DATABASE MANAGEMENT SYSTEMS

Authors – Raghu Ramakrishna, Johannes Gehrke

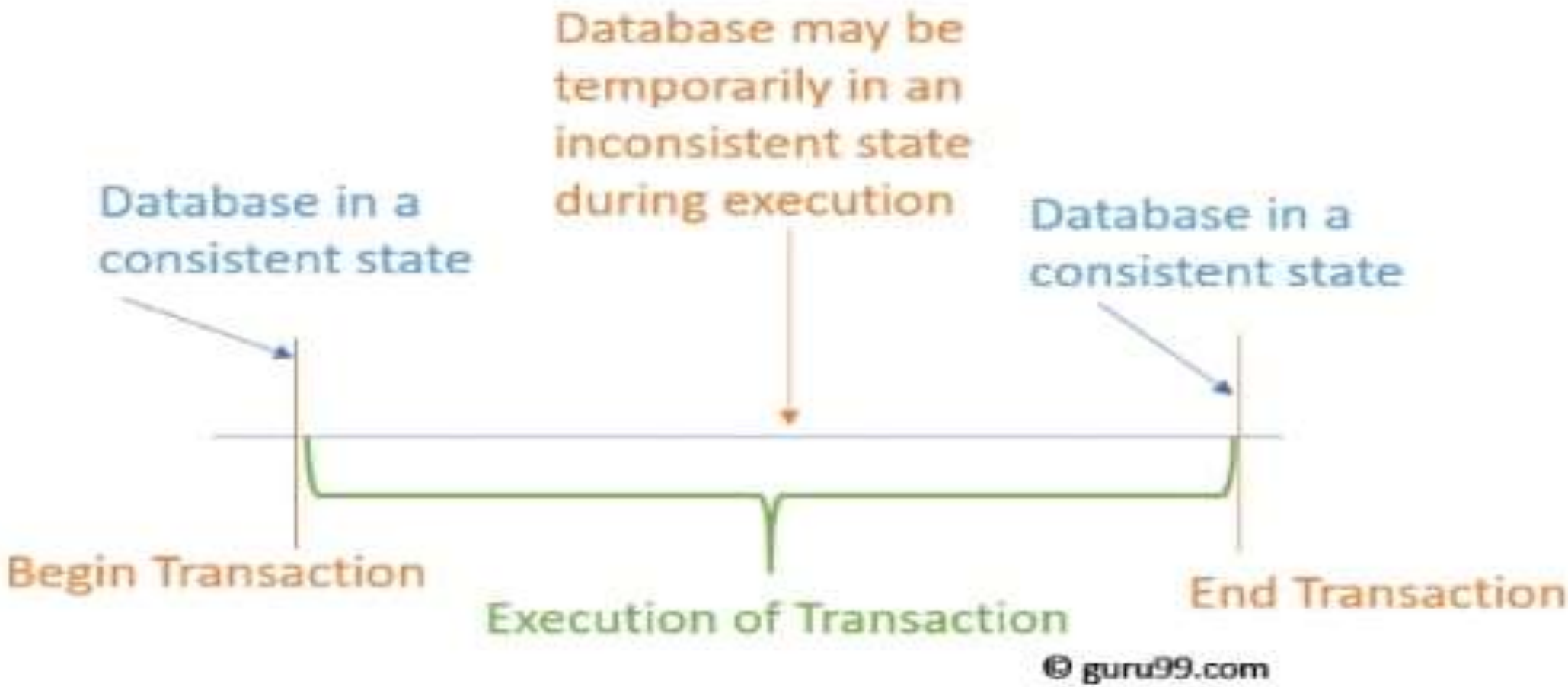
Edition – 3

UNIT - 4

- ACID properties – Concurrent Executions-Conflict serializability-view serializability - Concurrency Control: Lock – Based Protocols-Deadlock Handling-Timestamp Based Protocols Multiple Granularity. Advance Recovery systems- ARIES, Log, the Write–ahead Log Protocol, Checkpointing, and Recovering from a System Crash. Primary and Secondary Indexes – Index data structures – Hash-Based Indexing – Tree base Indexing – B+ Trees: A Dynamic Index Structure

Transaction

- The transaction is a set of logically related operation. It contains a group of tasks. A transaction is an action or series of actions.
- All types of database access operation which are held between the beginning and end transaction statements are considered as a single logical transaction in DBMS.
- During the transaction the database is inconsistent. Only one the database is committed the state is changed from one consistent state to another.
- If the database operations do not update the database but only retrieve data, this type of transaction is called a read – only transaction.



Database Transaction

Example:

- Suppose an employee of bank transfers Rs 800 from X's account to Y's account.
- This small transaction contains several low-level tasks:

X's Account

```
Open_Account(X)
Old_Balance = X.balance
New_Balance = Old_Balance - 800
X.balance = New_Balance
Close_Account(X)
```

Y's Account

```
Open_Account(Y)
Old_Balance = Y.balance
New_Balance = Old_Balance + 800
Y.balance = New_Balance
Close_Account(Y)
```

Operations of Transaction

- Read(X): Read operation is used to read the value of X from the database and stores it in a buffer in main memory.
- Write(X): Write operation is used to write the value back to the database from the buffer.

Let's take an example to debit transaction from an account which consists of following operations:

1. R(X);
2. X = X - 500;
3. W(X);

- Let's assume the value of X before starting of the transaction is 4000.

  1) R (X) ; 2) X= X-500; 3) W(X);
- The first operation reads X's value from database and stores it in a buffer. // X= 4000
- The second operation will decrease the value of X by 500. So buffer will contain 3500. // X=4000-500 => 3500
- The third operation will write the buffer's value to the database. So X's final value will be 3500.
- But it may be possible that because of the failure of hardware, software or power, etc. that transaction may fail before finished all the operations in the set.

For example:

If in the above transaction, the debit transaction fails after executing operation 2 then X's value will remain 4000 in the database which is not acceptable by the bank.

To solve this problem, we have two important operations:

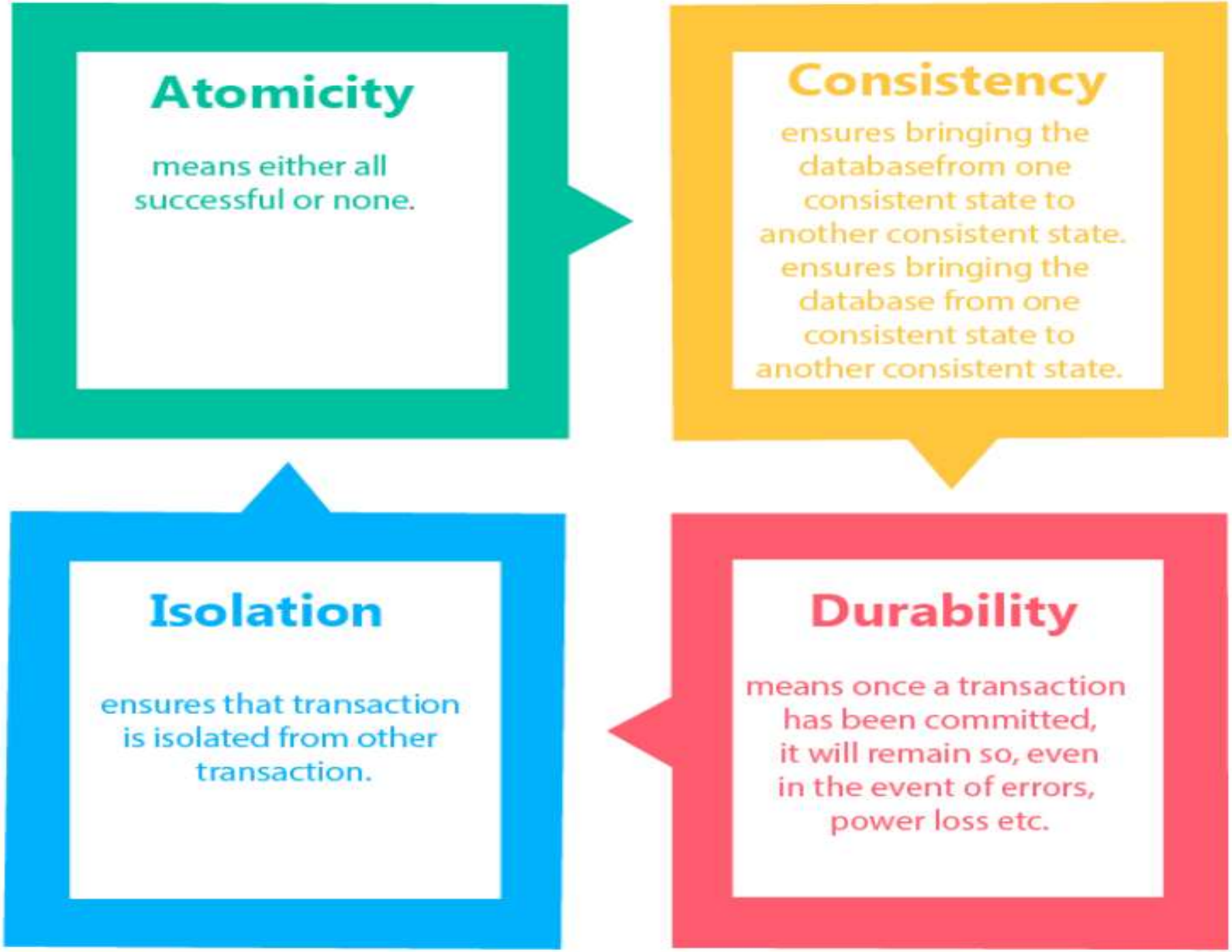
- Commit: It is used to save the work done permanently.
- Rollback: It is used to undo the work done.

Transaction Property

- The transaction has the four properties. These are used to maintain consistency in a database, before and after the transaction.

ACID Properties:

- Atomicity
- Consistency
- Isolation
- durability


Atomicity
means either all successful or none.
Consistency
ensures bringing the databasefrom one consistent state to another consistent state. ensures bringing the database from one consistent state to another consistent state.
Isolation
ensures that transaction is isolated from other transaction.
Durability
means once a transaction has been committed, it will remain so, even in the event of errors, power loss etc.

Atomicity

- It states that all operations of the transaction take place at once if not, the transaction is aborted.
- There is no midway, i.e., the transaction cannot occur partially. Each transaction is treated as one unit and either run to completion or is not executed at all.

Atomicity involves the following two operations:

- **Abort**: If a transaction aborts then all the changes made are not visible.
- **Commit**: If a transaction commits then all the changes made are visible

Example: Let's assume that following transaction T consisting of T1 and T2. A consists of Rs 600 and B consists of Rs 300. Transfer Rs 100 from account A to account B.

T1	T2
Read(A) A:= A-100 Write(A)	Read(B) Y:= Y+100 Write(B)

After completion of the transaction, A consists of Rs 500 and B consists of Rs 400.

If the transaction T fails after the completion of transaction T1 but before completion of transaction T2, then the amount will be deducted from A but not added to B. This shows the inconsistent database state. In order to ensure correctness of database state, the transaction must be executed in entirety.

Consistency

- The integrity constraints are maintained so that the database is consistent before and after the transaction.
- The execution of a transaction will leave a database in either its prior stable state or a new stable state.
- The transaction is used to transform the database from one consistent state to another consistent state.

For example: The total amount must be maintained before or after the transaction.

1. Total before T occurs = 600+300=900
2. Total after T occurs= 500+400=900

Therefore, the database is consistent. In the case when T1 is completed but T2 fails, then inconsistency will occur.

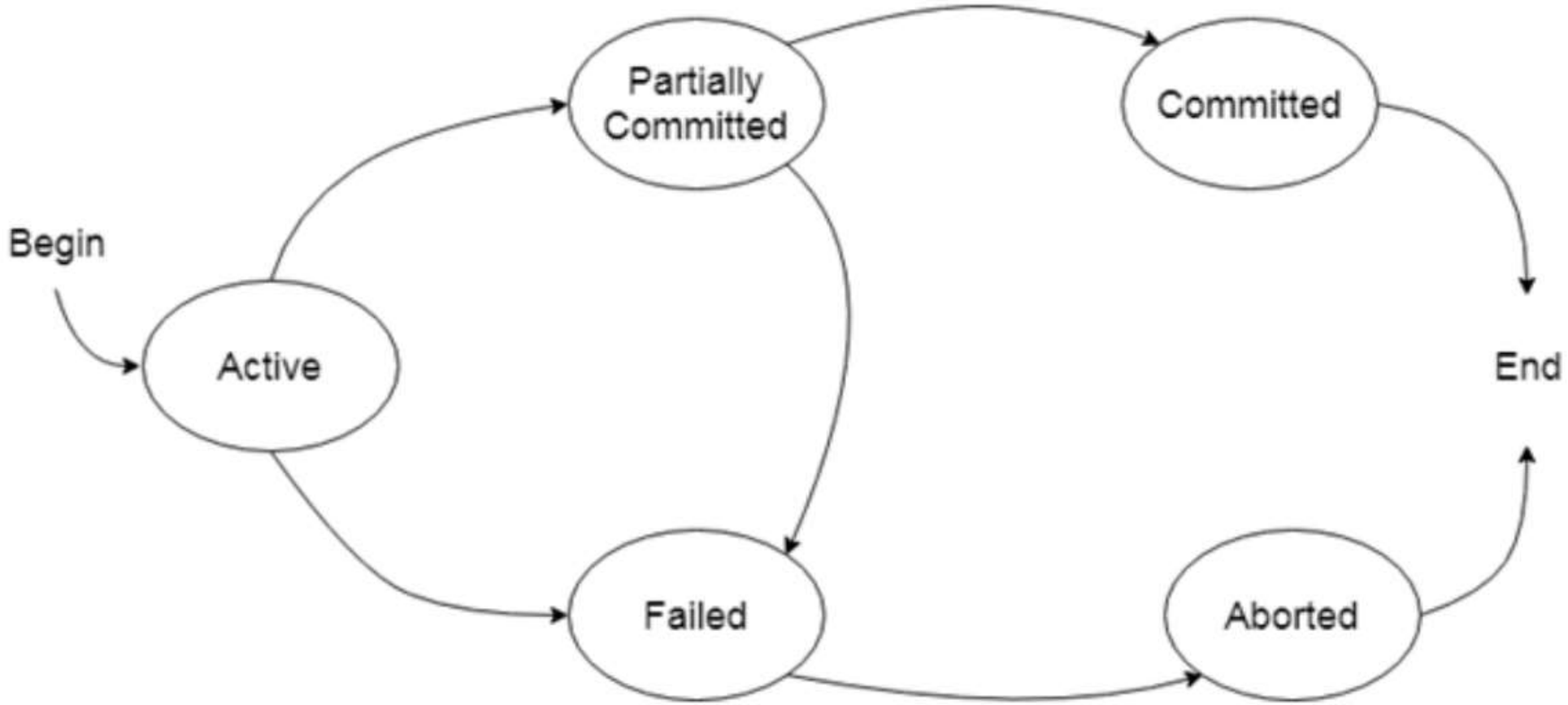
Isolation

- It shows that the data which is used at the time of execution of a transaction cannot be used by the second transaction until the first one is completed.
- In isolation, if the transaction T1 is being executed and using the data item X, then that data item can't be accessed by any other transaction T2 until the transaction T1 ends.
- The concurrency control subsystem of the DBMS enforced the isolation property.

Durability

- The durability property is used to indicate the performance of the database's consistent state. It states that the transaction made the permanent changes.
- They cannot be lost by the erroneous operation of a faulty transaction or by the system failure.
- When a transaction is completed, then the database reaches a state known as the consistent state. That consistent state cannot be lost, even in the event of a system's failure.
- The recovery subsystem of the DBMS has the responsibility of Durability property.

States of Transaction



States of Transaction

- **Active state**: The active state is the first state of every transaction. In this state, the transaction is being executed.
- **Partially Committed**: In the partially committed state, a transaction executes its final operation, but the data is still not saved to the database.
- **Committed**: A transaction is said to be in a committed state if it executes all its operations successfully. In this state, all the effects are now permanently saved on the database system.
- **Failed State**: If any of the checks made by the database recovery system fails, then the transaction is said to be in the failed state.

States of Transaction

Aborted:

- If any of the checks fail and the transaction has reached a failed state then the database recovery system will make sure that the database is in its previous consistent state.
- If not then it will abort or roll back the transaction to bring the database into a consistent state.
- If the transaction fails in the middle of the transaction then before executing the transaction, all the executed transactions are rolled back to its consistent state.
- After aborting the transaction, the database recovery module will select one of the two operations:
  - 1. Re-start the transaction
  - 2. Kill the transaction

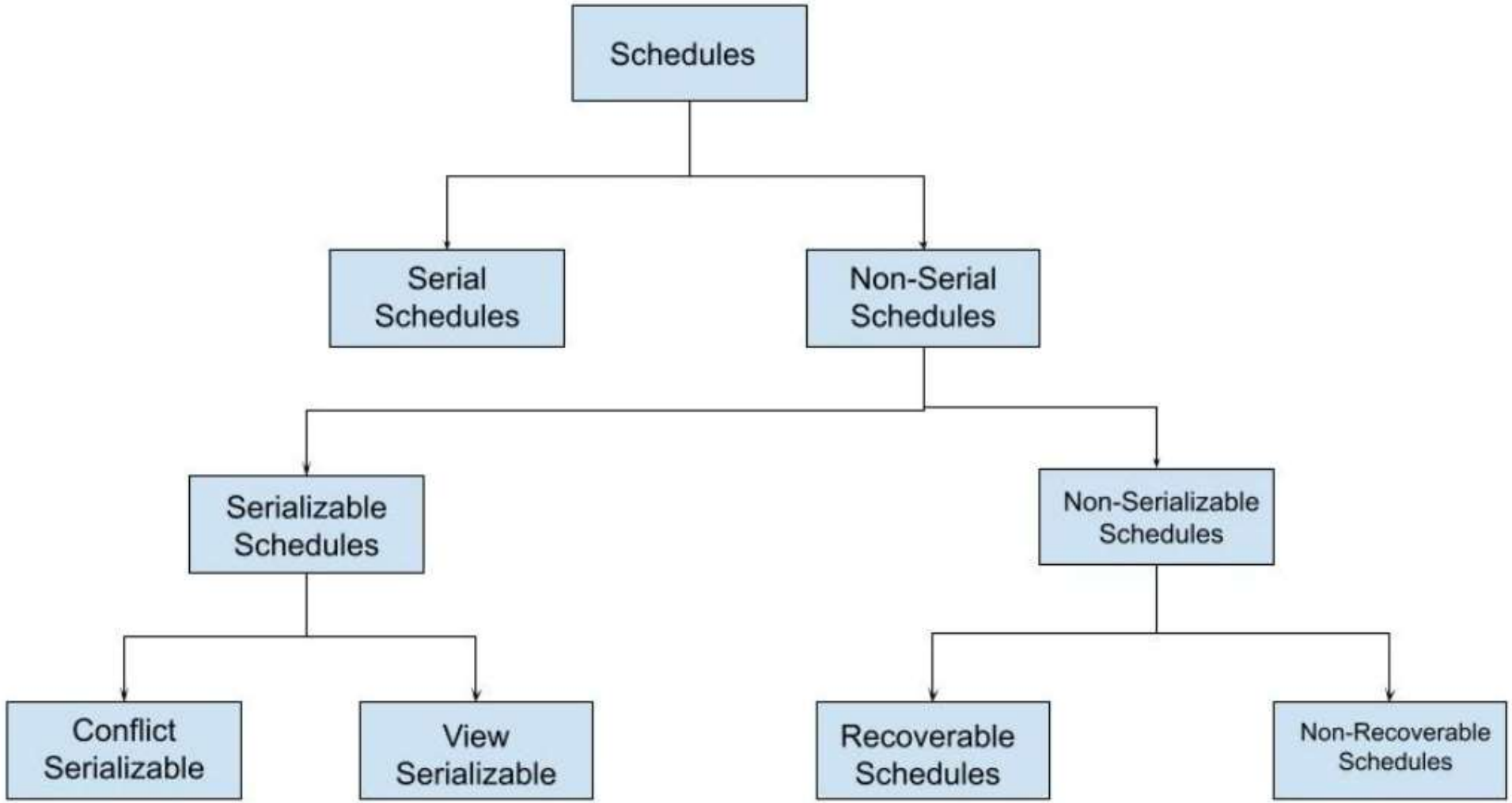
Concurrent Executions

Schedule:

A schedule is a series of operations from one transaction to another transaction. It is used to preserve the order of the operation in each of the individual transaction, a schedule can be of two types:

- Serial: The transactions are executed one after another, in a non-preemptive manner.
- Concurrent: The transactions are executed in a preemptive, time shared method.

Types of Schedules



1. Serial Schedule

- A schedule in which only one transaction is executed at a time, i.e., one transaction is executed completely before starting another transaction.

a)

Time

T1	T2

read(A);
A := A – N;
write(A);
read(B);
B := B + N;
write(B);	

	read(A);
	A := A + M;
	write(A);

Schedule A

b)

Time

T1	T2

	read(A);
	A := A + M;
	write(A);

read(A);
A:= A - N;
write(A);
read(B);
B:= B +N;
write(B);	

Schedule B

For example: Suppose there are two transactions T1 and T2 which have some operations. If it has no interleaving of operations, then there are the following two possible outcomes:

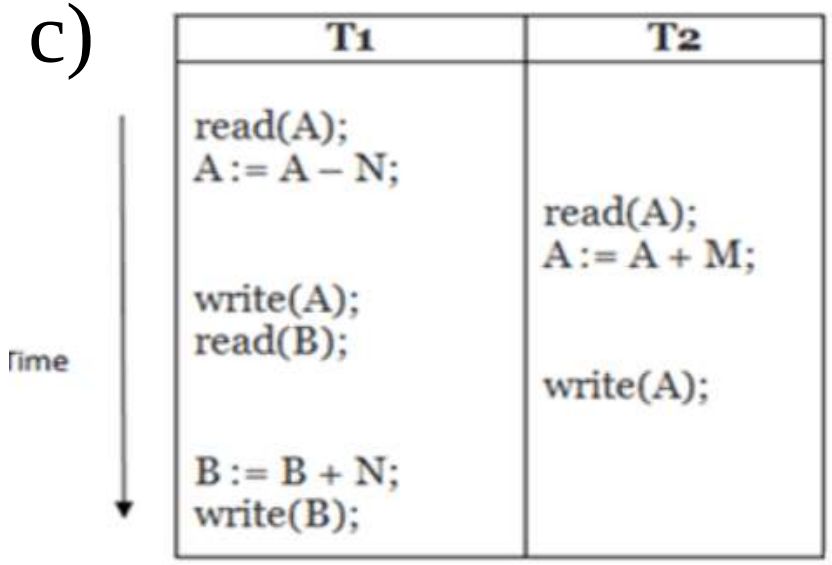
1. Execute all the operations of T1 which was followed by all the operations of T2.
2. Execute all the operations of T2 which was followed by all the operations of T1.

- In the given (a) figure, Schedule A shows the serial schedule where T1 followed by T2.
- In the given (b) figure, Schedule B shows the serial schedule where T2 followed by T1.

2. Non-Serial Schedule

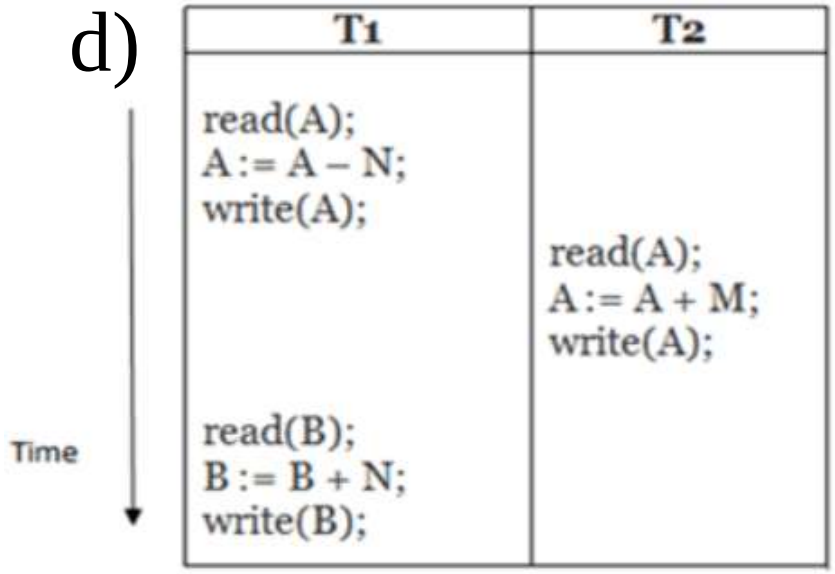
- If interleaving of operations is allowed, then there will be non-serial schedule. It contains many possible orders in which the system can execute the individual operations of the transactions.

c)



Schedule C

d)



Schedule D

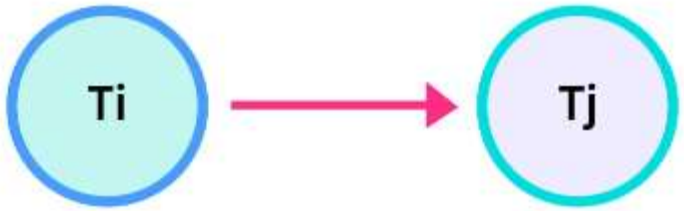
- In the given figure (c) and (d), Schedule C and Schedule D are the non-serial schedules. It has interleaving of operations.

2.1 Serializable schedule

- A non-serial schedule is called a serializable schedule if it can be converted to its equivalent serial schedule

Testing the Serializability:

- To test the serializability of a schedule, we can use **Serialization Graph** or **Precedence Graph**. A serialization Graph is nothing but a Directed Graph of the entire transactions of a schedule.
- It can be defined as a Graph G(V, E) consisting of a set of directed-edges E = {E1, E2, E3, ..., En} and a set of vertices V = {V1, V2, V3, ...,Vn}. The set of edges contains one of the two operations - READ, WRITE performed by a certain transaction.



Ti -> Tj, means Transaction-Ti is either performing read or write before the transaction-Tj.

What is a conflicting pair in transactions?

- Two operations inside a schedule are called conflicting if they meet these three conditions:

1.They belong to two different transactions.

2.They are working on the same data piece.

3.One of them is performing the WRITE operation.

- To conclude, let’s take two operations on data: "a". The conflicting pairs are:

1.READ(a) - WRITE(a)

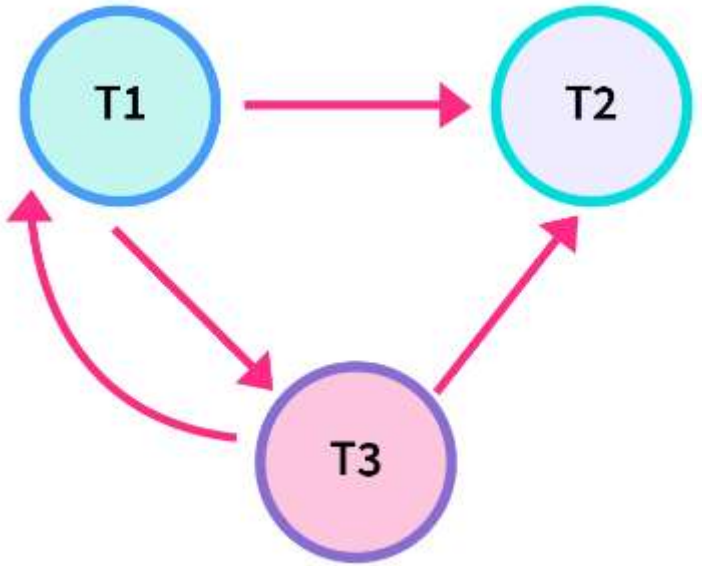
2.WRITE(a) - WRITE(a)

3.WRITE(a) - READ(a)

How to draw precedence Graph?

- Let's take an example of schedule "S" having three transactions t1, t2, and t3 working simultaneously,

T1	T2	T3
R(X)		
		R(Z)
		W(Z)
	R(Y)	
R(Y)		
	W(Y)	
		W(X)
	W(Z)	
W(X)		



Conflict Equivalent

- In the conflict equivalent, one can be transformed to another by swapping non-conflicting operations. In the given example, S2 is conflict equivalent to S1 (S1 can be converted to S2 by swapping non-conflicting operations) [S1=S2].

Non-serial schedule

T1	T2
Read(A)	
Write(A)	
	Read(A)
	Write(A)
Read(B)	
Write(B)	
	Read(B)
	Write(B)

Schedule S1

Serial Schedule

T1	T2
Read(A)	
Write(A)	
Read(B)	
Write(B)	
	Read(A)
	Write(A)
	Read(B)
	Write(B)

Schedule S2

Non Conflict Pair

R (A)	W(A)

Conflict pair

R(A)	W(A)
W(A)	W(A)
W(A)	R(A)

Non Conflict pair

R(A)	W(B)
W(A)	W(B)
W(A)	R(B)
R(A)	R(B)

Steps to check Conflict Equivalent:

T1	T2
READ(A) WRITE(A)	
	READ(A) WRITE(A)
READ(B) WRITE(B)	
	READ(B) WRITE(B)

W(A) conflicts R(A)
W(A) not conflicts with R(B)

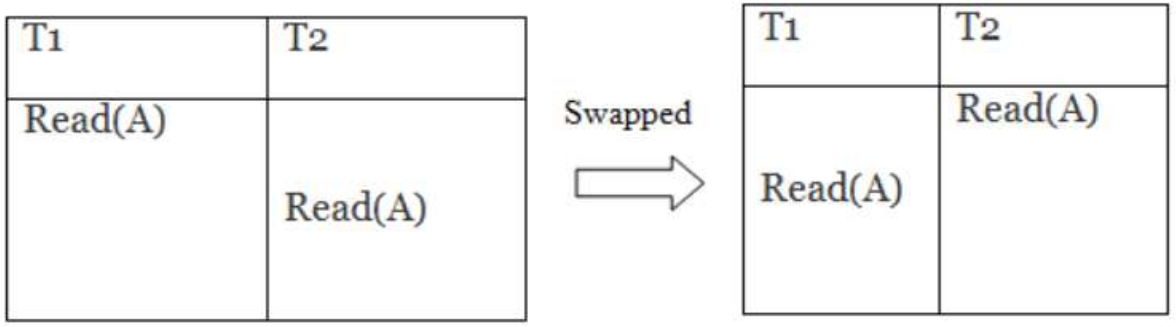
T1	T2
READ(A) WRITE(A) READ(B)	
	READ(A)
WRITE(B)	
	WRITE(A) READ(B) WRITE(B)

T1	T2
READ(A) WRITE(A) READ(B) WRITE(B)	
	READ(A) WRITE(A) READ(B) WRITE(B)

Conflict Serializability

- A schedule is called conflict serializability if after swapping of non-conflicting operations, it can transform into a serial schedule.
- The schedule will be a conflict serializable if it is conflict equivalent to a serial schedule.
- Swapping is possible only if S1 and S2 are logically equal.

1. T1: Read(A) T2: Read(A)



Schedule S1 **Schedule S2**

2. T1: Read(A) T2: Write(A)

T1	T2
Read(A)	
	Write(A)

Schedule S1

Swapped

T1	T2
	Write(A)
Read(A)	

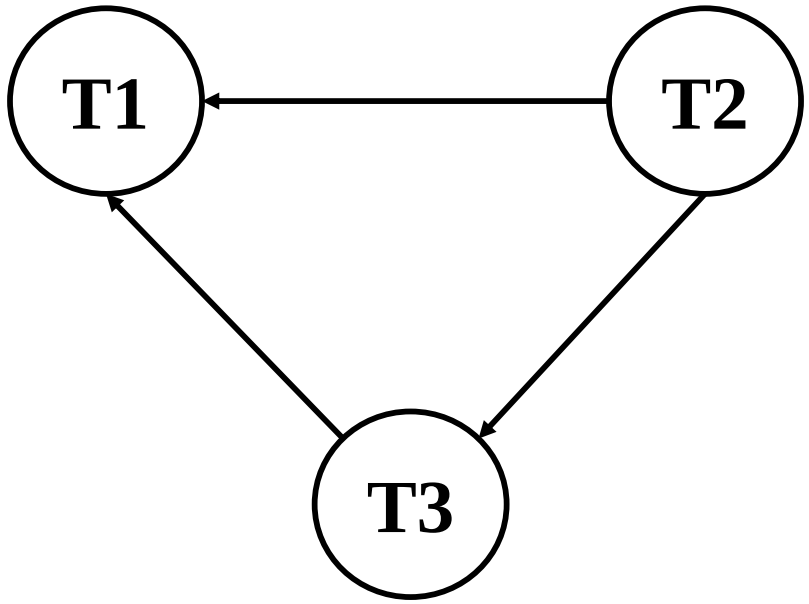
Schedule S2

Here, S1 ≠ S2. That means it is conflict.

Check Conflict pairs in other transactions and draw edges

T1	T2	T3
READ(X)		
		READ(Y)
		READ(X)
	READ(Y)	
	READ(Z)	
		WRITE(Y)
	WRITE(Z)	
READ(Z)		
WRITE(X)		
WRITE(Z)		

PRESEDENCE GRAPH



- No loop / Cycle
- Conflict Serializable
- Serializable (now check this serializable or not)
- Consistent

No of transactions possible for the graph

T1 → T2→ T3

T1→T3→ T2

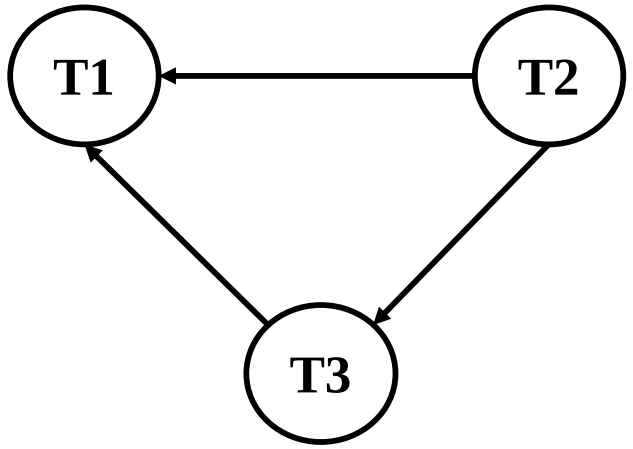
T2→T1→T3

T2→ T3→T1

T3→T1→T2

T3→T2→T3

To find out which sequence can be serializable we need to check indegree of each vertex and it should be 0.



