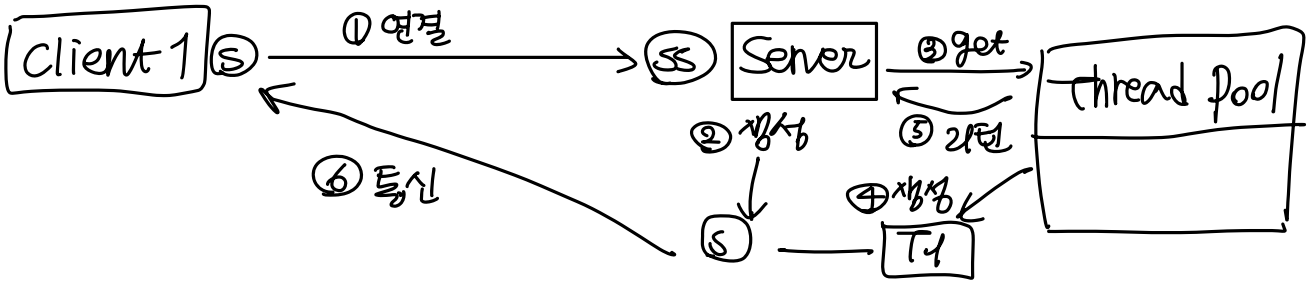
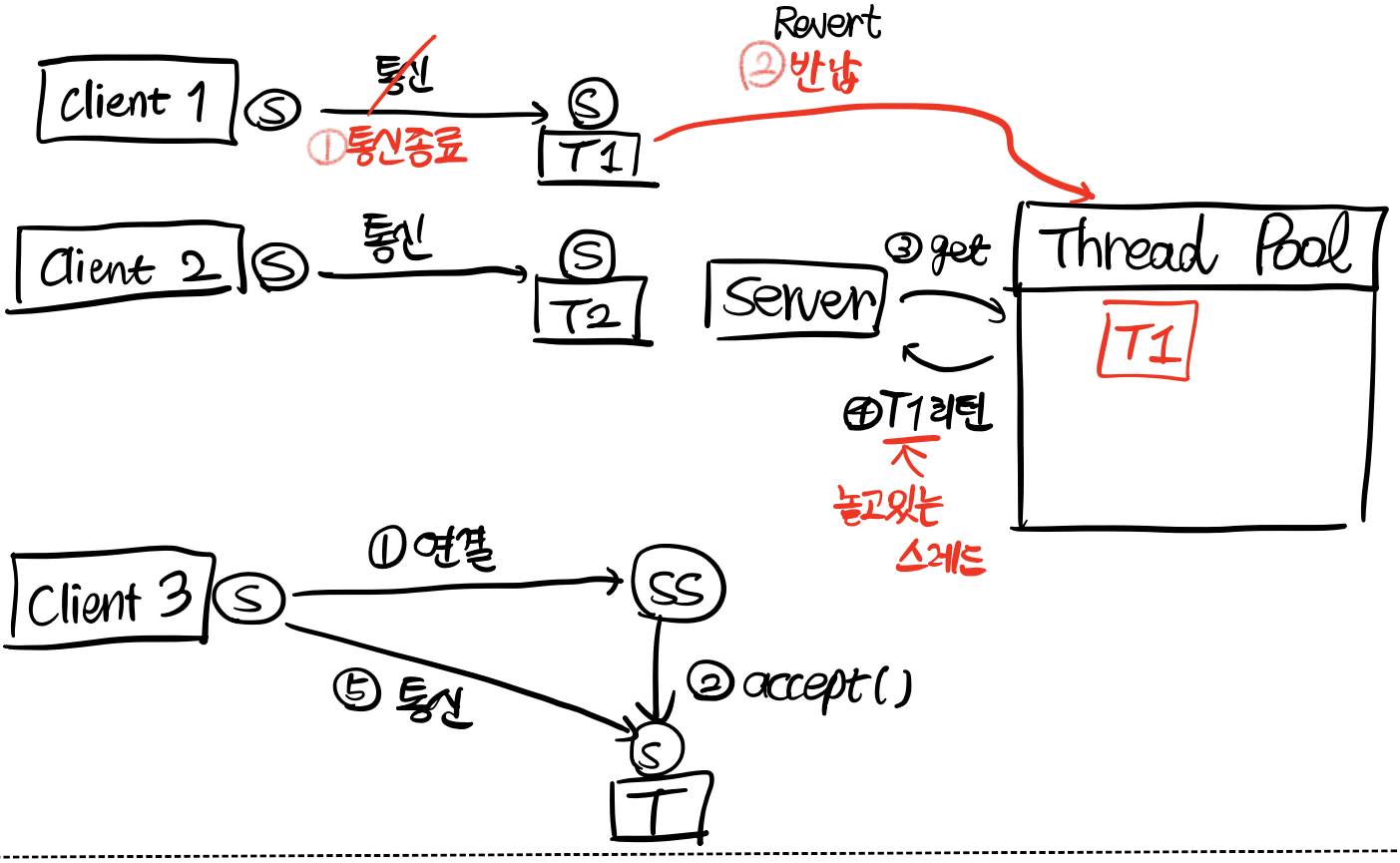
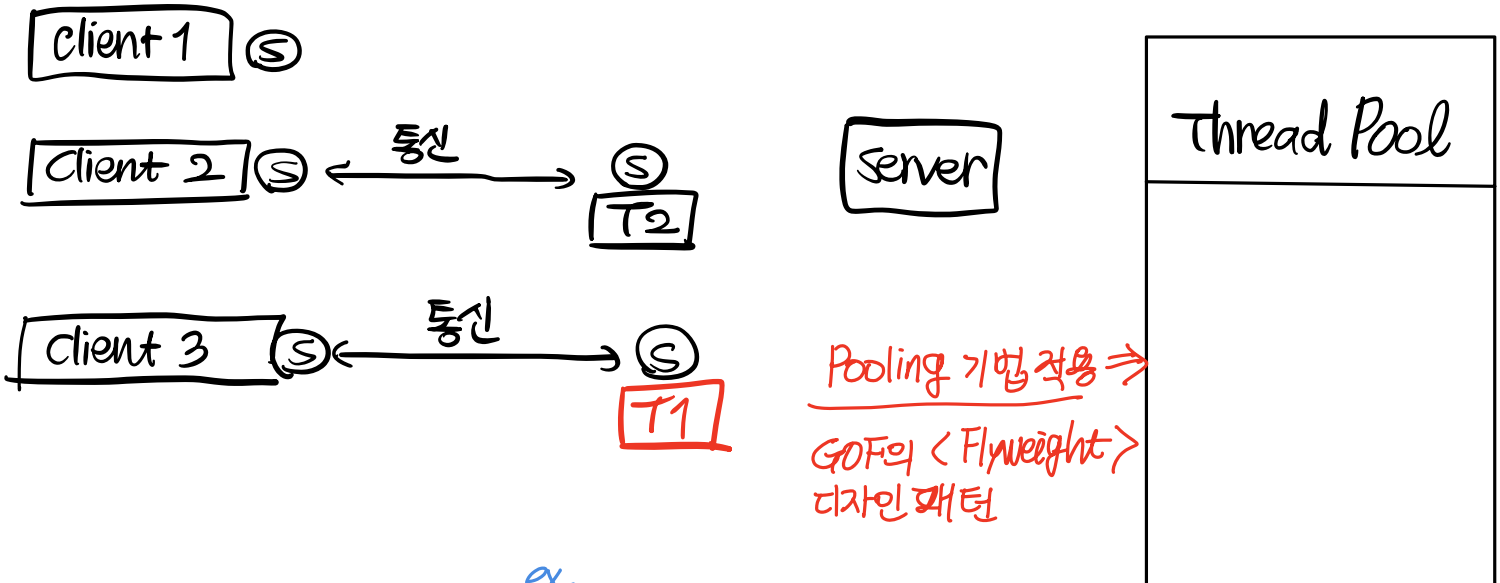
2) 개선

