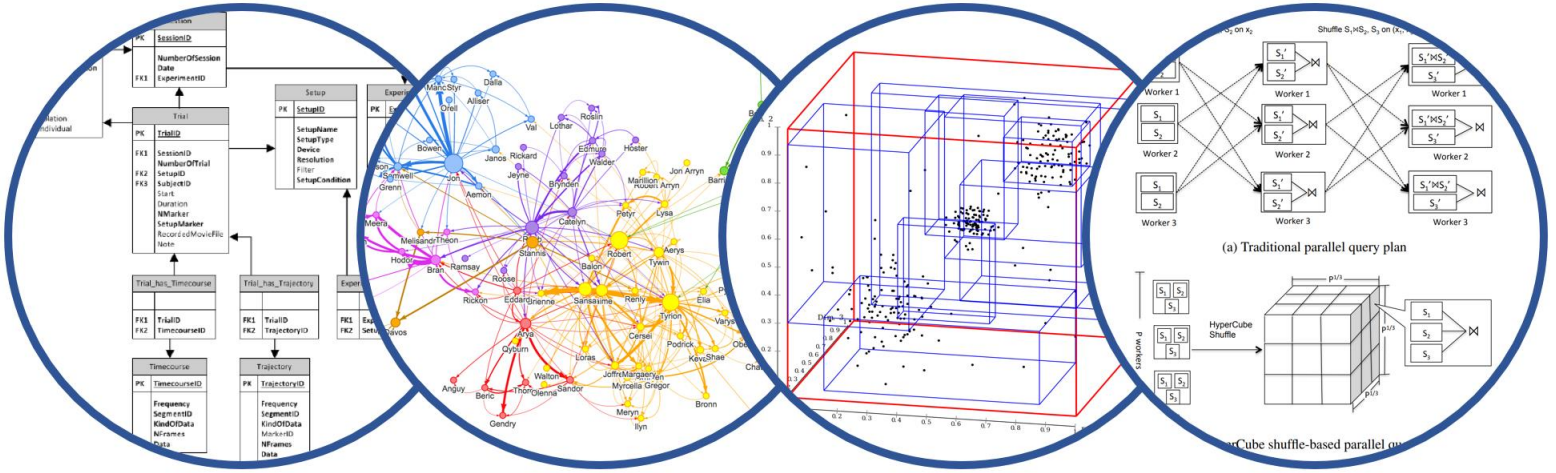


Introduction to Data Management

Transactions: Isolation Levels

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

Announcements

- Wednesday: guest lecture by Amal Jacob
