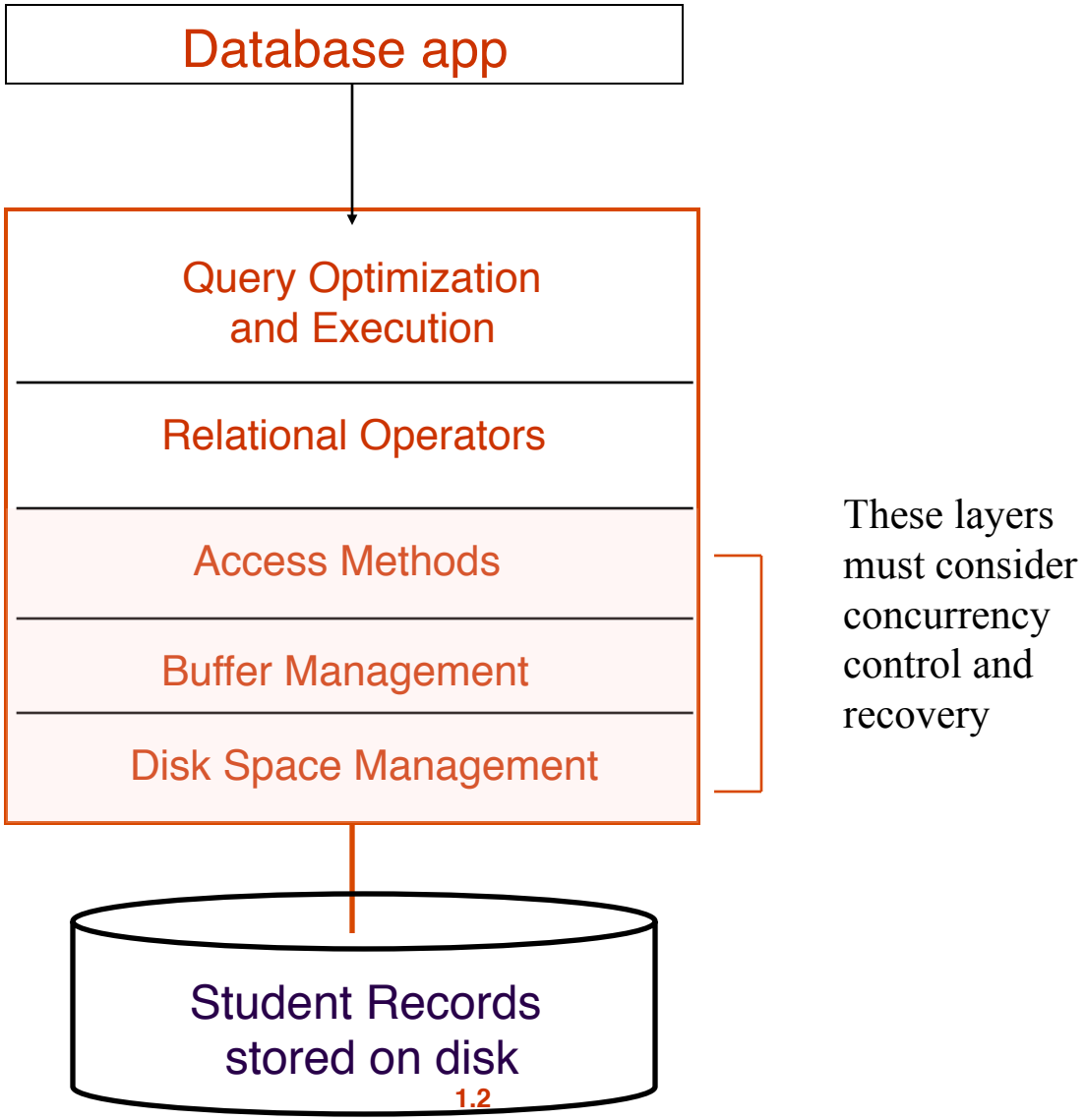
CAS CS 460/660
Introduction to Database Systems

Disks, Buffer Manager

DBMS architecture



These layers must consider concurrency control and recovery

Why Not Store It All in Main Memory?

- *Costs too much.* $100 will buy you either

  ~100 GB of RAM or around 2000 GB (2 TB) of disk today.

  - High-end Databases today can be in the Petabyte (1000TB) range.
  - Approx 60% of the cost of a production system is in the disks.

