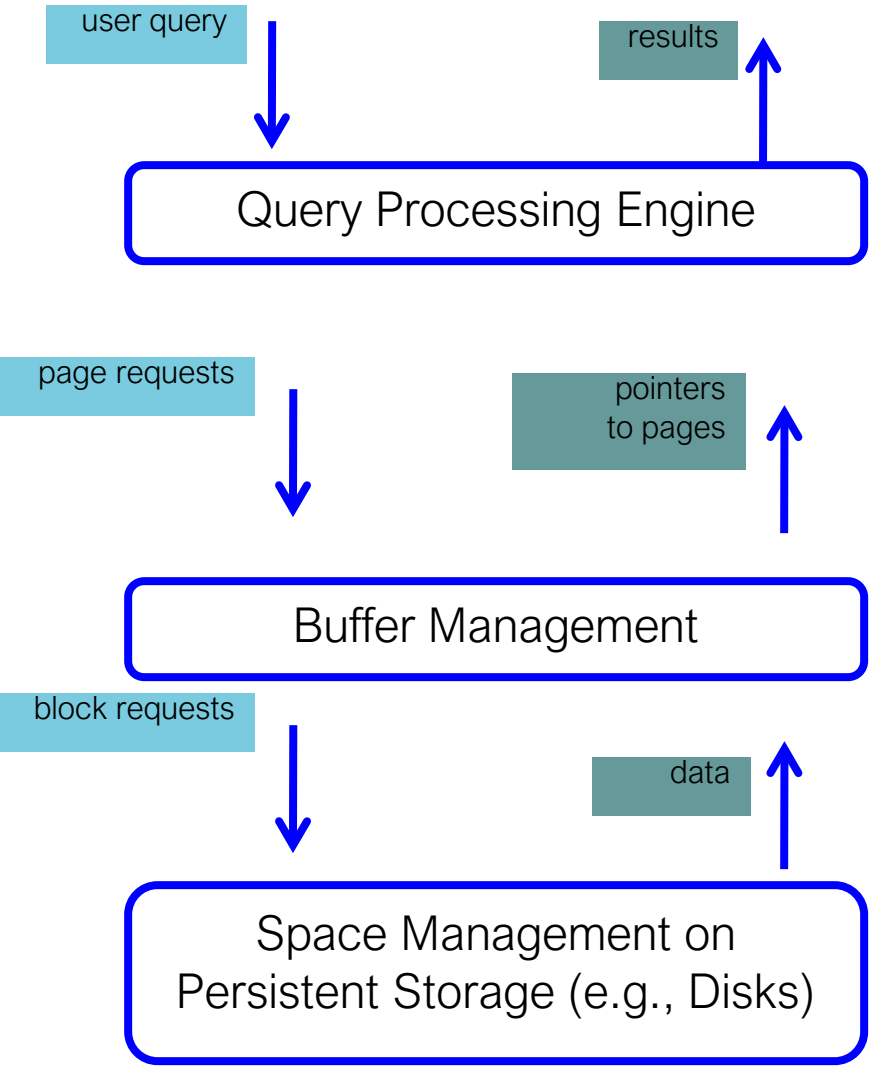
CMSC424: Storage

Query Processing/Storage

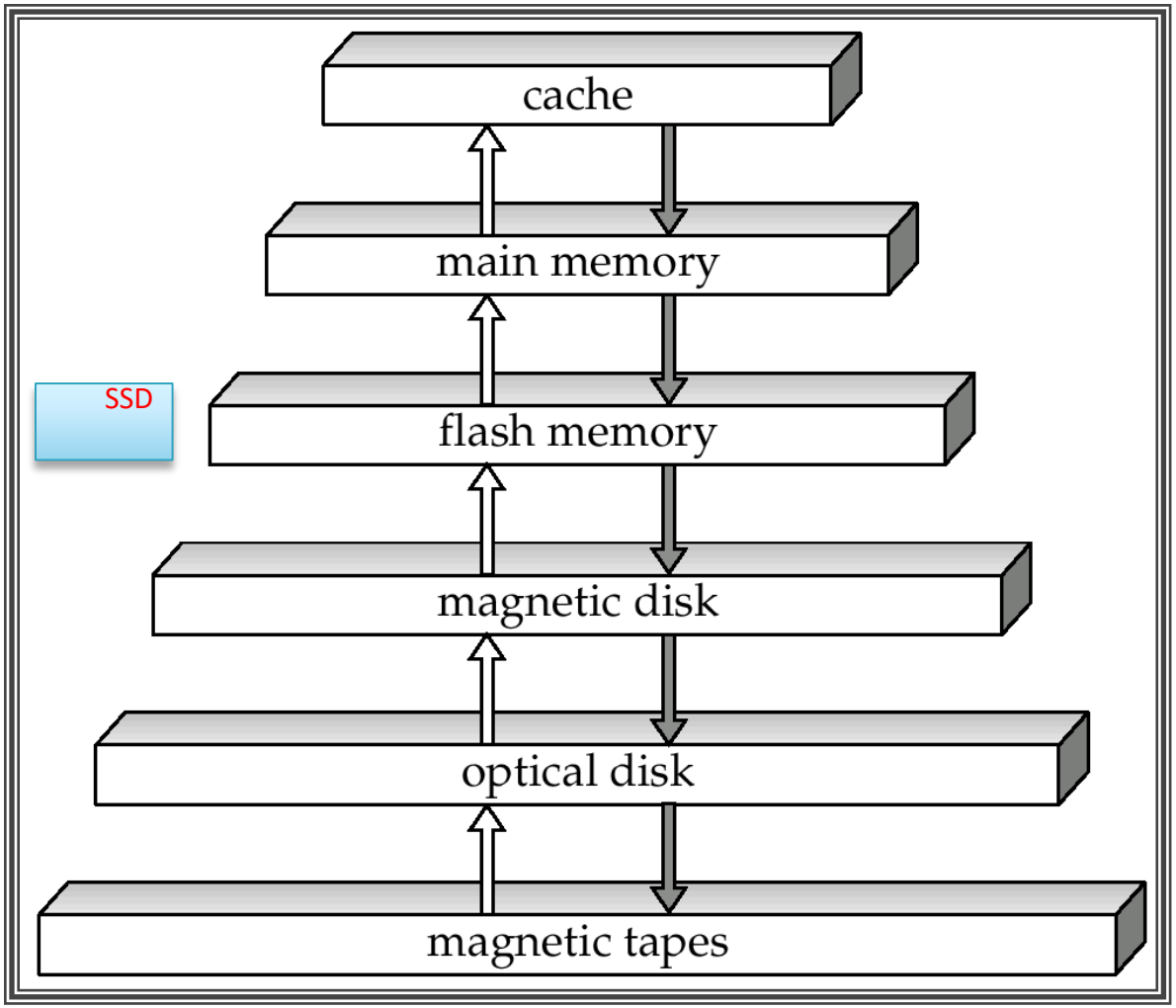


- Given a input user query, decide how to "execute" it
- Specify sequence of pages to be brought in memory
- Operate upon the tuples to produce results

- Bringing pages from disk to memory
- Managing the limited memory

- Storage hierarchy
- Where are relations stored?
- How are tuples mapped to disk blocks?

Storage Hierarchy



Storage Hierarchy: Cache

- Cache
  - Super fast; volatile; Typically on chip
  - L1 vs L2 vs L3 caches
    - L1 about 64KB or so; L2 about 1MB; L3 8MB (on chip) to 256MB (off chip)
    - Huge L3 caches available now-a-days
  - Becoming more and more important to care about this
    - Cache misses are expensive
  - Similar tradeoffs as we will see between main memory and disks

Storage Hierarchy

- Main memory
  - 10s or 100s of ns; volatile
  - Pretty cheap and dropping: 1GByte < 10$
  - Many databases can fit entirely in main memory these days
- Flash memory (EEPROM)
  - Limited number of write/erase cycles
  - Non-volatile, slower than main memory (especially writes)

