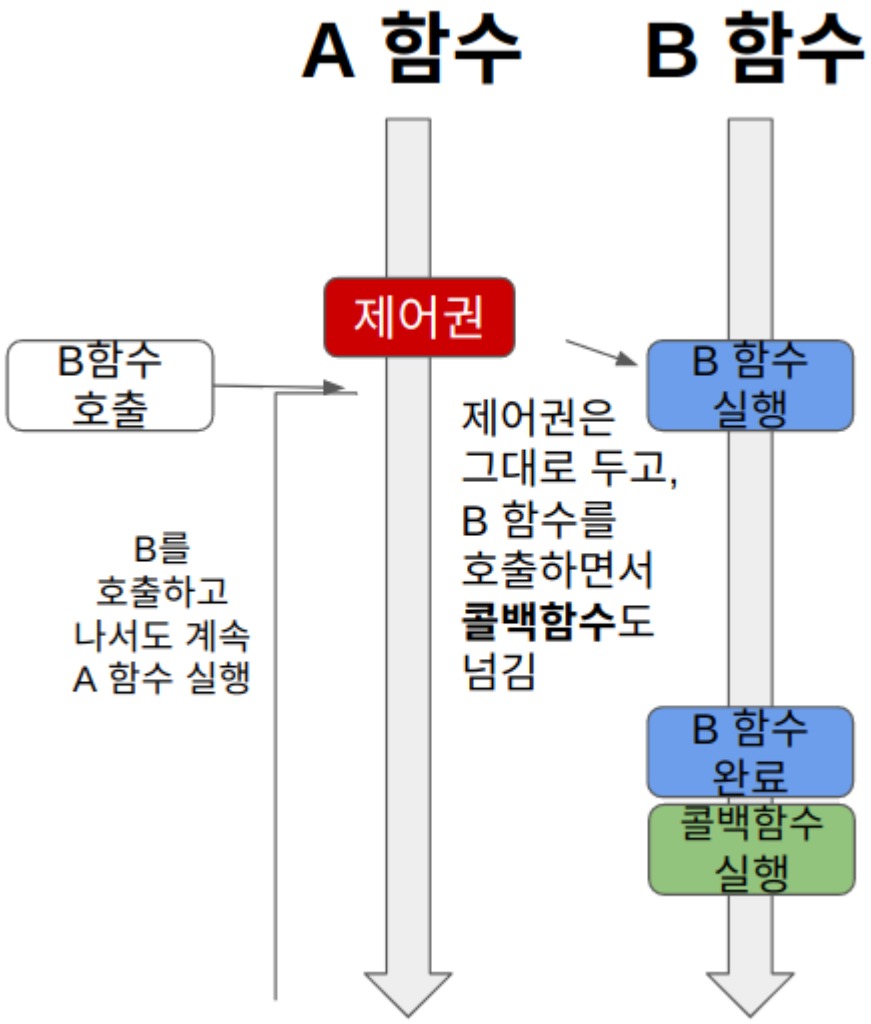
그래서 제어권을 함수 B에게 넘겨주고, 함수 B가 실행을 완료하여 리턴값과 제어권을 돌려줄때까지 **기다린다** (**블로킹**)



EX) C나 JAVA의 코드 실행후 커맨드에서 입력 받기
(제어권이 시스템에서 사용자로 넘어가, 리턴값을 필요로해서 사용자 입력할때 까지 기다림)

