Local Recovery

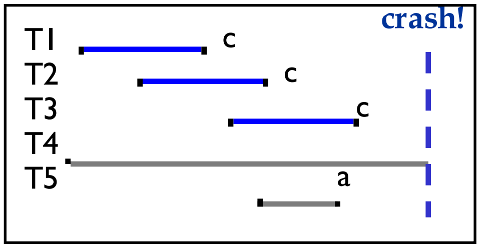
Failure Types

- ❑ Transaction Failures
  - ☆ transaction abort has been discussed before
- ❑ System Failure:
  - ☆ Main memory is lost
  - ☆ Disk survives
  - ☆ have to write objects on regular basis to disk
  - ☆ Assumption: one disk I/O is atomic:
    - ● assume a system failure and disk I/O for object:
      - ▲ either object is completely written to disk or not at all (disk value is completely the old value)
- ❑ Media Failure
  - ☆ we ignore disk failures

Transaction Failures

- Local UNDO during normal processing
  - ☆ whenever a transaction aborts, *undo* updates of aborted Xact
  - ☆ How to undo an update on object X:
    - keep before-image of x (what is the before-image in case of an object?)
    - do the original update on a local copy (similar to as discussed in optimistic concurrency control); discard local copy upon abort
    - provide compensating operation:
      - ▲ increment(X,5) ==> decrement(X,5)fr
      - ▲ set x=5: hard to find compensating operation: need before image
      - ▲ compensating operation application dependent
  - ☆ how to abort r1(x), w1(x), r1(y), w1(y), w1(x), w1(z) ?
    - undo in reverse order: $w^{-1}(z)$, $w^{-1}(x)$, $w^{-1}(y)$, $w^{-1}(x)$
  - ☆ for local UNDO it is enough to keep information in main memory

System Failure



- ❖ Desired Behavior after system restarts:
  - – T1,T2 & T3 should be durable.
  - – T4 & T5 should be aborted (effects not seen).

What to do?

- ❑ Ideal: Write changes exactly at commit time to disk
  - ☆ If transaction commits before crash: all changes on disk
  - ☆ If transaction aborts before crash: no changes on disk
  - ☆ If transaction active at time of crash: no changes on disk
- ❑ Problem: database larger than one I/O
- ❑ Idea: Shadowing
  - ☆ two copies of database: one committed version and one working version
  - ☆ one "master record" that can be updated in one I/O
  - ☆ master record indicates latest committed version
- ❑ Note: Shadowing is not used in real systems because of efficiency problems; it is described here only for illustration purposes

