

Transaction locks

TXN sequence of reads & writes

isolation 2 TXN must not interfere

schedule interleaving of TXN ops

serial sch. one at a time

serializable sch. equiv. to some serial sch.

conflict-serializable sch. conflict-equivalent to serial

precedence graph acyclic \rightarrow conflict-serializable

TXN sequence of reads & writes

isolation 2 TXN must not interfere

schedule interleaving of TXN ops

how to get this?

serial sch. one at a time

serializable sch. equiv. to some serial sch.

conflict-serializable sch. conflict-equivalent to serial

precedence graph acyclic → conflict-serializable

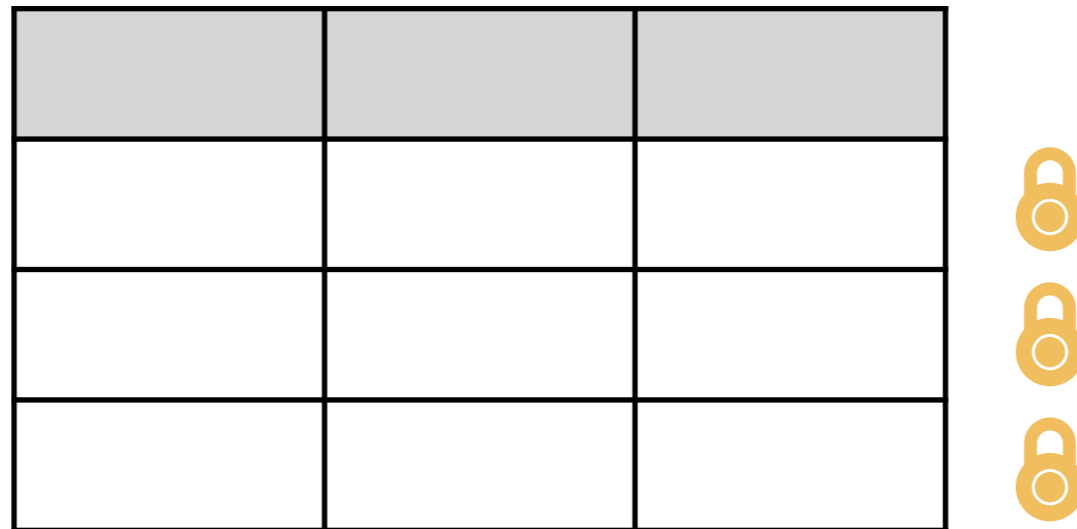
deterministic scheduling

collect a batch of TXNs

compute serializable schedule

locking

TXNs lock & unlock DB items



enforce serial

T1	T2
L(DB)	
...	
U(DB)	
	L(DB)
	...
	U(DB)

"only lock what you need"

T1	T2
L(A)	
W(A)	L(B)
U(A)	W(B)
	U(B)

$A = B = 10$

$x = ??$

$L(A), x = R(A)$	
$W(A, x), U(A)$	
	$L(A), y = R(A)$
	$W(A, y), U(A)$
	$L(B), y = R(B)$
	$W(B, y), U(B)$
$L(B), x = R(B)$	
$W(B, x), U(B)$	

$y = ??$

$y = ??$

$x = ??$

"lock **all** you need"

L(A), L(B)	
R(A), W(A)	
R(B), W(B)	
U(A), U(B)	
	L(A), L(B)
	R(A), W(A)
	R(B), W(B)
	U(A), U(B)

"lock **all** you need"

achieves isolation

"proof": TXN holds all locks, so no one interferes

"lock **all** you need"

L(A), L(B)	
R(A), W(A)	
R(B), W(B)	
U(A), U(B)	
	L(B)
	R(B), W(B)
	U(B)

2 phase locking (2PL)

all locks before any unlock

allow read/write sooner/later

2 phase locking (2PL)

