Storage and File Structure -1

20 June 2022 17:26

Overview of Physical Storage Media

- Speed with which data can be accessed
- Cost per unit of data
- Reliability
  - data loss on power failure or system crash
  - physical failure of the storage device
- Can differentiate storage into:
  - **volatile storage:** loses contents when power is switched off
  - **non-volatile storage**:
    - Contents persist even when power is switched off
    - Includes secondary and tertiary storage, as well as batter-backed up main-memory

- **Cache**
  - fastest and most costly form of storage
  - volatile
  - managed by the computer system hardware
- **Main memory**
  - fast access (10s to 100s of nanoseconds; 1 nanosecond = 10^{-9} seconds)
  - generally too small (or too expensive) to store the entire database
    - capacities of up to a few Gigabytes widely used currently
    - Capacities have gone up and per-byte costs have decreased steadily and rapidly (roughly factor of 2 every 2 to 3 years)
  - **Volatile**
    - contents of main memory are usually lost if a power failure or system crash occurs

→ Registers are faster than cache but we are not talking about them as they are used for temporary computations & operations & don't act as a storage.

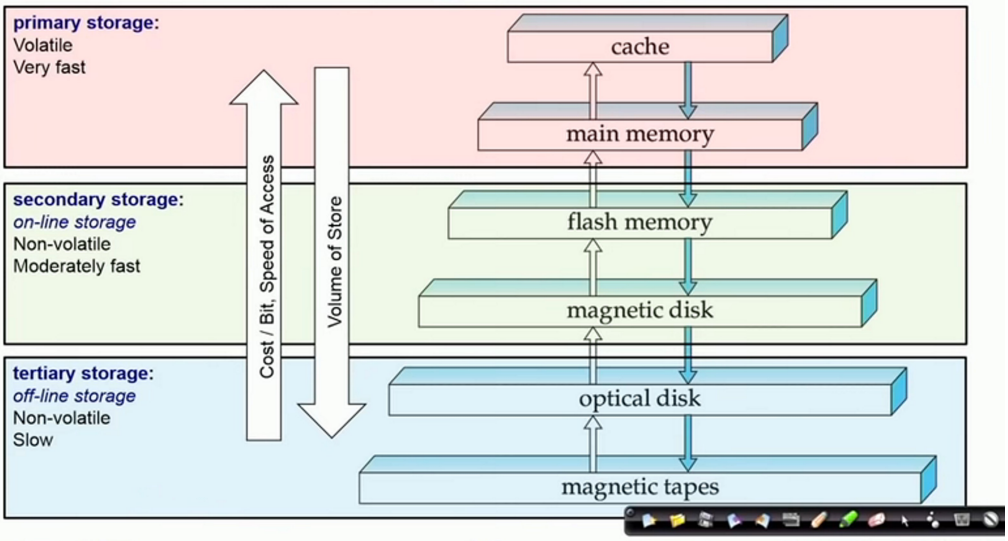
- **Flash memory**
  - Data survives power failure
  - Data can be written at a location only once, but location can be erased and written to again
    - Can support only a limited number (10K – 1M) of write/erase cycles
    - Erasing of memory has to be done to an entire bank of memory
  - Reads are roughly as fast as main memory
  - But writes are slow (few microseconds), erase is slower
  - Widely used in embedded devices such as digital cameras, phones, and USB keys

- **Magnetic-disk**
  - Data is stored on spinning disk, and read/written magnetically
  - Primary medium for the long-term storage of data
    - typically stores entire database
  - Data must be moved from disk to main memory for access, and written back for storage
    - Much slower access than main memory
  - **direct-access**
    - possible to read data on disk in any order, unlike magnetic tape
  - Capacities range up to roughly 16~32 TB
    - Much larger capacity and cost/byte than main memory/flash memory
    - Growing constantly and rapidly with technology improvements (factor of 2 to 3 every 2 years)
  - Survives power failures and system crashes
    - disk failure can destroy data, but is rare

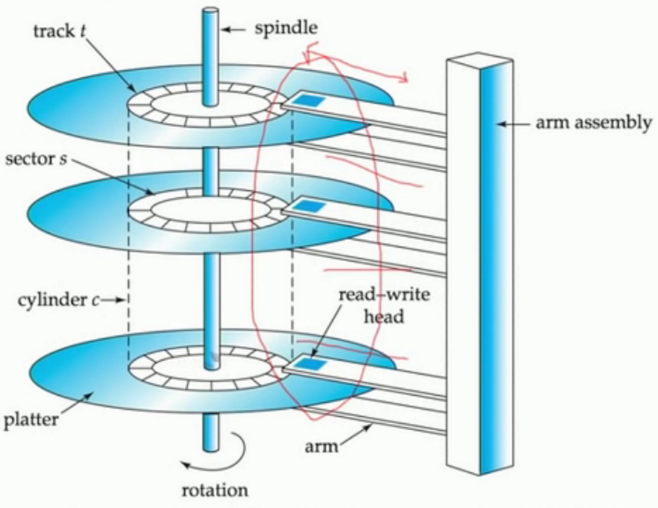
- **Optical storage**
  - non-volatile, data is read optically from a spinning disk using a laser
  - CD-ROM (640 MB) and DVD (4.7 to 17 GB) most popular forms
  - Blu-ray disks: 27 GB to 54 GB
  - Write-one, read-many (WORM) optical disks used for archival storage (CD-R, DVD-R, DVD+R)
  - Multiple write versions also available (CD-RW, DVD-RW, DVD+RW, and DVD-RAM)
  - Reads and writes are slower than with magnetic disk
  - **Juke-box** systems, with large numbers of removable disks, a few drives, and a mechanism for automatic loading/unloading of disks available for storing large volumes of data

- **Tape storage**
  - non-volatile, used primarily for backup (to recover from disk failure), and for archival data
  - **sequential-access**
    - much slower than disk
  - very high capacity (40 to 300 TB tapes available)
  - tape can be removed from drive ⇒ storage costs much cheaper than disk, but drives are expensive
  - Tape jukeboxes available for storing massive amounts of data
    - hundreds of terabytes (1 terabyte = 10^{12} bytes) to even multiple **petabytes** (1 petabyte = 10^{15} bytes)

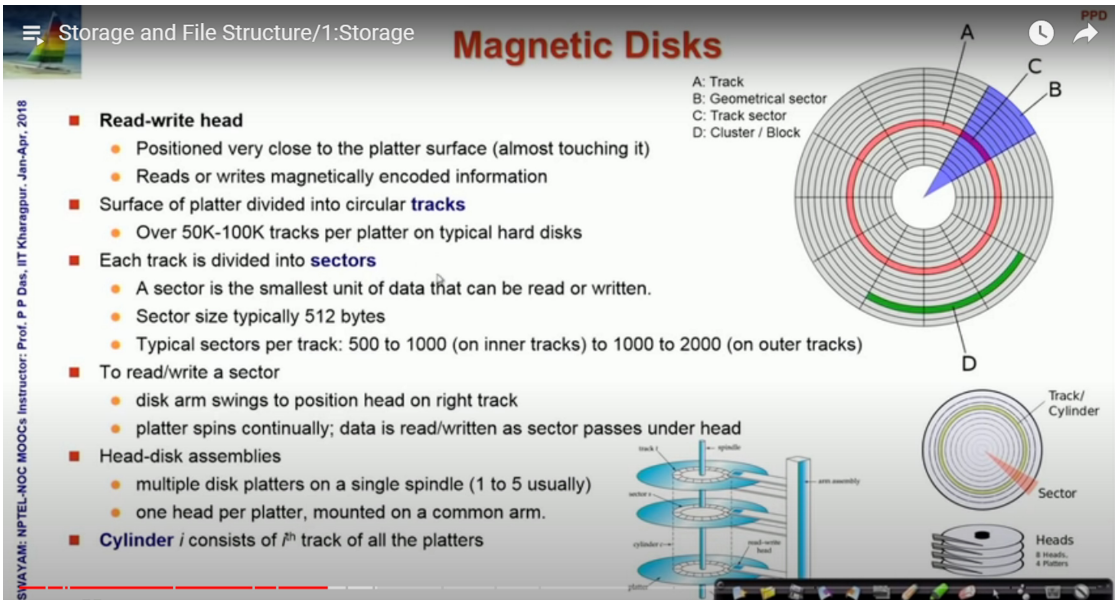
Storage Hierarchy

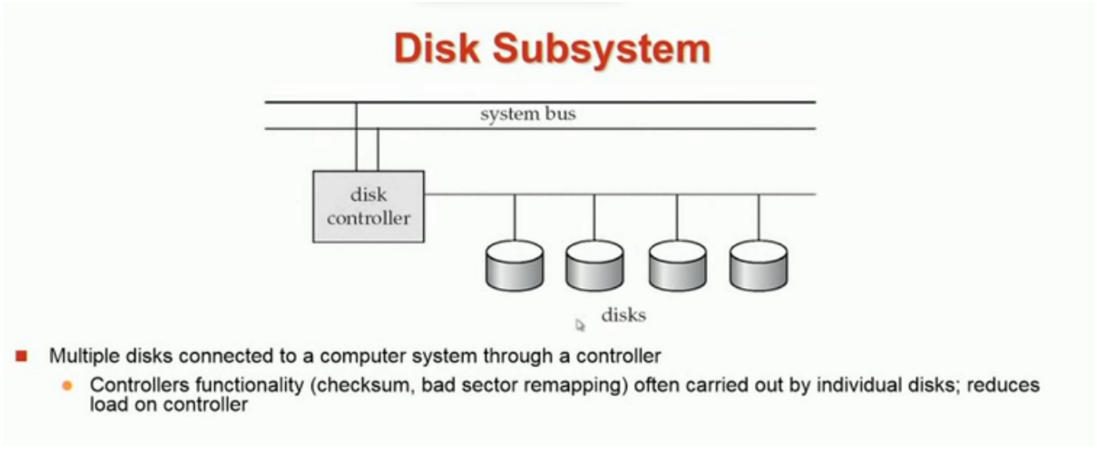


Magnetic Hard Disk Mechanism



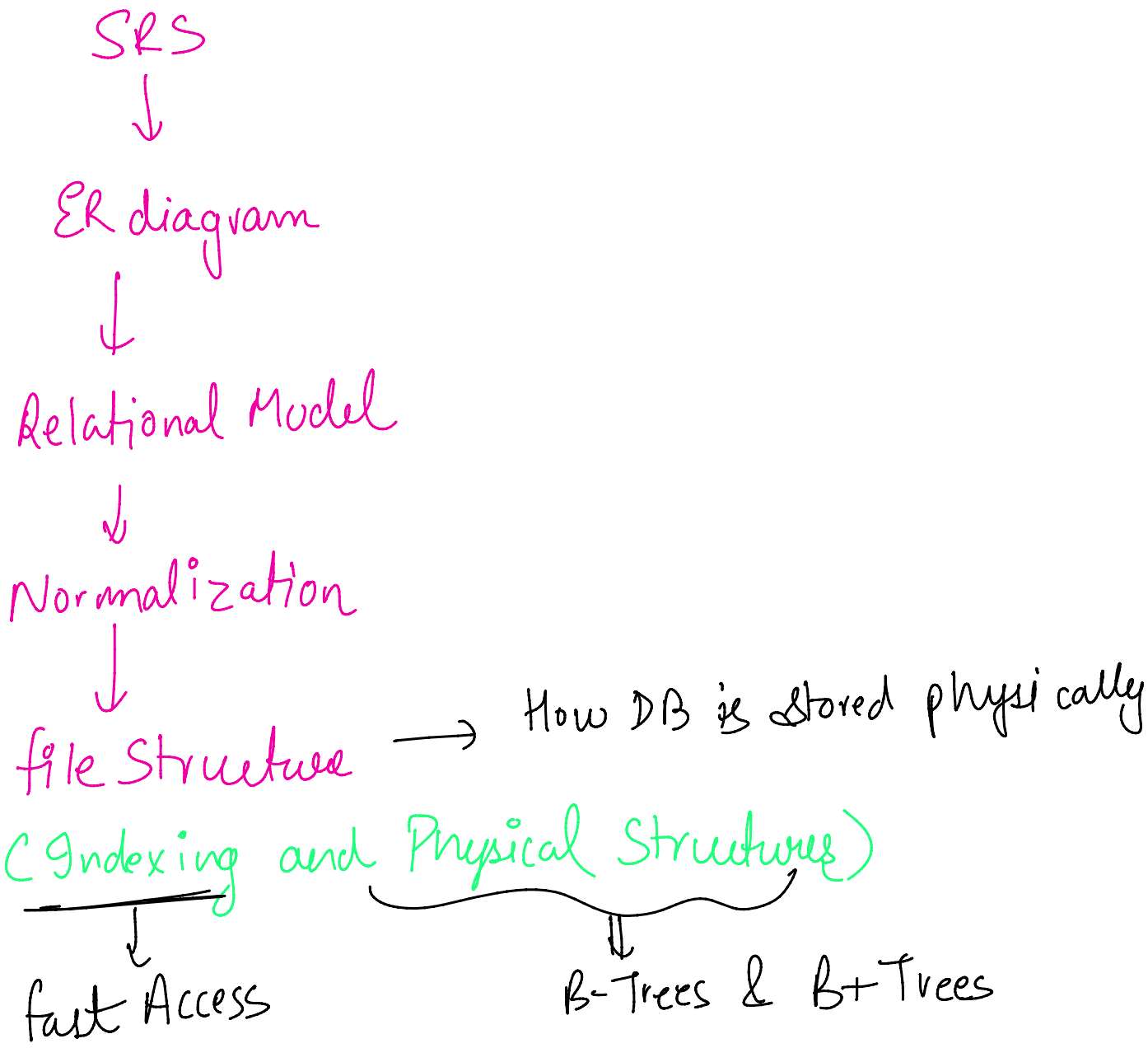
NOTE: Diagram is schematic, and simplifies the structure of actual disk drives


Storage and File Structure/1:Storage
Magnetic Disks
A: Track
B: Geometrical sector
C: Track sector
D: Cluster / Block
A
B
C
D
Read-write head
Positioned very close to the platter surface (almost touching it)
Reads or writes magnetically encoded information
Surface of platter divided into circular tracks
Over 50K-100K tracks per platter on typical hard disks
Each track is divided into sectors
A sector is the smallest unit of data that can be read or written.
Sector size typically 512 bytes
Typical sectors per track: 500 to 1000 (on inner tracks) to 1000 to 2000 (on outer tracks)
To read/write a sector
disk arm swings to position head on right track
platter spins continually; data is read/written as sector passes under head
Head-disk assemblies
multiple disk platters on a single spindle (1 to 5 usually)
one head per platter, mounted on a common arm.
Cylinder i consists of ith track of all the platters
Track/
Cylinder
Sector
Heads
SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur. Jan-Apr, 2018


Disk Subsystem
system bus
disk
controller
disks
Multiple disks connected to a computer system through a controller
Controllers functionality (checksum, bad sector remapping) often carried out by individual disks; reduces load on controller

File Structures

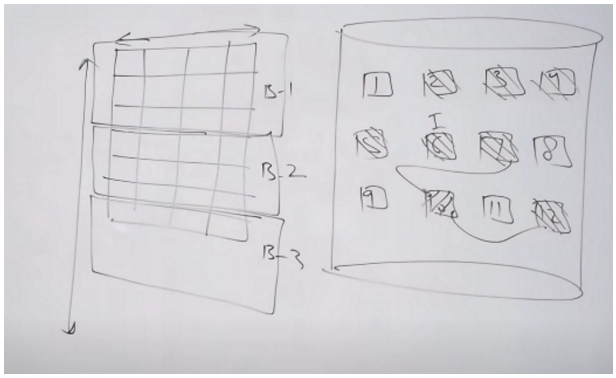
20 June 2022 17:27



Sorted Unsorted File Structures, Spanned Un-spanned Mapping in DBMS

20 June 2022 18:09

In DBMS, we are assuming that we are allocating the memory to a file contiguously i.e starting from a block, the next continous blocks will be filled.



Sorted

Usually, sorting is done based on the attribute that is most frequently used/ searched.

Unsorted

The tuples are randomly stored & any tuple can be any where

⇒ Search : faster in sorted than unsorted.

- sorted → Bin Search $O(\log_2 N)$
- unsorted → Linear Search $O(N)$

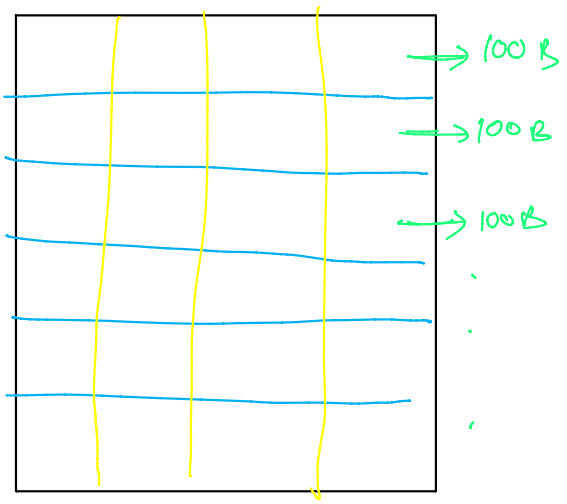
⇒ Insertion : Insertion is easy in unsorted than sorted

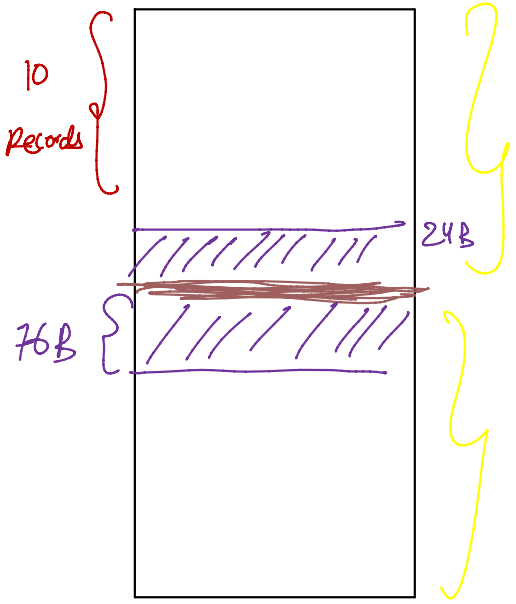
Here, we will have to find the correct position of insertion.

