Internals of a DBS I

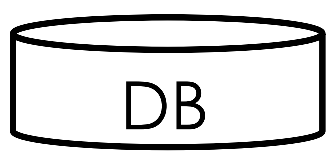
Query Optimization
And Execution

Relational Operators

Files and
Access Methods

Buffer Management

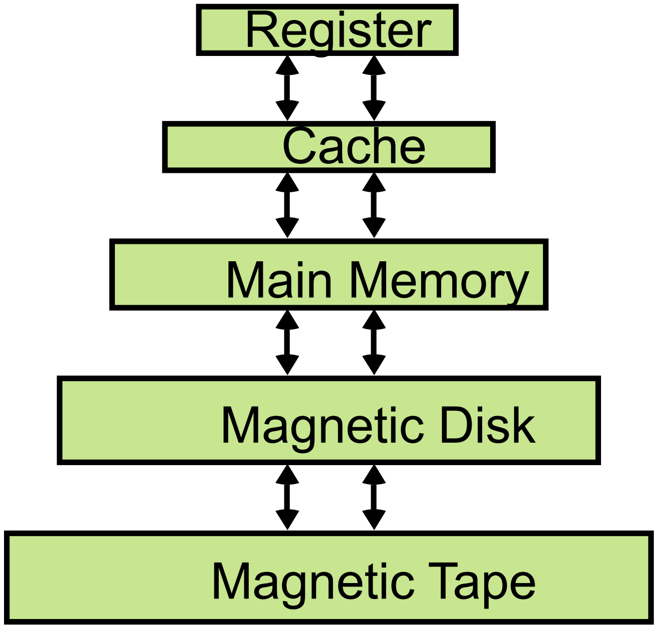
Disk Space
Management

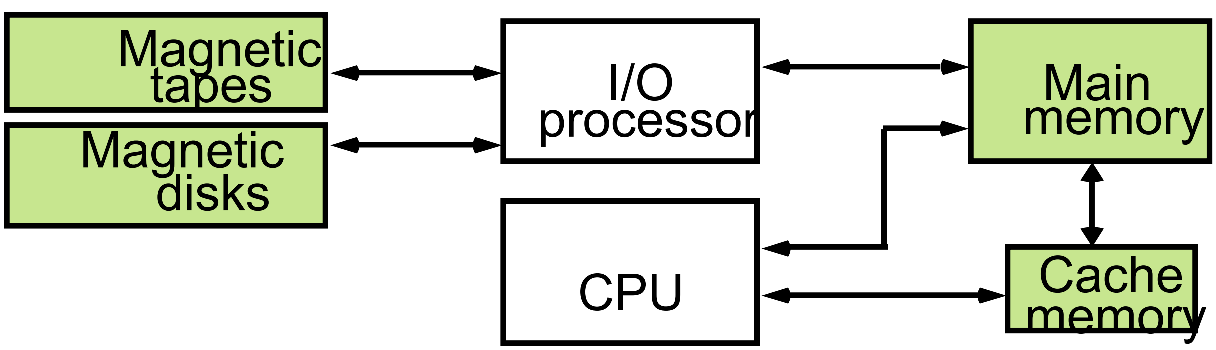


The very Essentials of Disk and Buffer Management

Memory Hierarchy

Memory Hierarchy is to obtain the highest possible
access speed while minimizing the total cost of the memory system





Tape drives

https://en.wikipedia.org/wiki/9_track_tape

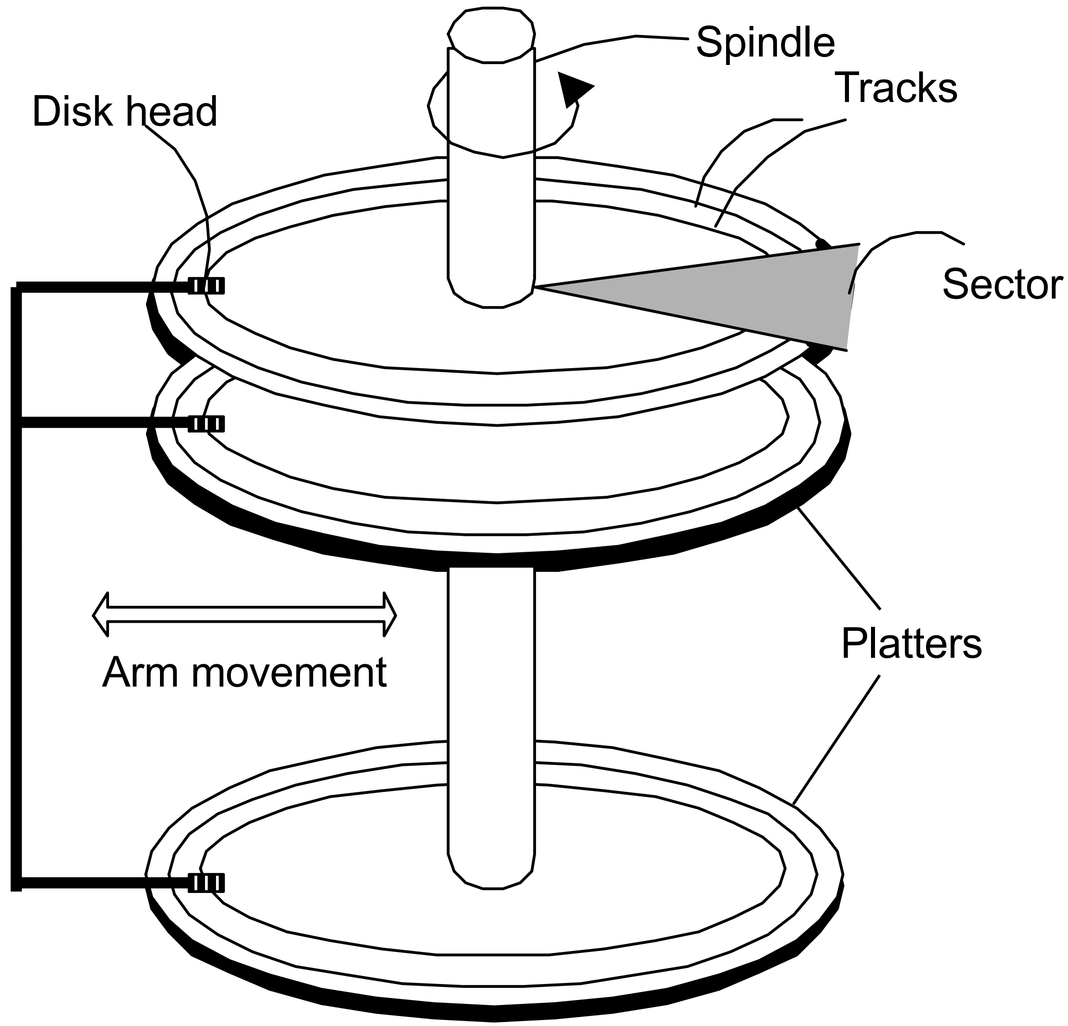
https://en.wikipedia.org/wiki/Tape_library