$L(A), x = R(A)$	
$x = f(x)$	$L(B), R(B), W(B), U(B)$
$W(A), L(B), R(B), W(B)$	
$U(A), U(B)$	

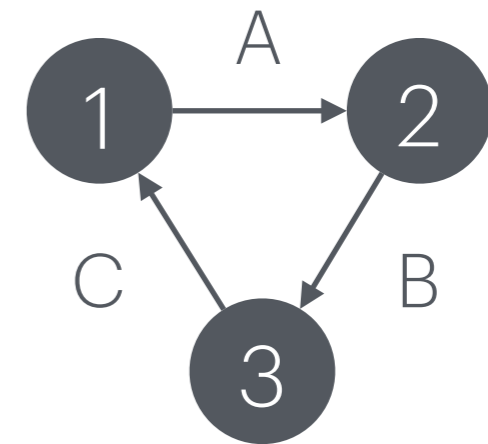
2 phase locking (2PL)

$L(A), x = R(A)$	
$x = f(x)$	$L(B), R(B), W(B), U(B)$
$W(A), L(B), U(A)$	
$R(B), W(B), U(B)$	

Theorem: 2PL \rightarrow conflict-serializable

Proof: assume cycle in precedence graph

lemma: an edge $i \xrightarrow{x} j \rightarrow U_i(X) \dots L_j(X)$



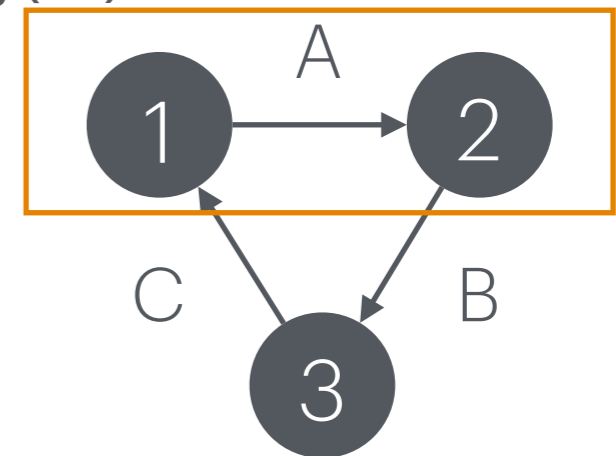
Theorem: 2PL \rightarrow conflict-serializable

Proof: assume cycle in precedence graph

lemma: an edge $i \xrightarrow{x} j \rightarrow U_i(X) \dots L_j(X)$

$W_1(A) \dots$

$\dots R_2(A) \dots$

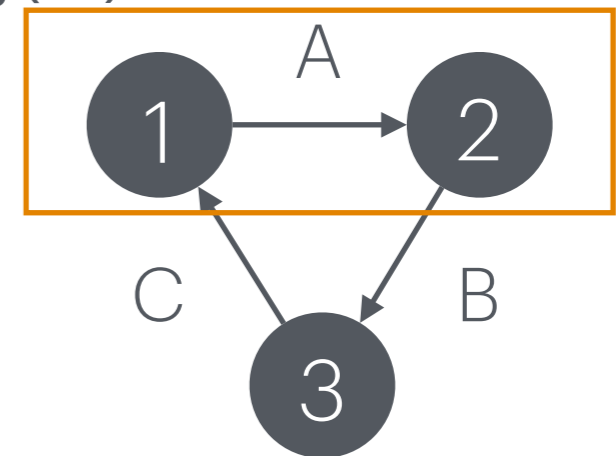


Theorem: 2PL \rightarrow conflict-serializable

Proof: assume cycle in precedence graph

lemma: an edge $i \xrightarrow{x} j \rightarrow U_i(X) \dots L_j(X)$

W1(A) ... **U1(A)** ... **L2(A)** ... R2(A)



