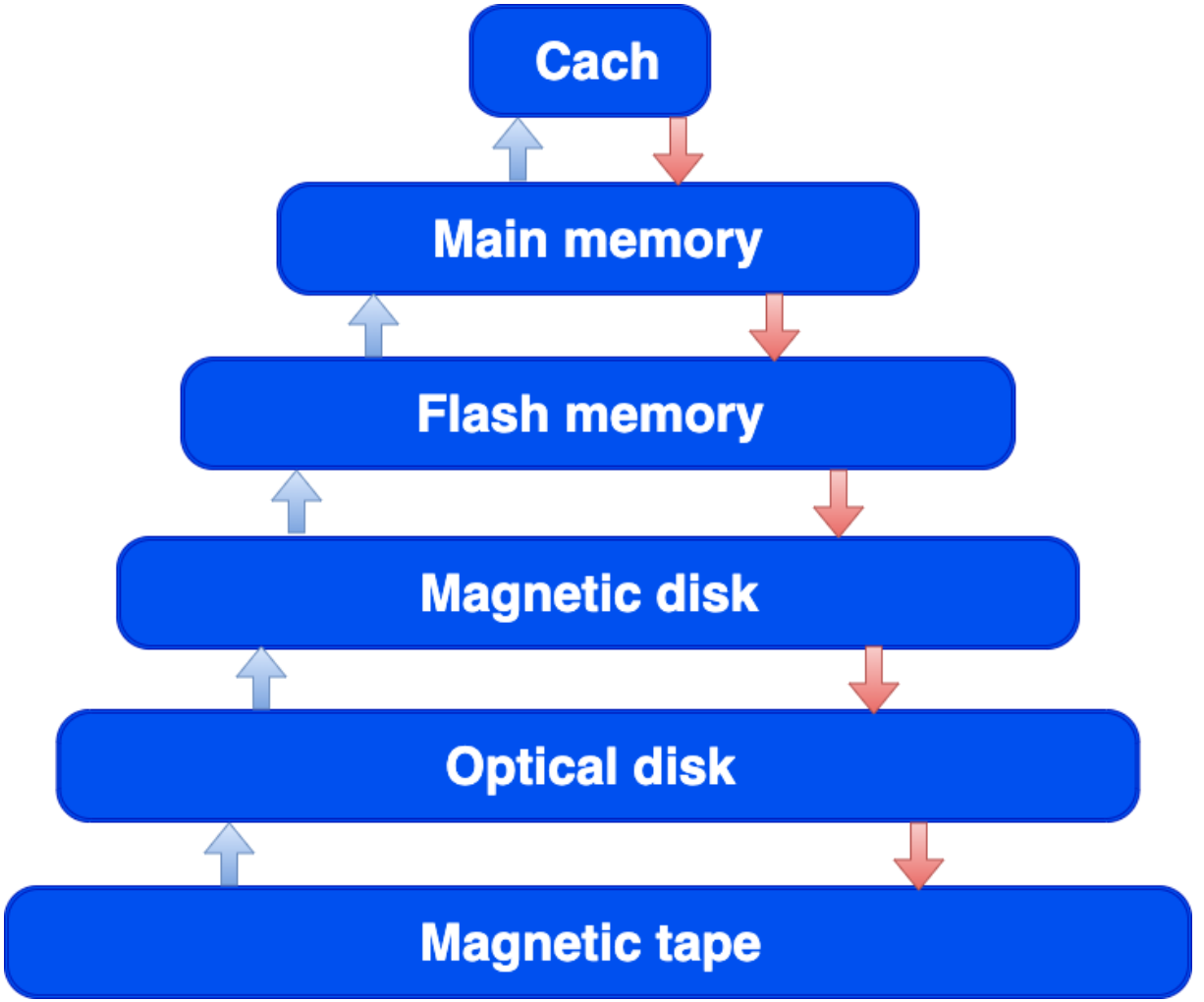
STORAGE STRUCTURE

Storage hierarchy



Overview of Physical Storage media

- In general holds: the faster, the smaller and more expensive.

Standard storage systems:

- Cache - non permanent
  - located on the CPU
  - very fast and normally used for pipelining and prefetching
- Main memory - non permanent
  - today mostly between 4 and 32 GB
  - sometimes offers the possibility to store entire databases in the main memory
- Hard drives - permanent
  - capacity in terabytes
  - Special type flesh memories: Solid State Drives (SSD) - often used between DRAM and Hard drives
    - NAND technology devices gradually replace Hard disk drives

