CS186 Discussion #5

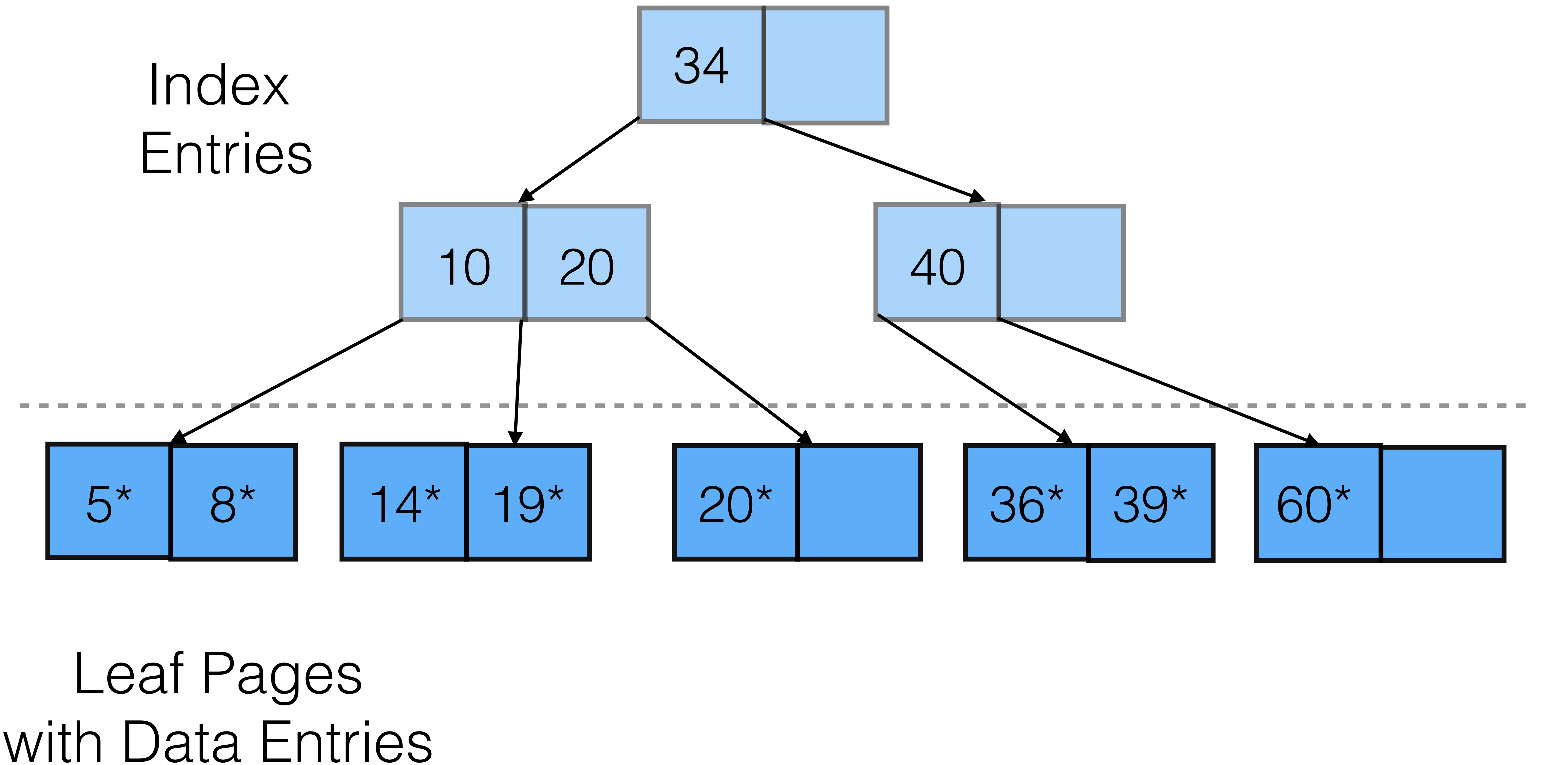
(Tree-Structured Indexes, Relational Algebra)

Tree-Structured Indexes

ISAM

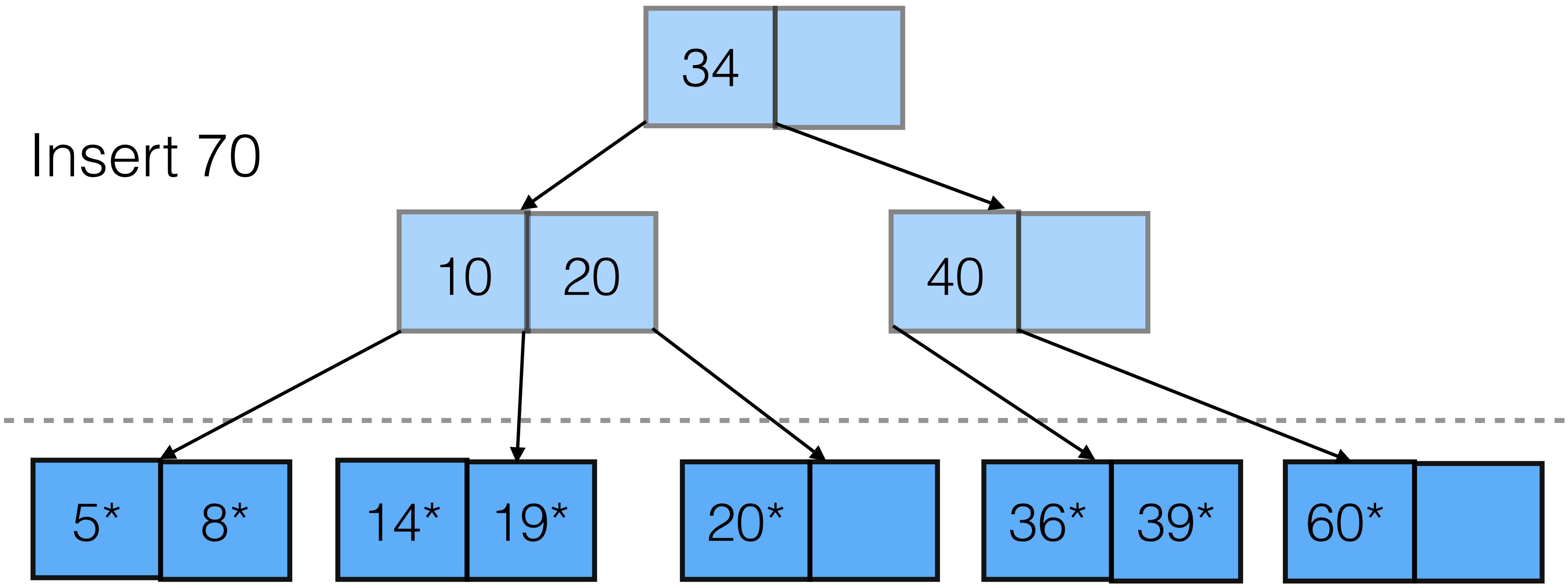
- Simple, static structure
- Created by:
  - Sorting records by index search key (e.g. “gpa”)
  - Building a tree on top of those records

ISAM


Index
Entries
34
10
20
40
5*
8*
14*
19*
20*
36*
39*
60*
Leaf Pages
with Data Entries

