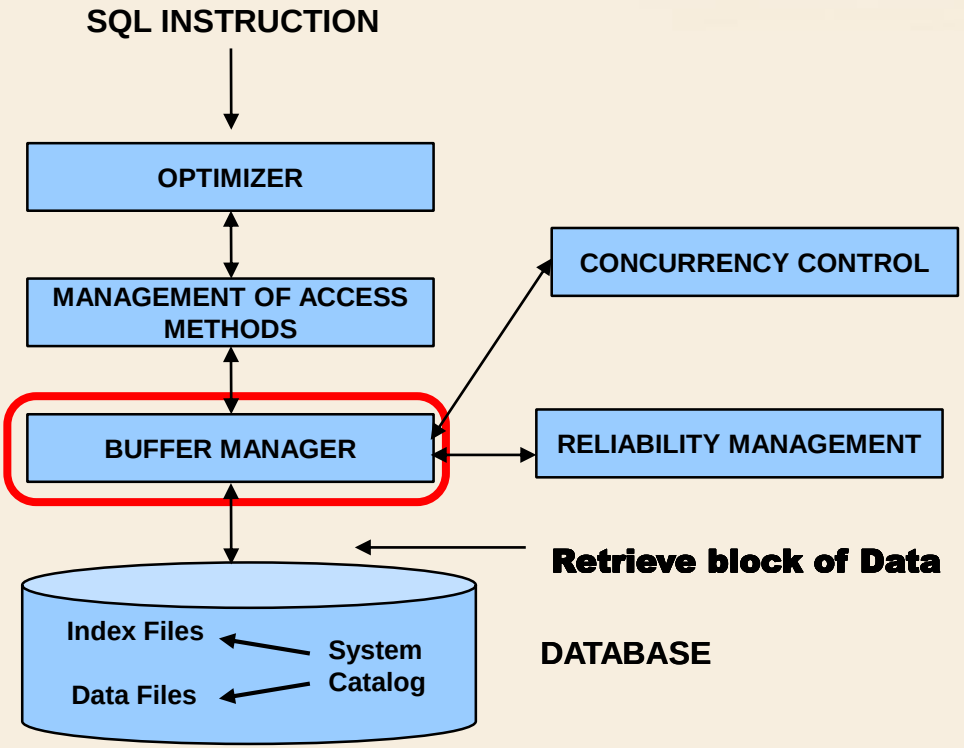
Buffer manager

DBMS Architecture



Buffer Manager

- Buffer Manager
  - It manages page transfer from disk to main memory and vice versa
  - It is in charge of managing the DBMS buffer
- Efficient buffer management is a key issue for DBMS performance

DBMS buffer

- Buffer
  - A large main memory block
  - Pre-allocated to the DBMS
  - Shared among executing transactions
- Buffer organization
  - Memory is organized in pages
  - The size of a page depends on the size of the operating system I/O block