- HW5 is due on Friday

Agenda

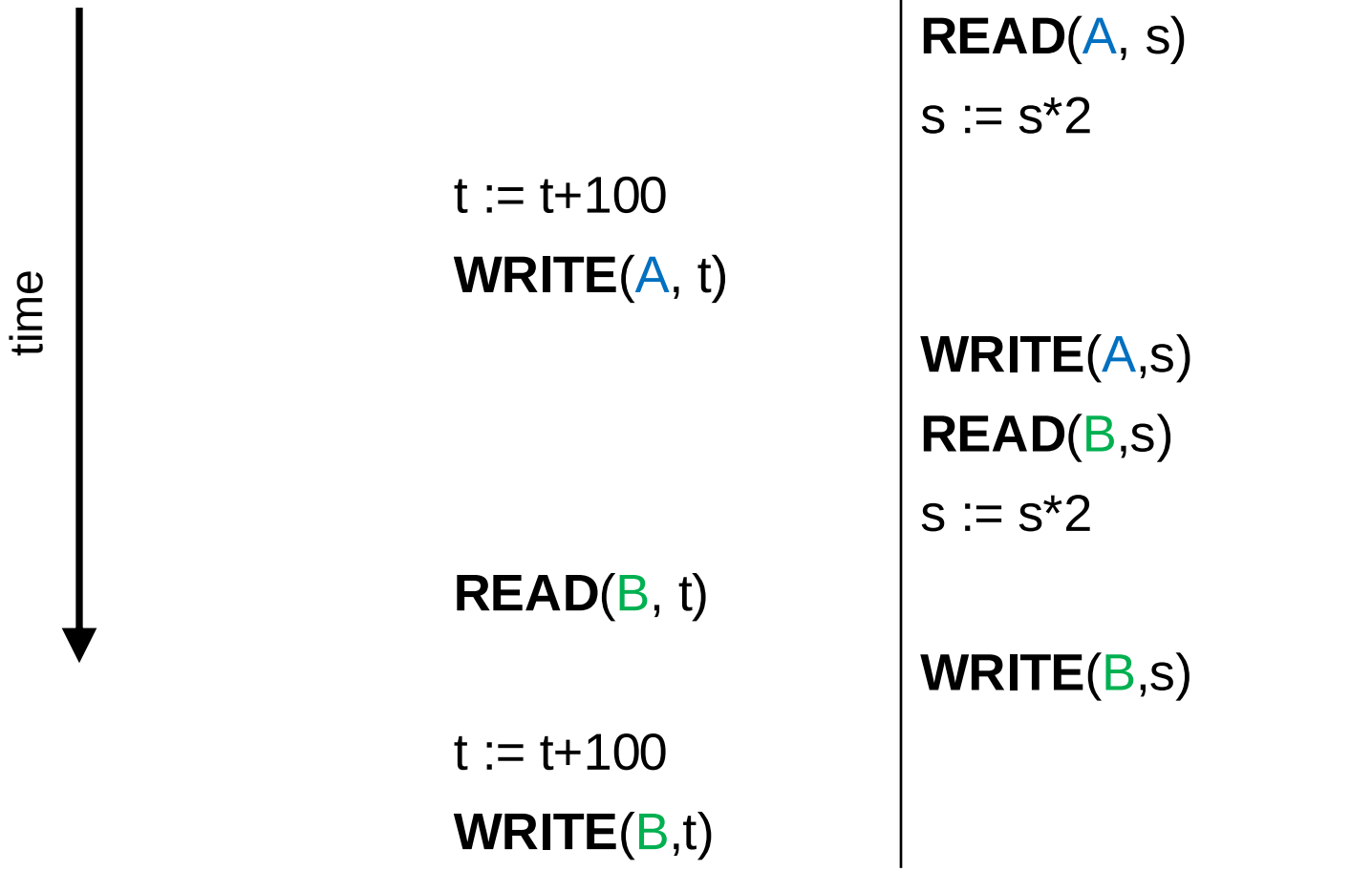
- Recap what we have seen so far
- Types of locks
- Conflicts between concurrent operations
- Weaker isolation level