Overview of Physical Storage media

Computer storage media can be divided in two categories

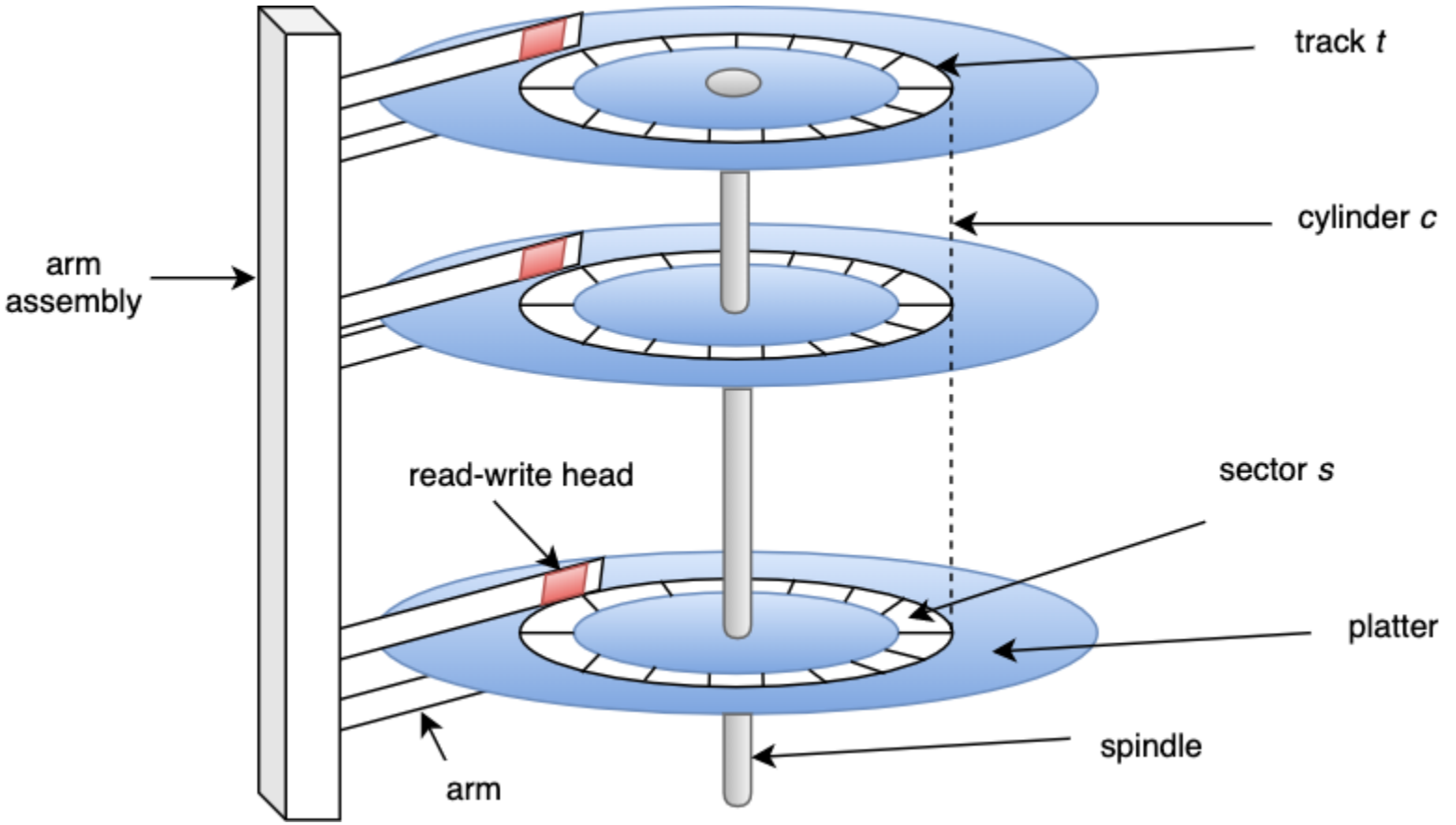
- **primary storage** - can be operated directly by CPUs - **direct access**
  - provides quick access to data but has limited storage capacity (main memory, cache memories)
- **secondary and tertiary storage** - data cannot be processed by CPUs but has to be transferred to the primary storage - **block-addressable**
  - secondary storage includes hard disk drives and flash memories
  - tertiary storage includes optical disks and tapes

Databases are used for permanent storage of data

- bigger databases are too big to be stored in the main memory
- secondary storage is more safe than primary storage
- costs for the secondary storage are lower than those for the primary storage

In our lectures, we will be focused on databases on secondary storage media

Hard drives as a secondary storage media



Physical characteristics of magnetic disks

Hard disk drive (HDD) - parts

Hard disk drives are storage mediums that physically consists of more disk **platters** which have a circular shape

- disk surface is logically divided in **tracks**
- tracks of all platters with the same diameter are called a **cylinder**
- tracks are divided in **sectors**
- *sector* is the smallest unit of information that can be read or written (4096 bytes as of 2011 - hardware block size)
- Information is recorded on the surface which is covered with a magnetic material
- **read-write** heads are mounted on a single assembly called **disk arm**
- current HDDs have platters that spin mostly at speeds 5400 or 7200 rotations per minute.

Operation of HDDs

Average Access Time consists of the following three elements

- **seek time** - moving read-write head to the position (6-8ms)
- **latency time** - waiting for the right sector to come under the head (approximately. 2-3ms)
- **block transfer time** - time to retrieve or store data to the disk
  - block transfer time is usually much shorter than seek and latency time (~0.5 ms)

Data are transferred to the main memory in **blocks**

- Transferring non-consecutive blocks - implies seek time + latency time for each new block (Random I/O)
  - this random access on disks is slow
- Transferring several consecutive blocks (called a *cluster*) from the same track or cylinder is more efficient (Chained I/O)
  - allocating more blocks at the same time

Buffering of Blocks

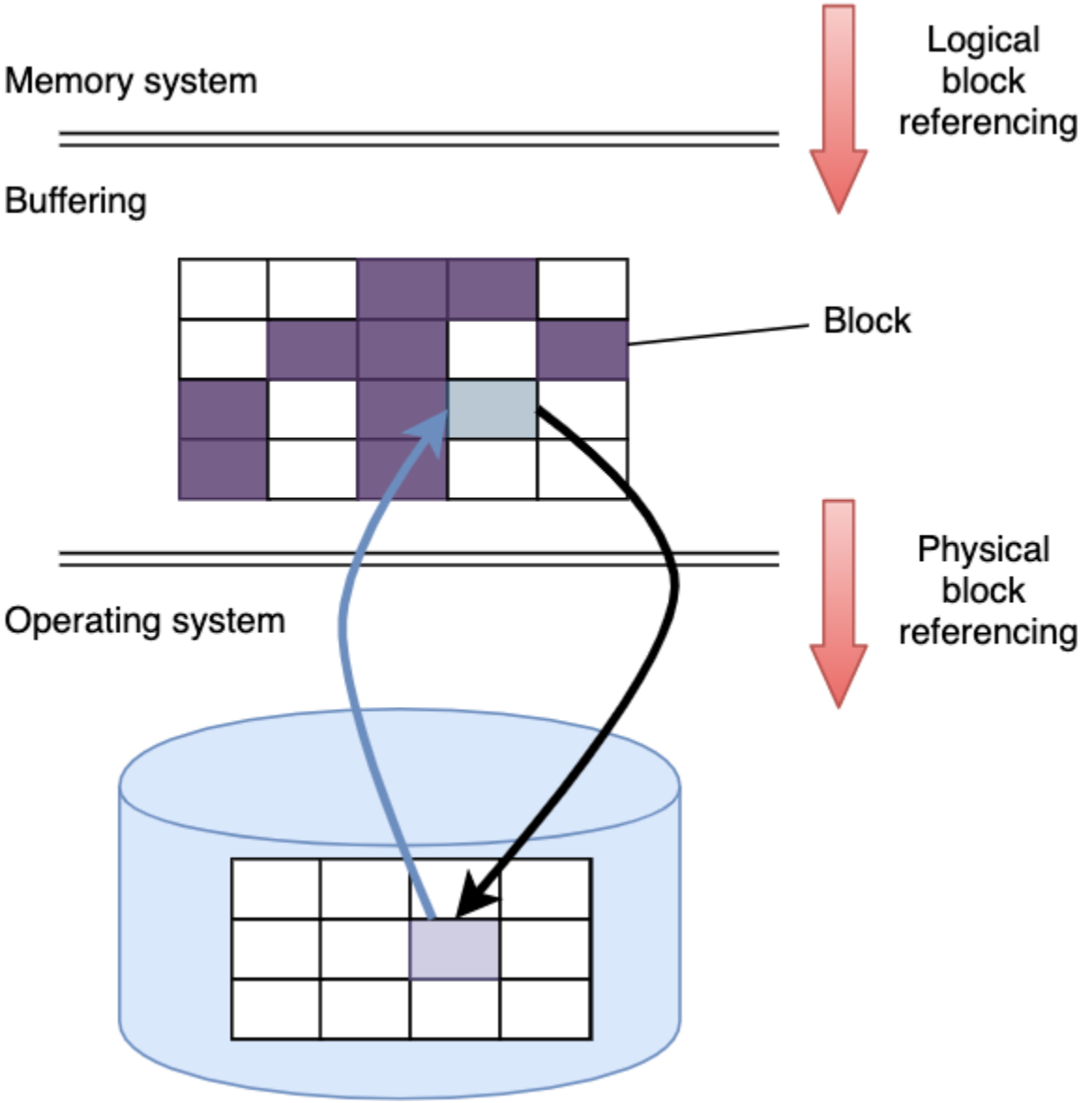
Blocks are transferred from platters to the buffer

