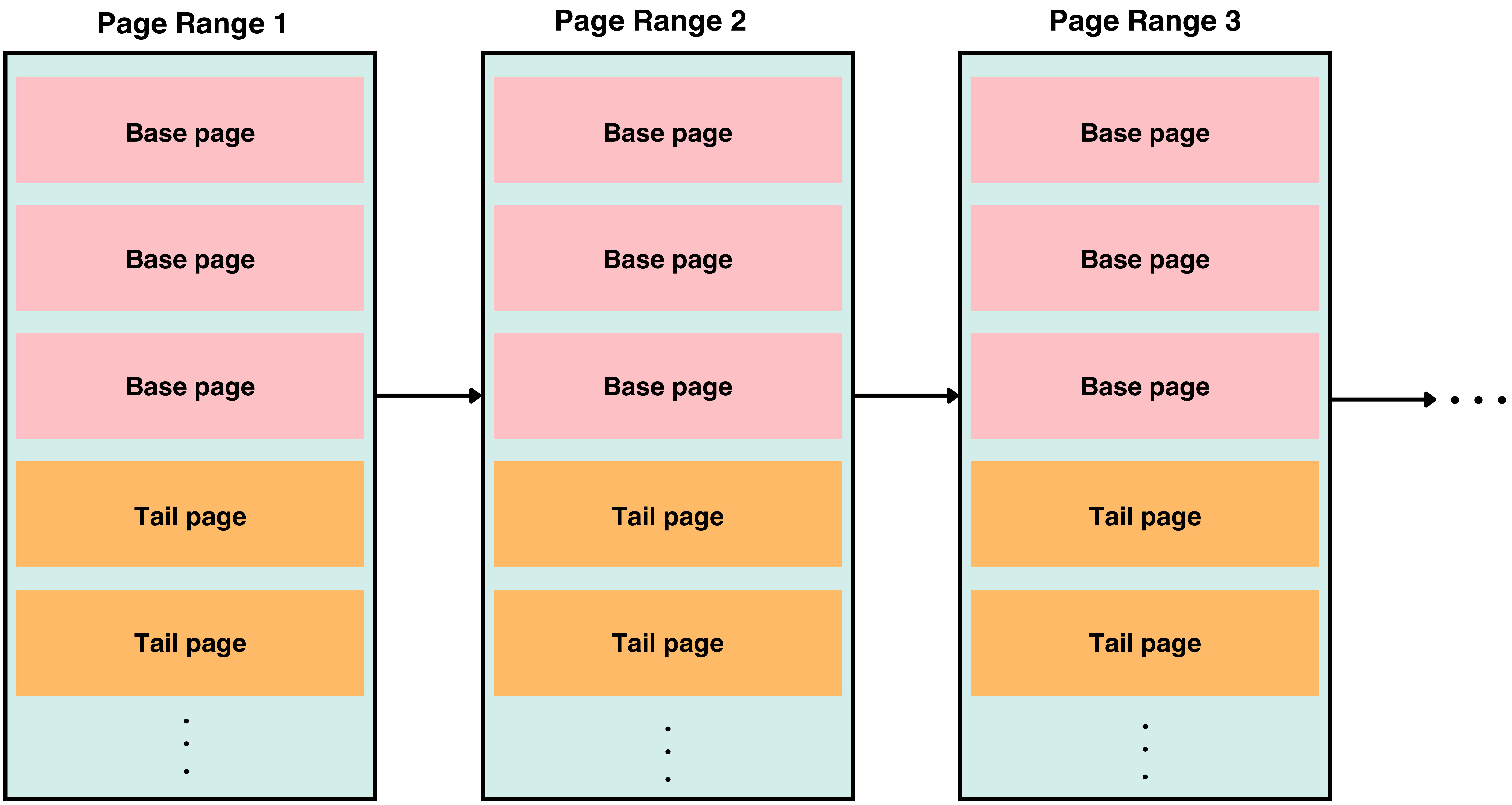
# ECS 165A Milestone 1

Aadvika Ahuja, Manvi Nawani, Veda Periwal, Neerja Natu, Rahul Lakshmanan, Vibha Raju

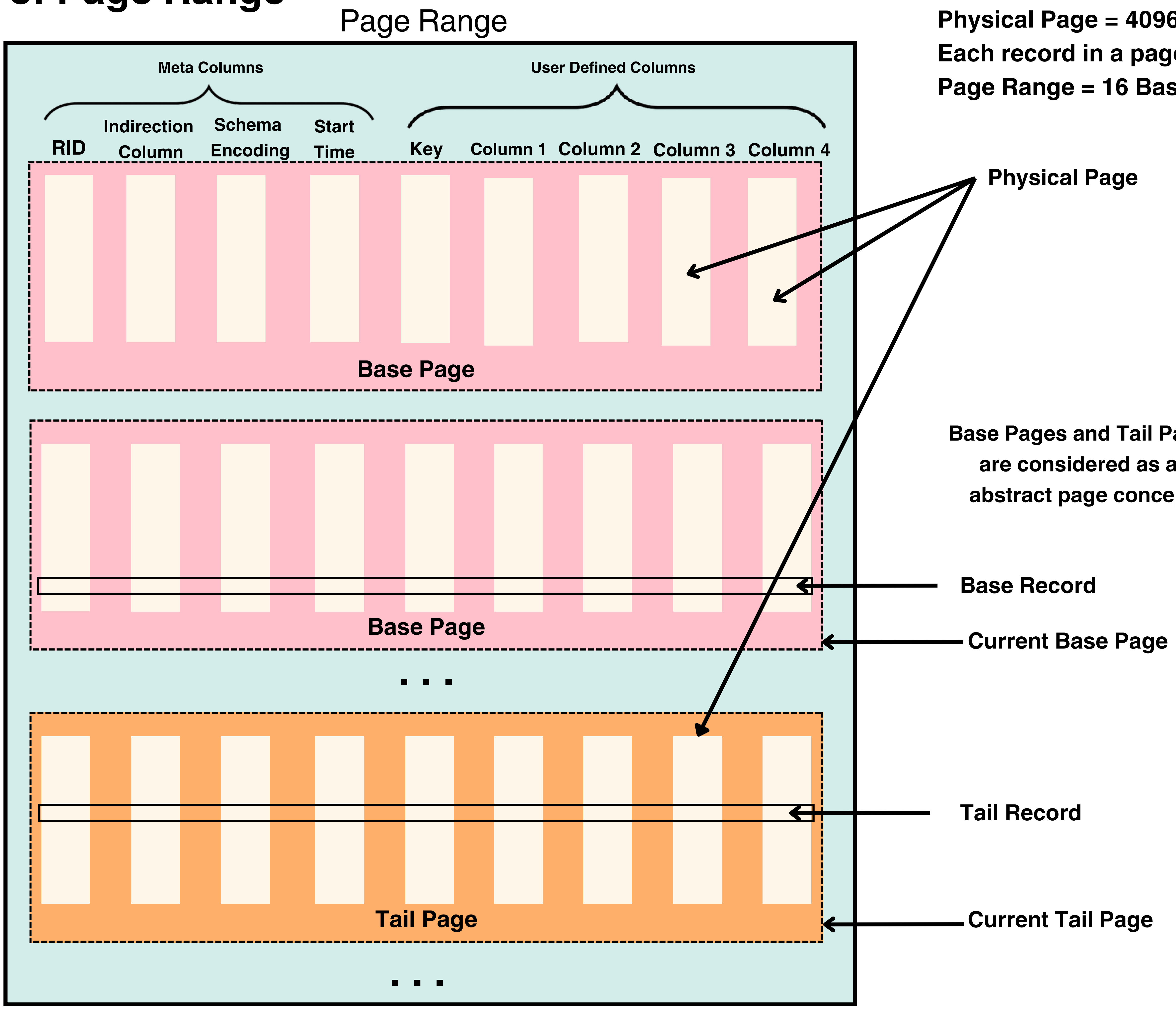
## Outline

## 1. Data Model Design

# High Level Structure of the Database



## Structure of Page Range



**Physical Page = 4096 KB**
**Each record in a page = 8 bytes**
**Page Range = 16 Base Pages**

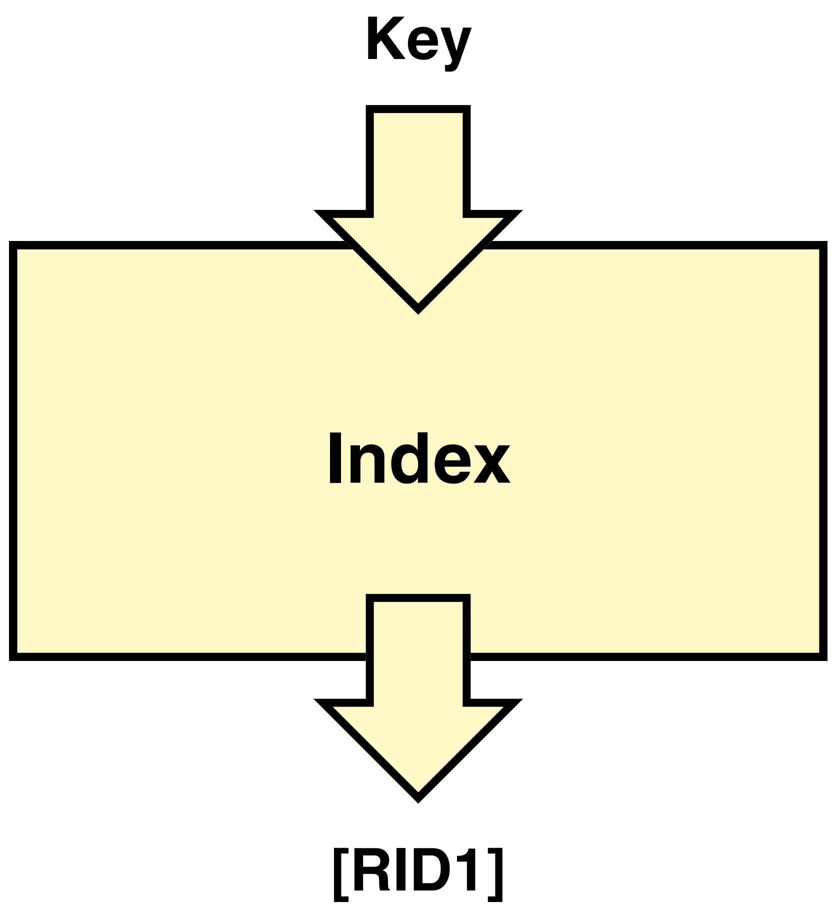
**Base Pages and Tail Pages are considered as an abstract page concept.**

