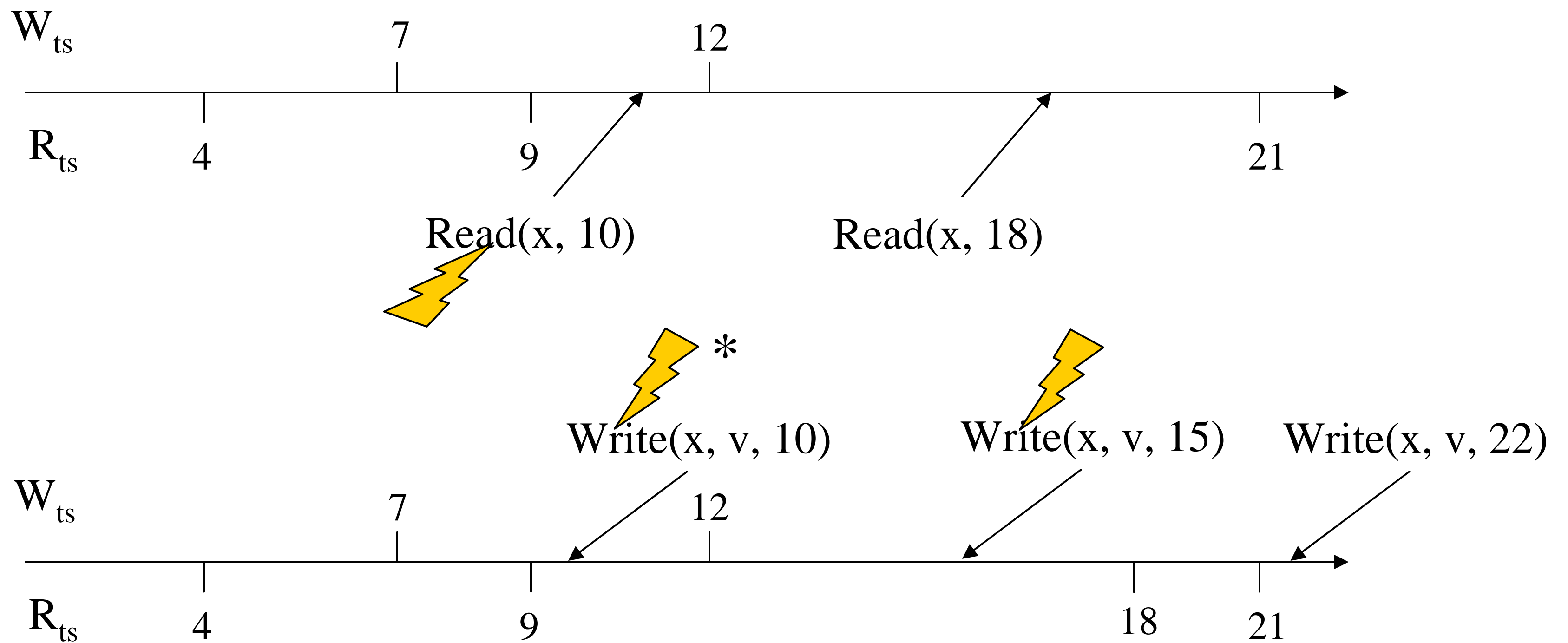
Timestamp based algorithms

- 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}(object), W_{ts}(object) are mantained for each object
  - Read/write requests are denoted with Read(x, TS), Write(x, v, TS)
- Rule for handling requests:
  - Read(x, TS):
    - if $TS < W_{ts}(x)$ then reject request, abort transaction
    - else execute the Read and set $R_{ts}(x)$ to $\max\{R_{ts}(x), TS\}$
  - 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

