COM6002 Big Data Management

Relational Database Management Systems

(RDBMS)

Is RDBMS important?

- Check this:
  - https://db-engines.com/en/ranking
- RDBMS dominates the market!
- Some important RDBMS:
  - Oracle
  - MySQL / MariaDB
  - Microsoft MS SQL
  - Postgre SQL
  - IBM DB2
  - SQLite
  - Microsoft Access (Not a real powerful RDBMS)

RDBMS

- Use tables to manage data

Table is called “relation”

Student ID	Name	Major
101101	Amy	BBA
101103	Bob	CS
101106	Cathy	CS

Module Code	Module Name
COM2103	Database Design and Management
COM2006	Database Management Systems

Student ID	Module Code	Grade
101101	COM2006	B
101101	COM2103	A
101103	COM2103	A
101106	COM2103	B

Search in RDBMS

- Q: What are the names of students who get A in the module "Database Design and Management"

Student ID	Name	Major
101101	Amy	BBA
101103	Bob	CS
101106	Cathy	CS

Module Code	Module Name
COM2103	Database Design and Management
COM2006	Database Management Systems

Student ID	Module Code	Grade
101101	COM2006	B
101101	COM2103	A
101103	COM2103	A
101106	COM2103	B

Search in RDBMS

- Joining tables
  - Q: What are the names of students who get A in the module "Database Design and Management"
  - We "join" the tables together to find the information that we need

Student ID	Name	Major	Module Code	Module Name	Grade
101101	Amy	BBA	COM2006	Database Management Systems	B
101101	Amy	BBA	COM2103	Database Design and Management	A
101103	Bob	CS	COM2103	Database Design and Management	A
101106	Cathy	CS	COM2103	Database Design and Management	B

Efficient search

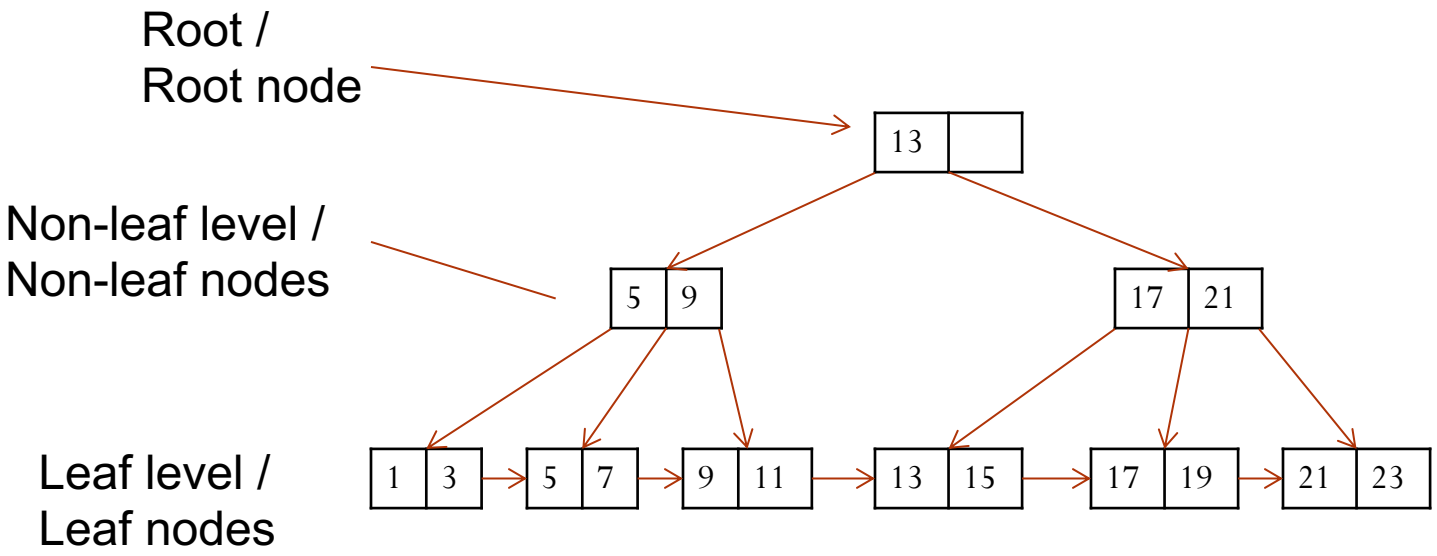
- How long does it take to find a user account in Instagram's (IG) database, e.g., in a login process?
  - IG has **2.4 billion** users as of 2024
  - Say your program can loop through 100M* records per second
  - How long does it take? ☺

- * Personal experience:
  - Time limit on Leetcode: 10 seconds (for Python)
  - Input size (n):
    - 1M: O(n) solution is needed
    - 100K: Probably O(n lg(n))
    - 1000: $O(n^2)$ is fine

Index on RDBMS

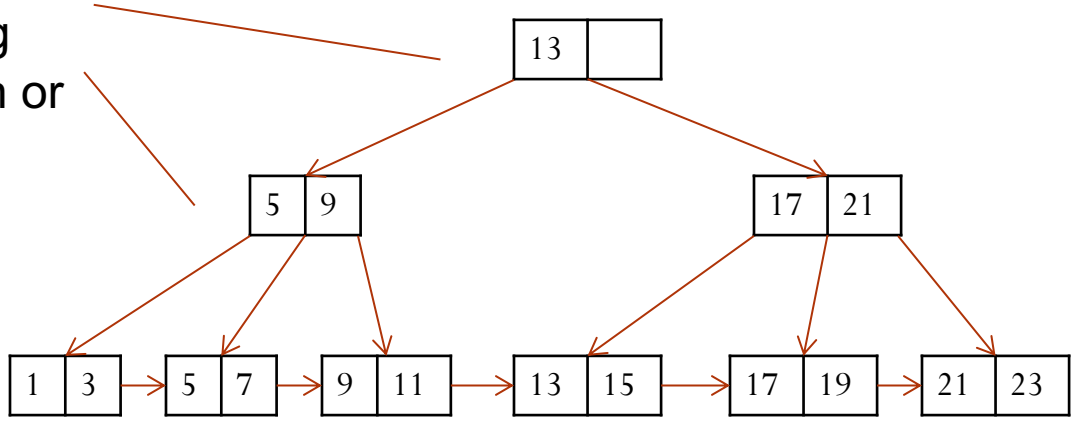
- Common method: B+ tree (or its variants)
  - The ideas are similar but there are minor implementation details in different versions
  - We will talk about one specific version – B+ tree
- Cost complexity of search: O(lg n)
  - Log(2.4B) = 9.38
  - Note: recall 9.38 is not the actual time. We are simply talking about the scale of complexity

Structure of B+ tree



Structure of B+ tree

Values at non-leaf level are separators only. Everything on the left is smaller than the separator. Everything on the right is larger than or equal to the separator

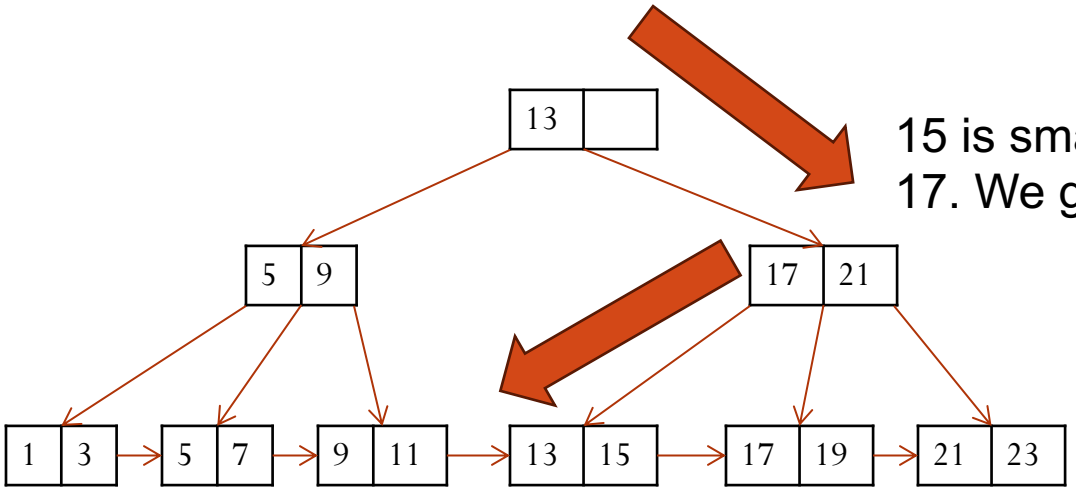


Record ***pointers*** are always at leaf level

Q: Find the record with value 15.

Finding 15

15 is bigger than 13. We go to the right



15 is smaller than 17. We go to the left

Bingo. 15 is here

Another important aspect of index

- The update cost is also O(lg n)
  - Example: Binary tree (a well-known data structure) has a linear update cost
- Animation with B+ tree maintenance and search
  - https://www.cs.usfca.edu/~galles/visualization/BPlusTree.html

Some key characteristics about B+ tree

- Each node can hold multiple keys and pointers to optimize disk I/O
- Each node is not too "empty" (half-full policy)
  - If the fan-out is 1, the height of the tree can be infinitely large
  - To optimize the query cost and update cost
- Balanced tree
  - The path length from to root node to any leaf node is the same
- B+ tree is for one-dimensional search
  - Although we can imagine we can combine multiple attributes into one attribute so that we can use B+ tree for indexing
  - There are other index structures for other queries
    - For example, R tree is for multi-dimensional queries

Discussions

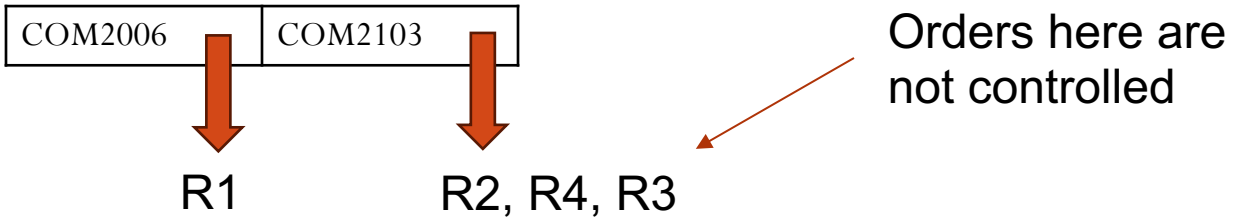
- A query is to list of student IDs of students who get A in COM3007. How to build the index and how can the index help?
  - If the index is built on module code?
  - If the index is built on Grade?
  - If we have two indices: one on module and one on grade?
    - Yes! We can have multiple indices on the same table!
  - If the index is built on (module code, grade)?

We will visualize these four indices on the next few slides

Student ID	Module Code	Grade
101101	COM2006	B
101101	COM2103	A
101103	COM2103	A
101106	COM2103	B

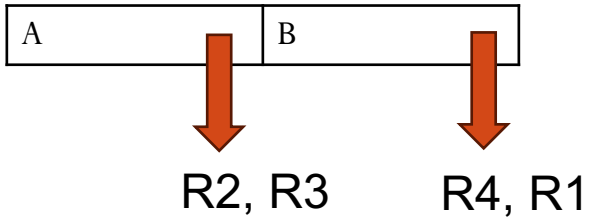
Index on module code

- Only one level
  - Recall: the keys at the leaf level are ordered!



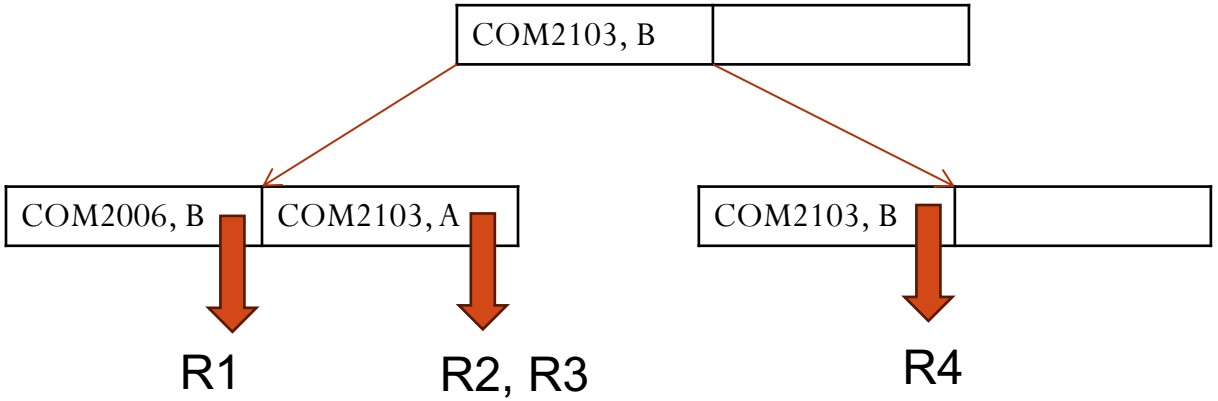
	Student ID	Module Code	Grade
R1	101101	COM2006	B
R2	101101	COM2103	A
R3	101103	COM2103	A
R4	101106	COM2103	B

Index on grade



	Student ID	Module Code	Grade
R1	101101	COM2006	B
R2	101101	COM2103	A
R3	101103	COM2103	A
R4	101106	COM2103	B

Index on module code, grade



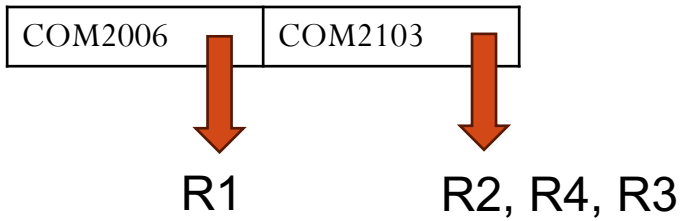
	Student ID	Module Code	Grade
R1	101101	COM2006	B
R2	101101	COM2103	A
R3	101103	COM2103	A
R4	101106	COM2103	B

The price of indexing

- The index requires additional space and maintenance effort
- The additional space required is less than the space required by the records
  - Note that only pointers to records are kept in the database. There is always only one copy of the record contents
- Q: Is the maintenance effort worth it?
  - How often is your index used?
  - Is it necessary?
- Q: How do we process a multi-dimensional query? (See next slide)

Query plan

- There are various query plans for the same query
- Following our example,
  - Query: Students who get A in COM3007
  - Query plan 1:
    - We don't use any index and scan all records
  - Query plan 2:
    - We use the index on module code and check all records of COM3007
  - Query plan 3:
    - Use the index on grade
  - Query plan 4:
    - Use the index on (module code, grade)



Query optimizer

- RDBMS will try to find the optimal **query plan** for your query
- It keeps statistics about the database, like number of records
- Automatically pick the "best" query plan

- Don't rely on query optimizer. It can perform query rewriting for simple patterns, e.g., filter-join query.
  - The query optimizer has hard time rewriting more complex queries

- Try to write efficient SQL queries for your tasks
  - SQL will be discussed in the next chapter

Database transaction

- Transaction
  - The basic unit in RDBMS
- Each transaction may contain several commands
- Example
  - Transfer money from one bank account to another
    - 1. Deduct $5,000 from your account
    - 2. Add $5,000 to your friend's account
  - Both actions 1 and 2 are done or none is
- Q: Is it hard for the computer to handle transactions?
  - Imagine what will happen when the computer hangs after step 1 but before step 2?

