

Database Study

2주차 - Transaction

01

트랜잭션과 ACID

세미나

02

DBMS가 ACID를 보장하는 방법

세미나

03

트랜잭션의 격리 레벨

세미나

04

MSA 환경의 트랜잭션

면접 대비

05

Q & A

질의응답

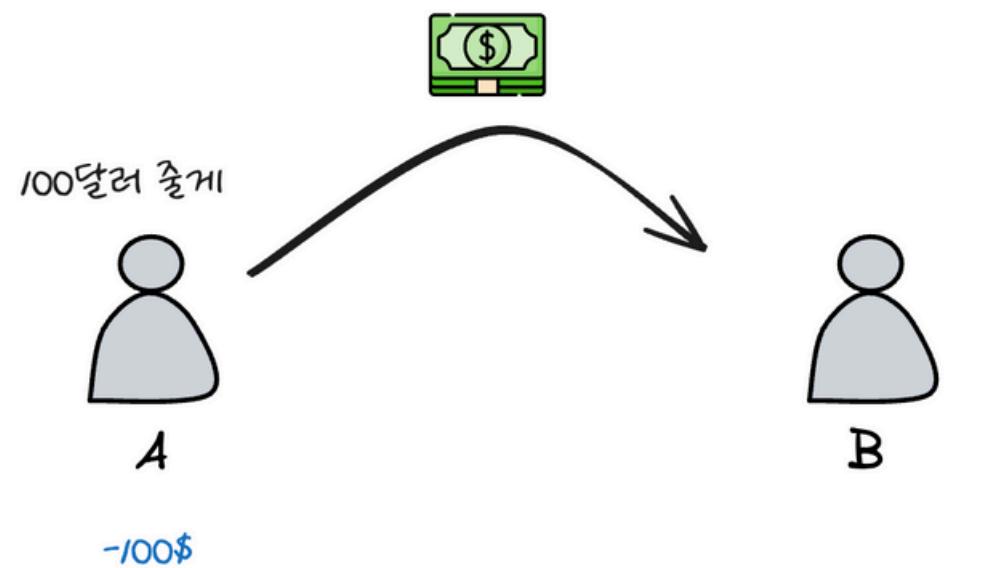
트랜잭션과 ACID

»»» 트랜잭션이란?

“데이터베이스의 상태를 바꾸게 하기 위해서 수행되는 작업의 최소 단위”

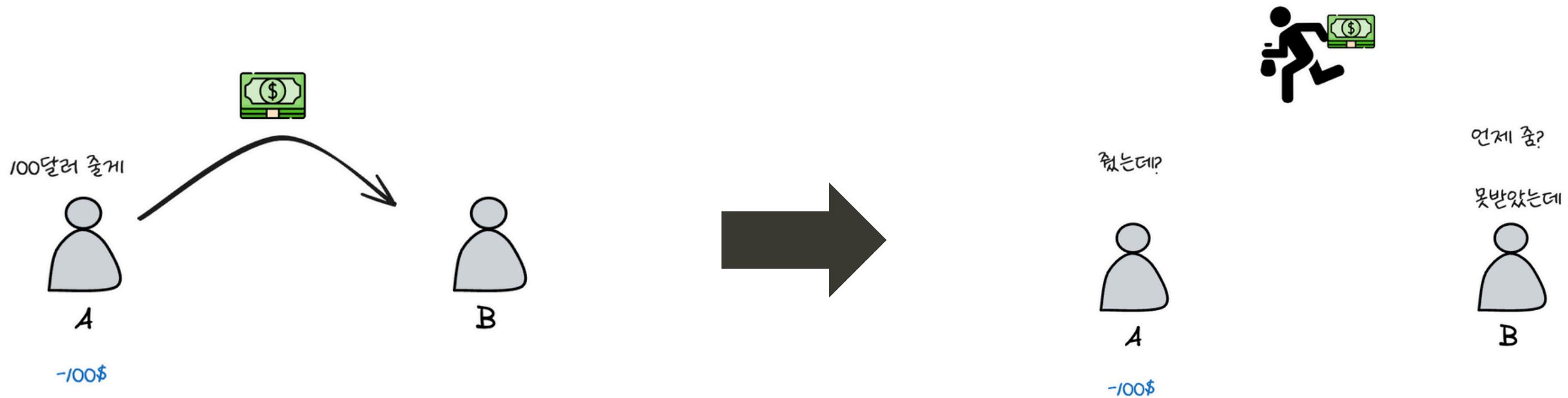
트랜잭션과 ACID

>>> 트랜잭션이라는 개념은 왜 필요한가?



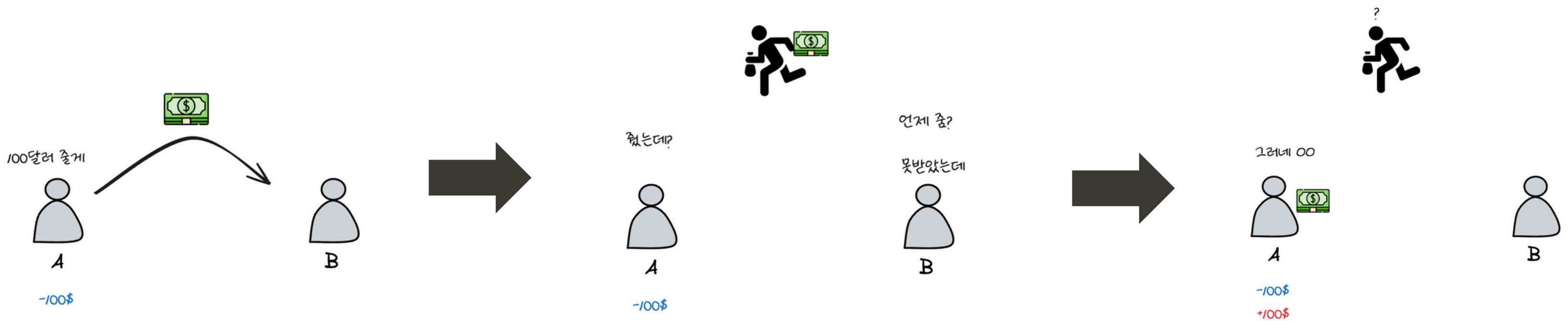
트랜잭션과 ACID

>>> 트랜잭션이라는 개념은 왜 필요한가?



트랜잭션과 ACID

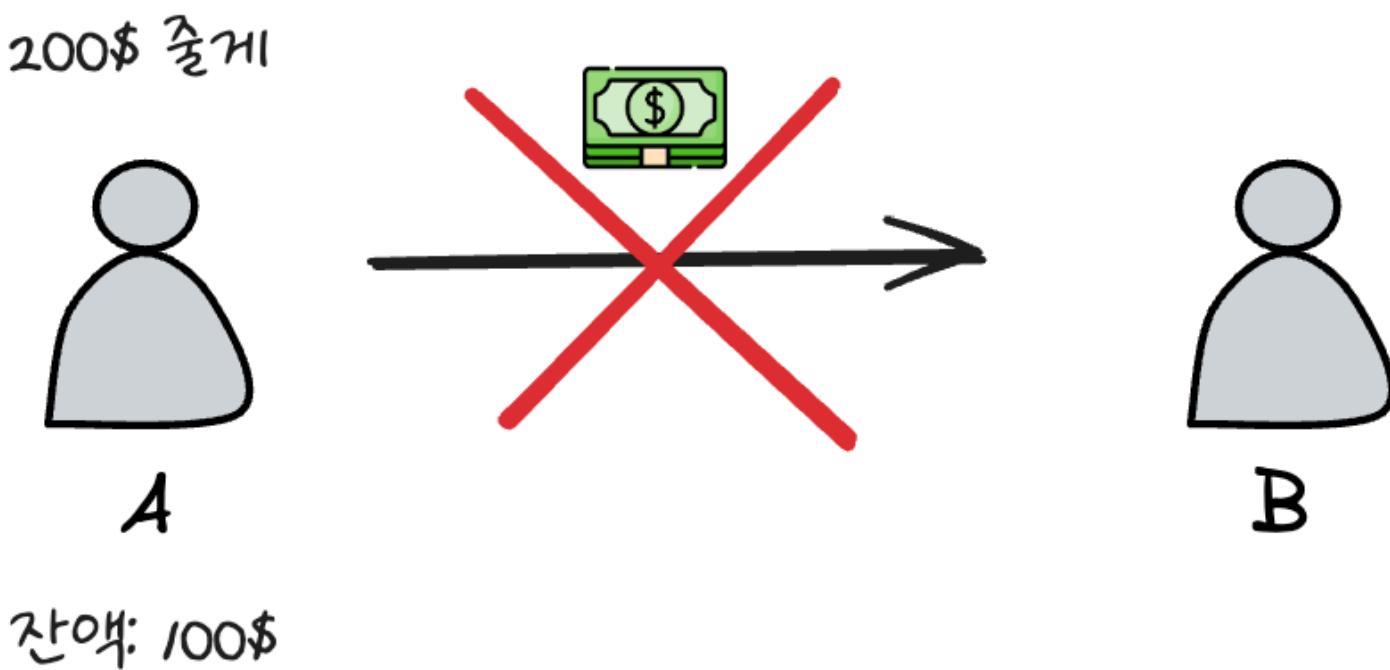
>>> 트랜잭션의 A: Atomicity (원자성)



