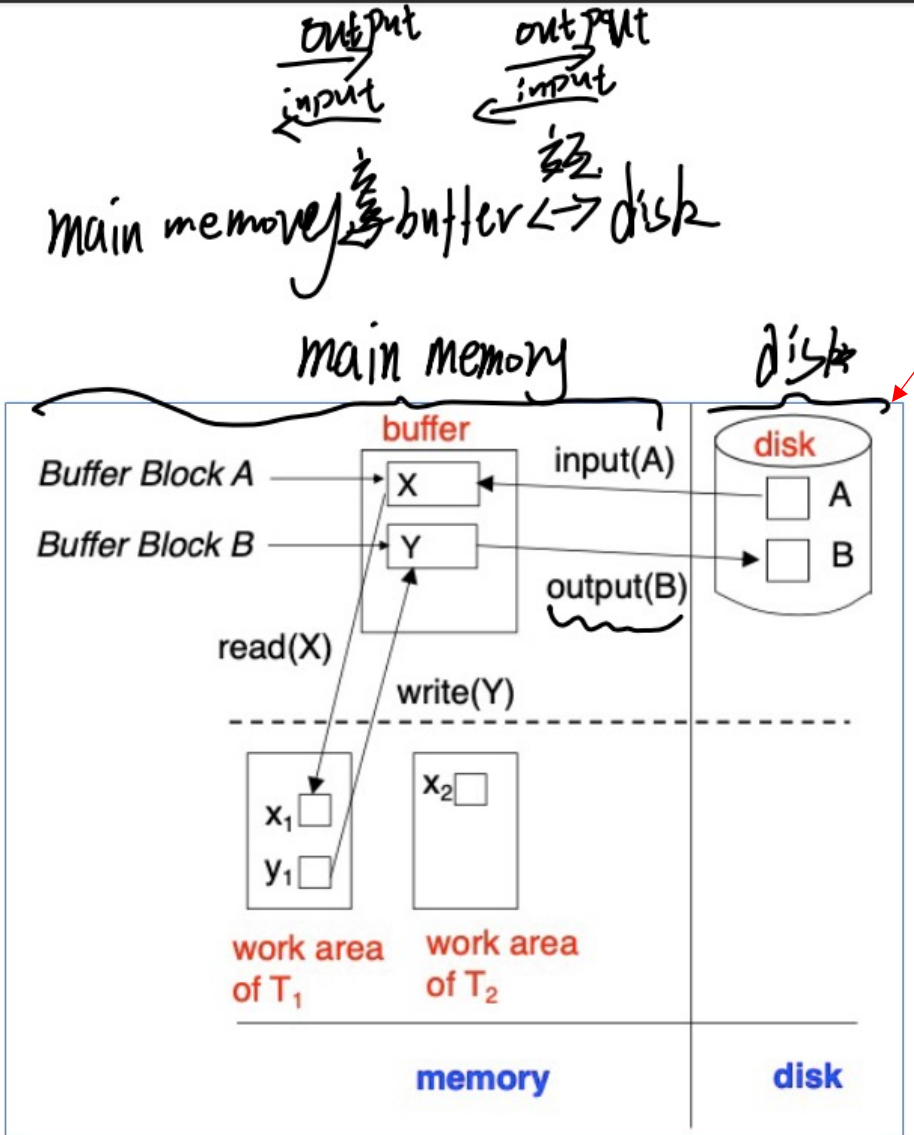
Crash Recovery

Recover from a failure either when a single-instance database crashes or all instances crash.

Crash recovery is the process by which the database is moved back to a consistent and usable state after a crash. This is done by **making the committed transactions durable** and **rolling back incomplete transactions**.



Data access

Two types of data blocks

- **Physical blocks** are those blocks residing on the disk.
- **Buffer blocks** are the blocks residing temporarily in main memory.

Two operations between disk and main memory

- **Input(A)** transfers the physical block A to main memory.
- **Output(B)** transfers the buffer block B to the disk and replaces the data there.

Logging

Definition - Record REDO (new value) and UNDO (old value) information, for every update, in a log.

Log is written sequentially (Log: An ordered list of REDO/UNDO actions)

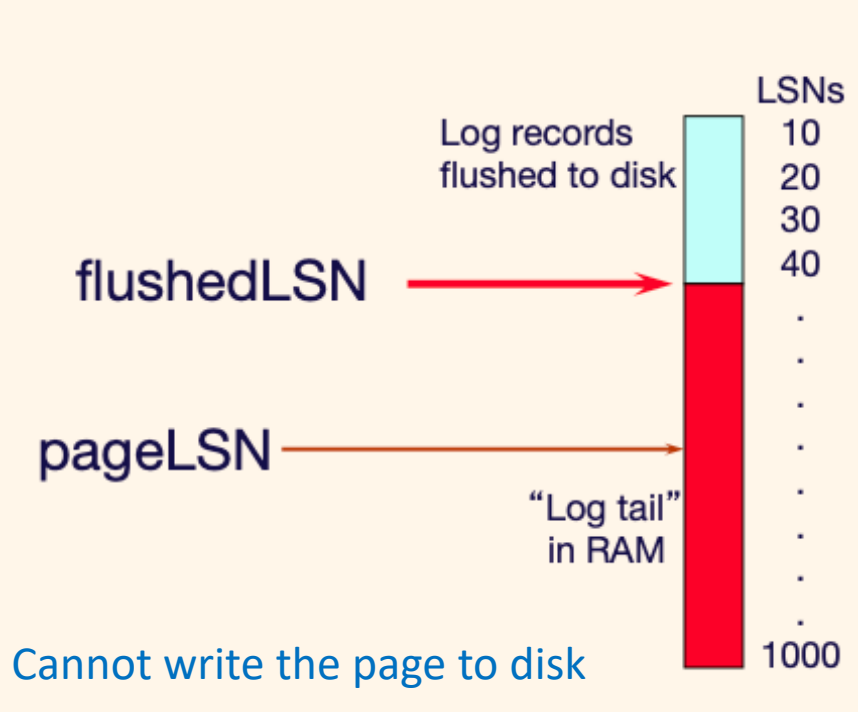
Each log takes very little space, so multiple updates fit in a single log page.