Transaction

- Transaction structure

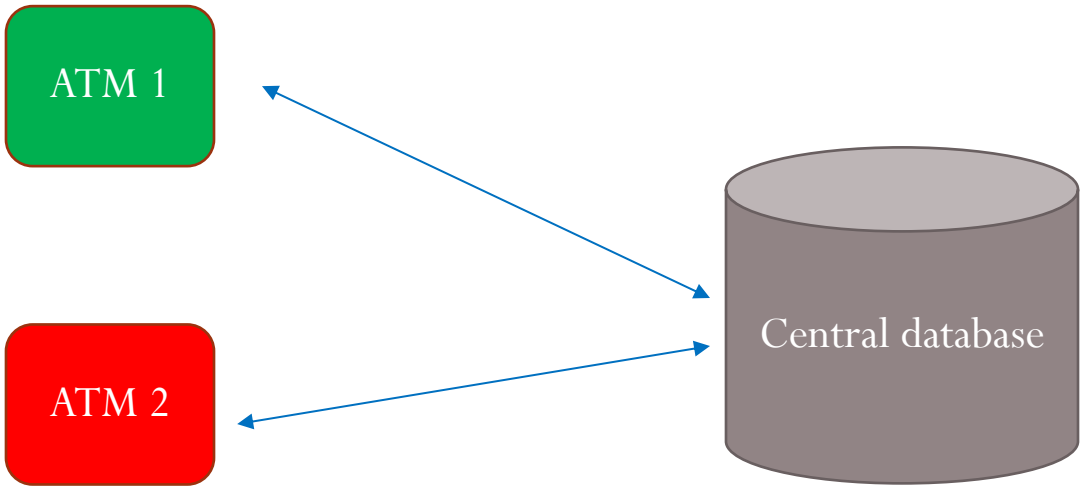
Transaction begins
… (All the actions you need to do)
Commit

- Once a transaction is **committed**, DBMS ensures that the transaction is really done entirely in the system
- Otherwise, the transaction should have no effect to the database

- Self-study [Transaction on MySQL]:
  - https://dev.mysql.com/doc/refman/8.4/en/commit.html

Challenges of handling multiple transactions

- Scenario – concurrent execution
  - We have multiple ATMs accessing the central database simultaneously



Why do we want concurrent execution?

- What happens if your system does not support concurrent execution?
  - The computer works on one transaction only at a time
  - Everyone queues up for one ATM only
    - Not just the customers, but also the staff in the bank

- Q: Is it difficult to handle concurrent transactions?

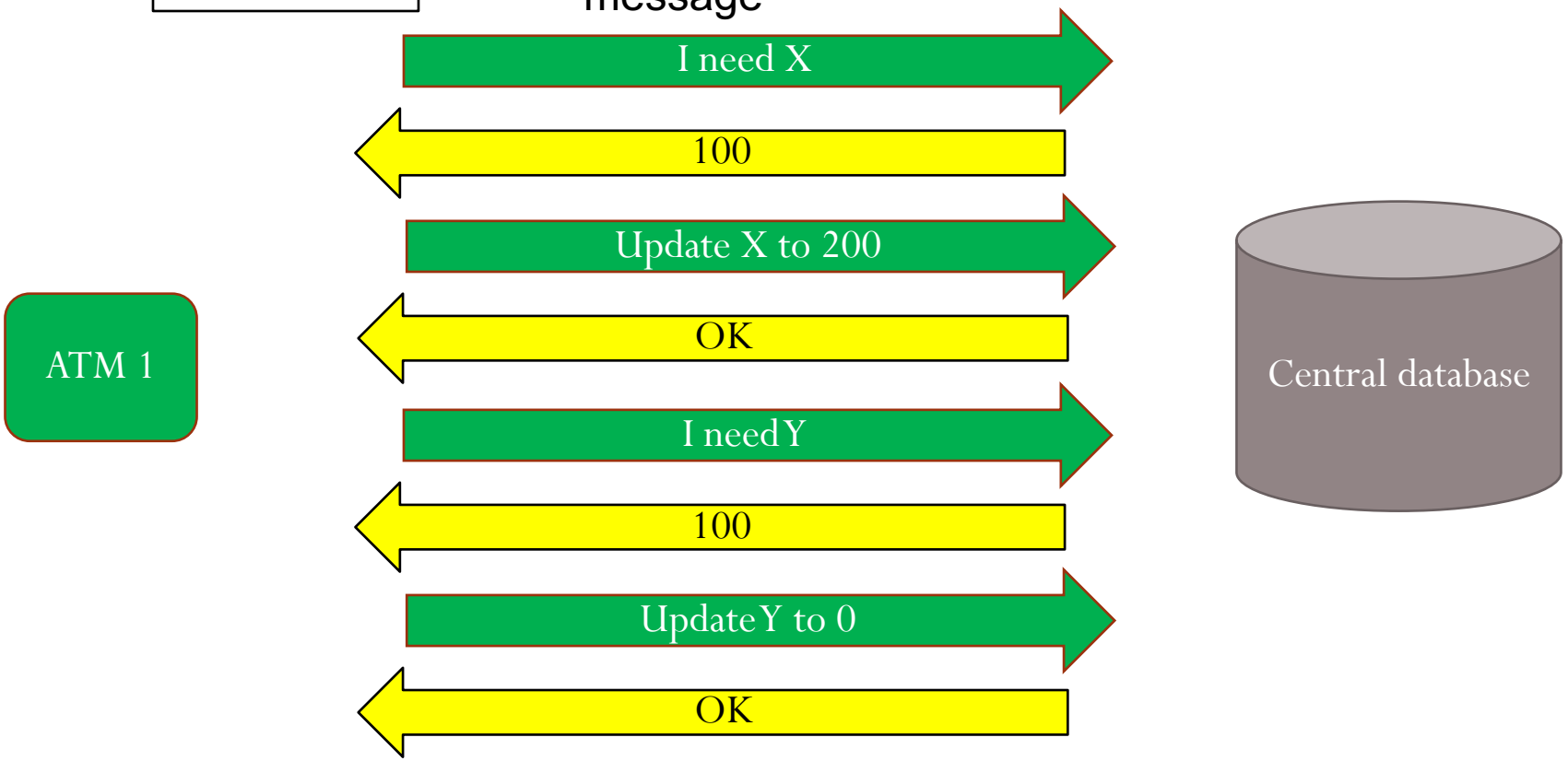
From the view of ATM 1

- Transaction:

```
X = X+ 100
Y = Y - 100
```

Q: Why do we need the "OK" response from the central database?

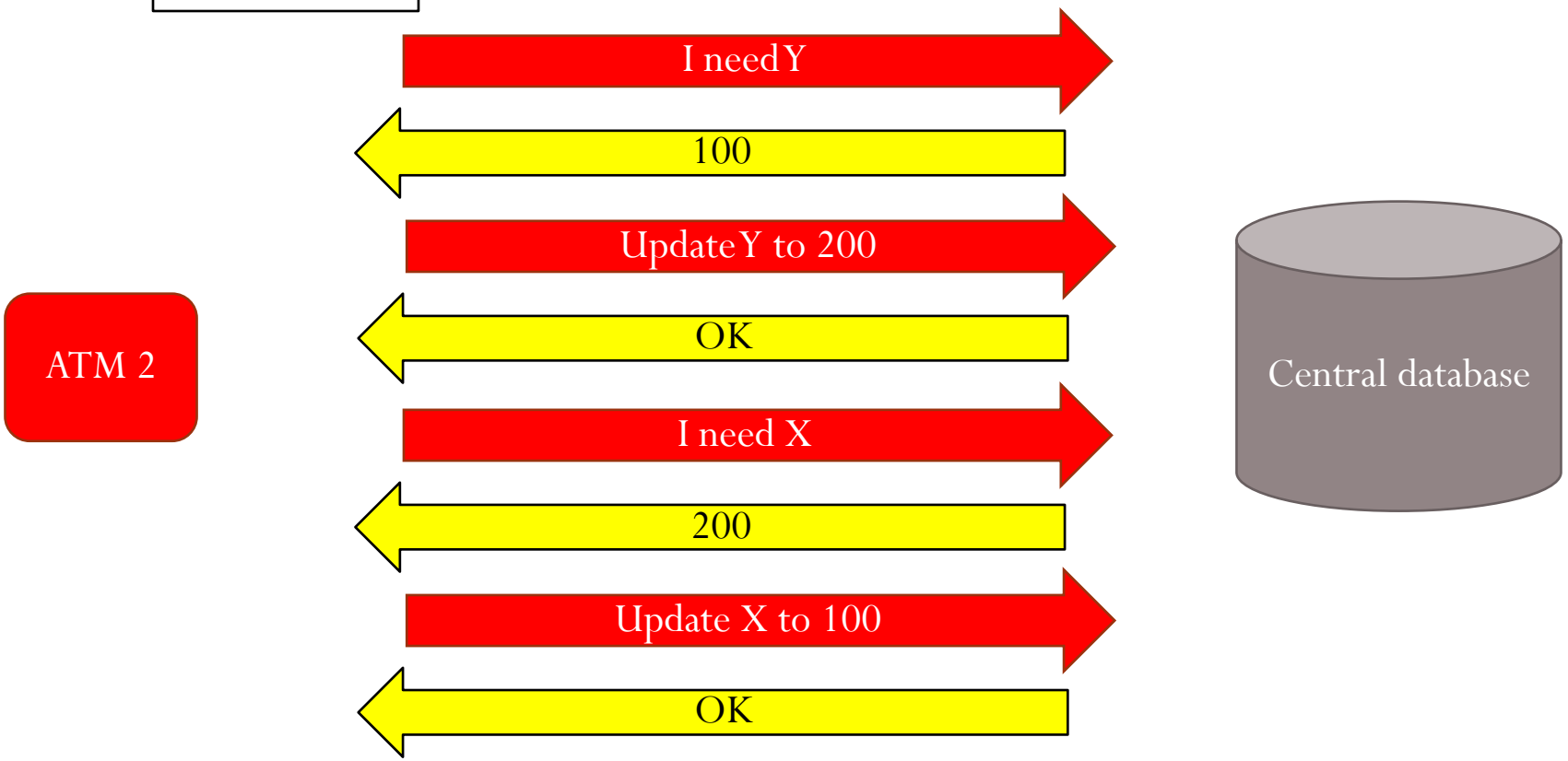
A: The network may drop a message. If we do not receive an OK for a long time, we will resend our message



From the view of ATM 2

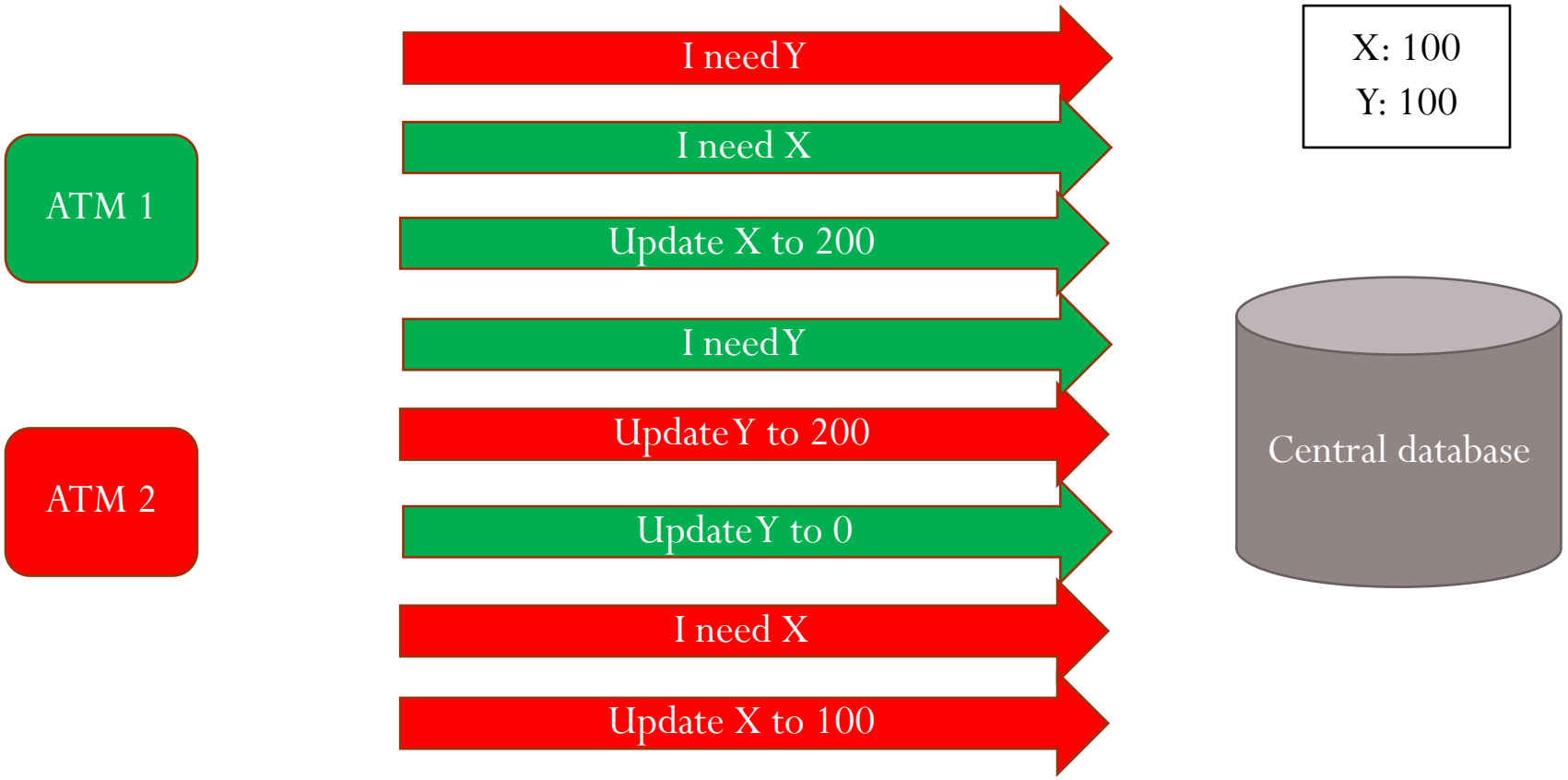
- Transaction:

```
Y = Y + 100
X = X - 100
```



