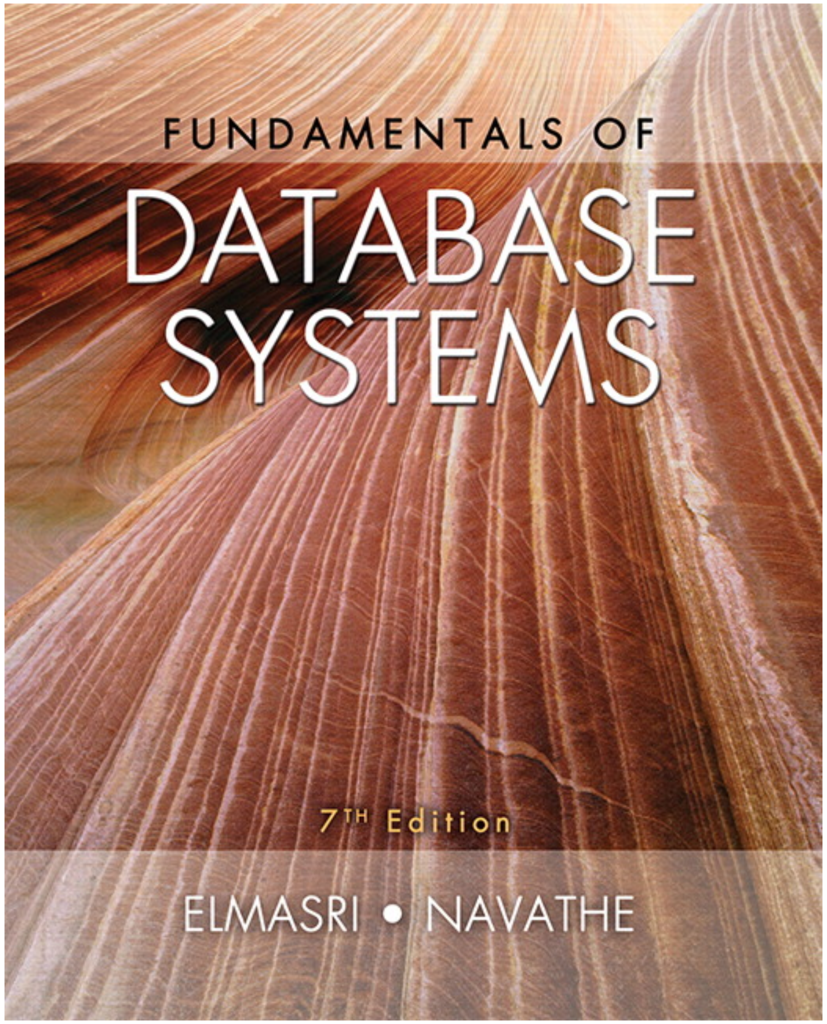


Copyright © 2016 Ramez Elmasri and Shamkant B. Navathe

CHAPTER 22

Database Recovery Techniques

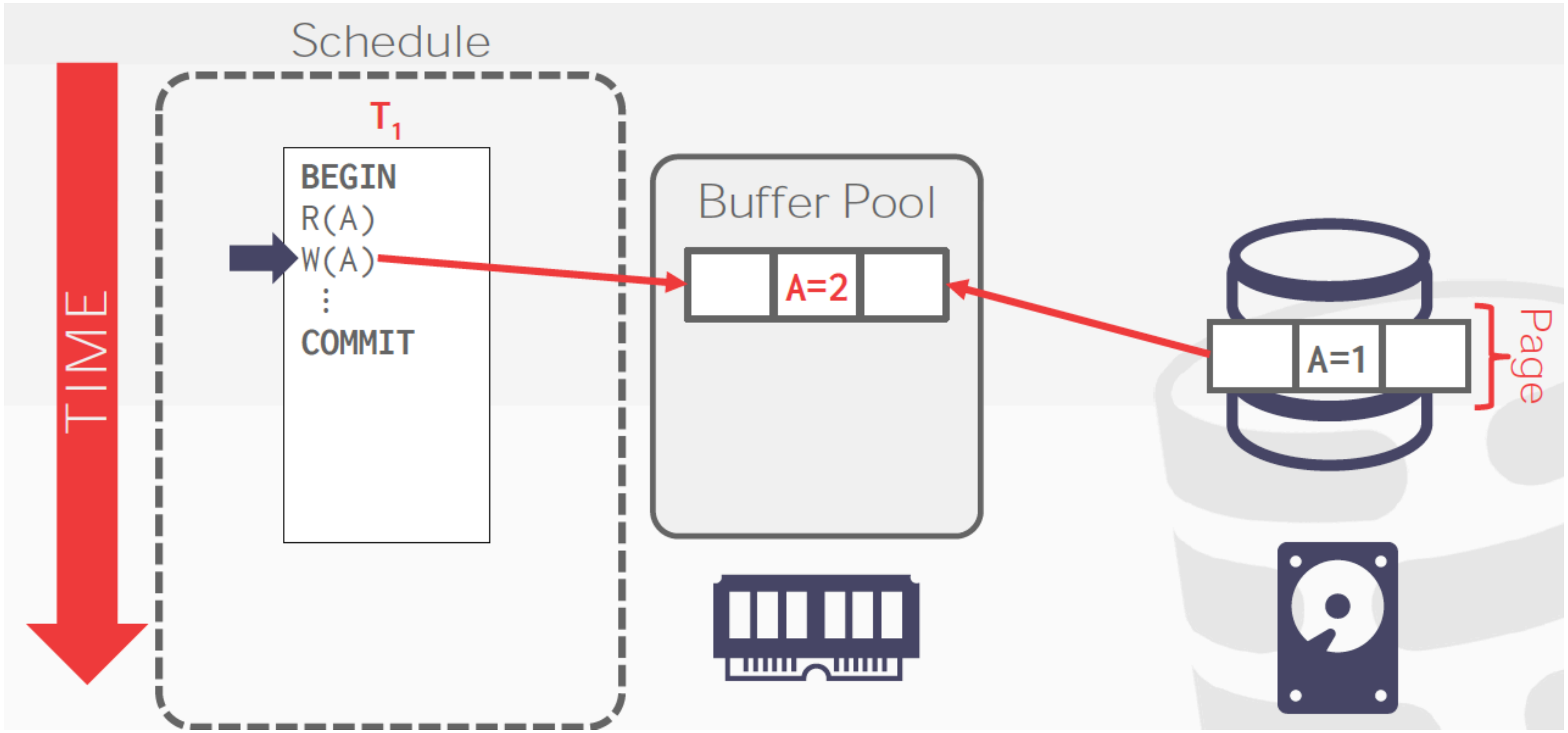
Copyright © 2016 Ramez Elmasri and Shamkant B. Navathe

Introduction

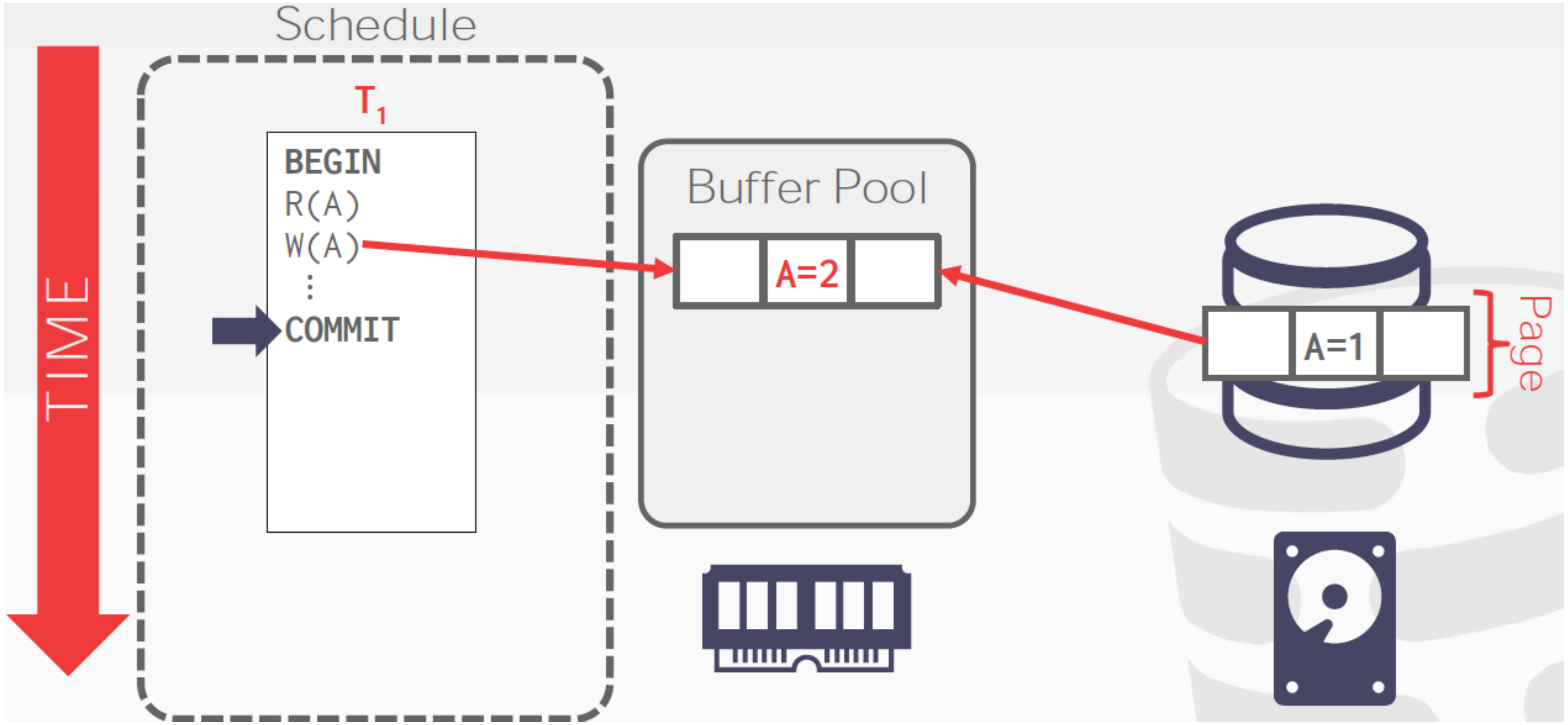
- Recovery algorithms
- Recovery concepts
  - Write-ahead logging
  - In-place versus shadow updates
  - Rollback
  - Deferred update
  - Immediate update
- Certain recovery techniques best used with specific concurrency control methods

Copyright © 2016 Ramez Elmasri and Shamkant B. Navathe

Motivation

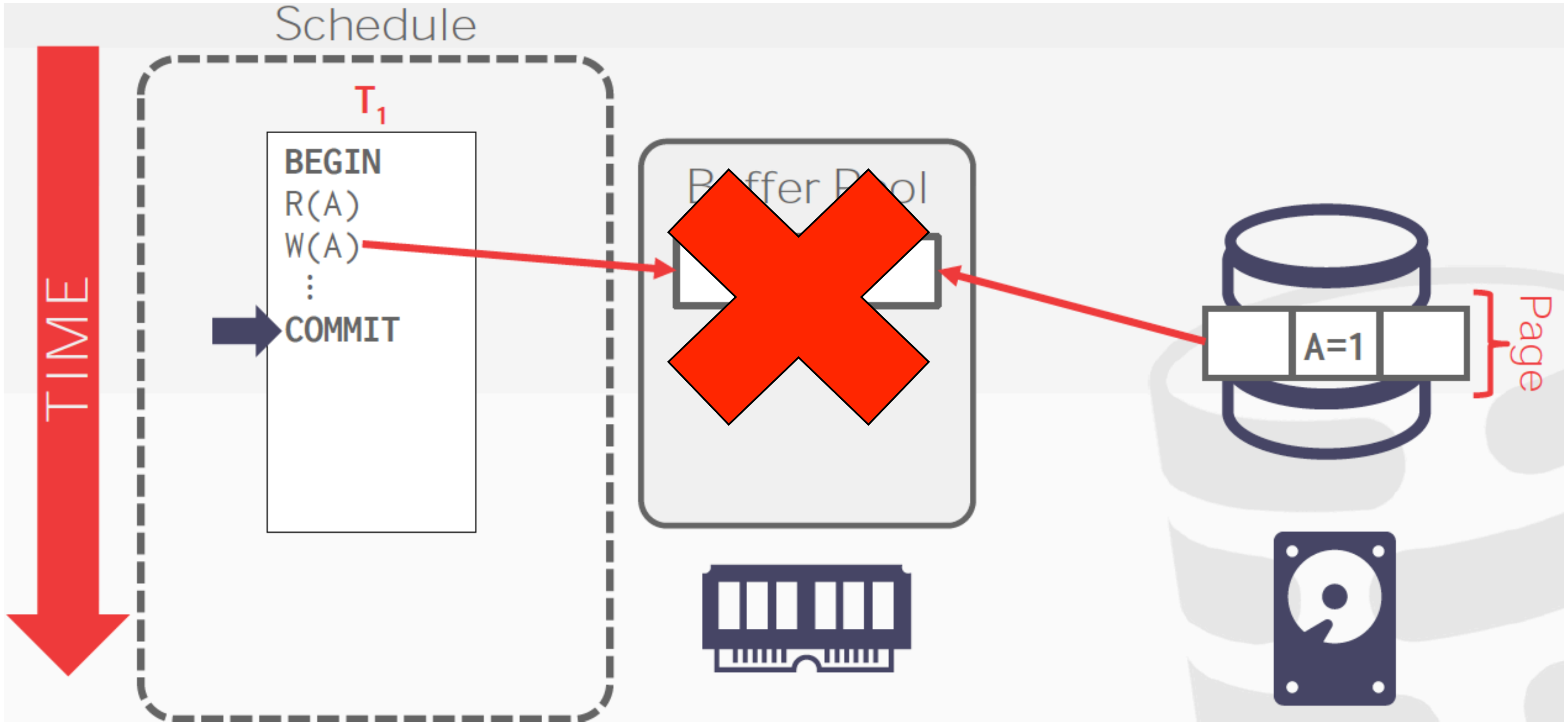


Motivation



slide from CMU 15-445/645

Motivation



22.1 Recovery Concepts

- Recovery process restores database to most recent consistent state before time of failure
- Information kept in system log
- Typical recovery strategies
  - Restore backed-up copy of database
    - Best in cases of extensive damage
  - Identify any changes that may cause inconsistency
    - Best in cases of noncatastrophic failure
    - Some operations may require redo

Copyright © 2016 Ramez Elmasri and Shamkant B. Navathe

Recovery Concepts (cont'd.)

