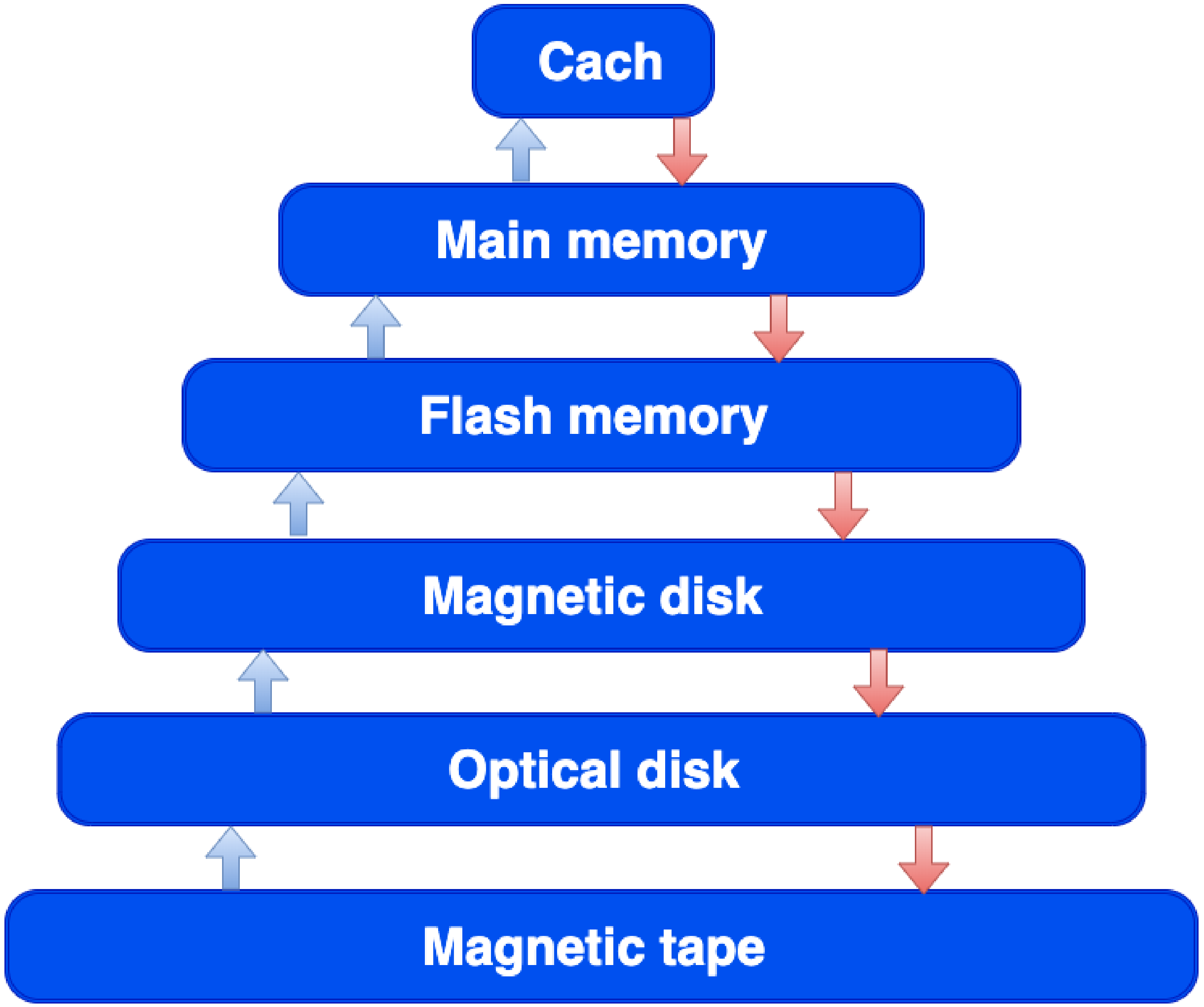
STORAGE STRUCTURE

Storage hierarchy



Overview of Physical Storage media

– In general holds: the faster, the smaller and the 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

