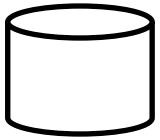
Database Systems I

CMPT 354 Summer 2024
Zhengjie Miao

July 3 2024

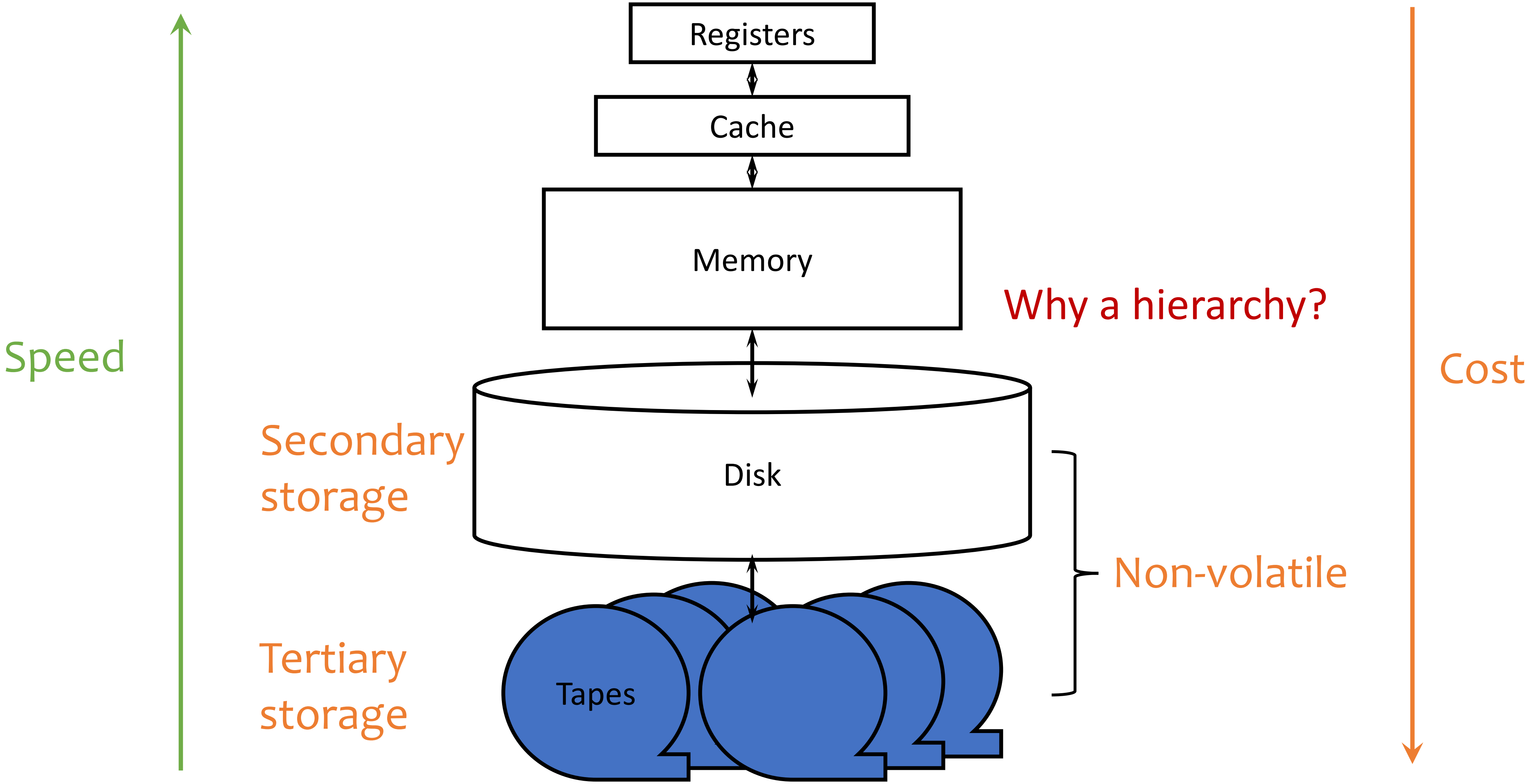
Announcements (Wed. July 3)

Outline

- It’s all about disks!
  - That’s why we always draw databases as 
  - And why the single most important metric in database processing is (oftentimes) the number of disk I/O’s performed

- Storing data on a disk
  - Record layout
  - Block layout
  - Column stores

Storage hierarchy



How far away is data?

