CS5300
Database Systems

Database Recovery

A.R. Hurson
323 CS Building
hurson@mst.edu

Database Systems

Note, this unit will be covered in two lectures. In case you finish it earlier, then you have the following options:

1) Take the early test and start CS5300.module8
2) Study the supplement module (supplement CS5300.module7)
3) Act as a helper to help other students in studying CS5300.module7

Note, options 2 and 3 have extra credits as noted in course outline.

Database Systems

Enforcement of background

Glossary of prerequisite topics

Familiar with the topics?

No

Review CS5300.module7.background

Yes

Take Test

Pass?

No

Remedial action

Yes

Glossary of topics

At the end: take exam, record the score, impose remedial action if not successful

Current Module

Familiar with the topics?

No

Take the Module

Yes

Take Test

Pass?

No

Yes

Options

Study next module?

Lead a group of students in this module (extra credits)?

Study more advanced related topics (extra credits)?

Extra Curricular activities

Database Systems

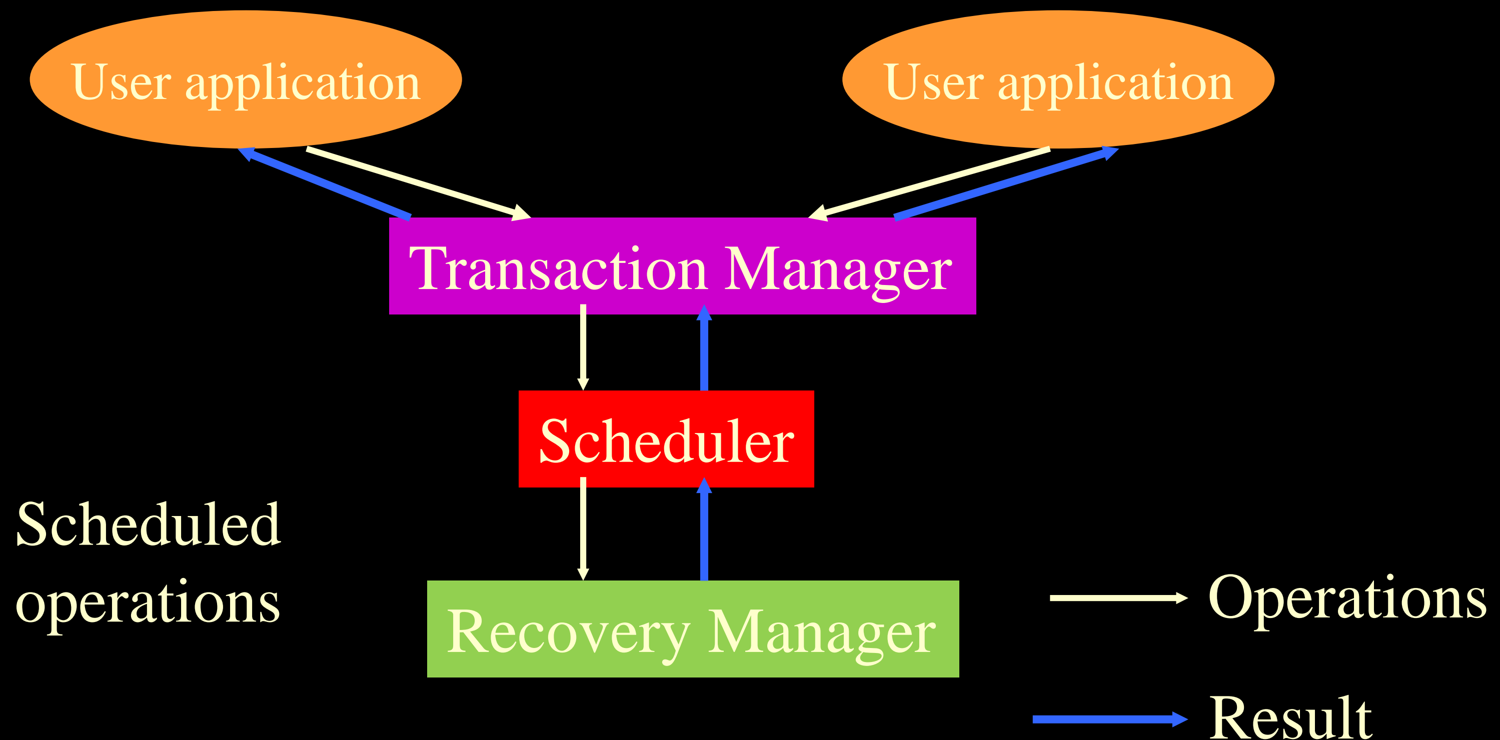
- You are expected to be familiar with:
  - Relational database model,
  - SQL
  - Transaction processing and concurrency control
- If not, you need to study CS5300.module7.background