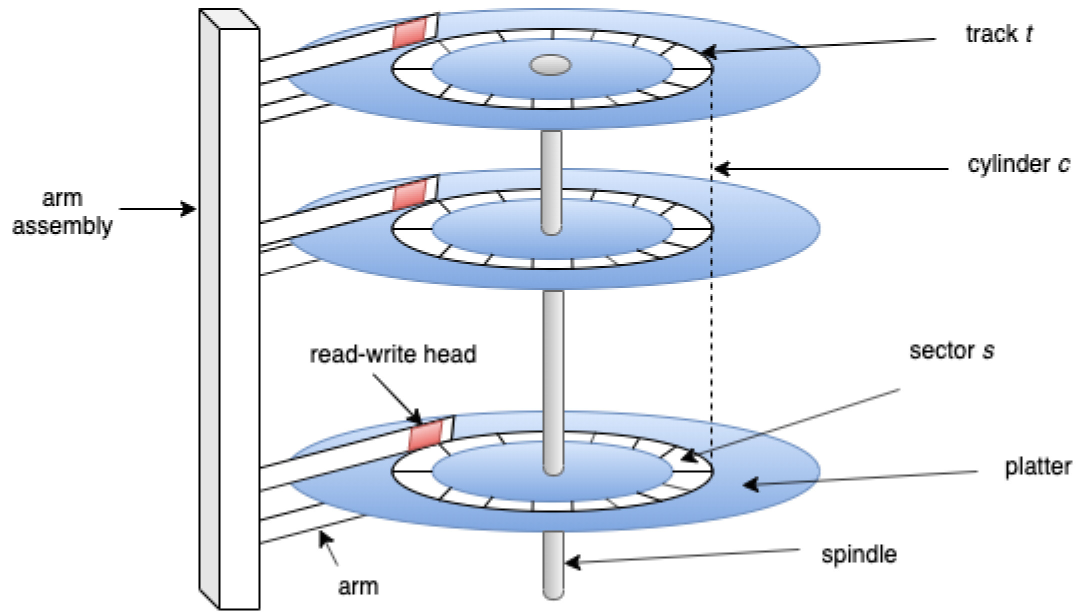
- **primary storage** - can be operated directly by CPUs - **direct access**
  - provides quick access to data but has limited storage capacity (main memory, cache memories)
- **secondary and tertiary storage** - data cannot be processed by CPUs, but has to be transferred to primary storage - **block-addressable**
  - 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 small to be stored in 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 that the right sector appears 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 then seek and latency time (~0.5 ms)

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

- Transferring non consecutive blocks - 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

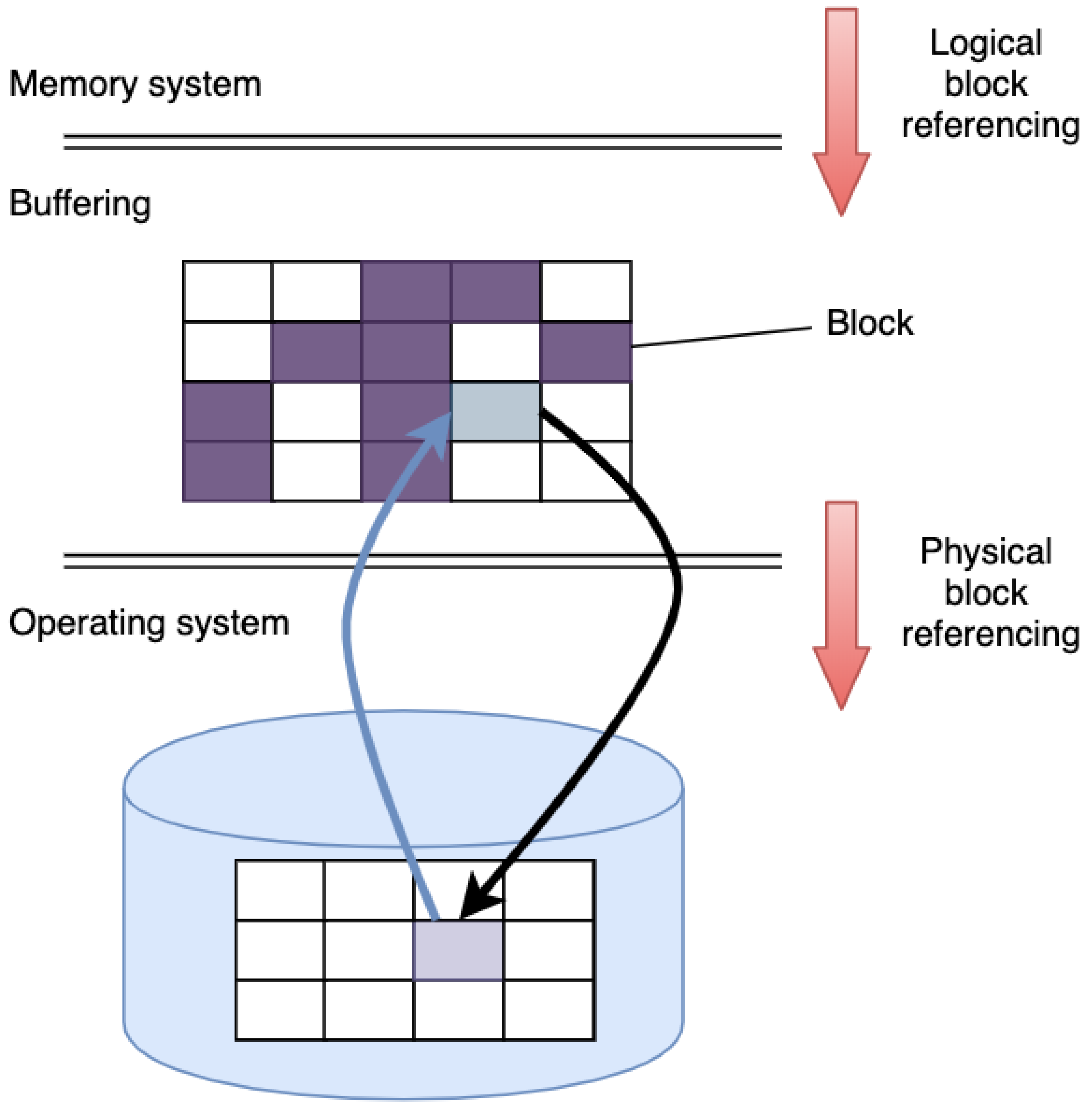
Blocks are transferred from platters to the buffer

