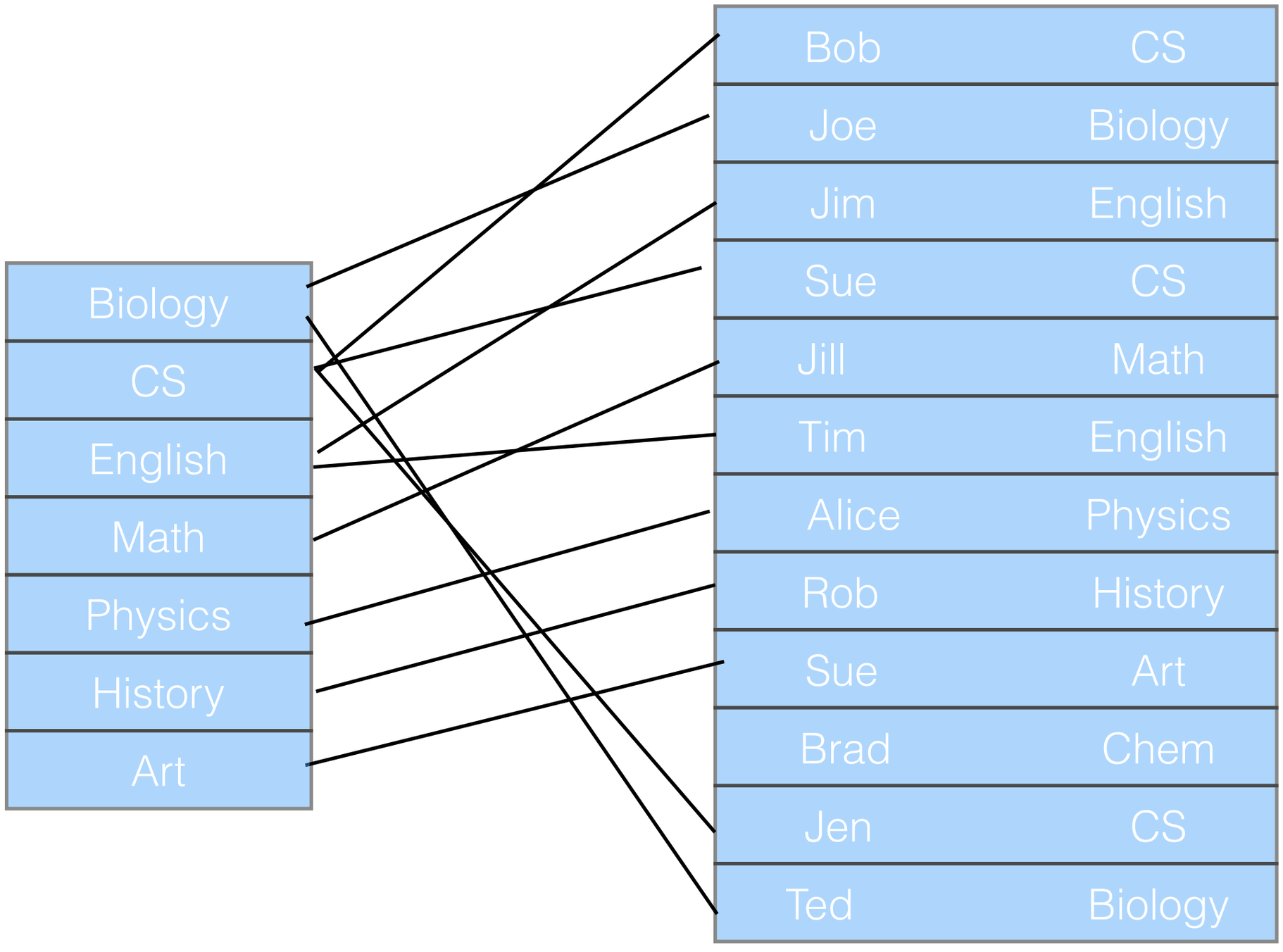
CS186 Discussion #5

(Indexes, B+ Trees)

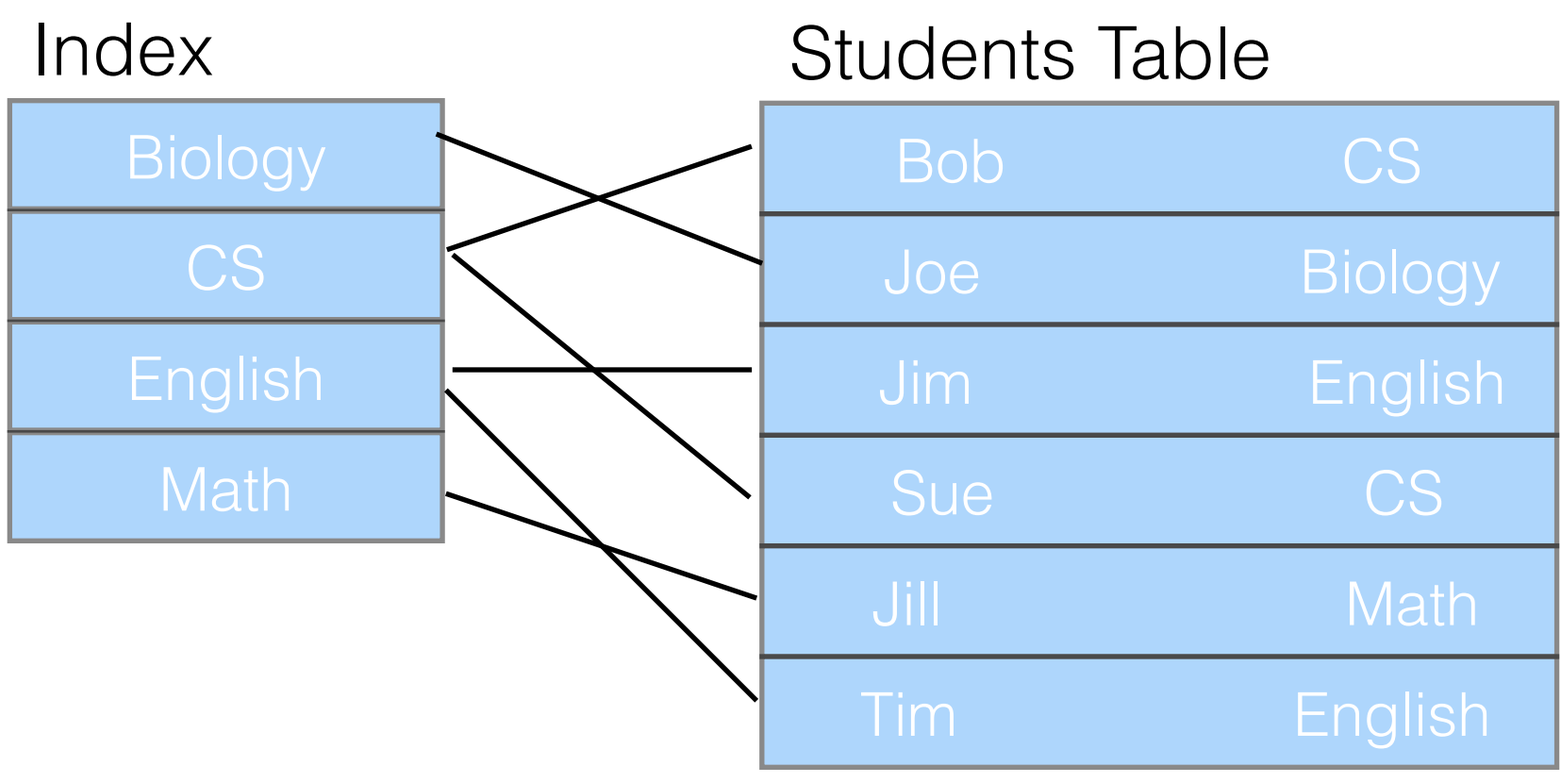
Fill out the Index definitions on your worksheet.

Bob	CS
Joe	Biology
Jim	English
Sue	CS
Jill	Math
Tim	English
Alice	Physics
Rob	History
Sue	Art
Brad	Chem
Jen	CS
Ted	Biology


Biology
CS
English
Math
Physics
History
Art
Bob CS
Joe Biology
Jim English
Sue CS
Jill Math
Tim English
Alice Physics
Rob History
Sue Art
Brad Chem
Jen CS
Ted Biology

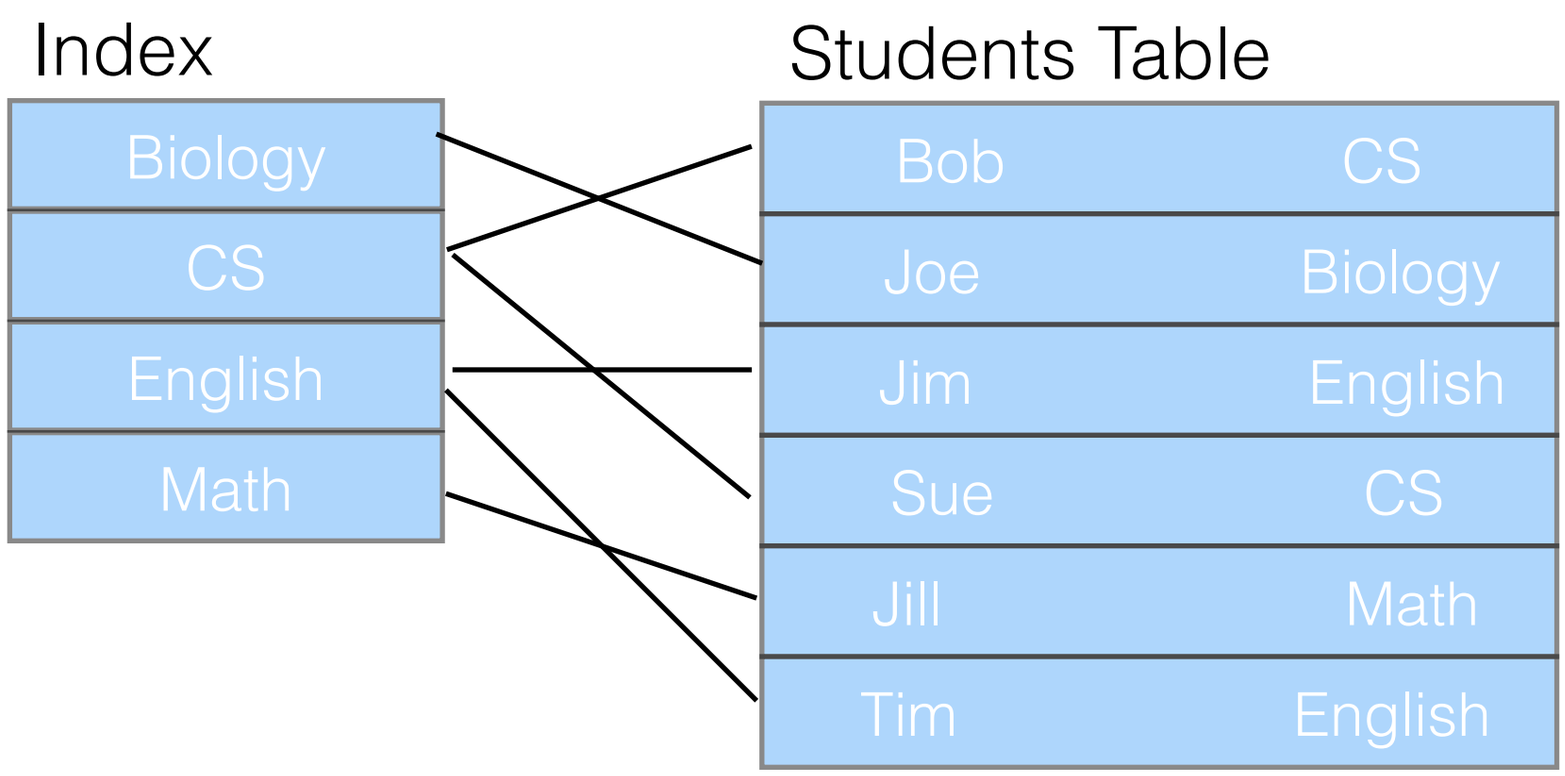
Indexes

- Disk-based data structure for fast lookup by value (search key)
  - Ex: Find students in the CS department