- Deferred update techniques
  - Do not physically update the database until after transaction commits
  - Undo is not needed; redo may be needed
- Immediate update techniques
  - Database may be updated by some operations of a transaction before it reaches commit point
  - Operations also recorded in log
  - Recovery still possible

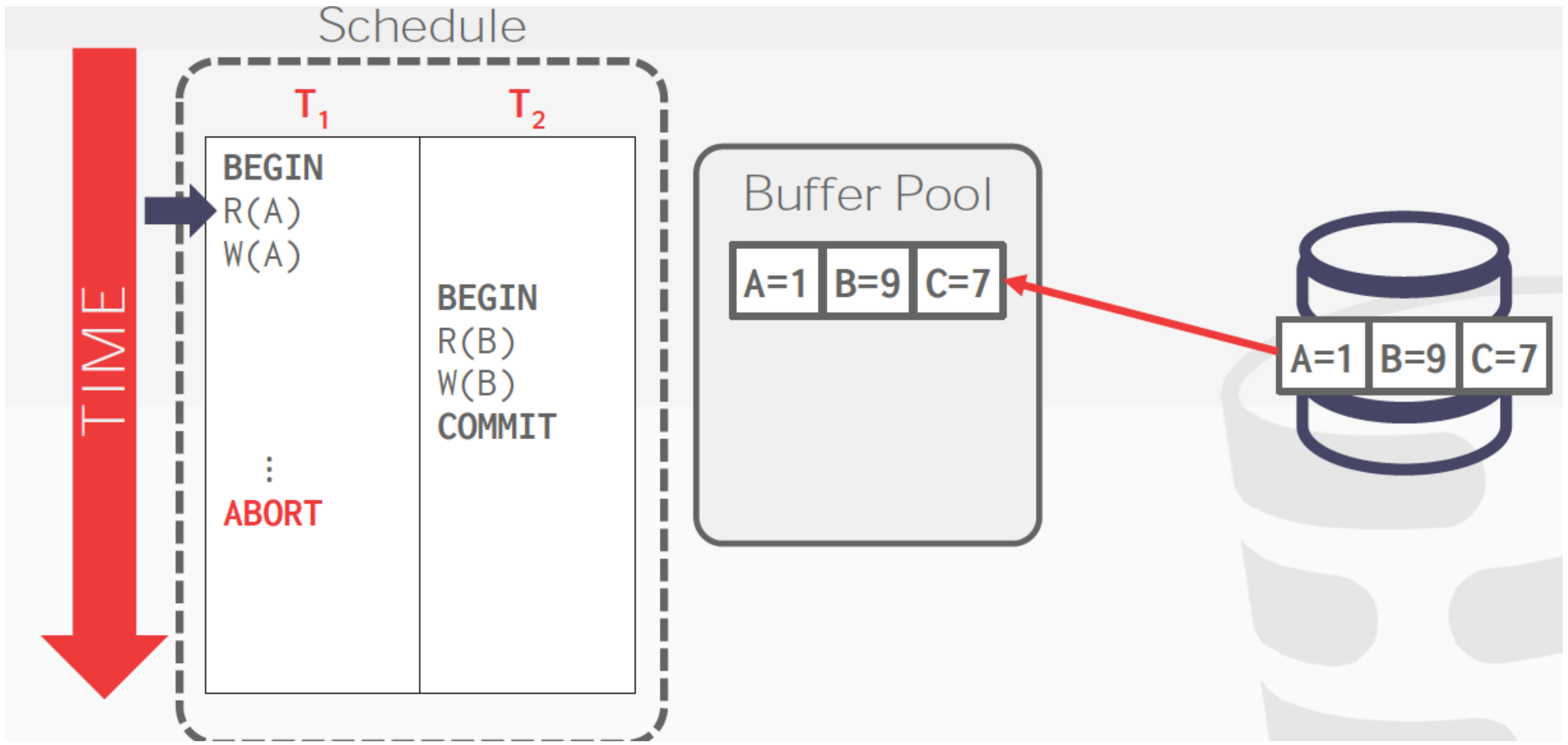
Copyright © 2016 Ramez Elmasri and Shamkant B. Navathe

Recovery Concepts (cont'd.)

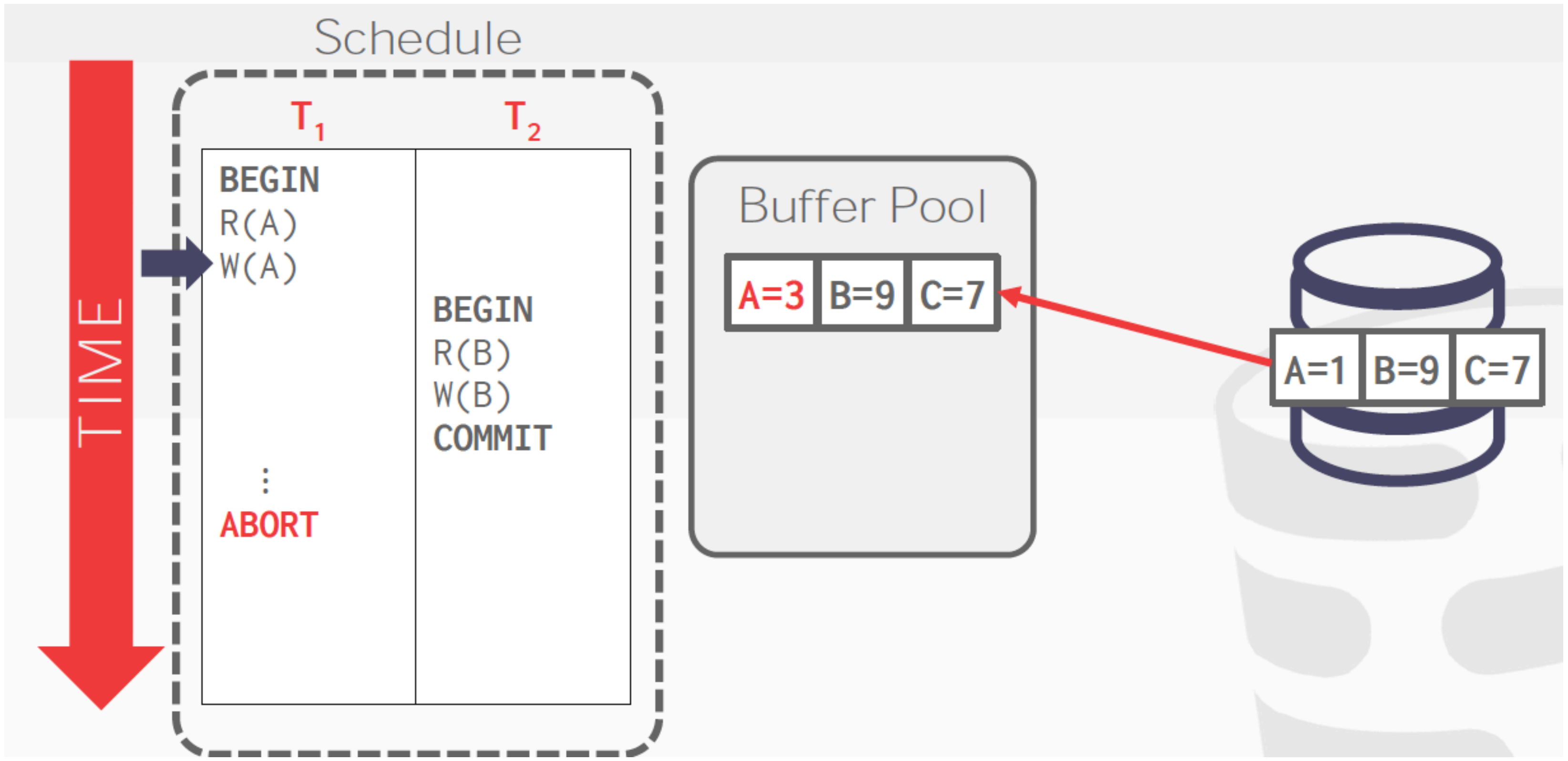
- Undo and redo operations required to be idempotent
  - Executing operations multiple times equivalent to executing just once
  - Entire recovery process should be idempotent
- Caching (buffering) of disk blocks
  - DBMS cache: a collection of in-memory buffers
  - Cache directory keeps track of which database items are in the buffers

Copyright © 2016 Ramez Elmasri and Shamkant B. Navathe

Buffer Pool

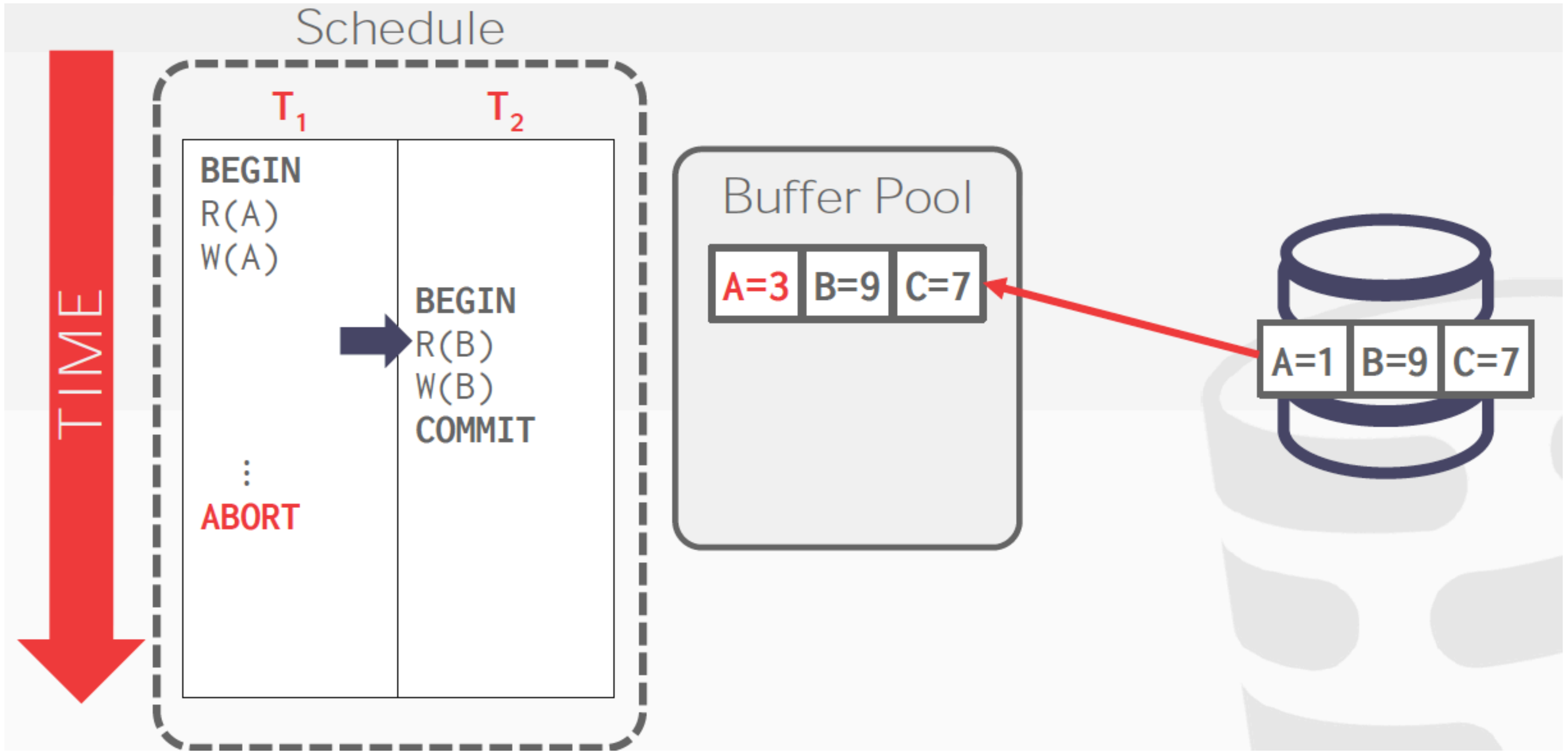

Schedule
TIME
T1
T2
BEGIN
R(A)
W(A)
⋮
ABORT
BEGIN
R(B)
W(B)
COMMIT
Buffer Pool
A=1
B=9
C=7
A=1
B=9
C=7

