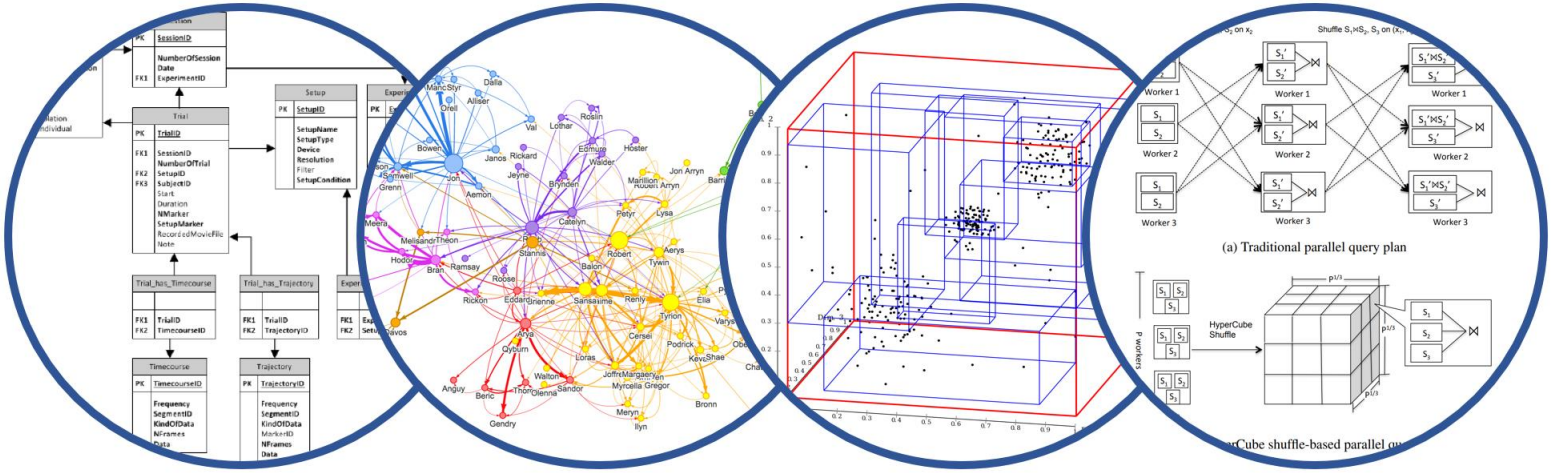


Introduction to Data Management

Transactions: Isolation Levels

Paul G. Allen School of Computer Science and Engineering
University of Washington, Seattle

Announcements

- HW5 is due tonight
- HW6 is posted:
  - Milestone 1 is due next Friday. **NO LATE DAYS**
  - Milestone 2 is due on Wednesday, 11/27

Lock Types

Shared/Exclusive Locks

Reads don't conflict with each other.

- Exclusive/Write Lock → $\mathbf{X_i(A)}$
  - May read or write
  - No other locks may exist
- Shared/Read Lock → $\mathbf{S_i(A)}$
  - May only read
  - May exist with other shared locks
- Unlocked
  - No access

Shared/Exclusive Locks

…but another TXN holds this…

If a TXN requests this…

	unlocked	S	X
S	Yes	Yes	No
X	Yes	No	No

…then we do or don’t grant permission

Discussion

- When TXN wants to read A, it requests S(A)
- If later it wants to write A, then it requests X(A)
- This is a form of lock escalation:
  - Lock escalation: fine grained → coarse grained

Example

T1	T2
S1(A), **READ**(A, t)	
t := t+100	
	S2(A), **READ**(A, s)
	s := s*2

Example

T1	T2
S1(A), **READ**(A, t)	
t := t+100	
	S2(A), **READ**(A, s)
	s := s*2
X1(A)	
WRITE(A, t)	

Example

T1	T2
S1(A), **READ**(A, t)	
t := t+100	
	S2(A), **READ**(A, s)
	s := s*2
X1(A)	
WRITE(A, t)	

Denied

Example

T1	T2
S1(A), **READ**(A, t)	
t := t+100	
	S2(A), **READ**(A, s)
	s := s*2
X1(A)	
WRITE(A, t)	
	COMMIT **U**(A)

Denied

Example

T1	T2
S1(A), **READ**(A, t)	
t := t+100	
	S2(A), **READ**(A, s)
	s := s*2
X1(A)	
WRITE(A, t)	
	COMMIT **U**(A)
…**X1**(A)	

Denied

Granted

Example

T1	T2
S1(A), **READ**(A, t)	
t := t+100	
	S2(A), **READ**(A, s)
	s := s*2
X1(A)	
WRITE(A, t)	
	COMMIT **U**(A)
…**X1**(A)	
WRITE(A,t)	

Denied

Granted

Starvation

- When a TXN waits for a lock, and never gets it
- Usually prevented by placing TXN in a queue
- Need to pay more attention to S/X locks
  - Some TXNs hold an S lock
  - One TXN requests X lock and waits
  - But more TXNs arrive and requests S locks, granted
  - Solution: stop granting S locks when X requests exists

Lock Escalation May Deadlock

- Shared/Exclusive locks increase the likelihood of deadlocks (next)

Lock Escalation May Deadlock

T1	T2
S1(A), **READ**(A, t)	
t := t+100	
	S2(A), **READ**(A, s)
	s := s*2
X1(A)	
WRITE(A, t)	**X2**(A)
	WRITE(A,s)