* Ⓢ : Socket







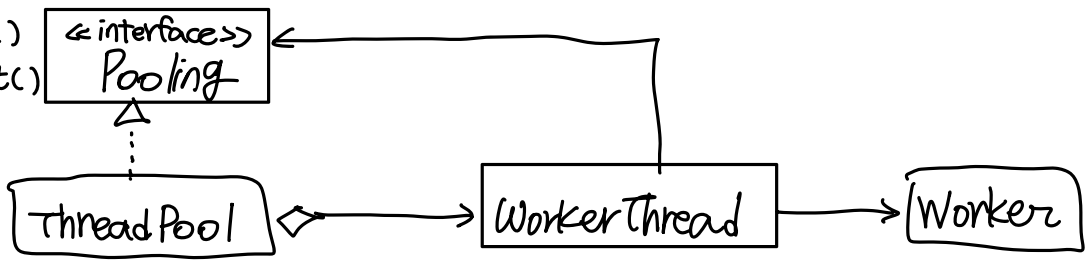
아직 정돈되지 않은 방법 ▷ ex. 동시 접속 1000명. 스레드 100개가 생성된다고 하면 일정시간동안 쓰이지 않는 스레드도 많을 것 ⟶

이들을 메모리에 남기는 것보다 가비지로 만들어 버리는 것이 필요함.
(정교한 작업)

* 객체지향에서 피해야 할 것 : 쌍방 (순환) 참조
  - 유지보수에 안좋다.