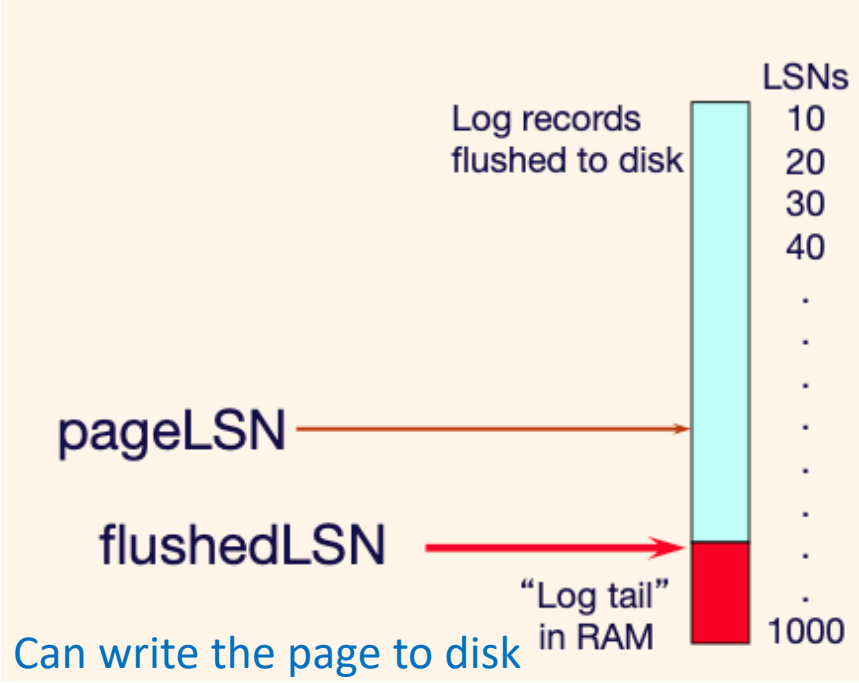
Log record contains: <XID, pageID, offset, length, old data, new data> and some other additional control info

Write-Ahead Logging (WAL)

Must force the log record (has both old and new values) for an update before the corresponding data page gets to disk
Must write all log records to disk for a transaction before it commits
不管干没干，都先记录到log上

- LSN (Log Sequence Nmber) – LSNs always increase
- pageLSN - The LSN of the most recent log record for an update to that page. (每个page的最近一次更新的LSN)
- flushedLSN - max LSN flushed so far (flush to disk) (目前最大的LSN需要上disk)
- WAL - Before a page is written to disk make sure **pageLSN <= flushedLSN (保证log去了disk=write ahead)(因为log先到disk上，有的仅是更新行为不是commit)**





Transaction Table（end了就不写了）

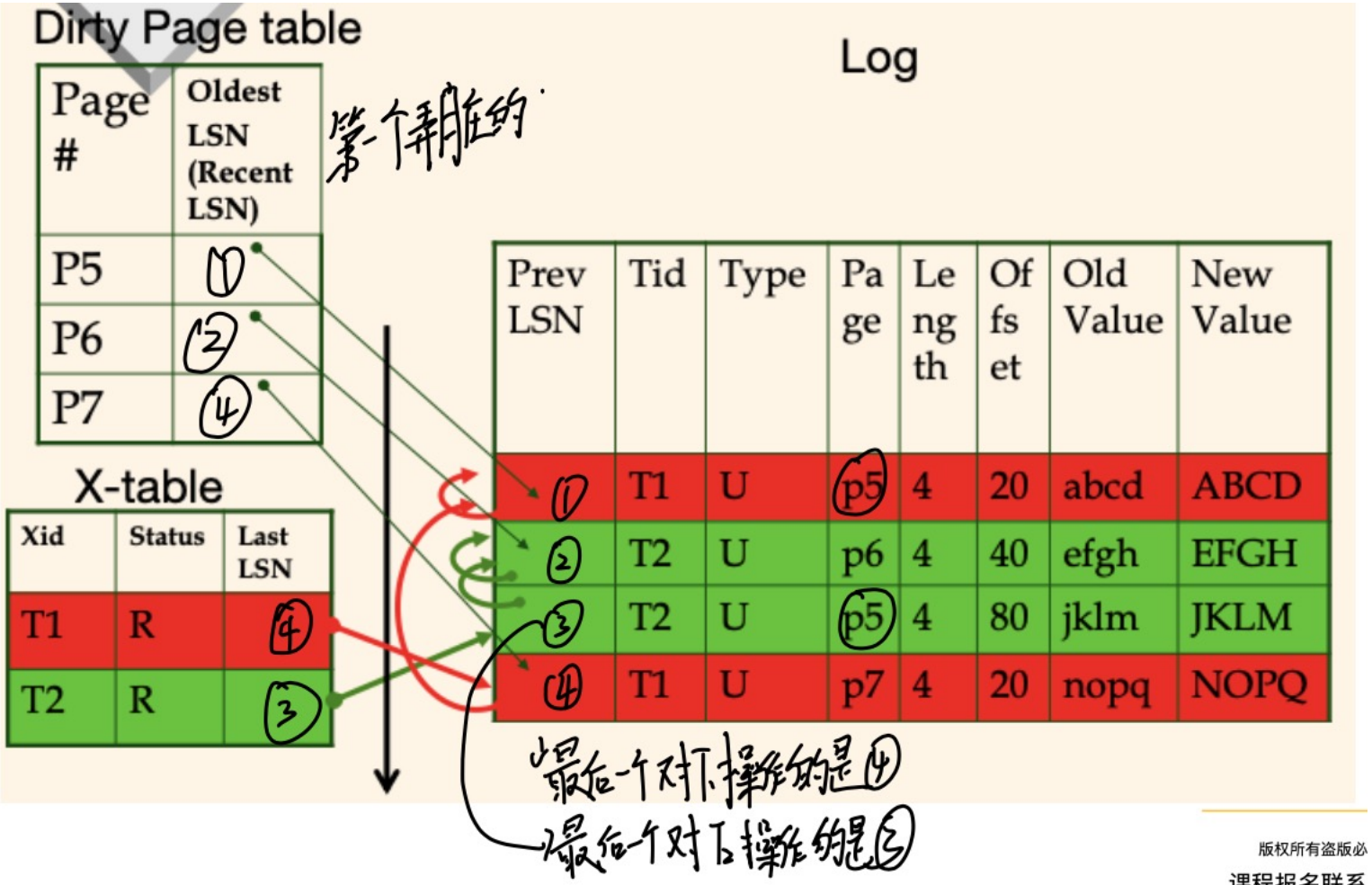
One entry per active transaction

Contains transactionID, status (running/committed/aborted), and lastLSN. (T_i最新的LSN)

Dirty Page Table（不管Transaction提交与否，全部写上去）

One entry per dirty page in buffer pool.

Contains recLSN: the LSN of the log record which **first** caused the page to be dirty (**第一个把 P_i 弄脏的**) since loaded into the buffer cache from the disk.



Checkpoint

Periodically, the DBMS creates a checkpoint, in order to **minimize the time taken to recover in the event of a system crash**. (保证之前的东西没问题，只从checkpoint之后看)

Write to log:

- Begin checkpoint record: Indicates when checkpoint began.
- End checkpoint record: Contains current Transaction table and dirty page table.

