More flexible locks

Overview of video

Basic locks are very simple, but not very precise (as an example, even if we just want to read we need a full lock, even though conflicts does not happen on read-read)

We will also see a basic problem with locks, called **deadlocks**

- It is present with all the kind of locks we will see, but we will come back to it later

Still Some Issues

2PL ensures conflict-serializability, but might lead to

- **Deadlocks**: transactions might be forced to wait forever
- Other issues (in later video)

Risk of Deadlocks

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

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

$l_1(X); r_1(X); w_1(X);$ $l_2(Y); r_2(Y); w_2(Y);$ ___?___

T_2's request for lock on X denied

T_1's request for lock on Y denied

Risk of Deadlocks

We will see in a later video how to solve this problem.

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

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

$l_1(X); r_1(X); w_1(X);$ $l_2(Y); r_2(Y); w_2(Y);$ $\underline{\ ?\ }$

T_2’s request for lock on X denied

T_1’s request for lock on Y denied

How can we make 2PL more flexible?

(e.g., allow read-only access by multiple transactions)

Solution: **different lock modes**

Shared & Exclusive Locks

Shared lock ("read lock"):

- Requested by transactions to read an item X
- Granted to *several transactions at the same time*

Operation:
s-lock(X)

Exclusive lock ("write lock"):

- Requested by transactions to write an item X
- Granted to *at most one transaction at a time*

Operation:
x-lock(X)

Additional rules:

- Shared lock on X is granted only if no *other* transaction holds an exclusive lock on X.
- Exclusive lock on X is granted only if no *other* transaction holds a lock (of any kind) on X.

Schedules With Shared/Exclusive Locks

Shorthand notation:

- **$sl_i(X)$**: transaction i requests a *shared* lock for item X
- **$xl_i(X)$**: transaction i requests an *exclusive* lock for item X
- **$u_i(X)$**: transaction i releases all locks on item X

Example:

T_1
s-lock(X)
read(X)
unlock(X)

T_2
s-lock(X)
read(X)
x-lock(X)
write(X)
unlock(X)

S: $sl_1(X)$; $r_1(X)$;
$sl_2(X)$; $r_2(X)$;
$u_1(X)$;
$xl_2(X)$; $w_2(X)$; $u_2(X)$

Note: An individual transaction may hold both a shared lock and an exclusive lock for the same item X.

Problems With ”Upgrading” Locks

A shared lock on an item X can be upgraded later to an exclusive lock on X.

Can use this to be “friendly” to other transactions.

Caveat: risk of deadlock

T_1
s-lock(X)
read(X)
x-lock(X)
write(X)
unlock(X)

T_2
s-lock(X)
read(X)
x-lock(X)
write(X)
unlock(X)

$sl_1(X); r_1(X); sl_2(X); r_2(X);$ $\underline{\ ?\ }$

Update Locks to the Rescue

Update lock:

- Requested by transactions to read (not write) an item
- May be upgraded later to an exclusive lock (shared locks can no longer be upgraded)
- Granted to *at most one transaction at a time*

Operation:
u-lock(X)
or
$ul_i(X)$

New upgrading policy:

Not symmetric

Transaction requests lock of type …

	Shared	Update	Exclusive
Shared	yes	yes	no
Update	no	no	no
Exclusive	no	no	no

Grant if the only types of locks held by *other* transactions are those with a "yes"

Example 1: Avoiding the Deadlock

No longer possible: Shared locks can no longer be upgraded. This now requires an update lock.

