

# **Failure Recovery**

Hector Garcia-Molina and  
Mahmoud SAKR

# Integrity or consistency constraints

- Predicates data must satisfy
- Examples:
  - $x$  is key of relation  $R$
  - $x \rightarrow y$  holds in  $R$
  - $\text{Domain}(x) = \{\text{Red}, \text{Blue}, \text{Green}\}$
  - $\alpha$  is valid index for attribute  $x$  of  $R$
  - no employee should make more than twice the average salary

## Definition:

- Consistent state: satisfies all constraints
- Consistent DB: DB in consistent state

Observation: DB cannot be consistent  
always!

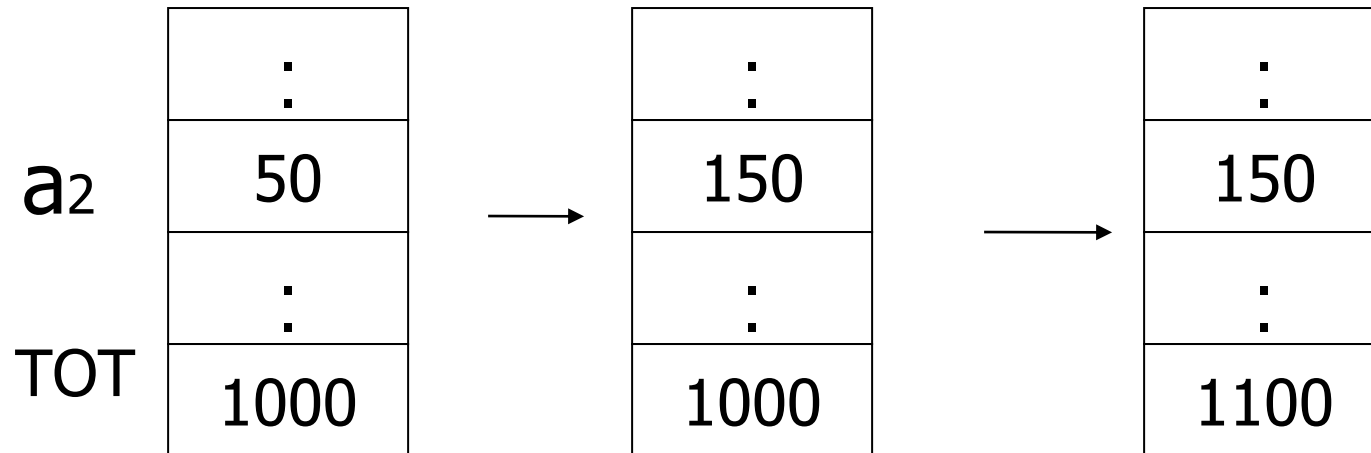
Example:  $a_1 + a_2 + \dots + a_n = \text{TOT}$

Deposit \$100 in  $a_2$ :  $\left\{ \begin{array}{l} a_2 \leftarrow a_2 + 100 \\ \text{TOT} \leftarrow \text{TOT} + 100 \end{array} \right.$

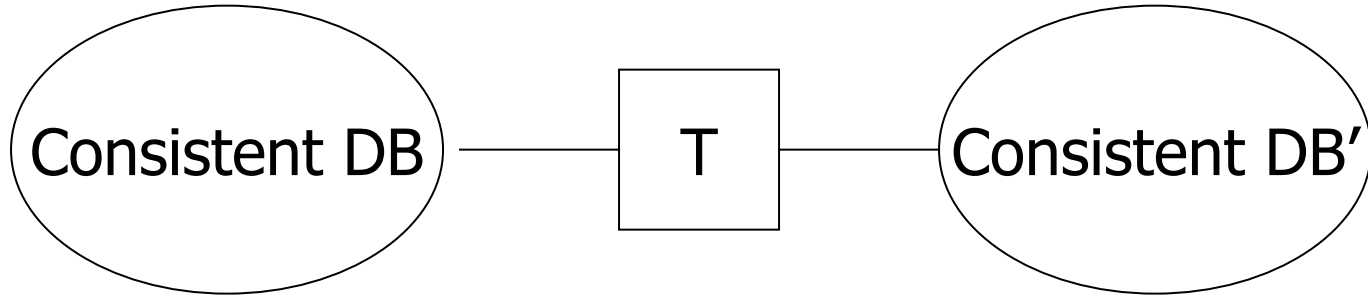
Example:  $a_1 + a_2 + \dots + a_n = \text{TOT}$

Deposit \$100 in  $a_2$ :  $a_2 \leftarrow a_2 + 100$

$\text{TOT} \leftarrow \text{TOT} + 100$



Transaction: collection of actions  
that preserve consistency



## Big assumption:

If T starts with consistent state +

T executes in isolation

⇒ T leaves consistent state

## Correctness (informally)

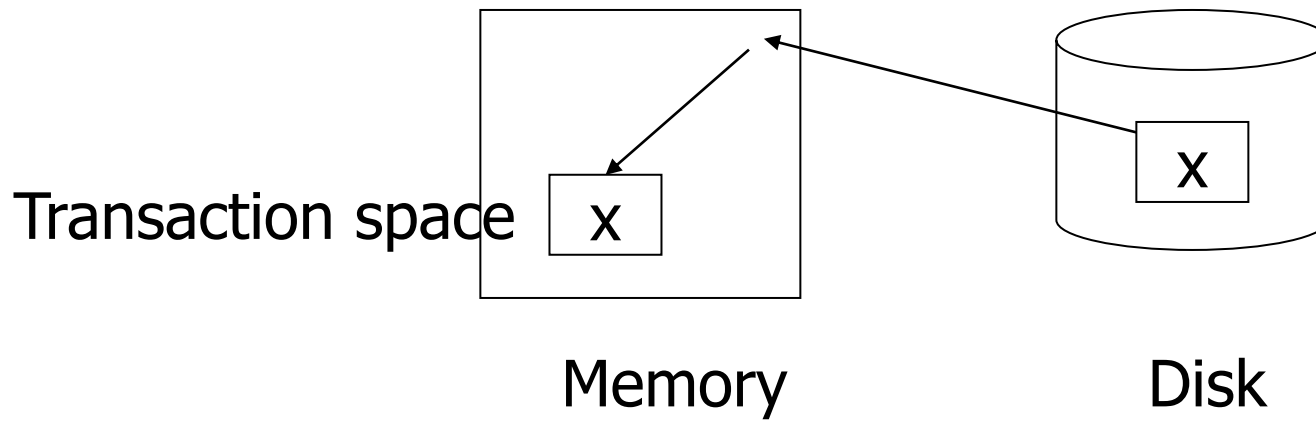
- If we stop running transactions,  
DB left consistent
- Each transaction sees a consistent DB



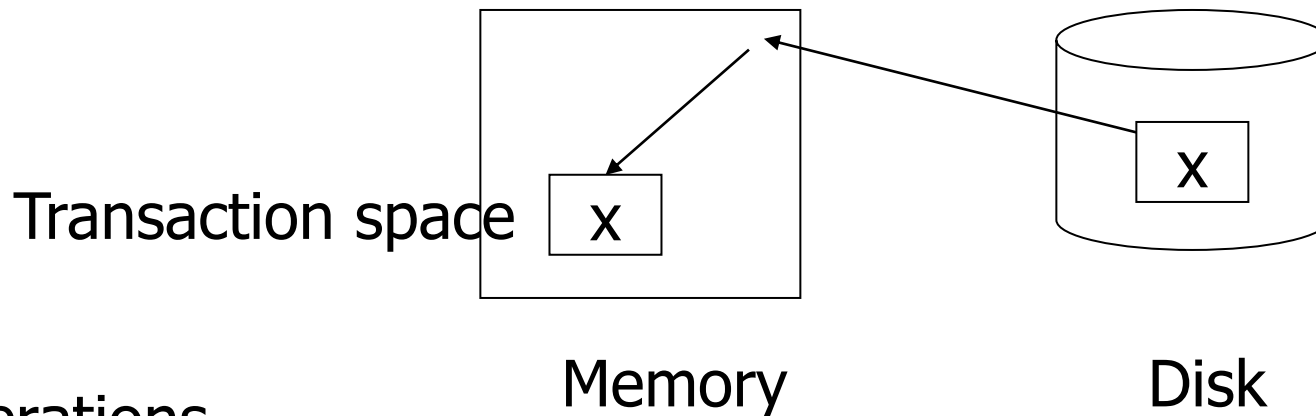
# How can consistency be violated?

