

CS460: Intro to Database Systems

# Class 11: The Storage Layer

Instructor: Manos Athanassoulis

<https://bu-disc.github.io/CS460/>

# The Storage Layer

## DBMS layers and storage hierarchy

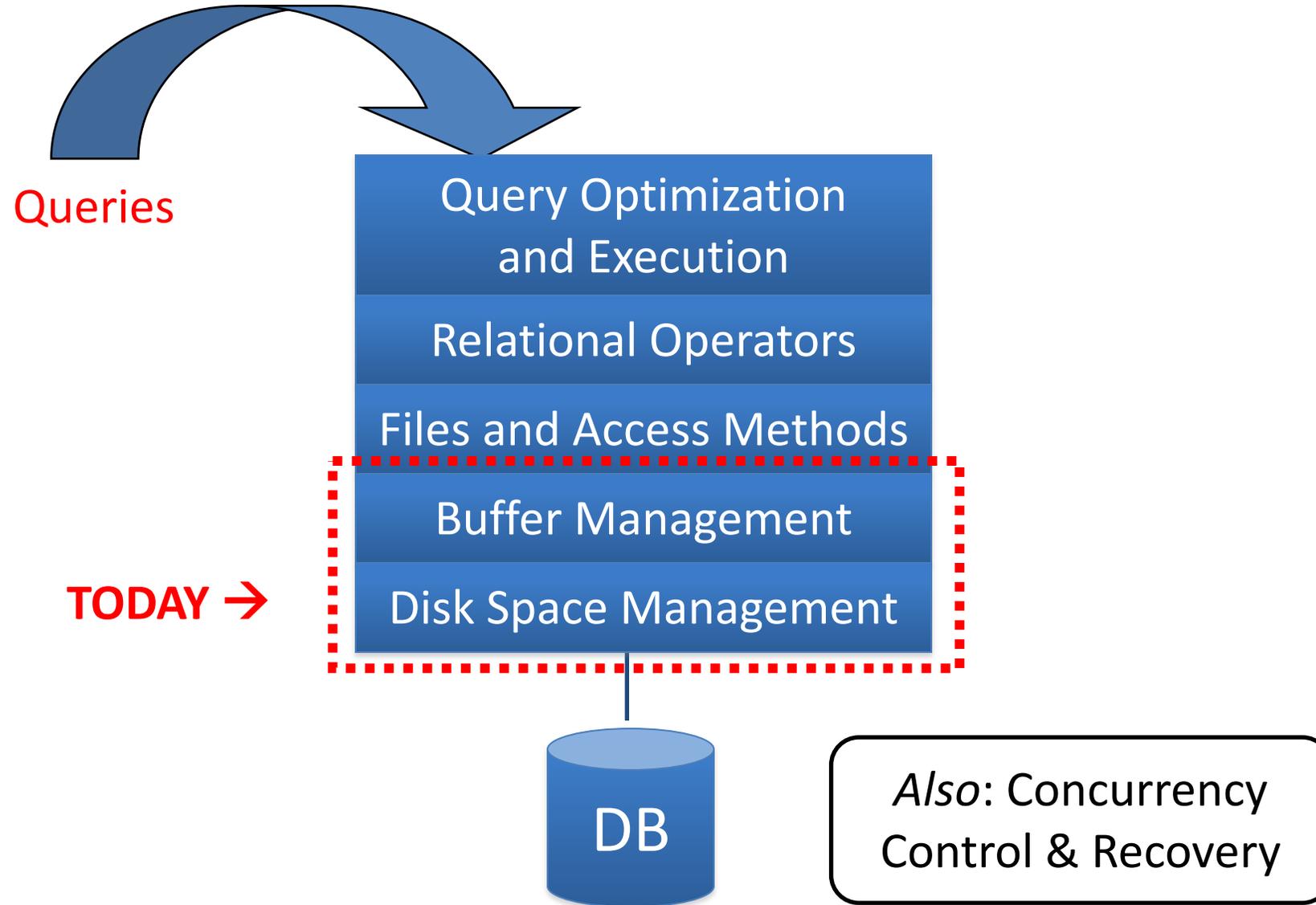
Readings: Chapter 9.1

Disks

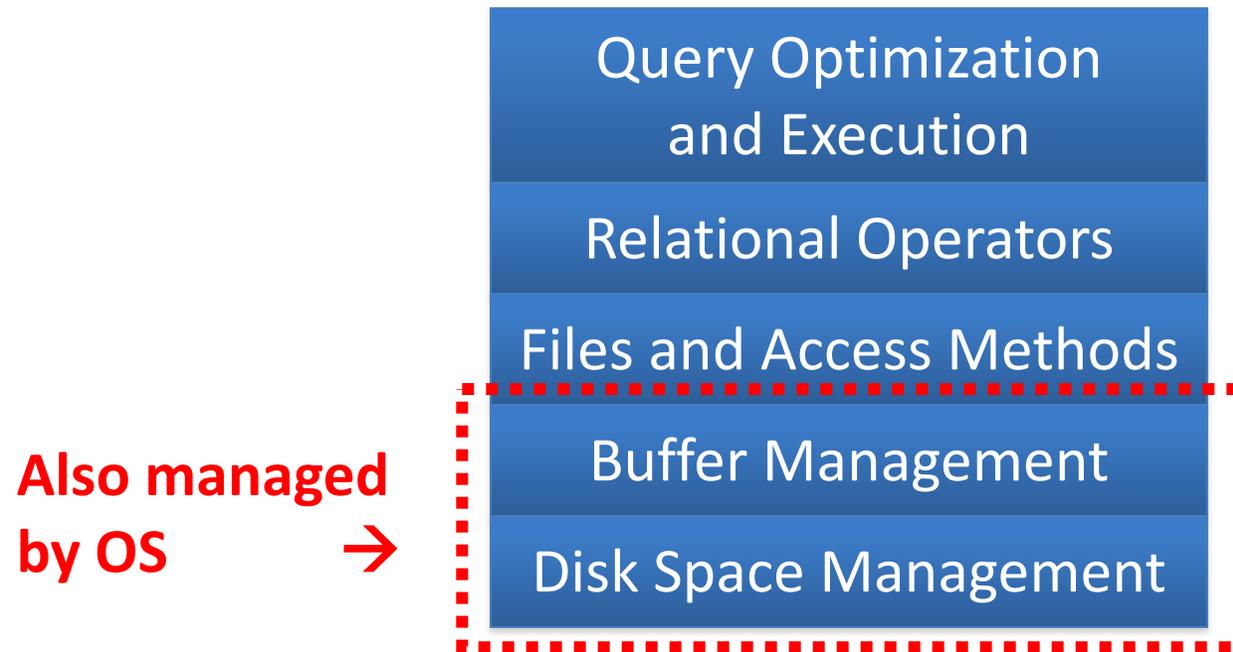
Flash disks

Buffer Management

# DBMS Layer-Cake



# DBMS Layer-Cake



# Why not OS?

Layers of abstraction are good ... but:

Unfortunately, OS often **gets in the way** of DBMS

DBMS needs to do things “its own way”

**Specialized prefetching**

**Control over buffer replacement policy**

LRU not always best (sometimes worst!!)

**Control over thread/process scheduling**

“Convoy problem” arises when OS scheduling conflicts with DBMS locking

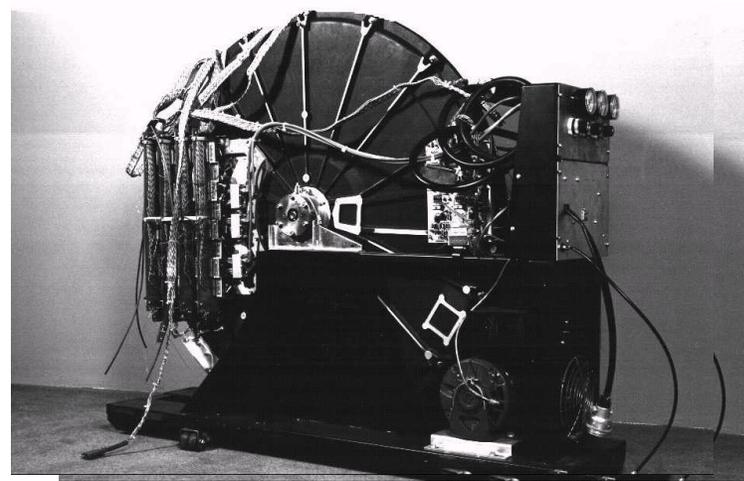
**Control over flushing data to disk**

WAL protocol requires flushing log entries to disk

# Disks and Files

## DBMS stores information

In an electronic world, disks are a mechanical anachronism!



on disks.

## This has major implications for DBMS design!

**READ:** transfer data from disk to main memory (RAM).

**WRITE:** transfer data from RAM to disk.

Both are high-cost operations, relative to in-memory operations, so must be planned carefully!



# Why Not Store It All in Main Memory?

## *Costs too high*

High-end Databases today in the Petabyte range.

~ 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!)