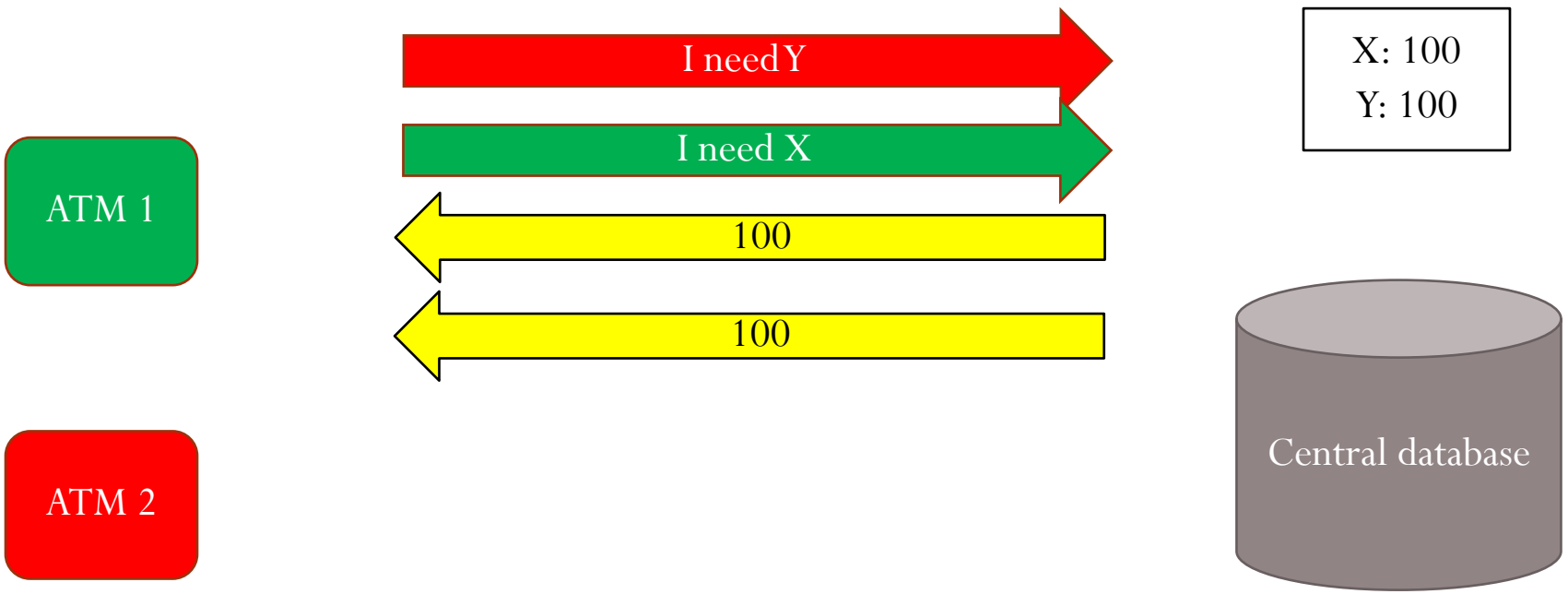
From the view of the database

- What will happen?



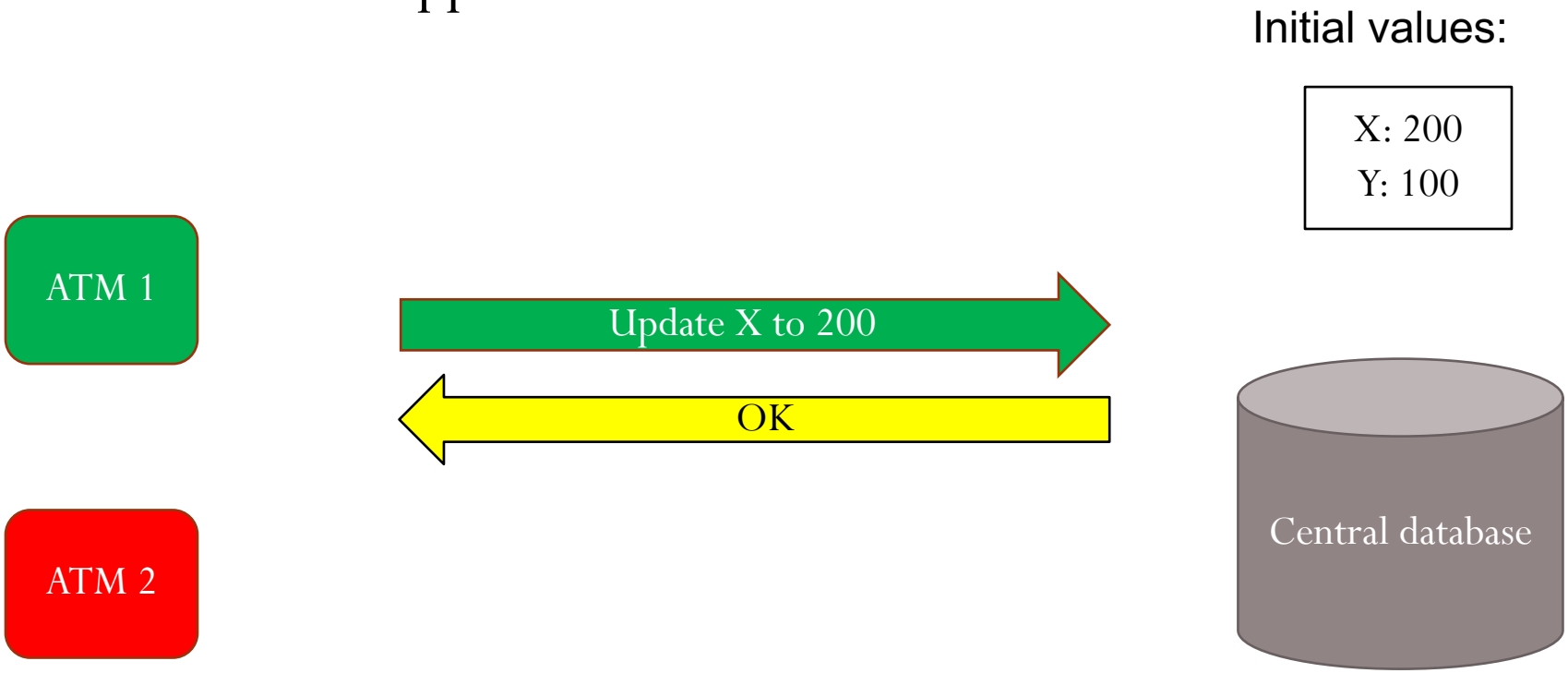
From the view of the database

- What will happen?



From the view of the database

- What will happen?

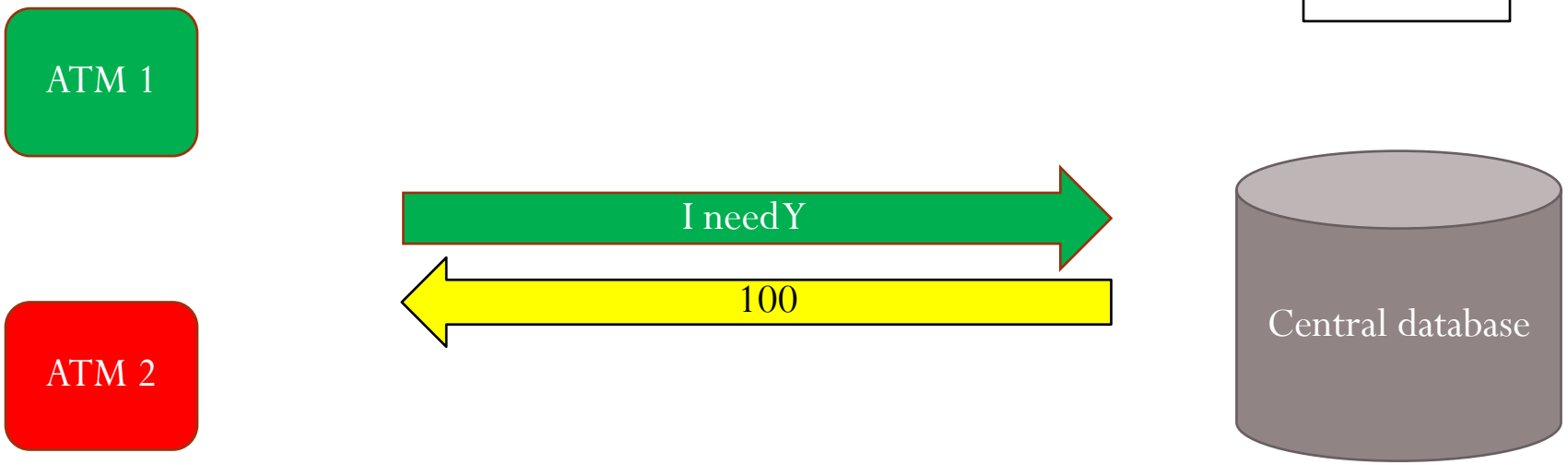


From the view of the database

- What will happen?

Initial values:

X: 200
Y: 100

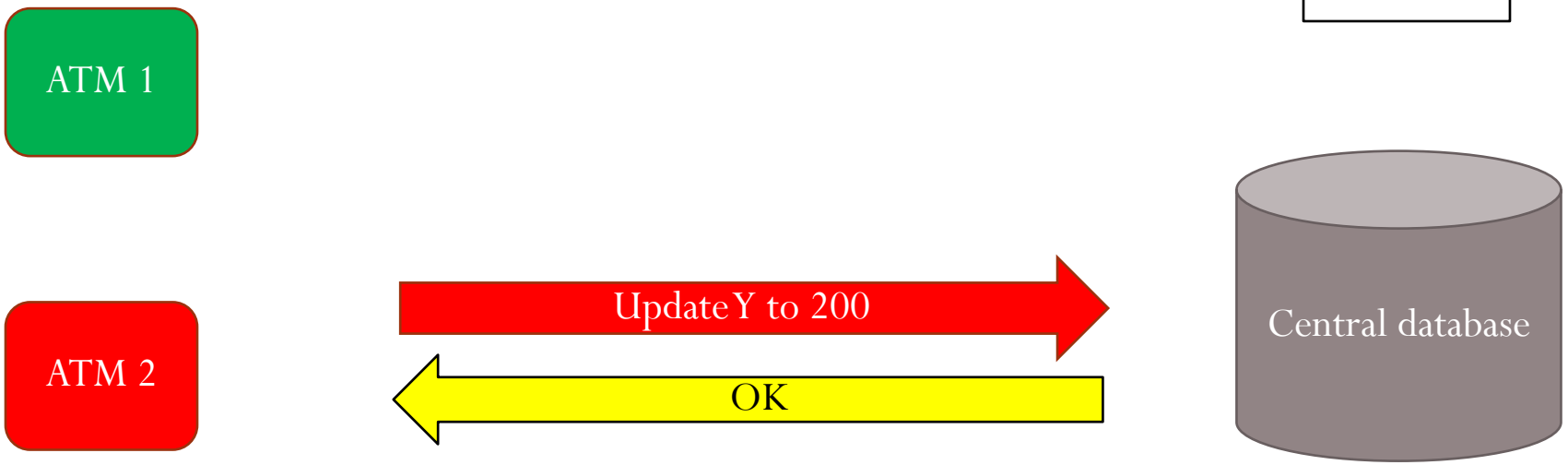


From the view of the database

- What will happen?

Initial values:

X: 200
Y: 200

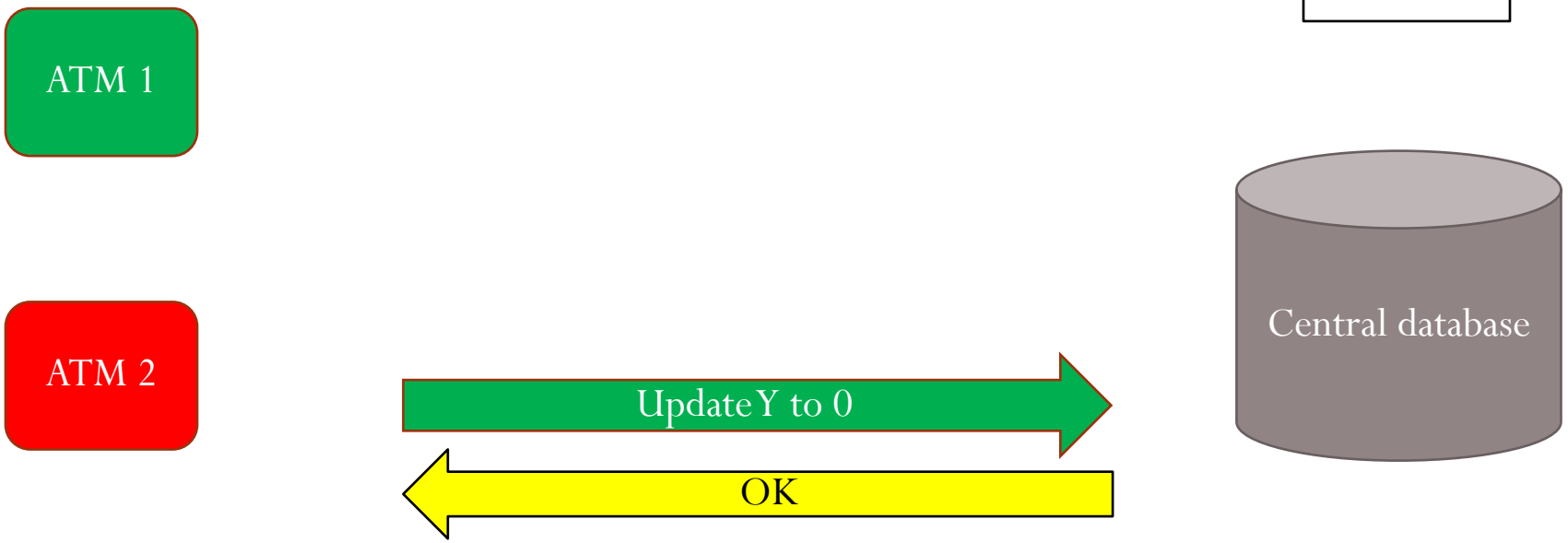


From the view of the database

- What will happen?