Denied

Denied

Dealock

Discussion

- All DBMS that use a locking-based CC implement multiple types of locks
- This usually increases the degree of concurrency, e.g. READ ONLY transactions don’t wait
- Lock escalation:
  - From more permissive to stricter lock
  - E.g. shared lock to exclusive lock

Next Topic

Weaker isolation levels :

- Increase TPS by giving up on serializability
- But what exactly do they guarantee?
  - Imprecise: they just avoid certain conflicts
  - Formal definition is operational, using locks

Conflicts Between Concurrent Operations

Common Concurrency Conflicts

These never happen in serializable schedules, but may happen in weaker levels of isolation

- Dirty/Inconsistent Read
- Lost Update
- Unrepeatable Read
- Phantom Read

Dirty/Inconsistent Read

- **Dirty/Inconsistent Read**
- Lost Update
- Unrepeatable Read
- Phantom Read

Dirty read reading data of uncommitted TXN
a.k.a. inconsistent read

time

Manager wants to balance project budgets

CEO wants to check company balance

-$10mil from project A

+$7mil to project B

+$3mil to project C

Dirty/Inconsistent Read

- **Dirty/Inconsistent Read**
- Lost Update
- Unrepeatable Read
- Phantom Read

Dirty read reading data of uncommitted TXN
a.k.a. inconsistent read

time

Manager wants to balance project budgets

-$10mil from project A

+$7mil to project B

+$3mil to project C

CEO wants to check company balance

SELECT SUM(money) …

Dirty/Inconsistent Read

- **Dirty/Inconsistent Read**
- Lost Update
- Unrepeatable Read
- Phantom Read

Dirty read reading data of uncommitted TXN
a.k.a. inconsistent read

time

Manager wants to balance project budgets

-$10mil from project A

+$7mil to project B

+$3mil to project C

CEO wants to check company balance

SELECT SUM(money) …

Dirty/Inconsistent Read

- **Dirty/Inconsistent Read**
- Lost Update
- Unrepeatable Read
- Phantom Read

Dirty read reading data of uncommitted TXN
a.k.a. inconsistent read

time

Manager wants to balance project budgets

CEO wants to check company balance

SELECT SUM(money) …

-$10mil from project A

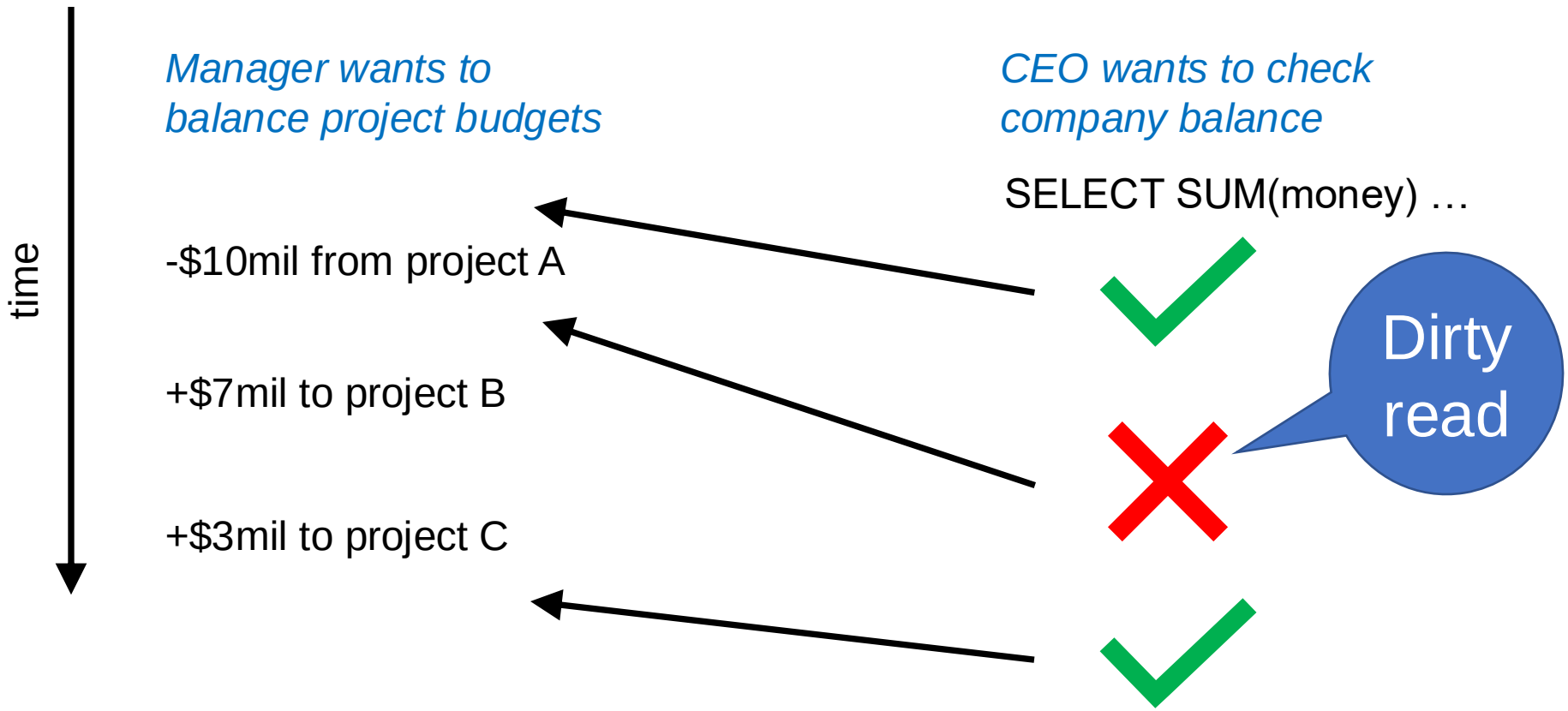
+$7mil to project B

+$3mil to project C

Dirty/Inconsistent Read

- **Dirty/Inconsistent Read**
- Lost Update
- Unrepeatable Read
- Phantom Read

Dirty read reading data of uncommitted TXN
a.k.a. inconsistent read



Lost Update

- Dirty/Inconsistent Read
- **Lost Update**
- Unrepeatable Read
- Phantom Read

A **lost update** happens when a write is overwritten by another TXN

time

Account 1 = 100, Account 2 = 100

User 1 wants to pool money into account 1

User 2 wants to pool money into account 2

Set account 1 = 200

Set account 2 = 0

Lost Update

- Dirty/Inconsistent Read
- **Lost Update**
- Unrepeatable Read
- Phantom Read

A **lost update** happens when a write is overwritten by another TXN

time

Account 1 = 100, Account 2 = 100

User 1 wants to pool money into account 1	*User 2 wants to pool money into account 2*
Set account 1 = 200	
Set account 2 = 0	
	Set account 2 = 200
	Set account 1 = 0

Lost Update

- Dirty/Inconsistent Read
- **Lost Update**
- Unrepeatable Read
- Phantom Read

A **lost update** happens when a write is overwritten by another TXN

time

Account 1 = 100, Account 2 = 100

User 1 wants to pool money into account 1	*User 2 wants to pool money into account 2*