Deleting a particular record can't be compared as such. However, we have to search the record to be deleted & searching is faster in sorted.

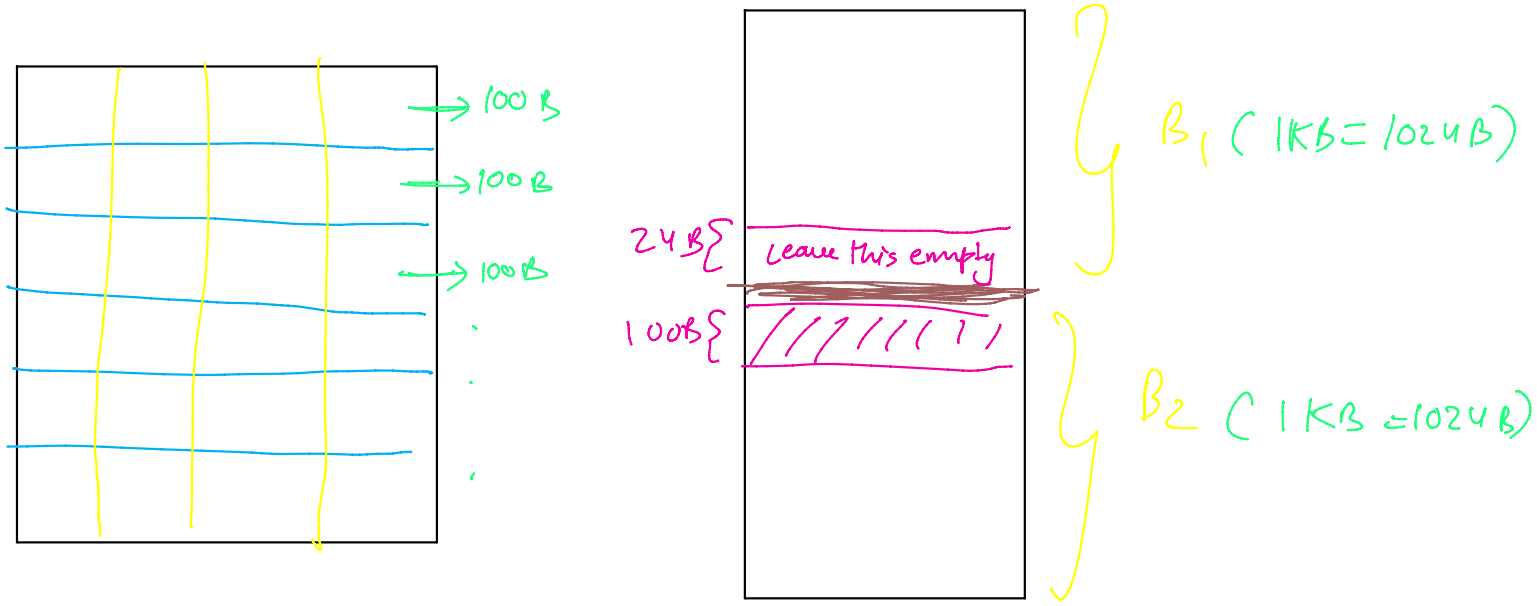
Spanned and Unspanned Mapping

Consider a DB with each record of size 100 B. Now, consider the H/W (Secondary storage) has blocks each of size 1KB (1024B)





Spanned Mapping



Unspanned Mapping

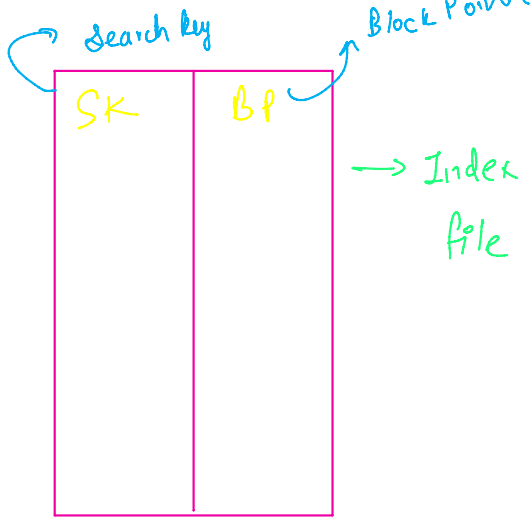
Spanned Mapping → Memory utilization is better

Unspanned Mapping → The time complexity (search time) is better. This is because when we search a record, first we identify the block of memory it is in & then search within the block. So, in unspanned there will always be 1 block. Unspanned is usually used more.

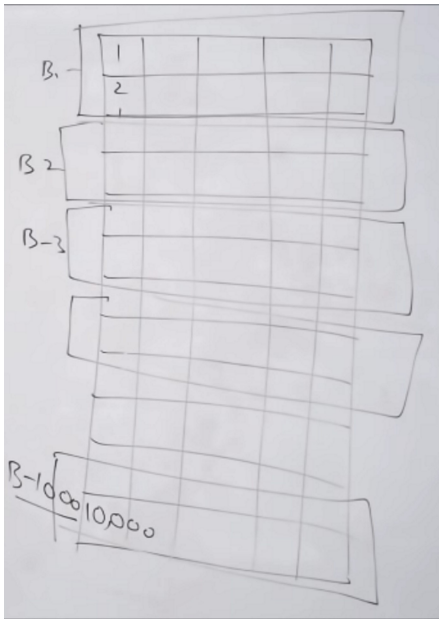
to 1 block. Unspanned is usually used more.

Introduction to Indexing

20 June 2022 19:01



Try to reduce size of index file as much as possible



Indexing does not change the structure of the database as data is stored seperately & indices are created seperately

We usually create spare indexes. This means that we don't store every record in the Index file. Rather we store the records that are at the start of a mem block. The next record will be the first record in the next block & so on.

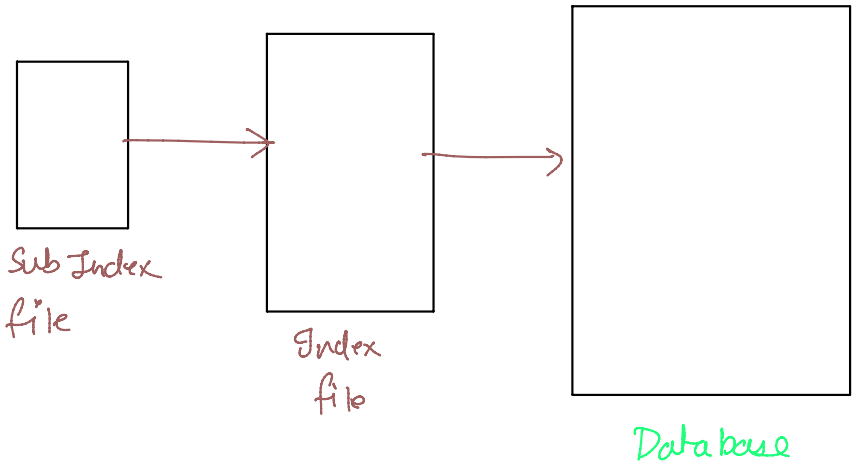
Dense Index: When every record of the DB is stored in the Index file, it is dense indexing. Dense indexing is used in case of unsorted files. This is because we won't be able to get all the records by just storing the starting records in each block as the data is not sorted & we can't be sure of our record being in a particular block.

Types of Indexing

21 June 2022 7:42

Single Level Indexing : Simple indexing that we studied earlier

Multi-Level Indexing : When the size of index file is very large because the database is very large, then we do indexing on index file also called multilevel indexing.



Multi-Level Indexing

Types of Single Level Indexing

Primary Indexing : When the primary key of DB is used as the search key in index table, it is called primary indexing. It is a type of Sparse Indexing. DB is sorted

Clustered Indexing : When the DB is sorted and Primary key is not used as the Search Key in DB, then

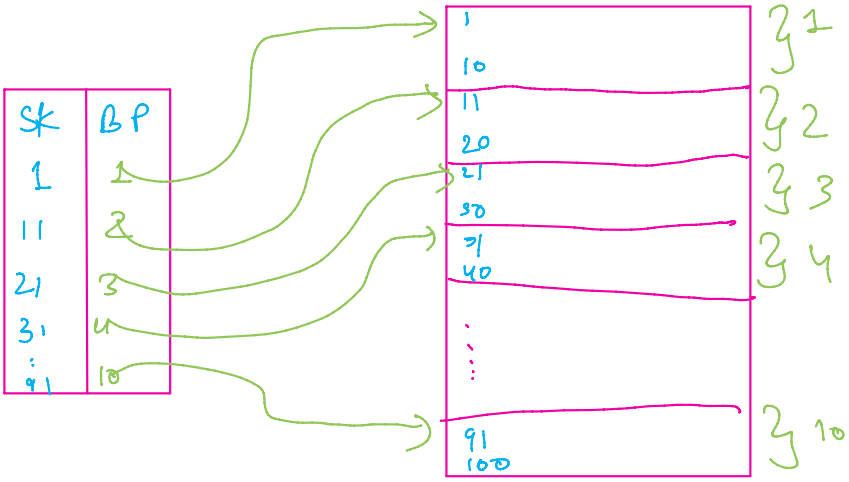
is not used as the search key in DB, then the non-primary key attribute will have some repeating values. This is called as clustured indexing.

(This is what I have in my Project)

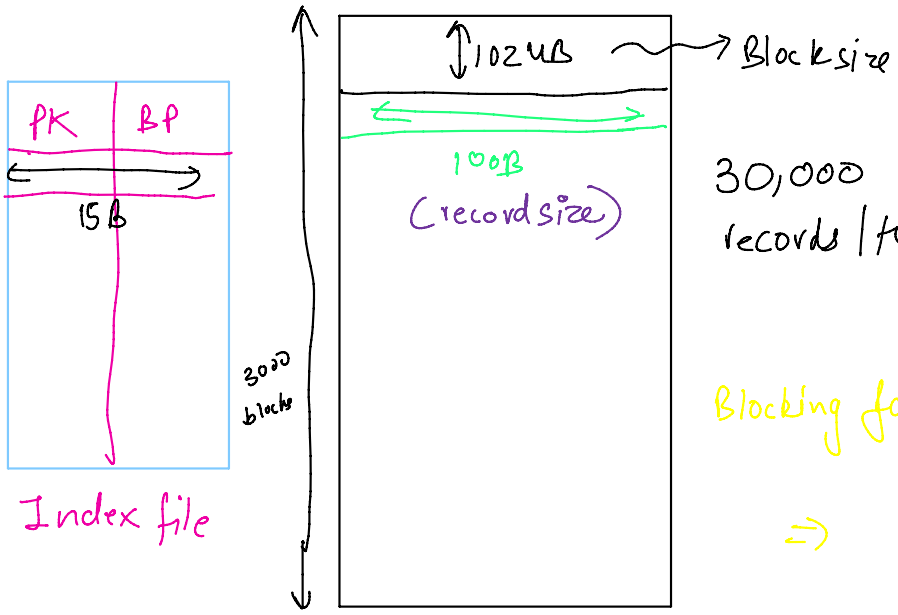
Secondary Indexing: When the DB is not sorted, the indexes created upon any attribute (whether PK or not) are called secondary indexes.

Primary Indexing

21 June 2022 8:27



No of block access = $\log_2 N + 1$ → to reach the data block from index



Blocking factor = $\lfloor \frac{\text{Block size}}{\text{Record size}} \rfloor$ (floor value)

$\Rightarrow \frac{1024}{100} = \lfloor 10.24 \rfloor$ = 10 records per block

No of blocks acquired by the main file = $\lceil \frac{\text{Total no of tuples}}{\text{Blocking factor}} \rceil = \frac{30,000}{10} = 3000$ (ceil value)

Access Time $= \log_2 N = \log_2 3000 = \lceil 11.55 \rceil = 12$

N ↓ total no of blocks acq by the DB

$\lceil \cdot \rceil$ → ceil value

Now, let us see the access time using the index file.

Blocking factor for index file $= \frac{1024}{15} = \lfloor 68.266 \rfloor = 68$

Total No of blocks acquired by index file $= \frac{\text{Total No of Records in Idx file}}{\text{Blocking factor}} = \frac{3000}{68}$

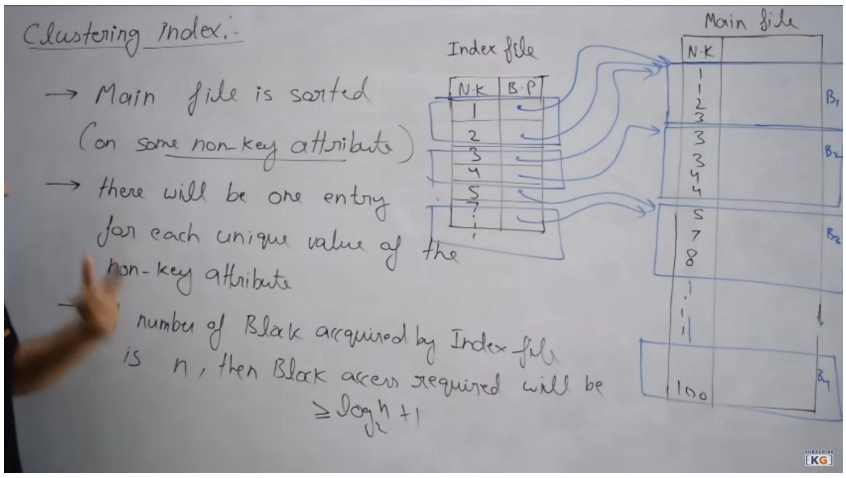
$= \lceil 44.11 \rceil = 45$

Access Time $= \log_2 45 + 1 = \lceil 5.49 \rceil + 1$

$= 6 + 1 = 7$

Clustering Index

21 June 2022 9:02



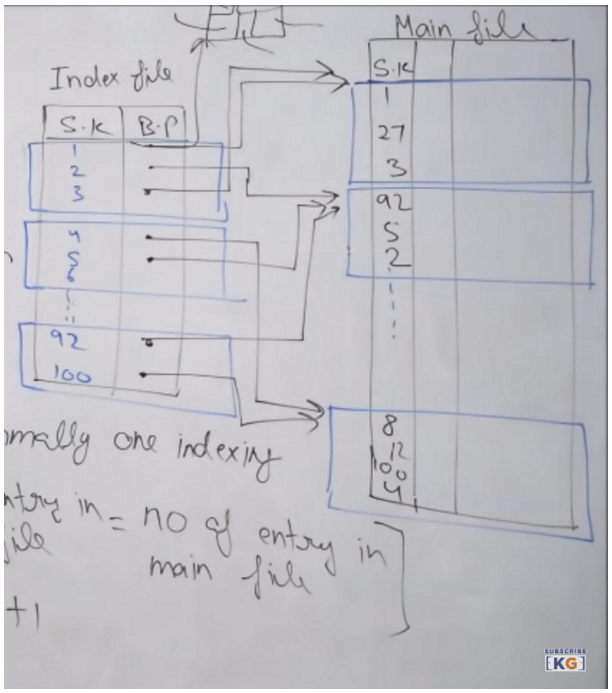
Here, we are pointing at every unique value. So, there might be multiple tuples in the Idx file pointing to the same block & it is also possible that a Block might not have even a single pointer on it.

Every record of the main file is not getting an entry in the index file. So, it behaves like sparse indexing.

Now, every value of the DB is getting an entry in the index file. So, it behaves like dense indexing as well.

Secondary Indexing

21 June 2022 9:11



→ Main file is unsorted

→ Search key in index file can either be a PK or even a Non PK.