Storage Hierarchy

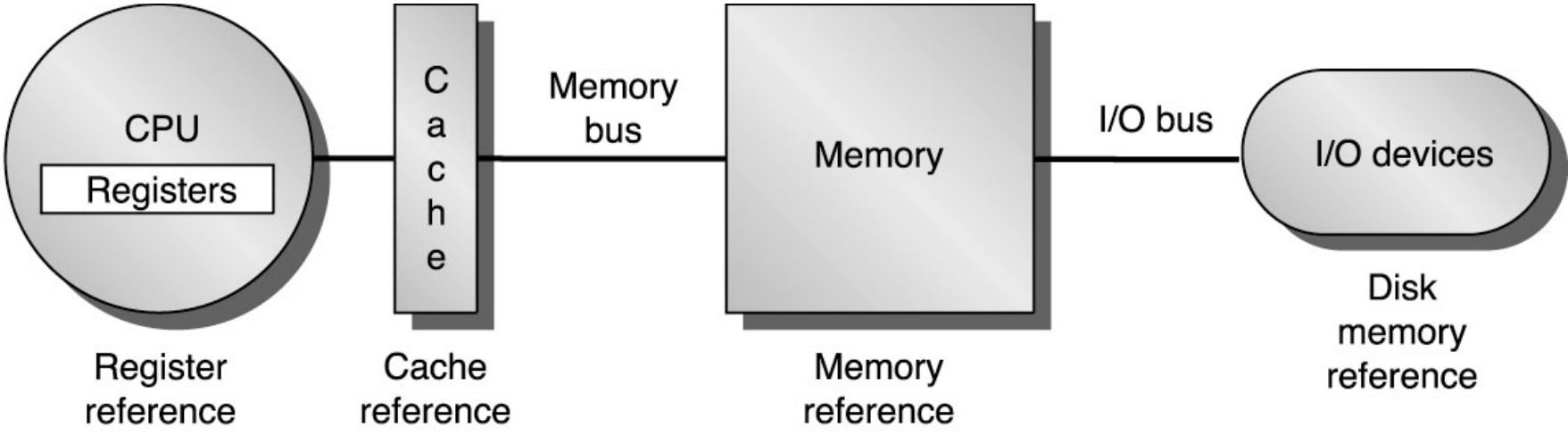
- Magnetic Disk (Hard Drive)
  - Non-volatile
  - Much slower than memory (more details soon)
- Optical Storage - CDs/DVDs
  - Sometimes used as backups
  - Very slow to write (if possible at all)
- Tape storage
  - Backups; super-cheap; painful to access
  - IBM recently released a secure tape drive storage solution

Storage…

- Primary
  - e.g. Main memory, cache; typically volatile, fast
- Secondary
  - e.g. Disks; Solid State Drives (SSD); non-volatile
- Tertiary
  - e.g. Tapes; Non-volatile, super cheap, slow

