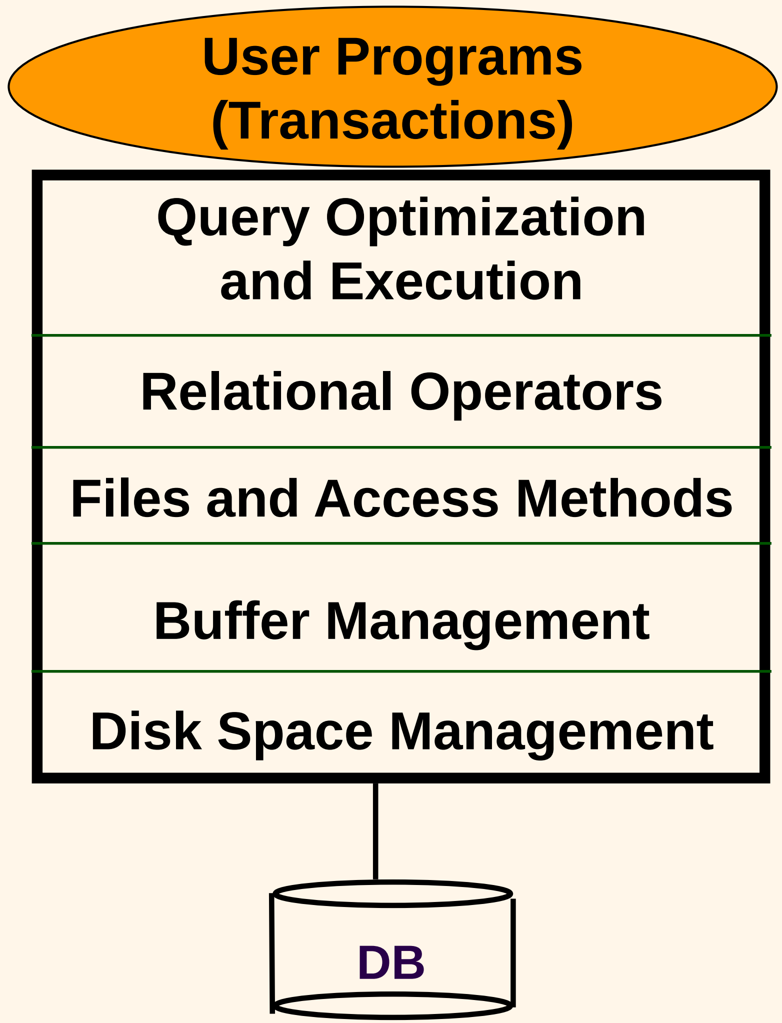
COMP7640
Database Systems & Administration

Crash Recovery

Where Are We Now?



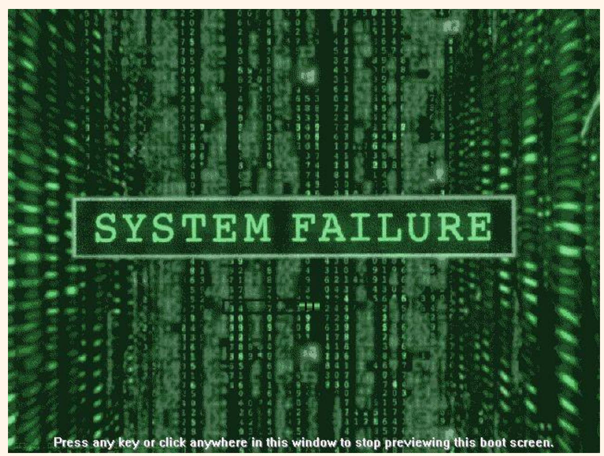
ACID Properties

Each transaction must have:

- **Atomicity.** All actions completed or nothing at all.
- **Consistency.** No violation of any user-defined constraint after the transaction finishes.
- **Isolation.** Concurrent transactions don't interfere each other. Each can think that it's the only running transaction.
- **Durability.** If a transaction is committed, its changes to the database are permanent, even in the presence of system failures.

