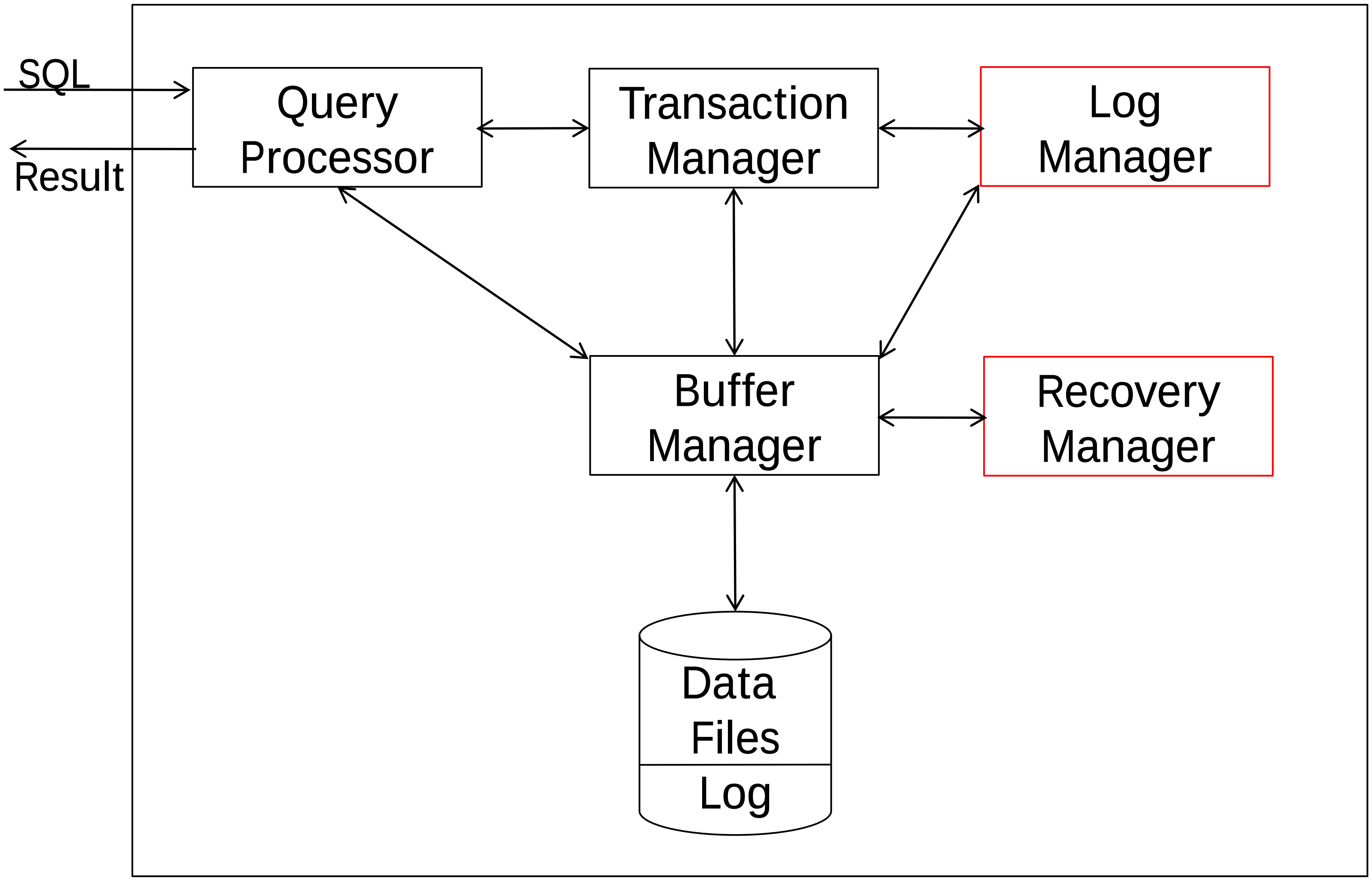
CSCD43: Database Systems Technology

Lecture 11

Wael Aboulsaadat

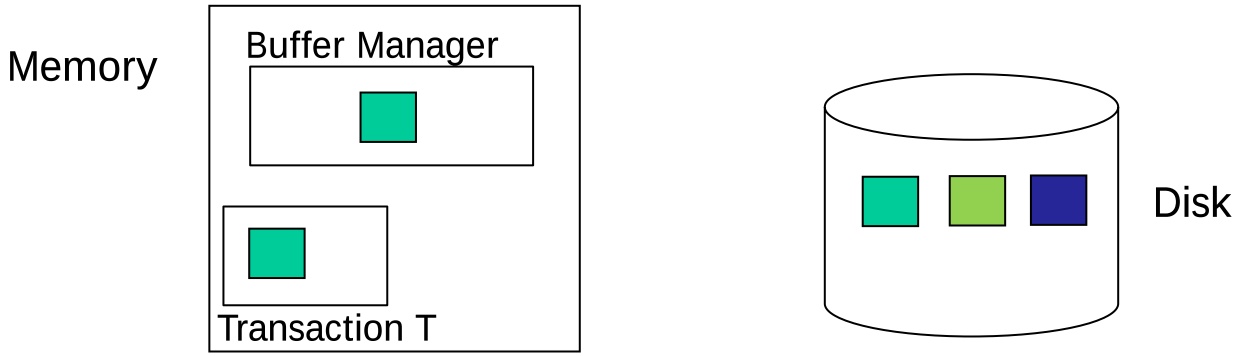
Acknowledgment: these slides are based on Prof. Garcia-Molina & Prof. Ullman slides accompanying the course's textbook.

DBMS Architecture



Second order of business:

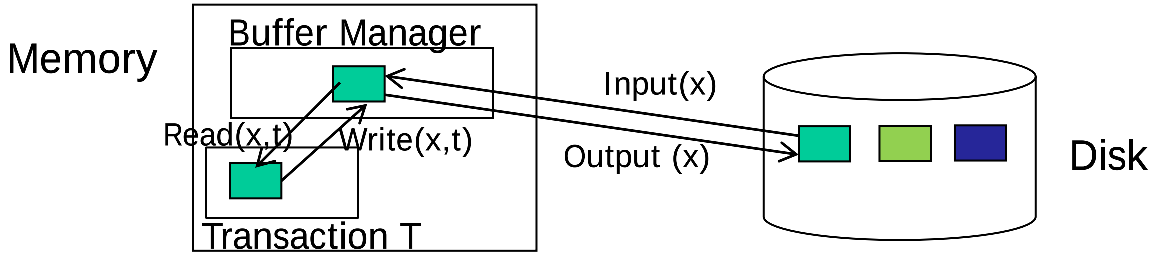
Storage hierarchy



Operations:

- Input (x): block containing x → memory
- Output (x): block containing x → disk

- Read (x,t): do input(x) if necessary
  t ← value of x in block
- Write (x,t): do input(x) if necessary
  value of x in block ← t