Disks and Main Memory

- DBMS stores information persistently on (“hard”) disks.
- Unit of transfer main-memory/disk: *disk blocks* or *pages*.
-

-
-

Why Block/Page Concept



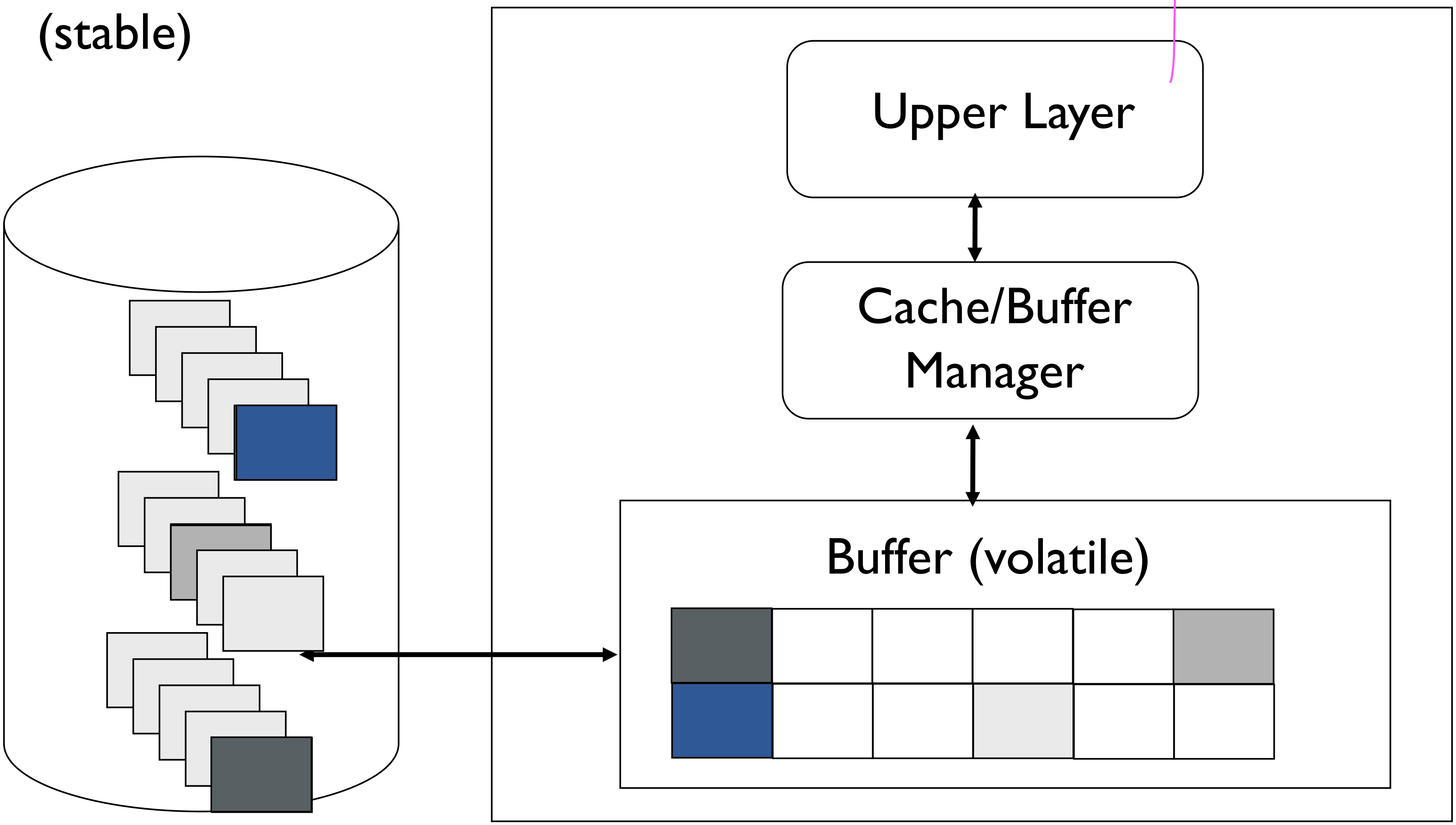
Disks and Main Memory

- DBMS stores information persistently on ("hard") disks.
- Unit of transfer main-memory/disk: *disk blocks* or *pages*.
- Time to read/write data block:
  - 2- 10 msec for random data block (main factor is seek time)
  - If blocks are sequentially on disk, each additional block only 1 msec
  - Compare main memory access: in nanoseconds
  - SSD: 0.1 ms (still slower than main memory access and calculations…)
