Database Processing
CS 451 / 551

Lecture 5:

Searching and Indexing: Part 2

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

UNIVERSITY OF OREGON

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

Start collaborating with your groups!

Quiz 1: Oct 16, 2025 (in class)

Last Class

- We discussed sequential indexes: sparse, dense, multi-level.

- What are the challenges with these indexes?
  - A lot of file reorganization is needed when adding or deleting a record.
  - Can we avoid the reorganization? Yes, but
  - Then records are no longer mapped sequentially on the disk.

- Can we do better?

How to determine a Good Index?

- A good index should help to search a record fast!
- Characteristics of a good index:
  - **Access Types:** Supports accessing a particular record (point query) and/or records within a specified range (range query).
  - **Access Time:** Time to find a particular record.
  - **Insertion Time:** Time to insert a new record in the index (includes time to find the right place to insert).
  - **Deletion Time:** Time to delete a new record in the index (includes time to find the item to be deleted).
  - **Space Overhead:** The space consumed by the index.

A More desirable Index Structure

- Should ensure minimal reorganization.

- Should support sequential data access from disk.

B^+-Tree

B+-Tree

- Another tree from the family of Balanced Trees.

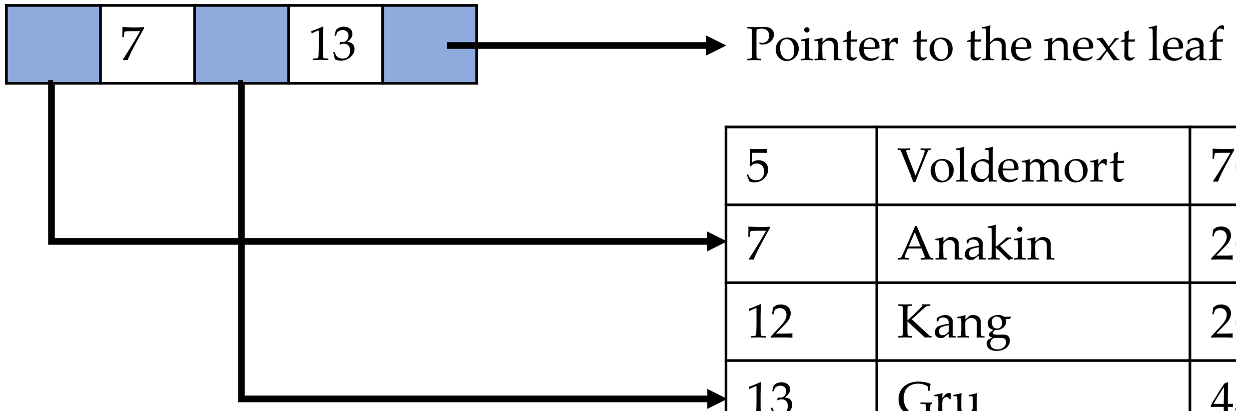
- Three types of nodes: root, internal nodes, and leaf nodes.

- Every leaf node is at the same height.

- Give a value $\boldsymbol{n}$, each internal node has:
  - $\boldsymbol{k}$ children
  - $\boldsymbol{k-1}$ search keys
  - where, $\boldsymbol{k}$ is between $\lceil \boldsymbol{n/2} \rceil$ to $\boldsymbol{n}$.

- Root can have less than $\lceil \boldsymbol{n/2} \rceil$ children but should have at least 2 children if there are more than one node in the tree.

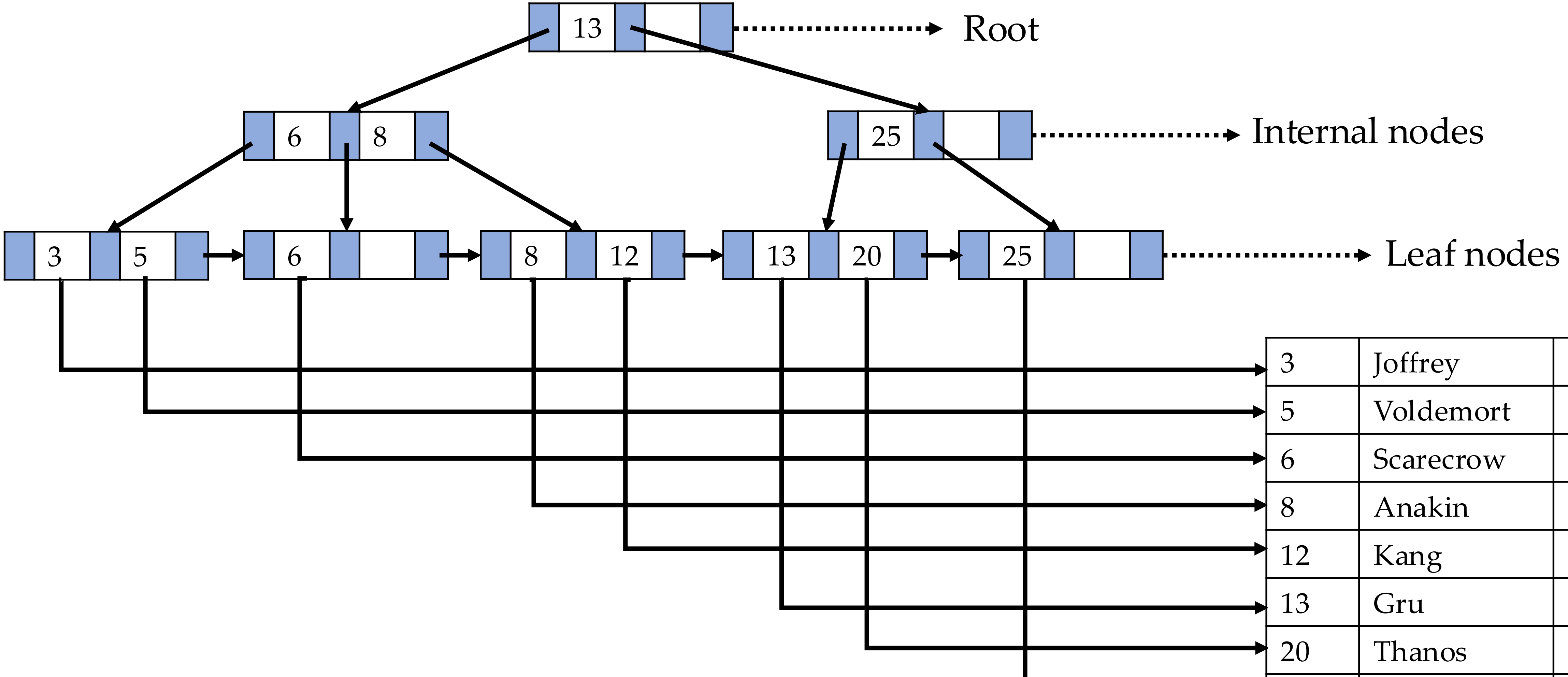
B+-Tree Leaf Node Structure



5	Voldemort	70	400
7	Anakin	20	200
12	Kang	20	500
13	Gru	45	100
20	Thanos	100	400

Internal nodes also have similar structure, except they point to other tree nodes

B^+-Tree At a Glance



3	Joffrey	18	600
5	Voldemort	70	400
6	Scarecrow	30	150
8	Anakin	20	200
12	Kang	20	500
13	Gru	45	100
20	Thanos	100	400
25	Joker	66	200

File

B^+ Tree with ID used for indexing

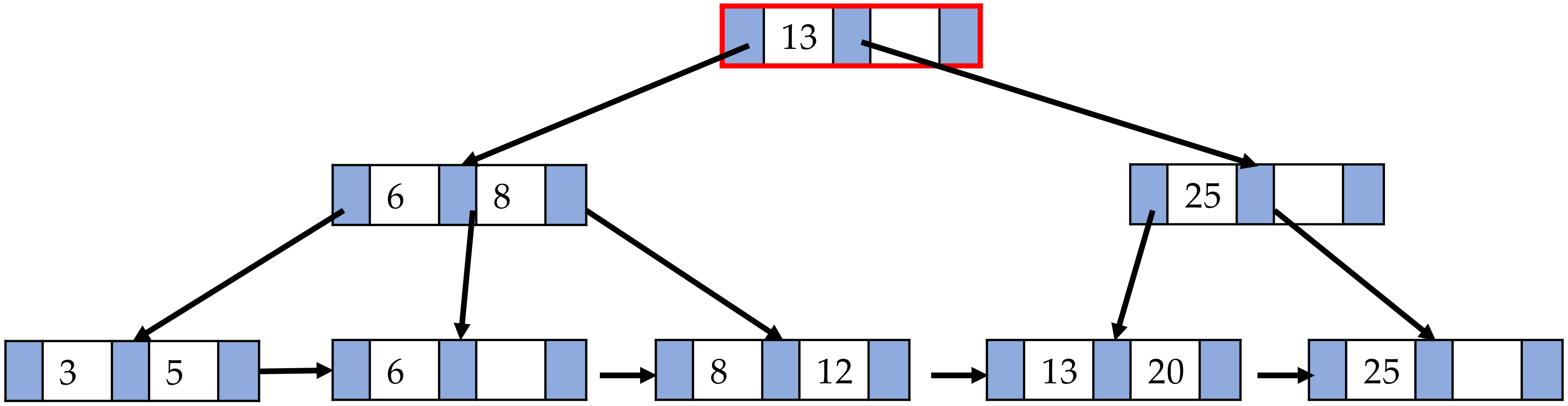
Searching data in B^+-Tree

- Notice that the keys are stored in B+ tree in a **sorted manner**.

- We claim that the data is stored in B+ tree in sorted order because if you perform an in-order traversal, then you will get a sorted list.

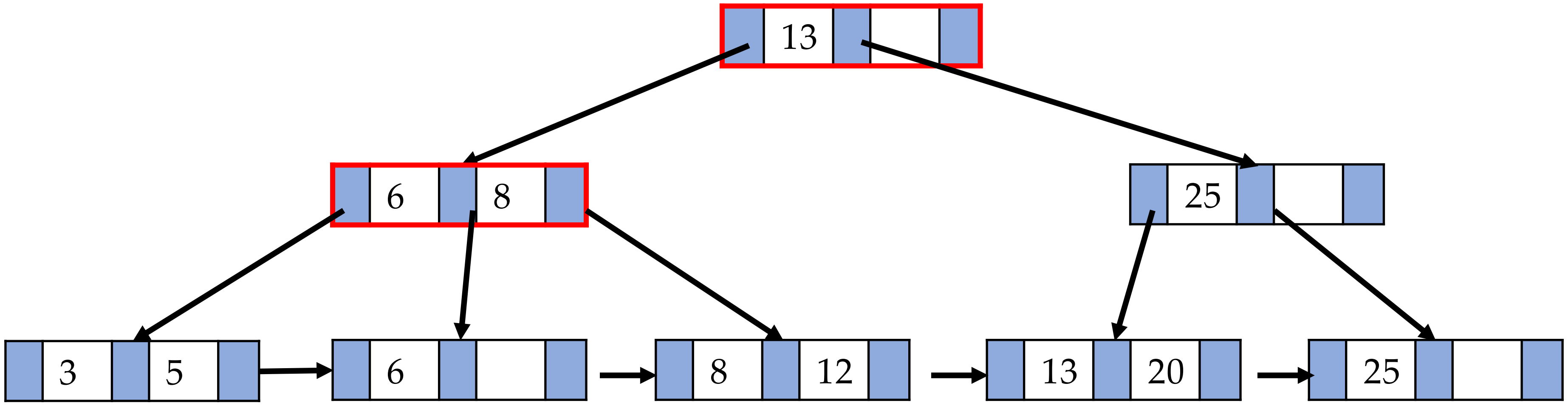
Searching data in B^+-Tree

- Let's run an **in-order traversal**, where we will only output data in the leaf nodes.



