Other types of locking-Snapshot locking

```
Loop:
        Read C into C1
        Read D into D1      可能读了之后有人改了这个值,但你也不知道
        Read A into A1
        Read B into B1      读的时候不上锁
Compute new values based on A1 and B1
% Start taking locks on records that need modification.
        Let new value for C is C3 and for D is D3
              Xlock C
              Xlock D       只对修改的值上锁
              Read C into C2
              Read D into D2  ——> For optimistic lock: also slock A and slock B
   verify     if (C1 = = C2 & D1 = = D2)
        %     first writer commits
                   write C3 to C
                   write D3 to D
                   commit
                   unlock(C and D)
              else   % not first modifier
                   C1 = C2
                   D1 = D2
                   unlock(C and D)
                   goto Loop
   end
```

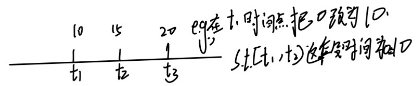
A more relaxed lock, Reading operations are done without any check or locks, just check the records that we need to write on, read records doesn't take any account into.
If someone change values of A and B, inconsistency may arise.

Snapshot Isolation method is used in Oracle but it will not guarantee Serializability. However, its transaction throughput is very high compared to two phase locking scheme

Cons: It does not guarantee consistency (serializability). It executes transactions in a serial order, final output can be different if run concurrently.

Pros:1. Higher throughput: there is no locks on the unchanged values, hence such locking mechanism has less # of locks, and all writing locks are taken right before the 'commit' take place, hence the duration is shorter. 2.It allows multiple reading transactions run concurrently.

Time Stamping



These are a special case of optimistic concurrency control. At commit, time stamps are examined. If time stamp is more recent than the transaction read time the transaction is aborted.

Time Domain Versioning

Data is never overwritten a new version is created on update.

<o,<V1, [t1, t2)>, <V2, [t2,t3)>, <V3,[t3,*)>

At the commit time, the system validates all the transaction's updates and writes updates to durable media. This model of computation unifies concurrency, recovery and time domain addressing.

We can track what transaction did before system down, which helps for crash recovery

Other types of locking

Optimistic Locking When conflicts are rare, transactions can execute operations without managing locks and without waiting for locks - higher throughput(认为系统中没有conflict, 没有trans对一个shared resource 进行修改)

- Use data without locks
- Before committing
  - each transaction verifies that no other transaction has modified the data (by taking appropriate locks)
  - Brief locks may be needed at the end - duration of locks are very short
- If any conflict found, the transaction repeats the attempt
- If no conflict, make changes and commit
- Loop内给只读的object也上锁(take read locks before commit)

Once the condition is true – it is effectively 2 phase locking but duration of locking is very short but can force many repeated attempts due to failure of the condition.

If conflict are rare, this technique is very efficient, and performance will be better compared with dead lock. However, if some values are changed by many transactions frequently, the conflict will arise, in which case many repeated attempts in the loop may need to be required, which is time consuming.

```
Read phase:        Read A into A1
                   Read B into B1
                   Read C into C1
Loop: Compute new values based on A1 and B1
% Start taking locks
Validation phase:  Slock A; Read A into A2
                   Slock B; Read B into B2
                   Xlock C; Read C into C2
        if (A1 == A2 & B1 == B2 & C1 == C2)
Write phase:            Write new value into C
                        commit
                        Unlock A, B and C
        else   % read data is changed
                        A1 = A2
                        B1 = B2
                        C1 = C2
                        unlock A ,B and C
                        goto Loop
   end
```

Two phase locking transaction

Integrity constraint A+B >= 0; A = 100; B = 100;

```
T1:                       T2:
Lock(X,A)                 Lock(S,A)
Lock(S,B)                 Lock(X,B)
Read A to A1;             Read A to A1;
Read B to B1;             Read B to B1;
A1 = A1 -200;             B1 = B1 -200;
if (A1+ B1 >= 0)          If (A1+ B1 >= 0)
  Write A1 to A             Write B1 to B
  Commit                    Commit
else abort                else abort
end                       end
Unlock (all locks)        Unlock (all locks)
```

Only one transaction can commit.

