OBJECTIVES

- Define COMMIT and ROLLBACK
- To understand the need for concurrency control and backup/recovery
- To learn about typical problems that can occur when multiple users process a database concurrently
- To understand the use of locking and the problem of deadlock
- To learn the difference between optimistic and pessimistic locking

More OBJECTIVES

- To know the meaning of an ACID transaction
- To learn the four 1992 ANSI standard isolation levels
- To know the difference between recovery via reprocessing and recovery via rollback/rollforward
- To understand the nature of the tasks required for recovery using rollback/rollforward

DBMS Transaction Logging

Start with BACKUP:

The DBAs' "bottom line" – never, ever lose data

So we run backups. How often?

- It depends on the size of the database, the number of users, the frequency of updates, and how critical the database is to the organization.

Some systems are SO CRITICAL that we keep redundant copies up-to-date at all times. Lose one, use the other.

DBMS Transaction Logging

Redundant copies of a database:

Relies on technology called REPLICATION –

a topic for a later date…

DBMS Transaction Logging

Backups and Replication both rely on Transaction Logs

Every activity against the database that changes the database is written to a log file. Called “Binary Logs” or “BinLogs” in MySQL.

Log files reside in memory and are written to disk periodically. (In MySQL, this is called a “flush”.)

Transactions are written to the log file first, then written to the database.   (Write to log is sequential and faster.)

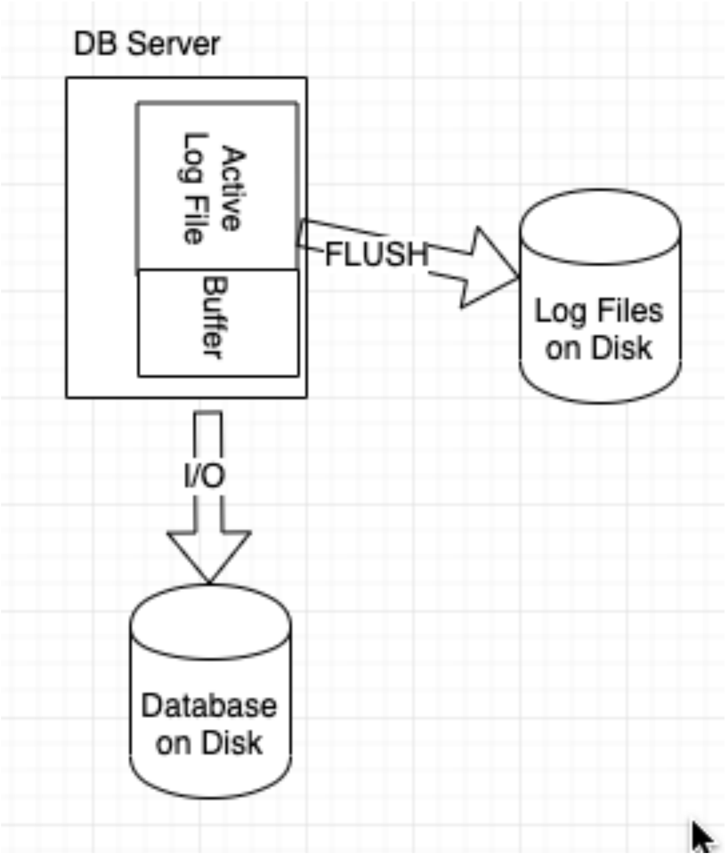
DBMS Transaction Logging

- **Logs in memory must be written to disk**
- **A log flush in MySQL**
- **(“redo logs” ➔ “archive logs” in Oracle.)**

- **Log files can be automatically written to disk**
  - On a timer, every nn minutes
  - On a size limit, whenever the log file grows to nn MB or GB
- **Closes out the old log file and opens up a new one**
  - Increments the log number in the filename

DBMS Transaction Logging

The log file:



DBMS Transaction Logging

Types of backups – usually we combine all of these

1. **Stop all database traffic and back it up quickly**
   - "Cold" backup ("full" or "incremental")
   - MySQLDUMP versus a file-level backup (OS level)
   - Faster
   - Requires downtime
2. **Take the backup while database traffic is in-progress**
   - "Warm" backup ("full" or "incremental")
   - Takes longer
   - Less inconvenience to user community

DBMS Transaction Logging

Types of backups

3. Keep a duplicate up-to-date at all times

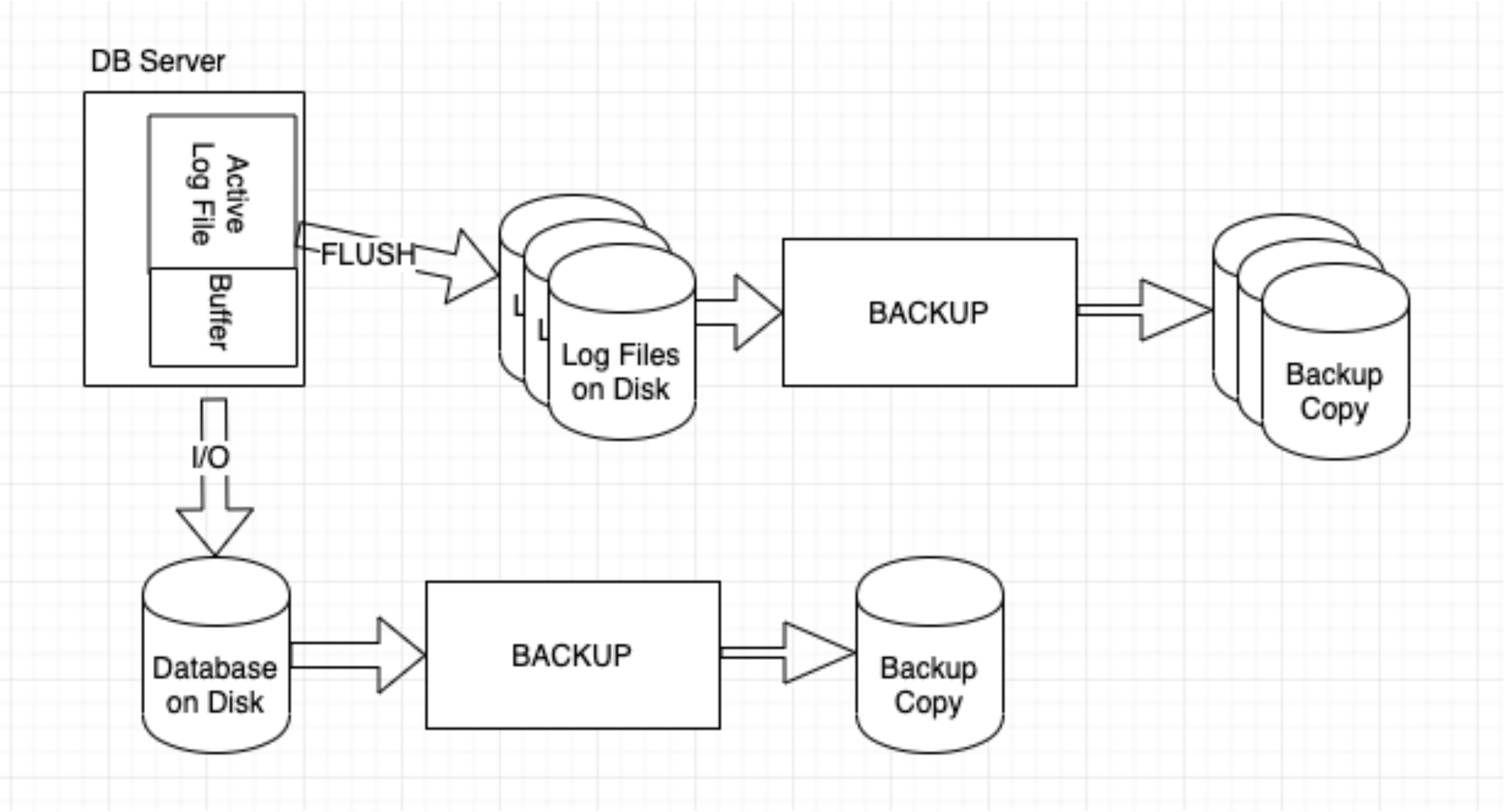
- “Hot standby” via replication from logs

4. Full versus Incremental

- The backup software copies the entire database (“full”) versus copying only rows changed since the last “full” (“incremental”)

