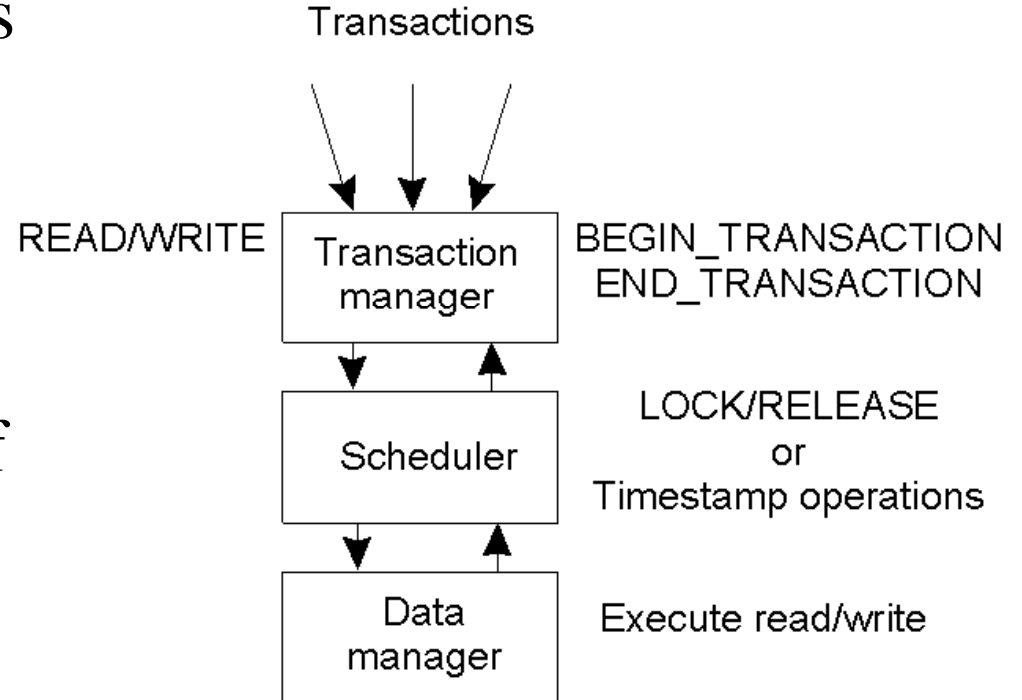


# Concurrency control (1)

- *Concurrency control* is the set of mechanisms put in place to preserve consistency and isolation
  - Trivial solution
    - execute only one transaction at a time
  - But, we are interested in providing concurrency
    - allowing several transactions to be executed simultaneously
- Distributed transactions further complicates concurrency control
  - requires handling multiple databases

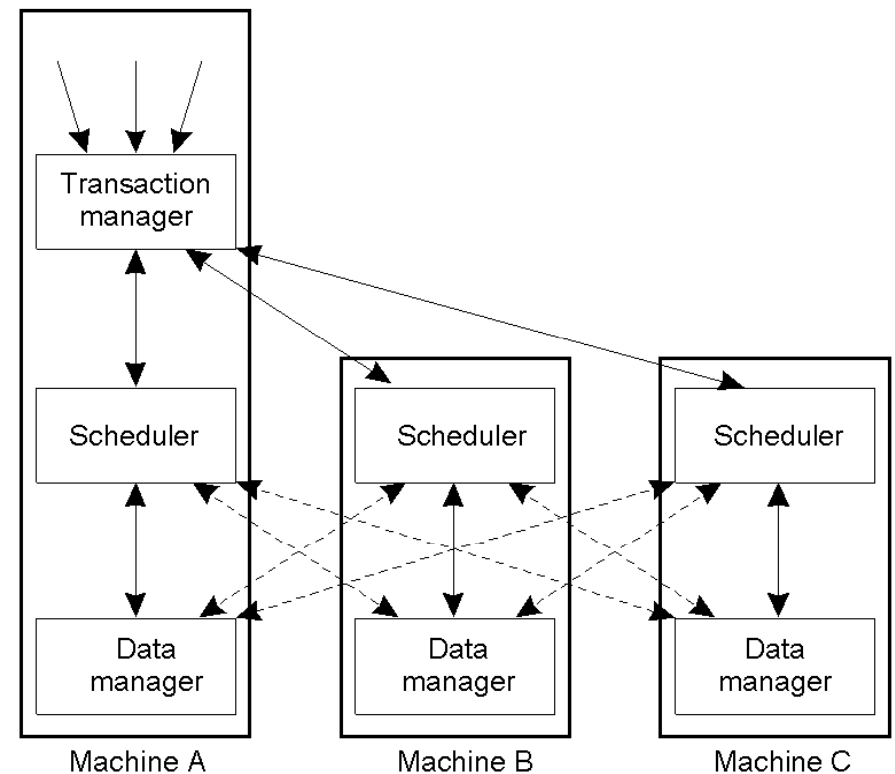
# Concurrency Control (2)

- Concurrency control is best understood in terms of three managers
  - Data manager
  - Scheduler
  - Transaction manager
- The three managers implement a division of tasks
  - concurrency control is responsibility of the scheduler



