COMP207
Database Development

Lecture 8

Transaction Management:
Dealing With Transaction Aborts &
System Failures

No lecture later today!

- Lecture today 17-18 is canceled
  - People can go to prof Paul Spirakis Inaugural lecture (if they want). That is 17:30-20 in ELT

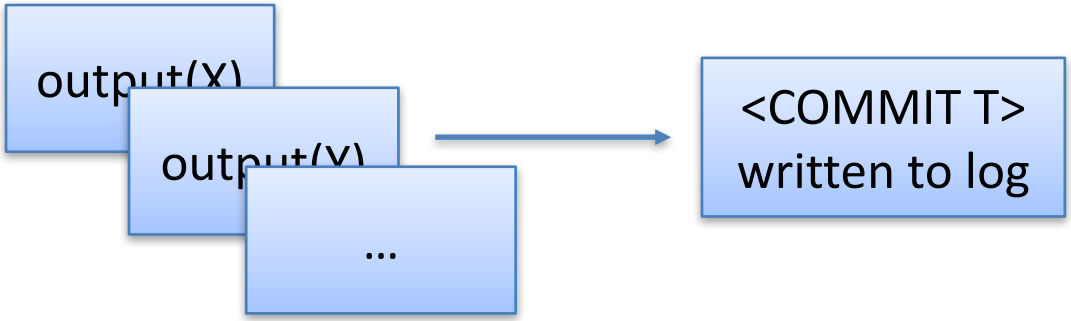
Review of Undo Logging

- 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_item(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 logging (later…)

