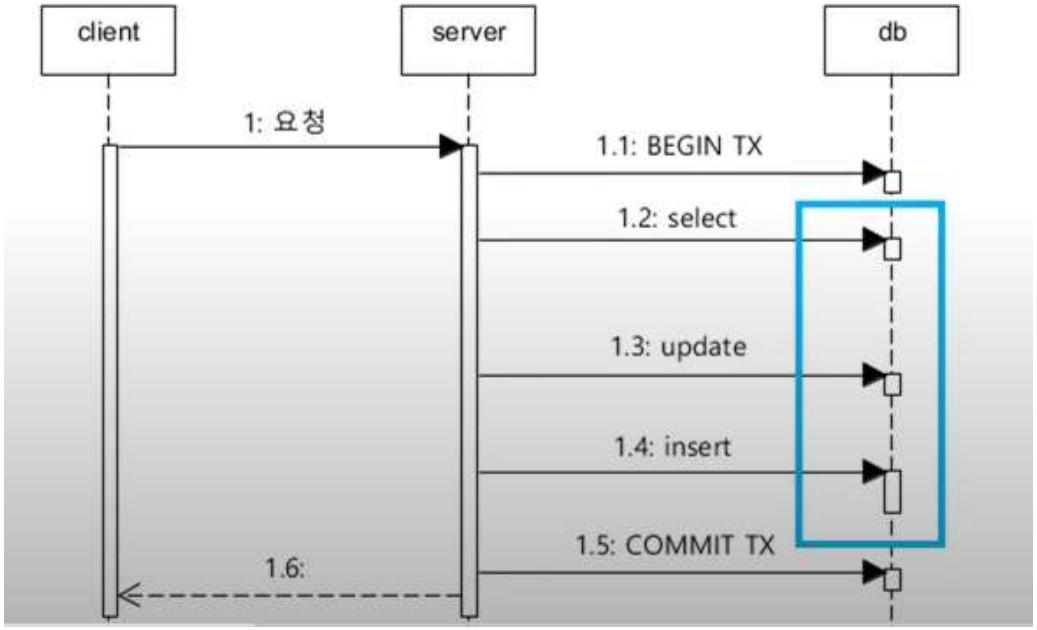
트랜잭션의 이해

트랜잭션이란 무엇인가?

여러 읽기/쓰기 작업을 묶어둔
DB의 논리적인 작업 단위



트랜잭션 작업의 핵심은?

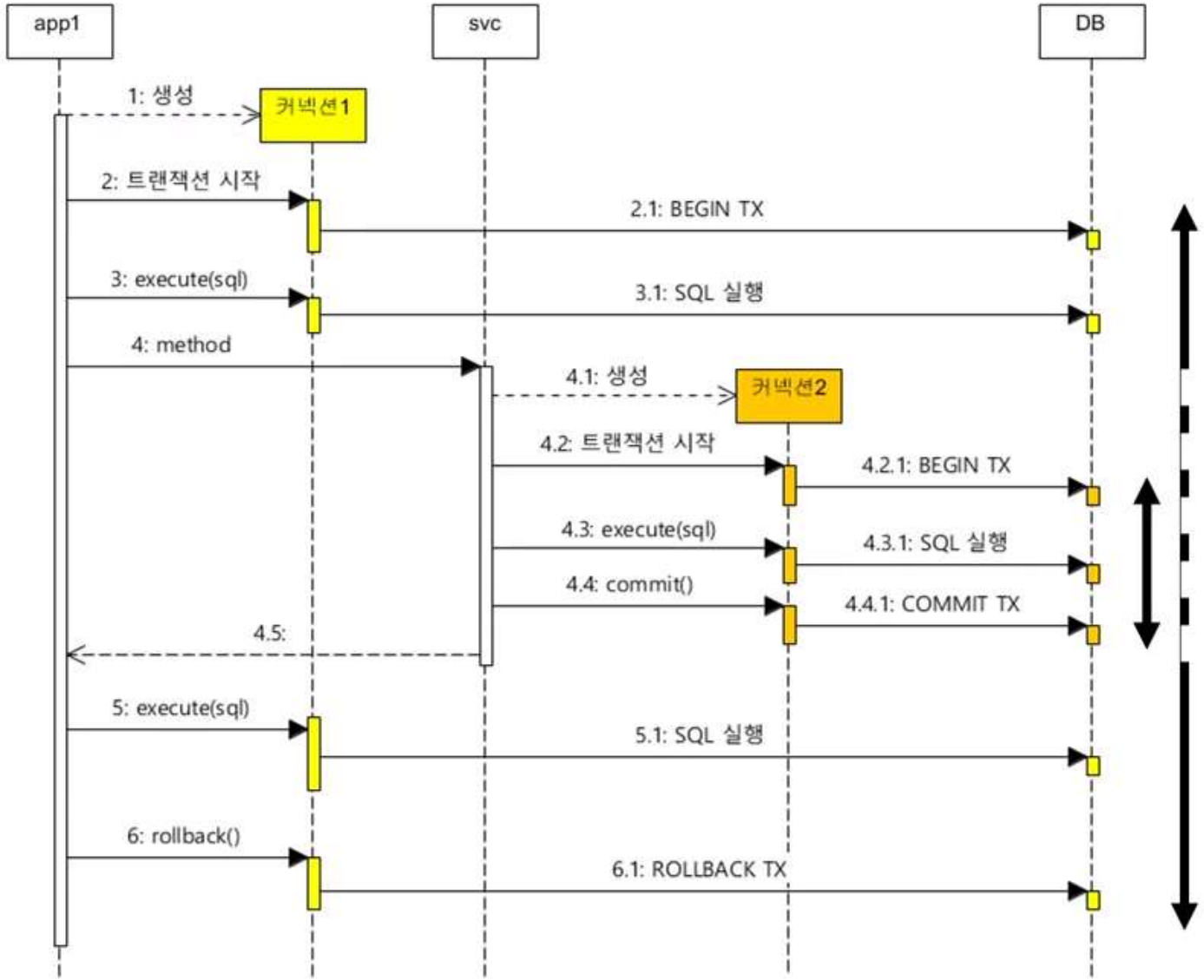
모두 반영하거나 모두 반영하지 않거나.
All or Nothing

원자성 (Atomicity)

왜 원자란 용어를 사용했을까?

