Database Processing
CS 451 / 551

Lecture 7:

LSM Trees

Suyash Gupta
Assistant Professor
Distopia Labs and ONRG
Dept. of Computer Science
(E) suyash@uoregon.edu
(W) gupta-suyash.github.io

UNIVERSITY OF OREGON

Assignment 1 is Out!
Deadline: Today! Oct 28, 2025 at 11:59pm

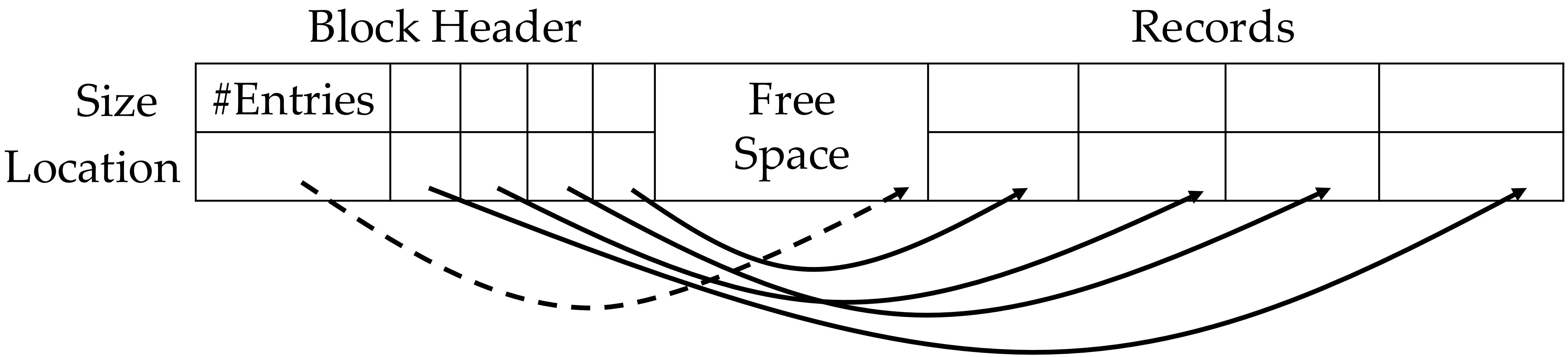
Start collaborating with your groups!

Some Lectures Ago…

- We learnt how data is stored in disks.
- We learnt how records are mapped across pages.

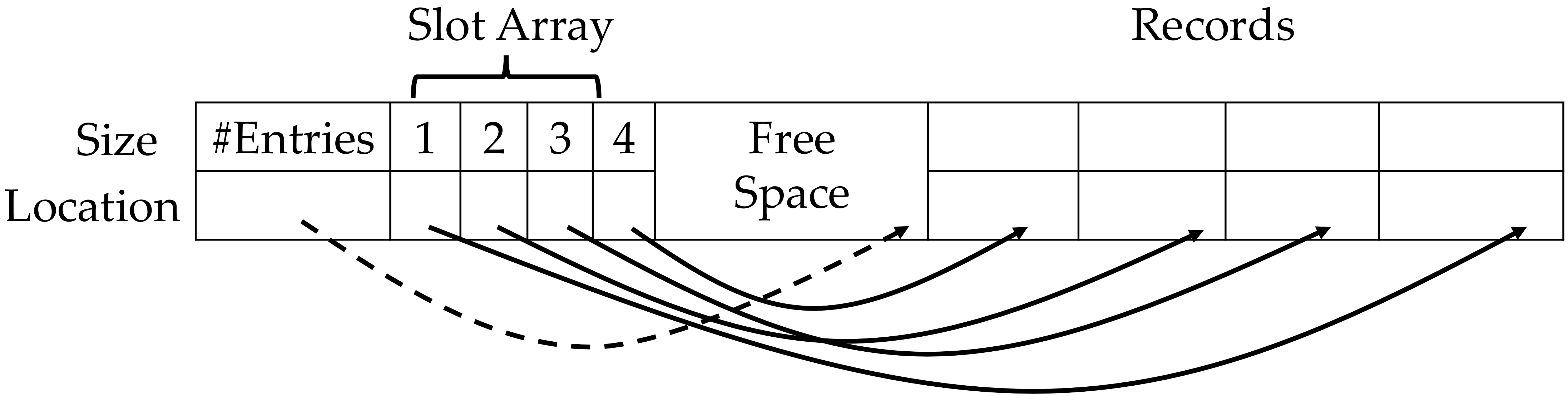
Mapping Records to Disk Blocks

- We saw the following structure for storing records in a page.
- Block header kept track of **records** and the **free space**.