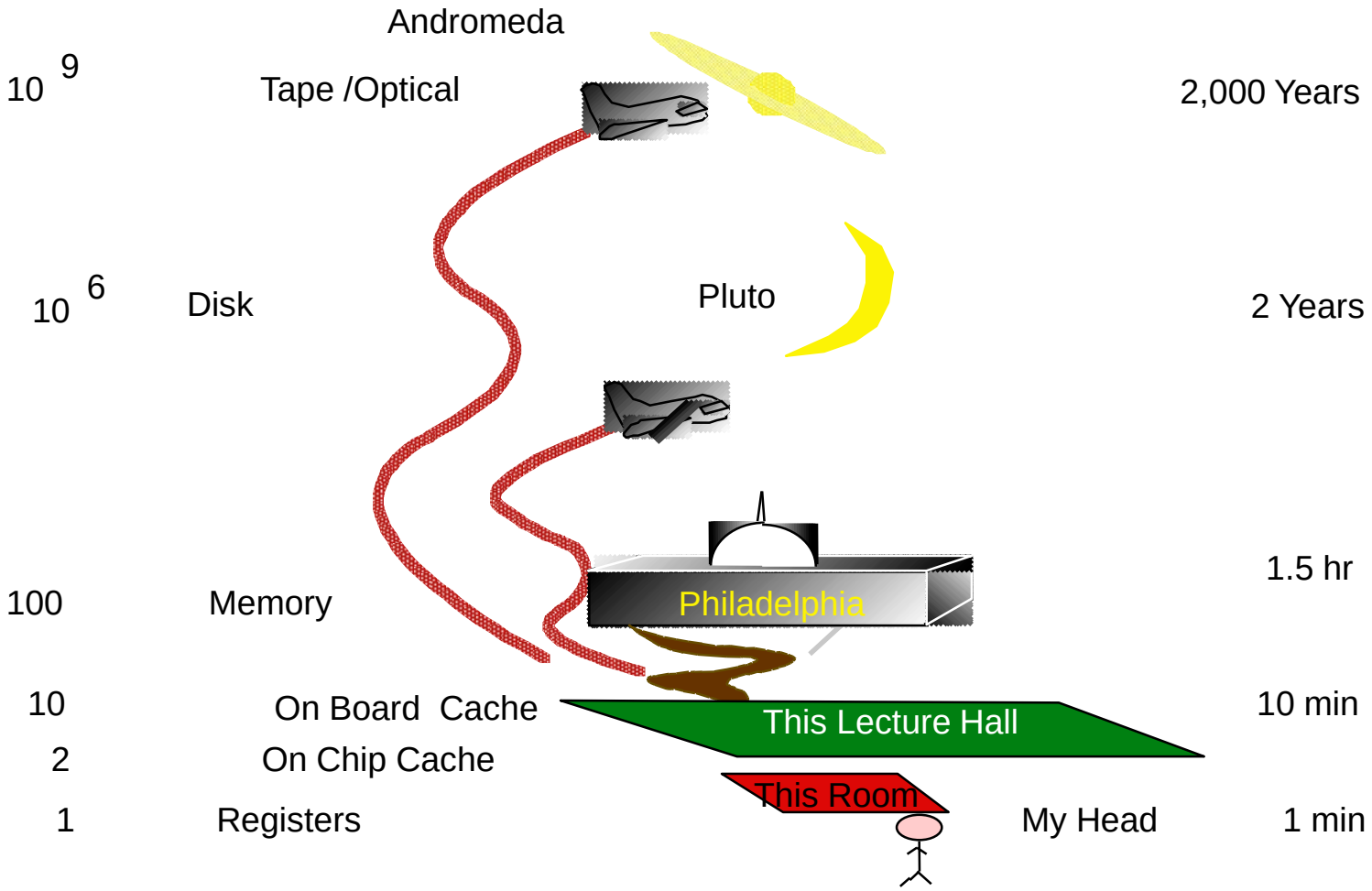
Simple Storage Hierarchy

- Note many orders of magnitude change in characteristics between levels:

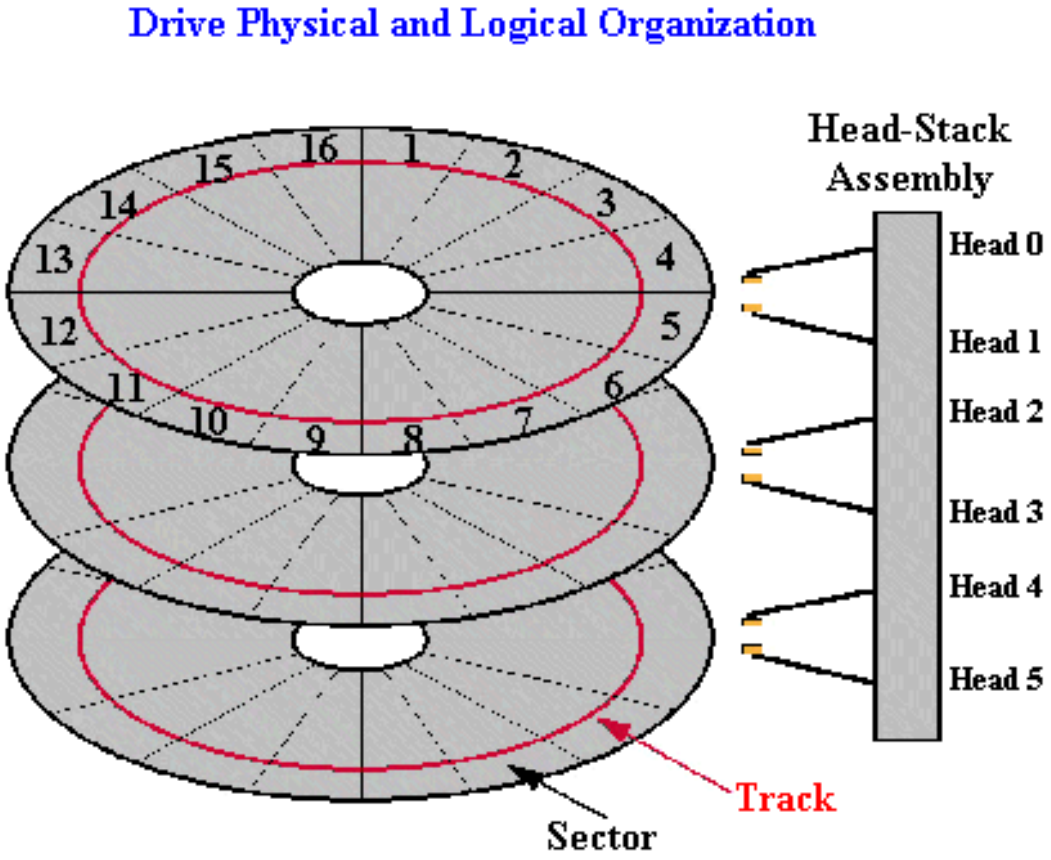


Size	**1 KB**	**16 MB**	**512 GB**	**2 TB**
Access Time (ns)	**0.25-0.5**	**0.5-25**	**50-250**	**5000000**
Bandwidth (MB/s)	**50K-500K**	**5000-20000**	**2500-10000**	**50-500**

Jim Gray's Storage Latency Analogy: How Far Away is the Data?



Magnetic Disk



Accessing Data

- Accessing a sector
  - Time to *seek* to the track (seek time)
    - average 4 to 10ms
  - Waiting for the sector to get under the head (rotational latency)
    - average 4 to 11ms
  - Time to transfer the data (transfer time)
    - very low
  - About 10ms per access
    - So if randomly accessed blocks, can only do 100 block transfers / sec
    - 100 x 512bytes = 50 KB/s
- Data transfer rates
  - Rate at which data can be transferred (w/o any seeks)
  - 30-50MB/s to up to 500MB/s (Compare to above)
    - Seeks are bad !

Seagate Barracuda: 1TB

- Heads 8, Disks 4
- Bytes per sector: 512 bytes
- Defaults sectors per track: 63
- Spindle speed: 7200 rpm
- Average seek: 8.5-9.5msec
- Average latency: 4.16msec

(We also worry about power now)

Reliability

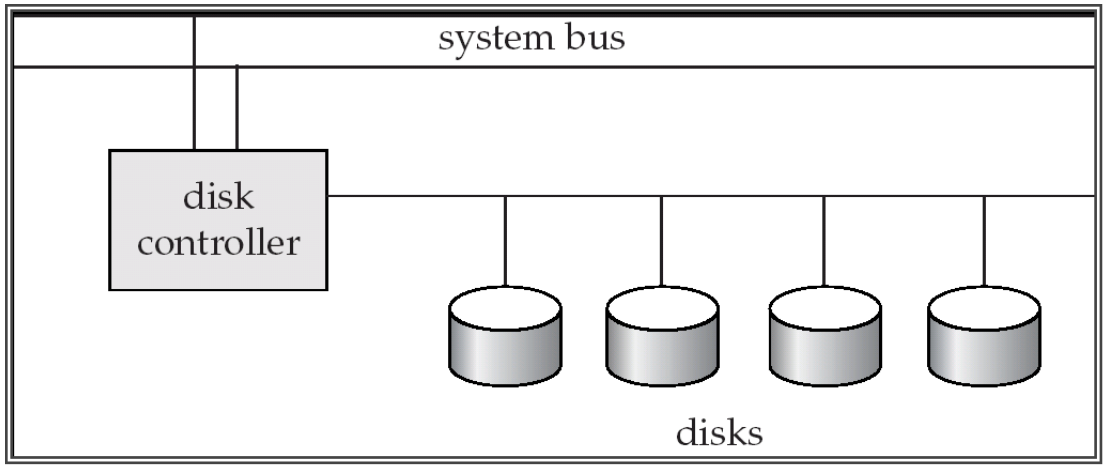
- Mean time to/between failure (MTTF/MTBF):
  - 57 to 136 years
- Consider:
  - 1000 new disks
  - 1,200,000 hours of MTTF each
  - On average, one will fail 1200 hours = 50 days !

Sequential Reads vs. Seeks

- Disk density doubling every 18 months
- Disk bandwitdth rises ~sqrt(density)
- Arm movement increases ~7%/year
- Today:
  - ~0.05ms to read a sequential block
  - ~5 ms to read a random block
- Bottom line: DBMS must make sure data that is co-accessed is co-located

Disk Controller

- Interface between the disk and the CPU
- Accepts the commands
- *checksums* to verify correctness
- Remaps bad sectors