Transaction – Abort

- Get lastLSN of transaction from transaction table.
- Can follow chain of log records backward via the prevLSN field.
- Before starting UNDO, write a Compensation Log Record (CLR) （abort了就会有这个东西）
  - Continue logging while you Undo
  - Extra field: undonextLSN (point to the next LSN to undo)
  - CLR never undone
- At end of UNDO, write an “end” log record.

LSN	Log
400	begin checkpoint
405	end checkpoint
410	update: T1 writes P5
420	update: T2 writes P3
430	update: T1 writes P4
440	T1 abort 把T1作的恢复回去 先回左430 410
445	CLR: Undo T1 LSN 430
450	CLR: Undo T1 LSN 410
455	T1 End
460	update: T3 writes P5

Transaction – Commit （write to disk）

- Write commit record to log.
- Change transaction Status to “Commit” in Transaction table
- All log records up to transaction’s lastLSN are flushed.
  - Guarantees that flushedLSN >= lastLSN
  - Flushes are sequential, synchronous writes to disk (very fast writes to disk)
  - Many log records per log page
- Write “end” record to log.

Crash recovery

1. Analysis phase

1.1 Transaction table

1.2Dirty page table

2. Redo phase (top - bottom)

2.1 Start from the **smallest** number in Dirty page table – x （从最小的DPT开始）

2.2 For each CLR or update log record (exclude “commit”, “abort”, “end”, “checkpoint”), check **if the page is in Dirty page table** && **LSN >= x** && **pageLSN < LSN** (assume manually pageLSN if you don’t know and **reset pageLSN after redo**) （符合这些条件就redo）

2.3 Then, for these record, redo the action for LSN

3. Undo phase (bottom - top)

3.1 For the transactions which are in Transaction table, need to be undone(在 transaction中没做完的)

3.2 Find the **largest** LSN of these transactions and put it into ToUndo （从最大LSN开始undo）

3.3 Continuously find all transactions that need to be undone

A system uses logged writes to ensure consistency of disk writes. Given a data block where some modification is made, which one of the following options is true?

Logged write takes more time (i.e., less efficient) compared to duplex writes. （The method is very efficient if the changes to a block are small）

The modification needs to be logged first before the updated data block is written on disk (对)

Both the data block and the log about the modification made to that data block, need to be written twice in two different places （only duplex）

If the system fails to log about the modification made to that data block for some reason, still the data block can be written to disk. （不能）

We have a simplified log at the time of a system crash. Assume that there is no log record before the checkpoint. The format of a log record is (LSN, Operation Details).

(00, begin checkpoint)

(05, end checkpoint)

(10, T1 write page1)

(20, T2 write page2)

(30, T1 write page5)

(40, T2 commit)

(45, T2 end)

(50, T3 write page5)

(60, T1 abort)

(70, CLR undo T1 LSN 30)

(80, CLR undo T1 LSN 10)

(90, T1 end)

CRASH

The system recovery consists of three phases: analysis, redo and undo. Please answer each of the following questions.

A. What information will be in the dirty page table after the analysis phase (write as a list of the format (Page id, LSN))?

B. If the pageLSN of Page5 stored in the database is found as **30**, then what will be the order of the LSNs to be redone in the Redo phase? Assume that all the necessary pages are in the dirty page table, all LSNs in the log are greater than or equal to the corresponding page's recLSN, and all LSNs in the log are greater than the corresponding page's pageLSN (except for Page5).

C. What is the order of the LSNs to undo in the Undo phase?

(A) DPT

P-id	LSN
P_1	10
P_2	20
P_5	30

(B) X-act

T-id	Status	LastLSN
T_3	R	50

(B) redo page.LSN(P5) = 30

LSN 10 : 10 > pageLSN(P1), 10 > 10 redo LSN 10 and
set pageLSN(P1) = 10

(C) T3 undo 50

LSN 20 20 > pageLSN(P2), 20 > 10 redo LSN 20 and
set pageLSN(P2) = 20

LSN 30 30 = pageLSN(P5) = 30, not redo

LSN 50 50 > pageLSN(P5) = 30, 50 > 10 redo LSN 50
and set pageLSN(P5) = 50

LSN 70 70 > pageLSN(P5) = 50, 70 > 10 redo LSN 70
and set pageLSN(P5) = 70

LSN 80 : 80 > pageLSN(P1) = 10 80 > 10, redo LSN 80
and set pageLSN(P1) = 80

LSN 10 → 20 → 50 → 70 → 80

Distributed DB Two Phase Commit Protocol - Different from Two phase locking

- Goal: either all of the servers commit the transaction, or all of them abort the transaction **(Atomicity)**
- One of the servers becomes the coordinator who must ensures the same outcome (commit or abort) at all of the servers

Phase 1 (voting phase)

- coordinator asks all participants if they can commit
- each participant votes (yes or no)
- If it votes to commit, it cannot change its mind. In case it can crash, it must save updates in permanent storage.
- If participant requests to abort (一个人abort所有人abort), the coordinator informs all participants immediately

Phase 2 (completion phase)

- coordinator tells all participants to commit or abort=>> 告诉所有人commit or abort
- the participants carry out the joint decision

A local lock manager cannot release any locks until it knows that the transaction has been committed or aborted at all the servers involved in the transaction.

When/how will a node know when to release the locks of a transaction?

When global_commit sent, it means coordinator tells participates others have successfully finished their part of the transaction, if they receive that, then each of them can release their own locks for that transaction.

When one node fails, then when will other nodes to release their locks? When any of subtransactions abort, others will also abort as well. So in phase 1, the down node sending 'no' message at that time, T1 releases its lock in phase 1, others will know this in phase 2 and they will release their locks for T.

Distributed DBs

Two Phase Commit Protocol – Abort

Coordinator or participant can abort transaction
If a participant abort, it must inform coordinator
If a participant does not respond within a timeout period, coordinator will abort
If abort, coordinator asks all participants to rollback
If abort, abort logs are forced to disk at coordinator and all participants

Concurrency Control in Distributed System

- Problems of the example
- Servers independently acting would not work
- If transaction **T is before transaction U** in their conflicting access to objects at one of the servers
- Then: They **must be in that order at all of the servers** whose objects are accessed in a conflicting manner by both T and U
- The Coordinator should assure this

Locking Based System

- A local lock manager cannot release any locks until it knows that the transaction has been committed or aborted at all the servers involved in the transaction. （所有server 需要保持一致）
- The objects remain locked and are unavailable for other transactions during the commit protocol.
- An aborted transaction releases its locks after phase 1 of the protocol. 先abort再放锁

