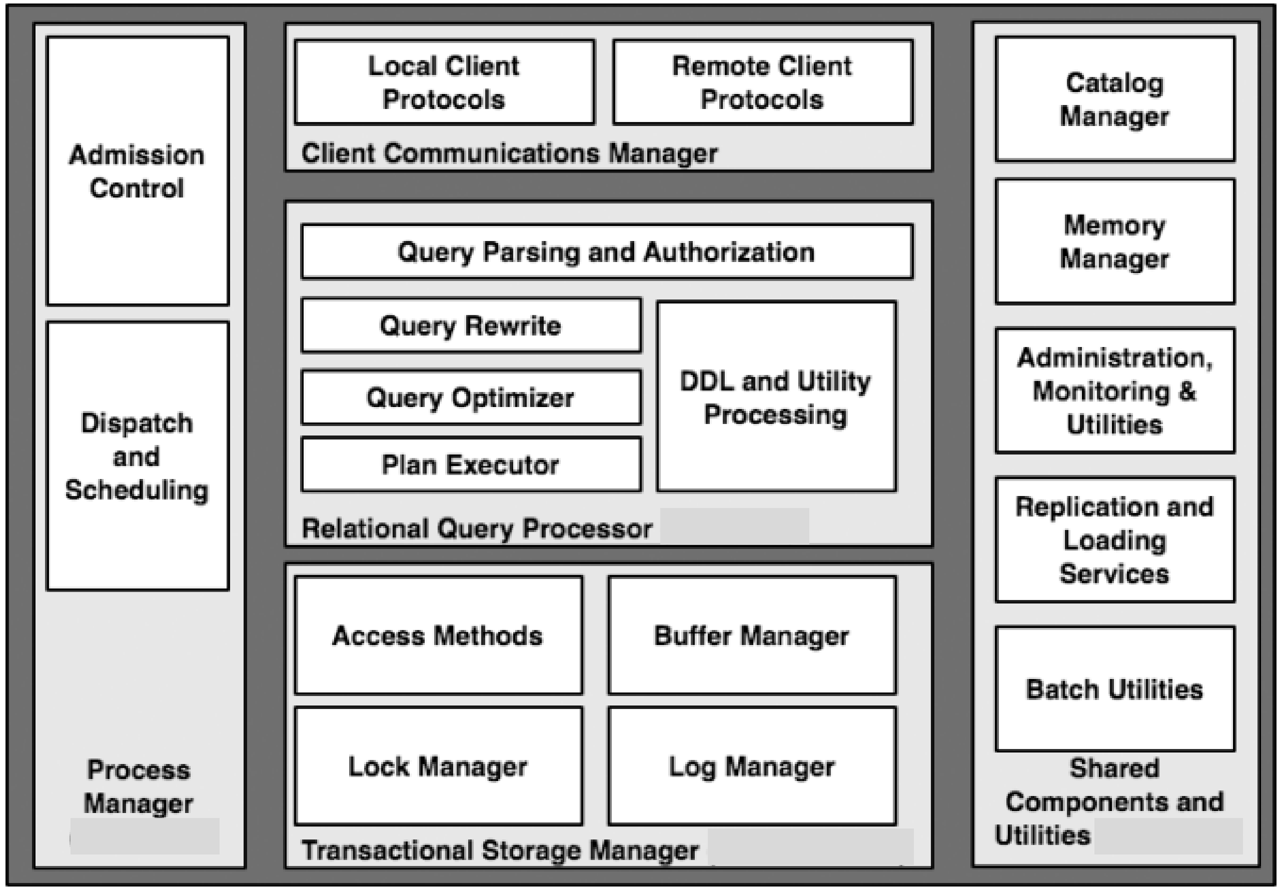
Advanced Databases

Query Processing

Foundations and Trends in Databases
J. M. Hellerstein, M. Stonebraker
and J. Hamilton
Vol. 1, No. 2 (2007) 141–259, 2007
DOI: 10.1561/1900000002

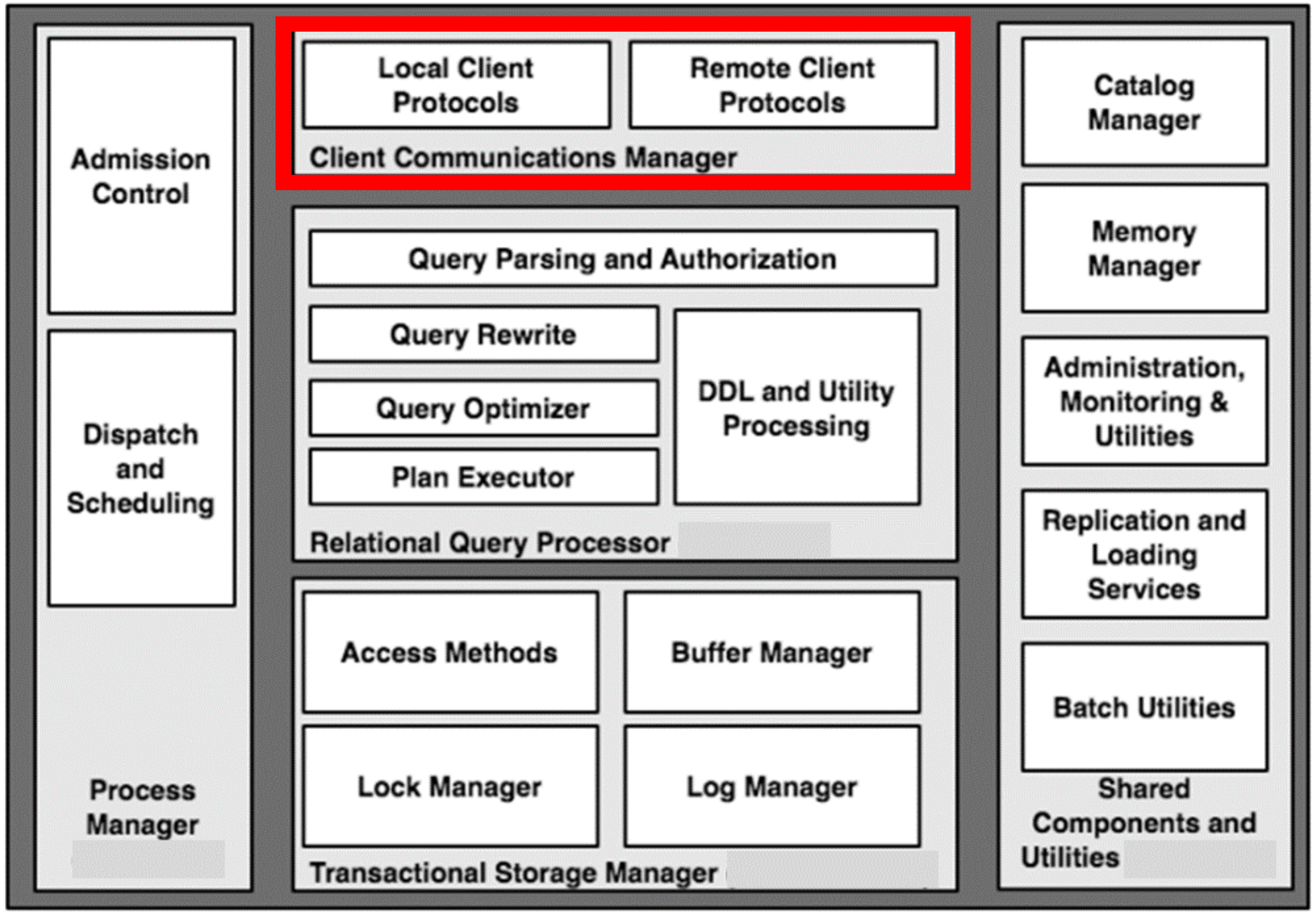
RDBMS Architecture

- Source: Foundations and Trends in Databases, J. M. Hellerstein, M. Stonebraker and J. Hamilton, Vol. 1, No. 2 (2007) 141–259, 2007, DOI: 10.1561/1900000002


Admission Control
Dispatch and Scheduling
Process Manager
Local Client Protocols
Remote Client Protocols
Client Communications Manager
Query Parsing and Authorization
Query Rewrite
Query Optimizer
Plan Executor
DDL and Utility Processing
Relational Query Processor
Access Methods
Buffer Manager
Lock Manager
Log Manager
Transactional Storage Manager
Catalog Manager
Memory Manager
Administration, Monitoring & Utilities
Replication and Loading Services
Batch Utilities
Shared Components and Utilities

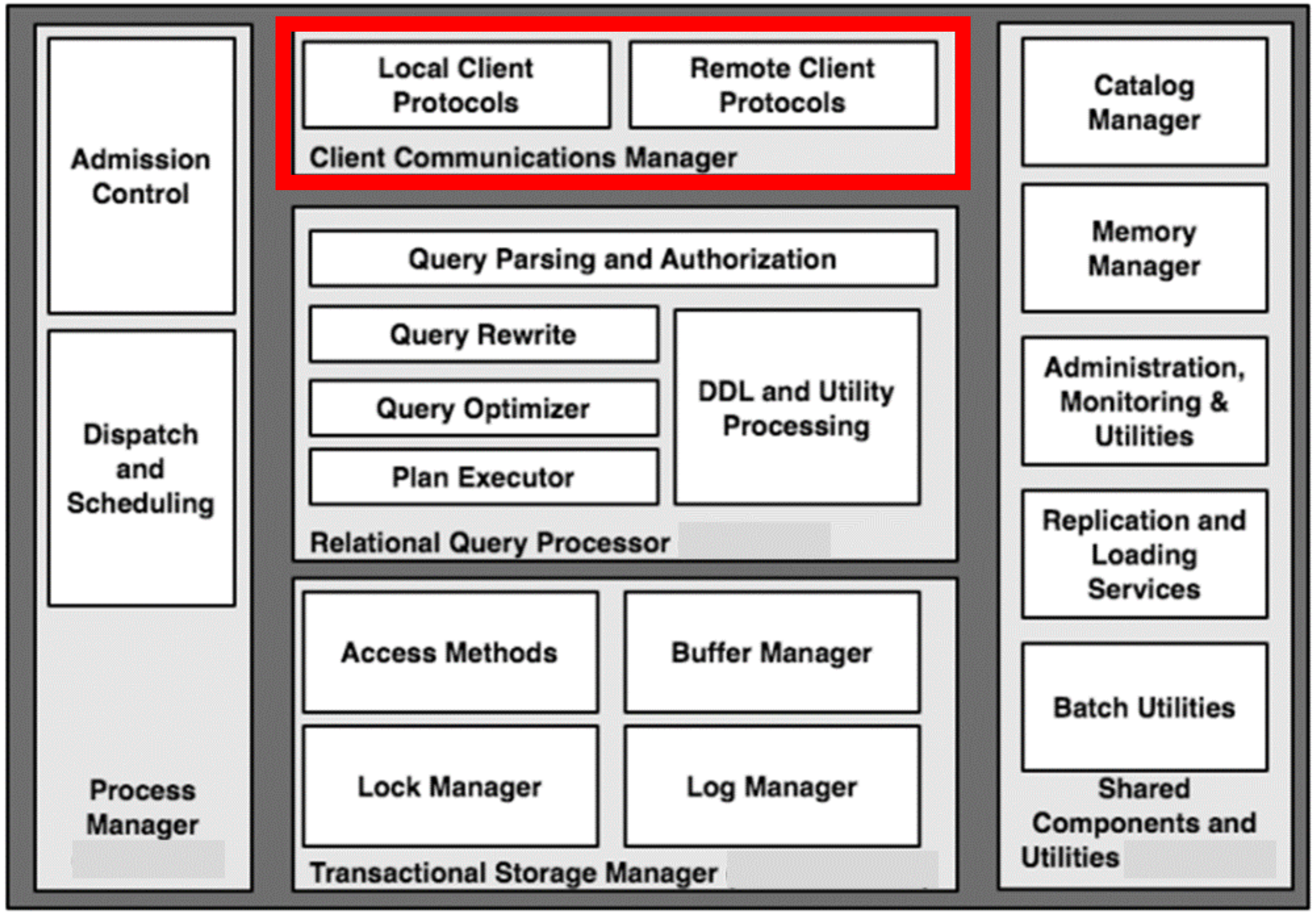
The Life of a Query

- Application (the client) calls an API
- API establishes a connection with the *Client Communications Manager* of a DBMS
  - Can be established between the client and the database server directly
    - e.g. via ODBC or JDBC (connectivity protocols)
    - “client-server”
  - Or the client may communicate with a “middle-tier server” (e.g. a web server, transaction processing monitor),
    - This may uses a protocol to proxy the communication between the client and the DBMS
    - “three-tier”
  - Or there may be another layer between the webserver and DBMS – an application server
    - “n-tier”


Admission Control
Dispatch and Scheduling
Process Manager
Local Client Protocols
Remote Client Protocols
Client Communications Manager
Query Parsing and Authorization
Query Rewrite
Query Optimizer
Plan Executor
DDL and Utility Processing
Relational Query Processor
Access Methods
Buffer Manager
Lock Manager
Log Manager
Transactional Storage Manager
Catalog Manager
Memory Manager
Administration, Monitoring & Utilities
Replication and Loading Services
Batch Utilities
Shared Components and Utilities

The Life of a Query

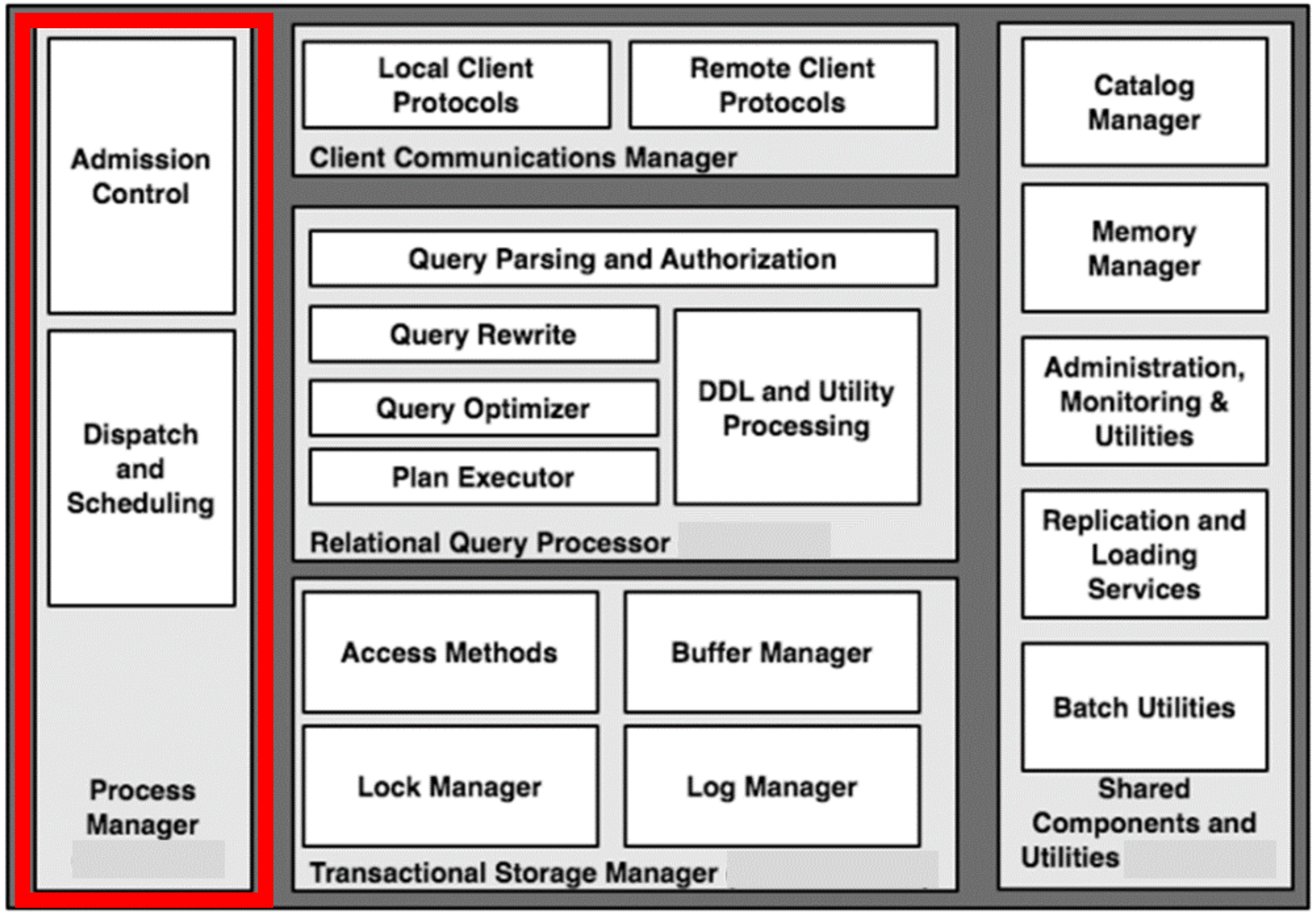
- DBMS needs to work with a variety of connectivity protocols
- Fundamentally, the client communications manager
  - Establishes and remembers the connection state for the caller (be it a client or a middleware server)
  - Responds to SQL commands from the caller, and to return both data and control messages (result codes, errors, etc.)


