Discussion 8

Tree index & Midterm Exam Solutions
EECS 484

Logistics

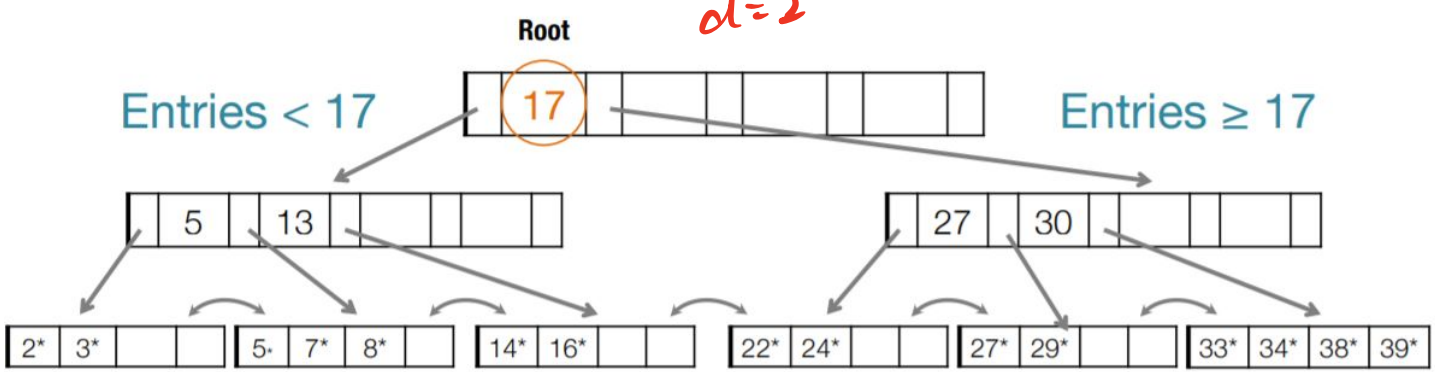
- Homework 4 due this Thursday (Nov 3) at 11:55 PM EST
- Project 3 due Nov 10 at 11:55 PM EST
- Midterm exam grade released
  - Solutions on canvas
  - Accepting regrade requests on Gradescope by Nov 10
- Today
  - Tree index
  - Midterm exam question

Tree index

B+ Trees

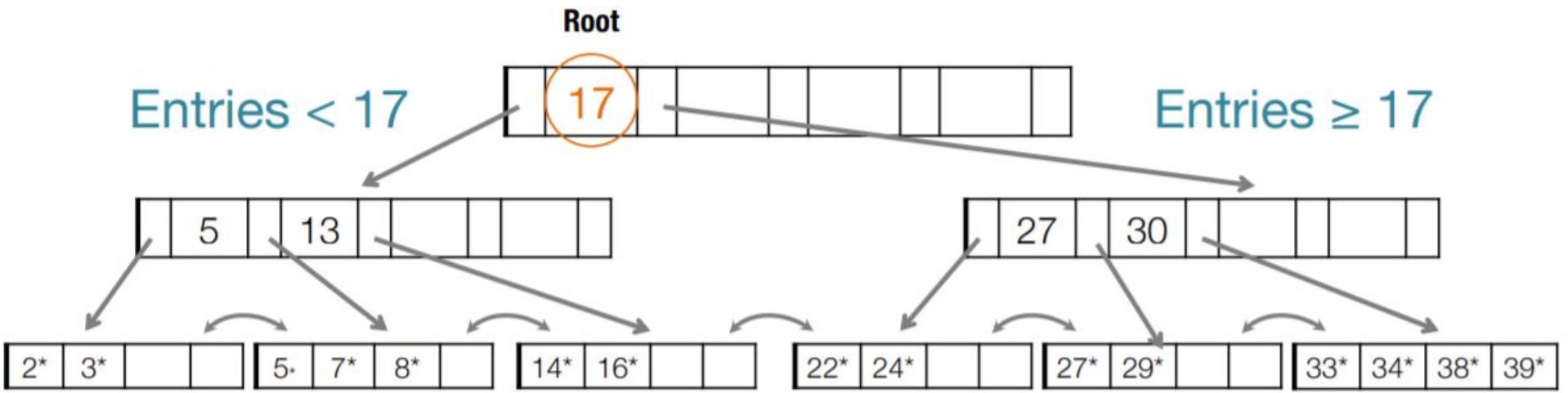
$d \le n \le 2d$

- Self balancing tree structure with multiple element in each node
  - All leaf nodes are the same height/depth
    - Height = length of any path from root to the leaf (# steps to take to get to the root)
  - Order is the minimum number of entries in all (non-root) nodes
  - 2*order is the maximum number (capacity) of entries in all nodes
  - Max fanout
    - Max pointers in an inner node (maximum number of children for a node)
    - 2*order+1 $2d+1$
- 3 main operations
  - Search
  - Insert
  - Delete



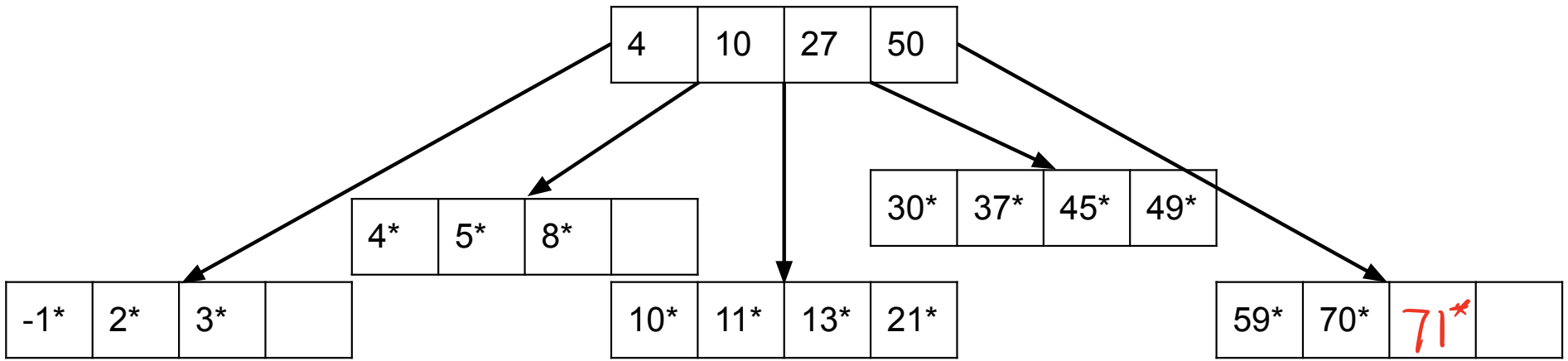
Searching in B+ Tree

- Searching for a particular element
  - Follow the pointers in each node until you find the leaf the element SHOULD exist in
    - No guarantee, if it doesn't exist in the leaf node it doesn't exist in the tree
  - Pointers are "guides"



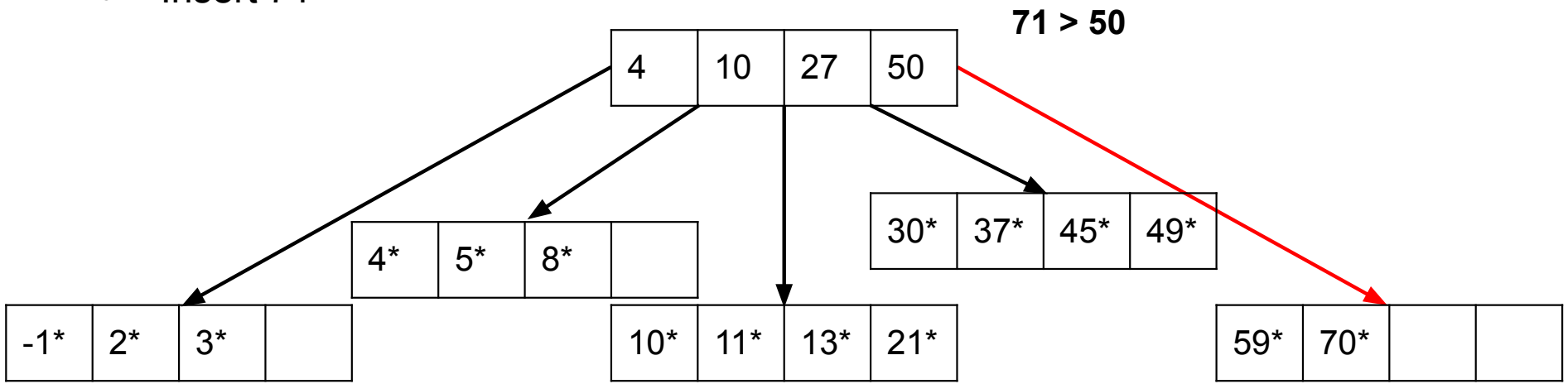
Inserting into a B+ Tree

- Add an element to the correct leaf node
  - If the desired leaf node has capacity, easy
    - Otherwise need to either split or redistribute
- Normal Insert
  - Insert 71*



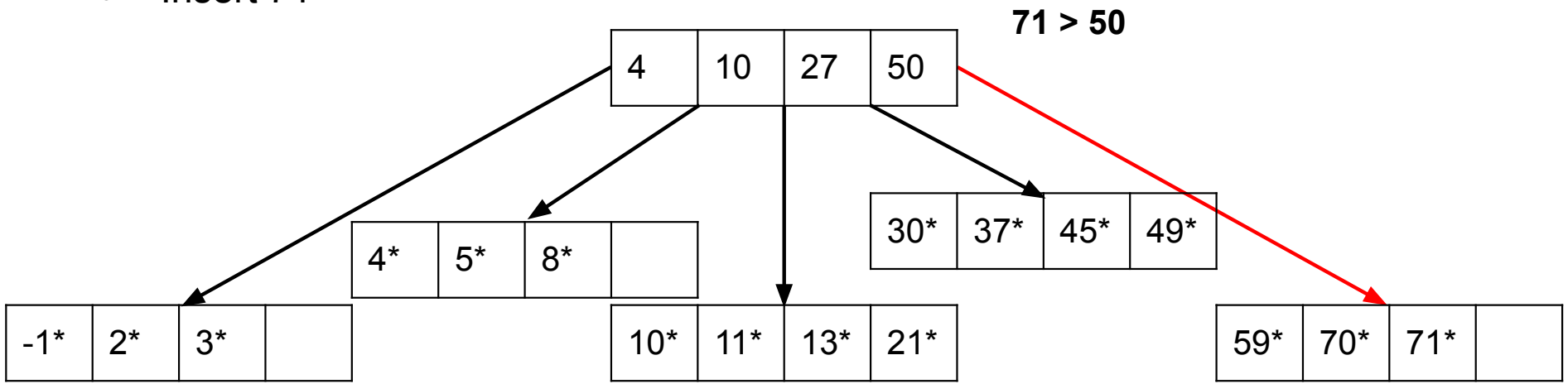
Inserting into a B+ Tree

- Add an element to the correct leaf node
  - If the desired leaf node has capacity, easy
    - Otherwise need to either split or redistribute
- Normal Insert
  - Insert 71*



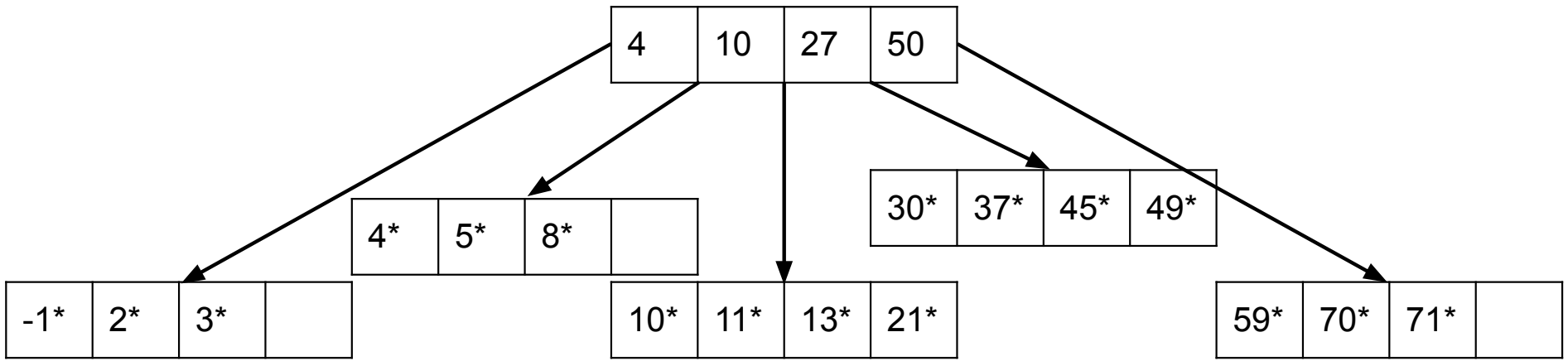
Inserting into a B+ Tree

- Add an element to the correct leaf node
  - If the desired leaf node has capacity, easy
    - Otherwise need to either split or redistribute
- Normal Insert
  - Insert 71*



