Welcome to CS 186, Section 5!

TA: Bryan Munar
OH: Mondays 11-12pm and Thursdays 2:30-3:30pm (651 Soda)
DISC: Tuesdays 11-12am (136 Barrows) and Wednesdays 10-11am (130 Wheeler)

Announcements and Such

- Midterm 1 next week (Oct. 5!!)

- ONE cheatsheet!!

- Project/HW 3 out soon (probably after midterm)

Discussion 5: Tree-Structured Indexes!

Overview:

1. Indexes
2. Worksheet exercises
3. Tree-Structured Indexes
4. Worksheet exercises

(A majority of the slides are from Michelle and lecture!)

Indexes

IMPORTANT

What’s an index? Why is it important?

Indexes

IMPORTANT

Allow record retrieval *by value* in ≥1 field:

- Find all students in the "CS" department
- Find students with a gpa > 3
- Find students with firstname "Bob", lastname "Nob"

Index : disk-based data structure for fast lookup by value

- *Search key:* any subset of columns in the relation.
  - *Search key* need not be a *key* of the relation
  - I.e. There can be multiple items matching a search key

Index contains a collection of *data entries*

- (***k***, {items})
- Items associated with each search key value ***k***
- Data entries come in various forms, as we'll see

Alternatives for Data Entry **k*** in Index

Three alternatives:

1. Actual data record (with key value **k**)
2. <**k**, rid of matching data record>
3. <**k**, list of rids of matching data records>

IMPORTANT

Choice is orthogonal to the indexing technique.

B+ trees, hash-based structures, R trees, GiSTs, …

Can have multiple (different) indexes per file.

E.g. file sorted by *age*, with a hash index on *salary* and a B+tree index on *name*.

Alternatives for Data Entries (Contd.)

Alternative 1:

Actual data record (with key value ***k***)

- Index as a file organization for records
  - Alongside Heap files or sorted files
- At most one Alt. 1 index per relation
- No “pointer lookups” to get data records

Alternatives for Data Entries (Contd.)

Alternative 2

<**k**, rid of matching data record>

Alternative 3

<**k**, list of rids of matching data records>

- Alts. 2 or 3 *required* to support multiple indexes per relation!
- Alt. 3 more compact than Alt. 2,

  … but *variable sized data entries*
  - even if search keys are of fixed length.
- For large rid lists, data entry spans multiple blocks (!)

Clustered vs. Unclustered Index

In a clustered index:

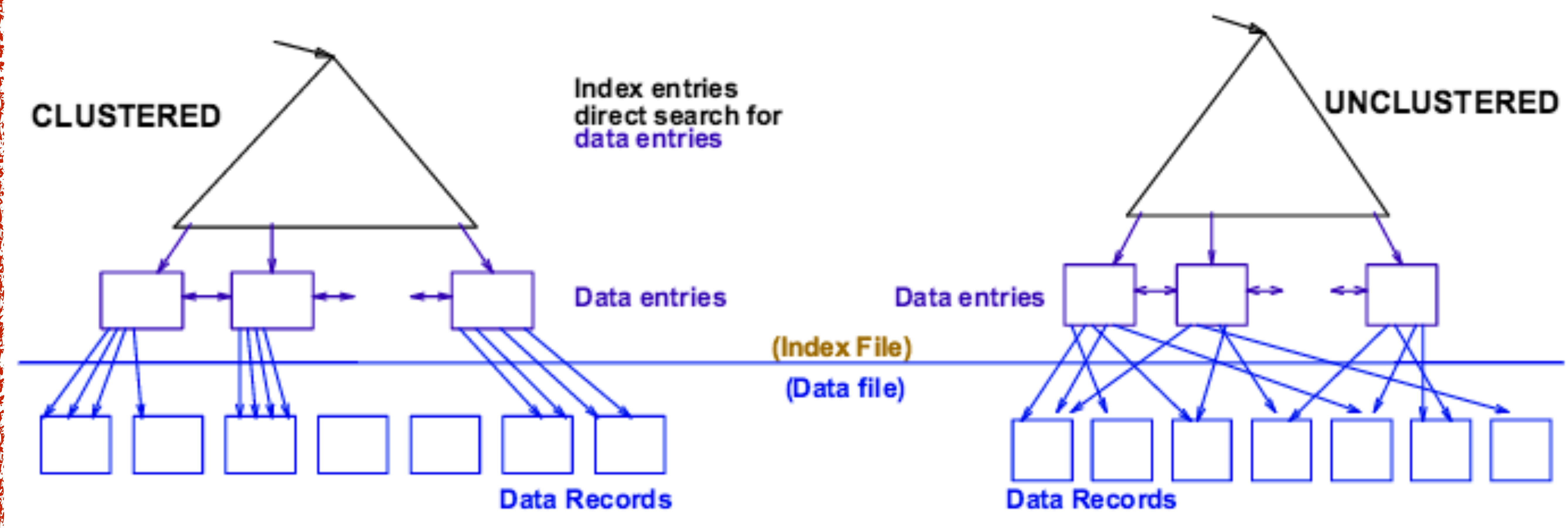
- index data entries are stored in (approximate) order by value of search key in data records
- A file can be clustered *on at most one* search key.
- Cost of retrieving data records through index varies *greatly* based on whether index is clustered or not!
- Alternative 1 => clustered
    - but not vice-versa!

Note: there is another definition of “clustering”

- Data Mining/AI: grouping similar items in n-space

Clustered vs. Unclustered Index

- Alternative (2) data entries, data records in a Heap file.
  - To build clustered index, first sort the Heap file
    - with some free space on each block for future inserts
  - Overflow blocks may be needed for inserts.
    - Thus, order of data recs is `close to', but not identical to, the sort order.



Cost of Operations

B: The number of data pages
R: Number of records per page
D: (Average) time to read or write disk page

	Heap File	Sorted File	Clustered File
Scan all records	BD	BD	1.5 BD
Equality Search	½BD (primary, if found)	$(\log_2 B) * D$	$(\log_F 1.5B+1) * D$
Range Search	BD	$[(\log_2 B) +$ #match pages$]*D$	$[(\log_F 1.5B) +$ *#match pages*$]*D$
Insert	D+D (read/write)	$((\log_2 B)+½B+½B)D$ (read/write half)	$((\log_F 1.5B)+2) * D$
Delete	½BD + D (primary, if found)	$((\log_2 B)+½B+½B)D$ (read/write half)	$((\log_F 1.5B)+2) * D$

Unclustered vs. Clustered Indexes

IMPORTANT

- Clustered Pros
  - Efficient for range searches
  - Potential locality benefits
    - Disk scheduling, prefetching, etc.
  - Support certain types of compression
    - More soon on this topic
- Clustered Cons
  - More expensive to maintain
    - on the fly or "sloppily" via reorgs
    - Heap file usually only packed to 2/3 to accommodate inserts

Hash based index:

Cannot do range queries. Slightly faster on equality search.

Tree based index:

Unclustered: Fast equality search, insertion, deletion.

Clustered: Fast search, insertion, deletion, scan, range search. Needs space.

Do first 2 pages of worksheet!