트랜잭션과 ACID

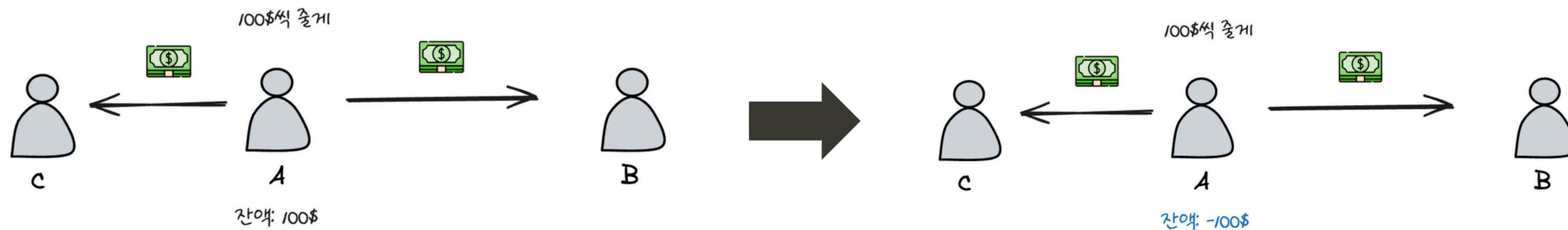
»»» 트랜잭션의 C: Consistency(일관성)

계좌의 규칙: 잔액은 0 이상을 유지한다.



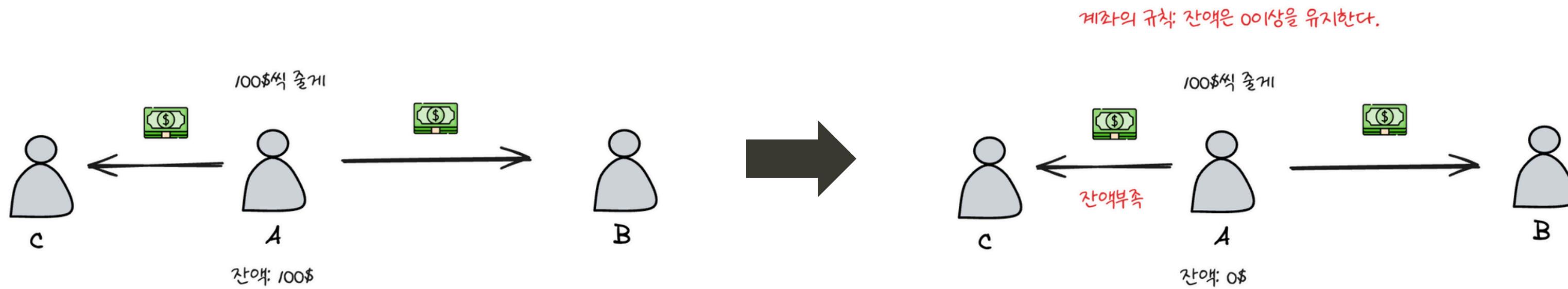
트랜잭션과 ACID

>>> 트랜잭션의 I: Isolation(격리성, 고립성)



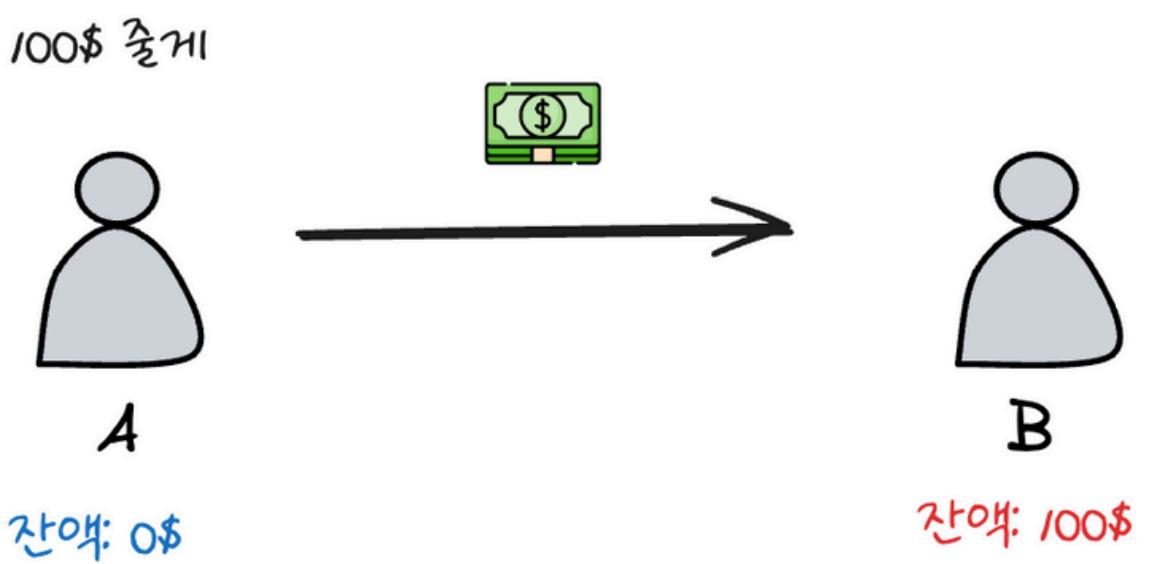
트랜잭션과 ACID

>>> 트랜잭션의 I: Isolation(격리성, 고립성)



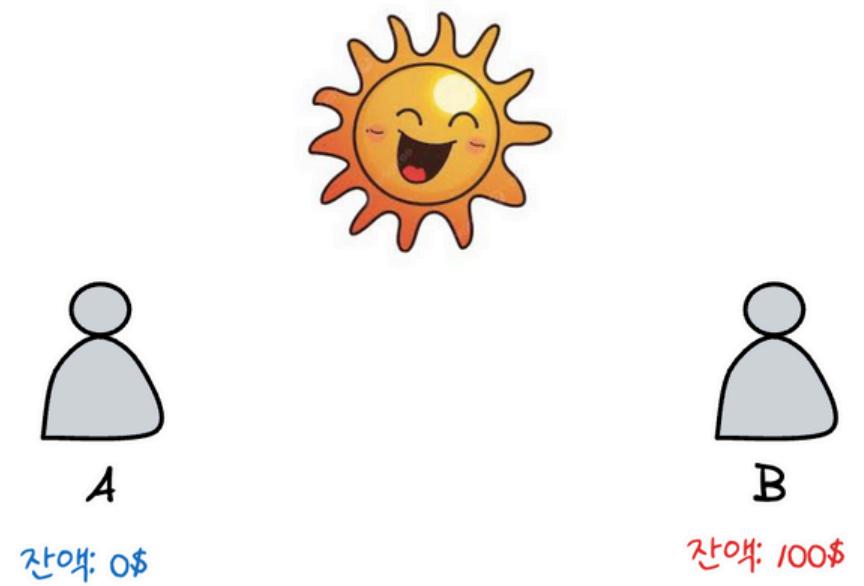
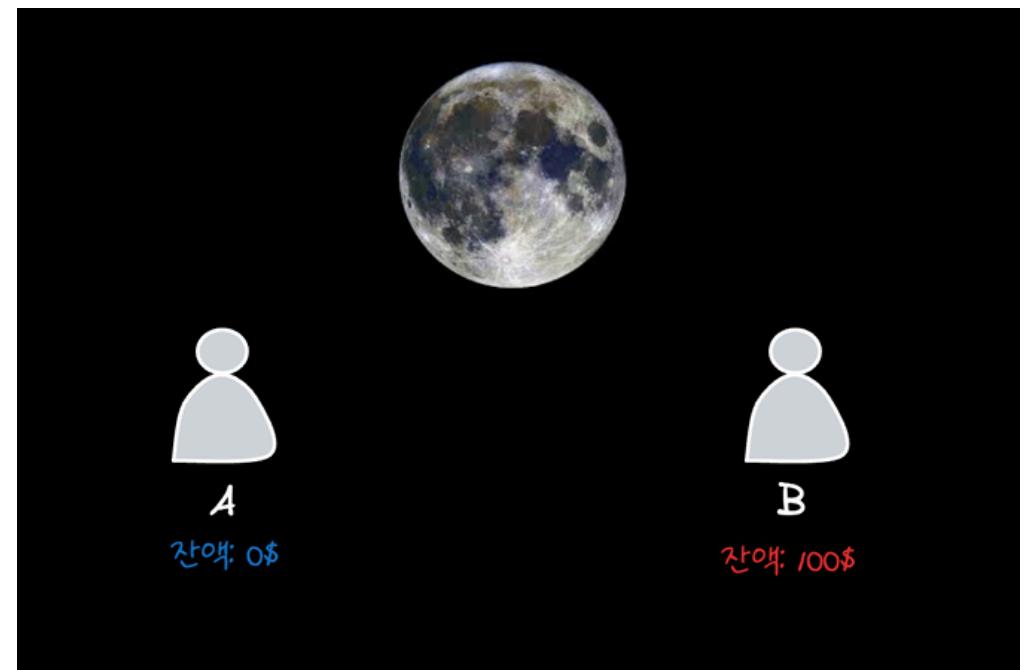
트랜잭션과 ACID

