Unit 3.
Storage Strategies

Why indexing is required
There may be two types of data (Ordererved, Unorderved) (Sorted, Unsorted).
To reduce the I/O cost indexing is required

Consider a HDD in which block size = 1000 bytes
Each record size = 250 bytes
If total number of records are 10,000 and the data entered in HDD without any order. What is average time complexity to search a record from hard disk.

no of record in a block = $\frac{1000}{250} = 4$

no of blocks required = $\frac{10,000}{4} = 2500$

No of records in each block = One block size / Each record size = 4

Total no of blocks required = Total no of records / records in one block = 2500

Best Case = 1
Avg = 1250
Worst = 2500

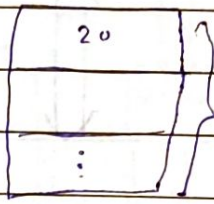
for Sorted

$\frac{1}{2}$ $\log_2 n$ $\frac{\log n}{2}$

What is avg time complexity to search a record from index table.
Index table entry = 20 bytes (10 for key + 10 for pointer).

Index → Dense
Index → Sparse

Dense is used when data is unordered it consist records for all records in HDD



$\frac{1000}{20} = 50$ blocks

Size of index is same as a block
Sparse only stores initial value of each block. Thus reducing size.

Each block in index table contains total = total size of index / index entries. = 50 records

In Sparse data should be ordered

$\therefore \frac{2500}{50} = 50$

$\log_2 50 = 6$

No of Searches in index = 6
No of Searches to go to record = 6+1 = 7

camlin

For dense

$S = \frac{10,000}{50} = 200$

$\log_2 200 = 8$

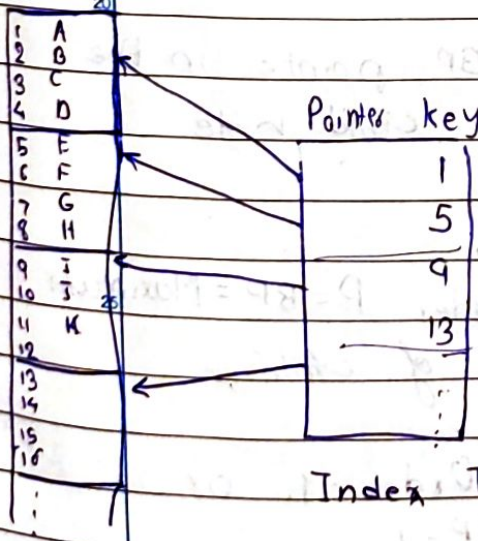
Total searches = 8 + 1 = 9.

We use indexing to reduce input output cost.

* Types of Indexes:

Sorted	Primary	Clustered.
Unsorted	Secondary	Secondary
	Key	Non key.

1] Primary:



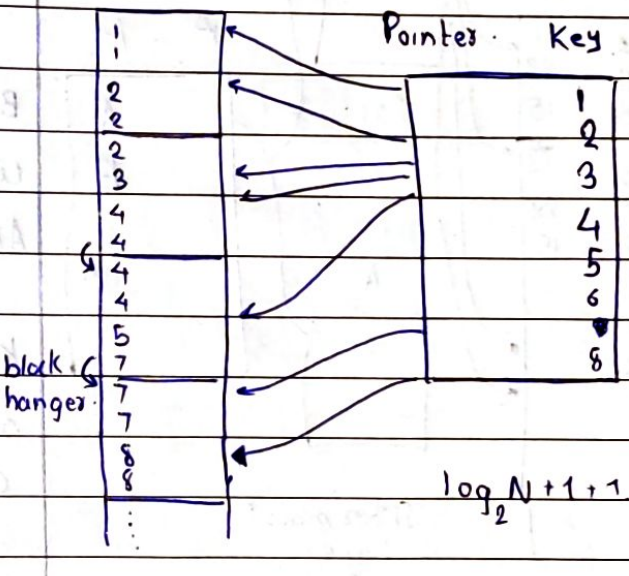
Sparse... $\log_2 N + 1$

Index Search · Data Search.

