Database Processing
CS 451 / 551

Lecture 8:
Workloads, Storage Models, and Filters

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!

How to plan your Database Design

- What is the information you need to decide how your data should be stored or how should you allow access to your data?

- You need to know about the workloads that your database may encounter?
  - Type of queries your clients may send.
  - Type of data you may want to input to your data.
  - The frequency of data input versus data analysis.

Database Workloads

- The design of your database is often based on the workload.
  - Workload by definition means the queries or inputs to your database.
  - It can imply read or write queries.

- Database Workloads can be broadly sub-divided into three categories:
  - OLTP → On-Line Transaction Processing
  - OLAP → On-Line Analytical Processing
  - HTAP → Hybrid Transaction + Analytical Processing

On-Line Transaction Processing

- OLTP Workloads are the most common type of database workloads.
- They comprise of fast, simple read and write operations.
- You can think of these workloads as write-heavy.
- Essentially, when you design your database, this is the first type of workloads you will target.
  - Help you to gauge performance of insertion, deletion, and simple queries.

Wikipedia Example

create table useracct (
userID **int primary key**,
userName **varchar unique**,
…
);

create table pages (
pageID **int primary key**,
title **varchar unique**,
latest **int,**
references revisions (revID)
);

create table revisions (
revID **int primary key**,
userID **int**,
references useracct (userID),
pageID **int**,
references pages (pageID),
content text,
updated datetime
);

OLTP Queries

select p.*, r.*
from pages **as** p
inner join
revisions **as** r
on p.latest = r.revID
where p.pageID = 10

update useracct
set
lastlogin = **now()**,
hostname = 10.1.1.9
where userid = “Voldemort”

insert into revisions
values
(?,?…,?)

On-Line Analytical Processing

- OLAP Workloads help to perform analysis on your database.
- They comprise of complex queries that help you to learn useful information from your database.
- This can also be termed as data mining → you are trying to mine patterns or knowledge from your database.

OLAP Queries

```
select count( u.lastlogin ),
extract(month from u.lastlogin) as month
from useracct as u
where u.hostname like '%.gov'
group by extract(month from u.lastlogin)
```

Storage Models

- Until now, we discussed how data is stored in pages and how records are arranged in a page or disk.
- But, we are yet to discuss about the different storage models.
- These storage models help the database designer decide what is the right way for storing the records of their database.
- Does your database targets OLAP or OLTP or HTAP?
  - These decisions help you select an appropriate storage model.

Storage Models

- Storage Models can be broadly divided into three categories:
  - N-ary Storage Model (NSM)
  - Decomposition Storage Model (DSM)
  - Hybrid Storage Model (PAX)

N-ary Storage Models

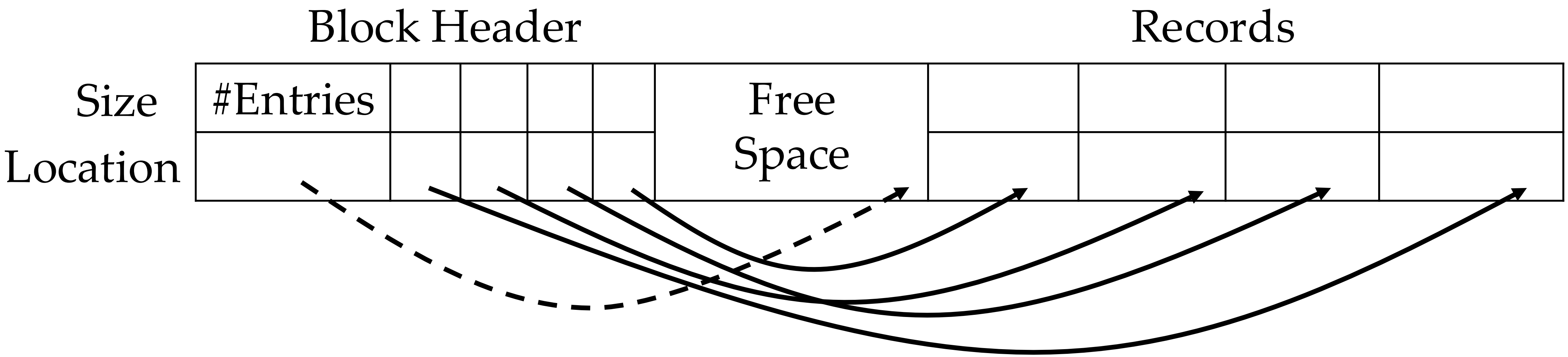
- So how does an N-ary storage model look?
- In NSM, you store the full record contiguously.
- All the attributes (columns) of the record are stored together in the same page.
- This design is also called as row-store.

N-ary Storage Models

- NSM is useful for which type of database workload?
- NSM is useful for OLTP workloads.
- You would like to have queries that try to fetch multiple or all attributes of the record.
- Further, NSM is ideal for write-heavy queries.

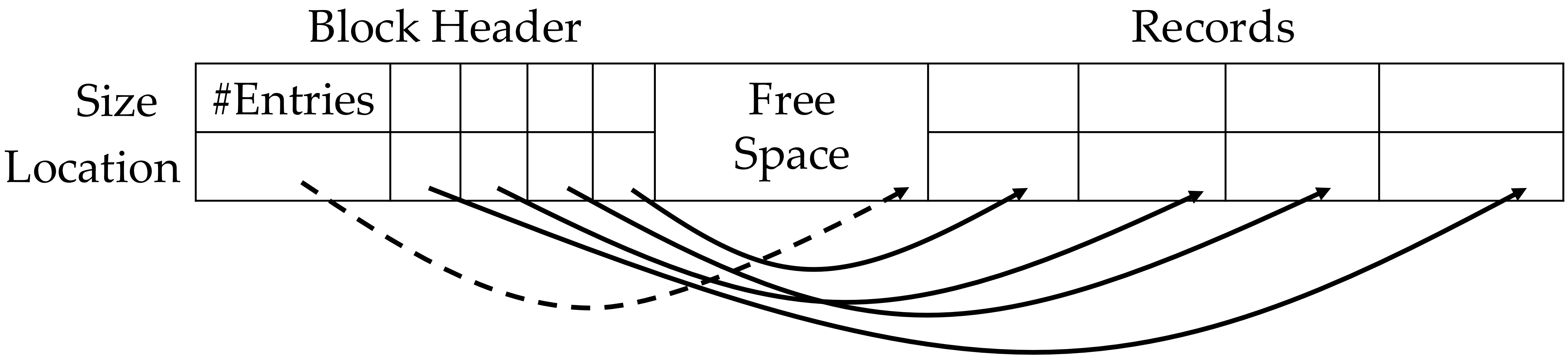
Slotted Pages

- Last class, we saw this slotted pages architecture for storing records.



Slotted Pages

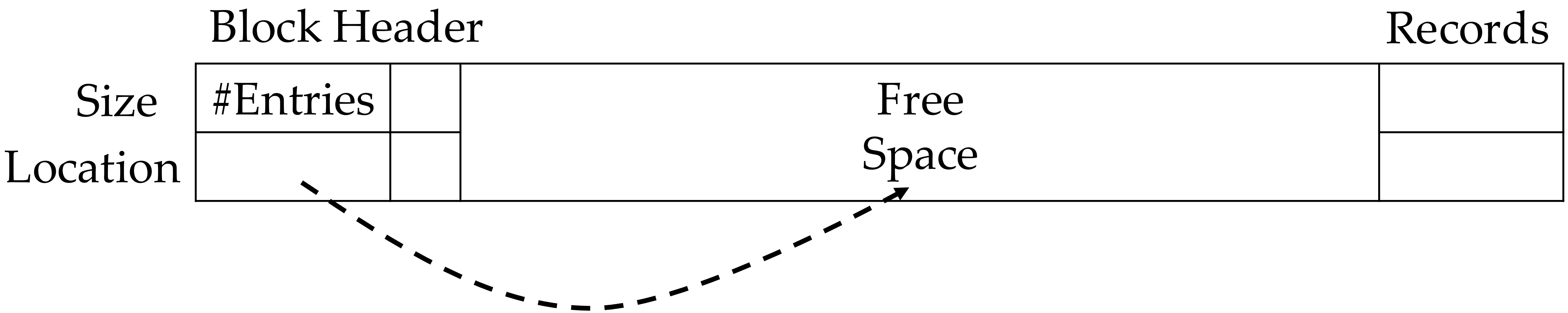
- Last class, we saw this slotted pages architecture for storing records.
- If we want to store records according to NSM, then how will we proceed?



Slotted Pages

Say, we have the following record. Storing it in NSM requires storing all the attributes of the record contiguously.

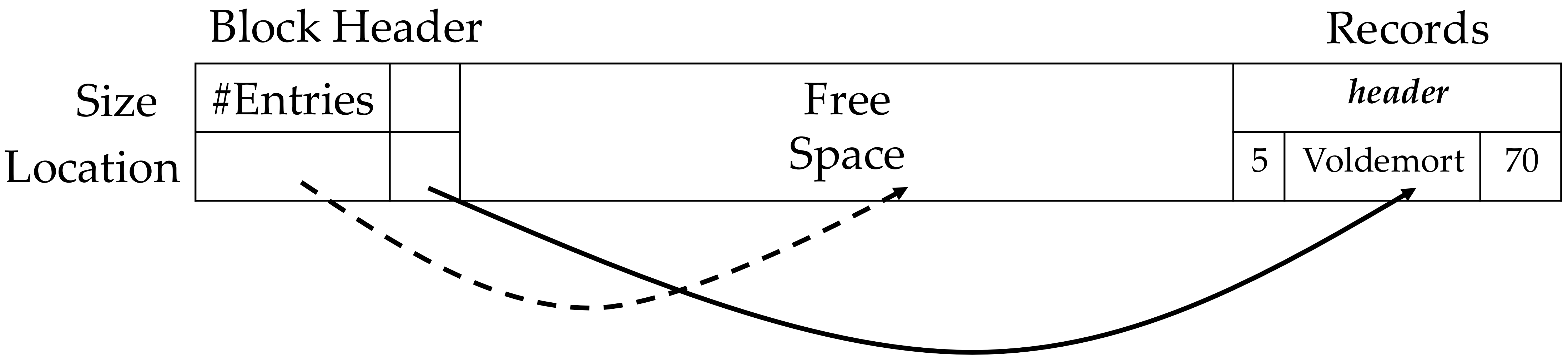
5	Voldemort	70



Slotted Pages

Say, we have the following record. Storing it in NSM requires storing all the attributes of the record contiguously.