Snapshot Isolation Transaction

Integrity constraint A+B >= 0; A = 100; B = 100;

```
T1:                            T2:
Loop: Read A to A1;            Loop: Read A to A1;
      Read B to B1;                  Read B to B1;
      A3 = A1 -200;                  B3 = B1 -200;
      Lock(X, A)                     Lock(X, B)
      Read A to A2                   Read B to B2
      if (A1 != A2)                  if (B1 != B2)
          Unlock(A)                      Unlock(B)
          goto Loop                      goto Loop
      elseif (A3+ B1 >= 0)           elseif (A1+ B3 >= 0)
          Write A3 to A                  Write B3 to B
          Commit                         Commit
      else abort                     else abort
Unlock (all locks)             Unlock (all locks)
```

If T1 commit, then in T2 the A1+B1 =-200<0 false, T2 cannot commit. But in snapshot final output of database will not be correct result if both trans commit. (A=-100, B=-100 for snapshot)

Inconsistency will arise if use snapshot, because the 'reading operation' has no locks. Therefore, we have to commit trans one by one if snapshot were used.

Other types of locks

Phantoms: They commonly occur when a transaction lock records/table but another transaction adds new records in that table

So reading the same table twice by one transaction presents two different sets of records as the other has inserted some in the meantime

Solution: 1.Predicate locks solve this problem. Rather than locking records, lock based on condition

Select * from Employee where salary > 80000

Another transaction will be able to insert new records in Employee table as long as the salary is not within this predicate range (only lock on specified range of records in table, other transaction cannot change or update the value within this range)

2.Page Locks can solve this problem too. In this case, the storage should be organized based on values of attributes.(要在属性值（工资）被排好序的前提下，不排序不能用)

When teacher wants to send emails to all students who are in the enroll list, when she is going to compose the email and send to the list, some more student just finish the enrollment.(In this case, I miss these new student if the system weren't used phantom lock, and I have to manually check all list to find newly added students.) If I use phantom lock, it can prevent any other trans from inserting records. When we working on a certain subset of table, if the set changes or adding new records into it, then it is not a desirable condition of our database.

Time Stamping

T1	T2
select average (salary) from employee	update employee set salary = salary*1.1 where salary < $40000

If transaction T1 commences first and holds a read lock on a employee record with salary < $40000, T2 will be delayed until T1 finishes. But with time stamps T2 does not have to wait for T1 to finish!

- Timestamp ordering assigns orders for transactions based on time of commencement
- Locking has some sort of order which is decided at object access time
- When there are many updates, two-phase locking is good as it has less aborts
- Timestamp-based methods abort immediately which may be good sometimes
- If there aren't many updates an optimistic approach is better

Hence, **there is no winner** for all DBMSs for all types of data/queries.

Q What is the difference between a classical Optimistic Concurrency control mechanism versus a Snapshot Isolation-based one. What is the implication of this difference? Briefly explain.

Ans:OCC:In validation phase, OCC add read locks (Slock) before commit.Pros: If conflicts are rare, this technique has higher throughput because it has short duration of locking only being taken before commit for some quick checks. Cons: In a high contention system, the system may roll back frequently and many repeated attempts in the loop may be required because all work may be discarded if inconsistency is detected at the end. Snapshot Isolation does not have any read locks and just check the records for writing operation, which can be seen as a more relaxed locking and tends to perform better in read-heavy systems

Implication: Snapshot isolation transaction should be executed in a serial way, it cannot guarantee consistency when transactions running concurrently, as one transaction may change the value that has been read by another transaction during their execution, so such method is unsuitable areas like banking systems where constraints involve multiple data points. In contrast, OCC is prone to frequent rollbacks, can be configured to provide stricter consistency at the cost of efficiency in high concurrency cases.

Q What are the benefits of using snapshot isolation, compared to two-phase well-formed isolation? Also, give an example of an application scenario where snapshot isolation can be useful over a two-phase well-formed isolation.

Ans: SI provides a high level of concurrency by allowing multiple transactions to read the same data without conflict. Only write-write conflicts result in transaction aborts, which is especially beneficial for applications with frequent reads and fewer writes. 2PWF restricts concurrency more due to the shared lock and exclusive locks held until commit, which can limit the number of simultaneous transactions.

