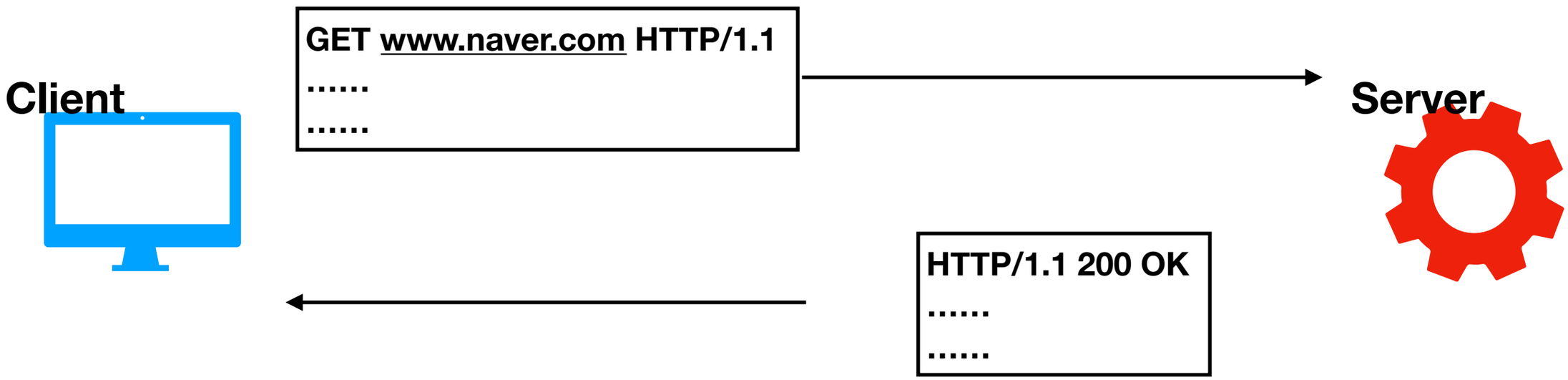
Event loop를 이용한 Webserver 구현

목차

- Webserver
- 전통적인 Webserver의 Architecture
- Event loop architecture
- Event loop architecture를 적용한 Webserver
- 제안하는 과제

