Discussion 8

Transactions and Concurrency & Quiz 5

Concurrent Execution

- Advantages:
  - **Throughput** (transactions per second)
  - **Latency** (response time per transaction)

- Problems:
  - **Inconsistent Reads** (Write-Read Conflict)
  - **Lost Update** (Write-Write Conflict)
  - **Dirty Reads** (Write-Read Conflict)
  - **Unrepeatable Reads** (Read-Write Conflict)

Interleave transactions

- A transaction is a sequence of one or more operations (reads or writes)
- should be executed as a single, logical, atomic unit
- Property
  - **Atomicity**: A transaction ends in two ways: it either **commits** or **aborts**: either all actions in the transaction happen, or none happen.
  - **Consistency**: If the DB starts out consistent, it ends up consistent at the end of the transaction.
  - **Isolation**: Execution of each transaction is isolated from that of others. Each transaction executes as if it ran by itself.
  - **Durabilty**: If a transaction commits, its effects persist.

Concurrency Control

Serial schedule:

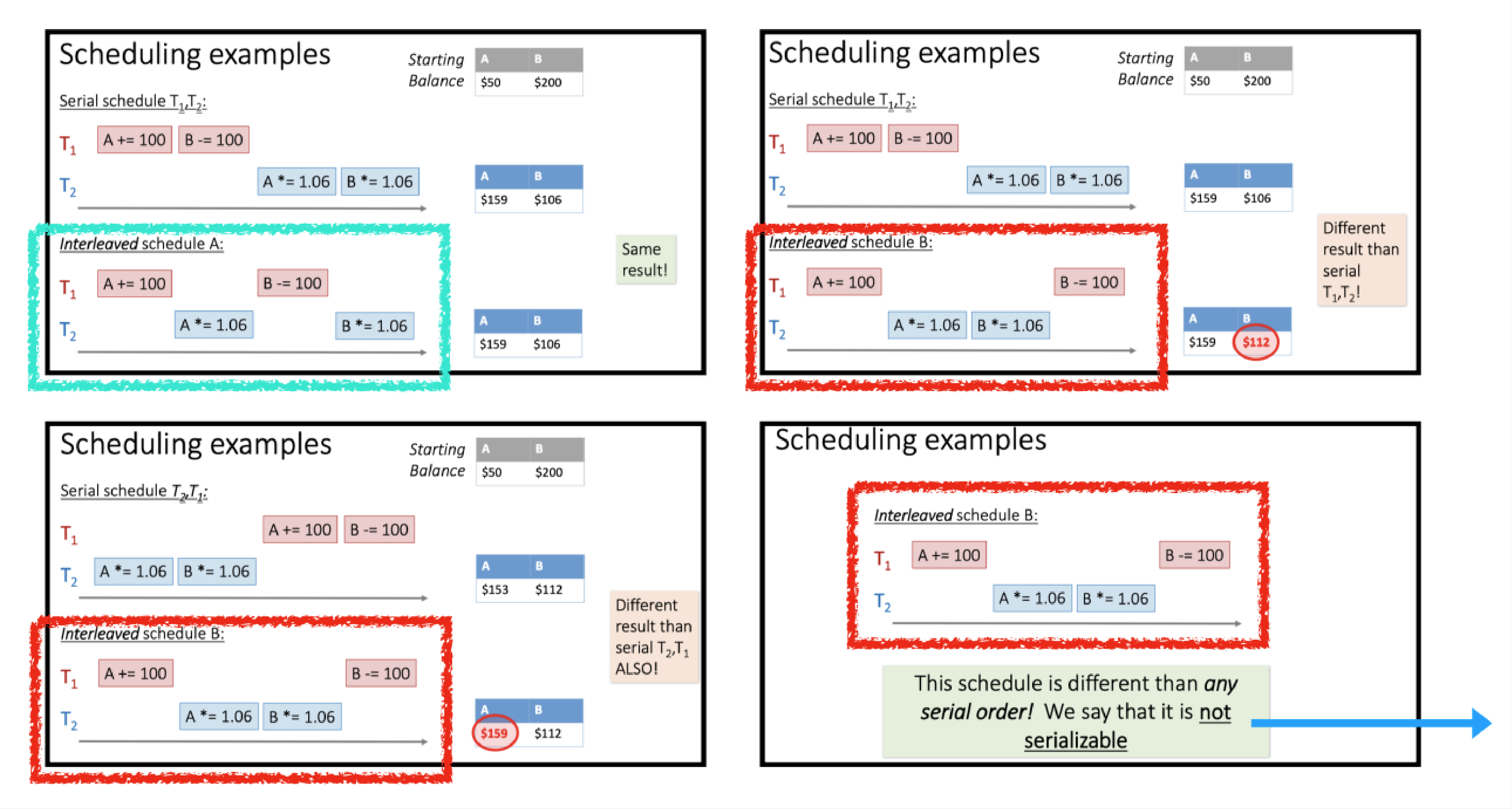
- run all the operations of one transaction to completion before beginning the operations of next transaction
  - enforce the isolation property of transactions
  - not efficient to wait for an entire transaction to finish before starting another one

