CS 6400 A

Database Systems Concepts and Design

Lecture 21

11/10/25

Announcements

- Assignment 3 will be released on Wednesday
  - Due Nov 24

- What's remaining:
  - Dec 1: Final project report and code
  - Dec 5: Final exam (take-home)

Desirable Properties of Transactions: ACID

- Atomicity: A transaction is an atomic unit of processing; it is either performed in its entirety or not performed at all.

- Consistency: A correct execution of the transaction must take the database from one consistent state to another.

- Isolation: A transaction should not make its updates visible to other transactions until it is committed.

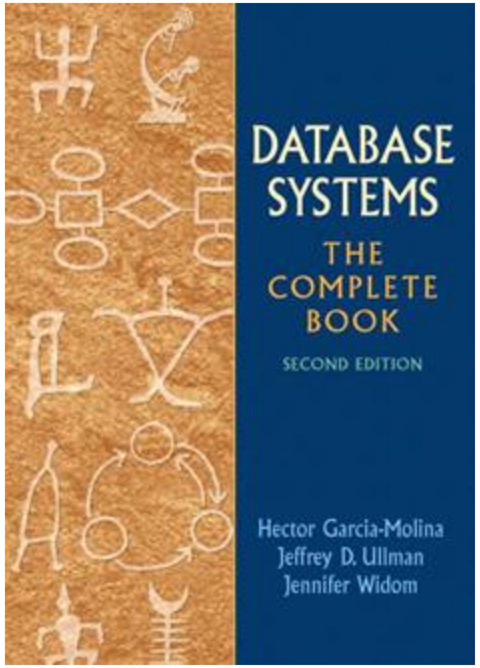
- Durability: Once a transaction changes the database and the changes are committed, these changes must never be lost because of subsequent failure.

This class: ensuring atomicity and durability with logging and recovery manager

Reading Materials

Database Systems: The Complete Book (2nd edition)

- Chapter 17 - Copying with System Failures



Supplementary materials

Fundamental of Database Systems (7th Edition)

- Chapter 22 - Database Recovery Techniques

Agenda

1. WAL Protocol
2. Undo Logging
3. Redo Logging
4. Undo/redo logging

Failure modes and solutions

Erroneous data entry

- Typos
  → Write constraints and triggers

Media failures

- Local disk failure, head crashes
  → Parity checks, RAID, archiving and copying

Catastrophic failures

- Explosions, fires
  → Archiving and copying

System failures

- Transaction state lost due to power loss and software errors
  → Logging

Our focus today

Summary Recovery Mechanism

Atomicity

- by ”undo”ing actions of “aborted transactions”

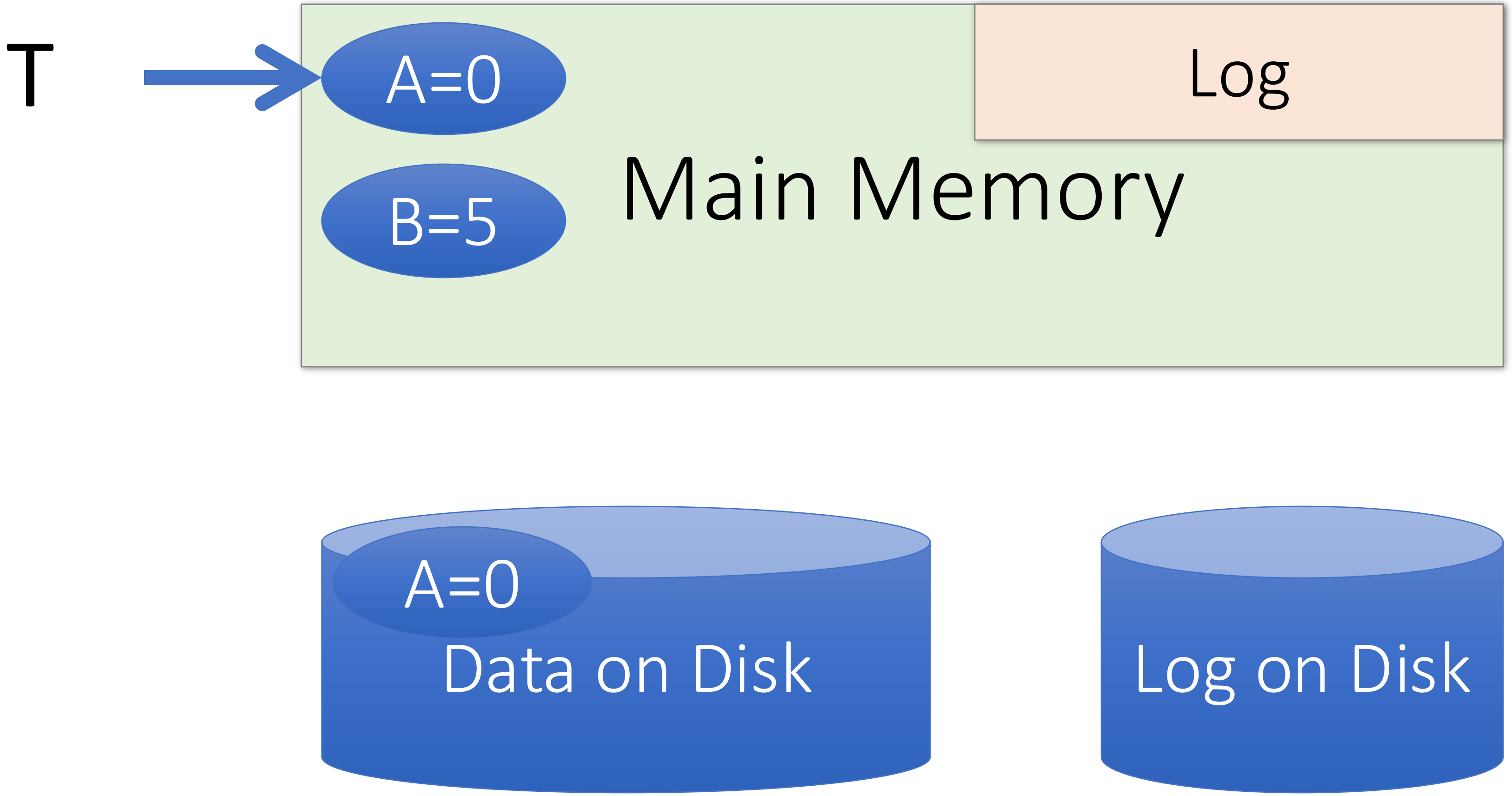
Durability

- by making sure that all actions of committed transactions survive crashes and system failure
- – i.e. by “redo”-ing actions of “committed transactions”

