

Database System Internals

Transactions: Recovery (part 1)

Paul G. Allen School of Computer Science and Engineering
University of Washington, Seattle

Main textbook (Garcia-Molina)

- Ch. 17.2-4, 18.1-3, 18.8-9

Second textbook (Ramakrishnan)

- Ch. 16-18

Also: M. J. Franklin. Concurrency Control and Recovery. The Handbook of Computer Science and Engineering, A. Tucker, ed., CRC Press, Boca Raton, 1997.

Transaction Management

Two parts:

- Concurrency control: ACID
- Recovery from crashes: ACID

We already discussed concurrency control

You are implementing locking in lab3

Today, we start recovery

Type of Crash	Prevention
Wrong data entry	Constraints and Data cleaning
Disk crashes	Redundancy: e.g. RAID, archive
Data center failures	Remote backups or replicas
System failures: e.g. power	DATABASE RECOVERY

System Crash

Client 1:
BEGIN TRANSACTION
UPDATE Account1
SET balance= balance – 500

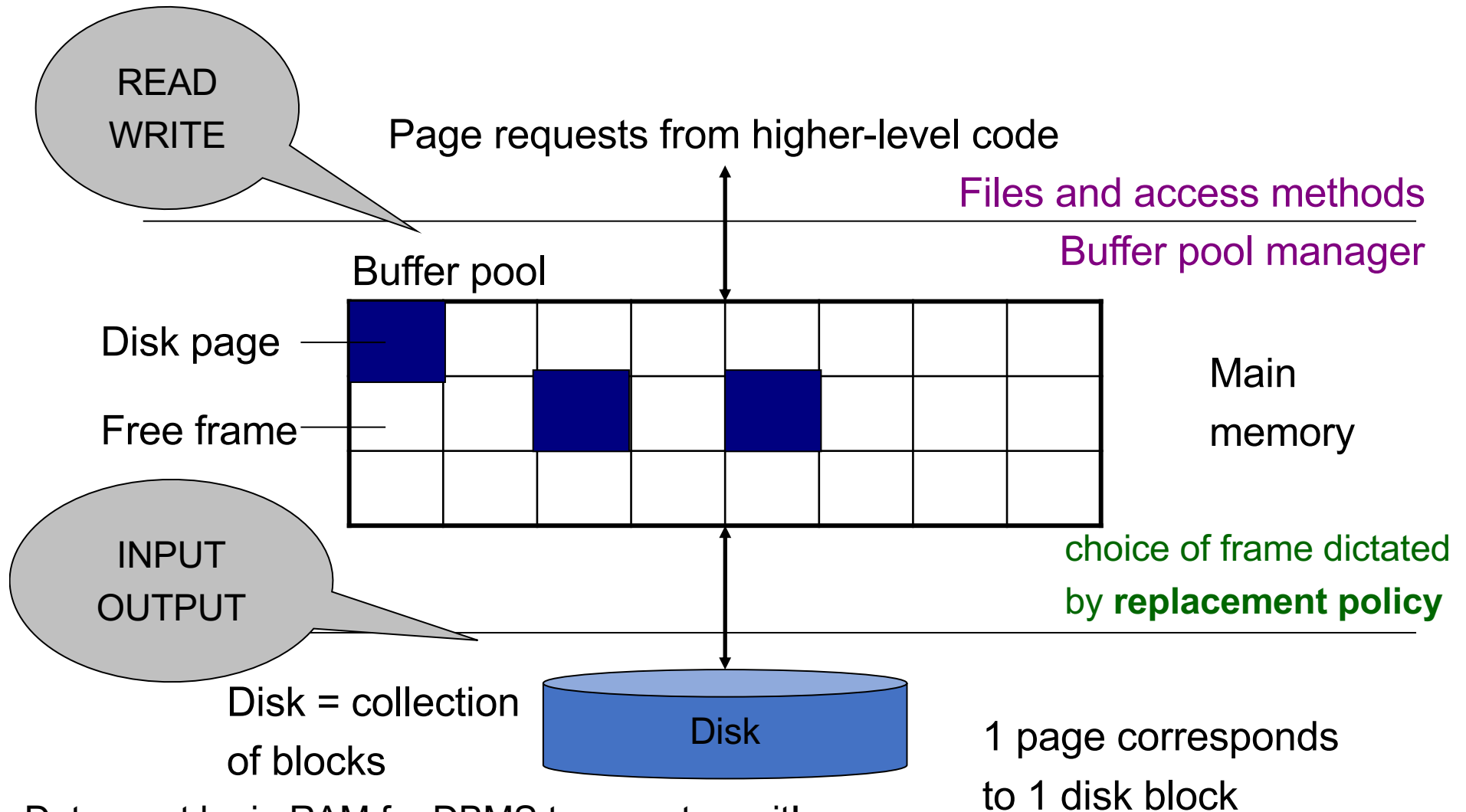
UPDATE Account2
SET balance = balance + 500
COMMIT



System Failures

- Each transaction has *internal state*
- When system crashes, internal state is lost
 - Don't know which parts executed and which didn't
 - Need ability to *undo* and *redo*

Buffer Manager Review



Data must be in RAM for DBMS to operate on it!

Buffer pool = table of <frame#, pageid> pairs

Buffer Manager Review

- Enables higher layers of the DBMS to assume that needed data is in main memory
- Caches data in memory. Problems when crash occurs:
 1. If committed data was not yet written to disk
 2. If uncommitted data was flushed to disk

Transactions

- Assumption: the database is composed of *elements*.
- 1 element can be either:
 - 1 page = physical logging
 - 1 record = logical logging
- In Lab 4 we use page-level elements

Primitive Operations of Transactions

- **READ(X,t)**
 - copy element X to transaction local variable t
- **WRITE(X,t)**
 - copy transaction local variable t to element X
- **INPUT(X)**
 - read element X to memory buffer
- **OUTPUT(X)**
 - write element X to disk

Running Example

```
BEGIN TRANSACTION
READ(A,t);
t := t*2;
WRITE(A,t);
READ(B,t);
t := t*2;
WRITE(B,t)
COMMIT;
```

Initially, $A=B=8$.

