Schedules and Serializability

17.4-17.5

Schedules

- A "schedule" is the abstraction of the activity of simultaneous transactions.
- Everything is stripped away except:
  - transaction identifiers (as subscripts)
  - DB READs and WRITEs and the data they operate on
  - COMMITs and ABORTs
  - The order of these operations

Schedule Notation

- S_a A schedule of one or more transactions
- T_i A transaction.
- $r_i(X)$ Transaction Ti performs a READ of data item X.
- $w_i(X)$ Transaction Ti performs a WRITE of data item X.
- a_i Transaction i aborts.
- c_i Transaction i commits.

Serial Schedules

- If T_i does not overlap T_j, ACID behavior is assured.
  - Notation: $T_i;T_j$ means T_i executes fully, then T_j executes
  - **Called a "serial" schedule**
- $T_1;T_{2;}T_3$ and $T_2;T_3;T_1$ might have different results, but either is acceptable.

Serial<u>izable</u> Schedules

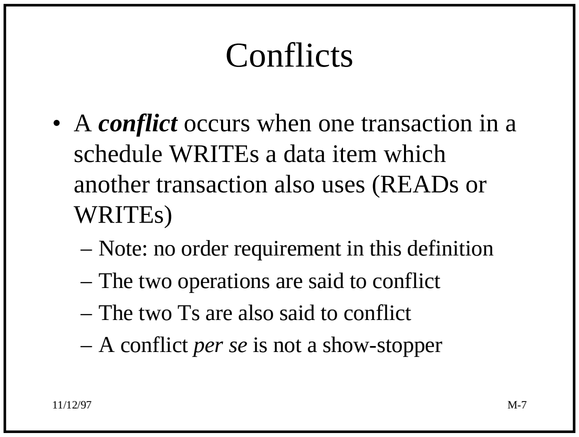
- Serial schedules, though safe, are unacceptably costly
  - Transactions are I/O-bound (most elapsed time is spent waiting for I/O)
- Non-serial (overlapped) schedules allow shorter turn-around and better resource utilization
- A ***serializable*** schedule is one which is equivalent to some serial schedule

Schedules and Serializability Theory

- Schedules are a major tool in studying concurrent processing of transactions
- Goals:
  - be able to recognize when a schedule is serializable

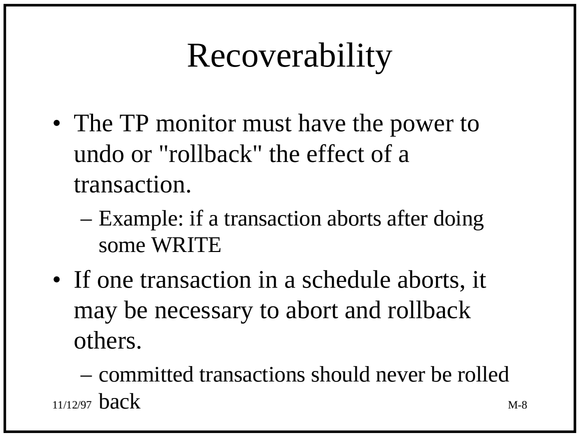
  and/or:
  - be able to force schedules to be serializable

  and/or
  - be able to recognize when a schedule is recoverable if an abort occurs



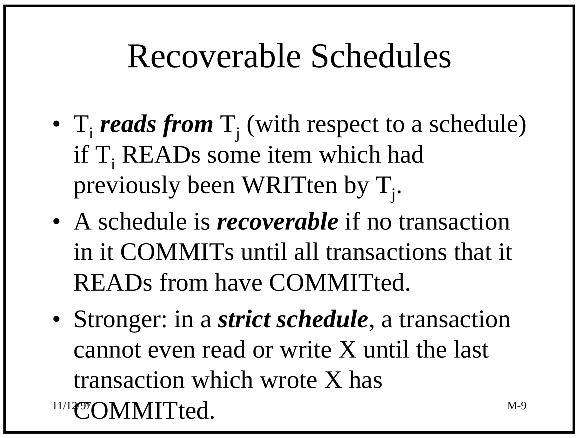
Conflicts

- A ***conflict*** occurs when one transaction in a schedule WRITEs a data item which another transaction also uses (READs or WRITEs)
  - Note: no order requirement in this definition
  - The two operations are said to conflict
  - The two Ts are also said to conflict
  - A conflict *per se* is not a show-stopper



