(Undo-)Logging for your database

Overview of this video

We will try to handle failures using a simple kind of logs, specifically using something called an undo-log

Logging in DBMS

Idea: write important activities to a log so that a desired database state can be recovered later

Examples of important activities:

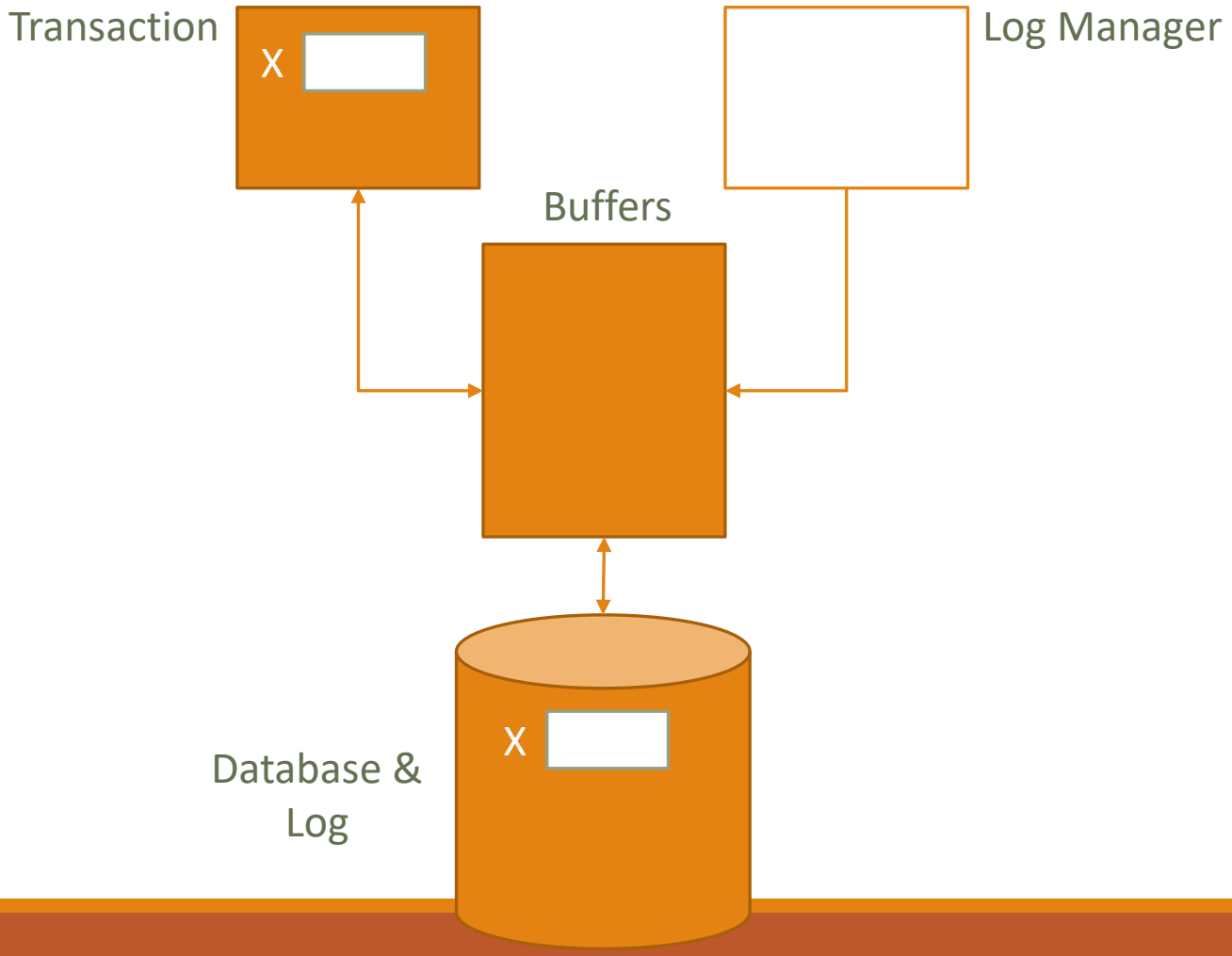
- Starts of transactions, commits, aborts
- Modification of database items

Should work *even in case of system failures*!