>>> 트랜잭션의 D: Durability(지속성)



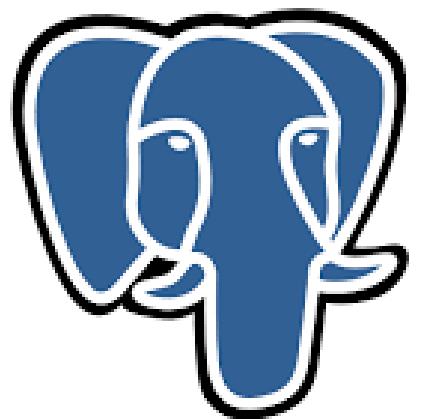
트랜잭션과 ACID

>>> 트랜잭션의 D: Durability(지속성)



☰ DBMS가 ACID를 보장하는 방법

»»» DBMS가 ACID를 보장하는 방법



☰ DBMS가 ACID를 보장하는 방법

»»» 트랜잭션의 A: Atomicity (원자성)



MVCC(Multi-Version Concurrency Control, 다중 버전 동시성 제어)

1. 갱신 요청이 들어오면 갱신 이전 값을 undo log에 복사 (스냅샷)
2. 만약 트랜잭션에 실패한다면, undo log에 있는 백업된 데이터를 InnoDB 버퍼 풀로 복구해 롤백
3. 트랜잭션에 성공해 commit이 되어도 undo log를 바로 삭제하지는 X

☰ DBMS가 ACID를 보장하는 방법

»»» 트랜잭션의 A: Atomicity (원자성)



Q. MVCC에서 Undo log를 바로 삭제하지 않는 이유는?

☰ DBMS가 ACID를 보장하는 방법

»»» 트랜잭션의 A: Atomicity (원자성)



Q. MVCC에서 Undo log를 바로 삭제하지 않는 이유는?

A. 해당 트랜잭션보다 먼저 시작했지만, 나중에 끝나는 트랜잭션이 있을 수 있기 때문

☰ DBMS가 ACID를 보장하는 방법

»»» 트랜잭션의 C: Consistency(일관성)



각종 제약 조건: unique, pk, not null, fk 등등

innoDB는 제약 조건을 transaction commit 때마다 검사

☰ DBMS가 ACID를 보장하는 방법

»»» 트랜잭션의 I: Isolation(격리성, 고립성)

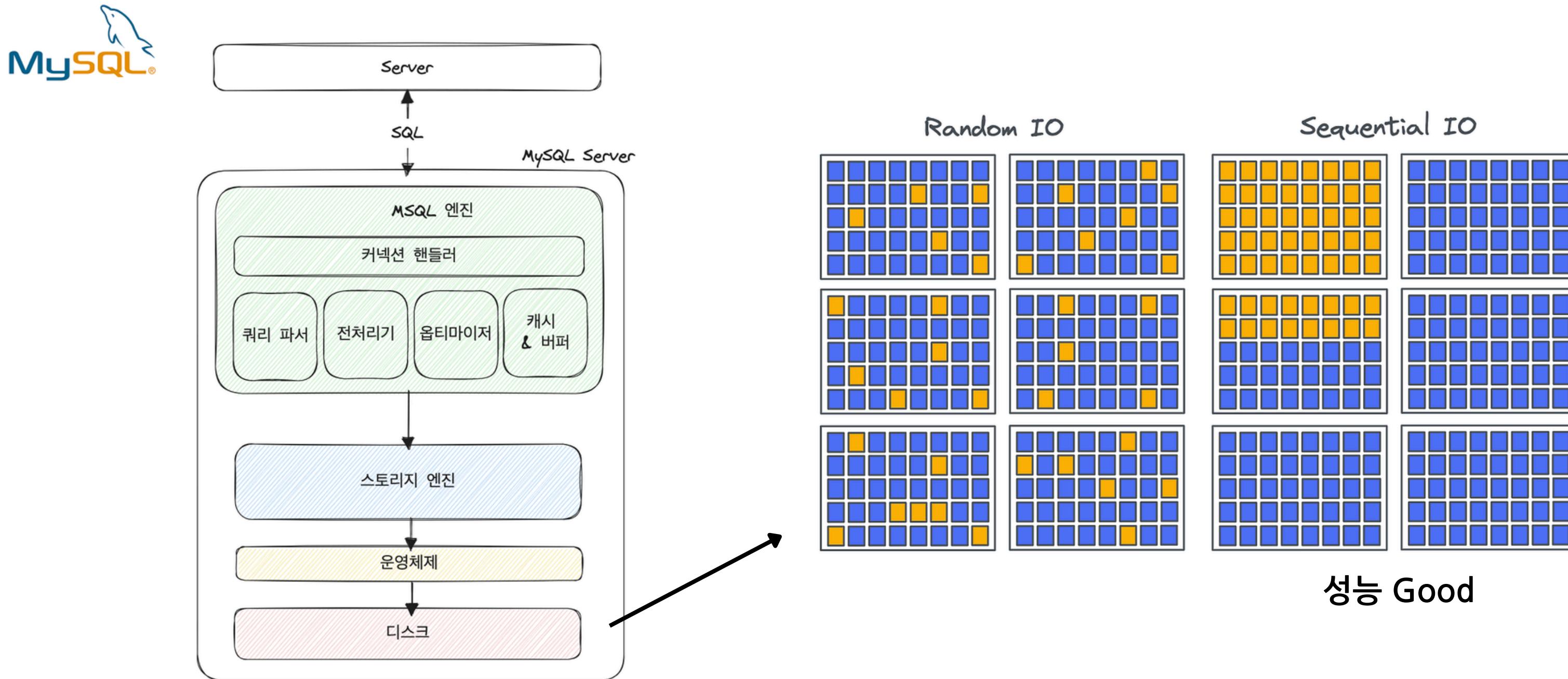


격리 레벨을 이용해 구현.

자세한 격리 레벨은 뒤에서..