Initial values:

X: 200
Y: 0

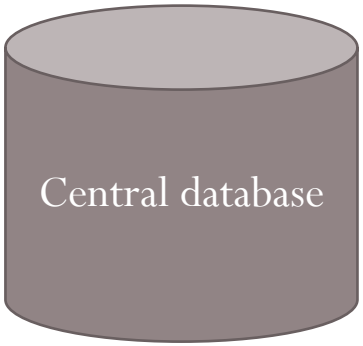


From the view of the database

- What will happen?

Initial values:

X: 200
Y: 0



From the view of the database

- What will happen?

Initial values:

X: 100
Y: 0

ATM 1

ATM 2

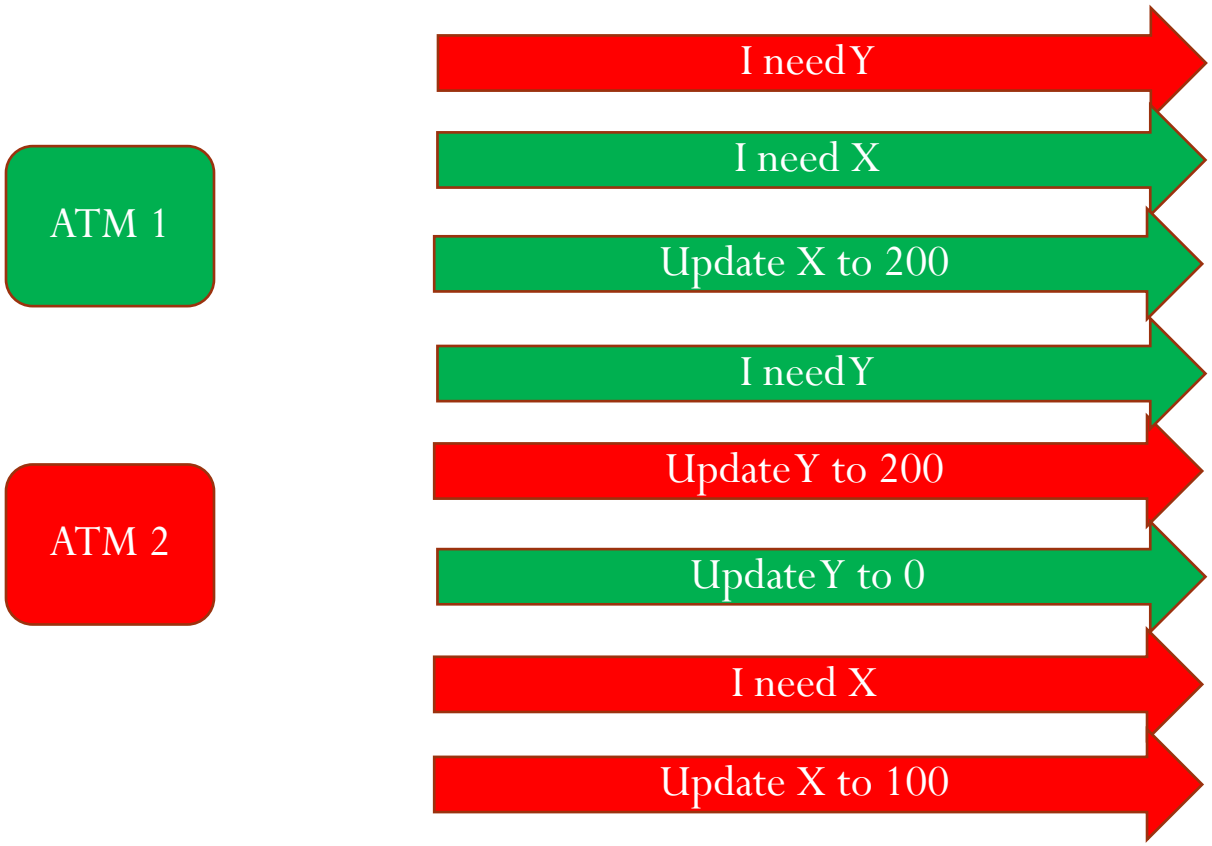
Central database

Update X to 100

OK

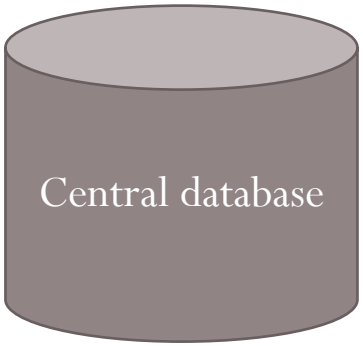
From the view of the database

- Something wrong?



Initial values:

X: 100
Y: 100

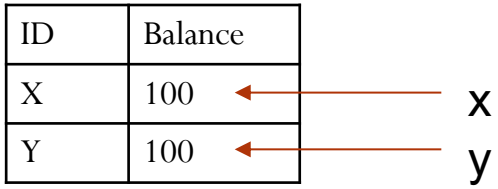


Final values:

X: 100
Y: 0

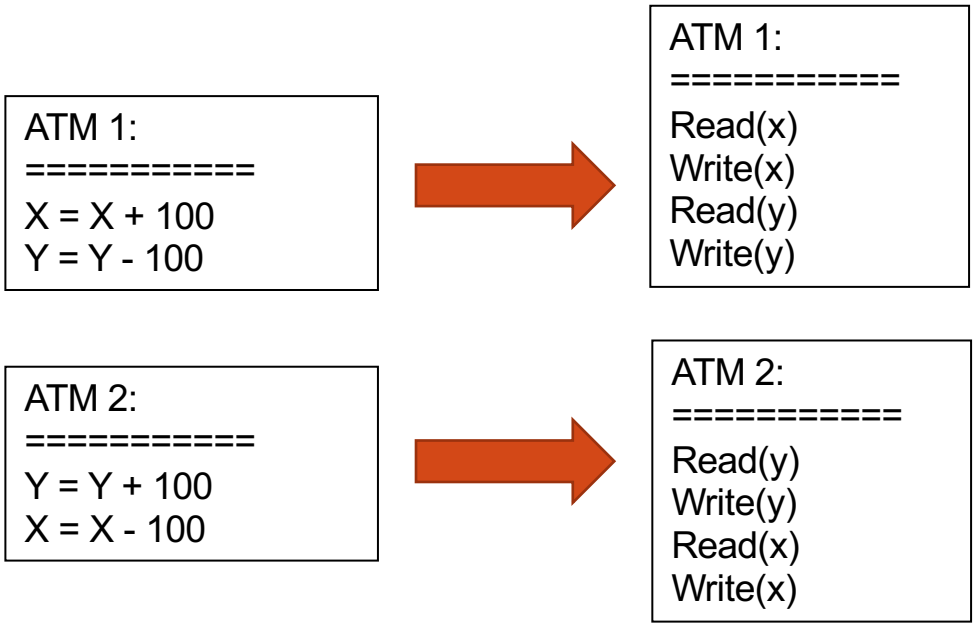
Model for transaction processing

- Not all actions in the program are related to DBMS
- We use the following model to simplify the discussions
  - Each value in the database is denoted by a variable



  - We consider only read and write actions
    - Each read/write action is done on one variable

Describing a transaction



What does it mean by a correct result?

- What is the correct result if the two transactions are run concurrently?

ID	Balance
X	1000

Initial database

Transaction 1:
==========
X increases by 1%

The bank gives you interest

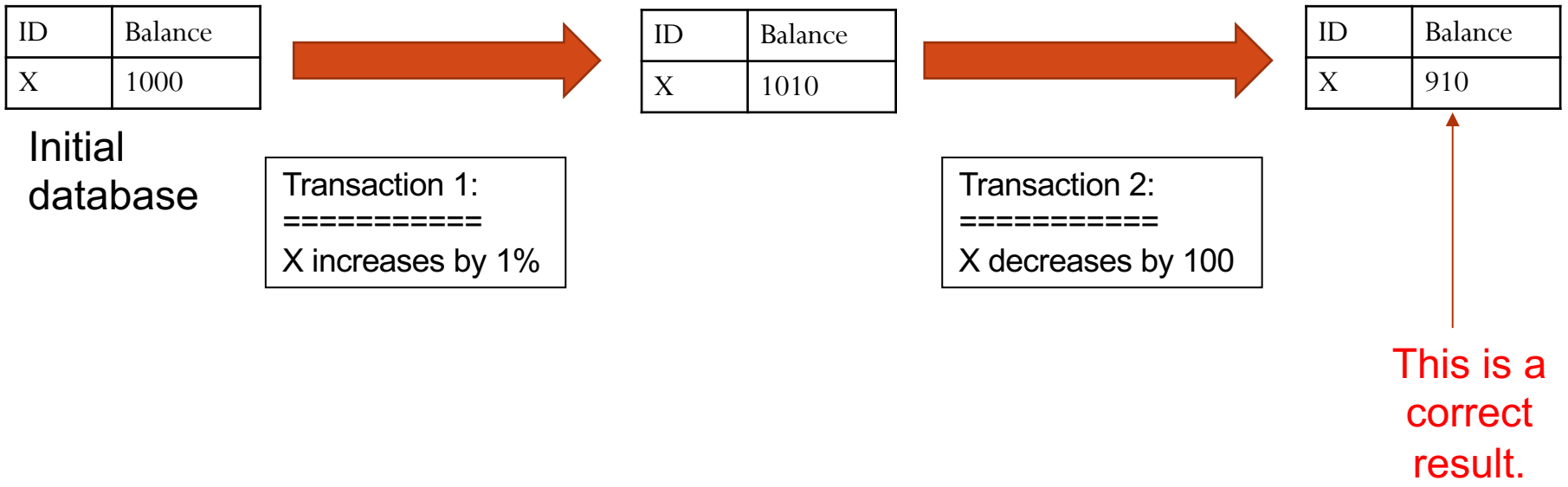
Transaction 2:
==========
X decreases by 100

Withdraw money from ATM

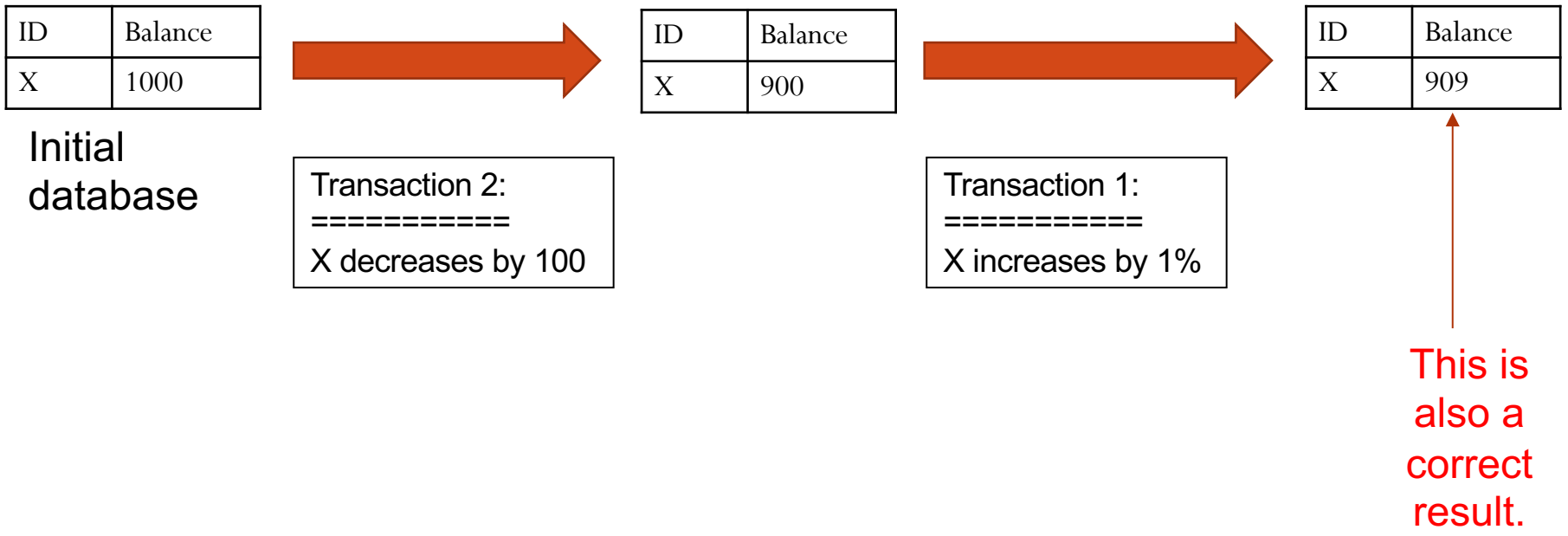
Serial execution

- Assumption: each transaction is itself correct, i.e., it brings the database from a consistent state to another consistent state
- Serial execution:
  - The transactions are executed one by one
  - Serial execution is correct
- There are multiple possible serial executions in the scenario giving different results
  - They are both deemed correct

Example serial execution 1



Example serial execution 2



Example concurrent execution

Transaction 1:
==========
X increases by 1%

Transaction 2:
==========
X decreases by 100

ID	Balance
X	1000

Read(x)

Write(x)

Read(x) of Transaction 1 is done first.
Then, Read(x) of Transaction 2 is done.
Next, Write(x) of Transaction 1 is done.
At last, Write(x) of Transaction 2 is done.

