41장 타이머 /
42장 비동기 프로그래밍

모던 자바스크립트 Deep Dive

호출 스케줄링

- 함수를 명시적으로 호출하지 않고, **일정 시간이 경과된 이후에 호출**되도록 타이머 함수를 사용하여 **함수 호출을 예약**하는 것

타이머 생성	타이머 제거
setTimeout	clearTimeout
setInterval	clearInterval

타이머 함수 - setTimeout

- 첫 번째 인수로 전달받은 콜백 함수가 두 번째 인수로 전달받은 시간 이후 단 **한 번 실행**되도록 호출 스케줄링

```
const timeoutId =
      setTimeout(func|code[, delay, param1, param2, …]);
```

타이머가 만료된 뒤 호출될 콜백 함수

타이머 만료 시간

콜백 함수에 전달할 인수

타이머 함수 - setTimeout

- 첫 번째 인수로 전달받은 콜백 함수가 두 번째 인수로 전달받은 시간 이후 단 한 번 실행되도록 호출 스케줄링

```
setTimeout((n) => console.log(`Hi! ${n}.`), 1000, 'Lee');
```

콜백 함수 | 시간 | 인수

- 1초(1000ms) 후 타이머가 만료되면 콜백 함수가 호출됨
- 이 때 콜백 함수에 'Lee'가 인수로 전달됨

타이머 함수 - clearTimeout

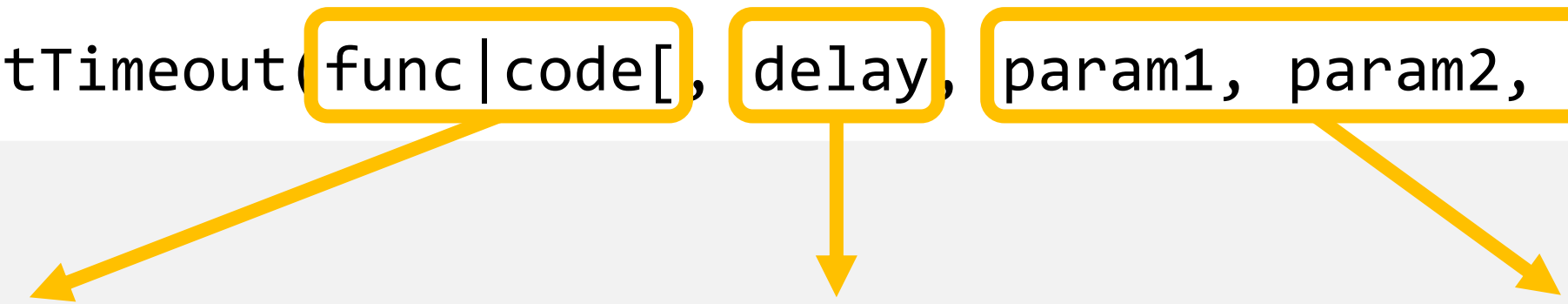
- setTimeout 함수가 반환한 타이머 id를 clearTimeout 함수의 인수로 전달하여 **타이머를 취소**할 수 있음 (호출 스케줄링 취소)

```
const id = setTimeout(() => console.log('Hi!'), 1000);

clearTimeout(id);
```

타이머 함수 - setInterval

- 첫 번째 인수로 전달받은 콜백 함수가 두 번째 인수로 전달받은 시간이 경과할 때마다 **반복 실행**되도록 호출 스케줄링

```
const timerId =
      setTimeout(func|code[, delay, param1, param2, …]);
```



타이머가 만료된 뒤 호출될 콜백 함수

타이머 만료 시간

콜백 함수에 전달할 인수

타이머 함수 - clearInterval

- setInterval 함수가 반환한 타이머 id를 clearInterval 함수의 인수로 전달하여 **타이머를 취소**할 수 있음 (호출 스케줄링 취소)

```
let count = 1;


const timeoutId = setInterval(() => {
  console.log(count); // 1 2 3 4 5
  if (count++ === 5) clearInterval(timeoutId);
}, 1000);
```

디바운스와 스로틀

- scroll, resize, input, mousemove 같은 이벤트는 **짧은 시간 간격으로 연속해서 발생**하고, 이러한 이벤트에 바인딩한 이벤트 핸들러는 과도하게 호출되어 **성능에 문제** 일으킬 수 있음