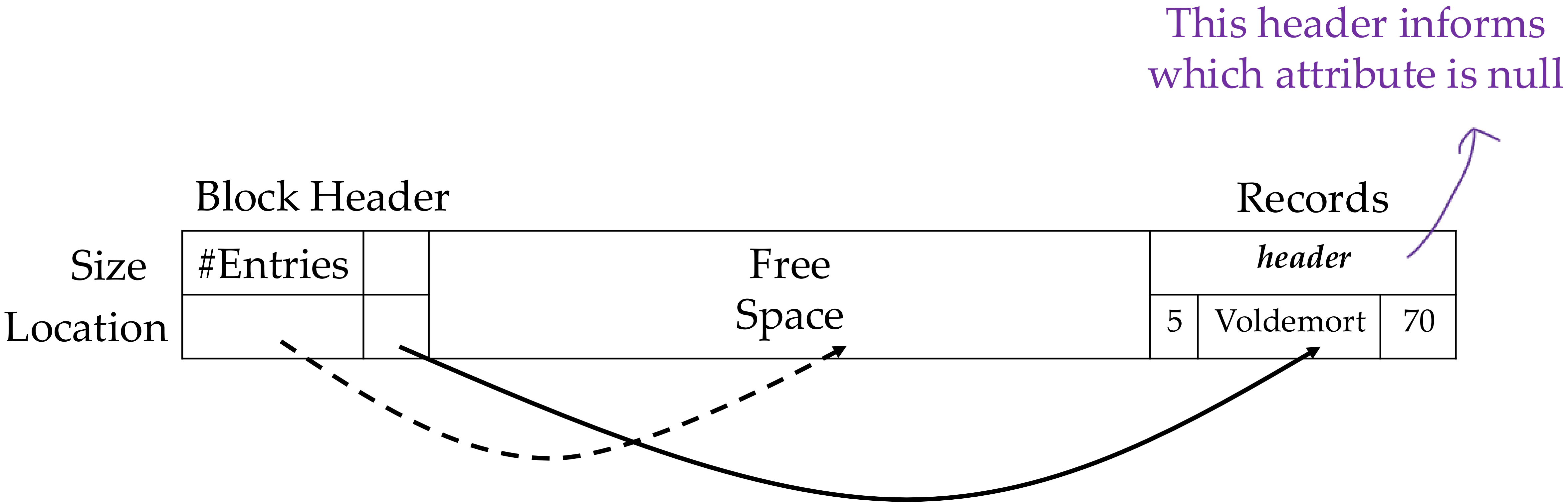
5	Voldemort	70



Slotted Pages

Say, we have the following record. Storing it in NSM requires storing all the attributes of the record contiguously.

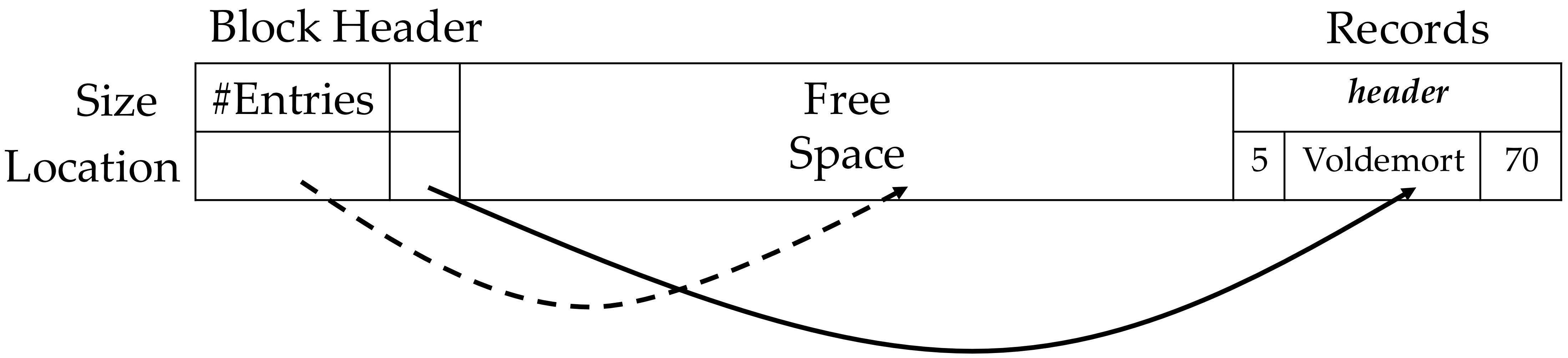
5	Voldemort	70



Slotted Pages

Now lets add another record.

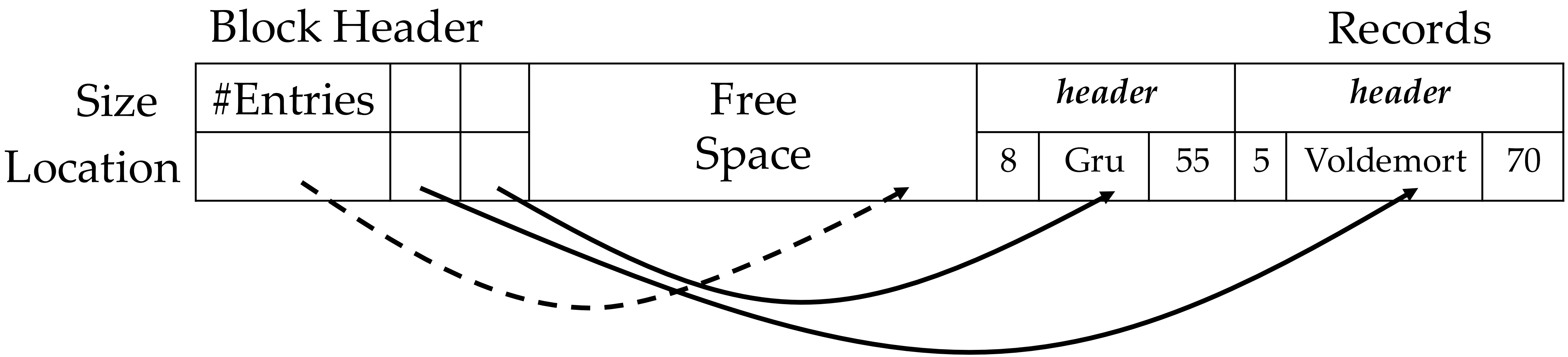
8	Gru	55



Slotted Pages

Now lets add another record.

8	Gru	55

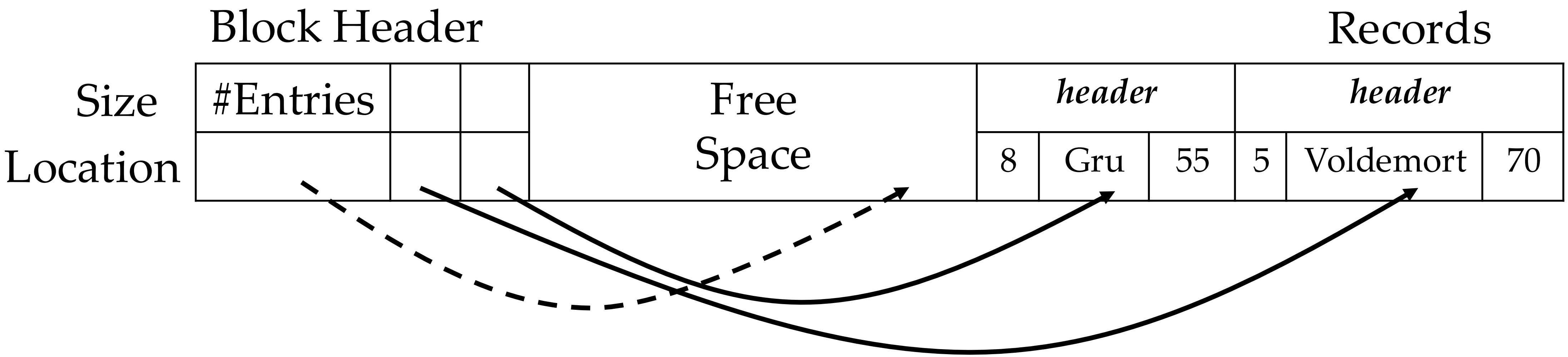


Slotted Pages

Now lets add another record.

8	Gru	55

We can use this process to continue adding records until the page is full.