- buffer is a part of the main memory reserved to speed up the transfer
- currently buffers have size mostly from 512 to 8192 bytes (8KB)
  - whole buffer (buffer pool) size can be adjusted to match the size of the cluster
- e.g. data are first changed in the buffer and then written on the HDD

Double buffering represents a process in which CPU can do in parallel two activities

1. process a block in the main memory
2. read and transfer the next (another) block into a different buffer

Buffering


Memory system
Logical block referencing
Buffering
Block
Physical block referencing
Operating system

Records, Blocks (Pages)

Databases keep data stored in records.

- records correspond to tuples in the relational model
- records(tuples) are sequences of bytes
  - DBMS has to have a functionality to read attributes and values from those records

Relational tables are saved in **files** consisting of records

- records are stored in blocks (pages) because it is more efficient to transfer blocks than to transfer individual records between the main memory and secondary storage

File records on Disk

Record is a collection of related data values or items.

- *value* is formed of one or more bytes and corresponds to a **field**
- **record type** is a collection of field names and their corresponding data types

Data types are standard data types used in programming (integer, floating point, boolean, etc.). Special data types are used to store large values:

- **BLOB** (bynary large objects) -used for storing data items that consists of large unstructured objects - (images, audios, videos, etc.)
  - in PostgreSQL, types *LO* (large object) and *BYTEA* are used instead of BLOB
  - typically stored separately in a pool on disk with a **pointer** which is just included in the record
- **CLOB** (character large object) - *TEXT* - corresponding type for large text in PostgreSQL

Data types

Data types to store character strings are usually:

- CHAR - fixed-length character strings
- VARCHAR - variable-length character strings

Data types for time:

- TIME, DATE, TIMESTAMP - in PostgrSql, number of seconds elapsed since January 1, 1970 (Unix epoch)

Integer numbers are represented using:

- SMALLINT - small-range integer (2 bytes)
- INTEGER - typically used (4 bytes with range from -2^{31} to $+2^{31}$)
- BIGINT/LONG - large range integer with 8 bytes

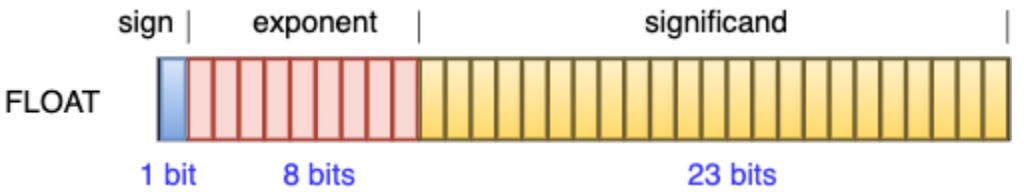
Real numbers can be represented using:

- floating point numbers(with variable precision)
- fixed point numbers (with fixed precision)

Numeric data types for real numbers

Variable precision numbers store data as specified by the IEEE-754 standard

- used to store approximate values - $value = significand \times (base)^{exponent}$
- operations typically faster than fixed precision numbers (problems with rounding errors)



- FLOAT (4 bytes),
- REAL/DOUBLE (8 bytes)

Fixed precision numbers

- used when round errors are not acceptable
- stored typically as a variable binary representation with additional metadata
  - similar to varchar
- examples: NUMERIC/DECIMAL
- recommended for storing monetary amounts

Database files

DBMS stores database as one or more files on disk

A **file** is a sequence of records. All records in the file are mostly of the same record type.

- each relation attribute is represented by a field of the record
  - there are cases when more record types are stored in the same file

There are in general two types of records:

- **Fixed-length records** - all records in a file have the same size
- **Variable-length records** - different records in a file can have different size

Records belonging to both types can be stored on disks in different ways

Fixed and Variable Length Records

Fixed Length Record

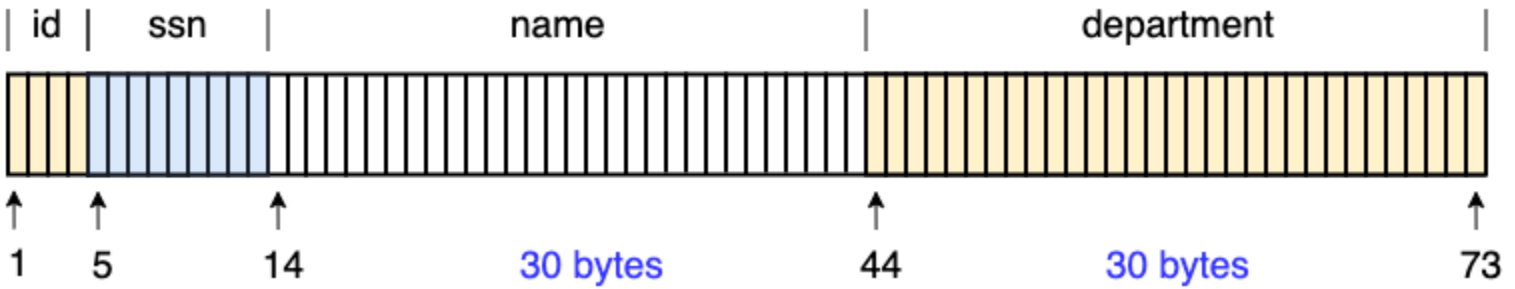
```
struct student {
        int id,
        char ssn[9];
        char name[30] ;
        char department[30];
  }
```