- **짧은 시간 간격으로 연속해서 발생하는 이벤트를 그룹화**해서 과도한 이벤트 핸들러의 호출을 방지함

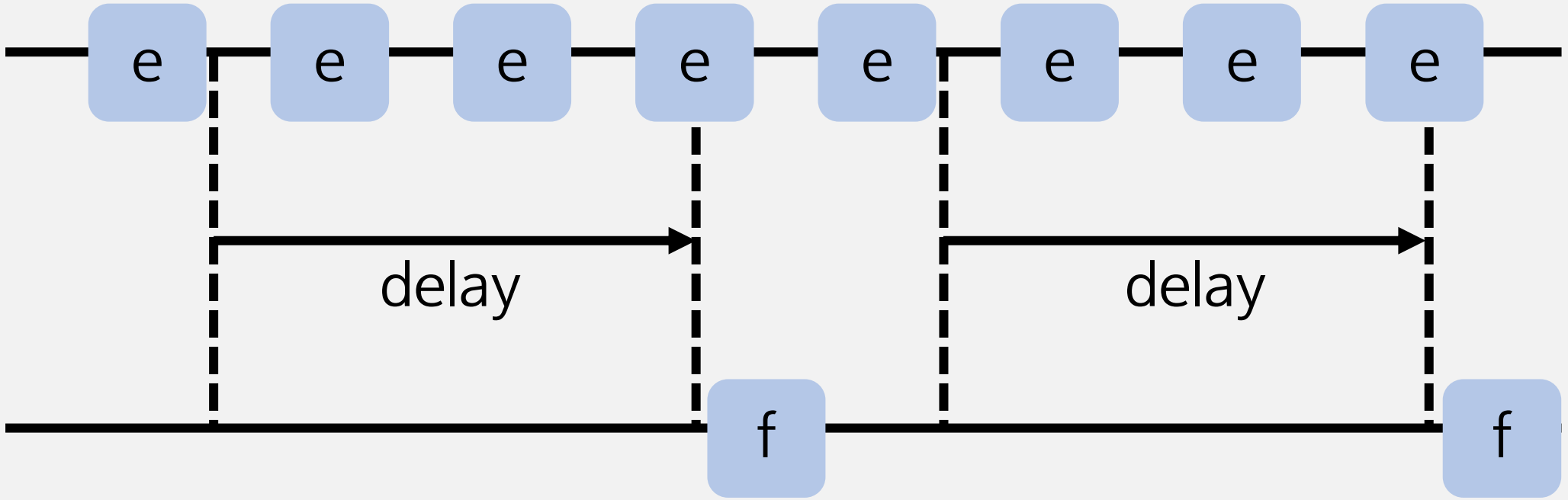
디바운스

- 짧은 시간 간격으로 이벤트가 연속해서 발생하면 이벤트 핸들러를 호출하지 않다가, **일정 시간이 경과한 이후에 이벤트 핸들러가 한 번만 호출**되도록 함

```
const debounce = (callback, delay) => {
  let timerId;
  return (event) => {
    if (timerId) clearTimeout(timerId);
    timerId = setTimeout(callback, delay, event);
  };
};
```

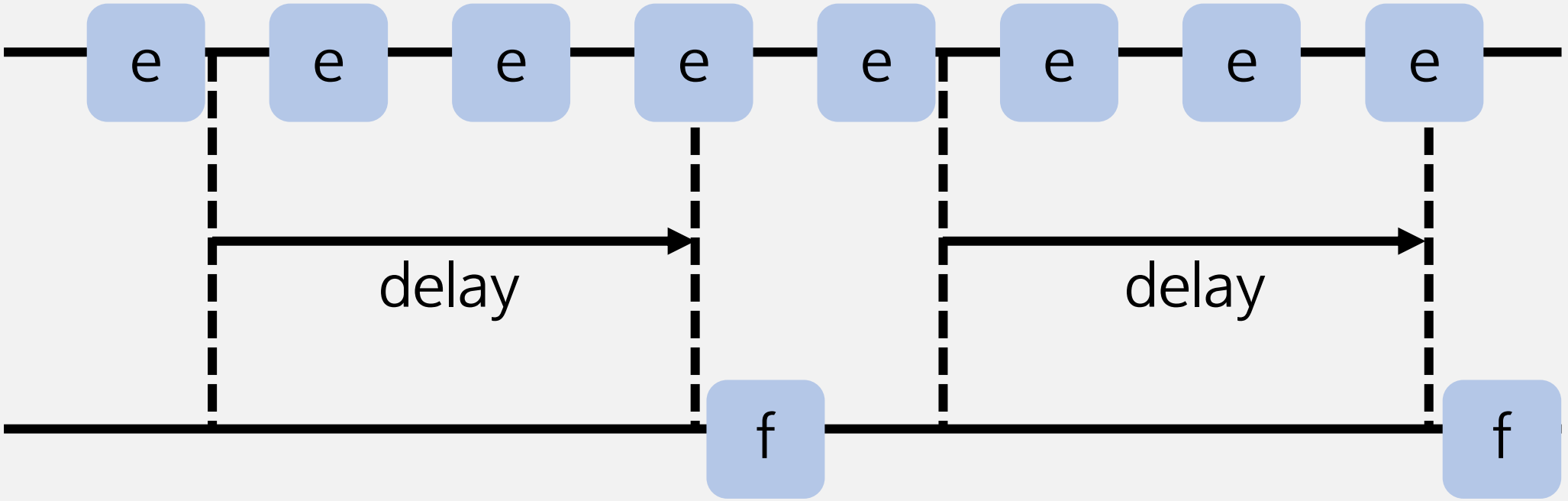
디바운스

- 짧은 시간 간격으로 이벤트가 연속해서 발생하면 이벤트 핸들러를 호출하지 않다가, **일정 시간이 경과한 이후에 이벤트 핸들러가 한 번만 호출**되도록 함