Slotted Pages

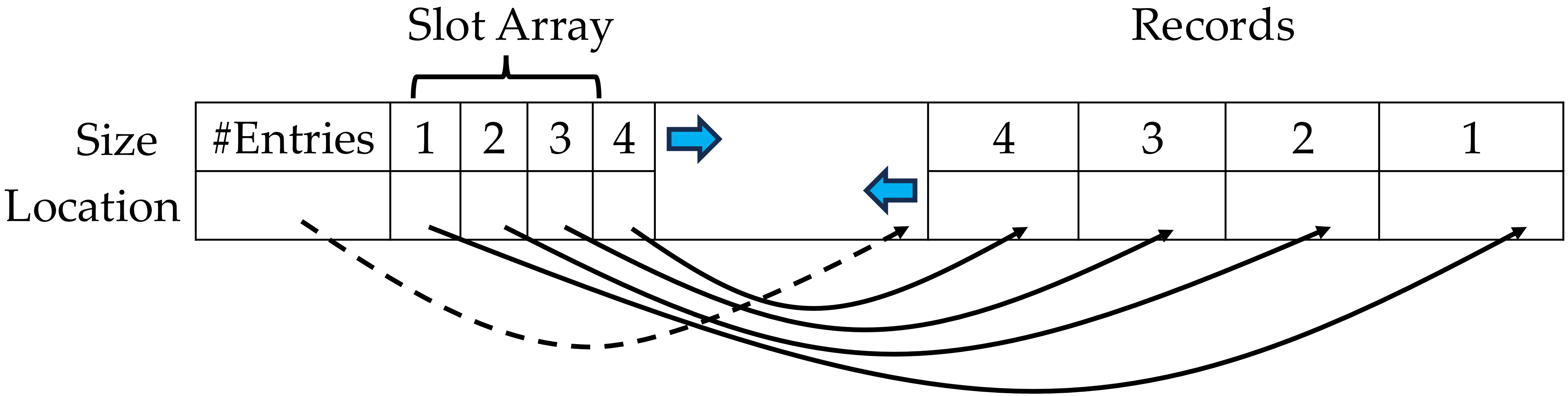
- This layout scheme is also called as **slotted pages**.
- It is one of the most common record storage and tracking scheme.
- These pointer arrows are relative **offsets** to the specific location.



Slotted Pages

- These pointer arrows are relative **offsets** to the specific location.

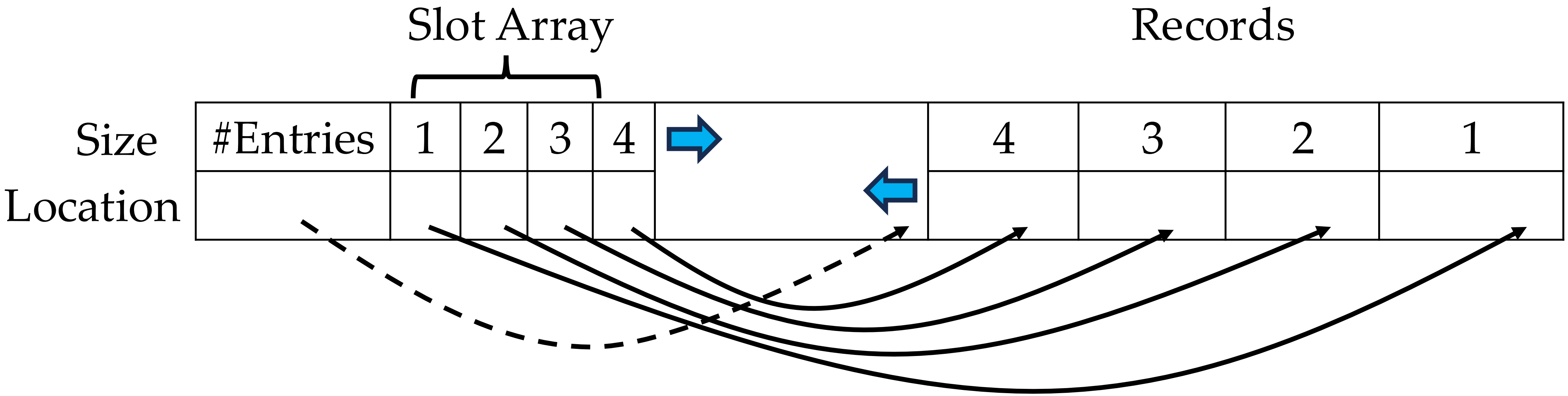
- Blue arrow represents the direction in which slot array and records can grow (use the free space)



Slotted Pages: Record Deletion

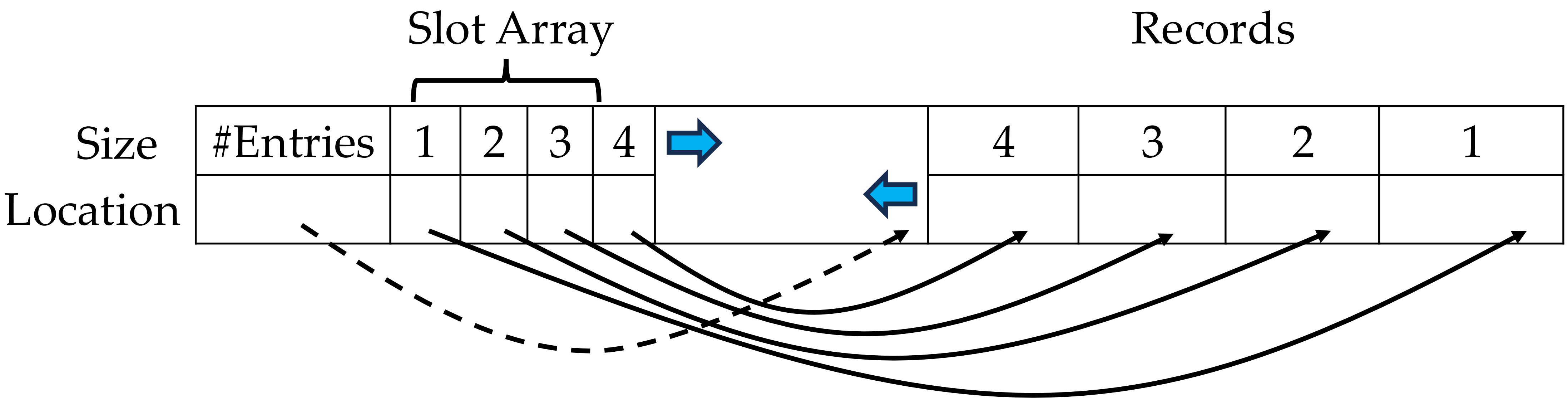
- How to delete a record in this record layout scheme.