Variable Length Record

```
struct student{
    int id,
    char ssn[9];
    char name[30] ;
    char* department;/*Pointer*/
}
```

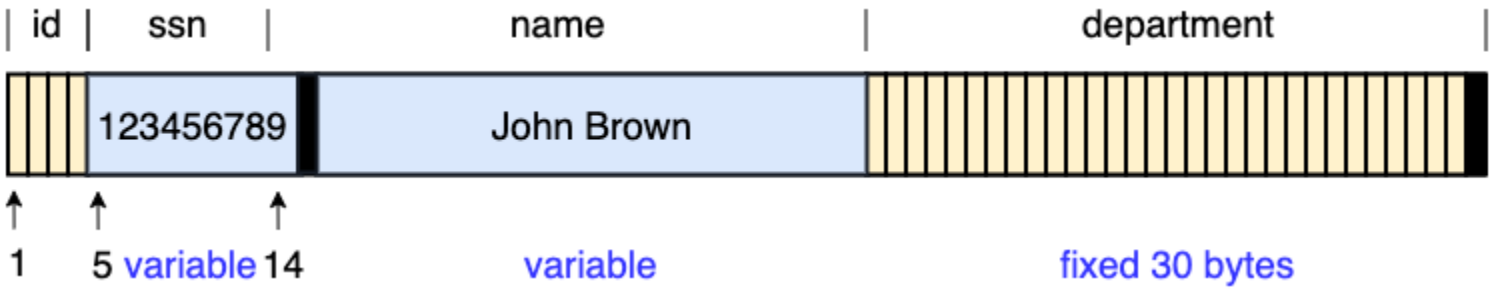
Analogy to the fixed and variable records in programming languages

Fixed-length record example



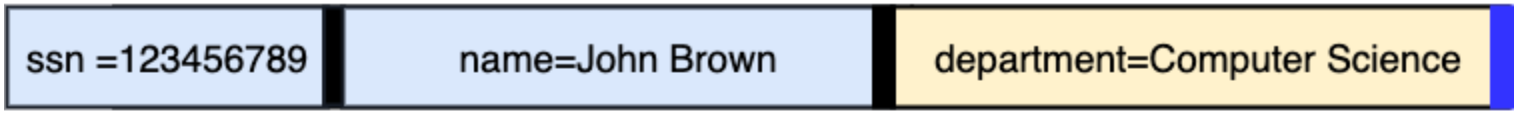
- fixed-length record with 4 fields and the total record size of 73 bytes.
- location of fields easily determined by programs (relative to the starting position)
- organization is not flexible
- space is wasted if many fields are optional but empty
  - fields are optional if they can contain NULL values

Variable-length record - type 1



- variable-length record with 2 fixed fields and 2 variable-length fields.
- two variable-length fields are divided with a special **separator character**
- separator character is a character that doesn't appear in any field
- access time is longer
- space is not wasted
- number indicating the length of the field (in bytes) can be sometimes stored in front of the field value

Variable-length record - (key-value pairs)



- separator character

- terminates record

= - separates field name and value

Variable-length record type with many *optional fields*

- fields stored in the form **<field-name><field-value>**
- useful for records with large number of optional fields
  - typical record can have just a small number of non-empty fields

In the example we use three types of separator characters

- even two types of separator characters are enough

File organization, Blocks (Pages)

File organization refers to the organization of data into records and blocks in databases files

- good file organization has a goal to locate a block (or blocks) containing the desired records with a minimal number of block transfers

Block (page) is a *unit of data transfer* between main memory and disk

- each block has a fixed size and a unique identifier
- blocks can contain tuples, indexes, metadata, etc.
- block (page) size can be 1 - 16 KB (can be different from the hardware block size 4KB)
  - PostgreSQL - block with the size of 8KB
  - Oracle 4KB, MySQL 16KB

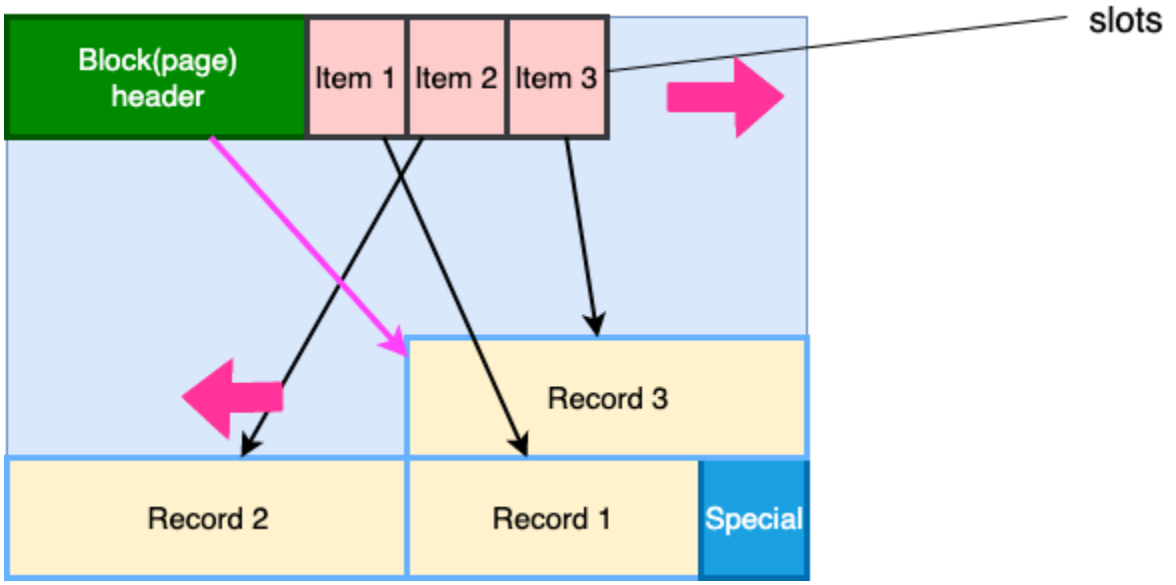
Block in PostgreSQL file organization

Each table is an array of blocks (called pages or slotted-pages) of a fixed size (typically 8KB)

– all pages are logically equivalent

– heap structure is used to store files

– e.g. simplified page structure (slotted pages)



– page header contains checksum, start of the unused space, end of unused space, etc.

