COMP90050 Advanced Database Systems

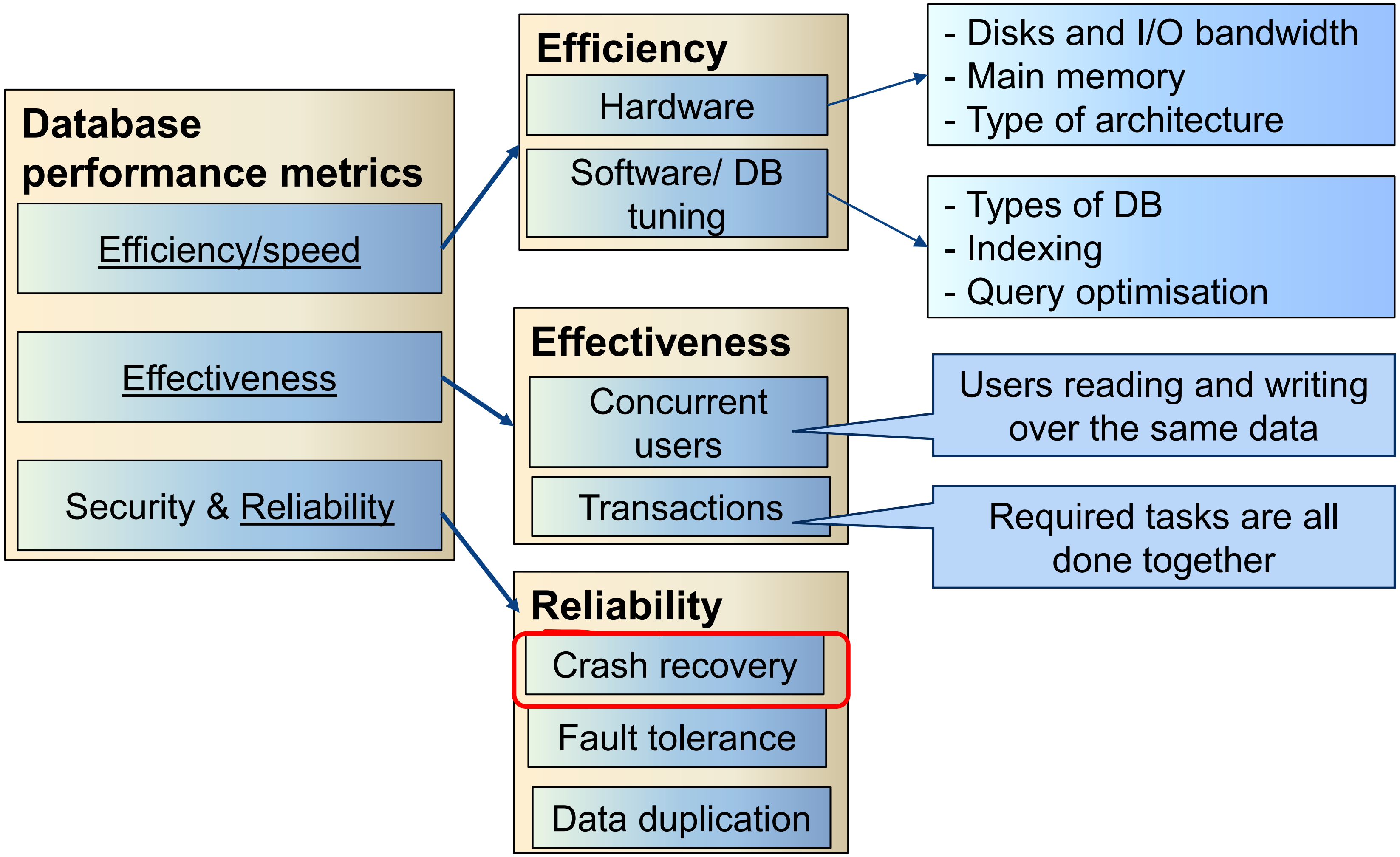
THE UNIVERSITY OF MELBOURNE

Winter Semester, 2023

Lecturer: Farhana Choudhury (PhD)

