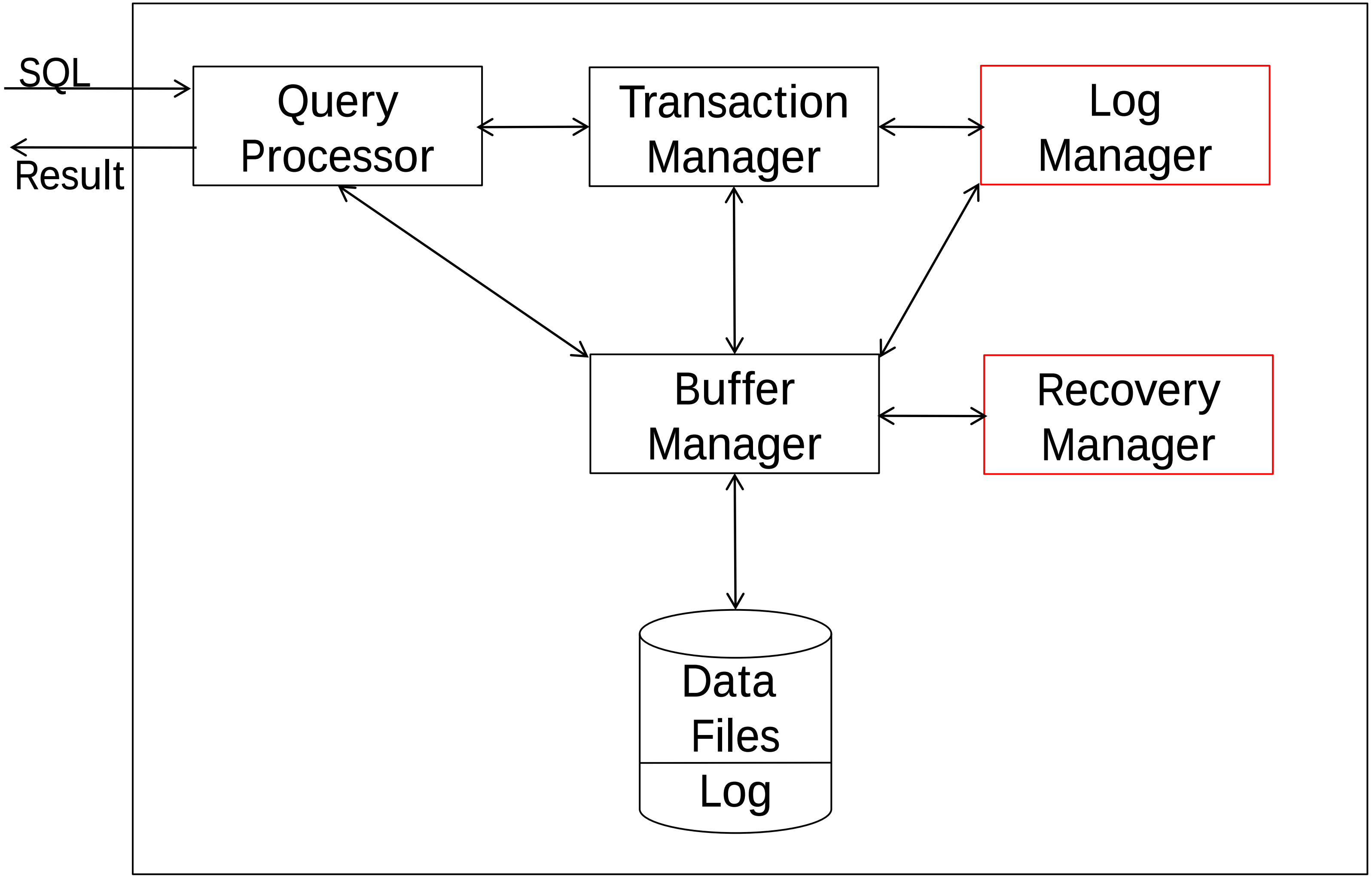
CSCD43: Database Systems Technology

Lecture 10

Wael Aboulsaadat

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

DBMS Architecture



Integrity or correctness of data

- Would like data to be "accurate" or "correct" at all times

EMP

Name	Age
White	52
Green	3421
Gray	1

Integrity or consistency constraints

- Predicates data must satisfy
- Examples:
  - x is key of relation R
  - $x \rightarrow y$ holds in R
  - Domain(x) = {Red, Blue, Green}
  - α 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

Constraints (as we use here) may not capture "full correctness"

Examples   Transaction constraints

- When salary is updated,
  new salary > old salary
- When account record is deleted,
  balance = 0

☞ in any case, continue with constraints...

Observation: DB cannot be consistent always if something goes wrong!

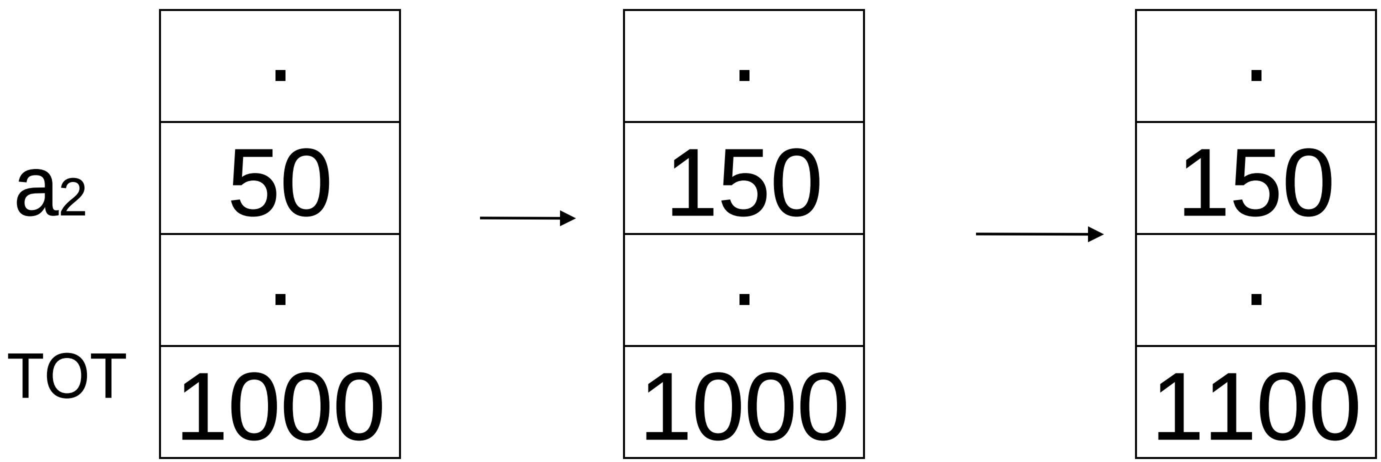
Example: $a_1 + a_2 + \ldots\ a_n = TOT$ (constraint)