스로틀

- 짧은 시간 간격으로 이벤트가 연속해서 발생하더라도 **일정 시간 간격으로 이벤트 핸들러가 최대 한 번만 호출되도록 함**



디바운스 vs 스로틀

- **디바운스** -〉 연이어 호출되는 함수들 중 가장 마지막 (혹은 가장 처음) 함수만 호출

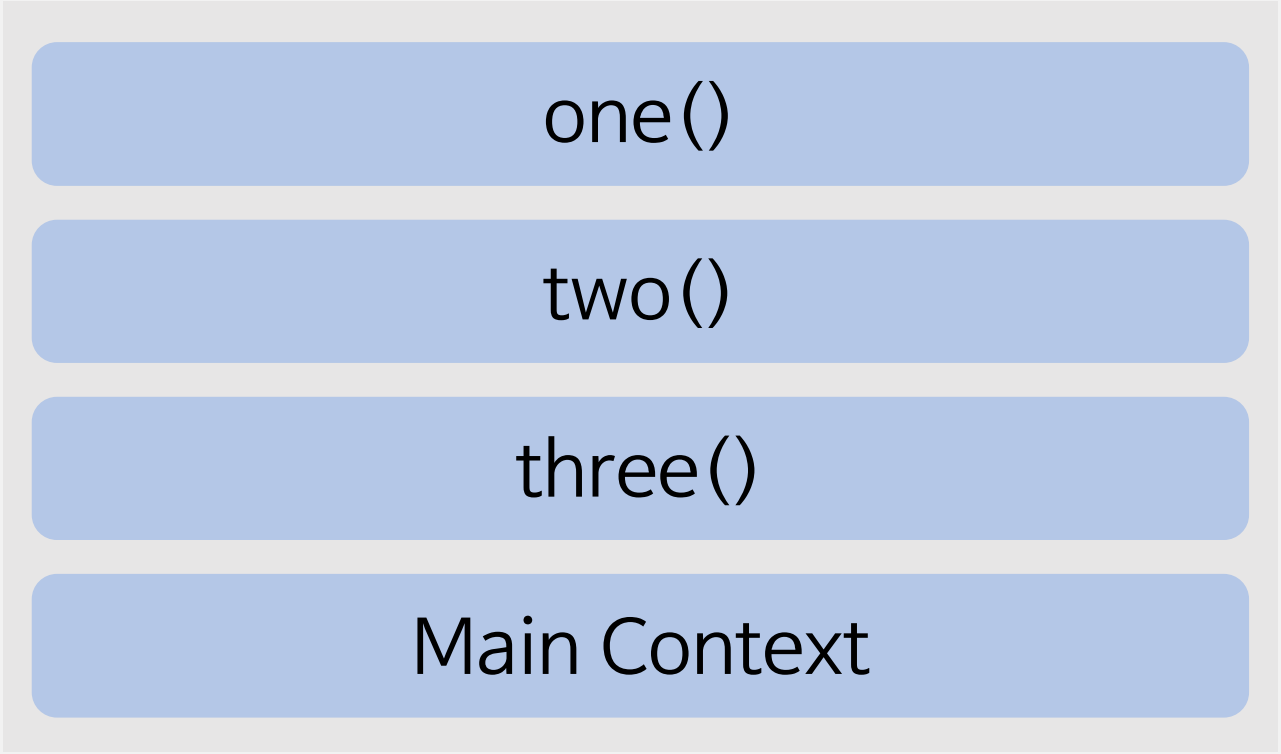
- **스로틀** -〉 마지막 함수가 호출된 후 일정 시간이 지나기 전에 다시 호출되지 않도록 함

동기 처리와 비동기 처리

- 함수 실행 컨텍스트가 **실행 컨텍스트 스택(콜 스택)에 푸시**되고 함수 코드가 실행됨

```
function one() {
  return 1;
}

function two() {
  return one() + 1;
}

function three() {
  return two() + 1;
}

console.log(three());
```

실행 컨텍스트 스택 (Call Stack)