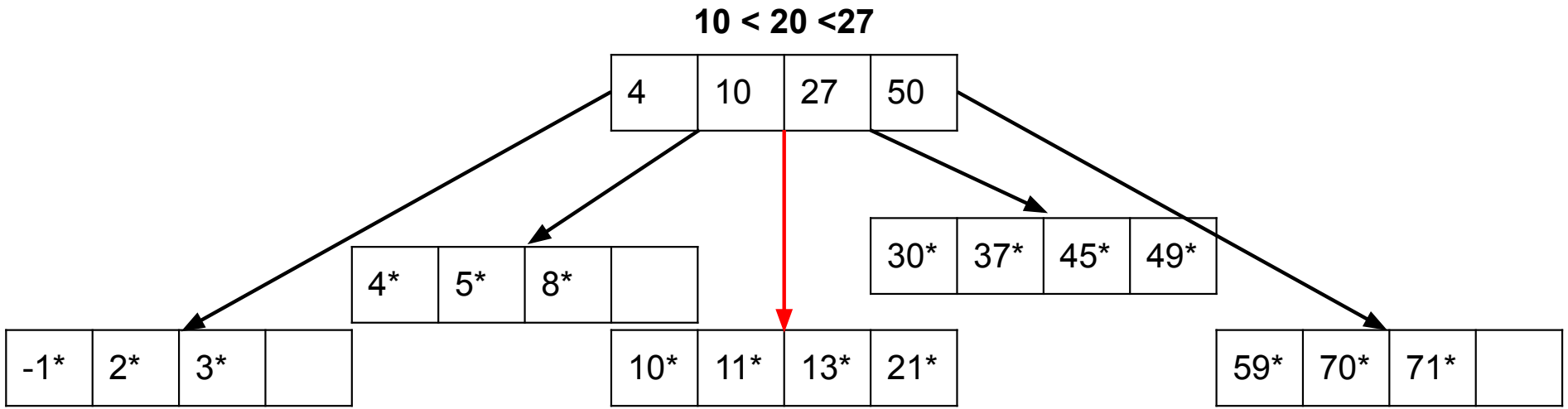
Inserting into a B+ Tree

- Redistribute elements to left sibling
  - Insert 20*



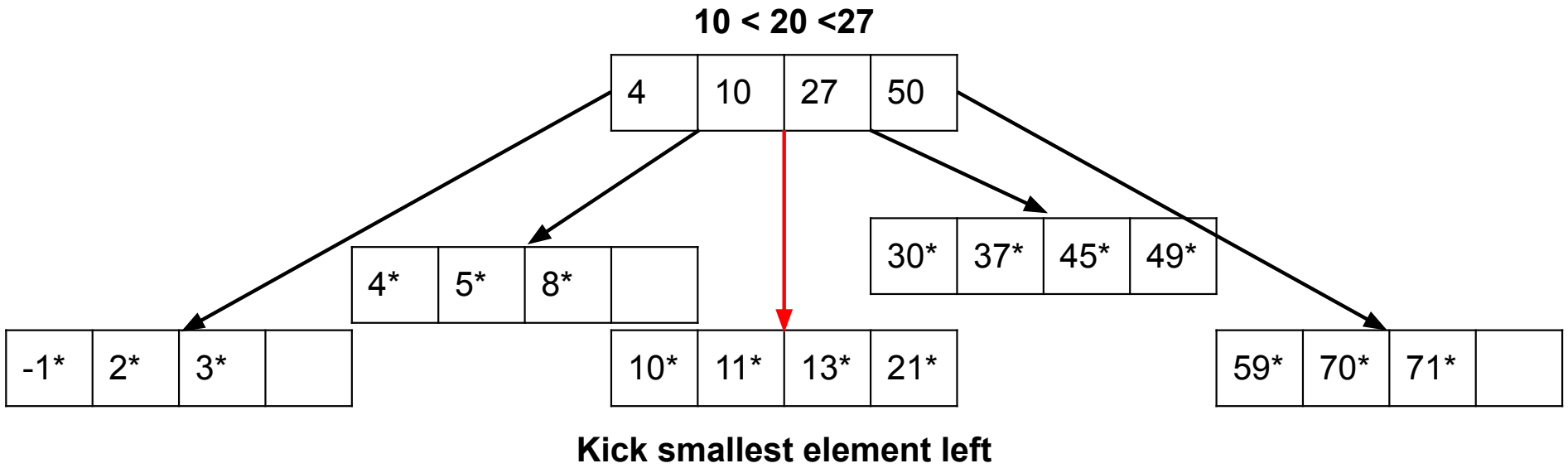
Inserting into a B+ Tree

- Redistribute elements to left sibling
  - Insert 20*



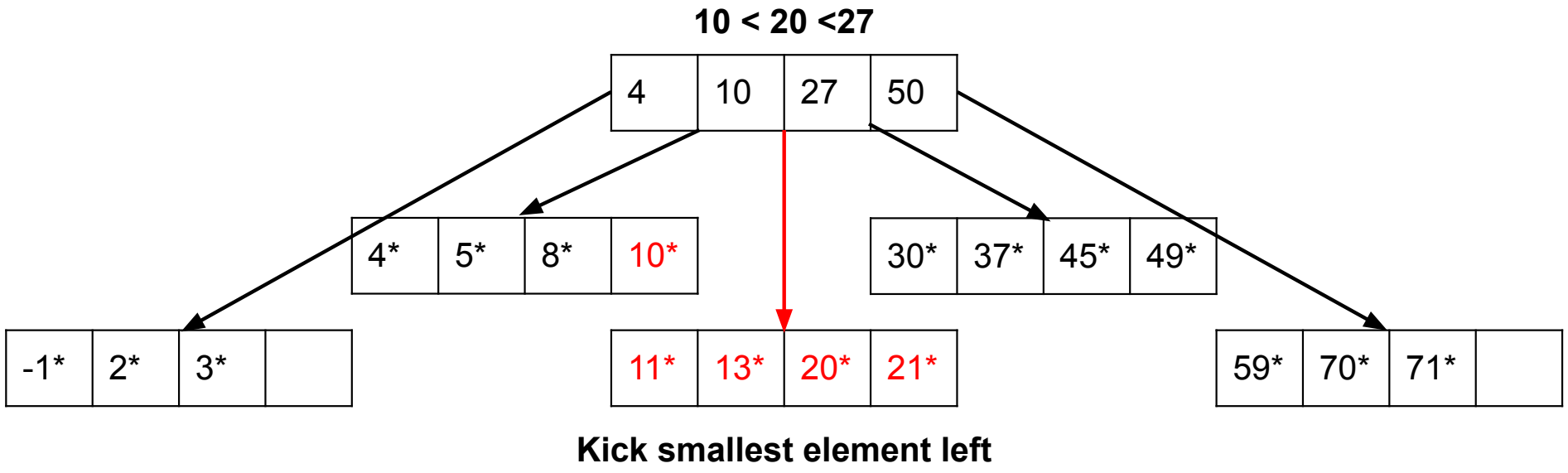
Inserting into a B+ Tree

- Redistribute elements to left sibling
  - Insert 20*



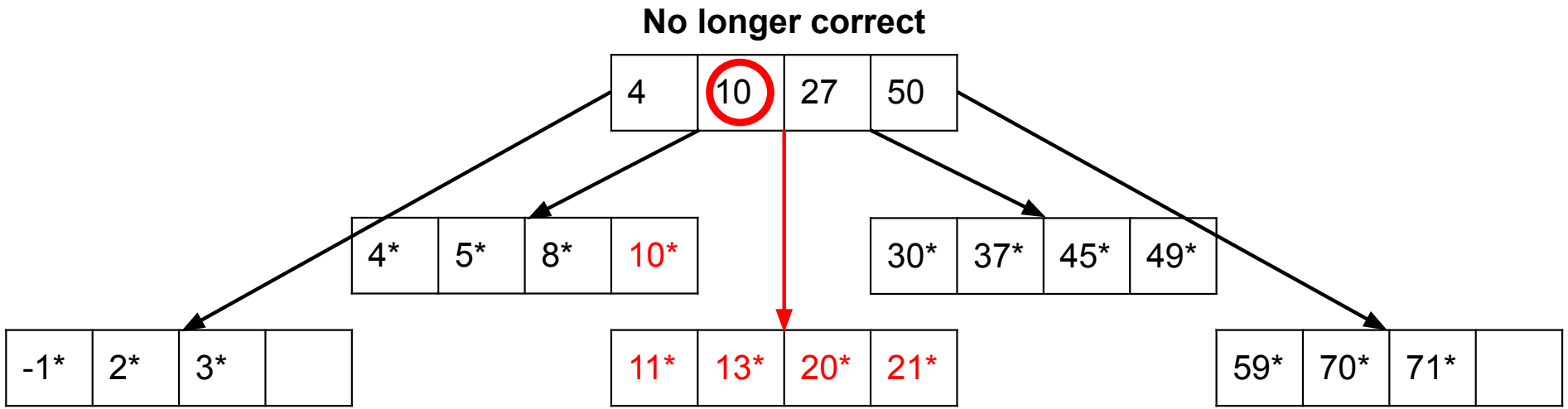
Inserting into a B+ Tree

- Redistribute elements to left sibling
  - Insert 20*



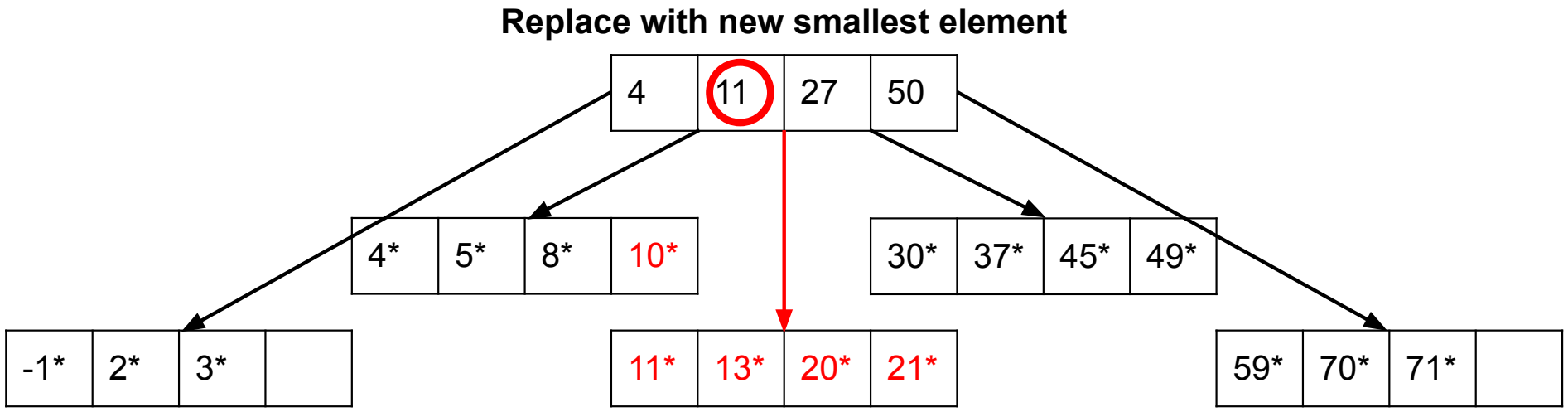
Inserting into a B+ Tree

- Redistribute elements to left sibling
  - Insert 20*



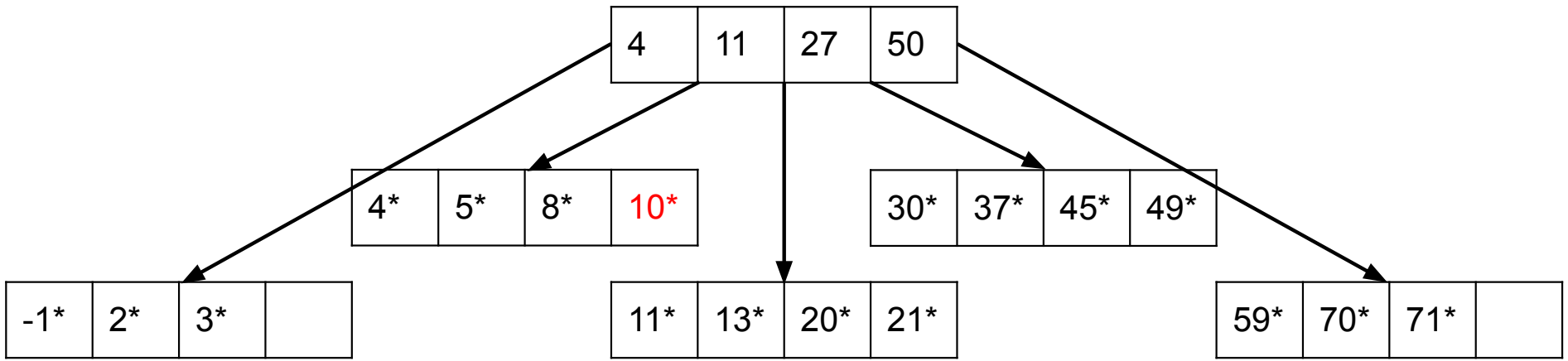
Inserting into a B+ Tree

- Redistribute elements to left sibling
  - Insert 20*



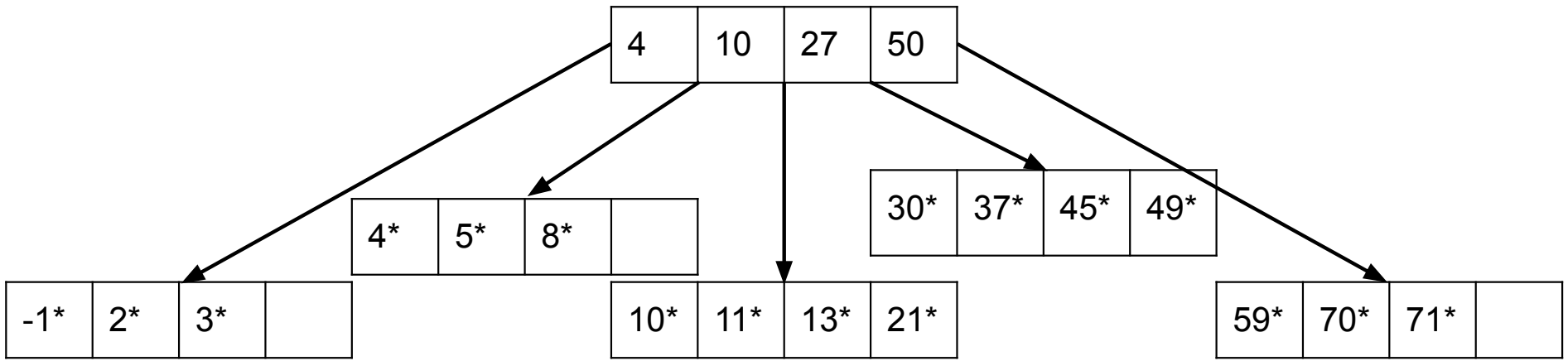
Inserting into a B+ Tree

- Redistribute elements to left sibling
  - Insert 20*
  - And we're done :)