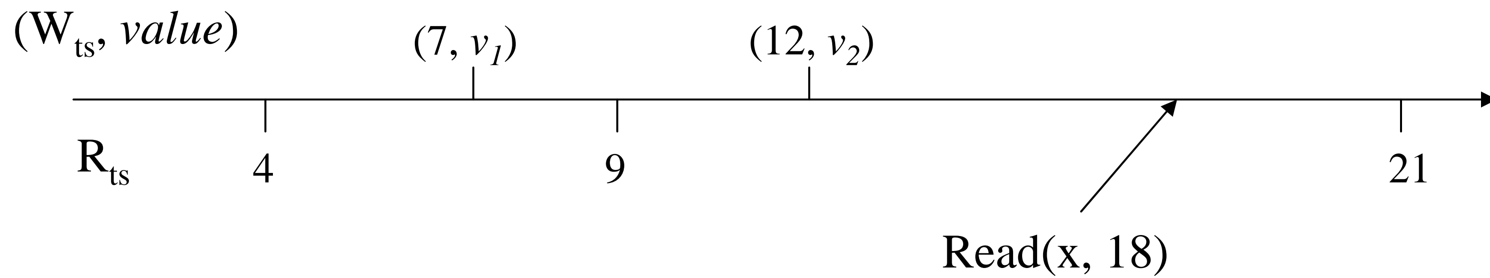
- BTO is obviously correct
  - every execution is equivalent to a serial execution in timestamp order

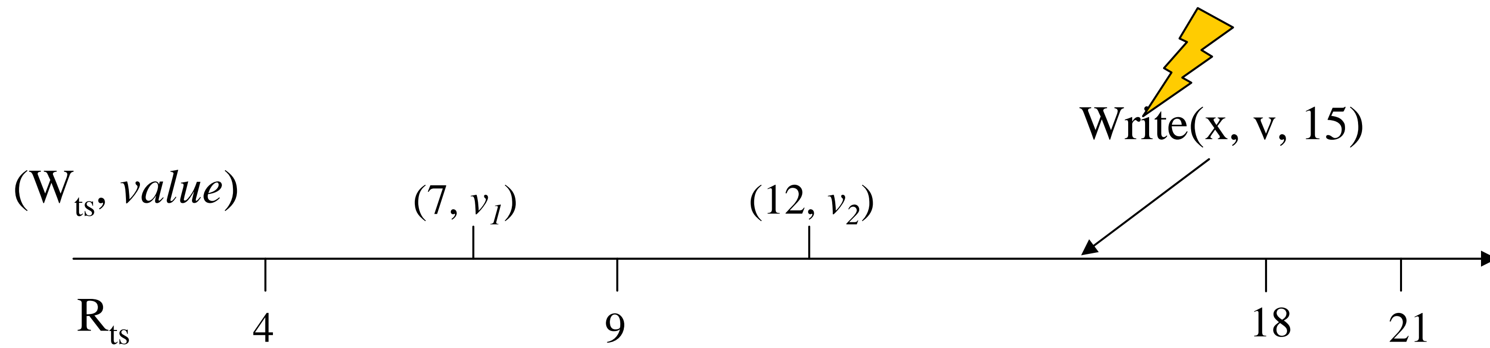
  … but has some shortcomings
- Storage overhead
  - Two timestamps need to be maintained for each object
- The abort-restart method is inefficient
  - It can result in a transaction repeatedly restarting without ever completing

Multiversion Timestamp Ordering Alg. (MTO)

- MTO maintains a history of R_{ts} and (W_{ts}, *value*) pairs - called *versions* - for each data object
- Rule for handling requests:
  - Read(x, TS):
    - Read the version of x with timestamp < TS, then add TS to x's history
  - Write(x, v, TS):
    - If there is a $R_{ts}(x)$ larger than TS (and smaller than next W_{ts}) then request is rejected
    - else new version of x is created with timestamp TS

MTO examples





MTO trade-offs

- Pros:
  - It can be shown that the MTO algorithm is correct
    - i.e. every execution is equivalent to a serial execution in timestamp order
  - MTO reduces the number of transaction aborts over BTO and TWR
- Cons:
  - Requires a lot of storage since multiple versions are kept for each data object
    - techniques exist to delete old versions

Conservative Timestamp Ordering alg. (CTO)

- The scheduler maintains two queues – Rq and Wq - per TM
  - Queues hold respectively read and write requests in timestamp order
  - TM sends requests in timestamp order, communication is FIFO
- Rule for handling requests:
  - Read(x, TS):
    - If every Wq is nonempty and the first write on each Wq has timestamp > TS, then execute request
    - Else request is enqueued in Rq
  - Write(x, v, TS):
    - If all Rq and Wq are nonempty and the first read/write on each Rq/Wq has timestamp > TS, then execute request
    - Else request is enqueued in Wq
  - Whenever a request is either enqueued or executed, previously enqueued requested are tested to see if they can be executed according to the above rules