Buffer Pool



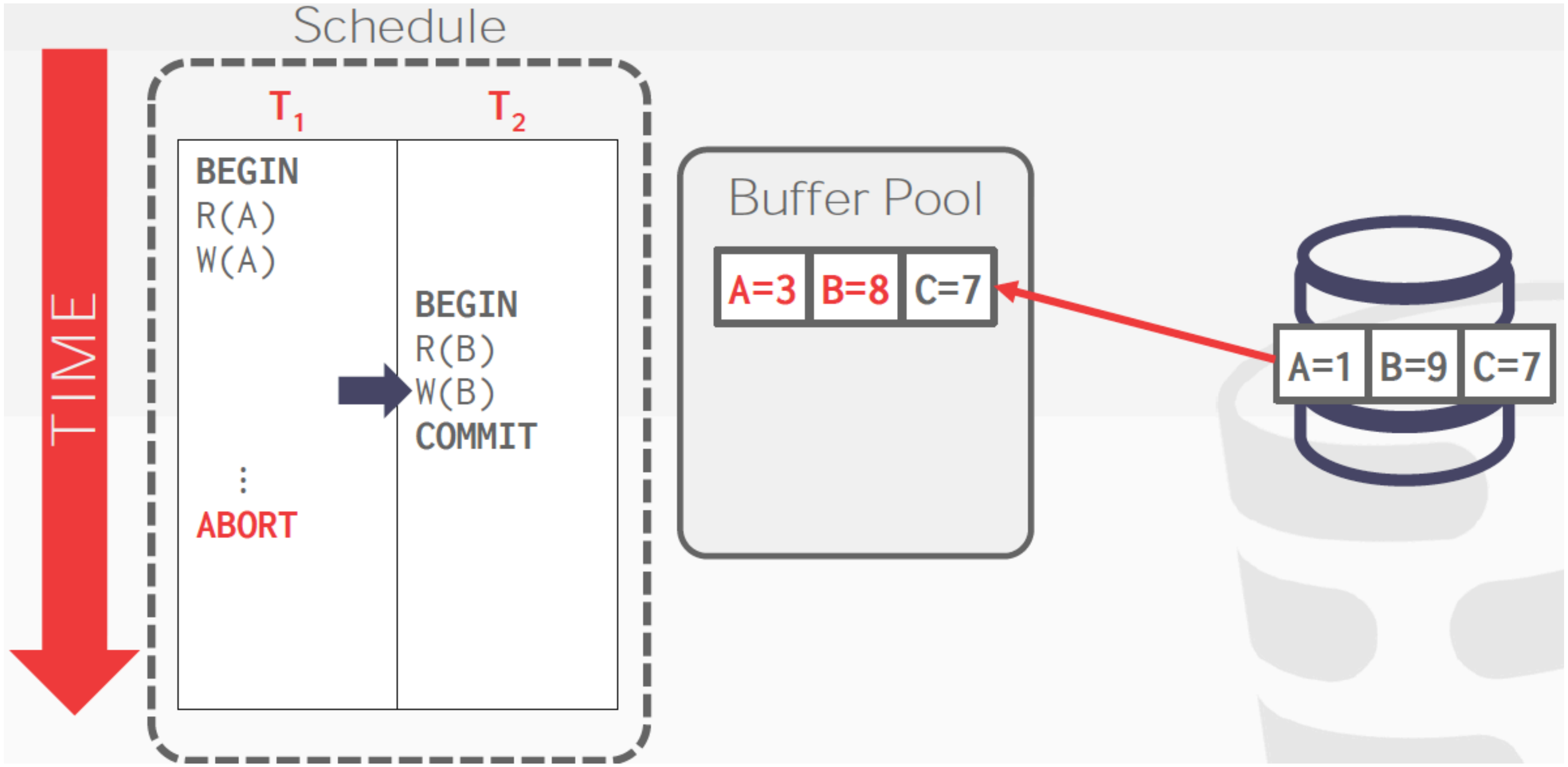
slide from CMU 15-445/645

Buffer Pool

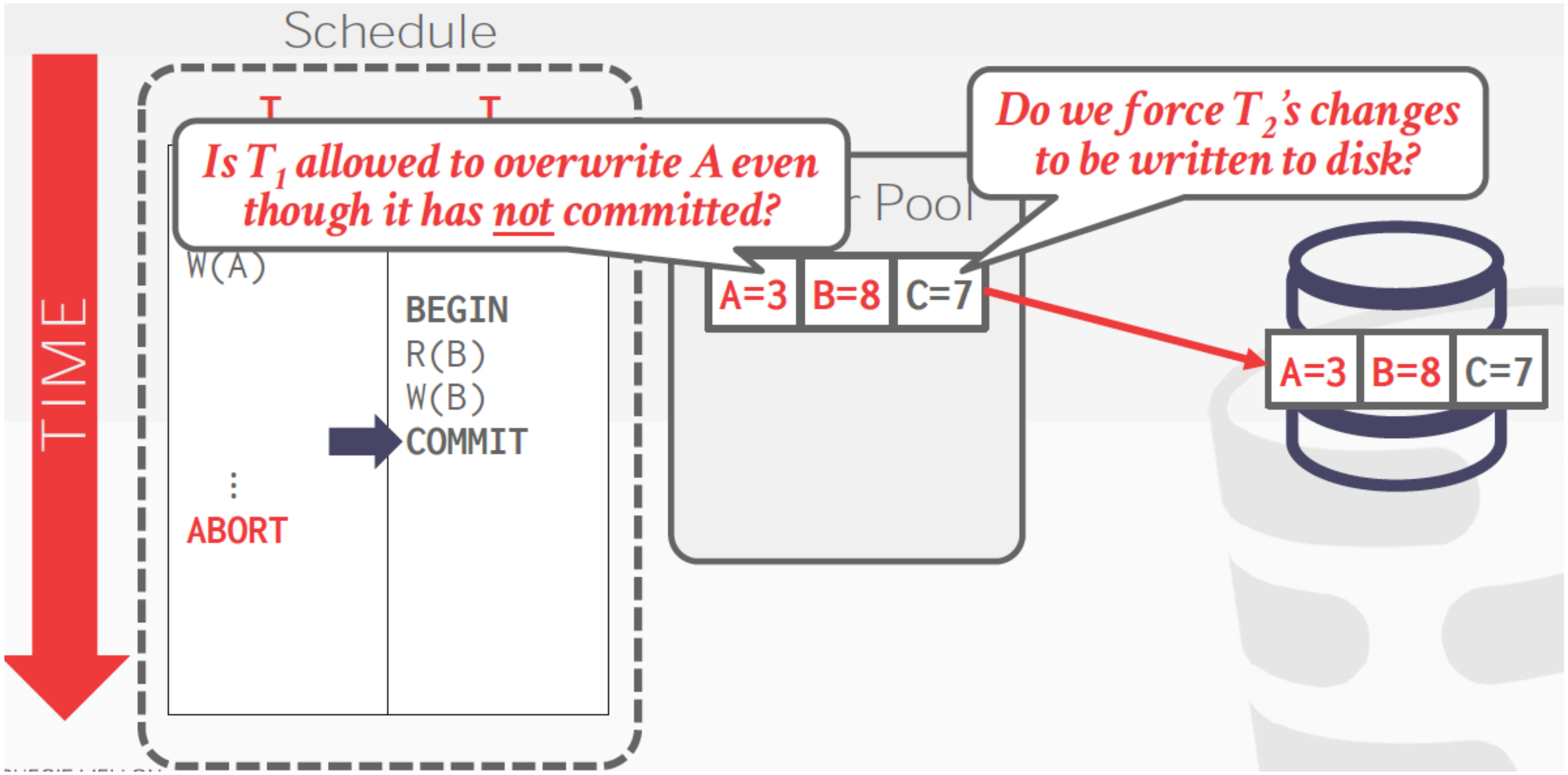


slide from CMU 15-445/645

Buffer Pool


Schedule
TIME
T$_1$
T$_2$
BEGIN
R(A)
W(A)
⋮
ABORT
BEGIN
R(B)
W(B)
COMMIT
Buffer Pool
A=3
B=8
C=7
A=1
B=9
C=7

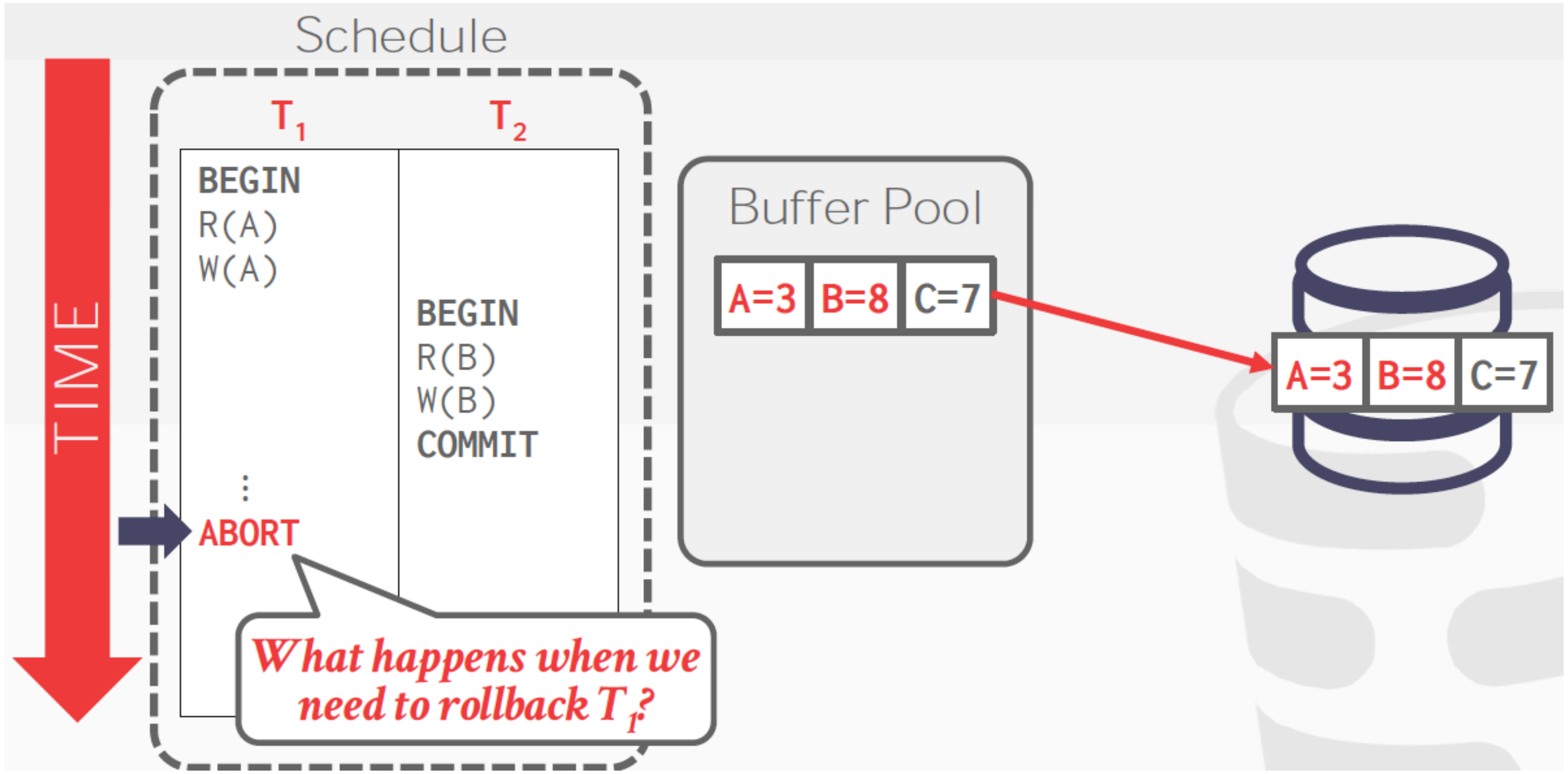
Buffer Pool



Is T_1 allowed to overwrite A even though it has <u>not</u> committed?

Do we force T_2's changes to be written to disk?

Buffer Pool



slide from CMU 15-445/645

Recovery Concepts (cont'd.)

- Cache buffers replaced (flushed) to make space for new items
- Dirty bit associated with each buffer in the cache
  - Indicates whether the buffer has been modified
- Contents written back to disk before flush if dirty bit equals one
- Pin-unpin bit
  - Page is pinned if it cannot be written back to disk yet

Copyright © 2016 Ramez Elmasri and Shamkant B. Navathe

Recovery Concepts (cont'd.)

- Main strategies
  - In-place updating
    - Writes the buffer to the same original disk location
    - Overwrites old values of any changed data items
  - Shadowing
    - Writes an updated buffer at a different disk location, to maintain multiple versions of data items
    - Not typically used in practice
- Before-image: old value of data item
- After-image: new value of data item

Copyright © 2016 Ramez Elmasri and Shamkant B. Navathe

Recovery Concepts (cont'd.)

- Write-ahead logging
  - Ensure the before-image (BFIM) is recorded
  - Appropriate log entry flushed to disk
  - Necessary for UNDO operation if needed
- UNDO-type log entries
- REDO-type log entries

Copyright © 2016 Ramez Elmasri and Shamkant B. Navathe

Recovery Concepts (cont'd.)

- Steal/no-steal and force/no-force
  - Specify rules that govern when a page from the database cache can be written to disk
- No-steal approach
  - Cache buffer page updated by a transaction cannot be written to disk before the transaction commits
- Steal approach
  - Recovery protocol allows writing an updated buffer before the transaction commits

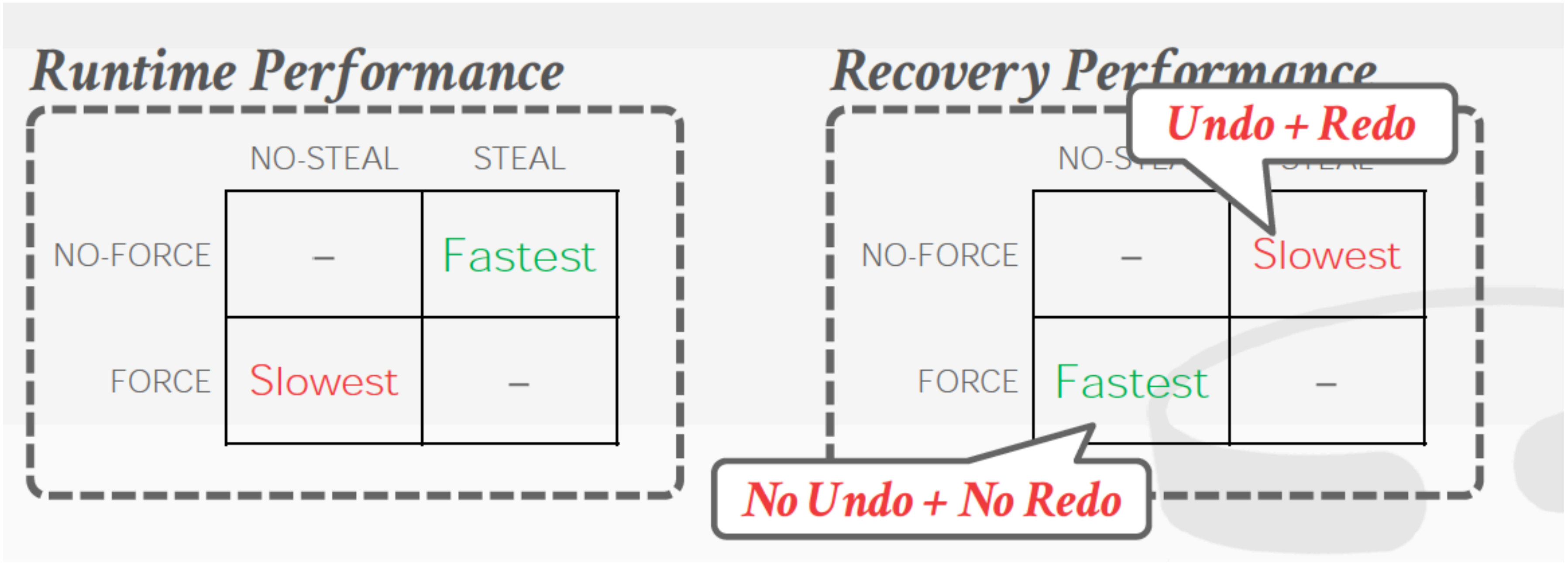
Copyright © 2016 Ramez Elmasri and Shamkant B. Navathe

Recovery Concepts (cont'd.)