Tree-Structured Indexes

IMPORTANT

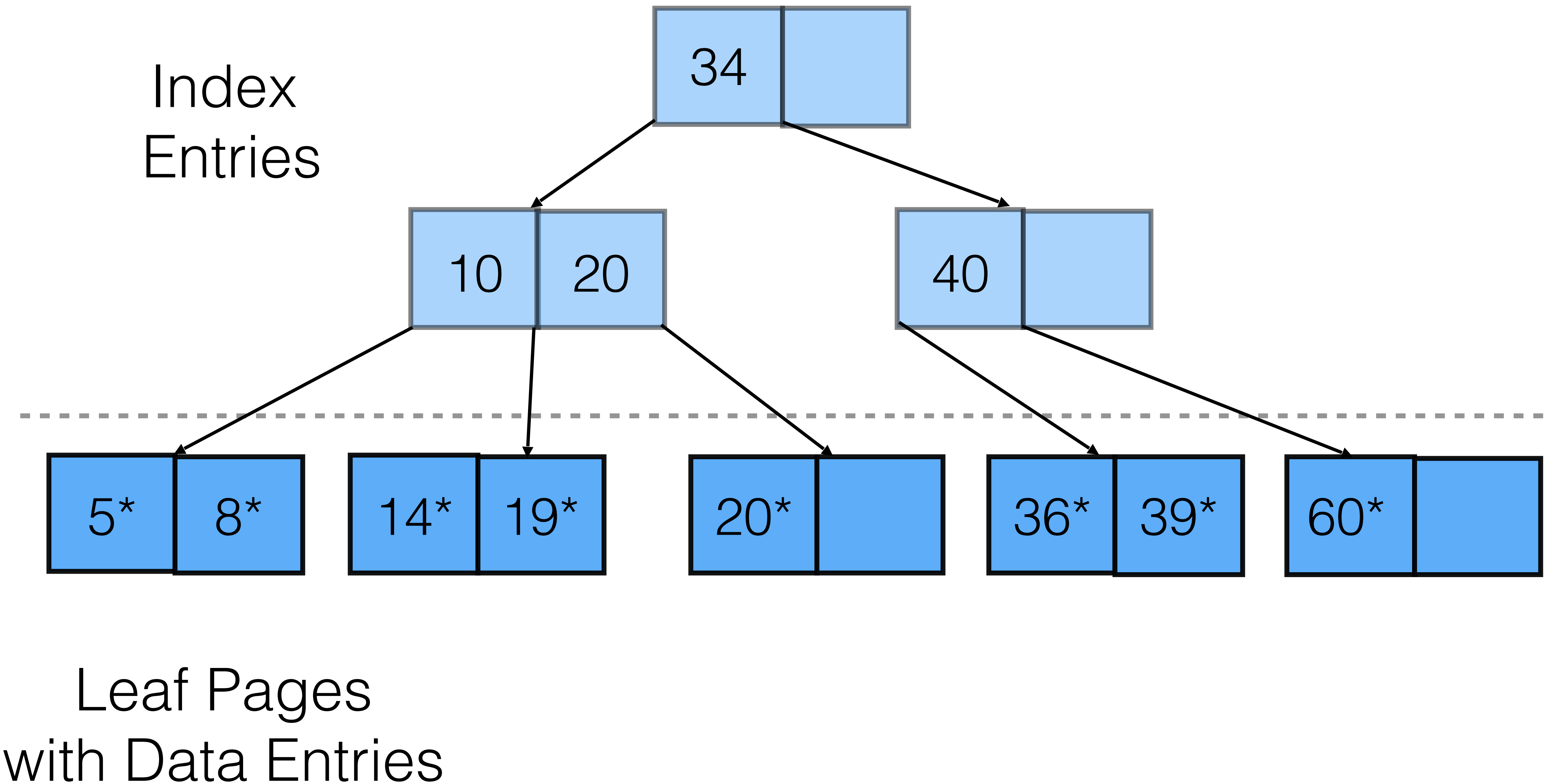
Terminology

- Fanout (F): number of entries per page
- Order (d): minimum number of entries per node
- N: number of leaf pages
- leaf page: page that stores file data (or reference to file data)
- non-leaf page: index entries in the tree

ISAM

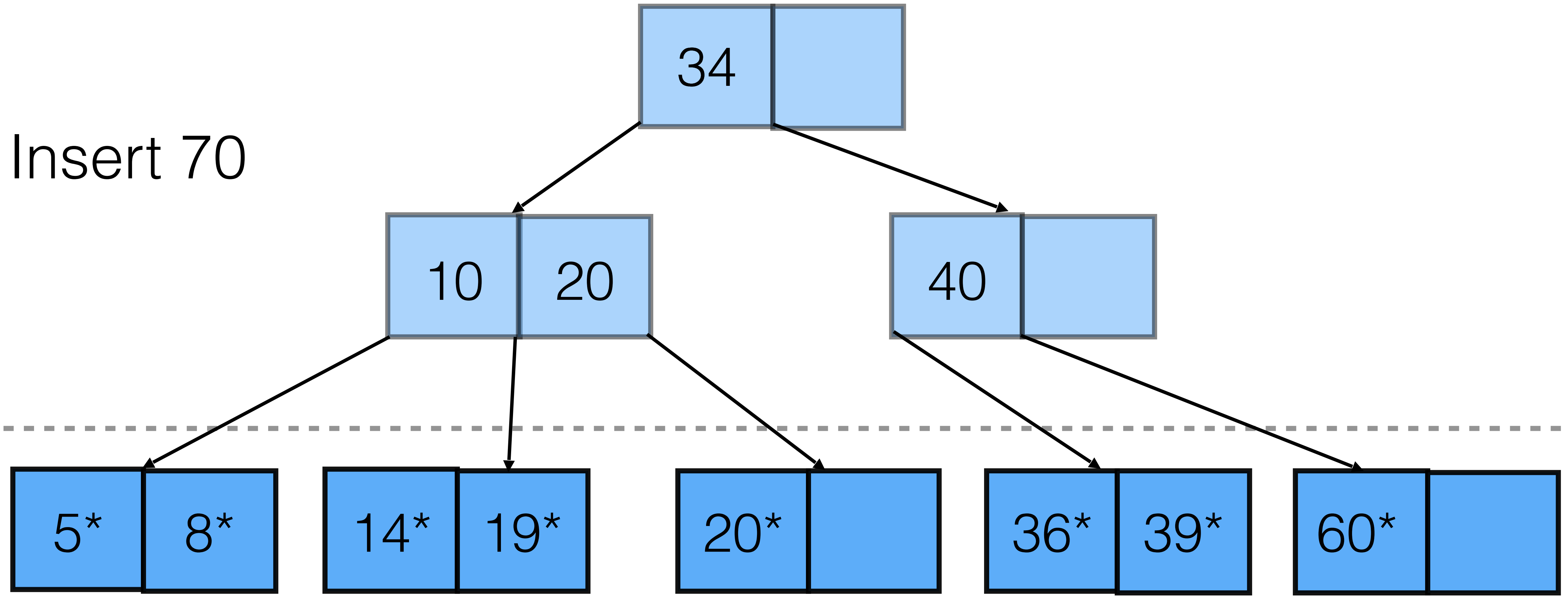
- Simple, static structure
- Created by:
  - Sorting records by index search key (e.g. “gpa”)
  - Building a tree on top of those records

