UNIVERSITY of HOUSTON

DEPARTMENT OF COMPUTER SCIENCE

COSC 3380 Spring 2024

Database Systems

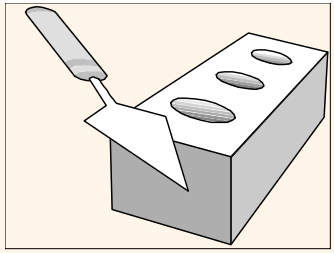
M & W 4:00 to 5:30 PM

Prof. **Victoria Hilford**

PLEASE TURN your webcam ON (must have)

NO CHATTING during LECTURE

VH, UNhide Section 20, 21



COSC 3380

4 to 5:30

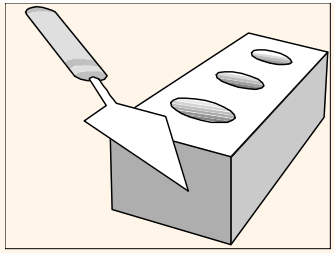
PLEASE

LOG IN

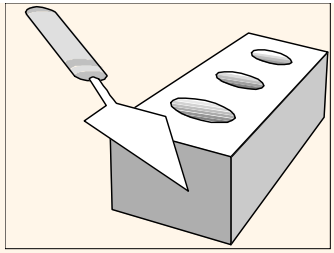
CANVAS

Please close all other windows.

COSC 3380



04.17.2024 (25 – We)	**ZyBook SET 4 - 3**	**Set 4** LECTURE 18 CRASH RECOVERY LECTURE 19 SECURITY and AUTHORIZATION
04.22.2024 (26 – Mo)		**EXAM 4 Practice** **(PART of 20 points)**
04.24.2024 (27 – We)	**TA Download** **ZyBook SET 4 Sections** **(4 PM)** **(PART of 30 points)**	**EXAM 4 Review** **(PART of 20 points)**
04.29.2024 (28 –Mo)		**EXAM 4** **(PART of 50 points)**

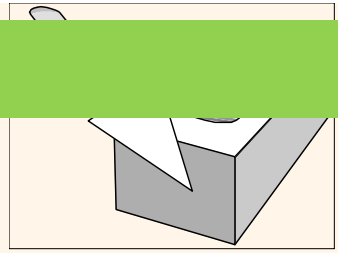


COSC 3380

Class 25

04.17.2024 (25 – We)	**ZyBook SET 4 - 3**	**Set 4** LECTURE 18 CRASH RECOVERY LECTURE 19 SECURITY and AUTHORIZATION

From 4:00 to 4:07 PM – 5 minutes.



04.17.2024 (25 – We)		ZyBook SET 4 - 3	Set 4 LECTURE 18 CRASH RECOVERY LECTURE 19 SECURITY and AUTHORIZATION

CLASS PARTICIPATION 20 points — 20% of Total

RECOVERY

Class 25 BEGIN PARTICIPATION
Not available until Apr 17 at 4:00pm | Due Apr 17 at 4:07pm

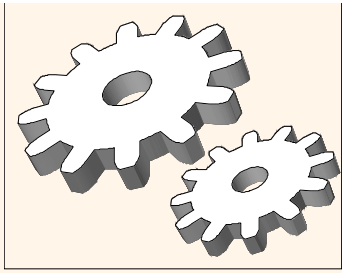
VH, publish

This is a synchronous online class.

Attendance is required.

Recording or distribution of class materials is prohibited.

1. At the beginning of selected classes there is an **assessment in the first 10 minutes**. (**beige** BOX in the Detailed Syllabus)
2. At the end of selected classes there is an **assessment in the last 10 minutes**. (**blue** BOX in the Detailed Syllabus)
3. ZyBook sections will be downloaded and used for 30% of Total Score on the dates specified in the Detailed Syllabus.
4. EXAMS are in CANVAS. **No late EXAMS.**
5. I have to be **present in TEAMS in order to take any graded assignment assigned during that class.**

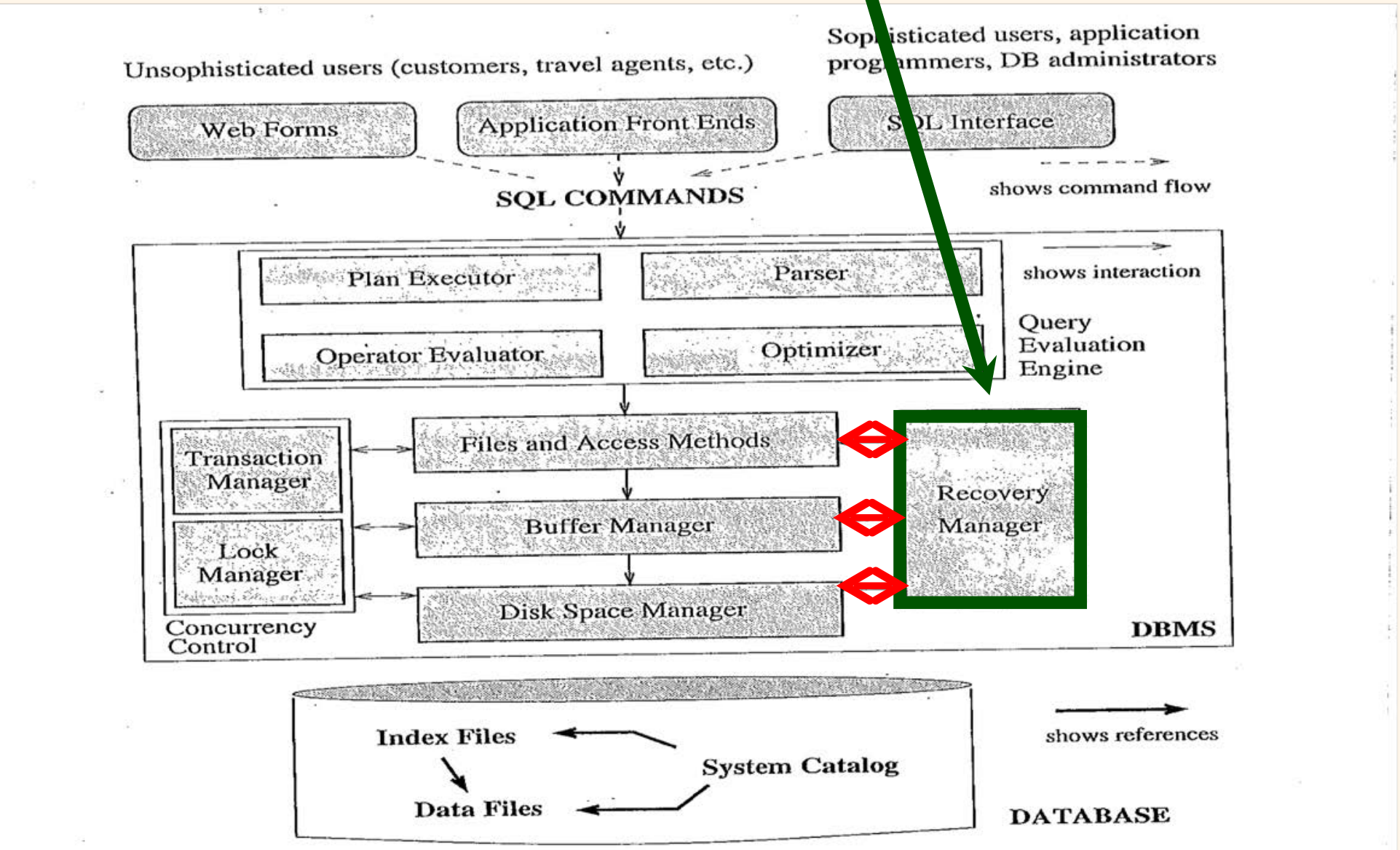


COSC 3380

Lecture 18

Crash Recovery

DBMS – Recovery Manager



Failure scenarios

The recovery system supports atomic transactions by ensuring partial transaction results are not saved in a database. The recovery system supports durable transactions by ensuring committed transactions are not lost due to hardware or software failures.

The recovery system must manage three failure scenarios:

1. A ***transaction failure*** results in a rollback. The application program may initiate a rollback due to logical errors. The database system may initiate rollback due to deadlock or insufficient disk space. The operating system may initiate rollback if a hardware or software component fails. Regardless of the cause, the recovery system restores all data changed by the transaction to the original values.

2. A ***system failure*** includes a variety of events resulting in the loss of main memory. Databases initially write to main memory blocks, which are lost when an application, the operating system, or the database system fails. Blocks are subsequently saved on storage media, which normally survive a system failure. If main memory is lost before blocks are written to storage media, data written by committed transactions might be lost. In this event, the recovery system:

    - Recovers data written to main memory, but not storage media, by committed transactions.
    - Rolls back data written to storage media by uncommitted transactions.

3. A ***storage media failure*** occurs when the database is corrupted or the database connection is lost.

    Many storage systems automatically make redundant copies of data. If one copy of data is corrupted, the storage system automatically switches to a backup copy without intervention by the database or operating system. Nevertheless, storage media do fail occasionally. Alternatively, the connection between the processor and database server might fail. Either way, the database is unavailable to the application.

In principle, recovery from storage media failure is similar to recovery from system failure. In both cases, the recovery system must restore committed transactions and roll back uncommitted transactions. However, storage media failure may cause massive data loss with lengthy recovery times. Consequently, most commercial databases implement special techniques for rapid recovery from storage media failure.

A

D
A

D
A

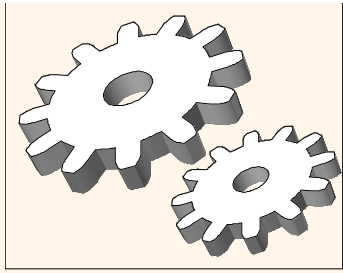
D

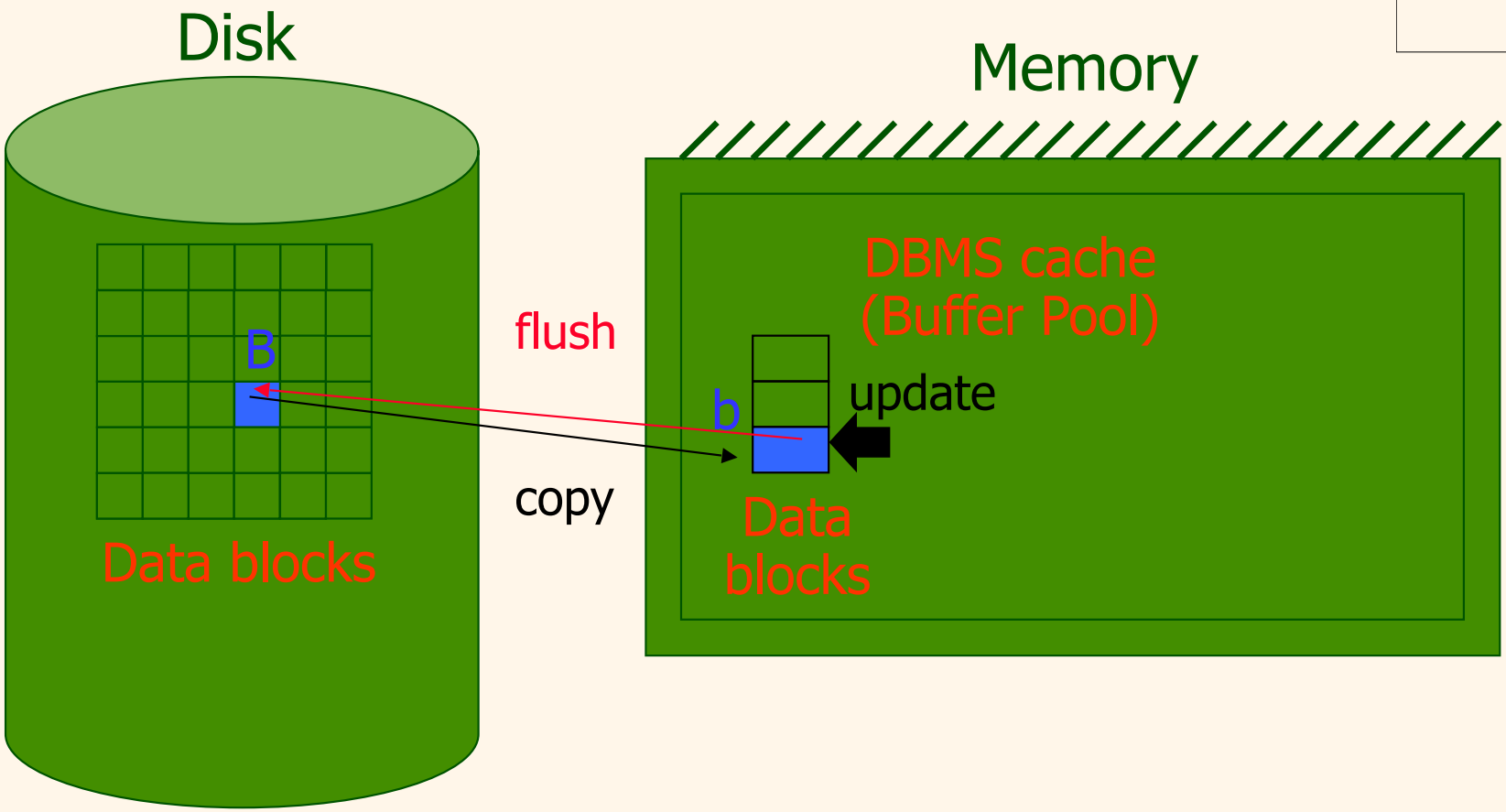
TA, Jordan (A – L).

TA, Alvaro (M – Z).

Please compare CANVAS vs. TEAMS Attendance.
Print screens of students in CANVAS but not in the TEAMS meeting.
(4.17.2024 Attendance X missing LastName.docx)

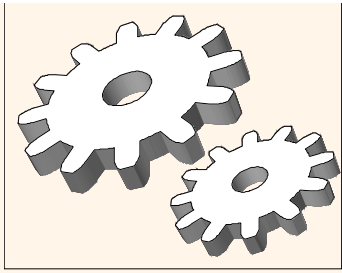
Physical View





(1) **copy** (**data** from the disk to the cache)
(2) **update** the cached **data**
(3) **flush** the **data** (from the cache to the disk)

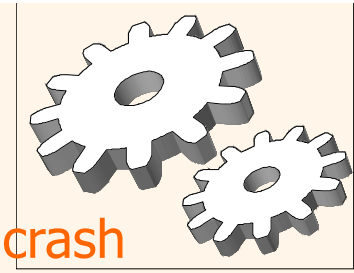
The ACID properties

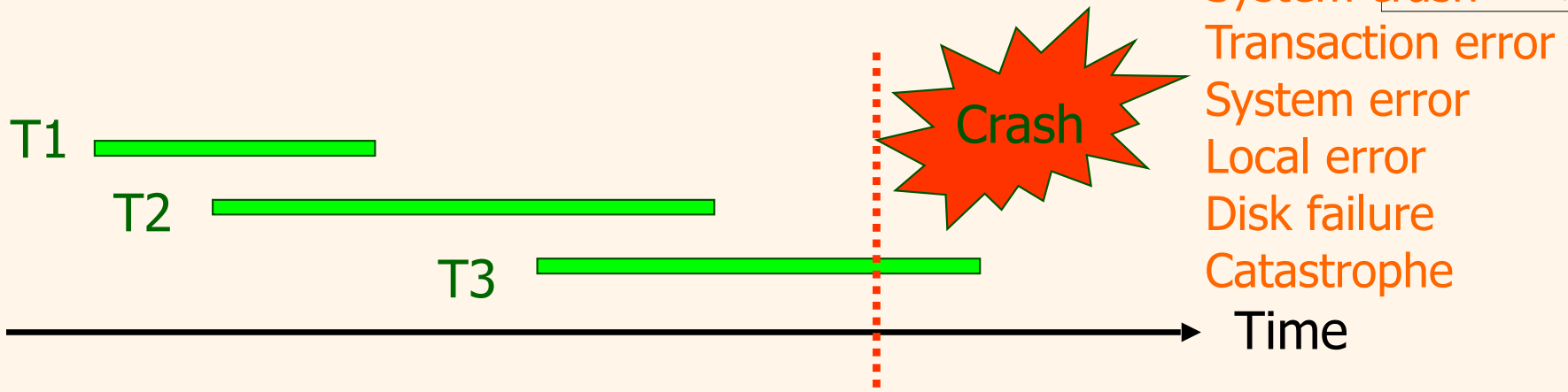


- **A** tomicity: **All** actions in the Xact happen, **or none** happen.
- **C** onsistency: If each Xact is consistent, and the DB starts consistent, it ends up consistent.
- **I** solation: Execution of one Xact is isolated from that of other Xacts.
- **D** urability: If a Xact commits, its effects **persist**.

- The **Recovery Manager** guarantees **A**tomicity & **D**urability.

Why "Database Recovery Techniques"?





System crash
Transaction error
System error
Local error
Disk failure
Catastrophe

- **A**CI **D** properties of Transactions

Database Management System should guarantee:

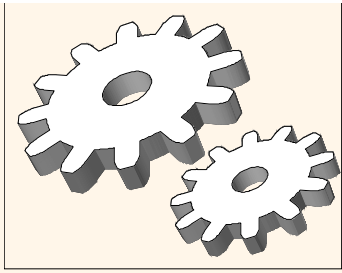
- **A**tomicity : Transactions can be **aborted**. **UNDO**

  ~ T3

- **D**urability : Applied changes by Transactions **must not be lost**. **REDO**

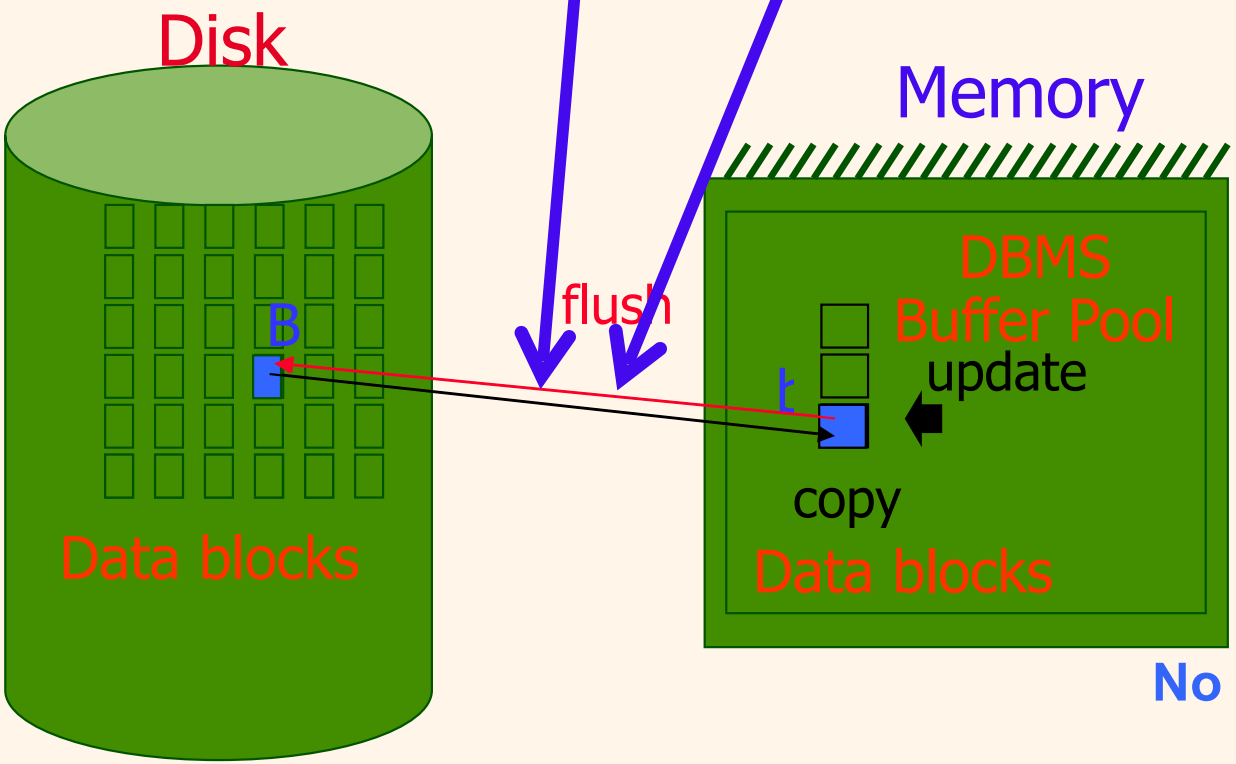
  ~ T1, T2

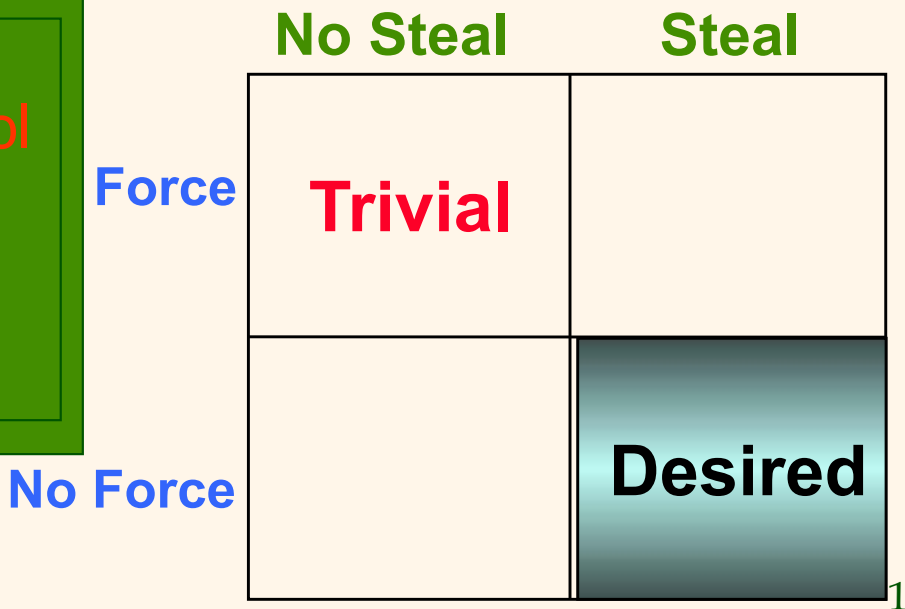
Handling the Buffer Pool (Memory)



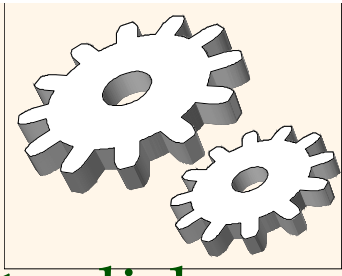
- Force every write to disk?
  - Poor response time.
  - But provides Durability.

- Steal buffer-pool frames from uncommitted Transactions?
  - If not, poor throughput.
  - If so, how can we ensure Atomicity?





More on *Steal* and *Force*

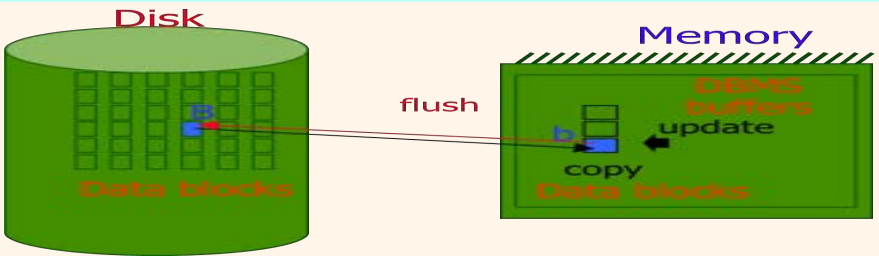


- **STEAL** (why enforcing **A**tomicity is hard)
  - *To steal frame F:* Current page in *F* (say P) is written to disk; some **Xact** holds lock on P.
    - What if the **Xact** with the lock on P Aborts?
    - Must remember the **old value of P** at steal time (to support **UNDO**ing the write to page P).

- Atomicity : Transactions can be aborted. UNDO

- **NO FORCE** (why enforcing **D**urability is hard)
  - What if system crashes before a modified page is written to disk?
    - Write as little as possible, in a convenient place, at **Commit time**,to support **REDO**ing modifications.