In an e-commerce platform's product catalog, especially during off-peak seasons, Snapshot Isolation offers advantages because occasional updates by an administrator (e.g., changes to product descriptions or stock levels) have minimal impact on users' decisions. Multiple users can browse products simultaneously, while the administrator can modify product information (write operations) without considering any reading locks, avoiding delays for users and improving system efficiency. Under Two-Phase Well-Formed Isolation, all read operations require shared locks, and write operations need exclusive locks, increasing the chance of blocking. In read-heavy scenarios, this can lead to delays caused by write locks, resulting in degraded performance. Therefore, SI is more suitable for such read-dominant applications.

What kind of applications use relaxed isolation? - Have you experienced any inconsistency in reading values as users?

Web browsing: if one is using=>then the web is locked, others are not able browse, leading to bad experience.

Adding product to shopping coat, everybody can add this item into cart, but only one of them will be available. (We can use degree 1, they can read concurrently at any time.)

1.Although multiple users can add it into cart, but only one user can pay at the end. If others have bought that previously=> then the system informs you this item is not available, and shopping cart gets updated. 2. Even sometimes, we've paid for it already, after a while we receive email saying sorry we have to return money.

1,2 have inconsistency=>but when we actually want to write on it=> it shows us the value has been changed by somone.

Fault tolerance by voting: at least $\frac{n+1}{2}$ for 奇数 can work, $\frac{n}{2}+1$ for 偶数

- Failfast

0 devices are faulty, we have 10 working and we need at least 6 to agree
1 device is faulty, we have 9 working and we need at least 5 to agree
2 devices are faulty, we have 8 working and we need at least 5 to agree
3 devices are faulty, we have 7 working and we need at least 4 to agree
4 devices are faulty, we have 6 working and we need at least 4 to agree
5 devices are faulty, we have 5 working and we need at least 3 to agree
6 devices are faulty, we have 4 working and we need at least 3 to agree
7 devices are faulty, we have 3 working and we need at least 2 to agree
8 devices are faulty, we have 2 working and we need both to agree
9 devices are faulty, we have 1 working and we have to stop as nothing to compare!

- Failvote (majority vote 不会变)

10个nodes: 至少要6个能工作，挂1,2,3,4都可以, 挂5个时候,就剩5个<6，不能工作
Consider a system with modules each with MTTF of 10 years➔ 一个坏掉的概率是1/10
Failvoting with 2 devices: MTTF = 10/2 = 5 years (system fails with 1 device failure) 只要有一个坏了就不行了, 整个系统坏掉的概率是 $\binom{2}{1}$*(1/10)=1/5=1/MTTF. 整个系统的MTTF=5
Failvoting with 3 devices: MTTF = 10/3 for the first failure + 10/2 for 2nd failure = 8.3 years.

Lower availability for higher reliability (multiple modules agreeing on a value means that value is more likely to be accurate/reliable)

Supermodule:只要系统有一个存活就能工作

Fault tolerance by voting

Q: Which of the following system has higher reliability than the other? (i) System A with 4 devices on failvote (ii) System B with 5 devices on failvote.

For failvote： A: 4/2+1=3 有三个就行; For B: (5+1)/2=3 有三个就行
挂一个:A 剩3个,可以工作; B剩4个,可以工作; 挂2个:A剩2个<3,fail; B剩3个,可以工作. B更好

For failfast：

	A	B
挂1	剩3,(3+1)/2=2<3可以	剩4,(4/2)+1=3可以
挂2	剩2,(2)/2+1=2=2可以	剩3,(3+1)/2=2可以
挂3.	剩1,(1+1)/2=1 nothing to compare	剩2,(2/2)+1=2可以 （B强）

Q: A failfast system has 8 devices. Assume 5 out of the 8 devices are unavailable but there are still 2 agreeing devices. Can the system continue to operate in this scenario? Please explain your answer.

Ans: 8-5=3 available, in such case for failfast: if (3+1)/2=2 devices agree the system can work. So there are 2 agreeing devices the system can still work.