Atomicity requires that either
(1) T commits and $A=B=16$, or
(2) T does not commit and $A=B=8$.

Running Example

```
BEGIN TRANSACTION
```

```
READ(A,t);
```

```
t := t*2;
```

```
WRITE(A,t);
```

```
READ(B,t);
```

```
t := t*2;
```

```
WR
```

```
CO
```

Initially, $A=B=8$.

Atomicity requires that either
(1) T commits and $A=B=16$, or
(2) T does not commit and $A=B=8$.

Will look at various crash scenarios

What behavior do we want in each case?

READ(A,t); t := t*2; WRITE(A,t);
 READ(B,t); t := t*2; WRITE(B,t)

Transaction

Buffer pool

Disk

Action	t	Mem A	Mem B	Disk A	Disk B
INPUT(A)		8		8	8
READ(A,t)					
t:=t*2					
WRITE(A,t)					
INPUT(B)					
READ(B,t)					
t:=t*2					
WRITE(B,t)					
OUTPUT(A)					
OUTPUT(B)					
COMMIT					

READ(A,t); t := t*2; WRITE(A,t);
 READ(B,t); t := t*2; WRITE(B,t)

Transaction

Buffer pool

Disk

Action	t	Mem A	Mem B	Disk A	Disk B
INPUT(A)		8		8	8
READ(A,t)	8	8		8	8
t:=t*2					
WRITE(A,t)					
INPUT(B)					
READ(B,t)					
t:=t*2					
WRITE(B,t)					
OUTPUT(A)					
OUTPUT(B)					
COMMIT					

READ(A,t); t := t*2; WRITE(A,t);
 READ(B,t); t := t*2; WRITE(B,t)

Transaction

Buffer pool

Disk

Action	t	Mem A	Mem B	Disk A	Disk B
INPUT(A)		8		8	8
READ(A,t)	8	8		8	8
t:=t*2	16	8		8	8
WRITE(A,t)					
INPUT(B)					
READ(B,t)					
t:=t*2					
WRITE(B,t)					
OUTPUT(A)					
OUTPUT(B)					
COMMIT					

READ(A,t); t := t*2; WRITE(A,t);
 READ(B,t); t := t*2; WRITE(B,t)

Transaction

Buffer pool

Disk

Action	t	Mem A	Mem B	Disk A	Disk B
INPUT(A)		8		8	8
READ(A,t)	8	8		8	8
t:=t*2	16	8		8	8
WRITE(A,t)	16	16		8	8
INPUT(B)					
READ(B,t)					
t:=t*2					
WRITE(B,t)					
OUTPUT(A)					
OUTPUT(B)					
COMMIT					

READ(A,t); t := t*2; WRITE(A,t);
 READ(B,t); t := t*2; WRITE(B,t)

Transaction

Buffer pool

Disk

Action	t	Mem A	Mem B	Disk A	Disk B
INPUT(A)		8		8	8
READ(A,t)	8	8		8	8
t:=t*2	16	8		8	8
WRITE(A,t)	16	16		8	8
INPUT(B)	16	16	8	8	8
READ(B,t)					
t:=t*2					
WRITE(B,t)					
OUTPUT(A)					
OUTPUT(B)					
COMMIT					

READ(A,t); t := t*2; WRITE(A,t);
 READ(B,t); t := t*2; WRITE(B,t)

Transaction

Buffer pool

Disk

Action	t	Mem A	Mem B	Disk A	Disk B
INPUT(A)		8		8	8
READ(A,t)	8	8		8	8
t:=t*2	16	8		8	8
WRITE(A,t)	16	16		8	8
INPUT(B)	16	16	8	8	8
READ(B,t)	8	16	8	8	8
t:=t*2					
WRITE(B,t)					
OUTPUT(A)					
OUTPUT(B)					
COMMIT					

READ(A,t); t := t*2; WRITE(A,t);
 READ(B,t); t := t*2; WRITE(B,t)

Transaction Buffer pool Disk

Action	t	Mem A	Mem B	Disk A	Disk B
INPUT(A)		8		8	8
READ(A,t)	8	8		8	8
t:=t*2	16	8		8	8
WRITE(A,t)	16	16		8	8
INPUT(B)	16	16	8	8	8
READ(B,t)	8	16	8	8	8
t:=t*2	16	16	8	8	8
WRITE(B,t)					
OUTPUT(A)					
OUTPUT(B)					
COMMIT					

READ(A,t); t := t*2; WRITE(A,t);
 READ(B,t); t := t*2; WRITE(B,t)

Transaction

Buffer pool

Disk

Action	t	Mem A	Mem B	Disk A	Disk B
INPUT(A)		8		8	8
READ(A,t)	8	8		8	8
t:=t*2	16	8		8	8
WRITE(A,t)	16	16		8	8
INPUT(B)	16	16	8	8	8
READ(B,t)	8	16	8	8	8
t:=t*2	16	16	8	8	8
WRITE(B,t)	16	16	16	8	8
OUTPUT(A)					
OUTPUT(B)					
COMMIT					

READ(A,t); t := t*2; WRITE(A,t);
 READ(B,t); t := t*2; WRITE(B,t)

Transaction

Buffer pool

Disk

Action	t	Mem A	Mem B	Disk A	Disk B
INPUT(A)		8		8	8
READ(A,t)	8	8		8	8
t:=t*2	16	8		8	8
WRITE(A,t)	16	16		8	8
INPUT(B)	16	16	8	8	8
READ(B,t)	8	16	8	8	8
t:=t*2	16	16	8	8	8
WRITE(B,t)	16	16	16	8	8
OUTPUT(A)	16	16	16	16	8
OUTPUT(B)					
COMMIT					

