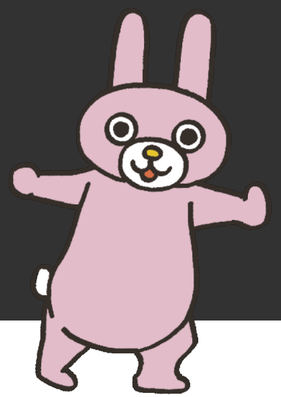
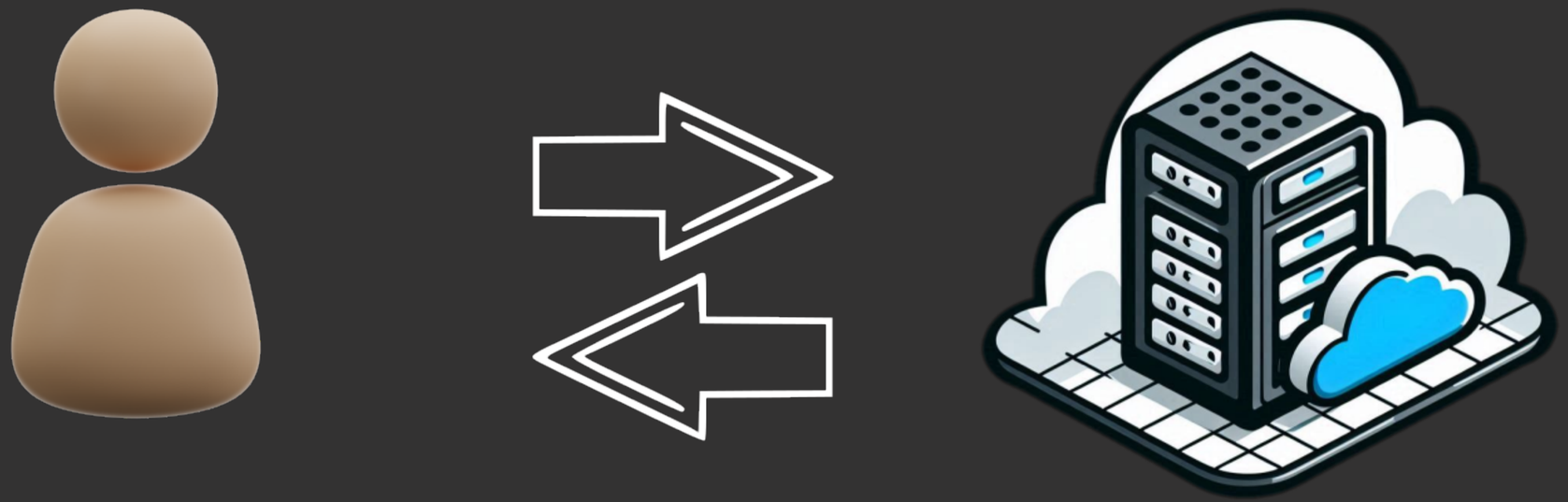
HTTP에서 실시간 통신 구현하기