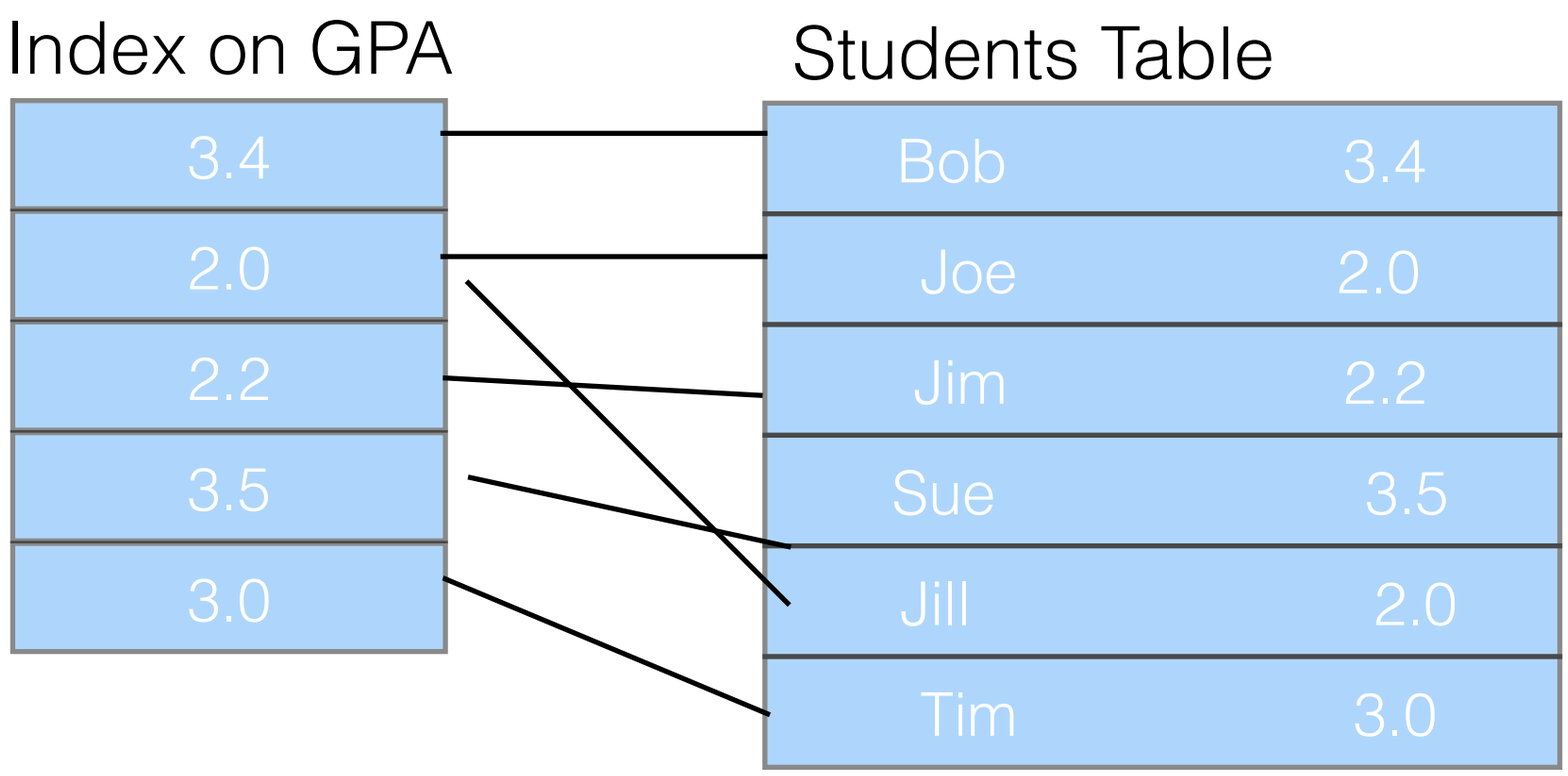
Hash Index

- Good for equality search



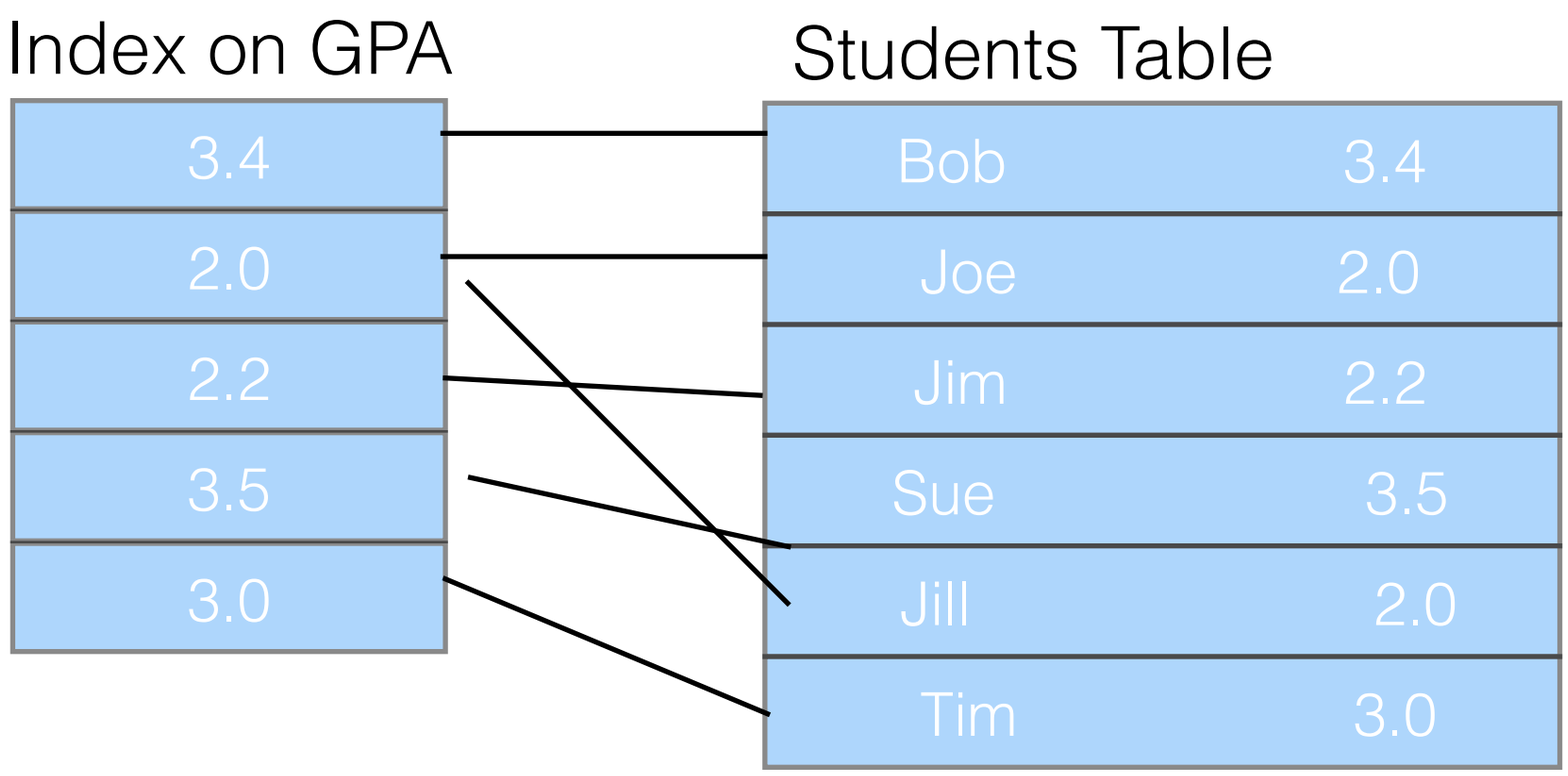
Hash Index

- Good for equality search



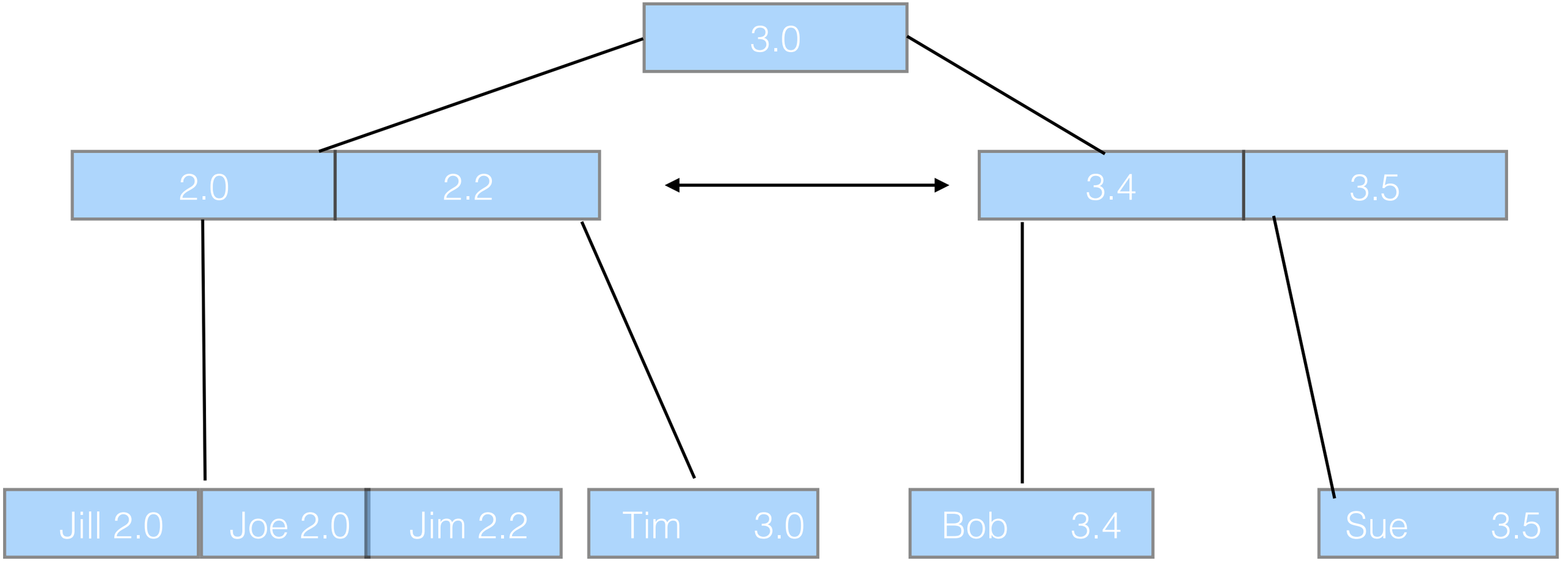
Hash Index

- Good for equality search
- Can’t do range queries



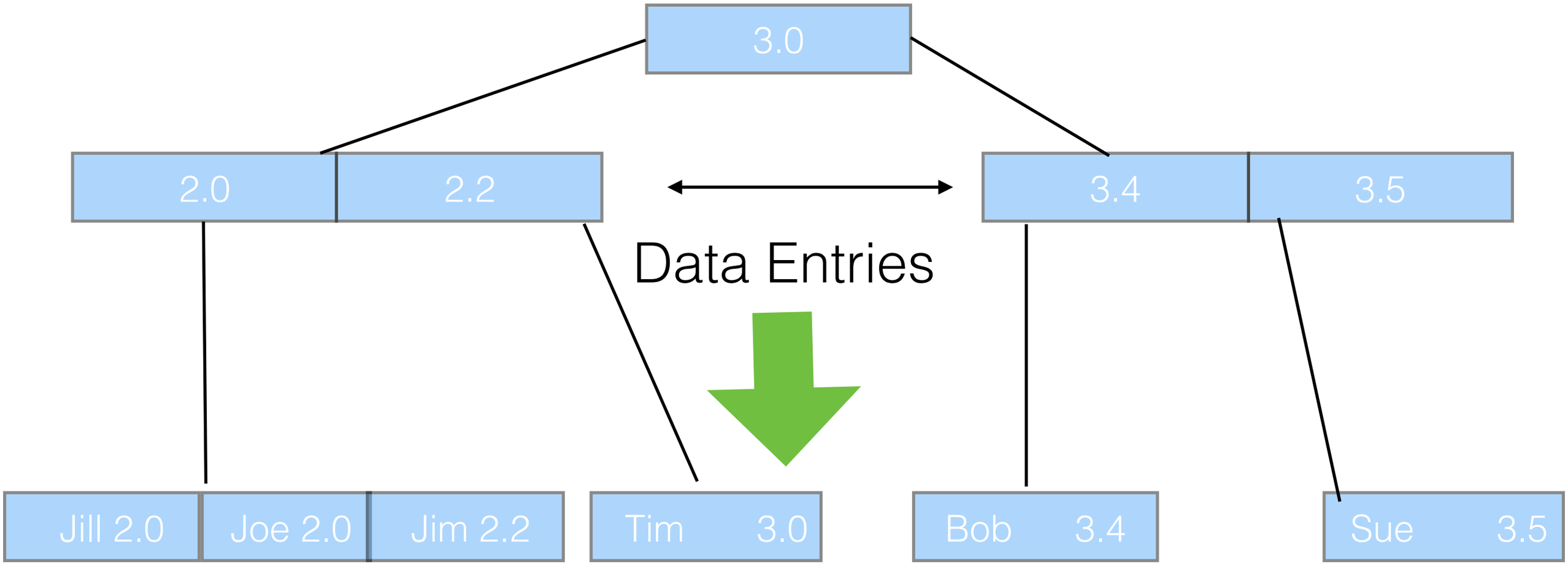
Tree Index

- Fast equality search, insertion, deletion
- Can do range search



Tree Index

- Fast equality search, insertion, deletion
- Can do range search

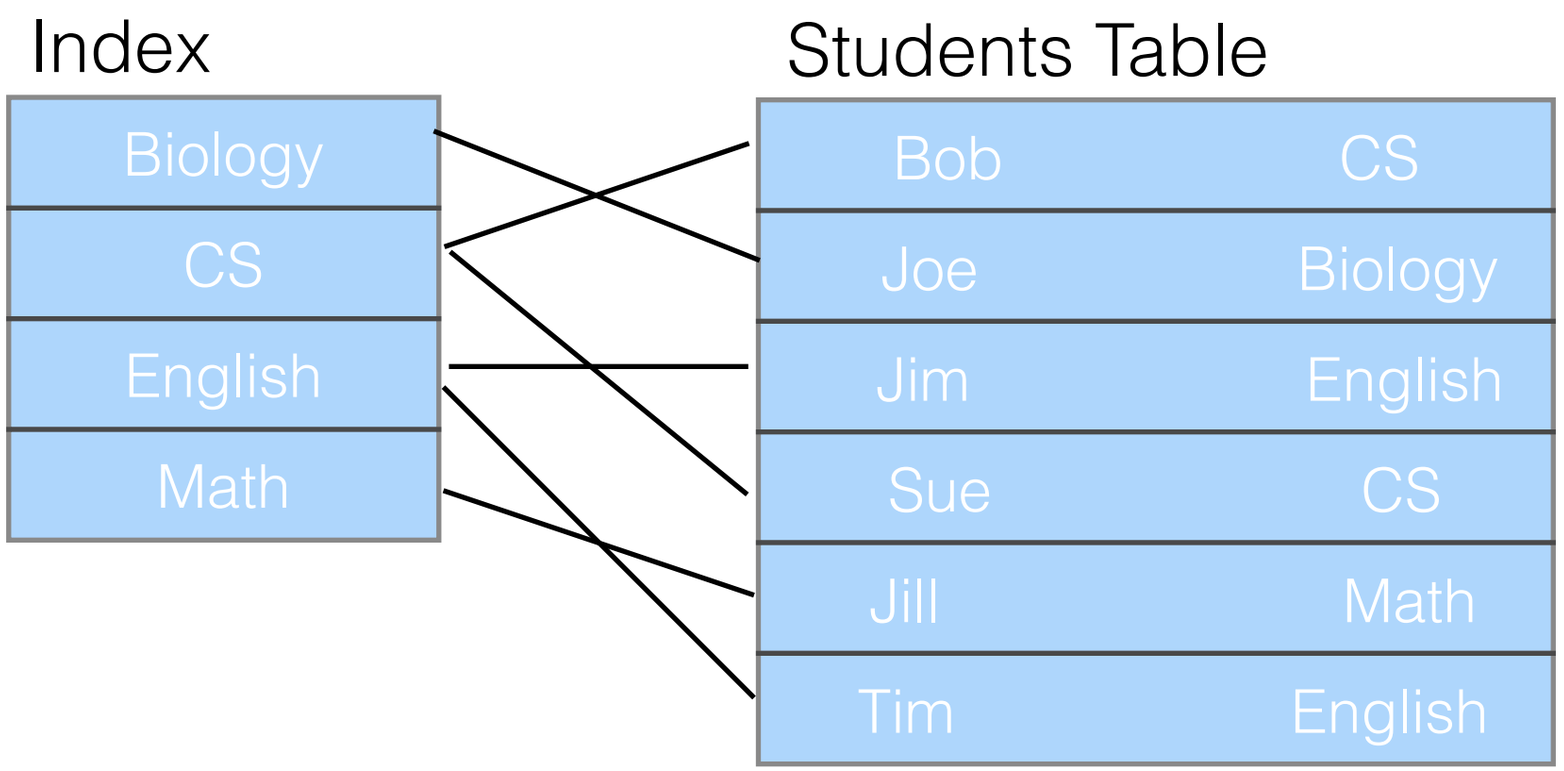


3 Ways to Store Entries in Index

- Alt 1: Actual data stored at index
  - Can have at most one index per table
- Alt 2: Store key and record ID of the matching record
- Alt 3: Store key and list of record IDs

