Including Aborts in Serializability

- Extend the definition of a serializable schedule to include aborts
- Serializable schedule: a schedule that is equivalent to some serial execution of the set of *committed* transactions.

Conflict Serializable Schedules

- Two schedules are conflict equivalent if:
  - Involve the same actions of the same transactions
  - Every pair of conflicting actions is ordered the same way
- Schedule S is conflict serializable if S is conflict equivalent to some serial schedule

Recall Conflicts

- Two writes by Ti, Tj to same element
  - Wi(X); Wj(X)
- Read/write by Ti, Tj to same element
  - Wi(X); Rj(X)
  - Ri(X); Wj(X)

Conflict Equivalent

- Outcome of a schedule depends on the order of conflicting operations
- Can interchange non-conflicting ops without changing effect of the schedule
- If two schedules S1 and S2 are conflict equivalent then they have the same effect

  S1 ←→ S2 by swapping non-conflicting ops

Conflict Serializability

- Every conflict serializable schedule is serializable

- R1(A); W1(A); R2(A); W2(A); R1(B); W1(B); R2(B); W2(B)

Can we transform into a serial schedule by swapping of adjacent non-conflicting actions?

R1(A); W1(A); R1(B); W1(B); R2(A); W2(A); R2(B); W2(B)

Adapted from M. Balazinska

Precedence Graph Test

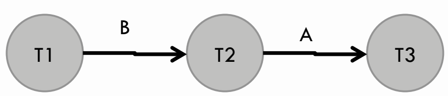
Is a schedule conflict-serializable ?

Simple test:

- Build a graph of all transactions Ti
- Edge from Ti to Tj if Ti comes first, and makes an action that conflicts with one of Tj
- The test: if the graph has no cycles, then it is conflict serializable !

Example 1

R2(A); R1(B); W2(A); R3(A); W1(B); W3(A); R2(B); W2(B)

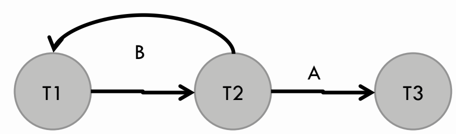


This schedule is conflict serializable

Example 2

R2(A); R1(B); W2(A); R3(A); W1(B); W3(A); R2(B); W2(B)

R2(A); R1(B); W2(A); R2(B); R3(A); W1(B); W3(A); W2(B)



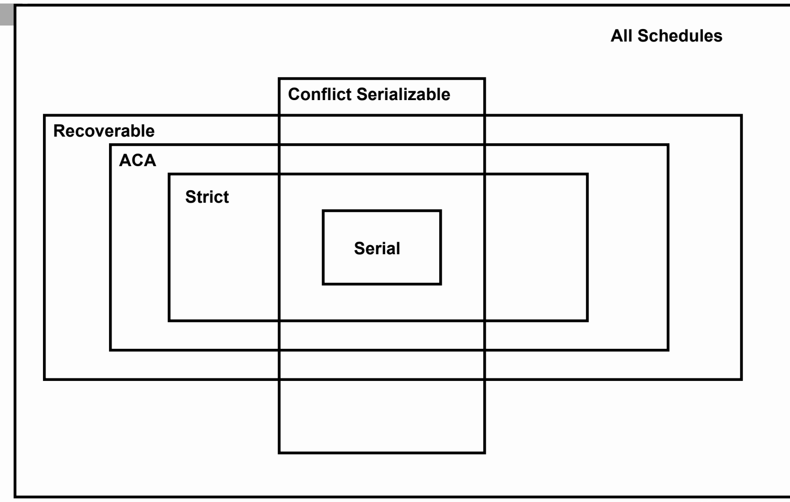
This schedule is NOT conflict serializable

The cycle indicates that the output of T1 depends on T2, and vice-versa.

Strict Schedule

- A schedule S is strict if a value written by Ti is not read or overwritten by other Tj until Ti aborts or commits
- Example:
  W1(A); W1(B), C1; W2(A); R2(B); C2;
- Strict schedules are recoverable, and avoid cascading aborts.