현재 코드 상태:

끊어낼 방법: ·get()
·revert()



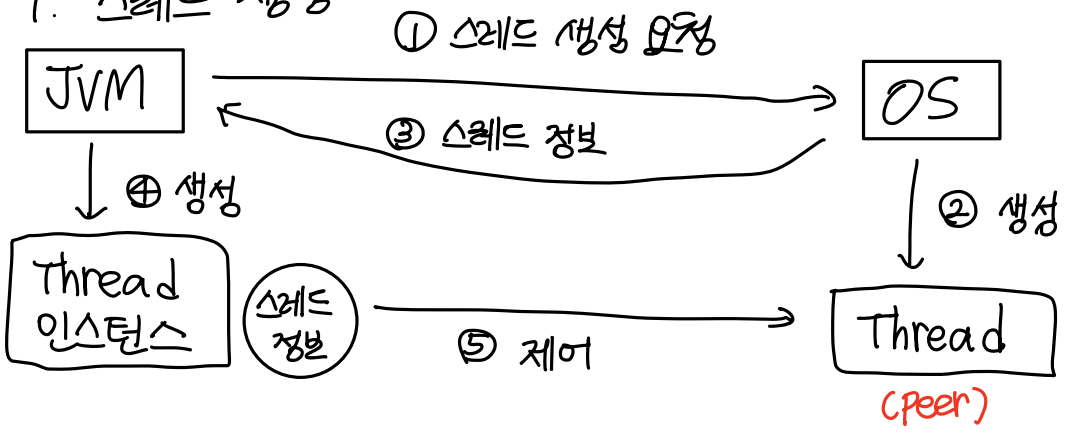
Pooling 기법 → GOF의 Flyweight 기법

* Pooling 기법 ⇒ · 객체 생성에 비용이 큰 경우 (시간 + 메모리)

객체 생성
↓
(재) 사용
↓
보관

Thread 정리

1. 스레드 생성

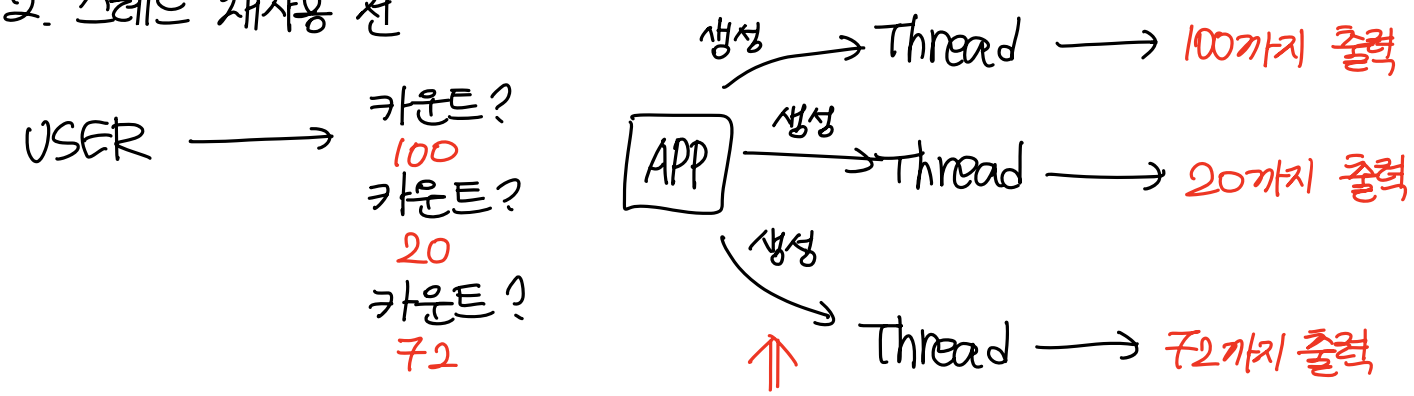


→ ① 스레드 생성에 시간 소요
② 메모리 사용

자주 스레드를 생성하고 버리면 실행시간과 메모리 효율성이 떨어진다.

개선 방안? ⟸ "Thread 재사용"