Theorem: 2PL \rightarrow conflict-serializable

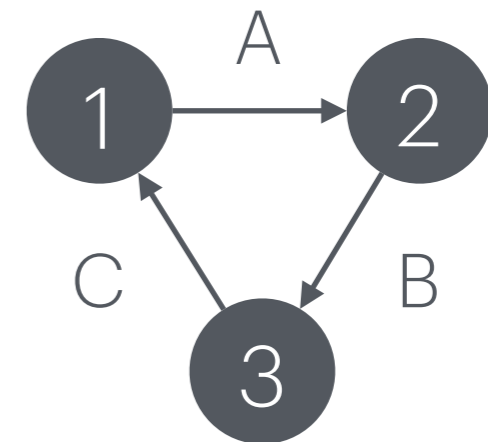
Proof: assume cycle in precedence graph

lemma: an edge $i \xrightarrow{x} j \rightarrow U_i(X) \dots L_j(X)$

$W_1(A) \dots \mathbf{U_1(A)} \dots \mathbf{L_2(A)} \dots R_2(A) \dots$

$\mathbf{U_2(B)} \dots \mathbf{L_3(B)}$

$\mathbf{U_3(C)} \dots \mathbf{L_1(C)}$



Theorem: 2PL \rightarrow conflict-serializable

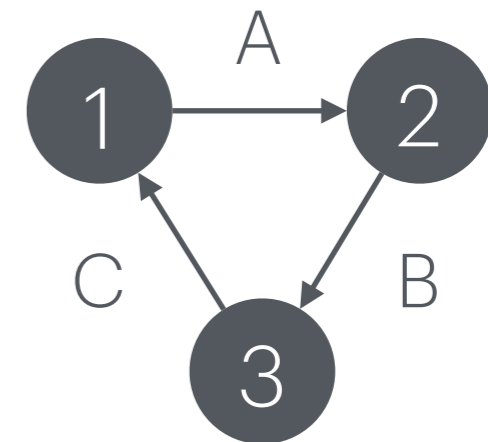
Proof: assume cycle in precedence graph

lemma: an edge $i \xrightarrow{x} j \rightarrow U_i(X) \dots L_j(X)$

W1(A) ... **U1(A)** ... **L2(A)** ... R2(A)

2PL **U2(B) ... L3(B)**

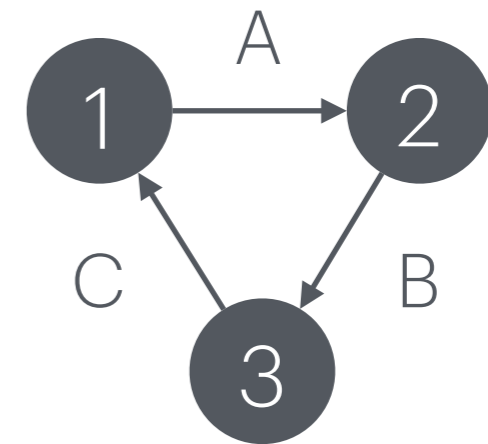
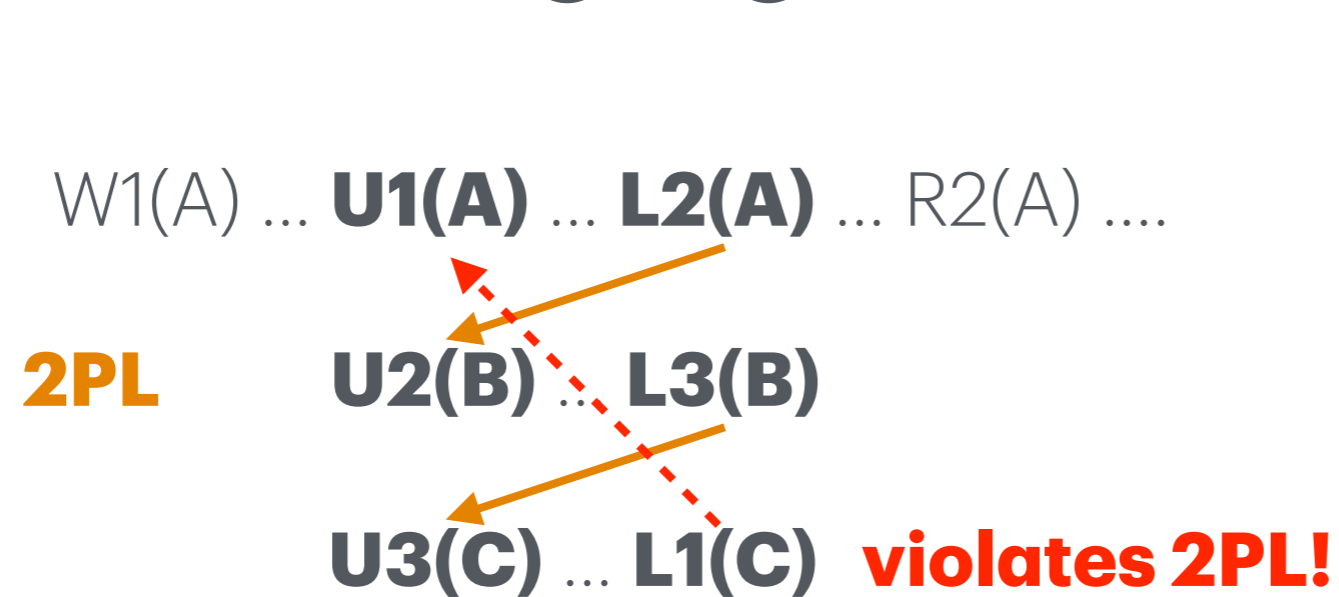
U3(C) ... L1(C)