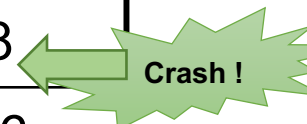
READ(A,t); t := t*2; WRITE(A,t);
 READ(B,t); t := t*2; WRITE(B,t)

Transaction Buffer pool Disk

Action	t	Mem A	Mem B	Disk A	Disk B
INPUT(A)		8		8	8
READ(A,t)	8	8		8	8
t:=t*2	16	8		8	8
WRITE(A,t)	16	16		8	8
INPUT(B)	16	16	8	8	8
READ(B,t)	8	16	8	8	8
t:=t*2	16	16	8	8	8
WRITE(B,t)	16	16	16	8	8
OUTPUT(A)	16	16	16	16	8
OUTPUT(B)	16	16	16	16	16
COMMIT					

Is this bad ?

Action	t	Mem A	Mem B	Disk A	Disk B
INPUT(A)		8		8	8
READ(A,t)	8	8		8	8
t:=t*2	16	8		8	8
WRITE(A,t)	16	16		8	8
INPUT(B)	16	16	8	8	8
READ(B,t)	8	16	8	8	8
t:=t*2	16	16	8	8	8
WRITE(B,t)	16	16	16	8	8
OUTPUT(A)	16	16	16	16	8
OUTPUT(B)	16	16	16	16	16
COMMIT					



Is this bad ?

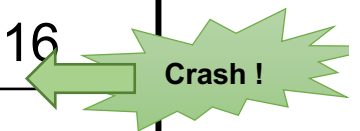
Yes it's bad: A=16, B=8....

Action	t	Mem A	Mem B	Disk A	Disk B
INPUT(A)		8		8	8
READ(A,t)	8	8		8	8
t:=t*2	16	8		8	8
WRITE(A,t)	16	16		8	8
INPUT(B)	16	16	8	8	8
READ(B,t)	8	16	8	8	8
t:=t*2	16	16	8	8	8
WRITE(B,t)	16	16	16	8	8
OUTPUT(A)	16	16	16	16	8
OUTPUT(B)	16	16	16	16	16
COMMIT					

Crash !

Is this bad ?

Action	t	Mem A	Mem B	Disk A	Disk B
INPUT(A)		8		8	8
READ(A,t)	8	8		8	8
t:=t*2	16	8		8	8
WRITE(A,t)	16	16		8	8
INPUT(B)	16	16	8	8	8
READ(B,t)	8	16	8	8	8
t:=t*2	16	16	8	8	8
WRITE(B,t)	16	16	16	8	8
OUTPUT(A)	16	16	16	16	8
OUTPUT(B)	16	16	16	16	16
COMMIT					



Is this bad ?

Yes it's bad: A=B=16, but not committed

Action	t	Mem A	Mem B	Disk A	Disk B
INPUT(A)		8		8	8
READ(A,t)	8	8		8	8
t:=t*2	16	8		8	8
WRITE(A,t)	16	16		8	8
INPUT(B)	16	16	8	8	8
READ(B,t)	8	16	8	8	8
t:=t*2	16	16	8	8	8
WRITE(B,t)	16	16	16	8	8
OUTPUT(A)	16	16	16	16	8
OUTPUT(B)	16	16	16	16	16
COMMIT					

Crash !

Is this bad ?

Action	t	Mem A	Mem B	Disk A	Disk B
INPUT(A)		8		8	8
READ(A,t)	8	8		8	8
t:=t*2	16	8		8	8
WRITE(A,t)	16	16		8	8
INPUT(B)	16	16	8	8	8
READ(B,t)	8	16	8	8	8
t:=t*2	16	16	8	8	8
WRITE(B,t)	16	16	16	8	8
OUTPUT(A)	16	16	16	16	8
OUTPUT(B)	16	16	16	16	16
COMMIT					

Crash !

Is this bad ?

No: that's OK

Action	t	Mem A	Mem B	Disk A	Disk B
INPUT(A)		8		8	8
READ(A,t)	8	8		8	8
t:=t*2	16	8		8	8
WRITE(A,t)	16	16		8	8
INPUT(B)	16	16	8	8	8
READ(B,t)	8	16	8	8	8
t:=t*2	16	16	8	8	8
WRITE(B,t)	16	16	16	8	8
OUTPUT(A)	16	16	16	16	8
OUTPUT(B)	16	16	16	16	16
COMMIT					



OUTPUT can also happen **after** COMMIT (details coming)

Action	t	Mem A	Mem B	Disk A	Disk B
INPUT(A)		8		8	8
READ(A,t)	8	8		8	8
t:=t*2	16	8		8	8
WRITE(A,t)	16	16		8	8
INPUT(B)	16	16	8	8	8
READ(B,t)	8	16	8	8	8
t:=t*2	16	16	8	8	8
WRITE(B,t)	16	16	16	8	8
COMMIT					
OUTPUT(A)	16	16	16	16	8
OUTPUT(B)	16	16	16	16	16

OUTPUT can also happen **after** COMMIT (details coming)

Action	t	Mem A	Mem B	Disk A	Disk B
INPUT(A)		8		8	8
READ(A,t)	8	8		8	8
t:=t*2	16	8		8	8
WRITE(A,t)	16	16		8	8
INPUT(B)	16	16	8	8	8
READ(B,t)	8	16	8	8	8
t:=t*2	16	16	8	8	8
WRITE(B,t)	16	16	16	8	8
COMMIT					
OUTPUT(A)	16	16	16	16	8
OUTPUT(B)	16	16	16	16	16



Atomic Transactions

- **FORCE or NO-FORCE**

- Should all updates of a transaction be forced to disk before the transaction commits?

- **STEAL or NO-STEAL**

- Can an update made by an uncommitted transaction overwrite the most recent committed value of a data item on disk?