- Say, we want to delete record 3. What do you think will happen?



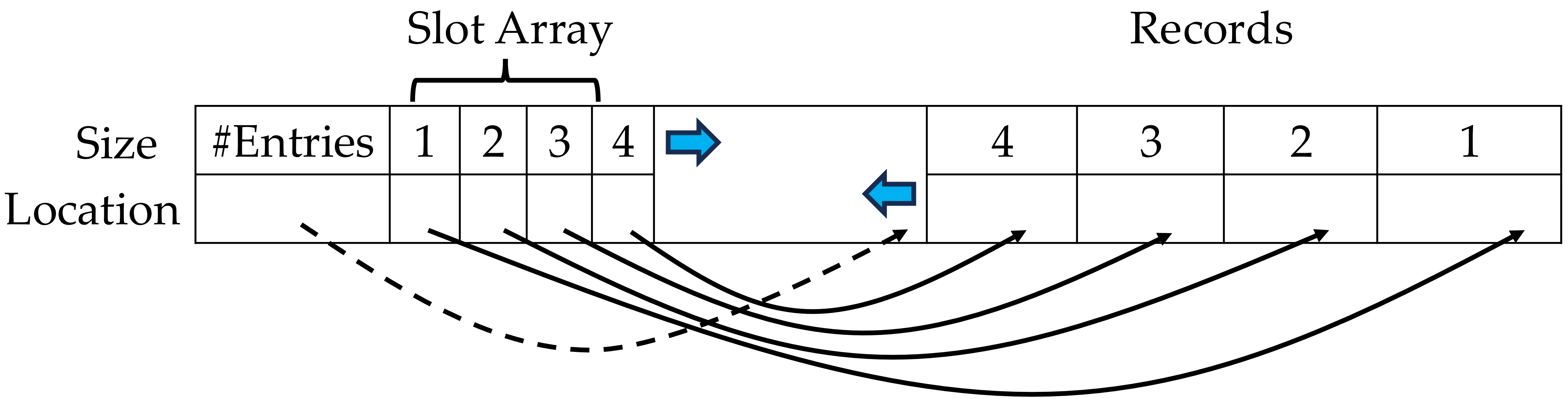
Slotted Pages: Record Deletion

- Generally, following are the steps, which databases take:
  - Delete the record and remove the pointer to the record from the slot array.
  - Reorganize or garbage collect if necessary.
  - "If necessary"? Depends on the database.



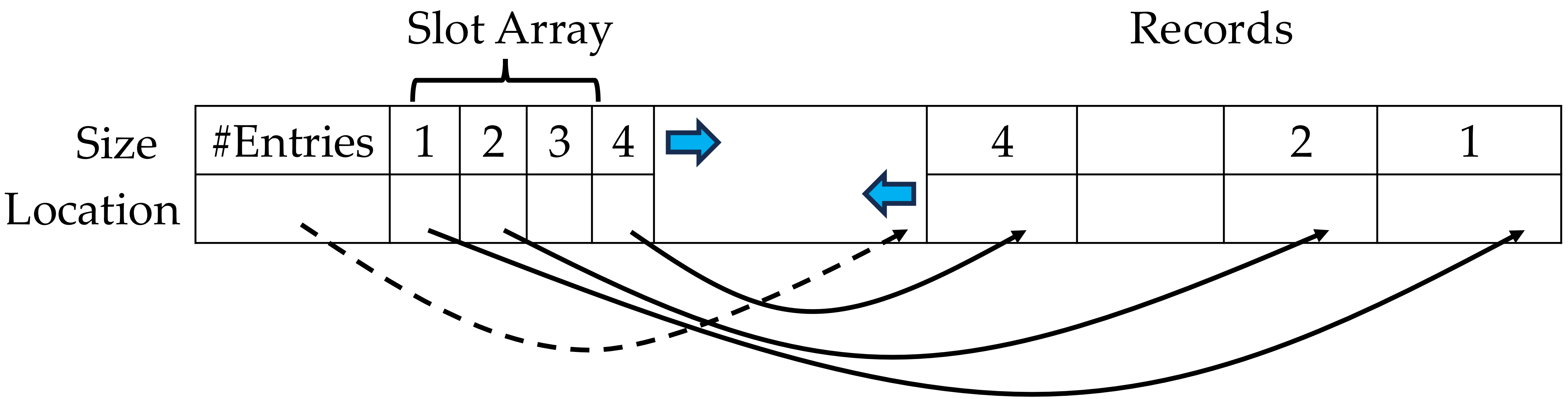
Slotted Pages: Record Deletion

- For example, **in Postgres**, how will record 3 get deleted?