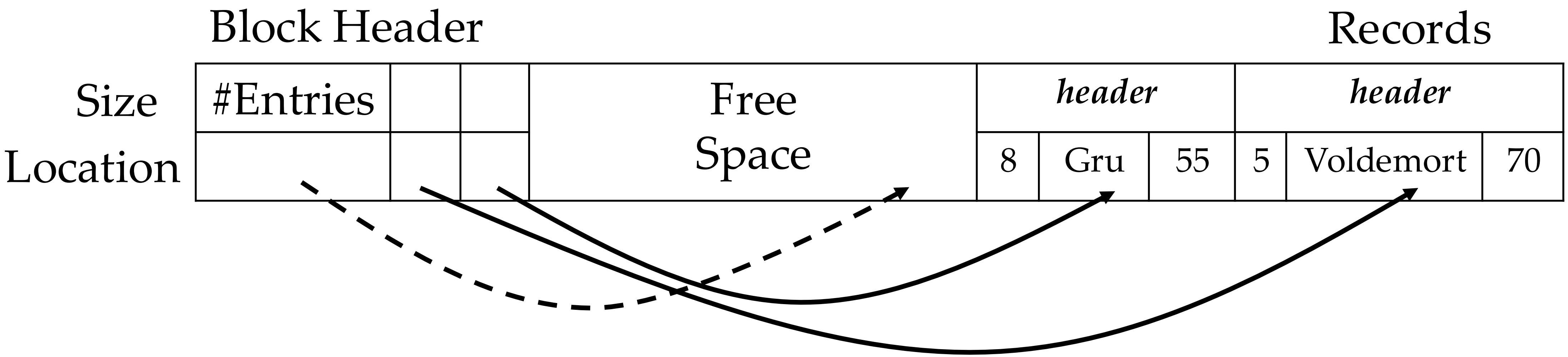


Slotted Pages

Now lets add another record.

8	Gru	55

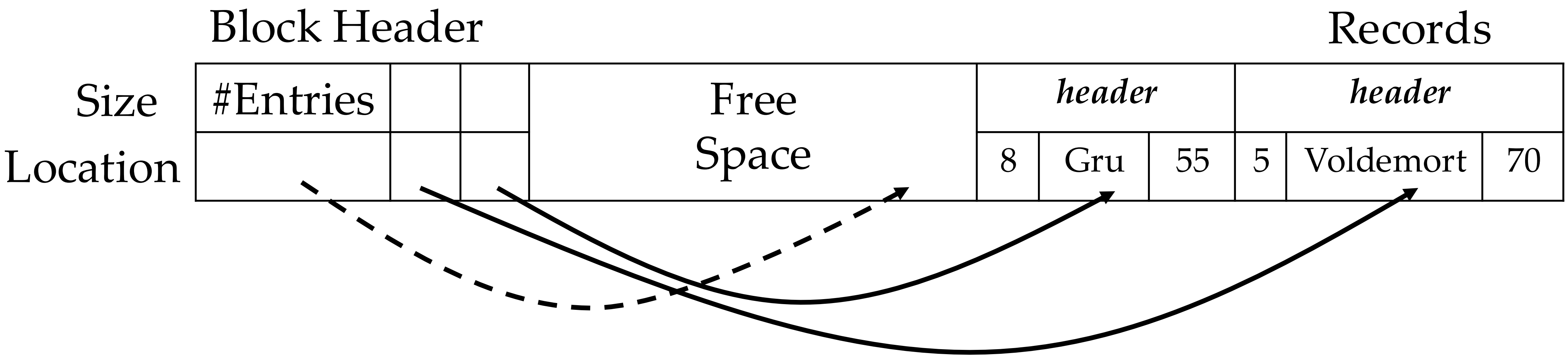
Notice that it does not matter if the attribute is fixed or variable-length.



OLTP on NSM

- Let's try to run an OLTP query on our NSM architecture.

select * **from** cs_employees
where age > 50;

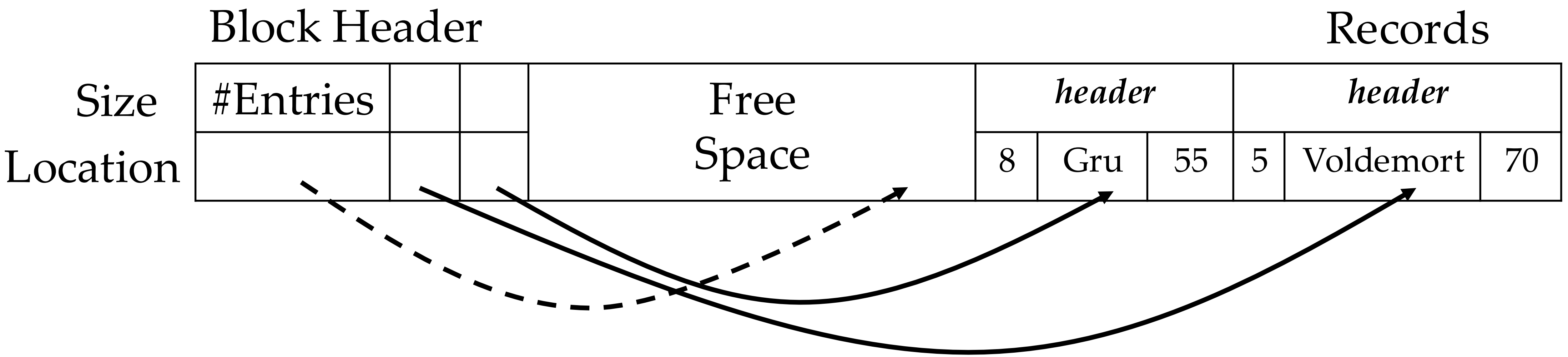


OLTP on NSM

- Let's try to run an OLTP query on our NSM architecture.

select * **from** cs_employees
where age > 50;

This query is perfect for NSM as we want to access all the attributes.

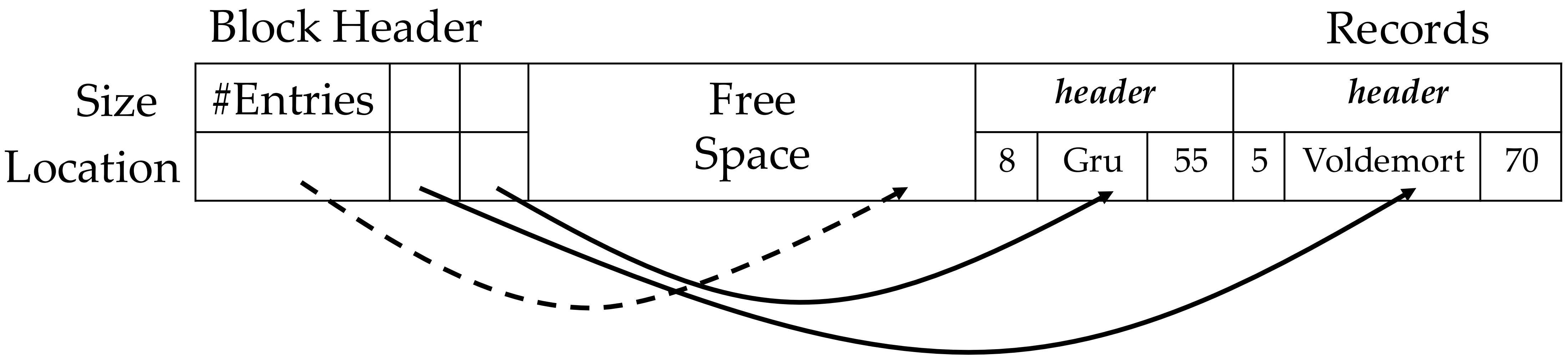


OLTP on NSM

- Or, how about an OLTP insert query on our NSM architecture.

insert into cs_employees **values**

(9, Kang, 100);

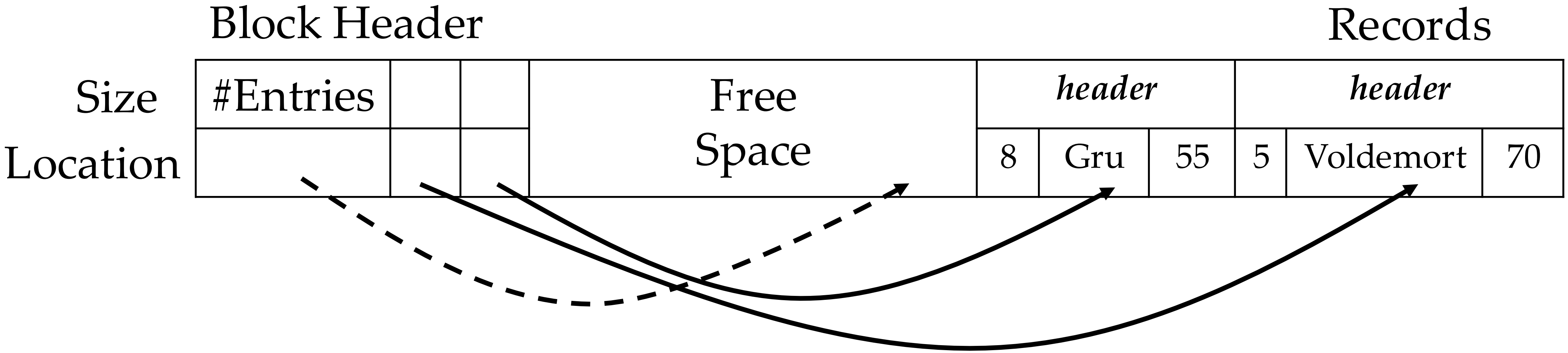


OLTP on NSM

- Or, how about an OLTP insert query on our NSM architecture.

insert into cs_employees **values**
(9, Kang, 100);

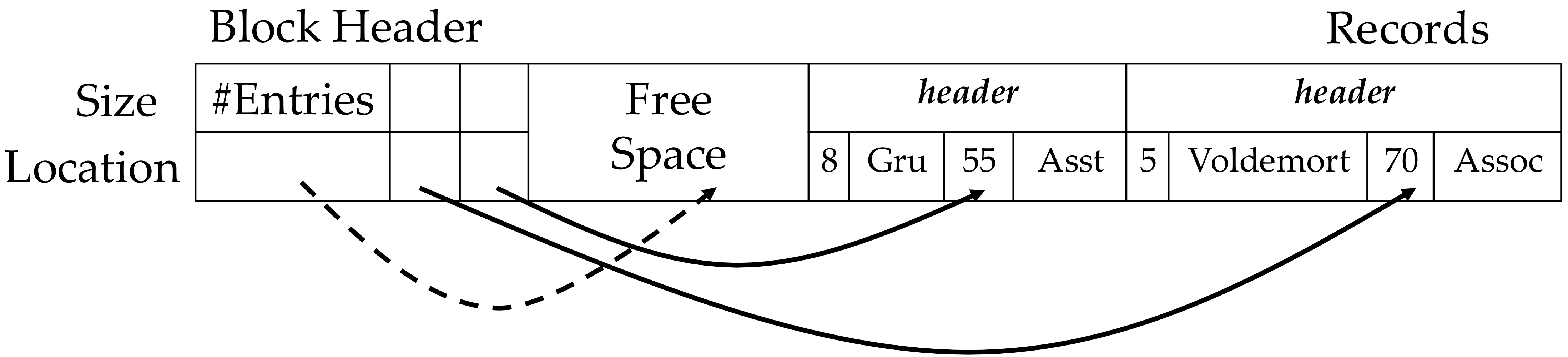
This query is also good for NSM as we want to insert a new record.