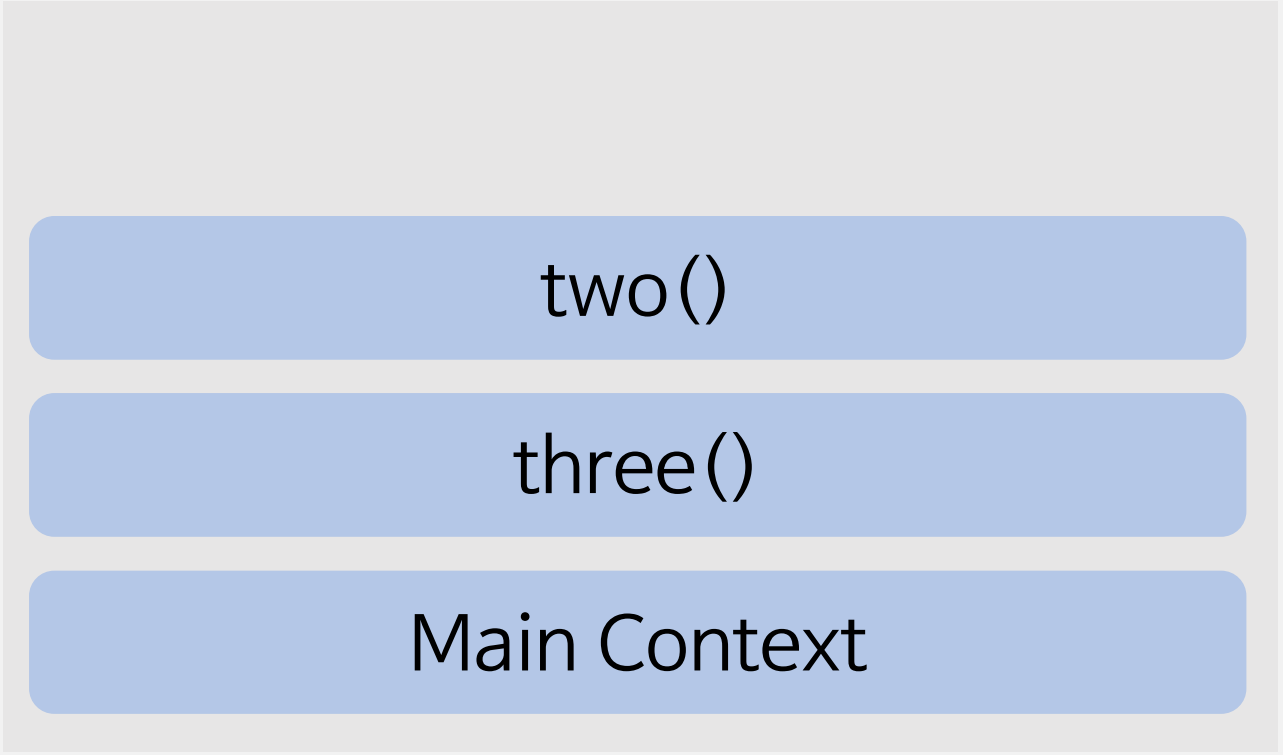


동기 처리와 비동기 처리

- 함수 실행 컨텍스트가 **실행 컨텍스트 스택(콜 스택)에 푸시**되고 함수 코드가 실행됨

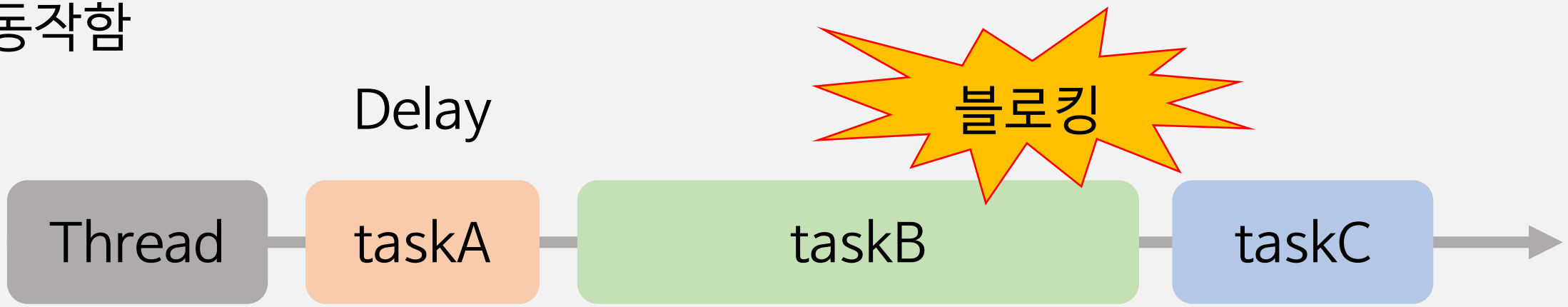
```
function one() {
  return 1;
}

function two() {
  return one() + 1;
}

function three() {
  return two() + 1;
}

console.log(three());
```

