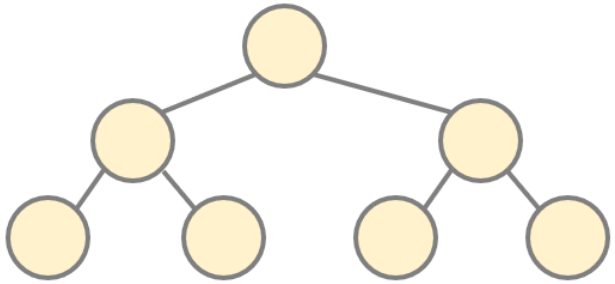
External Memory
(Algorithms and Data Structures)

File Systems and Databases



balanced search trees are working extremely fine and they can be stored in the ***main memory*** *(RAM)*

WHAT IF WE WANT TO STORE > 1GB HUGE DATA?

File Systems and Databases

There are **2** types of memory:

1.) **main memory (RAM):** all the **data structures** considered so far are stored in the main memory

→ **stack memory** and **heap memory** are located in the main memory as well

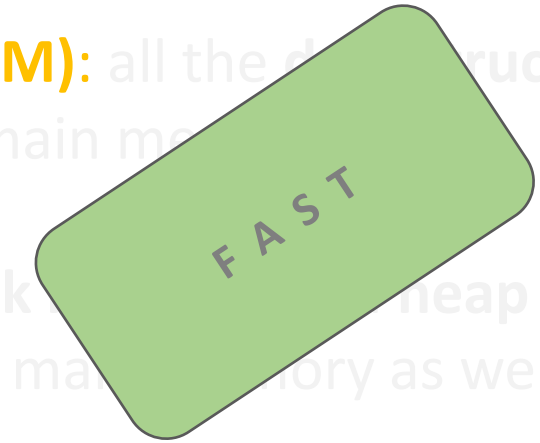
2.) **external memory** (peripheral memory): **hard disk**, CD-ROM etc.

→ hard drive storage can store large amounts and sizes of files such as **file systems** or **databases**

File Systems and Databases

There are **2** types of memory:

1.) main memory (RAM): all the **data structures** considered so far are stored in the main memory



→ **stack** and **heap memory** are located in the main memory as well

2.) external memory (peripheral memory): **hard disk**, CD-ROM etc.

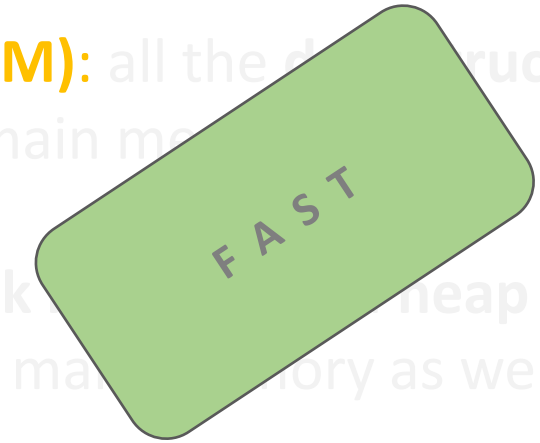
→ hard drive storage can store large amounts and sizes of files such as **file systems** or **databases**

File Systems and Databases

There are **2** types of memory:

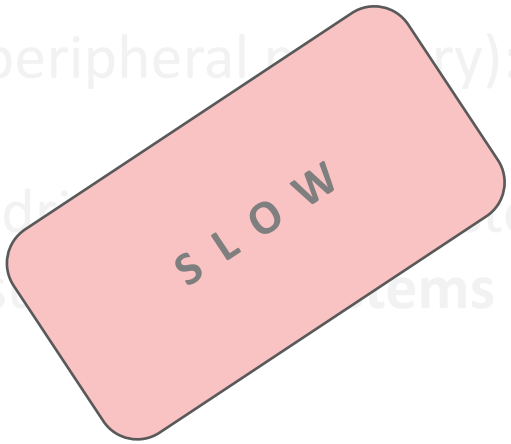
1.) **main memory (RAM)**: all the **data structures** considered so far are stored in the main memory

→ **stack** and **heap memory** are located in the main memory as well



2.) **external memory** (peripheral memory): **hard disk**, CD-ROM etc.

→ hard drives store large amounts and sizes of files such as **file systems** or **databases**

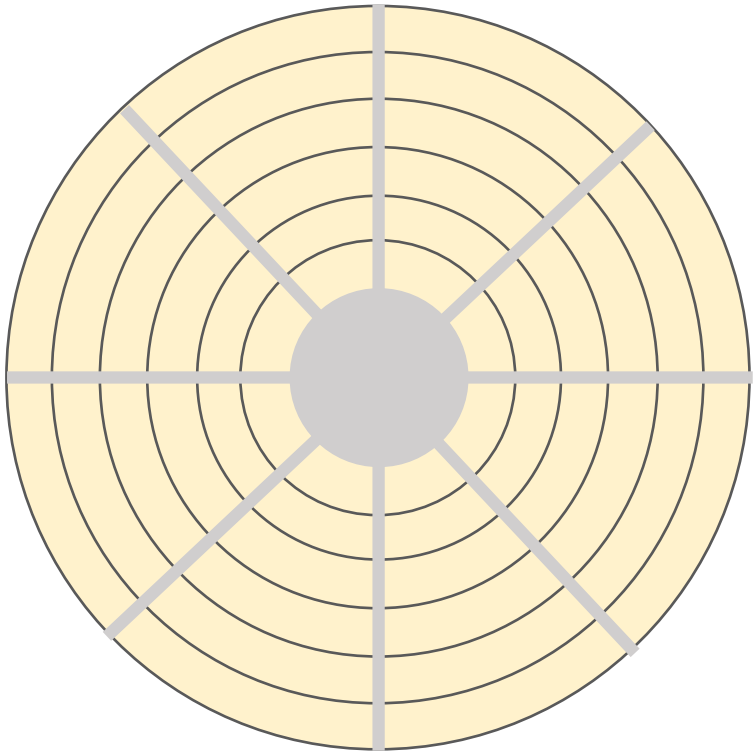


File Systems and Databases

EXTERNAL MEMORY

hard drive disk (HDD) *is* one or more rigid rapidly rotating platters coated with **magnetic material**

CAN RETAIN DATA EVEN WHEN POWERED OFF !!!