Primary index has Sparse key values.

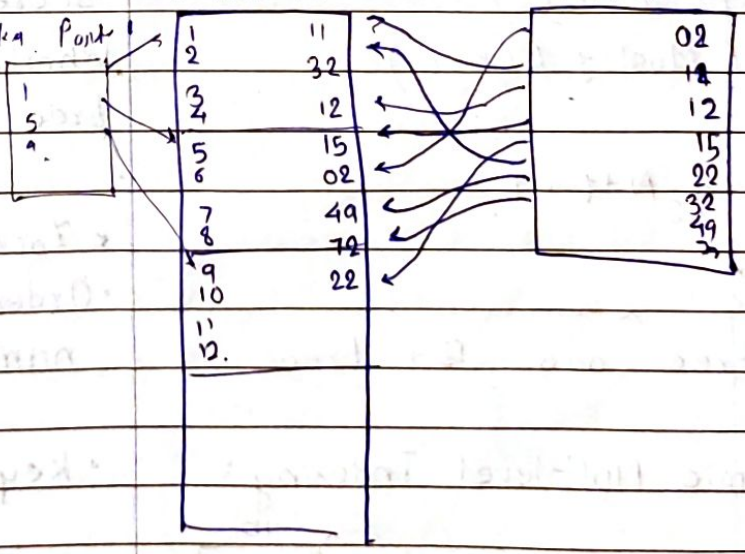
- No of entries in index table is equals to number of blocks in hard disk.
- Time complexity is $\log_2 N + 1$. where N = No of blocks in index tables.

2] Clustered:



$\log_2 N + 1 + 1$

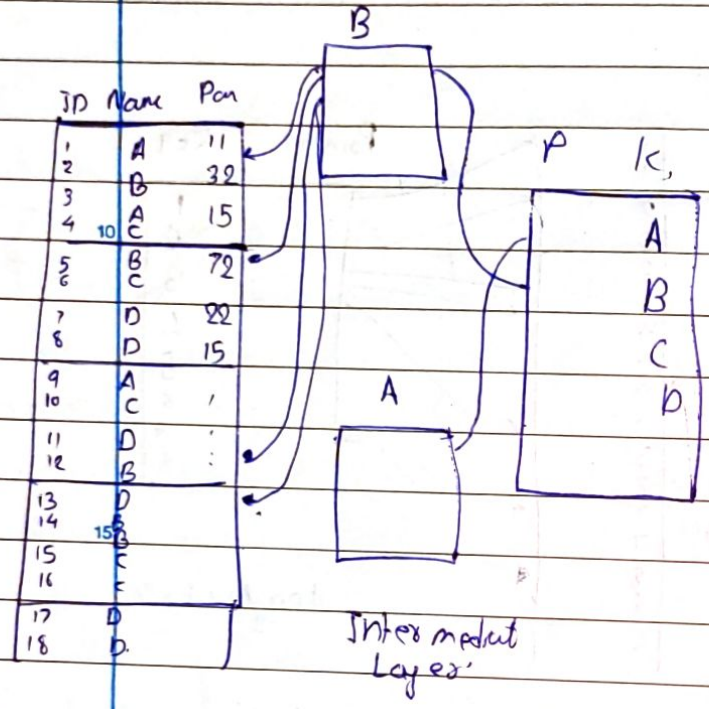
3] Secondary:



No of records in index is equals to no of records in hard disk.

$\log_2 N + 1$

N = Total No of blocks in index table.

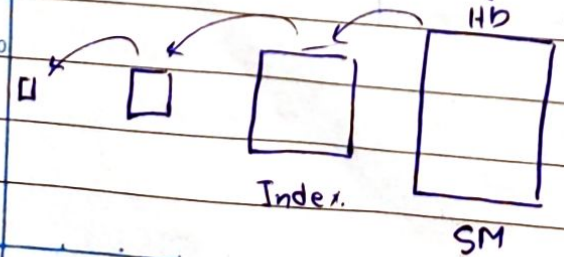


Sparse + Dense.
(Mostly Dense)

$\log_2 N + 1 + 1$.