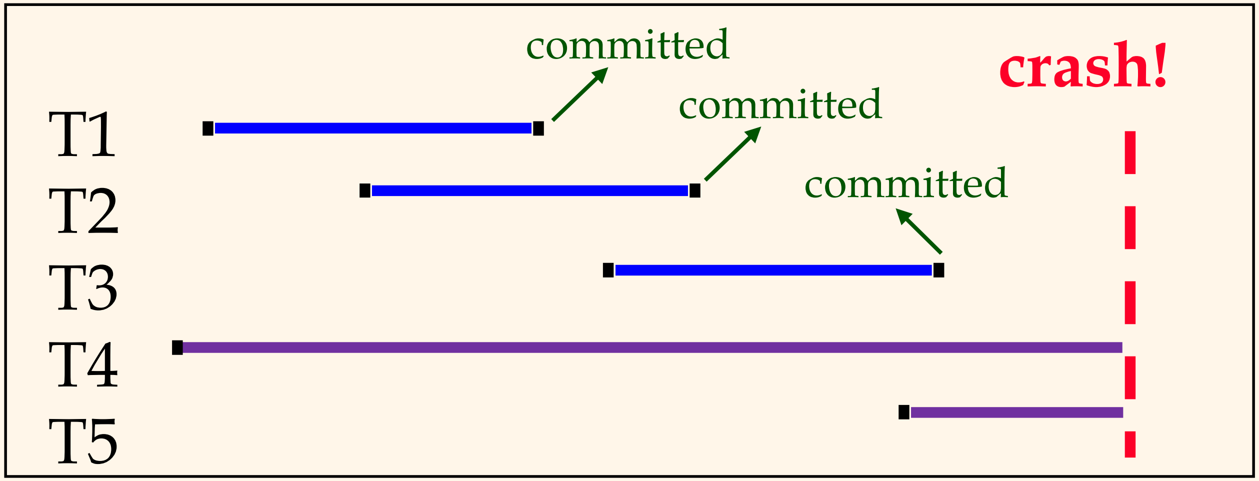
Failures Classification

- A transaction may need to be aborted for many reasons:
  - Manual cancellation; (human errors)
  - Violation of user constraints;
  - Software glitches; (bugs in software)
  - Hardware problems; (broken storage)
  - Natural disasters; (e.g., earthquake, flooding,…)
  - etc.…



Crash Example

- Desired behavior after system restarts:
  - T1, T2 and T3 should be durable.
  - T4 and T5 should be aborted and re-executed.



Challenge

- The database system does not know *which line was executing* just before the crash.
  - Only knows the final values (after the crash) for those objects in the database system
  - **Example** (Initial values: A = \$1100 and B = \$2000):

Transaction T

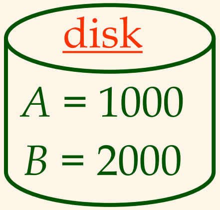
(Transfer *\$100* from A to B in the same bank)

1. **read**(A)
2. $A = A - 100$
3. **write**(A)
4. **read**(B)
5. $B = B + 100$

Crash! - - - - - - - - - - - - - - -

6. **write**(B)
7. **read**(B)
8. $B = B - 100$
9. **write**(B)

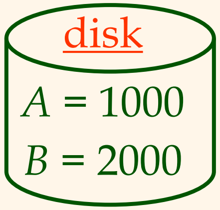
(Transfer *\$100* to C in other bank)



Transaction T

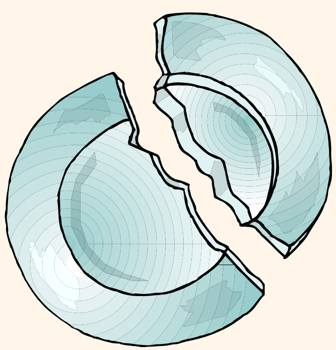
1. **read**(A)
2. $A = A - 100$
3. **write**(A)
4. **read**(B)
5. $B = B + 100$
6. **write**(B)
7. **read**(B)
8. $B = B - 100$
9. **write**(B)

Crash! - - - - - - - - - - - - - - -



High-Level Solution

- During normal transaction processing, prepare the necessary information for recovery
  - Store log records (e.g., more information)
- After a failure/crash, take action to recover the database content.
  - Recover the transactions using these log records after the system crashes



Outline

- Log-based recovery
  - Deferred-Modification recovery method
  - Immediate-Modification recovery method

Deferred-Modification Recovery Method

- A log consists of log records which are stored in the disk, and created as follows:
  - Before a transaction T starts, it writes a <T **start**> record.
  - Before T executes **write**(X), a log record <T, X, V_{new}> is created.
    - V_{new} is the new value of X.
  - Before T finishes, it creates a log record <T **commit**>.