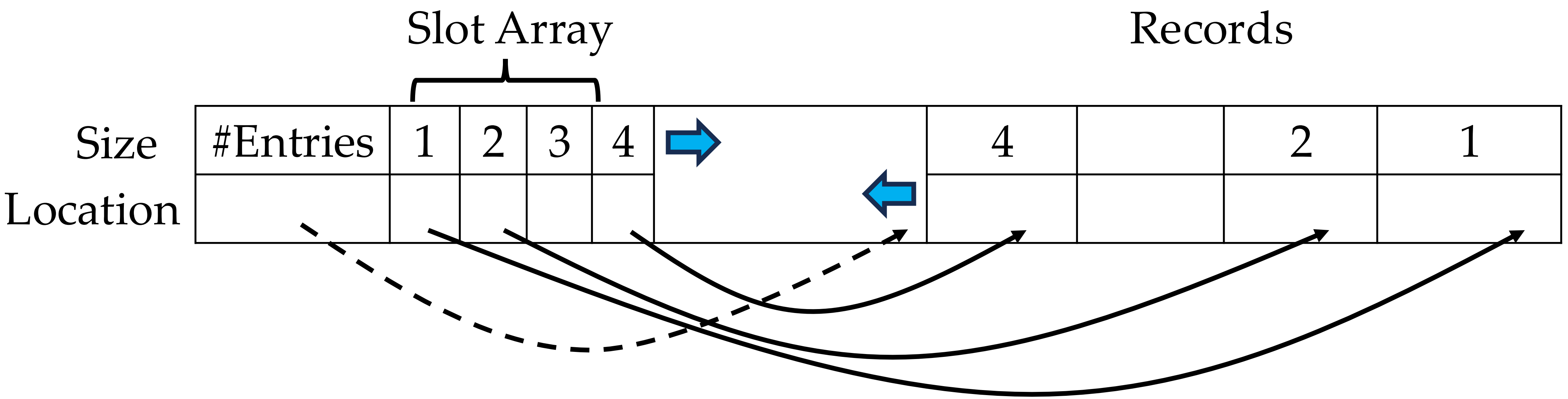
Slotted Pages: Record Deletion

- Record 3 and the pointer to record 3 are removed.
- Then what?



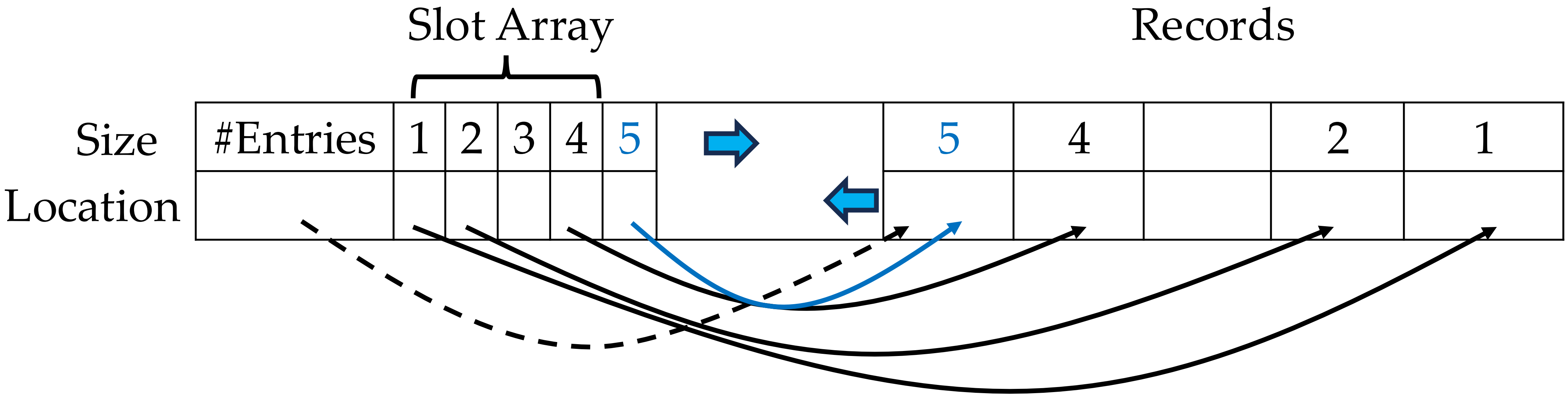
Slotted Pages: Record Deletion & Insertion

- Let’s try to add a new record 5.
- Where do you think record 5 will get added?



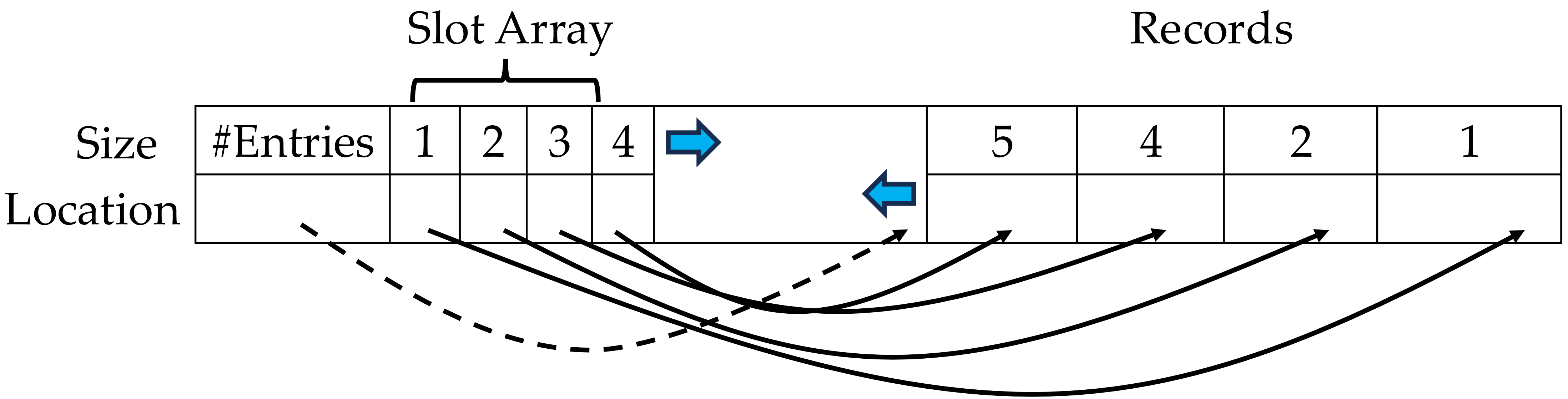
Slotted Pages: Record Deletion & Insertion

- In Postgres, record 5 gets added in the free space.
- Does not use the existing free locations.
- Then how can we reclaim this empty slot?