Database Systems

- Need for recovery
  - According to ACID property, transactions are logical basic unit of database processing. Either all of the operations of a transaction must be executed successfully and their results are permanently recorded (transaction is committed) or transaction did not have any effect on the database and any other transactions (transaction is aborted).
  - Therefore, if a transaction fails after executing some of its operations, their effect must be wiped out of database (i.e., operations already executed must be undone and have no lasting effect).

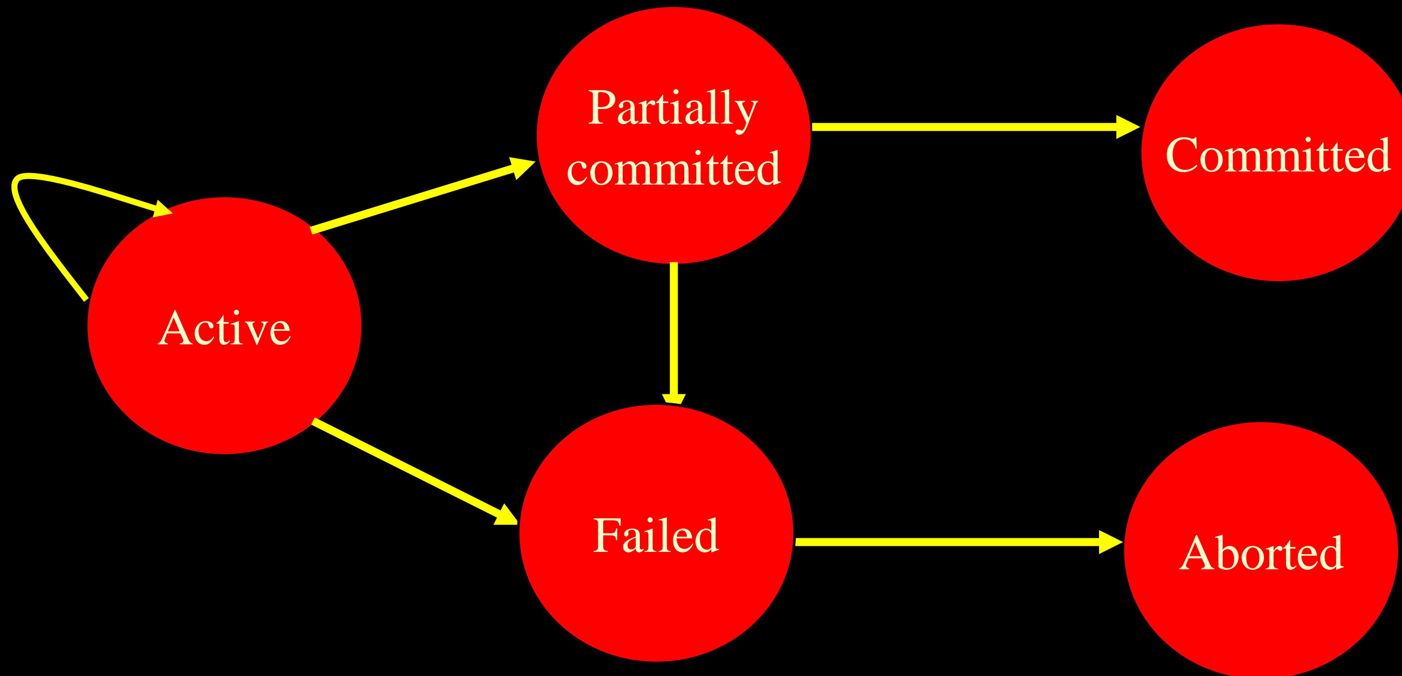
Database Systems

- Centralized Transaction Execution



Database Systems

- Transaction states



Database Systems

- Centralized Transaction Execution
  - Transaction Manager is responsible for coordinating the execution of the database operations on behalf of an application.
  - Scheduler is responsible for the implementation of a specific concurrency control algorithm.
  - Recovery manager is responsible to implement procedures that transform database into a consistent state after a failure.