Location	Cycles
Registers	1
On-chip cache	2
On-board cache	10
Memory	100
Disk	10^6
Tape	10^9

Location	Time
My head	1 min.
This classroom	2 min.
AQ building	10 min.
Bellingham, WA	1.5 hr.
Pluto	2 yr.
Andromeda	2000 yr.

(Source: AlphaSort paper, 1995)
The gap has been widening!

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

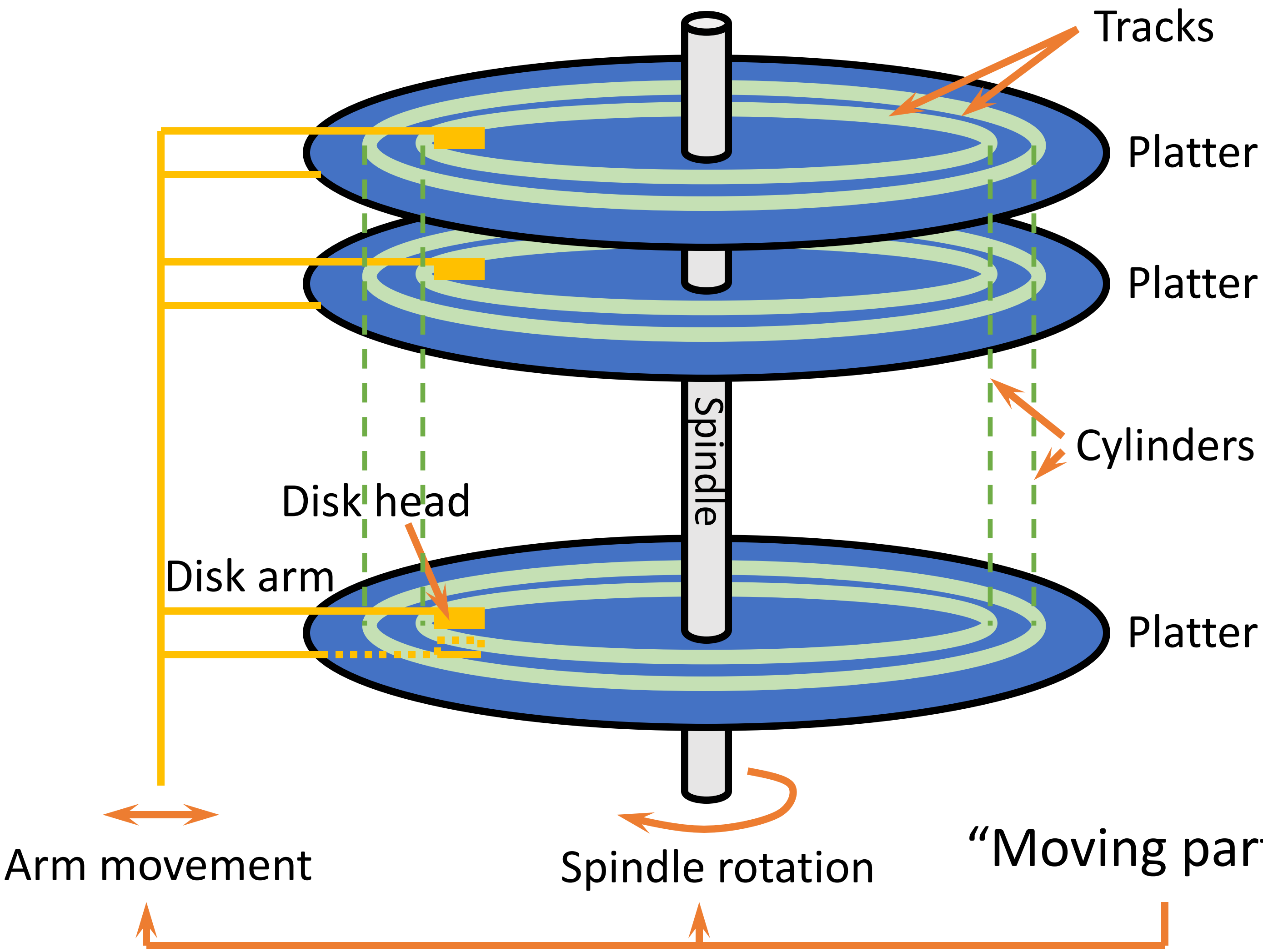
Latency Numbers Every Programmer Should Know

```
Latency Comparison Numbers
--------------------------
L1 cache reference                           0.5 ns
Branch mispredict                            5   ns
L2 cache reference                           7   ns                      14x L1 cache
Mutex lock/unlock                           25   ns
Main memory reference                      100   ns                      20x L2 cache, 200x L1 cache
Compress 1K bytes with Zippy             3,000   ns        3 us
Send 1K bytes over 1 Gbps network       10,000   ns       10 us
Read 4K randomly from SSD*             150,000   ns      150 us          ~1GB/sec SSD
Read 1 MB sequentially from memory     250,000   ns      250 us
Round trip within same datacenter      500,000   ns      500 us
Read 1 MB sequentially from SSD*     1,000,000   ns    1,000 us    1 ms  ~1GB/sec SSD, 4X memory
Disk seek                           10,000,000   ns   10,000 us   10 ms  20x datacenter roundtrip
Read 1 MB sequentially from disk    20,000,000   ns   20,000 us   20 ms  80x memory, 20X SSD
Send packet CA->Netherlands->CA    150,000,000   ns  150,000 us  150 ms

Notes
-----
1 ns = 10^-9 seconds
1 us = 10^-6 seconds = 1,000 ns
1 ms = 10^-3 seconds = 1,000 us = 1,000,000 ns

Credit
------
By Jeff Dean:               http://research.google.com/people/jeff/
Originally by Peter Norvig: http://norvig.com/21-days.html#answers
```