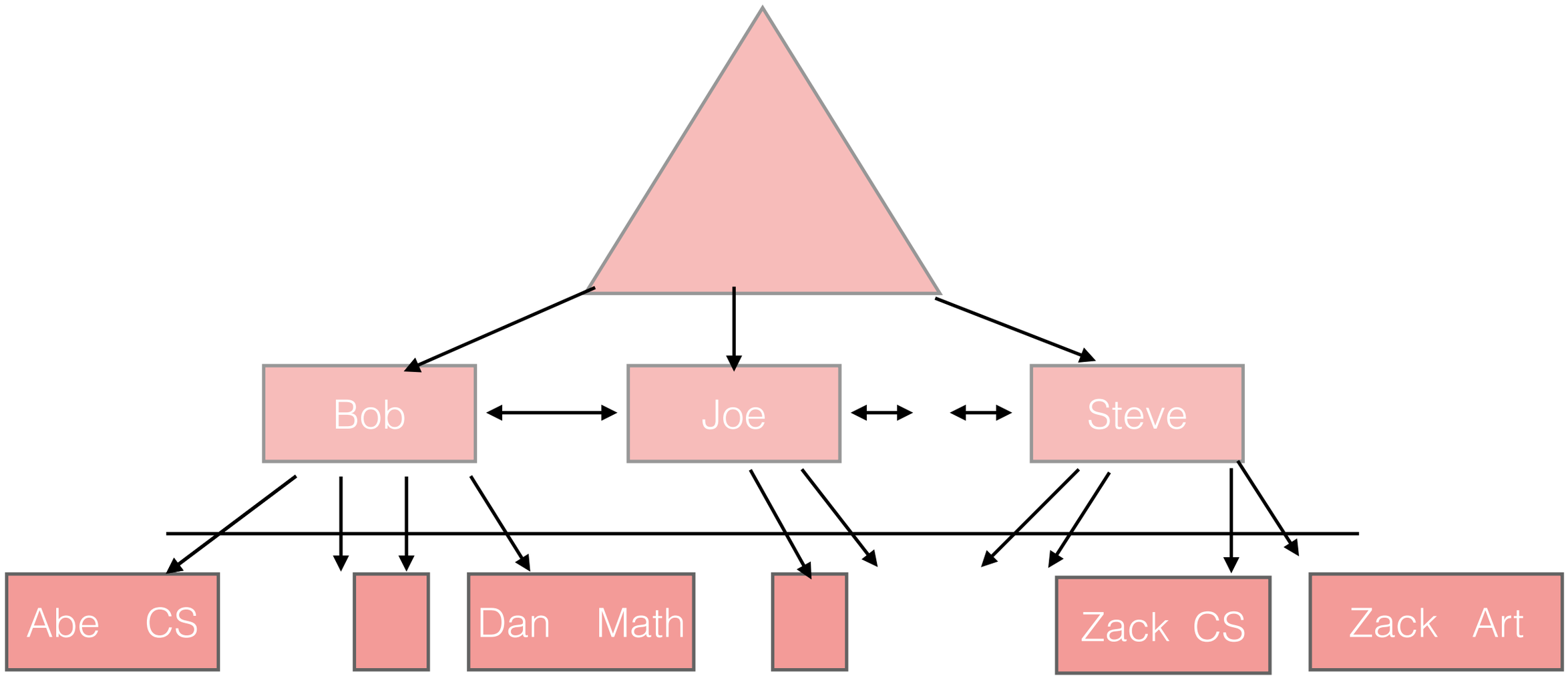
Alt 1

Actual data stored at the index



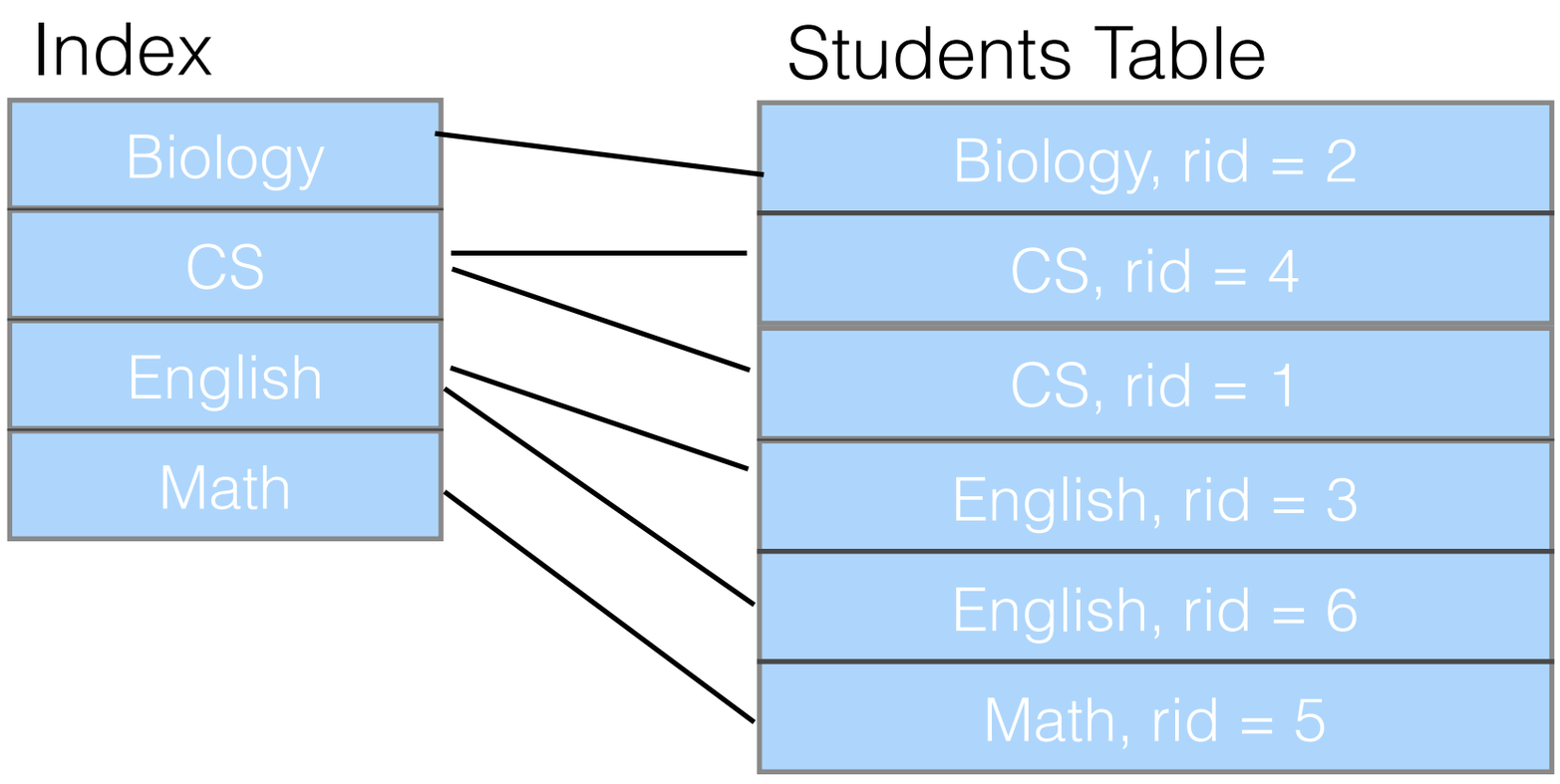
Alt 1

Actual data stored at the index



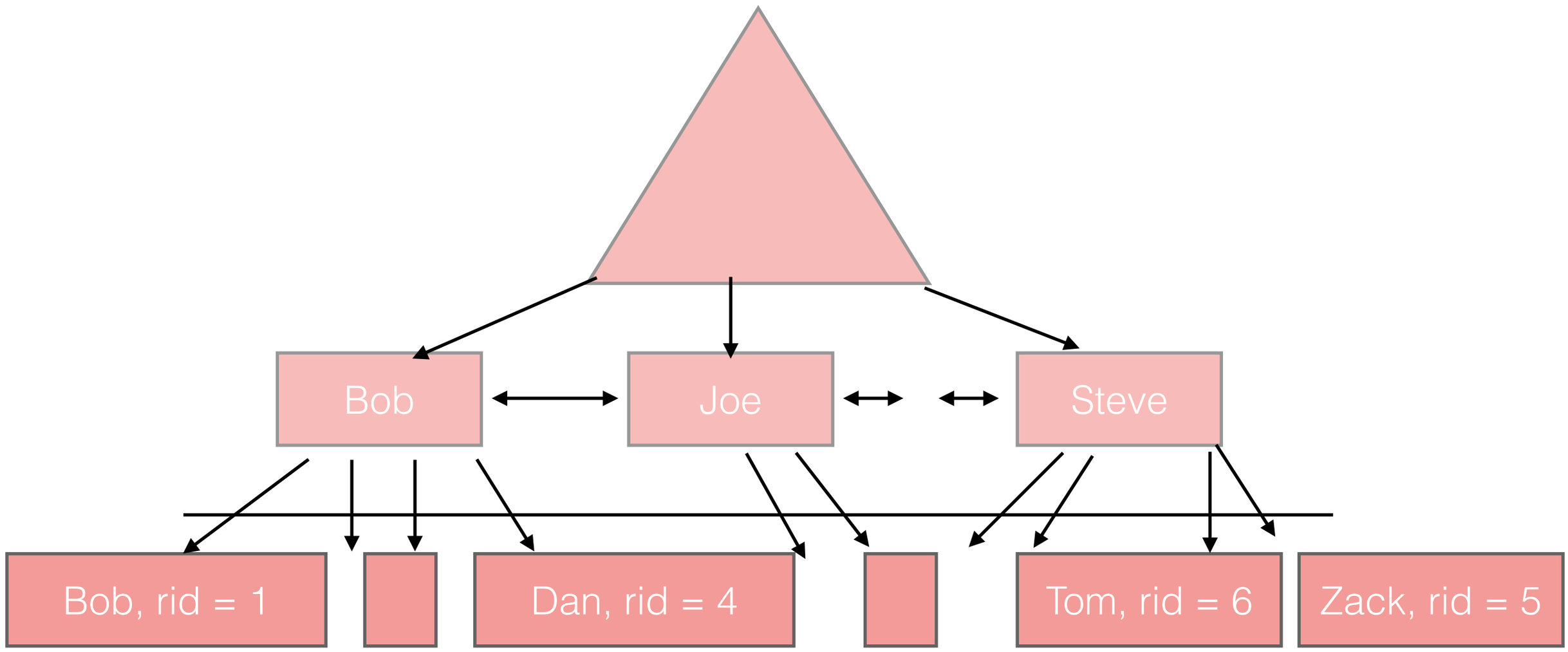
Alt 2

<key, record ID>



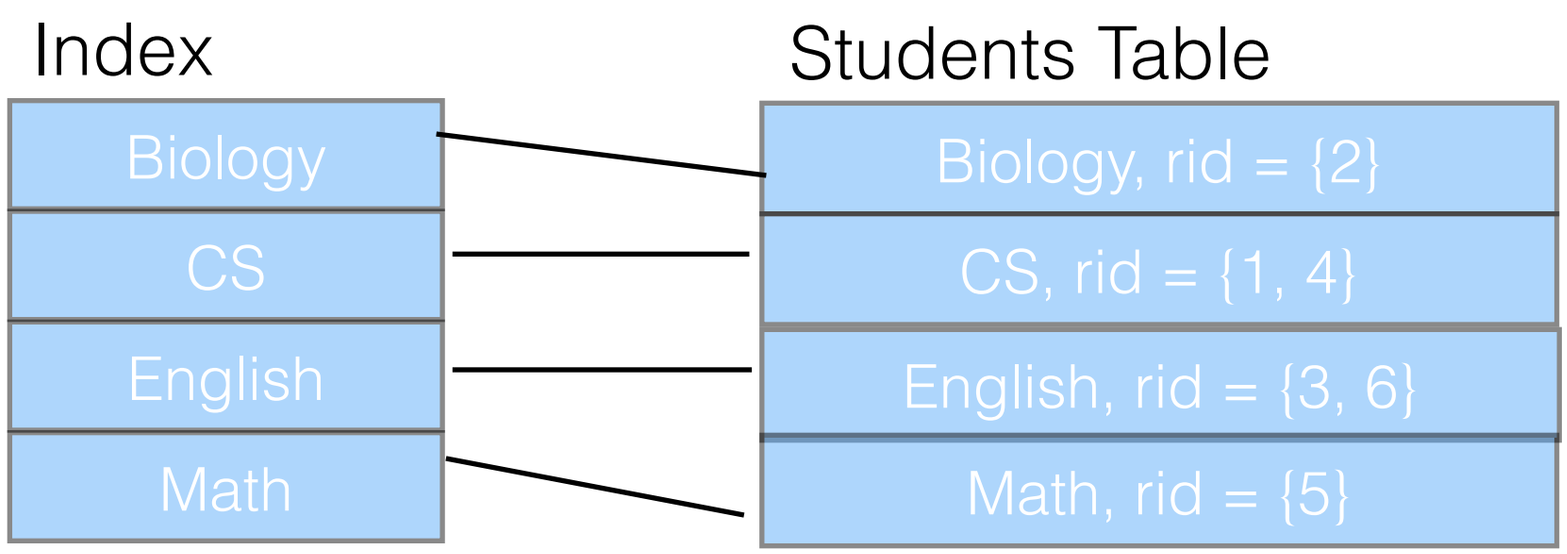
Alt 2

<key, record ID>



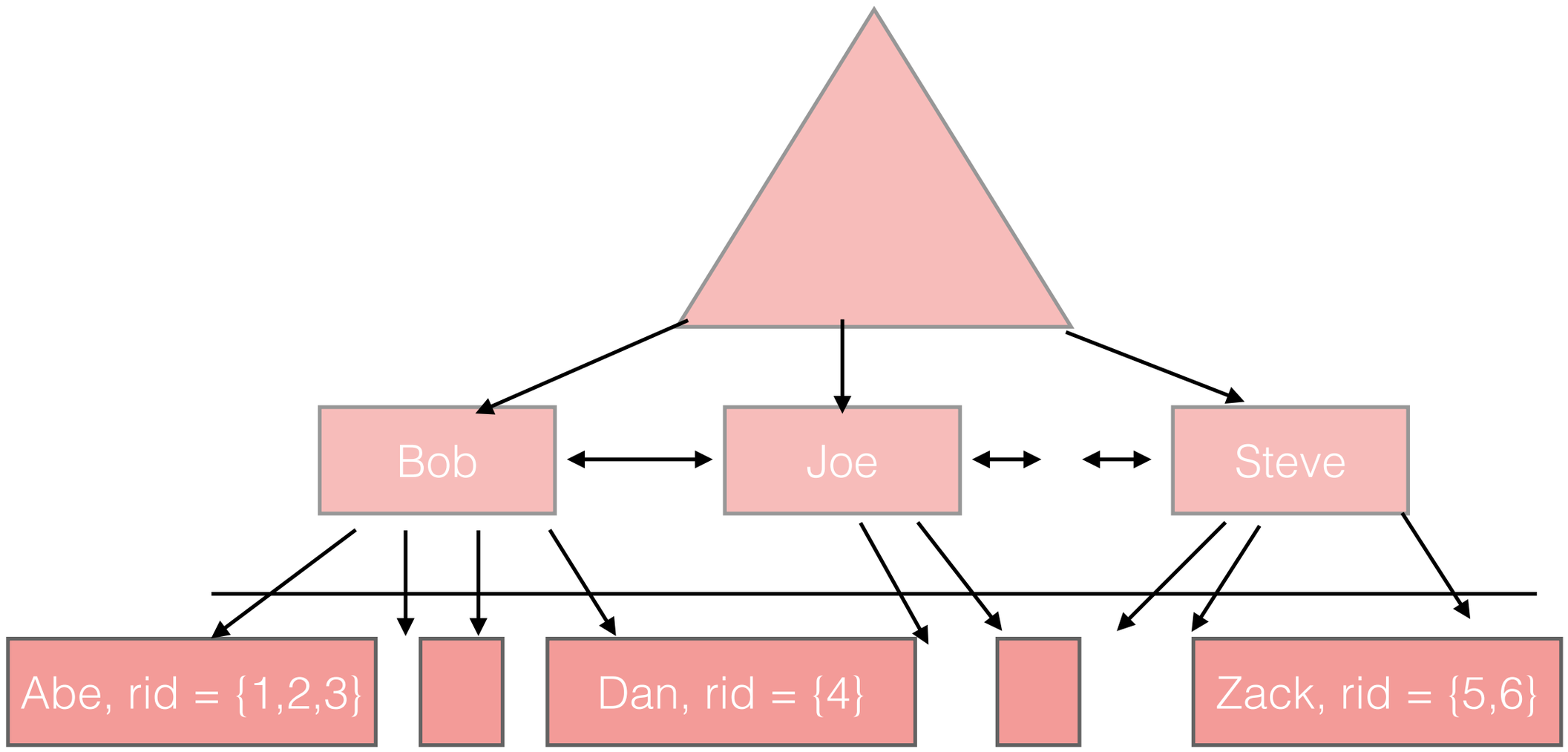
Alt 2

<key, record ID>



Alt 3

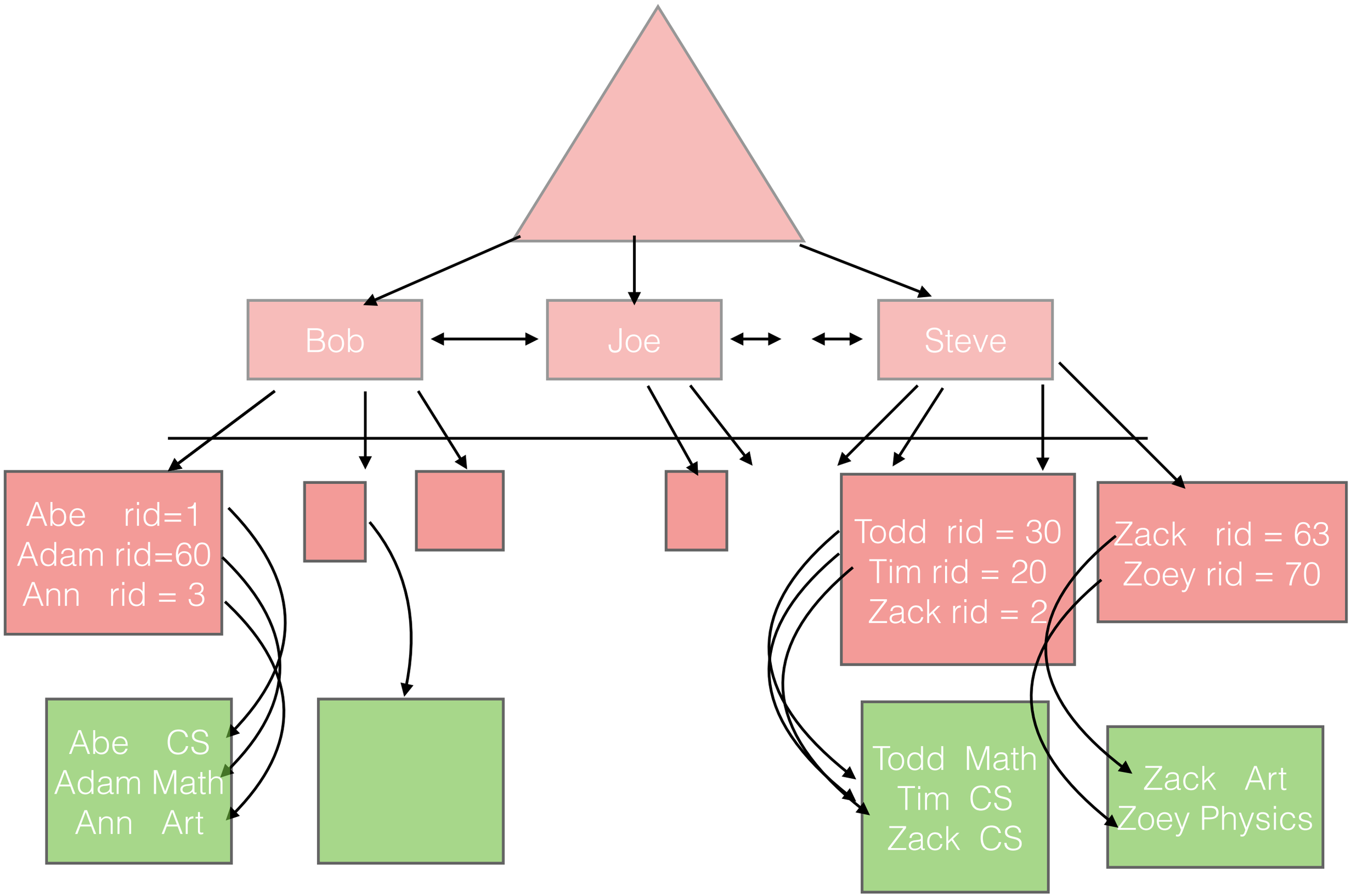
<key, list of matching record IDs>



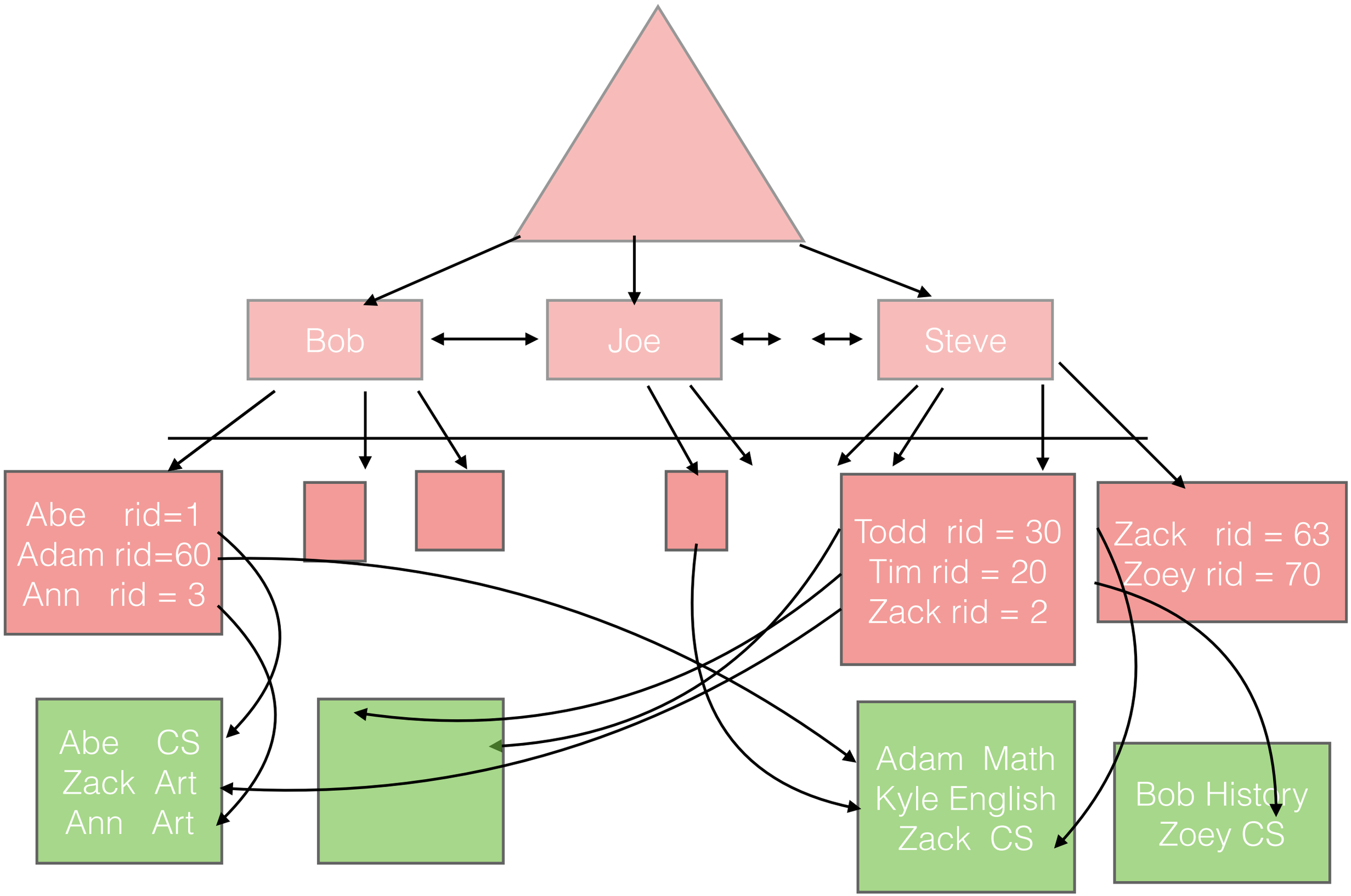
Clustered vs Unclustered

- Clustered - index data entries are stored in (approximate) order by value of search keys in data records
- Can be clustered on at most one key
- Alternative 1 is always clustered