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



Review of Undo Logging

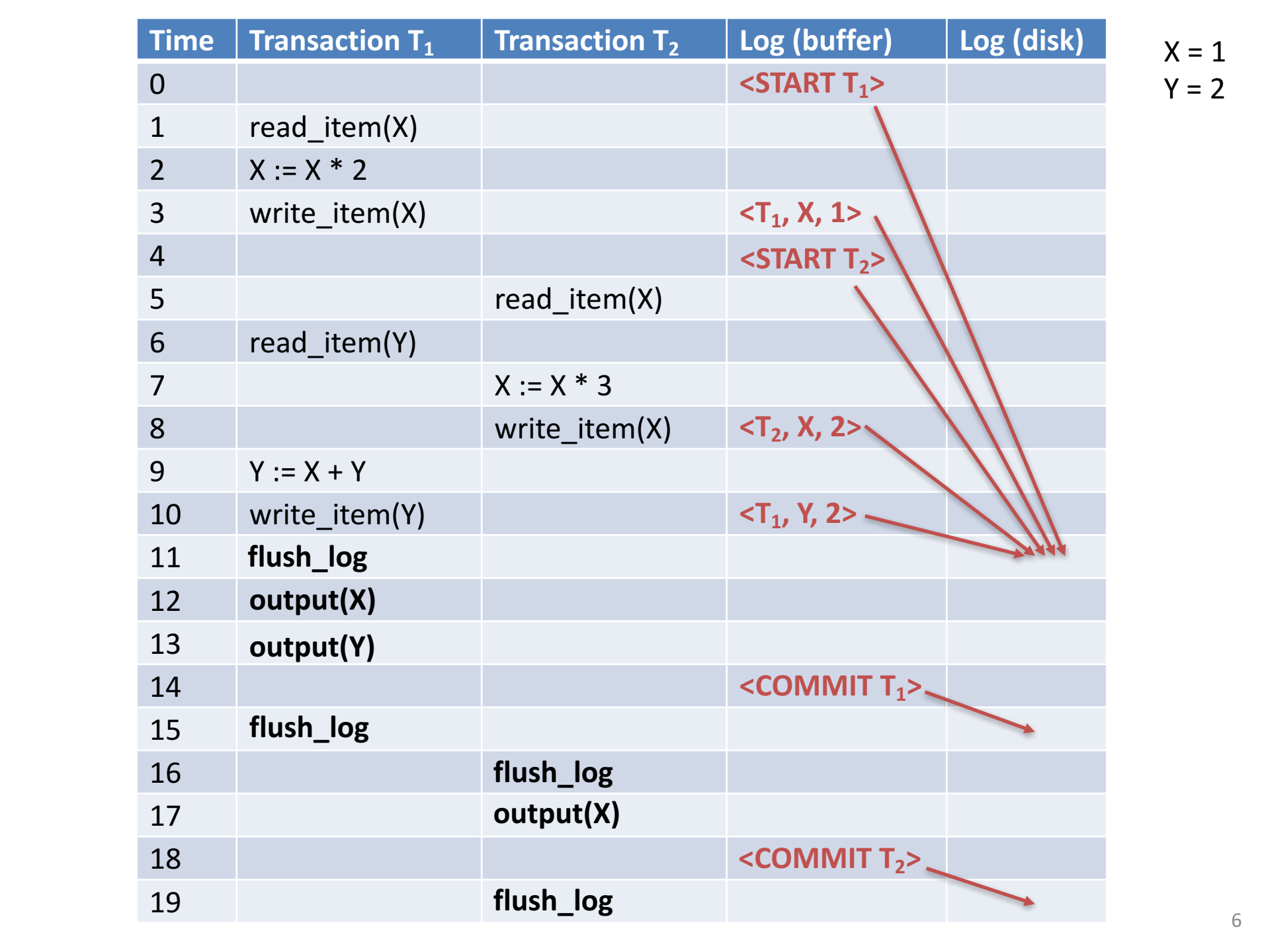
5 min

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

X = 1
Y = 2

- 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_item(X)			
2	X := X * 2			
3	write_item(X)		<T_1, X, 1>	
4			<START T_2>	
5		read_item(X)		
6	read_item(Y)			
7		X := X * 3		
8		write_item(X)	<T_2, X, 2>	
9	Y := X + Y			
10	write_item(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.

Other Logging Variants

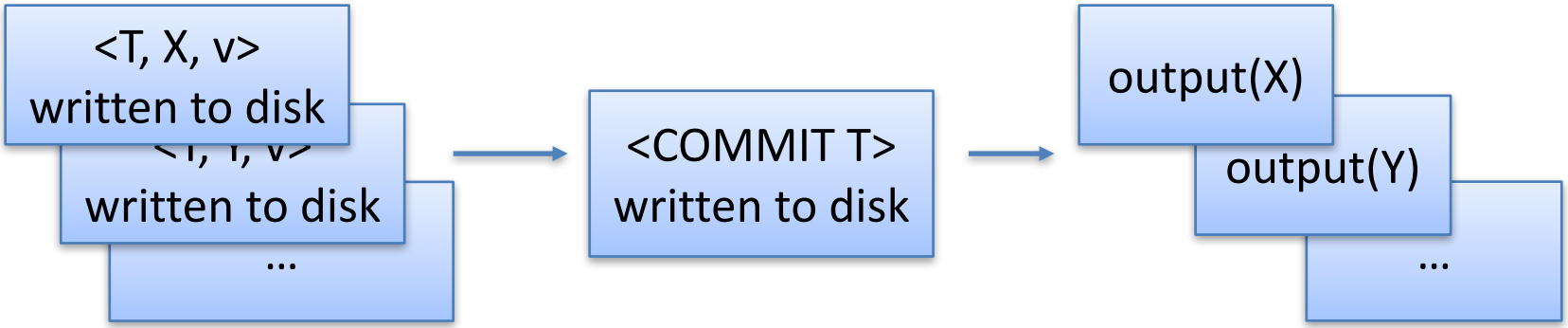
Redo Logging
Undo/Redo Logging

Redo Logging

- Logs activities with the goal of *restoring* committed transactions (ignores incomplete transactions).
- Log records:
  - Same as before, but…
  - New meaning of **<T, X, v>**: "Transaction T has updated the value of database item X & the new value of X is v."
    - Direct response to **write_item(X)**
    - Haven't changed X on disk yet!
- Have to modify the logging procedure…

Redo Logging: Procedure

1. T first writes all log records for all updates to disk
2. T writes <COMMIT T> to the log on disk
3. T writes all updates to disk

