Storing Data: Disk and Files

CS 564- Spring 2025

ACKs: *Dan Suciu, Jignesh Patel, AnHai Doan*

WHAT IS THIS LECTURE ABOUT?

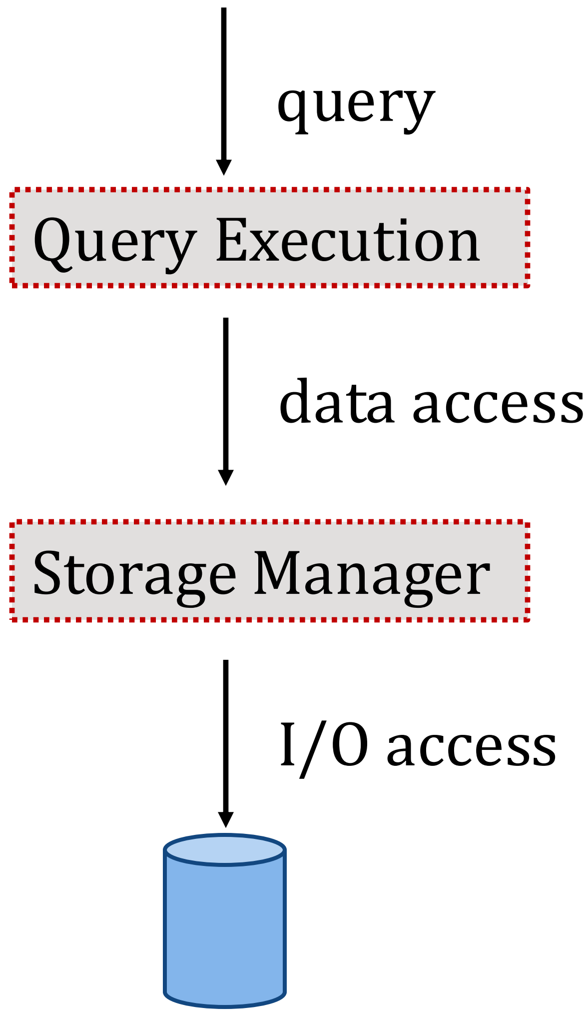
How does a DBMS store data?

- disk, SSD, main memory

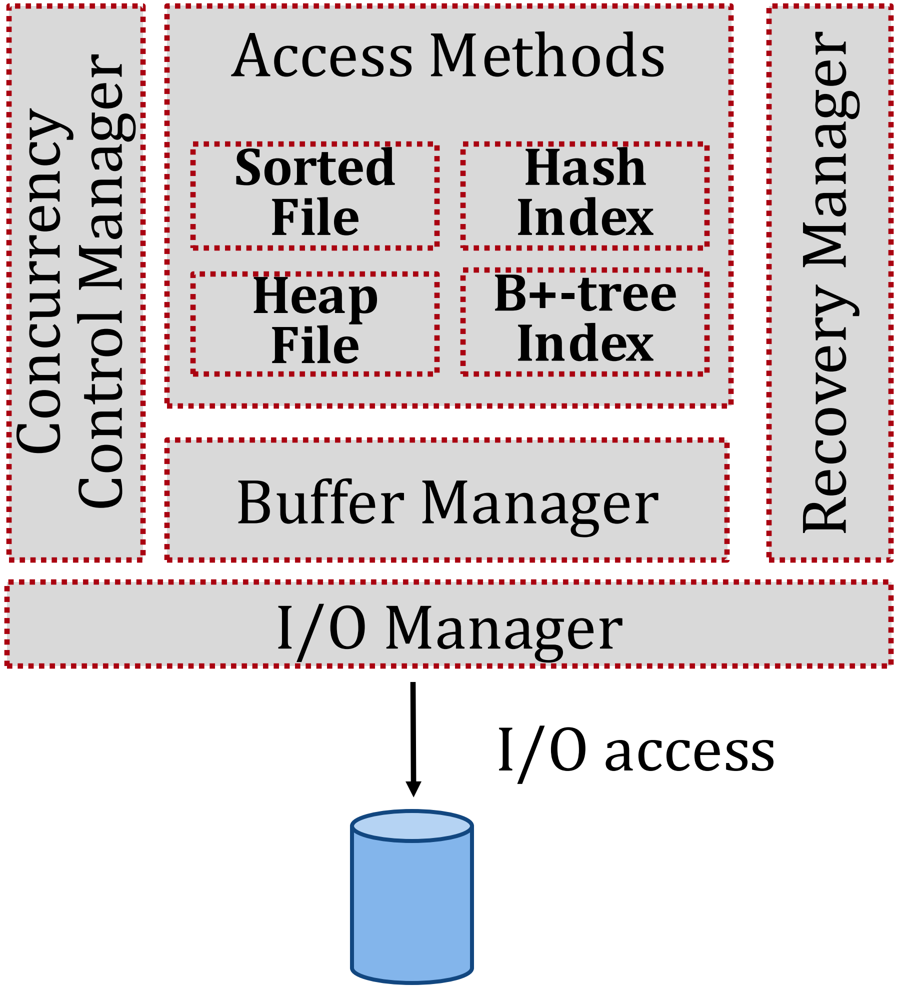
The **buffer manager**

- controls how the data moves between main memory and disk
- uses various replacement policies (LRU, Clock)