Set account 1 = 200	
Set account 2 = 0	
	Set account 2 = 200
	Set account 1 = 0

At end: Account 1 = 0, Account 2 = 200

Lost Update

- Dirty/Inconsistent Read
- **Lost Update**
- Unrepeatable Read
- Phantom Read

A **lost update** happens when a write is overwritten by another TXN

time

Account 1 = 100, Account 2 = 100

User 1 wants to pool money into account 1	*User 2 wants to pool money into account 2*
Set account 1 = 200	
	Set account 2 = 200
Set account 2 = 0	
	Set account 1 = 0

Lost Update

- Dirty/Inconsistent Read
- **Lost Update**
- Unrepeatable Read
- Phantom Read

A **lost update** happens when a write is overwritten by another TXN

Account 1 = 100, Account 2 = 100

User 1 wants to pool money into account 1

User 2 wants to pool money into account 2

Lost update

time

Set account 1 = 200

Set account 2 = 200

Set account 2 = 0

Set account 1 = 0

At end: Account 1 = 0, Account 2 = 0

Unrepeatable Read

- Dirty/Inconsistent Read
- Lost Update
- **Unrepeatable Read**
- Phantom Read

An **unrepeatable read** happens when data read twice differs

time

Accountant wants to check company assets

Warehouse updates inventory levels

```
SELECT inventory
FROM Products
WHERE pid = 1
```

```
SELECT inventory*price
FROM Products
WHERE pid = 1
```

Unrepeatable Read

- Dirty/Inconsistent Read
- Lost Update
- **Unrepeatable Read**
- Phantom Read

An **unrepeatable read** happens when data read twice differs

time

Accountant wants to check company assets

SELECT inventory
FROM Products
WHERE pid = 1

Warehouse updates inventory levels

UPDATE Products
SET inventory = 0
WHERE pid = 1

SELECT inventory*price
FROM Products
WHERE pid = 1

Unrepeatable Read

- Dirty/Inconsistent Read
- Lost Update
- **Unrepeatable Read**
- Phantom Read

An **unrepeatable read** happens when data read twice differs

time

Accountant wants to check company assets

SELECT inventory
FROM Products
WHERE pid = 1

Warehouse updates inventory levels

UPDATE Products
SET inventory = 0
WHERE pid = 1

SELECT inventory*price
FROM Products
WHERE pid = 1

Second read of Products.inventory is different

Phantom Read

- Dirty/Inconsistent Read
- Lost Update
- Unrepeatable Read
- **Phantom Read**

A **phantom read** happens when
a record is inserted/delete during reads

time

Accountant wants to check company assets

Warehouse receives new products

```
SELECT *
FROM products
WHERE price < 10.00
```

```
SELECT *
FROM products
WHERE price < 20.00
```

Phantom Read

- Dirty/Inconsistent Read
- Lost Update
- Unrepeatable Read
- **Phantom Read**

A **phantom read** happens when
a record is inserted/delete during reads

time

Accountant wants to check company assets

```
SELECT *
FROM products
WHERE price < 10.00
```

Warehouse receives new products

```
INSERT INTO Products
VALUES ('nuts', 10, 8.99)
```

```
SELECT *
FROM products
WHERE price < 20.00
```

Phantom Read

- Dirty/Inconsistent Read
- Lost Update
- Unrepeatable Read
- **Phantom Read**

A **phantom read** happens when
a record is inserted/delete during reads

time

Accountant wants to check company assets

Warehouse receives new products

```
SELECT *
FROM products
WHERE price < 10.00
```

```
INSERT INTO Products
VALUES ('nuts', 10, 8.99)
```

```
SELECT *
FROM products
WHERE price < 20.00
```