# Concurrency Control (3)

- The managers scheme can be extended to the distributed scheme



# Concurrency Control Theory

- We want to execute several transactions simultaneously
  - Each transaction individually preserves consistency
  - Serial execution is inefficient
  - what we really want is concurrent (i.e. interleaved) execution
- What we need is a method that enables us to answer the following questions:
  - Give the log of a certain execution of a set of transaction, is that execution legal? (i.e. preserves consistency and isolation)

# Logs

- *Log*: chronological order of actions performed by transactions under a concurrency control algorithm
  - a log over  $T$  describes an interleaved execution of  $T_0, T_1, \dots, T_n$
  - Example:

$T_1 = r1[x] \ r1[z] \ w1[x]$

$T_2 = r2[y] \ r2[z] \ w2[y]$

$T_3 = w3[x] \ r3[y] \ w3[z]$

L1

w3[x] r1[x] r3[y] r2[y] w3[z] r2[z] r1[z] w2[y] w1[x]

L2

w3[x] r3[y] w3[z] r2[y] r2[z] w2[y] r1[x] r1[z] w1[x]

# Serializability

BEGIN\_TRANSACTION  
 $x = 0;$   
 $x = x + 1;$   
 END\_TRANSACTION

(a)

BEGIN\_TRANSACTION  
 $x = 0;$   
 $x = x + 2;$   
 END\_TRANSACTION

(b)

BEGIN\_TRANSACTION  
 $x = 0;$   
 $x = x + 3;$   
 END\_TRANSACTION

(c)

Schedule 1	$x = 0;$ $x = x + 1;$ $x = 0;$ $x = x + 2;$ $x = 0;$ $x = x + 3$	Legal
Schedule 2	$x = 0;$ $x = 0;$ $x = x + 1;$ $x = x + 2;$ $x = 0;$ $x = x + 3;$	Legal
Schedule 3	$x = 0;$ $x = 0;$ $x = x + 1;$ $x = 0;$ $x = x + 2;$ $x = x + 3;$	Illegal

(d)

- a) – c) Three transactions  $T_1$ ,  $T_2$ , and  $T_3$
- d) Possible schedules

# Conflicts

- Two transactions  $T_1$  and  $T_2$  are said to conflict if
  - they access the same data objects, and
  - at least one access is a write, i.e. r-w, w-r, or w-w conflict
- Note that if two transactions have no conflict, then their execution can be interleaved without restrictions

# Equivalent logs

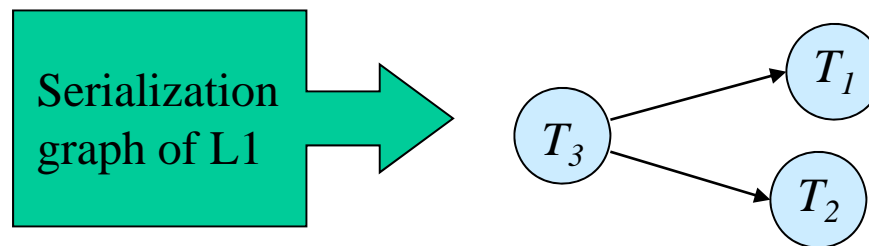
- Two logs are *equivalent* if
  - all the transactions in both logs see the same state of the database,
  - they leave the database in the same final state
- Two logs over a transaction set are clearly equivalent iff:
  - every read operation reads from (i.e. sees the value written by) the same write in both logs, and
  - both logs have the same final writes
- Example: L1 and L2 in previous example are equivalent

# Serial Logs

- *Serial* log:
  - a log in which all actions of a transaction terminate before any action of the next transaction starts
  - example: L2 is a serial log
- *Serializable* log:
  - a log in which actions from several transactions  $T_0, T_1, \dots, T_n$  are interleaved, and that has the same output and the same effect on the database as the serial execution of a permutation of  $T_0, T_1, \dots, T_n$
  - Example: log L1 is equivalent to serial log L2
- A serializable log is equivalent to a serial log, thus represents a correct execution
  - question: is there some criterion for telling if a log is serializable?

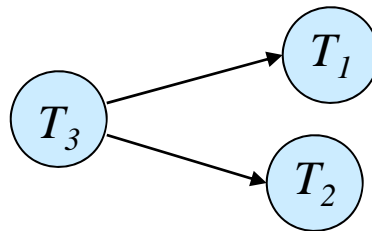
# Serialization Graph

- Let  $L$  be a log over a set of transactions  $T_0, T_1, \dots, T_n$ 
  - the *serialization graph*  $SG(L)$  of  $L$  is a directed graph in which the nodes are the transactions  $T_i$ , and whose edges satisfy the condition:
    - edge  $T_i \rightarrow T_j$  then for some  $x$  either  $ri[x] < wj[x]$  or  $wi[x] < rj[x]$  or  $wi[x] < wj[x]$
    - i.e. an edge  $T_i \rightarrow T_j$  denotes a conflict between actions of  $C$  and  $T_j$
  - example:



# The Serializability Theorem

- Th.: A log  $L$  is serializable iff  $SG(L)$  is acyclic
  - no cycles in  $AG(L) \Leftrightarrow L$  is serializable
- The serial log corresponding to  $L$  can be determined by topologically sorting  $AG(L)$ 
  - example:



$T_3 \rightarrow T_1 \rightarrow T_2$  OR  $T_3 \rightarrow T_2 \rightarrow T_1$