OLAP on NSM

- Let's try to run an OLAP query on our NSM architecture.

select avg(age**) from** cs_employees
group by title **having** avg(age) > 50;

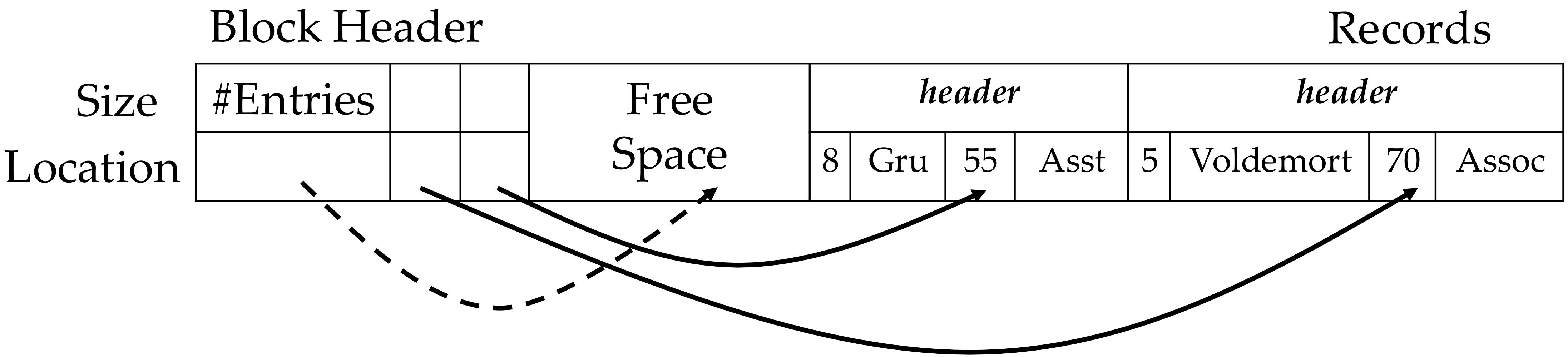


OLAP on NSM

- Let's try to run an OLAP query on our NSM architecture.

select avg(age**) from** cs_employees
group by title **having** avg(age) > 50;

This query is not suitable for NSM as we need to traverse all the records and every attribute when we need only two attributes.



N-ary Storage Models

- **Advantages**
  - Fast inserts, updates, and deletes.
  - Good for queries that need the entire record.

- **Disadvantages**
  - Yield poor performance for queries that access only a subset of attributes.
  - Poor memory locality in access patterns.
  - Not ideal for compression because of multiple value domains within a single page.

Decomposition Storage Models

- So how do DSM look?
- In DSM, each page only stores one attribute for all the records.
- This design is also called as column-store.
- In your assignment, you are following the column-store design.

Decomposition Storage Models

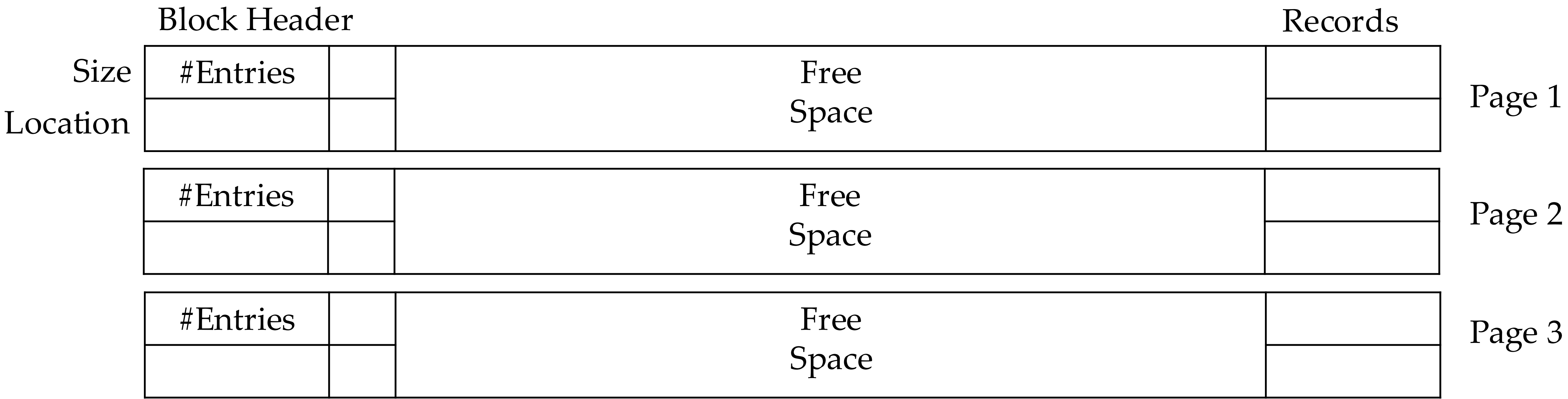
- DSM is useful for which type of database workload?
- DSM is useful for OLAP workloads.
- You would like to have queries that try to fetch a small subset of all the attributes of the record.
- Further, DSM is ideal for read-heavy queries → Queries where the goal is to perform analysis.
- It is the job of the DBMS to combine/split the attributes of a record when reading/writing.

Records in DSM

Say, we have the following record. Storing it in DSM requires storing all the attributes of the record contiguously.

5	Voldemort	70

Notice that we have three different pages, one for each attribute.

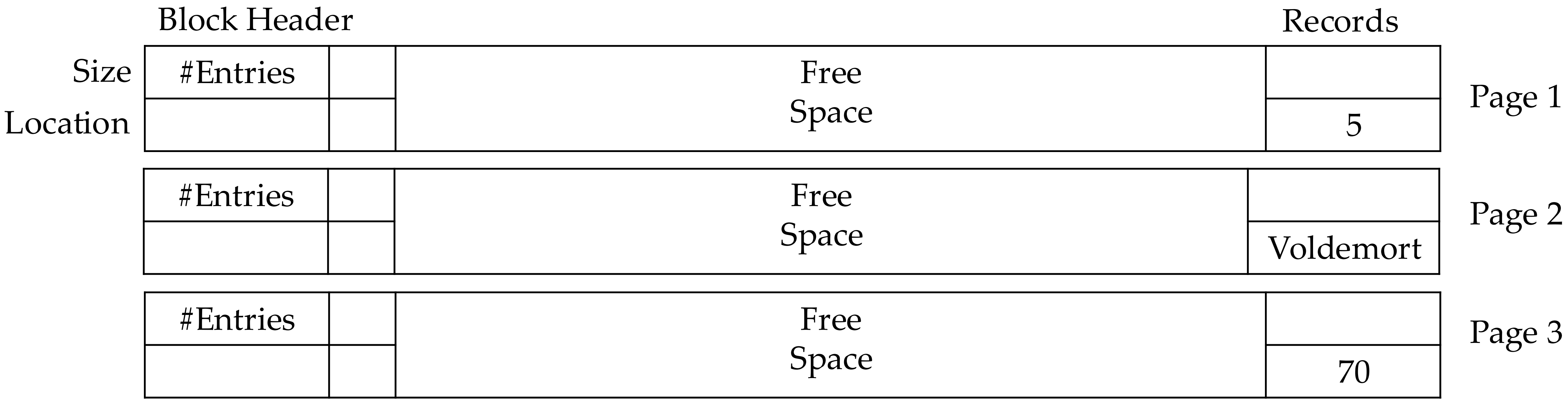


Records in DSM

Say, we have the following record. Storing it in DSM requires storing all the attributes of the record contiguously.