1. Write-Ahead Logging (WAL) TXN Commit Protocol

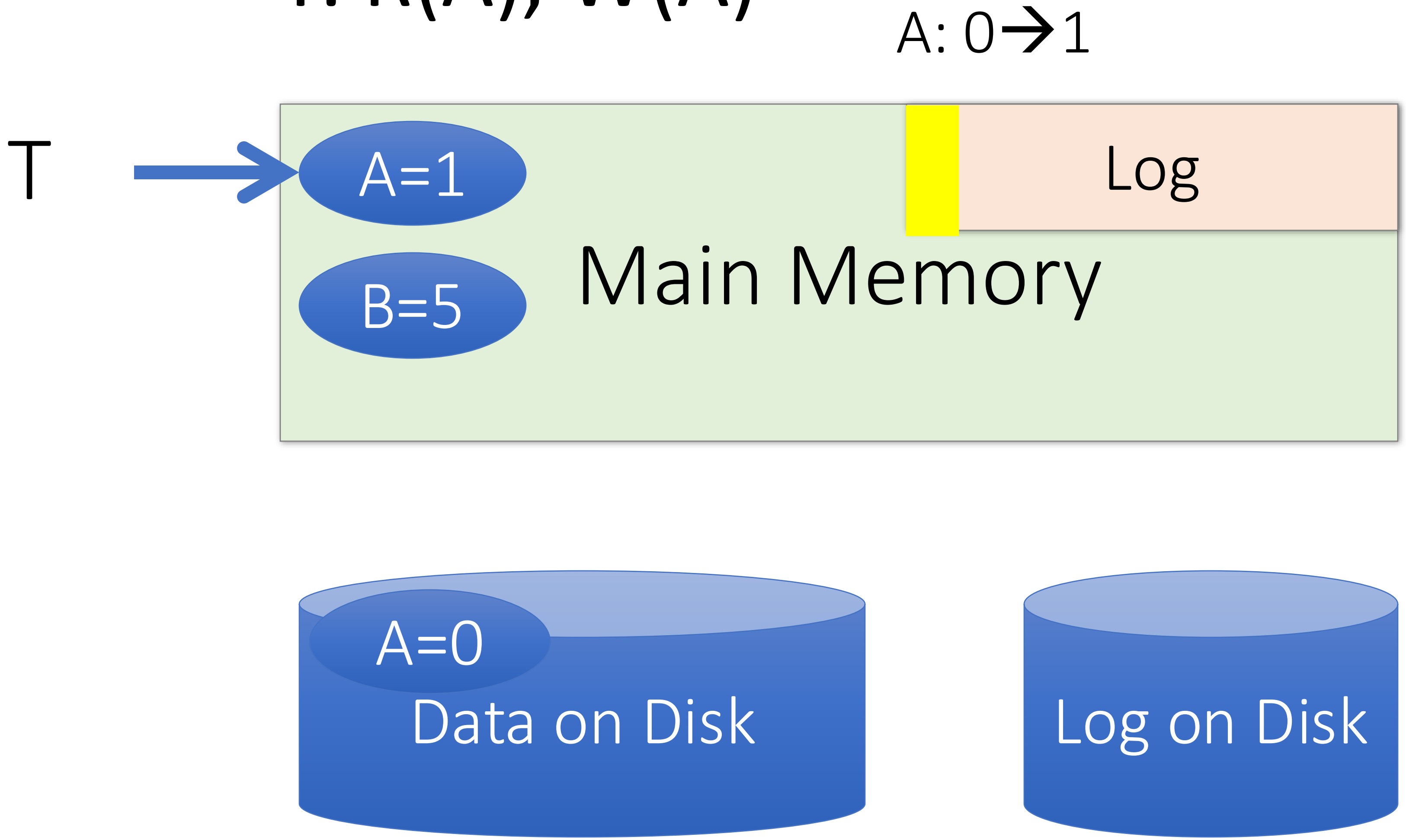
A picture of logging

T: R(A), W(A)



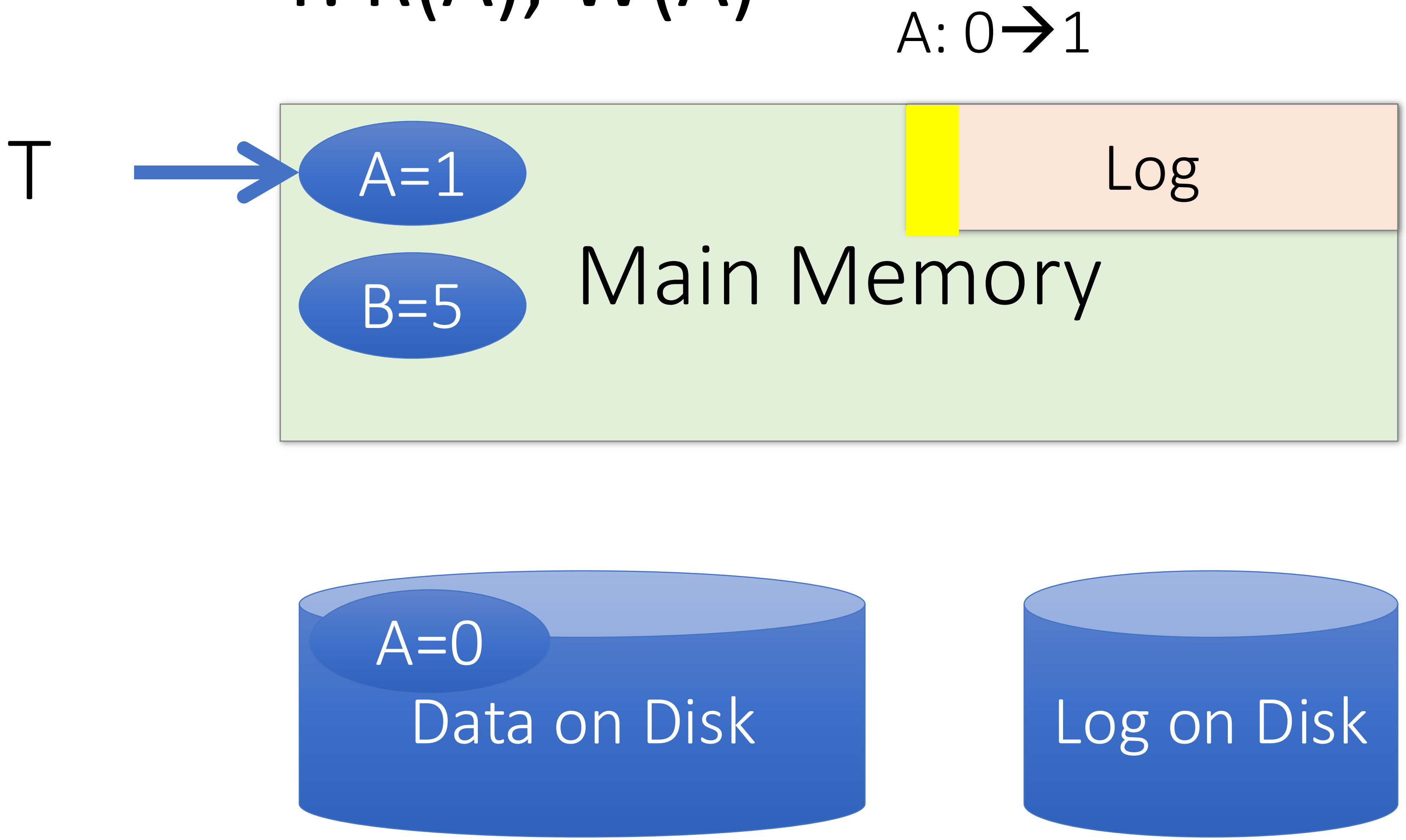
A picture of logging

T: R(A), W(A)



A picture of logging

T: R(A), W(A)



Operation recorded in log in main memory!

What is the correct way to write this all to disk?

- We'll look at the *Write-Ahead Logging (WAL)* protocol
- We'll see why it works by looking at other protocols which are incorrect!

Remember: Key idea is to ensure durability *while* maintaining our ability to "undo"!