B tree and B+ trees

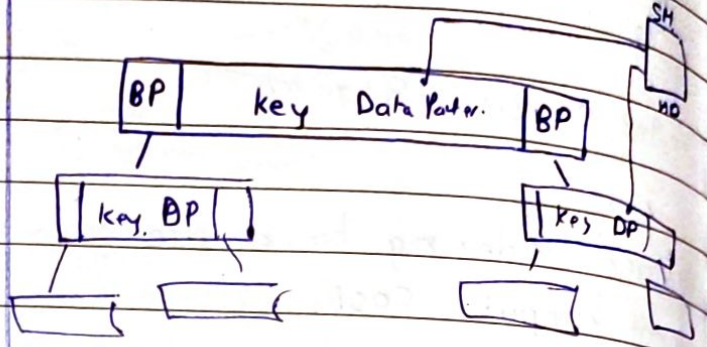
Dynamic Multilevel Indexing:



B trees are used for dynamic multilevel indexing

Key points to remember:

TP/BP → Block Pointer.



BP Called as block pointer which is number of children. Also called as Tree Pointer

Keys: key denotes the node value and DP denotes data pointer also called as record pointer.

Record Pointer points to Secondary memory where the data is Store.

Whereas, BP points in the tree for child node

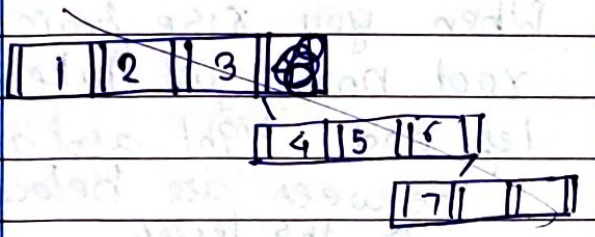
* Formula
- Order: Order P = BP = Maximum number of children.
- key = Order - 1. or = BP - 1

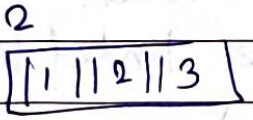
Create B tree order = 4.
Find maximum number of children, keys, minimum key & minimum number of key

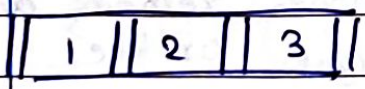
min no of children $= \lceil \frac{p}{2} \rceil = 2$.