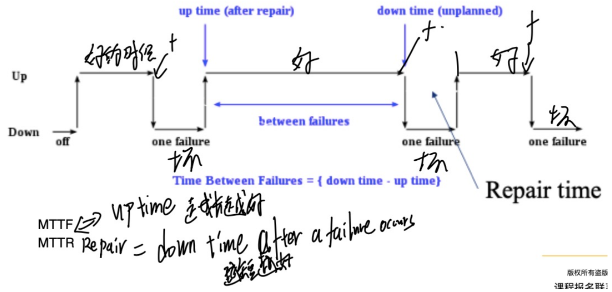
Q: Which system is expected to be available more (e.g., expected to function for longer) than the other system? (i) System A with 5 devices on failfast (ii) System B with 5 devices as a supermodule

Ans:

	A	B
挂1	剩4, 需要4/2+1=3个,可以	
挂2	剩3, 需要(3+1)/2=2个,可以	
挂3	剩2, 需要2/2+1=2个,可以	挂4个都能工作，B更强
挂3	剩2, 需要2/2+1=2个,可以	
挂4	down	

Module Availability

measures the ratio of service accomplishment to elapsed time.



$$\text{module availability} = \frac{mean\ time\ to\ failure}{mean\ time\ to\ failure + mean\ time\ to\ repair} = \frac{MTTF}{MTTF + MTTR}$$

Probability of a particular module is not available:

$$\frac{mean\ time\ to\ repair}{mean\ time\ to\ failure + mean\ time\ to\ repair} = \frac{MTTR}{MTTF + MTTR} = \frac{MTTR}{MTTF}$$

(MTTR << MTTF)

Fault tolerance by voting

Probability that (n-1) modules are unavailable, $P_{n-1} = \left(\frac{MTTR}{MTTF}\right)^{n-1}$

Probability that a particular i[th] module fails, $P_f = \left(\frac{1}{MTTF}\right)$

Probability that the system fails with a particular i[th] module failing last =

$$P_f * P_{n-1} = \left(\frac{1}{MTTF}\right)\left(\frac{MTTR}{MTTF}\right)^{n-1}$$

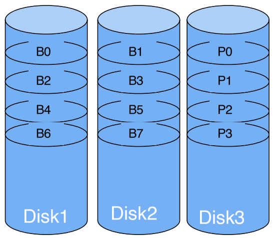
Probability that a supermodule fails due to any one of the n modules failing last, when other (n-1) modules are unavailable $= \left(\frac{n}{MTTF}\right)\left(\frac{MTTR}{MTTF}\right)^{n-1}$

Question 9: [3 Marks]

We have seen the supermodule concept in class with repairs. Explain in your own words even when we have disks where failures per disk may happen almost every other year, a supermodule using these disks may be fault tolerant at the level of many years or even hundreds of years. Briefly explain your rationale.

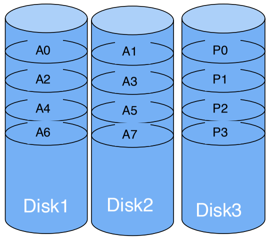
Ans: System can work if only one node can work, hence one disk failure would not impact the whole system, the reliability of the whole system will be improved greatly. So even if one disk failure we can use others to work and recovery the failed one.The concept of a supermodule achieves high fault tolerance by organizing multiple disks into a redundant structure, allowing it to tolerate individual disk failures without losing data. Even though each disk might fail every other year, a supermodule's redundancy means that it can withstand multiple independent failures by storing data across disks in a way that enables recovery from individual disk loss.

RAID



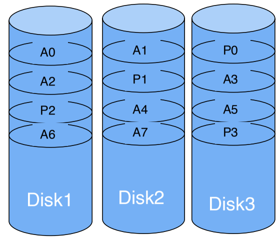
RAID 3 Byte level striping

1.Bi are bytes of data of file
2.Striping takes place at byte level so Rarely used
2.Provides higher transfer rate as in RAID 0
4.P0 is parity for bytes B0 and B1
5.P(system fail)=P(two of three fail)=$\binom{3}{2}$p*p=3p*p
6.MTTF=1/(3p*p)=$\frac{1}{3}MTTF^2$



RAID 4 Block level striping

1.Striping takes place at block level
2.Dedicated disk for parity blocks
3. MTTF=1/(3p*p)=$\frac{1}{3}MTTF^2$
Pros: Provides higher throughput.
Cons: 3和4都有,very slow writes. Disk3 has more writes than any of these two disks as Parity needs to be updated for every data write. So the whole write operation gets bottlenecked by the write speed of parity disk3



RAID 5 Block level striping

1.Parity blocks are also striped
2.Provides higher throughput (read from all disks in parallel)
3.slower writes but better than RAID 4 as Parity bits are distributed among all disks and the number of write operations on average equal among all 3 disks. (There is no single disk with high # of writes, write is better than 4,3)
4.MTTF=1/(3p*p)=$\frac{1}{3}MTTF^2$

RAID

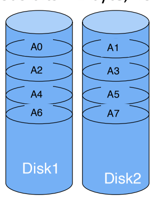
Q Which of the following RAID configurations has the lowest disk space utilization? (1) RAID 0 with 2 disks, (2) RAID 1 with 2 disks, (3) RAID 3 with 3 disks. Where does this lack of utilization of space go, i.e., where we can use such a configuration as it has some benefits gained due to the loss of space utilization?