## *But, main-memory database systems do exist!*

Smaller size, performance optimized

Volatility is ok for some applications

# What about Flash?

Flash chips used for >20 years

Flash evolved

USB keys



Storage in mobile devices



Consumer and enterprise flash disks (SSD)

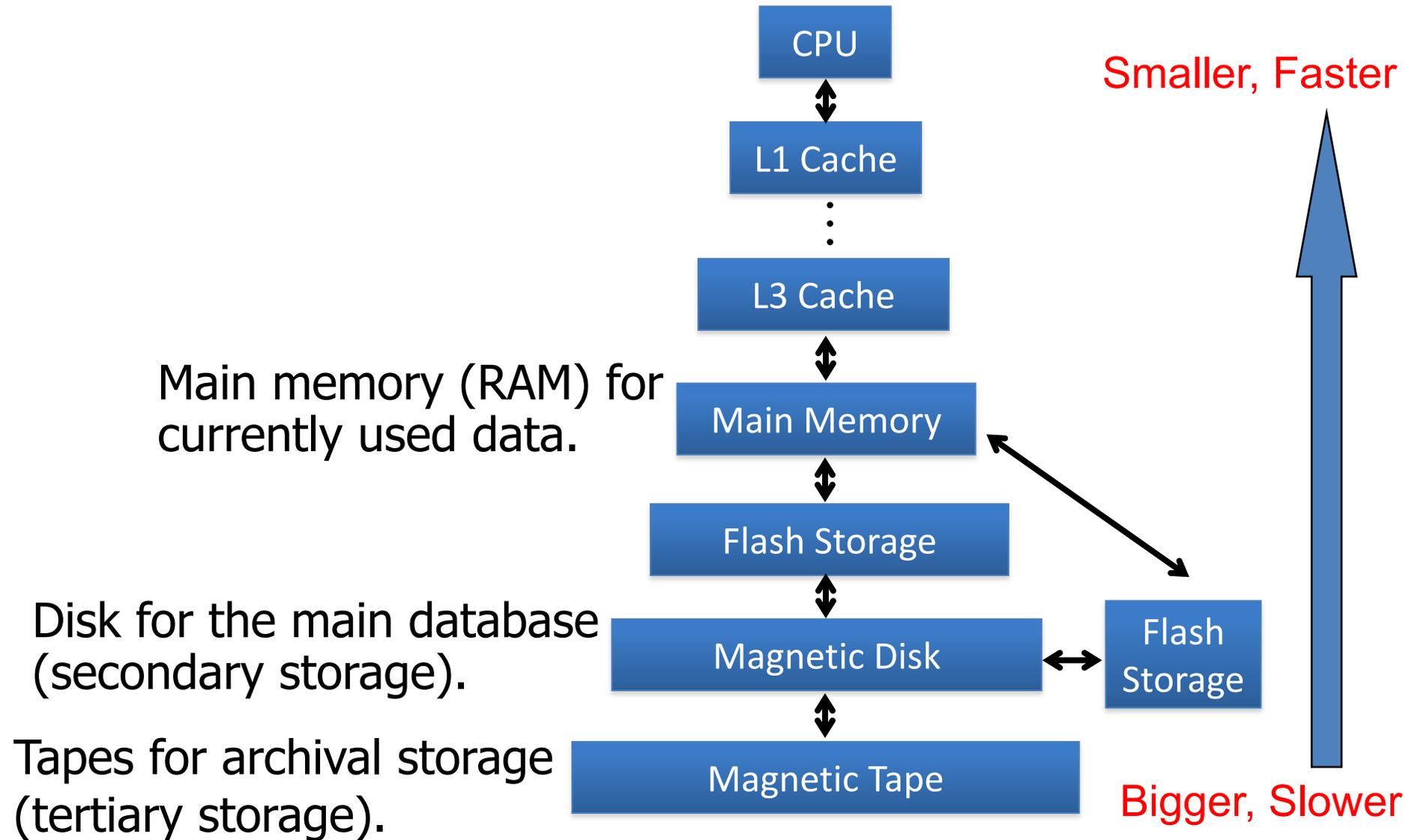


Flash in a DBMS

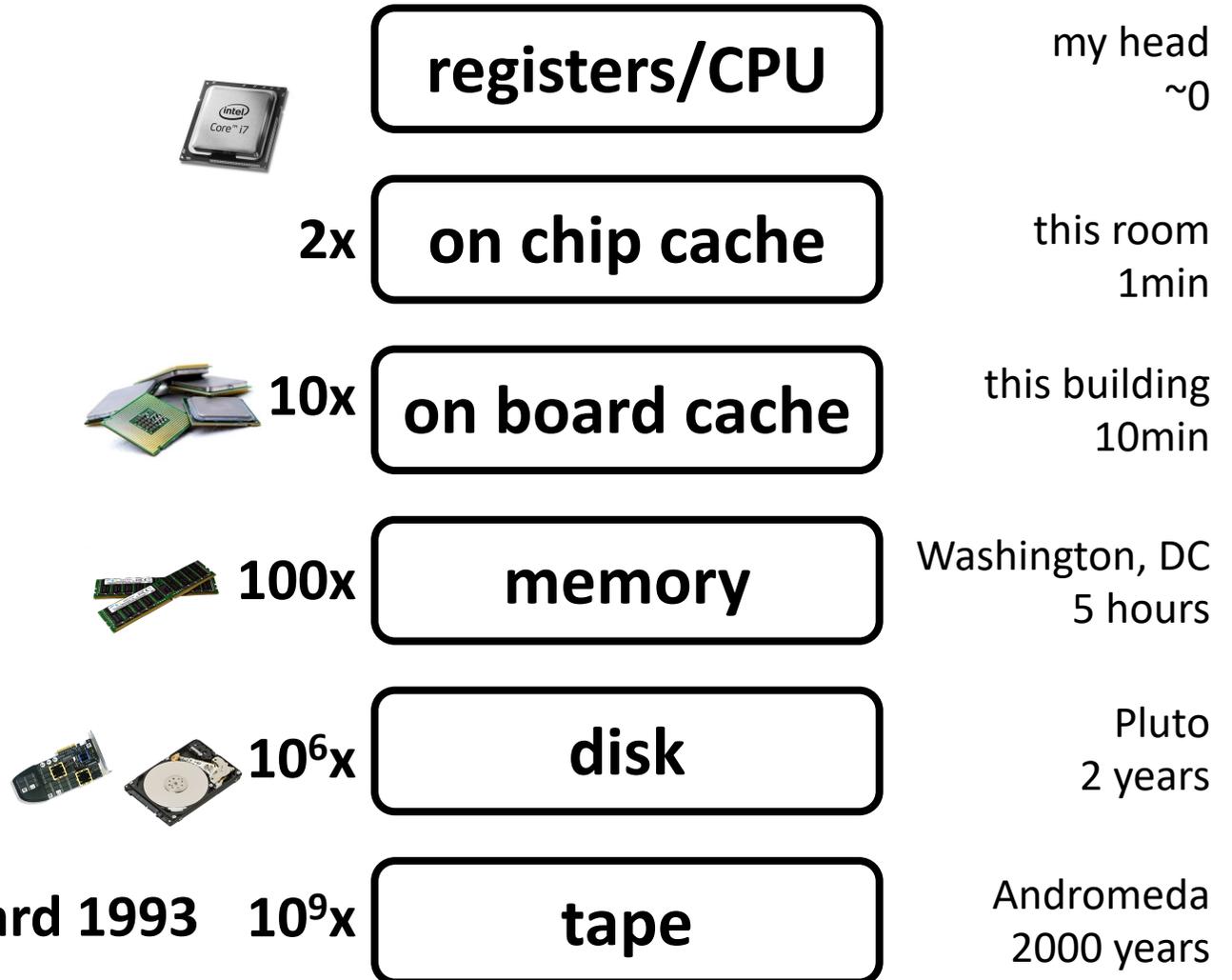
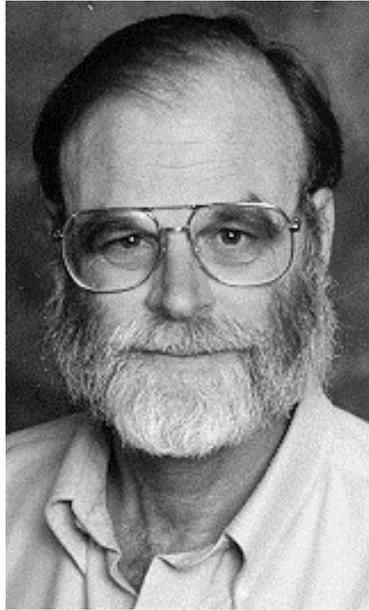
Main storage

Accelerator/enabler (Specialized cache, logging device)

# The Storage Hierarchy



# memory hierarchy (by Jim Gray)

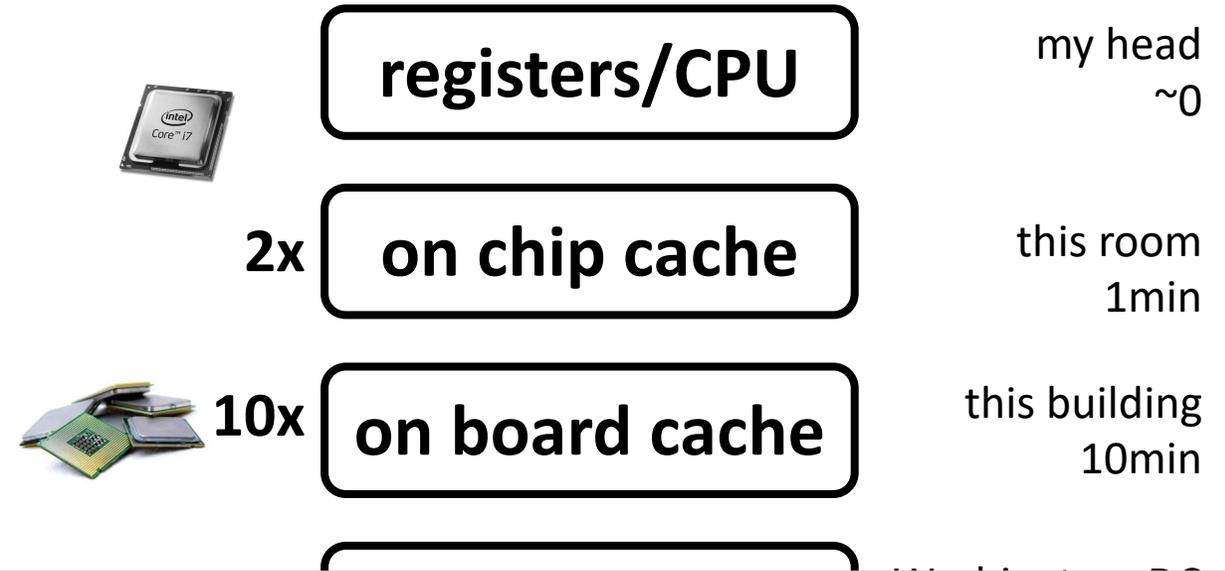
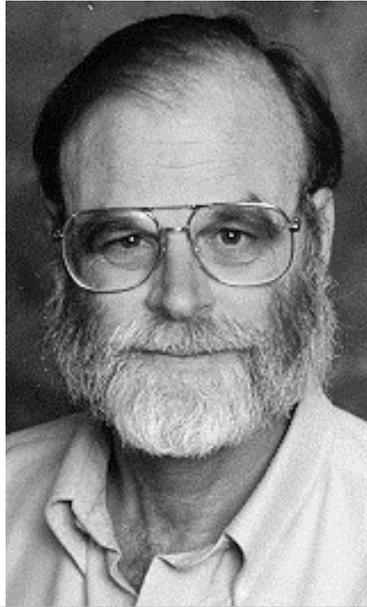


Jim Gray, IBM, Tandem, Microsoft, DEC  
 “The Fourth Paradigm” is based on his vision

**ACM Turing Award 1998**

**ACM SIGMOD Edgar F. Codd Innovations award 1993**

# memory hierarchy (by Jim Gray)



tape?

sequential-only magnetic storage  
still a multi-billion industry

# The Storage Layer

DBMS layers and storage hierarchy

## Disks

Readings: Chapter 9.1, 9.2, HDD paper

Flash disks

Buffer Management

# Disks

Secondary storage device of choice.