min no of key $= \lceil \frac{p}{2} \rceil - 1 = 1$

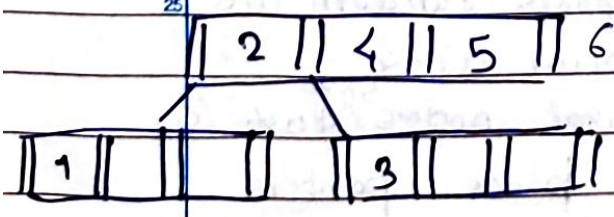
Keys are 1, 2, 3, 4, 5, 6, 7, 8, 9, 10



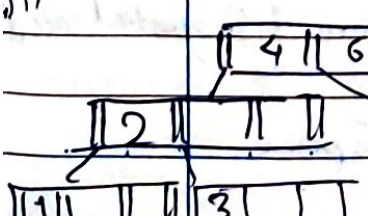


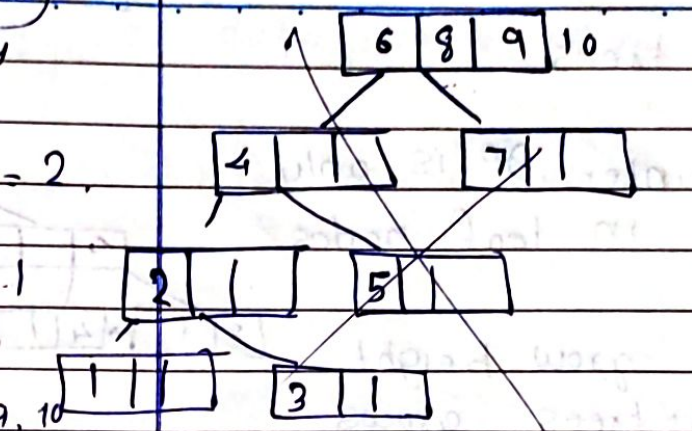


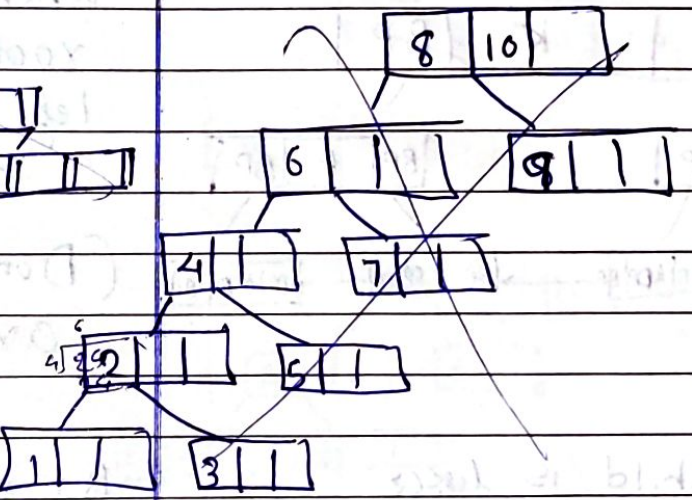
$1+2+3+4/2 = 2.5 \rightarrow 2$.



$2+4+5+6 = 4. \Rightarrow 4$







HW

Q Order = 4

Keys are 10, 20, 40, 50, 60, 70, 80, 30, 35, 5, 15

* Consider a B-tree with key size = 10 bytes block size 512 bytes
Data pointer 8 bytes
Block pointer 5 bytes
Find the order of b-tree

$n \times BP + (n-1)k + (n-1)BP \leq 512$

$n \times 5 + (n-1)10 + (n-1)8 \leq 512$

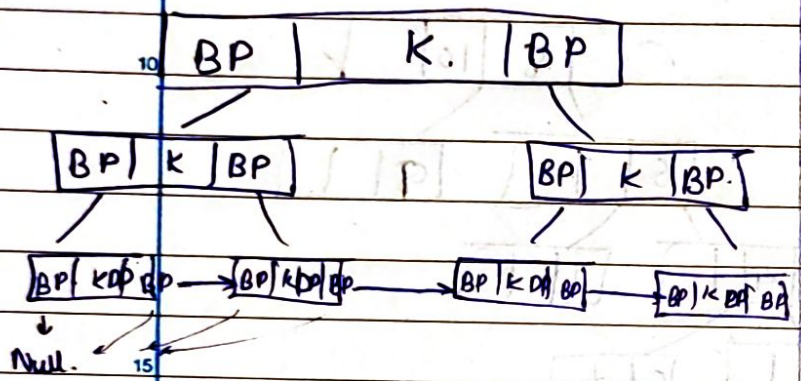
$5n + 10n - 10 + 8n - 8 \leq 512$

$23n \leq 530 \Rightarrow n \leq 23$

* B+ trees

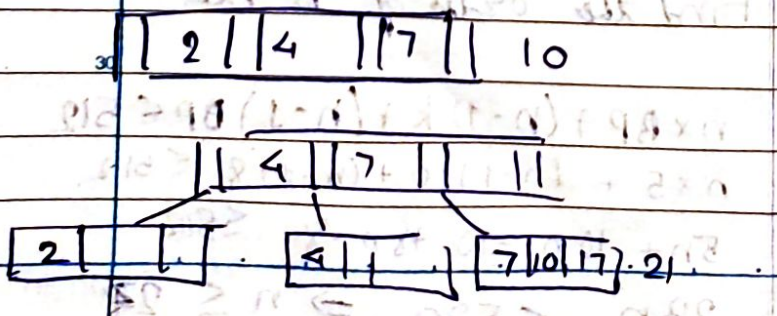
Data Pointer DP is only present in leaf nodes.