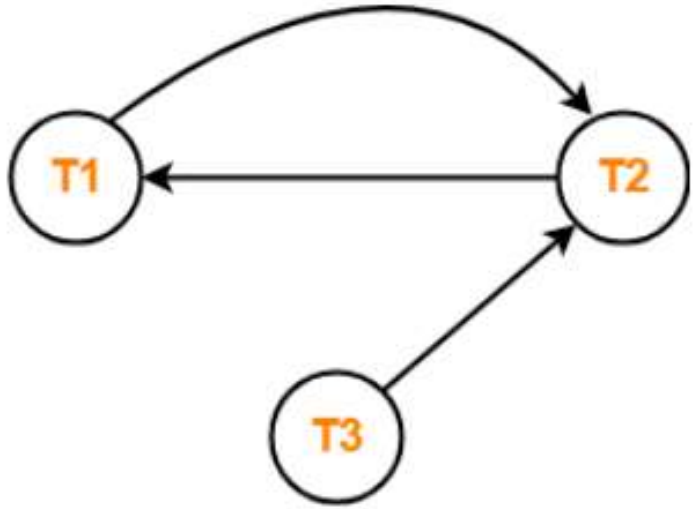
Indegree (T2) → 0 then
Indegree (T3) becomes 0
Finally Indegree (T1) becomes 0
So the final order is:
T2→T3→T1
It is serializable

- Check whether the given schedule S is conflict serializable or not-**S : $R_1(A)$, $R_2(A)$, $R_1(B)$, $R_2(B)$, $R_3(B)$, $W_1(A)$, $W_2(B)$.**

Solution:

List all the conflicting operations and determine the dependency between the transactions-

- $R_2(A)$, $W_1(A)$ $(T_2 \rightarrow T_1)$
- $R_1(B)$, $W_2(B)$ $(T_1 \rightarrow T_2)$
- $R_3(B)$, $W_2(B)$ $(T_3 \rightarrow T_2)$



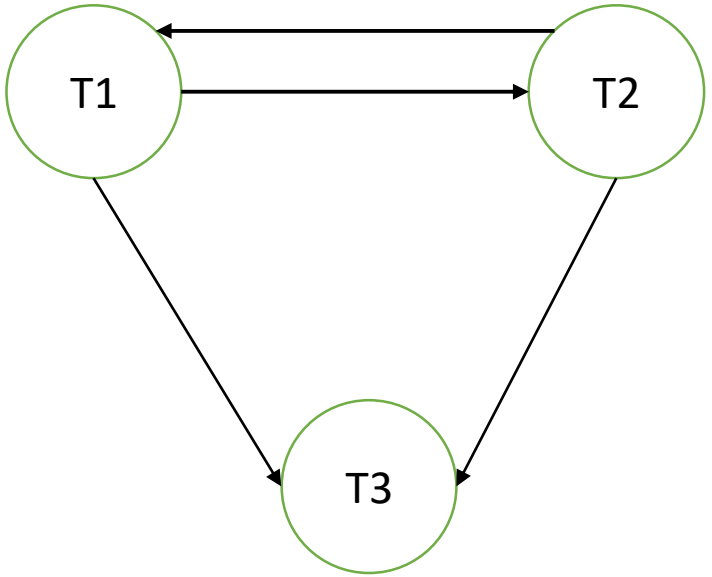
- Clearly, there exists a cycle in the precedence graph.
- Therefore, the given schedule S is not conflict serializable.

2.1.2 View Serializability

- **View Serializability** is a procedure to check if the given schedule is consistent or not.
- It is performed only if the given schedule is not conflict-serializable.
- It is important to check the consistency of the schedule as an inconsistent schedule leads to an inconsistent state of the program which is undesirable.
- If its precedence graph doesn't contain any loop/cycle given schedule is consistent, but if it contains a loop/cycle then the schedule may or may not be consistent.
- To figure out the state of the schedule, we use the concept of View-Serializability because we do not want to bind the concept of Serializability only to Conflict with Serializability.

- Let us understand View-Serializability with an example of a schedule **S1.**
- **First we need to check conflict serializability.**

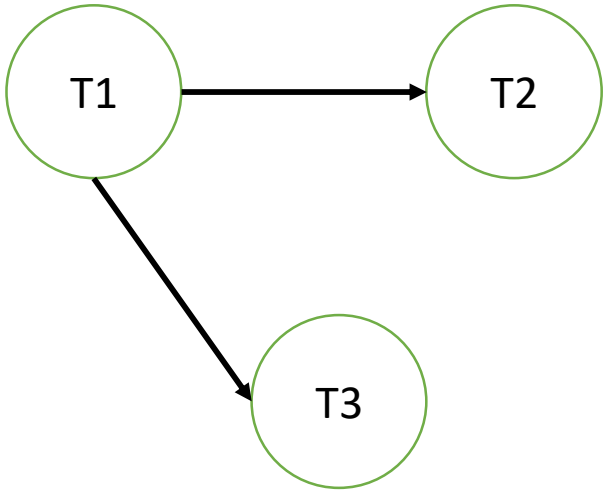
T1	T2	T3
R(X)		
	W(X)	
W(X)		
		W(X)



The above graph contains cycle/loop thus, it is not conflict serializable but by only considering the precedence graph we can't conclude if the given schedule is consistent or not.

- In the above example if we swap a few transaction operations, and format the table as follows:

T1	T2	T3
R(X) W(X)		
	W(X)	
		W(X)



Now check the both schedules whether it is view equivalent or not

VIEW EQUIVALENT

T1	T2	T3
R(X) =100		
	X=X-40 W(X) X=60	
X=X-40 W(X) X=20		
		X=X-20 W(X) X=0

T1	T2	T3
X=100 R(X) X=X-40 W(X) //60		
	X=X-40 W(X) //20	
		X=X-20 W(X) //0

S1 = S2

Recoverability of Schedule

- Sometimes a transaction may not execute completely due to a software issue, system crash or hardware failure. In that case, the failed transaction has to be rollback. But some other transaction may also have used value produced by the failed transaction. So we also have to rollback those transactions.
- There are certain types involved:
  - Irrecoverable schedule
  - Recoverable schedule with cascading rollback
  - cascade less recoverable schedule

Irrecoverable Schedule

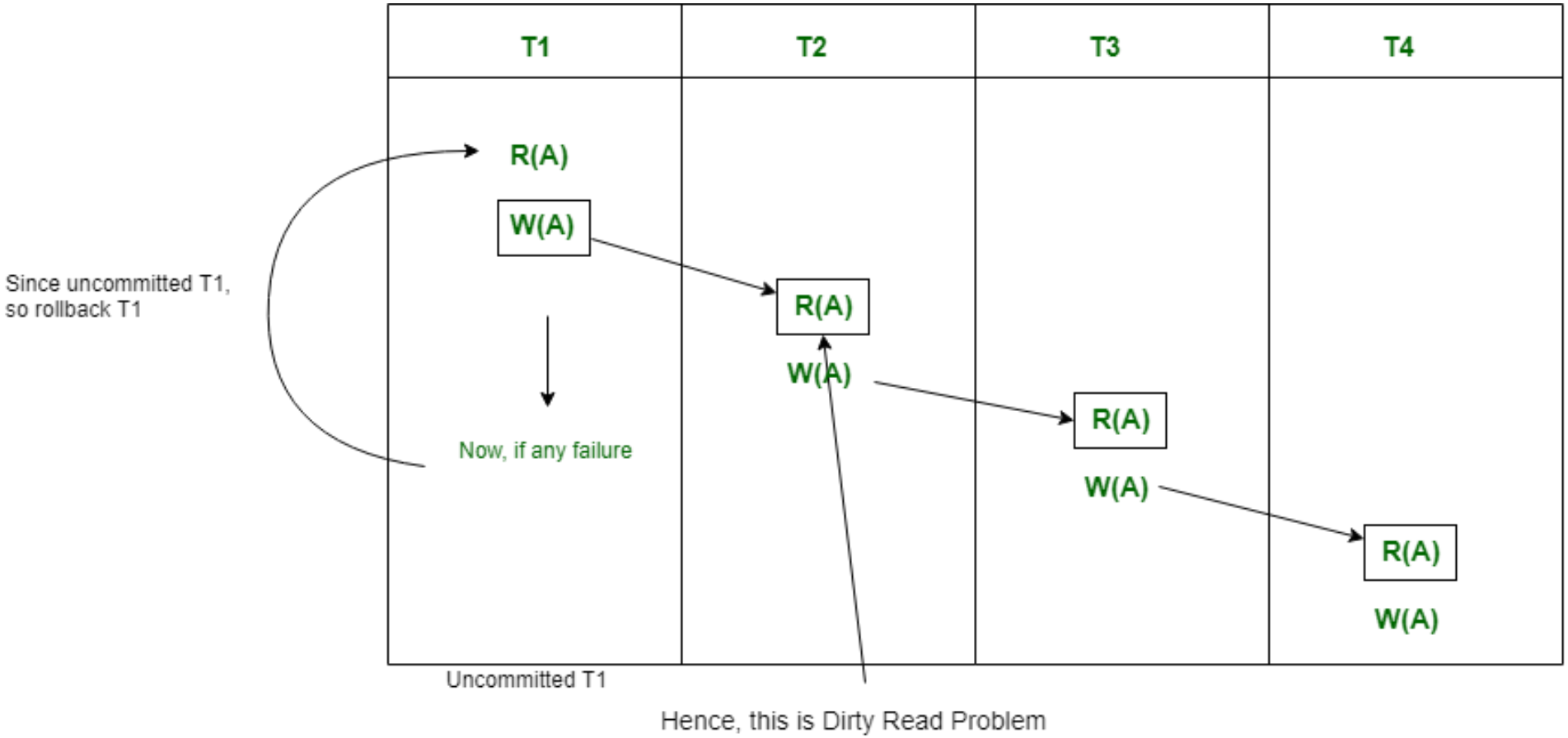
T1	T1's buffer space	T2	T2's buffer space	Database
				A = 6500
Read(A);	A = 6500			A = 6500
A = A - 500;	A = 6000			A = 6500
Write(A);	A = 6000			A = 6000
		Read(A);	A = 6000	A = 6000
		A =A + 1000;	A = 7000	A = 6000
		Write(A);	A = 7000	A = 7000
		Commit;		
Failure Point				
Commit;				

- T1 reads and writes the value of A and that value is read and written by T2. T2 commits but later on, T1 fails. Due to the failure, we have to rollback T1.
- T2 should also be rollback because it reads the value written by T1, but T2 can't be rollback because it already committed. So this type of schedule is known as irrecoverable schedule.

Recoverable with cascading Rollback

What is Cascading rollback?

- If in a schedule, failure of one transaction causes several other dependent transactions to rollback or abort, then such a schedule is called as a Cascading Rollback or Cascading Abort or Cascading Schedule.
- It simply leads to the wastage of CPU time.
- These Cascading Rollbacks occur because of **Dirty Read problems**.



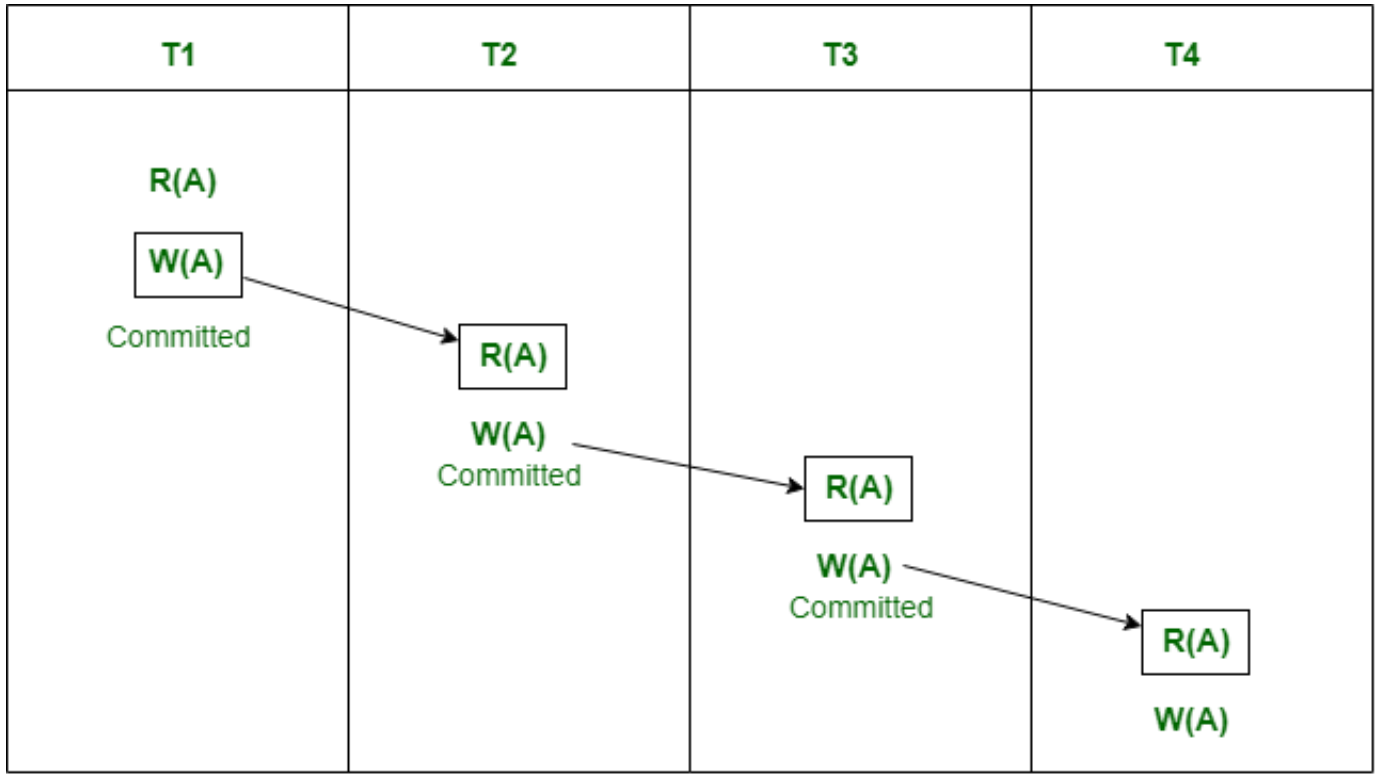
For example, transaction T1 writes uncommitted x that is read by Transaction T2. Transaction T2 writes uncommitted x that is read by Transaction T3.Suppose at this point T1 fails. T1 must be rolled back, since T2 is dependent on T1, T2 must be rolled back, and since T3 is dependent on T2, T3 must be rolled back.

T1	T1's buffer space	T2	T2's buffer space	Database
				A = 6500
Read(A);	A = 6500			A = 6500
A = A - 500;	A = 6000			A = 6500
Write(A);	A = 6000			A = 6000
		Read(A);	A = 6000	A = 6000
		A =A + 1000;	A = 7000	A = 6000
		Write(A);	A = 7000	A = 7000
Failure Point				
Commit;				
		Commit;		

- A schedule with two transactions. Transaction T1 reads and writes A, and that value is read and written by transaction T2. But later on, T1 fails. Due to this, we have to rollback T1.
- T2 should be rollback because T2 has read the value written by T1. As it has not committed before T1 commits so we can rollback transaction T2 as well. So it is recoverable with cascade rollback.

Cascade less Recoverable schedule

- Cascade less Schedule avoids cascading aborts/rollbacks (ACA). Schedules in which transactions read values only after all transactions whose changes they are going to read commit are called cascadeless schedules. Avoids all possible *Dirty Read Problem*

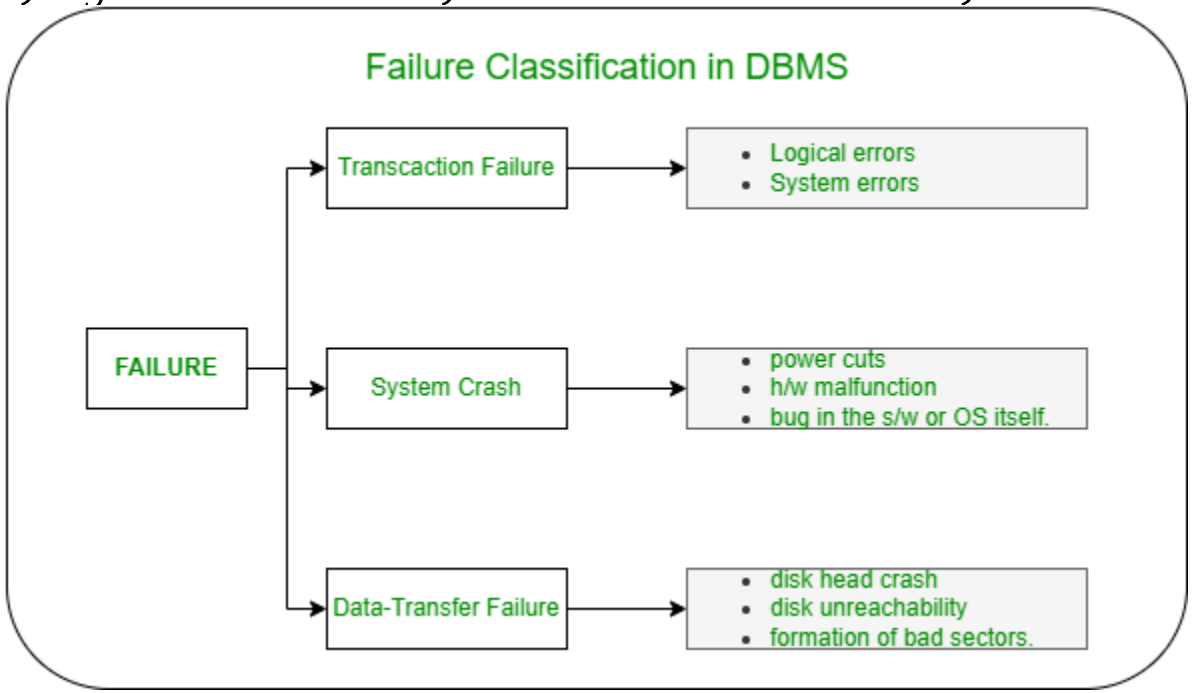


T1	T1's buffer space	T2	T2's buffer space	Database
				A = 6500
Read(A);	A = 6500			A = 6500
A = A - 500;	A = 6000			A = 6500
Write(A);	A = 6000			A = 6000
Commit;		Read(A);	A = 6000	A = 6000
		A =A + 1000;	A = 7000	A = 6000
		Write(A);	A = 7000	A = 7000
		Commit;		

- A schedule with two transactions. Transaction T1 reads and write A and commits, and that value is read and written by T2. So this is a cascade less recoverable schedule.

Failure classification

- Failure in terms of a database can be defined as its inability to execute the specified transaction or loss of data from the database.
- A DBMS is vulnerable to several kinds of failures and each of these failures needs to be managed differently.
- There are many reasons that can cause database failures such as network failure, system crash, natural disasters, software errors etc.



1. Transaction Failure: If a transaction is not able to execute or it comes to a point from where the transaction becomes incapable of executing further then it is termed as a failure in a transaction.

Reason for a transaction failure in DBMS:

1. **Logical error:** A logical error occurs if a transaction is unable to execute because of some mistakes in the code or due to the presence of some internal faults.
2. **System error:** Where the termination of an active transaction is done by the database system itself due to some system issue or because the database management system is unable to proceed with the transaction.

For example– The system ends an operating transaction if it reaches a deadlock condition or if there is an unavailability of resources.

2. **System Crash**:

- A system crash usually occurs when there is some sort of hardware or software breakdown.
- Some other problems which are external to the system and cause the system to abruptly stop or eventually crash include failure of the transaction, operating system errors, power cuts, main memory crash, etc.

3. Disk Failure:

- It occurs where hard-disk drives or storage drives used to fail frequently. It was a common problem in the early days of technology evolution.
- Disk failure occurs due to the formation of bad sectors, disk head crash, and unreachability to the disk or any other failure, which destroy all or part of disk storage.

Concurrency Control

- In the concurrency control, the multiple transactions can be executed simultaneously. It may affect the transaction result.
- It is highly important to maintain the order of execution of those transactions.

Problems of concurrency control:

Several problems can occur when concurrent transactions are executed in an uncontrolled manner. Following are the three problems in concurrency control.

1. Lost updates
2. Dirty read
3. Unrepeatable read

Problem 1: Lost Update Problems (W - W Conflict)

- The problem occurs when two different database transactions perform the read/write operations on the same database items in an interleaved manner (i.e., concurrent execution) that makes the values of the items incorrect hence making the database inconsistent.

Time	T_x	T_y
t_1	READ (A) 300	—
t_2	A = A - 50 250	
t_3	—	READ (A) 300
t_4	—	A = A + 100
t_5	—	—
t_6	WRITE (A) 250	—
t_7		WRITE (A) 400

Here we lost the information of 250/-

- At time t1, transaction T_X reads the value of account A, i.e., $300 (only read).
- At time t2, transaction T_X deducts $50 from account A that becomes $250 (only deducted and not updated/write).
- Alternately, at time t3, transaction T_Y reads the value of account A that will be $300 only because T_X didn't update the value yet.
- At time t4, transaction T_Y adds $100 to account A that becomes $400 (only added but not updated/write).
- At time t6, transaction T_X writes the value of account A that will be updated as $250 only, as T_Y didn't update the value yet.
- Similarly, at time t7, transaction T_Y writes the values of account A, so it will write as done at time t4 that will be $400. It means the value written by T_X is lost, i.e., $250 is lost.

Problem 2: Dirty Read Problems (W-R Conflict)

- The dirty read problem occurs *when one transaction updates an item of the database, and somehow the transaction fails, and before the data gets rollback, the updated database item is accessed by another transaction. There comes the Read-Write Conflict between both transactions.*

Roll back to the previous data so now the T1 contains 100. here we lost T2 data this is dirty read problem

T1	T2

R(A) //100
W(A) //100-50=50
.
.
.
.
.
Failed
	R(A) // 50
	W(A) // 50
	COMMIT;

Time	T_x	T_y
t_1	READ (A)	—
t_2	A = A + 50	—
t_3	WRITE (A)	—
t_4	—	READ (A)
t_5	SERVER DOWN ROLLBACK	—

- At time t1, transaction T_X reads the value of account A, i.e., $300.
- At time t2, transaction T_X adds $50 to account A that becomes $350.
- At time t3, transaction T_X writes the updated value in account A, i.e., $350.
- Then at time t4, transaction T_Y reads account A that will be read as $350.
- Then at time t5, transaction T_X rollbacks due to server problem, and the value changes back to $300 (as initially).
- But the value for account A remains $350 for transaction T_Y as committed, which is the dirty read and therefore known as the Dirty Read Problem.

Problem 3: Unrepeatable read (W – R)

- *Also known as Inconsistent Retrievals Problem that occurs when in a transaction, two different values are read for the same database item.*

Time	T_x	T_y
t_1	READ (A)	—
t_2	—	READ (A)
t_3	—	A = A + 100
t_4	—	WRITE (A)
t_5	READ (A)	—

- At time t1, transaction T_X reads the value from account A, i.e., $300.
- At time t2, transaction T_Y reads the value from account A, i.e., $300.
- At time t3, transaction T_Y updates the value of account A by adding $100 to the available balance, and then it becomes $400.
- At time t4, transaction T_Y writes the updated value, i.e., $400.
- After that, at time t5, transaction T_X reads the available value of account A, and that will be read as $400.
- It means that within the same transaction T_X, it reads two different values of account A, i.e., $ 300 initially, and after updation made by transaction T_Y, it reads $400. It is an unrepeatable read and is therefore known as the Unrepeatable read problem.

Concurrency Control Protocols

- To avoid the concurrency control problems and to maintain consistency and serializability during the execution of concurrency transactions some rules are made those are called concurrency control protocols.
- The concurrency control protocols ensure the *atomicity*, *consistency*, *isolation*, *durability* and *serializability* of the concurrent execution of the database transactions.
- Therefore, these protocols are categorized as:
  - Lock Based Concurrency Control Protocol
  - Time Stamp Concurrency Control Protocol
  - Validation Based Concurrency Control Protocol

Lock Based Protocol

- To attain consistency, isolation between the transactions is the most important tool. Isolation is achieved if we disable the transaction to perform a read/write operation. This is known as locking an operation in a transaction.
- There are two kinds of locks
  - **Shared lock (S):** The locks which *disable the write operations* but *allow read operations* for any data in a transaction are known as shared locks. They are also known as read-only locks and are **represented by 'S'**.
  - **Exclusive lock (X)**:The locks which *allow both the read and write operations for any data in a transaction* are known as exclusive locks. This is a one-time use mode that can't be utilized on the exact data item twice. They are **represented by 'X'.**

- Lets see how this shared and exclusive lock works:

R(A)-read
W(A)-write
S(A)- share lock
X(A)-exclusive lock
U(A)-unlock

	S	X
S	Yes	No
X	No	No

T1	T2
S(A)	
R(A)	S(A)
U(A)	R(A)
	U(A)

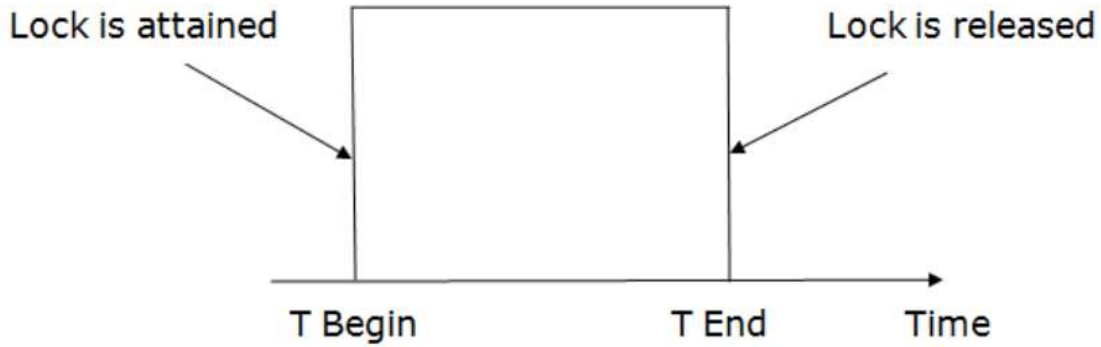
T1	T2
X(A)	
R(A)	
U(A)	
	S(A)
	R(A)
	U(A)

T1	T2
X(A)	
R(A)	
W(A)	
U(A)	
	S(A)
	R(A)

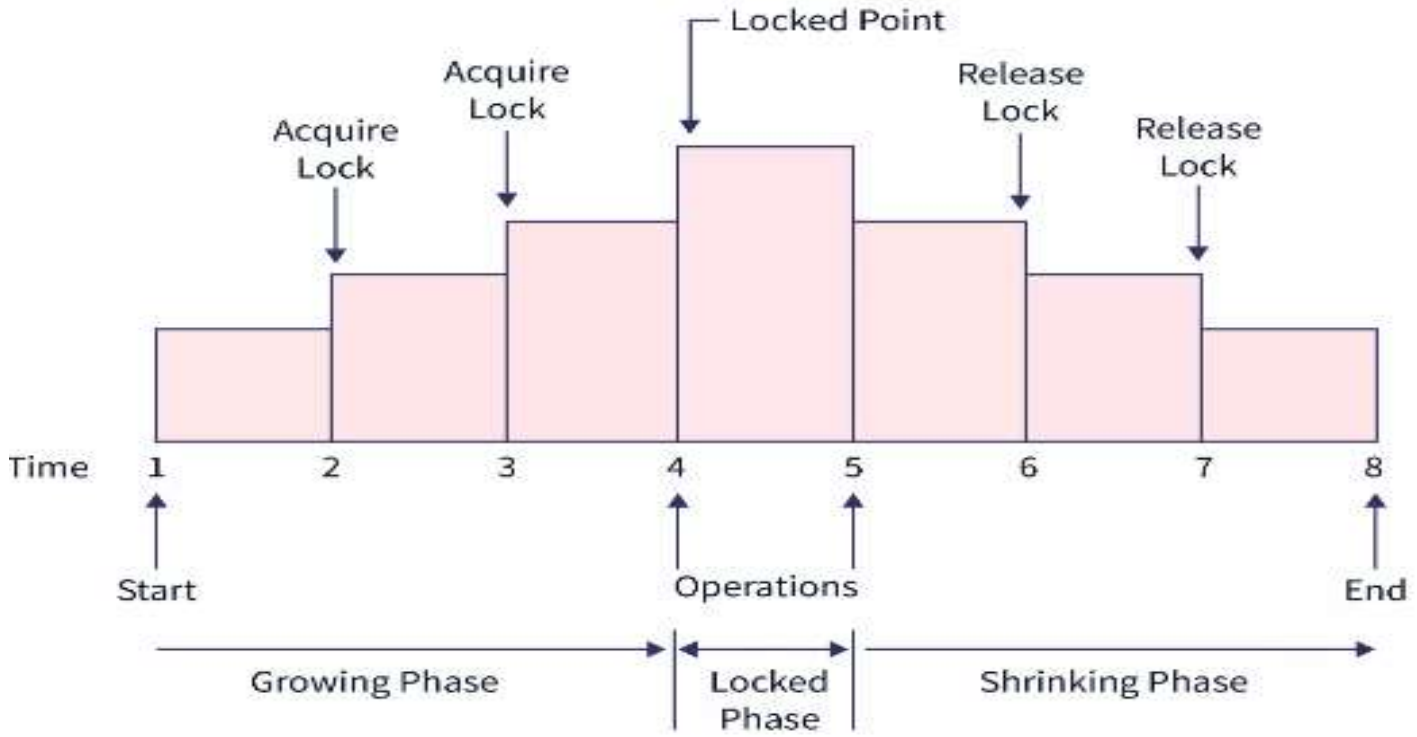
T1	T2
X(A)	
R(A)	
W(A)	
U(A)	
	X(A)
	R(A)

Lock Based protocol is again classified into 4 types:

- Simplistic lock protocol: This protocol instructs to lock all the other operations on the data when the data is going to get updated. All the transactions may unlock all the operations on the data after the write operation
- Pre-claiming lock protocol:
  - As name suggests, this protocol checks the transaction to see what all locks it requires before it begins.
  - Before the transaction begins, it places the request to acquire all the locks on the data items
  - If all the locks are granted, the transaction begins execution and release all the locks once its done execution.
  - If all the locks are not granted this transaction waits until the required locks are granted.



- **Two phase locking protocol(2PL):** In two phase locking protocol the locking and unlocking of data items is done in two phases:
  - **Growing Phase:** In this phase, the locks are acquired on the data items but none of the acquired locks can be released in this phase.
  - **Shrinking Phase**: The existing locks can be released in this phase but no new locks can be acquired in this phase.



- **Strict Two Phase Locking Protocol (Strict – 2PL):** It is similar to 2PL except that it doesn’t have shrinking phase. This protocol releases all the locks only after the transaction is completed successfully and used the commit statement to make the changes permanent in the database.
- It doesn’t release locks after performing an operation on data items. It releases all the locks at the same time once the transaction commit successfully.

Timestamp Ordering Protocol

- The conflicting pairs of operations can be resolved by the timestamp ordering protocol through the utilization of the timestamp values of the transactions. Therefore, guaranteeing that the transactions take place in the correct order. Timestamp is nothing but the time in which a transaction enters into the system.
- For example: T1 at 10 time stamp(older), T2 at 20 time stamp(younger) like wise.

→Unique value assigned to each transaction

→ Read_TS(RTS) = Last (latest) transaction num which performed read successfully

→Write_TS(WTS) = last (latest) transaction num which performed write successfully.

Time stamp ordering protocol functions

Example:

- TS(Ti): Indicate Time stamp of transaction Ti

  Example: T1: 10, T2:20,T3:30

- RTS(A) : = Last (latest) transaction num

  Example: RTS(A) =30

- WTS(A): = Last (latest) transaction num

  Example: WTS(A) =20

RTS (A)

TS(10)	TS(20)	TS(30)
T1	T2	T3
R(A)		
	R(A)	
		R(A)

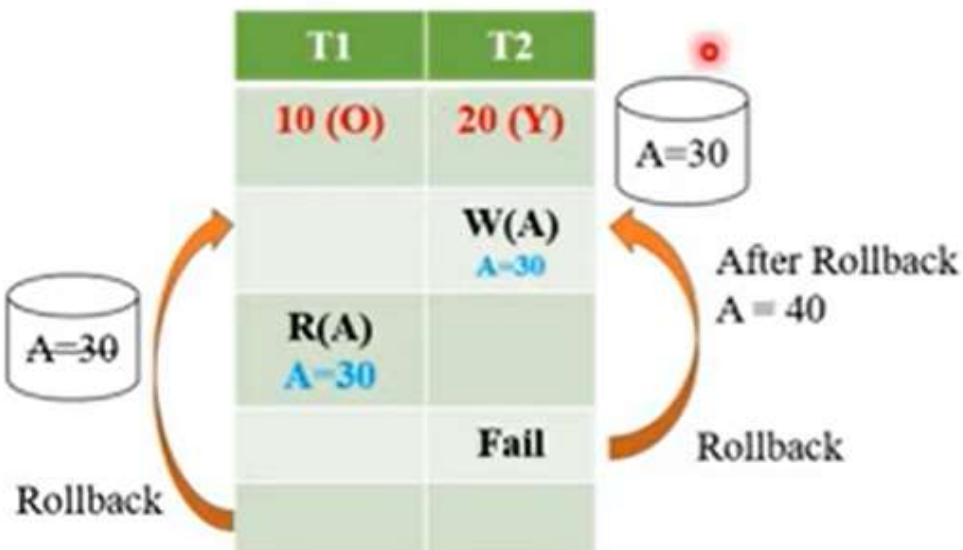
WTS (A)

TS (10)	TS(20)	TS(30)
T1	T2	T3
W(A)		
		W(A)
	W(A)	

Timestamp Ordering Rules : Read()

1. **Check the following condition whenever a transaction Ti issues a Read (X) operation:**
   a. If W_TS (A) > TS (Ti) then Operation rejected & rollback. (Not Allow)
   b. Otherwise execute R(A) operation. Set R_TS(A) = Last (R_TS(A), TS(Ti)). (Allow)

a. Not Allow



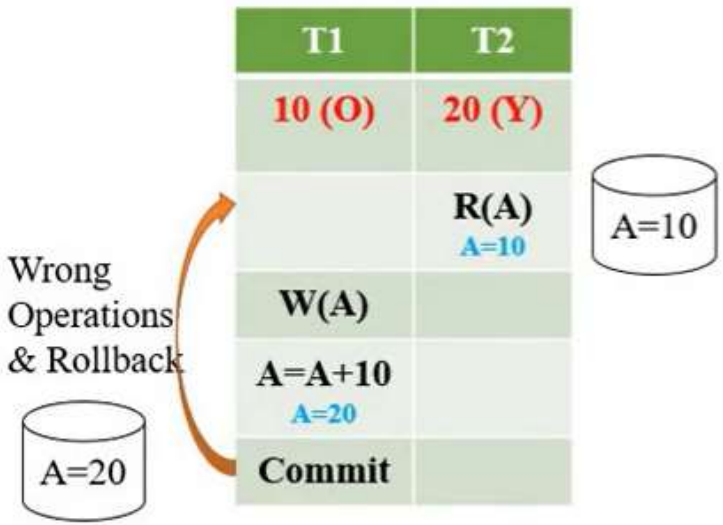
b. Allow

R_TS(A) = 20

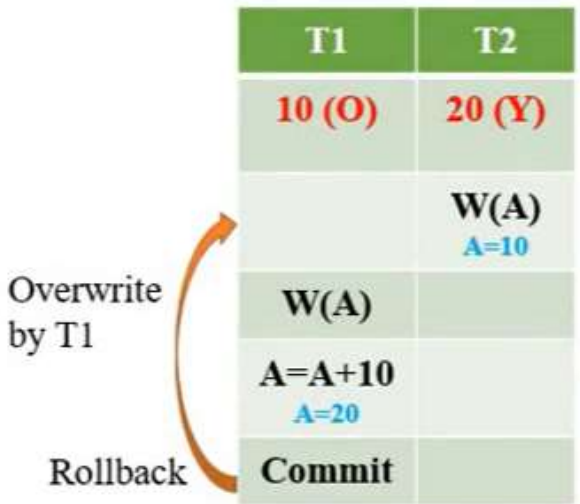
Timestamp Ordering Rules: Write()

1. **Check the following condition whenever a transaction Ti issues a Write(X) operation:**
   a. If R_TS (A) > TS (Ti) then Operation rejected & rollback. (Not Allow)
   b. If W_TS (A) > TS (Ti) then Operation rejected & rollback. (Not Allow)
   c. Otherwise execute Write(A) operation. Set W_TS(A) = Last (TS(Ti)). (Allow)

a. Not Allow



b. Not Allow

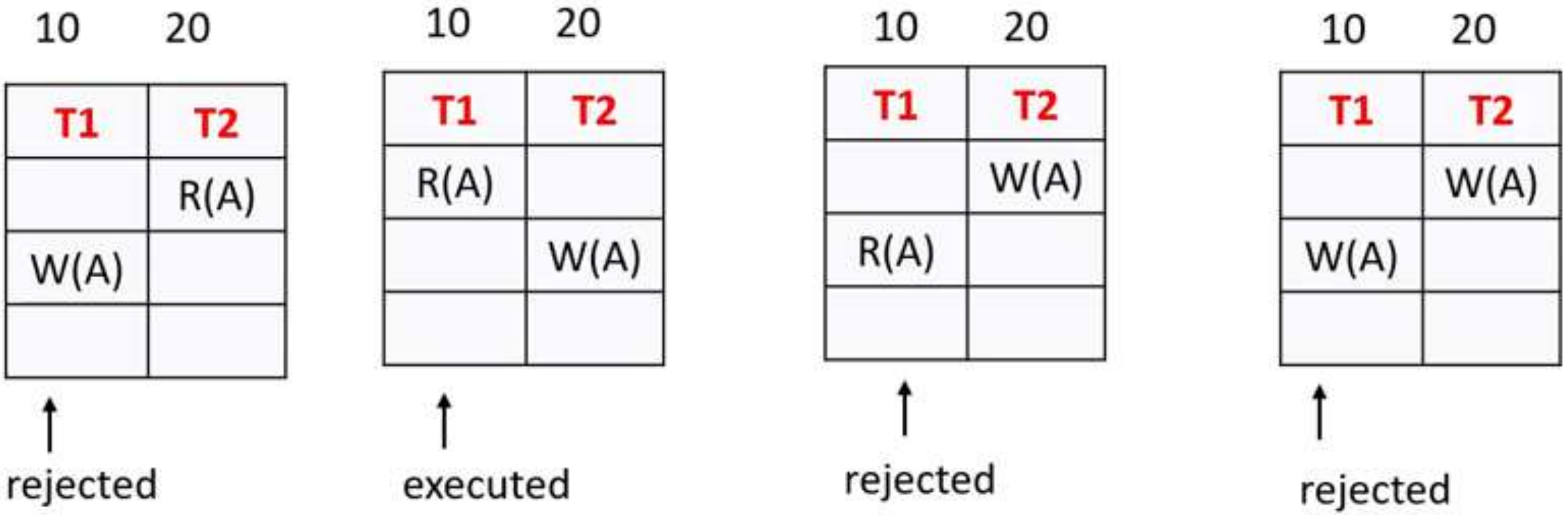


c. Allow

T1	T2
10 (O)	20 (Y)
R(A)	
	W(A)

T1	T2
10 (O)	20 (Y)
W(A)	
	W(A)

Overview


10
20
T1
T2
R(A)
W(A)
rejected
10
20
T1
T2
R(A)
W(A)
executed
10
20
T1
T2
W(A)
R(A)
rejected
10
20
T1
T2
W(A)
W(A)
rejected

Validation Based Protocol

- **Read phase:** In this phase, the transaction T is read and executed. It is used to read the value of various data items and stores them in temporary local variables. It can perform all the write operations on temporary variables without an update to the actual database
- **Validation phase:** In this phase, the temporary variable values will be validated against the actual data to see if it violates serializability. Validation tests have performed on transaction A executing concurrently with transaction B such that TS(A)<TS(B). The transactions must follow one of the following conditions:
  1. **Finish(A)<Start(B):** The operations in transaction A are finished its execution before transaction B starts. Consider two transactions A and B executing its operations. Hence serializability order is maintained.
  2. **Start(B)<Finish(A)<Validate(B):** The list of data items written by transaction A during its write operation should not intersect with the read of the transaction B.
- **Write phase:** If the transaction passes the tests of the validation phase, then the values get copied to the database, otherwise the transaction rolls back.