Clustered


Bob
Joe
Steve
Abe rid=1
Adam rid=60
Ann rid = 3
Todd rid = 30
Tim rid = 20
Zack rid = 2
Zack rid = 63
Zoey rid = 70
Abe CS
Adam Math
Ann Art
Todd Math
Tim CS
Zack CS
Zack Art
Zoey Physics

Unclustered



Operation	Heap File	Sorted File	Clustered File
Scan all records	B	B	1.5B
Equality Search	0.5B	log_2(B)	(log_F1.5B + 1)
Range Search	B	log_2(B) + # pages matched	(log_F1.5B + #matches)
Insert	2	log_2(B)+ (B/2) *2	(log_F1.5B + 2)
Delete	0.5B+1	log_2(B)+ (B/2) *2	(log_F1.5B + 2)

Worksheet: Indexing

What are important factors in determining whether or not you should add an index to add to a table?

What are important factors in determining whether or not you should add an index to add to a table?

- Should know which field to cluster on (calculate based on typical queries run)
- Decide if you even want to cluster (high maintenance cost)

Clowns (Name text, SID int, Experience int,
Age int)

RID: 022 - Casey | 134234 | 2 | 35

RID: 051 - Merk | 955551 | 2 | 35

RID: 101 - Test | 045671 | 11 | 15

```
Clowns (Name text, SID int, Experience int,
                    Age int)
    RID: 022 - Casey | 134234 | 2  | 35
    RID: 051 - Merk  | 955551 | 2  | 35
    RID: 101 - Test  | 045671 | 11 | 15
```

Alternative 1

```
Casey | 134234 | 2  | 35
Merk  | 955551 | 2  | 35
Test  | 045671 | 11 | 15
```

```
Clowns (Name text, SID int, Experience int,
                    Age int)
    RID: 022 - Casey | 134234 | 2  | 35
    RID: 051 - Merk  | 955551 | 2  | 35
    RID: 101 - Test  | 045671 | 11 | 15
```

Alternative 2

<key, RID>

<35, 022>
<35, 051>
<15, 101>

```
Clowns (Name text, SID int, Experience int,
                    Age int)
    RID: 022 - Casey | 134234 | 2  | 35
    RID: 051 - Merk  | 955551 | 2  | 35
    RID: 101 - Test  | 045671 | 11 | 15
```

Alternative 3

<key, [RID list]>

<35, [022, 051]>
<15, [101]>

True or False? Given the table Students(sid char(20), gpa float, age integer), a clustered tree based index on gpa will increase the performance of the following query:
SELECT * FROM Students where age > 20 AND gpa > 3.5;

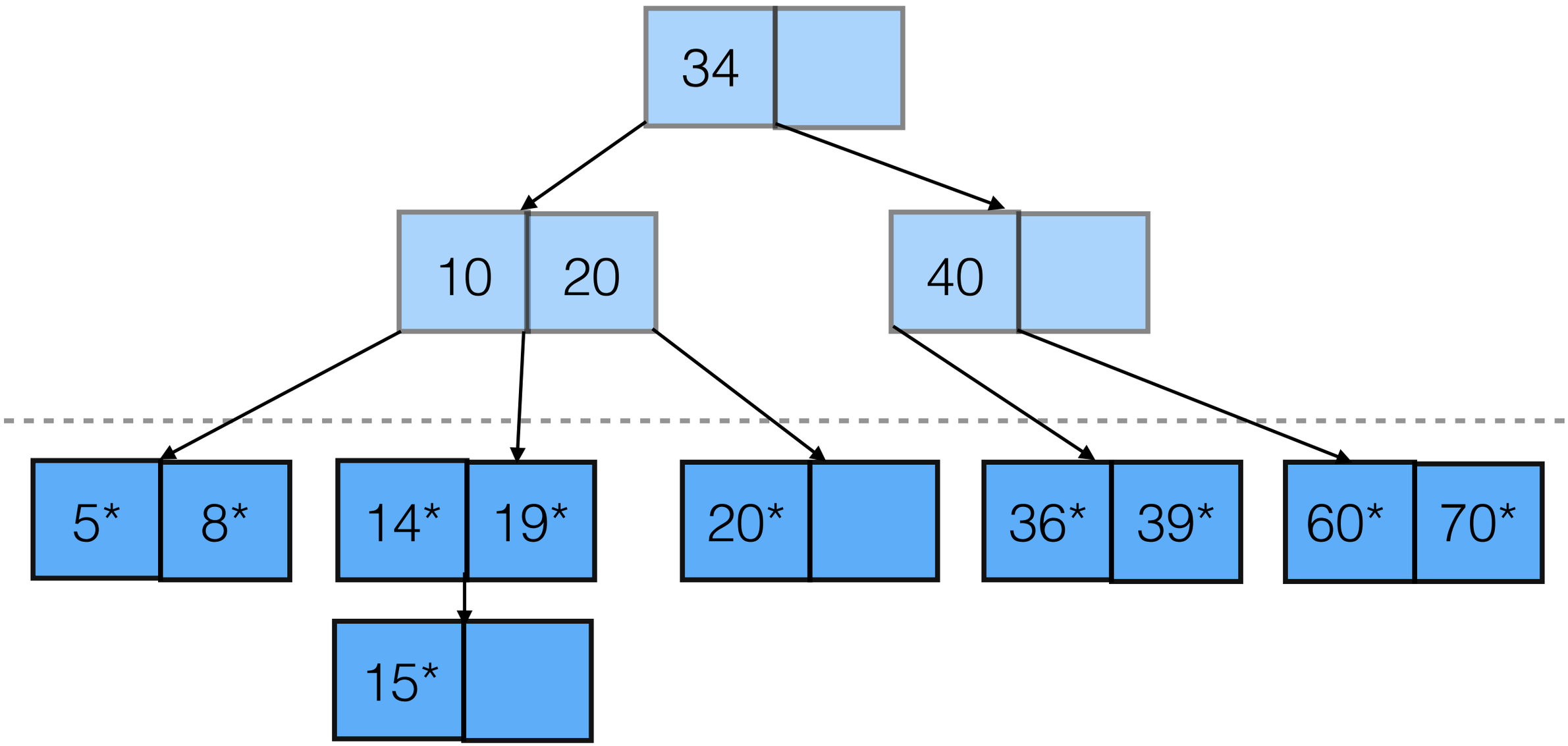
True or False? Given the table Students(sid char(20), gpa float, age integer), a clustered tree based index on gpa will increase the performance of the following query:
SELECT * FROM Students where age > 20 AND gpa > 3.5;