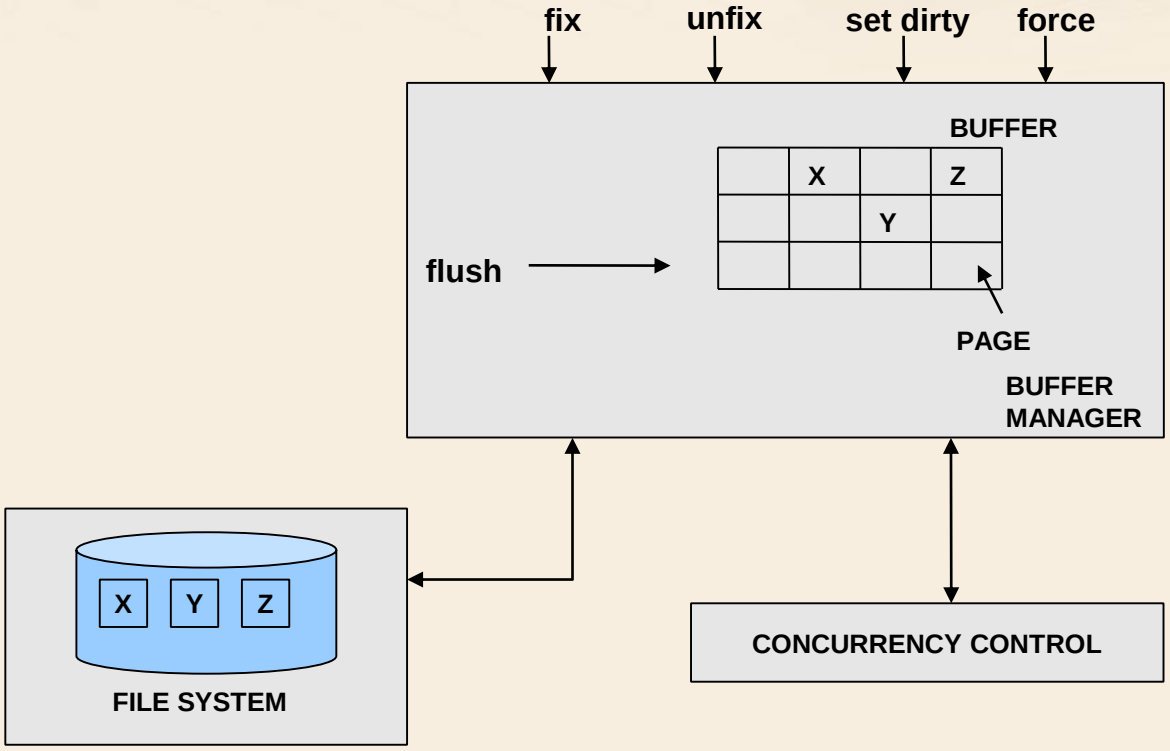
DBMS buffer

- Memory management strategies
  - Data locality
    - Data referenced recently is likely to be referenced again
  - Empirical law: 20-80
    - 20% of data is read/written by 80% of transactions

DBMS buffer

- The Buffer Manager keeps additional "snapshot" information on the current content of the buffer
- For each buffer page
  - Physical location of the page on disk
    - File identifier
    - Block Number
  - State variables
    - *Count* of the number of transactions using the page
    - *Dirty bit* which is set if the page has been modified

Buffer Manager Architecture



Buffer Manager

- Provides the following primitives to access methods to load pages from disk and vice versa
  - Fix
  - Unfix
  - Force
  - Set dirty
  - Flush
- Requires shared access permission from the concurrency control manager

Fix primitive

- Used by transactions to require access to a disk page
  - The page is loaded into the buffer
  - A pointer to a page into the buffer is returned to the requesting transaction
- At the end of the Fix primitive, the requested page
  - Is in the buffer
  - Is valid (i.e, allocated to an active transaction)
  - The Count state variable of the page is incremented by 1
- The Fix primitive requires an I/O operation only if the requested page is not yet in the buffer

Behavior of the Fix primitive

- The Fix primitive looks for the requested page among those already in the buffer
- If it finds the requested page
  - It returns to the requesting transaction the address of the page in the buffer
    - It happens often because of data locality

Behavior of the Fix primitive

- If it does not find the requested page
  - A page is searched into the buffer where the new page can be loaded
    - First, among free pages
    - Next, among pages which are not free, but with Count=0
      - called victim pages
      - may still be locked
    - If the selected page has Dirty=1
      - it is synchronously written on disk
  - The new page is loaded in the buffer and its address is returned to the requesting transaction

Unfix primitive

- It tells the buffer manager that the transaction is no longer using the page
  - The state variable Count of the page is decremented by 1

Set dirty primitive

- It tells the buffer manager that the page has been modified by the running transaction
  - The dirty state variable of the page is set to 1

Force primitive

- It requires a *synchronous* transfer of the page to disk
  - The requesting transaction is suspended until the Force primitive is executed
  - It always entails a *disk write*

Flush primitive

- It transfers pages to disk, independently of transaction requests
  - It is internal to the Buffer Manager
  - It runs when the CPU is not fully loaded
    - In CPU idle time
  - It downloads pages which
    - are not valid (state variable Count=0)
    - are not accessed since a longer time

Buffer Manager writing strategies

- Steal
  - The Buffer Manager is allowed to select a locked page with Count=0 as victim
    - The page belongs to an active transaction
- No steal
  - The Buffer Manager is not allowed to select pages belonging to active transactions as victims
- The steal policy writes on disk *dirty pages* belonging to *uncommitted* transactions
  - In case of failure these changes *must be undone*
    - same operations as in transaction rollback

Buffer Manager writing strategies

- Force
  - All active pages of a transaction are *synchronously* written on disk by the Buffer Manager during the commit operation
- No Force
  - Pages are written on disk *asynchronously* by the Buffer Manager
    - by means of the Flush primitive
- Pages belonging to a *committed* transaction may be written on disk *after commit*
  - In case of failure these changes *must be redone*

Buffer Manager writing strategies

- Typical usage is steal/no force, because of its efficiency
  - No force provides better I/O performance
  - Steal may be mandatory for queries accessing a very large number of pages

File System and Buffer Manager

- The Buffer Manager exploits services provided by the file system
  - Creation/deletion of a file
  - Open/close of a file
  - Read
    - It provides a direct access to a block in a file
    - It requires
      - File identifier
      - Block number
      - Buffer page where to load data in memory

File System and Buffer Manager

- Sequential Read
  - It provides sequential access to a fixed number of blocks in a file
  - It requires
    - File identifier
    - Starting block
    - Count of the number of blocks to be read
    - Starting buffer page where to load data in memory
- Write and Sequential Write
  - Analogous for writing data
- Directory management functions