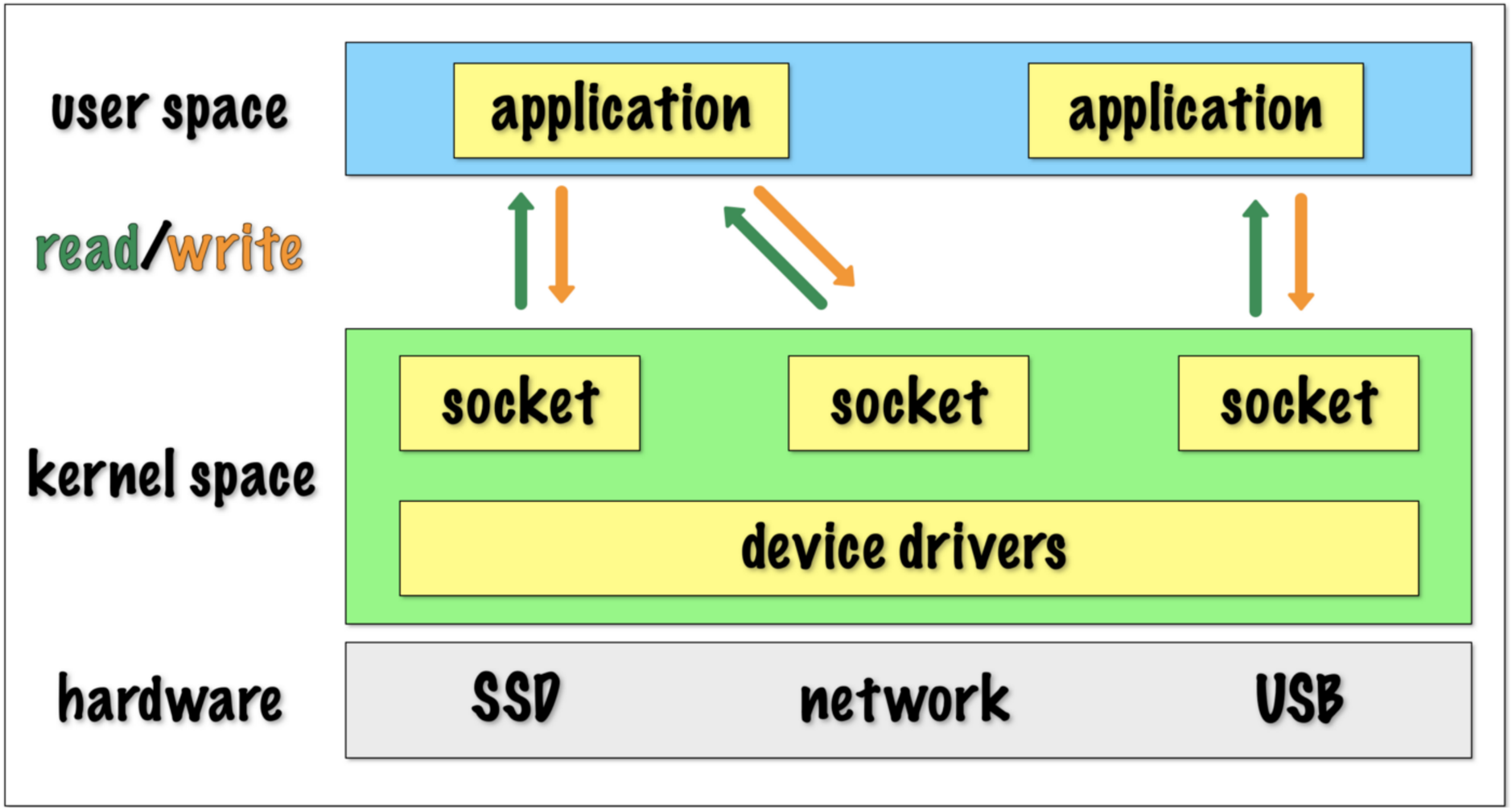
# C10k 문제와 Node.js

# C10k 문제

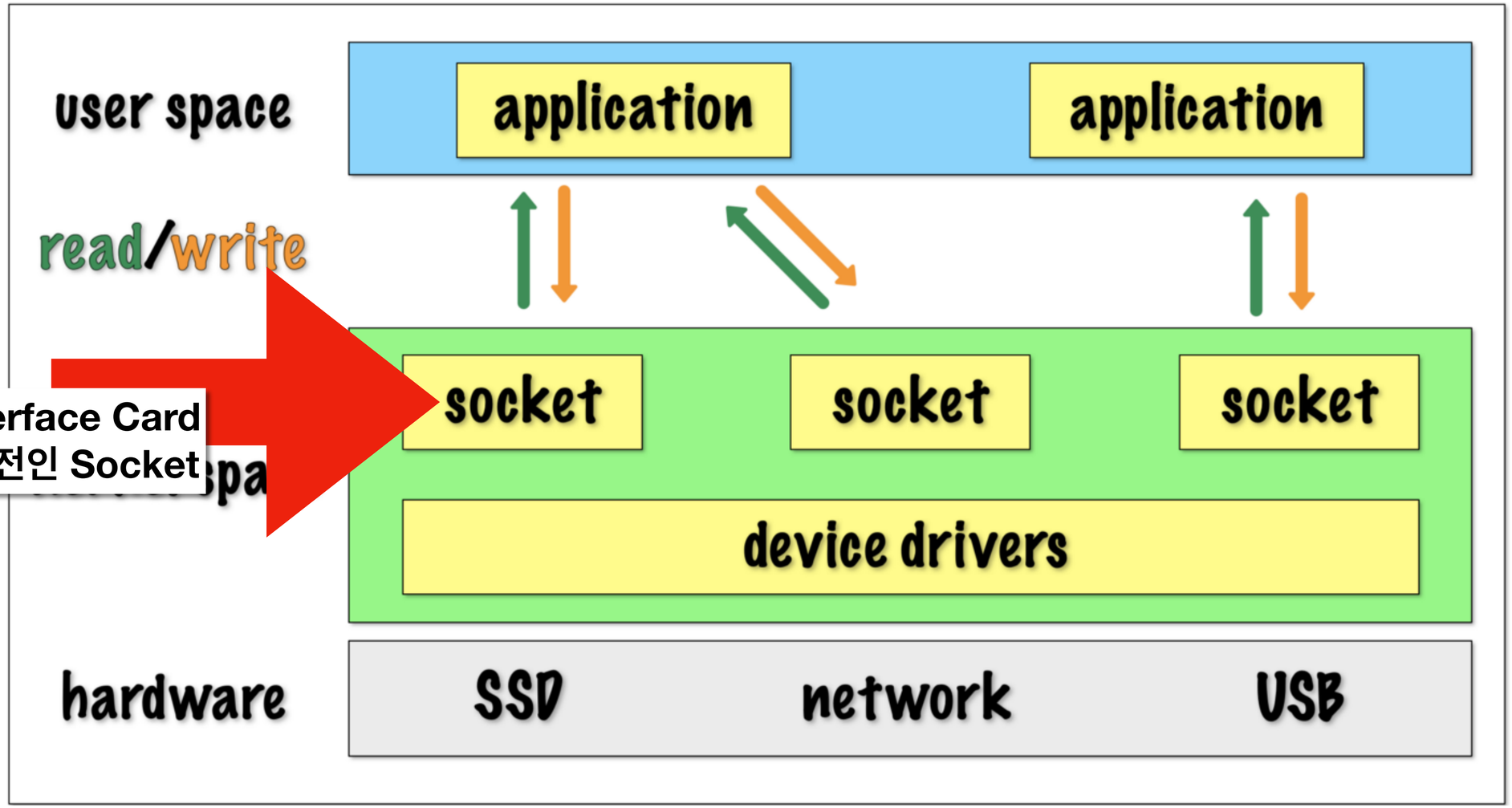
- The C10k problem is the problem of optimizing **network sockets to handle a large number of clients at the same time.**
- **많은 숫자의 클라이언트의 요청을 동시에 처리해야하는 문제.**