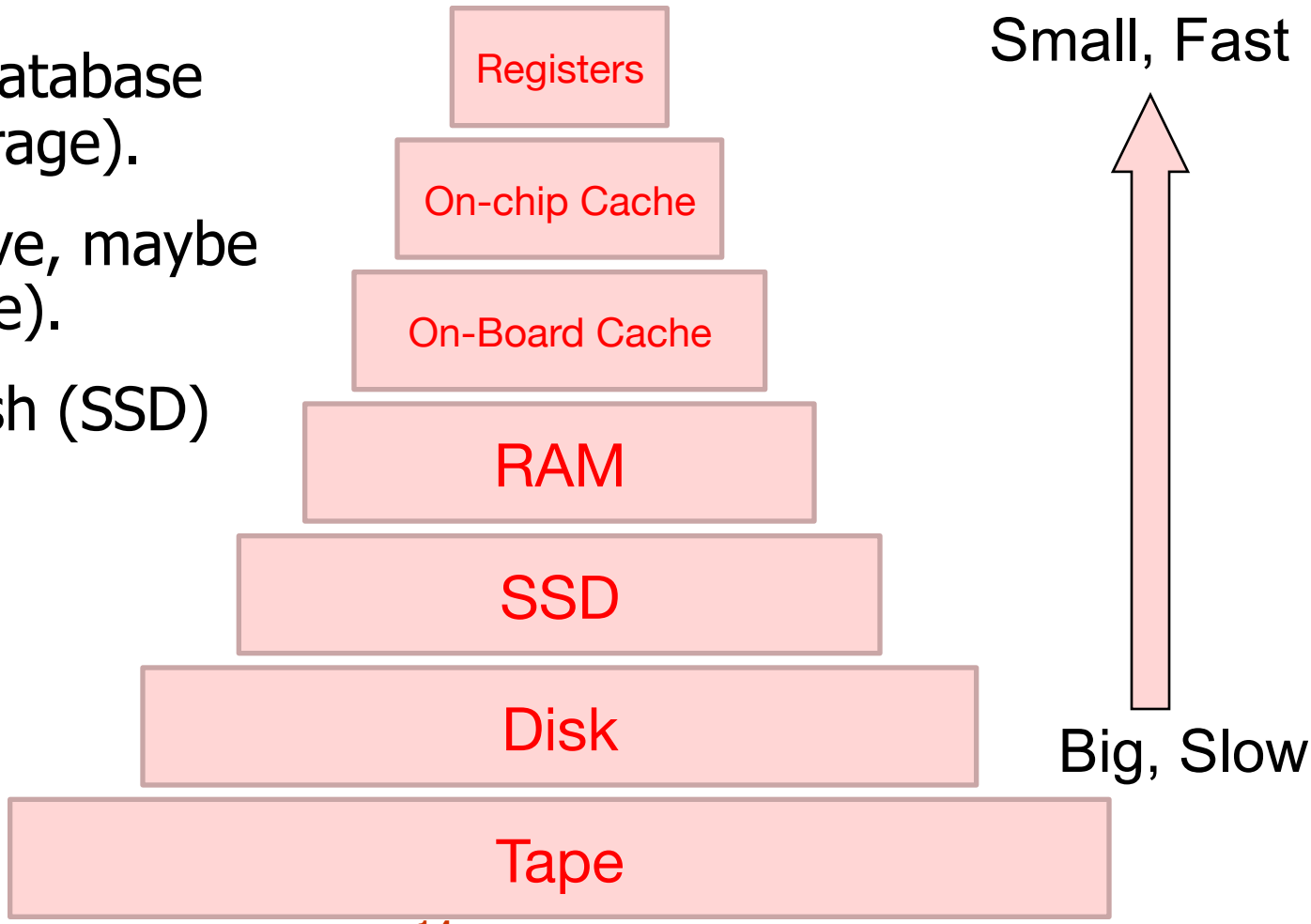
- *Main memory is volatile.* We want data to be saved between runs. (Obviously!)

- Note, some specialized systems do store entire database in main memory.

  - Vendors claim 10x speed up vs. traditional DBMS running in main memory.

The Storage Hierarchy

- Main memory (RAM) for currently used data.
- Disk for main database (secondary storage).
- Tapes for archive, maybe (tertiary storage).
- The role of Flash (SSD) still unclear