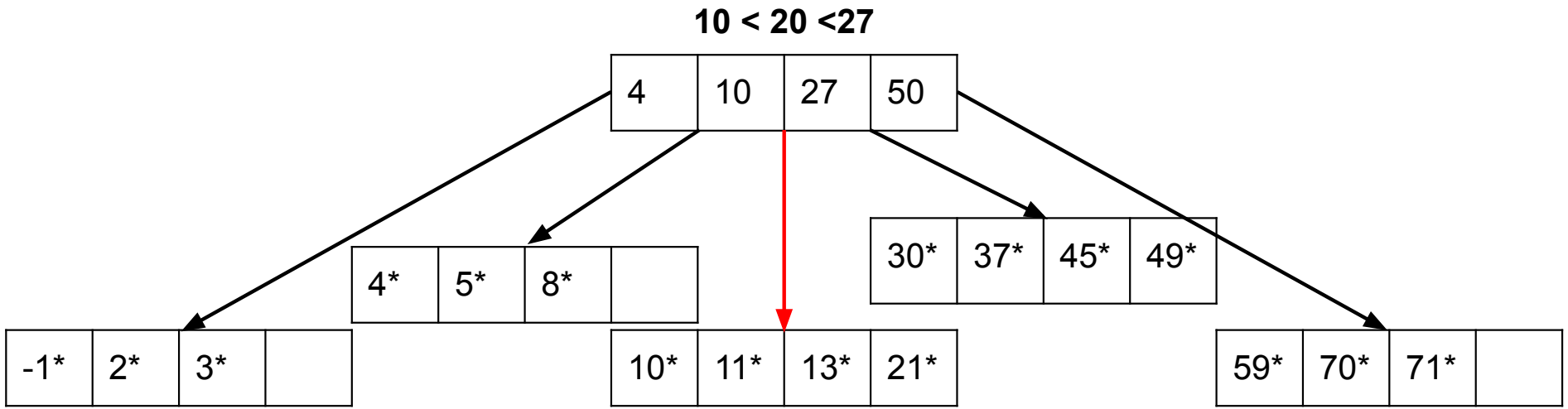
Inserting into a B+ Tree

- Split with extra elements in right child
  - Insert 20*



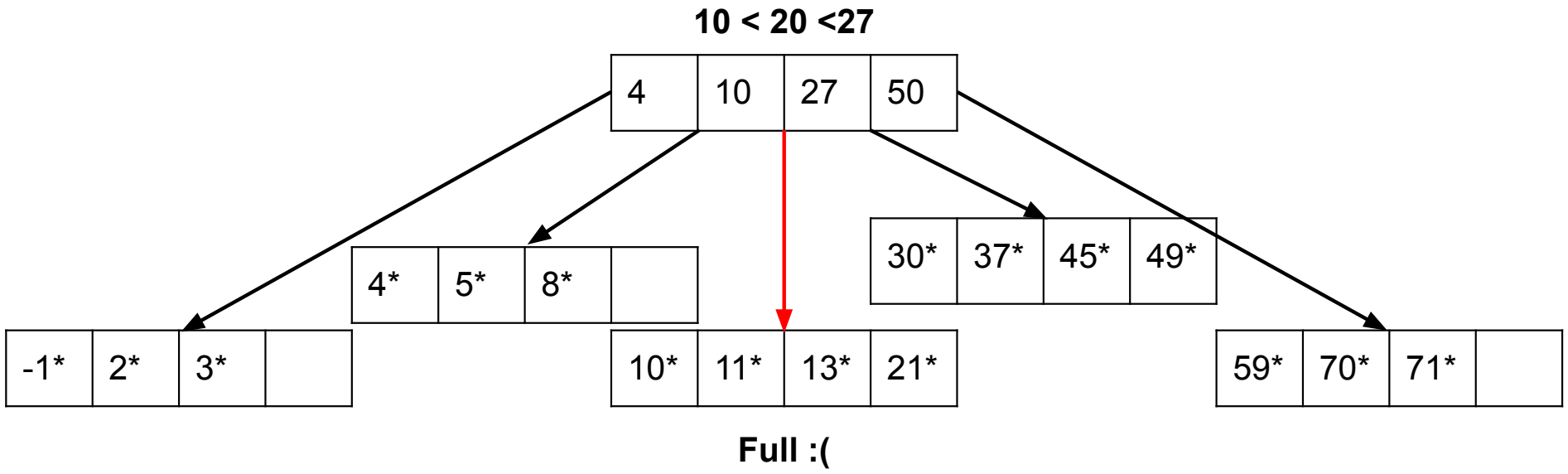
Inserting into a B+ Tree

- Split with extra elements in right child
  - Insert 20*



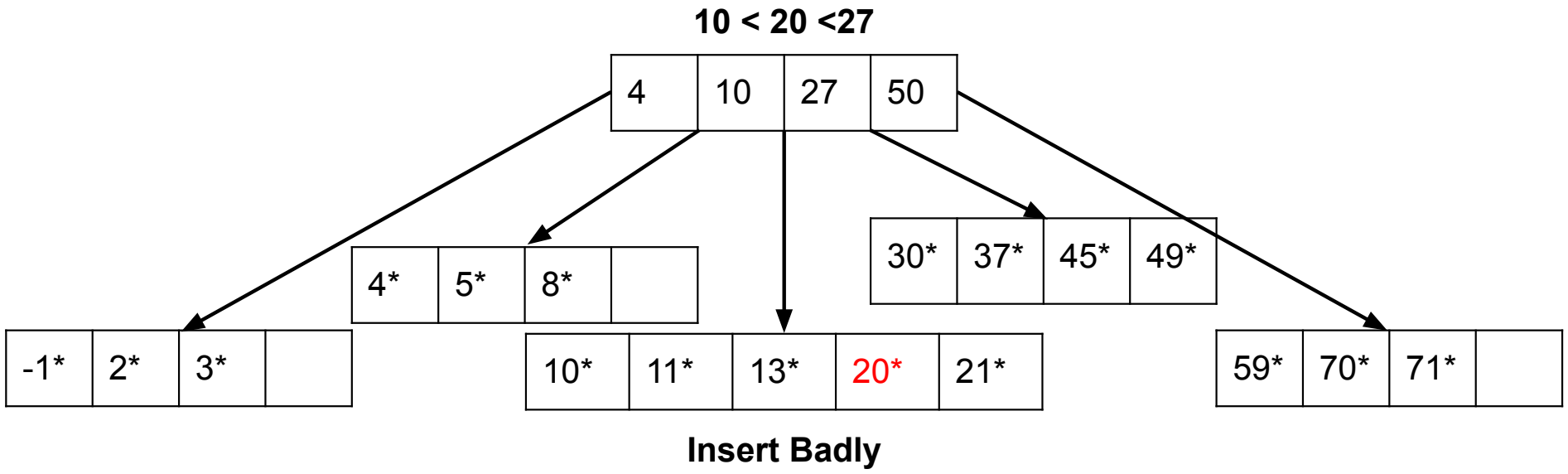
Inserting into a B+ Tree

- Split with extra elements in right child
  - Insert 20*



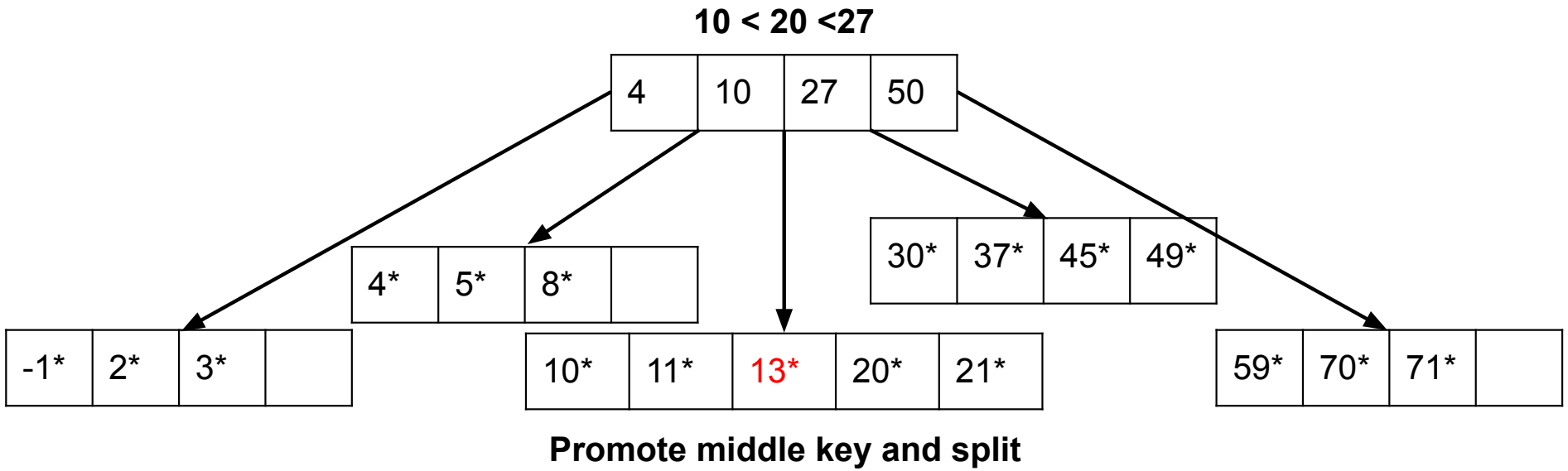
Inserting into a B+ Tree

- Split with extra elements in right child
  - Insert 20*



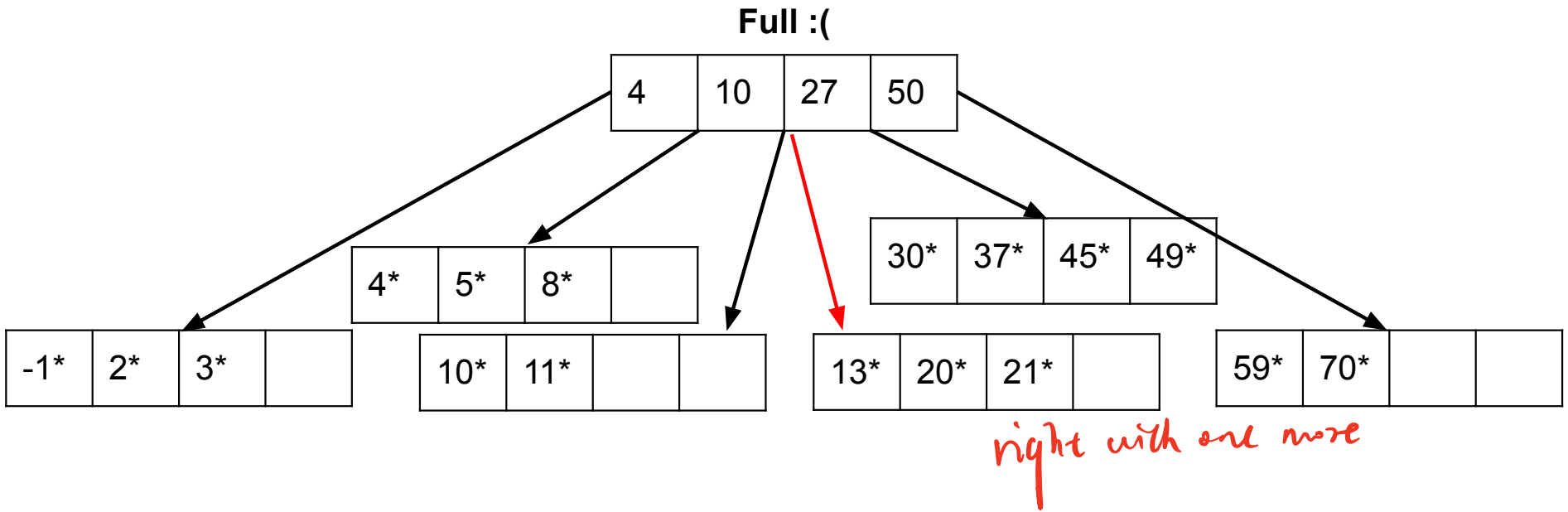
Inserting into a B+ Tree

- Split with extra elements in right child
  - Insert 20*



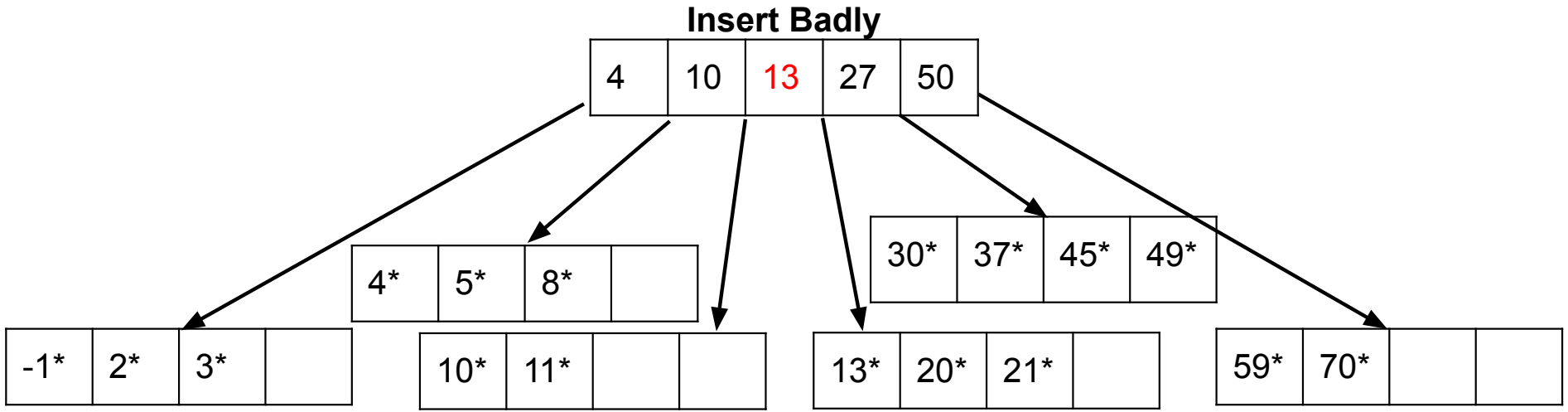
Inserting into a B+ Tree

- Split with extra elements in right child
  - Insert 20*



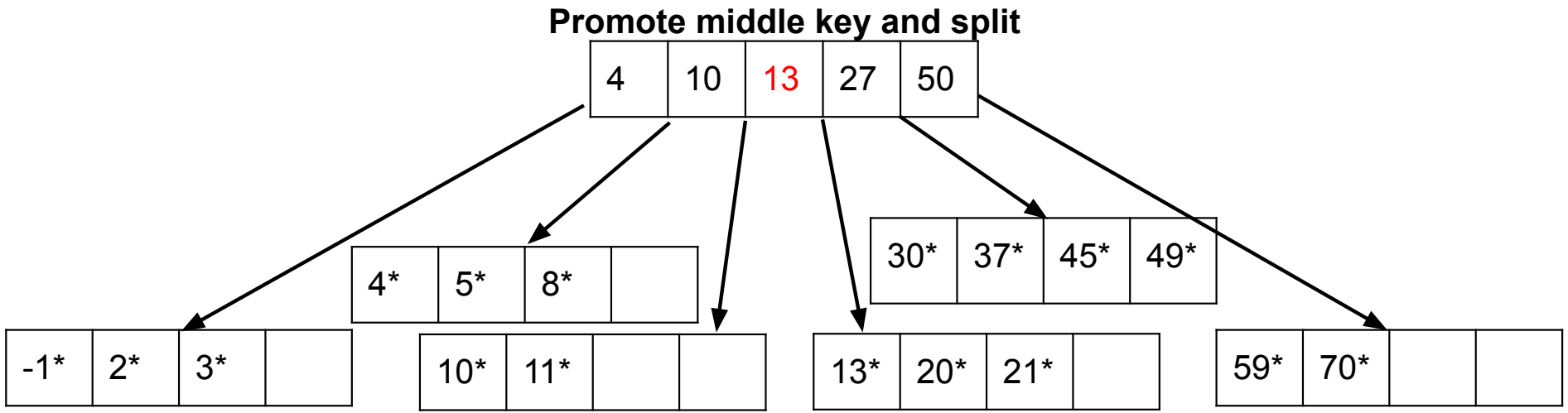
Inserting into a B+ Tree

- Split with extra elements in right child
  - Insert 20*



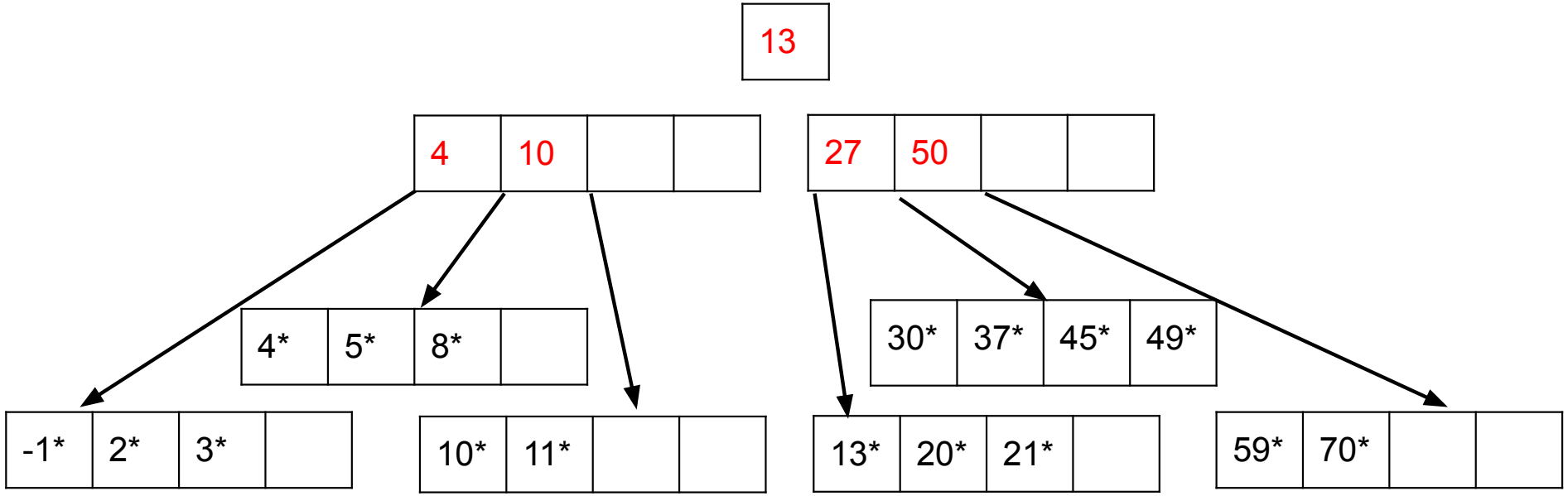
Inserting into a B+ Tree

- Split with extra elements in right child
  - Insert 20*



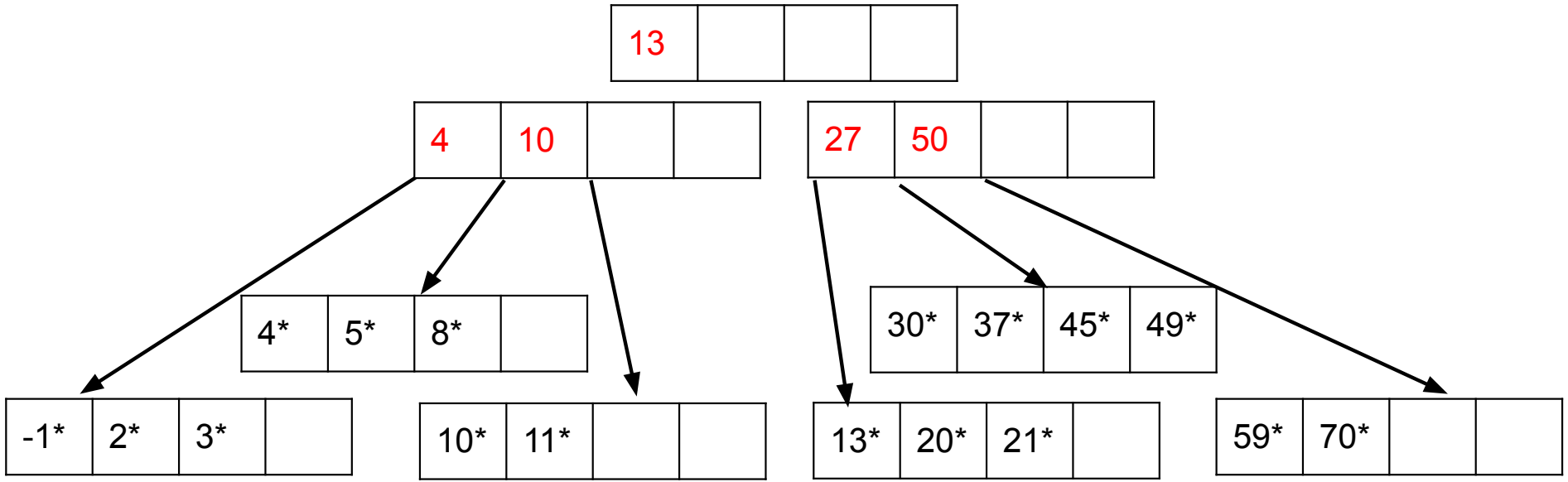
Inserting into a B+ Tree

- Split with extra elements in right child
  - Insert 20*



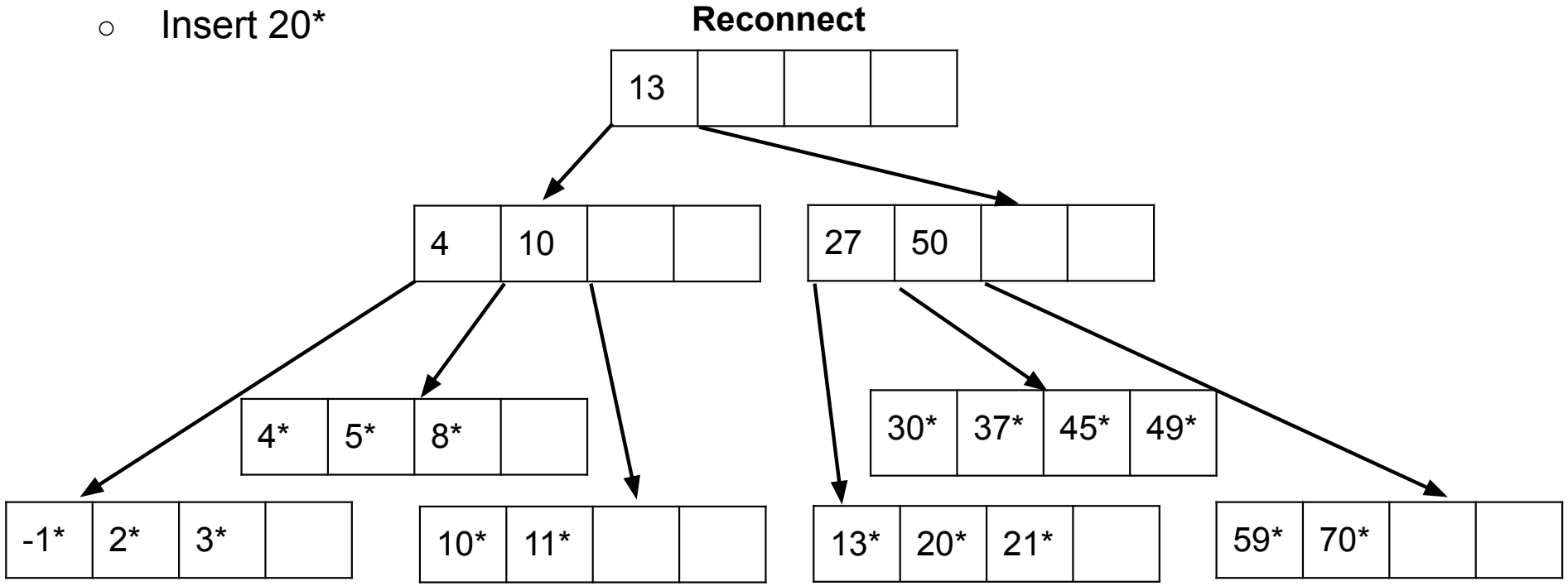
Inserting into a B+ Tree

- Split with extra elements in right child
  - Insert 20*



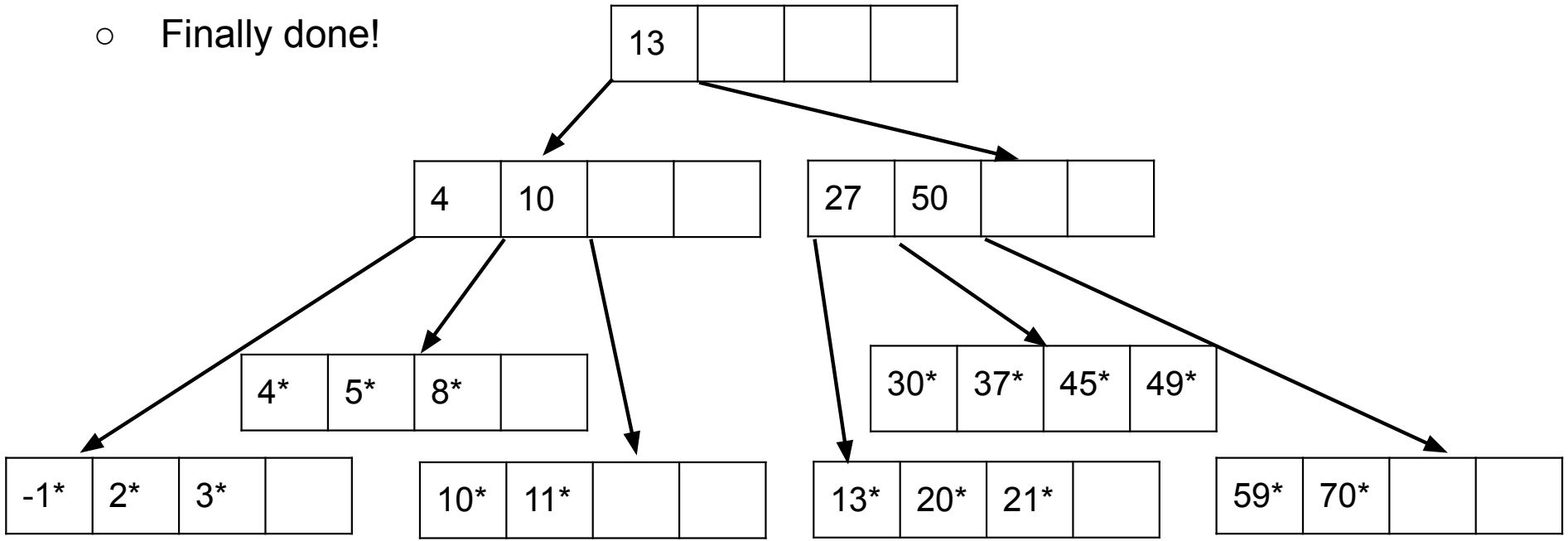
Inserting into a B+ Tree

- Split with extra elements in right child