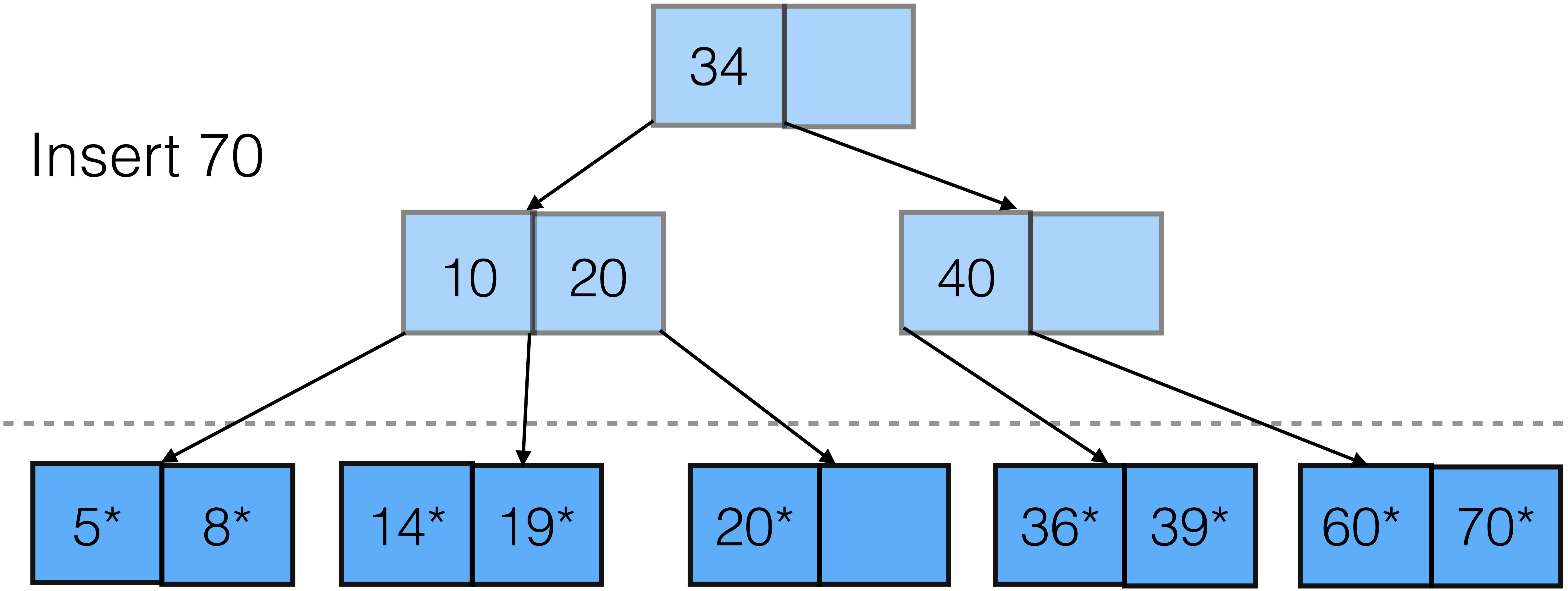
ISAM

Insert 70



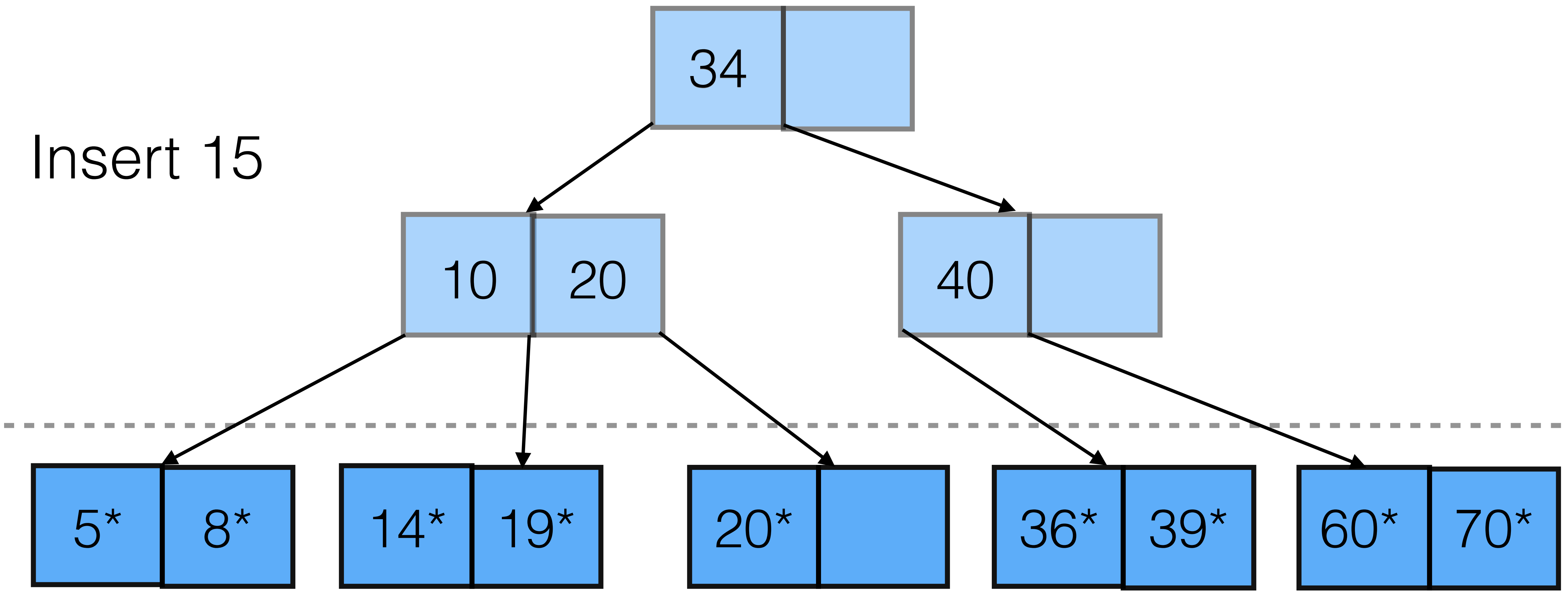
ISAM

Insert 70



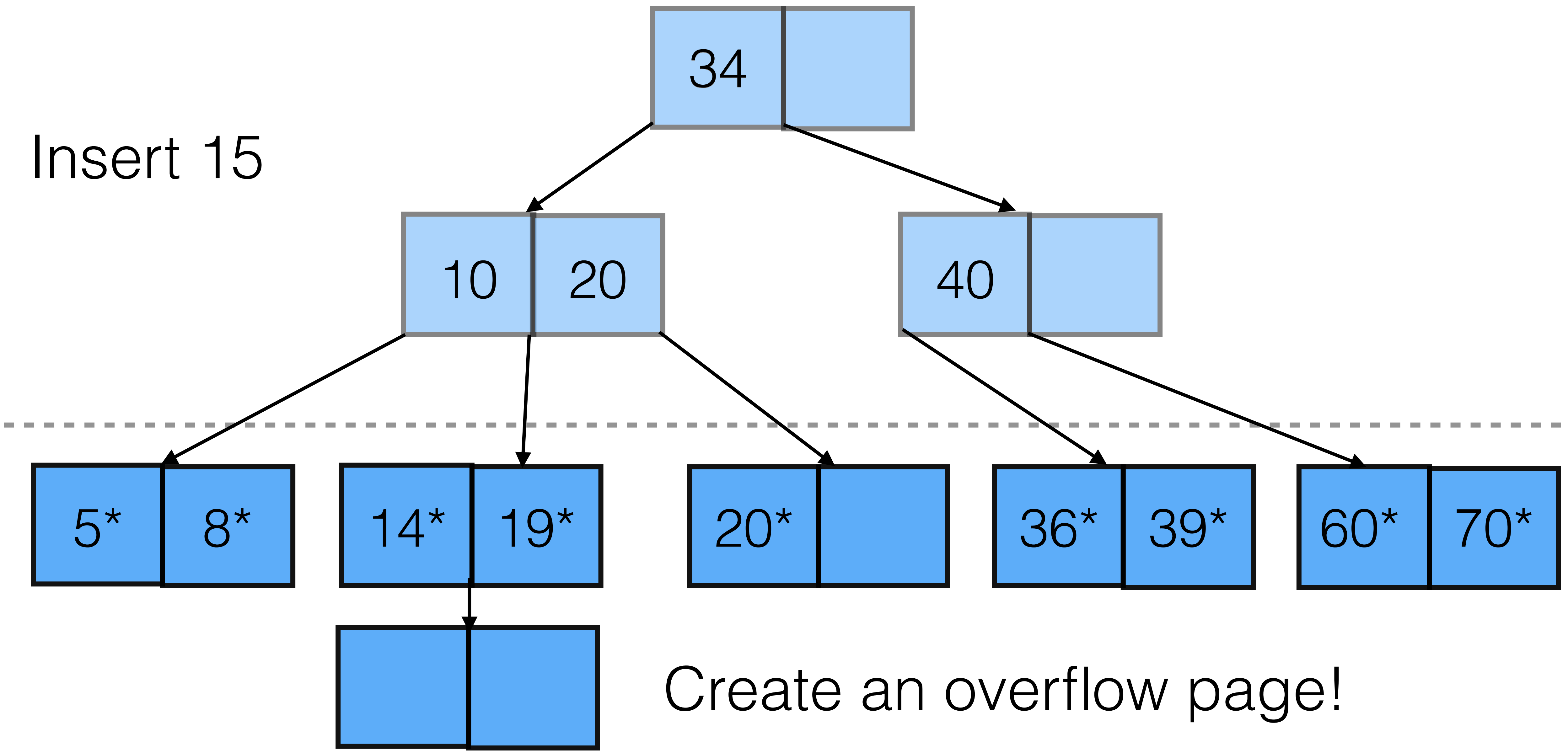
ISAM

Insert 15



ISAM

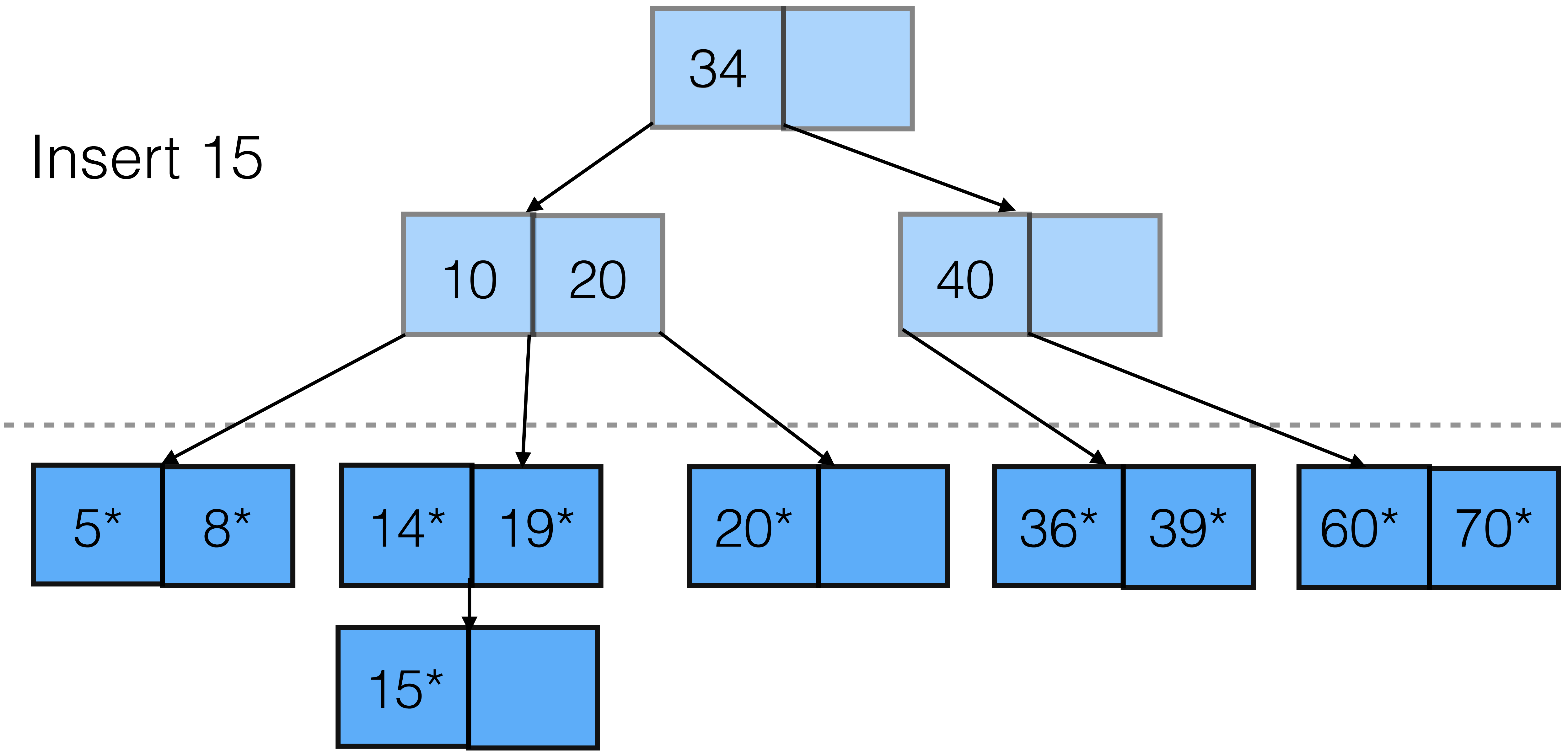
Insert 15



Create an overflow page!