PostgreSQL file organization

Each table is stored in a separate file

- database cluster (**PGDATA**) contains all databases (this is not an allocation cluster)
- each database has an **OID** (Object Identifier) - which can be found by the query:

```
select oid, datname from pg_database
```

- database files (system catalogs) are stored in files in the subdirectory $PGDATA/base/ < oid >$
- if table exceeds the size of 1 GB then it's divided into different files of size of at most 1GB

PostgreSQL file organization - large values

To store records larger then block size PostreSQL uses

- *The Oversized-Attribute Storage Technique (TOAST)*
- Large objects facility

TOAST is a mechanism used to store records that have size greater than 8 KB (block upper limit) using overflow blocks

- each table has TOAST table which can be used to store big columns
  - column size in a record cannot be greater than 2KB
- big columns are split into 2 KB chunks and stored in blocks of TOAST tables
- represents an alternative to the spanned organization of blocks
- total column size with TOAST is limited to 1 GB

Large object facility allows values up to 4 TB

File header

File header (file descriptor) stores information which are needed by the system programs that access file records. It contains information such as:

- information to determine disk addresses of the file blocks
- record format descriptions
  - for fixed-length unspanned - length and order of fields
  - for variable-length records - field type codes, separator characters, records-type codes
- these information facilitate the process of the search for the desired record

Allocating records to blocks

Records of a file are allocated to disk blocks in different ways

– assume that B represents block size and R fixed-length record size where $B \geq R$ then we can define **blocking factor** (bfr) as a ratio:

$$bfr = \left\lfloor \frac{B}{R} \right\rfloor$$

In terms of dealing with the unused space organization can be:

Unspanned organization

– records do not span over multiple blocks - unused space is left untouched

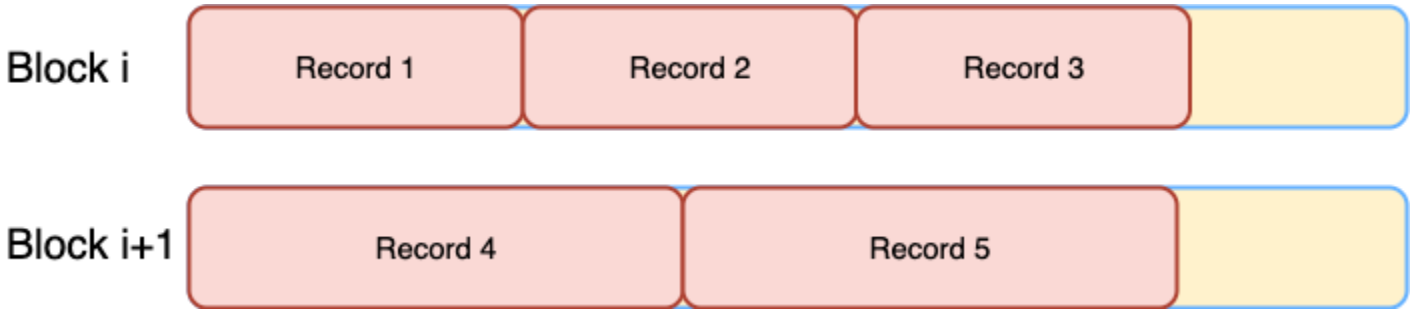
– suitable for fixed length records

Spanned organization

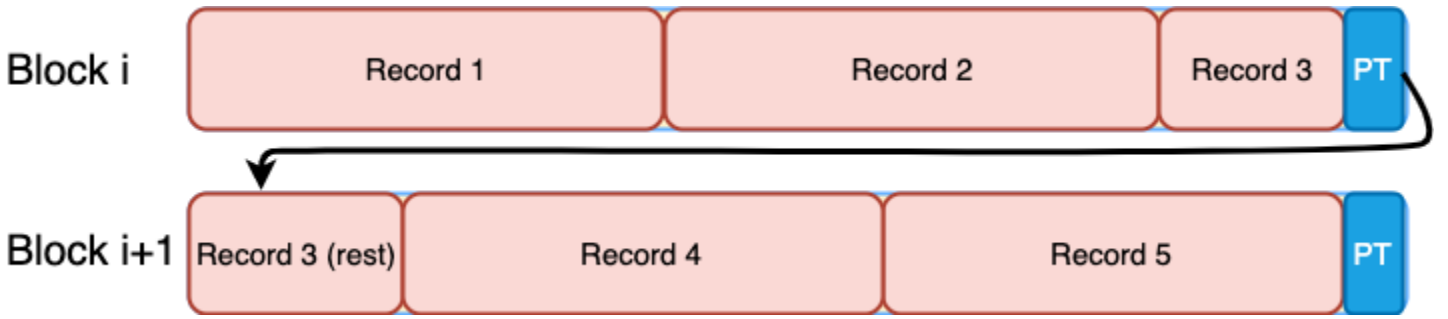
– Unused space of the block(page) is used to store part of the record and pointer which points to the block containing remainder

Unspanned and spanned organization of blocks

Unspanned organization



Spanned organization



Block organization and record types

Considering *fixed-length records*

1. suppose that $B \geq R$ and R does not divide B exactly
   – unspanned organization speeds up processing but leaves in each block unused space of the size:

$$B - (bfr \cdot R)$$

   – spanned organization saves space and part of the record is stored in remaining space together with the pointer

2. suppose $B < R$ - organization *must* be spanned

Considering *variable-length records*

– *blocking factor* (*bfr*) defined as the average number of records per block

– organization can be either spanned or unspanned

– number of blocks *b* for a given file with *r* records can be calculated as $b = \left\lceil \frac{r}{bfr} \right\rceil$

Block organization - example

Suppose that fixed-length records with unspanned blocks are used to store a file with:

B = 1024 and R = 150

Then we have:

$$bfr = \left\lfloor \frac{1024}{150} \right\rfloor = 6$$

Unused space left in the block after it's filled with 6 records:

$$B - (bfr \cdot R) = 1024 - (6 \cdot 150) = 124 \text{ bytes}$$

How many of these blocks is required to store the file with $r = 1550$ records?