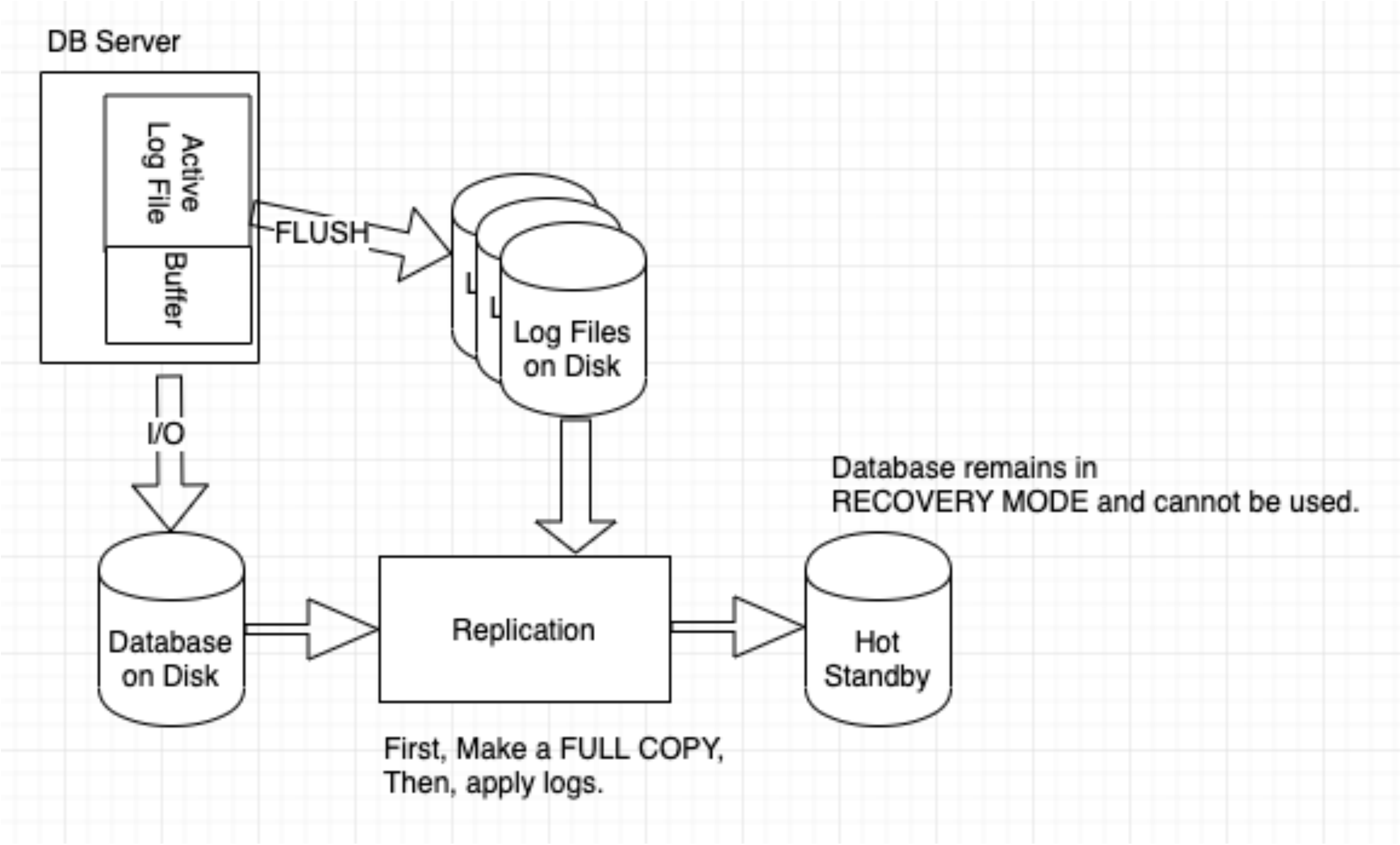
DBMS Transaction Logging

Standard Backup:



DBMS Transaction Logging

Hot Standby (via Replication):

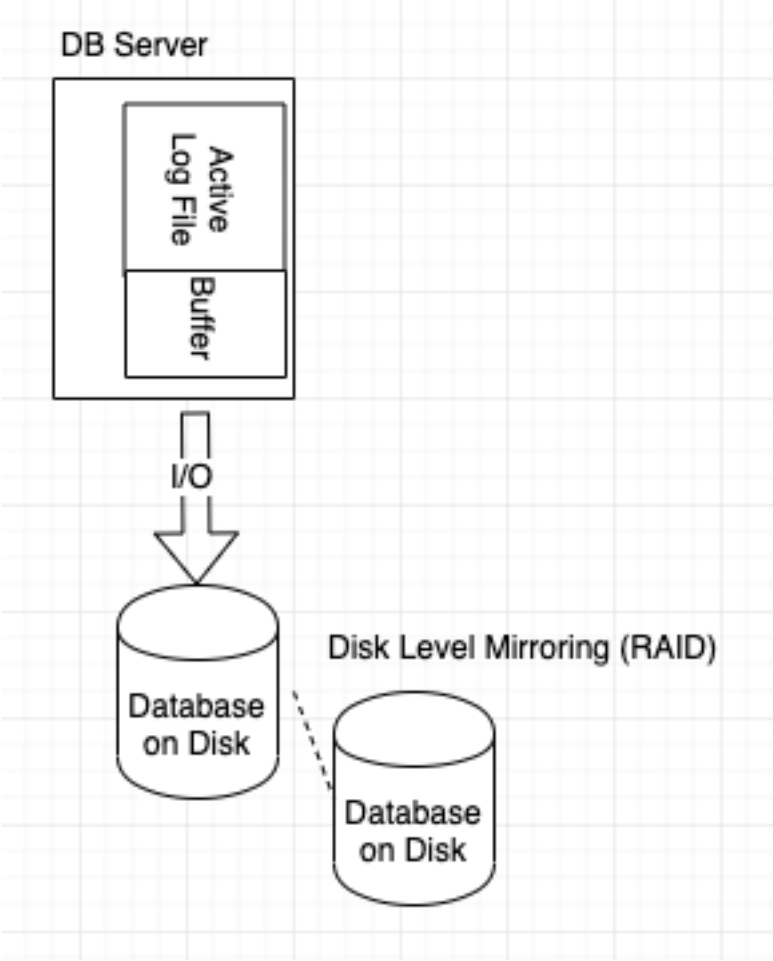


Database remains in RECOVERY MODE and cannot be used.

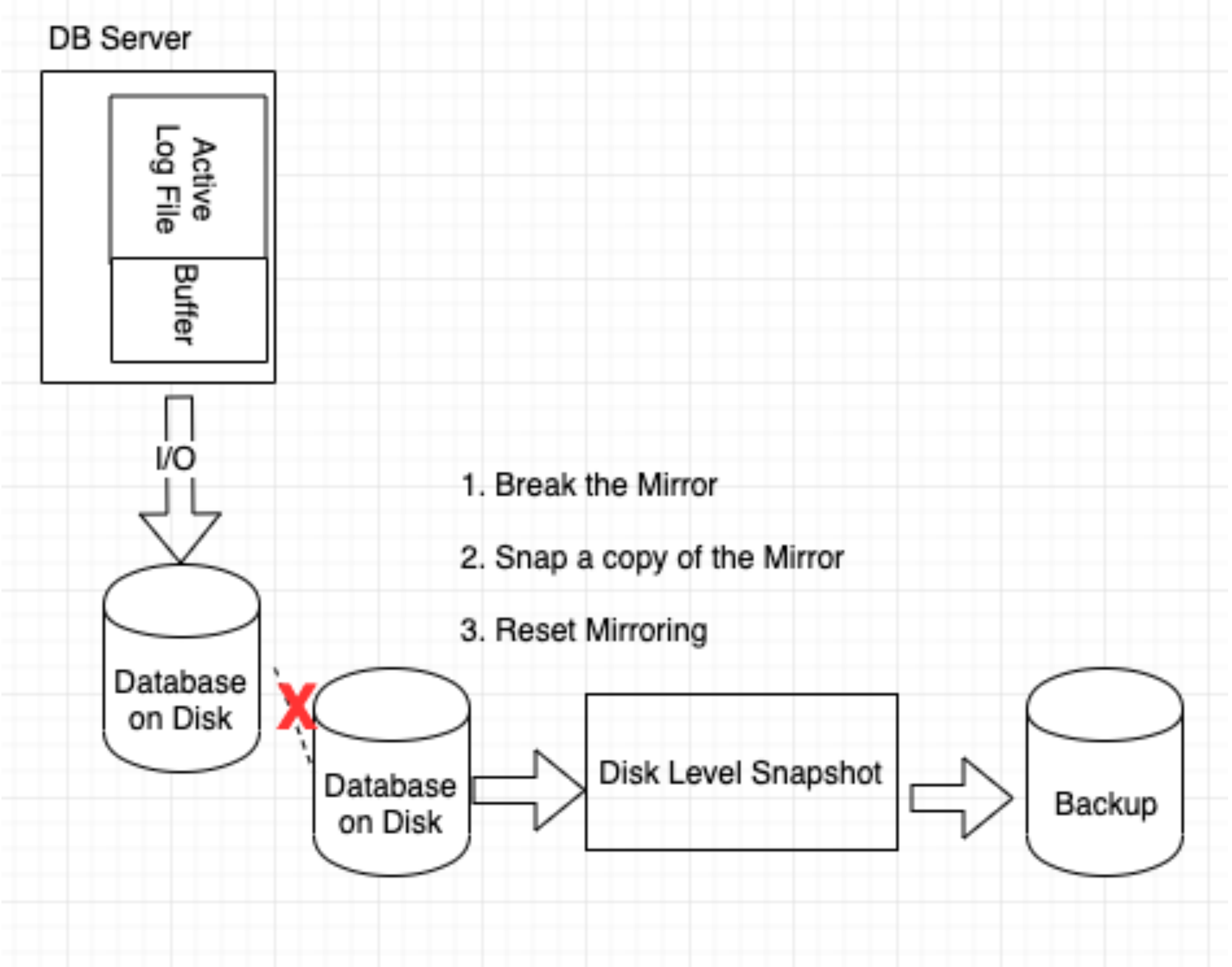
First, Make a FULL COPY, Then, apply logs.

DBMS Transaction Logging

Disk-Level Backup:



Disk-Level Backup:



1. Break the Mirror

2. Snap a copy of the Mirror

3. Reset Mirroring

DBMS Transaction Logging

Logs can be used to UNDO a transaction or REDO a transaction.

Logs capture an image of the row BEFORE and AFTER it was changed.