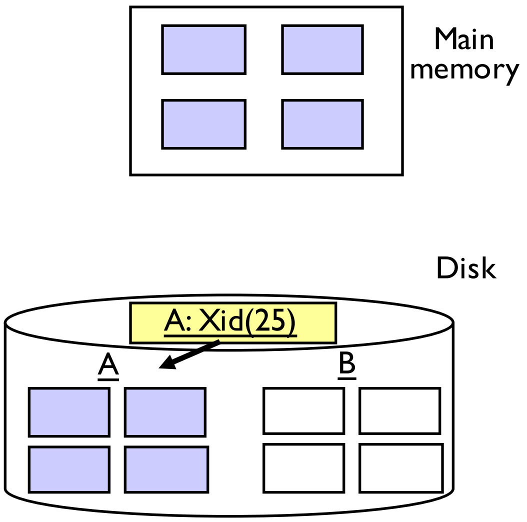
Shadowing

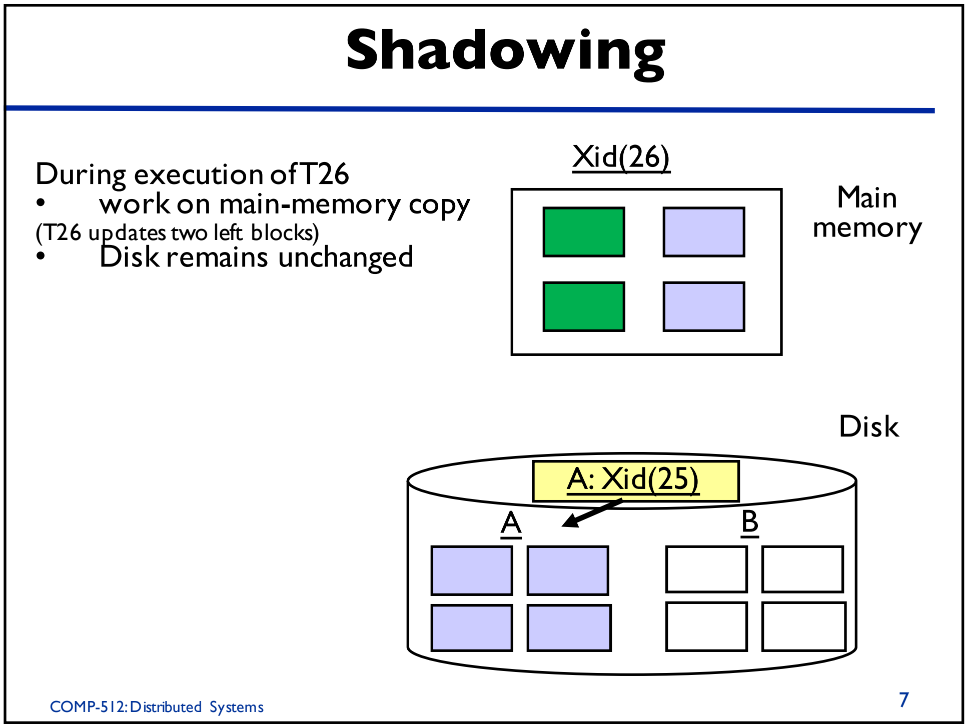
Assume:

- T25 writes all 4 pages purple

After commit of T25

- Location A has changes of T25
- Yellow Master record points to location A and indicates that T25 was the last t commit





Shadowing

During execution of T26
- work on main-memory copy

(T26 updates two left blocks)
- Disk remains unchanged

Xid(26)

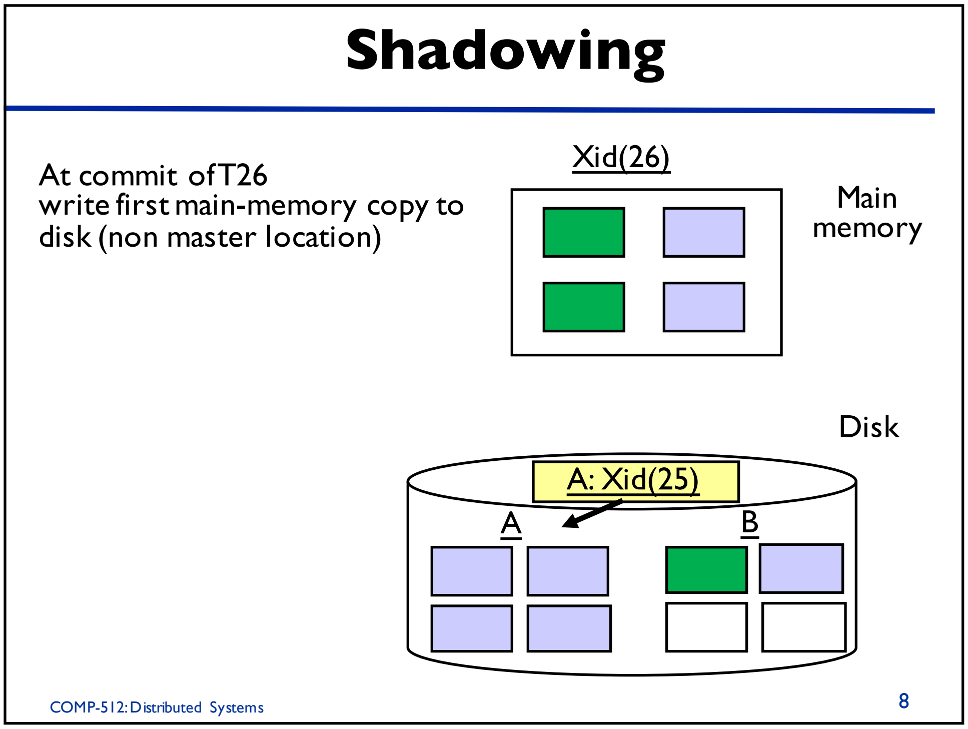
Main memory

Disk

A: Xid(25)

A

B



Shadowing

At commit of T26
write first main-memory copy to disk (non master location)

Xid(26)

Main memory

Disk

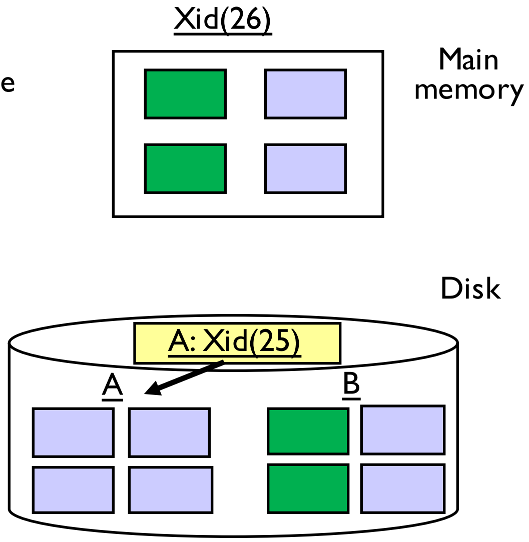
A: Xid(25)