Serializable Schedule: results equivalent to a serial schedule

Equivalent:

- involve the same transactions
- operations are ordered the same way within the individual transactions
- each leave the database in the same state

Example



equivalent to a serial schedule

Conflict Serializability

Conflicting operations:

- The operations are from different transactions
- Both operations operate on the same resource
- At least one operation is a write

- The order of non-conflicting operations has no effect on the final state of the database!

Conflict equivalent schedules:

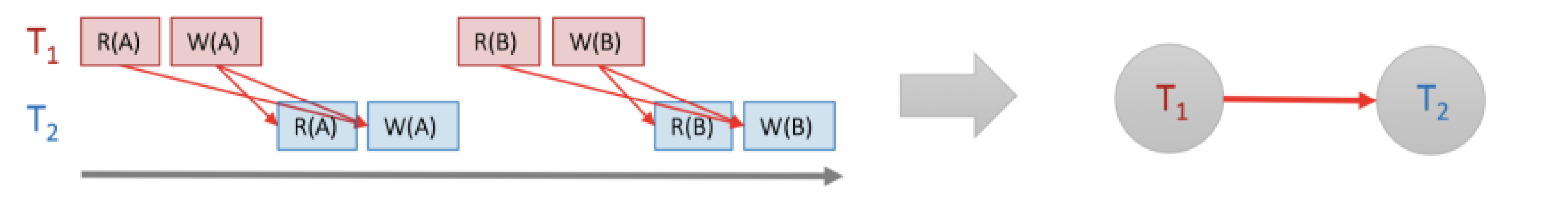
- involve the same actions of the same transactions
- every pair of conflicting actions is ordered the same way

Conflict serializable:

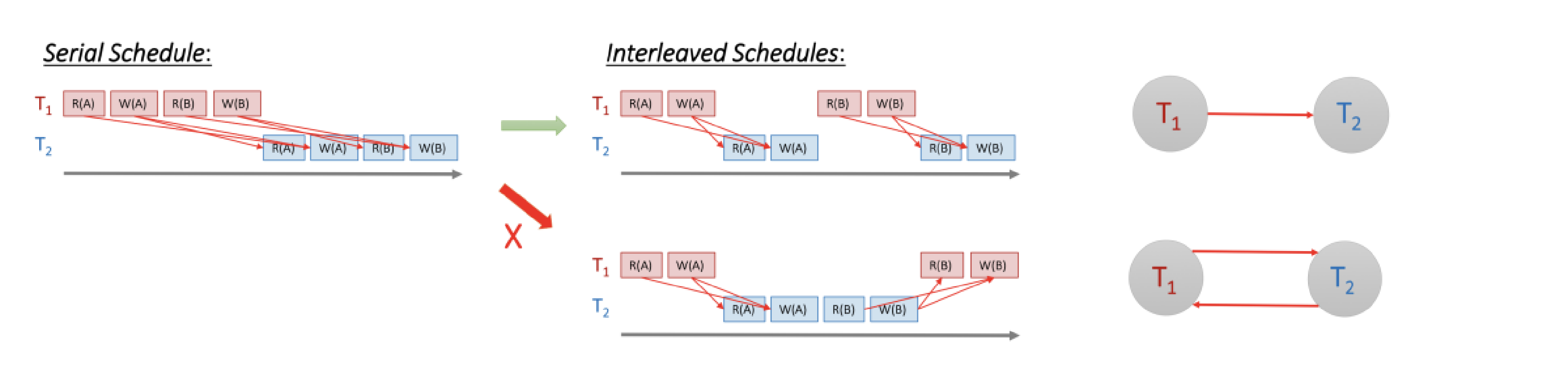
- conflict equivalent to some serial schedule
- also Serializable