We will not finish today! We will continue on Friday

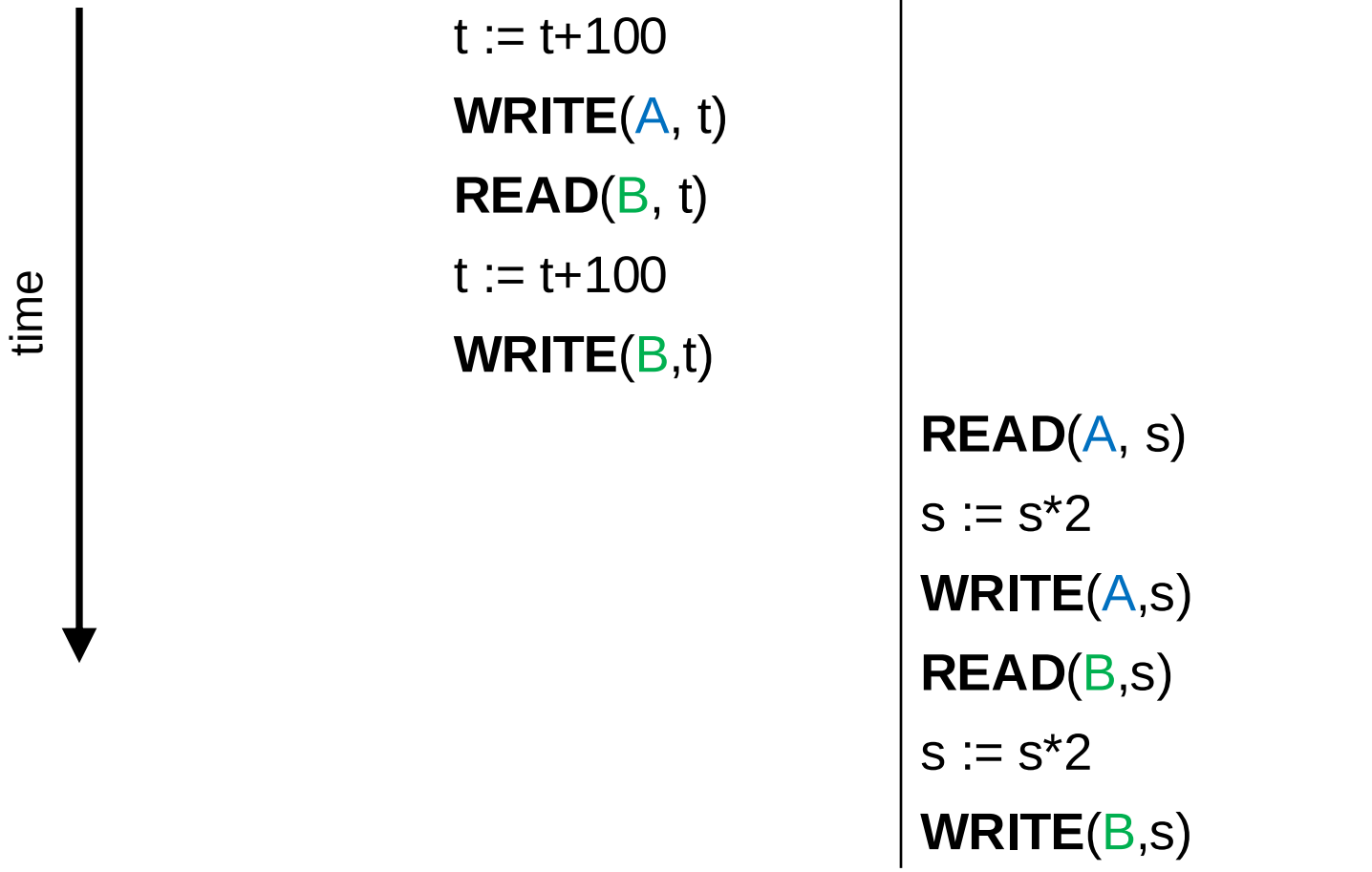
Recap

- TXN = sequence of Reads and Writes of elements
  - BEGIN TRANSACTION
  - COMMIT or ROLLBACK
- Schedule = interleaving of operations of TXNs
- Serial Schedule = one TXN after the other

A Schedule


time
T1
T2
READ(A, t)
READ(A, s)
s := s*2
t := t+100
WRITE(A, t)
WRITE(A,s)
READ(B,s)
s := s*2
READ(B, t)
WRITE(B,s)
t := t+100
WRITE(B,t)

A Serial Schedule



Recap

- Serializable Schedule = equivalent to a serial one
- Conflict Serializable Schedule = …