CTO tradeoffs

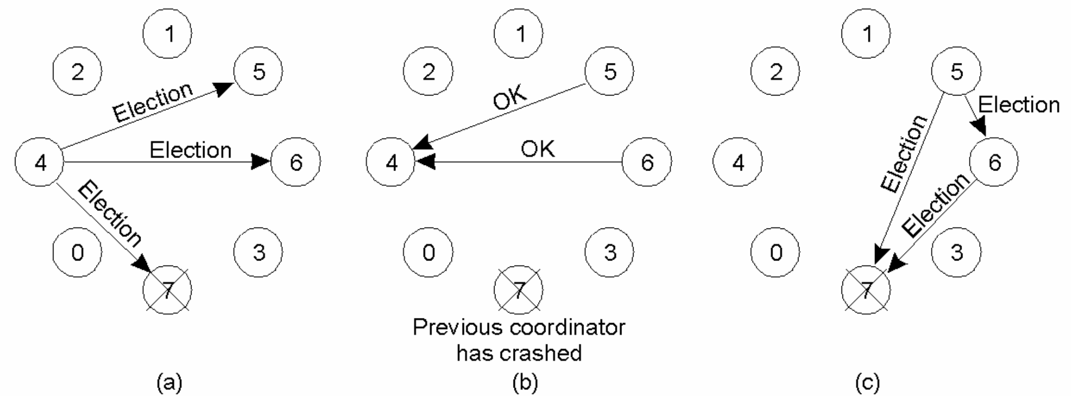
- Cons:
  - It eliminates aborts and restart of transactions
- Pros:
  - Termination is not guaranteed
    - For example if some TM never sends a request to some DM, the DM will wait forever due to the empty queue
    - The problem can be eliminated if all TM communicate regularly
  - The algortihm is overly conservative
    - Not only conflicting actions but all actions are executed in timestamp order

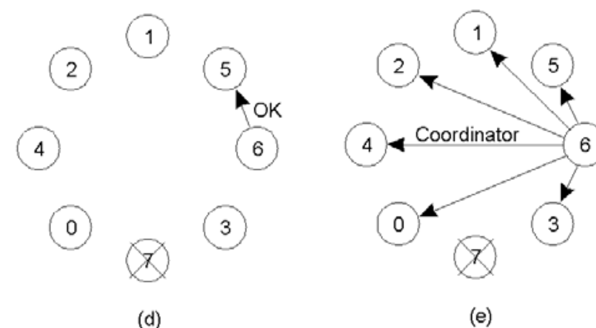
Election algorithms

- Many distributed algorithms require one process to act as a coordinator
  - Problem: how to coordinate the choice of a coordinator in a distributed fashion?
- The purpose of election algorithms is to ensure that when an election is started, it terminates with all processes agreeing on the identity of the coordinator
  - Bully algorithm
  - Ring algorithm
- Assumptions:
  - Processes are numbered, and each process knows the process number of everybody else
  - Processes do not know which ones are currently up and running and which ones are down

The Bully Algorithm (1)

- The bully algorithm steps:
  - Step 1: *Election* message is sent out by starting node; all those with larger ID reply with OK message and start new election
  - Repeat until only node with largest ID remains
  - Step 3: *Coordinator* messages inform everybody of election outcome
- Example
  - Process 4 starts an election
  - Process 5 and 6 respond, telling 4 to stop
  - Now 5 and 6 each hold an election
  - Process 6 tells 5 to stop
  - Process 6 wins and tells everyone





A Ring Algorithm

- Election algorithm steps:
  - Assumption: nodes are (logically or physically) arranged in a ring
  - Step 1: *election* message is sent by starting node to its successor
  - Every node appends its own ID to the messages and passes it over
  - Step 2: once the starting node gets message back, it chooses coordinator (nodes with largest ID); *coordinator* message is then sent around to inform everybody
- Example
  - Both nodes 2 and 5 start election, send *election* message around
  - They will end up choosing same coordinator (node 6) out of their list of candidates