Theorem: 2PL \rightarrow conflict-serializable

Proof: assume cycle in precedence graph

lemma: an edge $i \xrightarrow{x} j \rightarrow U_i(X) \dots L_j(X)$



rollback

undo entire TXN

abort affected TXNs

$x = 2x$

$L(A), x = R(A)$	
$W(A, x), U(A)$	
	$L(A), y = R(A)$
	$W(A, y), U(A)$
	COMMIT
ROLLBACK	

$y = 2y$

strict 2PL

unlock exactly at commit/rollback

$x = 2x$

$L(A), x = R(A)$	
$W(A, x)$	
ROLLBACK, U(A)	
	$L(A), y = R(A)$
	$W(A, y)$
	COMMIT, U(A)

$y = 2y$

strict 2PL guarantees:

isolation: conflict-serializability

atomicity: rollback TXNs are undone

😊 perfect 😊

T1: W(A),W(B)

T2: W(B),W(C)

T3: W(C),W(D)

L(A),W(A)		
	L(B),W(B)	
		L(C),W(C)
L(B) blocked!	L(C) blocked!	L(A) blocked!
DEADLOCK!		



deadlock 🦴

cannot make progress

because no TXN can acquire lock

checking for deadlock

construct wait-for graph

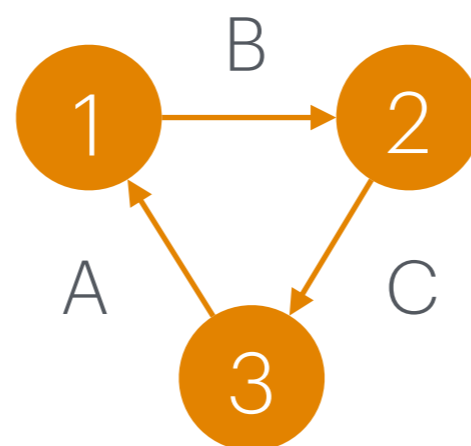
check for cycle

T1: W(A),W(B)

T2: W(B),W(C)

T3: W(C),W(D)

L(A),W(A)		
	L(B),W(B)	
		L(C),W(C)
L(B)	L(C)	L(A)



if deadlock happens

rollback TXN to break cycle

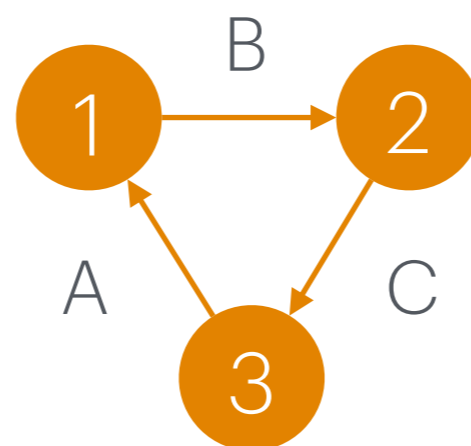
make progress

T1: W(A),W(B)