2개의 커넥션을
한 트랜잭션에 묶어서 작업하려면?

혼난다.



트랜잭션은
같은 커넥션 내에서만
동작한다.

여러 개의 메서드에서 사용하는
데이터베이스 연산을
하나의 트랜잭션으로 묶으려면 어떻게 해야 하나?

님 이것보다 무거움?

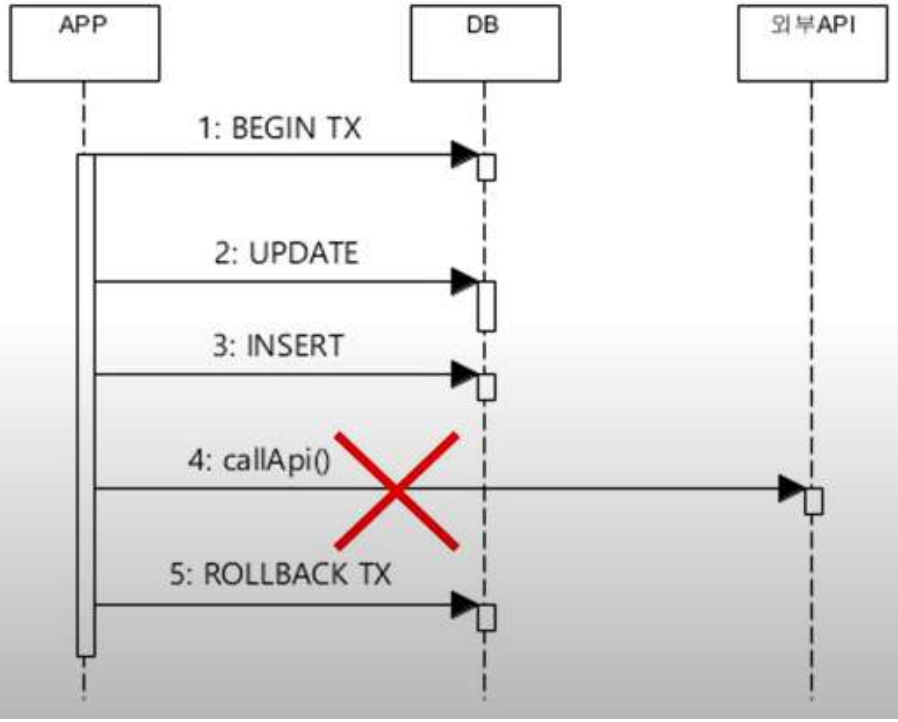
- 트랜잭션 전파

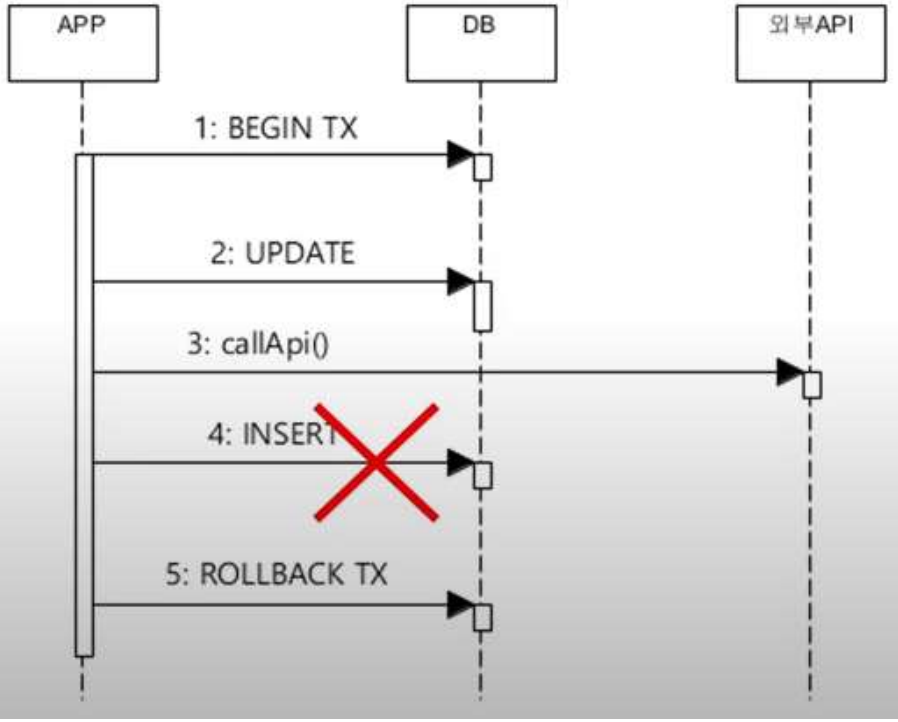
```
@Transactional
public void create() {
    if (dao.checkDuplicate(req.getEmail())) {
        throw new DupEmailException();
    }

    ...

    dao.insert(member);


}
```

```
public boolean checkDuplicate(String email) {
    return jdbcTemplate.queryForObject(…) > 0;
}
```

```
public void insert(Member m) {
    jdbcTemplate.update(…);
}
```

트랜잭션 작업 중에
외부 API를 사용했을 때는
따로 처리를 해줘야 한다.