- System crash
  - memory lost
  - cpu halts, resets
- Media failure, catastrophes

# Storage hierarchy



# Storage hierarchy



## Operations

- Input (x): block containing x  $\rightarrow$  memory
- Output (x): block containing x  $\rightarrow$  disk
- Read (x,t): do input(x) if necessary,  
t  $\leftarrow$  value of x in block
- Write (x,t): do input(x) if necessary,  
value of x in block  $\leftarrow$  t

# Key problem    Unfinished transaction

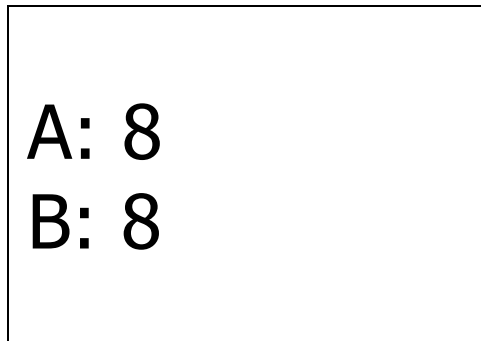
Example

Constraint:  $A=B$

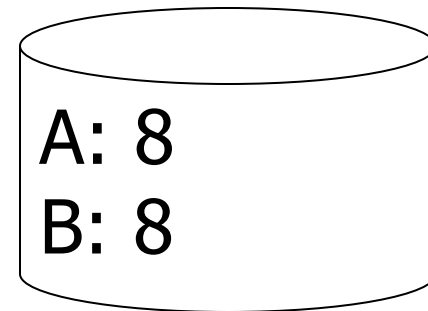
$T_1: A \leftarrow A \times 2$

$B \leftarrow B \times 2$

T<sub>1</sub>:    Read (A,t);     $t \leftarrow t \times 2$   
         Write (A,t);  
         Read (B,t);     $t \leftarrow t \times 2$   
         Write (B,t);  
         Output (A);  
         Output (B);

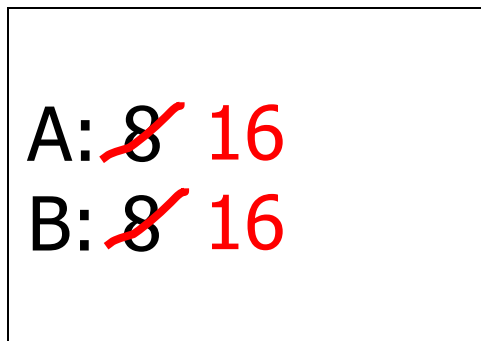


memory

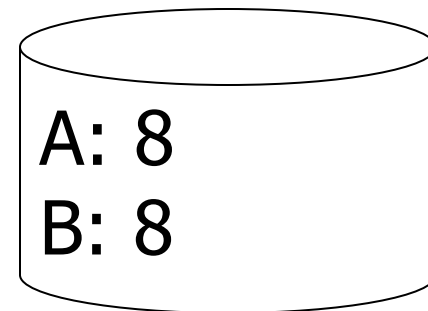


disk

T<sub>1</sub>: Read (A,t);  $t \leftarrow t \times 2$   
Write (A,t);  
Read (B,t);  $t \leftarrow t \times 2$   
Write (B,t);  
Output (A);  
Output (B);

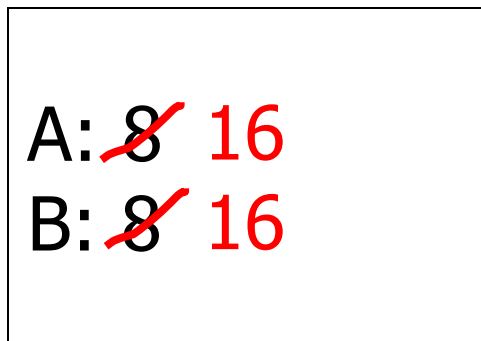


memory

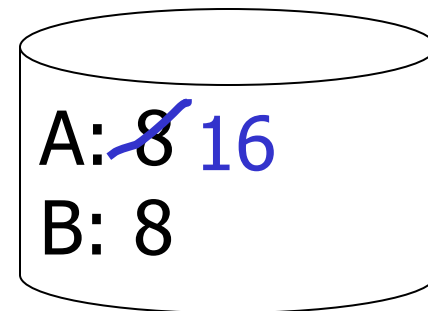


disk

T<sub>1</sub>: Read (A,t);  $t \leftarrow t \times 2$   
Write (A,t);  
Read (B,t);  $t \leftarrow t \times 2$   
Write (B,t);  
Output (A);  
Output (B);      failure!



memory



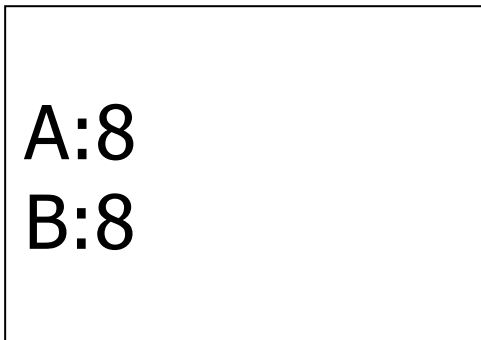
disk

- Need atomicity: execute all actions of a transaction or none at all

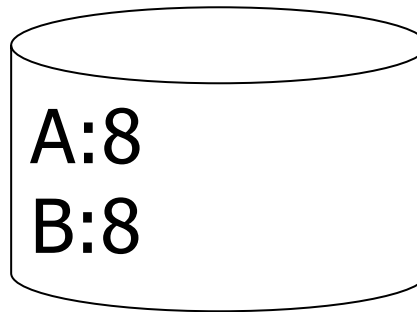


# Undo logging (Immediate modification)

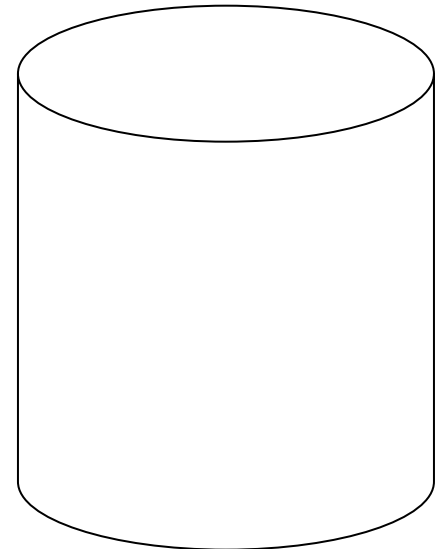
T<sub>1</sub>:    Read (A,t);     $t \leftarrow t \times 2$                     A=B  
         Write (A,t);  
         Read (B,t);     $t \leftarrow t \times 2$   
         Write (B,t);  
         Output (A);  
         Output (B);



memory



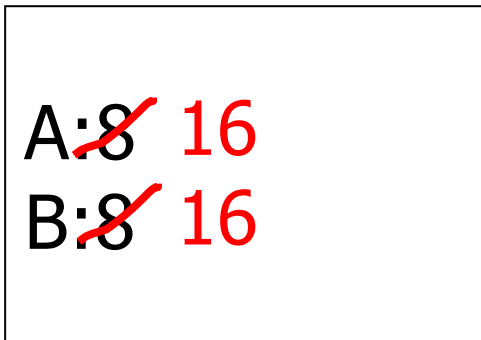
disk



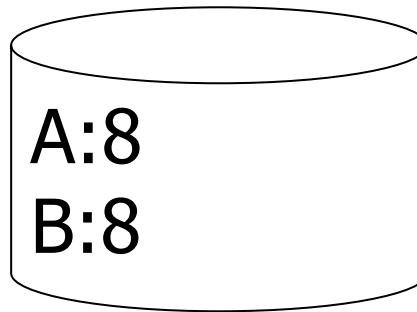
log

# Undo logging (Immediate modification)

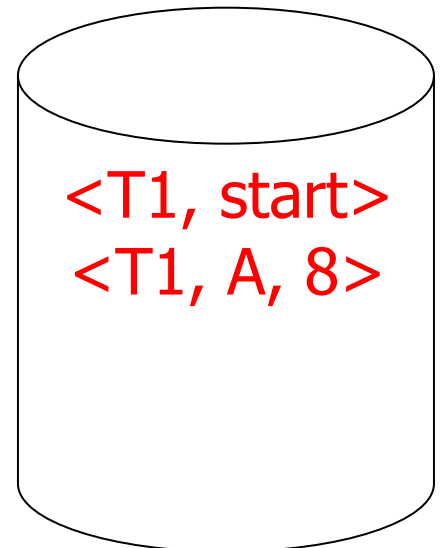
T<sub>1</sub>:    Read (A,t);     $t \leftarrow t \times 2$                     A=B  
         Write (A,t);  
         Read (B,t);     $t \leftarrow t \times 2$   
         Write (B,t);  
         Output (A);  
         Output (B);