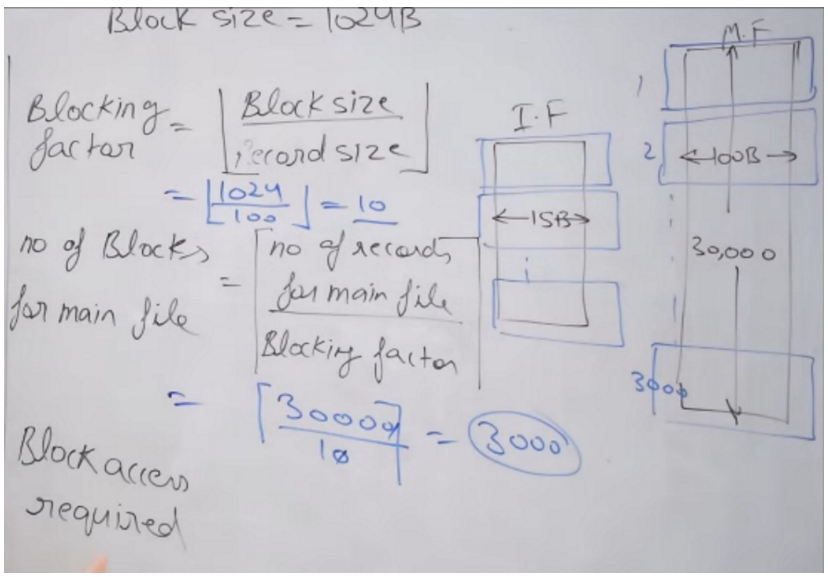
→ Since we know that the values can be repeating since SK might not be a PK, it can be handled in diff ways

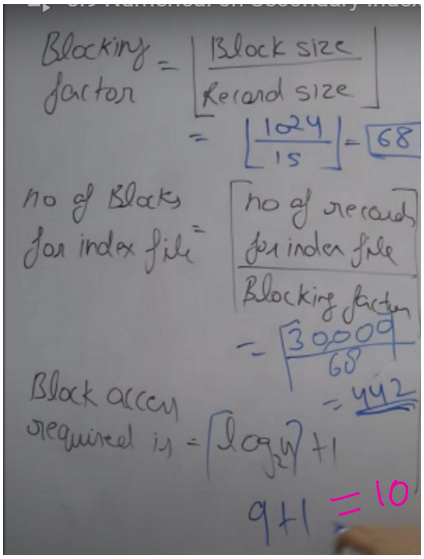
(Base Pointer → Array List)

→ No of entries in idx file = No of entries in main file.

Search Time = $\lceil \log_2 N \rceil + 1$

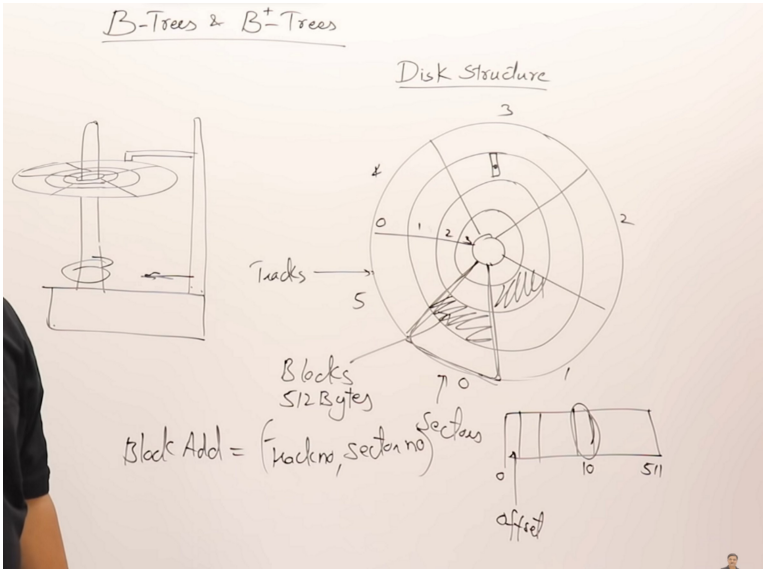
(i)

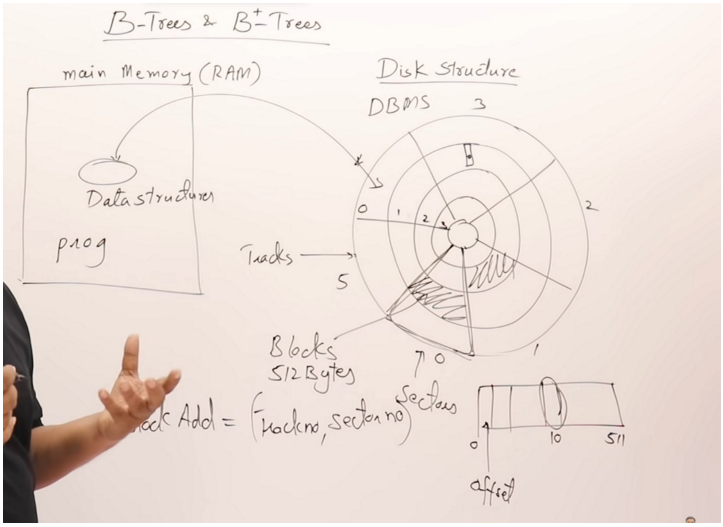



Blocking factor = ⌊Block size / Record size⌋
= ⌊1024/15⌋ = 68
no of Blocks for index file = ⌈no of records for index file / Blocking factor⌉
= ⌈30000/68⌉
= 442
Block accesses required is = ⌈log2(y)⌉ + 1
9 + 1 = 10

B and B+ Trees

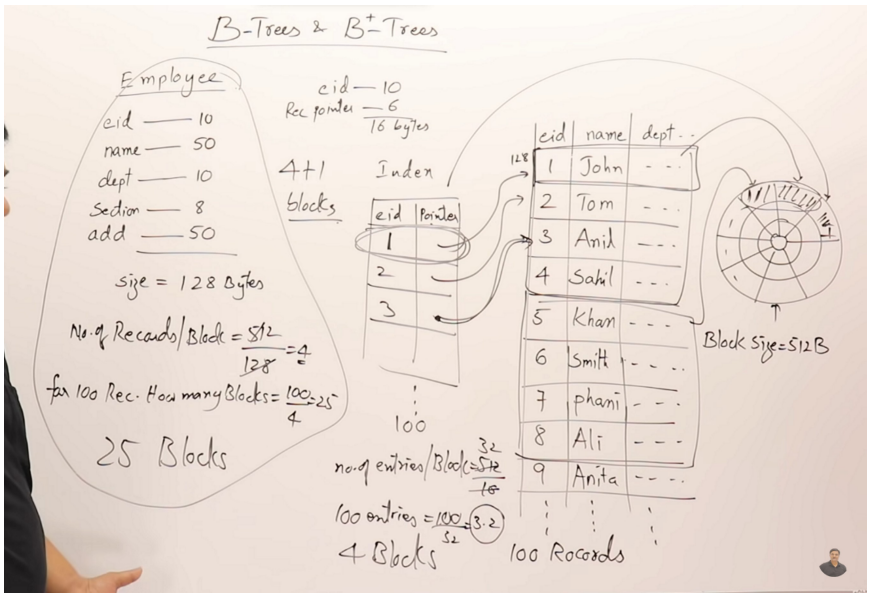
21 June 2022 10:23





The study of efficient structuring of data in the main memory that is directly utilized by the program in the main memory is data structure.

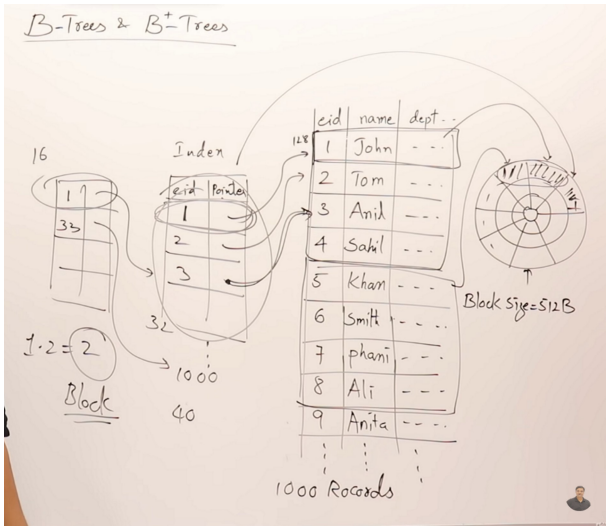
The study of efficient storing of data on the disk is DBMS.



Both the data and index are stored on the disk only.

The pointer pointing to a record of the DB in the index file is called as a record pointer.

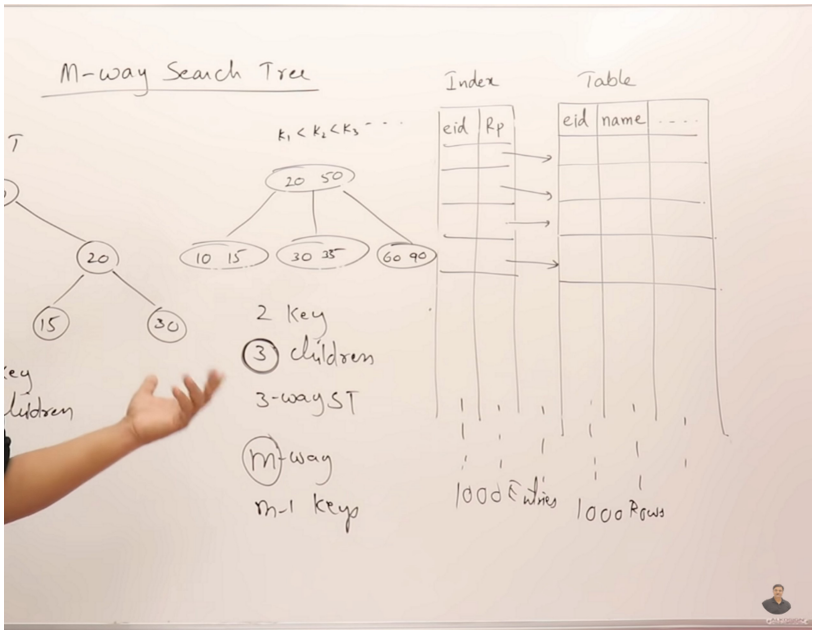
(Here, we have showed dense indexing)



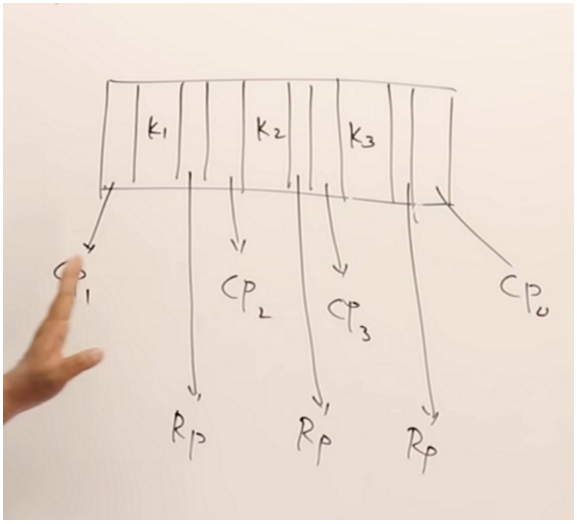
→ Creating multi-level index because the size of index file is also large & searching in index file is also a huge task.

→ Multilevel indexes reduce the search time a lot. They are always sparse as we store blocks of index table as an entry in multi-level index & never one record per entry.

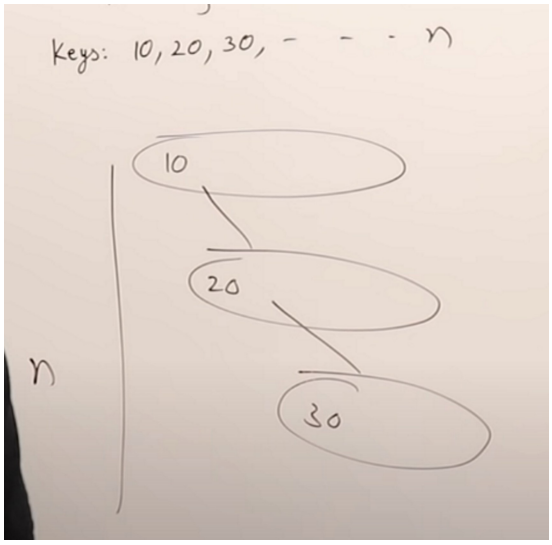
We don't want to check manually that the size of DB & index table is very large so, implement multi-level index. We want an automatic system for creation & deletion of multilevel indexes as the size of the DB grows & reduces respectively ⟹ This is the idea behind B & B+ Trees.



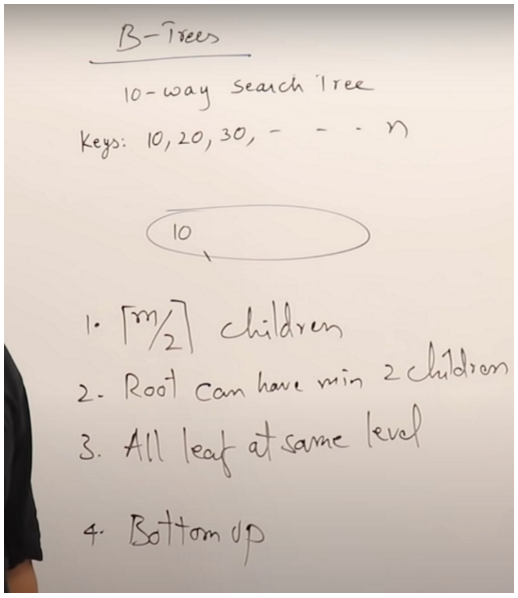
→ Binary Tree is a 2-way search tree.



⟹ Node structure of an M-way search Tree for multi-level indexing

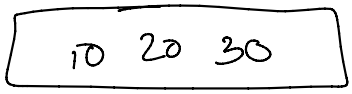


The problem with M-way search Trees is that there are no proper guidelines for insertion. So, if we insert the keys as shown, it will lead to a height of O(N) i.e search will become linear search.

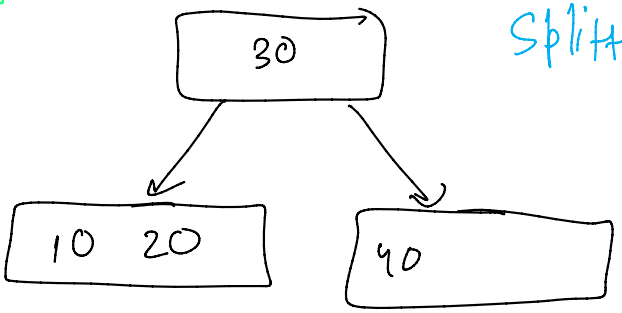


Rules for creating a B-Tree.

① keys : 10, 20, 30, 40 given m = 4 → means max 4 children
so, max 3 keys in a Node

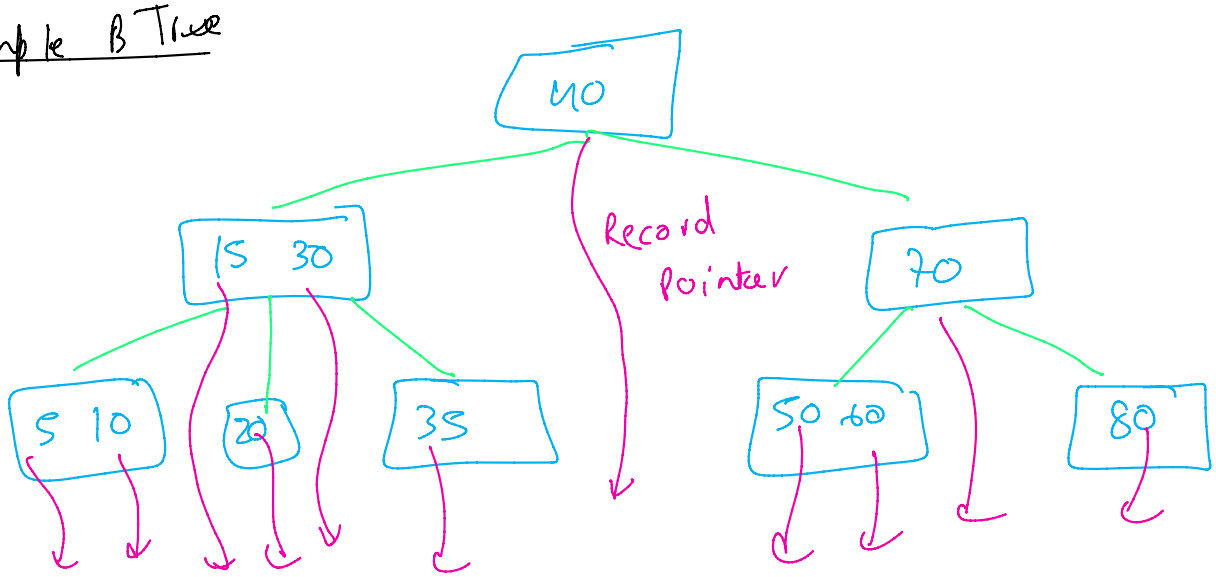


② Inserting 40

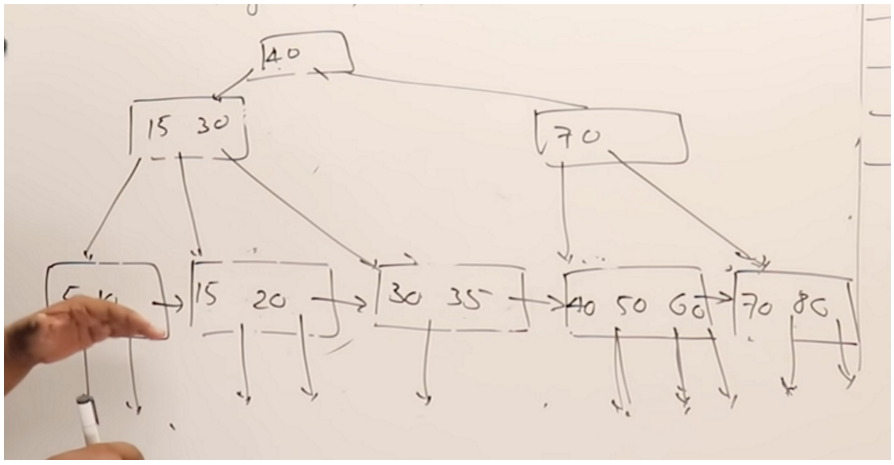


Splitting takes place & creation is bottom-up

Example B Tree



Example : B+ Tree



Leaf nodes have all the keys.