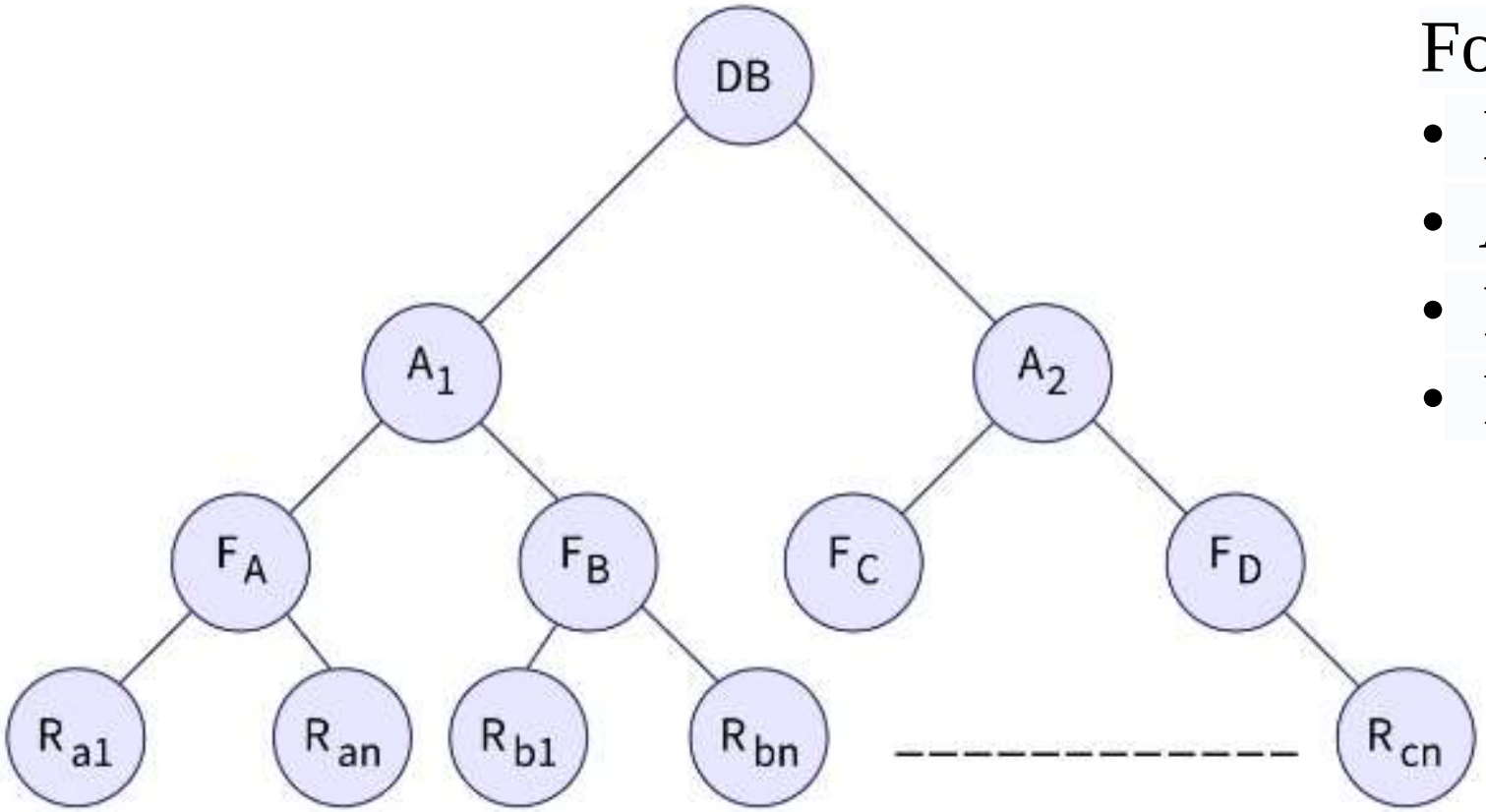
Multiple Granularity

- Transactions in database are atomic, which means when a transaction needs to access the values in the database, a locking protocol is used to lock the database.
- Locking prevents other transactions from modifying or viewing the database simultaneously because other ongoing transactions are locked.
- It is crucial to lock the database while performing a transaction optimally. Locking an entire database unnecessarily can cost a loss of concurrency. We have a concept of multiple granularity in DBMS to solve this issue.

What is Granularity?

- Granularity is the size of data that is allowed to be locked.
- Multiple granularity is hierarchically breaking up our database into smaller blocks that can be locked.
- This locking technique allows the locking of various data sizes and sets.
- This way of breaking the data into blocks that can be locked decreases lock overhead and increases the concurrency in our database.
- Multiple granularity helps maintain a track of data that can be locked and how the data is locked.

We can understand the concept of multiple granularity with a tree diagram. Consider a tree structure as mentioned below.



Four levels in tree are:

- Database
- Area
- File
- Record

Here, our tree has four node levels.

1.The first level, or the tree's root, represents our entire database containing several tables and records – Data Base.

2.The next level (second level) has nodes that represent areas. The database consists of these areas – Areas .

3.The children of the second level are known as files. A file can not be present within more than one area – File .

4.The bottommost layer in our trees is known as the records. Records are child nodes of files and can not be present in more than one file – Records .

Types of Intention Mode Locks in Multiple Granularity

1. **Intention Shared (IS) lock:** If there is an Intention-shared lock on a node, then the lower level tree only has explicit locking with the shared lock. This means that a node can have an Intention Shared (IS) lock only when only nodes at the lower level are locked with shared locks.
2. **Intention Exclusive (IX) lock:** Explicit lock on the lower level tree from the node with either shared or exclusive lock.
3. **Shared and Intention Exclusive (SIX) lock:** In such type of lock, the subtree of the locked node is explicitly locked in shared mode and with an exclusive mode lock at the lower level.

Let us see the compatibility matrix for the five locks that we now know:

	IS	IX	S	SIX	X
IS	✓	✓	✓	✓	✗
IX	✓	✓	✗	✗	✗
S	✓	✗	✓	✗	✗
SIX	✓	✗	✗	✗	✗
X	✗	✗	✗	✗	✗

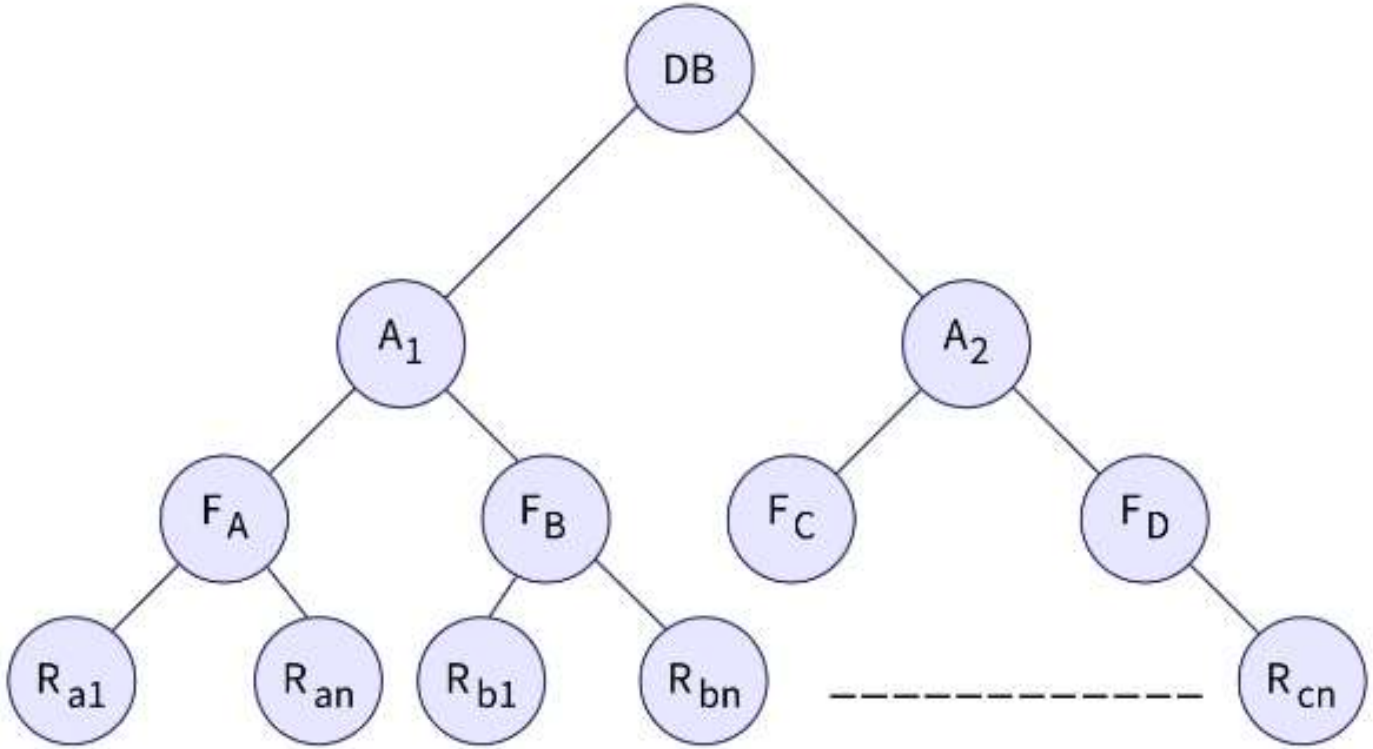
To ensure serializability in data, we use intention lock modes in multiple granularity protocols. If a transaction T attempts to lock a node, it needs to follow the following protocols:

- The transaction T must follow the lock-compatibility matrix.
- Transaction T first locks the root node (in any mode).
- If the parent node of T has IX or IS lock, then transaction T will lock the node only in IS or S mode.
- If the parent node of T has locked in mode SIX or IX, then transaction T will lock the node only in SIX, IX, or X mode.
- If transaction T has previously not unlocked any node, then transaction T can lock a node.
- If none of the children of transaction T is locked, only then can transaction T unlock a node.

The locks in multiple granularity are acquired from top to down (top-down order), but the locks must be released in the bottom-up order.

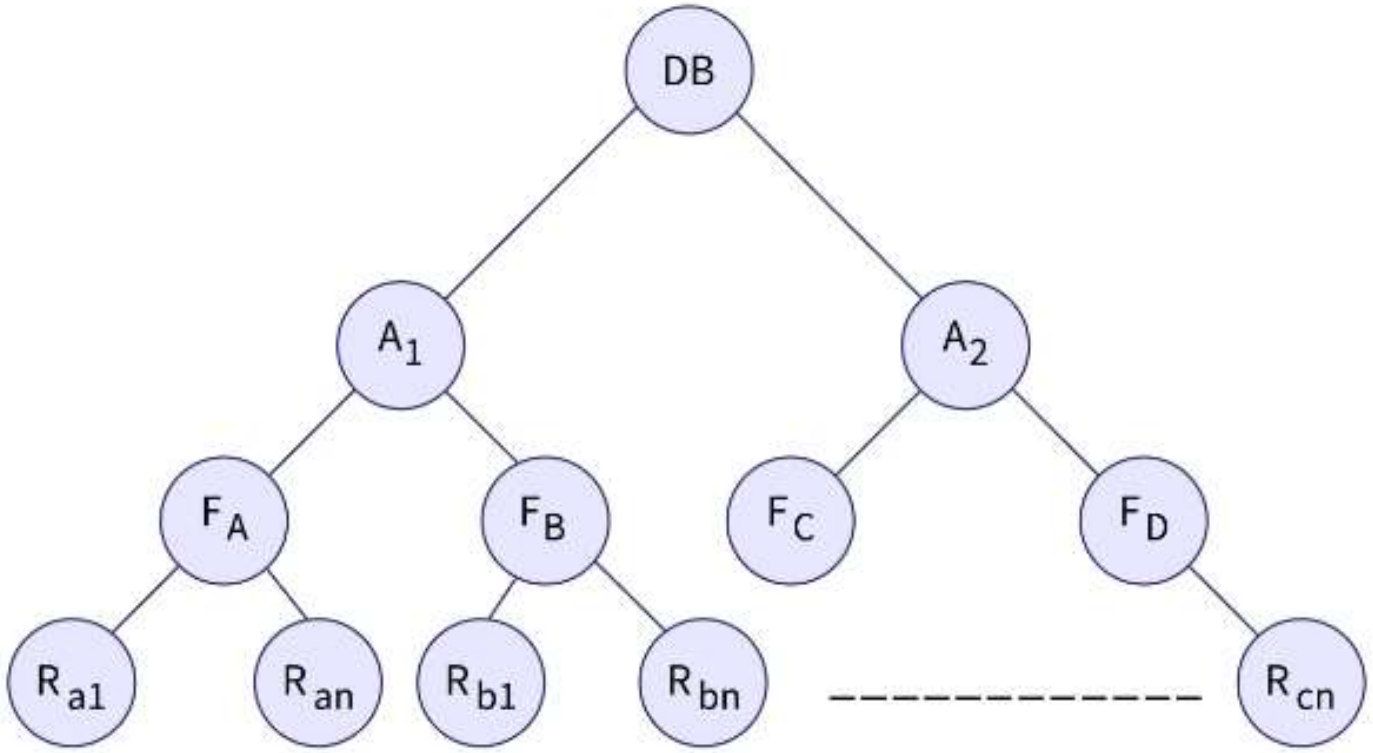
Example

- Suppose transaction T1 reads the recode Ra1 in a file Fa. Then another transaction, T2, will lock in the database, area A1, file Fa with IS mode, and record Ra1 in Shared mode in the same order.



Shared lock is only applied on reading transactions

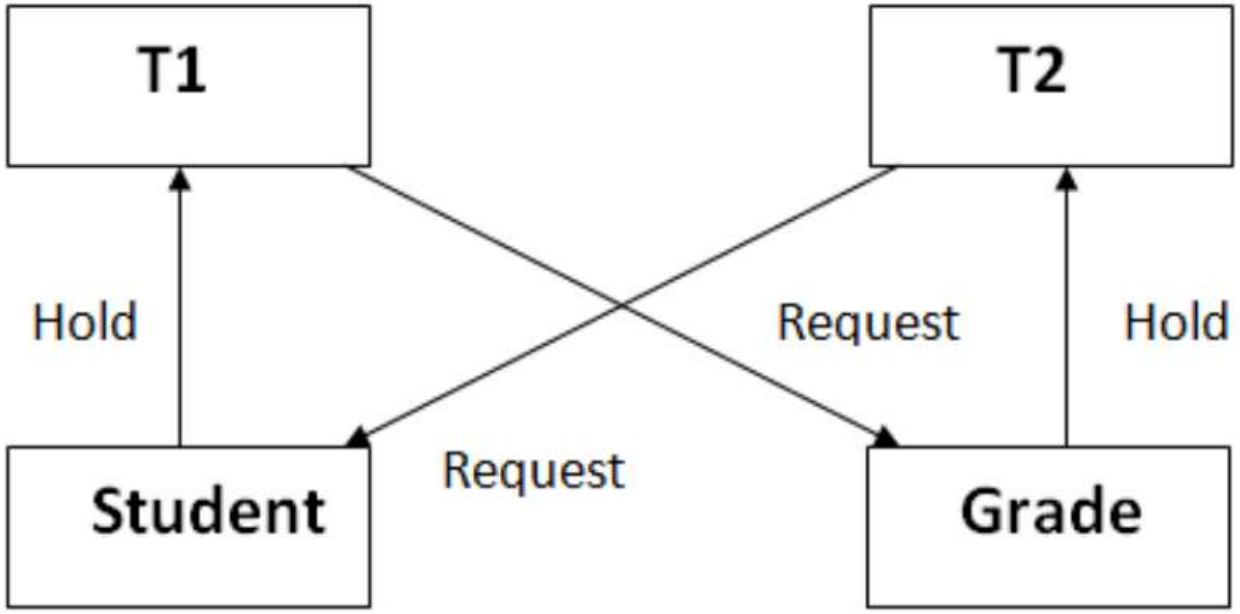
Let's say our transaction T2 modifies the record Ra8 in the same file Fa. For this operation database, area A1 and file Fa will be locked in IX mode, and the target record Ra8 will be locked in exclusive (X) mode.



- Another transaction T3, tries to read all records in our file Fa. Then T3 will lock the nodes in its path, i.e. database and area A1, in IS mode, and then file Fa will be locked in shared (S) mode.
- Lastly, let's say transaction T4 reads our entire database. Then transaction T4 will lock the database in S mode and read the database.
- Here, our transactions T1, T3, and T4 concurrently access the database, while transactions T2 and T1 can execute concurrently but can not execute concurrently with either transaction T3 and T4.

Deadlock handling

- A deadlock is a condition where two or more transactions are waiting indefinitely for one another to give up locks.
- Deadlock is said to be one of the most feared complications in DBMS as no task ever gets finished and is in waiting state forever.



Deadlock Avoidance

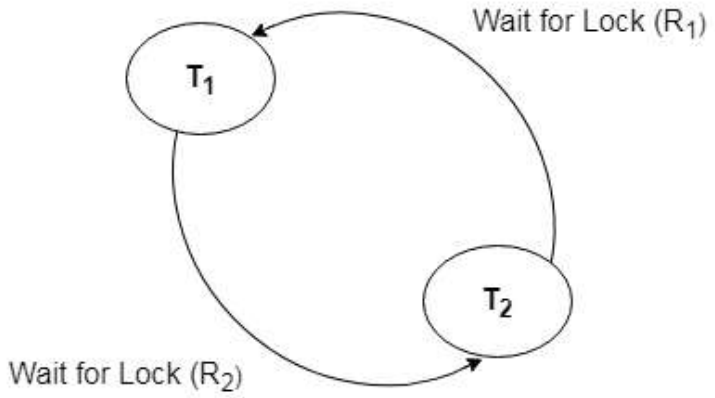
- When a database is stuck in a deadlock state, then it is better to avoid the database rather than aborting or restating the database. This is a waste of time and resource.
- Deadlock avoidance mechanism is used to detect any deadlock situation in advance.
- A method like "wait for graph" is used for detecting the deadlock situation but this method is suitable only for the smaller database. For the larger database, deadlock prevention method can be used.

Deadlock Detection

- In a database, when a transaction waits indefinitely to obtain a lock, then the DBMS should detect whether the transaction is involved in a deadlock or not. The lock manager maintains a Wait for the graph to detect the deadlock cycle in the database.

Wait for Graph

- This is the suitable method for deadlock detection. In this method, a graph is created based on the transaction and their lock. If the created graph has a cycle or closed loop, then there is a deadlock.



Deadlock Prevention

- Deadlock prevention method is suitable for a large database. If the resources are allocated in such a way that deadlock never occurs, then the deadlock can be prevented.

Wait-Die scheme:

- In this scheme, if a transaction requests for a resource which is already held with a conflicting lock by another transaction then the DBMS simply checks the timestamp of both transactions. It allows the older transaction to wait until the resource is available for execution.
- Let's assume there are two transactions Ti and Ti and let TS(T) is a timestamp of any transaction T. If T2 holds a lock by some other transaction and T1 is requesting for resources held by T2 then the following actions are performed by DBMS:
  1. Check if TS(Ti) < TS(Ti) - If Ti is the older transaction and Ti has held some resource, then Ti is allowed to wait until the data-item is available for execution. That means if the older transaction is waiting for a resource which is locked by the younger transaction, then the older transaction is allowed to wait for resource until it is available.
  2. Check if TS(Tj) < TS(Ti) - If Ti is older transaction and has held some resource and if Ti is waiting for it, then Tj is killed and restarted later with the random delay but with the same timestamp.

Crash Recovery/Advanced recovery systems

- Crash recovery is the process by which the database is moved back to a consistent and usable state.
- This is done by rolling back incomplete transactions and completing committed transactions that were still in memory when the crash occurred.
- When the database is in a consistent and usable state, it has attained what is known as a point of consistency.

Following a transaction failure, the database must be recovered:

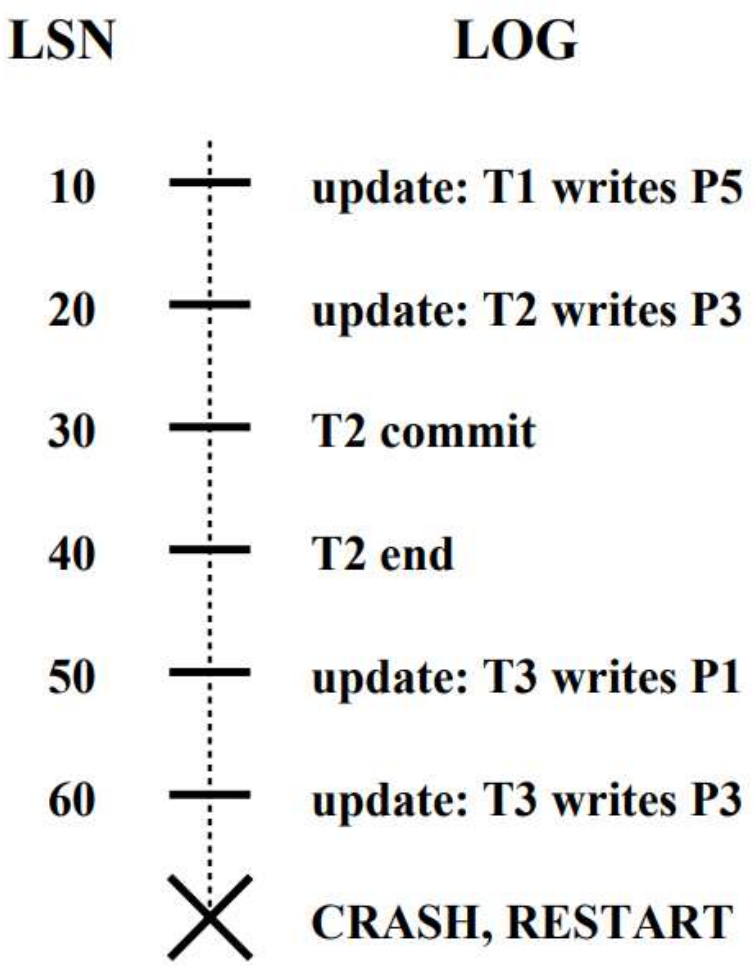
- A power failure on the machine causing the database manager and the database partitions on it to go down.
- A hardware failure such as memory corruption, or disk, CPU, or network failure.
- A serious operating system error that causes the DB to go down

1. ARIES

- ARIES stands for Algorithms for Recovery and Isolation Exploiting Semantics. ARIES is a recovery algorithm designed to work with no-force, steal database approach.
- When the recovery manager is invoked after a crash, restart proceeds in three phases:
  - **Analysis:** The first phase, analysis, compute all the necessary information from the log file.
  - **Redo**: The Redo phase restores the database to the exact state at the crash, including all the changes of uncommitted transactions that were running at that point time.
  - **Undo**: The undo phase then undoes all uncommitted changes, leaving the database in a consistent state. After the redo phase, the database reflects the exact state at the crash. However, the changes of uncommitted transactions have to be undone to restore the database to a consistent state

Example

- When the system restarted, the Analysis phase identifies T1 and T3 as transactions that were active at the time of the crash, and therefore to be undone;
- T2 as a committed transaction, and all its actions, therefore, to be written to disk; and P1, P3, and P5 as potentially dirty pages.
- All the updates (including those of T1 and T3) are reapplied in the order shown during the Redo phase.
- Finally, the actions of T1 and T3 are undone in reverse order during the Undo phase; that is, T3's write of P3 is undone, T3's write of P1 is undone, and then T1's write of P5 is undone



There are three main principles behind the ARIES recovery algorithm:

- **Write-ahead logging**: Any change to a database object is first recorded in the log; the record in the log must be written to stable storage before the change to the database object is written to disk.

- **Repeating history during Redo**: Upon restart following a crash, ARIES retraces all actions of the DBMS before the crash and brings the system back to the exact state that it was in at the time of the crash. Then, it undoes the actions of transactions that were still active at the time of the crash

- **Logging changes during Undo**: Changes made to the database while undoing a transaction are logged in order to ensure that such an action is not repeated in the event of repeated restarts.

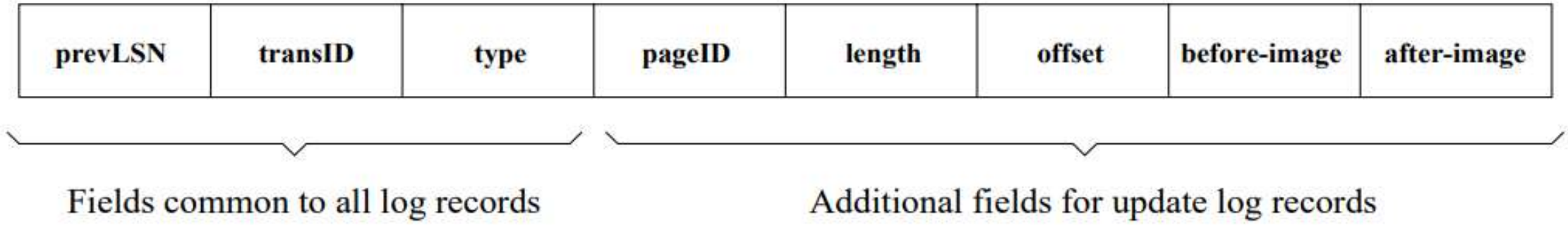
2. The LOG

- A transaction log is a history of actions executed by a database management system to guarantee ACID properties over crashes or hardware failure.
- Physically a log is a file of updates done to the database stored in a stable storage.
- The log, sometimes called the trail or journal, is a history of actions executed by the DBMS.
- Every log record is given a unique id called the log sequence number (LSN). As with any record id, we can fetch a log record with one disk access given the LSN.
- Further, LSNs should be assigned in monotonically increasing order; this property is required for the ARIES recovery algorithm

A log record is written for each of the following actions:

- **Updating a page**: After modifying the page, an update type record is appended to the log tail. The pageLSN of the page is then set to the LSN of the update log record.
- **Commit**: When a transaction decides to commit, it force-writes a commit type log record containing the transaction id. That is, the log record is appended to the log, and the log tail is written to stable storage, up to and including the commit record.
- **Abort:** When a transaction is aborted, an abort type log record containing the transaction id is appended to the log, and Undo is initiated for this transaction
- **End:** when a transaction is aborted or committed, some additional actions must be taken beyond writing the abort or commit log record. After all these additional steps are completed, an end type log record containing the transaction id is appended to the log.
- **Undoing an update:** When a transaction is rolled back ,its updates are undone. When the action described by an update log record is undone, a compensation log record, or CLR, is written.

- Update Log Records:



- The pageID field is the page id of the modified page;
- the length in bytes and the offset of the change are also included.
- The before-image is the value of the changed bytes before the change;
- the after-image is the value after the change.

3. The Write-Ahead Log Protocol

- A Write – Head logging protocol (WAL) is a family of techniques for providing atomicity and durability in database systems.
- In a system using WAL, all modifications are written to a log before they are applied.
- Usually, both redo and undo information is stored in a log.
- WAL allows updates of a database to be done in one place.

4. Check points

- The Checkpoint is used to declare a point before which the DBMS was in a consistent state, and all transactions were committed.
- During transaction execution, such checkpoints are traced. After execution, transaction log files will be created.
- Upon reaching the savepoint/checkpoint, the log file is destroyed by saving its update to the database.
- Then a new log is created with upcoming execution operations of the transaction and it will be updated until the next checkpoint and the process continues.

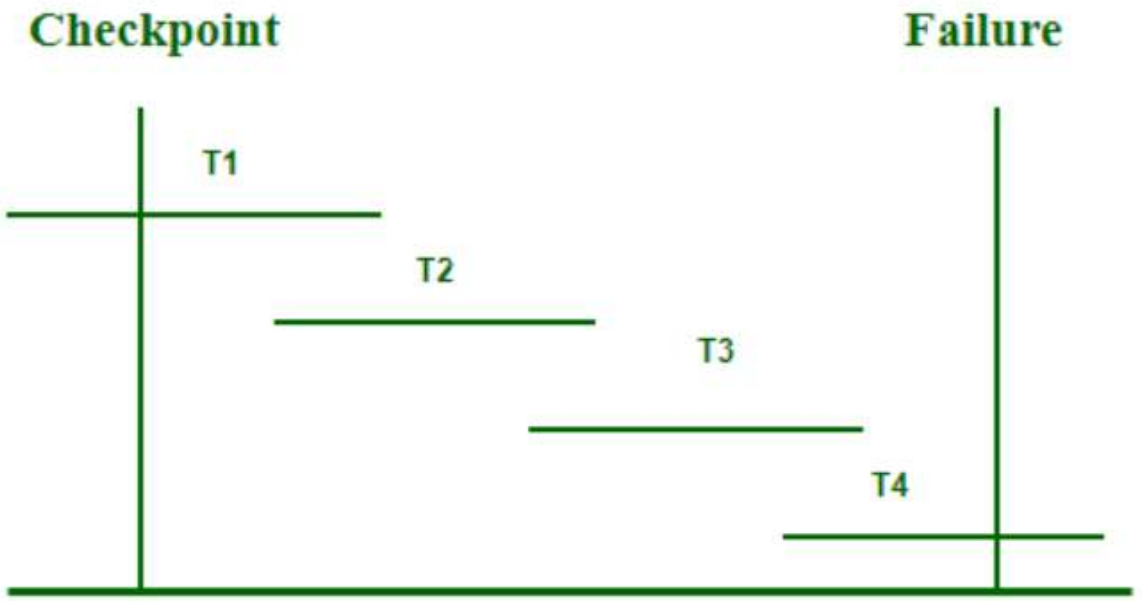
- **Steps to Use Checkpoints in the Database**

1.Write the begin_checkpoint record into a log.

2.Collect checkpoint data in stable storage.

3.Write the end_checkpoint record into a log.

