Implementing a DB Server (V1.13)

Operating Systems

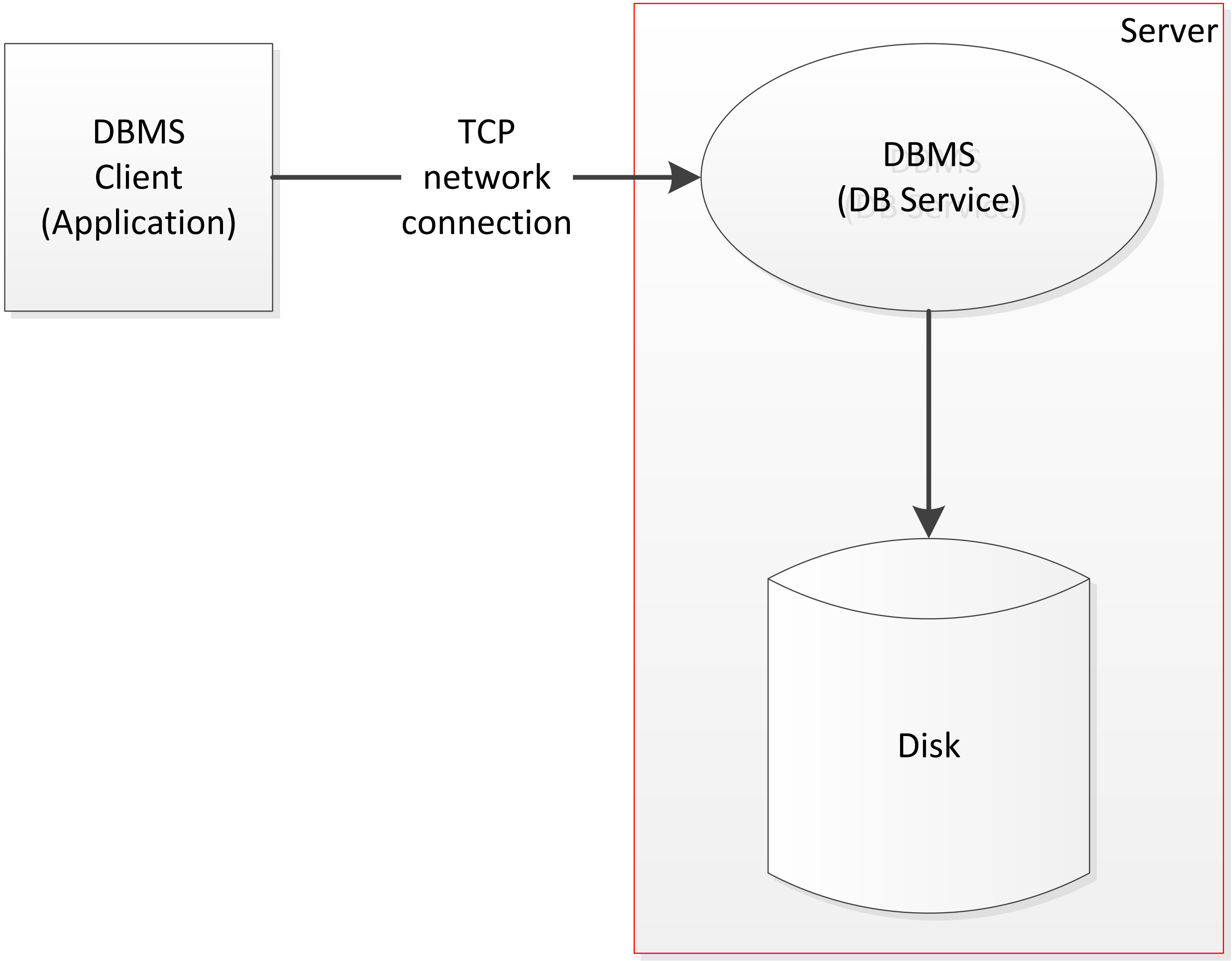
Computer Degree

Depto. de Arquitectura de Computadores
Universidad de Málaga

Sections

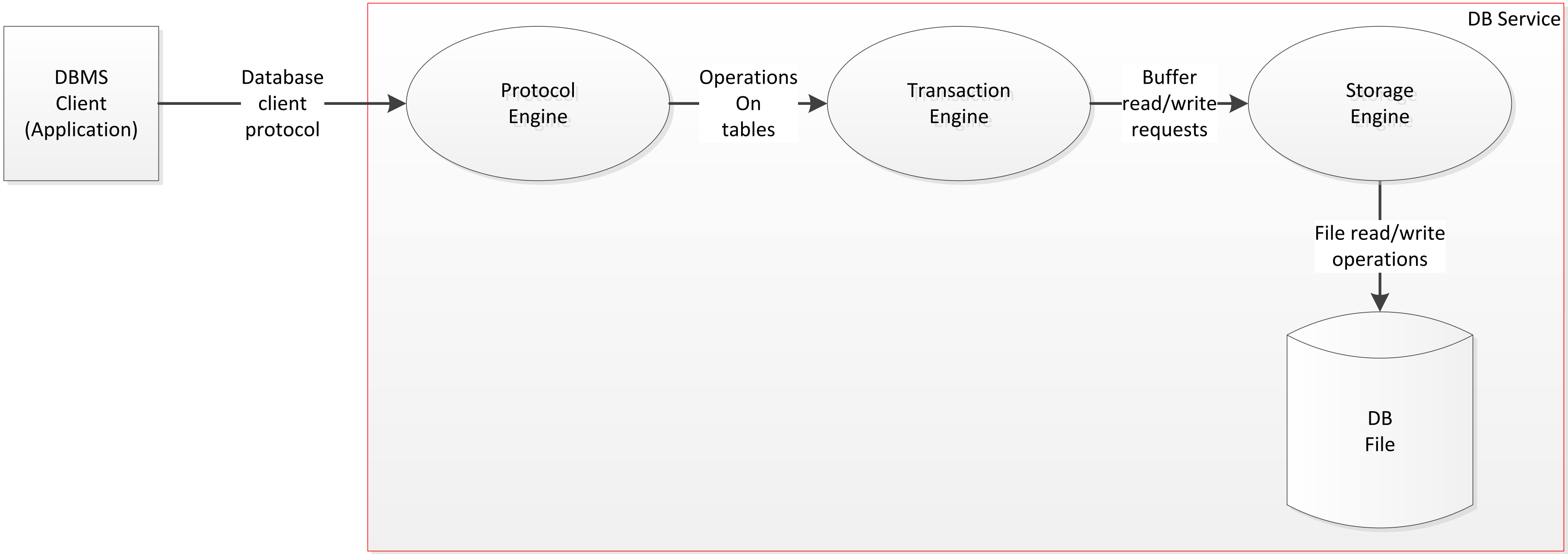
DBMS INTERNAL STRUCTURE

DBMS Server


DBMS
Client
(Application)
TCP
network
connection
Server
DBMS
(DB Service)
Disk

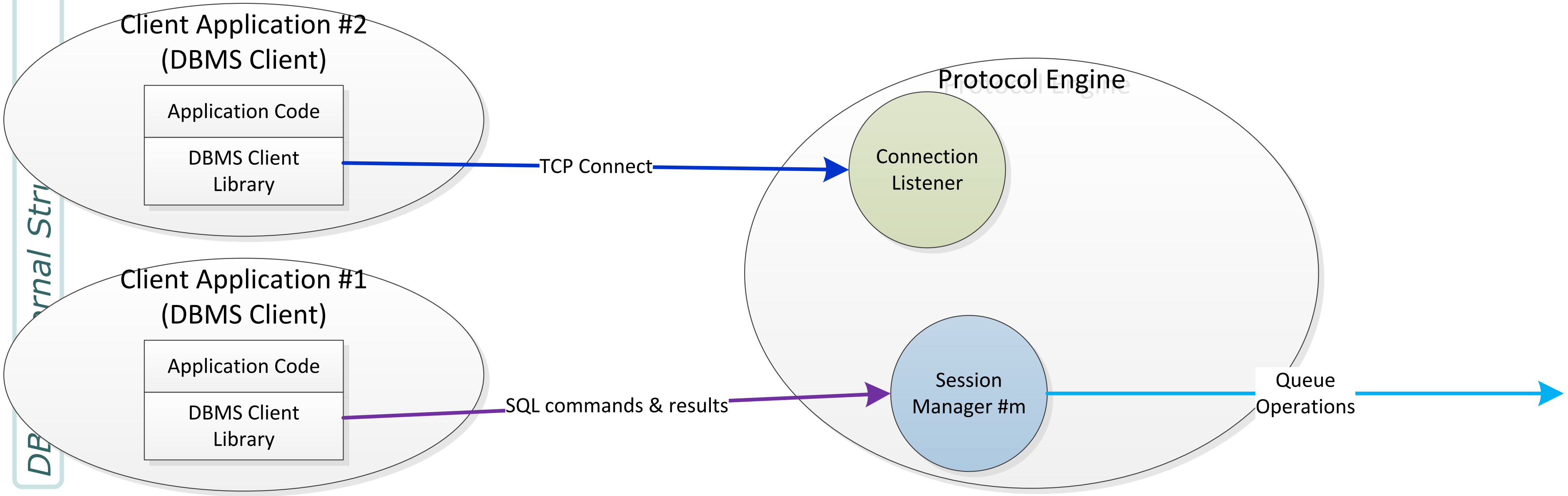