Non-leaf nodes just are a way to reach the leaf nodes where all the data is present.

Record ptrs are only with the leaf nodes-

Query Processing and Optimization

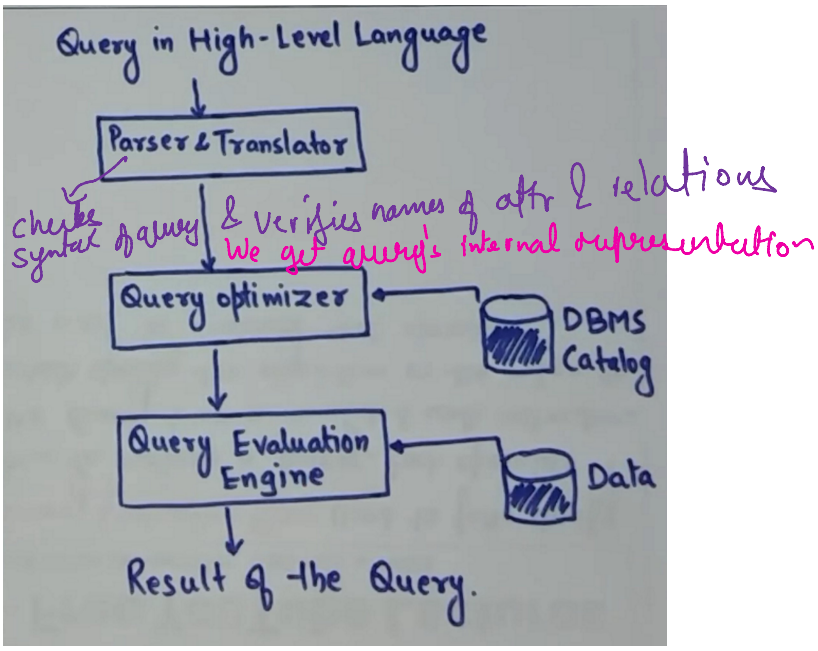
23 June 2022 11:08

Query Processing: Includes translation of high-level queries into low-level expressions that can be used at the physical level of the file system, query optimization and actual execution of the query to get the result.

Query optimization: It is the process in which multiple Query-execution plans for satisfying a Query are examined and a most efficient Query plan is identified for execution.

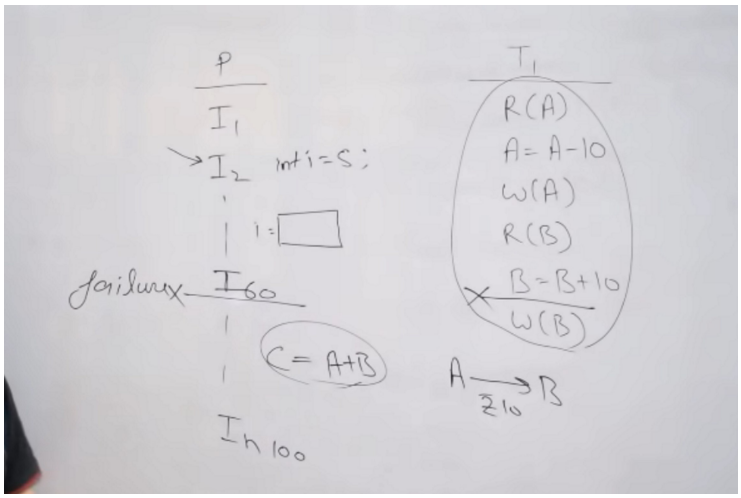
Query Processing steps:

Consists of Parsing and translation, optimization and execution of the Query.



Basic Idea of Transaction in DBMS

23 June 2022 11:16



Instructions are always atomic in nature.

↓

Due to computer architecture.

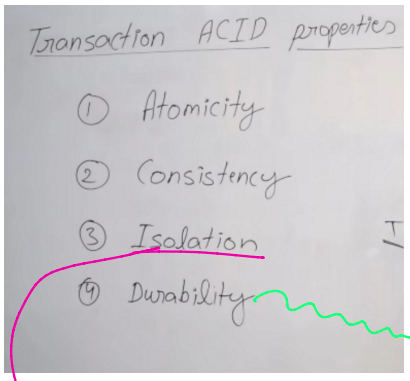
Transaction is a set of instructions that perform a logical unit of work. & that must be atomic in nature.

OS ki taraf fraction me kaam nahi kar sakte.

Agar 10 me se 6 instructions execute hone k baad failure ho gaya, 6 ins ko undo karna padega called as rollback.

ACID Properties

23 June 2022 11:38



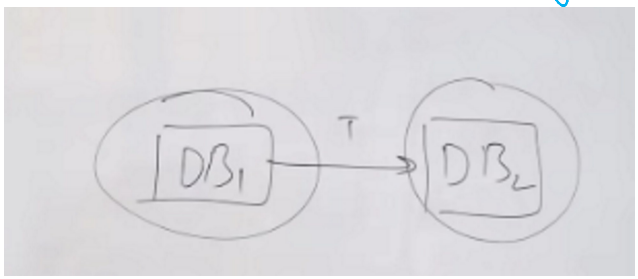
DBMS has a part called transaction management component that ensures atomicity.

changes must persist. (Recovery Management Component)

Isolation is not actual but logical. This means that one transaction should not affect another.

Concurrency control component takes care of isolation

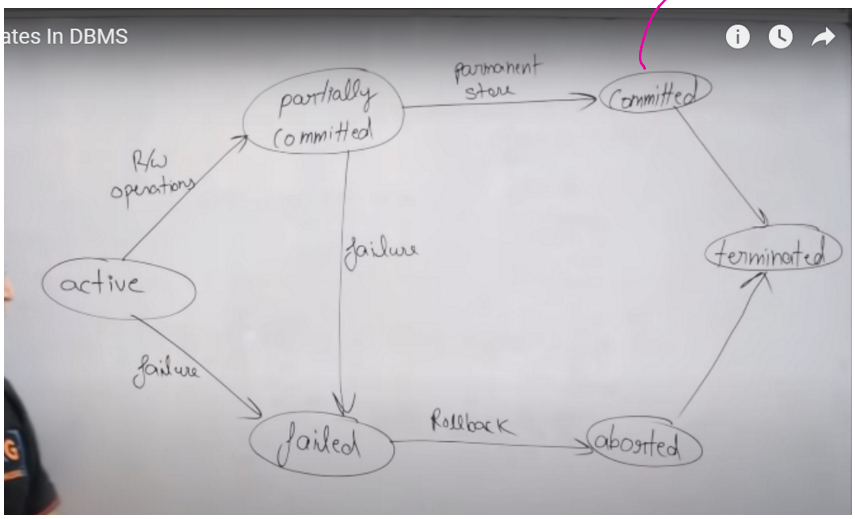
Consistency: If a DB was consistent before a transaction, it must be consistent after the transaction also.



There is no component of DB which takes care of consistency.

Transaction States

23 June 2022 11:50



Now, we can't rollback.
(We can run a
compensating transaction
but can't rollback the
same transaction once
commited.

Advantages of Concurrency

23 June 2022 17:02

Concurrency: Means running multiple transactions at the same time.

This means context switching only. We are not running multiple instructions simultaneously

Waiting Time ↓

Response Time ↓

Resource Utilization ↑

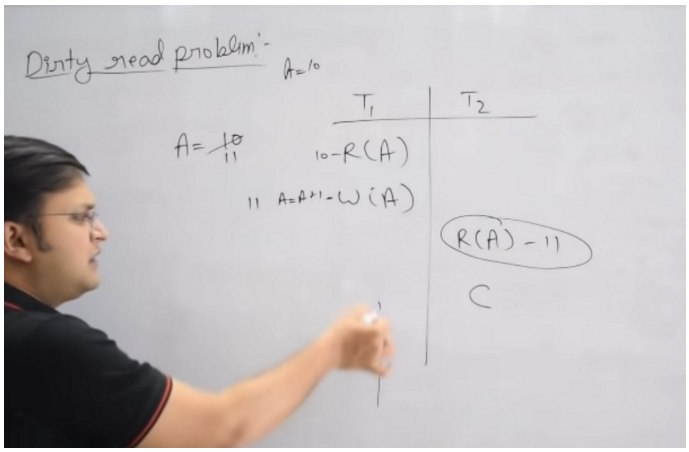
Efficiency ↑

Disadvantage is management becomes difficult.

Dirty Read Problem

23 June 2022 17:11

Agar koi transaction DB me se value read karne k bajae, local buffer me se kisi uncommitted transaction ki value ko read kare, this is called as dirty read problem.

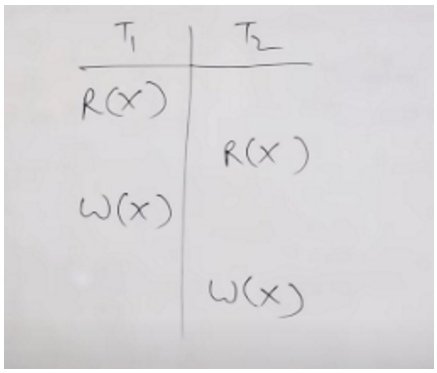


Now, there is no problem if T1 completes successfully. But, if T1 fails while reading the end, it will rollback & the value in DB will be unchanged (10) while T2 got committed with value 11 & can't rollback.

So, solution is that the transaction doing dirty read should get committed after the transaction it is doing dirty read from.

Unrepeatable Read Problem

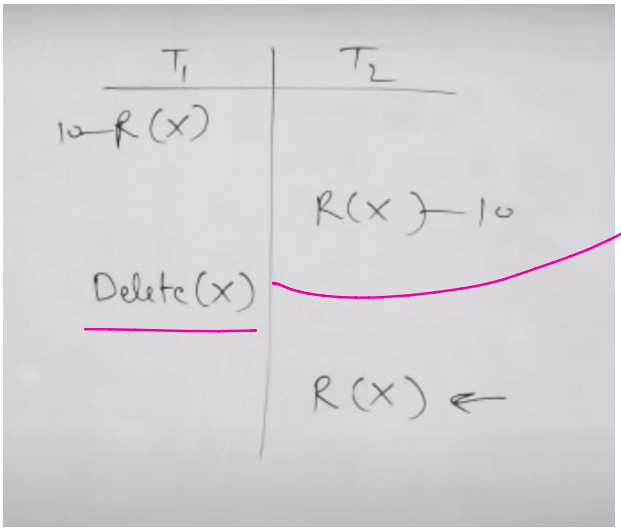
23 June 2022 17:22



T1 and T2 both think that they are isolated. T1 reads the value initially it is 10. T2 reads it & it is still 10. Now, T1 modifies it to 11 & now T2 reads it & it's 11. So, T2 gets confused that if it is isolated i.e if it is the only transaction in the system, how did the value change. So, the read operation when repeated for T2 caused a problem. Hence this is called unrepeatable read.

Phantom Read Problem

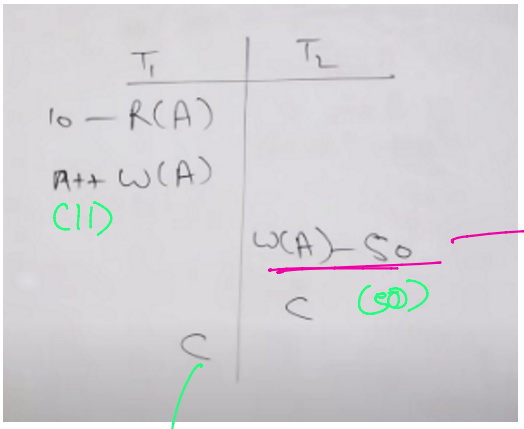
23 June 2022 17:46



is bavar variable change nahi hua, delete ho gaya.

Lost Update Problem (Write-Write Conflict)

23 June 2022 17:50

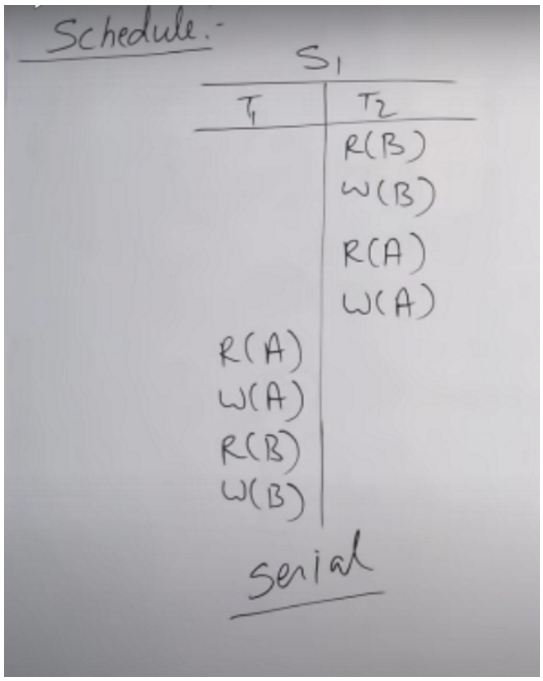


→ Directly write without read is called as Blind Write

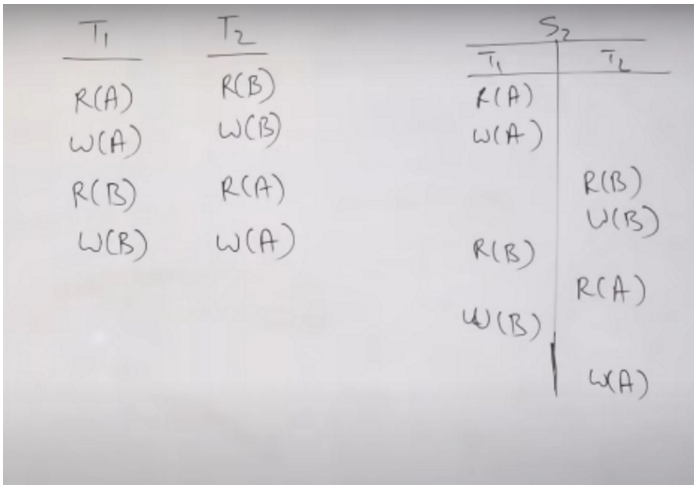
Since T_2 has been committed first, value 50 is there & now when T_1 commits, it thinks value will be 11 but it is 50.

Serial and Non-Serial Schedule in Transactions

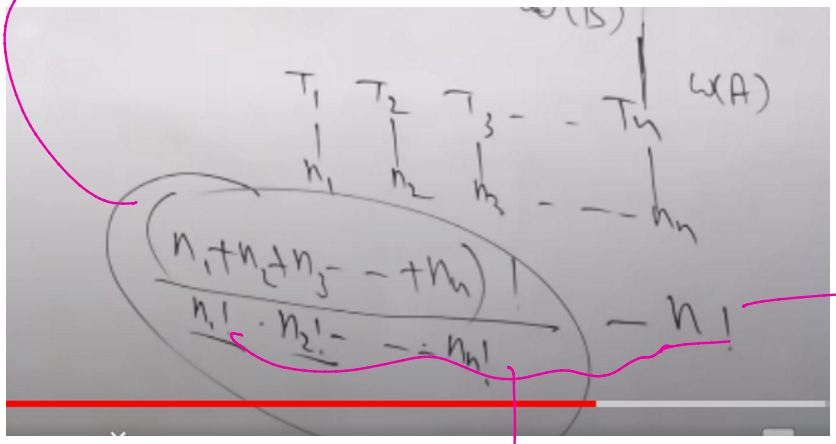
23 June 2022 17:56



→ When context switching does not take place it is serial schedule. else Non-serial.



