

Concurrency Control Algorithms

- Given a number of conflicting transactions, the serializability theory provides criteria to study the correctness of a possible schedule of execution
 - it does not provide a practical way to produce the schedule
 - for this purpose of a concurrency control algorithm is needed
- These algorithms control the interleaving of conflicting transactions so that database consistency is preserved
 - i.e. the outcome is equivalent to a serial execution
- Two main classes of algorithms:
 - lock based
 - timestamp based

Lock Based Algorithms

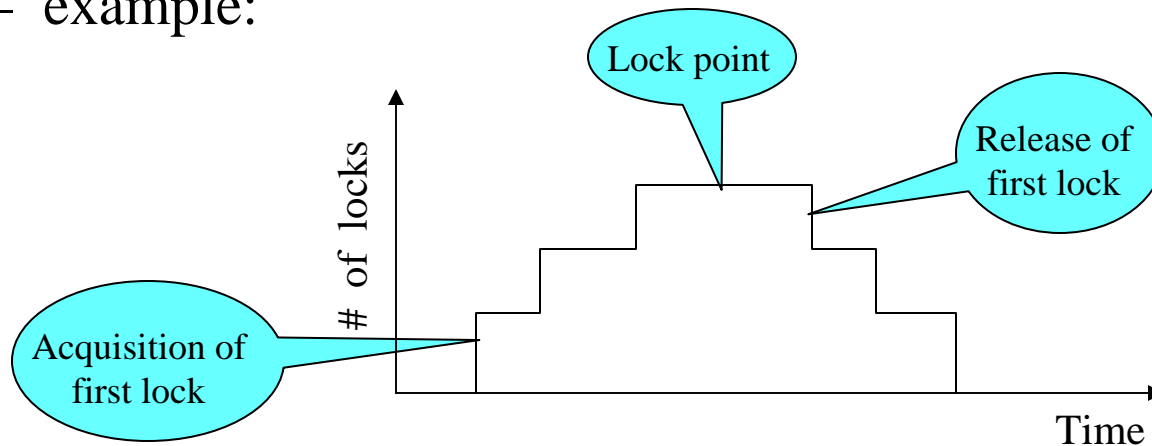
- In these algorithms, transactions must lock data objects before accessing them
- A transaction is *well-formed* if it
 - locks a data before accessing it
 - does not lock a data object more than once, and
 - unlocks all the locked data objects before completing

Static locking

- A transaction acquires the lock on all the objects it needs at start of execution
 - release all the locks at end of execution
- Pros:
 - very simple
- Cons:
 - zero concurrency between transactions with a conflict
 - requires a priori knowledge of all the data needed

Two-Phase locking

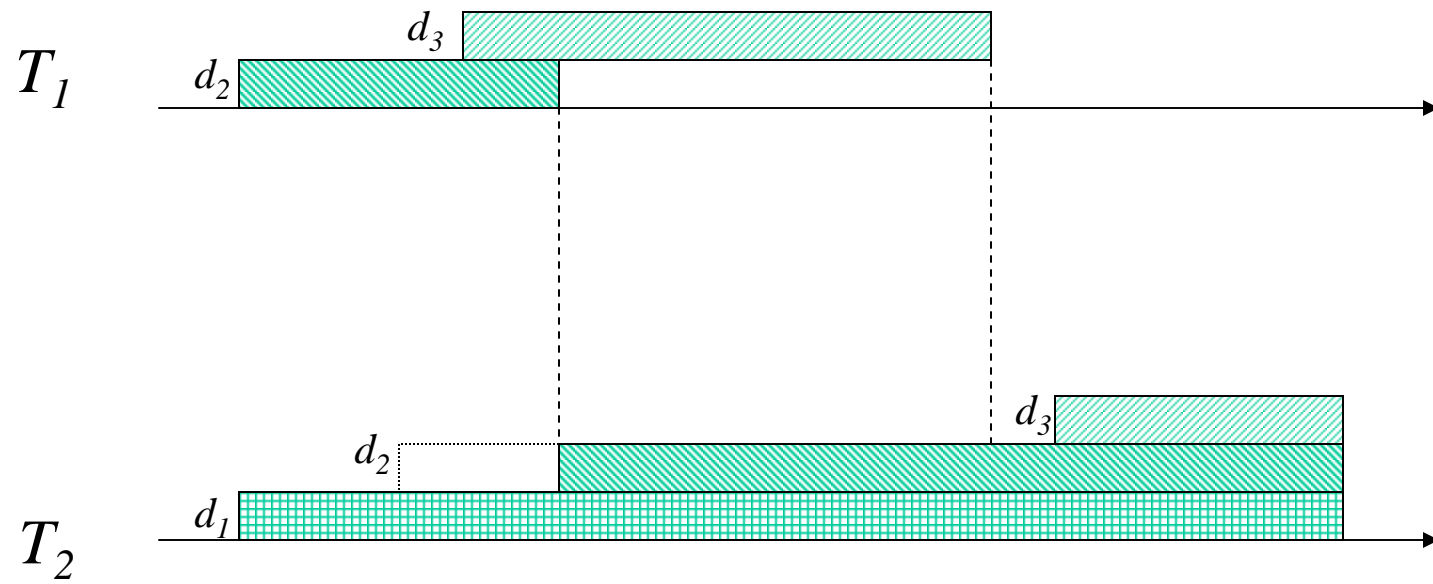
- Dynamic algorithm in which a transaction
 - requests a lock on a data object when it needs the object
 - cannot requests a lock anymore after it has unlocked an object
- Thus algorithm has *growing phase* plus a *shrinking phase*
 - intermediate phase is called *lock point*
 - example:



Two-Phase Locking: properties

- It can be shown that if a set of transactions
 - are well-formed, and
 - follow a two-phase scheme to get/release data objectsthen all legal logs are serializable
 - (legal log: a log in which a transaction trying to lock an already locked object waits on the lock)
- Two-Phase locking increases concurrency over the static scheme because objects are locked for a shorter period

2PL example

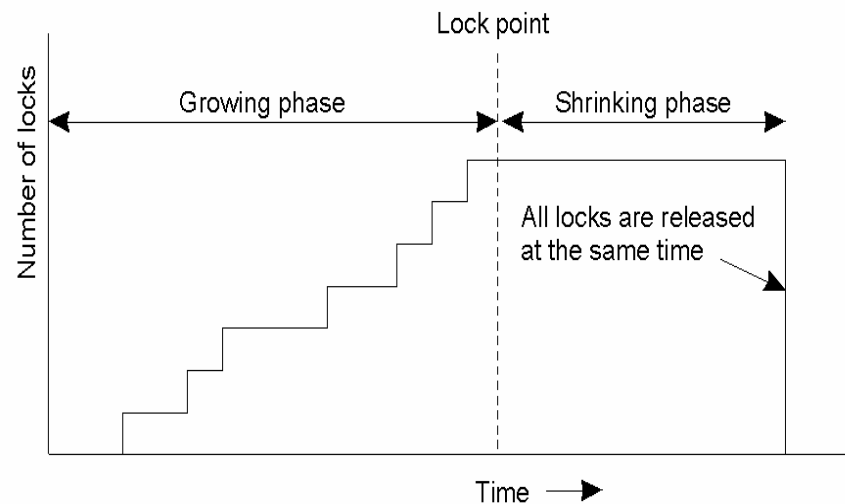
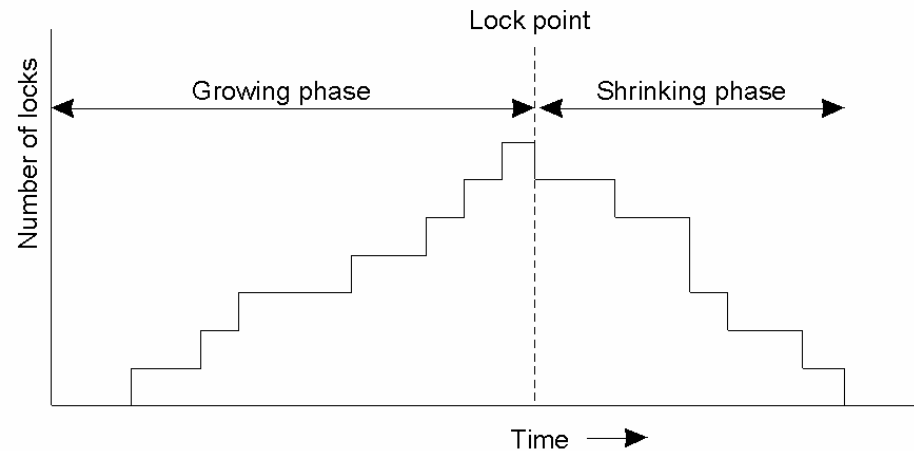


Two-Phase Locking: problems

- Deadlocks due to circular wait.
 - Possible solutions:
 - kill one blocked transaction, then restore and unlock data
 - either acquire all locks or none
 - assign priorities to transactions
- Cascaded rollback - one transaction being rolled back causes others that have accessed modified data to be rolled back too.
 - Solution: *strict 2PL*
 - all locks released in one atomic operation (commit)