Database Systems

- A computer system, like any other system, is subject to failure from a variety of causes. In the event of failure, information may be lost. Therefore, database system must take actions in advance to ensure that the information can be recovered (preserving atomicity and durability properties of transactions).
- A recovery scheme, restores the database to its consistent state before the failure.

Database Systems

- There are various types of failure that must be dealt with differently:
  - Transaction failure
    - Logical error
    - System error
  - System crash
  - Disk failure

Database Systems

- Another classification of Failures
  - Computer failure (system crash)
  - Transaction or system error
  - Local errors or exception
  - Concurrency control enforcement
  - Disk failure
  - Physical problems and catastrophes

Database Systems

- Recovery from a transaction failure means that the database must be restore to its consistent state before failure. To do this, a system log about the changes made in the database are maintained.
- Using the system log then the following general recovery strategy will be follows:
  - In case of massive failure, the database is restored by using its back up and redoing the operations of committed transactions.
  - In other cases, the database is forced to its earlier consistent state by undoing and/or redoing some of the operations of failed and/or finished transaction(s).

Database Systems

- Based on update policy, one can distinguish two main recovery protocol for non-catastrophic transaction failures:
  - Deferred update: where the database is not updated before the commit point.
  - Immediate update: where the database may be updated by some operations before the commit point.

Database Systems

- The log is the widely used structure for recording database modifications. It is a sequence of log records.

Database Systems

- An update log record describes a single database write and consists of the following information:
  - Transaction identifier
  - Data-item identifier
  - Old value
  - New value
- Other log records record significant events during the course of a transaction:
  - $<T_i \text{ start}>$
  - $<T_i, X_j, V_1, V_2>$
  - $<T_i \text{ commit}>$
  - $<T_i \text{ abort}>$

Database Systems

- Deferred Update (No-undo/Redo algorithm)
  - This approach ensures atomicity be recording all database modifications in the log, but deferring the execution of write operations until the transaction is done with its final action (partially committed).

Database Systems

- Deferred Update (No-undo/Redo algorithm)
  - The execution of transaction T_i proceeds as follows:
    - Before T_i starts its execution, a record <T_i start> is written to the log,
    - A write(x) operation by T_i, inserts a new record in the log,
    - When T_i partially commits, a record <T_i commit> is added to the log

Database Systems

Deferred Update (No-undo/Redo algorithm)

- Consider the following two transactions:

T_0:	Read (A);
	A := A – 50;
	Write (A);
	Read (B);
	B := B + 50;
	Write (B);

T_1:	Read (C);
	C:= C – 100;
	Write (C);

- Assume these two transactions are executed serially in the order of T_0, T_1. Further assume that the contents of A, B, and C are $1,000, $2,000, and $700.

Database Systems

- **Deferred Update** (No-undo/Redo algorithm)
  - Portion of the log related to these two transactions look like”

    <T_0 start>

    <T_0, A, 950>

    <T_0, B, 2050>

    <T_0 commit>

    <T_1 start>

    <T_1, C, 600>

    <T_1 commit>

Database Systems

- Deferred Update (No-undo/Redo algorithm)

Log	Database
<T_0 start>	
<T_0, A, 950>	
<T_0, B, 2050>	
<T_0 commit>	
	A=950
	B=2050
<T_1 start>	
<T_1, C, 600>	
<T_1 commit>	
	C=600

Database Systems

- Deferred Update (No-undo/Redo algorithm)
  - Let us look at the following scenarios:
  - Case1: Crash occurs just after the log record write(B) of T_0. Then, the log looks as:

    Log

    <T_0 start>

    <T_0, A, 950>

    <T_0, B, 2050>

  - When the system comes back up, no redo actions is needed, since no changes has been made to the database and the contents of A and B accounts remain the same. The log records of incomplete transaction T_0 may be deleted from the log.