발표자: 한지원

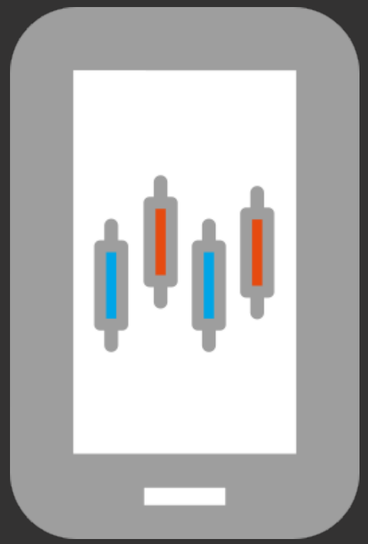
HTTP의 비연결성



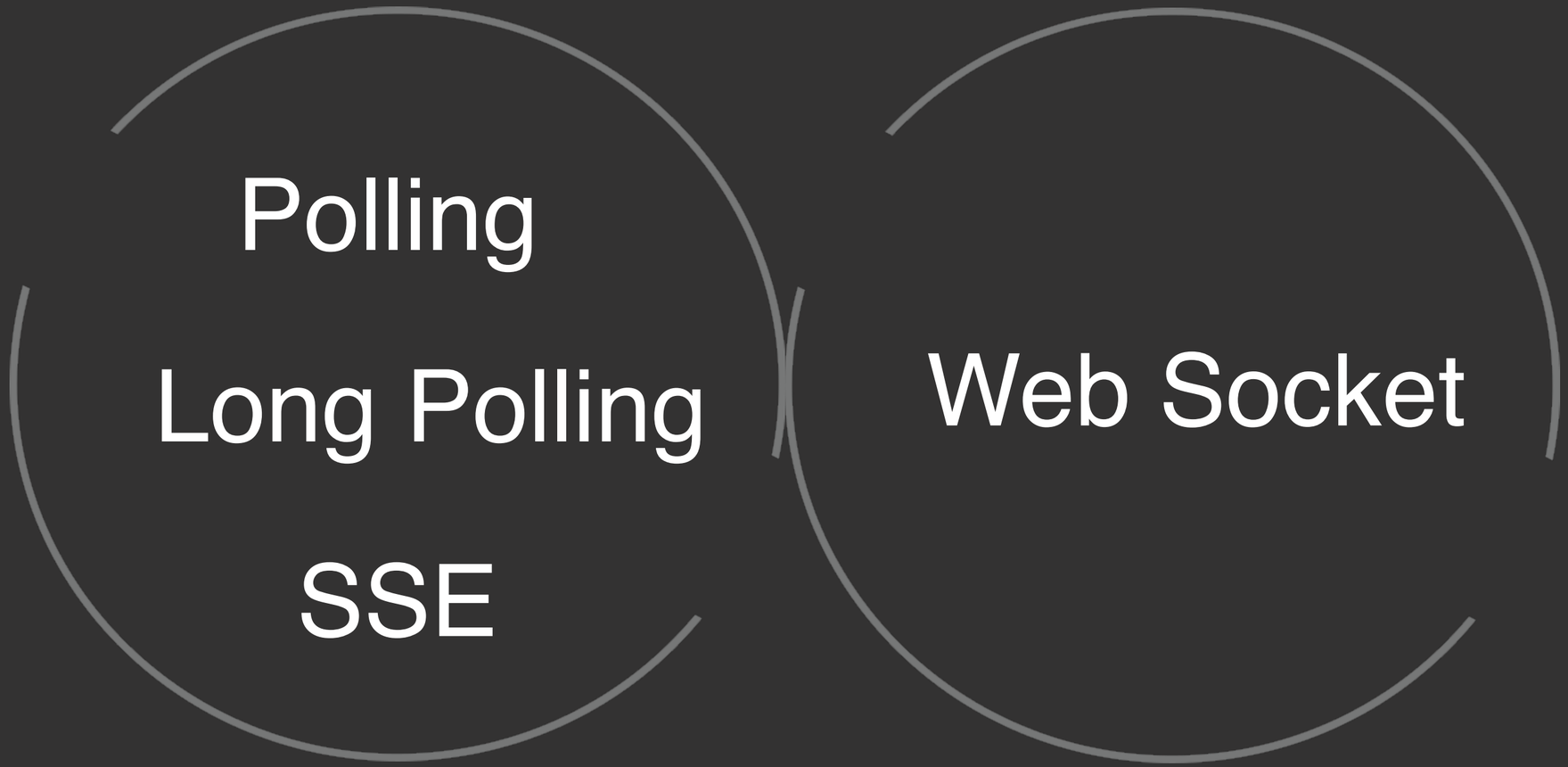
하나의 요청, 하나의 응답

알림 기능

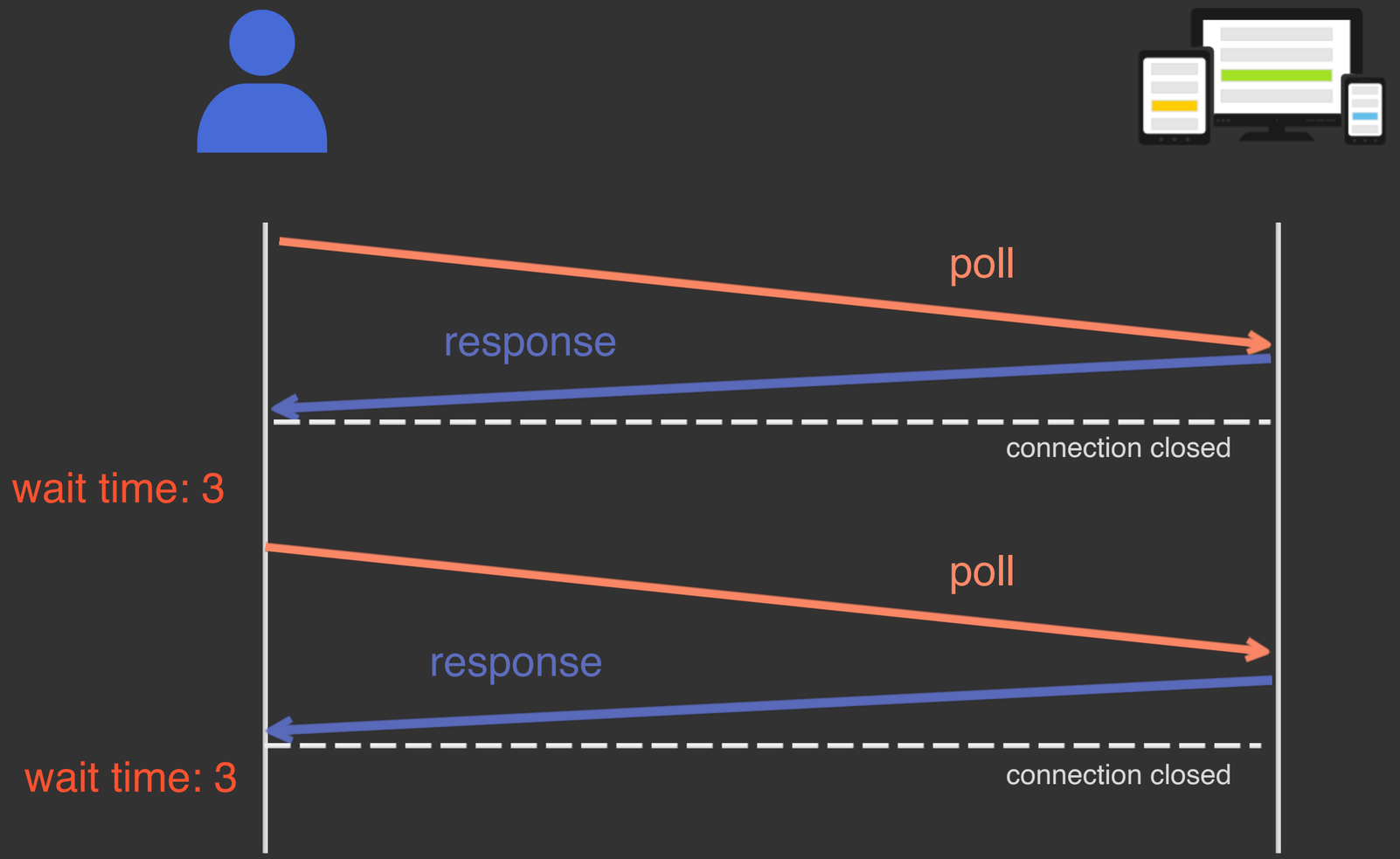
채팅 앱, 주문 상태 알림, 실시간 주식 거래



HTTP의 비연결성 극복



Polling


poll
response
connection closed
wait time: 3
poll
response
wait time: 3
connection closed