Logs capture timestamps for every activity.

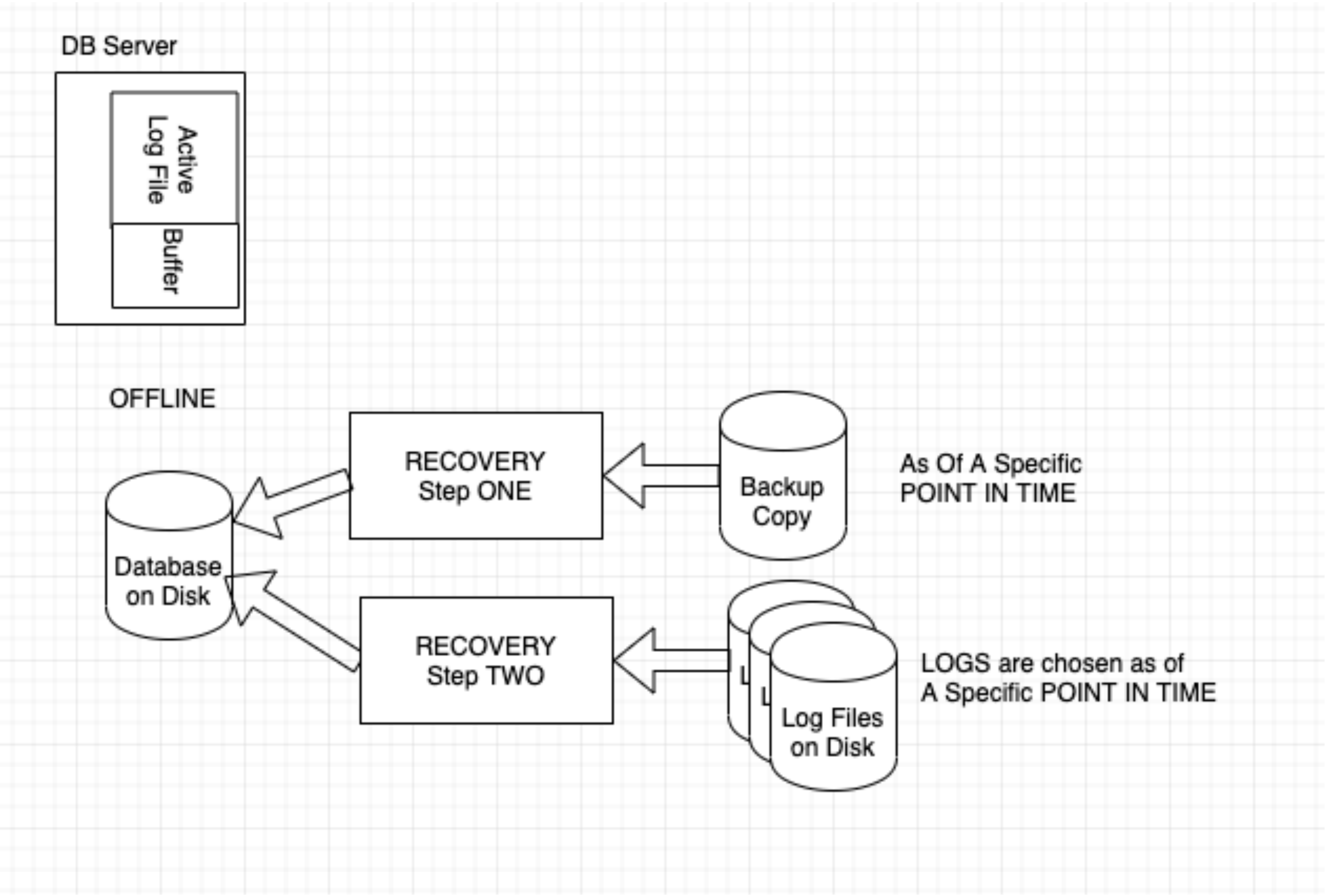
DBMS Transaction Logging

Recovering a database after a crash or corruption:

1. **Identify the most recent full backup**
2. **Identify the latest transaction log file**
3. **Identify the point-in-time of the failure**
4. **Restore the most recent full backup**
5. **Apply transaction logs up to the last commit before the failure occurred**

DBMS Transaction Logging

Standard Recovery:

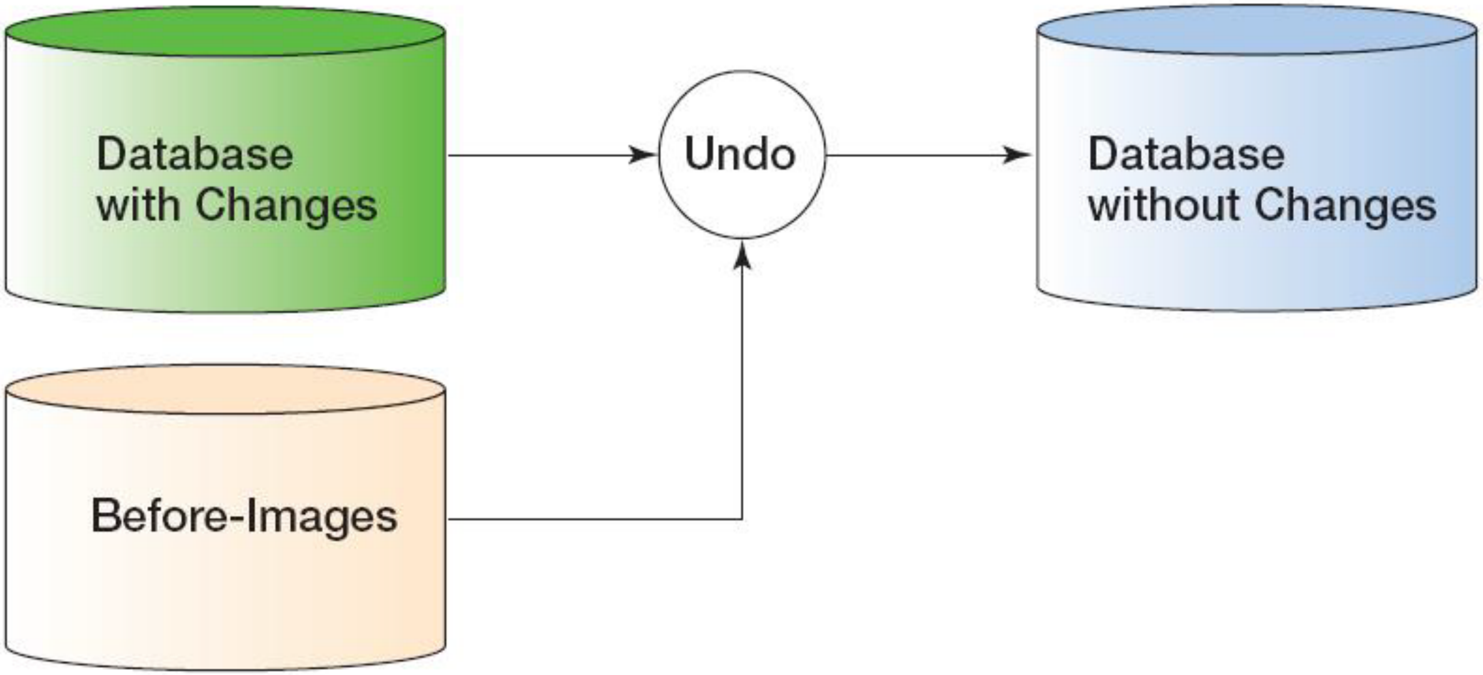


As Of A Specific
POINT IN TIME

LOGS are chosen as of
A Specific POINT IN TIME

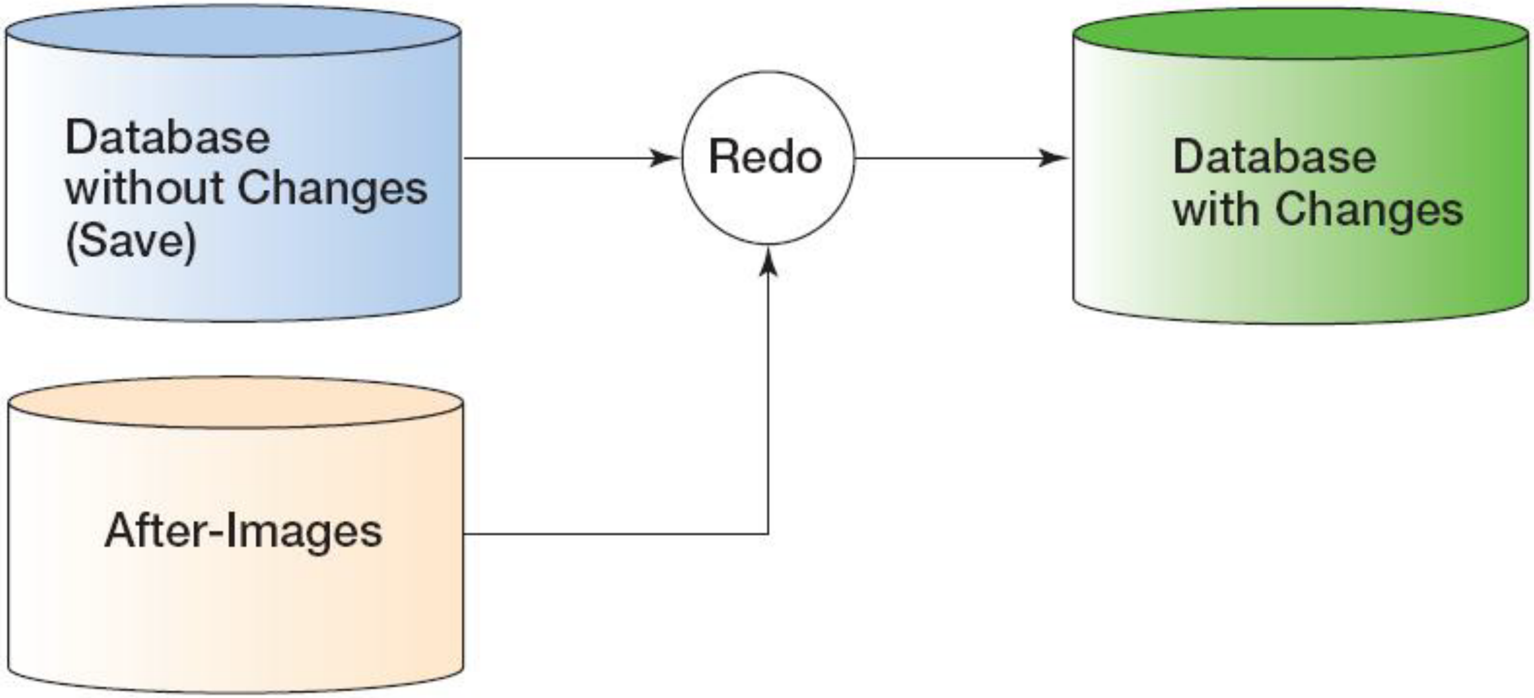
DBMS Transaction Logging

Before-image: a copy of every database record (or page) before it was changed

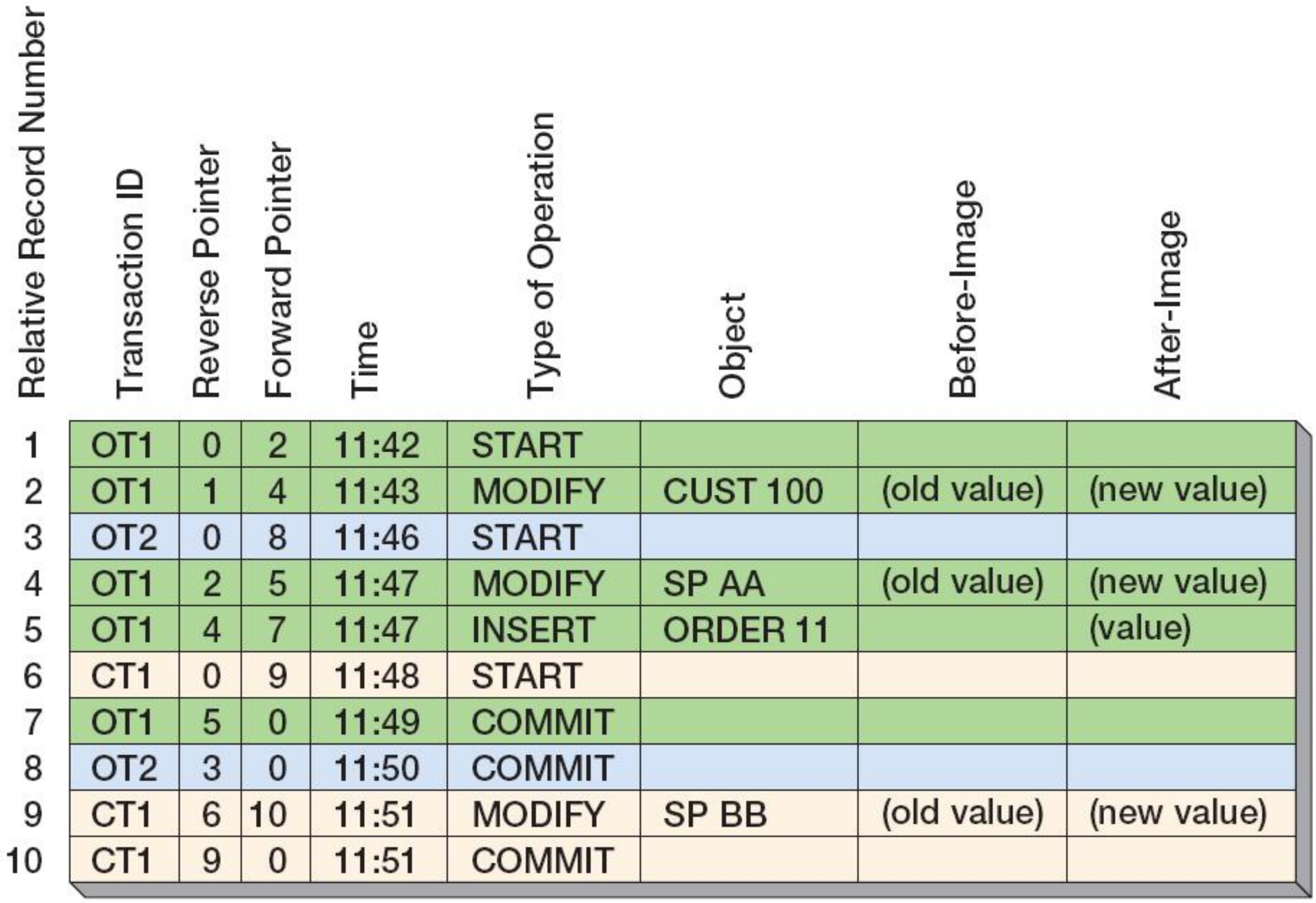


DBMS Transaction Logging

After-image: a copy of every database record (or page) after it was changed



DBMS Transaction Logging



Relative Record Number	Transaction ID	Reverse Pointer	Forward Pointer	Time	Type of Operation	Object	Before-Image	After-Image
1	OT1	0	2	11:42	START			
2	OT1	1	4	11:43	MODIFY	CUST 100	(old value)	(new value)
3	OT2	0	8	11:46	START			
4	OT1	2	5	11:47	MODIFY	SP AA	(old value)	(new value)
5	OT1	4	7	11:47	INSERT	ORDER 11		(value)
6	CT1	0	9	11:48	START			
7	OT1	5	0	11:49	COMMIT			
8	OT2	3	0	11:50	COMMIT			
9	CT1	6	10	11:51	MODIFY	SP BB	(old value)	(new value)
10	CT1	9	0	11:51	COMMIT			

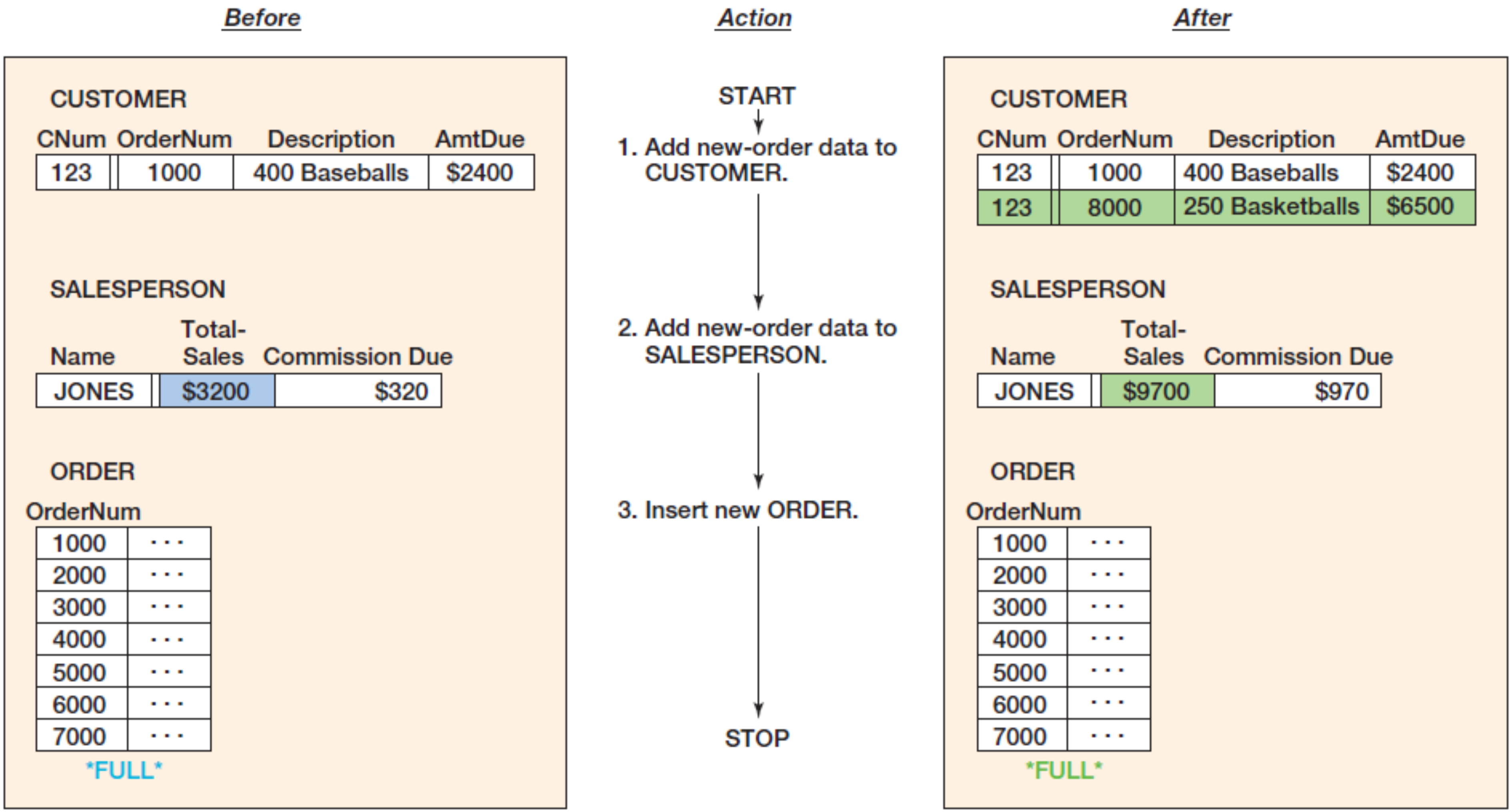
Concurrency Control

- **Concurrency control ensures that one user’s work does not inappropriately influence another user’s work.**
  - No single concurrency control technique is ideal for all circumstances.
  - Trade-offs need to be made between level of protection and throughput.