TRUE

ISAM

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

ISAM

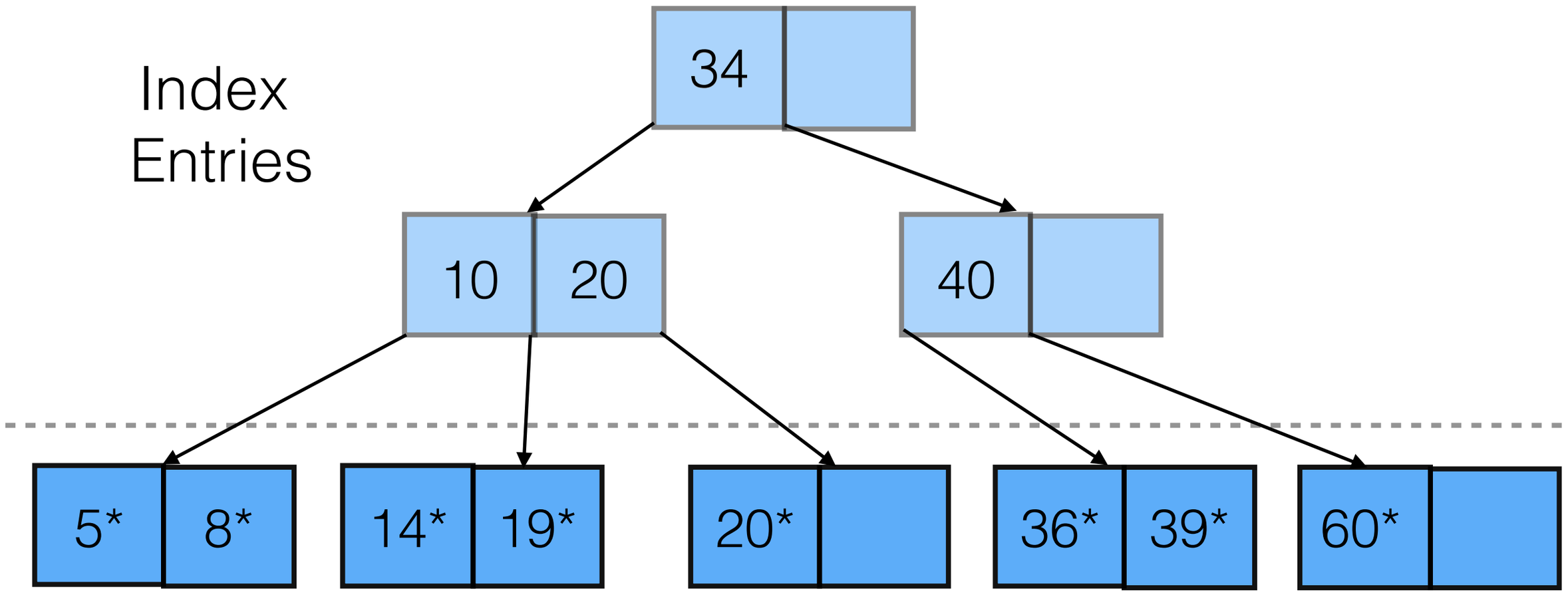


ISAM - Insert X

- 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

ISAM

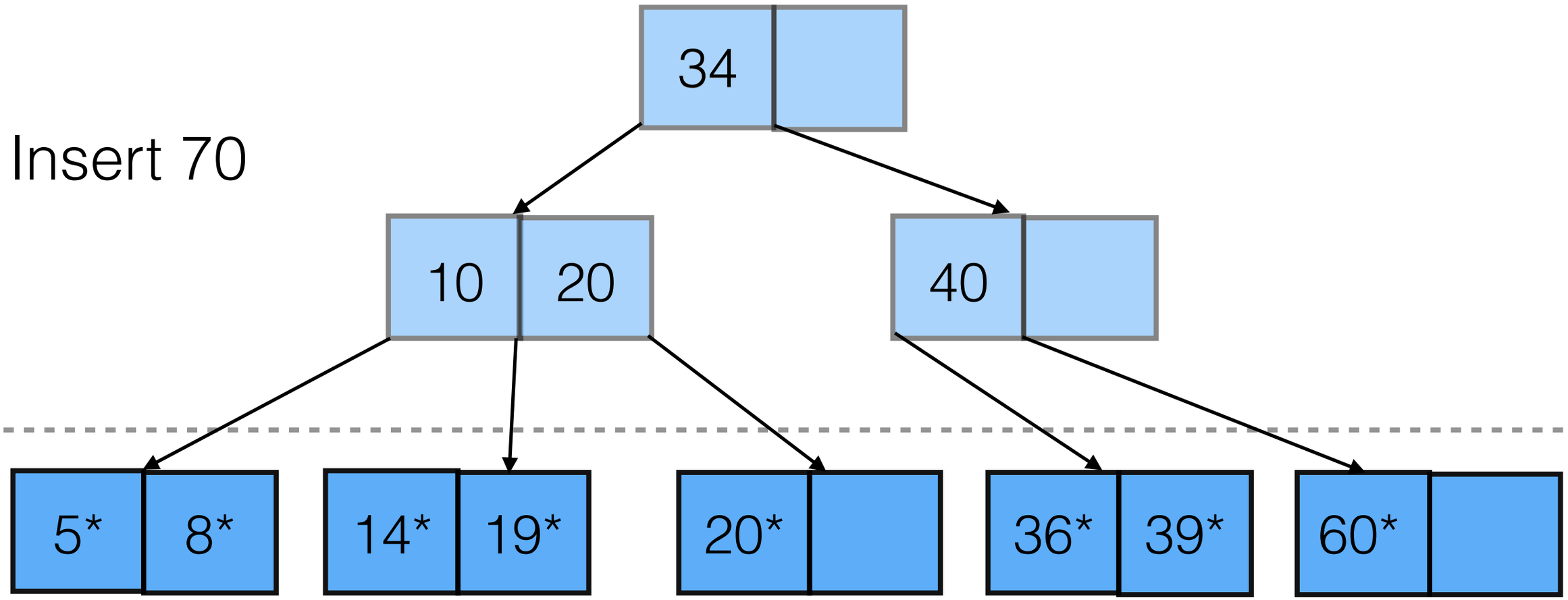
Index
Entries



Leaf Pages
with Data Entries

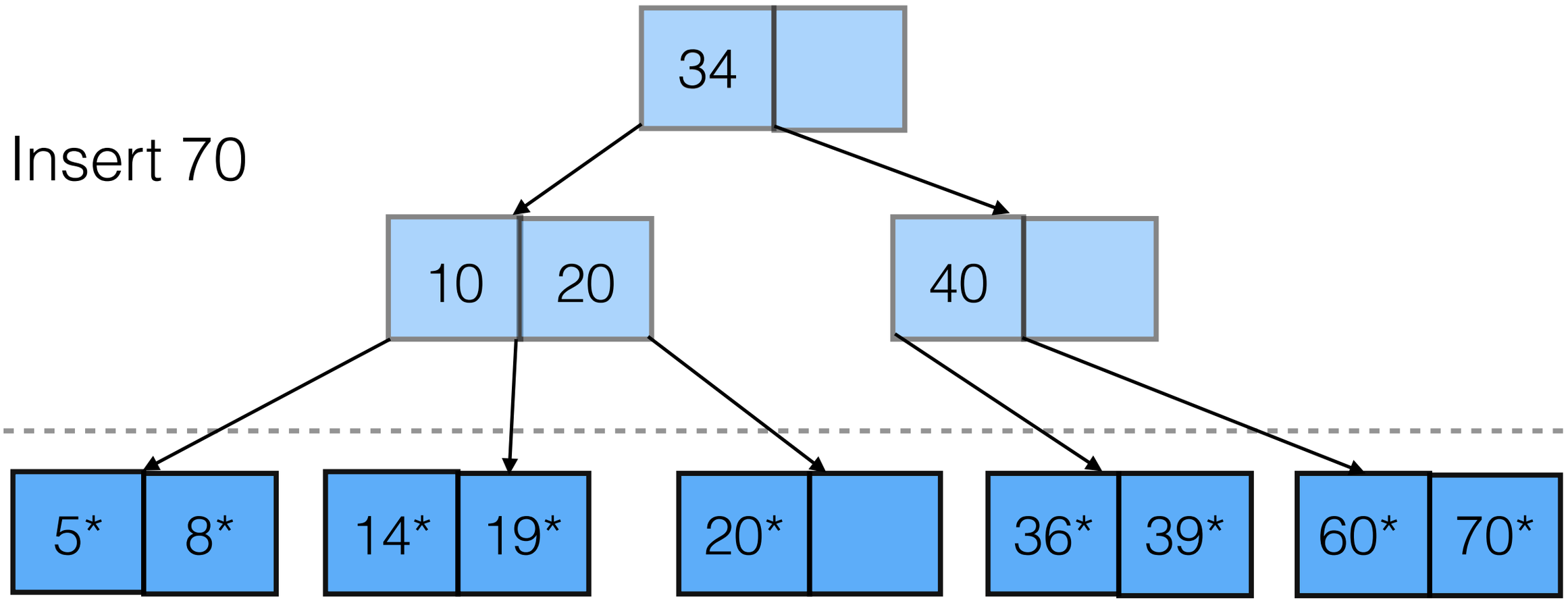
ISAM

Insert 70



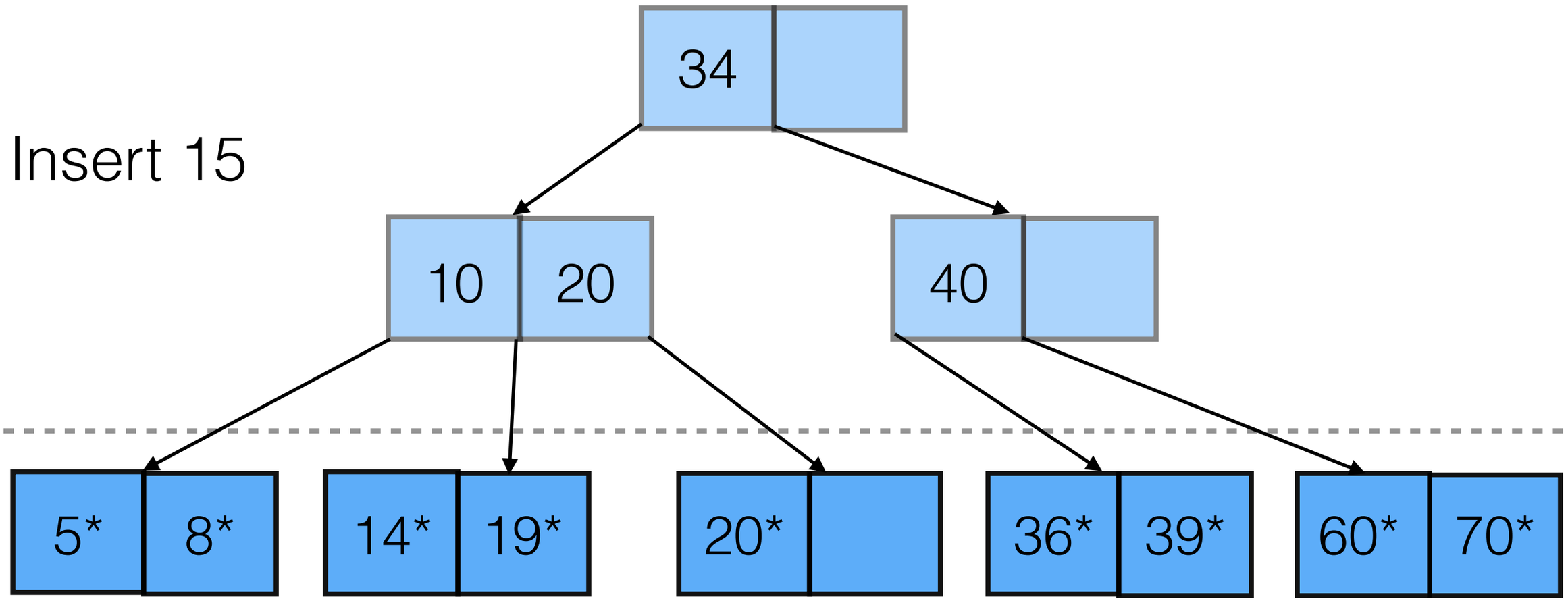
ISAM

Insert 70



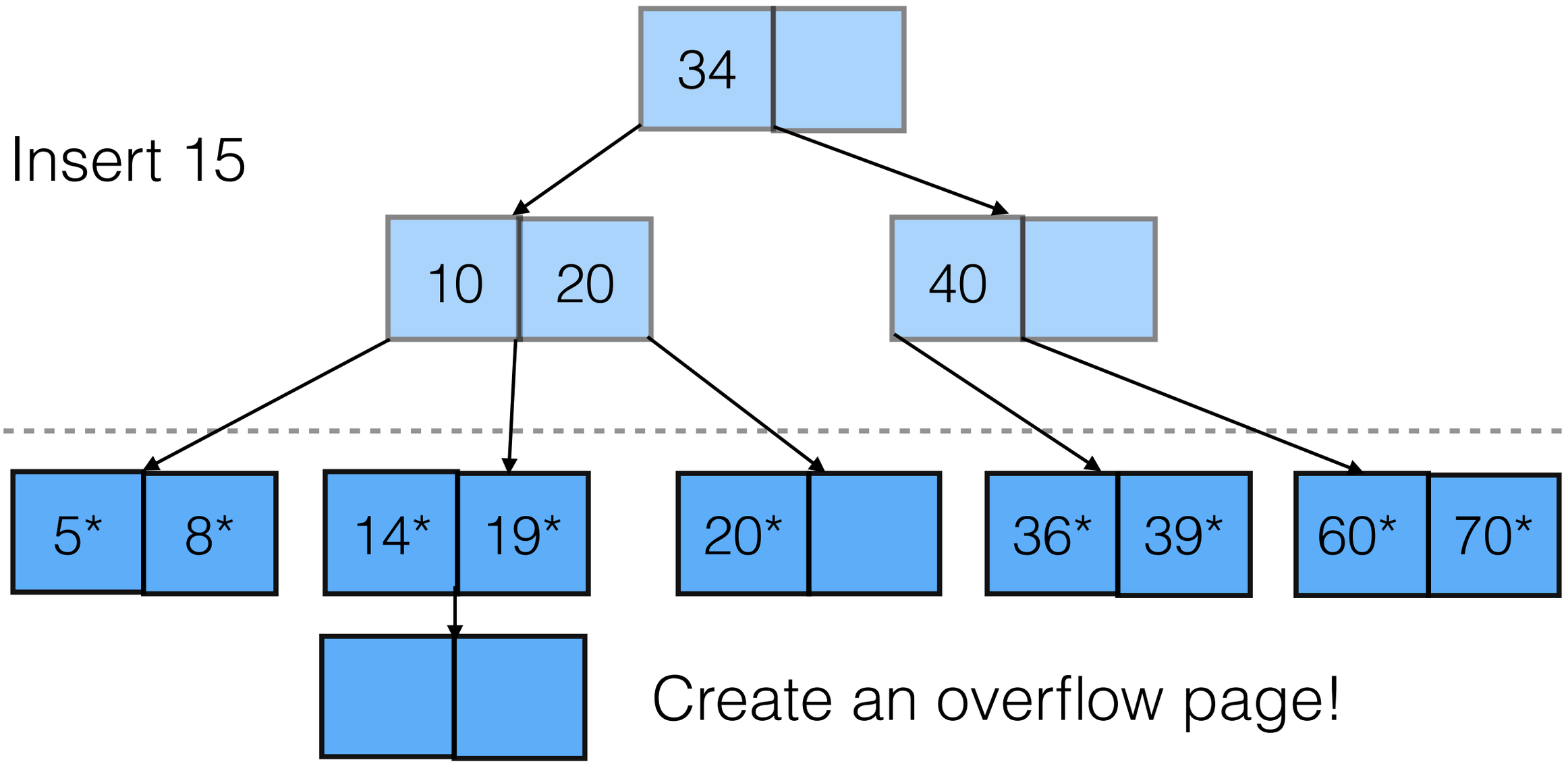
ISAM

Insert 15



ISAM

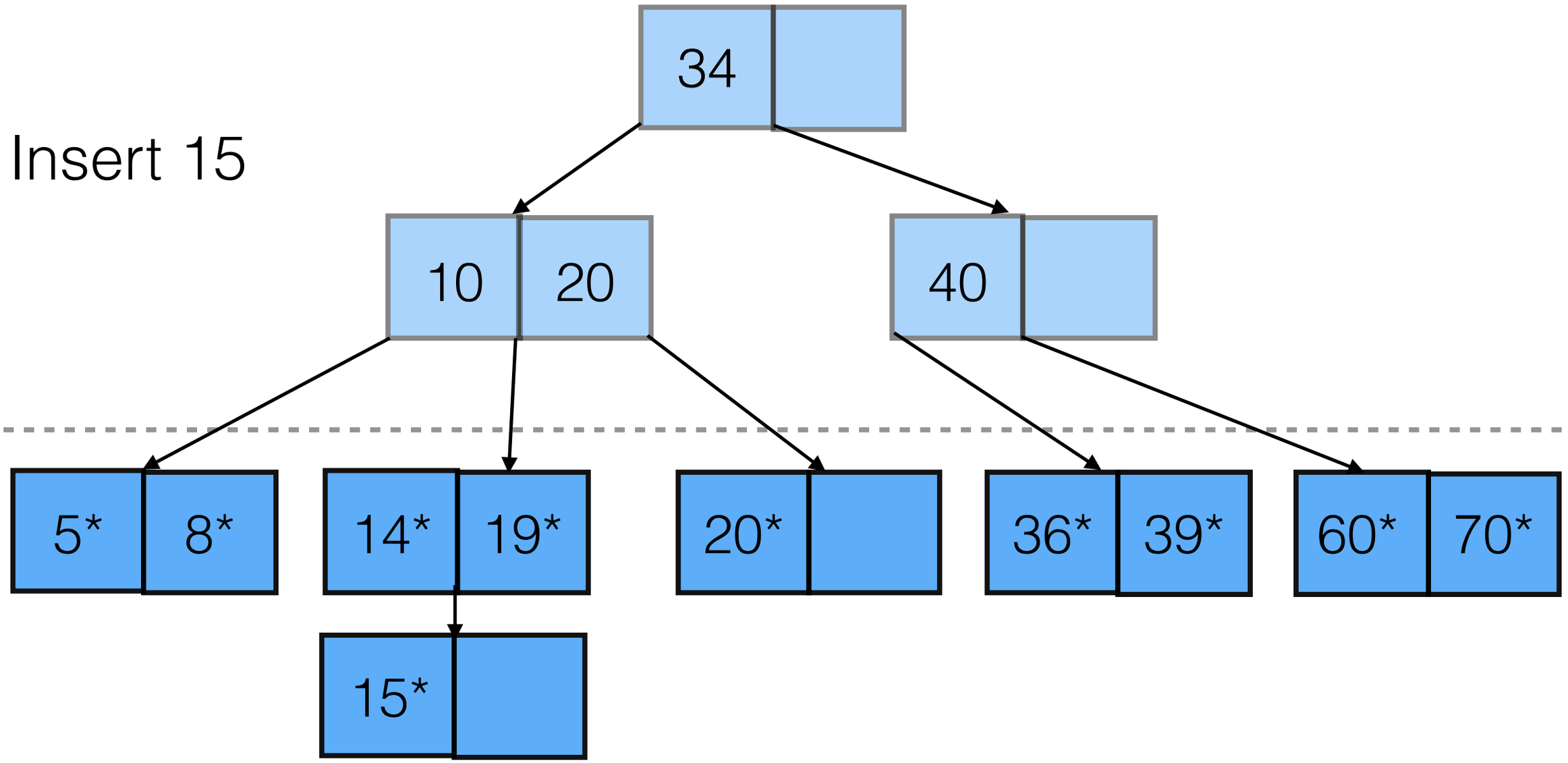
Insert 15



Create an overflow page!

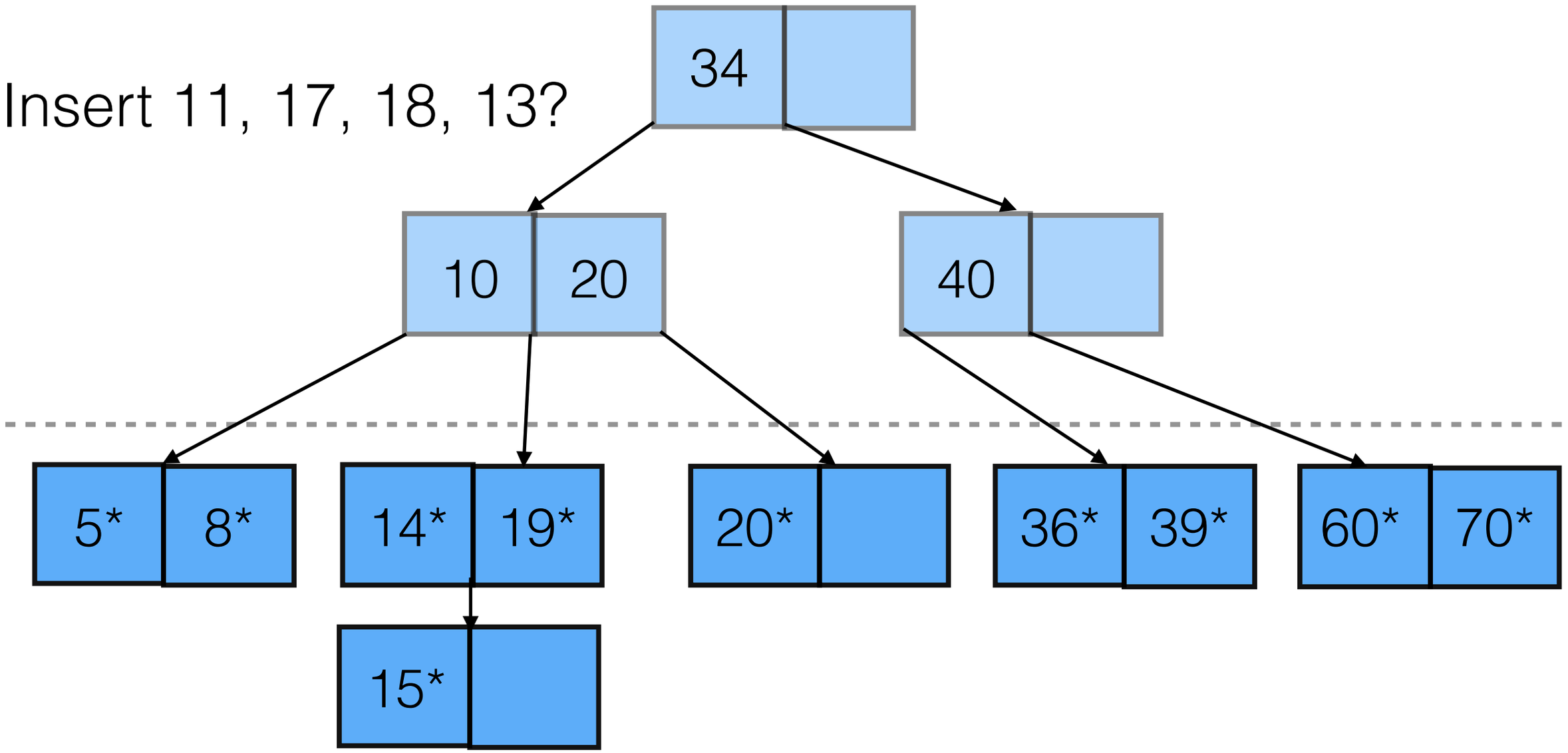
ISAM

Insert 15



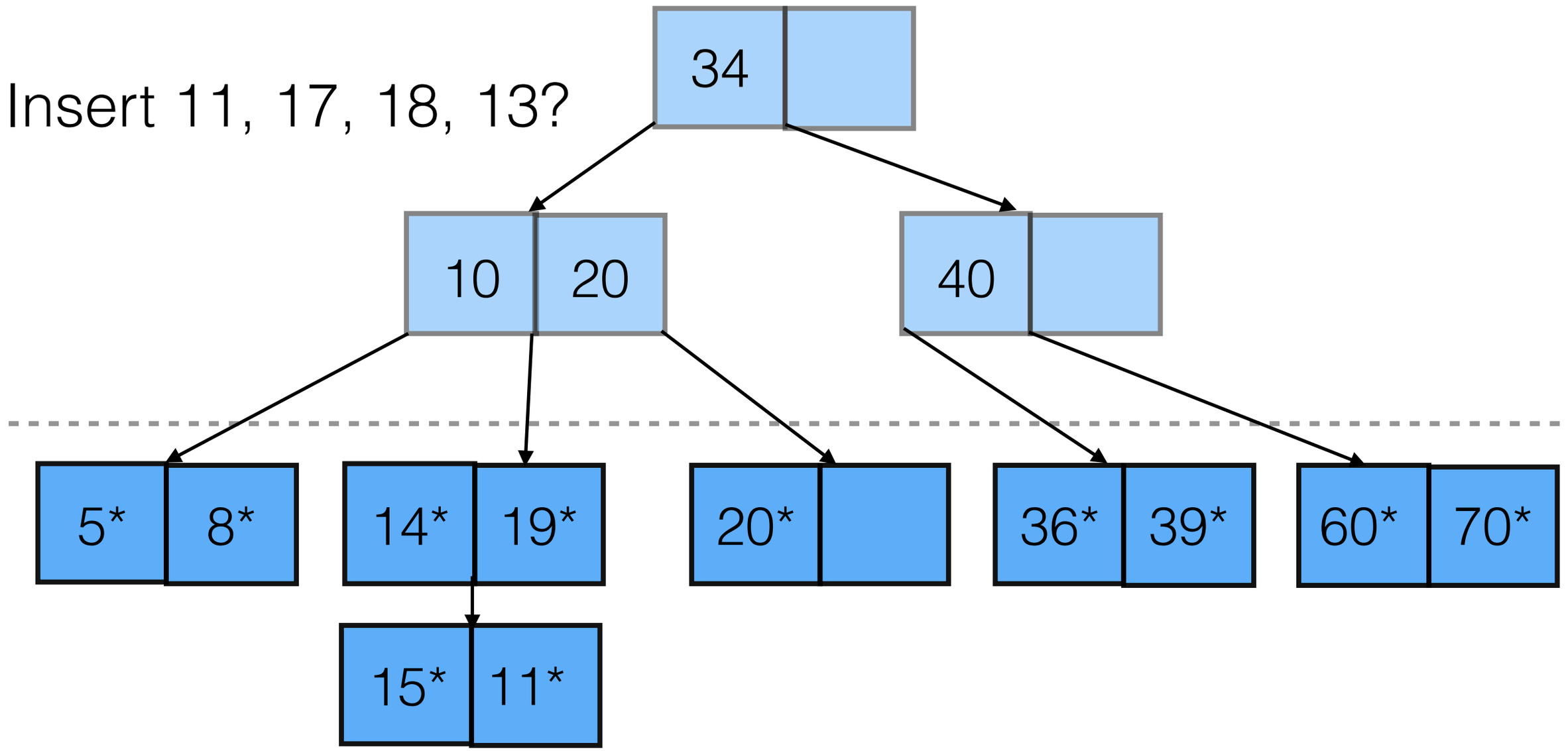
ISAM

Insert 11, 17, 18, 13?



