THE UNIVERSITY *of* EDINBURGH

Advanced Database Systems

Spring 2026

Q&A Session 1

If you require this document in an alternative format, such as large print or a coloured background, please contact milos.nikolic@ed.ac.uk

Administrivia

Coursework was released last week

- Start early – several things you can already begin working on
- Ask questions on Piazza

Q&A sessions

- Like office hours
- You can ask questions about material & provide feedback
- Each Q&A session includes a practice worksheet available on Learn

Files, Pages, Records

Tables stored as **logical files** consisting of **pages,** each containing a collection of **records**

File (corresponds to a table)

- Page (many per file)
  - Record (many per page)

The unit of access to physical disk is the page

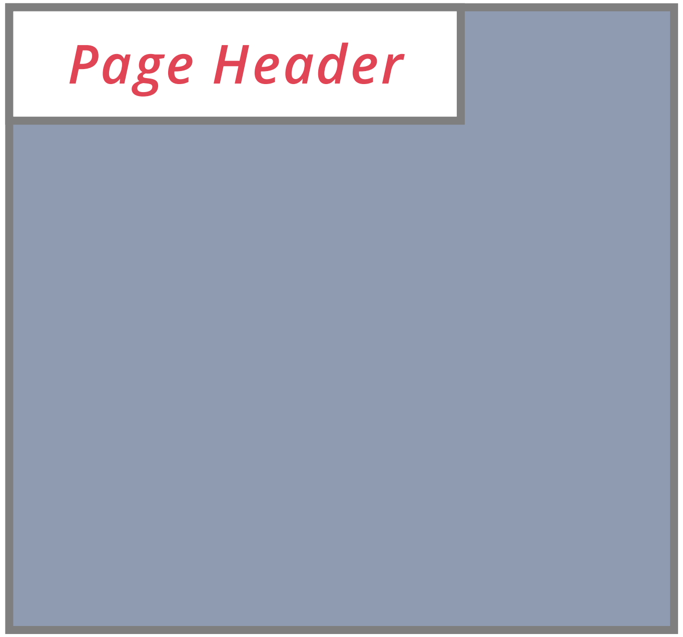
- 1 I/O = read or write 1 page

Page Basics

The page header keeps track of the records in the page

The page header may contain fields such as:

- Number of records in the page
- Pointer to segment of free space in the page
- Bitmap indicating which parts of the page are in use

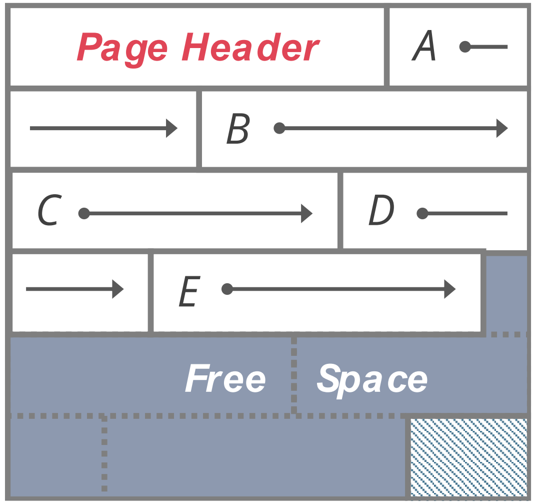


Page

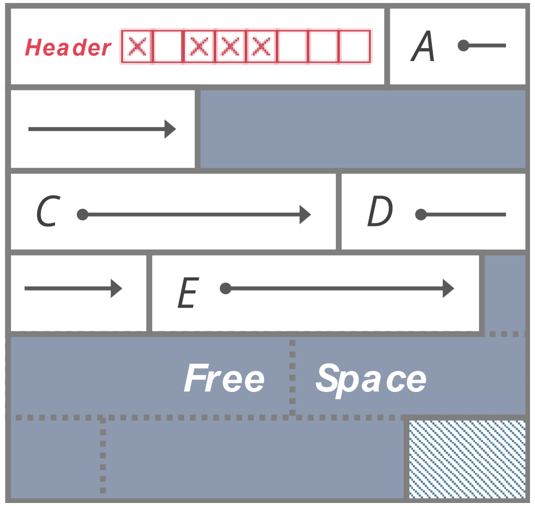
Fixed-Length Records

Fixed-length records = record lengths are fixed and field lengths are consistent

Packed Records: no gaps between records, record ID is location in page



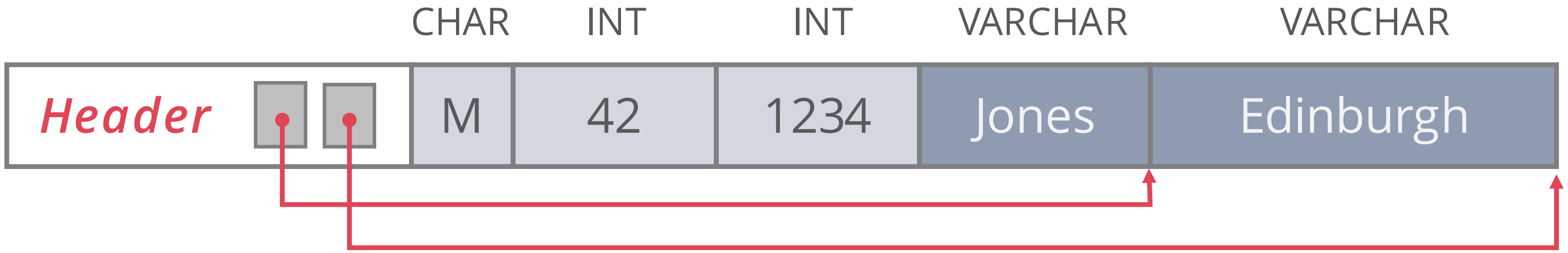
Unpacked Records: allow gaps between records, use a bitmap to keep track of where the gaps are



VARIABLE-LENGTH RECORDS

Variable-length records may not have fixed & consistent field lengths

We can store variable length length records with an array of field offsets:



Each record contains a **record header**

Variable length fields are placed *after* fixed length fields

Record header stores **field offset** (where variable length field ends)

Question 1

Consider the following relation:

Assume record header stores only pointers (4B) to variable-length fields

```
CREATE TABLE Customer (
  customer_id INTEGER PRIMARY KEY,
  age INTEGER NOT NULL,
  name VARCHAR(10) NOT NULL,
  address VARCHAR(20) NOT NULL
)
```

Record header size = ???

Min record size = ???

Max record size = ???