Slotted Pages: Record Deletion & Insertion

- In Postgres, call the garbage collection operation.
- Post garbage collection, pointer reordering and record reorganization.



Slotted Pages: Delayed Compaction

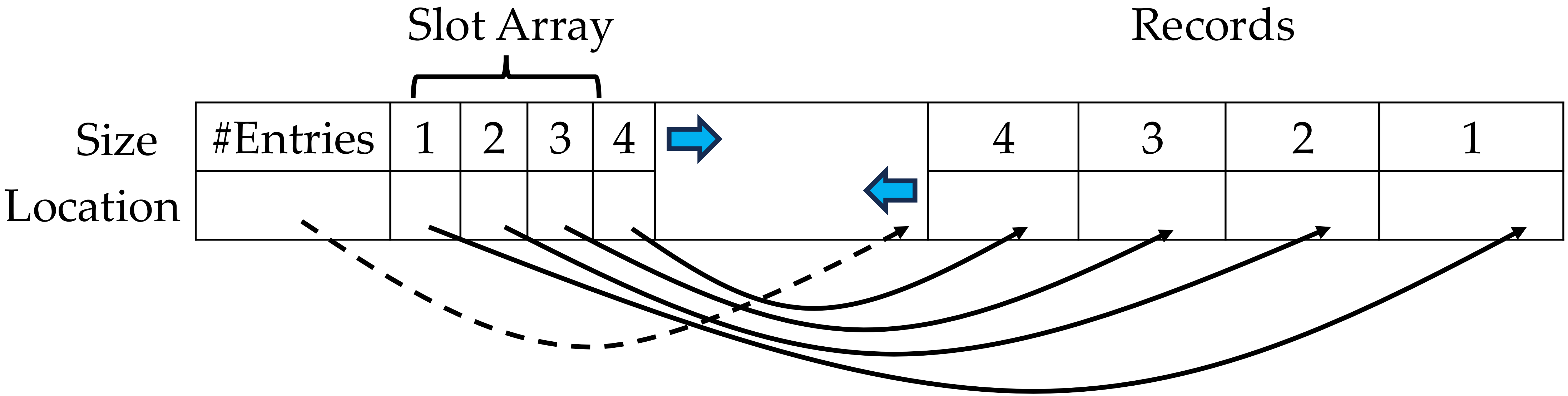
- This notion of delayed compaction post deletion is pretty common.

- Databases like Postgres, Sqlite, and Oracle do not compact unless required or an explicit call to compaction.

- However, what do you think happens in other databases, for example: SQLServer?

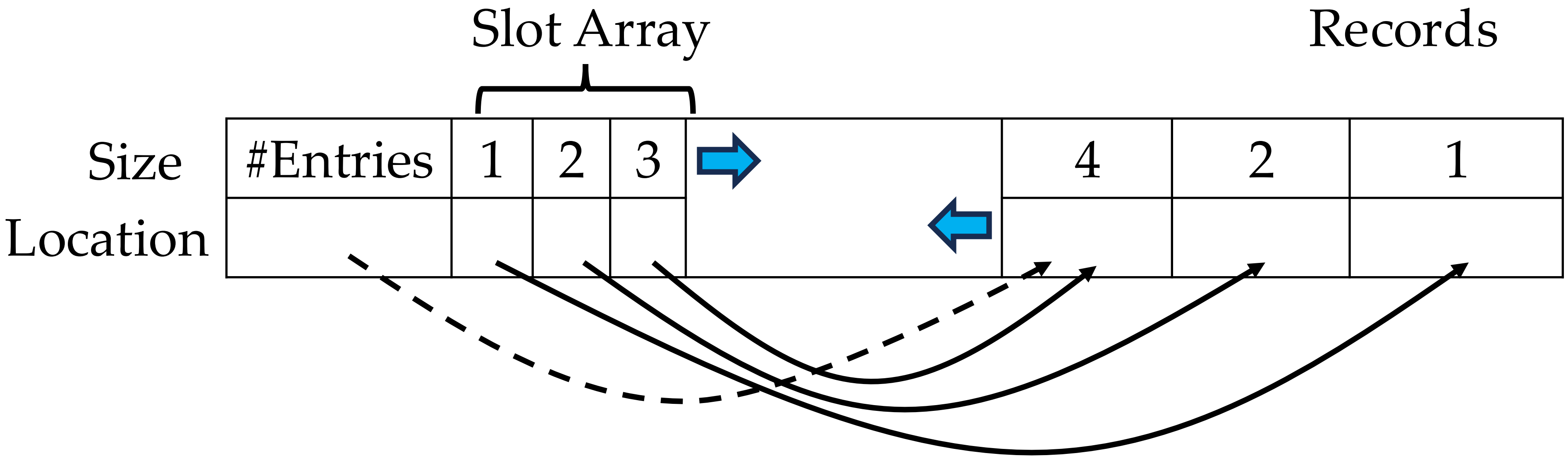
Slotted Pages: Record Deletion

- Lets try to delete Record 3 in SQLServer.



Slotted Pages: Record Deletion

- Lets try to delete Record 3 in SQLServer.
- Post deletion, it actively performs compaction.



Record-Oriented Storage

- Slotted pages layout tells us how the records are stored in a page, but how do we insert or delete records in a page.

- In Lecture 3, we discussed sequential file organization, where we said that all the records are stored sequentially in a table.

- We discussed how records are inserted and deleted in this sequential file organization.

- Lets go one step deeper!