Webserver

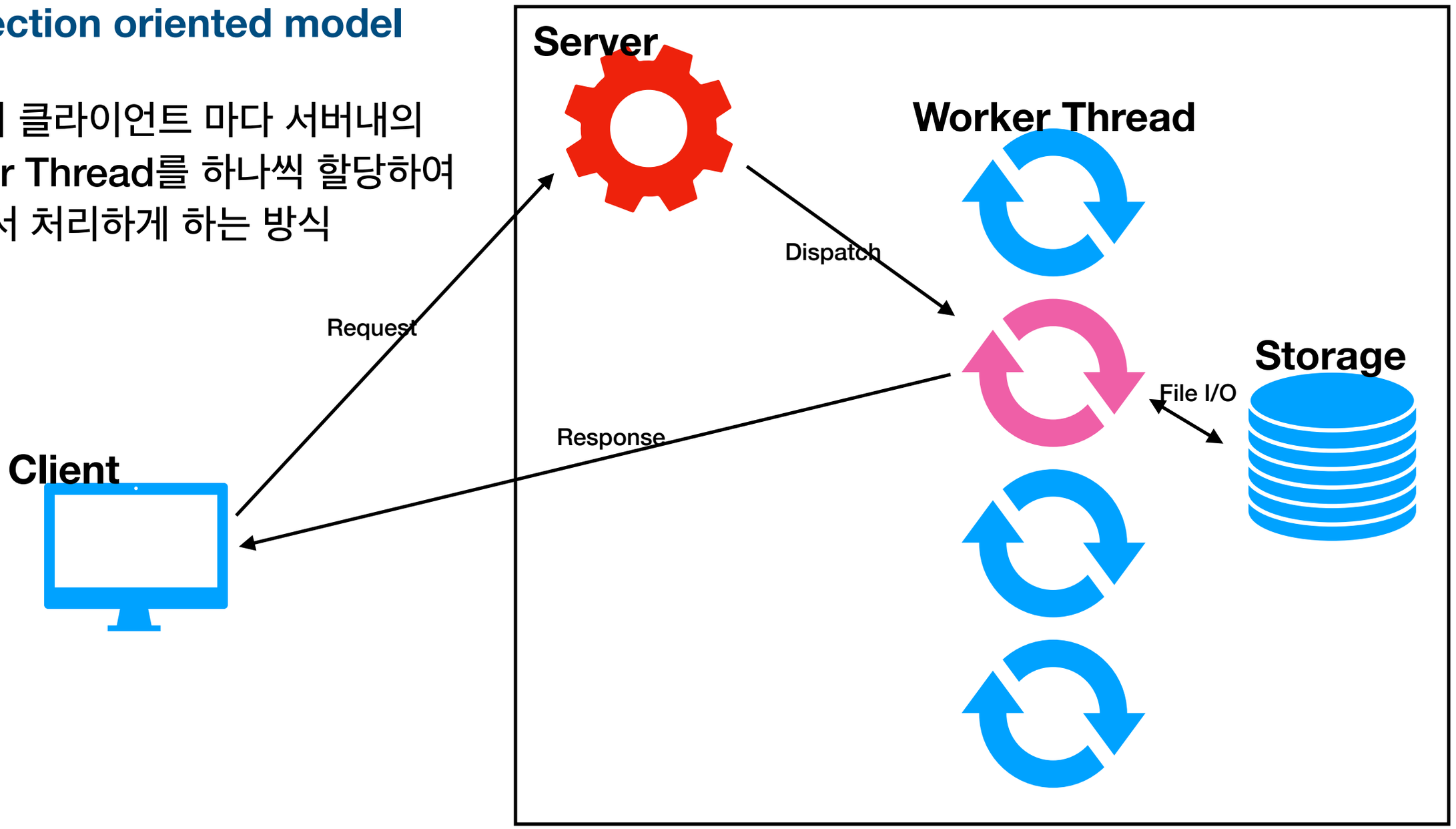
- HTTP(Hyper Text Transfer Protocol)로 된 요청에 대해서 응답을 내보내는 서버