- Force approach
  - All pages updated by a transaction are immediately written to disk before the transaction commits
  - Otherwise, no-force

Copyright © 2016 Ramez Elmasri and Shamkant B. Navathe

Recovery Concepts (cont'd.)



Runtime Performance

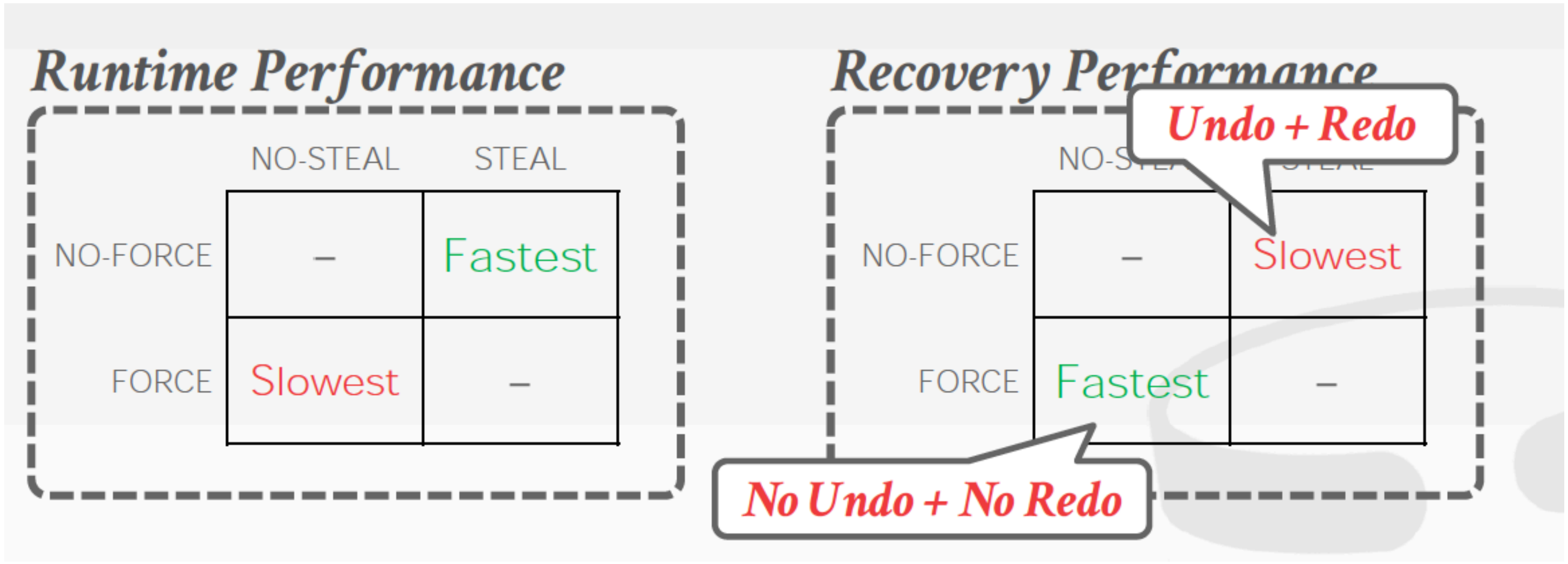
	NO-STEAL	STEAL
NO-FORCE	–	Fastest
FORCE	Slowest	–

Recovery Performance

	NO-STEAL	STEAL
NO-FORCE	–	Slowest
FORCE	Fastest	–

Undo + Redo
No Undo + No Redo

Recovery Concepts (cont’d.)



Runtime Performance

	NO-STEAL	STEAL
NO-FORCE	–	Fastest
FORCE	Slowest	–

Recovery Performance

	NO-STEAL	STEAL
NO-FORCE	–	Slowest
FORCE	Fastest	–

Undo + Redo
No Undo + No Redo

- Typical database systems employ a steal/no-force strategy
  - Avoids need for very large buffer space
  - Reduces disk I/O operations for heavily updated pages

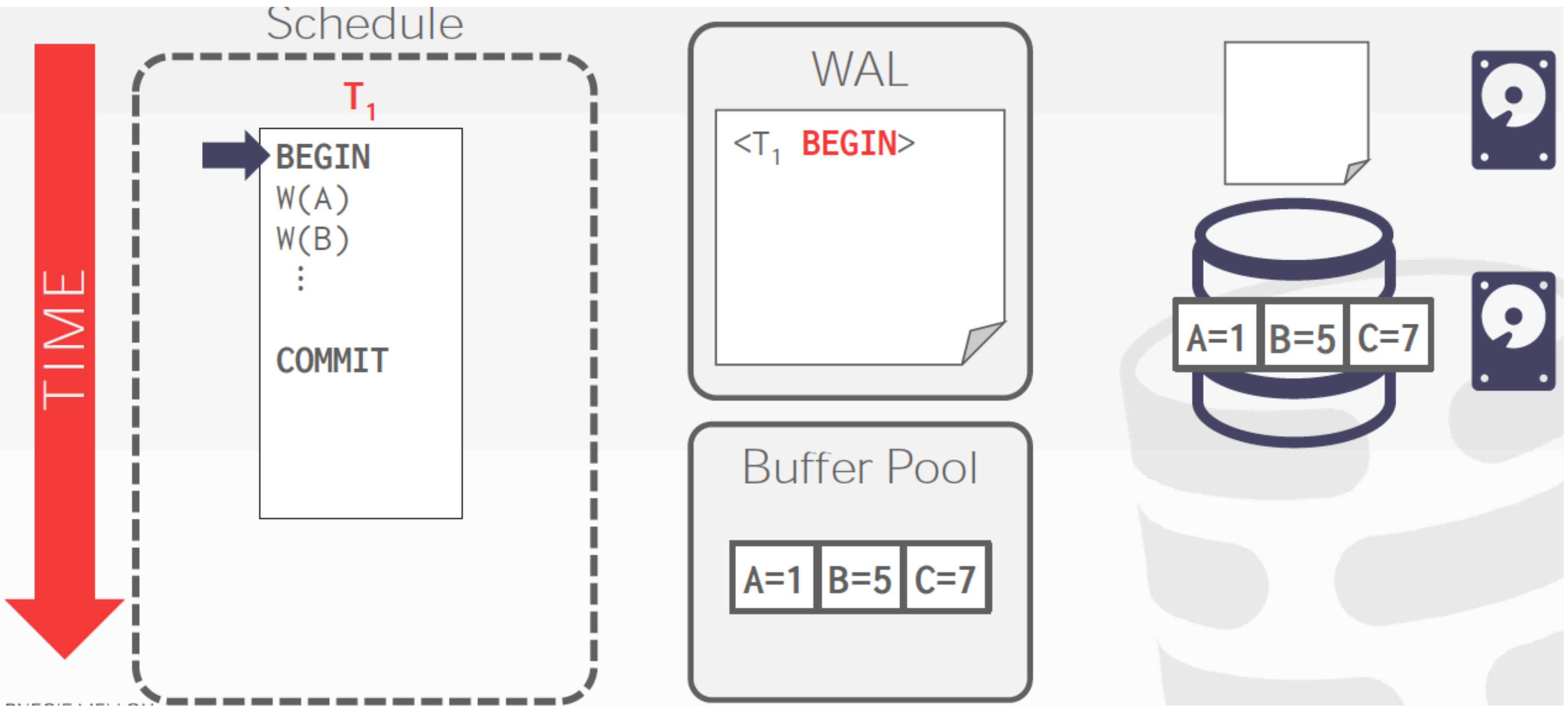
slide from CMU 15-445/645

Recovery Concepts (cont'd.)

- Write-ahead logging protocol for recovery algorithm requiring both UNDO and REDO
  - BFIM of an item cannot be overwritten by its after image until all UNDO-type log entries have been force-written to disk
  - Commit operation of a transaction cannot be completed until all REDO-type and UNDO-type log records for that transaction have been force-written to disk

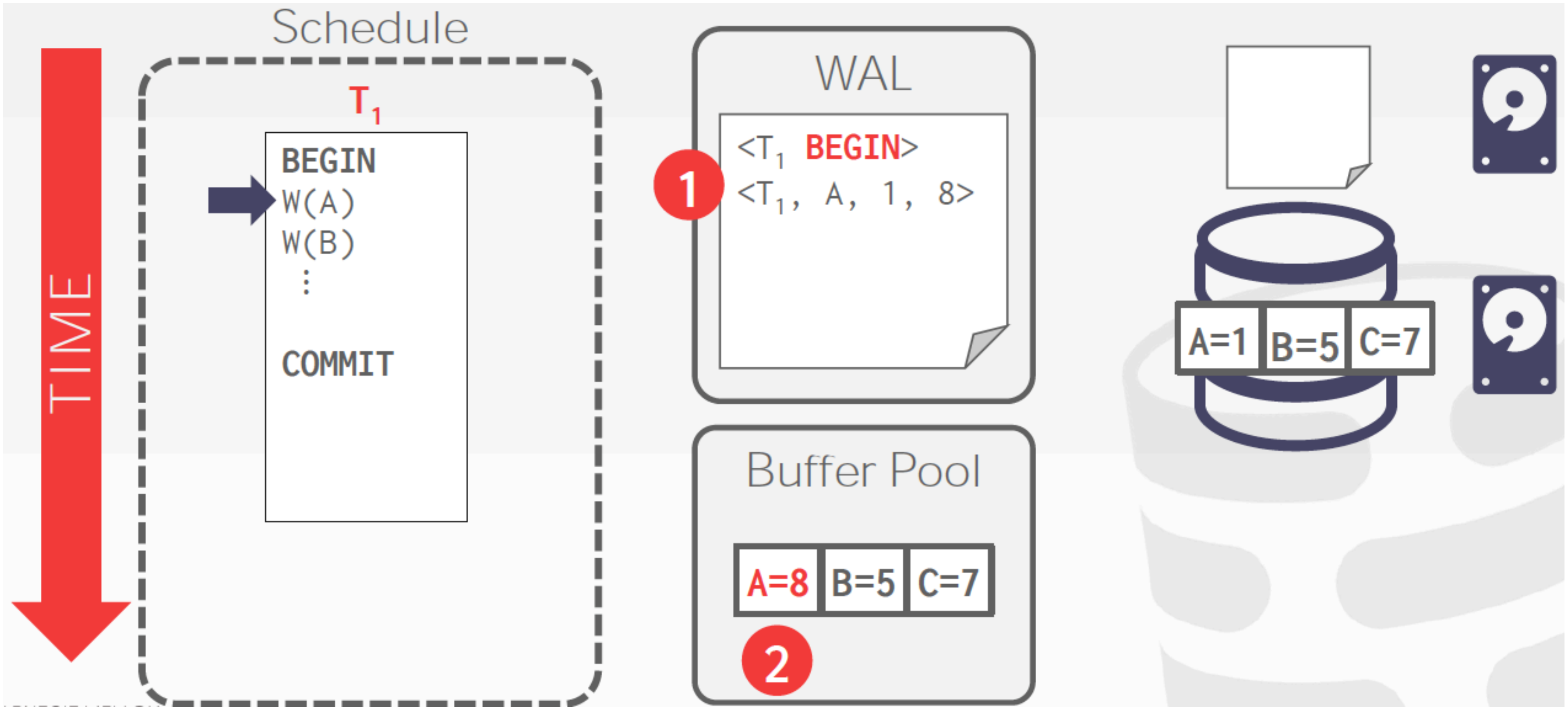
Copyright © 2016 Ramez Elmasri and Shamkant B. Navathe

