Project4

Multiple Column & Join Table

Before doing project 4..

- We allow you to use C++ features, not only C.

- Whether you use C or C++, you need to modify the name of existing delete API.
  - int ~~delete~~ (int table_id, int64_t key);
  - **int remove (int table_id, int64_t key);**

Multiple column

- Your database only supports storing a key and a value in a single record.
- Our first goal is to implement a multiple column record.

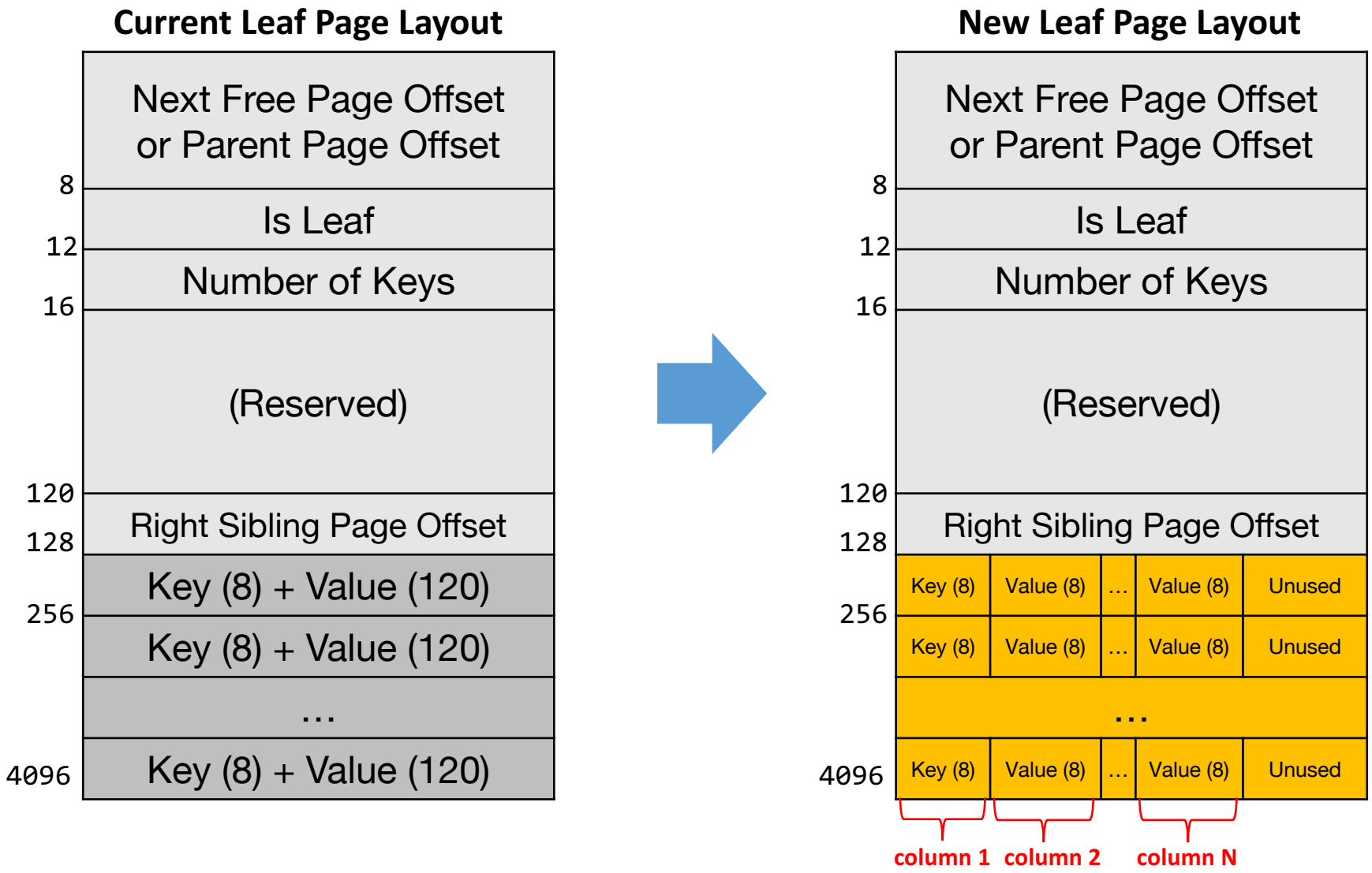
Need to change APIs

1. int open_table (char * pathname, **int num_column**);
   - If 'pathname' data file exist, open it and ignore the num_column parameter.
   - Otherwise, create a new one with **num_column** columns.
   - 2 <= **num_column** <= 16
     - (We suppose there must a key and at least one value in a single record)
2. int insert (int table_id, int64_t key, **int64_t* values**);
   - Insert a record with given key and **an array of values**
   - Example