Record-Oriented Storage

- Steps to insert a new record in the page?

Record-Oriented Storage

- **Steps to insert** a new record in the page?
  - Search page directory to find a page with a free slot.
  - If the page not in memory, fetch it from the disk.
  - Check slot array to find empty space in page that will fit.

Record-Oriented Storage

- **Steps to update** an existing record using its record id?
  - Search page directory to find the location of page (pointer to the page).
  - If the page not in memory, fetch it from the disk.
  - Find offset to the record using slot array in the page.
  - If new data fits, overwrite existing data.
  - Otherwise, mark existing tuple as deleted and insert new version in a different page.

Challenges for Record-Oriented Storage

- **Fragmentation:**
  - Not all pages are fully utilized → unusable space and empty slots.
- **Expensive Disk Fetching:**
  - To update one record, entire page needs to be fetched from the disk.
- **Random Disk Fetches:**
  - Say you want to update multiple unrelated tuples.
  - You would end up fetching multiple pages from disk → Too expensive!
- **How about no possibility of in-place updates?**
  - In-place updates essentially means that you can overwrite a record.
  - But a lot of systems prohibit in-place updates, e.g. HDFS, Google Colossus.
  - These systems support append-only → you can only insert a new copy of the record.

Log-Structured Merge Trees

Log-Structured Storage

- Originally proposed as log-structure merge trees (LSM Trees) in 1996.

- Designed to support efficient write (update) operations.

- Essentially, records are stored in a log-style data structure.

- Each record is represented as a **(key, value)** pair.
  - Key is the record ID.
  - Value could be one single value, or a tuple corresponding to all the records.

- Unlike record-oriented architecture where we were updating in-place, we now **append a new entry**.
  - In future, we will asynchronously **merge values**.

Log-Structured Storage

Memory

- All operations start at memory: search, insert, update, delete.
- All operations occur in an-memory table → **MemTable.**
- When MemTable becomes **full or you can't find something** in MemTable, then move to disk.

Disk

- When something not exists in memory, then move to disk.
- If memory is full, then create an on-disk table → **SSTable.**
- Disk is divided into multiple levels → Hierarchical structure.
- When a level gets full, merge occurs.

Log-Structured Storage

- MemTable can be any data structure:
  - A sorted list or B^+-tree.

- Supports in-place updates.

- If something found in MemTable, then we have quick access.

- As MemTable has limited size, so once full, it needs to be sent to the disk and a new Memtable is created.

Log-Structured Storage

Memory

Disk

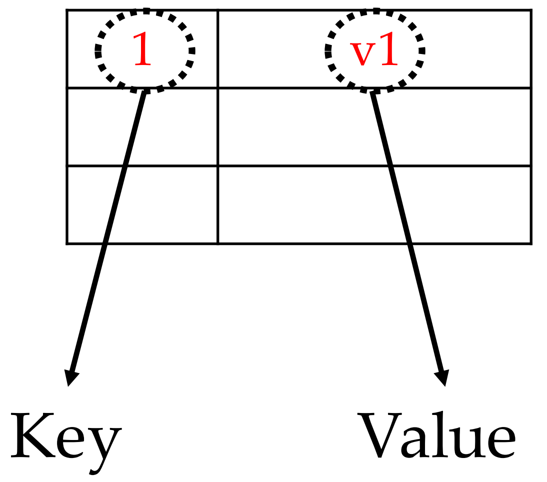
Initially, our system looks like this. We have an empty MemTable.

Log-Structured Storage

Memory

Disk

Say, we insert **record (1,v1).**



Log-Structured Storage

Memory

Disk

Next, insert **record (5,a1).**

1	v1
5	a1

Log-Structured Storage

Memory

Disk

Next, insert **record (8,b1).**

1	v1
5	a1
8	b1

Log-Structured Storage

Memory

Disk

Next, **update record (1,v2).**

We can do in-place update!

1	v2
5	a1
8	b1

Log-Structured Storage

Memory

Disk

Next, insert **record (2,v1).** What should we do?

1	v1
5	a1
8	b1

Log-Structured Storage

Memory

Disk

Create a new MemTable and add **(2,v1)**.

2	v1

1	v1
5	a1
8	b1

Log-Structured Storage

Memory

Convert the old MemTable to SSTable and push to disk. Records need to be sorted.

2	v1

1	v1
5	a1
8	b1

Disk

1	v1
5	a1
8	b1

Level 0

Log-Structured Storage

Memory

Next, insert **record (8,b2).**

2	v1
8	b2

Disk

1	v1
5	a1
8	b1

Level 0

Log-Structured Storage

Memory

Next, insert **record (4,b1).**

2	v1
4	b1
8	b2

Disk

1	v1
5	a1
8	b1

Level 0

Log-Structured Storage

Memory

Next, insert **record (3,c1).**

3	c1

Disk

2	v1
4	b1
8	b2

1	v1
5	a1
8	b1

Level 0

Log-Structured Storage

Memory

Let's assume at some point of time our this MemTable is **also full**.

3	c1
4	b2
8	b3

Disk

2	v1
4	b1
8	b2

1	v1
5	a1
8	b1

Level 0

Log-Structured Storage

Memory

So we will push this MemTable also to Disk.

3	c1
4	b2
8	b3

Disk

2	v1
4	b1
8	b2

1	v1
5	a1
8	b1

Level 0

Log-Structured Storage

Memory

But, say my Level 0 on disk can hold only 2 SSTables. **So what should we do?**

3	c1
4	b2
8	b3

Disk

2	v1
4	b1
8	b2

1	v1
5	a1
8	b1

Level 0

Log-Structured Storage

Memory

Merge tables at Level 0, and push merged table to Level 1.