A typical hard drive

http://upload.wikimedia.org/wikipedia/commons/f/f8/Laptop-hard-drive-exposed.jpg

A typical hard drive

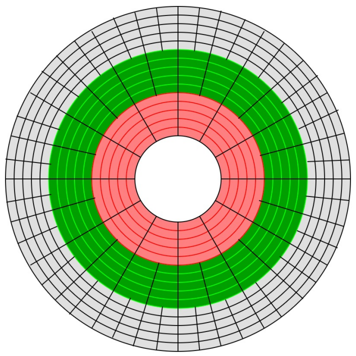


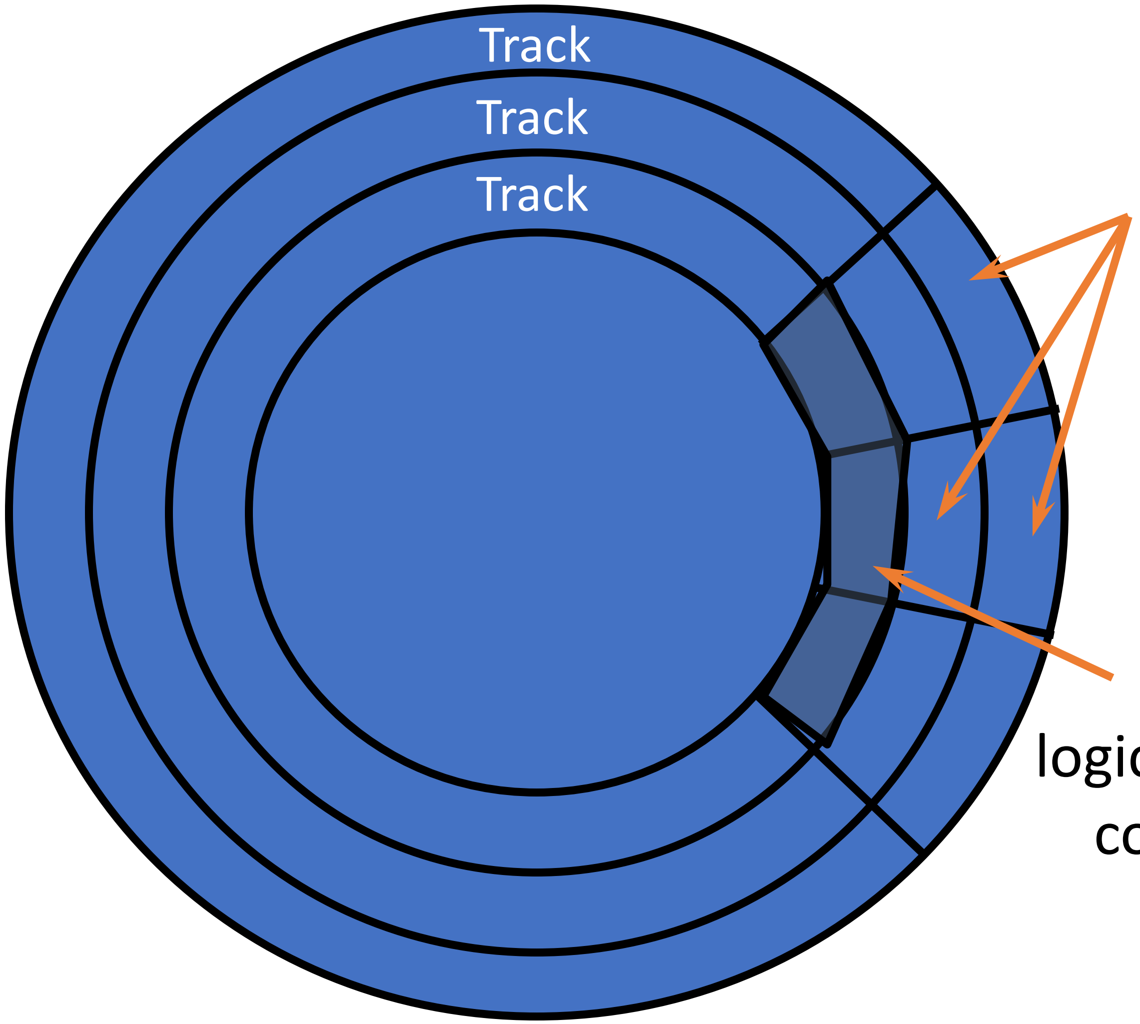
“Moving parts” are slow

Top view

Three to four sectors per block

“Zoning”: more sectors/data on outer tracks





A block is a logical unit of transfer consisting of one or more sectors

Disk access time

- Disk access time: time from when a read or write request is issued to when data transfer begins
- 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)

Random disk access

- Seek time + rotational delay + transfer time
- Average seek time

Random access of the blocks are distributed among all the cylinders

  - 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)

Sequential disk access

- Successive requests are for successive block numbers, which are on the same track, or on adjacent tracks
- 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!

What about SSD (solid-state drives)?

SSDs support random access to any block?
Hard drives are dominating storage medium

Data storage is tending towards SSDs, but there is a larger usage of hard drives since they are cheaper

http://www.techgoondu.com/wp-content/uploads/2012/12/SSD-6-25-121.jpg

What about SSD (solid-state drives)?

- No mechanical parts
- Mostly flash-based nowadays
- 1-2 orders of magnitude faster random access than hard drives (under 0.1ms vs. several ms)
  - But still much slower than memory (~0.1μs)
- Little difference between random vs. sequential read performance
- Random writes still hurt
  - In-place update would require erasing the whole "erasure block" and rewriting it!