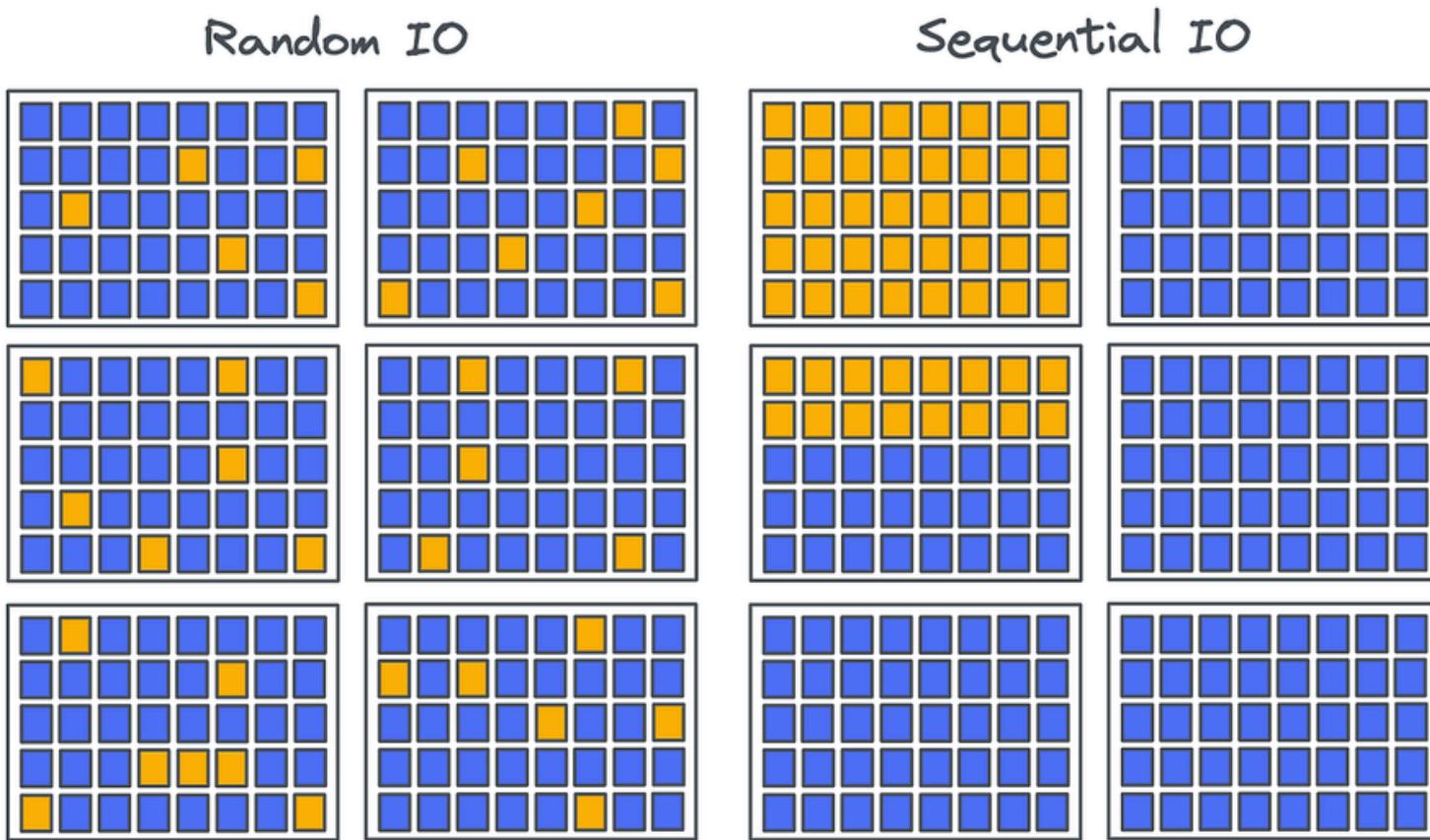
☰ DBMS가 ACID를 보장하는 방법

»»» 트랜잭션의 D: Durability(지속성)



☰ DBMS가 ACID를 보장하는 방법

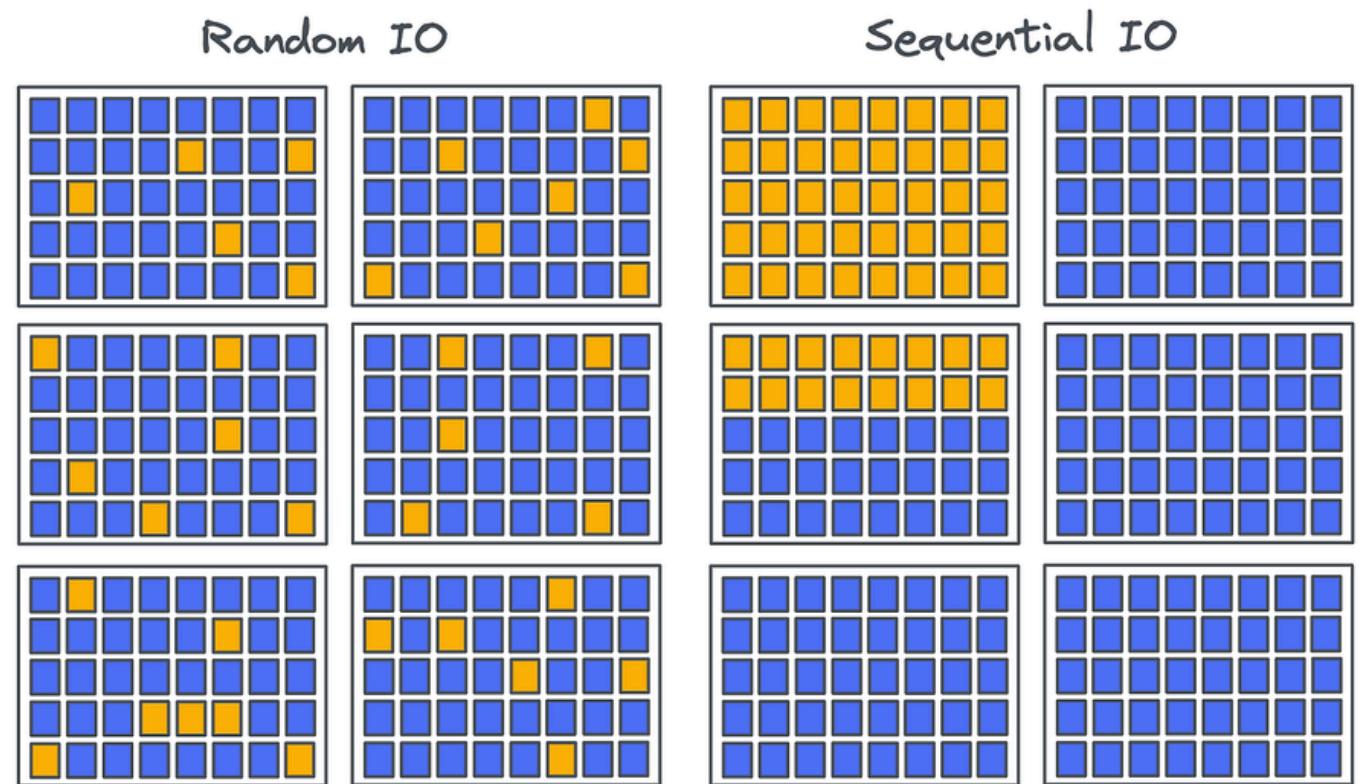
>>> 트랜잭션의 D: Durability(지속성)



순차 I/O가 성능이 좋은건 OK
그렇다면 Write 시 어떻게 데이터를 연속해서 넣을까?

☰ DBMS가 ACID를 보장하는 방법

»»» 트랜잭션의 D: Durability(지속성)



작업을 일부러 지연시킨다.

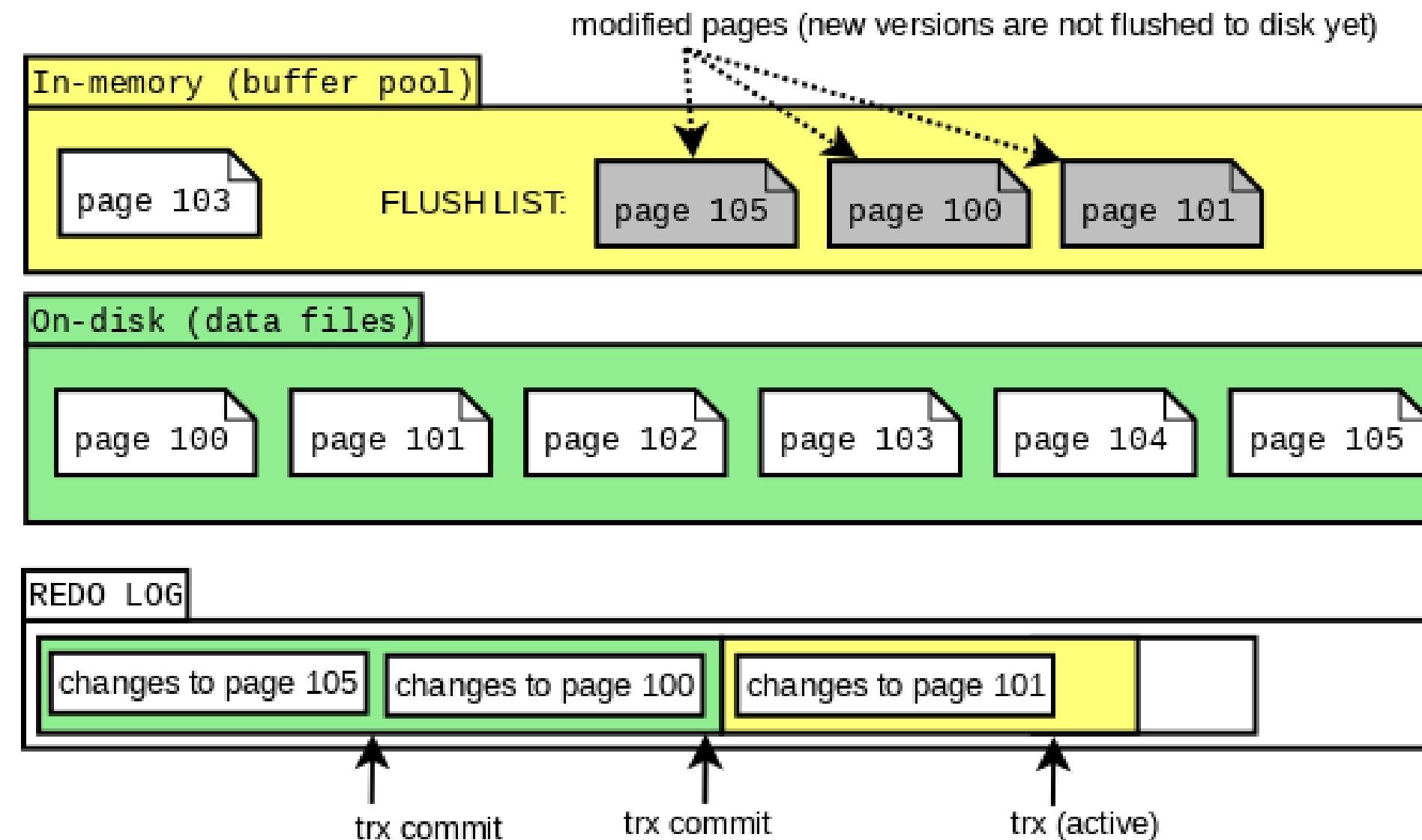
1. write 요청이 들어오면 메모리에 축적
2. 요청이 적정 사이즈 이상 쌓이면, write 연산을 통해 한 번에 디스크로