https://en.wikipedia.org/wiki/C10k_problem

# 어떻게 Network I/O는 동작할까?



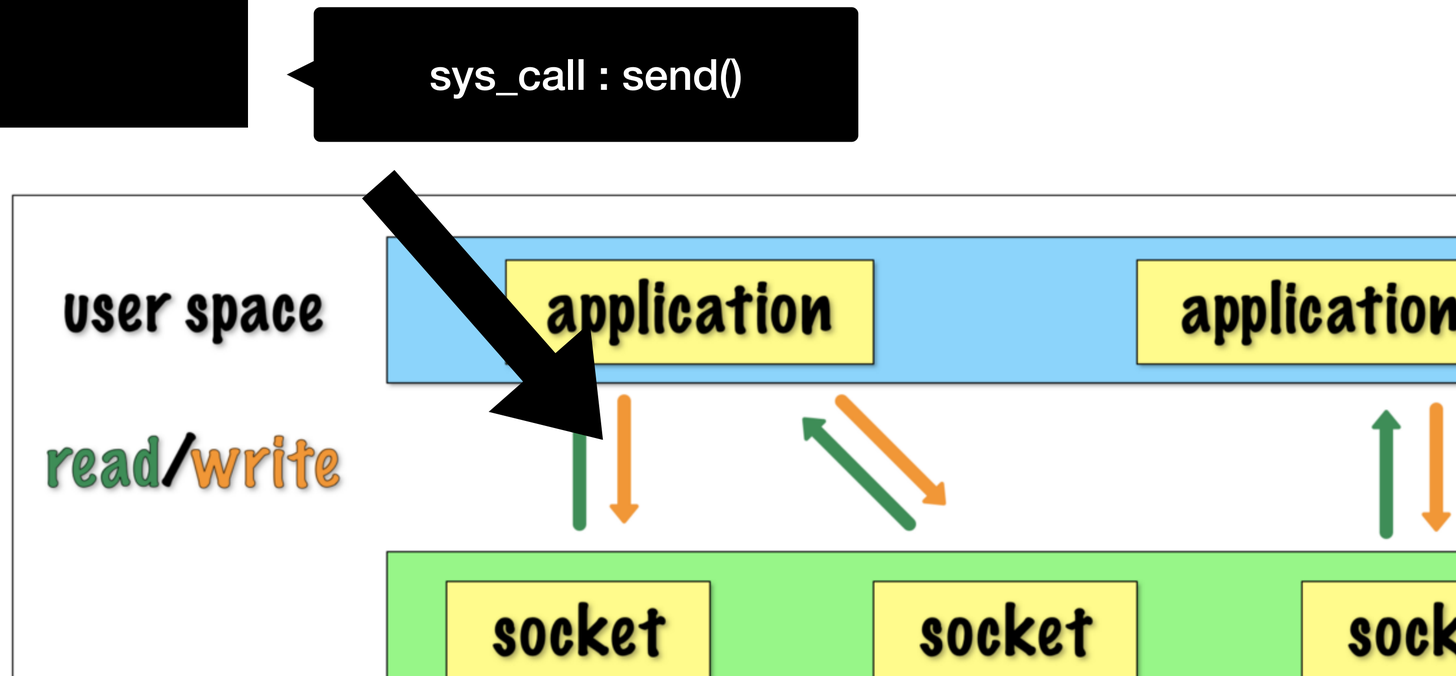
# 어떻게 Network I/O는 동작할까?




Network Interface Card
의 추상화된 버전인 Socket

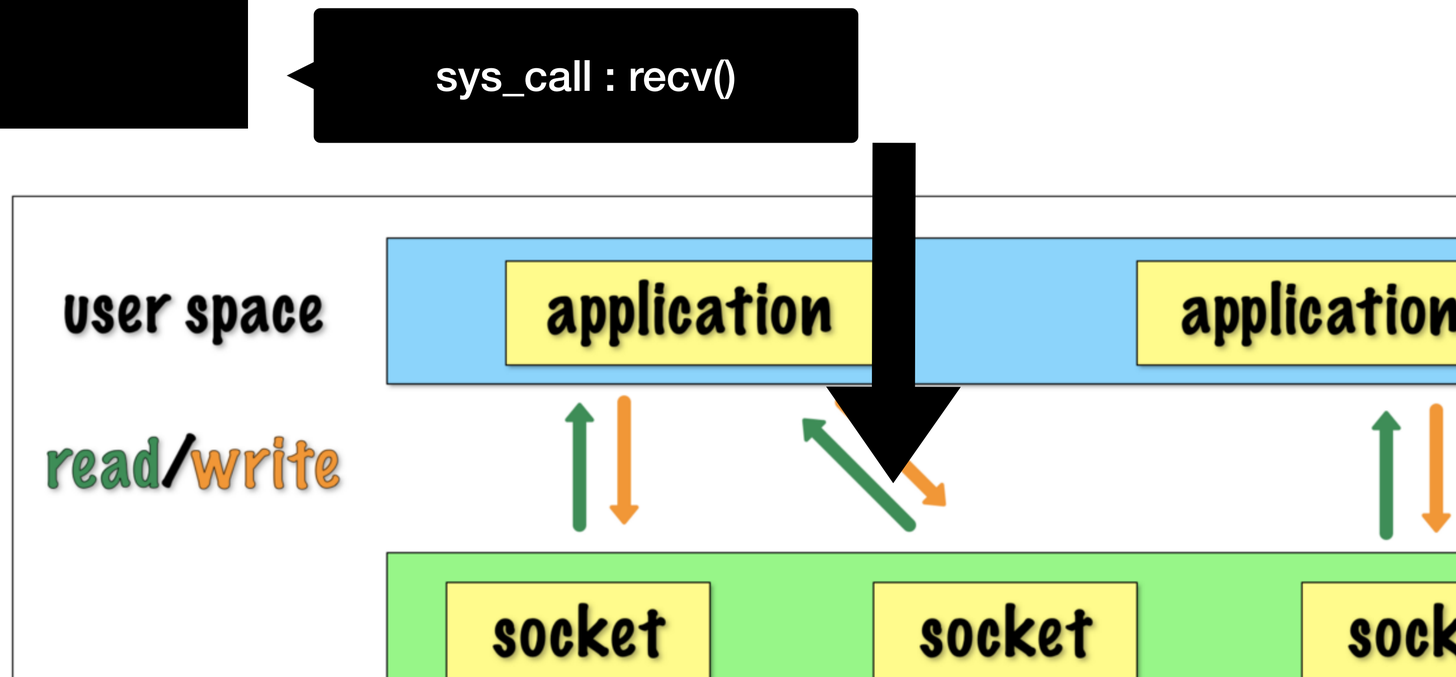
# 어떻게 Network I/O는 동작할까?