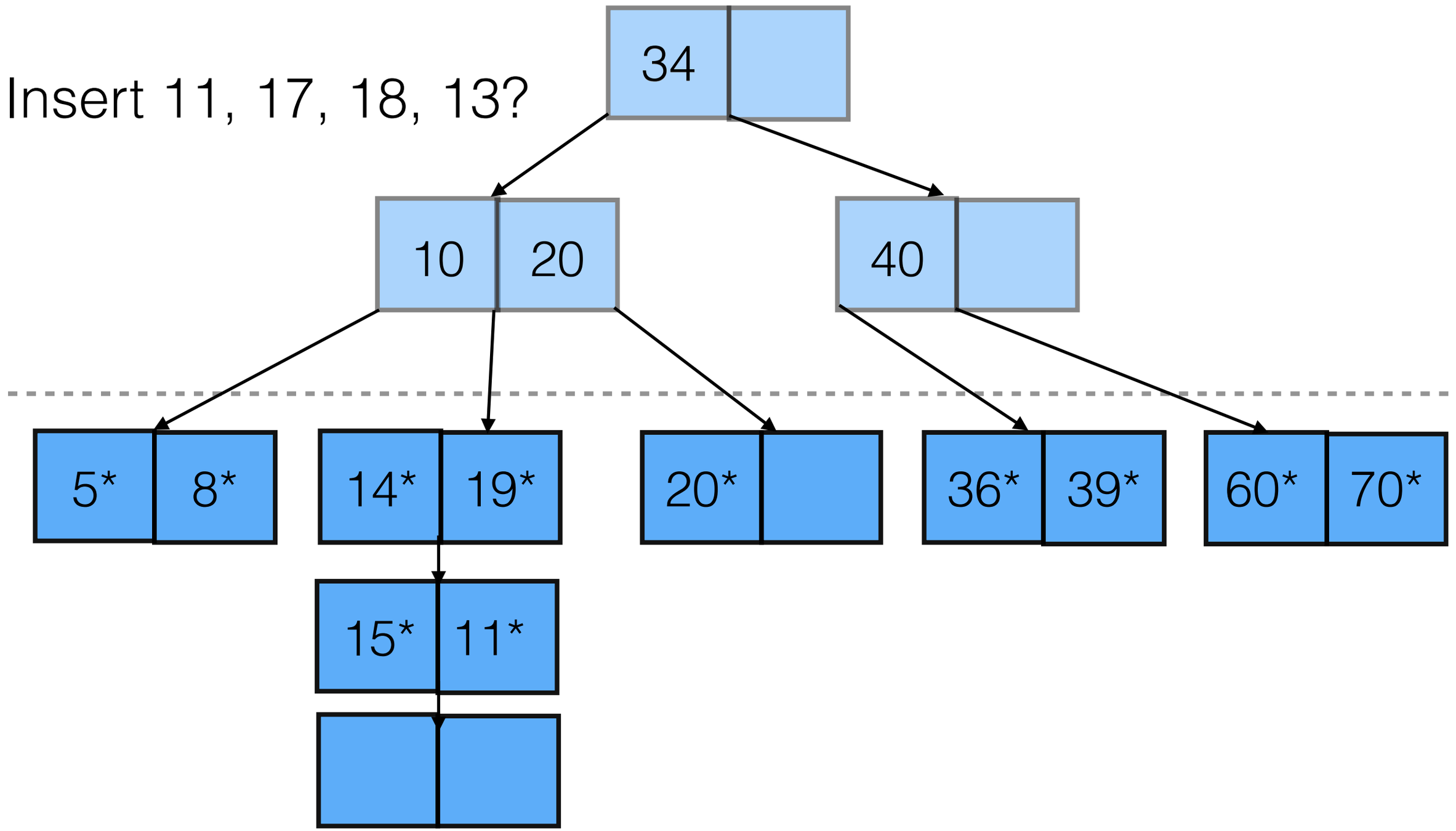
ISAM

Insert 11, 17, 18, 13?



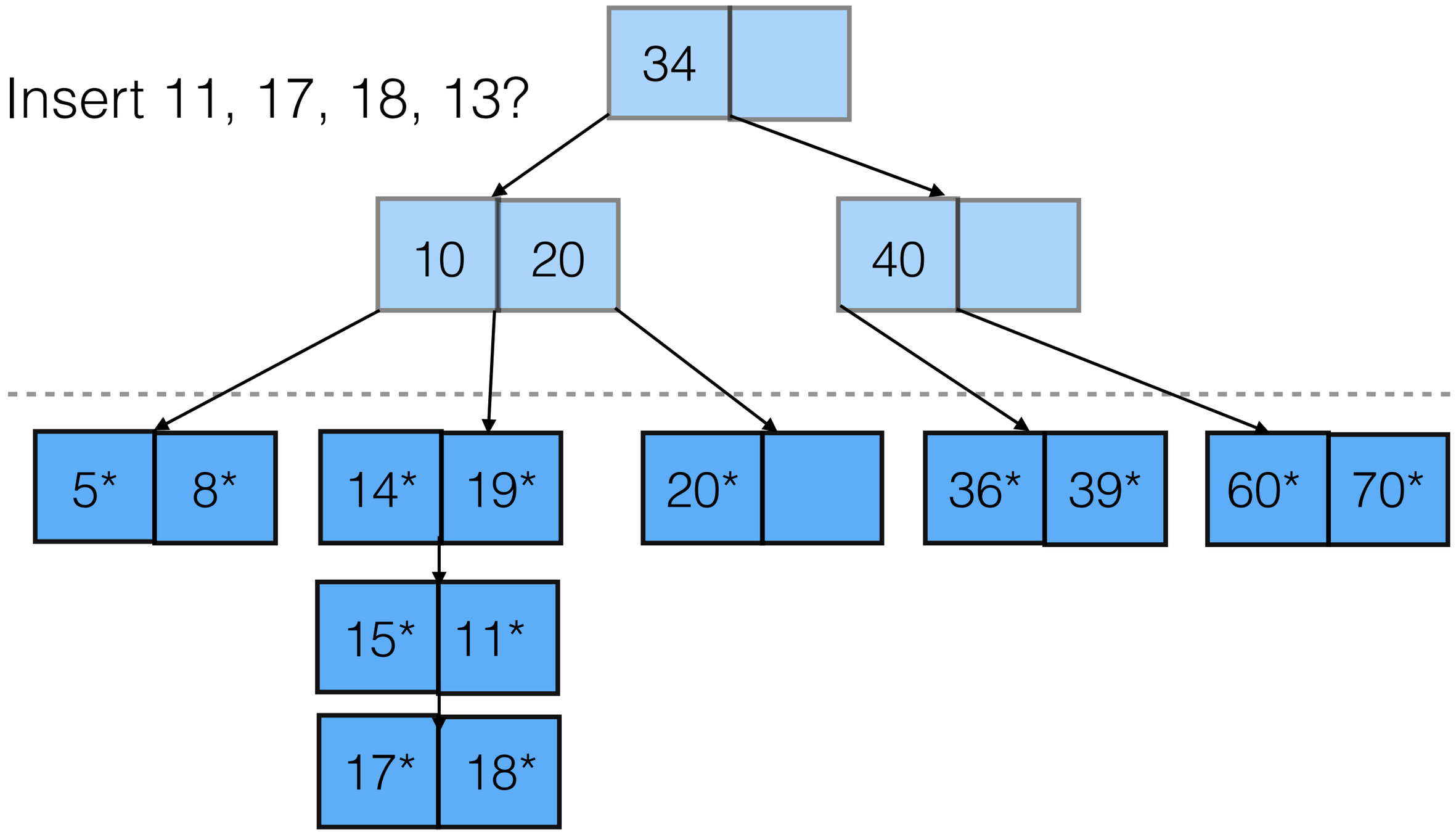
ISAM

Insert 11, 17, 18, 13?



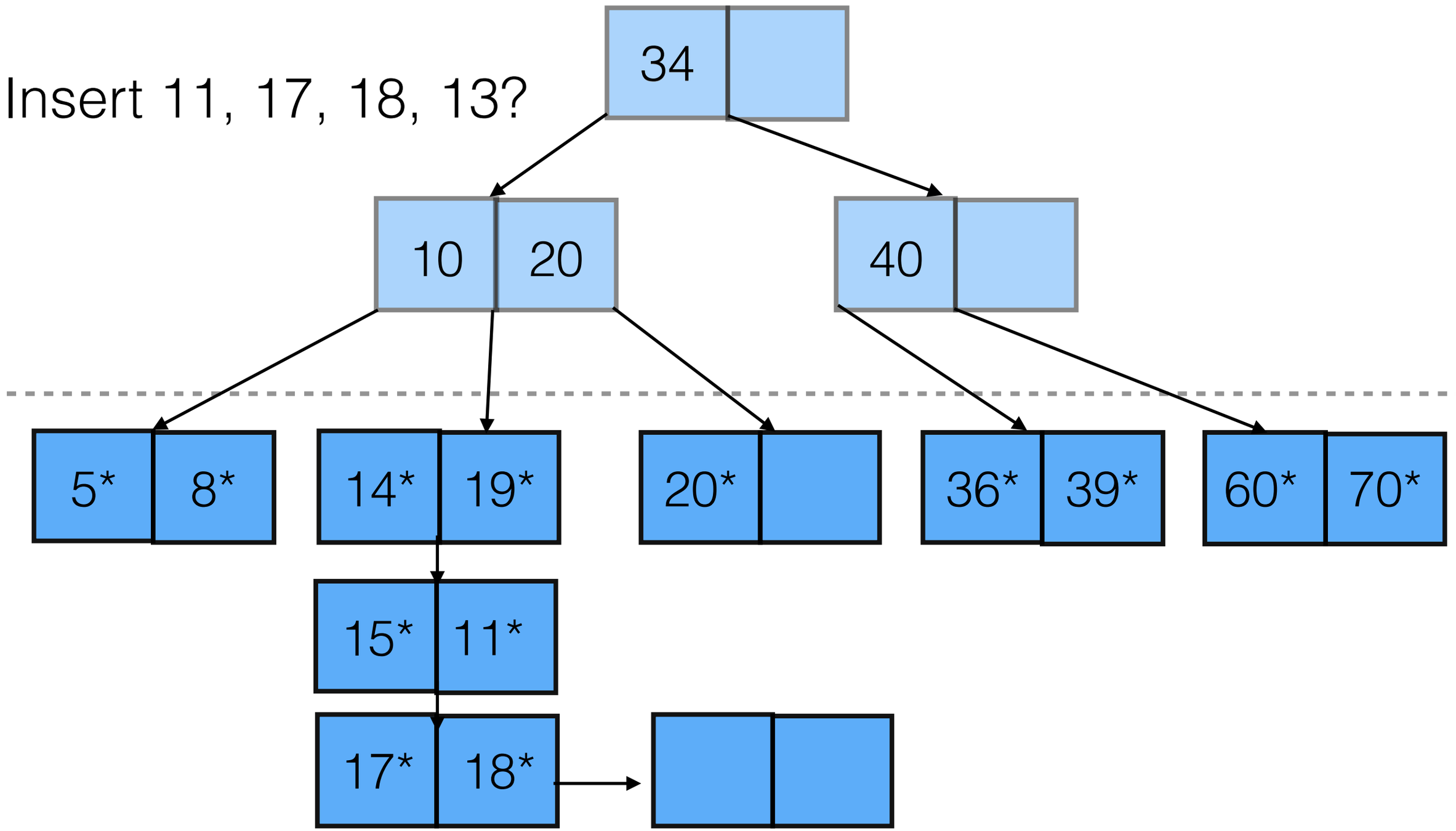
ISAM

Insert 11, 17, 18, 13?



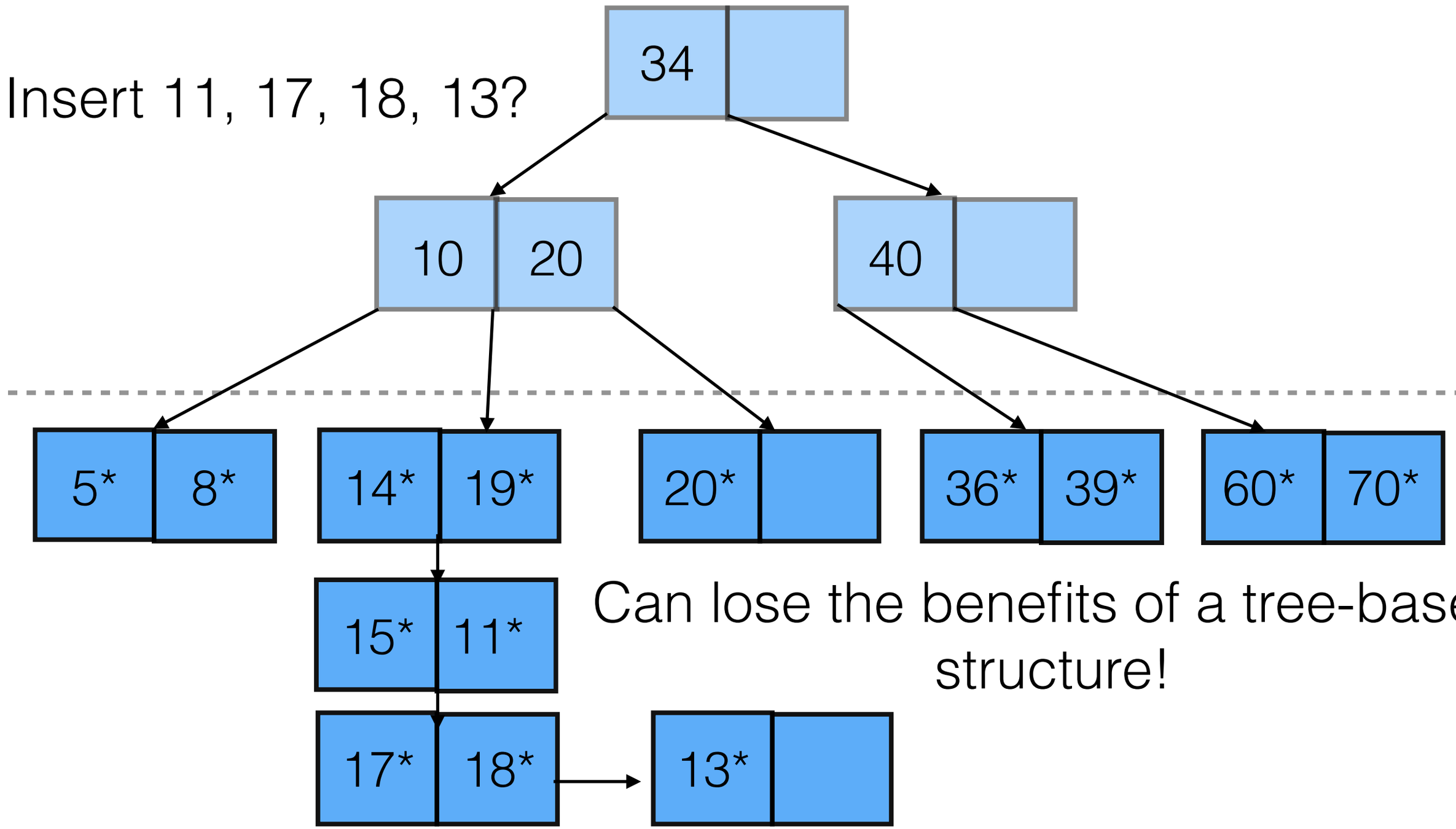
ISAM

Insert 11, 17, 18, 13?



ISAM

Insert 11, 17, 18, 13?



Can lose the benefits of a tree-based structure!