4 동기/비동기, blocking/non-blocking 조합

Async-Nonblocking



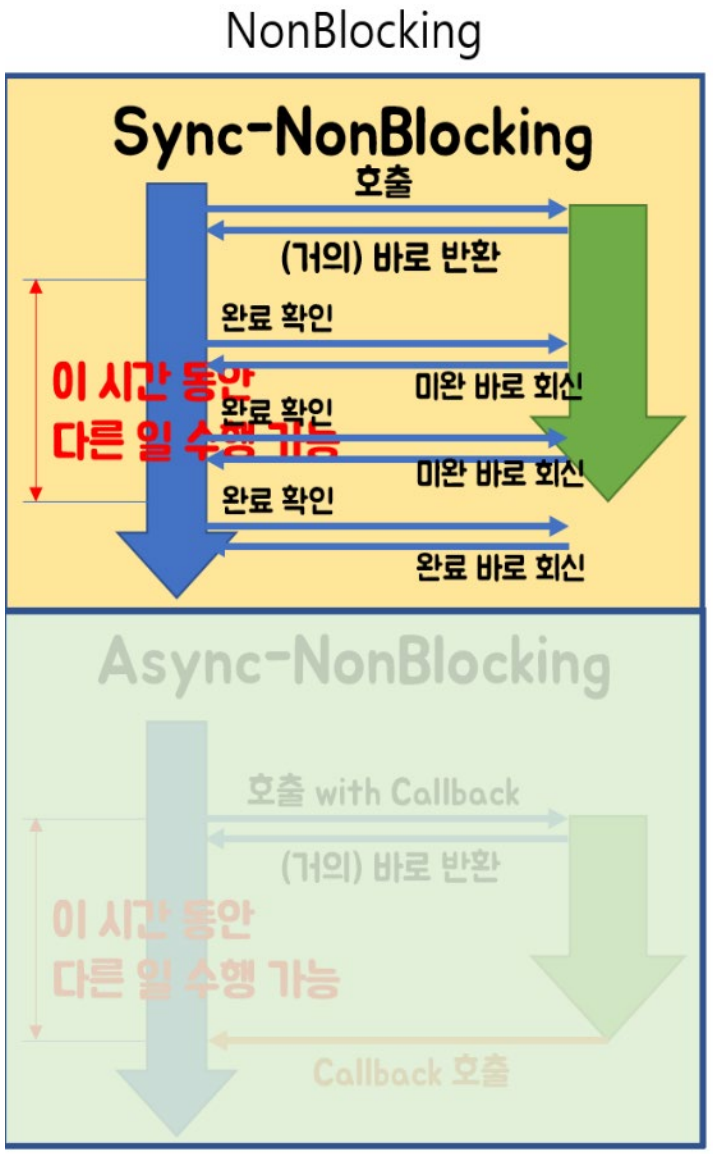
A 함수는 B 함수를 호출한다.
이 때 **제어권을** B 함수에 주지 않고, **자신이 계속 가지고 있는다.** 따라서 B 함수를 호출한 이후에도 멈추지 않고 **자신의 코드를 계속 실행**한다. (**논블로킹**)

그리고 B 함수를 호출할 때 **콜백함수**를 함께 준다. B 함수는 **자신의 작업이 끝나면** A 함수가 준 **콜백 함수를 실행**한다 (**비동기**).

```
const Apihost = "http://localhost:8000"
const getAllUsers = async () => {
    const response = await
axios.get(`${Apihost}/user/all`, {
      withCredentials: true,
    });
    console.log(response.data.data);
    setUsers(response.data.data);
  };
```