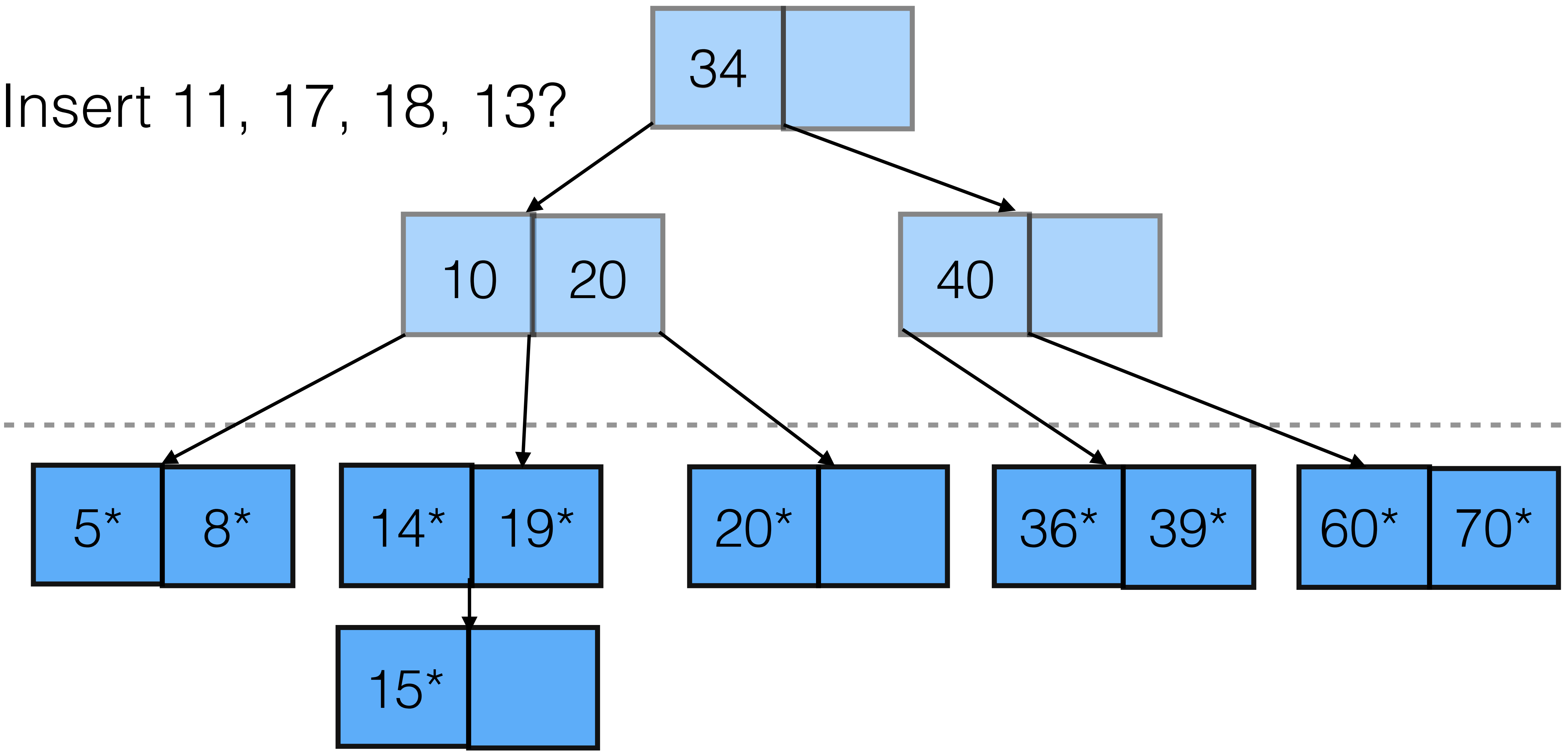
ISAM

Insert 15



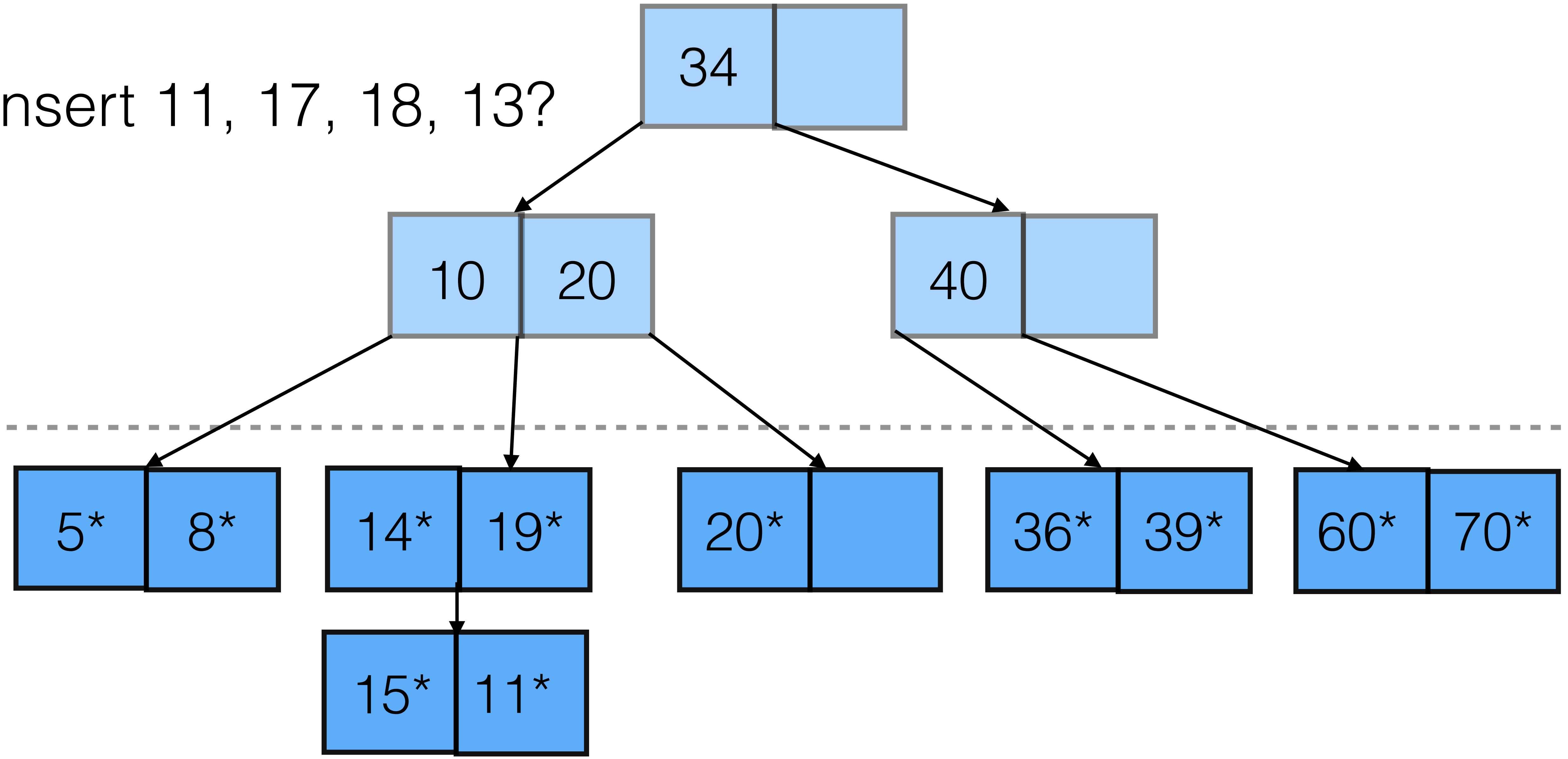
ISAM

Insert 11, 17, 18, 13?



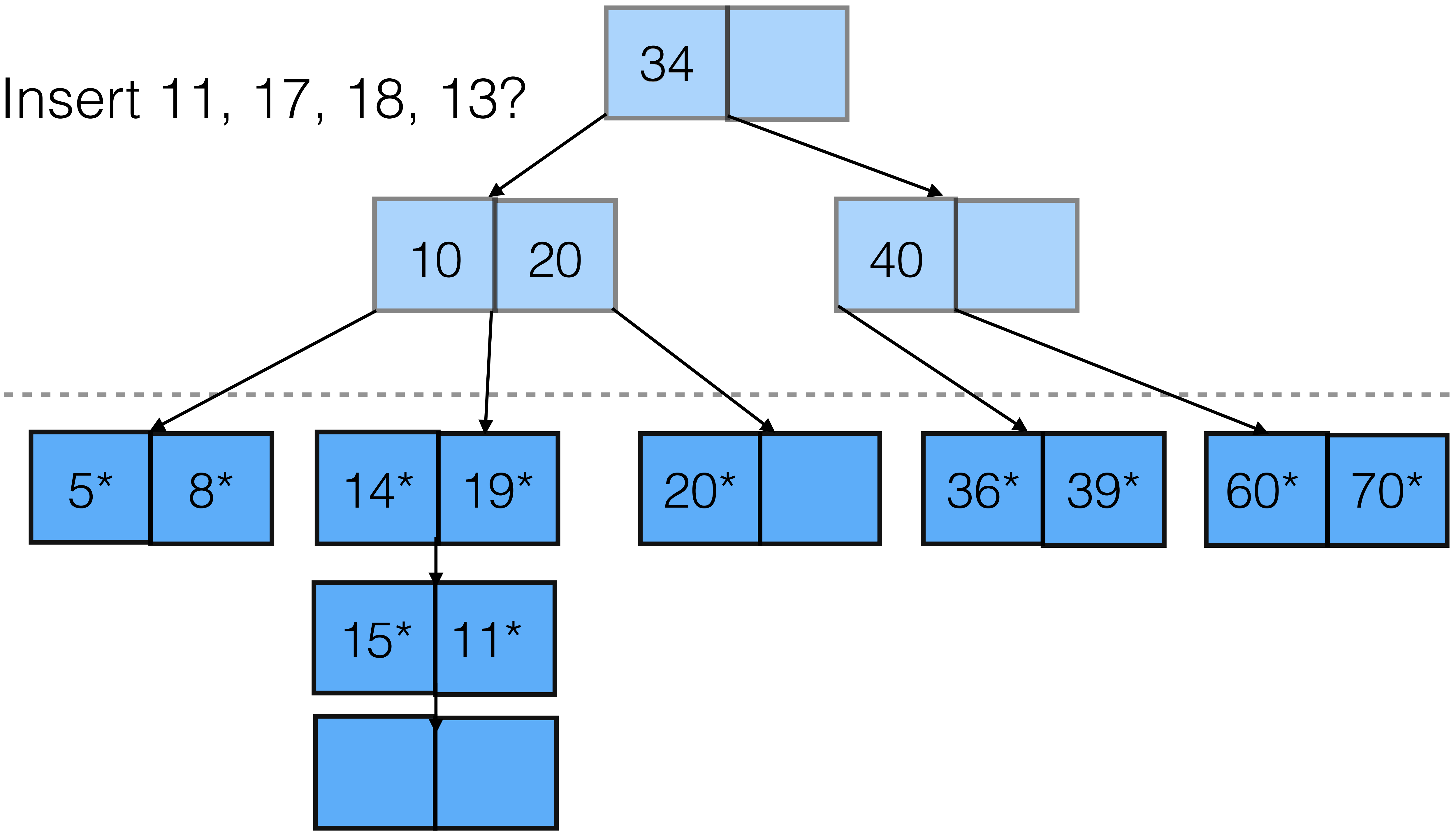
ISAM

Insert 11, 17, 18, 13?



