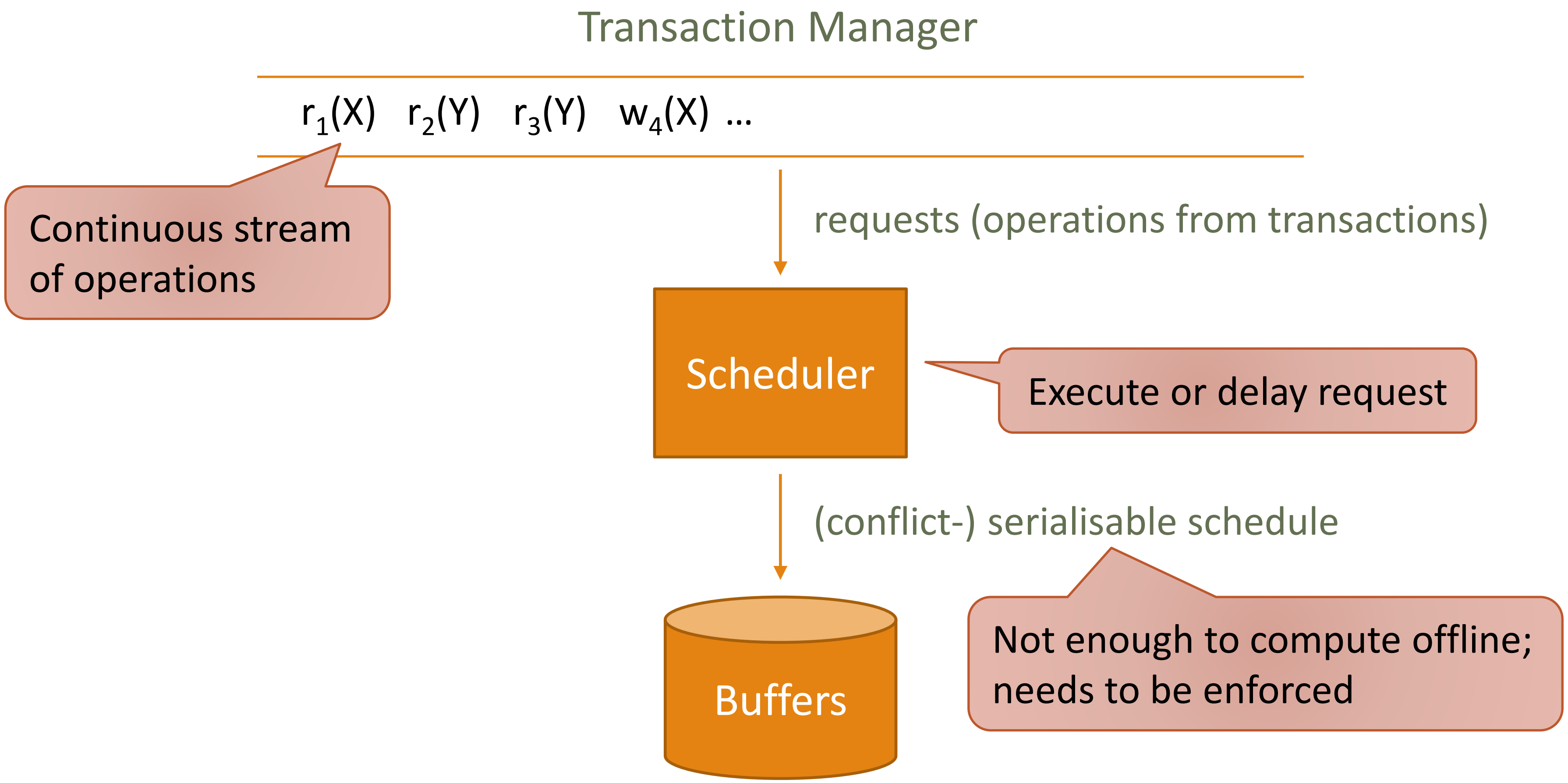
Locking conflict-serializable schedules

Overview of this video

How do we actually do conflict-serializable schedules?

Why ain’t we done?