Polling

Good 

쉬운 구현: 클라이언트에서 주기적으로 요청

적용 범위: 모든 서버, 클라이언트에서 작동

- 실시간 랭킹 조회 시
- 서버의 상태 측정 시

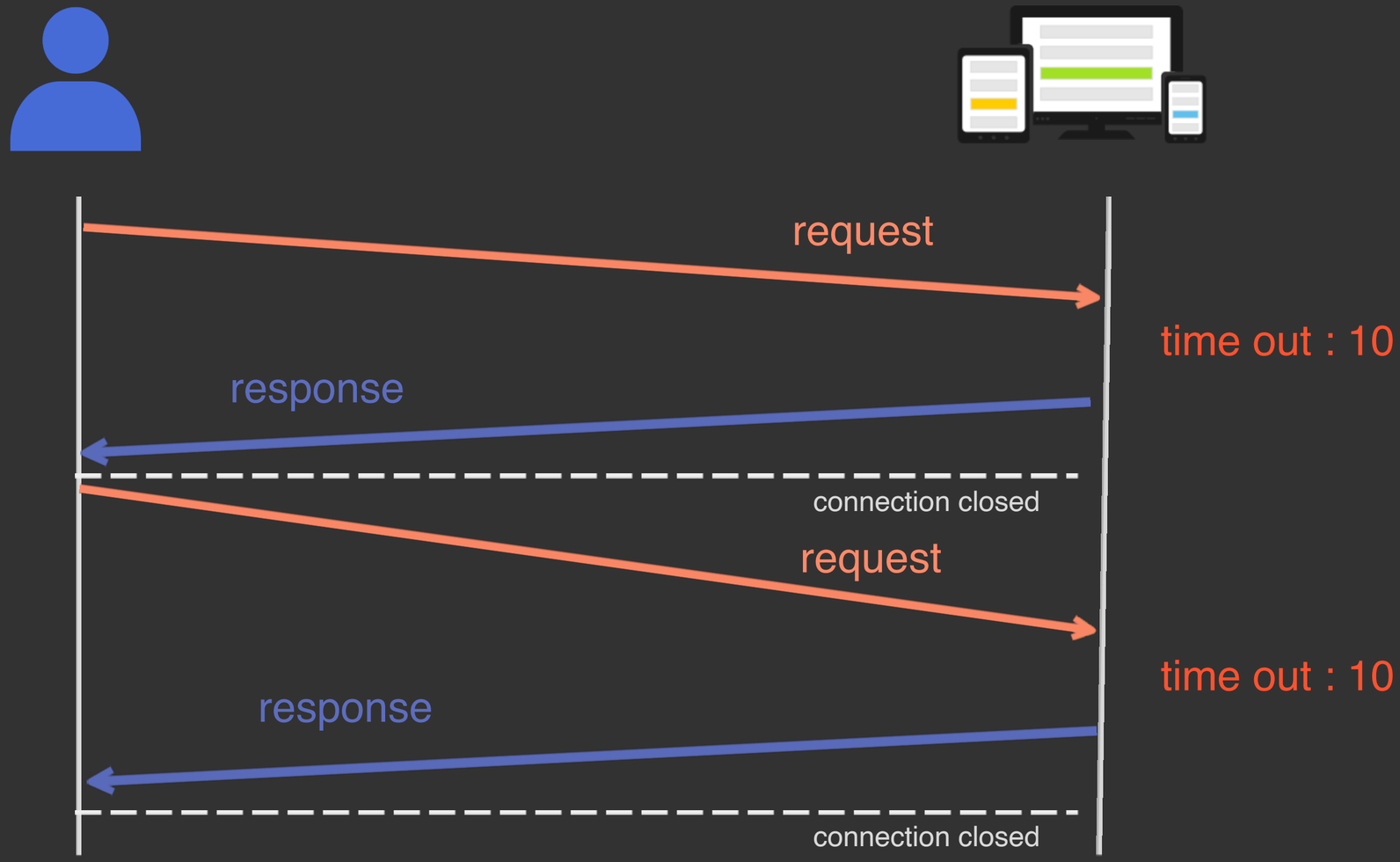
Bad 

Network Load

Connection Overload

실시간이라고 보기 어려움

Long Polling



Long Polling

Good 

Less Network Load

적용 범위: 모든 서버, 클라이언트에서 작동

Bad 

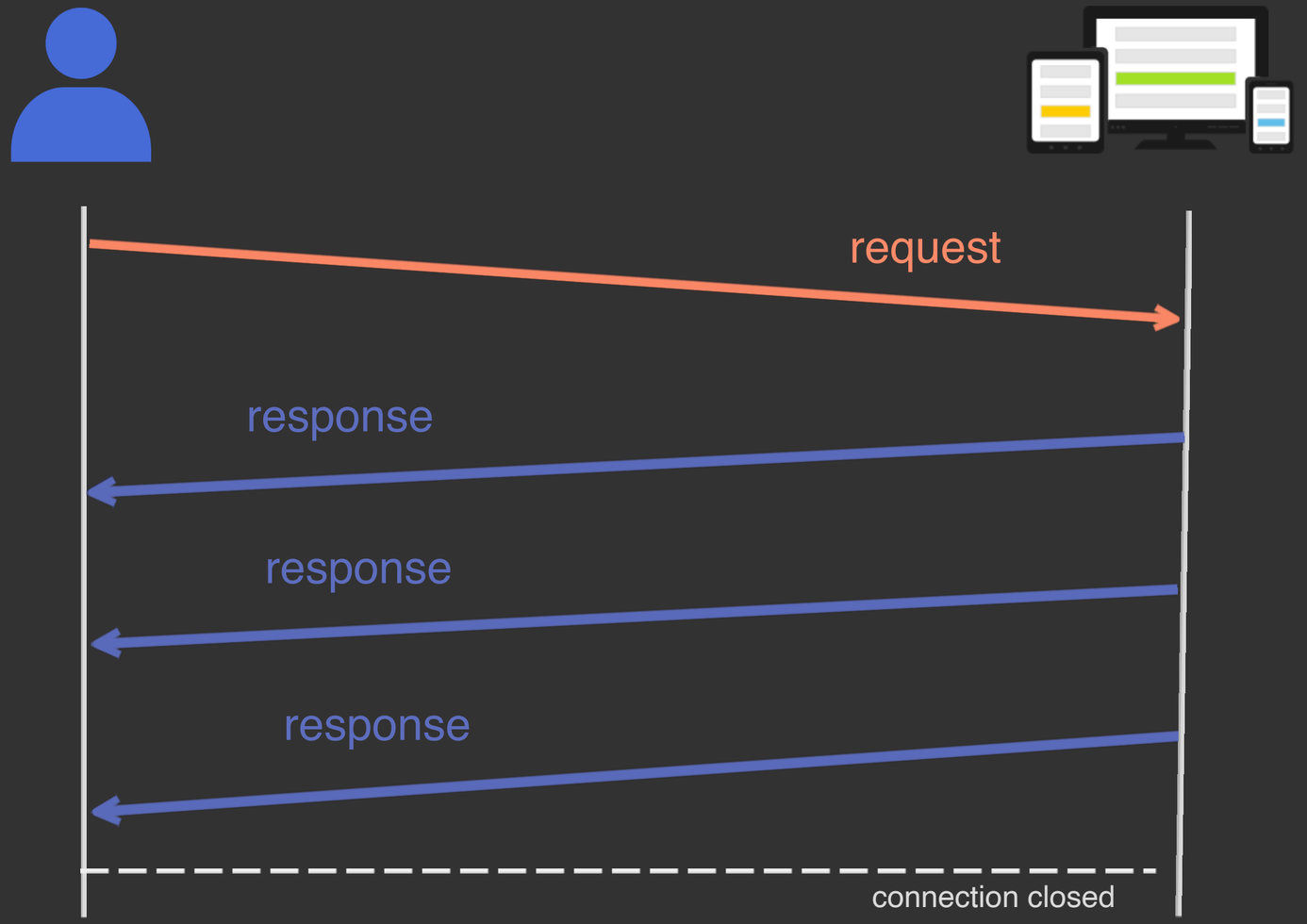
동시 연결 증가 시, Connection Overload

Polling 방식보다 구현 난이도 높음

여전히 실시간이라고 보기 어려움

- 즉각적인 응답이 요구되지 않는 실시간성 작업
- 전송 빈도가 그렇게 높지 않은 알림

Sever Sent Event



Sever Sent Event

Good 

높은 실시간성

Less Connection Overload

Bad 

한 방향으로만 데이터 전달 가능

비교적 높은 구현 난이도

지원하지 않는 브라우저도 있음

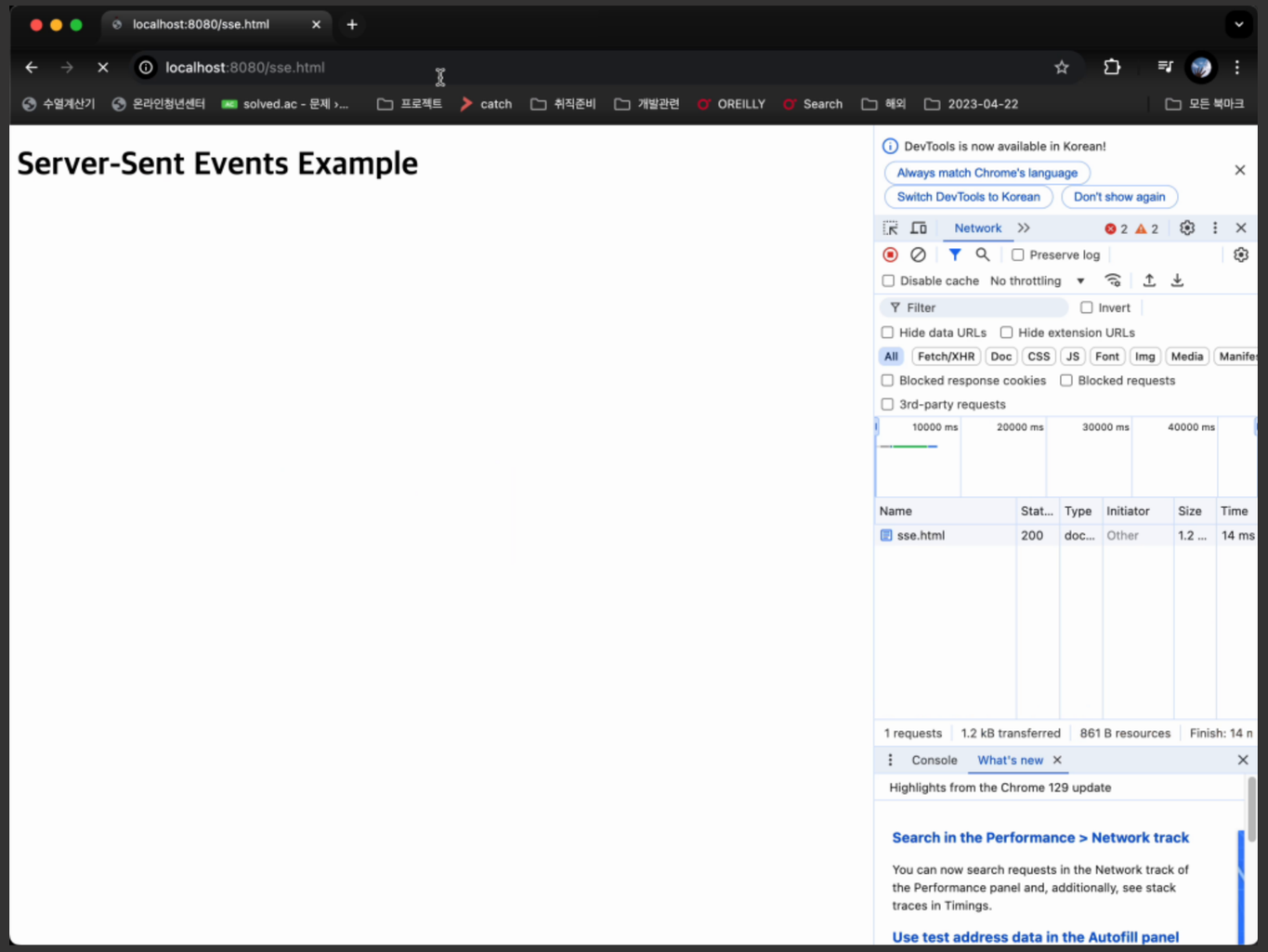
- 실시간 알림
- 실시간 주식 차트 업데이트

Sever Sent Event

```java
@RestController new *
public class SseController {