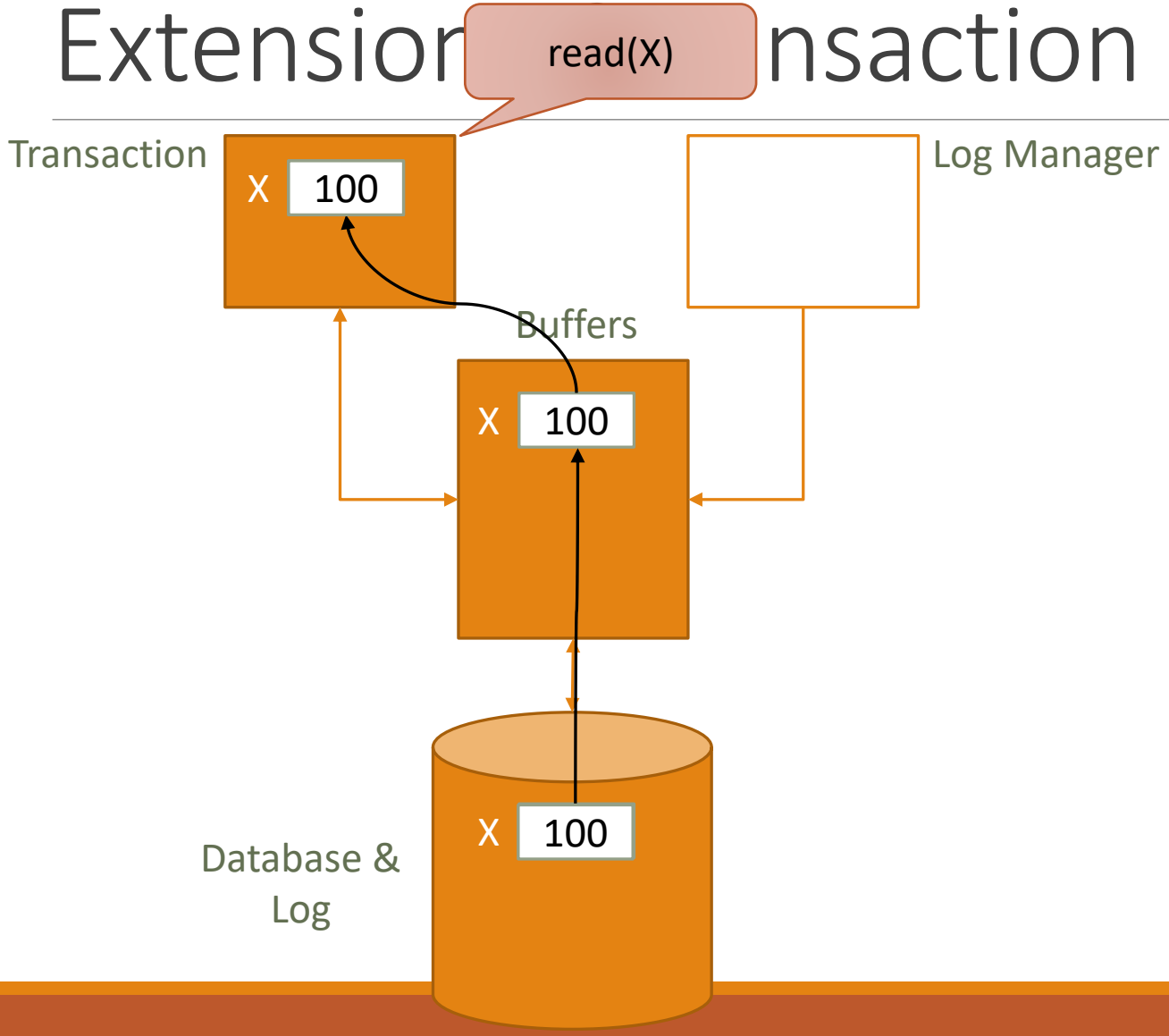
Techniques:

- Undo logging (for maintaining *Atomicity*)
- Redo logging (for maintaining *Durability*)
- Combinations thereof (specifically Undo/Redo for doing both Atomicity and Durability)