Admission Control
Dispatch and Scheduling
Process Manager
Local Client Protocols
Remote Client Protocols
Client Communications Manager
Query Parsing and Authorization
Query Rewrite
Query Optimizer
Plan Executor
DDL and Utility Processing
Relational Query Processor
Access Methods
Buffer Manager
Lock Manager
Log Manager
Transactional Storage Manager
Catalog Manager
Memory Manager
Administration, Monitoring & Utilities
Replication and Loading Services
Batch Utilities
Shared Components and Utilities

The Life of a Query

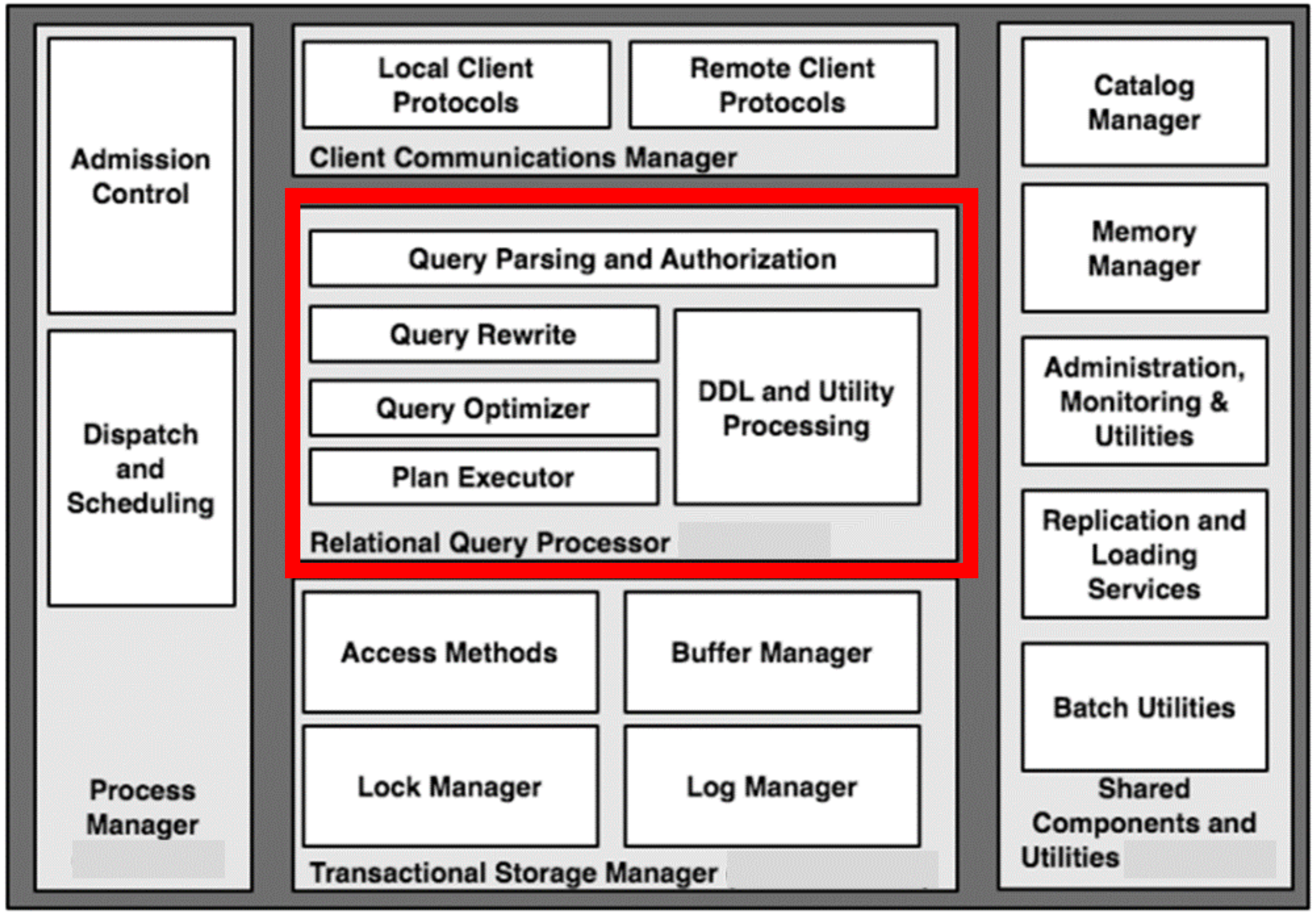
- Once a command has been received, the DBMS *Process Manager* takes responsibility for
  - assigning a "thread of computation" to the command
  - ensuring that the thread's data and control outputs are connected to the client via the communications manager
- When a command is received the process manager must decide on *admission control* and whether
  - the system should begin processing the query immediately

  or
  - defer execution until a time when enough system resources are available to devote to this query


Admission Control
Dispatch and Scheduling
Process Manager
Local Client Protocols
Remote Client Protocols
Client Communications Manager
Query Parsing and Authorization
Query Rewrite
Query Optimizer
Plan Executor
DDL and Utility Processing
Relational Query Processor
Access Methods
Buffer Manager
Lock Manager
Log Manager
Transactional Storage Manager
Catalog Manager
Memory Manager
Administration, Monitoring & Utilities
Replication and Loading Services
Batch Utilities
Shared Components and Utilities

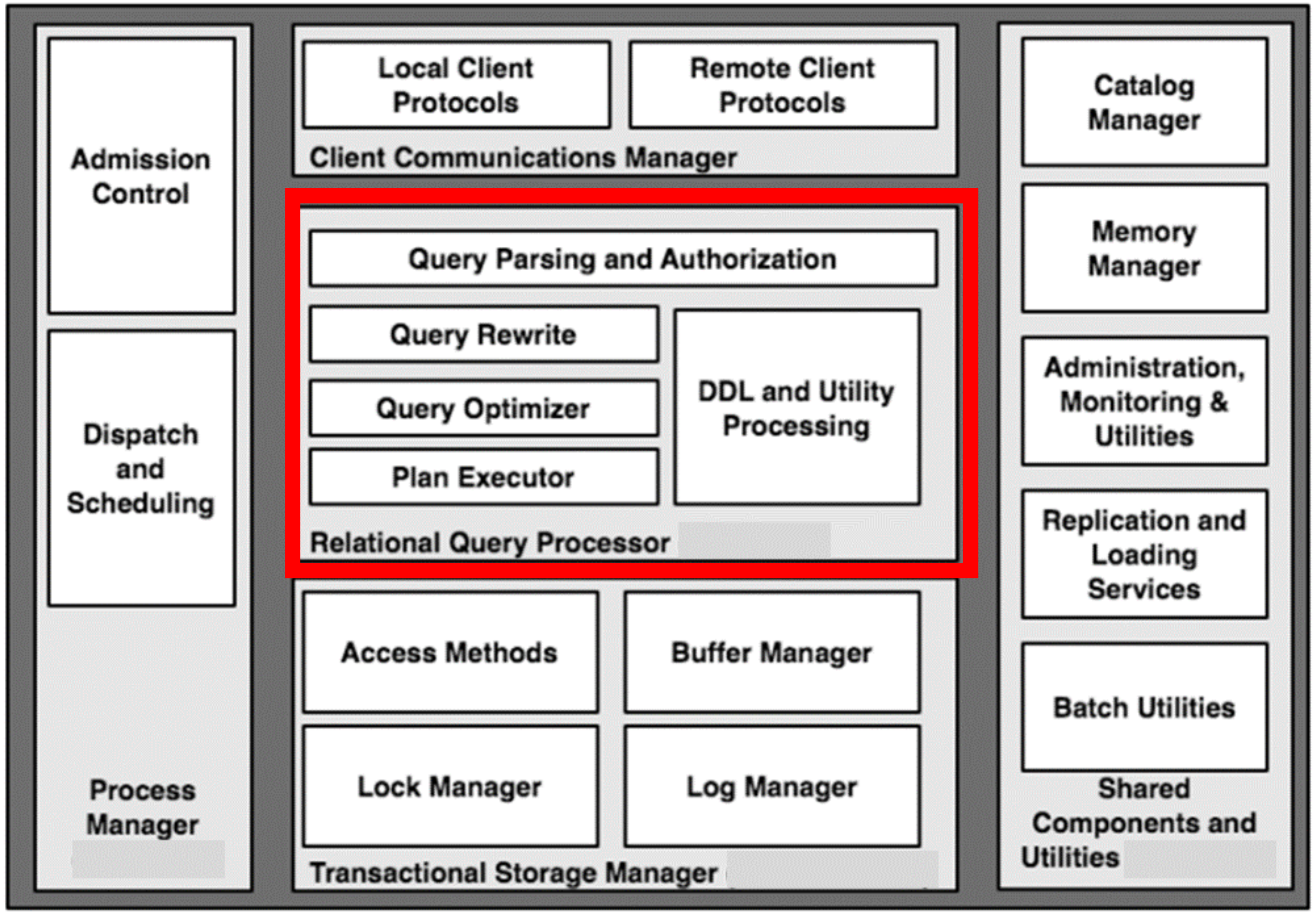
The Life of a Query

- Once admitted and allocated as a thread of control, the query can begin to execute
- The code is invoked in the *Relational Query Processor* which has a set of modules which
  - Checks that the user is authorized to run the query,
  - Compiles the user's SQL query text into and internal query plan
  - Executes the resulting compiled query plan (handled via the *plan executor*)


Admission Control
Dispatch and Scheduling
Process Manager
Local Client Protocols
Remote Client Protocols
Client Communications Manager
Query Parsing and Authorization
Query Rewrite
Query Optimizer
Plan Executor
DDL and Utility Processing
Relational Query Processor
Access Methods
Buffer Manager
Lock Manager
Log Manager
Transactional Storage Manager
Catalog Manager
Memory Manager
Administration, Monitoring & Utilities
Replication and Loading Services
Batch Utilities
Shared Components and Utilities

The Life of a Query

- The plan executor consists of:
  - A suite of “operators” (relational algorithm implementations) for executing any query
    - Typical operators implement relational query processing tasks including joins, selection, projection, aggregation, sorting etc
    - Plus calls to request data records from lower layers of the system.


Admission Control
Dispatch and Scheduling
Process Manager
Local Client Protocols
Remote Client Protocols
Client Communications Manager
Query Parsing and Authorization
Query Rewrite
Query Optimizer
Plan Executor
DDL and Utility Processing
Relational Query Processor
Access Methods
Buffer Manager
Lock Manager
Log Manager
Transactional Storage Manager
Catalog Manager
Memory Manager
Administration, Monitoring & Utilities
Replication and Loading Services
Batch Utilities
Shared Components and Utilities

Quick Question

Projection

VS

Selection

What is the difference?

Quick Question

Projection

VS

Selection

If used together, which is performed first in RDBMS?

Simple Example

- Suppose we have a simple database table of Employees with two attributes ID (unique) and name (non-unique):

ID	NAME
1	Fred
2	Joe
3	Jane
4	Mary
5	Paulette
6	Pierre

Simple Example

- Given the current data, these two queries will return the same result:

Query 1:

```
SELECT *
FROM employee
WHERE name='Joe';
```

Query 2:

```
SELECT *
FROM employee
WHERE ID=2;
```

ID	NAME
1	Fred
2	Joe
3	Jane
4	Mary
5	Paulette
6	Pierre

Simple Example

- But they may have different query plans since ID is unique and name is non-unique
- Query 1
  - This will use a *sequential scan* which means all rows of the database will be checked
    - If the name Joe is found it will be added to the resultset
    - As more than one row can have the value Joe for name all rows have to be checked to ensure all data is found

ID	NAME
1	Fred
2	Joe
3	Jane
4	Mary
5	Paulette
6	Pierre

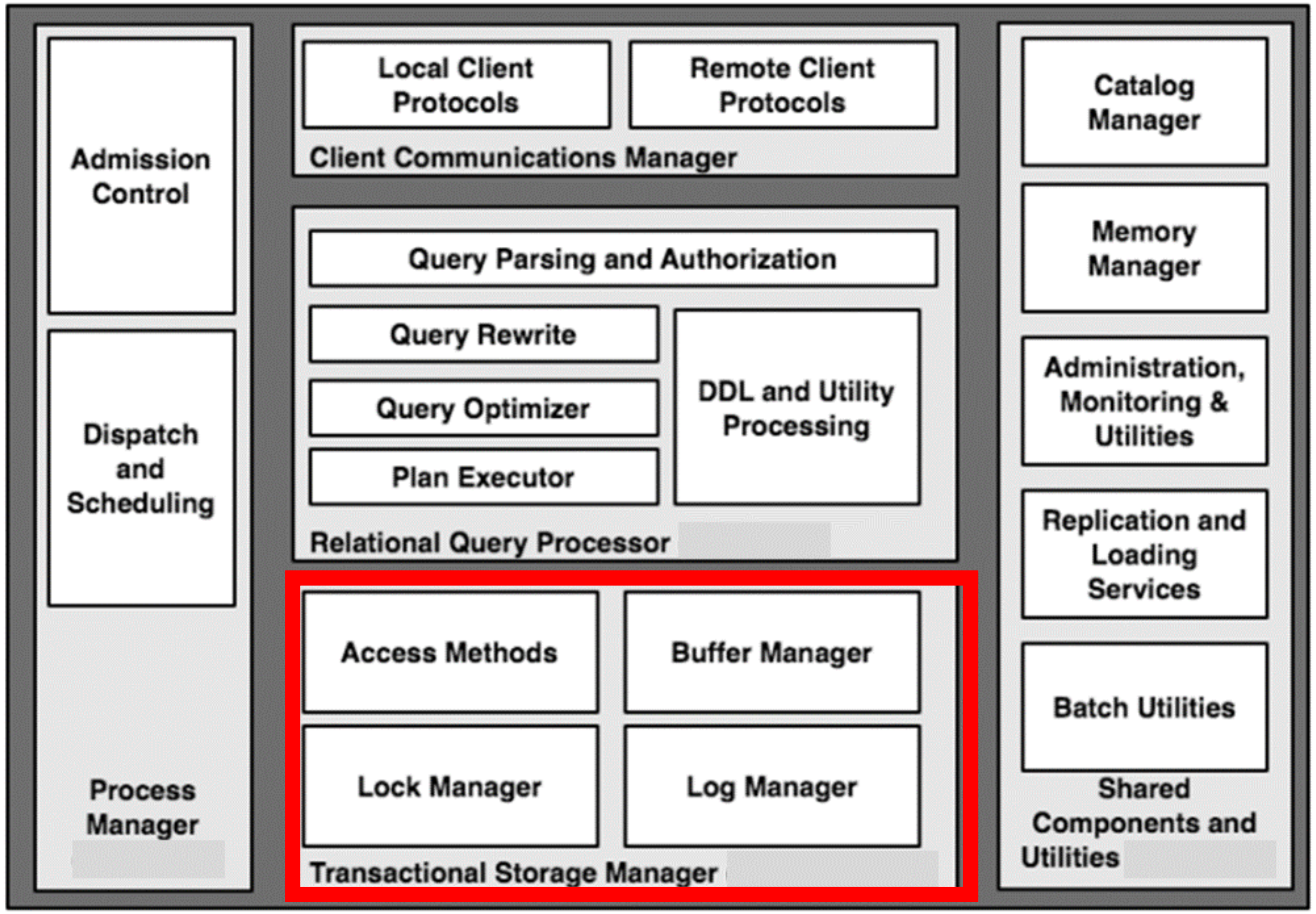
Simple Example

- But they may have different query plans since ID is unique and name is non-unique
- Query 2
  - This will use a *sequential seek*
  - It will sequentially go through each row to check if the ID is 2 but it will stop once it has found an entry since it knows IDs are unique

ID	NAME
1	Fred
2	Joe
3	Jane
4	Mary
5	Paulette
6	Pierre

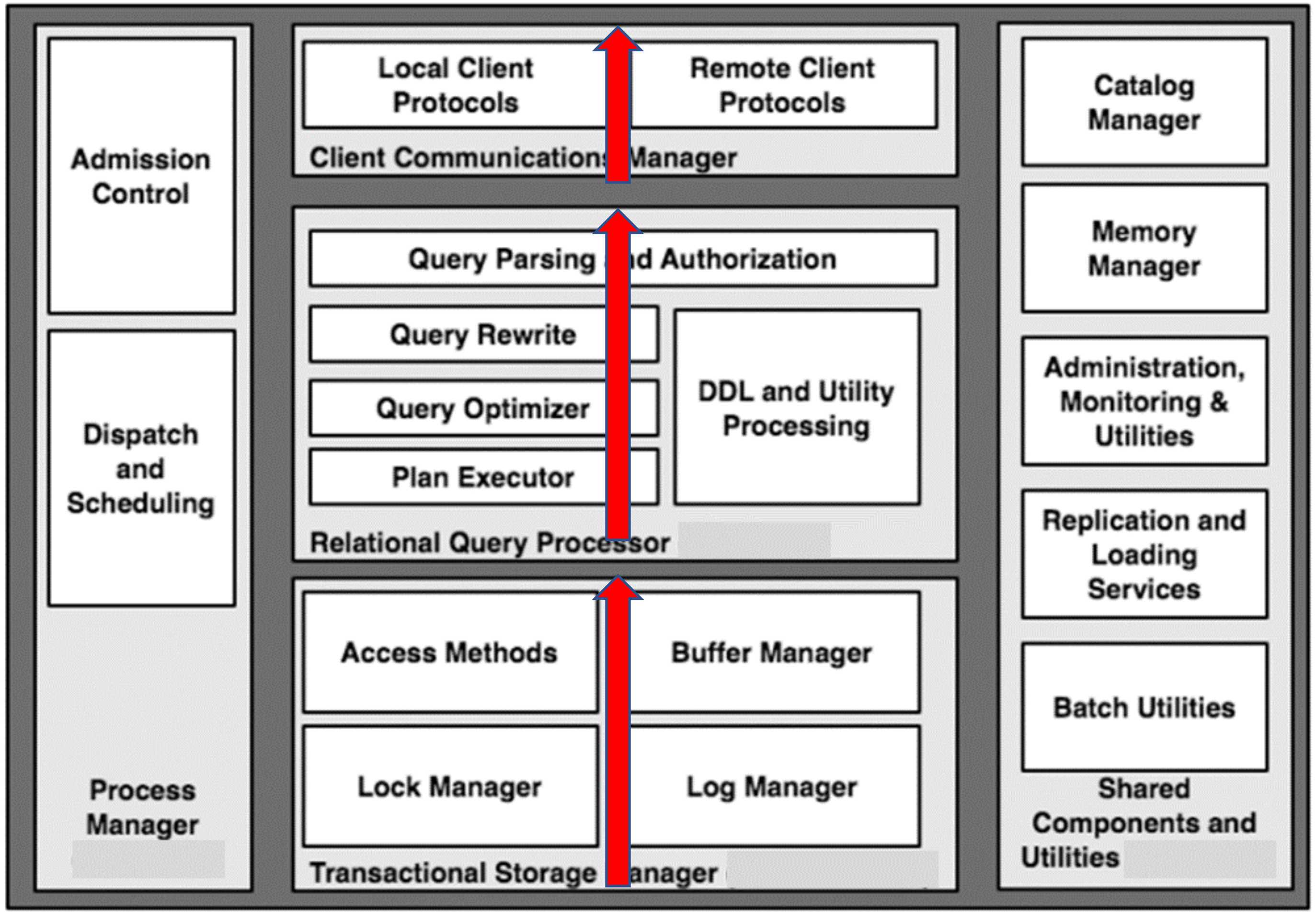
The Life of a Query

- In the query plan, one or more operators request data from the database
- This requires making fetch calls to the DBMS *Transactional Storage Manager*
  - Responsible for all data access (read) and manipulation (create, update, delete) calls
  - Includes algorithms and data structures for organizing and accessing data on disk ("access methods")
  - Includes basic structures like tables and indexes
  - Includes a buffer management module that decides when and what data to transfer between disk and memory buffers
  - This is where our ACID properties are ensured
    - Lock manager
    - Log manager