T_1
s-lock(X)
read(X)
x-lock(X)
write(X)
unlock(X)

T_2
s-lock(X)
read(X)
x-lock(X)
write(X)
unlock(X)

Example 1: Avoiding the Deadlock

T_1
u-lock(X)
read(X)
x-lock(X)
write(X)
unlock(X)

T_2
u-lock(X)
read(X)
x-lock(X)
write(X)
unlock(X)

$ul_1(X)$; $r_1(X)$; ____

T_2's request for update lock on X is denied

$xl_1(X)$; $w_1(X)$; $u_1(X)$;

$ul_2(X)$; $r_2(X)$; $xl_2(X)$; $w_2(X)$; $u_2(X)$

Example 2

T_1
s-lock(X)
read(X)
unlock(X)

T_2
u-lock(X)
read(X)
x-lock(X)
write(X)
unlock(X)

T_3
s-lock(X)
read(X)
unlock(X)

T_2 can request an update lock on X even though T_1 holds a shared lock on X

T_2's request for exclusive lock on X is denied (T_1 holds shared lock)

$sl_1(X)$; $r_1(X)$; $ul_2(X)$; $r_2(X)$;

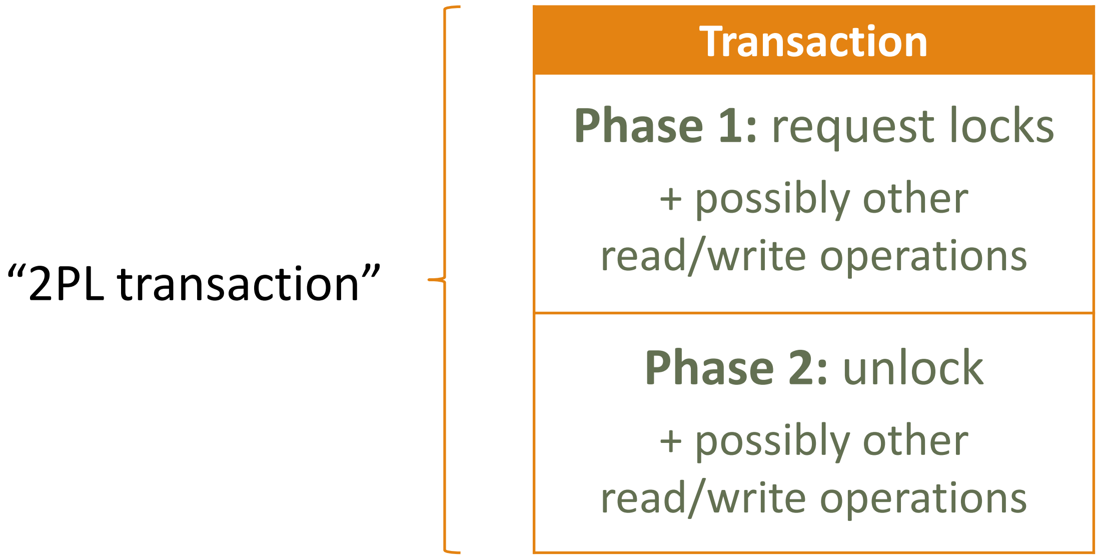
$u_1(X)$; $xl_2(X)$; $w_2(X)$; $u_2(X)$;

$sl_3(X)$; $r_3(X)$; $u_3(X)$

T_3's request for shared lock on X is denied (T_2 holds update lock)

Two-Phase Locking (2PL)
With Shared/Exclusive/Update Locks

Straightforward generalisation:
In each transaction, all lock operations (i.e., shared, exclusive, or update lock requests) precede all unlocks.



“2PL transaction”