For hard drive, you can replace a portion of the data. Random writes are not as impactful for random writes

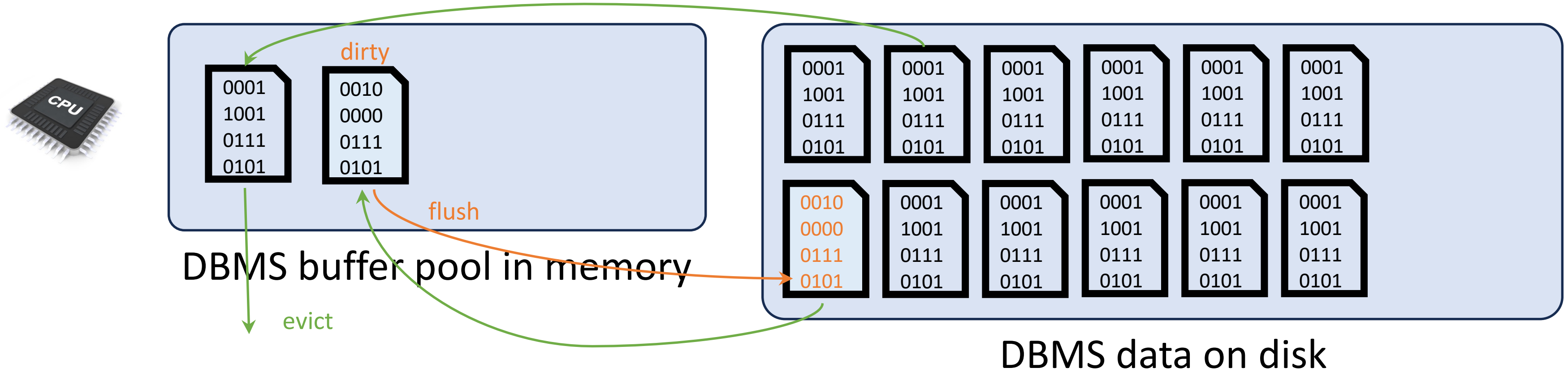
Important consequences

- It's all about reducing I/O's!
- Cache blocks from stable storage in memory
    - DBMS maintains a memory buffer pool of blocks
    - Reads/writes operate on these memory blocks
    - Dirty (updated) memory blocks are "flushed" back to stable storage

Important consequences

- It's all about reducing I/O's!
- Cache blocks from stable storage in memory
  - DBMS maintains a memory buffer pool of blocks
  - Reads/writes operate on memory blocks
  - Dirty (updated) memory blocks are "flushed" back to stable storage

If you want to work on the data, you will first need to bring it to the buffer pool



- Sequential I/O generally cheaper than random I/O

Performance tricks

- Disk layout strategy
  - Keep related things (what are they?) close together: same sector/block → same track → same cylinder → adjacent cylinder
- Prefetching
  - While processing the current block in memory, fetch the next block from disk (overlap I/O with processing)
- Parallel I/O
  - More disk heads working at the same time
- Disk scheduling algorithm
  - Example: “elevator” algorithm
- Track buffer
  - Read/write one entire track at a time

Record layout

Record = row in a table

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

Fixed-length fields

- All field lengths and offsets are constant
  - Computed from schema, stored in the system catalog
- Example:
  - `CREATE TABLE User(uid INT, name CHAR(20), age INT, pop FLOAT);`

0 4 24 28 36

`142`	`Bart` (padded with space)	`10`	`0.9`

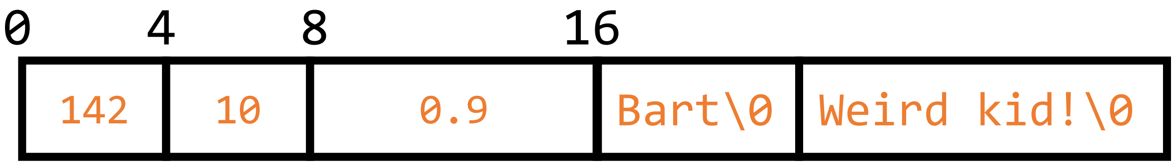
- Watch out for alignment
  - May need to pad; reorder columns if that helps
- What about NULL?
  - Add a bitmap at the beginning of the record