memory



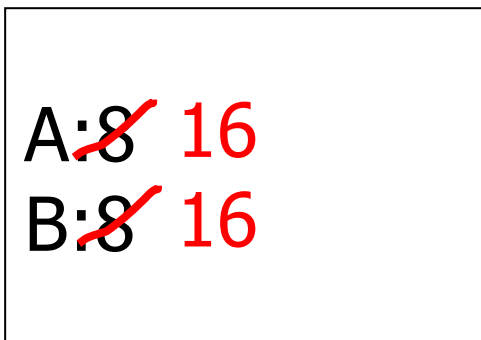
disk



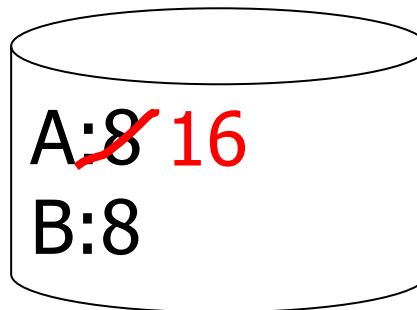
log

# Undo logging (Immediate modification)

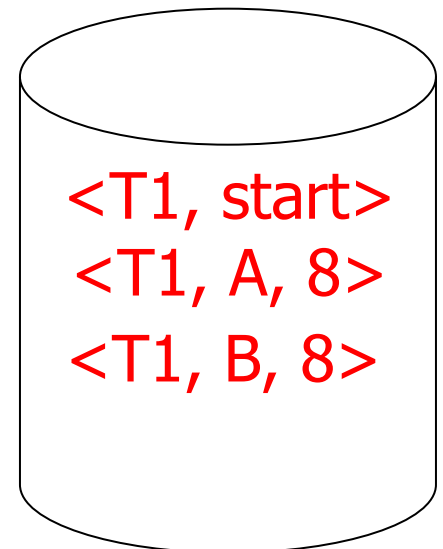
T<sub>1</sub>:    Read (A,t);     $t \leftarrow t \times 2$                     A=B  
         Write (A,t);  
         Read (B,t);     $t \leftarrow t \times 2$   
         Write (B,t);  
         Output (A);  
         Output (B);



memory



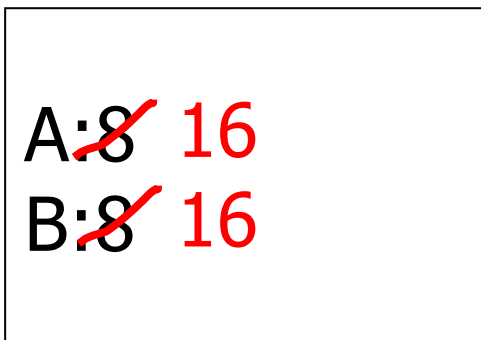
disk



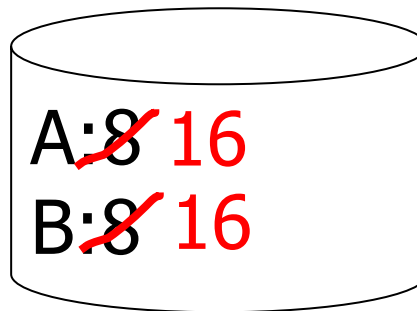
log

# Undo logging (Immediate modification)

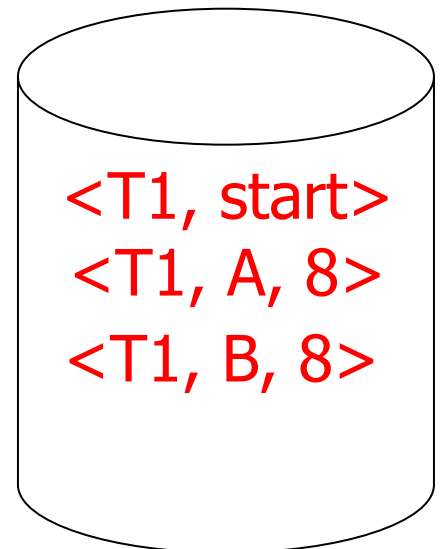
T<sub>1</sub>:    Read (A,t);     $t \leftarrow t \times 2$                     A=B  
         Write (A,t);  
         Read (B,t);     $t \leftarrow t \times 2$   
         Write (B,t);  
         Output (A);  
         Output (B);



memory



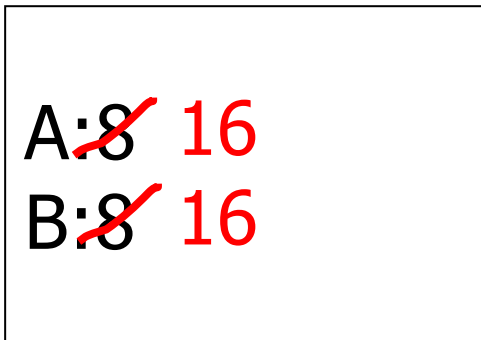
disk



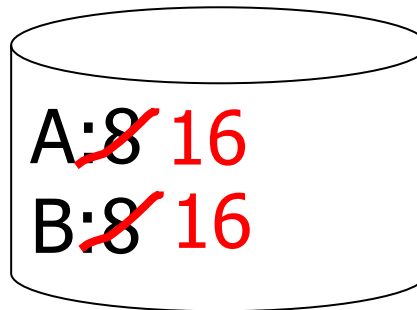
log

# Undo logging (Immediate modification)

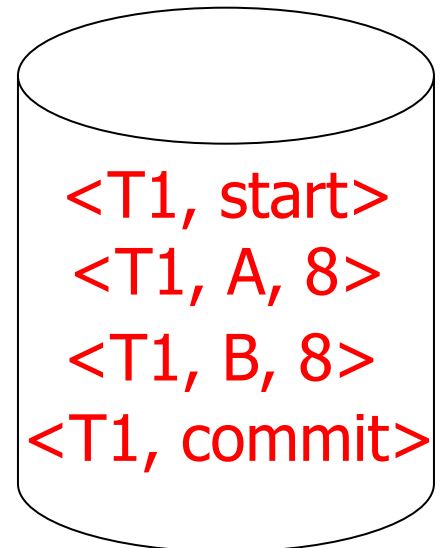
T<sub>1</sub>:    Read (A,t);     $t \leftarrow t \times 2$                     A=B  
         Write (A,t);  
         Read (B,t);     $t \leftarrow t \times 2$   
         Write (B,t);  
         Output (A);  
         Output (B);