  - Insert 20*



Inserting into a B+ Tree

- Split with extra elements in right child
  - Insert 20*
  - Finally done!



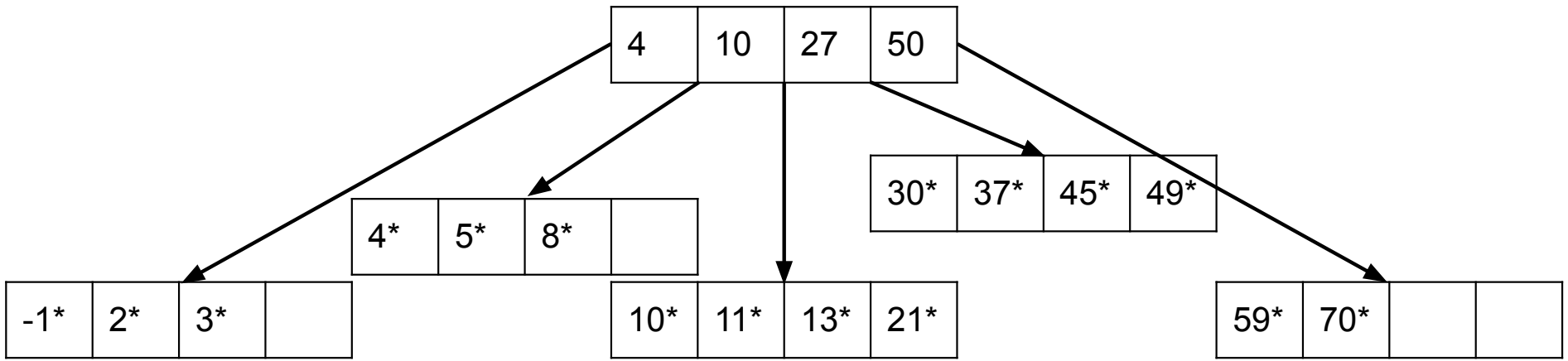
Inserting into a B+ Tree

Takeaways

- Redistributing is a lot less work
  - Usually smaller height
  - More data entries per page
  - More I/O (need to check right/left nodes)
  - Can’t do this if the right and left nodes are full

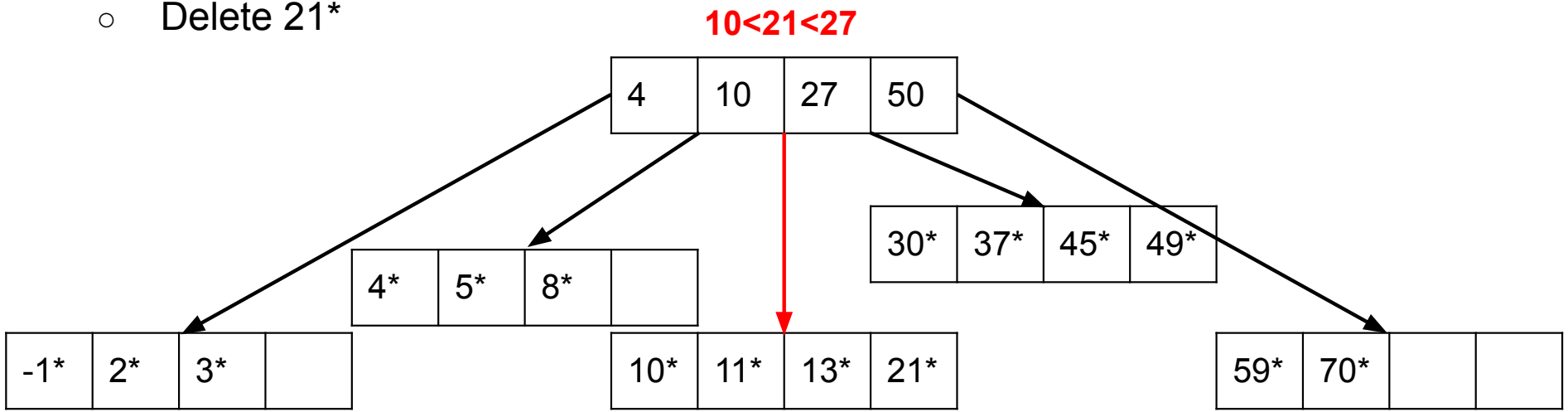
Deleting from a B+ Tree

- Delete an element from the tree
  - If the leaf node has elements >= order, then easy
    - Otherwise need to either redistribute or merge
- Normal Delete
  - Delete 21*



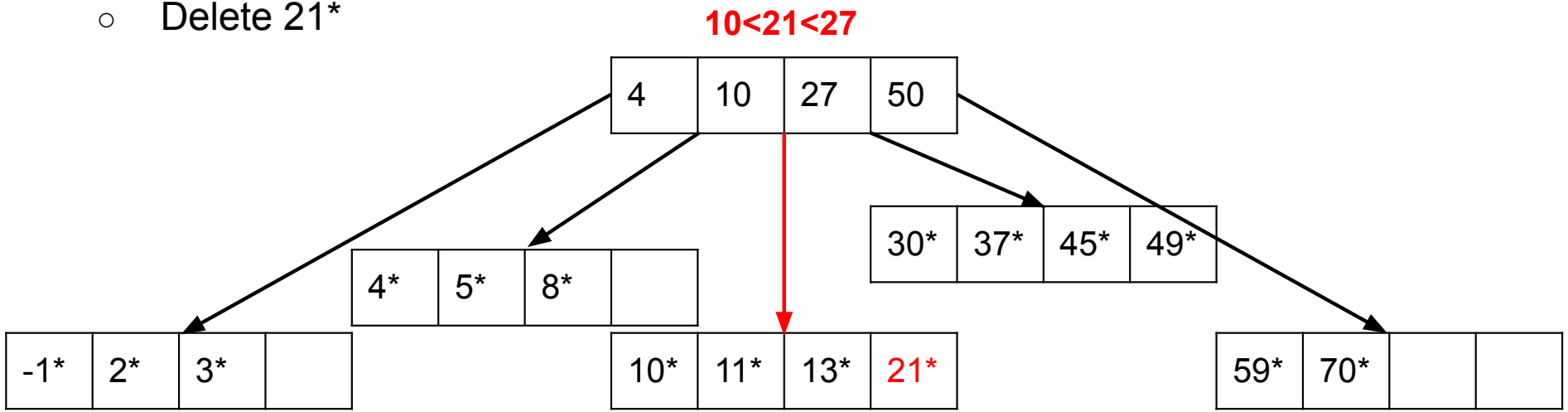
Deleting from a B+ Tree

- Delete an element from the tree
  - If the leaf node has elements >= order, then easy
    - Otherwise need to either redistribute or merge
- Normal Delete
  - Delete 21*



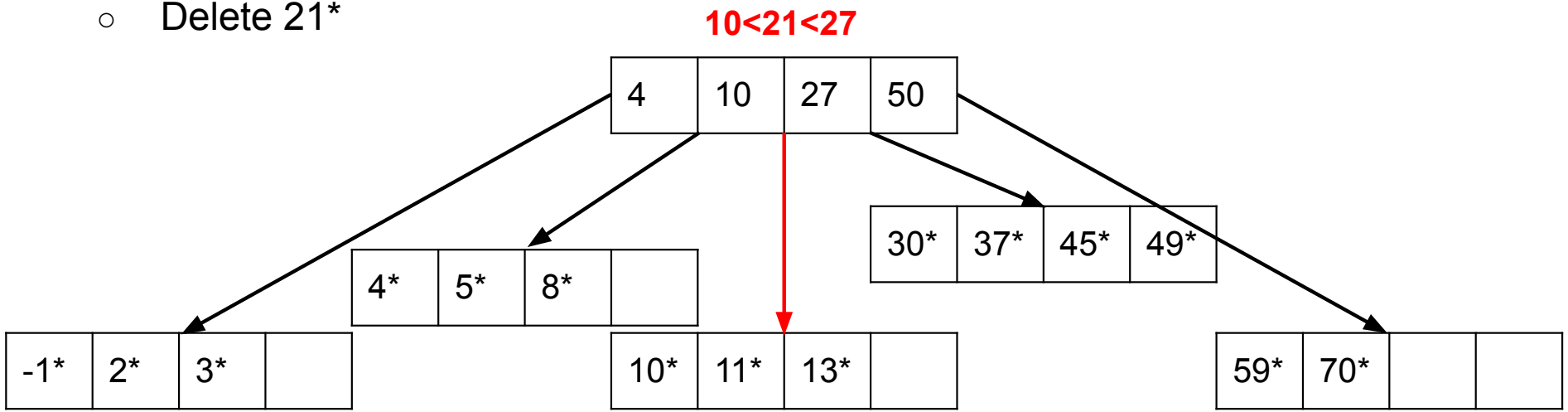
Deleting from a B+ Tree

- Delete an element from the tree
  - If the leaf node has elements >= order, then easy
    - Otherwise need to either redistribute or merge
- Normal Delete
  - Delete 21*



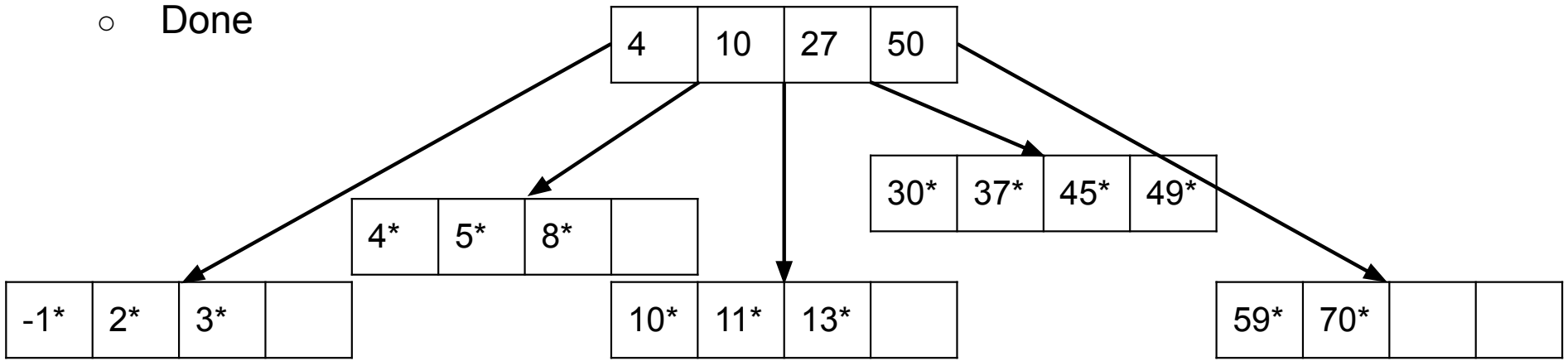
Deleting from a B+ Tree

- Delete an element from the tree
  - If the leaf node has elements >= order, then easy
    - Otherwise need to either redistribute or merge
- Normal Delete
  - Delete 21*