Serializable and Conflict-Serializable

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

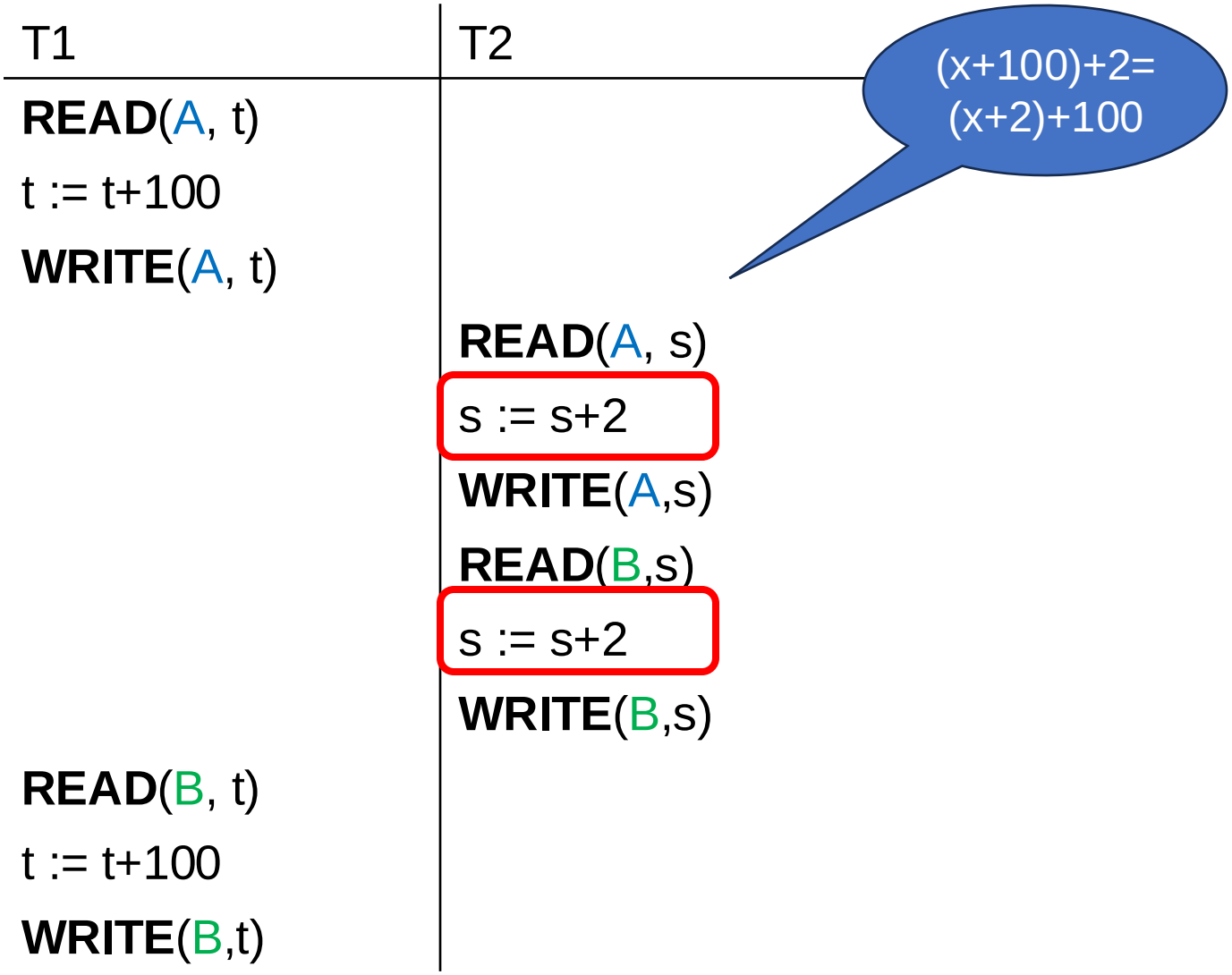
Non-Serializable, Non-Conflict-Serializable

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

Serializable, Non-Conflict-Serializable

T1	T2
READ(A, t)	
t := t+100	
WRITE(A, t)	
	READ(A, s)
	s := s+2
	WRITE(A,s)
	READ(B,s)
	s := s+2
	WRITE(B,s)
READ(B, t)	
t := t+100	
WRITE(B,t)	