Allocating File Blocks on Disks

Contiguous allocation

- file blocks are allocated to the consecutive disk blocks
- good for reading (especially with double buffering) and bad for adding new records

Linked allocation

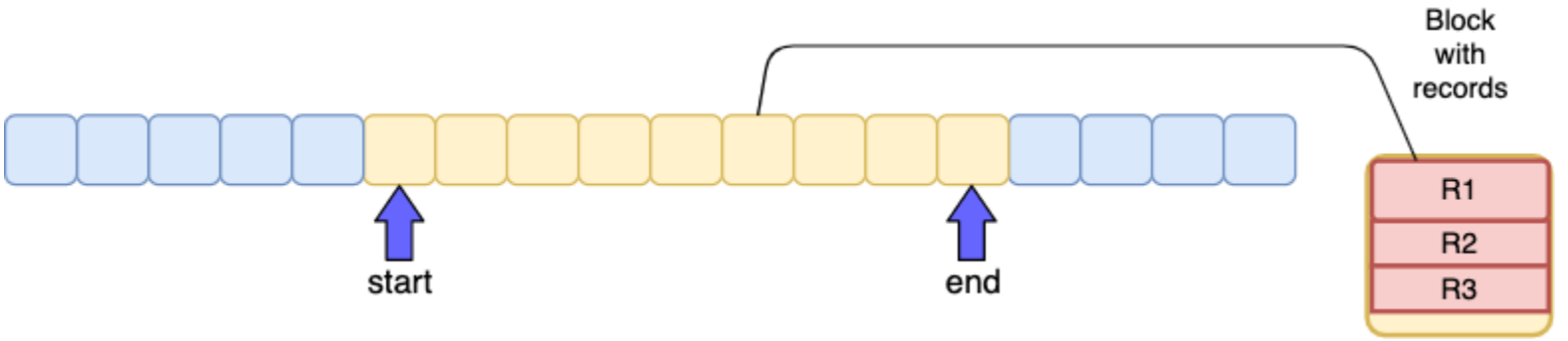
- each block contains a pointer to the next file block
- good for adding new records (file expansion)
- reading is slower

Clustered allocation

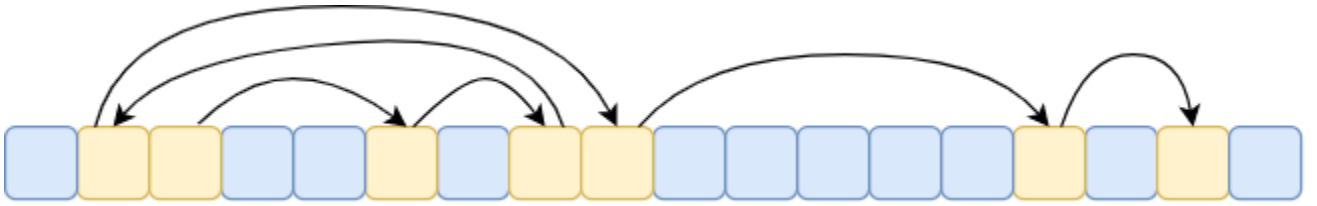
- represents a combination of the two previous allocation types
- clusters contain consecutive disk blocks
- clusters are related using links
- many different combinations with cluster organization

Allocating File Blocks on Disks

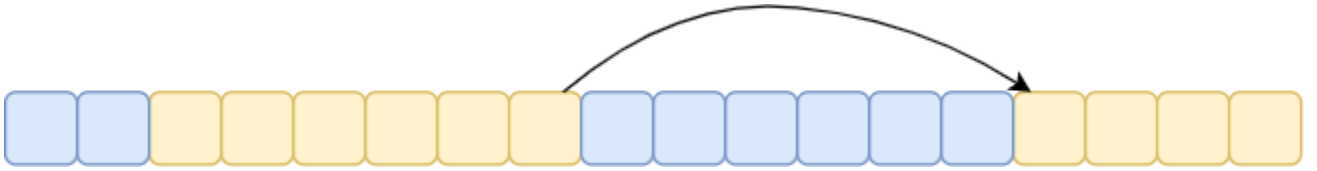
Contiguous allocation



Linked allocation



Allocation in clusters



Operations on files

Operations on files are grouped in:

- **retrieve operations**
- **update operations**

In any operation, **selecting** of one or more records is performed:

- based on a **selection condition**
- selection condition is decomposed by DBMS into simple selection conditions
- simple conditions are used to locate records on disk

Access operations

Typical operations used by DBMS for locating and accessing records in files are grouped in:

- **open** - prepares file for reading and writing
- **reset** - setting the file pointer to the beginning of the file
- **find (locate)** - searches for the first record satisfying search condition (current record), transfers the block into a buffer (if it's not already there)
- **read (get)** - copies the current record from the buffer into a program variable
- **findNext** - searches for the next record satisfying the selection conditions
- **delete** - deletes the current record and (eventually) updates the file on disk
- **insert** - inserts a new record in the suitable block and transfers the block into the main memory buffer
- **close** - completes the file access, releases the buffers and performs necessary cleanup operations

File organizations and access methods

A **primary file organization** represents physical organization of data in records, blocks and access structures

We focus on the following primary file organizations :

- **Heap files** files of unordered records
- **Sorted files** files of ordered records
- **Hashed files** files which use hashing techniques
- **B-trees** - files organized as B-trees

An **access method** provides a group of operations which can be applied to a file (as those described on the previous slide)

- specific access methods depend on the file organization
- index access methods, for instance, can be applied only on files with indexes

A **secondary organization** allows efficient access using auxiliary access structures in addition to those used for primary file organization.

Heap files - files of unordered records

A **heap** or **pile** represents a basic type of organization

- records are placed in the data file in the order in which they are inserted
- insertion of new records at the end of the file
- often used with additional **access paths** such as *secondary indexes*

Inserting of a new record is very efficient (append operation)

- the last block is copied in the buffer and the new record is added
- if there is no more space in the last block a new block is created and record is inserted in it
- the last block is then rewritten back to disk
- the address of the last block is updated in the file header if necessary
- operation has constant complexity (*O(1)*)

Search and delete operations

Searching for a record in the heap is an expensive operation

- if there is no index, the *linear search* is necessary
- in a file with *b* blocks it requires searching $\frac{b}{2}$ blocks, on average
- complexity is therefore linear $O(n)$ to the number of blocks (b) of the heap file

Deleting a record includes at first an operation of searching, then after the block is brought to the buffer

- deleting the record from the block in the buffer and reorganization of the rest of block records (if necessary)
- rewriting the block back to the disk
- if the deleted record was the last in the block, the operation requires removing entire block and updating the address of the last block in the file header
- complexity $O(n)$ because we exploited the search operation