WAL Example

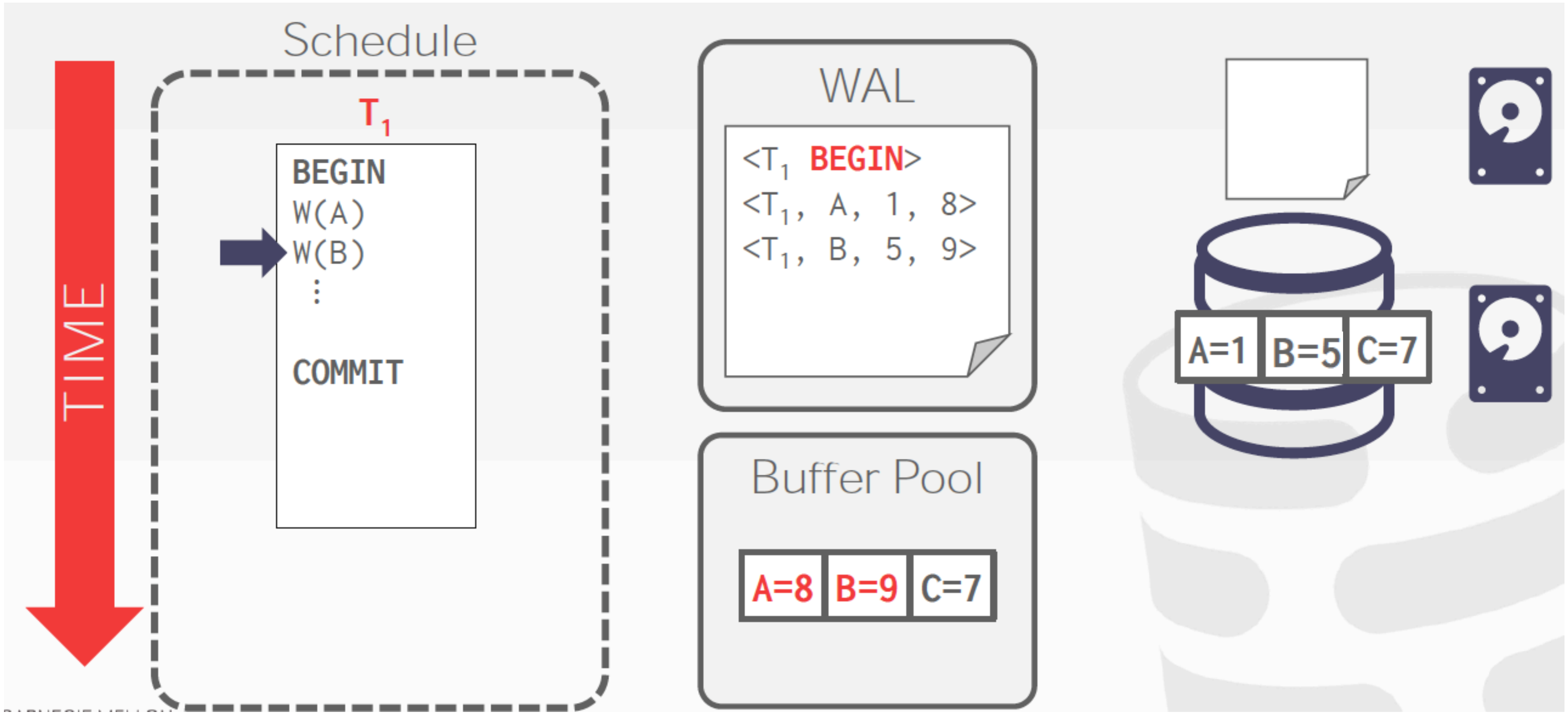


slide from CMU 15-445/645

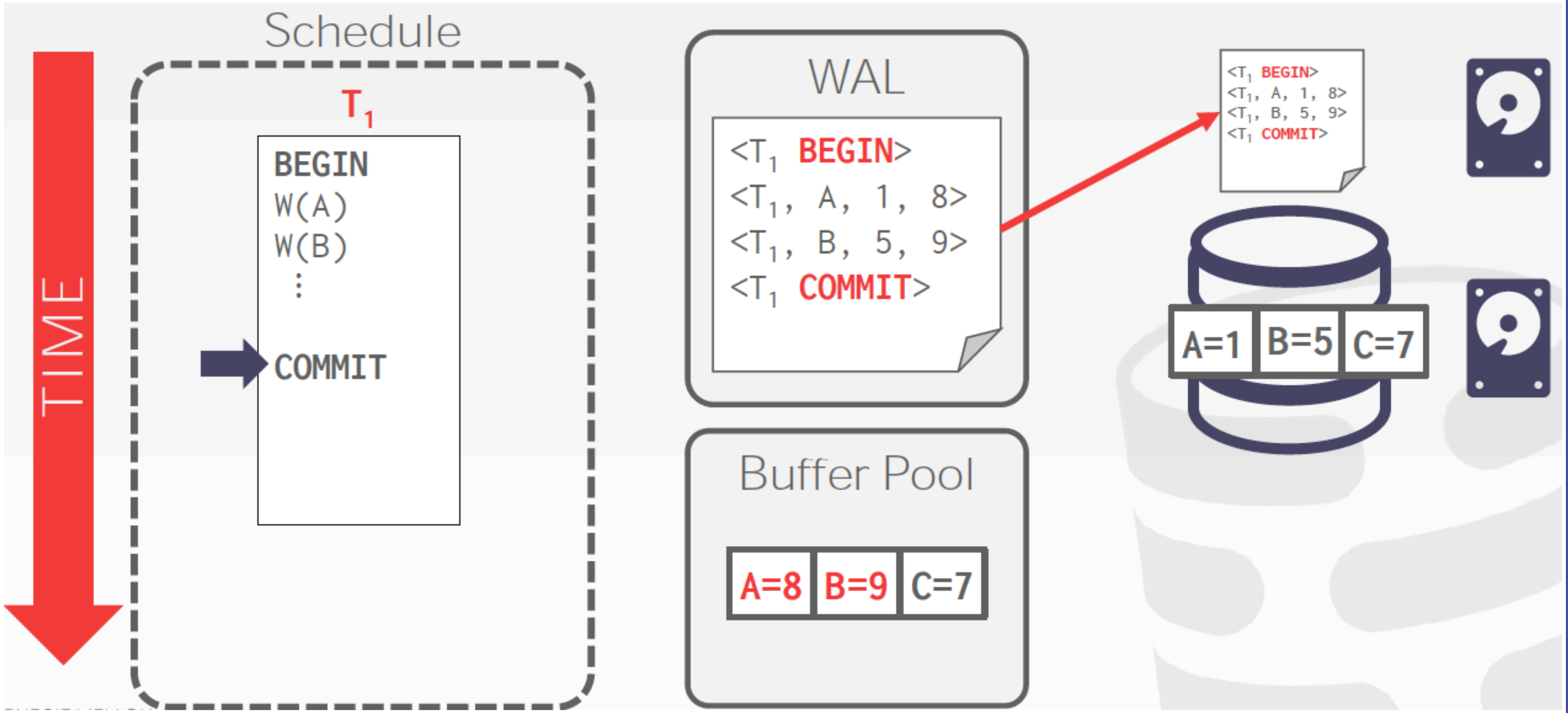
WAL Example


Schedule
TIME
T1
BEGIN
W(A)
W(B)
⋮
COMMIT
WAL
1
<T1 BEGIN>
<T1, A, 1, 8>
Buffer Pool
A=8
B=5
C=7
2
A=1
B=5
C=7

WAL Example

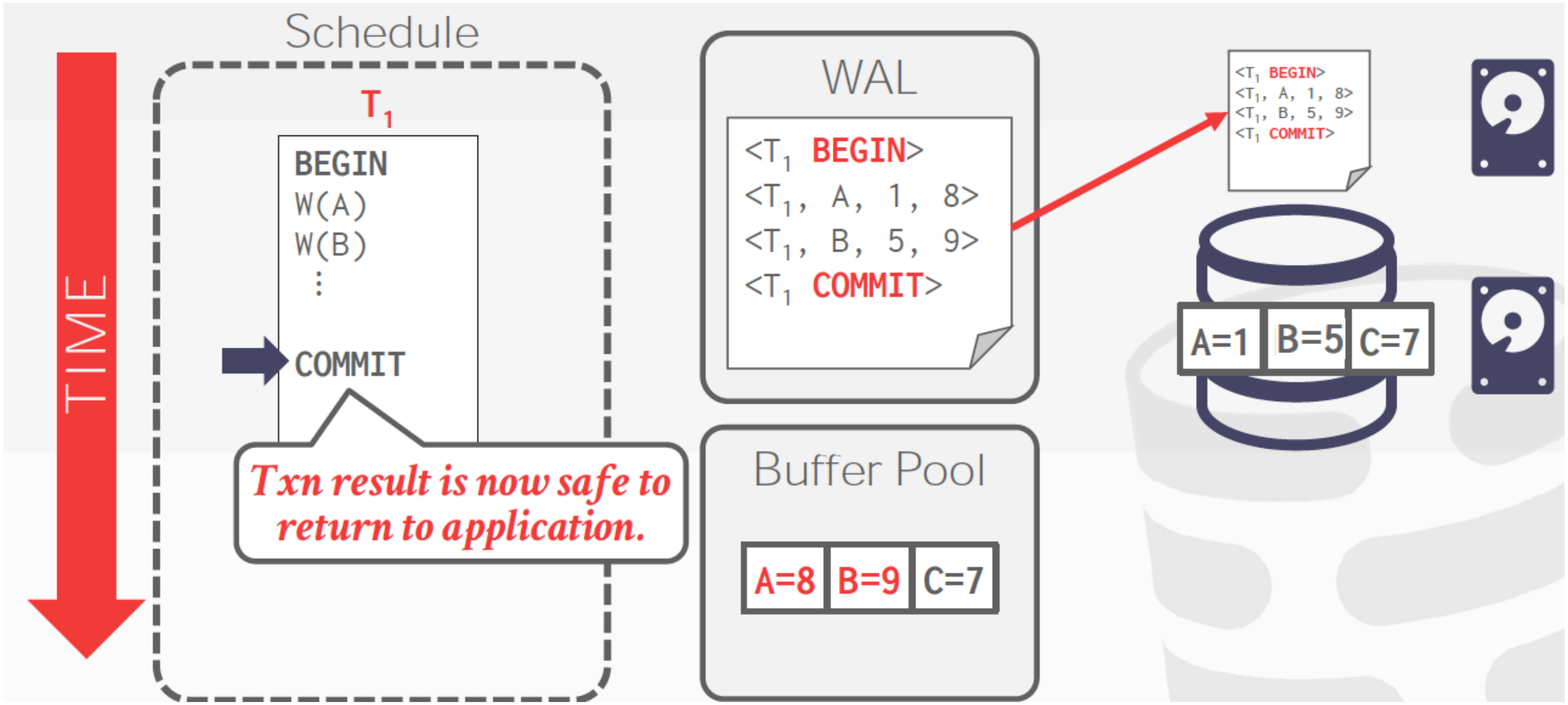

Schedule
TIME
T1
BEGIN
W(A)
W(B)
⋮
COMMIT
WAL
<T1 BEGIN>
<T1, A, 1, 8>
<T1, B, 5, 9>
Buffer Pool
A=8
B=9
C=7
A=1
B=5
C=7

WAL Example

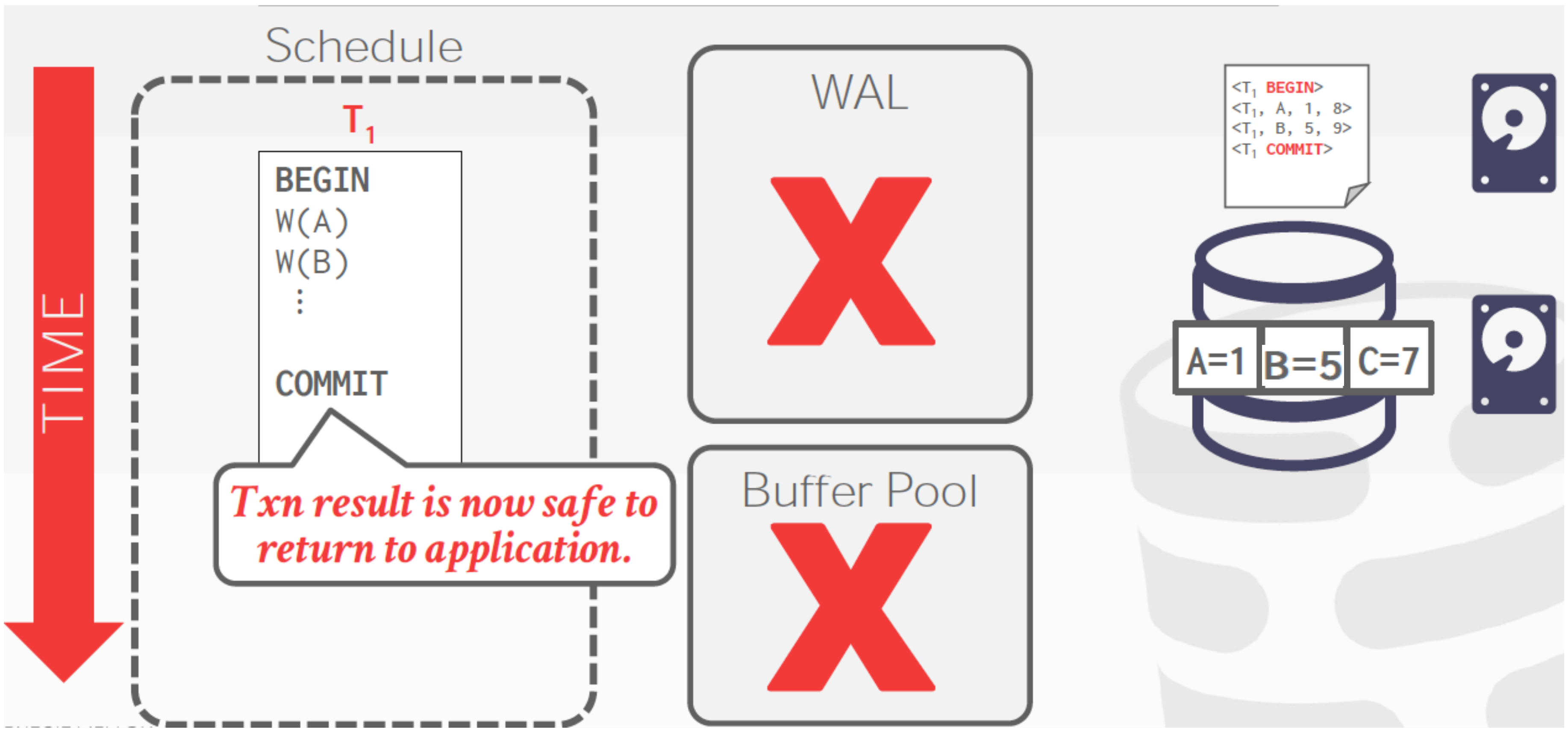


slide from CMU 15-445/645

WAL Example


Schedule
TIME
T_1
BEGIN
W(A)
W(B)
⋮
COMMIT
Txn result is now safe to return to application.
WAL
<T_1 BEGIN>
<T_1, A, 1, 8>
<T_1, B, 5, 9>
<T_1 COMMIT>
Buffer Pool
A=8
B=9
C=7
<T_1 BEGIN>
<T_1, A, 1, 8>
<T_1, B, 5, 9>
<T_1 COMMIT>
A=1
B=5
C=7

WAL Example



slide from CMU 15-445/645

Checkpoints in the System Log

- Taking a checkpoint
  - Suspend execution of all transactions temporarily
  - Force-write all main memory buffers that have been modified to disk
  - Write a checkpoint record to the log, and force-write the log to the disk
  - Resume executing transactions
- DBMS recovery manager decides on checkpoint interval

Copyright © 2016 Ramez Elmasri and Shamkant B. Navathe

Transaction Rollback

- Transaction failure after update but before commit
  - Necessary to roll back the transaction
  - Old data values restored using undo-type log entries
- Cascading rollback
  - If transaction T is rolled back, any transaction S that has read value of item written by T must also be rolled back
  - Almost all recovery mechanisms designed to avoid this

Copyright © 2016 Ramez Elmasri and Shamkant B. Navathe

Figure 22.1 Illustrating cascading rollback (a process that never occurs in strict or cascadeless schedules) (a) The read and write operations of three transactions (b) System log at point of crash (c) Operations before the crash

(a)

T_1	T_2	T_3
read_item(A)	read_item(B)	read_item(C)
read_item(D)	write_item(B)	write_item(B)
write_item(D)	read_item(D)	read_item(A)
	write_item(D)	write_item(A)

(b)

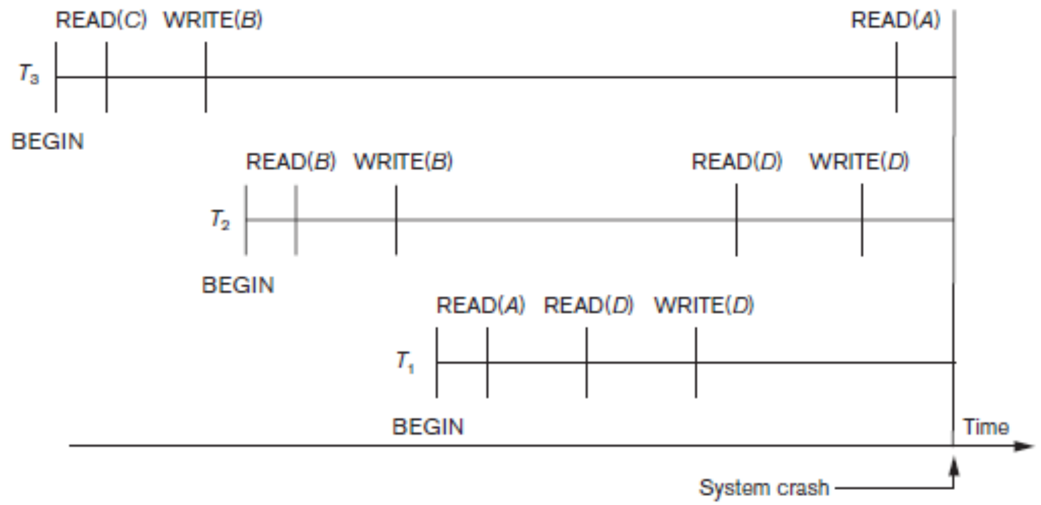
		A	B	C	D
		30	15	40	20
	[start_transaction,T_3]				
	[read_item,T_3,C]				
*	[write_item,T_3,B,15,12]		12		
	[start_transaction,T_2]				
	[read_item,T_2,B]				
**	[write_item,T_2,B,12,18]		18		
	[start_transaction,T_1]				
	[read_item,T_1,A]				
	[read_item,T_1,D]				
	[write_item,T_1,D,20,25]				25
	[read_item,T_2,D]				
**	[write_item,T_2,D,25,26]				26
	[read_item,T_3,A]				

← System crash

* T_3 is rolled back because it did not reach its commit point.