단, 메모리에 축적 후 저장하기 때문에 장애 발생 시 지속성의 문제 존재 ⚡

☰ DBMS가 ACID를 보장하는 방법

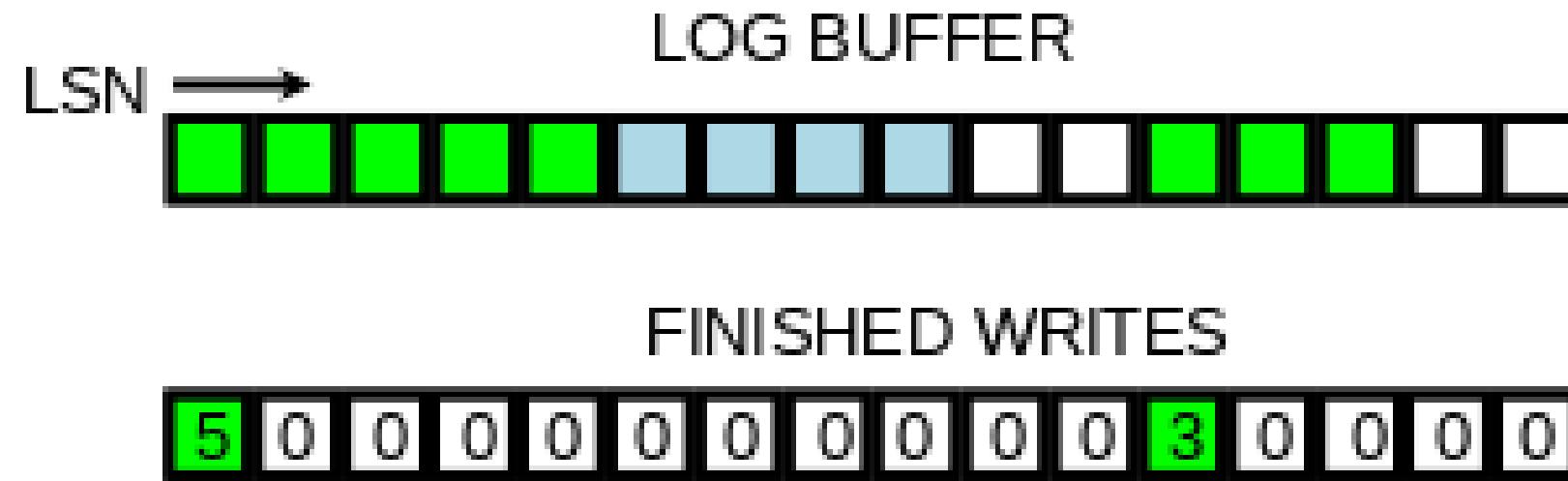
»»» 트랜잭션의 D: Durability(지속성)

해결법: WAL(Write Ahead Log)을 이용하자



☰ DBMS가 ACID를 보장하는 방법

»»» 트랜잭션의 D: Durability(지속성)



REDO,UNDO 여부는 LSN(Log Sequence Number)을 통해 파악

- page LSN < log LSN : 해당 페이지는 반드시 로그로 복구되어야 한다.
- page LSN \geq log LSN : 페이지는 이 로그보다 나중에 쓰인 로그로 갱신되었으므로, 복구될 필요가 없다.

트랜잭션의 격리 레벨

>>> 트랜잭션의 격리 레벨 종류

격리 레벨의 종류

트랜잭션의 상호 작용 종류

	DIRTY READ	NON-REPEATABLE READ	PHANTOM READ
READ UNCOMMITTED	발생	발생	발생
READ COMMITTED	발생하지 않음	발생	발생
REPEATABLE READ	발생하지 않음	발생하지 않음	발생
SERIALIZABLE	발생하지 않음	발생하지 않음	발생하지 않음

트랜잭션의 격리 레벨

>>> 트랜잭션의 상호 작용

Dirty Read:

- 아직 커밋(Commit)되지 않은 다른 트랜잭션의 데이터를 읽는 것을 의미.

Non-repeatable Read:

- 다른 트랜잭션이 커밋(Commit)한 데이터를 읽을 수 있는 것을 의미.
- 즉, 한 트랜잭션에서 같은 쿼리로 2번이상 조회했을 때 그 결과가 상이한 상황.
- 보통 데이터의 수정/삭제가 발생했을 경우 발생.

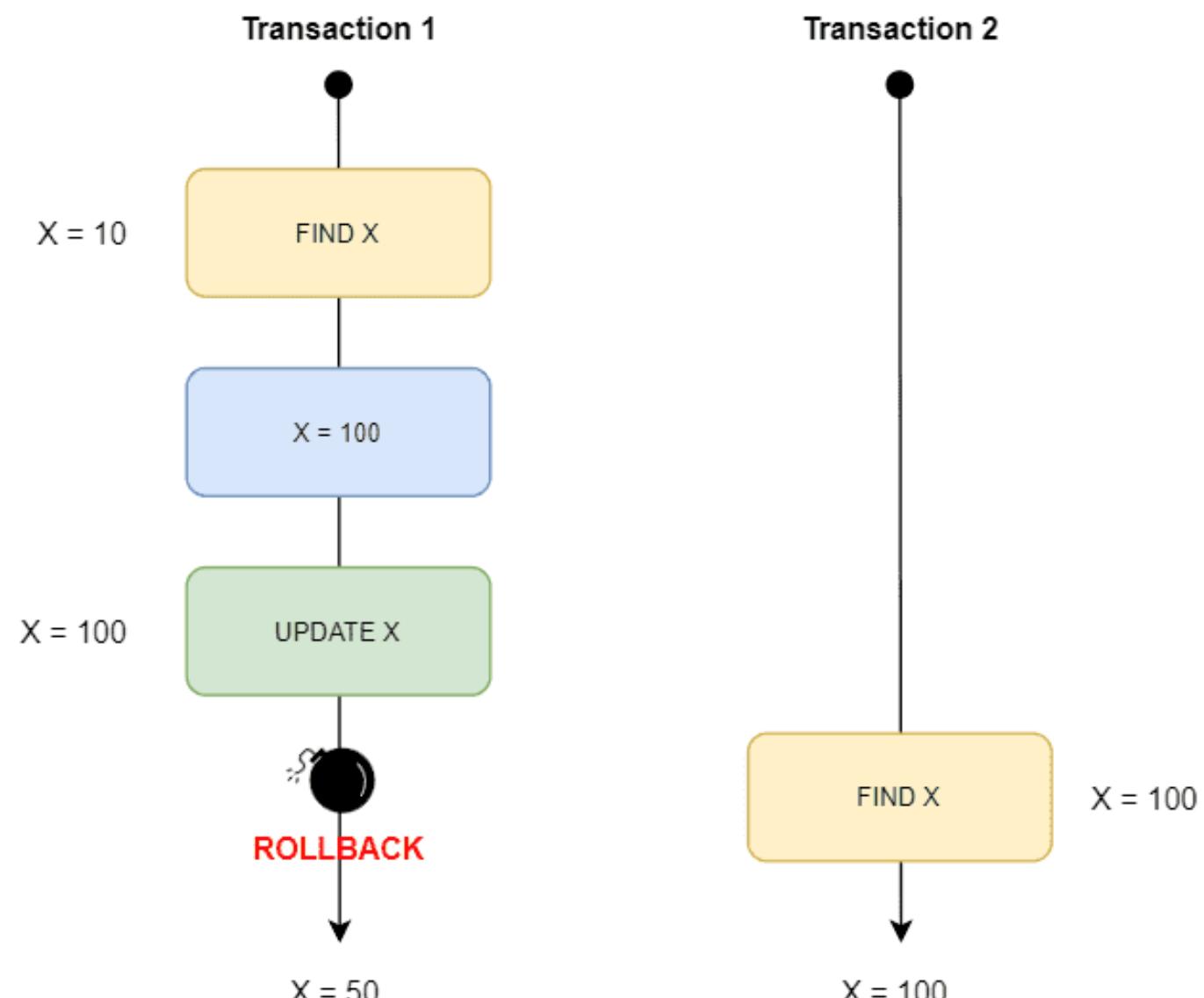
Phantom Read:

- 다른 트랜잭션이 커밋(Commit)한 데이터가 있더라도 자신의 트랜잭션에서 읽었던 내용만 사용하는 것을 의미.
- 즉, 한 트랜잭션에서 같은 쿼리를 2번이상 조회했을 때 없던 결과가 조회되는 상황.
- 보통 데이터의 삽입이 발생했을 경우 발생.