Serializable, Non-Conflict-Serializable



Non-Serializable, Non-Conflict-Serializable

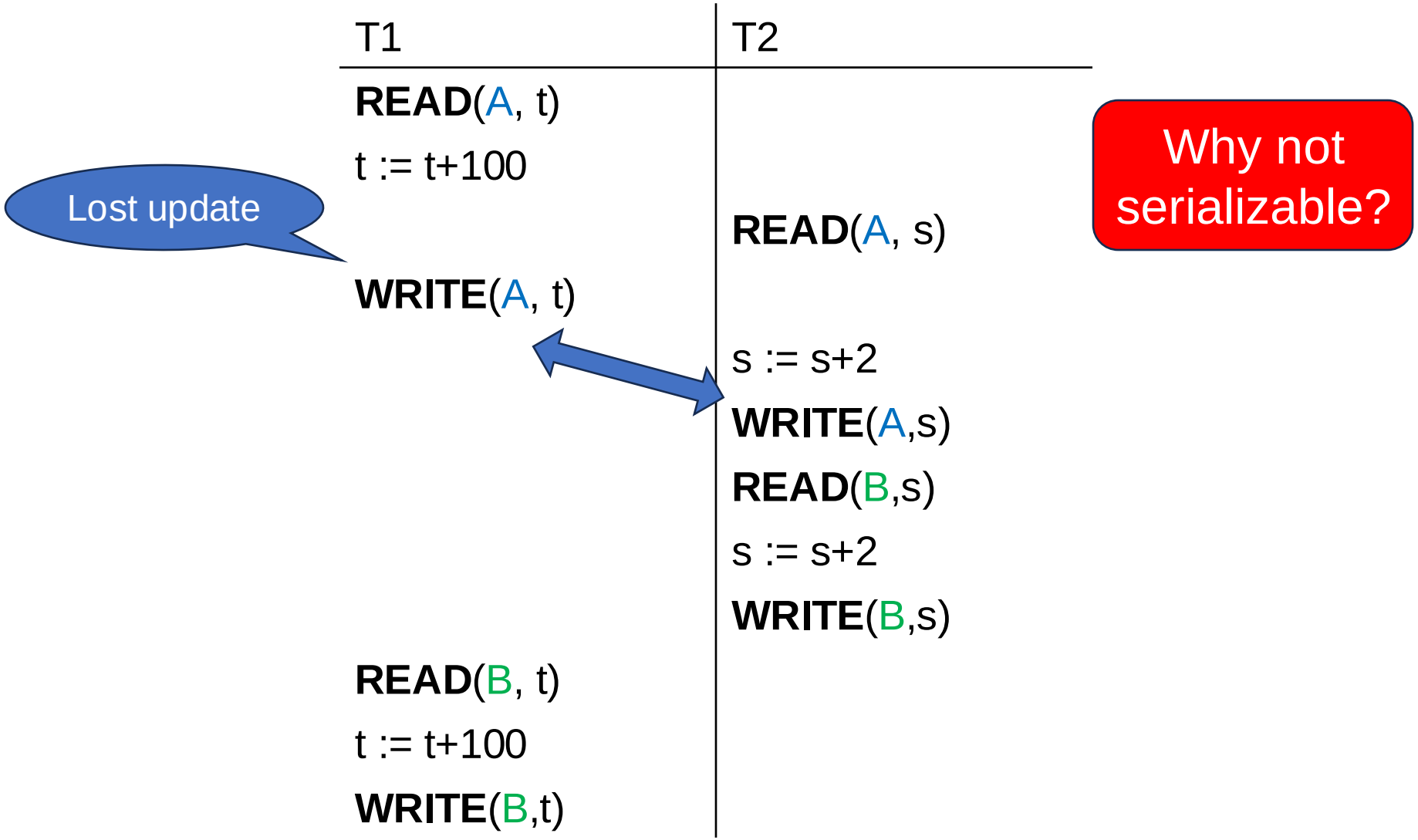
T1	T2
READ(A, t)	
t := t+100	
	READ(A, s)
WRITE(A, t)	
	s := s+2
	WRITE(A,s)
	READ(B,s)
	s := s+2
	WRITE(B,s)
READ(B, t)	
t := t+100	
WRITE(B,t)	

Non-Serializable, Non-Conflict-Serializable

T1	T2
READ(A, t)	
t := t+100	
	READ(A, s)
WRITE(A, t)	
	s := s+2
	WRITE(A,s)
	READ(B,s)
	s := s+2
	WRITE(B,s)
READ(B, t)	
t := t+100	
WRITE(B,t)	

Why not serializable?

Non-Serializable, Non-Conflict-Serializable



Discussion

- Every conflict-serializable schedule is serializable
- The converse is false
- Checking conflict-serializability: precedence graph
- Checking serializability:
  need to understand what TXNs are doing

Recap: Concurrency Control Manager

- Scheduler a.k.a. Concurrency Control Manager
- Pessimistic (Locks) or Optimistic (various…)

Locks:

- $L_i(A)$ = transaction T_i acquires lock for element A
- $U_i(A)$ = transaction T_i releases lock for element A

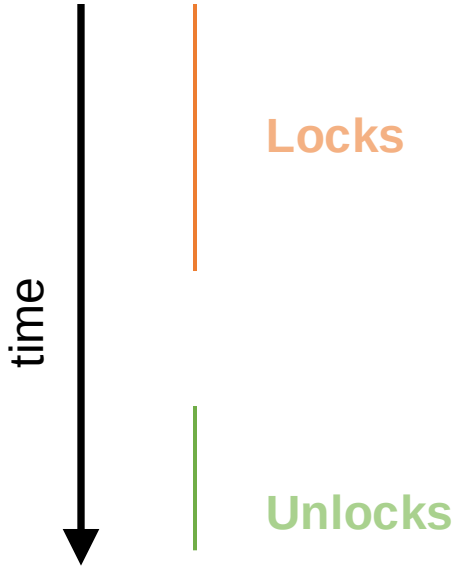
Locks Alone do not Enforce Serializability