Ans (2) has the lowest disk space utilization, because mirror method will store totally same data in two disk, hence only half of space is utilized. However, although (2) has low utilization, fault tolerance get improved a lot as its MTTF is higher. If one disk fail, we can still get whole same data from another disk. MTTF=$\frac{1}{p}*\frac{1}{p}=MTTF^2$, for (1) MTTF=$\frac{1}{2p}=\frac{1}{2}MTTF$, for (3) MTTF = $\frac{1}{\binom{3}{2}p^2}=\frac{1}{3}(MTTF)^2$

Q A researcher collected 10 Terabyte of data for her research. She needs to choose a RAID structure to store the data. She needs high reliability, high read throughput, and cheaper option for her work, but write throughput is not a concern for her. She is undecided between the following two options:
Option 1: RAID 1 with 2 disks Option 2: RAID 4 with 3 disks.
Assuming each disk has 10 Terabyte capacity and the same mean time to failure, explain the advantages and disadvantages of each option. If one of these options is clearly a more suitable choice over the other, please explain why. If not, please explain that as well.

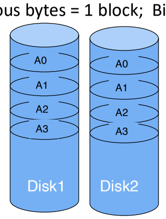
Ans For 1 Pros: 1. mirroring pattern is more reliable (MTTF)^2, avoiding data get lost. 2.higher read throughput, read can be done parallel on the two disks. 3.cost-effective as it only has two disks. Cons: lower space utilization (i.e. utilization is half), write cost is high, we have to write data on two disks. For 2: Read can be done parallelly so it has higher throughput. Higher space utilization Cons: Lower MTTF compared with (1), MTTF=1/(3p*p)=$\frac{1}{3}MTTF^2$. Higher cost, we need buy 3 disks.
Recommendation: (1) because it has higher reliability and cheaper satisfying the preference of this researcher.

RAID(Redundant Array of Independent Disks.) MTTF=1/p p是坏的概率
Different ways to combine multiple disks as a unit for **fault tolerance or performance improvement**
8 continuous bits = 1 byte; 4000/8000 continuous bytes = 1 block; Bit – b; byte – B; block – A



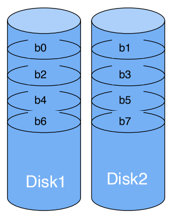
RAID0 (Block level Striping)

1.Disks can send data parallelly, throughput ~doubles (small overhead exists)
2.P(system fail)=P(one of disk fail) =p+p=2p
3.MTTF=1/(2p)=(1/2)*(1/p)=(1/2)*MTTF (减半)
Pros: Higher throughput at the cost of increased vulnerability to failures
Cons: Increased vulnerability to failures (一个坏了系统就坏了)



RAID1 mirroring (block)

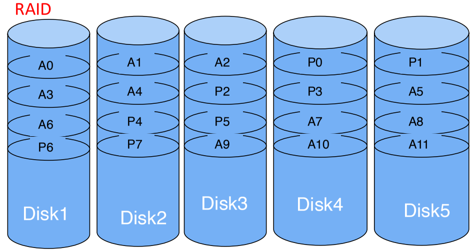
1.P(system fail)=P(both two disks fail) =p*p
2.MTTF=1/(p*p)= $MTTF^2$
Pros: Provides higher read throughput (reading can be done in parallel)
Cons: 1.lower write throughput (half of the total speed –i.e. single disk speed) , because anything written in disk 1 will also be written in disk2. 2.Half storage utilization



RAID2 bit level striping

Striping takes place at bit level
Provides higher transfer rate (double the single disk)
MTTF reduced by half as in RAID 0
rarely used (striping at bit level is difficult to main and read)