APP
DB
외부API
1: BEGIN TX
2: UPDATE
3: INSERT
4: callApi()
5: ROLLBACK TX


APP
DB
외부API
1: BEGIN TX
2: UPDATE
3: callApi()
4: INSERT
5: ROLLBACK TX

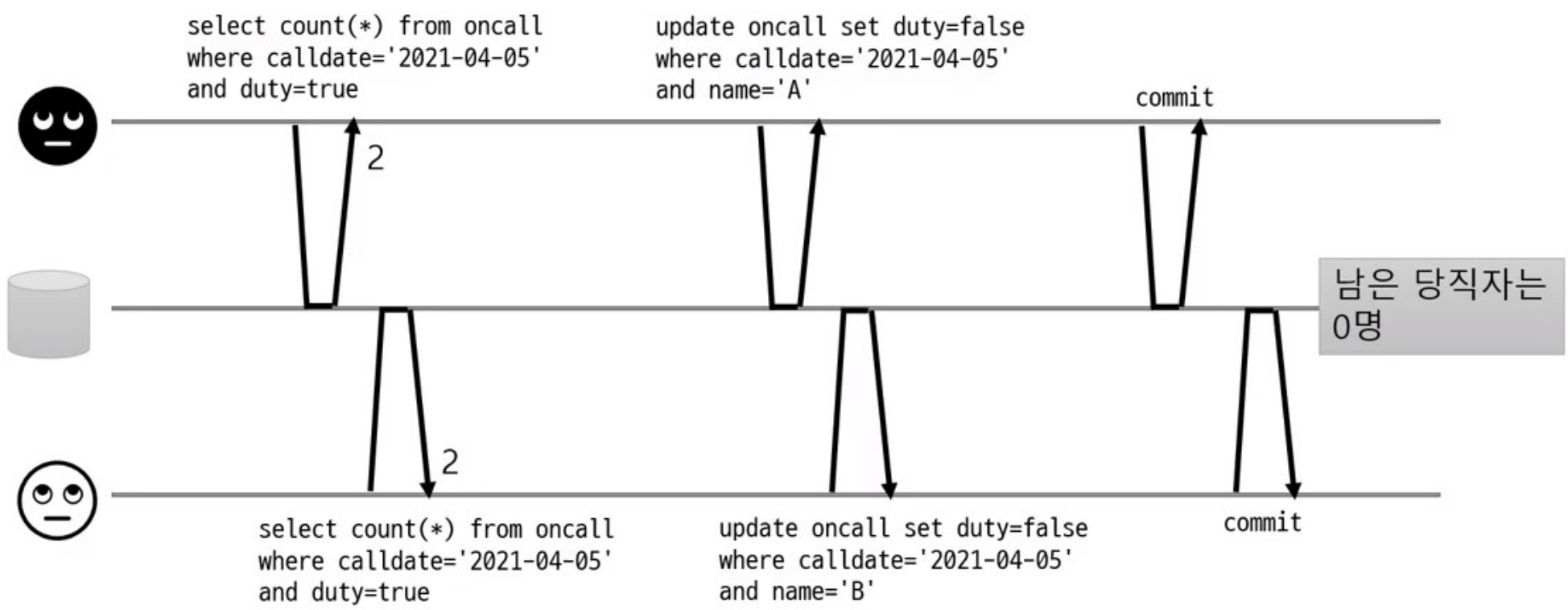
여러 개의 DB를
한 트랜잭션에 묶으려면?

성능이 떨어져서 잘 사용하지 않는다는.....

데이터에 동시에 접근하면
어떤 문제가 생길까?

당직자가 최소 1명은 남아 있어야 한다.
당직자 명단은 DB 조회를 통해서 확인할 수 있다.

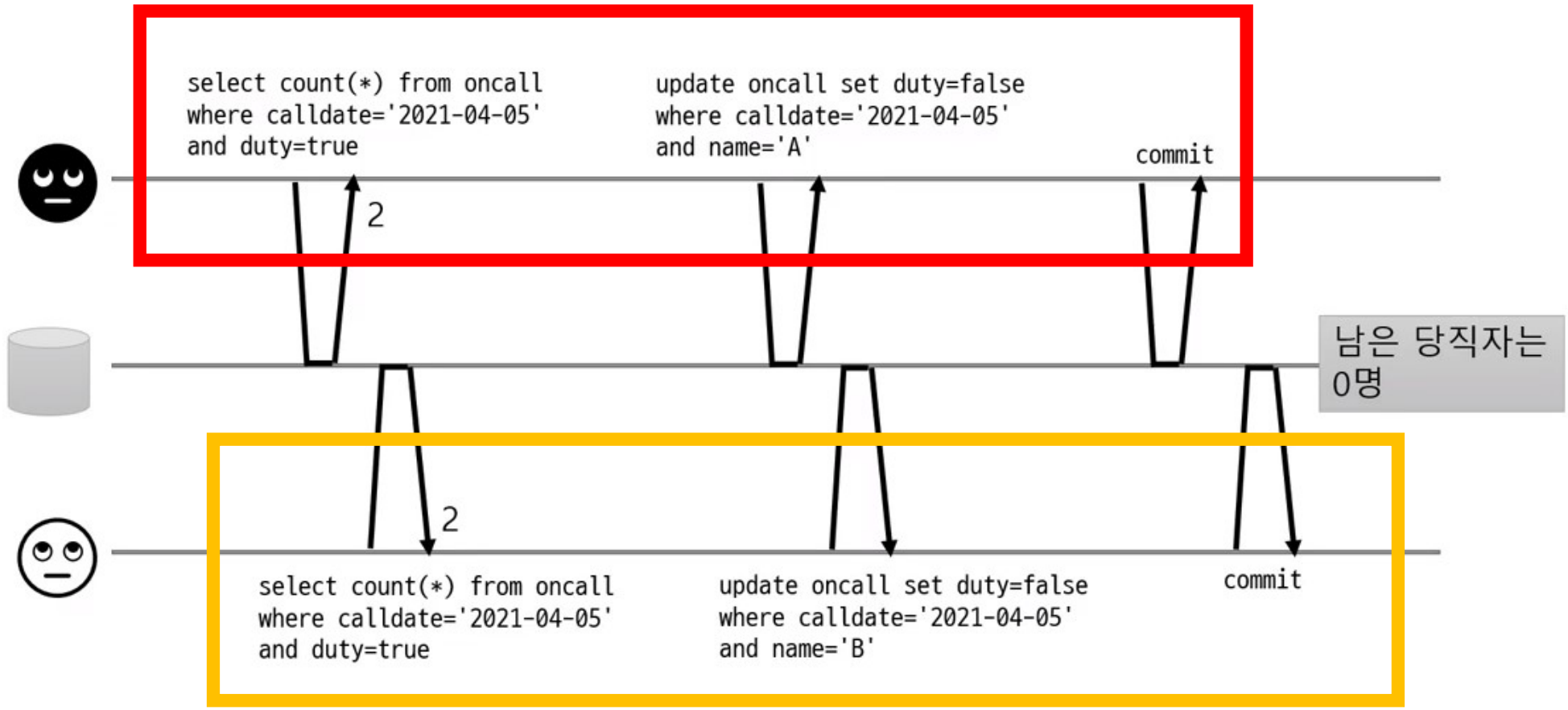
성진대리님과 준성대리님은 집에 가고 싶은 마음에
당직자를 조회 해보는데...