- The behavior when the system crashes and recovers when concurrent transactions are executed is shown below:



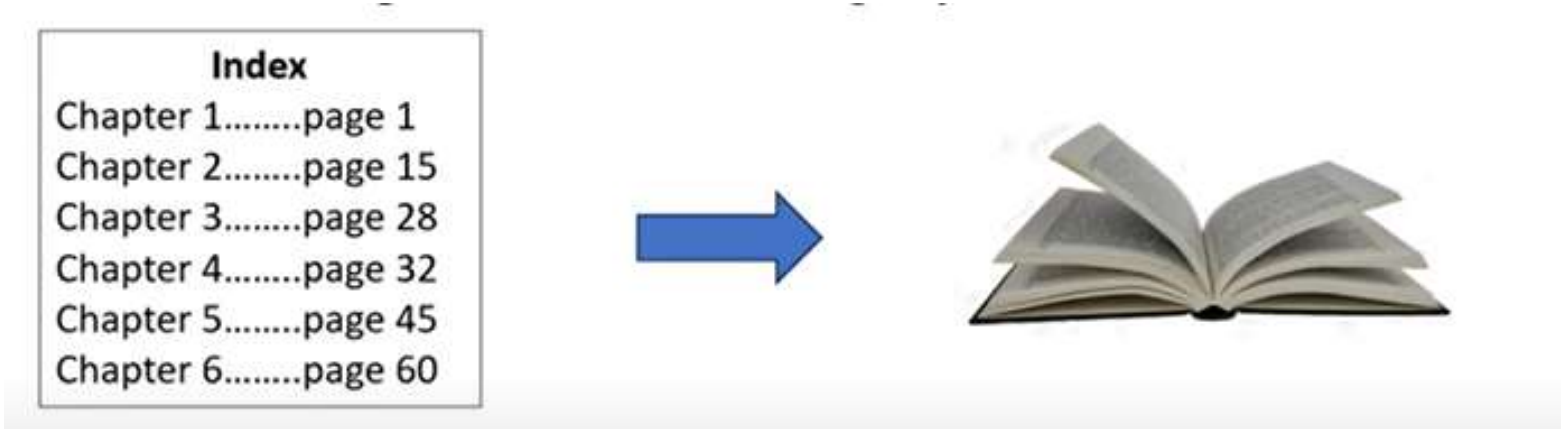
5. RECOVERING FROM A SYSTEM CRASH

When the system is restarted after a crash, the recovery manager proceeds in three phases,

- Analysis Phase
- Redo Phase
- Undo Phase

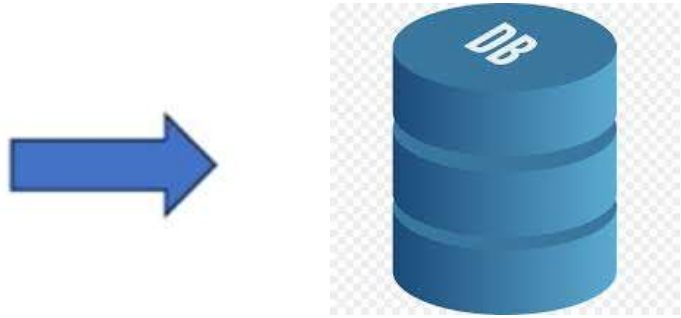
Indexing

- Indexing is used to quickly retrieve particular data from the database. Formally we can define indexing as a technique that uses data structures to optimize the searching time of a database query in DBMS.



Indexing

Search Key	Data reference
1	
2	



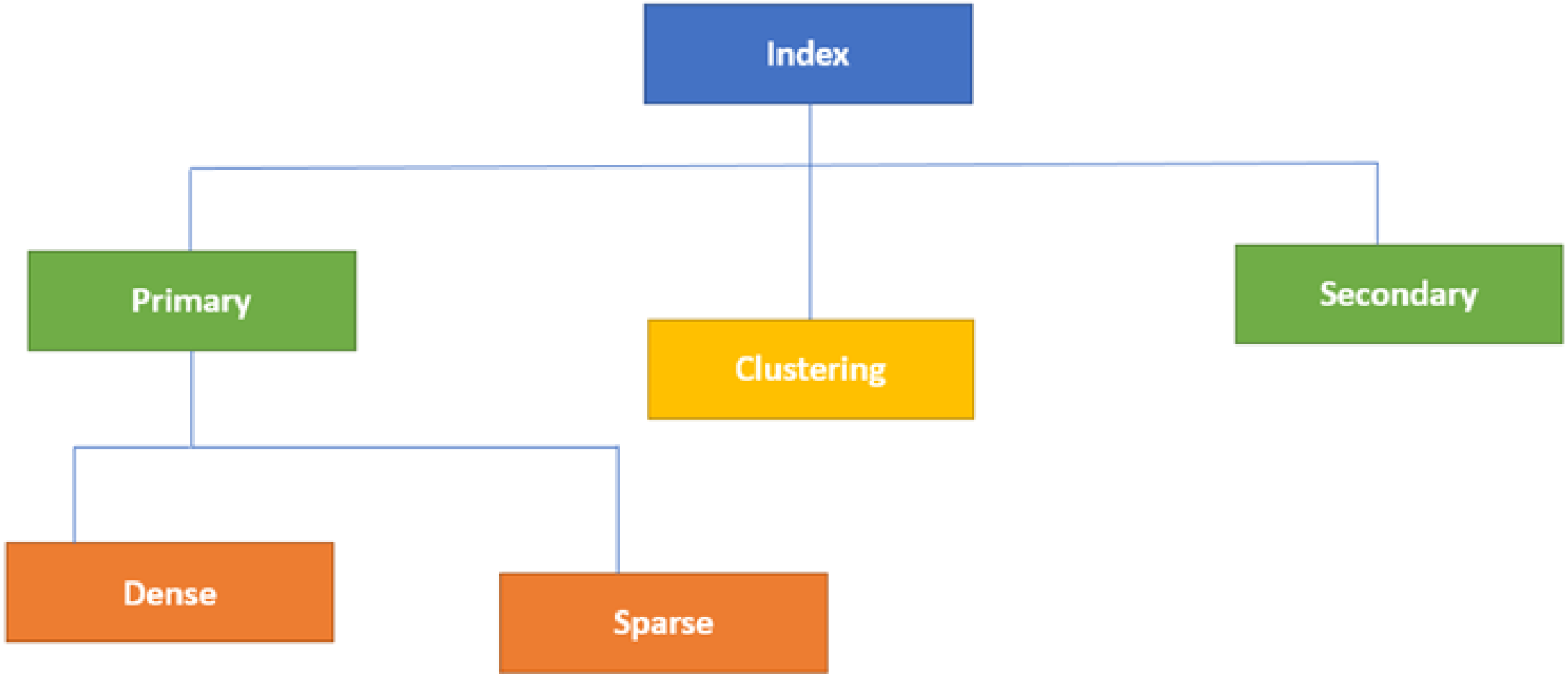
Index structure

- Index usually consists of two columns which are key-value pairs.

Search key	Data Reference

- The first column of the database is the search key that contains a copy of the primary key or candidate key of the table. The values of the primary key are stored in sorted order so that the corresponding data can be accessed easily.
- The second column of the database is the data reference. It contains a set of pointers holding the address of the disk block where the value of the particular key can be found.

Types of Indexes



Primary Index

Order & unique

Roll no	
1	
2	
3	
4	

Cluster Index

Order & not unique

Roll no	
1	
1	
2	
2	
2	
3	

Secondary Index

Unorder & Unique

mobile	
235	
135	
435	
352	
223	
153	

Unorder & Not Unique

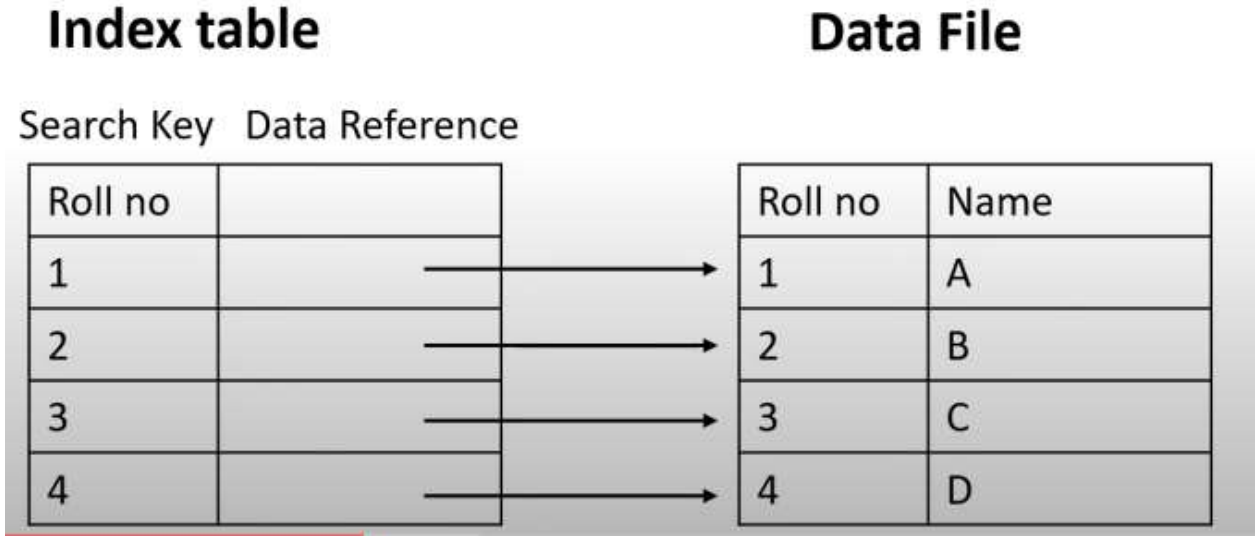
Name	
B	
A	
D	
A	
B	
F	

Primary Index

- If the index is created on the basis of the primary key of the table, then it is known as primary indexing. These primary keys are unique to each record and contains 1:1 relation between the records.
- As primary keys are stored in sorted order, the performance of the searching operation is quite efficient.
- The primary index can be classified into two types
  - Dense index
  - Sparse index

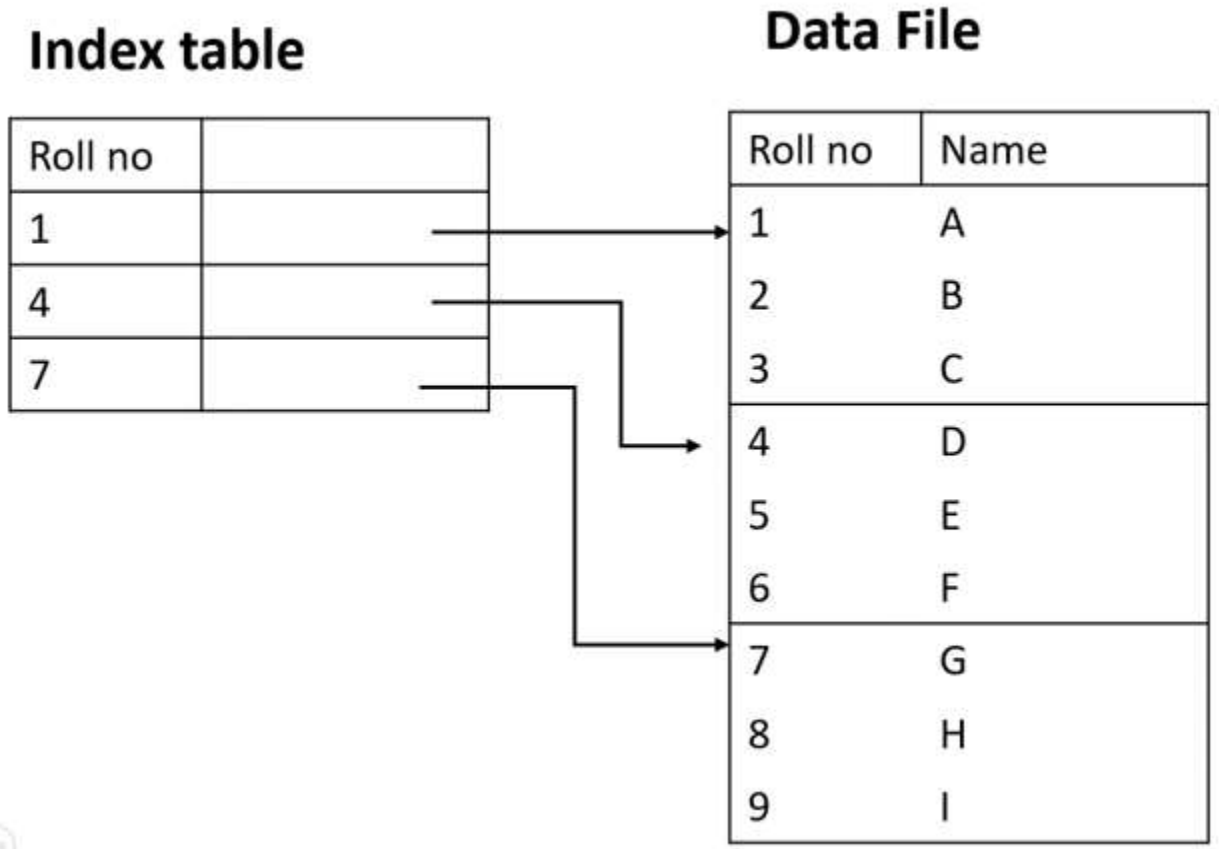
Dense index

- The dense index contains an index record for every search key value in the data file. It makes searching faster
- In this, the number of records in the index table is same as the number of records in the main table
- It needs more space to store index record itself. The index records have the search key and a pointer to the actual record on the disk



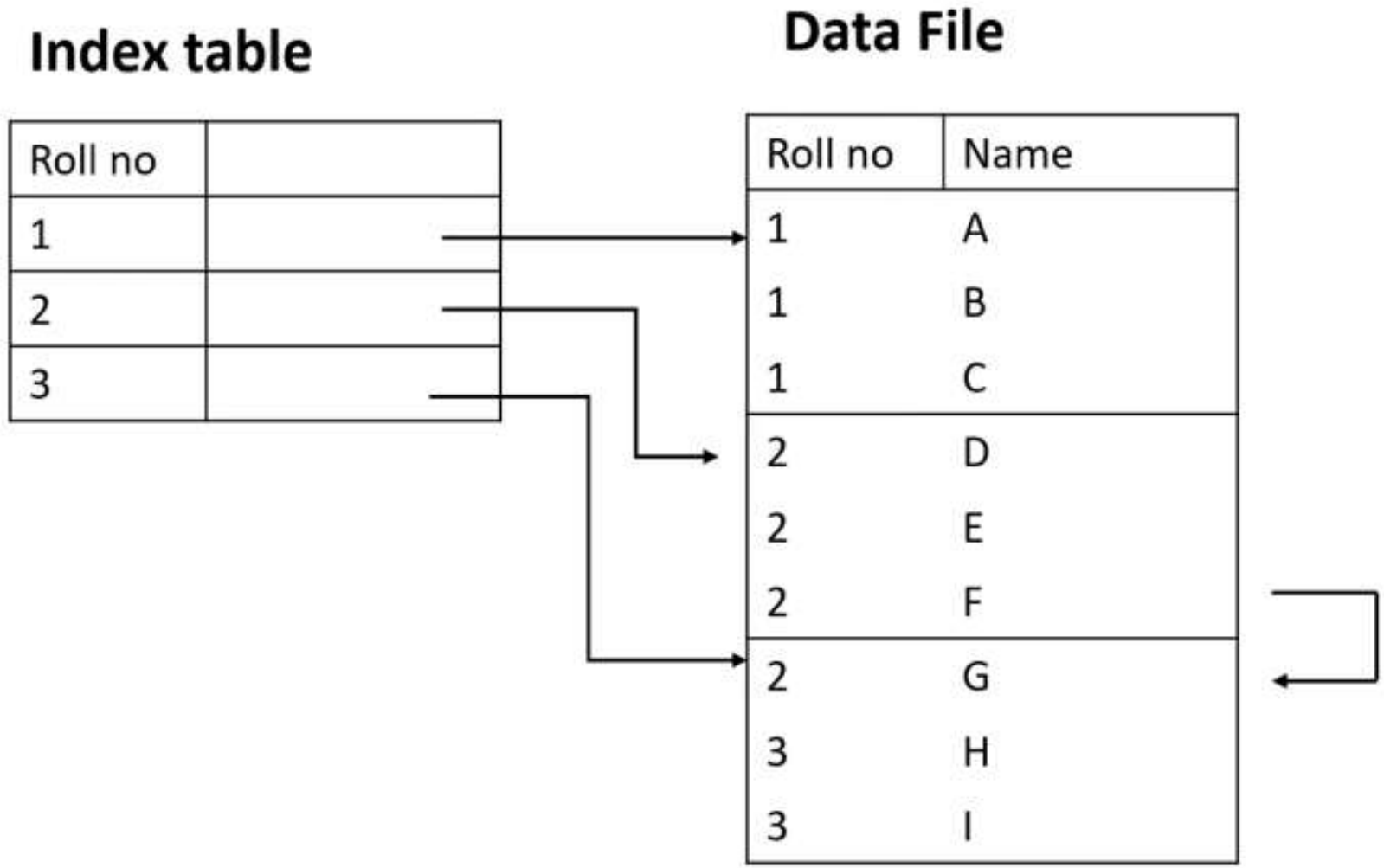
Sparse index

- In the data file, index records appears only for a few items. Each item points to a block. In this, instead of pointing to each record in the main table, the index points to the records in the main table in a gap.



Cluster index

- A clustered index can be defined as an ordered data file. Sometimes the index is created primary key columns which may not be unique for each record.
- In this case, to identify the record faster, we will group two or more columns to get the unique value and create index out of them. This method is called a clustering index.
- The records which have similar characteristics are grouped, and indexes are created for these group.

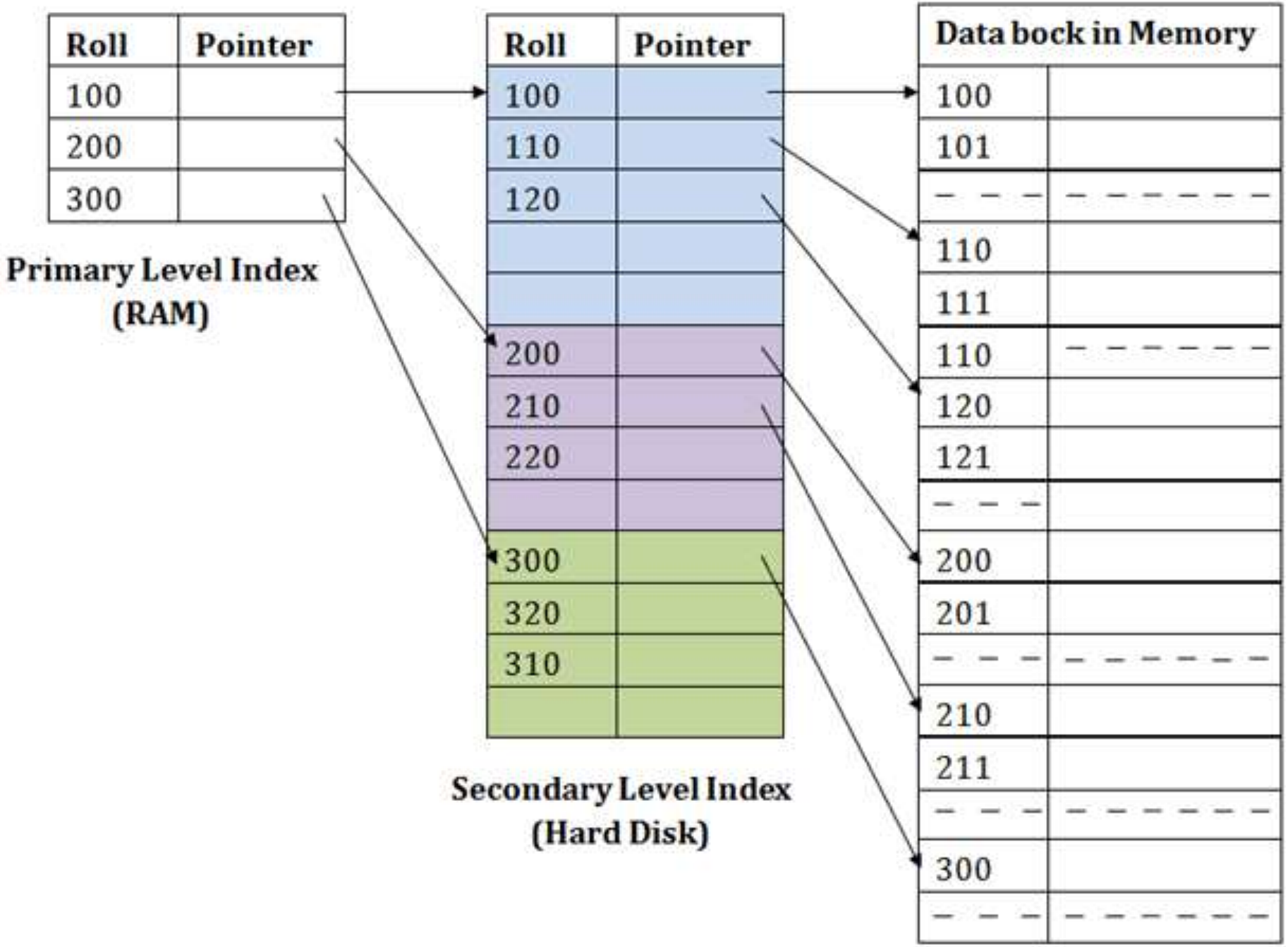

Index table
Roll no
1
2
3
Data File
Roll no
Name
1 A
1 B
1 C
2 D
2 E
2 F
2 G
3 H
3 I

Secondary index

- In the sparse indexing, as the size of the table grows, the size of mapping also grows.
- These mappings are usually kept in the primary memory so that address fetch should be faster.
- Then the secondary memory searches the actual data based on the address got from mapping.
- If the mapping size grows then fetching the address itself becomes slower. In this case, the sparse index will not be efficient.
- To overcome this problem, secondary indexing is introduced.

Secondary index

- In secondary indexing, to reduce the size of mapping, another level of indexing is introduced.
- In this method, the huge range for the columns is selected initially so that the mapping size of the first level becomes small.
- Then each range is further divided into smaller ranges. The mapping of the first level is stored in the primary memory, so that address fetch is faster.
- The mapping of the second level and actual data are stored in the secondary memory (hard disk).

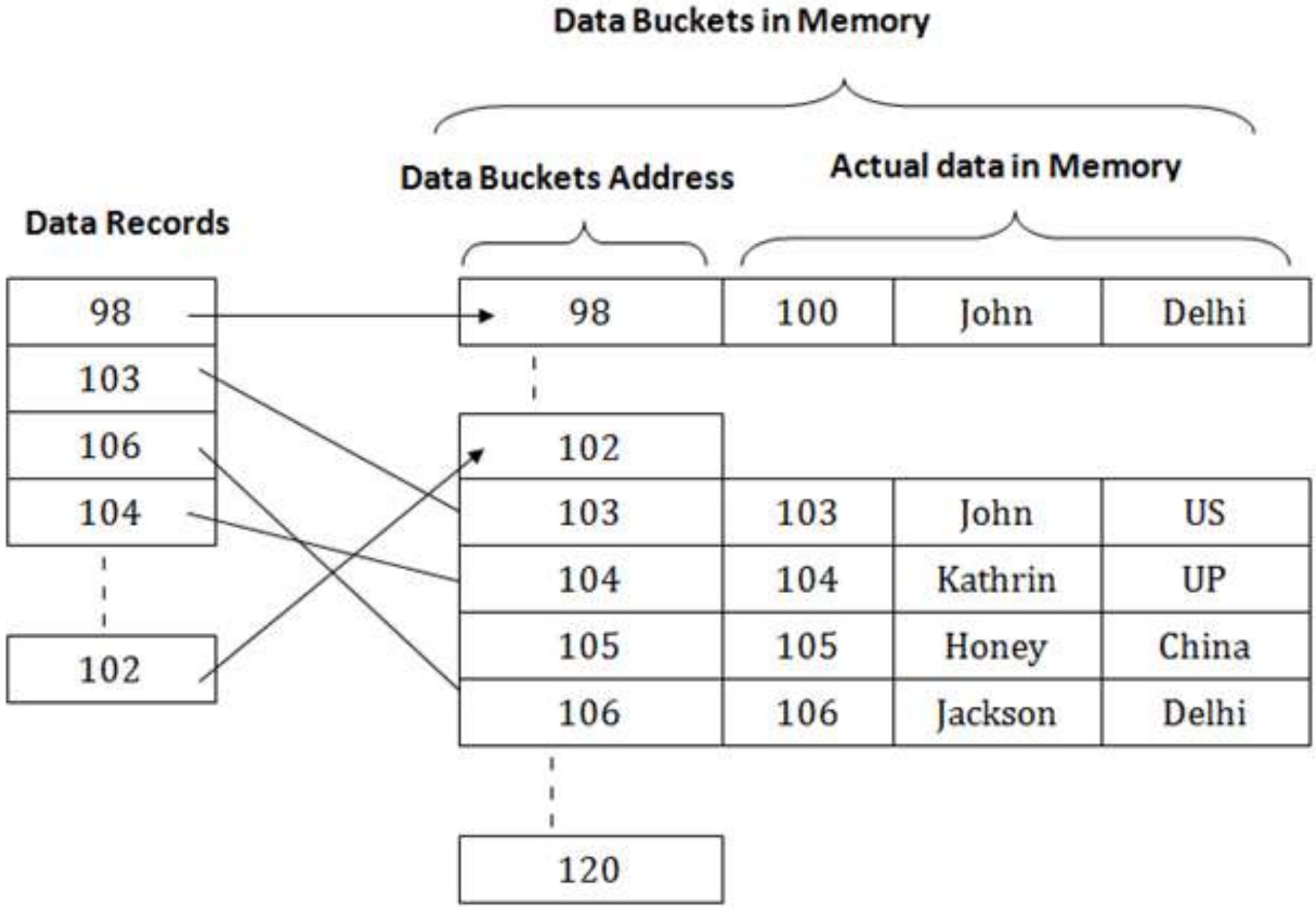

Roll
Pointer
100
200
300
Primary Level Index
(RAM)
Roll
Pointer
100
110
120
200
210
220
300
320
310
Secondary Level Index
(Hard Disk)
Data bock in Memory
100
101
110
111
110
120
121
200
201
210
211
300

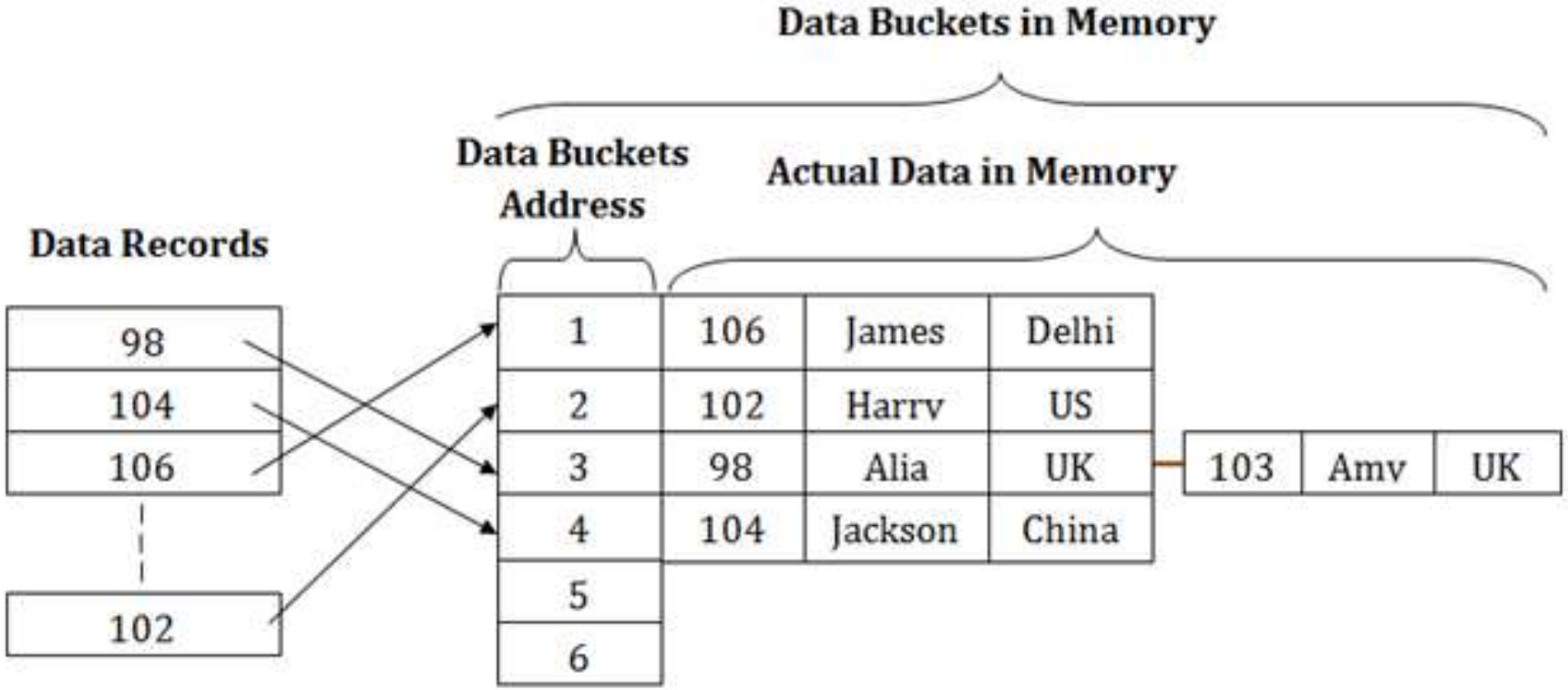
For example:

- If you want to find the record of roll 111 in the diagram, then it will search the highest entry which is smaller than or equal to 111 in the first level index. It will get 100 at this level.
- Then in the second index level, again it does max (111) <= 111 and gets 110. Now using the address 110, it goes to the data block and starts searching each record till it gets 111.
- This is how a search is performed in this method. Inserting, updating or deleting is also done in the same manner.

Hash based Indexing

- In a huge database structure, it is very inefficient to search all the index values and reach the desired data.
- Hashing technique is used to calculate the direct location of a data record on the disk without using index structure.
- In this, a hash function can choose any of the column value to generate the address.
- Most of the time, the hash function uses the primary key to generate the address of the data block.
- A hash function is a simple mathematical function to any complex mathematical function. We can even consider the primary key itself as the address of the data block.
- That means each row whose address will be the same as a primary key stored in the data block.

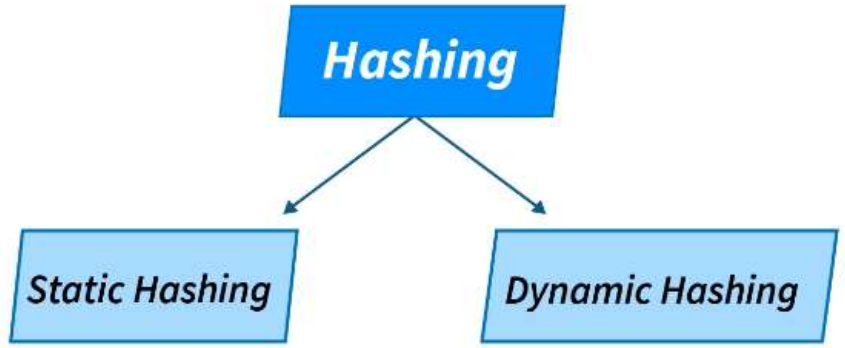

Data Buckets in Memory
Data Buckets Address
Actual data in Memory
Data Records
98
103
106
104
102
98
100
John
Delhi
102
103
103
John
US
104
104
Kathrin
UP
105
105
Honey
China
106
106
Jackson
Delhi
120

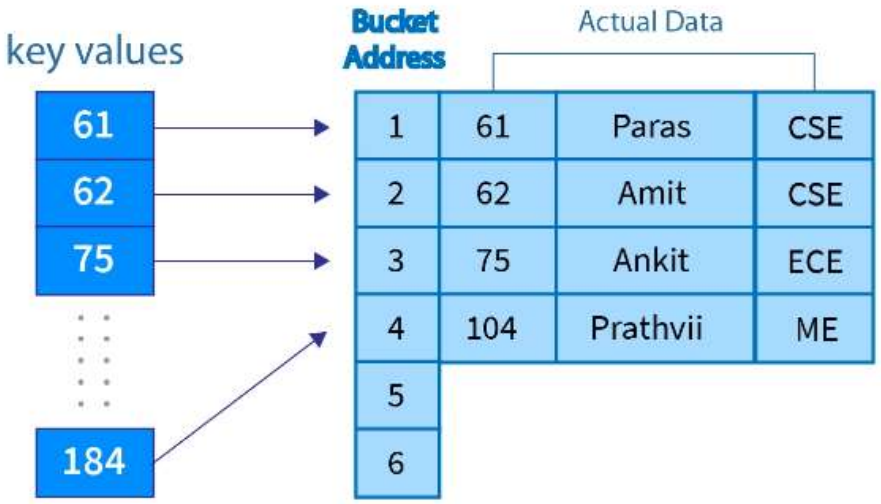


Suppose we have mod (5) hash function to determine the address of the data block. In this case, it applies mod (5) hash function on the primary keys and generates 3, 3, 1, 4 and 2 respectively, and records are stored in those data block addresses.

Types of Hashing

- Static hashing: In static hashing, for a key *k*, hash function ***h*(*x*)** always generates the same memory address. For example, if ***h*(x)=*x*%7** then for any ***x*** the value will always be the same.
- Thus if we generate the address for a key **107**, then ***h*(*x*)** will always return the same bucket address .,*i.e.*., 3. There will be no change in the bucket address. Hence the number of buckets in the memory always remains constant.





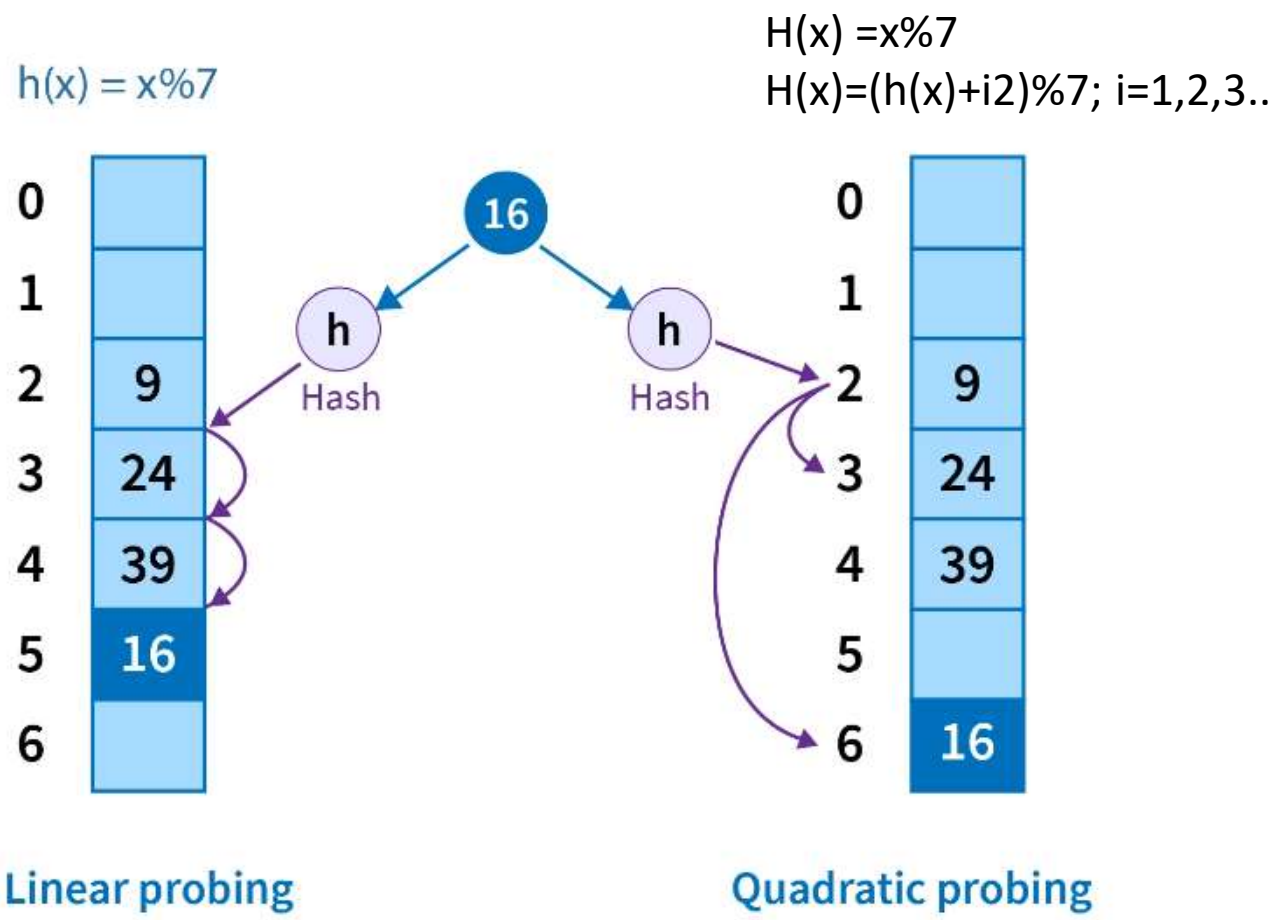
Operations in Static Hashing

- **Insertion -** To insert a new record in the table we will calculate the bucket address for search key *k*, and the record will be stored there.
- **Search -** To search a record we will use the same hash function to get the bucket address of the record and retrieve the data from there.
- **Deletion -** To delete a record from the table we will firstly search for that record and then simply delete it from that location.

