WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

CS 564: Database Management Systems

Lecture 33: Logging

Xiangyao Yu

4/15/2024

Module B4 Transactions

Concurrency control

Optimistic concurrency control (OCC)

Logging

ARIES recovery

Outline

Durability

Types of failures

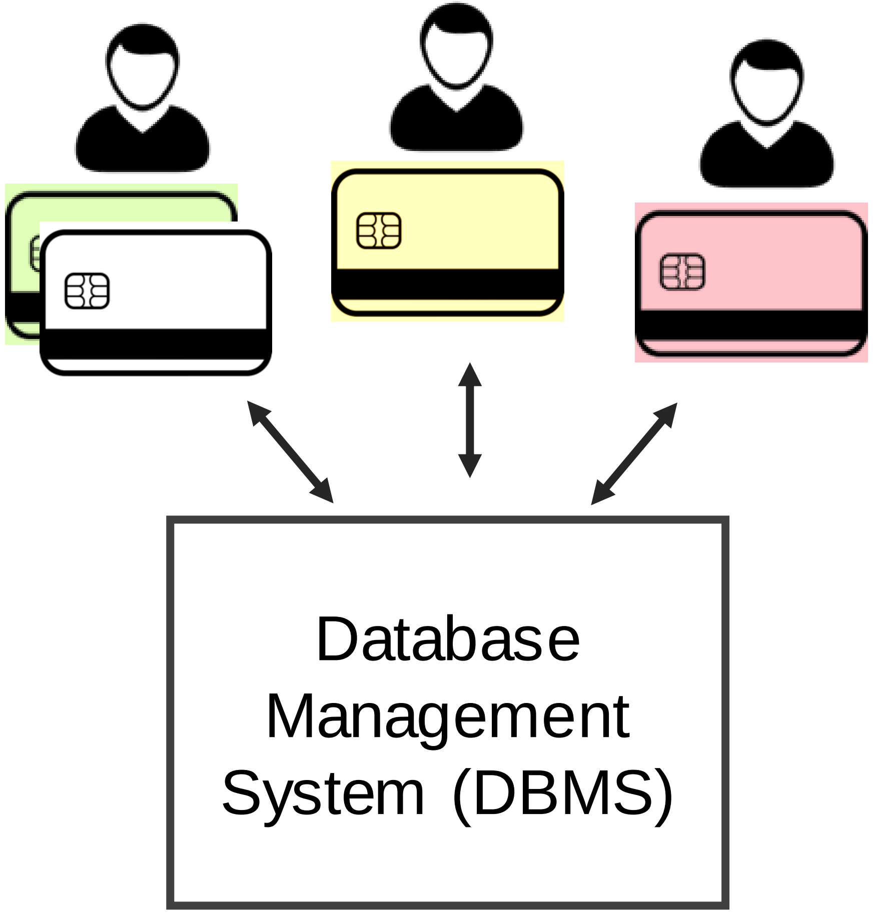
Failure examples

Write ahead logging (WAL)

Buffer management policies

Other discussion

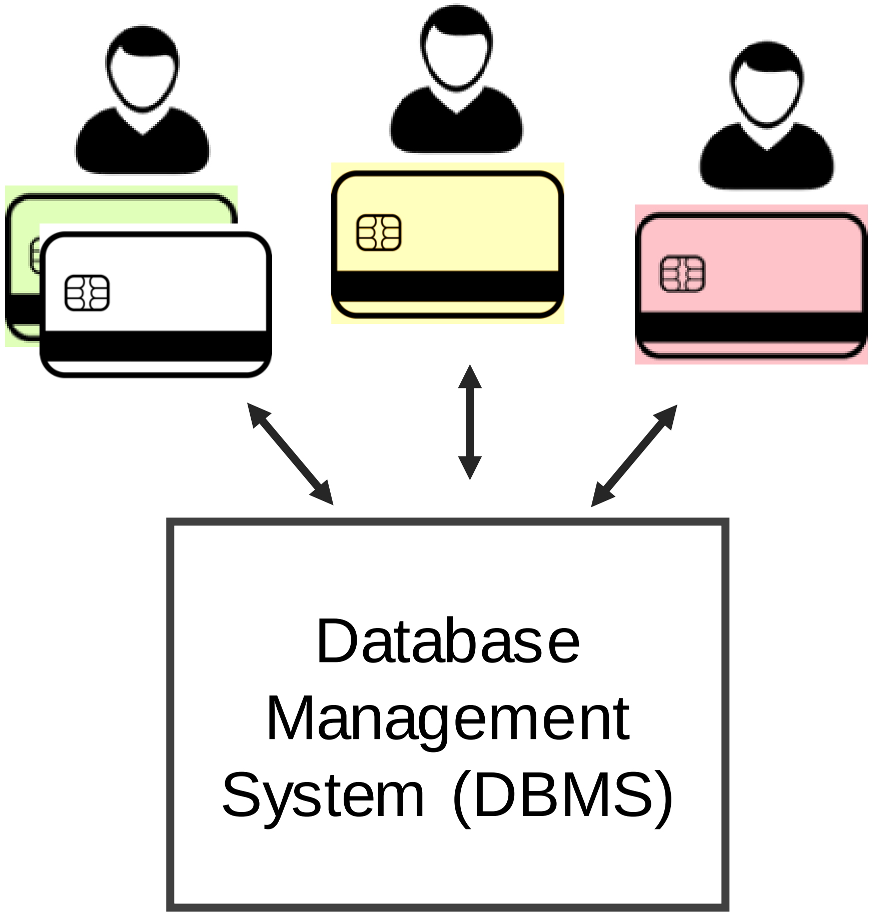
A Naïve Implementation of Transactions



Durability: The database must recover to a valid state no matter when a crash occurs