RAID



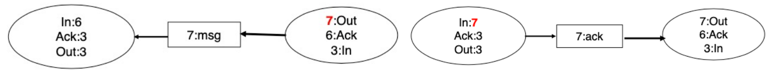
1.Similar to RAID 5 except two parity bocks used.
2.P(system fail)=P(three of five fail)=$\binom{5}{3}$*P*P*P
3.MTTF=1/(10p*p*p)=$\frac{1}{10}MTTF^3$ increase substantially.
4.P0 and P1 are parity blocks for blocks A0, A1 and A2. These are computed in such way that any two disk failures can be safe to recover the data.
5.All disks have similar # of writes and reading, so the throughput is higher.
Pros: Higher guarantee for fault tolerance, any two disks failures can be sate to recover the data.
Cons: Parity calculation and recovery process for RAID 6 is much more complex.

Q Which of the following RAID settings has the equal number of write operations on average among all the disks?
Ans RAID 4 with 3 disks w 个写操作, 每个data disk w/2,parity 需要写w, 任何disk写入都要写parity
RAID 4 with 5 disks: 共w, 每个data disk w/4,parity 要写w
RAID 3 with 3 disks:一样
RAID 1 with 3 disks: mirror 每个disk都相同
有parity的写操作都不均衡，parity都要多写

Q Given some data to be put on disks, which one of RAID configurations will have the best MTTF? Assume individual MTTF of the disks in this question are the same. Assum the propability of each of disks fail is p, so MTTF for this disk is 1/p.
(1) RAID 0 with 3 disks (any of the disk fail would cause system down) MTTF(坏一个就不行)=1/(3p)=(1/3)MTTF
(2) RAID 1 with 4 disks (for this case first disk is mirrored 3 times) MTTF=1/(p^4)=(MTTF)^4
(3) RAID 3 with 3 disks $\frac{1}{\binom{3}{2}p^2}=\frac{1}{3p^2}$=(MTTF)^2/3
(4) RAID 4 with 3 disks $\frac{1}{\binom{3}{2}p^2}=\frac{1}{3p^2}$=(MTTF)^2/3

Communication

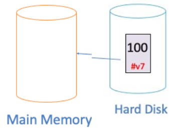
e.g., B wants to send message ID 7 to A. In: receive; Out: send; Ack: acknowledge; If B cannot receive from A, after waiting for a while, B will send message again. Keep doing…



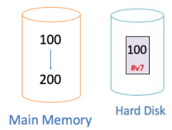
Atomicity: All changes to data are performed as if they are a single operation. That is, all the changes are performed, or none of them are. e.g., A transfer $100 to B

Duplex write: Each block of data is written in two places sequentially. If one of the writes fail, system can issue another write
Each block is associated with a version number. The block with the latest version number contains the most recent data.
While reading - we can determine error of a disk block by its CRC. It always guarantees at least one block has consistent data.

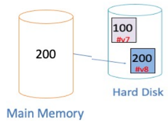
1.Read(from hard disk to main memory)



2.Modify in main memory

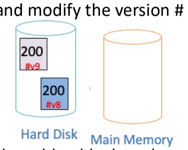


3.Write the change to the hard disk in a different block



We can compare #v8 and #v9 to check if the write is correct

4.Write the change to original block and modify the version #



If this step fails, we can have blue block as the latest version. We can use CRC to check if the write is correct

Data would be durable if it goes into disk, but actually why it cannot go into disk?

In fact, system avoid writing back to disk as long as possible. All sorts of reading and writing operations on disk are time-consuming and slow because of the performance issue. So the system tries to do such operation as less as possible unless the buffer is fulln (no capacity left). In that case, process or transaction needs a new page, and we really have to write a page back to disk to make. Room for new page.

Assuming 4 changes on the same block, all these changes generate a log (4 logs). However, page still in the buffer, when it needs to be written back to disk, four changes still many writing operations. Hence logs are written frequently, having an SSD for logs is better idea, speeding up the writing operation.

Let's assume that a bitmap index is in memory while the hard disk became unavailable. If the system is designed to still run* while the disk is being recovered, Can we still get answers for queries "How many people are in income level L1"?