## 2. Bufferpool

Indexing and page directory

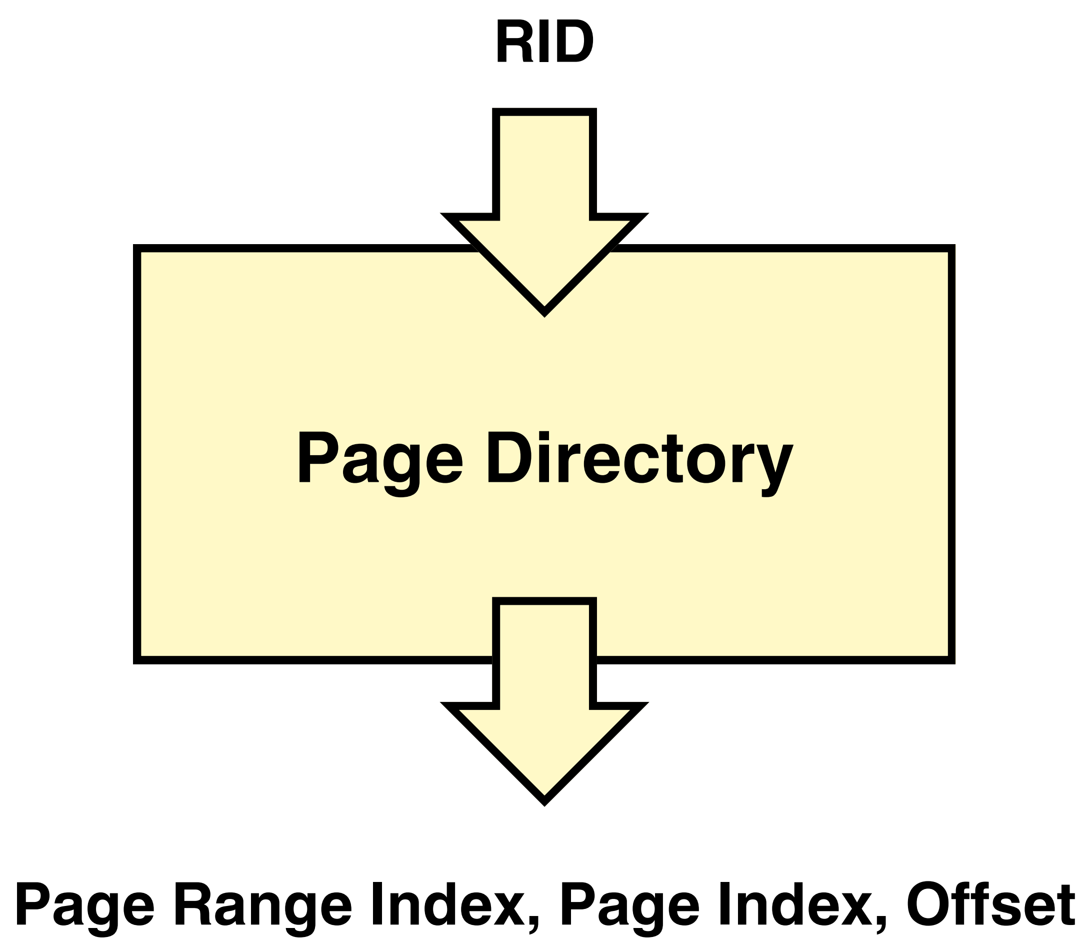
## Index Class & BTree

Using BTree to map primary key to corresponding RIDs.
For M1, primary key is unique so each key only maps to one RID.



## Page Directory & Dictionary

Mapping the RID of a record to the exact location in the database which consists of the page range number, page number, and offset.



## 3. Queries

insert, select, update, delete, sum

# Insert Base Records

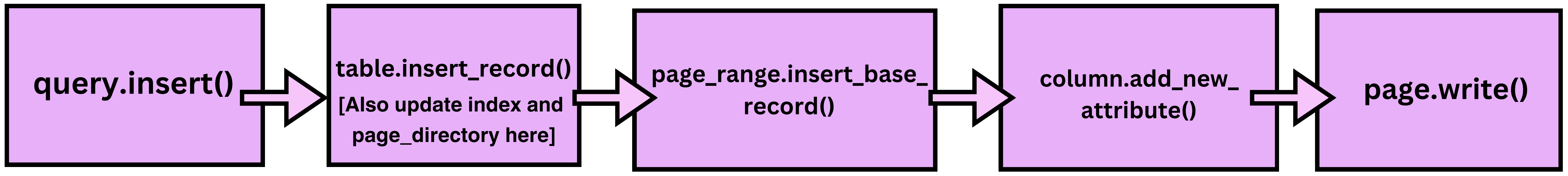
| RID | Indirection Column | Schema Encoding | Start Time | Key | Column 1 | Column 2 | Column 3 | Column 4 |
|---|---|---|---|---|---|---|---|---|
| 1 | None | 00000 | 10:04 | k1 | 10 | 2 | 16 | 20 |
| 2 | None | 00000 | 10:04 | k2 | 34 | 23 | 4 | 3 |
| 3 | None | 00000 | 10:08 | k3 | 11 | 12 | 15 | 9 |
| 4 | None | 00000 | 11:11 | k4 | 8 | 14 | 22 | 19 |

In Index: key k1 <-> RID 1
Page Directory:
Page Range: 0
Page: 0
Offset: 0

In Index: key k2 <-> RID 2
Page Directory:
Page Range: 0
Page: 0
Offset: 8

In Index: key k3 <-> RID 3
Page Directory:
Page Range: 0
Page: 0
Offset: 16

In Index: key k4 <-> RID 4
Page Directory:
Page Range: 0
Page: 0
Offset: 24


query.insert()
table.insert_record()
[Also update index and page_directory here]
page_range.insert_base_record()
column.add_new_attribute()
page.write()

# Update: Cumulative Tail Records

**Base page**

| RID | Indirection Column | Schema Encoding | Start Time | Key | Column 1 | Column 2 | Column 3 | Column 4 |
|---|---|---|---|---|---|---|---|---|
| 1 | t3 | 00000 | 11:45 | k1 | 10 | 2 | 16 | 20 |
| 2 | t4 | 01001 | 11:15 | k2 | 34 | 23 | 4 | 3 |
| 3 | None | 00000 | 10:08 | k3 | 11 | 12 | 15 | 9 |
| 4 | t2 | 01111 | 11:22 | k4 | 8 | 14 | 22 | 19 |

**Tail page**

| RID | Indirection Column | Schema Encoding | Start Time | Key | Column 1 | Column 2 | Column 3 | Column 4 | |
|---|---|---|---|---|---|---|---|---|---|
| t1 | **2** | 01000 | 11:15 | k2 | **17** | 23 | 4 | 3 | |
| t2 | **4** | 01111 | 11:22 | k4 | **9** | **32** | **14** | **11** | → **Latest version of record with k4** |
| t3 | **1** | 01010 | 11:45 | k1 | **13** | 2 | **22** | 20 | → **Latest version of record with k1** |
| t4 | **t1** | 01001 | 11:47 | k2 | 17 | 23 | 4 | **7** | → **Latest version of record with k2** |

## Update: Cumulative Tail Records

Cumulative tail records refer to the fact that for every update we make, the corresponding tail record carries the latest values for all columns