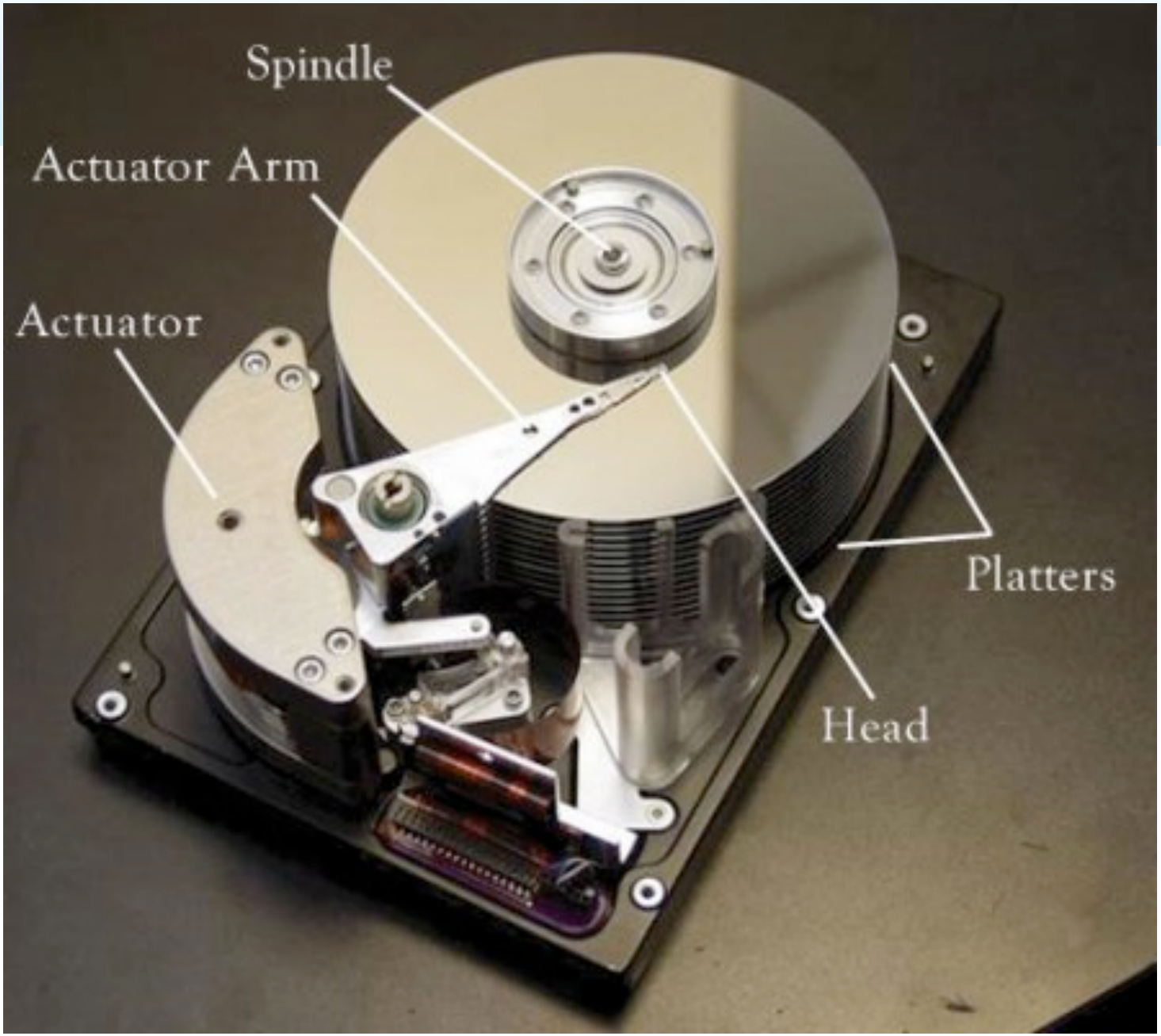
Economics

For $1000, you can get:

- ~400GB of RAM
- ~2.5TB of Solid State Disk
- ~30TB of Magnetic Disk

Disks and Files

- Today: Most data is stored on magnetic disks.
  - Disks are a mechanical anachronism!
- Major implications!
  - No “pointer derefs”. Instead, an API:
    - READ: transfer “page” of data from disk to RAM.
    - WRITE: transfer “page” of data from RAM to disk.
  - Both API calls expensive
    - Plan carefully!
  - An explicit API can be a good thing
    - Minimizes the kind of pointer errors you see in C