5	Voldemort	70

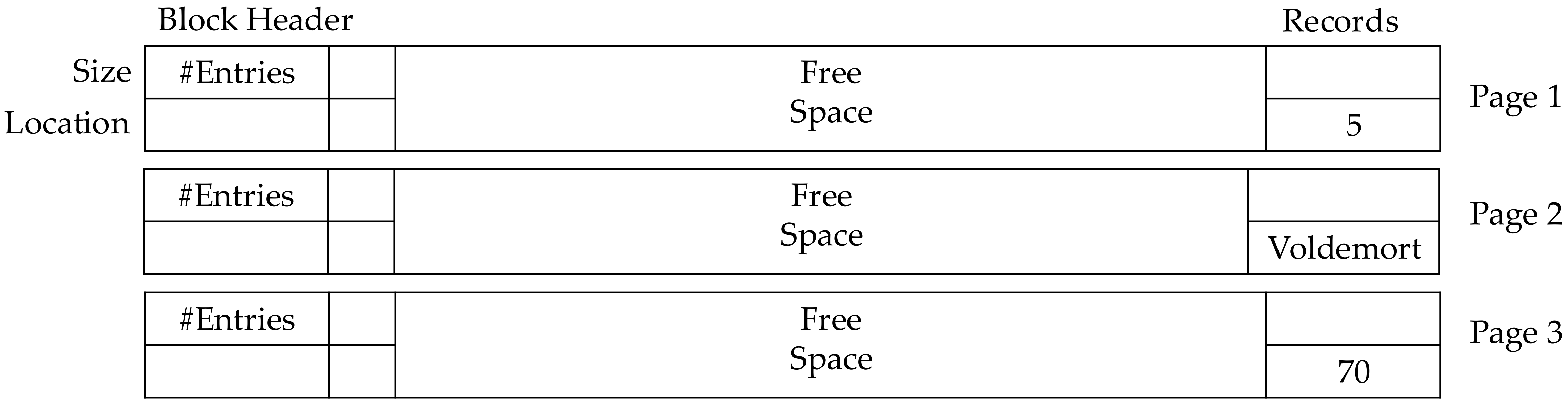
Each attribute goes to its own page



Records in DSM

Now, lets add another record.

8	Gru	55



Records in DSM

Now, lets add another record.

8	Gru	55

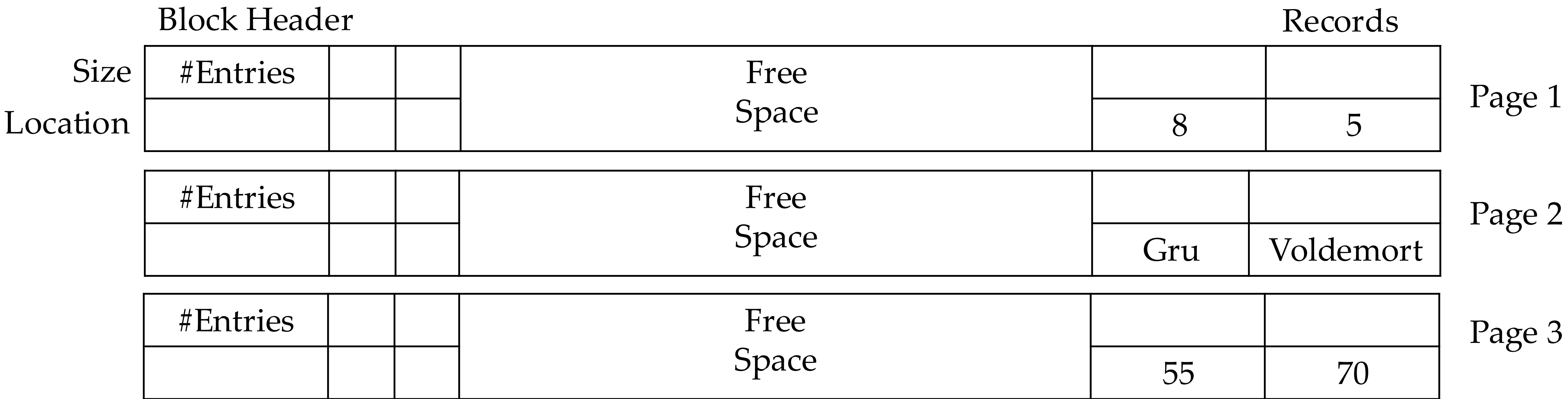
	Block Header					Records	
Size	#Entries			Free Space			Page 1
Location					8	5	
	#Entries			Free Space			Page 2
					Gru	Voldemort	
	#Entries			Free Space			Page 3
					55	70	

Records in DSM

Now, lets add another record.

8	Gru	55

In each page, the header informs about the record position.



Decomposition Storage Models

- Storing fixed-length attributes is easy as we can have relative offsets.

- Remember, relative offsets inform about the position of the record.

- But, how will you store variable-length attributes?

Decomposition Storage Models

- **Solution 1**:
    - Store the offset and length of the attribute.
    - Bad idea, too much storage per attribute.
- **Solution 2**:
    - Add some padding to each value to ensure each value has a fixed length.
    - Waste of space!
- **Solution 3**:
    - Use compression techniques to ensure each attribute value is of the same size.

OLAP on DSM

- Makes sense to do OLAP on DSM!

- Remember our earlier query, if we run that query on a DSM, then only a subset of attributes need to be fetched from disk!

Decomposition Storage Models

- Advantages
  - Reduces the amount wasted I/O per query because the DBMS only reads the data that it needs.
  - Faster query processing because of increased locality.
  - Better data compression.

- Disadvantages
  - Slow for point queries, inserts, updates, and deletes because of attribute splitting/stitching.

So can we do something better?

- We know that rarely a query will touch only one attribute or all attributes.
- So what can be a better design?

Partition Attributes Across (PAX)

- PAX is a hybrid model.
- It first splits rows into groups, and within each group all the attributes are stored as DSM.
  - Horizontally partition rows into groups, then vertically partition columns.
- Aim is to permit faster processing on columnar storage while retaining the spatial locality benefits of row storage.
- Example database: Parquet, ORC, and Arrow.

Records in DSM

Now, lets add another record.

5	Voldemort	70
8	Gru	55
12	Kang	100
20	Anakin	40

Page 1

5	8	Voldemort	Gru	70	55

Page 2

12	20	Kang	Anakin	100	40

Filters and Indexes

Bloom Filters

- We saw that bloom filter can be used to optimize search in LSM-trees.

- So, what is a bloom filter?

- It is a **probabilistic bitmap**, which informs about existence of a key.
  - If the key exists, bloom filter will say yes (**true positive**).
  - If the key exists, bloom filter will not say no (**no false negatives**).
  - If the key does not exist, bloom filter may say yes (**possible false positives**).

Bloom Filters

- How to calculate the size of your bloom filter?

- You need to know the following:
  - Number of items in the bloom filter
  - What is the desirable probability of false positives?
  - Number of hash functions.
  - Then use this tool.

Insertions in Bloom Filters

0	1	2	3	4	5	6

Say, this is our bloom filter with 7 bits.

Insertions in Bloom Filters

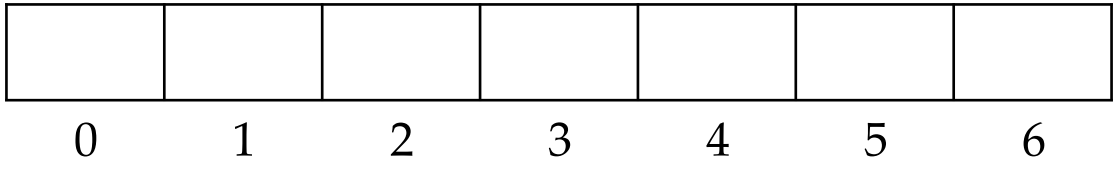
$\boldsymbol{h1(key)}$ $\boldsymbol{h2(key)}$

0	1	2	3	4	5	6

Say, we have two hash functions for mapping.

Insertions in Bloom Filters

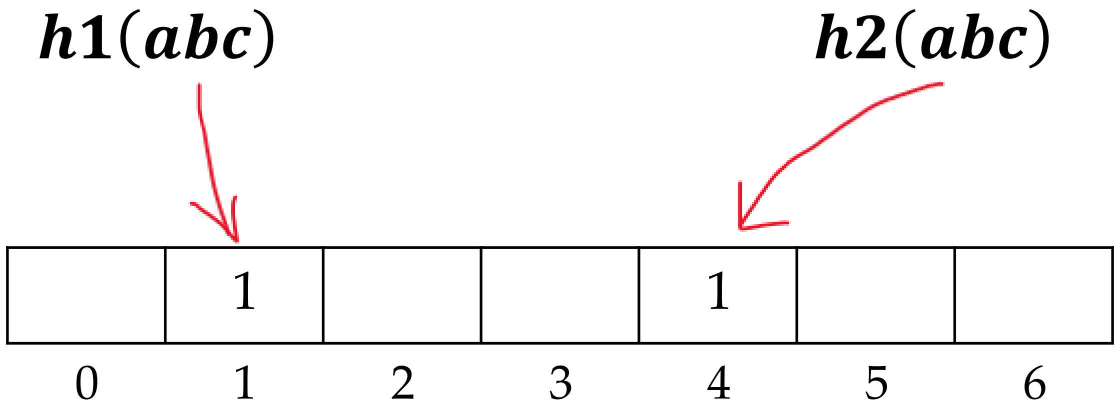
$\boldsymbol{h1(abc)}$ $\boldsymbol{h2(abc)}$



Let's add a key **"abc"** to the bloom filter.

Insertions in Bloom Filters

h1(abc) = 1
h2(abc) = 4

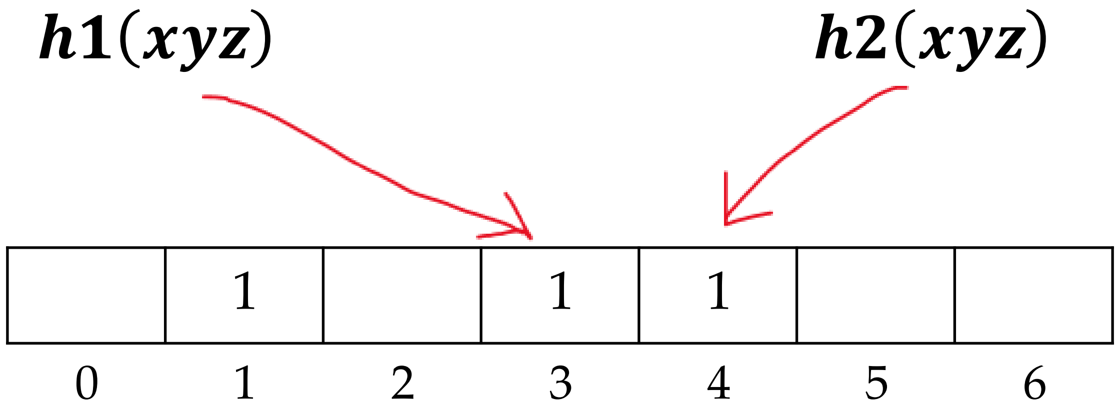


Let's add a key **"abc"** to the bloom filter.