Database Systems

- Deferred Update (No-undo/Redo algorithm)
  - Case2: Crash occurs just after the log record write(C) of T_1. Then, the log looks as:

    Log

    <T_0 start>

    <T_0, A, 950>

    <T_0, B, 2050>

    <T_0 commit>

    <T_1 start>

    <T_1, C, 600>
  - When the system comes back up, operation redo(T_0) is performed (because of <T_0 commit>) the contents of A and B accounts will be 950 and 2050, respectively, and C remains the same as before (700). The log records of incomplete transaction T_1 may be deleted from the log.

Database Systems

- **Deferred Update** (No-undo/Redo algorithm)
  - Case3: Crash occurs just after the log record <T_1 commit>. Then, the log looks as:

    Log

    <T_0 start>

    <T_0, A, 950>

    <T_0, B, 2050>

    <T_0 commit>

    <T_1 start>

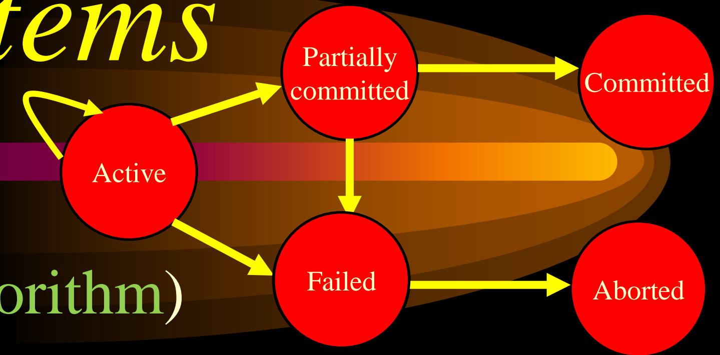
    <T_1, C, 600>

    <T_1 commit>
  - When the system comes back up, operations redo(T_0) and redo(T_1) are performed (because of <T_0 commit> and <T_1 commit>). The contents of A, B, and C accounts will be 950, 2050, and 600, respectively.

Database Systems

- Deferred Update (No-undo/Redo algorithm)
  - Before reaching the commit point, all updates are recorded in the local transaction workspace.
  - Before commit, the updates are recorded persistently in the log, and then after commit, the updates are written to the database on disk.
  - If the transaction fails before reaching commit point, the database is unchanged, so undo is not needed.
  - It may be necessary to redo the effect of the operations of a committed transaction from the log, though.

Database Systems



- Immediate Update (Undo/Redo algorithm)
  - This scheme allows database modification to happen while transaction is in active state. In the event of failure or crash, the system must use the old-value of the log records to store the modified data items to the value they had prior to the execution of transaction (i.e., undo).
  - Before the execution of transaction T_i, the record $<T_i$ start$>$ is written to the log. Any write(x) by T_i precedes by inserting a $<T_i, X, V_{old}, V_{new}>$ record to the log. When T_i partially commits, the $<T_i$ commit$>$ will be recorded to the log.

Database Systems

- Immediate Update (Undo/Redo algorithm)
  - Consider the following two transactions:

T_0:	Read (A);
	A := A – 50;
	Write (A);
	Read (B);
	B := B + 50;
	Write (B);

T_1:	Read (C);
	C:= C – 100;
	Write (C);

  - Assume these two transactions are executed serially in the order of T_0, T_1. Further assume that the contents of A, B, and C are $1,000, $2,000, and $700.

Database Systems

- Immediate Update (Undo/Redo algorithm)
  - Portion of the log related to these two transactions look like:

    <T_0 start>

    <T_0, A, 1000, 950>

    <T_0, B, 2000, 2050>

    <T_0 commit>

    <T_1 start>

    <T_1, C, 700, 600>

    <T_1 commit>

Database Systems

- Immediate Update (Undo/Redo algorithm)
  - One possible order in which the actual output took place in both database and log is as follows:

Log	Database
<T_0 start>	
<T_0, A, 1000, 950>	
<T_0, B, 2000, 2050>	
	A=950
	B=2050
<T_0 commit>	
<T_1 start>	
<T_1, C, 700, 600>	
	C=600
<T_1 commit>	

Database Systems

- Immediate Update (Undo/Redo algorithm)
  - The recovery scheme uses two recovery procedures:
    - Undo(T_i) to restore the value of all data items updated by T_i to the old values.
    - Redo(T_i) to set the value of all data items updated by T_i to the new values.