- 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
- 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 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 relational model
- records(tuples) are sequences of bytes
  - DBMS has to have possibility to read attributes and values from those tuples

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

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

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.)
  - number of bytes for a datatype is fixed for a computer system (for instance: int - 4 bytes, long - 8 bytes, boolean - 1 byte)
  - *BLOB* (bynary large objects) -used for storing data items that consists of large unstructured objects - (pictures, music)
    - in PostgreSQL type **lo** - large object
  - *CLOB* (character large object) - *TEXT* type in PostgreSQL
  - typically stored separately in a pool on disk and only a **pointer** is included in the record

File records and numeric datatypes

Numerical datatypes can be represented with variable precision or with fixed precision

Variable precision number store data as specified by IEEE-754 standard

- operations typically faster than fixed precision numbers
- problems with rounding errors
- examples FLOAT, REAL/DOUBLE

Fixed precision numbers

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

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 attribute of a relation corresponds to 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

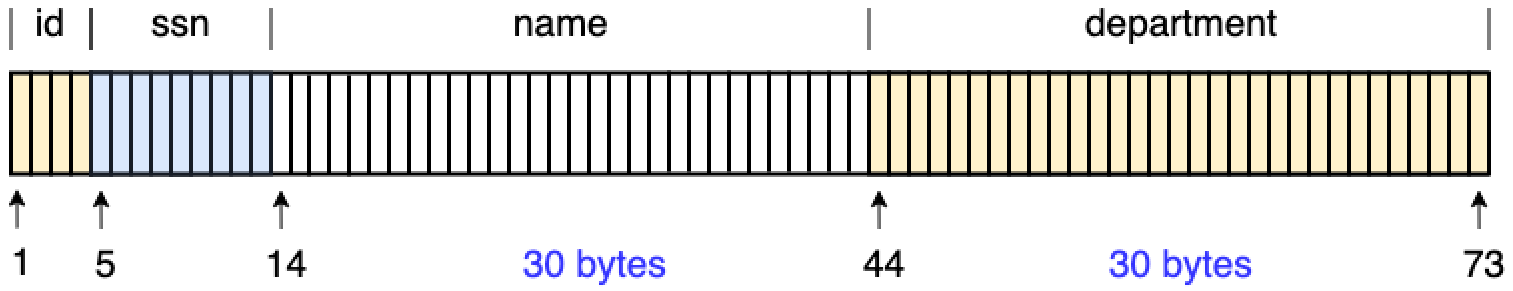
Fixed Length Record

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

Variable Length Record

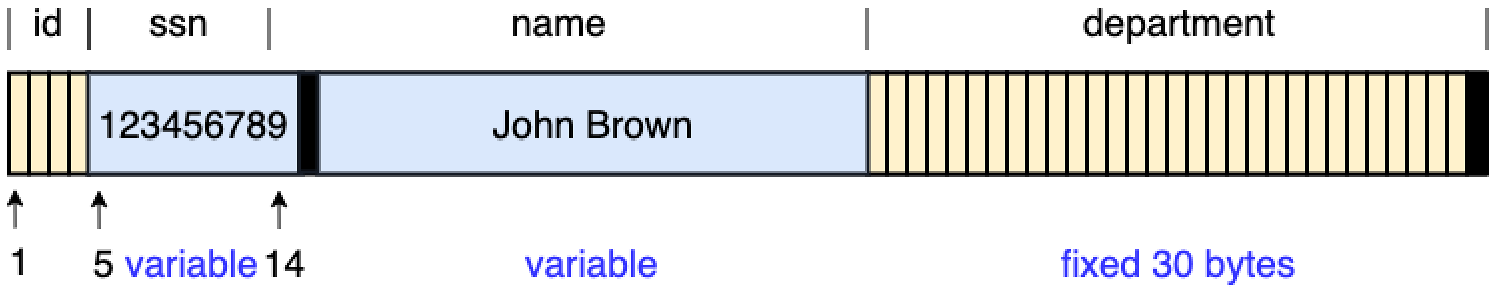
```
   struct student{
       int id,
       char ssn[9];
       char name[30] ;
       char* department; /*Pointer*
}
```

Fixed-length record example



- fixed-length record with 4 fields and the 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**