memory



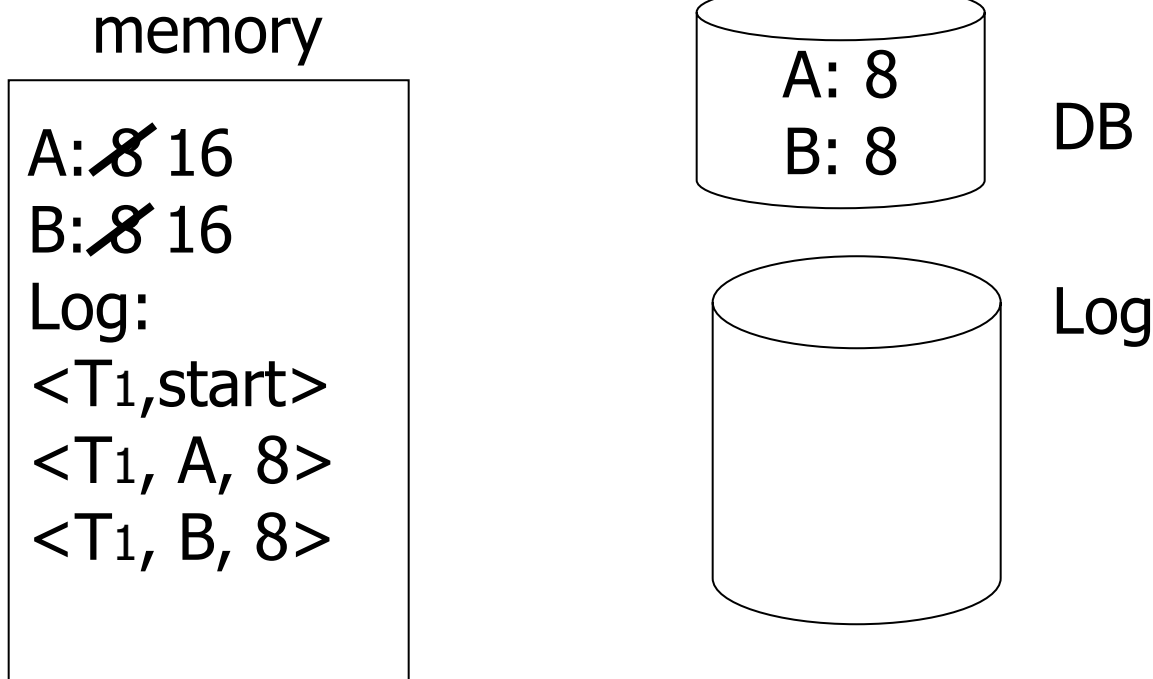
disk



log

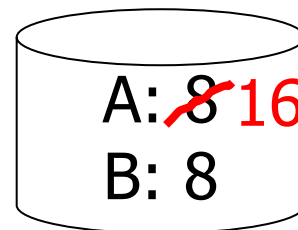
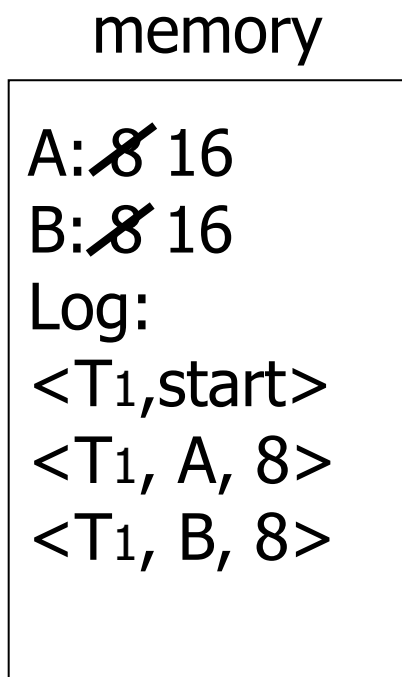
# One “complication”

- Log is first written in memory
- Not written to disk on every action



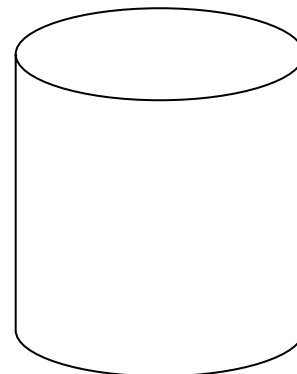
# One “complication”

- Log is first written in memory
- Not written to disk on every action



DB

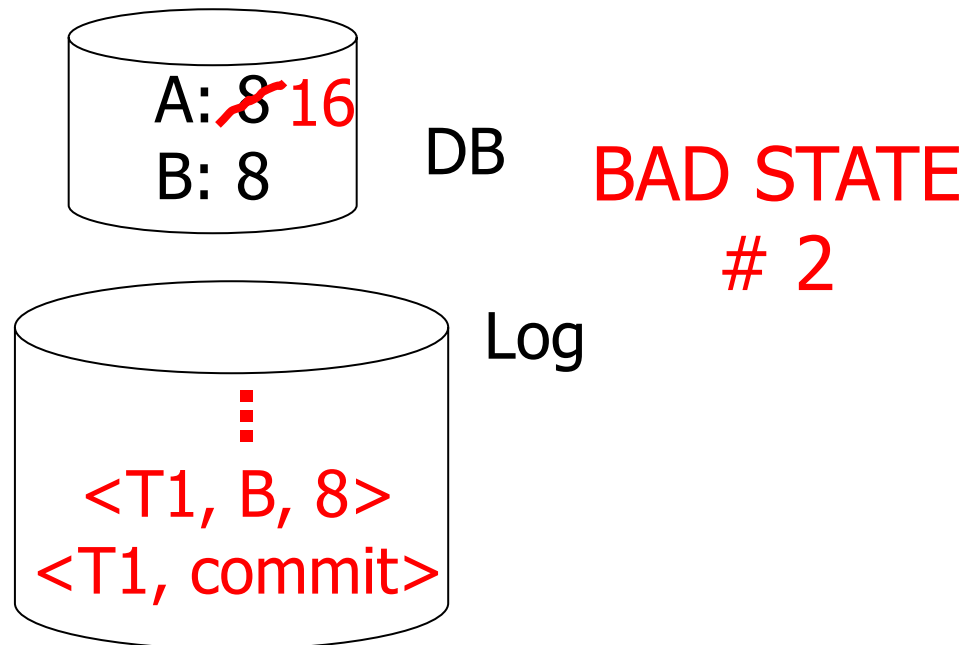
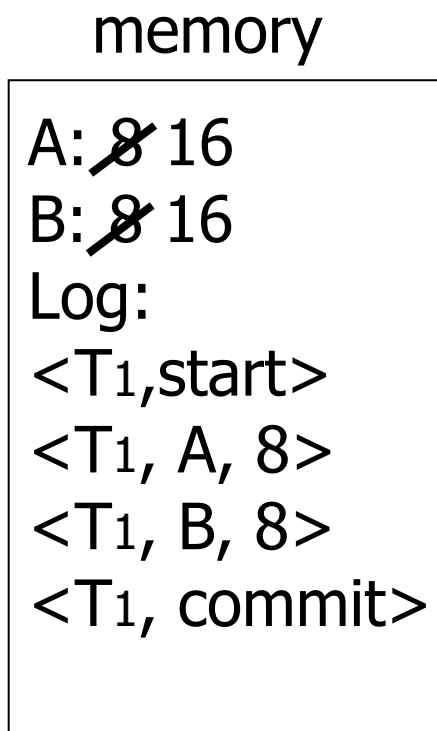
**BAD STATE**  
**# 1**



Log

# One “complication”

- Log is first written in memory
- Not written to disk on every action





# Undo logging rules

- (1) For every action generate undo log record (containing old value)
- (2) Before  $x$  is modified on disk, log records pertaining to  $x$  must be on disk (write ahead logging: WAL)
- (3) Before commit is flushed to log, all writes of transaction must be reflected on disk

# Undo logging example

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	< $T, A, 8$ >
5)	READ(B,t)	8	16	8	8	8	
6)	$t := t*2$	16	16	8	8	8	
7)	WRITE(B,t)	16	16	16	8	8	< $T, B, 8$ >
8)	FLUSH LOG						
9)	OUTPUT(A)	16	16	16	16	8	
10)	OUTPUT(B)	16	16	16	16	16	
11)							<COMMIT $T$ >
12)	FLUSH LOG						

Figure 17.3: Actions and their log entries

## Recovery rules:

## Undo logging

- For every  $T_i$  with  $\langle T_i, \text{start} \rangle$  in log:
  - If  $\langle T_i, \text{commit} \rangle$  or  $\langle T_i, \text{abort} \rangle$  in log, do nothing
  - Else  $\left\{ \begin{array}{l} \text{For all } \langle T_i, X, v \rangle \text{ in log:} \\ \quad \left\{ \begin{array}{l} \text{write } (X, v) \\ \text{output } (X) \end{array} \right. \\ \text{Write } \langle T_i, \text{abort} \rangle \text{ to log} \end{array} \right.$

## Recovery rules:

## Undo logging

- For every  $T_i$  with  $\langle T_i, \text{start} \rangle$  in log:
  - If  $\langle T_i, \text{commit} \rangle$  or  $\langle T_i, \text{abort} \rangle$  in log, do nothing
  - Else  $\left\{ \begin{array}{l} \text{For all } \langle T_i, X, v \rangle \text{ in log:} \\ \quad \left\{ \begin{array}{l} \text{write } (X, v) \\ \text{output } (X) \end{array} \right. \\ \text{Write } \langle T_i, \text{abort} \rangle \text{ to log} \end{array} \right.$

**✗ IS THIS CORRECT??**

## Recovery rules:

## Undo logging

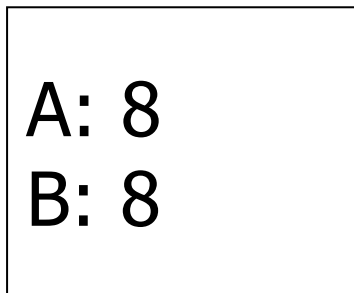
- (1) Let  $S$  = set of transactions with  
     $\langle T_i, \text{start} \rangle$  in log, but no  
     $\langle T_i, \text{commit} \rangle$  (or  $\langle T_i, \text{abort} \rangle$ ) record in log
- (2) For each  $\langle T_i, X, v \rangle$  in log,  
    in reverse order (latest  $\rightarrow$  earliest) do:
  - if  $T_i \in S$  then  $\left\{ \begin{array}{l} \text{- write } (X, v) \\ \text{- output } (X) \end{array} \right.$
- (3) For each  $T_i \in S$  do
  - write  $\langle T_i, \text{abort} \rangle$  to log

What if failure during recovery?