Deposit \$100 in a_2: $\begin{cases} a_2 \leftarrow a_2 + 100 \\ TOT \leftarrow TOT + 100 \end{cases}$

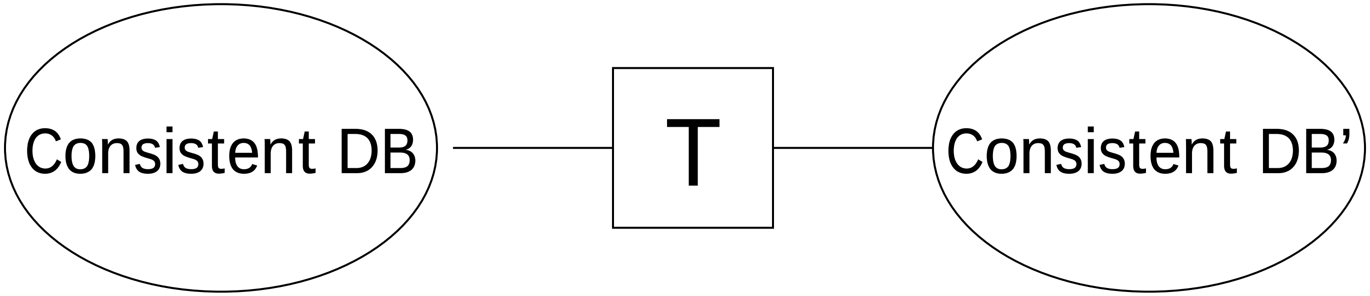
Example: $a_1 + a_2 + \ldots. a_n = TOT$ (constraint)

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

$TOT \leftarrow TOT + 100$



Transaction: collection of actions that preserve consistency



Big assumption:

If T starts with consistent state +

T executes in isolation

$\Rightarrow$ T leaves consistent state

Correctness (informally)

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

How can constraints be violated?

- Transaction bug
- DBMS bug
- Hardware failure

  e.g., disk crash alters balance of account

- Data sharing

  e.g.: T1: give 10% raise to programmers

  T2: change programmers $\Longrightarrow$ systems analysts

How can we prevent/fix violations?

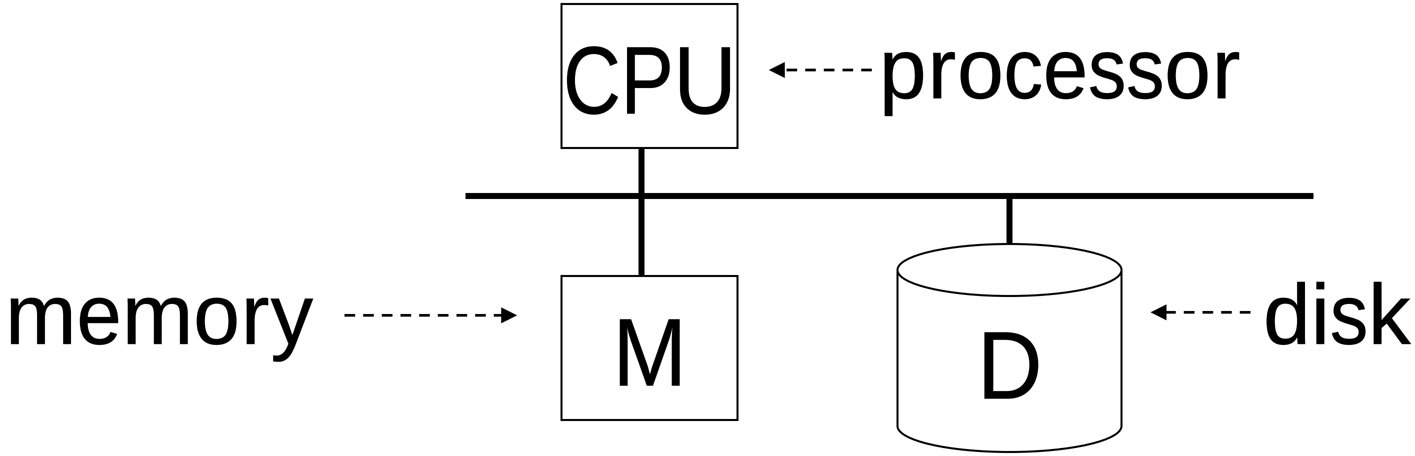
- Chapter 17: due to failures only
- Chapter 18: due to data sharing only
- Chapter 19: due to failures and sharing

Recovery

- First order of business:
  Failure Model

- Events
  - Desired
  - Undesired
    - Expected
    - Unexpected

Our failure model



Desired events: see product manuals....

Undesired expected events:

System crash

- memory lost
- cpu halts, resets

Desired events: see product manuals….

Undesired expected events:

System crash

- memory lost
- cpu halts, resets

that's it!!

Undesired Unexpected: Everything else!

Undesired Unexpected: Everything else!

Examples:

- Disk data is lost
- Memory lost without CPU halt
- CPU implodes wiping out universe….

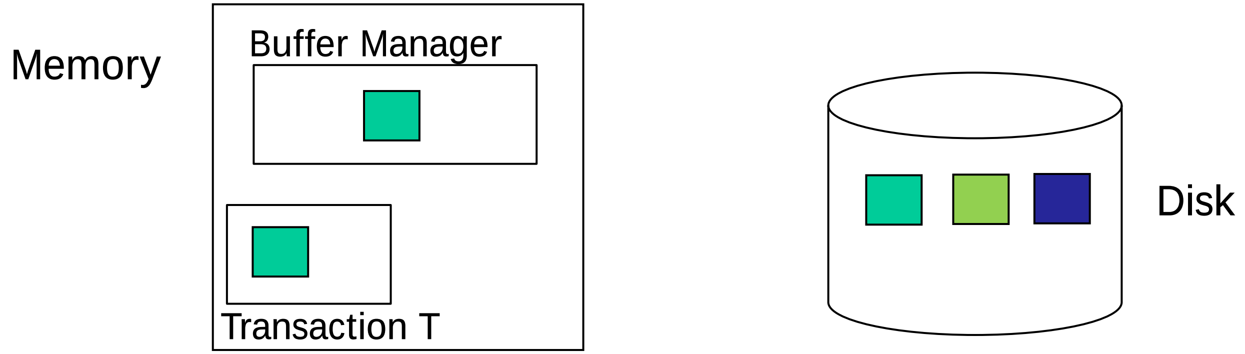
Is this model reasonable?

