Transaction: a set of <u>logical</u> operations

$R(A)$

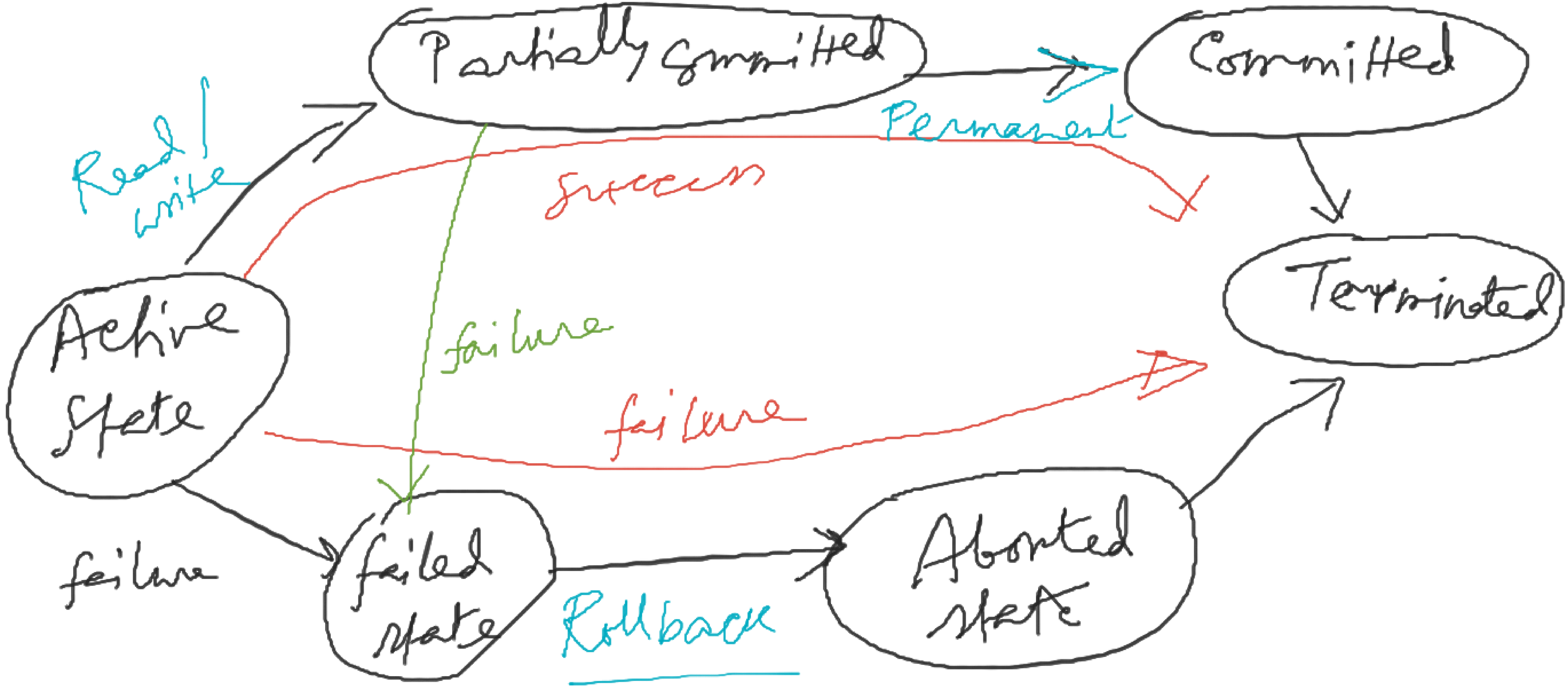
$A = A - 100$

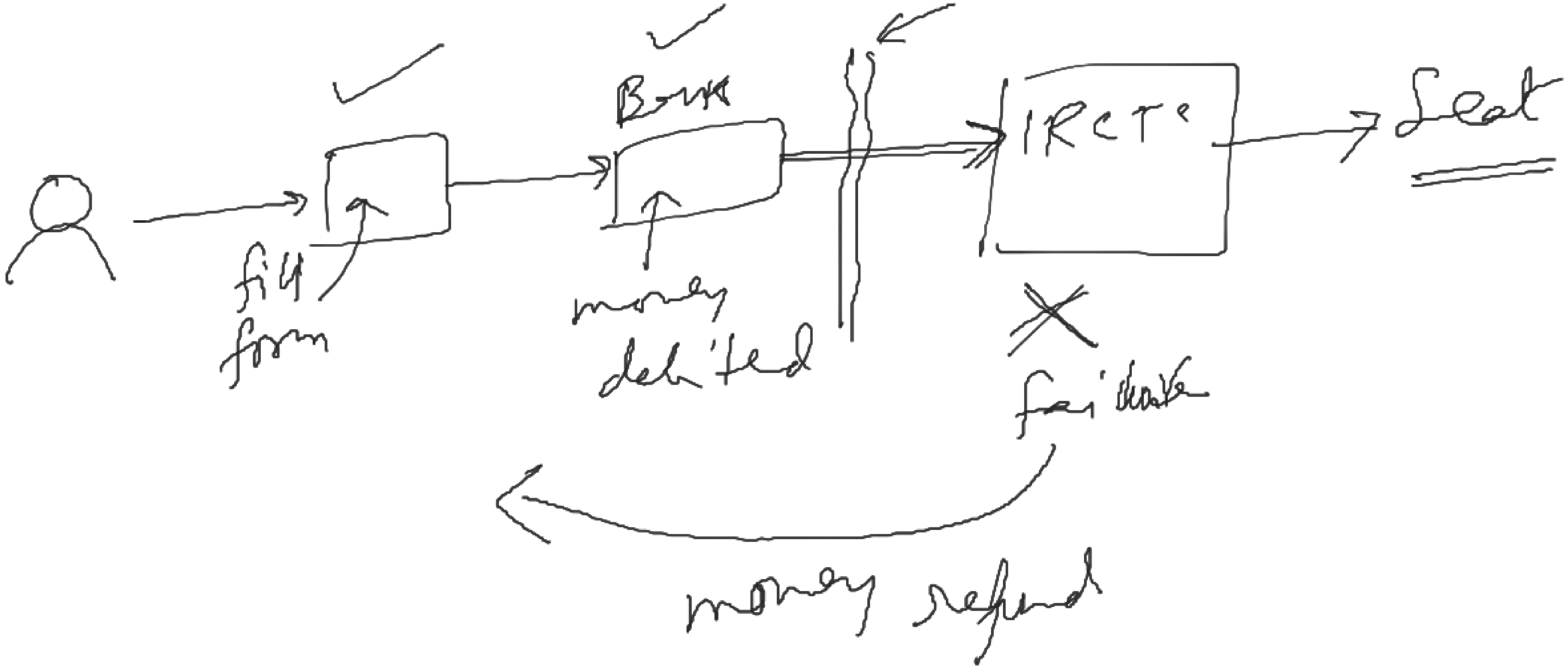
$W(A)$ ×

$R(B)$

$B = B + 100$

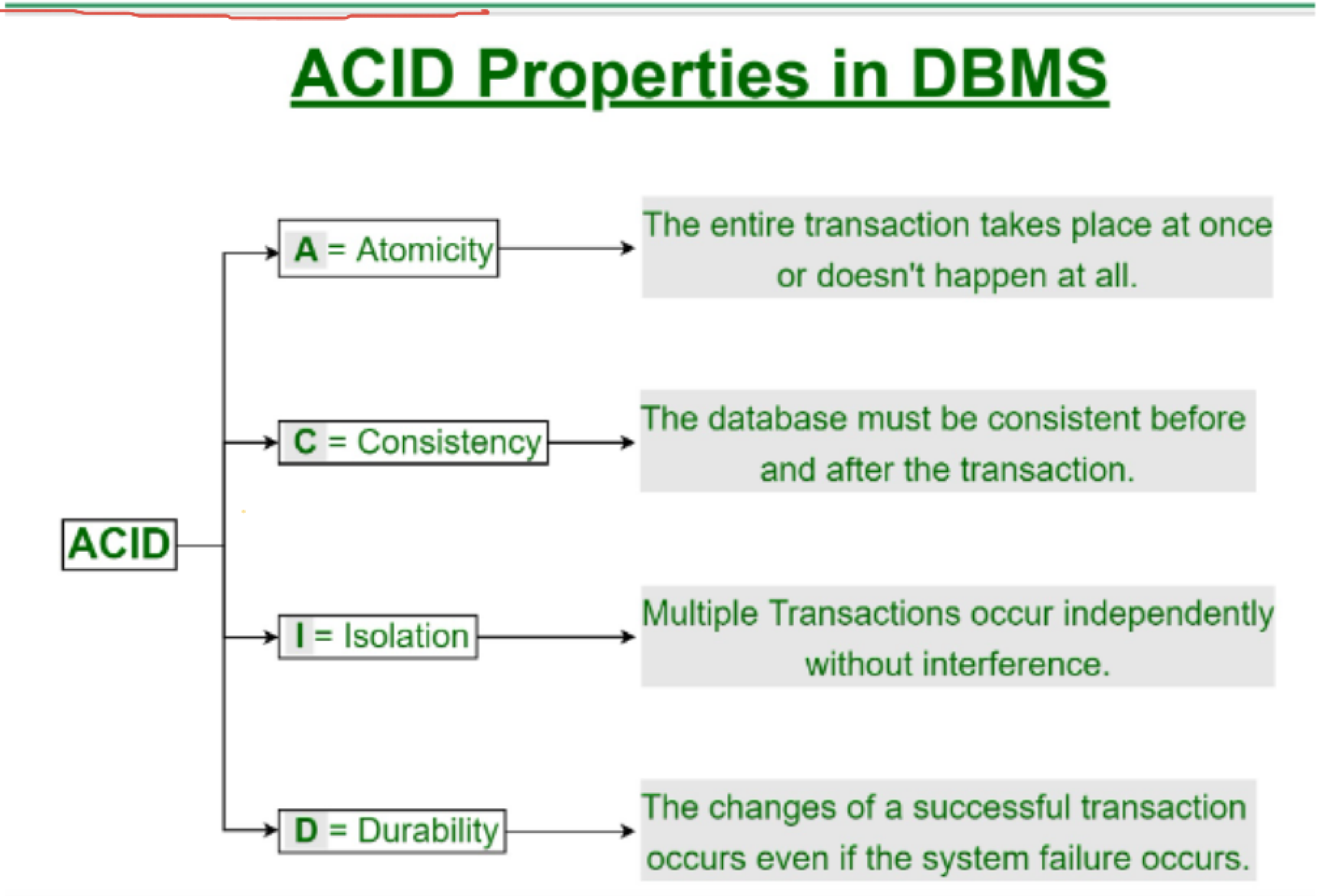
$W(B)$


Partially committed
Committed
Read write
Permanent
success
Terminated
Active State
failure
failure
failure
failed state
Rollback
Aborted state



Bank
IRCTC
Seat
fill form
money debited
failure
money refund

ACID properties: 4 important properties of a transaction
Atomicity
Consistency
Isolation
Durability



taken by: https://www.geeksforgeeks.org/acid-properties-in-dbms/

- Atomicity ensures
  - Either all the operations are executed successfully or none of them
  - It involves two operations:
    - Abort: If a transaction aborts, changes made to database are not visible
    - Commit: If a transaction commits, changes made are visible

Example:

$T = \{R(A); A = A + 100; W(A);\}$
This transaction is about: credit \$100 to Account A.
If all 3 operations are executed successfully, commit operation needs to be performed
If T fails after Operation 2, rollback operation needs to be performed

- Concurrent execution of database transactions in a multi-user system environment means (more than 1 user can access the same database at the same time)

Consistency

- Consistency ensures
  - Concurrent transactions must not leave the database in an inconsistent state