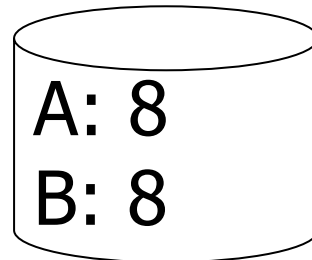
No problem! → Undo idempotent

# Redo logging (deferred modification)

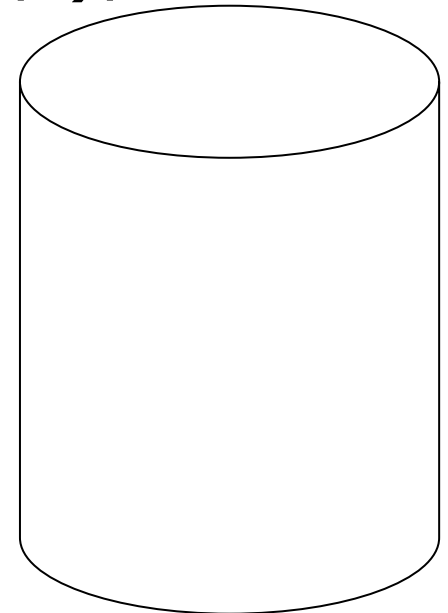
T<sub>1</sub>: Read(A,t);  $t \leftarrow t \times 2$ ; write (A,t);  
Read(B,t);  $t \leftarrow t \times 2$ ; write (B,t);  
Output(A); Output(B)



memory



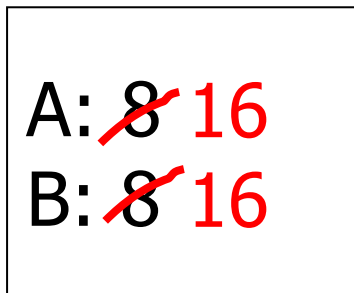
DB



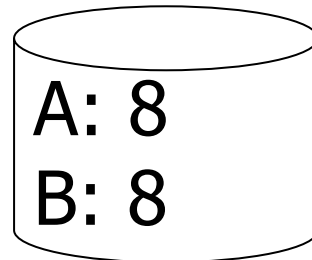
LOG

# Redo logging (deferred modification)

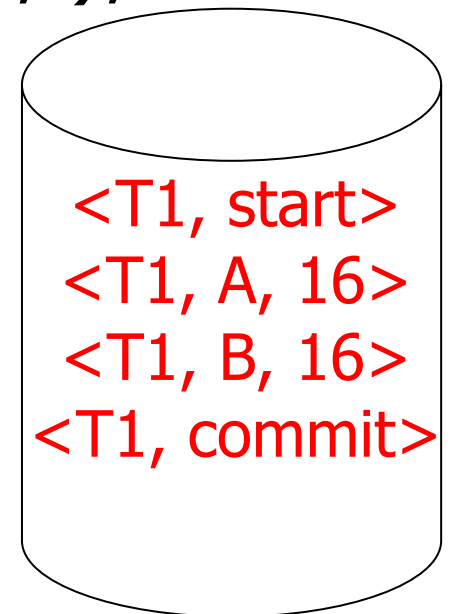
T<sub>1</sub>: Read(A,t);  $t \leftarrow t \times 2$ ; write (A,t);  
Read(B,t);  $t \leftarrow t \times 2$ ; write (B,t);  
Output(A); Output(B)



memory



DB

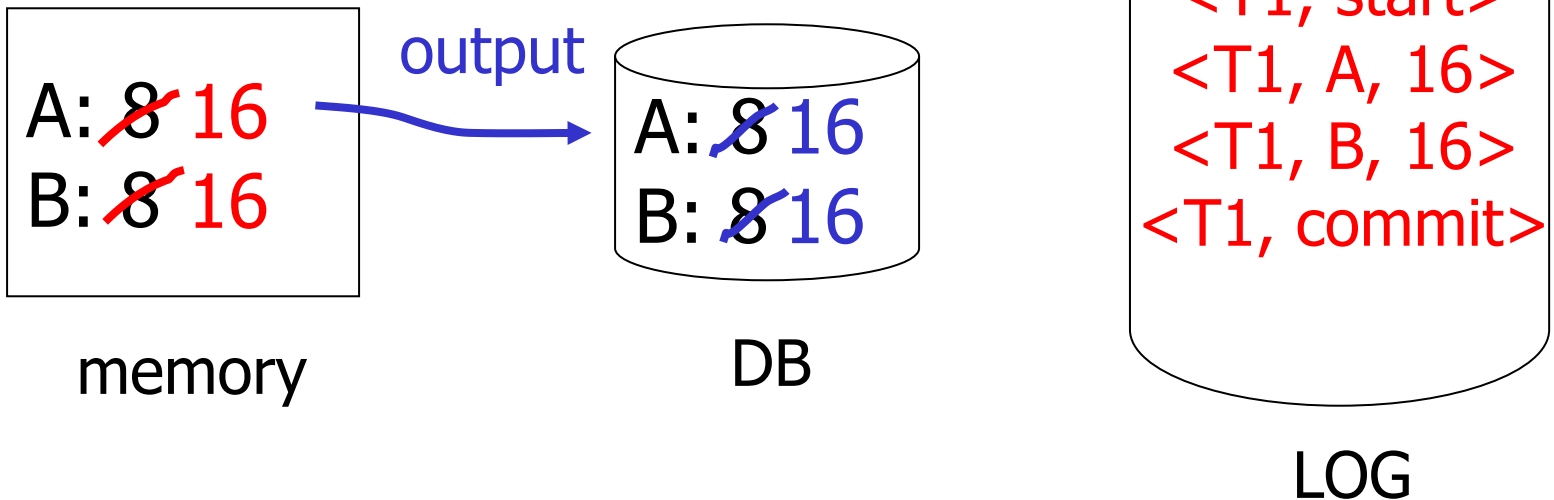


LOG



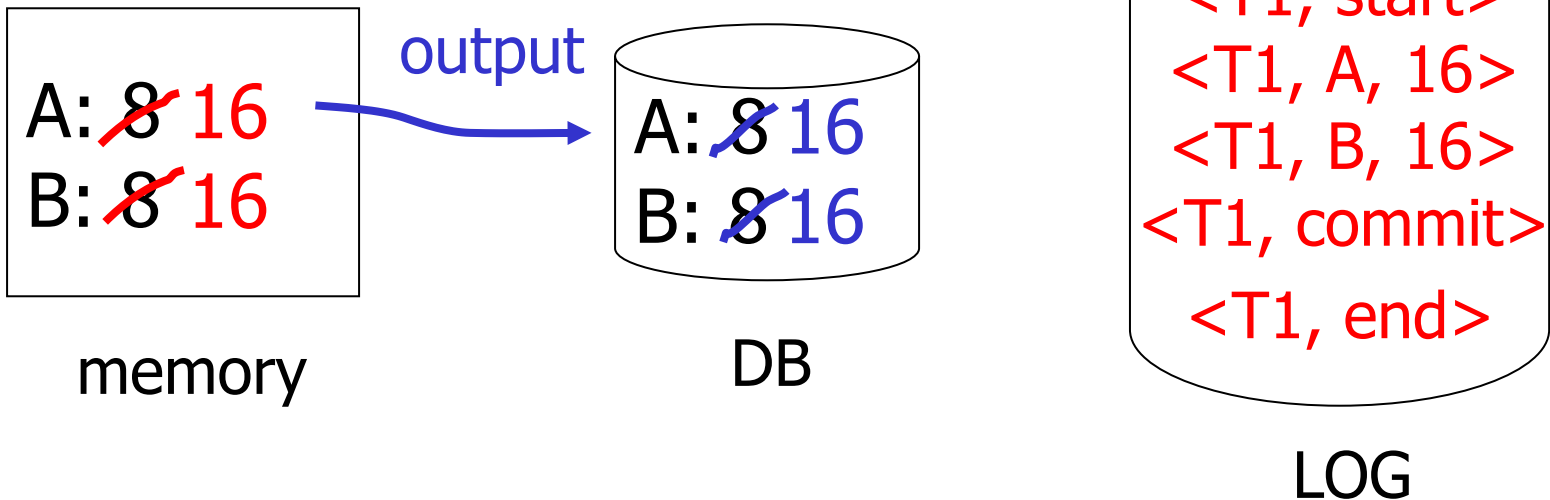
# Redo logging (deferred modification)