Transaction Commit Process

1. FORCE Write **commit** record to log

FORCE: write operation must be completed to persistent storage before proceeding

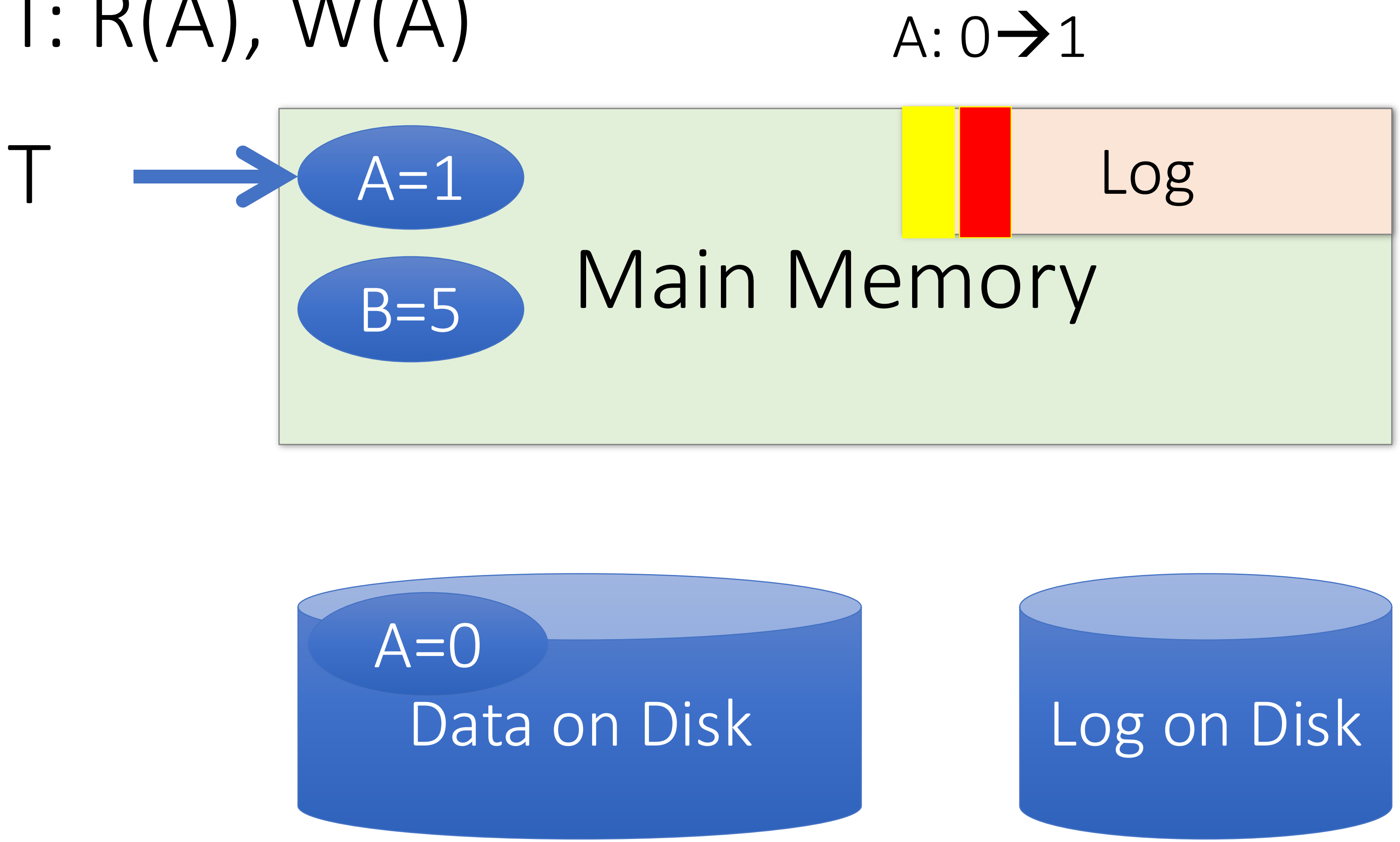
2. All log records up to last update from this TX are FORCED

3. Commit() returns

Transaction is committed *once commit log record is on stable storage*

Incorrect Commit Protocol #1

T: R(A), W(A)



Let's try committing *before* we've written either data or log to disk…

OK, Commit!

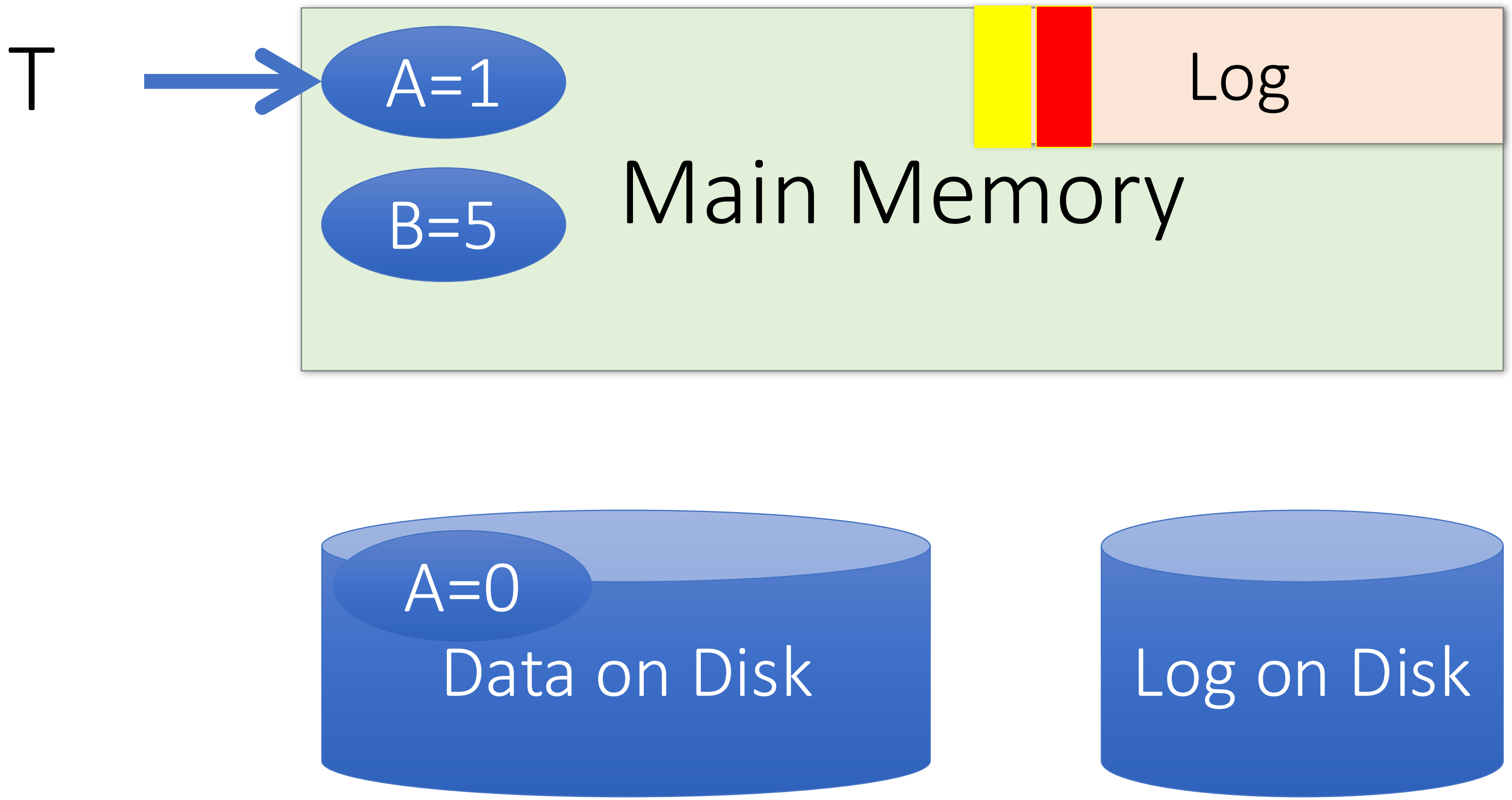
If we crash now, is T durable?

Lost T's update!