실행 컨텍스트 스택 (Call Stack)



동기 처리와 비동기 처리

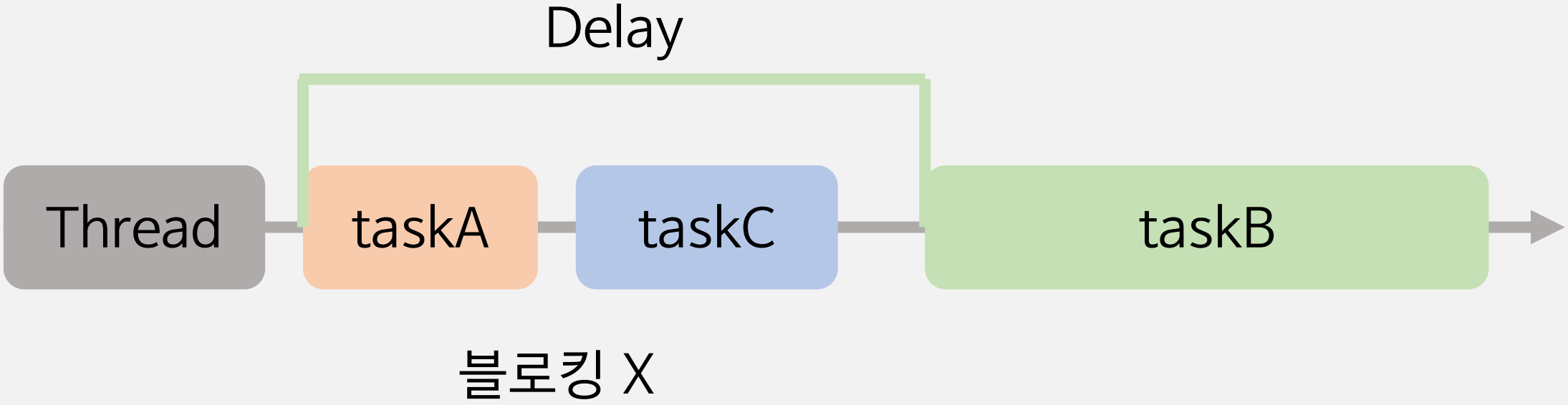
- 자바스크립트 엔진은 단 하나의 콜 스택만을 가지는 **싱글 스레드** 방식으로 동작함



- 자바스크립트는 코드가 작성된 순서대로 작업을 처리하며,
  한번에 하나의 태스크만 실행할 수 있기 때문에
  현재 실행 중인 태스크가 끝날 때까지 다음 태스크를 실행할 수 없음