Approach: Add low level checks + redundancy to increase probability model holds

E.g.,
- Replicate disk storage (stable store)
- Memory parity
- CPU checks

Second order of business:

Storage hierarchy



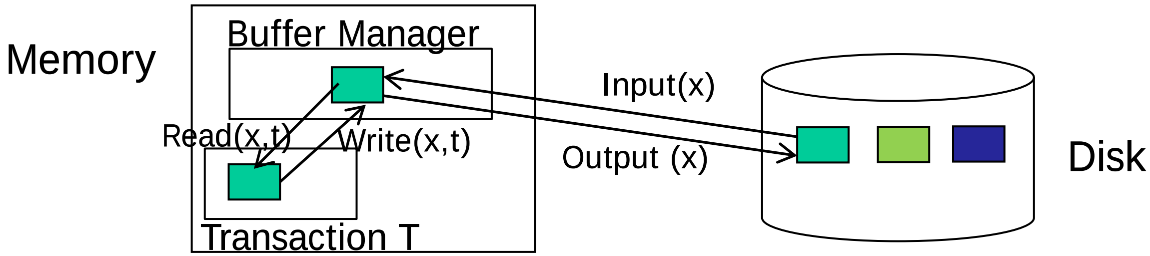
Operations:

- Input (x): block containing x $\rightarrow$ memory
- Output (x): block containing x $\rightarrow$ disk

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



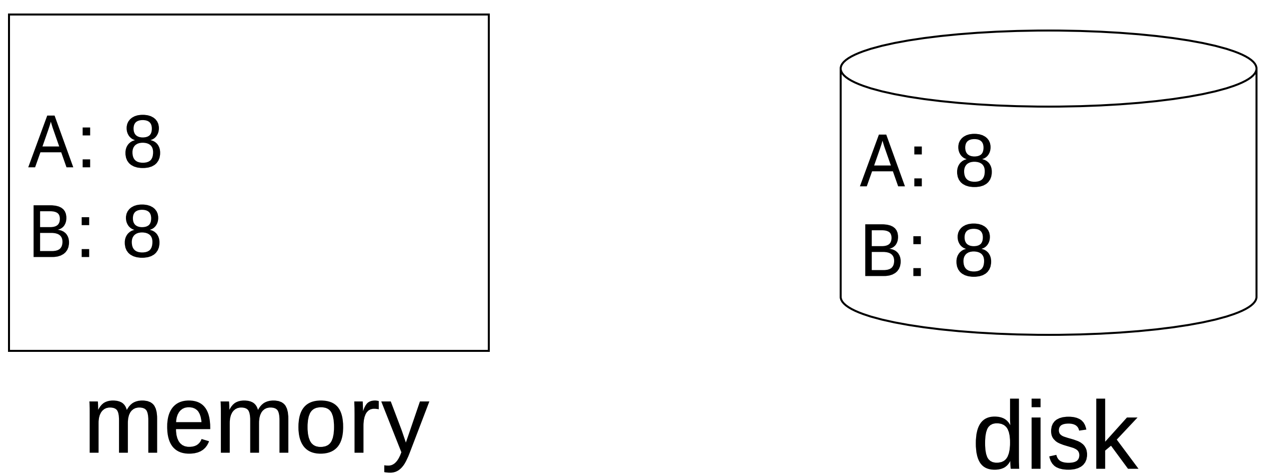
Key problem Unfinished transaction

Example Constraint: A=B

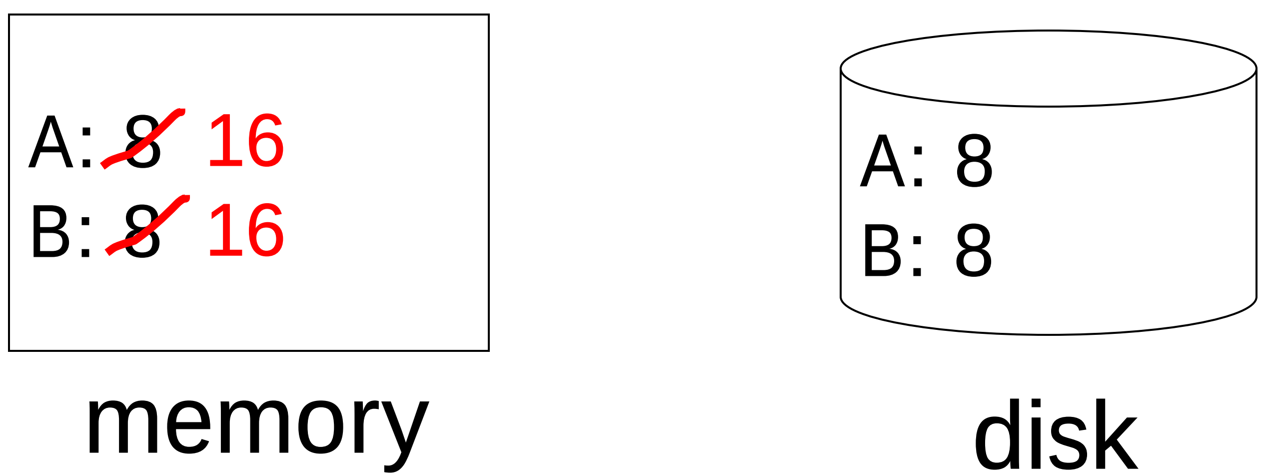
T_1: $A \leftarrow A \times 2$

$B \leftarrow B \times 2$

T_1: 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);

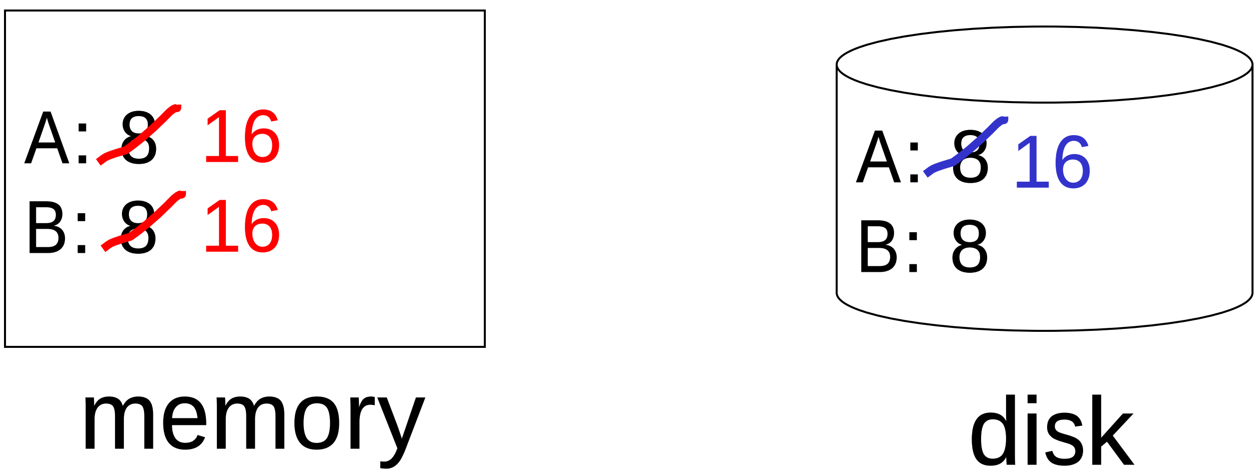


T_1: Read (A,t); $t \leftarrow t \times 2$
Write (A,t);
Read (B,t); $t \leftarrow t \times 2$
Write (B,t);
Output (A);
Output (B);