Concurrency Control Review

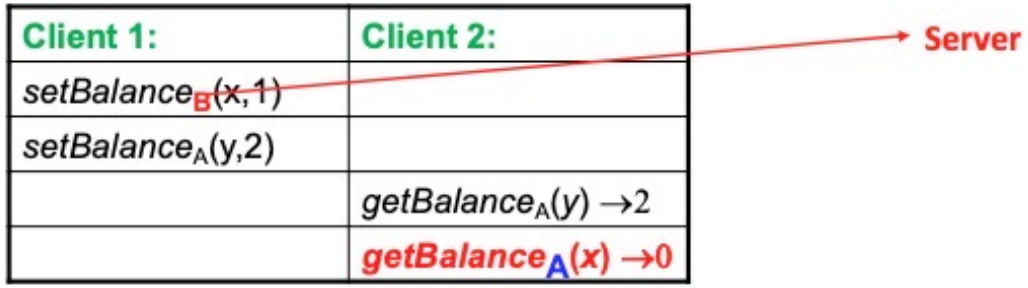
Timestamp ordering concurrency control

- The coordinator accessed by a transaction issues a globally unique timestamp
- The timestamp is passed with each object access
- The servers are jointly responsible for ensuring serial equivalence, that is if T access an object before U, then T is before U at all objects

Optimistic concurrency control (用 optimistic concurrency control时, validation 发生在phase1)

For distributed transactions to work:

- 1) Validation takes place in phase 1 of 2PC protocol at each server
- 2) Transactions use a globally unique order for validation



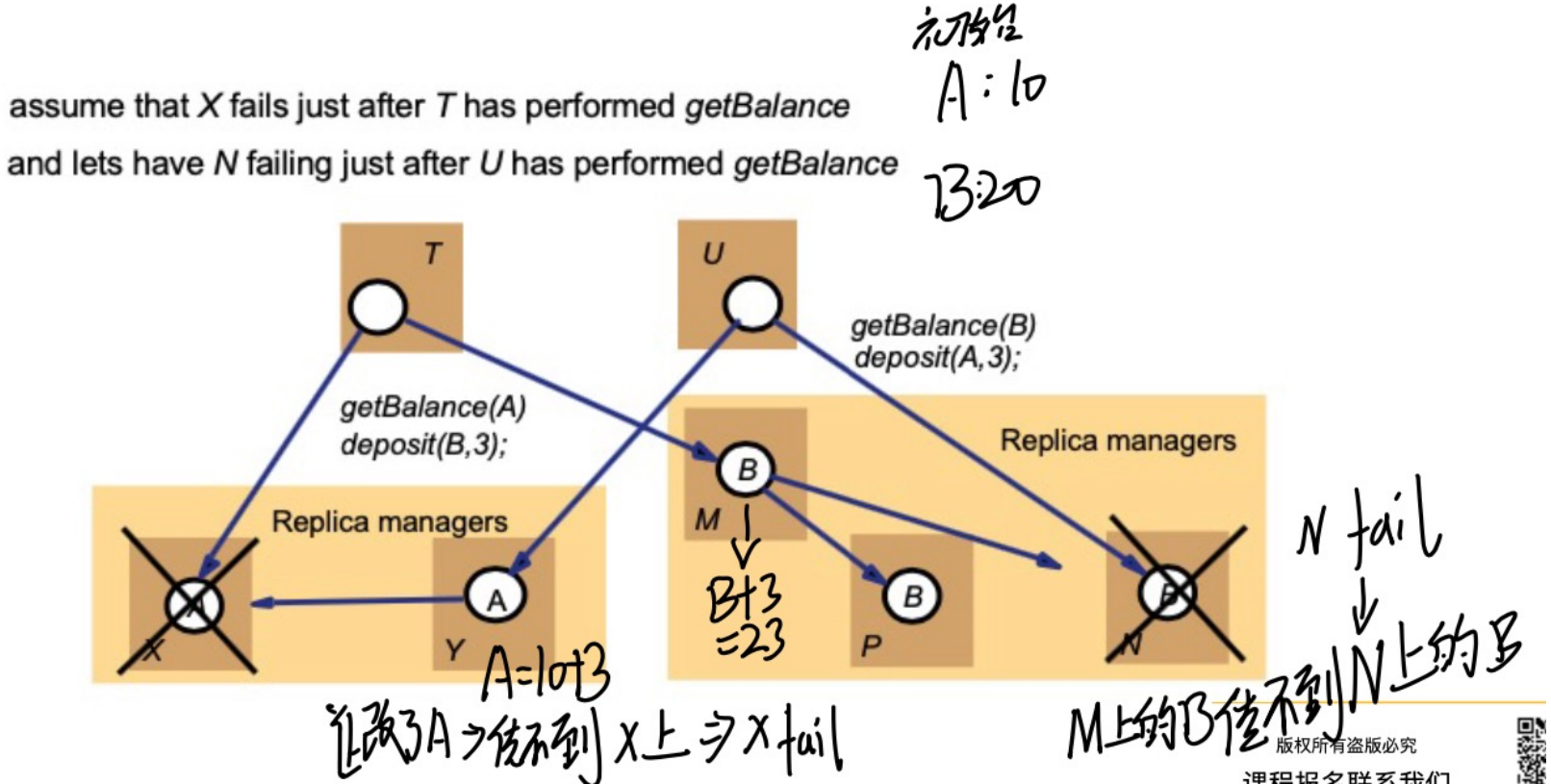
One-copy Serializability

The effect of transactions on replicated objects should be the same as if they had been performed one at a time on a single set of objects (A,B是server都存着X, 这个X要一致)

Distributed DBs

Read one / write all(不现实，没有考虑到节点unavailable的情况) :The read one/write all scheme is not realistic because it cannot be carried out if some of the servers are unavailable which beats the purpose in many cases

- Definition - one server is required for a read request and all servers for a write request
- Each read operation is performed by a single server, which sets a read lock
- Every write operation must be performed at all servers, each of which applies a write lock
- Any pair of write operations will require conflicting locks at all of the servers
- A read operation and a write operation will require conflicting locks on one server