ARCHITECTURE OF A DBMS



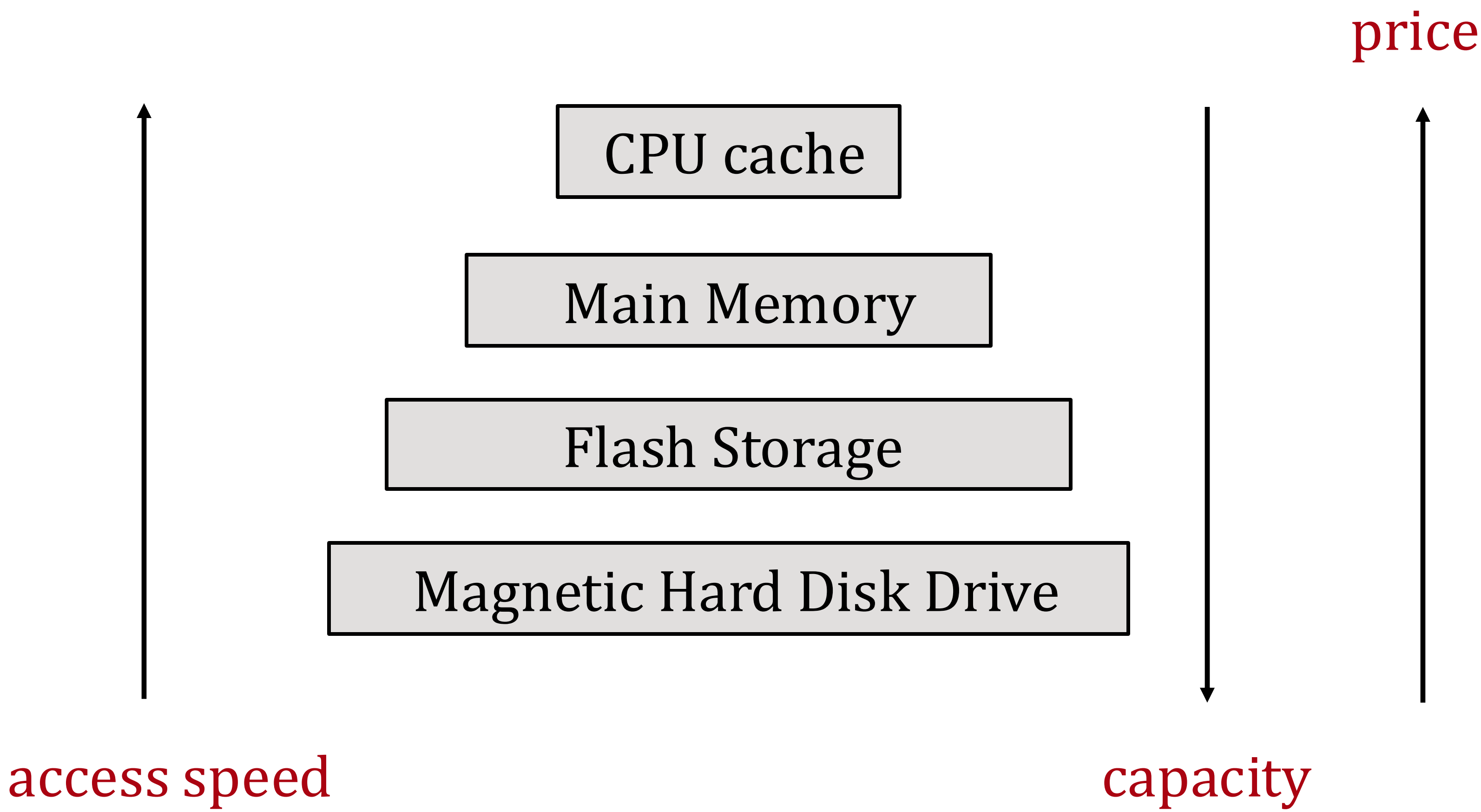
ARCHITECTURE OF STORAGE MANAGER



DATA STORAGE

- How does a DBMS store and access data?
  - main memory (fast, temporary)
  - disk (slow, permanent)
- How do we move data from disk to main memory?
  - buffer manager
- How do we organize relational data into files?
  - next lecture!

MEMORY HIERARCHY



WHY NOT MAIN MEMORY?

- Relatively high cost
- Main memory is not persistent!
- Typical storage hierarchy:
  - **Primary storage**: main memory (RAM) for currently used data
  - **Secondary storage**: disk for the main database
  - **Tertiary storage**: tapes for archiving older versions of the data