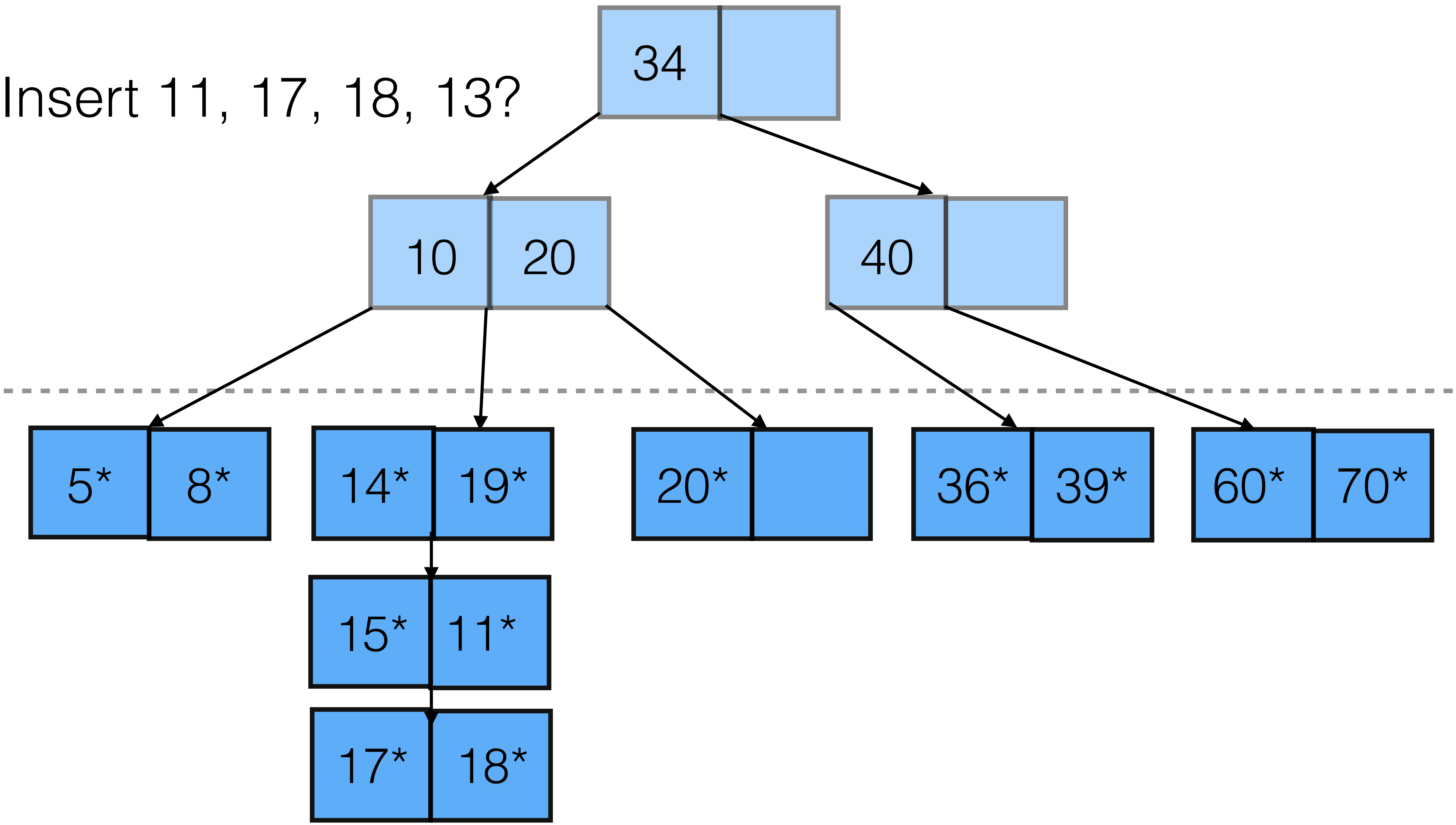
ISAM

Insert 11, 17, 18, 13?



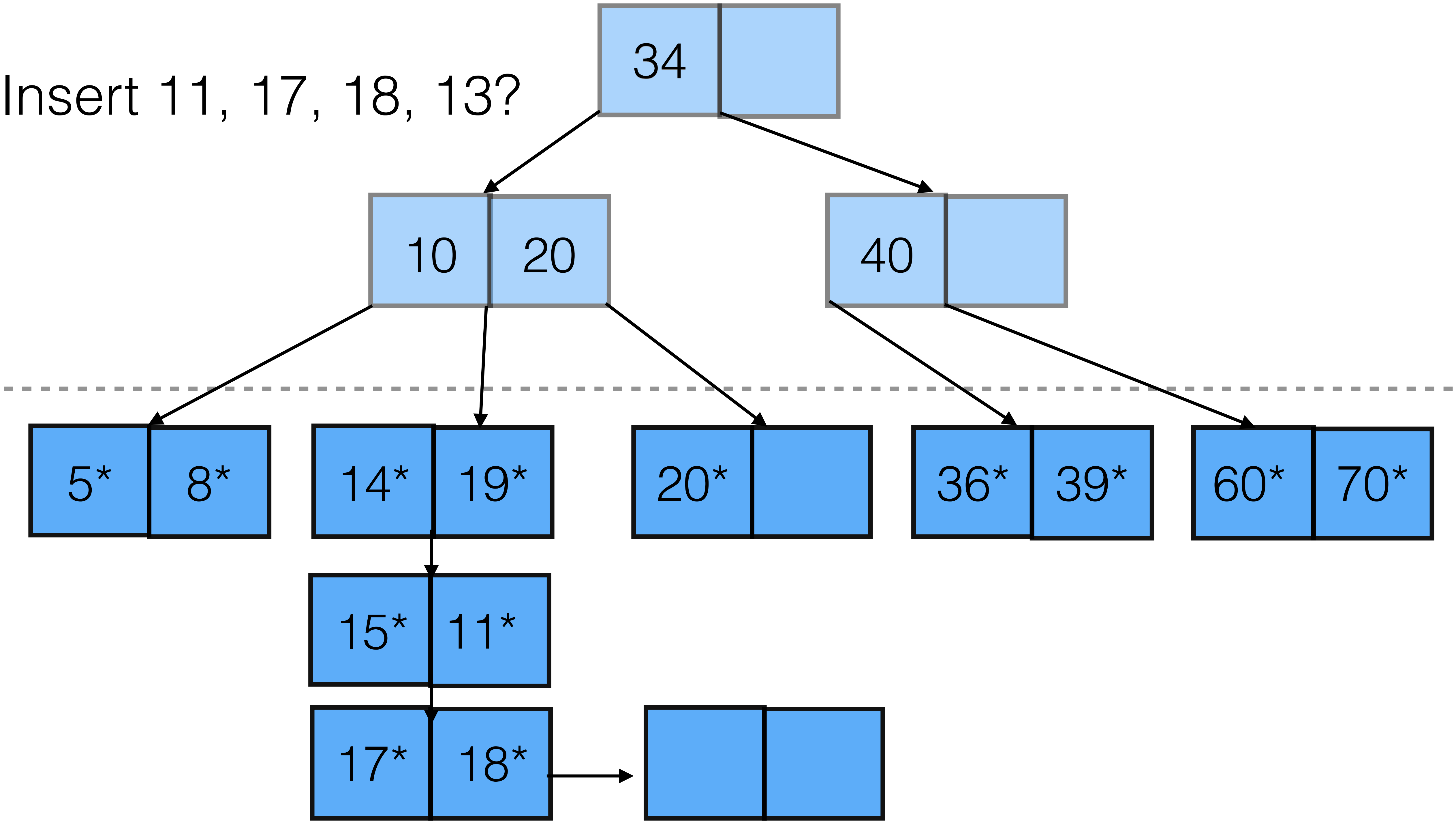
ISAM

Insert 11, 17, 18, 13?



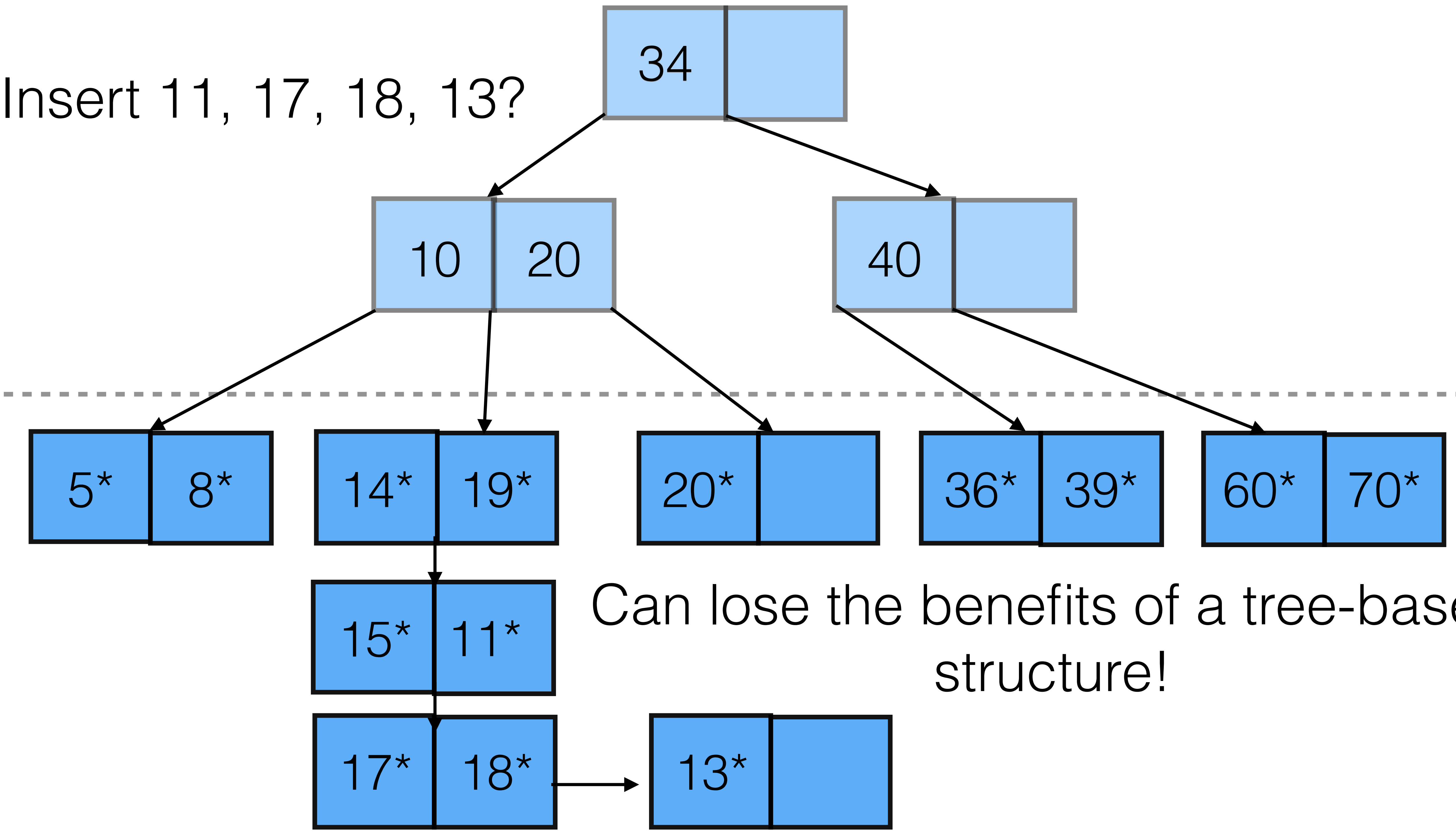
ISAM

Insert 11, 17, 18, 13?



ISAM

Insert 11, 17, 18, 13?



Can lose the benefits of a tree-based structure!