Incorrect Commit Protocol #2

T: R(A), W(A)

A: 0→1



Let's try committing *after* we've written data but *before* we've written log to disk…

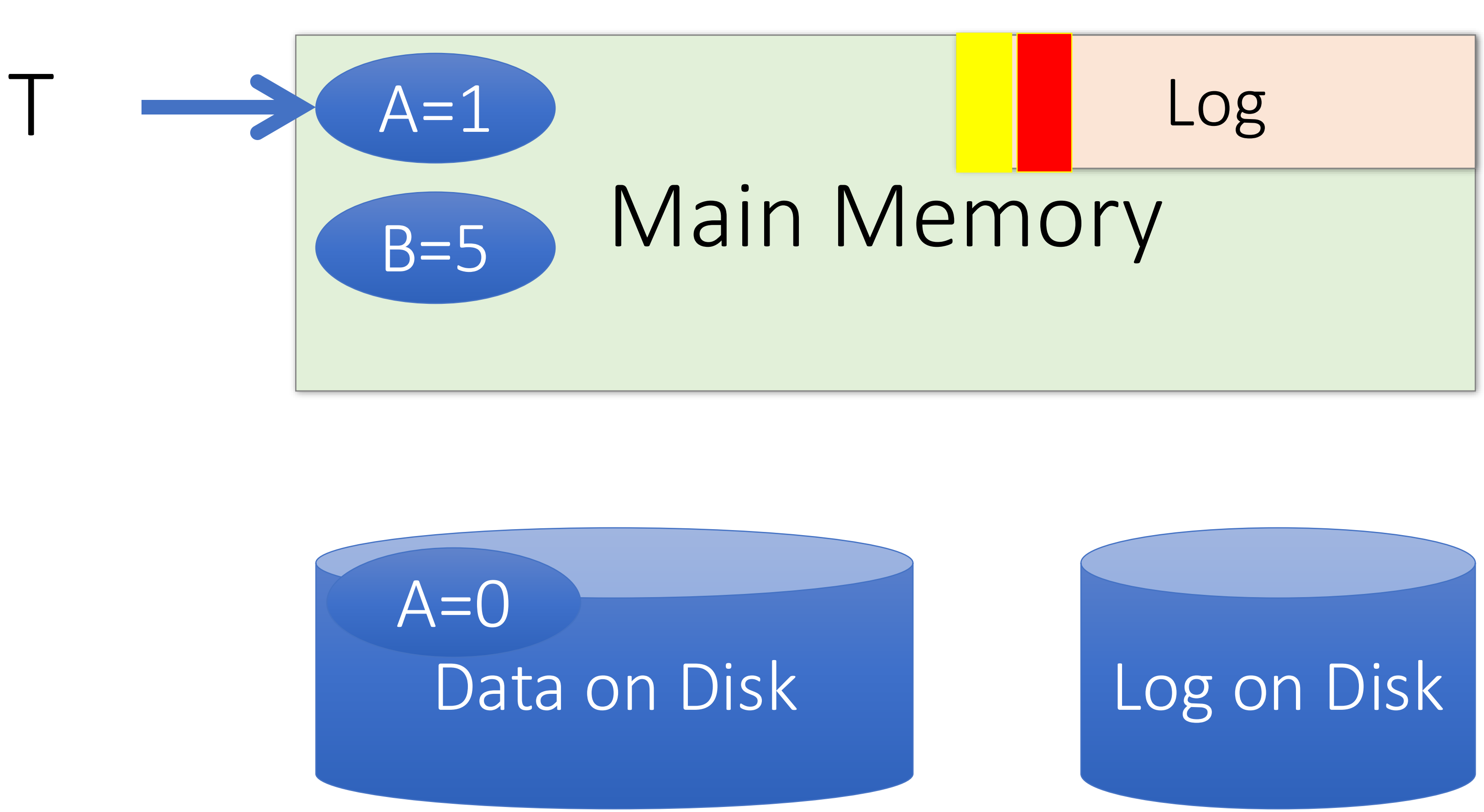
OK, Commit!

If we crash now, is T durable? Yes! Except…

How do we know whether T was committed??

Write-ahead Logging (WAL) Commit Protocol

T: R(A), W(A)



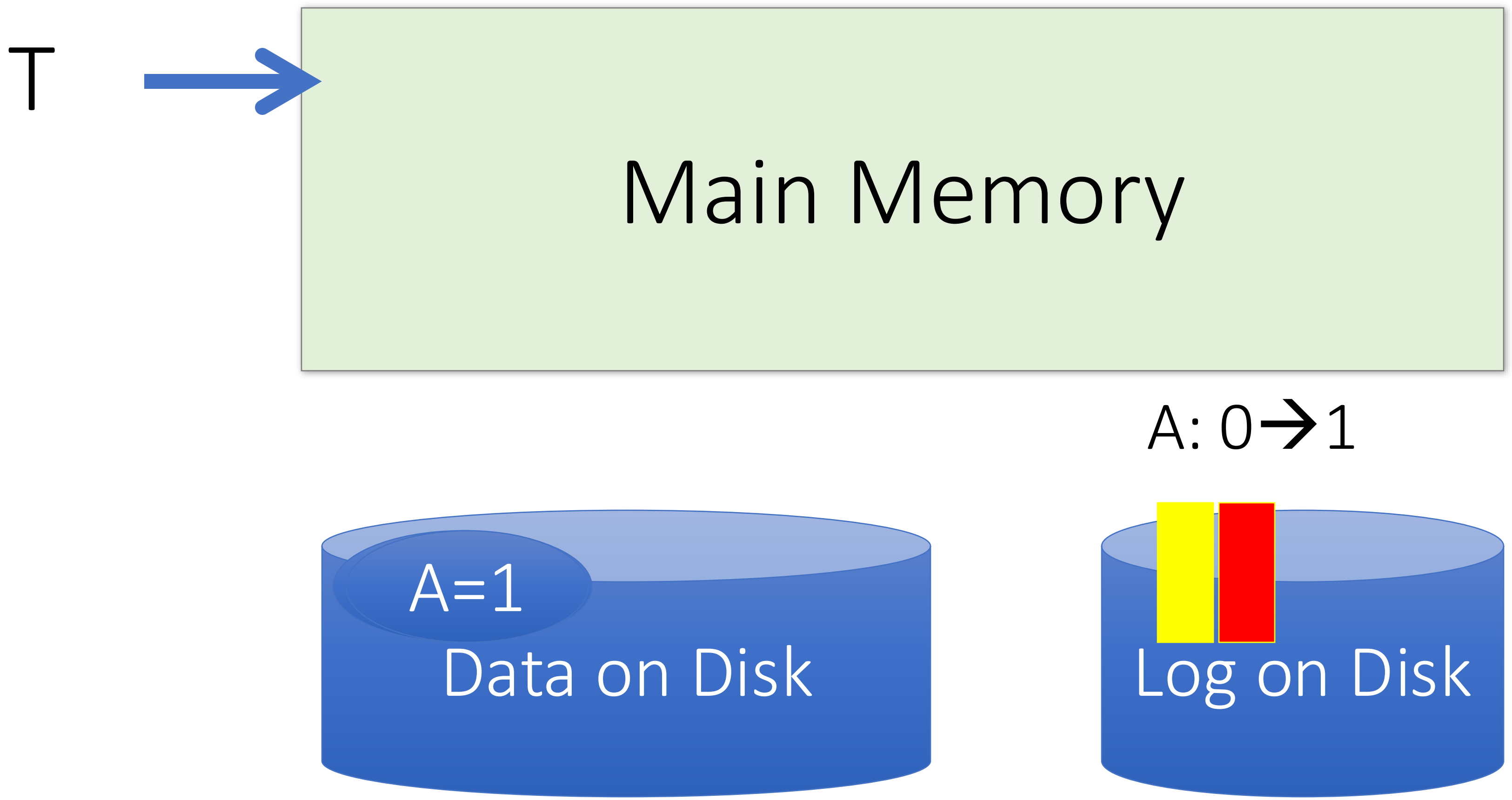
This time, let's try committing ***after* we've written log to disk but *before* we've written data to disk**... this is WAL!