Log Commands:

<Start T>

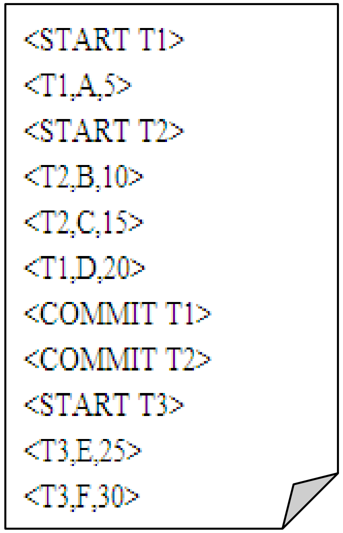
log the start of a transaction

<T1, X, value>

log that T1 (transaction identifier) modified X (database record) affecting value (value)

<COMMIT T>

log the completion of a transaction



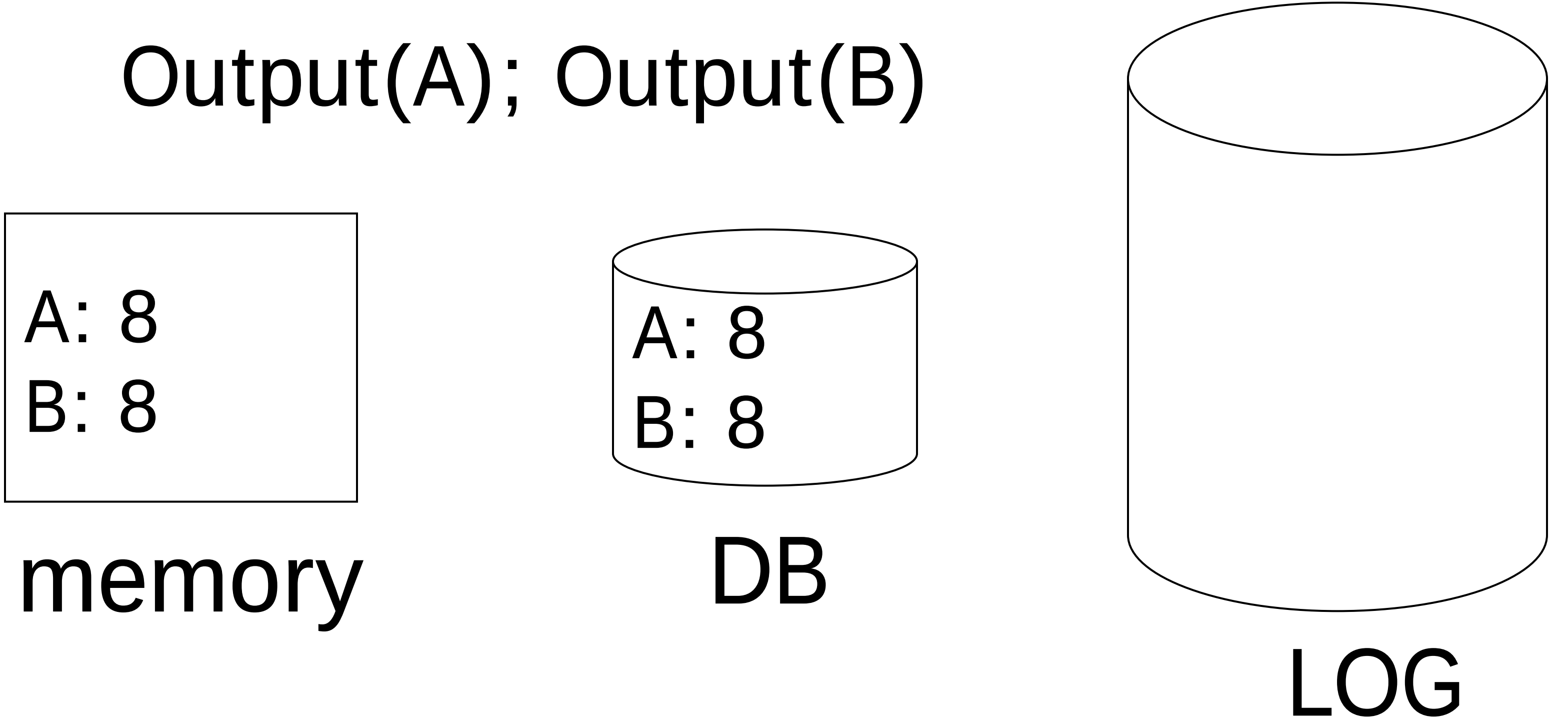
```
<START T1>
<T1,A,5>
<START T2>
<T2,B,10>
<T2,C,15>
<T1,D,20>
<COMMIT T1>
<COMMIT T2>
<START T3>
<T3,E,25>
<T3,F,30>
```

Redo logging (deferred modification)

T_1: Read(A,t); t← t×2; write (A,t);

Read(B,t); t←t×2; write (B,t);

Output(A); Output(B)