Read(x)

Write(x)

Analysis of concurrent execution

- **Two actions are not in conflict if they are done on different variables**
  - Write(x) and write(y) have no conflict
- If two actions are on the same variable

Action 1 (from T1)	Action 2 (from T2)	Conflict?	Explanation
Read(x)	Read(x)	No	Both read the same value in either order
Read(x)	Write(x)	Yes	Action 1 may read a different value in different order
Write(x)	Read(x)	Yes	Action 2 may read a different value in different order
Write(x)	Write(x)	Yes	The value in the database is set by the last write action

Execution plan

- Denote:
  - $R_i(x)$ as the action Read(x) by Transaction i
  - $W_i(x)$ as the action Write(x) by Transaction i
- We have the following transactions
  - T_1: $R_1(x)\ W_1(x)\ R_1(y)\ W_1(y)$
  - T_2: $R_2(x)\ W_2(x)\ R_2(y)\ W_2(y)$
- Is the following execution correct?
  - $R_1(x)\ W_1(x)\ R_1(y)\ W_1(y)\ R_2(x)\ W_2(x)\ R_2(y)\ W_2(y)$
- Is the following execution correct?
  - $R_1(x)\ W_1(x)\ R_2(x)\ W_2(x)\ R_1(y)\ W_1(y)\ R_2(y)\ W_2(y)$

Execution plan

They are on different variables.
No conflict

- $R_1(x)\, W_1(x)\, R_2(x)\, W_2(x)\, R_1(y)\, W_1(y)\, R_2(y)\, W_2(y)$

- $R_1(x)\, W_1(x)\, R_1(y)\, W_1(y)\, R_2(x)\, W_2(x)\, R_2(y)\, W_2(y)$

T_1 T_2

The execution has the same effect as a serial execution. This is a correct execution

Serializable execution

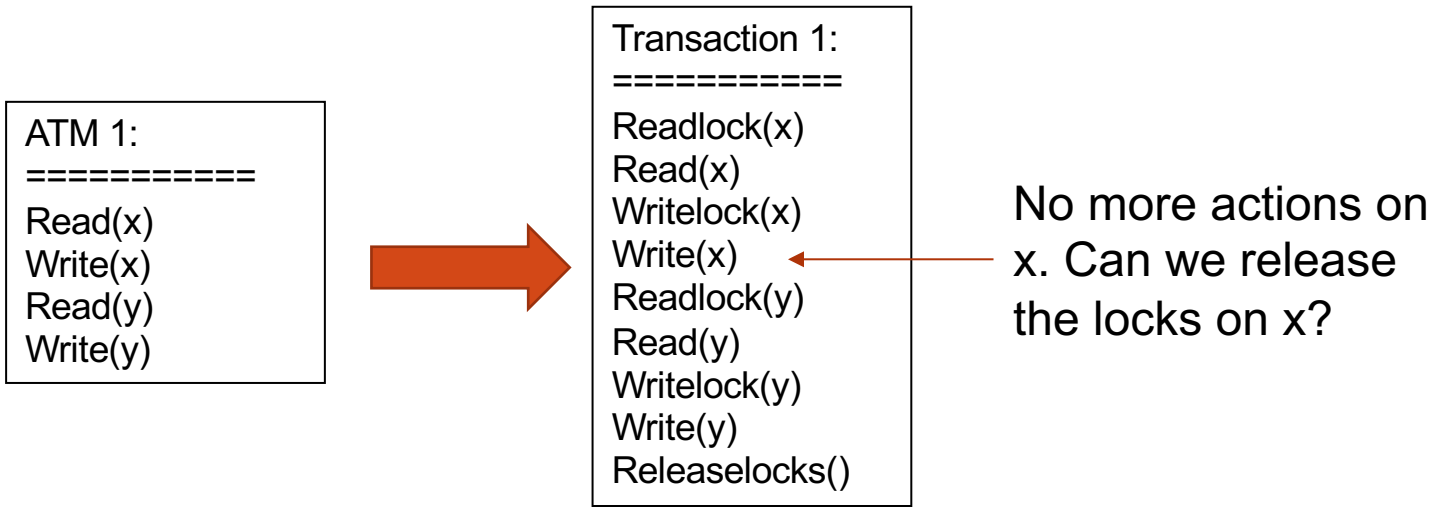
- If an execution is equivalent to a serial execution (in terms of conflicts), this is called a serializable execution
  - A serializable execution gives a correct result, resulting a consistent database state
- Exercise
  - Is the following execution serializable?
    - $R_2(x)\ R_1(x)\ R_1(y)\ W_1(y)\ R_2(y)\ W_2(y)$
    - $R_1(x)\ R_2(x)\ R_1(y)\ R_2(y)\ W_1(y)\ W_2(y)$

How to ensure a correct execution?

- Two phase locking
  - Deadlock
- Other methods [self-study]
  - Preemptive locking
  - Precedence graph and serializability test

Locking

- Make sure no other (active) transaction is having a conflict action at the same time
- Locking
  - There are two types of locks:
    - Readlock and writelock
    - Readlock is required before performing a read action
    - Writelock is required before performing a write action



Lock upgrade from readlock to writelock

- The writelock basically is an upgraded version of readlock
- The transaction that holds the writelock does not need to hold the readlock

- Conflicts between the locks
  - Note: they are on the same variable
  - Note: the two locks are requested by different transactions

	Readlock	Writelock
Readlock	No conflict	Conflict
Writelock	Conflict	Conflict

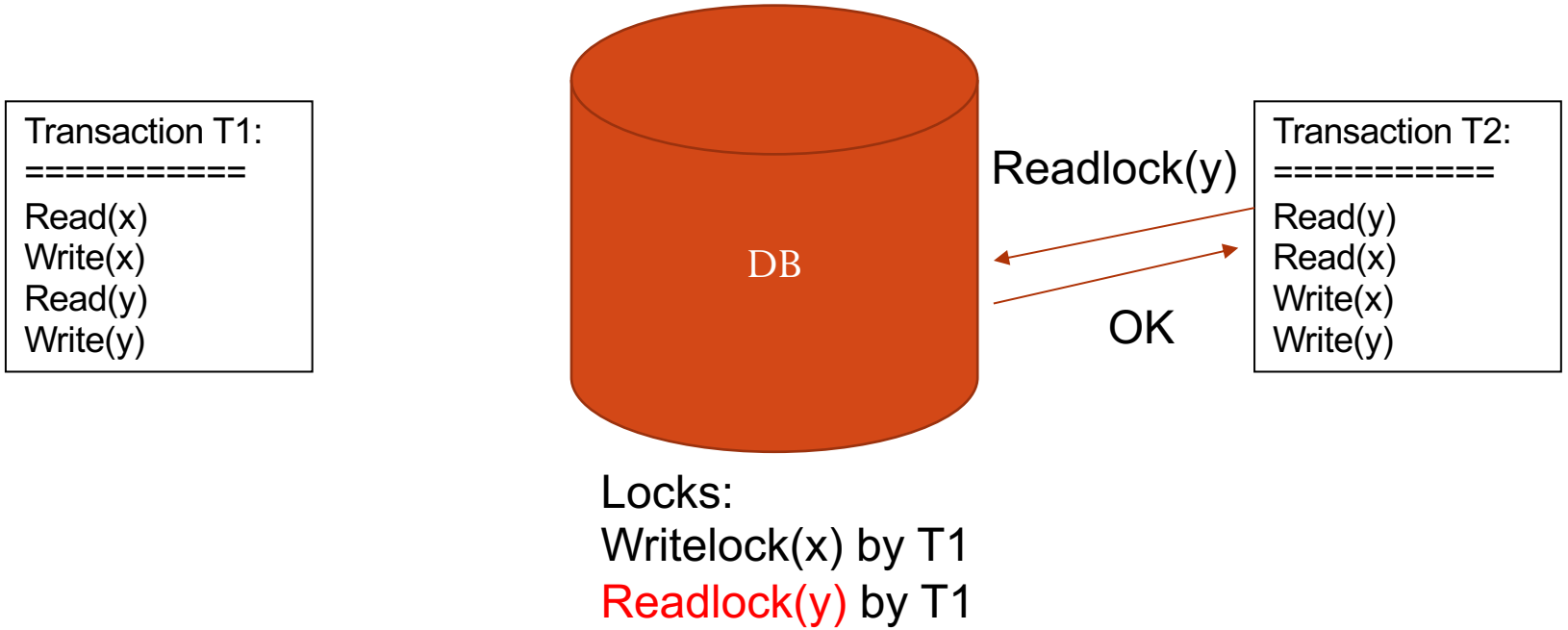
Two-phase locking

- To ensure correct result, the two-phase locking method has two phases:
- Phase 1: growing phase
  - The transaction obtains new locks in this phase
  - No lock is released in this phase
- Phase 2: shrinking phase

The transaction commits

  - The transaction releases obtained locks in this phase
  - No new lock can be obtained in this phase

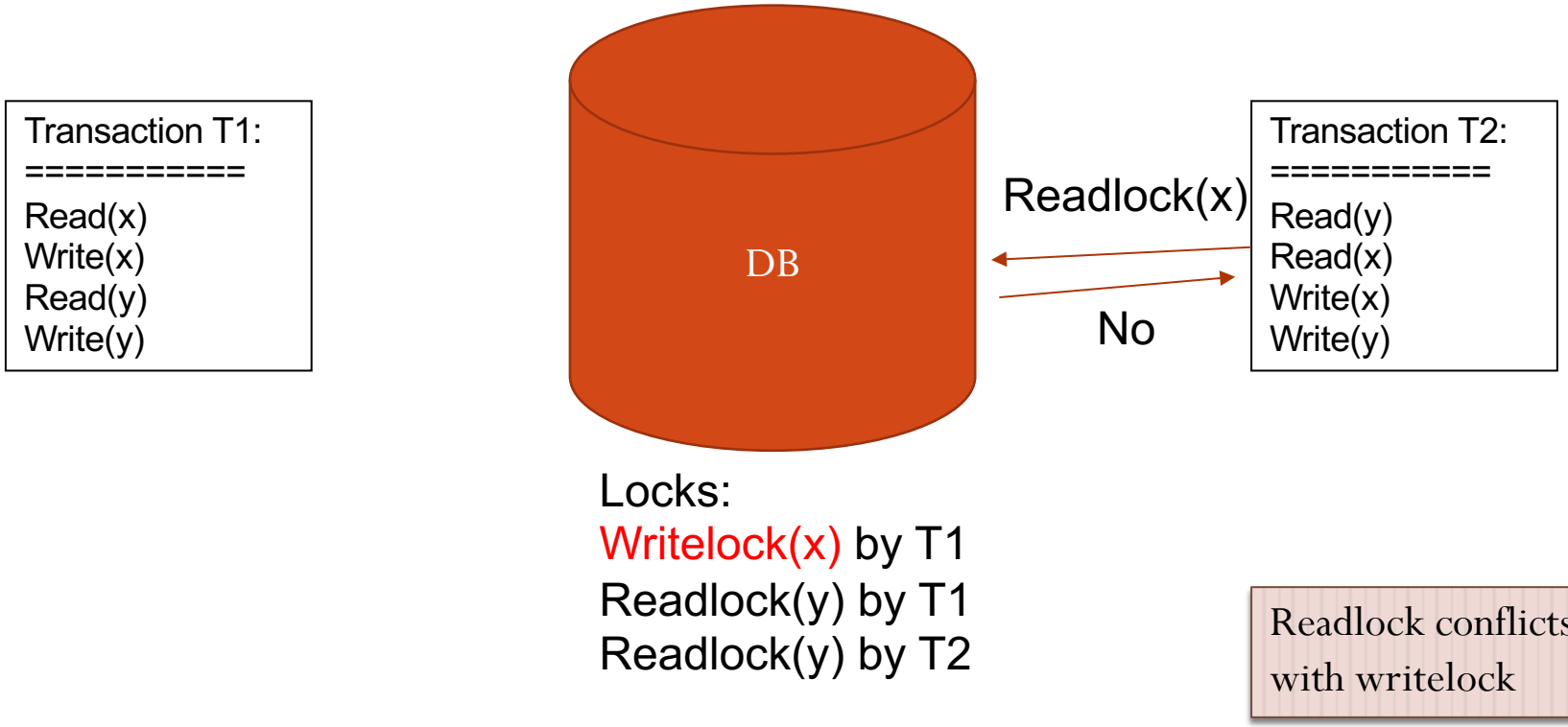
Example



Locks:
Writelock(x) by T1
Readlock(y) by T1

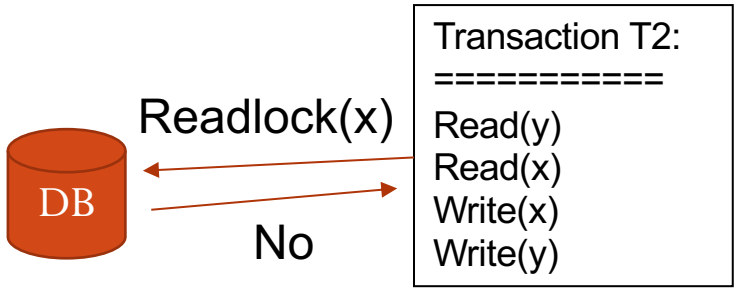
Readlock does not conflict with another readlock