DBMS Internal Tasks

DBMS Internal Structure



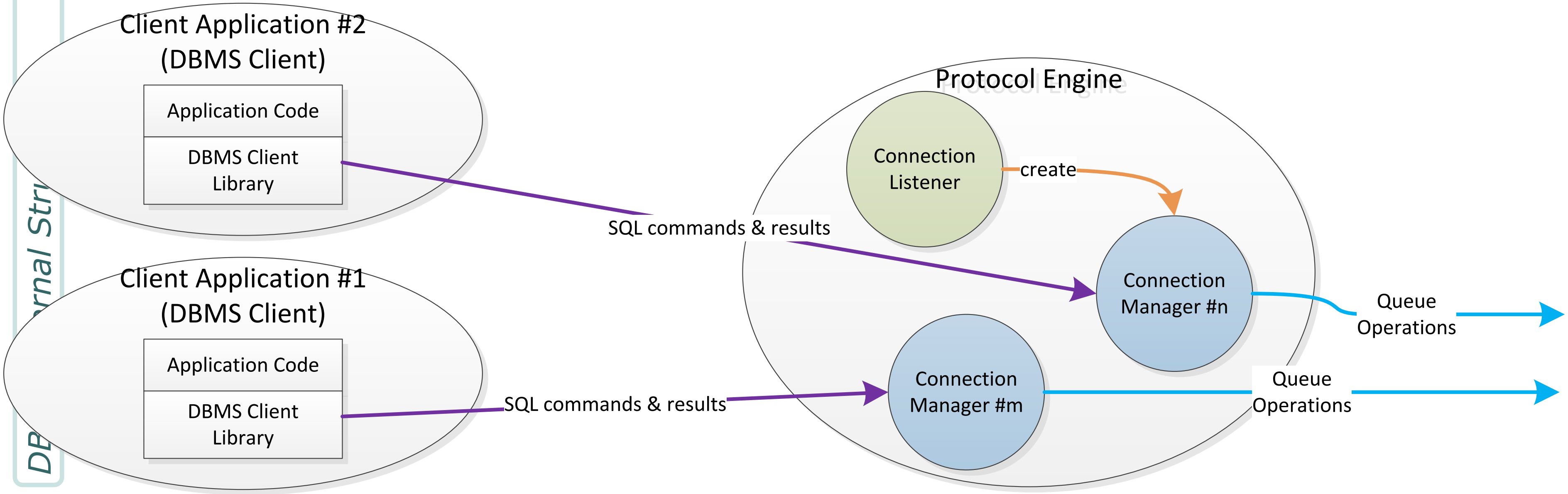
Protocol Engine: Connection Listener

- Connection listener only listens on a TCP port for new connections and creates one session manager for each connection.
- Each session manager decodes SQL syntax and converts commands into operations on tables for only one client.