**Original Version of Record (in Base Page)**

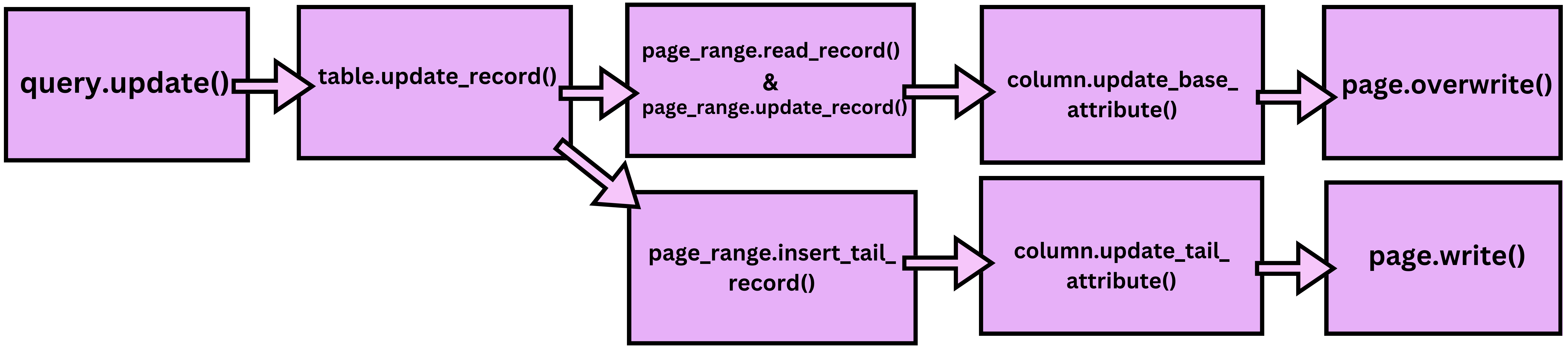
[919432, 4, 5, 4, 3]

**First Modification (in Tail Page)**

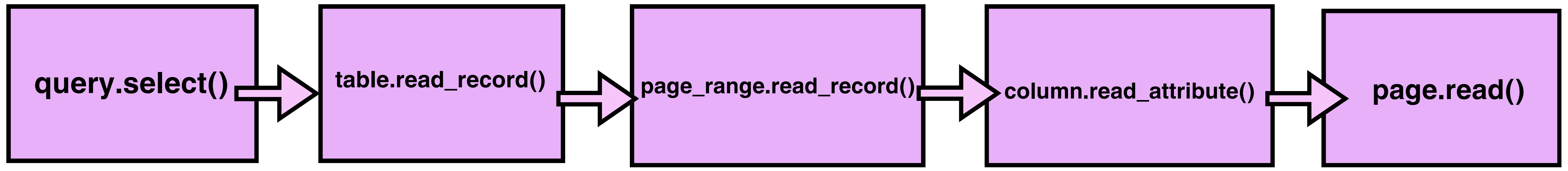
[919432, 4, 4, 4, 3]

**Second Modification (in Tail Page)**

[919432, 4, 4, 4, 4]

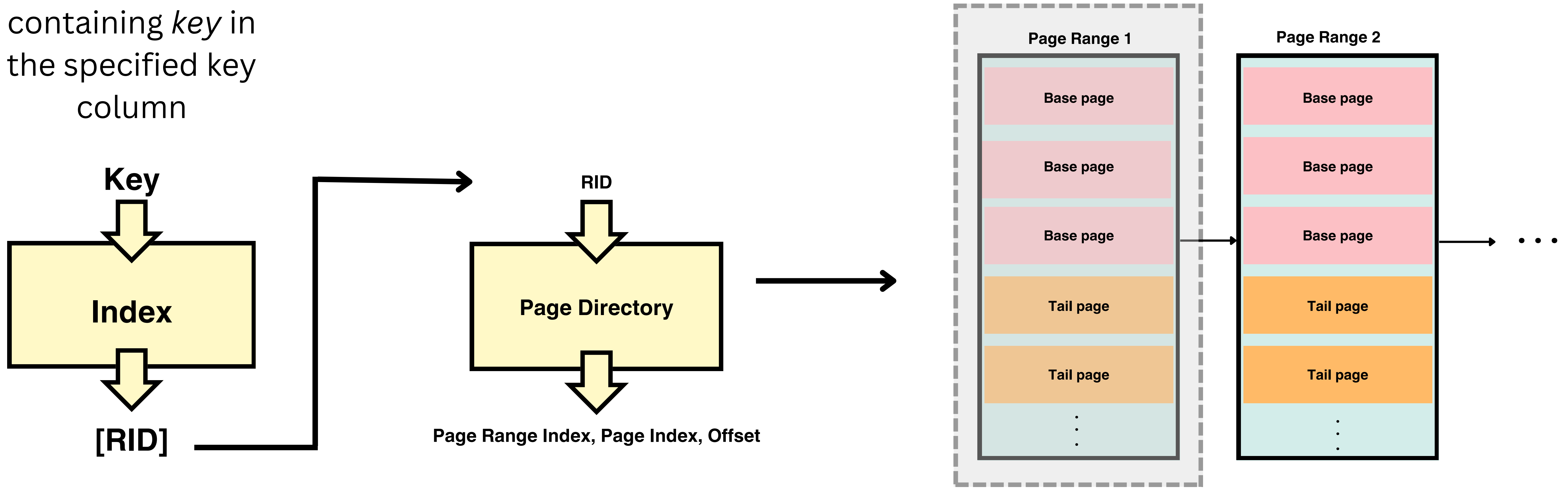

query.update()
table.update_record()
page_range.read_record() & page_range.update_record()
column.update_base_attribute()
page.overwrite()
page_range.insert_tail_record()
column.update_tail_attribute()
page.write()

## Select



### Step 1: Get the Page Range

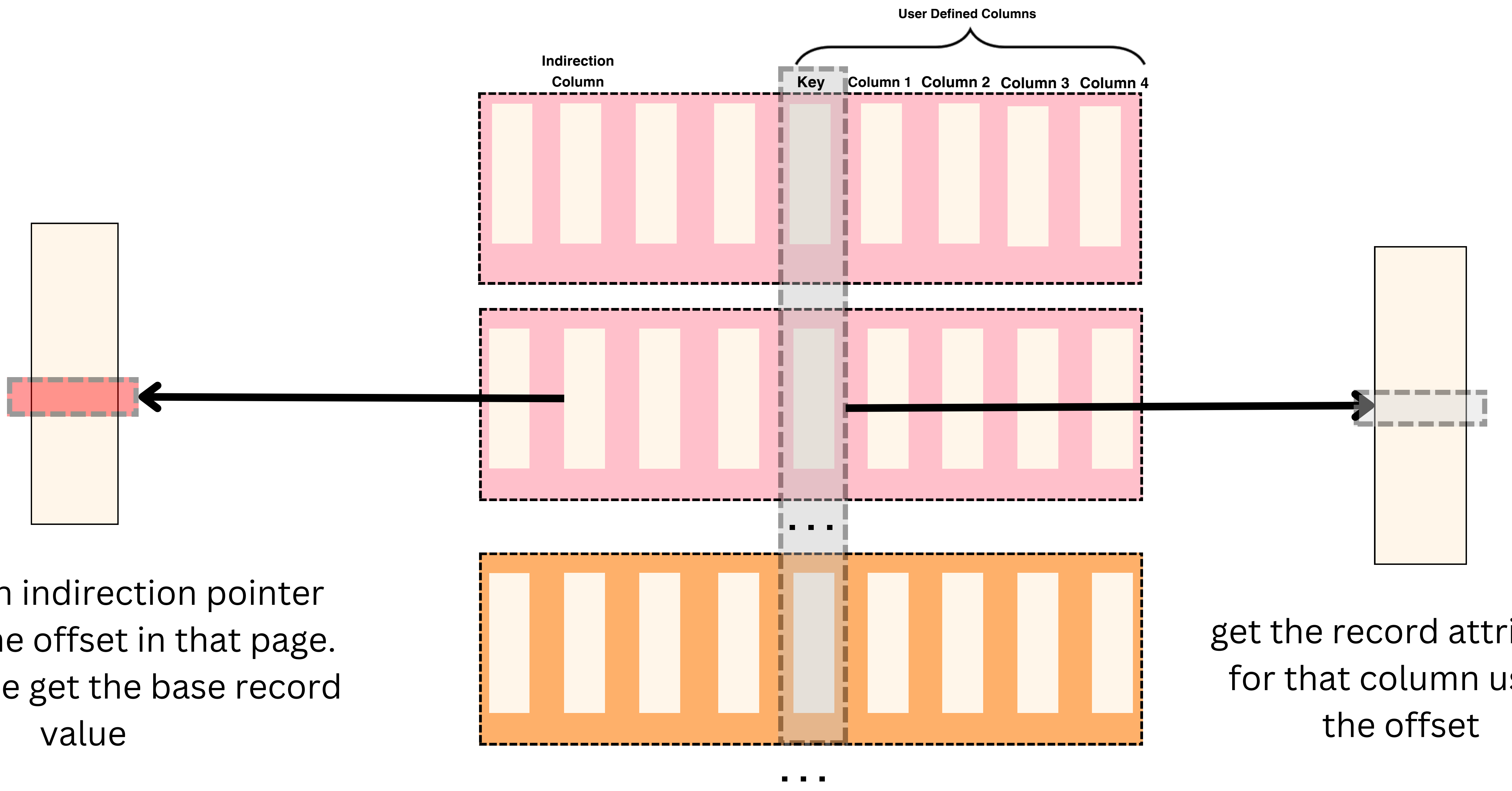
Get RIDs of records containing *key* in the specified key column