- Basic operations
  - READ: transfer data from disk to main memory (RAM).
  - WRITE: transfer data from RAM to disk.
- Why disks?
  - Cheaper than Main Memory
  - Higher Capacity
  - Main Memory is volatile

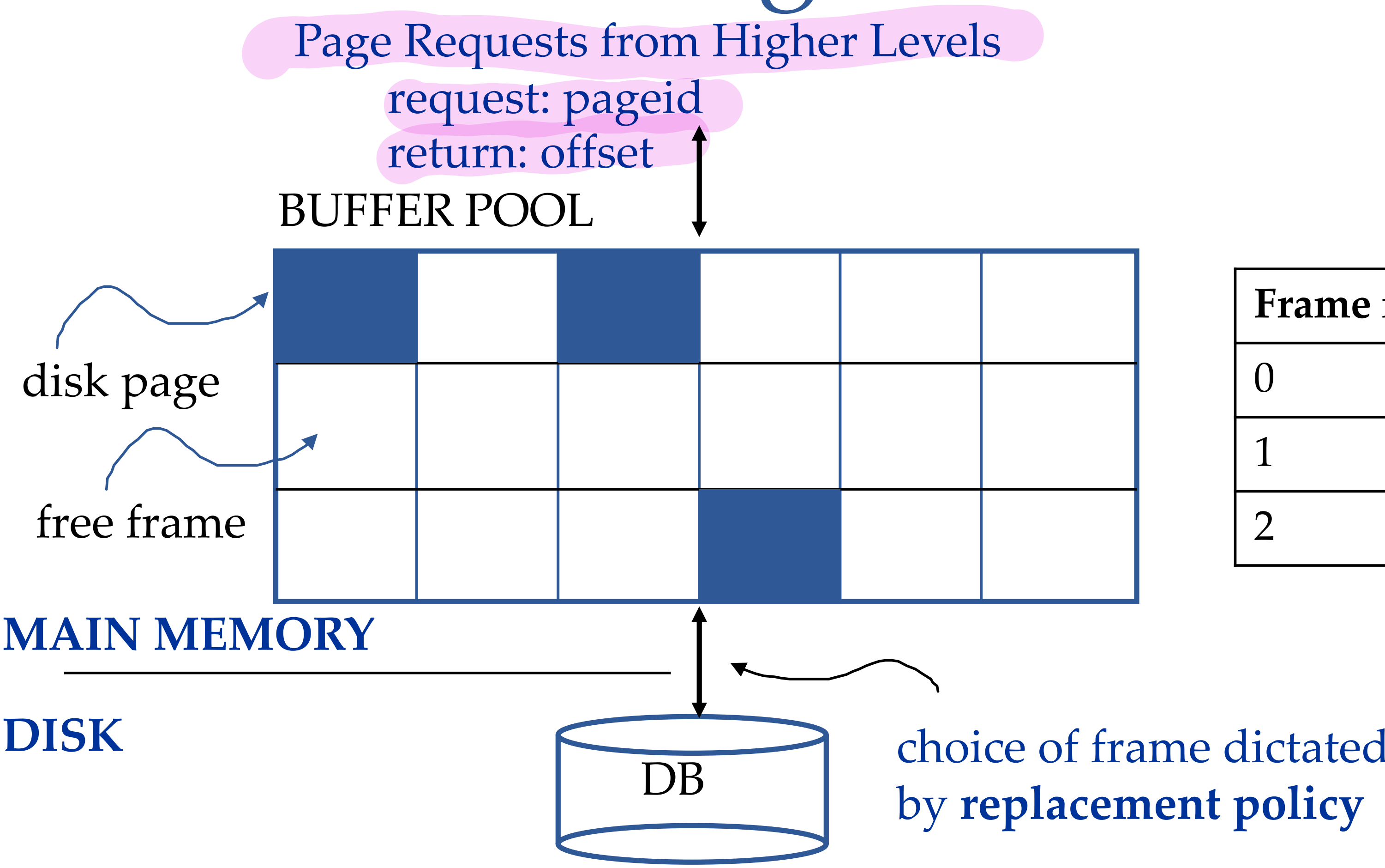
Architecture

must know page id to access

Secondary Storage (stable)



Buffer Management in a DBMS



Frame no	**Page id**
0	202
1	Free
2	421

- Data must be in RAM for DBMS to operate on it!
- Table of <frame#, pageid> pairs is maintained.
- Some more information about each page in buffer is maintained

Loading a page from disk...