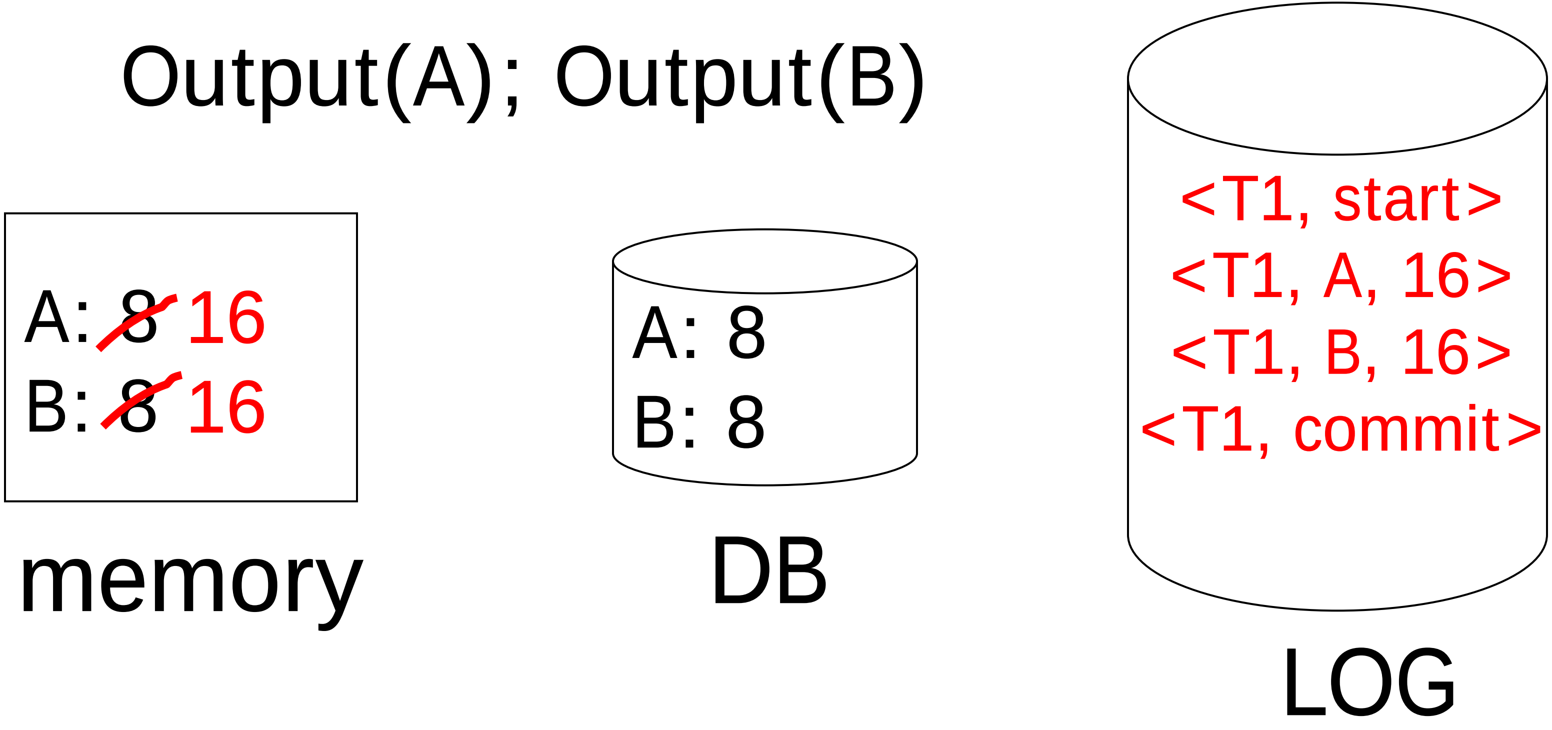


Redo logging (deferred modification)

T1: Read(A,t); t← t×2; write (A,t);

Read(B,t); t←t×2; write (B,t);

Output(A); Output(B)



A: ~~8~~ 16

B: ~~8~~ 16

memory

A: 8

B: 8

DB

<T1, start>

<T1, A, 16>

<T1, B, 16>

<T1, commit>

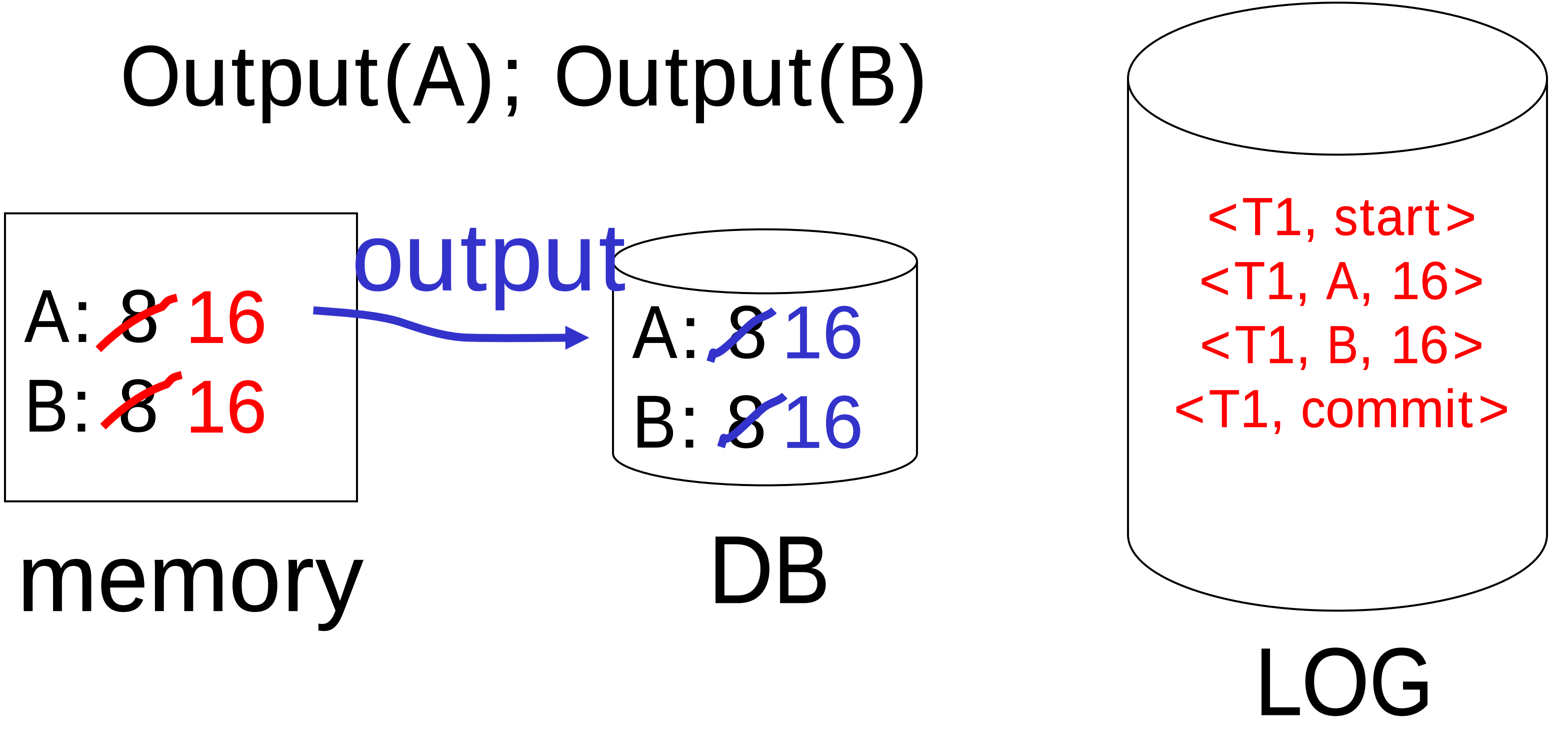
LOG

Redo logging (deferred modification)

T1: Read(A,t); t← t×2; write (A,t);

Read(B,t); t←t×2; write (B,t);

Output(A); Output(B)