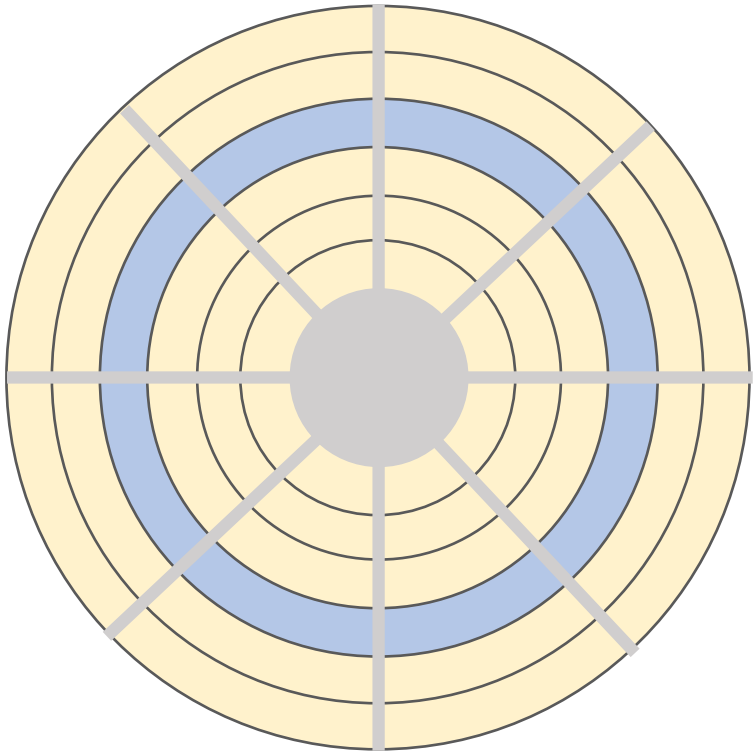


File Systems and Databases

EXTERNAL MEMORY

hard drive disk (HDD) *is* one or more rigid rapidly rotating platters coated with **magnetic material**

CAN RETAIN DATA EVEN WHEN POWERED OFF !!!



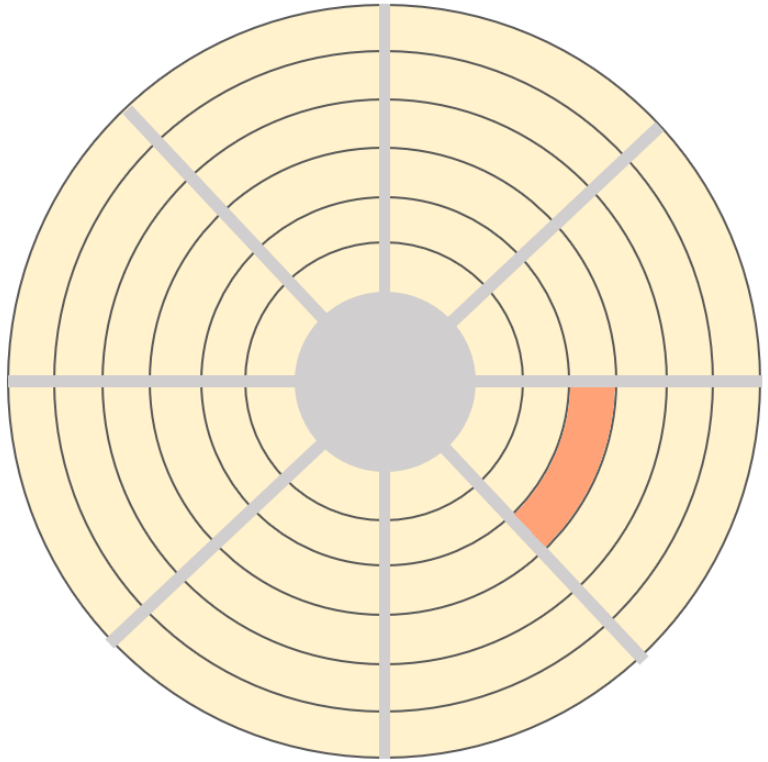
track is a circular path on the surface of the **HDD** on which information is recorded and read

File Systems and Databases

EXTERNAL MEMORY

hard drive disk (HDD) *is* one or more rigid rapidly rotating platters coated with **magnetic material**

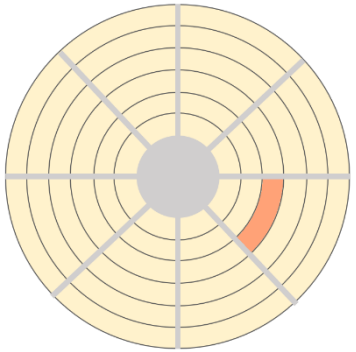
CAN RETAIN DATA EVEN WHEN POWERED OFF !!!



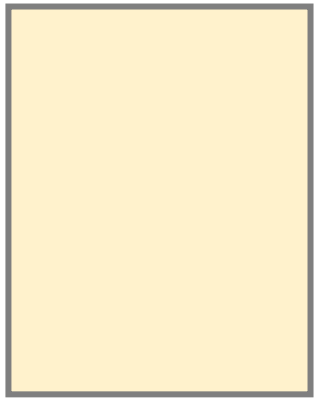
a ***block*** is a subdivision of the hard drive disk (HDD) storing ***512 bytes***

File Systems and Databases

DISK MEMORY (HDD)

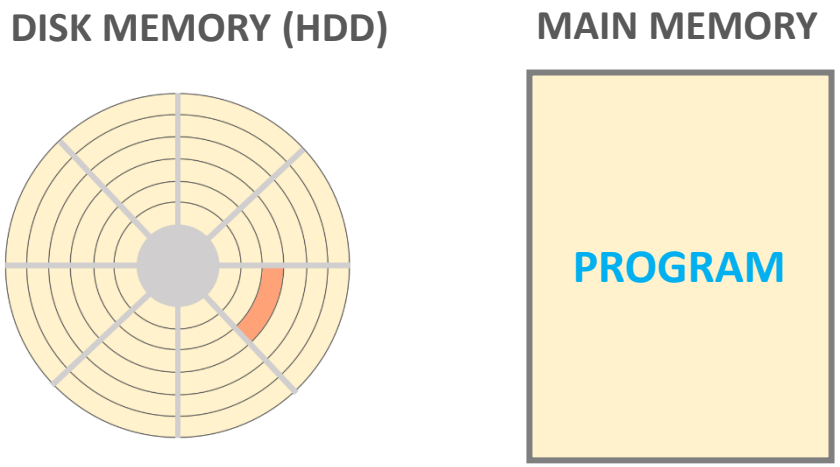


MAIN MEMORY



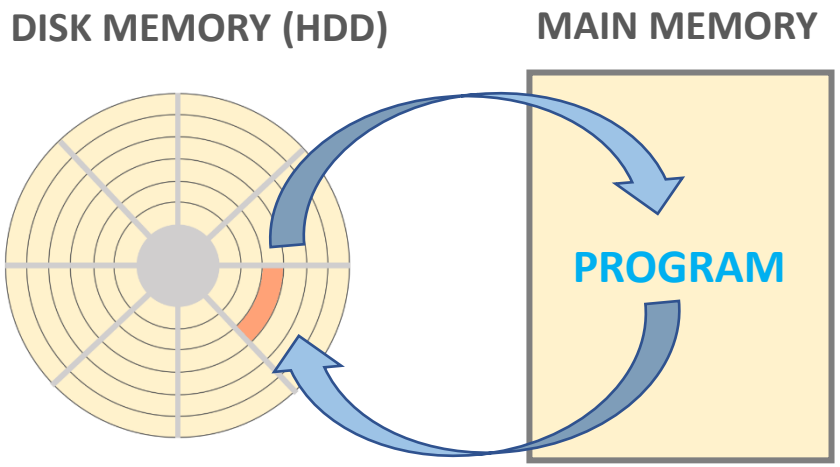
- the data located on the **hard drive disk** (HDD) can not be proccessed explicitly
- it must be brought into the **main memory**
- in the main memory (RAM) we can use either the **stack memory** or the **heap memory**
- we can manipulate (read or write) the blocks which means at least **512 bytes**
- **ACCESSING THE BLOCKS IS SLOW !!!**