- If requested page is not in pool:
  - ☆ If there is an empty frame
    - Choose empty frame
  - ☆ Else (no empty frame)
    - Choose a frame for *replacement*
    - If frame is dirty (current page was modified), write it to disk
  - ☆ Read requested page into chosen frame

☆ How does the buffer manager know whether a page is dirty?

↳ dirty marker

Page Pins

Frame no	Page id	Pin counter
0	202	2
1	431	0
2	…	…

- Pin counter for each frame
- When buffer manager returns a page (offset) to a request from upper layer (after possibly loading the page from disk):
  - Buffer manager increases pin counter of corresponding frame
- When upper layer has finished operations on page
  - Upper layer informs buffer manager (release page) and whether page was updated
  - Buffer manager decreases pin counter of frame (and sets dirty bit if modified)
- When loading a page from disk:
  - Replacement frame must have "pin counter" of 0
- Frame is chosen for replacement by a *replacement policy*:
  - Only unpinned page can be chosen (pin count = 0)
  - Least-recently-used (LRU), Clock, MRU etc.

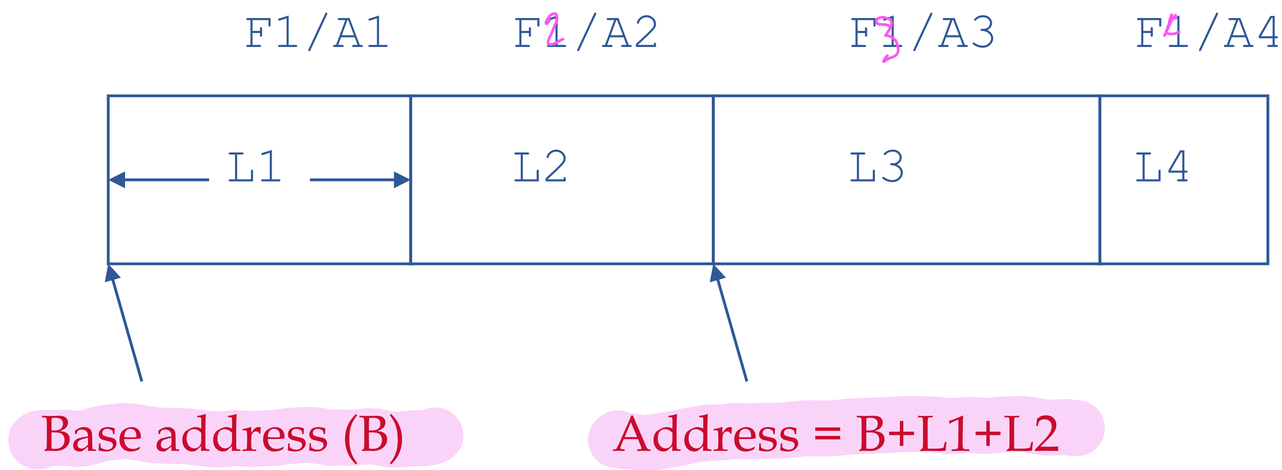
pin count = number of upper layer progs using page

DBMS vs. OS File System

OS does disk space & buffer mgmt: why not let OS manage these tasks?

- Differences in OS support: portability issues
- Some limitations, e.g., files can't span disks.
- Buffer management in DBMS requires ability to:
  - pin a page in buffer pool, force a page to disk (important for implementing CC & recovery),
  - adjust *replacement policy,* and pre-fetch pages based on access patterns in typical DB operations.

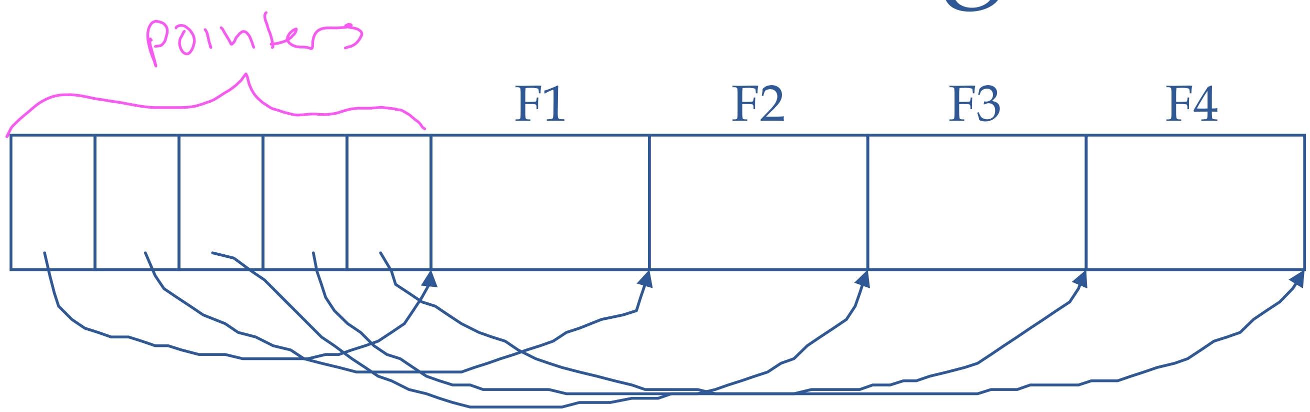
Record Format: Fixed Length



- Length of field (attribute) depends on type
- Works with fixed-length types
- Strings: padding ← to reach max value for string
- Offset of each field easy to calculate