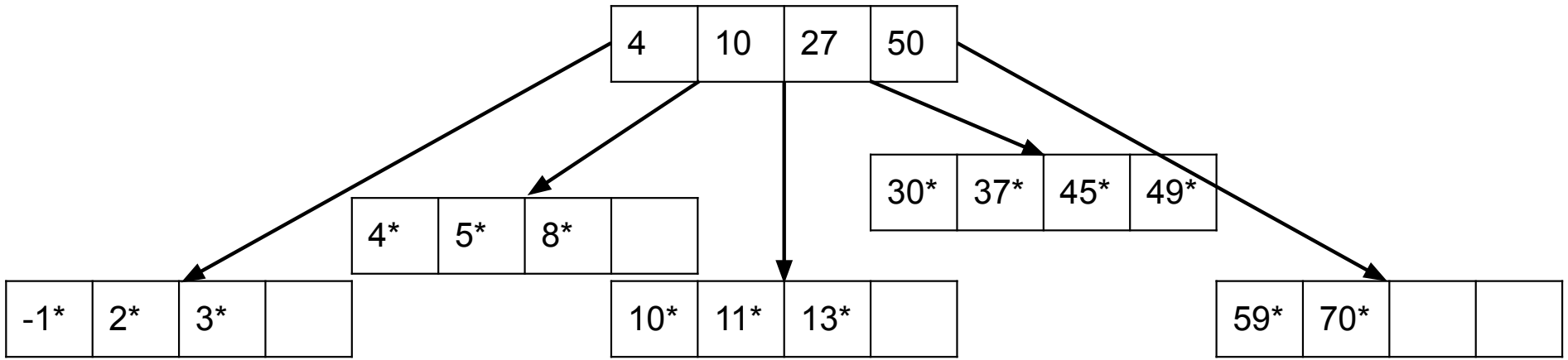
Deleting from a B+ Tree

- Delete an element from the tree
  - If the leaf node has elements >= order, then easy
    - Otherwise need to either redistribute or merge
- Normal Delete
  - Delete 21*
  - Done



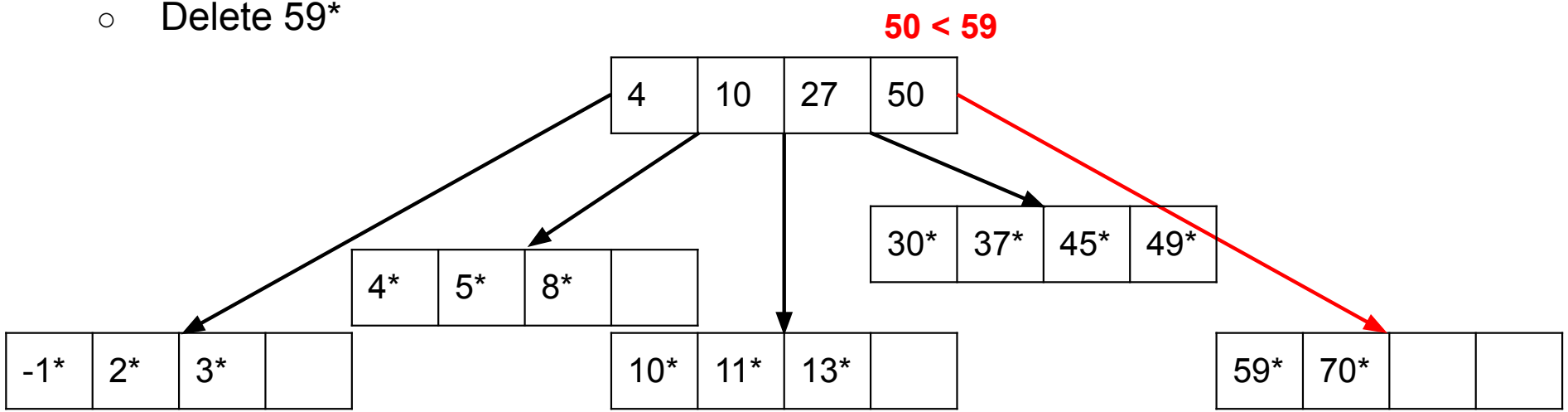
Deleting from a B+ Tree

- Delete an element from the tree
  - If the leaf node has elements >= order, then easy
    - Otherwise need to either redistribute or merge
- Try redistribution
  - Delete 59*



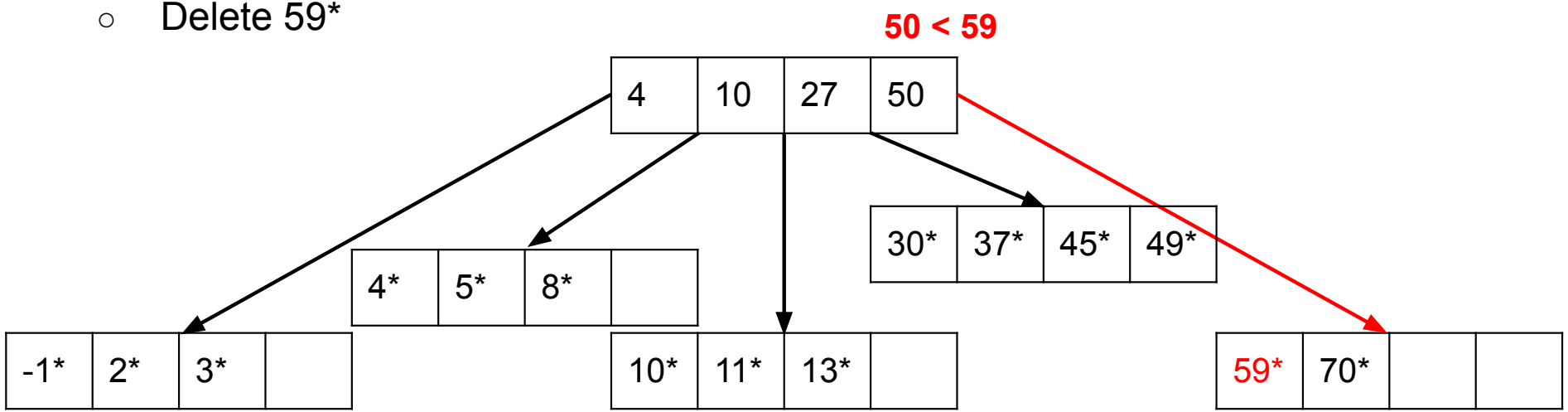
Deleting from a B+ Tree

- Delete an element from the tree
  - If the leaf node has elements >= order, then easy
    - Otherwise need to either redistribute or merge
- Try redistribution
  - Delete 59*



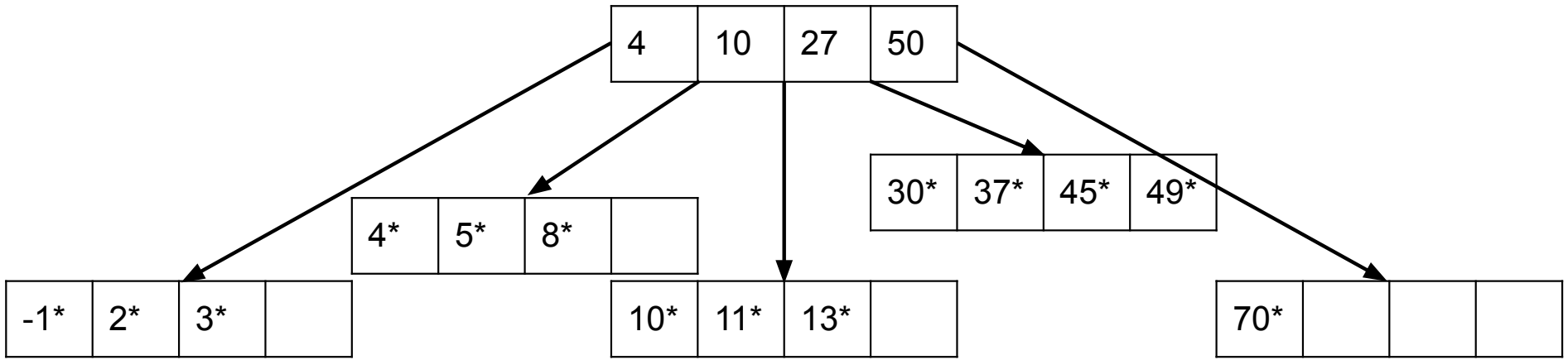
Deleting from a B+ Tree

- Delete an element from the tree
  - If the leaf node has elements >= order, then easy
    - Otherwise need to either redistribute or merge
- Try redistribution
  - Delete 59*



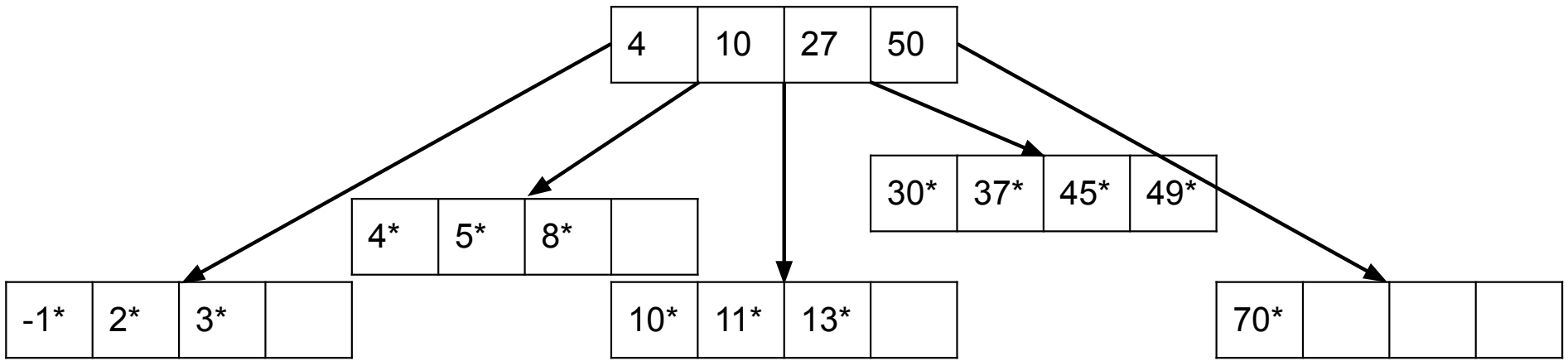
Deleting from a B+ Tree

- Delete an element from the tree
  - If the leaf node has elements >= order, then easy
    - Otherwise need to either redistribute or merge
- Try redistribution
  - Delete 59*



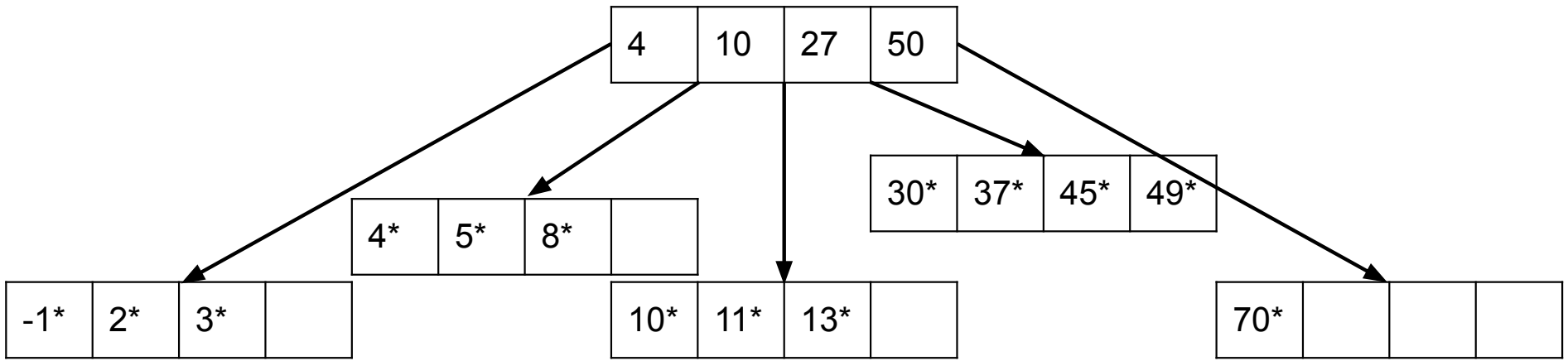
Deleting from a B+ Tree

- Delete an element from the tree
  - If the leaf node has elements >= order, then easy
    - Otherwise need to either redistribute or merge
- Try redistribution
  - Delete 59*



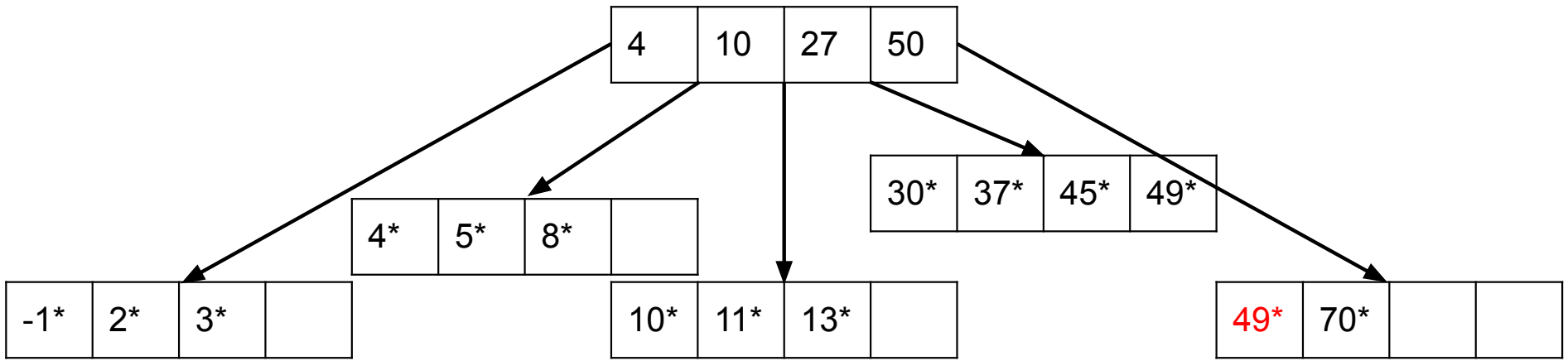
Deleting from a B+ Tree

- Delete an element from the tree
  - If the leaf node has elements >= order, then easy
    - Otherwise need to either redistribute or merge
- Try redistribution
  - Delete 59*



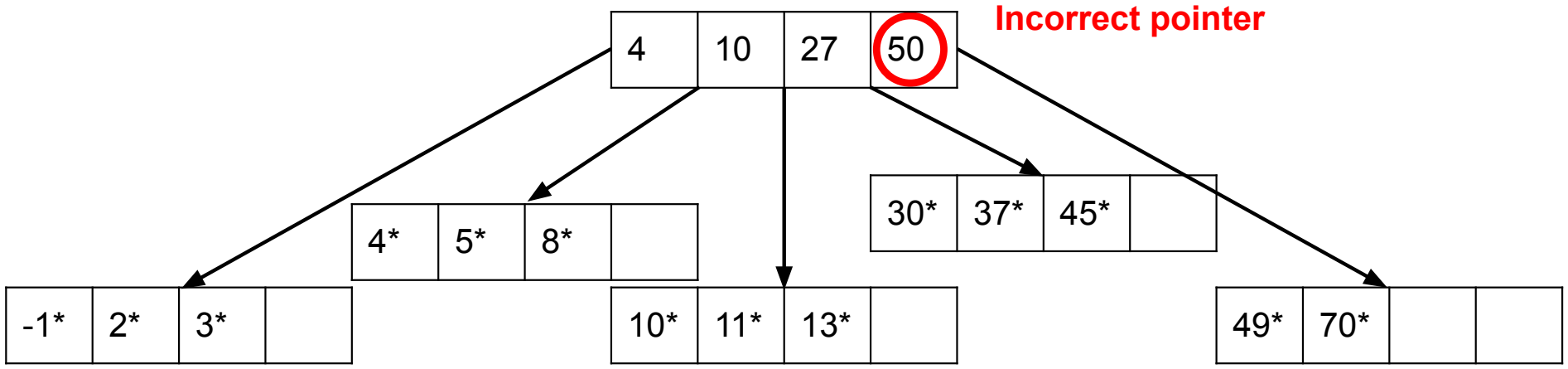
Deleting from a B+ Tree

- Delete an element from the tree
  - If the leaf node has elements >= order, then easy
    - Otherwise need to either redistribute or merge
- Try redistribution
  - Delete 59*