T<sub>1</sub>: Read(A,t);  $t \leftarrow t \times 2$ ; write (A,t);  
Read(B,t);  $t \leftarrow t \times 2$ ; write (B,t);  
Output(A); Output(B)



# Redo logging (deferred modification)

T<sub>1</sub>: Read(A,t);  $t \leftarrow t \times 2$ ; write (A,t);  
Read(B,t);  $t \leftarrow t \times 2$ ; write (B,t);  
Output(A); Output(B)



# Redo logging rules

- (1) For every action, generate redo log record (containing new value)
- (2) Before X is modified on disk (DB), all log records for transaction that modified X (including commit) must be on disk
- (3) Flush log at commit
- (4) Write END record after DB updates flushed to disk

# Redo logging example

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	16	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	

Figure 17.7: Actions and their log entries using redo logging

## Recovery rules:

## Redo logging

- For every  $T_i$  with  $\langle T_i, \text{commit} \rangle$  in log:
  - For all  $\langle T_i, X, v \rangle$  in log:
    - $\left\{ \begin{array}{l} \text{Write}(X, v) \\ \text{Output}(X) \end{array} \right.$

## Recovery rules:

## Redo logging

- For every  $T_i$  with  $\langle T_i, \text{commit} \rangle$  in log:
  - For all  $\langle T_i, X, v \rangle$  in log:
    - $\left\{ \begin{array}{l} \text{Write}(X, v) \\ \text{Output}(X) \end{array} \right.$

**✗ IS THIS CORRECT??**

## Recovery rules:

## Redo logging

- (1) Let  $S$  = set of transactions with  $\langle T_i, \text{commit} \rangle$  (and no  $\langle T_i, \text{end} \rangle$ ) in log
- (2) For each  $\langle T_i, X, v \rangle$  in log, in forward order (earliest  $\rightarrow$  latest) do:
  - if  $T_i \in S$  then  $\left\{ \begin{array}{l} \text{Write}(X, v) \\ \text{Output}(X) \end{array} \right.$
- (3) For each  $T_i \in S$ , write  $\langle T_i, \text{end} \rangle$

# Combining $\langle T_i, \text{end} \rangle$ Records

- Want to delay DB flushes for hot objects

## Actions:

Say X is branch balance:

T1: ... update X...

T2: ... update X...

T3: ... update X...

T4: ... update X...

write X  
output X

write X  
output X

write X  
output X

write X  
output X



# Combining $\langle T_i, \text{end} \rangle$ Records

- Want to delay DB flushes for hot objects

## Actions:

Say X is branch balance:

T1: ... update X...

T2: ... update X...

T3: ... update X...

T4: ... update X...

write X

~~output X~~

write X

~~output X~~

write X

~~output X~~

write X

output X

combined  $\langle \text{end} \rangle$  (checkpoint)

# Solution: Checkpoint

- no  $\langle ti, end \rangle$  actions
- simple checkpoint

## Periodically:

- (1) Do not accept new transactions
- (2) Wait until all transactions finish
- (3) Flush all log records to disk (log)
- (4) Flush all buffers to disk (DB) (do not discard buffers)
- (5) Write "checkpoint" record on disk (log)
- (6) Resume transaction processing

# Example: what to do at recovery?

Redo log (disk):

⋮	<T1,A,16>	⋮	<T1,commit>	⋮	Checkpoint	⋮	<T2,B,17>	⋮	<T2,commit>	⋮	<T3,C,21>	Crash
---	-----------	---	-------------	---	------------	---	-----------	---	-------------	---	-----------	-------

## Key drawbacks:

- *Undo logging*: increased I/O cost
- *Redo logging*: need to keep all modified blocks in memory until commit

Solution: undo/redo logging!

Update  $\Rightarrow$   $\langle Ti, Xid, \text{New } X \text{ val}, \text{Old } X \text{ val} \rangle$   
page X

# Rules

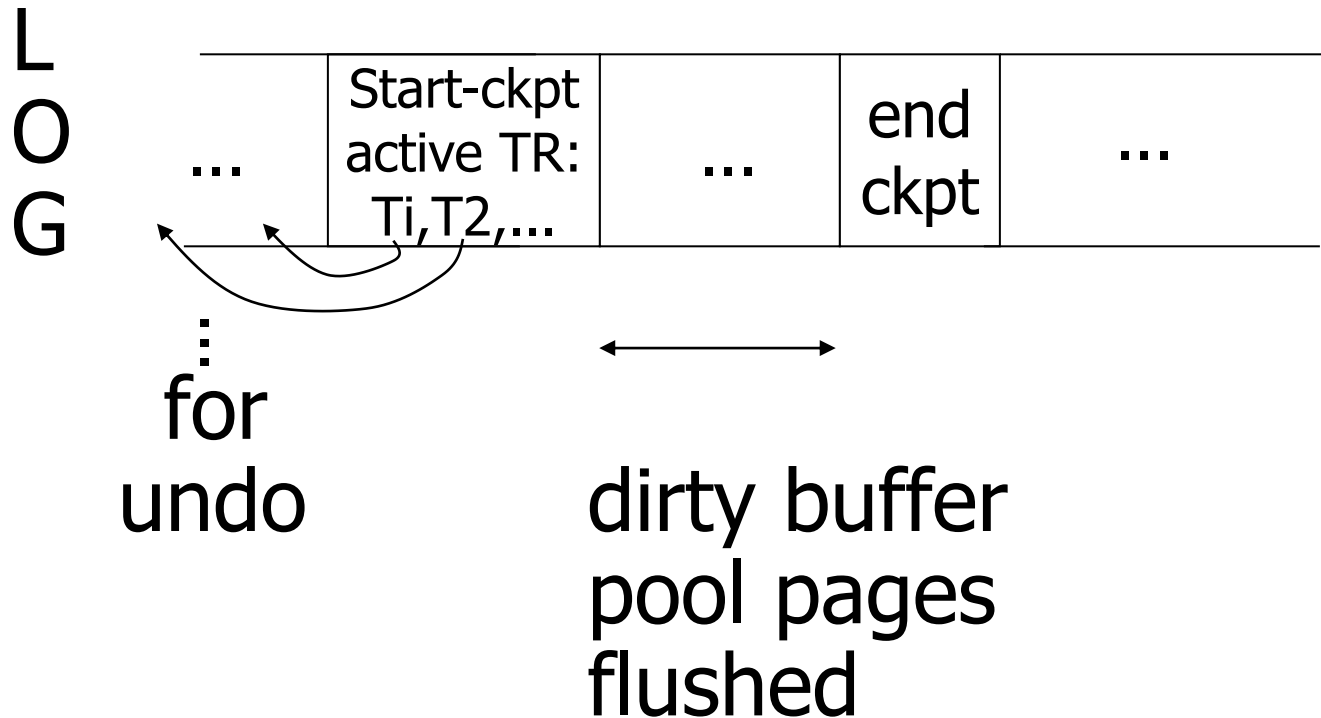
- Page X can be flushed before or after Ti commit
- Log record flushed before corresponding updated page (WAL)
- Flush at commit (log only)

# Example: Undo/Redo logging

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	16	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	

Figure 17.9: A possible sequence of actions and their log entries using undo/redo logging