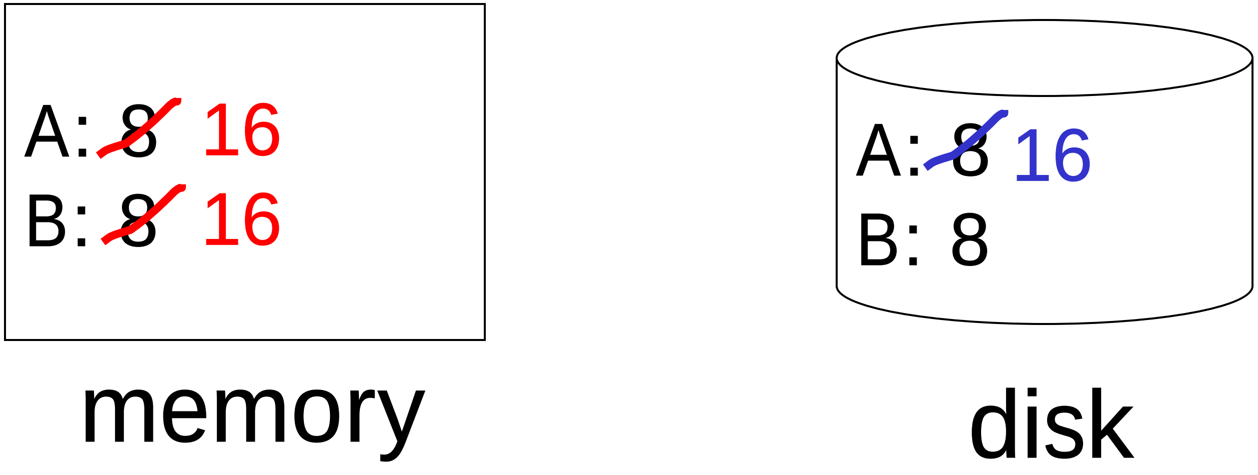


T_1: Read (A,t); $t \leftarrow t \times 2$
Write (A,t);
Read (B,t); $t \leftarrow t \times 2$
Write (B,t);
Output (A);
Output (B);

failure!



- Need <u>atomicity:</u> execute all actions of a transaction or none at all



Solution: keep a log to track

Which transaction started?

What did it do? (or what it is going to do?)

Which transaction finished?

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

Log Commands:

<Start T>

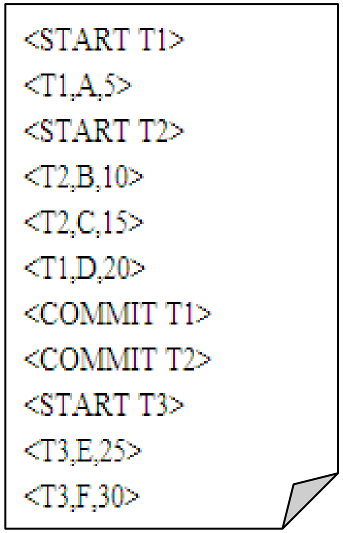
log the start of a transaction

<T1, X, value>

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

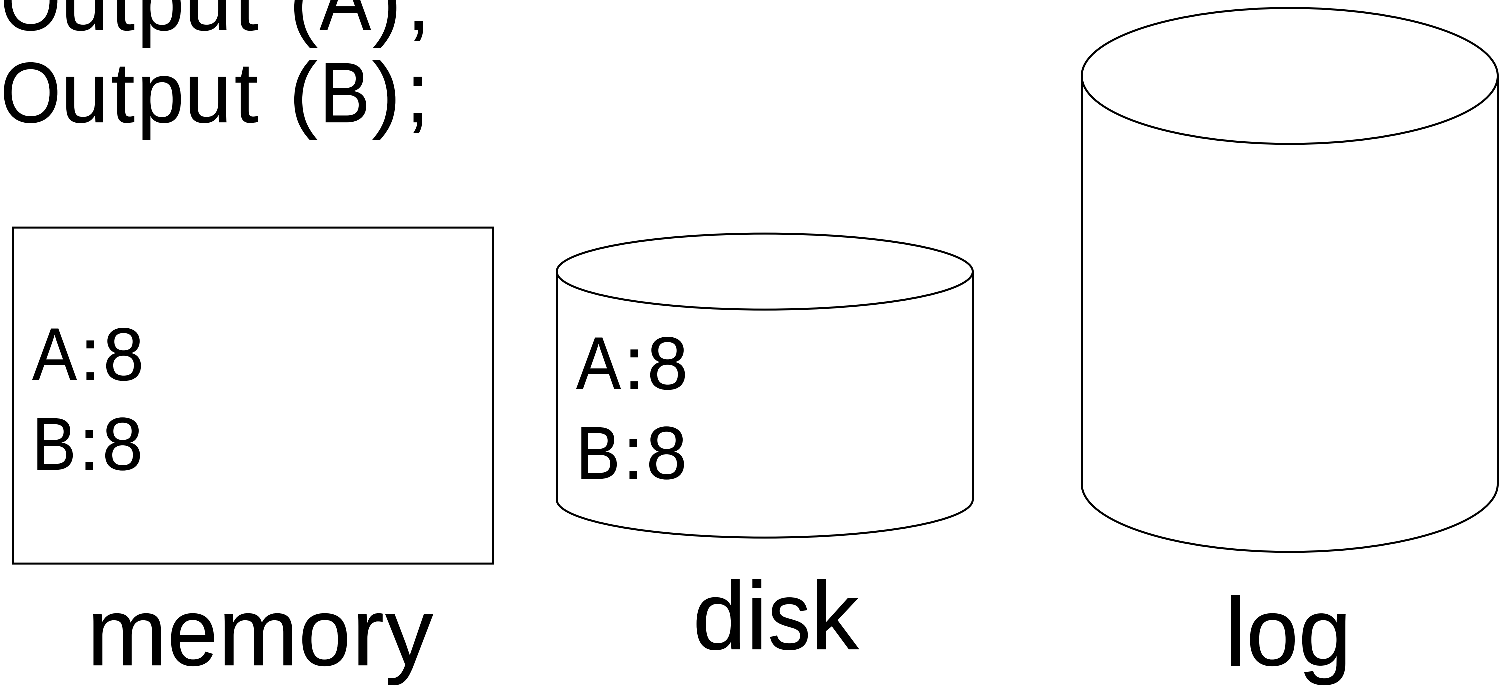
<COMMIT T>

log the completion of a transaction



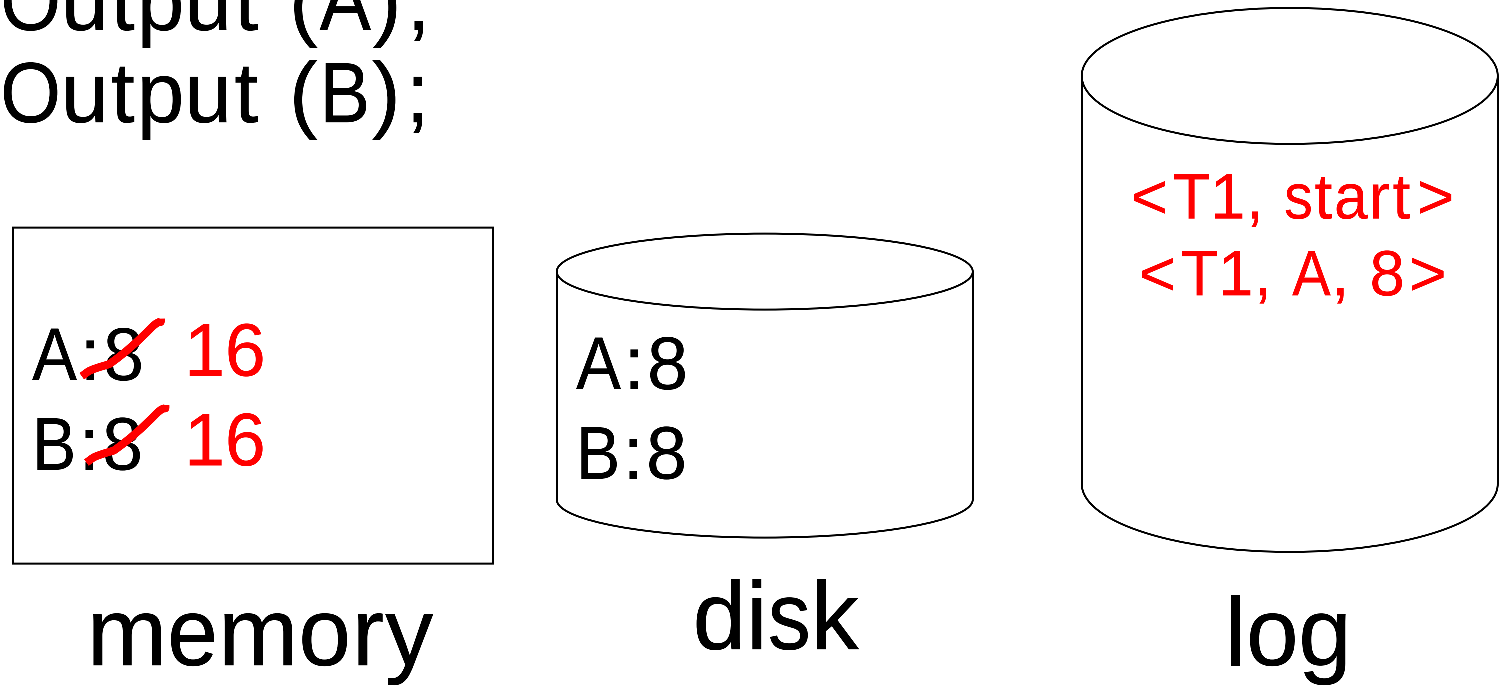
Undo logging (Immediate modification)

T1: 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);



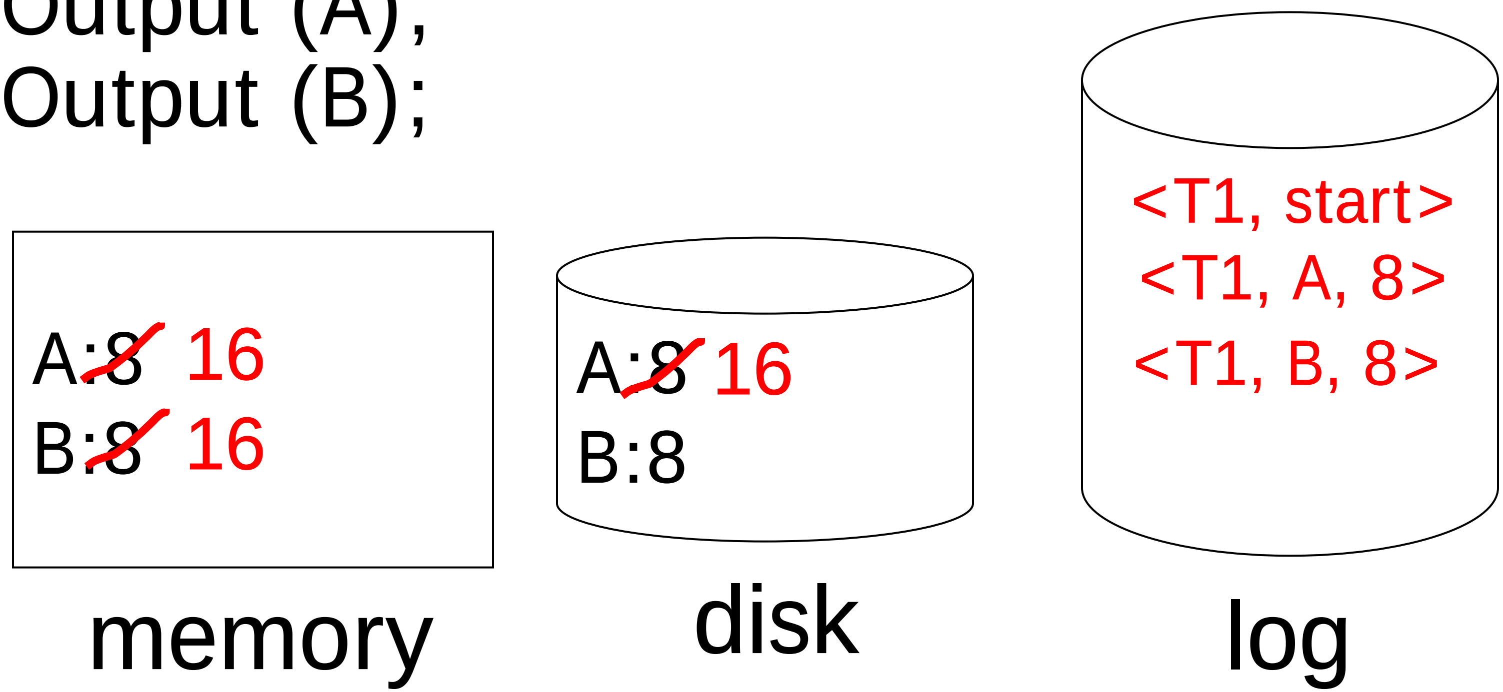
Undo logging (Immediate modification)

T1: 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);