Force/No-steal (most strict)

- **FORCE**: Pages of committed transactions must be forced to disk before commit
- **NO-STEAL**: Pages of uncommitted transactions cannot be written to disk

Easy to implement (how?) and ensures atomicity

No-Force/Steal (least strict)

- **NO-FORCE**: Pages of committed transactions need not be written to disk
- **STEAL**: Pages of uncommitted transactions may be written to disk

In both cases, need a Write Ahead Log (WAL) to provide atomicity in face of failures

Write-Ahead Log (WAL)

The Log: append-only file containing log records

- Records every single action of every TXN
- Forces log entries to disk as needed
- After a system crash, use log to recover

Three types: UNDO, REDO, UNDO-REDO

Aries: is an UNDO-REDO log

Policies and Logs

	NO-STEAL	STEAL
FORCE	Lab 3	Undo Log
NO-FORCE	Redo Log	Undo-Redo Log

“UNDO” Log


FORCE and STEAL

Undo Logging


Log records

- **<START T>**
 - transaction T has begun
- **<COMMIT T>**
 - T has committed
- **<ABORT T>**
 - T has aborted
- **<T,X,v>**
 - T has updated element X, and its old value was v
 - *Idempotent, physical* log records

Action	t	Mem A	Mem B	Disk A	Disk B	UNDO Log
						<START T>
INPUT(A)		8		8	8	
READ(A,t)	8	8		8	8	
t:=t*2	16	8		8	8	
WRITE(A,t)	16	16		8	8	<T,A,8>
INPUT(B)	16	16	8	8	8	
READ(B,t)	8	16	8	8	8	
t:=t*2	16	16	8	8	8	
WRITE(B,t)	16	16	16	8	8	<T,B,8>
OUTPUT(A)	16	16	16	16	8	
OUTPUT(B)	16	16	16	16	16	
COMMIT						<COMMIT T>

Action	t	Mem A	Mem B	Disk A	Disk B	UNDO Log
						<START T>
INPUT(A)		8		8	8	
READ(A,t)	8	8		8	8	
t:=t*2	16	8		8	8	
WRITE(A,t)	16	16		8	8	<T,A,8>
INPUT(B)	16	16	8	8	8	
READ(B,t)	8	16	8	8	8	
t:=t*2	16	16	8	8	8	
WRITE(B,t)	16	16	16	8	8	<T,B,8>
OUTPUT(A)	16	16	16	16	8	
OUTPUT(B)	16	16	16	16	16	
COMMIT						<COMMIT T>

WHAT DO WE DO ?

Action	t	Mem A	Mem B	Disk A	Disk B	UNDO Log
						<START T>
INPUT(A)		8		8	8	
READ(A,t)	8	8		8	8	
t:=t*2	16	8		8	8	
WRITE(A,t)	16	16		8	8	<T,A,8>
INPUT(B)	16	16	8	8	8	
READ(B,t)	8	16	8	8	8	
t:=t*2	16	16	8	8	8	
WRITE(B,t)	16	16	16	8	8	<T,B,8>
OUTPUT(A)	16	16	16	16	8	
OUTPUT(B)	16	16	16	16	16	 Crash !
COMMIT						<COMMIT T>

WHAT DO WE DO ?

We UNDO by setting B=8 and A=8

Action	t	Mem A	Mem B	Disk A	Disk B	UNDO Log
						<START T>
INPUT(A)		8		8	8	
READ(A,t)	8	8		8	8	
t:=t*2	16	8		8	8	
WRITE(A,t)	16	16		8	8	<T,A,8>
INPUT(B)	16	16	8	8	8	
READ(B,t)	8	16	8	8	8	
t:=t*2	16	16	8	8	8	
WRITE(B,t)	16	16	16	8	8	<T,B,8>
OUTPUT(A)	16	16	16	16	8	
OUTPUT(B)	16	16	16	16	16	
COMMIT						<COMMIT T>

What do we do now ?

Crash !

Action	t	Mem A	Mem B	Disk A	Disk B	UNDO Log
						<START T>
INPUT(A)		8		8	8	
READ(A,t)	8	8		8	8	
t:=t*2	16	8		8	8	
WRITE(A,t)	16	16		8	8	<T,A,8>
INPUT(B)	16	16	8	8	8	
READ(B,t)	8	16	8	8	8	
t:=t*2	16	16	8	8	8	
WRITE(B,t)	16	16	16	8	8	<T,B,8>
OUTPUT(A)	16	16	16	16	8	
OUTPUT(B)	16	16	16	16	16	
COMMIT						<COMMIT T>

What do we do now ?

Nothing: log contains COMMIT

After Crash

- This is all we see (for example):

Disk A	Disk B
8	16

<START T>
<T,A,8>
<T,B,8>

After Crash

- This is all we see (for example):

Disk A	Disk B
8	16

```
<START T>  
<T,A,8>  
<T,B,8>
```

After Crash

- This is all we see (for example):
- Need to step through the log

Disk A	Disk B
8	16

```
<START T>  
<T,A,8>  
<T,B,8>
```

After Crash

- This is all we see (for example):
- Need to step through the log

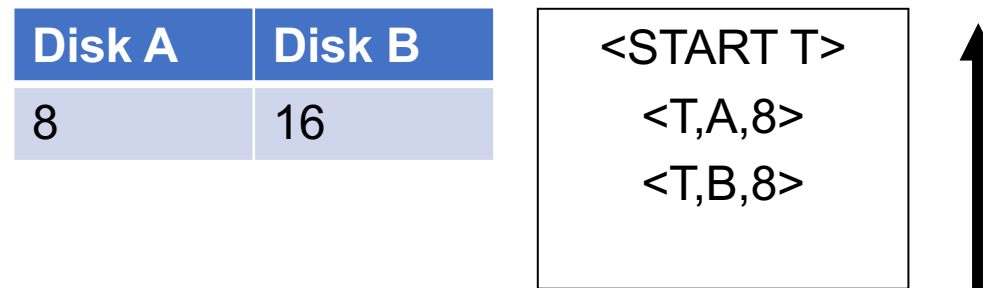
Disk A	Disk B
8	16

```
<START T>  
<T,A,8>  
<T,B,8>
```

- What direction?