B+ Trees

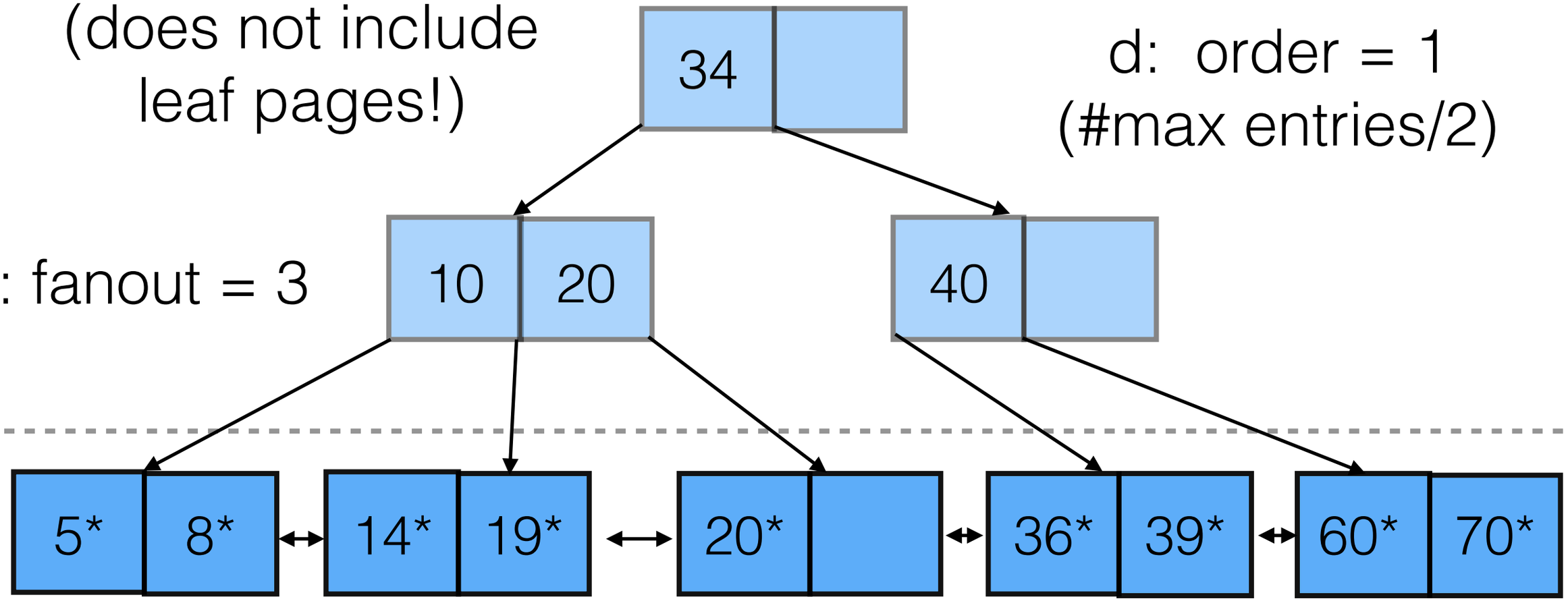
- Dynamic structure to keep tree height-balanced
- Adjusts under inserts and deletes
- Maintain minimum 50% occupancy for each page (except root)

B+ Trees

H: height = 2
(does not include
leaf pages!)

d: order = 1
(#max entries/2)

F: fanout = 3



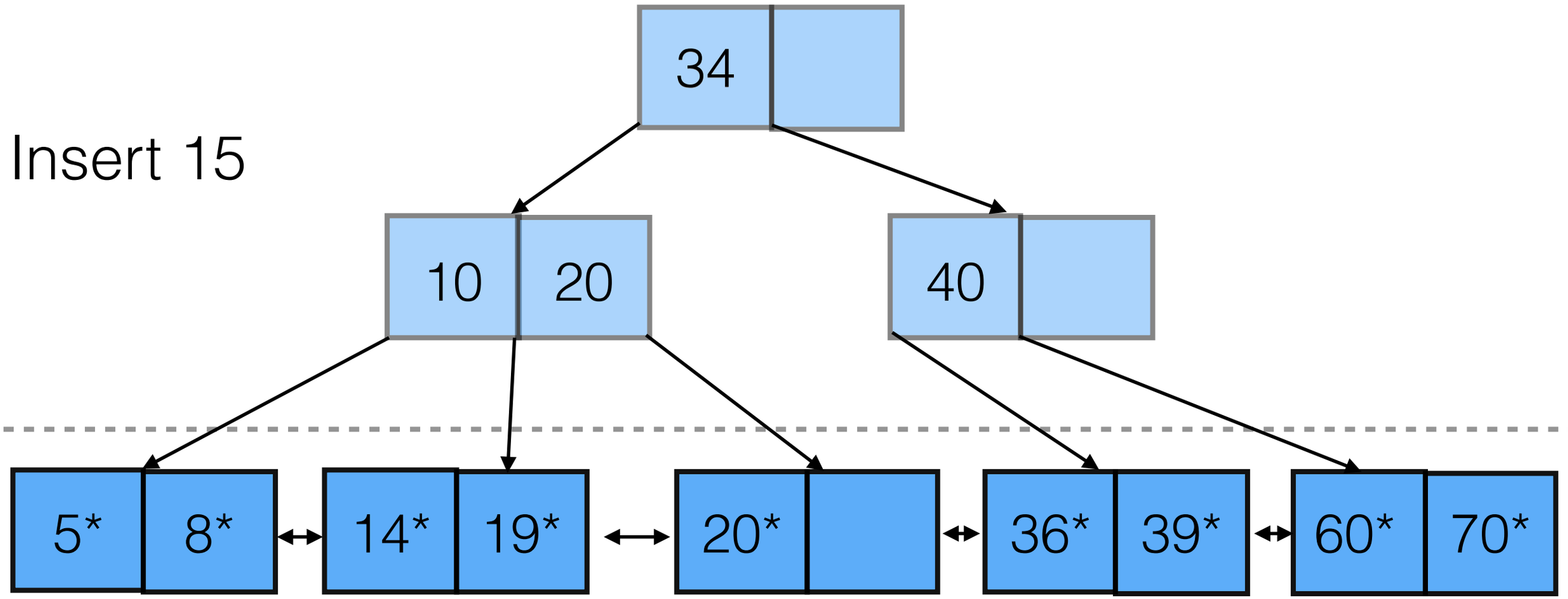
N: leaf pages = 5

B+ Trees - Insert X

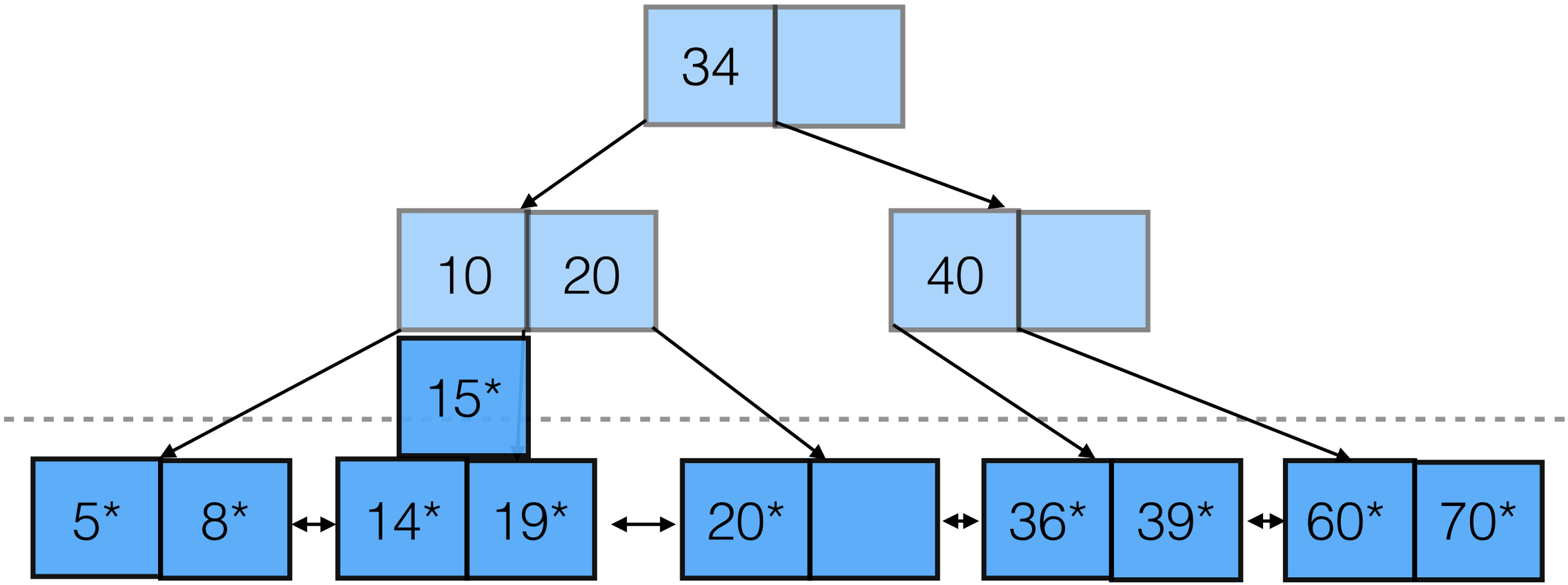
- Find correct leaf L
- Put X in L
  - If not enough space in L:
    - Split L into L and L2
    - **Copy** up middle key to parent
    - If not enough space in parent:
      - Apply algorithm recursively, except **push** up middle key