- Immediate Update (Undo/Redo algorithm)
  - After a failure, the recovery scheme consults the log to determine which transactions need to be redone and which transactions need to be undone.
    - Transaction T_i needs to be undone if the log contains <T_i start> but does not contain <T_i commit>.
    - Transaction T_i needs to be redone if the log contains <T_i start> and the <T_i commit>.

Database Systems

- Immediate Update (Undo/Redo algorithm)
  - Let us look at the following scenarios:
  - Case1: Crash occurs just after the log record write(B) of T_0. Then, the log looks as:

    Log

    <T_0 start>

    <T_0, A, 1000, 950>

    <T_0, B, 2000, 2050>
  - When the system comes back up, the record <T_0 start> is in the log but the record <T_0 commit> is not, so T_0 must be undone. Undo(T_0) restores A and B accounts to 1000 and 2000, respectively.

Database Systems

- Immediate Update (Undo/Redo algorithm))
  - Case2: Crash occurs just after the log record write(C) of T_1. Then, the log looks as:

    Log

    <T_0 start>

    <T_0, A, 1000, 950>

    <T_0, B, 2000, 2050>

    <T_0 commit>

    <T_1 start>

    <T_1, C, 700, 600>
  - When the system comes back up, two recovery actions need to be taken: Undo(T_1) and redo(T_0). At the end of recovery procedure the contents of A, B, and C accounts will be 950, 2050, and 700, respectively.

Database Systems

- Immediate Update (Undo/Redo algorithm)
  - Case3: Crash occurs just after the log record <T_1 commit>. Then, the log looks as:

    Log

    <T_0 start>

    <T_0, A, 1000, 950>

    <T_0, B, 2000, 2050>

    <T_0 commit>

    <T_1 start>

    <T_1, C, 700, 600>

    <T_1 commit>
  - When the system comes back up, both T_0 and T_1 need to be redone. The contents of A, B, and C accounts will be 950, 2050, and 600, respectively.

Database Systems

- Immediate Update (Undo/Redo algorithm)
  - The database may be updated by some operations of the transaction before reaching its commit point. However, these operations are recorded in the log permanently, by force-writing.
  - If the transaction fails before reaching its commit point then the effect of its operations must be undone (rolled back).

Database Systems

- Checkpoints
  - In case of failure and after recovery from failure, the log must be consulted to determine which transaction need to be undone and which transaction needs to be redone. In principle, we need to search the entire log!
  - This is too time consuming and in many instances unnecessary.

Database Systems

- Checkpoints
  - The concept of checkpoints is used to reduce the overhead.
  - While maintaining the log, system periodically performs checkpoints (the following sequence of actions):
    - Outputs onto stable storage all log records residing in main memory,
    - Outputs to the disk all modified buffer blocks,
    - Outputs onto stable storage a log record <checkpoints>.

Database Systems

- Checkpoints
  - Transactions are not allowed to perform any update while a checkpoint is in progress.
  - The presence of a <checkpoints> record allows the system to streamline the recovery procedure.

Database Systems

- Checkpoints
  - Assume transaction T_i has committed before a checkpoint. As a result a <T_i commit> record appears before <checkpoints> record in the log. As a result, all database modifications made by T_i must have been done to the database either prior to the checkpoint or as part of checkpoint process.

Database Systems

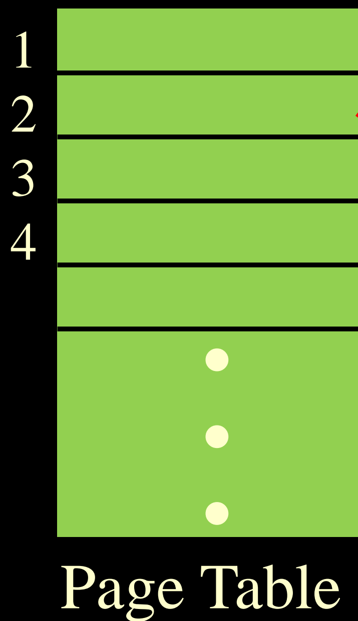
- Shadow Paging (No Undo/No Redo algorithm)
  - Shadow paging is an alternative to log-based recovery procedure.
  - Under certain circumstances, shadow paging offers fewer disk accesses. However, it is much harder to be extended for concurrent execution of transactions.