lots of strings
↳ don't use fixed length

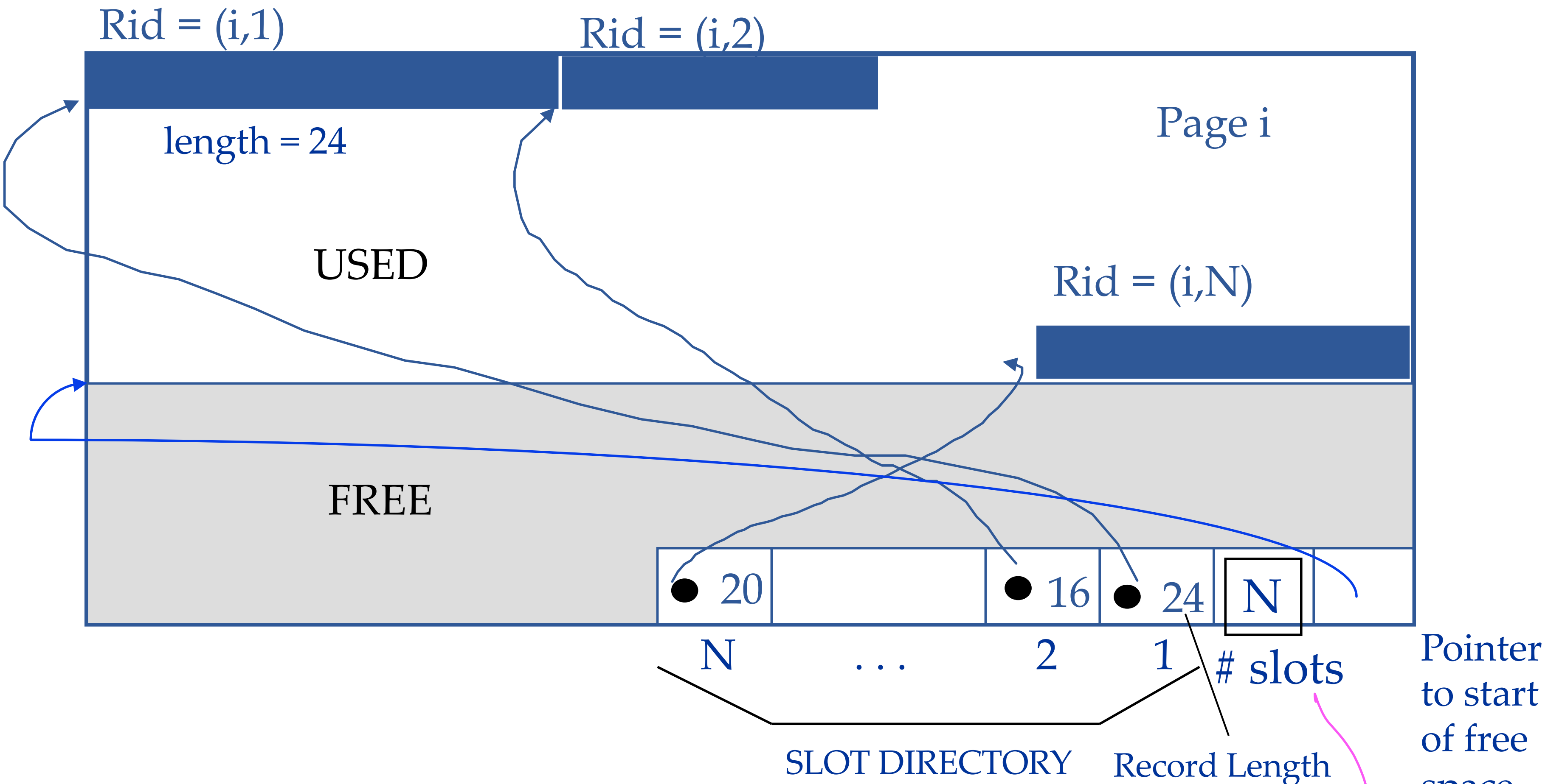
Variable Length



- Second offers direct access to i'th field, efficient storage of *nulls;* small directory overhead

trade off for pointers / meta info

Page Formats: Variable Length Records



- **Record id (rid)** = internal identifier of a record: <page id, slot #>.

  unique semantic identifier

- Can move records on page without changing rid;