ISAM - Insert X

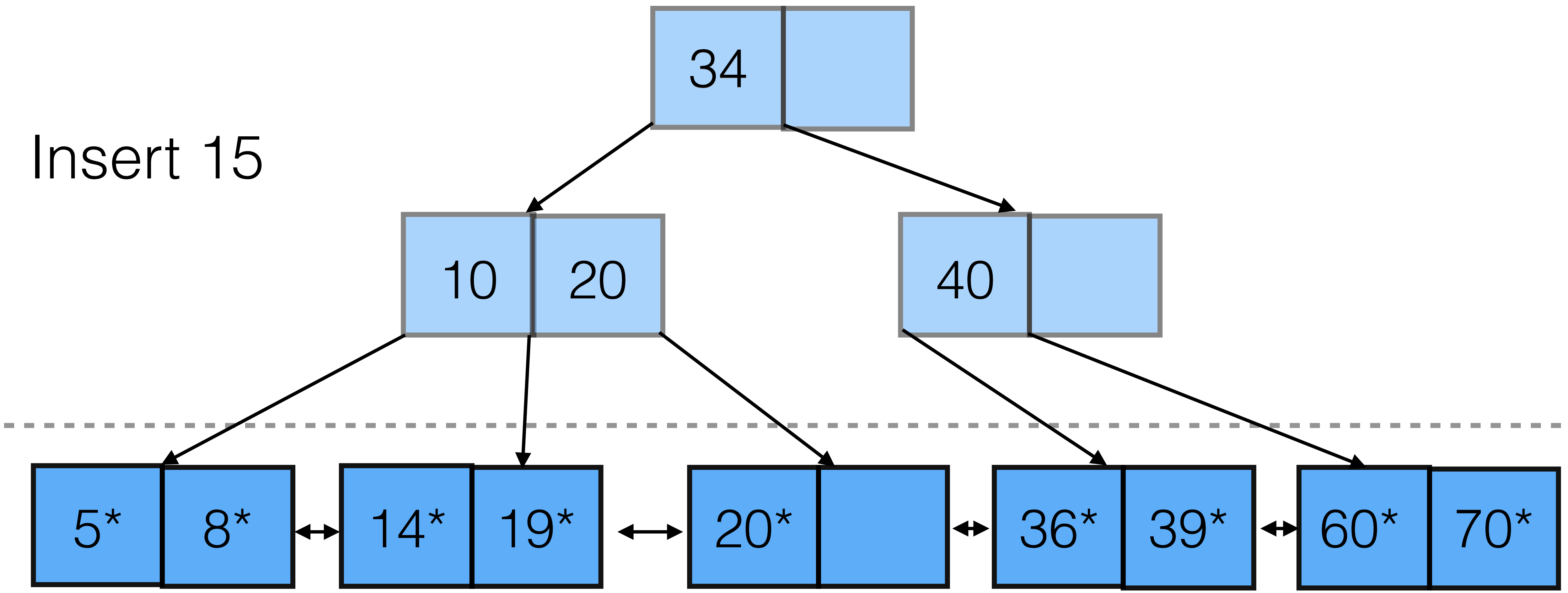
- Traverse index pages to find correct leaf L
- If L has space:
  - Insert X in that page
- Else:
  - If an overflow page has space, insert X in that page
  - Else, create a new overflow page and insert X

B+ Trees

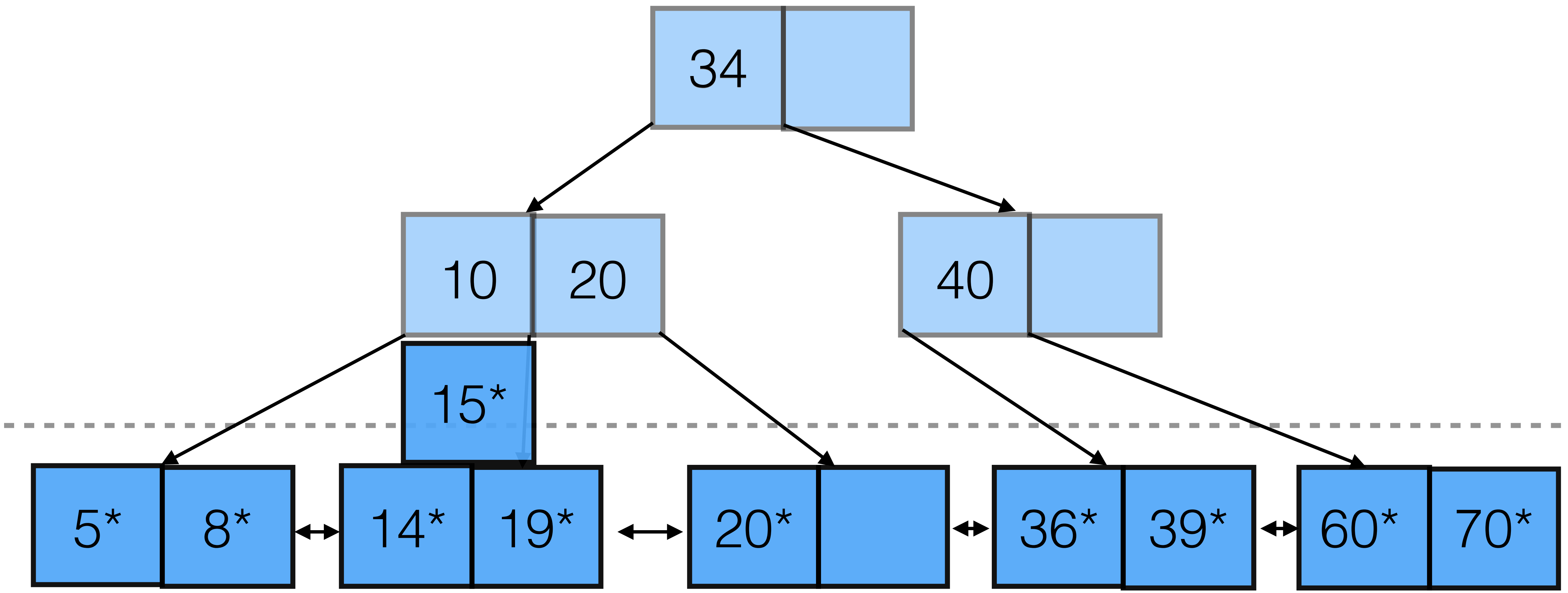
- Dynamic structure to keep tree height-balanced
- Adjusts under inserts and deletes
- Maintain minimum 50% occupancy for each page (except root)