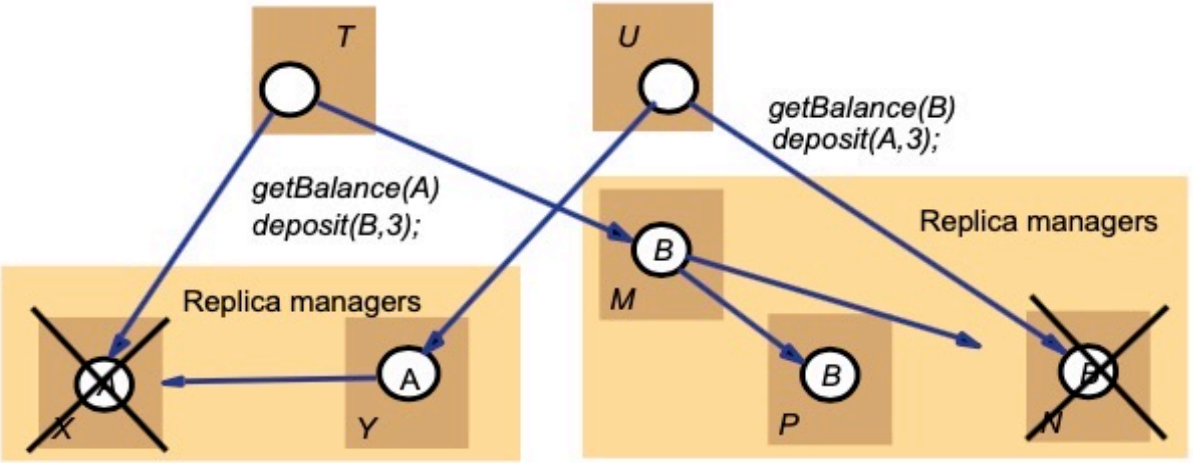


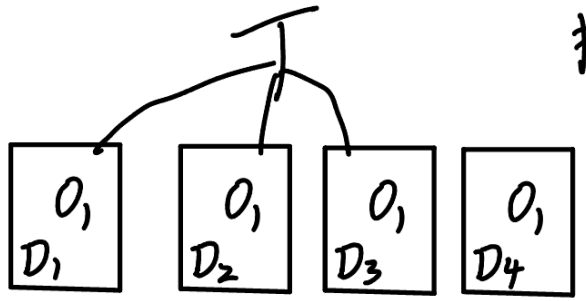
Distributed DBs

Available copies replication: commit 前检查是否所有的node都available，if有unavailable then abort

1.The available copies replication scheme is designed to allow some servers to be temporarily unavailable

2.Before a transaction commits, it checks for failures and recoveries of the RMs it has contacted, the set should not change during execution e.g., X fails before T's deposit. T would check if X is still available. If X is unavailable, then T will abort.





提交前要检查与T有关联的node的状态，必须几个环境状态与开始相同，否则abort)

before commit $\{D_1, D_2, D_3\}$ ⇒ $\{D_3, D_4$ unavailable $\{D_1, D_2\}$ available

after commit ⇒ $\{D_1, D_2\}$, D_4 available, D_3 unavailable

Although D_4 is available now, D_4 has no contact with T ⇒ 不影响 execution. (still the same set compare with starting)

Before a transaction commits, it checks for failed and available servers it has contacted, the set should not change during execution： （不管读还是写 只要连接过程中与它联系的set的状态不变就行）E.g., *T* would check if X is still available among others. We said X fails before *T*'s *deposit*, in which case, T would have to abort. Thus no harm can come from this execution now.

Exercise – 2023 Win

Question 8 3 pts

In a distributed database transaction system, a transaction T only needs to read an object o1. This object o1 is replicated in the nodes D1, D2, and D3 of this distributed system. There is another node D4 in this distributed system, but D4 does not store any copies of object o1. Answer for the following scenarios:

Scenario 1: When T1 started its execution, all nodes (that is, D1, D2, D3, D4) are available, but when T1 commits, D4 is unavailable. Can T successfully commit? Why or why not, explain your answer.

Scenario 2: When T1 started its execution, the nodes D1, D2 are available, and the other nodes are unavailable. When T1 commits, the nodes D1, D2, D4 are available and D3 is unavailable. Can T successfully commit? Why or why not, explain your answer.

CAP Theorem

Any distributed database with shared data, can have **at most two** of the three desirable properties, C, A or P.

- **Consistency**: every node always sees the same data at any given instance (i.e., strict consistency)
- **Availability**: the system continues to operate, even if nodes crash, or some hardware or software parts are down due to upgrades
- **Partition Tolerance**: the system continues to operate in the presence of network partitions

Any distributed database with shared data, can have **at most two** of the three desirable properties, C, A or P. (对于 distributed DB, P总是存在)

- **Availability + Partition Tolerance**
- **Consistency + Partition Tolerance**
- **Consistency + Availability**

e.g., Google and Amazon – focus on availability, sacrifice consistency: For some minor updates on product details, we don't have to maintain strong consistency. More importantly, we should make sure the availability (i.e., customers can still browse the website during the time when admin is changing those minor info). Because in e-commerce context, smooth using experience on online shopping for customers is more critical, although it sacrifice consistency and achieve eventual consistency, which is generally acceptable.

Type of Consistency

- Strong Consistency (e.g., ATM)
- Weak Consistency
- Eventual Consistency (e.g., Facebook posts, Dropbox) (对于更新有的显示有的不显示,但最终都显示)
  - Causal Consistency（A->B 由A引起的修改，user会先看到A再查询时看到B）
  - Read-your-write Consistency（自己改的自己总能看见）
  - Session Consistency（log in-out算一个session）
  - Monotonic read consistency（A->B->C, if有人看到了B那以后只能看到C不会看到A）
  - Monotonic write consistency

Monotonic reads and read-your-writes are most desirable e.g., Facebook change photos

e.g., Airline reservation system

- When most of the seats are available (weak consistency, strong availability)
- When the plane is close to be filled (strong consistency, sacrifice availability)

Segment C and A

- Segment the system into different components. Each provides different types of guarantees.(浏览和下单不同的机制)
- Overall guarantees neither consistency nor availability. Each part of the service gets exactly what it needs
- Can be partitioned along different dimensions
- E.g., partition according to different data: 1.Product information – availability; 2. Shopping cart – availability; 3.Checkout, pay the bill, shipping records - consistency

Why Facebook chooses eventual consistent model over the strong consistent one?

Unlike banking system, a bit less recent data is acceptable, hence strong consistency is not always needed. Here preference for the high availability over strong consistency, because millions of users are using the APP simultaneously, if strong consistency want to be achieved, those large number of users have to wait for update and maintenance, leading to bad user's experience. Strong consistency is more desirable for stock market and bank.

In design of automated teller machine (ATM):Strong consistency appear to be a nature choice However, in practice, **A beats C.** Higher availability means **higher revenue.** ATM will allow you to withdraw money *even if the machine is partitioned from the network.* However, it puts **a limit** on the amount of withdraw (e.g., $200). The bank might also charge you a fee when a overdraft happens

Although banks prefer strong consistency, such consistency may get relaxed under personal banking account. When partition happens, banks can still allow you to withdraw certain amount of money, violating strong consistency. However, such sacrifice can provide good service to customers (partition+availability) with only small controllable risk and weak consistency.

Airline reservation: we have to partitioning design under different scenario in terms of number of available tickets.
When # of available tickets is large, weak consistency is good (A+P). It doesn't matter when we see 401 tickets left or 399 left, hence outdated data is acceptable. We can keep availability, allowing users to book tickets even under network partition.
When # of available tickets is small, strong consistency over availability (A+C)=> more desirable to see accurate data, ensuring the plane is not overbooked.