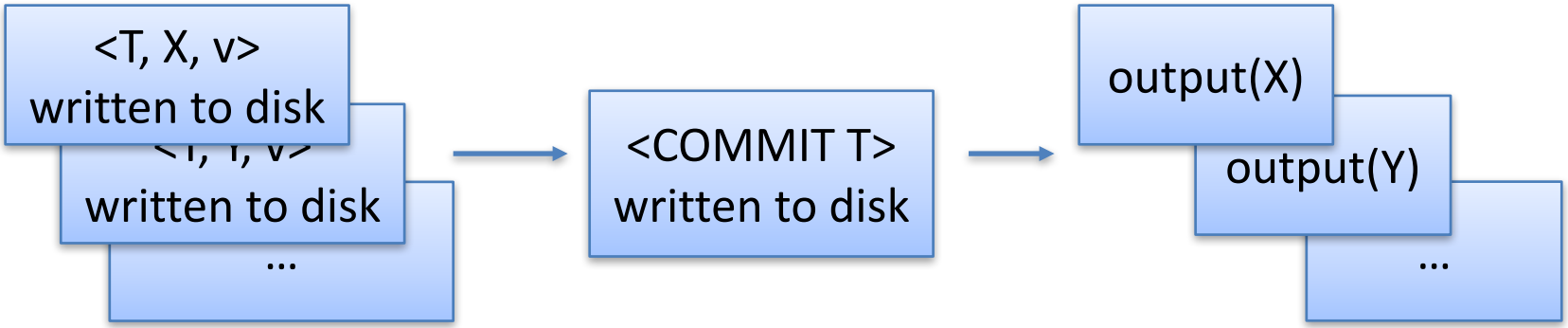


Example

		Local		Buffer		Database			
Time	Transaction	X	Y	X	Y	X	Y	Log (buffer)	Log (disk)
0						1	10	<START T>	
1	read_item(X)	1		1		1	10		
2	X := X*2	2		1		1	10		
3	write_item(X)	2		2		1	10	<T, X, 2>	
4	read_item(Y)	2	10	2	10	1	10		
5	Y := Y*2	2	20	2	10	1	10		
6	write_item(Y)	2	20	2	20	1	10	<T, Y, 20>	
7								<COMMIT T>	
8	**flush_log**								
9	**output(X)**	2	20	2	20	2	10		
10	**output(Y)**	2	20	2	20	2	20		

Redo Logging: Procedure

1. T first writes all log records for all updates to disk
2. T writes <COMMIT T> to the log on disk
3. T writes all updates to disk



Fundamental property of redo logs:

- <COMMIT T> occurs in log → log contains complete information on T
- <COMMIT T> doesn't occur in log → T hasn't written anything to disk

Recovery With Redo Logs

- Essentially: reverse of undo logging
- Procedure:
  - Identify all the transactions wi
  - Traverse the log from first to t
  - If we see **<T, X, v>** and T has a COMMIT log record, then change the value of X on disk to v.
  - For each *incomplete* transaction T, write **<ABORT T>** into the log on disk.

What could go wrong if we wrote to disk **before** flushing <commit T>?

Exercise with Redo Logging

5 min

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

X = 1
Y = 2

- How does redo 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_item(X)			
2	X := X * 2			
3	write_item(X)		<T_1, X, 2>	
4			<START T_2>	
5		read_item(X)		
6	read_item(Y)			
7		X := X * 3		
8		write_item(X)	<T_2, X, 6>	
9	Y := X + Y			
10	write_item(Y)		<T_1, Y, 4>	
11			<COMMIT T_1>	
12	**flush_log**			
13	**output(X)**			
14	**output(Y)**			
15			<COMMIT T_2>	
16		**flush_log**		
17		**output(X)**		
18				
19				

X = 1
Y = 2

Summary

- Aborted transactions & system failures can be dealt with using careful logging & restoring data using logs
- Undo logging:
  - Maintains atomicity
  - Supports undoing aborted transactions & recovers last consistent database state in case of system failure
- Other variants:
  - Redo logging: for Durability
  - Undo/Redo: Best variant
- Checkpoints speed up recovery
- Dirty reads my cause certain problems → problem and solutions tomorrow