T2: W(B),W(C)

T3: W(C),W(D)

L(A),W(A)		
	L(B),W(B)	
		L(C),W(C)
L(B)	L(C)	L(A)

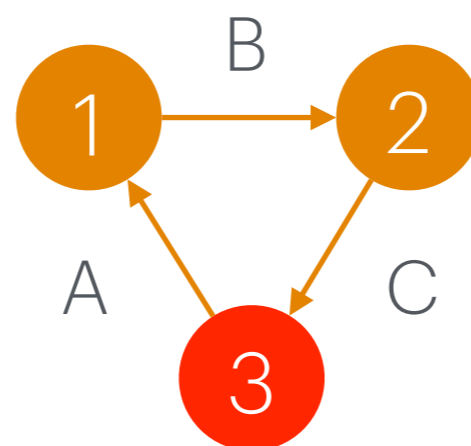


T1: W(A),W(B)

T2: W(B),W(C)

T3: W(C),W(D)

L(A),W(A)		
	L(B),W(B)	
		L(C),W(C)
L(B)	L(C)	RLBCK, U(C)

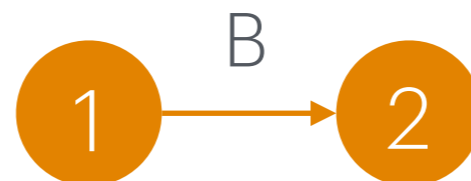


T1: W(A),W(B)

T2: W(B),W(C)

T3: W(C),W(D)

L(A),W(A)		
	L(B),W(B)	
		L(C),W(C)
L(B)	L(C)	RLBCK, U(C)

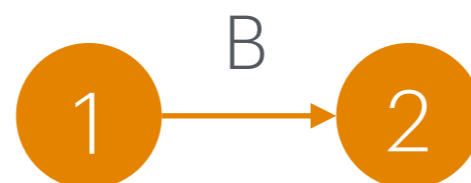


T1: W(A),W(B)

T2: W(B),W(C)

T3: W(C),W(D)

L(A),W(A)		
	L(B),W(B)	
		L(C),W(C)
		RLBCK, U(C)
	L(C),W(C),U(*)	
L(B),W(B),U(*)		



TXN sequence of reads & writes

isolation 2 TXN must not interfere

atomicity TXN either completes or rolled back

serial sch. one at a time

serializable sch. equiv. to some serial sch.

conflict-serializable sch. conflict-equivalent to serial

strict 2PL: conflict-serializable & recoverable (atomicity)

deadlock can still occur, so abort & retry if so

TXN sequence of reads & writes

isolation 2 TXN must not interfere

atomicity TXN either completes or rolled back

serial sch. one at a time

serializable sch. equiv. to some serial sch.

conflict-serializable sch. conflict-equivalent to serial

strict 2PL: conflict-serializable & recoverable (atomicity)

still expensive!

deadlock can still occur, so abort & retry if so

shared/exclusive locks

X(A) exclusive locks

allows R/W, no other locks may exist

S(A) shared locks


allows R only, may exist with other **S** locks

T1: R(A), W(A)

T2: R(A)

S(A), R(A)	
	S(A), R(A)
X(A), W(A)	
	COMMIT, U(A)
X(A), W(A)	

T1 T2 T3 T4 T5 ...
W(A) **R(A)** **R(A)** **R(A)** **R(A)** ...

X(A)	S(A)				
X(A)		S(A)			
X(A)			S(A)		
X(A)				S(A)	
					...

starvation

TXN waits for lock but never gets it

1 type of lock: queue TXNs

S/X lock: block **S** locks when **X** is waiting

starvation != deadlock

S/X locks also cause more deadlocks

T1: R(A), W(A)

T2: R(A), W(A)

S(A), R(A)	
	S(A), R(A)
X(A), W(A)	
	X(A), W(A)

isolation levels

FAST

CORRECT



None (Chaos?)