Main advantage over tapes: random access vs. *sequential*.

Data is stored and retrieved in units called *disk blocks* or *pages*.

Unlike RAM, time to retrieve a disk page varies depending upon location on disk.

Therefore, relative placement of pages on disk has major impact on DBMS performance!

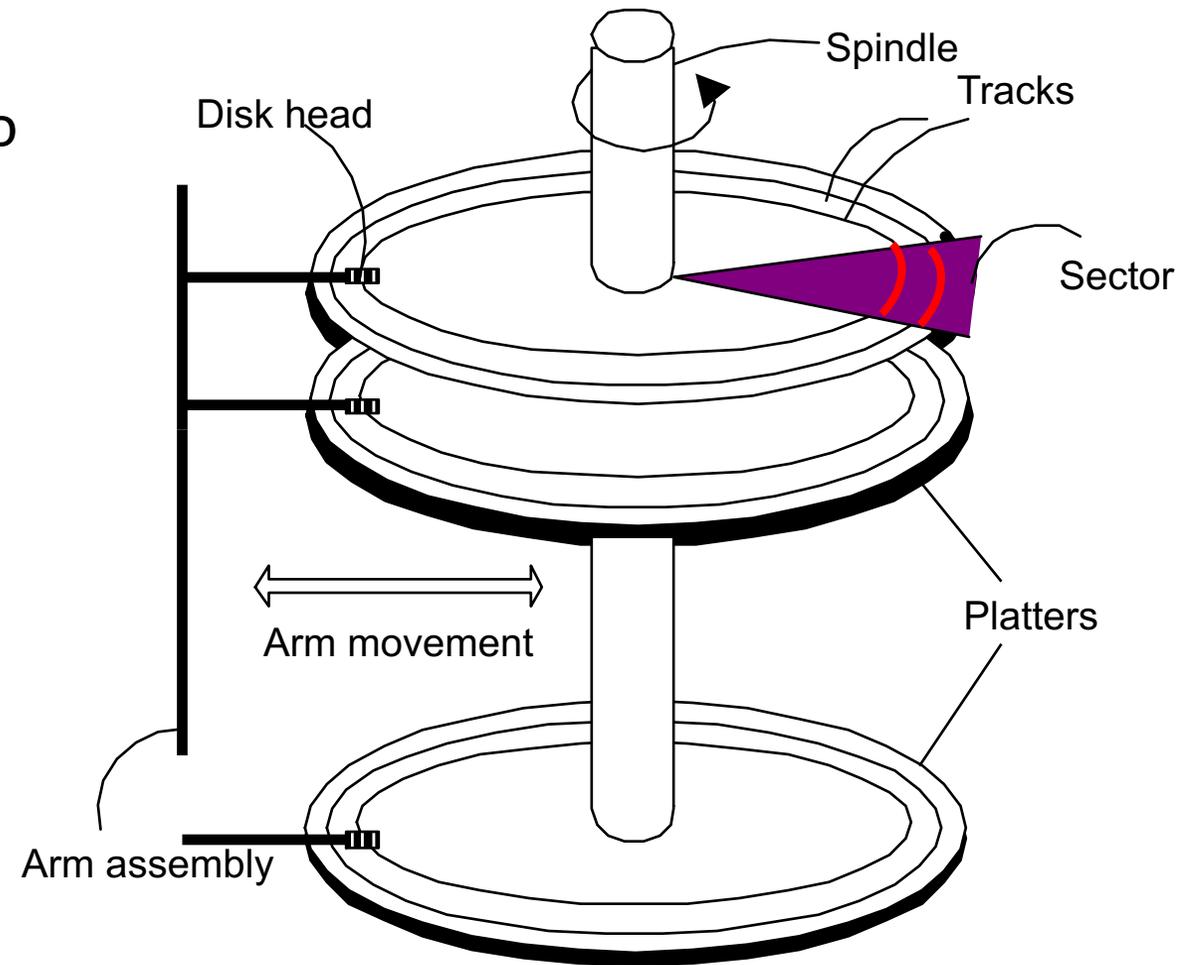
# Anatomy of a Disk

The platters spin (5-15 kRPM).

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).
- ❖ Newer disks have several “zones”, with more data on outer tracks.



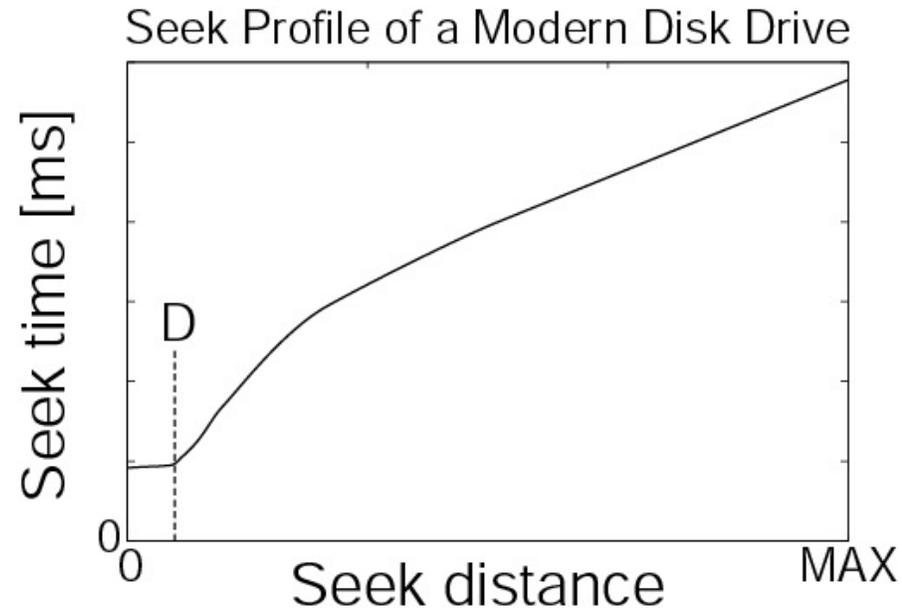
# Accessing a Disk Page

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)

# Seeking in modern disks

## Seek time discontinuity

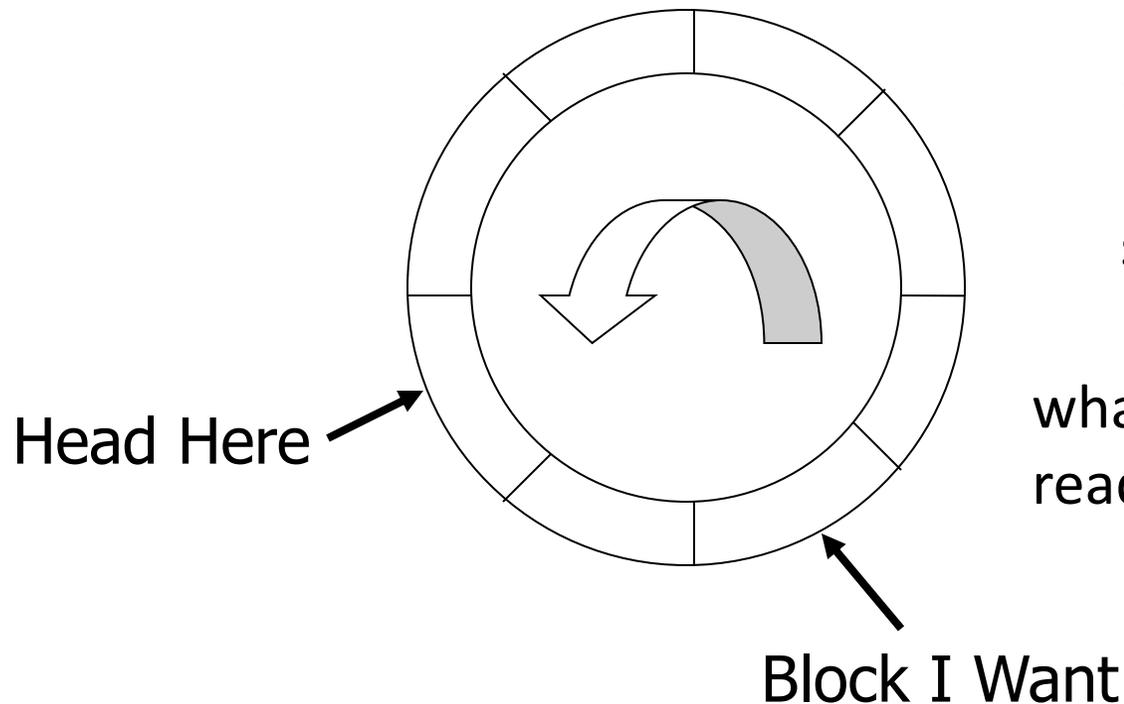


## Short seeks are dominated by “settle time”

- Move to one of **many nearby tracks** within settle time
- D is on the order of tens to hundreds
- D gets larger with increase of disk track density

# Rotational Delay

if the disk rotates with 10 KRPM, and I want to read 2/3 of the track away what is the rotational delay?



$$(1/10000) * 60 =$$

$$10^{-4} * 60 = 6 * 10^{-3} = 6\text{ms}$$

$$\text{so, } 2/3 * 6\text{ms} = 4\text{ms}$$

what if I am constantly reading 4KB pages like that?



$$4\text{KB}/4\text{ms} = 1\text{MB/s}$$

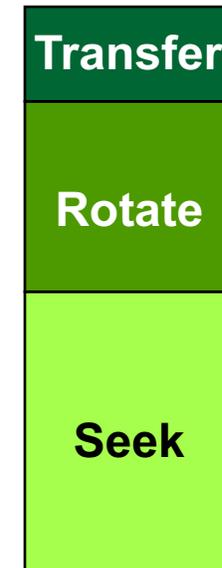
# Seek time & rotational delay dominate

- Seek time varies from about 1 to 20 ms
- Rotational delay varies from 0 to 10 ms
- Transfer rate is  $< 1\text{ms}$  per 4KB page

Key to lower I/O cost:

**reduce seek/rotation delays!**

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

Blocks in a file should be arranged sequentially on disk (by “next”), to minimize seek and rotational delay.