Total No of schedules

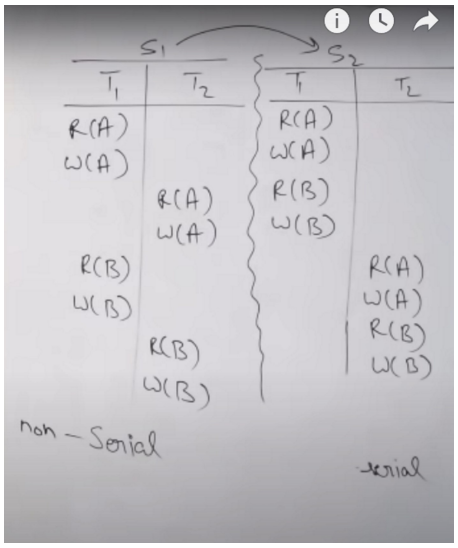


→ No of serial schedule

No of Non-serial schedules

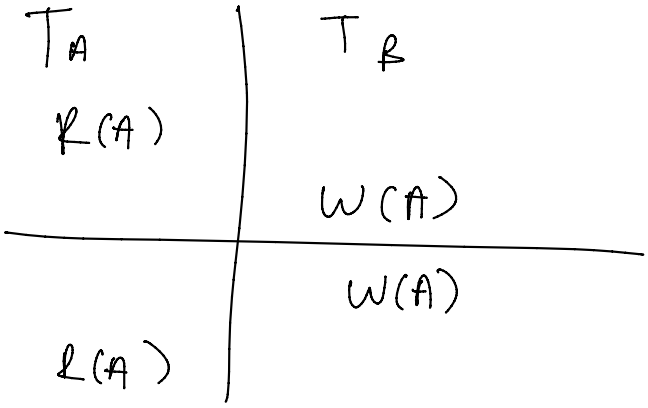
Conflict Serializable Schedule with Precedence Graph

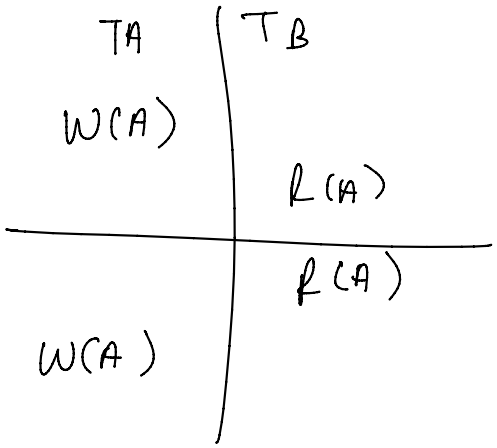
23 June 2022 18:06



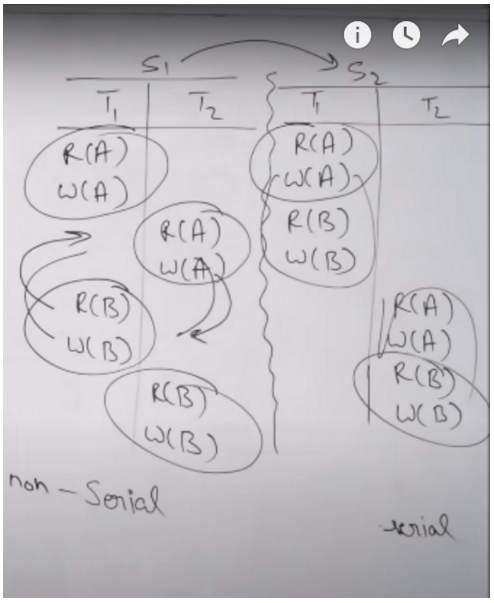
Instructions that can be swapped are non-conflicting & others are conflicting.

If instructions are on different transactions & on same data item & even one of them is Write, there will be a conflict.



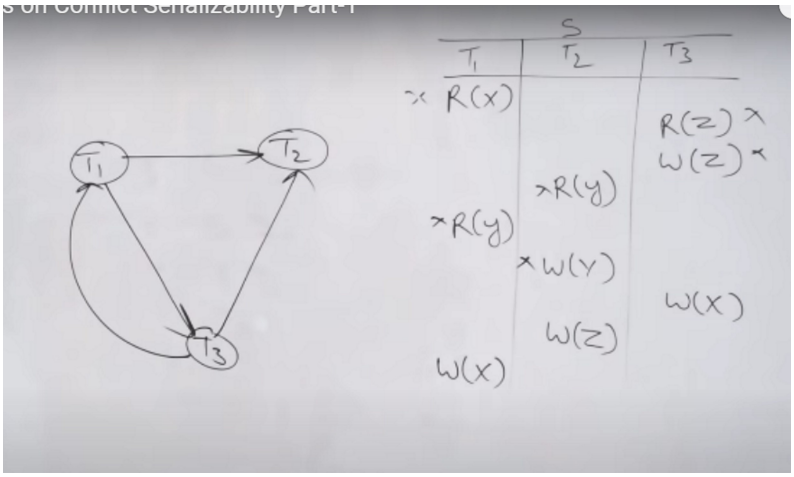


Similarly write-write.

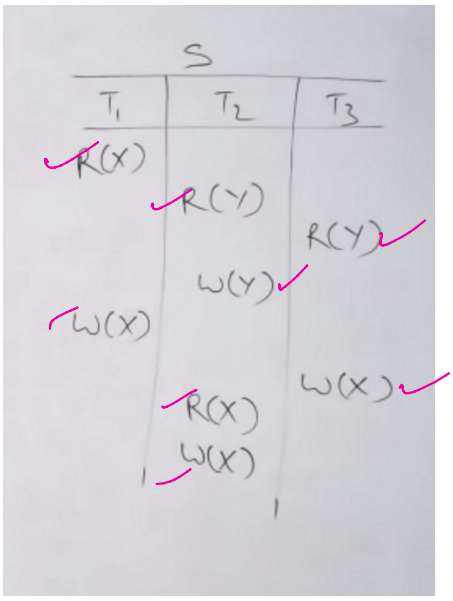


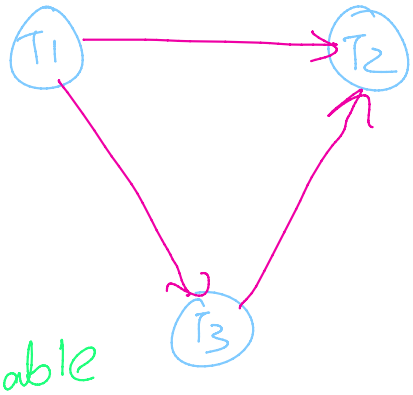
When a non-serial schedule is converted into a serial schedule by swapping non-conflicting instruction, the non-serial schedule is called

conflict - serializable schedule.



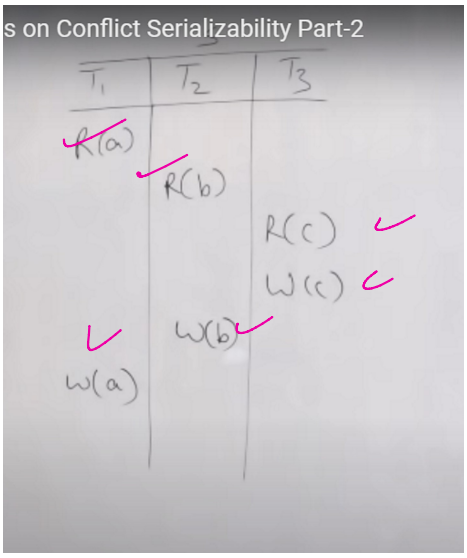
If graph is cycle, schedule is not conflict serialisable.





conflict

serialisable

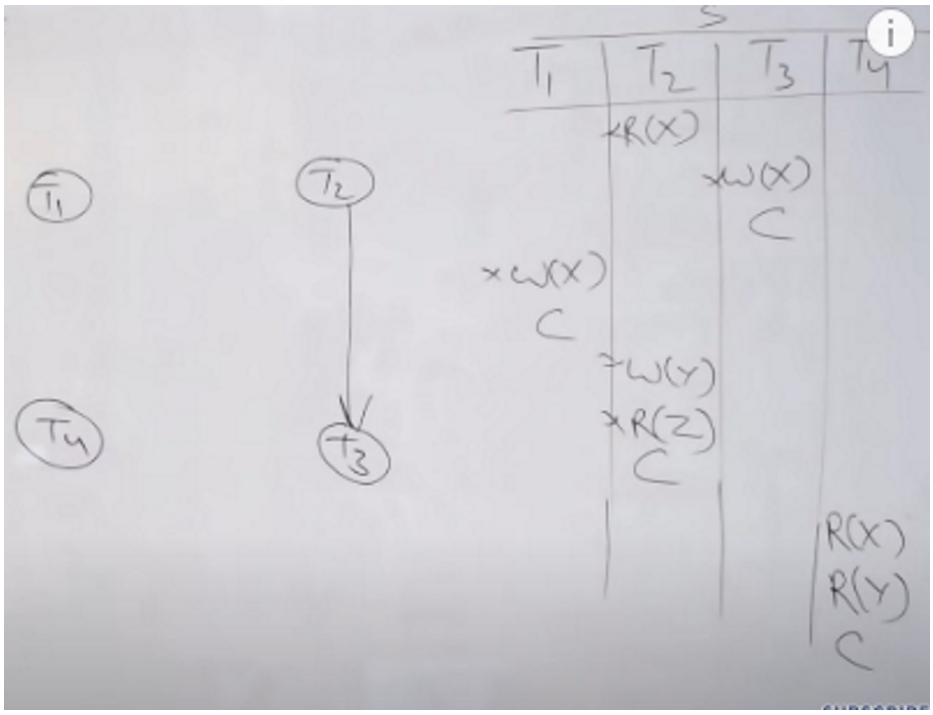


T1 T2

conflict

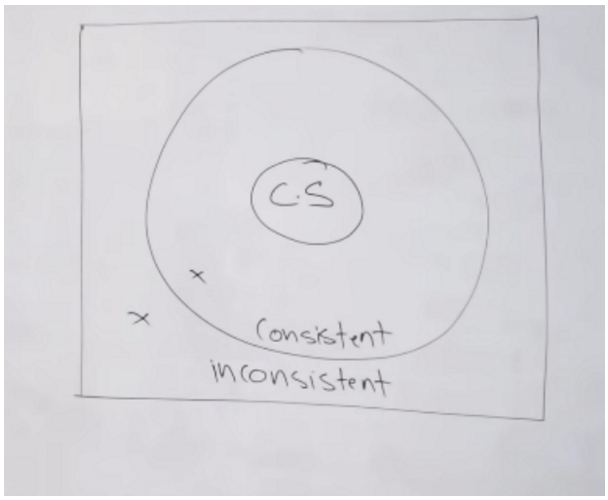
Serilizable

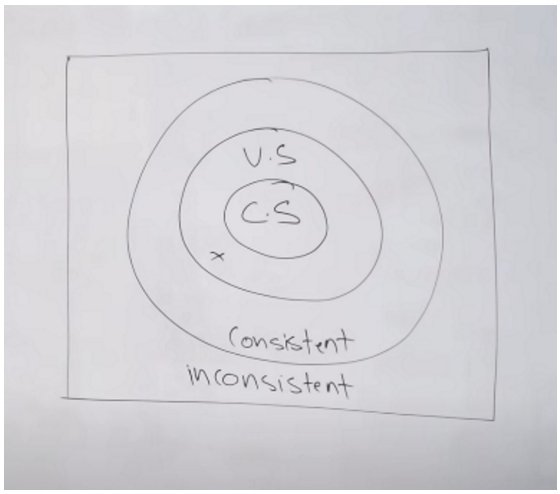
T3


S
T1
T2
T3
T4
R(X)
W(X)
C
W(X)
C
W(Y)
R(Z)
C
R(X)
R(Y)
C
T1
T2
T4
T3

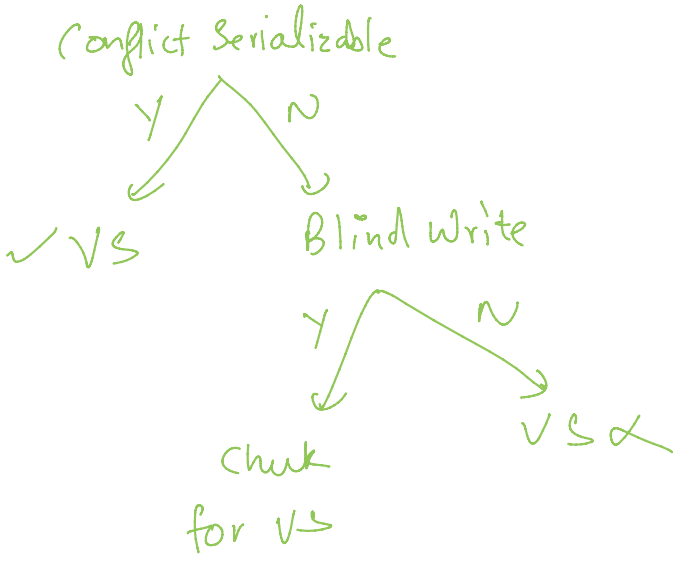
View Serializability

23 June 2022 20:56





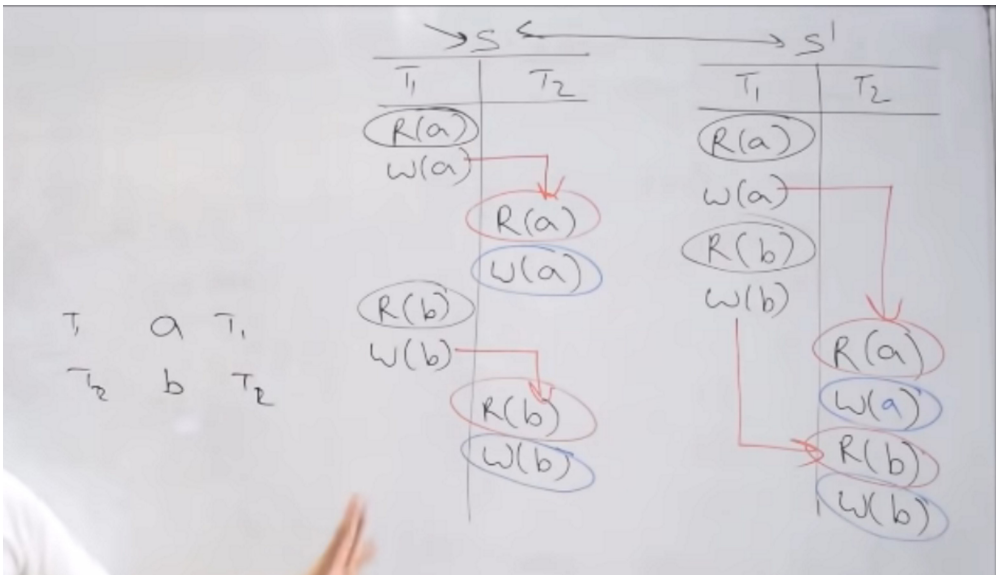
Agar koi schedule conflict serializable nahi hai but view serializable hai, usme pakka kam se kam 1 blind write hoga.



View Equivalence: # Initial read on same data item must be same

final write on same data item must be same

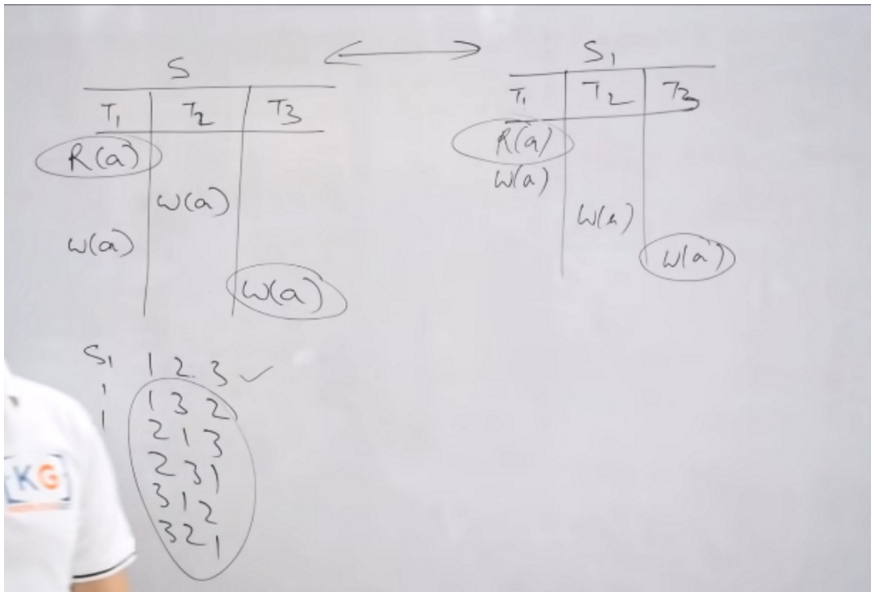
Intermediate reads on a data item should be same (b/w same transactions)



View Serializable prove karne k liye ek non serial schedule ko usk have possible serial permutation k sath V.E. prove karne ki koshish karte hai jab tak kisi ek k saath prove na ho jae.

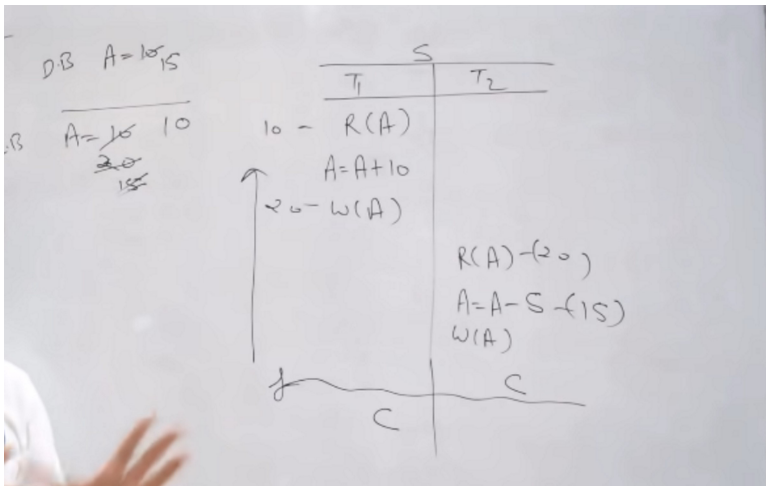
Hence it is so difficult to compute it is an N-P

complete problem.