Database Systems

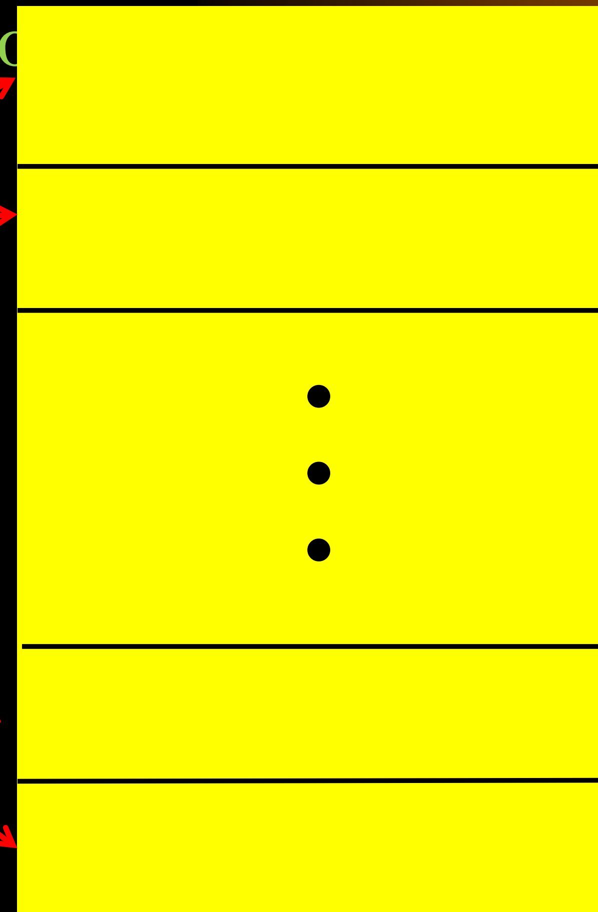
- Shadow Paging (No Undo/No Redo algorithm)
  - Database is partitioned into fixed-length blocks (i.e., pages). A page table is used to hold addresses of pages on the disk. Each page has an entry in the page table.

Database Systems

- Shadow Paging (No Undo ... n)