Conflict Dependency Graph

- One node per transaction
- Edge from *Ti* to *Tj* if:
  - an operation *Oi* of *Ti* conflicts with an operation *Oj* of *Tj*
  - *Oi* appears earlier in the schedule than *Oj*



A schedule is conflict serializable if and only if its dependency graph is acyclic.



Two phase locking (2PL)

- Transactions must acquire a S (shared) lock before reading, and an X (exclusive) lock before writing.

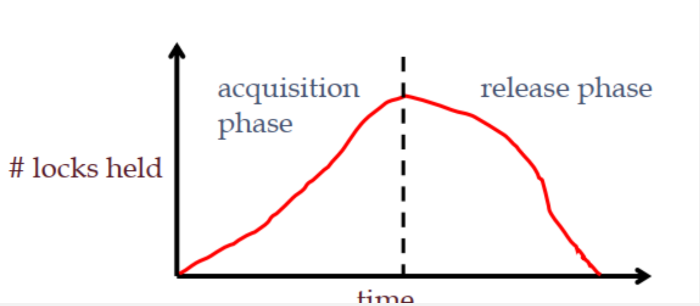
  Only one transaction may hold an exclusive lock on a resource, but many transactions can hold a shared lock on data.

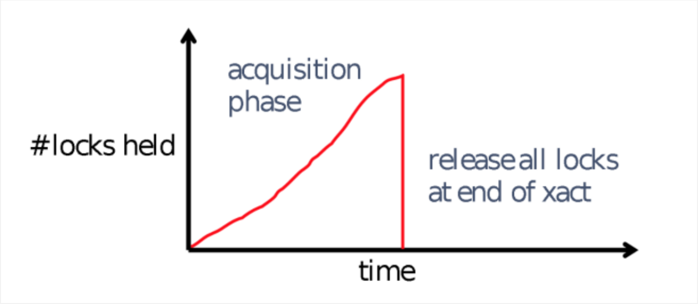
- Transactions cannot acquire new locks after releasing any locks

  –key to enforcing serializability through locking!

  does not prevent cascading aborts

- Strict Two Phase Locking
  - locks get released together when the transaction completes





Lock Management

- manages lock and unlock (or acquire and release) requests
- maintains a hash table, keyed on names of the resources being locked

	Granted Set	Mode	Wait Queue
A	{T1,T2}	S	T3(X)->T4(X)
B	{T6}	X	T5(X)->T7(X)

- Granted set (set of granted locks/the transactions holding the locks for each resource)
- Lock type (S or X or other)
- Wait queue (queue of lock requests that cannot yet be satisfied because they conflict with the locks that have already been granted)

Lock Management

- When a lock request arrives:
  - Lock Manager checks if any transactions in the Granted Set or in the Wait Queue want a conflicting lock
    - Yes: put the requester into Wait Queue
    - No: the requester is granted the lock and put into the Granted Set
- Lock upgrade:
  - a transaction with shared lock can request to upgrade to exclusive
  - add this upgrade request at the front of the queue