Spindle
Actuator Arm
Actuator
Platters
Head

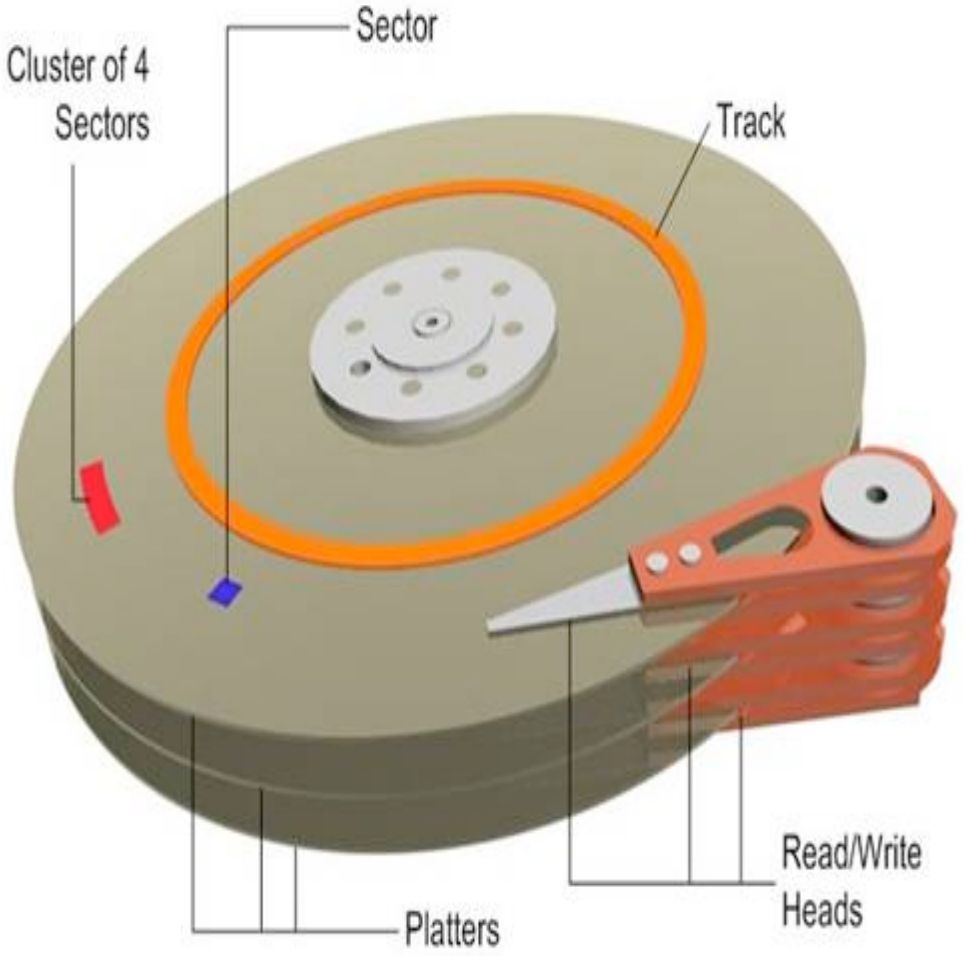
Anatomy of a Disk

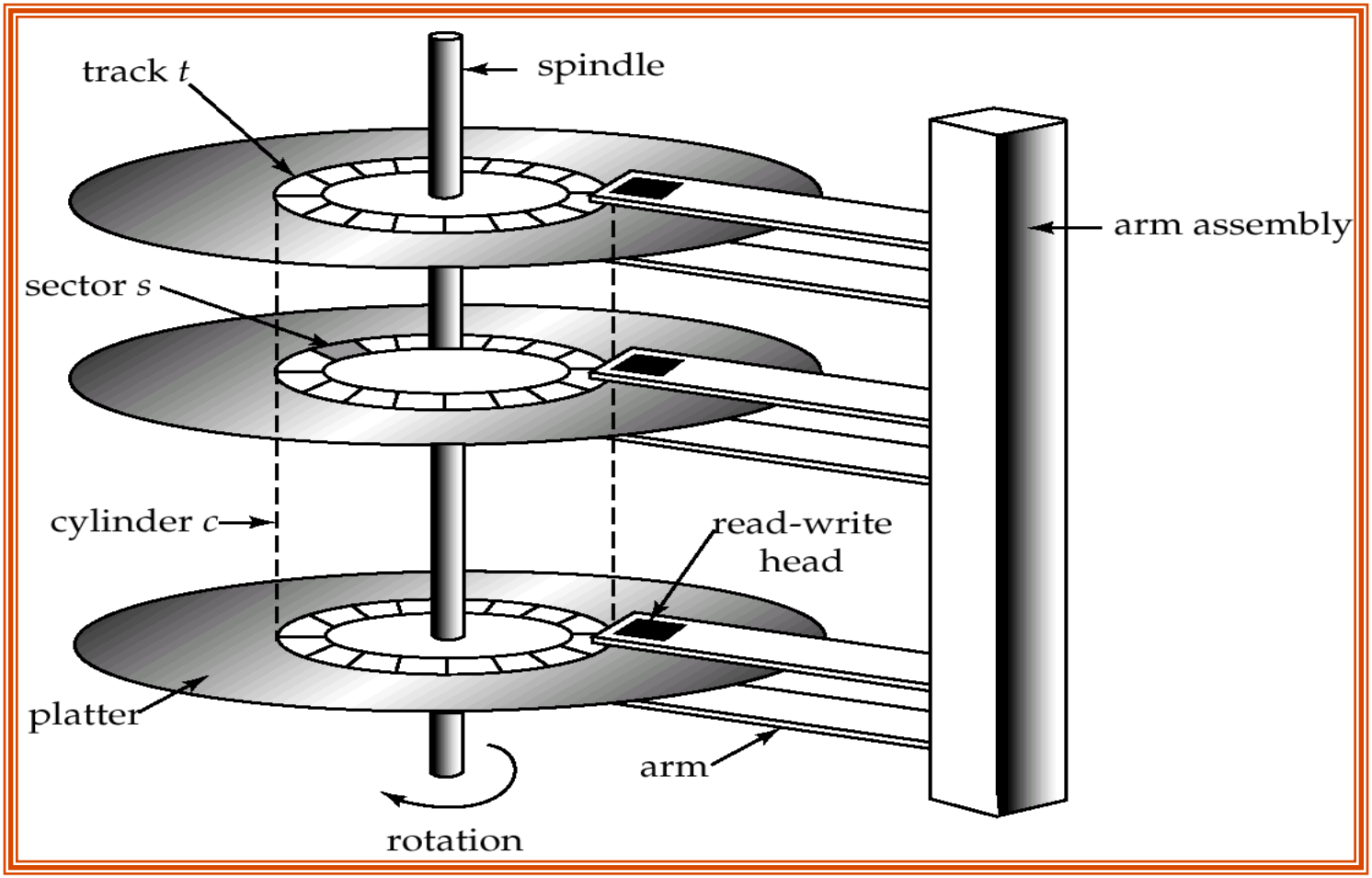
The platters spin

The arm assembly is moved in or out to position a head on a desired track. Tracks under heads make a *cylinder* (imaginary!).

Only one head reads/ writes at any one time.

- *Block size* is a multiple of *sector size* (which is fixed)




track t
spindle
arm assembly
sector s
cylinder c
read-write
head
platter
arm
rotation

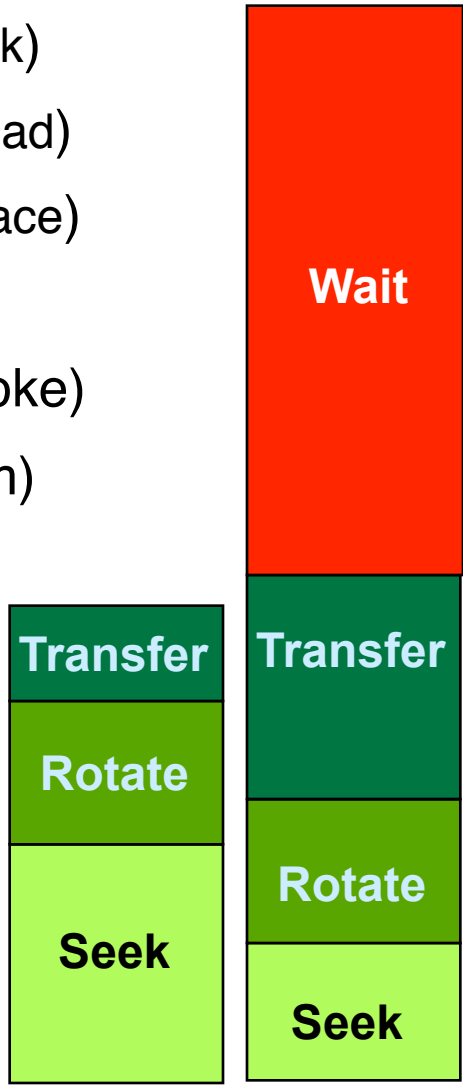
Accessing a Disk Page

- Time to access (read/write) a disk block:
  - ***seek time*** (moving arms to position disk head on track)
  - ***rotational delay*** (waiting for block to rotate under head)
  - ***transfer time*** (actually moving data to/from disk surface)
- Seek time and rotational delay dominate.
  - Seek time varies from about 1 to 15msec (full stroke)
  - Rotational delay varies from 0 to 8msec (7200rpm)
  - Transfer rate is < 0.1msec per 8KB block
- Key to lower I/O cost:

  reduce seek/rotation delays!

  Hardware vs. software solutions?

- Also note: For **shared disks** most time spent waiting in queue for access to arm/controller



Arranging Pages on Disk

- `Next' block concept:
  - blocks on same track, followed by
  - blocks on same cylinder, followed by
  - blocks on adjacent cylinder
- Arrange file pages sequentially on disk
  - minimize seek and rotational delay.
- For a sequential scan, pre-fetch
  - several pages at a time!