    @GetMapping("/sse") new *
    public SseEmitter streamSseEvents(){
        SseEmitter sseEmitter = new SseEmitter();

        Executors.newSingleThreadExecutor().execute(() ->{
            try {
                for (int i = 0; i < 5; i++) {
                    sseEmitter.send( object: i + " 번째 SSE 메시지 받아라! ");
                    TimeUnit.SECONDS.sleep( timeout: 1);
                }
            } catch (IOException | InterruptedException e) {
                throw new RuntimeException(e);
            }
        });
        return sseEmitter;
    }
}
```

```html
<script>
    // SSE 연결
    let eventSource;

    window.onload = function() {
        // EventSource가 이미 있으면 다시 생성하지 않음
        if (!eventSource) {
            eventSource = new EventSource("/sse");

            eventSource.onmessage = function(event) {
                const newElement = document.createElement("div");
                newElement.textContent = "Received message: " + event.data;
                document.getElementById("events").appendChild(newElement);
            };

            eventSource.onerror = function(event) {
                console.error("Error occurred:", event);
            };
        }
    };

</script>
```

Sever Sent Event



Spring은 어떻게 비동기 처리를?

스프링은 ThreadPoolExecutor와 같은 스레드 풀을 사용하여 스레드를 관리

하나의 요청은 스프링 Thread Pool 내 하나의 thread가 할당되어 처리된다.

하나의 요청에서 별도의 thread를 사용하는 비동기 작업이 어떻게 가능한걸까?

Spring은 어떻게 비동기 처리를?

하나의 요청에서 별도의 thread를 사용하는 비동기 작업이 어떻게 가능한걸까?