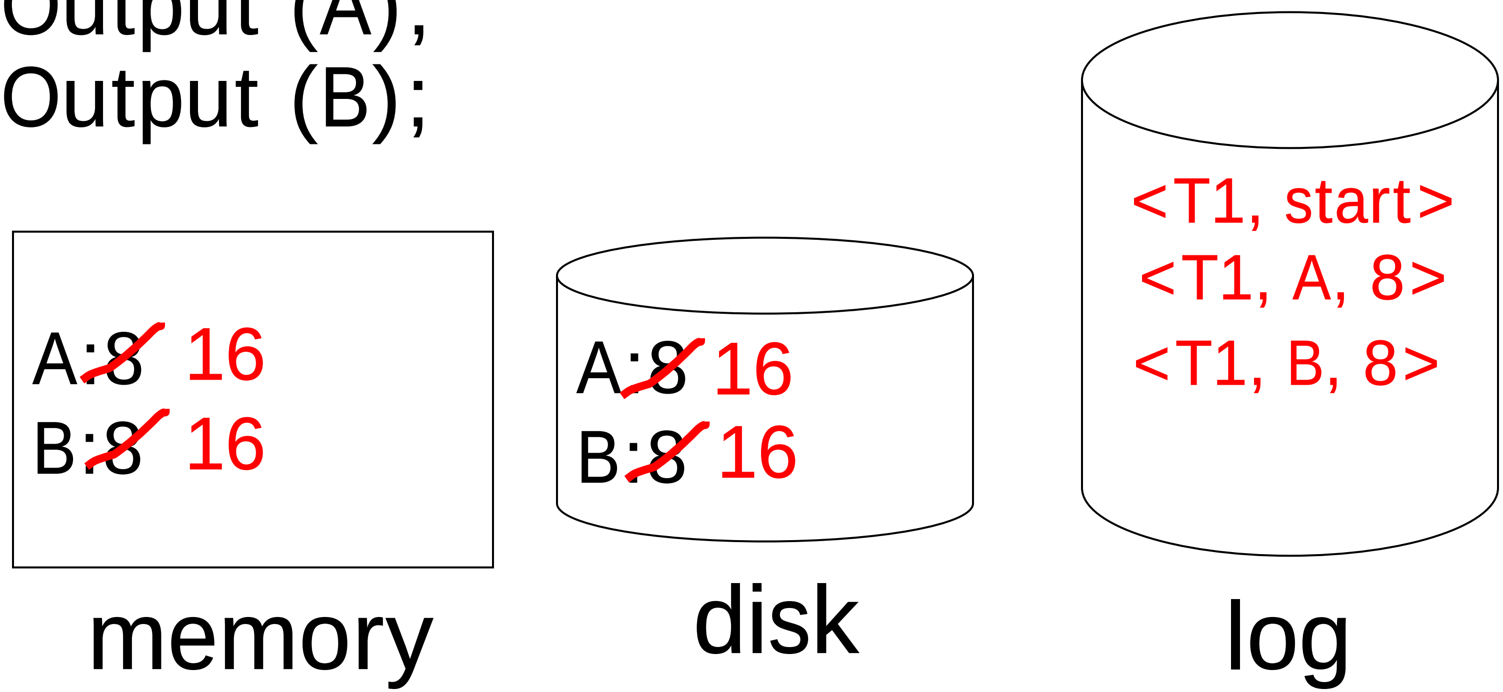
Undo logging (Immediate modification)

T_1: 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);



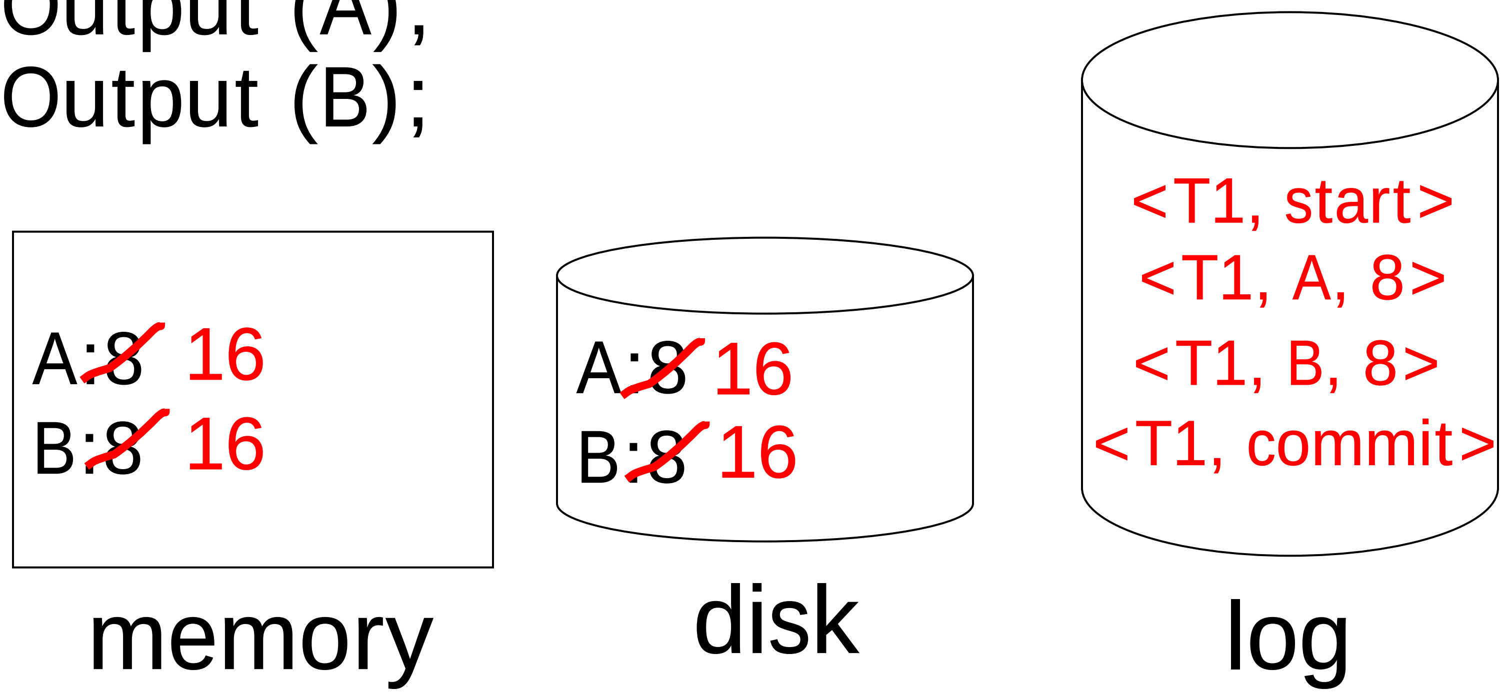
Undo logging (Immediate modification)

T1: 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);