Returns a product that should have been in the first query

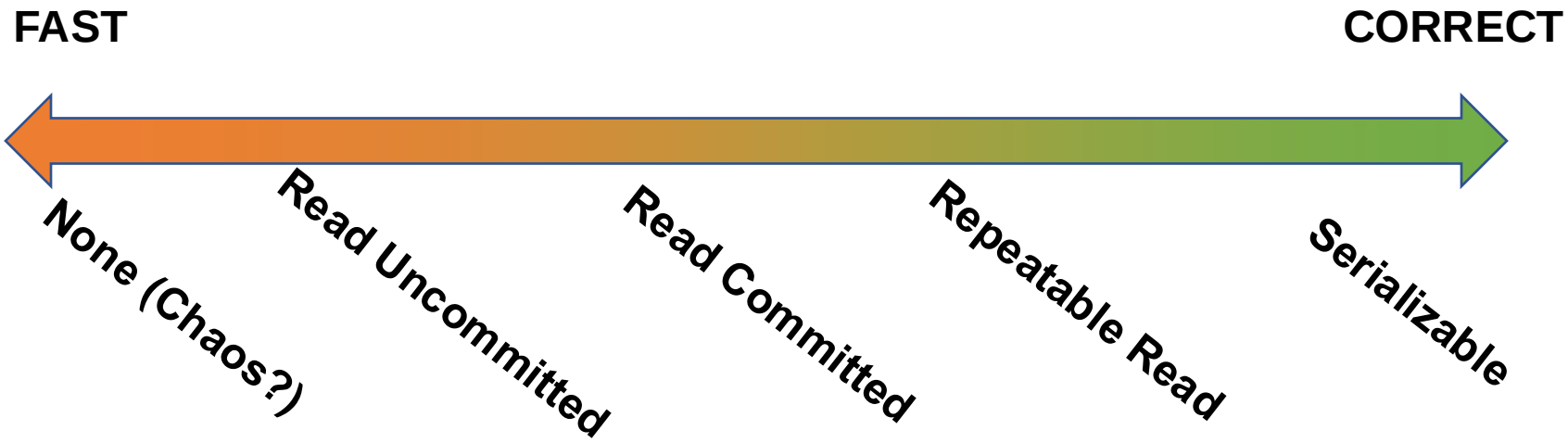
Weaker Isolation Levels

Isolation Levels

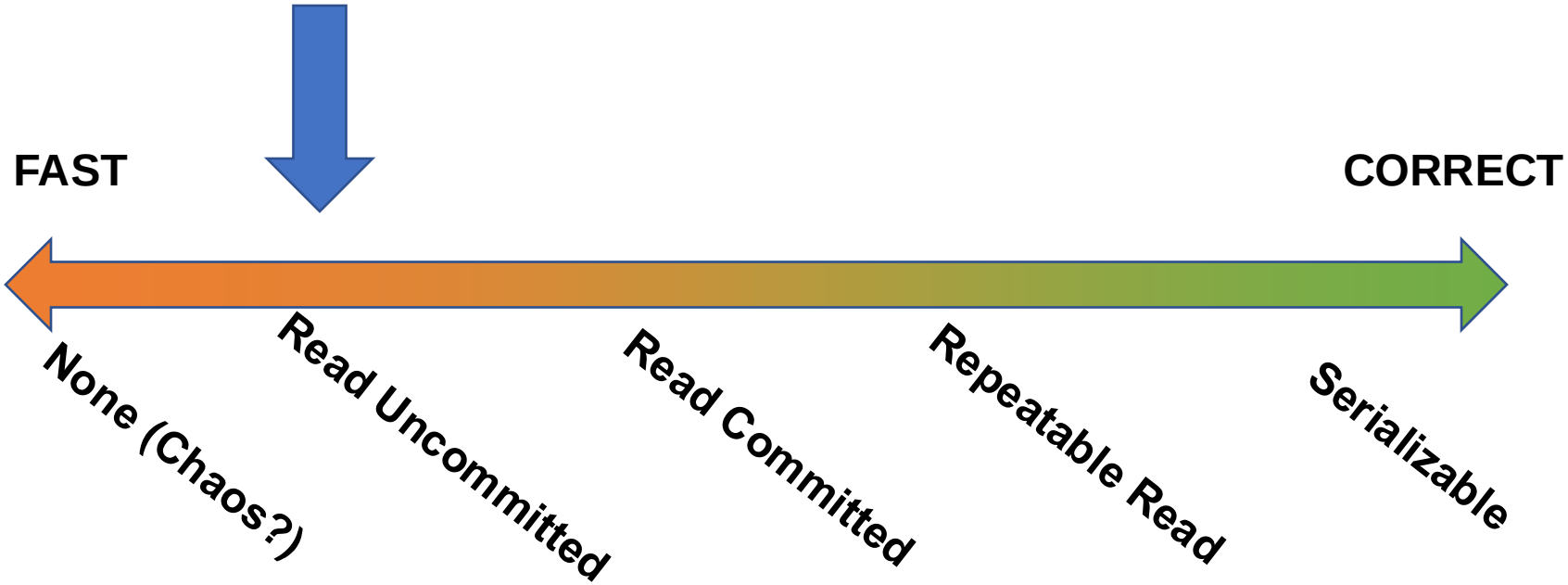
- **`SET TRANSACTION ISOLATION LEVEL ...`**
  - **`READ UNCOMMITED`**
  - **`READ COMMITED`**
  - **`REPEATABLE READ`**
  - **`SERIALIZABLE`**
  - `SNAPSHOT ISOLATION` (MVCC)