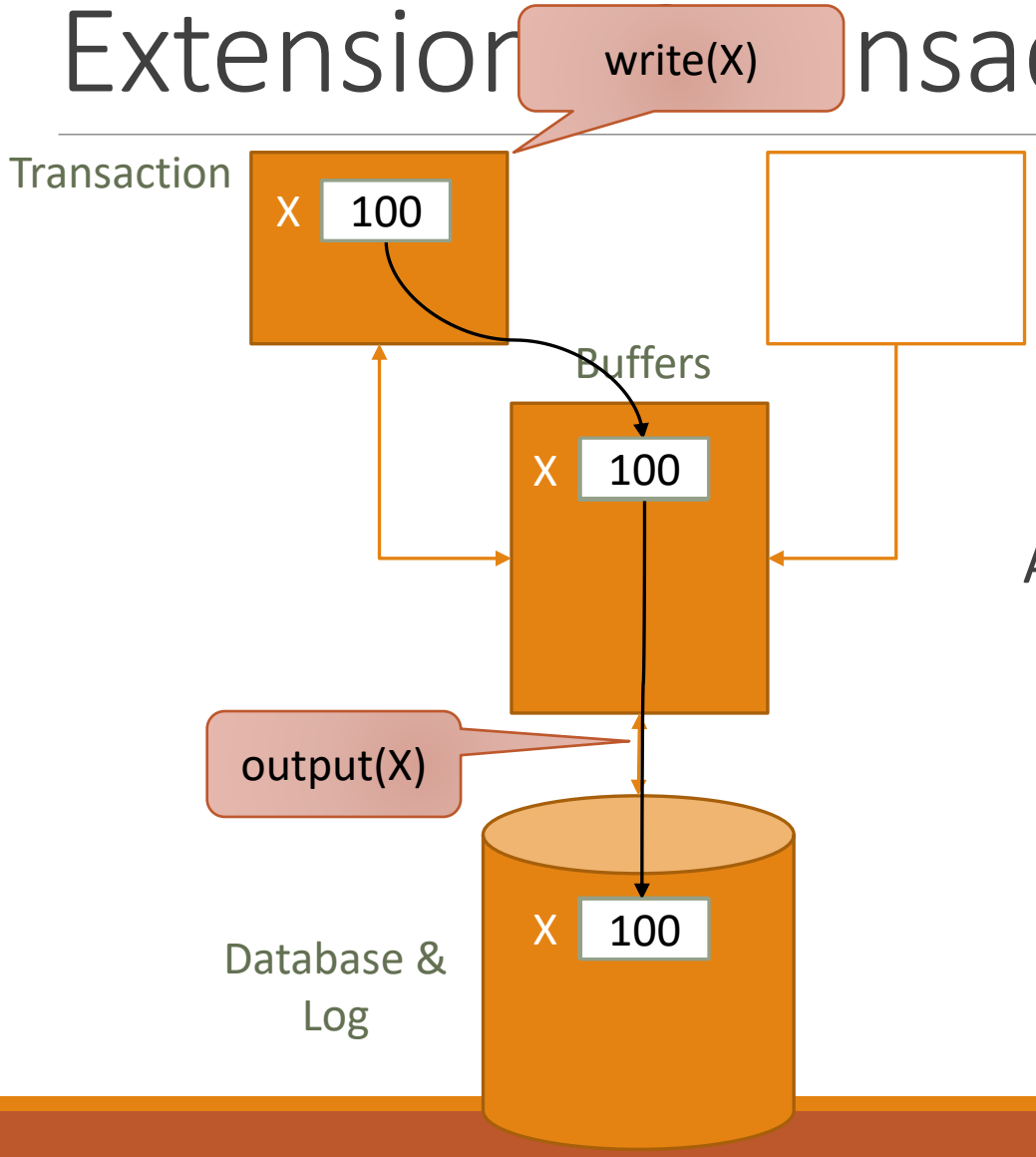
Extension of Transaction Syntax



Extensions to Transaction Syntax



Extension Transaction Syntax



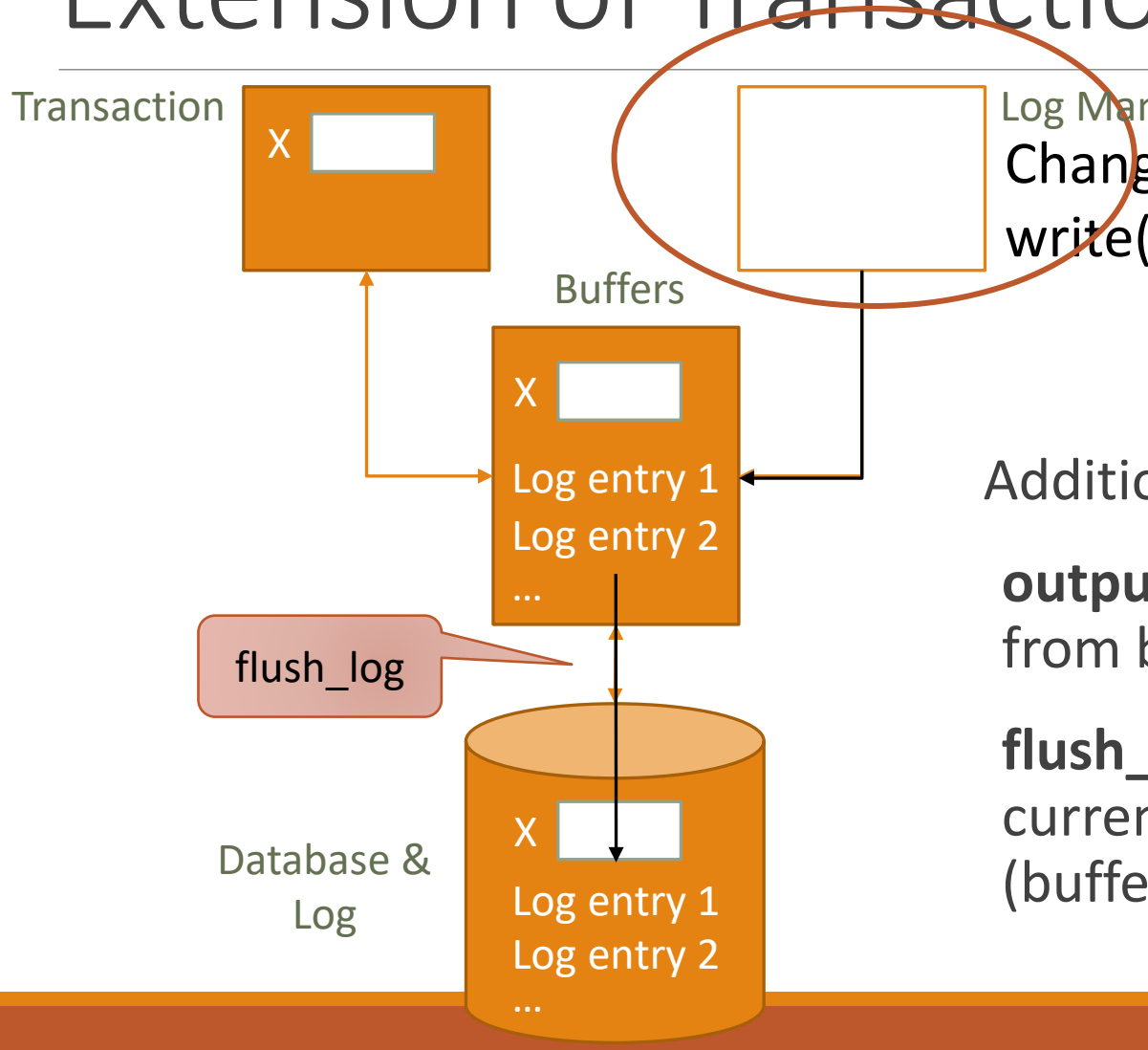
Changes:

write(X) writes **only** to buffer

Additional operations:

output(X): copy database item X from buffer to database

Extension of Transaction Syntax



Changes:

write(X) writes **only** to buffer

Additional operations:

output(X): copy database item X from buffer to database

flush_log: write log entries that are currently residing in main memory (buffer) to log

Undo Logging

Logs activities with the goal of restoring ("undoing") a previous database state.