Venn Diagram for Schedules



Scheduler

- The scheduler is the module that schedules the transaction's actions, ensuring serializability.
- How ?
  - Locks
  - Time stamps

Lock Based Concurrency Control

- DBMS aims to allow only recoverable and serializable schedules
- Ensure committed transactions are not un-done while aborting transactions
- Use a *locking protocol*: a set of rules to be followed by each transaction to ensure serializable schedules
- *Lock*: a mechanism to control concurrent access to a data object

Locking Scheduler

Simple idea:

- Each element has a unique lock
- Each transaction must first acquire the lock before reading/writing that element
- If the lock is taken by another transaction, then wait
- The transaction must release the lock(s)

Adapted from M. Balazinska

Notation

- Li(A) = transaction Ti acquires lock for element A
- Ui(A) = transaction Ti releases lock for element A

Example 1

T1	T2
L1(A)	
R1(A), W1(A)	
U1(A), L1(B)	
	L2(A)
	R2(A), W2(A)
	U2(A)
	L2(B) , **DENIED…**
R1(B), W1(B)	
U1(B)	
	GRANTED;
	R2(B), W2(B)
	U2(B)

Scheduler has enforced a conflict serializable schedule

Example 2

T1	T2
L1(A)	
R1(A), W1(A)	
U1(A)	
	L2(A)
	R2(A), W2(A)
	U2(A)
	L2(B)
	R2(B), W2(B)
	U2(B)
L1(B)	
R1(B), W1(B)	
U1(B)	

Scheduler has NOT enforced conflict serializability

For A: T1-T2
For B: T2-T1

Types of Locks

47

- Shared lock (for reading)
- Exclusive lock (for writing, and of course, also for reading)

- Notation
  - $S_T(A)$: transaction T requests shared lock on object A
  - $X_T(A)$: transaction T requests exclusive lock on object A

Lock Modes

48

- S = Shared lock (for read)
- X = Exclusive lock (for write)

Lock compatibility matrix

	None	S	X
None	OK	OK	OK
S	OK	OK	Conflict
X	OK	Conflict	Conflict

Strict Two Phase Locking (Strict 2PL)

49

- Most widely used locking protocol
- Two rules:
  1. Each Xact must obtain a S (*shared*) lock on object before reading, and an X (*exclusive*) lock on object before writing.
  2. All locks held by a transaction are released when the transaction completes
- If an Xact holds an X lock on an object, no other Xact can get a lock (S or X) on that object.
- Strict 2PL allows only schedules whose precedence graph is acyclic (i.e., serializable)
- Recoverable and ACA

Strict 2PL Example

50

T1	T2
L(A);	
R(A), W(A)	
	L(A); **DENIED…**
L(B);	
R(B), W(B)	
U(A), U(B)	
Commit;	
	…GRANTED
	R(A), W(A)
	L(B);
	R(B), W(B)
	U(A), U(B)
	Commit;

All locks held by T1 are released when T1 completes.

Implications

51

- The locking protocol only allows safe interleavings of transactions
- If T1 and T2 access different data objects, then no conflict and each may proceed
- Otherwise, if same object, actions are ordered serially.
  - The Xact who gets the lock first must complete before the other can proceed

Two Phase Locking Protocol (2PL)

52

- Variant of Strict 2PL
- Relaxes the 2nd rule of Strict 2PL to allow Xacts to release locks before the end (commit/abort)
- Two rules:
  - Each Xact must obtain a S (*shared*) lock on object before reading, and an X (*exclusive*) lock on object before writing.
  - A transaction cannot request additional locks once it releases any lock.
  - If an Xact holds an X lock on an object, no other Xact can get a lock (S or X) on that object.

2PL Example

53

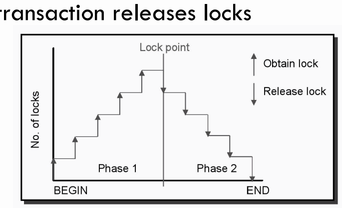
T1	T2
X(A), X(B)	
R(A), W(A)	
U(A)	
	X(A)
	R(A), W(A)
	X(B), **DENIED…**
R(B), W(B)	
U(B)	
	..GRANTED
	R(B), W(B)
	U(A), U(B)

All locks are first acquired, and then released.

2PL Implications

54

- In every transaction, all lock requests must precede all unlock requests.
- This ensures conflict serializability
  - Why? (Think of order Xacts enter their shrinking phase)
  - This induces a sort ordering of the transactions that can be serialized
- Each transaction is executed in two phases:
  - Growing phase: the transaction obtains locks
  - Shrinking phase: the transaction releases locks



Credit: A. Mazeika

Strict 2PL makes 2PL "strict"

- Recall: a strict schedule is one where a value written by T is not read/overwritten until T commits/aborts

  - Strict 2PL makes T hold locks until commit/abort
  - No other transaction can see or modify the data object until T is complete

Remarks

- What if a transaction releases its locks and then aborts?
- Recall: conflict serializable definition only considers *committed* transactions