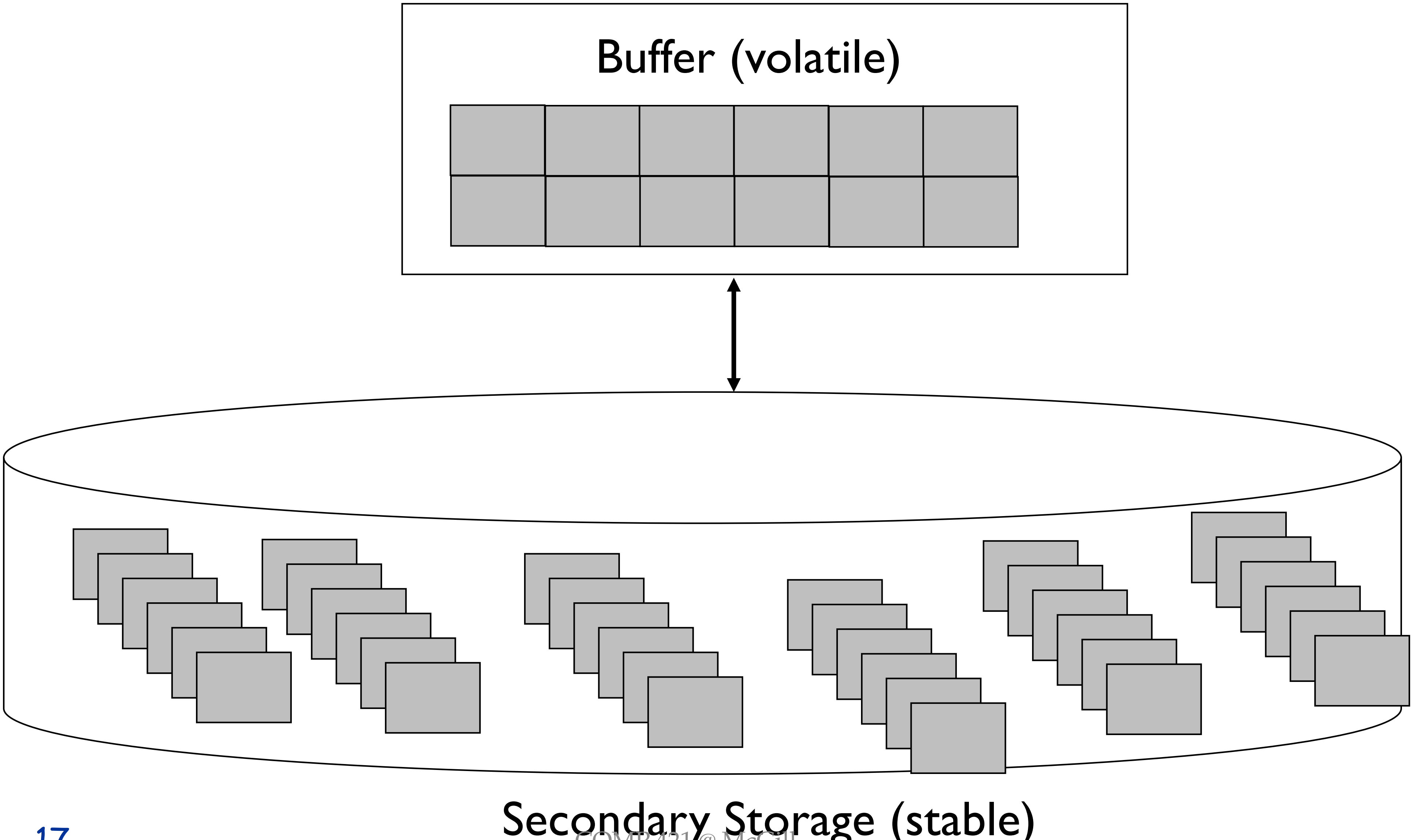
  ↳ just change where pointer points

#records

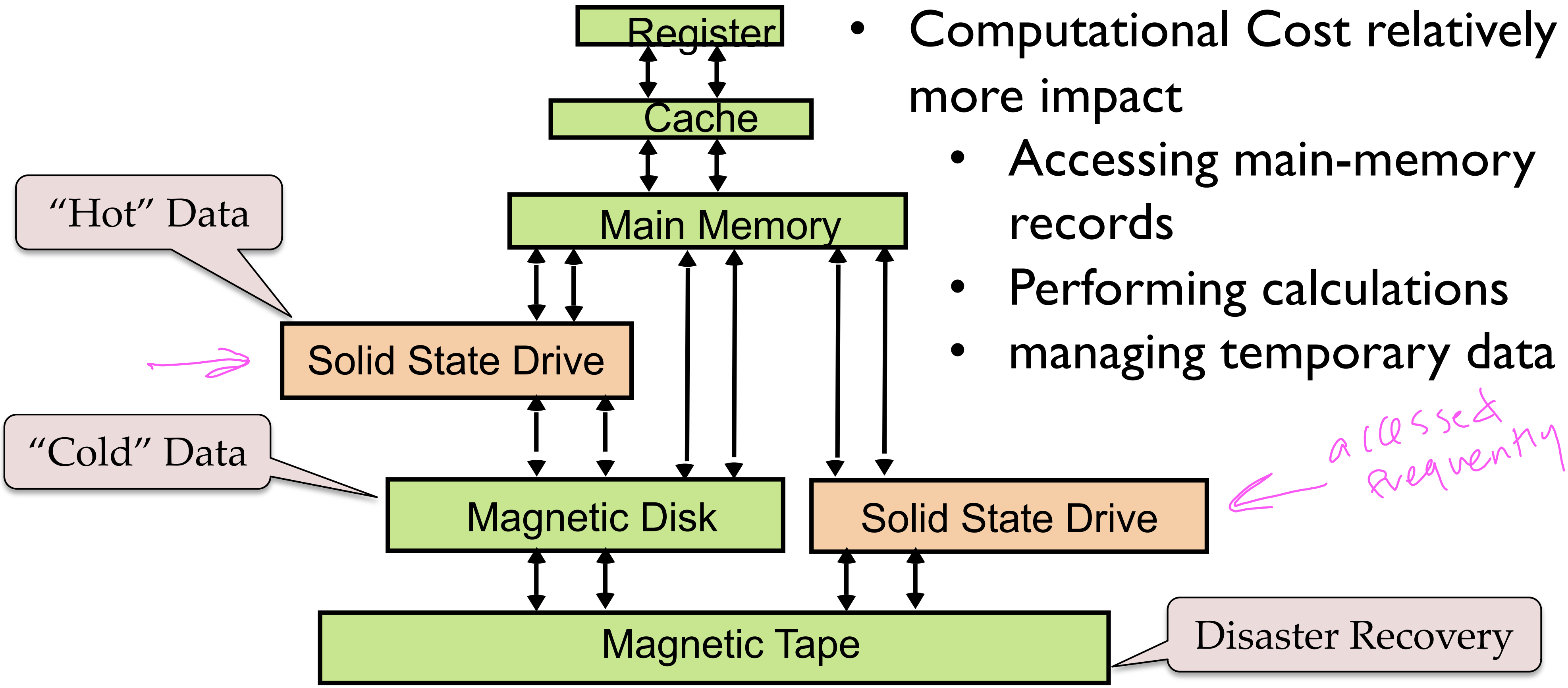
Summary and Assumptions

- Data records distributed across pages
- Data pages need to be in main memory to be accessed
- Page I/O is much more expensive than any computation that then accesses the data in main memory