ISAM


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

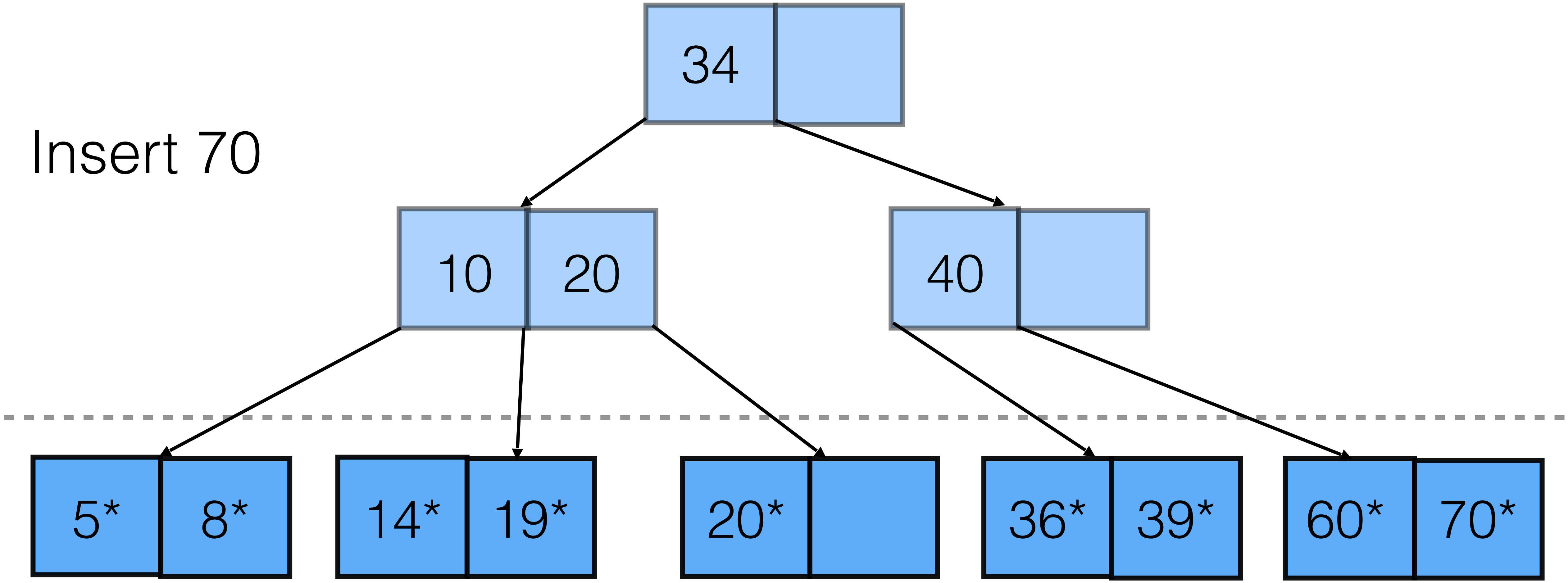
ISAM

Insert 70



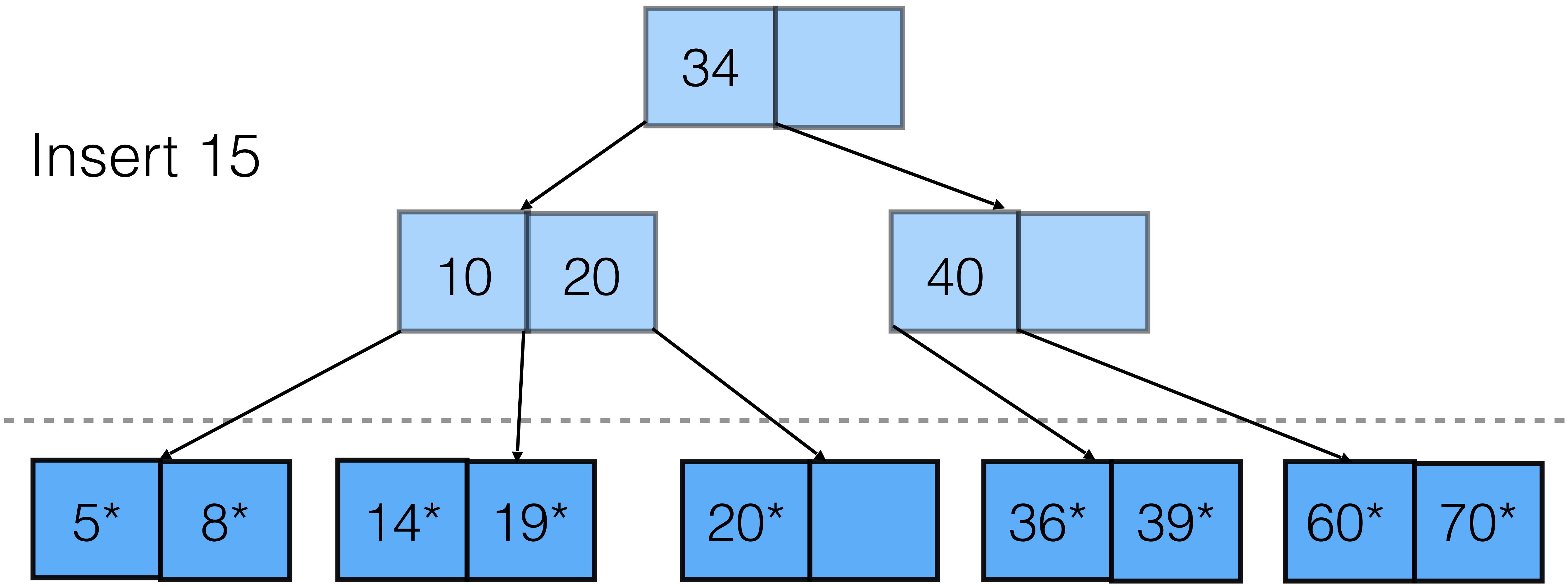
ISAM

Insert 70



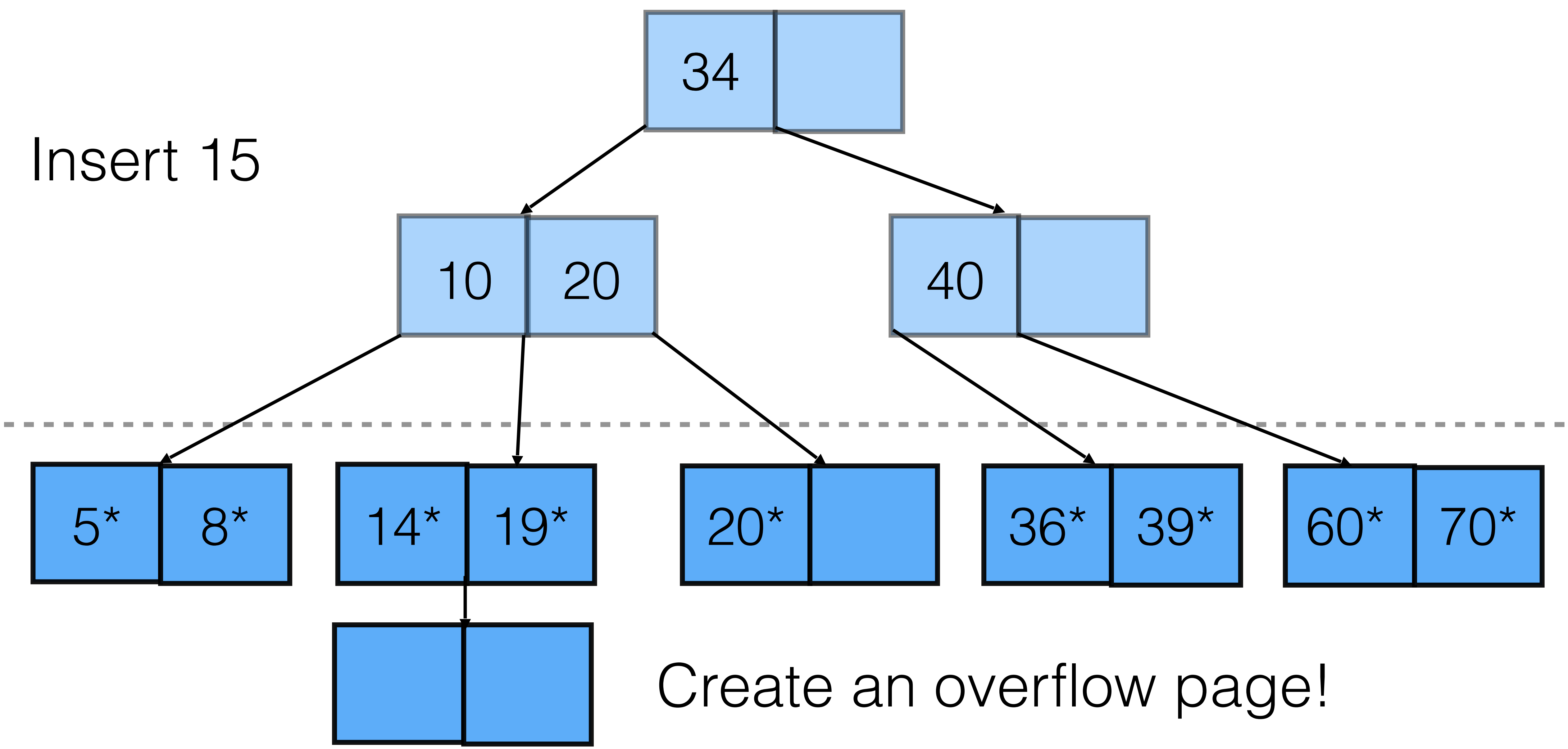
ISAM

Insert 15



ISAM

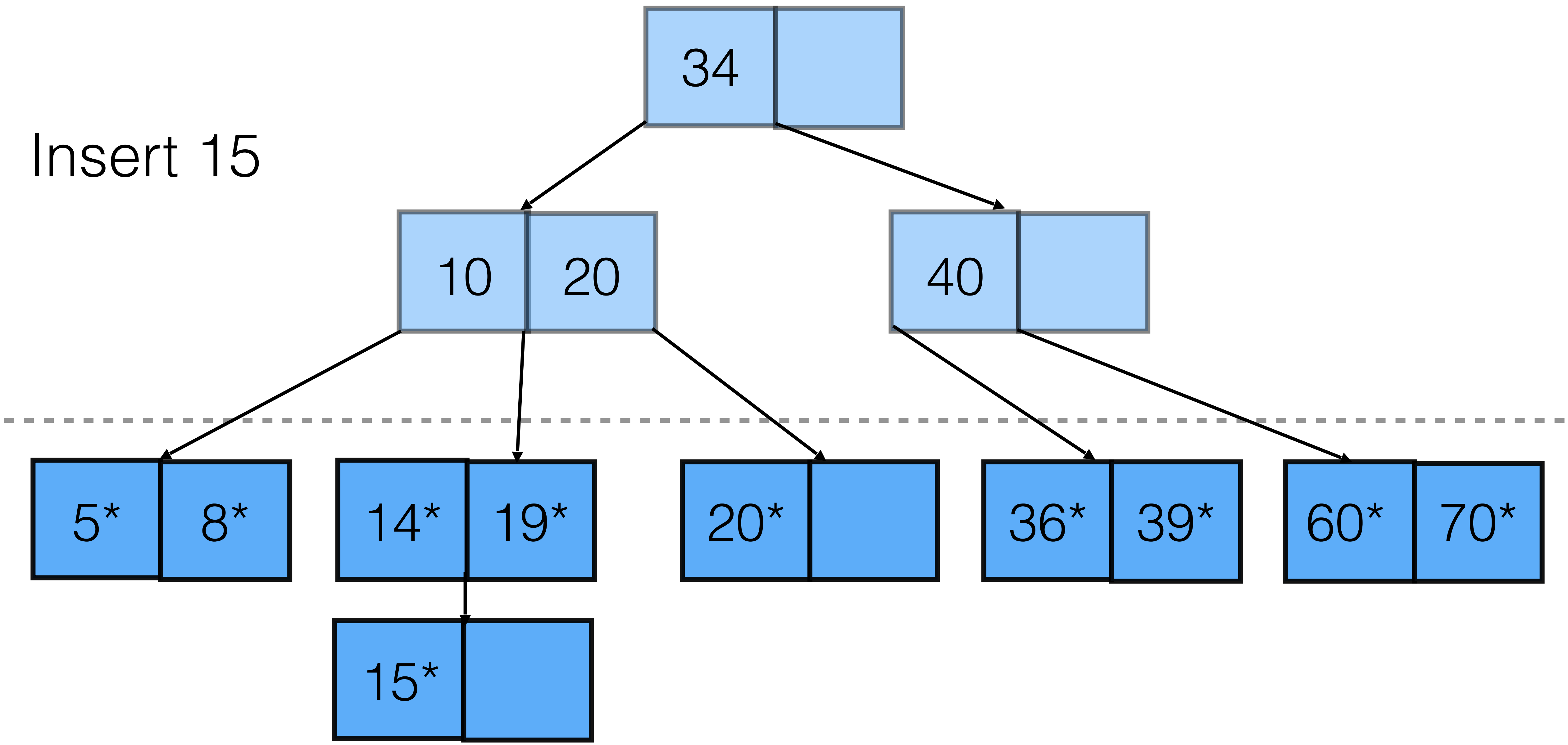
Insert 15



Create an overflow page!