B+ Trees

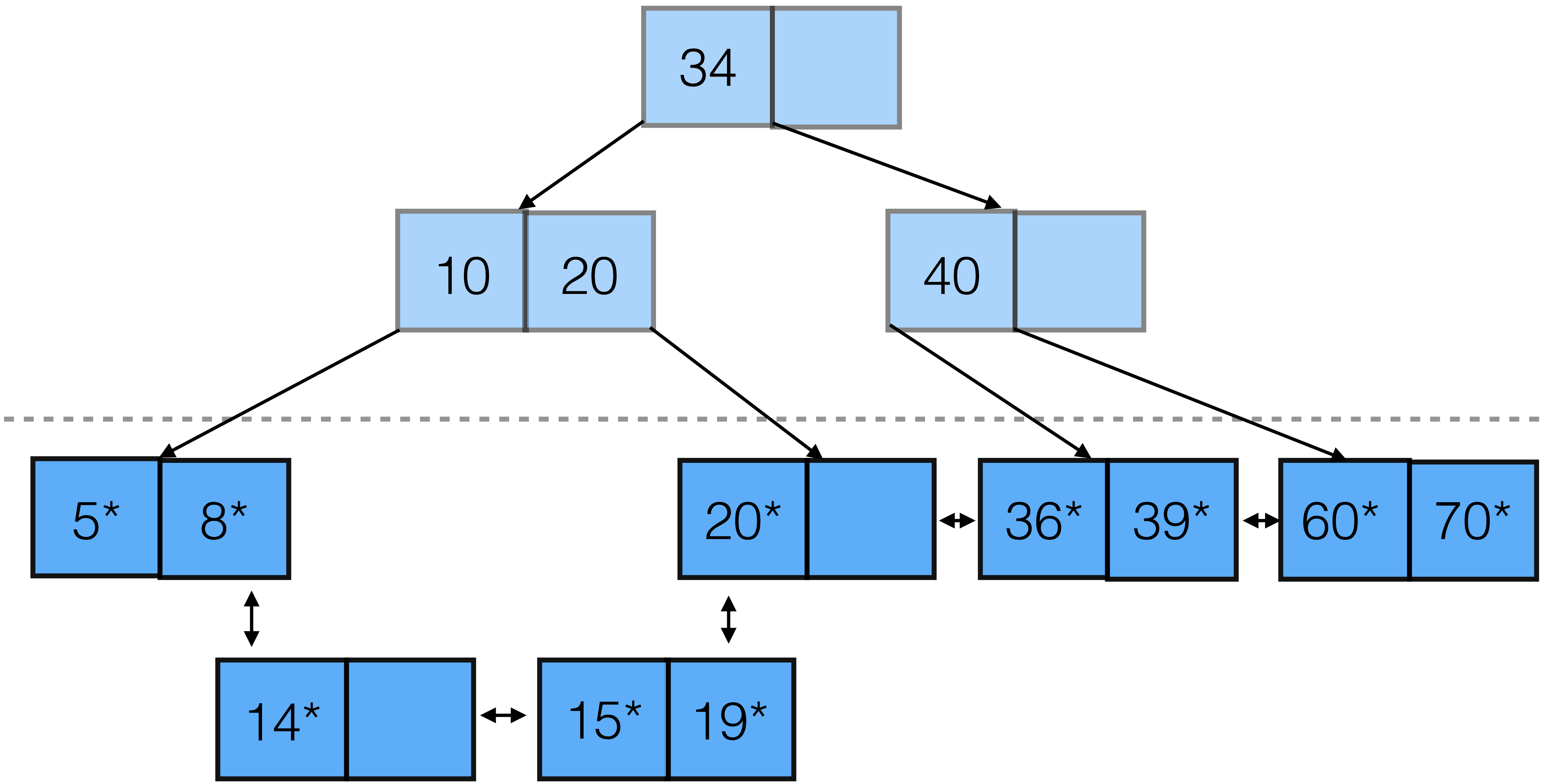
Insert 15



B+ Trees

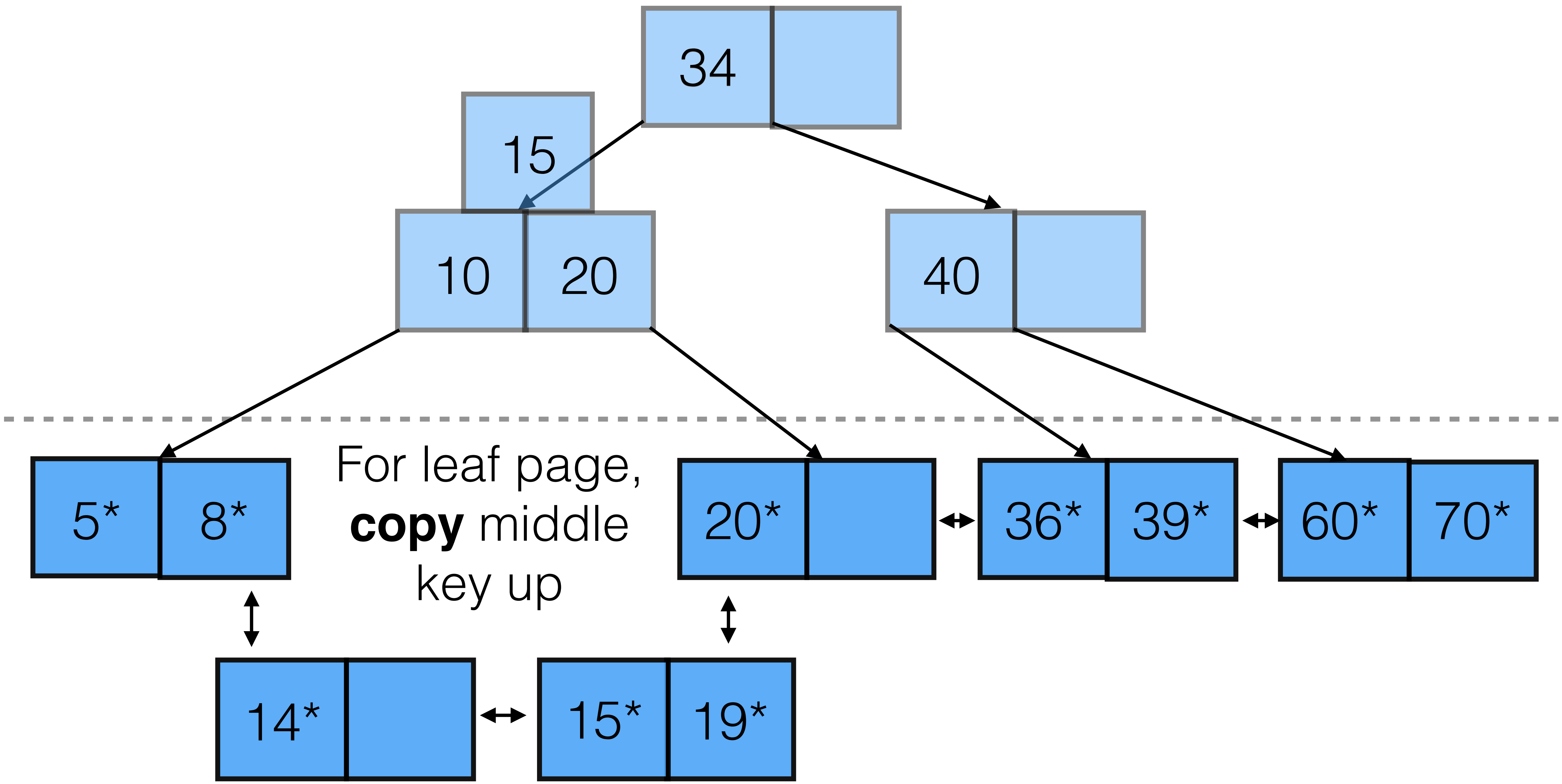


B+ Trees



Split the leaf node

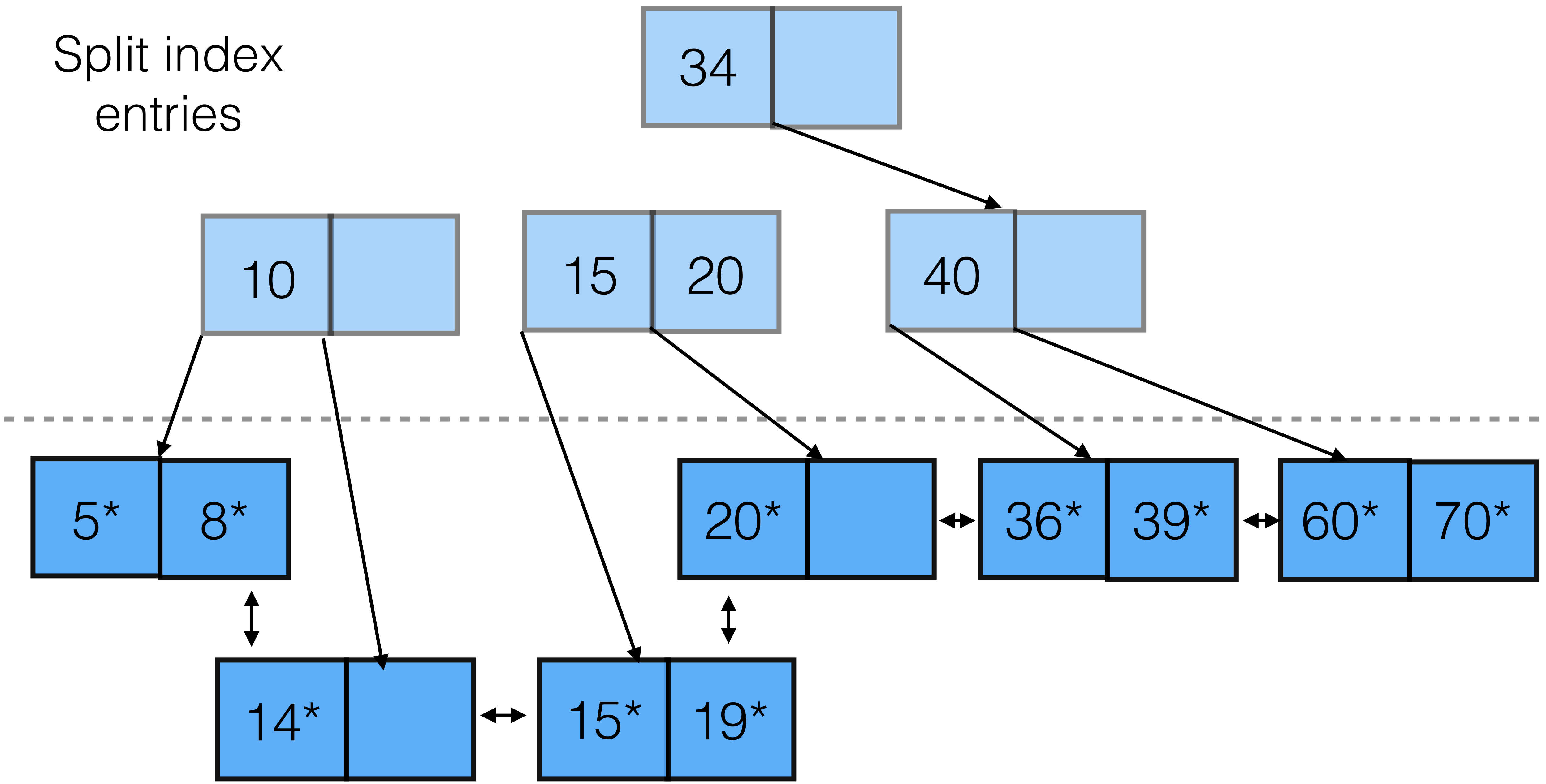
B+ Trees



34
15
10
20
40
5*
8*
For leaf page,
copy middle
key up
20*
36*
39*
60*
70*
14*
15*
19*

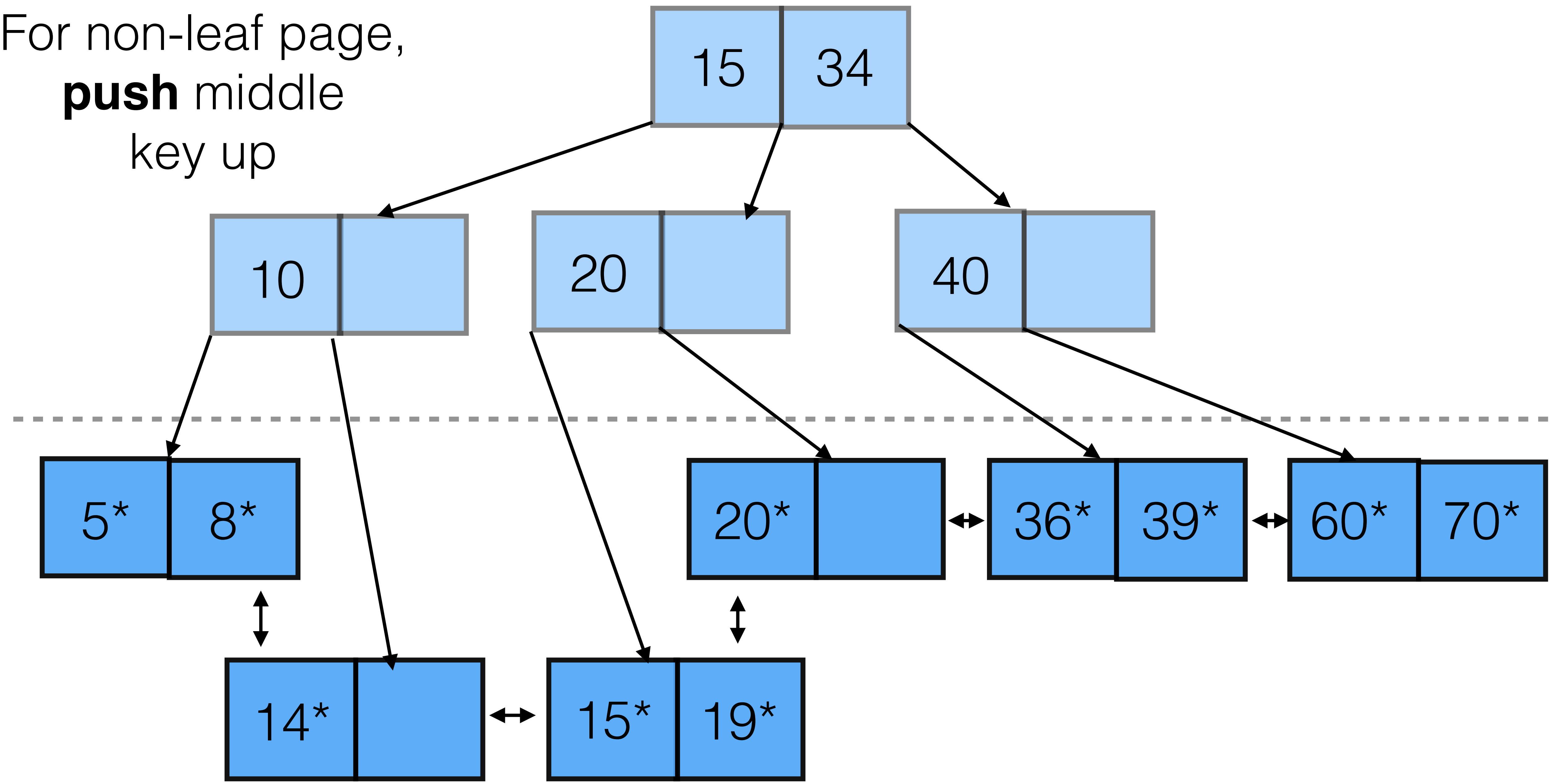
B+ Trees

Split index entries



B+ Trees

For non-leaf page, **push** middle key up



B+ Trees - Insert X

- Find correct leaf L
- Put X in L
  - If not enough space in L:
    - Split L into L and L2
    - **Copy** up middle key to parent
    - If not enough space in parent:
      - Apply algorithm recursively, except **push** up middle key

Worksheet 1, 2, 3, 4

Why do we use tree-structured indexes?

Why do we use tree-structured indexes?

- To speed up selection (lookups, and especially range) on search key fields.

What is the difference between an ISAM and B+ Tree Index?

What is the difference between an ISAM and B+ Tree Index?

- ISAM: Static structure. Consists of root, primary leaf pages and overflow pages. Long overflow chains can develop.
- B+ Tree: Dynamic structure. Height balanced. Usually preferable to ISAM.

Assume that there are 2 million users in your database, that each user entry is 2kB in size, and that you are mainly performing range queries based on a user's age. Assume the page size is 16kB.

- You have decided to create a clustered B+-Tree on the age field. The tree has a fanout of 200 and a height of 3. Assume that you are on average returning 50,000 users per query. On average, how many I/O's are performed by such a query?

Assume that there are 2 million users in your database, that each user entry is 2kB in size, and that you are mainly performing range queries based on a user's age. Assume the page size is 16kB.