# Non-quiesce checkpoint

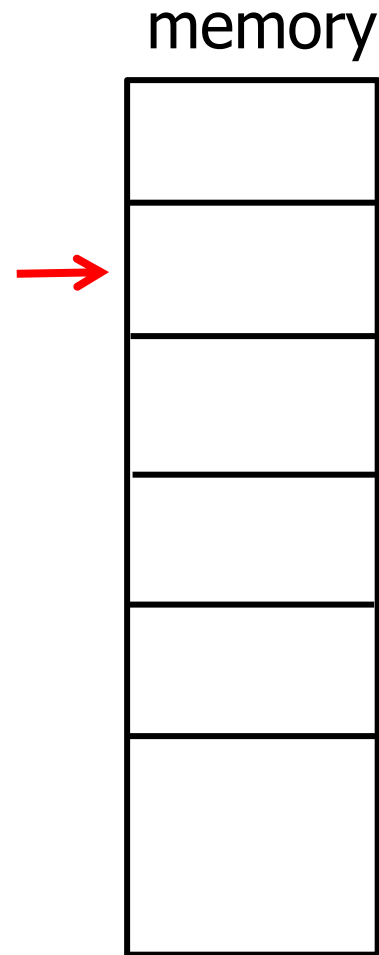




# Non-quiesce checkpoint

checkpoint process:  
for  $i := 1$  to  $M$  do  
    output(buffer  $i$ )

[transactions run concurrently]




# Examples    what to do at recovery time?

no T1 commit

L  
O  
G

...	T <sub>1</sub> , a	...	Ckpt T <sub>1</sub>	...	Ckpt end	...	T <sub>1</sub> - b	
-----	-----------------------	-----	------------------------	-----	-------------	-----	-----------------------	--




# Examples    what to do at recovery time?

no T1 commit

L  
O  
G

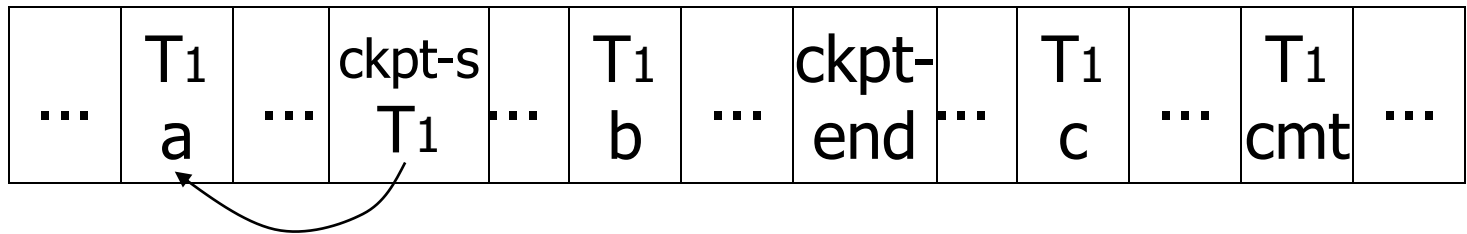
...	T <sub>1</sub> , a	...	Ckpt T <sub>1</sub>	...	Ckpt end	...	T <sub>1</sub> - b	
-----	-----------------------	-----	------------------------	-----	-------------	-----	-----------------------	--



☒ Undo T<sub>1</sub> (undo a,b)

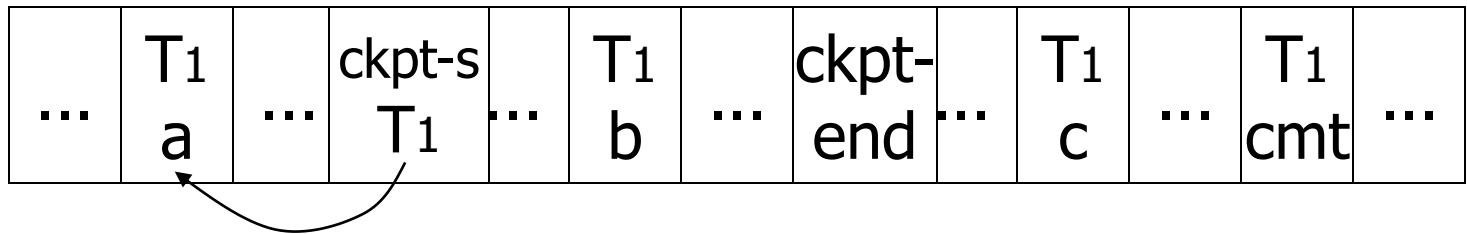
# Example

L  
O  
G



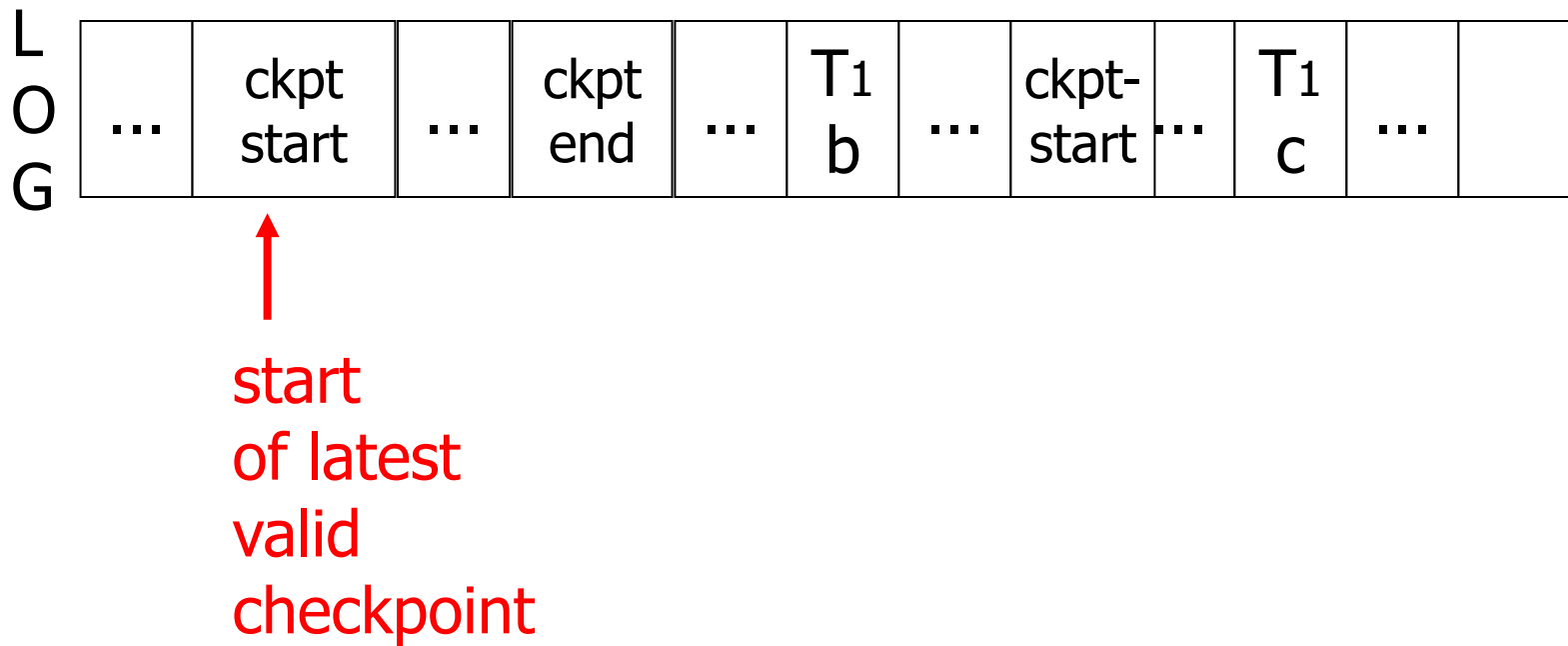
# Example

L  
O  
G



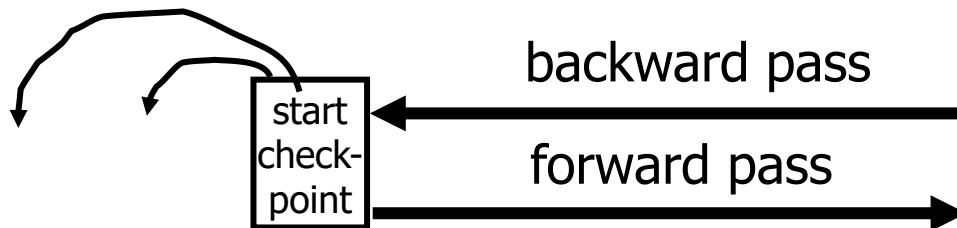
☒ Redo T<sub>1</sub>: (redo b,c)

# Recover From Valid Checkpoint:



# Recovery process:

- **Backwards pass** (end of log  $\Rightarrow$  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  $\Rightarrow$  end of log)
  - redo actions of  $S$  transactions