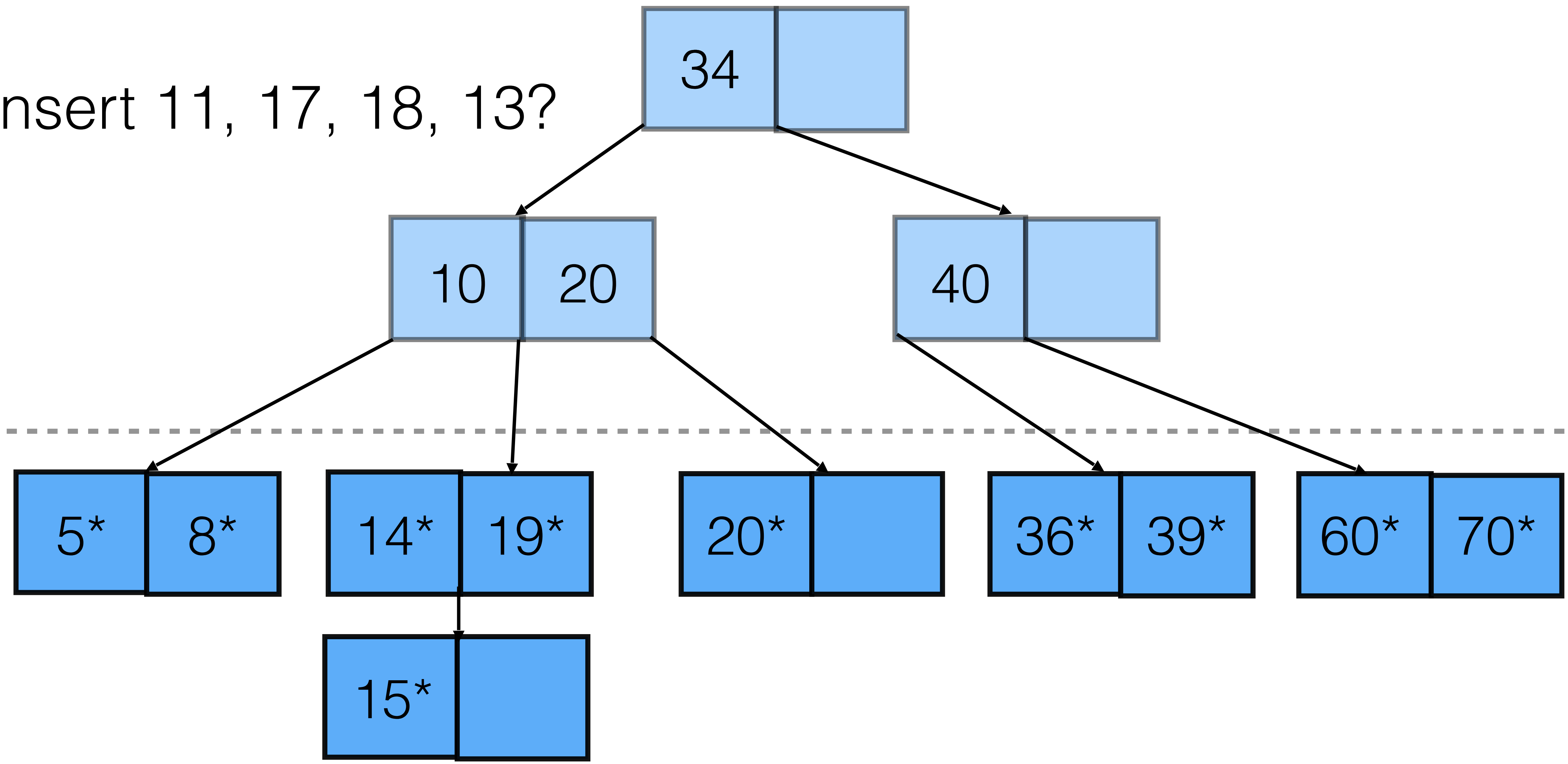
ISAM

Insert 15



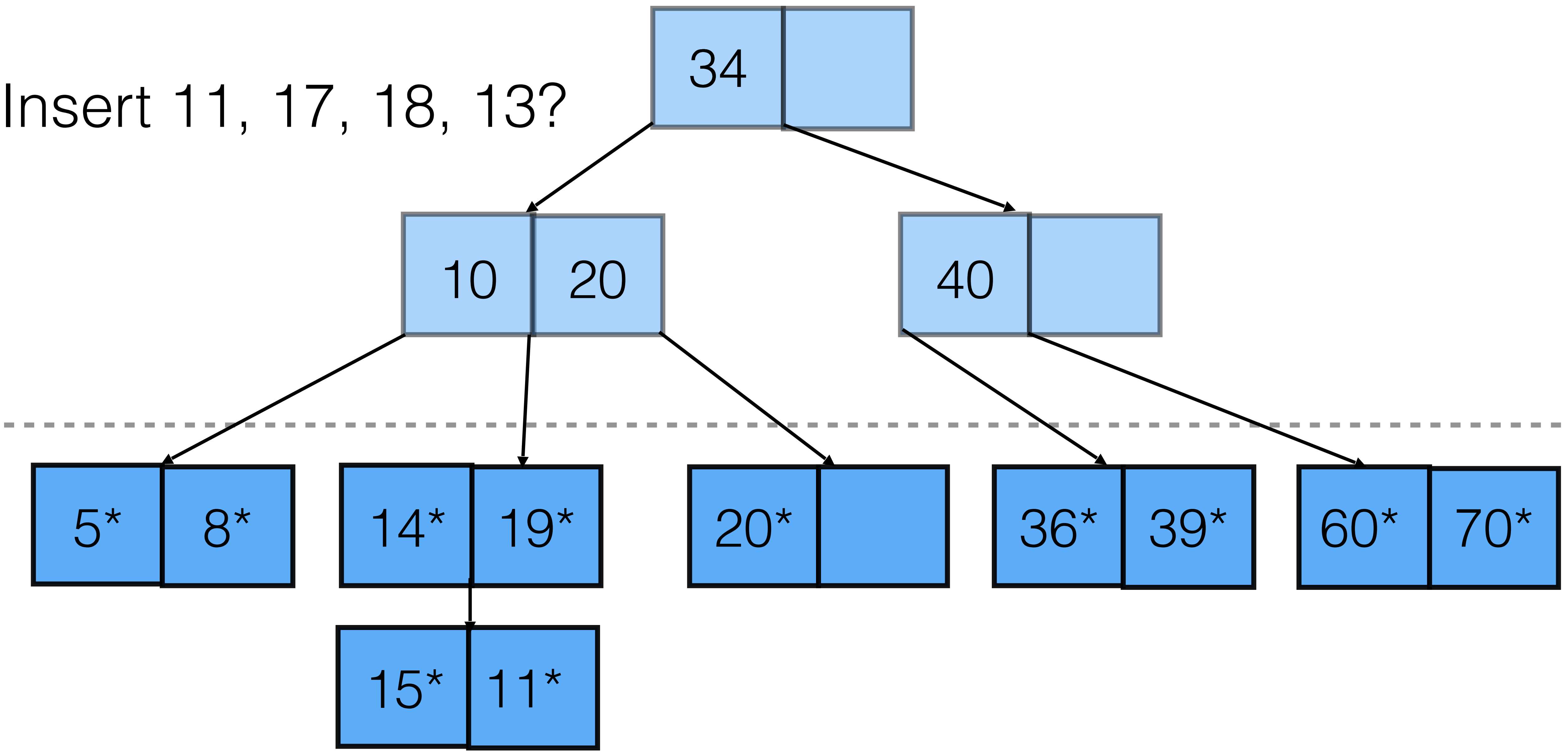
ISAM

Insert 11, 17, 18, 13?