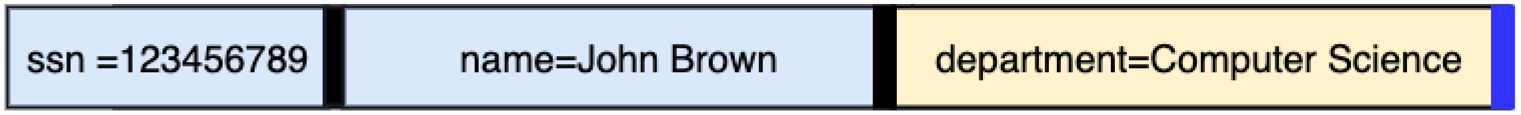
– separator character is a character that doesn’t appear in any field

– access time is longer

– space is not wasted

– length of the field in bytes can be sometimes stored before 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>**

– large number of optional fields

– typical record can have just a small number of fields

In the example we use three types of separator character

– only two types of separator characters are enough

File organization, Blocks (Pages)

File organization refers to the organization of the data of a file into records and blocks

– good file organization has a goal to locate the block containing the desired record with a minimal number of block transfers

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

– blocks have a fixed size and has a unique identifier

– blocks can contain tuples, indexes, metadata, etc.

– disk block (page) can have size 1 - 16 KB (different from the hardware block 4KB)

  – Oracle 4KB, MySQl 16KB

  – PostgreSQL uses a storage abstraction called page instead of block (size 8KB)

File header

A **file header** contains information on files which are needed by system programs which access the file records. It contains information such as:

- information to determine disk addresses of the file blocks
- record format descriptions
  - for fixed-length unspanned - field lengths and order of fields
  - for variable-length records - field type codes, separator characters, records-type codes
- if the address of a the desired record is not known then the *linear search* through all blocks is performed

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

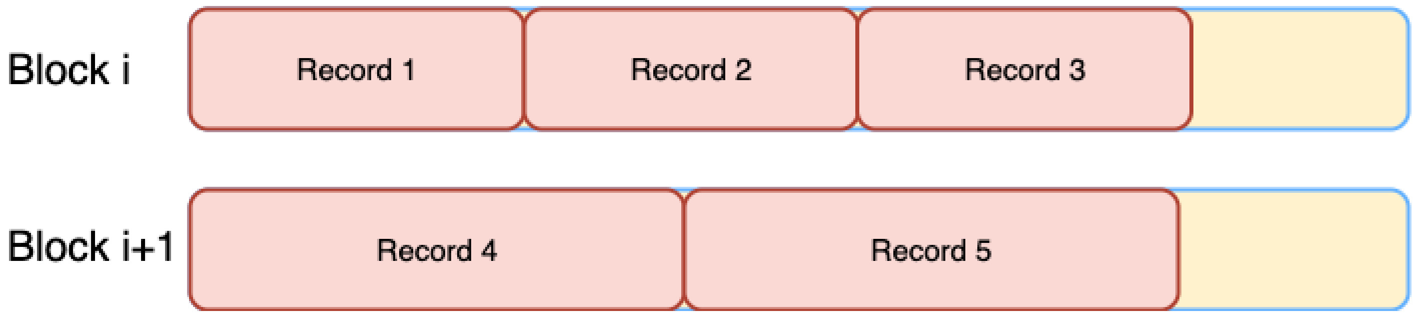
- records do not span over multiple blocks - unused space is left untouched
- suitable for fixed length records