- You have decided to create a clustered B+-Tree on the age field. The tree has a fanout of 200 and a height of 3. Assume that you are on average returning 50,000 users per query. On average, how many I/O's are performed by such a query?

- **3** I/O's to traverse index entries
- Number of leaf pages read: (50,000 * 2)/16 = **6250**
- 3 + 6250 = **6253 I/O's**

Assume that there are 2 million users in your database, that each user entry is 2kB in size, and that you are mainly performing range queries based on a user's age. Assume the page size is 16kB.

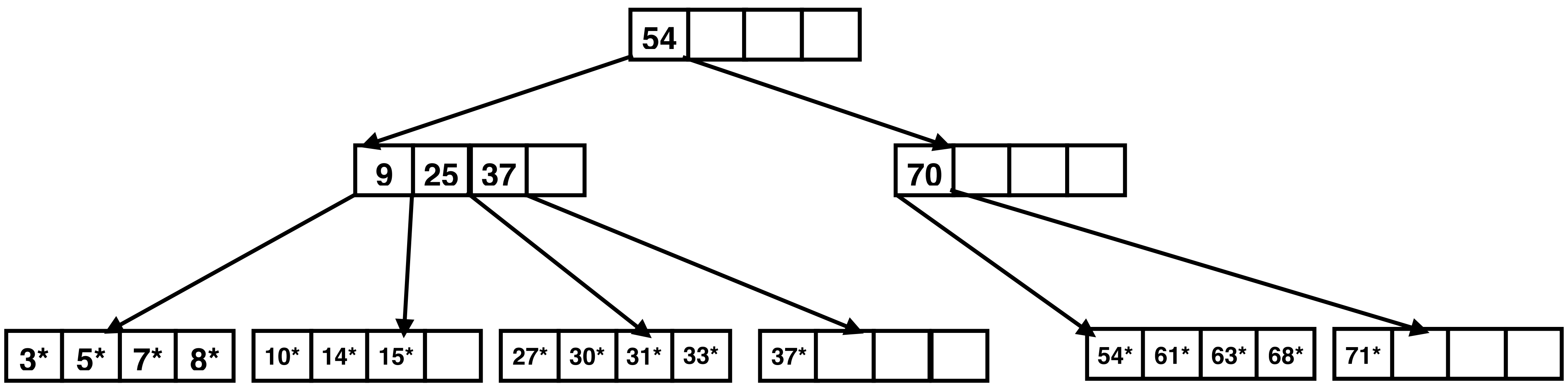
- Assume your B+ tree is unclustered. In the worst case, how many I/O's do you need now? Assume that you are still returning 50,000 users per query on average, and that an index entry is 3 times smaller than a user entry.

Assume that there are 2 million users in your database, that each user entry is 2kB in size, and that you are mainly performing range queries based on a user's age. Assume the page size is 16kB.

- Assume your B+ tree is unclustered. In the worst case, how many I/O's do you need now? Assume that you are still returning 50,000 users per query on average, and that an index entry is 3 times smaller than a user entry.

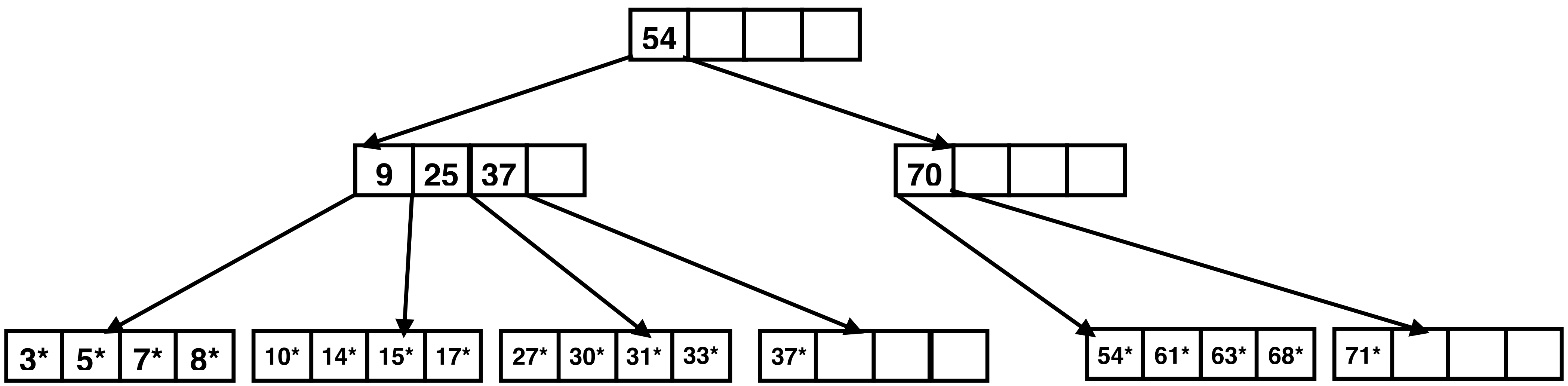
- **3** I/O’s to traverse index entries
- Number of leaf pages read: ceil(50,000 * 2/3 / 16) = **2084** I/Os
- Number of unordered data pages read: **50000**
- 3 + 2084 + 50000 = **52087 I/O’s**

Consider the B+ Tree below and insert the following in order: 17, 18, 29.



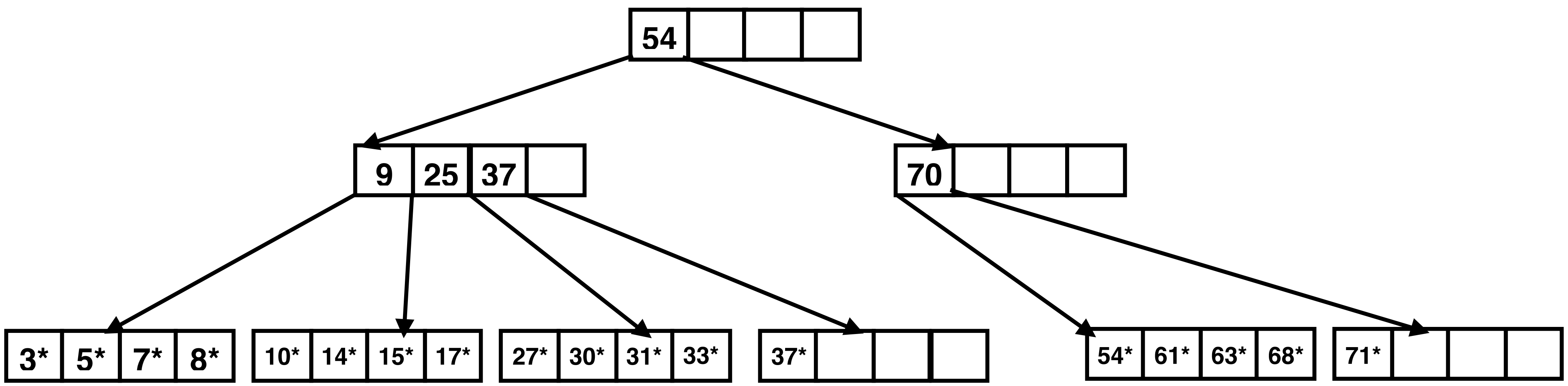
Insert 17

Consider the B+ Tree below and insert the following in order: 17, 18, 29.



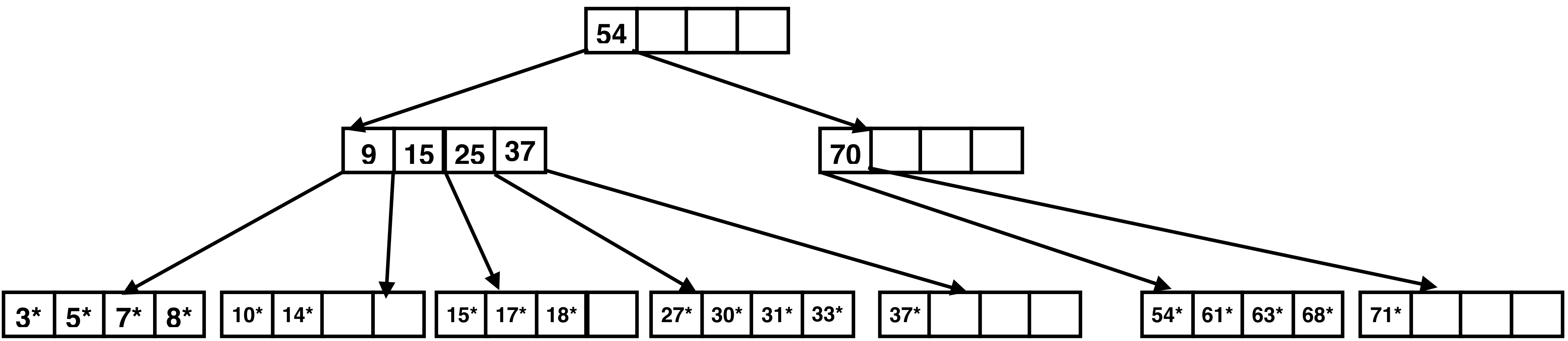
Insert 17

Consider the B+ Tree below and insert the following in order: 17, 18, 29.



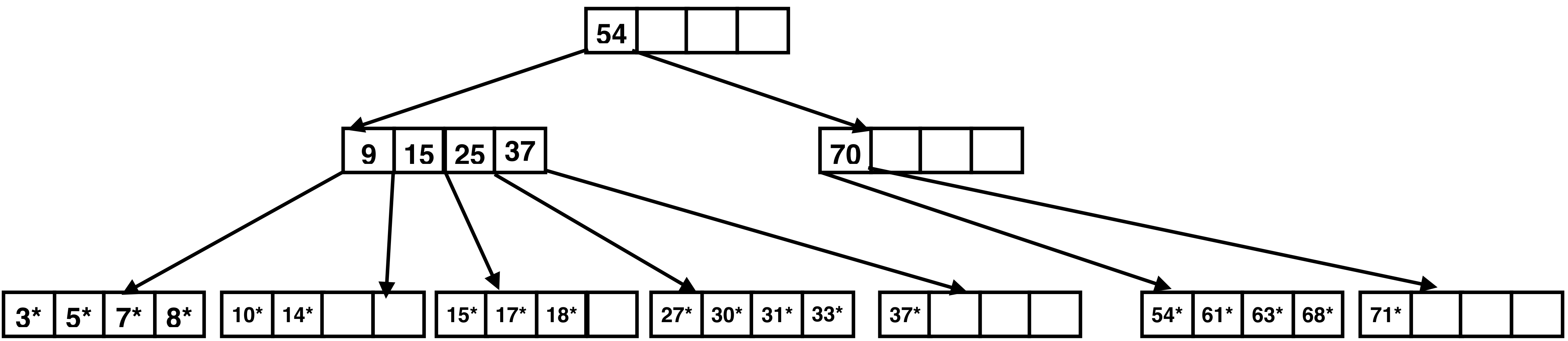
Insert 18

Consider the B+ Tree below and insert the following in order: 17, 18, 29.