File Systems and Databases



- the data located on the **hard drive disk** (HDD) can not be proccessed explicitly
- it must be brought into the **main memory**
- in the main memory (RAM) we can use either the **stack memory** or the **heap memory**
- we can manipulate (read or write) the blocks which means at least **512 bytes**
- **ACCESSING THE BLOCKS IS SLOW !!!**

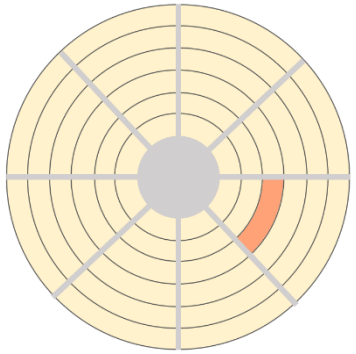
File Systems and Databases



- the data located on the **hard drive disk** (HDD) can not be proccessed explicitly
- it must be brought into the **main memory**
- in the main memory (RAM) we can use either the **stack memory** or the **heap memory**
- we can manipulate (read or write) the blocks which means at least **512 bytes**
- **ACCESSING THE BLOCKS IS SLOW !!!**

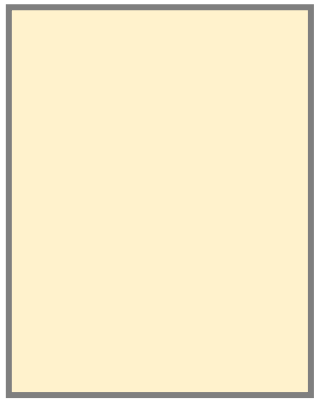
File Systems and Databases

DISK MEMORY (HDD)



*organizing the data efficiently stored on the **hard drive disk (HDD)** has something to do with **database management systems (DBMS)***

MAIN MEMORY



*storing the data efficiently on the main memory **(RAM)** has something to do with **data structures***

External Memory Access Time
(Algorithms and Data Structures)

File Systems and Databases

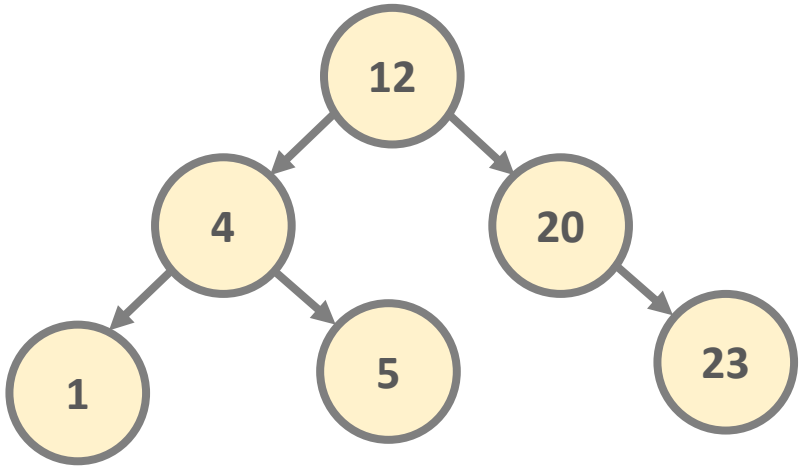
- accessig items on the external memory (HDD) is **way slower** than manipulating the mian memory
- we need totally different data structures

EXTERNAL MEMORY ACCES TIME: 12 ms

RAM ACCESS TIME: 0.0001 ms

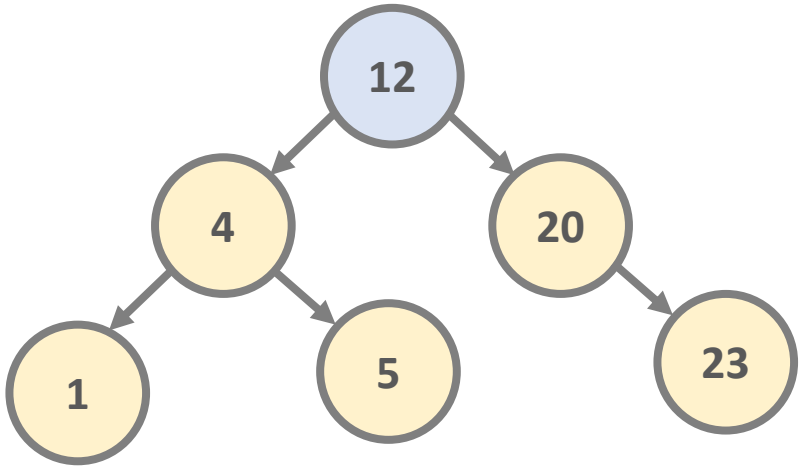
- so far we have manipulated data present on the main memory but now we have to fetch the data from the external memory first

File Systems and Databases



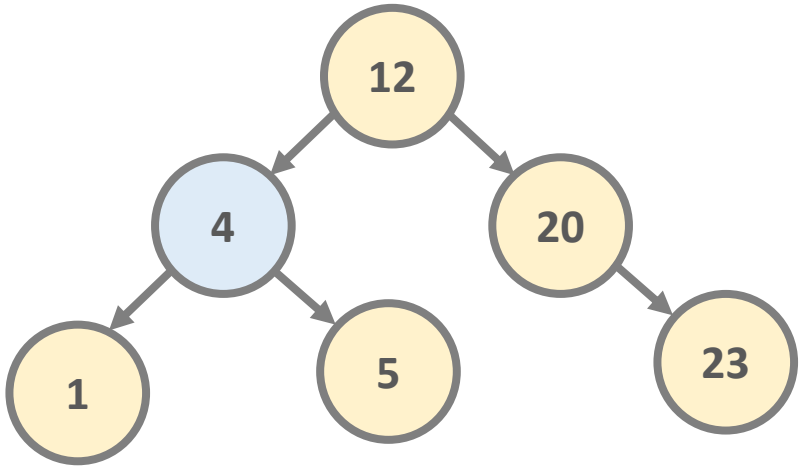
- recursive approaches are working quite fine when using the **main memory** (RAM)
- doing the same on the **external memory** (HDD) is slow because of the access time
- we can manipulate (read or write) the blocks which means at least **512 bytes**
- **ACCESSING THE BLOCKS IS SLOW !!!**
- conclusion: we should **minimize the amount of read operations**
- this is why **B-trees** are shallow structures