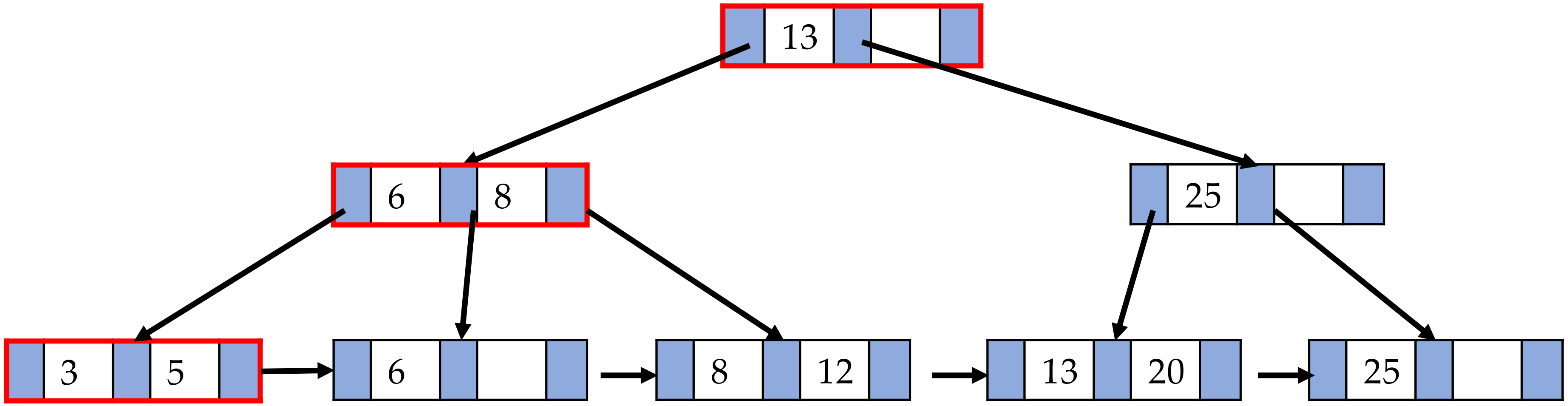
Searching data in B^{+}-Tree

- Let's run an in-order traversal, where we will only output data in the leaf nodes.



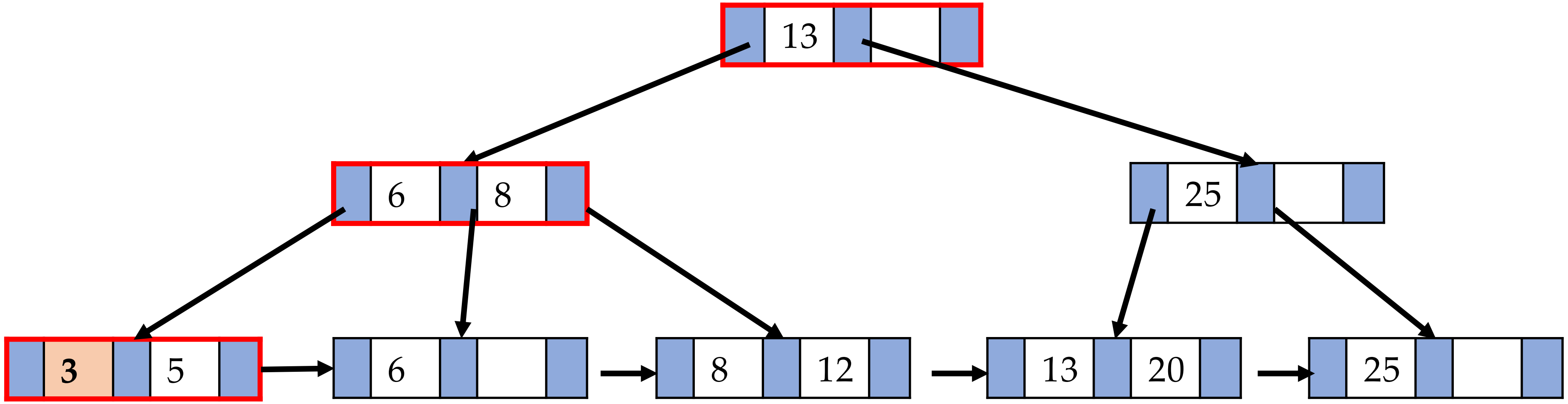
Searching data in B+-Tree

- Let's run an in-order traversal, where we will only output data in the leaf nodes.



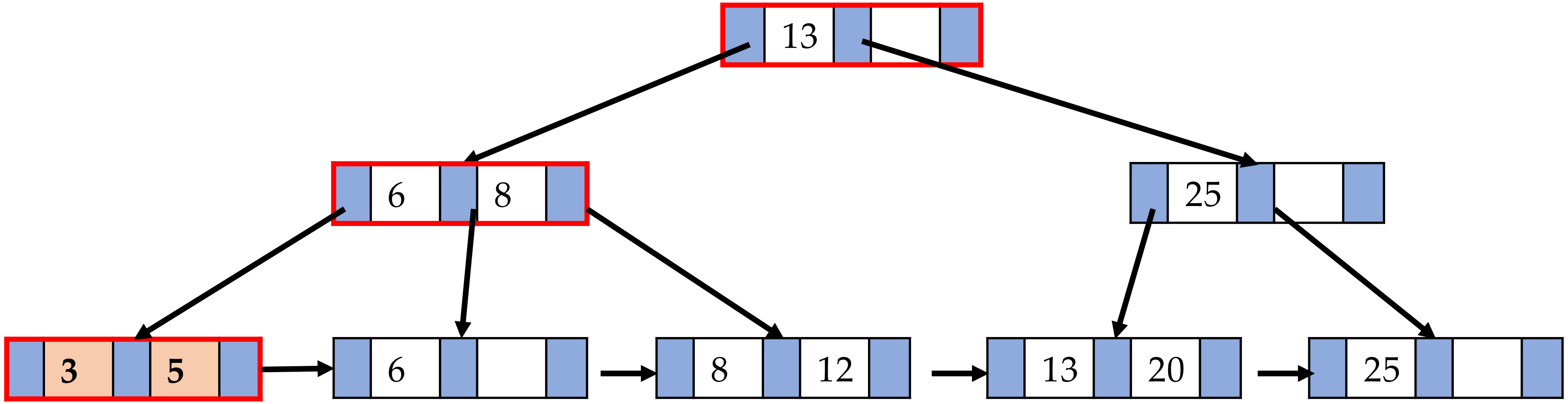
Searching data in B+-Tree

- Let's run an in-order traversal, where we will only output data in the leaf nodes.



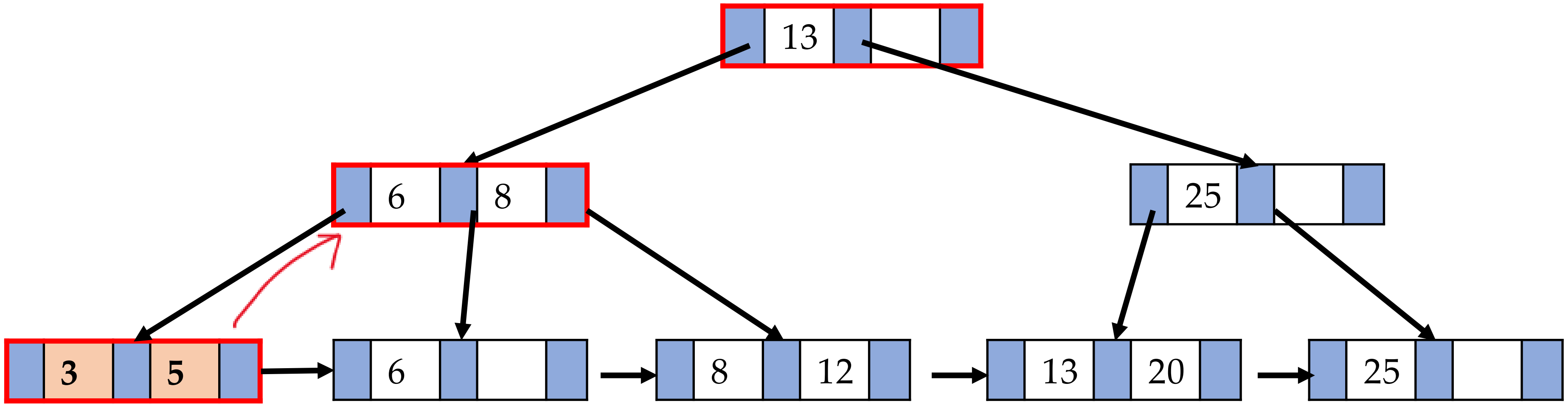
Searching data in B^+-Tree

- Let's run an in-order traversal, where we will only output data in the leaf nodes.



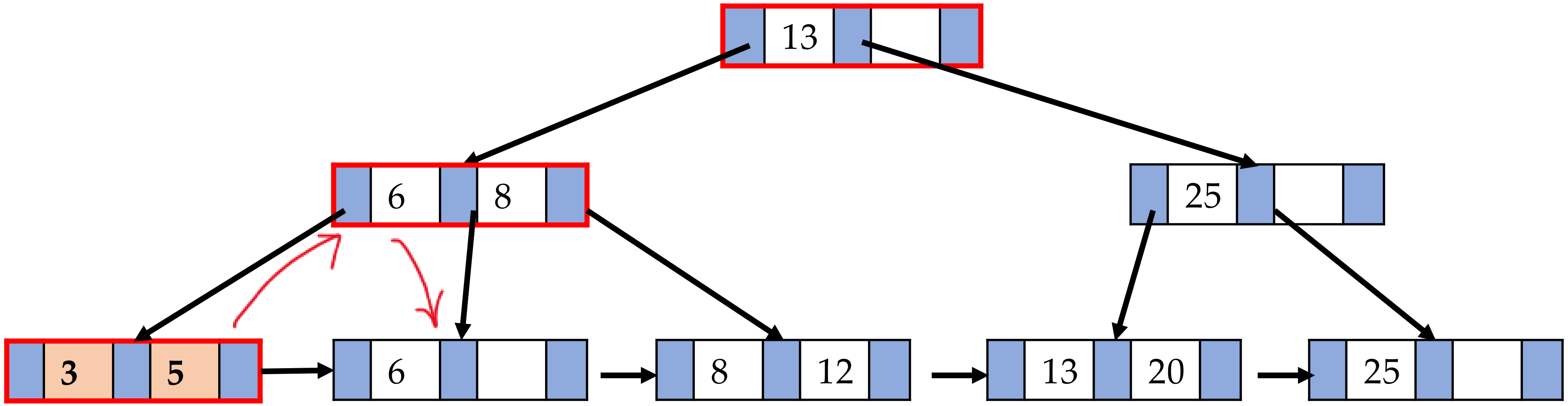
Searching data in B+-Tree

- Let's run an in-order traversal, where we will only output data in the leaf nodes.