Static hashing can also be further divided to open and closed hashing.

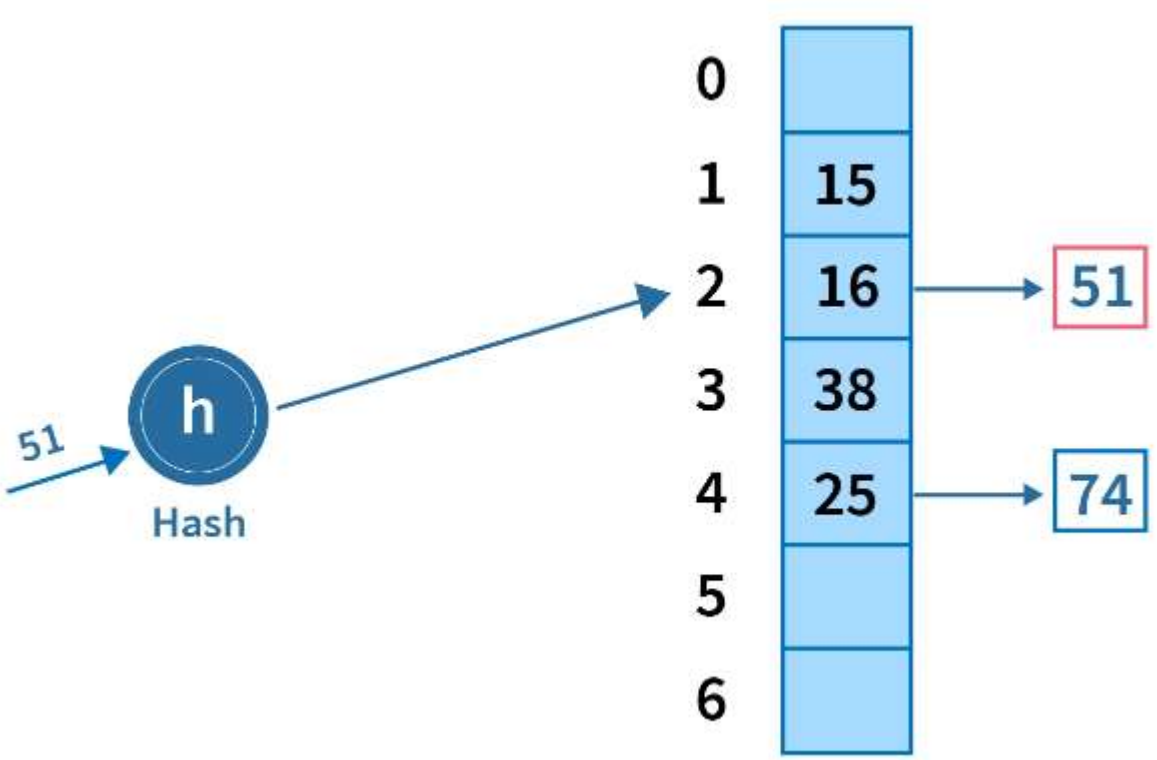
Open hashing

- In open hashing, whenever a collision occurs, we probe for the next empty bucket to enter the new record. The order of checking are majorly of two types -
- Linear Probing
- Quadratic Probing



Closed Hashing

- In closed hashing, the collision condition is handled by linking the new record after the previous one, due to which is also termed as "Hashing with separate chaining".



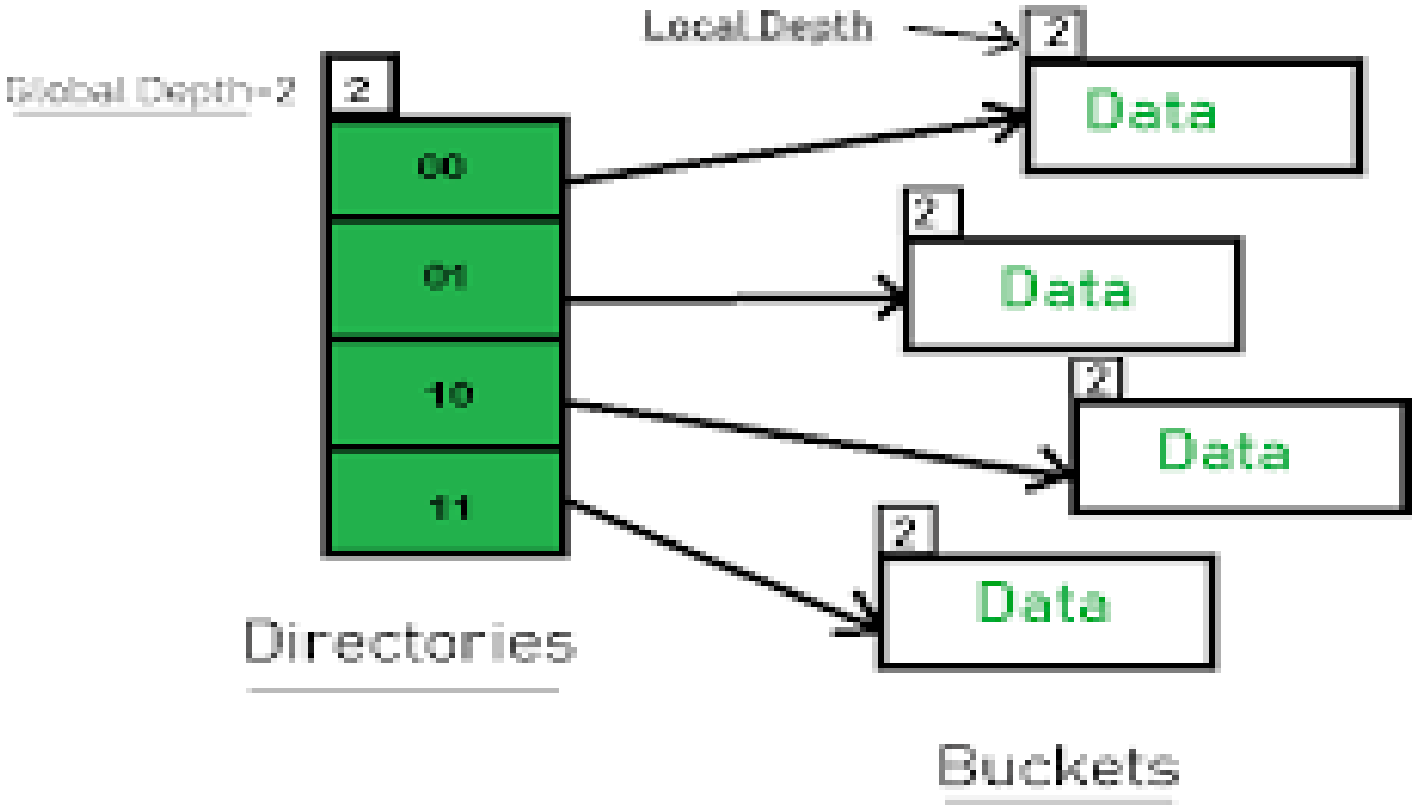
Dynamic Hashing/Extendible Hashing

- The dynamic hashing method is used to overcome the problems of static hashing like bucket overflow.
- In this method, data buckets grow or shrink as the records increases or decreases. This method is also known as Extendable hashing method.

Features:

- Directories: The directories store addresses of the buckets in pointers. An id is assigned to each directory which may change each time when Directory Expansion takes place.
- Buckets: The buckets are used to hash the actual data.

Basic Structure of Extendible Hashing



Extendible Hashing

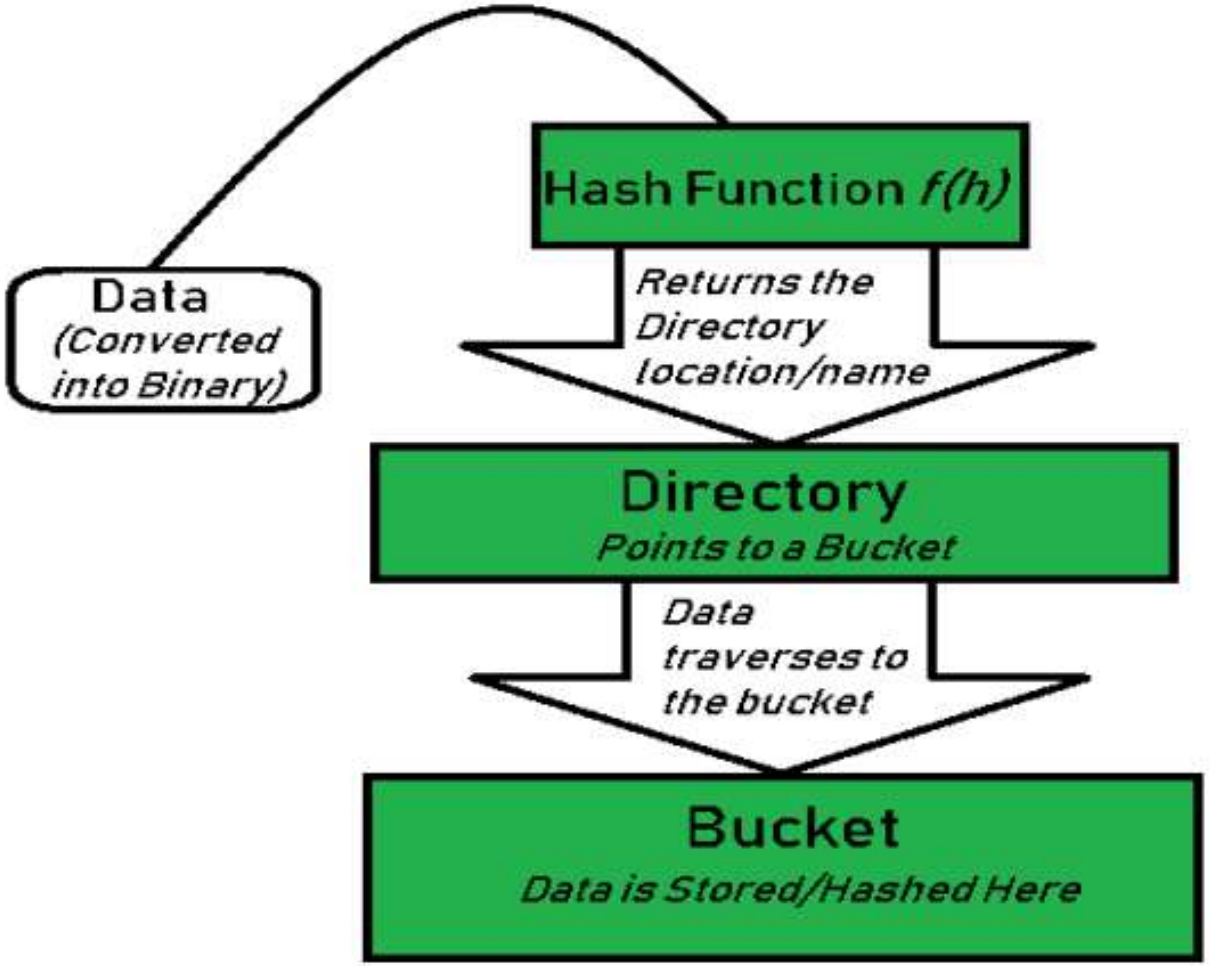
Terminologies

- Directories: These containers store pointers to buckets. Each directory is given a unique id which may change each time when expansion takes place. The hash function returns this directory id which is used to navigate to the appropriate bucket. Number of Directories = 2^Global Depth.
- Buckets: They store the hashed keys. Directories point to buckets. A bucket may contain more than one pointers to it if its local depth is less than the global depth.
- Global Depth: It is associated with the Directories. They denote the number of bits which are used by the hash function to categorize the keys. Global Depth = Number of bits in directory id.

Terminologies

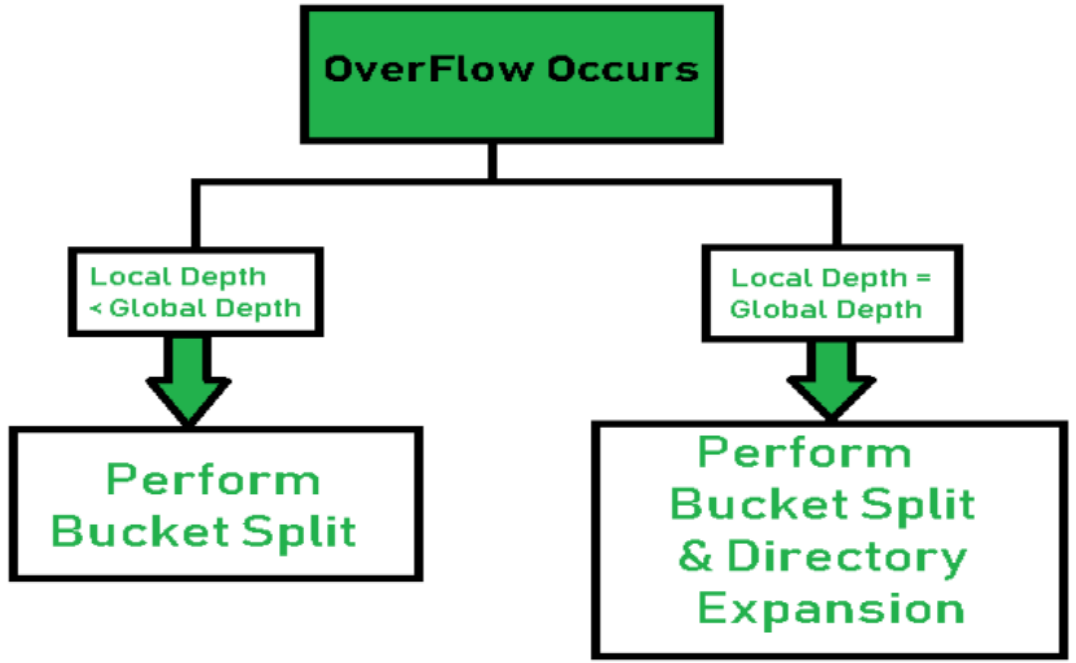
- Local Depth: It is the same as that of Global Depth except for the fact that Local Depth is associated with the buckets and not the directories. Local depth in accordance with the global depth is used to decide the action that to be performed in case an overflow occurs. Local Depth is always less than or equal to the Global Depth.
- Bucket Splitting: When the number of elements in a bucket exceeds a particular size, then the bucket is split into two parts.
- Directory Expansion: Directory Expansion Takes place when a bucket overflows. Directory Expansion is performed when the local depth of the overflowing bucket is equal to the global depth.

Basic working of extendible hashing



Algorithm

- Step 1 – Analyze Data Elements
- Step 2 – Convert into binary format
- Step 3 – Check Global Depth of the directory
- Step 4 – Identify the Directory
- Step 5 – Navigation
- Step 6 – Insertion and Overflow Check
- Step 7 – Tackling Over Flow Condition during Data Insertion:
- Step 8 – Rehashing of Split Bucket Elements
- Step 9 – The element is successfully hashed.



How to search a key:

- First, calculate the hash address of the key.
- Check how many bits are used in the directory, and these bits are called as i.
- Take the least significant i bits of the hash address. This gives an index of the directory.
- Now using the index, go to the directory and find bucket address where the record might be.

How to insert a new record

- Firstly, you have to follow the same procedure for retrieval, ending up in some bucket.
- If there is still space in that bucket, then place the record in it.
- If the bucket is full, then we will split the bucket and redistribute the records.

Extendible Hashing / Dynamic Hashing

Let's take an example :

12, 8, 4, 10, 9, 16, 7, 5, 30

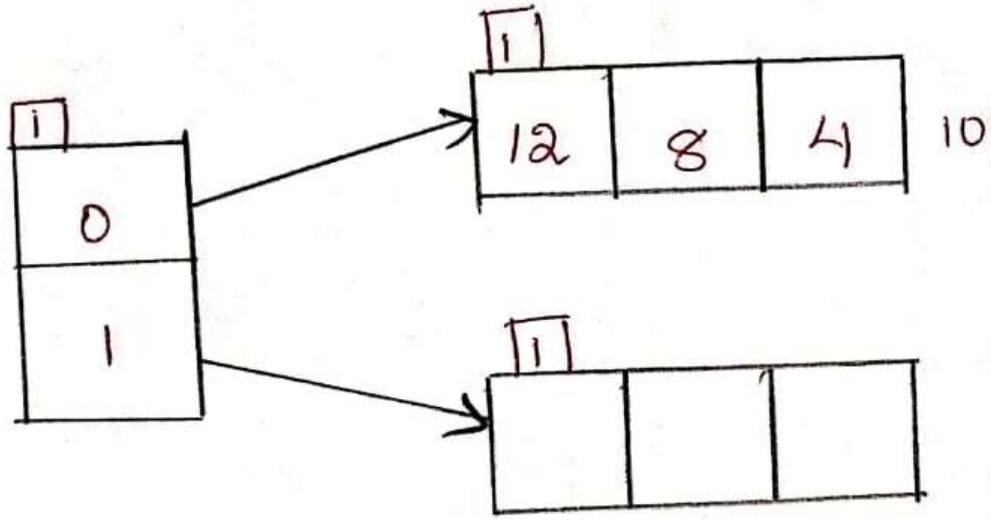
	2^4 2^3 2^2 2^1 2^0
12	0 1 1 0 0
8	0 1 0 0 0
4	0 0 1 0 0
10	0 1 0 1 0

9	0 1 0 0 1
16	1 0 0 0 0
7	0 0 1 1 1
5	0 0 1 0 1

30	1 1 1 1 0

Global depth = 1

Local depth = 1



Insert : 12

(01100)

Insert (8):

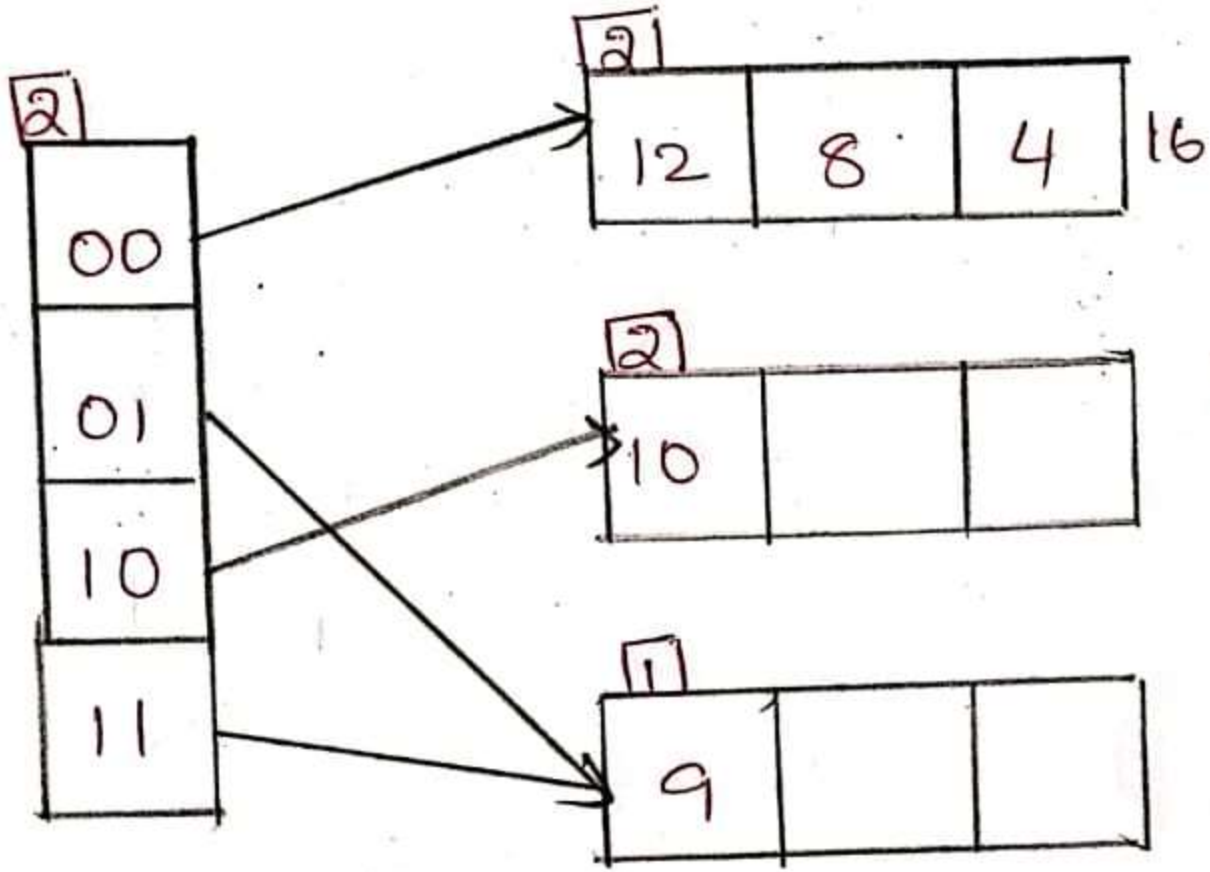
01000

Insert (4): 00100

Insert (10): 01010

Here 10 is not inserting to the bucket due to overflow condition ie, bucket size = 3., Now split the table (Bucket) by changing the global depth = 2.

Step 2 :



Insert(12) : 01100

Insert(8) : 01000

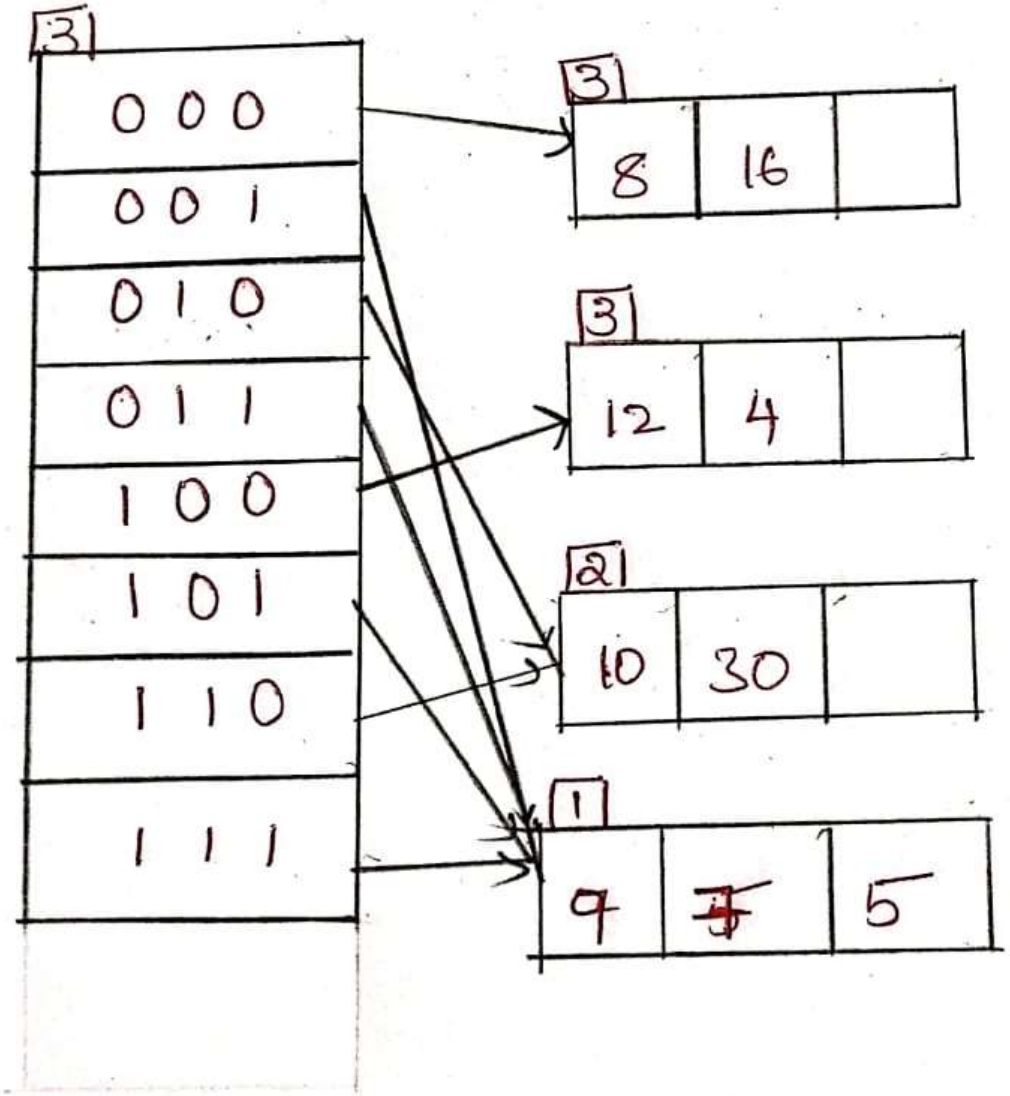
Insert(4) : 00100

Insert(10) : 01010

Insert(9) : 01001

Insert(16) : 10000

while Inserting (16) again the bucket get Overflow.
So split the bucket by changing the global depth = 3.



Insert(12) : 01100

Insert(8) : 01000

Insert(4) : 00100

Insert(10) : 01010

Insert(9) : 01001

Insert(16) : 10000

Insert(7) : 00111

Insert(5) : 00101

Insert(30) : 11110

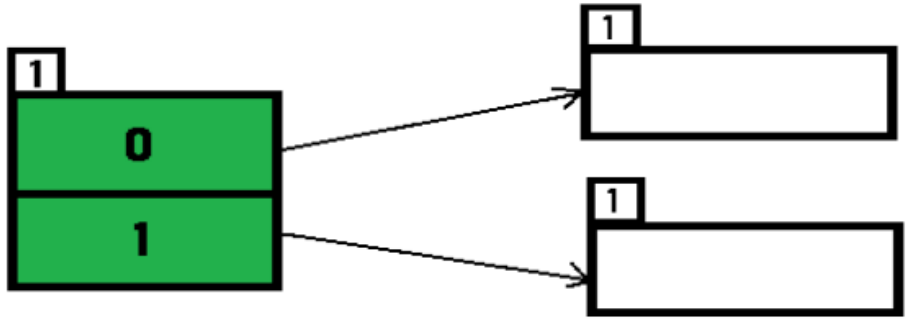
Example2:

Consider:
16,4,6,22,24,10,31,7,9,20,26.

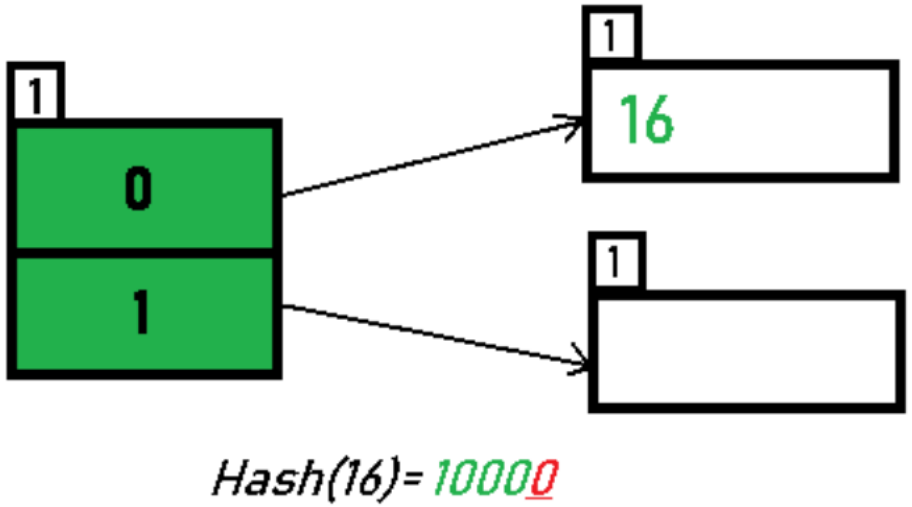
[Bucket Size: 3]

Key	Binary conversion
16	10000
4	00100
6	00110
22	10110
24	11000
10	01010
31	11111
7	00111
9	01001
20	10100
26	11010

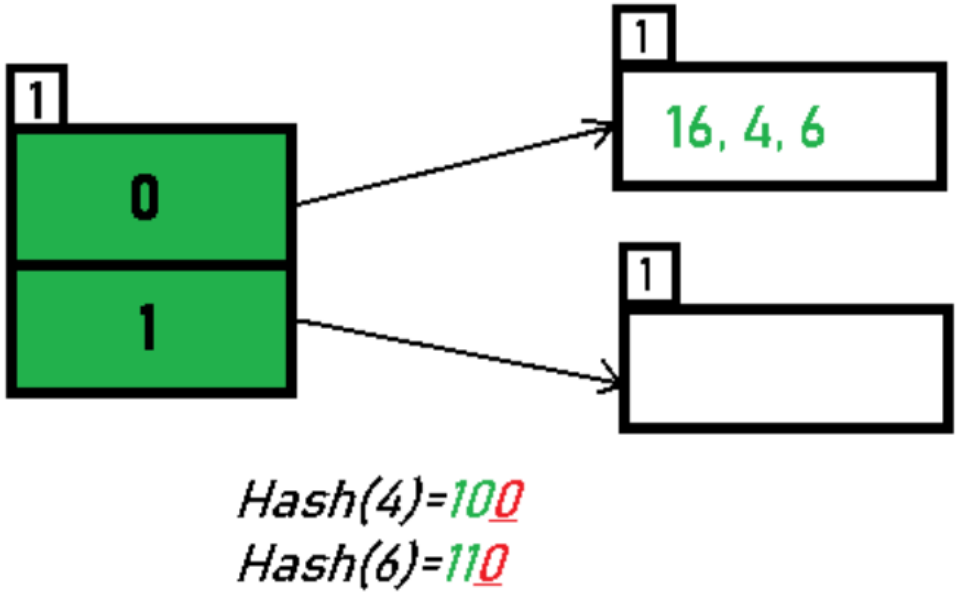
Initially, the global-depth and local-depth is always 1. Thus, the hashing frame looks like this

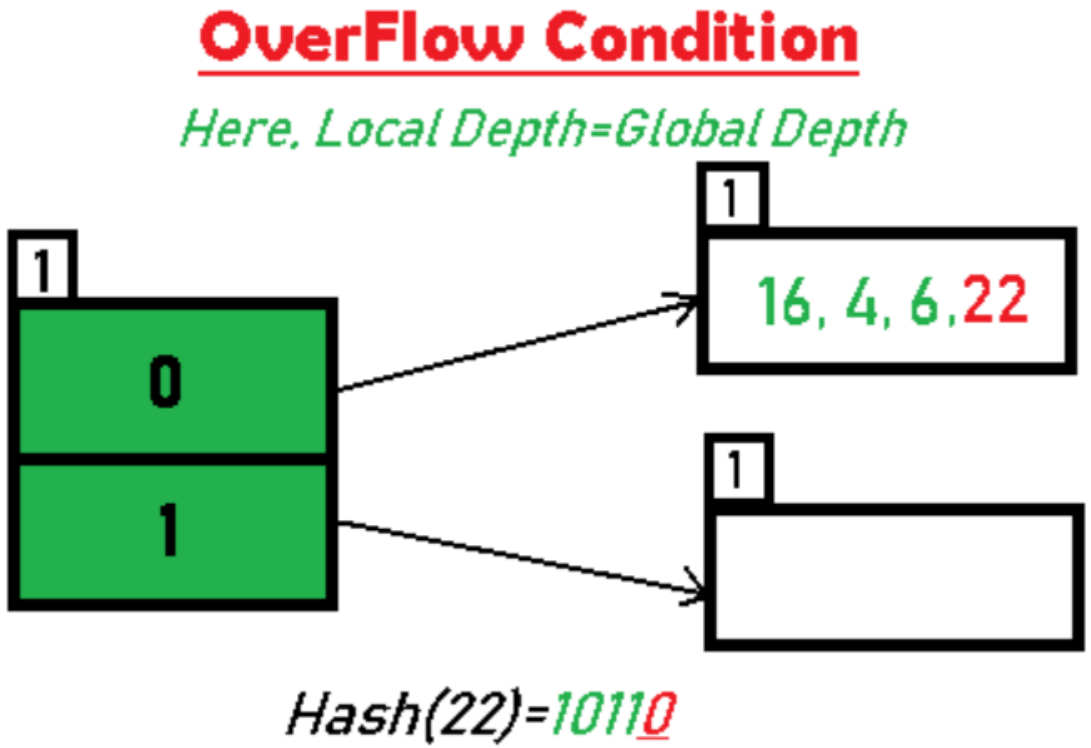


Inserting 16: The binary format of 16 is 10000 and global-depth is 1. The hash function returns 1 LSB of 10000 which is 0. Hence, 16 is mapped to the directory with id=0.