2. 스레드 재사용 전

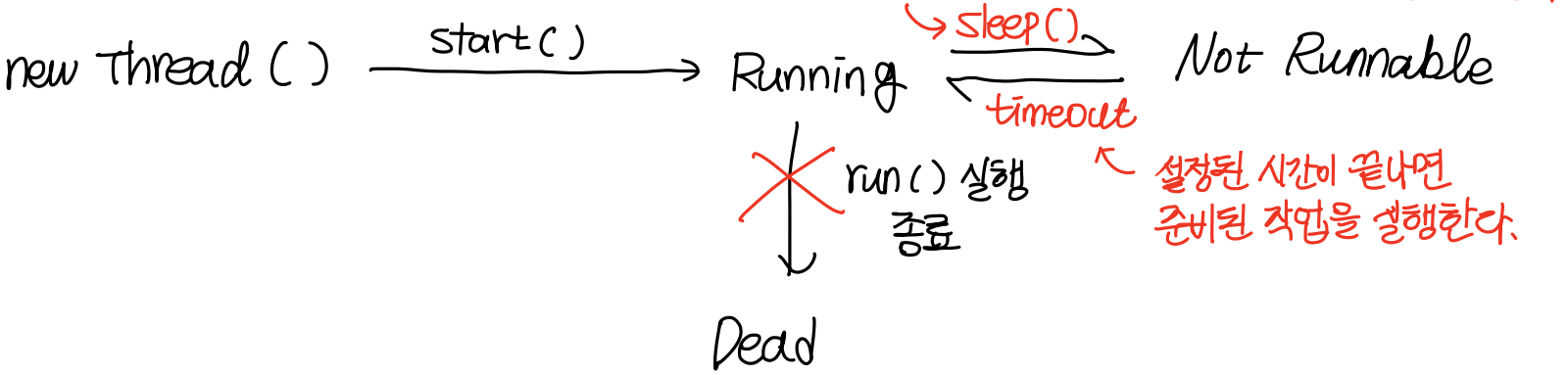


- 매번 스레드 인스턴스 생성 = OS의 스레드 생성
- 작업 종료 후 가비지 = 일정시간동안 메모리 차지

가비지 컬렉터가 메모리를 해제하기 전까지

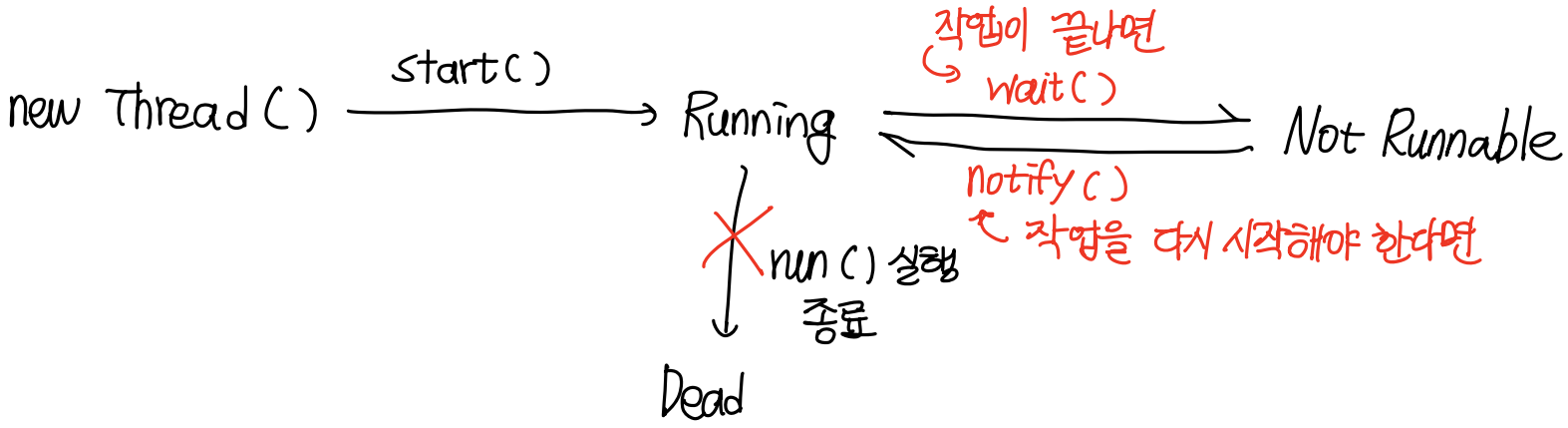
3. 스레드 재사용 방법 1

작업이 끝나면 다음작업을 준비할 수 있게 일정시간 동작을 멈춘다.



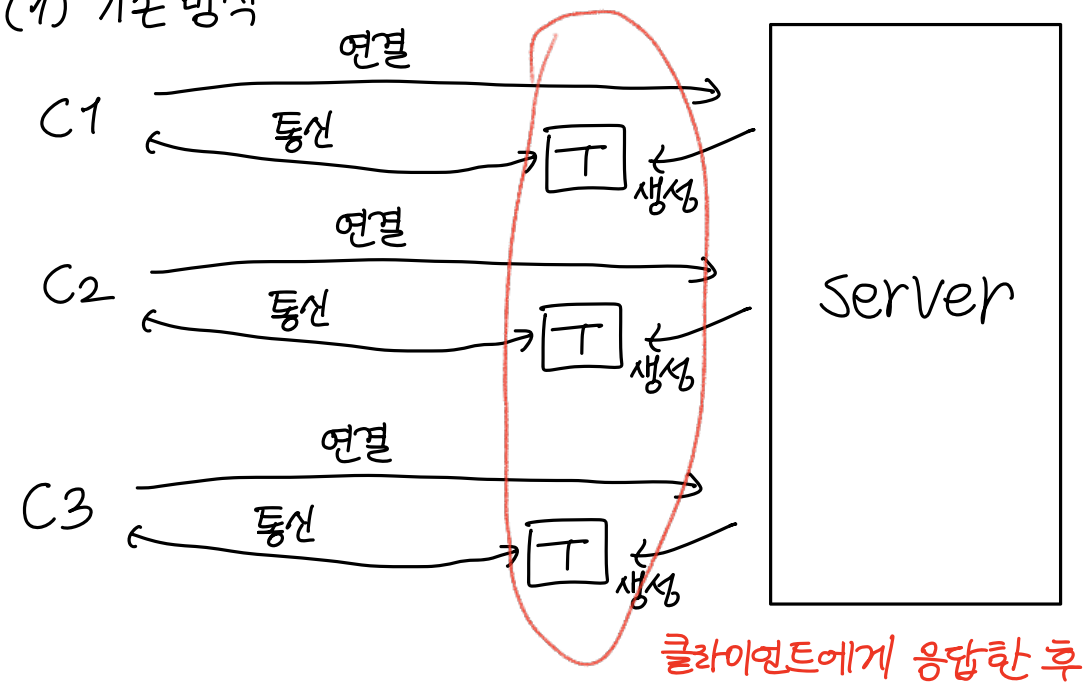
설정된 시간이 끝나면 준비된 작업을 실행한다.