# Select

## Step 2: Retrieve the Record

## Case 1: where there are no tail records for the record we are selecting



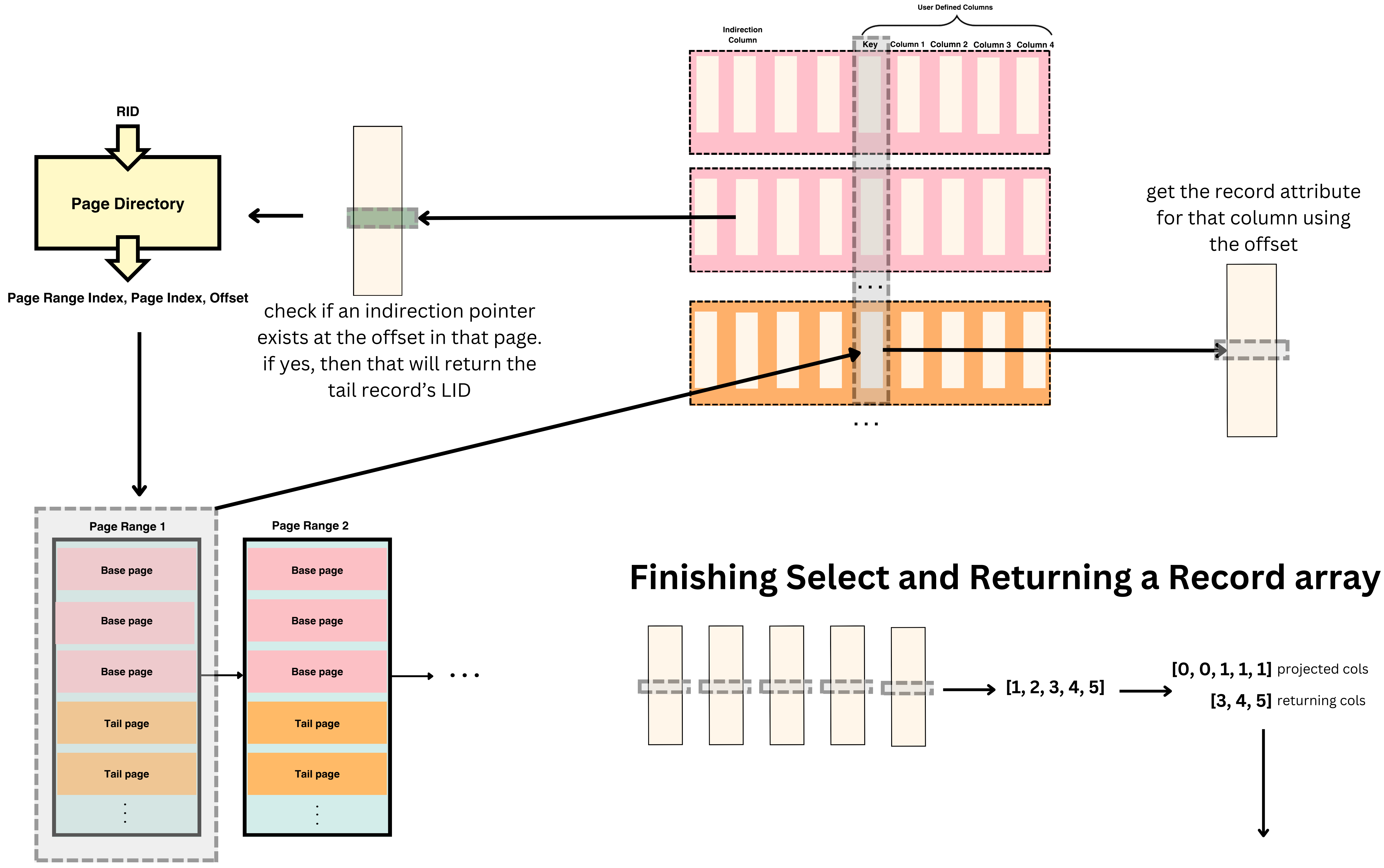
check if an indirection pointer exists at the offset in that page. since no, we get the base record value

get the record attribute for that column using the offset

for each column in the record

# Select

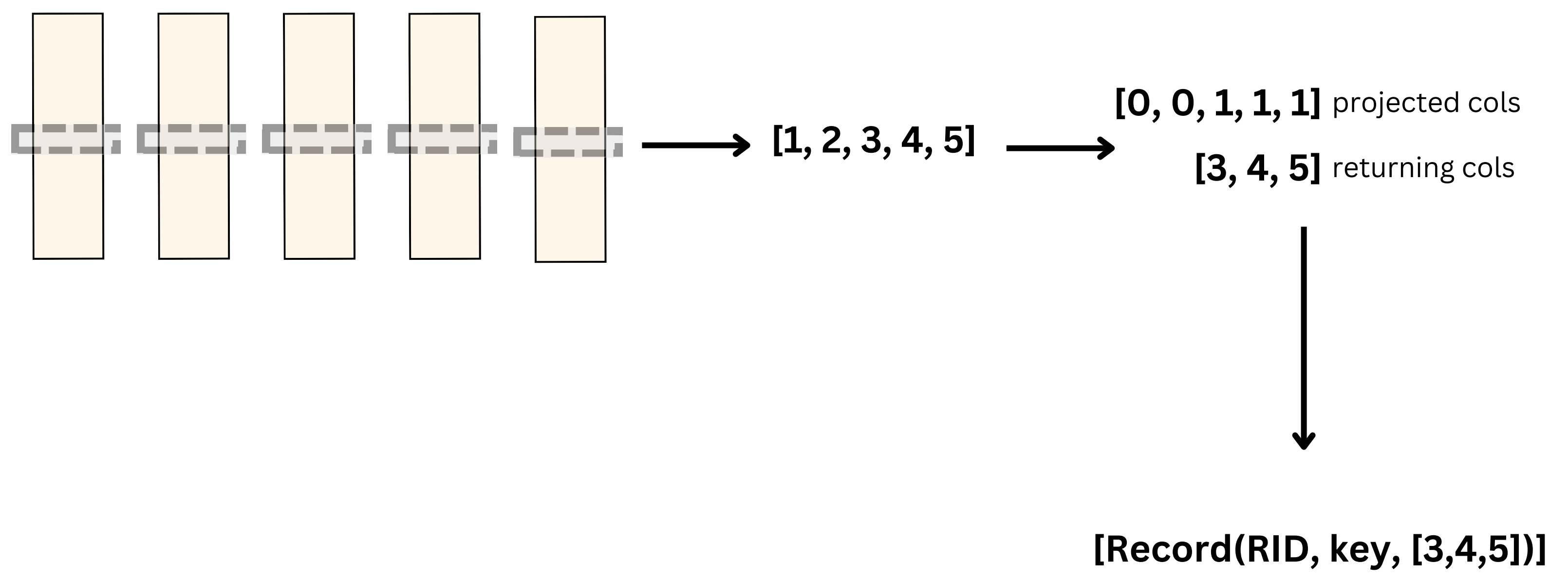
## Case 2: where there are tail records for the record we are selecting



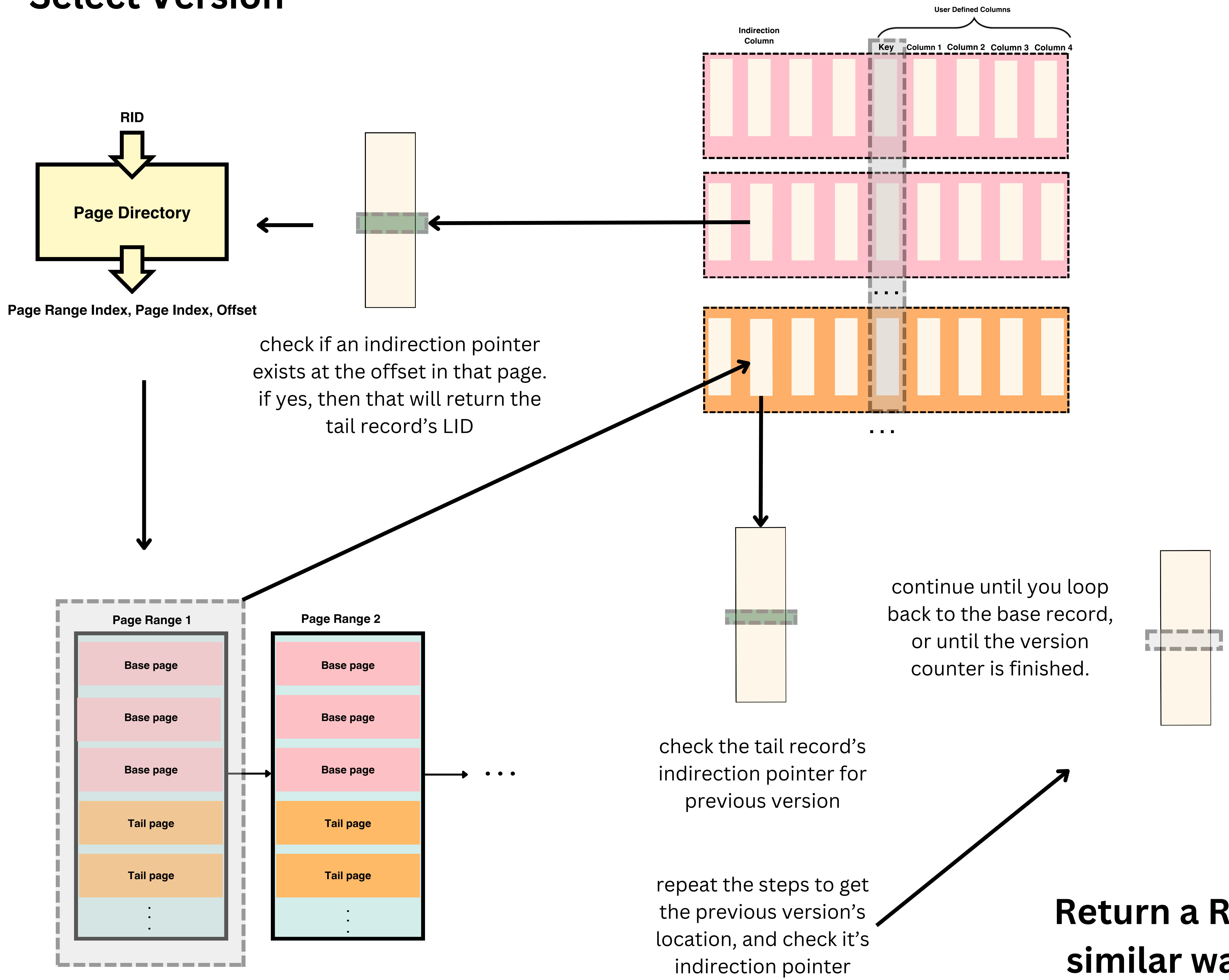
check if an indirection pointer exists at the offset in that page. if yes, then that will return the tail record's LID