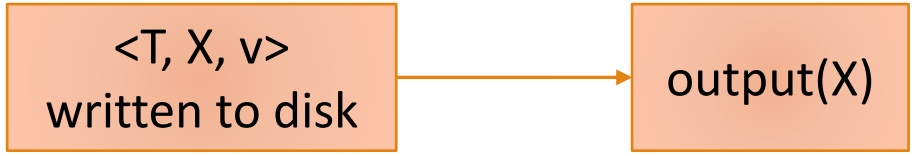
Log records (or log entries):

- **<START T>**: Transaction T has started.
- **<COMMIT T>**: Transaction T has committed.
- **<ABORT T>**: Transaction T was aborted.
- **<T, X, v>**: Transaction T has updated the value of database item X, and the old value of X was v.
  - Response to **write(X)**
  - If this entry occurs in the log, then the new value of X might not have been written to the database yet.

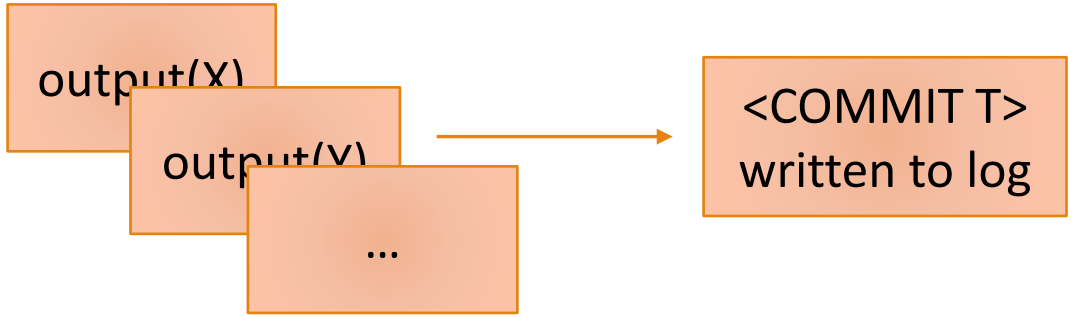
Slightly different records for redo- and undo/redo logging (later video…)

Undo Logging: Procedure

1. If transaction T updates database item X and
   the old value was v, then <T, X, v> must be written to the log on disk **before** X is written to disk.



2. If transaction T commits, then <COMMIT T> must be written to disk **as soon as** all database elements changed by T have been written to disk



Example

Transaction
read(X)
X := X*2
write(X)
read(Y)
Y := Y*2
write(Y)
flush_log
output(X)
output(Y)
flush_log

Writes all log entries for updates to disk

Writes all updates to disk

Writes the <COMMIT T> record to disk

Example

Time	Transaction	Local X	Local Y	Buffer X	Buffer Y	Database X	Database Y	Log record written
0						1	10	<START T>
1	read(X)	1		1		1	10	
2	X := X*2	2		1		1	10	
3	write(X)	2		2		1	10	<T, X, 1>
4	read(Y)	2	10	2	10	1	10	
5	Y := Y*2	2	20	2	10	1	10	
6	write(Y)	2	20	2	20	1	10	<T, Y, 10>
7	**flush_log**	2	20	2	20	1	10	
8	**output(X)**	2	20	2	20	2	10	
9	**output(Y)**	2	20	2	20	2	20	
10		2	20	2	20	2	20	<COMMIT T>
11	**flush_log**	2	20	2	20	2	20	

What if a system failure occurs? (The Recovery Manager)

Scenario 1

<COMMIT T> occurs in the log on disk

Time	Transaction	X	Y	X	Y	X	Y	
0						1	10	<START T>
1	read(X)	1		1		1	10	
2	X := X*2	2		1		1	10	
3	write(X)	2		2		1	10	<T, X, 1>
4	read(Y)	2	10	2	10	1	10	
5	Y := Y*2	2	20	2	10	1	10	
6	write(Y)	2	20	2	20	1	10	<T, Y, 10>
7	**flush_log**	2	20	2	20	1	10	
8	**output(X)**	2	20	2	20	2	10	
9	**output(Y)**	2	20	2	20	2	20	
10		2	20	2	20	2	20	<COMMIT T>
11	**flush_log**	2	20	2	20	2	20	