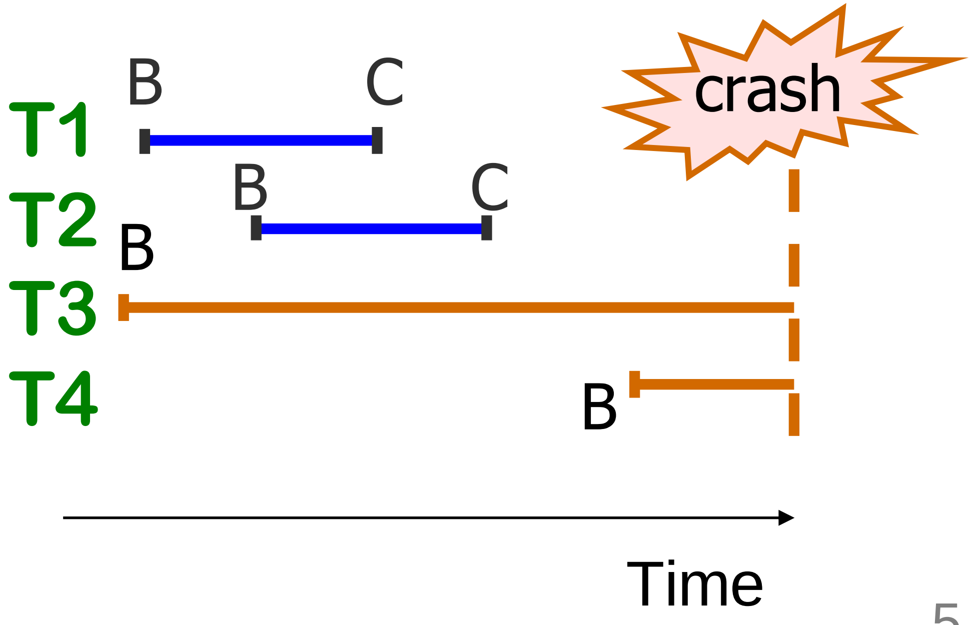
- Committed transactions should persist
- Uncommitted transactions should roll back

A Naïve Implementation of Transactions



Durability: The database must recover to a valid state no matter when a crash occurs

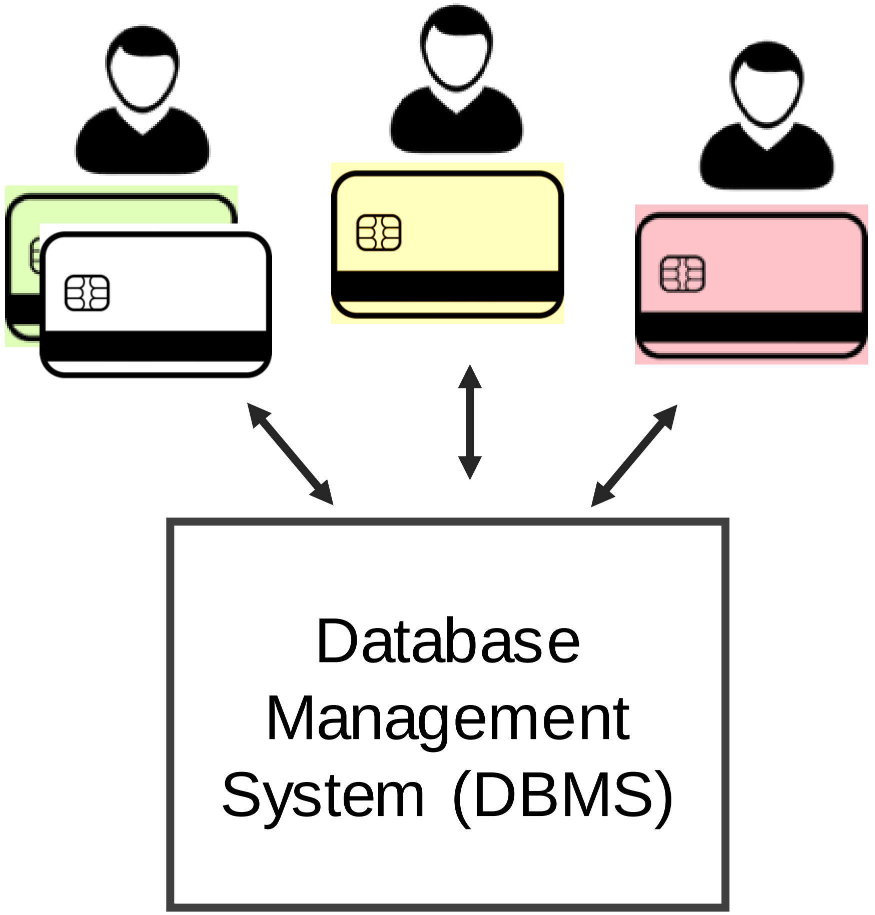
- Committed transactions should persist
- Uncommitted transactions should roll back



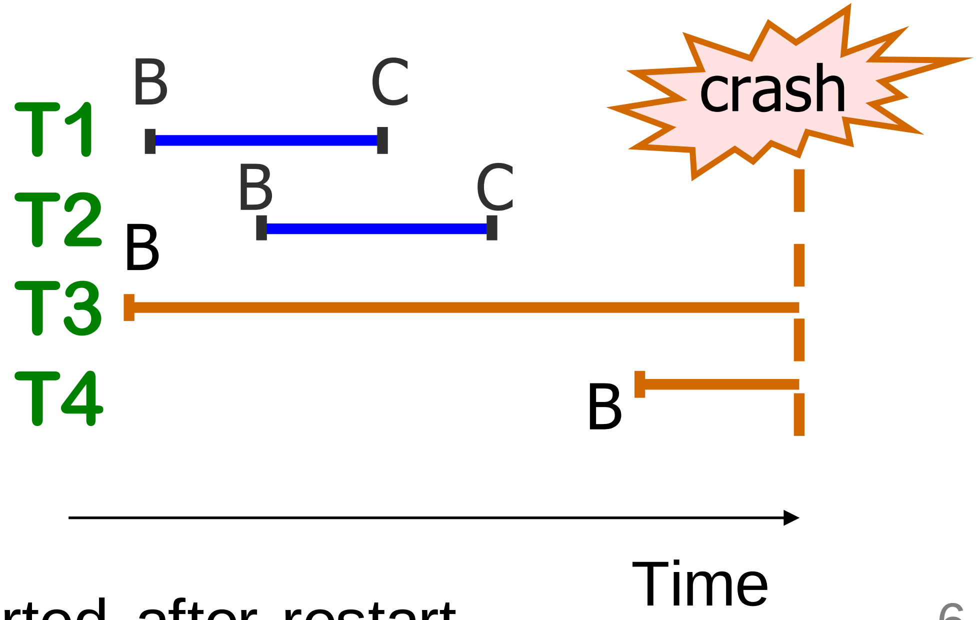
A Naïve Implementation of Transactions

Durability: The database must recover to a valid state no matter when a crash occurs

- Committed transactions should persist
- Uncommitted transactions should roll back



Database Management System (DBMS)



T1 and T2 commit;
T3, T4 should be aborted after restart

Outline

Durability

Types of failures

Failure examples

Write ahead logging (WAL)