- Durability : Applied changes by Transactions must not be lost. REDO

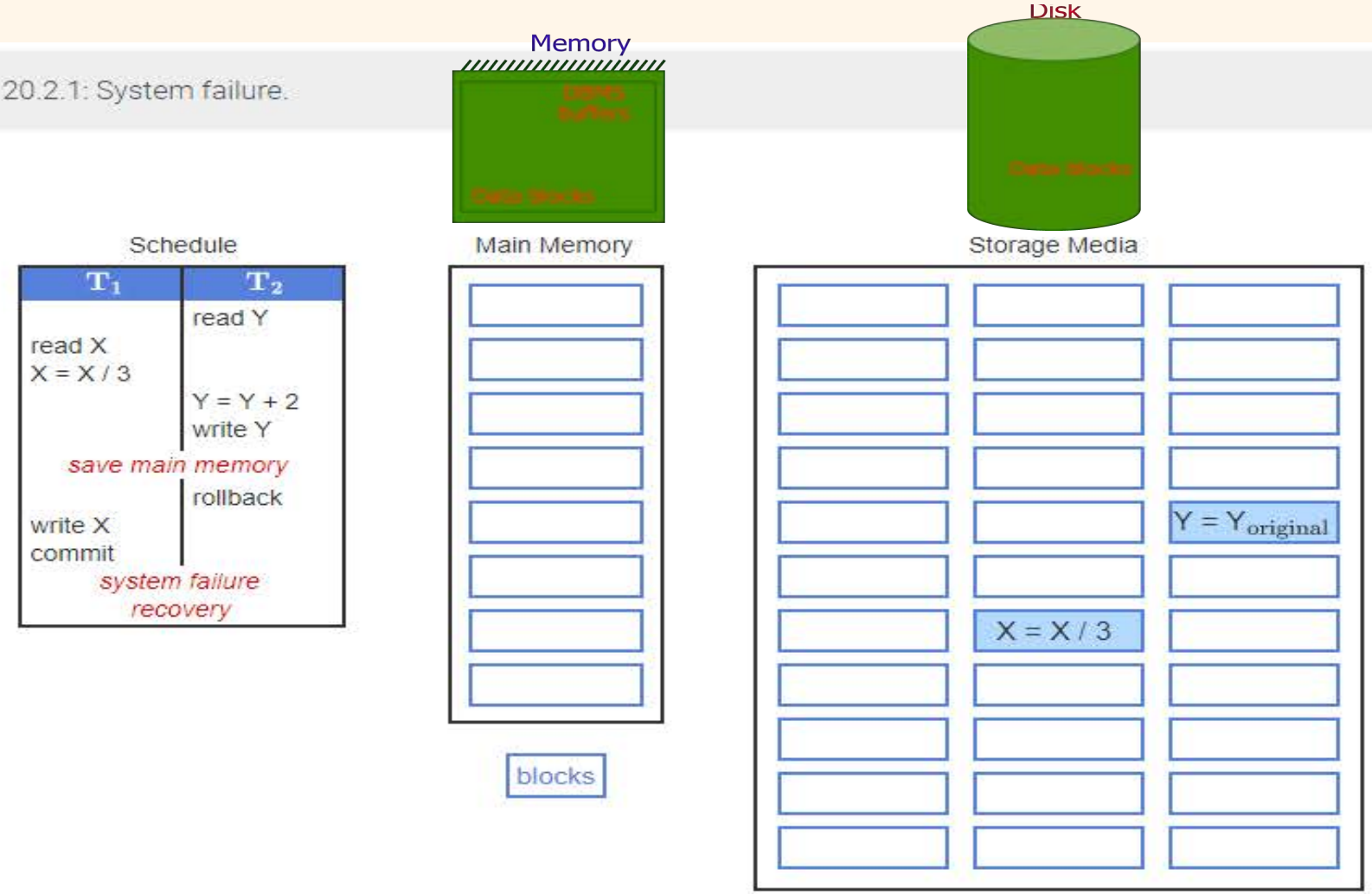


Failure scenarios

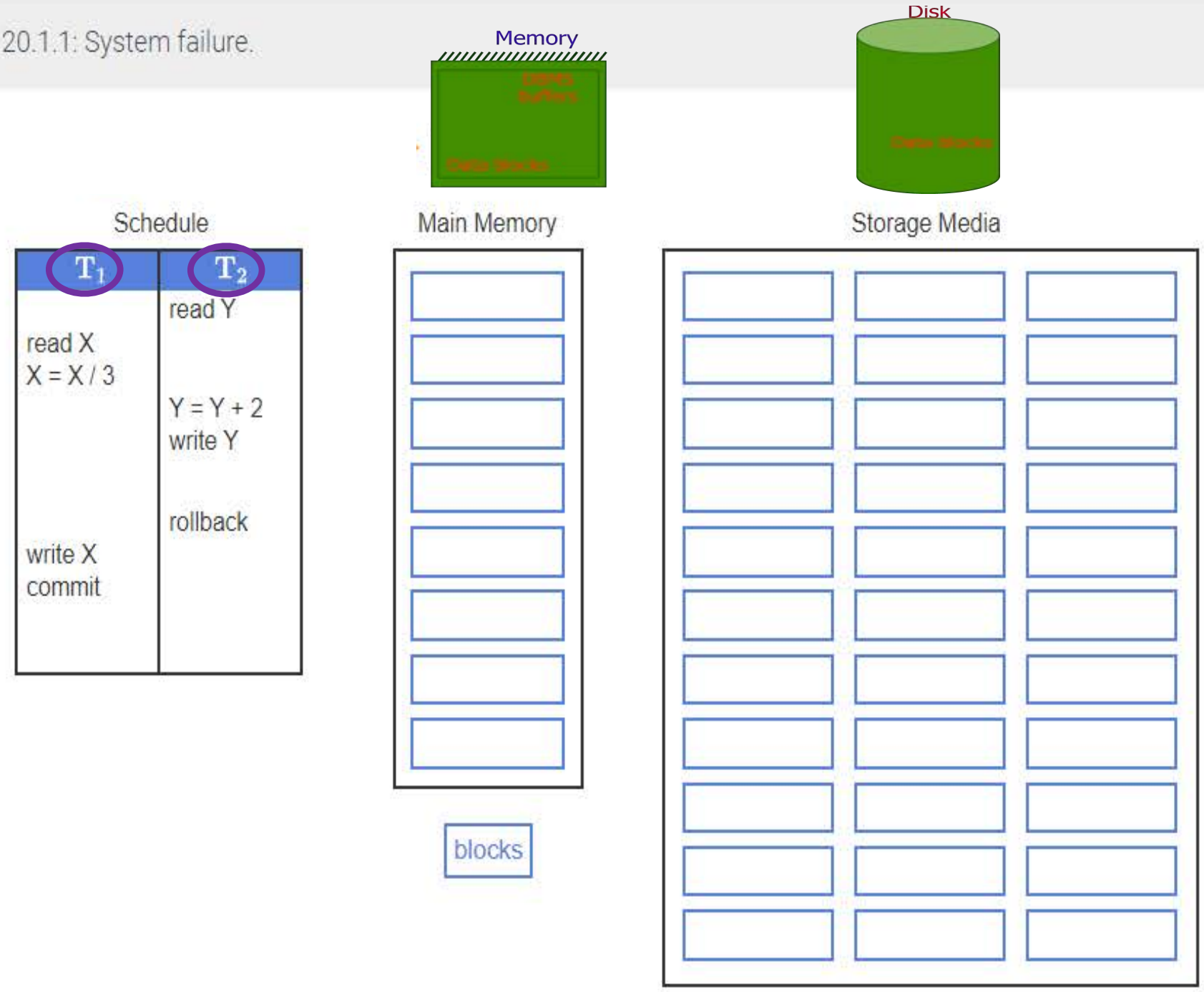
2. A ***system failure*** includes a variety of events resulting in the loss of main memory. Databases initially write to main memory blocks, which are lost when an application, the operating system, or the database system fails. Blocks are subsequently saved on storage media, which normally survive a system failure. If main memory is lost before blocks are written to storage media, data written by committed transactions might be lost. In this event, the recovery system:
   - Recovers data written to main memory, but not storage media, by committed transactions.
   - Rolls back data written to storage media by uncommitted transactions.

D
A

PARTICIPATION ACTIVITY | 20.2.1: System failure.



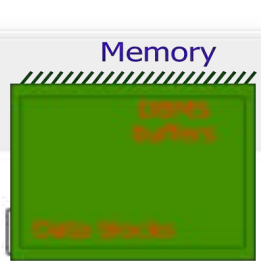
PARTICIPATION ACTIVITY | 20.1.1: System failure.



The database processes two transactions.

PARTICIPATION ACTIVITY

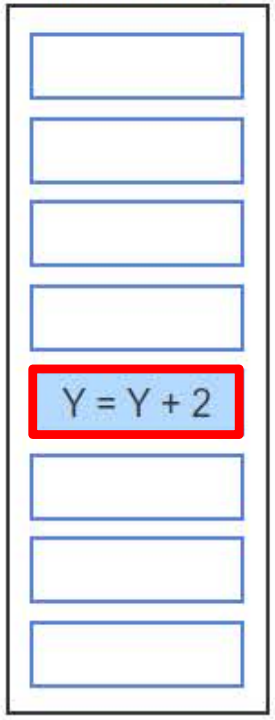
20.1.1: System failure.



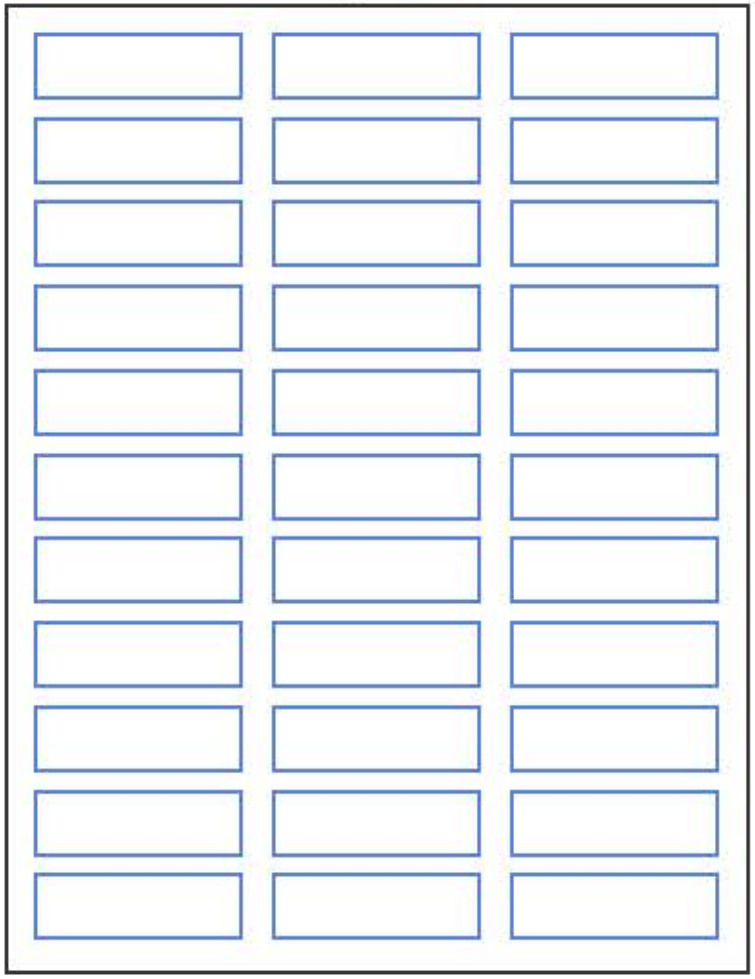
Schedule

T_1	T_2
	read Y
read X	
X = X / 3	
	Y = Y + 2
	write Y
	rollback
write X	
commit	

Main Memory

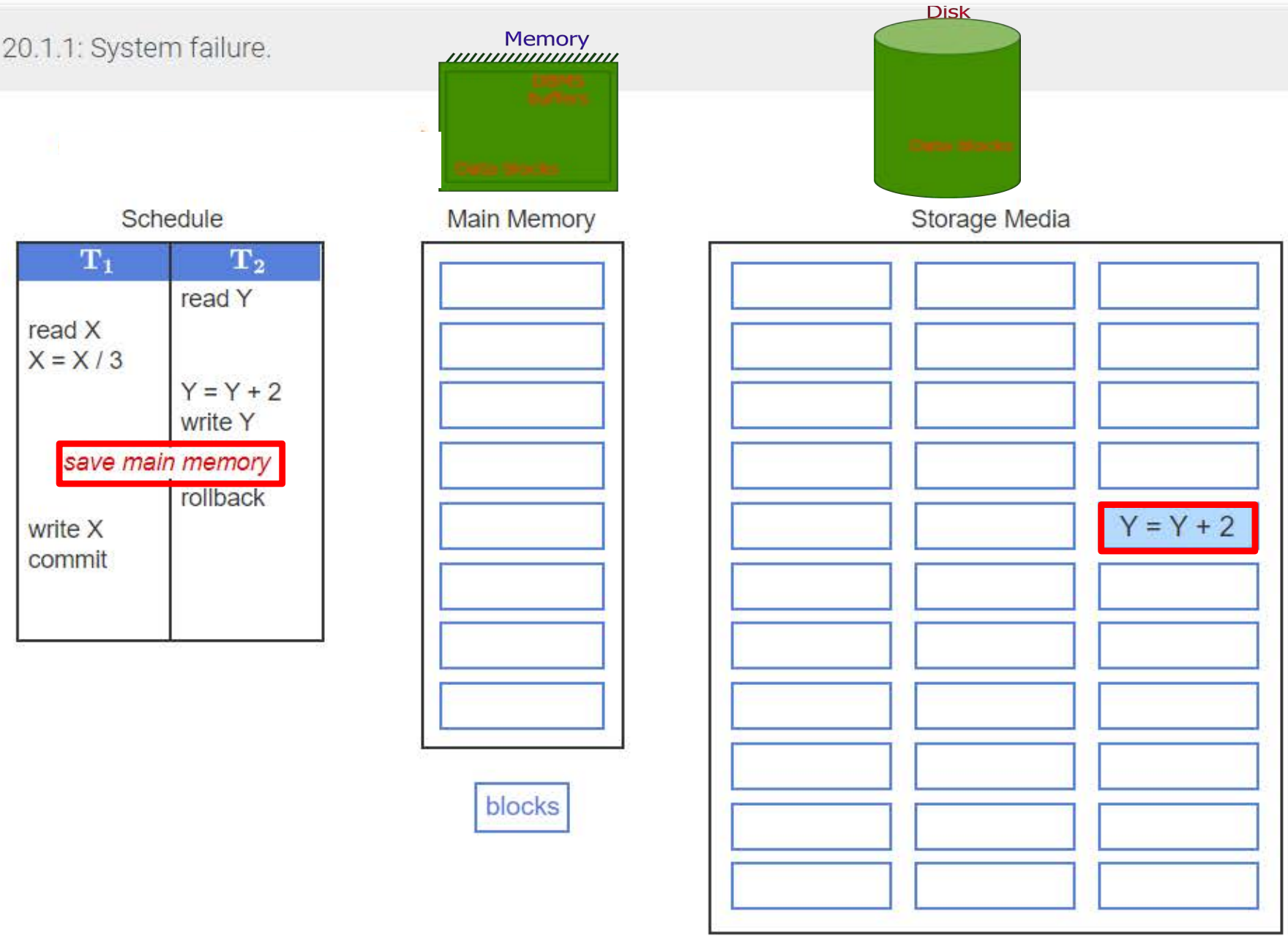


Storage Media



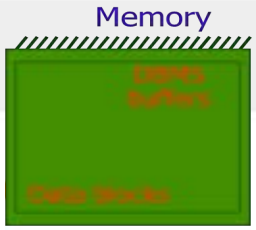
The database writes a new Y value to a block in main memory.

PARTICIPATION ACTIVITY | 20.1.1: System failure.



The database saves main memory to storage media.
All saved blocks in main memory are released for reuse.

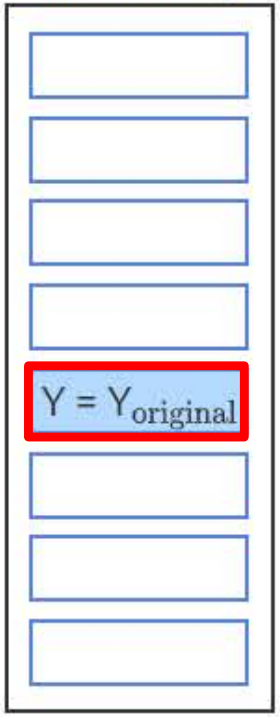
PARTICIPATION ACTIVITY | 20.1.1: System failure.



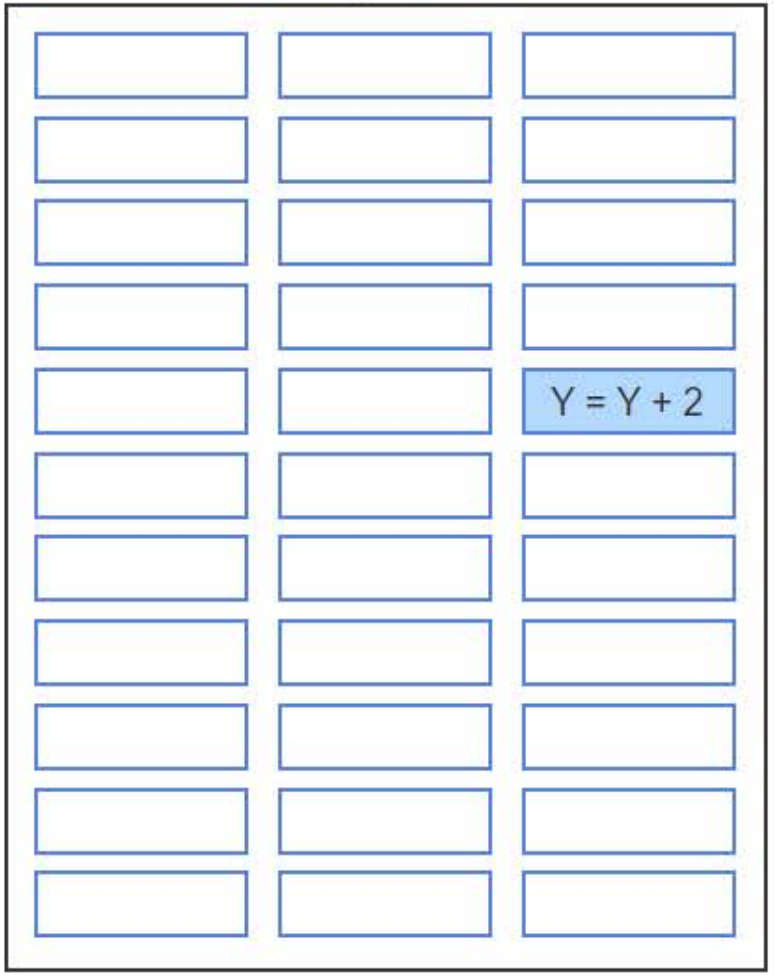
Schedule

T_1	T_2
	read Y
read X	
X = X / 3	
	Y = Y + 2
	write Y
save main memory	
	rollback
write X	
commit	

Main Memory

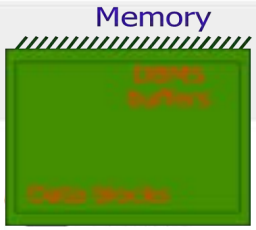


Storage Media



The database rolls back T_2, restoring Y's original value in main memory.

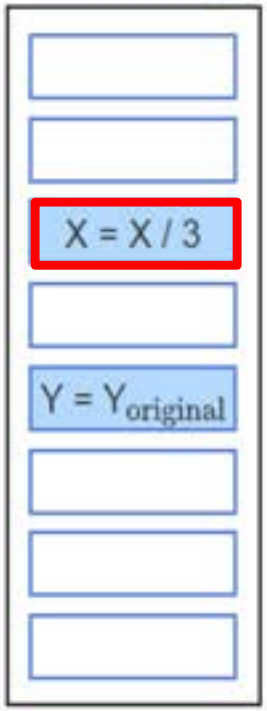
PARTICIPATION ACTIVITY | 20.1.1: System failure.



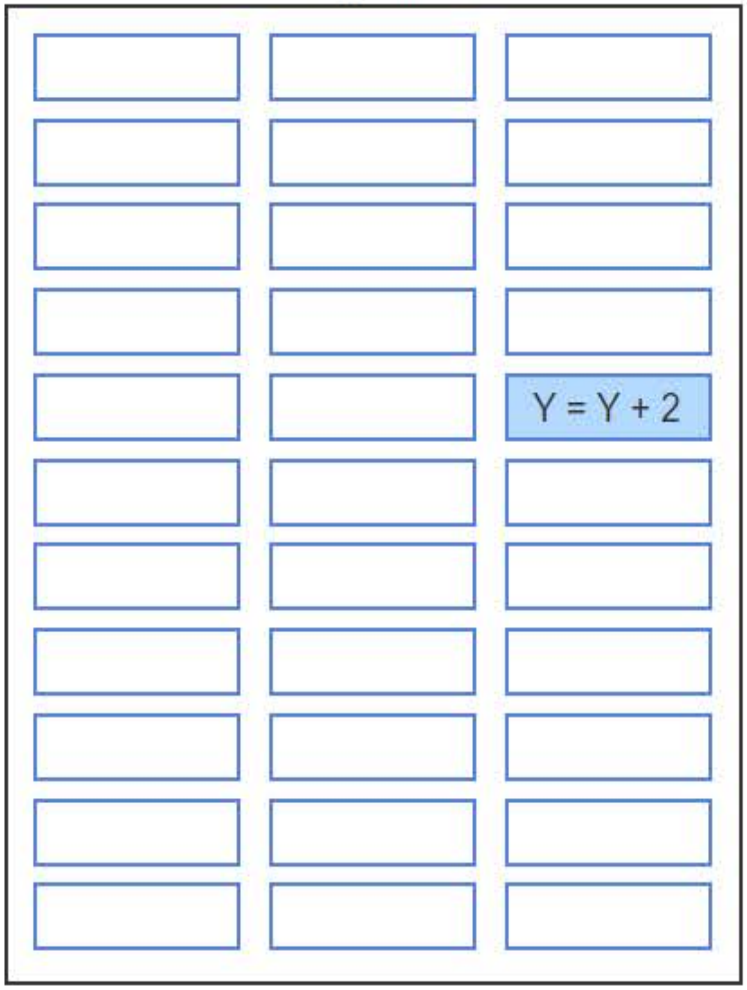
Schedule

T_1	T_2
	read Y
read X	
X = X / 3	
	Y = Y + 2
	write Y
save main memory	
	rollback
write X	
commit	

Main Memory

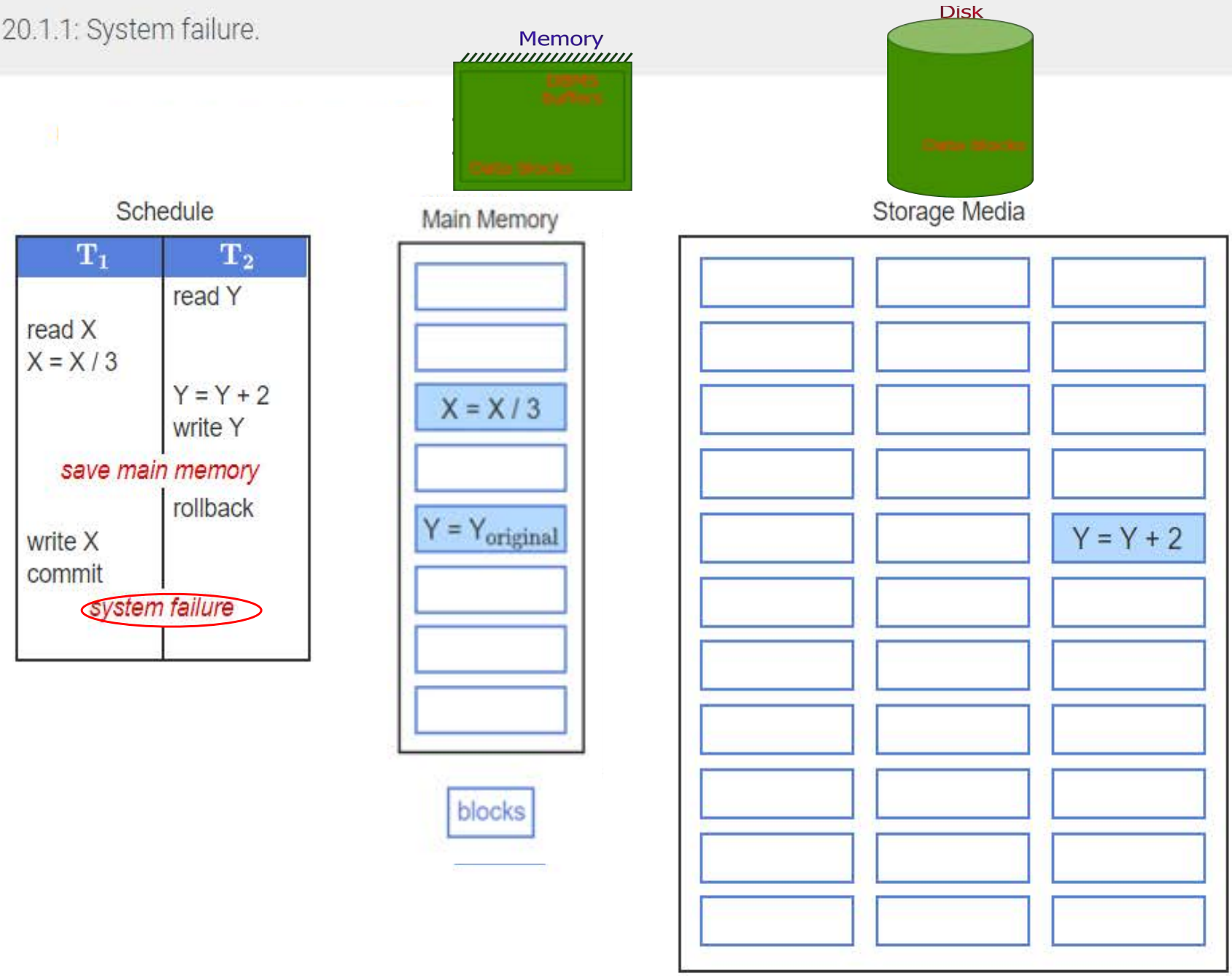


Storage Media



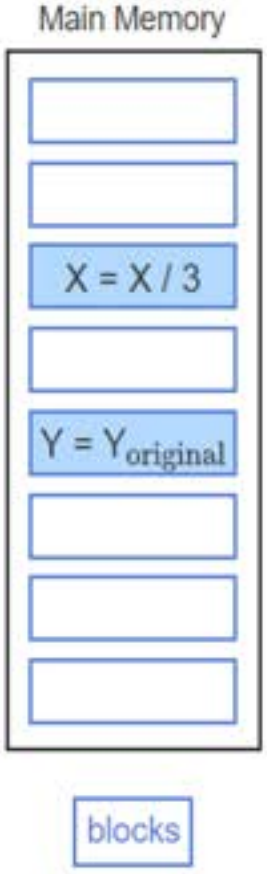
The database writes a new X value in main memory and commits T_1.