HTTP Request



# 어떻게 Network I/O는 동작할까?

HTTP Response



# Blocking I/O의 동작 방식 (Network)

- Blocking이 되기 때문에 쓰레드를 하나 생성하고 그곳에서 I/O 요청을 한다.
- I/O 요청을 하는 동안 Thread는 Sleep상태로 들어간다.
- I/O 응답이 오면 Thread는 깨어나서 다시 CPU에 의해 점유된다.
- 응답을 받고 데이터를 처리한다.

# Blocking I/O의 동작 방식 (Network)

## 어디서 병목이 생길까?

- Blocking이 되기 때문에 쓰레드를 하나 생성하고 그곳에서 I/O 요청을 한다.
- I/O 요청을 하는 동안 Thread는 Sleep상태로 들어간다.
- I/O 응답이 오면 Thread는 깨어나서 다시 CPU에 의해 점유된다.
- 응답을 받고 데이터를 처리한다.

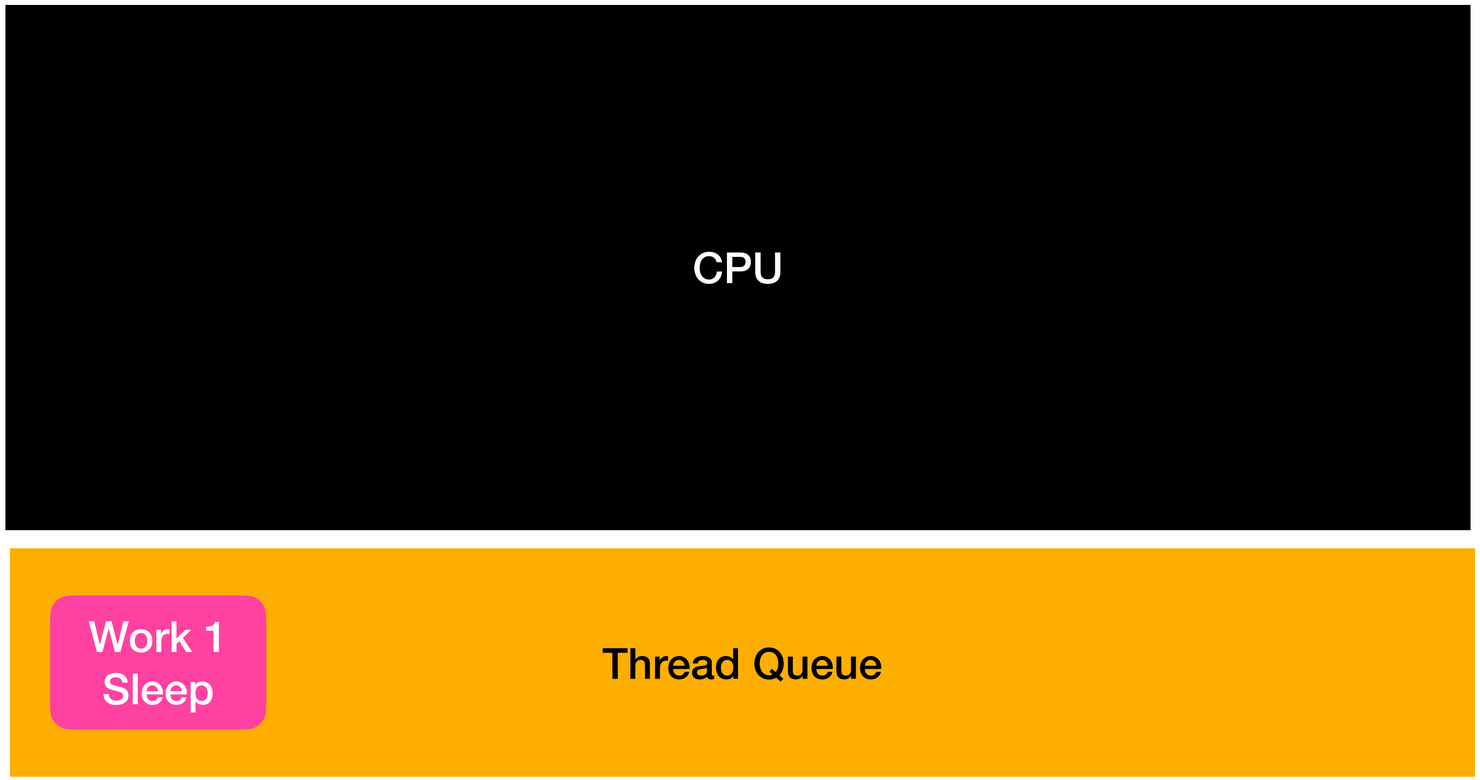
# Blocking I/O의 동작 방식 (Network)