OK, Commit!

If we crash now, is T durable?

Write-ahead Logging (WAL) Commit Protocol

T: R(A), W(A)



This time, let's try committing ***after* we've written log to disk but *before* we've written data to disk**... this is WAL!

OK, Commit!

If we crash now, is T durable?

USE THE LOG!

Write-Ahead Logging (WAL)

DB uses **Write-Ahead Logging (WAL)** Protocol:

Each update is logged! Why not reads?

1. Log before data: Must *force log record* for an update *before* the corresponding data page goes to storage

→ Atomicity

2. Force log on commit: Must *write all log records* for a TX *before commit*

→ Durability

Logging Mechanisms

Different logging schemes define how changes are logged, and what recovery actions are needed.

We will discuss three approaches (all follow WAL):

- Undo logging
- Redo logging
- Undo/Redo logging

Transaction primitives

- Example transaction
  - Consistent state: $A = B$

Logical steps

$A := A * 2$
$B := B * 2$

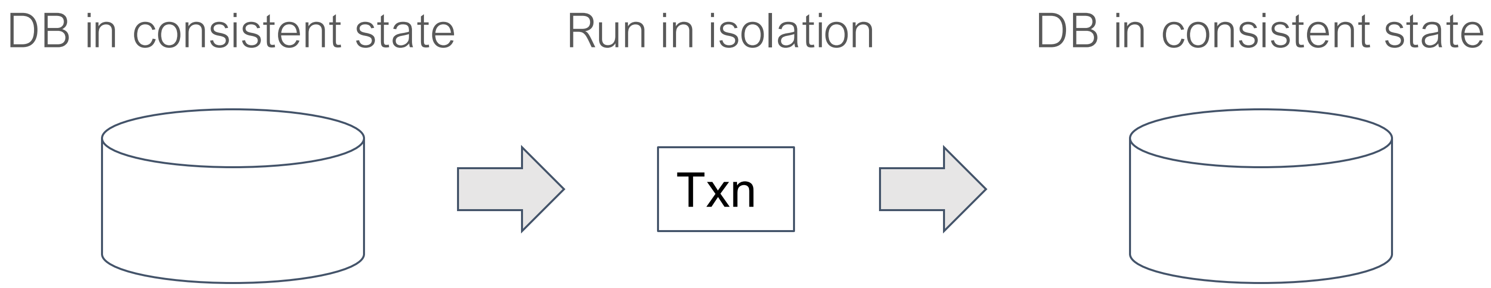
Execution

Action	t	Memory A	Memory 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
OUTPUT(A)	16	16	16	16	8
OUTPUT(B)	16	16	16	16	16

Recall: The Correctness Principle

A fundamental assumption about transaction is:

If a transaction executes in the absence of any other transactions or system errors, and it starts with the database in a consistent state, then the database is also in a consistent state when the transactions ends.



Transaction primitives

- Example transaction
  - Consistent state: $A = B$

Logical steps

$A := A * 2$
$B := B * 2$

Execution

Action	t	Memory A	Memory 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
OUTPUT(A)	16	16	16	16	8
OUTPUT(B)	16	16	16	16	16

Consistent (marked between WRITE(B, t) and OUTPUT(A))

Transaction primitives

- Example transaction
  - Consistent state: $A = B$

Logical steps

$A := A * 2$
$B := B * 2$

Execution

Action	t	Memory A	Memory 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
OUTPUT(A)	16	16	16	16	8
OUTPUT(B)	16	16	16	16	16

Consistent

Transaction primitives

- Example transaction
  - Consistent state: $A = B$

Execution

Logical steps

$A := A * 2$
$B := B * 2$

		Memory		Disk	
Action	t	A	B	A	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
OUTPUT(A)	16	16	16	16	8
OUTPUT(B)	16	16	16	16	16

Not consistent!
Either reset $A = 8$
or advance $B = 16$

2. Undo logging

Undo logging

- Idea: Undo incomplete transactions, and ignore committed ones

		Memory		Disk		
Action	*t*	*A*	*B*	*A*	*B*	Log
						<START *T*>
READ(*A*, t)	8	8		8	8	
t := *t* * 2	16	8		8	8	
WRITE(*A*, *t*)	16	16		8	8	<*T*, *A*, 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>
FLUSH LOG						
OUTPUT(*A*)	16	16	16	16	8	
OUTPUT(*B*)	16	16	16	16	16	
						<COMMIT *T*>
FLUSH LOG						

Undo log format:

<*T*, *X*, *v*>: T updated database element X whose old value is v

Undo logging

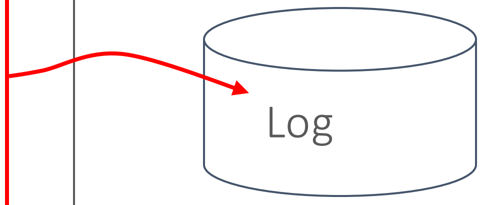
- Idea: Undo incomplete transactions, and ignore committed ones

		Memory		Disk			
Action	*t*	*A*	*B*	*A*	*B*	Log	
						<START *T*>	*T* started
READ(*A*, t)	8	8		8	8		
t := *t* * 2	16	8		8	8		
WRITE(*A*, *t*)	16	16		8	8	<*T*, *A*, 8>	*T* changed *A*, and its former value is 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>	
FLUSH LOG							
OUTPUT(*A*)	16	16	16	16	8		
OUTPUT(*B*)	16	16	16	16	16		
						<COMMIT *T*>	*T* completed successfully
FLUSH LOG							

Undo logging

- Idea: Undo incomplete transactions, and ignore committed ones

Action	*t*	Memory *A*	Memory *B*	Disk *A*	Disk *B*	Log
						<START *T*>
READ(*A*, t)	8	8		8	8	
t := *t* * 2	16	8		8	8	
WRITE(*A*, *t*)	16	16		8	8	<*T*, *A*, 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>
FLUSH LOG						
OUTPUT(*A*)	16	16	16	16	8	
OUTPUT(*B*)	16	16	16	16	16	
						<COMMIT *T*>
FLUSH LOG						



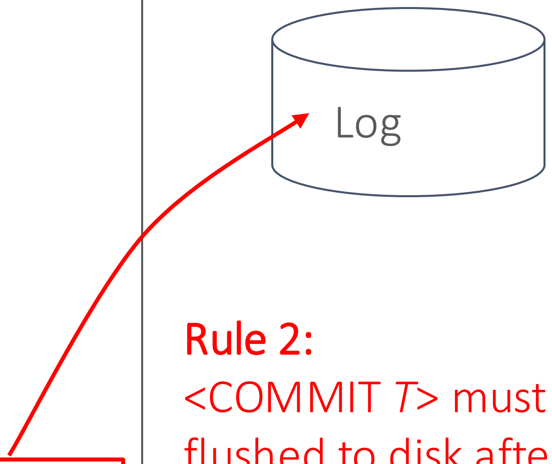
Rule 1:
<*T*, *A*, 8> must be flushed to disk before new *A* is written to disk (same for *B*)