Record Num	Name	State	Income_level
0	John	VIC	L1
1	Diana	NSW	L2
2	Xiaolu	WA	L1
3	Anil	VIC	L4
4	Peter	NSW	L3

Bitmap for income level	Income_level
L1	1 0 1 0 0
L2	0 1 0 0 0
L3	0 0 0 0 1
L4	0 0 0 1 0
L5	0 0 0 0 0

Yes, transaction does not have to access to the disk, as the page is store on the main memory.
Can B+tree achieve this?
No, because the records pointed by B+tree are on the disk. If the particular rows are on main memory, transaction will find them on memory, so that it will not go to the disk to read it.

RAID.

RAID 0 and RAID 2

1.Provides balanced I/O of disk drives (The same # of write and read operations happen at both disks. No bottleneck disk that we have to wait for disk write finish to move to the other one)
2.Provides higher throughput (~doubles)
3.Reliability performance: Any disk failure will be catastrophic 4.MTTF reduces by a factor of 2
5.System throughput: we can read from two disks in parallel, but Higher throughput (almost double) at the cost of increased vulnerability to failures.
6. Storage utilization: same. For RAID1 will be 50%.
7.Number of Disks needed: 2
8. Overhead difference: Striping in bit level cost more than striping in block level. In RAID 2, more dividing or partitioning overhead involved for writing operation. When we reading, we need to merge data from two disks more frequently to form meaningful data that can be displayed to users (much more overhead)

Which of the following RAID can safely recover data even for two disk failures? RAID 1 with 3 disks

Why communication reliability is important for databases?
The nodes of distributed database need to communicate with each other, also input in a DB does not only from users, but also from automated information or other system. If input comes, some acknowledgement or message passing involved, we need to check if the message or the input has been received correctly.
Additionally, many functionalities of DB need ensure the reliability during communication for consistency, because of huge number of operations (e.g., insertion, update or deletion).

Atomic Disk Write – logged write
Similar to duplex write, except one of the writes goes to a log.
Log – durable & can be read quickly.This method is very efficient if the changes to a block are small.
If just record what was the previous value, where's the changes in that entire block or when the changes happen=> ideal for using log (pros: save space, record small changes; writing operations are also quicker than writing on the entire block in the disk)
It is also possible that multiple users using the same block, all of the operation needed to be written in logs (number of logs increase.)

A company needs to store some sensitive medical records on disk. It is very critical that no error occurs, and no data is lost while storing the records. They can use multiple disks for the storage purpose if necessary. What strategies can they use to minimize the chance of any error while writing the data on disk? What strategies can they use to detect whether any error occurred or not? How can they still use the data if an error is detected?
Ans
We can use RAID1 with mirror pattern to store data, such configuration stores the same data on other disks, improving the reliability.
When writing data from main memory to disk, it is unsuitable to use logged write as multiple disks involved, leading to low efficiency in logged write. Instead, we can use **Duplex write** to store the data on disk. Each block of data is written in two places sequentially. If one of the writes fail, system can issue another write. We can use CRC to detect error. Verify whether a data in a block is correct, if not, the data with previous version number can be used. Because duplex write can duplicate data in two different places on disk, hence if errors were detected we can use the first modified version.

CRC – Cvclic Redundancv Check

Step

- 补最高次项个 0（n 个 0）
- 找多项式系数
- 做 XOR (从不为 0 的那一位开始)
- 剩下 n 位 （结果为 m）
- 将 m 补在 message 后，做 XOR
- 若结果为 0，则 correct

$X^4 + X^2 + X + 1$ original data 10100101/11100001

$x^4 \times 1 + x^3 \times 0 + x^2 \times 1 + x^1 \times 1 + x^0 \times 1$

10111

original data 1010010100000

4 补最高次项个0

做XOR

```
10100101 0000   同理
10111
-------------
00011101 0000   后面照抄
   10111
-------------
00001010 0000
    10111
-------------
00000001 1000
       10111
-------------
00000000 1111
```

结果数和最高次项数一样 4

Steal means if buffer pool is full, page 1 is chosen to evict from the buffer pool, page 1 hasn't committed but as there is no space in the buffer pool, page 1 is "forced" to be written in disk, and that caused problems, if the original transaction using page 1 is rolled back or aborted, since page 1 has already written to disk, the system has to managed to rollback the changes done to page 1.

Forcing: to write back to disk; No force: No write on disk just put them in buffer as long as possible; Steal: Take one of the pages (steal it) putting on disk, making room for buffer.