Shopping online: 1. adding products to cart(high availability, eventual consistency for better experience); 2.下单失败的原因：（对lock: consistency of data; for CAP: shopping cart focus more on availability, not always the most recently updated info on the # of items available in here. 3.checkout: (stronger Consistency needed) (financial billing, shipping records involved): It is ok to wait longer for safety (exclusive locks for checkout)

If no network partition(没有partition时，仍然有node file的情况=>>>> latency)

1.No network partition, but still some failure of the node

2.Tradeoff between Consistency and Latency (unavailable -> extreme high latency) (if partition AC选一， 无partition，LC选一)

3.Achieving different levels of consistency/availability takes different amount of time

4.Maintaining consistency should balance between the strictness of consistency versus availability

CAP -> PACELC； PA/EL:有p选A or c，没p选latency选 A or C

If there is a partition (P), how does the system trade off availability and consistency (A and C);

Else (E), when the system is running normally in the absence of partitions, how does the system trade off latency (L) and consistency (C)?

PA/EL Systems: Give up both Cs for availability and lower latency

PC/EC Systems: Refuse to give up consistency and pay the cost of availability and latency

PA/EC Systems: Give up consistency when a partition happens and keep consistency in normal operations

PC/EL System: Keep consistency if a partition occurs but gives up consistency for latency in normal operations

BASE Properties

The CAP theorem proves that it is impossible to guarantee strict Consistency and Availability while being able to tolerate network partitions. This resulted in databases with relaxed ACID guarantees

In particular, such databases apply the BASE properties:

- **B**asically **A**vailable: the system guarantees Availability
- **S**oft-State: the state of the system may change over time
- **E**ventual Consistency: the system will eventually become consistent
- NoSQL (or Not-Only-SQL) databases follow the BASE properties

In a distributed database with shared data, the following options show the desired properties in pairs. Which one of these pairs cannot be achieved together at the same time? Ans: (Availability and consistency)

Question 7: [4 Marks]

Describe why a bank may prefer to refer to ACID properties than BASE properties in its database systems. At the same time why would an online shopping system may be built with mainly BASE properties in mind? Briefly explain each.

Financial transactions require consistency and durability, as even minor errors may lead to significant financial losses or legal issues. ACID guarantees that transactions are fully completed or rolled back, which is critical for maintaining the trustworthiness and integrity of the bank's data.

In contrast, an online shopping system may use BASE, allowing for more flexibility and higher availability. For shopping platform, it is acceptable for data to be eventually consistent rather than immediately. This is because updating the value to every node may consume a bit long time, which impact the users' experience on browsing the product especially when stock is enough. Although minor inconsistency exists, availability can be guaranteed and it makes sure that platform can still provide service to users.

Why modern DBs rely on BASE rather than ACID? 1. EC: large scale databases are distributed, not possible to guarantee network tolerance, hence weaker consistency is applied to database. When system becomes eventually consistent, all the nodes are updated to the recent data, but which takes some time to propagate that info. 2. Soft-state: state of system may change with time (whether focusing more on consistency or A). 取决于应用本身

Type of NoSQL Databases

1. Document Stores

- Documents are stored in some standard format or encoding (e.g., XML, JSON, PDF or Office Documents)
- Documents can be indexed

2. Graph Databases

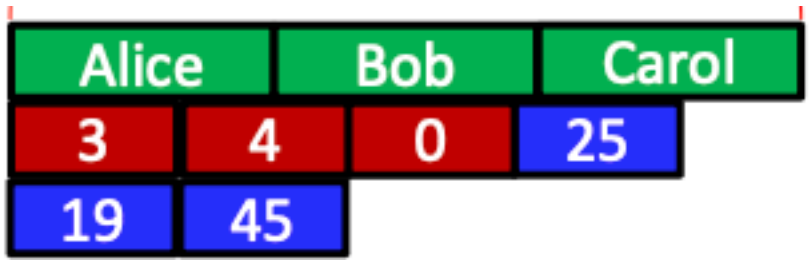
- Data are represented as vertices and edges
- Graph databases are powerful for graph-like queries (e.g., find the shortest path between two elements)

3. Key-Value Stores(key 不能重复)

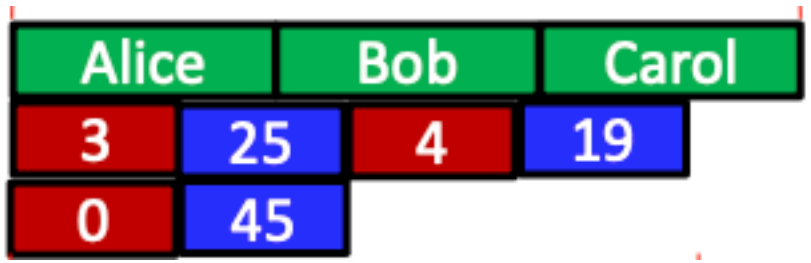
- Keys are mapped to (possibly) more complex value (e.g., lists)
- Keys can be stored in a hash table and can be distributed easily
- Such stores typically support regular CRUD (create, read, update, and delete) operations (no join functions)

4. Columnar Databases

- Columnar databases are a hybrid of RDBMSs and Key-Value stores
- Values are queried by matching keys



Columnar



Columnar with Locality Groups

Type of NoSQL Databases

Consider the following relational database table. The most common queries on this table are – (i) Finding all the postal addresses, and (ii) Finding all the email addresses. However, the company that is running these queries finds that the query time is not as fast as they would like, especially when this relational database table has many records. Can they use any different types of databases to improve the efficiency of these two queries (without adding any new hardware)? How do they need to store these data in those different types of databases to improve the efficiency of these two queries? Provide **three** such database solutions.

ID	Name	Postal address	… (many other columns)	Email address
1	Smith	10 Central Street, Disneyland	…	smith@email.com
2	Zhou	22 South Road, Someland	…	zhou@email.com
3	Lin	12 North Road, Lalaland	…	lin@email.com

1. Key-Value Pair Database: In this setup, each user ID would act as a key, and the corresponding postal address and email address would be the values. This structure allows for quick lookups when retrieving the entire dataset, as key-value stores are optimized for fast access, especially when retrieving large sets of records.

Choosing a column-based database can optimize read performance for specific columns, like postal and email addresses. By storing each column separately, only the relevant data (postal or email addresses) is accessed for each query, significantly reducing I/O and improving query speed without scanning unnecessary columns. This is ideal for high-read, selective queries.

simple document file:

- For postal addresses: You could structure each document with an `ID` field and a `postal_address` or ‘email addresses’ field. Querying for all postal addresses would be a simple, efficient retrieval of the `postal_address` field across documents.

This approach avoids complex joins and enables efficient storage and retrieval of user information, as each document contains the full set of relevant fields. Document databases are particularly suited for read-heavy workloads, which aligns with the requirement for efficient querying in this scenario.