Example

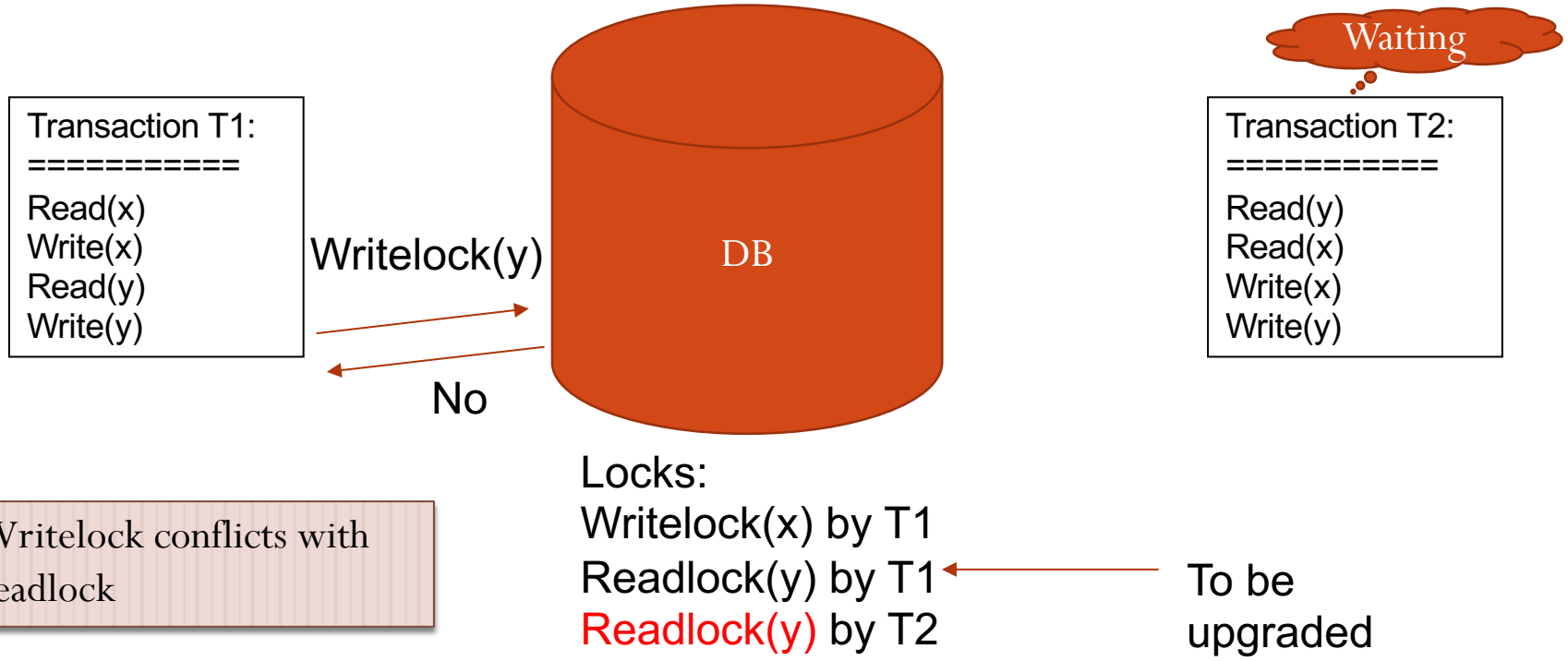

Transaction T1:
==========
Read(x)
Write(x)
Read(y)
Write(y)
DB
Readlock(x)
No
Transaction T2:
==========
Read(y)
Read(x)
Write(x)
Write(y)
Locks:
Writelock(x) by T1
Readlock(y) by T1
Readlock(y) by T2
Readlock conflicts with writelock

What should we do when there is a conflict?

- Option 1: wait
  - Wait for other transactions to release the locks
- Option 2: rollback
  - Undo what the transaction has done
  - Declare the transaction as failed
    - Retry later



Example



Writelock conflicts with readlock

Example

Transaction T1:
==========
Read(x)
Write(x)
Read(y)
Write(y)

Transaction T2:
==========
Read(y)
Read(x)
Write(x)
Write(y)

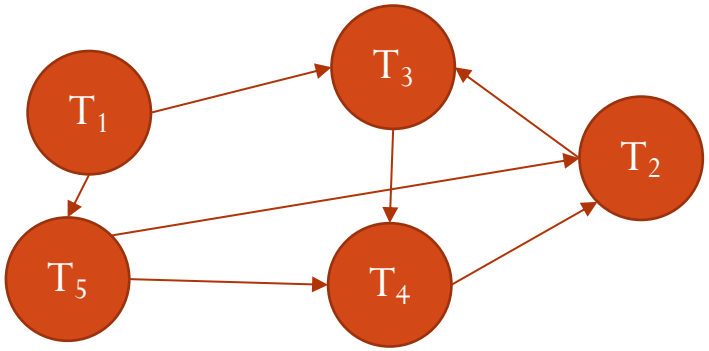
When will the transactions complete?
This is called a deadlock

Locks:
Writelock(x) by T1
Readlock(y) by T1
Readlock(y) by T2

Deadlock

- We use the following precedence graph to represent that T_1 is waiting for a lock that is obtained by T_2

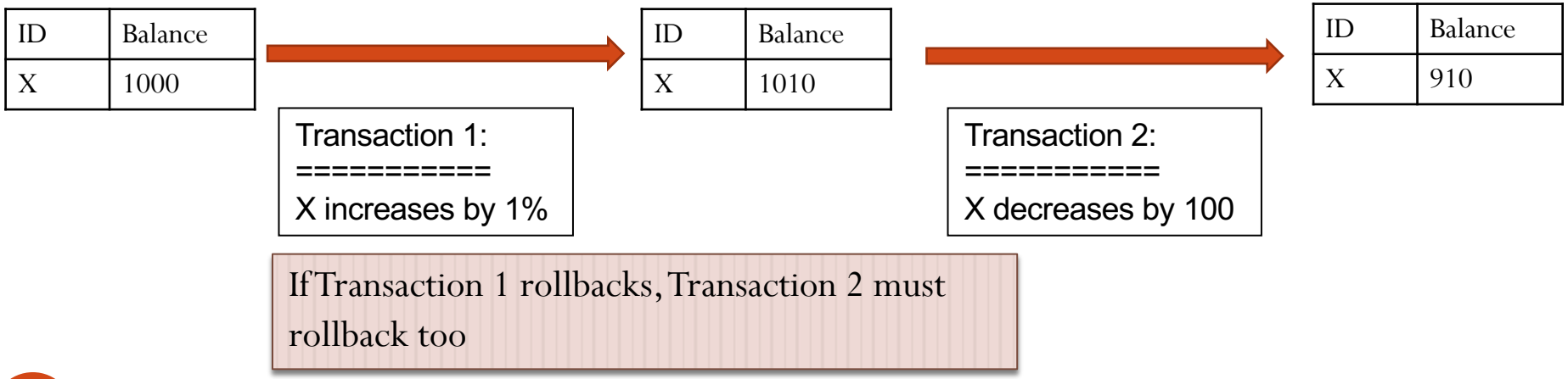
- A deadlock means that there is a cycle in the graph



Is there a deadlock?

Rollback

- Do you understand why there is a rollback option here?
  - To resolve a deadlock
- Cascading rollback
  - Some other transactions may have read the updated values of this transactions
  - They must be rollbacked as well



Self-study [Concurrent access in your program]

- A program by default can utilize one core of your CPU…
- Multithreading / multiprocessing in python
  - https://www.tutorialspoint.com/python/python_multithreading.htm
- Locking in python
  - https://www.pythontutorial.net/python-concurrency/python-threading-lock/

Database recovery

- Why does the database require recovery?
  - Cancelled transaction
    - The user press the cancel
  - System failure
    - Disk failure, system crash, system errors

- Log-based recovery is the mainstream method

T1	$w_1(x)$	T2	$w_1(a)$	$w_1(y)$	T1 commit	T3	$w_3(x)$	…

- Core principle
  - Save to the log before writing to the database system

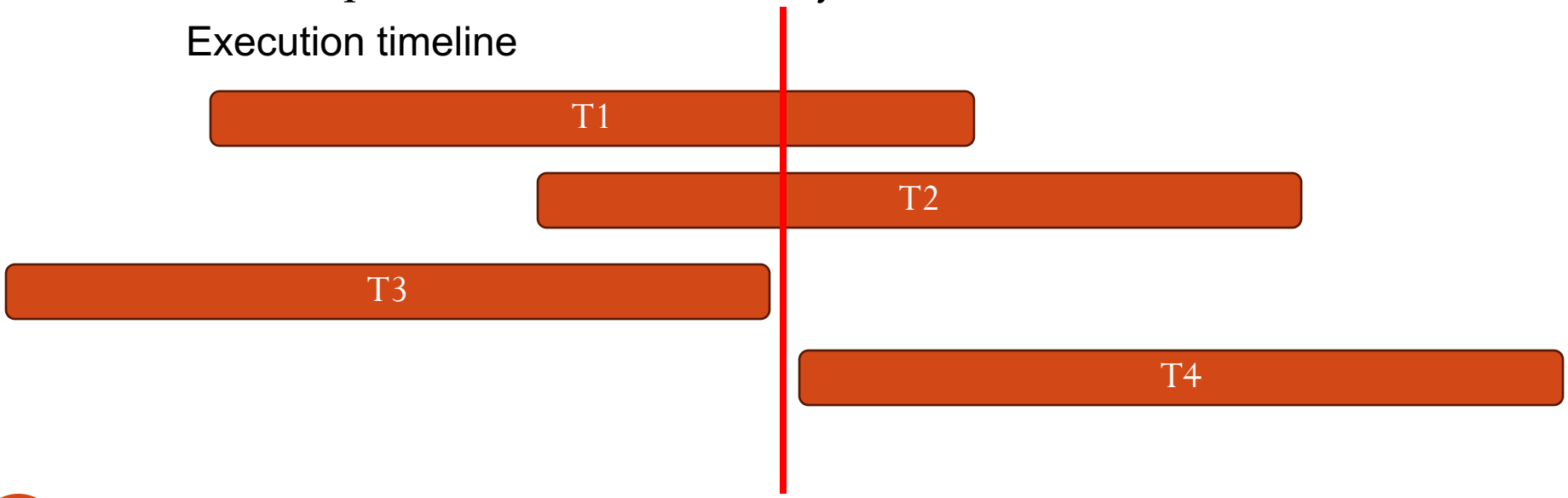
Challenges

- How to perform backup without stopping the database?
- How to make sure all contents before the crash can be recovered?

Discussions

- What do we put in the log?
  - The entire database?
    - Let's not consider efficiency now. Does it work?

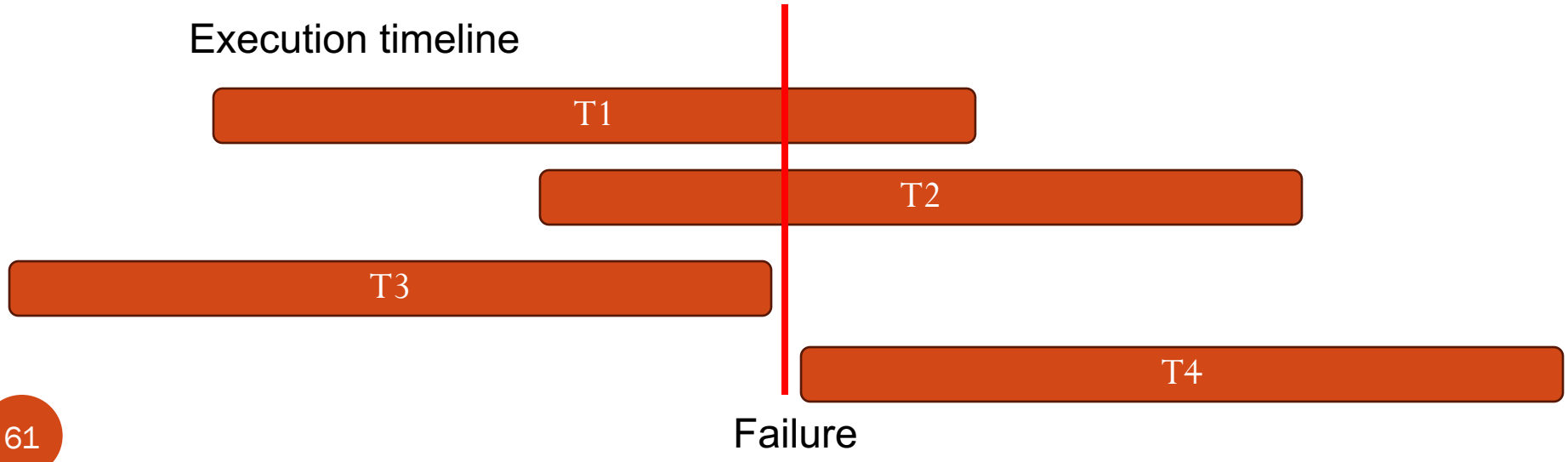
- The snapshot of a database may be inconsistent



At this point, the database may contain partial updates from T1 and T2

Recovery of failure

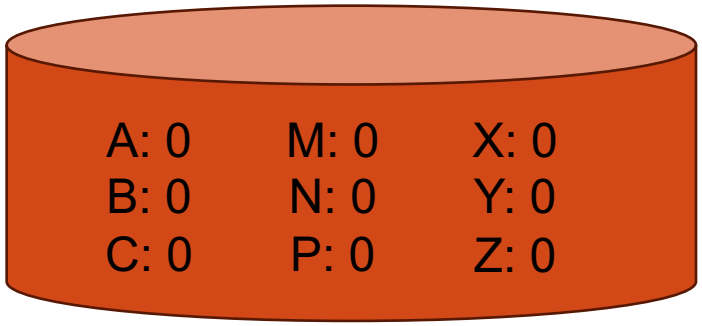
- After failure happens, all working transactions cannot communicate with the database
  - They will stop processing and rollback, and report failure to the users
- Only the effect of T3 should stay in the database after recovery



Contents of the log

- Clear record about the start of a transaction
- Clear record about the commit of a transaction
- The update contents
  - Many ways are feasible
    - Before-after values (e.g., x, 10, 15 --- x changes from 10 to 15)
      - More storage required
      - More straight-forward recovery (compared to the method below)
    - Delta values (e.g., x, +5 --- x increases by 5)
      - Less storage required
      - Need a mechanism to tell if the value on the harddisk is before or after the update