DISK

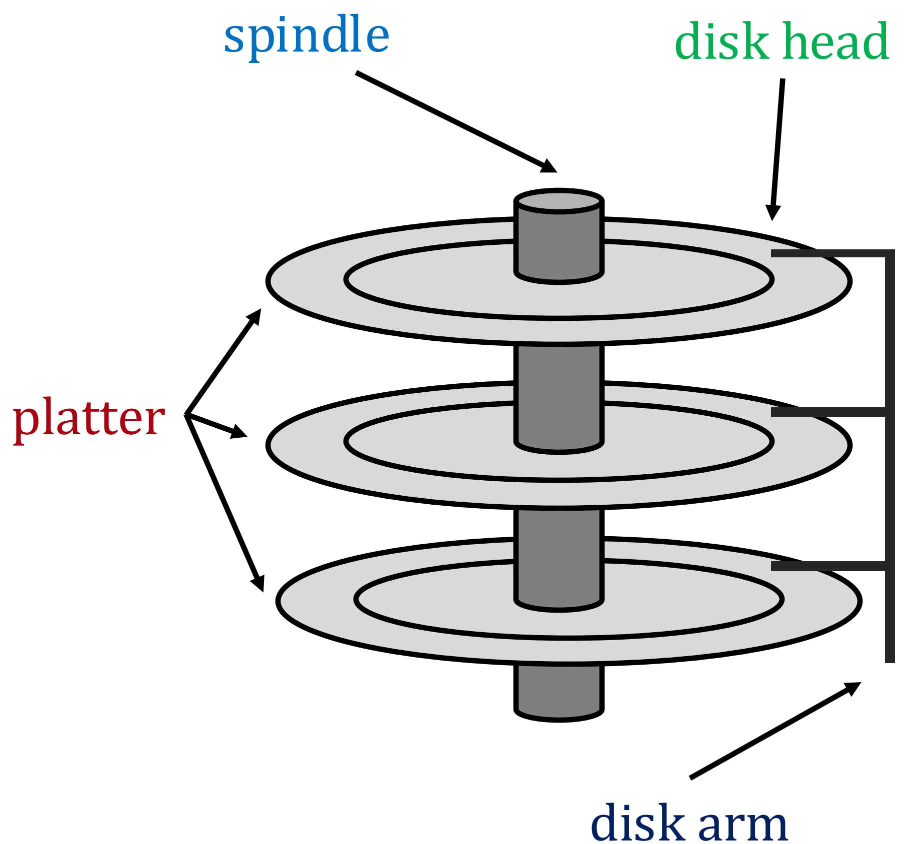
DISKS

- Secondary storage device of choice
- Data is stored and retrieved in units called <u>disk blocks</u>
- The time to retrieve a disk block varies depending upon location on disk (unlike RAM)

The placement of blocks on disk has major impact on DBMS performance!

COMPONENTS OF DISKS

- platter: circular hard surface on which data is stored by inducing magnetic changes
- spindle: axis responsible for rotating the platters
- disk head: mechanism to read or write data
- disk arm: moves to position a head on a desired track of the platter

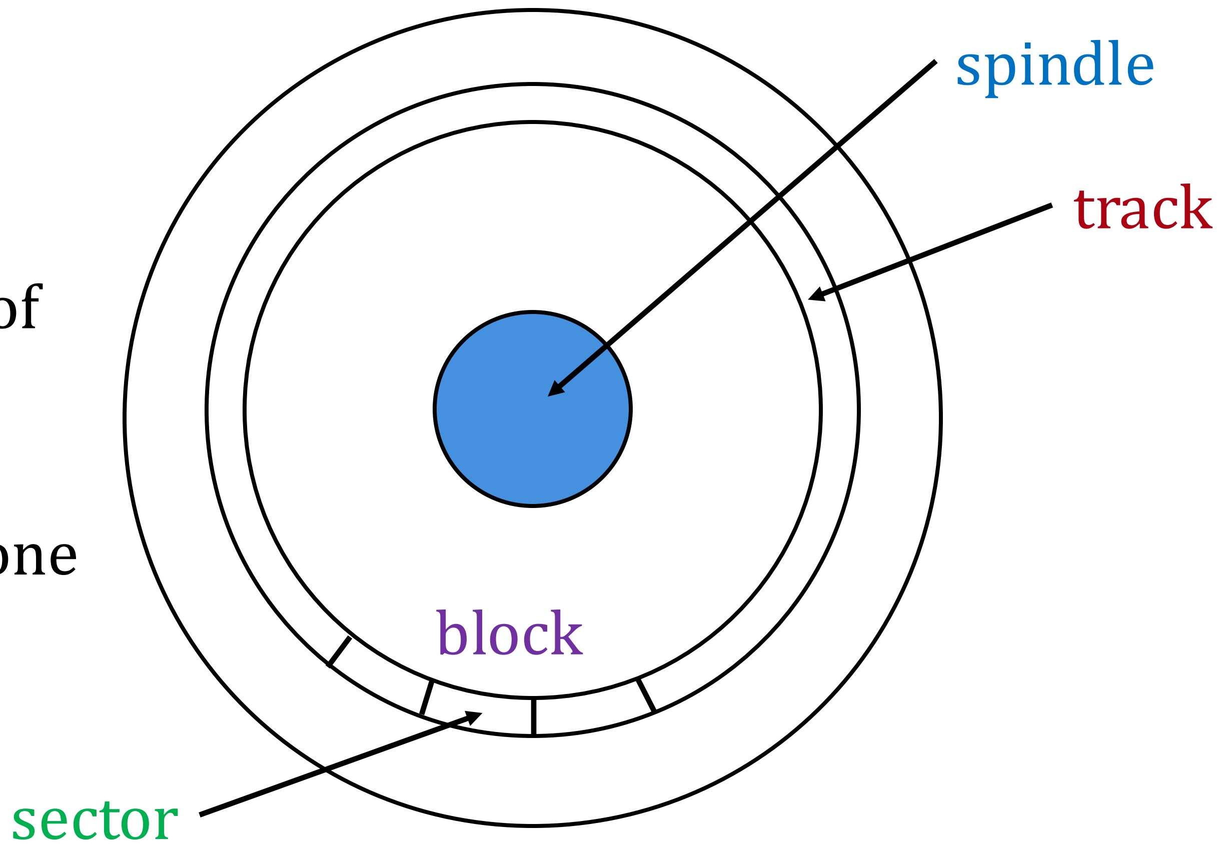


RPM (**R**otations **P**er **M**inute)