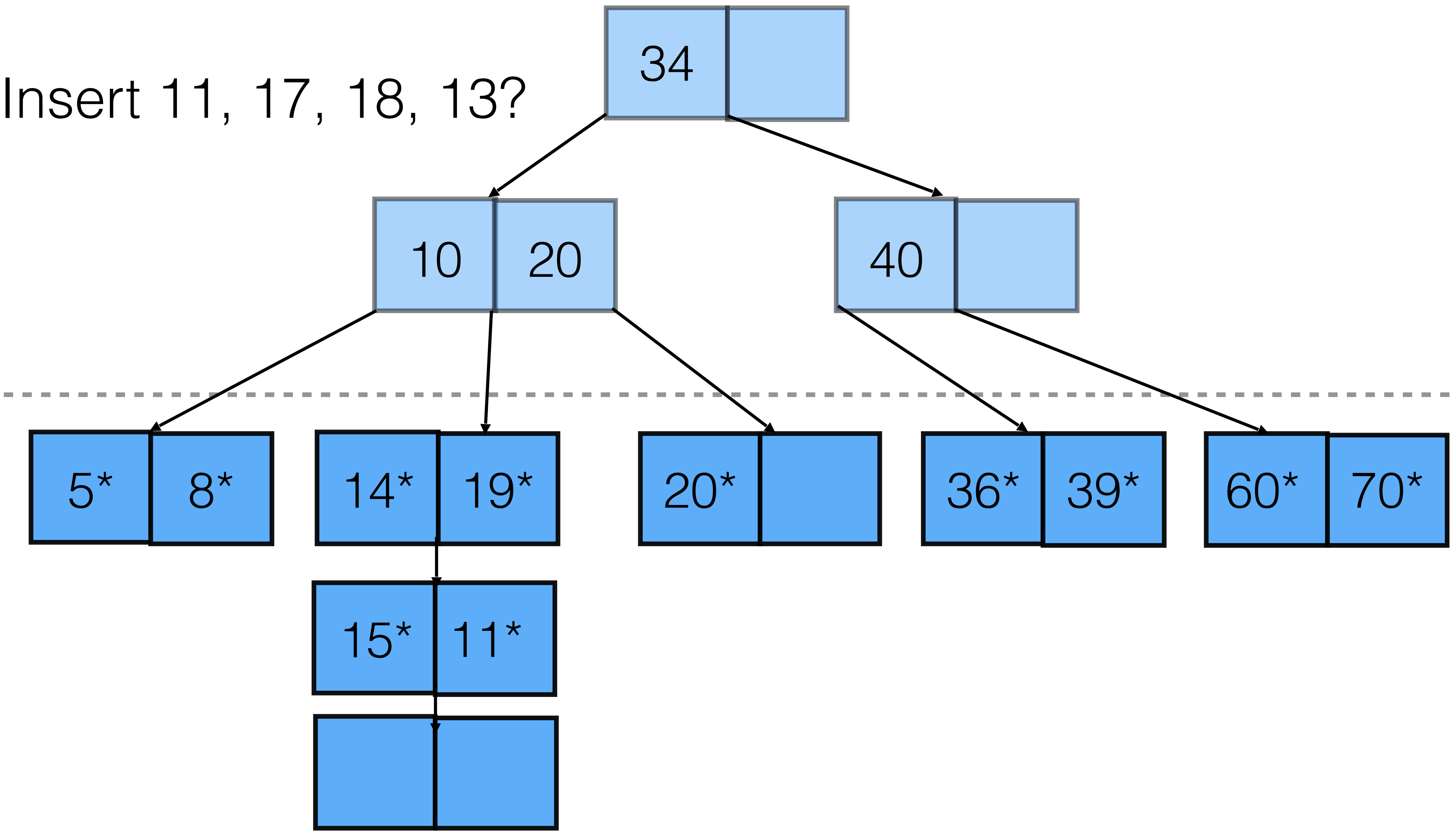
ISAM

Insert 11, 17, 18, 13?



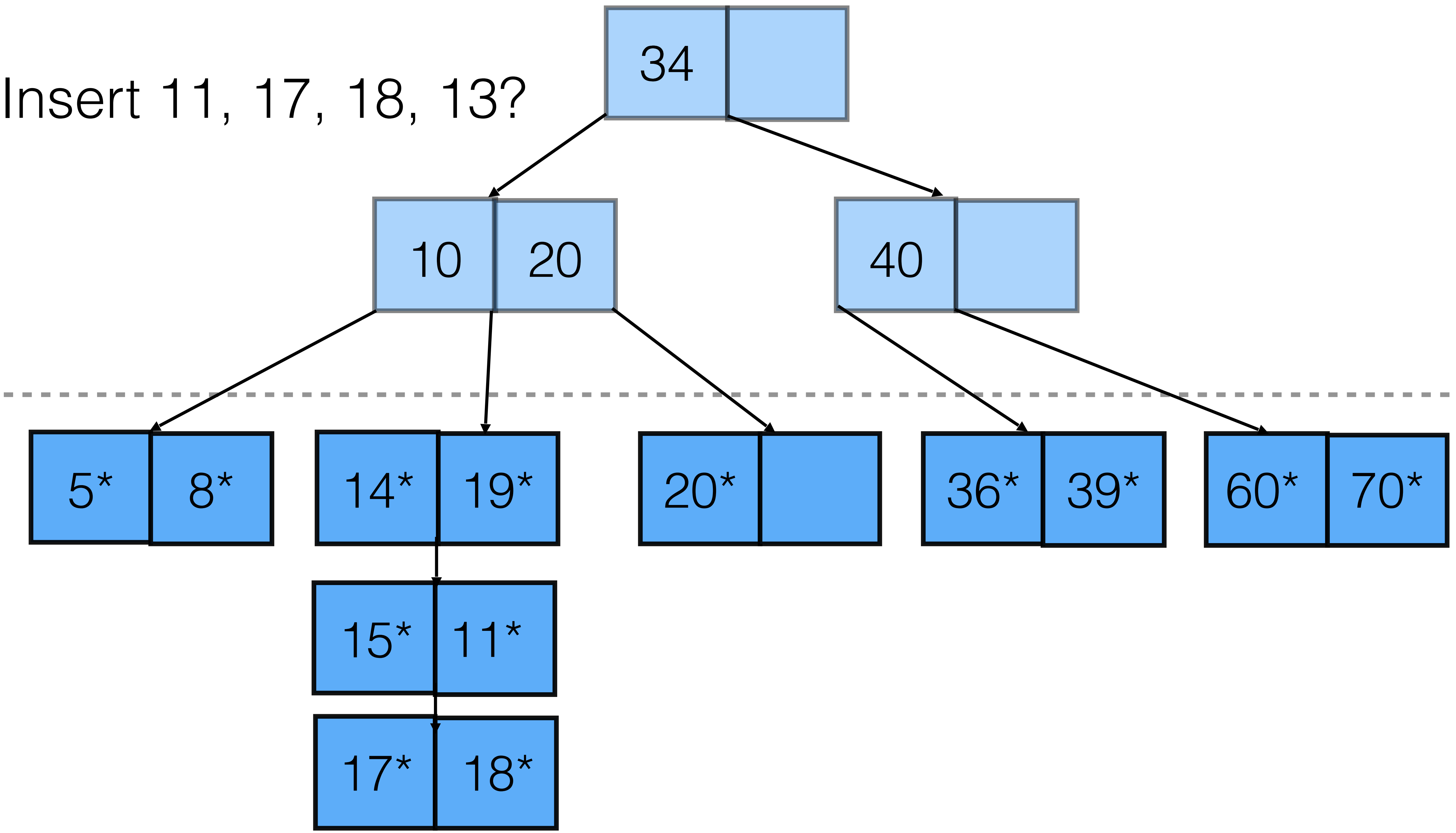
ISAM

Insert 11, 17, 18, 13?



