


# Physical Data Organization

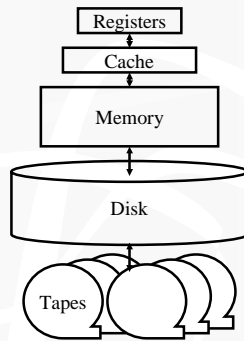
CPS 216  
Advanced Database Systems

## Outline

- It's all about disks
  - That's why you always draw a database as 
- Record layout
- Block layout

2

## Storage hierarchy



3

## How far away is data?

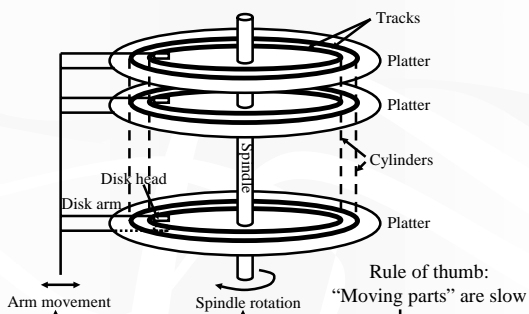
Location	Cycles	Location	Time
Registers	1	My head	1 min.
On-chip cache	2	This room	2 min.
On-board cache	10	Duke campus	10 min.
Memory	100	Washington D.C.	1.5 hr.
Disk	$10^6$	Pluto	2 yr.
Tape	$10^9$	Andromeda	2000 yr.

(Source: AlphaSort paper, 1995)

➤ I/O dominates—design your algorithms to reduce I/O!

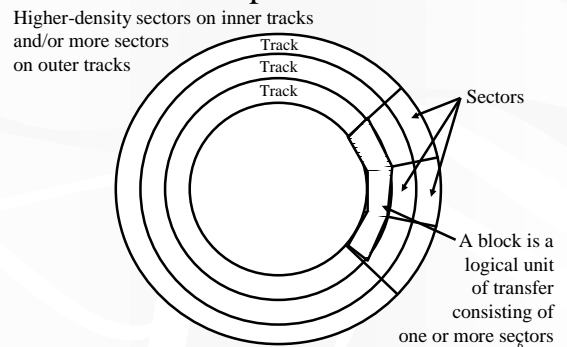
4

## A typical disk



5

## Top view



## Disk access time

Sum of:

- Seek time: time for disk heads to move to the correct cylinder
- Rotational delay: time for the desired block to rotate under the disk head
- Transfer time: time to read/write data in the block (= time for disk to rotate over the block)

7

## Random disk access

Seek time + rotational delay + transfer time

- Average seek time
  - Time to skip one half of the cylinders?
  - Not quite; should be time to skip a third of them (why?)
  - “Typical” value: 5 ms
- Average rotational delay
  - Time for a half rotation (a function of RPM)
  - “Typical” value: 4.2 ms (7200 RPM)

8

## Sequential disk access

Seek time + rotational delay + transfer time

- Seek time
  - 0 (assuming data is on the same track)
- Rotational delay
  - 0 (assuming data is in the next block on the track)
- Easily an order of magnitude faster than random disk access!

9

## Data layout strategy

Keep related things close together!

- Same sector/block
- Same track
- Same cylinder
- Adjacent cylinder

10

## More performance tricks

- Disk scheduling algorithm
  - Example: “elevator” algorithm
- Track buffer
  - Read/write one entire track at a time
- Double buffering
  - While processing the current block in memory, prefetch the next block from disk
- Parallel I/O
  - More disk heads working at the same time

11

## Record layout

Record = row in a table

- Variable-format records
  - Rare in DBMS—table schema dictates the format
  - Maybe relevant for semi-structured data such as XML
- Focus on fixed-format records
  - With fixed-length fields only, or
  - With possible variable-length fields

12

### Fixed-length fields

- All field lengths and offsets are constant
  - Computed from schema, stored in the system catalog
- Example: create table Student(SID integer, name CHAR(20), age integer, GPA float)
 

0	4	24	28	36
142	Bart (padded with "\0")		10	2.3
- Watch out for alignment!
  - May need to pad; reorder columns if that helps
- What about NULL?
  - Add a bitmap

13

### Variable-length fields

- Example: create table Student (SID integer, name VARCHAR(20), age integer, GPA float, comment VARCHAR(100))
  - Approach 1: use field delimiters
 

0	4	8	16	22	32
142	10	2.3	Bart\0	Weird kid!\0	
  - Approach 2: use an offset array
 

0	4	8	16	18	22	32
142	10	2.3	Bart	Weird kid!		
  - Update is messy if it changes the length of a field

14

### LOB fields

- Example: create table Student(SID integer, name CHAR(20), age integer, GPA float, picture BLOB(32000))
  - Student records get “de-clustered”
    - Bad because most queries do not involve picture
  - Decompose (automatically done by DBMS)
    - Student(SID, name, age, GPA)
    - StudentPicture(SID, picture)

15

### Block layout

How do you organize records in a block?