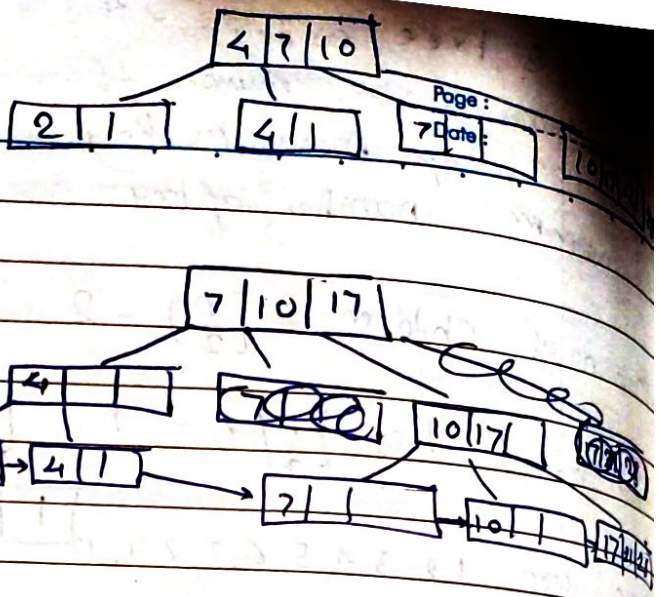
- B trees grow height wise B+ trees grows widthwise.



- LHS Child is lesser RHS is greater or equals to.
- All keys and Data pointers are present in leaf nodes
- Each nodes in leaf node level are connected by linked list.

Q keys 2, 4, 7, 10, 17, 21, 28
order = 4.





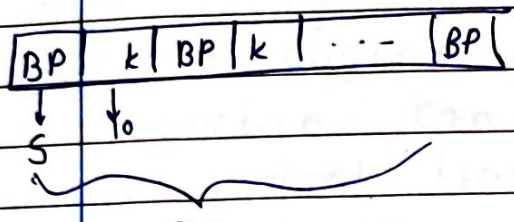
When you rise from root node just take left and right and all between are below not is the level.
(Dont forget linked list arrows)

- B+ tree is used to implement database index in b+ trees leaf nodes denotes actual data pointers.
- All leaf nodes remains at same height or level
- The leaf nodes are linked using linked list.
- It supports random and sequential access
- Internal nodes stores least key and block pointers

* Advantages
  - faster than B tree
  - easy access support random & sequential
* Disadv
  - req. more space.

Consider a b+ tree with
key size = 10 bytes
block size = 512 bytes
DP = 8 bytes. & BP = 5 bytes

Find the order of leaf and non leaf nodes.



Non leaf (internal)