- Default is not always SERIALIZABLE: see doc

Isolation Level Design Spectrum



Isolation Level Design Spectrum



READ UNCOMMITTED

- Writes → Strict 2PL write locks
- Reads → No locks needed
- Reads never wait! But dirty reads are possible

T1	T2
X(A) W(A)	
	R(A)
	COMMIT
ABORT U(A)	

READ UNCOMMITTED

- Writes → Strict 2PL write locks
- Reads → No locks needed
- Reads never wait! But dirty reads are possible

Write lock obeys Strict 2PL

Read executes whenever

T1	T2
X(A) W(A)	
	R(A)
	COMMIT
ABORT U(A)	

READ UNCOMMITTED

- Writes → Strict 2PL write locks
- Reads → No locks needed
- Reads never wait! But dirty reads are possible

Still possible to get isolated results, but you have to be "lucky" when a write operation is done

T1	T2
X(A) W(A)	
ABORT U(A)	
	R(A)
	COMMIT

Serial

READ UNCOMMITTED

- Writes → Strict 2PL write locks
- Reads → No locks needed
- Reads never wait! But dirty reads are possible

Still possible to get isolated results, but you have to be "lucky" when a write operation is done

T1	T2
X(A) W(A)	
ABORT U(A)	
	R(A)
	COMMIT

Serial

T1	T2
X(A) W(A)	
	R(A)
	COMMIT
ABORT U(A)	

Non-serializable

READ UNCOMMITTED

- Writes → Strict 2PL write locks
- Reads → No locks needed
- Reads never wait! But dirty reads are possible

Still possible to get isolated results, but you have to be "lucky" when a write operation is done

T1	T2
X(A) W(A)	
ABORT U(A)	
	R(A)
	COMMIT

Serial

T1	T2
X(A) W(A)	
	R(A)
	COMMIT
ABORT U(A)	

Non-serializable

T1	T2
	R(A)
X(A) W(A)	
ABORT U(A)	
	COMMIT

Serializable (lucky!)

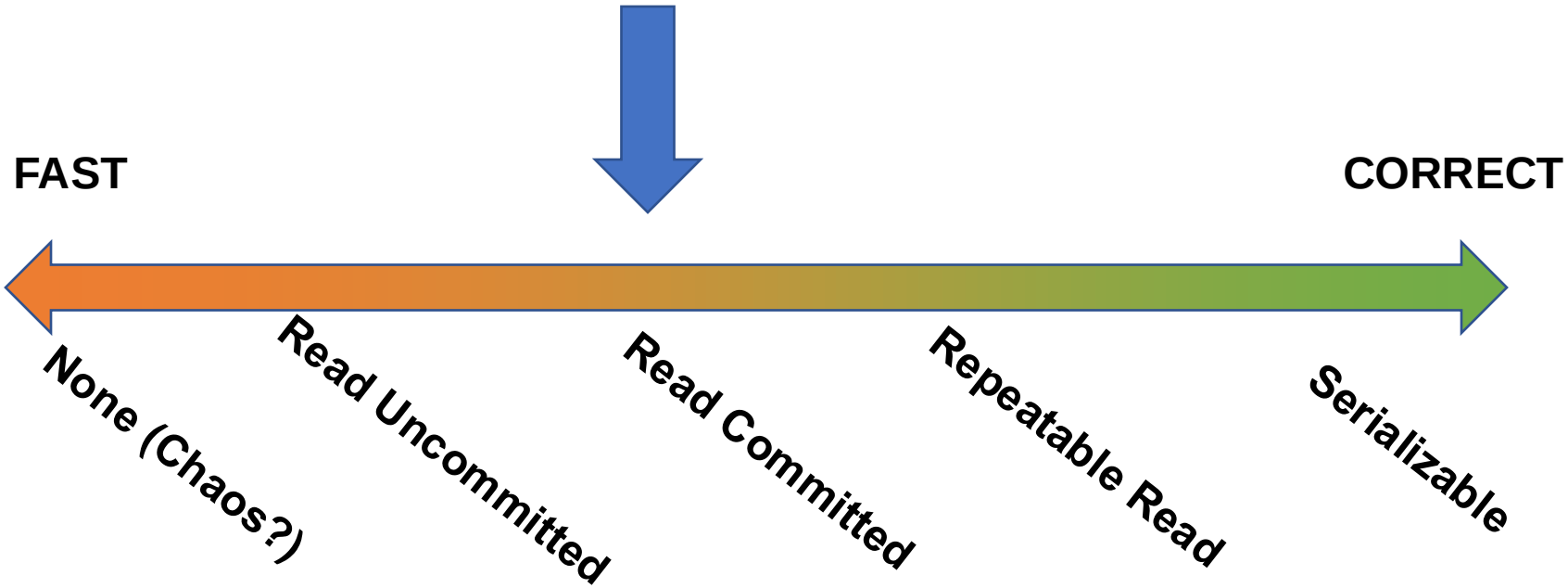
READ UNCOMMITTED

Fast READ due to zero lock management overhead

Use cases:

- Static data (few or no writes after data initialization)
- Read coverage/accuracy is not mission critical

Isolation Level Design Spectrum



READ COMMITTED

- Writes → Strict 2PL write locks
- Reads → Short-duration read locks
  - Acquire lock right before, release right after (not 2PL)
- No dirty reads. But non-repeatable reads possible.

READ COMMITTED

- Writes → Strict 2PL write locks
- Reads → Short-duration read locks
  - Acquire lock right before, release right after (not 2PL)
- No dirty reads. But non-repeatable reads possible.