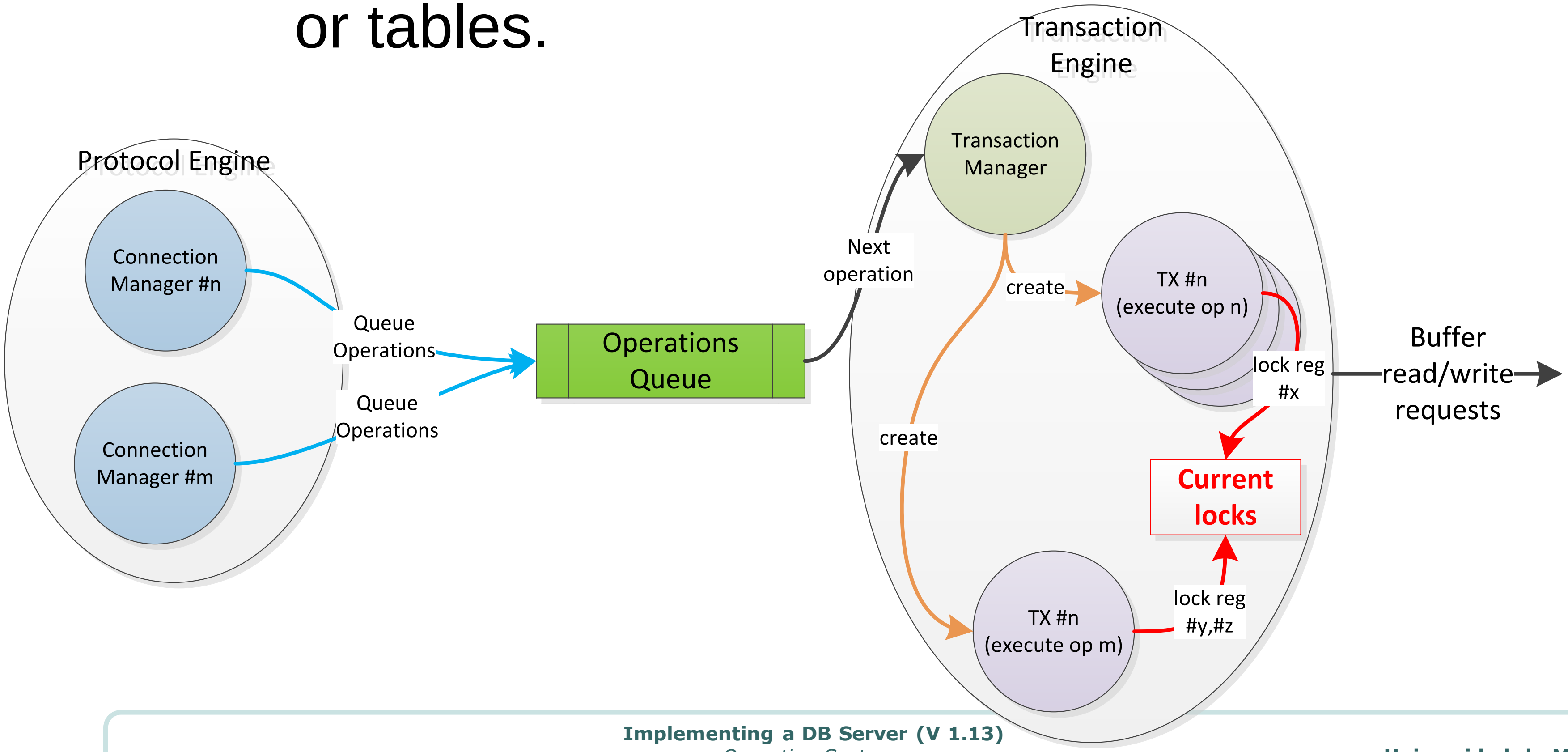
Protocol Engine: Session Manager

- ♦ Each client application dialogs with its own session manager process.



Transaction Engine