T1	T2
L1(A), **READ**(A, t)	
t := t+100	
WRITE(A, t),**U1**(A)	
	L2(A), **READ**(A, s)
	s := s*2
	WRITE(A,s),**U2**(A)
	L2(B), **READ**(B,s)
	s := s*2
	WRITE(B,s),**U2**(B)
L1(B)	
READ(B, t)	
t := t+100	
WRITE(B,t),**U1**(B)	

We used locks, but
this is a non-serializable schedule

Recap: Two-Phase Locking

In every TXN, all locks must come before any unlock



Theorem: If all TXNs follow 2PL, then schedule is conflict-serializable

Recap: Non-recoverable Schedule

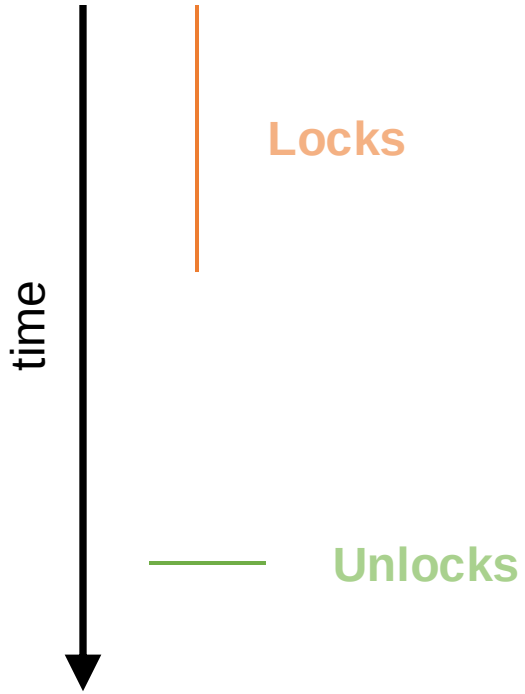
T1	T2
L1(A),**L1**(B),**READ**(A, t)	
t := t+100	
WRITE(A, t),	
READ(B, t)	
t := t+100	
WRITE(B,t), **U1**(A),**U1**(B)	
.	**L2**(A), **READ**(A, s)
.	s := s*2
.	**WRITE**(A,s),**U2**(A)
.	**L2**(B), **READ**(B,s)
.	s := s*2
.	**WRITE**(B,s),**U2**(B)
.	**COMMIT**
ROLLBACK	

Recap: Non-recoverable Schedule

T1	T2
L1(A),**L1**(B),**READ**(A, t)	
t := t+100	
WRITE(A, t),	
READ(B, t)	
t := t+100	
WRITE(B,t), **U1**(A),**U1**(B)	
.	**L2**(A), **READ**(A, s)
.	s := s*2
.	**WRITE**(A,s),**U2**(A)
.	**L2**(B), **READ**(B,s)
.	s := s*2
.	**WRITE**(B,s),**U2**(B)
.	**COMMIT**
ROLLBACK	

Recap: Strict 2PL

All locks are released at Commit/Rollback time



Theorem: If all TXNs follow strict 2PL, then schedule is conflict-serializable and recovarable

Discussion

Strict 2PL ensures conflict-serializable and recoverable schedules

- When the database is static (no insert/delete) then every conflict-serializable schedule is serializable
- When database is dynamic (has inserts/deletes) then it no longer holds because of fantoms (later)

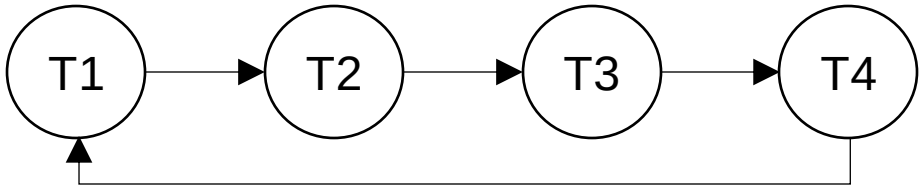
Recap: Deadlocks

T1 (A, B)	T2 (B, C)	T3 (C, D)	T4 (D, A)
L(A)	L(B)	L(C)	L(D)
L(B) blocked…			
	L(C) blocked…		
		L(D) blocked…	
			L(A) blocked…
…	…	…	…

Checking for deadlock:

- Construct the WAITS-FOR graph
- Check if it has a cycle

Checking for a cycle is fast (see CSE373), but it is very slow compared to the simple R/W operations



Recap: Deadlocks

- Deadlocks are unavoidable
  - Not quite true (in class…)
- System must check periodically for deadlocks:
  - If found, then sacrifice a TXN (abort it!)
- Application writer must catch exceptions
  - If TXN was aborted, retry a few times