Transaction Scheduling in a DBMS



Enforcing Conflict-Serializability Using Locks

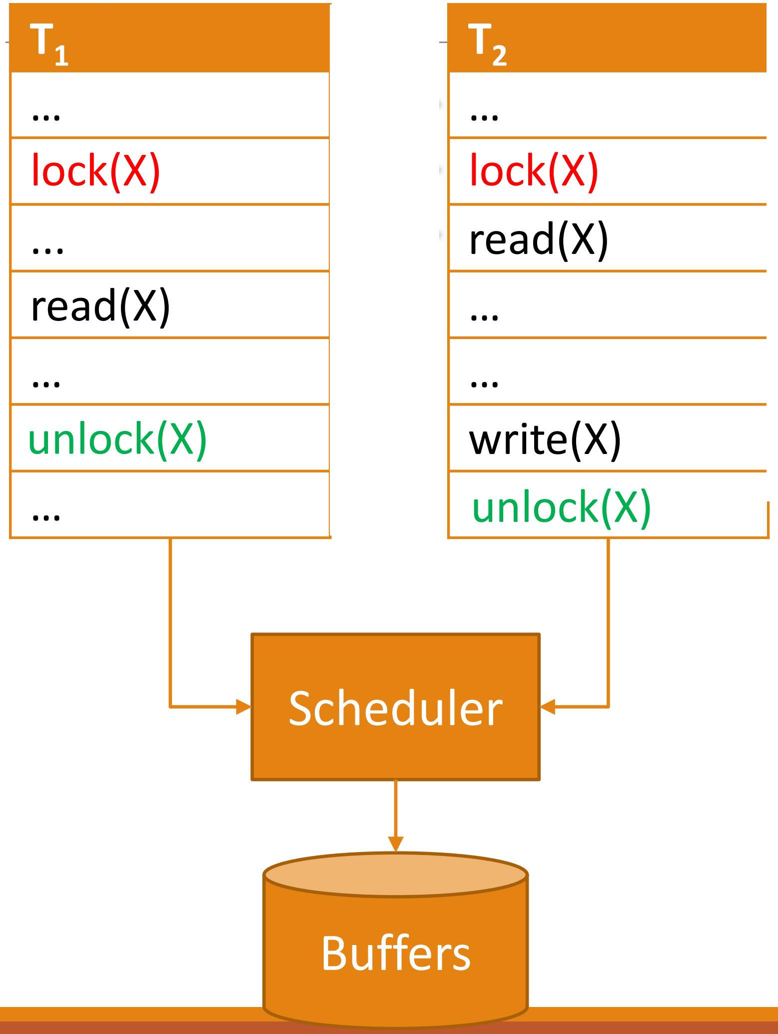
Simple Locking Mechanism

A transaction has to lock an item before it accesses it.

Locks are requested from & granted by the scheduler:

- Each item is locked by at most one transaction at a time.
- Transactions wait until a lock can be granted.

Each lock has to be released (unlocked) eventually.



Schedules With Simple Locks

Extend syntax for schedules by two operations:

- $l_i(X)$: transaction i requests a lock for item X
- $u_i(X)$: transaction i unlocks item X

Example:

$$S: \; l_1(X); r_1(X); u_1(X); l_2(X); r_2(X); w_2(X); u_2(X)$$

Rules:

- For each $r_i(X)$ / $w_i(X)$ there is an earlier $l_i(X)$ without any $u_i(X)$ occurring between $l_i(X)$ and $r_i(X)$ / $w_i(X)$.
- For each $l_i(X)$ there is a later $u_i(X)$.
- If $l_i(X)$ comes before $l_j(X)$, then $u_i(X)$ occurs between $l_i(X)$ and $l_j(X)$.

… May Not Be Serializable

T_1	T_2
lock(X)	lock(X)
read(X)	read(X)
X := X + 100	X := 2*X
write(X)	write(X)
unlock(X)	unlock(X)
lock(Y)	lock(Y)
read(Y)	read(Y)
Y := Y + 100	Y := 2*Y
write(Y)	write(Y)
unlock(Y)	unlock(Y)

not serialisable (why?)

S: $l_1(X); r_1(X); w_1(X); u_1(X); l_2(X); r_2(X); w_2(X); u_2(X);$
$l_2(Y); r_2(Y); w_2(Y); u_2(Y); l_1(Y); r_1(Y); w_1(Y); u_1(Y)$

A Serializable Schedule With Locks

T_1	T_2
lock(X)	lock(X)
read(X)	read(X)
X := X + 100	X := 2*X
write(X)	write(X)
unlock(X)	unlock(X)
lock(Y)	lock(Y)
read(Y)	read(Y)
Y := Y + 100	Y := 2*Y
write(Y)	write(Y)
unlock(Y)	unlock(Y)

T_2's request for lock on Y denied

S: $l_1(X)$; $r_1(X)$; $w_1(X)$; $l_1(Y)$; $u_1(X)$; $l_2(X)$; $r_2(X)$; $w_2(X)$; ___
$r_1(Y)$; $w_1(Y)$; $u_1(Y)$; $l_2(Y)$; $u_2(X)$; $r_2(Y)$; $w_2(Y)$; $u_2(Y)$

Two-Phase Locking (2PL)

Simple modification of the simple locking mechanism that *guarantees conflict-serializability*

Two-phase locking (2PL) condition:

In each transaction, all lock operations precede all unlocks.