Example:
T_1 = {R(A); A = A - 500; W(A);} T_2 = {R(A); A = A + 1000; W(A);}
T_1 and T_2 are two concurrent transactions.
Suppose Initially, A = 1000.
T_1 reads A(1000) and stores it in its local buffer. Then T_2 reads A(1000) and also stores it in its local buffer.
T_1 performs A = A - 500 and 500 is stored in local buffer of T_1. Then T_2 performs A = A + 1000 and 2000 is stored in buffer of T_2.
T_1 writes the value from its buffer back to database. Then T2 writes the value from its buffer back to database.
Finally, A = 2000 in the database

It leads to an inconsistent state in a database

- Isolation ensures
  - Result of a transaction is not be visible to others before it is committed

Example:

failed

T_1

T_2

$T_1 = \{R(A);\ A = A - 500;\ W(A);\}$ → 500

$T_2 = \{R(A);\ A = A + 1000;\ W(A);\}$ 1500

Suppose Initially, A = 1000.

After successful execution of T_1, A = 500

After successful execution of T_2, A = 1500

What happens T_1 fails just before committing?

We need to rollback T_2 as well, since T_2 used the value of A generated by T_1

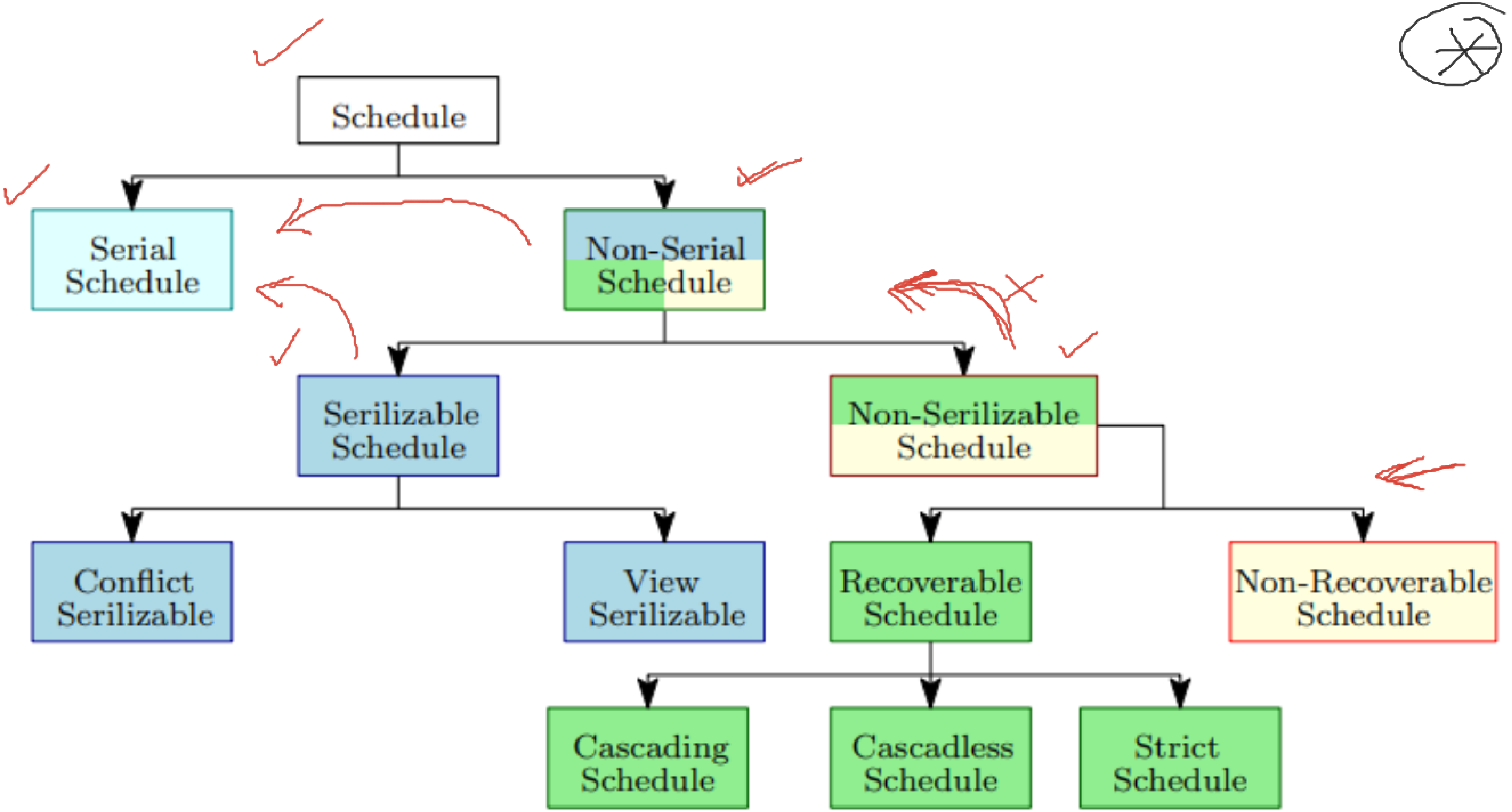
Conclusion: A transaction (T_1) results should not made visible to other (T_2) transactions before it commits