전통적인 Webserver의 Architecture

Connection oriented model

각각의 클라이언트 마다 서버내의 **Worker Thread**를 하나씩 할당하여 전담해서 처리하게 하는 방식



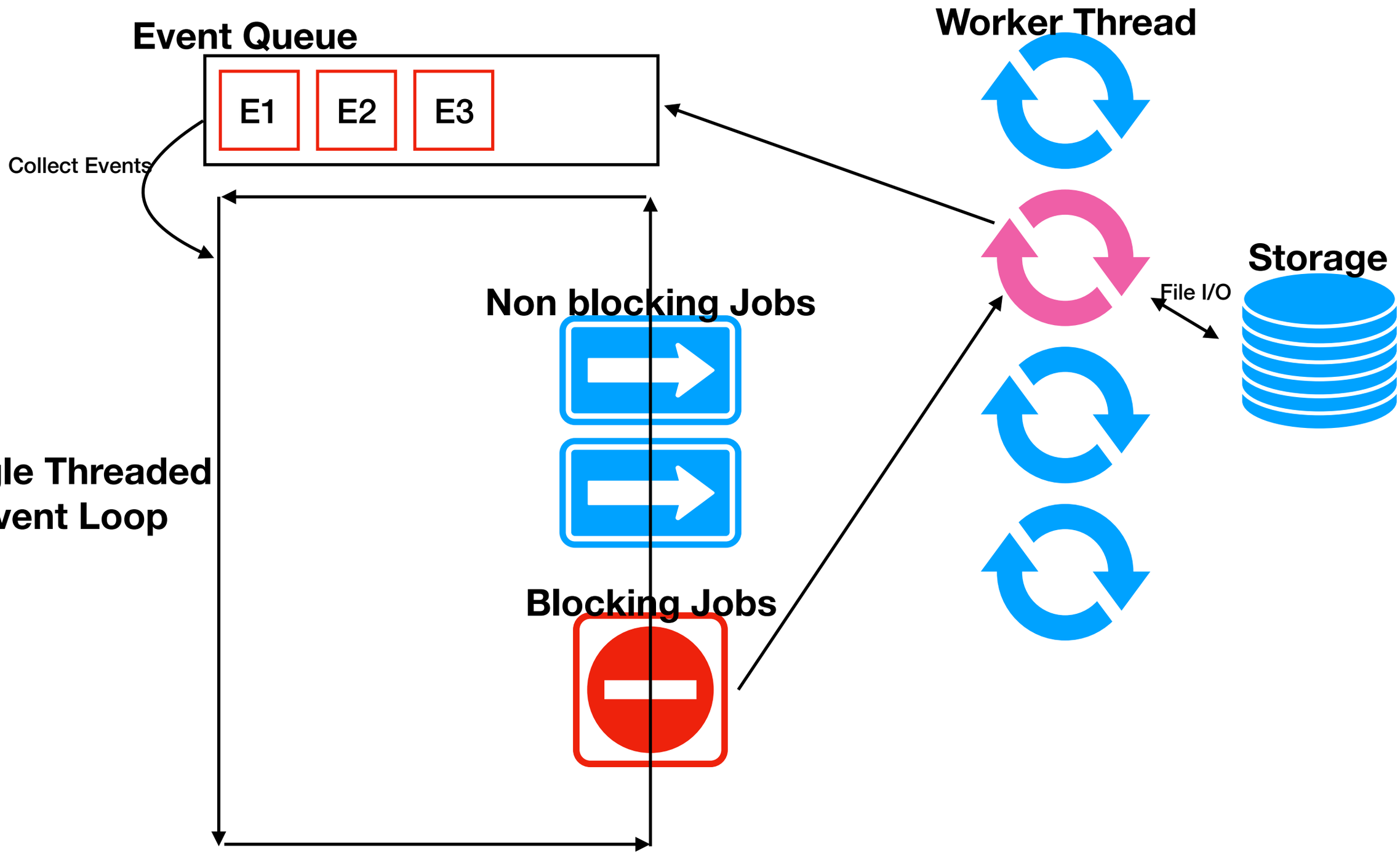
전통적인 Webserver의 Architecture (cont’)

- Worker Thread의 갯수가 곧 동시 처리 가능한 클라이언트의 갯수를 나타내게 됨

- Worker Thread를 무한정 늘릴 수는 없기 때문에 C10K 문제 등이 제기됨

- C10K : Concurrently handling ten thousand connections의 약어로 대량의 클라이언트를 효율적으로 빠르게 처리해야 하는 과제를 뜻함