After Crash

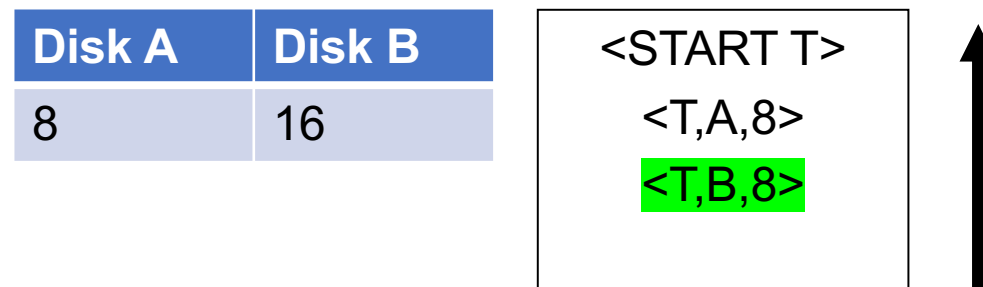
- This is all we see (for example):
- Need to step through the log



- What direction?
- In UNDO log, we start at the most recent and go backwards in time

After Crash

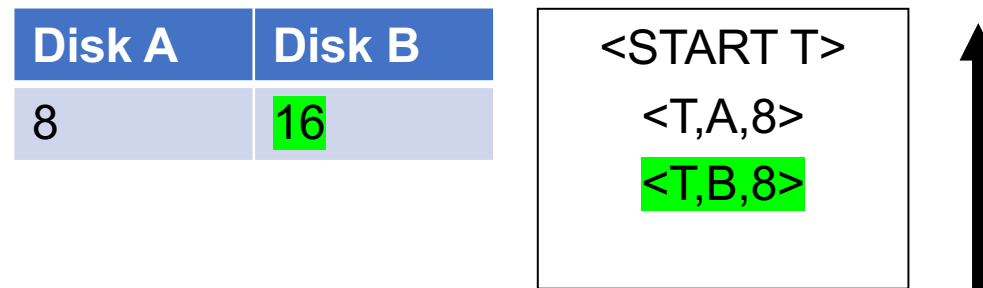
- This is all we see (for example):
- Need to step through the log



- What direction?
- In UNDO log, we start at the most recent and go backwards in time

After Crash

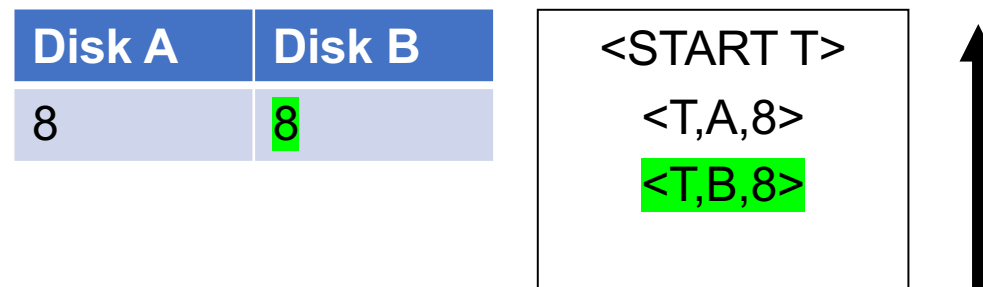
- This is all we see (for example):
- Need to step through the log



- What direction?
- In UNDO log, we start at the most recent and go backwards in time

After Crash

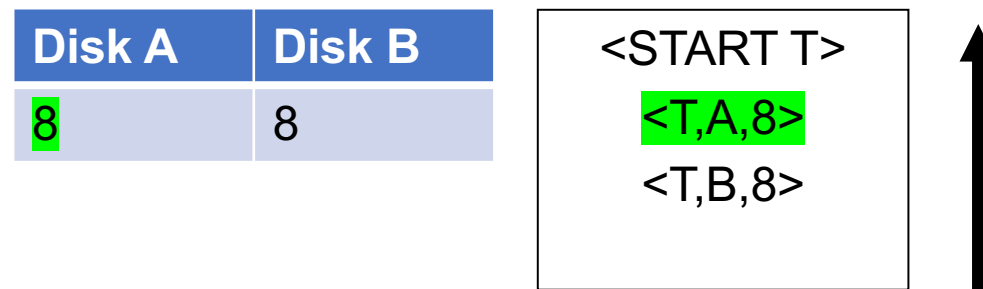
- This is all we see (for example):
- Need to step through the log



- What direction?
- In UNDO log, we start at the most recent and go backwards in time

After Crash

- This is all we see (for example):
- Need to step through the log



- What direction?
- In UNDO log, we start at the most recent and go backwards in time

After Crash

- If we see NO Commit statement:
 - We UNDO both changes: $A=8$, $B=8$
 - The transaction is atomic, since none of its actions have been executed
- In we see that T has a Commit statement
 - We don't undo anything
 - The transaction is atomic, since both it's actions have been executed