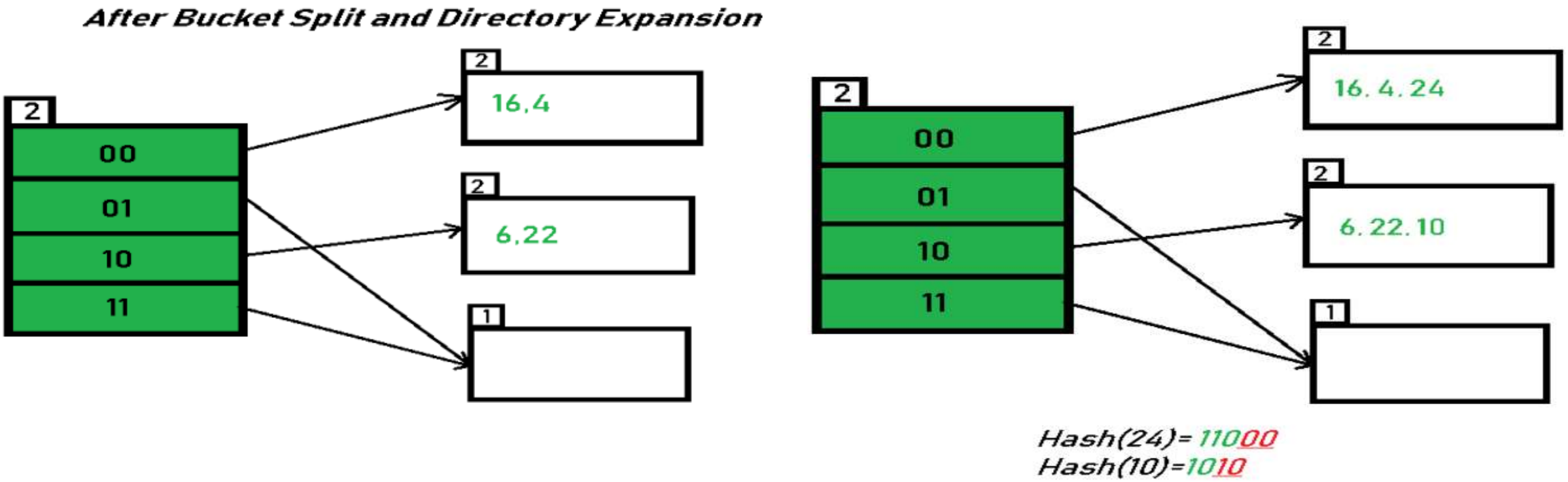


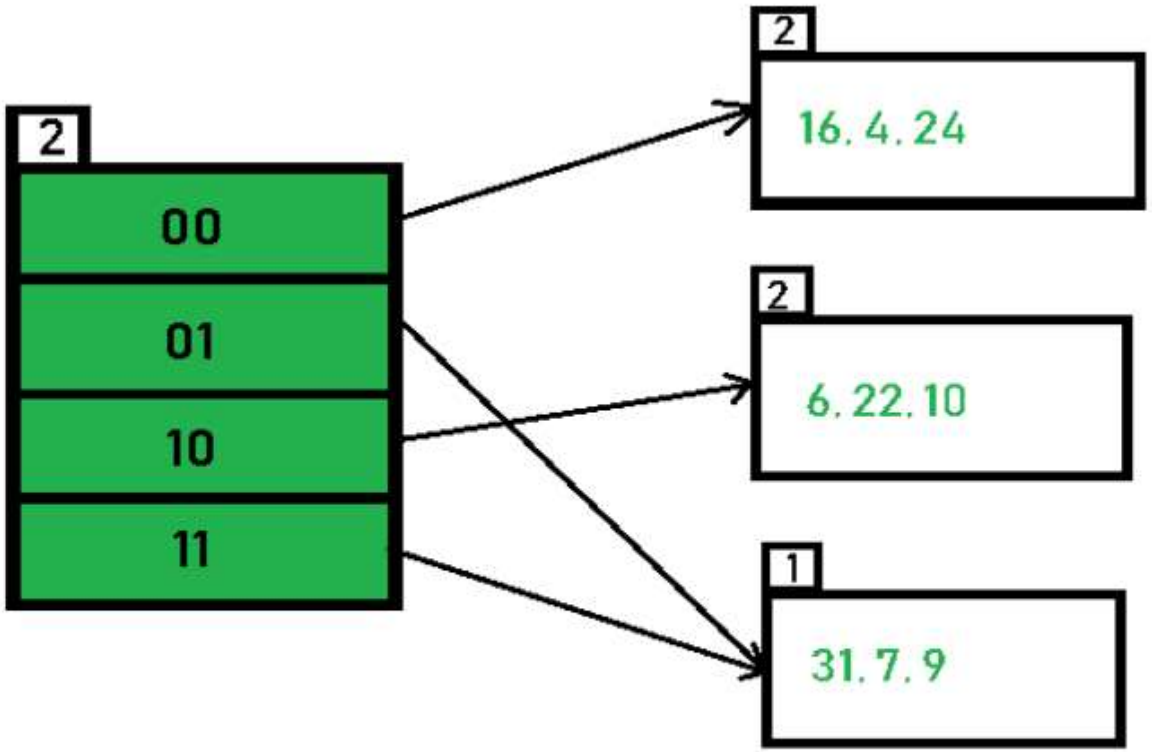
Inserting 4 and 6: Both 4(100) and 6(110)have 0 in their LSB. Hence, they are hashed as follows:





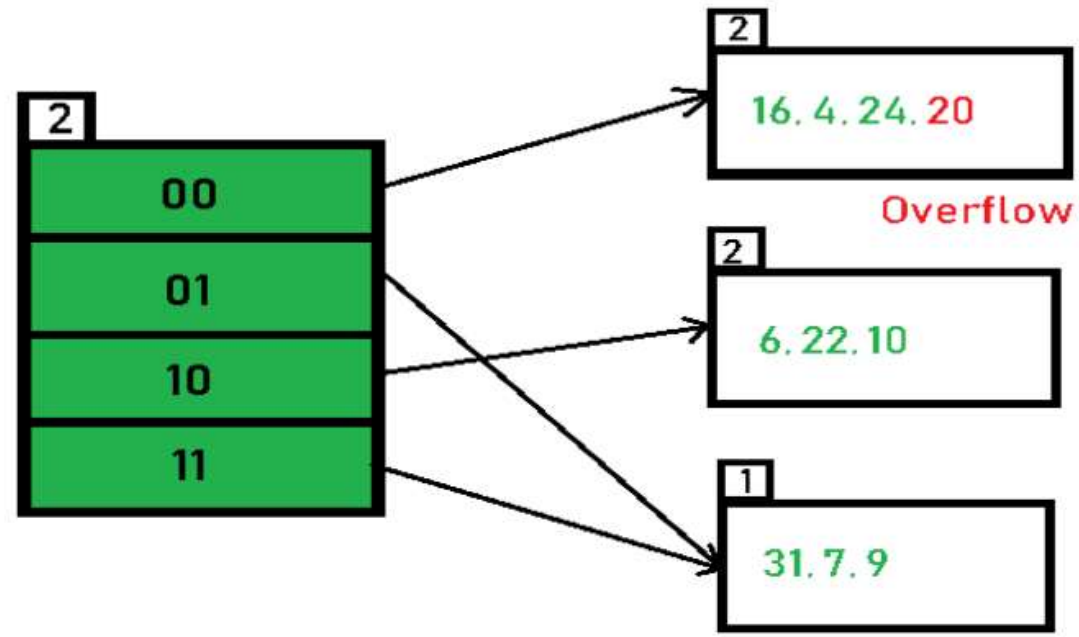
16,4,6,22 are now rehashed w.r.t 2 LSBs.
[16(10000),4(100),6(110),22(10110)]



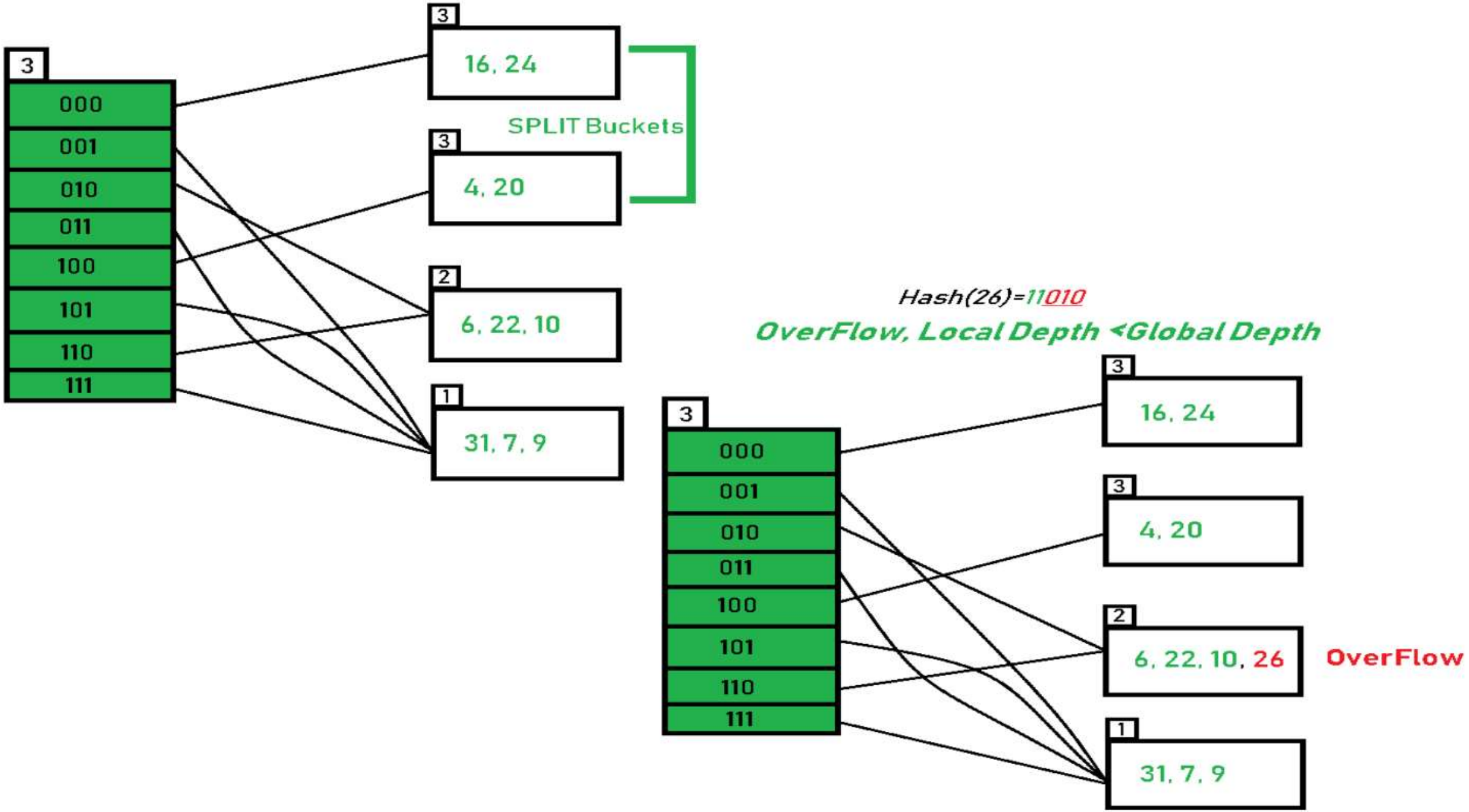


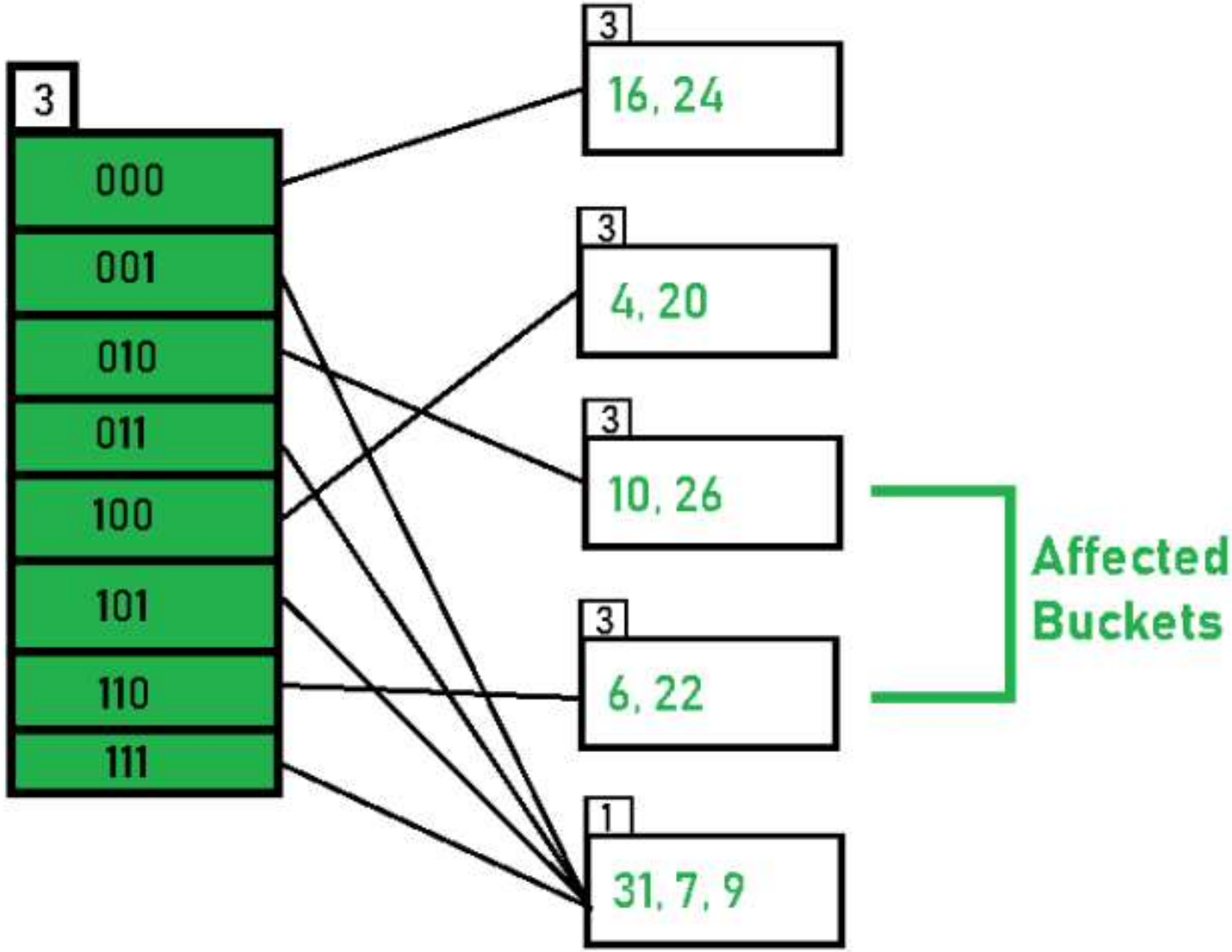
Hash(31)= 11111
Hash(7)= 111
Hash(9)= 1001

OverFlow, Local Depth= Global Depth



Hash(20)=10100


3
000
001
010
011
100
101
110
111
3
16, 24
SPLIT Buckets
3
4, 20
2
6, 22, 10
1
31, 7, 9
Hash(26)=11010
OverFlow, Local Depth <Global Depth
3
000
001
010
011
100
101
110
111
3
16, 24
3
4, 20
2
6, 22, 10, 26
OverFlow
1
31, 7, 9



Hashing of 11 Numbers is Thus Completed.

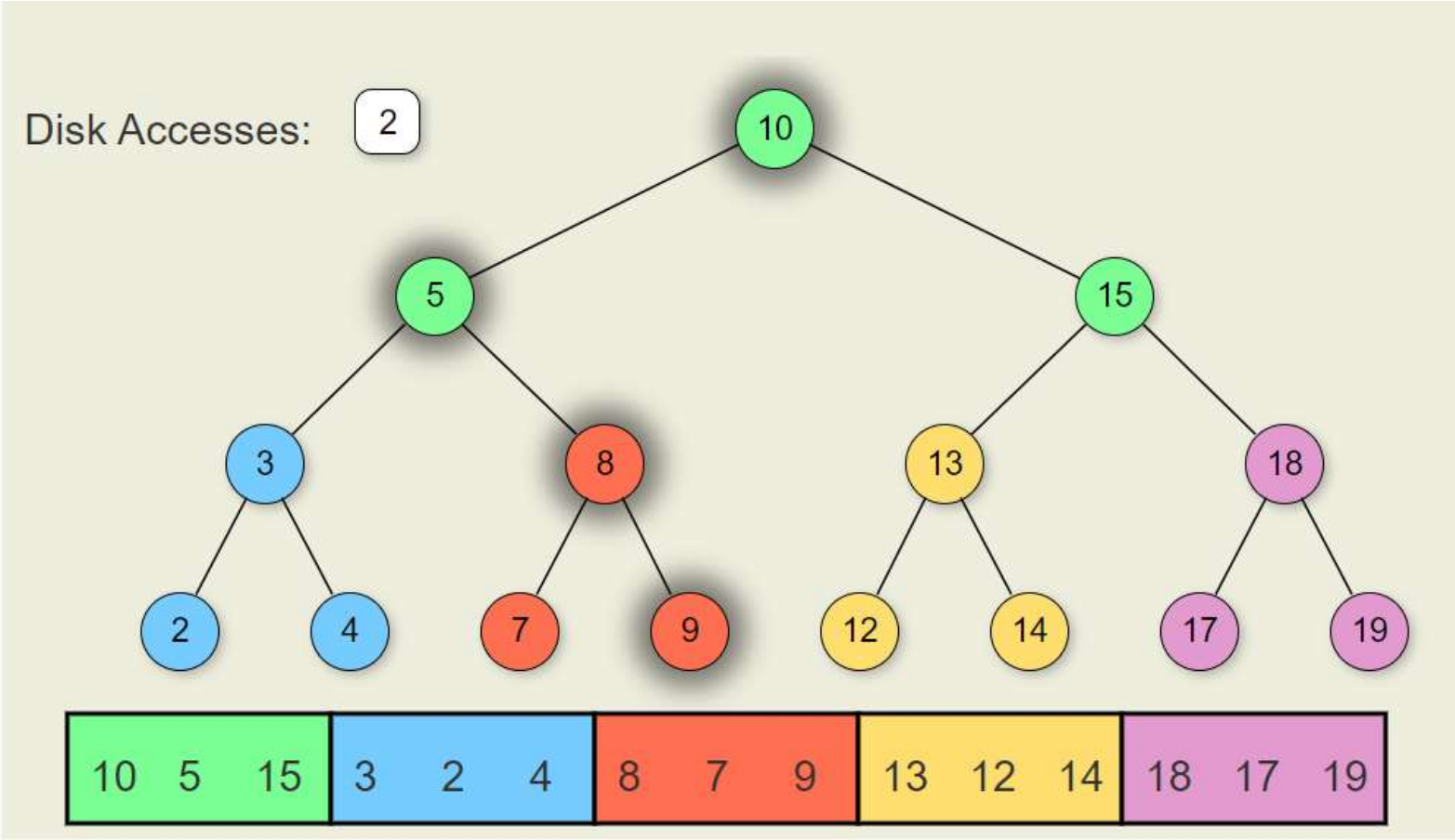
Tree Based Index

- Linear indexing is efficient when the database is static, that is, when records are inserted and deleted rarely or never.
- ISAM (Indexed sequential access method) is adequate for a limited number of updates, but not for frequent changes. Because it has essentially two levels of indexing, ISAM will also break down for a truly large database where the number of cylinders is too great for the top-level index to fit in main memory.
- In their most general form, database applications have the following characteristics:

1. Large sets of records that are frequently updated.
2. Search is by one or a combination of several keys.
3. Key range queries or min/max queries are used.

- For such databases, a better organization must be found. One approach would be to use the binary search tree (BST) to store primary and secondary key indices.
- BSTs can store duplicate key values, they provide efficient insertion and deletion as well as efficient search, and they can perform efficient range queries.
- When there is enough main memory, the BST is a viable option for implementing both primary and secondary key indices. Unfortunately, the BST can become unbalanced.
- We can solve these problems by selecting another tree structure that automatically remains balanced after updates, and which is amenable to storing in blocks i.e, Splay trees, 2-3 Tree, B tree, B+ trees ect.

Definition : Tree-based indexing involves structuring indexes in the form of a tree, where nodes represent data records and edges signify navigational paths. This hierarchy allows DBMSs to minimize the number of disk accesses required during searches, updates, deletions, and insertions.



B+ Trees

- A B+ Tree is simply a balanced binary search tree, in which all data is stored in the leaf nodes, while the internal nodes store just the indices.
- Each leaf is at the same height and all leaf nodes have links to the other leaf nodes. The root node always has a minimum of two children.
- The concept of **B+ trees** exists because of it being much more convenient, both in terms of operations on it, as well as efficiency.
- There are three operations in B+ trees such as Insertion, Deletion and Search

Properties of B+ Trees:

1. All data is stored in the leaf nodes, while the internal nodes store just the indices.
2. Each leaf is at the same height.
3. All leaf nodes have links to the other leaf nodes.
4. The root node has a minimum of two children.
5. Each node except root can have a maximum of m children and a minimum of m/2 children.
6. Each node can contain a maximum of m-1 keys and a minimum of ⌈m/2⌉ - 1 keys.

Insertion

B^+ Tree creation :-

consider the list of elements to construct B+Tree

7, 10, 1, 23, 5, 15, 17, 9, 11, 39, 35, 8, 40, 25

Order = 5 (m)

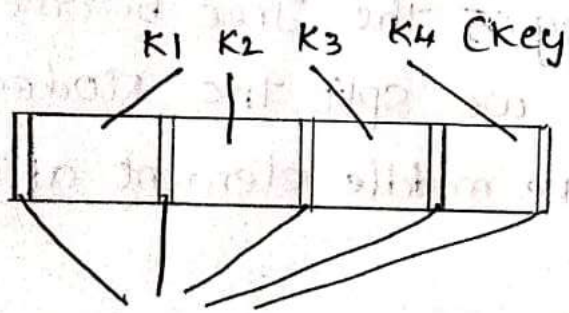
Max Children = 5

Min Children = $m/2 = 3$

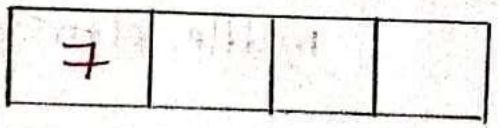
Max Keys : $m-1 = 4$

Min Keys = $\left[\frac{m}{2}\right] - 1 = 2$

Step 1 :- Insert (7)

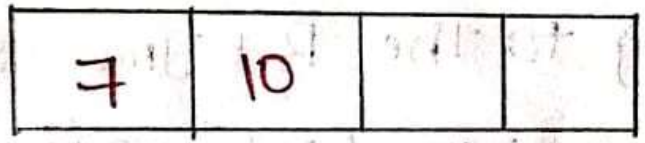


5 pointers because of 5 childrens

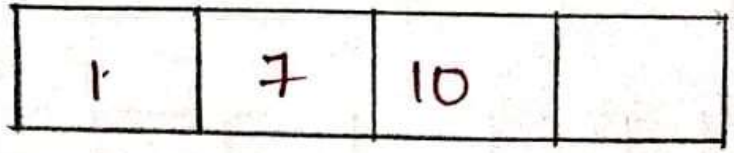


While inserting the keys into the structure, always follows Binary Search Tree property.

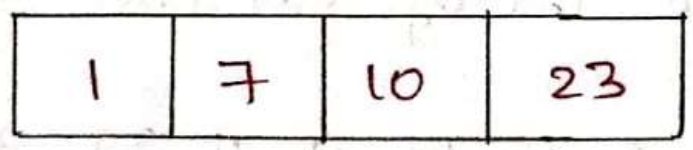
step2 : Insert (10) : [10 > 7] placed at right



step 3 : Insert (1) : [1 < 7] [1 < 10]

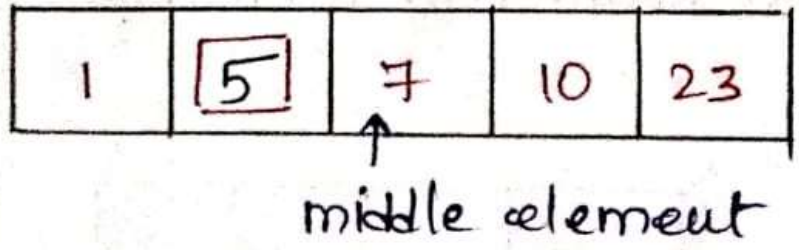


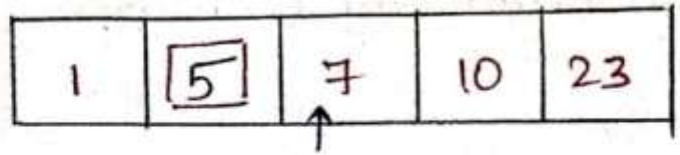
Step 4 :- Insert (23)

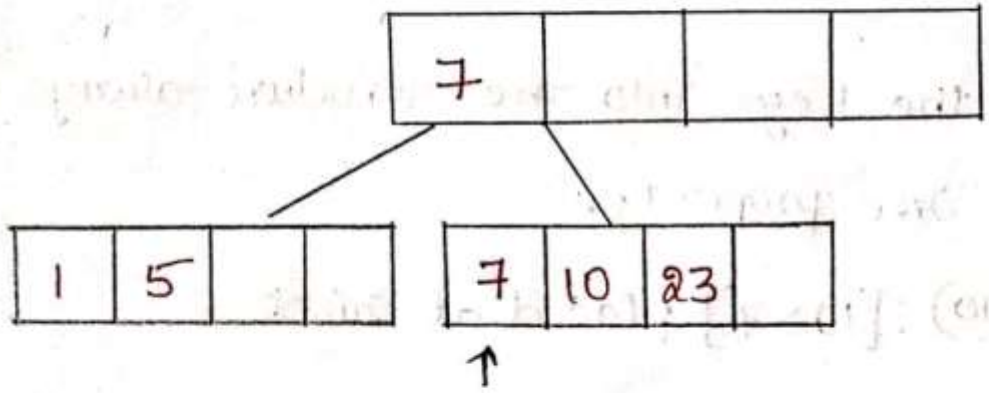


Step 5 : Insert (5):

Here, after inserting 5 the Tree become overflow / Size is overflow, so we split the Nodes by taking middle element. [Take middle element after inserting element 5.

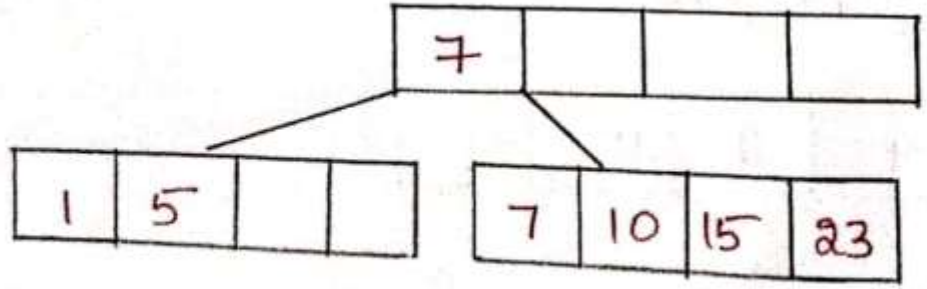




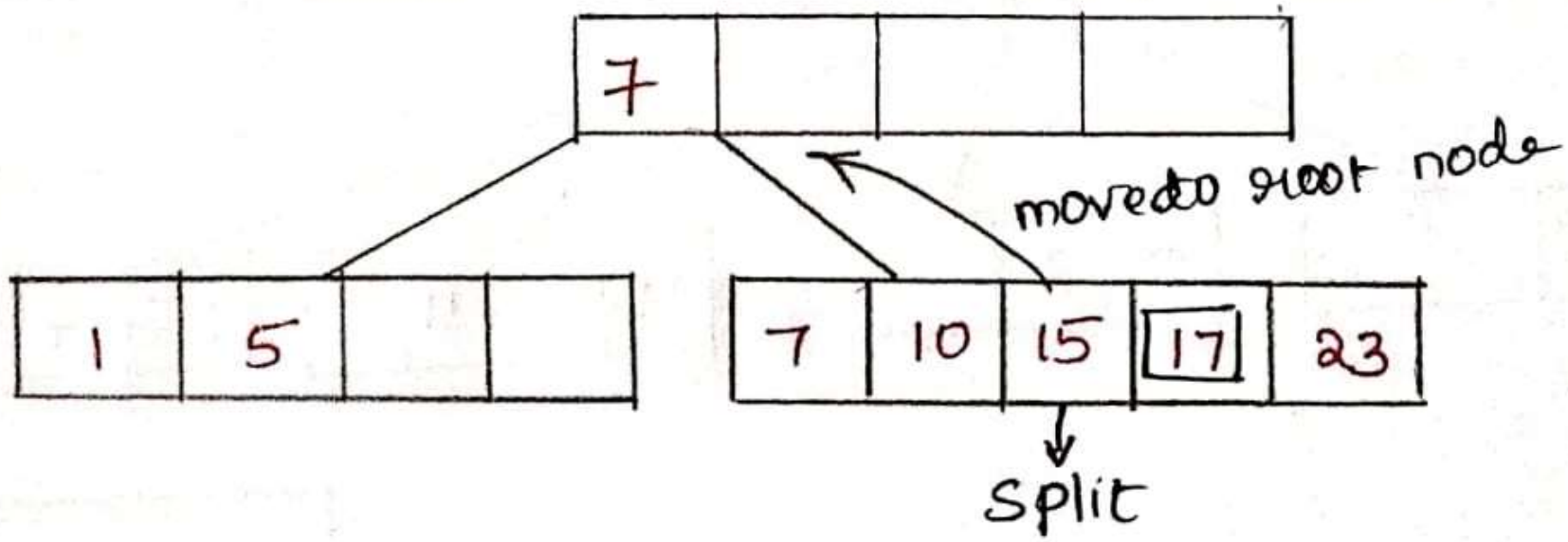


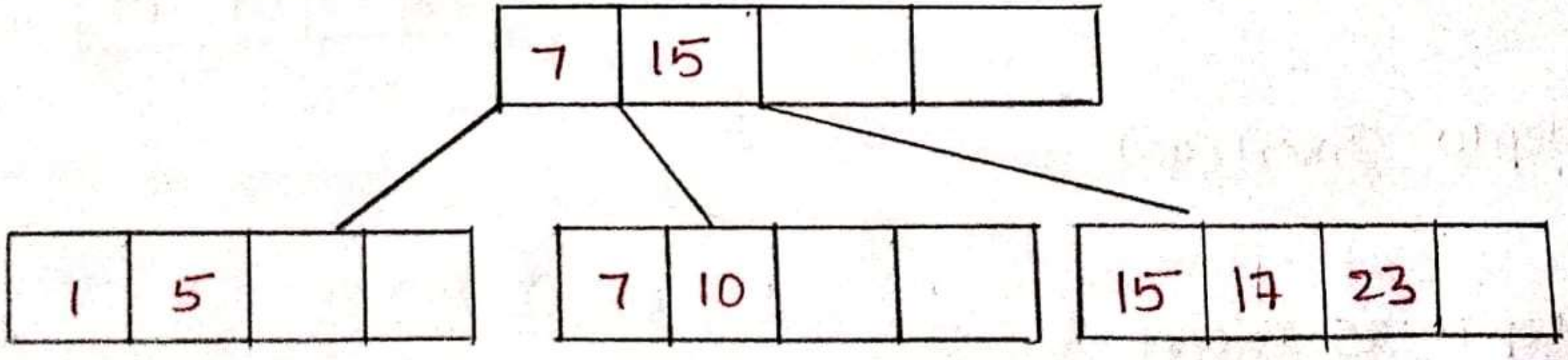
According to the B+Tree rule the data should be present in leaf node.

Step6: Insert (15): Data is always present in leaf node

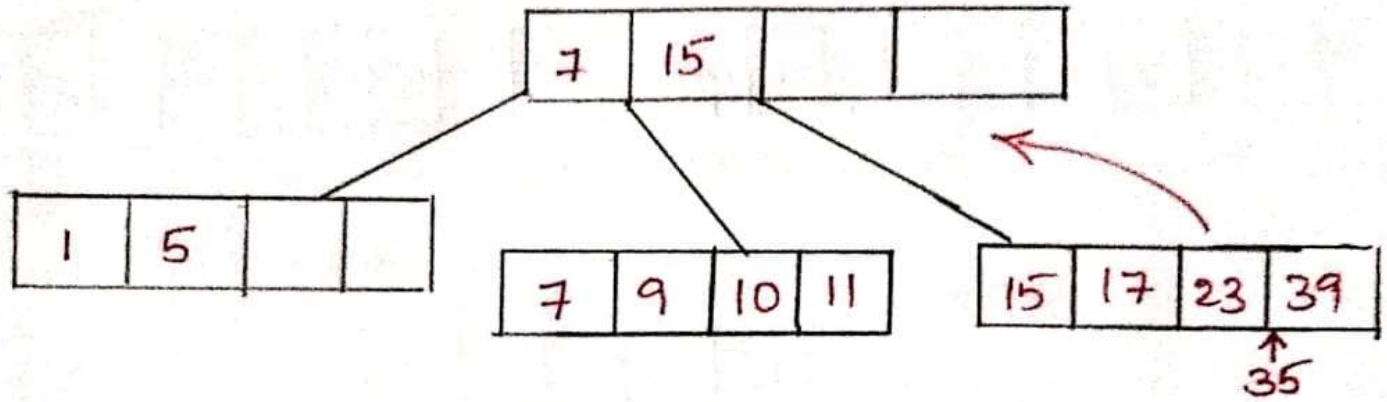


Step 7 : Insert (17)

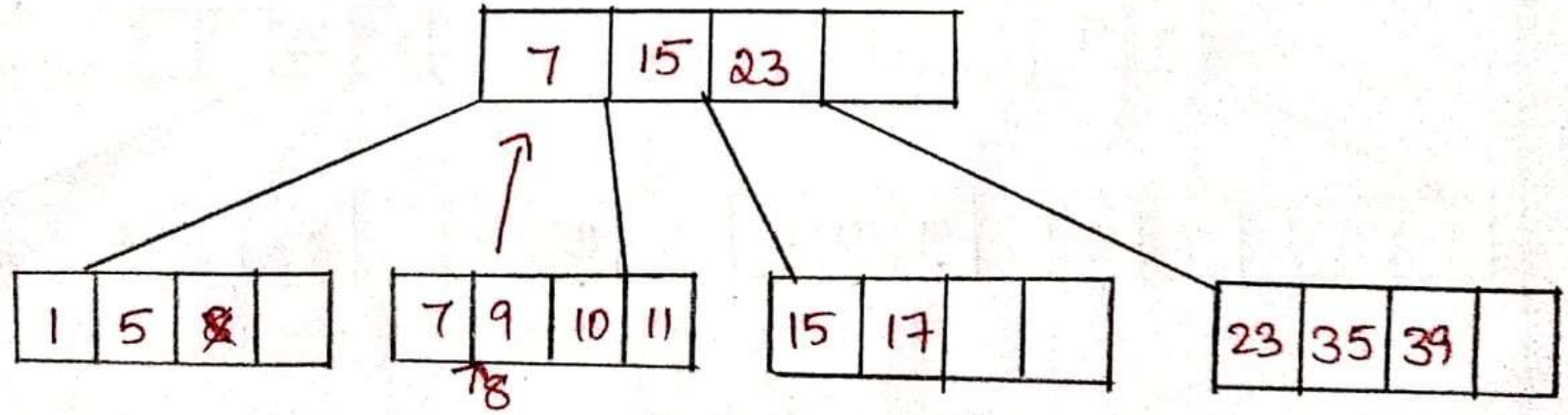


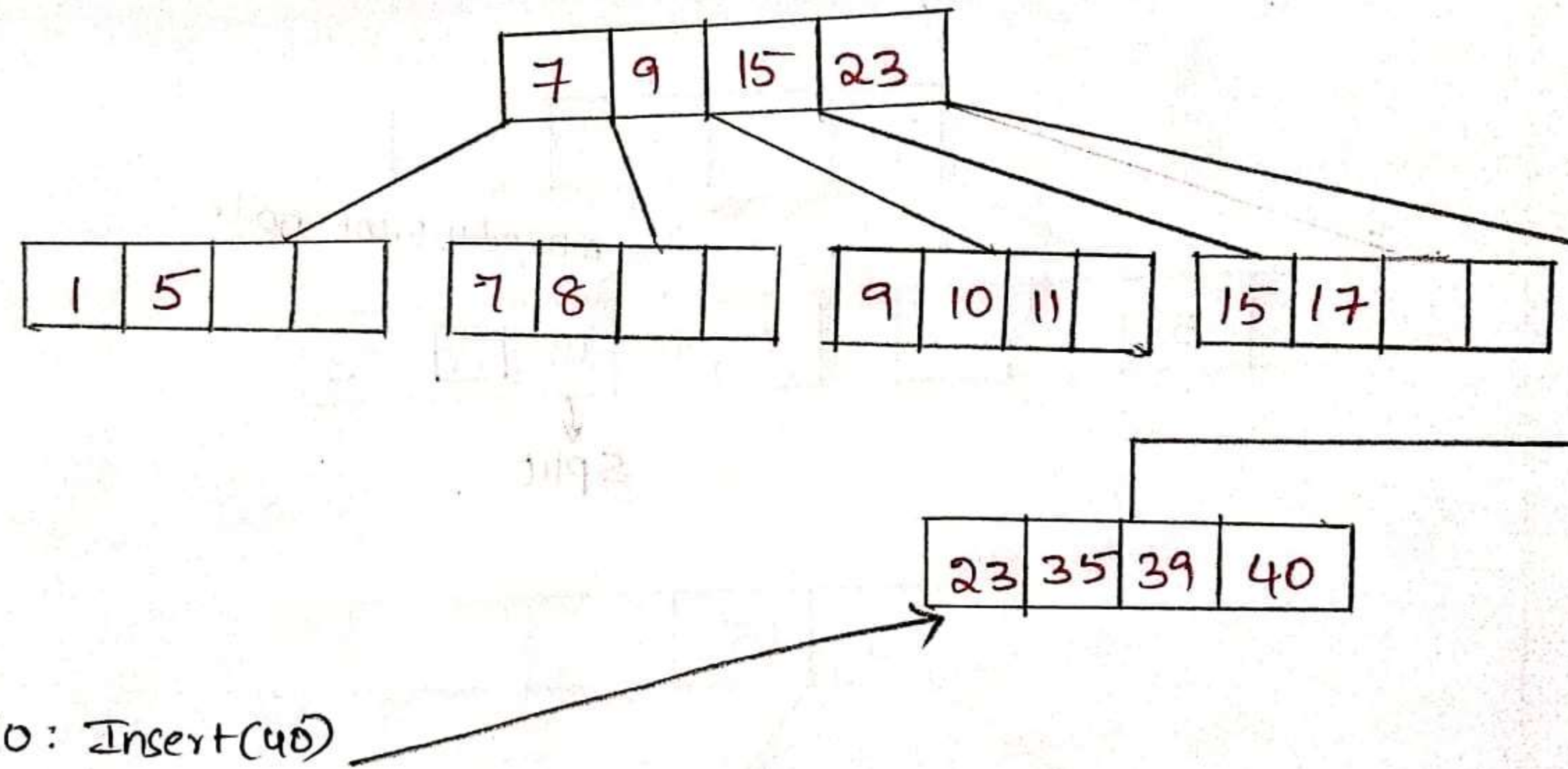


Step8 : Insert (9), Insert (11), Insert (39)