Spanned organization

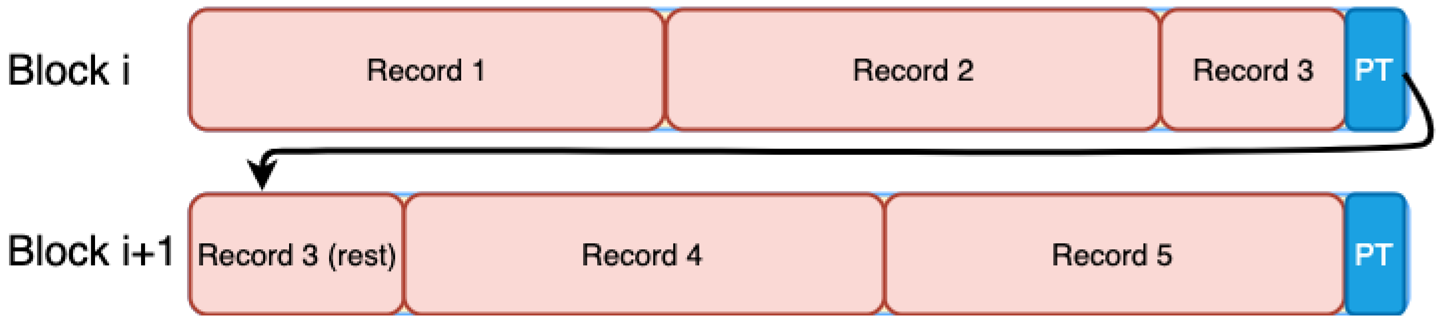
- Unused space of the block(page) is used to store part of the record and pointer which points to the block containing remainder

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 file of *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 after we fill a block 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

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

Linked allocation

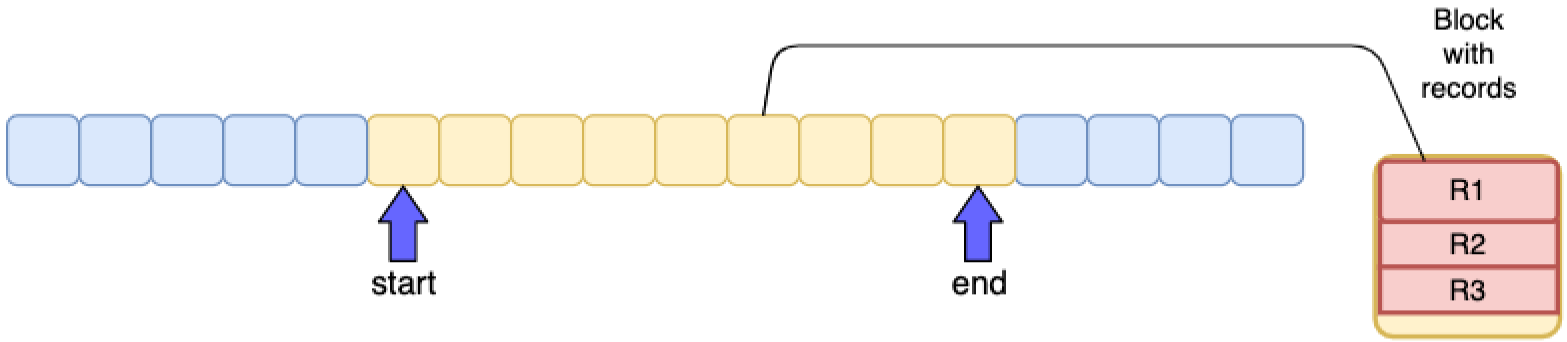
- each block contain a pointer to the next file blocks
- good for adding new records (file expansion)
- reading is slower