Recovery with Undo Log

After system's crash, run recovery manager

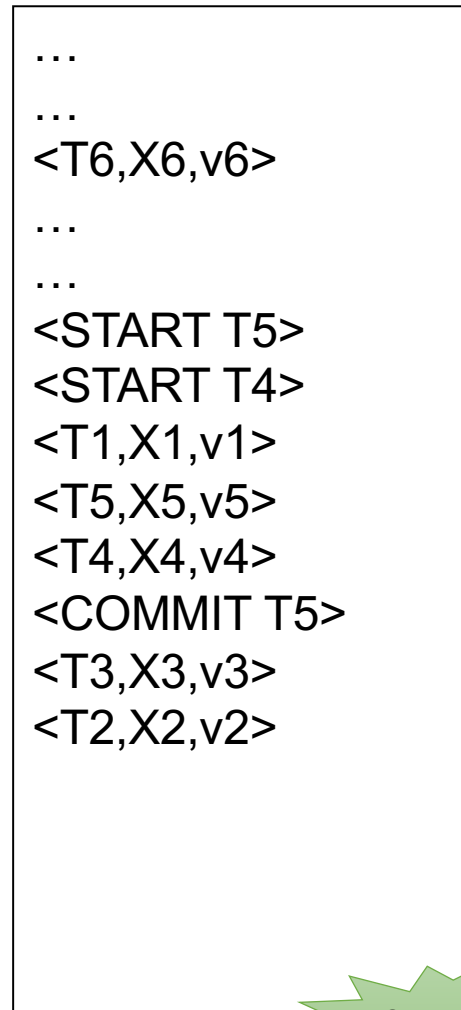
- Decide for each transaction T whether it is completed or not
 - <START T>....<COMMIT T>.... = yes
 - <START T>....<ABORT T>..... = yes
 - <START T>..... = no
- Undo all modifications by **incomplete** transactions

Recovery with Undo Log

Recovery manager:

- Read log from the end; cases:
 - <COMMIT T>: mark T as completed
 - <ABORT T>: mark T as completed
 - <T,X,v>: if T is not completed
then write $X=v$ to disk
else ignore
 - <START T>: ignore

Recovery with Undo Log



Question 1: Which updates are undone ?

Question 2: How far back do we need to read in the log ?

Question 3: What happens if second crash during recovery?

Recovery with Undo Log

...

...

<T6,X6,v6>

...

...

<START T5>

<START T4>

<T1,X1,v1>

<T5,X5,v5>

<T4,X4,v4>

<COMMIT T5>

<T3,X3,v3>

<T2,X2,v2>



Question 1: Which updates are undone ?

Question 2:
How far back do we need to read in the log ?
To the beginning.

Question 3:
What happens if second crash during recovery?

Recovery with Undo Log

...

...

<T6,X6,v6>

...

...

<START T5>

<START T4>

<T1,X1,v1>


<T5,X5,v5>

<T4,X4,v4>

<COMMIT T5>

<T3,X3,v3>

<T2,X2,v2>



Question 1: Which updates are undone ?

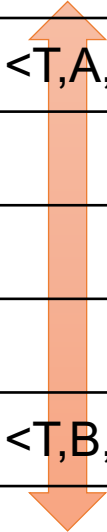
Question 2:
How far back do we need to read in the log ?
To the beginning.

Question 3:
What happens if second crash during recovery?
No problem! Log records are idempotent. Can reapply.

Crash !

Action	t	Mem A	Mem B	Disk A	Disk B	UNDO Log
						<START T>
INPUT(A)		8			8	
READ(A,t)	8	8			8	
t:=t*2	16	8			8	
WRITE(A,t)	16	16		8	8	<T,A,8>
INPUT(B)	16	16	8	8	8	
READ(B,t)	8	16	8	8	8	
t:=t*2	16	16	8	8	8	
WRITE(B,t)	16	16	16	8	8	<T,B,8>
OUTPUT(A)	16	16	16	16	8	
OUTPUT(B)	16	16	16	16	16	
COMMIT						<COMMIT T>

When must we force pages to disk ?



Action	t	Mem A	Mem B	Disk A	Disk B	UNDO Log
						<START T>
INPUT(A)		8		8	8	
READ(A,t)	8	8		8	8	
t:=t*2	16	8		8	8	
WRITE(A,t)	16	16		8	8	<T,A,8>
INPUT(B)	16	16	8	8	8	
READ(B,t)	8	16	8	8	8	
t:=t*2	16	16	8	8	8	
WRITE(B,t)	16	16	16	8	8	<T,B,8>
OUTPUT(A)	16	16	16	16	8	
OUTPUT(B)	16	16	16	16	16	
COMMIT						<COMMIT T>

RULES: log entry before OUTPUT before COMMIT

Undo-Logging Rules

U1: If T modifies X, then $\langle T, X, v \rangle$ must be written to disk before $\text{OUTPUT}(X)$

U2: If T commits, then $\text{OUTPUT}(X)$ must be written to disk before $\langle \text{COMMIT } T \rangle$