Transaction Control

- **A transaction, or logical unit of work (LUW), is a series of actions taken against the database that occurs as an atomic unit:**
  - Either all actions in a transaction occur or none of them do.

What can happen without transaction control



Before

CUSTOMER

CNum	OrderNum	Description	AmtDue
123	1000	400 Baseballs	$2400

SALESPERSON

Name	Total-Sales	Commission Due
JONES	$3200	$320

ORDER

OrderNum	
1000	· · ·
2000	· · ·
3000	· · ·
4000	· · ·
5000	· · ·
6000	· · ·
7000	· · ·

FULL

Action

START

1. Add new-order data to CUSTOMER.
2. Add new-order data to SALESPERSON.
3. Insert new ORDER.

STOP

After

CUSTOMER

CNum	OrderNum	Description	AmtDue
123	1000	400 Baseballs	$2400
123	8000	250 Basketballs	$6500

SALESPERSON

Name	Total-Sales	Commission Due
JONES	$9700	$970

ORDER

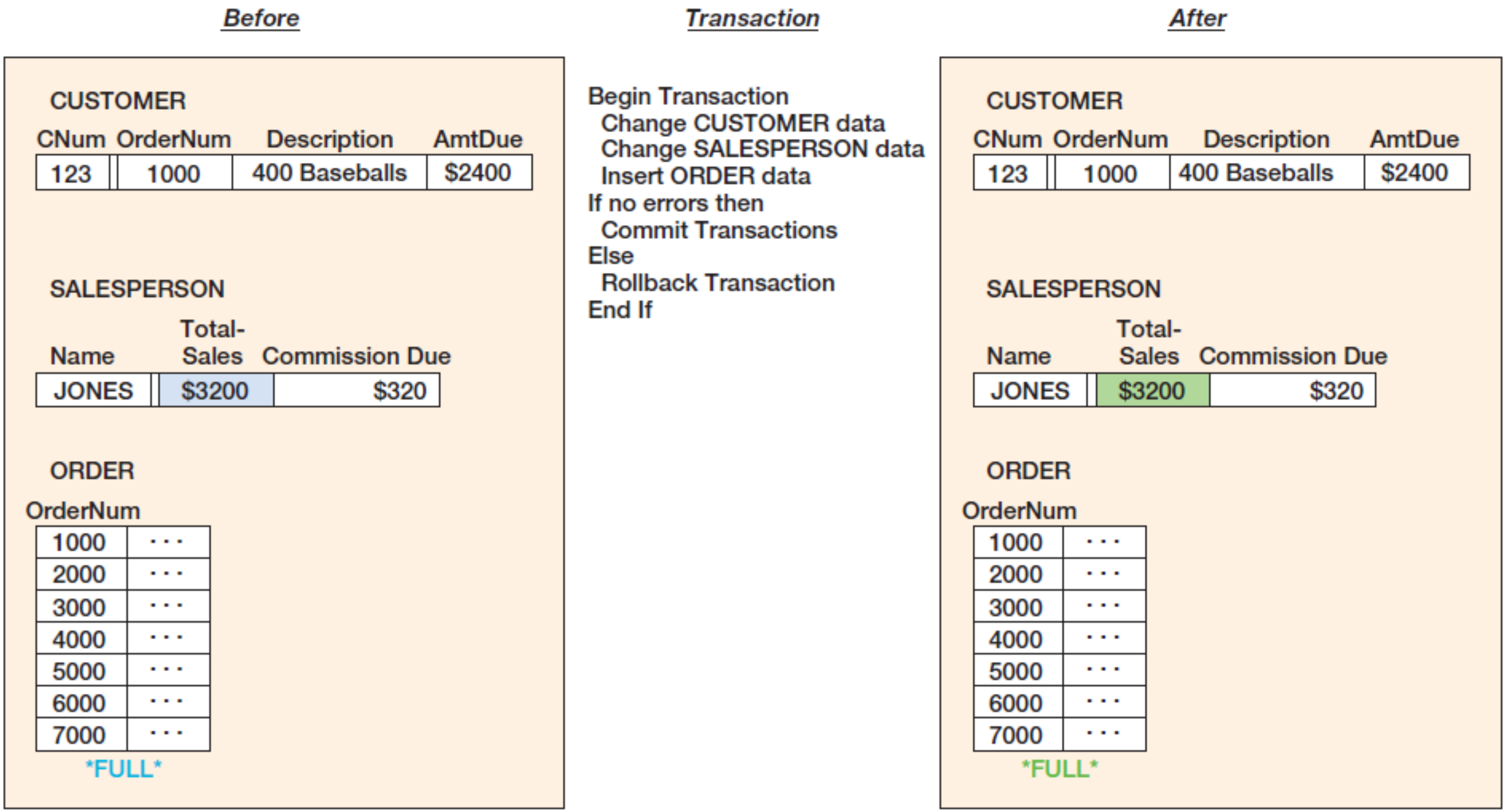
OrderNum	
1000	· · ·
2000	· · ·
3000	· · ·
4000	· · ·
5000	· · ·
6000	· · ·
7000	· · ·

FULL

(a) Errors Introduced Without Transaction

Transaction Control

With atomic transaction control



Before

CUSTOMER

CNum	OrderNum	Description	AmtDue
123	1000	400 Baseballs	$2400

SALESPERSON

Name	Total-Sales	Commission Due
JONES	$3200	$320

ORDER

OrderNum	
1000	· · ·
2000	· · ·
3000	· · ·
4000	· · ·
5000	· · ·
6000	· · ·
7000	· · ·

FULL

Transaction

```
Begin Transaction
  Change CUSTOMER data
  Change SALESPERSON data
  Insert ORDER data
If no errors then
  Commit Transactions
Else
  Rollback Transaction
End If
```

After

CUSTOMER

CNum	OrderNum	Description	AmtDue
123	1000	400 Baseballs	$2400

SALESPERSON

Name	Total-Sales	Commission Due
JONES	$3200	$320

ORDER

OrderNum	
1000	· · ·
2000	· · ·
3000	· · ·
4000	· · ·
5000	· · ·
6000	· · ·
7000	· · ·