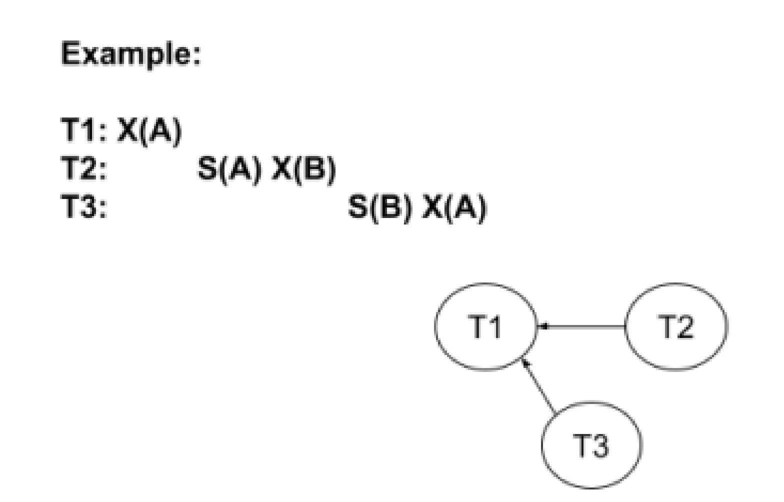
Deadlock

- a cycle of transactions waiting for locks to be released by each other
- Prevention:
  - resource ordering - Screen < Network Card < Printer
- Avoidance: Assign priorities based on age: (now – start_time).

  If Ti wants a lock that Tj holds,
  - Wait-Die: If Ti has higher priority, Ti waits for Tj; else Ti aborts
  - Wound-Wait: If Ti has higher priority, Tj aborts; else Ti waits
- Detection: detect deadlocks by creating and maintaining a "waits-for" graph

Deadlock Detection

- “waits-for” graph
- have one node per transaction
- an edge from Ti to Tj if :
  - Tj holds a lock on resource X
  - Ti tries to acquire a lock on resource X, but Tj must release its lock on resource X before Ti can acquire its desired lock.
- periodically check for cycles in the graph
- “shoot” a transaction in the cycle and abort it

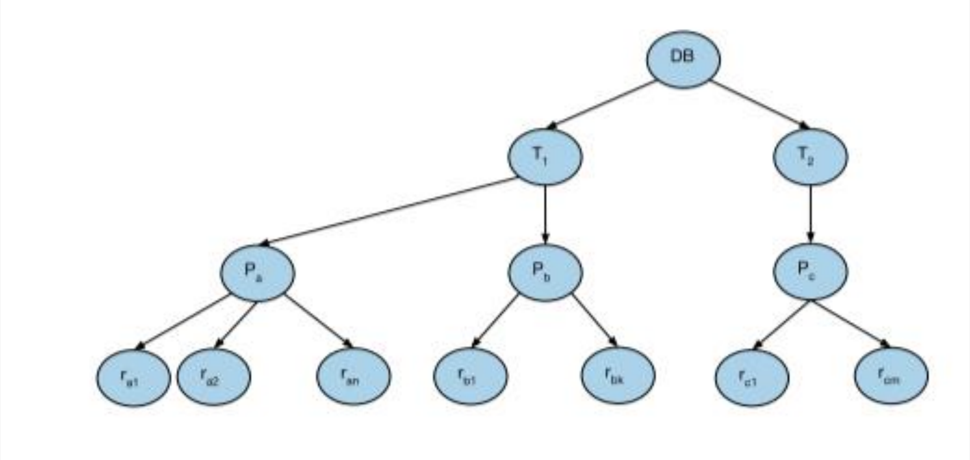


Reminder

- Conflict Dependency Graph: Draw an edge from Ti to Tj iff
  - Tj and Ti operate on the same resource, with Ti operation preceding Tj op
  - At least one of Ti and Tj is a write
  - Used to determine conflict serializability
- Waits-For Graph: Draw an edge from Ti to Tj iff
  - Tj holds a conflicting lock on the resource Ti wants to operate on, meaning Ti must wait for Tj
  - Used for deadlock detection

Lock Granularity

- when we place a lock on a node, we implicitly lock all of its children as well
  - Database -> Tables -> Pages -> Records
- Granularity of locking
  - Fine granularity (lower in tree): High concurrency, lots of locks (high overhead)
  - Coarse granularity (higher in tree): Few locks (low overhead), lost concurrency