- We use Block/Page interchangeably!!

From the DB Administrator's View

Modern disk structures are so complex even industry experts refer to them as "black boxes". Today there is no alignment to physical disk sectors, no matter what we believe. Disks do not map sectors to physical regions in a way that we can understand from outside the box; the simplistic "geometry" reported by the device is an artifice.

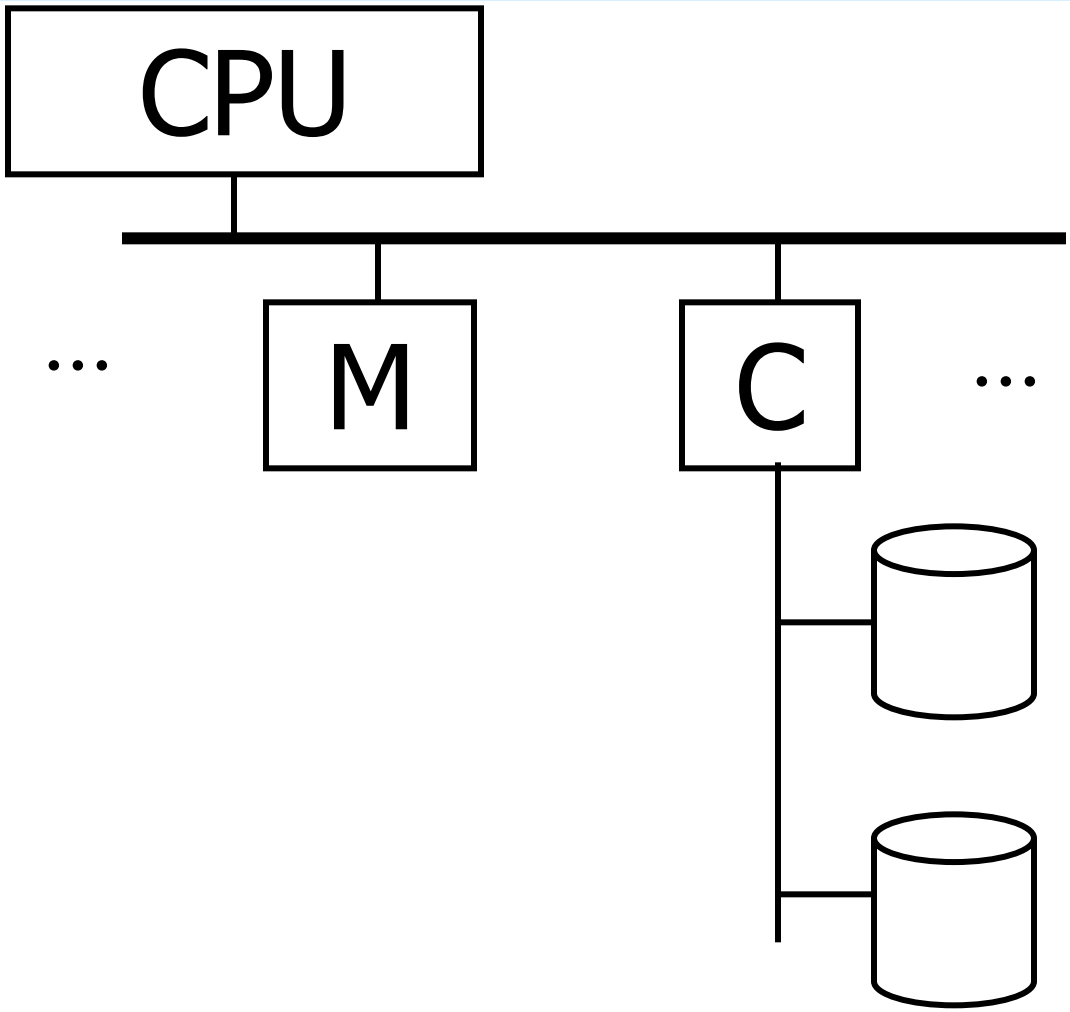
from Microsoft's "Disk Partition Alignment Best Practices for SQL Server"

Disk Space Management

- Lowest layer of DBMS software manages space on disk (using OS file system or not?).
- Higher levels call upon this layer to:
  - allocate/de-allocate a page
  - read/write a page
- Best if a request for a *sequence* of pages is satisfied by pages stored sequentially on disk! Higher levels don't need to know if/how this is done, or how free space is managed.