Thread Pool이 다른 스레드에 작업을 위임

Spring 비동기 처리

1.Client sends a request

2. Servlet container allocates a thread and invokes a servlet in it.

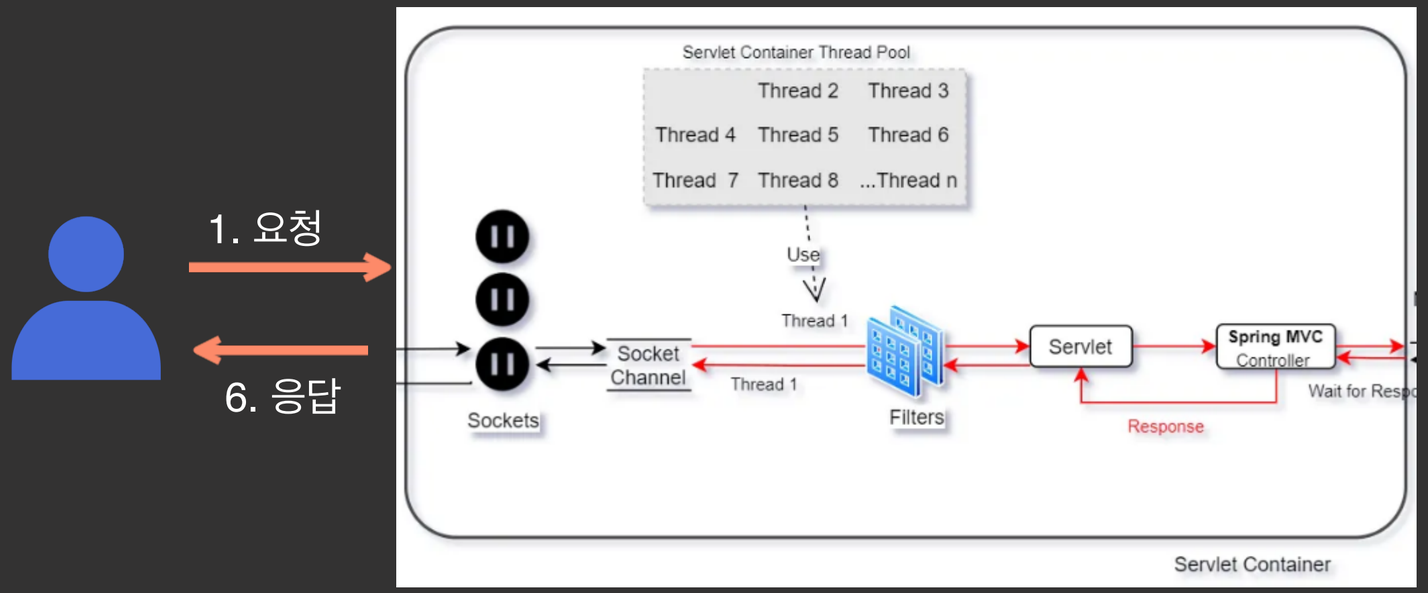
3. The servlet calls request.startAsync(), saves the AsyncContext, and returns

4. The container thread is exited all the way but the response remains open

5. Some other thread uses the saved AsyncContext to complete the responses

6. Client receives the response

Spring Container의 비동기 처리 과정



이미지 출처: Medium -Spring Webflux: EventLoop vs Thread per Request Model

2. dispather servlet이 request.startAsync() 호출
   & AsyncContext 저장 비동기 처리 시작!

3. 요청 받은 Thread 1은 스레드 풀에 스레드 반환

4. 다른 Thread 2가 비동기 작업을 처리하고 응답

5. 응답 후 Dispathcher Servlet이 사용자에게 전달

DeferredResult & Callable

- 비동기 컨트롤러 호출 시 동작 과정 -

> - `DeferredResult` and `Callable` return values in controller methods provide basic support for a single asynchronous return value.