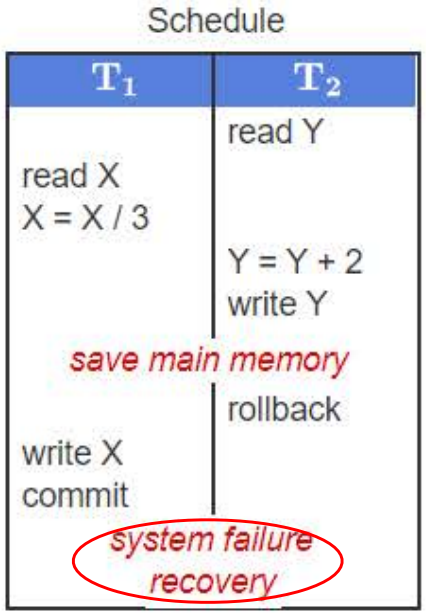
PARTICIPATION ACTIVITY | 20.1.1: System failure.

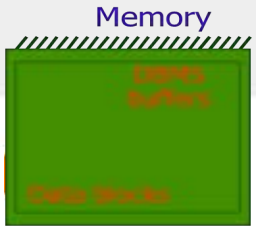


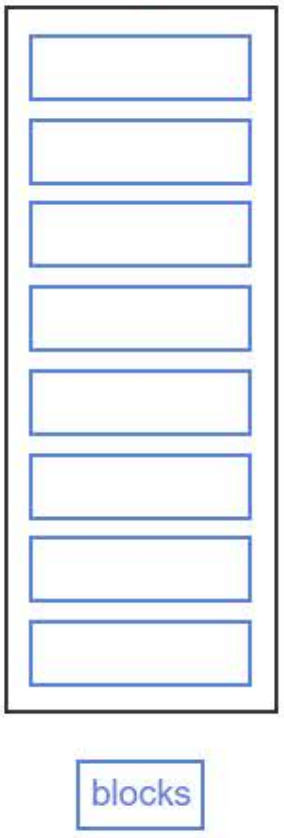
Before main memory is saved to storage media, the operating system fails, and main memory is lost.

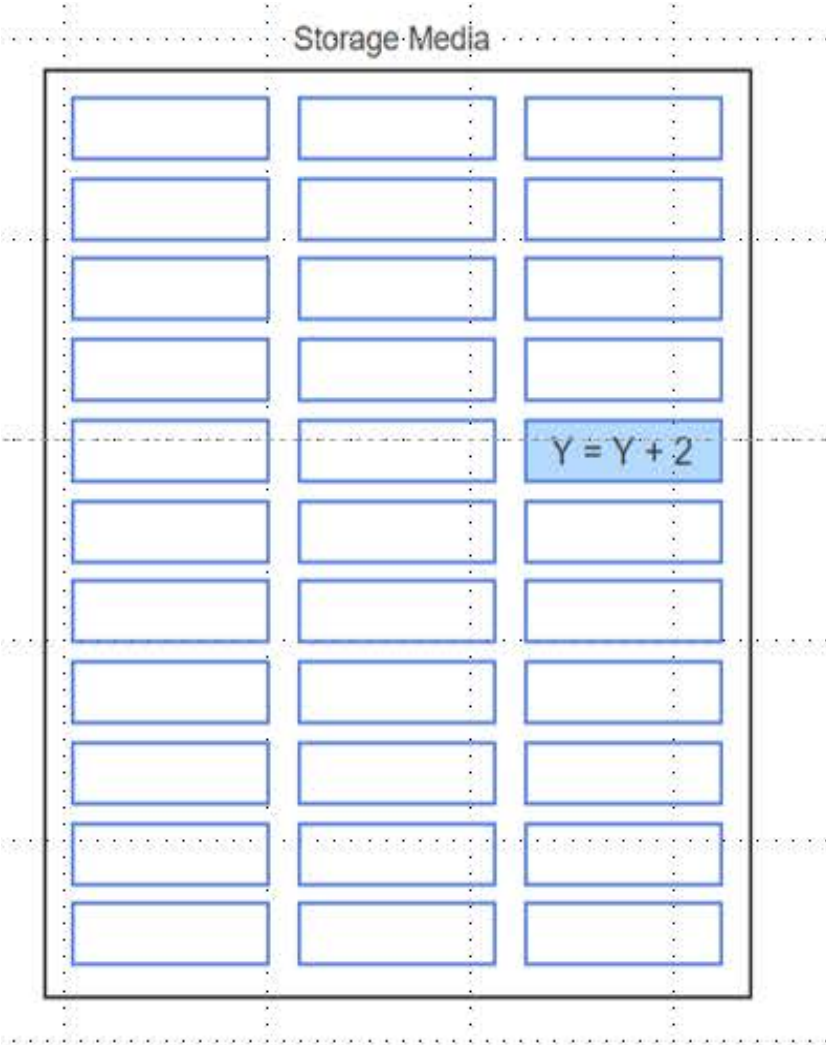
PARTICIPATION ACTIVITY | 20.1.1: System failure.









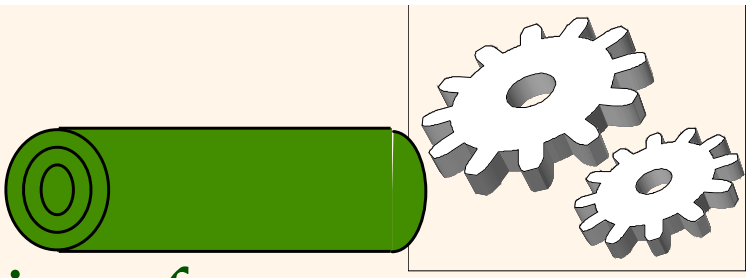


REDO **UNDO**

The recovery system must restore 'X = X / 3' and roll back 'Y = Y + 2' in storage media.

D A

Basic Idea: Logging



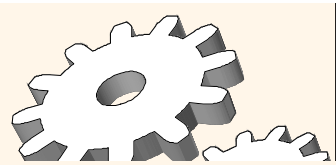
- Record **REDO** and **UNDO** information, for every UPDATE, in a ***Log***.
  - Sequential writes to ***Log*** (put it on a separate disk).
  - Minimal info (diff) written to ***Log***, so multiple UPDATEs fit in a single **Log** page.

- **Log**: An ordered list of **REDO**/**UNDO** actions
  - ***Log*** records contain
    - UPDATE information<XID, pageID, offset, length, old data, new data>
    - and additional **control info: COMMIT**; **ABORT**; …

A Transaction Log

TRL_ ID	TRX_ NUM	PREV PTR	NEXT PTR	OPERATION	TABLE	ROW ID	ATTRIBUTE	BEFORE VALUE	AFTER VALUE
341	101	Null	352	START	****Start				
352	101	341	363	UPDATE	PRODUCT	1558-QW1	PROD_ QOH	25	23
365	101	363	Null	COMMIT	**** End of				

The Transaction Log



Transaction Log stores (LSN – Log Sequence #):

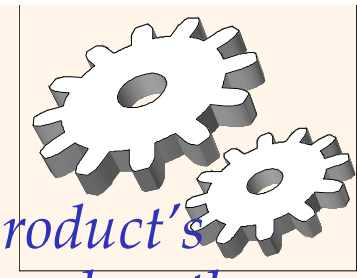
- A record for the beginning of **Transaction** (START)
- For each **Transaction** component (**SQL** statement):
  - Type of operation being performed (UPDATE)
  - Names of objects affected by **Transaction**
  - "Before" and "after" values for updated fields
  - Pointers to previous and next **Transaction** Log entries (LSN) for the ***same*** **Transaction**
- Ending (COMMIT) of the **Transaction**
- A record for the end of **Transaction** (END)

A Transaction Log

TRL	TRX	PREV	NEXT	OPERATION	TABLE	ROW	ATTRIBUTE	BEFORE	AFTER
341	101	Null	352	START	****Start Transaction				
						QW1	QOH		
363	101	352	365	UPDATE	CUSTOMER	10011	CUST_	525.75	615.73
365	101	363	Null	COMMIT	**** End of Transaction				

TRX_NUM = Transaction number
(Note: The transaction number is auto-

The Transaction Log

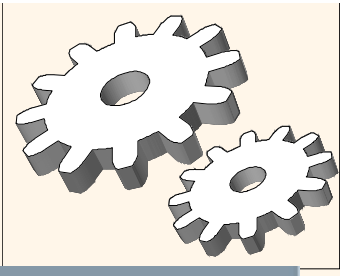


- The following is a simplified sales example, which *updates a product's quantity on hand (PROD_QOH) and the customer's 10011 balance when the customer buys 2 units of product 1558-QW1 priced at $43.99 per unit (for a total of $87.98) and charges the purchase to his/her account*:

```
START;

UPDATE   PRODUCT
SET      PROD_QOH = PROD_QOH - 2
WHERE    PROD_CODE = '1558-QW1';

UPDATE   CUSTOMER
SET      CUST_BALANCE = CUST_BALANCE + 87.98
WHERE    CUST_NUMBER = '10011';

COMMIT;
END;
```

How many **Log** entries would this **Transaction** produce? **5**

The Transaction Log



LSN – Log Sequnce Number

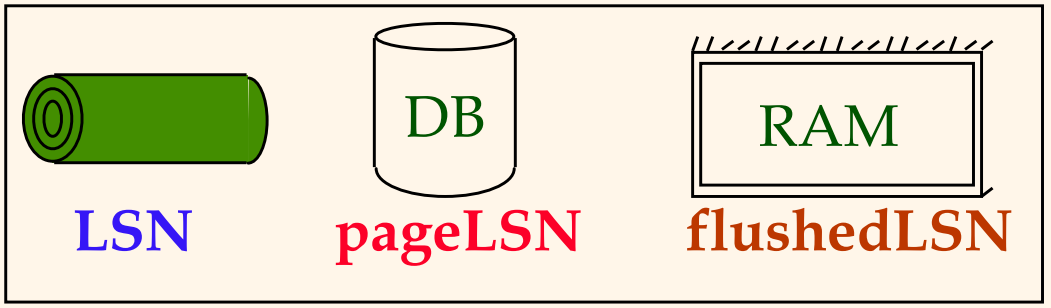
A Transaction Log

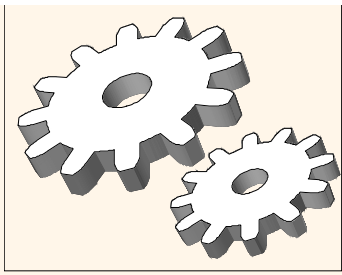
TRL_ ID	TRX_ NUM	PREV PTR	NEXT PTR	OPERATION	TABLE	ROW ID	ATTRIBUTE	BEFORE VALUE	AFTER VALUE
341	101	Null	352	START	****Start Transaction				
352	101	341	363	UPDATE	PRODUCT	1558-QW1	PROD_ QOH	25	23
363	101	352	365	UPDATE	CUSTOMER	10011	CUST_ BALANCE	525.75	615.73
365	101	363	367	COMMIT	**** End of Transaction				
367	101	365	Null						

END

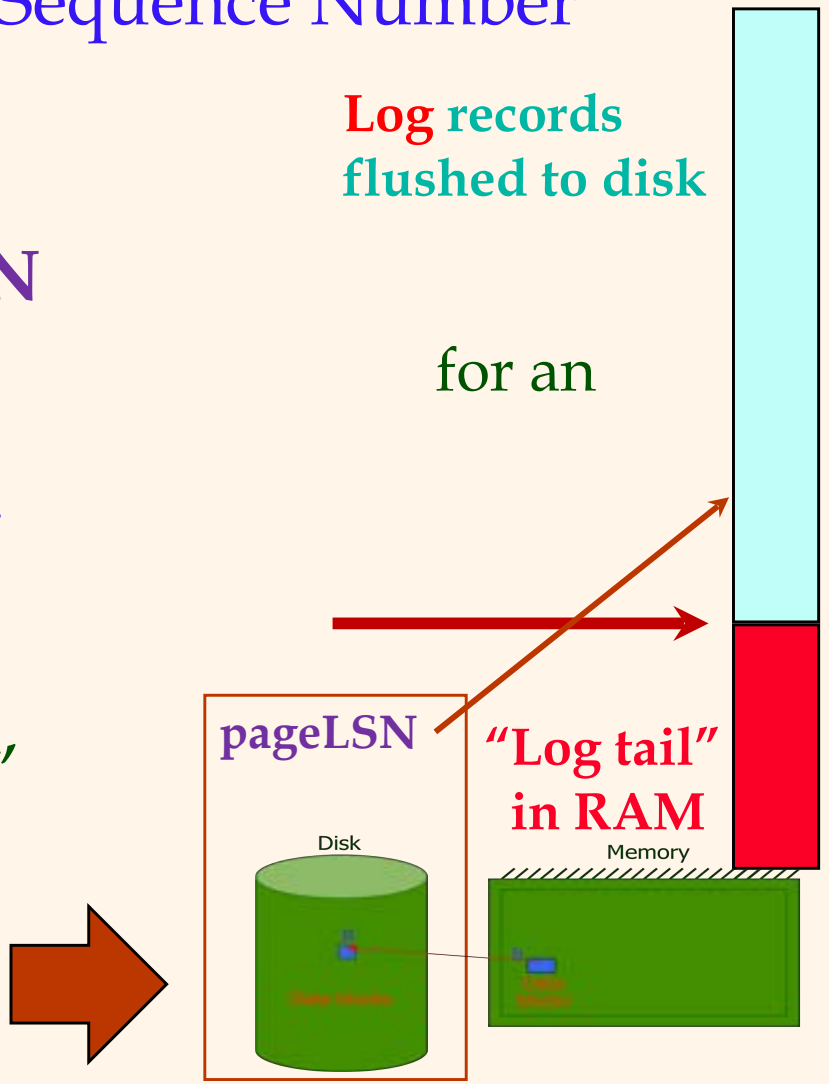
```
START;
UPDATE   PRODUCT
SET      PROD_QOH = PROD_QOH - 2
WHERE    PROD_CODE = '1558-QW1';

UPDATE   CUSTOMER
SET      CUST_BALANCE = CUST_BALANCE + 87.98
WHERE    CUST_NUMBER = '10011';

COMMIT;
END;
```

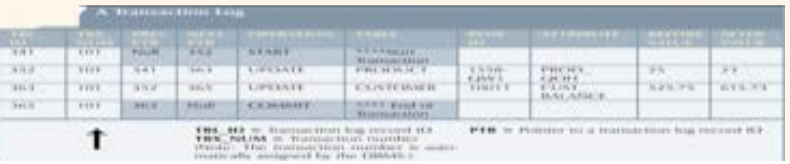
WAL & the Log



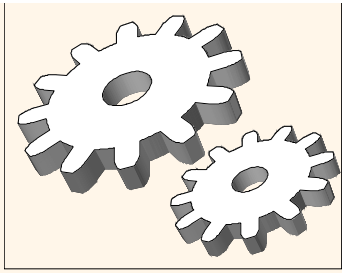


- Each **Log** record has a unique Log Sequence Number (**LSN**)
  - **LSN**s always increasing.
- Each ***data page*** contains a **pageLSN**
  - The LSN of the ***first Log*** *record* for an update to that **page**.
- System keeps track of **flushedLSN**
  - The max LSN flushed so far.
- WAL: *Before* a ***data page*** is written,
  pageLSN ≤ flushedLSN





Log Records



Possible Log record types:

- **UPDATE**
- **Start**
- **Commit**
- **Abort**
- **End** (signifies end of **Commit** or **Abort**)
- **Compensation Log Records** (CLRs)
  - for **UNDO** actions (LSN of the next **UNDO** undonextLSN)

Log Record fields:

prevLSN
XID
type

UPDATE records only:
- pageID
- length
- offset
- before-image
- after-image

A Transaction Log

TRL_ID	TRX_NUM	PREV PTR	NEXT PTR	OPERATION	TABLE	ROW ID	ATTRIBUTE	BEFORE VALUE	AFTER VALUE
341	101	Null	352	START	****Start Transaction				
352	101	341	363	UPDATE	PRODUCT	1558-QWI	PROD_QOH	25	23
363	101	352	365	UPDATE	CUSTOMER	10011	CUST_BALANCE	525.75	615.73
365	101	363	Null	COMMIT	**** End of Transaction				

TRL_ID = Transaction log record ID
TRX_NUM = Transaction number
(Note: The transaction number is automatically assigned by the DBMS.)
PTR = Pointer to a transaction log record ID

Recovery log

A ***recovery log*** is a file containing a sequential record of all database operations. The log and database are stored on different storage media so the log survives database failures. The recovery system uses the log to restore the database after a failure.

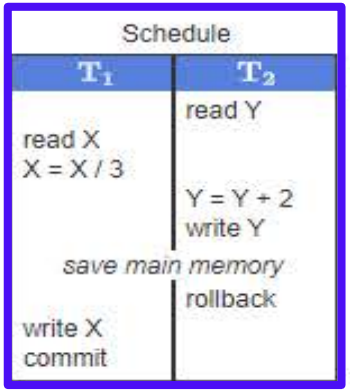
The log contains transaction and data identifiers. Transaction identifiers are assigned by the database system for internal use by the recovery and concurrency systems. Data identifiers correspond to table name, column name, and primary key value, but are compressed or encoded for efficiency.

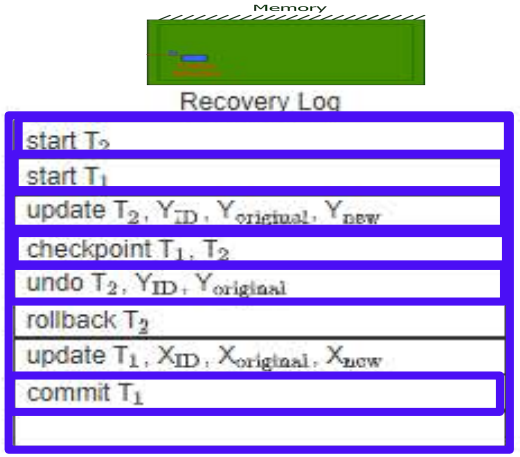
The recovery log contains four types of records:

1. An ***update record*** indicates a transaction has changed data. Update records include the transaction identifier, the data identifier, the original data value, and the new data value. Update records may optionally track insert and delete operations.
2. A ***compensation record*** also known as an ***undo record*** , indicates data has been restored to the original value during a rollback. Compensation records include the transaction identifier, the data identifier, and the restored (original) data value. Compensation records are necessary because the log must capture every database change, including changes executed by a rollback.
3. A ***transaction record*** indicates a transaction boundary. Three types of transaction records exist: start, commit, and rollback. Transaction records include the 'start', 'commit', or 'rollback' indicator and the transaction identifier.
4. A ***checkpoint record*** indicates that all data in main memory has been saved on storage media. When the database executes a checkpoint, transaction processing is suspended while all unsaved data and log records are written to storage media. In the event of a system failure, the recovery system reads only log records following the last checkpoint, rather than the entire file. Checkpoint records include a 'checkpoint' indicator along with the identifiers of all transactions that are active (uncommitted) at the time of the checkpoint.

The above four record types provide a complete history of database operations and enable recovery from all three failure scenarios.

PARTICIPATION ACTIVITY 20.1.3: Recovery log records.





Recovery log

A ***recovery log*** is a file containing a sequential record of all database operations. The log and database are stored on different storage media so the log survives database failures. The recovery system uses the log to restore the database after a failure.

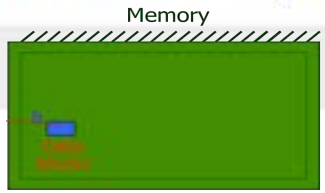
The log contains transaction and data identifiers. Transaction identifiers are assigned by the database system for internal use by the recovery and concurrency systems. Data identifiers correspond to table name, column name, and primary key value, but are compressed or encoded for efficiency.

The recovery log contains four types of records:

1. An ***update record*** indicates a transaction has changed data. Update records include the transaction identifier, the data identifier, the original data value, and the new data value. Update records may optionally track insert and delete operations.
2. A ***compensation record*** , also known as an ***undo record*** , indicates data has been restored to the original value during a rollback. Compensation records include the transaction identifier, the data identifier, and the restored (original) data value. Compensation records are necessary because the log must capture every database change, including changes executed by a rollback.
3. A ***transaction record*** indicates a transaction boundary. Three types of transaction records exist: start, commit, and rollback. Transaction records include the 'start', 'commit', or 'rollback' indicator and the transaction identifier.
4. A ***checkpoint record*** indicates that all data in main memory has been saved on storage media. When the database executes a checkpoint, transaction processing is suspended while all unsaved data and log records are written to storage media. In the event of a system failure, the recovery system reads only log records following the last checkpoint, rather than the entire file. Checkpoint records include a 'checkpoint' indicator along with the identifiers of all transactions that are active (uncommitted) at the time of the checkpoint.

The above four record types provide a complete history of database operations and enable recovery from all three failure scenarios.

PARTICIPATION ACTIVITY 20.1.3: Recovery log records.



Schedule

T_1	T_2
	read Y
read X	
X = X / 3	
	Y = Y + 2
	write Y
save main memory	
	rollback
write X	
commit	

Recovery Log

start T_2
start T_1

"Log tail" in RAM

When the transactions execute, start records are written in the recovery log.

Recovery log

A ***recovery log*** is a file containing a sequential record of all database operations. The log and database are stored on different storage media so the log survives database failures. The recovery system uses the log to restore the database after a failure.

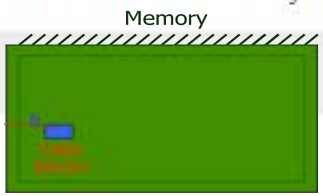
The log contains transaction and data identifiers. Transaction identifiers are assigned by the database system for internal use by the recovery and concurrency systems. Data identifiers correspond to table name, column name, and primary key value, but are compressed or encoded for efficiency.