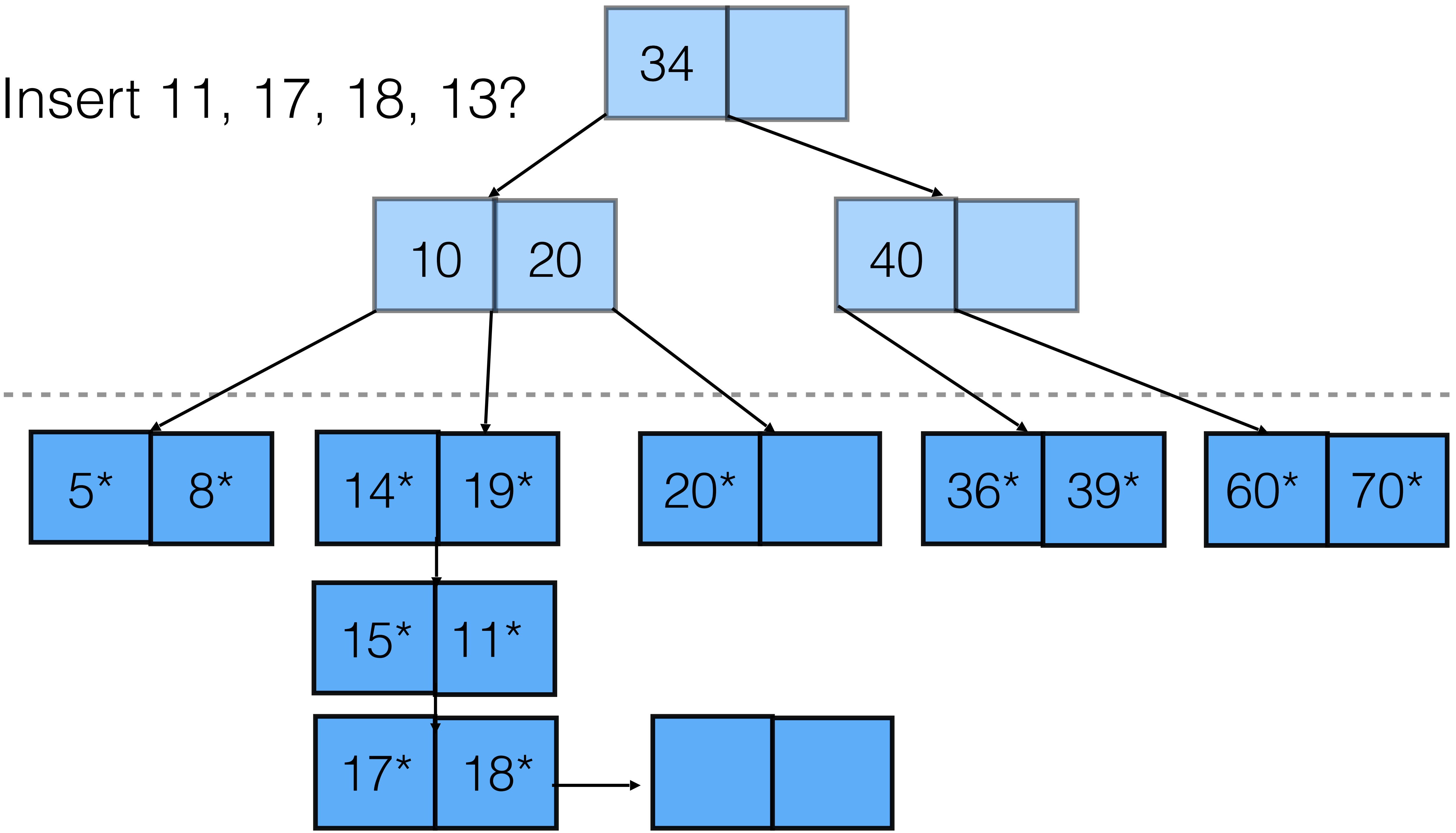
ISAM

Insert 11, 17, 18, 13?



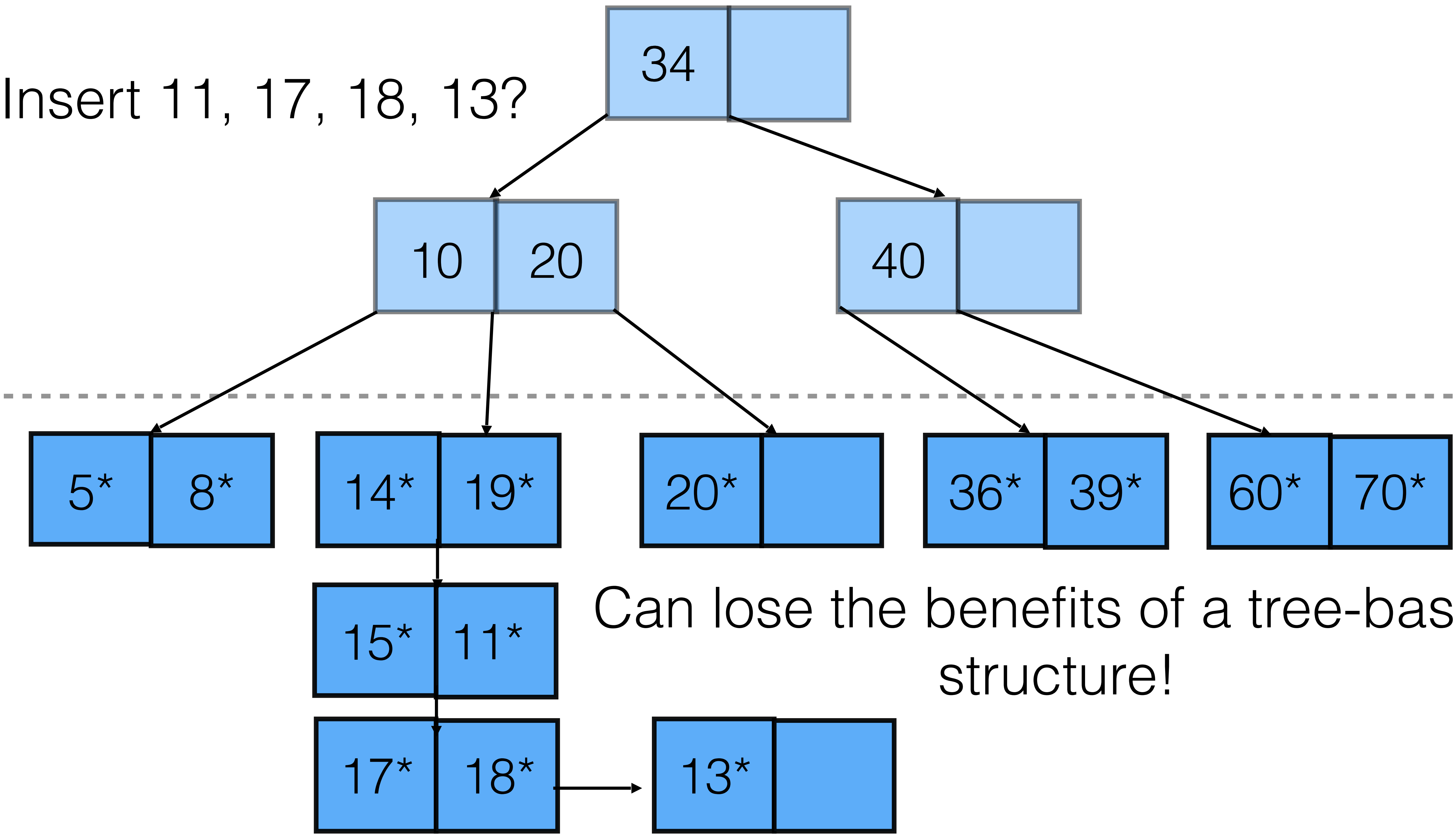
ISAM

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

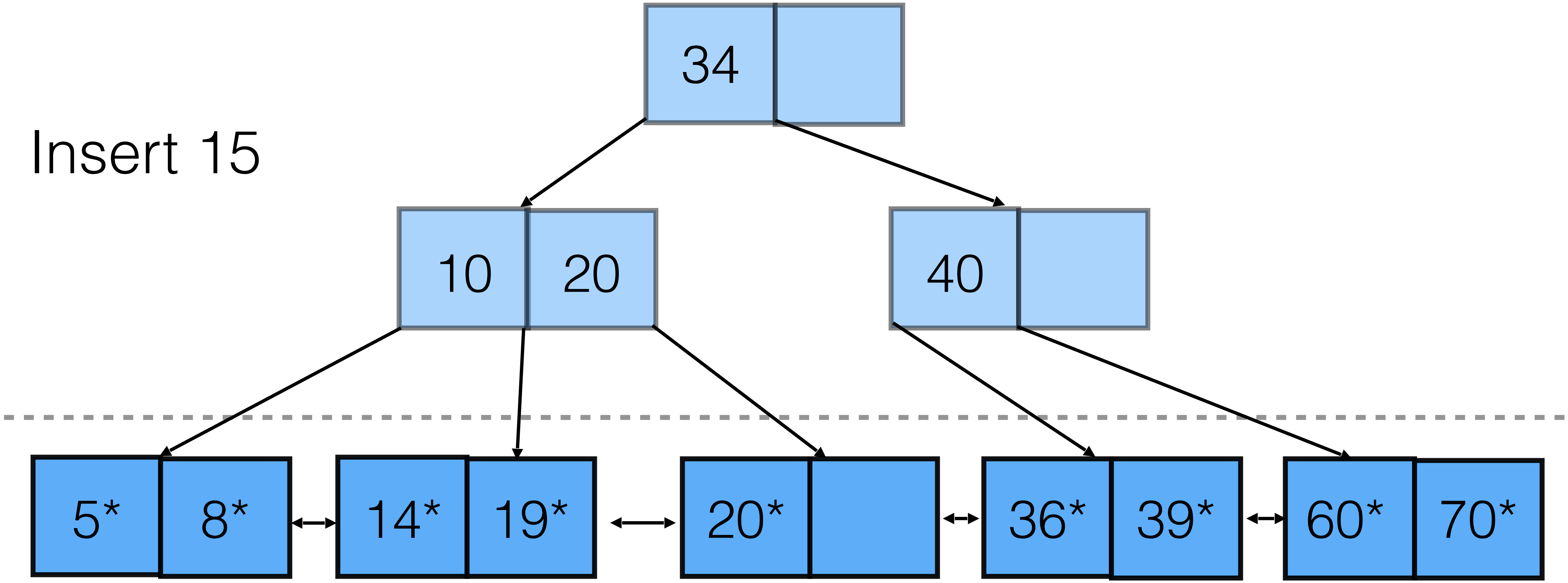
- If L has space:
  - Insert X in that page
- Else:
  - If an overflow page has space, insert X in that page
  - Else, create a new overflow page and insert X