A: ~~8~~ 16

B: ~~8~~ 16

memory

output

A: ~~8~~ 16

B: ~~8~~ 16

DB

<T1, start>

<T1, A, 16>

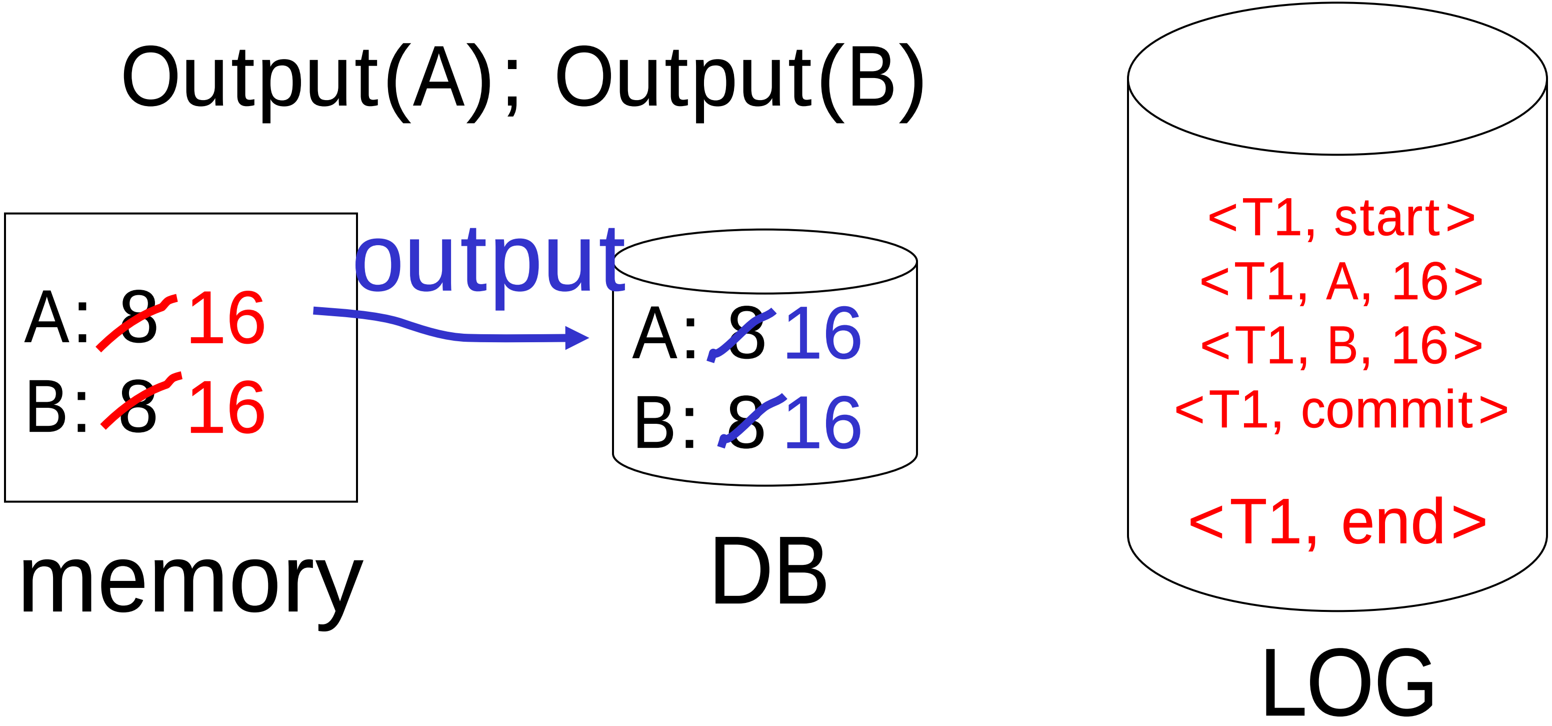
<T1, B, 16>

<T1, commit>

LOG

Redo logging (deferred modification)

T_1: Read(A,t); t ← t×2; write (A,t);
Read(B,t); t ← t×2; write (B,t);
Output(A); Output(B)



A: ~~8~~ 16
B: ~~8~~ 16

memory

output

A: ~~8~~ 16
B: ~~8~~ 16

DB

<T1, start>
<T1, A, 16>
<T1, B, 16>
<T1, commit>

<T1, end>

LOG

Redo Log Rule:

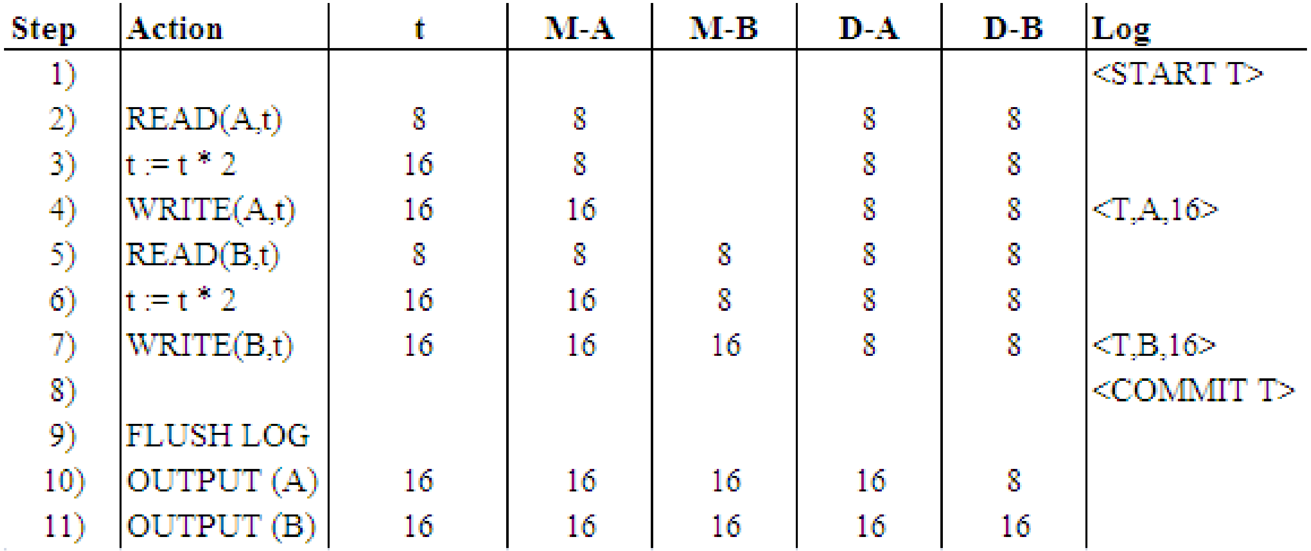
R_1:

Before modifying any database element X on disk, it is necessary that all log records pertaining to this modification of X, including both the update record <T,X,v> and the <COMMIT T> record, must appear on disk

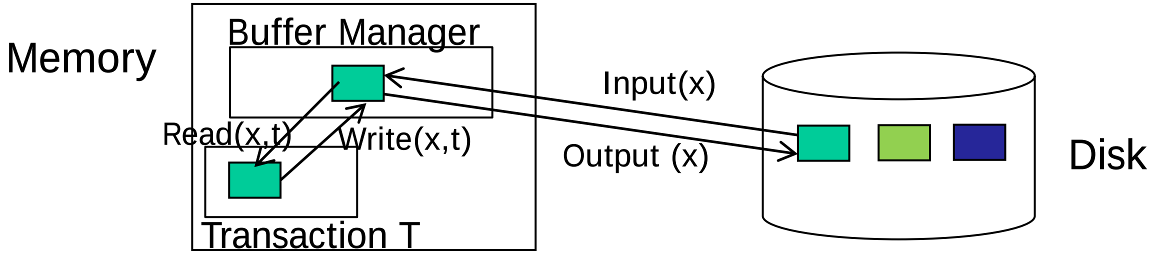
Redo Example:

A := A * 2;
B := B * 2;

R1: Before modifying any database element X on disk, it is necessary that all log records pertaining to this modification of X, including both the update record <T,X,v> and the <COMMIT T> record, must appear on disk

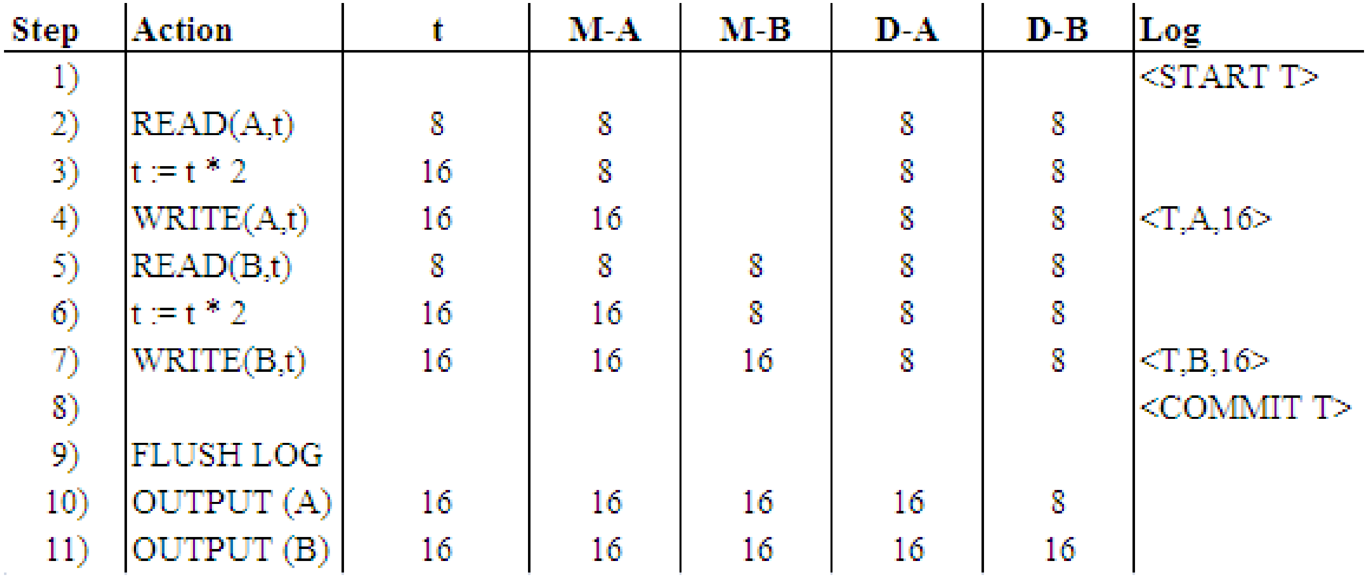


Step	Action	t	M-A	M-B	D-A	D-B	Log
1)							<START T>
2)	READ(A,t)	8	8		8	8	
3)	t := t * 2	16	8		8	8	
4)	WRITE(A,t)	16	16		8	8	<T,A,16>
5)	READ(B,t)	8	8	8	8	8	
6)	t := t * 2	16	16	8	8	8	
7)	WRITE(B,t)	16	16	16	8	8	<T,B,16>
8)							<COMMIT T>
9)	FLUSH LOG						
10)	OUTPUT (A)	16	16	16	16	8	
11)	OUTPUT (B)	16	16	16	16	16	



Redo Log: what if a crash happens?

A := A * 2;
B := B * 2;



Step	Action	t	M-A	M-B	D-A	D-B	Log
1)							<START T>
2)	READ(A,t)	8	8		8	8	
3)	t := t * 2	16	8		8	8	
4)	WRITE(A,t)	16	16		8	8	<T,A,16>
5)	READ(B,t)	8	8	8	8	8	
6)	t := t * 2	16	16	8	8	8	
7)	WRITE(B,t)	16	16	16	8	8	<T,B,16>
8)							<COMMIT T>
9)	FLUSH LOG						
10)	OUTPUT (A)	16	16	16	16	8	
11)	OUTPUT (B)	16	16	16	16	16	

Recovery rules: Redo logging

(1) Scan log from beginning. For each log record <T,X,v> encountered:

a. If T is not committed, do nothing
b. If T is committed, write value of v for database element X

(2) For each incomplete transaction T, write an <ABORT T> record to the log

Redo Log: what if a crash happens?

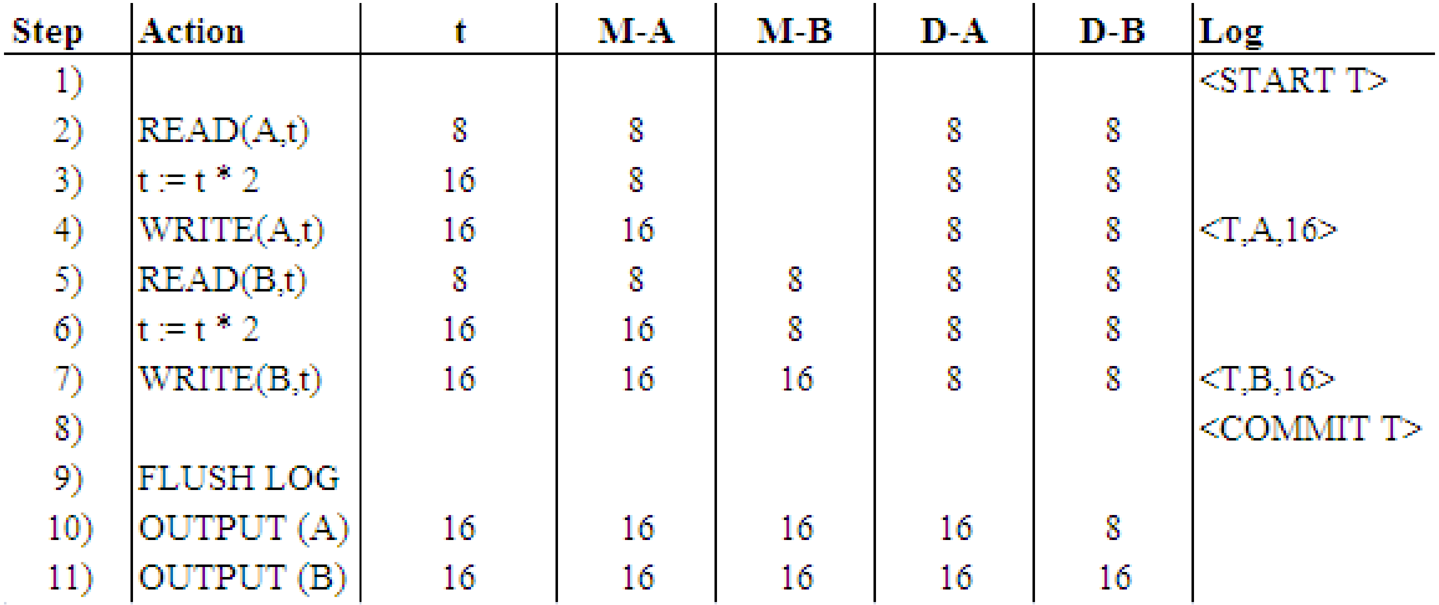
A := A * 2;
B := B * 2;