Question 1, Part 2

Consider the following relation:

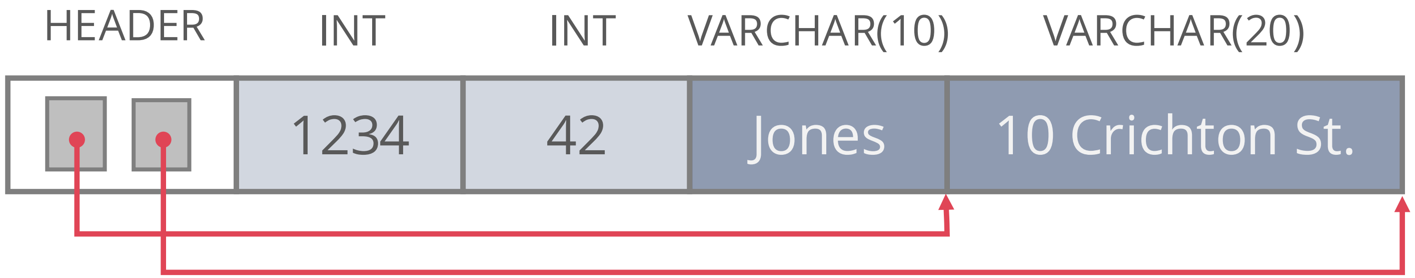
Assume record header stores only pointers (4B) to variable-length fields

```
CREATE TABLE Customer (
  customer_id INTEGER PRIMARY KEY,
  age INTEGER NOT NULL,
  name VARCHAR(10) NOT NULL,
  address VARCHAR(20) NOT NULL
)
```

Record header size = 8

Min record size = 16

Max record size = 46



SLOTTED PAGES

Most common layout scheme is called **slotted pages**

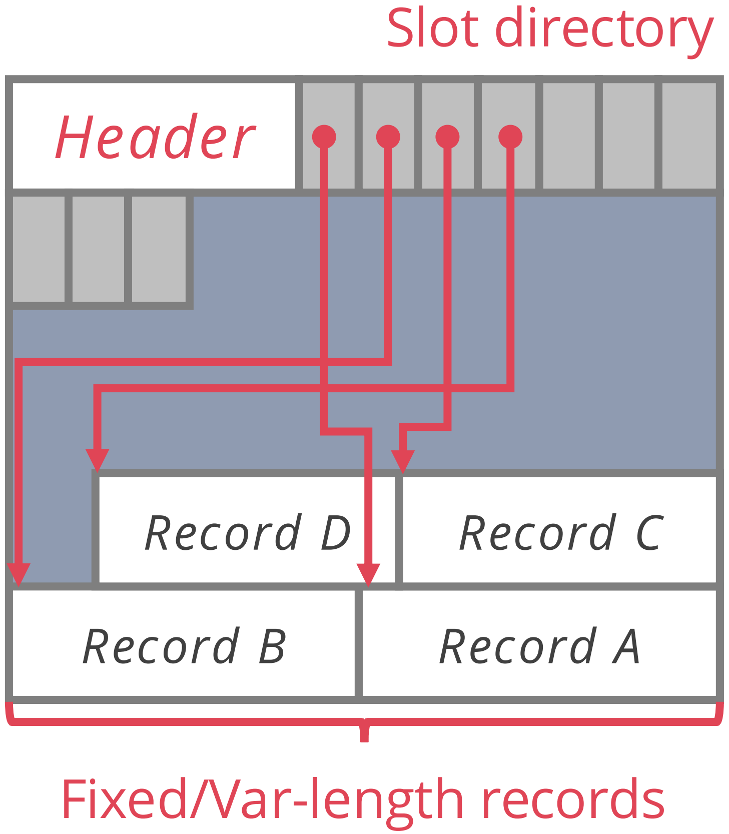
Slot directory maps "slots" to the records' starting position offsets

- Record ID = (page ID, slot ID)

Header keeps track of:

- The number of used slots
- The offset of the last slot used

Records stored at the end of page

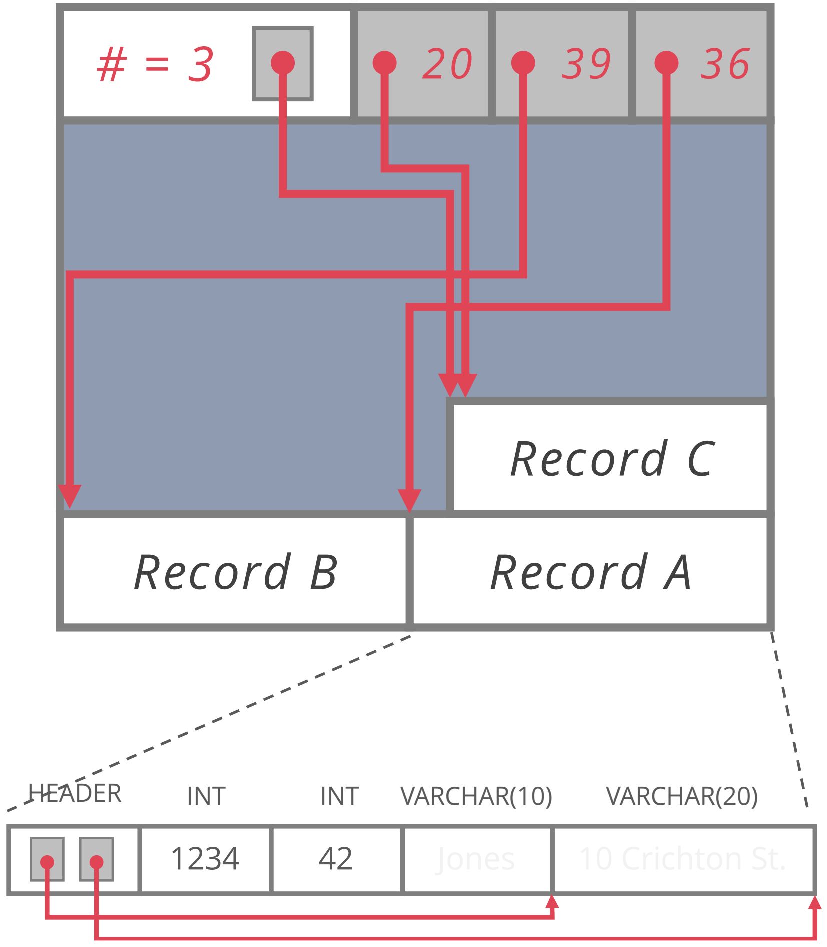


QUESTION 2

Suppose the Customer relation is stored using a slotted page layout

- **Page header** stores the number of records and a pointer to free space
- **Directory slot** stores a pointer and length
- **Page size** is 8KB

Max number of records = ???



Question 2, Part 2

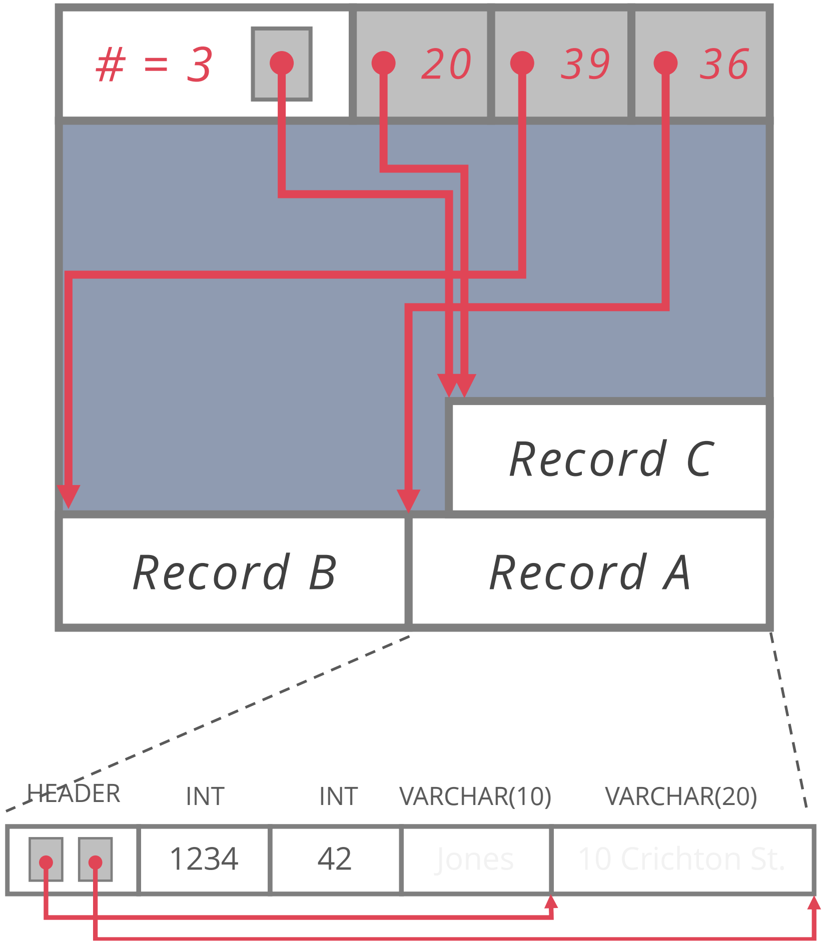
Suppose the Customer relation is stored using a slotted page layout