$5n + (n-1)10 \leq 512$

$5n + 10n - 10 \leq 512$

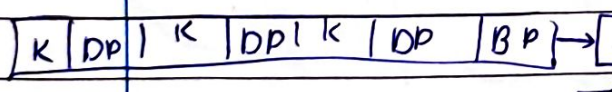
$15n \leq 522$

$n \leq 34.8$

$n \leq 34$

keys = 33

leaf node



$x(K+DP) + BP \leq 512$

$x(10+8) + 5 \leq 512$

$10x + 8x \leq 507$

$x \leq 28.16$

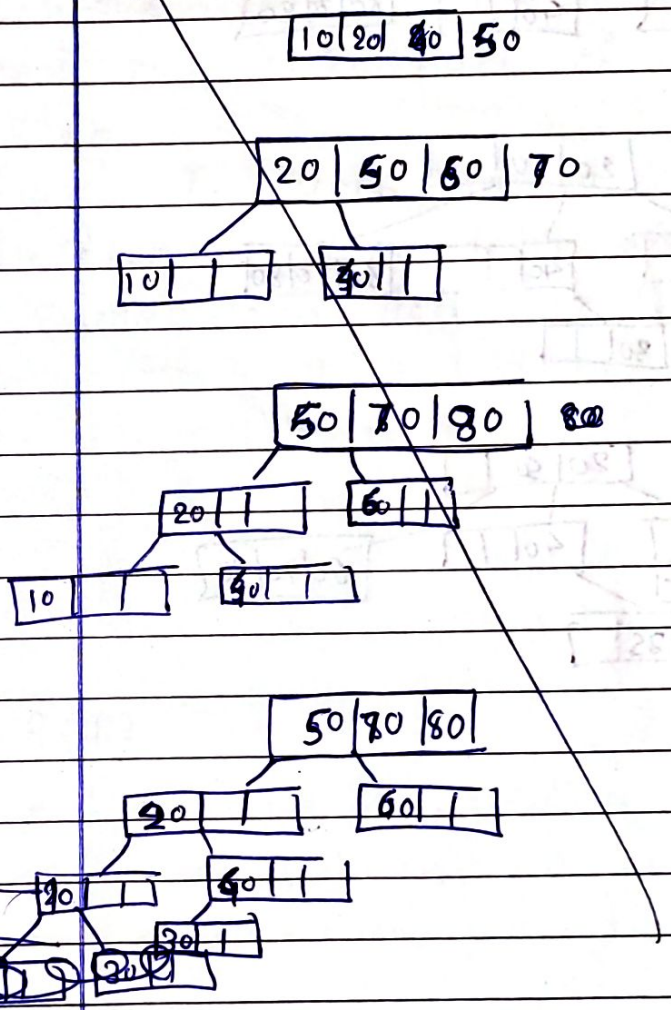
$x \leq 28$

For Non leaf order is maximum number of block pointer which is maximum number of child

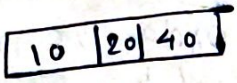
For leaf order is maximum number of key and data pointer as pair

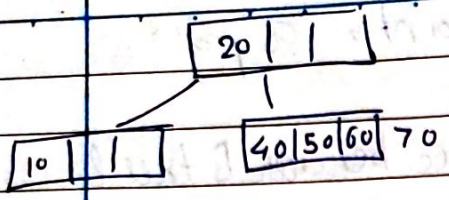
* Difference between B trees & B+ trees.

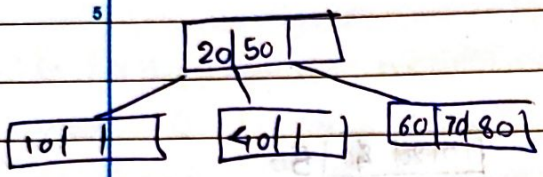
HW.

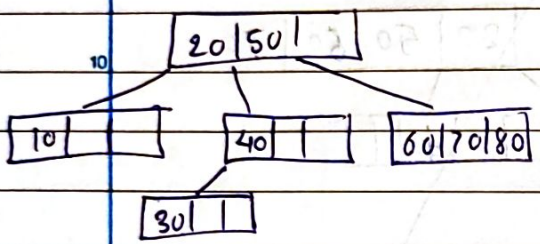


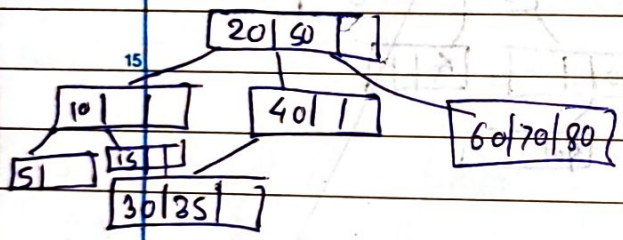
Camlin

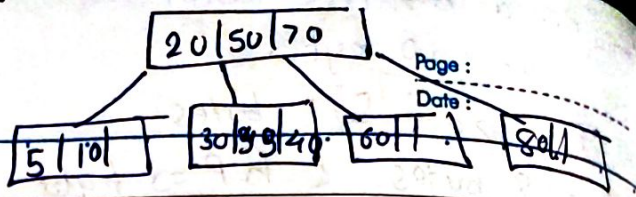












Page :
Date :