File Systems and Databases



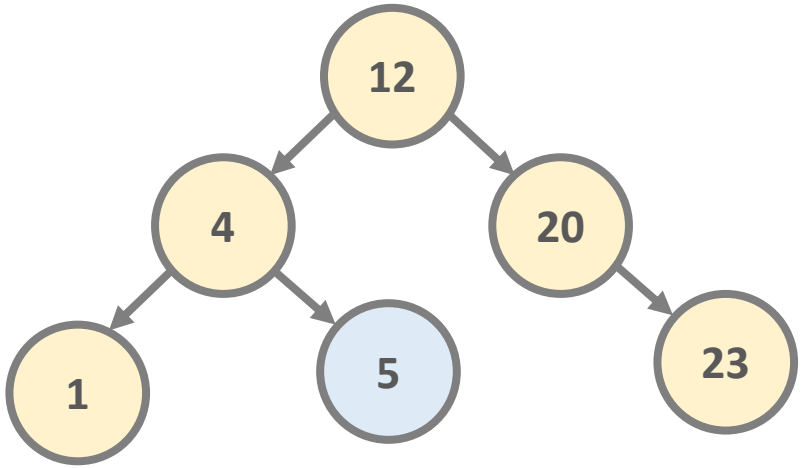
- recursive approaches are working quite fine when using the **main memory** (RAM)
- doing the same on the **external memory** (HDD) is slow because of the access time
- we can manipulate (read or write) the blocks which means at least **512 bytes**
- **ACCESSING THE BLOCKS IS SLOW !!!**
- conclusion: we should **minimize the amount of read operations**
- this is why **B-trees** are shallow structures

File Systems and Databases



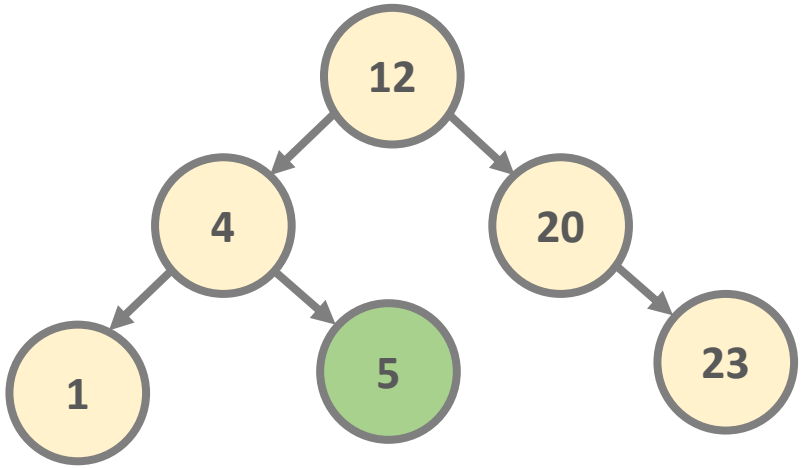
- recursive approaches are working quite fine when using the **main memory** (RAM)
- doing the same on the **external memory** (HDD) is slow because of the access time
- we can manipulate (read or write) the blocks which means at least **512 bytes**
- **ACCESSING THE BLOCKS IS SLOW !!!**
- conclusion: we should **minimize the amount of read operations**
- this is why **B-trees** are shallow structures

File Systems and Databases



- recursive approaches are working quite fine when using the **main memory** (RAM)
- doing the same on the **external memory** (HDD) is slow because of the access time
- we can manipulate (read or write) the blocks which means at least **512 bytes**
- **ACCESSING THE BLOCKS IS SLOW !!!**
- conclusion: we should **minimize the amount of read operations**
- this is why **B-trees** are shallow structures

File Systems and Databases



- recursive approaches are working quite fine when using the **main memory** (RAM)
- doing the same on the **external memory** (HDD) is slow because of the access time
- we can manipulate (read or write) the blocks which means at least **512 bytes**
- **ACCESSING THE BLOCKS IS SLOW !!!**
- conclusion: we should **minimize the amount of read operations**
- this is why **B-trees** are shallow structures

B-Trees

(Algorithms and Data Structures)

B-Trees

- it was first constructed in **1971** by **Rudolf Bayer** and **Ed McCreight**
- B-trees are **self balancing** tree like data structures
- supports operations such as insertion, deletion, sequential access and searching in **O(logN)** time complexity
- the nodes may have more than **2** children + **multipley keys**
- B-tree data sturctures are optimized for systems that **read and write large blocks of data**
- B-trees are a good example of a data structure for external memory
- commonly used in **databases** and **filesystems**

B-Trees

Every node may have multiple children (more than **2**) but the running time is still **O(logN)** logartithmic

$$\log_b N = \frac{\log_a N}{\log_a b}$$

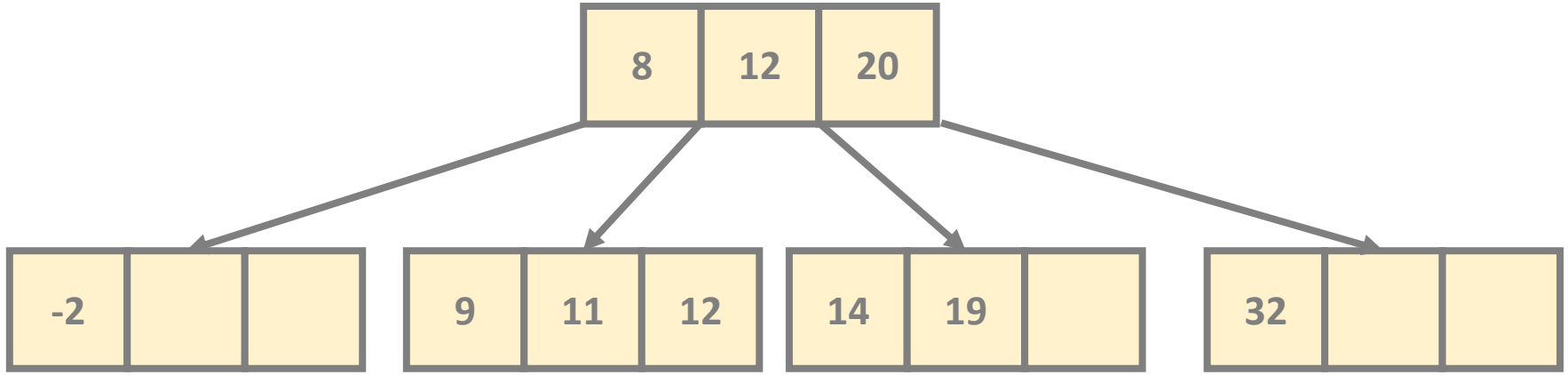
$\log_a b$ *is just a constant so does not matter*