Admission Control
Dispatch and Scheduling
Process Manager
Local Client Protocols
Remote Client Protocols
Client Communications Manager
Query Parsing and Authorization
Query Rewrite
Query Optimizer
Plan Executor
DDL and Utility Processing
Relational Query Processor
Access Methods
Buffer Manager
Lock Manager
Log Manager
Transactional Storage Manager
Catalog Manager
Memory Manager
Administration, Monitoring & Utilities
Replication and Loading Services
Batch Utilities
Shared Components and Utilities

The Life of a Query

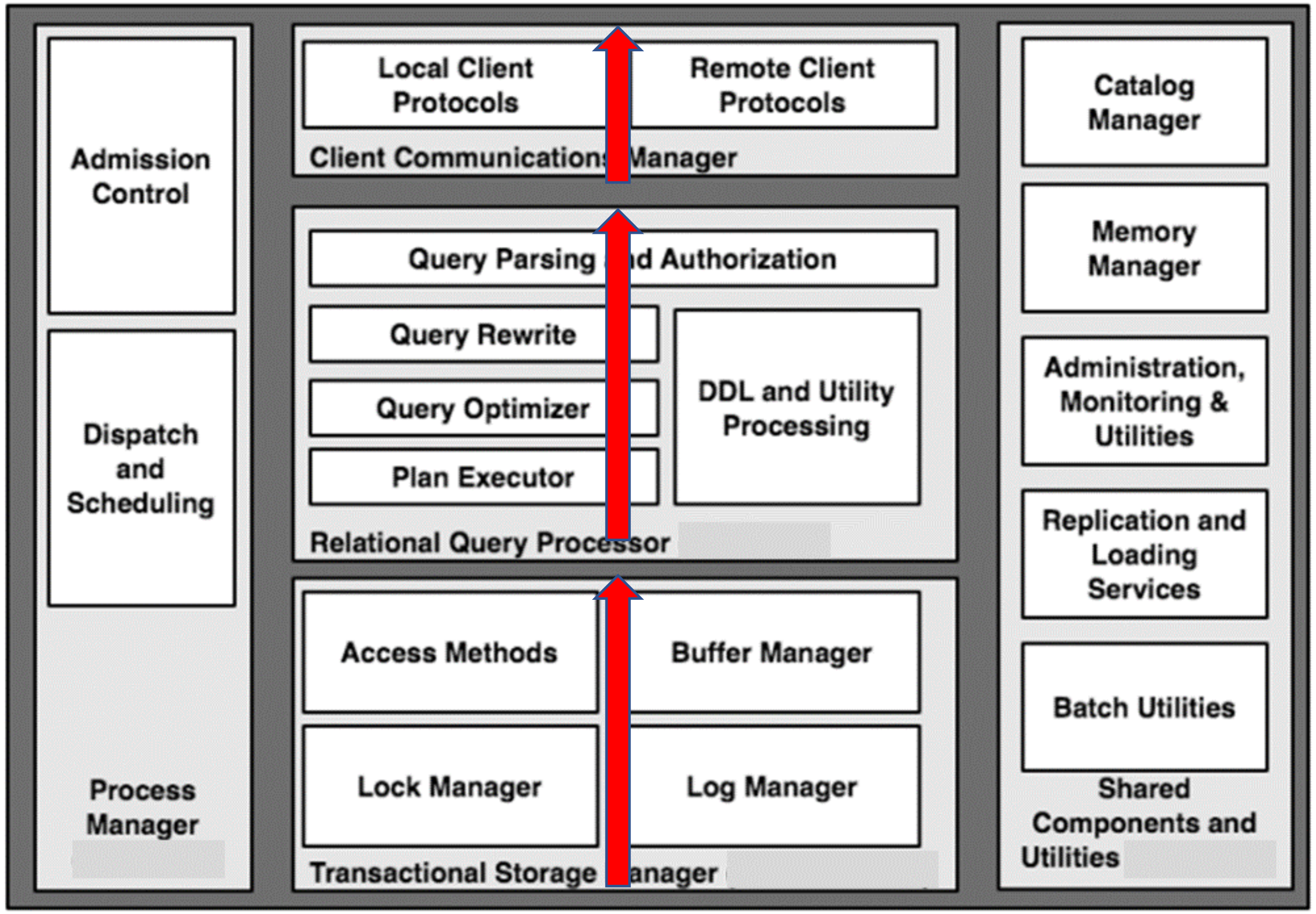
- At this point we should have some data to answer our query so we start “unwinding the stack”:
  - Access methods return control to the query executor’s operators
  - Query executor operators compute a set of tuples (resultset) from the database data
  - Once generated, tuples are placed in a buffer for the client communications manager
  - Then the client communications manager ships the results back to the caller



Admission Control
Dispatch and Scheduling
Process Manager
Local Client Protocols
Remote Client Protocols
Client Communications Manager
Query Parsing and Authorization
Query Rewrite
Query Optimizer
Plan Executor
DDL and Utility Processing
Relational Query Processor
Access Methods
Buffer Manager
Lock Manager
Log Manager
Transactional Storage Manager
Catalog Manager
Memory Manager
Administration, Monitoring & Utilities
Replication and Loading Services
Batch Utilities
Shared Components and Utilities

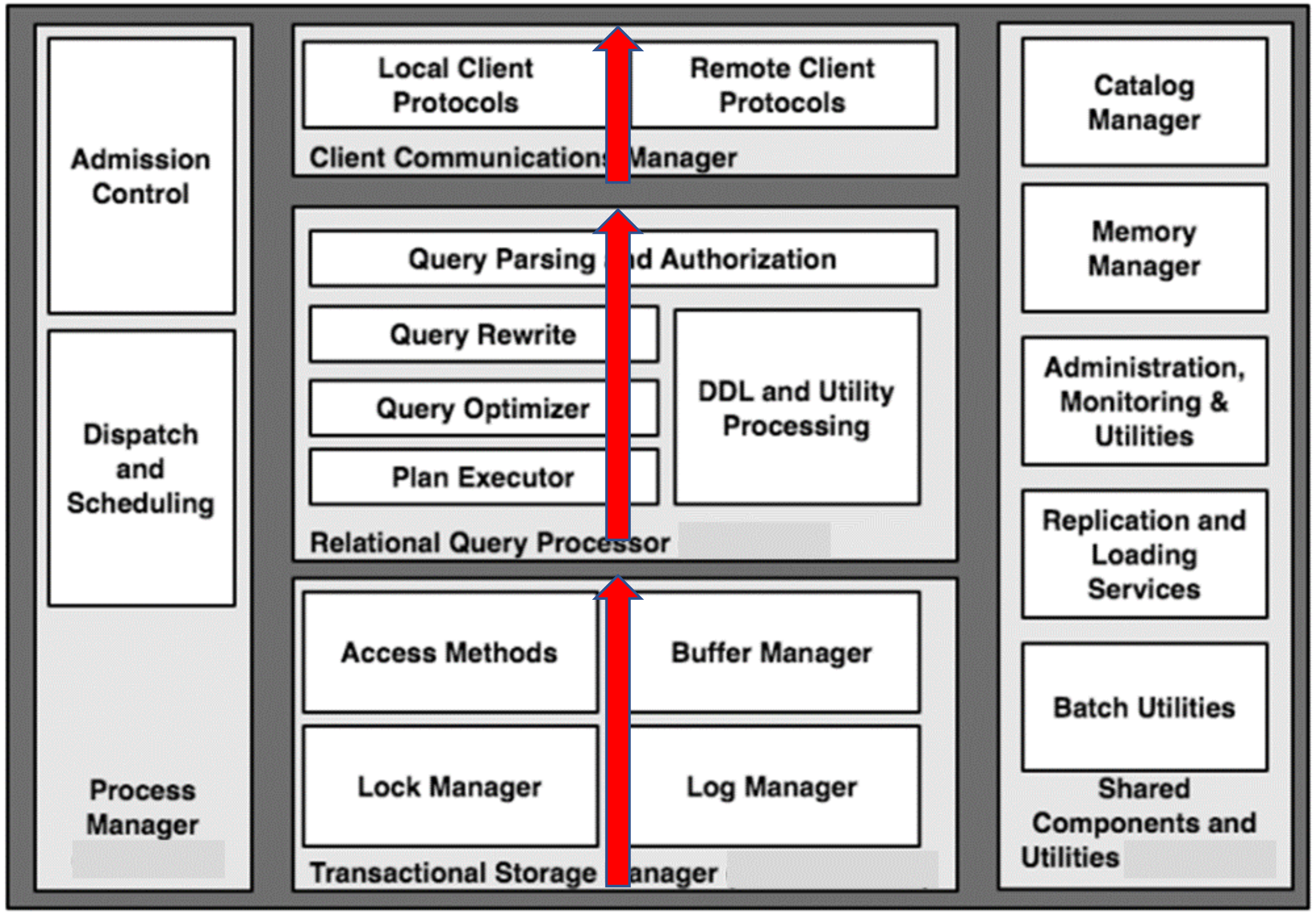
The Life of a Query

- When the transaction is completed and the connection is closed
  - the transaction manager cleans up state for the transaction
  - the process manager frees any control structures for the query
  - the communications manager cleans up communication state for the connection


Admission Control
Dispatch and Scheduling
Process Manager
Local Client Protocols
Remote Client Protocols
Client Communications Manager
Query Parsing and Authorization
Query Rewrite
Query Optimizer
Plan Executor
DDL and Utility Processing
Relational Query Processor
Access Methods
Buffer Manager
Lock Manager
Log Manager
Transactional Storage Manager
Catalog Manager
Memory Manager
Administration, Monitoring & Utilities
Replication and Loading Services
Batch Utilities
Shared Components and Utilities

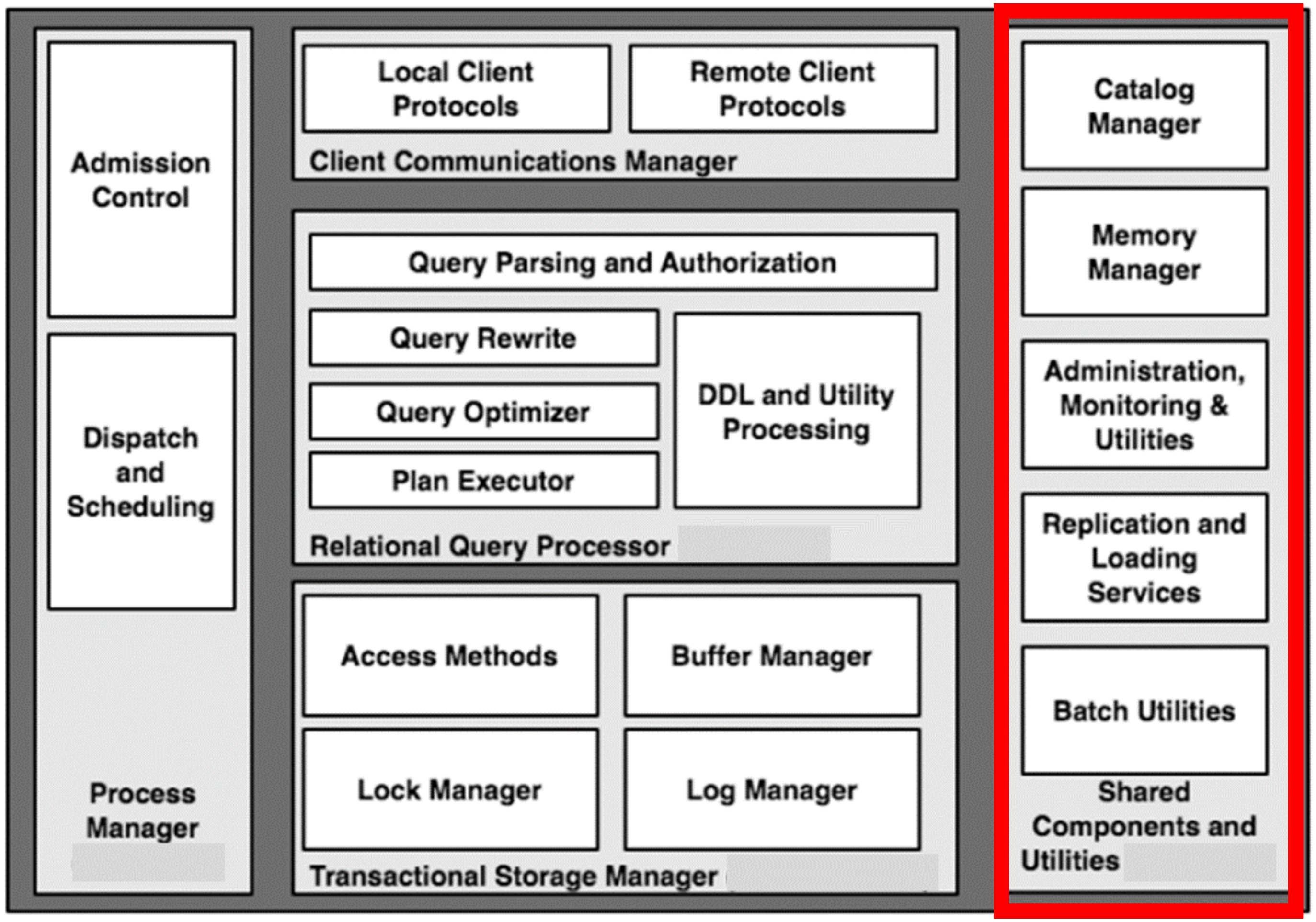
The Life of a Query

- For large result sets, the client typically will make additional calls to fetch more data incrementally from the query
- This requires multiple iterations through the communications manager, query executor, and storage manager interactions


Admission Control
Dispatch and Scheduling
Process Manager
Local Client Protocols
Remote Client Protocols
Client Communications Manager
Query Parsing and Authorization
Query Rewrite
Query Optimizer
Plan Executor
DDL and Utility Processing
Relational Query Processor
Access Methods
Buffer Manager
Lock Manager
Log Manager
Transactional Storage Manager
Catalog Manager
Memory Manager
Administration, Monitoring & Utilities
Replication and Loading Services
Batch Utilities
Shared Components and Utilities

The Life of a Query
Shared Components

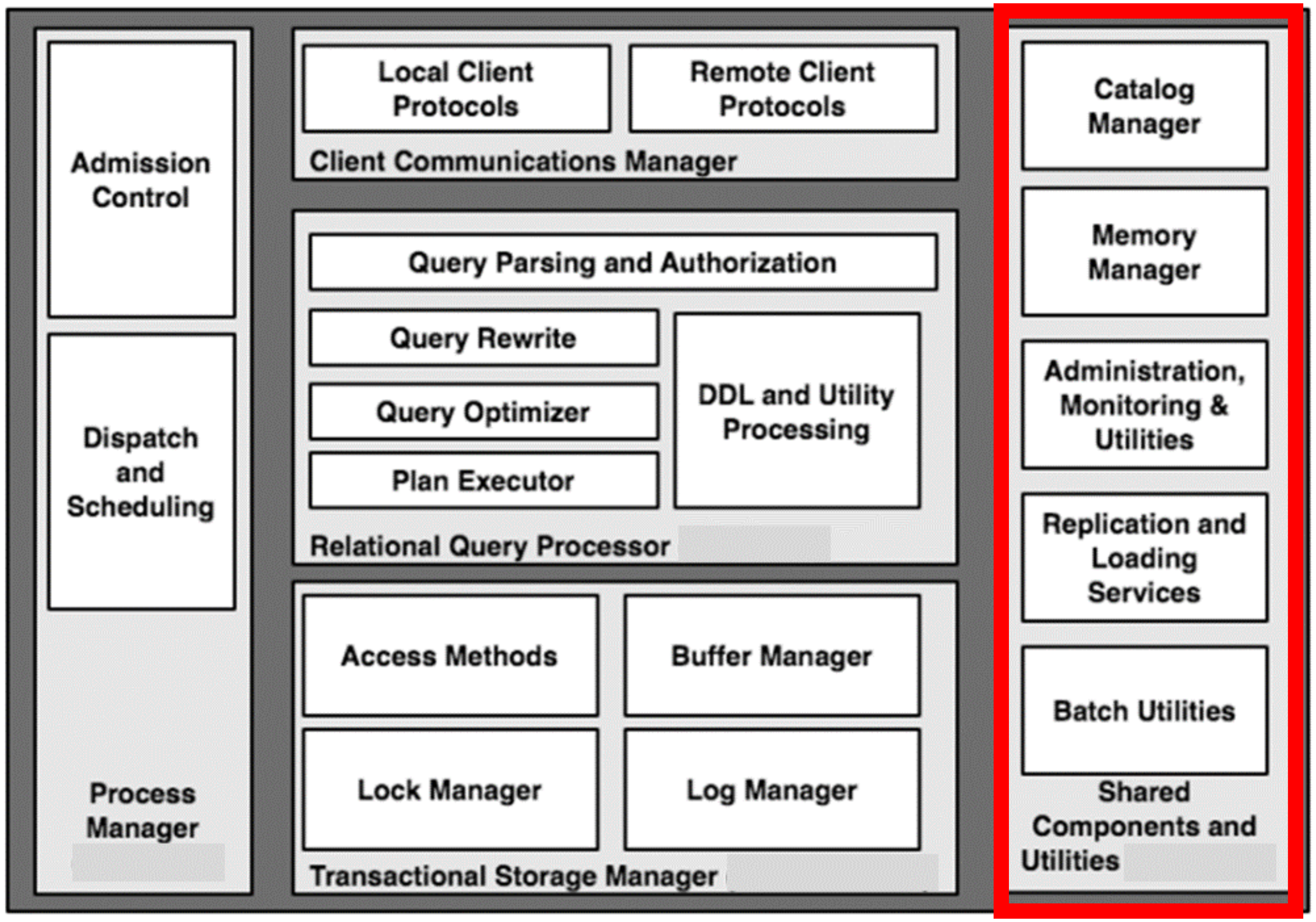
- Catalog Manager and Memory Manager
  - Invoked as utilities during any transaction
- Catalog is used by the query processor during authentication, parsing, and query optimization
- Memory manager is used throughout the DBMS whenever memory needs to be dynamically allocated or deallocated


Admission Control
Dispatch and Scheduling
Process Manager
Local Client Protocols
Remote Client Protocols
Client Communications Manager
Query Parsing and Authorization
Query Rewrite
Query Optimizer
Plan Executor
DDL and Utility Processing
Relational Query Processor
Access Methods
Buffer Manager
Lock Manager
Log Manager
Transactional Storage Manager
Catalog Manager
Memory Manager
Administration, Monitoring & Utilities
Replication and Loading Services
Batch Utilities
Shared Components and Utilities