4. 스레드 재사용 방법 2



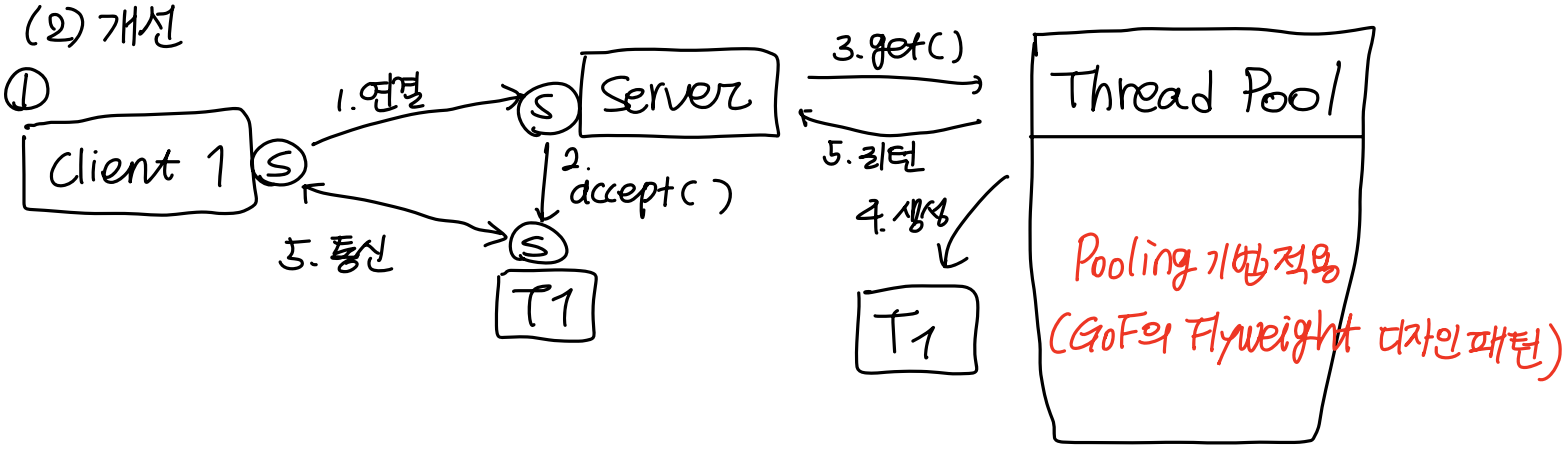
Step 45. 스레드 재사용하기

(1) 기존 방식



클라이언트에게 응답한 후 스레드는 종료된다.
= 스레드는 가비지가 된다.