→ we can change the base of the logarithm and the running time complexity for the algorithm stay the same

$$O(c * \log N) = c * O(\log N) = O(\log N)$$

→ thats why the **branching factor** does not matter in the running time complexities

B-Trees



B-Trees

B-TREE PROPERTIES

1.) all the nodes of the tree structure can contain **m** keys – so it may have **m+1** children (branching factor)

2.) every node is at least half full – so contain at least $\frac{\mathbf{m}}{\mathbf{2}}$ items

3.) if the **N** number of items in a node is less than $\frac{\mathbf{m}}{\mathbf{2}}$ then we merge it with another node and if **N>m** then we split the node

4.) all leaf nodes are at the same level (balanced)

B-Trees Insertion

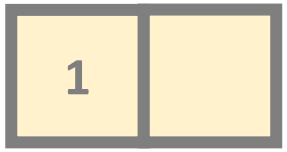
(Algorithms and Data Structures)

B-Trees

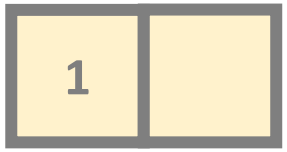
INSERT(1)

B-Trees

INSERT(1)

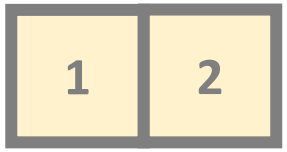


B-Trees

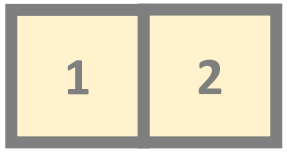


B-Trees

INSERT(2)

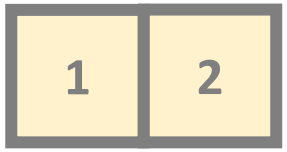


B-Trees



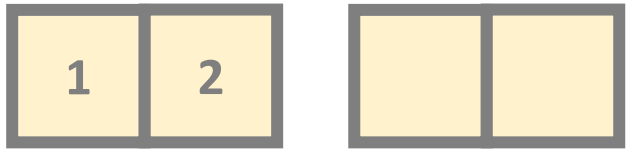
B-Trees

INSERT(3)



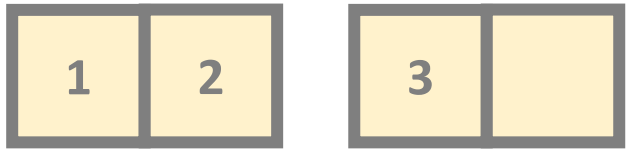
B-Trees

INSERT(3)



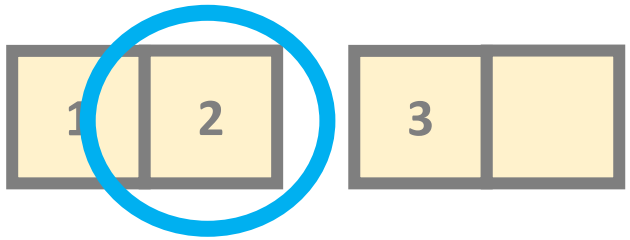
B-Trees

INSERT(3)



B-Trees

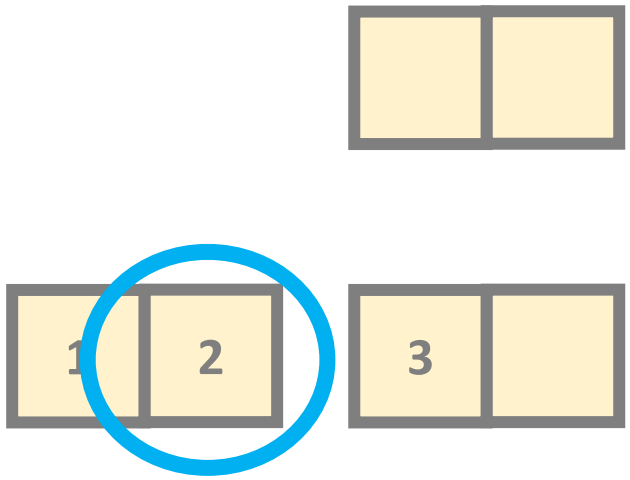
INSERT(3)



we always promote the
middle value *in these cases*

B-Trees

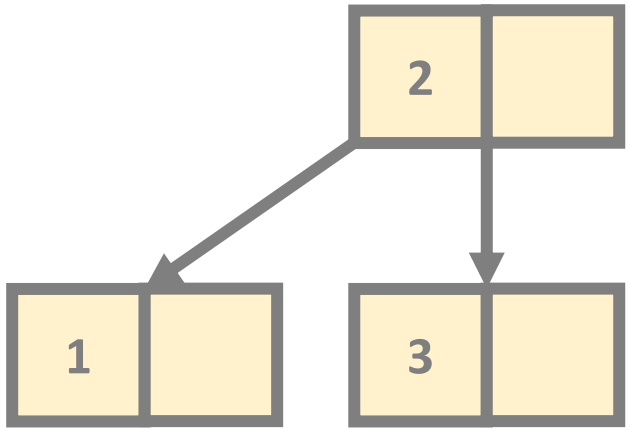
INSERT(3)



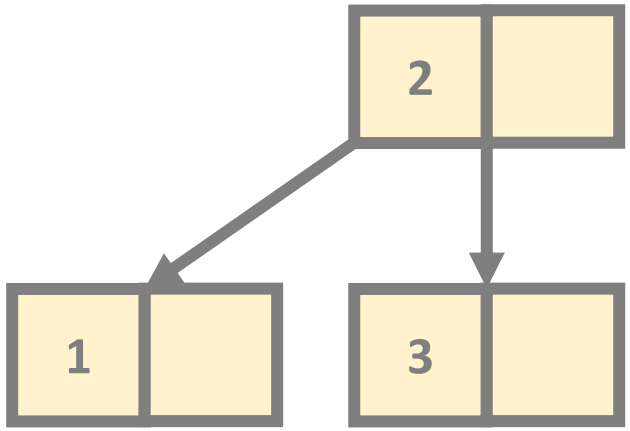
we always promote the
middle value *in these cases*

B-Trees

INSERT(3)

