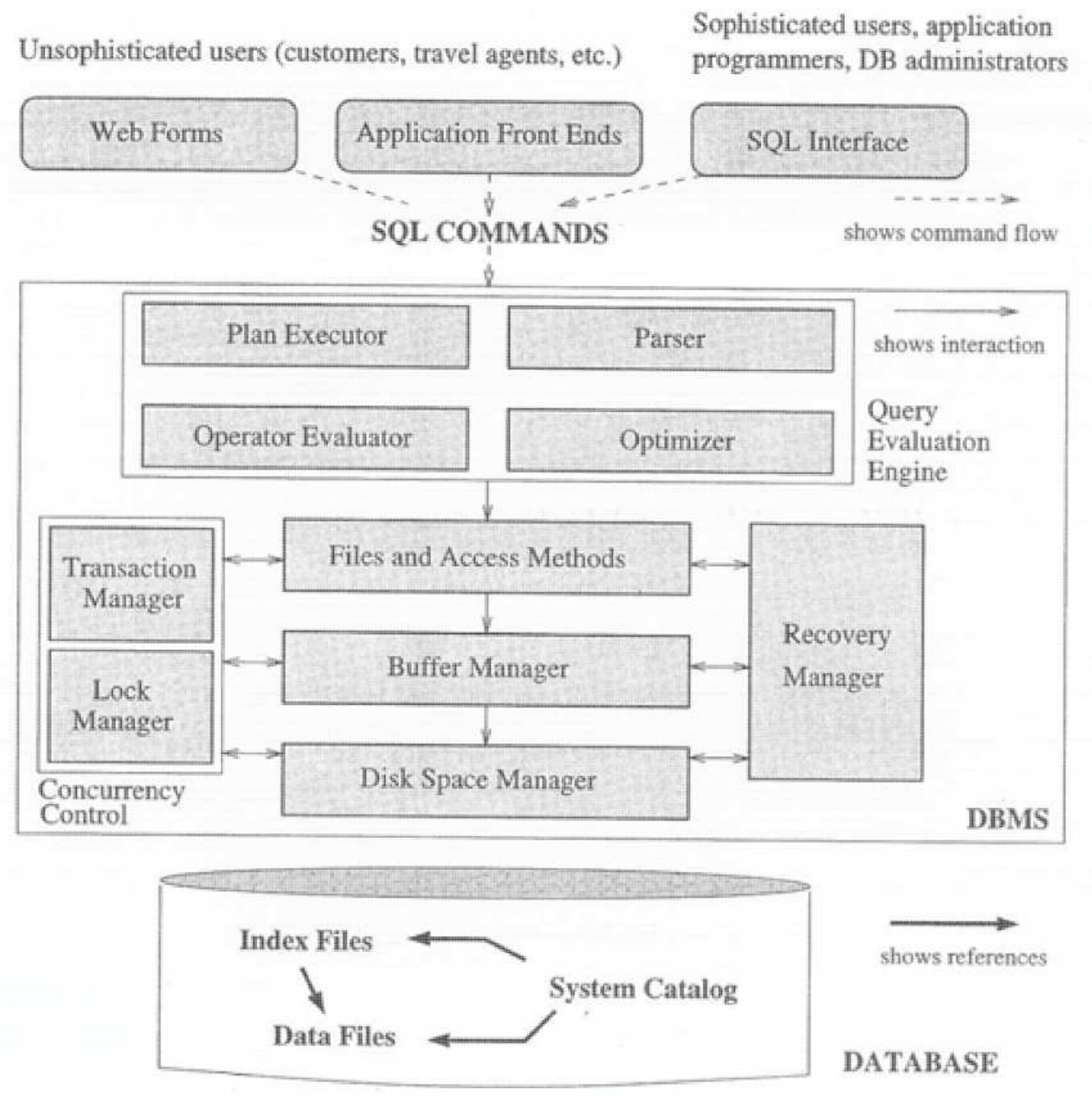