- Not all data fits into main memory
- → costs calculated by number of I/O
- → optimize to have as little I/O as possible

Assumed Distribution



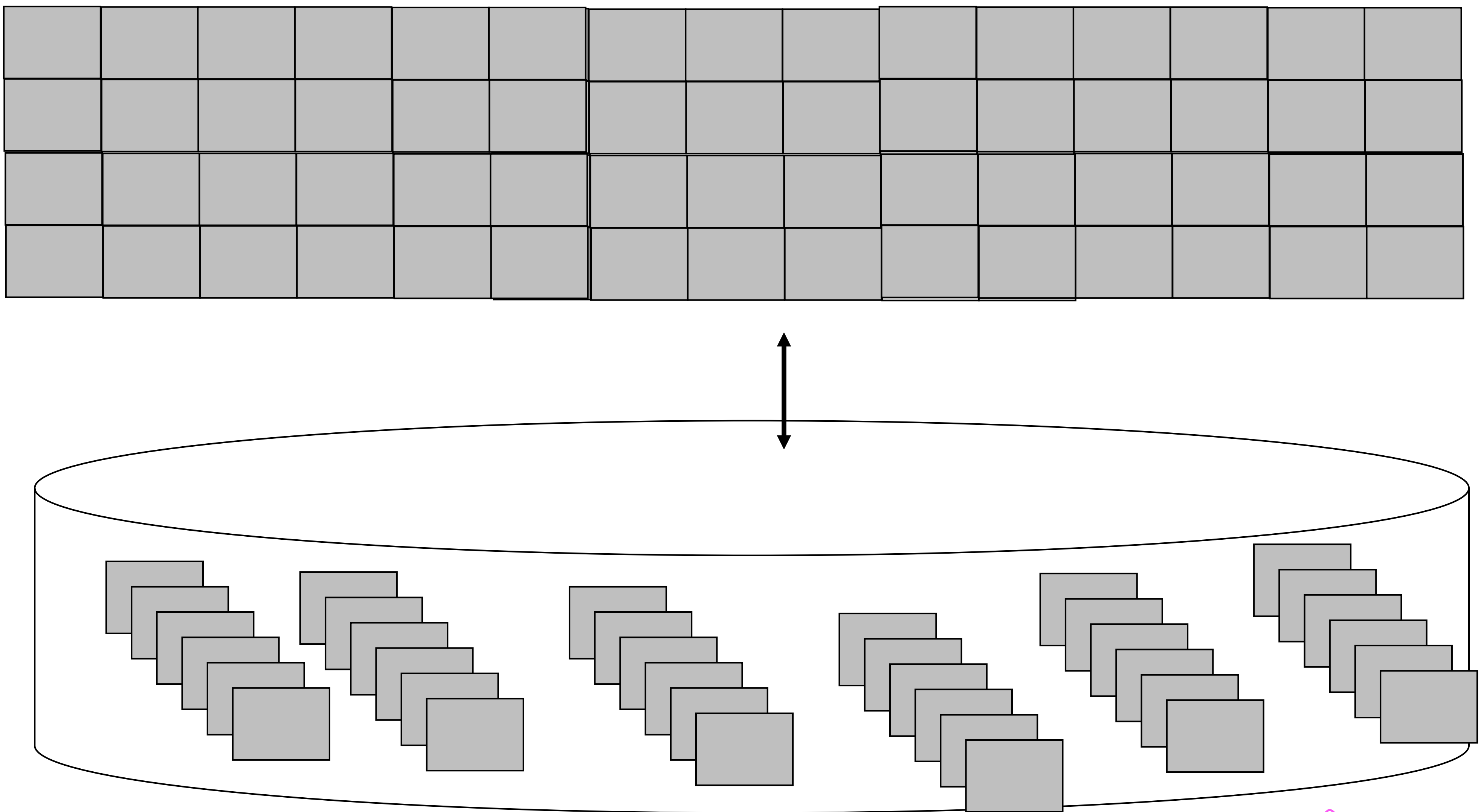
Trends we ignore: Faster stable storage



- Computational Cost relatively more impact
  - Accessing main-memory records
  - Performing calculations
  - managing temporary data