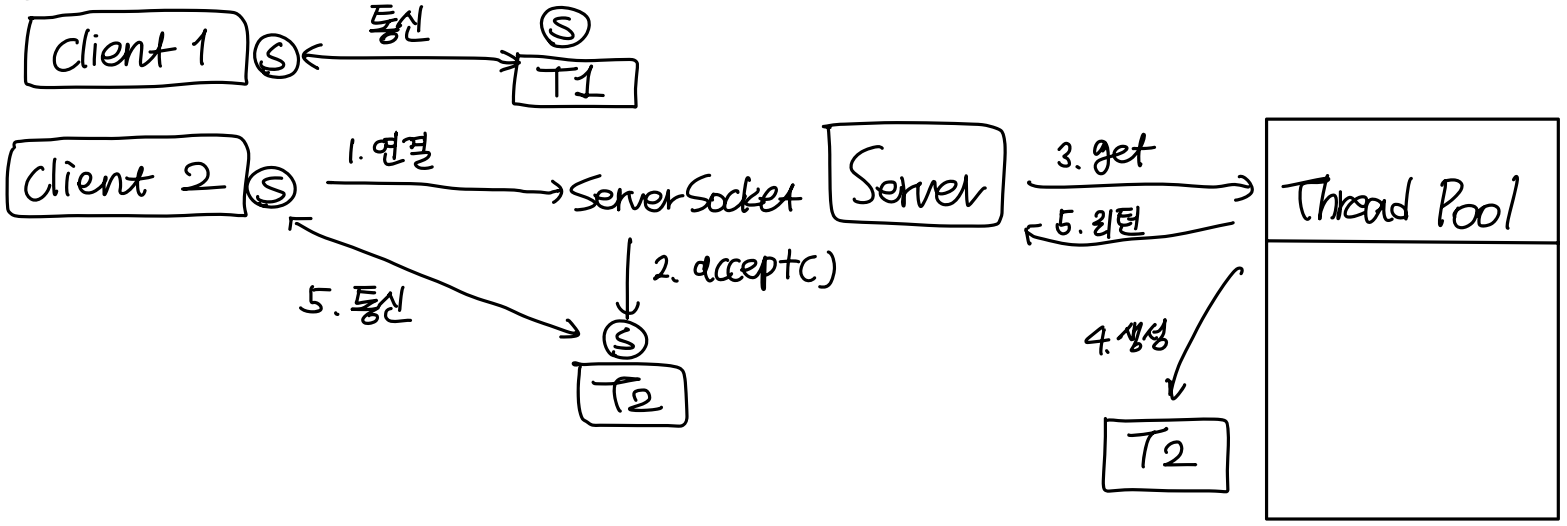
(2) 개선



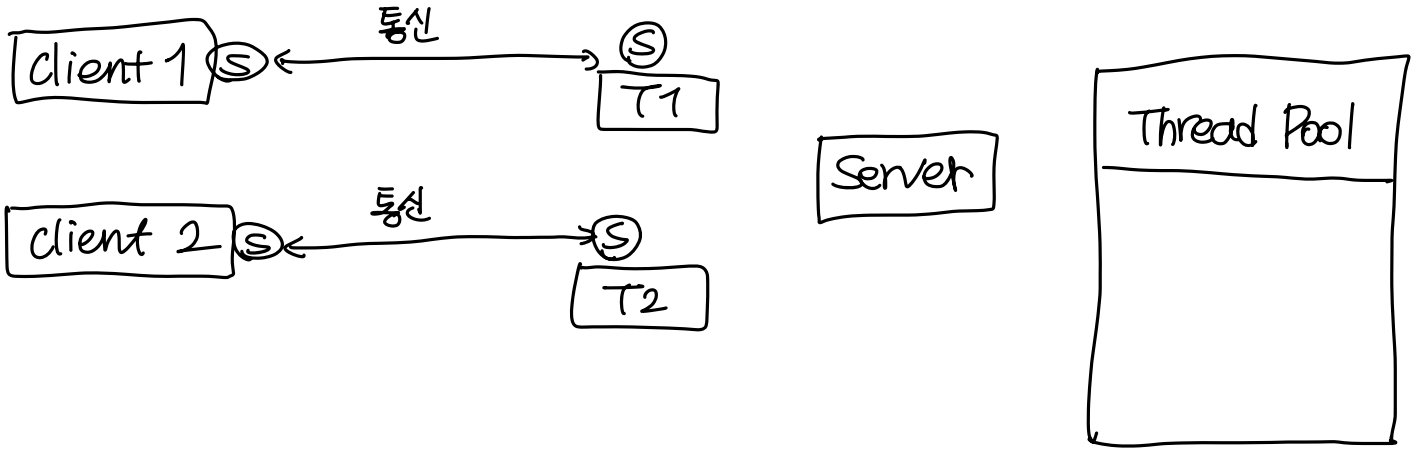
* Pooling 기법 : 객체 생성 → (재)사용 → 보관

: 객체 생성에 비용이 큰 경우
시간 + 메모리

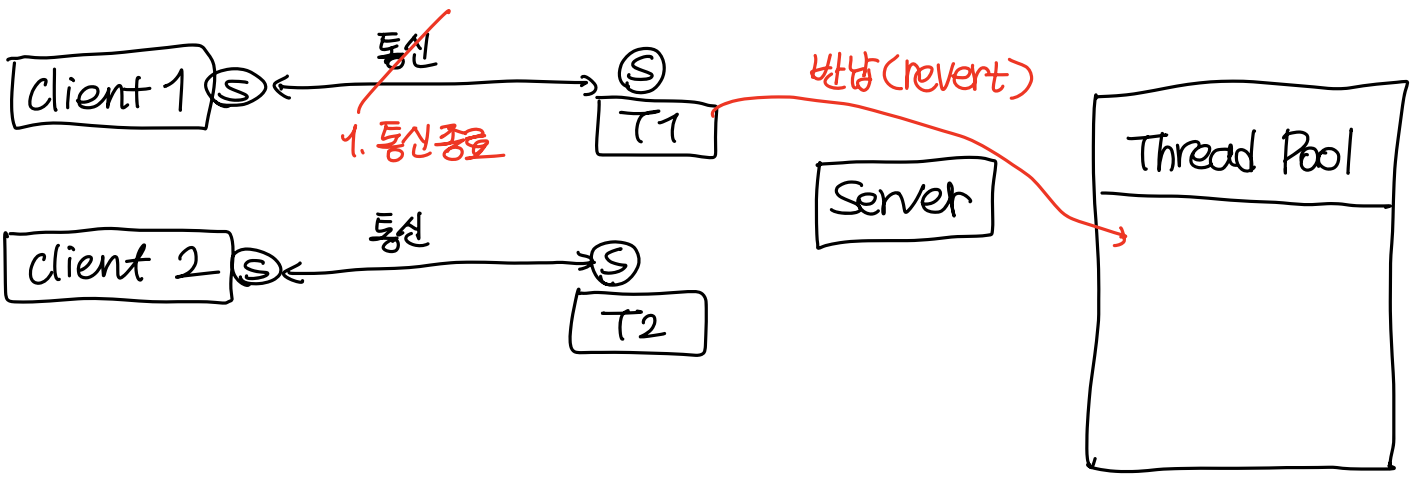
②


Client 1
통신
T1
Client 2
1. 연결
ServerSocket
Server
3. get
Thread Pool
5. 리턴
2. accept()
5. 통신
4. 생성
T2
T2