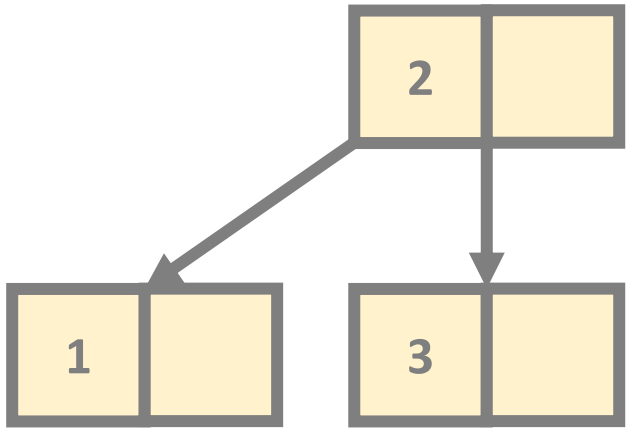


B-Trees



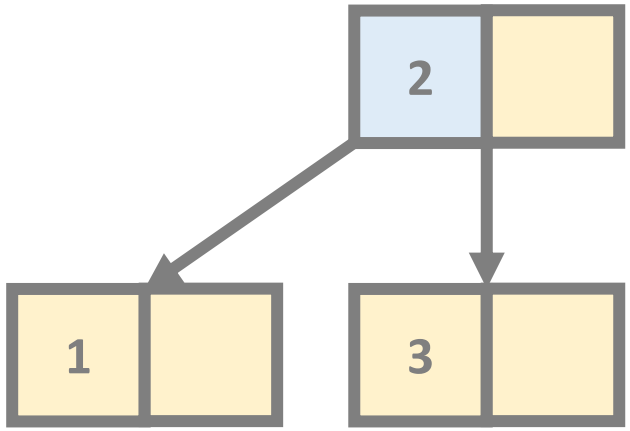
B-Trees

INSERT(4)



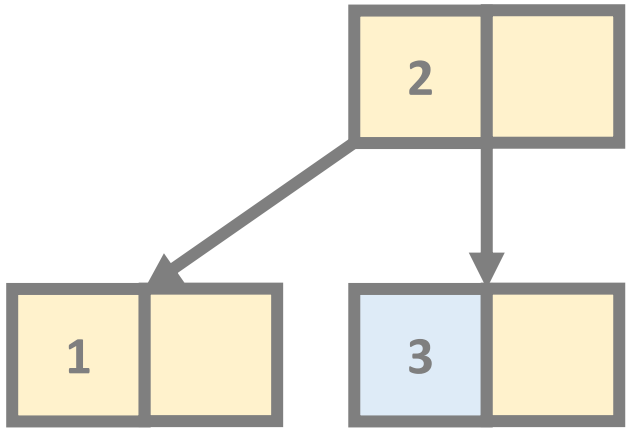
B-Trees

INSERT(4)



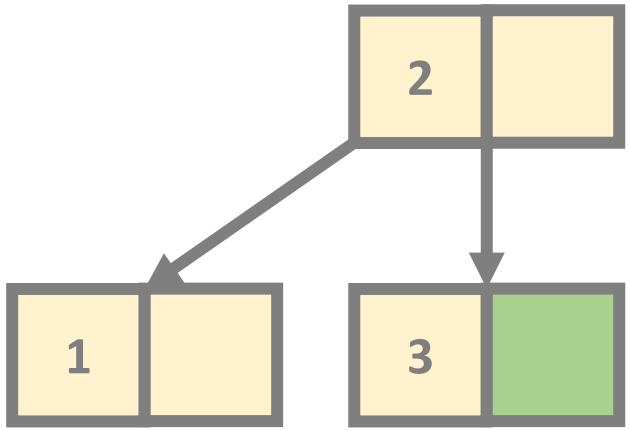
B-Trees

INSERT(4)



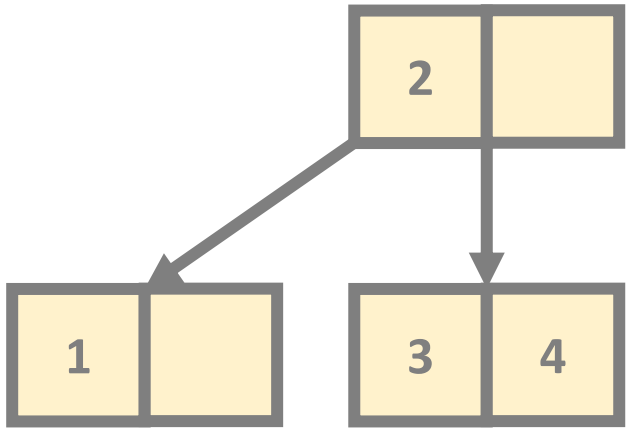
B-Trees

INSERT(4)

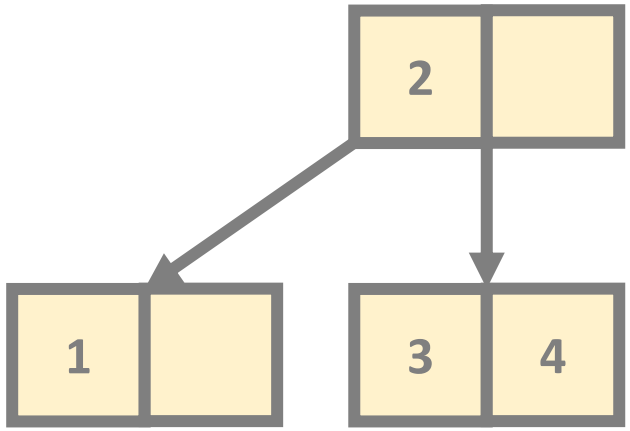


B-Trees

INSERT(4)

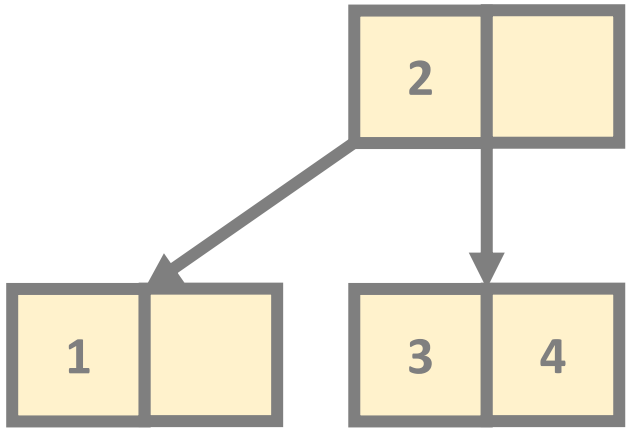


B-Trees



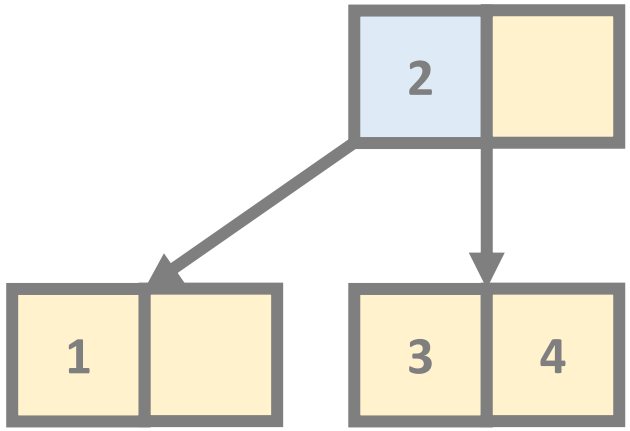
B-Trees

INSERT(5)