B+ Trees

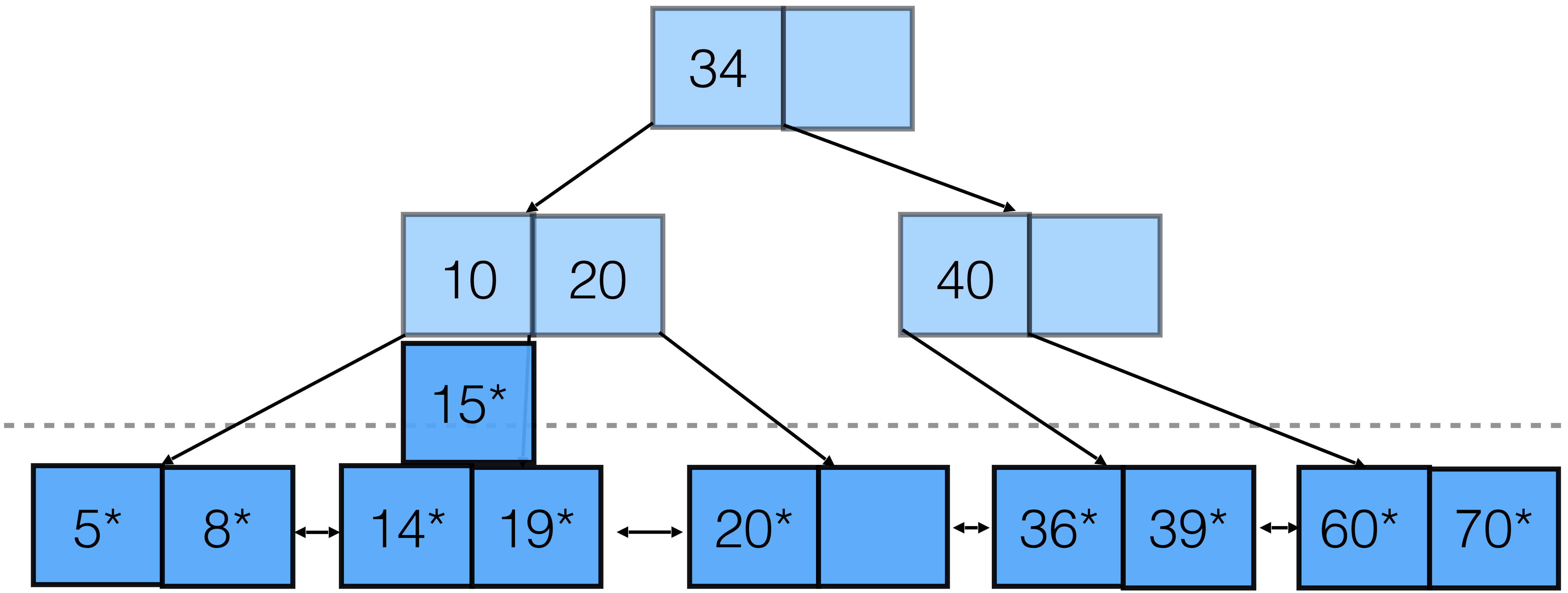
- Dynamic structure to keep tree height-balanced
- Adjusts under inserts and deletes
- Maintain minimum 50% occupancy for each page (except root)