get the record attribute for that column using the offset

## Finishing Select and Returning a Record array



# Select Version



check if an indirection pointer exists at the offset in that page. if yes, then that will return the tail record's LID

check the tail record's indirection pointer for previous version

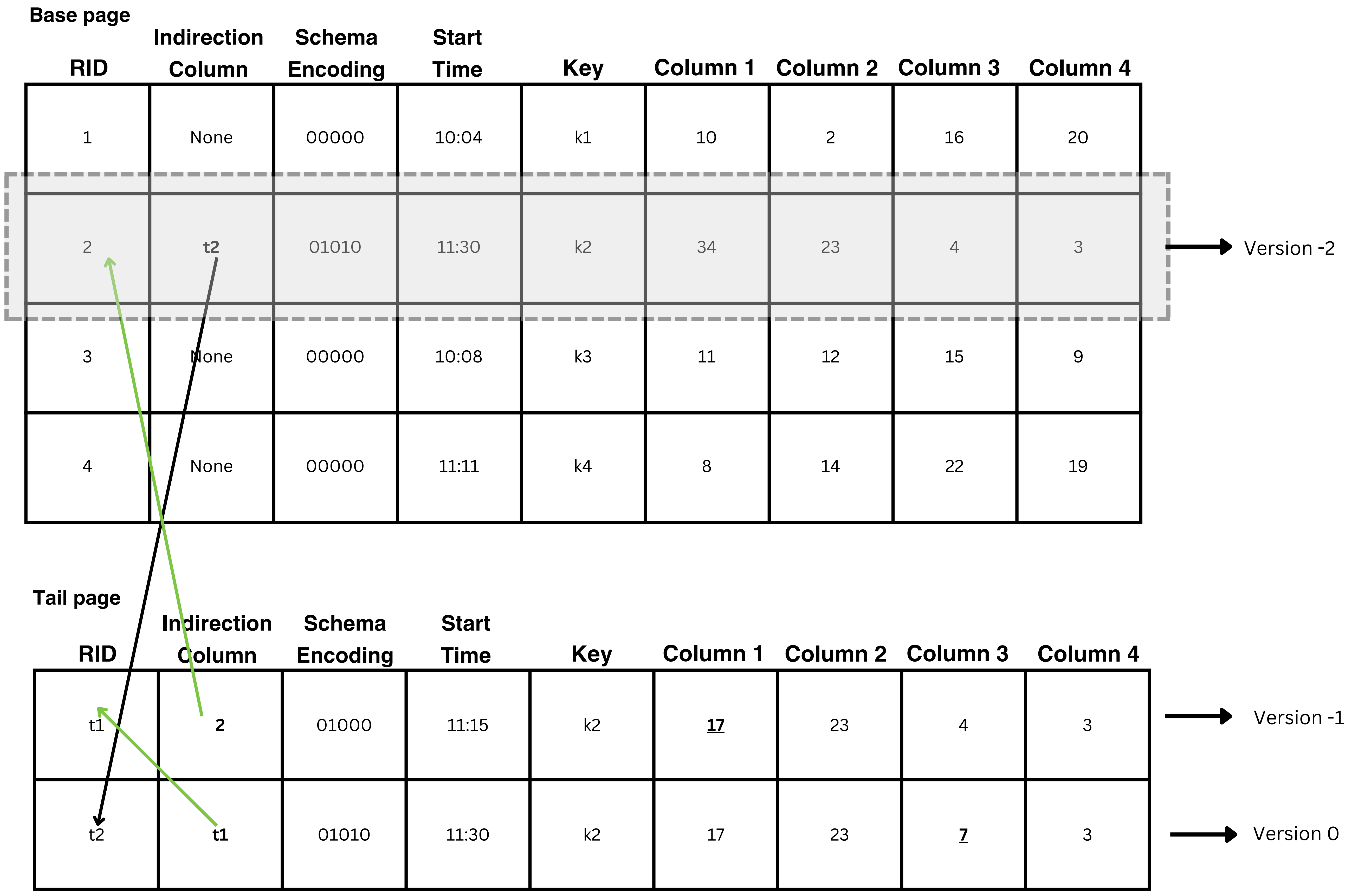
repeat the steps to get the previous version's location, and check it's indirection pointer

continue until you loop back to the base record, or until the version counter is finished.

**Return a Record array in similar way as in select**

# Example: Select Version

## Find Version -2 for k2



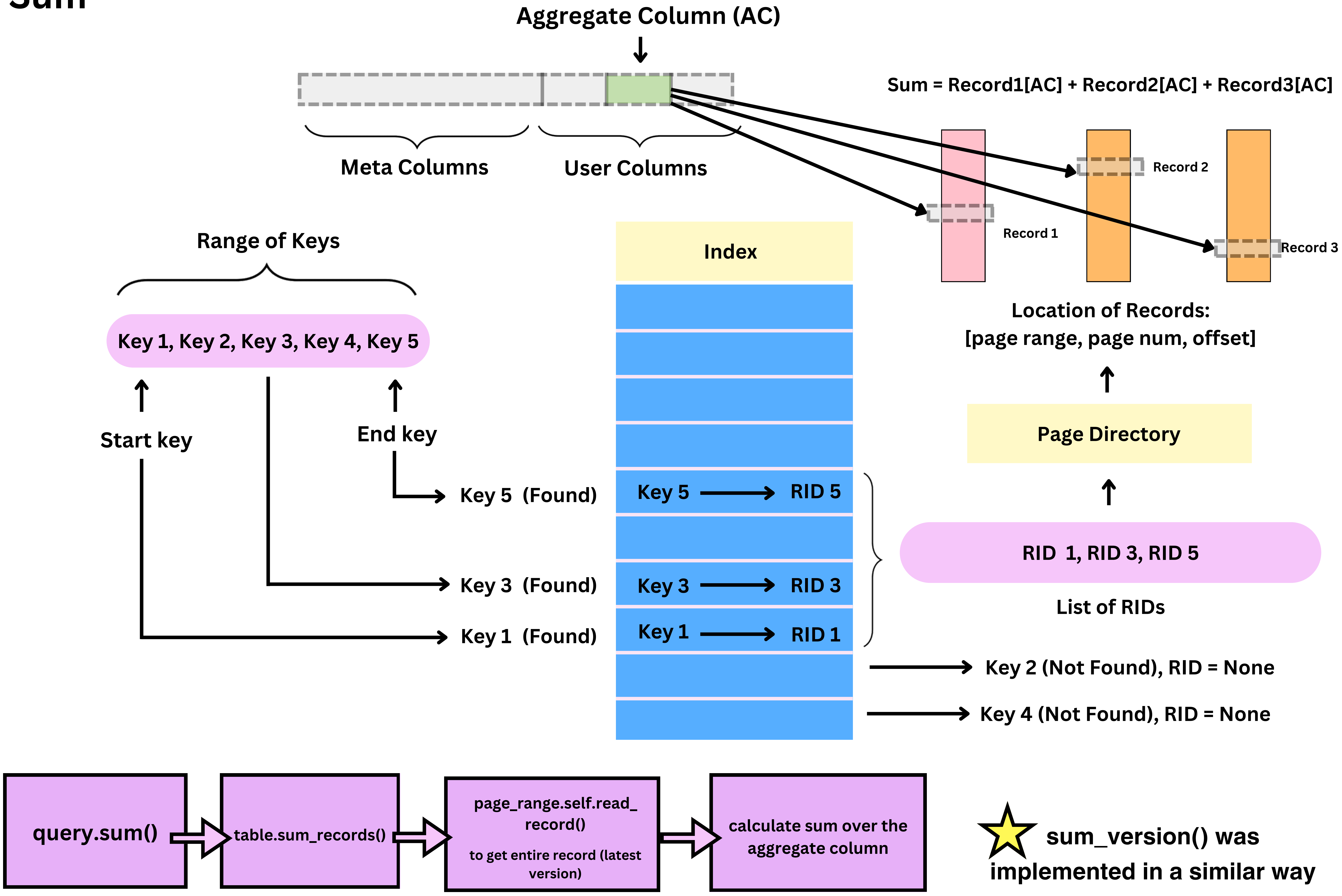
**Base page**

| RID | Indirection Column | Schema Encoding | Start Time | Key | Column 1 | Column 2 | Column 3 | Column 4 |
|---|---|---|---|---|---|---|---|---|
| 1 | None | 00000 | 10:04 | k1 | 10 | 2 | 16 | 20 |
| 2 | **t2** | 01010 | 11:30 | k2 | 34 | 23 | 4 | 3 |
| 3 | None | 00000 | 10:08 | k3 | 11 | 12 | 15 | 9 |
| 4 | None | 00000 | 11:11 | k4 | 8 | 14 | 22 | 19 |