- NSM (N-ary Storage Model)
  - Most commercial DBMS
- DSM (Decomposition Storage Model)
- PAX (Partition Attributes Across)
  - Recent work (Ailamaki et al., VLDB 2001)

16

### NSM

- Store records from the beginning of each block
- Use a directory at the end of each block
  - To locate records and manage free space
  - Necessary for variable-length records

142	Bart	10	2.3	123	Milhouse	10	3.1
357	Lisa	8	4.3				
		456	Ralph	8	2.3		

17

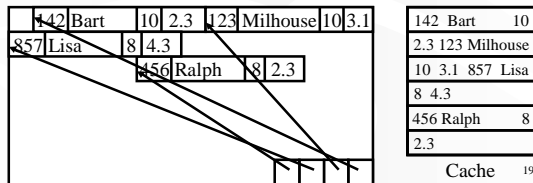
### Options

- Reorganize after every update/delete to avoid fragmentation
  - Need to rewrite half of the block on average
- What if records are fixed-length?
  - Reorganize after delete
    - Only need to move one record
    - Need a pointer to the beginning of free space
  - Do not reorganize after update
    - Need a bitmap indicating which slots are in use

18

## Cache behavior of NSM

- Query: SELECT SID FROM Student WHERE GPA > 2.0;
- Assumption: cache block size < record size
- Lots of cache misses!
  - Things are not close enough (by memory standard)



Cache 19

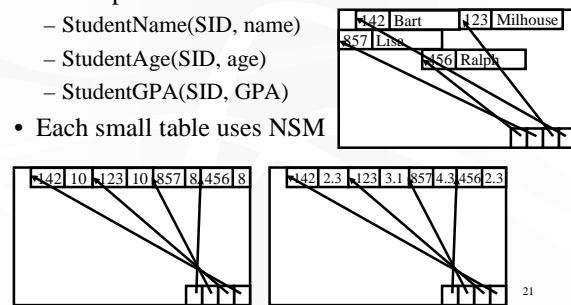
## Do cache misses matter in DBMS?

- Yes? Percentage of memory-related stall time due to data cache misses:
  - 90% for OLAP workloads  
(lots of large, complex queries; few updates)
  - 50-70% for OLTP workloads  
(lots of small queries and updates)
- No? Compared to disk I/Os, memory-related stall time is nothing

20

## DSM

- Decompose table into smaller ones
  - StudentName(SID, name)
  - StudentAge(SID, age)
  - StudentGPA(SID, GPA)
- Each small table uses NSM



21

## Pros and cons of DSM

### Pros

- Smaller records = more will fit in a cache block
- Values in the same column are closer = better locality for queries that just access the key and one column

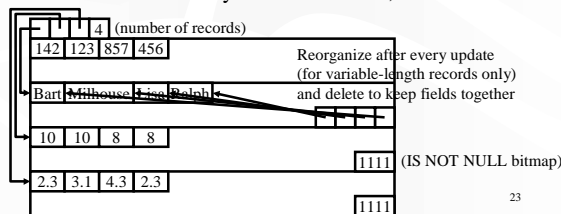
### Cons

- Need to keep the key (or a row ID) in every decomposed table = higher storage overhead
- Different parts of the row are farther apart = bad locality for queries accessing many columns—need joins!

22

## PAX

- Keep entire rows in a block
  - Intra-record locality in a disk block, like NSM
- Within a block, cluster columns
  - Inter-record locality in a cache block, like DSM



23

## PAX versus NSM

- Space requirement
  - Roughly the same
- Cache performance
  - PAX incurs 75% less data cache misses than NSM
- Overall performance
  - For OLAP, PAX is 11-48% faster
  - For OLTP
    - Updates: PAX is 10%-16% faster (assuming NSM reorganizes as well)
    - Queries (typically very selective): I/O still dominates? 24

Next time

Indexing

25