Remote Backup System

Detection of failure: Backup site must detect when primary site has failed

- To distinguish primary site failure from link failure, maintain several communication links between the primary and the remote backup(不止一个 communication links)
- Use heart-beat messages

Transfer of control:

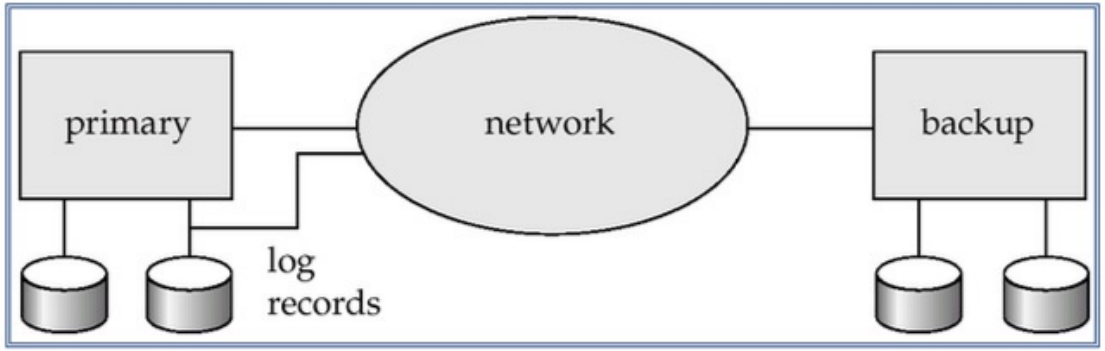
- To take over control, backup site first perform recovery using its copy of the database and all the log records it has received from primary. Thus, completed transactions are redone and incomplete transactions are rolled back
- When the backup site takes over processing it becomes the new primary(backup 接手后变为primary)

Time to recover:

- To reduce delay in takeover, backup site periodically processes the redo log records
- In effect, it performs a checkpoint, and can then delete earlier parts of the log

Hot-Spare configuration permits very fast takeover:

- Backup continually processes redo log record as they arrive, applying the updates locally
- When failure of the primary is detected the backup rolls back incomplete transactions, and is ready to process new transactions



Remote Backup System

Ensure durability of updates by delaying transaction commit until update is logged at backup. But we can avoid this delay by permitting lower degrees of durability. (先把log发到backup, log过去后再进行真正的transaction,像WAL)

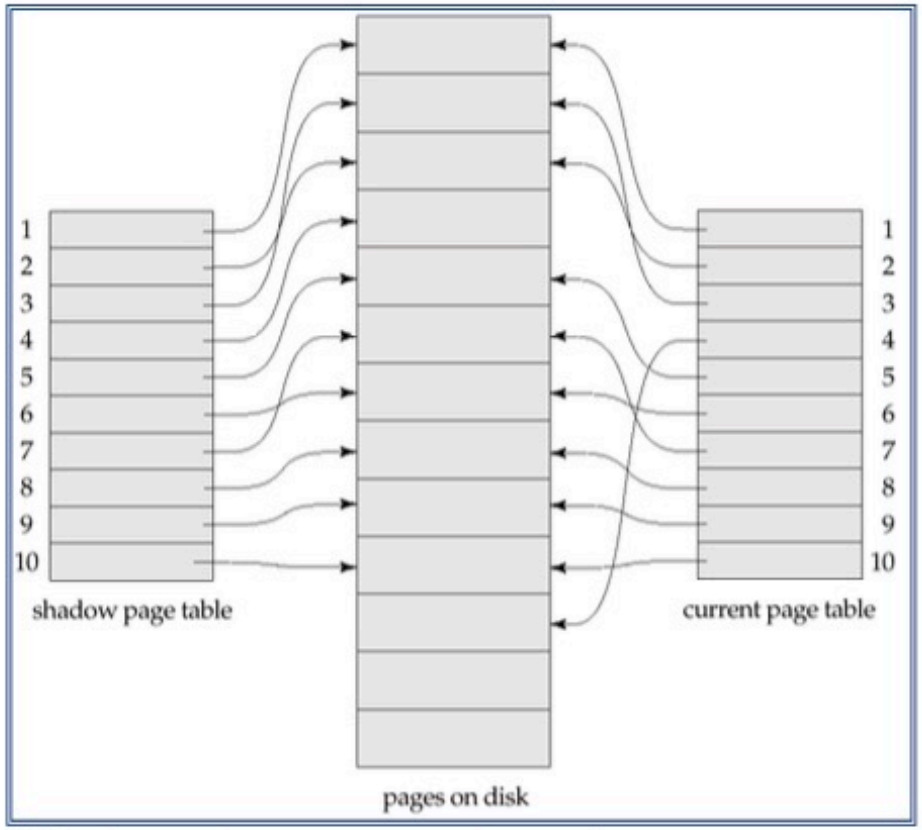
- One-safe: commit as soon as transaction's commit log record is written at primary
  (Problem: updates may not arrive at backup before it takes over.) （log写在primary不放在backup）
- Two-very-safe: commit when transaction's commit log record is written at primary and backup
  (Reduces availability since transactions cannot commit if either site fails.) （等log完全写在backup后上再commit）
- Two-safe: proceed as in two-very-safe if both primary and backup are active. If only the primary is active, the transaction commits as soon as its commit log record is written at the primary

(Better availability than two-very-safe; avoids problem of lost transactions in one-safe.)
中间值，两个都active时用慢的，一个fail用one-safe.

Q. A company has 20TBs of data. Only a particular 2GBs of that data gets frequently changed by its users, and the remaining data stays the same. This company has a limited budget for extra storage and hardware for backups of their data. What good backup strategies can they follow within their limited budget?
Ans: Using logged writes is an efficient solution for the company's needs. By logging only the changes made to the 2GB of frequently modified data, this approach avoids the need for full data backups each time, reducing storage demands and eliminating the need for additional hardware or storage. Logged writes ensure that data can be restored quickly and accurately, supporting the company's limited budget without sacrificing data integrity.

Shadow Paging:两张表 开始时两表一样，crash发生要recover 角色转换。

- Idea: maintain two page tables during the lifetime of a transaction, i.e., the current page table, and the shadow page table
- Store the shadow page table in **nonvolatile storage**, such that state of the database prior to transaction execution may be recovered. Shadow page table is never modified during execution.
- To start with, both the page tables **are identical**. Only current page table is used for data item accesses during execution of the transaction
- Whenever any page is about to be written
  - A copy of this page is made onto an unused page
  - The current page table is then made to point to the copy
  - The update is performed on the copy



Shadow and current page tables after write to page 4

Shadow Paging - Commit

- To commit a transaction
  1. Flush all modified pages in main memory to disk
  2. Output current page table to disk
  3. Make the current page table the new shadow page table
     - keep a pointer to the shadow page table at a fixed (known) location on disk.
     - to make the current page table the new shadow page table, simply update the pointer to point to current page table on disk.
- Once pointer to shadow page table has been written, transaction is committed.