** T_2 is rolled back because it reads the value of item B written by T_3.

(c)



Transactions that Do Not Affect the Database

- Example actions: generating and printing messages and reports
- If transaction fails before completion, may not want user to get these reports
  - Reports should be generated only after transaction reaches commit point
- Commands that generate reports issued as batch jobs executed only after transaction reaches commit point
  - Batch jobs canceled if transaction fails

Copyright © 2016 Ramez Elmasri and Shamkant B. Navathe

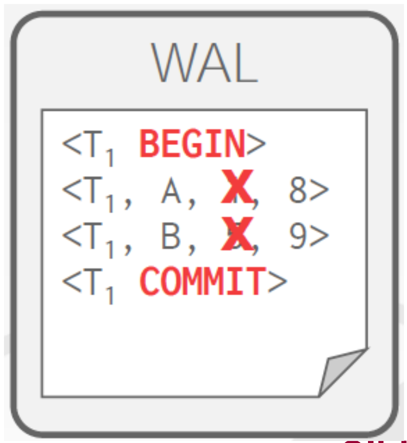
22.2 NO-UNDO/REDO Recovery Based on Deferred Update

- Deferred update concept
  - Postpone updates to the database on disk until the transaction completes successfully and reaches its commit point
  - Redo-type log entries are needed
  - Undo-type log entries not necessary
  - Can only be used for short transactions and transactions that change few items
    - Buffer space an issue with longer transactions

Copyright © 2016 Ramez Elmasri and Shamkant B. Navathe

NO-UNDO/REDO Recovery Based on Deferred Update (cont'd.)

- Deferred update protocol
  - Transaction cannot change the database on disk until it reaches its commit point
    - All buffers changed by the transaction must be pinned until the transaction commits (no-steal policy)
  - Transaction does not reach its commit point until all its REDO-type log entries are recorded in log and log buffer is force-written to disk



Copyright © 2016 Ramez Elmasri and Shamkant B. Navathe

NO-UNDO/REDO Recovery Based on Deferred Update (cont'd.)

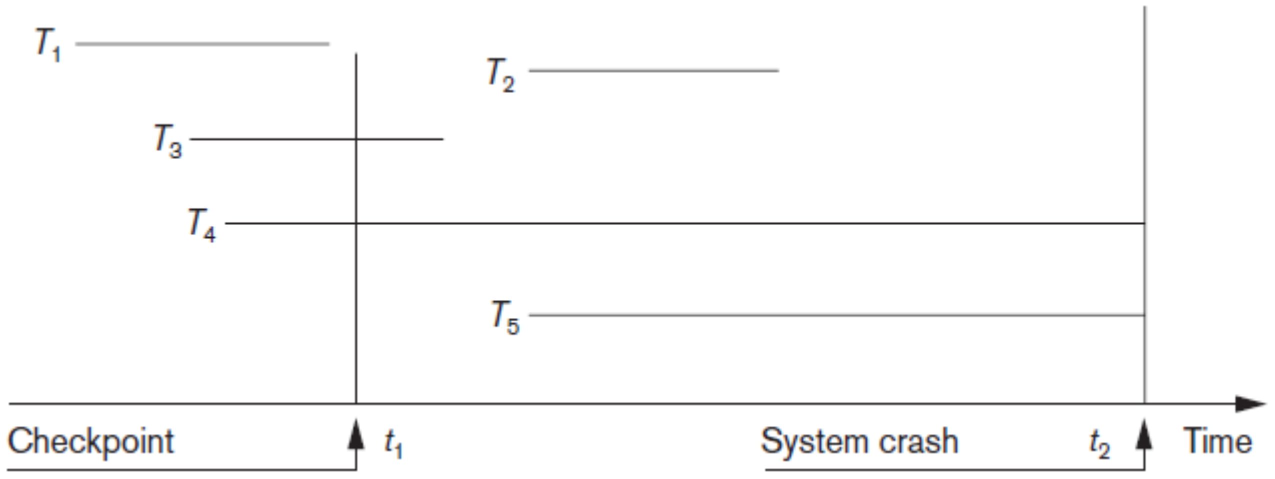


Figure 22.2 An example of a recovery timeline to illustrate the effect of checkpointing

Copyright © 2016 Ramez Elmasri and Shamkant B. Navathe

22.3 Recovery Techniques Based on Immediate Update

- Database can be updated immediately
  - No need to wait for transaction to reach commit point
  - Not a requirement that every update be immediate
- UNDO-type log entries must be stored
- Recovery algorithms
  - UNDO/NO-REDO (steal/force strategy)
  - UNDO/REDO (steal/no-force strategy)

Copyright © 2016 Ramez Elmasri and Shamkant B. Navathe

T_1	T_2	T_3	T_4
read_item(A)	read_item(B)	read_item(A)	read_item(B)
read_item(D)	write_item(B)	write_item(A)	write_item(B)
write_item(D)	read_item(D)	read_item(C)	read_item(A)
	write_item(D)	write_item(C)	write_item(A)

Figure 22.3 An example of recovery using deferred update with concurrent transactions (a) The READ and WRITE operations of four transactions (b) System log at the point of crash

[start_transaction, T_1]
[write_item, T_1, D, 20]
[commit, T_1]
[checkpoint]
[start_transaction, T_4]
[write_item, T_4, B, 15]
[write_item, T_4, A, 20]
[commit, T_4]
[start_transaction, T_2]
[write_item, T_2, B, 12]
[start_transaction, T_3]
[write_item, T_3, A, 30]
[write_item, T_2, D, 25]

$\longleftarrow$ System crash

T_2 and T_3 are ignored because they did not reach their commit points.

T_4 is redone because its commit point is after the last system checkpoint.

Copyright © 2016 Ramez Elmasri and Shamkant B. Navathe

22.4 Shadow Paging

- No log required in a single-user environment
  - Log may be needed in a multiuser environment for the concurrency control method
- Shadow paging considers disk to be made of *n* fixed-size disk pages
  - Directory with *n* entries is constructed
  - When transaction begins executing, directory copied into shadow directory to save while current directory is being used
  - Shadow directory is never modified

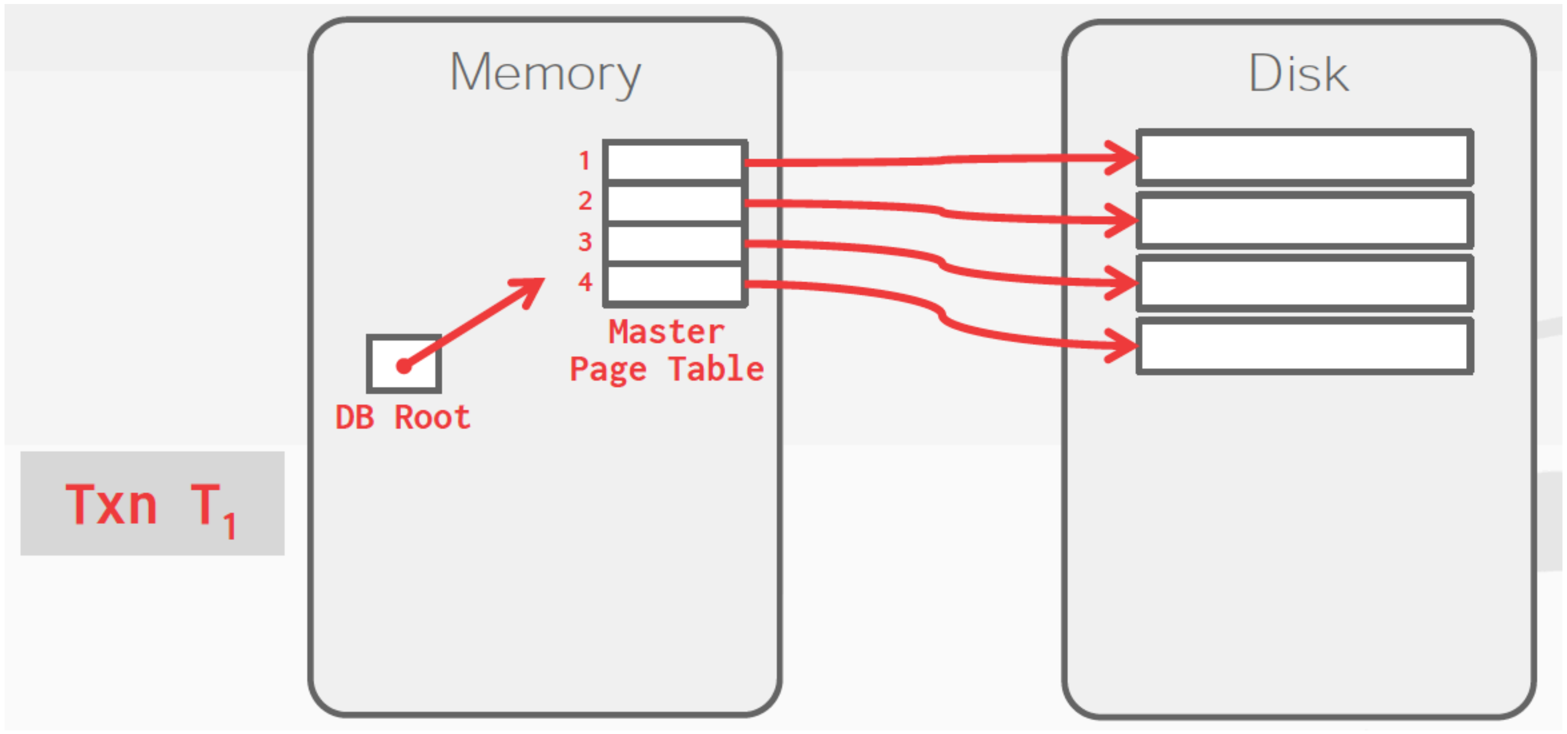
Copyright © 2016 Ramez Elmasri and Shamkant B. Navathe

Shadow Paging (cont'd.)

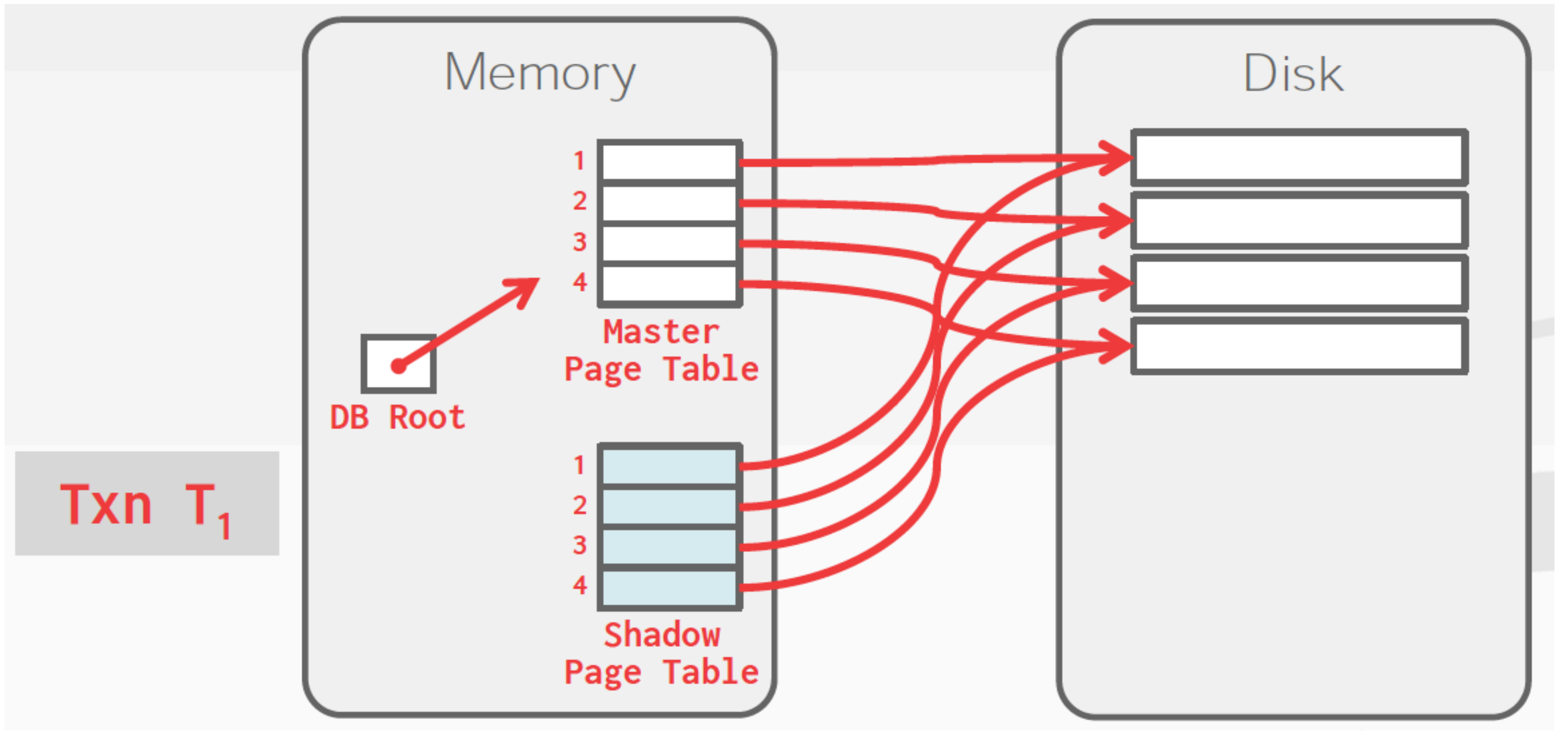
- New copy of the modified page created and stored elsewhere
  - Current directory modified to point to new disk block
  - Shadow directory still points to old disk block
- Failure recovery
  - Discard current directory
  - Free modified database pages
  - NO-UNDO/NO-REDO technique