- **Page header** stores the number of records and a pointer to free space **(4B + 4B)**
- **Directory slot** stores a pointer and length **(4B + 4B)**
- **Page size** is 8KB

Max number of records

= (page size – header size) / (min record size + slot size)

= (8192 – 8) / (16 + 8) = **341 records**



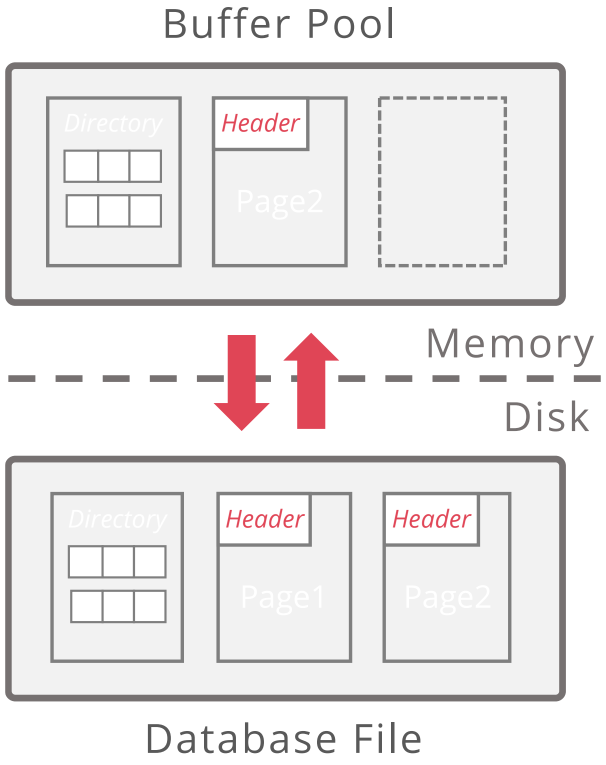
Buffer Management

Buffer Manager

Layer that manages which pages are loaded in memory

Controls when pages are read from & written to disk

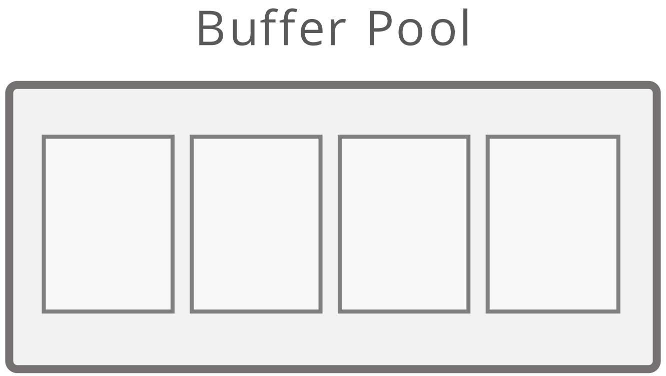
- When no space in memory, decides what page to **evict**
- Decision process is the **page replacement policy**
  - Big impact on I/Os depending on **access pattern**
- Common policies:
  - **LRU** (Least Recently Used)
  - **MRU** (Most Recently Used)
  - **Clock**



Question 3

Page access sequence:

A B C D E B A D C A E C



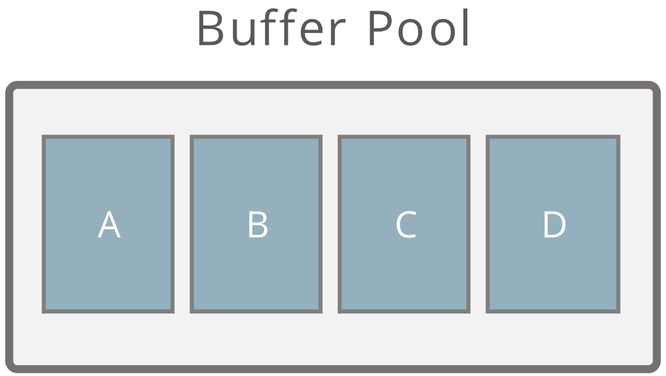
Most Recently Used (MRU)

Buffer hits = ???

Question 3 – After 4 Requests

Page access sequence:

A B C D E B A D C A E C


Buffer Pool

A B C D

Most Recently Used (MRU)

Buffer hits = 0

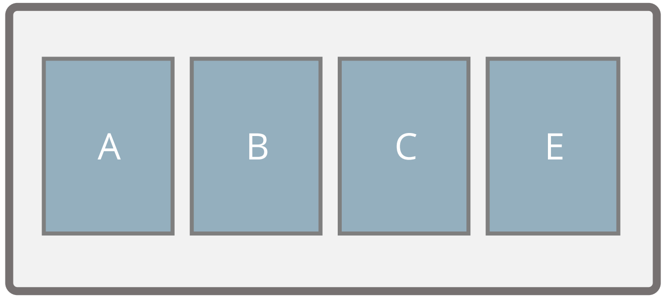
A (miss), B (miss), C (miss), D (miss)

Question 3 – After 7 Requests

Page access sequence:

A B C D E B A D C A E C

Buffer Pool



Most Recently Used (MRU)

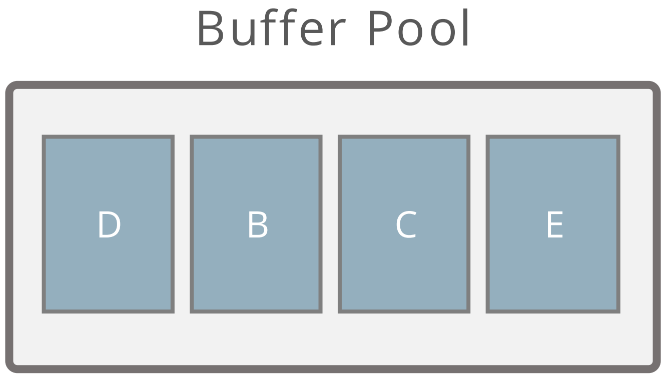
Buffer hits = 2

A (miss), B (miss), C (miss), D (miss), E (miss, D out), **B (hit)**, **A (hit)**

Question 3 – After 10 Requests

Page access sequence:

A B C D E B A D C A E C



Most Recently Used (MRU)

Buffer hits = 3

A (miss), B (miss), C (miss), D (miss), E (miss, D out), **B (hit)**, **A (hit)**,

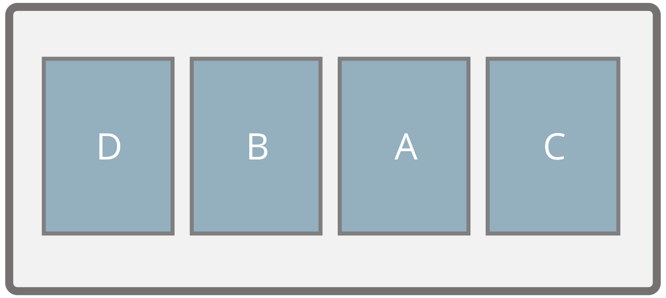
D (miss, A out), **C (hit)**

Question 3 – After 12 Requests

Page access sequence:

A B C D E B A D C A E C

Buffer Pool



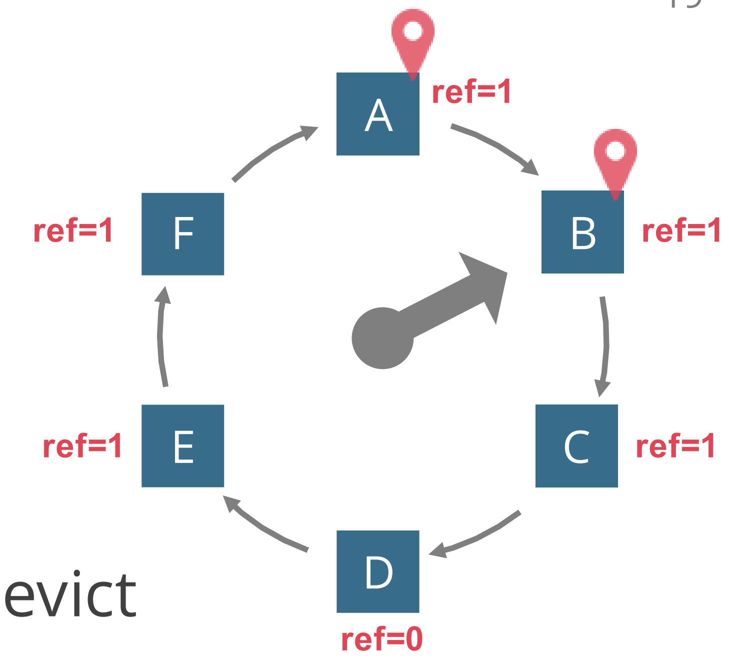
D B A C

Most Recently Used (MRU)

Buffer hits = 4

A (miss), B (miss), C (miss), D (miss), E (miss, D out), **B (hit)**, **A (hit)**, D (miss, A out), **C (hit)**, A (miss, C out), **E (hit)**, C (miss, E out)

Clock



Efficient approximation of LRU

Arrange frames in a circle (like numbers on a clock)

Advance clock hand around the clock to find pages to evict

- Only do this if you need to evict a page

To make this approximate least recently *used* (rather than least recently *loaded*): add a reference bit to each frame

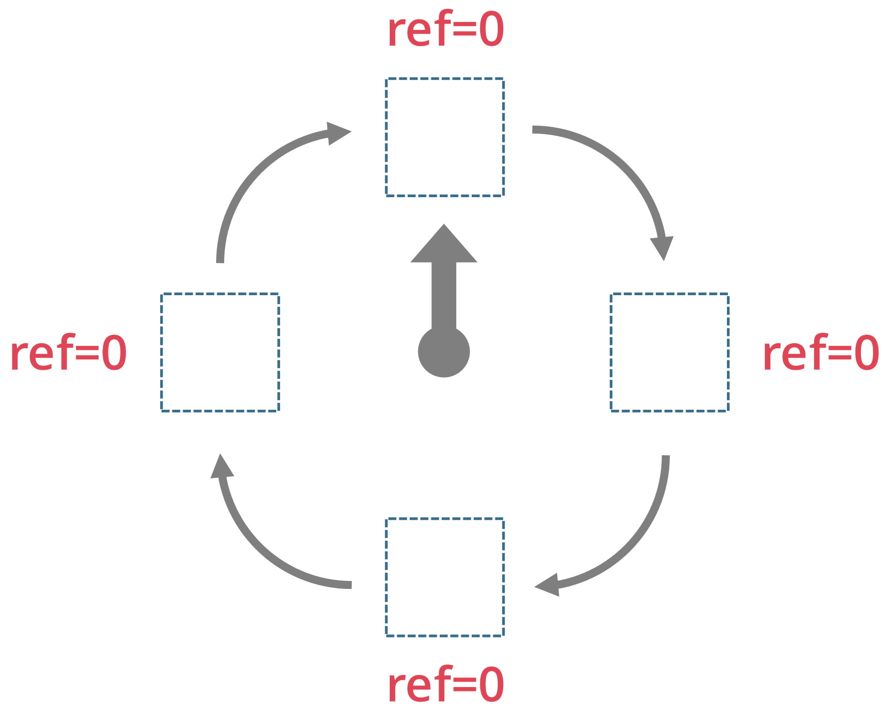
- Set to 1 on load/hit, 0 if clock hand passes the frame and the frame is unpinned
- Evict unpinned frame if clock hand reaches it and bit = 0
- (bit = 0 means less recently used than those with bit = 1)

Question 4

Page access sequence:

A B C D E B A D C A E C

Assume pages are immediately unpinned after being pinned



Buffer hits = ???

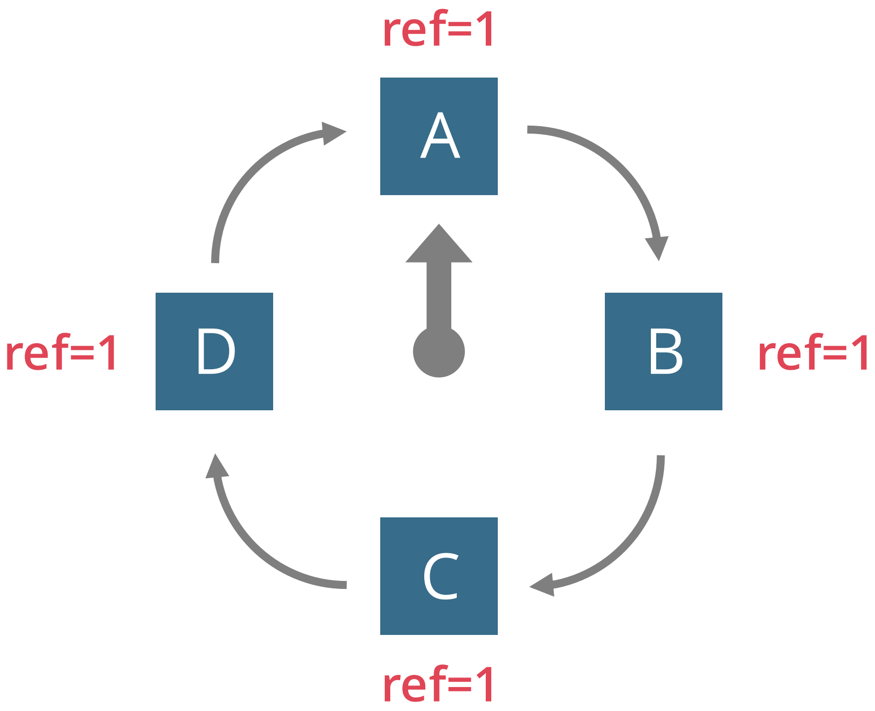
Question 4, Part 2

Page access sequence:

A B C D E B A D C A E C

Pages A, B, C, D populate the buffer pool

The clock hand stays still



Buffer hits (so far) = 0

Question 4, Part 3

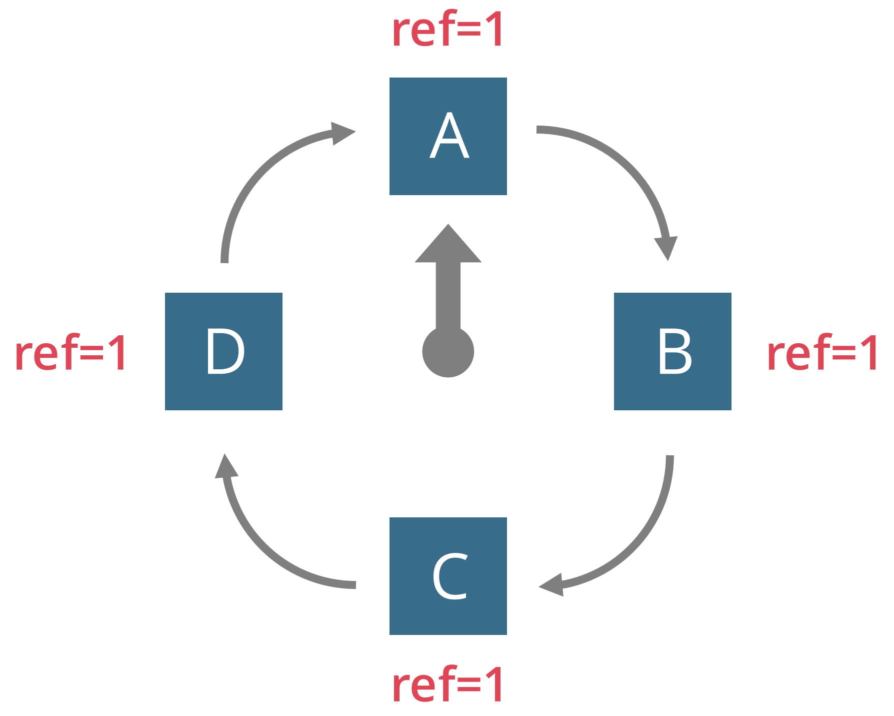
Page access sequence:

A B C D E B A D C A E C

Page E not present ⇒ buffer miss!

Find first frame with ref = 0

If ref = 1, unset it and move the hand



Buffer hits (so far) = 0

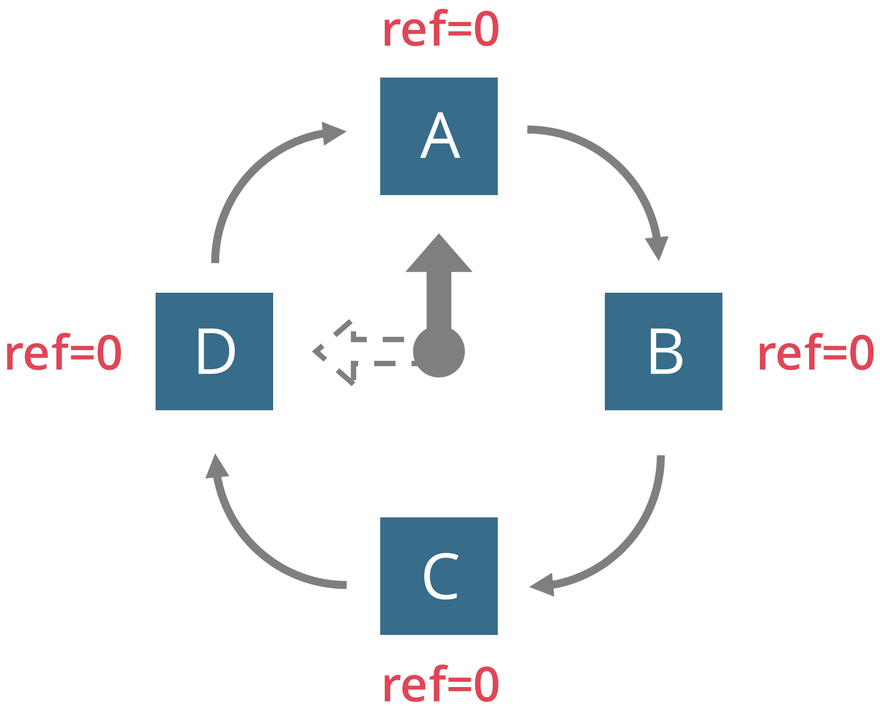
Question 4, Part 4

Page access sequence:

A B C D E B A D C A E C

Resets bits of A, B, C, D while moving the hand

First frame with ref = 0 holds A



Buffer hits (so far) = 0

Question 4, Part 5

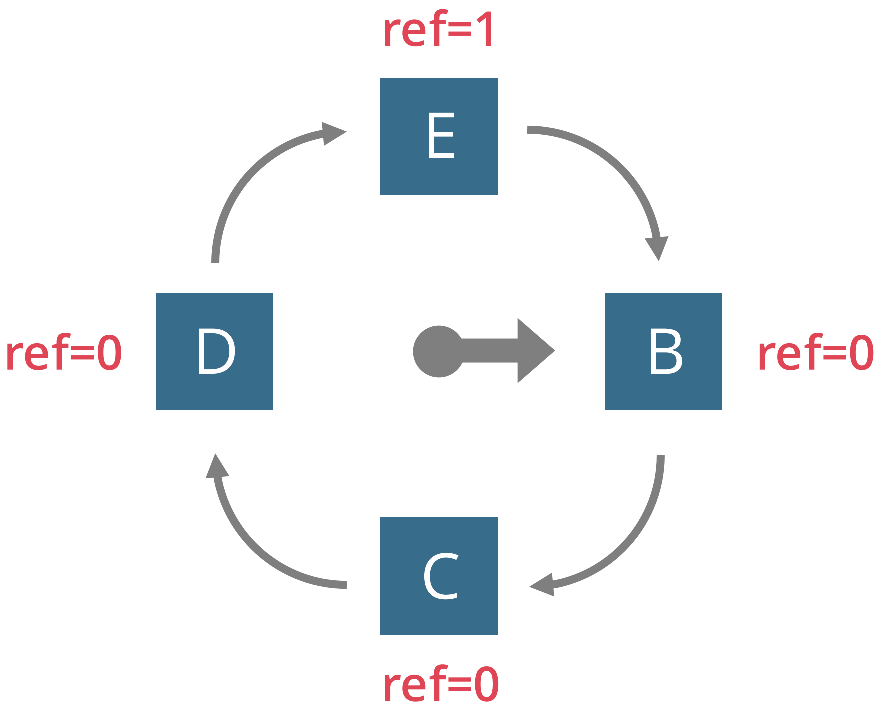
Page access sequence:

A B C D E B A D C A E C

Resets bits of A, B, C, D while moving the hand

First frame with ref = 0 holds A

Replace A with E, set reference bit, move the hand



Buffer hits (so far) = 0

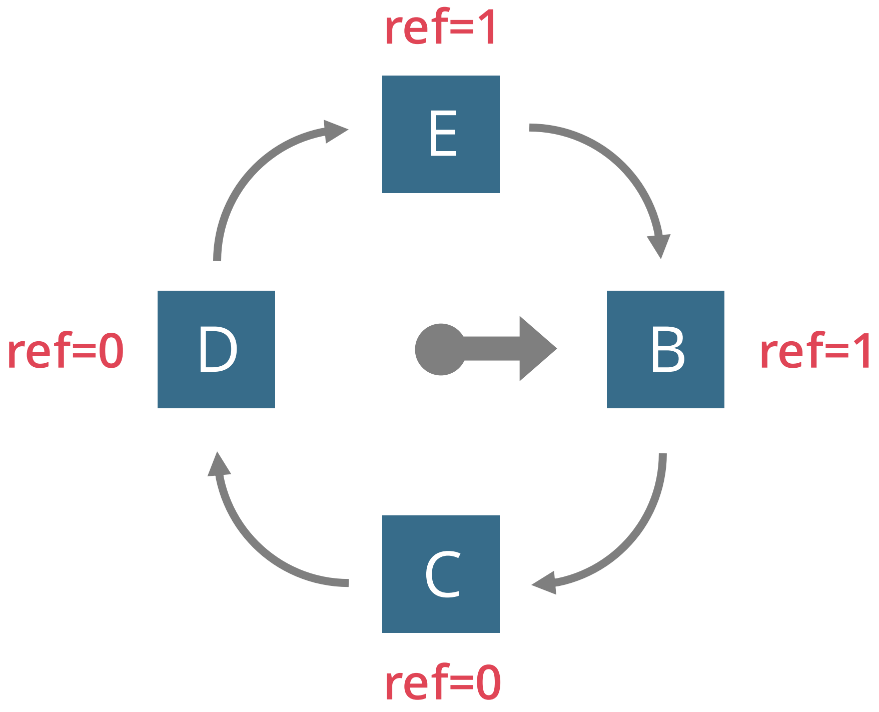
Question 4, Part 6

Page access sequence:

A B C D E B A D C A E C

Page B is present ⇒ buffer hit!

Set refence bit



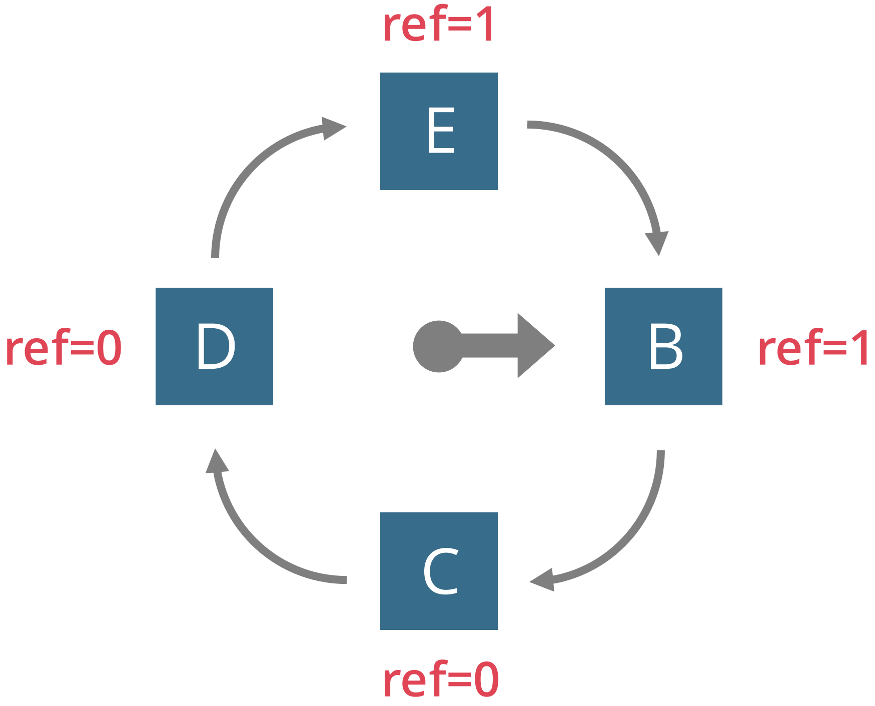
Buffer hits (so far) = 1

Question 4, Part 7

Page access sequence:

A B C D E B A D C A E C

Page A not present ⇒ buffer miss!



Buffer hits (so far) = 1

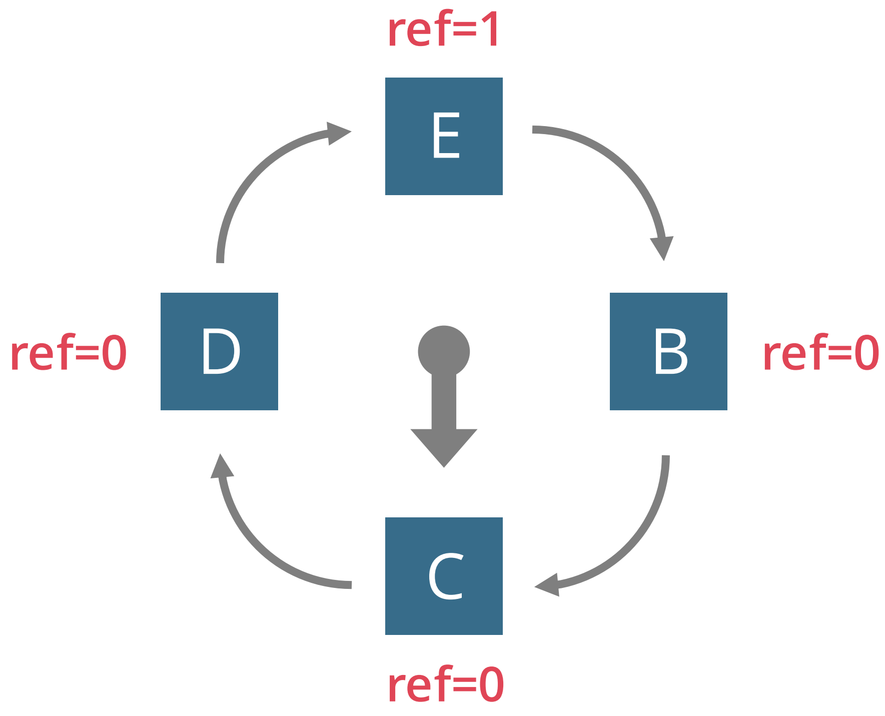
Question 4, Part 8

Page access sequence:

A B C D E B A D C A E C

Page A not present ⇒ buffer miss!

Unset refence bit for B, move the hand



Buffer hits (so far) = 1

Question 4, Part 9

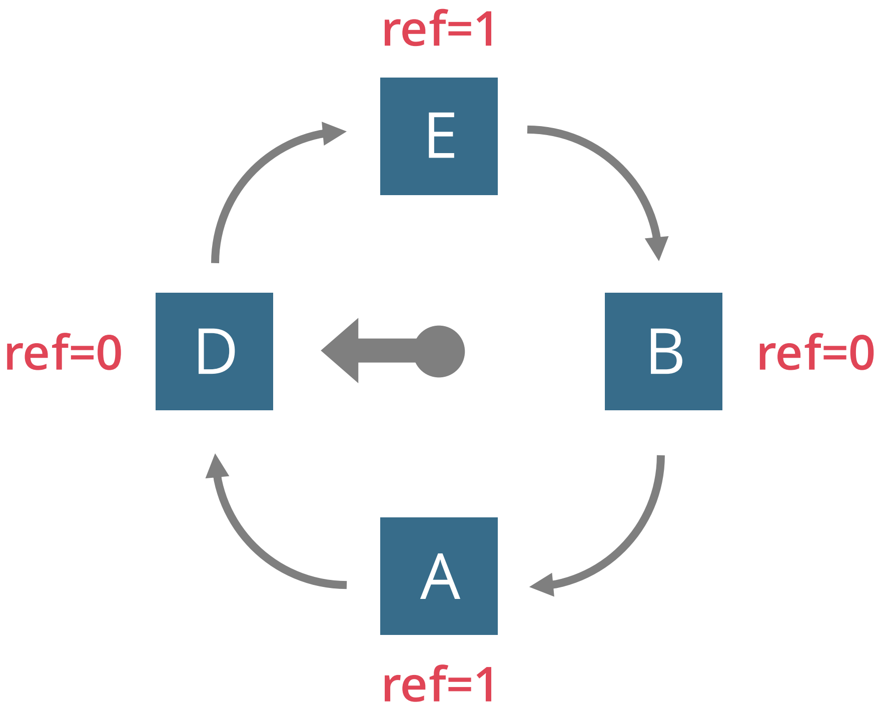
Page access sequence:

A B C D E B A D C A E C

Page A not present ⇒ buffer miss!