Buffer management policies

Other discussion

Types of Failures

Software failure

- Crash of the operating system, the DBMS, or the application software

Hardware failure

- Crashes due to memory or disk errors
- Power failures
- Storage media failure

Types of Failures

Software failure

- Crash of the operating system, the DBMS, or the application software

Hardware failure

- Crashes due to memory or disk errors
- Power failures
- Storage media failure

Logging can handle these failures

Types of Failures

Software failure

- Crash of the operating system, the DBMS, or the application software

Hardware failure

- Crashes due to memory or disk errors
- Power failures
- Storage media failure

Requires data replication, beyond the scope of this class

Outline

Durability

Types of failures

Failure examples

Write ahead logging (WAL)

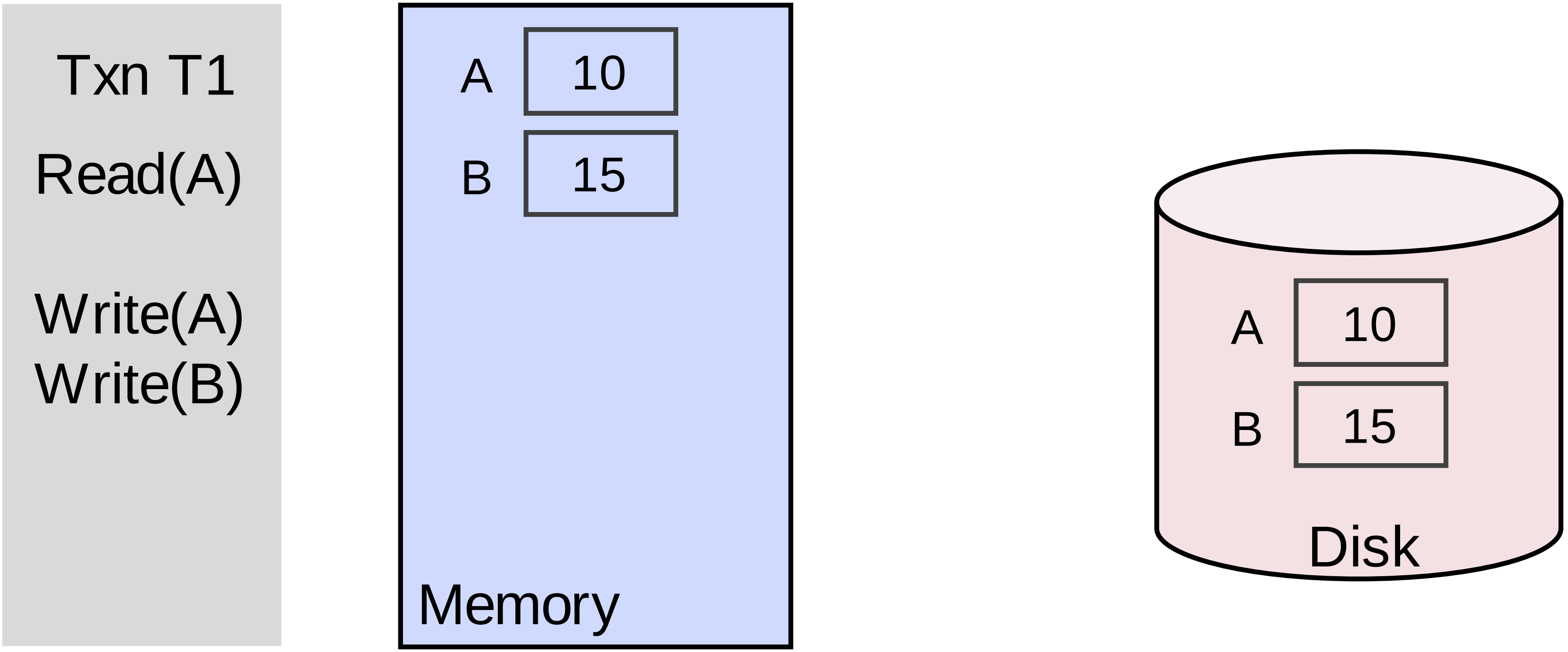
Buffer management policies

Other discussion

Failure Example 1

On a crash, data in disk persists, data in memory disappears

- Failure recovery depends on buffer management policies



Txn T1

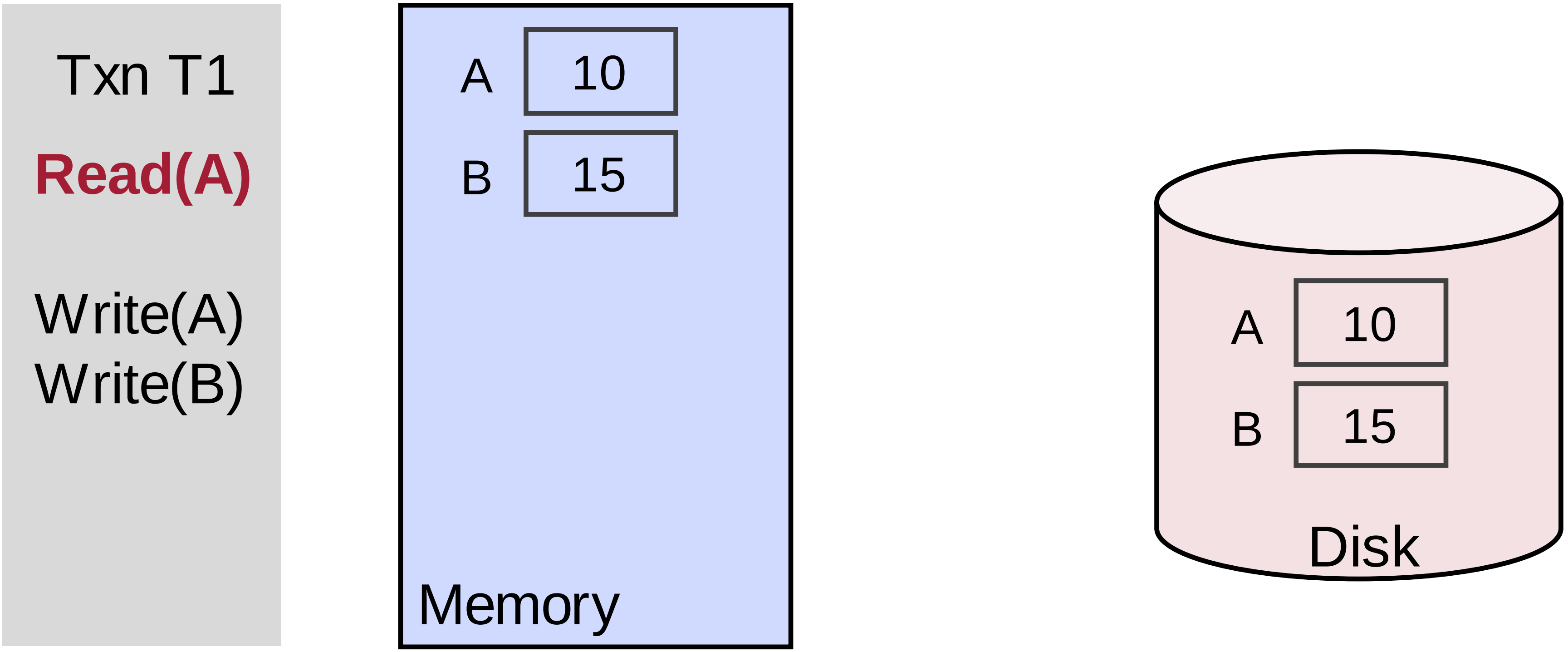
Read(A)