The Life of a Query
Shared Components

- Administration, Monitoring and Utilities
- Replication and Loading Services
- Batch Services

- All run independently of a query – focus is on keeping the database as a whole reliable


Admission Control
Dispatch and Scheduling
Process Manager
Local Client Protocols
Remote Client Protocols
Client Communications Manager
Query Parsing and Authorization
Query Rewrite
Query Optimizer
Plan Executor
DDL and Utility Processing
Relational Query Processor
Access Methods
Buffer Manager
Lock Manager
Log Manager
Transactional Storage Manager
Catalog Manager
Memory Manager
Administration, Monitoring & Utilities
Replication and Loading Services
Batch Utilities
Shared Components and Utilities

Database Performance Tuning

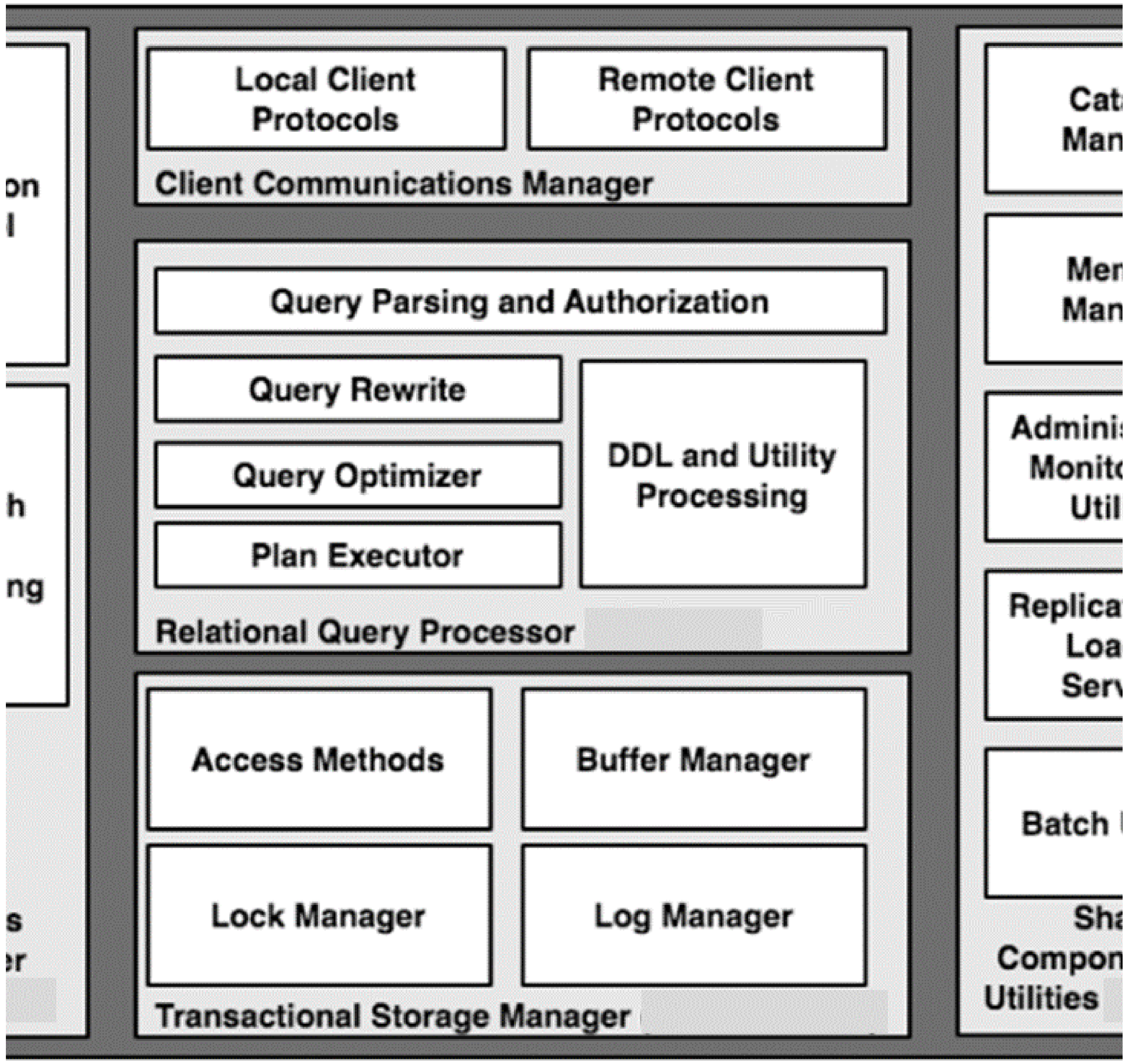
- Objective of tuning is to minimize the response time of your queries by making the best use of your system resources.

Query Processing

- Process of converting high level queries to low level expressions to extract the data from physical level file system like databases
- A relational query processor
  - Takes in a declarative SQL statement
  - Validates it
  - Optimizes it into a procedural dataflow execution plan
  - and executes that dataflow program on behalf of a client program (subject to admission control)

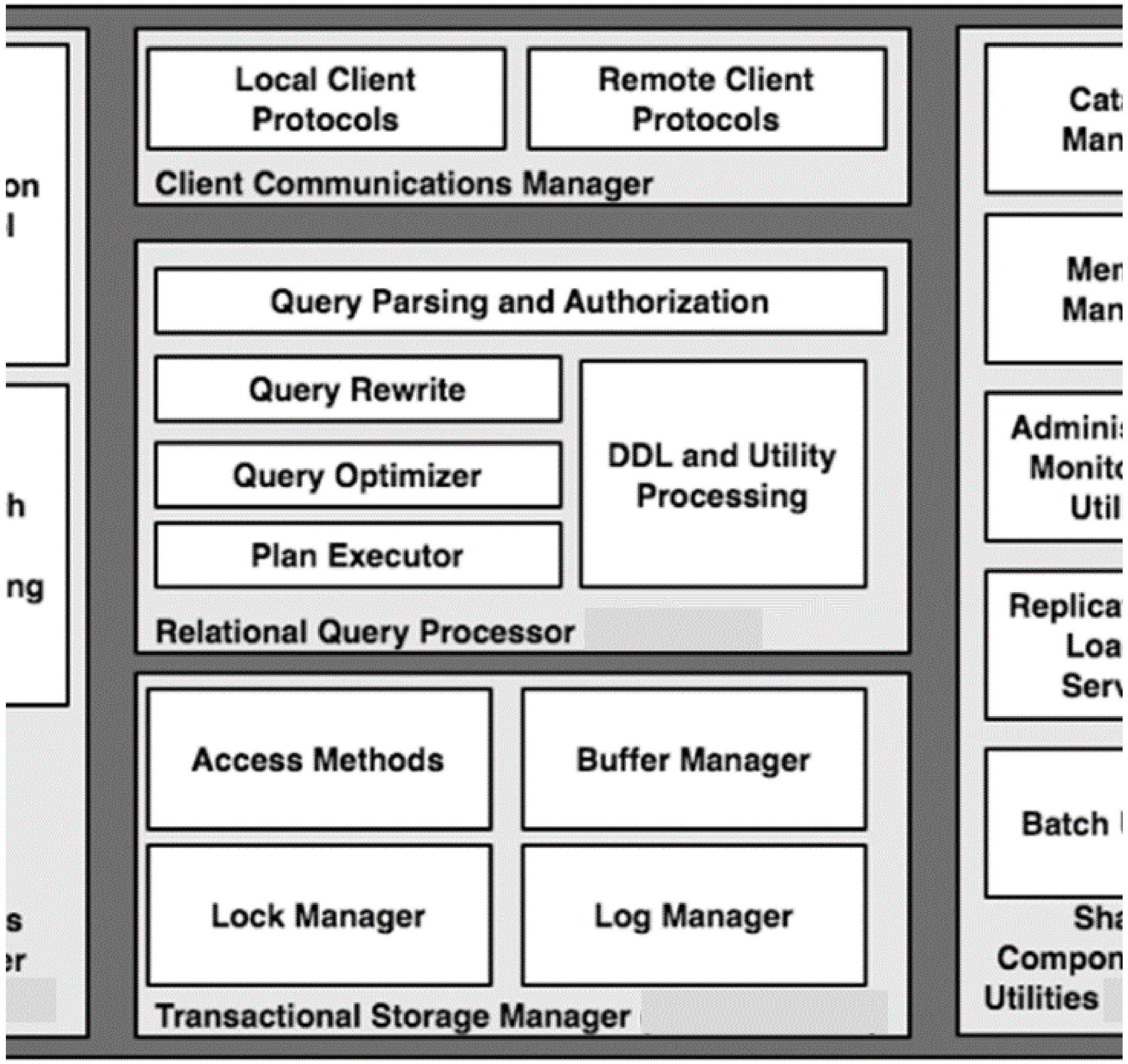
Query Processing

- Treated as a single-user, single-threaded task (concurrency control etc is handled elsewhere)
- Parsing and translation
- Optimisation
- Evaluation
- Execution



Query Processing

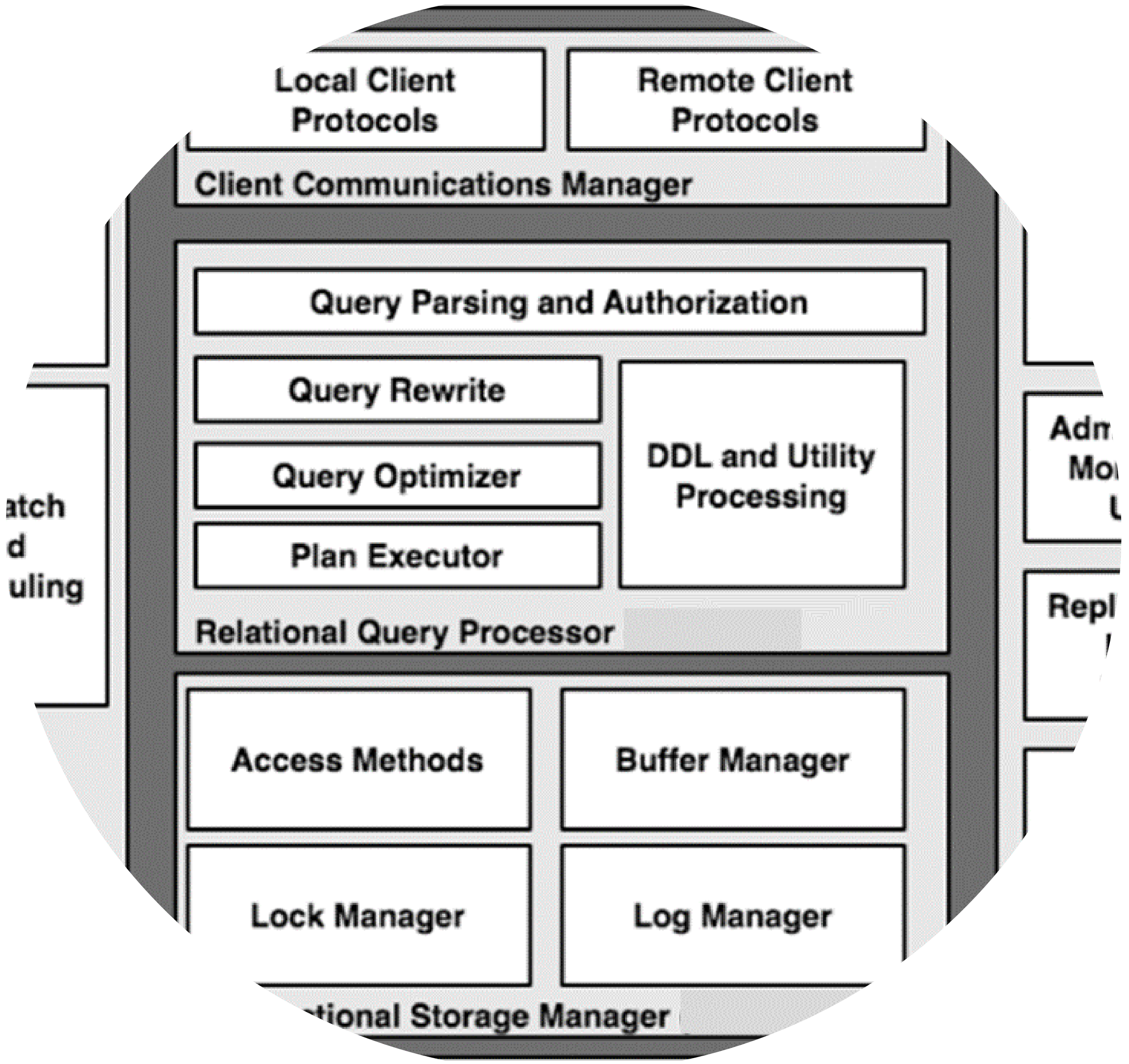
- Focus is on DML
  - INSERT, UPDATE, DELETE, SELECT
- DDL is not handled by the optimizer
  - Implemented procedurally in static DBMS logic through explicit calls to the transactional storage engine and catalog manager



Local Client Protocols
Remote Client Protocols
Client Communications Manager
Query Parsing and Authorization
Query Rewrite
Query Optimizer
Plan Executor
DDL and Utility Processing
Relational Query Processor
Access Methods
Buffer Manager
Lock Manager
Log Manager
Transactional Storage Manager

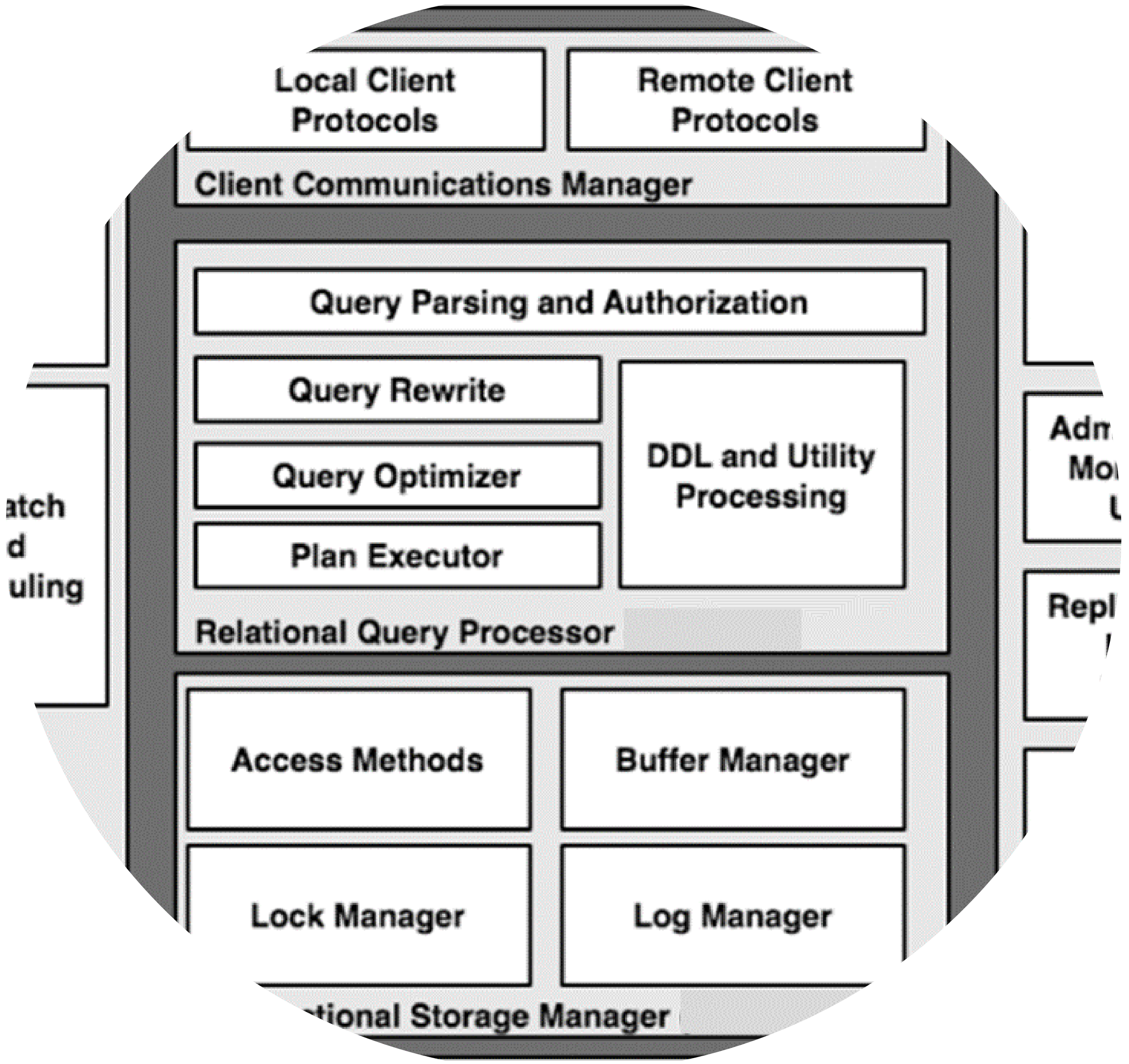
Query Parsing

- Given an SQL statement the SQL Parser will
  1. checks that the query is correctly specified
  2. resolves names and references
  3. converts the query into the internal format used by the Query Optimizer
  4. verifies that the user is authorized to execute the query.
- Note:
  - Some DBMSs defer some or all security checking to execution time but the parser is still responsible for gathering the data needed for the execution-time security check