트랜잭션의 격리 레벨

>>> Read Uncommitted (Level 0)

커밋되지 않은 데이터를 다른 트랜잭션이 조회할 수 있도록 허용하는 격리 수준

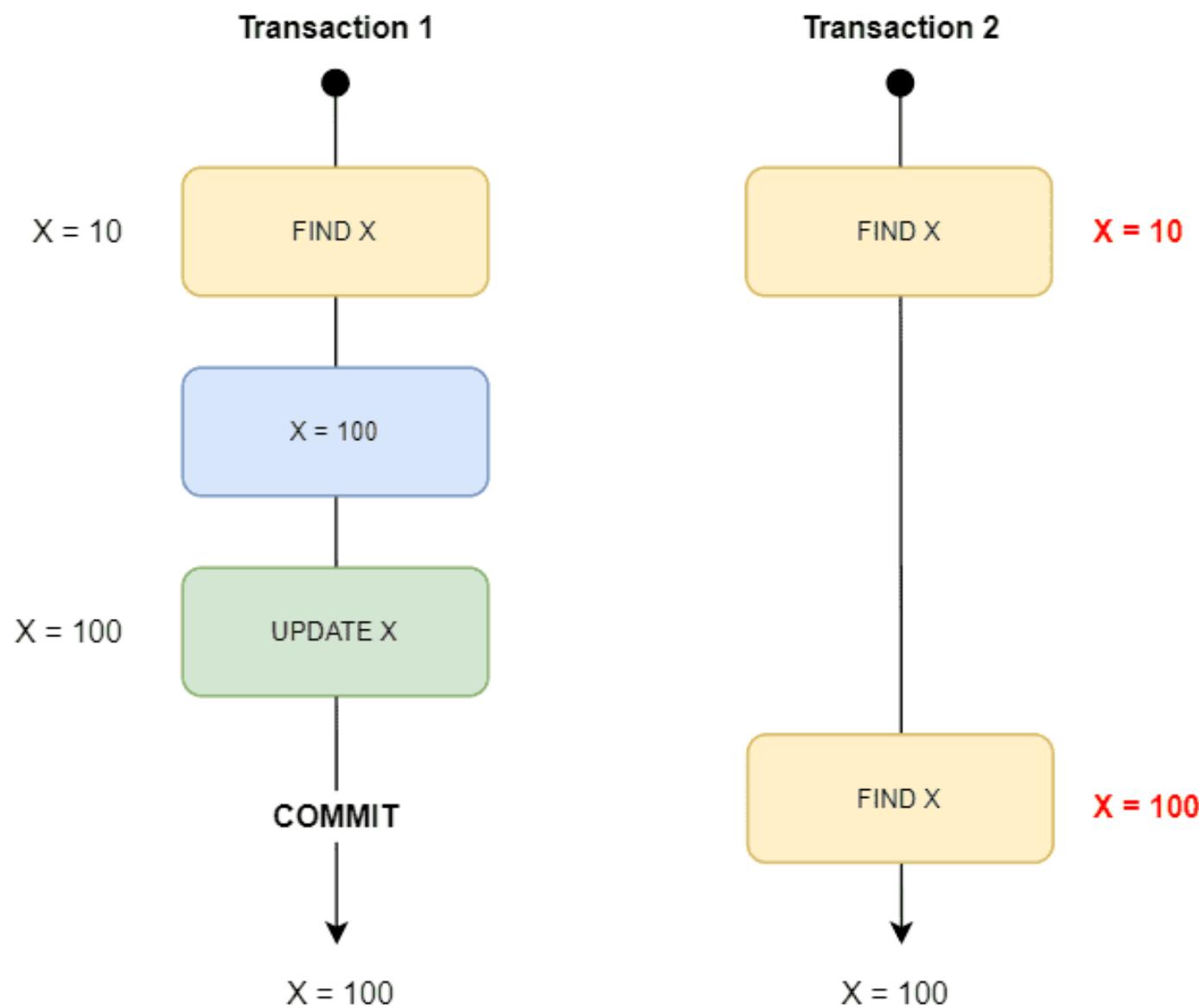


Dirty Read 발생!

트랜잭션의 격리 레벨

>>> Read Committed (Level 1)

트랜잭션의 변경 작업을 외부에서 조회할 수 없게 하는 격리 수준

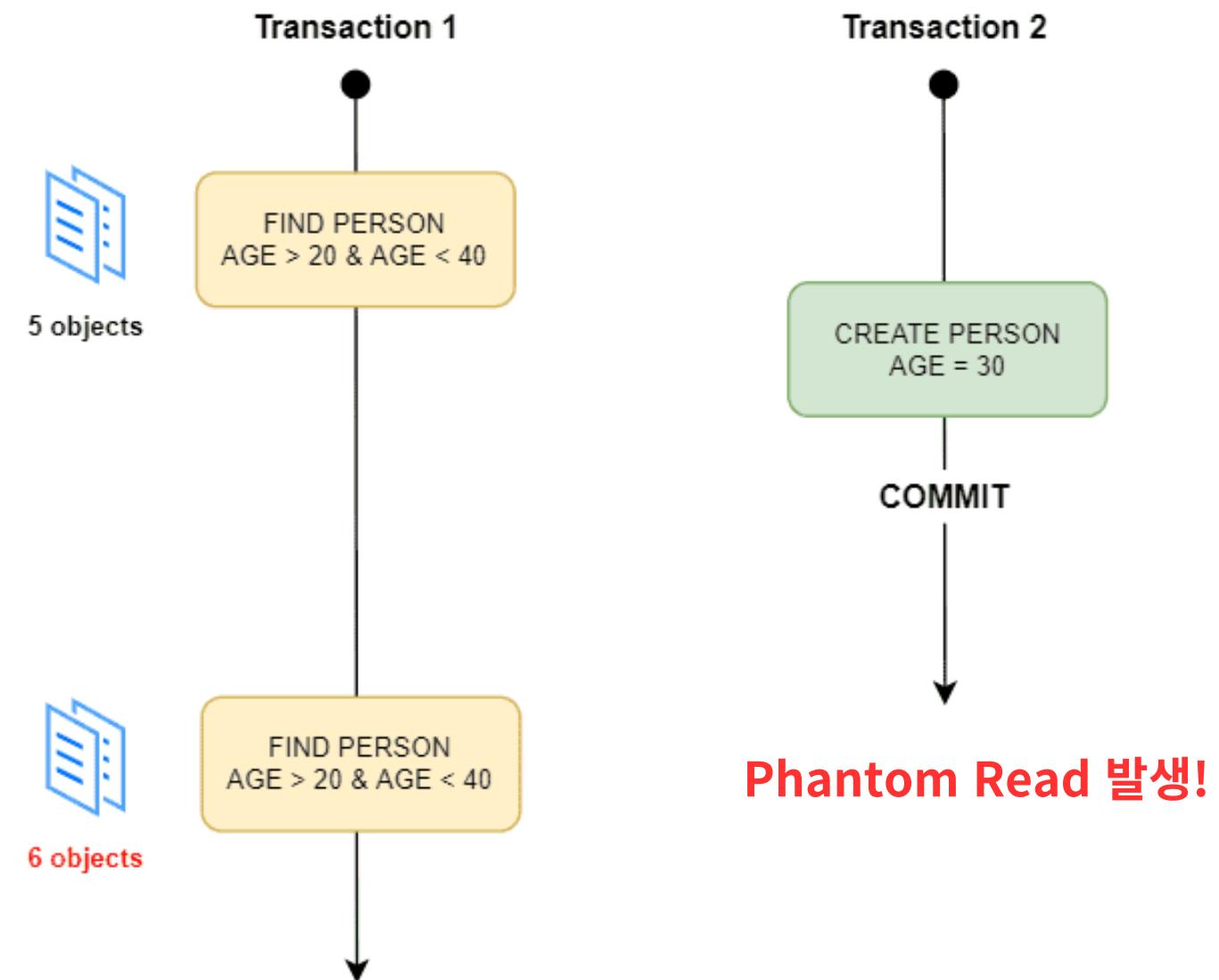


Non-repeatable Read 발생!