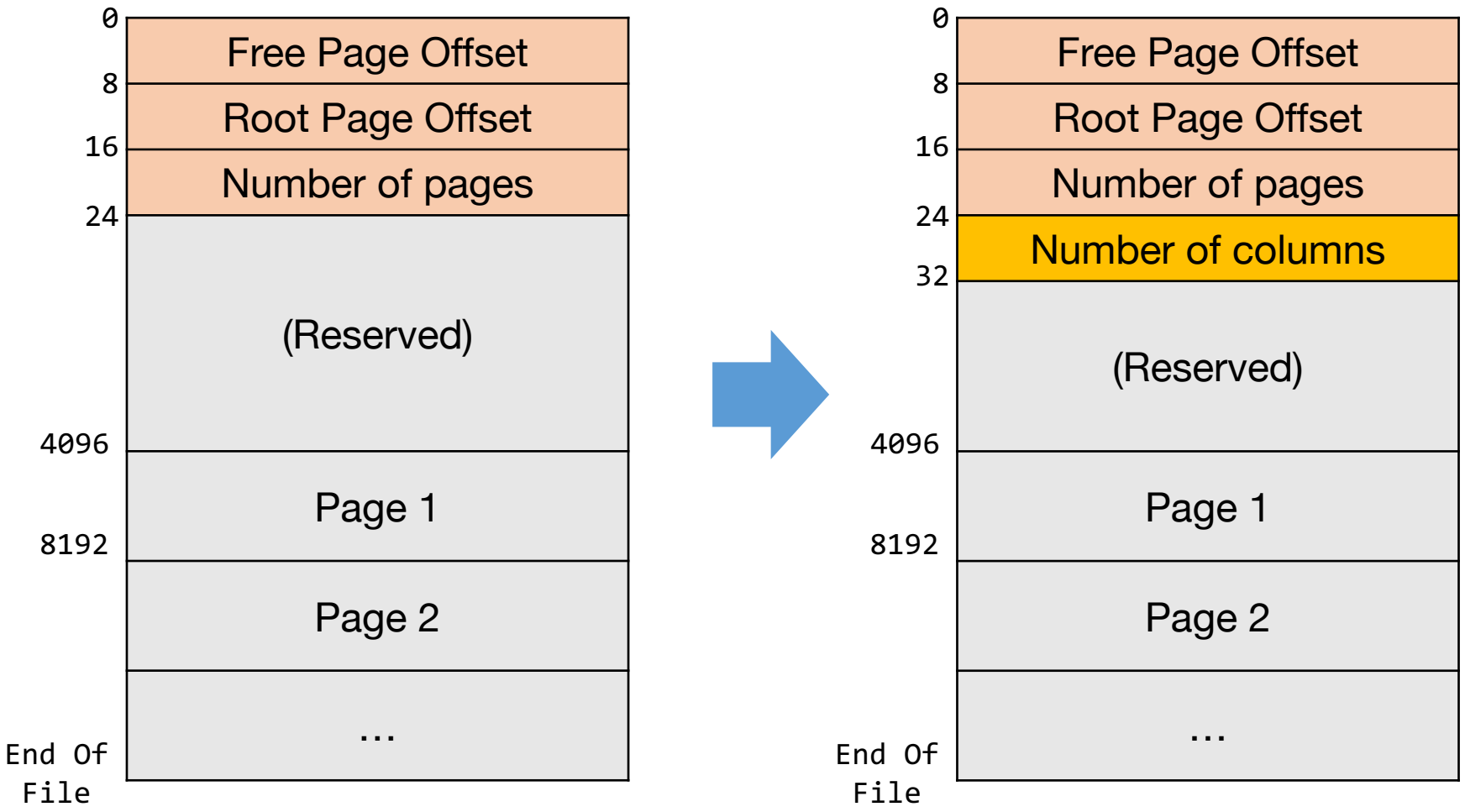
```
int64_t values = (int64_t*)malloc(sizeof(int64_t) * (ncolumn - 1));
for (int i = 0; i < ncolumn – 1; i++) values[i] = 1234;
int ret = insert(tid, key, values);
```

3. **int64_t** * find (int table_id, int64_t key);
   - Returns a pointer of an array of values(do not include a key) if the key is found.

New leaf page format



New header page format



Join Tables

- Your database system doesn’t consider JOIN table operation yet.
- Our second goal is to implement a **JOIN query** that consist of multiple join operations and an aggregation.