Strict Two-Phase Locking

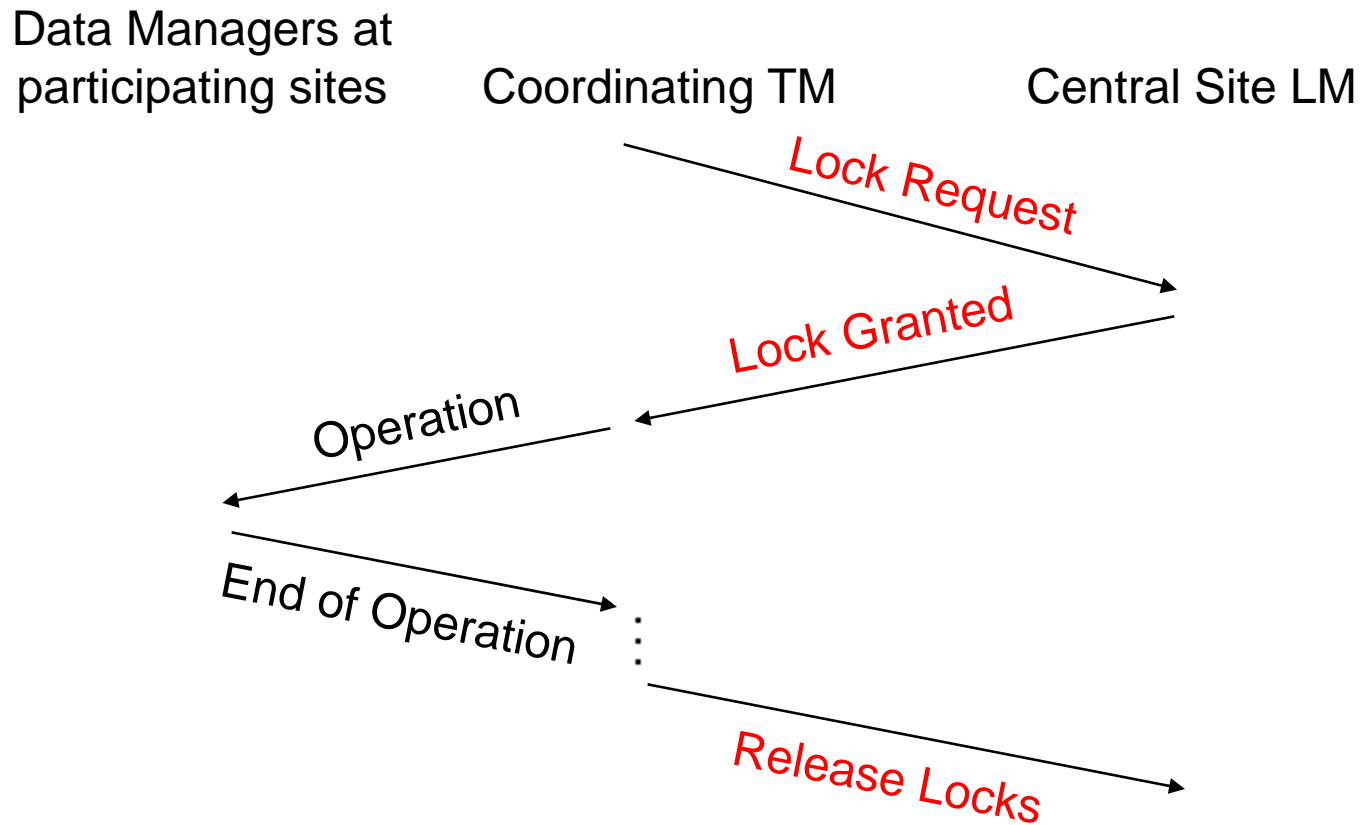
- Real cause of cascaded rollback is first transaction releasing the lock on **some** objects making them prematurely accessible
- Strict 2PL
 - all locks released in one atomic operation
 - Cascaded rollback eliminated at the cost of reduced concurrency



Implementations of 2PL

- Centralized 2PL
 - Single site put in charge of granting and releasing locks
 - each transaction manager
 - communicates with centralized lock manager,
 - Talks directly to data manager once lock is acquired
- Distributed 2PL
 - Schedulers in each machine
 - Takes care of granting and releasing locks
 - Forward read/write operations to local data manager

Centralized 2PL



Distributed 2PL