Insertions in Bloom Filters

h1(abc) = 1
h2(abc) = 4

h1(xyz) = 3
h2(xyz) = 4



Let's add a key **"xyz"** to the bloom filter.

Search in Bloom Filters

h1(abc) = 1
h2(abc) = 4

h1(xyz) = 3
h2(xyz) = 4

$\boldsymbol{h1(xyz)}$ $\boldsymbol{h2(xyz)}$

	1		1	1		
0	1	2	3	4	5	6

Let’s search for the key **“xyz”.**

Search in Bloom Filters

h1(abc) = 1
h2(abc) = 4

h1(xyz) = 3
h2(xyz) = 4

Key found → True positive!

$\boldsymbol{h1(xyz)}$ $\boldsymbol{h2(xyz)}$

	1		1	1		
0	1	2	3	4	5	6

Let's search for the key **"xyz".**

Search in Bloom Filters

h1(abc) = 1
h2(abc) = 4

h1(xyz) = 3
h2(xyz) = 4

$\boldsymbol{h1(uvw)}$ $\boldsymbol{h2(uvw)}$

	1		1	1		
0	1	2	3	4	5	6

Let's search for the key **"uvw".**

Search in Bloom Filters

h1(abc) = 1
h2(abc) = 4

h1(xyz) = 3
h2(xyz) = 4

h1(uvw) = 1
h2(uvw) = 3

$\boldsymbol{h1(uvw)}$ $\boldsymbol{h2(uvw)}$

	1		1	1		
0	1	2	3	4	5	6

Let's search for the key **"uvw".**

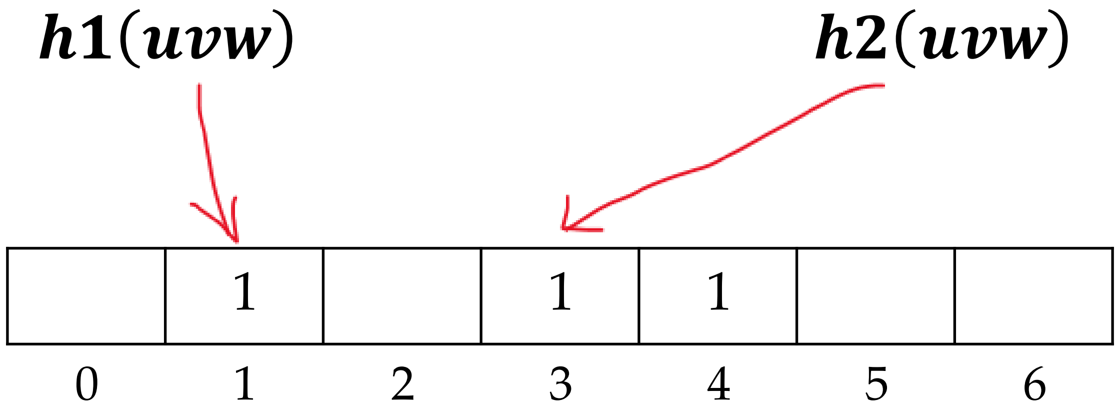
Search in Bloom Filters

h1(abc) = 1
h2(abc) = 4

h1(xyz) = 3
h2(xyz) = 4

h1(uvw) = 1
h2(uvw) = 3

Key found??
Should not happen → False positive!



Let's search for the key **"uvw".**

Deletions in Bloom Filters

- How to delete something from the bloom filter?

- Extremely dangerous!
  - To delete some key, you need to set the corresponding bits to 0.
  - But, it can lead to marking some other key as does not exist.
  - Deletions can lead to false negatives!

Deletions in Bloom Filters

- How to delete something from the bloom filter?

- Extremely dangerous!
  - To delete some key, you need to set the corresponding bits to 0.
  - But, it can lead to marking some other key as does not exist.
  - Deletions can lead to false negatives!

- **Possible solutions?**

Counting Bloom Filters

- Instead of setting each bit as 0 or 1, now we store a **counter** for each bit.

- Each time a hash function points to a bit, increment the counter for that bit.
  - Each bit's counter starts from 0 $\rightarrow$ every time hash function tells you a bit, increment its counter!

- The value of the counter tells you possible number of keys that have caused it to increment.

Counting Bloom Filters

- Can there be false positives?
  - Yes, because the counter values does not tell existence of which specific key.

- Is deletion possible?
  - Yes, deletion is possible by decrementing the counter.

Skip Lists

Skip Lists

- So, some lectures ago, we discussed ways to search a key from this sorted list?

6 8 12 19 23 34 45 67 76 78 83 98

Skip Lists

- What were the possible solutions?
  - Binary Search
  - BST
  - B^+-Tree

6	8	12	19	23	34	45	67	76	78	83	98

Skip Lists

- Can we do something better in the context of memory?

6 8 12 19 23 34 45 67 76 78 83 98

Skip Lists

- Can we do something better in the context of memory?
  - B^+-Tree are great but require rebalancing.
  - Use a lot of memory → Main memory is way smaller than disk.

6 8 12 19 23 34 45 67 76 78 83 98

Skip Lists

- Any good solution will also help to improve searching from the MemTable (LSMTrees), which is stored in the memory.

Skip Lists

- For such specific settings, a skip list can be useful.

Skip Lists

- Skip list has a search complexity of $\boldsymbol{O}(\boldsymbol{log\ N})$,
  - where N is the number of elements.

Skip Lists

- Why Skip list?
  - A multi-level list, where each level skips almost half the number of keys in the previous level.
  - Level 1 → N keys
  - Level 2 → N/2 keys
  - Level 3 → N/4 keys

- This is why Skip List is also called a probabilistic data
  - You toss a coin and decide whether to add a key to a level or not.
  - If coin toss is 1, you add the key to the higher level.
  - At each level you keep tossing and moving up till you get a 0.

Skip Lists

Lets try to insert keys to an empty skip list with 3 levels.

Level 3
(N/4 keys)

Level 2
(N/2 keys)

Level 1
(N keys)

Skip Lists

Lets insert 6 by tossing a coin.

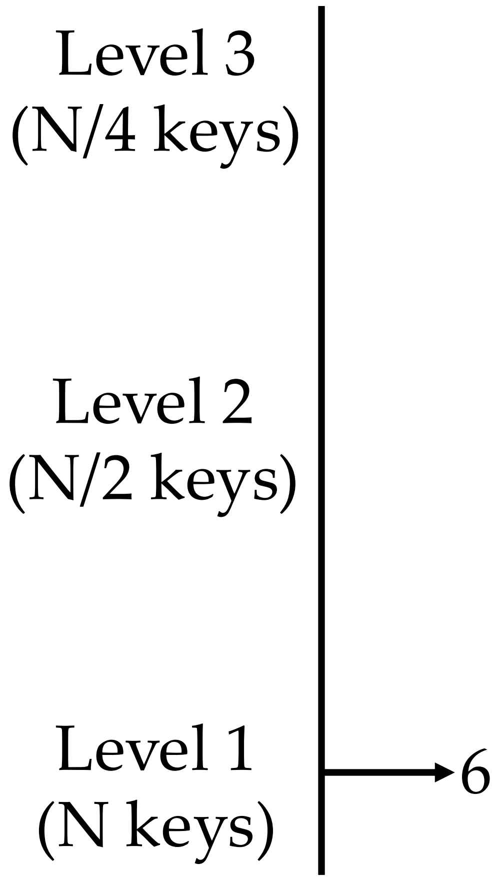
Level 3
(N/4 keys)

Level 2
(N/2 keys)

Level 1
(N keys)

Skip Lists

Say coin toss is 0, so we only insert 6 at Level 1.



Skip Lists

Lets insert 23 and say coin toss is 0.