select count(*) from oncall
where calldate='2021-04-05'
and duty=true
update oncall set duty=false
where calldate='2021-04-05'
and name='A'
commit
2
남은 당직자는
0명
2
select count(*) from oncall
where calldate='2021-04-05'
and duty=true
update oncall set duty=false
where calldate='2021-04-05'
and name='B'
commit

• 경쟁 상태 (Race Condition)

: 여러 클라이언트가 같은 데이터에 접근하는 상황.

경쟁상태를 해결하려면?


select count(*) from oncall
where calldate='2021-04-05'
and duty=true
update oncall set duty=false
where calldate='2021-04-05'
and name='A'
commit
2
남은 당직자는
0명
2
select count(*) from oncall
where calldate='2021-04-05'
and duty=true
update oncall set duty=false
where calldate='2021-04-05'
and name='B'
commit

한번에 한 트랜잭션의 작업만 하면
경쟁상태로 인한 문제가 발생하지 않는다.

하지만 프로그램의 전체 성능이 저하된다.

- **트랜잭션 격리(Isolation)**

: 트랜잭션을 서로 격리해서
다른 트랜잭션에 영향을 주지 못하게 함

• 격리 수준

Read Uncommitted
Read Committed
Repeatable Read
Serializable

아래로 갈수록
트랜잭션 간에
격리수준이 높아진다

격리수준이 높아지면
병행처리가 어렵기 때문에
성능이 떨어진다.

그렇기 때문에
발생하는 문제에 맞는
격리수준을 적용해야 한다.

Read Uncommitted

: 한 트랜잭션이 작업 중 이어도
다른 트랜잭션이 해당 데이터를 읽을 수 있다.

거의 사용하지 않음 => Les't skip

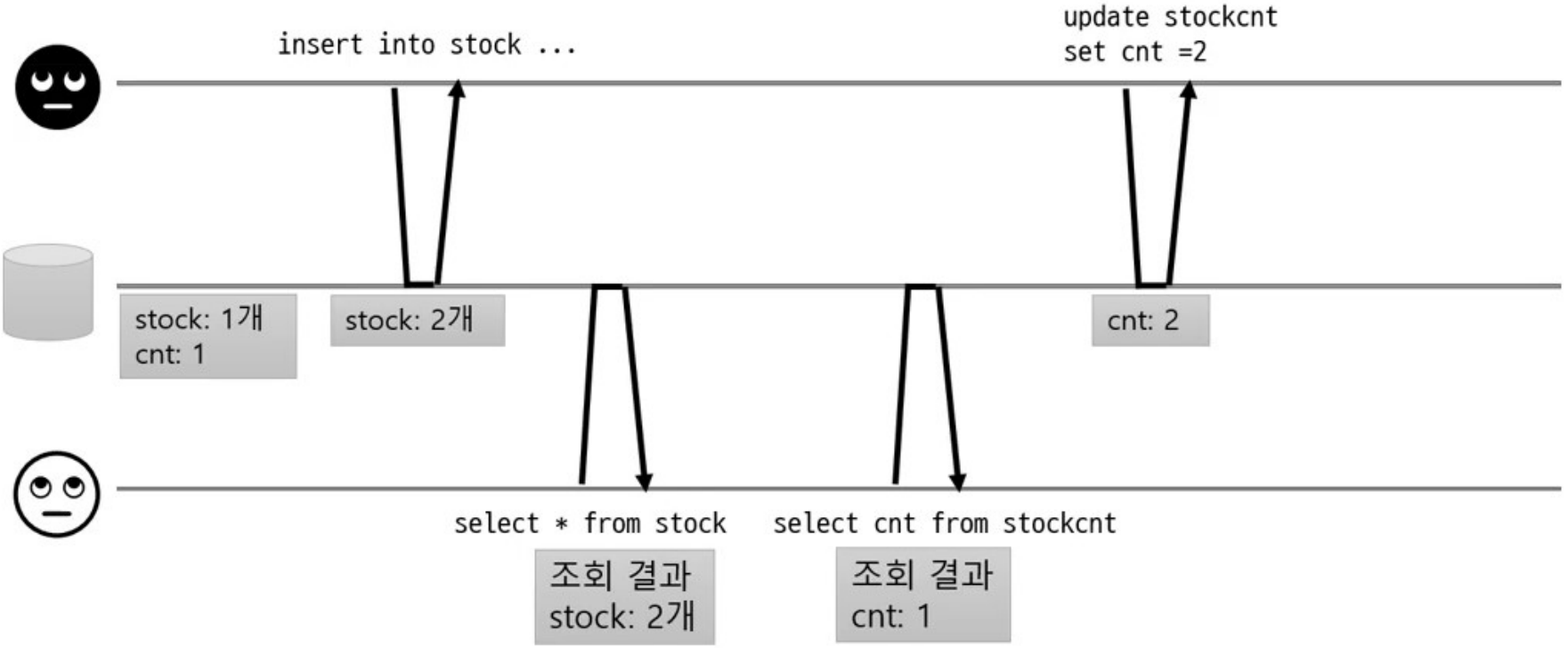
Read Committed

- 커밋된 데이터만 읽기
  : 한 트랜잭션이 시작되고
  update를 실행 경우 commit을 해야지
  그 updat한 데이터를 다른 트랜잭션에서 읽을 수 있다.