Clustered allocation

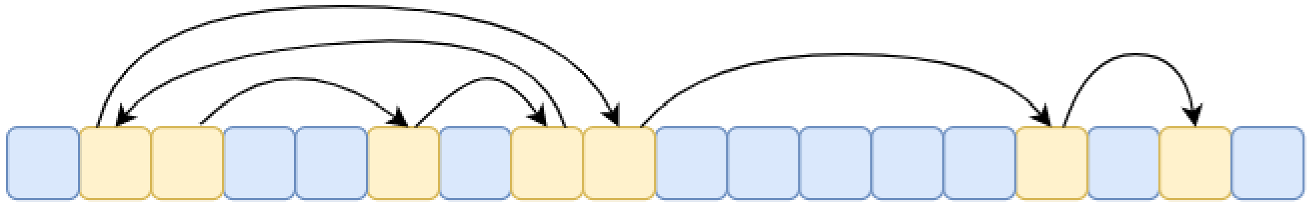
- represents a combination of the two previous allocation types
- clusters contain consecutive disk blocks
- clusters are related using links
- many different combinations with cluster organization

Allocating File Blocks on Disks

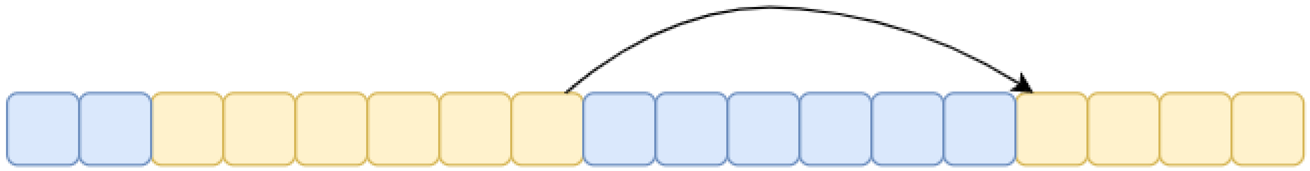
Contiguous allocation



Linked allocation



Allocation in clusters



Operations on files

Operations on files are grouped in:

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

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

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

Access operations

Typical operations used by DBMS for locating and accessing records on 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 main memories 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 groups 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 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 is an expensive operations

- if there is no indexes involves the *linear search* is required
- for a files with *b* blocks it requires searching $\frac{b}{2}$ blocks, on average
- complexity is therefore linear $O(b)$ to the number of blocks b of the heap file

Deleting a record includes the operation of searching and after the block is in the buffer

- deleting the record from the block in the buffer
- rewriting the block back to the disk
- if the block in now empty the operation requires updating the address of the last block in the file header
- complexity $O(b)$ because we exploited 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 results in wasted storage space

Alternatives to tackle this problem:

1. introducing an extra byte or bit, called **deletion marker** which is stored within each record
   - search programs consider only valid records
   - deleted records can be restored if needed
2. inserting new records in place of those which are deleted
   - can requires maintaining the list of these gaps in the blocks and the size of those gaps

reorganization of blocks is necessary for those alternatives

- 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 position $(i \ \ mod \ \ bfr)$ in it's block
- however this i-th record is the record in the unsorted file and **do not speed up** search for i-th record in any order

Sorting of heap files requires external sorting which is a 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**