## 어디서 병목이 생길까?

- Blocking이 되기 때문에 **쓰레드를 하나 생성하고** 그곳에서 I/O 요청을 한다.
- **I/O 요청을 하는 동안 Thread는 Sleep상태로 들어간다.**
- **I/O 응답이 오면 Thread는 깨어나서 다시 CPU에 의해 점유된다.**
- 응답을 받고 데이터를 처리한다.

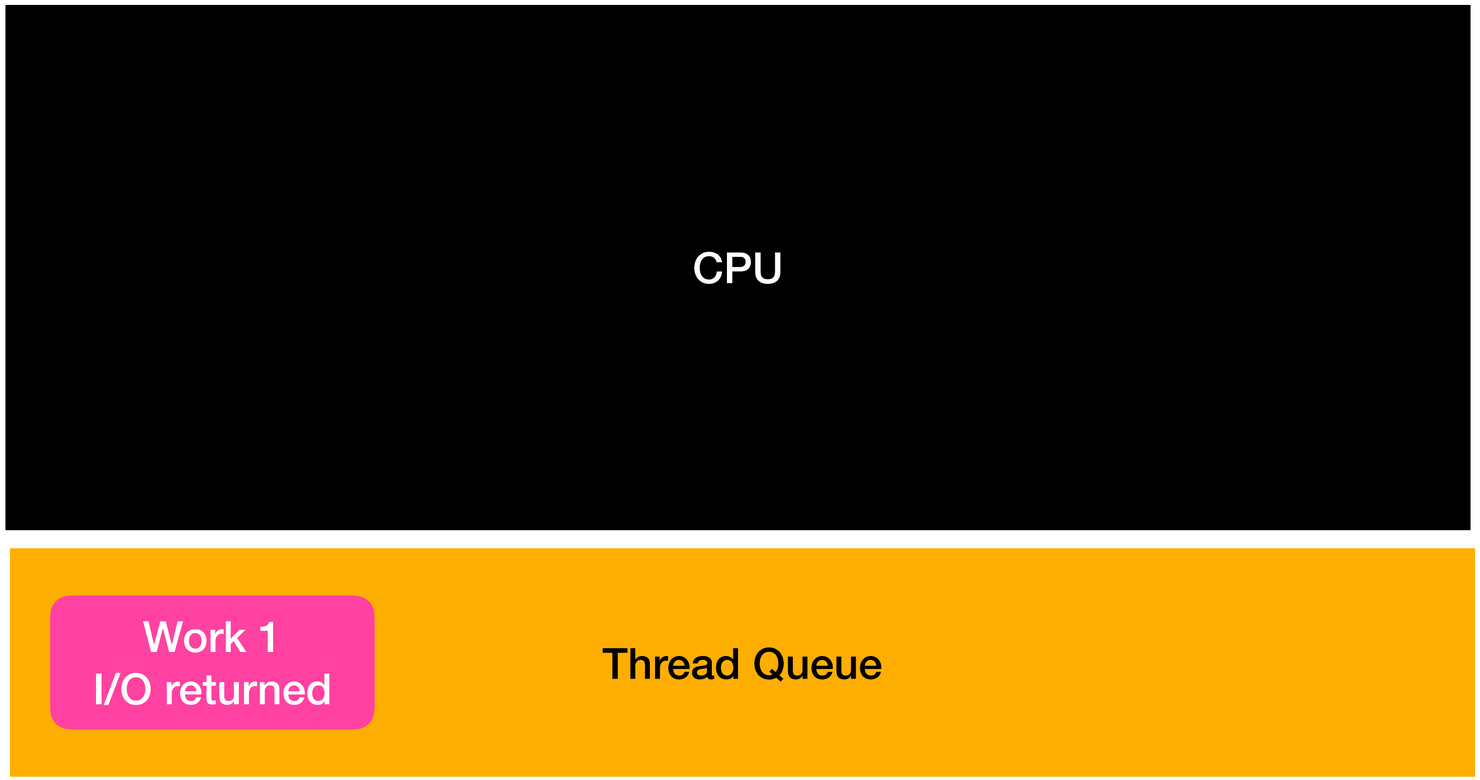
# Blocking I/O의 동작 방식 (Network)