- Hence: OUTPUTs are done early, before the transaction commits



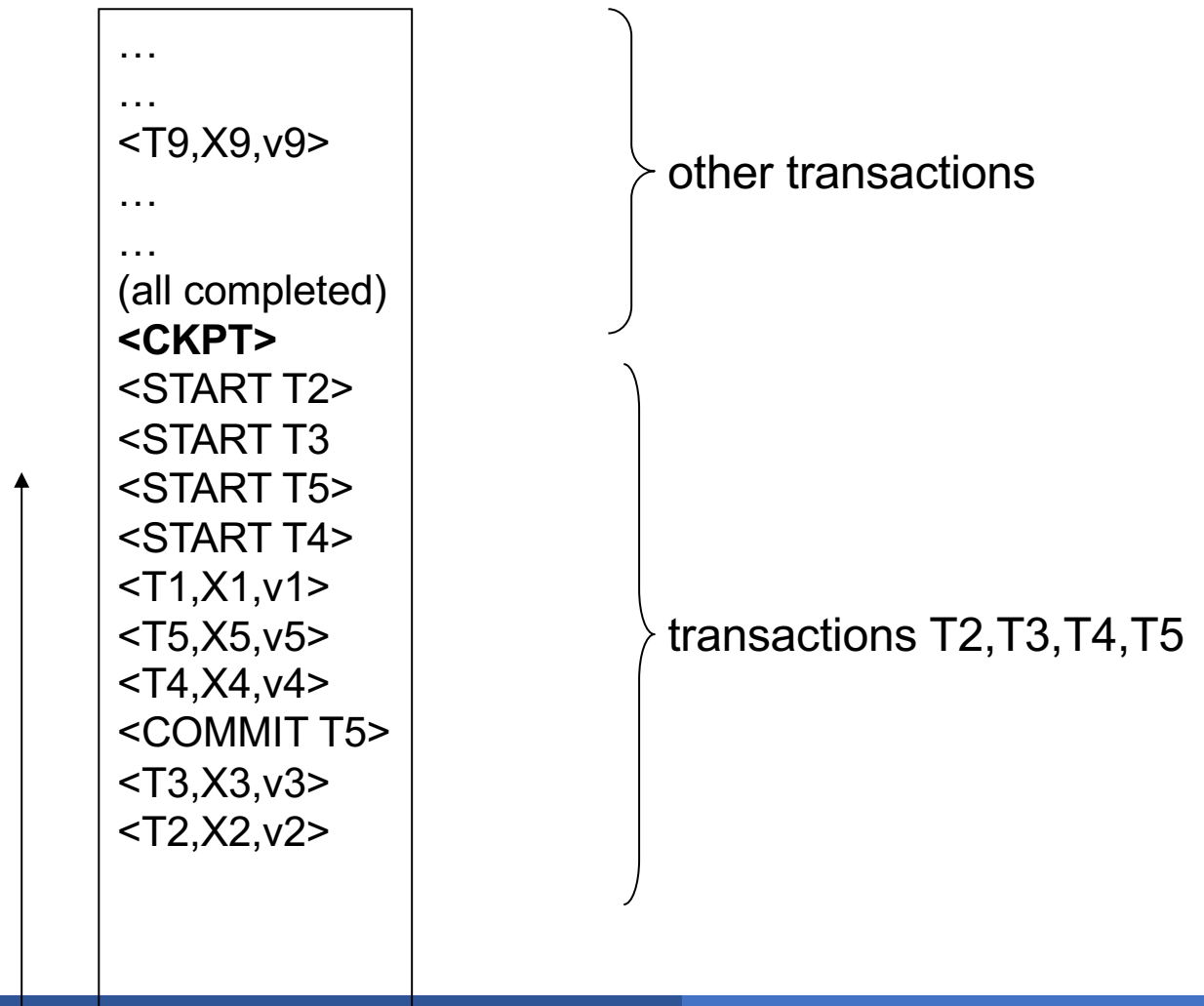
Checkpointing

Checkpoint the database periodically

- Stop accepting new transactions
- Wait until all current transactions complete
- Flush log to disk
- Write a <CKPT> log record, flush
- Resume transactions

Undo Recovery with Checkpointing

During recovery,
Can stop at first
<CKPT>



Nonquiescent Checkpointing

- Problem with checkpointing: database freezes during checkpoint
- Would like to checkpoint while database is operational
- Idea: nonquiescent checkpointing

Quiescent = being quiet, still, or at rest; inactive
Non-quiescent = allowing transactions to be active