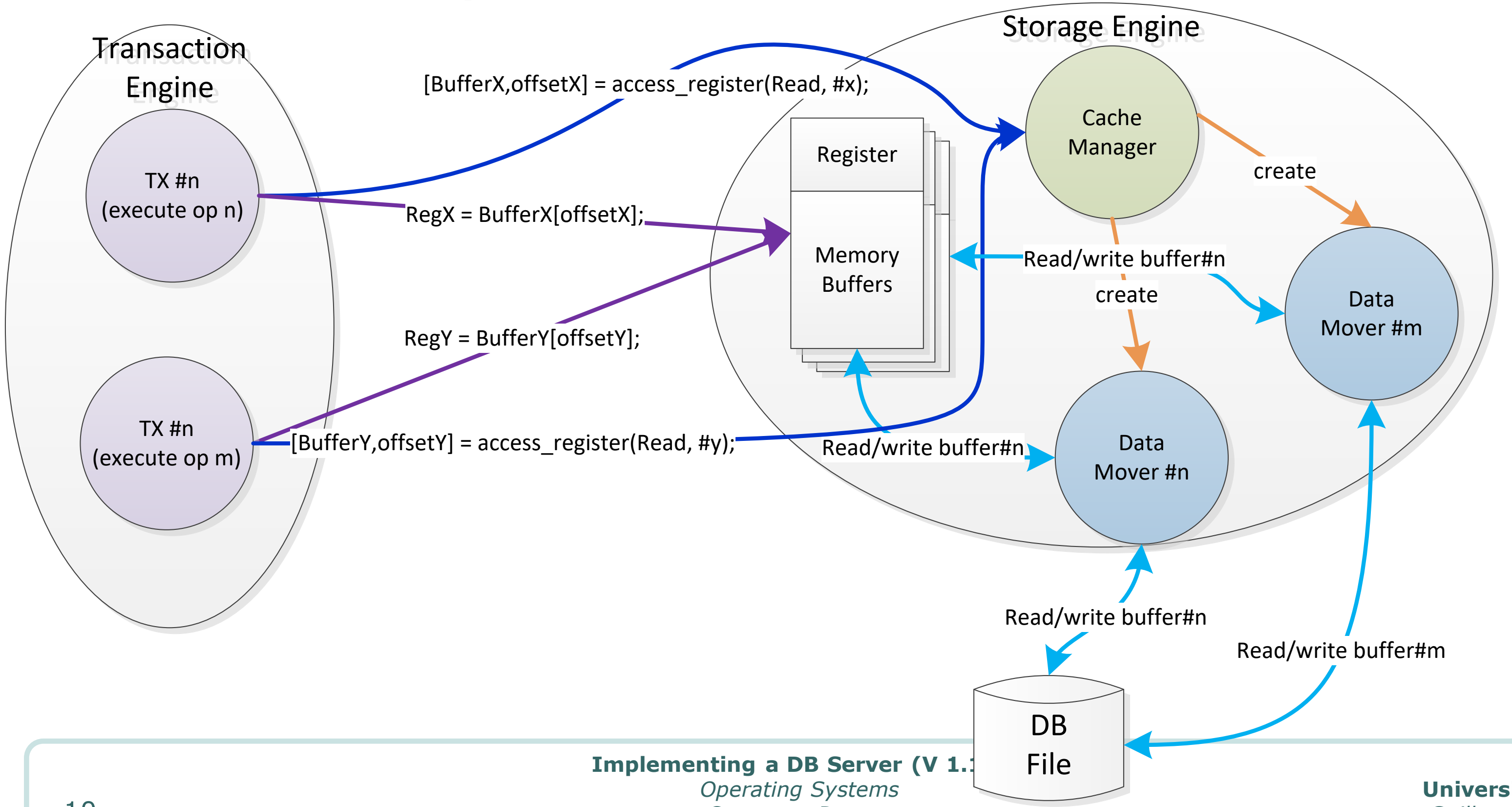
- Executes each operation in a new thread to achieve as much parallelism as possible.
- This engine also controls locks on registers or tables.