The recovery log contains four types of records:

1. An ***update record*** indicates a transaction has changed data. Update records include the transaction identifier, the data identifier, the original data value, and the new data value. Update records may optionally track insert and delete operations.
2. A ***compensation record*** , also known as an ***undo record*** , indicates data has been restored to the original value during a rollback. Compensation records include the transaction identifier, the data identifier, and the restored (original) data value. Compensation records are necessary because the log must capture every database change, including changes executed by a rollback.
3. A ***transaction record*** indicates a transaction boundary. Three types of transaction records exist: start, commit, and rollback. Transaction records include the 'start', 'commit', or 'rollback' indicator and the transaction identifier.
4. A ***checkpoint record*** indicates that all data in main memory has been saved on storage media. When the database executes a checkpoint, transaction processing is suspended while all unsaved data and log records are written to storage media. In the event of a system failure, the recovery system reads only log records following the last checkpoint, rather than the entire file. Checkpoint records include a 'checkpoint' indicator along with the identifiers of all transactions that are active (uncommitted) at the time of the checkpoint.

The above four record types provide a complete history of database operations and enable recovery from all three failure scenarios.

PARTICIPATION ACTIVITY 20.1.3: Recovery log records.



Schedule

T_1	T_2
	read Y
read X	
X = X / 3	
	Y = Y + 2
	write Y
save main memory	
	rollback
write X	
commit	

Recovery Log

start T_2
start T_1
update T_2, Y_{ID}, $Y_{original}$, Y_{new}

"Log tail" in RAM

When T_2 writes Y, an update record is written in the log.

Recovery log

A ***recovery log*** is a file containing a sequential record of all database operations. The log and database are stored on different storage media so the log survives database failures. The recovery system uses the log to restore the database after a failure.

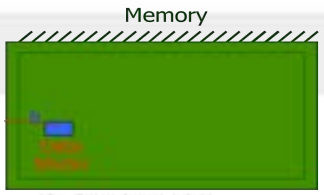
The log contains transaction and data identifiers. Transaction identifiers are assigned by the database system for internal use by the recovery and concurrency systems. Data identifiers correspond to table name, column name, and primary key value, but are compressed or encoded for efficiency.

The recovery log contains four types of records:

1. An ***update record*** indicates a transaction has changed data. Update records include the transaction identifier, the data identifier, the original data value, and the new data value. Update records may optionally track insert and delete operations.
2. A ***compensation record*** , also known as an ***undo record*** , indicates data has been restored to the original value during a rollback. Compensation records include the transaction identifier, the data identifier, and the restored (original) data value. Compensation records are necessary because the log must capture every database change, including changes executed by a rollback.
3. A ***transaction record*** indicates a transaction boundary. Three types of transaction records exist: start, commit, and rollback. Transaction records include the 'start', 'commit', or 'rollback' indicator and the transaction identifier.
4. A ***checkpoint record*** indicates that all data in main memory has been saved on storage media. When the database executes a checkpoint, transaction processing is suspended while all unsaved data and log records are written to storage media. In the event of a system failure, the recovery system reads only log records following the last checkpoint, rather than the entire file. Checkpoint records include a 'checkpoint' indicator along with the identifiers of all transactions that are active (uncommitted) at the time of the checkpoint.

The above four record types provide a complete history of database operations and enable recovery from all three failure scenarios.

PARTICIPATION ACTIVITY 20.1.3: Recovery log records.



Schedule

T_1	T_2
	read Y
read X	
X = X / 3	
	Y = Y + 2
	write Y
	save main memory
	rollback
write X	
commit	

Recovery Log

start T_2
start T_1
update T_2 , Y_{ID} , $Y_{original}$, Y_{new}
checkpoint T_1 , T_2

"Log tail" in RAM

When the system saves all updates from main memory to storage media, a checkpoint record is written in the log.

Recovery log

A ***recovery log*** is a file containing a sequential record of all database operations. The log and database are stored on different storage media so the log survives database failures. The recovery system uses the log to restore the database after a failure.

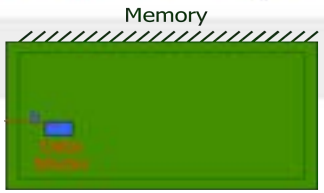
The log contains transaction and data identifiers. Transaction identifiers are assigned by the database system for internal use by the recovery and concurrency systems. Data identifiers correspond to table name, column name, and primary key value, but are compressed or encoded for efficiency.

The recovery log contains four types of records:

1. An ***update record*** indicates a transaction has changed data. Update records include the transaction identifier, the data identifier, the original data value, and the new data value. Update records may optionally track insert and delete operations.
2. A ***compensation record*** , also known as an ***undo record*** , indicates data has been restored to the original value during a rollback. Compensation records include the transaction identifier, the data identifier, and the restored (original) data value. Compensation records are necessary because the log must capture every database change, including changes executed by a rollback.
3. A ***transaction record*** indicates a transaction boundary. Three types of transaction records exist: start, commit, and rollback. Transaction records include the 'start', 'commit', or 'rollback' indicator and the transaction identifier.
4. A ***checkpoint record*** indicates that all data in main memory has been saved on storage media. When the database executes a checkpoint, transaction processing is suspended while all unsaved data and log records are written to storage media. In the event of a system failure, the recovery system reads only log records following the last checkpoint, rather than the entire file. Checkpoint records include a 'checkpoint' indicator along with the identifiers of all transactions that are active (uncommitted) at the time of the checkpoint.

The above four record types provide a complete history of database operations and enable recovery from all three failure scenarios.

PARTICIPATION ACTIVITY 20.1.3: Recovery log records.



Schedule

T_1	T_2
	read Y
read X	
X = X / 3	
	Y = Y + 2
	write Y
save main memory	
	rollback
write X	
commit	

Recovery Log

start T_2
start T_1
update T_2 , Y_{ID} , $Y_{original}$, Y_{new}
checkpoint T_1 , T_2
undo T_2 , Y_{ID} , $Y_{original}$
rollback T_2

"Log tail" in RAM

When T_2 rolls back, compensation and rollback records are written in the log.

Recovery log

A ***recovery log*** is a file containing a sequential record of all database operations. The log and database are stored on different storage media so the log survives database failures. The recovery system uses the log to restore the database after a failure.

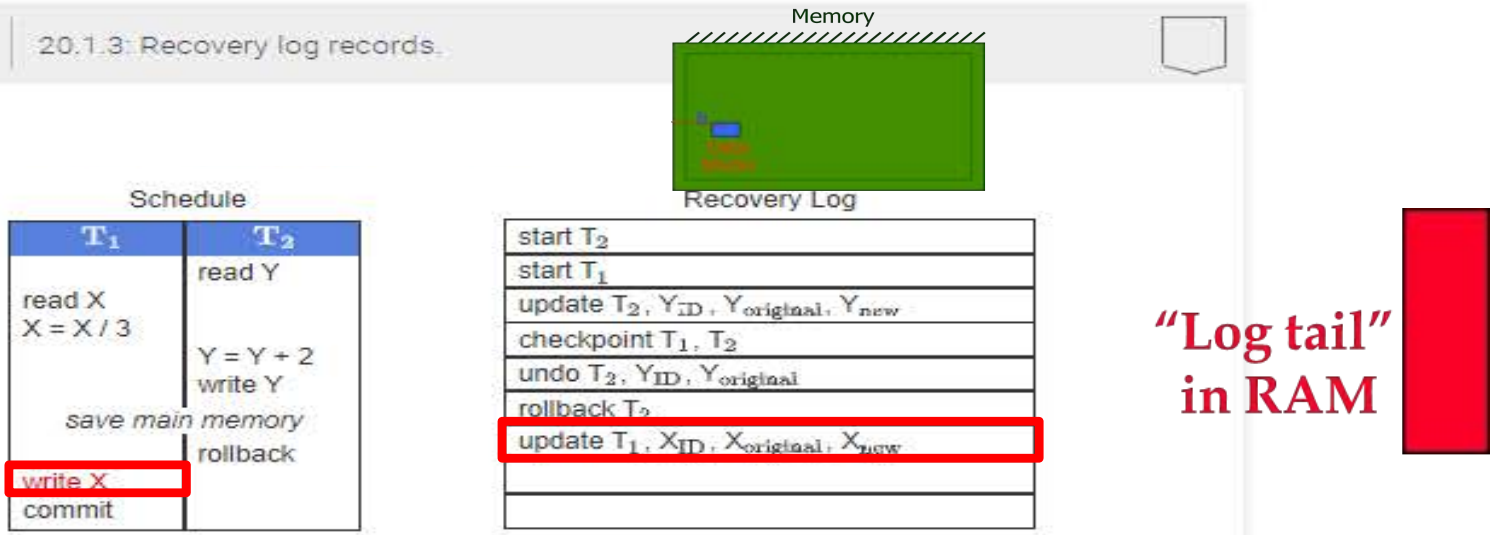
The log contains transaction and data identifiers. Transaction identifiers are assigned by the database system for internal use by the recovery and concurrency systems. Data identifiers correspond to table name, column name, and primary key value, but are compressed or encoded for efficiency.

The recovery log contains four types of records:

1. An ***update record*** indicates a transaction has changed data. Update records include the transaction identifier, the data identifier, the original data value, and the new data value. Update records may optionally track insert and delete operations.
2. A ***compensation record*** , also known as an ***undo record*** , indicates data has been restored to the original value during a rollback. Compensation records include the transaction identifier, the data identifier, and the restored (original) data value. Compensation records are necessary because the log must capture every database change, including changes executed by a rollback.
3. A ***transaction record*** indicates a transaction boundary. Three types of transaction records exist: start, commit, and rollback. Transaction records include the 'start', 'commit', or 'rollback' indicator and the transaction identifier.
4. A ***checkpoint record*** indicates that all data in main memory has been saved on storage media. When the database executes a checkpoint, transaction processing is suspended while all unsaved data and log records are written to storage media. In the event of a system failure, the recovery system reads only log records following the last checkpoint, rather than the entire file. Checkpoint records include a 'checkpoint' indicator along with the identifiers of all transactions that are active (uncommitted) at the time of the checkpoint.

The above four record types provide a complete history of database operations and enable recovery from all three failure scenarios.

PARTICIPATION ACTIVITY | 20.1.3: Recovery log records.



When T_1 writes X, an update record is written in the log

Recovery log

A ***recovery log*** is a file containing a sequential record of all database operations. The log and database are stored on different storage media so the log survives database failures. The recovery system uses the log to restore the database after a failure.

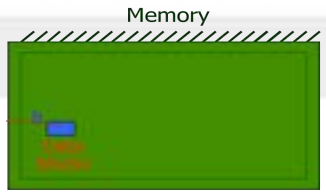
The log contains transaction and data identifiers. Transaction identifiers are assigned by the database system for internal use by the recovery and concurrency systems. Data identifiers correspond to table name, column name, and primary key value, but are compressed or encoded for efficiency.

The recovery log contains four types of records:

1. An ***update record*** indicates a transaction has changed data. Update records include the transaction identifier, the data identifier, the original data value, and the new data value. Update records may optionally track insert and delete operations.
2. A ***compensation record*** , also known as an ***undo record*** , indicates data has been restored to the original value during a rollback. Compensation records include the transaction identifier, the data identifier, and the restored (original) data value. Compensation records are necessary because the log must capture every database change, including changes executed by a rollback.
3. A ***transaction record*** indicates a transaction boundary. Three types of transaction records exist: start, commit, and rollback. Transaction records include the 'start', 'commit', or 'rollback' indicator and the transaction identifier.
4. A ***checkpoint record*** indicates that all data in main memory has been saved on storage media. When the database executes a checkpoint, transaction processing is suspended while all unsaved data and log records are written to storage media. In the event of a system failure, the recovery system reads only log records following the last checkpoint, rather than the entire file. Checkpoint records include a 'checkpoint' indicator along with the identifiers of all transactions that are active (uncommitted) at the time of the checkpoint.

The above four record types provide a complete history of database operations and enable recovery from all three failure scenarios.

PARTICIPATION ACTIVITY 20.1.3: Recovery log records.



Schedule

T_1	T_2
	read Y
read X	
X = X / 3	
	Y = Y + 2
	write Y
save main memory	
	rollback
write X	
commit	

Recovery Log

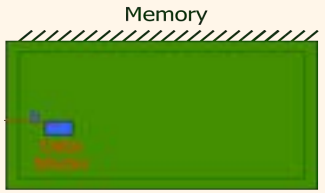
start T_2
start T_1
update T_2, Y_{ID}, $Y_{original}$, Y_{new}
checkpoint T_1, T_2
undo T_2, Y_{ID}, $Y_{original}$
rollback T_2
update T_1, X_{ID}, $X_{original}$, X_{new}
commit

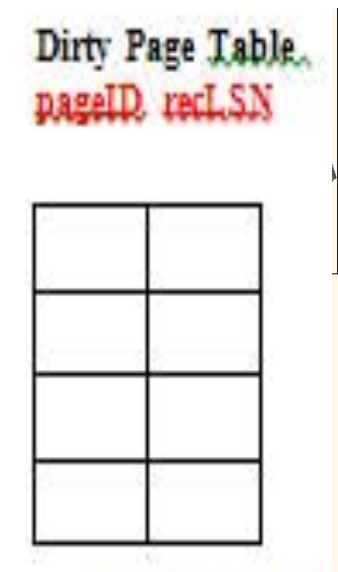
When T_1 commits, a commit record is written in the log.

"Log tail" in RAM

T2 aborted
T1 committed

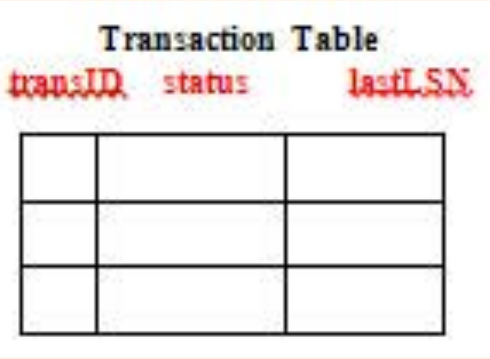
*Other **Log** - Related Data Structures*





Dirty Page Table:

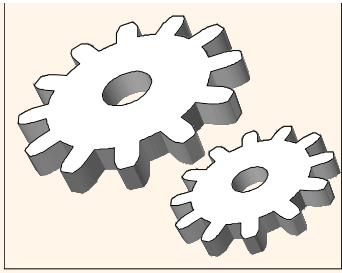
- One entry per Dirty Page in **buffer pool**.
- Contains **rec**LSN or **first**LSN -- the **LSN** of the **Log** record which *first* caused the **Data Page** to be Dirty.



Transaction Table:

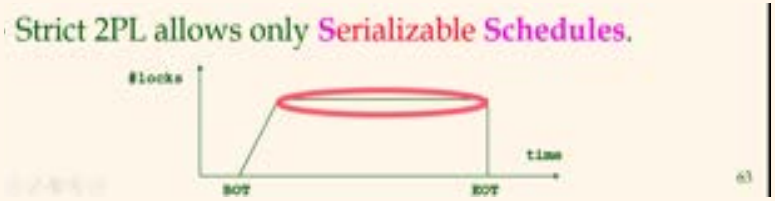
- One entry per ***active*** **Xact**.
- Contains XID, status (Running/Commited/Aborted), and the ***last***LSN of the **Log** record for that **Xact** **last**LSN.

Normal Execution of an Xact

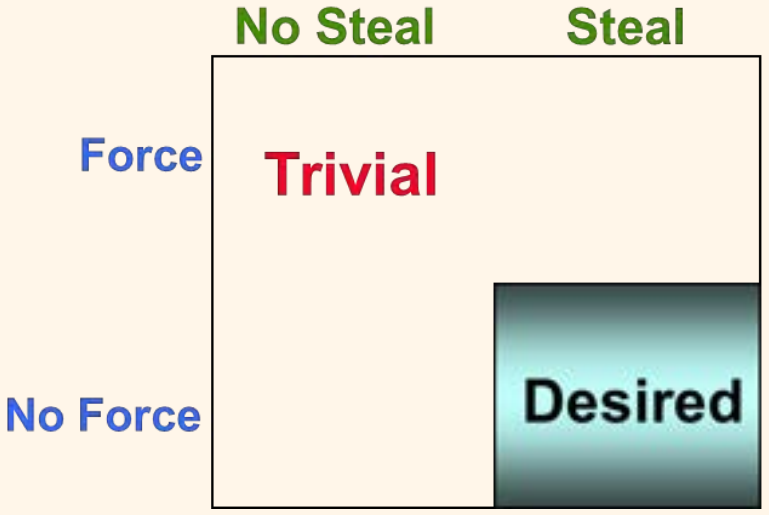


- Series of reads & writes, followed by Commit or Abort.
  - We will assume that write is atomic on disk.
    - In practice, additional details to deal with non-atomic writes.

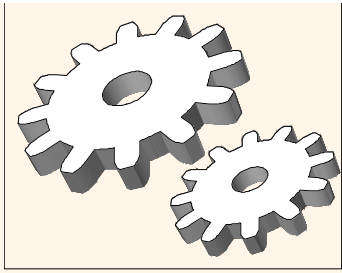
- Strict 2PL.



- STEAL, NO-FORCE buffer management, with **WAL**.



Transaction Recovery



A Transaction Log for Transaction Recovery Examples

TRL ID	TRX NUM	PREV PTR	NEXT PTR	OPERATION	TABLE	ROW ID	ATTRIBUTE	BEFORE VALUE	AFTER VALUE
341	101	Null	352	START	**** Start Transaction				
352	101	341	363	UPDATE	PRODUCT	54778-2T	PROD_QOH	45	43
363	101	352	365	UPDATE	CUSTOMER	10011	CUST_BALANCE	[illegible]	675.62
[illegible]	101	363	Null	COMMIT	**** End of Transaction				
397	106	Null	405	START	**** Start Transaction				
405	106	397	415	INSERT	INVOICE	1009			1009,10016, ...
415	106	405	419	INSERT	LINE	1009,1			1009,1, 89-WRE-Q,1, ...
419	106	415	427	UPDATE	PRODUCT	89-WRE-Q	PROD_QOH	12	11
423				CHECKPOINT					
427	106	419	431	UPDATE	CUSTOMER	10016	CUST_BALANCE	0.00	277.55
431	106	427	457	INSERT	ACCT_TRANSACTION	10007			1007,18-JAN-2004, ...
[illegible]	106	431	Null	COMMIT	**** End of Transaction				
521	155	Null	525	START	**** Start Transaction				
525	155	521	528	UPDATE	PRODUCT	2232/QWE	PROD_QOH	[illegible]	26
[illegible]	155	525	Null	COMMIT	**** End of Transaction				
				***** C*R*A*S*H *****					

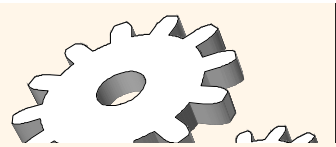
REDO D

REDO D

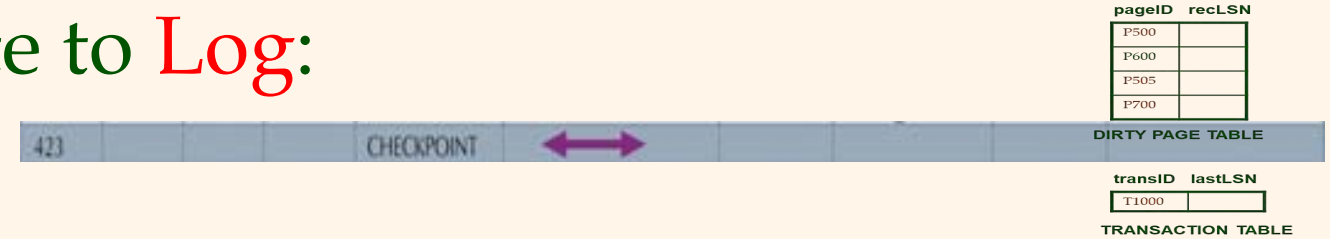
REDO D

Status of transactions before the CRASH?

Checkpointing

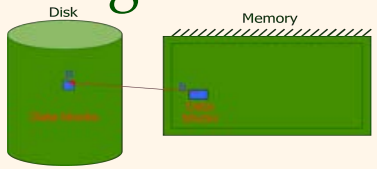


Periodically, the **DBMS** creates a checkpoint, in order to minimize the time taken to recover in the event of a system CRASH. Write to Log:



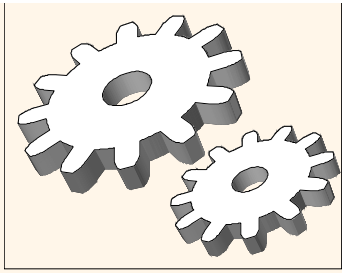
- begin_checkpoint record: Indicates when checkpoint began.

- end_checkpoint record: Contains current *Tact Table* and *Dirty Page Table*. This is a `fuzzy checkpoint':
  - Other Xacts continue to run; so these tables accurate only as of the time of the begin_checkpoint record.
  - No attempt to force **dirty pages** to **disk**; effectiveness of checkpoint limited by **oldest unwritten change to a dirty page**. (So it's a good idea to periodically **flush dirty pages** to **disk**!)



- Store **LSN** of checkpoint record in a **safe place** (***Master* Record**).

The Big Picture: What's Stored Where



LogRecords

prevLSN
XID
type

UPDATE records only

pageID
length
offset
before-image
after-image

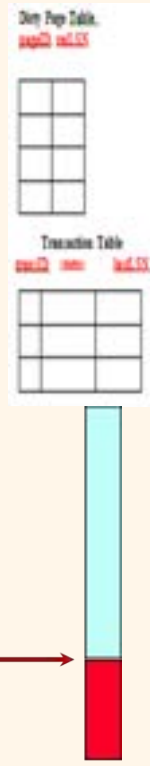
Data pages
each
with a
pageLSN

Master record
(checkpoint)

Dirty Page Table
firstLSN

Xact Table
lastLSN
status

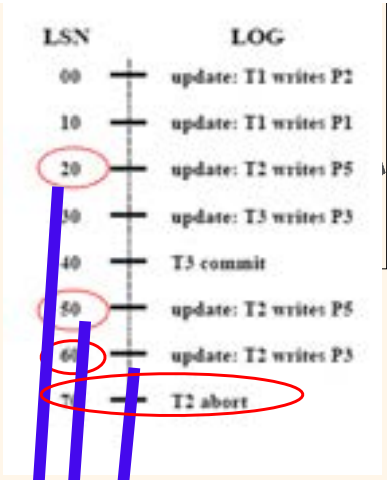
flushedLSN



Simple Transaction Abort

For now, consider an explicit ABORT of a Xact.

- No **CRASH** involved.



We want to "play back" the log in reverse order, **UNDO**ing UPDATES.

- Get last**LSN** of Xact from **Tact Table**.

Transaction Table

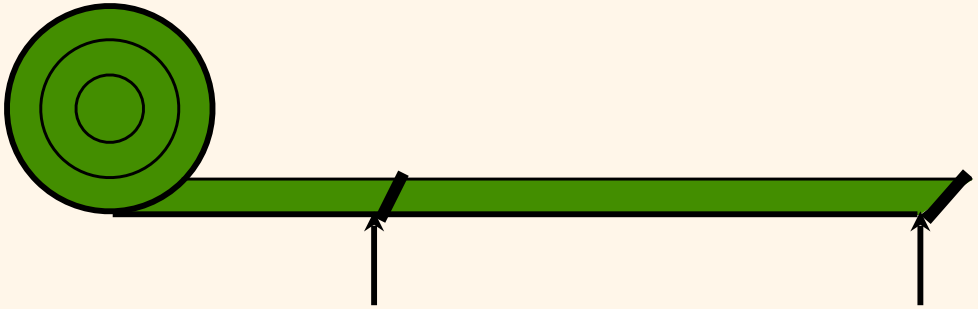
transID	status	lastLSN
T2	Abort	70
T1	Active	10

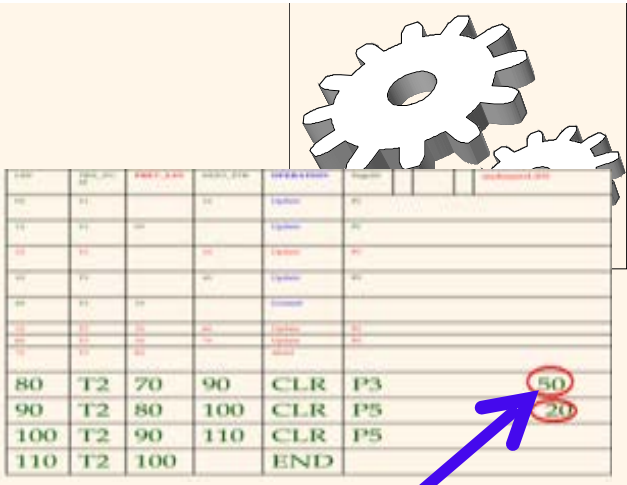
- Can follow chain of Log records backward via the prev**LSN** field.