Write(A)
Write(B)

Failure Example 1

On a crash, data in disk persists, data in memory disappears

- Failure recovery depends on buffer management policies



Txn T1

Read(A)

Write(A)
Write(B)

A 10
B 15

Memory

A 10
B 15

Disk

Failure Example 1

On a crash, data in disk persists, data in memory disappears

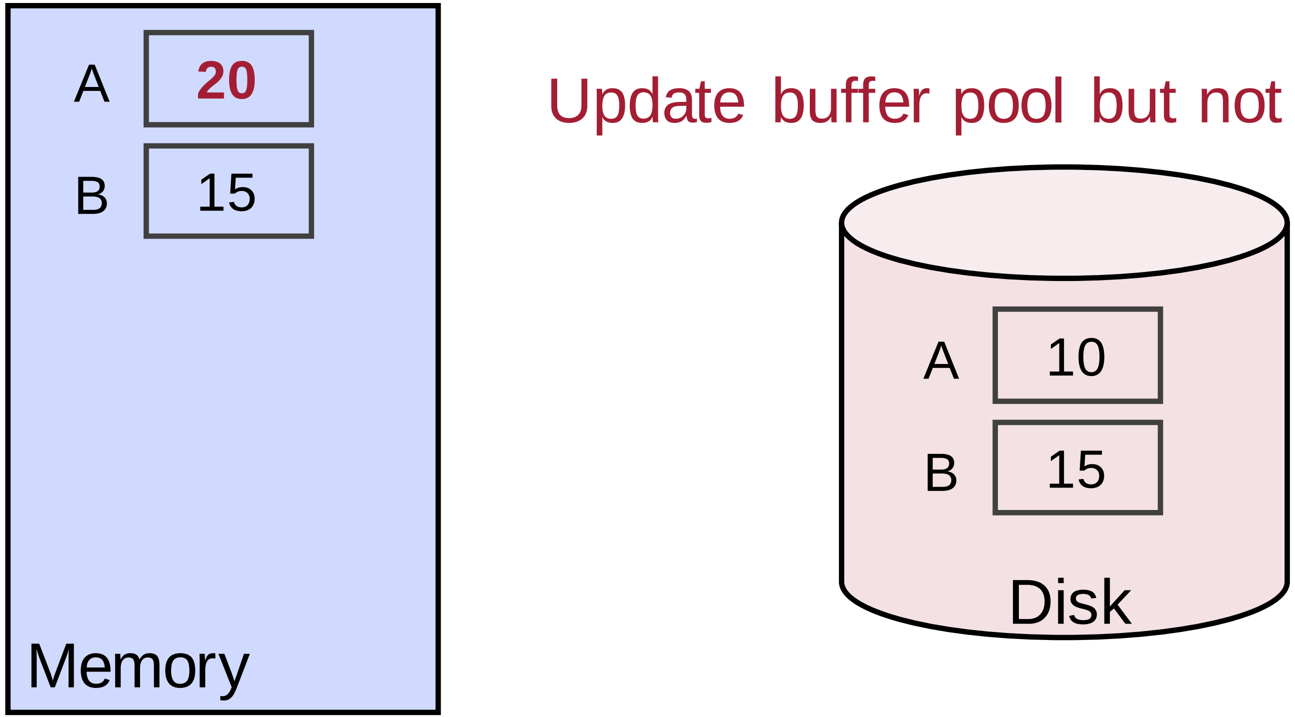
- Failure recovery depends on buffer management policies

Txn T1

Read(A)

Write(A)

Write(B)