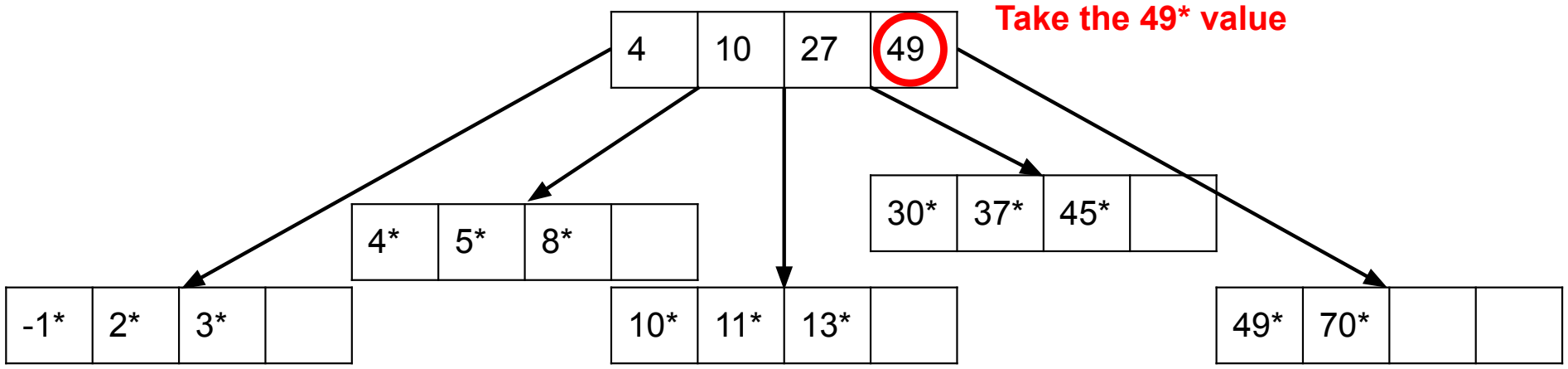
Deleting from a B+ Tree

- Delete an element from the tree
  - If the leaf node has elements >= order, then easy
    - Otherwise need to either redistribute or merge
- Try redistribution
  - Delete 59*



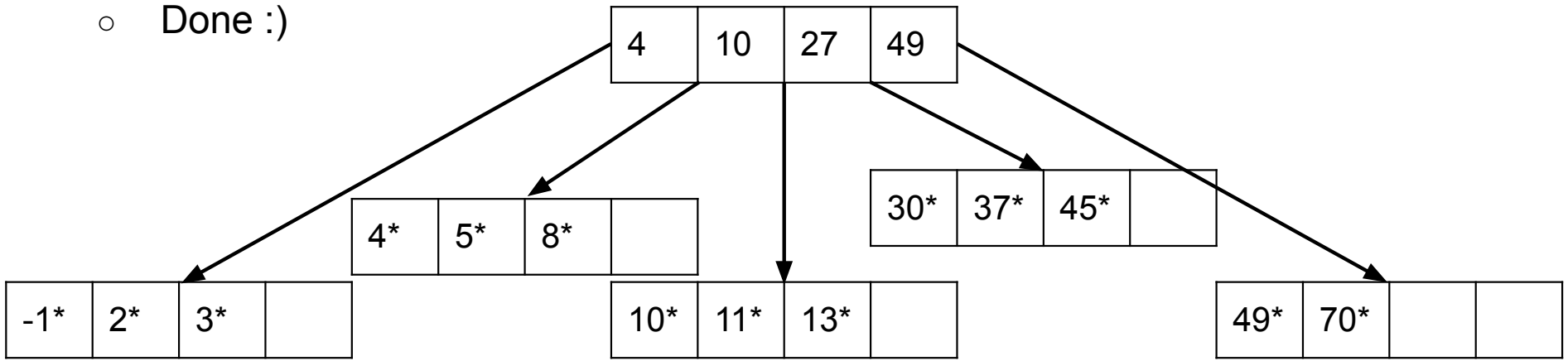
Deleting from a B+ Tree

- Delete an element from the tree
  - If the leaf node has elements >= order, then easy
    - Otherwise need to either redistribute or merge
- Try redistribution
  - Delete 59*



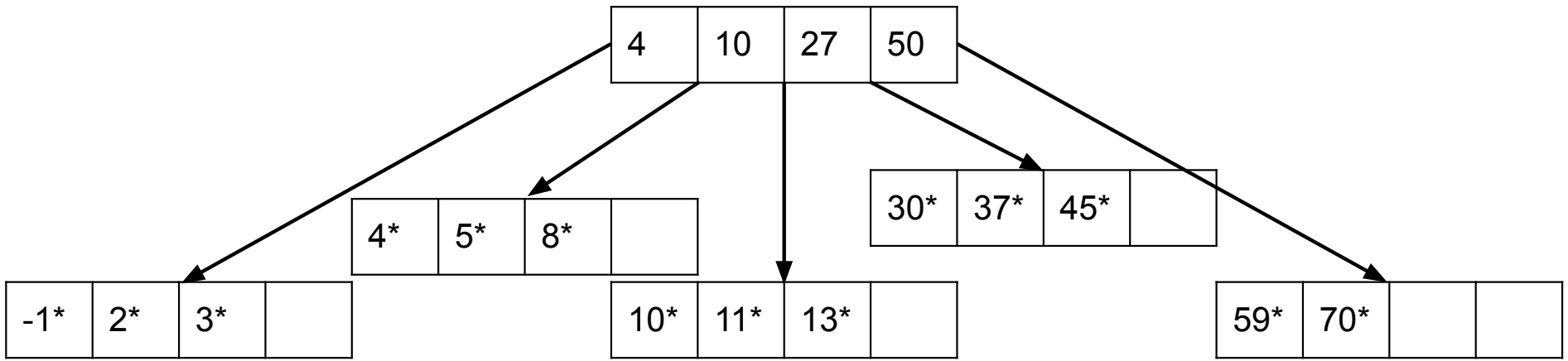
Deleting from a B+ Tree

- Delete an element from the tree
  - If the leaf node has elements >= order, then easy
    - Otherwise need to either redistribute or merge
- Try redistribution
  - Delete 59*
  - Done :)



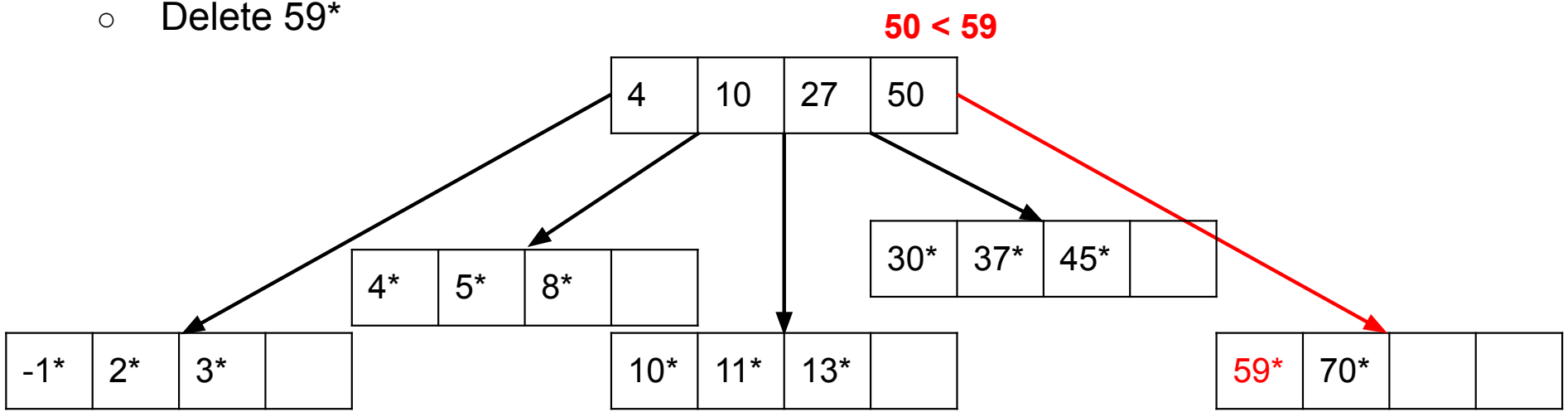
Deleting from a B+ Tree

- Delete an element from the tree
  - If the leaf node has elements >= order, then easy
    - Otherwise need to either redistribute or merge
- Try merging
  - Delete 59*



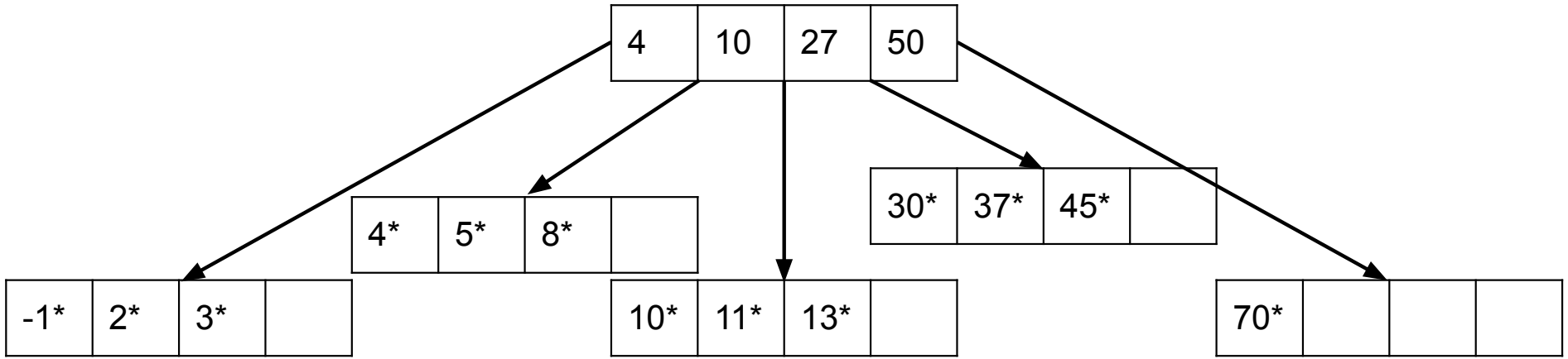
Deleting from a B+ Tree

- Delete an element from the tree
  - If the leaf node has elements >= order, then easy
    - Otherwise need to either redistribute or merge
- Try merging
  - Delete 59*



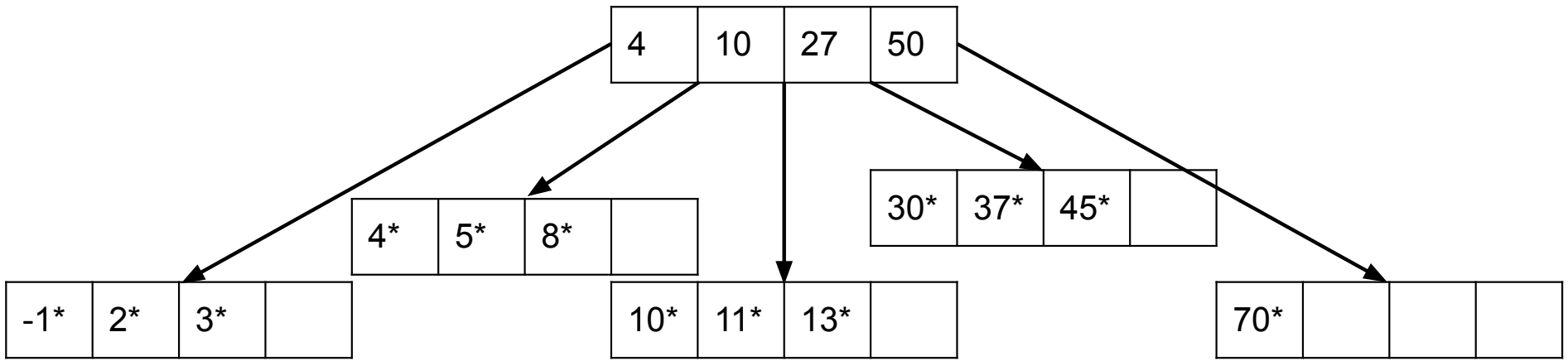
Deleting from a B+ Tree

- Delete an element from the tree
  - If the leaf node has elements >= order, then easy
    - Otherwise need to either redistribute or merge
- Try merging
  - Delete 59*



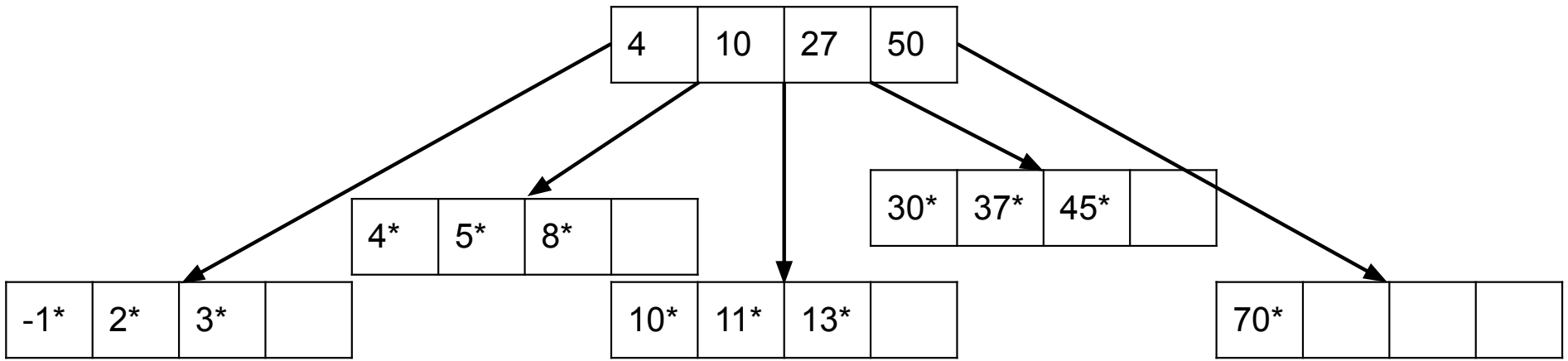
Deleting from a B+ Tree

- Delete an element from the tree
  - If the leaf node has elements >= order, then easy
    - Otherwise need to either redistribute or merge
- Try merging
  - Delete 59*



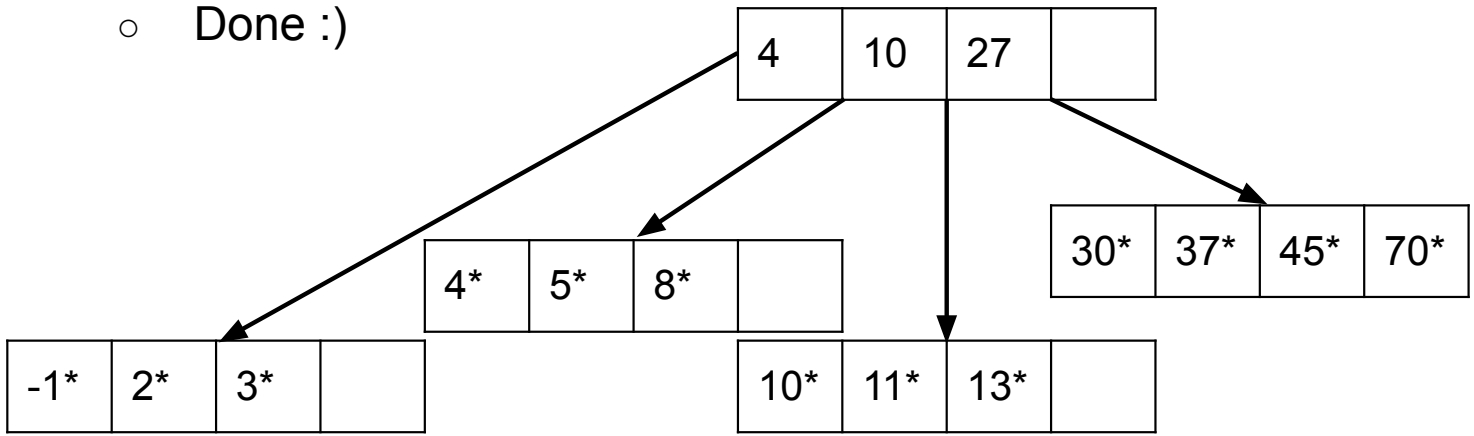
Deleting from a B+ Tree

- Delete an element from the tree
  - If the leaf node has elements >= order, then easy
    - Otherwise need to either redistribute or merge
- Try merging
  - Delete 59*