Copyright © 2016 Ramez Elmasri and Shamkant B. Navathe

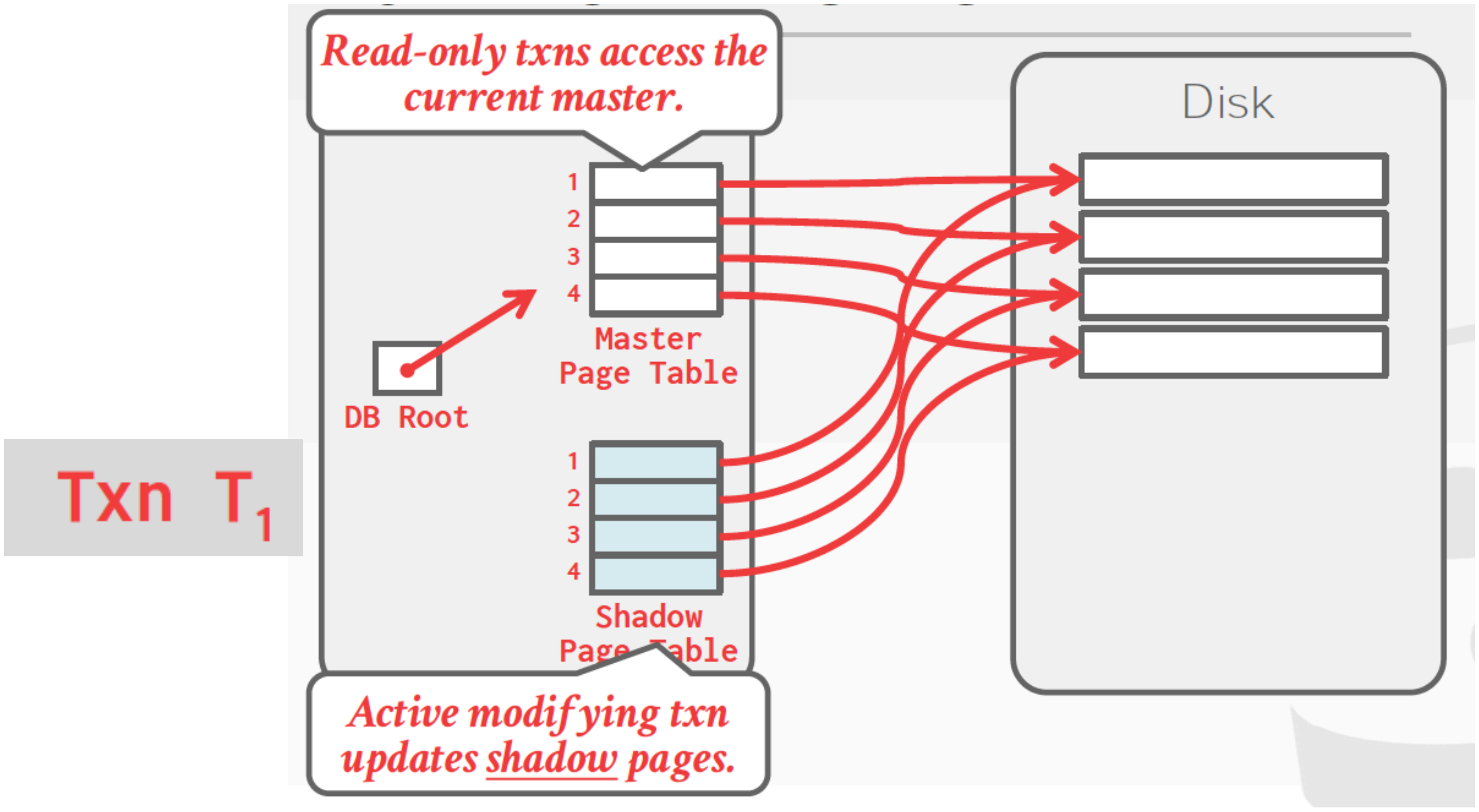
Shadow Paging



Shadow Paging

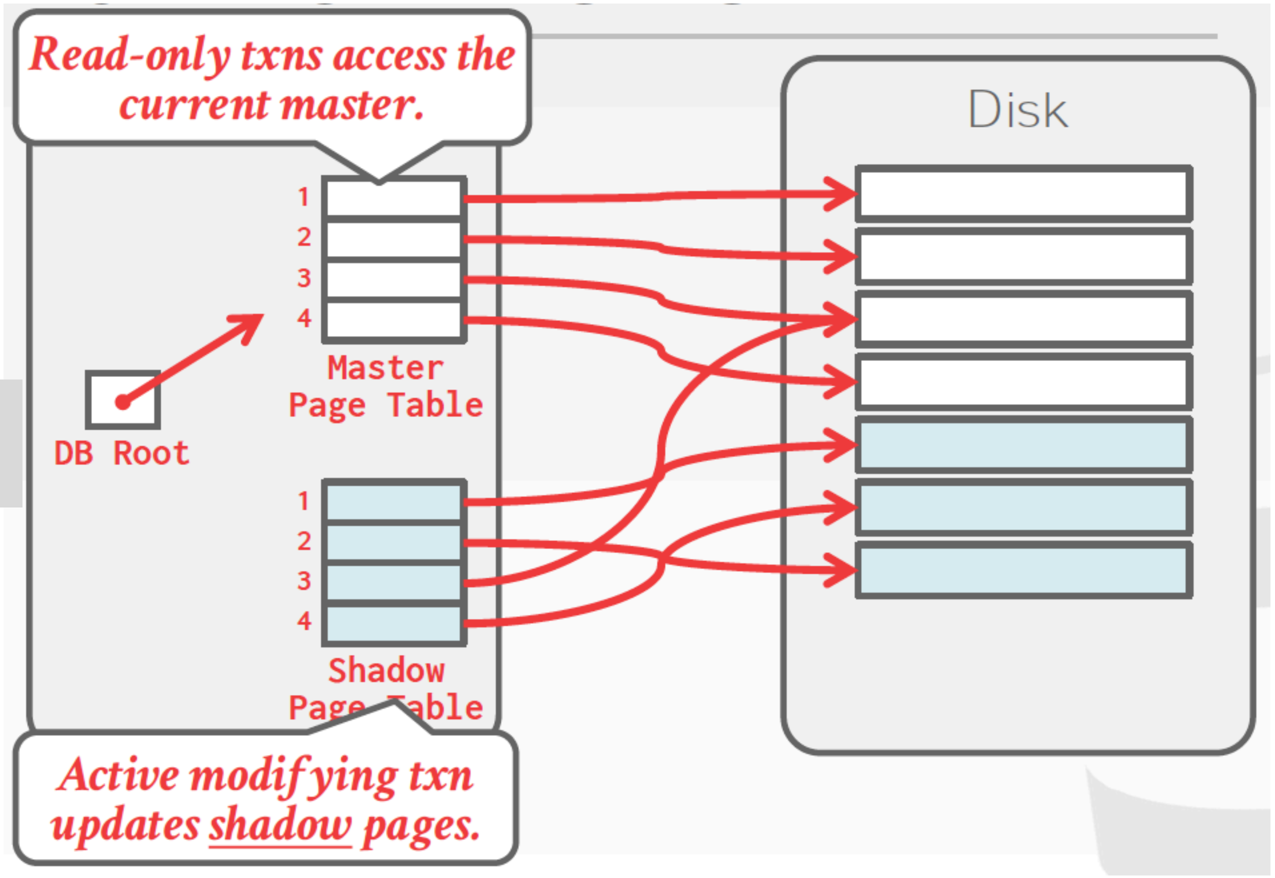


Shadow Paging


Read-only txns access the current master.
Disk
1
2
3
4
Master Page Table
DB Root
Txn T_1
1
2
3
4
Shadow Page Table
Active modifying txn updates shadow pages.

Shadow Paging

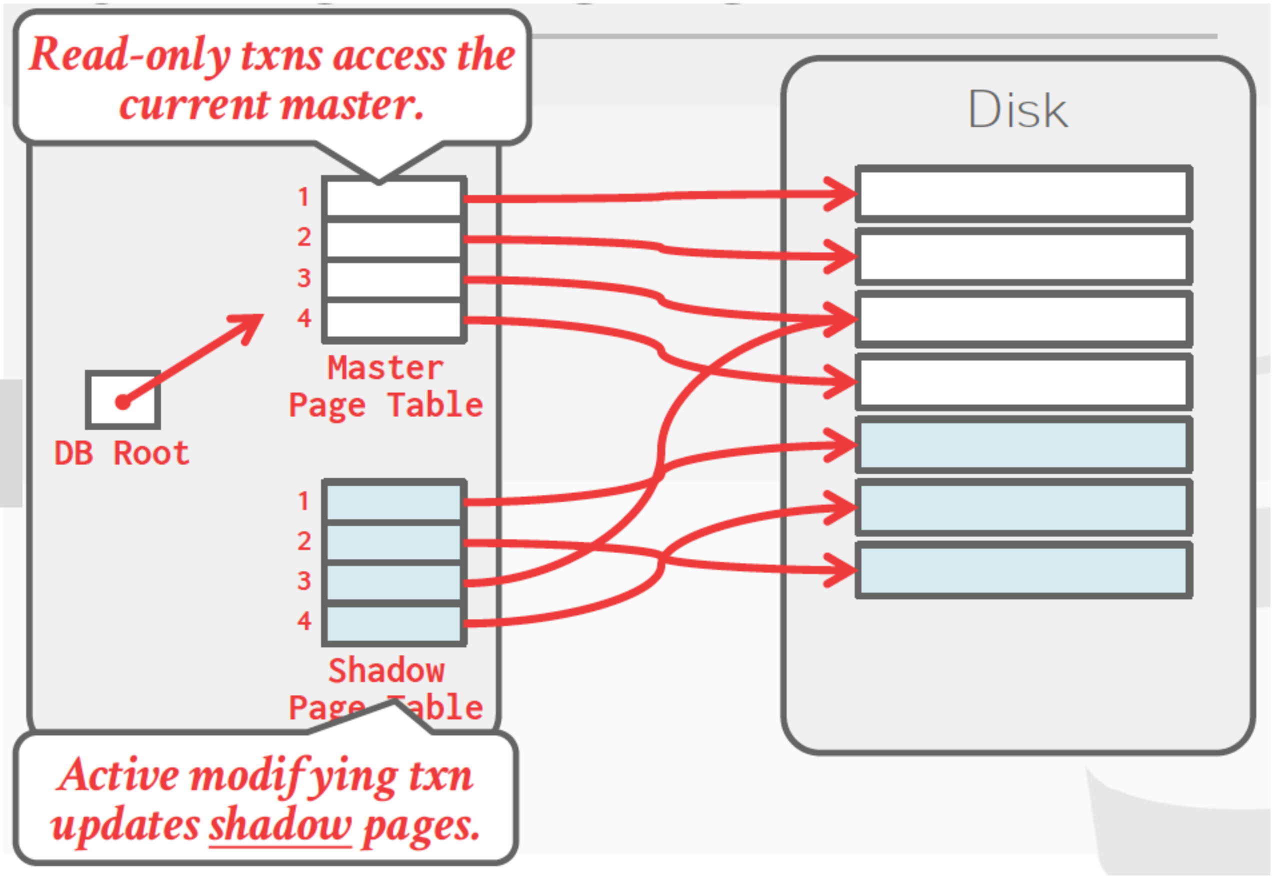
Txn T_1


Read-only txns access the current master.
Disk
1
2
3
4
Master Page Table
DB Root
1
2
3
4
Shadow Page Table
Active modifying txn updates shadow pages.