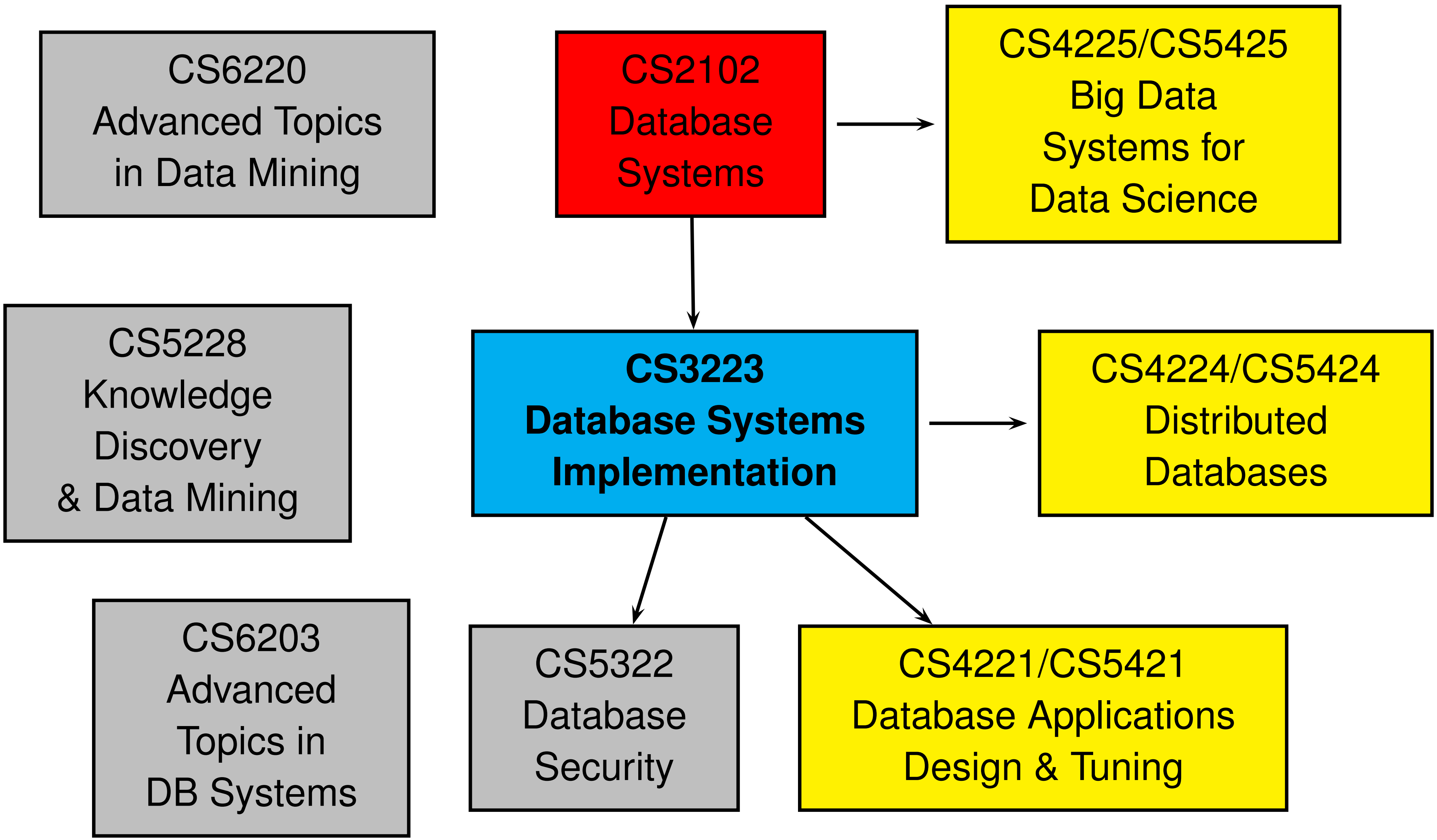
CS3223 Lecture 11
Review