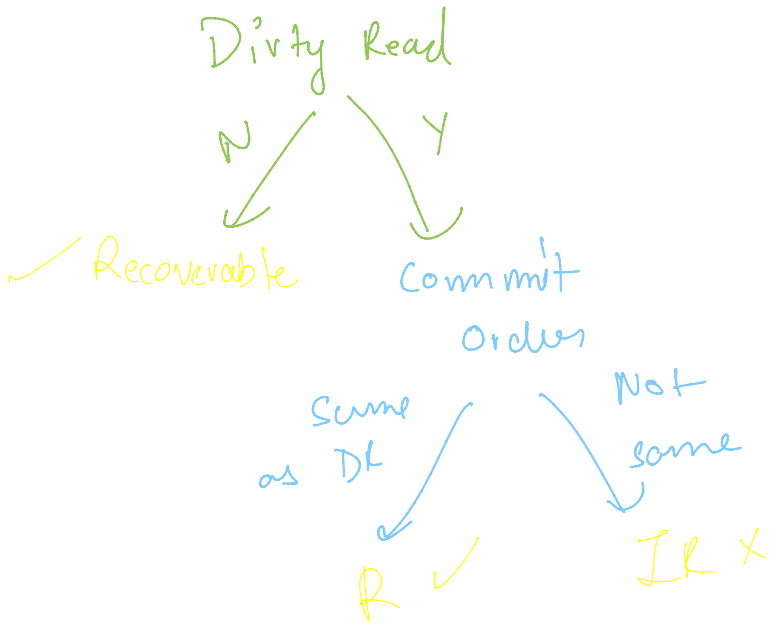


Recoverable Schedule in DBMS

24 June 2022 9:00



Even though S was serial, the DB is now in an irrecoverable state.



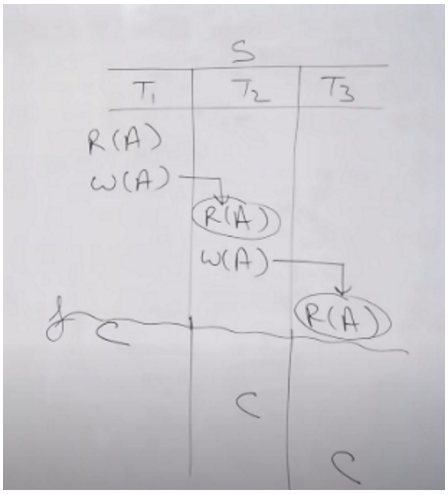
Recoverability is a necessary property.

Cascadeless Schedule in DBMS

24 June 2022 9:12

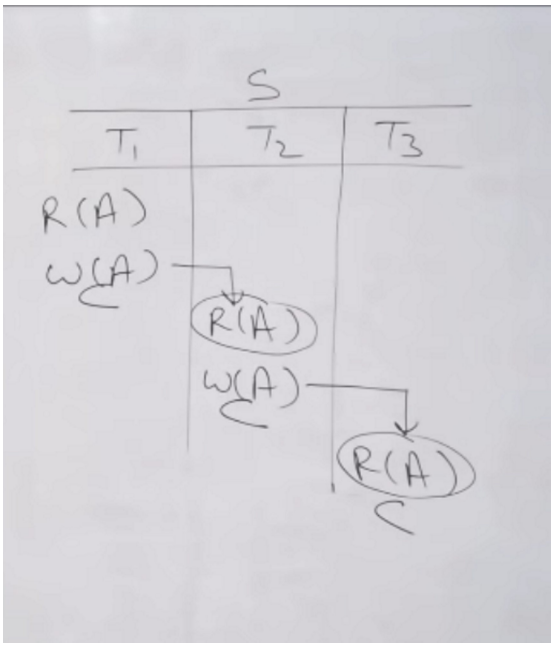
consistency is there, but not efficiency

Cascading Rollback



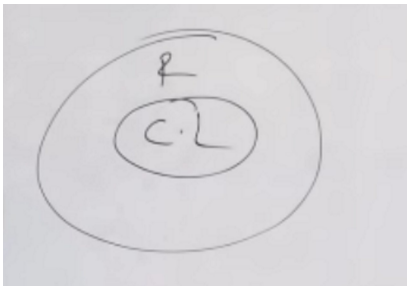
Jab ek transaction ko dekh ke aur transactions Rollback kare, then it is cascading rollback.

So, a schedule without any cascading Rollbacks is a cascadeless schedule.



Cascadelessness is optional.

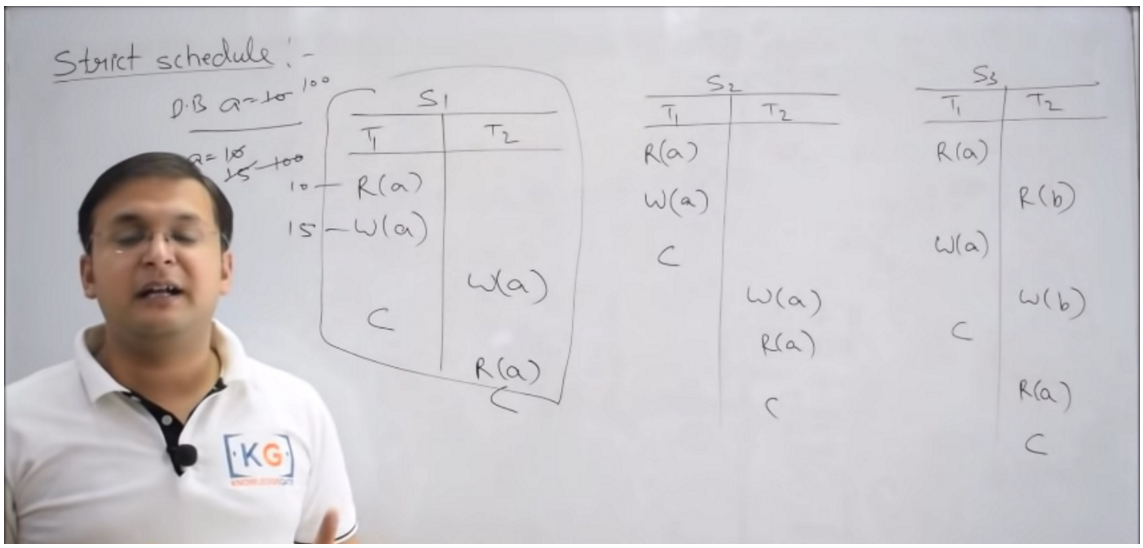
Dirty Read remove karke schedule cascadeless ho jata hai.

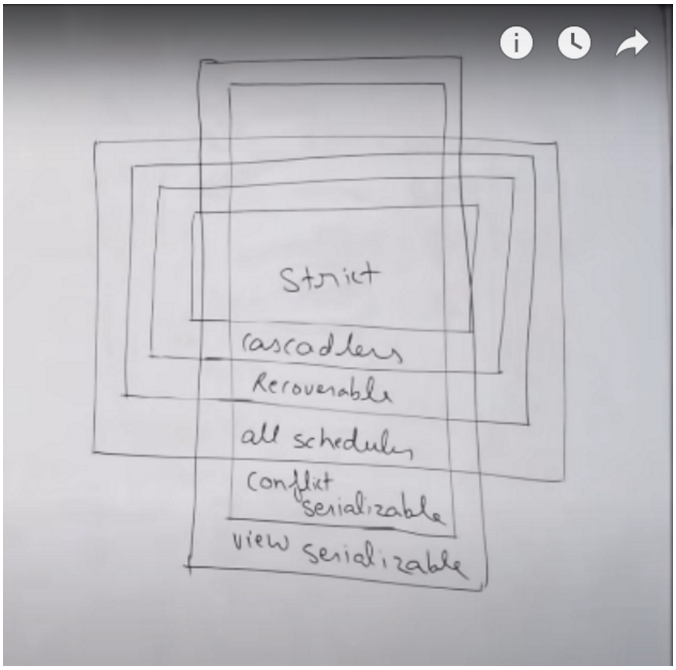


Strict Schedule

24 June 2022 9:22

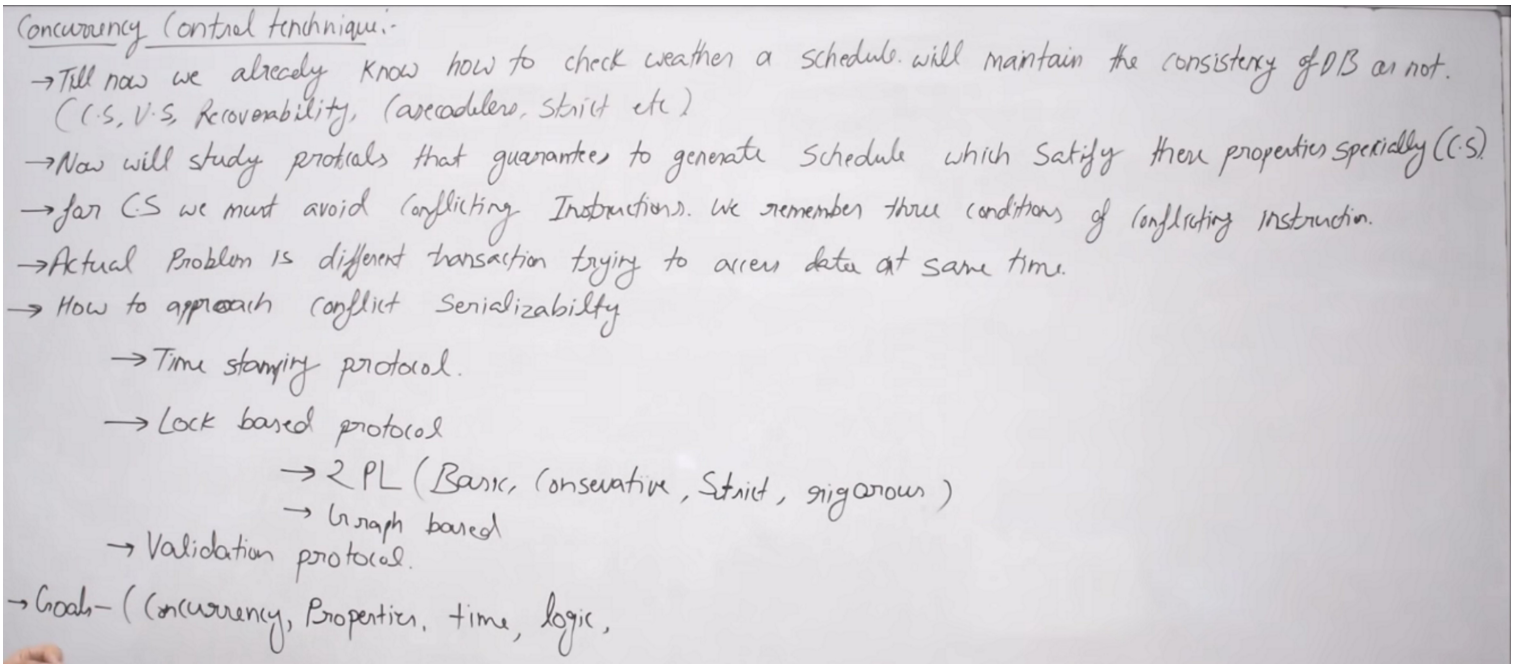
Jab tak ek transaction ek data item pe write operation karne k baad usko commit nahi kar deta tab tak koi aur trans us data item ko R/W nahi kar sakti. Aise schedule ko strict schedule kehte hai.





Concurrency Control Technique

24 June 2022 9:34



Time Stamping Protocol

24 June 2022 9:40

Time stamp Ordering Protocol:-

→ Basic idea of Time stamping is to decide the order between the transactions before that enters into the system. so that in case of conflict during execution, we can resolve the conflict using ordering.

→ the reason we call time-stamp not stamp because, for stamping we the value of system clock. (as it will always be unique. can never repeat itself)

→ Two ideas of time stamping

→ **Time stamp with transaction**:- with each transaction t_i, we associate a time-stamp denoted by $TS(T_i)$,

→ It is the value of the system clock when a transaction enters into the system. so if a new transaction T_j enters after T_i then, $TS(T_i) < TS(T_j)$. always unique, will remain fixed through the execution.

→ also determine serializability order if $TS(T_i) < TS(T_j)$ then, sytem ensure that in the resultant C.S.S T_i will execute first before T_j.

→ **Time stamp with data item**:- for each data item Q, protocol maintains two time-stamps.

W-time stamp(Q): is the largest time-stamp of any transaction that executed write(Q) successfully.

R-time-stamp(Q): is the largest time-stamp of any transaction that executed read(Q) successfully.

T_i request for Read(Q):-

→ if $TS(T_i) < W\text{-}TS(Q)$, means T_i needs to read a value of Q that was already overwritten. Hence request must be rejected & T_i must rollback.

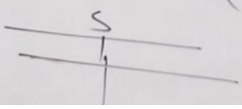
if $TS(T_i) \geq W\text{-}TS(Q)$, operation can be allowed and $R\text{-}TS(Q)$ will be $\max(RTS(Q), TS(T_i))$

T_i request for Write(Q):-

if $TS(T_i) < RTS(Q)$, means value of Q that T_i is producing was needed previously and the system assumed that the value would never be produced. Hence reject and Rollback

if $TS(T_i) < WTS(Q)$, T_i is attempting to write an obsolete value of Q. reject and Rollback

otherwise ok. $WTS(Q) = \max(WTS(Q), TS(T_i))$



Junior ko allowed hai senior ko nahi.

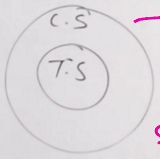
Properties of Time Stamping Protocol:

→ It ensure conflict serializability

→ It ensure view serializability

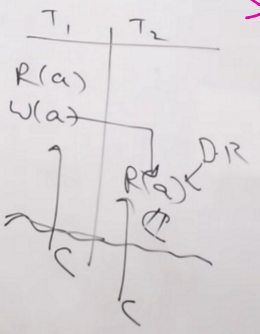
→ possibility of dirty read no restriction on commit, irrecoverable schedules and cascading rollbacks are possible.

→ Here either we allow or reject so no idea of deadlock.

