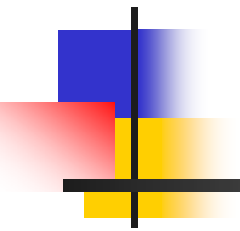
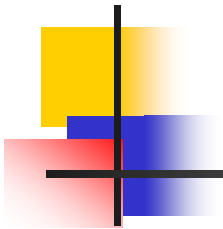


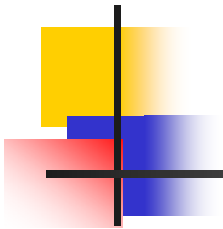
# Representing Data Elements

Chapter 12 of GUW



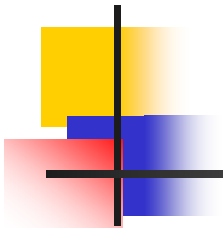
# Objectives

- To understand:
  - How SQL data types are represented as fields
  - How tuples are represented as records
  - How records are organized in blocks of memory
  - How to handle variable size records
  - How to cope with a record when it size changes as a result of modifying some of its fields



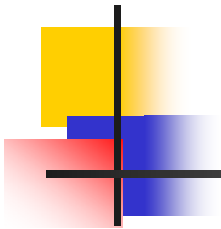
# - Lecture outline

- Representing SQL data types
- Fixed-length Records
- Representing Block and Record Addresses
- Variable-length Records
- Record modifications
- Summary
- References



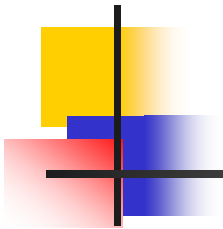
# - Representing SQL data types

- Representing:
  - Numbers
  - Strings
  - Dates and times
  - Bits
  - Boolean
  - Enumerated types



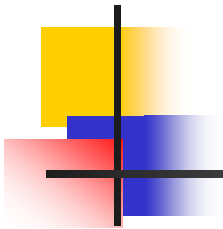
# -- Representing Numbers

- INTEGER:
  - Represented as Bit strings
  - 2 to 4 bytes long
- FLOAT
  - Represented as Bit strings
  - 4 to 8 bytes long



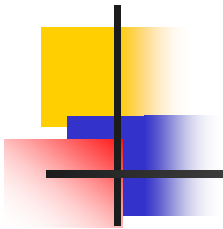
# -- Representing Strings

- Fixed sized:
  - The SQL type CHAR(n)
    - Sized n bytes
      - Example: A → ASCII code A → 8 bits
    - Filled with special pad character if assigned less than n characters.
- Variable sized
  - The SQL type VARCHAR(n) == (VARCHAR2(n) of Oracle)
    - Can hold m <= n characters
    - Sized m+1 bytes, if m < 256
    - Represented as:
      - Length plus content
      - Null-terminated string



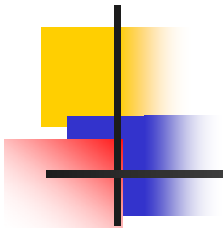
# -- Representing Dates and Times

- Date
  - Fixed-length character string as CHAR(n)
- Time
  - May include fractions of a second
  - Represented as:
    - variable-sized string as VARCHAR(n) which is of limited precision;
    - True variable-length value as will be discussed later on the chapter.



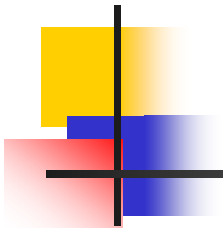
# -- Representing BITS, BOOLEAN, Enumerated types

- BITS(n)
  - ROOF(n/8) bytes
- BOOLEAN
  - TRUE as 00000000
  - FALSE as 11111111
- Enumerated types
  - Typically represented as integers 0, 1, 2, ... n.
  - Example:
    - {Sat, Sun, Mon, Tue, Wed, Thu, Fri} can be represented as {00000001, 00000010, 00000011, ..., 00000110}



# - Fixed-Length Records

- Records and Database Schema
- Building Fixed-Length  Records
- Record Headers
- Packing Fixed-Length Records into Blocks