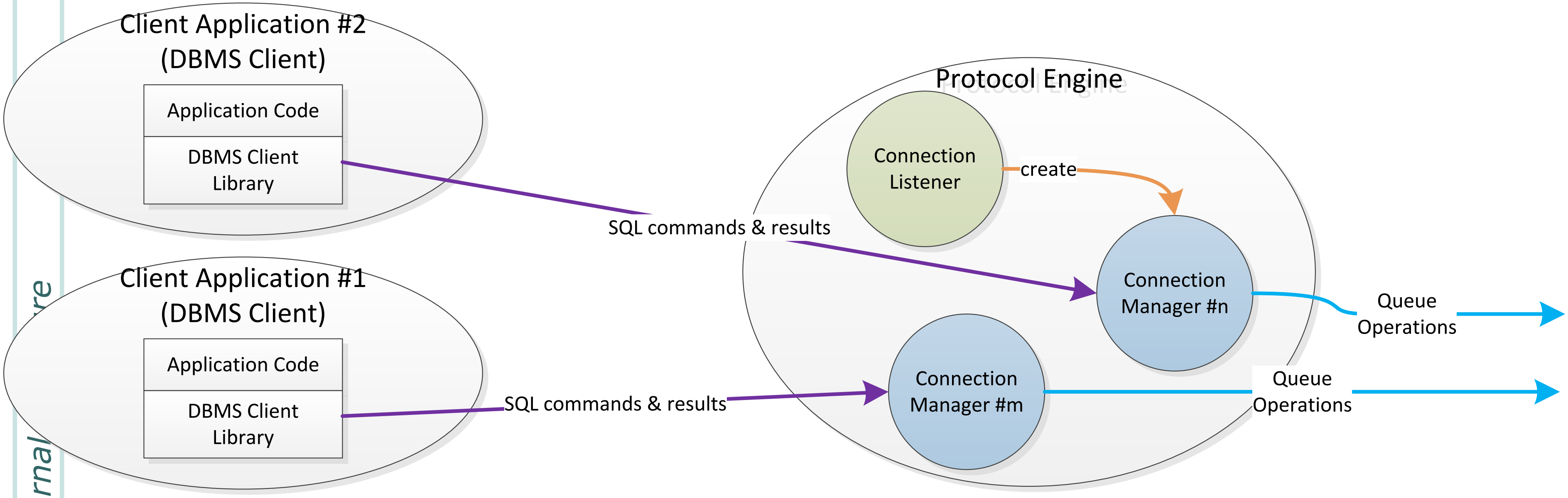
DBMS Internal Structure

Storage Engine

- Maintains a cache of the file in memory.
- Reads/writes blocks of cache from/to disk in parallel using dedicated threads.