- 7200 RPM – 15000 RPM

COMPONENTS OF DISKS

- data is encoded in concentric circles of sectors called tracks
- block size: multiple of sector size (which is fixed)
- at any time, exactly one head can read/write



ACCESSING THE DISK

- unit of read or write: block size
- once in memory, we refer to it as a **page**
- typically: 4k or 8k or 16k

access time = rotational delay + seek time + transfer time

ACCESSING THE DISK (1)

access time = rotational delay + seek time **+** transfer time

rotational delay: time to wait for sector to rotate under the disk head

- typical delay: *0–10 ms*
- maximum delay = 1 full rotation
- average delay ~ half rotation

RPM	**Average delay**
5,400	5.56
7,200	4.17
10,000	3.00
15,000	2.00

ACCESSING THE DISK (2)

access time = rotational delay **+ seek time +** transfer time

seek time: time to move the arm to position disk head on the right track

- typical seek time: ~ *9 ms*
- ~ *4 ms* for high-end disks

ACCESSING THE DISK (3)

access time = rotational delay **+** seek time **+ transfer time**

data transfer time: time to move the data to/from the disk surface

- typical rates: *~100 MB/s*
- the access time is dominated by the seek time and rotational delay!

EXAMPLE: SPECS

	Seagate HDD
Capacity	*3 TB*
RPM	*7,200*
Average Seek Time	*9 ms*
Max Transfer Rate	*210 MB/s*
# Platters	*3*

What are the I/O rates for block size *4 KB* and:

- random workload (~ *0.3 MB/s*)
- sequential workload (~ *210 MB/s*)

EXAMPLE: RANDOM WORKLOAD

	Seagate HDD
Capacity	*3 TB*
RPM	*7,200*
Average Seek Time	*9 ms*
Max Transfer Rate	*210 MB/s*
# Platters	*3*

For a 4KB block:

- rotational delay = 4.17 ms
- seek time = 9 ms
- transfer time = (4KB) / (210 MB/s) ~ 0.019 ms
- total time per block = 13.1 ms
- I/O rate = (4KB) / (13.1 ms) ~ 0.3 MB/s

ACCESSING THE DISK

- Blocks in a file should be arranged sequentially on disk to minimize seek and rotational delay!

- next block concept:
  - blocks on same track, followed by
  - blocks on same cylinder, followed by
  - blocks on adjacent cylinder

MANAGING DISK SPACE

- The disk space is organized into files
- Files are made up of pages
- Pages contain records

- Data is allocated/deallocated in increments of pages
- Logically close pages should be nearby in the disk

SSD (Solid State Drive)

- SSDs use flash memory
- No moving parts (no rotate/seek motors)
  - eliminates seek time and rotational delay
  - very low power and lightweight
- Data transfer rates: 300-600 MB/s
- SSDs can read data (sequential **or** random) very fast!

SSDs

- Small storage (0.1-0.5x of HDD)
- expensive (20x of HDD)
- Writes are much more expensive than reads (10x)
- Limited lifetime
  – 1-10K writes per page
  – the average failure rate is 6 years