- 커밋된 데이터만 덮어쓰기
  : 한 트랜잭션이 update를 실행한 경우 commi해야지
  다른 트랜잭션이 그 데이터를 다시 update할 수 있다.

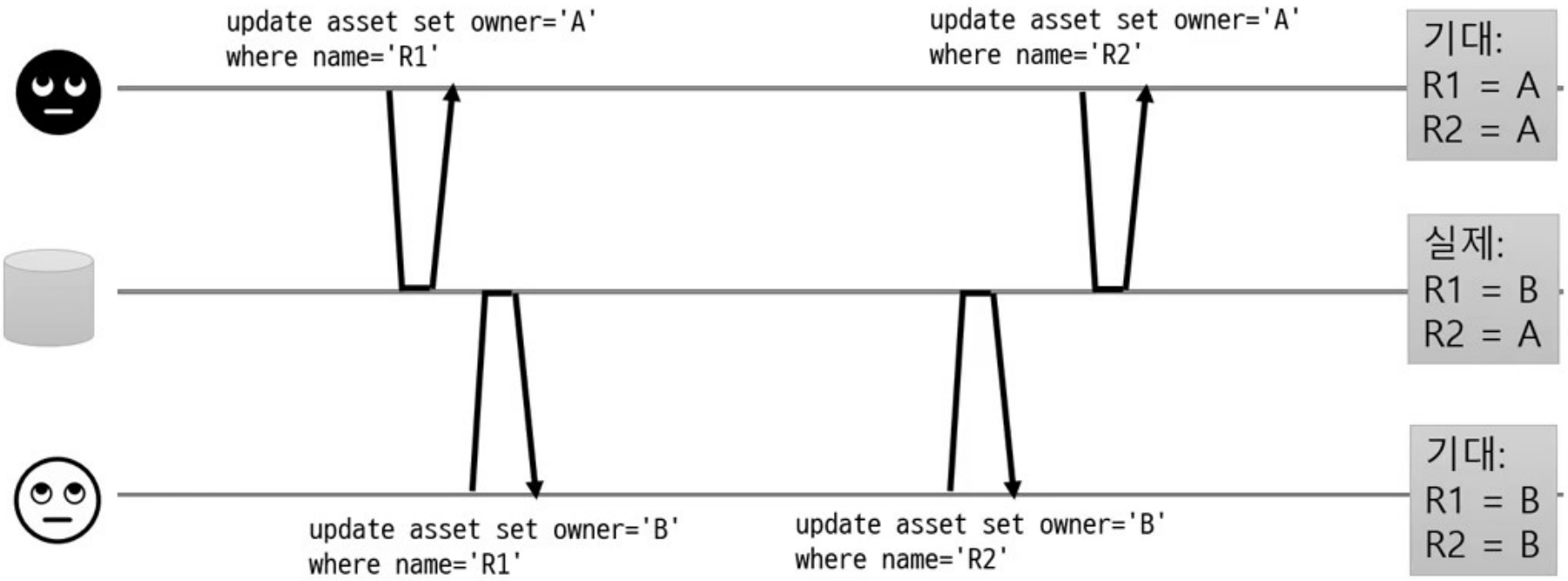
- Point : 커밋 전의 데이터를 읽게 한다.
- Point : Oracle에서 사용하는 수준.

Dirty read를 해결해 준다.



수정 중인 행은 읽지 못하게 한다.

Dirty write를 해결해 준다.



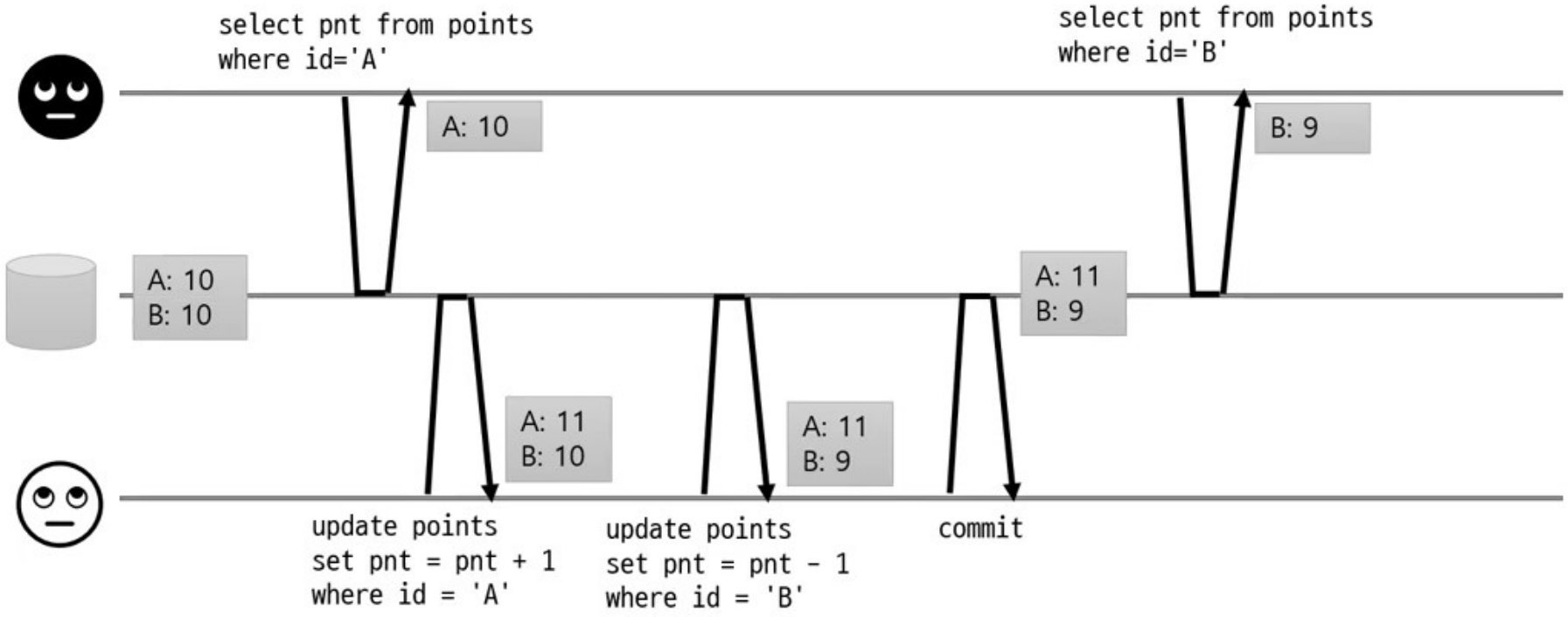
행 단위로 잠금을 실행한다.
수정중인 행이 있으면 잠금을 한다.