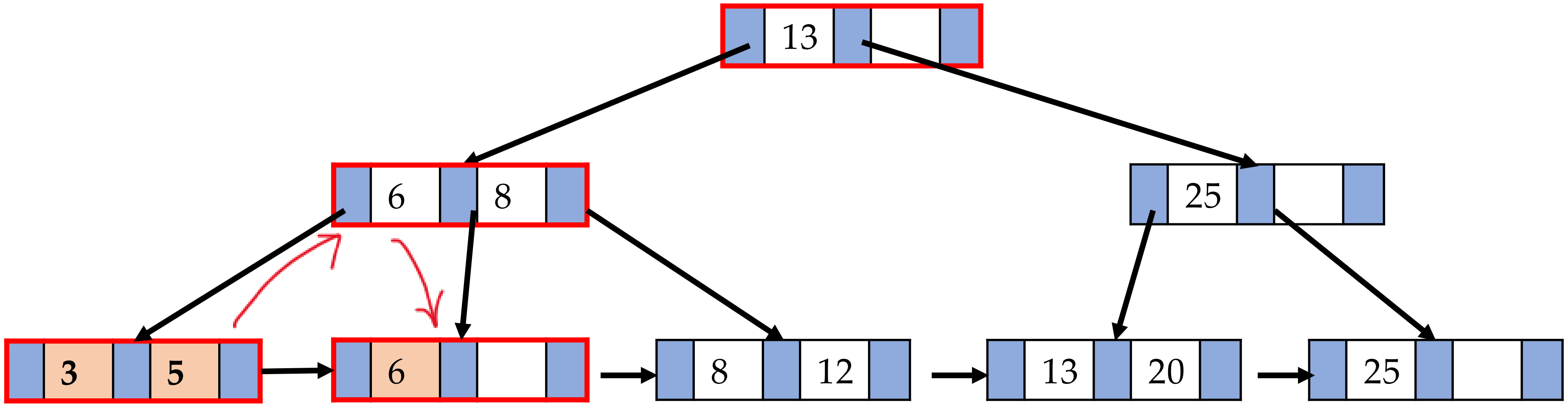
Searching data in B+-Tree

- Let's run an in-order traversal, where we will only output data in the leaf nodes.



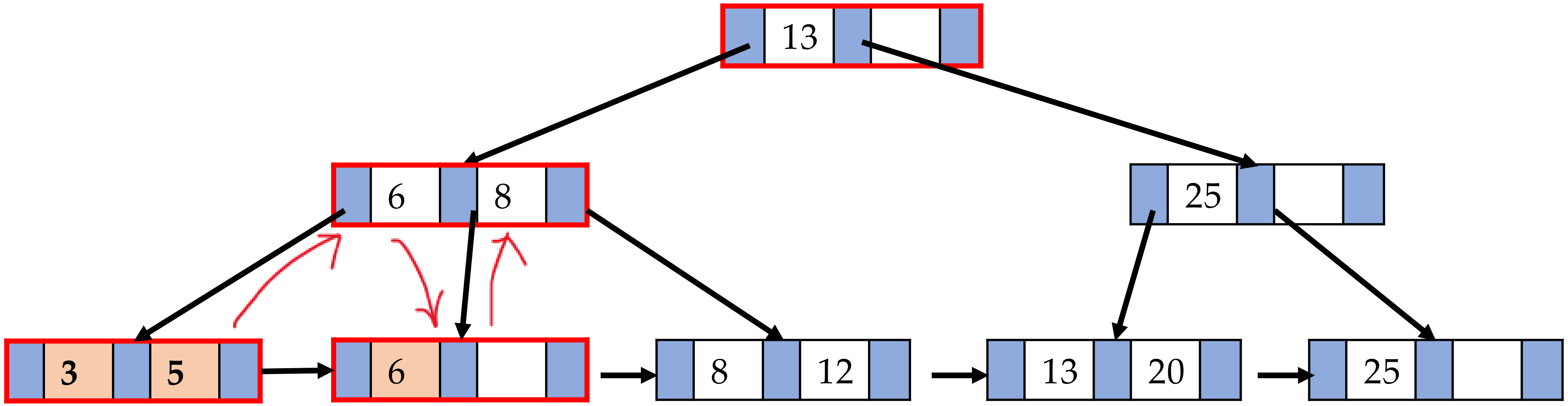
Searching data in B⁺-Tree

- Let's run an in-order traversal, where we will only output data in the leaf nodes.



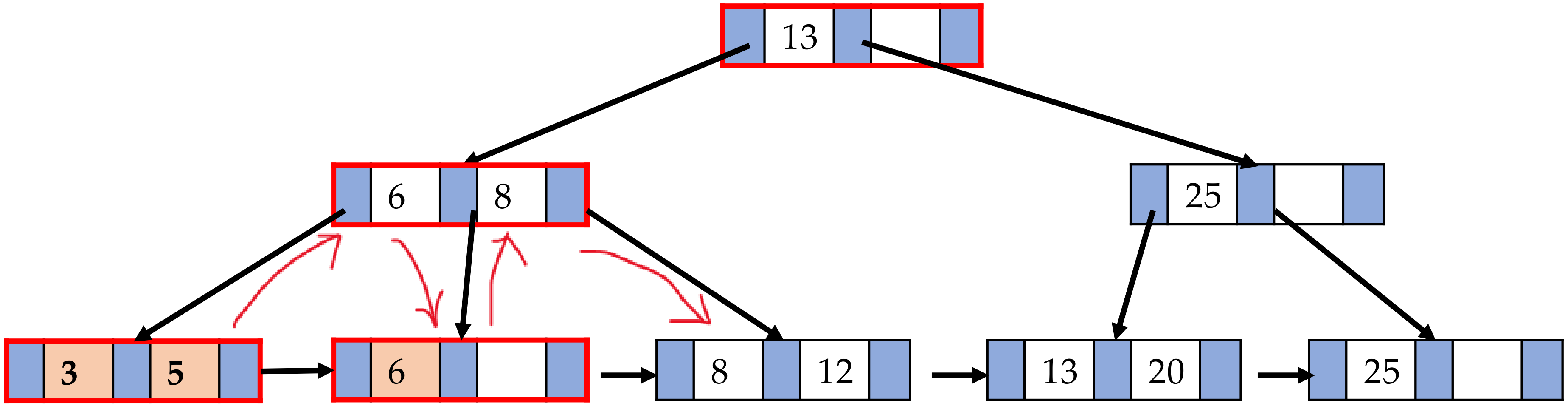
Searching data in B+-Tree

- Let's run an in-order traversal, where we will only output data in the leaf nodes.



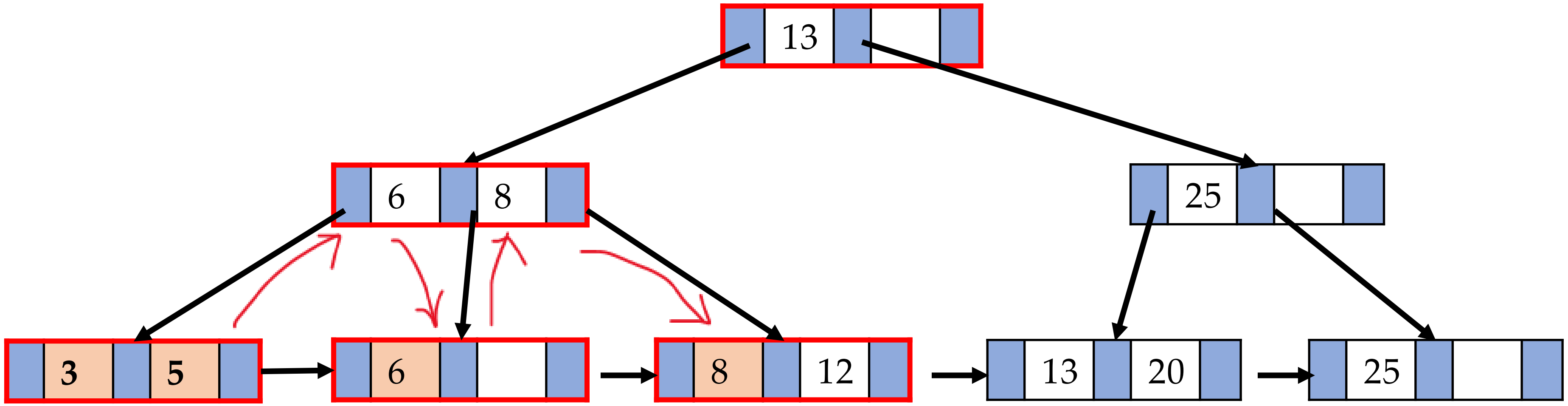
Searching data in B+-Tree

- Let's run an in-order traversal, where we will only output data in the leaf nodes.



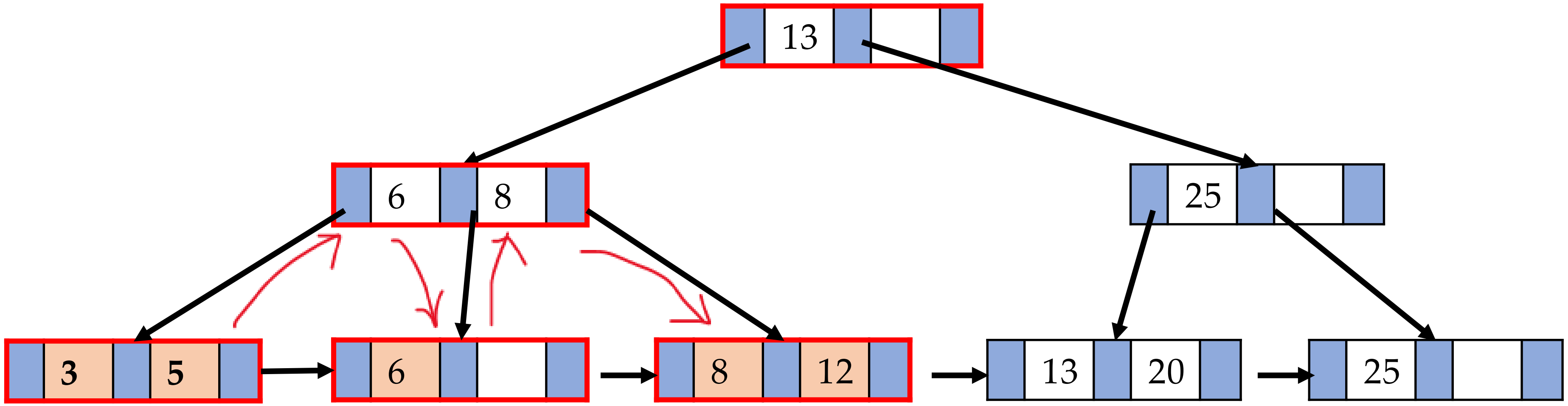
Searching data in B+-Tree

- Let's run an in-order traversal, where we will only output data in the leaf nodes.



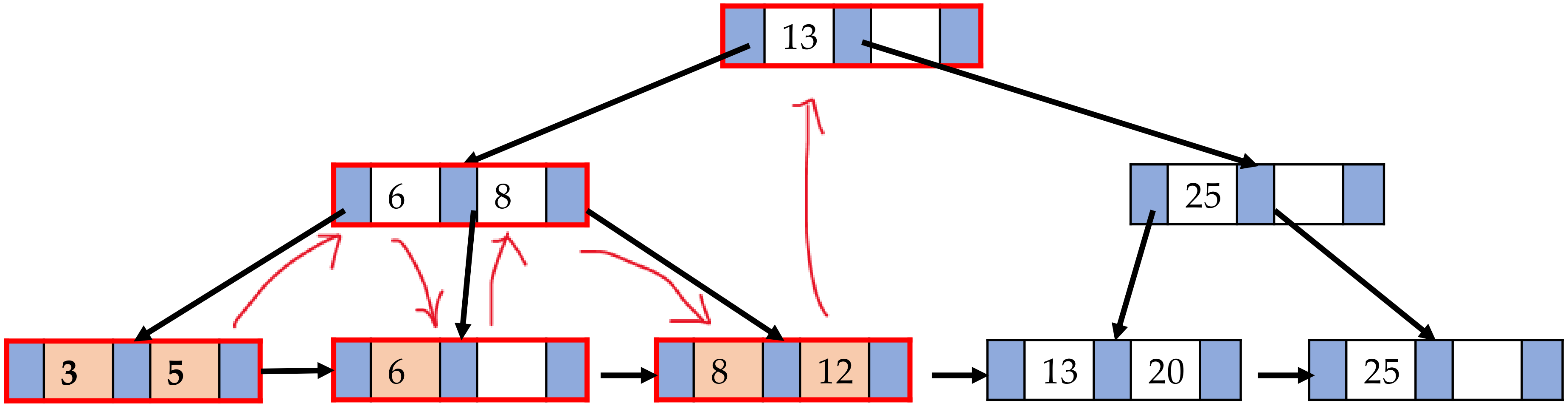
Searching data in B^+-Tree

- Let's run an in-order traversal, where we will only output data in the leaf nodes.