- DeferredResult: 외부 시스템의 이벤트 대기하고 결과를 받아야 하는 경우

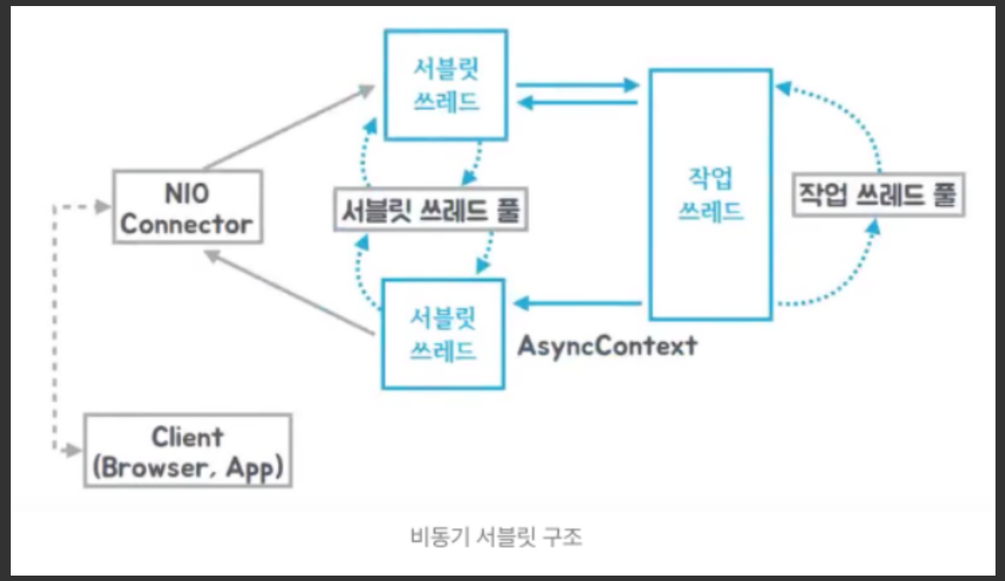
> The controller can produce the return value asynchronously, from a different thread—for example, in response to an external event (JMS message), a scheduled task, or other event.

- Callable: 작업 완료 시 바로 결과 반환. 긴 시간 동안 처리해야하는 작업에 유용

> The return value can then be obtained by running the given task through the configured `AsyncTaskExecutor`.

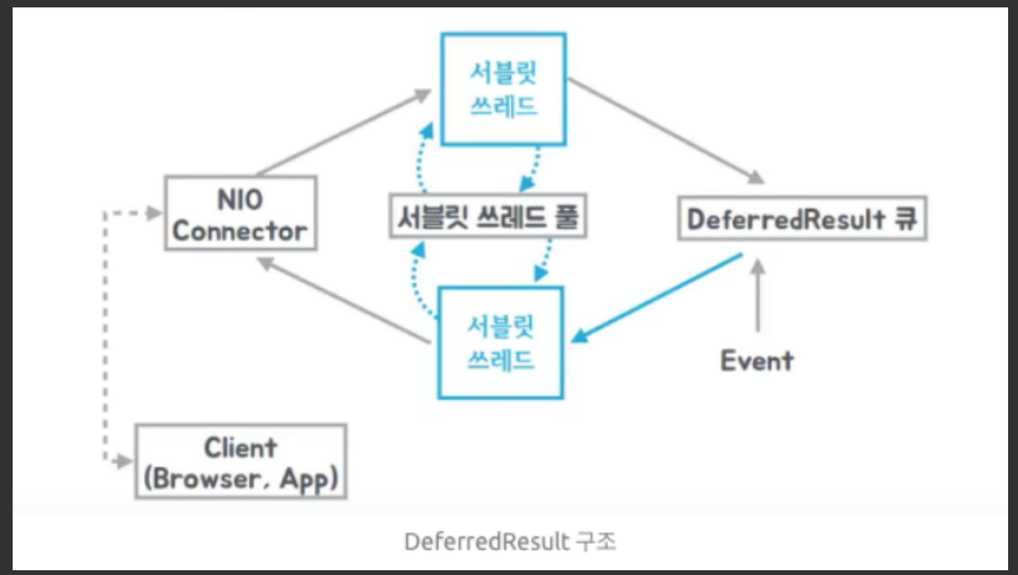
Callable vs DeferredResult

- Callable



비동기 서블릿 구조

- DeferredResult



DeferredResult 구조

이미지 출처: joinmin92.github.io

Review

HTTP의 비연결성을 극복하는 기술

Polling, Long Polling, SSE, Web Socket

Polling vs Long Polling vs SSE

Polling: 각 요청마다 서버가 바로 응답. 매번 커넥션 생성 및 해제

Long Polling: 각 요청마다 작업이 완료되면 응답. 매 요청 시 커넥션 생성 및 해제

SSE: 한 번의 요청. 커넥션을 끊지 않고 스트림 형태로 유지

Review

Spring의 비동기 처리 기술

Thread Pool을 사용해 여러 스레드로 비동기 작업을 구현, 높은 재사용성

Callable vs DeferredResult

Callable: 오랜 시간 걸리는 작업 후 즉시 응답

DeferredResult: 외부 시스템, 스케쥴링 큐 등을 통해 지연된 응답

References

- Spring 공식 문서- 비동기 처리
- Medium - 스프링 컨테이너 구조 사진
- Spring 공식 문서- SevelT3 비동기
- Polling, Long Polling, Stream 방식 비교

The End ...