→ 

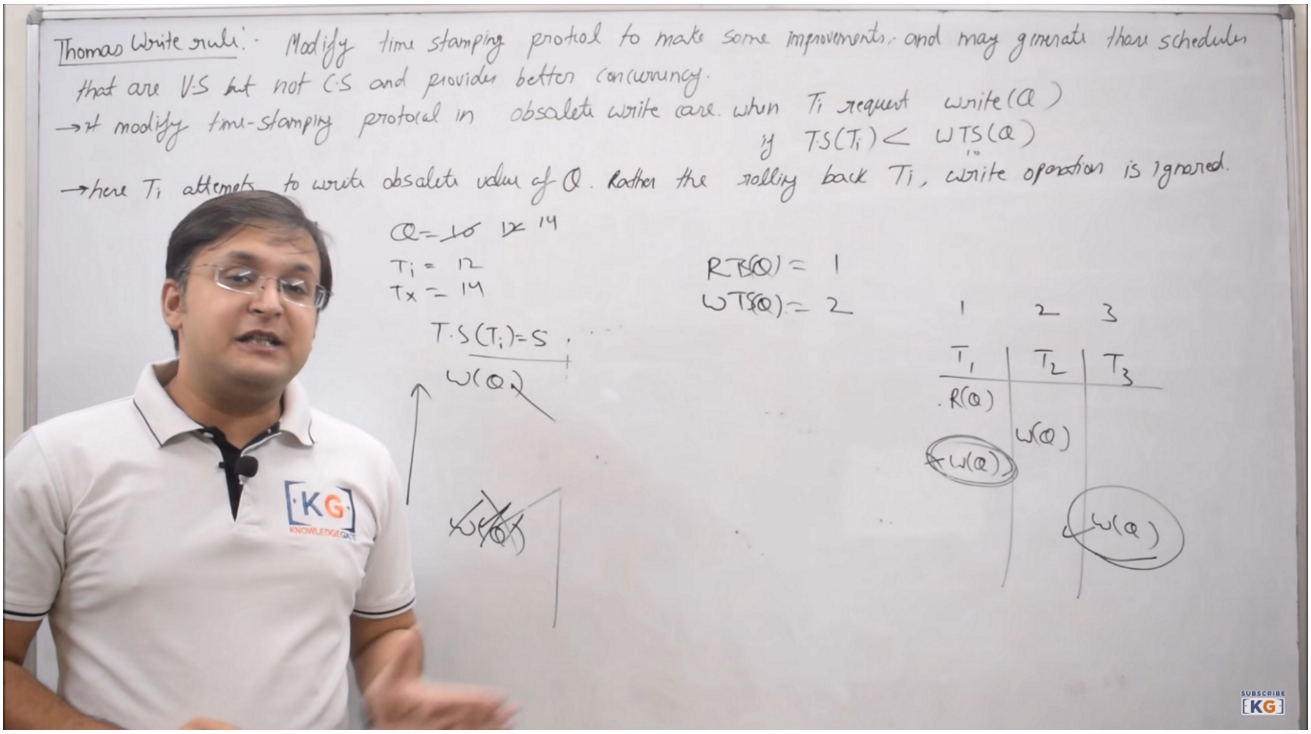
→ TS se banne vala schedule CS hoga

→ may suffer from starvation. relatively slow



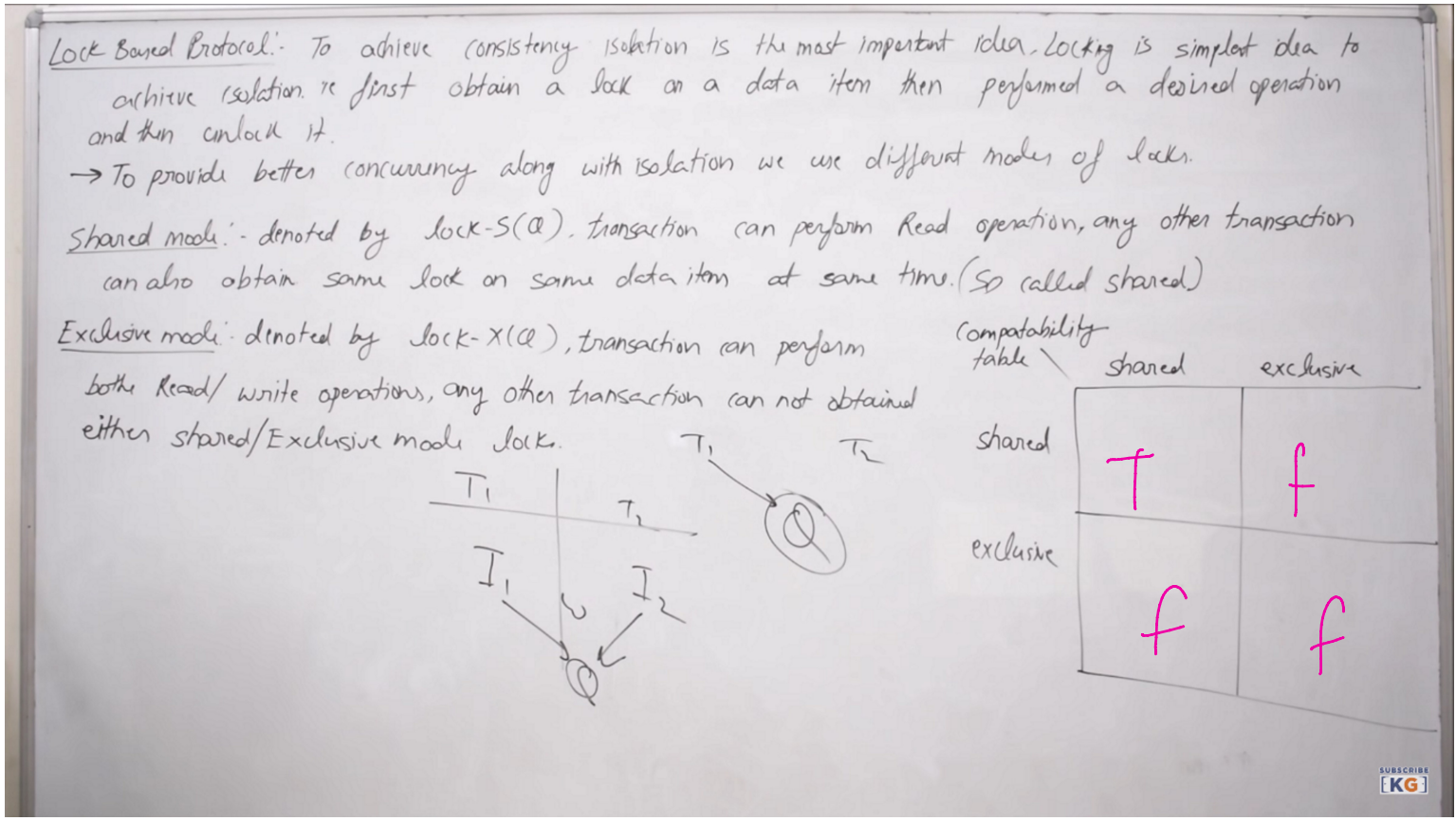
but han CS schedule TS se ban pai, this is not possible.

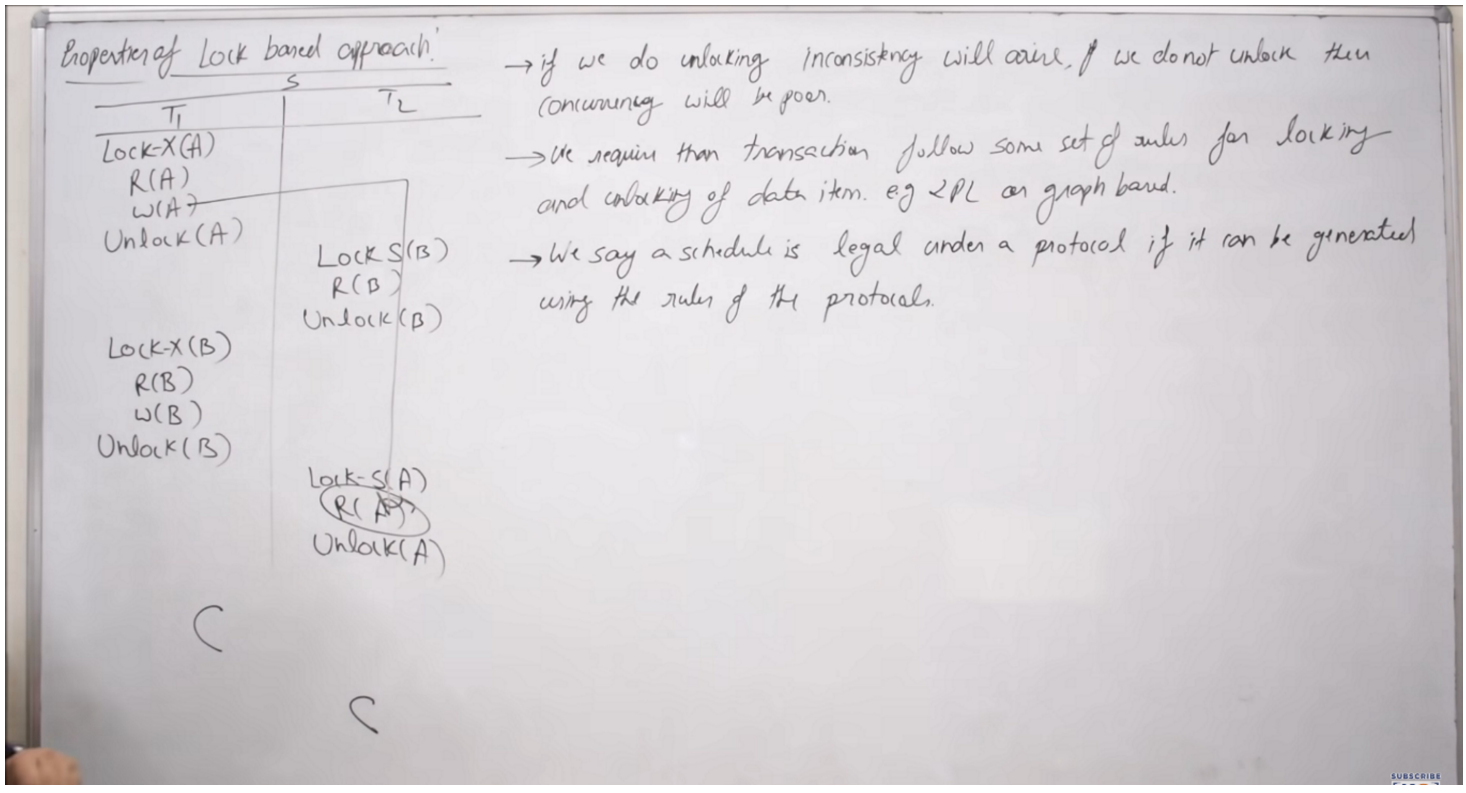
Thomas Write Rule



Lock Based Protocol

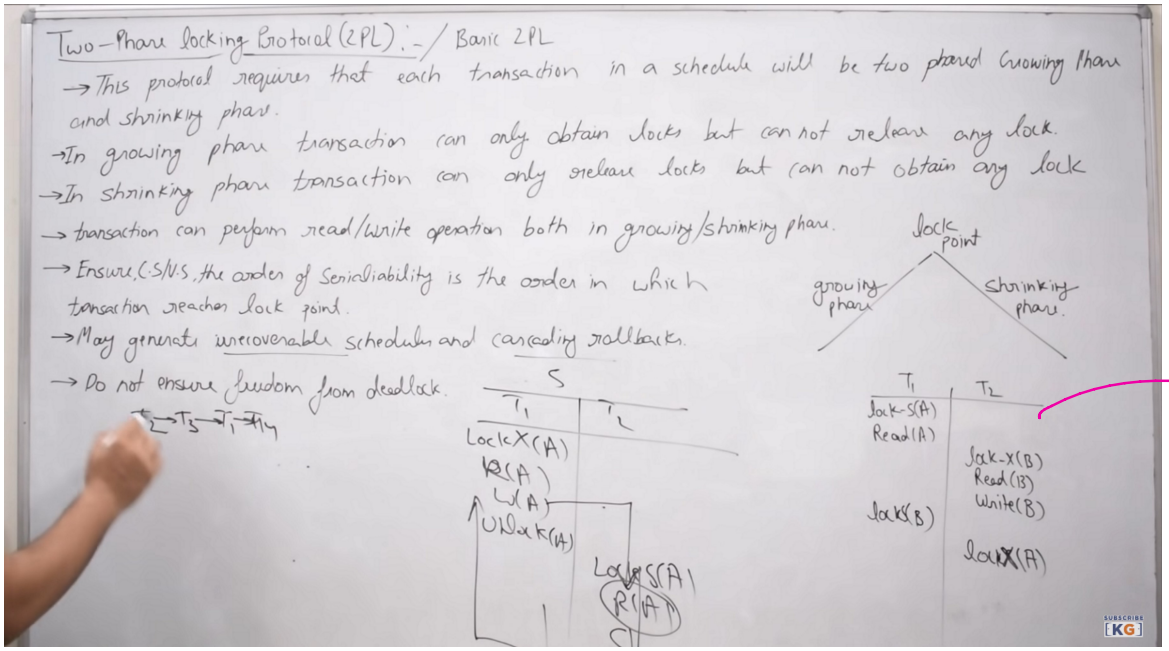
24 June 2022 12:06



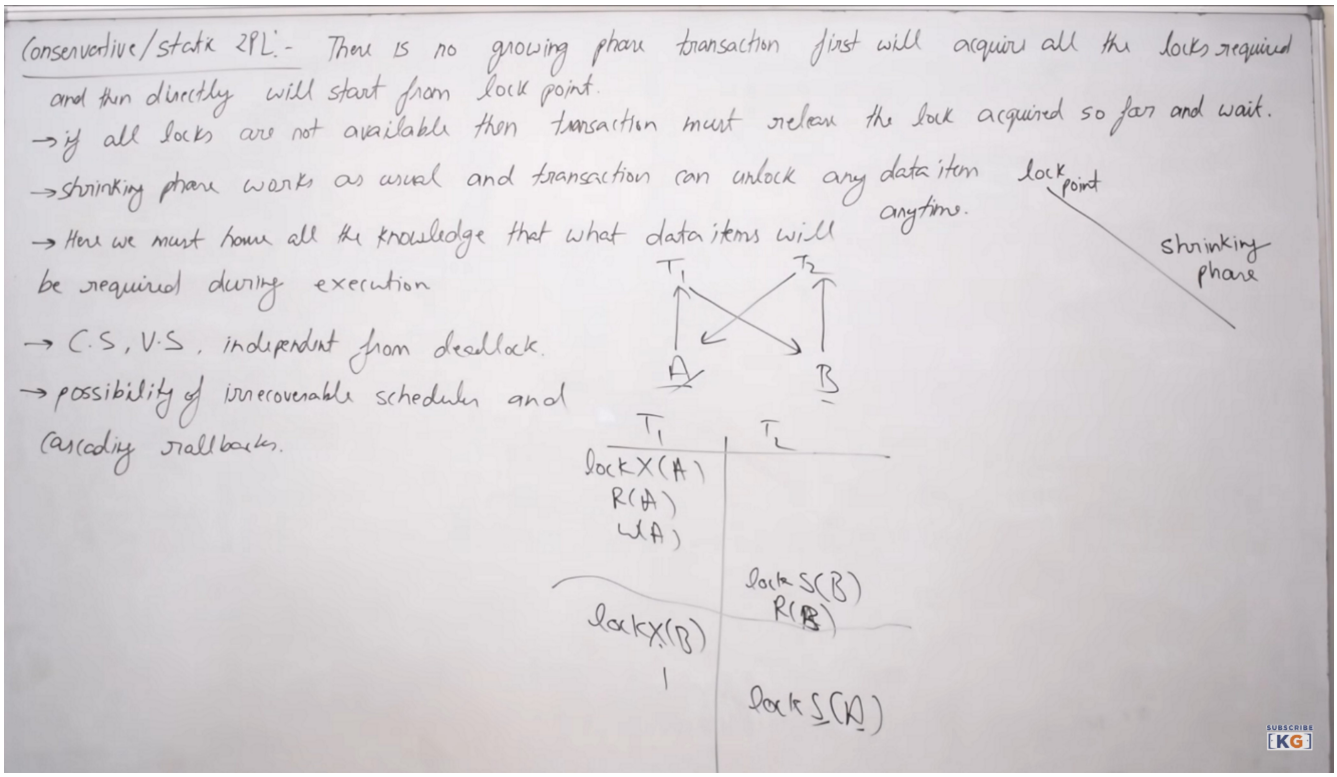


2 Phase Locking

24 June 2022 17:07



Conservative 2PL



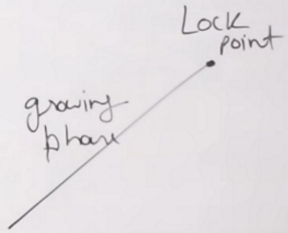
Rigorous 2PL

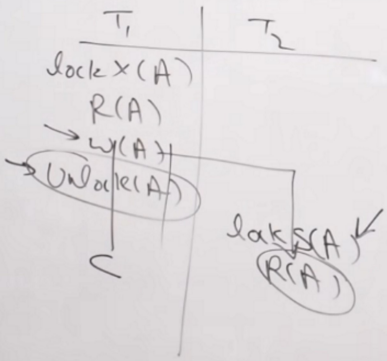
Rigorous 2PL:- it is an improvement over 2PL protocol where we try to ensure recoverability and cascadelessness.

→ Rigorous 2PL requires that all the locks must be held until transaction commits. i.e there is no shrinking phase in the life of a transaction.

→ will ensure C.S, V.S, recoverability, cascadelessness

→ suffer from deadlock and inefficiency.



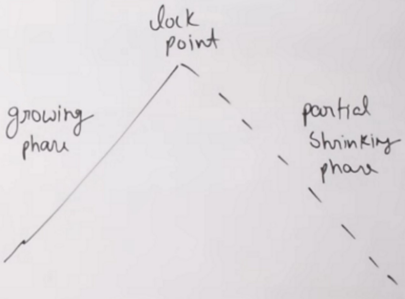


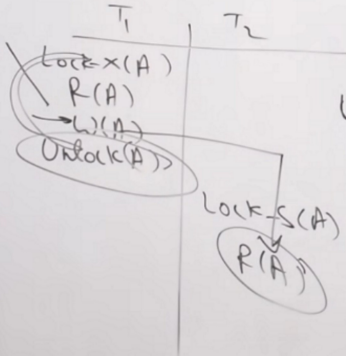
Strict 2PL

Strict 2PL:- It is an improvement over Rigorous 2PL.

→ In the shrinking phase unlocking of Exclusive lock are not allowed but Unlocking of shared locks can be done.

→ all the properties are same as that of rigorous 2PL, but it provide better concurrency





Graph Based Concurrency Control Protocol

24 June 2022 17:54

Graph based Protocol:-

→ If we wish to develop lock based protocol that are not based on 2PL, we need additional info that how each transaction will access the data.

→ There are various model that can give additional information each differing in the amount of info provided.

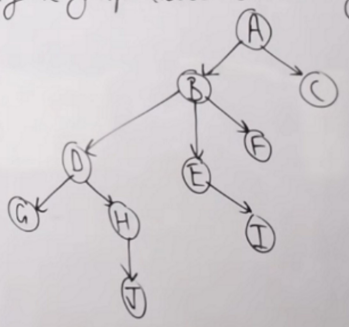
→ One idea is to have prior knowledge about the order in which the database items will be accessed.

→ We impose partial ordering → on set all data items $D = \{d_1, d_2, \ldots, d_n\}$. If $d_i \rightarrow d_j$, then any transaction accessing both d_i and d_j, must access d_i before d_j.

→ P.O may be because logical or physical organization or only because of concurrency control.

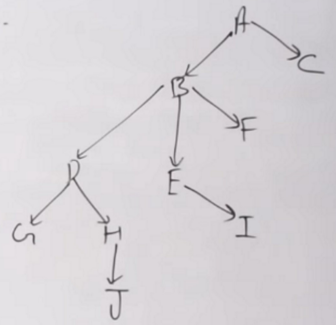
→ After P.O, set of all data items D will know be viewed as directed acyclic graph called database graph.

→ For sake of simplicity, we follow two restrictions.
- → will study graphs that are rooted trees.
- → will use only exclusive mode locks.



Tree Protocol:-

→ Each transaction T_i can lock a data item Q with following rules:-
- → First lock by T_i may be on any data item.
- → Subsequently, a data item Q can be locked by T_i only if the parent of Q is currently locked by T_i.
- → Data item may be unlocked at any time.
- → Data item Q that has been locked and unlocked by T_i can not subsequently be relocked by T_i.



T_1	T_2
lock X (B)	lock (A)
lock X (D)	lock (B)
lock X (E)	lock (C)
Unlock (B)	lock (B)
lock X (I)	Unlock (A)
Unlock (D)	lock (D)
Unlock (E)	Unlock (B)
Unlock (I)	lock (G)
	Unlock (D)
	Unlock (G)
	Unlock (C)

Properties of tree protocol:-

→ All schedules that are legal under the tree protocol are C.S & V.S.

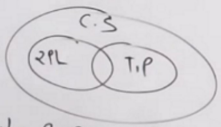
→ Tree Protocol ensure freedom from deadlock.

→ Tree Protocol do not ensure Recoverability and cascadelessness.