A

B

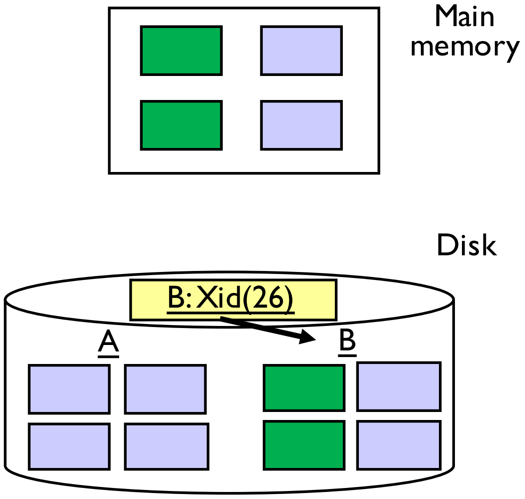
Shadowing

the write at commit time is non-atomic and might take time



Shadowing

Last step of commit of T26: change master record to point to new location and identify last committed transaction

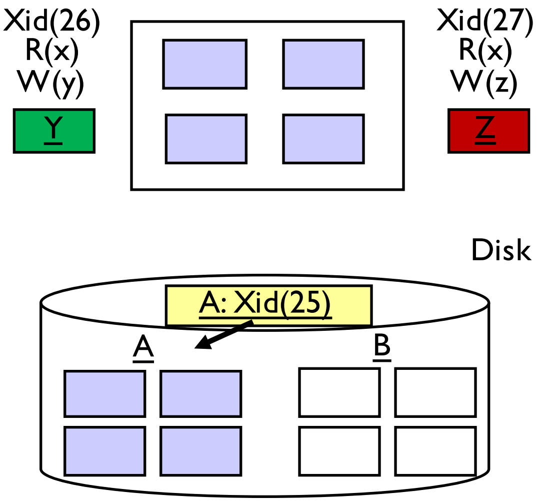


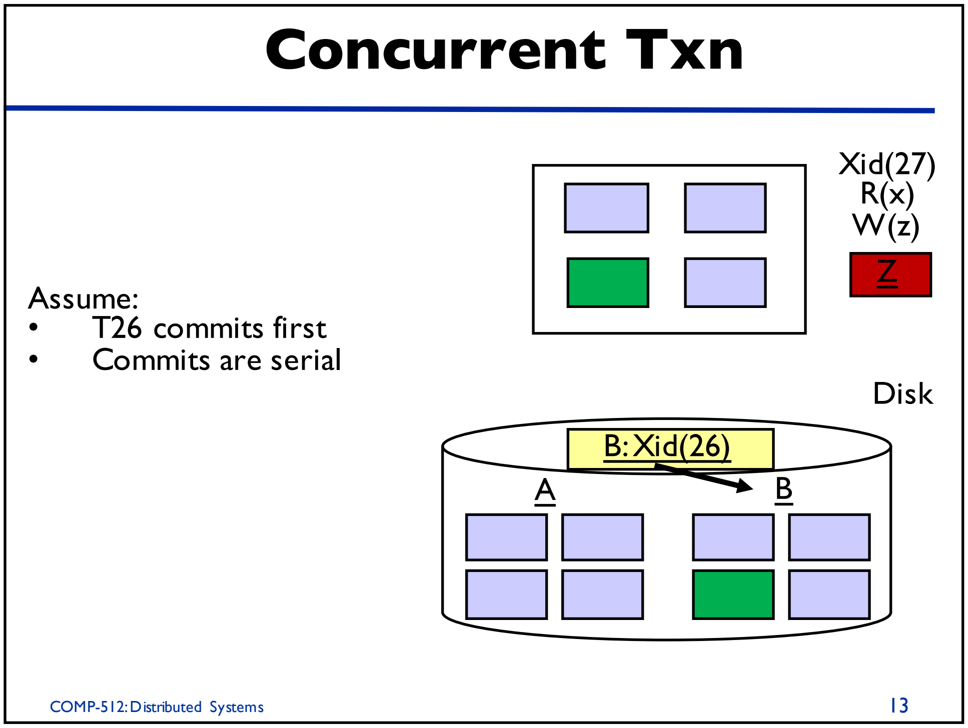
Solution

- ❑ Simple solution (shadow paging) assuming serial execution of transactions
  - ☆ two copies A/B of database on stable storage
    - Assume A to be the latest committed copy
  - ☆ latest version of database in main-memory (*main-memory copy*)
  - ☆ master record
    - contains id of last committed transaction
    - contains pointer to latest committed copy (i.e., A)
  - ☆ Upon execution of T: update main-memory copy
  - ☆ Upon commit request of T:
    - first write main-memory copy to B (might be non-atomic).
    - Then write master with pointer to B and transaction id to disk (one atomic I/O).
  - ☆ Upon abort request of T:
    - discard main-memory copy, read A into main memory
  - ☆ If crash before writing master record:
    - A remains latest committed copy
    - Upon restart of system A is read into main-memory copy
    - T automatically aborted
  - ☆ If crash after commit of T but before commit of any other transaction:
    - master record contains T and pointer to B.
    - i.e., B is latest committed copy