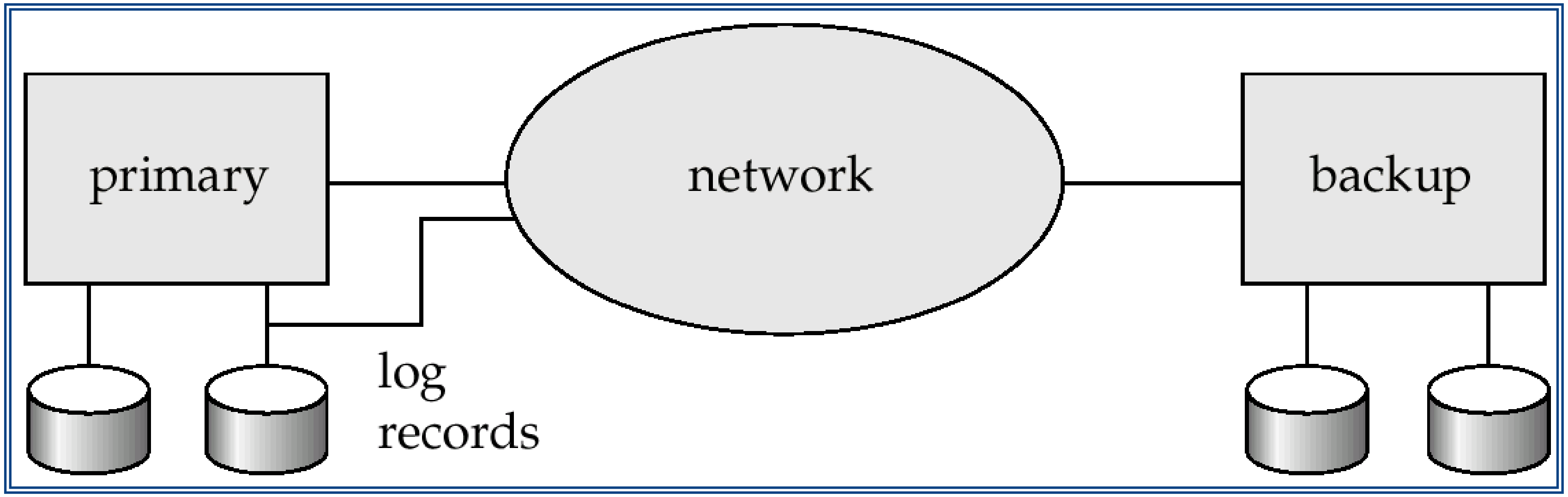
Week 4 part 3

Core Concepts of Database management system



Other Considerations: Remote Backup Systems

Remote backup systems provide high availability by allowing transaction processing to continue even if the primary site is destroyed



Remote Backup Systems Contd.

Detection of failure: Backup site **<u>must detect when primary site has failed</u>**

- To distinguish primary site failure from link failure, **maintain several communication links between the primary and the remote** backup
- **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**

Remote Backup Systems Contd.

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 Systems Contd.

To ensure durability of updates - delay transaction commit until update is logged at backup

But we can avoid this delay by permitting lower degrees of durability

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.

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.

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 is commit log record is written at the primary

- Better availability than two-very-safe; avoids problem of lost transactions in one-safe.

Alternative to Logs: Shadow Paging

Shadow paging is an alternative to log-based recovery

Idea: **maintain *two* pageTables during the lifetime of a transaction** –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

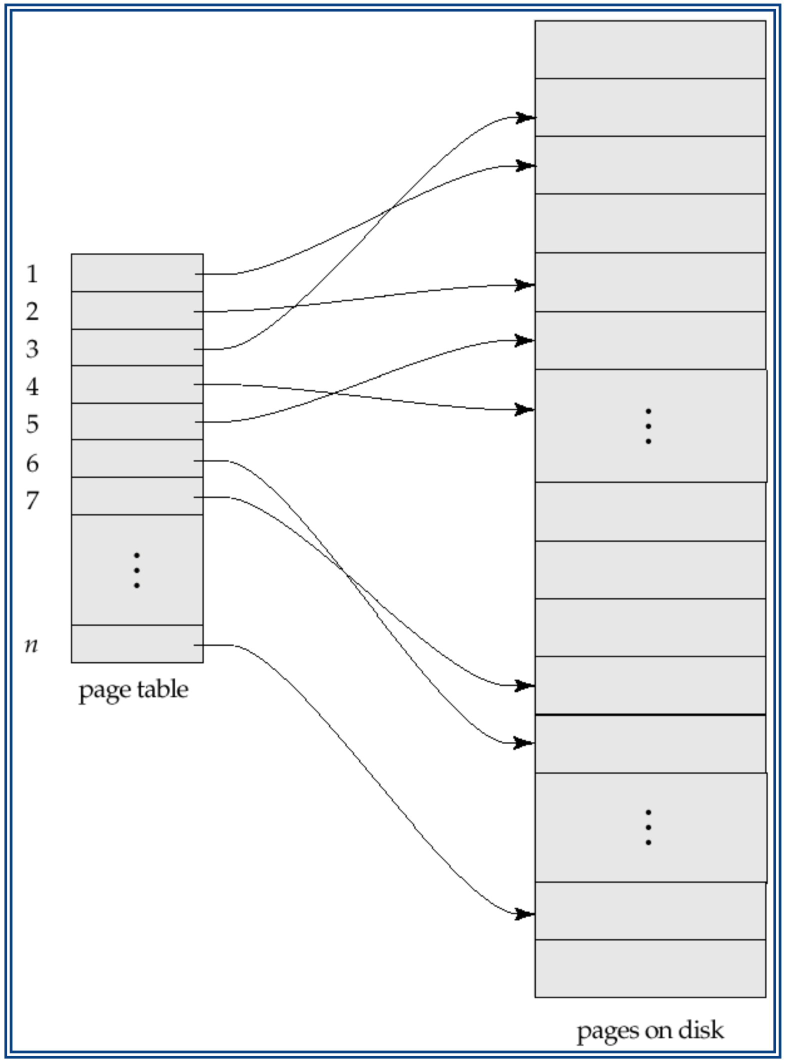
Alternative to Logs: Shadow Paging

To start with, both the page tables are identical. **Only the current page table is used for data item accesses** during execution of the transaction