Level 3
(N/4 keys)

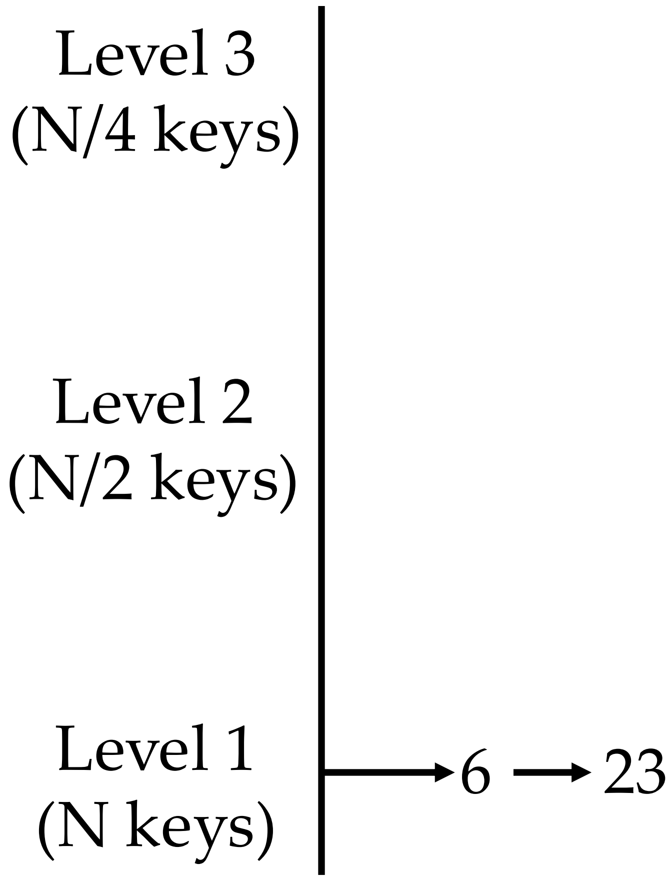
Level 2
(N/2 keys)

Level 1
(N keys)

→ 6 → 23

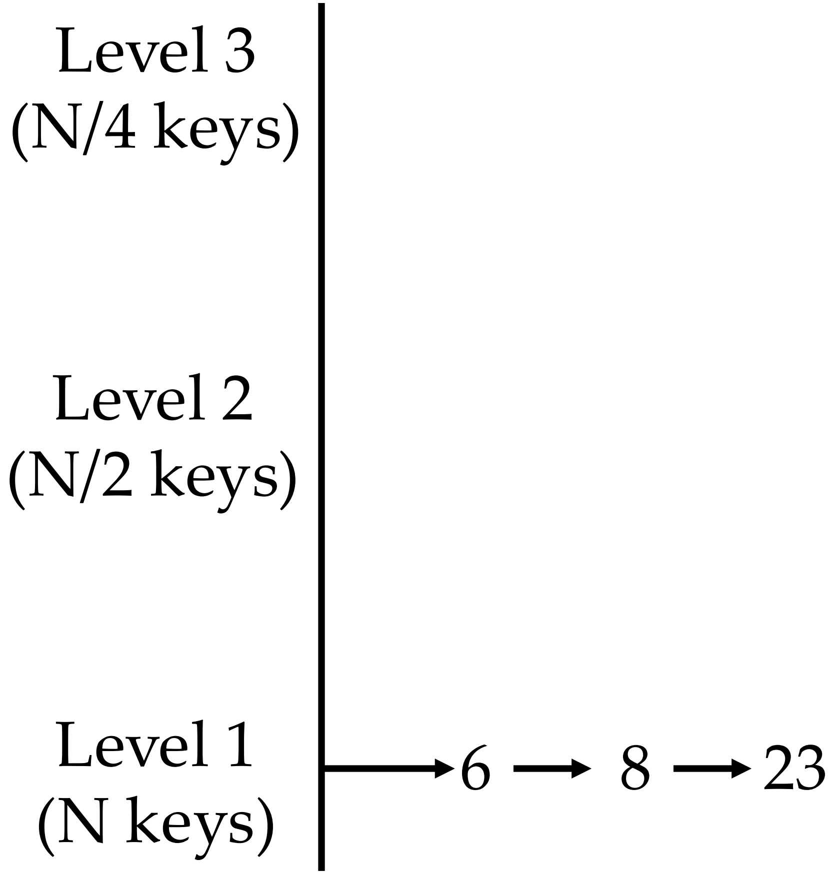
Skip Lists

Lets insert 8 and say coin toss is 1.



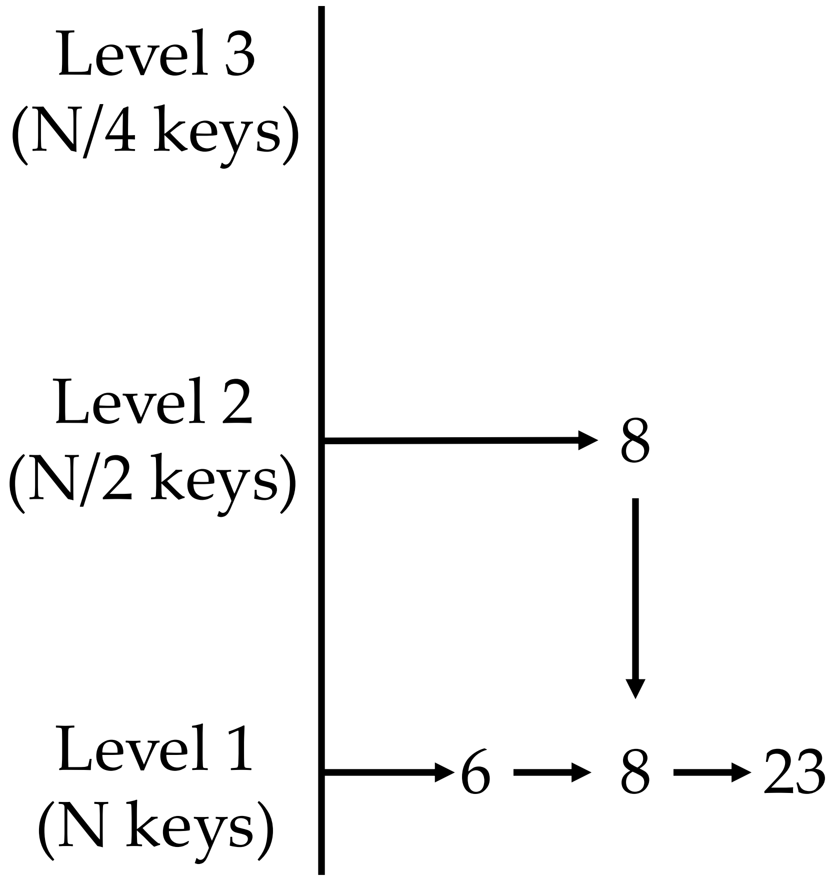
Skip Lists

First, we insert 8 as Level 1 and then move to Level 2.