Delete operation and fragmentation of heap files

Due to delete operation, files can become very fragmented (there are many empty records in blocks) and this further results in wasted storage space

Alternative deletion technique to tackle this problem:

- introducing an extra byte or bit, called **deletion marker** which is stored within each record
- search programs consider only valid records
- inserting new records in place of those which are deleted
- deleted records can be restored if other records are not already put on their place
- can require maintaining the list of these gaps in the blocks and the size of those gaps

reorganization of blocks is necessary for both deleting techniques

- blocks are accessed consecutively and packed again to the full capacity

Heap files

Heap can use either spanned or unspanned organization

For heaps with fixed-sized records and unspanned organization i-th record of the file can be found in block:

$$\left\lfloor \frac{i}{bfr} \right\rfloor$$

- blocks are here counted 0,…,b-1 and rows are counted 0,…,r-1
- additionally, i-th record takes the position $(i \mod bfr)$ in it's block
- however this i-th record is the record of the unsorted file and it **does not speed up** search for i-th record in any order but can help addressing of indices

Sorting of heap files requires external sorting which is computationally very expensive operation

Sorted files - files of ordered records

An **ordered file** (sorted file) has records physically ordered on disk based on the values of one of the fields called **ordering field**

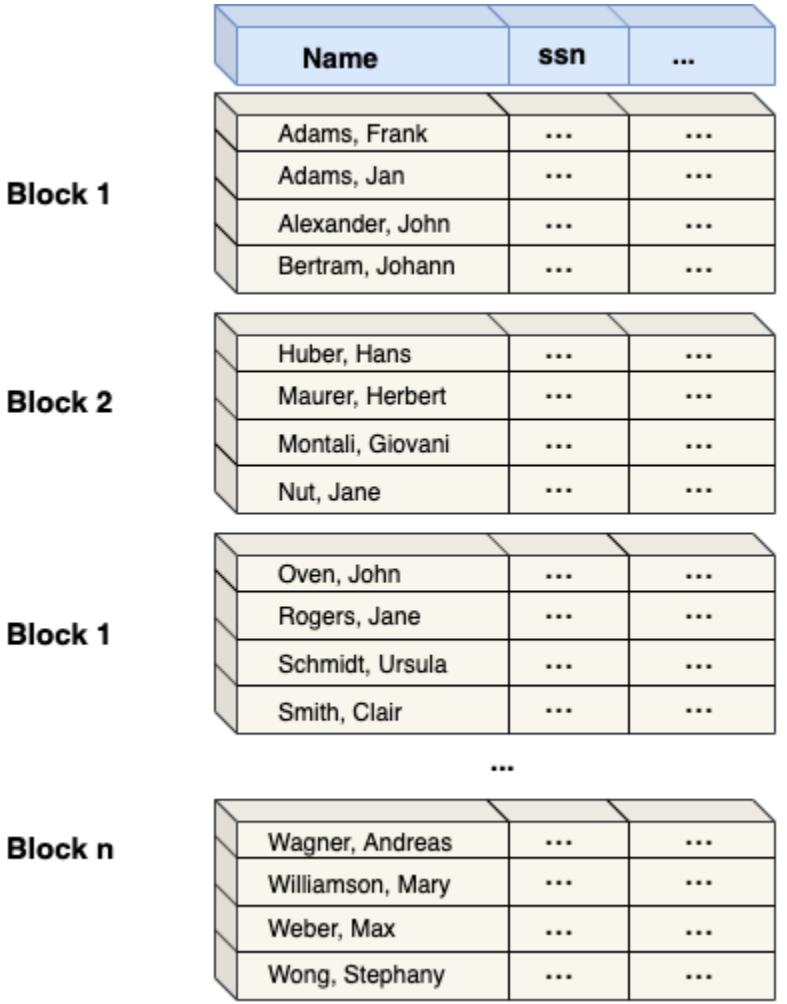
- ordering field is called **ordering key** if it is a *key field*

Advantages of sorted files

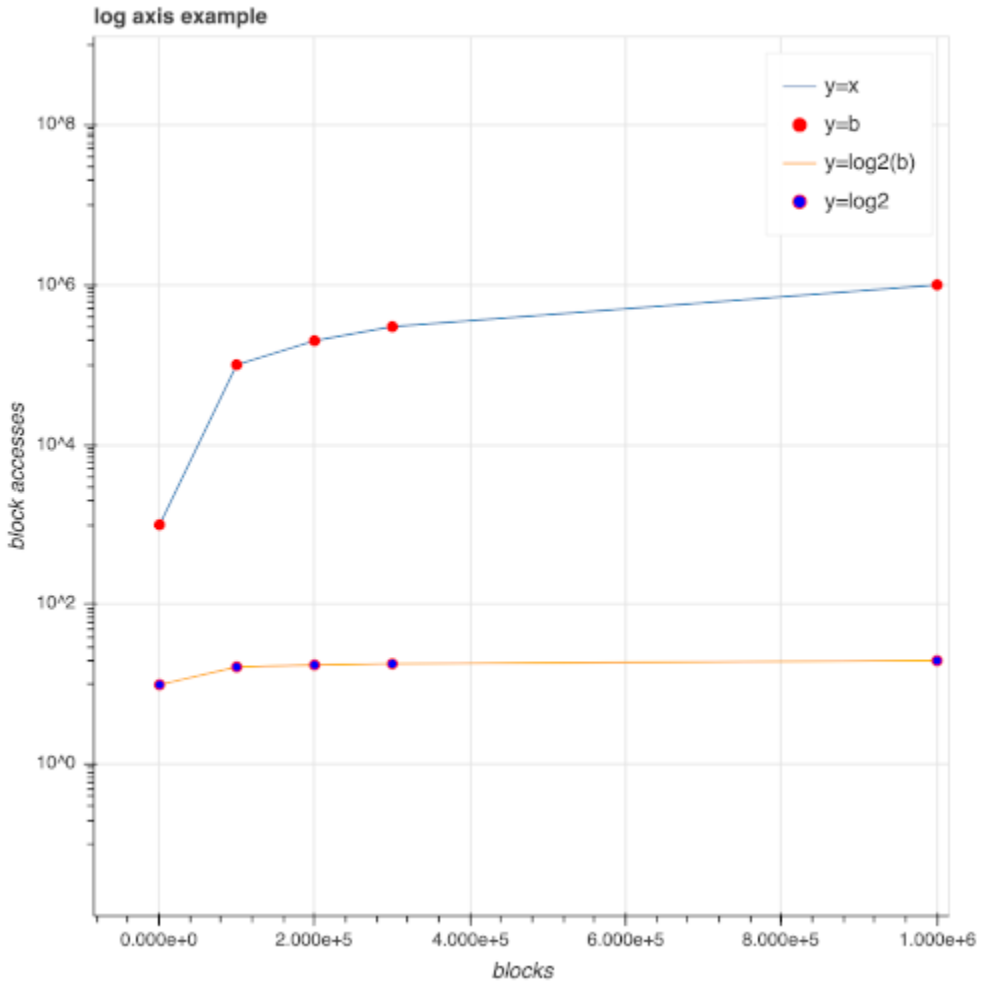
- searching for a record with the condition on the ordering key requires $log_2(b)$ block accesses and has complexity $O(log(n))$ (n is the number of file blocks)
  - big improvement in comparison to the linear search which requires even b accesses when record is not found (worst case)
- reading values in the order of ordering keys is extremely efficient
  - finding the next record is mostly reading from the same block