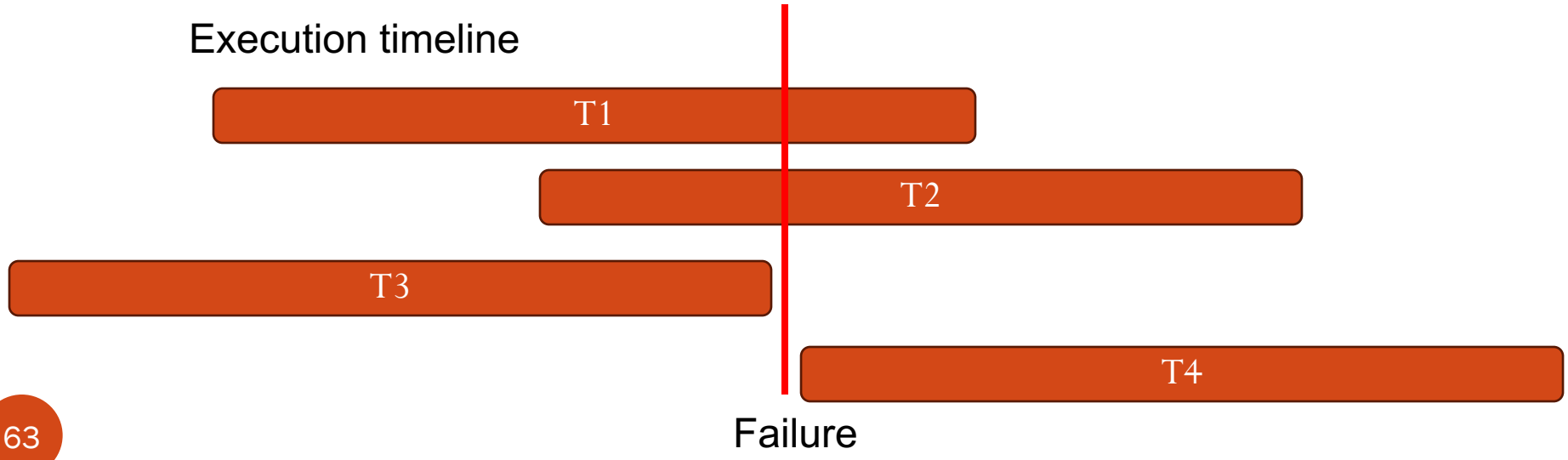
Example logs



Initial database

Q: What are the values in the database after recovery?

T3	T3, X, 12	T1	T3,Y, 10	T2	T1, A, 60	T1, B, 30	T3, Z, 8	T2, M, 100	T3 commit	Crash



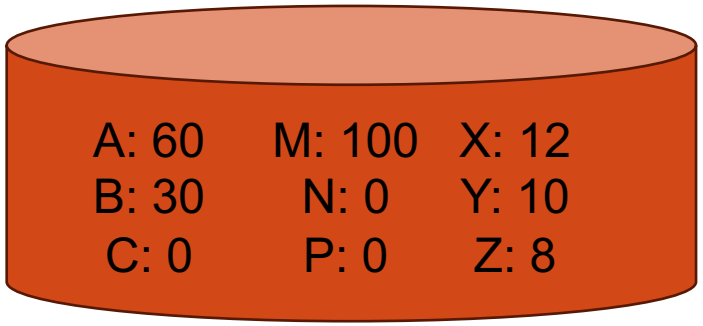
Meaning of the log entries

- T3: Start of the transaction 3
- T3, X, 12:T3 sets X to 12
- T3 commit: End of the transaction 3 (commit)

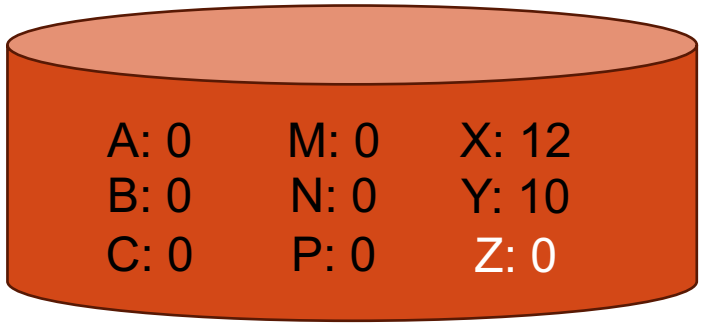
T3	T3, X, 12	T1	T3,Y, 10	T2	T1, A, 60	T1, B, 30	T3, Z, 8	T2, M, 100	T3 commit	Crash

Recovery actions

- Undo incomplete transactions
- Redo committed transactions



Database when failed

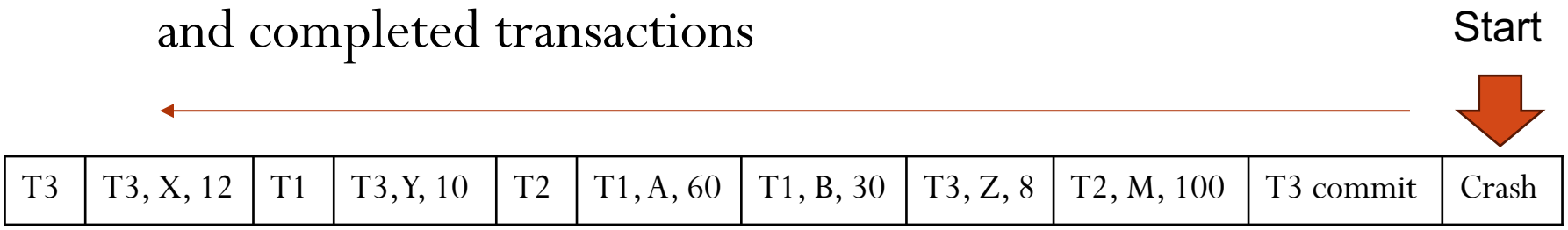


Database when failed and some data are still in memory but not on disk

T3	T3, X, 12	T1	T3,Y, 10	T2	T1, A, 60	T1, B, 30	T3, Z, 8	T2, M, 100	T3 commit	Crash

Recovery algorithm

- Start from the end of the log
- Scan the log backwards to discover incomplete transactions and completed transactions



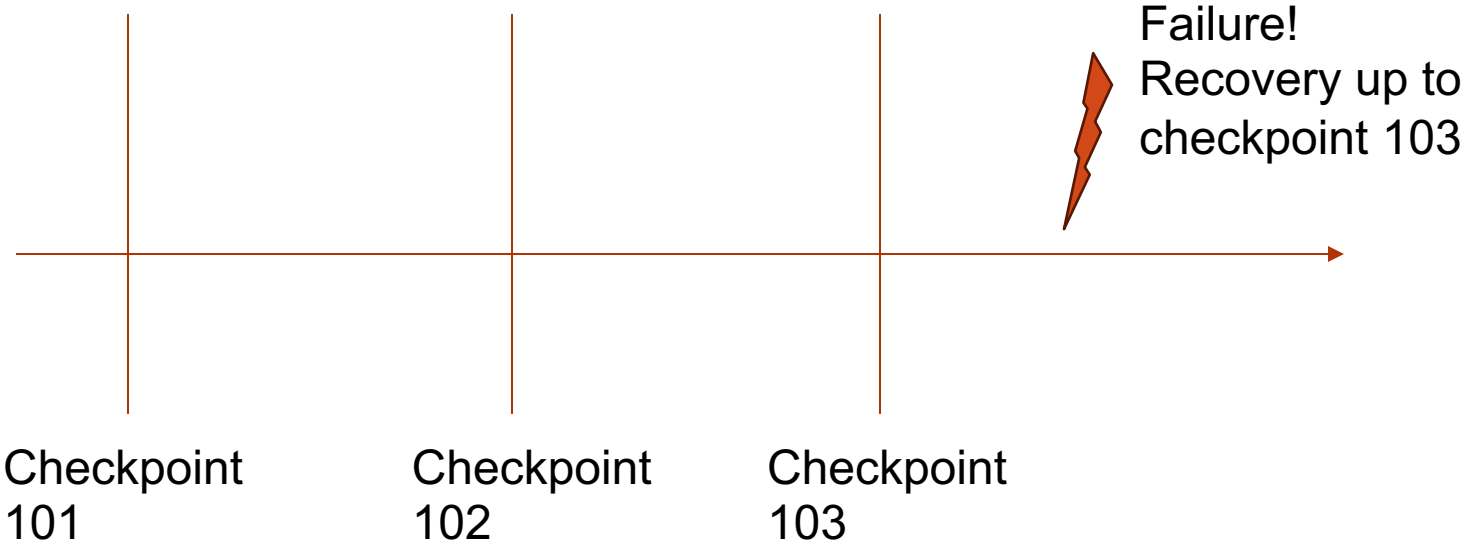
- Q: When do we stop?
  - We need to make sure all completed transactions are discovered
    - Scan until the starting point of the log!
    - Meaning we need to scan the entire log! (What if the log starts from 30 years ago??!!)

Database checkpoints

- Blocking checkpoints
- Fuzzy checkpoint (Non-blocking checkpoints)

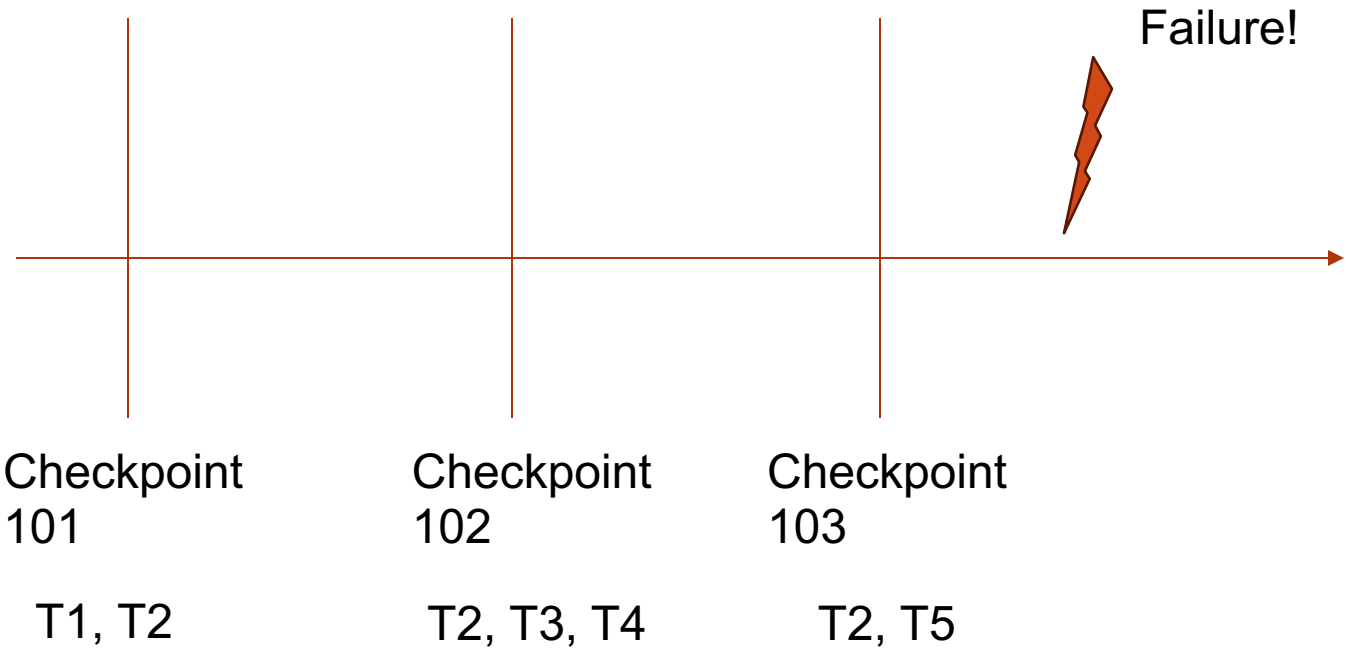
Blocking checkpoints

- We stop all the updates to the database
- Create a snapshot of the database
- All recovery only needs to scan up to the closest checkpoint



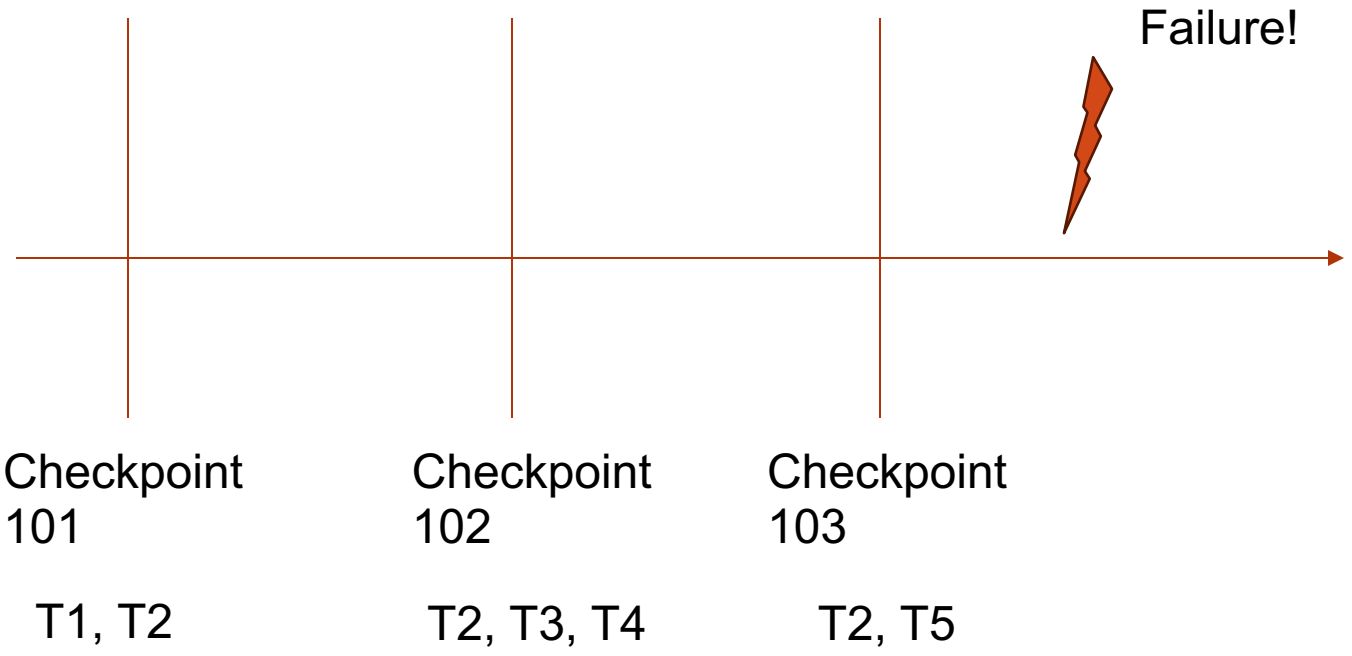
Fuzzy checkpoints

- We do NOT stop the updates to the database
- Create a snapshot of the database and keep records about what transactions are running