# Real world actions

E.g., dispense cash at ATM

$$T_i = a_1 a_2 \dots a_j \dots a_n$$



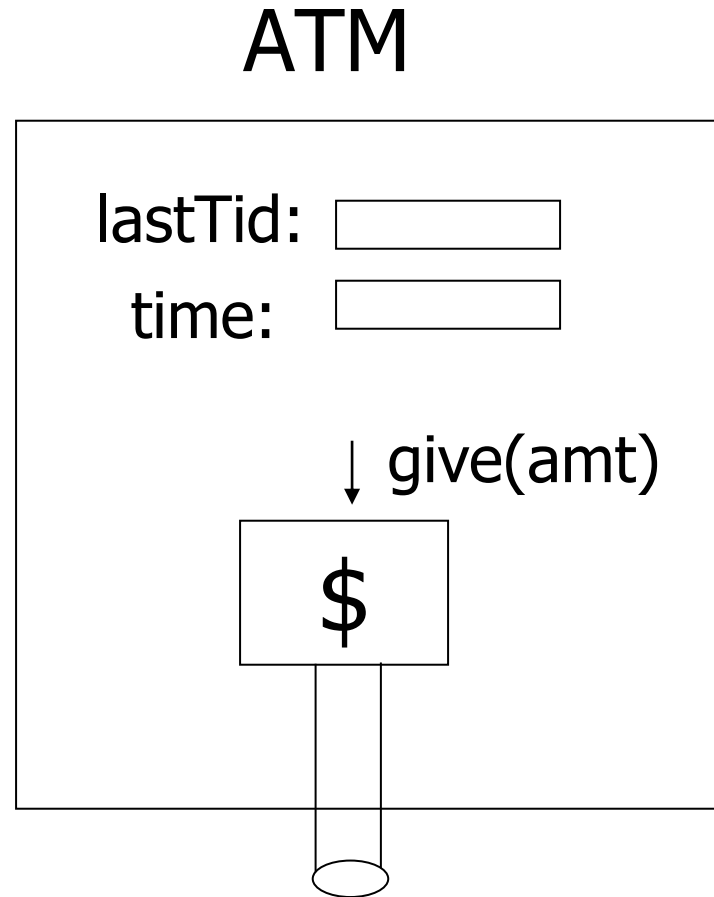

\$



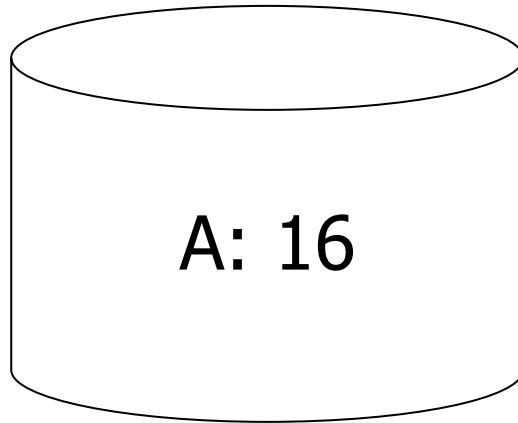
# Solution

- (1) execute real-world actions after commit
- (2) try to make idempotent

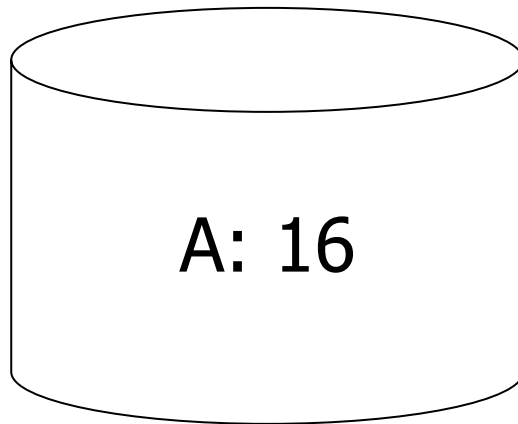
Give\$\$  
(amt, Tid, time)



# Media failure (loss of non-volatile storage)



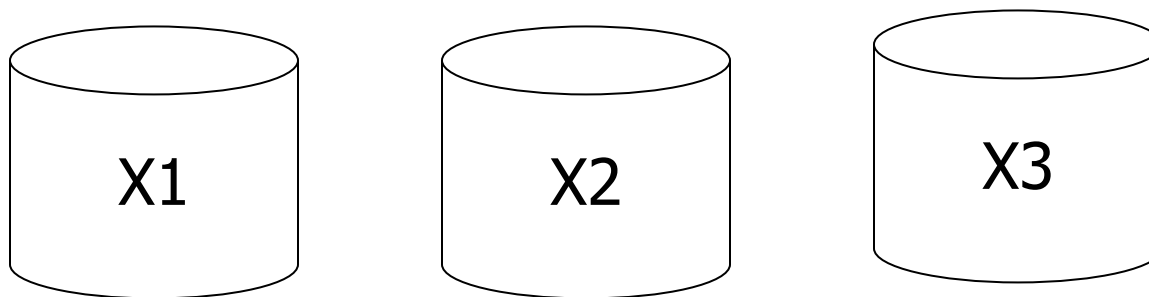
Media failure (loss of non-volatile storage)



Solution: Make copies of data!

## Example 1 Triple modular redundancy

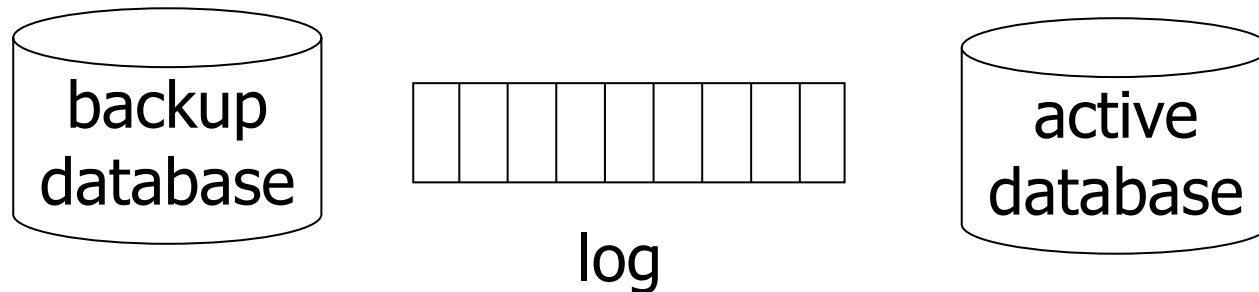
- Keep 3 copies on separate disks
- Output(X) --> three outputs
- Input(X) --> three inputs + vote



## Example #2      Redundant writes, Single reads

- Keep N copies on separate disks
  - Output(X) --> N outputs
  - Input(X) --> Input one copy
    - if ok, done
    - else try another one
- ⇔ Assumes bad data can be detected

## Example #3: DB Dump + Log

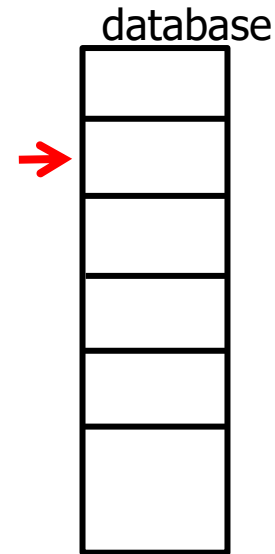


- If active database is lost,
  - restore active database from backup
  - bring up-to-date using redo entries in log

# Backup Database

- Just like checkpoint, except that we write full database

```
create backup database:  
for i := 1 to DB_Size do  
    [read DB block i; write to backup]  
  
[transactions run concurrently]
```

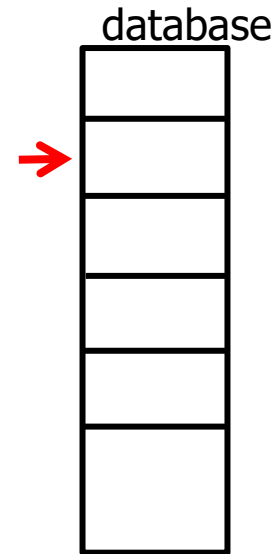




# Backup Database

- Just like checkpoint, except that we write full database

```
create backup database:  
for i := 1 to DB_Size do  
    [read DB block i; write to backup]  
  
[transactions run concurrently]
```



- Restore from backup DB and log:  
Similar to recovery from checkpoint and log

# Summary

- Consistency of data
- One source of problems: failures
  - Logging
  - Redundancy
- Another source of problems:  
Data Sharing (not in this lecture)