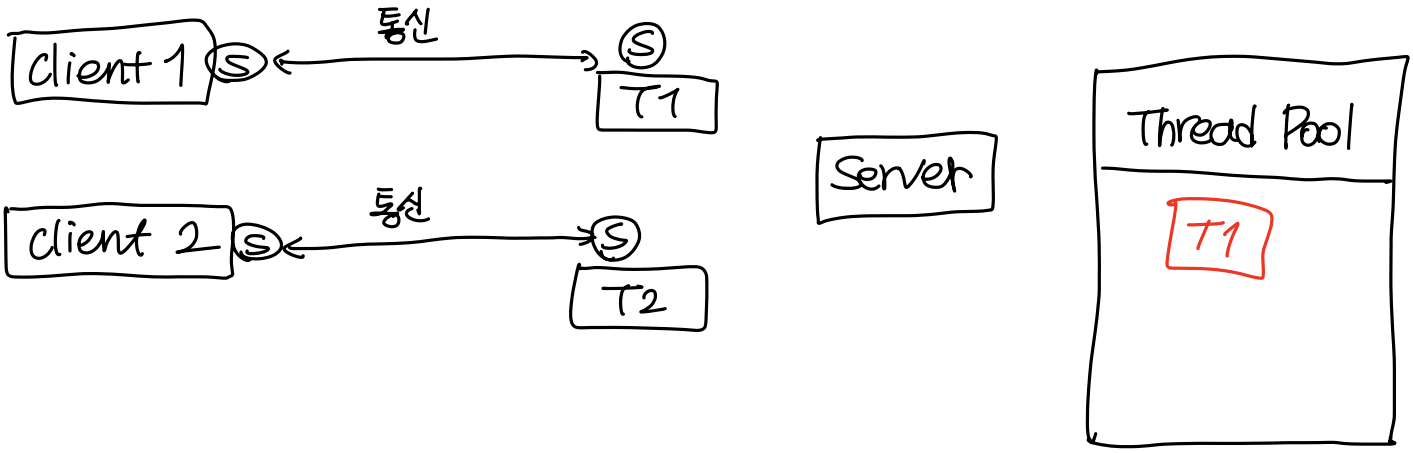
③


Client 1
통신
T1
Server
Thread Pool
Client 2
통신
T2

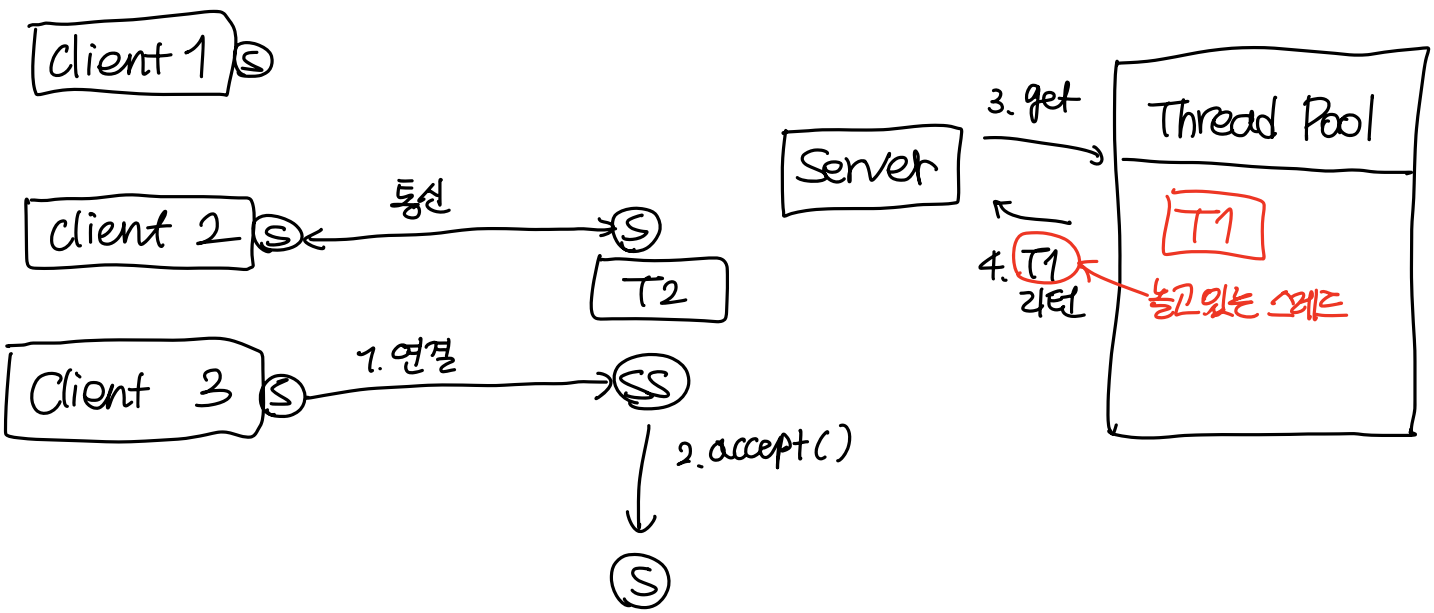
④


Client 1
통신
1. 통신종료
T1
반납(revert)
Server
Thread Pool
Client 2
통신
T2

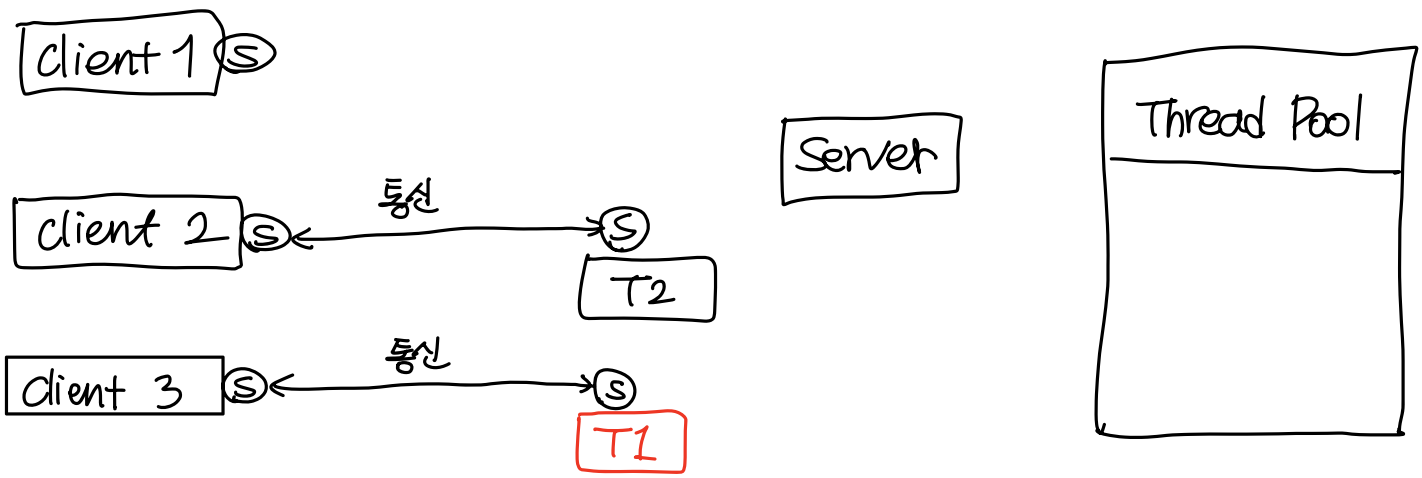
⑤


client 1
S
통신
S
T1
client 2
S
통신
S
T2
Server
Thread Pool