Sorted files

File sorted according to the primary key “Name”



Binary vs. linear search



comparison of binary and linear search on logarithmic axis

– maximal value for 1 million blocks - log(1000000) = 14 block accesses

Sorted files - challenges

Disadvantages of sorted files:

- Inserting and deleting records are *expensive operations*
  - to insert or delete a record, on the average, half of the file blocks have to be read and written
- sorted files do not provide any advantages for searching on nonordering fields
  - linear search has to be used as in the case of heap files

Improvements:

1. keeping some unused space for new records in each block
2. keeping an **overflow** (**transaction**) file besides the main file
   - new records are stored at the end of the overflow file and not in the main file

Organizations with improvements require periodical **reorganization**

- overflow file is sorted and merged with the master file

Comparison of main operations

	Heap files	Sorted files
Search operation	accessing all files $O(n)$ (on average b/2) in the worst case b blocks	binary search O(log b)
Insert operation	O(1) - effective	O(n) blocks has to be read (on average b/2)
Delete operation	$O(n)$- the record has to be found first.	when ordered file is searched than it's less expensive
Sorting	very expensive	very expensive except on the ordering field which is already sorted

OLTP and OLAP

Two classes of database workloads (online processing systems) are particularly important: OLTP and OLAP

OLTP (Online transaction processing)

- primary goal is data processing
- simple queries that read or update small amount of data related to a single entry
- supports transaction-oriented applications, e.g. day to day transaction of an organization, bank transactions
- **row-based** database organization

OLAP (Online Analytical Processing)

- primary goal data analysis
- complex queries that read large portions of the database related to multiple entries (with many joins)
- supports business decision making (Business intelligence)
- financial reporting, forecasting, data warehouse system are examples of OLAP
- **column-based** database organization

Column-based databases

Row-based databases organize data by rows (n-ary storage model)

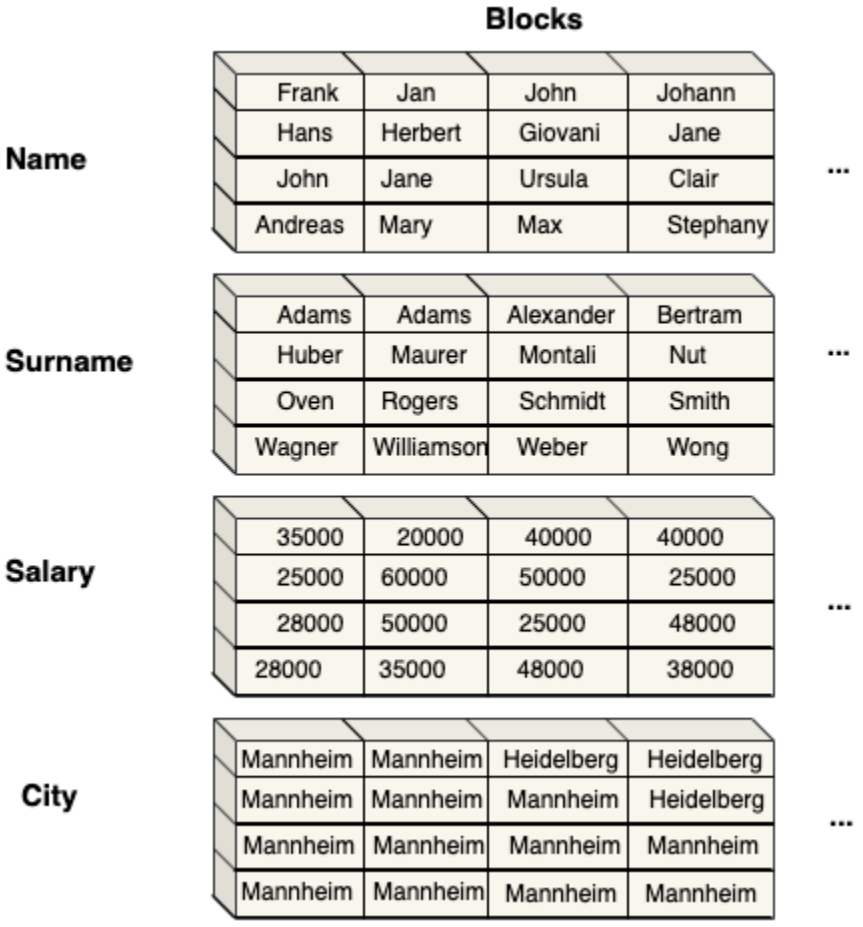
- type of organization we presented so far

Column-based or column-oriented databases organize data **by column** rather than by row

- values of a single attribute (column) for all tuples stored continuously in the block
- values of a column serialized together and stored one after another in blocks
- blocks belonging to columns of a single table are stored in independent files
- advanced compression possibilities (values in blocks are of the same type)
- reduces the amount of wasted space
- insert operations are challenging
- nowadays majority of DBMS vendors have column-oriented implementations: Oracle Database, SAP HANA, Microsoft SQL Server 2012, Vertica, etc. (open source tools: MonetDB, MariaDB ColumnStore, C-Store, etc.)

Column-oriented database example

Employee table with one file for each column



Review questions

- What represents the notion of block (page)?
- Explain unspanned file organization.
- Explain what is blocking factor and how it can be calculated.
- What is the common page size in PostgreSQL databases?
- Calculate the average number of block accesses needed to search for an arbitrary record in the file, using linear search.
- What are the advantages of the heap file organization?
- Explain terms file and record.
- What is OLTP?