Repeatable Read

: 한 트랜잭션이 시작되어 update를 한 경우
  다른 트랜잭션은 이전 버전의 데이터를 읽을 수 있다.

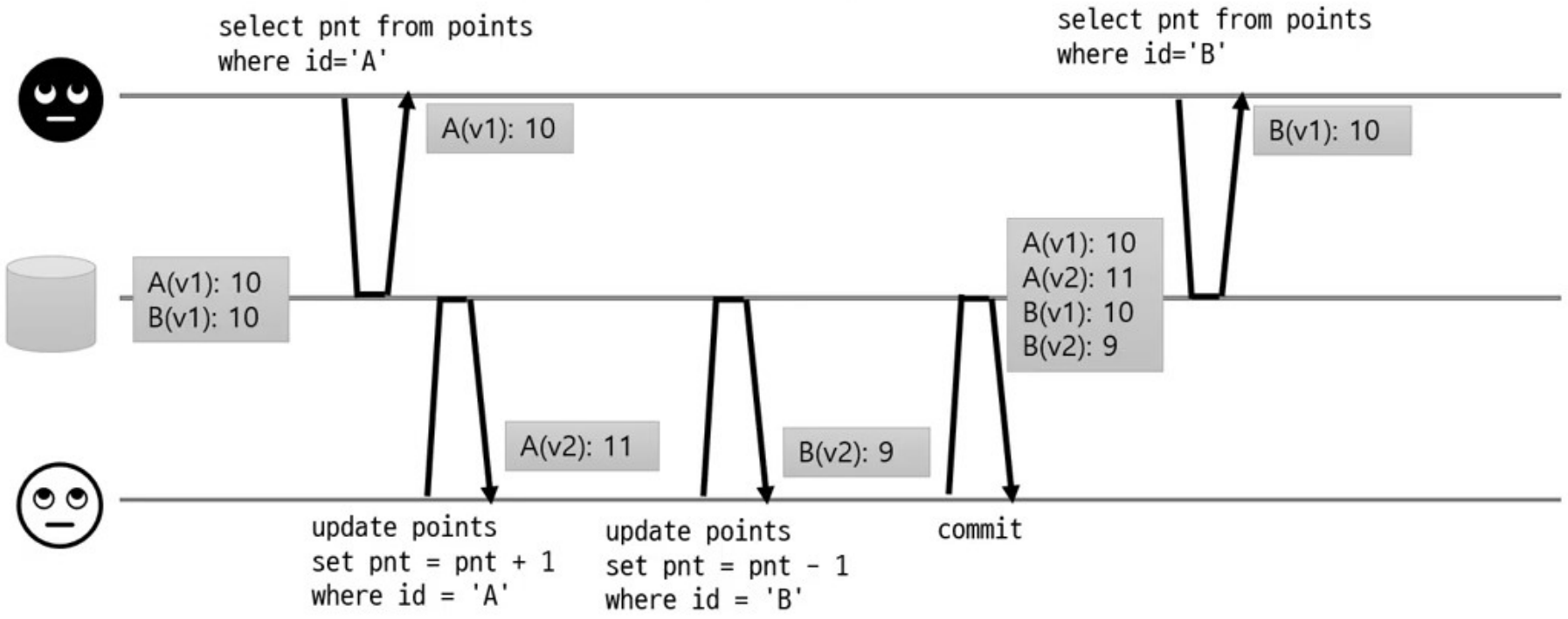
- Point : 커밋 전에는다른 이전 버전의 데이터를 읽게 한다.
- Point : MySQL에서 사용하는 수준.

Read skew



읽는 시점에 따라 데이터가 달라진다.

Read skew를 해결해 준다.



읽는 시점에 특정 버전을 읽게 한다.