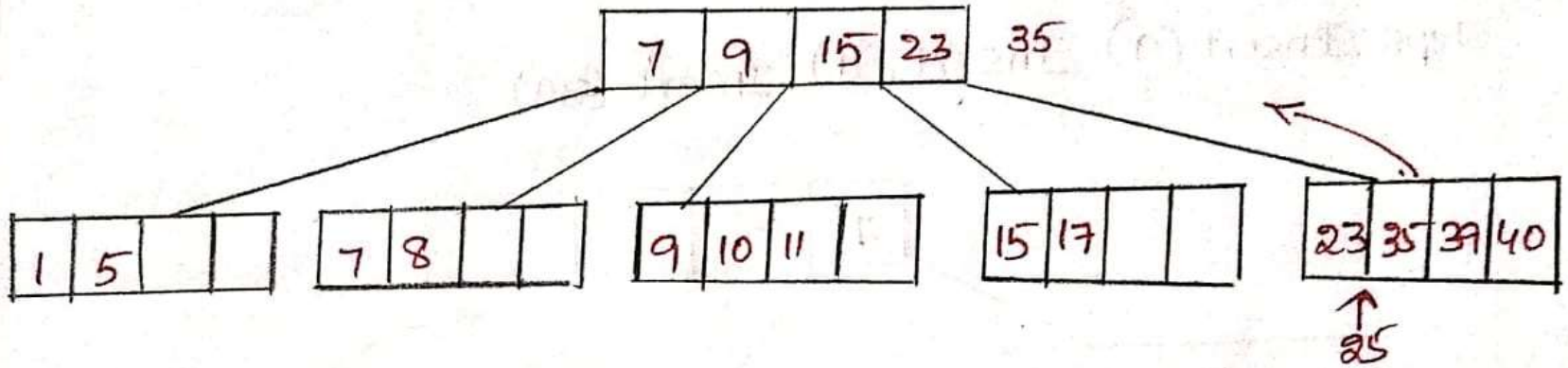
Step 9 : Insert (35), Insert (8), ~~Insert (40)~~

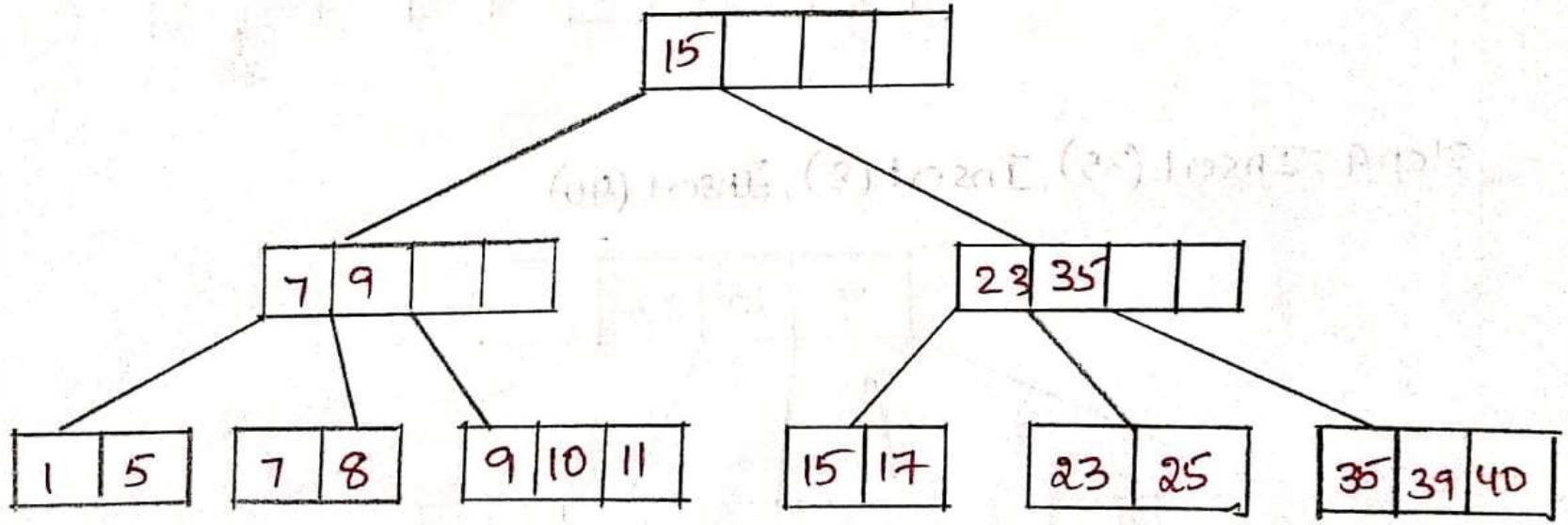




Step 10 : Insert(40)

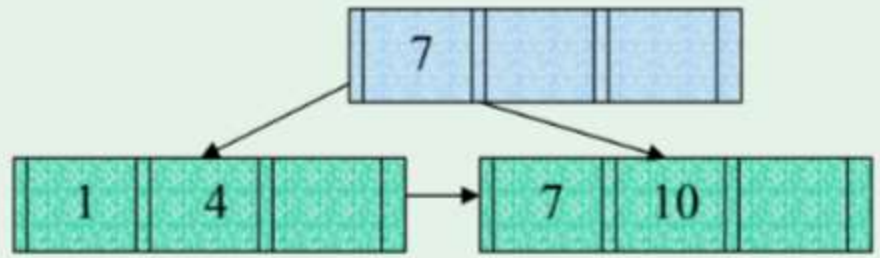
Step 11 : 25 Insert

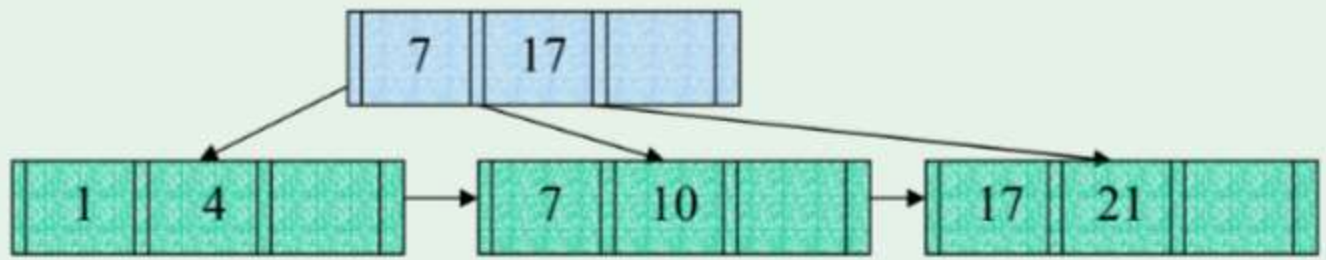




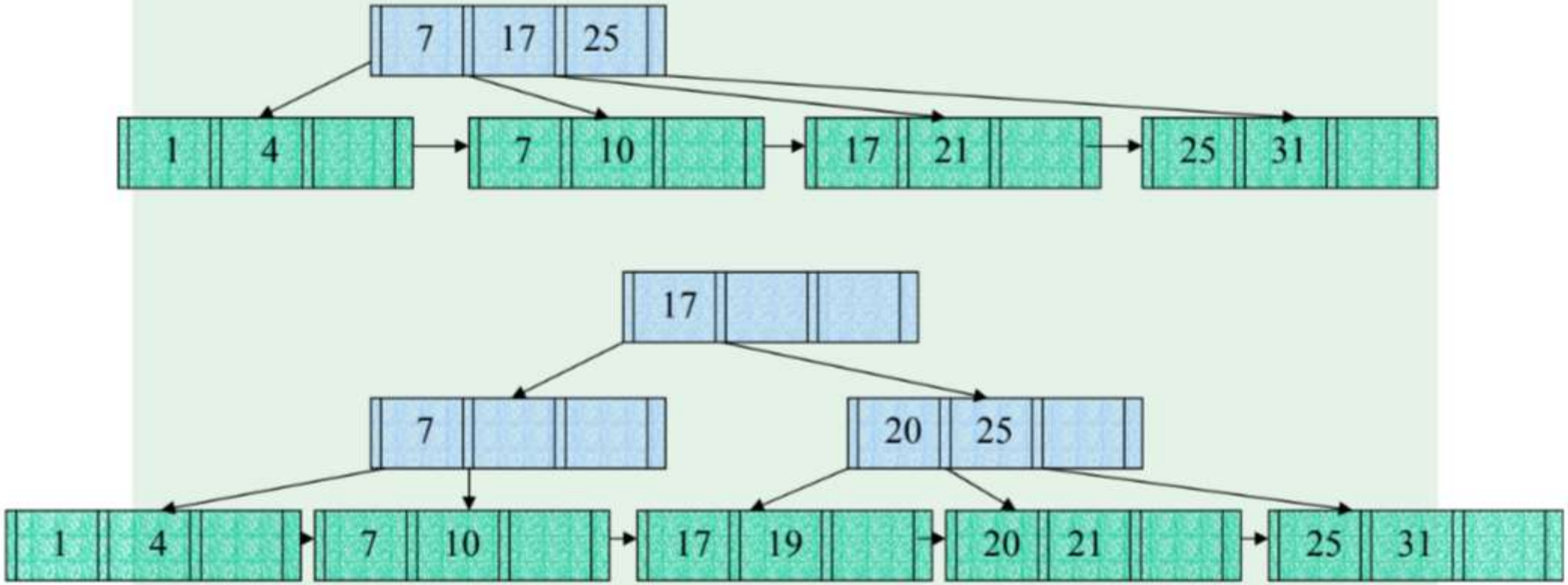
Final B+ Jree

- Example 1: Construct a B^+ tree for (1, 4, 7, 10, 17, 21, 31, 25, 19, 20, 28, 42) with n=4.

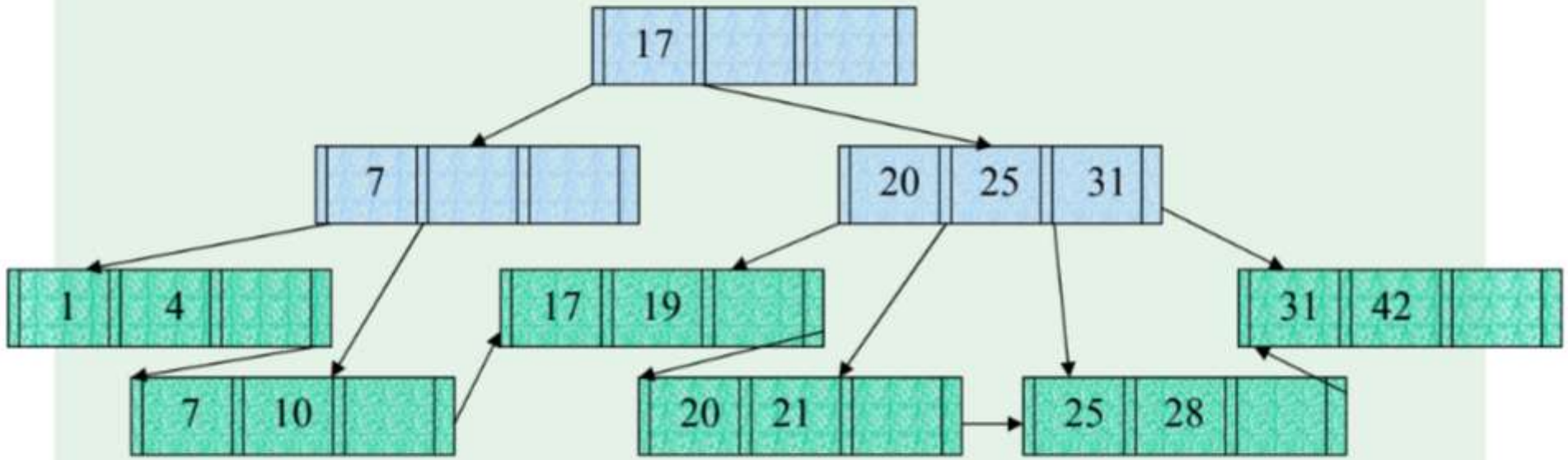




- 1, 4, 7, 10, 17, 21, 31, 25, 19, 20, 28, 42



- 1, 4, 7, 10, 17, 21, 31, 25, 19, 20, 28, 42



Deletion

Before going through the steps below, one must know these facts about a B+ tree of degree **m**.

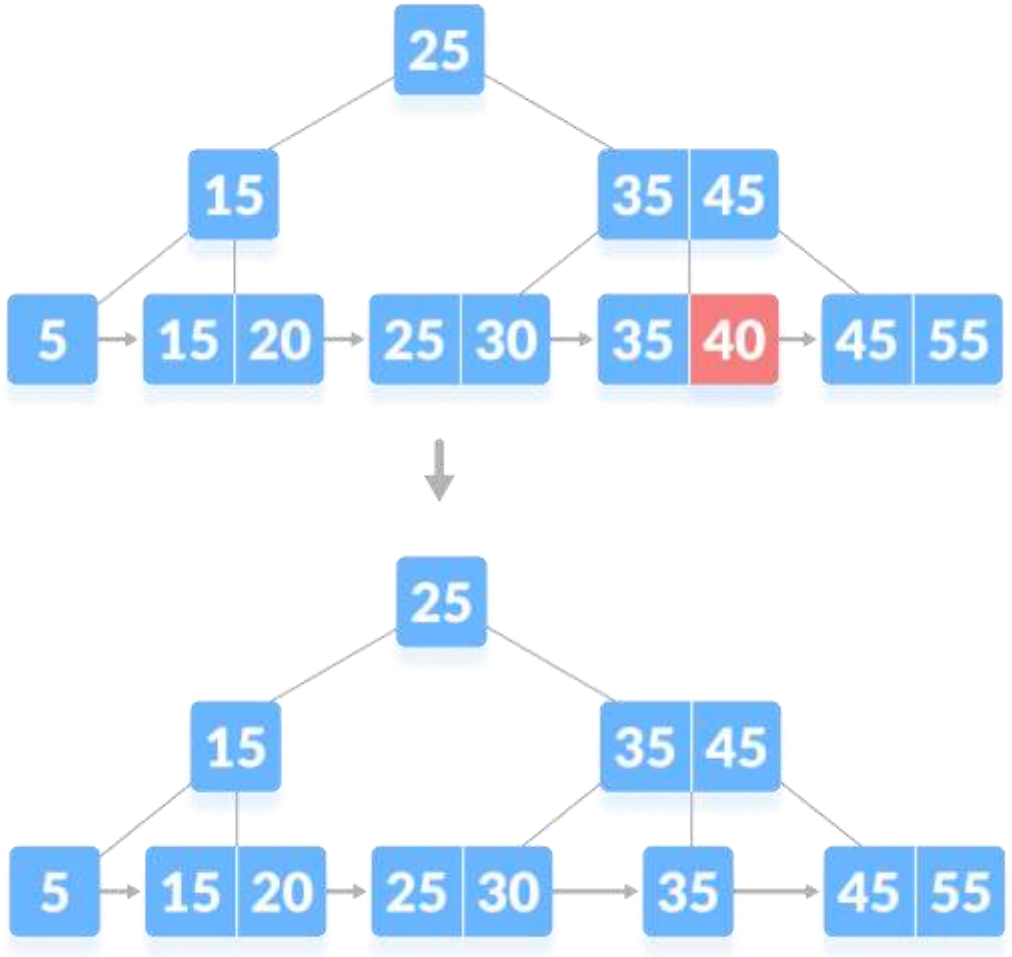
1. A node can have a maximum of m children. (i.e. 3)
2. A node can contain a maximum of `m - 1` keys. (i.e. 2)
3. A node should have a minimum of `⌈m/2⌉` children. (i.e. 2)
4. A node (except root node) should contain a minimum of `⌈m/2⌉ - 1` keys. (i.e. 1)

Case I

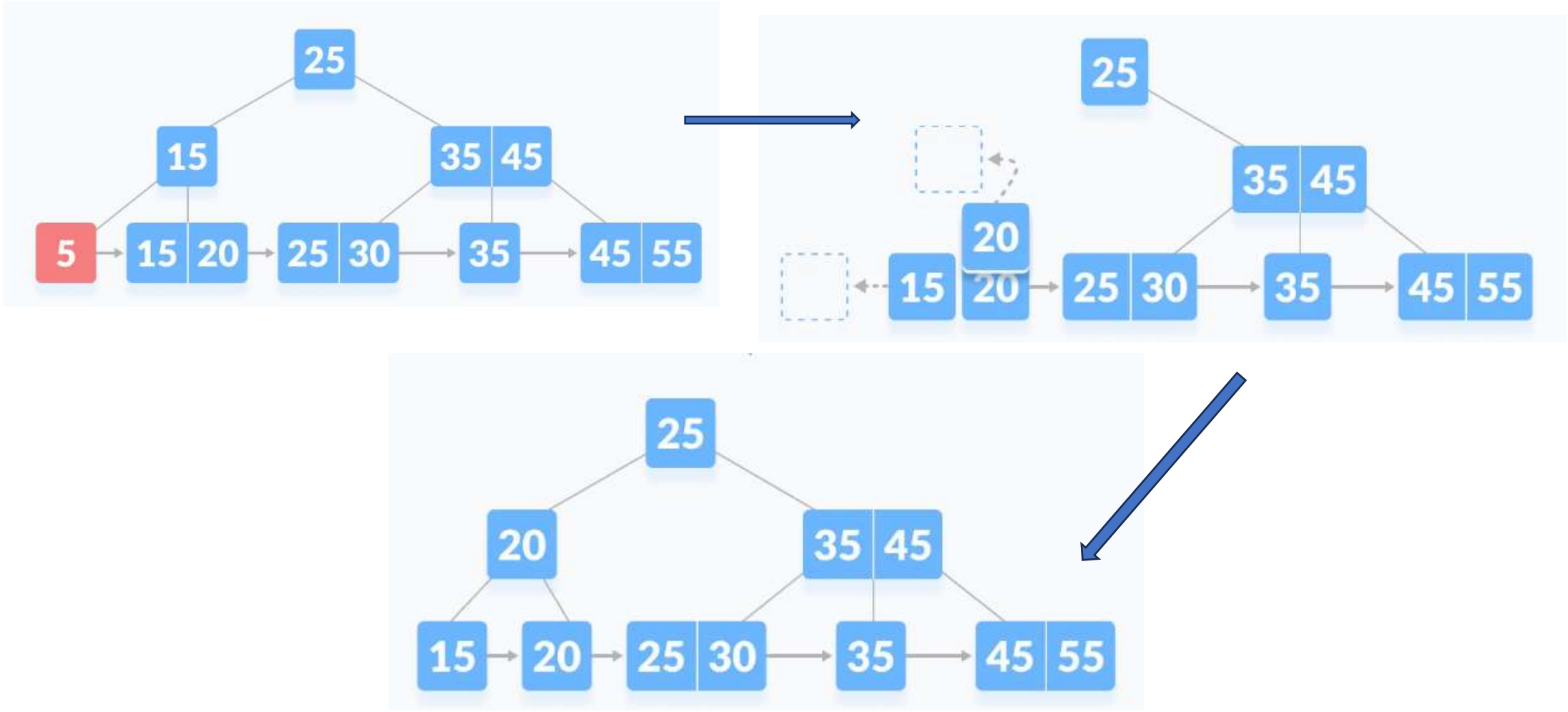
The key to be deleted is present only at the leaf node not in the indexes (or internal nodes).

There are two cases for it:

1. There is more than the minimum number of keys in the node. Simply delete the key.



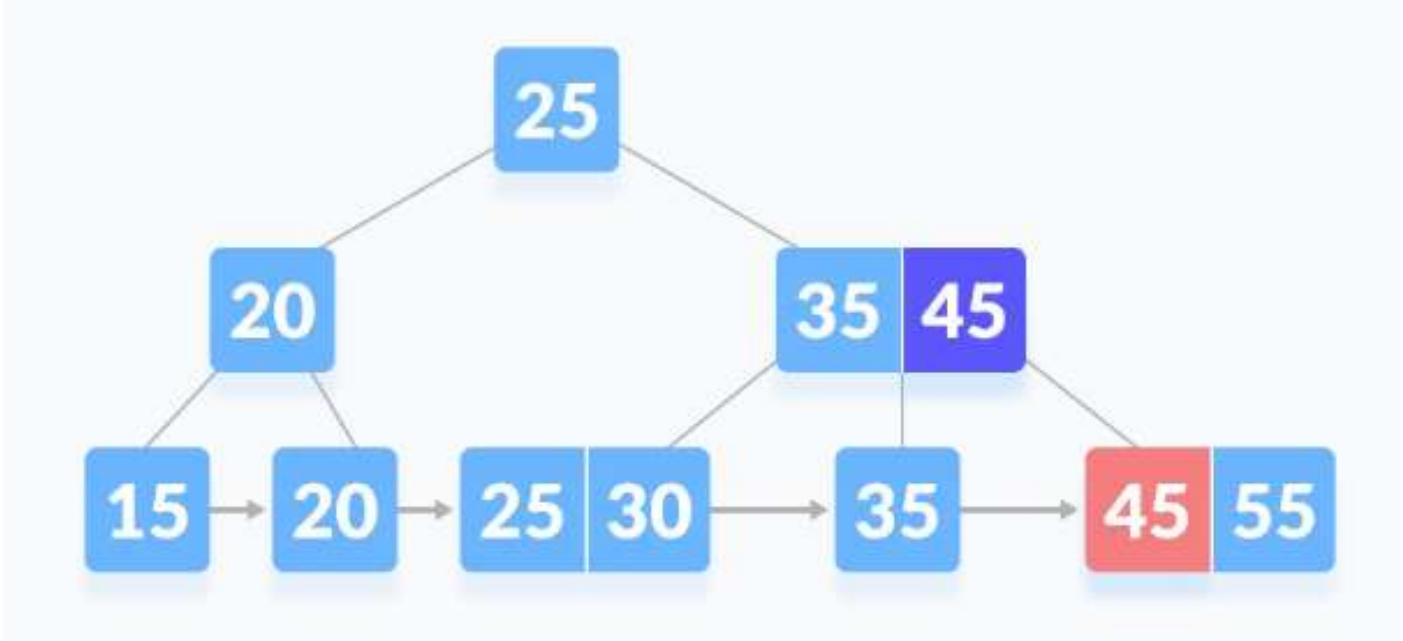
2. There is an exact minimum number of keys in the node. Delete the key and borrow a key from the immediate sibling. Add the median key of the sibling node to the parent.

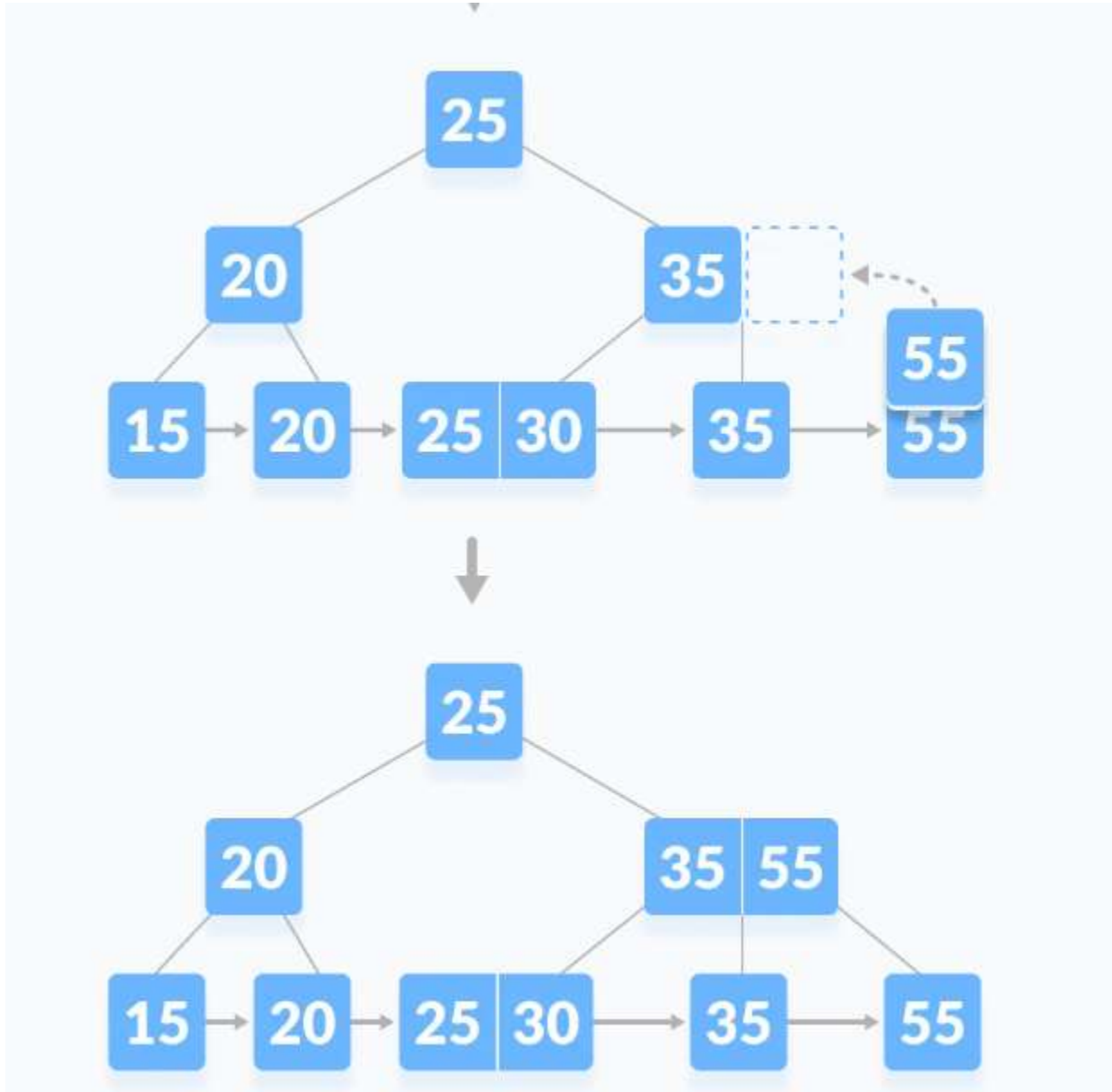


Case II

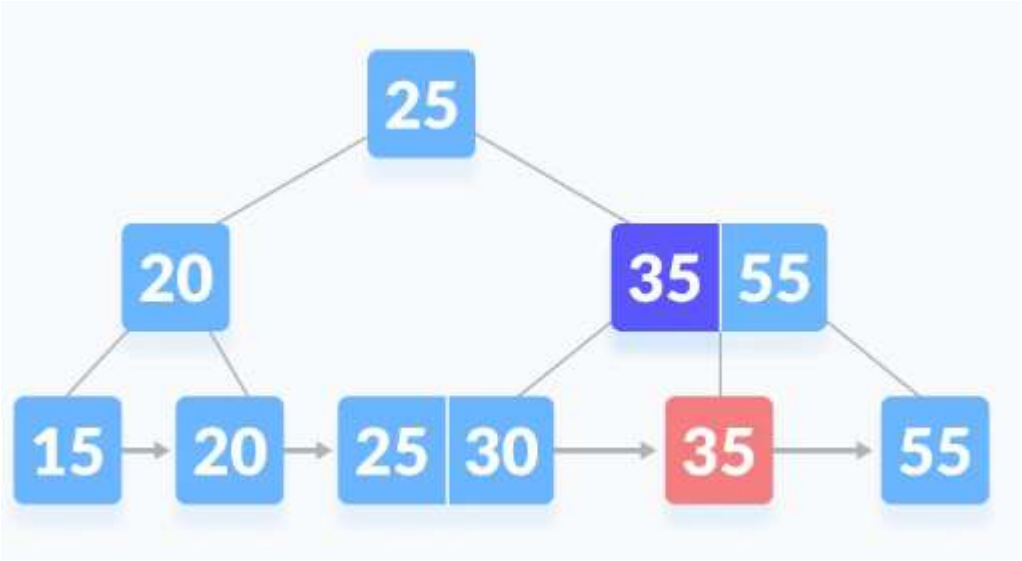
The key to be deleted is present in the internal nodes as well. Then we have to remove them from the internal nodes as well. There are the following cases for this situation.

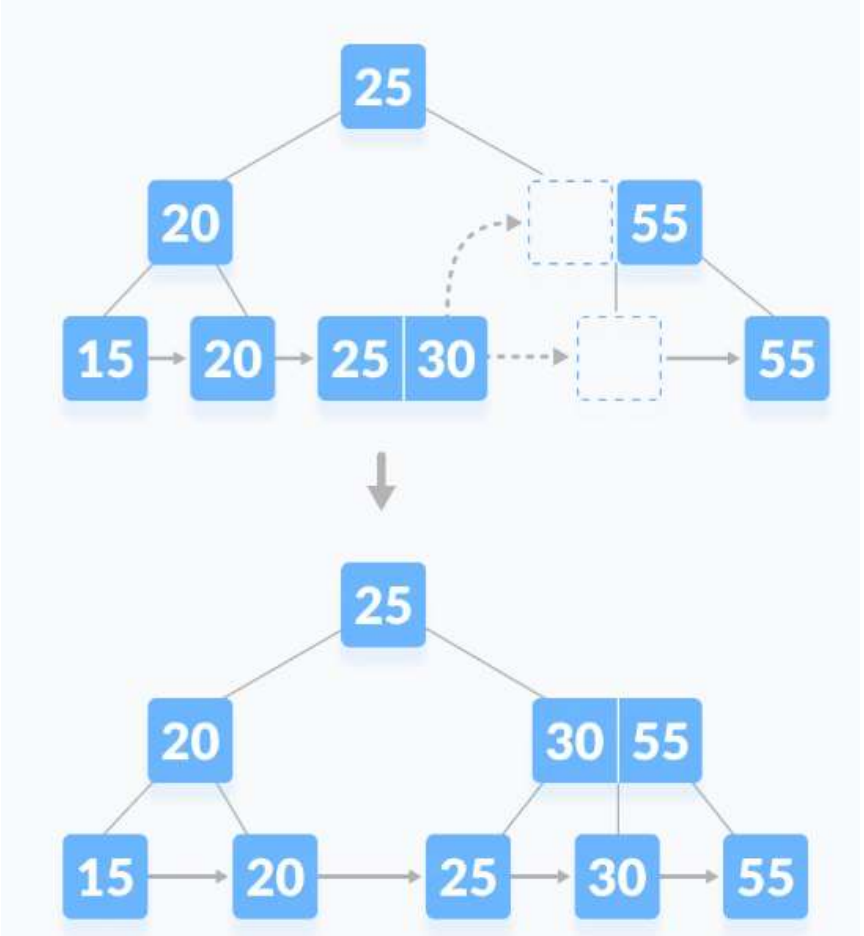
1.If there is more than the minimum number of keys in the node, simply delete the key from the leaf node and delete the key from the internal node as well. Fill the empty space in the internal node with the inorder successor.