Notes on Flash (SSD)

- Various technologies, we focus on NAND
  - suited for volume data storage
  - alternative: NOR Flash
- Read is random access and fast
  - E.g. 512 ***Bytes*** at a time
- Write is coarser grained and slower
  - E.g. 16-512 ***KBytes*** at a time.
  - Can get slower over time
- Some concern about write endurance
  - 100K cycle lifetimes?
- Still changing quickly



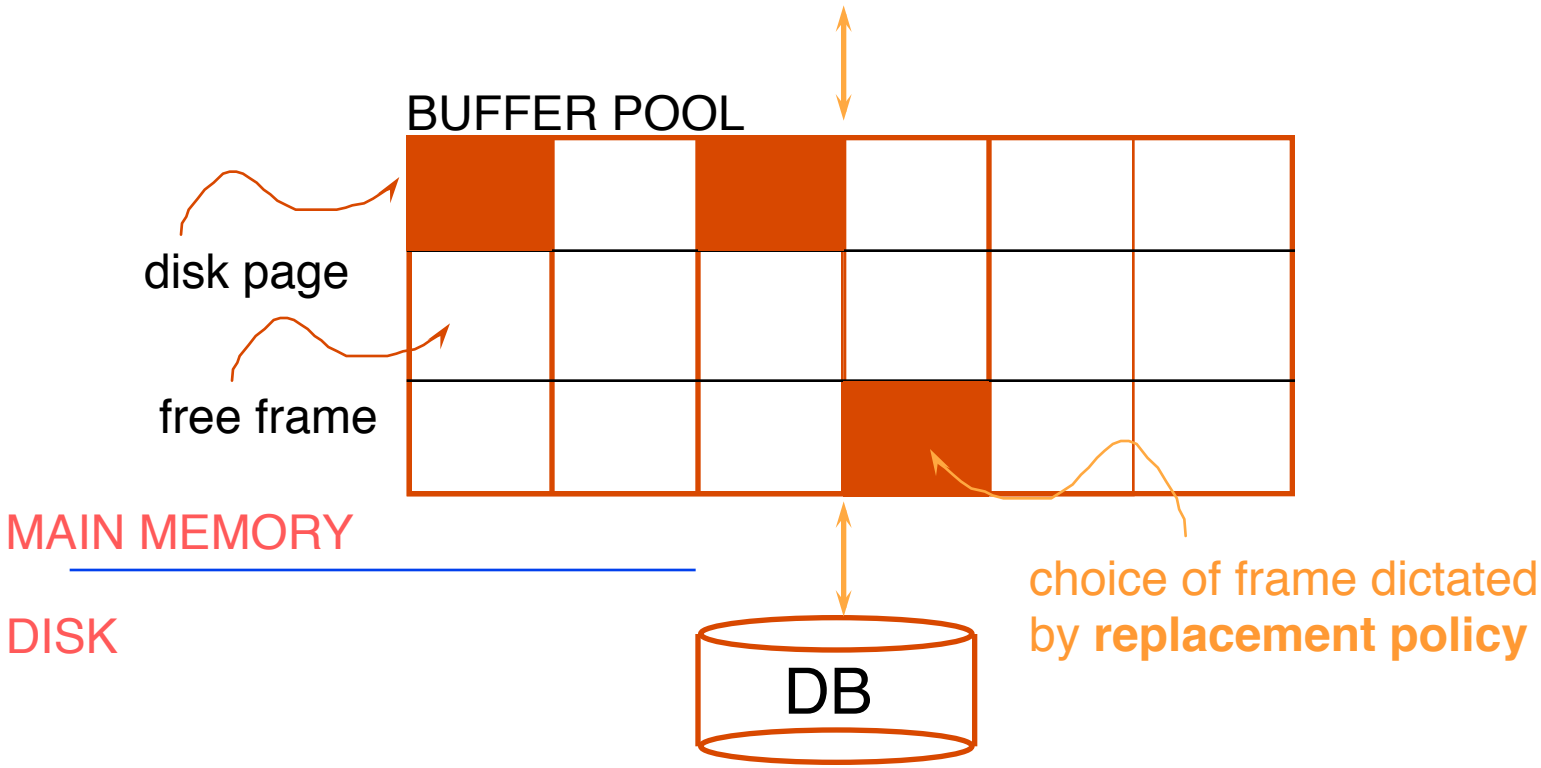
Typical Computer

Secondary Storage

Storage Pragmatics & Trends

- Many significant DBs are not that big.
  - Daily weather, round the globe, 1929-2009: 20GB
  - 2000 US Census: 200GB
  - 2009 English Wikipedia: 14GB
  - NYC Taxi Rides (~20GB per year)
- But data sizes grow faster than Moore's Law
- What is the role of disk, flash, RAM?
  - The subject of much debate/concern!