Durability

- Durability ensures
  - Once database has committed a transaction, the changes made by the transaction are permanent

Example:

$T = \{R(A); A = A - 500; W(A);\}$
Suppose Initially, $A = 1000$.
After successful execution of T, $A = 500$
The changes of A should be permanent in the database
To ensure durability, multiple copies of database are stored at different locations


Schedule
Serial Schedule
Non-Serial Schedule
Serilizable Schedule
Non-Serilizable Schedule
Conflict Serilizable
View Serilizable
Recoverable Schedule
Non-Recoverable Schedule
Cascading Schedule
Cascadless Schedule
Strict Schedule

Serial Schedule

- Transactions are executed in non-interleaved manner
- No transaction starts until a running transaction has ended

T_1	T_2
R(A) R(B) W(B) R(C) W(A)	
	R(A) R(B) W(A) R(C) W(C)

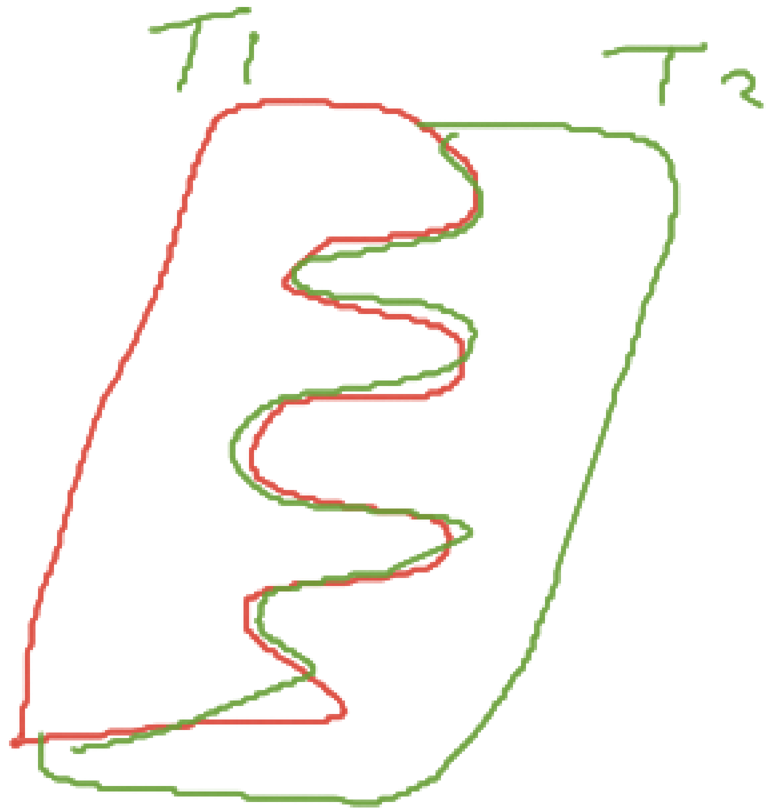
Figure: Example of a serial schedule

Non-Serial Schedule

- Operations of multiple transactions are interleaved
- The end result has to be correct and same as the serial schedule AIM
- One transaction does not wait for another to complete
- Provide benefit of concurrent transaction
- May lead to concurrency problem