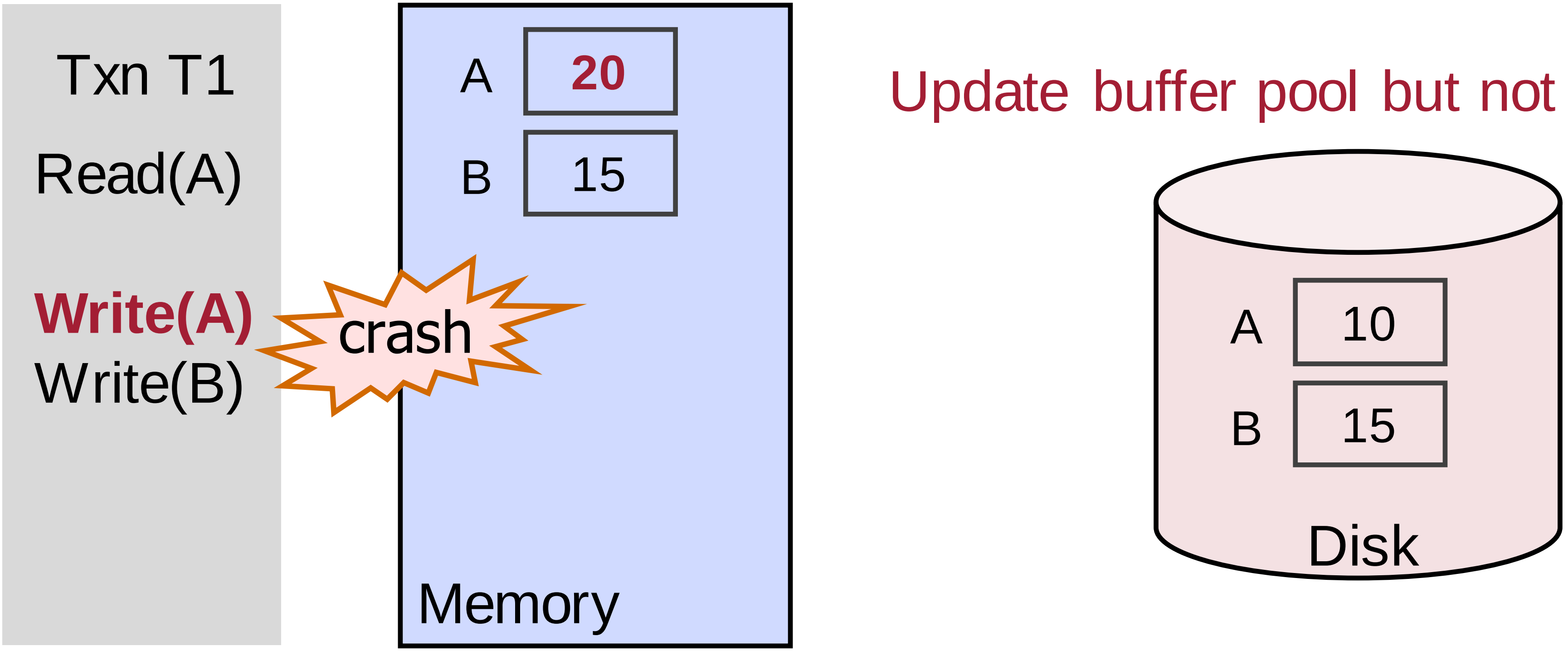


Update buffer pool but not disk

Failure Example 1

On a crash, data in disk persists, data in memory disappears

- Failure recovery depends on buffer management policies



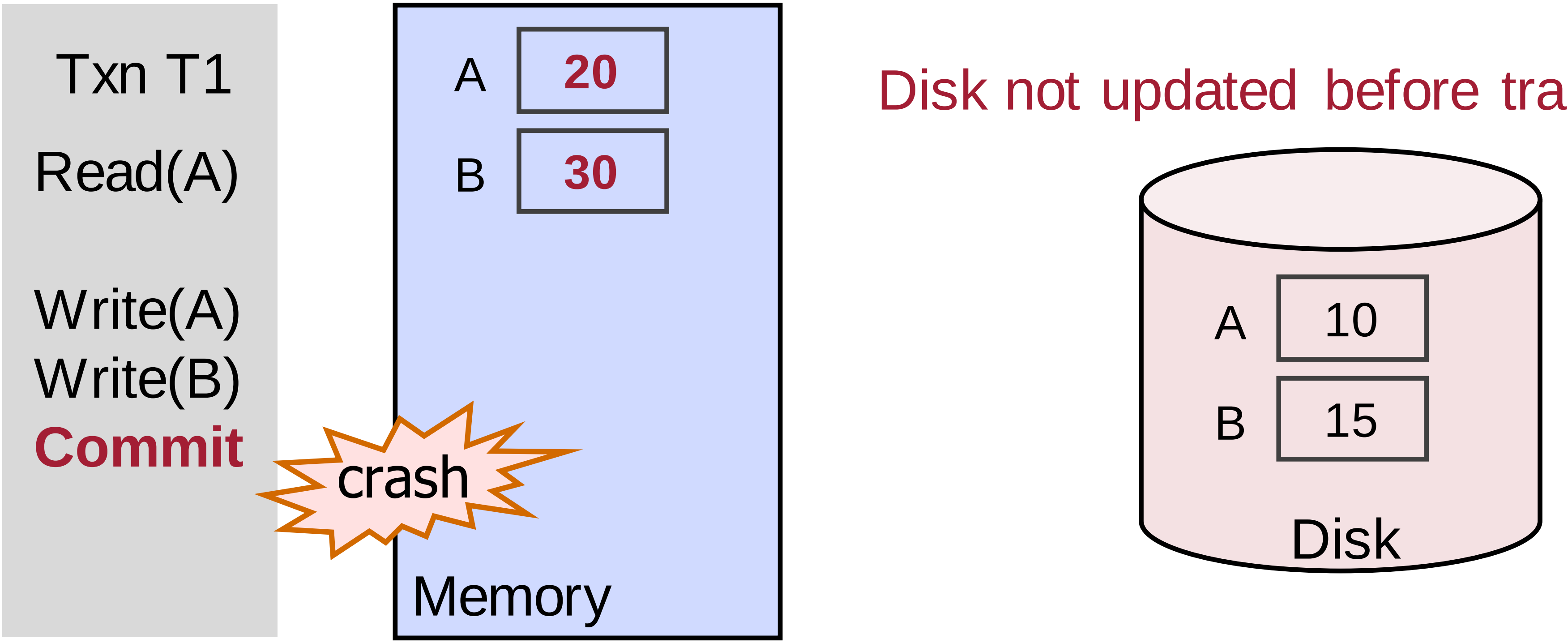
Update buffer pool but not disk

Update not reflected on disk, no need to recovery this transaction

Failure Example 2

On a crash, data in disk persists, data in memory disappears

– Failure recovery depends on buffer management policies

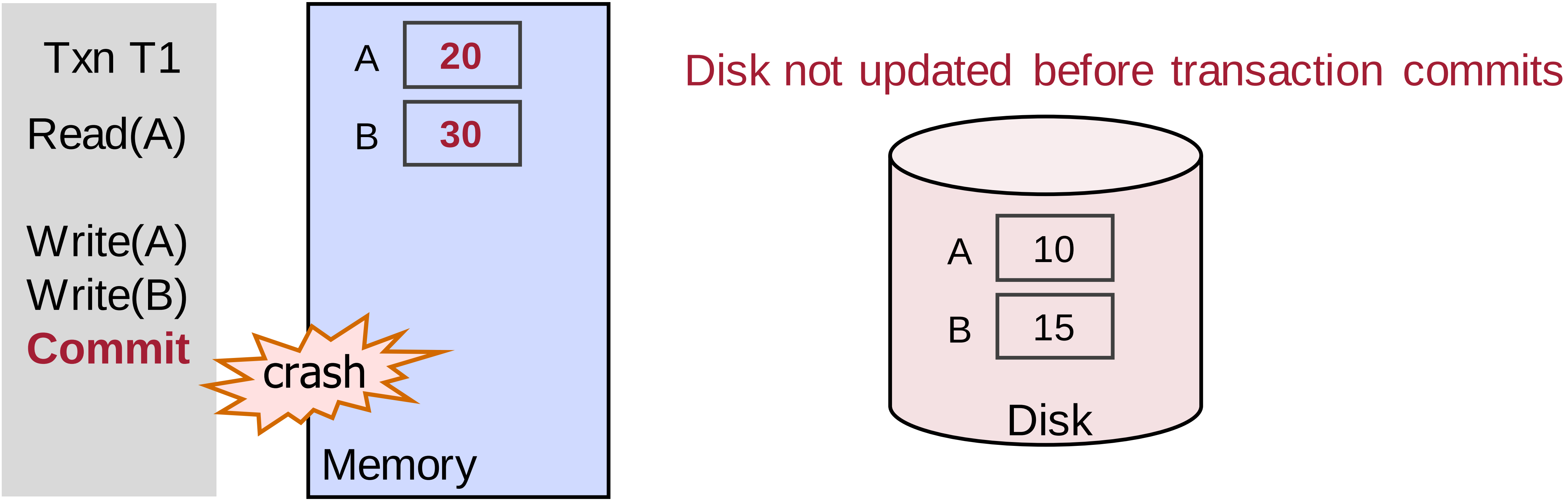


Disk not updated before transaction commits

Failure Example 2

On a crash, data in disk persists, data in memory disappears