Transaction

Phase 1: request locks

+ possibly other read/write operations

Phase 2: unlock

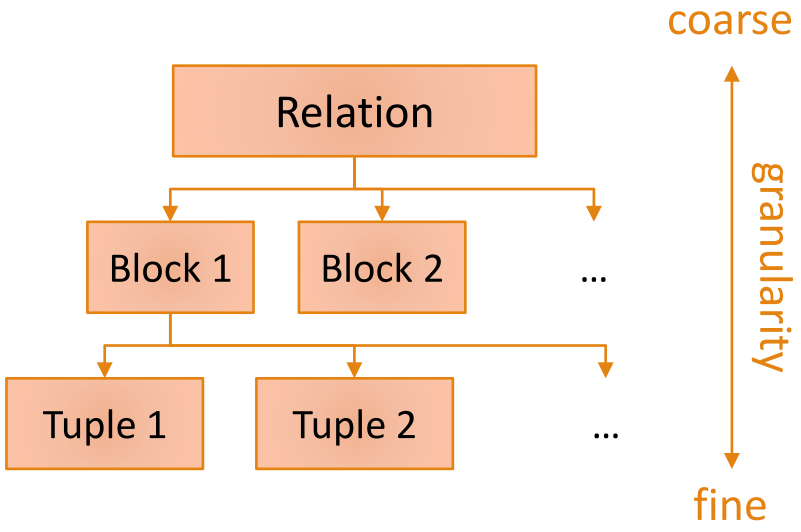
+ possibly other read/write operations

Still guarantees conflict-serializability – same argument

Locks With Multiple Granularity

DBMS may use locks at different levels of granularity

- May lock relations
- May lock disk blocks
- May lock tuples



Examples:

- SELECT name FROM Student WHERE studentID = 123456;
- SELECT avg(salary) FROM Employee;

Shared lock on tuple suffices

Shared lock on relation might be necessary

Trade-Offs

Locking at **too coarse** granularity:

- Low overhead (don’t need to store too much information)
- Less degree of concurrency: may cause unnecessary delays

Locking at **too fine** granularity:

- High overhead: need to keep track of all locked items
- High degree of concurrency: no unnecessary delays

Need to prevent issues such as the following
to guarantee (conflict-) serialisability:

- A transactions holds shared lock for a tuple.
- Another transaction holds exclusive lock for the relation.

Intention Locks

(a.k.a. Warning Locks)

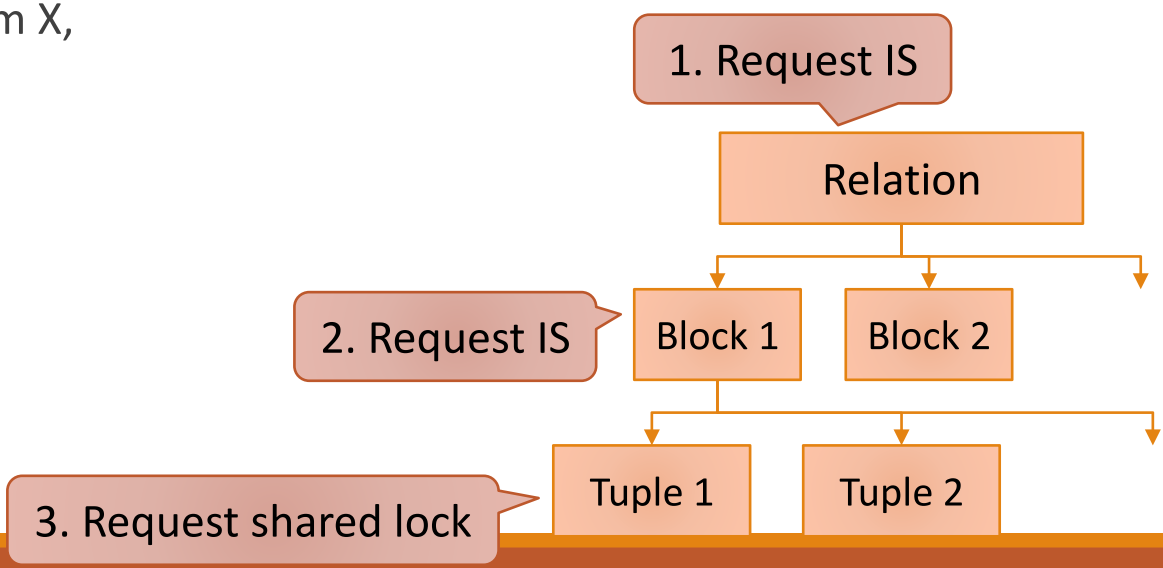
We use shared and exclusive locks (no update locks)

New **intention locks**:

- **IS**: Intention to request a shared lock on a sub-item
- **IX**: Intention to request an exclusive lock on a sub-item