Advantages of shadow-paging over log-based schemes

- No overhead of writing log records
- Recovery is trivial - **new transactions can start right away, using the shadow page table.？？？ Why**

Disadvantages

- Copying the entire page table is very expensive when the page table is large
- Commit overhead is high - Need to flush every updated page, and page table
- Pages not pointed to from current/shadow page table should be freed (garbage collected)
- Data gets fragmented (related pages get separated on disk)
- Hard to extend algorithm to allow transactions to run concurrently

Shadow Paging

- Recovery is trivial - **new transactions can start right away, using the shadow page table.？？？ Why**

Shadow page table maintains all status and info before the transaction start so that shadow PT can be used directly to recover before crash recovery without overhead of writing any log records.

1. Atomic Switch: If the transaction commits successfully, the system simply replaces the shadow page table with the current page table, making the changes permanent. If the transaction fails or needs to be rolled back, the system discards the current page table, reverting to the shadow page table. This operation is atomic and does not require complex rollback procedures.

2. **No Logging Required:** Unlike methods that rely on logging, the shadow page table method does not need logs to track changes, making the recovery process faster and simpler. The shadow page table always holds a consistent version of the data, so there's no need to undo changes or replay logs.

3. **Immediate Availability:** Since the shadow page table remains unmodified and can be used immediately after a rollback or crash, new transactions can start right away. The system just points to the stable shadow page table, ensuring that data is available without additional recovery steps.

Backup & Crash Recovery

Strategy plan based on:

- Goals and requirement of your organization/task
- The nature of your data and usage pattern
- Constraint on resources

Design backup strategy:

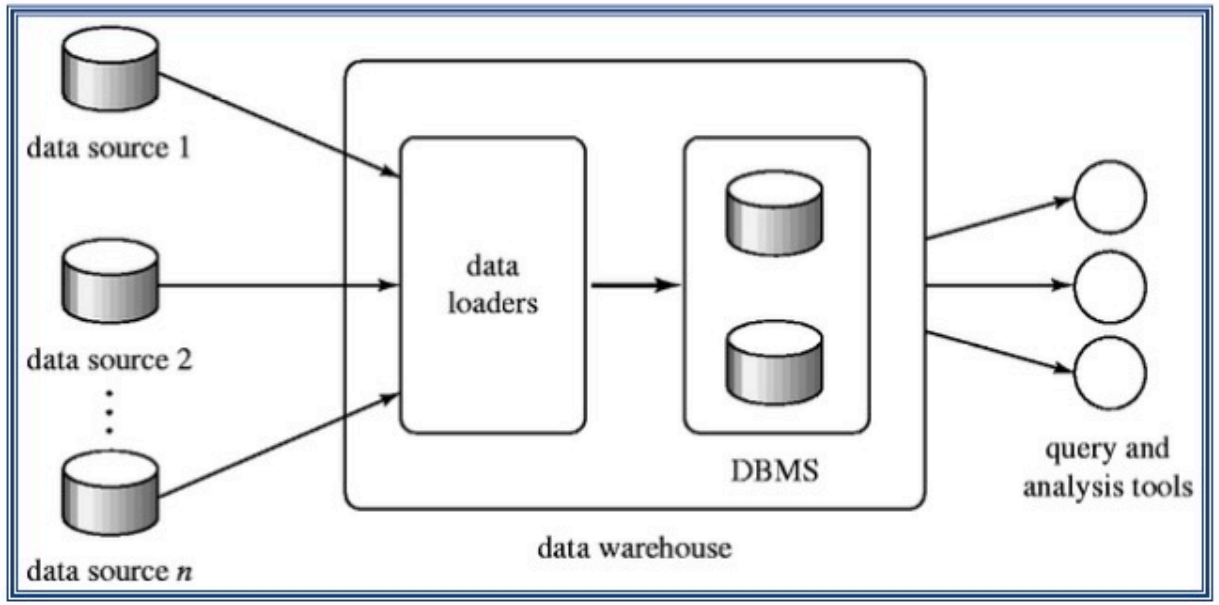
- Full disk backup vs partial - Are changes likely to occur in only a small part of the database or in a large part of the database?
- How frequently data changes
  - If frequent: use differential backup that captures only the changes since the last full database backup (partial backup: just copy changes not all part)
- Space requirement of the backups – depends on the resource
- Multiple past instances of backup – useful if point-in-time recovery is needed (一个上周，一个为上个月 what the state was at that point)

Choose the right recovery model for your application

- Types of recovery models in MS SQL server:
- Simple: No logs, but has backups. Recovery is done from the last backup
- Full: Uses logs plus backups, regular checkpoints
- Bulk logged: Logs are not maintained for each individual writes, but for multiple writes together* (reduce overhead of number of logs generate every time, frequent checkpoints come with costly computational power but faster recovery)

Data Warehousing

- Corporate decision making requires a unified view of all organizational data, including historical data
- A data warehouse is a repository (archive) of information gathered from multiple sources, stored under a unified schema, at a single site
- Usually OK to have slightly out-of-date data at warehouse



- When and how to gather data (要或者给)
  - Source driven architecture: data sources transmit new information to warehouse, either continuously or periodically (e.g. at night)
  - Destination driven architecture: warehouse periodically requests new information from data sources

Q In a data warehouse, when should the data sources transmit new information to warehouse frequently, instead of periodically (e.g., at night)?

Ans: Some data that need real time data and historical data combined analytics and decision making. Stock market: We need to record all exchange data of all companies or stockholders in stock market frequently so that we can make use of these data to compare their recent behavior and then promptly identify and prevent potential fraud or abnormal transactions.

Real-time inventory management: in e-commerce or retail, real-time inventory data is crucial for operation, especially for the period of important holiday when the number of orders increasing substantially. Because we need to analyze and updated inventory information frequently also combined with historical orders to make decision on whether increasing or decreasing production, preventing overselling or backlog of products (instantly)

Data Warehousing – Design Issues

- Keeping warehouse exactly synchronized with data sources (e.g., using two-phase commit) is too expensive
  - Usually **OK to have slightly out-of-date data** at warehouse
  - Data/updates are periodically downloaded form online transaction processing (OLTP) systems (most of the DBMS work we have seen so far)
- What schema to use
  - Depends on purpose (e.g., How to reduce budget? / Forcasting)
  - Schema integration
- Data cleansing
  - e.g., Correct mistakes in addresses (misspellings, zip code errors)
  - e.g., Merge address lists from different sources and purge duplicates
- How to propagate updates - The data stored in a data warehouse is documented with an element of time, either explicitly or implicitly
- What data to summarize
  - Raw data may be too large to store
  - **Aggregate values (totals/subtotals) often suffice（有时候汇总数据就够了 月度季度销量数据）**
  - Queries on raw data can often be transformed by query optimizer to use aggregate values