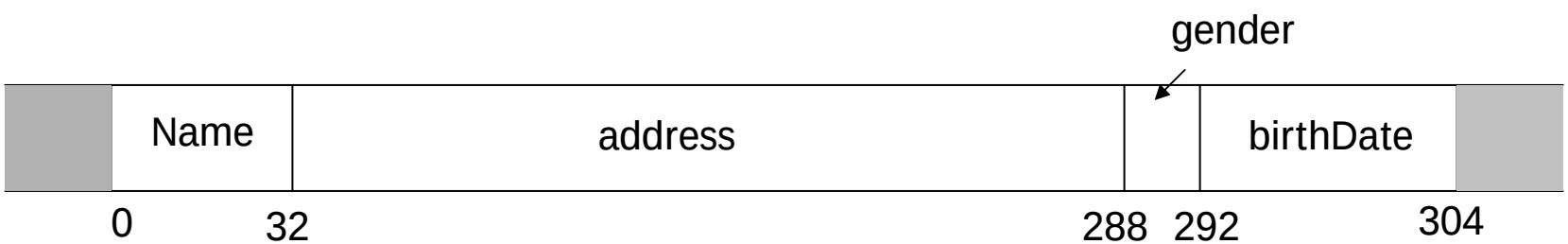
# -- Records and Database Schema

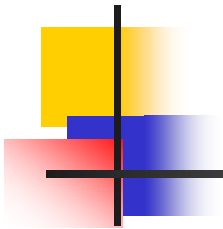
- Fields are grouped together to form records
- A database record must conform to a schema
- The schema includes:
  - Names and types of each field in the record
  - Their offset within the record

# -- Building Fixed-Length Records

```
CREATE TABLE MovieStar (
    name        CHAR(30),
    address     VARCHAR(255),
    gender      CHAR(1),
    birthDate   DATE );  // assume takes 10 bytes
```

- Thus, a record of MovieStar takes 30+256+1 + 10 = 297 bytes
- Some data types or for performance reasons: each field or record starts at an address which is a mulltiple of 4 or 8.



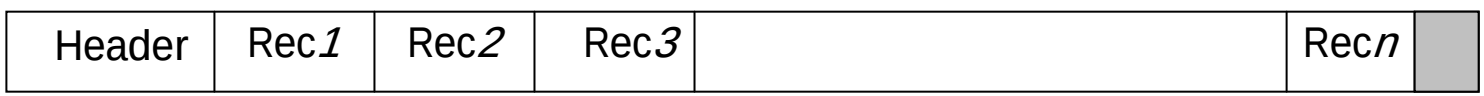


# -- Record Header

- Including data a record contains a header which may have:
  - The record schema, more likely a pointer where the DBMS stores the schema the record.
    - To check, the types pf the attributes and their constraints.
  - The record length
    - Can be computed but done for performance reason
  - Timestamps:
    - Time the record was last modified
    - Time the record was last read.
      - Used for transaction management

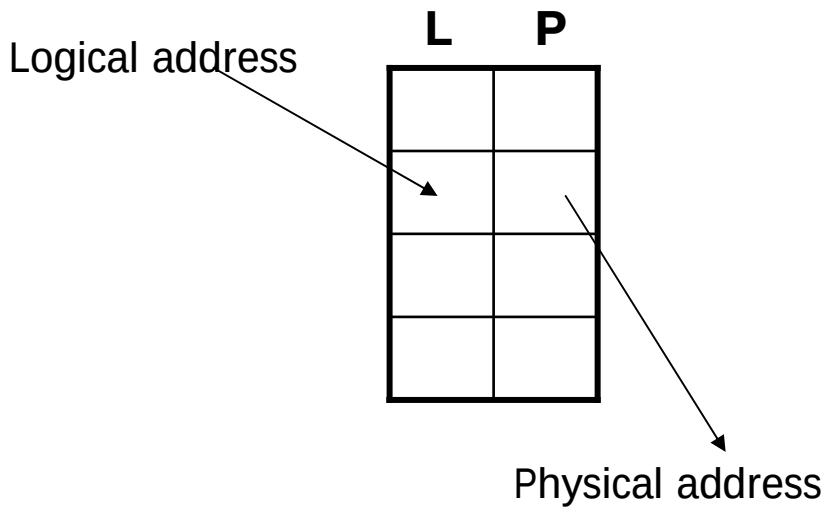
# -- Packing Fixed-Length Records into Blocks

- A number of records are stored in a block.
- Each block can have an optional header which holds:
  - Links to other blocks
  - The role of the block (E.g. Data or index block)
  - The name of the object to which the block belong
  - A “directory” giving the offset of each record in the block.
  - Block ID
  - Timestamp: Indicating the time of the block’s last modification and/or accessed.