Deleting from a B+ Tree

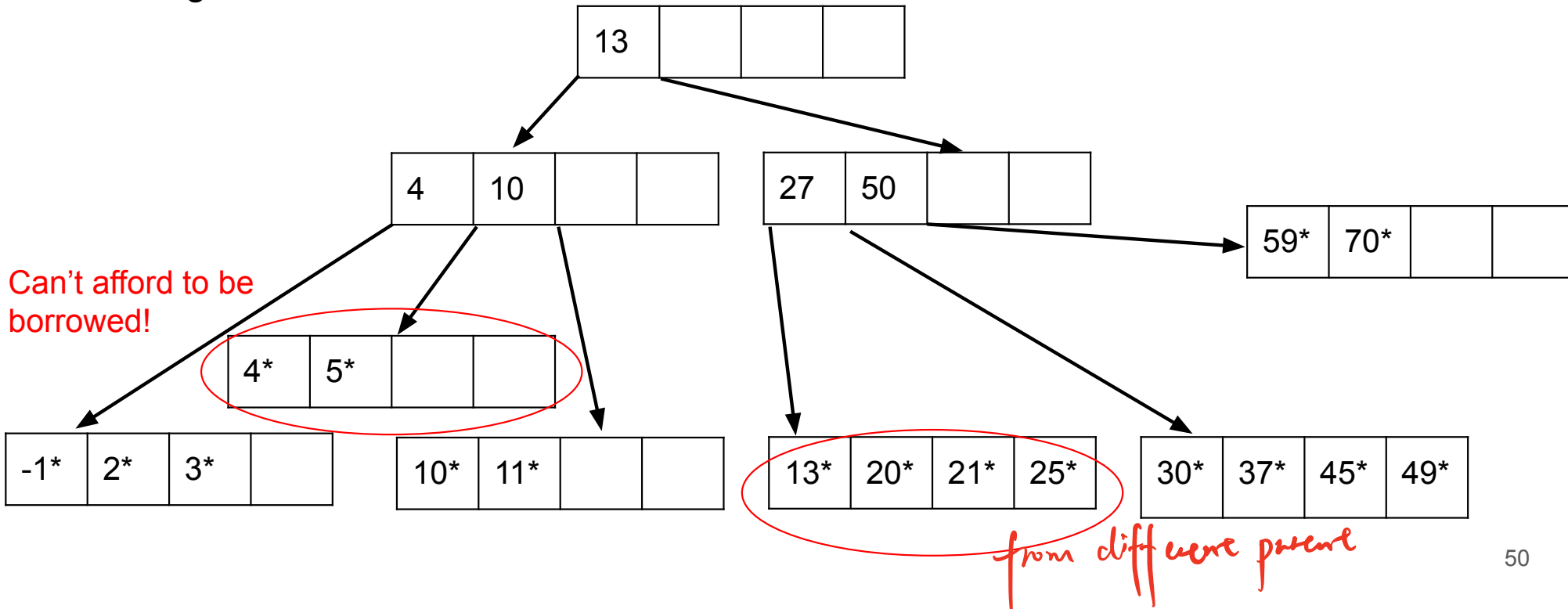
- Delete an element from the tree
  - If the leaf node has elements >= order, then easy
    - Otherwise need to either redistribute or merge
- Try merging
  - Delete 59*
  - Done :)



Redistribution with adjacent nodes (for leaf nodes)

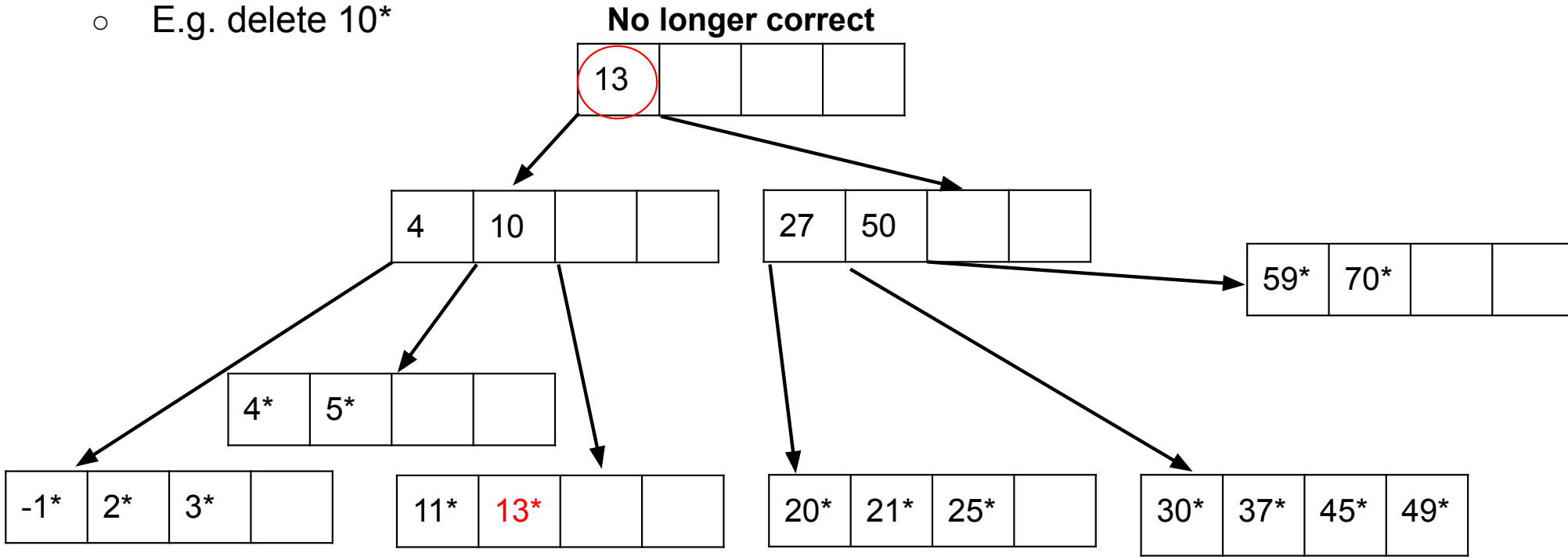
neiboring

- Redistribution is possible even with “siblings” who have a different parent
  - E.g. delete 10*



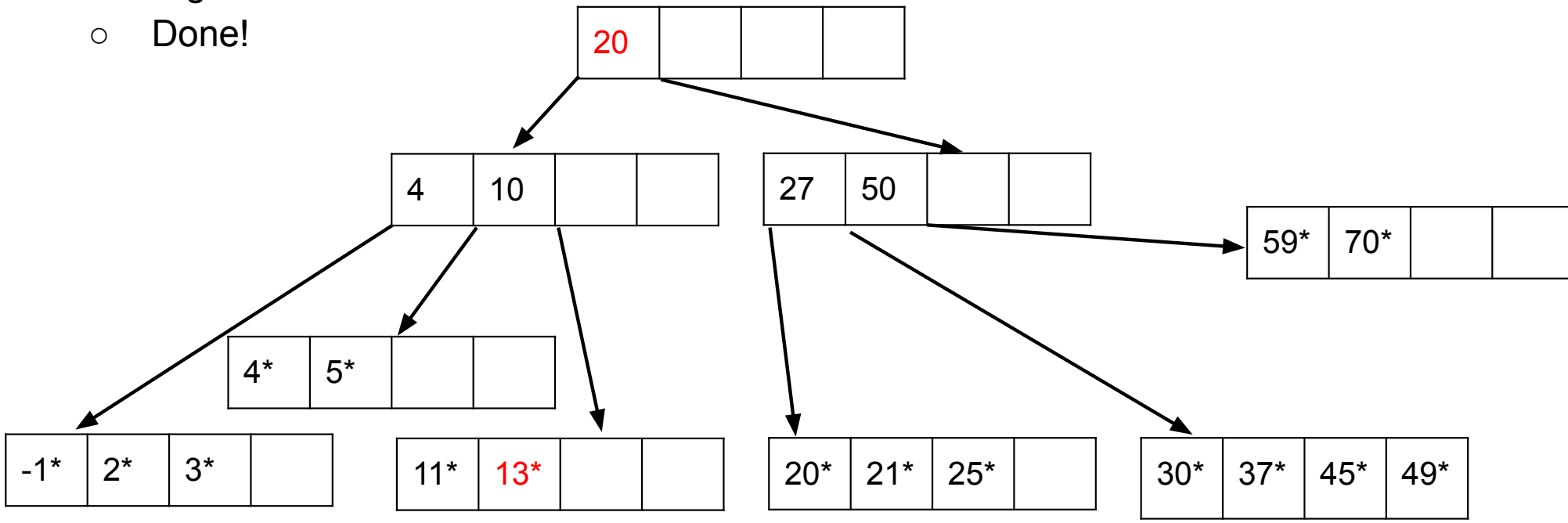
Redistribution with adjacent nodes (for leaf nodes)

- Redistribution is possible even with “siblings” who have a different parent
  - E.g. delete 10*



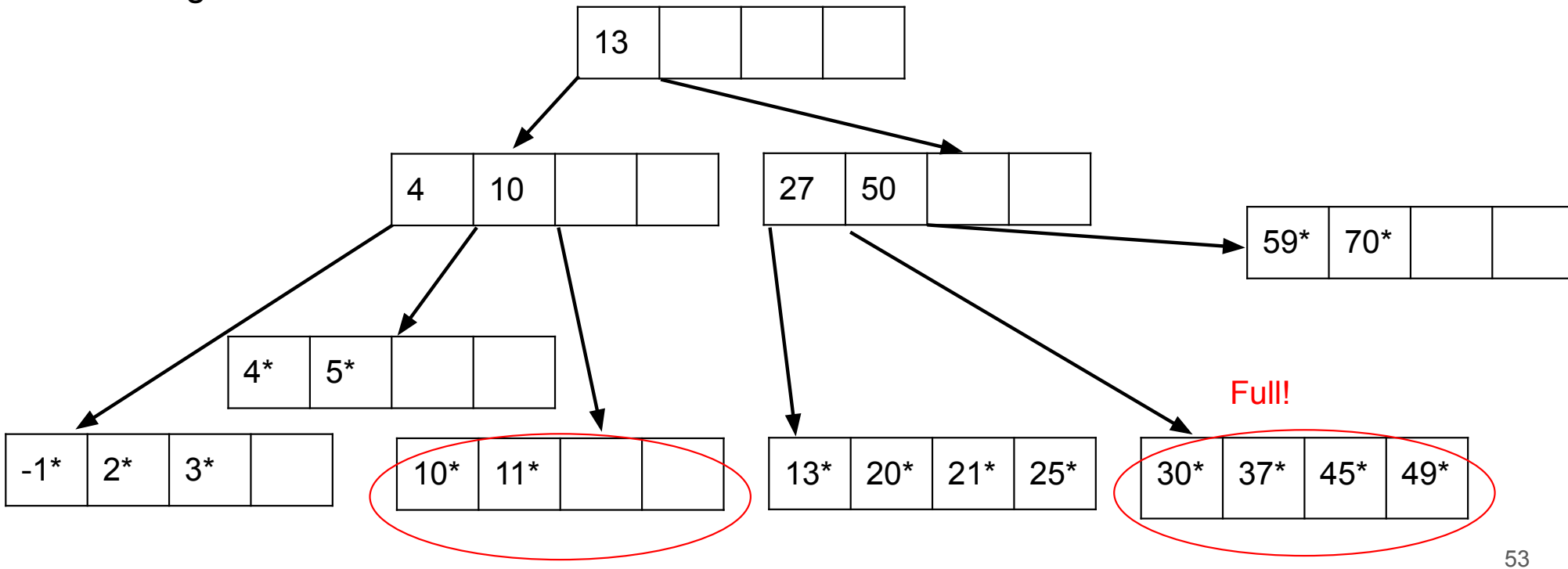
Redistribution with adjacent nodes (for leaf nodes)

- Redistribution is possible even with “siblings” who have a different parent
  - E.g. delete 10*
  - Done!



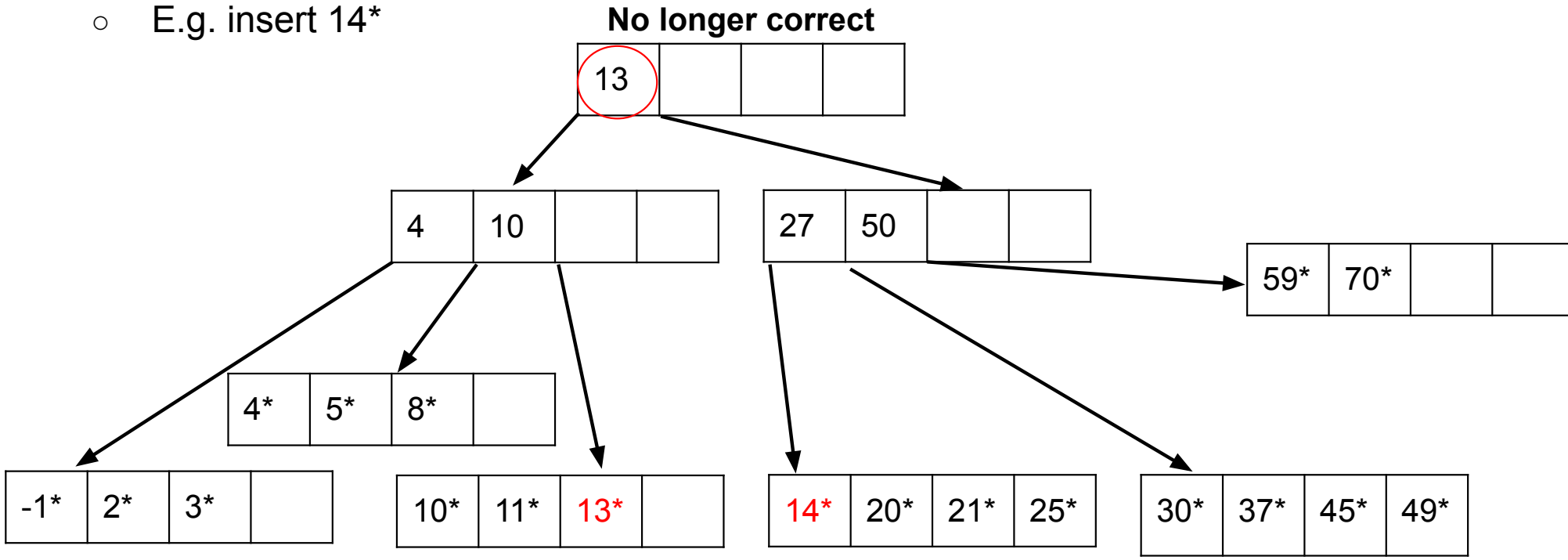
Redistribution with adjacent nodes (for leaf nodes)

- Redistribution is possible even with “siblings” who have a different parent
  - E.g. insert 14*



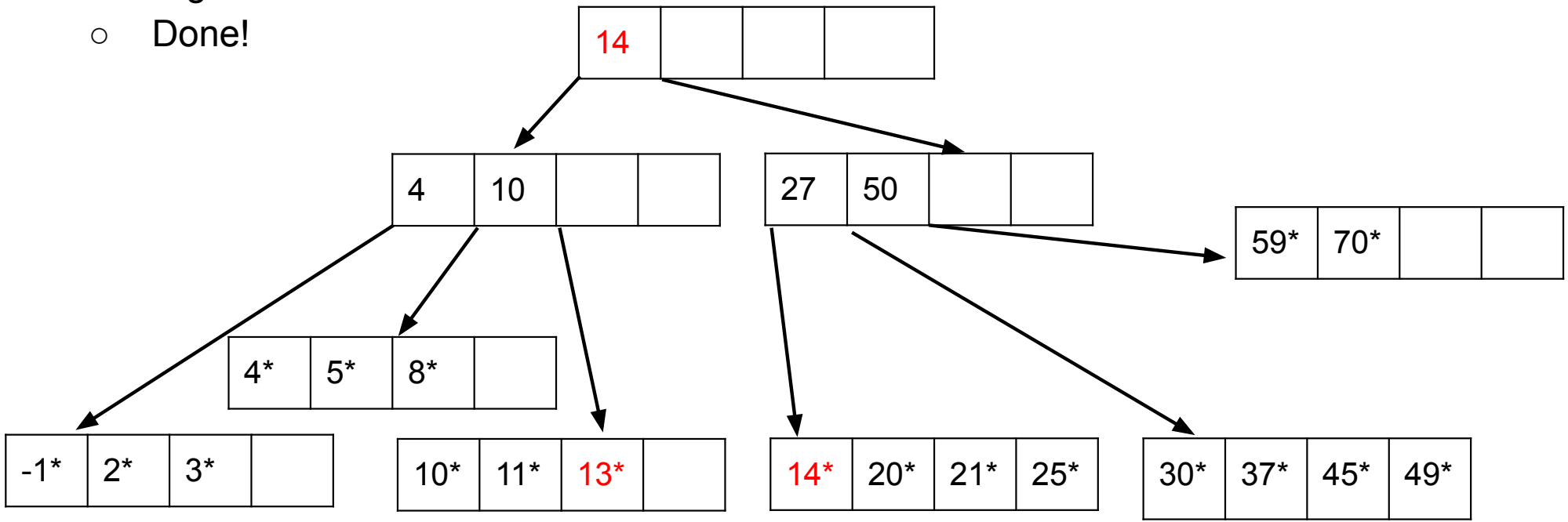
Redistribution with adjacent nodes (for leaf nodes)

- Redistribution is possible even with “siblings” who have a different parent
  - E.g. insert 14*



Redistribution with adjacent nodes (for leaf nodes)

- Redistribution is possible even with “siblings” who have a different parent
  - E.g. insert 14*
  - Done!