3	c1
4	b2
8	b3

Disk

Level 0

2	v1
4	b1
8	b2

1	v1
5	a1
8	b1

Level 1

1	v1
2	v1
4	b1
5	a1
8	b2

Log-Structured Storage

Memory

Merge tables at Level 0, and push merged table to Level 1.

3	c1
4	b2
8	b3

Disk

Level 0

2	v1
4	b1
8	b2

1	v1
5	a1
8	b1

Level 1

1	v1
2	v1
4	b1
5	a1
8	b2

Essentially, the idea is that Level 1 can hold far greater number of records than Level 0.

Log-Structured Storage

Memory

Similarly, when Level 1 is full, we merge Level 1 SSTables to Level 2.

5	a2
10	a2
12	b2

Disk

Level 0

3	c2
4	b3
8	b3

Level 1

1	v2
3	c1
6	c1
10	a1
12	b1

1	v1
2	v1
4	b1
5	a1
8	b2

Level 2

Log-Structured Storage

- The number of levels depend on the database.
  - Generally, at most 7 levels.

- Size of each level up to the administrator.

- Merges happen asynchronously.

Log-Structured Storage: Search

Memory

Say, we want to search a **record 5** (get value corresponding to 5).

5	a2
10	a2
12	b2

Disk

Level 0

3	c2
4	b3
8	b3

Level 1

1	v2
3	c1
6	c1
10	a1
12	b1

1	v1
2	v1
4	b1
5	a1
8	b2

Level 2

Log-Structured Storage: Search

Memory

First, go to the MemTable and see if it exists!

5	a2
10	a2
12	b2

Disk

3	c2
4	b3
8	b3

Level 0

1	v2
3	c1
6	c1
10	a1
12	b1

1	v1
2	v1
4	b1
5	a1
8	b2

Level 1

Level 2

Log-Structured Storage: Search

Memory

Say, we want to search a **record 4.**

5	a2
10	a2
12	b2

Disk

Level 0

3	c2
4	b3
8	b3

Level 1

1	v2
3	c1
6	c1
10	a1
12	b1

1	v1
2	v1
4	b1
5	a1
8	b2

Level 2

Log-Structured Storage: Search

Memory

First, go to the MemTable and see if it exists!

5	a2
10	a2
12	b2

Disk

Level 0

3	c2
4	b3
8	b3

Level 1

1	v2
3	c1
6	c1
10	a1
12	b1

1	v1
2	v1
4	b1
5	a1
8	b2

Level 2

Log-Structured Storage: Search

Memory

Unfortunately, it does not. So, we go to disk Level 0.

5	a2
10	a2
12	b2

Disk

Level 0

3	c2
4	b3
8	b3

Level 1

1	v2
3	c1
6	c1
10	a1
12	b1

1	v1
2	v1
4	b1
5	a1
8	b2

Level 2

Log-Structured Storage: Search

Memory

Found 4.

5	a2
10	a2
12	b2

Disk

Level 0

3	c2
4	b3
8	b3

Level 1

1	v2
3	c1
6	c1
10	a1
12	b1

1	v1
2	v1
4	b1
5	a1
8	b2

Level 2

Log-Structured Storage: Search

Memory

Say, we want to search a **record 6.**

5	a2
10	a2
12	b2

Disk

Level 0

3	c2
4	b3
8	b3

Level 1

1	v2
3	c1
6	c1
10	a1
12	b1

1	v1
2	v1
4	b1
5	a1
8	b2

Level 2

Log-Structured Storage: Search

Memory

First, go to the MemTable and see if it exists!

5	a2
10	a2
12	b2

Disk

Level 0

3	c2
4	b3
8	b3

Level 1

1	v2
3	c1
6	c1
10	a1
12	b1

1	v1
2	v1
4	b1
5	a1
8	b2

Level 2

Log-Structured Storage: Search

Memory

Unfortunately, it does not. So, we go to disk Level 0.

5	a2
10	a2
12	b2

Disk

Level 0

3	c2
4	b3
8	b3

Level 1

1	v2
3	c1
6	c1
10	a1
12	b1

1	v1
2	v1
4	b1
5	a1
8	b2

Level 2

Log-Structured Storage: Search

Memory

Does not exist at even Level 0. So, we go to disk Level 1.

5	a2
10	a2
12	b2

Disk

Level 0

3	c2
4	b3
8	b3

Level 1

1	v2
3	c1
6	c1
10	a1
12	b1

1	v1
2	v1
4	b1
5	a1
8	b2

Level 2

Log-Structured Storage: Search

Memory

Two SSTables at Level 1, so which to search first? Try newest!

5	a2
10	a2
12	b2

Disk

Level 0

3	c2
4	b3
8	b3

Level 1

1	v2
3	c1
6	c1
10	a1
12	b1

1	v1
2	v1
4	b1
5	a1
8	b2

Level 2

Log-Structured Storage: Search

- What are the limitations of this search process?

Log-Structured Storage: Search

- What are the limitations of this search process?

- Manually traverse over all the tables in each level.

- Search all levels until you find the record.

Log-Structured Storage: Search

- How can we do better?

Log-Structured Storage: Search

- How can we do better?

- Two design optimizations:
  - Range Pointer
  - Bloom filter

Log-Structured Storage: Search

- How can we do better?

- Two design optimizations:
  - **Range Pointer**:
    - Tells the minimum and maximum key identifiers in a table.

  - **Bloom filter**:
    - Tells if a record exists of not.

Log-Structured Storage: Delete

Memory

Say, we want to delete a **record 5**.