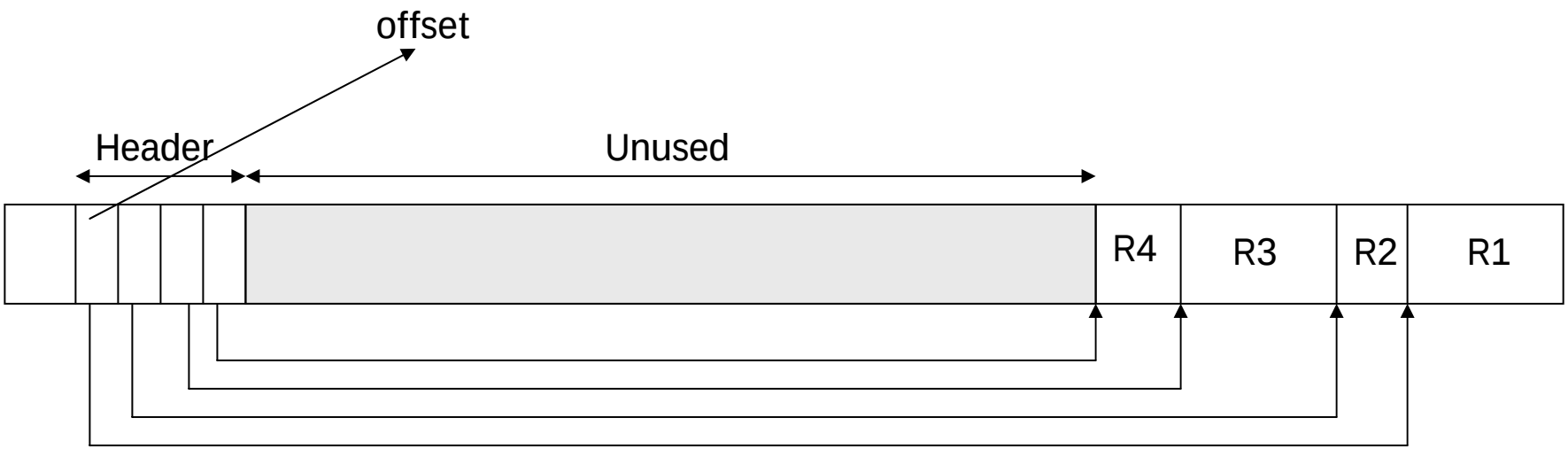


# - Representing Block and Record Addresses ...

- Physical Address of a record
  - Host address
  - Disk ID
  - Cylinder number
  - Track number
  - Block number
  - Record offset within the block
- Logical address
  - Each block or record has a logical address.
  - A table is need to map logical to physical



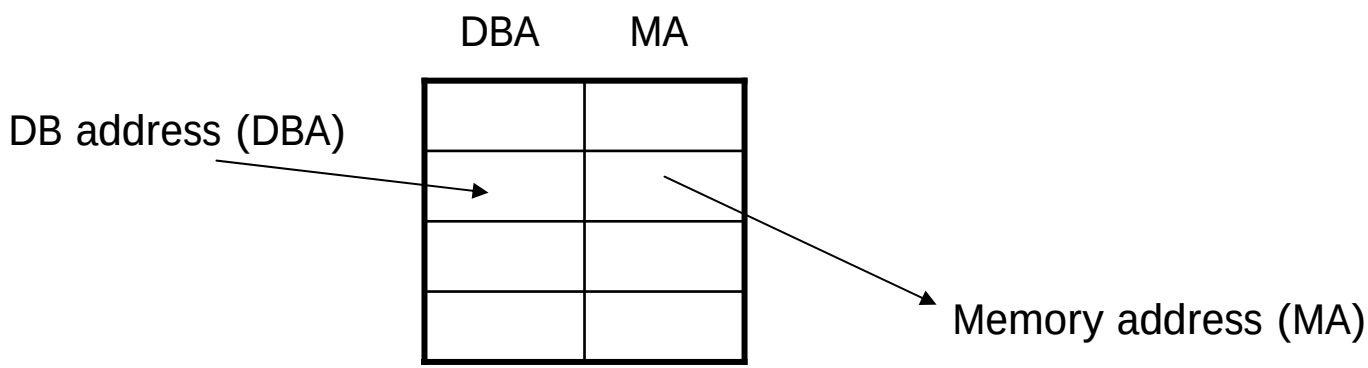
# ... - Representing Block and Record Addresses