**Tail page**

| RID | Indirection Column | Schema Encoding | Start Time | Key | Column 1 | Column 2 | Column 3 | Column 4 |
|---|---|---|---|---|---|---|---|---|
| t1 | **2** | 01000 | 11:15 | k2 | **17** | 23 | 4 | 3 |
| t2 | **t1** | 01010 | 11:30 | k2 | 17 | 23 | **7** | 3 |

# Sum

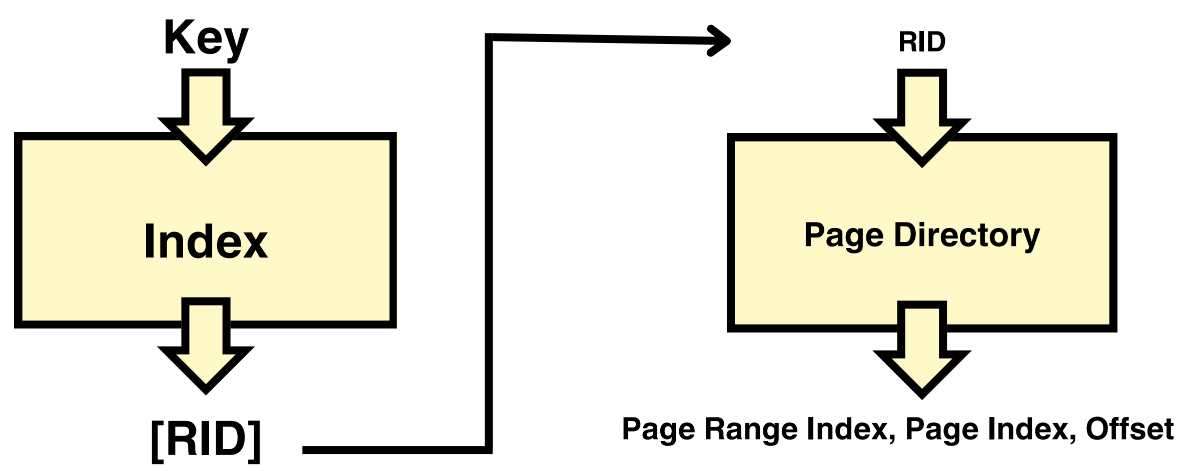


sum_version() was implemented in a similar way

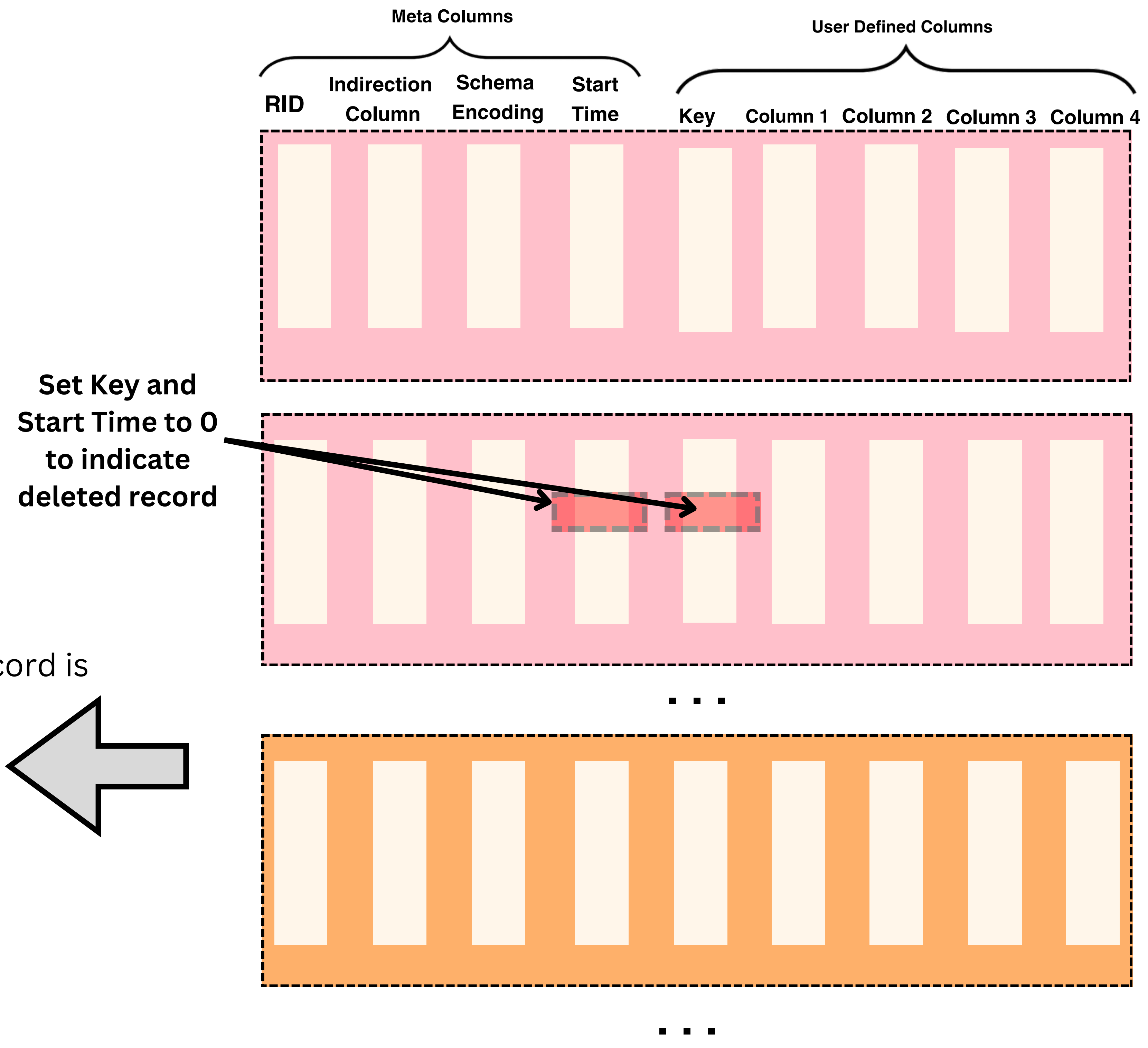
# Delete

## *(Lazy Delete)*

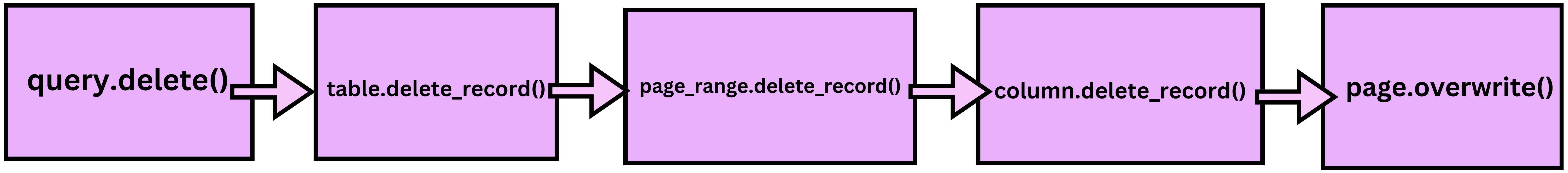
Get RIDs of records containing *key* to be deleted



if RID not found, return false



- Once both key and timestamp have been set to 0, record is marked as deleted.
- Drop key from Index BTree
- Return true.
- Decrement the number of records.



## ☆ Going the extra mile

For this milestone, we decided to also implement versions. This means that for our select() and sum() queries, we created an additional set of functions select_version() and sum_version() that would be able to perform select and sum queries for specific versions

```
select_version(self, search_key, search_key_index, projected_columns_index, relative_version)
```

**takes in additional parameter of version**

```
sum_version(self, start_range, end_range, aggregate_column_index, relative_version)
```