You add a bit indicating if it is null or not

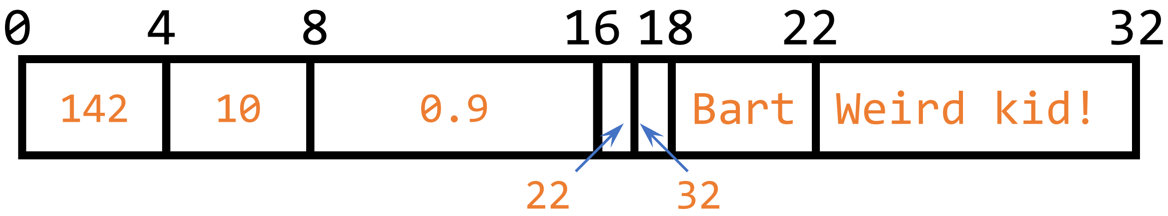
Variable-length records

Ask the professor about resources he would recommend (asides from W3Schools) for practicing XPath queries

- Example: `CREATE TABLE User(uid INT, name VARCHAR(20),`
  `age INT, pop FLOAT, comment VARCHAR(100));`
- Approach 1: use field delimiters (‘\0’ okay?)



- Approach 2: use an offset array



- Put all variable-length fields at the end (why?)

When updating the fields, you will need to update the offset. It will affect the offset for fixed length fields. Ask professor after class why this is

- Update is messy if it changes the length of a field

LOB fields

When accessing the data most of the time, you will not be using the picture. However, every time you access data, you will also be getting the photo

- Example:
```
CREATE TABLE User(uid INT,
         name CHAR(20), age INT,
         pop FLOAT, picture BLOB(32000));
```
- User records get “de-clustered”
  - Bad because most queries do not involve `picture`
- Decomposition (automatically and internally done by DBMS without affecting the user)
  - (*uid*, *name*, *age*, *pop*)
  - (*uid*, *picture*)

Block layout

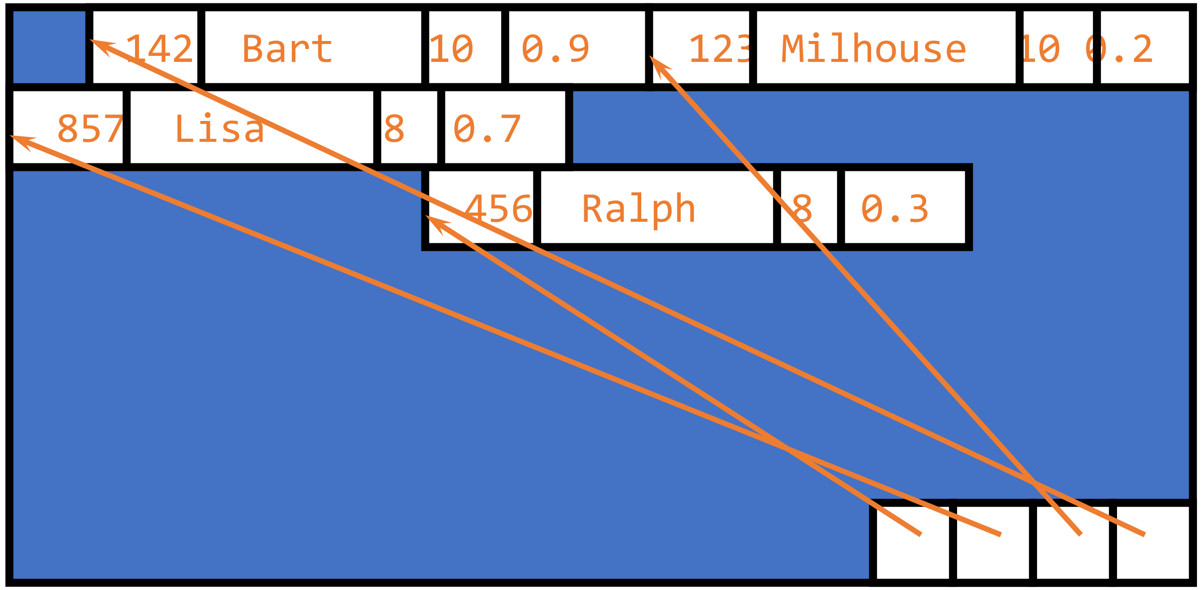
How do you organize records in a block?