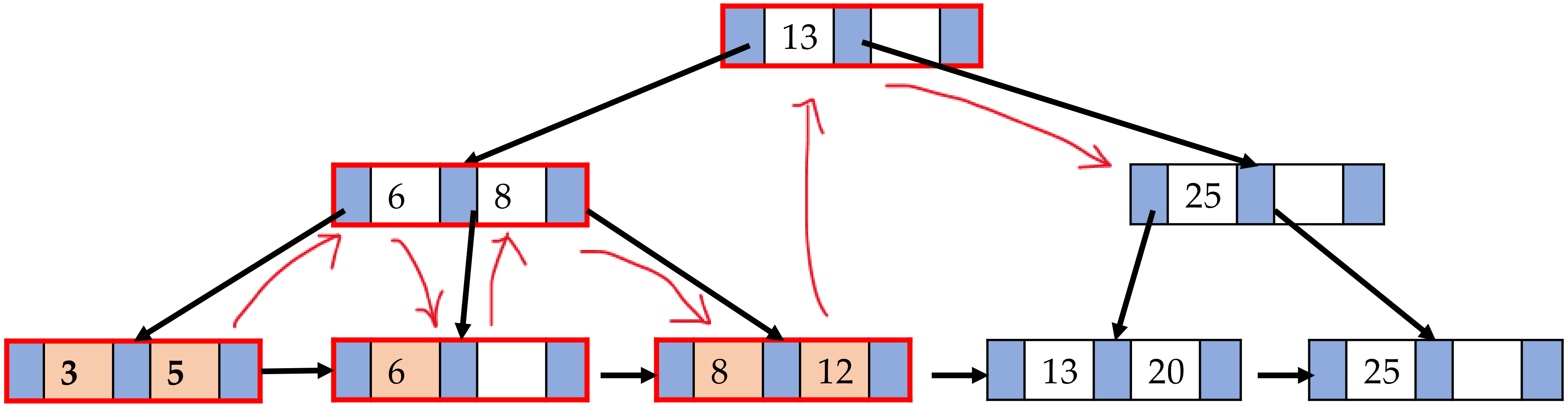
Searching data in B+-Tree

- Let's run an in-order traversal, where we will only output data in the leaf nodes.



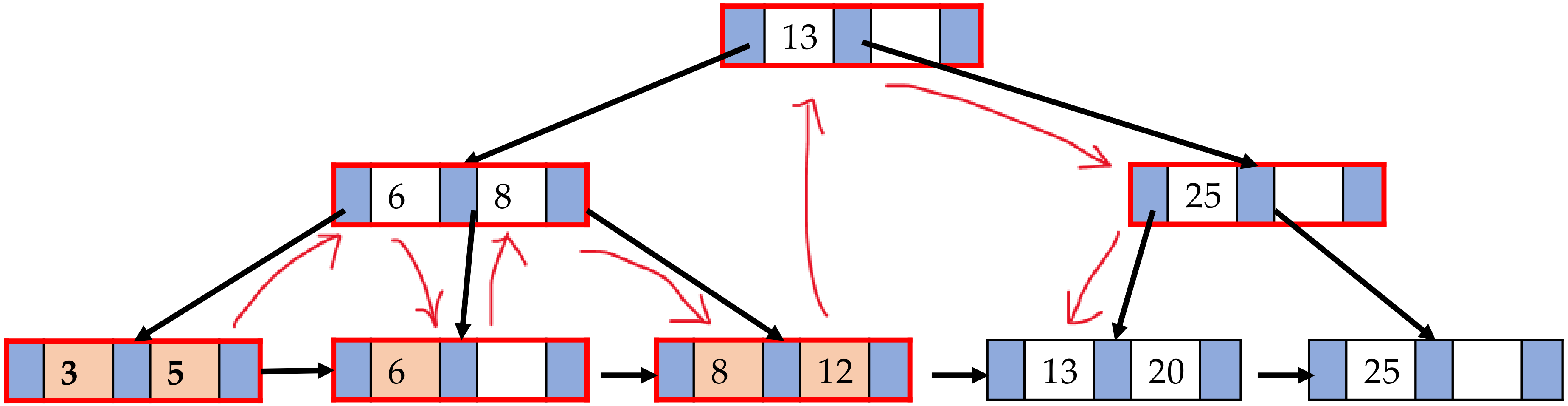
Searching data in B^+-Tree

- Let's run an in-order traversal, where we will only output data in the leaf nodes.



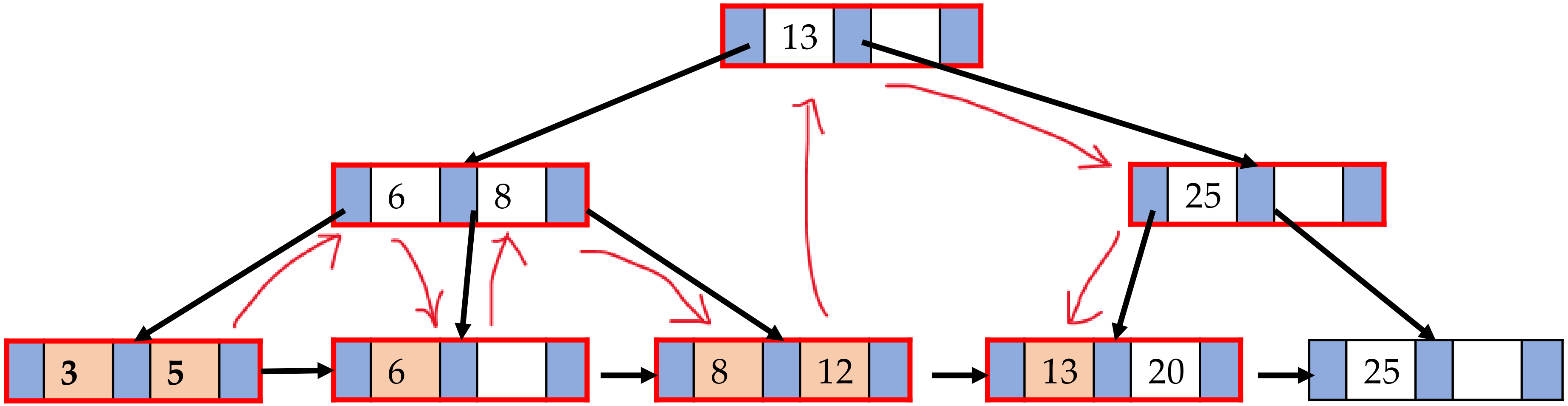
Searching data in B$^+$-Tree

- Let's run an in-order traversal, where we will only output data in the leaf nodes.



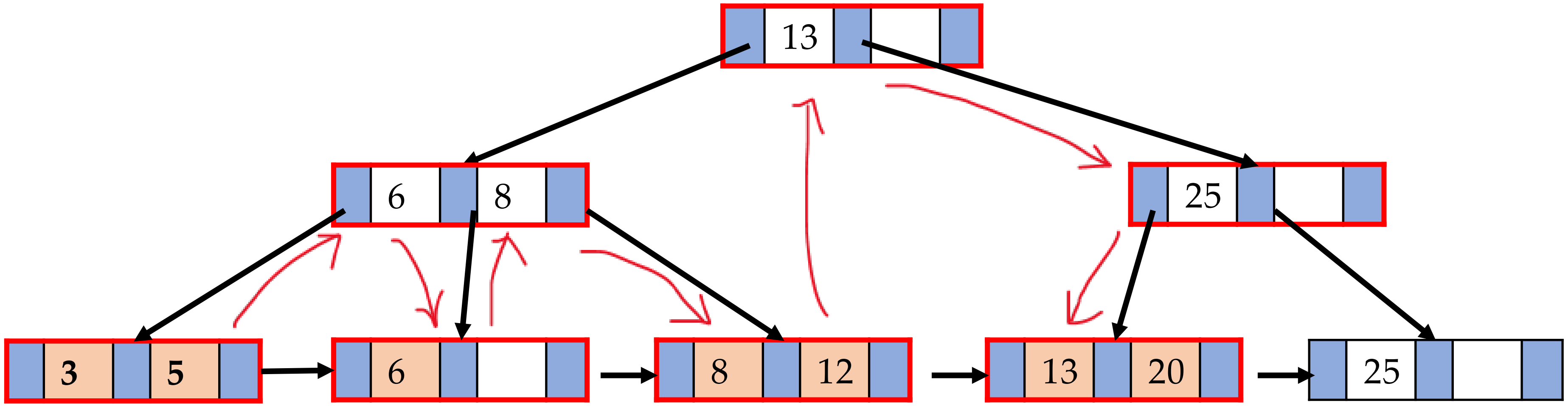
Searching data in B^+-Tree

- Let's run an in-order traversal, where we will only output data in the leaf nodes.



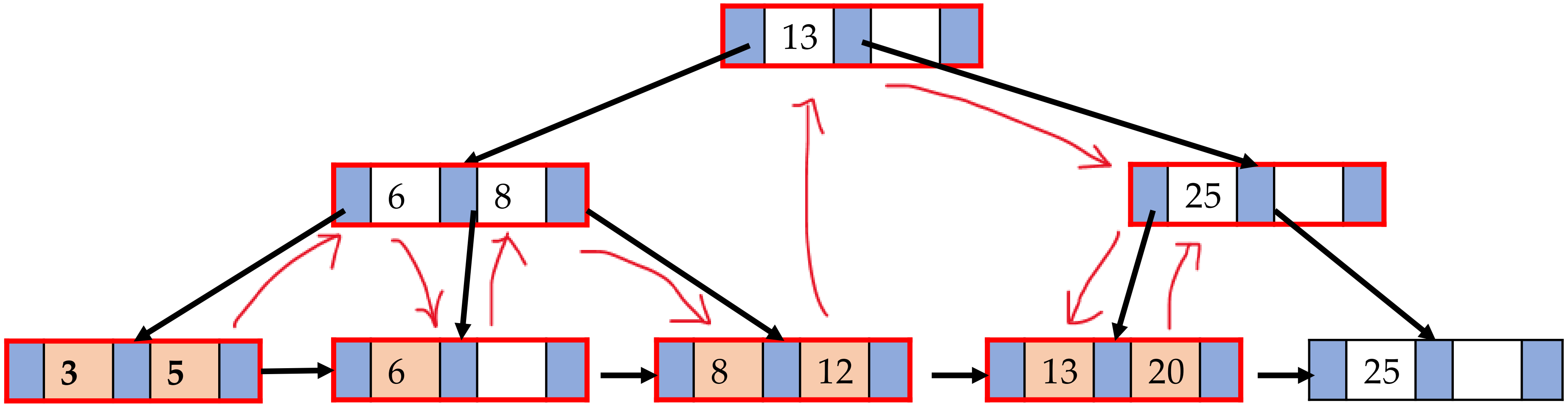
Searching data in B^+-Tree

- Let's run an in-order traversal, where we will only output data in the leaf nodes.