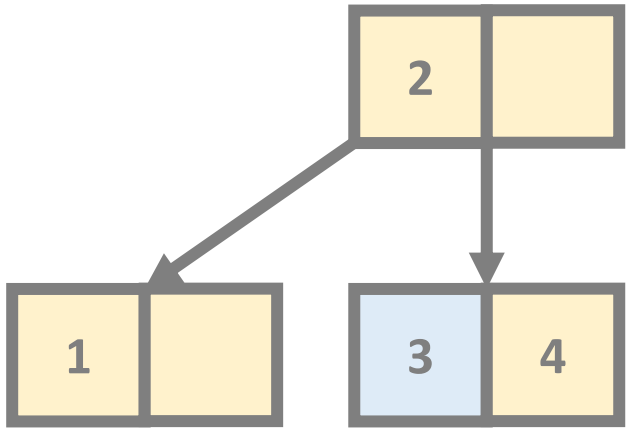
B-Trees

INSERT(5)



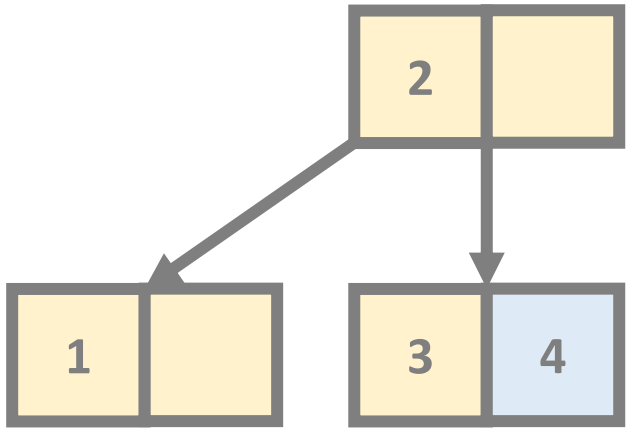
B-Trees

INSERT(5)



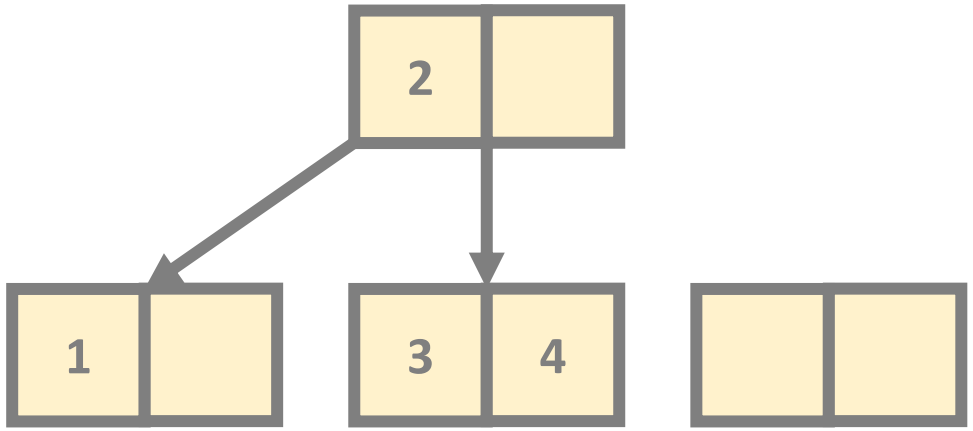
B-Trees

INSERT(5)



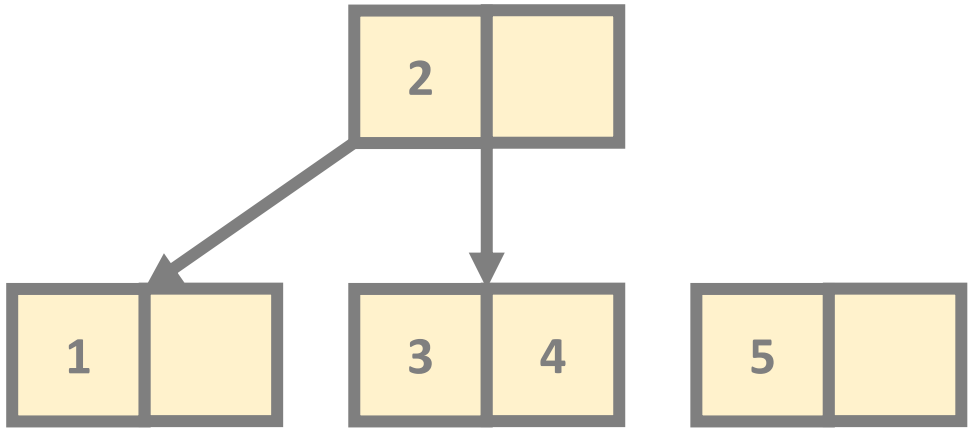
B-Trees

INSERT(5)



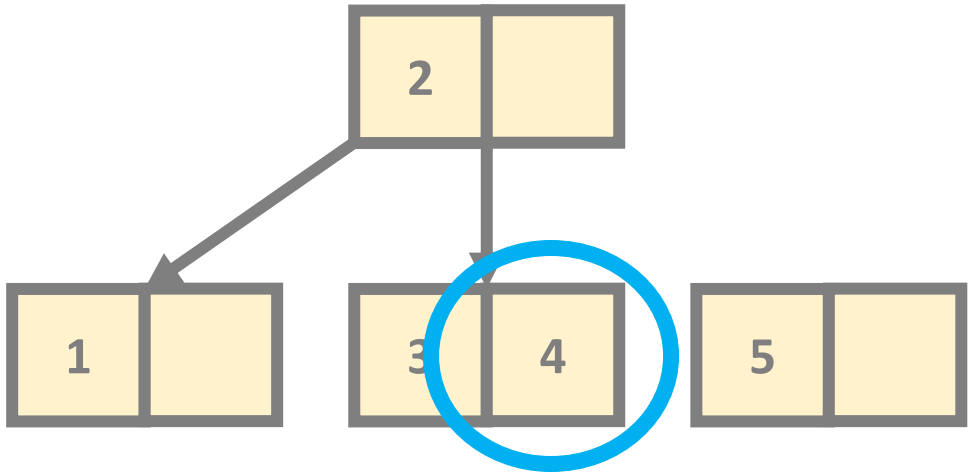
B-Trees

INSERT(5)