Undo logging

- Idea: Undo incomplete transactions, and ignore committed ones

		Memory		Disk		
Action	*t*	*A*	*B*	*A*	*B*	Log
						<START *T*>
READ(*A*, t)	8	8		8	8	
t := *t* * 2	16	8		8	8	
WRITE(*A*, *t*)	16	16		8	8	<*T*, *A*, 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>
FLUSH LOG						
OUTPUT(*A*)	16	16	16	16	8	
OUTPUT(*B*)	16	16	16	16	16	
						<COMMIT *T*>
FLUSH LOG						

Rule 1:
<*T*, *A*, 8> must be flushed to disk before new *A* is written to disk (same for *B*)



Rule 2:
<COMMIT *T*> must be flushed to disk after *A* and *B* are written to disk

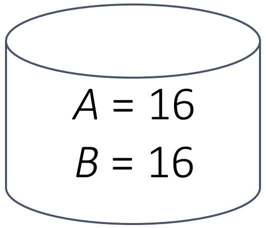
Recovery using undo logging

- Simplifying assumption: use entire log, no matter how long

		Memory		Disk		
Action	*t*	*A*	*B*	*A*	*B*	Log
						<START *T*>
READ(*A*, t)	8	8		8	8	
t := *t* * 2	16	8		8	8	
WRITE(*A*, *t*)	16	16		8	8	<*T*, *A*, 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>
FLUSH LOG						
OUTPUT(*A*)	16	16	16	16	8	
OUTPUT(*B*)	16	16	16	16	16	
						<COMMIT *T*>
FLUSH LOG						

Crash

Recovery



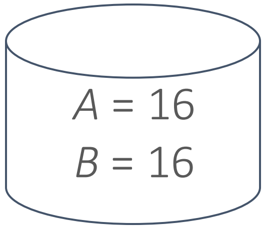
Recovery using undo logging

- Simplifying assumption: use entire log, no matter how long

		Memory		Disk		
Action	*t*	*A*	*B*	*A*	*B*	Log
						<START *T*>
READ(*A*, t)	8	8		8	8	
t := *t* * 2	16	8		8	8	
WRITE(*A*, *t*)	16	16		8	8	<*T*, *A*, 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>
FLUSH LOG						
OUTPUT(*A*)	16	16	16	16	8	
OUTPUT(*B*)	16	16	16	16	16	
						<COMMIT *T*>
FLUSH LOG						

Crash

Recovery



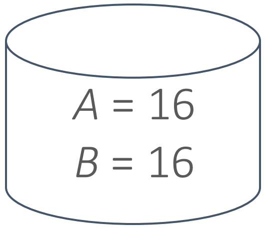
Observe <COMMIT *T*> record

Recovery using undo logging

- Simplifying assumption: use entire log, no matter how long

		Memory		Disk			Recovery
Action	*t*	*A*	*B*	*A*	*B*	Log	
						<START *T*>	
READ(*A*, t)	8	8		8	8		
t := *t* * 2	16	8		8	8		
WRITE(*A*, *t*)	16	16		8	8	<*T*, *A*, 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>	Ignore (*T* was committed)
FLUSH LOG							
OUTPUT(*A*)	16	16	16	16	8		⇧
OUTPUT(*B*)	16	16	16	16	16		
						<COMMIT *T*>	Observe <COMMIT *T*> record
FLUSH LOG							

Crash



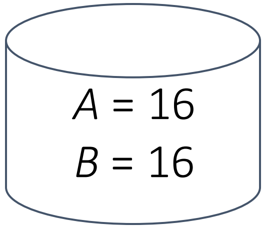
Recovery using undo logging

- Simplifying assumption: use entire log, no matter how long

	Memory			Disk		
Action	*t*	*A*	*B*	*A*	*B*	Log
						<START *T*>
READ(*A*, t)	8	8		8	8	
t := *t* * 2	16	8		8	8	
WRITE(*A*, *t*)	16	16		8	8	<*T*, *A*, 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>
FLUSH LOG						
OUTPUT(*A*)	16	16	16	16	8	
OUTPUT(*B*)	16	16	16	16	16	
						<COMMIT *T*>
FLUSH LOG						

Crash

Recovery



Ignore (*T* was committed)

Ignore (*T* was committed)

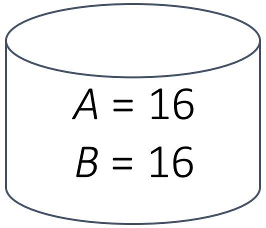
Observe <COMMIT *T*> record

Recovery using undo logging

- Simplifying assumption: use entire log, no matter how long

Action	*t*	Memory *A*	Memory *B*	Disk *A*	Disk *B*	Log
						<START *T*>
READ(*A*, t)	8	8		8	8	
t := *t* * 2	16	8		8	8	
WRITE(*A*, *t*)	16	16		8	8	<*T*, *A*, 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>
FLUSH LOG						
OUTPUT(*A*)	16	16	16	16	8	
OUTPUT(*B*)	16	16	16	16	16	
						<COMMIT *T*>
Crash						
FLUSH LOG						

Recovery



Recovery using undo logging

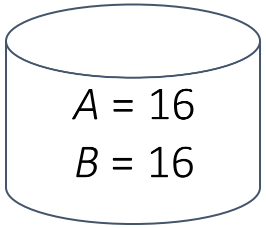
- Simplifying assumption: use entire log, no matter how long

		Memory		Disk		
Action	*t*	*A*	*B*	*A*	*B*	Log
						<START *T*>
READ(*A*, t)	8	8		8	8	
t := *t* * 2	16	8		8	8	
WRITE(*A*, *t*)	16	16		8	8	<*T*, *A*, 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>
FLUSH LOG						
OUTPUT(*A*)	16	16	16	16	8	
OUTPUT(*B*)	16	16	16	16	16	
						<COMMIT *T*>
FLUSH LOG						

Crash

<COMMIT *T*> may or may not have been flushed to disk. If so, same as previous scenario. If not, *T* is considered incomplete

Recovery



Recovery using undo logging

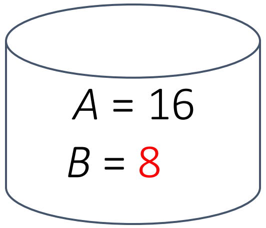
- Simplifying assumption: use entire log, no matter how long

		Memory		Disk		
Action	*t*	*A*	*B*	*A*	*B*	Log
						<START *T*>
READ(*A*, t)	8	8		8	8	
t := *t* * 2	16	8		8	8	
WRITE(*A*, *t*)	16	16		8	8	<*T*, *A*, 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>
FLUSH LOG						
OUTPUT(*A*)	16	16	16	16	8	
OUTPUT(*B*)	16	16	16	16	16	
						<COMMIT *T*>
FLUSH LOG						

Crash

If *T* was incomplete, set *B* to previous value 8 on disk

Recovery



Recovery using undo logging

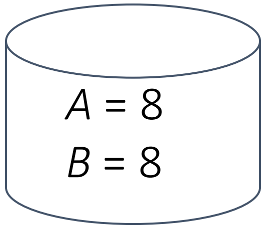
- Simplifying assumption: use entire log, no matter how long

Action	Memory			Disk		
Action	*t*	*A*	*B*	*A*	*B*	Log
						<START *T*>
READ(*A*, t)	8	8		8	8	
t := *t* * 2	16	8		8	8	
WRITE(*A*, *t*)	16	16		8	8	<*T*, *A*, 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>
FLUSH LOG						
OUTPUT(*A*)	16	16	16	16	8	
OUTPUT(*B*)	16	16	16	16	16	
						<COMMIT *T*>
FLUSH LOG						