FULL

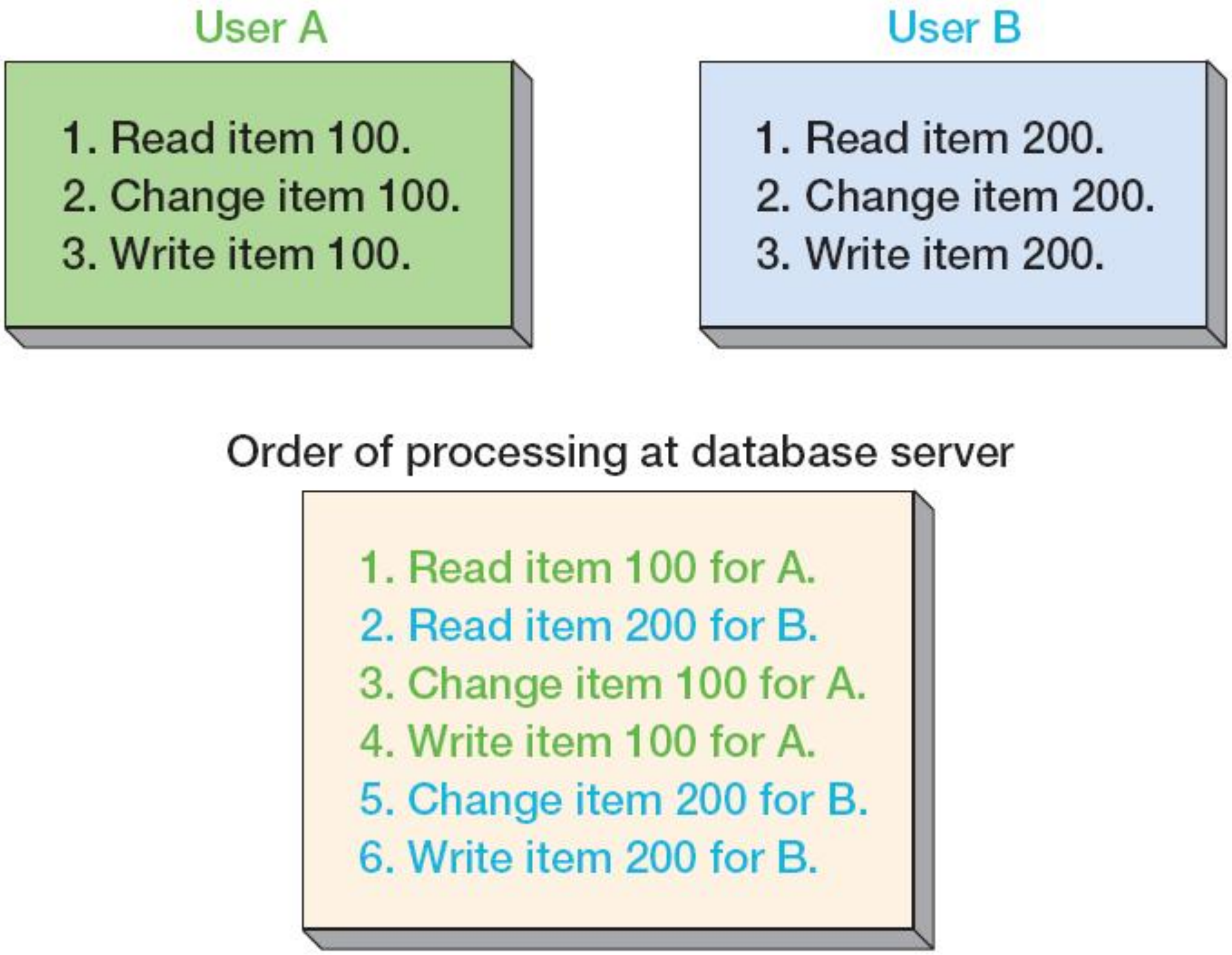
(b) Atomic Transaction Prevents Errors

Transaction Control

- **Concurrent transactions refer to two or more transactions that appear to users as they are being processed against a database at the same time.**
- **In reality, CPU can execute only one instruction at a time.**
  - **Transactions are interleaved** meaning that the operating system quickly switches CPU services among tasks so that some portion of each of them is carried out in a given interval.
- **Concurrency problems are lost updates and inconsistent reads.**

Transaction Control

- **Sequence of events without conflict**



User A

1. Read item 100.
2. Change item 100.
3. Write item 100.

User B

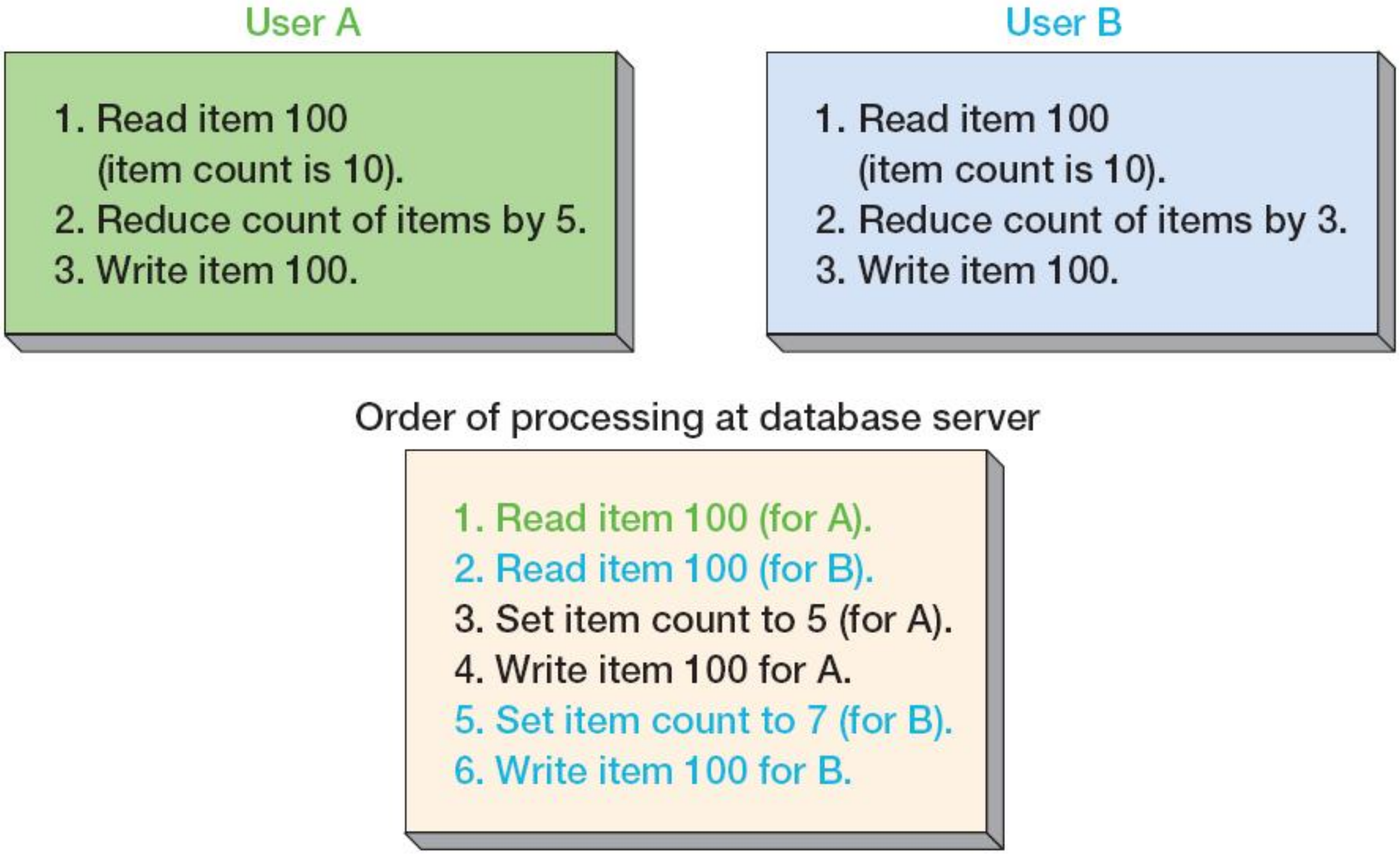
1. Read item 200.
2. Change item 200.
3. Write item 200.

Order of processing at database server

1. Read item 100 for A.
2. Read item 200 for B.
3. Change item 100 for A.
4. Write item 100 for A.
5. Change item 200 for B.
6. Write item 200 for B.

Transaction Control

- **Events with conflict – Lost Update**



User A

1. Read item 100
   (item count is 10).
2. Reduce count of items by 5.
3. Write item 100.

User B

1. Read item 100
   (item count is 10).
2. Reduce count of items by 3.
3. Write item 100.

Order of processing at database server

1. Read item 100 (for A).
2. Read item 100 (for B).
3. Set item count to 5 (for A).
4. Write item 100 for A.
5. Set item count to 7 (for B).
6. Write item 100 for B.

Note: The change and write in steps 3 and 4 are lost.

Handling Concurrency

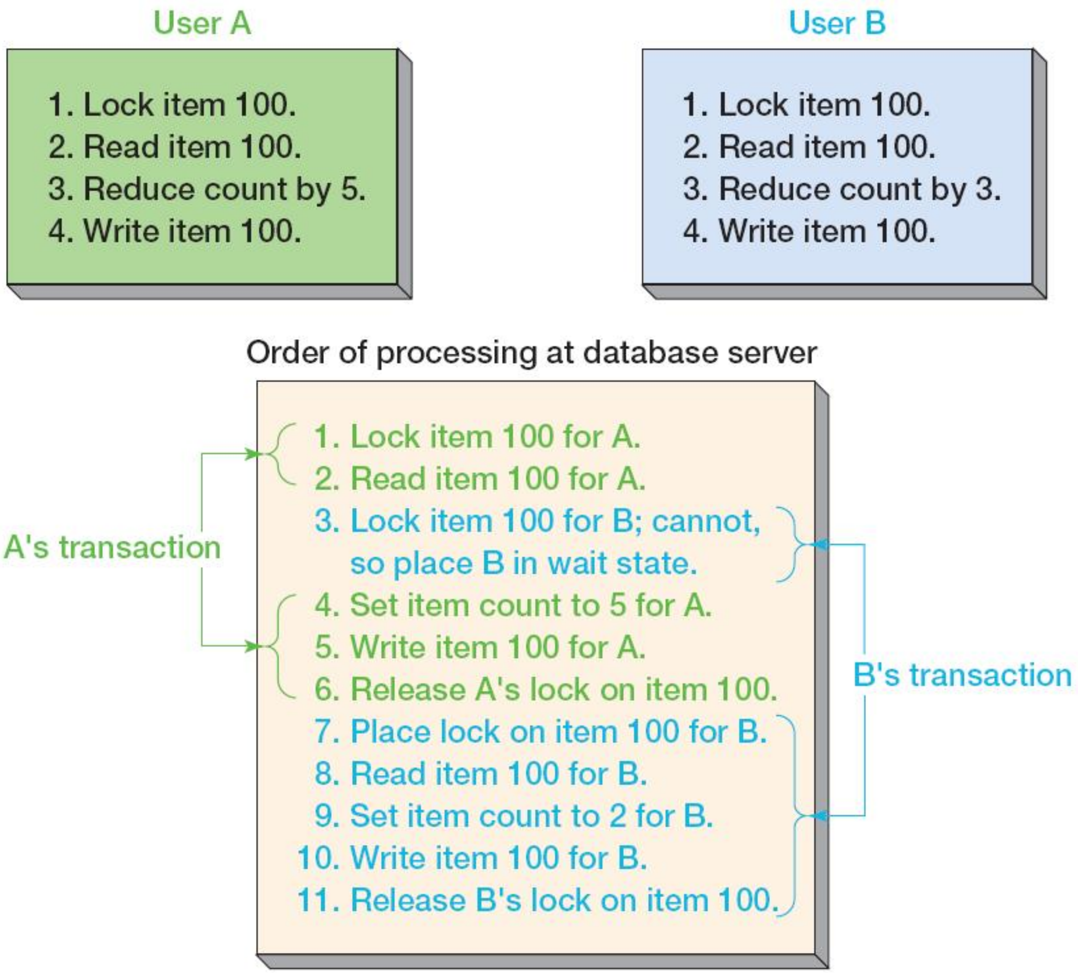
- **Resource locking prevents multiple applications from obtaining copies of the same record when the record is about to be changed.**

Handling Concurrency

- **Implicit locks are locks placed by the DBMS.**
- **Explicit locks are issued by the application program.**
- **Lock granularity refers to size of a locked resource:**
  - Rows, page, table, and database level.
- **Large granularity is easier to manage but frequently causes conflicts.**
- **Types of lock:**
  - An **exclusive lock** prohibits other users from reading the locked resource.
  - A **shared lock** allows other users to read the locked resource, but they cannot update it.