4 동기/비동기, blocking/non-blocking 조합

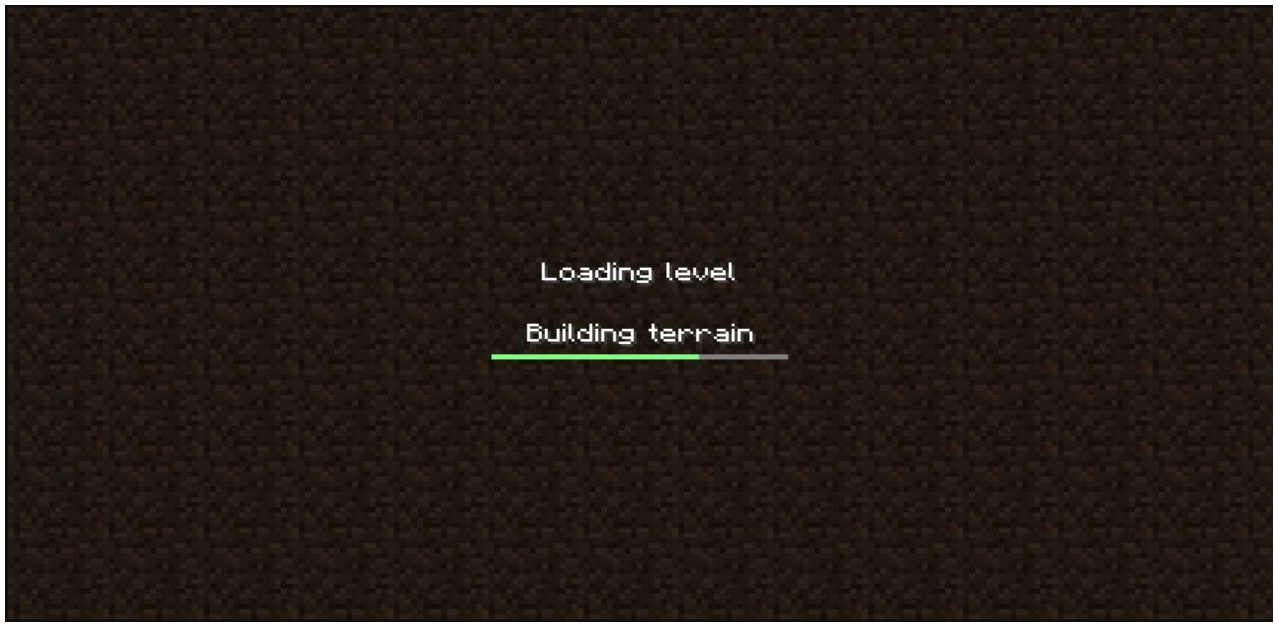
non-blocking + Synchronous



A 함수는 B 함수를 호출한다.