트랜잭션의 격리 레벨

>>> Repeatable Read (Level 2)

같은 트랜잭션 내에서 조회한 데이터의 값이 항상 동일함을 보장하는 격리 수준(Snapshot으로 해결)



트랜잭션의 격리 레벨

>>> SERIALIZABLE (Level 3)

트랜잭션의 읽기 작업 시에도 공유락을 획득하게 함으로써, 동시에 다른 트랜잭션이 같은 데이터에 접근할 수 없도록 하는 격리 레벨

모든 문제 해결, But 동시성 ↓↓↓

☰ MSA 환경의 트랜잭션

»»» Two phase commit

싱글노드에서 실행되는 DB의 원자성은 스토리지 엔진이 보장.
멀티 노드라면?

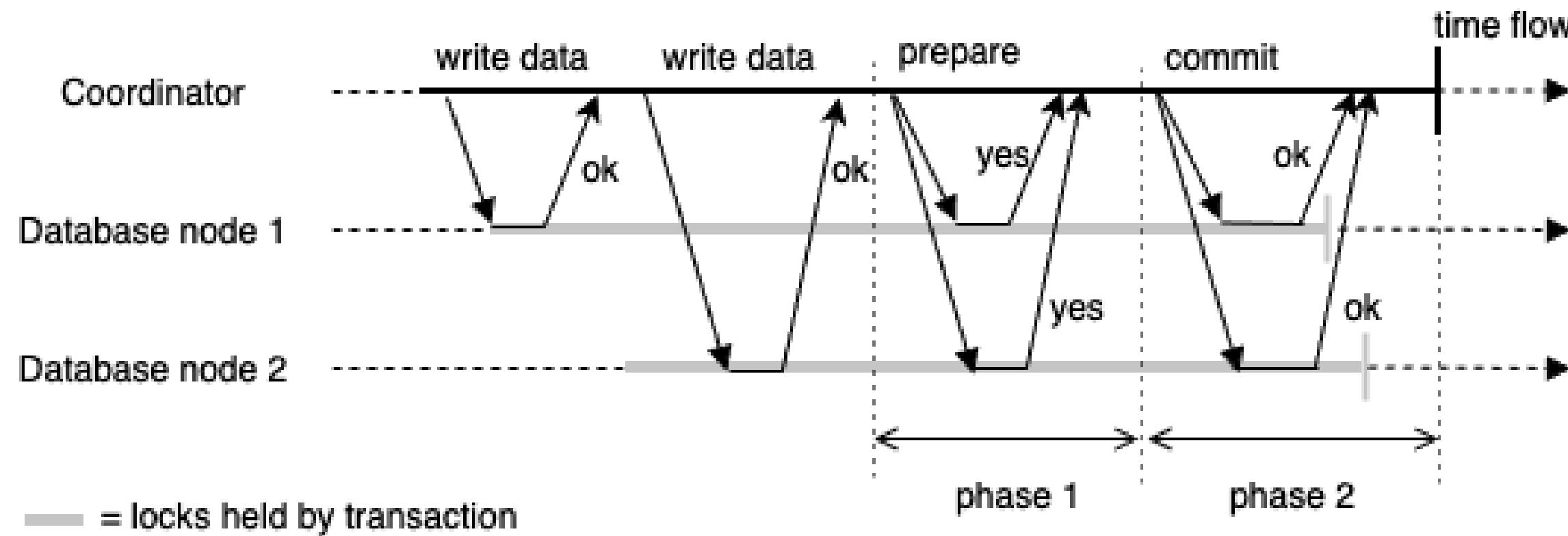
일부 노드에서만 제약위반 발생을 감지하거나,
일부 노드에 대한 커밋 요청이 네트워크 상에서 소실되거나,
일부 노드가 커밋 레코드가 write되기 전에 다운되는 등,

멀티 노드 DB의 일부 노드는 트랜잭션을 커밋하고, 일부는 하지 않는다면.. 데이터 정합성 오류 발생! ⚡

☰ MSA 환경의 트랜잭션

>>> Two phase commit

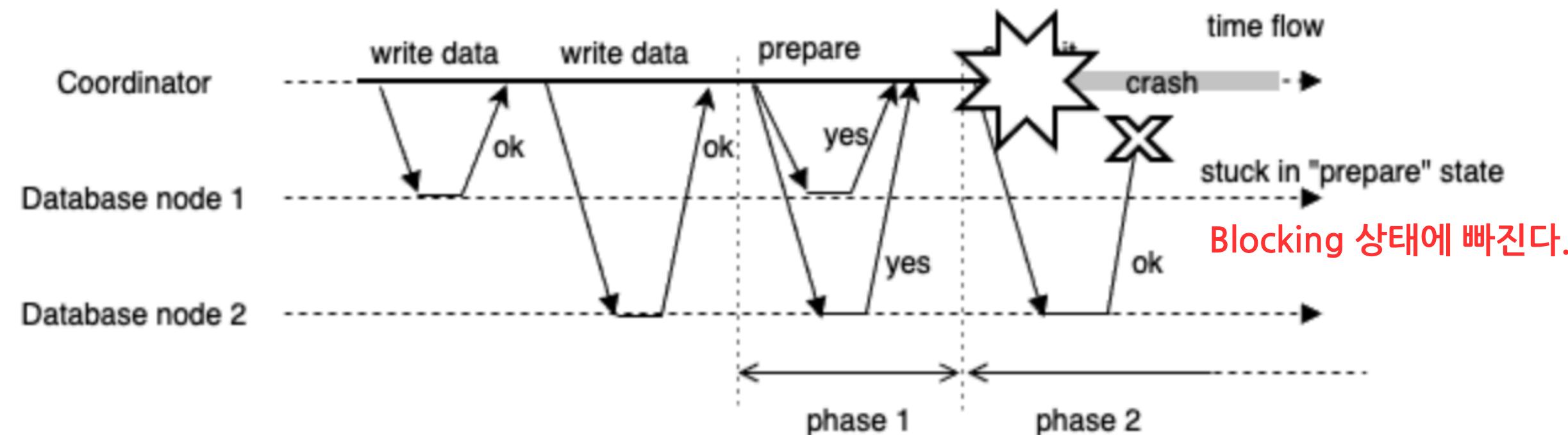
Two phase commit으로 해결



☰ MSA 환경의 트랜잭션

>>> Two phase commit

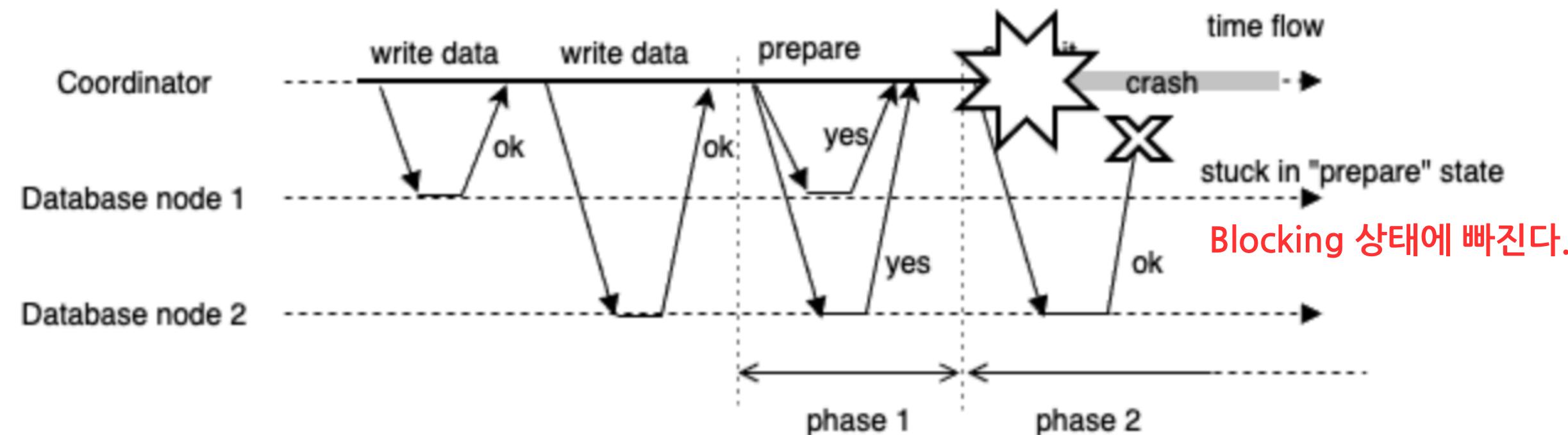
Two phase commit의 취약점



☰ MSA 환경의 트랜잭션

>>> Two phase commit

Two phase commit의 취약점



Time out, 3PC, 합의 기반 알고리즘(Paxos, Raft)로 위 문제 해결가능.