– Failure recovery depends on buffer management policies

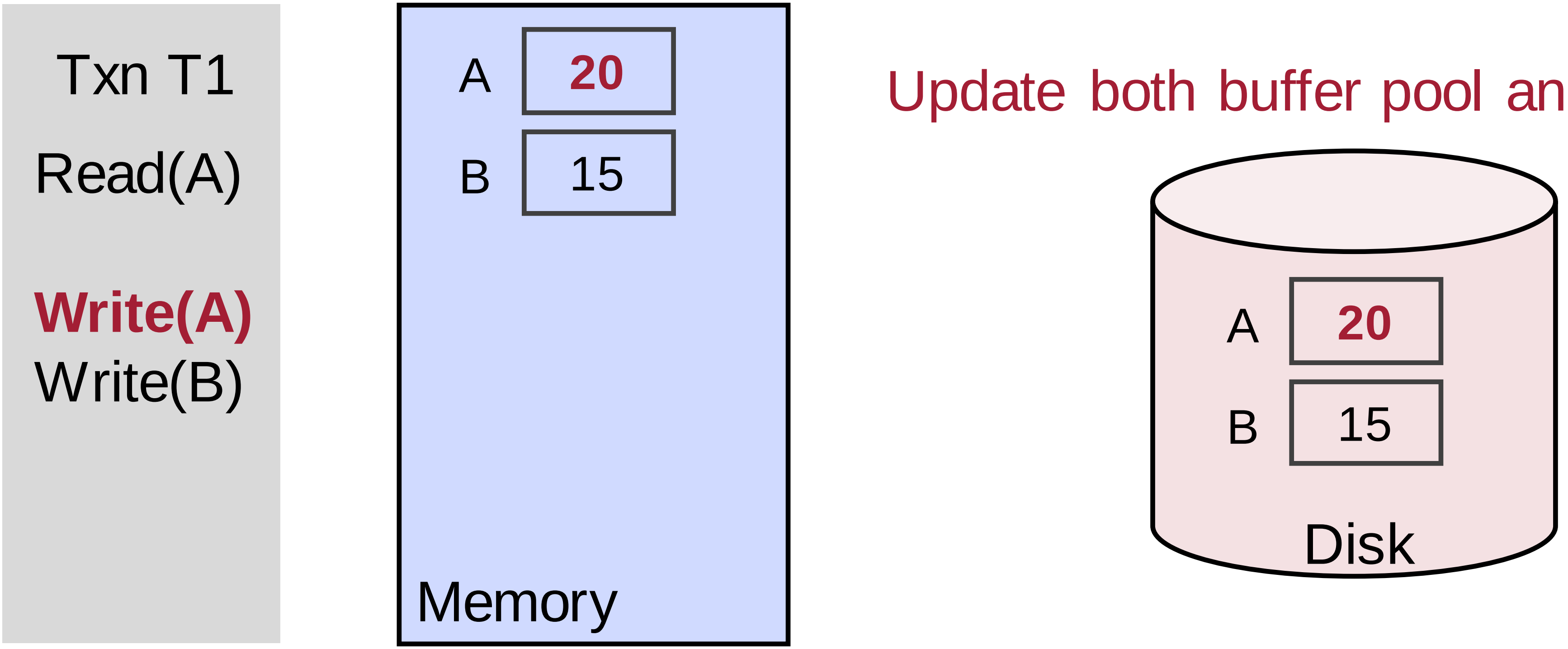


For recoverability, updates must be stored on disk before transaction commits

Failure Example 3

On a crash, data in disk persists, data in memory disappears

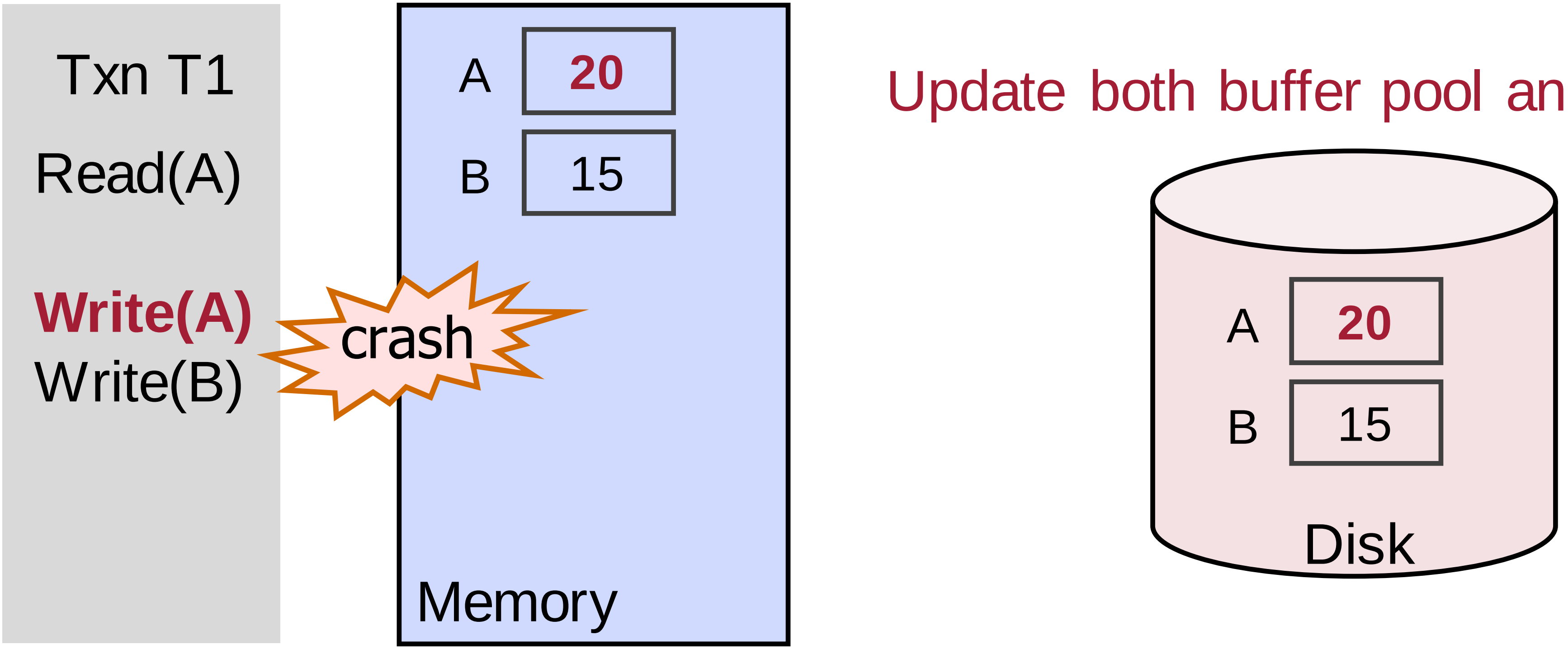
- Failure recovery depends on buffer management policies



Failure Example 3

On a crash, data in disk persists, data in memory disappears

- Failure recovery depends on buffer management policies



The transaction does not commit before failure, must rollback updates on disk

Outline

Durability

Types of failures