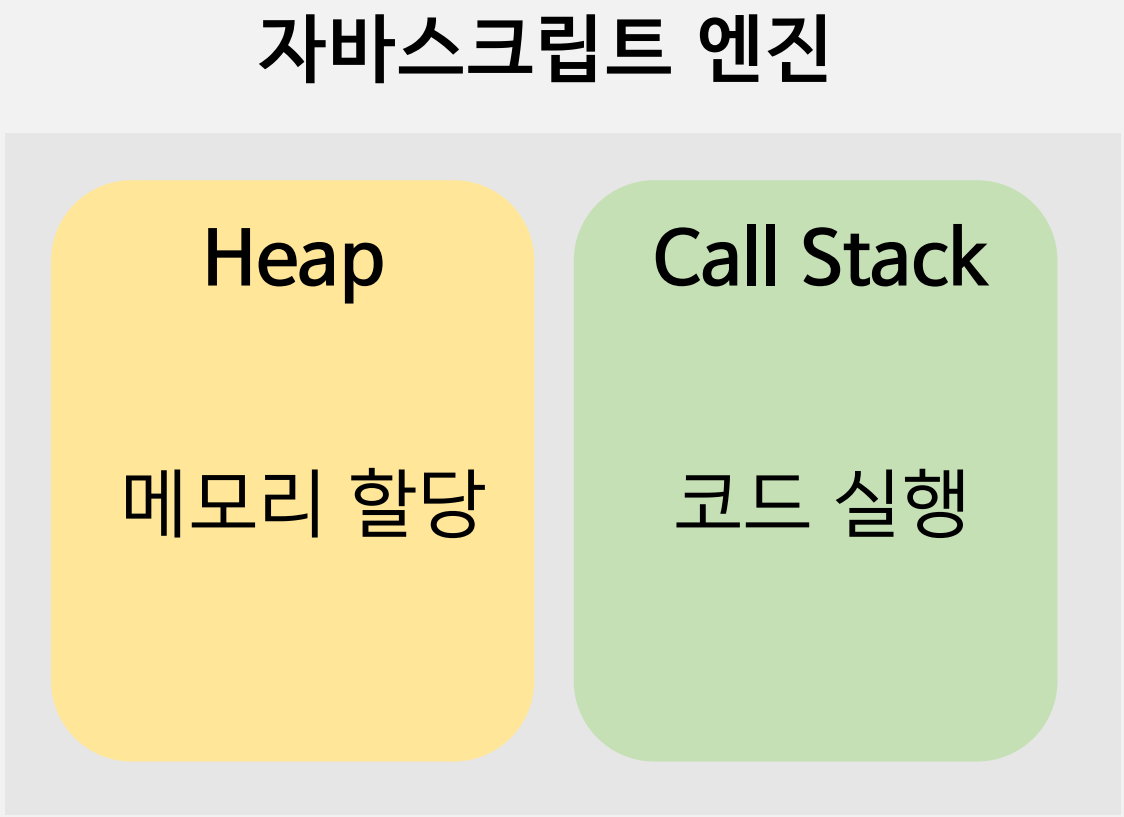
동기 처리와 비동기 처리

- 현재 실행 중인 태스크가 종료되지 않은 상태여도 다음 태스크를 곧바로 실행하는 방식을 **비동기 처리**라고 함



힙과 콜 스택

- 자바스크립트 엔진은 크게 **힙**, **콜 스택** 2개의 영역으로 구분할 수 있음



이벤트 루프와 태스크 큐

- 브라우저 환경은 **태스크 큐**와 **이벤트 루프**를 제공

Task Queue

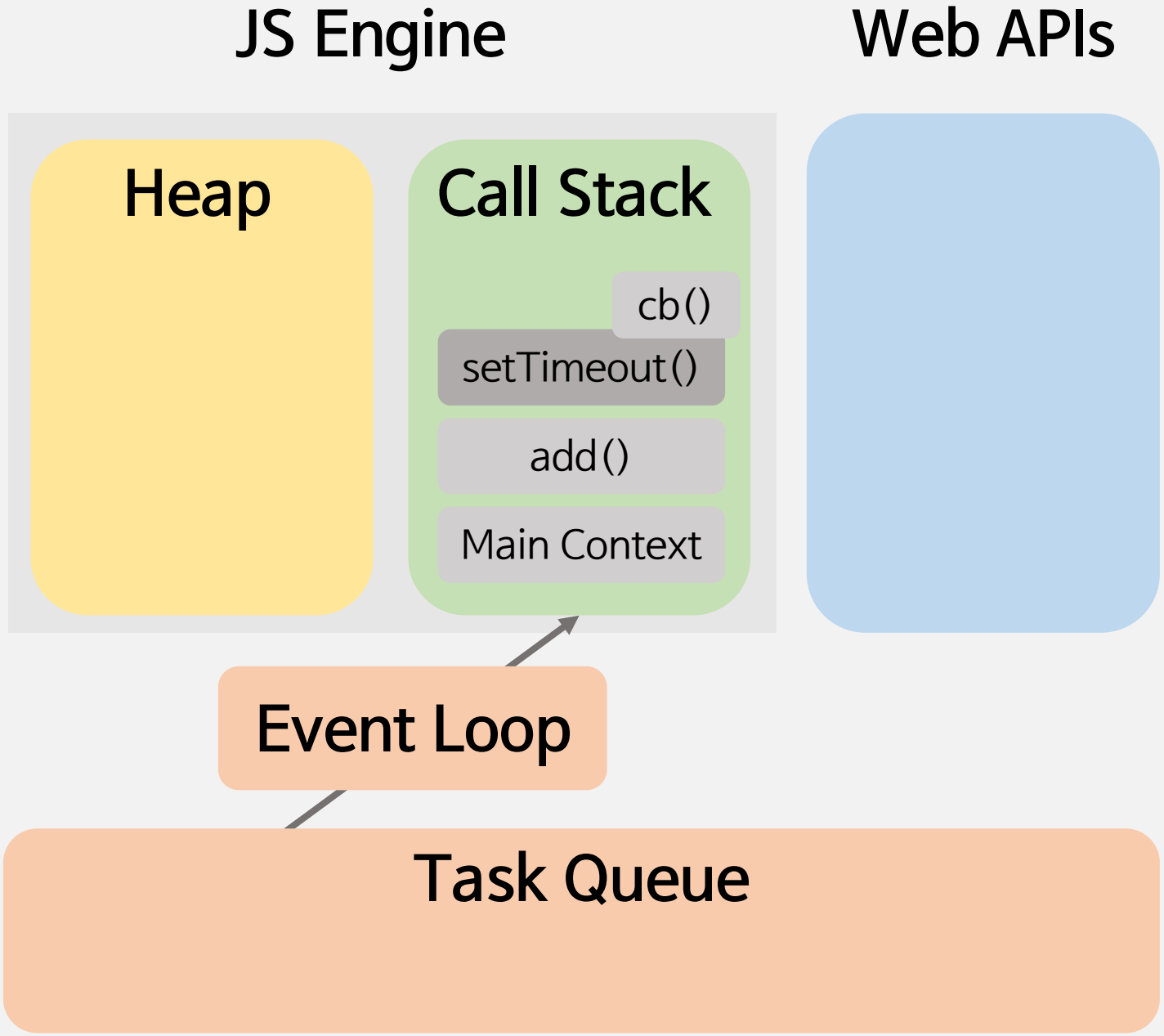