B+ Trees

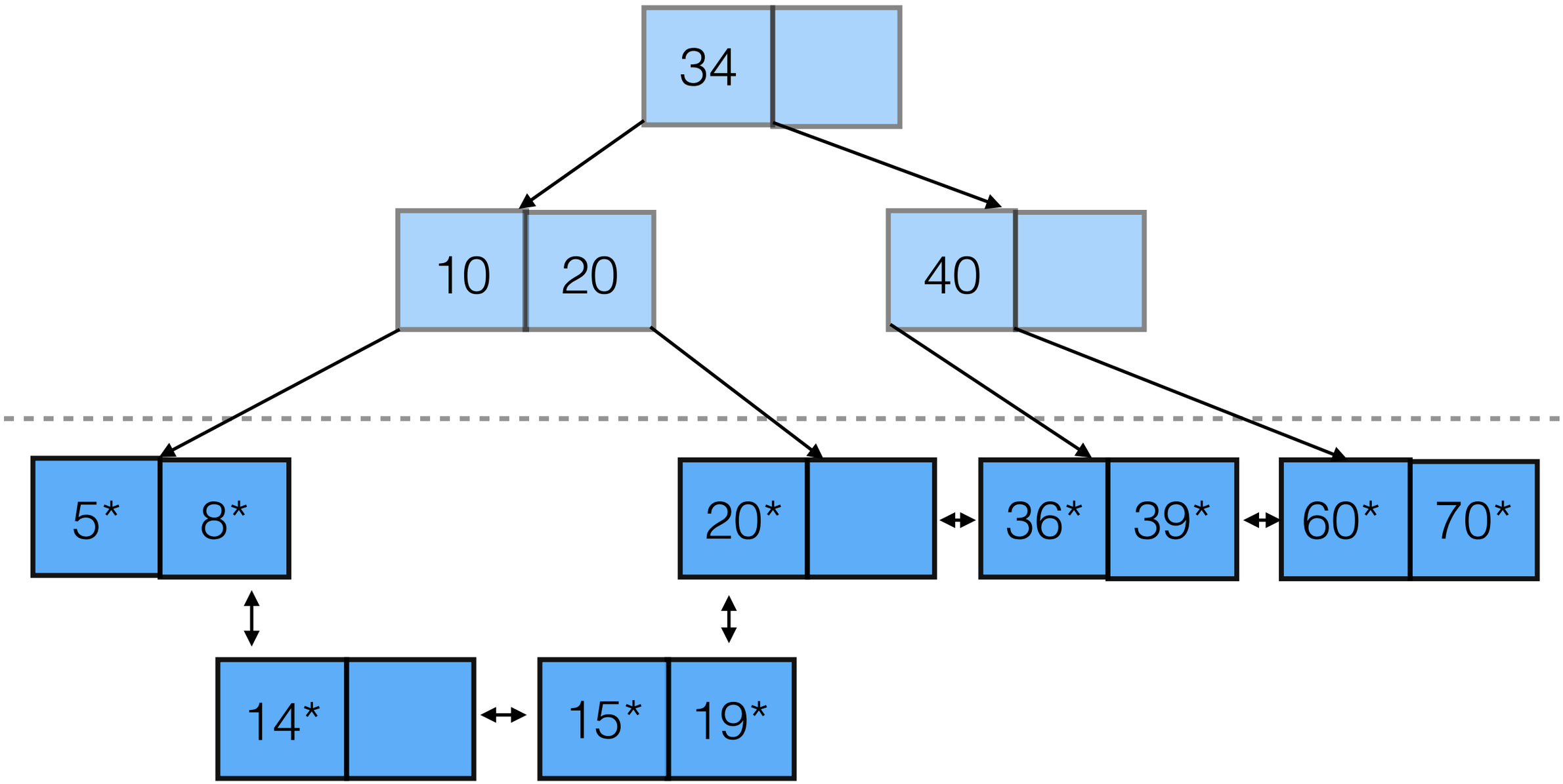
Insert 15



B+ Trees

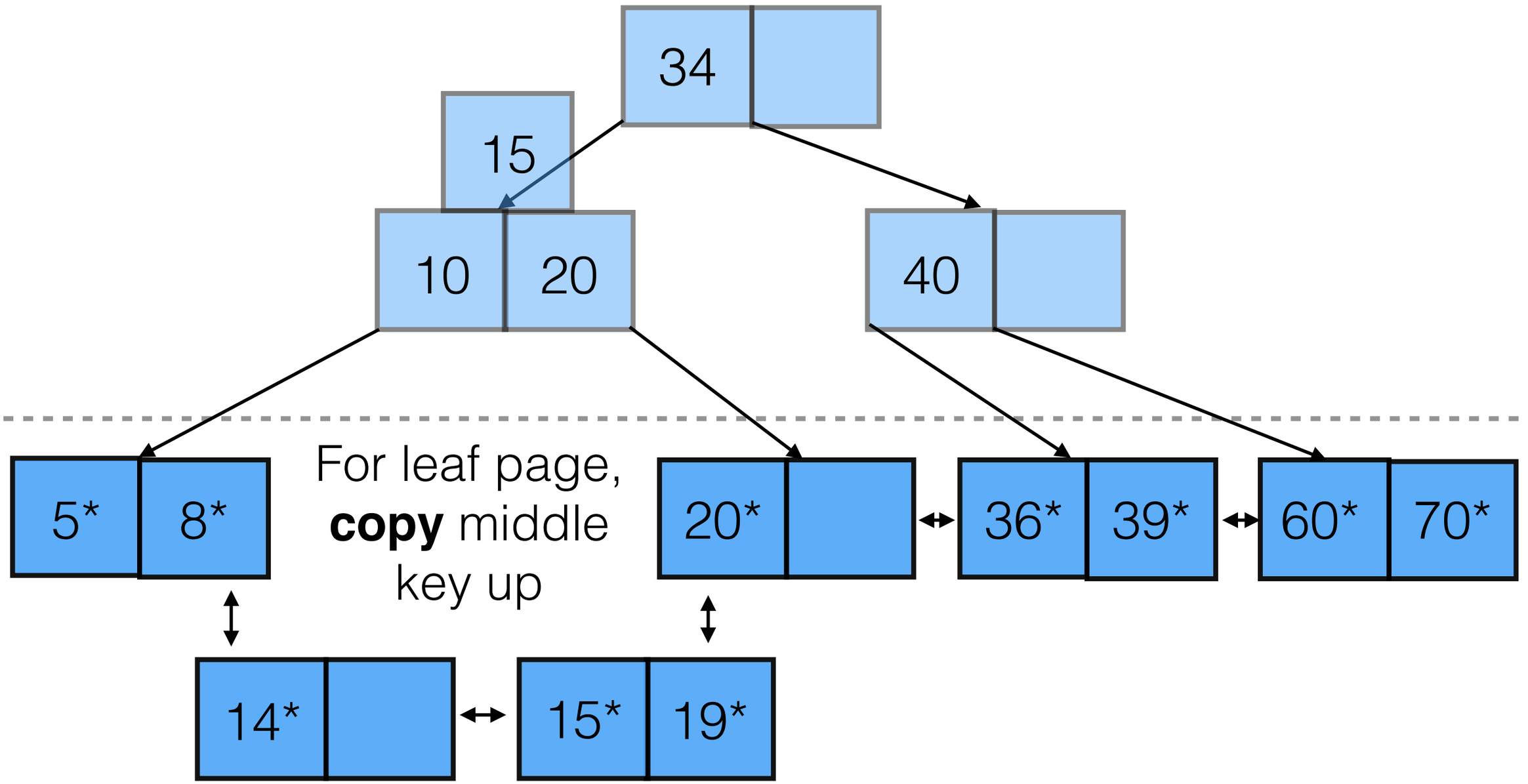


B+ Trees



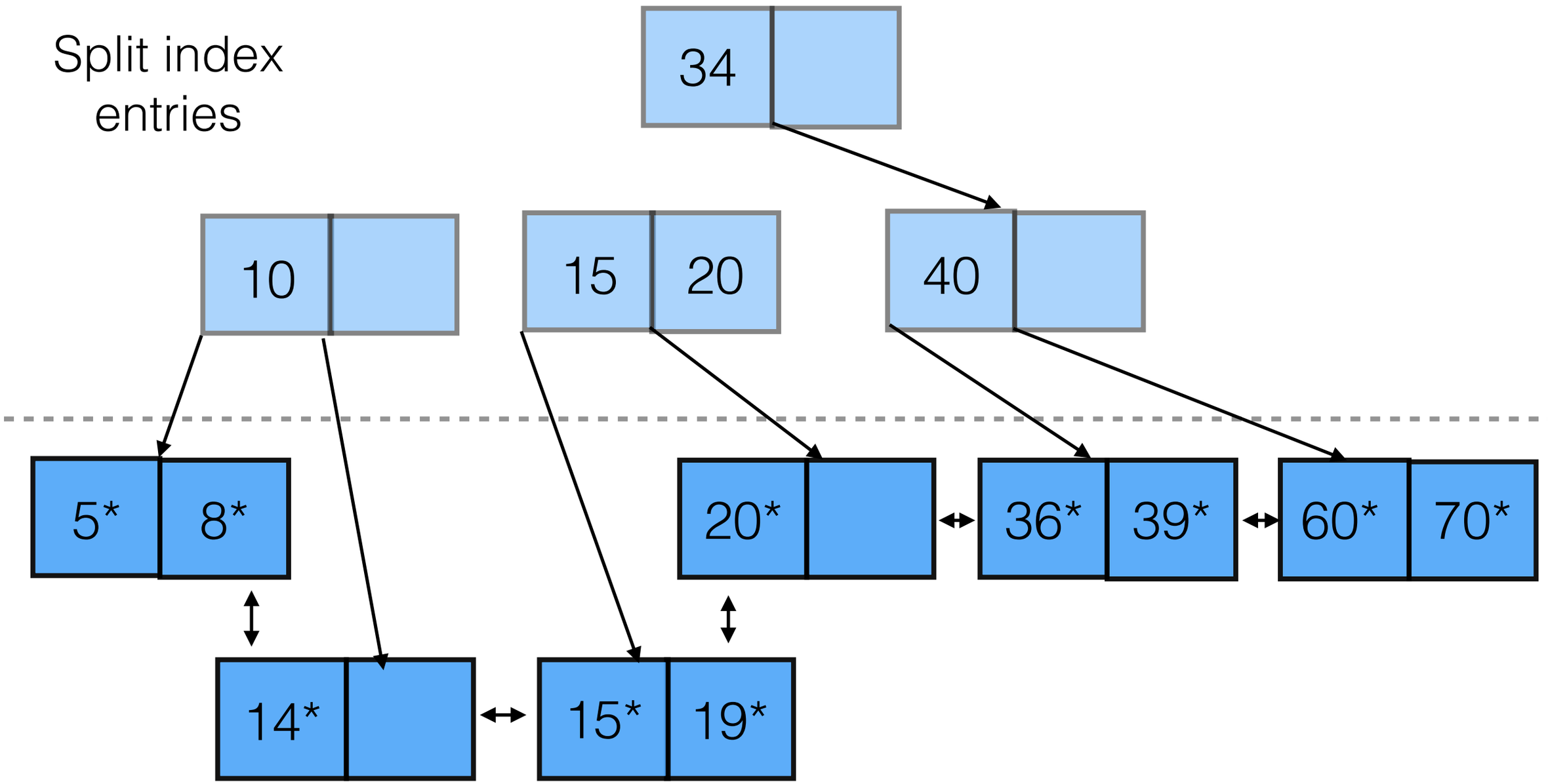
Split the leaf node

B+ Trees


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

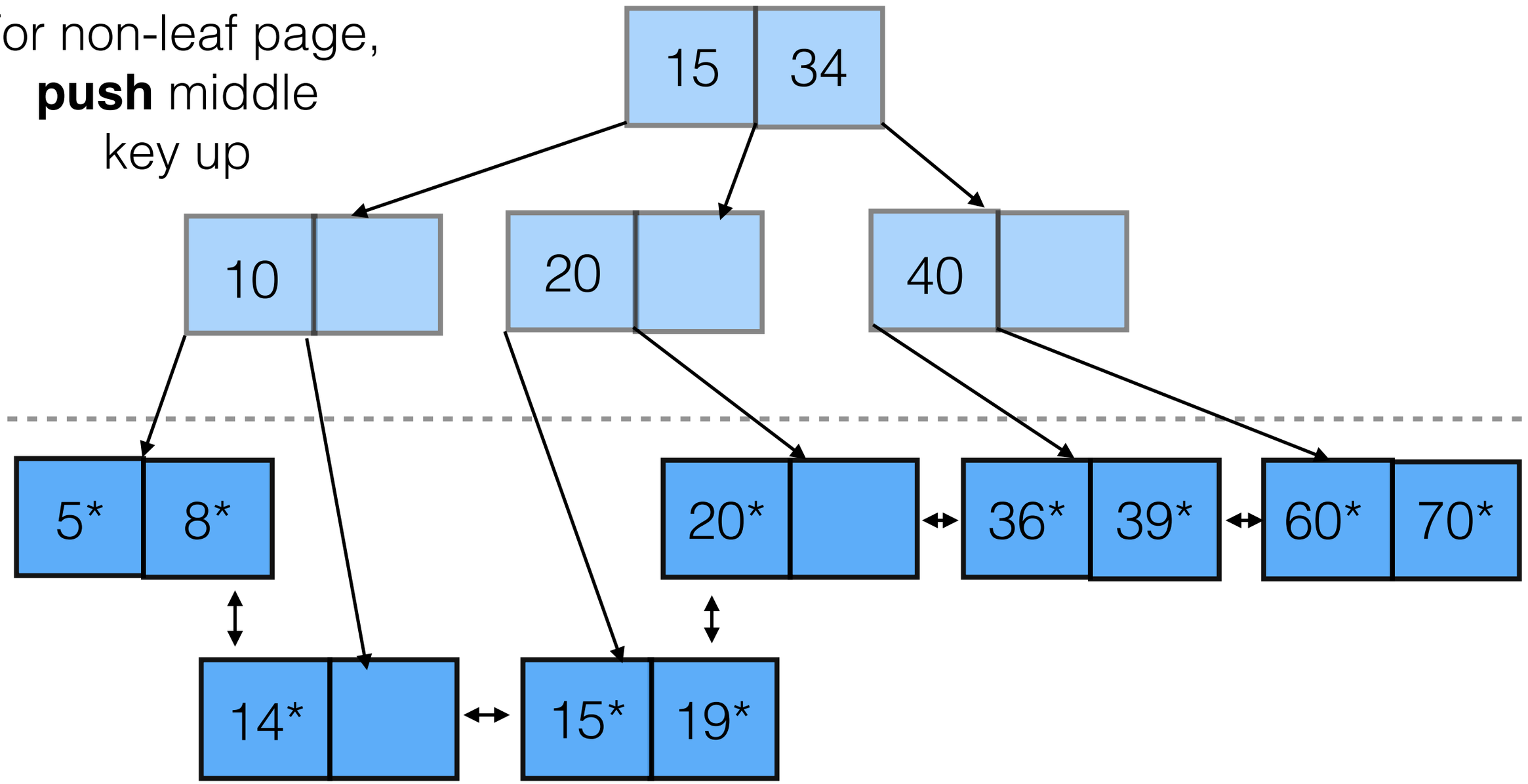
B+ Trees

Split index entries



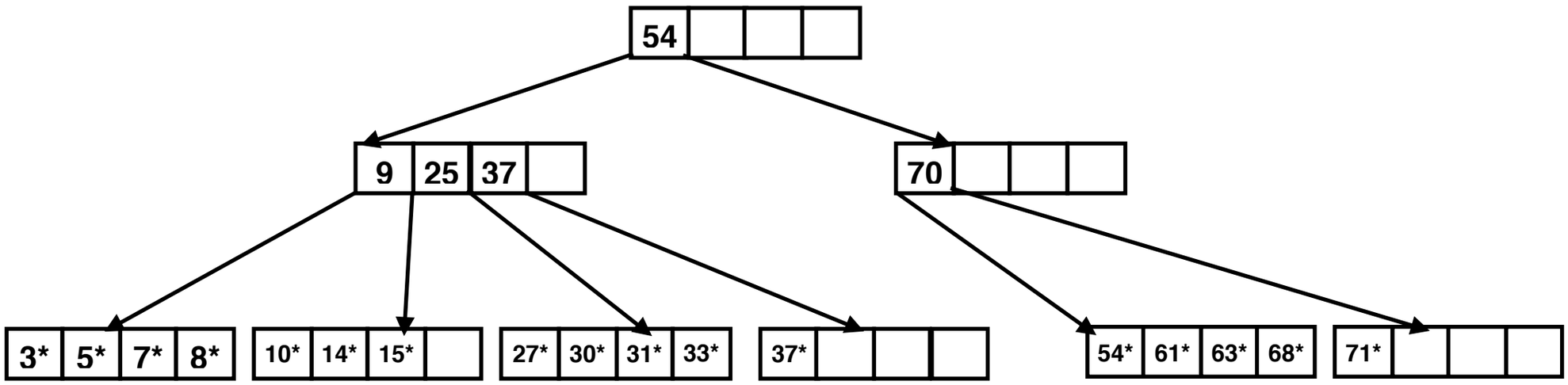
B+ Trees

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



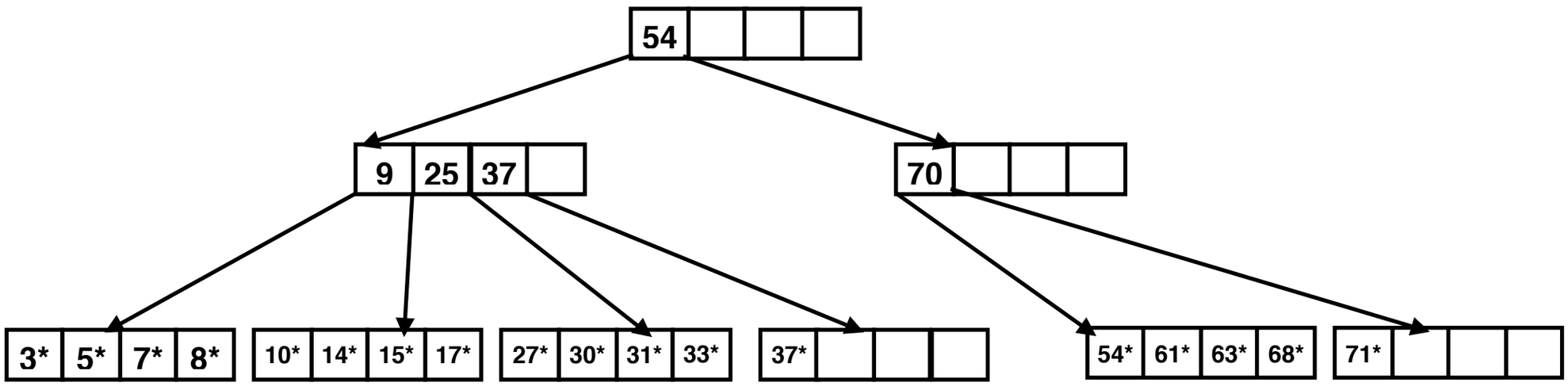
Worksheet

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



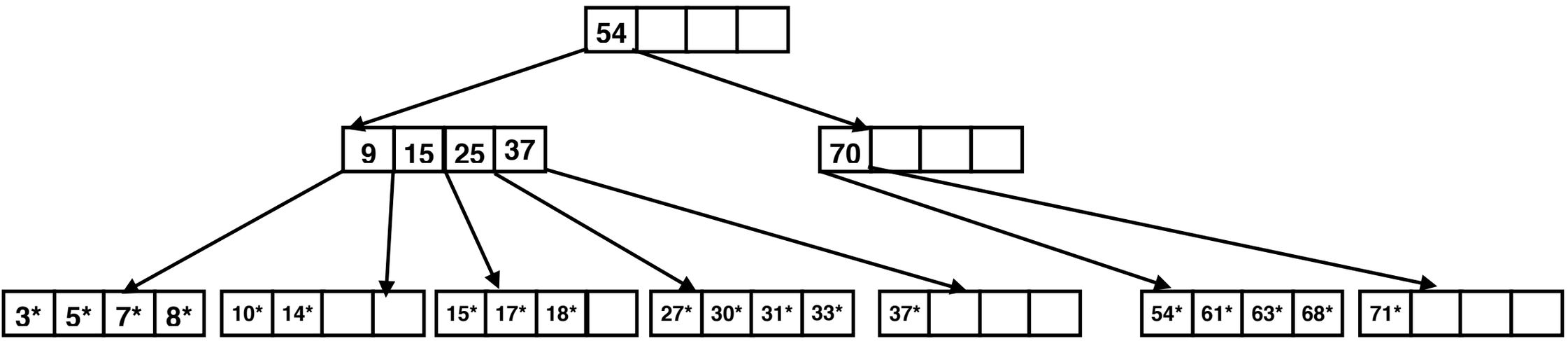
Insert 17

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



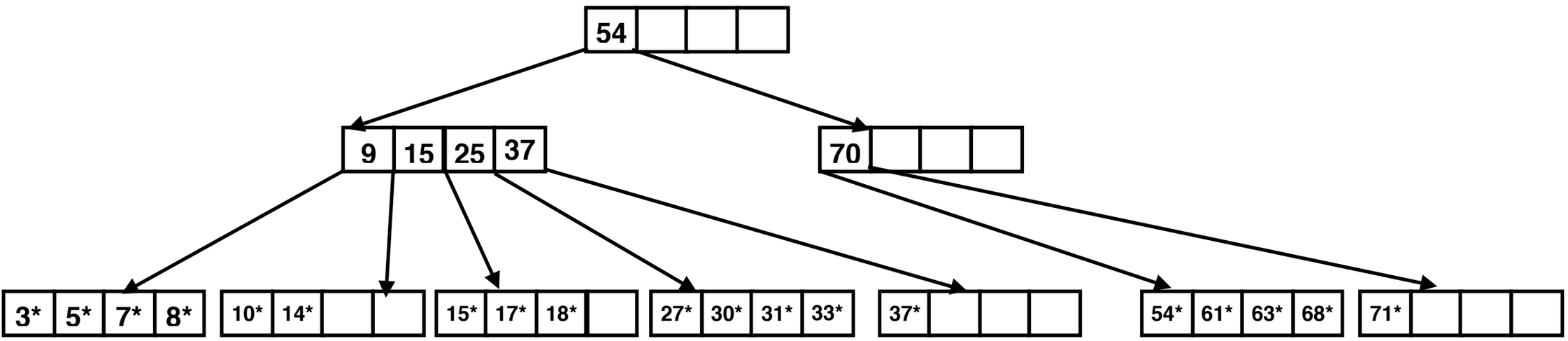
Insert 18

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



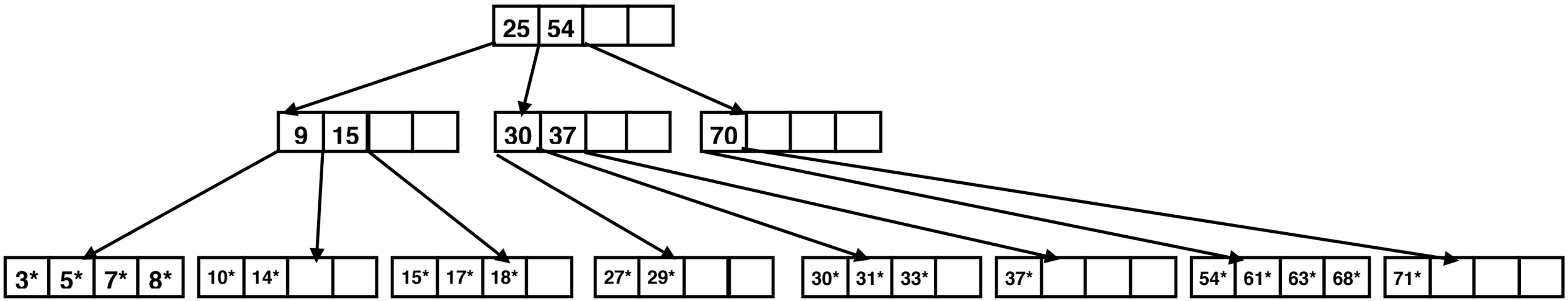
Insert 18

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

We are using a B+ tree with alternative 1 to store one billion records. Each record is 200 bytes, each disk page has 16kB (16,384 Bytes) and will always be at most 67% full.

How many leaf pages are required?

We are using a B+ tree with alternative 1 to store one billion records. Each record is 200 bytes, each disk page has 16kB (16,384 Bytes) and will always be at most 67% full.

How many leaf pages are required?

Records/page?

We are using a B+ tree with alternative 1 to store one billion records. Each record is 200 bytes, each disk page has 16kB (16,384 Bytes) and will always be at most 67% full.

How many leaf pages are required?

Records/page = 16384*.67/200 = ~54

We are using a B+ tree with alternative 1 to store one billion records. Each record is 200 bytes, each disk page has 16kB (16,384 Bytes) and will always be at most 67% full.

How many leaf pages are required?

Records/page = 16384*.67/200 = ~54

1^9/54 = ~18.5 * 10^6 pages.

We are using a B+ tree with alternative 1 to store one billion records. Each record is 200 bytes, each disk page has 16kB (16,384 Bytes) and will always be at most 67% full.

Assume each index entry takes 32 bytes. What is approximately the maximum fanout of the index?

We are using a B+ tree with alternative 1 to store one billion records. Each record is 200 bytes, each disk page has 16kB (16,384 Bytes) and will always be at most 67% full.

Assume each index entry takes 32 bytes. What is approximately the maximum fanout of the index?

F = index entries/page
= (bytes/page) / (bytes/index_entry)
(16384 * 0.67) / (32) = ~343

We are using a B+ tree with alternative 1 to store one billion records. Each record is 200 bytes, each disk page has 16kB (16,384 Bytes) and will always be at most 67% full.

What is the height (# levels of non-leaf nodes) of the tree?

We are using a B+ tree with alternative 1 to store one billion records. Each record is 200 bytes, each disk page has 16kB (16,384 Bytes) and will always be at most 67% full.

What is the height (# levels of non-leaf nodes) of the tree?

Height
= log_F(leaf pages)
= log_343(18.5 * 10^6) = ~3.

We are using a B+ tree with alternative 1 to store one billion records. Each record is 200 bytes, each disk page has 16kB (16,384 Bytes) and will always be at most 67% full.

How many I/O operations are required to insert a new record

We are using a B+ tree with alternative 1 to store one billion records. Each record is 200 bytes, each disk page has 16kB (16,384 Bytes) and will always be at most 67% full.

How many I/O operations are required to insert a new record

4 Reads (3 non-leaf reads, 1 leaf read) + 1 Write = 5 I/O’s.

We are using a B+ tree with alternative 1 to store one billion records. Each record is 200 bytes, each disk page has 16kB (16,384 Bytes) and will always be at most 67% full.

How many pages are required to store the non-leaf nodes?

We are using a B+ tree with alternative 1 to store one billion records. Each record is 200 bytes, each disk page has 16kB (16,384 Bytes) and will always be at most 67% full.

How many pages are required to store the non-leaf nodes?

1 + 343 + 343^2 = 117993

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?

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?

cost to traverse tree + matching pages

= height + (size matching user entries/page size)

= 3 + (50,000 * 2 / (16*.67))

= 9332

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 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/Os to descend non-leaf index pages

ceil(50,000 * 2/3 / (16*.67)) = **3110** I/Os to read leaf index pages

50,000 I/Os to read unordered data pages.

3 + 3110 + 50,000 = 53,113 I/Os.