    - Upon restart of system B is read into main-memory copy

Concurrent Txn

Assume:
- T26 and T27 concurrent
- Each has their own local copies

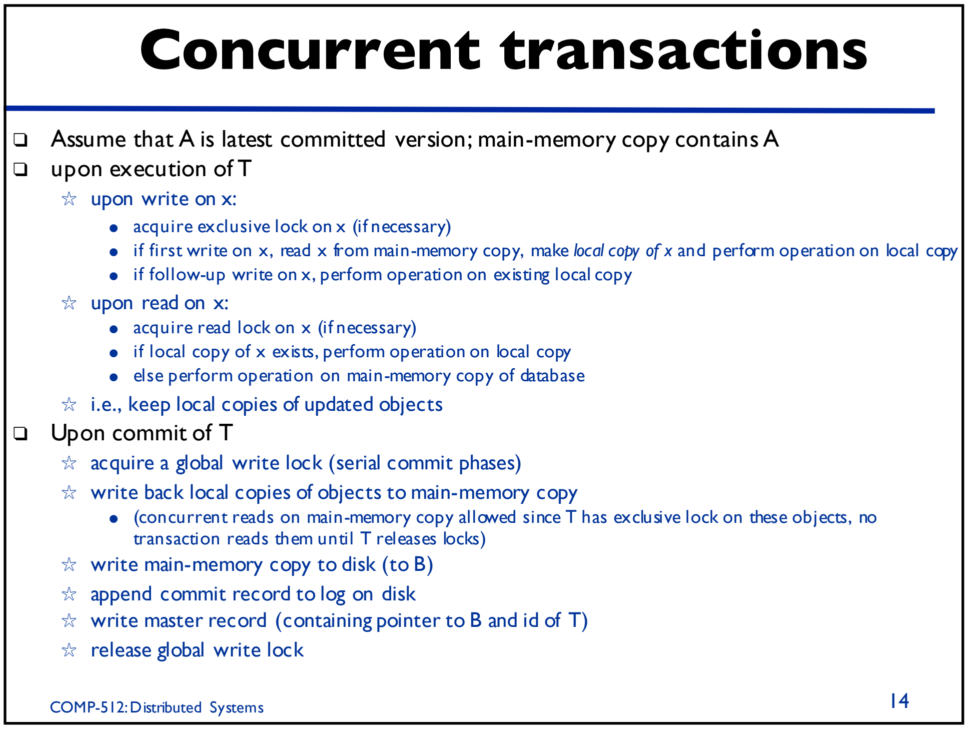




Concurrent Txn

Assume:

- T26 commits first
- Commits are serial



Concurrent transactions

- ❑ Assume that A is latest committed version; main-memory copy contains A
- ❑ upon execution of T
  - ☆ upon write on x:
    - acquire exclusive lock on x (if necessary)
    - if first write on x, read x from main-memory copy, make *local copy of x* and perform operation on local copy
    - if follow-up write on x, perform operation on existing local copy
  - ☆ upon read on x:
    - acquire read lock on x (if necessary)
    - if local copy of x exists, perform operation on local copy
    - else perform operation on main-memory copy of database
  - ☆ i.e., keep local copies of updated objects
- ❑ Upon commit of T
  - ☆ acquire a global write lock (serial commit phases)
  - ☆ write back local copies of objects to main-memory copy
    - (concurrent reads on main-memory copy allowed since T has exclusive lock on these objects, no transaction reads them until T releases locks)
  - ☆ write main-memory copy to disk (to B)
  - ☆ append commit record to log on disk
  - ☆ write master record (containing pointer to B and id of T)
  - ☆ release global write lock