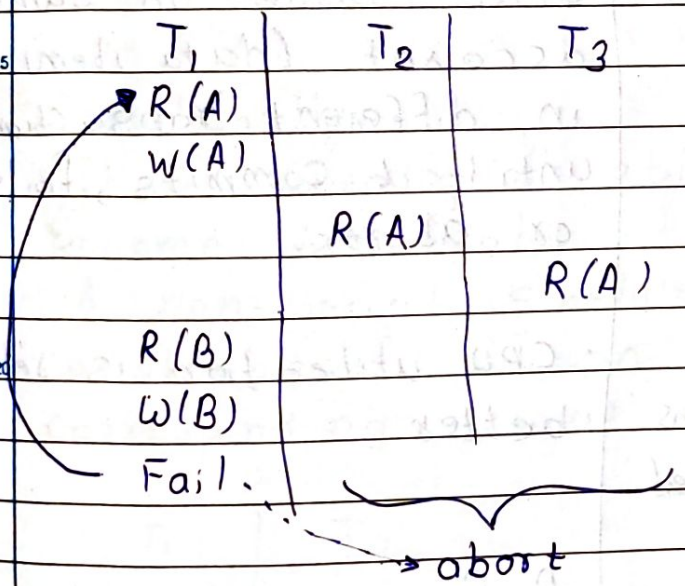
Page :
Date :

Cascading VS Cascadless Schedule.

In DBMS, schedules define the order of executions, ensuring data consistency.

1. Cascading Schedule :

- In a cascading schedule, the rollback of one tranction can lead to the rollback of other dependent transaction.
- If a transaction T_1 gets failed all other transactions dependent on T_1 (working on same data item as T_1) are aborted.



2. Cascadless Schedule :

- In a cascadeless schedule, transactions are not allowed to read uncommitted data from othe transaction. This eliminates the possibility of cascading rollback.

ex.

T_1	T_2
R(A)	
W(A)	
	R(A) ✗ Not allowed till Commit.
Commit	
	R(A) ✓ Allowed

Cascade	Cascadless
• If there is a read after write operation on same data item but in different transaction then after failure you have to abort all the transactions which comes after write.	• You are not allowed to read a data item after write on same account (data item) in different transaction until it commits, fails or aborts.
• CPU utilization is not proper as all transactions are aborted and rollbacked.	• CPU utilization is relatively better
• Performance is low	• Performance is better

* Strict Schedule:
- It states that after write operation the transaction must commit or fail.
- It guarantees recoverability and consistency by imposing strict rules on timing of read and write operation.

- A transaction can read or write a data item only after the transaction that last wrote to the data item has committed or aborted.
- No transaction can read or write from another transaction that is still in progress.

T_1	T_2
R(A)	
	R(A)
W(A)	
Commit	

commit after write

* Serializability:

- It is checking of ability of a schedule to become serializable.
- A non-serial schedule is said to be serializable if it can be transformed to its corresponding serial schedule.

T_1	T_2
R(A)	
W(A)	
	R(A)
	W(A)

T_1	T_2
	R(A)
	W(A)
R(A)	
W(A)	