## 어디서 병목이 생길까?



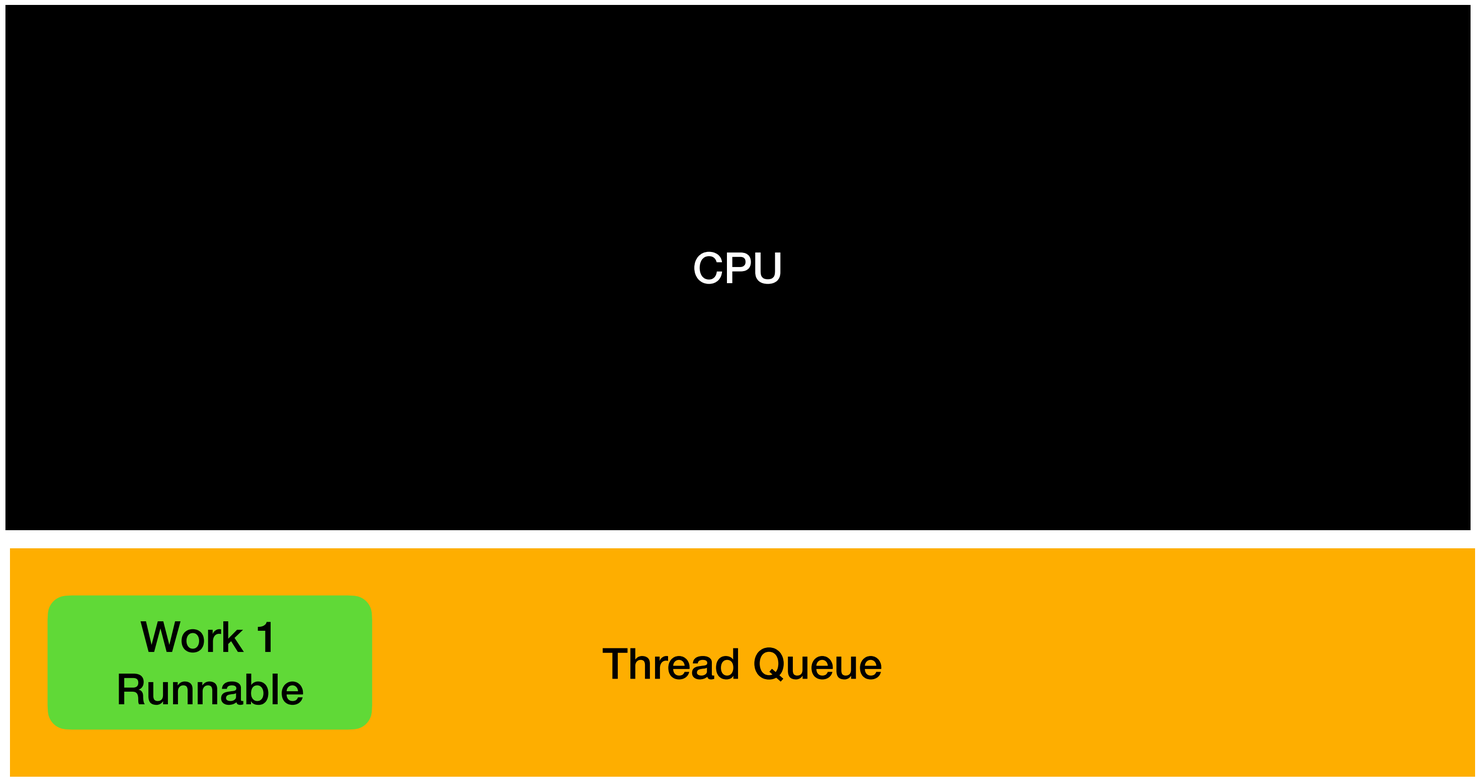
# Blocking I/O의 동작 방식 (Network)