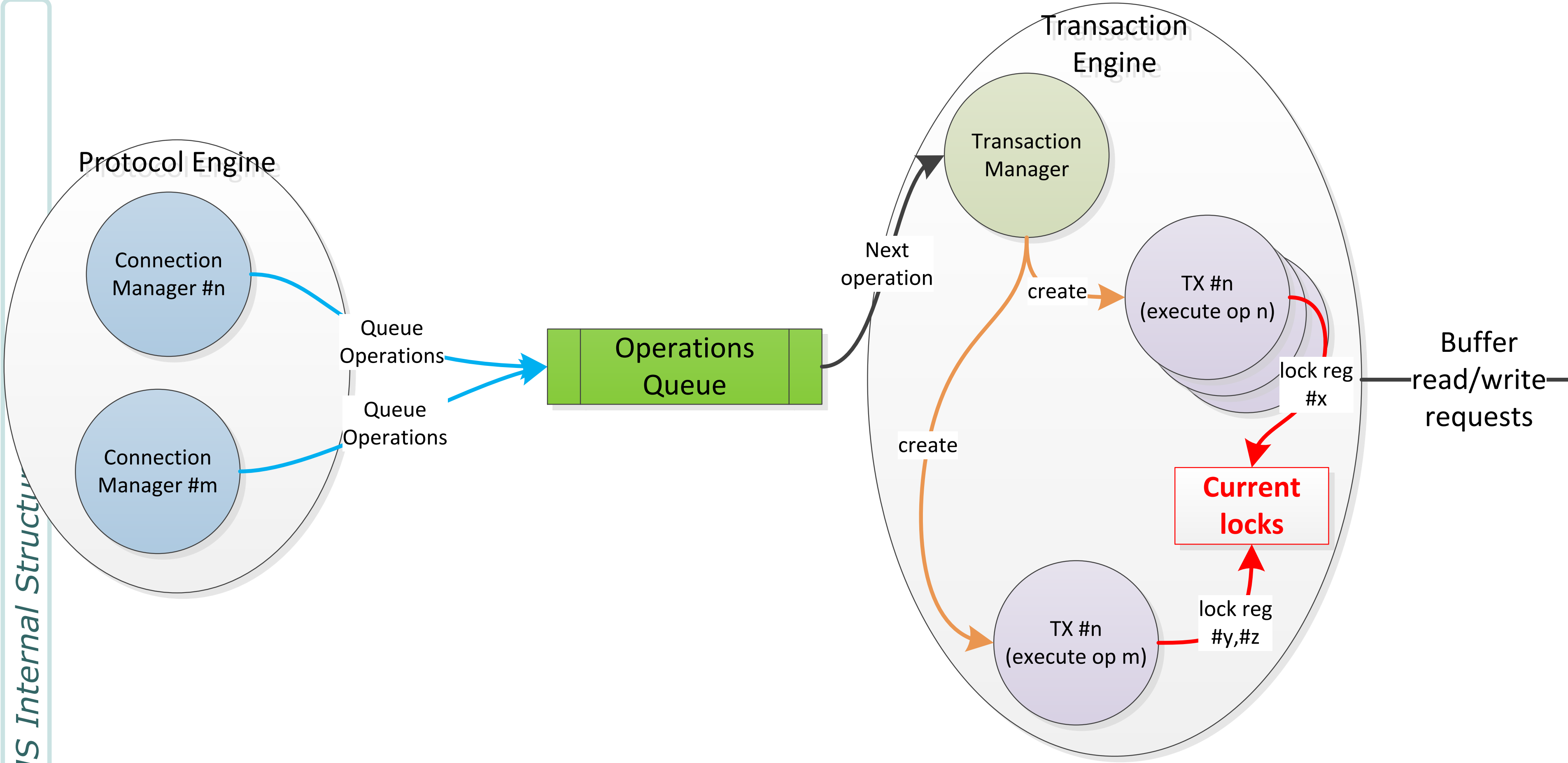


Protocol Engine: Session Manager

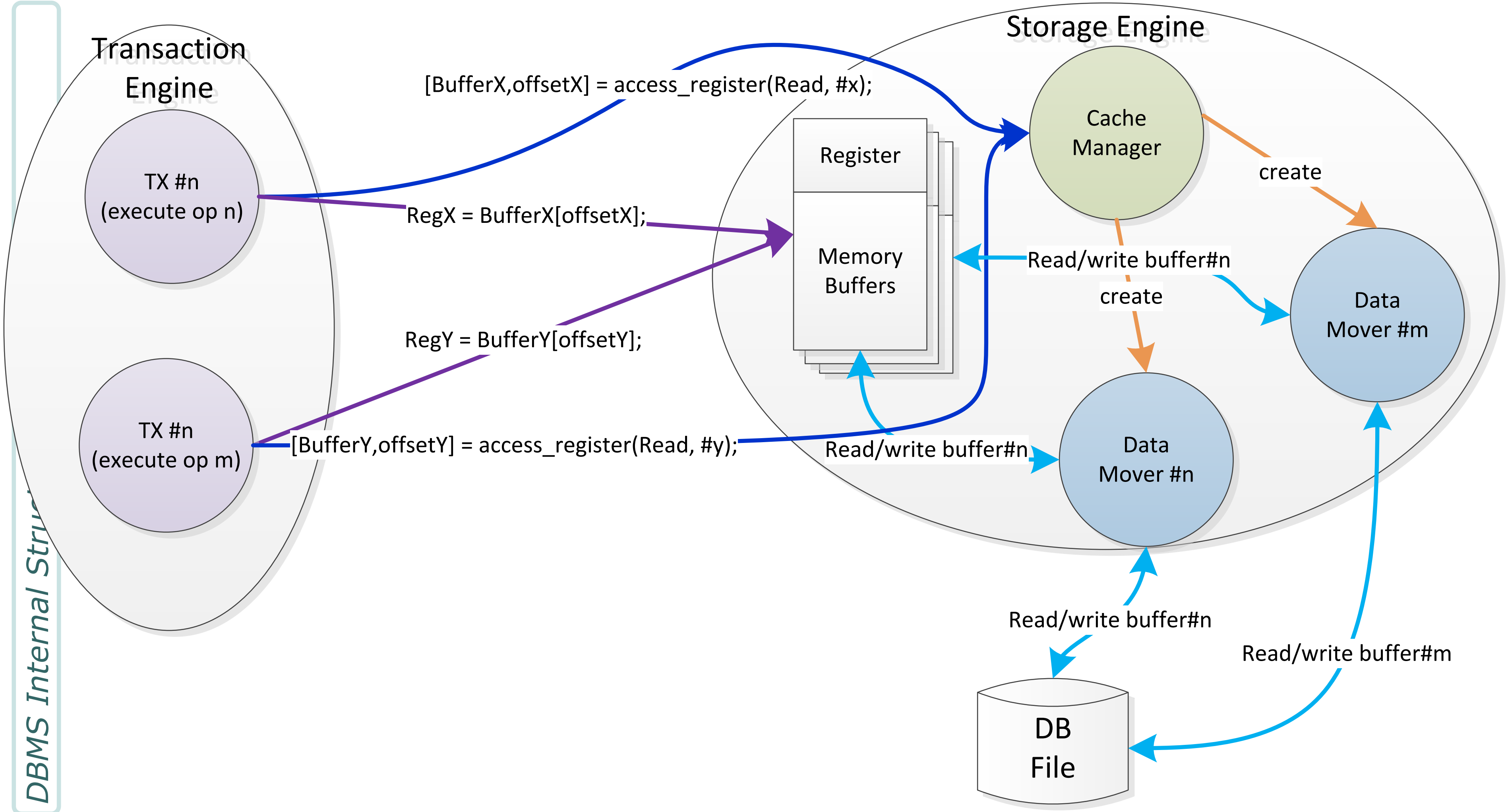


Transaction Engine

DBMS Internal Structu



Storage Engine



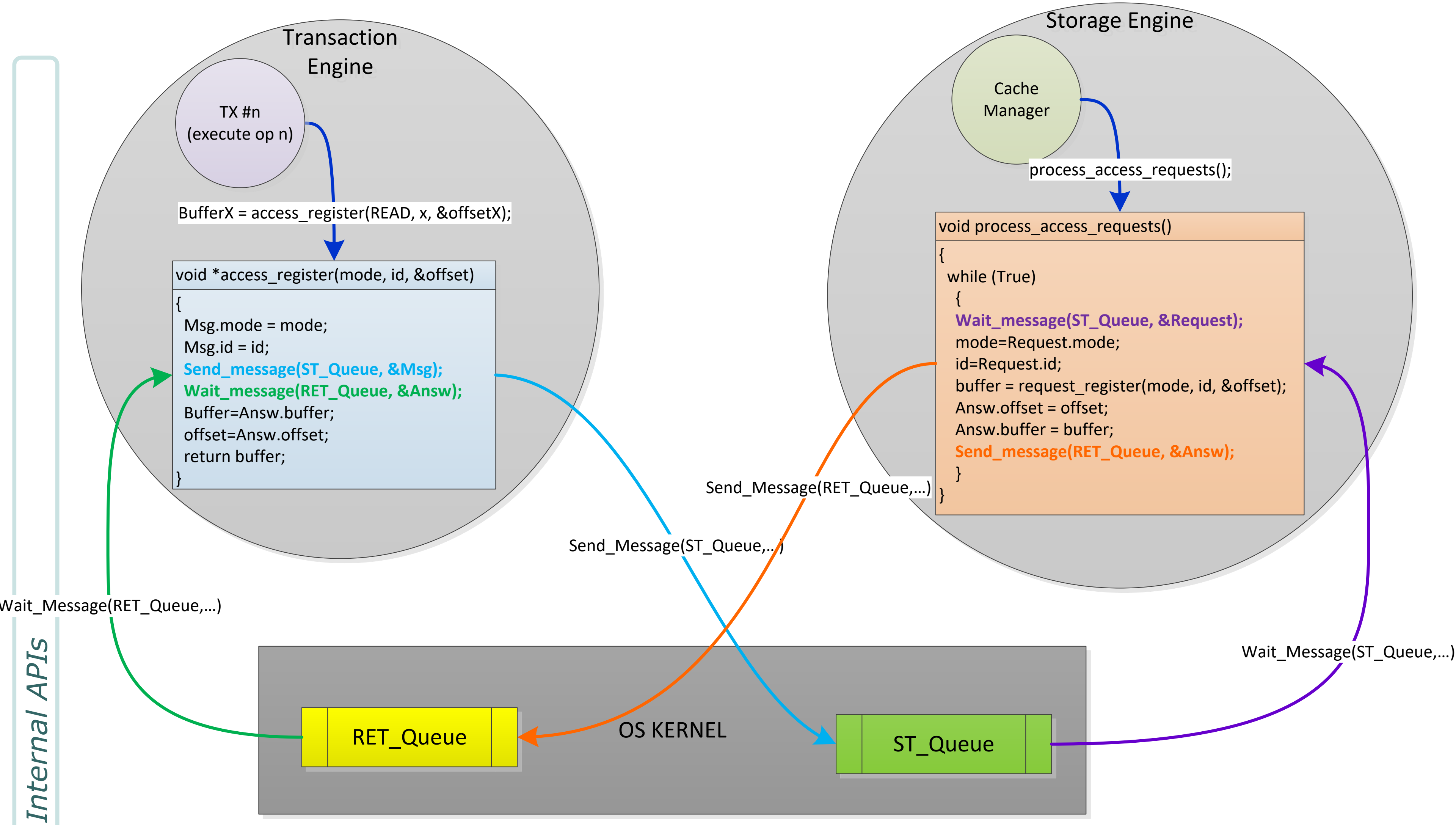
INTERNAL APIS

Need for internal APIs

- Each layer (engine) needs to provide an API for its client layer.
  - The API is a **client library** which can be used to program the upper layer.
    - It only contains the minimal set of functions needed to dialog with the engine.
  - The implementation of the API is done by the lower layer.
  - The interaction between the client library and the lower layer is implemented using **communication services** of the OS like:
    - Semaphores
    - Message queues
    - Shared memory

How to implement an API on OS

Internal APIs


Transaction Engine
TX #n (execute op n)
BufferX = access_register(READ, x, &offsetX);