Rules:

- If a transaction wants to lock an item X, it must *first* put an intention lock on the super-items of X.
- Shared locks → IS
- Exclusive locks → IX



Intention Locks
(a.k.a. Warning Locks)

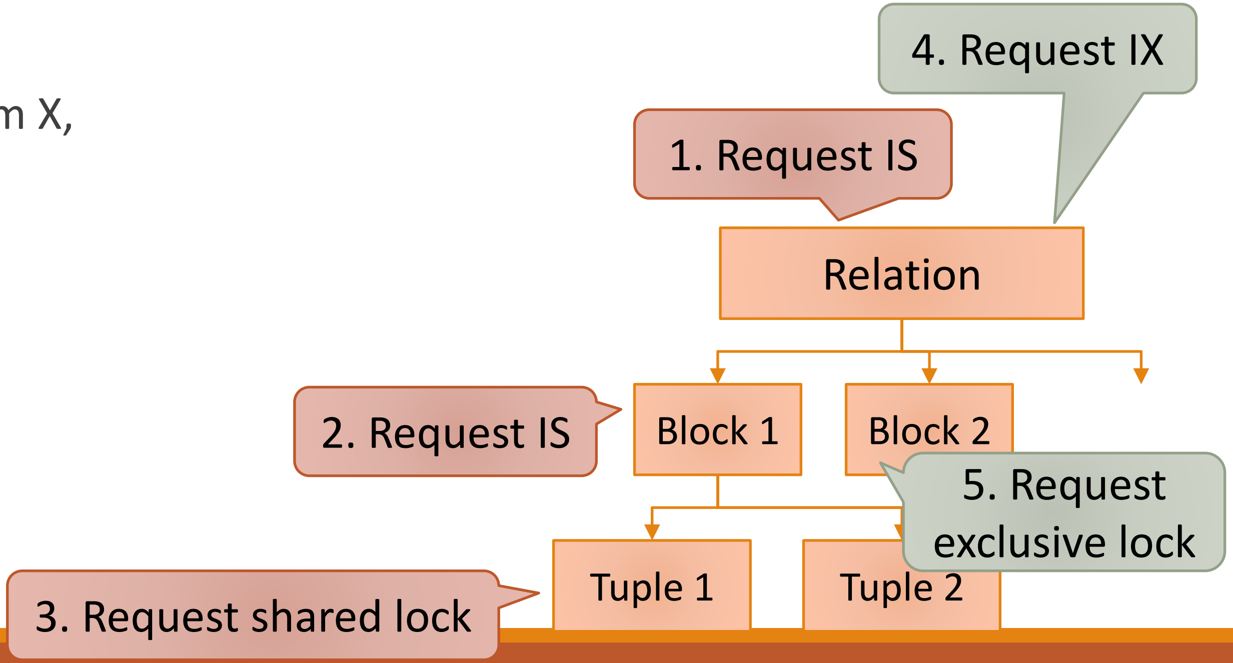
We use shared and exclusive locks (no update locks)

New **intention locks**:

- **IS**: Intention to request a shared lock on a sub-item
- **IX**: Intention to request an exclusive lock on a sub-item

Rules:

- If a transaction wants to lock an item X, it must *first* put an intention lock on the super-items of X.
- Shared locks → IS
- Exclusive locks → IX



Policy for Granting Locks

Transaction requests lock of type …

	Shared (S)	Exclusive (X)	IS	IX
Shared (S)	yes	no	yes	no
Exclusive (X)	no	no	no	no
IS	yes	no	yes	yes
IX	no	no	yes	yes

Grant if the only types of locks held by *other* transactions are those with a "yes"

Summary

Video explained deadlocks

- Where a schedule gets stuck because multiple transactions are each waiting for one of the others to release a lock

Also, shared/exclusive/update locks

- You only needed a shared or update lock to read, but an exclusive lock to write

Finally, discussed intention locks to handle different levels of precision on locks

- I.e. tuple vs. relation level of locks