Read Uncommitted

Read Committed

Repeatable Read

Serializable

isolation levels

avoid certain types of problems:

dirty read/inconsistent read

lost update

unrepeatable read

dirty/inconsistent read

seeing updates from uncommitted TXN

*Manager wants to
balance project budgets*

*CEO wants to check
company balance*

SELECT SUM(budget) ...

-\$10mil from project A



+\$7mil to project B



+\$3mil to project C



time



lost update

update overwritten by another TXN

$$A = B = 100$$

$W(A, 200)$	
$W(B, 0)$	
	$W(B, 200)$
	$W(A, 0)$

$$A + B = 200 \checkmark$$

$$A = B = 100$$

$W(A, 200)$	
	$W(B, 200)$
$W(B, 0)$	
	$W(A, 0)$

$$A = B = 0 \text{ X}$$

unrepeatable read

two reads give different results

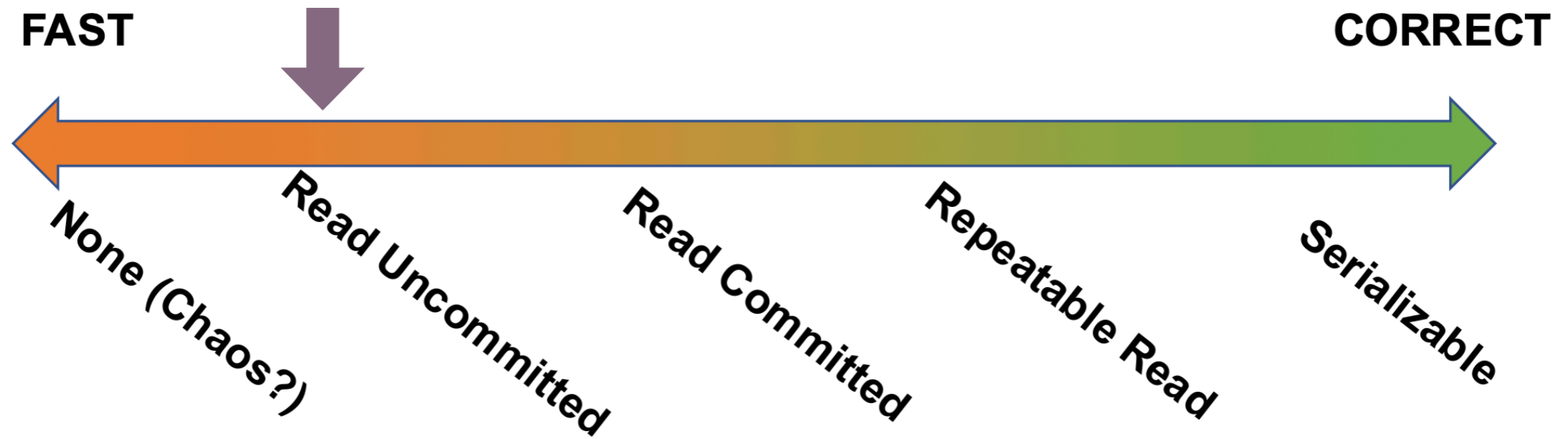
$A = 100$

	$R(A)$
$W(A, 0)$	
	$R(A)$

$A = 100$

$A = 0$

isolation levels



read uncommitted

Strict 2PL for writes

no locks at all for reads!