Shadow Paging

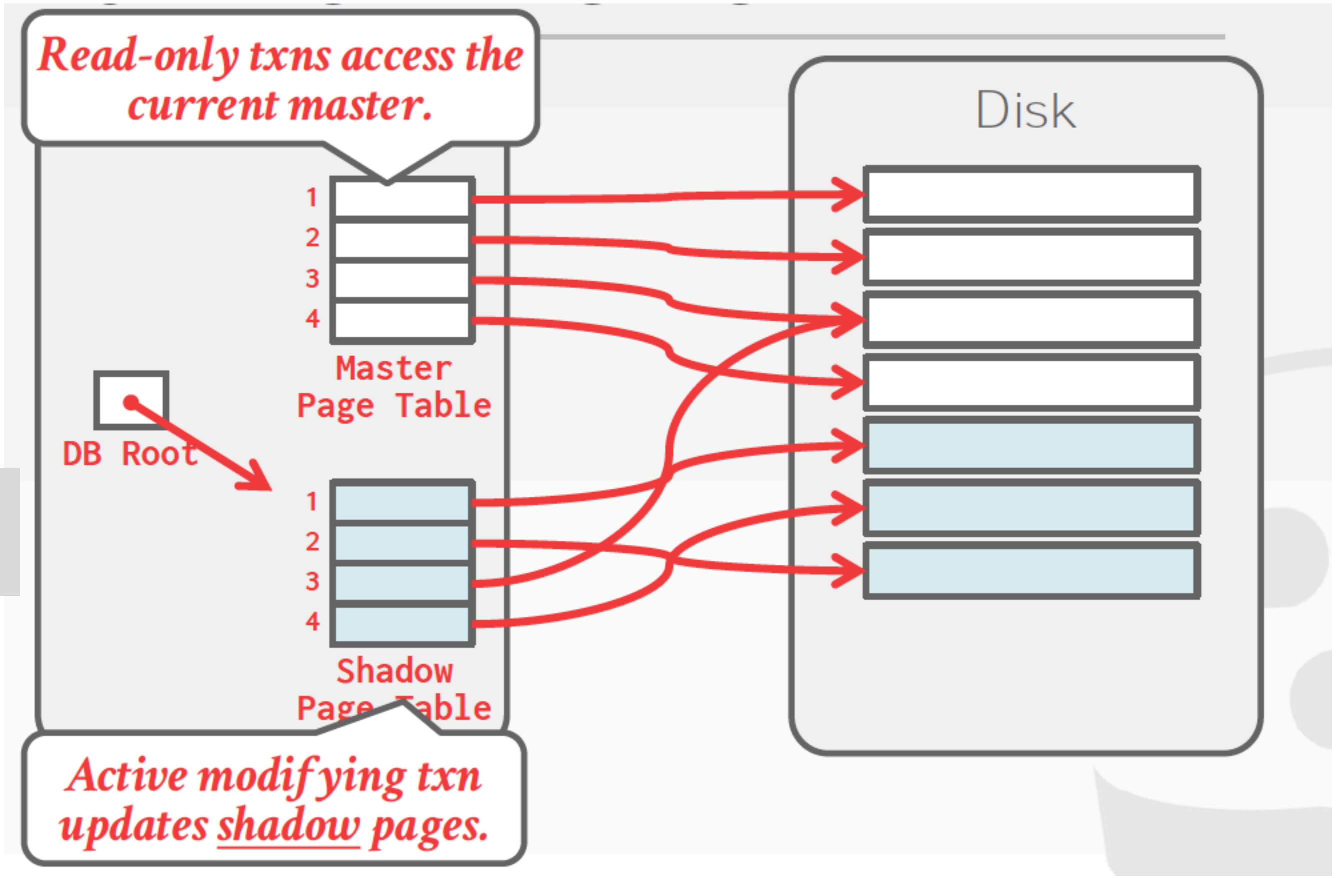
Txn T_1

COMMIT



Read-only txns access the current master.
Disk
1
2
3
4
Master Page Table
DB Root
1
2
3
4
Shadow Page Table
Active modifying txn updates shadow pages.

Shadow Paging

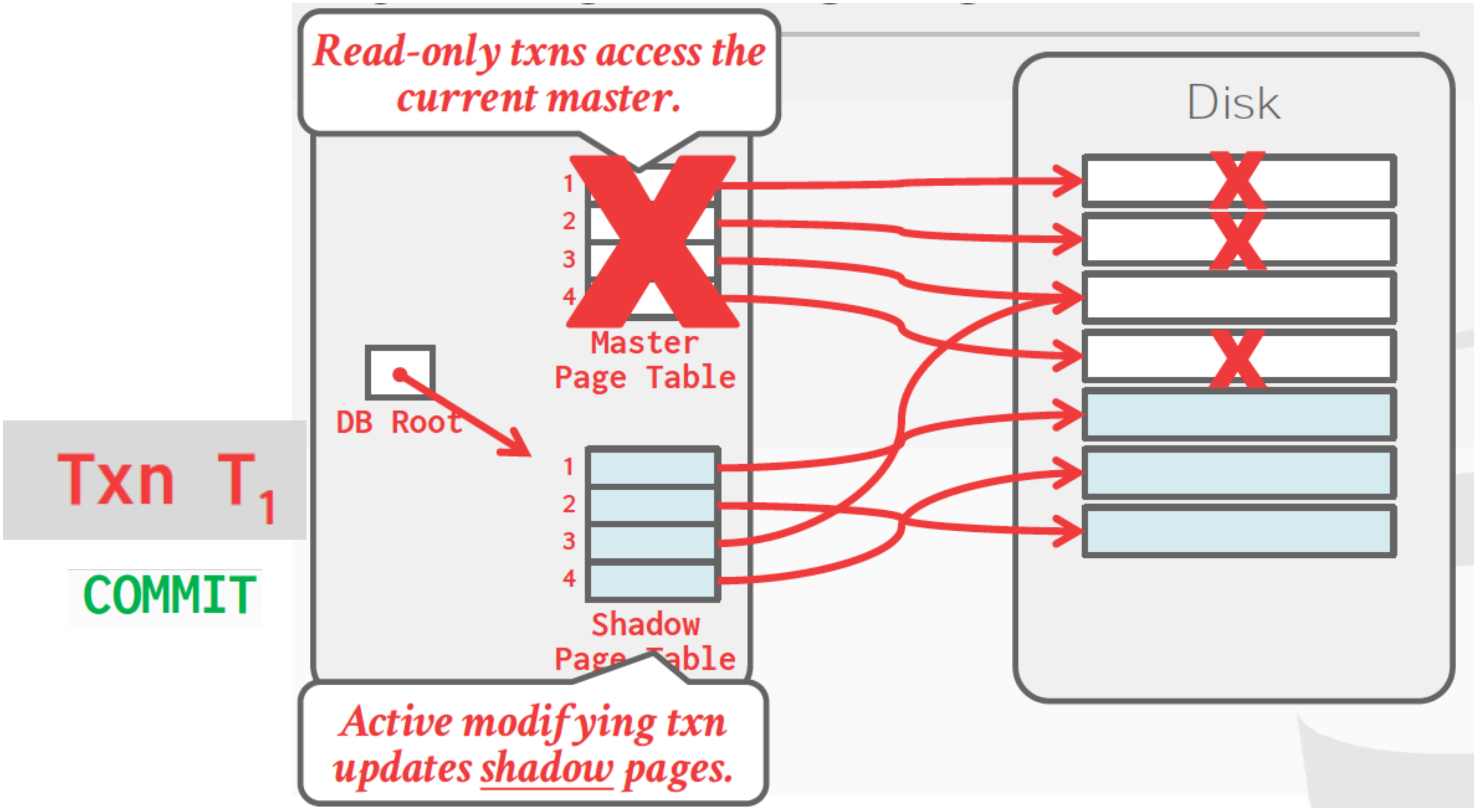

Read-only txns access the current master.
1
2
3
4
Master Page Table
DB Root
1
2
3
4
Shadow Page Table
Disk
Active modifying txn updates shadow pages.

Txn T_1

COMMIT

slide from CMU 15-445/645

Shadow Paging


Read-only txns access the current master.
Disk
1
2
3
4
Master Page Table
DB Root
Txn T_1
COMMIT
1
2
3
4
Shadow Page Table
Active modifying txn updates shadow pages.

Algorithms for Recovery and Isolation Exploiting Semantics (ARIES)

- Used in many IBM relational database products
- Uses a steal/no-force approach for writing
- Concepts
  - Write-ahead logging
  - Repeating history during redo
    - Retrace all database system actions prior to crash to reconstruct database state when crash occurred
  - Logging changes during undo
    - Prevents ARIES from repeating completed undo operations if failure occurs during recovery

Copyright © 2016 Ramez Elmasri and Shamkant B. Navathe

The ARIES Recovery Algorithm (cont'd.)

- Analysis step
  - Identifies dirty (updated) pages in the buffer and set of transactions active at the time of crash
  - Determines appropriate start point in the log for the REDO operation
- REDO
  - Reapplies updates from the log to the database
  - Only necessary REDO operations are applied

Copyright © 2016 Ramez Elmasri and Shamkant B. Navathe

The ARIES Recovery Algorithm (cont'd.)

- UNDO
  - Log is scanned backward
  - Operations of transactions that were active at the time of the crash are undone in reverse order
- Every log record has associated log sequence number (LSN)
  - Indicates address of log record on disk
  - Corresponds to a specific change of some transaction

Copyright © 2016 Ramez Elmasri and Shamkant B. Navathe

ARIES Recovery Example

Figure 22.5 An example of recovery in ARIES (a) The log at point of crash (b) The Transaction and Dirty Page Tables at time of checkpoint (c) The Transaction and Dirty Page Tables after the analysis phase

(a)

Lsn	Last_lsn	Tran_id	Type	Page_id	Other_information
1	0	T_1	update	*C*	. . .
2	0	T_2	update	*B*	. . .
3	1	T_1	commit		. . .
4	begin checkpoint				
5	end checkpoint				
6	0	T_3	update	*A*	. . .
7	2	T_2	update	*C*	. . .
8	7	T_2	commit		. . .

(b)

TRANSACTION TABLE

Transaction_id	Last_lsn	Status
T_1	3	commit
T_2	2	in progress

DIRTY PAGE TABLE

Page_id	Lsn
C	1
B	2

(c)

TRANSACTION TABLE

Transaction_id	Last_lsn	Status
T_1	3	commit
T_2	8	commit
T_3	6	in progress

DIRTY PAGE TABLE

Page_id	Lsn
C	7
B	2
A	6

Copyright © 2016 Ramez Elmasri and Shamkant B. Navathe

22.6 Recovery in Multidatabase Systems

- Two-level recovery mechanism
- Global recovery manager (coordinator) needed to maintain recovery information
- Coordinator follows two-phase commit protocol
  - Phase 1: Prepare for commit message
    - Ready to commit or cannot commit signal returned
  - Phase 2: Issue commit signal
- Either all participating databases commit the effect of the transaction or none of them do

Copyright © 2016 Ramez Elmasri and Shamkant B. Navathe

Recovery in Multidatabase Systems (cont'd.)

- Always possible to recover to a state where either the transaction is committed or it is rolled back
- Failure during phase 1 requires rollback
- Failure during phase 2 means successful transaction can recover and commit

Copyright © 2016 Ramez Elmasri and Shamkant B. Navathe

22.7 Database Backup and Recovery from Catastrophic Failures

- Database backup
  - Entire database and log periodically copied onto inexpensive storage medium
  - Latest backup copy can be reloaded from disk in case of catastrophic failure
- Backups often moved to physically separate locations
  - Subterranean storage vaults

Copyright © 2016 Ramez Elmasri and Shamkant B. Navathe

Database Backup and Recovery from Catastrophic Failures (cont'd.)

- Backup system log at more frequent intervals and copy to magnetic tape
  - System log smaller than database
    - Can be backed up more frequently
  - Benefit: users do not lose all transactions since last database backup

Copyright © 2016 Ramez Elmasri and Shamkant B. Navathe

Checkpoints in the System Log and Fuzzy Checkpointing (cont'd.)

- Fuzzy checkpointing
  - System can resume transaction processing after a begin_checkpoint record is written to the log
  - Previous checkpoint record maintained until end_checkpoint record is written

Copyright © 2016 Ramez Elmasri and Shamkant B. Navathe

Exercise (immediate update)

```
[start_transaction, T1]
[read_item, T1, A]
[read_item, T1, D]
[write_item, T1, D, 20, 25]
[commit, T1]
[checkpoint]
[start_transaction, T2]
[read_item, T2, B]
[write_item, T2, B, 12, 18]
[start_transaction, T4]
[read_item, T4, D]
[write_item, T4, D, 25, 15]
[read_item, T4, B]
[write_item, T4, B, 18, 15]
[start_transaction, T3]
[write_item, T3, A, 30, 40]
[read_item, T4, A]
[write_item, T4, A, 30, 20]
[commit, T4]
[read_item, T2, D]
[write_item, T2, D, 15, 35]   ◄──── System crash
```

Describe the recovery process from the system crash. Specify which transactions are rolled back, which operations in the log are redone and which (if any) are undone, and whether any cascading rollback takes place.

Exercise (immediate update)

```
[start_transaction, T1]
[read_item, T1, A]
[read_item, T1, D]
[write_item, T1, D, 20, 25]
[commit, T1]
[checkpoint]
[start_transaction, T2]
[read_item, T2, B]
[write_item, T2, B, 12, 18]
[start_transaction, T4]
[read_item, T4, D]
[write_item, T4, D, 25, 15]
[read_item, T4, B]
[write_item, T4, B, 18, 15]
[start_transaction, T3]
[write_item, T3, A, 30, 40]
[read_item, T4, A]
[write_item, T4, A, 30, 20]
[commit, T4]
[read_item, T2, D]
[write_item, T2, D, 15, 35]   ←—— System crash
```

Describe the recovery process from the system crash. Specify which transactions are rolled back, which operations in the log are redone and which (if any) are undone, and whether any cascading rollback takes place.

Copyright © 2016 Ramez Elmasri and Shamkant B. Navathe

Exercise (deferred update)

```
[start_transaction, T1]
[read_item, T1, A]
[read_item, T1, D]
[write_item, T1, D, 20, 25]
[commit, T1]
[checkpoint]
[start_transaction, T2]
[read_item, T2, B]
[write_item, T2, B, 12, 18]
[start_transaction, T4]
[read_item, T4, D]
[write_item, T4, D, 25, 15]
[read_item, T4, B]
[write_item, T4, B, 18, 15]
[start_transaction, T3]
[write_item, T3, A, 30, 40]
[read_item, T4, A]
[write_item, T4, A, 30, 20]
[commit, T4]
[read_item, T2, D]
[write_item, T2, D, 15, 35]     ←—— System crash
```

1. Show how the log would be different in the case of deferred update .

2. describe the recovery process, using your modified log.

Assume that only redo operations are applied, and specify which operations in the log are redone and which are ignored.

Copyright © 2016 Ramez Elmasri and Shamkant B. Navathe

Exercise (deferred update)

[start_transaction, T1]
[read_item, T1, A]
[read_item, T1, D]
[write_item, T1, D, 20, 25]
[commit, T1]
[checkpoint]
[start_transaction, T2]
[read_item, T2, B]
[write_item, T2, B, 12, 18]
[start_transaction, T4]
[read_item, T4, D]
[write_item, T4, D, 25, 15]
[read_item, T4, B]
$\boxed{\text{[write\_item, T4, B, 18, 15]}}$ **[write_item, T4, B, 12, 15]**
[start_transaction, T3]
[write_item, T3, A, 30, 40]
[read_item, T4, A]
$\boxed{\text{[write\_item, T4, A, 30, 20]}}$
[commit, T4]
[read_item, T2, D]
$\boxed{\text{[write\_item, T2, D, 15, 35]}}$

Copyright © 2016 Ramez Elmasri and Shamkant B. Navathe