Buffer Management

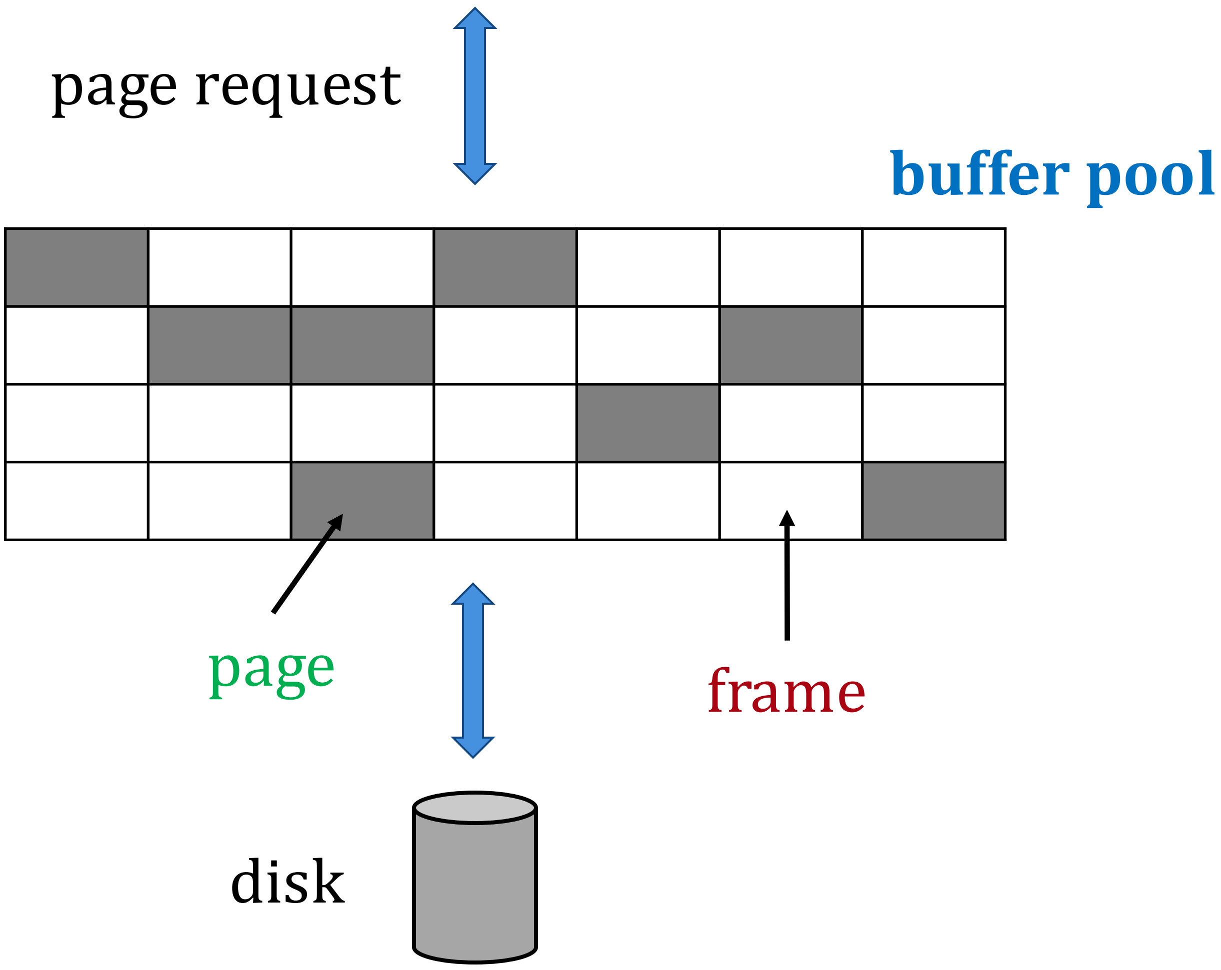
BUFFER MANAGER

- Data must be in RAM for DBMS to operate on it
- All the pages may not fit into main memory

Buffer manager: responsible for bringing pages from disk to main memory as needed

- pages brought into main memory are in the buffer pool
- the buffer pool is partitioned into frames: slots for holding disk pages

BUFFER MANAGER



BUFFER MANAGER: REQUESTS

- **<u>Read</u> (page):** read a page from disk and add to the buffer pool *(if not already in buffer)*

- **<u>Flush</u> (page):** evict page from buffer pool & write to disk

- **<u>Release</u> (page):** evict page from buffer pool *without* writing to disk

BOOKKEEPING

Bookkeeping per frame:

- pin count: # current users of the page
  - *pinning* : increment the pin count
  - *unpinning* : decrement the pin count
- dirty bit: indicates if the page has been modified
  - bit = 1 means that the changes to the page must be propagated to the disk

PAGE REQUEST

- Page is in the buffer pool:
  - return the address to the frame
  - increment the pin count
- Page is not in the buffer pool:
  - choose a frame for replacement (with pin count = 0)
  - if frame is dirty, write the page to disk
  - read requested page into chosen frame
  - pin the page and return the address

BUFFER REPLACEMENT POLICY

- How do we choose a frame for replacement?
  - LRU (**L**east **R**ecently **U**sed)
  - Clock
  - MRU (**M**ost **R**ecently **U**sed)
  - FIFO, random, …

- The replacement policy has big impact on # of I/O's (depends on the access pattern)

LRU

LRU (**L**east **R**ecently **U**sed)

- uses a **queue** of pointers to frames that have pin count = 0
- a page request uses frames only from the *head* of the queue
- when the pin count of a frame goes to 0, it is added to the *end* of the queue

LRU EXAMPLE

	frame	dirty	pincount
1		0	0
2		0	0
3		0	0

priority queue: 1 **|** 2 **|** 3

Sequence of requests:

request A, modify A, request B, request B, release A, request C, release B, request D, modify D, release B, request A, request E

LRU EXAMPLE

	frame	dirty	pincount
1	**A**	0	**1**
2		0	0
3		0	0

priority queue: 2 | 3

one I/O to read the page

Sequence of requests:

request A, modify A, request B, request B, release A, request C, release B, request D, modify D, release B, request A, request E

LRU EXAMPLE

	frame	dirty	pincount
1	A	**1**	1
2		0	0
3		0	0

priority queue: 2 | 3

no I/O here!

Sequence of requests:

request A, **modify A**, request B, request B, release A, request C, release B, request D, modify D, release B, request A, request E

LRU EXAMPLE

	frame	dirty	pincount
1	A	1	1
2	**B**	0	**1**
3		0	0

priority queue: 3

one I/O to read the page

Sequence of requests:

request A, modify A, **request B**, request B, release A, request C, release B, request D, modify D, release B, request A, request E

LRU EXAMPLE

	frame	dirty	pincount
1	A	1	1
2	B	0	**2**
3		0	0

priority queue: 3

No I/O here
The pincount increases!

Sequence of requests:

request A, modify A, request B, **request B**, release A, request C, release B, request D, modify D, release B, request A, request E

LRU EXAMPLE

	frame	dirty	pincount
1	A	1	**0**
2	B	0	2
3		0	0

priority queue: 3 | 1

no I/O yet!