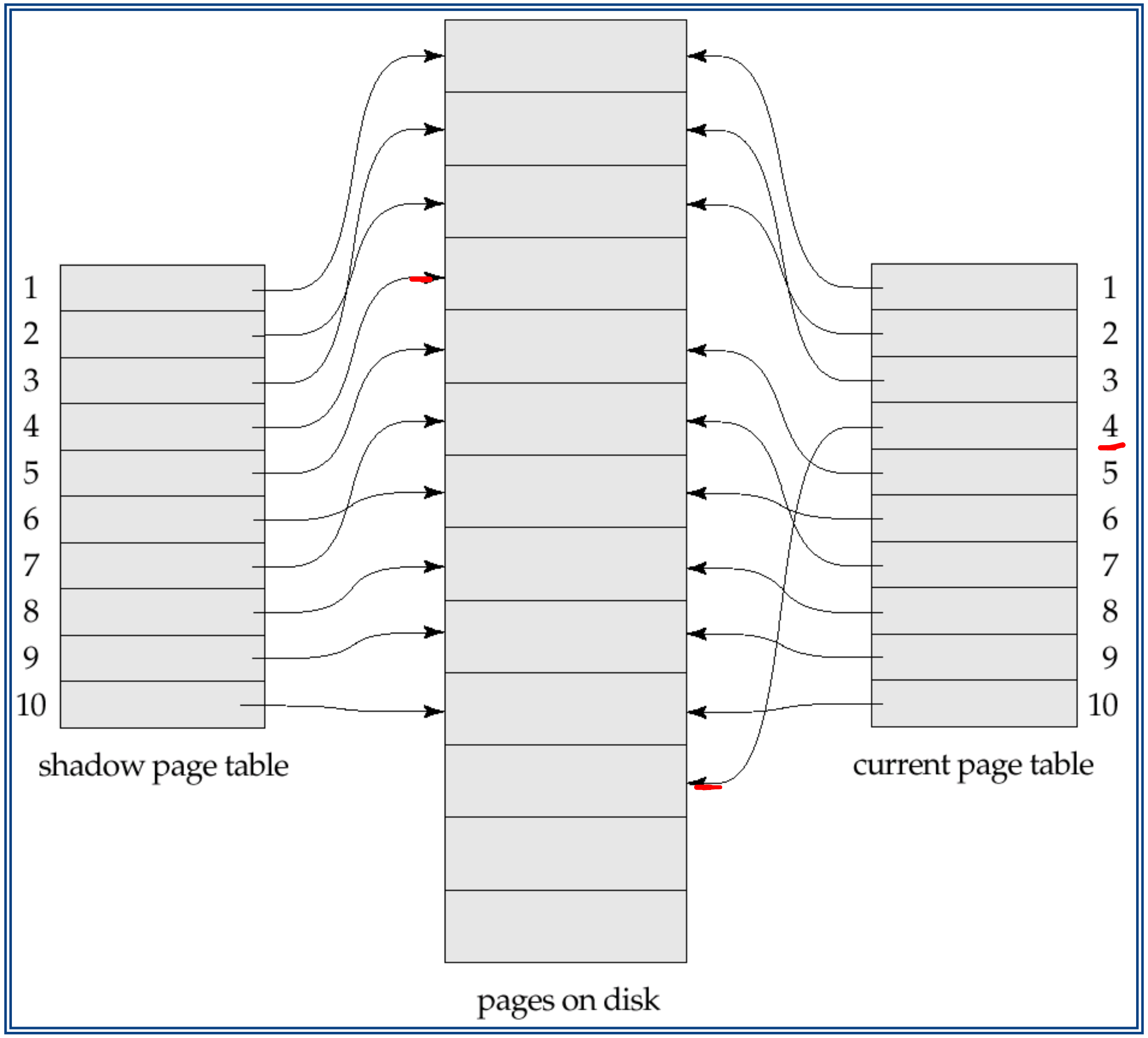
Whenever **any page is about to be written**:

- 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**

Sample Page Table



Example of Shadow Paging



Shadow and current page tables after write to page 4

Shadow Paging Contd.

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**, as follows:

– 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.

Shadow Paging Contd.

No recovery is needed after a crash — new transactions can start right away, using the shadow page table.

Advantages of shadow-paging over log-based schemes:

- No overhead of writing log records
- Recovery is trivial

Disadvantages:

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

Backups and crash recovery in practice

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
- Space requirement of the backups – depends on the resource
- Multiple past instances of backup – useful if point-in-time recovery is needed

Resource: https://learn.microsoft.com/en-us/sql/relational-databases/backup-restore/back-up-and-restore-of-sql-server-databases?view=sql-server-ver16