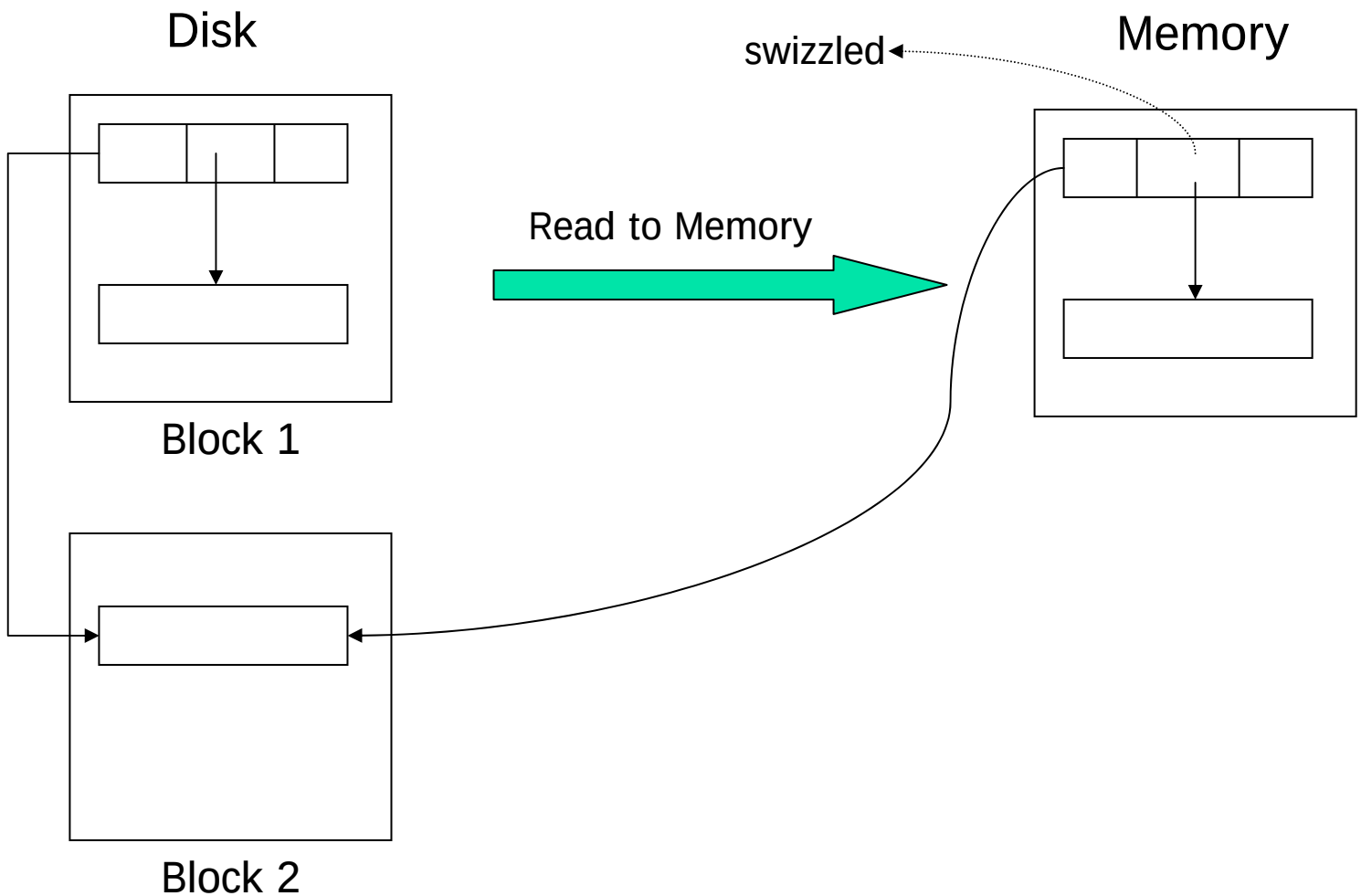

A block with a table of offsets telling the position of each record in the block.

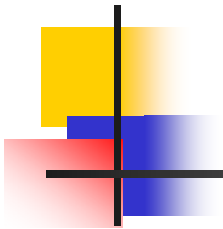
# -- Pointer Swizzling ...

- Data has two forms of addresses
  - Database address:
    - Its address in the disk
  - Memory address:
    - Its address in the memory.
- When a data moves from disk to memory a transilation table is used to map its disk address to its memory address.
- A bit can be used as indicator of the type of address: DBA or MA



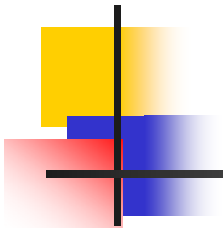
# ... -- Pointer Swizzling ...





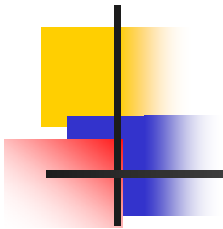
# ... -- Pointer Swizzling ...

- Strategies when to swizzle:
  - Automatic swizzling
    - As soon as the block comes to memory.
  - Swizzling on demand
    - Swizzle a particular address when its is requested
  - No swizzling
    - Keep pointers in their DBA form.
    - Use the translation table
  - Programmer control of swizzling
    - Explicitly done by the programmer.