Log Example

- The initial values of A and B are 1000 and 2000, respectively.

<u>Transaction *T*</u>

read(A)

$A = A - 50$

write(A)

read(B)

$B = B + 50$

write(B)

(can omit) **commit**

<u>Log records</u>

<*T* **start**>

<*T*, *A*, 950>

<*T*, *B*, 2050>

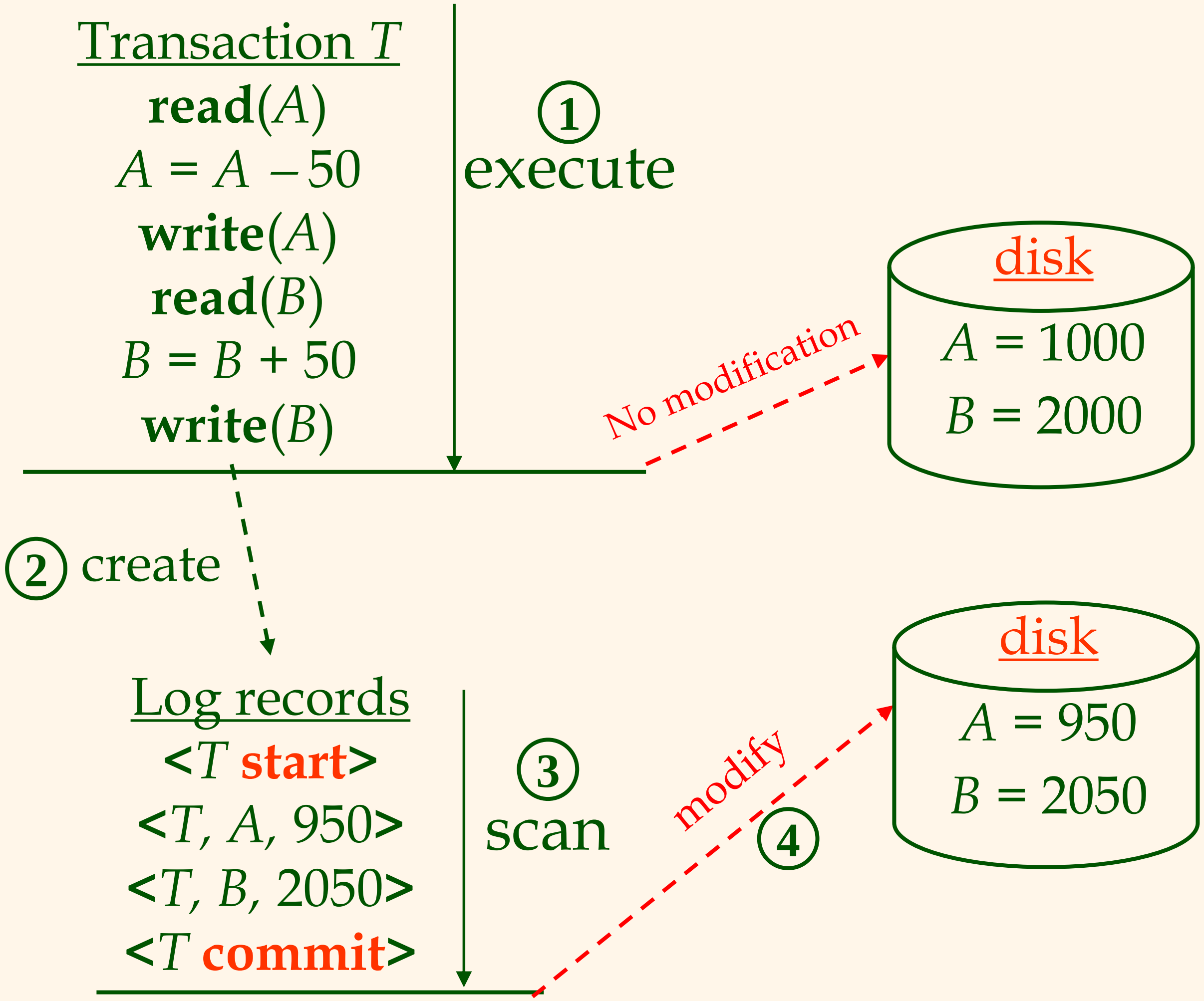
<*T* **commit**>

General Idea

- Only create the *log records* but *do not modify* the values of data objects on the disk immediately.

- After obtaining the log, the database system scans the *log records* again and modifies the values of corresponding data objects.

Example


Transaction T
read(A)
$A = A - 50$
write(A)
read(B)
$B = B + 50$
write(B)
1 execute
disk
$A = 1000$
$B = 2000$
No modification
2 create
Log records
<T **start**>
<T, A, 950>
<T, B, 2050>
<T **commit**>
3 scan
modify
4
disk
$A = 950$
$B = 2050$

Crash before Commit

- After the system recovers from crash, it *ignores* those transactions without the log record <*T* commit> and executes those transactions again from scratch. (atomicity ☺)

Transaction *T*		create	Log records
read(*A*)	execute	⇨	<*T* **start**>
$A = A - 50$			<*T*, *A*, 950>
write(*A*)			
read(*B*)			
- - - - - - - - - -	Crash!		
$B = B + 50$			
write(*B*)			

No <*T* commit>. So this is an incomplete transaction. Do not scan log records but re-execute the transition *T* from scratch.

Crash after Commit

❖ After the system recovers from crash and it has created the **complete** log records, it needs to scan this log again to recover the values of the objects. (durability ☺)

Transaction *T*			Log records
read(A)		create ⇨	<*T* **start**>
$A = A - 50$			<*T*, *A*, 950>
write(A)	execute		<*T*, *B*, 2050>
read(B)			<*T* **commit**>