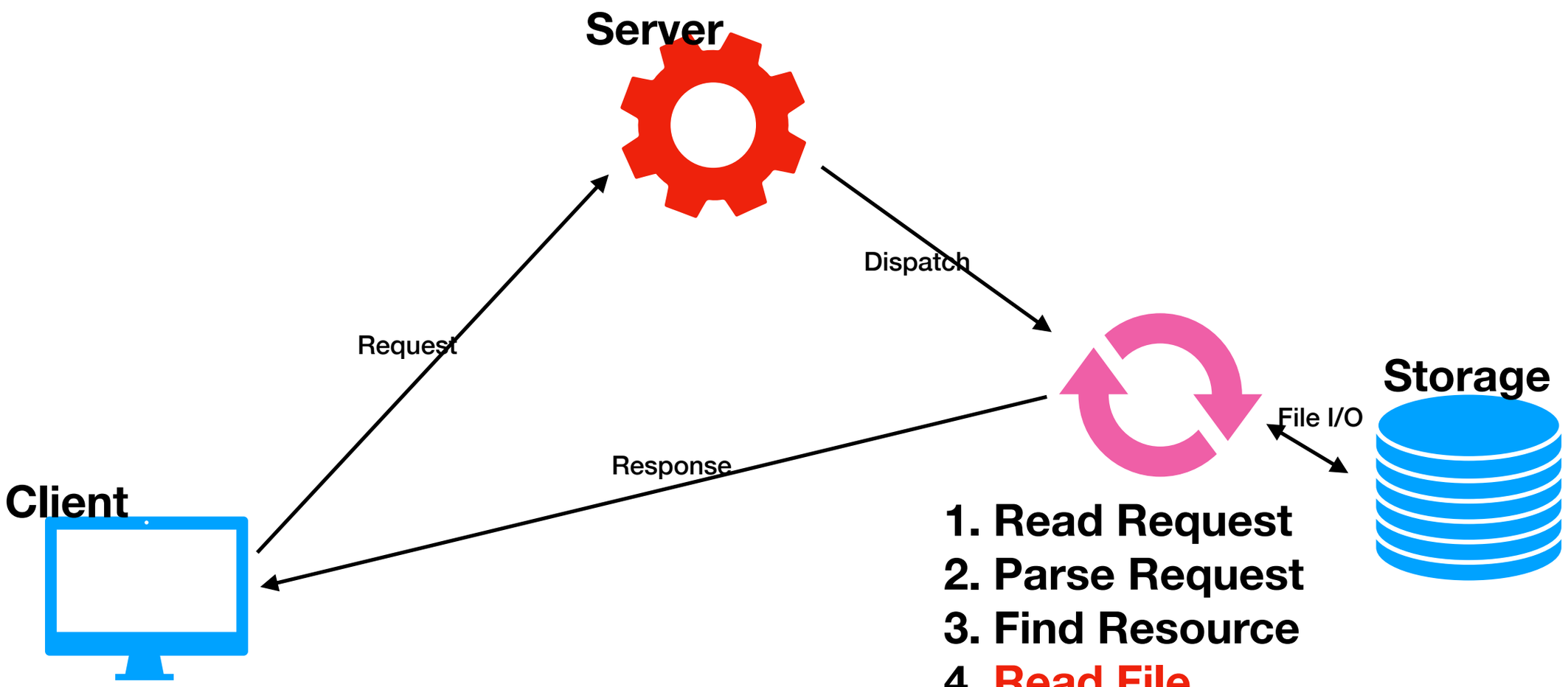
Event loop architecture



Event loop architecture

- Single Thread인 Event loop가 발생한 Event들을 Event Queue로부터 수집하여 Event에 해당하는 Job을 순서대로 수행하는 구조
  - Event의 종류는 다양하게 구성할 수 있음
  - ex) Client가 Server에 연결했다는 Event, Server가 Client의 요청을 받았다는 Event 등등
- CPU가 대기할 필요없는 Job들은 바로바로 수행되고, CPU가 대기해야하는 I/O 작업등은 별도의 Worker Thread에 시키고 넘어감
- Worker Thread가 I/O 작업을 마무리한 후 Event Queue에 ‘작업을 완료했다는 Event’를 전달하여 다시 Event loop에서 이어서 처리할 수 있도록 함
- 결국 Event loop thread는 쉬지않고 끊임없이 Job을 처리할 수 있게 되어 매우 높은 처리 효율을 보임