(1) Scan log from beginning. For each log record <T,X,v> encountered:

a. If T is not committed, do nothing

b. If T is committed, write value of v for database element X

(2) For each incomplete transaction T, write an <ABORT T> record to the log



Step	Action	t	M-A	M-B	D-A	D-B	Log
1)							<START T>
2)	READ(A,t)	8	8		8	8	
3)	t := t * 2	16	8		8	8	
4)	WRITE(A,t)	16	16		8	8	<T,A,16>
5)	READ(B,t)	8	8	8	8	8	
6)	t := t * 2	16	16	8	8	8	
7)	WRITE(B,t)	16	16	16	8	8	<T,B,16>
8)							<COMMIT T>
9)	FLUSH LOG						
10)	OUTPUT (A)	16	16	16	16	8	
11)	OUTPUT (B)	16	16	16	16	16	

Redo Log with CheckPoint

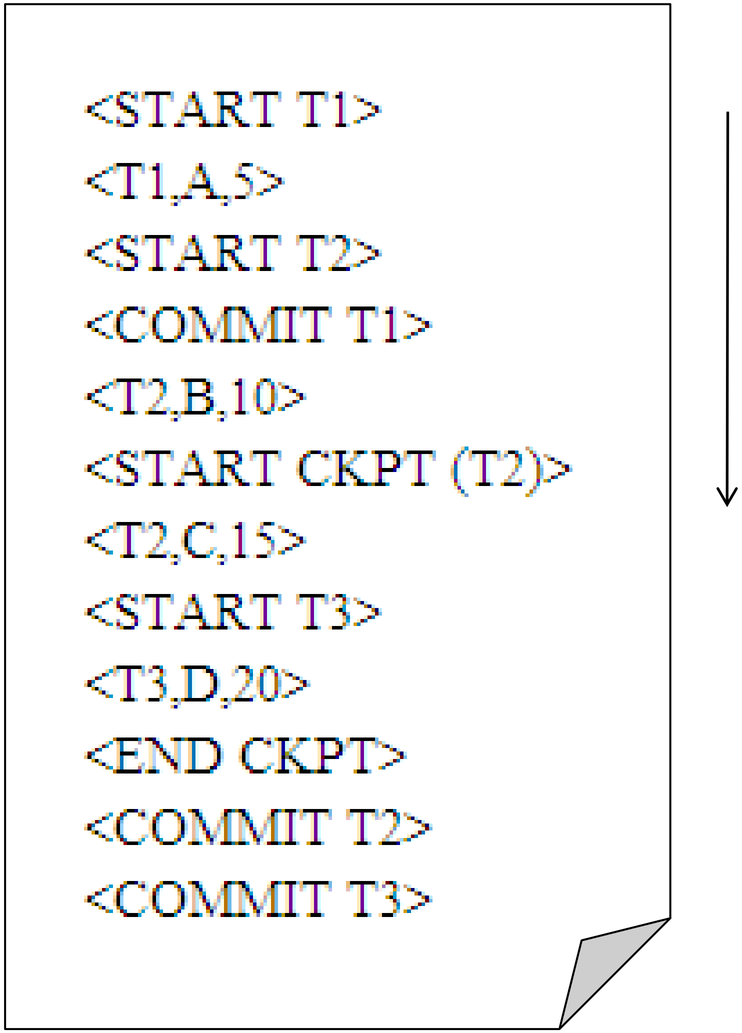
(1) Write a log record <CKPT (T1,...Tk) for all active transactions

(2) Flush the log

(3) Write to disk all database elements that were written to buffers (dirty) by transactions that had already committed when <CKPT .....> was inserted to log

(4) Write <END CKPT>

Redo Log with CheckPoint Example



```
<START T1>
<T1,A,5>
<START T2>
<COMMIT T1>
<T2,B,10>
<START CKPT (T2)>
<T2,C,15>
<START T3>
<T3,D,20>
<END CKPT>
<COMMIT T2>
<COMMIT T3>
```

Key drawbacks:

- *Undo logging:* cannot bring backup DB copies up to date
- *Redo logging:* need to keep all modified blocks in memory until commit

Solution: undo/redo logging!

Update ⇒ <Ti, Xid, New X val, Old X val>

page X

Undo/Redo Log Rules:

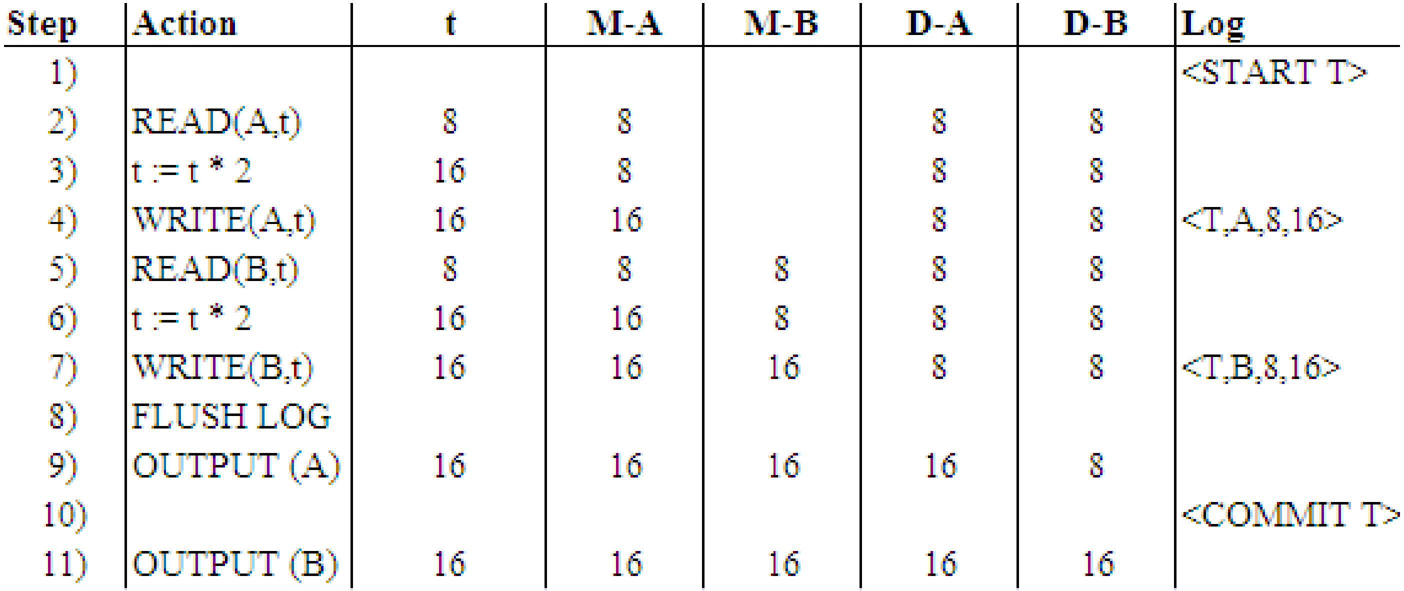
UR_1:

Before modifying any database element X on disk because of changes made by some transaction T, it is necessary that the update record <T,X,v,w> appear on disk

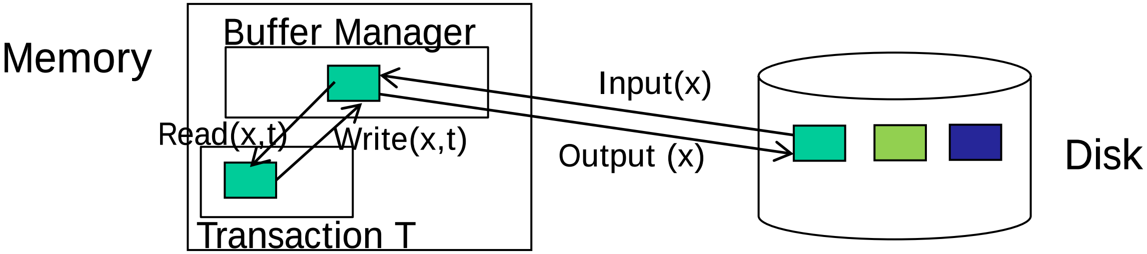
Undo/Redo Log Example:

A := A * 2;
B := B * 2;

UR1: Before modifying any database element X on disk because of changes made by some transaction T, it is necessary that the update record <T,X,v,w> appear on disk

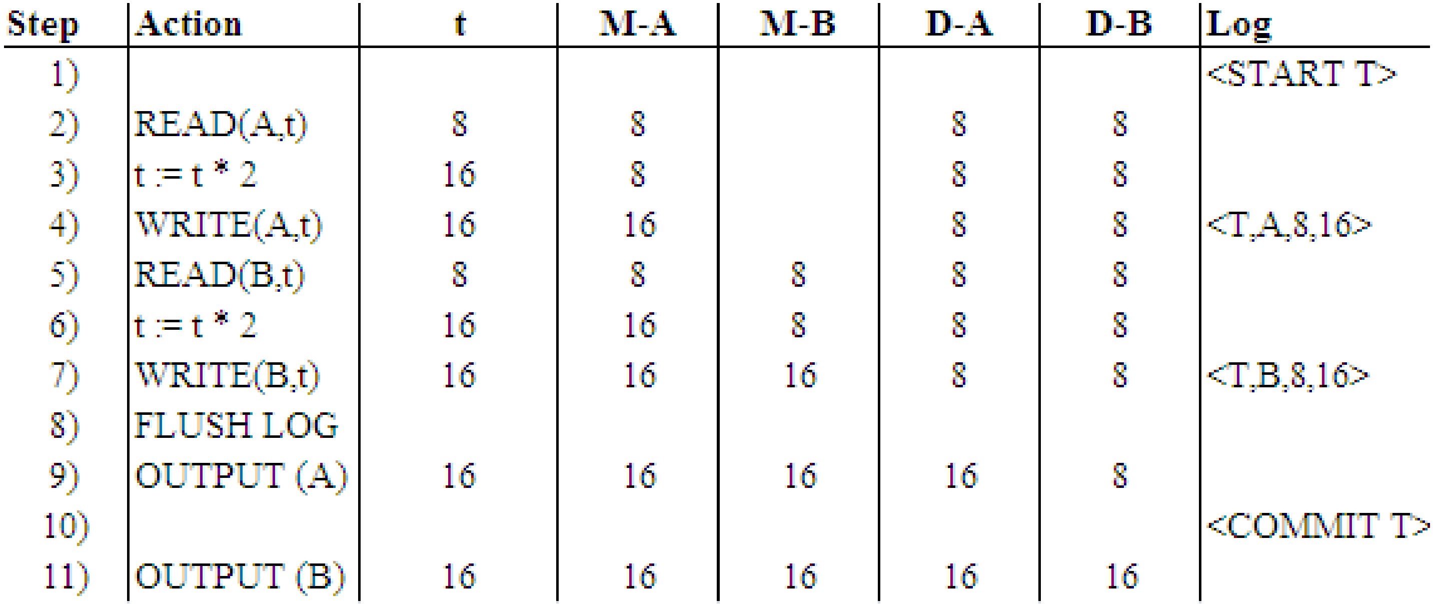


Step	Action	t	M-A	M-B	D-A	D-B	Log
1)							<START T>
2)	READ(A,t)	8	8		8	8	
3)	t := t * 2	16	8		8	8	
4)	WRITE(A,t)	16	16		8	8	<T,A,8,16>
5)	READ(B,t)	8	8	8	8	8	
6)	t := t * 2	16	16	8	8	8	
7)	WRITE(B,t)	16	16	16	8	8	<T,B,8,16>
8)	FLUSH LOG						
9)	OUTPUT (A)	16	16	16	16	8	
10)							<COMMIT T>
11)	OUTPUT (B)	16	16	16	16	16	



Undo/Redo Log: what if a crash happens?