An important optimization: pre-fetching

- See R&G page 323

# Rules of thumb...

1. Memory access much faster than disk I/O (~ 1000x)
2. “Sequential” I/O faster than “random” I/O (~ 10x)

# Disk Space Management

Lowest layer of DBMS software manages space on disk

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.

# The Storage Layer

DBMS layers and storage hierarchy

Disks

Flash disks

SSD paper

Buffer Management

# Flash disks

Secondary storage *or* caching layer.

Main advantage over disks:

*random reads* as fast as *sequential* reads

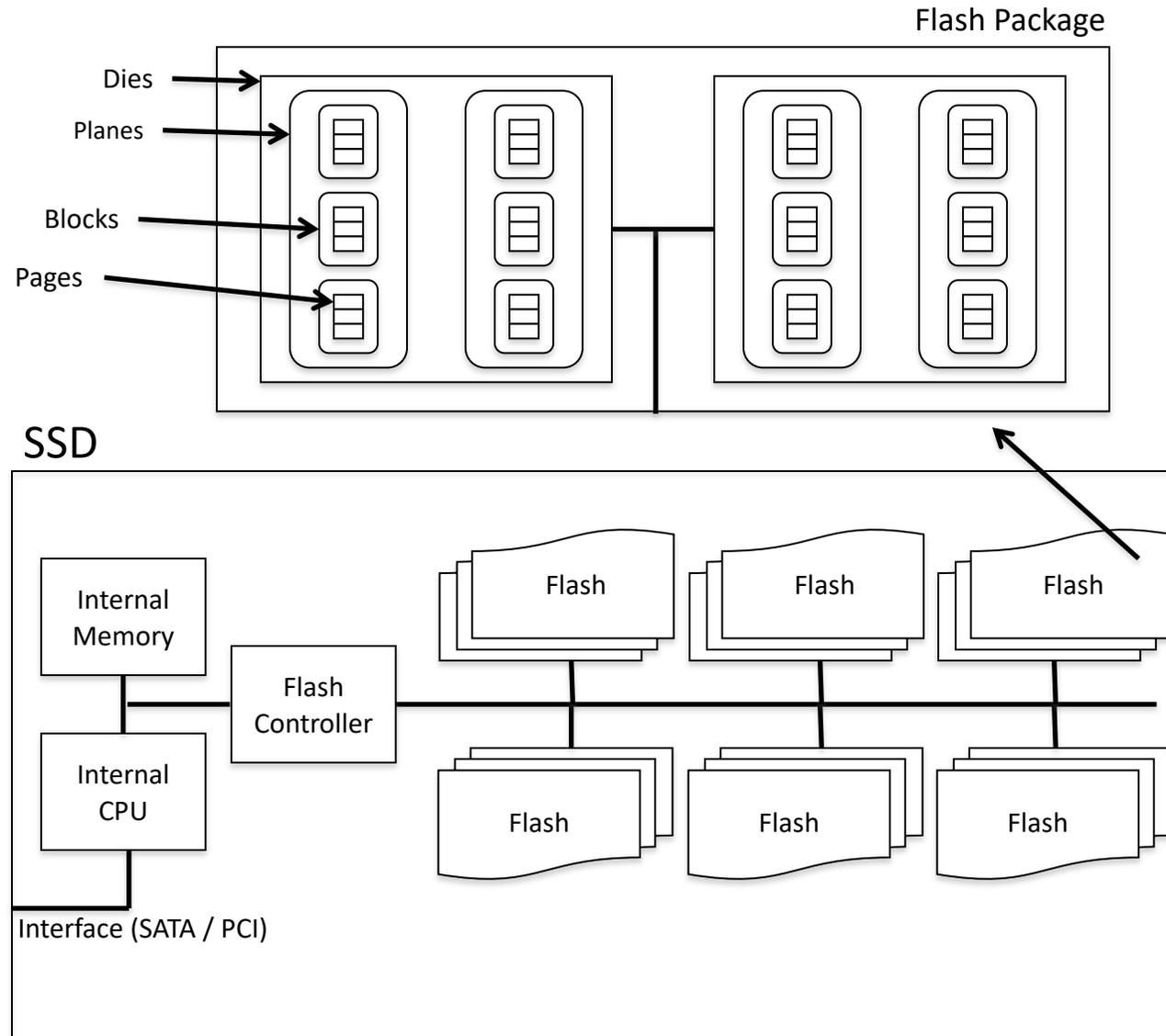
***BUT***: slow *random writes* (slower than reads)

*pages* (like disks) and pages organized in *flash blocks*

*unlike HDD, like RAM:*

time to retrieve a page is not related to location on flash disk.

# The internals of flash disks



Interconnected flash chips

No mechanical limitations

Maintain the block API – compatible with disks layout

Internal parallelism in read/write

Complex software driver

# Accessing a flash page

## Access time depends on

- Device organization (**internal parallelism**)
- Software efficiency (**driver**)
- Bandwidth of flash packages (**bus speed**)