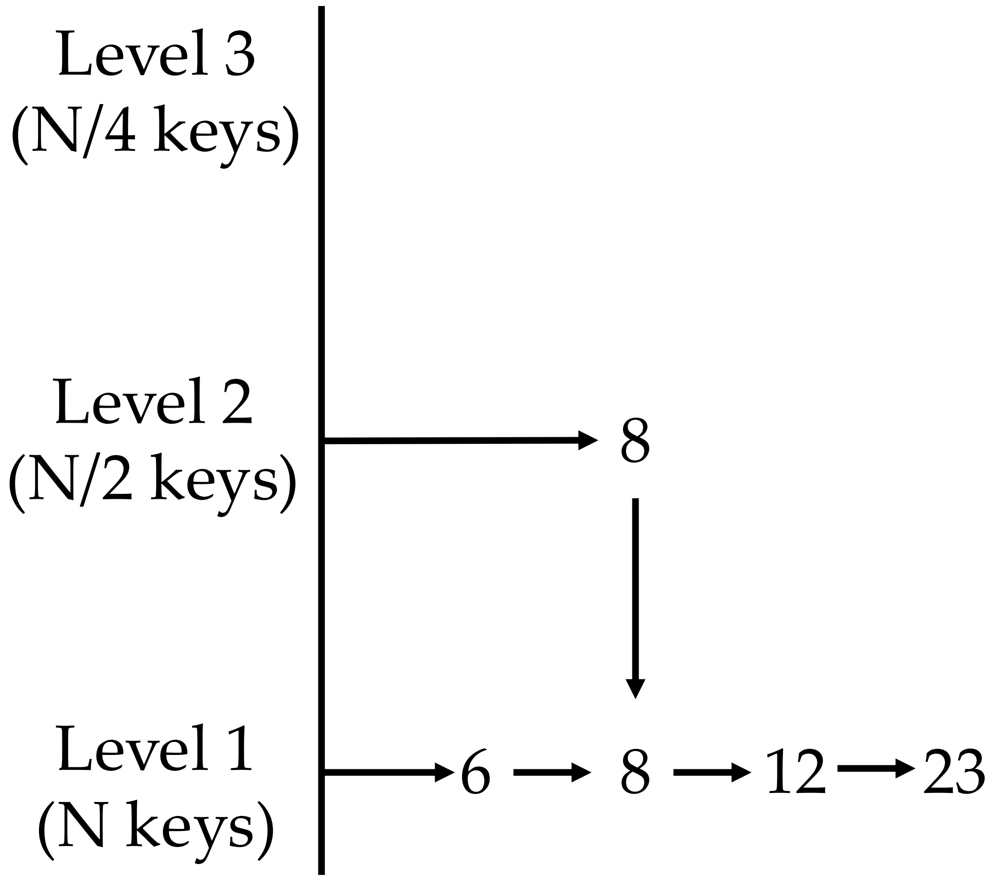
Skip Lists

We again toss the coin, and say it is now 0, so we insert 8 at Level 2 and stop



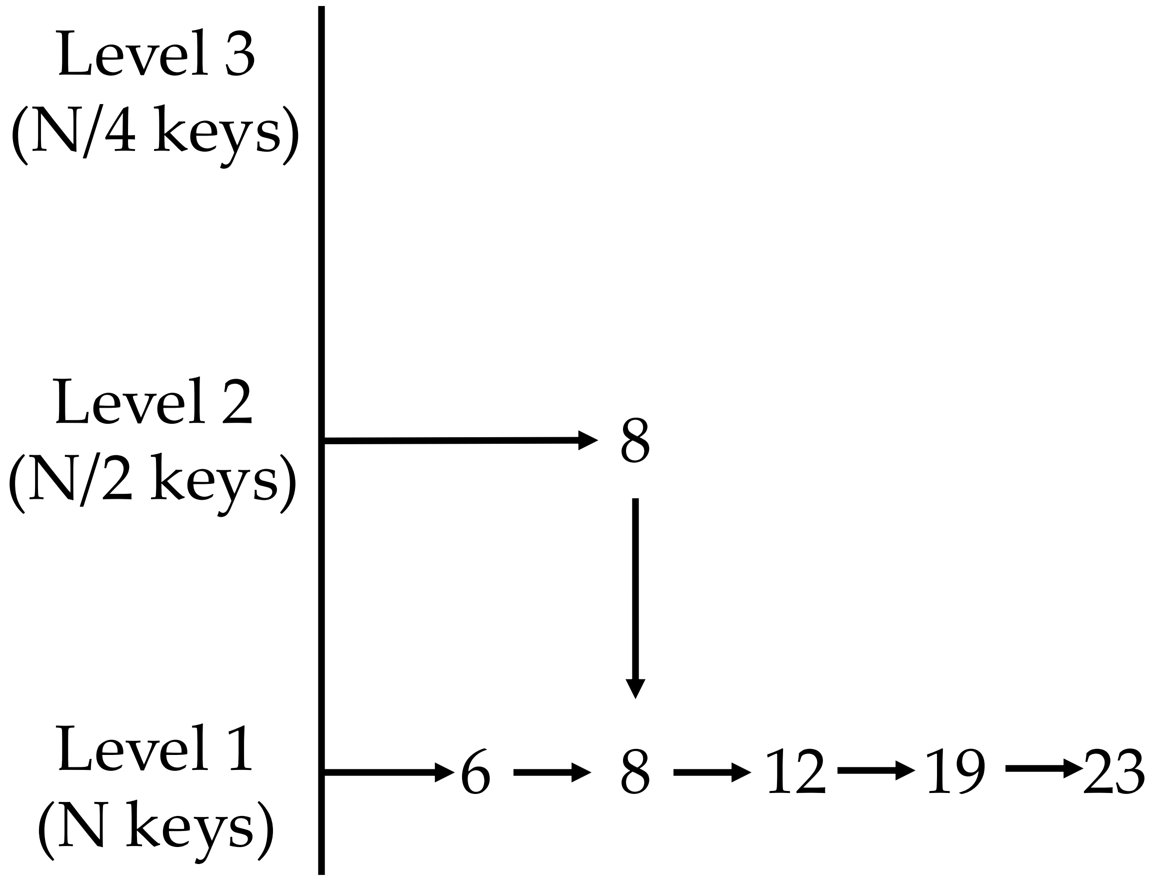
Skip Lists

Let's insert a value 12 and say coin toss is 0, so we stop at Level 1.



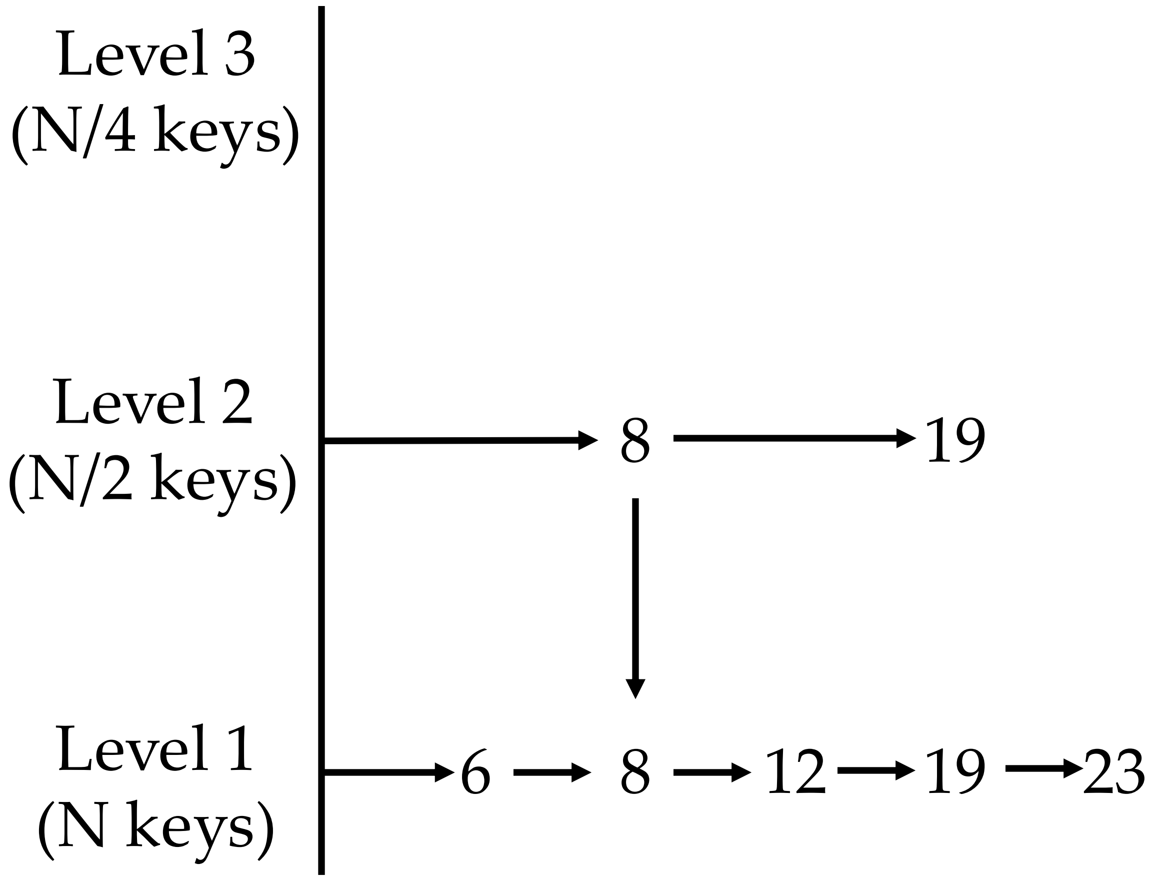
Skip Lists

Let's insert a value 19 and say coin toss is 1, so we move to Level 2.



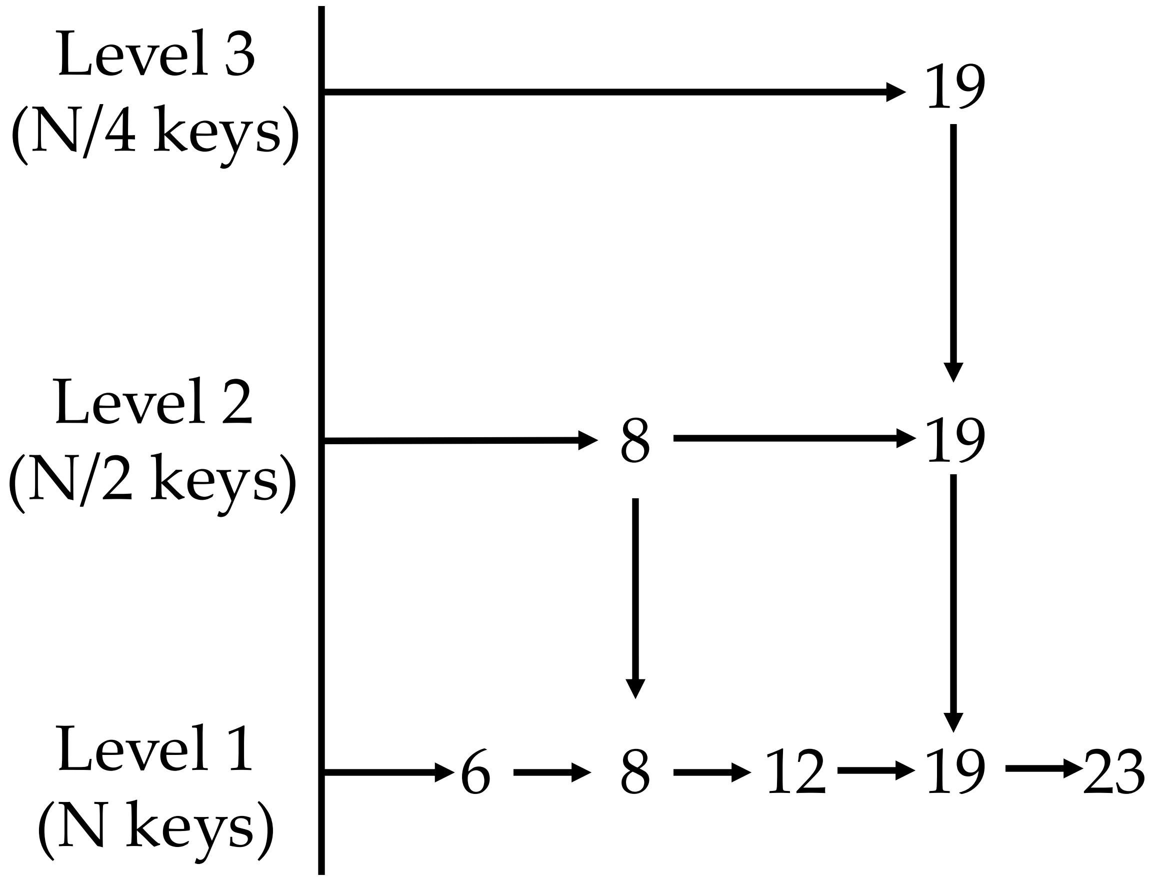
Skip Lists

Say coin toss is again 1, so we move to Level 3.



Skip Lists

Now, we insert at Level 3.



Search in Skip Lists

- To search a key in the skip list:
  - Start from the top most level.
  - Then, you need to search only in the nodes at the left or right side.
  - Probability of reduction of search space at each level is by half.

Deletion in Skip Lists

- To delete in a skip list, do it carefully.
  - We first mark the entry to be deleted → add a tombstone.
  - Then start deleting the entry from the topmost level.