Join Tables

- You need to support this API
  - **int64_t *join* (const char* *query*);**
  - *query* format
    - "$T_1.C_1=T_2.C_2$" in case the number of join op. = 1
    - "$T_1.C_1=T_2.C_2\&T_3.C_3=T_4.C_4$" in case the number of join op. = 2
    - "$T_1.C_1=T_2.C_2\&T_3.C_3=T_4.C_4\&T_5.C_5=T_6.C_6$" in case the number of join op. = 3
    - Same pattern for the number of join op. > 3
  - Return a sum of all keys in the query result.
    - If there is no result, return 0
  - 1 <= number of join op. <= 9

Join Tables

- Example
  - int64_t *ret* = *join*("1.1=2.2&2.3=3.2"); // *ret*: 29(1+3+9+3+4+9)

table_id *column_id*

table_id = 1

key	value
1	5
2	3
3	11
4	8
5	7

column_id=1

⋈

table_id = 2

key	value	value
1	6	4
3	3	2
5	2	8
7	8	9
9	1	6

column_id=2 *column_id=3*

⋈

table_id = 3

key	value
2	3
4	6
5	10
7	4
9	2

column_id=2

➡

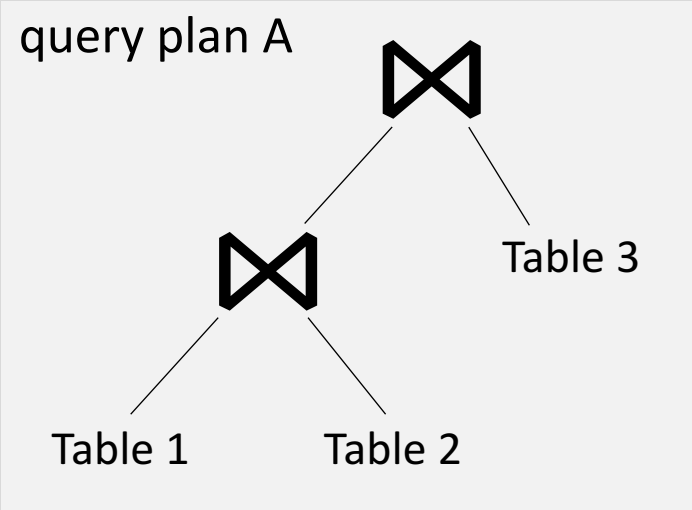
Join Result

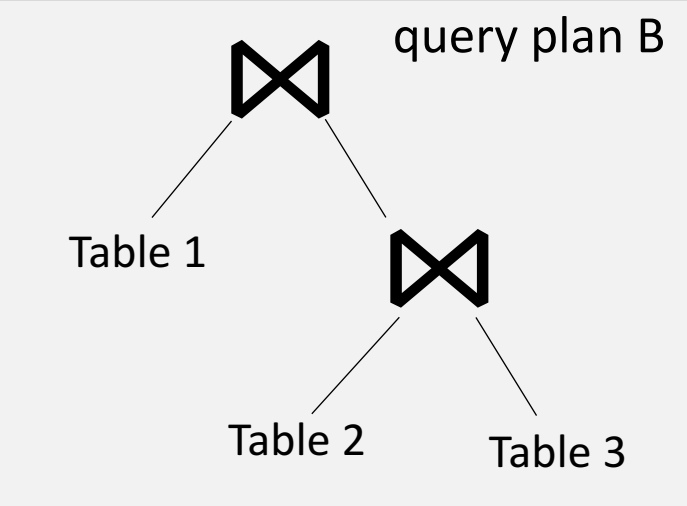
key	value	key	value	value	key	value
1	5	**9**	1	6	**4**	6
3	11	**3**	3	2	**9**	2

Join Tables

- Example
  - All join graphs are **connected**. No cross product.
  - All graphs are **acyclic** and there will be **no self join**.
    - [cyclic graph example: ”1.1=2.1&2.2=1.2”] [self join example: “1.1=1.2”]
  - You can select a query plan yourself.

int64_t *ret* = *join*(“1.1=2.2&2.3=3.2”);





Either way is fine.

Tip

- We will test your project by
  1. Calling *open_table* API multiple times to open all test tables.
  2. Calling join API multiple times with different queries and measure the time spent.
- You can preprocess in the *open_table* API.
  - Scanning opened table, gathering some statistics, … will be fine.
  - Timeout for *open_table* call is **10 sec**.
    - It will be enough time to scan any database file on disk.
- You can use multiple threads for this project.
  - Our test machine spec will be announced later.