Handling Concurrency

- **Serialized Transactions**



User A

1. Lock item 100.
2. Read item 100.
3. Reduce count by 5.
4. Write item 100.

User B

1. Lock item 100.
2. Read item 100.
3. Reduce count by 3.
4. Write item 100.

Order of processing at database server

1. Lock item 100 for A.
2. Read item 100 for A.
3. Lock item 100 for B; cannot, so place B in wait state.
4. Set item count to 5 for A.
5. Write item 100 for A.
6. Release A's lock on item 100.
7. Place lock on item 100 for B.
8. Read item 100 for B.
9. Set item count to 2 for B.
10. Write item 100 for B.
11. Release B's lock on item 100.

A's transaction

B's transaction

Handling Concurrency

- Serializable transactions refer to two transactions that run concurrently and generate results that are consistent with the results that would have occurred if they had run separately.
- Two-phased locking is one of the techniques used to achieve serializability.

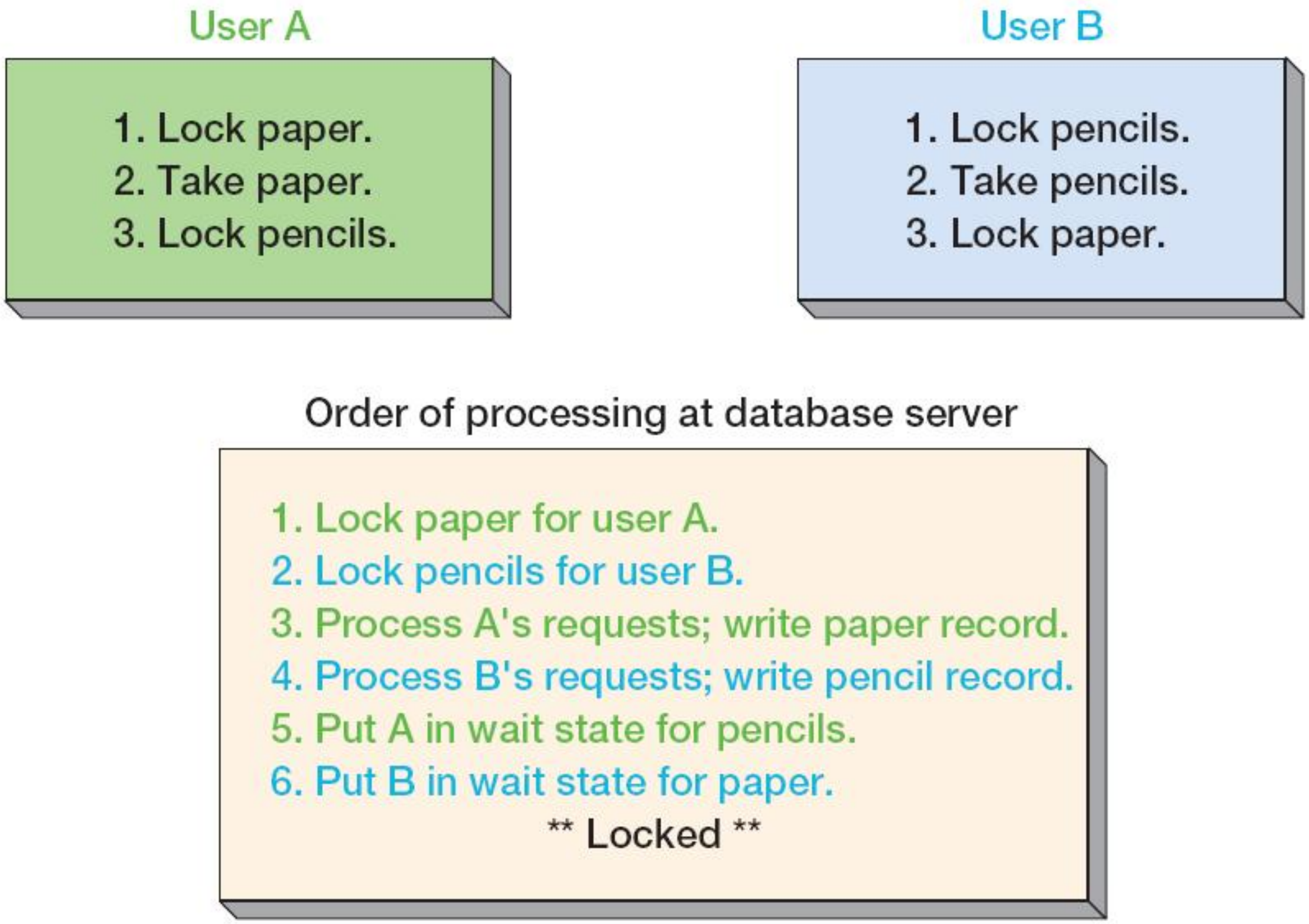
Handling Concurrency

- **Two-phased locking**:
  - Transactions obtain locks as necessary (**growing phase**) until all necessary objects are locked.
  - Once any lock is released (**shrinking phase**), subsequent locks will be released until all are released, but the transaction cannot grab more new locks.
- **A special simpified case of two-phased locking:**
  - Locks are obtained throughout the transaction.
  - No lock is released until the COMMIT or ROLLBACK command is issued.
  - This strategy is more restrictive but easier to implement than two-phase locking.

Handling Concurrency

- **Deadlock, or the deadly embrace, occurs when two transactions are each waiting on a resource that the other transaction holds.**
- **Preventing deadlock:**
  - Allow users to issue all lock requests up front.
- **Breaking deadlock:**
  - Almost every DBMS has algorithms for detecting deadlock.
  - When deadlock occurs, DBMS aborts one of the transactions (selected randomly), allowing one transaciton to complete, while the aborted transaction rolls back any partially completed work.

Handling Concurrency



User A

1. Lock paper.
2. Take paper.
3. Lock pencils.

User B

1. Lock pencils.
2. Take pencils.
3. Lock paper.

Order of processing at database server

1. Lock paper for user A.
2. Lock pencils for user B.
3. Process A's requests; write paper record.
4. Process B's requests; write pencil record.
5. Put A in wait state for pencils.
6. Put B in wait state for paper.