Failure examples

Write ahead logging (WAL)

Buffer management policies

Other discussion

Write-Ahead Logging (WAL)

Write-ahead logging

- Flush a log record before updating the data page on disk
- Log incurs only sequential IO

Write-Ahead Logging (WAL)

Write-ahead logging

- Flush a log record before updating the data page on disk
- Log incurs only sequential IO

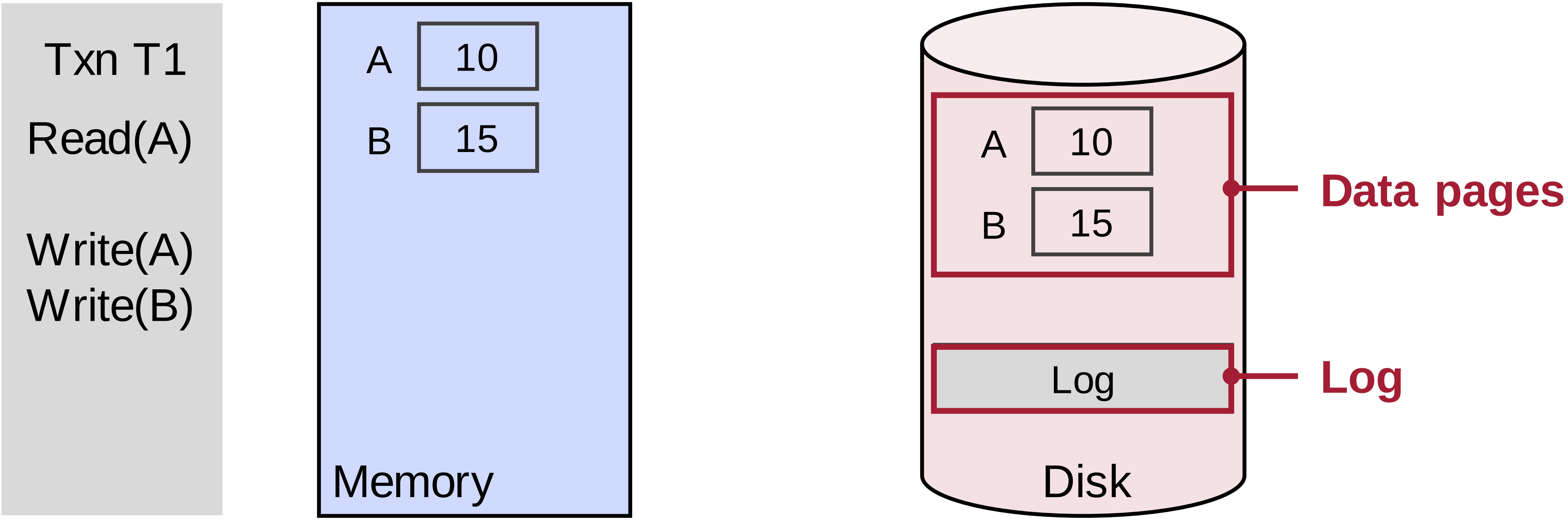
REDO information: information about the change made by the transaction (e.g., new value in the page)

UNDO information: information to reverse a change by the transaction (e.g., old value in the page)

Write-Ahead Logging (WAL)