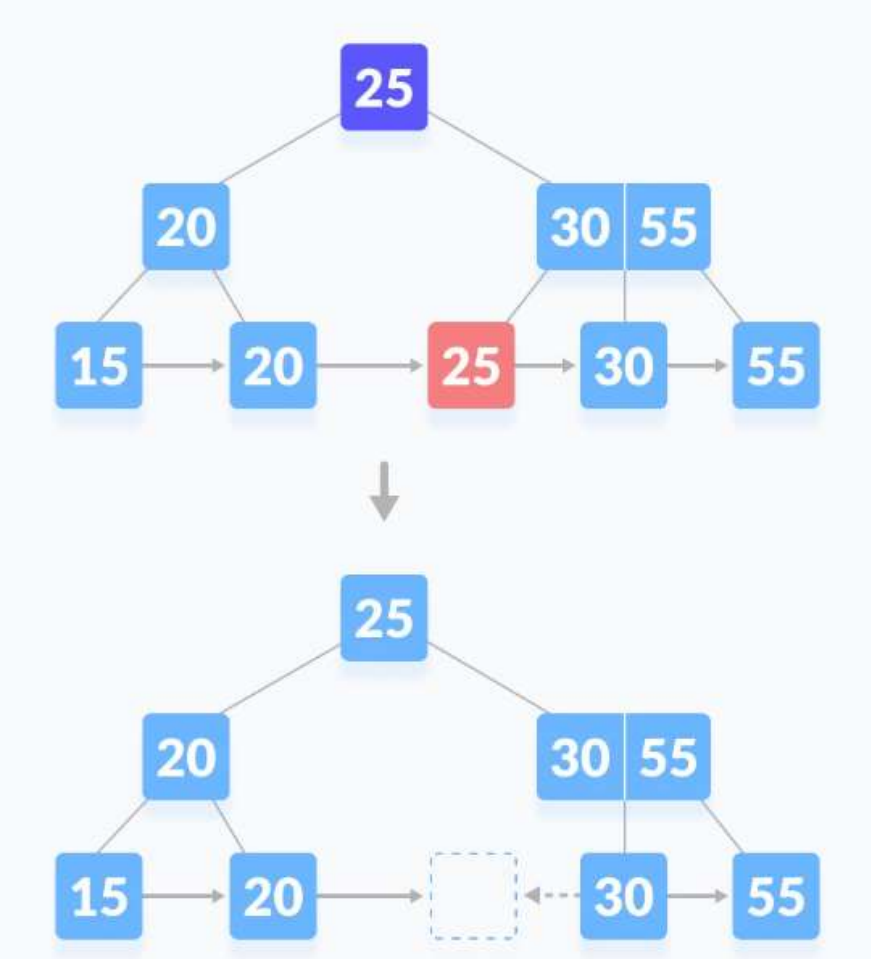

25
20
35
55
15
20
25
30
35
55
25
20
35
55
15
20
25
30
35
55

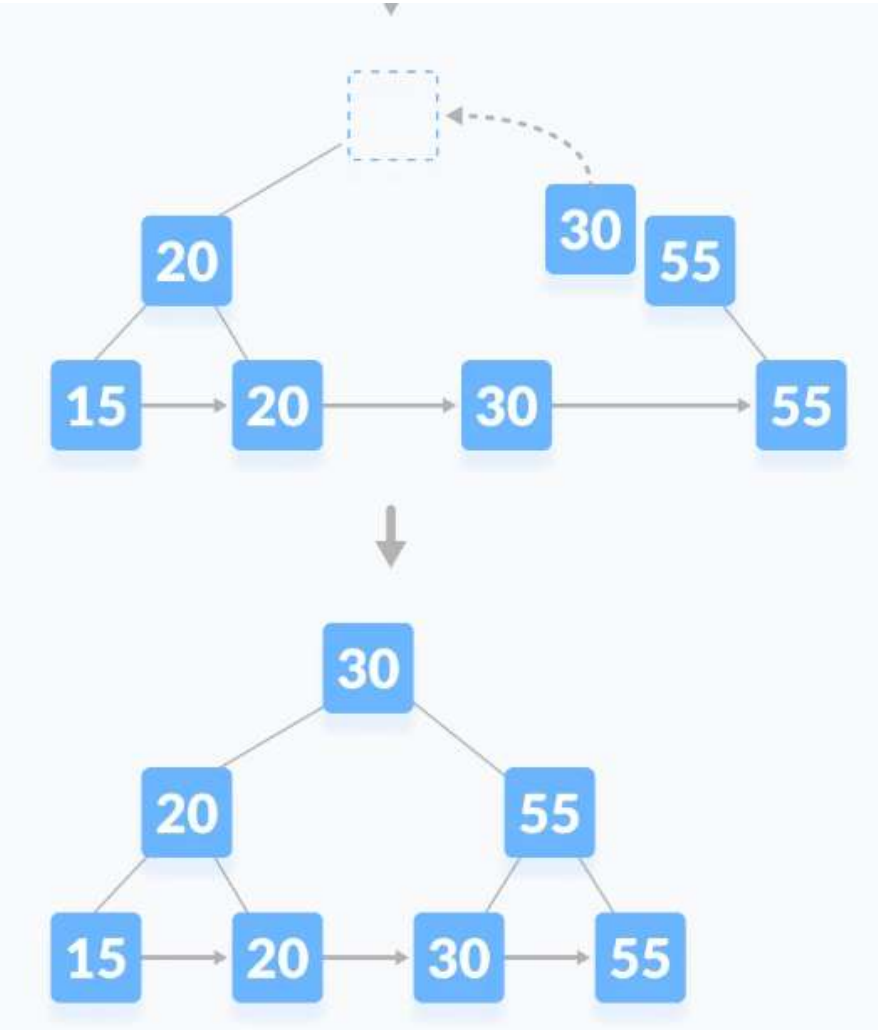
2. If there is an exact minimum number of keys in the node, then delete the key and borrow a key from its immediate sibling (through the parent). Fill the empty space created in the index (internal node) with the borrowed key.





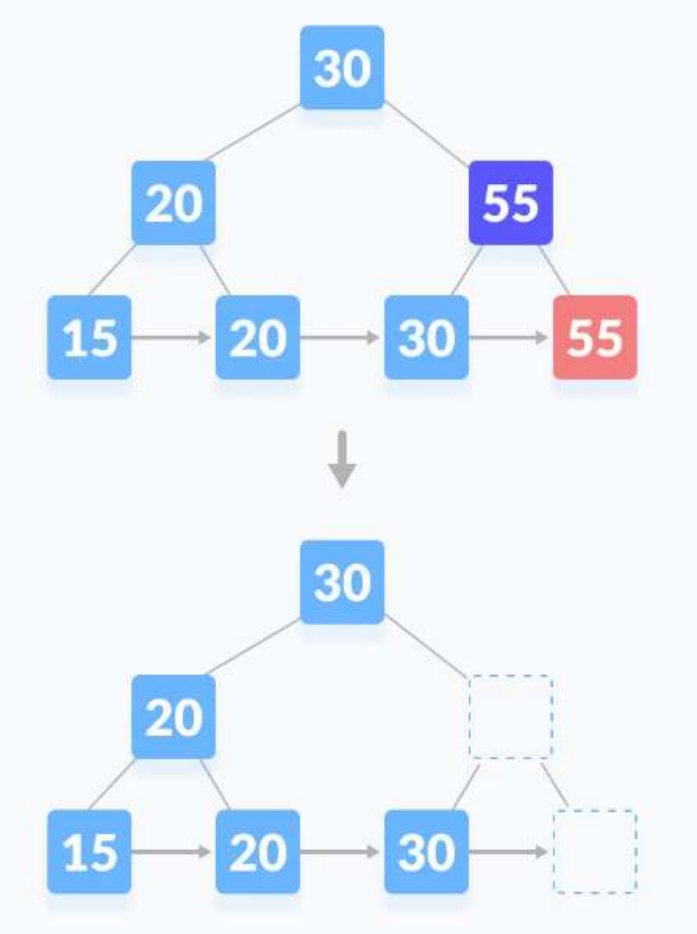
3. This case is similar to Case II(1) but here, empty space is generated above the immediate parent node. After deleting the key, merge the empty space with its sibling. Fill the empty space in the grandparent node with the inorder successor.

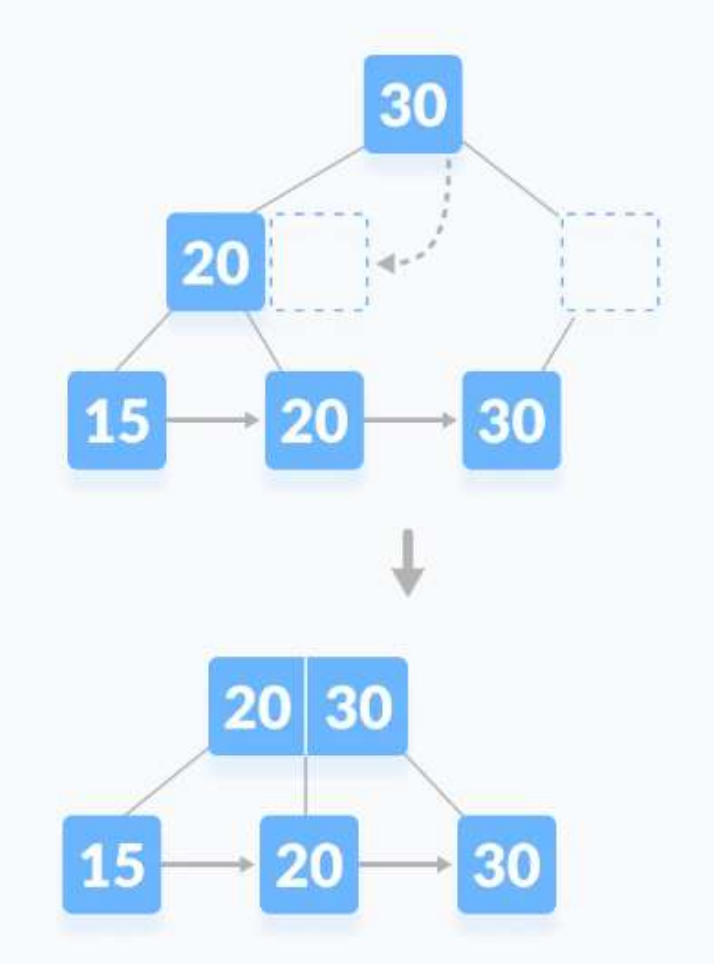




Case III

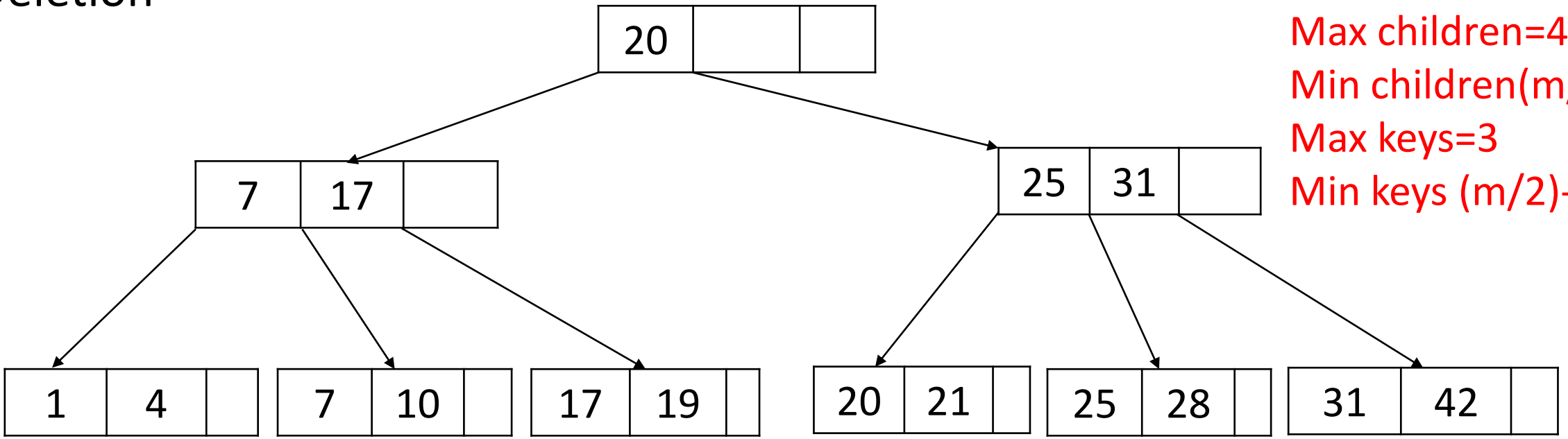
In this case, the height of the tree gets shrinked. It is a little complicated. Deleting 55 from the tree below leads to this condition. It can be understood in the illustrations below.





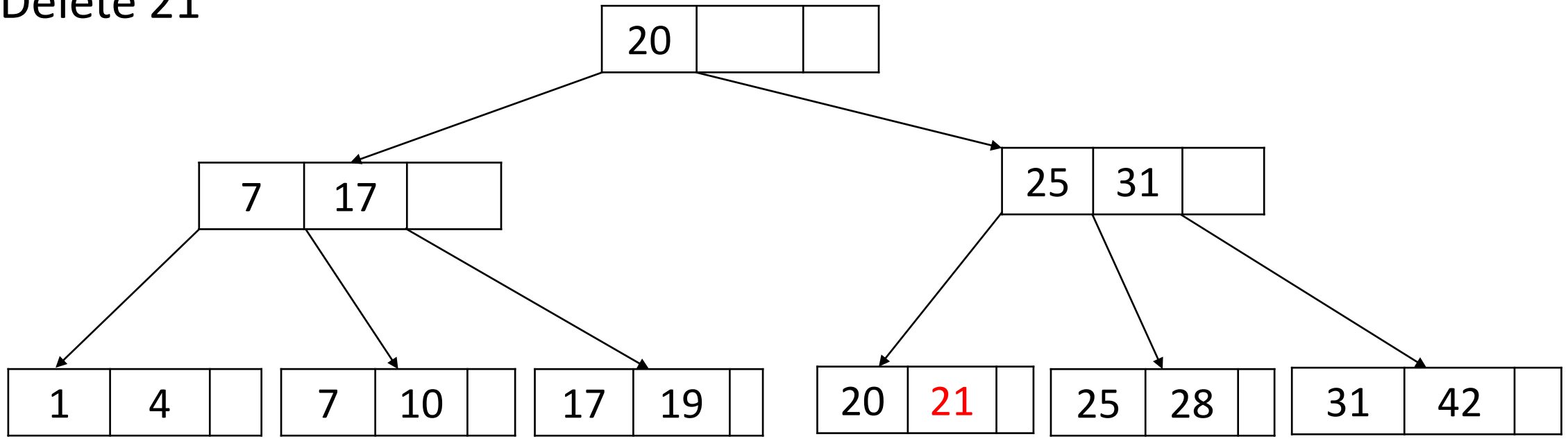
Deletion

Order (m) =4
Max children=4
Min children(m/2)=2
Max keys=3
Min keys (m/2)-1 = 1



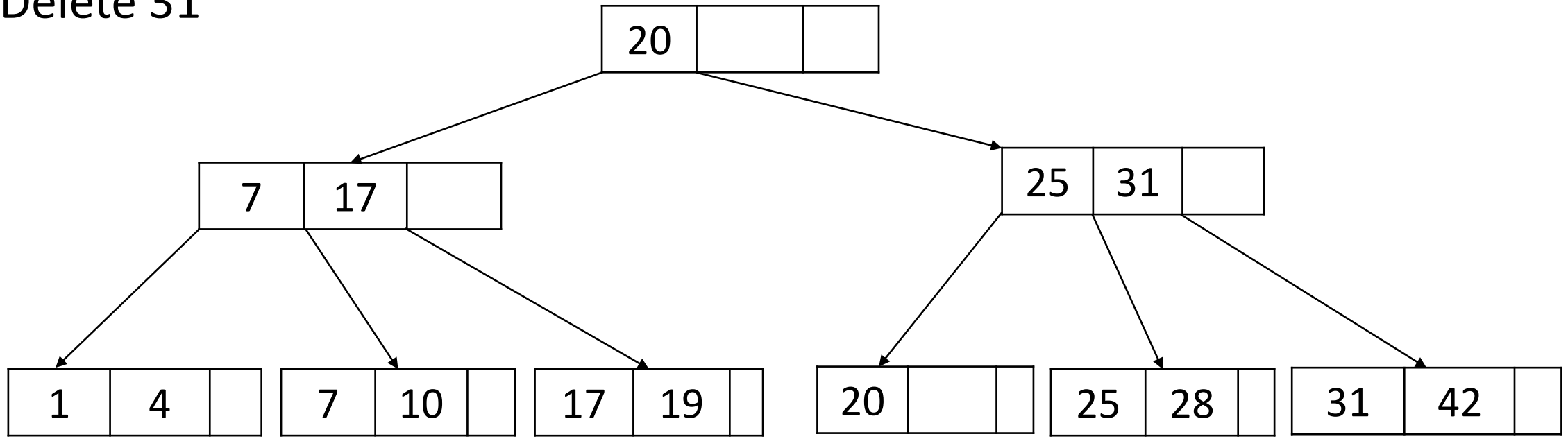
Delete the keys : 21, 31, 20,10,7,25

Delete 21

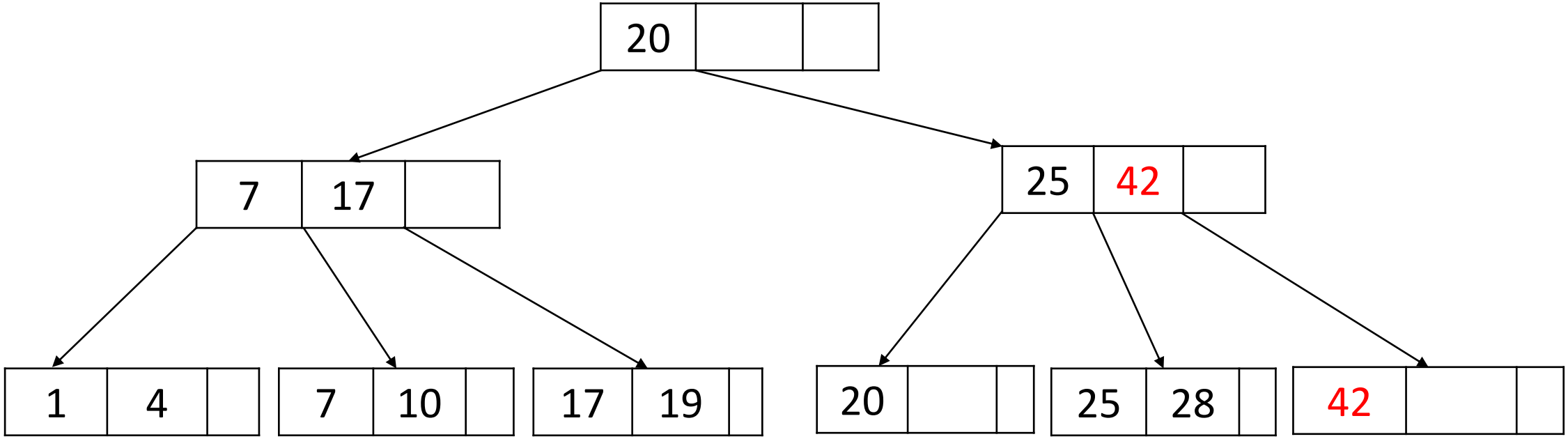


Here we have more than minimum
key size so directly we can remove 21

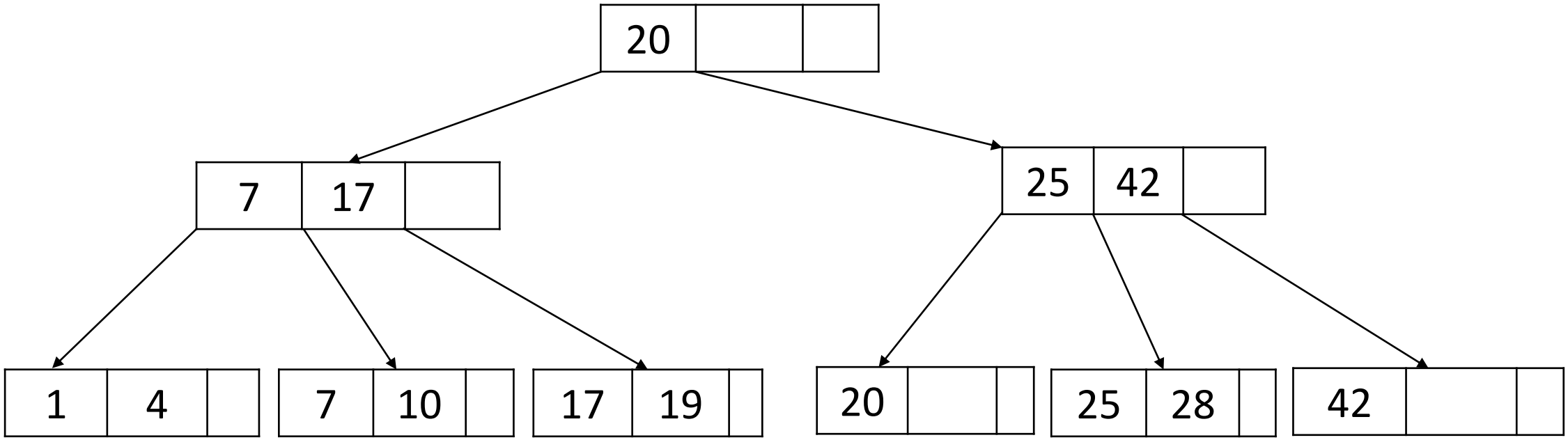
Delete 31



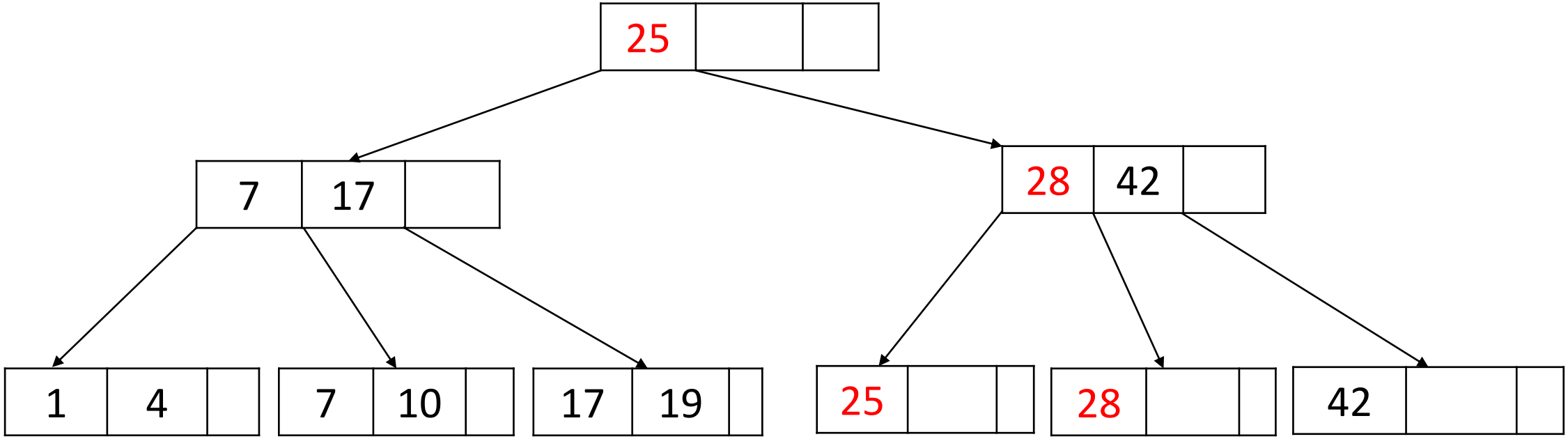
Here 31 is appeared in index level also so we should remove from leaf node as well as internal node. After deleting internal node 31 the minimum number from the right subtree will be added to the internal node (42)


20
7
17
25
42
1
4
7
10
17
19
20
25
28
42

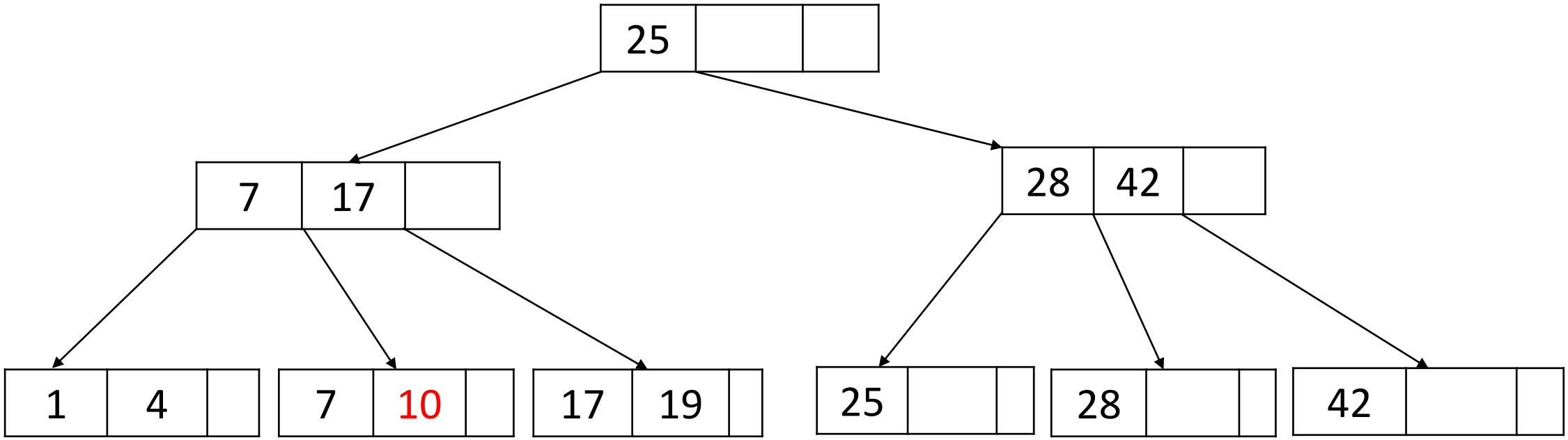
Delete 20



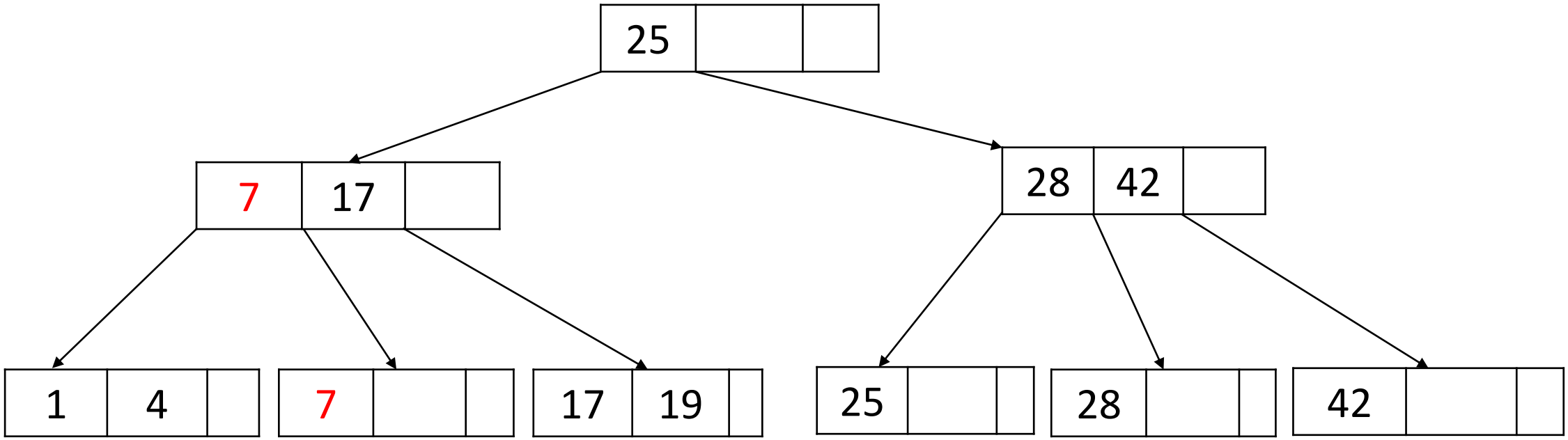
When we delete 20 the bucket is underflow ,it violates B+ tree property so we can borrow minimum key from right subtree(25)

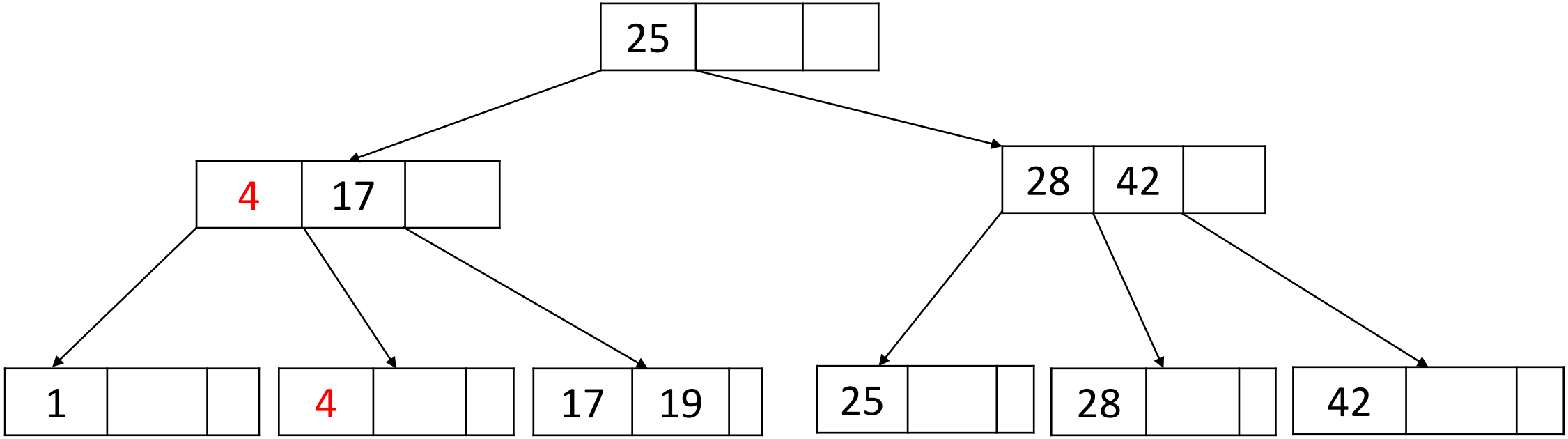

25
7
17
28
42
1
4
7
10
17
19
25
28
42

Delete 10

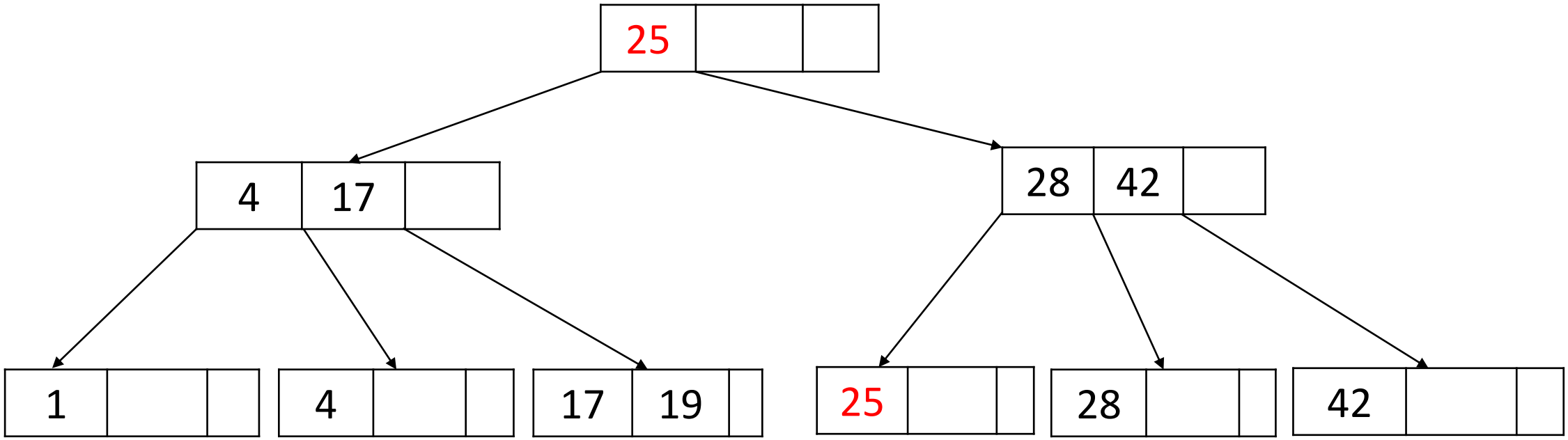


Delete 7

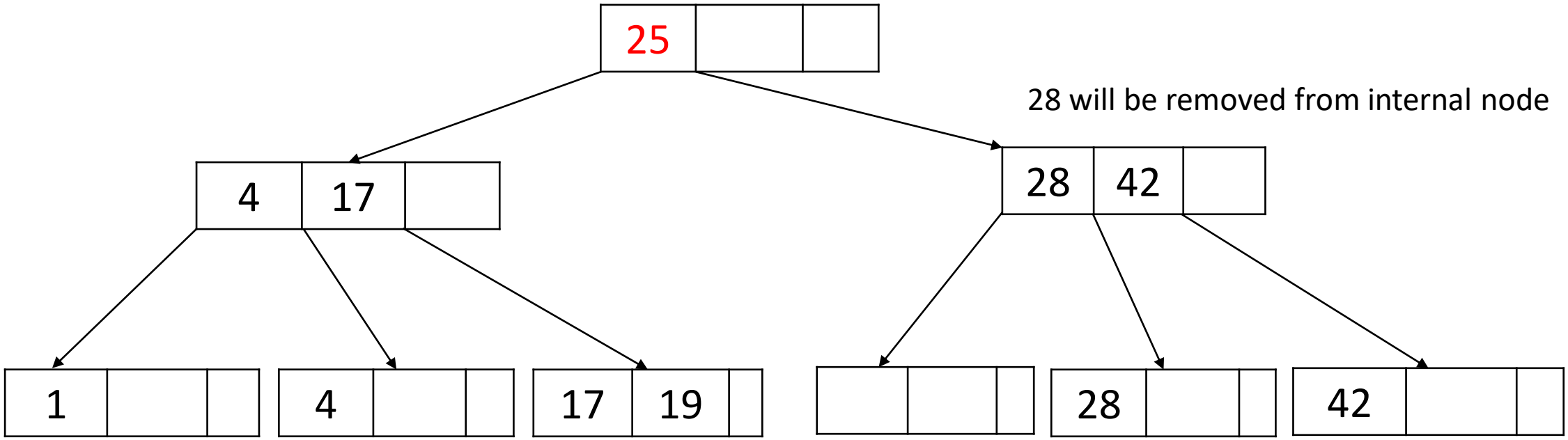


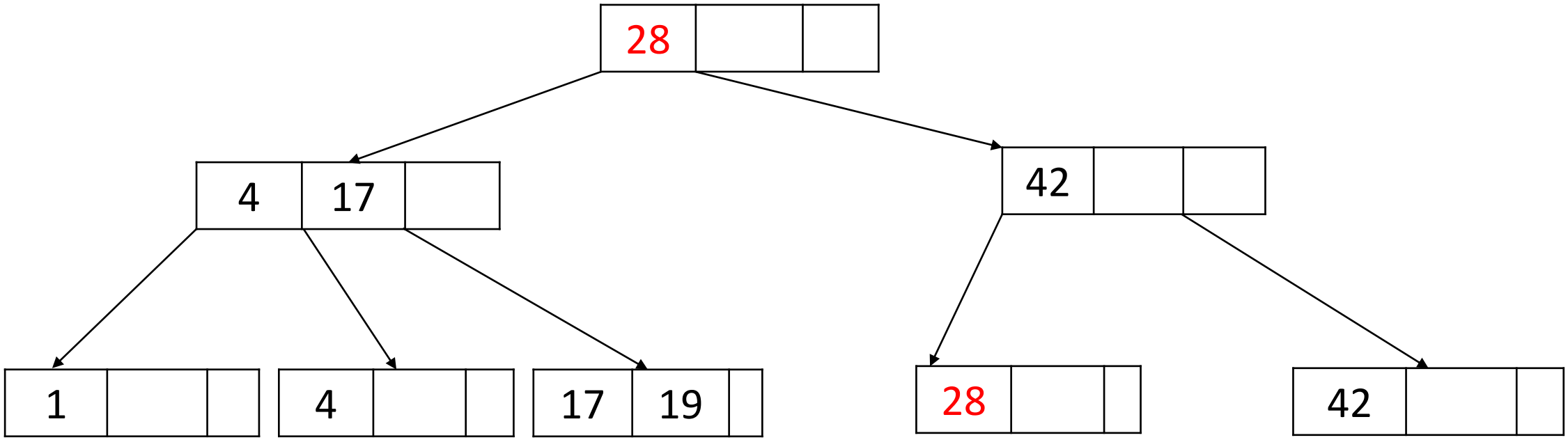

25
4
17
28
42
1
4
17
19
25
28
42

Delete 25



If 25 is delete bucket will be underflow, alternately we can borrow from right subtree but there only one key is there. Now we will merge the subtrees


25
28 will be removed from internal node
4
17
28
42
1
4
17
19
28
42



Final B+ tree

Thank You