T1

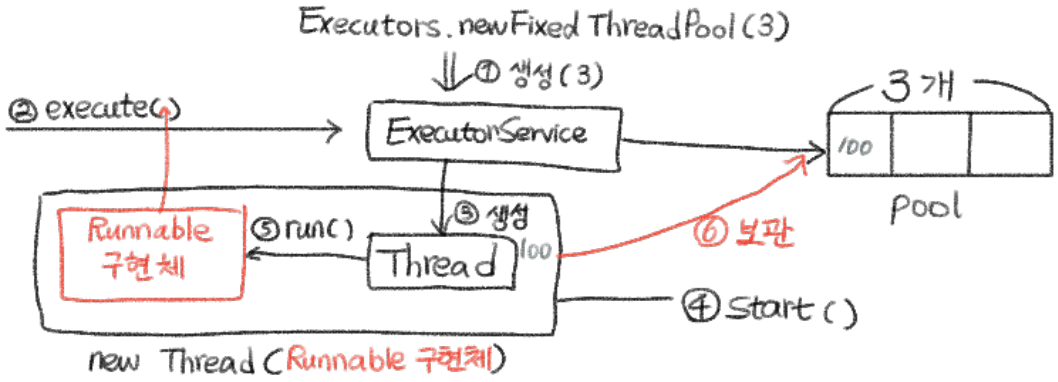
⑥


client 1
S
client 2
S
통신
S
T2
Client 3
S
1. 연결
SS
2. accept()
S
Server
3. get
Thread Pool
T1
4. T1
리턴
놀고있는 스레드

⑦


client 1
S
client 2
S
통신
S
T2
client 3
S
통신
S
T1
Server
Thread Pool

* Java 의 Thread Pool ⇒ ExecutorService



* ExecutorService.execute()