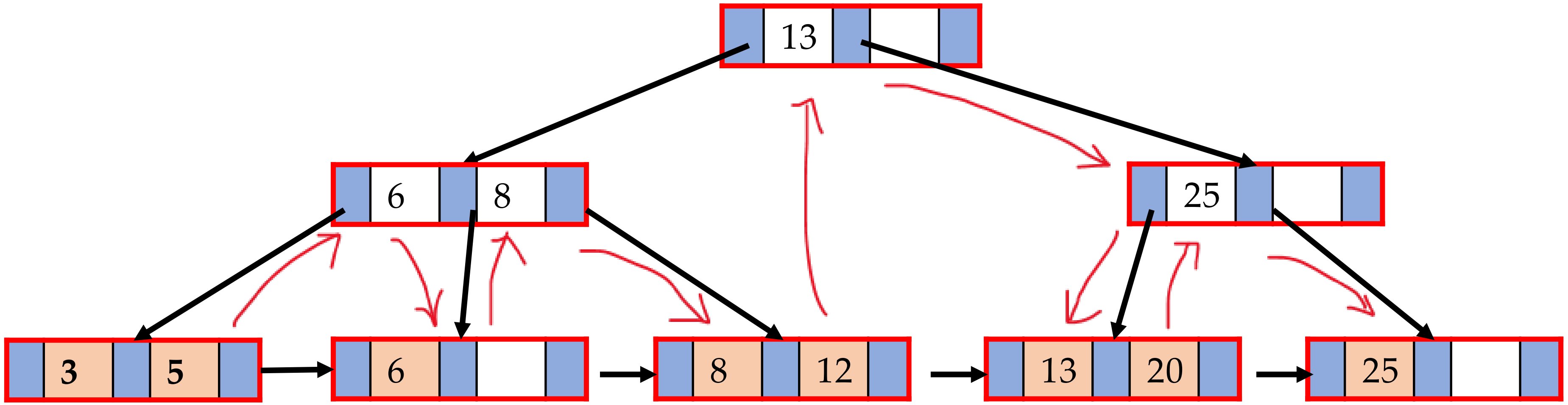
Searching data in B+-Tree

- Let's run an in-order traversal, where we will only output data in the leaf nodes.



Searching data in B$^+$-Tree

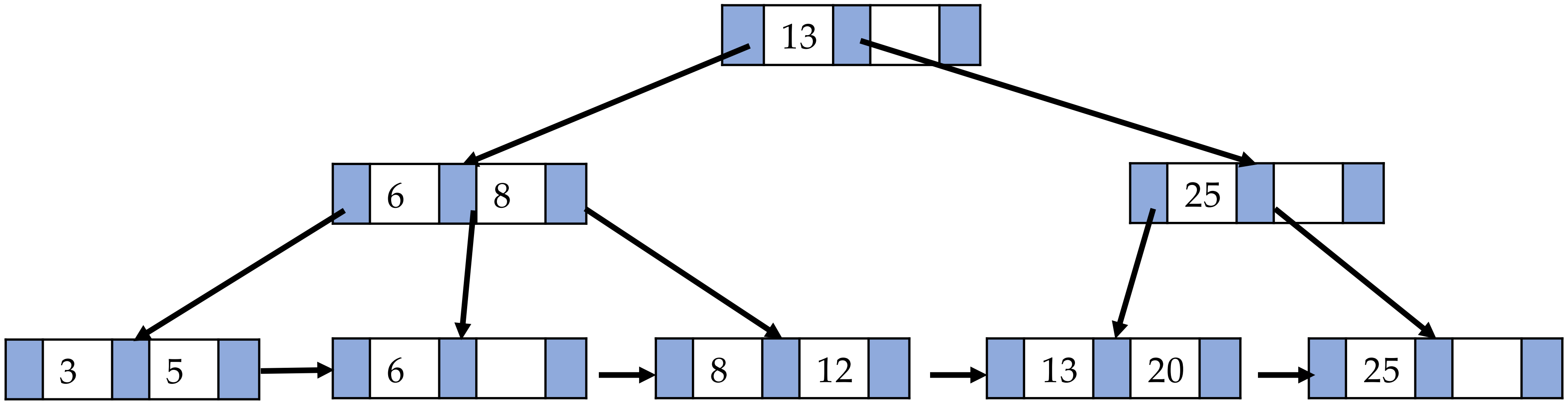
- Let's run an in-order traversal, where we will only output data in the leaf nodes.



Searching data in B+-Tree

- Now, lets try to search a key → Say we want to search key 12.
  - We need to traverse the tree in the in-order fashion.

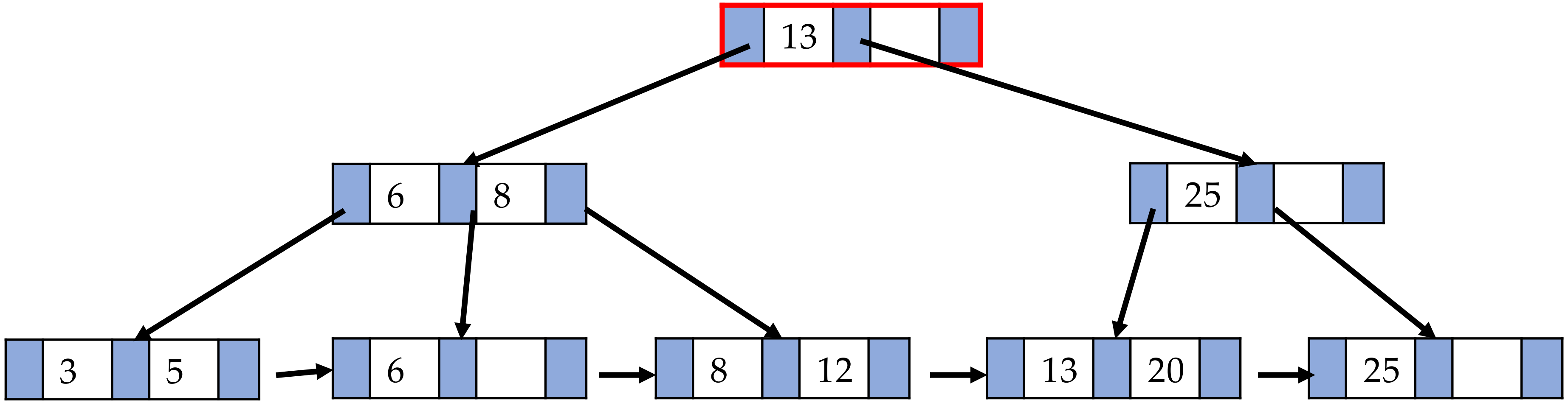
  - Stop traversing if one of the following three cases occur:
    - Key is found!
    - You encounter a Key greater than the search key.
    - You have reached the last key or leaf node of the tree.



Searching data in B+-Tree

- Now, lets try to **search a key** → Say we want to **search key 12**.
  - We need to traverse the tree in the in-order fashion.

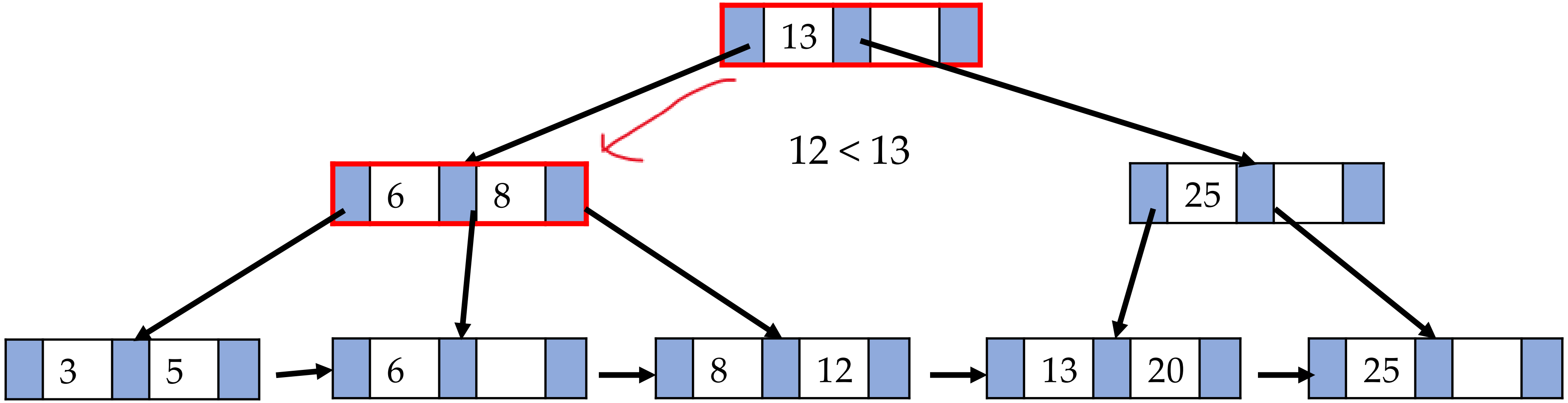
  - **Stop traversing** if one of the following three cases occur:
    - Key is found!
    - You encounter a Key greater than the search key.
    - You have reached the last key or leaf node of the tree.



Searching data in B+-Tree

- Now, lets try to search a key → Say we want to search key 12.
  - We need to traverse the tree in the in-order fashion.

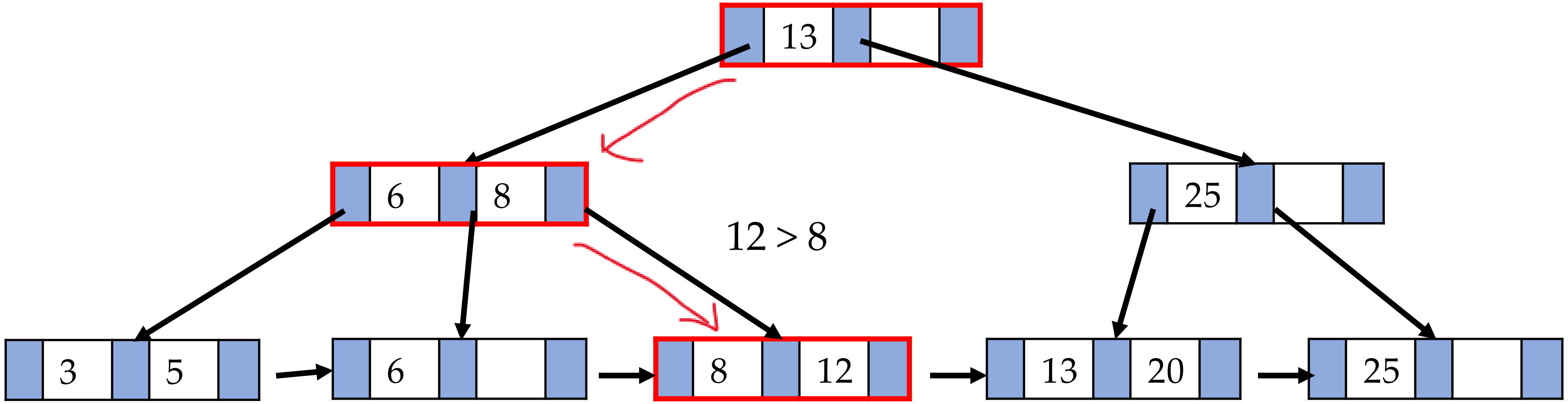
  - Stop traversing if one of the following three cases occur:
    - Key is found!
    - You encounter a Key greater than the search key.
    - You have reached the last key or leaf node of the tree.