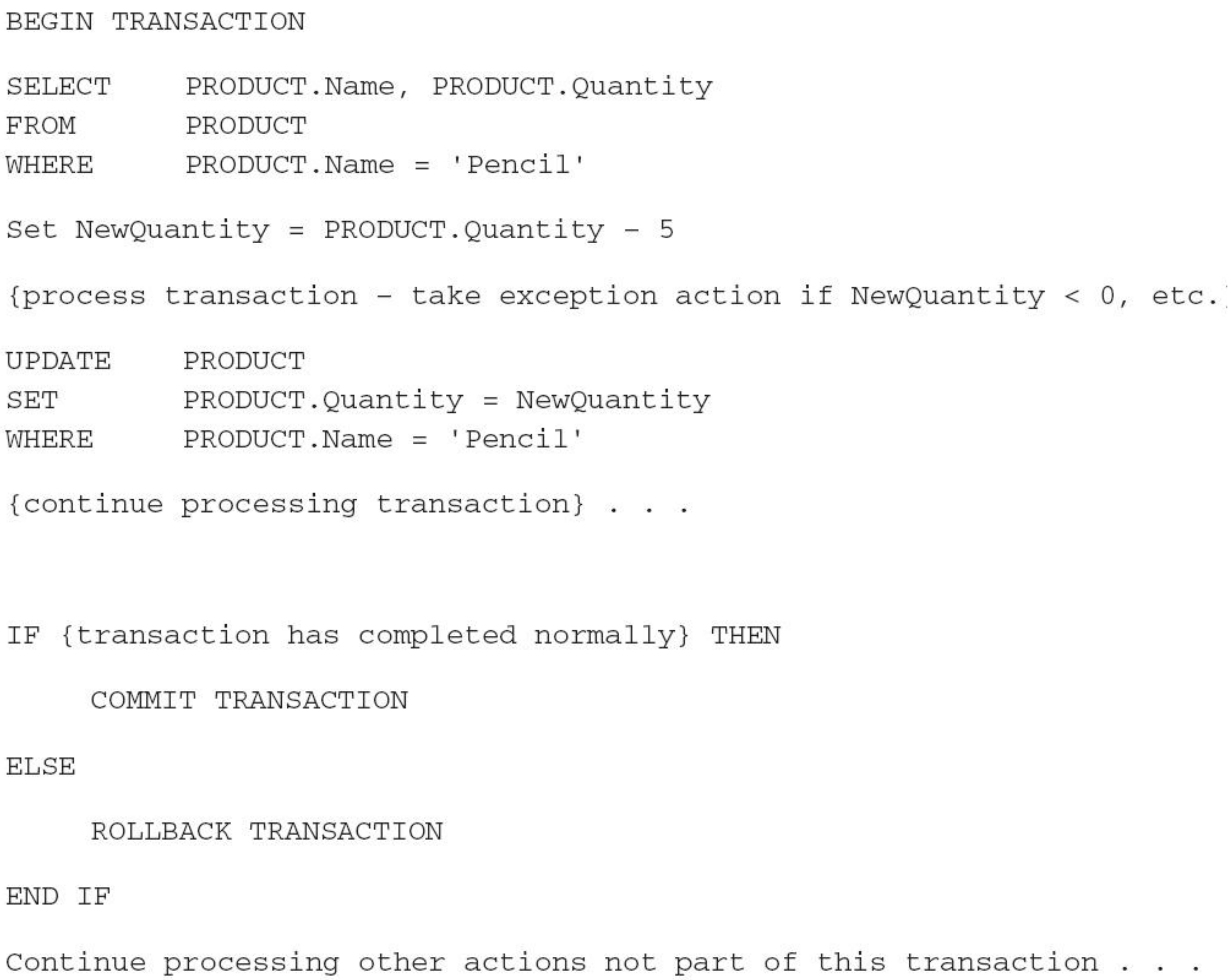
** Locked **

Handling Concurrency

- **Most application programs do not explicitly declare locks due to the complexities of managing them. An exception: some non-relational DBMS systems.**
- **Instead, they mark transaction boundaries and declare the locking behavior they want the DBMS to use.**
  - Transaction boundary markers: START, COMMIT, and ROLLBACK TRANSACTION.

Handling Concurrency

- **Psuedo-Code**



```
BEGIN TRANSACTION

SELECT     PRODUCT.Name, PRODUCT.Quantity
FROM       PRODUCT
WHERE      PRODUCT.Name = 'Pencil'

Set NewQuantity = PRODUCT.Quantity - 5

{process transaction - take exception action if NewQuantity < 0, etc.}

UPDATE     PRODUCT
SET        PRODUCT.Quantity = NewQuantity
WHERE      PRODUCT.Name = 'Pencil'

{continue processing transaction} . . .


IF {transaction has completed normally} THEN

     COMMIT TRANSACTION

ELSE

     ROLLBACK TRANSACTION

END IF

Continue processing other actions not part of this transaction . . .
```

Handling Concurrency

- **The acronym ACID**
  - Transaction management that is Atomic, Consistent, Isolated, and Durable.
- **Atomic means either ALL or NONE of the database actions within a transaction are committed.**
- **Durable means database committed changes are permanent.**

Handling Concurrency

- Consistency means the DBMS software will not allow any application programs to violate any database constraints. Once committed, the transaction leaves the data in a consistent state.

Handling Concurrency

- **Isolation means the DBMS software**
  - – Allows application programmers are able to declare the desired isolation level,
  - – So that they can trust that the the DBMS manages locks so as to achieve that level of isolation.

Handling Concurrency

Anomalies:

"Dirty Read" – a READ request is allowed to read from cache the updated but uncommitted copy of the data

"Unrepeatable Read" – One READ request following another READ request gets two different results.

Alice and Bob are using the Fandango website to book tickets for a specific movie showing. Only one ticket is left for the specific show. Alice signs on first to see that only one ticket is left, and feels like it is a bit expensive. Alice takes time to decide. Bob signs on and also finds one ticket left, and buys it instantly. Bob makes his purchase and logs off. Alice decides to buy a ticket, and finds there are no more tickets. This is a typical unrepeatable read situation.

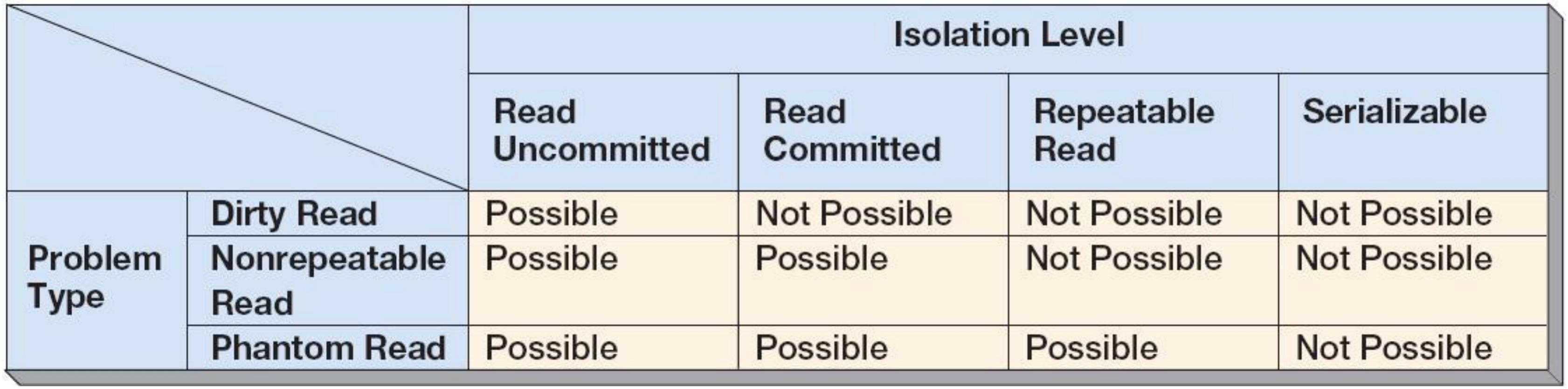
Anomalies:

"Phantom Read" – a READ request returns a different set of rows at different times. For example the first SELECT returns 10 rows; the next SELECT returns 11 rows (including a "phantom" row) that was inserted after the first read.

Handling Concurrency

- The ANSI SQL-92 standard defines four transaction isolation levels:
  - Read uncommitted – allows reads of uncommitted changes ‘i.e. “dirty” reads, unrepeatable reads and phantom reads
  - Read committed – allows unrepeatable reads and phantom reads, but no dirty reads
  - Repeatable read – allows phantom reads, but no unrepeatable reads
  - Serializable – guarantees no read anomalies

Handling Concurrency



		Isolation Level			
		Read Uncommitted	Read Committed	Repeatable Read	Serializable
Problem Type	Dirty Read	Possible	Not Possible	Not Possible	Not Possible
	Nonrepeatable Read	Possible	Possible	Not Possible	Not Possible
	Phantom Read	Possible	Possible	Possible	Not Possible

Isolation

- **What's the difference?**
  - – The speed at which I/O ops get done
  - – The frequency and duration of various locks
- **Trade-offs**
  - – Faster throughput versus data consistency and reliability

Handling Concurrency

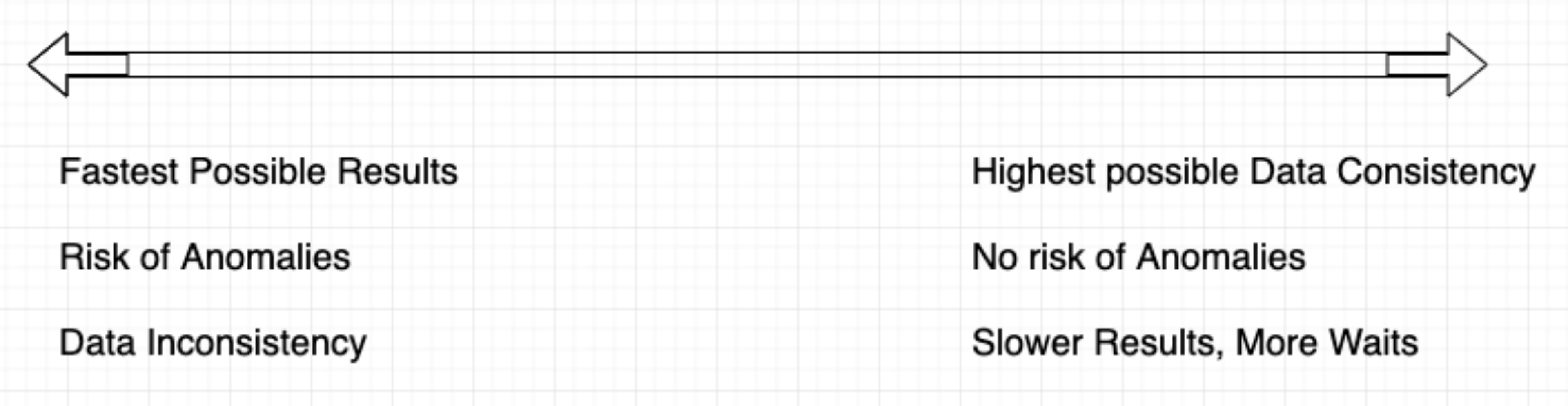
Why do we care about ACID?

- Relational DBMS software is written with extremely complex algorithms to ensure ACID compliance

- Ensuring ACID compliance has trade-offs
  - Do I want the fastest possible processing? That is, applications/users never have to wait for commits, locks, etc.
  - Do I want to be sure that my application/users maintains complete data integrity?

Handling Concurrency

The DBMS software itself determines where you land on this scale.



Fastest Possible Results

Risk of Anomalies

Data Inconsistency

Highest possible Data Consistency

No risk of Anomalies

Slower Results, More Waits

NoSQL databases ⟺ Relational Databases