Unset refence bit for B, move the hand

Replace C with A, set refence bit, move the hand



Buffer hits (so far) = 1

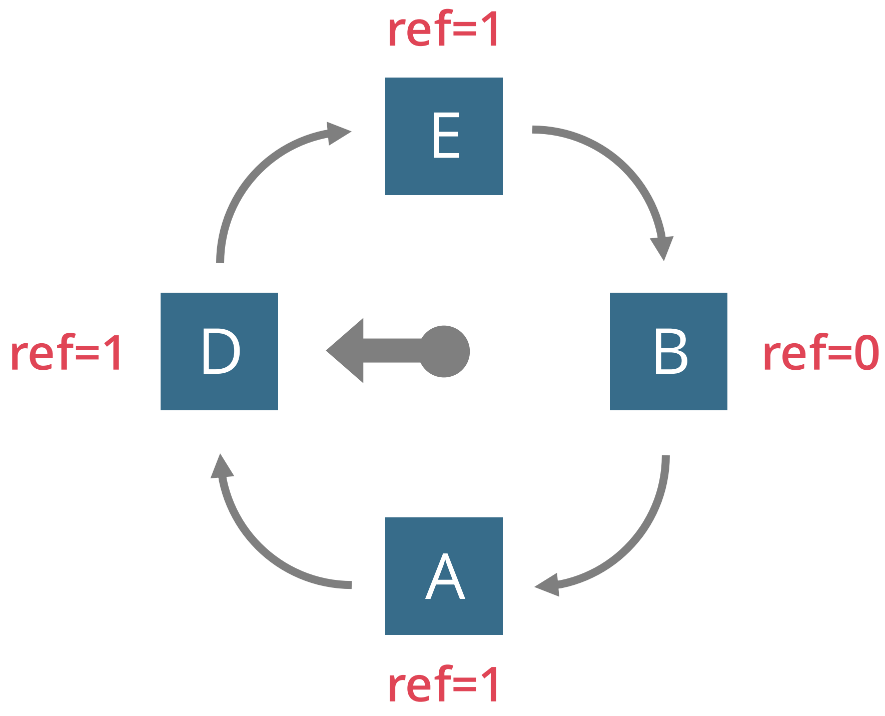
Question 4, Part 10

Page access sequence:

A B C D E B A D C A E C

Page D is present ⇒ buffer hit!

Set refence bit



Buffer hits (so far) = 2

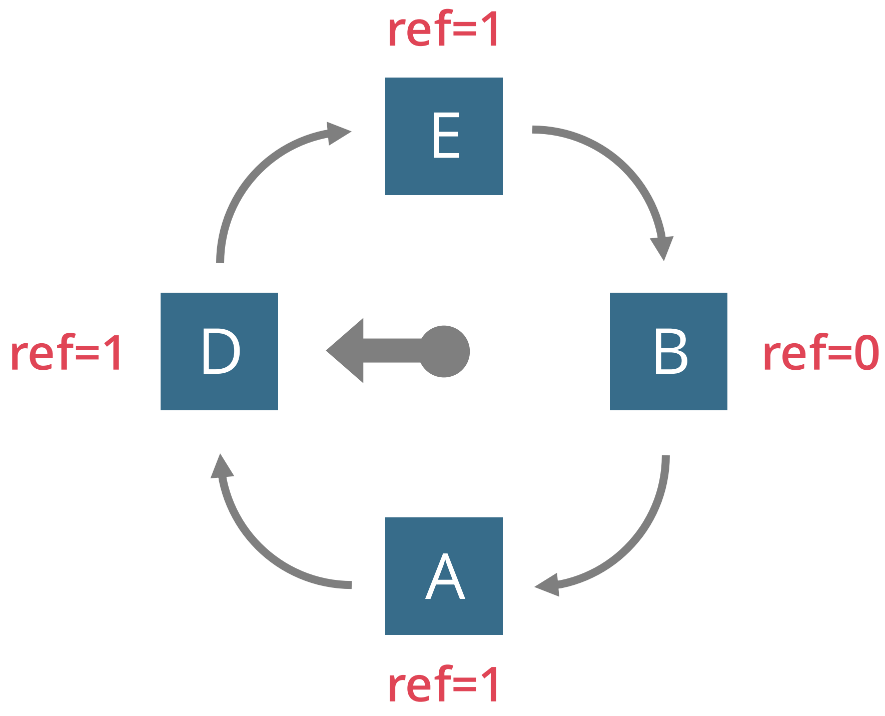
Question 4, Part 11

Page access sequence:

A B C D E B A D C A E C

⬆

Page C is not present ⇒ buffer miss!



Buffer hits (so far) = 2

Question 4, Part 12

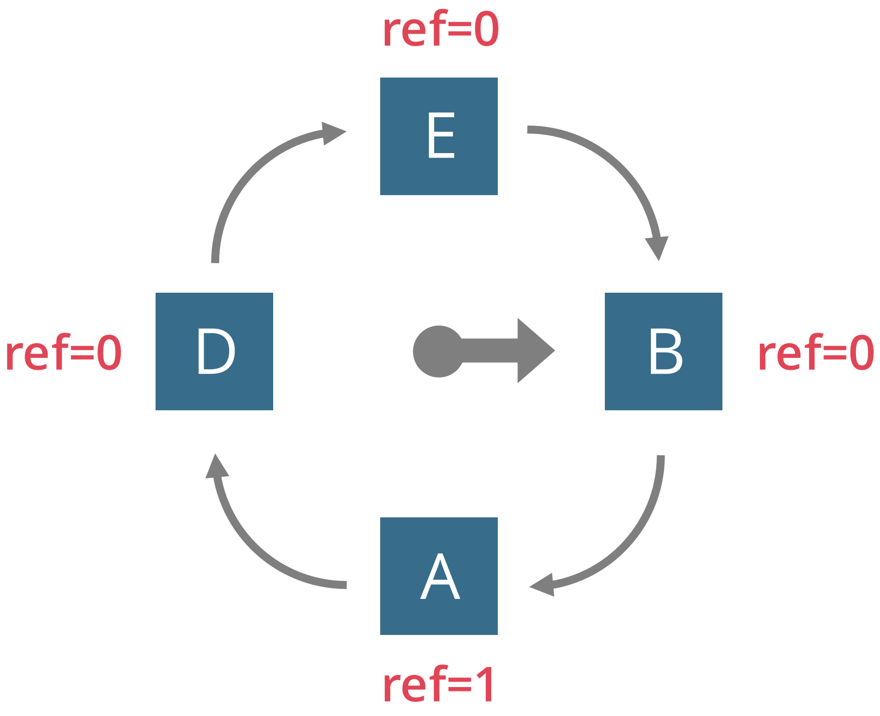
Page access sequence:

A B C D E B A D C A E C

⬆

Page C is not present ⇒ buffer miss!