콜백 함수나 이벤트 핸들러 일시적으로 보관

Event Loop

태스크 큐에 대기 중인 함수를 콜 스택으로 이동시킴

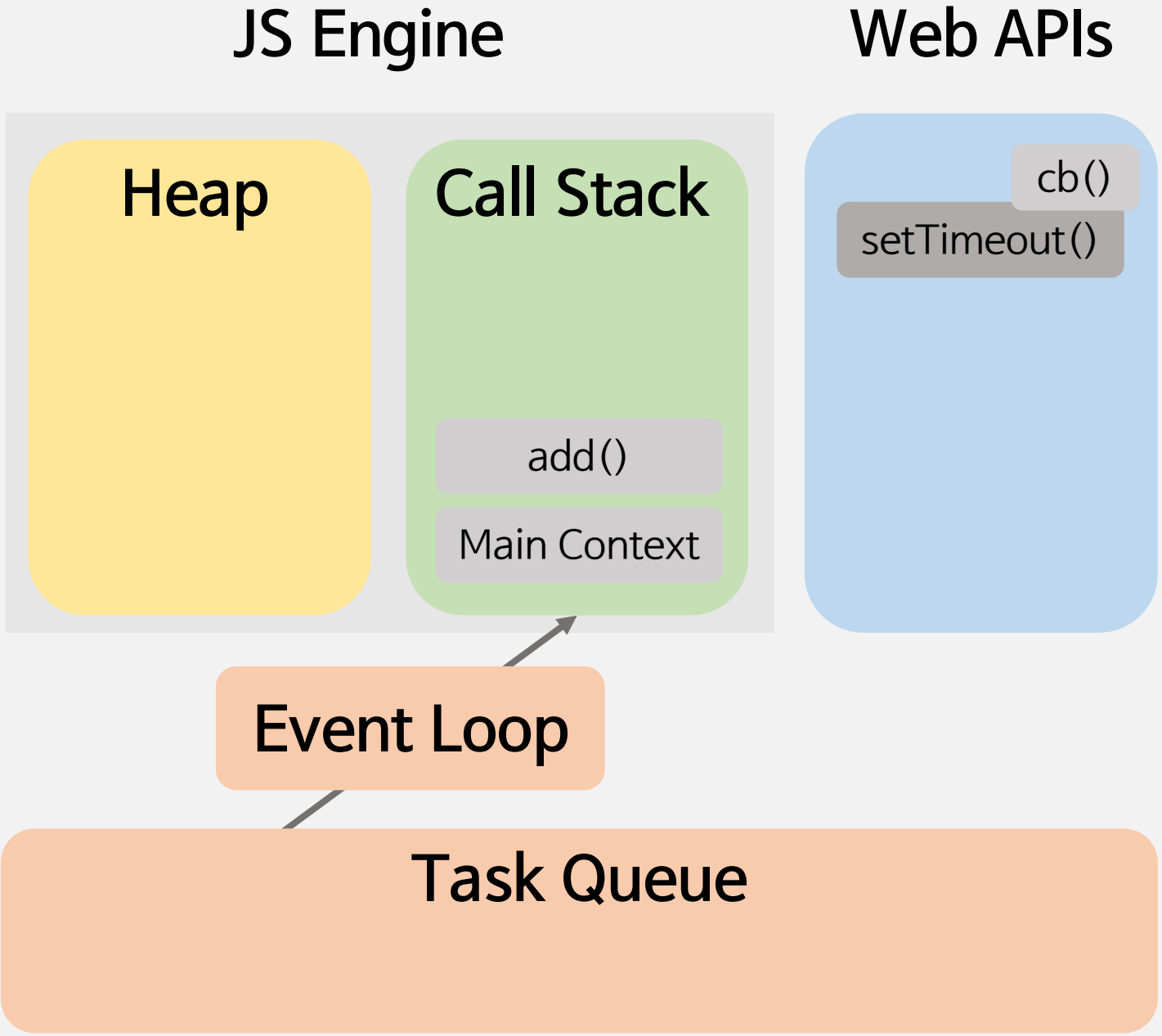
비동기 처리

```
function add(a, b, cb) {
  setTimeout(() => {
    const res = a + b;
    cb(res);
  }, 3000);
}

add(1, 3, (res) => {
  console.log('결과: ', res);
});
```