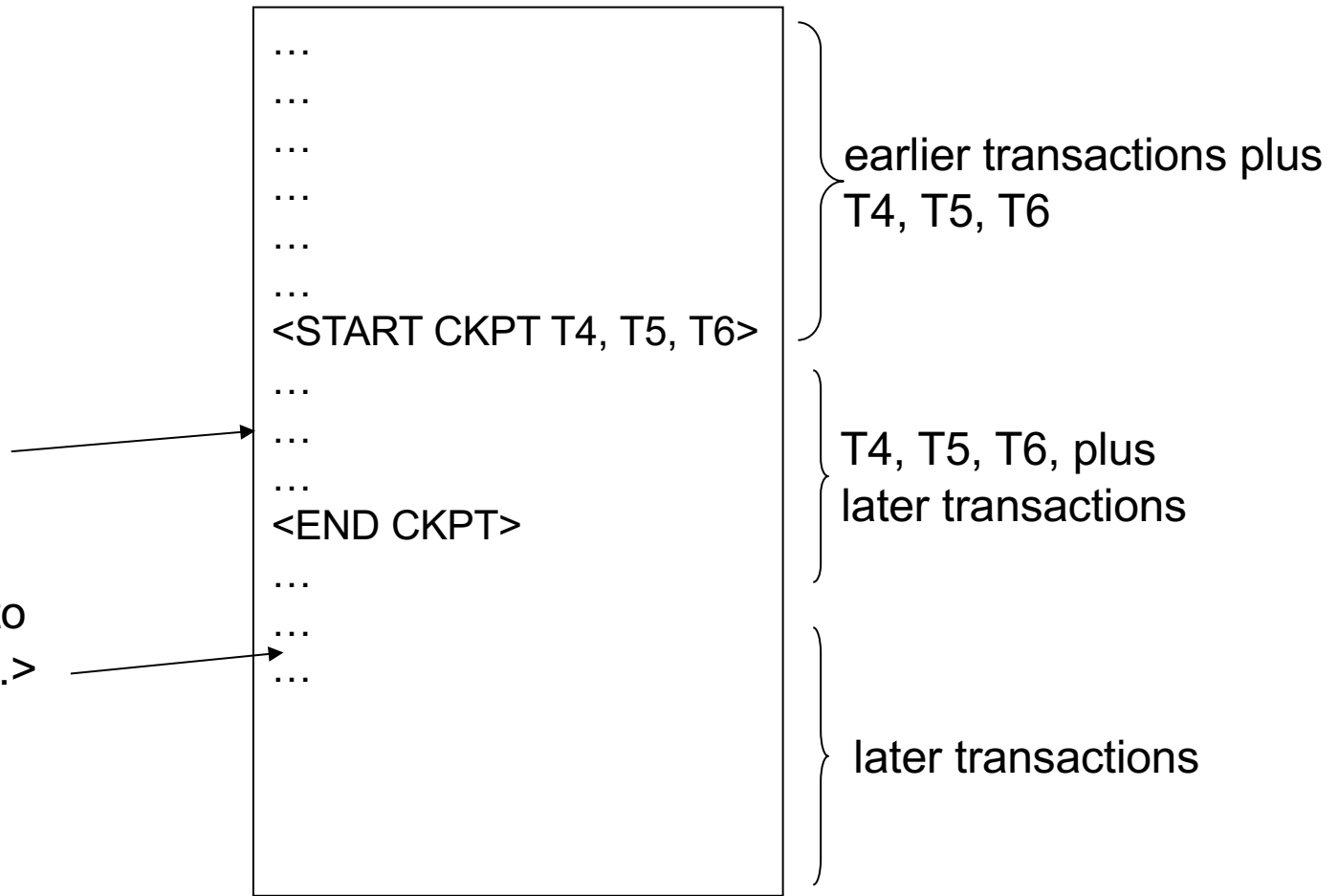
Nonquiescent Checkpointing

- Write a `<START CKPT(T1,...,Tk)>`
where T_1, \dots, T_k are all current active transactions.
Flush log to disk
- Continue normal operation
- When all of T_1, \dots, T_k have completed, write `<END CKPT>`, flush log to disk

Undo with Nonquiescent Checkpointing

If we crash here:
Need to read
Back to start of
T4, T5, T6

If we crash here:
Need to read only to
<START CKPT T4..>



Implementing ROLLBACK

- Recall: a transaction can end in COMMIT or ROLLBACK
- Idea: use the undo-log to implement ROLLBACK
- How ?
 - LSN = Log Sequence Number
 - Log entries for the same transaction are linked, using the LSN's
 - Read log in reverse, using LSN pointers

Implementing ROLLBACK

- Rec
- RO
- Ide
- How
-
-
-

```
...  
...  
<T9,X9,v9>  
...  
...  
(all completed)  
<CKPT>  
<START T2>  
<START T3  
<START T5>  
<START T4>  
<T1,X1,v1>  
<T5,X5,v5>  
<T2,X1,v2>  
<T4,X4,v4>  
<COMMIT T5>  
<T3,X3,v3>  
<T2,X2,v2>
```

CK
sing



REDO

NO-FORCE and NO-STEAL

Action	t	Mem A	Mem B	Disk A	Disk B
READ(A,t)	8	8		8	8
t:=t*2	16	8		8	8
WRITE(A,t)	16	16		8	8
READ(B,t)	8	16	8	8	8
t:=t*2	16	16	8	8	8
WRITE(B,t)	16	16	16	8	8
COMMIT					
OUTPUT(A)	16	16	16	16	8
OUTPUT(B)	16	16	16	16	16

Is this bad ?

Action	t	Mem A	Mem B	Disk A	Disk B
READ(A,t)	8	8		8	8
t:=t*2	16	8		8	8
WRITE(A,t)	16	16		8	8
READ(B,t)	8	16	8	8	8
t:=t*2	16	16	8	8	8
WRITE(B,t)	16	16	16	8	8
COMMIT					
OUTPUT(A)	16	16	16	16	8
OUTPUT(B)	16	16	16	16	16

Crash !

Is this bad ?

Yes, it's bad: A=16, B=8

Action	t	Mem A	Mem B	Disk A	Disk B
READ(A,t)	8	8		8	8
t:=t*2	16	8		8	8
WRITE(A,t)	16	16		8	8
READ(B,t)	8	16	8	8	8
t:=t*2	16	16	8	8	8
WRITE(B,t)	16	16	16	8	8
COMMIT					
OUTPUT(A)	16	16	16	16	8
OUTPUT(B)	16	16	16	16	16

Crash !

Is this bad ?

Action	t	Mem A	Mem B	Disk A	Disk B
READ(A,t)	8	8		8	8
t:=t*2	16	8		8	8
WRITE(A,t)	16	16		8	8
READ(B,t)	8	16	8	8	8
t:=t*2	16	16	8	8	8
WRITE(B,t)	16	16	16	8	8
COMMIT					
OUTPUT(A)	16	16	16	16	8
OUTPUT(B)	16	16	16	16	16

Crash !

Is this bad ?

Yes, it's bad: lost update

Action	t	Mem A	Mem B	Disk A	Disk B
READ(A,t)	8	8		8	8
t:=t*2	16	8		8	8
WRITE(A,t)	16	16		8	8
READ(B,t)	8	16	8	8	8
t:=t*2	16	16	8	8	8
WRITE(B,t)	16	16	16	8	8
COMMIT					
OUTPUT(A)	16	16	16	16	8
OUTPUT(B)	16	16	16	16	16



Crash !

Is this bad ?

Action	t	Mem A	Mem B	Disk A	Disk B
READ(A,t)	8	8		8	8
t:=t*2	16	8		8	8
WRITE(A,t)	16	16		8	8
READ(B,t)	8	16	8	8	8
t:=t*2	16	16	8	8	8
WRITE(B,t)	16	16	16	8	8
COMMIT					
OUTPUT(A)	16	16	16	16	8
OUTPUT(B)	16	16	16	16	16



Crash !

Is this bad ?

No: that's OK.

Action	t	Mem A	Mem B	Disk A	Disk B
READ(A,t)	8	8		8	8
t:=t*2	16	8		8	8
WRITE(A,t)	16	16		8	8
READ(B,t)	8	16	8	8	8
t:=t*2	16	16	8	8	8
WRITE(B,t)	16	16	16	8	8
COMMIT					
OUTPUT(A)	16	16	16	16	8
OUTPUT(B)	16	16	16	16	16



Crash !