Unset ref bits for D & E, move the hand to B



Buffer hits (so far) = 2

Question 4, Part 13

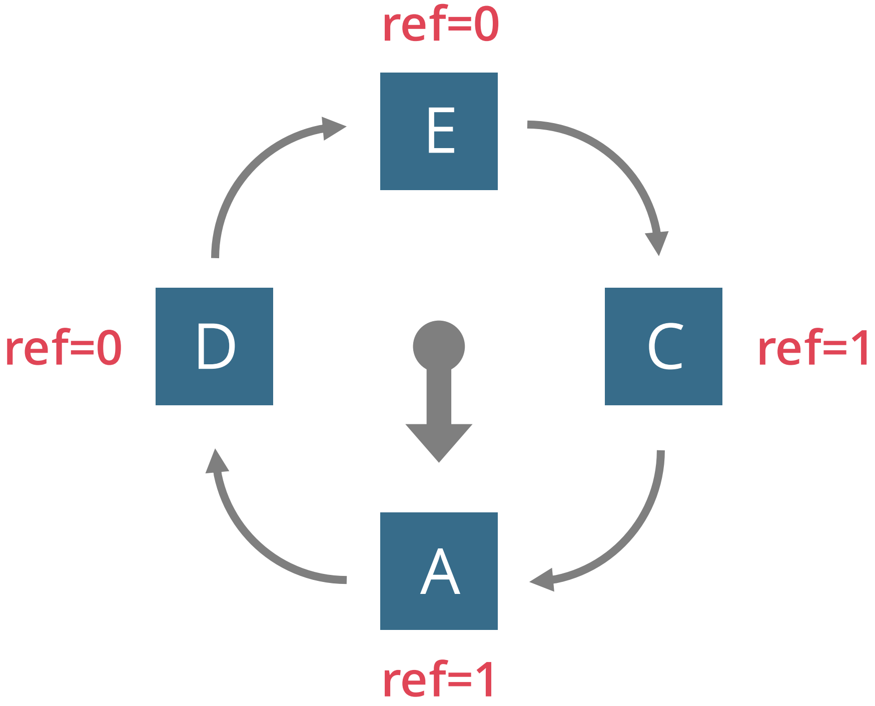
Page access sequence:

A B C D E B A D C A E C

Page C is not present ⇒ buffer miss!

Unset ref bits for D & E, move the hand to B

Replace B with C, set refence bit, move the hand



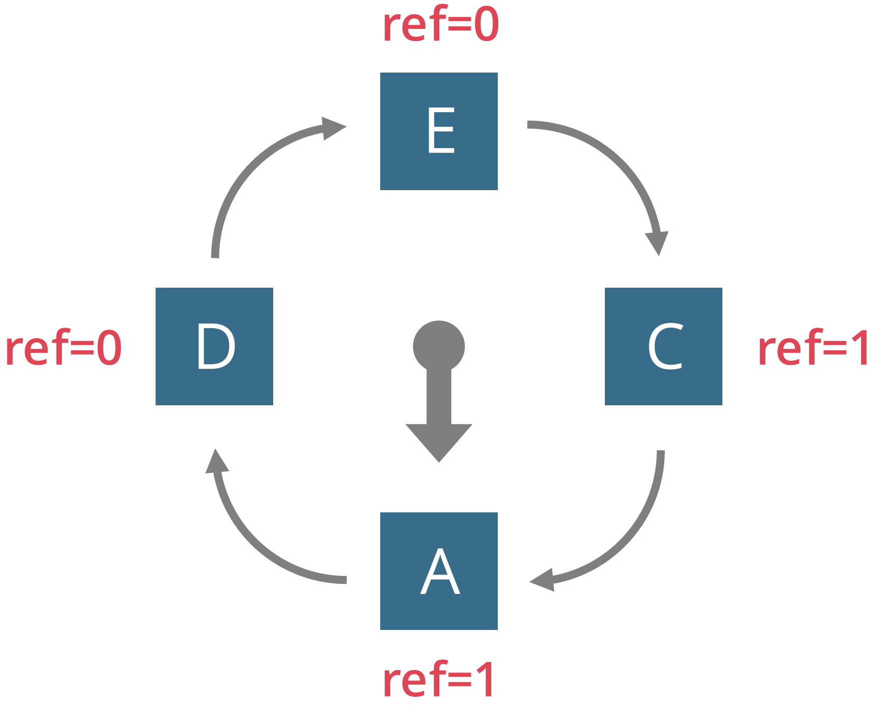
Buffer hits (so far) = 2

Question 4, Part 14

Page access sequence:

A B C D E B A D C A E C

Pages A, E, C are present ⇒ buffer hits!



Buffer hits (so far) = 2

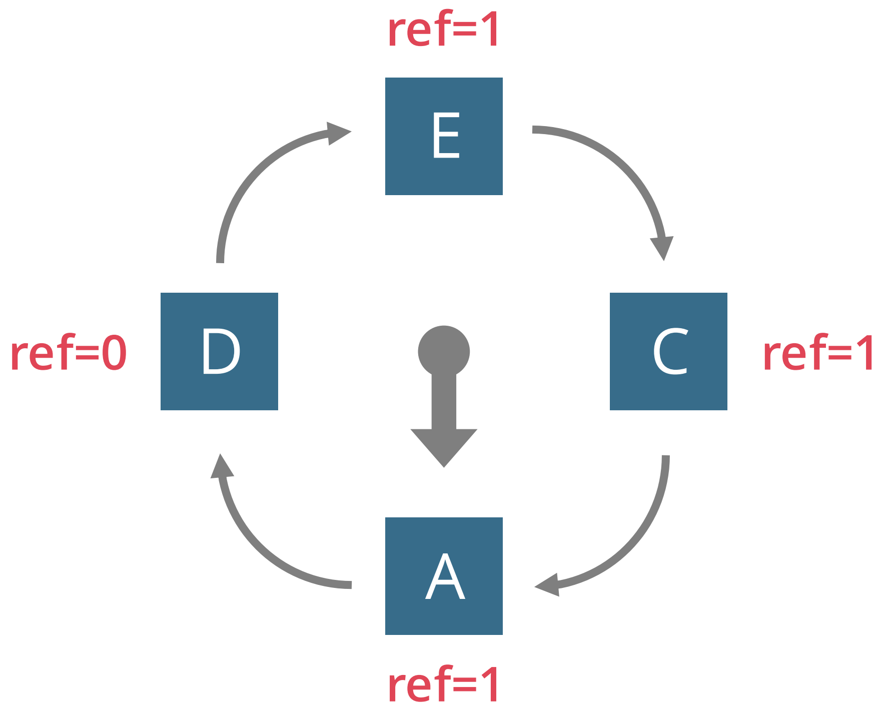
Question 4, Part 15

Page access sequence:

A B C D E B A D C A E C

Pages A, E, C are present ⇒ buffer hits!

Set their reference bits



Buffer hits = 5

PostgreSQL – Buffer Pool Demo

Purpose

What the PostgreSQL buffer pool (*shared_buffers*) is

How sequential scans use a ring buffer to avoid cache pollution

How index-based access uses the normal buffer pool

Key ideas

PostgreSQL stores data in 8 KB pages

Pages are cached in *shared_buffers*

Different access patterns use the cache differently

Try it out

`buffer_demo.sql` is available on Learn → Practice Worksheets

Requires PostgreSQL installed locally

Run the script step by step and observe buffer behaviour

PostgreSQL – Slotted Pages Demo

Purpose

How tables are stored as slotted pages

How inserts/deletes create dead space

How **VACUUM** and **VACUUM FULL** reclaim space

Key ideas

Pages contain headers, pointers, tuples, and free space

Dead tuples persist until cleaned

Try it out

`page_demo.sql` is available on Learn → Practice Worksheets

Run the script step by step and observe space usage before and after VACUUM