T1	T2
X(A) W(A)	
	R(A)
	COMMIT
ABORT U(A)	

READ COMMITTED

- Writes → Strict 2PL write locks
- Reads → Short-duration read locks
  - Acquire lock right before, release right after (not 2PL)
- No dirty reads. But non-repeatable reads possible.

A dirty read could only happen if a read occurs <u>after</u> a write and <u>before</u> a COMMIT/ROLLBACK

T1	T2
X(A) W(A)	
	R(A)
	COMMIT
ABORT U(A)	

READ COMMITTED

- Writes → Strict 2PL write locks
- Reads → Short-duration read locks
  - Acquire lock right before, release right after (not 2PL)
- No dirty reads. But non-repeatable reads possible.

A dirty read could only happen if a read occurs <u>after</u> a write and <u>before</u> a COMMIT/ROLLBACK

T1	T2
X(A) W(A)	
	R(A)
	COMMIT
ABORT U(A)	

T1	T2
X(A) W(A)	
	S(A) blocked…
ABORT U(A)	…granted S(A)
	R(A)
	COMMIT U(A)

READ COMMITTED

- Writes → Strict 2PL write locks
- Reads → Short-duration read locks
  - Acquire lock right before, release right after (not 2PL)
- No dirty reads.

READ COMMITTED

- Writes → Strict 2PL write locks
- Reads → Short-duration read locks
  - Acquire lock right before, release right after (not 2PL)
- No dirty reads.
  But non-repeatable reads possible.

T1	T2
	R(A)
W(A)	
COMMIT U(A)	
	R(A)
	COMMIT U(A)

READ COMMITTED

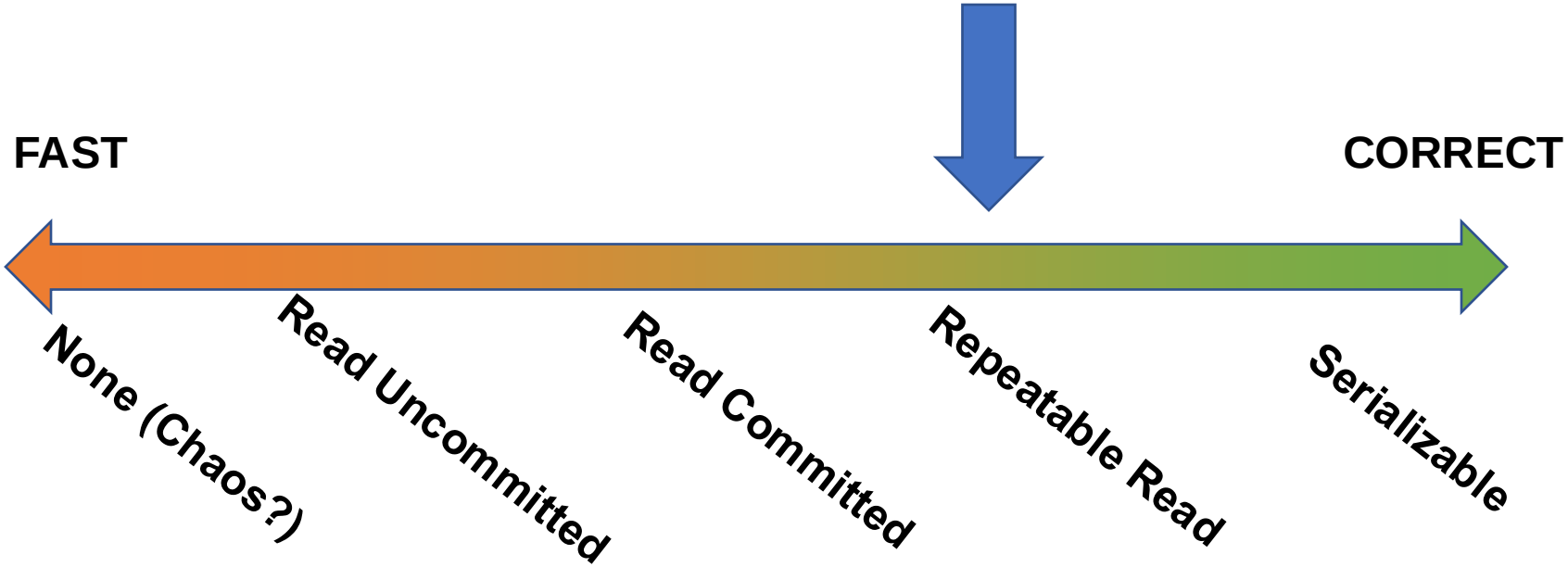
- Writes → Strict 2PL write locks
- Reads → Short-duration read locks
  - Acquire lock right before, release right after (not 2PL)
- No dirty reads.
  But non-repeatable reads possible.