- Before starting **UNDO**, write an *Abort* log record (**CLR**).
  - For recovering from **CRASH** during **UNDO**!

LSN	TRX_NUM	PREV_LSN	NEXT_PTR	OPERATION	PageID	undonextLSN
80	T2	70	90	CLR	P3	50
90	T2	80	100	CLR	P5	20
100	T2	90	110	CLR	P5	
110	T2	100		END		

Abort





80	T2	70	90	CLR	P3	50
90	T2	80	100	CLR	P5	20
100	T2	90	110	CLR	P5	
110	T2	100		END		

- Before restoring old value of a **page**, write a CLR (Compensation Log Record):
  - You continue logging while you **UNDO**!!
  - CLR has one extra field: undonextLSN
    - Points to the next LSN to **UNDO** (i.e. the prevLSN of the record we're currently undoing).
  - CLRs *never* UNDOne (but they might be REDOne when repeating history: guarantees Atomicity!)

- At end of **UNDO**, write an "**END**" log record.

Recovery from transaction failure

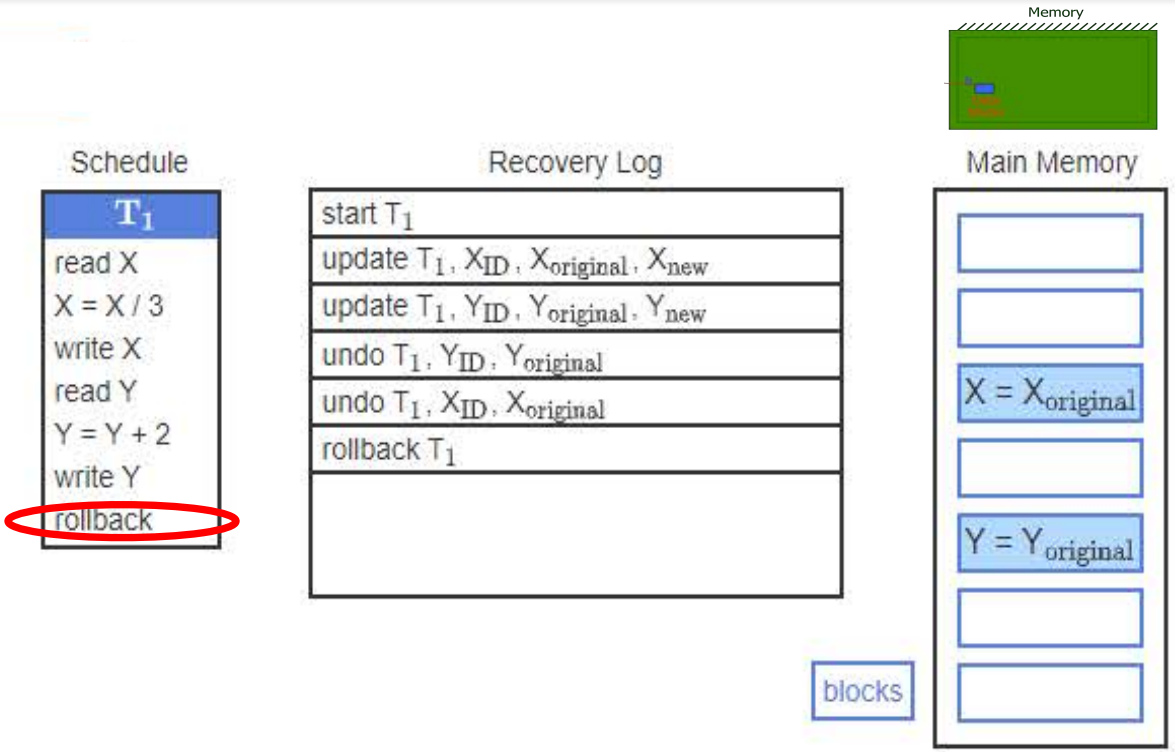
Recovery from a transaction failure is relatively simple. A database component, such as the concurrency system, instructs the recovery system to roll back transaction T. The recovery system reads the recovery log **backwards**, searching for update records for T. For each update record, the recovery system:

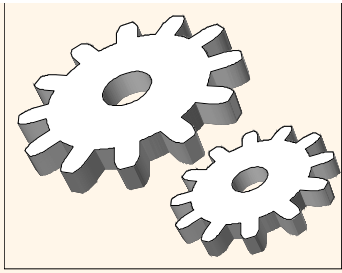
- Restores data D to original value V in the update record.
- Writes a compensation record for T, D, and V to the end of the log.

Eventually, the recovery system reads the 'start T' transaction record and writes a 'rollback T' record to the end of the log. At this point, the rollback is complete.

The recovery system reads the log backwards because data must be restored in reverse order of transaction operations.

PARTICIPATION ACTIVITY 20.1.5: Recovery from transaction failure.



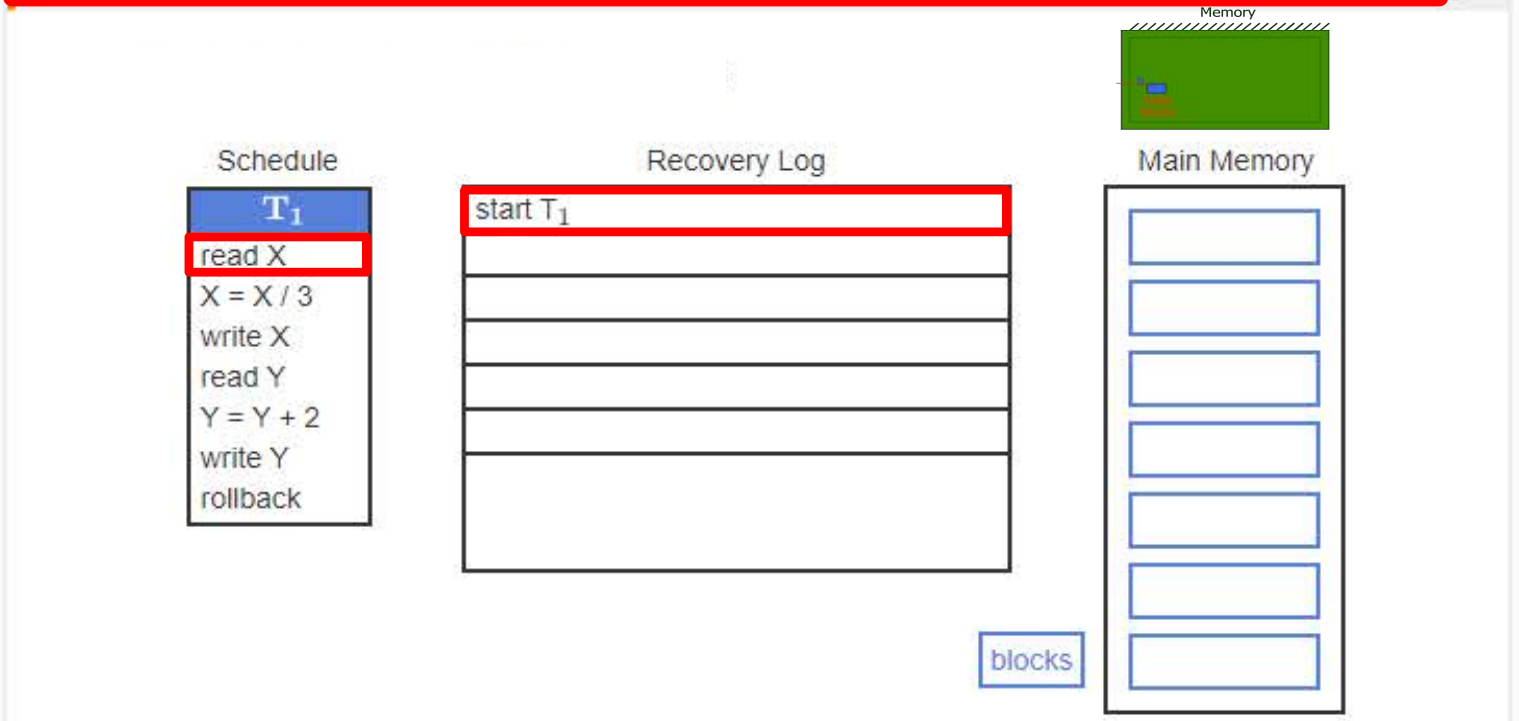


A Transaction Log for Transaction Recovery Examples

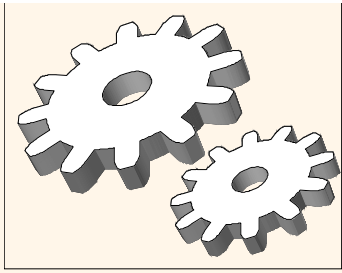
TRL ID	TRX NUM	PREV PTR	NEXT PTR	OPERATION	TABLE	ROW ID	ATTRIBUTE	BEFORE VALUE	AFTER VALUE
341	101	Null	352	START	**** Start Transaction				
352	101	341	363	UPDATE	PRODUCT	54778-2T	PROD_QOH	45	43
363	101	352	365	UPDATE	CUSTOMER	10011	CUST_BALANCE	615.73	675.62
365	101	363	Null	COMMIT	**** End of Transaction				
397	106	Null	405	START	**** Start Transaction				
405	106	397	415	INSERT	INVOICE	1009			1009,10016, ..
415	106	405	419	INSERT	LINE	1009,1			1009,1, 89-WRE-Q,1, ..
419	106	415	427	UPDATE	PRODUCT	89-WRE-Q	PROD_QOH	12	11
423				CHECKPOINT					
427	106	419	431	UPDATE	CUSTOMER	10016	CUST_BALANCE	0.00	277.55
431	106	427	457	INSERT	ACCT_TRANSACTION	10007			1007,18-JAN-2004, ..
457	106	431	Null	COMMIT	**** End of Transaction				
521	155	Null	525	START	**** Start Transaction				
525	155	521	528	UPDATE	PRODUCT	2232/QWE	PROD_QOH	6	26
528	155	525	Null	COMMIT	**** End of Transaction				

* * * * * C *R*A* S* H * * * * *

PARTICIPATION ACTIVITY 20.1.5: Recovery from transaction failure.



As the transaction executes, a start record is written to the recovery log.

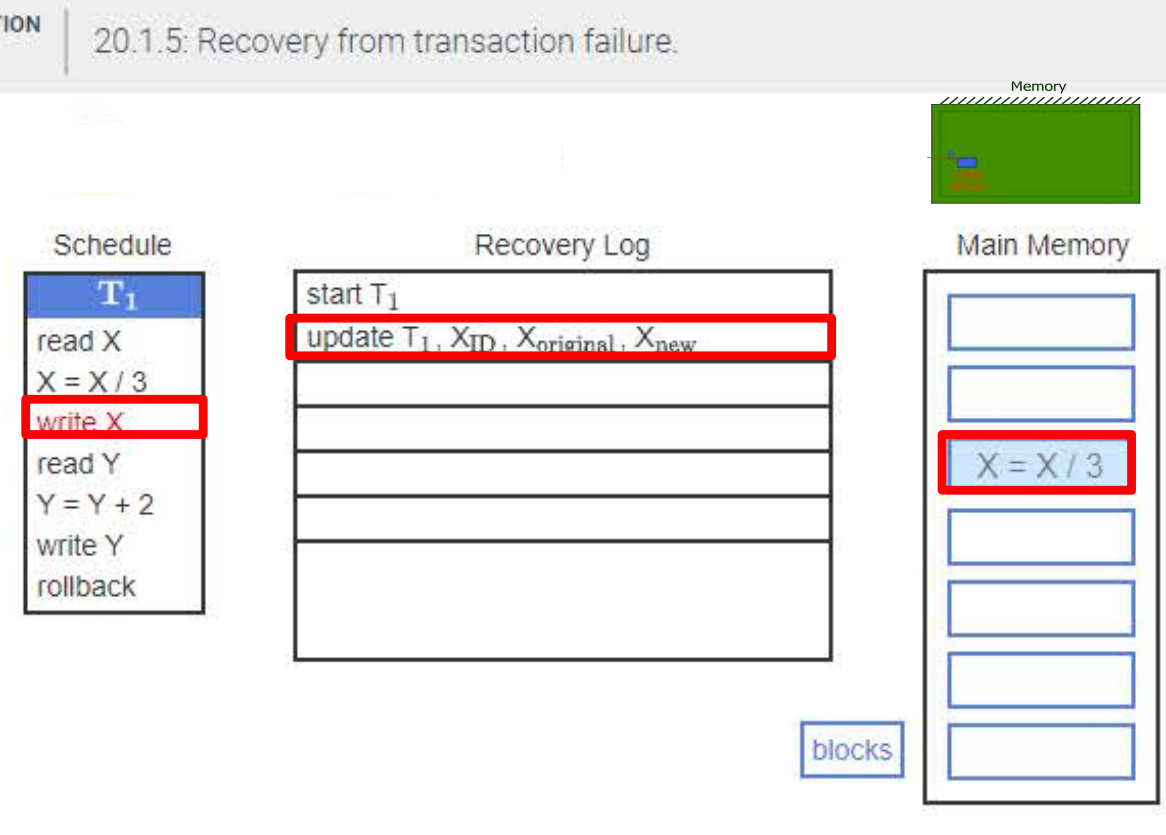


A Transaction Log for Transaction Recovery Examples

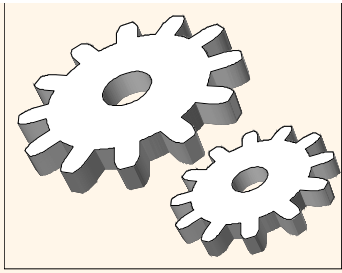
TRL ID	TRX NUM	PREV PTR	NEXT PTR	OPERATION	TABLE	ROW ID	ATTRIBUTE	BEFORE VALUE	AFTER VALUE
341	101	Null	352	START	**** Start Transaction				
352	101	341	363	UPDATE	PRODUCT	54778-2T	PROD_QOH	45	43
363	101	352	365	UPDATE	CUSTOMER	10011	CUST_BALANCE	615.73	675.62
365	101	363	Null	COMMIT	**** End of Transaction				
397	106	Null	405	START	**** Start Transaction				
405	106	397	415	INSERT	INVOICE	1009			1009,10016, ..
415	106	405	419	INSERT	LINE	1009,1			1009,1, 89-WRE-Q,1, ..
419	106	415	427	UPDATE	PRODUCT	89-WRE-Q	PROD_QOH	12	11
423				CHECKPOINT					
427	106	419	431	UPDATE	CUSTOMER	10016	CUST_BALANCE	0.00	277.55
431	106	427	457	INSERT	ACCT_TRANSACTION	10007			1007,18-JAN-2004, ..
457	106	431	Null	COMMIT	**** End of Transaction				
521	155	Null	525	START	**** Start Transaction				
525	155	521	528	UPDATE	PRODUCT	2232/QWE	PROD_QOH	6	26
528	155	525	Null	COMMIT	**** End of Transaction				

* * * * * C *R*A* S* H * * * * *

PARTICIPATION ACTIVITY 20.1.5: Recovery from transaction failure.



Write statements generate update records in the log and save data in main memory.

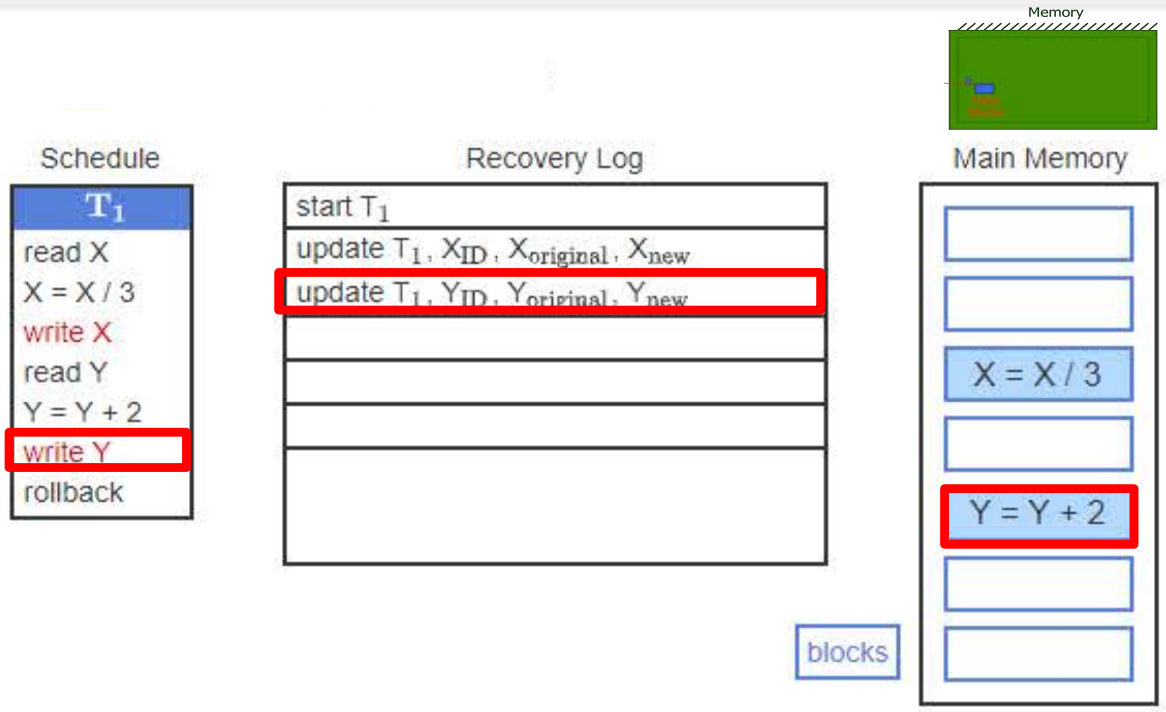


A Transaction Log for Transaction Recovery Examples

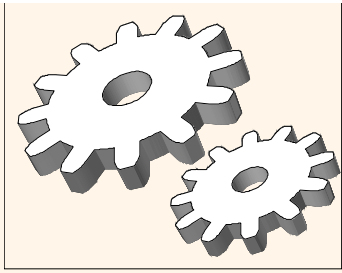
TRL ID	TRX NUM	PREV PTR	NEXT PTR	OPERATION	TABLE	ROW ID	ATTRIBUTE	BEFORE VALUE	AFTER VALUE
341	101	Null	352	START	**** Start Transaction				
352	101	341	363	UPDATE	PRODUCT	54778-2T	PROD_QOH	45	43
363	101	352	365	UPDATE	CUSTOMER	10011	CUST_BALANCE	615.73	675.62
365	101	363	Null	COMMIT	**** End of Transaction				
397	106	Null	405	START	**** Start Transaction				
405	106	397	415	INSERT	INVOICE	1009			1009,10016, ..
415	106	405	419	INSERT	LINE	1009,1			1009,1, 89-WRE-Q,1, ..
419	106	415	427	UPDATE	PRODUCT	89-WRE-Q	PROD_QOH	12	11
423				CHECKPOINT					
427	106	419	431	UPDATE	CUSTOMER	10016	CUST_BALANCE	0.00	277.55
431	106	427	457	INSERT	ACCT_TRANSACTION	10007			1007,18-JAN-2004, ..
457	106	431	Null	COMMIT	**** End of Transaction				
521	155	Null	525	START	**** Start Transaction				
525	155	521	528	UPDATE	PRODUCT	2232/QWE	PROD_QOH	6	26
528	155	525	Null	COMMIT	**** End of Transaction				

* * * * * C *R*A* S* H * * * * *

PARTICIPATION ACTIVITY 20.1.5: Recovery from transaction failure.



Write statements generate update records in the log and save data in main memory.

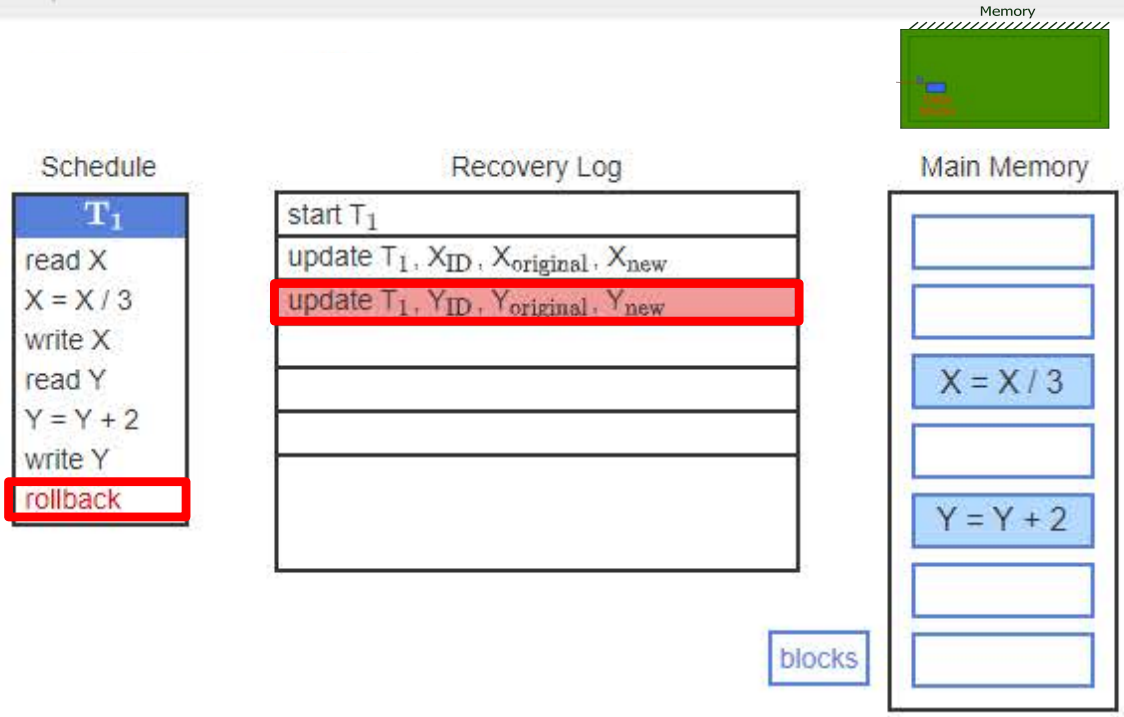


A Transaction Log for Transaction Recovery Examples

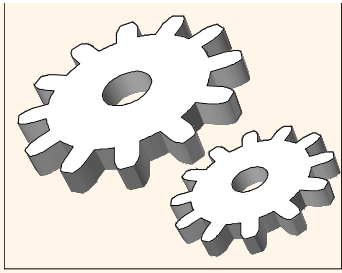
TRL ID	TRX NUM	PREV PTR	NEXT PTR	OPERATION	TABLE	ROW ID	ATTRIBUTE	BEFORE VALUE	AFTER VALUE
341	101	Null	352	START	**** Start Transaction				
352	101	341	363	UPDATE	PRODUCT	54778-2T	PROD_QOH	45	43
363	101	352	365	UPDATE	CUSTOMER	10011	CUST_BALANCE	615.73	675.62
365	101	363	Null	COMMIT	**** End of Transaction				
397	106	Null	405	START	**** Start Transaction				
405	106	397	415	INSERT	INVOICE	1009			1009,10016, ..
415	106	405	419	INSERT	LINE	1009,1			1009,1, 89-WRE-Q,1, ..
419	106	415	427	UPDATE	PRODUCT	89-WRE-Q	PROD_QOH	12	11
423				CHECKPOINT					
427	106	419	431	UPDATE	CUSTOMER	10016	CUST_BALANCE	0.00	277.55
431	106	427	457	INSERT	ACCT_TRANSACTION	10007			1007,18-JAN-2004, ..
457	106	431	Null	COMMIT	**** End of Transaction				
521	155	Null	525	START	**** Start Transaction				
525	155	521	528	UPDATE	PRODUCT	2232/QWE	PROD_QOH	6	26
528	155	525	Null	COMMIT	**** End of Transaction				

* * * * * C *R*A* S* H * * * * *

PARTICIPATION ACTIVITY | 20.1.5: Recovery from transaction failure.