## 어디서 병목이 생길까?



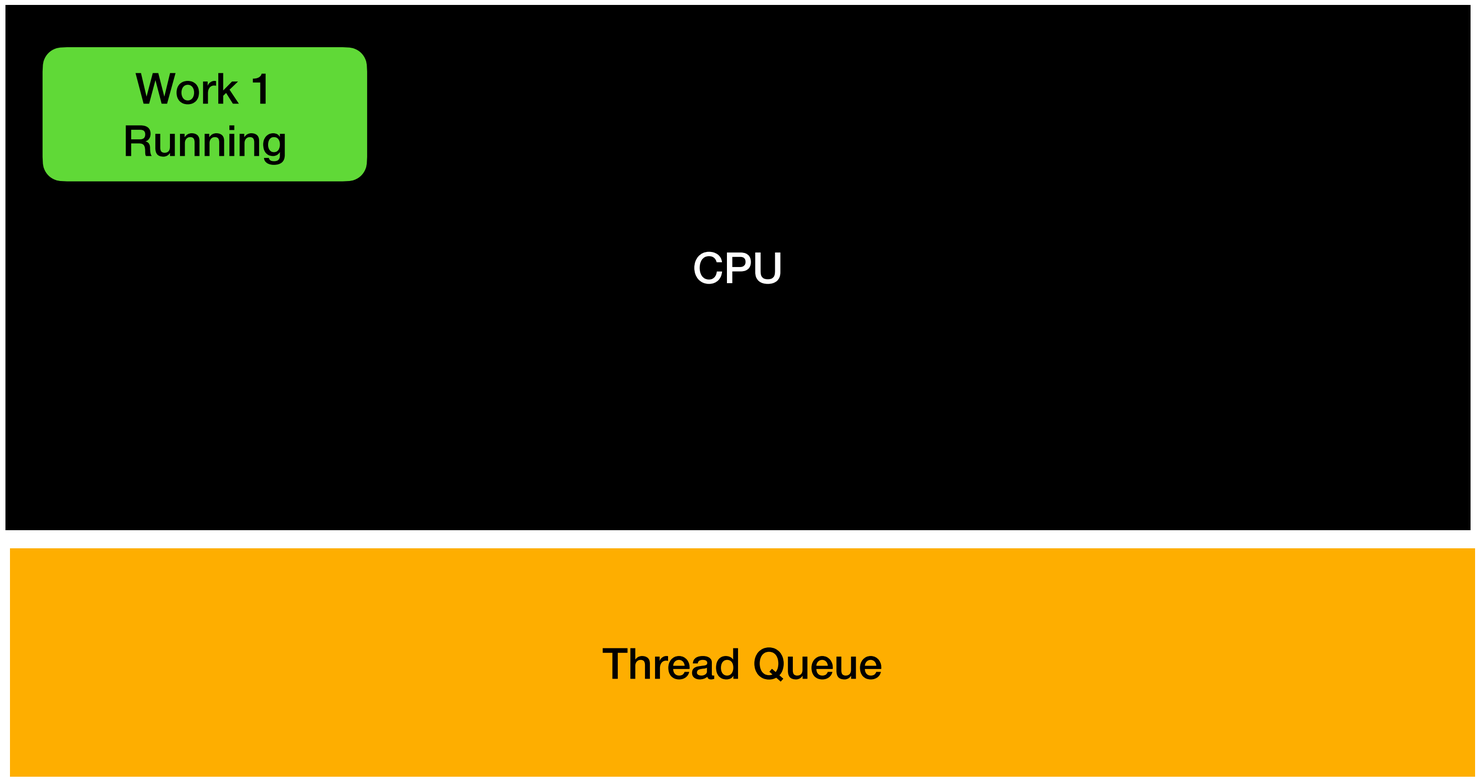
# Blocking I/O의 동작 방식 (Network)