Sequence of requests:

request A, modify A, request B, request B, **release A**, request C, release B, request D, modify D, release B, request A, request E

LRU EXAMPLE

	frame	dirty	pincount
1	A	1	0
2	B	0	2
3	**C**	0	**1**

priority queue: 1

one I/O to read the page

Sequence of requests:

request A, modify A, request B, request B, release A, **request C**, release B, request D, modify D, release B, request A, request E

LRU EXAMPLE

	frame	dirty	pincount
1	A	1	0
2	B	0	**1**
3	C	0	1

priority queue: 1

the pincount decreases

Sequence of requests:

request A, modify A, request B, request B, release A, request C, **release B**, request D, modify D, release B, request A, request E

LRU EXAMPLE

	frame	dirty	pincount
1	**D**	0	**1**
2	B	0	1
3	C	0	1

priority queue:

two I/Os: one to write A to disk and one to read D

Sequence of requests:

request A, modify A, request B, request B, release A, request C, release B, **request D**, modify D, release B, request A, request E

LRU EXAMPLE

	frame	dirty	pincount
1	D	**1**	1
2	B	0	1
3	C	0	1

priority queue:

no I/O here

Sequence of requests:

request A, modify A, request B, request B, release A, request C, release B, request D, **modify D**, release B, request A, request E

LRU EXAMPLE

	frame	dirty	pincount
1	D	1	1
2	B	0	**0**
3	C	0	1

priority queue: 2

no I/O

Sequence of requests:

request A, modify A, request B, request B, release A, request C, release B, request D, modify D, **release B**, request A, request E

LRU EXAMPLE

	frame	**dirty**	**pincount**
1	D	1	1
2	**A**	0	**1**
3	C	0	1

priority queue:

one I/O to read A

Sequence of requests:

request A, modify A, request B, request B, release A, request C, release B, request D, modify D, release B, **request A**, request E

LRU EXAMPLE

	frame	dirty	pincount
1	D	1	1
2	A	0	1
3	C	0	1

priority queue:

The buffer pool is full, the request must wait!

Sequence of requests:

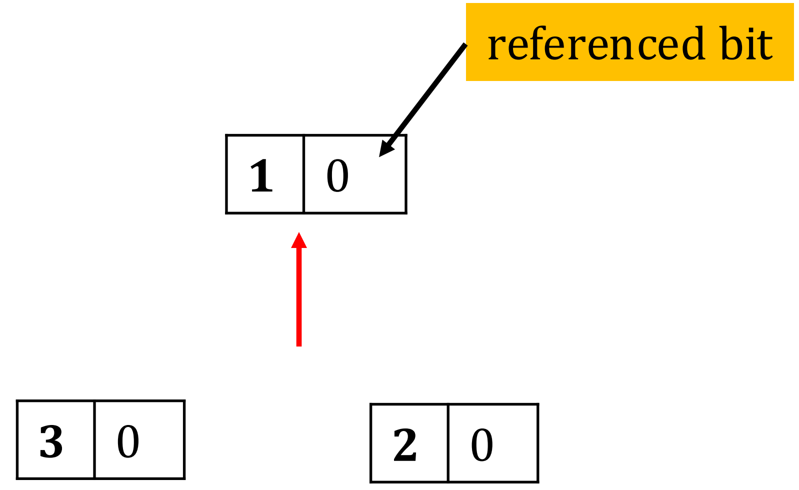
request A, modify A, request B, request B, release A, request C, release B, request D, modify D, release B, request A, **request E**

CLOCK

- Variant of LRU with lower memory overhead
- The N frames are organized into a cycle
- Each frame has a referenced bit that is set to 1 when pin count becomes 0
- A current variable points to a frame
- When a frame is considered:
  - If pin count > 0, increment current
  - If referenced = 1, set to 0 and increment
  - If referenced = 0 and pin count = 0, choose the page

CLOCK EXAMPLE

	frame	dirty	pincount
1		0	0
2		0	0
3		0	0

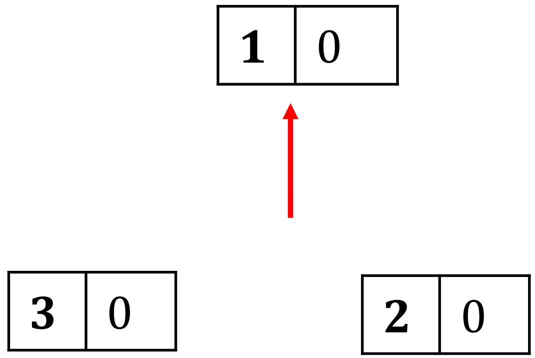


Sequence of requests:

request A, modify A, request B, request B, release A, request C, release B, request D, modify D, release B, request A, request E

CLOCK EXAMPLE

	frame	dirty	pincount
1	A	0	1
2		0	0
3		0	0

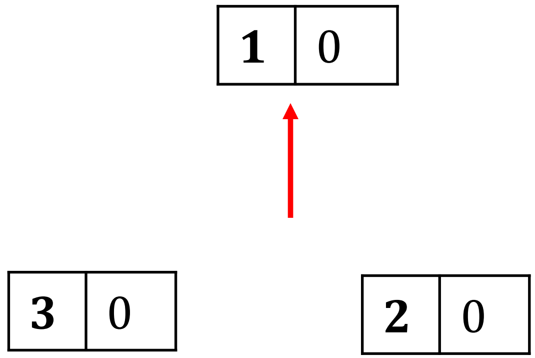


Sequence of requests:

request A, modify A, request B, request B, release A, request C, release B, request D, modify D, release B, request A, request E