Searching data in B+-Tree

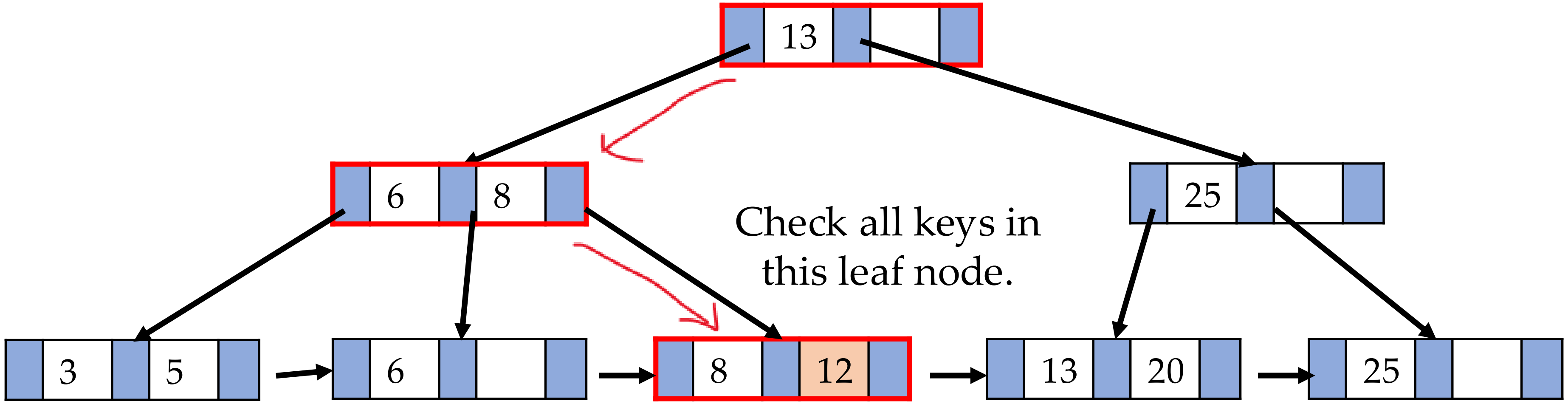
- Now, lets try to search a key → Say we want to search key 12.
  - We need to traverse the tree in the in-order fashion.

  - Stop traversing if one of the following three cases occur:
    - Key is found!
    - You encounter a Key greater than the search key.
    - You have reached the last key or leaf node of the tree.



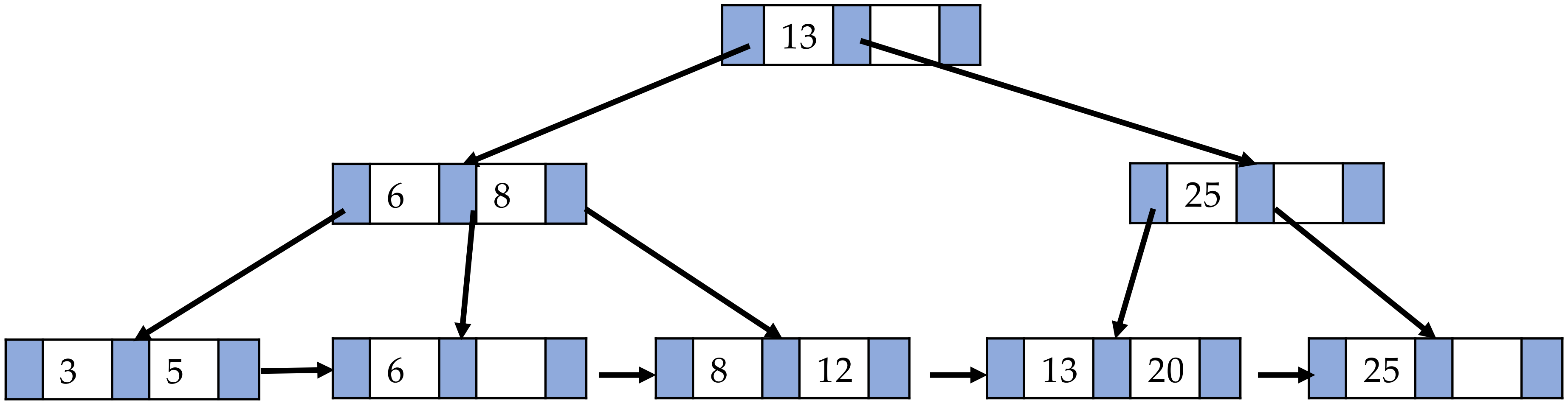
Searching data in B^+-Tree

- Now, lets try to search a key → Say we want to search key 12.
  - We need to traverse the tree in the in-order fashion.
  - Stop traversing if one of the following three cases occur:
    - Key is found!
    - You encounter a Key greater than the search key.
    - You have reached the last key or leaf node of the tree.



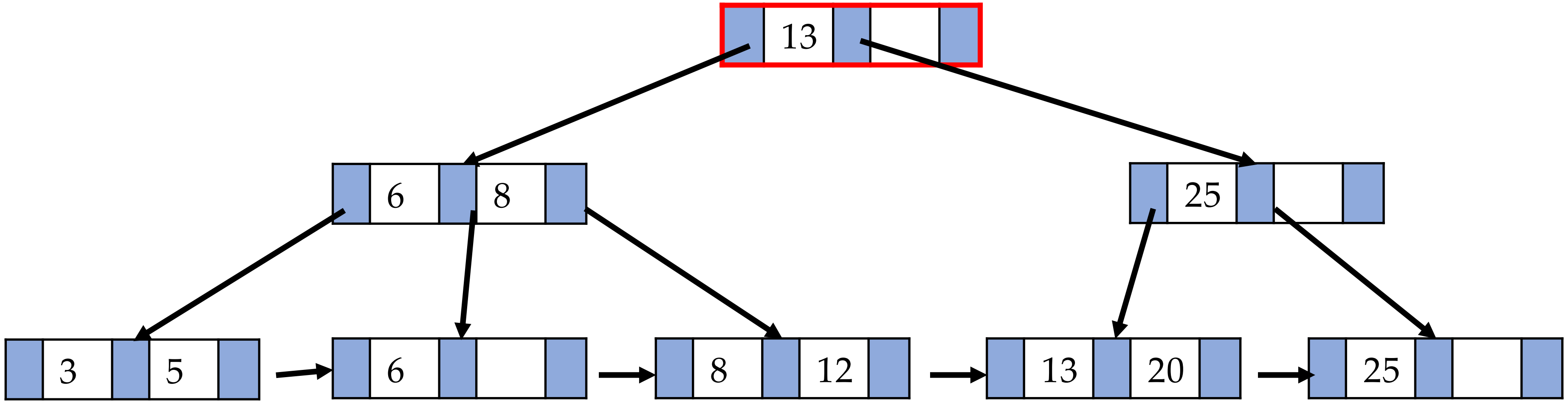
Searching data in B$^+$-Tree

- What we just did was a **Point Query**, where I wanted to search a specific item.
- Say we want to search a range of keys (**Range Query**) → **Keys from 12 to 24**.
  - We need to traverse the tree in the in-order fashion to reach the first key in the range, that is, first leaf node.
  - Then, perform linearly scan → follow the leaf pointers till you hit the last key or a key greater than the range.



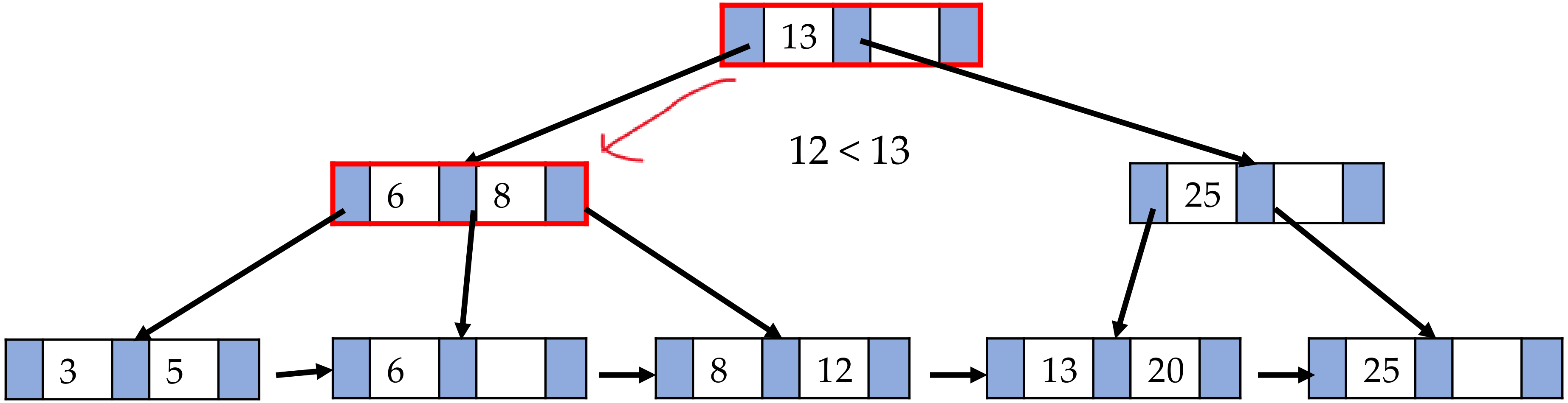
Searching data in B+-Tree

- What we just did was a **Point Query**, where I wanted to search a specific item.
- Say we want to search a range of keys (**Range Query**) → **Keys from 12 to 24**.
  - We need to traverse the tree in the in-order fashion to reach the first key in the range, that is, first leaf node.
  - Then, perform linearly scan → follow the leaf pointers till you hit the last key or a key greater than the range.



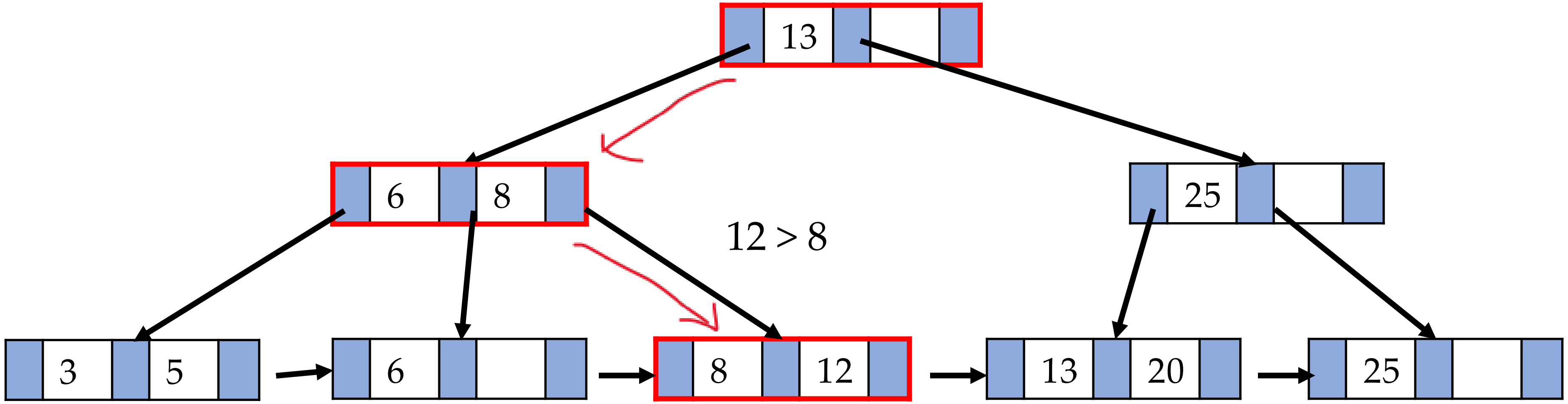
Searching data in B⁺-Tree

- What we just did was a **Point Query**, where I wanted to search a specific item.
- Say we want to search a range of keys (**Range Query**) → **Keys from 12 to 24**.
  - We need to traverse the tree in the in-order fashion to reach the first key in the range, that is, first leaf node.
  - Then, perform linearly scan → follow the leaf pointers till you hit the last key or a key greater than the range.