## Flash Translation Layer (FTL)

- Complex device driver (**firmware**)
- Tunes performance and device lifetime

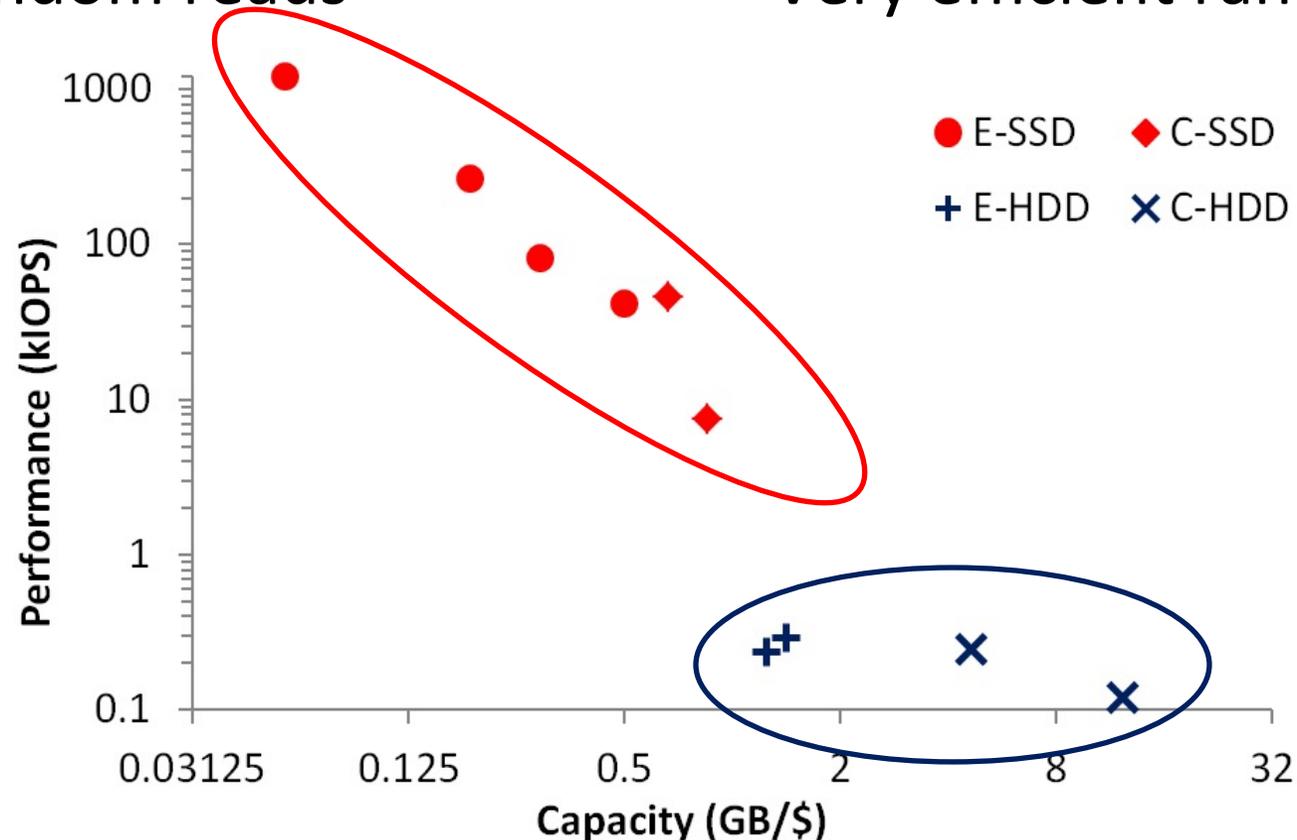
# Flash disks vs HDD

## HDD

- ✓ Large – inexpensive capacity
- x Inefficient random reads

## Flash disks

- x Small – expensive capacity
- ✓ Very efficient random reads



# The Storage Layer

DBMS layers and storage hierarchy

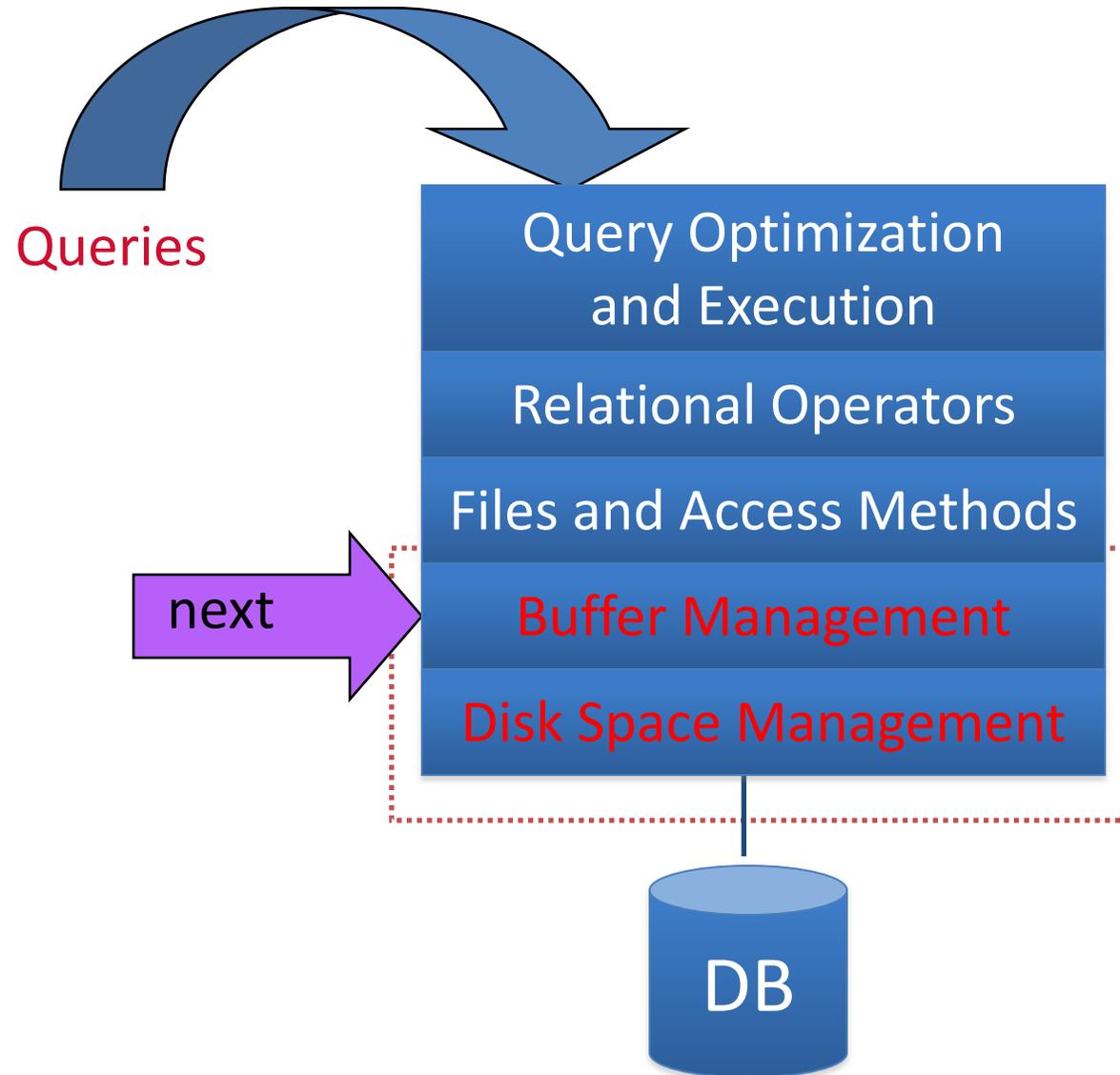
Disks

Flash disks

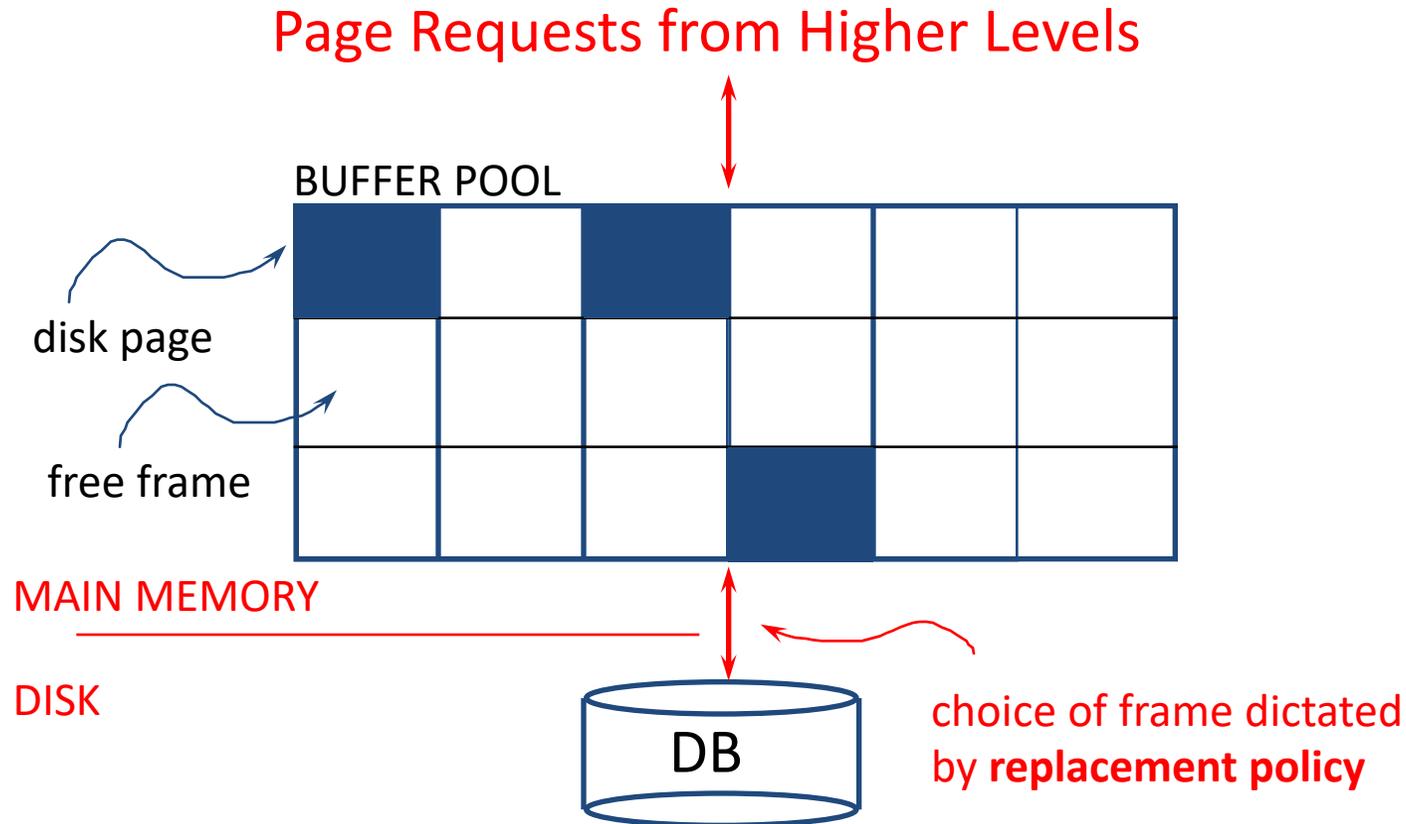
**Buffer Management**

Readings: Chapter 9.3, 9.4

# Recall the BIG Picture



# Buffer Management in a DBMS



*Data must be in RAM for DBMS to operate on it!*

Buffer Manager hides the fact that not all data is in RAM

*(just like hardware cache policies hide the fact that not all data is in the caches)*

# When a Page is Requested ...

Buffer pool information table contains:

*<frame#, pageid, pin\_count, dirty>*  
has the page been updated  
how many queries *still* need the page

If requested page is not in pool & buffer pool is full:

- Choose a frame for *replacement (only un-pinned pages are candidates)*
- If frame is “dirty”, write it to disk
- Read requested page into chosen frame

*Pin* the page and return its address.

➡ *If requests can be predicted (e.g., sequential scans)  
pages can be pre-fetched several pages at a time!*

# More on Buffer Management

Requestor of page must unpin it, and indicate whether page has been modified:

- *dirty* bit is used for this.

Page in pool may be requested many times,

- a *pin count* is used. A page is a candidate for replacement iff *pin count* = 0 (“unpinned”)

CC & recovery may entail additional I/O when a frame is chosen for replacement. (*Write-Ahead Log* protocol; more later.)

# Buffer Replacement Policy

Frame is chosen for replacement by a *replacement policy*:

- Least-recently-used (LRU), MRU, Clock, etc.

Policy can have big impact on # of I/O's;  
depends on the *access pattern*.

# LRU Replacement Policy

## Least Recently Used (LRU)

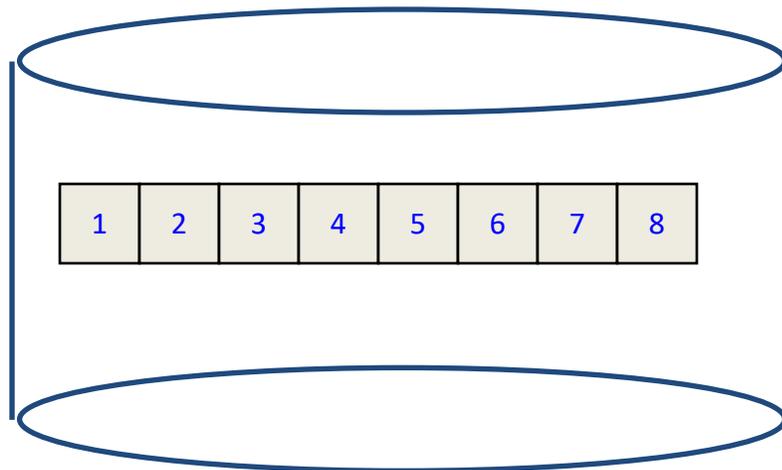
- for each page in buffer pool, keep track of time last *unpinned*
- replace the frame which has the oldest (earliest) time
- very common policy: intuitive and simple

## Problems?

### Problem: Sequential flooding

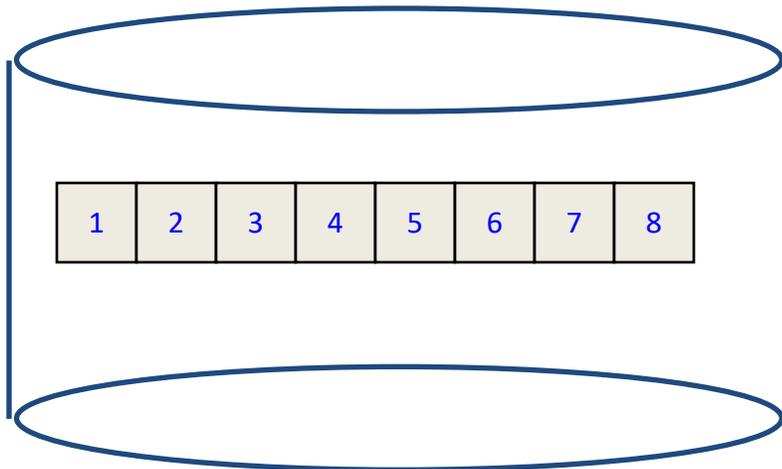
- LRU + repeated sequential scans.
- *# buffer frames < # pages in file* means each page request causes an I/O.  
MRU much better in this situation (but not in all situations, of course).