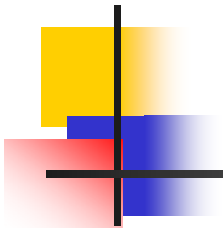
# ... -- Pointer Swizzling ...

- When returning blocks to disk:
  - Pointers within that block must be unswizzled
  - This is done using the DBA in the address translation table.
- Some blocks are pinned
  - They can not be written back to disk until unpinned.
  - Pinning is done for performance reasons
- Make sure not to follow dangling pointers. So appropriate clean up is needed.
  - If block B is written back to disk. All pointers pointing to block be must be unswizzled.



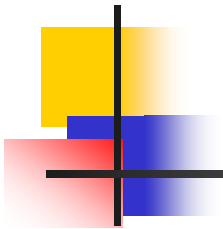
# - Variable-Length Records

- Reasons why records not always have the same size:
  - Fields of variable length. Attribute content vary in size.
  - Repeating fields. An attribute that appear several times, but how many times is not specified by the schema. (Not allowed in Relational)
  - Records of variable format. When different tuples in a relation have different sets of attributes. E.g., if many attributes have no content. (Not allowed in relational)
  - Enormous fields. Data like movies and pictures in the relation. The record may not fit into one block.



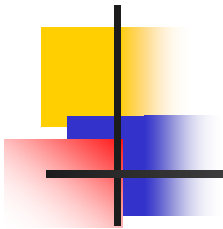
# -- Fields of Variable Length

- When a field has variable size we still have to be able to find all fields in the record. Since the offset cannot be read from the relation schema some extra information is stored in the record header.
- Example of how it can be solved:
  - Store fixed length fields first in the record.
  - Store the total size of the record.
  - Store offsets for variable sized fields (except the first).
- Frequently null fields:
  - Are represented by null pointer
  - Keep them at the end of the record



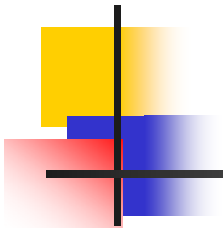
# -- Spanned records

- A record is called spanned record if it is split between two or more blocks.
- Reasons for spanned records:
  - Space utilization.
  - Records larger then block.
- For each fragment of a record extra information on where to find next and previous fragment is needed.



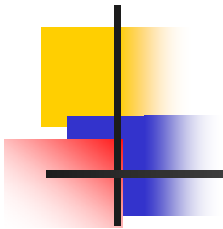
# -- BLOBS

- Binary, Large OBjectS = BLOBS
- BLOBS can be images, movies, audio files and other very large values that can be stored in files.
- Storing BLOBS
  - Stored in several blocks.
  - Preferable to store them consecutively on a cylinder or multiple disks for efficient retrieval.
- Retrieving BLOBS
  - A client retrieving a 2 hour movie may not want it all at the same time.
  - Retrieving a specific part of the large data requires an index structure to make it efficient. (Example: An index by seconds on a movie BLOB.)



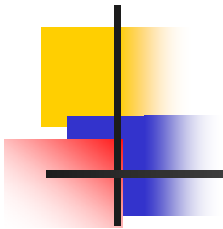
# - Record Modification ...

- We will look at three types of updates:
  - Insertions of new tuples
  - Deletions of tuples
  - Tuple updates
- What problems may arise when updates are performed on the database?
- Think of the different situations where we have:
  - fixed length vs. variable length tuples
  - no order vs. sorted tuples



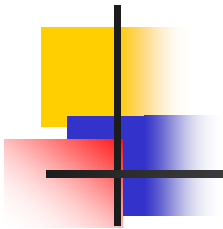
# ... - Record Modification

- Insert
  - No order: No problem, just find a block with enough space or use a new block.
  - Fixed order: May be a problem if there is not enough room in the correct block. Solutions:
    - Find space in nearby block and rearrange.
    - Create a overflow block.
- Delete
  - Pack data in the block to prepare for new inserts. Remove overflow blocks. if possible. Leave a tombstone if there may be pointers to the record.
- Update
  - Fixed length: No problem.
  - Variable length: Same as for insert and delete. (But no tombstones.)



# - Summary

- Fields
  - organizing bits
    - Numbers, Date, Bollean, etc
    - Field terminators
    - Counters
- Records
  - organizing fields
    - header
    - Fixed Vs Variable
    - null
    - Modification
    - Spanned
- Blocks
  - organizing records
    - Header
- Files
  - arranging blocks (to be discussed in the next chapter)



# - References

- Chapter 12 of GUW.

END