Storage Engine
Cache Manager
process_access_requests();
Send_Message(RET_Queue,...)
Send_Message(ST_Queue,...)
Wait_Message(RET_Queue,...)
Wait_Message(ST_Queue,...)
RET_Queue
OS KERNEL
ST_Queue

```
void *access_register(mode, id, &offset)
{
  Msg.mode = mode;
  Msg.id = id;
  Send_message(ST_Queue, &Msg);
  Wait_message(RET_Queue, &Answ);
  Buffer=Answ.buffer;
  offset=Answ.offset;
  return buffer;
}
```

```
void process_access_requests()
{
  while (True)
   {
    Wait_message(ST_Queue, &Request);
    mode=Request.mode;
    id=Request.id;
    buffer = request_register(mode, id, &offset);
    Answ.offset = offset;
    Answ.buffer = buffer;
    Send_message(RET_Queue, &Answ);
   }
}
```

Proposed functions for APIs

- Protocol engine API (DB protocol)
  - Used to write client applications

```
struct Connection *con;
int status;
char *sql_cmd=“select * from T;”
struct Results *results_ptr;

con = connect(serverip);
status = execute_SQL(sql_cmd, &results_ptr);
disconnect(con);
```

Internal APIs

Proposed functions for APIs

- Transaction engine API
  - Used to write the session manager of Protocol Engine.

```
struct tQueue *tx_queue, *tx_queue;
Struct tOperation *operation;
struct tResults *result;

tx_queue = get_opqueue();
rx_queue = new_rxqueue(sessionid);
queue_operation(tx_queue, operation, rx_queue);
wait_result(rx_queue, &result);
delete_rxqueue(rx_queue);
```

Proposed functions for APIs

- Storage engine API
  - Used to write the execution thread in Transaction Engine.

```
tMode mode;
int regid;
struct Register *buffer;
long offset;

mode = REG_WRITE;
mode = REG_WRITESYNC;
mode = REG_READ;
status = access_register(mode, regid, &buffer, &offset);
buffer[offset] = register;
status = release_register(regid);
```