Event loop architecture를 적용한 Webserver



대략적인 HTTP 요청 처리 과정

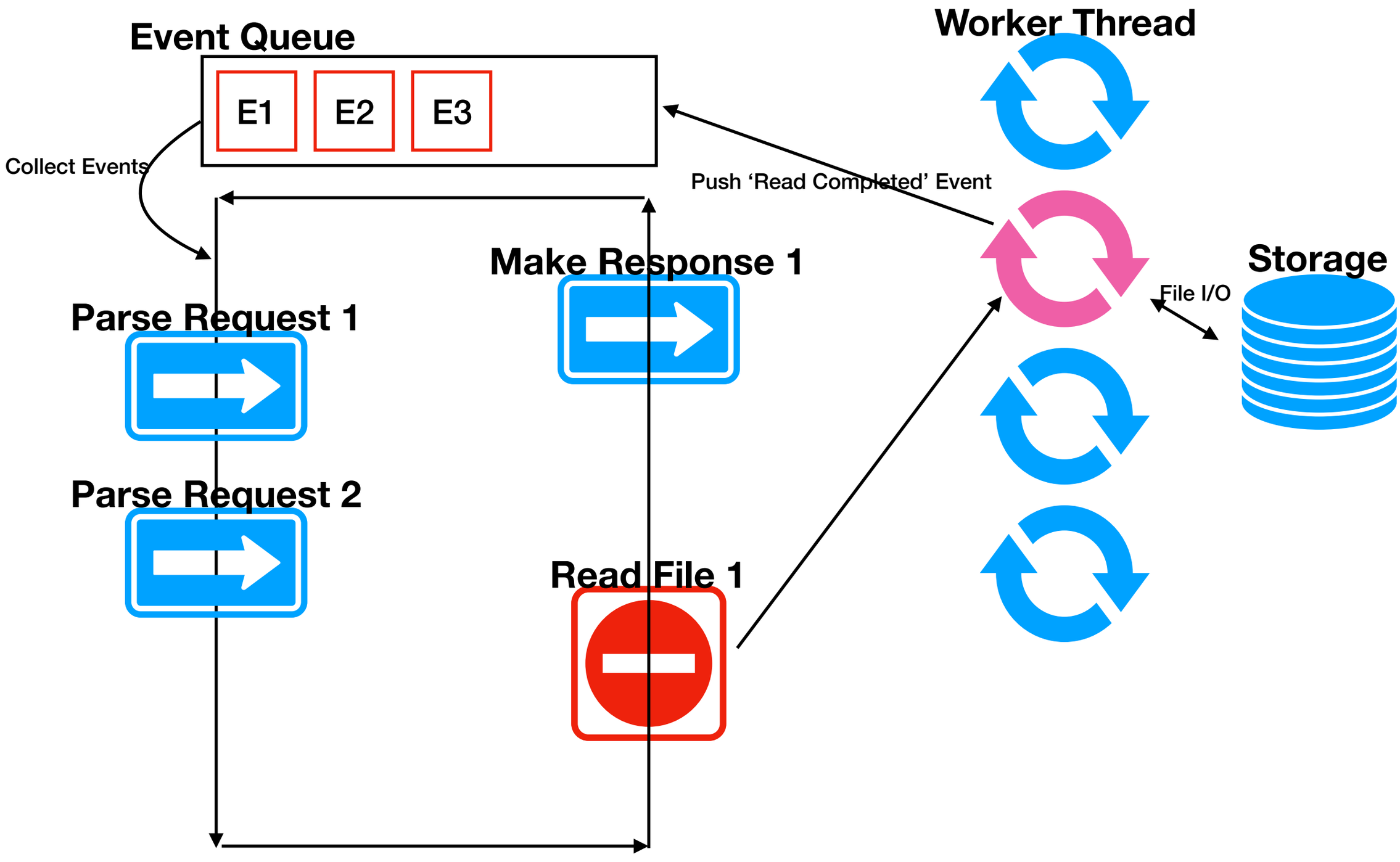
Event loop architecture를 적용한 Webserver (cont’)

- Connection oriented 모델에서는 클라이언트를 전담하는 Thread가 배정되어있기 때문에 여섯가지 Job을 단순히 순서대로 수행만 하면 됨

- Event loop 모델에서는 하나의 Event loop thread가 모든 클라이언트를 담당하기 때문에 각각의 Job은 연속으로 수행될 수 없을 수도 있으며 Event에 의해 통제되어야 함

- 특히 File I/O 등 CPU가 일을하지 못하고 대기해야 하는 종류의 Job은 반드시 별도의 Worker Thread에 처리를 위임해서 Event loop thread가 아무일도 하지 못하고 대기하는 상황을 피해야 함

Event loop architecture를 적용한 Webserver (cont')



제안하는 과제

- Event loop 모델을 적용한 Webserver 구현
  - HTTP 통신 규약에 대한 이해 및 그것을 다룰 수 있는 간략한 Parser의 구현
  - Event loop 모델에 대한 이해 및 그것을 응용한 Webserver의 구현
    - Event loop와 Worker Thread간의 Job 및 Event의 교환 메커니즘에 대한 이해 및 구현
- 기타 (Optional)
  - Cache의 구현
  - Event loop 모델을 사용한 대표적인 Webserver인 Node.js와의 성능 비교

감사합니다.