$B = B + 50$			
write(B)			

Crash!

<*T* commit> exists. So this is an complete transaction. Scan log records and modify the values of *A* and *B* on the disk

Example (multiple transactions)

- The initial values of A, B, C are 1000, 2000, and 700 respectively. Execute T_0 first and then T_1

<u>Transaction T_0</u>	<u>Transaction T_1</u>
read(A)	**read**(C)
$A = A - 50$	$C = C - 100$
write(A)	**write**(C)
read(B)	
$B = B + 50$	
write(B)	

create ⇒

<u>Log records</u>

< T_0 **start**>
< T_0, A, 950>
< T_0, B, 2050>
< T_0 **commit**>
< T_1 **start**>
< T_1, C, 600>
< T_1 **commit**>

scan

<u>disk</u>

$A = 1000$
$B = 2000$
C=700

<u>disk</u>

$A = 950$
$B = 2050$
C=600

Example (multiple transaction)

Transaction T_1

read(A)
$A = A - 50$
write(A)
read(B)
$B = B + 50$
write(B)

Transaction T_2

read(C)
$C = C - 100$
write(C)

Schedule S

T_1 : **read**(A)
T_1 : $A = A - 50$
T_2 : **read**(C)
T_2 : $C = C - 100$
T_1 : **write**(A)
T_2 : **write**(C)
T_2 : commit
T_1 : **read**(B)
T_1 : $B = B + 50$
T_1 : **write**(B)
T_1 : commit

Example (multiple transactions)

Initial values: A=1000, B = 600, and C = 1000

Schedule S

T_1 : **read**(A)
T_1 : $A = A - 50$
T_2 : **read**(C)
T_2 : $C = C - 100$
T_1 : **write**(A)
T_2 : **write**(C)
T_2 : **commit**
T_1 : **read**(B)
T_1 : $B = B + 50$
T_1 : **write**(B)
T_1 : **commit**

create ⇒

Log records for S

<T_1 **start**>
<T_2 **start**>
<T_1, A, 950>
<T_2, C, 900>
<T_2 **commit**>
<T_1, B, 650>
<T_1 **commit**>

Question 1

- Suppose that the system crashes when it executes the below schedule using the deferred-modification recovery method. Describes how this system recovers if it finds the following logs (a)-(b), respectively.