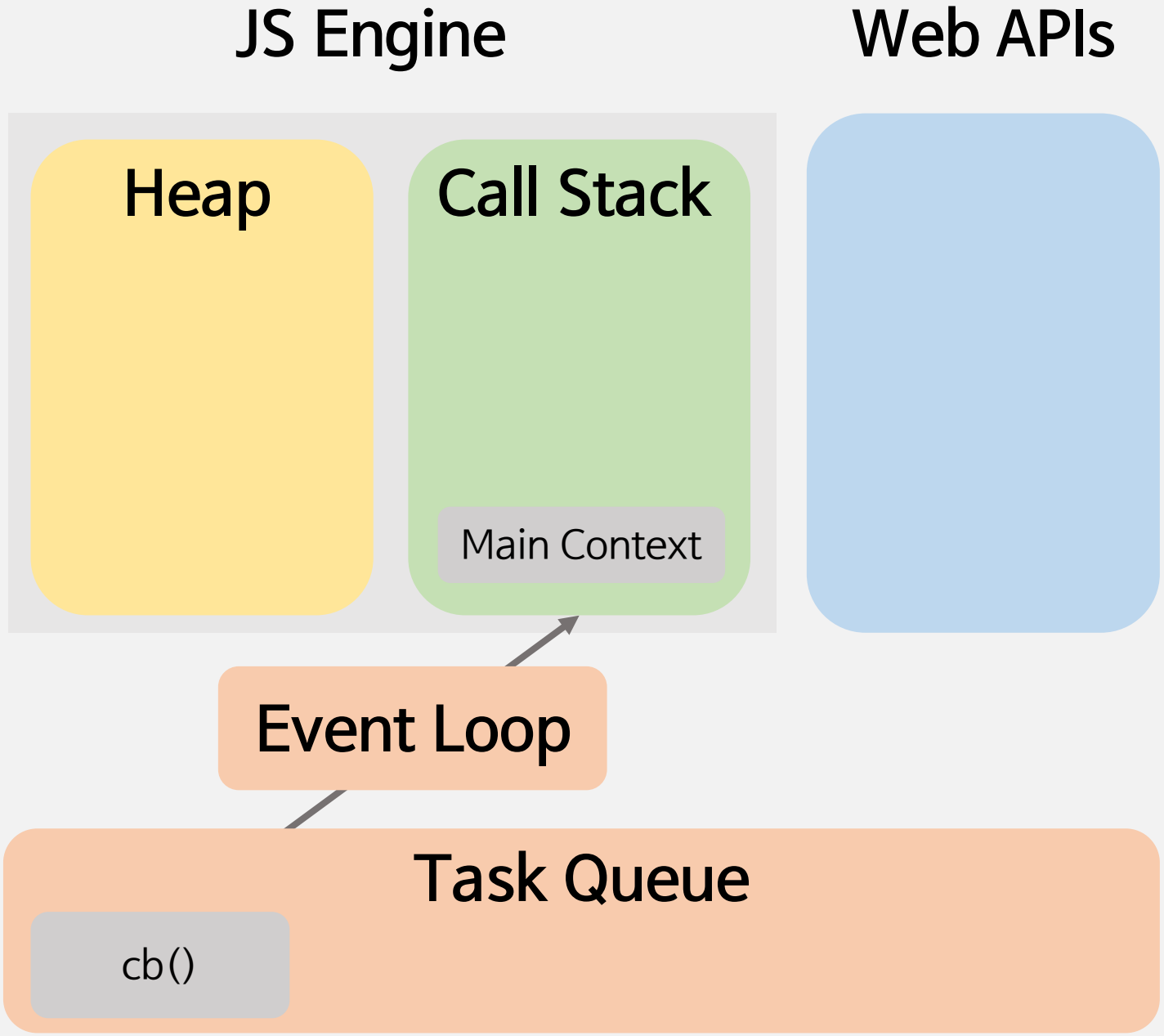
비동기 처리

```
function add(a, b, cb) {
  setTimeout(() => {
    const res = a + b;
    cb(res);
  }, 3000);
}

add(1, 3, (res) => {
  console.log('결과: ', res);
});
```



비동기 처리

```
function add(a, b, cb) {
  setTimeout(() => {
    const res = a + b;
    cb(res);
  }, 3000);
}

add(1, 3, (res) => {
  console.log('결과: ', res);
});
```



비동기 처리

```
function add(a, b, cb) {
  setTimeout(() => {
    const res = a + b;
    cb(res);
  }, 3000);
}

add(1, 3, (res) => {
  console.log('결과: ', res);
});
```