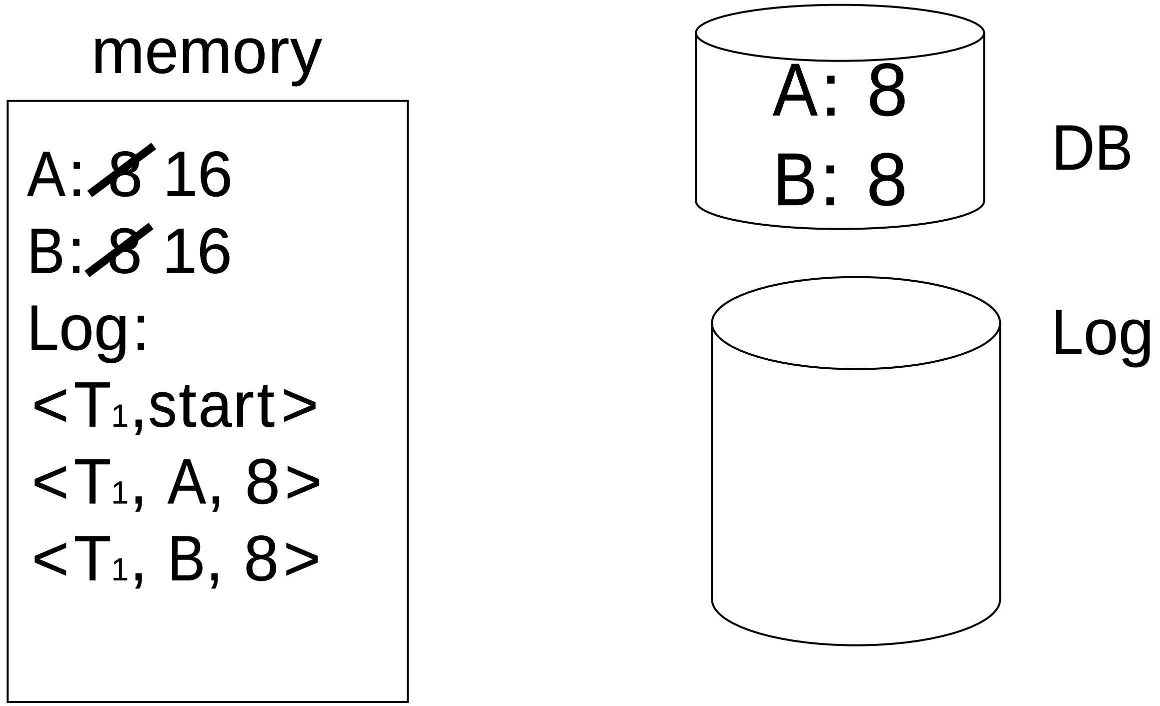
Undo logging (Immediate modification)

T1: Read (A,t); t ← t×2 A=B
Write (A,t);
Read (B,t); t ← t×2
Write (B,t);
Output (A);
Output (B);



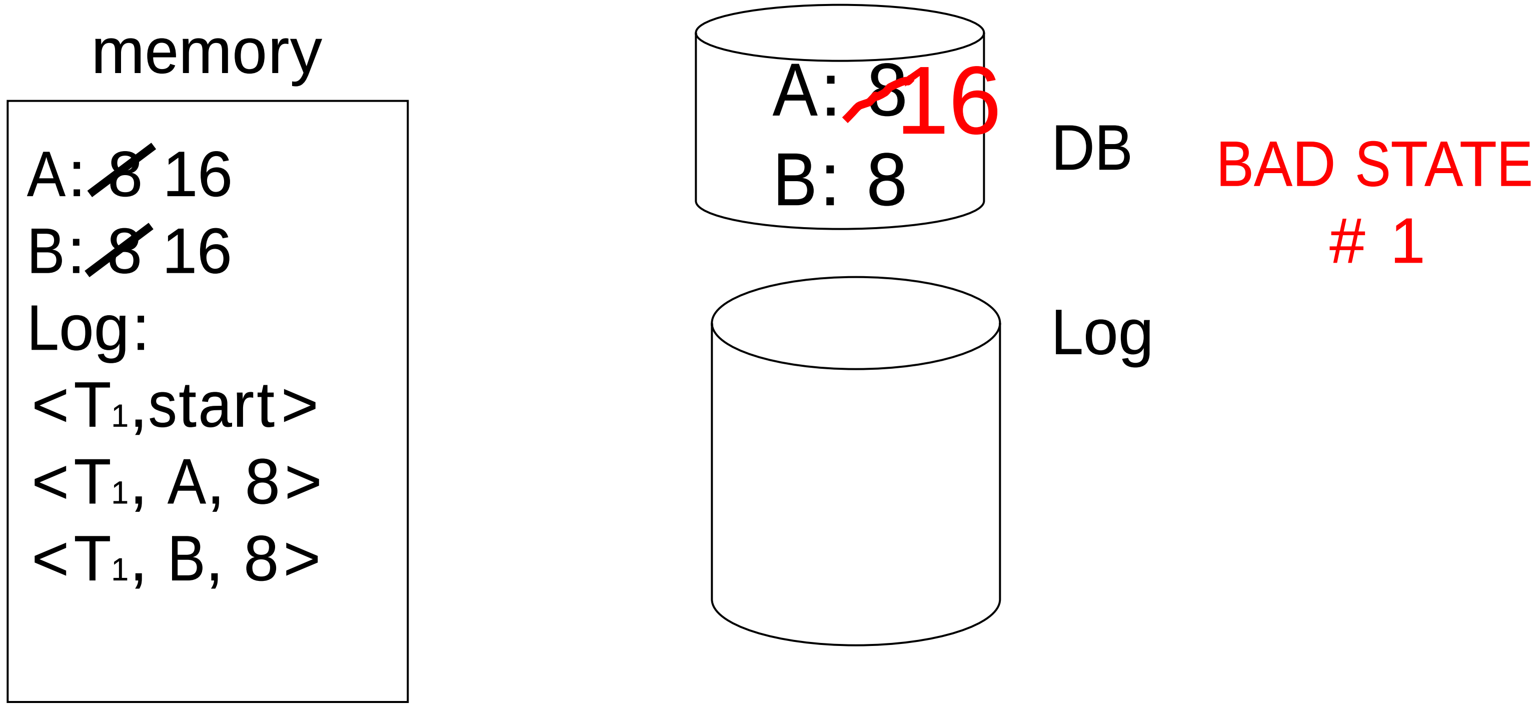
One “complication”