CLOCK EXAMPLE

	frame	dirty	pincount
1	A	1	1
2		0	0
3		0	0

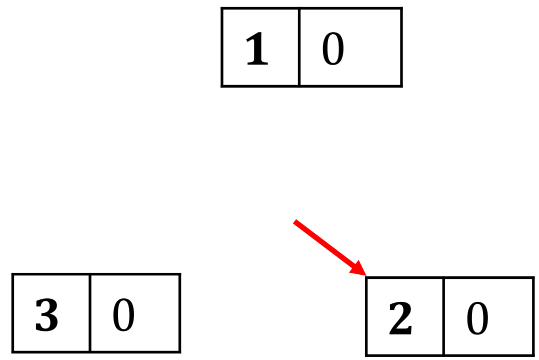


Sequence of requests:

request A, **modify A**, request B, request B, release A, request C, release B, request D, modify D, release B, request A, request E

CLOCK EXAMPLE

	frame	dirty	pincount
1	A	1	1
2	B	0	1
3		0	0

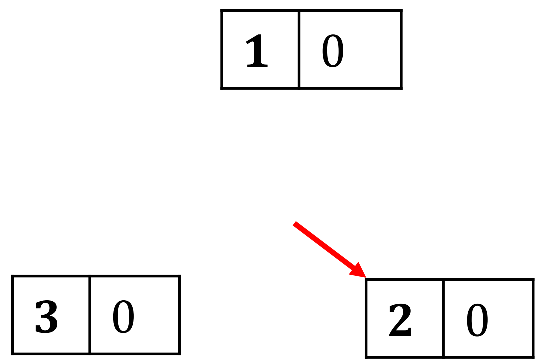


Sequence of requests:

request A, modify A, **request B**, request B, release A, request C, release B, request D, modify D, release B, request A, request E

CLOCK EXAMPLE

	frame	dirty	pincount
1	A	1	1
2	B	0	2
3		0	0

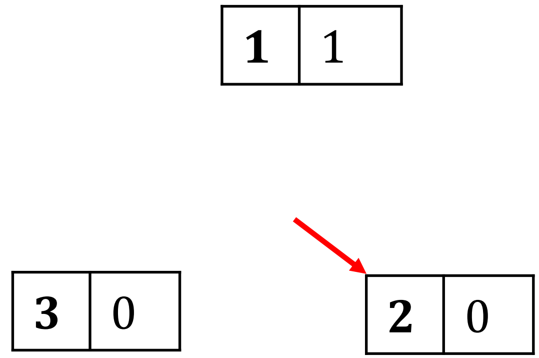


Sequence of requests:

request A, modify A, request B, **request B**, release A, request C, release B, request D, modify D, release B, request A, request E

CLOCK EXAMPLE

	frame	dirty	pincount
1	A	1	0
2	B	0	2
3		0	0

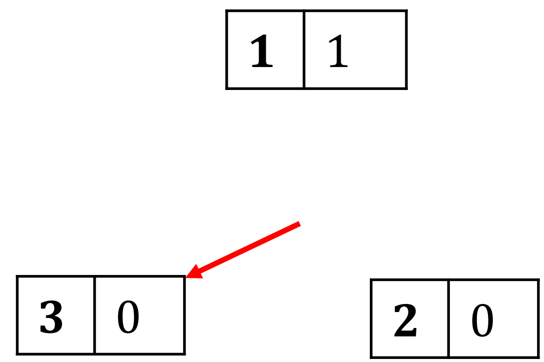


Sequence of requests:

request A, modify A, request B, request B, **release A**, request C, release B, request D, modify D, release B, request A, request E

CLOCK EXAMPLE

	frame	dirty	pincount
1	A	1	0
2	B	0	2
3	C	0	1

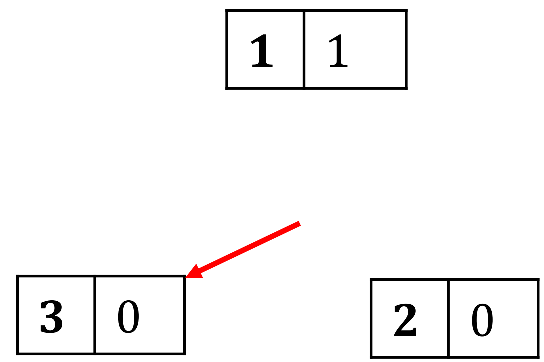


Sequence of requests:

request A, modify A, request B, request B, release A, **request C**, release B, request D, modify D, release B, request A, request E

CLOCK EXAMPLE

	frame	dirty	pincount
1	A	1	0
2	B	0	1
3	C	0	1

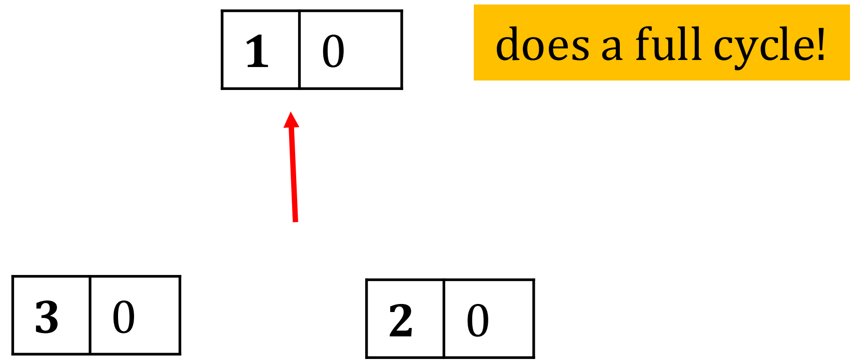


Sequence of requests:

request A, modify A, request B, request B, release A, request C, **release B**, request D, modify D, release B, request A, request E