Write-ahead logging

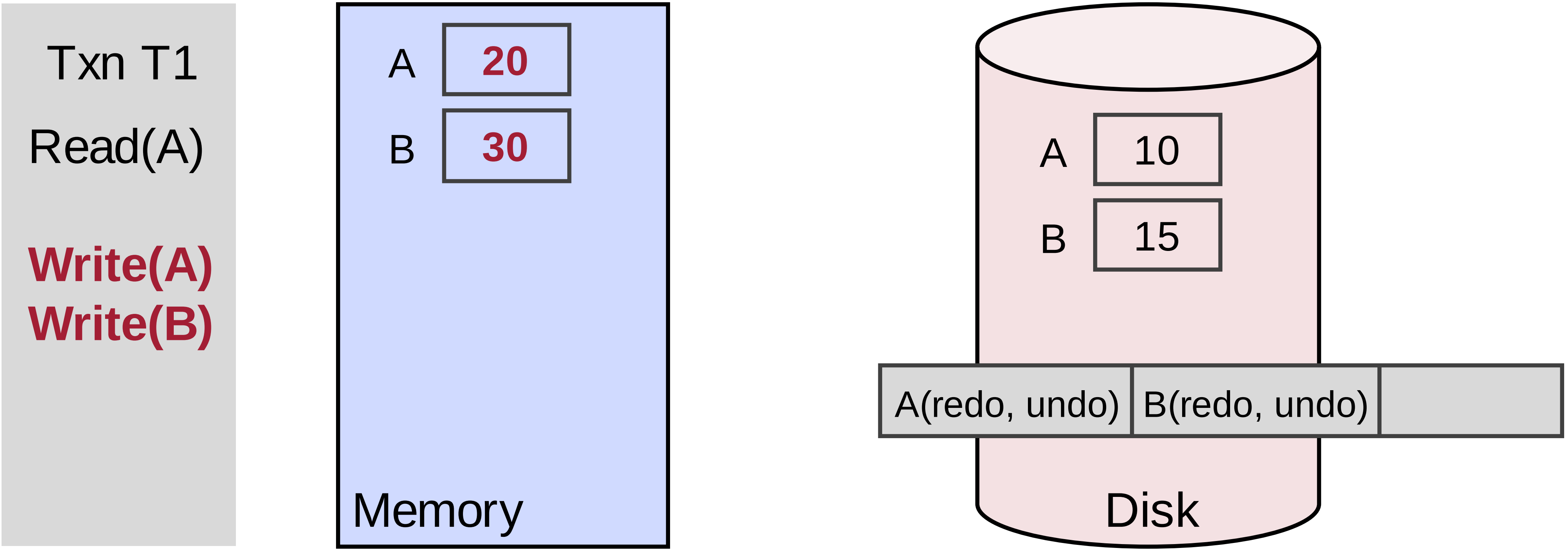
- Flush a log record before updating the data page on disk



Write-Ahead Logging (WAL)

Write-ahead logging

- Flush a log record before updating the data page on disk



Use REDO info to reapply changes; use UNDO info to reverse changes

Outline

Durability

Types of failures

Failure examples

Write ahead logging (WAL)

Buffer management policies

- Steal vs. No Steal
- Force vs. No Force

Other discussion

Buffer Management Policy

No Steal: Dirty pages stay in DRAM until the transaction commits

Steal: Dirty pages can be flushed to disk before the transaction commits

Why called steal? Another transaction can “steal” the buffer slot that the current transaction is using and thus the current transaction will flush the dirty page to disk

Buffer Management Policy

No Steal: Dirty pages stay in DRAM until the transaction commits

- No need to reverse changes during recovery
- No need to log UNDO information

Steal: Dirty pages can be flushed to disk before the transaction commits

Buffer Management Policy

No Steal: Dirty pages stay in DRAM until the transaction commits

- No need to reverse changes during recovery
- No need to log UNDO information

Steal: Dirty pages can be flushed to disk before the transaction commits

- Log UNDO information so that dirty changes can be reversed during recovery
- **Advantage**: other transactions can use the buffer slot in DRAM
- **Disadvantage**: need to log UNDO information

Buffer Management Policy

Force: All dirty pages must be flushed before the transaction commits

No Force: Dirty pages may stay in memory after the transaction commits

Buffer Management Policy

Force: All dirty pages must be flushed before the transaction commits

- No need to replay changes during recovery
- No need to log REDO information

No Force: Dirty pages may stay in memory after the transaction commits

Buffer Management Policy

Force: All dirty pages must be flushed before the transaction commits

- No need to replay changes during recovery
- No need to log REDO information

No Force: Dirty pages may stay in memory after the transaction commits

- Log REDO information so that committed changes can be replayed during recovery
- **Advantage**: can reduce random disk IO
- **Disadvantage**: need to log REDO information

Buffer Management Policy — Summary

	Steal	No Steal
Force		
No Force		**REDO** only

Buffer Management Policy

	Steal	No Steal
Force		
No Force		**REDO** only

REDO only (e.g., main memory database)

- – Log REDO record before each update to database
- – Log COMMIT record when a transaction finishes execution
- – Modified data pages can be written to disk after COMMIT record is logged
- – **Recovery**: replay REDO records for committed transactions

Buffer Management Policy

	Steal	No Steal
Force	**UNDO** only	
No Force		**REDO** only

UNDO only (e.g., NVM-based database)

- – Log UNDO record before each update to database
- – Log COMMIT record when a transaction finishes execution
- – Modified data pages can be written to disk after the corresponding UNDO record but before the COMMIT record
- – **Recovery**: UNDO uncommitted transactions

Buffer Management Policy

	Steal	No Steal
Force	**UNDO** only	No REDO nor UNDO
No Force		**REDO** only

No REDO and no UNDO

- Atomically write all updates to disk
- Check out [1] if you are interested in more details

[1] Philip Bernstein, Vassos Hadzilacos, Nathan Goodman, *Concurrency Control and Recovery in Database Systems*, 1987

Buffer Management Policy

	Steal	No Steal
Force	**UNDO** only	No REDO nor UNDO
No Force	**REDO and UNDO** logging (ARIES [2])	**REDO** only

REDO and UNDO (the most flexible!)

- Modified data pages can be written to disk any time after the corresponding log record
- Log both REDO and UNDO before updating data pages
- Log COMMIT record when a transaction finishes execution
- **Recovery**: UNDO uncommitted transactions and REDO committed transactions

[2] C. Mohan, et al. ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging. ACM Trans. Database Syst. 1992.

Outline

Durability

Types of failures

Failure examples

Write ahead logging (WAL)

Buffer management policies

Other discussion

- Checkpoints
- Group commit
- Physical vs. logical logging

Checkpoints

The log file can grow unbounded over time

Solution: **checkpoint**

Naïve implementation of a checkpoint

- Stop admitting new transactions and commit/abort all ongoing transactions
- Flush all dirty pages to disk
- Log CHECKPOINT (Can truncate the log at this point)
- Resume execution

Group Commit

Flushing a log record to disk for each write incurs significant performance overhead

Observation: Database log is a sequential structure that captures log records from all transactions

Group commit flushes multiple log records with a single disk IO

- **Tradeoff**: each write has a slightly higher latency, but the logging throughput is significantly improved

Physical vs. Logical Logging

Physical logging

- Record the byte changes
- E.g., REDO contains the value in the new page; UNDO contains the value in the old page

Logical logging

- Record the logical operations (UPDATE, INSERT, DELETE)
- **Advantages**: consumes less disk space than physical logging

Physiological logging

- Physical to a page, and logical within a page

Summary

Durability

Types of failures

Failure examples

Write ahead logging (WAL)

Buffer management policies

- Steal vs. No Steal
- Force vs. No Force

Other discussion

- Checkpoints
- Group commit
- Physical vs. logical logging