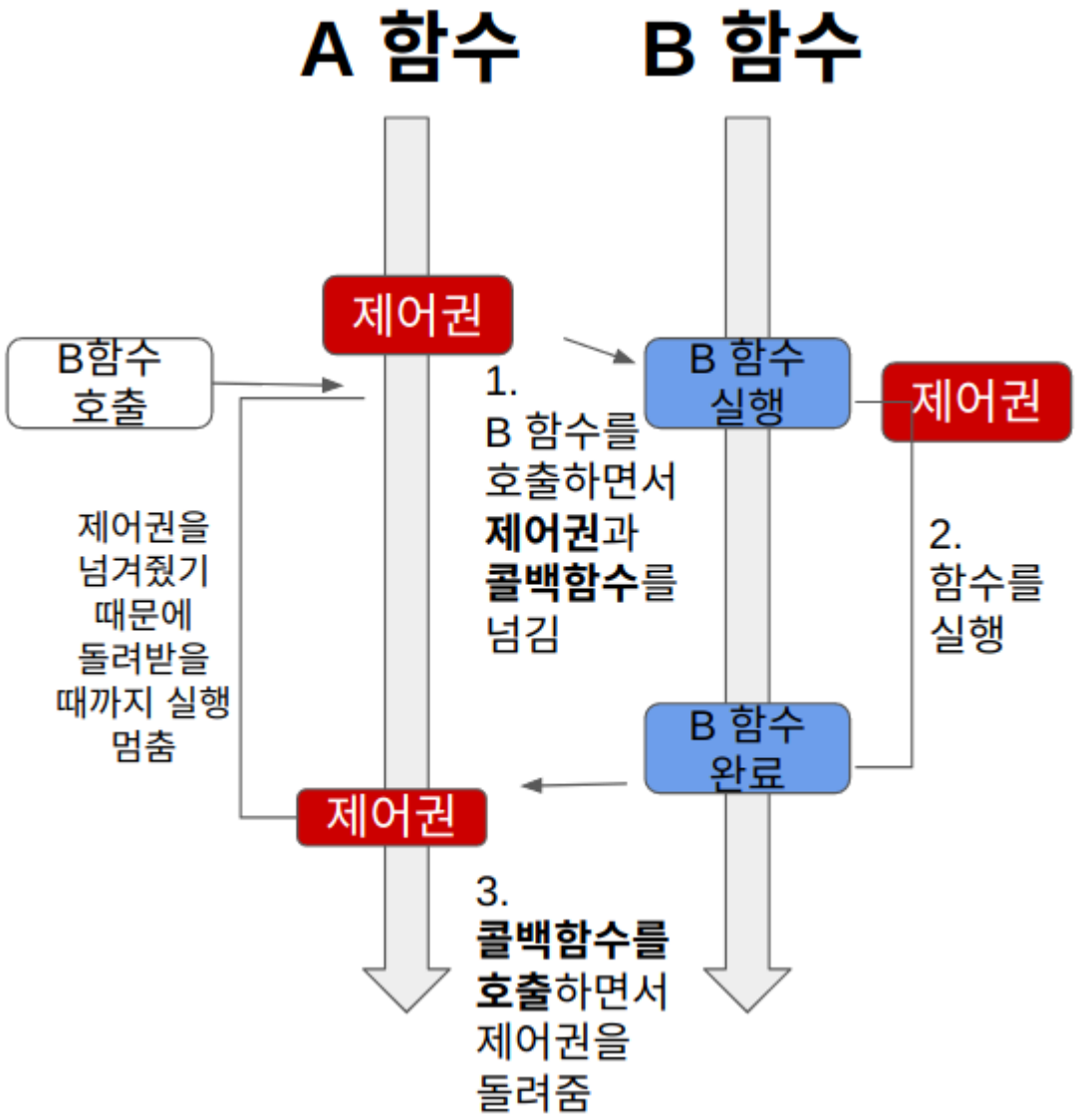
이 때 **A 함수는 B 함수에게 제어권을 주지 않고**, 자신의 코드를 계속 실행한다 (논블로킹)

그런데 **A 함수는 B 함수의 리턴값이 필요하기 때문**에, 중간 중간 B 함수에게 함수 실행을 완료했는지 물어본다 (동기)



4 동기/비동기, blocking/non-blocking 조합

Async-blocking



A 함수는 B 함수의 리턴값에 신경쓰지 않고, 콜백함수를 보낸다 (**비동기**).

따라서, A 함수는 자신과 관련 없는 B 함수의 작업이 끝날 때까지 기다려야 한다.

그런데, B 함수의 작업에 관심없음에도 불구하고, A 함수는 B 함수에게 제어권을 넘긴다 (**블로킹**).

결론

Blocking VS Non-Blocking : 제어의 관점

Sync VS Async: 순서와 결과(처리)의 관점

블로킹/논블로킹 요청받는 함수가 제어권(함수실행권)을 언제 넘겨주느냐의 차이

블로킹 : 요청받는 함수가 작업을 모두 마치고 나서야 요청자에게 제어권을 넘겨줌 (그동안 요청자는 아무것도 하지않고 기다림)

논블로킹 : 요청받은 함수가 요청자에게 제어권을 바로 넘겨줌 (그동안 요청자는 다른 일을 할 수 있음)

동기/비동기 요청받은 함수가 작업을 완료했는지를 체크해서 순차적 흐름의 차이

동기 : 요청자가 요청받은 함수의 작업이 완료되었는지 계속 확인 (여러 함수들이 시간을 맞춰 실행됨)

비동기 : 요청자는 요청 후 신경X, 요청받은 함수가 작업을 마치면 알려줌 (함수들의 작업 시작/종료 시간이 맞지 않을수도)

THANK YOU