Midterm exam questions

Fakeify (RA, RC, SQL)

Artists(ArtistID, Name, Age)

Albums(AlbumID, Title, Year)

Album_Publishers(**AlbumID**, **ArtistID**)

Songs(SongID, **AlbumID**, Title)

Playlists(PlaylistID, Title)

Entries(**PlaylistID**, **SongID**, Description)

- Primary keys are not null
- **Foreign keys** are not null
- Other attributes can be null

- (1) Albums in *Albums* but not in *Album_Publishers* are unpublished
- (2) Playlists in *Playlists* but not in *Entries* do not have any songs
- (same for other primary/foreign key relationships)

- Do not return duplicates for RA and SQL!!

Fakeify (RA Q12)

Artists(ArtistID, Name, Age)

Albums(AlbumID, Title, Year)

Album_Publishers(**AlbumID**, **ArtistID**)

Songs(SongID, **AlbumID**, Title)

Playlists(PlaylistID, Title)

Entries(**PlaylistID**, **SongID**, Description)

- Albums in *Albums* but not in *Album_Publishers* are unpublished
- Do not return duplicates

Q: Return the IDs and titles of songs that are in unpublished albums.

Things to consider:

- We are looking for *titles* and *IDs* of songs
- Unpublished albums are albums in *Albums* but not in *Album_Publishers*
- A song is either in a published album or in an unpublished album

Fakeify (RA Q12)

Artists(ArtistID, Name, Age)

Albums(AlbumID, Title, Year)

Album_Publishers(**AlbumID**, **ArtistID**)

Songs(SongID, **AlbumID**, Title)

Playlists(PlaylistID, Title)

Entries(**PlaylistID**, **SongID**, Description)

- Albums in *Albums* but not in *Album_Publishers* are unpublished
- Do not return duplicates

Q: Return the IDs and titles of songs that are in unpublished albums.

Approach 1:

1. Get all unpublished albums

π_{AlbumID} (Albums) - π_{AlbumID} (Album_Publishers)

2. The songs in the albums in 1 are what we need
   a. Rename the albums in 1 as unpublished_albums for convenience

ρ (Unpublished_Albums, π_{AlbumID} (Albums) - π_{AlbumID} (Album_Publishers))

Fakeify (RA Q12)

Artists(ArtistID, Name, Age)

Albums(AlbumID, Title, Year)

Album_Publishers(**AlbumID**, **ArtistID**)

Songs(SongID, **AlbumID**, Title)

Playlists(PlaylistID, Title)

Entries(**PlaylistID**, **SongID**, Description)

- Albums in *Albums* but not in *Album_Publishers* are unpublished
- Do not return duplicates

Q: Return the IDs and titles of songs that are in unpublished albums.

Approach 1:

1. Get all unpublished albums

$\pi_{\text{AlbumID}}(\text{Albums}) - \pi_{\text{AlbumID}}(\text{Album\_Publishers})$

2. The songs in the albums in 1 are what we need
   b. Get the songs in unpublished_albums

$\pi_{\text{SongID, Title}}(\text{Songs ? Unpublished\_Albums})$

What should we use in ? — Equijoin!

Join

- Way to combine information from two tables with correlation
- Conditional join
  - RA: Relation1 $\bowtie_{condition}$ Relation2
  - Equivalent to $\sigma_{condition}$(Relation1 X Relation2) => selection from cross product
- Equijoin is a conditional join with restrictions on condition to only involve equalities
- Natural join (without specifying condition)
  - Relation1 $\bowtie$ Relation2
  - Equijoin but automatic on all columns with the same name (must be same type)
  - Duplicate columns are dropped

Fakeify (RA Q12)

Artists(ArtistID, Name, Age)

Albums(AlbumID, Title, Year)

Album_Publishers(**AlbumID**, **ArtistID**)

Songs(SongID, **AlbumID**, Title)

Playlists(PlaylistID, Title)

Entries(**PlaylistID**, **SongID**, Description)

- Albums in *Albums* but not in *Album_Publishers* are unpublished
- Do not return duplicates

Q: Return the IDs and titles of songs that are in unpublished albums.

Approach 1:

1. Get all unpublished albums

$\pi_{\text{AlbumID}}(\text{Albums}) - \pi_{\text{AlbumID}}(\text{Album\_Publishers})$

2. The songs in the albums in 1 are what we need
    b. Get the songs in unpublished_albums

$\pi_{\text{SongID, Title}}(\text{Songs} \bowtie \text{Unpublished\_Albums})$ or
$\pi_{\text{SongID, Title}}(\text{Songs} \bowtie_{\text{AlbumID}} \text{Unpublished\_Albums})$

Fakeify (RA Q12)

Artists(ArtistID, Name, Age)

Albums(AlbumID, Title, Year)

Album_Publishers(**AlbumID**, **ArtistID**)

Songs(SongID, **AlbumID**, Title)

Playlists(PlaylistID, Title)

Entries(**PlaylistID**, **SongID**, Description)

- Albums in *Albums* but not in *Album_Publishers* are unpublished
- Do not return duplicates

Q: Return the IDs and titles of songs that are in unpublished albums.

Approach 2:

1. Get all the songs in published albums

   Songs ⋈ Album_Publishers

2. The songs in Songs but not in 1 are what we need

   Songs - Songs ⋈ Album_Publishers

   Wrong!

   Conditions for "compatibility" for set operations:

   - Relations have the same number of fields
   - Corresponding fields are of the same datatype

Fakeify (RA Q12)

Artists(ArtistID, Name, Age)

Albums(AlbumID, Title, Year)

Album_Publishers(**AlbumID**, **ArtistID**)

Songs(SongID, **AlbumID**, Title)

Playlists(PlaylistID, Title)

Entries(**PlaylistID**, **SongID**, Description)

- Albums in *Albums* but not in *Album_Publishers* are unpublished
- Do not return duplicates

Q: Return the IDs and titles of songs that are in unpublished albums.

Approach 2:

1. Get all the songs in published albums

   Songs $\bowtie$ Album_Publishers

2. The songs in Songs but not in 1 are what we need

   ~~Songs - Songs $\bowtie$ Album_Publishers~~

   $\pi_{\text{SongID, Title}}$(Songs) - $\pi_{\text{SongID, Title}}$(Songs $\bowtie$ Album_Publishers)

Fakeify (SQL Q15)

Artists(ArtistID, Name, Age)

Albums(AlbumID, Title, Year)

Album_Publishers(**AlbumID**, **ArtistID**)

Songs(SongID, **AlbumID**, Title)

Playlists(PlaylistID, Title)

Entries(**PlaylistID**, **SongID**, Description)

- Playlists in *Playlists* but not in *Entries* do not have any songs
- Do not return duplicates

Q: Return the titles of playlists with less than 3 songs titled ‘Let Me Love You’.

Things to consider:

- We are looking for *titles* and not *IDs*
- Playlists have Entries with Songs
- Our songs must be titled ‘Let Me Love You’
- We should group by playlists and count number of relevant songs for each

Fakeify (SQL Q15 Take 1)

Songs(SongID, **AlbumID**, Title)

Playlists(PlaylistID, Title)

Entries(**PlaylistID**, **SongID**, Description)

- Playlists in *Playlists* but not in *Entries* do not have any songs
- Do not return duplicates

Q: Return the titles of playlists with less than 3 songs titled ‘Let Me Love You’.

Things to consider:

- We are looking for *titles* and not *IDs*
- Playlists have Entries with Songs
- Our songs must be titled ‘Let Me Love You’
- We should group by playlists and count number of relevant songs for each

```
SELECT DISTINCT P.Title
    FROM Playlists P
    JOIN Entries E ON P.PlaylistID = E.PlaylistID
    JOIN Songs S ON E.SongID = S.SongID
    WHERE S.Title = ‘Let Me Love You’
    GROUP BY E.PlaylistID
    HAVING COUNT(*) < 3;
```

ORA-00979: Not a GROUP BY expression

Fakeify (SQL Q15 Take 2)

Songs(SongID, **AlbumID**, Title)

Playlists(PlaylistID, Title)

Entries(**PlaylistID**, **SongID**, Description)

- Playlists in *Playlists* but not in *Entries* do not have any songs
- Do not return duplicates

Q: Return the titles of playlists with less than 3 songs titled ‘Let Me Love You’.

Things to consider:

- We are looking for *titles* and not *IDs*
- Playlists have Entries with Songs
- Our songs must be titled ‘Let Me Love You’
- We should group by playlists and count number of relevant songs for each

```
SELECT DISTINCT P.Title
From Playlists P
WHERE P.PlaylistID IN (
        SELECT E.PlaylistID
        FROM Entries E
        JOIN Songs S ON E.SongID = S.SongID
        WHERE S.Title = ‘Let Me Love You’
        GROUP BY E.PlaylistID
        HAVING COUNT(*) < 3);
```

We are not including playlists with 0 songs. HAVING COUNT(*) < 3 is equivalent to HAVING (COUNT(*) = 1 OR COUNT(*) = 2).

Fakeify (SQL Q15 Take 3 Final Solution)

Songs(SongID, **AlbumID**, Title)

Playlists(PlaylistID, Title)

Entries(**PlaylistID**, **SongID**, Description)

- Playlists in *Playlists* but not in *Entries* do not have any songs
- Do not return duplicates

Q: Return the titles of playlists with less than 3 songs titled ‘Let Me Love You’.

Things to consider:

- We are looking for *titles* and not *IDs*
- Playlists have Entries with Songs
- Our songs must be titled ‘Let Me Love You’
- We should group by playlists and count number of relevant songs for each

```
SELECT DISTINCT P.Title
From Playlists P
WHERE P.PlaylistID NOT IN (
        SELECT E.PlaylistID
        FROM Entries E
        JOIN Songs S ON E.SongID = S.SongID
        WHERE S.Title = ‘Let Me Love You’
        GROUP BY E.PlaylistID
        HAVING COUNT(*) >= 3);
```

Fakeify (SQL Q15 Alternative Solutions)

To fix the not a GROUP BY expression error, we can

- GROUP BY PlaylistID in a subquery
- add Title to the GROUP BY expression
- GROUP BY PlaylistID and use a dummy aggregate function (e.g. MAX) on Title
- use a correlated subquery

To consider playlists with 0 songs (and playlists with 0 'Let Me Love You' songs and at least 1 other song), we can

- compute the complement (NOT IN or MINUS)
- UNION two SELECT statements
- apply LEFT JOINs and COUNT(SongID)
- use a correlated subquery

Any solution that addresses these two ideas should be (close to) correct.

Fakeify RC

Artists(ArtistID, Name, Age)

Albums(AlbumID, Title, Year)

Album_Publishers(**AlbumID**, **ArtistID**)

Songs(SongID, **AlbumID**, Title)

Playlists(PlaylistID, Title)

Entries(**PlaylistID**, **SongID**, Description)

Q: Translate the following RC to everyday English:

{T | T ϵ Playlists Λ ∀ A ϵ Artists ∃ P ϵ Album_Publishers ∃ S ϵ Songs ∃ E ϵ Entries (A.ArtistID = P.ArtistID Λ P.AlbumID = S.AlbumID Λ S.SongID = E.SongID Λ E.PlaylistID = T.PlaylistID)

Fakeify RC

Artists(ArtistID, Name, Age)

Albums(AlbumID, Title, Year)

Album_Publishers(**AlbumID**, **ArtistID**)

Songs(SongID, **AlbumID**, Title)

Playlists(PlaylistID, Title)

Entries(**PlaylistID**, **SongID**, Description)

{T | T ϵ Playlists ∧ ∀ A ϵ Artists ∃ P ϵ Album_Publishers ∃ S ϵ Songs ∃ E ϵ Entries (A.ArtistID = P.ArtistID ∧ P.AlbumID = S.AlbumID ∧ S.SongID = E.SongID ∧ E.PlaylistID = T.PlaylistID)

Joins

we just ID, and also title

1. T ϵ Playlists => T is a playlist.

   (The result set is a set of playlists)

2. A ϵ Artists => A is an artist
3. P ϵ Album_Publishers => P is a record of a published album
4. S ϵ Songs => S is a song
5. E ϵ Entries => E is an entry in a playlist

Fakeify RC

{T | T ϵ Playlists Λ ∀ A ϵ Artists ∃ P ϵ Album_Publishers ∃ S ϵ Songs ∃ E ϵ Entries (A.ArtistID = P.ArtistID Λ P.AlbumID = S.AlbumID Λ S.SongID = E.SongID Λ E.PlaylistID = T.PlaylistID)

1. T is a playlist
2. A is an artist
3. P is a record of a published album
4. S is a song
5. E is an entry in a playlist

1. A.ArtistID = P.ArtistID => A published an album specified in P
2. P.AlbumID = S.AlbumID => S is in the album specified by P
3. S.SongID = E.SongID => S is the song specified in E
4. E.PlaylistID = T.PlaylistID => T is the playlist where E is

Next step: combine them all and take the quantifier into account!

Fakeify RC

{T | T ϵ Playlists Λ ∀ A ϵ Artists ∃ P ϵ Album_Publishers ∃ S ϵ Songs ∃ E ϵ Entries (A.ArtistID = P.ArtistID Λ P.AlbumID = S.AlbumID Λ S.SongID = E.SongID Λ E.PlaylistID = T.PlaylistID)

1. T is a playlist
2. A is an artist
3. P is a record of a published album
4. S is a song
5. E is an entry in a playlist
6. A.ArtistID = P.ArtistID => A published an album specified in P
7. P.AlbumID = S.AlbumID => S is in the album specified by P
8. S.SongID = E.SongID => S is the song specified in E
9. E.PlaylistID = T.PlaylistID => T is the playlist where E is

Next step: combine them all and take the quantifier into account!

For every artist A in Artists, there exists an album published by A, a song S in that album and an entry which says that S is in playlist T.

=> Further translation:

All playlists that include at least one song in a published album by every artist in Artists.

Responsible data management Q19

e.g. avg: add random error

Select all that apply.

a. Differential privacy means that each record is processed differently, depending on its value.

b. Differential privacy provides different privacy guarantees depending on the query posed. ✗

c. Differential privacy only applies to aggregate queries.

d. Exact answers cannot be obtained with differential privacy.

e. All of the above.

NoSQL Q22

Select all that apply. Which of these are benefits that NoSQL databases provide compared to SQL databases?

a. Fewer joins between entities are required when querying data
b. They are more suited for scale out (i.e. using commodity hardware)
c. Transactions with stronger consistency guarantees
d. Sharding allows data to be distributed across multiple machines
e. All of the above

(some even doesn't support)

do sharding (horizontal scaling)

traditional can: but need human effort

now it can done by database

Get started on P3!