Architecture of DBMS



(Ramakrishnan & Gehrke, Figure 1.3)

Database Courses @ SoC



Part 1: Review

Storage & Access Methods

- Buffer Manager
  - Pin / unpin interface
  - Replacement policies: Clock
- Storage Manager
  - Heap file: linked-list vs page directory
  - Page formats
    - fixed-length records: packed vs unpacked organization
    - variable-length records: slotted-page organization
  - Record formats: delimiters vs offsets for variable-length records
- Access Methods
  - Access patterns: sequential vs random I/O
  - B^+-tree
  - Linear hashing, Extendible hashing

Query Evaluation

- Query plan evaluation: materialized / pipelined
- Sorting - external merge sort
- Selection - table scan, index scan / intersection
  - primary conjuncts, covered conjuncts
- Projection
  - Sort-based, optimized sort-based,
  - hash-based
- Join
  - block nested loop, index nested loop,
  - sort-merge join, optimized sort-merge join,
  - Grace hash join
- Set operations & Aggregation
  - Sort-based, hash-based
- Possibility of exploiting index for operator evaluation

Query Optimization

- Search Space
  - linear plans
  - bushy plans
- Plan Enumeration
  - Query rewriting based on equivalence rules
  - Dynamic programming algorithms
    - Basic dynamic programming approach
    - Enhanced dynamic programming approach
- Cost Model
  - Size estimation assumptions: uniformity, independence, inclusion
  - Histograms: equiwidth, equidepth, equidepth with MCV

Transaction Management

- Atomicity, Consistency, Isolation, Durability
- Concurrency Control Manager: Isolation
  - Anomalies: dirty read, unrepeatable read, lost update, phantom read
- Recovery Manager: Atomicity, Durability
  - Deals with commit, abort, and restart operations
  - Recovery with before-images
  - Schedules: Strict $\subsetneq$ Cascadeless $\subsetneq$ Recoverable

Concurrency Control

- Theory
  - View / Multiversion View Serializability
  - Conflict Serializability
    - Conflicting actions, CSG
  - CSS $\subsetneq$ VSS $\subsetneq$ MVSS
- Protocols
  - **Lock-based**: Two-Phase Locking (2PL), Strict 2PL (S2PL)
    - lock modes, multigranularity locking
    - lock upgrades
    - deadlocks - detection, prevention
  - **Multiversion**: Snapshot Isolation (SI), Serializable SI
    - First-Committer-Wins Rule, First-Updater-Wins Rule
- Isolation Levels
  - Read uncommitted, Read committed, Repeatable read, Serializable
  - Lock-based protocols: Short/Long duration locks

Log-based Crash Recovery

- Interactions with buffer manager
  - Steal policy: requires undo recovery operations
  - No-force policy: requires redo operations
- Protocols
  - Write-ahead logging
  - Force-at-commit
- Checkpointing
  - Simple Checkpointing
  - Fuzzy Checkpointing
- ARIES
  - Three phases: Analysis, Redo, Undo
  - Data structures: transaction table & dirty page table
  - Concepts: CLRs, PageLSN, prevLSN chain, RedoLSN, undoNextLSN chain

Part 2: Q&A

Pipelined Query Evaluation

- Materialize intermediate result only when it's beneficial
- Query Q1: select * from R join S on R.a = S.a where R.c $<$ 50
  - Assume $|R| < |S|$
  - Consider evaluating Q1 using **Block Nested Loop Join** (i.e., R is outer relation)
  - No benefit to materialize $\sigma_{R.c<50}(R)$
- Query Q2: select * from R join S on R.a = S.a where R.c $<$ 50 and S.d = 100
  - Assume $|R| < |S|$
  - Consider evaluating Q2 using **Block Nested Loop Join** (i.e., R is outer relation)
  - No benefit to materialize $\sigma_{R.c<50}(R)$
  - May be beneficial to materialize $\sigma_{S.d=100}(S)$

Object Versions

- x_i denote that x_i is a version of x created by T_i
- Consider two versions of x: x_i & x_j, where $i < j$
  - Incorrect to conclude that x_i is an earlier version than x_j just because $i < j$!
- x_i is an earlier version than x_j iff T_i commits before T_j

SI: Transaction Dependencies

- Transaction dependencies are not the same as conflicting actions
- ww & wr dependencies must be non-concurrent
- rw dependencies could be non-concurrent or current
- $\mathbf{T_i} \overset{\mathbf{rw}}{\dashrightarrow} \mathbf{T_j}$: order of $R_i(x)$ and $W_j(x)$ does not matter as T_i & T_j are concurrent
- Example 1:

  T_1: $R_1(a)$, $R_1(x)$, $R_1(b)$, $Commit_1$

  T_2: $W_2(x)$, $Commit_2$
- Example 2:

  T_1: $R_1(a)$, $R_1(x)$, $R_1(b)$ $Commit_1$

  T_2: $W_2(x)$ $Commit_2$