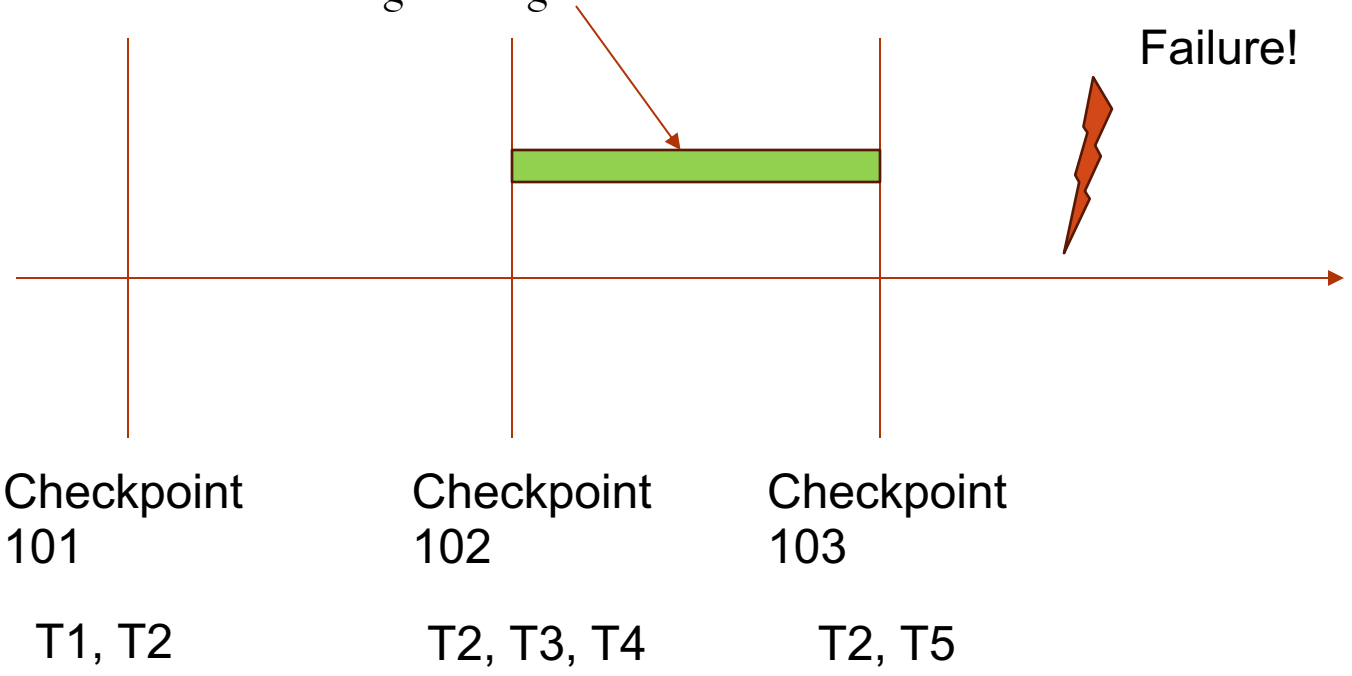
Example recovery

- Scan the log backwards until Checkpoint 103
  - All committed transactions before Checkpoint 103 are already in the checkpoint image
  - Only need to chase further for T2 and T5



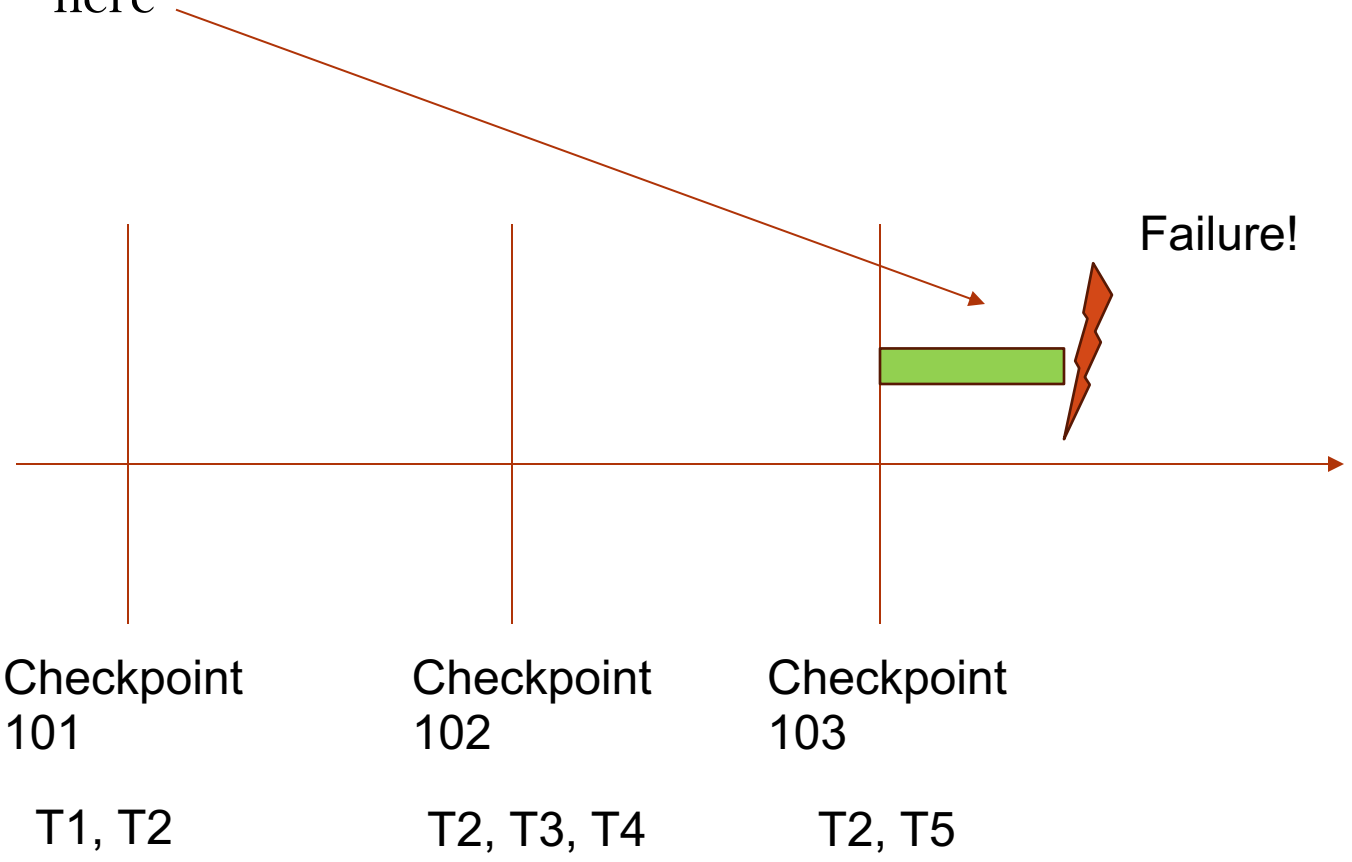
Questions

- Say, we reach checkpoint 102 (recall we are scanning backward from the failure point.
  - Do we need to chase for T2 / T3 / T4?
  - T2: Maybe yes (check next slide)! T3: No! T4: No!
- Where does T5 start?
  - Somewhere in this green region



Questions

- Is T2 / T5 committed?
  - Depends on whether we see T2 / T5 commit in the latest log here

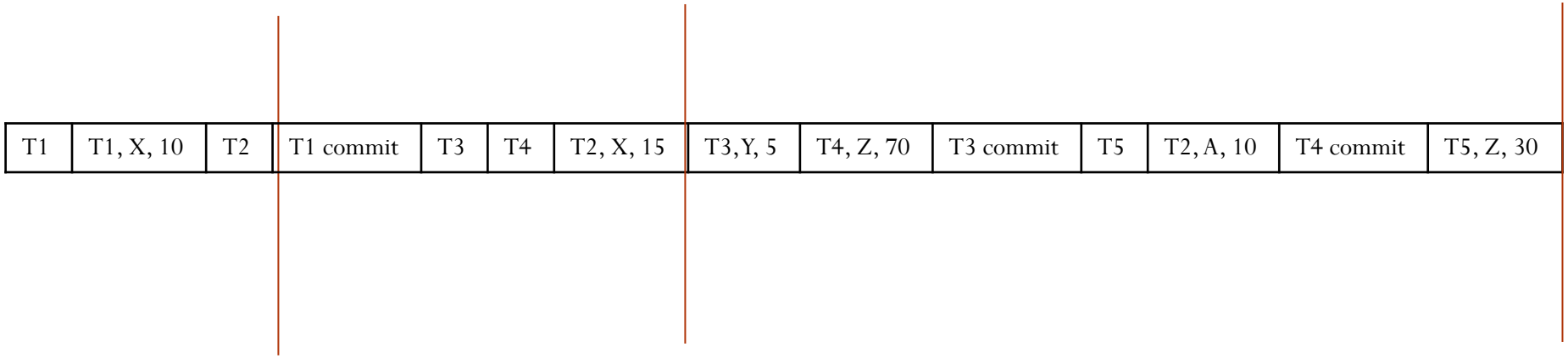


Question

- What is the worst-case recovery cost when using fuzzy checkpoints?
  - O(n). Scan backwards until the first log entry.
  - But this is almost impossible. If this is the case, either the log is very short or the transaction is unreasonably long

Example

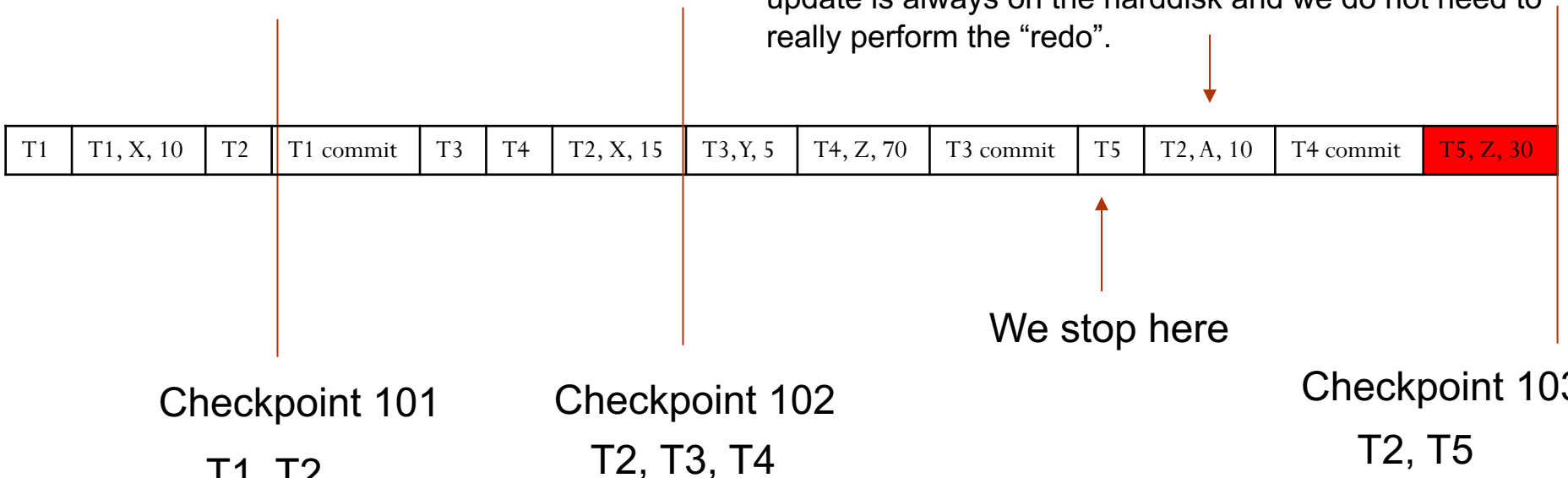
- How to recover from the crash?

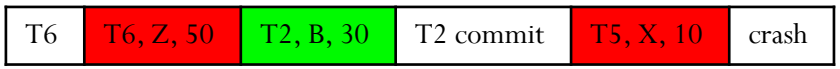


T6	T6, Z, 50	T2, B, 30	T2 commit	T5, X, 10	crash

Example

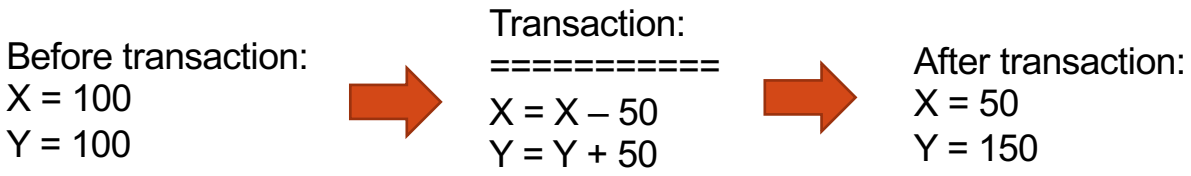
- Undo incomplete transactions (marked in red)
- Redo committed transactions (marked in green)
  - All actions before the checkpoints are guaranteed to be on the harddisk





ACID property of database

- Atomicity
  - Transaction is handled in an "all-or-nothing" manner
    - Either the entire transaction is done, or none is done
- Consistency
  - Any transaction will bring the database from one valid state to another



Very important!

ACID property of database

- Isolation
  - It **appears like** transactions are executed independently
- Durability
  - Once a transaction is committed, its effect stays in the database forever

Very important!