# Sequential Flooding – Illustration



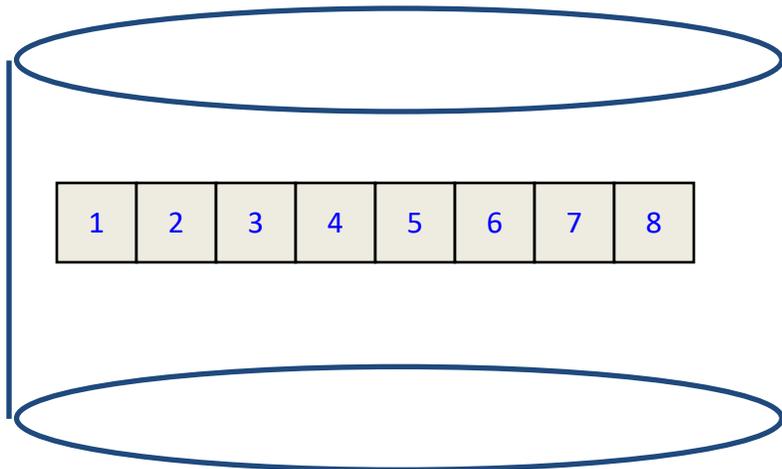
Repeated scan of file ...

# Sequential Flooding – Illustration



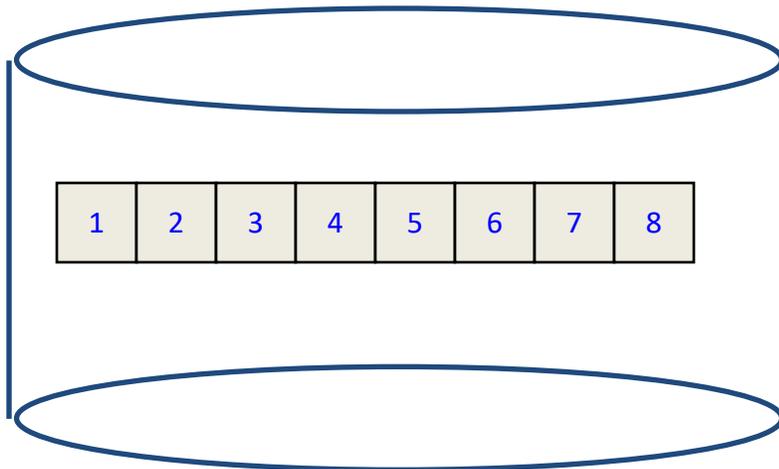
Repeated scan of file ...

# Sequential Flooding – Illustration



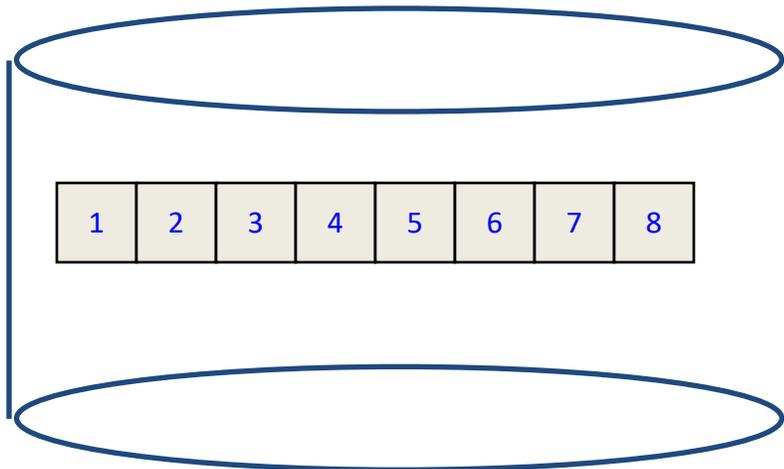
Repeated scan of file ...

# Sequential Flooding – Illustration



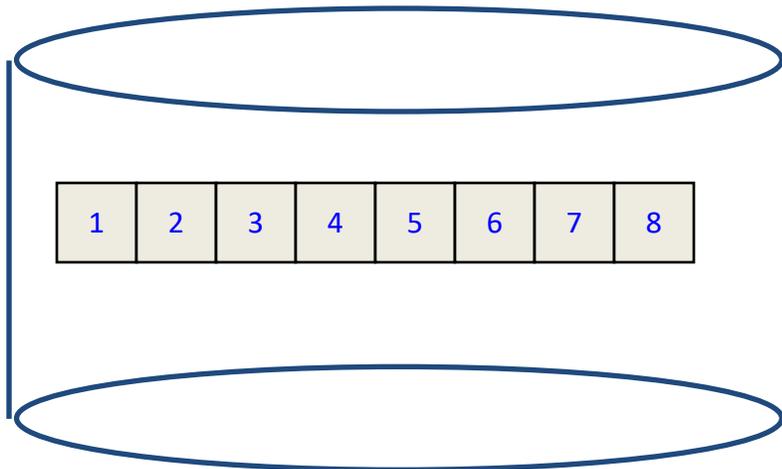
Repeated scan of file ...

# Sequential Flooding – Illustration



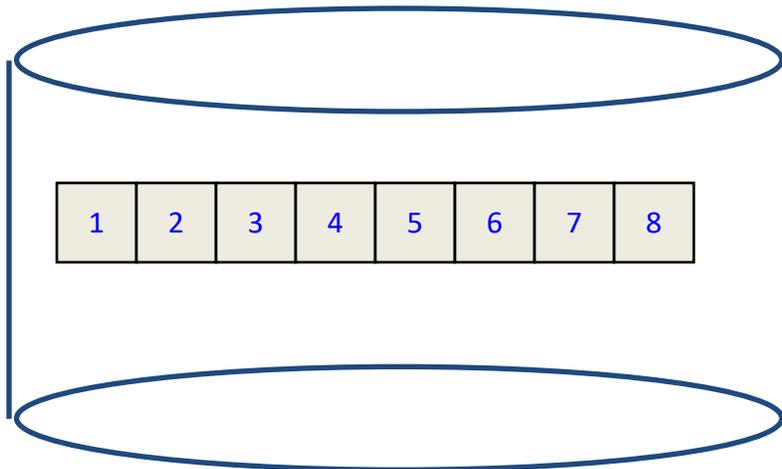
Repeated scan of file ...

# Sequential Flooding – Illustration



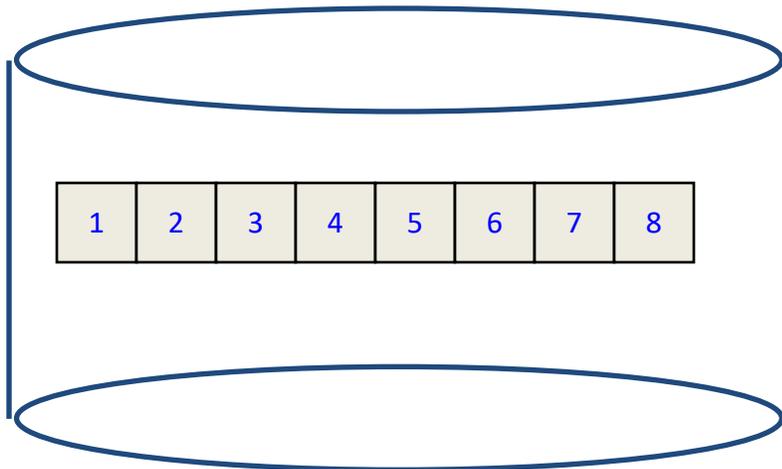
Repeated scan of file ...

# Sequential Flooding – Illustration



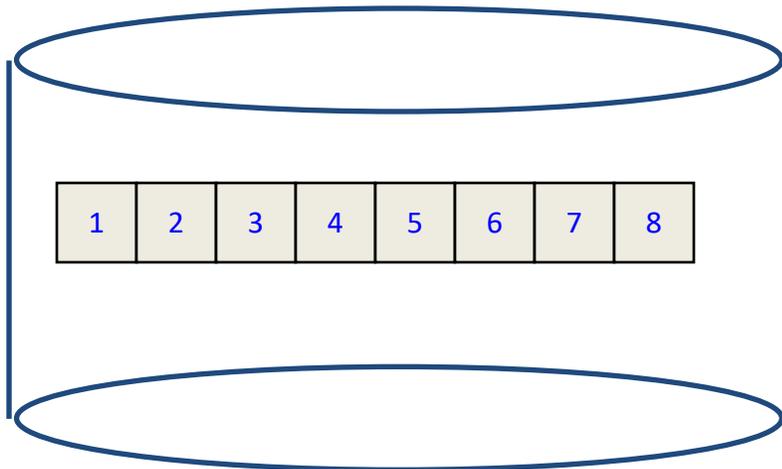
Repeated scan of file ...