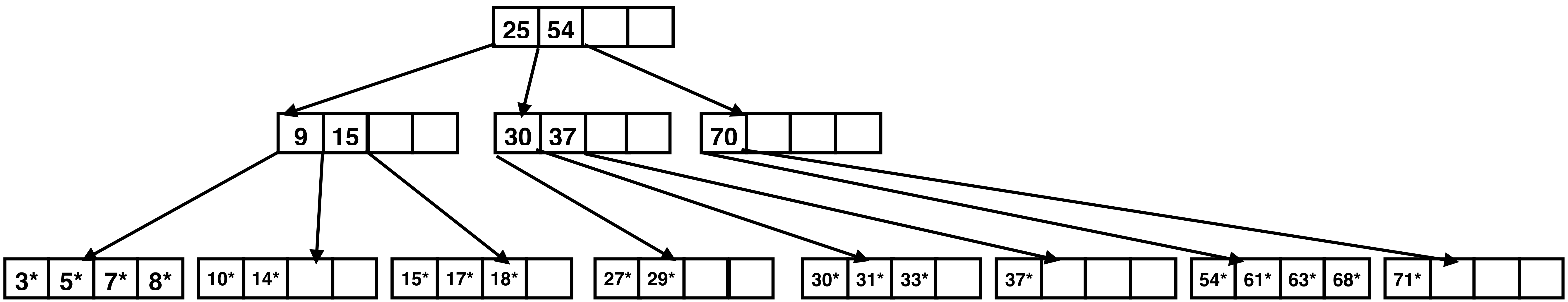
Insert 18

Consider the B+ Tree below and insert the following in order: 17, 18, 29.



Insert 29

Consider the B+ Tree below and insert the following in order: 17, 18, 29.



Insert 29

Relational Algebra

Relational Algebra

- Input and output: Relation instances (tables)
- Has set semantics
  - **No** duplicate tuples in a relation
- Useful for representing semantics of execution plans in a DBMS (more later!)

Relational Algebra

Operation	Symbol	Explanation
Selection	σ	Selects rows
Projection	π	Selects columns
Union	$\cup$	Tuples in r1 or r2
Intersection	$\cap$	Tuples in r1 and r2
Cross-product	$\times$	Combines two relations
Join	$\bowtie$	Conditional cross-product
Difference	$-$	Tuples in r2 not in r1

Selection

- Select rows
- Example: $\sigma_{gpa > 3.5}(R)$

name	sid	gpa
Bob	1	3.7
Sue	3	2.9
Ron	2	1.2
Al	4	4.0
Sally	5	3.6

Selection

- Select rows
- Example: $\sigma_{gpa > 3.5}(R)$

name	sid	gpa
Bob	1	3.7
Sue	3	2.9
Ron	2	1.2
Al	4	4.0
Sally	5	3.6

Projection

- Select columns
- Example: $\pi_{\text{name, sid}}(R)$

name	sid	gpa
Bob	1	3.7
Sue	3	2.9
Ron	2	1.2
Al	4	4.0
Sally	5	3.6

Projection

- Select columns
- Example: $\pi_{\text{name, sid}}(R)$

name	sid	gpa
Bob	1	3.7
Sue	3	2.9
Ron	2	1.2
Al	4	4.0
Sally	5	3.6

Selection & Projection

- Example: $\pi_{\text{name, sid}}(\sigma_{\text{gpa} > 3.5}(R))$

name	sid	gpa
Bob	1	3.7
Sue	3	2.9
Ron	2	1.2
Al	4	4.0
Sally	5	3.6

Selection & Projection

- Example: $\pi_{name,\ sid}(\sigma_{gpa > 3.5}(R))$

name	sid	gpa
Bob	1	3.7
Sue	3	2.9
Ron	2	1.2
Al	4	4.0
Sally	5	3.6

Union

- Set union between two relations
- Example: $\sigma_{sid < 3}(R) \cup \sigma_{sid\%2 == 0}(R)$

name	sid	gpa
Bob	1	3.7
Sue	3	2.9
Ron	2	1.2
Al	4	4.0
Sally	5	3.6

Union

- Set union between two relations
- Example: $\sigma_{sid < 3}(R) \cup \sigma_{sid\%2 == 0}(R)$

name	sid	gpa
Bob	1	3.7
Ron	2	1.2

$\cup$

name	sid	gpa
Ron	2	1.2
Al	4	4.0

Union

- Set union between two relations

- Example: $\sigma_{sid < 3}(R) \cup \sigma_{sid\%2 == 0}(R)$

name	sid	gpa
Bob	1	3.7
Ron	2	1.2
Al	4	4.0

Intersection

- Set intersection between two relations
- Example: $\sigma_{sid < 3}(R) \cap \sigma_{sid\%2 == 0}(R)$

name	sid	gpa
Bob	1	3.7
Sue	3	2.9
Ron	2	1.2
Al	4	4.0
Sally	5	3.6

Intersection

- Set intersection between two relations
- Example: $\sigma_{sid < 3}(R) \cap \sigma_{sid\%2 == 0}(R)$

name	sid	gpa
Bob	1	3.7
Ron	2	1.2

$\cap$

name	sid	gpa
Ron	2	1.2
Al	4	4.0

Intersection

- Set intersection between two relations
- Example: $\sigma_{sid < 3}(R) \cap \sigma_{sid\%2 == 0}(R)$

name	sid	gpa
Ron	2	1.2

Cross Product

- Takes all rows from A and combines with all rows in B
- Example: $\pi_{name}(R) \times \pi_{gpa}(R)$

name	sid	gpa
Bob	1	3.7
Sue	3	2.9
Ron	2	1.2

Cross Product

- Takes all rows from A and combines with all rows in B

- Example: $\pi_{name}(R) \times \pi_{gpa}(R)$

name
Bob
Sue
Ron

$\times$

gpa
3.7
2.9
1.2

Cross Product

- Takes all rows from A and combines with all rows in B
- Example: $\pi_{name}(R) \times \pi_{gpa}(R)$

name
Bob
Sue
Ron

×

gpa
3.7
2.9
1.2

=

name	gpa
Bob	3.7
Bob	2.9
Bob	1.2
Sue	3.7
Sue	2.9
Sue	1.2
Ron	3.7
Ron	2.9
Ron	1.2

Join

- Joins A and B based on some column
- Example: $\pi_{name,sid}(R) \bowtie \pi_{name,gpa}(R)$

name	sid
Bob	1
Sue	3
Ron	2

$\bowtie$

name	gpa
Bob	3.7
Sue	2.9
Ron	1.2

Join

- Joins A and B based on some column
- Example: $\pi_{name,sid}(R) \bowtie \pi_{name,gpa}(R)$

name	sid	gpa
Bob	1	3.7
Sue	3	2.9
Ron	2	1.2

Difference

- Takes rows in A that are not in B
- Example: $\sigma_{gpa > 3.5}(R) - \sigma_{sid\%2==0}(R)$

name	sid	gpa
Bob	1	3.7
Sue	3	2.9
Ron	2	1.2
Al	4	4.0
Sally	5	3.6

Difference

- Takes rows in A that are not in B
- Example: $\sigma_{gpa > 3.5}(R) - \sigma_{sid\%2==0}(R)$

name	sid	gpa
Bob	1	3.7
Al	4	4.0
Sally	5	3.6

\-

name	sid	gpa
Ron	2	1.2
Al	4	4.0

Difference

- Takes rows in A that are not in B
- Example: $\sigma_{gpa > 3.5}(R) - \sigma_{sid\%2==0}(R)$

name	sid	gpa
Bob	1	3.7
Sally	5	3.6

Worksheet 5

Consider the schema:

```
Songs  (song_id,   song_name,   album_id,  weeks_in_top_40)
Artists(artist_id, artist_name, first_year_active)
Albums  (album_id,  album_name,  artist_id, year_released, genre)
```

Write relational algebra expressions for the following query:

- Find the name of the artists who have albums with a genre of either ‘pop’ or ‘rock’.

Consider the schema:

```
Songs  (song_id,   song_name,   album_id,  weeks_in_top_40)
Artists(artist_id, artist_name, first_year_active)
Albums  (album_id,  album_name,  artist_id, year_released, genre)
```

Write relational algebra expressions for the following query:

- Find the name of the artists who have albums with a genre of either ‘pop’ or ‘rock’.

$$\pi_{\text{Artists.artist\_name}} (\text{Artists} \bowtie (\sigma_{\text{Albums.genre = ‘pop’} \lor \text{Albums.color = ‘rock’}} \text{Albums}))$$

Consider the schema:

```
Songs  (song_id,   song_name,   album_id,  weeks_in_top_40)
Artists(artist_id, artist_name, first_year_active)
Albums  (album_id,  album_name,  artist_id, year_released, genre)
```

Write relational algebra expressions for the following query:

- Find the name of the artists who have albums of genre ‘pop’ and ‘rock’.

Consider the schema:

`Songs (song_id, song_name, album_id, weeks_in_top_40)`
`Artists(artist_id, artist_name, first_year_active)`
`Albums (album_id, album_name, artist_id, year_released, genre)`

Write relational algebra expressions for the following query:

- Find the name of the artists who have albums of genre ‘pop’ and ‘rock’.

$$\pi_{\text{Artists.artist\_name}} (\text{Artists} \bowtie (\sigma_{\text{Albums.genre} = \text{‘pop’}} \text{Albums}))$$

$$\cap$$

$$\pi_{\text{Artists.artist\_name}} (\text{Artists} \bowtie (\sigma_{\text{Albums.genre} = \text{‘rock’}} \text{Albums}))$$

Consider the schema:

```
Songs  (song_id,   song_name,   album_id,  weeks_in_top_40)
Artists(artist_id, artist_name, first_year_active)
Albums  (album_id,  album_name,  artist_id, year_released, genre)
```

Write relational algebra expressions for the following query:

- Find the id of the artists who have albums of genre ‘pop’ or have spent over 10 weeks in the top 40.

Consider the schema:

```
Songs   (song_id,   song_name,   album_id,  weeks_in_top_40)
Artists(artist_id, artist_name, first_year_active)
Albums  (album_id,  album_name,  artist_id, year_released, genre)
```

Write relational algebra expressions for the following query:

- Find the id of the artists who have albums of genre ‘pop’ or have spent over 10 weeks in the top 40.

$$\pi_{\text{Artists.artist\_id}} (\text{Artists} \bowtie (\sigma_{\text{Albums.genre = ‘pop’}} \text{Albums}))$$

$$\cup$$

$$\pi_{\text{Albums.artist\_id}} (\text{Albums} \bowtie (\sigma_{\text{Songs.weeks\_in\_top\_40} > 10} \text{Songs}))$$

Consider the schema:

```
Songs  (song_id,   song_name,   album_id,  weeks_in_top_40)
Artists(artist_id, artist_name, first_year_active)
Albums  (album_id,  album_name,  artist_id, year_released, genre)
```

Write relational algebra expressions for the following query:

- Find the names of all artists who do not have any albums.

Consider the schema:

`Songs (song_id, song_name, album_id, weeks_in_top_40)`
`Artists(artist_id, artist_name, first_year_active)`
`Albums (album_id, album_name, artist_id, year_released, genre)`

Write relational algebra expressions for the following query:

- Find the names of all artists who do not have any albums.

$$\pi_{\text{Artists.artist\_name}} (\text{Artists} \bowtie ((\pi_{\text{Artists.artist\_id}} \text{Artists}) - (\pi_{\text{Albums.artist\_id}} \text{Albums}))$$