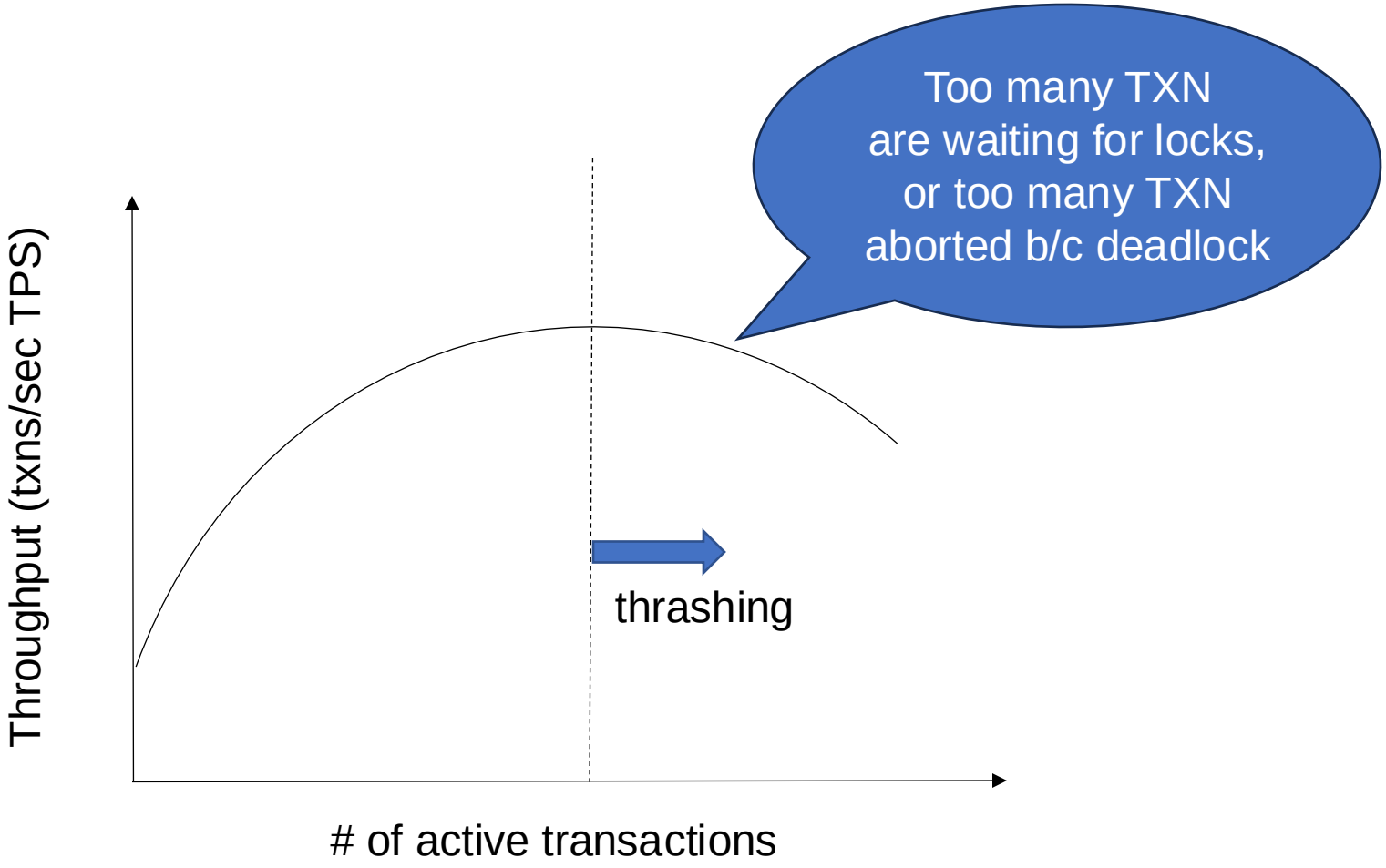
Discussion

Enforcing ACID properties slows down the RDBMs

- Concurrency (I): need to wait, need to abort
- Recovery (A): need to double write to the log

Typical throughput: 1000 – 10000 TPS

Thrashing


Too many TXN
are waiting for locks,
or too many TXN
aborted b/c deadlock
Throughput (txns/sec TPS)
thrashing
# of active transactions