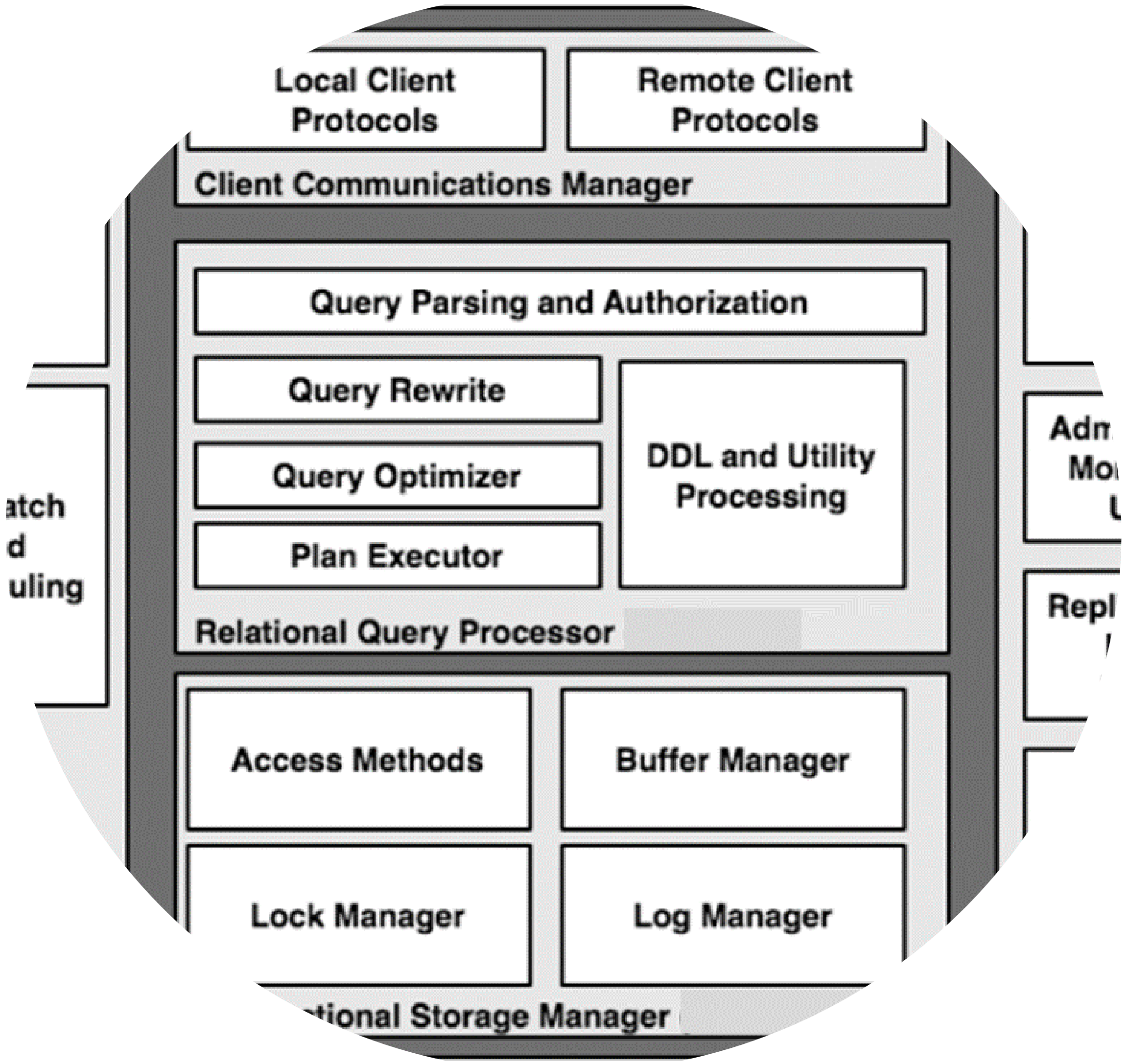
Query Parsing

- The parser first considers each of the tables referenced in the FROM clause.
- It converts all table names used into the fully qualified name
  - Form used is server.database.schema.table (called *four part name*)
  - If you are not spanning multiple servers the form database.schema.table is used



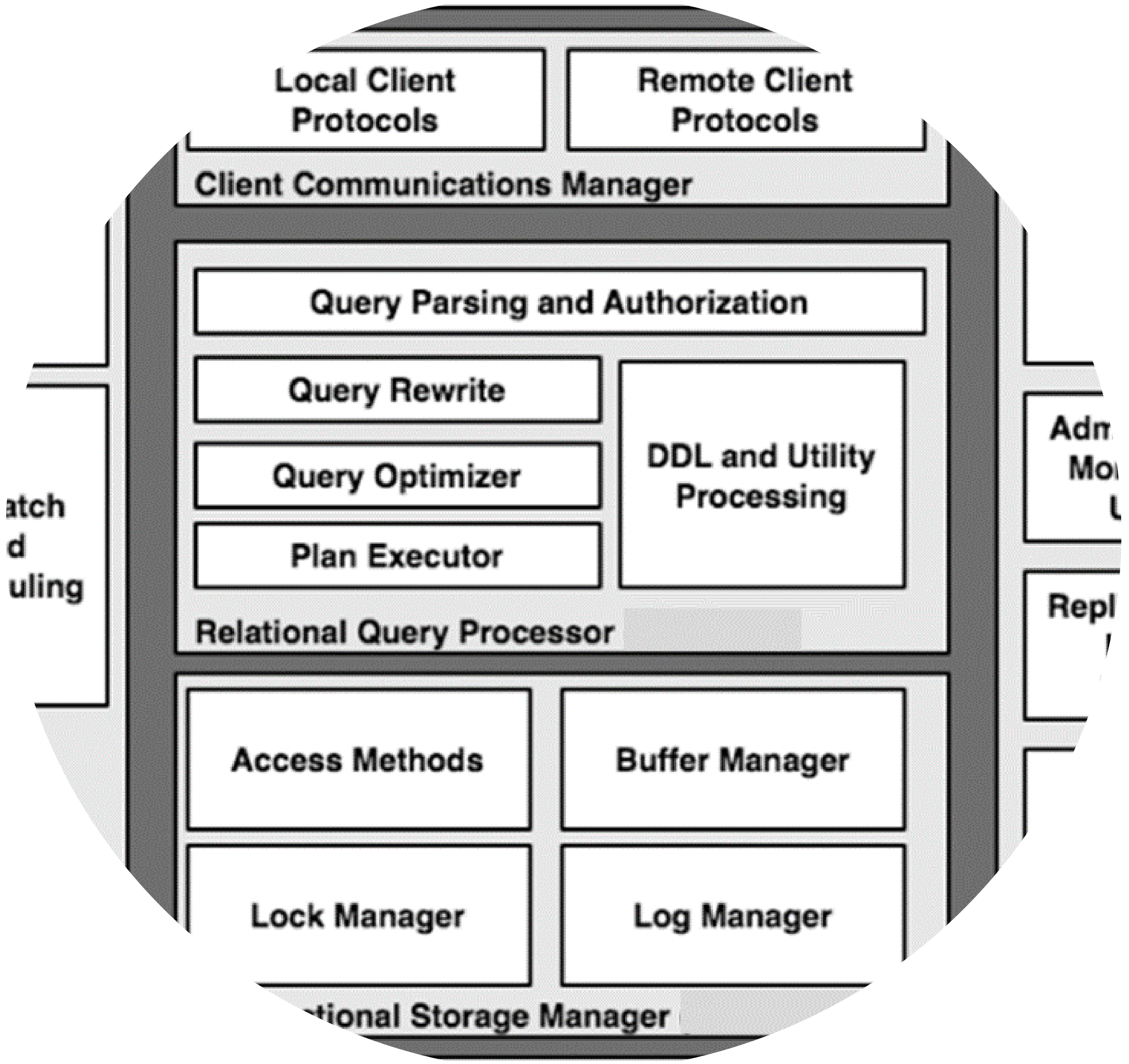
Query Parsing

- The query processor then invokes the catalog manager to check that the table is registered in the system catalog
- It then uses the catalog to ensure that attribute references are correct.



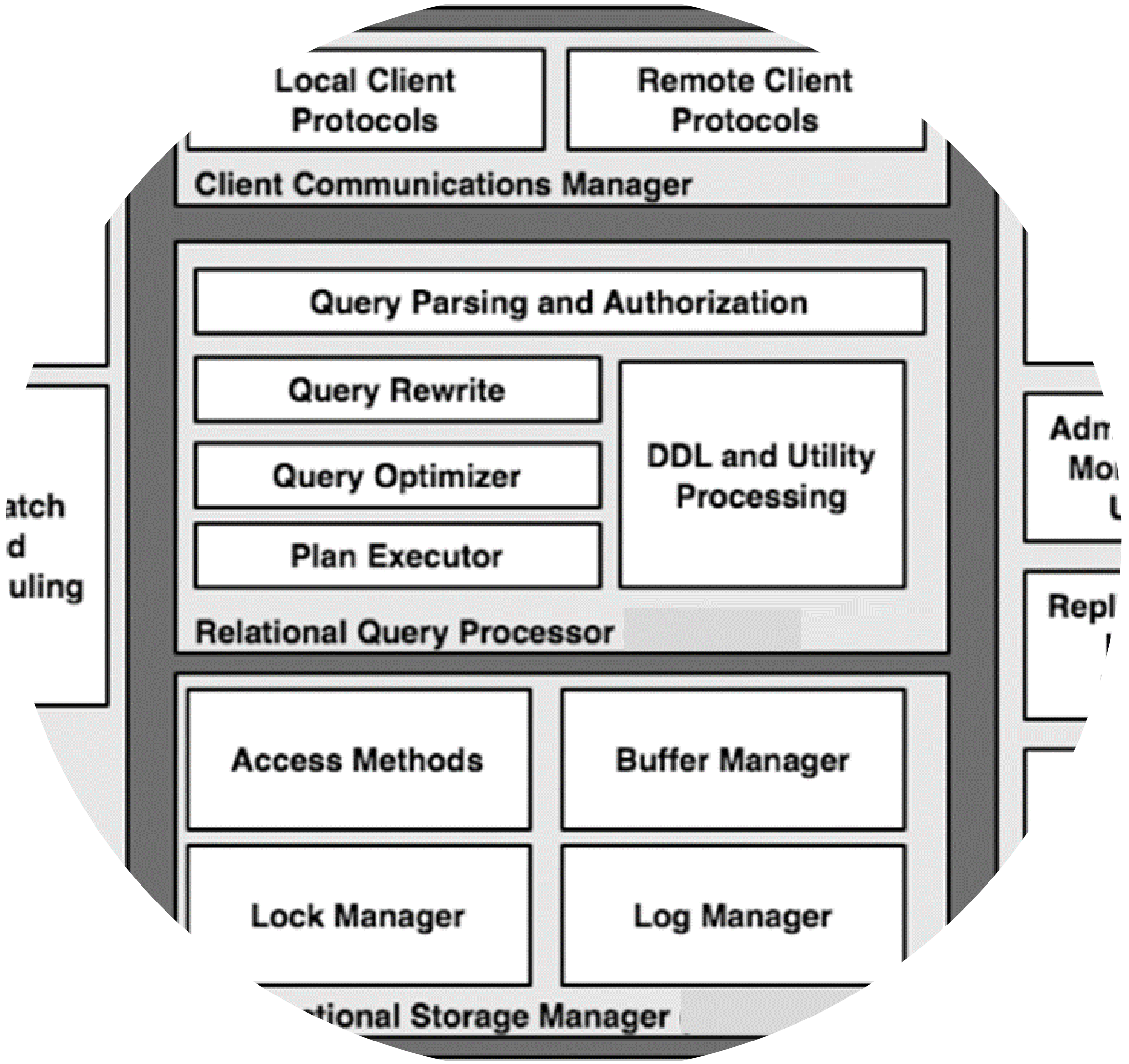
Query Parsing

- The data types of attributes are used to drive the disambiguation logic for overloaded functional expressions, comparison operators, and constant expressions
- For example
    - Suppose we include the expression (EMP.salary * 1.15) < 75000 in our query
    - Here decisions need to be made about
        - the code for the multiplication function and comparison operator
        - the assumed data type and internal format of the strings “1.15” and “75000”
    - These decisions depend on the data type of the EMP.salary attribute.
        - Could be an integer, a floating-point number, or a “money” value



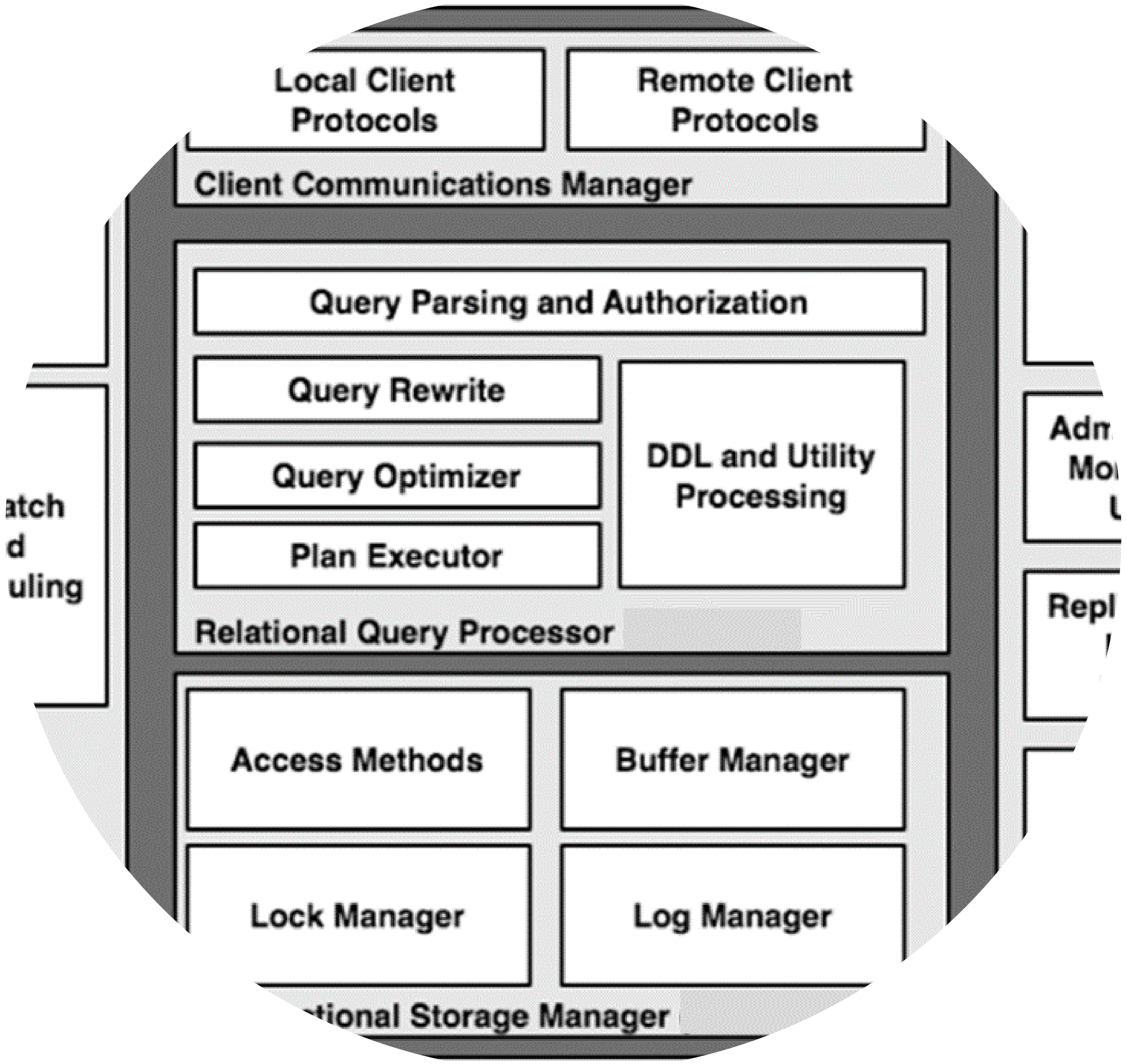
Query Parsing

- Additional SQL syntax checks are needed. E.g.:
  - Are tuple variables used consistently
  - Are tables compatible for use with set operators (UNION/INTERSECT/EXCEPT)
  - Are attributes suitable for use in aggregation
  - Are sub-queries nested appropriately ….



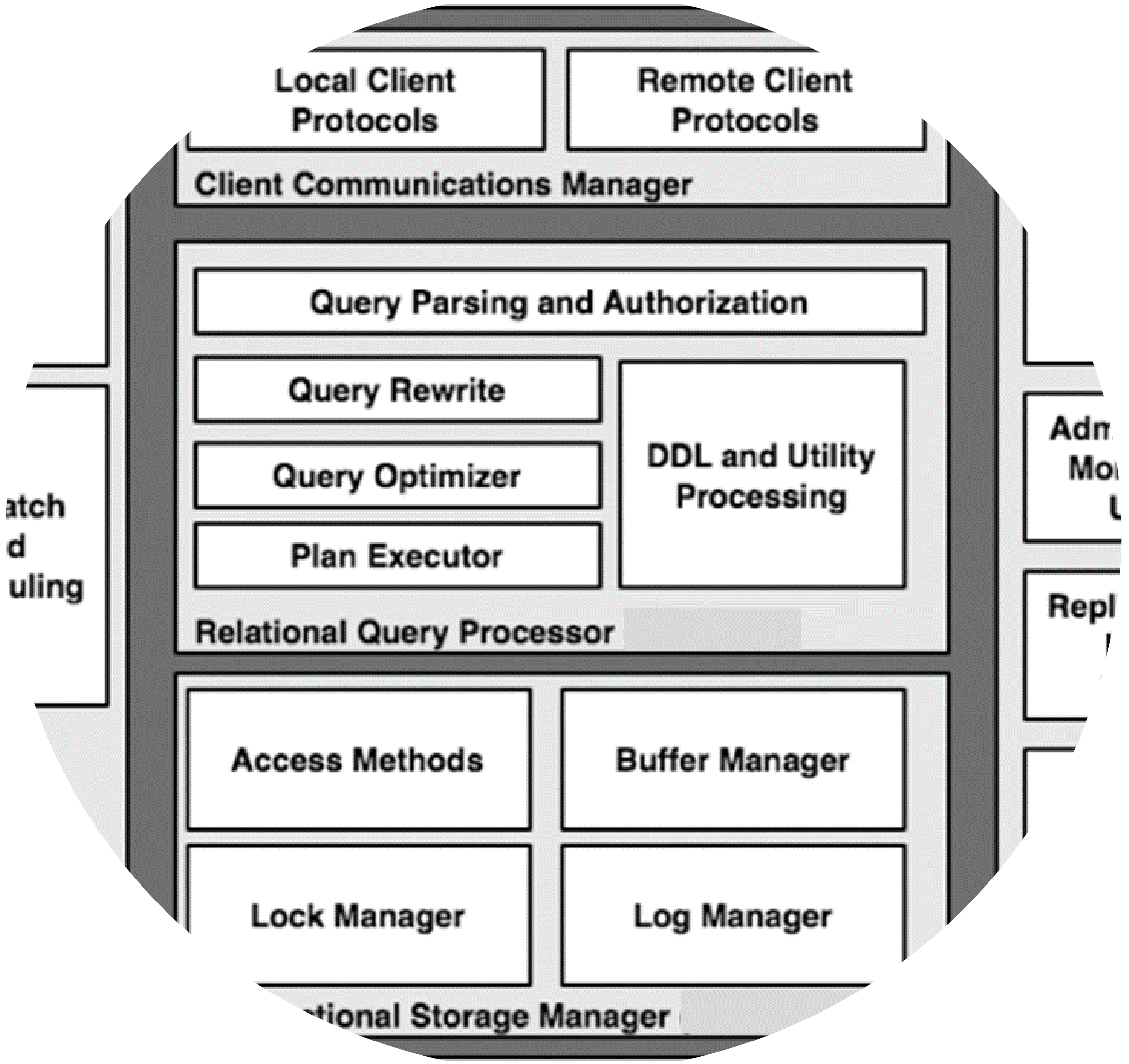
Query Authorization

- If query parses the next step is authorization
- Check to ensure that the user has appropriate permissions (SELECT/DELETE/INSERT/UPDATE) on the tables, user defined functions, or other objects referenced in the query