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

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

The directory tells you where to find the record



Why store data and directory at two different ends?

Putting the directory at the end allows you to expand the number of records you have

If the directory is at the beginning, rewriting data may become more expensive. Ask professor about this

Need a way to tell where the directory ends too

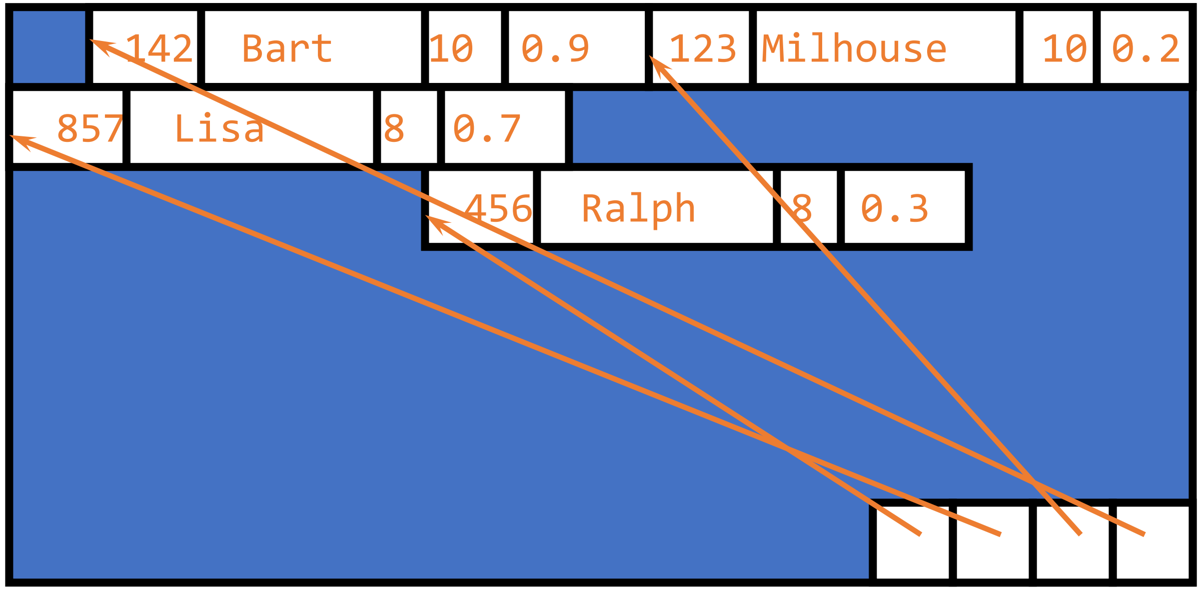
Options

- Reorganize after every update/delete to avoid fragmentation (gaps between records)
  - Need to rewrite half of the block on average
- A special case: What if records are fixed-length?
  - Option 1: reorganize after delete
    - Only need to move one record
    - Need a pointer to the beginning of free space
  - Option 2: do not reorganize after update
    - Need a bitmap indicating which slots are in use

Cache behavior of NSM

- Query: `SELECT uid FROM User WHERE pop > 0.8;`
- Assumptions: no index, and cache line size < record size
- Lots of cache misses
  - uid and pop are not close enough by memory standards

Not all the information in a single row can fit in a single line.
The gap after Lisa's popularity indicates there is a gap between Lisa's and Ralph's record