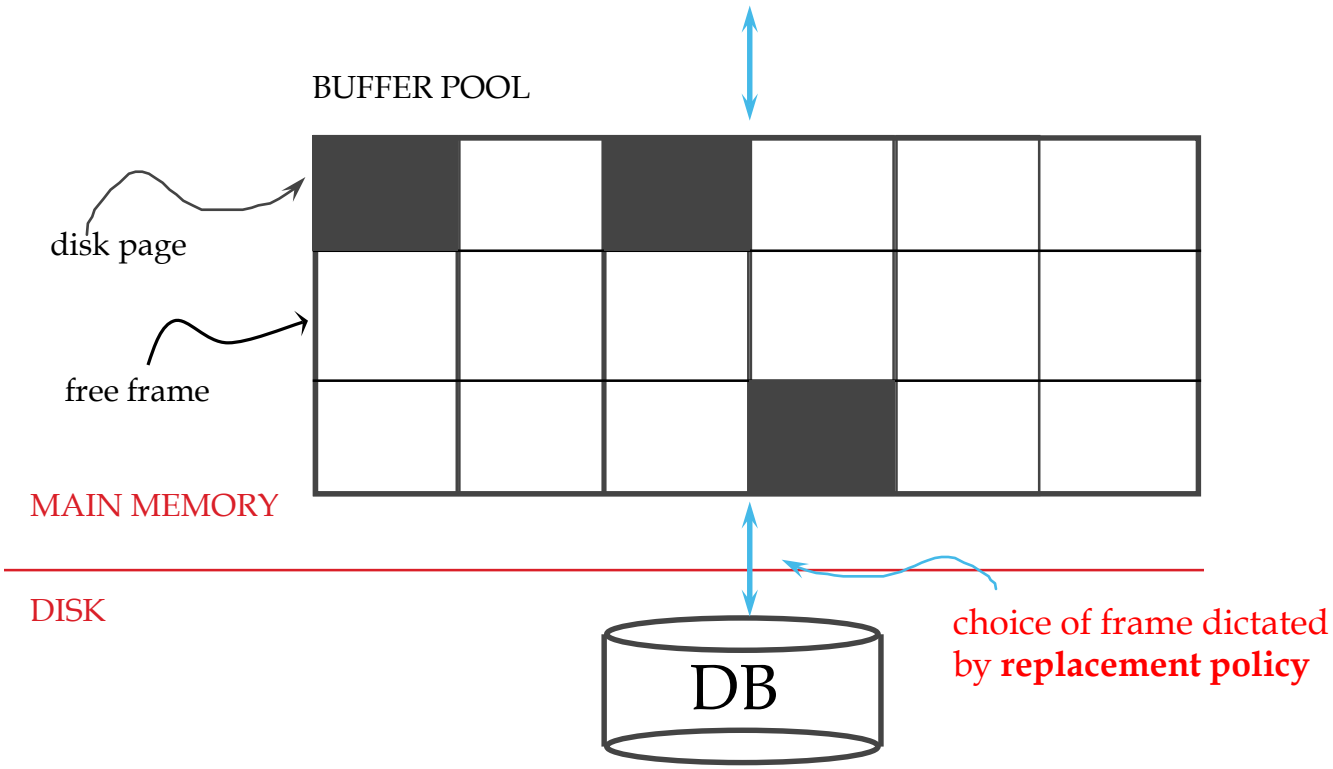
Optimizing block accesses

- Typically sectors too small
- Block: A contiguous sequence of sectors
  - 512 bytes to several Kbytes
  - All data transfers done in units of blocks
- Scheduling of block access requests ?
  - Considerations: *performance* and *fairness*
  - *Elevator algorithm*

Solid State Drives

- Essentially flash that emulates hard disk interfaces
- No seeks → Much better random reads performance
- Writes are slower, the number of writes at the same location limited
  - *Must write an entire block at a time*
- About a factor of 10 more expensive right now

Buffer Manager



Buffer Manager

- When QP wants a block, it asks buffer manager
  - The block must be in memory to operate upon
- Buffer manager:
  - If block already in memory: return a pointer
  - If not:
    - evict a current page
      - write it to temporary storage, or
      - write it back to its original location, or
      - or toss it (if “clean”)
    - and make a request to the storage subsystem to fetch it

Buffer Manager

- Buffer replacement policies
  - What page to evict ?
  - LRU: Least Recently Used
    - Throw out the page that was not used in a long time
  - MRU: Most Recently Used
    - The opposite
    - Why ?
  - Clock ?
    - An efficient implementation of LRU