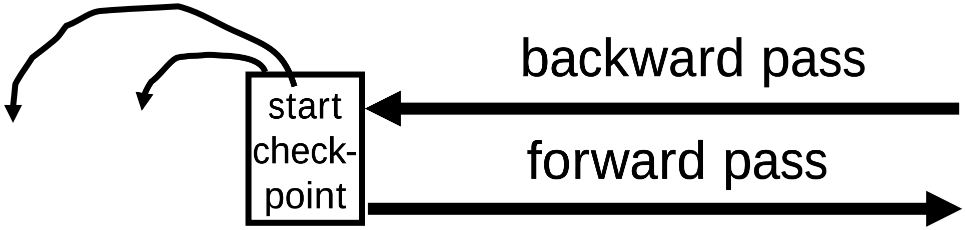
A := A * 2;
B := B * 2;



Step	Action	t	M-A	M-B	D-A	D-B	Log
1)							<START T>
2)	READ(A,t)	8	8		8	8	
3)	t := t * 2	16	8		8	8	
4)	WRITE(A,t)	16	16		8	8	<T,A,8,16>
5)	READ(B,t)	8	8	8	8	8	
6)	t := t * 2	16	16	8	8	8	
7)	WRITE(B,t)	16	16	16	8	8	<T,B,8,16>
8)	FLUSH LOG						
9)	OUTPUT (A)	16	16	16	16	8	
10)							<COMMIT T>
11)	OUTPUT (B)	16	16	16	16	16	

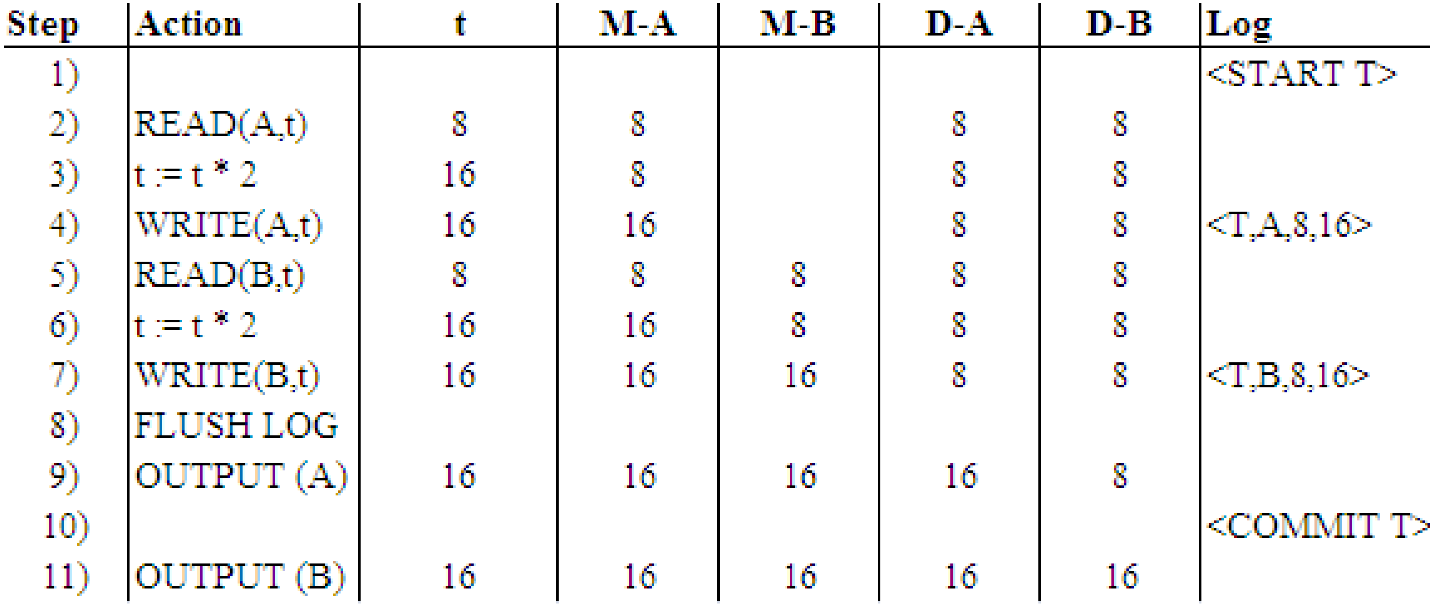
Recovery process:

- Backwards pass (end of log ➲ latest valid checkpoint start)
  - – construct set S of committed transactions
  - – undo actions of transactions not in S
- Undo pending transactions
  - – follow undo chains for transactions in (checkpoint active list) - S
- Forward pass (latest checkpoint start ➲ end of log)
  - – redo actions of S transactions



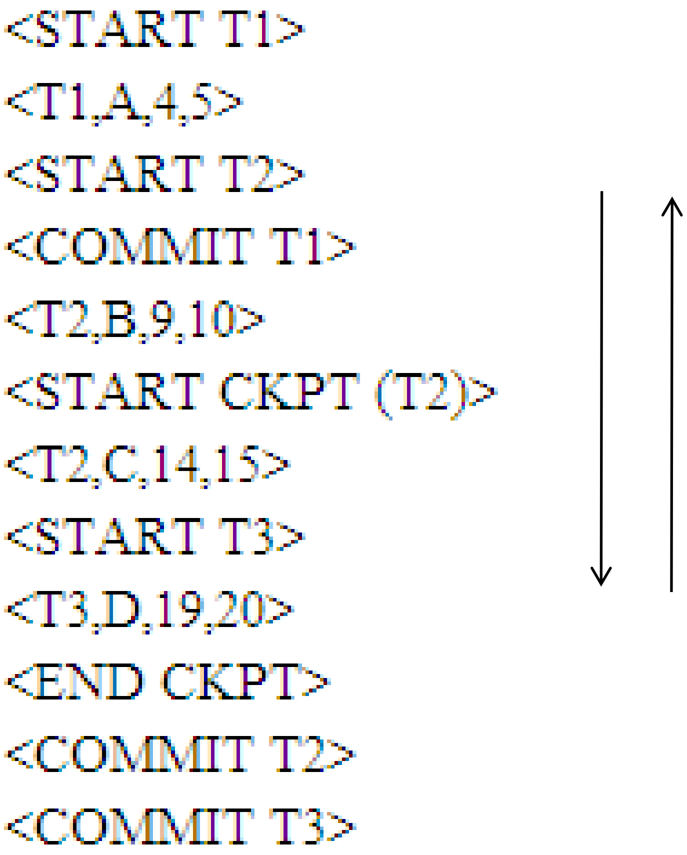
Undo/Redo Log: what if a crash happens?

```
A := A * 2;
B := B * 2;
```



Step	Action	t	M-A	M-B	D-A	D-B	Log
1)							<START T>
2)	READ(A,t)	8	8		8	8	
3)	t := t * 2	16	8		8	8	
4)	WRITE(A,t)	16	16		8	8	<T,A,8,16>
5)	READ(B,t)	8	8	8	8	8	
6)	t := t * 2	16	16	8	8	8	
7)	WRITE(B,t)	16	16	16	8	8	<T,B,8,16>
8)	FLUSH LOG						
9)	OUTPUT (A)	16	16	16	16	8	
10)							<COMMIT T>
11)	OUTPUT (B)	16	16	16	16	16	

Undo/Redo Log with CheckPoint



```
<START T1>
<T1,A,4,5>
<START T2>
<COMMIT T1>
<T2,B,9,10>
<START CKPT (T2)>
<T2,C,14,15>
<START T3>
<T3,D,19,20>
<END CKPT>
<COMMIT T2>
<COMMIT T3>
```