Lock Granularity

- 3 new lock modes:
  - IS: Intent to get S lock(s) at finer granularity
  - IX: Intent to get X lock(s) at finer granularity
  - SIX: Like S and IX at the same time
- Compatibility Matrix

Mode	NL	IS	IX	S	SIX	X
NL	Yes	Yes	Yes	Yes	Yes	Yes
IS	Yes	Yes	Yes	Yes	Yes	No
IX	Yes	Yes	Yes	No	No	No
S	Yes	Yes	No	Yes	No	No
SIX	Yes	Yes	No	No	No	No
X	Yes	No	No	No	No	No

Multiple Granularity Locking Protocol

- Each Transaction starts from the root of the hierarchy.
- To get S or IS lock on a node, must hold IS or IX on parent node.
- To get X or IX on a node, must hold IX or SIX on parent node.
- Must release locks in bottom-up order.
- 2-phase and lock compatibility matrix rules enforced as well.
- Protocol is correct in that it is equivalent to directly setting locks at leaf levels of the hierarchy.

Quiz 5

1. **Q 1: T/F - If a term has a large reduction factor, the output of the query will have more tuples than if it had a small reduction factor.** *

Mark only one oval.

- True
- False

True - a larger reduction factor means higher $\frac{|\text{output}|}{|\text{input}|}$ ratio

2. **Q 2: T/F - An equidepth histogram gives better resolution on low-frequency entries than a equiwidth histogram.** *

i.e. it gives more detailed information for these entries.

Mark only one oval.

- True
- False

False - in an equidepth histogram, if you have an entry with low frequency, then it's either going to get clumped together with a bunch of other low frequency entries or with 1 other high frequency entry.

3. **Q 3: When doing a cross join on tables A, B, C, and D, which of the following query plans do we consider?** *

Mark all that apply.

Check all that apply.

- [ ] None of the above
- [ ] (A join (B join C)) join D
- [ ] A join ((B join C) join D)
- [x] ((A join B) join C) join D
- [ ] A join (B join (C join D))
- [ ] (A join B) join (C join D)

left deep joins only

4. **Q4: Which of the following access or join methods will result in an interesting order in a query where we require the output to be sorted?** *

Check all that apply.

- [ ] File scan
- [x] Sort-Merge Join
- [ ] Block-Nested Loops Join
- [x] Clustered Index Traversal
- [ ] Hash Join

Suppose that we have three tables, R, S, and T. We are running the following query:

```
SELECT *
FROM R, S, T
WHERE R.a = S.a
AND S.b = T.b;
```

Assume that our database has no indices and that none of the relations are sorted in any interesting or useful way. Since we only have one possible single-table access method for each table, we ignore the costs of accessing a single table.

Assume that all provided join costs are for the optimal join algorithm for that join.

These are the two-table join costs:
1) S join R = 2,000
2) R join S = 6,000
3) R join T = 5,000
4) T join R = 1,000
5) T join S = 3,000
6) S join T = 4,000

5.**Q 5: Which of the above two-table join plans will be selected?** *

Check all that apply.

1, 5

1)+5) S join R + T join S 5,000

1)+6) S join R + S join T 6,000

2)+5) R join S + T join S 9,000

2)+6) R join S + S join T 10,000

We now add the third table and have the following join costs:
1) (R join S) join T = 10,000
2) T join (R join S) = 6,000
3) R join (S join T) = 12,000
4) T join (S join R) = 11,000
5) (R join T) join S = 10,000
6) S join (R join T) = 7,000
7) (T join R) join S = 14,000
8) S join (T join R) = 16,000
9) (S join T) join R = 13,000
10) (S join R) join T = 15,000
11)(T join S) join R = 20,000
12) R join (T join S) = 9,000

6.**Q 6: Which of these will the optimizer select as your final query plan?** *

Mark only one oval.

For example, the best left-deep plan to join tables A, B, C is either:

- (The best plan for joining A, B) ⋈ C
- (The best plan for joining A, C) ⋈ B
- (The best plan for joining B, C) ⋈ A

10

S join R 2,000
R join S 6,000

10) (S join R) join T 15,000

S join T 4,000
T join S 3000

11) (T join S) join R 20,000