	T_1	T_2	
8 AM	R(A)		
8:01	R(B)		
		R(A)	8:02
	W(B)		
		R(B)	
	R(C)		
		W(A)	
		R(C)	
		W(C)	

Figure: Example of a non-serial schedule



Seralizable Schedule

- A non-serial schedule is said to be serializable only if it is equivalent to a serial schedule, for an n number of transactions
- It helps in improving resource utilization and CPU throughput

LHS RHS

Non-Serial Schedule		Equivalent Serial Schedule	
T_1	T_2	T_1	T_2
R(A)		R(A)	
R(B)		R(B)	
	R(A)	W(B)	
W(B)		R(C)	
	R(B)		R(A)
R(C)			R(B)
	W(A)		W(A)
	R(C)		R(C)
	W(C)		W(C)

Figure: Example of a seralizable schedule

LHS S_1

$R_1(A)$
$R_1(B)$
$R_2(A)$
$W_1(B)$
$R_2(B)$
$R_1(C)$
$W_2(A)$
$R_2(C)\ W_2(C)$

Conflict Seralizable Schedule

Conflicting operations

Two operations are said to be conflicting if **all** the following conditions are satisfied:

- They belong to different transactions
- They operate on the same data item
- At Least one of them is a **write** operation

Example:

- $(R_1(A), W_2(A))$: Conflicting operations. They belong to two different transactions on same data A and one of them is write operation
- $(W_1(A), W_2(A))$: Conflicting operations pair
- $(W_1(A), R_2(A))$: Conflicting operations pair
- $(R_1(A), W_2(B))$: Non-conflicting pair, as they operate on different data items
- $((W_1(A), W_2(B))$: Non-conflicting pair

Conflict Seralizable Schedule (CSS)

Conflict Serializable

A schedule is called conflict serializable if it can be transformed into a serial schedule by swapping non-conflicting operations

Example: Consider two following transactions

$T_1 : R_1(A), W_1(A), R_1(B), W_1(B)$

$T_2 : R_2(A), W_2(A), R_2(B), W_2(B)$

S_1 is CSS

Consider following schedule:

$S_1 : R_1(A), W_1(A), R_2(A), W_2(A), R_1(B), W_1(B), R_2(B), W_2(B)$

The schedule is conflict serializable

$S_{11} : R_1(A), W_1(A), R_2(A), W_2(A), R_1(B), W_1(B), R_2(B), W_2(B)$

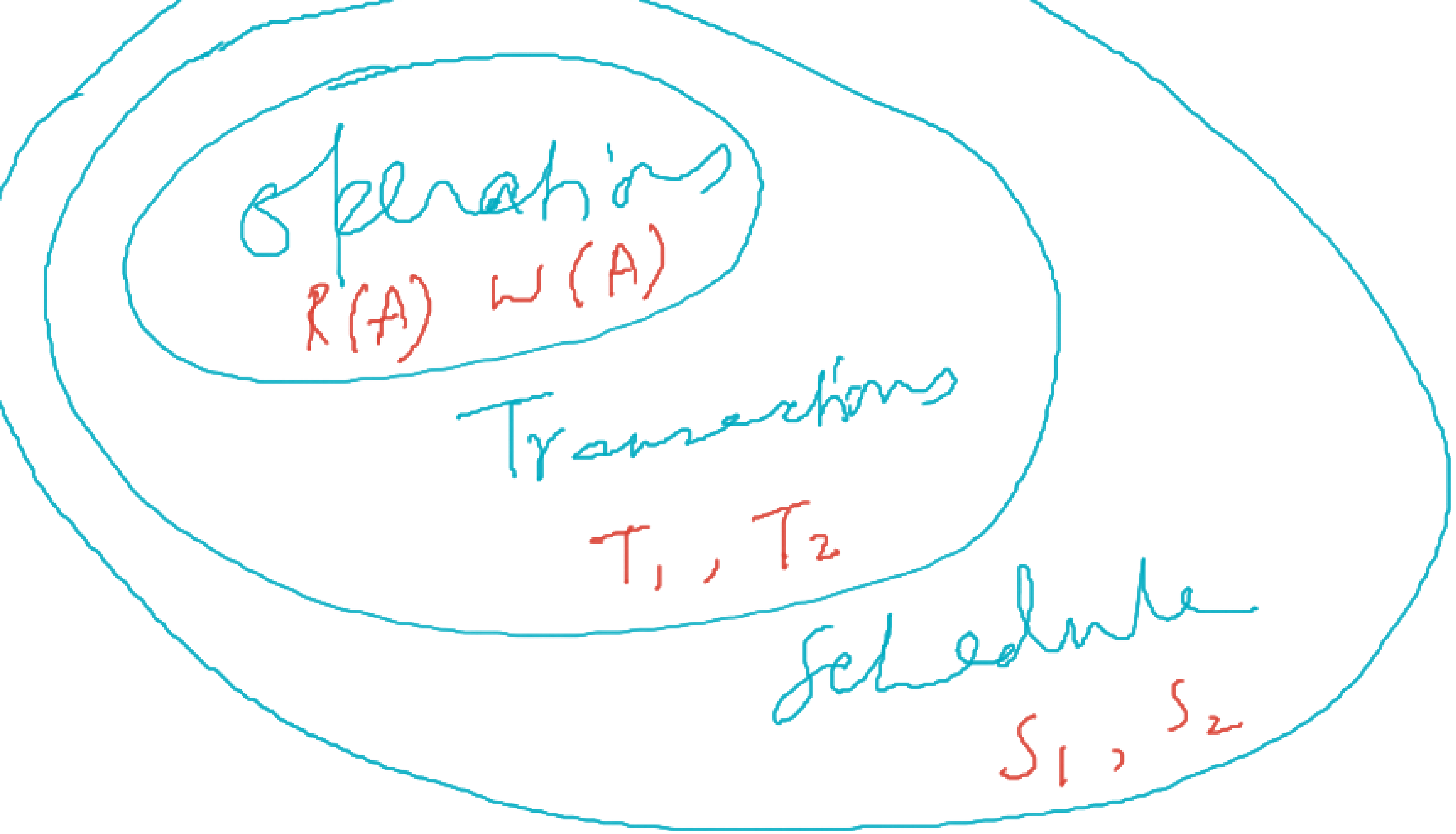
$S_{12} : R_1(A), W_1(A), R_2(A), R_1(B), W_2(A), W_1(B), R_2(B), W_2(B)$

$S_{13} : R_1(A), W_1(A), R_1(B), R_2(A), W_2(A), W_1(B), R_2(B), W_2(B)$

$S_{14} : R_1(A), W_1(A), R_1(B), R_2(A), W_1(B), W_2(A), R_2(B), W_2(B)$

$S_{15} : R_1(A), W_1(A), R_1(B), W_1(B), R_2(A), W_2(A), R_2(B), W_2(B)$

S_{15} is the equivalent serial schedule of S_1


operations
R(A) W(A)
Transactions
T1, T2
Schedule
S1, S2

Conflict Equivalent

Conflict Equivalent

Two schedules are said to be conflict equivalent when one can be transformed to another by swapping non-conflicting operations

Example: Consider two following transactions
$T_1 : R_1(A), W_1(A), R_1(B), W_1(B)$
$T_2 : R_2(A), W_2(A), R_2(B), W_2(B)$
Consider following two schedules:
$S_1 : R_1(A), W_1(A), R_2(A), W_2(A), R_1(B), W_1(B), R_2(B), W_2(B)$
$S_2 : R_1(A), W_1(A), R_1(B), W_1(B), R_2(A), W_2(A), R_2(B), W_2(B)$
S_1 and S_2 are conflict equivalent