Buffer Management in a DBMS



- *Data must be in RAM for DBMS to operate on it!*
  - *The query processor refers to data using virtual memory addresses.*
- *Buffer Mgr hides the fact that not all data is in RAM*

Some Terminology…

- Disk Page – the unit of transfer between the disk and memory
  - Typically set as a config parameter for the DBMS.
  - Typical value between 4 KBytes to 32 KBytes.
- Frame – a unit of memory
  - Typically the same size as the Disk Page Size
- Buffer Pool – a collection of frames used by the DBMS to temporarily keep data for use by the query processor (CPU).
  - note: We will sometime use the term “buffer” and “frame” synonymously.

When a Page is Requested ...

- If requested page IS in the pool:
  - *Pin* the page and return its address.
- Else, if requested page IS NOT in the pool:
  - If a free frame exists, choose it, Else:
    - Choose a frame for *replacement (only un-pinned pages are candidates)*
    - If chosen frame is “dirty”, write it to disk
  - Read requested page into chosen frame
  - *Pin* the page and return its address.

Q: What information about buffers and their contents must the system maintain?

Buffer Control Blocks (BCBs):

<frame#, pageid, pin_count, dirty>

- A page may be requested many times, so
  - a *pin count* is used.
  - To pin a page, pin_count++
  - A page is a candidate for replacement iff *pin_count* == 0 (*"unpinned"*)
- Requestor of page must eventually unpin it.
  - pin_count--
- Must also indicate if page has been modified:
  - *dirty* bit is used for this.

Q: Why is this important?

Q: How should BCB's be organized?

Additional Buffer Mgr Notes

- BCB's are hash indexed by pageID
- Concurrency Control & Recovery may entail additional I/O when a frame is chosen for replacement.

  (*Write-Ahead Log* protocol; more later.)
- If requests can be predicted (e.g., sequential scans) pages can be pre-fetched several pages at a time.

Buffer Replacement Policy

- Frame is chosen for replacement by a *replacement policy:*
  - Least-recently-used (LRU), MRU, Clock, etc.
- This policy can have big impact on the number of disk reads and writes.
  - Remember, these are slooooooooooow.
- **BIG IDEA** – throw out the page that you are least likely to need in the future.
  - Q: How do you predict the future?

LRU Replacement Policy

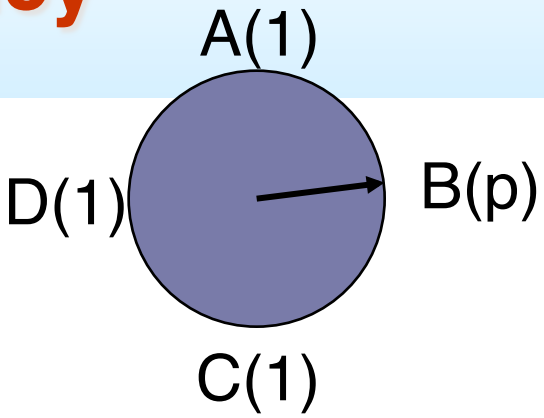
Least Recently Used (LRU)

1) for each page in buffer pool, keep track of time last *unpinned*

2)Replace the frame that has the oldest (earliest) time

- Most common policy: intuitive and simple
  - Based on notion of “Temporal Locality”
  - Works well to keep “working set” in buffers.
- Implemented through doubly linked list of BCBs
  - Requires list manipulation on unpin

"Clock" Replacement Policy



- An approximation of LRU
- Arrange frames into a cycle, store one *reference bit per frame*
  - Can think of this as the 2nd chance bit
- When pin count reduces to 0, turn on ref. bit
- When replacement necessary

```
do for each page in cycle {
        if (pincount == 0 && ref bit is on)
                turn off ref bit;
        else if (pincount == 0 && ref bit is off)
                choose this page for
                        replacement;
} until a page is chosen;
```

Some issues with LRU

- *Problem: Sequential flooding*
  - LRU + repeated sequential scans.
  - # buffer frames < # pages in file means each page request causes an I/O. *MRU* much better in this situation (but not in all situations, of course).
- *Problem: "cold" pages can hang around a long time before they are replaced.*

DBMS vs. OS File System

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

- Some limitations, e.g., files can't span disks.
  - Note, this is changing --- OS File systems are getting smarter (i.e., more like databases!)

- Buffer management in DBMS requires ability to:
  - pin a page in buffer pool, force a page to disk & order writes (important for implementing CC & recovery)
  - adjust *replacement policy,* and pre-fetch pages based on access patterns in typical DB operations.