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



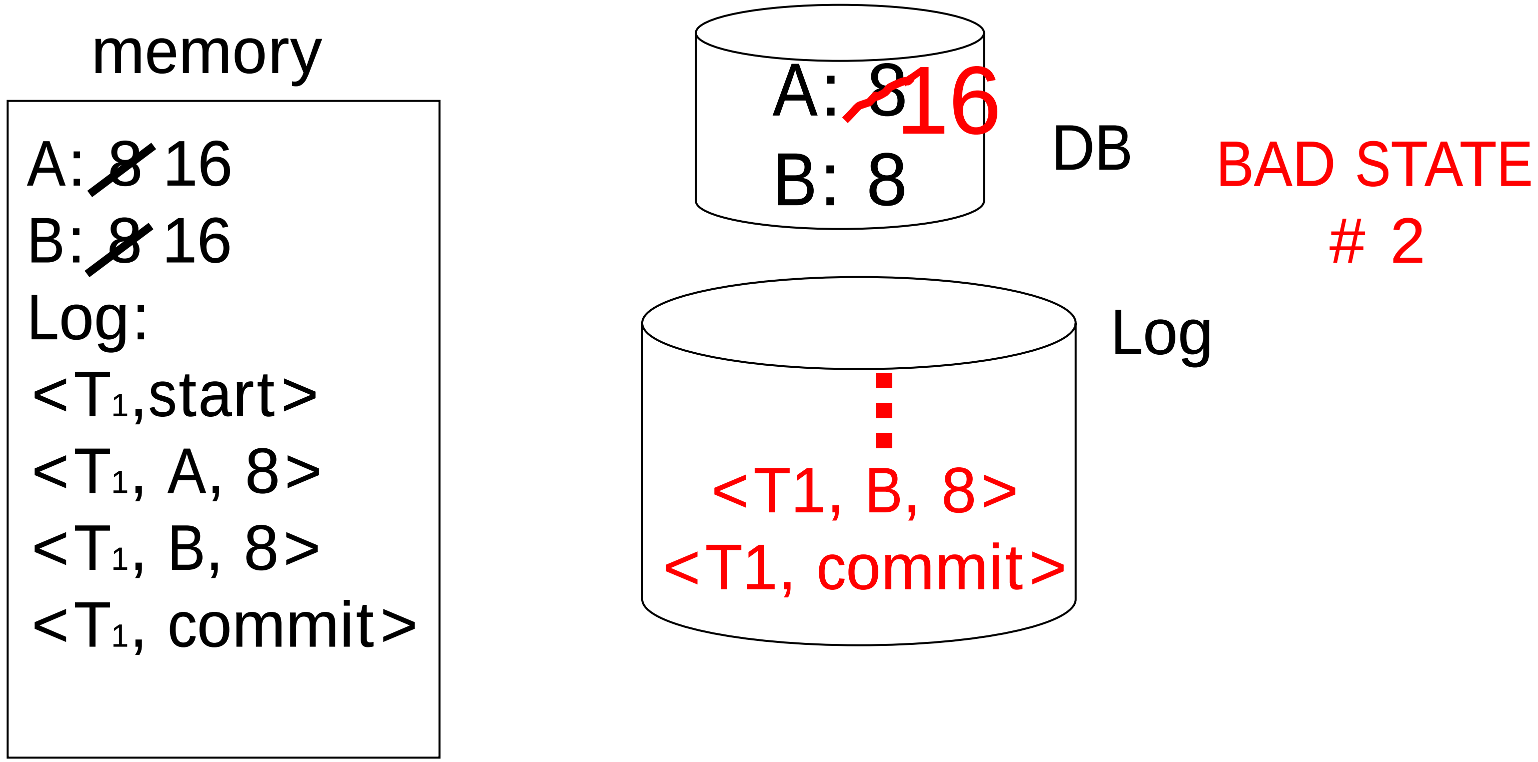
One “complication”

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



One "complication"

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



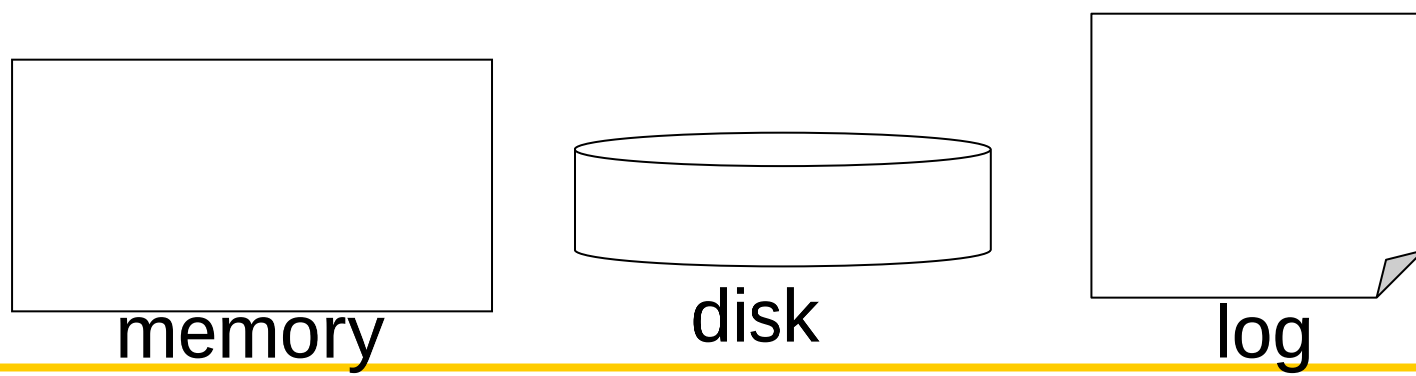
Undo Log Steps:

U_1:

If transaction T modifies X, then the log record <T,X,v> must be written to disk before the new value of X is written to disk

U2:

If a transaction commits, then its COMMIT log record must be written to disk only after database record written to disk.



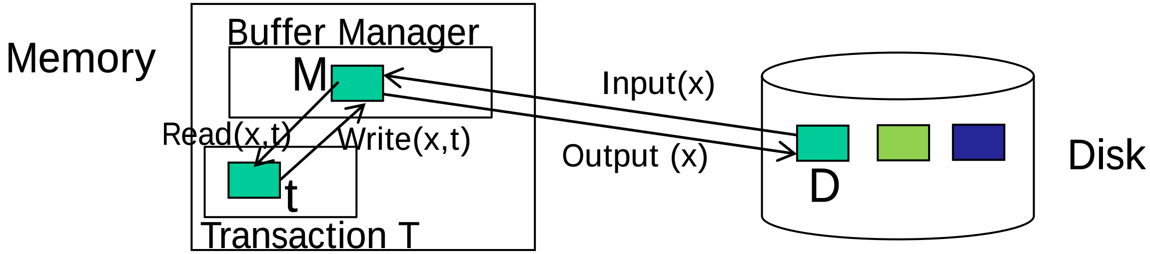
Undo Log Example:

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

U1: If transaction T modifies X, then the log record <T,X,v> must be written to disk before the new value of X is written to disk

U2: If a transaction commits, then its COMMIT log record must be written to disk only after database record written to 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>
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>
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						



Undo 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>
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>
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						

Recovery rules: Undo logging

1) If T is a transaction whose COMMIT record has been seen, then do nothing. (T is committed and must not be undone)

2) Otherwise, T is an incomplete transaction, or an aborted transaction. The recovery manager change the value of X in the database to v.

Undo 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>
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>
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						

Undo Log Example

<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>

<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>
<ABORT T3>

Undo Log: how far to recover?

Real Problem!

Undo log file could contain Mn of records/lines

Need to check all!!

Undo Log: how far to recover?

Solution: insert checkpoints in log file

How it works?

1) Stop accepting new transactions
2) Wait until all running transactions commit
3) Flush the log
4) Write a log <CKPT>
5) Resume accepting transactions

Undo Log with CheckPoint Example

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