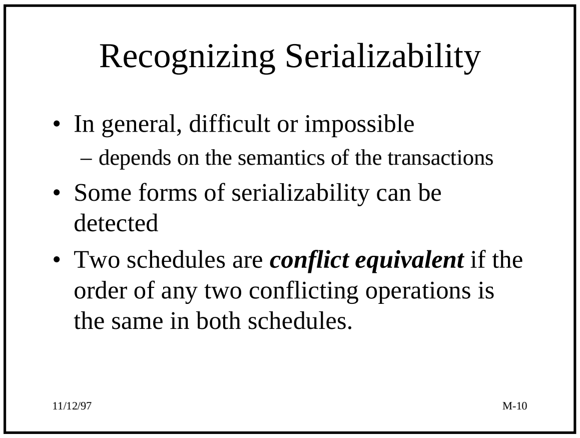
Recoverability

- The TP monitor must have the power to undo or "rollback" the effect of a transaction.
  - Example: if a transaction aborts after doing some WRITE
- If one transaction in a schedule aborts, it may be necessary to abort and rollback others.
  - committed transactions should never be rolled back



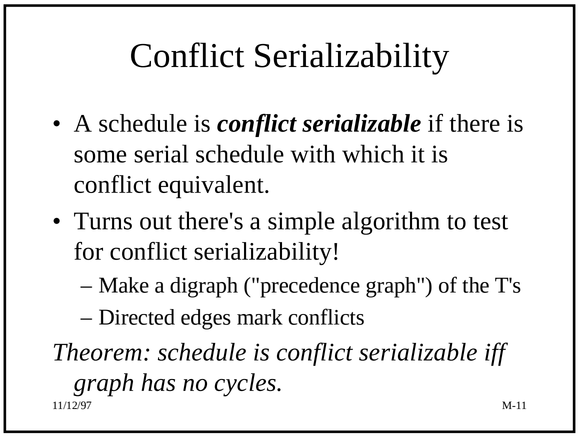
Recoverable Schedules

- T_i ***reads from*** T_j (with respect to a schedule) if T_i READs some item which had previously been WRITten by T_j.
- A schedule is ***recoverable*** if no transaction in it COMMITs until all transactions that it READs from have COMMITted.
- Stronger: in a ***strict schedule***, a transaction cannot even read or write X until the last transaction which wrote X has COMMITted.



Recognizing Serializability

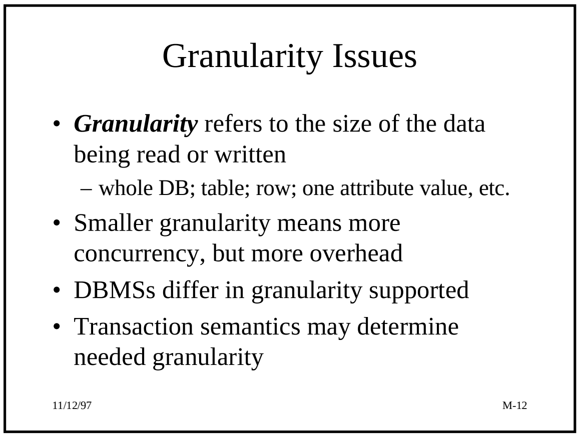
- In general, difficult or impossible
  - depends on the semantics of the transactions
- Some forms of serializability can be detected
- Two schedules are ***conflict equivalent*** if the order of any two conflicting operations is the same in both schedules.



Conflict Serializability

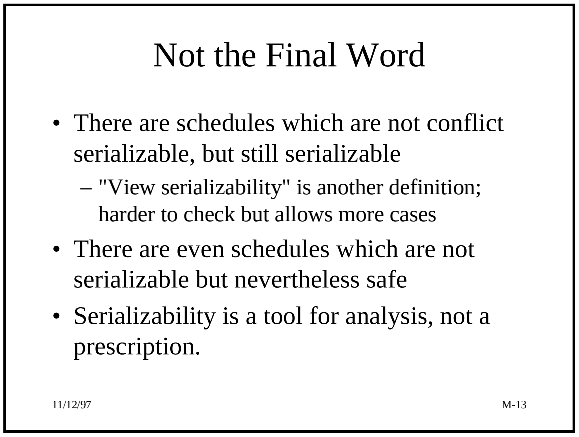
- A schedule is ***conflict serializable*** if there is some serial schedule with which it is conflict equivalent.
- Turns out there's a simple algorithm to test for conflict serializability!
  - Make a digraph ("precedence graph") of the T's
  - Directed edges mark conflicts

Theorem: schedule is conflict serializable iff graph has no cycles.



Granularity Issues

- ***Granularity*** refers to the size of the data being read or written
  - whole DB; table; row; one attribute value, etc.
- Smaller granularity means more concurrency, but more overhead
- DBMSs differ in granularity supported
- Transaction semantics may determine needed granularity



Not the Final Word

- There are schedules which are not conflict serializable, but still serializable
  - "View serializability" is another definition; harder to check but allows more cases
- There are even schedules which are not serializable but nevertheless safe
- Serializability is a tool for analysis, not a prescription.

11/12/97 M-13