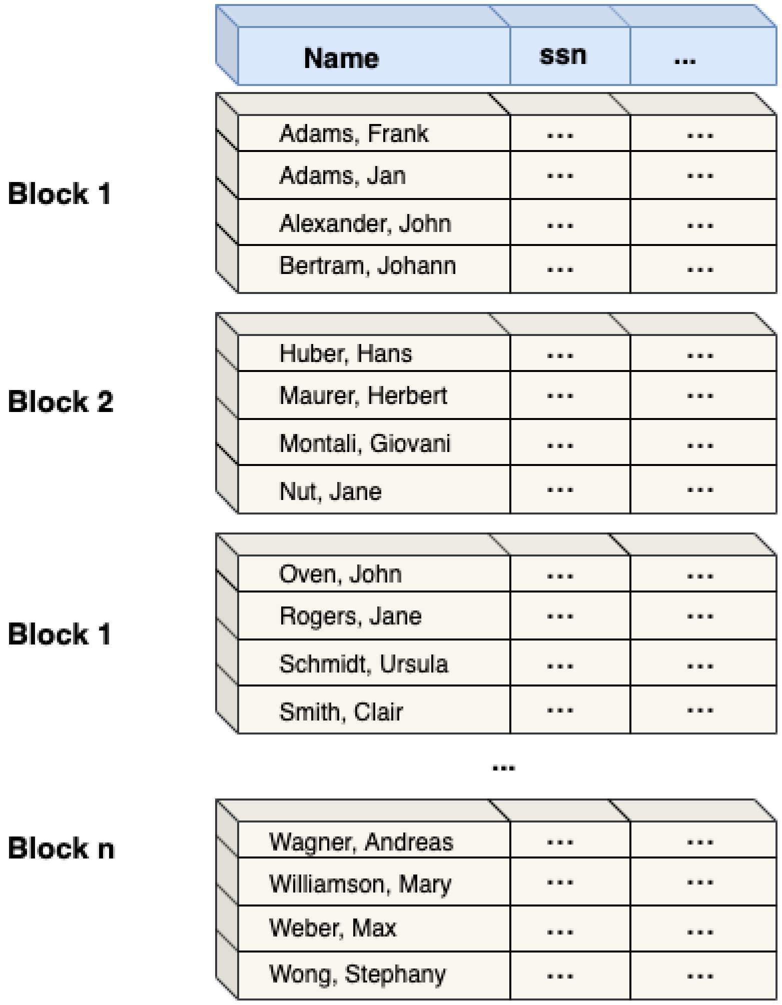
- ordering field is called **ordering key** if it is a *key field*