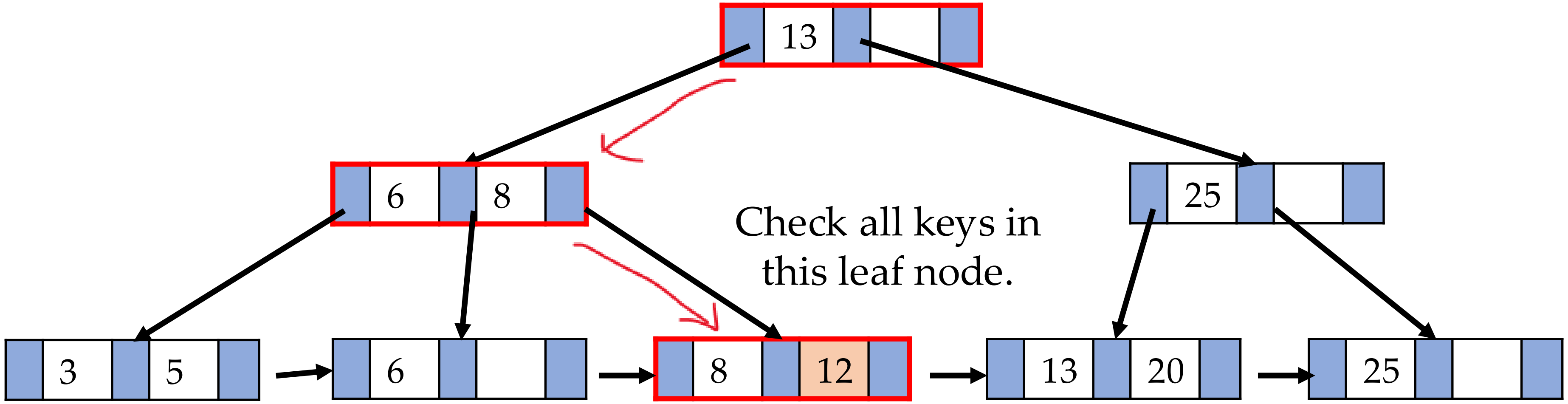
Searching data in B+-Tree

- What we just did was a **Point Query**, where I wanted to search a specific item.
- Say we want to search a range of keys (**Range Query**) → **Keys from 12 to 24**.
  - We need to traverse the tree in the in-order fashion to reach the first key in the range, that is, first leaf node.
  - Then, perform linearly scan → follow the leaf pointers till you hit the last key or a key greater than the range.



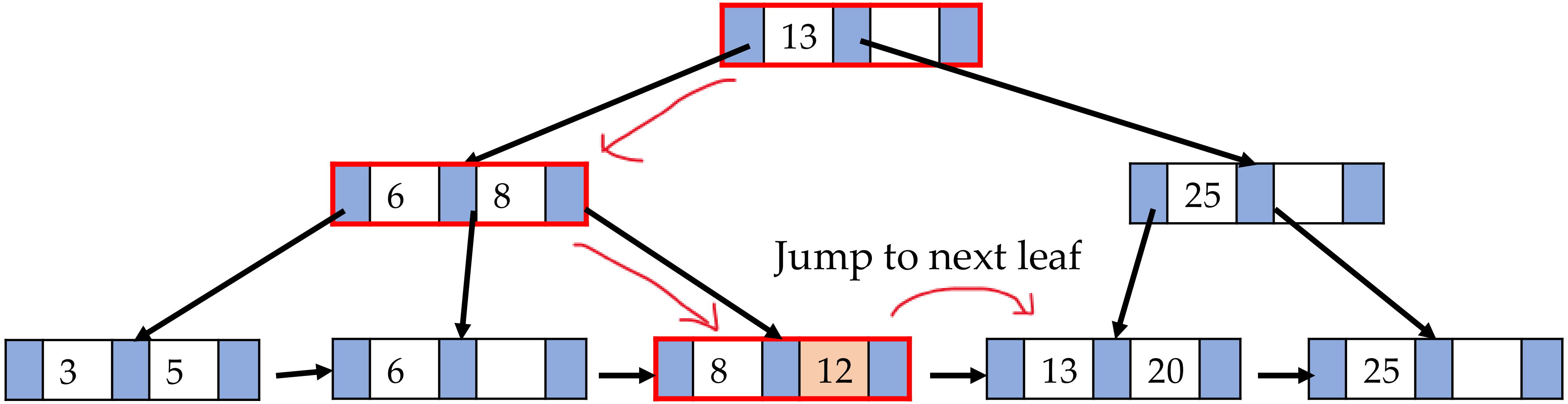
Searching data in B+-Tree

- What we just did was a **Point Query**, where I wanted to search a specific item.
- Say we want to search a range of keys (**Range Query**) → **Keys from 12 to 24**.
  - We need to traverse the tree in the in-order fashion to reach the first key in the range, that is, first leaf node.
  - Then, perform linearly scan → follow the leaf pointers till you hit the last key or a key greater than the range.



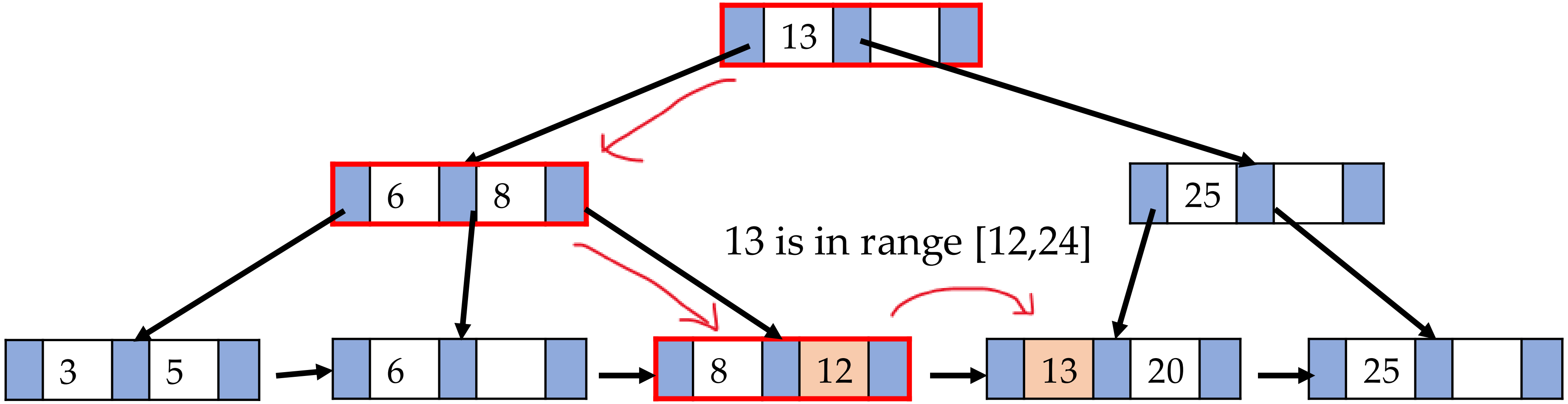
Searching data in B$^+$-Tree

- What we just did was a **Point Query**, where I wanted to search a specific item.
- Say we want to search a range of keys (**Range Query**) → **Keys from 12 to 24**.
  - We need to traverse the tree in the in-order fashion to reach the first key in the range, that is, first leaf node.
  - Then, perform linearly scan → follow the leaf pointers till you hit the last key or a key greater than the range.



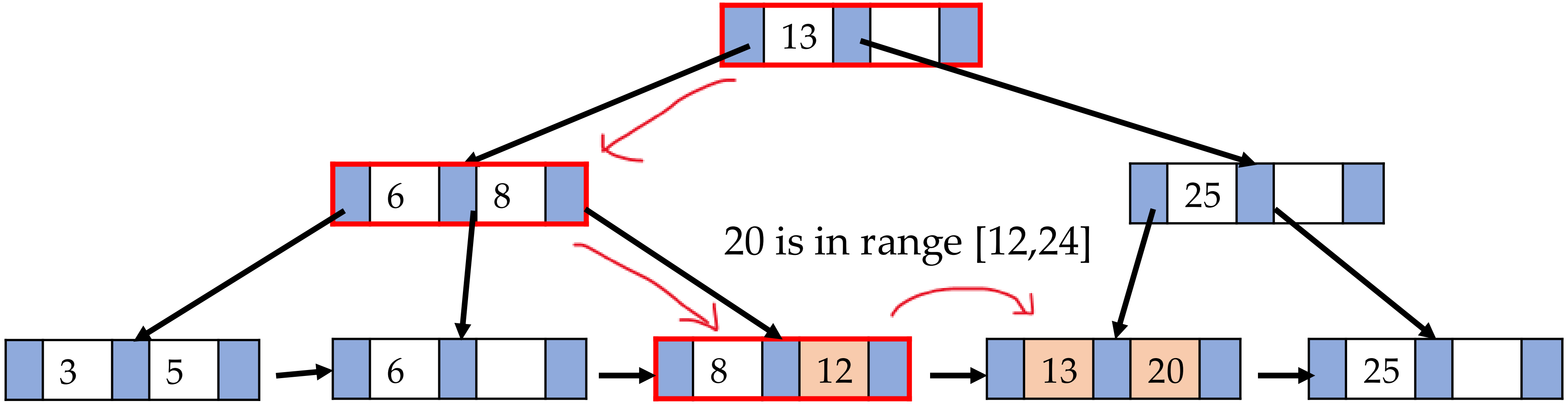
Searching data in B+-Tree

- What we just did was a **Point Query**, where I wanted to search a specific item.
- Say we want to search a range of keys (**Range Query**) → **Keys from 12 to 24**.
  - We need to traverse the tree in the in-order fashion to reach the first key in the range, that is, first leaf node.
  - Then, perform linearly scan → follow the leaf pointers till you hit the last key or a key greater than the range.



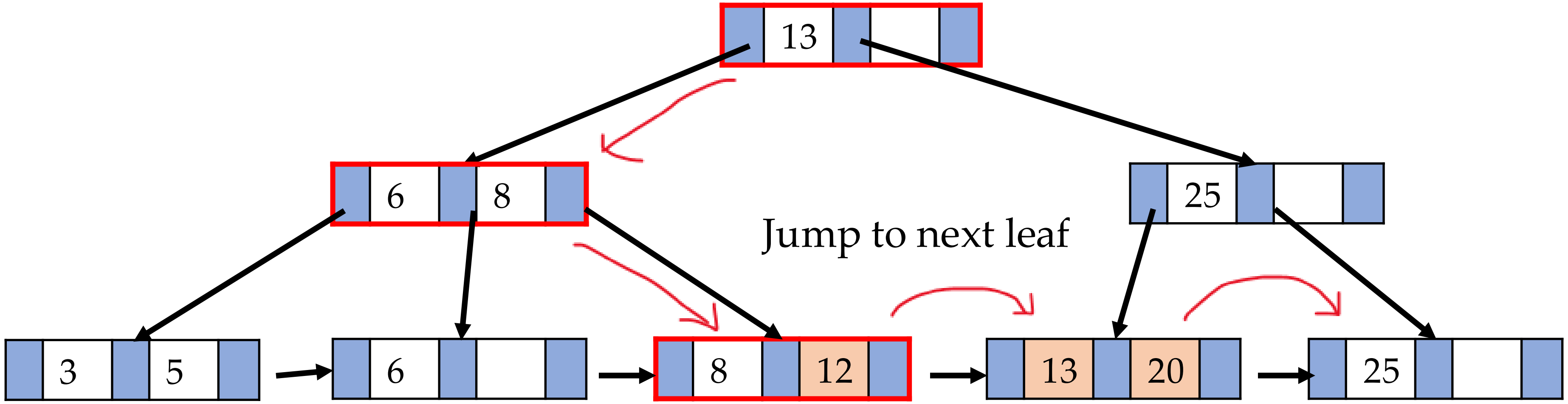
Searching data in B$^+$-Tree

- What we just did was a **Point Query**, where I wanted to search a specific item.
- Say we want to search a range of keys (**Range Query**) → **Keys from 12 to 24**.
  - We need to traverse the tree in the in-order fashion to reach the first key in the range, that is, first leaf node.
  - Then, perform linearly scan → follow the leaf pointers till you hit the last key or a key greater than the range.