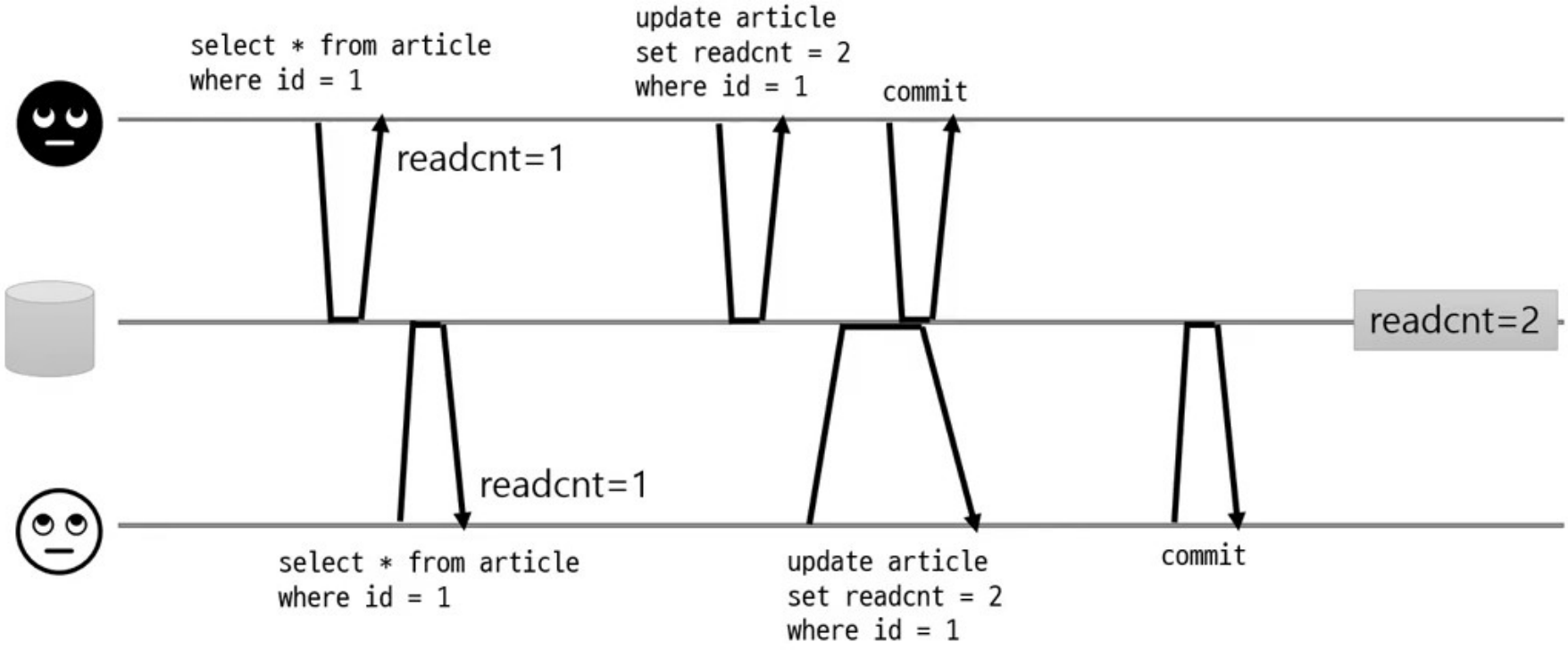
Read Committed와 Repeatable Read의 차이점

- Read Committed
  : 한 트랜잭션에서 update를 하고 해당 데이터를 다시 읽으면 변경 전 데이터가 조회 된다.

- Repeatable Read
  : 한 트랜잭션에서 update를 하고 해당 데이터를 다시 읽으면 변경 된 데이터가 조회 된다.

Read Committed와 Repeatable Read를
사용해도 발생하는 오류가 있다.

Lost Update

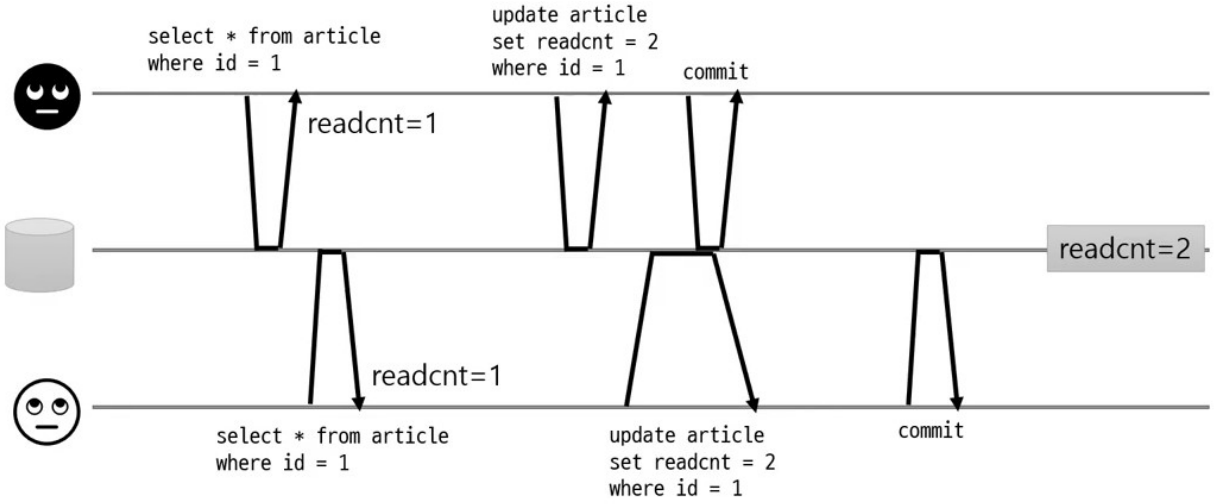

select * from article
where id = 1
update article
set readcnt = 2
where id = 1
commit
readcnt=1
readcnt=2
readcnt=1
select * from article
where id = 1
update article
set readcnt = 2
where id = 1
commit