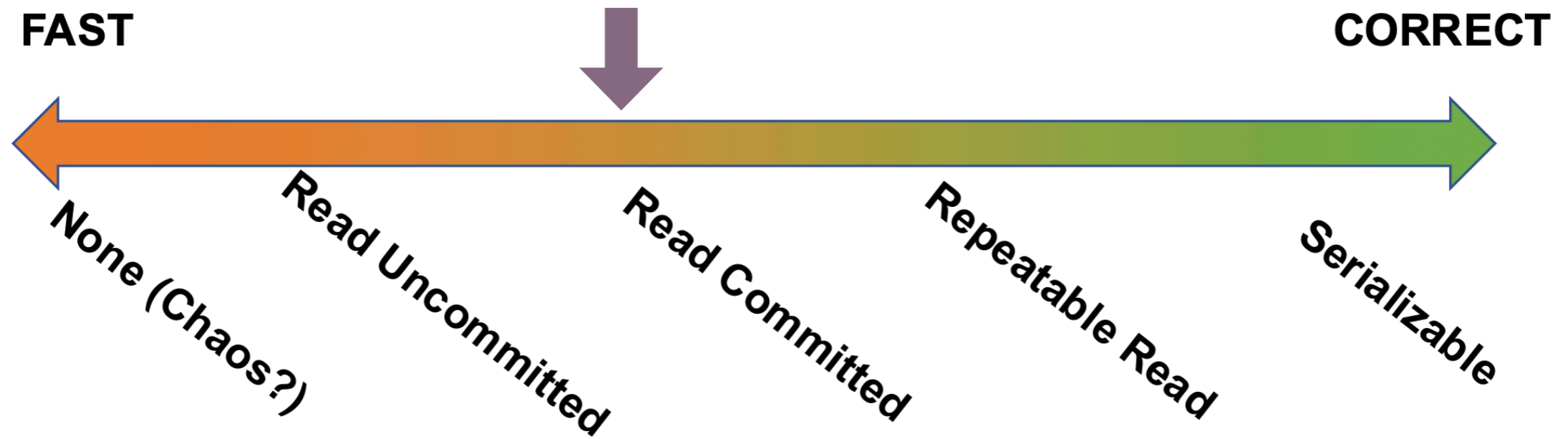
read uncommitted

very fast reads

assumes few/no writes

read accuracy is not critical

isolation levels



read committed

Strict 2PL for writes

on-demand read locks (not 2PL!)