T1	T2
	S(A)
X(A) blocked…	
…	**R(A)**
…granted X(A)	U(A)
	S(A) blocked…
W(A)	…
COMMIT U(A)	…granted S(A)
	R(A)
	X(A)
	W(A)
	COMMIT U(A)

READ COMMITTED

- Fast READ since operation happens as soon as write txns are done
- Use cases:
  - Guarantee that read result is valid at some point
  - Often useful for e-commerce situations
    - Guarantee customer has good info to start with but doesn't block other customers from purchasing

Isolation Level Design Spectrum



REPEATABLE READ

- Writes → Strict 2PL write locks
- Reads → Strict 2PL read locks
- Unrepeatable reads are prevented

REPEATABLE READ

- Writes → Strict 2PL write locks
- Reads → Strict 2PL read locks
- Unrepeatable reads are prevented

T1	T2
	S(A)
X(A) blocked…	
…	**R(A)**
…granted X(A)	U(A)
	S(A) blocked…
W(A)	…
COMMIT U(A)	…granted S(A)
	R(A)
	COMMIT U(A)

REPEATABLE READ

- Writes → Strict 2PL write locks
- Reads → Strict 2PL read locks
- Unrepeatable reads are prevented

T1	T2
	S(A)
X(A) blocked…	
…	**R(A)**
…granted X(A)	U(A)
	S(A) blocked…
W(A)	…
COMMIT U(A)	…granted S(A)
	R(A)
	COMMIT U(A)

T1	T2
	S(A)
X(A) blocked…	
…	**R(A)**
…	**R(A)**
…granted X(A)	COMMIT U(A)
W(A)	
COMMIT U(A)	

REPEATABLE READ

- Writes → Strict 2PL write locks
- Reads → Strict 2PL read locks

Conflict serializable!

- Unrepeatable reads are prevented

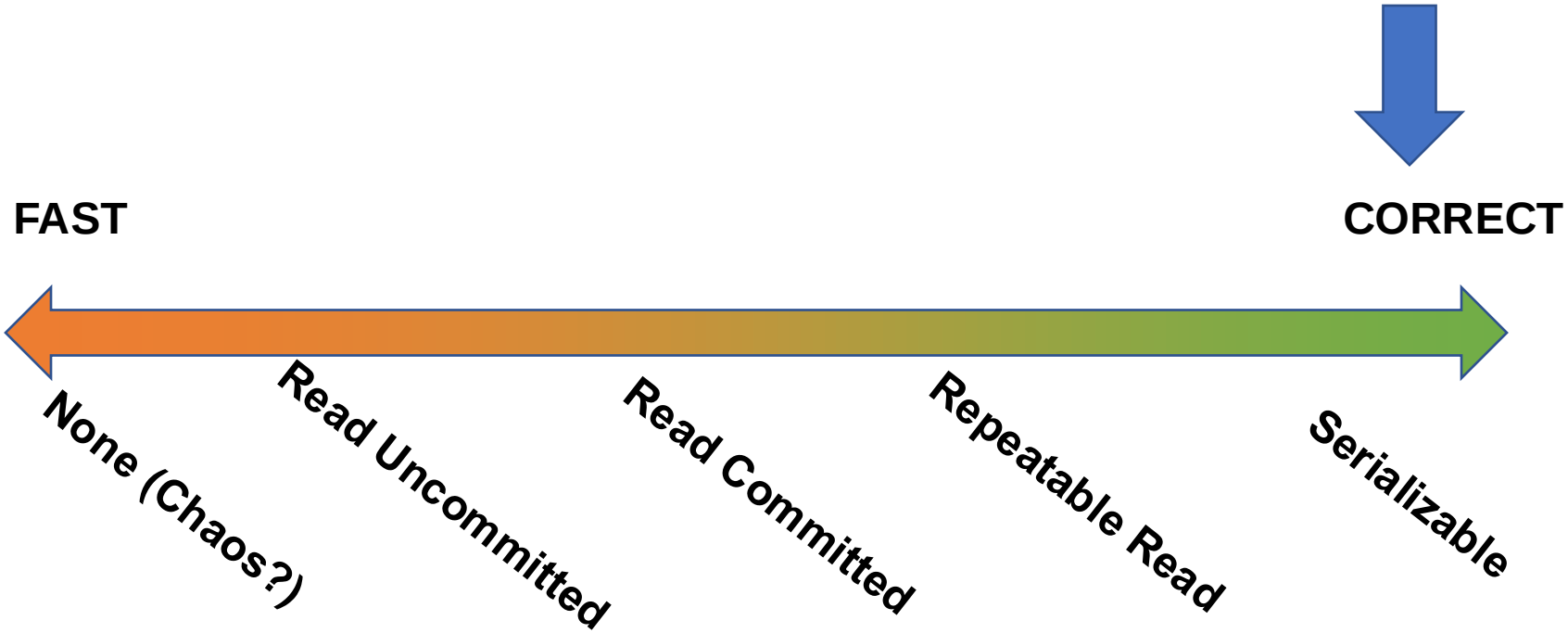
T1	T2
	S(A)
X(A) blocked…	
…	**R(A)**
…granted X(A)	U(A)
	S(A) blocked…
W(A)	…
COMMIT U(A)	…granted S(A)
	R(A)
	COMMIT U(A)

T1	T2
	S(A)
X(A) blocked…	
…	**R(A)**
…	**R(A)**
…granted X(A)	COMMIT U(A)
W(A)	
COMMIT U(A)	

REPEATABLE READ

- Ensures conflict serializability
- Recall: if the database is static (no insert/delete) then conflict serializability implies serializability
- Use cases: few insert/deletes

Isolation Level Design Spectrum



The Phantom Menace

- Same read has more rows
- Asset checking scenario:

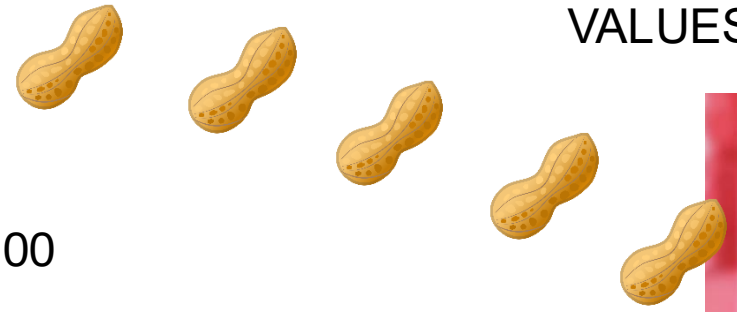
time

Accountant wants to check company assets

```
SELECT *
FROM products
WHERE price < 10.00
```