해결방안 3가지

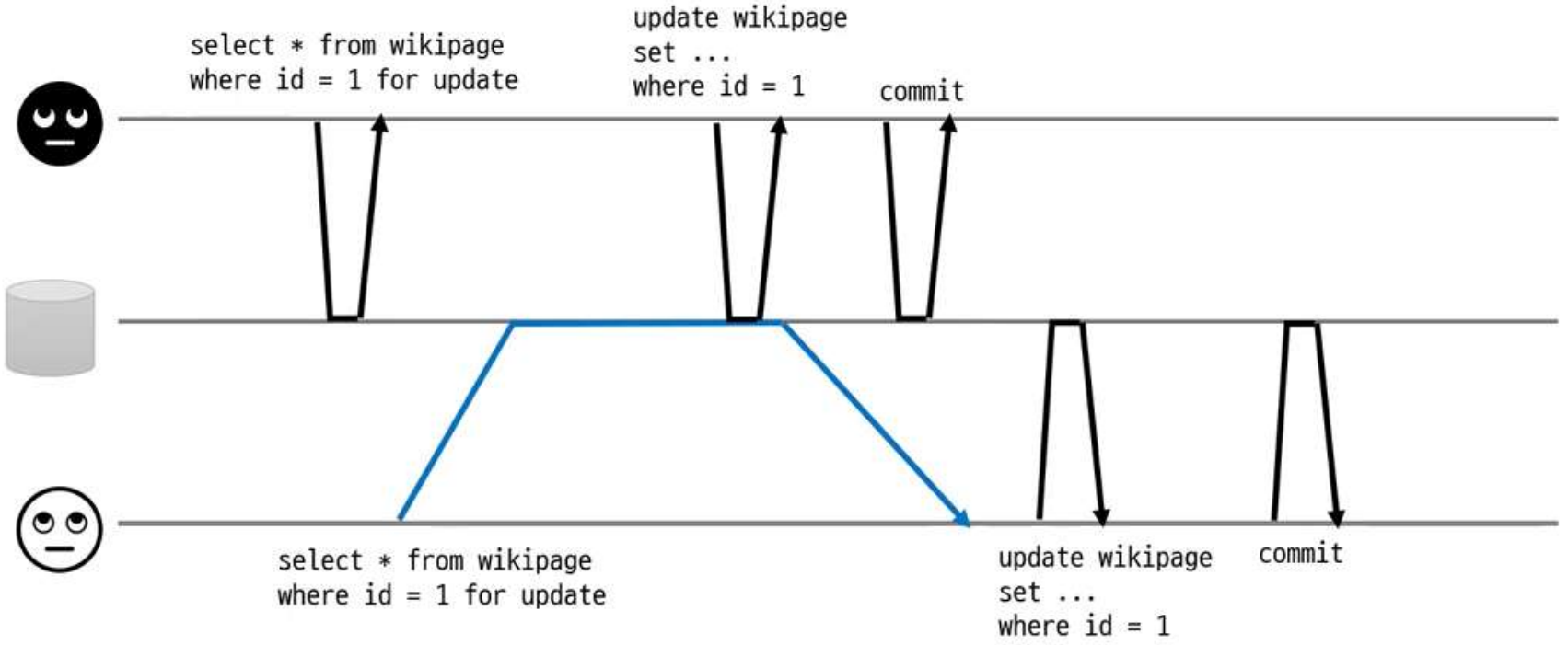
- 원자적 연산 사용
- 명시적 잠금
- CAS (Compare And Set)

원자적 연산 사용

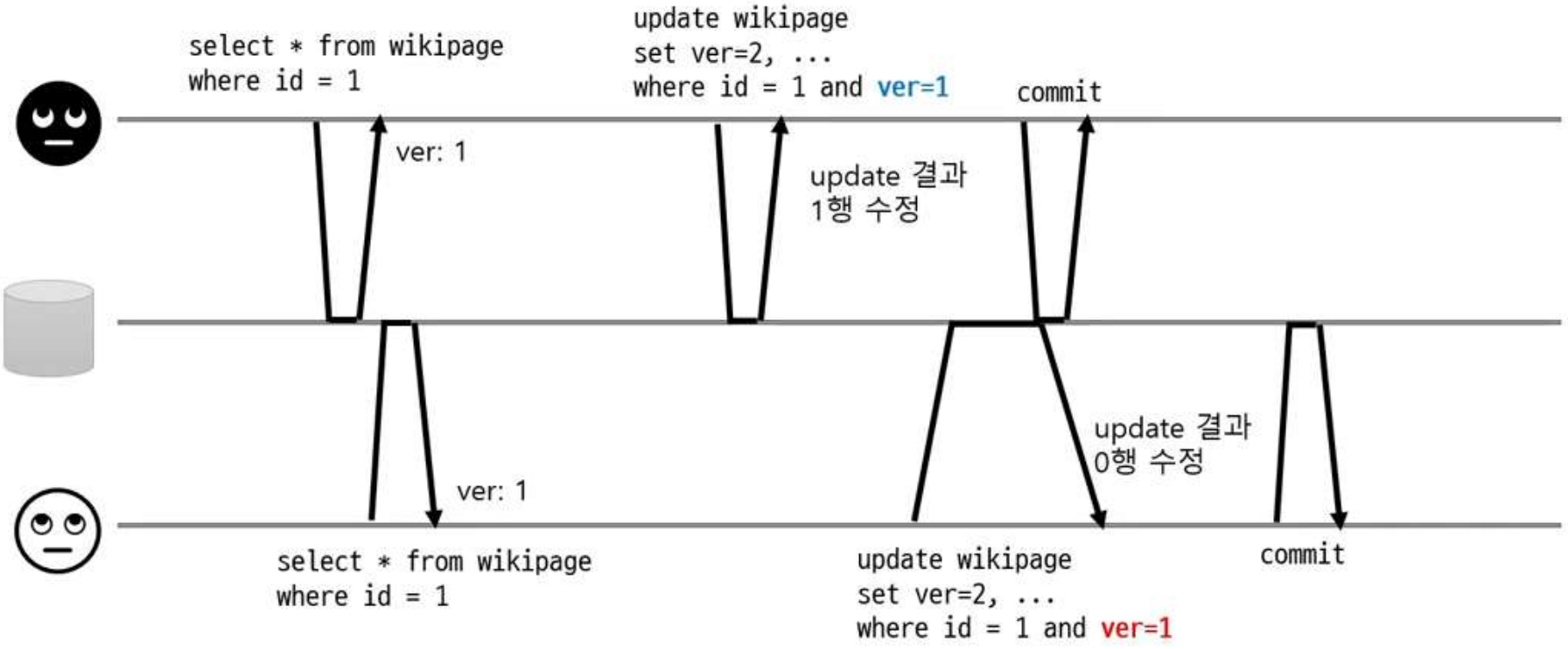
update artice
set readcnt = readcnt+1
where id = 1



명시적 잠금 (for update)



CAS (Compare And Set)



SERIALIZABLE

완전한 격리

한 트랜잭션이 데이터를 읽기만 해도
공유잠금을 설정해서
다른 트랜잭션에서 해당 데이터를 변경하지 못하게 한다.