lock \rightarrow R \rightarrow unlock

no dirty reads, possible unrepeatable reads

no dirty reads

$W(A, 0)$	
	$L(A), R(A)$
COMMIT, $U(A)$	

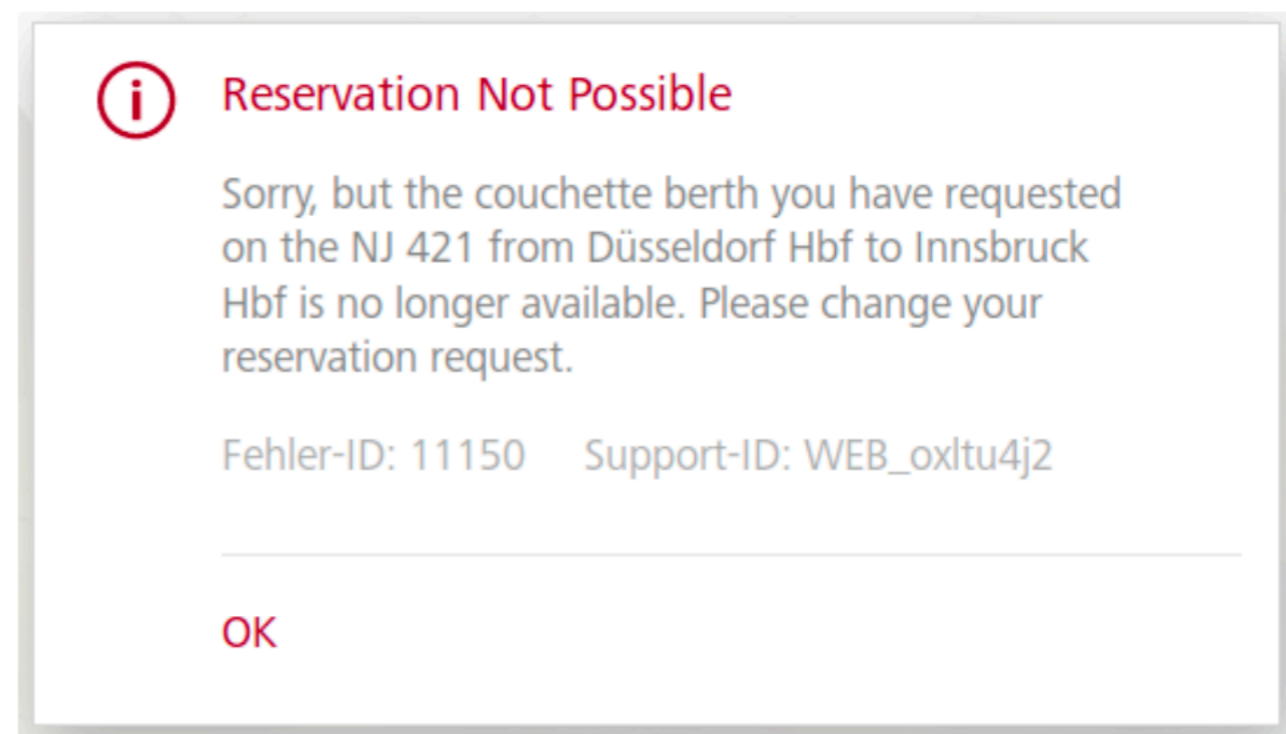
possible unrepeatable reads

	L(A),R(A),U(A)
L(A),W(A),U(A)	
	L(A),R(A),U(A)

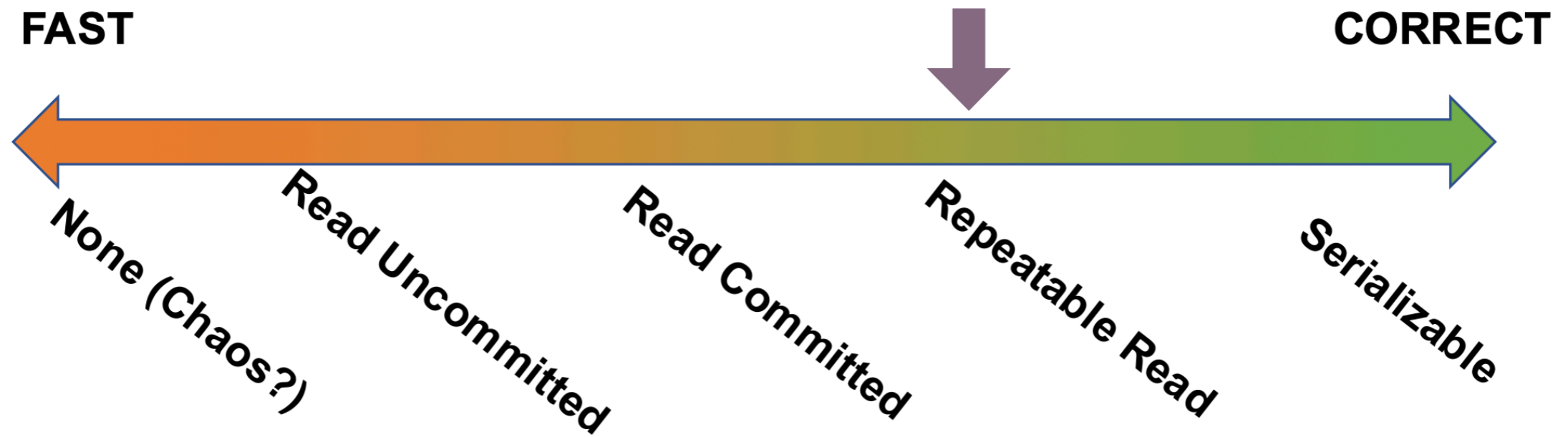
read committed

guarantee read result is valid at *some* point

useful for online shops



isolation levels



repeatable read

Strict 2PL write locks

Strict 2PL read locks

conflict serializable!

but not serializable???

The Phantom Menace

- Same read has more rows
- Asset checking scenario:

Accountant wants to check company assets

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

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

Warehouse catalogs new products

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

time



possible unrepeatable reads

SELECT *	R(A), R(B)	
		W(C)
SELECT *	R(A), R(B), R(C)	

INSERT

the phantom problem

conflict serializable → serializable **w/o inserts**

solution: lock entire table

possible unrepeatable reads

SELECT *	L(T) , R(T)	
		L(T), W(C)
SELECT *	R(T), C, U(T)	