# Sequential Flooding – Illustration



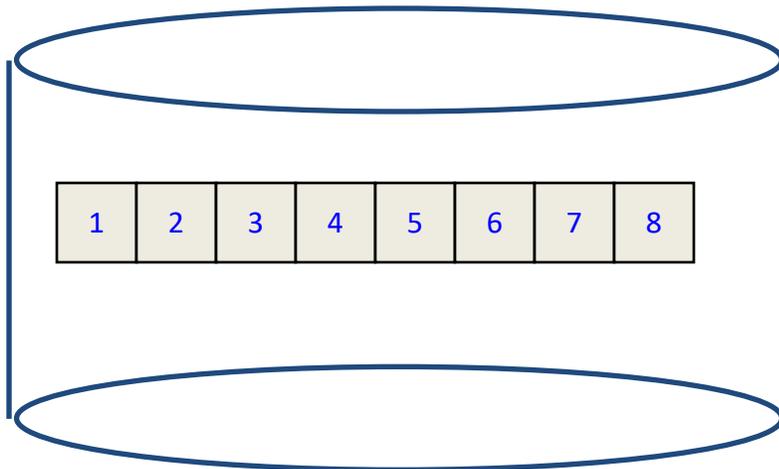
Repeated scan of file ...

# Sequential Flooding – Illustration



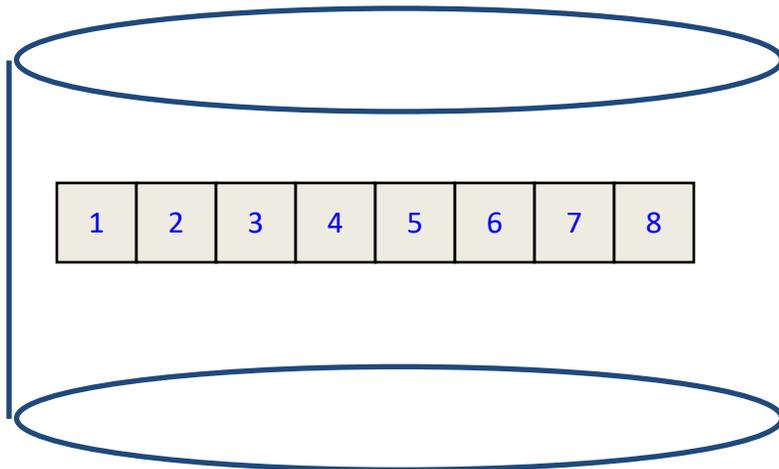
Repeated scan of file ...

# Sequential Flooding – Illustration



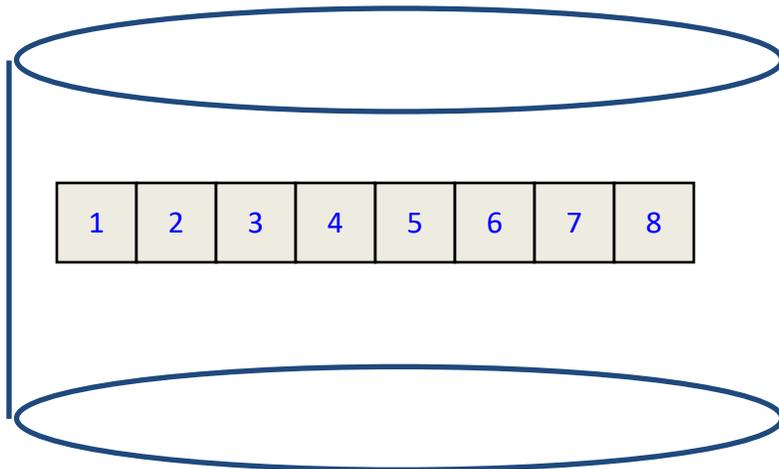
Repeated scan of file ...

# Sequential Flooding – Illustration



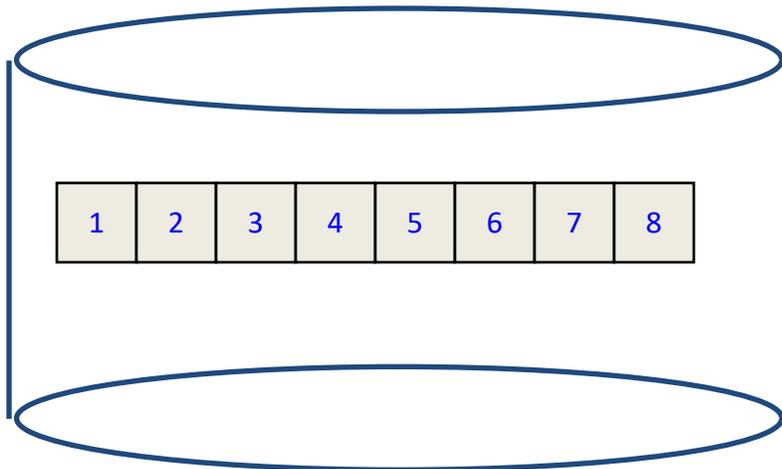
Repeated scan of file ...

# Sequential Flooding – Illustration



Repeated scan of file ...

# Sequential Flooding – Illustration

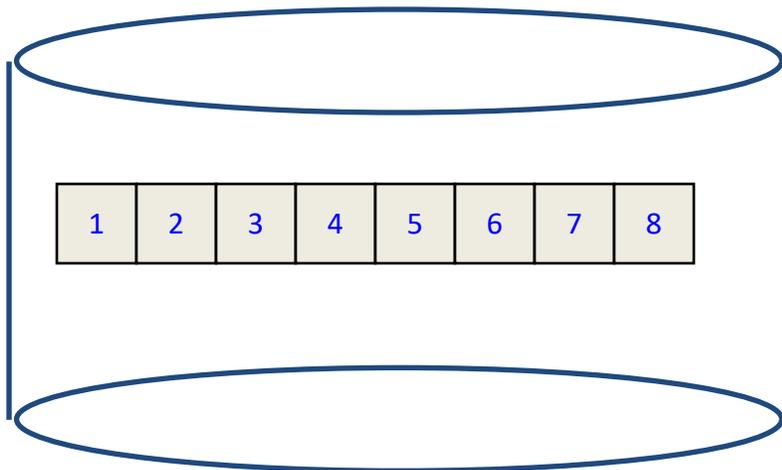


Repeated scan of file ...

# Sequential Flooding – Illustration

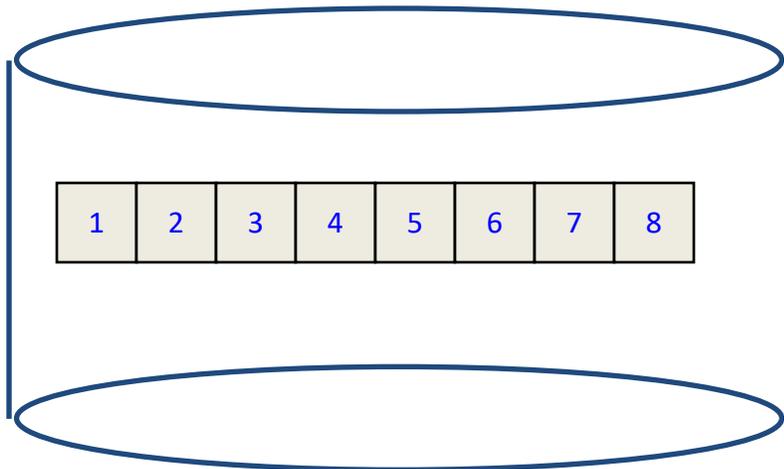


for 2 scans every page access  
was a miss (had to go to disk)  
 $2 * 8 = 16$  disk accesses



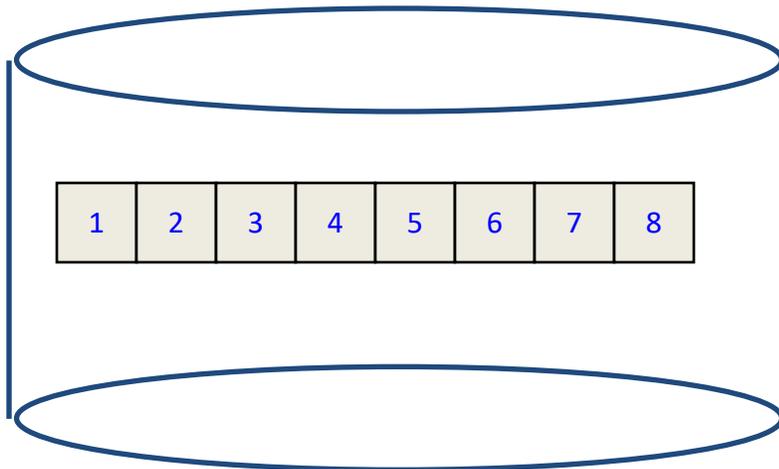
Repeated scan of file ...

# Sequential Flooding – Illustration



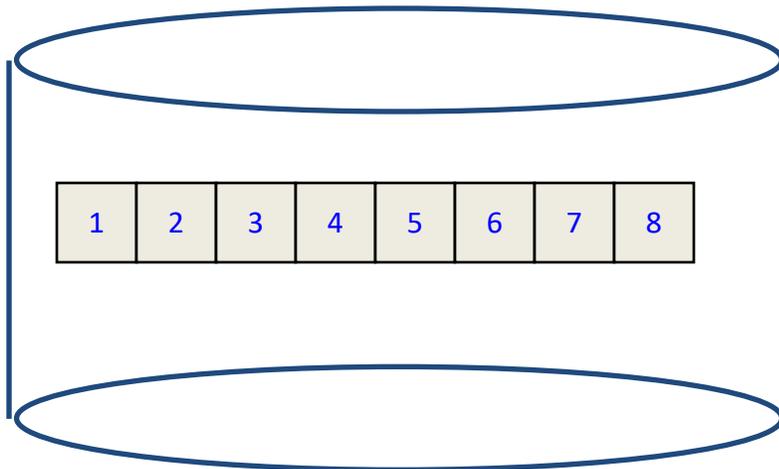
Repeated scan of file ...