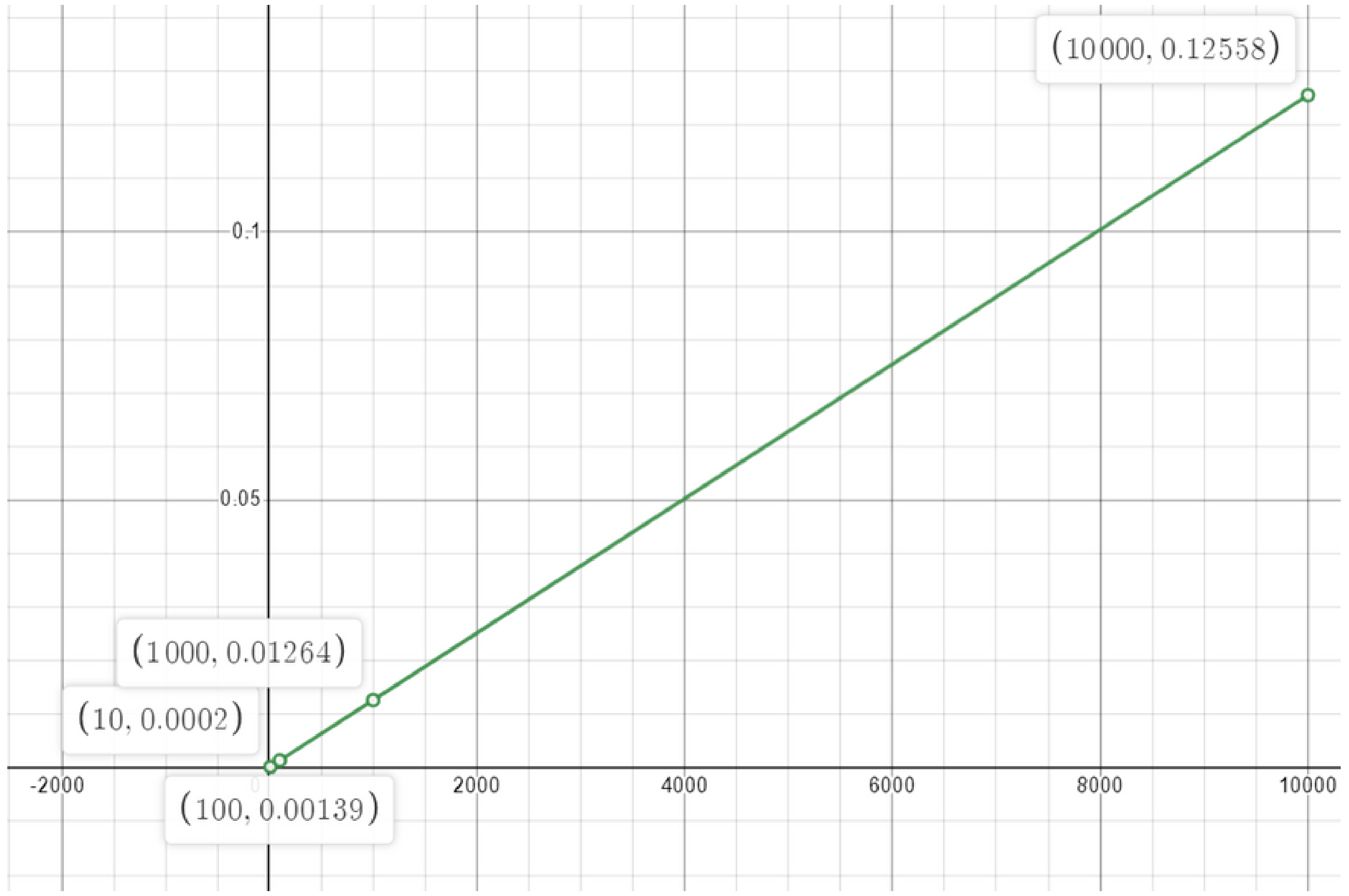
**takes in additional parameter of version**

## 4. Performance

Each query on different data sizes

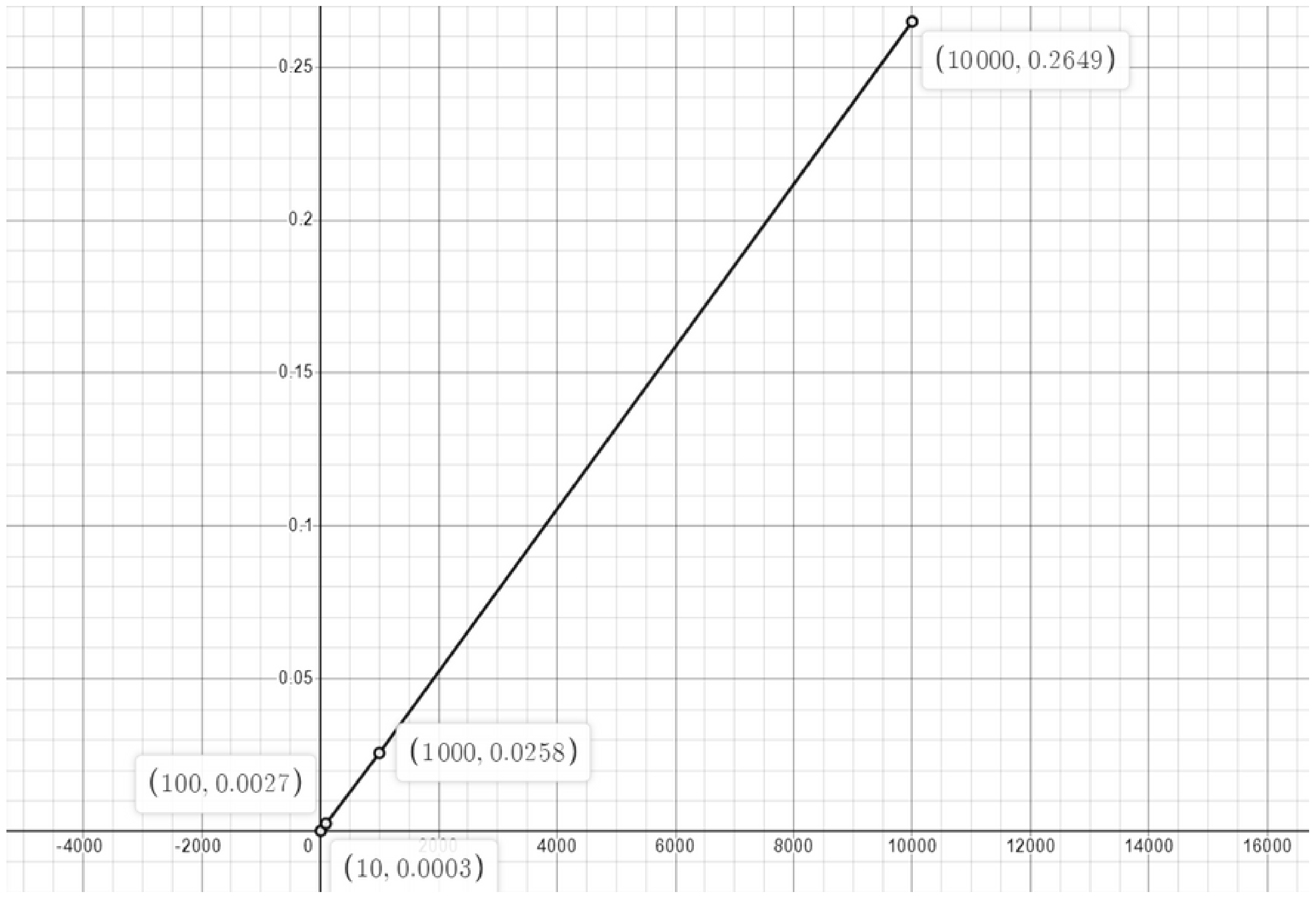
## Performance for each query with different data sizes

### Insert: time vs. # records



| # of records | time |
|---|---|
| 10 | 0.000196 |
| 100 | 0.001392 |
| 1000 | 0.012642 |
| 10000 | 0.125576 |