Trends we ignore: memory databases

all data in main memory



— only for persistance / fault tolerance

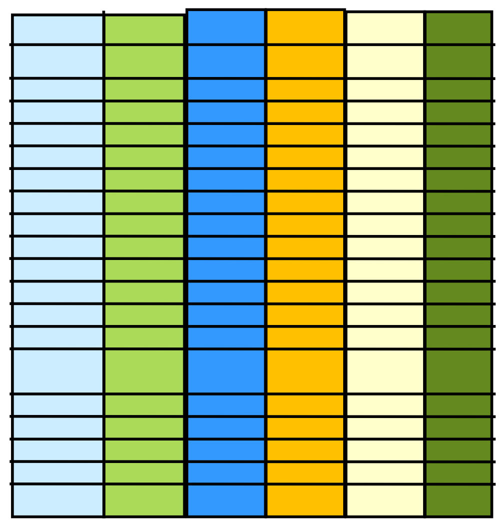
Focus on main memory

Main-memory DBS

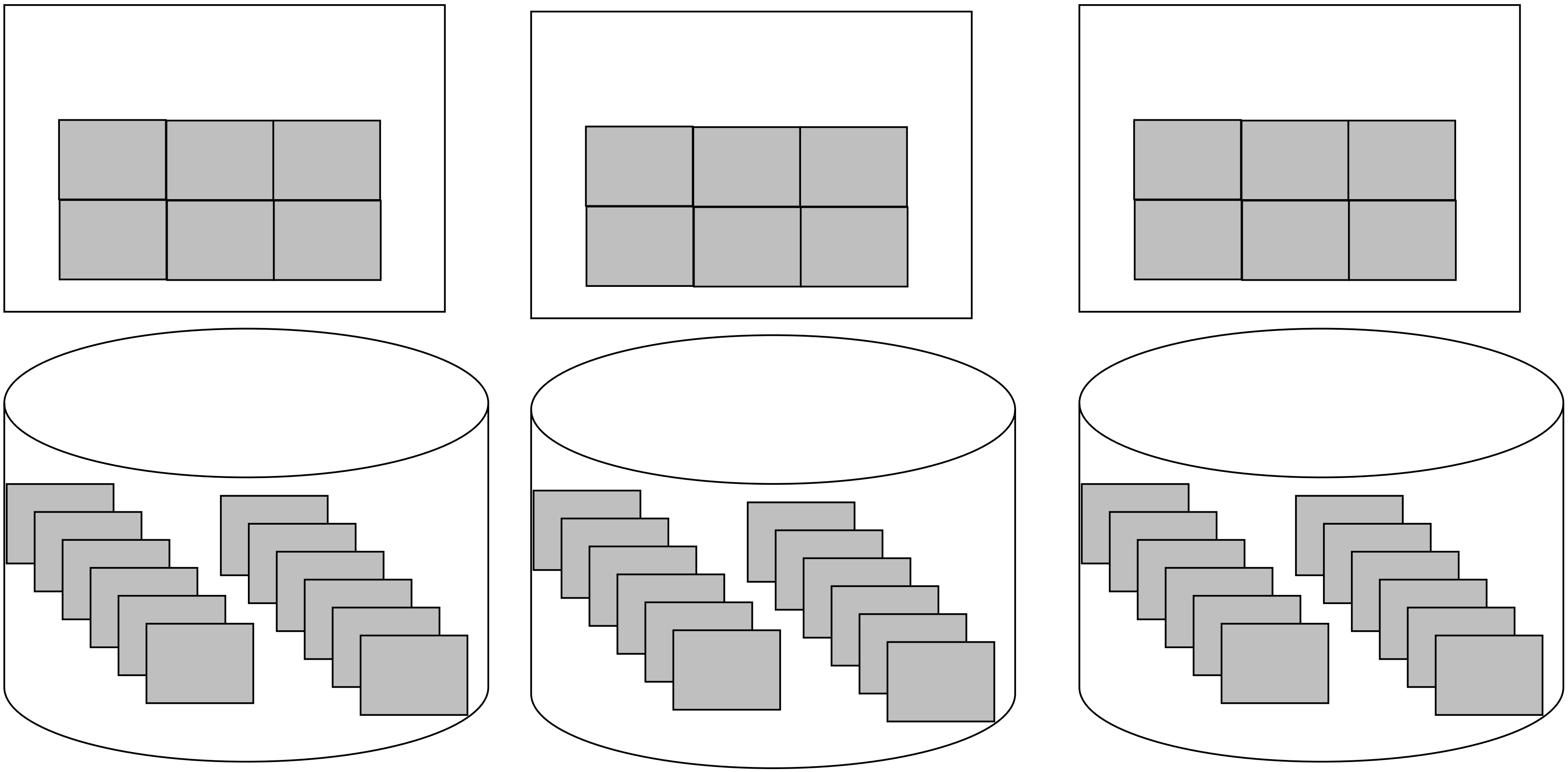
- Only I/O for updates (can be highly optimized)
- Optimizing for quick execution becomes main focus
- Layout in main-memory much more important
  - the concept of splitting everything into pages no more relevant
- Main memory algorithms

Trends we ignore: Column-based database systems

- Store each column of an array of items of the same type
- SELECT sid from Skaters;
- SELECT avg(age) from Skaters;



Later: large scale distribution



Large Scale distribution

- Communication and Coordination Overhead have huge impact on performance