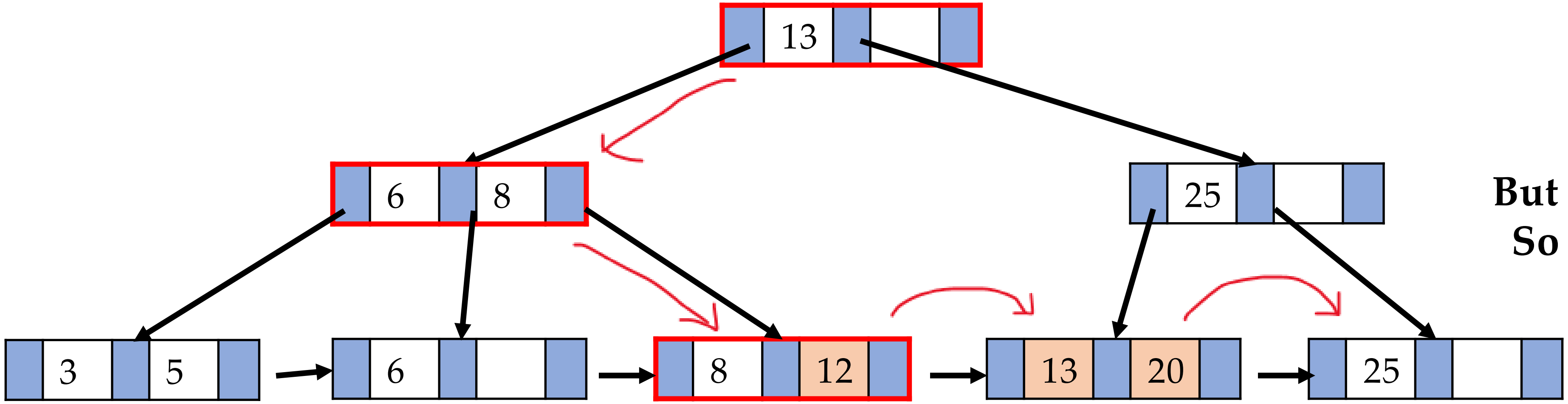
Searching data in B^{+}-Tree

- What we just did was a **Point Query**, where I wanted to search a specific item.
- Say we want to search a range of keys (**Range Query**) → **Keys from 12 to 24**.
    - We need to traverse the tree in the in-order fashion to reach the first key in the range, that is, first leaf node.
    - Then, perform linearly scan → follow the leaf pointers till you hit the last key or a key greater than the range.



Searching data in B$^+$-Tree

- What we just did was a **Point Query**, where I wanted to search a specific item.
- Say we want to search a range of keys (**Range Query**) → **Keys from 12 to 24**.
  - We need to traverse the tree in the in-order fashion to reach the first key in the range, that is, first leaf node.
  - Then, perform linearly scan → follow the leaf pointers till you hit the last key or a key greater than the range.



But 25 is out of range, So terminate search

Search Complexity of B^+-Tree

Search Complexity of B^+-Tree

- If each node can have n search keys (pointers), and total N records in the tree,
  - $\boldsymbol{O(log_{n/2}\ N)}$ is the length of the path.

- Ex: If $n = 100$ and $N = 1{,}000{,}000$, only 4 nodes need to be accessed.
  - Only 4 blocks need to be read from disk.

- This is also an important **distinction between B^+-tree and Binary trees**.
  - We can design B^+-tree, where node size is large enough to be block size.
  - So one block fetch gives access to one node of the B^+-tree.

- Notice that **root is most frequently accessed**.
  - Place it in your database buffer, which will save lookup cost.

Insertions and Deletions in B^+-Tree

Insertions and Deletions in B^+-Tree

- Insertions and Deletions are slightly more complex.

- You may need to **split a node** or **merge two nodes**.

- Split and merge operations can be avoided if there is a space, or you are not violating the B^+-tree conditions.

- Remember; Give a value $\boldsymbol{n}$, each internal node has:
  - $\boldsymbol{k}$ children
  - $\boldsymbol{k-1}$ search keys
  - where, $\boldsymbol{k}$ is between $\lceil \boldsymbol{n/2} \rceil$ to $\boldsymbol{n}$.

- Lets look at a live demonstration.

Insertions and Deletions in B^+-Tree

30, 12, 56, **45**, 18, **16**, 10, **14**, 8, **6**, 90, 83, 67, 76, 49, 78,

56, 49, **67**, 83, 78, 90, 18, 30, 76,

Insertions and Deletions Complexity

- If each node can have n search keys (pointers), and total N records in the tree,
  - $\boldsymbol{O(log_{n/2}\ N)}$ is the number of I/O operations needed.
- Notice that insertion and deletion complexity is still same as search!
- This is the worst case complexity, on average fewer I/O operations are required.

Can we use B^+-tree for File organization?

- Till now, we used B+-tree for designing an index for our file.
- How about we use it to even organize our files.
- The leaf nodes of the B+-tree can store actual records.
- If each leaf has same size as the disk block, then one disk block I/O fetches necessary records.

Self Reading Task

- Difference between **B-tree and B^+-tree**.
- Disadvantages of B-tree when compared to B^+-tree?

Special Indices?

- Often, some attributes have only a small set of possible values.
  - Course with grades Pass or Fail.
  - Daily Attendance: Present or Absent

- For some attributes, we can create groups for their values.
  - Faculty Title: Assistant Prof., Associate Prof., Professor
  - Salary Payscale: L1 (< 100); L2 (100 - 300); L3 (300 – 500); L4 (> 500)

Bitmap Indices

- Often, some attributes have only a small set of possible values.
  - Course with grades Pass or Fail.
  - Daily Attendance: Present or Absent

- For some attributes, we can create groups for their values.
  - Faculty Title: Assistant Prof., Associate Prof., Professor
  - Salary Payscale: L1 (< 100); L2 (100 - 300); L3 (300 – 500); L4 (> 500)

- Constructing Bitmap indices is useful for such attributes.

- Each value is represented with the help of a bitmap.
  - Each record needs a sequential identifier.
  - The size of the bitmap is equal to number of records.

- One bitmap for each value!

Bitmap Indices

- Assume that the following is our table:
  - We can construct bitmaps for Grade and Payscale.

ID	Name	Grade	Payscale
1	Voldemort	P	L1
2	Anakin	P	L2
3	Kang	F	L1
4	Gru	F	L2
5	Thanos	P	L4
6	Joker	F	L3
7	Jeoffrey	P	L1

Bitmap for Grade

P							
F							

Bitmap for Payscale

L1							
L2							
L3							
L4							

Bitmap Indices

- Assume that the following is our table:
  - We can construct bitmaps for Grade and Payscale.

ID	Name	Grade	Payscale
1	**Voldemort**	**P**	**L1**
2	Anakin	P	L2
3	Kang	F	L1
4	Gru	F	L2
5	Thanos	P	L4
6	Joker	F	L3
7	Jeoffrey	P	L1

Bitmap for Grade

P	1						
F	0						

Bitmap for Payscale

L1	1						
L2	0						
L3	0						
L4	0						

Bitmap Indices

- Assume that the following is our table:
  - We can construct bitmaps for Grade and Payscale.

ID	Name	Grade	Payscale
1	Voldemort	P	L1
2	**Anakin**	**P**	**L2**
3	Kang	F	L1
4	Gru	F	L2
5	Thanos	P	L4
6	Joker	F	L3
7	Jeoffrey	P	L1

Bitmap for Grade

P	1	1					
F	0	0					

Bitmap for Payscale

L1	1	0					
L2	0	1					
L3	0	0					
L4	0	0					

Bitmap Indices

- Assume that the following is our table:
  - We can construct bitmaps for Grade and Payscale.

ID	Name	Grade	Payscale
1	Voldemort	P	L1
2	Anakin	P	L2
3	**Kang**	**F**	**L1**
4	Gru	F	L2
5	Thanos	P	L4
6	Joker	F	L3
7	Jeoffrey	P	L1

Bitmap for Grade

P	1	1	0				
F	0	0	1				

Bitmap for Payscale

L1	1	0	1				
L2	0	1	0				
L3	0	0	0				
L4	0	0	0				

Bitmap Indices

- Assume that the following is our table:
  - We can construct bitmaps for Grade and Payscale.

ID	Name	Grade	Payscale
1	Voldemort	P	L1
2	Anakin	P	L2
3	Kang	F	L1
4	**Gru**	**F**	**L2**
5	Thanos	P	L4
6	Joker	F	L3
7	Jeoffrey	P	L1

Bitmap for Grade

P	1	1	0	0			
F	0	0	1	1			

Bitmap for Payscale

L1	1	0	1	0			
L2	0	1	0	1			
L3	0	0	0	0			
L4	0	0	0	0			

Bitmap Indices

- Assume that the following is our table:
  - We can construct bitmaps for Grade and Payscale.

ID	Name	Grade	Payscale
1	Voldemort	P	L1
2	Anakin	P	L2
3	Kang	F	L1
4	Gru	F	L2
5	**Thanos**	**P**	**L4**
6	Joker	F	L3
7	Jeoffrey	P	L1

Bitmap for Grade

P	1	1	0	0	1		
F	0	0	1	1	0		

Bitmap for Payscale

L1	1	0	1	0	0		
L2	0	1	0	1	0		
L3	0	0	0	0	0		
L4	0	0	0	0	1		

Bitmap Indices

- Assume that the following is our table:
  - We can construct bitmaps for Grade and Payscale.

ID	Name	Grade	Payscale
1	Voldemort	P	L1
2	Anakin	P	L2
3	Kang	F	L1
4	Gru	F	L2
5	Thanos	P	L4
6	**Joker**	**F**	**L3**
7	Jeoffrey	P	L1

Bitmap for Grade

P	1	1	0	0	1	0	
F	0	0	1	1	0	1	

Bitmap for Payscale

L1	1	0	1	0	0	0	
L2	0	1	0	1	0	0	
L3	0	0	0	0	0	1	
L4	0	0	0	0	1	0	

Bitmap Indices

- Assume that the following is our table:
  - We can construct bitmaps for Grade and Payscale.

ID	Name	Grade	Payscale
1	Voldemort	P	L1
2	Anakin	P	L2
3	Kang	F	L1
4	Gru	F	L2
5	Thanos	P	L4
6	Joker	F	L3
7	**Jeoffrey**	**P**	**L1**

Bitmap for Grade

P	1	1	0	0	1	0	1
F	0	0	1	1	0	1	0

Bitmap for Payscale

L1	1	0	1	0	0	0	1
L2	0	1	0	1	0	0	0
L3	0	0	0	0	0	1	0
L4	0	0	0	0	1	0	0

When are Bitmap Indices useful?

- Say we have the following query:

  select * **from** cs_employees
  where grade = 'P';

- Is Bitmap index useful for this query?

- Not much,
  - You will scan the bitmap index.
  - For every record where grade is equal to P, you will fetch it from the disk.
  - So, you did not have to fetch every record.
  - However, records are stored sequentially in blocks on the disk, so you may end up fetching a lot of blocks with not required blocks!

When are Bitmap Indices useful?

- Say we have another query:

  select * **from** cs_employees
  where grade = ‘P’ and payscale = ‘L1’;

- Is Bitmap index useful for this query?

- Significantly more,
  - We have bitmap indices on both grade and pay attributes.
  - So first, we will take an intersection of these bitmaps and then fetch!

Bitmap for Grade

P	1	1	0	0	1	0	1

Bitmap for Payscale

L1	1	0	1	0	0	0	1

Intersection Bitmap

1	0	0	0	0	0	1

So only two records are fetched!