"2PL transaction"

Transaction

Phase 1: request locks

+ possibly other read/write operations

Phase 2: unlock

+ possibly other read/write operations

Example 1

T_1	T_2
lock(X)	lock(X)
read(X)	read(X)
X := X + 100	X := 2*X
write(X)	write(X)
unlock(X)	unlock(X)
lock(Y)	lock(Y)
read(Y)	read(Y)
Y := Y + 100	Y := 2*Y
write(Y)	write(Y)
unlock(Y)	unlock(Y)

2PL?

S: $l_1(X); r_1(X); w_1(X); u_1(X); l_2(X); r_2(X); w_2(X); u_2(X);$
$l_2(Y); r_2(Y); w_2(Y); u_2(Y); l_1(Y); r_1(Y); w_1(Y); u_1(Y)$

Example 2

T_1	T_2
lock(X)	lock(X)
read(X)	read(X)
X := X + 100	X := 2*X
write(X)	write(X)
lock(Y)	lock(Y)
unlock(X)	unlock(X)
read(Y)	read(Y)
Y := Y + 100	Y := 2*Y
write(Y)	write(Y)
unlock(Y)	unlock(Y)

2PL?

S: $l_1(X);\ r_1(X);\ w_1(X);\ l_1(Y);\ u_1(X);\ l_2(X);\ r_2(X);\ w_2(X);$
$r_1(Y);\ w_1(Y);\ u_1(Y);\ l_2(Y);\ u_2(X);\ r_2(Y);\ w_2(Y);\ u_2(Y)$

How to test if a transaction is 2PL

Find first unlock

- Phase 1 is up to, but not including, that unlock
- Phase 2 is from that unlock until the end

Transaction is 2PL iff phase 2 contains ***no*** lock

(A schedule is 2PL iff all its transactions are 2PL)

Example 1

T_1
lock(X)
read(X)
X := X + 100
write(X)
unlock(X)
lock(Y)
read(Y)
Y := Y + 100
write(Y)
unlock(Y)

Phase 1

Phase 2

T_2
lock(X)
read(X)
X := 2*X
write(X)
unlock(X)
lock(Y)
read(Y)
Y := 2*Y
write(Y)
unlock(Y)

Phase 1

Phase 2

2PL? No!

S: $l_1(X); r_1(X); w_1(X); u_1(X); l_2(X); r_2(X); w_2(X); u_2(X);$
$l_2(Y); r_2(Y); w_2(Y); u_2(Y); l_1(Y); r_1(Y); w_1(Y); u_1(Y)$

Example 2

Is 2PL

T_1
lock(X)
read(X)
X := X + 100
write(X)
lock(Y)
unlock(X)
read(Y)
Y := Y + 100
write(Y)
unlock(Y)

Phase 1: lock(X) … lock(Y)

Phase 2: unlock(X) … unlock(Y)

Is 2PL

T_2
lock(X)
read(X)
X := 2*X
write(X)
lock(Y)
unlock(X)
read(Y)
Y := 2*Y
write(Y)
unlock(Y)

Phase 1: lock(X) … lock(Y)

Phase 2: unlock(X) … unlock(Y)

S: $l_1(X);\ r_1(X);\ w_1(X);\ l_1(Y);\ u_1(X);\ l_2(X);\ r_2(X);\ w_2(X);\ r_1(Y);\ w_1(Y);\ u_1(Y);\ l_2(Y);\ u_2(X);\ r_2(Y);\ w_2(Y);\ u_2(Y)$

2PL? Yes!

Summary

A simple way to implement conflict-serializable schedules is to use locks

You must have a lock before accessing items in that case

We saw a basic variant of locks and defined 2PL (i.e. the last lock in each transaction is before the first unlock)

That ensures conflict-serializability – here: claim; next video: proof!