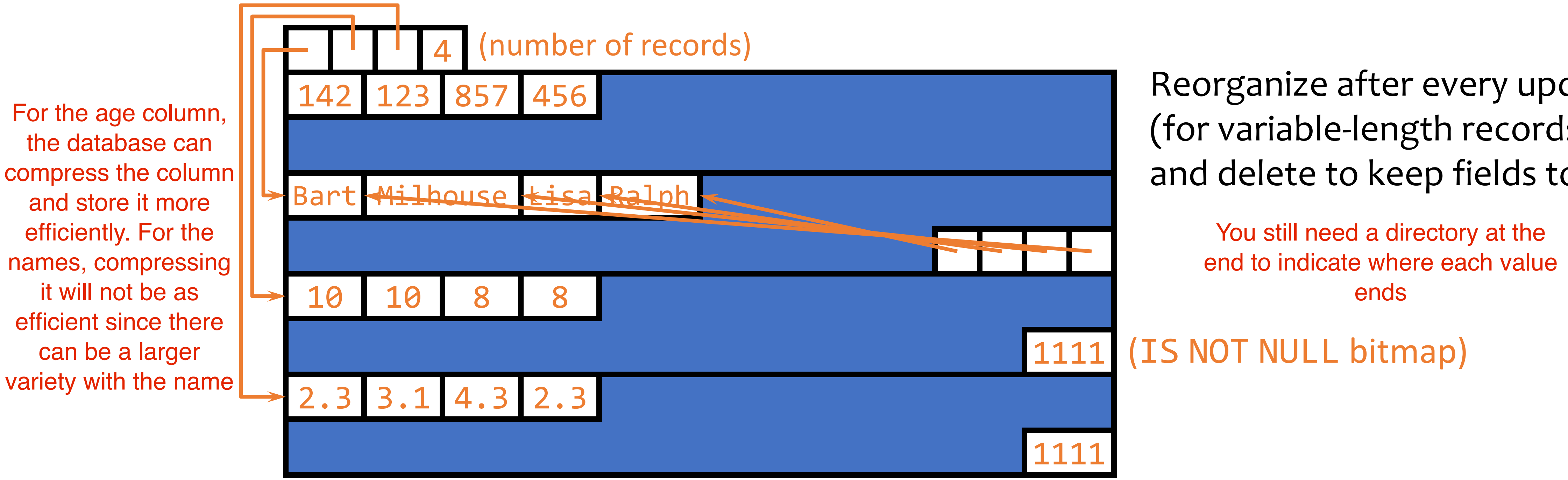


142 Bart 10
0.9 123 Milhouse
10 0.2 857 Lisa
8 0.7
456 Ralph 8
0.3

Cache

PAX

- Most queries only access a few columns
- Cluster values of the same columns in each block
    - When a particular column of a row is brought into the cache, the same column of the next row is brought in together



Reorganize after every update (for variable-length records only) and delete to keep fields together

Beyond block layout: column stores

- The other extreme: store tables by columns instead of rows
- Advantages (and disadvantages) of PAX are magnified
  - Better cache performance
  - Fewer I/O’s for queries involving many rows but few columns
  - Aggressive compression to further reduce I/O’s
- More disruptive changes to the DBMS architecture are required than PAX
  - Not only storage, but also query execution and optimization

Column vs. row stores

uid	*name*	*age*	*pop*
142	Bart	10	0.9
123	Milhouse	10	0.2
857	Lisa	8	0.7
456	Ralph	8	0.3
…	…	…	…

Row oriented

142	Bart	10	0.9
123	Milhouse	10	0.2
857	Lisa	8	0.7
…	…	…	…

Column oriented

142	123	857	…
Bart	Milhouse	Lisa	…
10	10	8	…
0.9	0.2	0.7	…

Example: Apache Parquet

- A table is horizontally partitioned into row groups (~512MB-1GB/row group); each group is stored consecutively
  - On a “block” of HDFS (Hadoop Distributed File System)
- A row group is vertically divided into column chunks, one per column
- Each column chunk is stored in pages (~8KB/page); each page can be compressed/encoded independently

- Not designed for in-place updates though!

Summary

- Storage hierarchy
  - Why I/O's dominate the cost of database operations
- Disk
  - Steps in completing a disk access
  - Sequential versus random accesses
- Record layout
  - Handling variable-length fields
  - Handling `NULL`
  - Handling modifications
- Block layout
  - NSM: the traditional layout
  - PAX: a layout that tries to improve cache performance
- Column stores: NSM transposed, beyond blocks