☰ MSA 환경의 트랜잭션

>>> raft Algorithm

2PC, 3PC vs Paxos

Paxos vs two phase commit

Asked 9 years, 9 months ago Modified 4 years, 3 months ago Viewed 17k times

I am trying to understand the difference between paxos and two phase commit as means to reach consensus among multiple machines. Two phase commit and three phase commit is very easy to understand. It also seems that 3PC solves the failure problem that would block in 2PC. So I don't really understand what Paxos is solving. Can anyone illuminate me about what problem does Paxos exactly solve?

database distributed-computing paxos

Share Improve this question Follow

asked Dec 4, 2014 at 21:56

Keeto
4,170 ● 9 □ 37 □ 62

Add a comment

2 Answers

Sorted by: Highest score (default) ▾

48 2PC blocks if the transaction manager fails, requiring human intervention to restart. 3PC algorithms (there are several such algorithms) try to fix 2PC by electing a new transaction manager when the original manager fails.

48 Paxos does not block as long as a majority of processes (managers) are correct. Paxos actually solves the more general problem of consensus, hence, it can be used to implement transaction commit as well. In comparison to 2PC it requires more messages, but it is resilient to manager failures. In comparison to most 3PC algorithms, Paxos renders a simpler, more efficient algorithm (minimal message delay), and has been proved to be correct.

Gray and Lamport compare 2PC and Paxos in an excellent [paper](#) titled "Consensus on Transaction Commit".

(In the answer of peter, I think he is mixing 2PC with 2PL (two-phase locking).)

Paxos vs raft

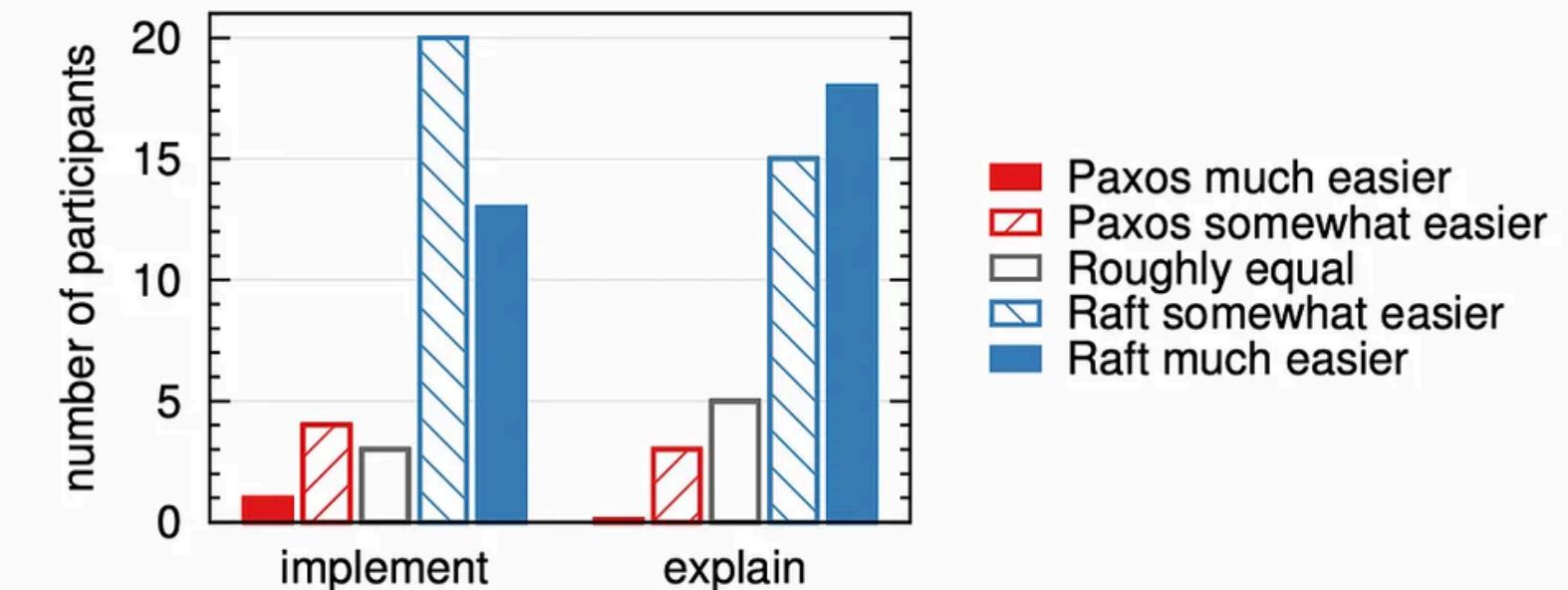


Figure 13: Using a 5-point scale, participants were asked (left) which algorithm they felt would be easier to implement in a functioning, correct, and efficient system, and (right) which would be easier to explain to a CS graduate student.

<https://stackoverflow.com/questions/27304887/paxos-vs-two-phase-commit>

☰ MSA 환경의 트랜잭션

>>> raft Algorithm

분산 시스템의 노드는 세 가지 중 하나의 상태를 가진다.

Leader : 리더는 클라이언트의 모든 요청을 수신받고, 로컬에 로그를 적재하고 모든follower에게 전달

Follower : 클라이언트로 받은 요청을 리더로 redirect 시켜주고, 리더의 요청을 수신하여 처리하는 역할

Candidate : 새로운 리더를 선출하기 위해 후보가 된 상태

AppendEntries RPC: Raft 알고리즘에서 리더가 팔로워와 통신하는 주요 방법

☰ MSA 환경의 트랜잭션

>>> raft Algorithm

1. Heart Beat

- 분산 시스템에서 리더 노드는 임기를 가지고 있고, 리더로 선출된 노드는 주기적으로 비어있는 AppendEntries라는 RPC프로토콜을 사용해 follower들에게 자신이 살아있다는 것을 알린다. (새로운 엔트리를 follower에게 전파할때도 동일한 프로토콜 사용)

2. 리더 선출

- 클러스터가 시작될 때 또는 현재의 Leader가 다운될 때, Follower 중 하나가 Candidate로 전환되고, 다른 노드들로부터 투표를 요청한다.
- 다수의 투표를 얻은 Candidate는 새로운 Leader로 선출된다.

3.로그 복제

- Client가 데이터 변경 요청을 Leader에 보내면, Leader는 로그 항목을 생성한다.
- Leader는 AppendEntries RPC를 통해 변경된 로그 항목(log entries)을 Follower에게 복제한다.
- Follower는 로그 항목을 수신하고 저장한다.
- 모든 Follower가 로그 항목을 수신하면 Leader는 트랜잭션을 커밋한다.

ACID 예상 질문

>>> ACID 원칙 중, Durability를 DBMS는 어떻게 보장하나요?

답변 | 트랜잭션과 관련된 모든 변경 사항은 로그에 먼저 기록되기 때문에, 장애 발생시 로그를 사용해 복구

ACID 예상 질문

>>> 트랜잭션을 사용해 본 경험이 있나요? 어떤 경우에 사용할 수 있나요?

답변 | 은행 거래, 재고 관리, 예약 시스템 등 한 단위로 처리해야 하는 일들을 수행할 때

ACID 예상 질문

>>> 읽기에는 트랜잭션을 걸지 않아도 될까요?

답변 | Dirty read와 Repeatable read 등을 방지하기 위해선 필요

트랜잭션 격리 레벨 예상 질문

»»» 트랜잭션 격리 레벨에 대해 설명해 주세요.

답변 | 트랜잭션 격리 레벨은 동시성 제어를 위해 트랜잭션 간의 데이터 상호작용을 제어하는 기준이다.

네 가지의 트랜잭션 격리 레벨이 존재하고, 격리 수준이 높아질 수록 읽기 오류는 줄어들지만 동시성 또한 줄어든다.

트랜잭션 격리 레벨 예상 질문

»»> 모든 DBMS가 4개의 레벨을 모두 구현하고 있나요? 그렇지 않다면 그 이유는 무엇일까요?

답변 | 각 DBMS의 Isolation 레벨 구현은 모두 다르고, 같은 수준이더라도 그 구현은 다를 수 있다.

MySql의 경우 4개의 레벨을 모두 구현.

Read-Uncommitted의 경우 일관성을 보장하지 못해 구현하지 않기도 함.

트랜잭션 격리 레벨 예상 질문

»»» 만약 MySQL을 사용하고 있다면, (InnoDB 기준) Undo 영역과 Redo 영역에 대해 설명해 주세요.

답변 | Undo 영역은 트랜잭션이 수행한 변경 작업을 되돌릴 수 있게 하는 정보를 저장하는 영역이다.

MVCC를 구현하는 데 사용.

Redo 영역은 트랜잭션이 성공적으로 완료된 후에도 해당 변경 사항을 보존하는 기능을 담당하는 영역이다.

서버가 예기치 않게 종료되면, Redo 로그를 사용하여 트랜잭션의 커밋 상태를 재구성하고 데이터 일관성을 유지

트랜잭션 격리 레벨 예상 질문

>>> 그런데, 스토리지 엔진이 정확히 무엇을 하는 건가요?

답변 | DB에서 데이터를 어떻게 저장하고 접근할 것인지에 대한 기능을 제공

THANK YOU

GITHUB

<https://github.com/CS-Computer-Science-Study/Database>