When the system or programmer initiates rollback, the recovery system reads the log in reverse.

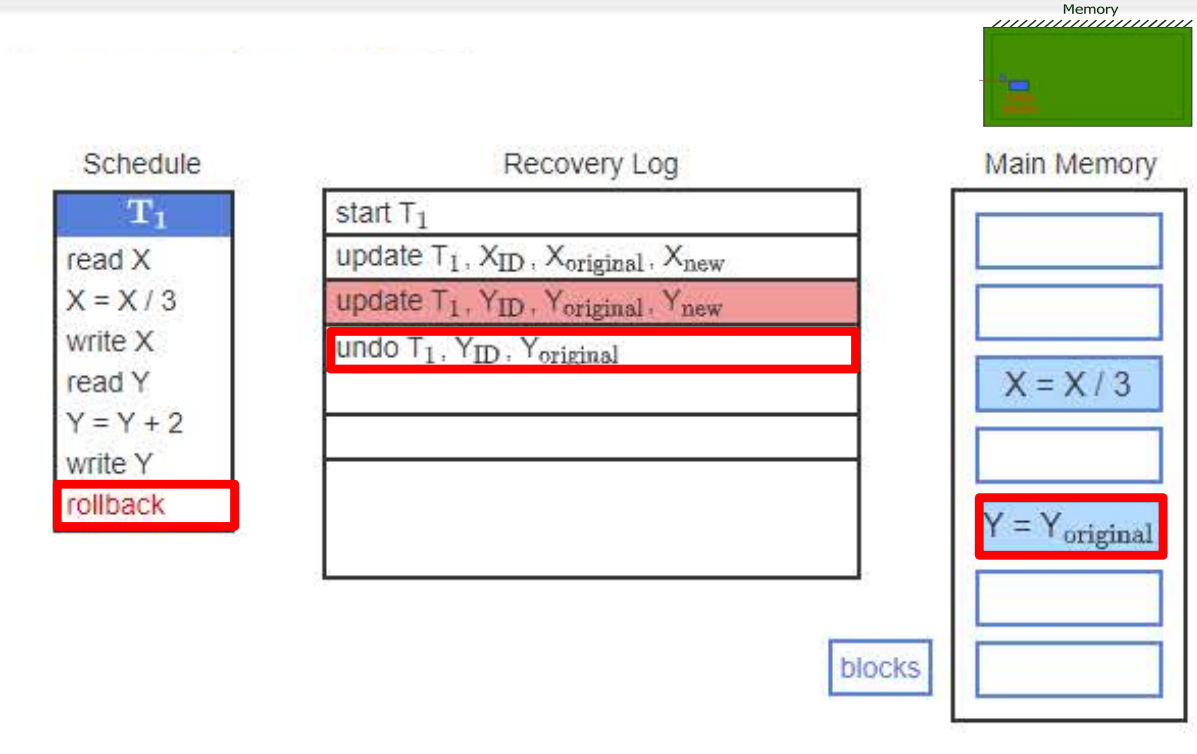


A Transaction Log for Transaction Recovery Examples

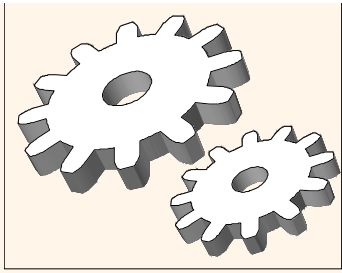
TRL ID	TRX NUM	PREV PTR	NEXT PTR	OPERATION	TABLE	ROW ID	ATTRIBUTE	BEFORE VALUE	AFTER VALUE
341	101	Null	352	START	**** Start Transaction				
352	101	341	363	UPDATE	PRODUCT	54778-2T	PROD_QOH	45	43
363	101	352	365	UPDATE	CUSTOMER	10011	CUST_BALANCE	615.73	675.62
365	101	363	Null	COMMIT	**** End of Transaction				
397	106	Null	405	START	**** Start Transaction				
405	106	397	415	INSERT	INVOICE	1009			1009,10016, ..
415	106	405	419	INSERT	LINE	1009,1			1009,1, 89-WRE-Q,1, ..
419	106	415	427	UPDATE	PRODUCT	89-WRE-Q	PROD_QOH	12	11
423				CHECKPOINT					
427	106	419	431	UPDATE	CUSTOMER	10016	CUST_BALANCE	0.00	277.55
431	106	427	457	INSERT	ACCT_TRANSACTION	10007			1007,18-JAN-2004, ..
457	106	431	Null	COMMIT	**** End of Transaction				
521	155	Null	525	START	**** Start Transaction				
525	155	521	528	UPDATE	PRODUCT	2232/QWE	PROD_QOH	6	26
528	155	525	Null	COMMIT	**** End of Transaction				

* * * * * C *R*A* S* H * * * * *

PARTICIPATION ACTIVITY 20.1.5: Recovery from transaction failure.



Y is restored to original value in main memory, and a compensation record is written to the log.

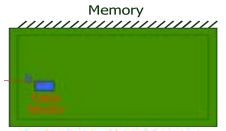


A Transaction Log for Transaction Recovery Examples

TRL ID	TRX NUM	PREV PTR	NEXT PTR	OPERATION	TABLE	ROW ID	ATTRIBUTE	BEFORE VALUE	AFTER VALUE
341	101	Null	352	START	**** Start Transaction				
352	101	341	363	UPDATE	PRODUCT	54778-2T	PROD_QOH	45	43
363	101	352	365	UPDATE	CUSTOMER	10011	CUST_BALANCE	615.73	675.62
365	101	363	Null	COMMIT	**** End of Transaction				
397	106	Null	405	START	**** Start Transaction				
405	106	397	415	INSERT	INVOICE	1009			1009,10016, ..
415	106	405	419	INSERT	LINE	1009,1			1009,1, 89-WRE-Q,1, ..
419	106	415	427	UPDATE	PRODUCT	89-WRE-Q	PROD_QOH	12	11
423				CHECKPOINT					
427	106	419	431	UPDATE	CUSTOMER	10016	CUST_BALANCE	0.00	277.55
431	106	427	457	INSERT	ACCT_TRANSACTION	10007			1007,18-JAN-2004, ..
457	106	431	Null	COMMIT	**** End of Transaction				
521	155	Null	525	START	**** Start Transaction				
525	155	521	528	UPDATE	PRODUCT	2232/QWE	PROD_QOH	6	26
528	155	525	Null	COMMIT	**** End of Transaction				

* * * * * C *R*A* S* H * * * * *

PARTICIPATION ACTIVITY 20.1.5: Recovery from transaction failure.



Schedule

T_1
read X
X = X / 3
write X
read Y
Y = Y + 2
write Y
rollback

Recovery Log

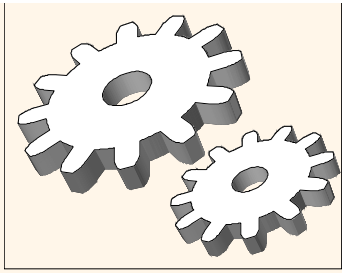
start T_1
update T_1, X_{ID}, $X_{original}$, X_{new}
update T_1, Y_{ID}, $Y_{original}$, Y_{new}
undo T_1, Y_{ID}, $Y_{original}$

Main Memory

X = X / 3
Y = $Y_{original}$

blocks

Y is restored to original value in main memory, and a compensation record is written to the log.

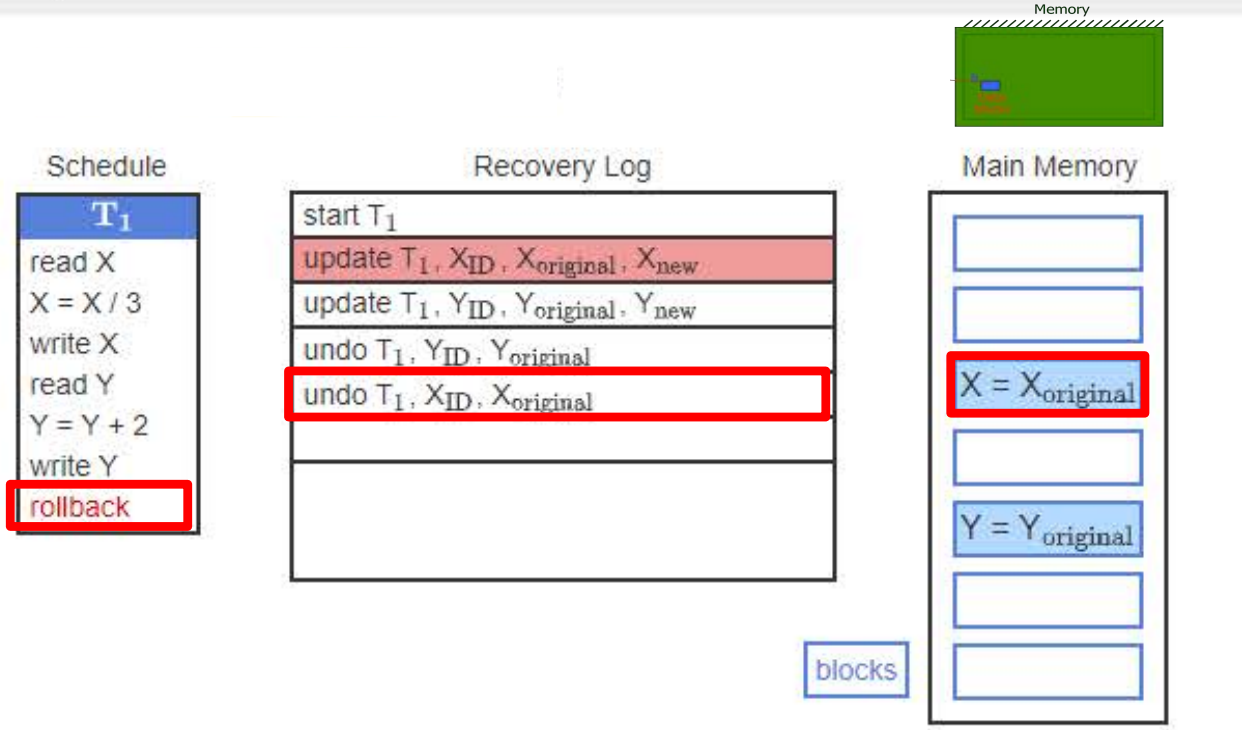


A Transaction Log for Transaction Recovery Examples

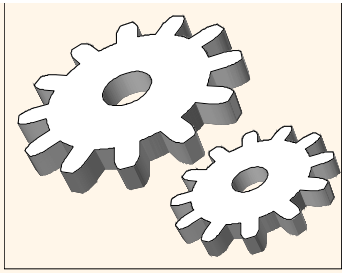
TRL ID	TRX NUM	PREV PTR	NEXT PTR	OPERATION	TABLE	ROW ID	ATTRIBUTE	BEFORE VALUE	AFTER VALUE
341	101	Null	352	START	**** Start Transaction				
352	101	341	363	UPDATE	PRODUCT	54778-2T	PROD_QOH	45	43
363	101	352	365	UPDATE	CUSTOMER	10011	CUST_BALANCE	615.73	675.62
365	101	363	Null	COMMIT	**** End of Transaction				
397	106	Null	405	START	**** Start Transaction				
405	106	397	415	INSERT	INVOICE	1009			1009,10016, ..
415	106	405	419	INSERT	LINE	1009,1			1009,1, 89-WRE-Q,1, ..
419	106	415	427	UPDATE	PRODUCT	89-WRE-Q	PROD_QOH	12	11
423				CHECKPOINT					
427	106	419	431	UPDATE	CUSTOMER	10016	CUST_BALANCE	0.00	277.55
431	106	427	457	INSERT	ACCT_TRANSACTION	10007			1007,18-JAN-2004, ..
457	106	431	Null	COMMIT	**** End of Transaction				
521	155	Null	525	START	**** Start Transaction				
525	155	521	528	UPDATE	PRODUCT	2232/QWE	PROD_QOH	6	26
528	155	525	Null	COMMIT	**** End of Transaction				

* * * * * C *R*A* S* H * * * * *

PARTICIPATION ACTIVITY 20.1.5: Recovery from transaction failure.



X is restored to original value in main memory, and a compensation record is written to the log.

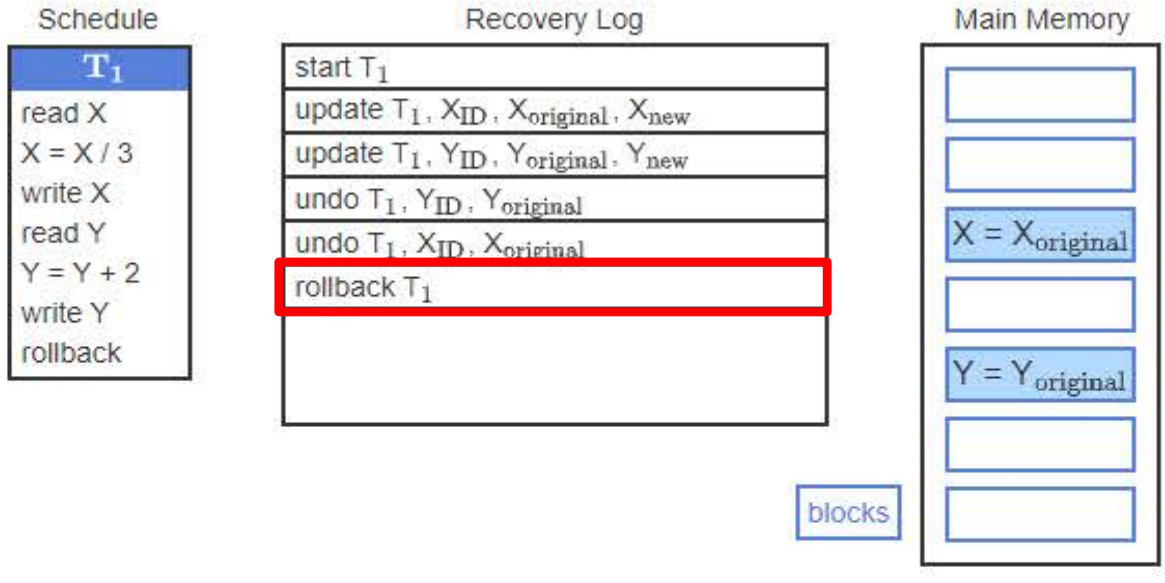


A Transaction Log for Transaction Recovery Examples

TRL ID	TRX NUM	PREV PTR	NEXT PTR	OPERATION	TABLE	ROW ID	ATTRIBUTE	BEFORE VALUE	AFTER VALUE
341	101	Null	352	START	**** Start Transaction				
352	101	341	363	UPDATE	PRODUCT	54778-2T	PROD_QOH	45	43
363	101	352	365	UPDATE	CUSTOMER	10011	CUST_BALANCE	615.73	675.62
365	101	363	Null	COMMIT	**** End of Transaction				
397	106	Null	405	START	**** Start Transaction				
405	106	397	415	INSERT	INVOICE	1009			1009,10016, ..
415	106	405	419	INSERT	LINE	1009,1			1009,1, 89-WRE-Q,1, ..
419	106	415	427	UPDATE	PRODUCT	89-WRE-Q	PROD_QOH	12	11
423				CHECKPOINT					
427	106	419	431	UPDATE	CUSTOMER	10016	CUST_BALANCE	0.00	277.55
431	106	427	457	INSERT	ACCT_TRANSACTION	10007			1007,18-JAN-2004, ..
457	106	431	Null	COMMIT	**** End of Transaction				
521	155	Null	525	START	**** Start Transaction				
525	155	521	528	UPDATE	PRODUCT	2232/QWE	PROD_QOH	6	26
528	155	525	Null	COMMIT	**** End of Transaction				

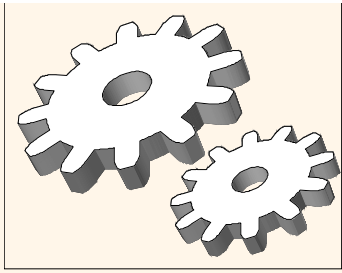
* * * * * C *R*A* S* H * * * * *

PARTICIPATION ACTIVITY 20.1.5: Recovery from transaction failure.

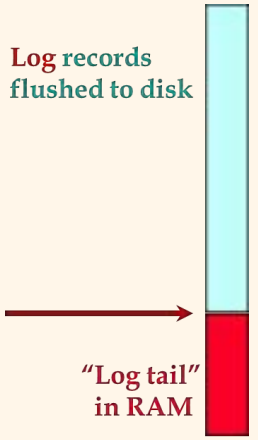


Finally, the rollback record is written to the log, and rollback is complete

Transaction Commit



- Write **COMMIT** record to **Log**.
- All **Log** records up to **Xact**'s lastLSN are **flushed**.
  - Guarantees that flushed**LSN** ≥ last**LSN**.
  - Note that **Log** flushes are sequential, synchronous writes to disk.
  - Many **Log** records per **Log Page**.
- COMMIT() returns.



- Write **END** record to **Log**.

Transaction Table

transID	status	lastLSN
T2	Abort	70
T1	Active	10

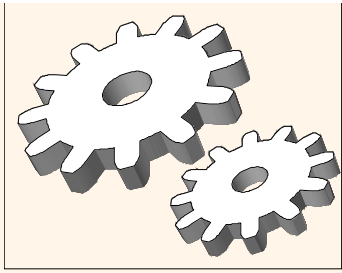
A Transaction Log

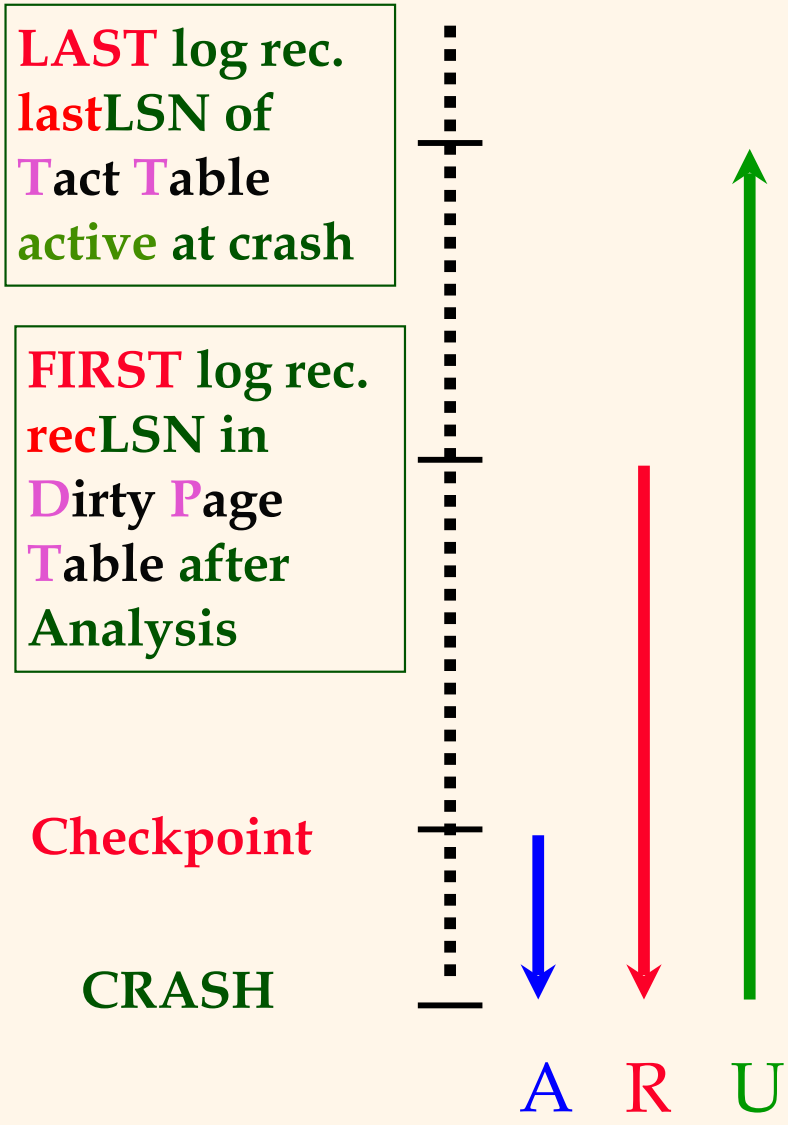
TRL_ID	TRX_NUM	PREV PTR	NEXT PTR	OPERATION	TABLE	ROW ID	ATTRIBUTE	BEFORE VALUE	AFTER VALUE
341	101	Null	352	START	****Start Transaction				
352	101	341	363	UPDATE	PRODUCT	1558-QW1	PROD_QOH	25	23
363	101	352	365	UPDATE	CUSTOMER	10011	CUST_BALANCE	525.75	615.73
365	101	363	Null	COMMIT	**** End of Transaction				

END

matically assigned by the DBMS.)

Crash Recovery: Big Picture



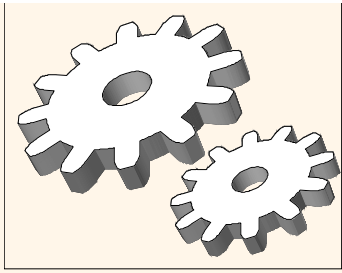


Start from a Checkpoint (found via **Master Record**).

Three Phases. Need to:

- Figure out which Xacts **COMMIT**ted since Checkpoint, which **FAILED** (**A**nalysis).
- REDO *all* actions.
  - (repeat history) (**R**).
- UNDO effects of **FAILED Xacts** (**U**).

Recovery: The Analysis Phase



Assume checkpointing here

pageID	recLSN
P500	
P600	
P505	
P700	

DIRTY PAGE TABLE

transID	lastLSN
T1000	

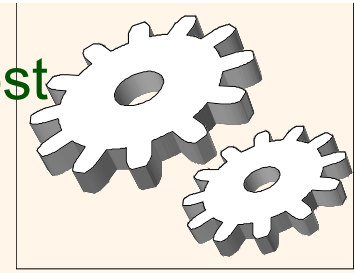
TRANSACTION TABLE

prevLSN	transID	type	pageID	length	offset	before	after
	T1000	Update	P500	3	21	ABC	DEF
	T2000	Update	P600	3	41	HIJ	KLM
	T2000	Update	P500	3	20	GDE	QRS
	T1000	Update	P505	3	21	TUV	WXY
	T2000	Commit END					
	T1000	Update	P700	3	23	ABC	XYZ

LOG

System crashes before this log entry is written to stable storage!