Crash

If *T* was incomplete, set *A* to previous value 8 on disk

Recovery



Recovery using undo logging

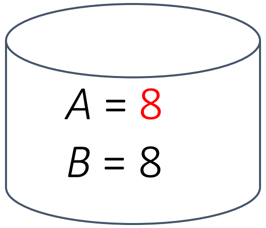
- Simplifying assumption: use entire log, no matter how long

		Memory		Disk		
Action	*t*	*A*	*B*	*A*	*B*	Log
						<START *T*>
READ(*A*, t)	8	8		8	8	
t := *t* * 2	16	8		8	8	
WRITE(*A*, *t*)	16	16		8	8	<*T*, *A*, 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>
FLUSH LOG						
OUTPUT(*A*)	16	16	16	16	8	
OUTPUT(*B*)	16	16	16	16	16	
						<COMMIT *T*>
FLUSH LOG						

Crash

Recovery

Write <ABORT *T*> to log and flush to disk

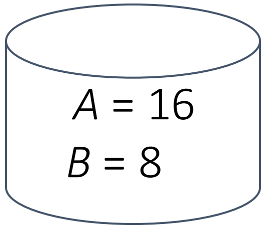


Recovery using undo logging

- Simplifying assumption: use entire log, no matter how long

Action	*t*	Memory *A*	Memory *B*	Disk *A*	Disk *B*	Log
						<START *T*>
READ(*A*, t)	8	8		8	8	
t := *t* * 2	16	8		8	8	
WRITE(*A*, *t*)	16	16		8	8	<*T*, *A*, 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>
FLUSH LOG						
OUTPUT(*A*)	16	16	16	16	8	
Crash						
OUTPUT(*B*)	16	16	16	16	16	
						<COMMIT *T*>
FLUSH LOG						

Recovery

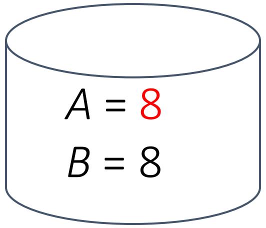


Recovery using undo logging

- Simplifying assumption: use entire log, no matter how long

		Memory		Disk		
Action	*t*	*A*	*B*	*A*	*B*	Log
						<START *T*>
READ(*A*, t)	8	8		8	8	
t := *t* * 2	16	8		8	8	
WRITE(*A*, *t*)	16	16		8	8	<*T*, *A*, 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>
FLUSH LOG						
OUTPUT(*A*)	16	16	16	16	8	
Crash						
OUTPUT(*B*)	16	16	16	16	16	
						<COMMIT *T*>
FLUSH LOG						

Recovery



Same recovery as before, but only *A* is set to previous value

What happens if the system crashes during the recovery?

- Undo-log recovery is idempotent, so repeating the recovery is OK

Image source: https://insightrsblog.com/2010/09/17/do-you-want-to-recreate-your-entire-database/

In-class Exercise

- Given the undo log, describe the action of the recovery manager

<START T>
<T, *A*, 10>
<START U>
<U, *B*, 20>
<T, *C*, 30>
<U, *D*, 40>
<COMMIT U>
Crash

Checkpointing

- Entire log can be too long
- Cannot truncate log after a COMMIT because there are other running transactions

Checkpointing

- Solution: checkpoint log periodically

<START T1>
<T1, *A*, 5>
<START T2>
<T2, *B*, 10>

Checkpointing

- Solution: checkpoint log periodically

<START T1>
<T1, *A*, 5>
<START T2>
<T2, *B*, 10>

Stop accepting new transactions

Checkpointing

- Solution: checkpoint log periodically

<START T1>
<T1, *A*, 5>
<START T2>
<T2, *B*, 10>
<T2, *C*, 15>
<T1, *D*, 20>
<COMMIT T1>
<COMMIT T2>

Stop accepting new transactions

Wait until all transactions commit or abort

Checkpointing

- Solution: checkpoint log periodically

<START T1>
<T1, *A*, 5>
<START T2>
<T2, *B*, 10>
<T2, *C*, 15>
<T1, *D*, 20>
<COMMIT T1>
<COMMIT T2>
<CKPT>

Stop accepting new transactions

Wait until all transactions commit or abort

Flush log

Write <CKPT> and flush

Checkpointing

- Solution: checkpoint log periodically

<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>

Stop accepting new transactions

Wait until all transactions commit or abort

Flush log

Write <CKPT> and flush

Resume transactions

Nonquiescent checkpointing

- Motivation: avoid shutting down system while checkpointing
- Checkpoint all active transactions, but allow new transactions to enter system

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

Nonquiescent checkpointing

- Motivation: avoid shutting down system while checkpointing
- Checkpoint all active transactions, but allow new transactions to enter system

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

Crash

If we first meet <END CKPT>, only need to recover until <START CKPT (T1, T2)>

Nonquiescent checkpointing

- Motivation: avoid shutting down system while checkpointing
- Checkpoint all active transactions, but allow new transactions to enter system

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

If we first meet <START CKPT (T1, T2)>, only need to recover until <START T1>

3. Redo logging

Redo logging

Redo logging ignores incomplete transactions and repeats committed ones

- Undo logging cancels incomplete transactions and ignores committed ones

<*T*, *X*, *v*> now means *T* wrote new value *v* for database element *X*

One rule: all log records (e.g., <*T*, *X*, *v*> and <COMMIT *T*>) must appear on disk before modifying any database element *X* on disk

Redo logging

- Example

Action	*t*	Memory *A*	Memory *B*	Disk *A*	Disk *B*	Log
						<START *T*>
READ(*A*, t)	8	8		8	8	
t := *t* * 2	16	8		8	8	
WRITE(*A*, *t*)	16	16		8	8	<*T*, *A*, 16>
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*, 16>
						<COMMIT *T*>
FLUSH LOG						
OUTPUT(*A*)	16	16	16	16	8	
OUTPUT(*B*)	16	16	16	16	16	

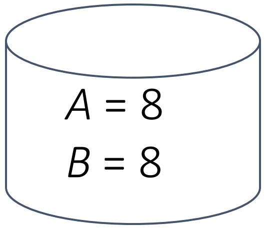
Recovery with redo logging

- Scan log forward and redo committed transactions

Action	Memory *t*	Memory *A*	Memory *B*	Disk *A*	Disk *B*	Log
						<START *T*>
READ(*A*, t)	8	8		8	8	
t := *t* * 2	16	8		8	8	
WRITE(*A*, *t*)	16	16		8	8	<*T*, *A*, 16>
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*, 16>
						<COMMIT *T*>
FLUSH LOG						
OUTPUT(*A*)	16	16	16	16	8	
OUTPUT(*B*)	16	16	16	16	16	

Crash

Recovery



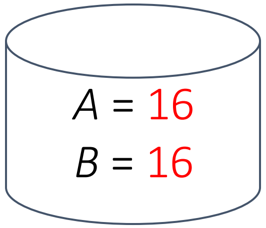
Recovery with redo logging

- Scan log forward and redo committed transactions

Action	*t*	Memory *A*	Memory *B*	Disk *A*	Disk *B*	Log
						<START *T*>
READ(*A*, t)	8	8		8	8	
t := *t* * 2	16	8		8	8	
WRITE(*A*, *t*)	16	16		8	8	<*T*, *A*, 16>
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*, 16>
						<COMMIT *T*>
FLUSH LOG						
OUTPUT(*A*)	16	16	16	16	8	
OUTPUT(*B*)	16	16	16	16	16	