CLOCK EXAMPLE

	frame	dirty	pincount
1	D	0	1
2	B	0	1
3	C	0	1

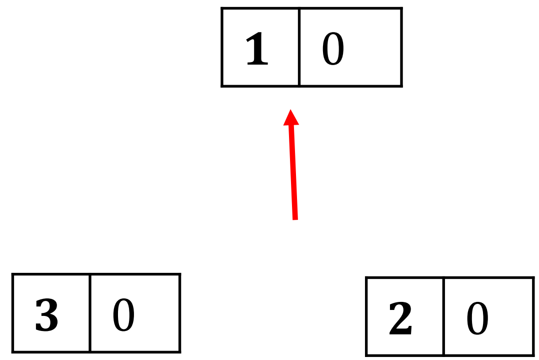


Sequence of requests:

request A, modify A, request B, request B, release A, request C, release B, **request D**, modify D, release B, request A, request E

CLOCK EXAMPLE

	frame	dirty	pincount
1	D	1	1
2	B	0	1
3	C	0	1

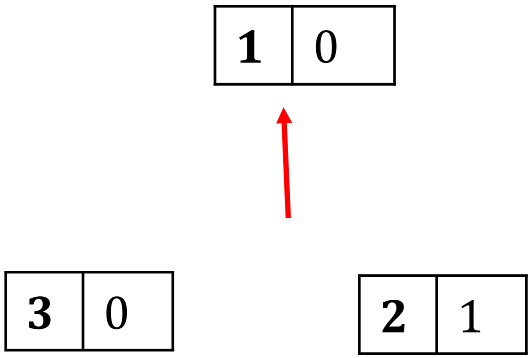


Sequence of requests:

request A, modify A, request B, request B, release A, request C, release B, request D, **modify D**, release B, request A, request E

CLOCK EXAMPLE

	frame	dirty	pincount
1	D	1	1
2	B	0	0
3	C	0	1

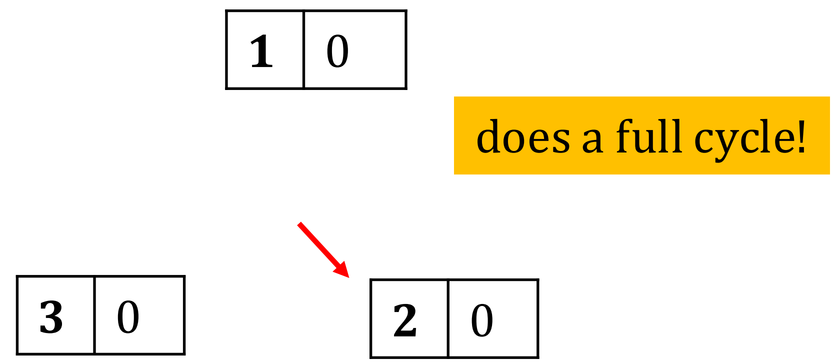


Sequence of requests:

request A, modify A, request B, request B, release A, request C, release B, request D, modify D, **release B**, request A, request E

CLOCK EXAMPLE

	frame	dirty	pincount
1	D	1	1
2	A	0	1
3	C	0	1

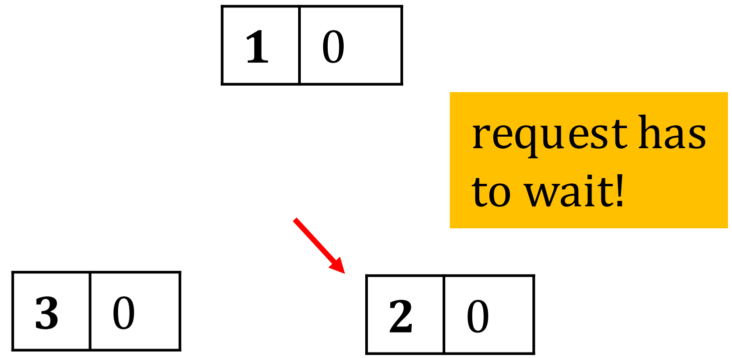


Sequence of requests:

request A, modify A, request B, request B, release A, request C, release B, request D, modify D, release B, **request A**, request E

CLOCK EXAMPLE

	frame	dirty	pincount
1	D	1	1
2	A	0	1
3	C	0	1



Sequence of requests:

request A, modify A, request B, request B, release A, request C, release B, request D, modify D, release B, request A, **request E**

SEQUENTIAL FLOODING: EXAMPLE

- 3 frames in the buffer pool
- request sequence:
  - A, B, C, D, A, B, C, D, A, B, C, D, …
- With LRU policy, every page access needs an I/O!

	frame	dirty	pincount
1		0	0
2		0	0
3		0	0

SEQUENTIAL FLOODING

Sequential Flooding: nasty situation caused by LRU policy + repeated sequential scans

- # buffer frames < # pages in file
- each page request causes an I/O !!
- MRU much better in this situation

DBMS VS OS FILE SYSTEM

Why not let the OS handle disk management?

- DBMS better at predicting the reference patterns
- Buffer management in DBMS requires ability to:
  - pin a page in buffer pool
  - force a page to disk (for recovery & concurrency)
  - adjust the replacement policy
  - pre-fetch pages based on predictable access patterns
- can better control the overlap of I/O with computation
- can leverage multiple disks more effectively