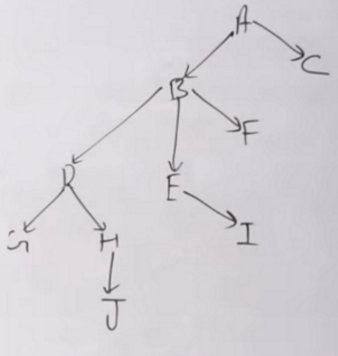
→ Early unlocking is possible. which leads shorted waiting time an increase concurrency.

→ A transaction may have to lock data items that it does not access. lead to overhead. waiting time. and decrease in concurrency.

→ transaction must know exactly what data item are to be accessed.

→ 

→ Recoverability & Cascadelessness can be provided by not unlocking before commit.



Deadlock Handling and Prevention

24 June 2022 18:27

Deadlock handling:-

→A system is in deadlock state if there exists a set of transaction such that every transaction in the set is waiting for another transaction in the set.

→If there exists a set of waiting transaction $T_0, T_1 \cdots T_{n-1}$, such that $T_0 \rightarrow T_1$, $T_1 \rightarrow T_2$; $\cdots\cdots T_{n-1} \rightarrow T_0$

So none transaction can progress in such situation.

→System must have a proper methods to deal with deadlock, otherwise

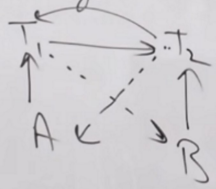
→ In real time system it may lead to life and money.

→will reduce resource utilization and increase inefficiency.

→ There are two principle for dealing with deadlock problem

→ Prevention:- which ensure that system will never enter a deadlock state.

→ Detection and Recovery: allow system to enter into deadlock, then try to recover.



DeadLock Prevention:-

→To ensure no hold & wait, each transaction locks all the data item before it begins execution eg C-2PL.

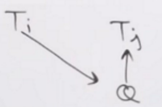
→To ensure no cyclic wait, impose an ordering of all data item, and requires that a transaction lock data item only in the sequence consistent with ordering. e.g tree protocol.

→It is difficult to predict what data items are required.

→data item utilization will be low.

→ordering of data item may be difficult, as time taking to follow.

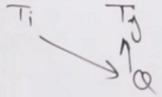
wait-die: (Non-Preemptive)



if $T.S(T_i) < T.S(T_j)$ then T_i must wait (T_i is older)

if $T.S(T_i) > T.S(T_j)$ then T_i rollback (die)

Wound-wait: (Preemptive)



if $T.S(T_i) > T.S(T_j)$ then T_i can wait (T_i is younger)

if $T.S(T_i) < T.S(T_j)$ the T_j rollback

Lock-timeouts:

Multiple Granularity

25 June 2022 7:53

Granularity : It is the size of data items allowed to lock.

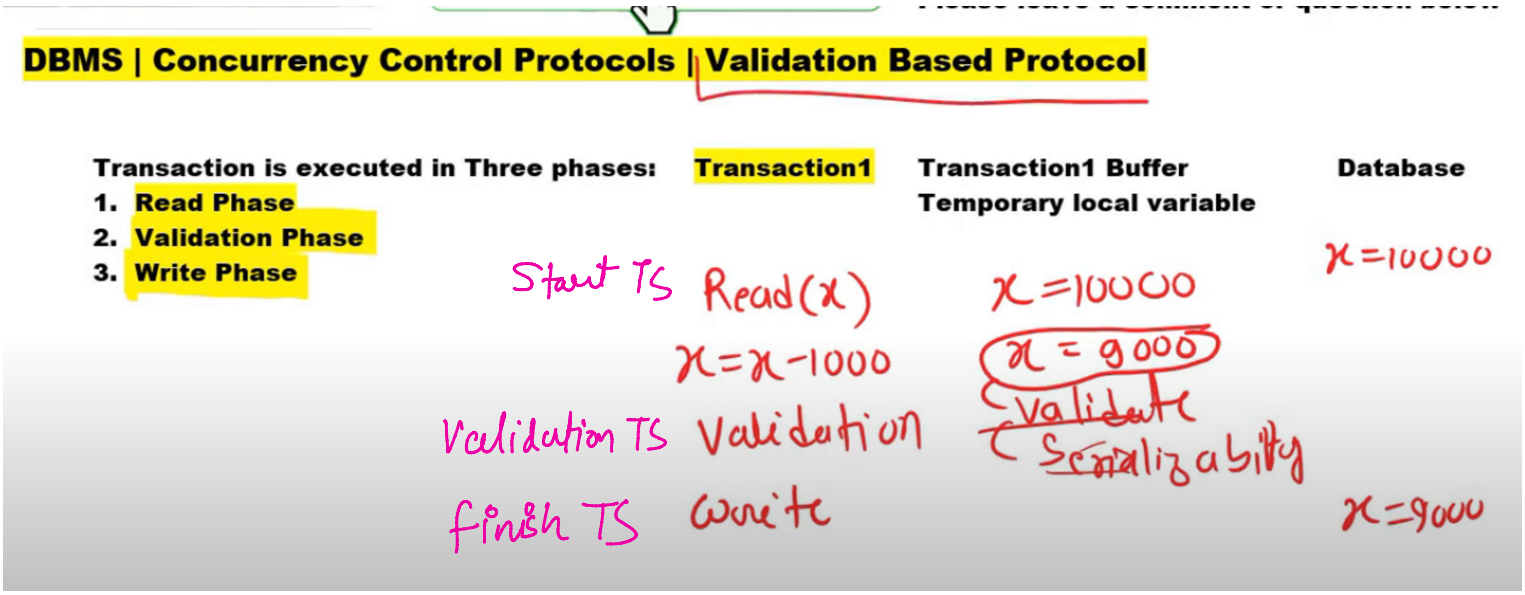
Multiple granularity :

Breaking the DB into multiple blocks hierarchically so that they can be locked efficiently.

Maintain concurrency
Reduces lock overhead
It maintains what to lock?
How to lock?
Decide which data item to lock or to unlock

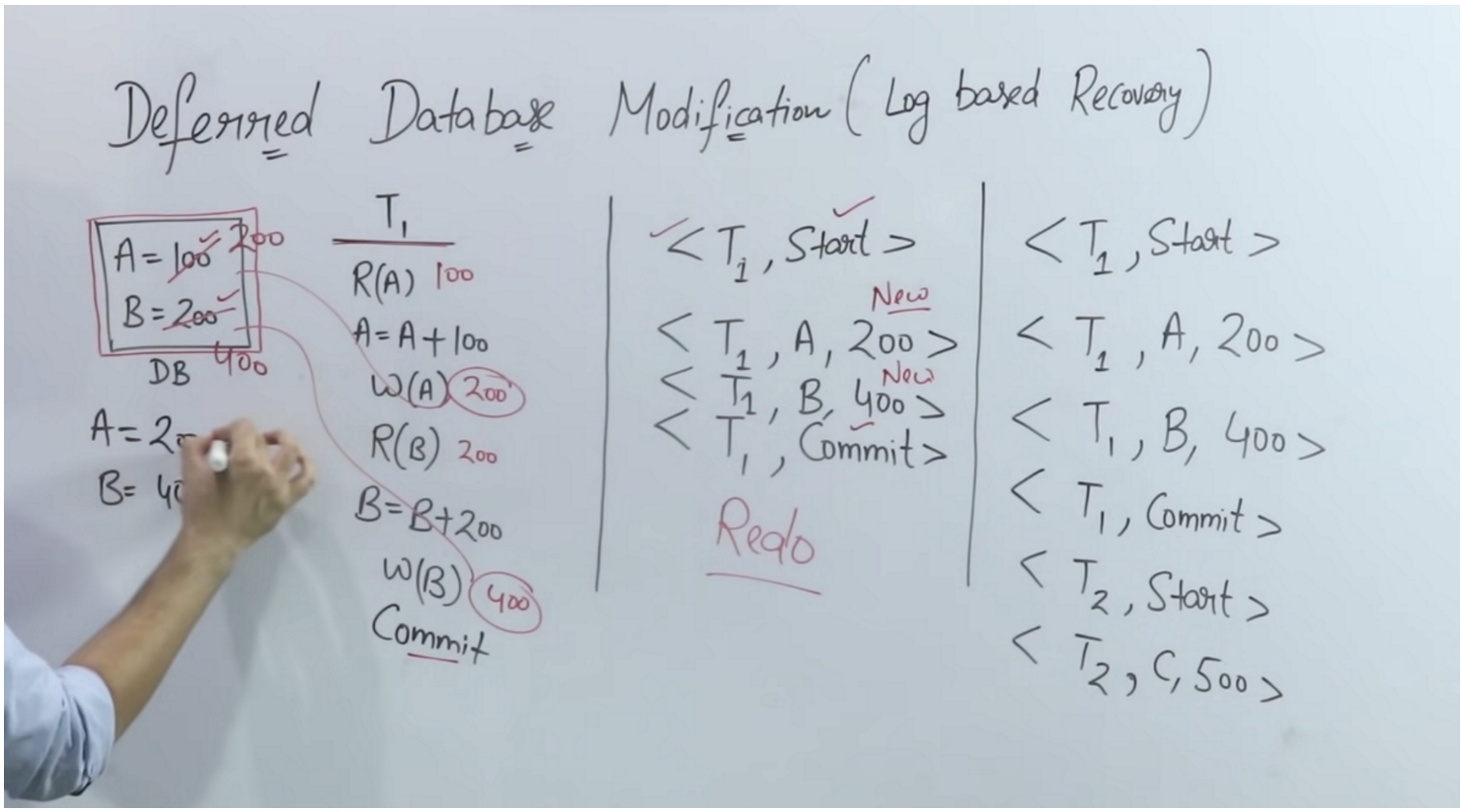
Validation based Protocol

25 June 2022 8:00



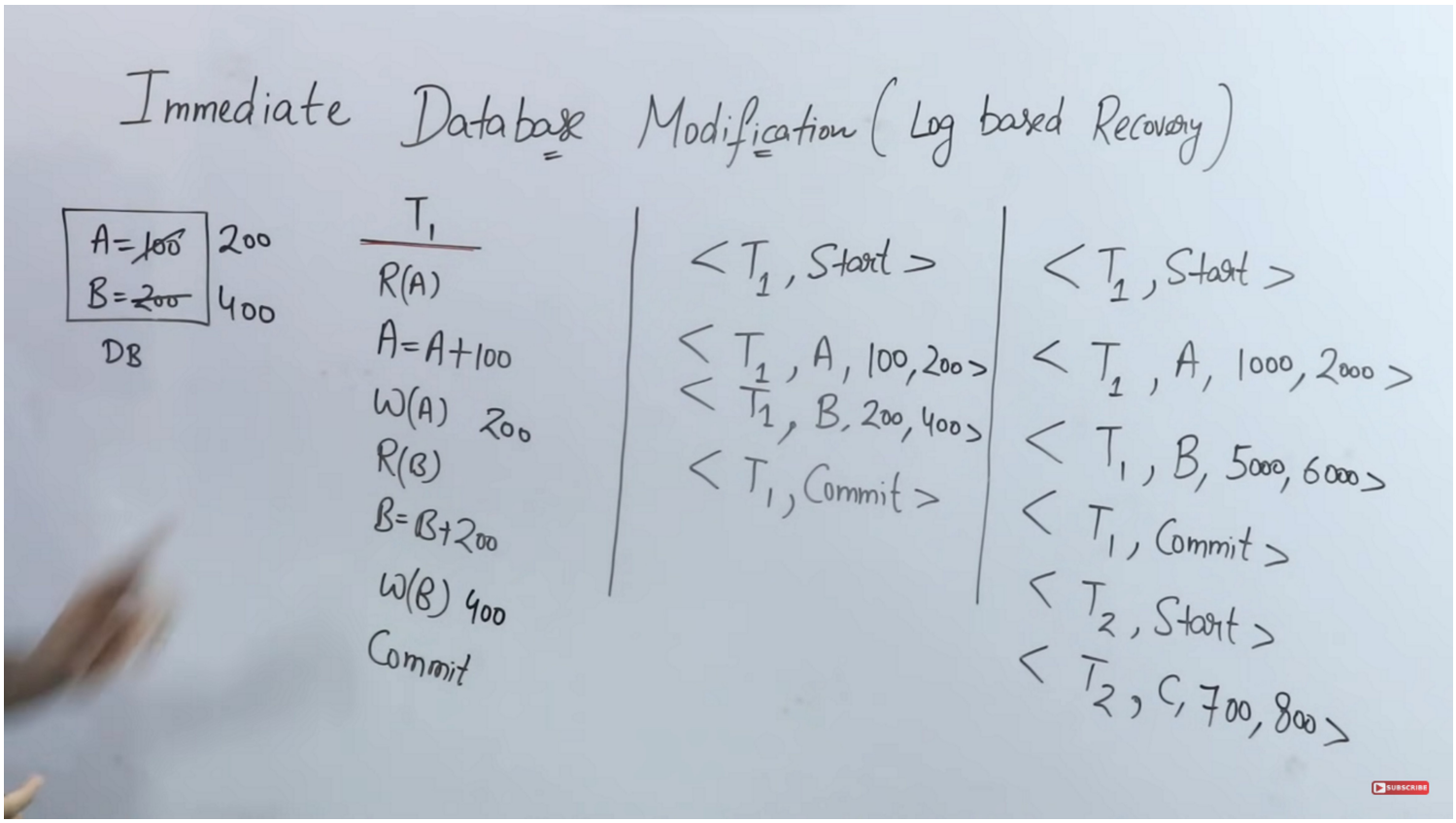
Deferred Database Modification (Log Based Recovery)

25 June 2022 8:05



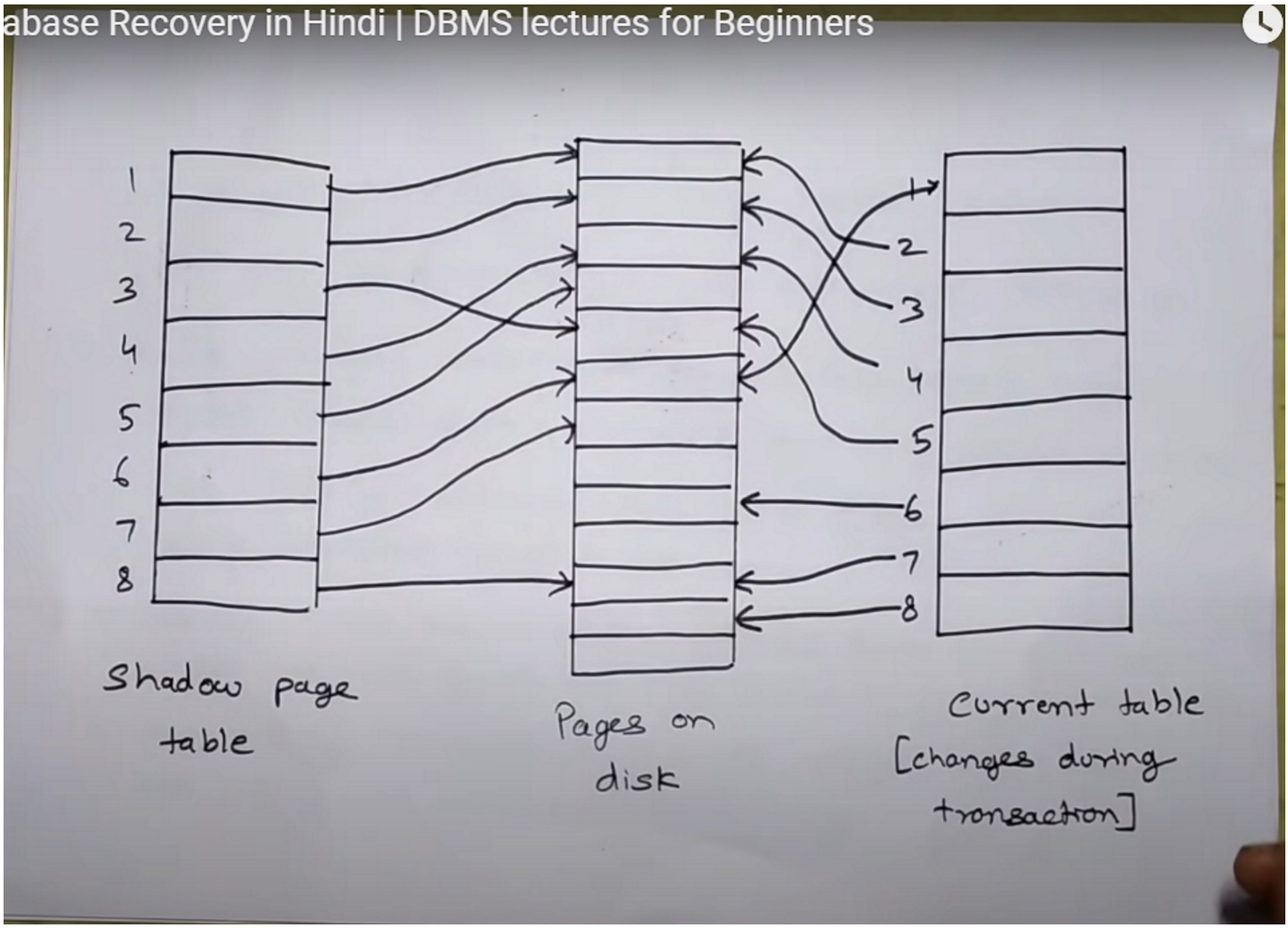
Immediate DB Modification

25 June 2022 8:07



Shadow Paging

25 June 2022 8:17



Shadow Copy Technique

25 June 2022 8:20