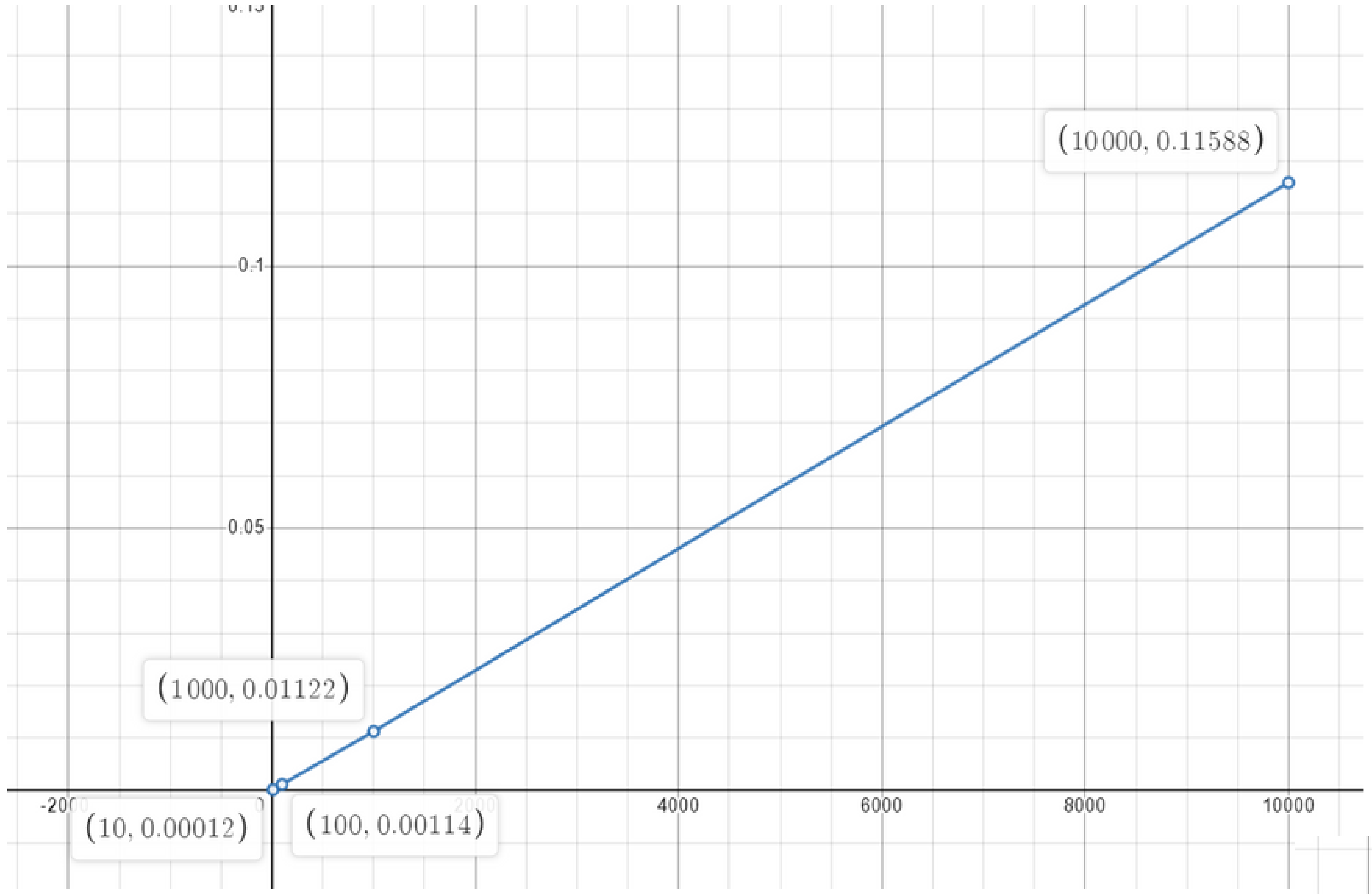
### Update: time vs. # records



| # of records | time |
|---|---|
| 10 | 0.000315 |
| 100 | 0.002705 |
| 1000 | 0.025756 |
| 10000 | 0.264926 |

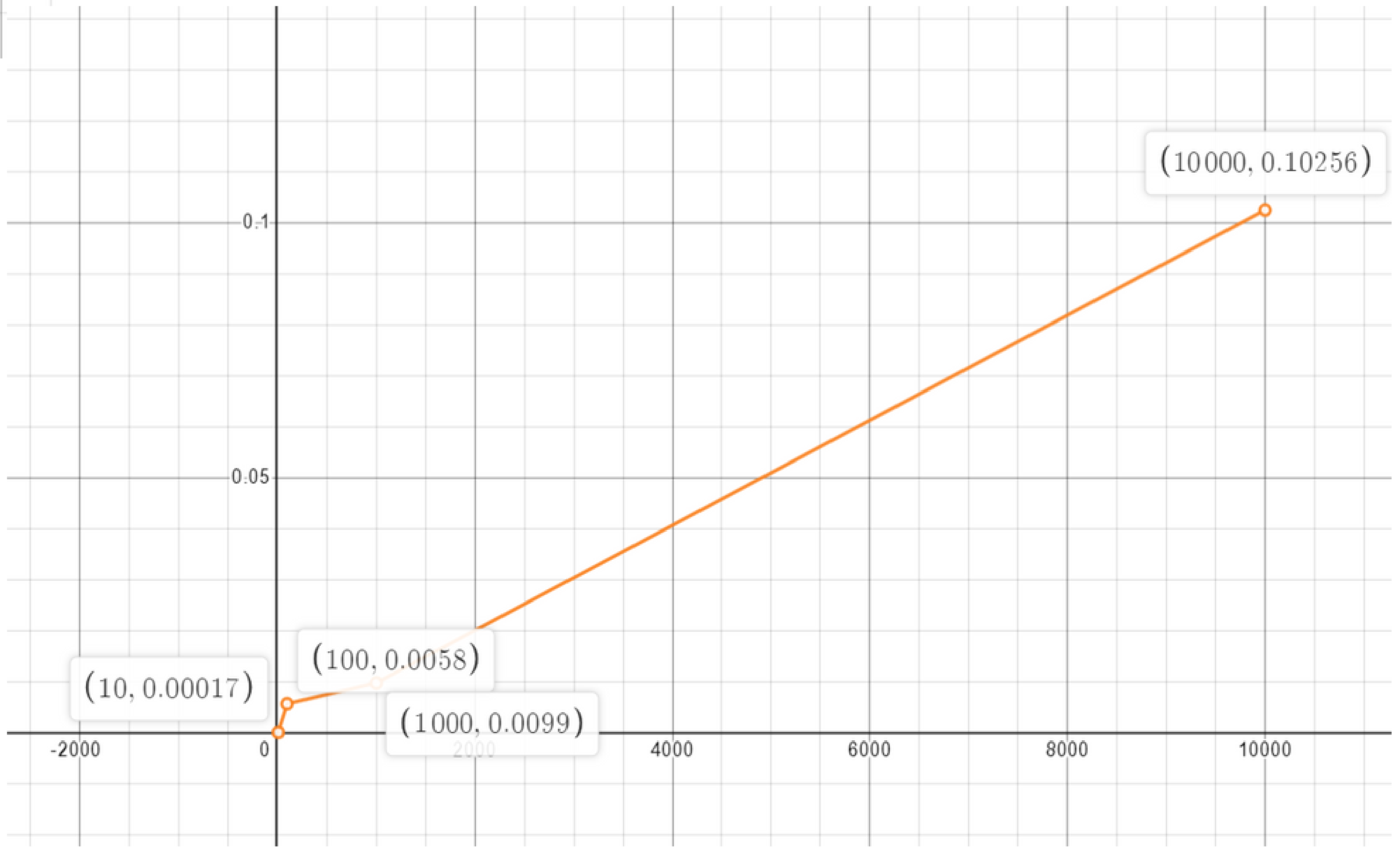
## Performance for each query with different data sizes

**<u>Select</u>: time vs. # records**



| # of records | time |
|---|---|
| 10 | 0.00012 |
| 100 | 0.001138 |
| 1000 | 0.011218 |
| 10000 | 0.115877 |

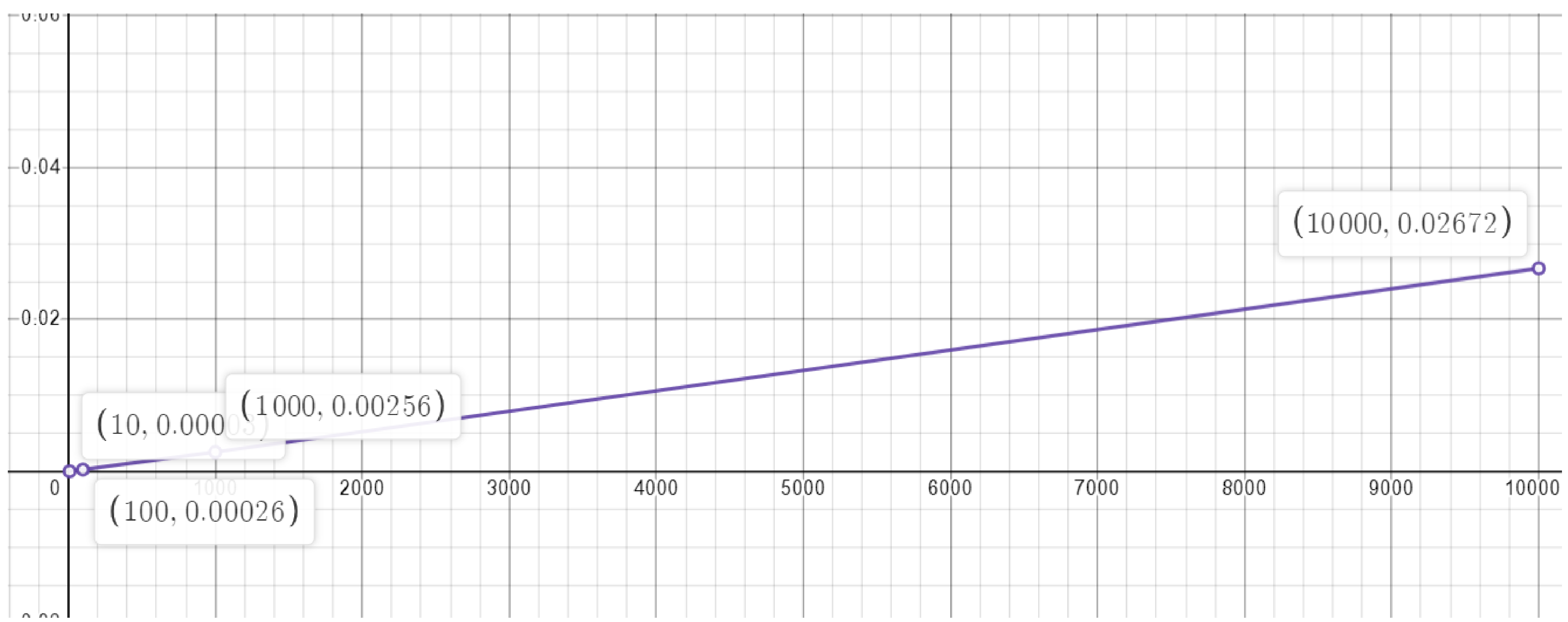
**<u>Sum</u>: time vs. # records**



| # of records | time |
|---|---|
| 10 | 0.000173 |
| 100 | 0.0058 |
| 1000 | 0.0099 |
| 10000 | 0.102556 |

## Performance for each query with different data sizes

### <u>Delete</u>: time vs. # records



| # of records | time |
|---|---|
| 10 | 2.80E-05 |
| 100 | 0.000259 |
| 1000 | 0.002556 |
| 10000 | 0.026722 |