Recovery – DIRTY PAGE & TRANSACTION TABLES are lost



pageID recLSN

DIRTY PAGE TABLE

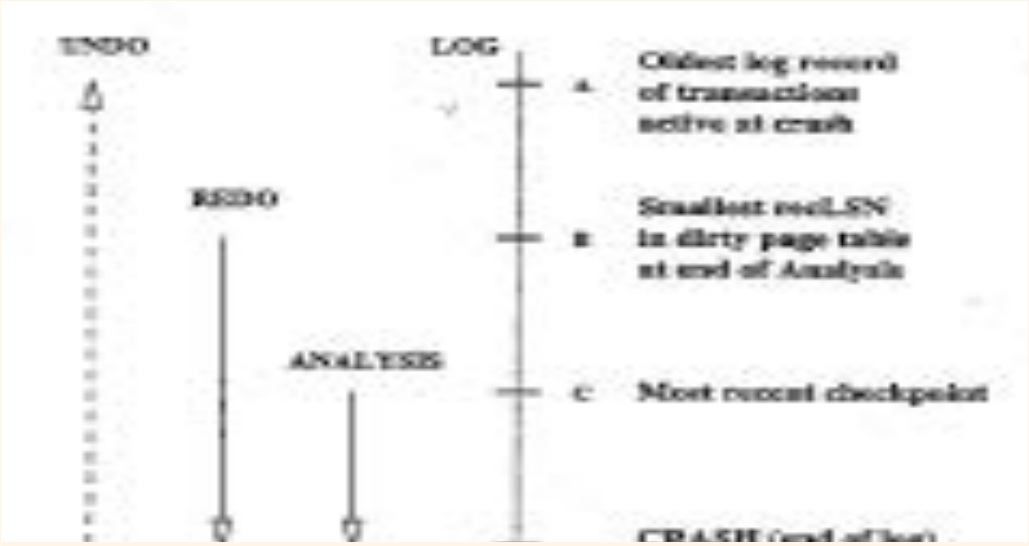
transID lastLSN

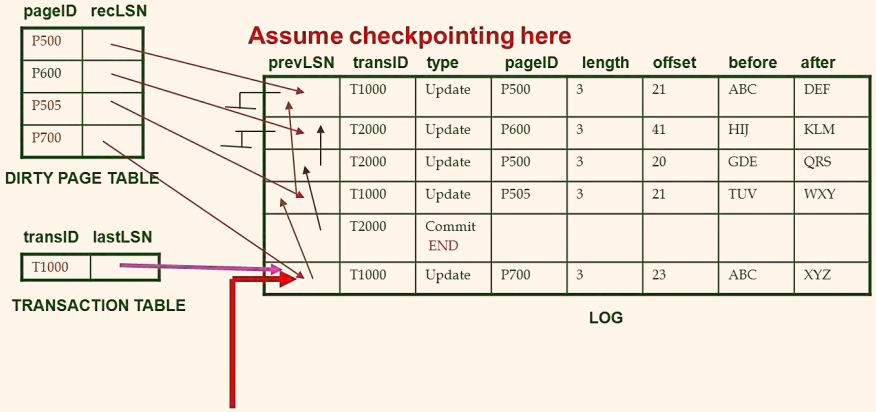
TRANSACTION TABLE

Assume checkpointing here

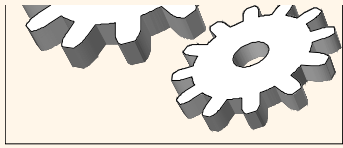
prevLSN	transID	type	pageID	length	offset	before	after
	T1000	Update	P500	3	21	ABC	DEF
	T2000	Update	P600	3	41	HIJ	KLM
	T2000	Update	P500	3	20	GDE	QRS
	T1000	Update	P505	3	21	TUV	WXY
	T2000	Commit					

LOG





Recovery - Analysis – DIRTY PAGE & TRANSACTION TABLES are lost



pageID recLSN

P500	
P600	
P505	

DIRTY PAGE TABLE

transID lastLSN

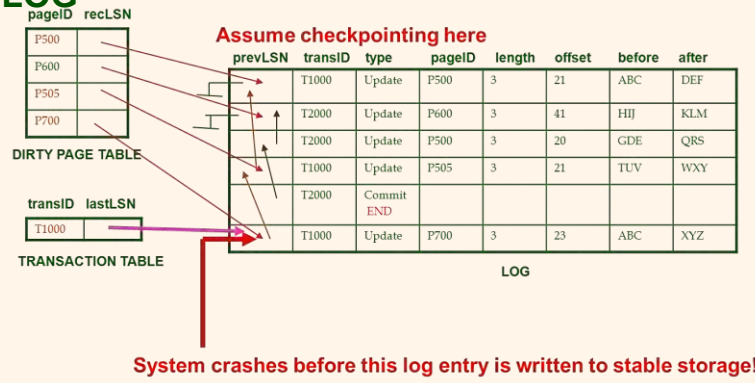
T1000	
T2000	

TRANSACTION TABLE

Assume checkpointing here

prevLSN	transID	type	pageID	length	offset	before	after
	T1000	Update	P500	3	21	ABC	DEF
	T2000	Update	P600	3	41	HIJ	KLM
	T2000	Update	P500	3	20	GDE	QRS
	T1000	Update	P505	3	21	TUV	WXY
	T2000	Commit END					

LOG

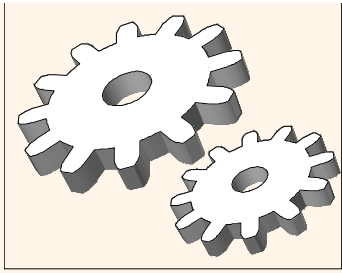


This log record will cause T2000 to be removed from the TRANSACTION TABLE!

Analysis phase recognizes that the only **active** transaction at the time of the crash is **T1000**, with last**LSN** equal to the LSN of the fourth record in this figure.

*NOTE that page **P600** is NOT dirty, yet it is included in the DIRTY PAGE TABLE!*

Recovery - Redo – D



Assume checkpointing here

pageID	recLSN
P500	
P600	
P505	

DIRTY PAGE TABLE

transID	lastLSN
T1000	

TRANSACTION TABLE

prevLSN	transID	type	pageID	length	offset	before	after
	T1000	Update	P500	3	21	ABC	DEF
	T2000	Update	P600	3	41	HIJ	KLM
	T2000	Update	P500	3	20	GDE	QRS
	T1000	Update	P505	3	21	TUV	WXY
	T2000	Commit					

LOG

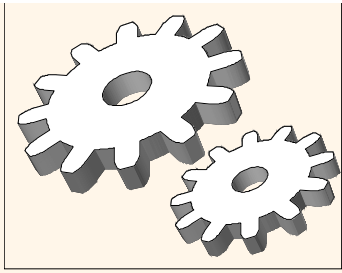
R

Smallest recSLN of all pages in the DIRTY PAGE TABLE is the **first record**.
It identifies the oldest update that may not have been written to disk prior to the **CRASH**.
REDO fetches page P500 and compares the LSN of this log record with the **page LSN** on the page and , because we assumed that this page was not written to disk before the crash, finds that the **pageLSN** is less. The update is therefore reapplied; bytes 21 though 23 are changed to 'DEF', and the **pageLSN** is set to the LSN of this **UPDATE** log record.

Starting from there, REDO scans forward until the end of the Log. REDO will check to see if this logged action needs to be redone. **The second log record**. The affected page, **P600**, is fetched and the **pageLSN** is compared to the LSN of the update log record. In this case, because we assumed that **P600 was written to disk before the crash**, they are equal, and the update does not have to be redone.

The remaining log records are processed similarly, bringing the system back to the exact state it was at the time of the **CRASH**.

Recovery - Undo – A



Assume checkpointing here

pageID	recLSN
P500	
P600	
P505	

DIRTY PAGE TABLE

transID	lastLSN
T1000	

TRANSACTION TABLE

prevLSN	transID	type	pageID	undonexLSN	offset	before	after
	T1000	Update	P500		21	ABC	DEF
	T2000	Update	P600		41	HIJ	KLM
	T2000	Update	P500		20	GDE	QRS
	T1000	Update	P505		21	TUV	WXY
	T2000	Commit					
	T1000	CLR		1st rec			
	T1000	CLR					
	T1000	End					

LOG

U

The **UNDO** phase scans backward from the end of the log. The goal of this phase is to undo the actions of all transactions active at the time of the crash, that is, to effectively **ABORT them**.

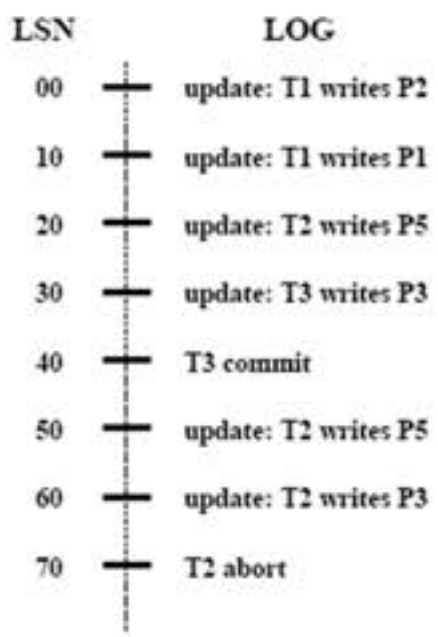
In our case, T1000 is the only transaction that needs to be aborted. From the TRANSACTION TABLE we get the LSN of its **LAST** log record, which is the fourth update log record. The update is undone, and a **CLR record** is written with **undoNextLSN** equal to the **LSN** of the first log record. The next record to be undone for transaction T1000 is the **first log record**. After this is undone, a **CLR log** record and an **END** log record for T1000 are written, and the **UNDO** phase is complete.

3.

Consider the execution shown below:

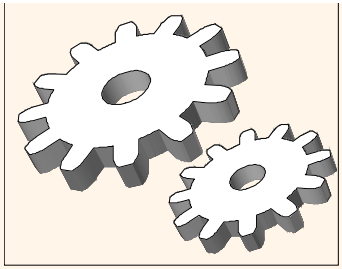
Dirty Page Table
pageID recLSN

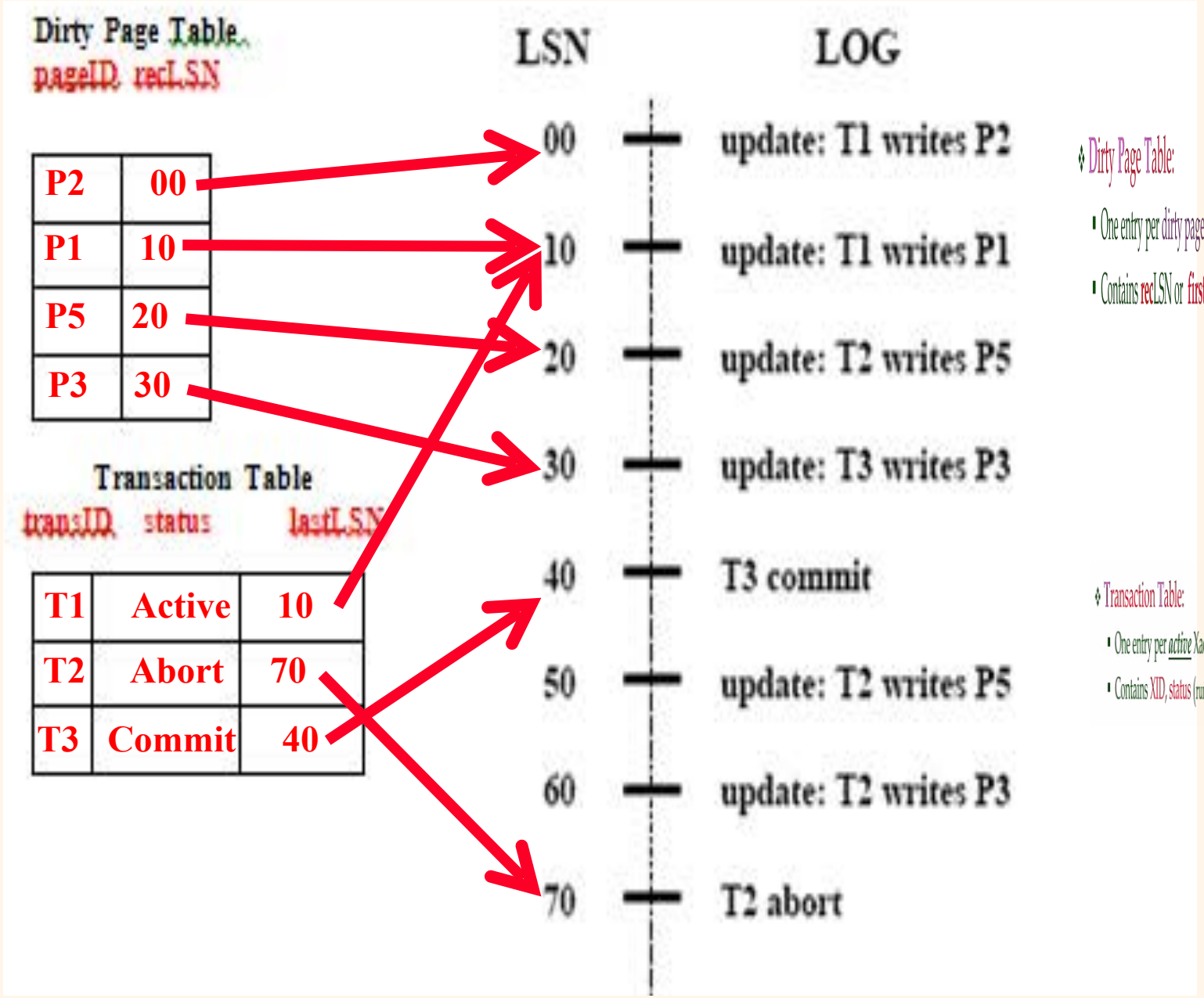
Transaction Table
transID status lastLSN



1. Extend the figure to show prevLSN and undonextLSN values.
2. Describe the actions taken to rollback transaction $T2$.
3. Show the log after $T2$ is rolled back, including all prevLSN and undonextLSN values in log records.

LSN	TRX_NUM	PREV_PTR	NEXT_PTR	OPERATION	PageID				undonextLSN
00	T1			Update	P2				
10	T1			Update	P1				
20	T2			Update	P5				
30	T3			Update	P3				
40	T3			Commit					
50	T2			Update	P5				
60	T2			Update	P3				
70	T2			Abort					

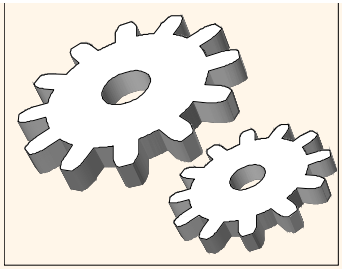




- Dirty Page Table:
  - One entry per dirty page in buffer pool.
  - Contains recLSN or firstLSN -- the LSN of the Log record which first caused the page to be dirty
- Transaction Table:
  - One entry per active Xact.
  - Contains XID, status (running/committed/aborted), and the lastLSN of the Log record for that Xact lastLSN.

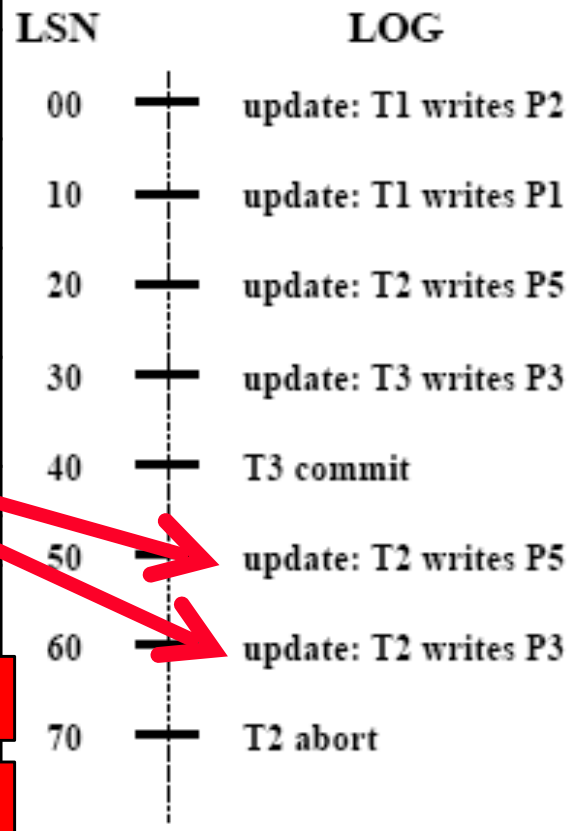
Fill in these DSs!

The Transaction Log



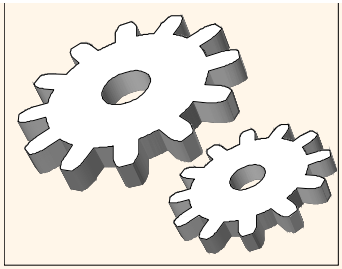
1. Extend the figure to show undonextLSN values for **T2**.

LSN	TRX_NUM	PREV_LSN	NEXT_PTR	OPERATION	PageID				undonextLSN
00	T1		10	Update	P2				
10	T1	00		Update	P1				
20	T2		50	Update	P5				
30	T3		40	Update	P3				
40	T3	30		Commit					
50	T2	20	60	Update	P5				
60	T2	50	70	Update	P3				
70	T2	60		Abort					



The Transaction Log

2. Describe the actions taken to **ROLLBACK T2**.



LSN	TRX_NUM	PREV_LSN	NEXT_PTR	OPERATION	PageID				undonextLSN
00	T1		10	Update	P2				
10	T1	00		Update	P1				
20	T2		50	Update	P5				
30	T3		40	Update	P3				
40	T3	30		Commit					
50	T2	20	60	Update	P5				
60	T2	50	70	Update	P3				
70	T2	60		Abort					

UNDO

UNDO 20

UNDO 50

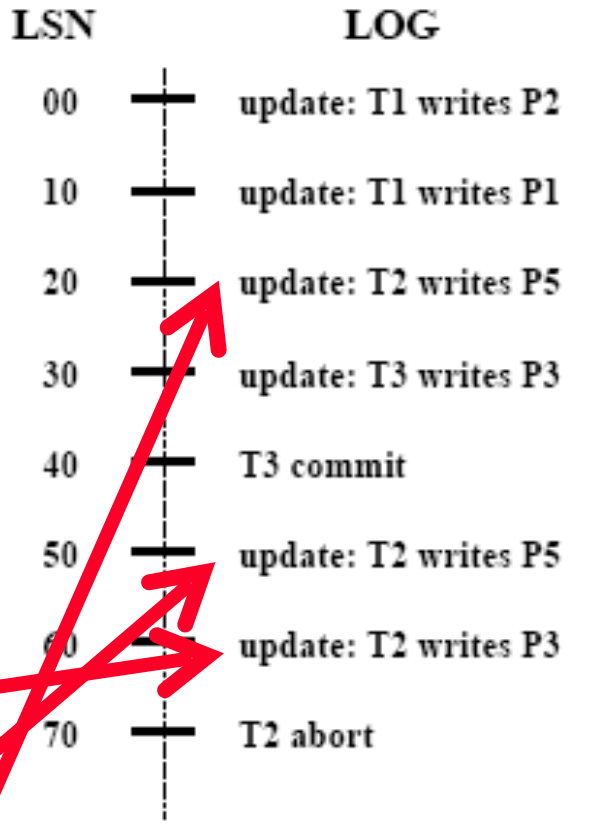
LSN	LOG
00	update: T1 writes P2
10	update: T1 writes P1
20	update: T2 writes P5
30	update: T3 writes P3
40	T3 commit
50	update: T2 writes P5
60	update: T2 writes P3
70	T2 abort

The Transaction Log

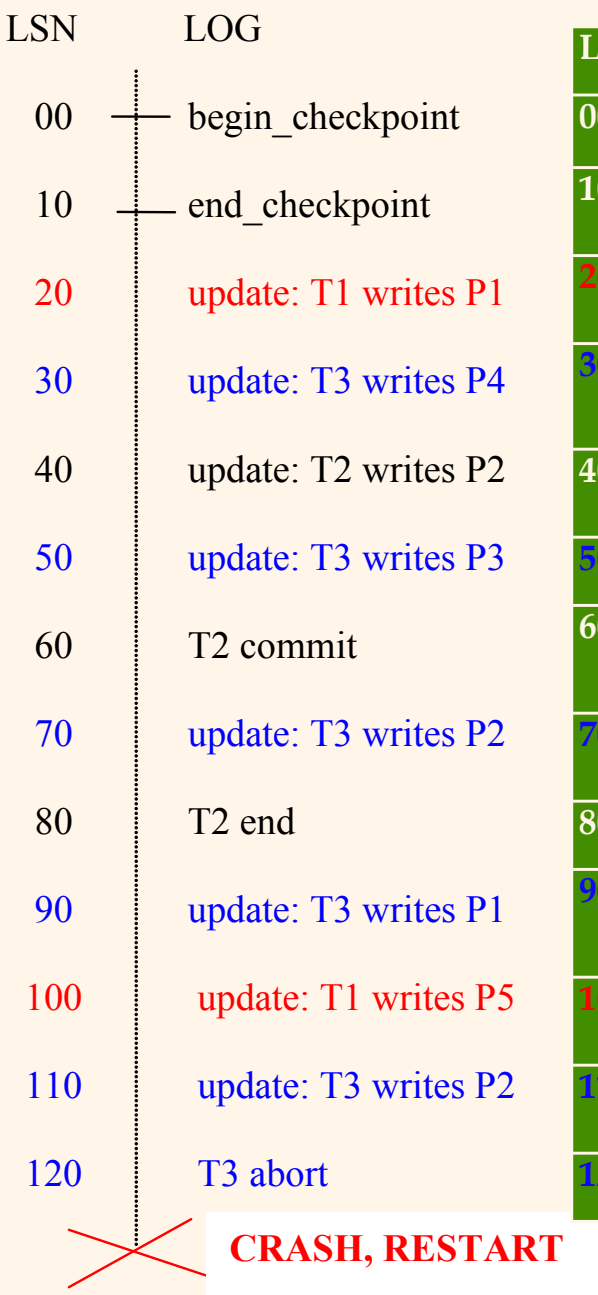
3. Show the Log after **T2 is rolledback**, including prevLSN and undonextLSN values in Log records. For OPERATION enter CLR. After that add END Log record.

LSN	TRX_NUM	PREV_LSN	NEXT_PTR	OPERATION	PageID				undonextLSN
00	T1		10	Update	P2				
10	T1	00		Update	P1				
20	T2		50	Update	P5				
30	T3		40	Update	P3				
40	T3	30		Commit					
50	T2	20	60	Update	P5				
60	T2	50	70	Update	P3				
70	T2	60	80	Abort					
80	T2	70	90	CLR	P3				60
90	T2	80	100	CLR	P5				20
100	T2	90	110	CLR	P3				NULL
110	T2	100		END					

20
50



Consider the execution shown below.

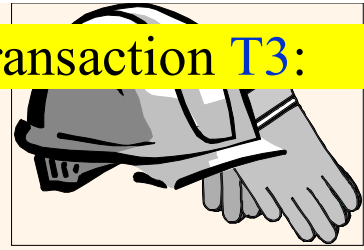


a. Rewrite with more details the log entries till LSN 120:

LSN	Type	prevLSN	undonextLSN
00	begin_cehckpoint		
10	end_checkpoint		
20	update		
30	update		
40	update		
50	update	30	30
60	commit	40	
70	update	50	50
80	end	60	
90	update	70	70
100	update	20	20
110	update	90	90
120	abort	110	

Consider the execution shown below.

b. Describe the actions taken to rollback transaction T3:



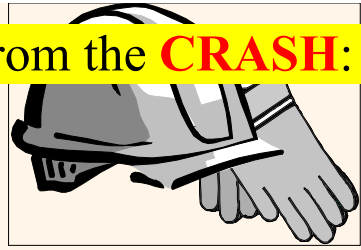
LSN	LOG
00	begin_checkpoint
10	end_checkpoint
20	update: T1 writes P1
30	update: T3 writes P4
40	update: T2 writes P2
50	update: T3 writes P3
60	T2 commit
70	update: T3 writes P2
80	T2 end
90	update: T3 writes P1
100	update: T1 writes P5
110	update: T3 writes P2
120	T3 abort

CRASH, RESTART

1. Restore P2 to the before-image stored in LSN 110
2. Restore P1 to the before-image stored in LSN 90
3. Restore P2 to the before-image stored in LSN 70
4. Restore P3 to the before-image stored in LSN 50
5. Restore P4 to the before-image stored in LSN 30

Consider the execution shown below.

c. Show the tail end of the log after the recovery from the **CRASH**:

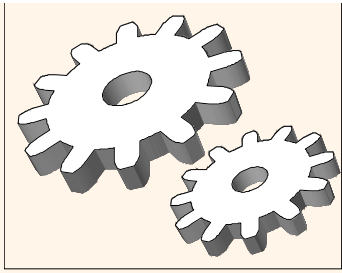


LSN	LOG
00	begin_checkpoint
10	end_checkpoint
20	update: T1 writes P1
30	update: T3 writes P4
40	update: T2 writes P2
50	update: T3 writes P3
60	T2 commit
70	update: T3 writes P2
80	T2 end
90	update: T3 writes P1
100	update: T1 writes P5
110	update: T3 writes P2
120	T3 abort

CRASH, RESTART

LSN	**next**LSN	**prev**LSN	tansID	Type	pageID	**undonext**LSN
130	140	120	T3	CLR	P2	90
140	150	130	T3	CLR	P1	70
150	160	140	T3	CLR	P2	50
160	170	150	T3	CLR	P3	30
170	180	160	T3	CLR	P4	
180		170	T3	END		
190	200	100	T1	CLR	P5	20
200	210	190	T1	CLR	P1	
210		200	T1	END		

Example of ***Recovery***



RAM

Tact Table
- lastLSN
- status

Dirty Page Table
- recLSN

flushedLSN

ToUndo

LSN	LOG
00	begin_checkpoint
05	end_checkpoint
10	update: T1 writes P5
20	update **T2** writes P3
30	T1 Abort
40	CLR: UNDO T1 LSN 10
45	T1 END
50	update: T3 writes P1
60	update: **T2** writes P5
	CRASH, RESTART

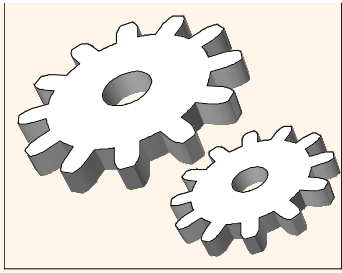
prevLSNs

How many **UPDATES** need to be **UNDOne** by T2? 2!