T has committed successfully (no recovery needed)

Error

Scenario 2

<COMMIT T> **does not occur** in the log on disk

Time	Transaction	X	Y	X	Y	X	Y	
0								
1	read(X)	1		1				
2	X := X*2	2		1				
3	write(X)	2		2		1	10	<T, X, 1>
4	read(Y)	2	10	2	10	1	10	
5	Y := Y*2	2	20	2	10	1	10	
6	write(Y)	2	20	2	20	1	10	<T, Y, 10>
7	**flush_log**	2	20	2	20	1	10	
8	**output(X)**	2	20	2	20	2	10	
9	**output(Y)**	2	20	2	20	2	20	
10		2	20	2	20	2	20	<COMMIT T>
11	**flush_log**	2	20	2	20	2	20	

Error

Must undo all updates to database items that were written to disk.

For each log record <T, X, v> on disk, replace X on disk by v.

Scenario 3

<COMMIT T> **does not occur** in the log on disk

What to do in this case? Nothing!

Time	Transaction	X	Y	X	Y	X	Y	Lo
0						1	10	<START T>
1	read(X)	1		1		1	10	
2	X := X*2	2		1		1	10	
3	write(X)	2		2		1	10	<T, X, 1>
4	read(Y)	2	10	2	10	1	10	
5	Y := Y*2	2	20	2	10	1	10	
6	write(Y)	2	20	2	20	1	10	<T, Y, 10>
7	**flush_log**	2	20	2	20	1	10	
8	**output(X)**	2	20	2	20	2	10	
9	**output(Y)**	2	20	2	20	2	20	
10		2	20	2	20	2	20	<COMMIT T>
11	**flush_log**	2	20	2	20	2	20	

Error

Recovery With Undo Logs
(Simple Variant)

If an error occurs, the recovery manager restores the last consistent database state

Traverses the undo log backwards

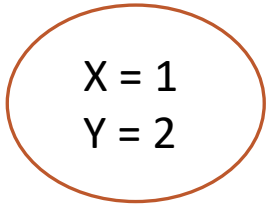
If the current entry is…

- **<COMMIT T>**: remember that T was committed successfully
- **<ABORT T>**: remember that T was aborted
- **<T, X, v>**: if T has not finished successfully (no COMMIT, no ABORT), change the value of X on disk to v

Write **<ABORT T>** for each *uncommitted* transaction T that was not previously aborted & call flush_log

Example of Undo Logging

Time	Transaction T_1	Transaction T_2
1	read(X)	
2	X := X * 2	
3	write(X)	
4		read(X)
5	read(Y)	
6		X := X * 3
7		write(X)
8	Y := X + Y	
9	write(Y)	



How does undo logging work on this schedule?

- Which **log entries** are written to buffer/disk & when?
- Which **other operations** must be executed & when?

Time	Transaction T_1	Transaction T_2	Log (buffer)	Log (disk)
0			<START T_1>	
1	read(X)			
2	X := X * 2			
3	write(X)		<T_1, X, 1>	
4			<START T_2>	
5		read(X)		
6	read(Y)			
7		X := X * 3		
8		write(X)	<T_2, X, 2>	
9	Y := X + Y			
10	write(Y)		<T_1, Y, 2>	
11	**flush_log**			
12	**output(X)**			
13	**output(Y)**			
14			<COMMIT T_1>	
15	**flush_log**			
16		**flush_log**		
17		**output(X)**		
18			<COMMIT T_2>	
19		**flush_log**		

X = 1
Y = 2

What if a transaction aborts?

If a Transaction Aborts…

Use the undo log to undo all changes made by the transaction.

- Similar to recovery with undo logs
- But focuses on a single transaction

Procedure:

- Assume T aborts
- Traverse the undo log from the last to the first item
- If we see **<T, X, v>**, change the value of X on disk back to v.

Summary

Aborted transactions & system failures can be dealt with using careful

Undo logging

- Maintains Atomicity
- Logs old value for each updated database item
- Recovery manager: use this information to restore the last consistent database state