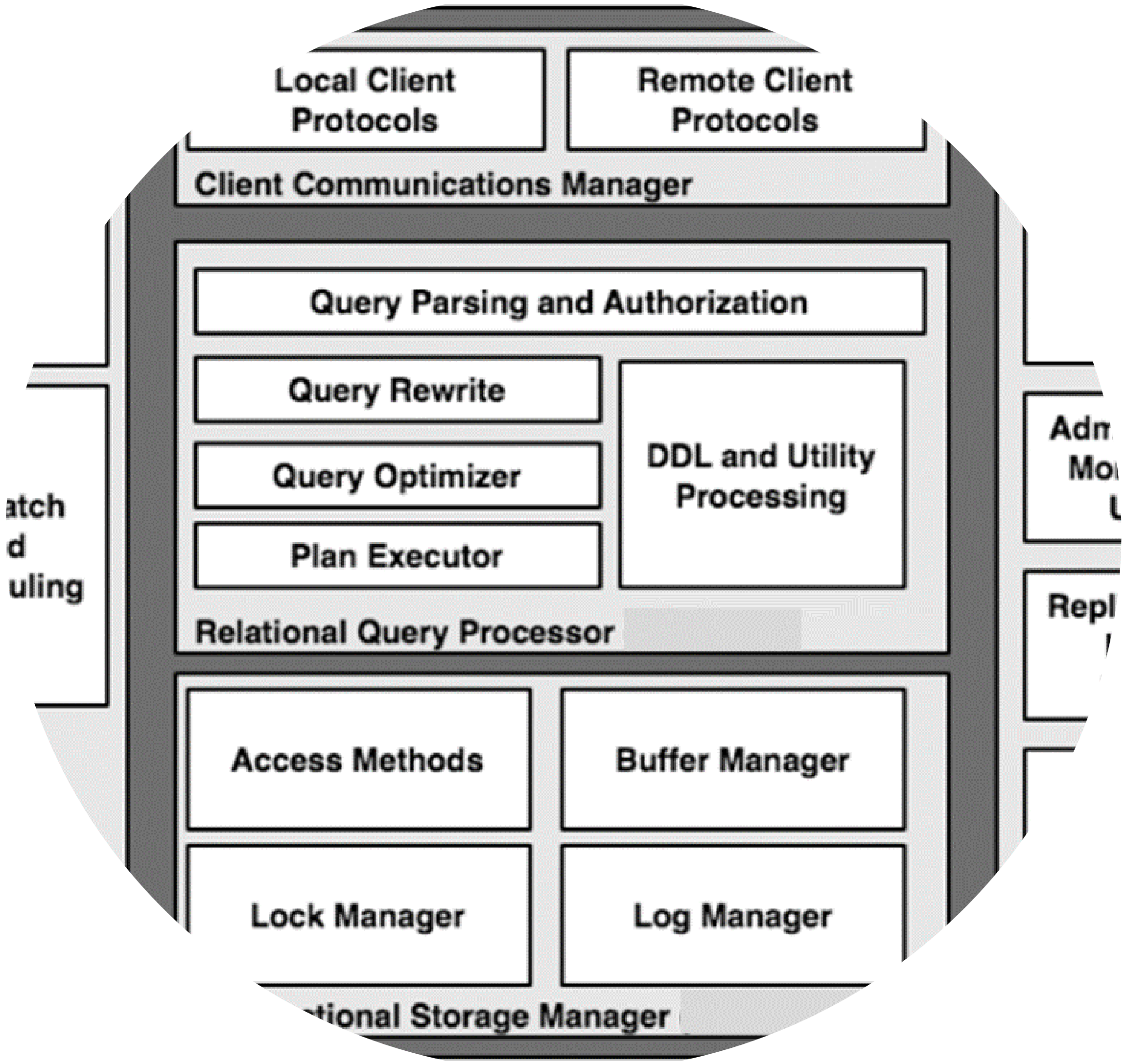
Query Rewrite

- Simplifies and normalizes the query without changing its semantics
- Can only use the query and the metadata in the catalog
  - Cannot use the data in the tables
- Rewrite = Translation
  - To an internal representation
- So what does it do?



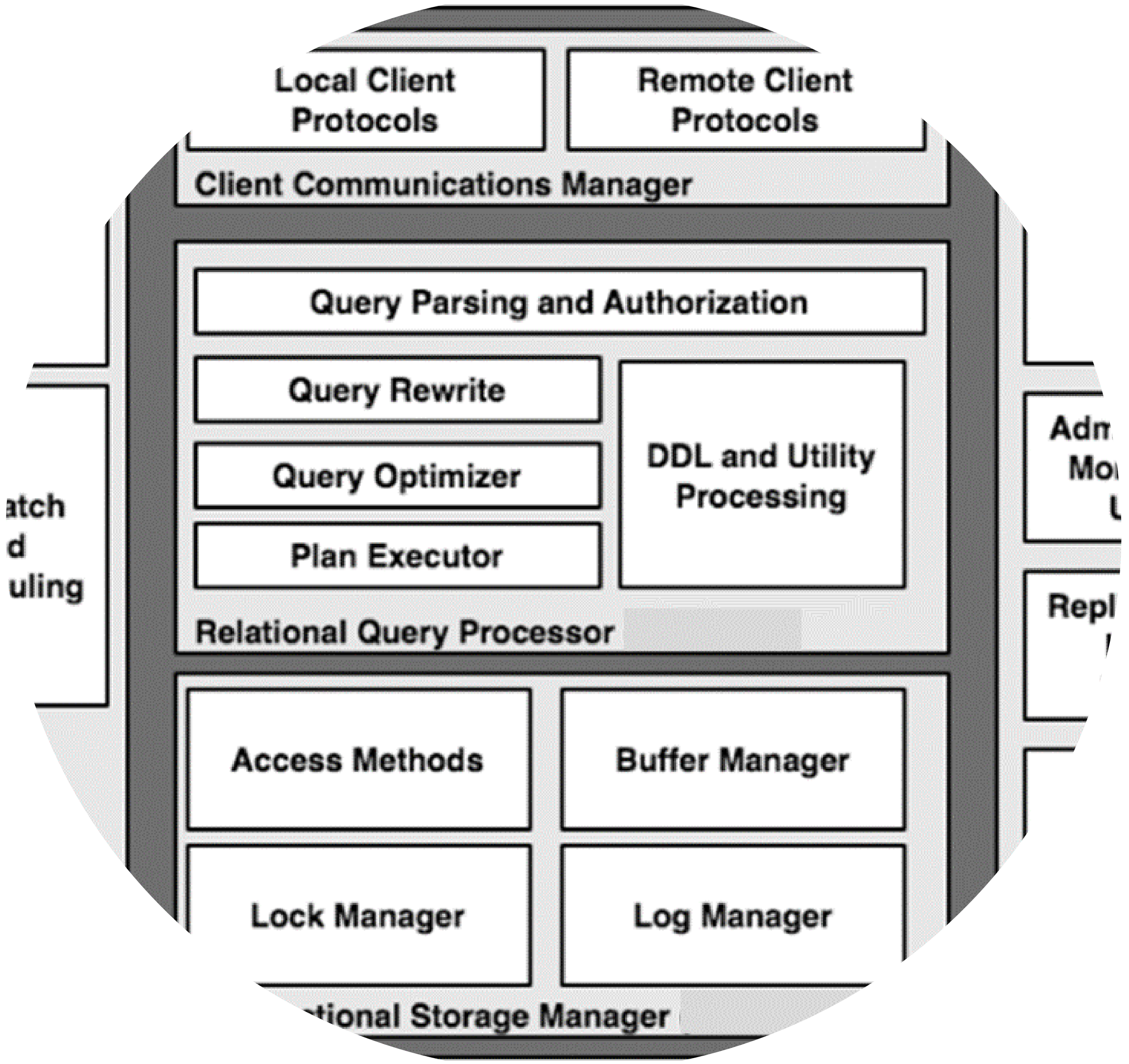
Query Rewrite

- View expansion
  - For each view reference that appears in the FROM clause
    - The rewriter retrieves the view definition from the catalog manager
    - It rewrites the query
      - replacing that view with the tables and predicates referenced by the view and
      - substituting any references to that view with column references to tables used in the view
  - This process is applied recursively until the query is expressed exclusively over tables and includes no views



Query Rewrite

- Logical Rewriting of Predicates
    - Logical rewrites are applied based on the predicates and constants in the WHERE clause
    - Simple Boolean logic is often applied to improve the match between expressions and the capabilities of index-based access methods
        - E.g A predicate such as NOT Emp.Salary > 1000000, for example, may be rewritten as Emp.Salary <= 1000000
    - Transitivity of predicates is used to produce new predicates
        - E.g. R.x < 10 AND R.x = S.y Could become R.x < 10 AND S.y < 10.”
        - Adding these transitive predicates increases the ability of the optimizer to choose plans that filter data early in execution, especially through the use of index-based access methods

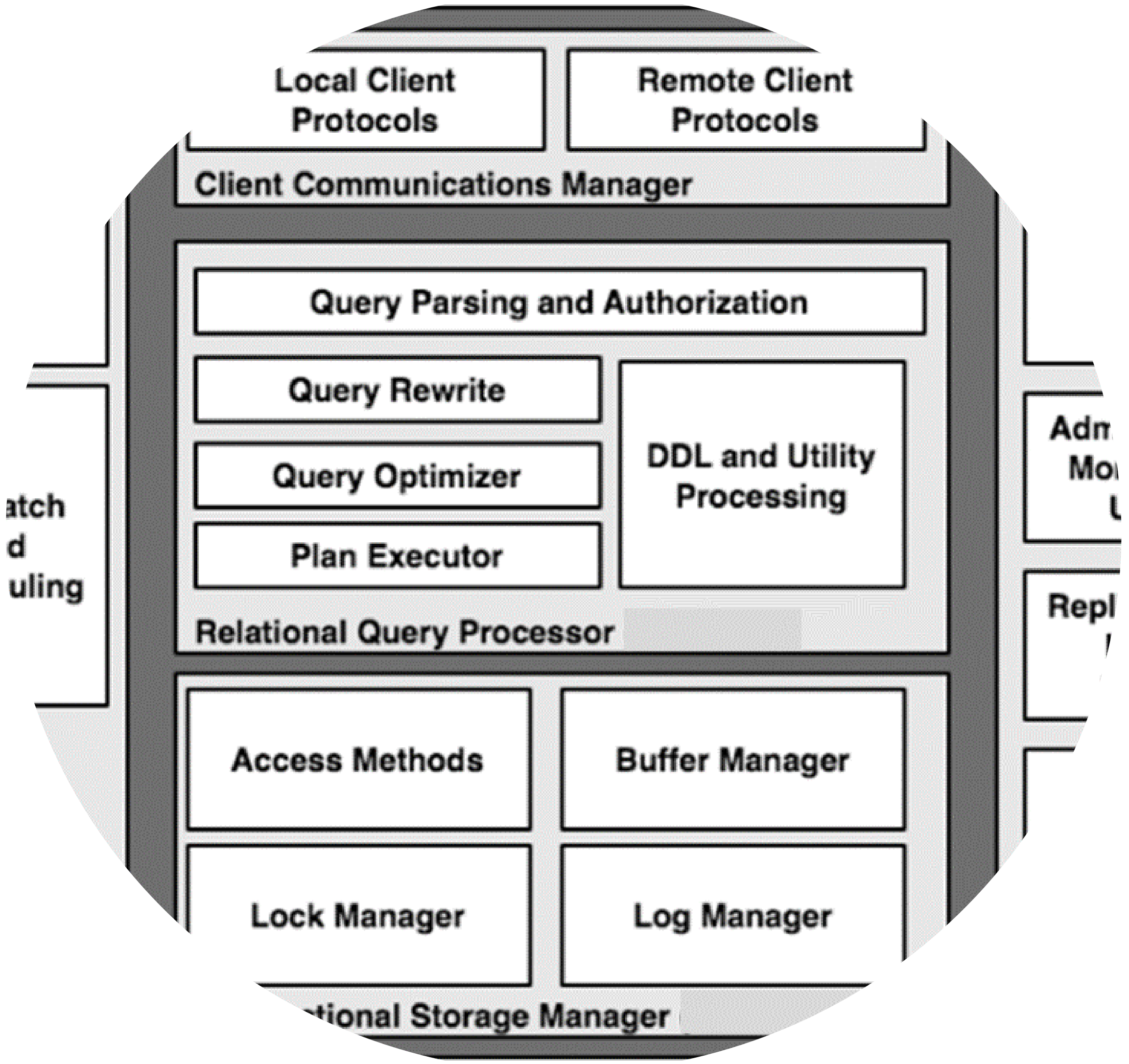


Query Rewrite

- Semantic Optimization
  - Integrity constraints on the schema are stored in the catalog, and can be used to help rewrite some queries.
  - Can lead to redundant join elimination.

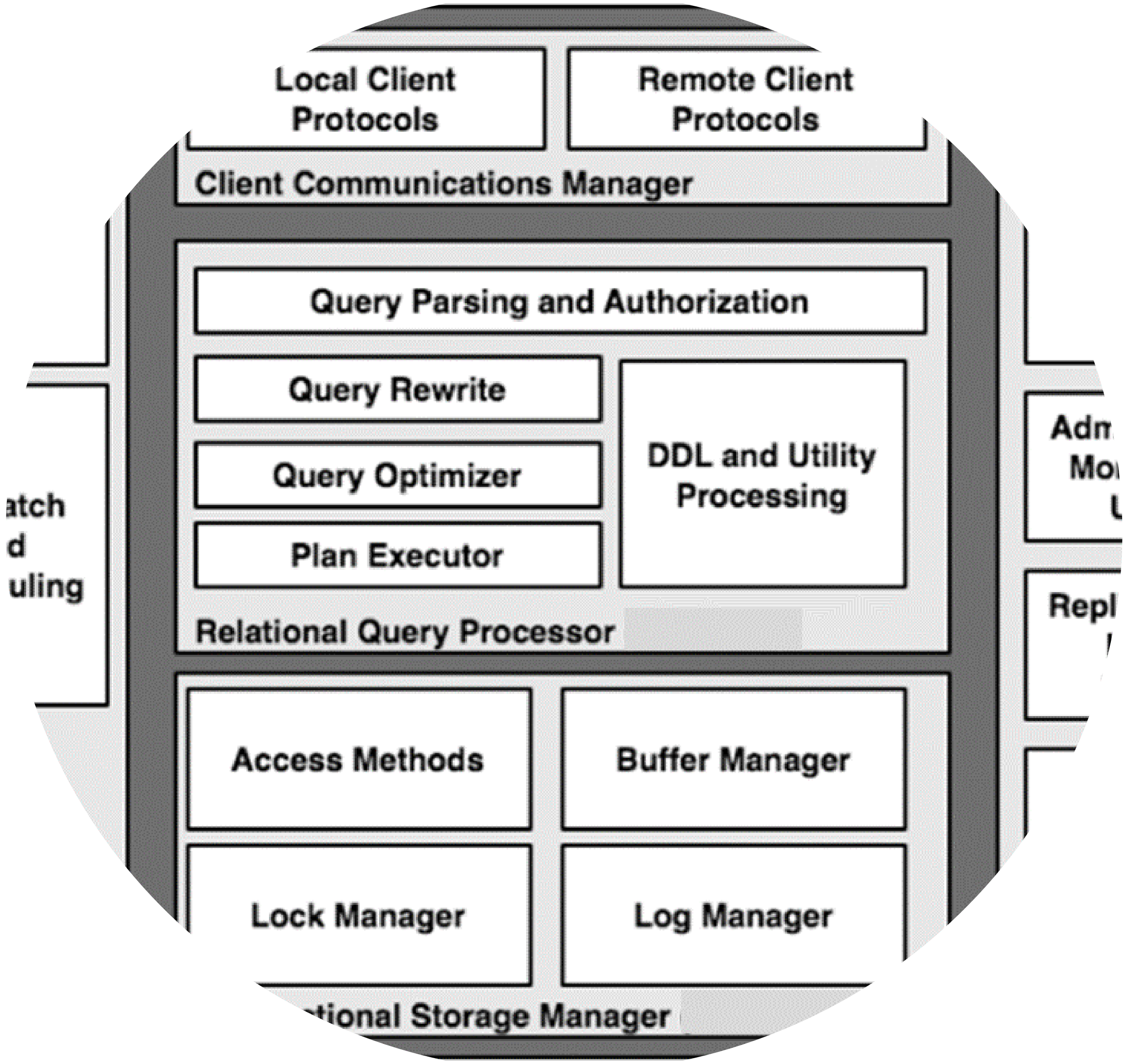
```
SELECT        E.Lname, M.Lname
FROM          EMPLOYEE AS E, EMPLOYEE AS M
WHERE         E.Super_ssn=M.Ssn AND E.Salary >
M.Salary
```

  - Suppose we had implemented a constraint that an employee can never earn more than their manager?
  - If the semantic query optimizer checks for the existence of this constraint then it will know the result of this query will be zero so it doesn't have to run it at all



