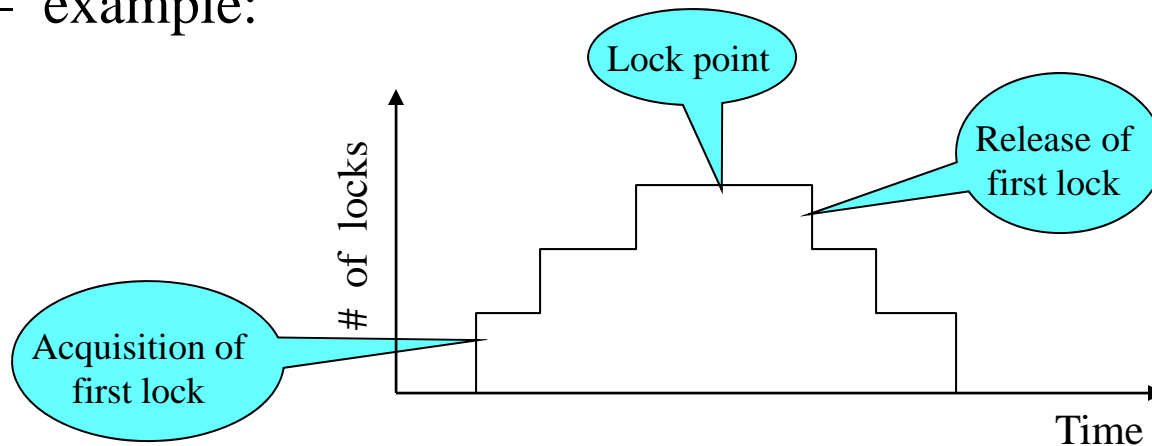


Two-Phase locking

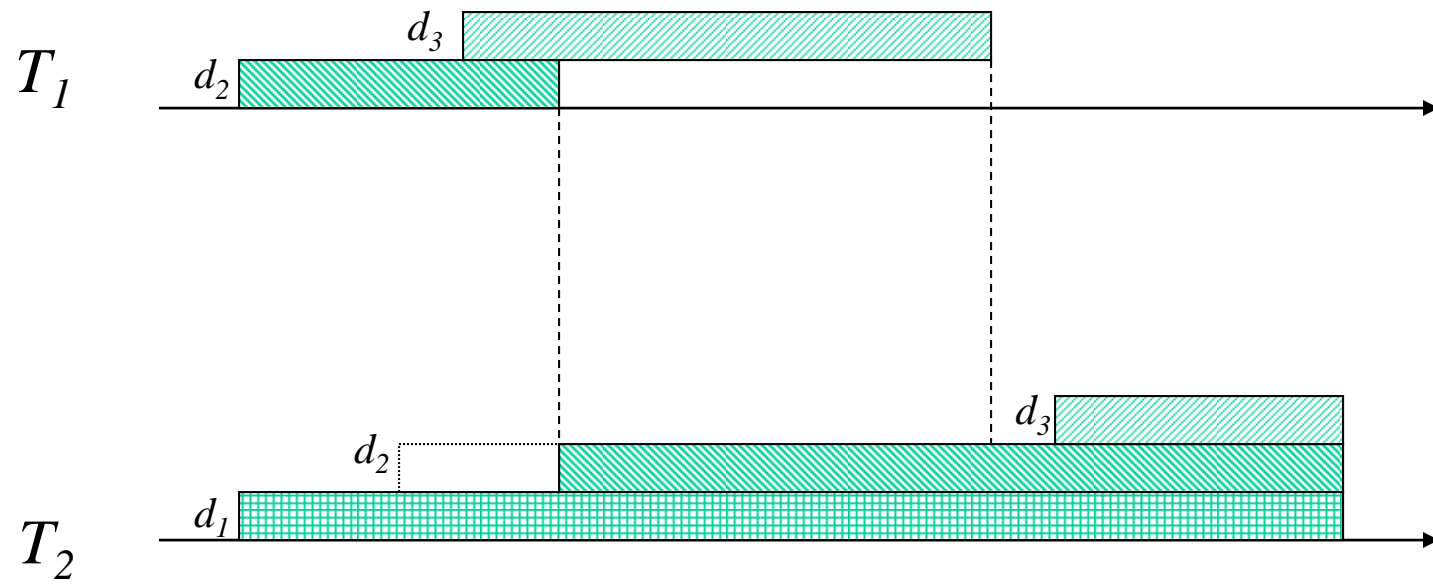
- Dynamic algorithm in which a transaction
 - requests a lock on a data object when it needs the object
 - cannot request a lock anymore after it has unlocked an object
- Thus algorithm has *growing phase* plus a *shrinking phase*
 - intermediate phase is called *lock point*
 - example:



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

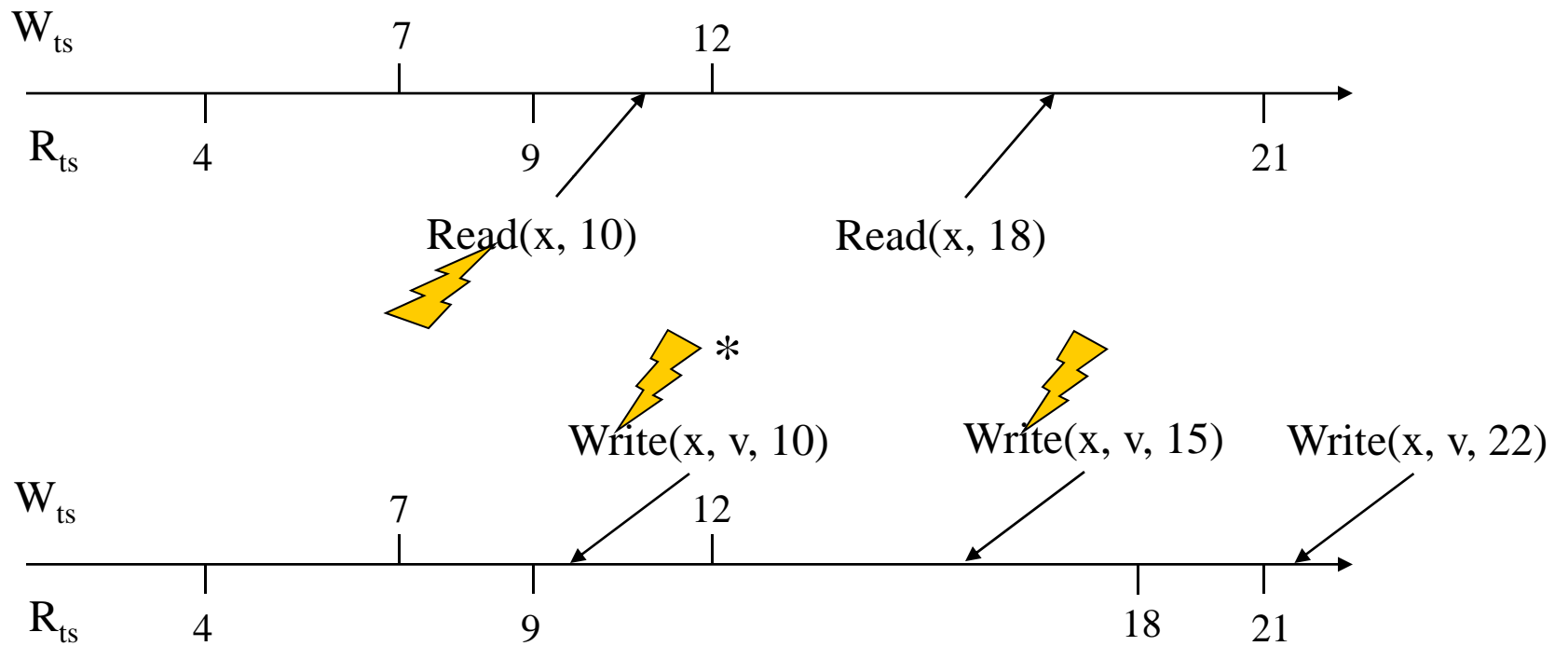
Timestamp based algorithms

- Basic idea: use transaction timestamp to decide the order of execution of conflicting actions
 - serialization order of transaction is thus decided a priori
- Lamport's logical clocks used to timestamp all transactions
 - The TM assigns a timestamp to each transaction
 - Timestamp is then used to label each read/write requests
 - Scheduler uses timestamps to order read/write requests
- Various algorithms have been proposed that differ in how ordering of read/write actions is enforced

Basic Timestamp Ordering Alg. (BTO)

- For each data objects, the largest timestamp so far is kept for both read and write operations:
 - $R_{ts}(\text{object})$, $W_{ts}(\text{object})$ are maintained for each object
 - Read/write requests are denoted with $\text{Read}(x, TS)$, $\text{Write}(x, v, TS)$
- Rule for handling requests:
 - $\text{Read}(x, TS)$:
 - if $TS < W_{ts}(x)$ then reject request, abort transaction
 - else execute the Read and set $R_{ts}(x)$ to $\max\{R_{ts}(x), TS\}$
 - $\text{Write}(x, v, TS)$:
 - if $TS < R_{ts}(x)$ or $TS < W_{ts}(x)$ then reject request, abort transaction
 - else execute the Write and set $W_{ts}(x)$ to TS
 - Aborted transactions are restarted with new timestamp

BTO examples



BTO disadvantages

- BTO is obviously correct
 - every execution is equivalent to a serial execution in timestamp order
- ... but has some shortcomings
- Storage overhead
 - Two timestamps need to be maintained for each object
- The abort-restart method is inefficient
 - It can result in a transaction repeatedly restarting without ever completing

Deadlock & starvation: definitions

- **Deadlock** occurs when a set of processes is blocked waiting on requests for resources that can never be satisfied
 - while holding some resources, the processes request other resources held by processes in the same set
 - i.e. there is a notion of circular wait
- **Starvation** occurs when a process waits for a resource that continually becomes available but is never assigned to it for priority reasons or for a design flaw

Deadlock and starvation: differences

- **Process status:**
 - deadlock: processes are permanently blocked because the resources never become available
 - starvation: it is not certain the process will ever acquire the requested resource
- **Resource status**
 - deadlock: contended resources are not in use
 - starvation: contended resources in continuous use (by others)

Causes of deadlocks

- Four necessary conditions for deadlock to occur are:
 - **Exclusive access:** processes require exclusive access to a resource
 - **Wait while hold:** processes hold on previously acquired resources while waiting for additional resources
 - **No preemption:** a resources cannot be preempted from a process without aborting the process
 - **Circular wait:** there is a set of blocked processes involved in a circular wait
- The first three properties are generally desirable
 - respectively to i) preserve resource integrity, ii) increase resource utilization, iii) reduce waste of CPU time

Deadlock handling policies

- Deadlock prevention
 - the system is designed so that granting requests never leads to a deadlock
- Deadlock detection
 - the system periodically (or when deadlock suspected) checks for deadlocks, and a recovery procedure is started if one is detected
- Deadlock avoidance
 - resources are granted only if the resulting system state is *safe* i.e. there is at least one sequence of execution in which all processes run to completion