# Sequential Flooding – Illustration



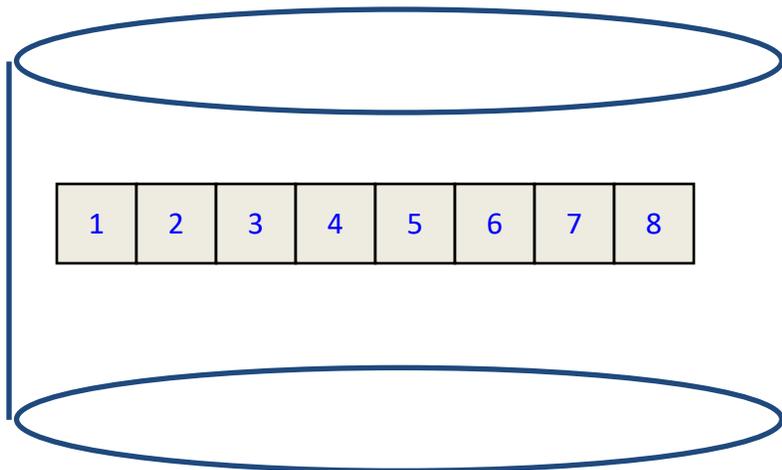
Repeated scan of file ...

# Sequential Flooding – Illustration



Repeated scan of file ...

# Sequential Flooding – Illustration

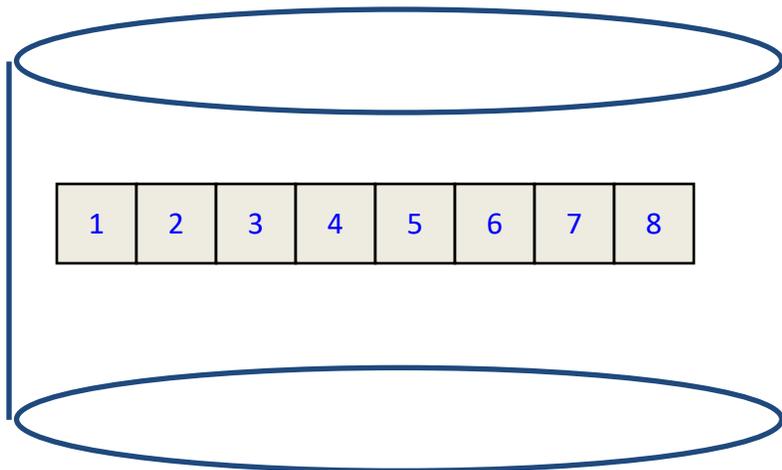


Repeated scan of file ...

# Sequential Flooding – Illustration

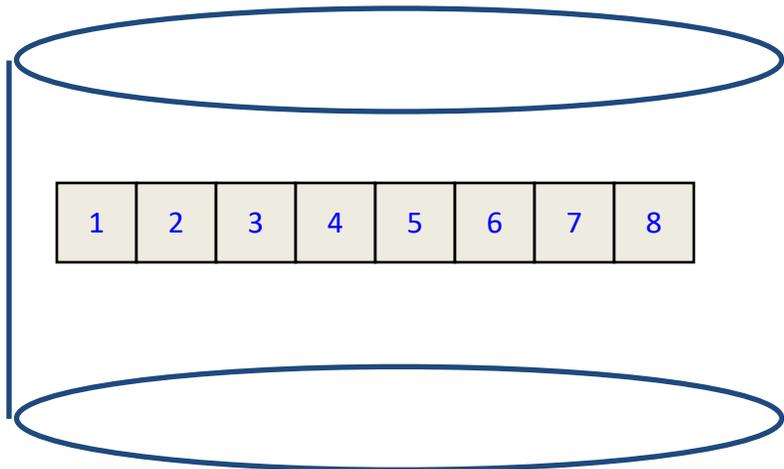


can re-use those!



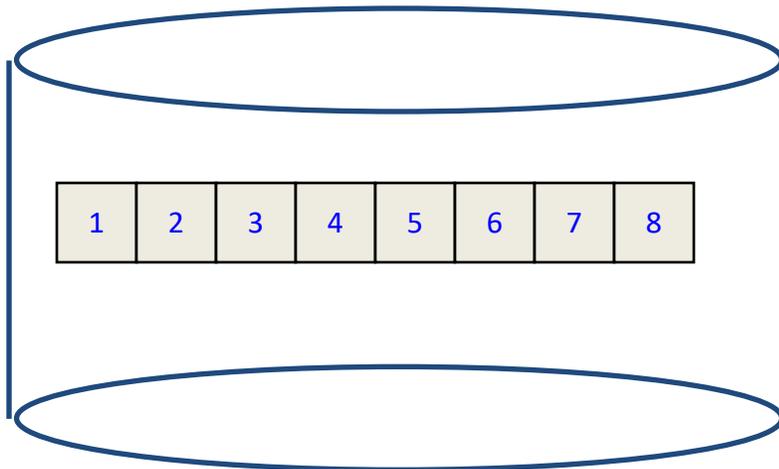
Repeated scan of file ...

# Sequential Flooding – Illustration



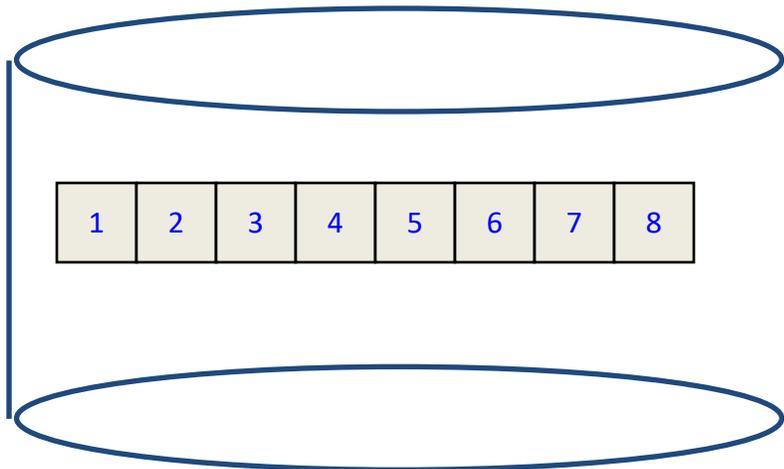
Repeated scan of file ...

# Sequential Flooding – Illustration



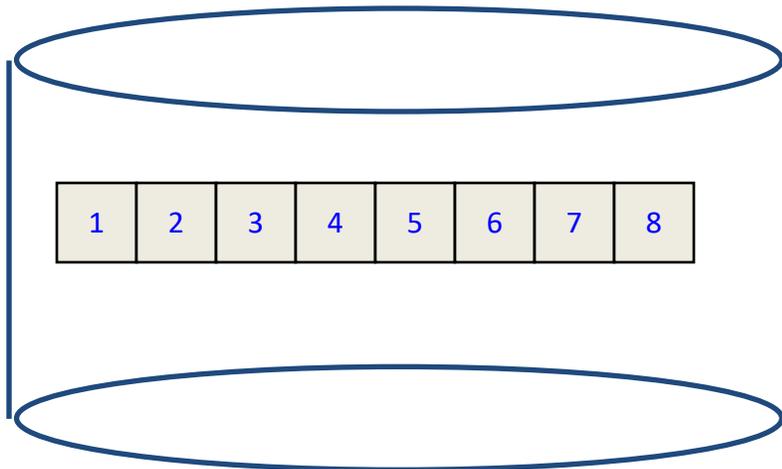
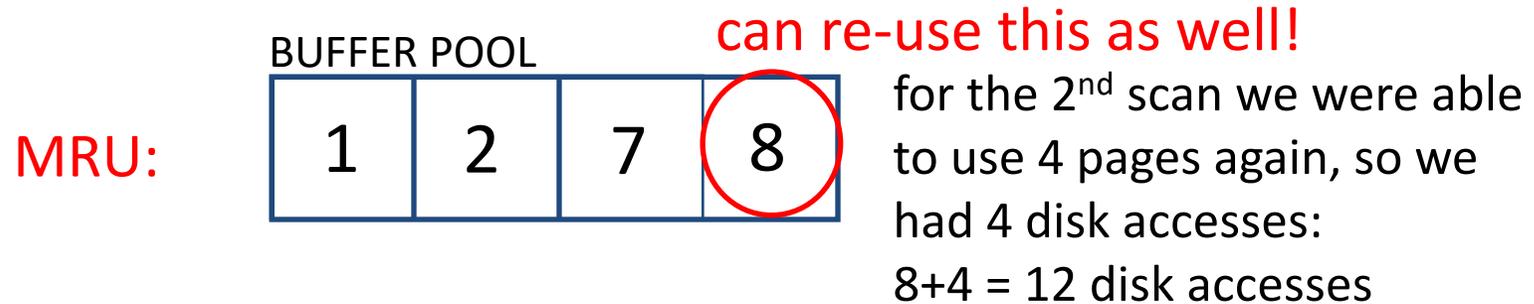
Repeated scan of file ...

# Sequential Flooding – Illustration



Repeated scan of file ...

# Sequential Flooding – Illustration



Repeated scan of file ...

# “Clock” Replacement Policy

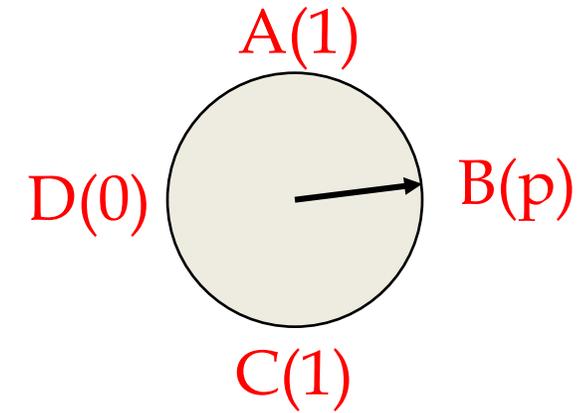
An approximation of LRU.

Arrange frames into a cycle, store  
one “reference bit” per frame

When pin count goes to 0, reference bit set on.

When replacement necessary:

```
do {  
    if (pincount == 0 && ref bit is off)  
        choose current page for replacement;  
    else if (pincount == 0 && ref bit is on)  
        turn off ref bit;  
    advance current frame;  
} until a page is chosen for replacement;
```



# Summary

Disks provide cheap, non-volatile storage.

- Random access, but **cost depends on location** of page on disk; important to arrange data **sequentially** to minimize *seek* and *rotation* delays.

Buffer manager brings pages into RAM.

- Page stays in RAM until released by requestor.
- Written to disk when frame chosen for replacement (which is sometime after requestor releases the page).
- Choice of frame to replace based on ***replacement policy***.
- Good to ***pre-fetch*** several pages at a time.