Schedule

<T_1 **start**>
<T_1, V, 200>
<T_2 **start**>
<T_2, L, 300>
<T_2, D, 300>
<T_2 **commit**>
<T_1, B, 250>
<T_1 **commit**>

(a)
<T_1 **start**>
<T_1, V, 200>
<T_2 **start**>
<T_2, L, 300>

(b)
<T_1 **start**>
<T_1, V, 200>
<T_2 **start**>
<T_2, L, 300>
<T_2, D, 300>
<T_2 **commit**>

(c)
<T_1 **start**>
<T_1, V, 200>
<T_2 **start**>
<T_2, L, 300>
<T_2, D, 300>
<T_2 **commit**>
<T_1, B, 250>
<T_1 **commit**>

Weakness of Deferred-Modification Recovery Method

- Can be slow, i.e., low throughput (#transactions/sec)
  - Update the values of objects only when the database system scans the log records
  - Need to scan log records for *all* transactions

- Hard to determine when to scan the log records
  - If do not update the values of objects on the disk in time, subsequent transactions might be affected

Immediate-Modification Recovery Method

- A log consists of log records which are stored in the disk, and created as follows:
  - Before a transaction *T* starts, it writes a <*T* **start**> record.
  - Before *T* executes **write**(*X*), a log record <*T*, *X*, V_{old}> is created.
    - V_{old} is the old value of *X*.
  - Before *T* finishes, it creates a log record <*T* **commit**>.

Log Example

- The initial values of A and B as 1000 and 2000, respectively.

Transaction T	Log records
	<T **start**>
read(A)	<T, A, 1000>
$A = A - 50$	<T, B, 2000>
write(A)	<T **commit**>
read(B)	
$B = B + 50$	
write(B)	
(can omit) **commit**	

General Idea

- Directly modifies the values of data objects (e.g., A, B) *on the disk* when the database system executes the transaction.

Initially A=1000, B = 2000

<u>Transaction T</u>

read(A)
$A = A - 50$
write(A)
read(B)
$B = B + 50$
write(B)

execute

create

<u>Log records</u>

<T **start>**
<T, A, 1000**>**
<T, B, 2000**>**
<T **commit>**

modify

<u>disk</u>
$A = 950$
$B = 2050$

Crash before Commit

- *Undo all modifications* based on the log records for those transactions that have not committed when the database system recovers from crash. (atomicity ☺)

Transaction T

read(A)
$A = A - 50$
write(A)
read(B)
$B = B + 50$

Crash!

write(B)

execute

modify

create

Log records
<T **start**>
<T, A, 1000>

No <T commit>. Need to undo the modifications of A.

disk
$A = 950$
$B = 2000$

Crash before Commit

- *Undo all modifications* based on the log records for those transactions that have not committed when the database system recovers from crash. (atomicity ☺)

Crash after Commit

- Do not need to undo the modifications for those transactions that have committed. (durability ☺)

Transaction T		Log records
read(A)	create ⇒	<*T* **start**>
$A = A - 50$		<*T*, *A*, 1000>
write(A)	execute	<*T*, *B*, 2000>
read(B)		<*T* **commit**>
$B = B + 50$		
write(B)		

Crash!

modify

disk

$A = 950$

$B = 2050$

<*T* commit> exists.
Do nothing.

Solution to Question 1

a) The system scans the log records and finds that both T_1 and T_2 have not committed. It simply re-executed transactions T_1 and T_2 again.

b) The system scans the log records and finds that T_1 has not committed and T_2 has committed. As such, it modifies L and D to be 300 after it scans the log records. The system re-executes T_1 again.

c) The system scans the log records. Since both T1 and T2 have committed, the system modifies V, L, D, and B to be 200, 300, 300, and 250, respectively.