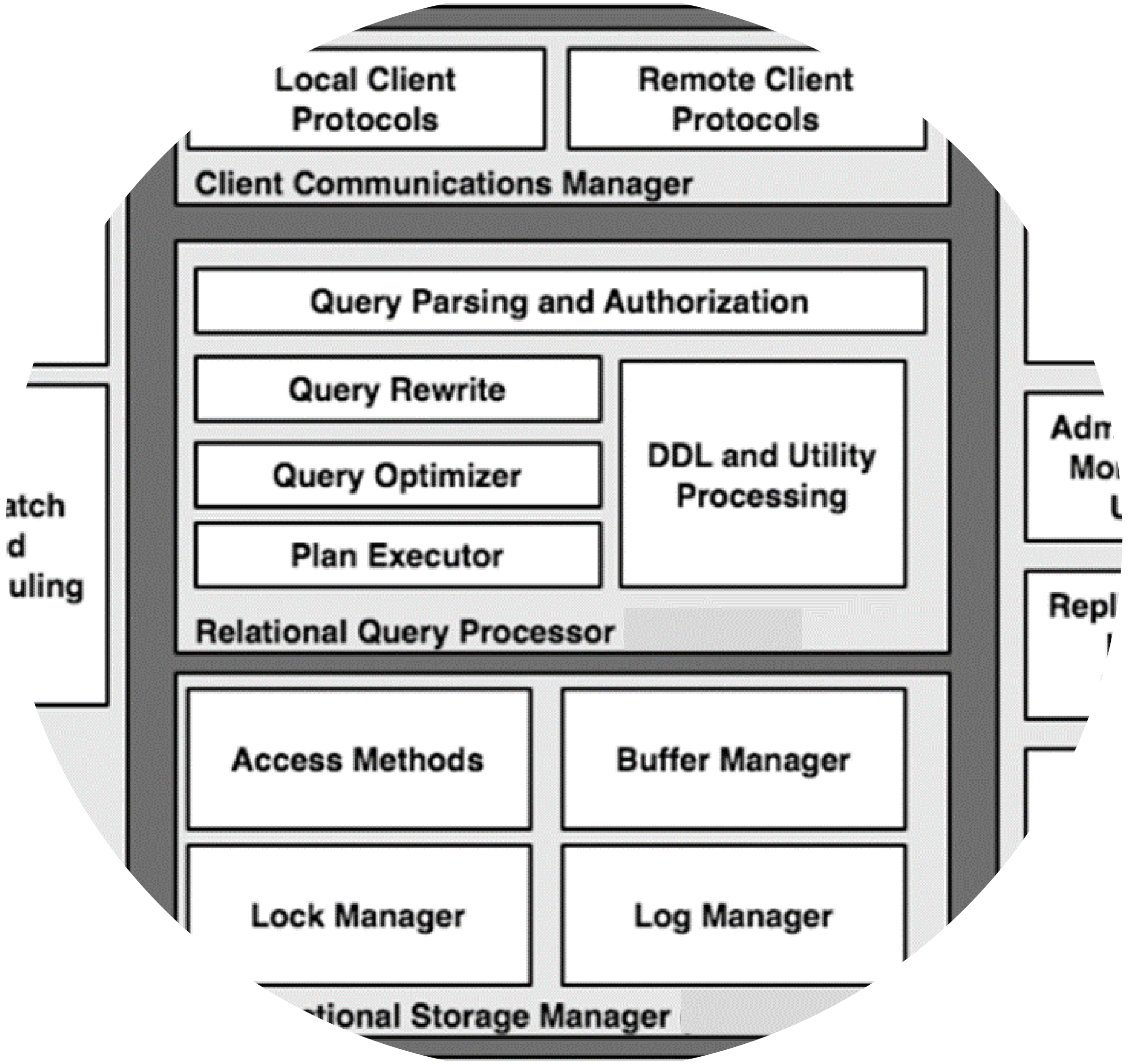
Query Optimizer

- SQL is a declarative language
    - A SQL query entered by a user describes what the user wants
    - The query is then parsed and rewritten into an internal query representation
- The query optimizer transforms an internal query representation into an efficient query plan for executing the query
- Query plan is a list of instructions that the DBMS needs to follow in order to execute a query on the data
    - Can be thought of (and depicted) as a dataflow diagram that pipes table data through a graph of query operators



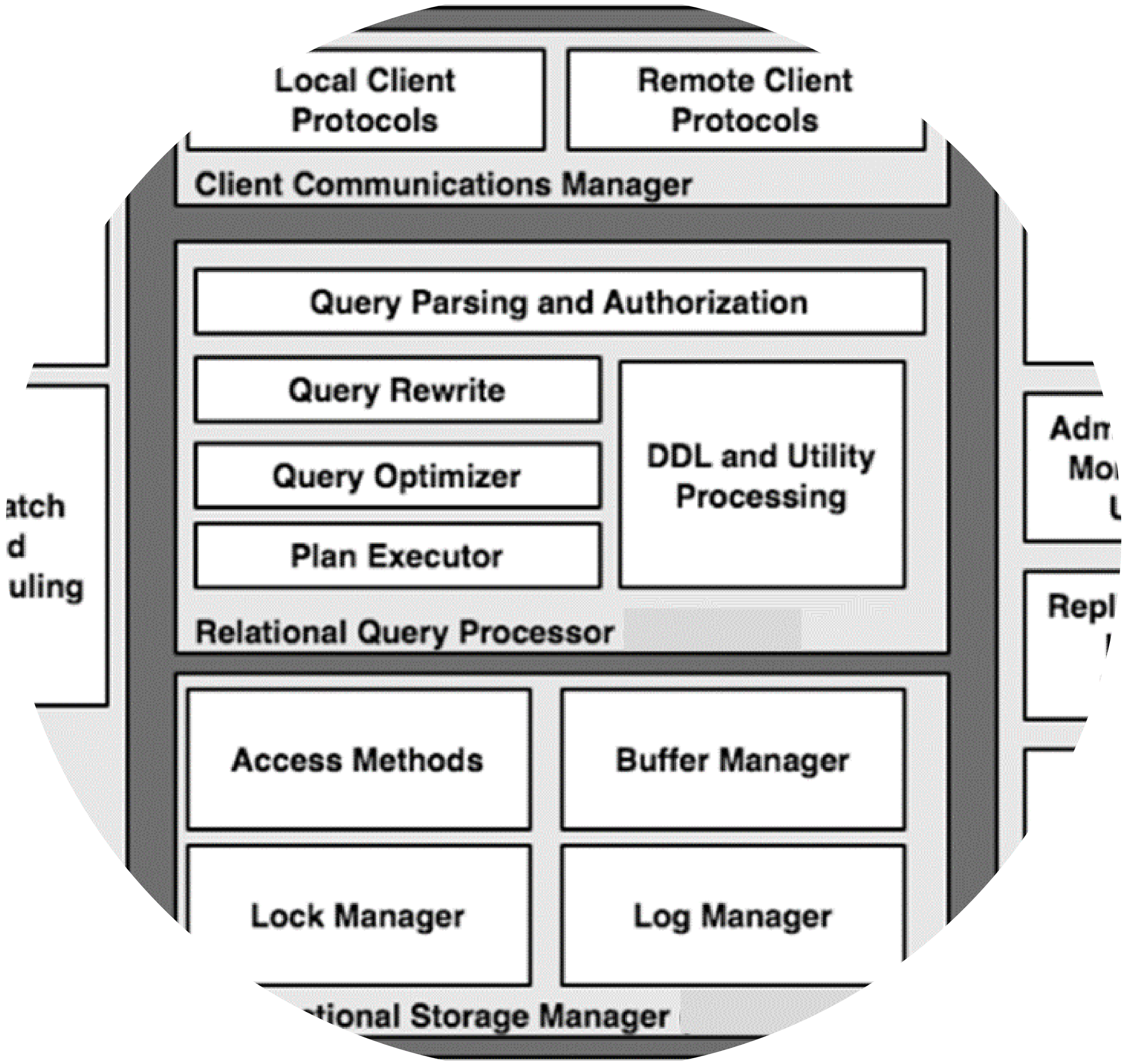
Query Optimizer Objective

- Objective:
  - For a given query, find a correct execution plan that has the lowest “cost”.
  - This is the part of a DBMS that is the hardest to implement well (proven to be NP-Complete).
- No optimizer truly produces the “optimal” plan
  - Use estimation techniques to guess real plan cost.
  - Use heuristics to limit the search space.



Query Optimization

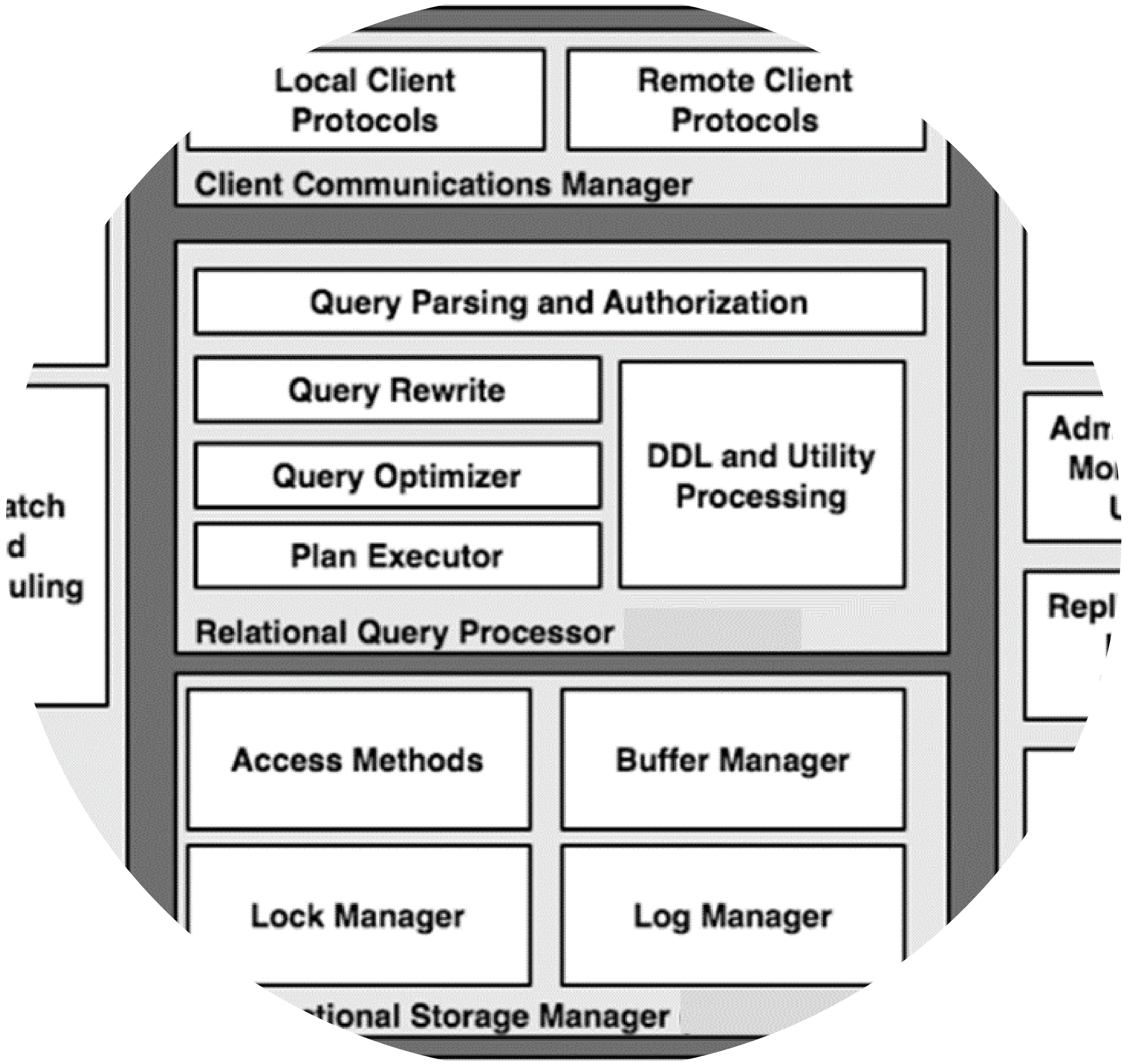
- A single query can be executed through different algorithms or re-written in different forms and structures.
- The query optimizer tries to determine the most efficient way to execute a given query by considering the possible query plans
- Benefits
  - Provides the user with faster results, which makes the application seem faster to the user
  - Allows the system to service more queries in the same amount of time (optimized queries take less time than unoptimized)
  - Ultimately reduces the amount of wear on the hardware (e.g. disk drives), and allows the server to run more efficiently (e.g. lower power consumption, less memory usage)



Query Optimizer

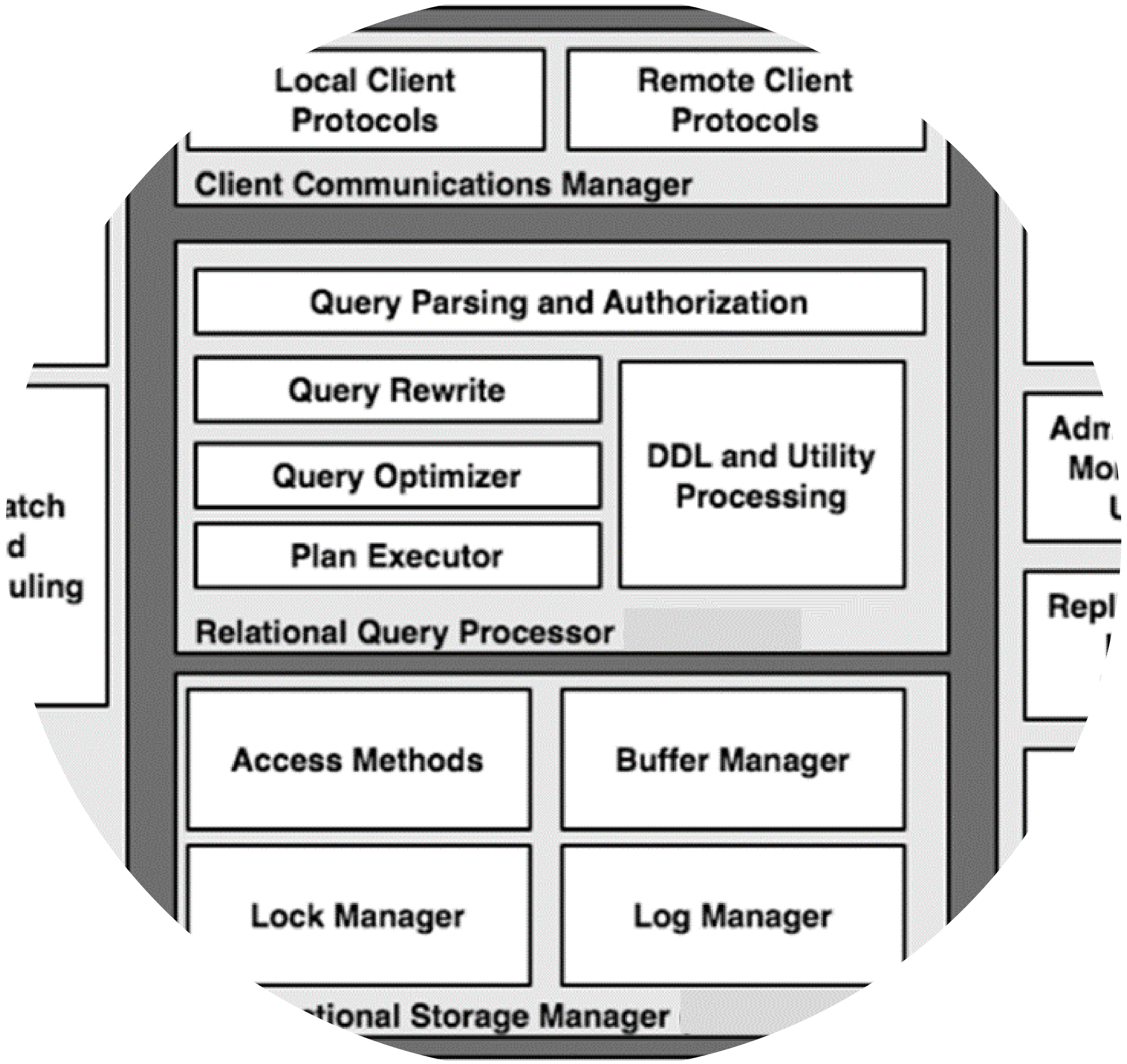
- Three components:
  - 1. Search space
  - 2. Plan enumeration algorithms
  - 3. Cardinality and cost estimation

- Note: First query optimizer was for System R, from IBM, in 1979, you will often see this references when reading about optimization in relational databases



Query Optimizer

- 1. Search space
- Two ways to optimize
  - Analyze and transform equivalent relational expressions
    - Attempting to minimize the tuple and column counts of the intermediate and final query processes
  - Using different algorithms for each operation
    - Underlying algorithms determine how tuples are accessed from the data structures they are stored in, indexing, hashing, data retrieval and hence influence the number of disk and block accesses



Relational Algebra

- Every SQL query can be rewritten as a relation.

```
SELECT R.A, R.B
FROM R
WHERE R.A > 5;
```

- Can be written as:

```
PROJECT(A, B) [ SELECT(A > 5) [ R ] ]
```

- Form:
  - An operator takes as input a relation (or two in the case of joins and unions) and produces another relation as output
- A series of operators can therefore be chained together, each consuming the output of the operator that precedes it, right down to the source
  - You imagine the data as streams of information.
  - You channel these streams through operators that filter out unneeded data, join it with other streams of information or process some kind of aggregate function like sum() or count().
- The appropriate operators applied in the appropriate order materializes the query's results

Relational Algebra

- We can model the chain of operators as a tree.
- The leaves of the tree are the raw tables.
- The output of the root operator is the query’s result.
- Since the operator interface is uniform for all relational operators, it is easy to define each operator as a class implementing the operator interface.
- We can translate SQL into a tree of Relational Operators, also termed as an **Abstract Syntax Tree (AST).**

Relational Algebra

- We can describe tables in a relational database as sets of tuples
- We can describe query operators using set theory
- The query language is called relational algebra
  - Every query can be converted to relational algebra
  - Operands --- variables or values from which new values can be constructed
  - Operators --- symbols denoting procedures that construct new values from given values
  - Expressions can be constructed by applying operators to atomic operands and/or other expressions
- Relational algebra can be converted to tree with joins as branches
- Each operator has implementation choices
- Operators can also be applied in different order

Relational Algebra SELECT (σ)

- The SELECT operation is used for selecting a subset of the tuples according to a given selection condition.
- Sigma(σ)Symbol denotes it.
- It is used as an expression to choose tuples which meet the selection condition.
- Select operator selects tuples that satisfy a given predicate.
- σp(r)
    - σ is the predicate
    - r stands for relation which is the name of the table
    - p is prepositional logic

Relational Algebra SELECT (σ)

- Select operator selects tuples that satisfy a given predicate.
- σp(r)
  - σ is the predicate
  - r stands for relation which is the name of the table
  - p is prepositional logic
- Examples
  - σ ModuleName = "Advanced Databases" (Modules)
    - Selects tuples from Modules where ModuleName = 'Advanced Databases'
  - σ ModuleName = "Advanced Databases" and Lecturer = "Joe Lecturer" (Modules)
    - Selects tuples from Modules where ModuleName = 'Advanced Databases' and Lecturer = "Joe Lecturer"
  - σ sales > 50000 (Customers)
    - Output – Selects tuples from Customers where sales is greater than 50000

Relational Algebra Projection(π)

- Projection eliminates all attributes of the input relation but those mentioned in the projection list.
- The projection method defines a relation that contains a vertical subset of Relation.
- This operator helps you to keep specific columns from a relation and discards the other columns.
- This operator helps you to keep specific columns from a relation and discards the other columns.
- (pi) symbol is used to choose attributes from a relation.