How many **UPDATES** need to be **UNDOne** by T3? 1!

How many **Log** Entries? 5!

Example: CRASH During Restart!



Xact Table
- lastLSN
- status

Dirty Page Table
- recLSN

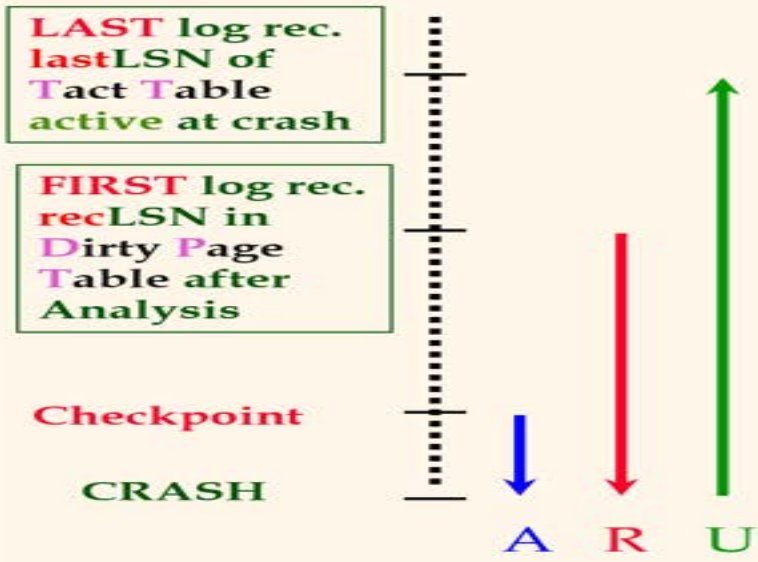
flushedLSN

ToUndo

LSN	LOG
00,05	begin_checkpoint, end_checkpoint
10	update: T1 writes P5
20	update T2 writes P3
30	T1 Abort
40,45	CLR: UNDO T1 LSN 10, T1 END
50	update: T3 writes P1
60	update: T2 writes P5
	CRASH, RESTART

undonextLSN

Crash Recovery: Big Picture



Start from a Checkpoint (found via **Master Record**).

Three Phases. Need to:

- Figure out which Xacts **COMMIT**ted since Checkpoint, which **FAILED** (Analysis).
- REDO *all* actions.
  - (repeat history) (**R**).
- UNDO effects of **FAILED** Xacts (**U**).

51

PARTICIPATION ACTIVITY | 20.1.7: Recovery from system failure.

CHALLENGE ACTIVITY | 20.2.1: Recovery.

Given the schedule below, complete the recovery log to match the schedule.

Schedule

T_1	T_2
	read Y
read X	
X = X * 6	
	Y = Y - 3
write X	

save main memory

rollback	
	write Y
	commit

Recovery Log

(A)
start T_1
update T_1, X_{ID}, $X_{original}$, X_{new}
checkpoint T_1, T_2
undo T_1, X_{ID}, $X_{original}$
rollback T_1
(B)
(C)

(A) Pick
(B) Pick
(C) Pick

1 2

TA time (Jordan)
(CA 20.2.1 Step 1 – Recovery)

TA time (Jordan)
(CA 20.2.1 Step 2 – Recovery)

COSC 3380 23578 M & W 4 to 5:30 PM LECTURE (in TEAMS from 3:50 to 5:15 PM)

01:15:26

Pop out | Chat | People 106 | Raise | View | Rooms | Apps | More | Camera | Mic | Share | Leave

JN Nizam, Jaer

Yu, Jordan T

view all

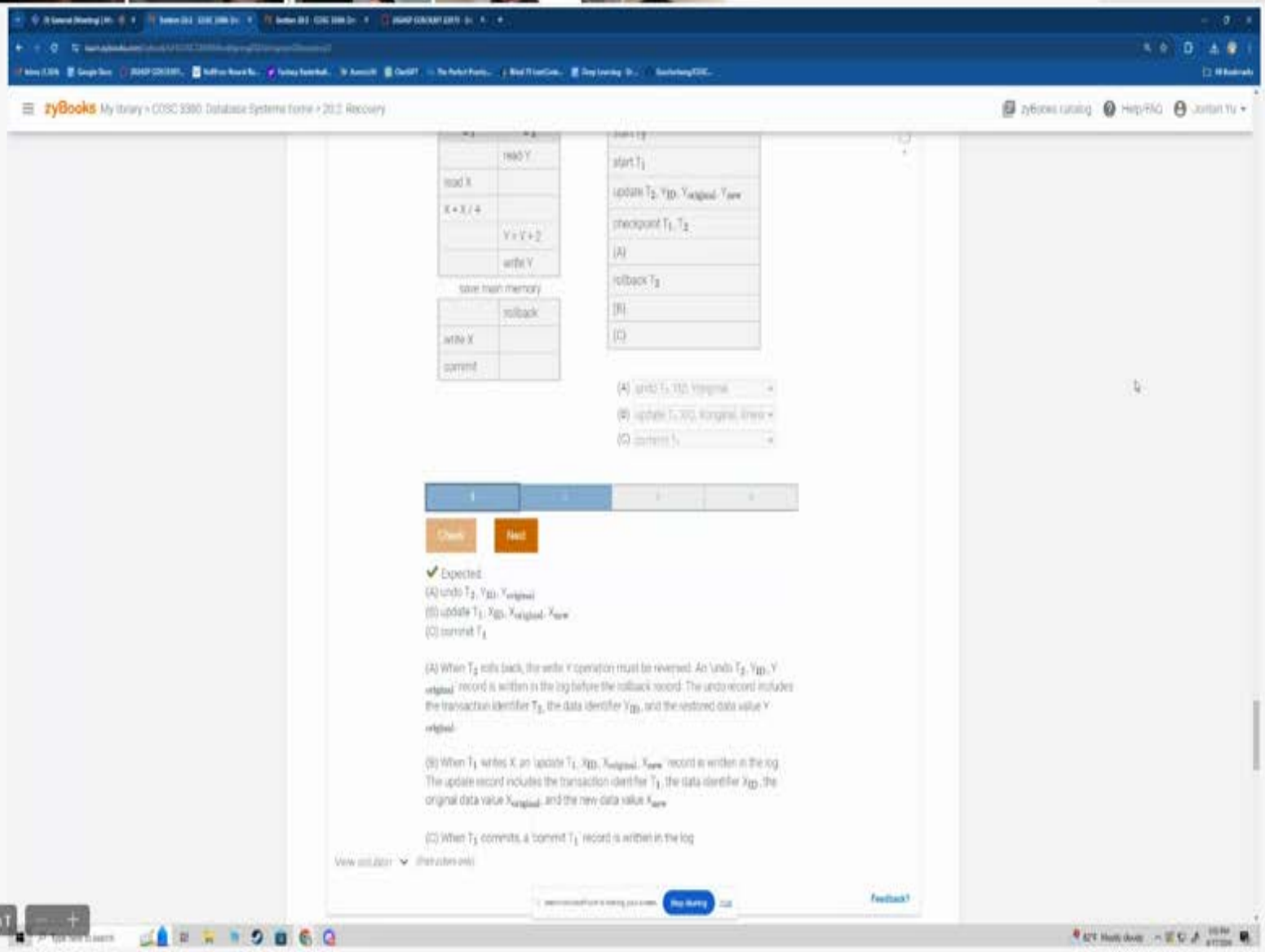
Participants

Invite someone or dial a number

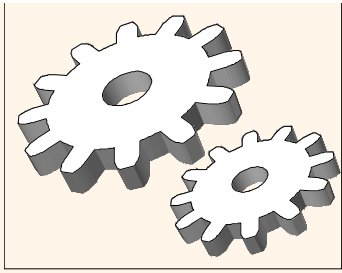
Share invite

In this meeting (106) Mute all

- Hilford, Victoria, Organizer
- RA Adhikari, Rohit
- AA Akram, Ali
- SA Altaf, Sameer
- SA Alvarez, Stephanie
- OA Anayor-Achu, Ogochukwu E
- HA Avci, Hatice Kubra
- RA Aysola, Riya
- AB Bahl, Anish
- SB Banza, Sean Paolo B
- HB Bui, Hieu
- Burger, Jake
- VC Carrillo-Zepeda, Victor E
- CW Christopher Wi... (Unverified)



Yu, Jordan T

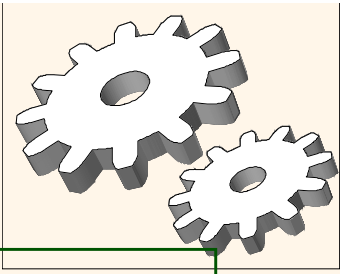


COSC 3380

Lecture 19

Security and Authorization

SQL GRANT Command



GRANT privileges ON object TO users [WITH GRANT OPTION]

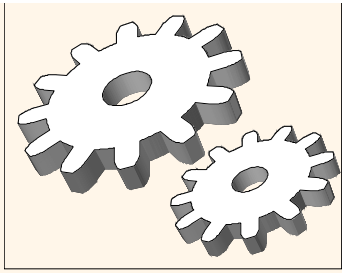
- The following privileges can be specified:
  - SELECT: Can read all columns (including those added later via ALTER TABLE command).
  - INSERT(col-name): Can insert tuples with non-null or non-default values in this column.
    - INSERT means same right with respect to all columns.
  - DELETE: Can delete tuples.
  - REFERENCES (col-name): Can define foreign keys (in other tables) that refer to this column.
- If a **user** has a privilege WITH GRANT OPTION, they can pass privilege on to other **users** (with or without passing on the GRANT OPTION).
- Only **owner** can execute CREATE, ALTER, and DROP.

SQL GRANT and REVOKE of Privileges

- **GRANT** INSERT, SELECT ON Sailors TO Horatio
  - Horatio can INSERT tuples or query **Sailors**.
- **GRANT** DELETE ON **Sailors** TO Yuppy WITH GRANT OPTION
  - Yuppy can DELETE tuples, and also can authorize others to do so.
- **GRANT** UPDATE (*rating*) ON **Sailors** TO Dustin
  - Dustin can UPDATE (only) the *rating* field of **Sailors** tuples.
- **GRANT** SELECT ON ActiveSailors TO Guppy, Yuppy
  - This does NOT allow the 'uppies to query **Sailors** directly!
- **REVOKE:** When a privilege is revoked from X, it is also **revoked** from all users who got it *solely* from X.

GRANT privileges ON object TO users [WITH GRANT OPTION]

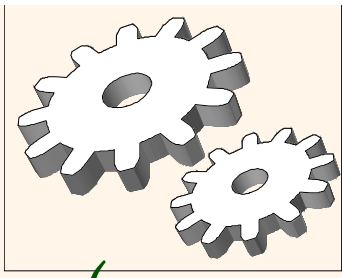
GRANT/REVOKE on Views



- If the creator of a View loses the SELECT privilege on an **underlying table**, the View is dropped!

- If the creator of a View loses a privilege held WITH GRANT OPTION on an **underlying table**, (s)he loses the privilege on the View as well; so do users who were GRANTed that privilege on the View!

Views and Security



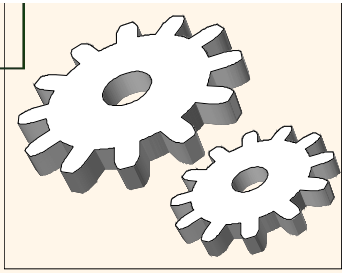
Views can be used to present necessary information (or a summary), while hiding details in **underlying Relation(s)**.

- Given ActiveSailors, but not **Sailors** or **Reserves**, we can find sailors who have a reservation, but not the *bid*'s of boats that have been reserved.

Creator of View has a privilege on the View if (s)he has the privilege on all **underlying Tables/Relations**.

Together with **GRANT/REVOKE** commands, Views are a very powerful **access control Tool**.

GRANT privileges ON object TO users [WITH GRANT OPTION]



Views and Security

4. Consider the Employee(ENAME, DEPT, SALARY) below:

ENAME	DEPT	SALARY
CHUCK NORRIS	TOYS	100000
JOHN SMITH	TOYS	300000
HANK GREEN	TOYS	200000
BOB WHITE	CS	100000
CHRIS BROWN	CS	50000

For authorization purposes you also define Views **EmployeeNames** (with **ENAME** as the only attribute) and **DeptInfo** with fields **DEPT** and **AVGSALARY**. The latter lists the average salary for each department.

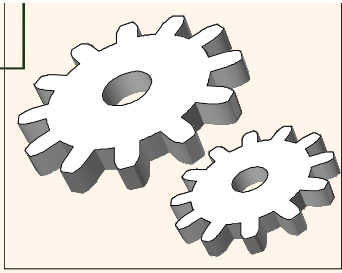
a. Show the View definition statements for **EmployeeName** and **DeptInfo**.

```
CREATE VIEW EmployeeNames (ename)
        AS SELECT E.ename
           FROM   Employees E
```

```
CREATE VIEW DeptInfo (dept, avgsalary)
        AS SELECT   DISTINCT E.dept, AVG (E.salary) AS avgsalary
           FROM     Employees E
           GROUP BY E.dept
```

GRANT privileges ON object TO users [WITH GRANT OPTION]

Views and Security



4. Consider the Employee(ENAME, DEPT, SALARY) below:

ENAME	DEPT	SALARY
CHUCK NORRIS	TOYS	100000
JOHN SMITH	TOYS	300000
HANK GREEN	TOYS	200000
BOB WHITE	CS	100000
CHRIS BROWN	CS	50000

```
CREATE VIEW EmployeeNames (ename)
       AS SELECT E.ename
          FROM   Employees E
```

EmployeeName

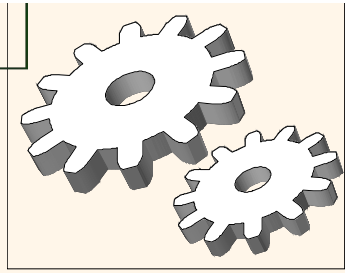
ENAME
CHUCK NORRIS
JOHN SMITH
HANK GREEN
BOB WHITE
CHRIS BROWN

```
CREATE VIEW DeptInfo (dept, avgsalary)
       AS SELECT   DISTINCT E.dept, AVG (E.salary) AS avgsalary
          FROM     Employees E
          GROUP BY E.dept
```

DeptInfo

DEPT	AVGSALARY
CS	75000
TOYS	200000

b. Show contents of Views **EmployeeName** and **DeptInfo**.



Views and Security

Employee

ENAME	DEPT	SALARY
CHUCK NORRIS	TOYS	100000
JOHN SMITH	TOYS	300000
HANK GREEN	TOYS	200000
BOB WHITE	CS	100000
CHRIS BROWN	CS	50000

EmployeeName

ENAME
CHUCK NORRIS
JOHN SMITH
HANK GREEN
BOB WHITE
CHRIS BROWN

DeptInfo

DEPT	AVGSALARY
TOYS	200000
CS	75000

CANNOT create accounts ;

c. What **privileges** should be granted to a user **who needs to know only** average department salaries for the **Toy** and **CS** departments?

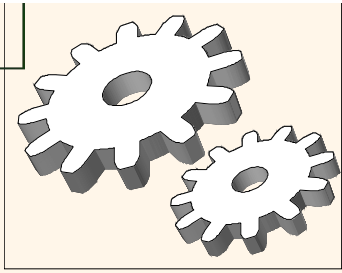
GRANT SELECT on **DeptInfo** **to public;**

```
SQL> GRANT SELECT on DeptInfo to public;
Grant succeeded.
SQL>
```

Note that it is impossible to allow the user to access only the average salaries of **Toy** and **CS** departments. Only another View will allow it.

GRANT privileges ON object TO users [WITH GRANT OPTION]

Views and Security



Employee

ENAME	DEPT	SALARY
CHUCK NORRIS	TOYS	100000
JOHN SMITH	TOYS	300000
HANK GREEN	TOYS	200000
BOB WHITE	CS	100000
CHRIS BROWN	CS	50000

DeptInfo

DEPT	AVGSALARY
TOYS	200000
CS	75000

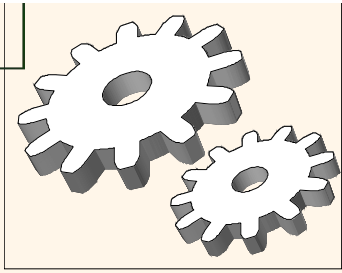
EmployeeName

ENAME
CHUCK NORRIS
JOHN SMITH
HANK GREEN
BOB WHITE
CHRIS BROWN

d. You are to **authorize** your secretary **to fire people** (you will probably tell him whom to fire, but you want to be able to delegate this task), CANNOT create accounts ;

GRANT DELETE on Employee to public;

```
SQL> GRANT DELETE on Employee to public;
Grant succeeded.
SQL>
```



DeptInfo

DEPT	AVGSALARY
TOYS	200000
CS	75000

e. Assume existence of user Assistant Manager **assistant**. What privilege should you grant to **assistant** who needs to know only average department salaries? **Express it in proper SQL**.

GRANT **SELECT** ON **DeptInfo** TO **assistant**

CANNOT create accounts ;

Recovery Practice Questions

Refer to the sequence below. What is the isolation level of transaction E?

```
session begins
SET GLOBAL TRANSACTION
ISOLATION LEVEL SERIALIZABLE;
session ends


session begins
SET SESSION TRANSACTION
ISOLATION LEVEL REPEATABLE READ;
transaction A
transaction B
SET TRANSACTION
ISOLATION LEVEL READ UNCOMMITTED;
transaction C
SET TRANSACTION
ISOLATION LEVEL READ COMMITTED;
transaction D
transaction E
session ends
```

Isolation levels

Relational databases allow database administrators and application programmers to specify strict or relaxed levels of isolation for each transaction. The SQL standard defines four isolation levels:

1. ***SERIALIZABLE*** transactions run in a serializable schedule with concurrent transactions. Isolation is guaranteed.
2. ***REPEATABLE READ*** transactions read only committed data. After the transaction reads data, other transactions **cannot** update the data. REPEATABLE READ prevents most types of isolation violations but allows phantom reads.
3. ***READ COMMITTED*** transactions read only committed data. After the transaction reads data, other transactions **can** update the data. READ COMMITTED allows nonrepeatable and phantom reads.
4. ***READ UNCOMMITTED*** transactions read uncommitted data. READ UNCOMMITTED processes concurrent transactions efficiently but allows a broad range of isolation violations, including dirty, nonrepeatable, and phantom reads.

a. Serializable

b. Repeatable read

c. Read committed

d. Read uncommitted

A transaction ____ indicates the starting and ending statement of a database transaction.

a. limit

b. boundary

c. coverage

d. scope

____ allows all executed database instructions to be rolled back and to be restored in a prior transaction state.

a. SET TRANSACTION

b. COMMIT

c. SAVEPOINT

d. RELEASE SAVEPOINT

A document that sequentially writes all the database operations is known as a/an ____.

a. recovery log

b. system log

c. application log

d. task log

A recovery system should manage which three failure scenarios?

a. Transaction failure, Memory failure, and Storage media failure

b. Transaction failure, System failure, and Storage media failure

c. Transaction failure, System failure, and Application failure

d. Transaction failure, Program failure, and Storage media failure

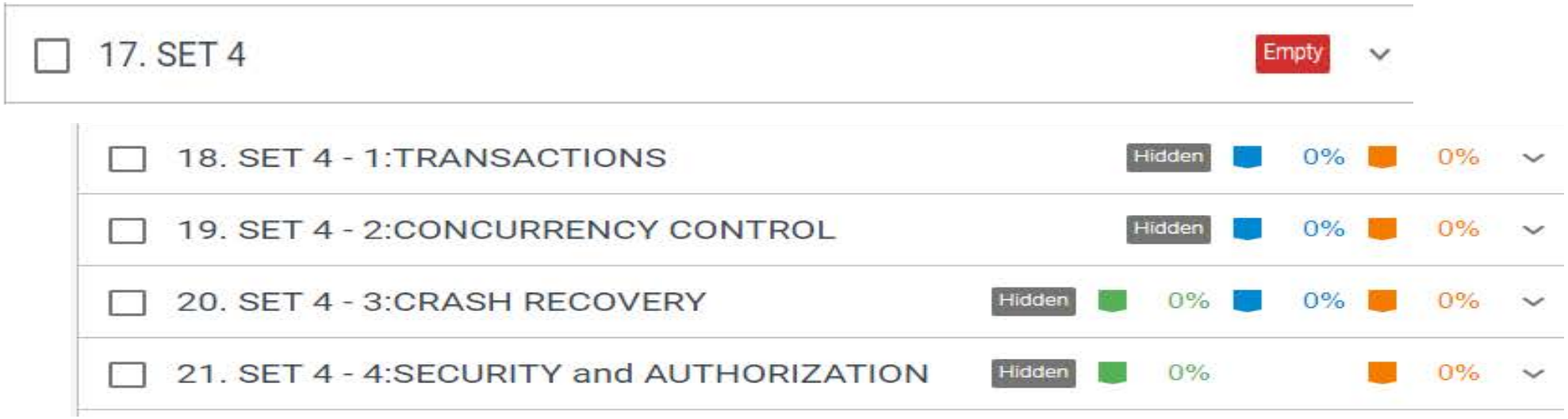
At 5:00 PM .

04.17.2024 (25 – We)		ZyBook SET 4 - 3	Set 4 LECTURE 18 CRASH RECOVERY LECTURE 19 SECURITY and AUTHORIZATION

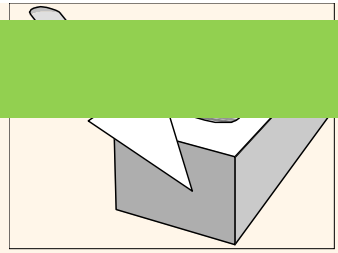
VH work on
SET 4 – 3, 4: Recovery & Security

Next

04.22.2024 (26 – Mo)		**EXAM 4 Practice** **(PART of 20 points)**



From 5:05 to 5:15 PM – 5 minutes.



04.17.2024 (25 – We)	ZyBook SET 4 - 3	Set 4 LECTURE 18 CRASH RECOVERY LECTURE 19 SECURITY and AUTHORIZATION

▾ CLASS PARTICIPATION 20 points — 20% of Total

SECURITY

Class 25 END PARTICIPATION
Not available until Apr 17 at 5:05pm | Due Apr 17 at 5:15pm

VH, publish

This is a synchronous online class.

Attendance is required.

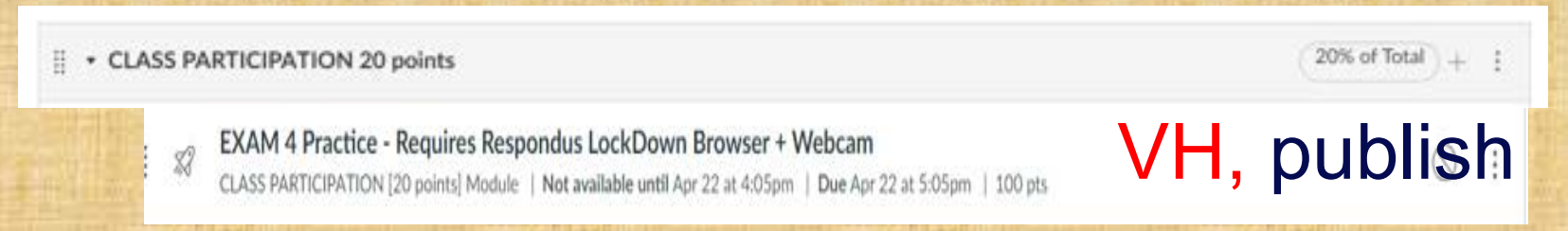
Recording or distribution of class materials is prohibited.

1. At the beginning of selected classes there is an **assessment in the first 10 minutes**. (**beige** BOX in the Detailed Syllabus)
2. At the end of selected classes there is an **assessment in the last 10 minutes**. (**blue** BOX in the Detailed Syllabus)
3. ZyBook sections will be downloaded and used for 30% of Total Score on the dates specified in the Detailed Syllabus.
4. EXAMS are in CANVAS. **No late EXAMS.**
5. I have to be **present in TEAMS in order to take any graded assignment assigned during that class.**

At 5:15 PM.

End Class 25

VH, Download Attendance Report
Rename it:
4.17.2024 Attendance Report FINAL TEAMS



VH, upload Class 25 to CANVAS.