B+ Trees

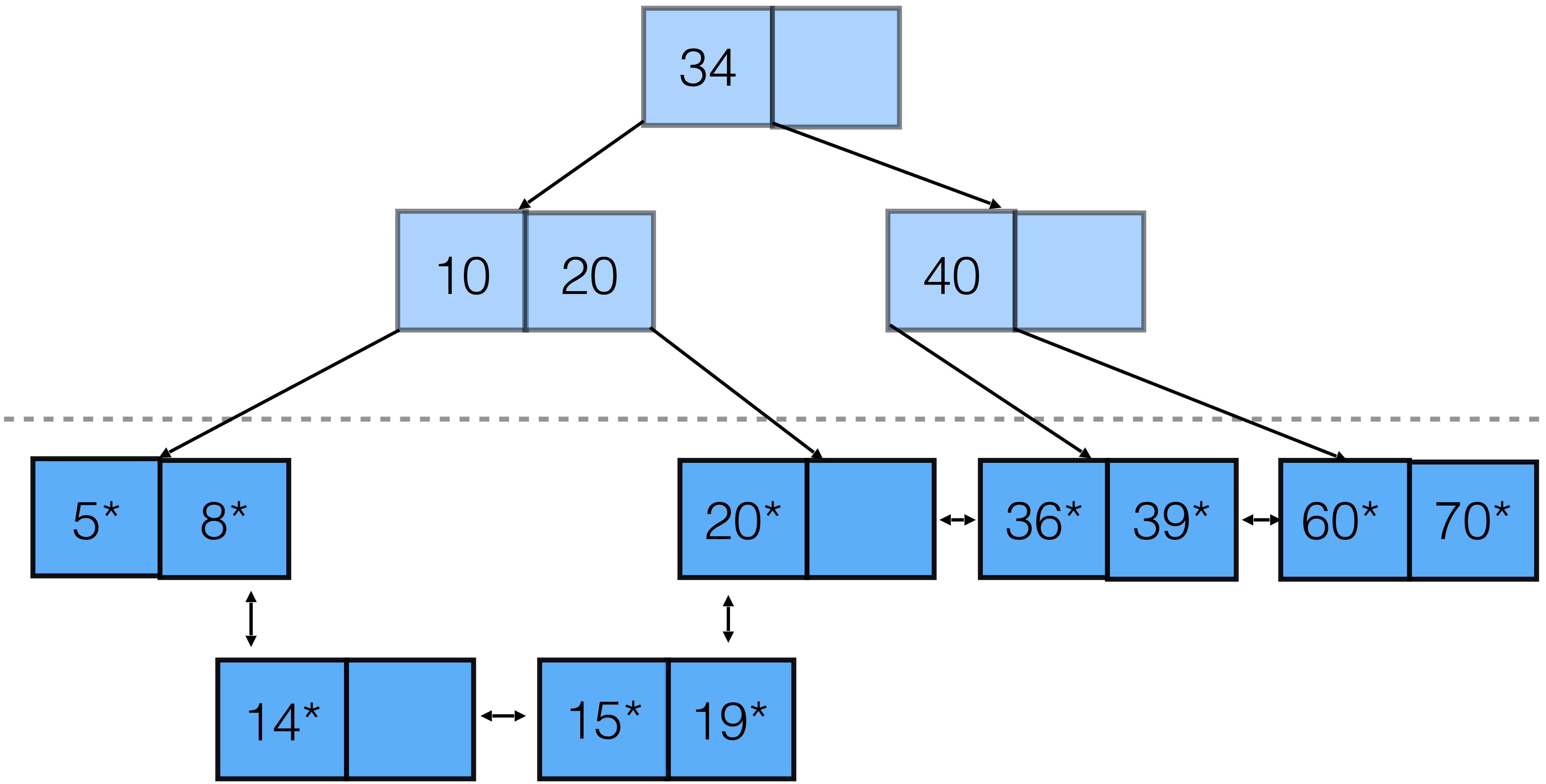
Insert 15



B+ Trees

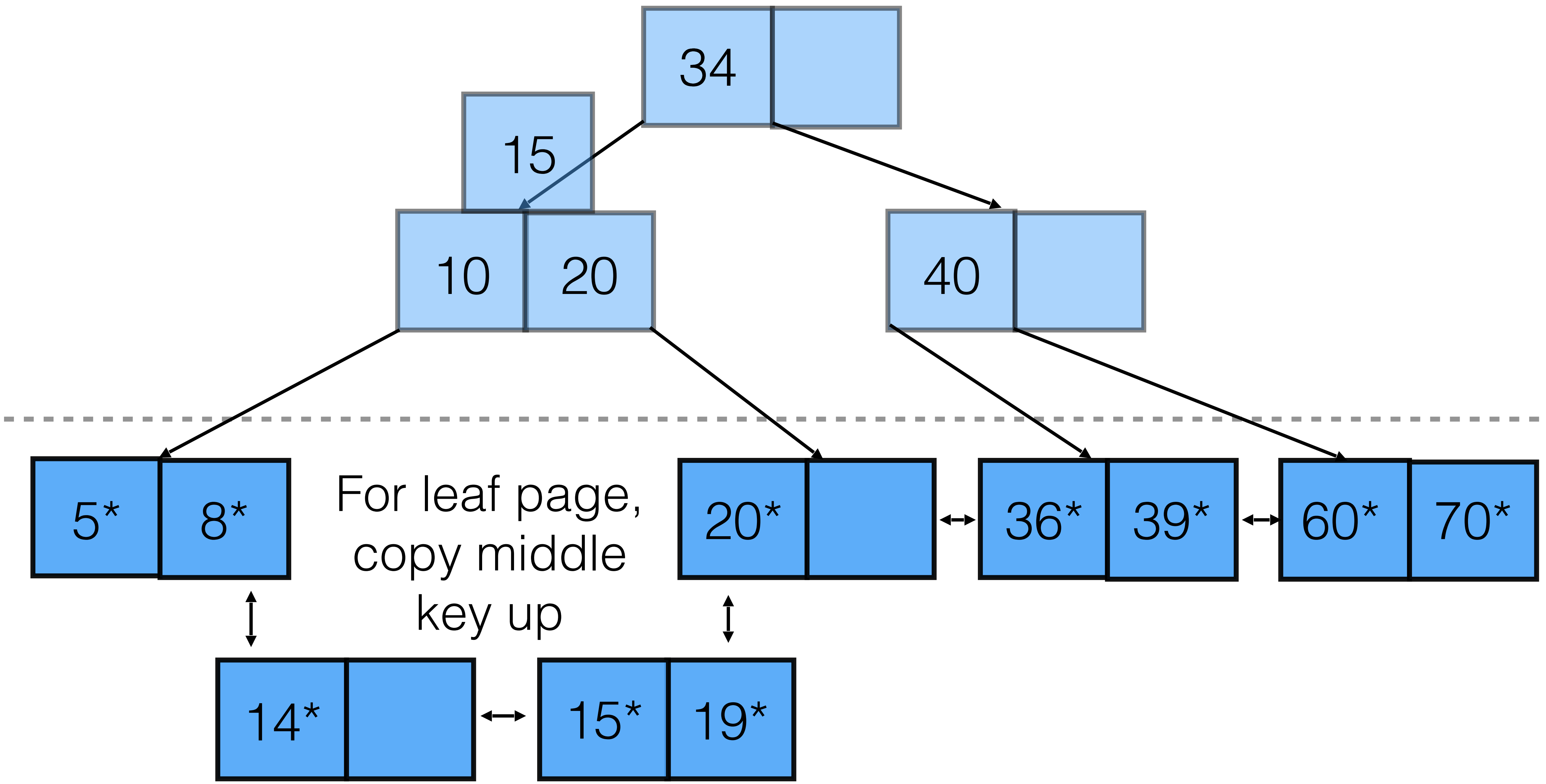


B+ Trees



Split the leaf node

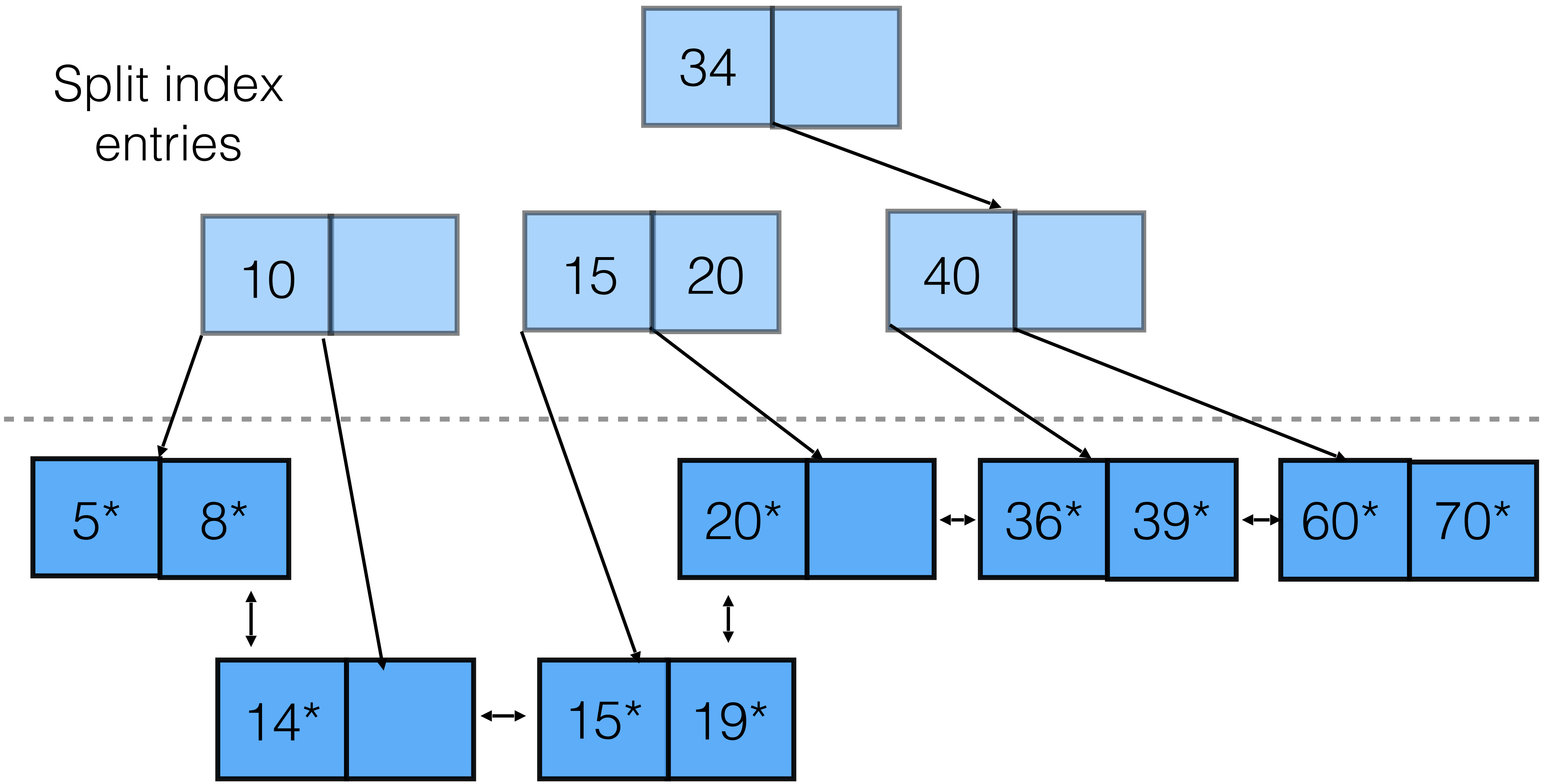
B+ Trees



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

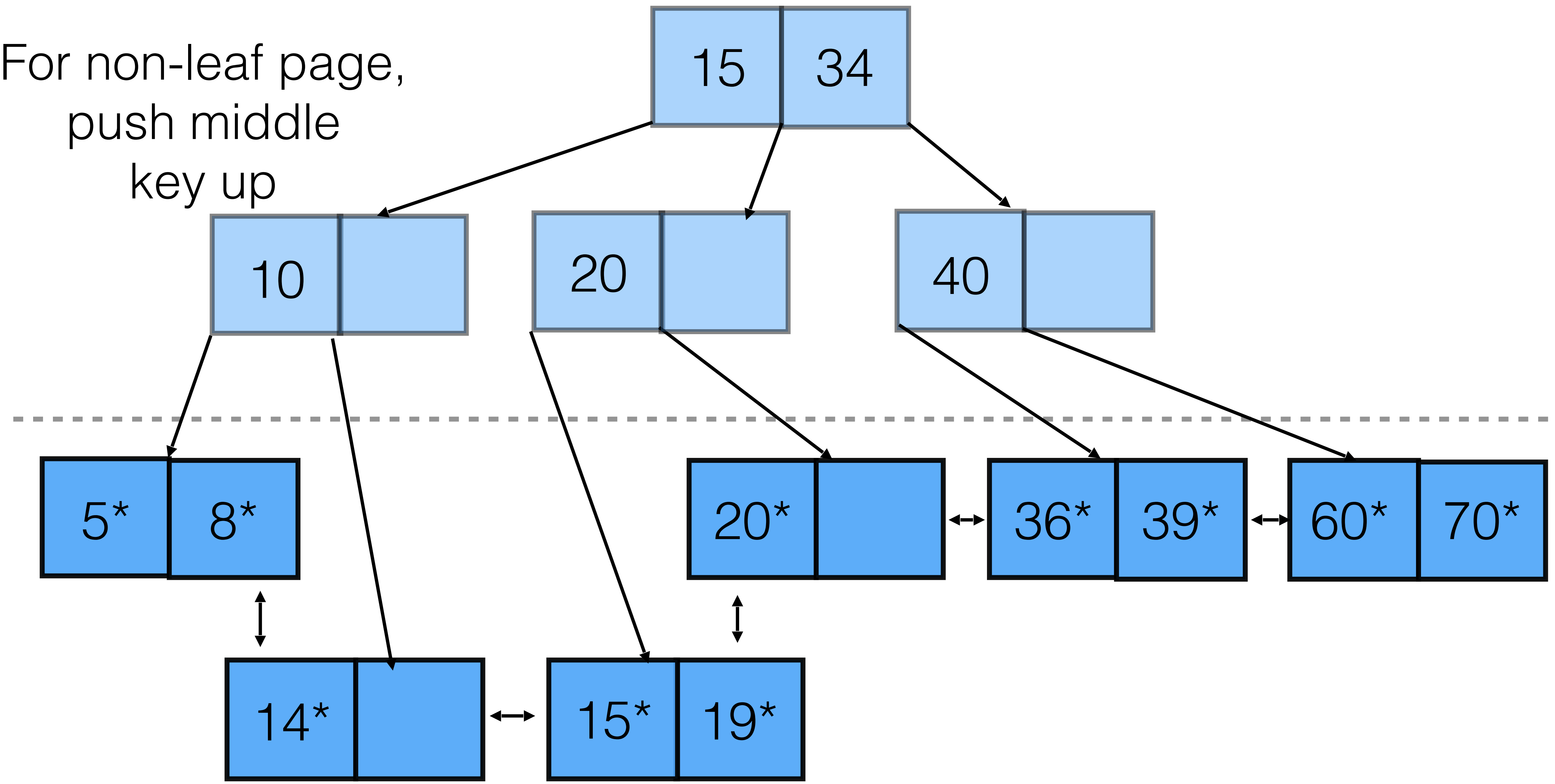
B+ Trees

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

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.

Do last 2 pages of worksheet!