Relational Algebra Projection(π)

StudentID	StudentName	RegStatus
1	Joe	Registered
2	Jane	Eligible
3	Jim	Registered
4	Jemma	Registered

Π StudentName, RegStatus (Students)

StudentName	RegStatus
Joe	Registered
Jane	Eligible
Jim	Registered
Jemma	Registered

Relational Algebra Inner Join (⋈)

- Join operation is essentially a cartesian product followed by a selection criterion

A ⋈ A.column 2 = B.column 2 (B)

Relational Algebra Outer Joins

Left Outer Join(A ⟕ B)

Keep all tuple in the left relation and if there is no matching tuple is found in right relation, then the attributes of right relation in the join result are filled with null values

Right Outer Join: (A ⟖ B)

Keeping all tuple in the right relation and if there is no matching tuple is found in the left relation, then the attributes of the left relation in the join result are filled with null values.

Relational Algebra Set Operations

- Union operation (υ)
  - It includes all tuples that are in tables A or in B.
  - It also eliminates duplicate tuples.
  - So, set A UNION set B would be expressed as:
  - $A \cup B$
- An intersection is defined by the symbol ∩
  - Defines a relation consisting of a set of all tuple that are in both A and B.
  - However, A and B must be union-compatible
  - $A \cap B$

Relational Algebra Expressing as a Tree

- Leaves are operands
  - Either variables standing for relations or particular, constant relations
- Interior nodes are operators, applied to their child or children

Operators

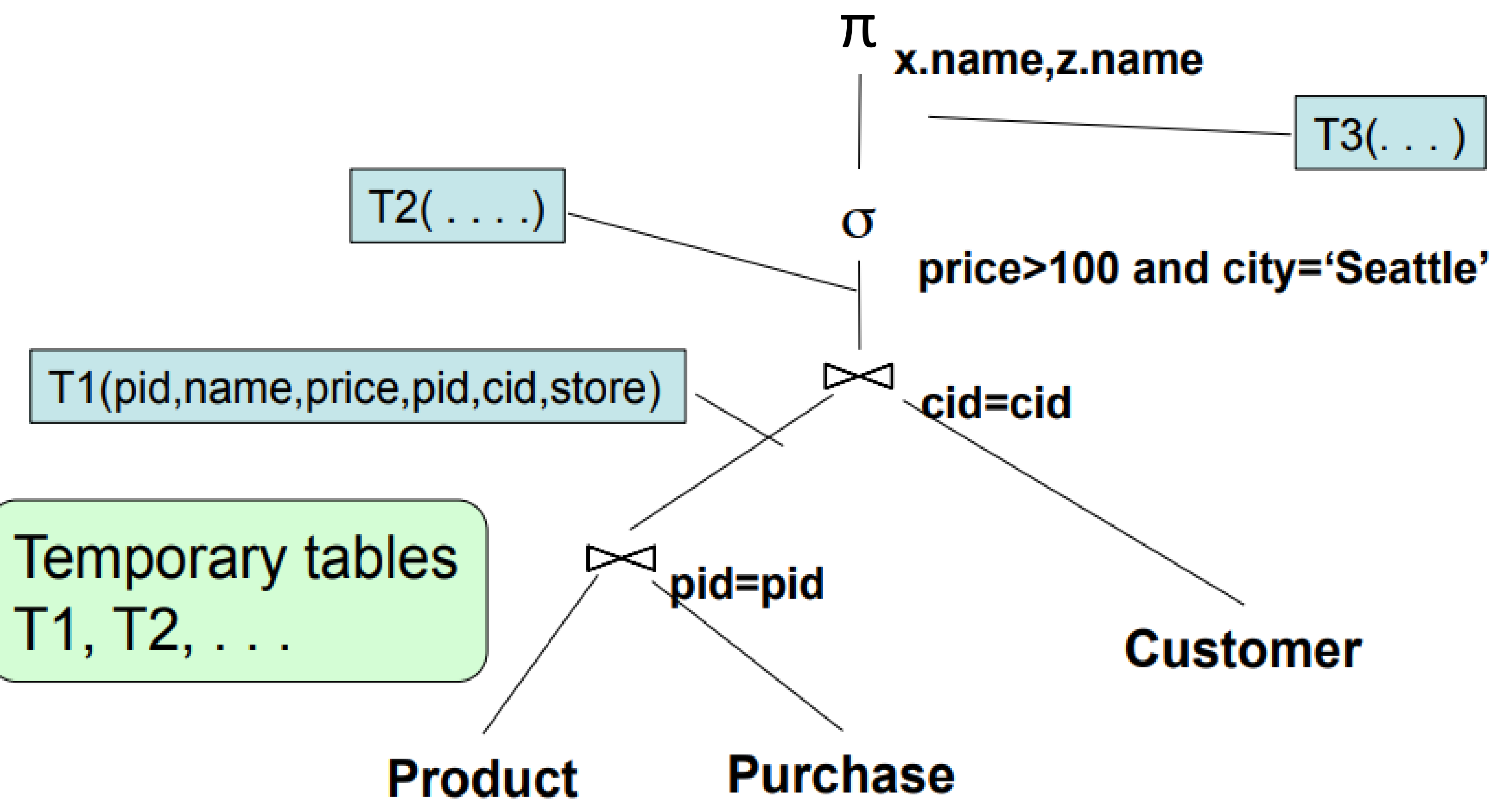
- Selection σ
- Projection Π
- Join ⋈ Left Outer Join(A ⟕ B) Right Outer Join: (A ⟖ B)
- Rename ρ
- Duplicate elimination δ
- Grouping and aggregation γ
- Sorting τ
- Union ∪
- Intersection ∩

Example SQL states the WHAT

- Product(pid, name, price)
- Purchase(pid, cid, store)
- Customer(cid, name, city)

```
SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid and y.cid = y.cid and
x.price > 100 and z.city = ‘Seattle’
```

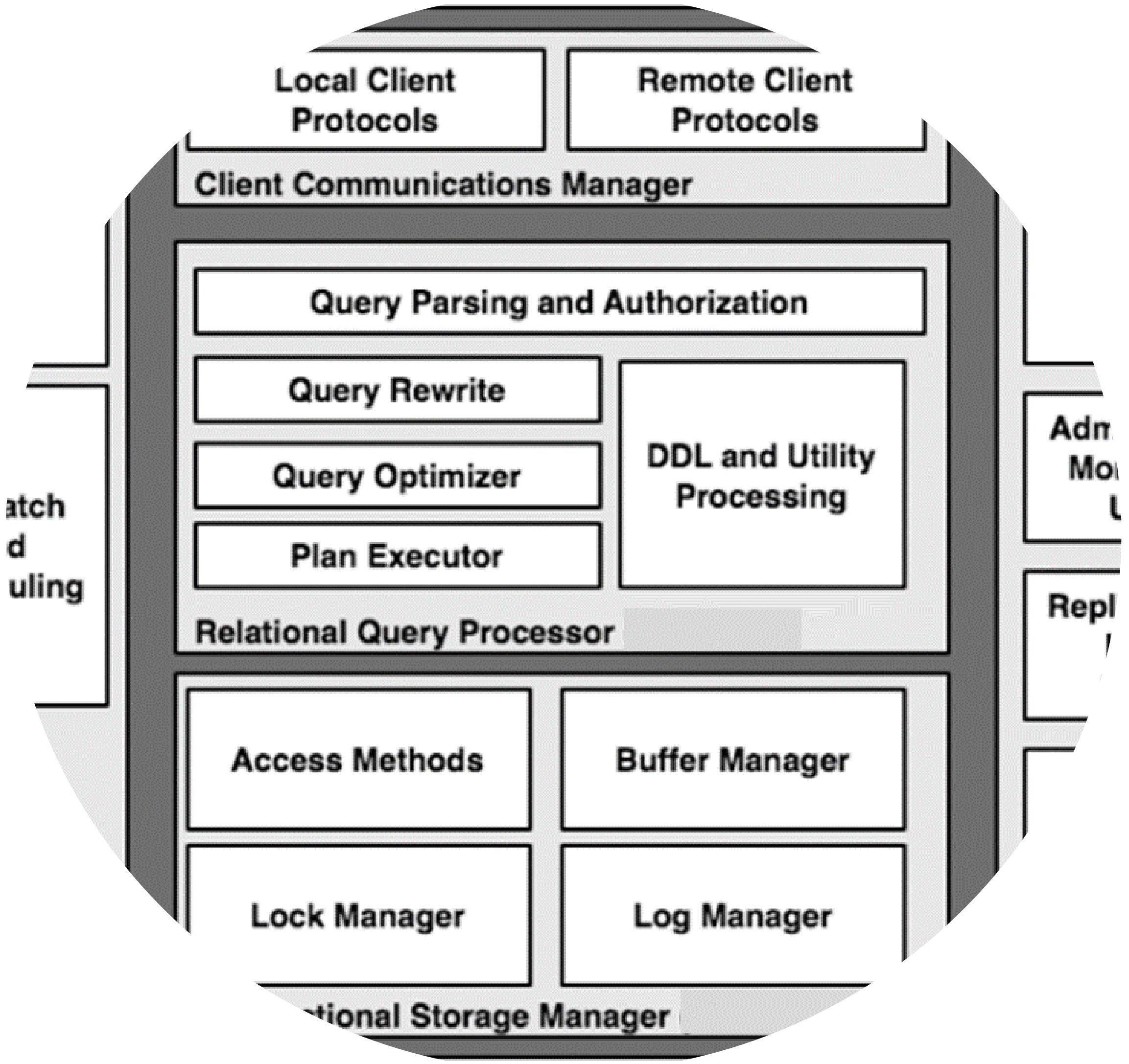
Example
Optimizer figures out the HOW


π x.name,z.name
T3(. . .)
σ price>100 and city='Seattle'
T2(. . . .)
⋈ cid=cid
T1(pid,name,price,pid,cid,store)
⋈ pid=pid
Temporary tables T1, T2, . . .
Customer
Product
Purchase

Iterate over PRODUCT…
…join with PURCHASE…
…join with CUSTOMER…
…select tuples with Price>100 and City='Seattle'…
…eliminate duplicates…
…to get your resultset

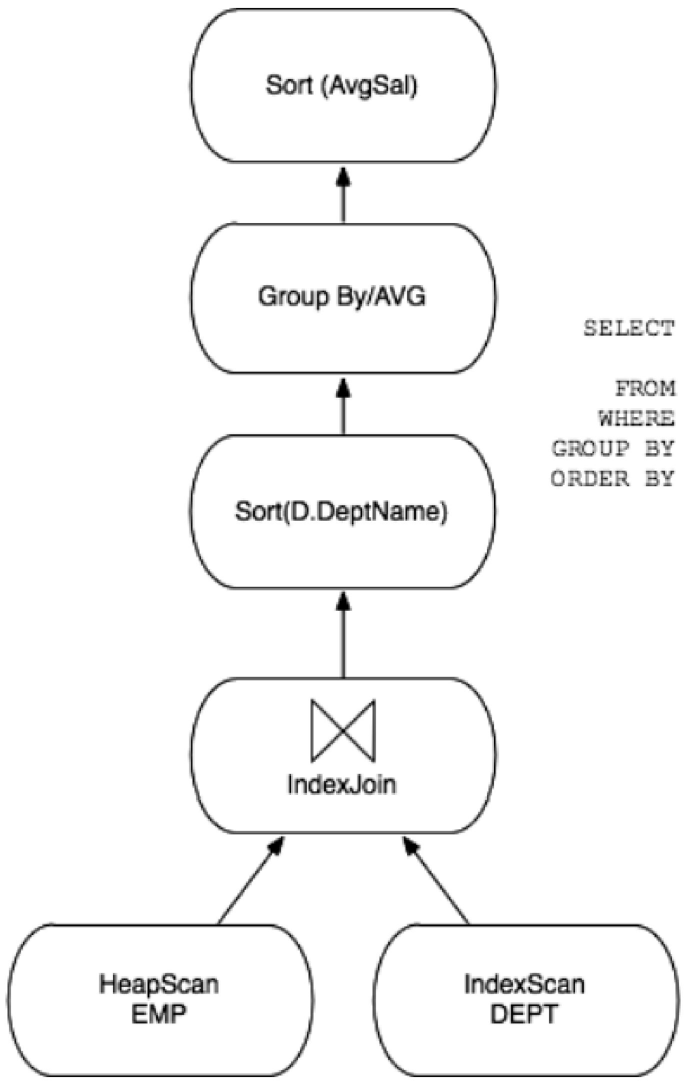
Query Optimizer

- 1. Search space
  - Set of all alternative plans that are considered by the optimizer
  - Defined by the set of algebraic laws and the set of plans used by the optimizer



Query Optimizer

- On completion, a few operators are typically added to the top of each query block as post-processing to compute GROUP BY, ORDER BY, HAVING and DISTINCT clauses if they exist. The various blocks are then stitched together in a straightforward fashion



```
 SELECT D.DeptName,
        AVG(E.Salary) AS AvgSal
   FROM EMP E, DEPT D
  WHERE E.Dno = D.DeptID
GROUP BY DeptName
ORDER BY AvgSal DESC
```

Example

- Find names of stars and the length of the movies they have appeared in 2014
  - Stars(name, address)
  - AppearIn(star_name,title, year),
  - Movies(title, year, length, type, studio_name)

Example

- Find names of stars and the length of the movies they have appeared in 2014
  - Stars(name, address)
  - AppearIn(star_name,title, year),
  - Movies(title, year, length, type, studio_name)

- Possible Plans
  - $\pi_{name,length}(\sigma_{year=2014}(Stars \bowtie AppearIn \bowtie Movies))$
  - $\pi_{name,length}(Stars \bowtie AppearIn \bowtie (\sigma_{year=2014}(Movies))$

Example

- Find names of stars and the length of the movies they have appeared in 2014
  - Stars(name, address)
  - AppearIn(star_name,title, year),
  - Movies(title, year, length, type, studio_name)

- Most Efficient Plan (Can be found by the optimizer)

πname,length(Stars $\bowtie$

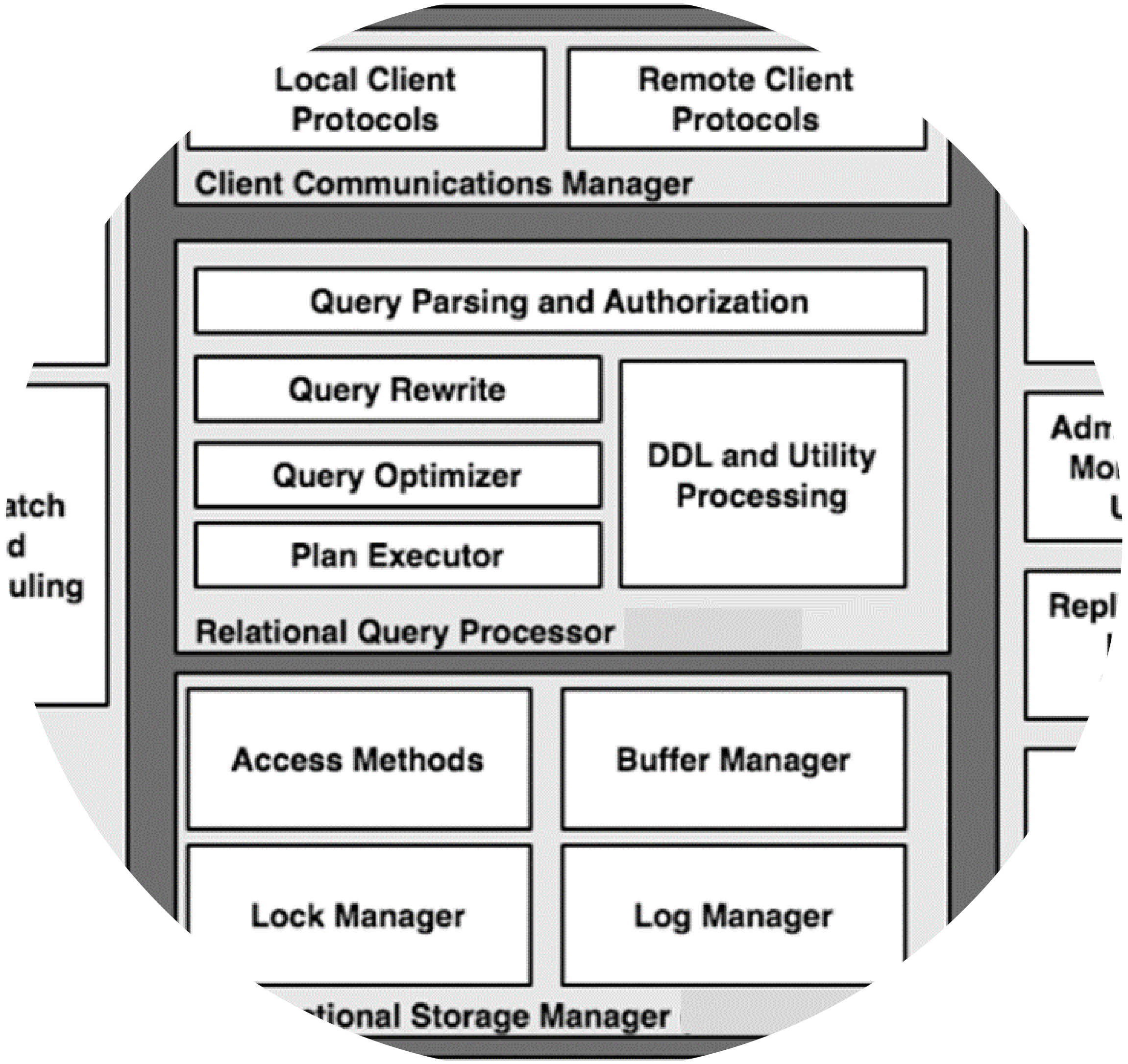
πname,length(AppearIn $\bowtie$

(πtitle,year,length σyear=2014(Movies)))

Query Optimizer

- Three components:
  - 1. Search space
  - 2. Plan enumeration algorithms
  - 3. Cardinality and cost estimation

- Note: First query optimizer was for System R, from IBM, in 1979, you will often see this references when reading about optimization in relational databases



Approaches

- Heuristics
- Heuristics + Cost-based Join Order Search
- Randomized Algorithms
- Stratified Search
- Unified Search

More about approaches Next Week

- Heuristics
- Heuristics + Cost-based Join Order Search
- Randomized Algorithms
- Stratified Search
- Unified Search