Advantages of sorted files

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

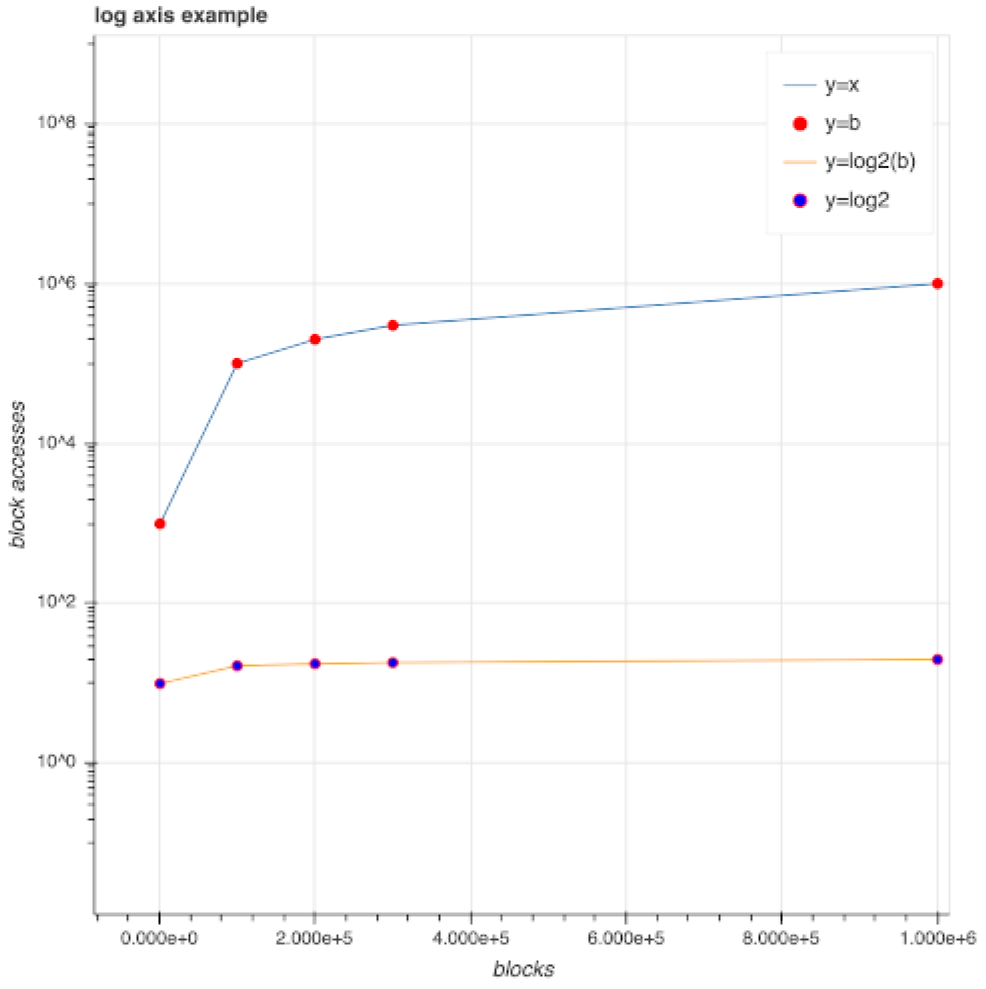
Sorted files

File sorted according to the primary key



	Name	ssn	...
Block 1	Adams, Frank	...	...
	Adams, Jan	...	...
	Alexander, John	...	...
	Bertram, Johann	...	...
Block 2	Huber, Hans	...	...
	Maurer, Herbert	...	...
	Montali, Giovani	...	...
	Nut, Jane	...	...
Block 1	Oven, John	...	...
	Rogers, Jane	...	...
	Schmidt, Ursula	...	...
	Smith, Clair	...	...
	...		
Block n	Wagner, Andreas	...	...
	Williamson, Mary	...	...
	Weber, Max	...	...
	Wong, Stephany	...	...

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 file
2. keeping **overflow** or **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 $O(b)$ (on averaga b/2) in the worst case b blocks	binary search O(log b)
Insert operation	O(1) - effective	O(b) blocks has to be read (on avereage b/2)
Delete operation	$O(b)$- 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

Example - PostgreSQL file organization

Each table is stored in a separate file

- 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
```

- all files of a database can be found in the subdirectory $PGDATA/base/ < oid >$
- if a table exceeds size of 1 GB then it's divided in different files of the size of at most 1GB

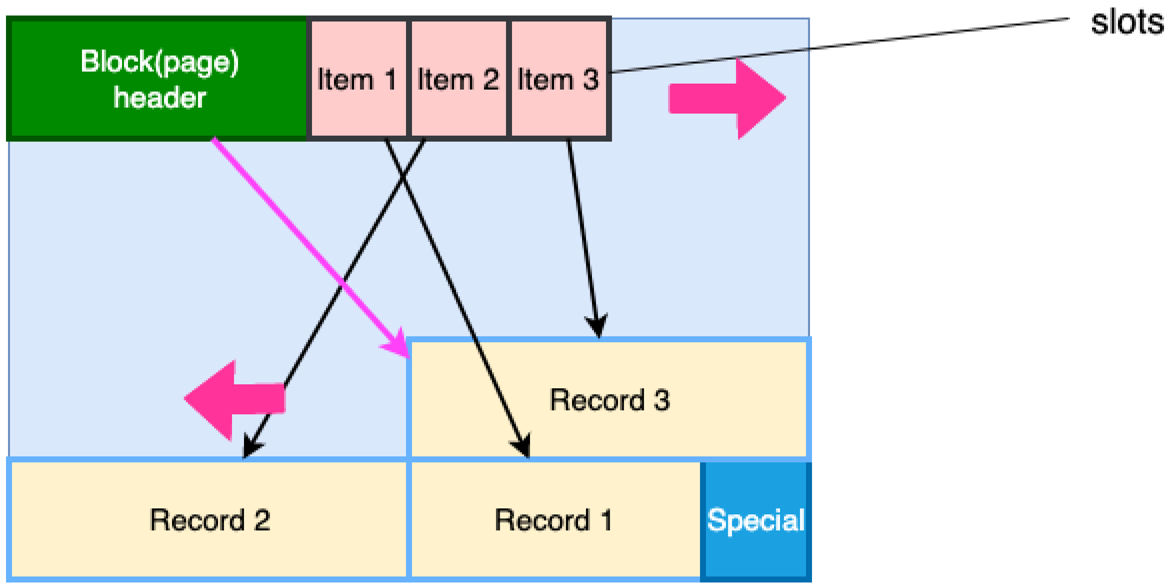
PostgreSQL file organization - page

Each table is an array of blocks (pages) of a fixed size (usually 8KB)

– all pages are logically equivalent

– heap structure is used to store files

– page structure is as the following image (slotted pages)



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

PostgreSQL file organization (TOAST)

PostreSQL uses **The Oversized-Attribute Storage Technique (TOAST)** technology to

– manages cases when columns have the size greater than 8 KB

– each table has TOAST table which is used for big columns

– big columns are split up into 2 KB chunks and stored in TOAST tables

– represents an alternative to the spanned organization of blocks

– columns have mostly a limit of 1 GB

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.