Crash

Recovery



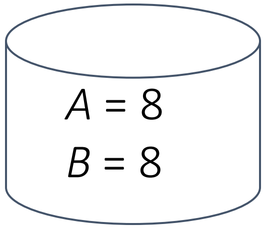
Recovery with redo logging

- Scan log forward and redo committed transactions

		Memory		Disk		
Action	*t*	*A*	*B*	*A*	*B*	Log
						<START *T*>
READ(*A*, t)	8	8		8	8	
t := *t* * 2	16	8		8	8	
WRITE(*A*, *t*)	16	16		8	8	<*T*, *A*, 16>
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*, 16>
						<COMMIT *T*>
FLUSH LOG						
OUTPUT(*A*)	16	16	16	16	8	
OUTPUT(*B*)	16	16	16	16	16	

Crash

Recovery



Recovery with redo logging

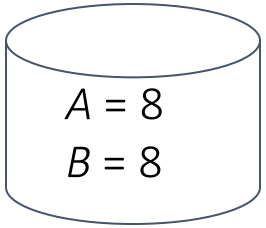
- Scan log forward and redo committed transactions

		Memory		Disk		
Action	*t*	*A*	*B*	*A*	*B*	Log
						<START *T*>
READ(*A*, t)	8	8		8	8	
t := *t* * 2	16	8		8	8	
WRITE(*A*, *t*)	16	16		8	8	<*T*, *A*, 16>
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*, 16>
						<COMMIT *T*>
FLUSH LOG						
OUTPUT(*A*)	16	16	16	16	8	
OUTPUT(*B*)	16	16	16	16	16	

Crash

Recovery

Do nothing



Nonquiescent checkpointing for redo log

- Write to disk all DB elements modified by committed transactions

<START T1>
<T1, *A*, 5>
<START T2>
<COMMIT T1>
<T2, *B*, 10>
<START CKPT (T2)>

Nonquiescent checkpointing for redo log

- Write to disk all DB elements modified by committed transactions

<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>

Write to disk all DB elements by transactions that already committed when START CKPT was written to log (i.e., T1)

Nonquiescent checkpointing for redo log

- Write to disk all DB elements modified by committed transactions

<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>

Write to disk all DB elements by transactions that already committed when START CKPT was written to log (i.e., T1)

Nonquiescent checkpointing for redo log

- After crash, redo committed transactions that either started after START CKPT or were active during START CKPT

<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>
Crash
<COMMIT T3>

Nonquiescent checkpointing for redo log

- After crash, redo committed transactions that either started after START CKPT or were active during START CKPT

<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>
Crash
<COMMIT T3>

Only redo writes by T2
Write <ABORT T3> in log after recovery

4. Undo/redo logging

Undo/redo logging

More flexible than undo or redo logging in ordering actions

<T, X, v, w> : T changed value of X from v to w

One rule: <T, X, v, w> must appear on disk before modifying X on disk

Undo/redo logging

- Example

Action	*t*	Memory *A*	Memory *B*	Disk *A*	Disk *B*	Log
						<START *T*>
READ(*A*, t)	8	8		8	8	
t := *t* * 2	16	8		8	8	
WRITE(*A*, *t*)	16	16		8	8	<*T*, *A*, 8, 16>
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, 16>
FLUSH LOG						
OUTPUT(*A*)	16	16	16	16	8	
						<COMMIT *T*>
OUTPUT(*B*)	16	16	16	16	16	

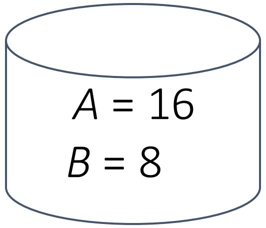
Recovery with undo/redo logging

- Redo all committed transactions and undo all incomplete transactions

		Memory		Disk		
Action	*t*	*A*	*B*	*A*	*B*	Log
						<START *T*>
READ(*A*, t)	8	8		8	8	
t := *t* * 2	16	8		8	8	
WRITE(*A*, *t*)	16	16		8	8	<*T*, *A*, 8, 16>
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, 16>
FLUSH LOG						
OUTPUT(*A*)	16	16	16	16	8	
						<COMMIT *T*>
OUTPUT(*B*)	16	16	16	16	16	

Crash

Recovery



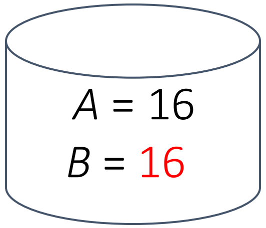
Recovery with undo/redo logging

- Redo all committed transactions and undo all incomplete transactions

Action	t	A (Memory)	B (Memory)	A (Disk)	B (Disk)	Log
						<START *T*>
READ(*A*, t)	8	8		8	8	
t := *t* * 2	16	8		8	8	
WRITE(*A*, *t*)	16	16		8	8	<*T*, *A*, 8, 16>
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, 16>
FLUSH LOG						
OUTPUT(*A*)	16	16	16	16	8	
						<COMMIT *T*>
OUTPUT(*B*)	16	16	16	16	16	

Crash

Recovery



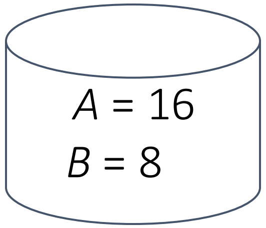
T is commited
Redo by writing the value 16 for both *A* and *B* to the disk.

Recovery with undo/redo logging

- Redo all committed transactions and undo all incomplete transactions

		Memory		Disk		
Action	*t*	*A*	*B*	*A*	*B*	Log
						<START *T*>
READ(*A*, t)	8	8		8	8	
t := *t* * 2	16	8		8	8	
WRITE(*A*, *t*)	16	16		8	8	<*T*, *A*, 8, 16>
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, 16>
FLUSH LOG						
OUTPUT(*A*)	16	16	16	16	8	
Crash						
						<COMMIT *T*>
OUTPUT(*B*)	16	16	16	16	16	

Recovery

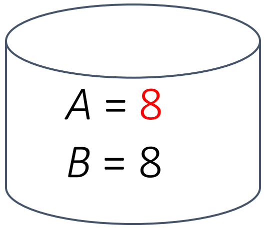


Recovery with undo/redo logging

- Redo all committed transactions and undo all incomplete transactions

		Memory		Disk		
Action	*t*	*A*	*B*	*A*	*B*	Log
						<START *T*>
READ(*A*, t)	8	8		8	8	
t := *t* * 2	16	8		8	8	
WRITE(*A*, *t*)	16	16		8	8	<*T*, *A*, 8, 16>
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, 16>
FLUSH LOG						
OUTPUT(*A*)	16	16	16	16	8	
Crash						
						<COMMIT *T*>
OUTPUT(*B*)	16	16	16	16	16	

Recovery



T is incomplete
Undo by resetting *A* and *B* to the previous value of 8

Nonquiescent checkpointing for undo/redo logging

- Simpler than other logging methods

<START T1>
<T1, *A*, 4, 5>
<START T2>
<COMMIT T1>
<T2, *B*, 9, 10>
<START CKPT (T2)>

Nonquiescent checkpointing for undo/redo logging

- Simpler than other logging methods

<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>

Write to disk all the buffers that are dirty

Nonquiescent checkpointing for undo/redo logging

- Simpler than other logging methods

<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>

Write to disk all the buffers that are dirty

Nonquiescent checkpointing for undo/redo logging

- After a crash, redo committed transactions, and undo uncommitted ones

<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>
Crash
<COMMIT T3>

Nonquiescent checkpointing for undo/redo logging

- After a crash, redo committed transactions, and undo uncommitted ones

<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>
Crash
<COMMIT T3>

Redo T2 by setting C to 15 on disk
(No need to set B to 10 thanks to CKPT)
Undo T3 by setting D to 19 on disk

Summary

Write-ahead logging protocol

- All changes to a transaction log should be written to disk before modifying the actual database

Coping with System Failures

- Undo logging
- Redo logging
- Undo/redo logging
- Checkpointing