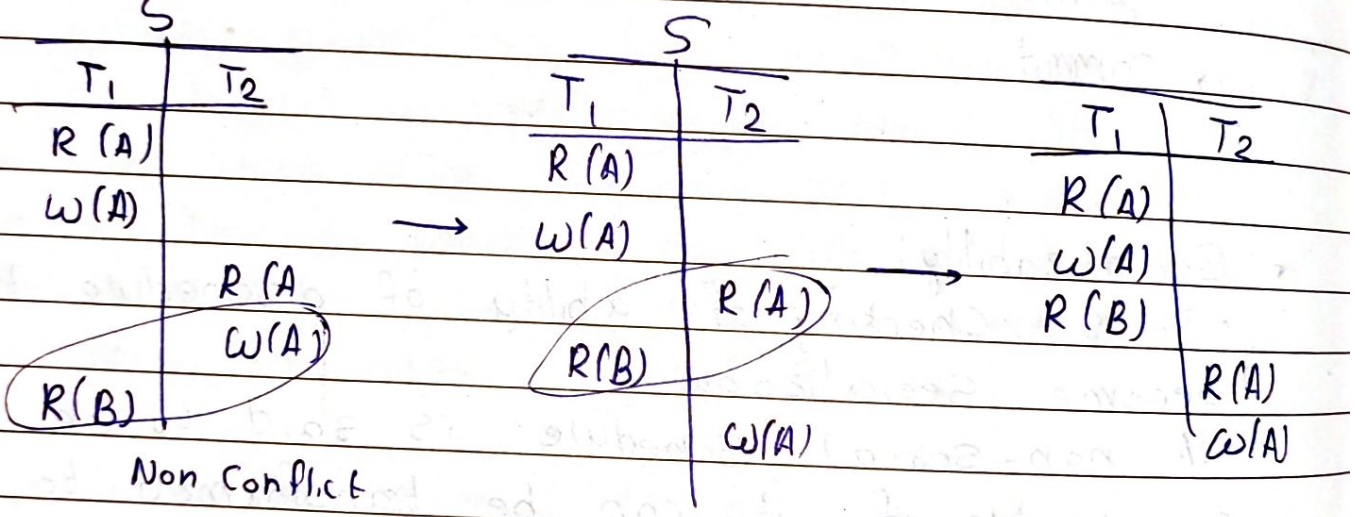
Serial.

Serializability

- Conflict
- View.

There are two ways to check whether any non serial schedule is serializable

1. Conflict Serializability
- In conflict serializability method if the adjacent pairs of transactions are non conflict then we can swap their position.
- A schedule is conflict serializable if it can be transformed into a serial schedule by swapping non-conflict operations.



- It uses a precedence graph (or conflict graph) to test serializability. Conflict.
- The schedule is serializable if precedence graph has no loops.
- We cannot say if the schedule is serializable or not. We can just say it is not conflict serializable if loop is found.
- After loop is found we have to check by view serializability.

Page :
Date :

2. View Serializability

- It is a condition to ensure that the outcome of executing transactions concurrently is equivalent to some serial execution of those transactions.
- A schedule will view serializable if it is view equivalent to a serial schedule.
- If a schedule is conflict serialzable, then it will also be view Serializable.
- View serializability focuses on the read and write operations of transactions and ensures that the views of the data seen by transactions remain consistent with a serial execution.

* View Equivalence : Two schedules are view equivalent if
  - Initial read match : for every transaction, if a read operation reads a value written by a previous transaction
  - Final Write Match : The final write of each data item in both schedules must be by the same transaction.
  - Intermediate Reads Match : If a transaction reads a value written by another transaction, the same write-read relationship must exist in both schedules.

* View Serializable Schedule : A schedule is view Serializable if it is view equivalent to a serial schedule.

* Initial Read:

S1

T_1	T_2
R(A)	
	W(A

S2

T_1	T_2
	W(A)
R(A)	

above two schedules are view equivalent as initial read by S_1 & S_2 is done by T_1 for A in both.

* Update Read.

S1

T_1	T_2	T_3
w(A)		
	w(A)	
		r(A

S2

T_1	T_2	T_3
	w(A)	
w(A)		
		r(A)

Here S_1, S_2 are not view serializable as in S_1 T_3 is reading A updated by T_2 but in S_2 T_3 is reading A updated by T_1

* Final Write

A final write must be the same between both the schedules

S1

T_1	T_2	T_3
w(A)		
	r(A)	
		w(A

S2

T_1	T_2	T_3
	r(A)	
w(A)		
		w(A)

S_1, S_2 are view serializable as final write for A is done by T_3 in both.