5	a2
10	a2
12	b2

Disk

Level 0

3	c2
4	b3
8	b3

Level 1

1	v2
3	c1
6	c1
10	a1
12	b1

1	v1
2	v1
4	b1
5	a1
8	b2

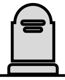
Level 2

Log-Structured Storage: Delete

Memory

First, go to the MemTable and see if it exists!

If it does, then add a **tombstone.**

5	
10	a2
12	b2

Disk

Level 0

3	c2
4	b3
8	b3

Level 1

1	v2
3	c1
6	c1
10	a1
12	b1

1	v1
2	v1
4	b1
5	a1
8	b2

Level 2

Log-Structured Storage: Delete

Memory

Say, we want to delete a **record 4**.

5	a2
10	a2

Disk

Level 0

3	c2
4	b3
8	b3

Level 1

1	v2
3	c1
6	c1
10	a1
12	b1

1	v1
2	v1
4	b1
5	a1
8	b2

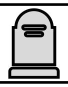
Level 2

Log-Structured Storage: Delete

Memory

First, go to the MemTable and see if it exists!

Record 4 does not exist, so create an entry with **tombstone.**

5	a2
4	
10	a2

Disk

3	c2
4	b3
8	b3

Level 0

1	v2
3	c1
6	c1
10	a1
12	b1

1	v1
2	v1
4	b1
5	a1
8	b2

Level 1

Level 2

Log-Structured Storage: Delete

- So what happens to this tombstone or the deleted record once the MemTable is full?

Log-Structured Storage: Delete

- So what happens to this tombstone or the deleted record once the MemTable is full?

- This tombstone will trickle down to SSTables.

- The most recent entry for record 4 will indicate a tombstone, which means the record is deleted.

Log-Structured Storage: Complexity

- If you have a lot of insert operations, LSM-tree is good for you.

- But, if you have read-then-write type updates, then it is expensive as you will have to fetch the record, check the value, and then update.
  - For example, a query that states, find all employees with salary > 100.

- Deletes again are cheap.

Log-Structured Storage: Merge

- Merge operation helps in compaction and reduce size.

- Too many redundant entries make it hard to search or update.

- When to merge?
  - Definitely when the level is full.
  - You can also merge whenever there are more than one table at a level.

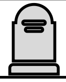
Log-Structured Storage: Merge Algorithm

- How to merge two SSTables?

Log-Structured Storage: Merge Algorithm

- How to merge two SSTables?
- Have two iterators, and sequentially scan.
- As entries in both the tables are sorted, so each to merge.

Log-Structured Storage: Merge Algorithm

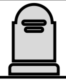
1	v2
3	c1
4	b2
5	
12	c3

+

1	v1
2	v1
4	b1
5	a1
8	b2

=

Log-Structured Storage: Merge Algorithm

1	v2
3	c1
4	b2
5	
12	c3

Take one entry of this table

+

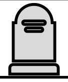
1	v1
2	v1
4	b1
5	a1
8	b2

Go over all entries of this table.

Stop the search if the key is greater.

=

Log-Structured Storage: Merge Algorithm

1	v2
3	c1
4	b2
5	
12	c3

Take one entry of this table

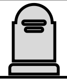
1	v1
2	v1
4	b1
5	a1
8	b2

Go over all entries of this table.

Stop the search if the key is greater.

1	v2

Log-Structured Storage: Merge Algorithm

1	v2
3	c1
4	b2
5	
12	c3

Take one entry of this table

+

1	v1
2	v1
4	b1
5	a1
8	b2

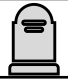
Go over all entries of this table.

Stop the search if the key is greater.

=

1	v2

Log-Structured Storage: Merge Algorithm

1	v2
3	c1
4	b2
5	
12	c3

Take one entry of this table

+

1	v1
2	v1
4	b1
5	a1
8	b2

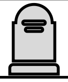
Go over all entries of this table.

Stop the search if the key is greater.

=

1	v2
2	v1

Log-Structured Storage: Merge Algorithm

1	v2
3	c1
4	b2
5	
12	c3

Take one entry
of this table

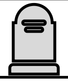
1	v1
2	v1
4	b1
5	a1
8	b2

Go over all entries
of this table.

Stop the search if
the key is greater.

1	v2
2	v1
3	c1

Log-Structured Storage: Merge Algorithm

1	v2
3	c1
4	b2
5	
12	c3

Take one entry of this table

+

1	v1
2	v1
4	b1
5	a1
8	b2

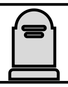
Go over all entries of this table.

Stop the search if the key is greater.

=

1	v2
2	v1
3	c1
4	b2

Log-Structured Storage: Merge Algorithm

1	v2
3	c1
4	b2
5	
12	c3

Take one entry of this table

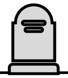
+

1	v1
2	v1
4	b1
5	a1
8	b2

Go over all entries of this table.

Stop the search if the key is greater.

=

1	v2
2	v1
3	c1
4	b2
5	
8	b2
12	c3

NoSQL Databases

NoSQL Databases

- As the name suggests, by default do not support SQL.
- The idea became popular from Facebook.
- Today, an extensive number of popular databases are NoSQL.
- These databases have only two operations: **get()** and **put()**.
  - **Get()** – Searching or looking up a record in the database.
  - **Put()** – Writing or updating a record in the database.

NoSQL Databases

- The design of LSM-trees became popular through NoSQL databases like RocksDB, LevelDB.

- NoSQL database consists of just a pair → (**key, value**)
  - Key → record identifier, possibly a number.
  - Value → Any information about the record.

- How do you implement a NoSQL database?

- You can use an ordered hash-map, or an array, or a vector!