Warehouse catalogs new products

```
INSERT INTO Products
VALUES ('nuts', 10, 8.99)
```



```
SELECT *
FROM products
WHERE price < 20.00
```

Phantom Reads

- Conflict serializability does not prevent phantoms.


```
SELECT * FROM Table;

                                INSERT INTO Table
                                VALUES (C…);
SELECT * FROM Table;
```

Phantom Reads

- Conflict serializability does not prevent phantoms.

These are the SQL queries

And this is how we modeled the TXNs using R/W to elements

	T1	T2
SELECT * FROM Table;	R(A)	
	R(B)	
		I(C)
SELECT * FROM Table;	R(A)	
	R(B)	
	R(C)	

INSERT INTO Table VALUES (C…);

Phantom Reads

- Conflict serializability does not prevent phantoms.

	T1	T2	
SELECT * FROM Table;	R(A)		
	R(B)		
		I(C)	INSERT INTO Table VALUES (C…);
SELECT * FROM Table;	R(A)		
	R(B)		
	R(C)		

Phantom Reads

- Conflict serializability does not prevent phantoms.

A conflict-serializable schedule!

	T1	T2	
SELECT * FROM Table;	R(A)		
	R(B)		
		I(C)	INSERT INTO Table VALUES (C…);
SELECT * FROM Table;	R(A)		
	R(B)		
	R(C)		

Phantom Reads

- Conflict serializability does not prevent phantoms.

A conflict-serializable schedule!

What is the equivalent serial schedule?

	T1	T2
SELECT * FROM Table;	R(A)	
	R(B)	
		I(C)
SELECT * FROM Table;	R(A)	
	R(B)	
	R(C)	

INSERT INTO Table VALUES (C…);

Recap

In a static database:

- Conflict serializability implies serializability

In a dynamic database:

- This no longer holds: we need to handle phatoms

SERIALIZABLE Level

- Write Lock → Strict 2PL
- Read Lock → Strict 2PL
- Locks on tables to handle phantom problem

SERIALIZABLE Level

- Write Lock → Strict 2PL
- Read Lock → Strict 2PL
- Locks on tables to handle phantom problem

T1	T2
R(A)	
R(B)	
	I(C)
R(A)	
R(B)	
R(C)	

SERIALIZABLE Level

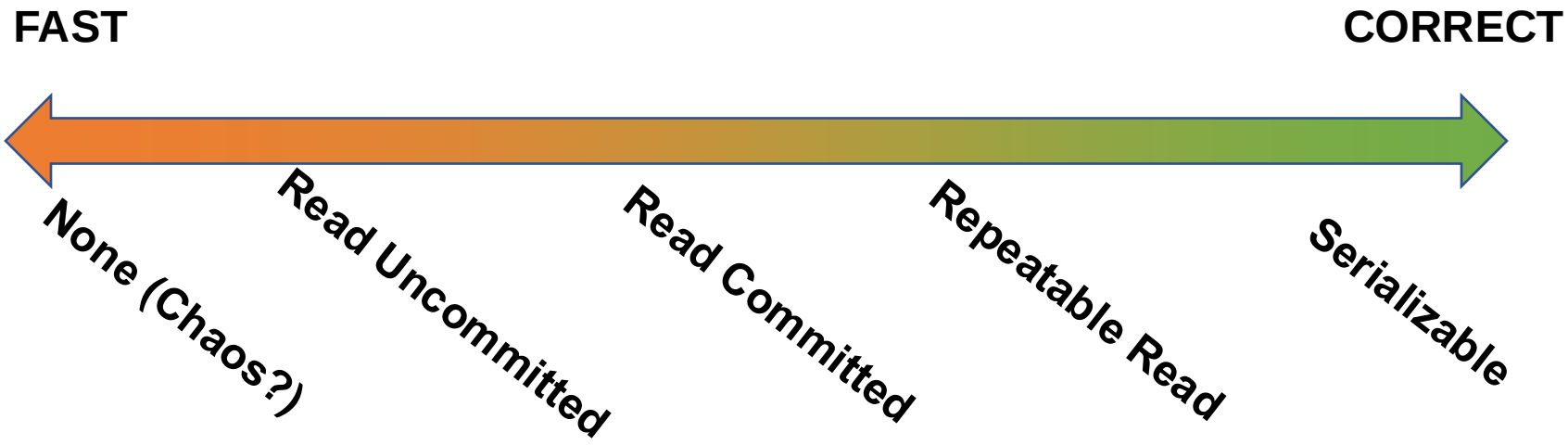
- Write Lock → Strict 2PL
- Read Lock → Strict 2PL
- Locks on tables to handle phantom problem

T1	T2
R(A)	
R(B)	
	I(C)
R(A)	
R(B)	
R(C)	

Change element granularity to Table

T1	T2
S(T)	
R(T)	
	X(T) blocked…
R(T)	…
COMMIT U(T)	…granted X(T)
	W(T)
	COMMIT U(T)

Summary


FAST
CORRECT
None (Chaos?)
Read Uncommitted
Read Committed
Repeatable Read
Serializable