## 어디서 병목이 생길까?

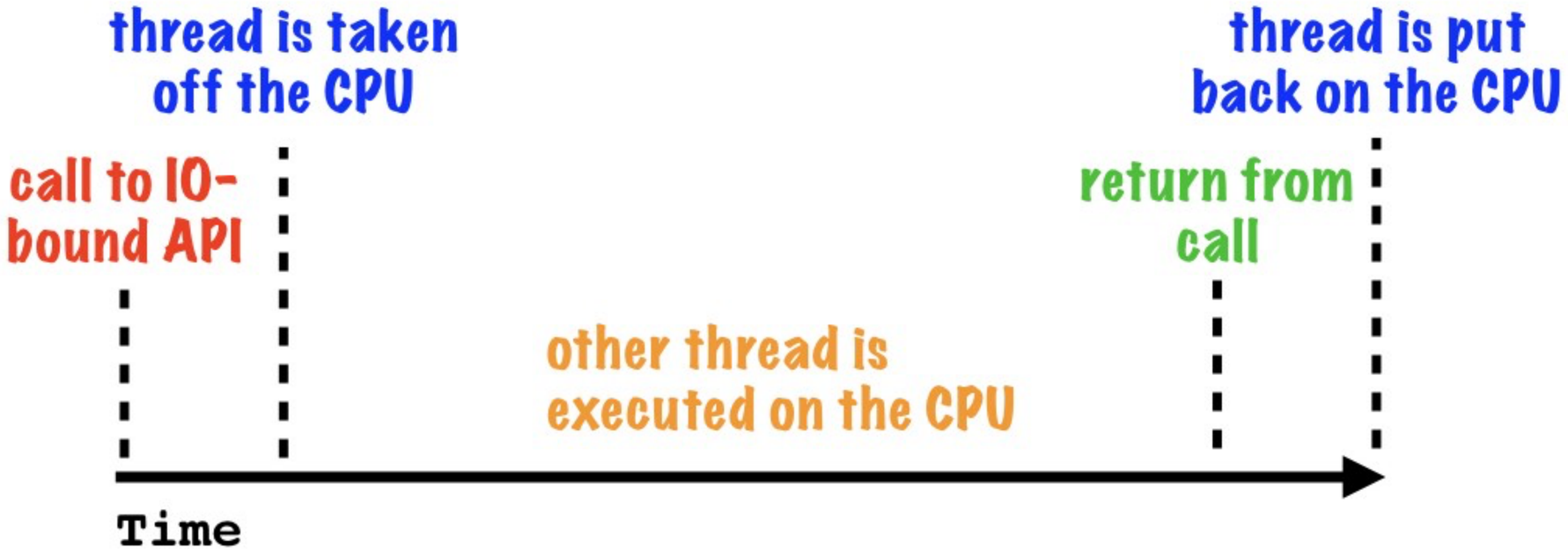


# Blocking I/O의 동작 방식 (Network)

## 어디서 병목이 생길까?



# I/O bounded blocking



# Non-blocking I/O

- 쓰레드 생성비용 절감
- Context-switch 비용 절감

# Node.js

- Libuv, V8 엔진을 가지고 있는 javascript framework.
- 빠르게 비동기 프로그래밍을 할 수 있도록, non-blocking I/O를 지원한 프레임워크.

# 이벤트루프

## Non-blocking에서 I/O 요청을 통지받는 방법 중 한가지 (Node.js)

- I/O 요청을 통지받는 무한 루프 코드
- while(1) {
  - checkI/O()
- }

# 이벤트루프

## Non-blocking에서 I/O 요청을 통지받는 방법 중 한가지 (Node.js)

- 무한 루프를 돈다는것? (Spin lock) => 좋은 성능을 담보받기 어렵다.
- 최적화 전략.
  - epoll (kqueue in BSD) : I/O 통지를 할 모델들만 모아서 (Interest) 체크하는 커널 API. 요청을 받고있는 숫자만 체크하면 되는 것이라 O(이벤트 숫자) 만 체크하면 된다. 딱 상주하고 있는 이벤트만 체크하면 되기 때문에 event loop의 시간복잡도가 준다.
    - epoll은 기존 select나 poll방식에 비해서 이벤트가 들어왔을 때 한번만 trigger(통지) 되도록 할 수 있는 옵션이 있기 때문에 (edge triggered mode), 이런 점에서도 루프를 도는 횟수를 줄일 수 있다.

# 이벤트루프

## Non-blocking에서 I/O 요청을 통지받는 방법 중 한가지 (Node.js)

```
while(true) {
    // check if one of the sockets in your "interest list" has ready data.
    // This call will block if there is no ready data.
    int amount_of_new_events = kqueue(events_to_check, new_events, ...)

    // When there is ready data in at least one file descriptor, execution continues
    // and all the events in `new_events` is checked.
    for (int i = 0; i < amount_of_new_events; i++) {
        Event event = new_events[i]

        // process event
    }
}
```

# Node.js가 고성능 프레임워크이다?

- Node.js는 고성능 프레임워크가 아니다!
- 과거 버전에 비해서 성능이 많이 개선되었다. (https://benchmarksgame-team.pages.debian.net/benchmarksgame/fastest/javascript.html?fbclid=IwAR1iBV6dvUYD3cO96DkitFUrXq56ZiJ9LcAQUwRnN4Og7B_sMeoh1ACzS5w)
- CPU-bounded Blocking (CPU intensive 한 작업)을 하기에는 무리가 있다.
- 최근에는 Clustering같은 API가 추가되면서, 멀티 프로세스기능을 할 수 있도록 하였다.
- node.js 같은 싱글쓰레드 기준 아키텍쳐를 채용하였을 경우 맥락전환 (context-switching)의 비용이 없기 때문에 I/O가 많은 서버에서는 유리하다.
- 스타트업에서는 하나의 코드 베이스로 여러개를 쓸 수 있지만, 결국 코드양이 늘어나면 디버깅이 헬이된다. (JS자체의 문제..)