Page Table

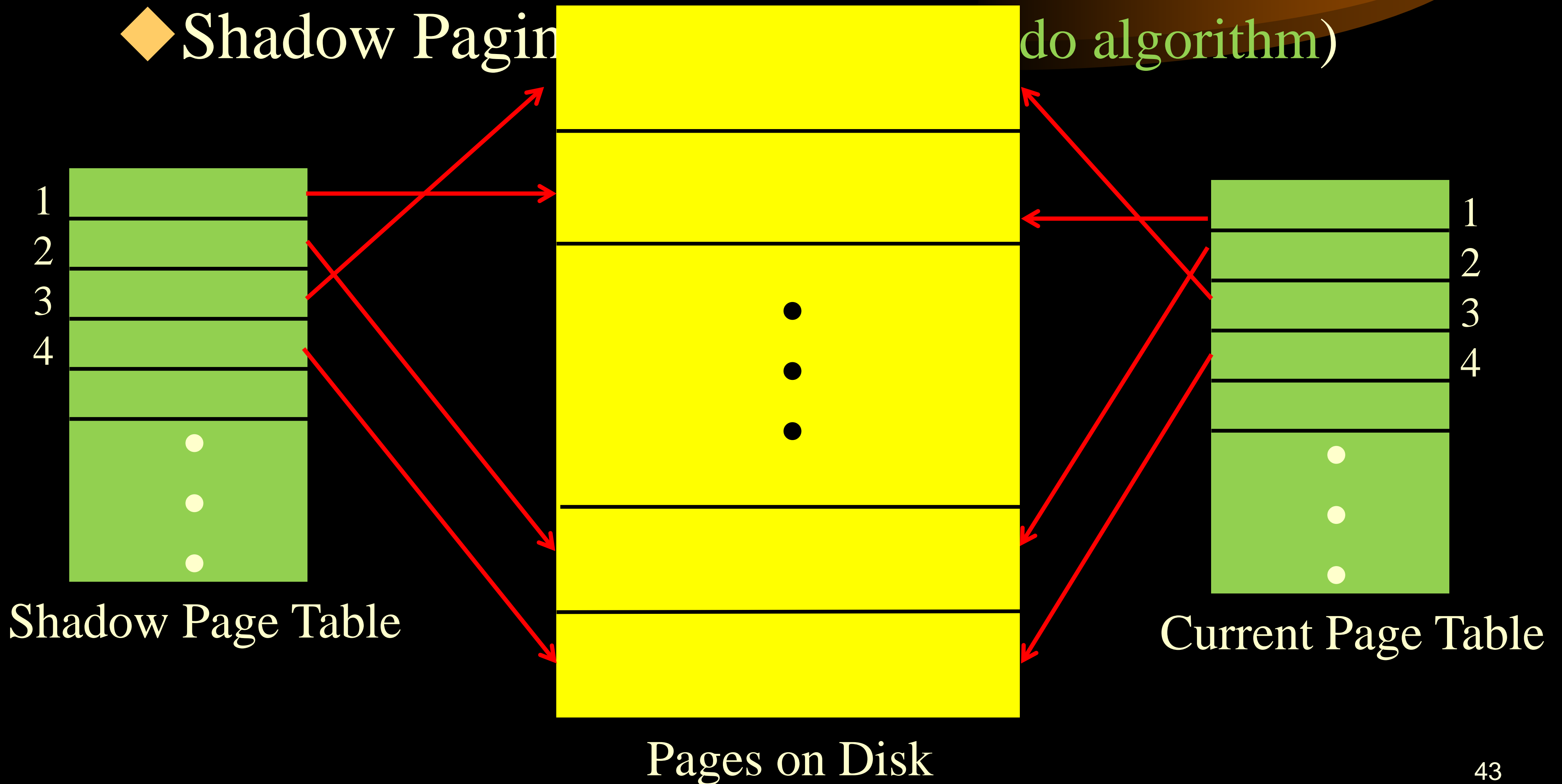


Pages on Disk

Database Systems

- Shadow Paging (No Undo/No Redo algorithm)
  - In shadow paging, two page tables are going to be maintained during the course of a transaction:
    - Current Page Table
    - Shadow page table
  - At the start of a transaction, both page tables are identical. During the course of transaction, current page table may change (due to write operation), but shadow page table remains unchanged.

Database Systems



Database Systems

- Shadow Paging (No Undo/No Redo algorithm)
  - All input and output operations use the current page table to locate database pages on disk.

Database Systems

- Shadow Paging (No Undo/No Redo algorithm)
  - Assume T_j performs a write(X) operation and X resides on the ith page. The system executes write(X) as follows:
    I. If the ith page is not in main memory, then the system issues input(X)
    II. If this is the 1st write performs on the ith page, the current page table is modified as follows
       a. System finds an unused page on disk
       b. System copies the content of the ith page to the page found in (a)
       c. Current page table is modified to the page found in (a)
    III. Value of x_j is assigned to X in the buffer page

Database Systems

- Shadow Paging (No Undo/No Redo algorithm)
  - When a transaction commits, the current page table becomes the new shadow page table and the next transaction is allowed to start execution. Critical issue is the fact that the shadow page table must be stored in a non-volatile storage since it provides the only means of locating database pages.
  - After system comes back up, it copies the shadow page table into main memory and uses it for subsequent transactions. Unlike log-based schemes, it does not need to invoke undo operations.

Database Systems

- Shadow Paging (No Undo/No Redo algorithm)
  - To commit a transaction we must do the following:
    I. Ensure that all buffer pages in main memory that have been modified are output to disk
    II. Output the current page table to disk
    III. Output the disk address of the current page table to the fixed location in stable storage containing the address of the shadow page table

Database Systems

- Concurrent transactions
  - So far, we considered recovery in an environment where only a single transaction at a time is executing. We extend the scope of log-based recovery scheme to deal with multiple concurrent transactions.

Database Systems

- Concurrent transactions
  - We are going to make the following assumptions:
    - The system has one disk buffer and a single lock,
    - If transaction T has updated a data item Q, no other transaction may update the same data item until T has committed or has been rolled back (using strict two-phase locking ensures this requirement), and
    - Immediate update scheme is used and we permit a buffer block to have data items updated by one or more transactions.

Database Systems

- Transaction Rollback
  - Failed transaction T_i is rolled back by using the log. The system scans the log backward; for every log record of the form $<T_i, X_j, V_1, V_2>$ in the log, data item X_j is restored with its old value V_1. Scanning stops when log record $<T_i\ start>$ is detected.