B-Trees

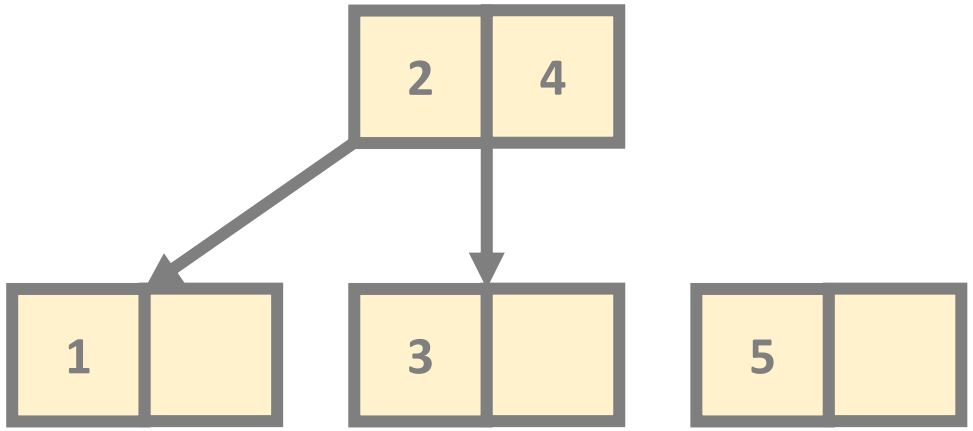
INSERT(5)



we always promote the
middle value *in these cases*

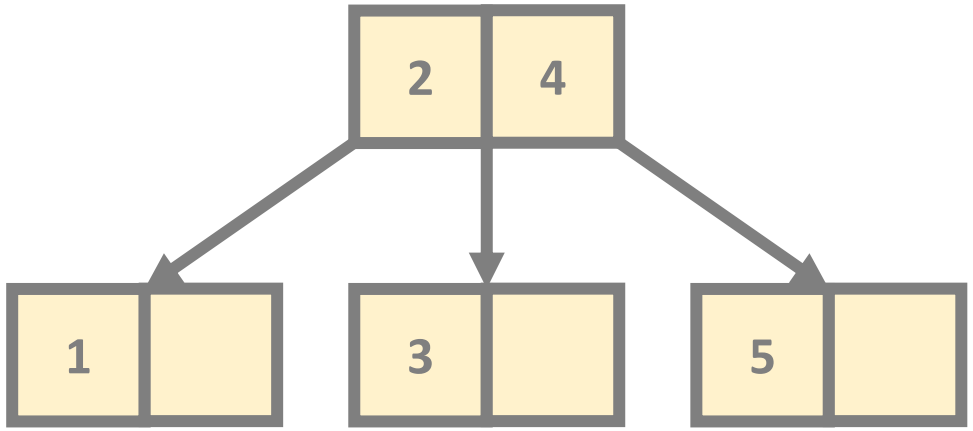
B-Trees

INSERT(5)

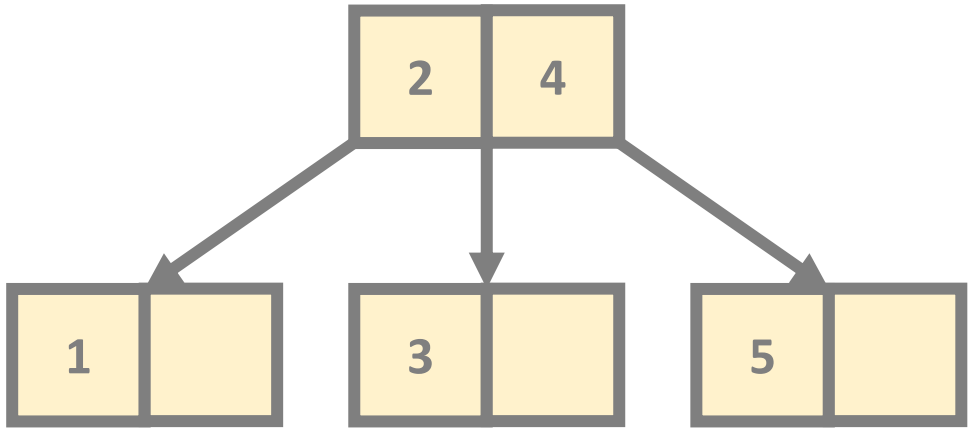


B-Trees

INSERT(5)

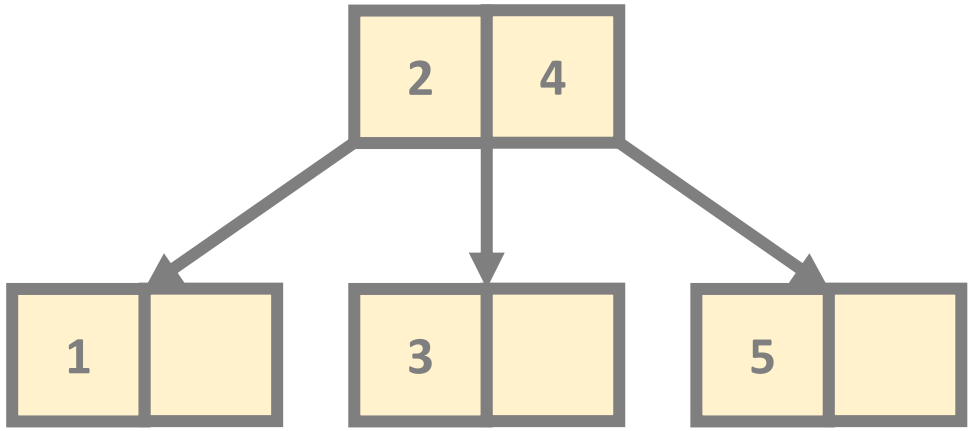


B-Trees



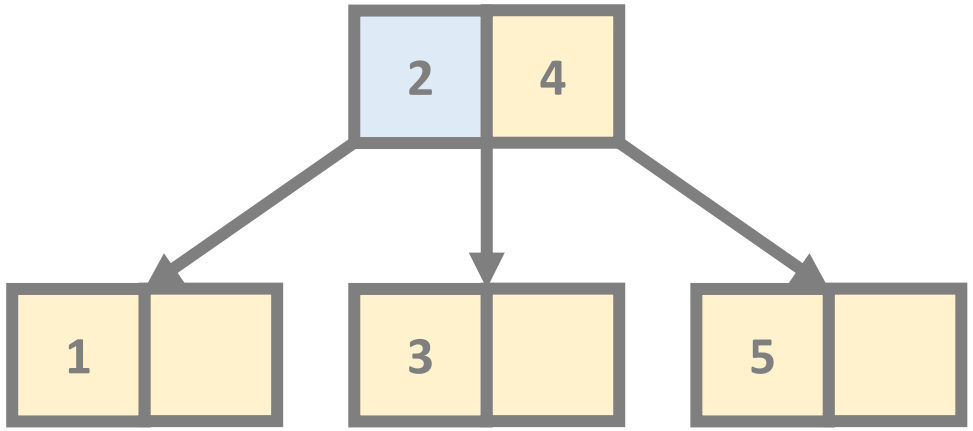
B-Trees

INSERT(6)



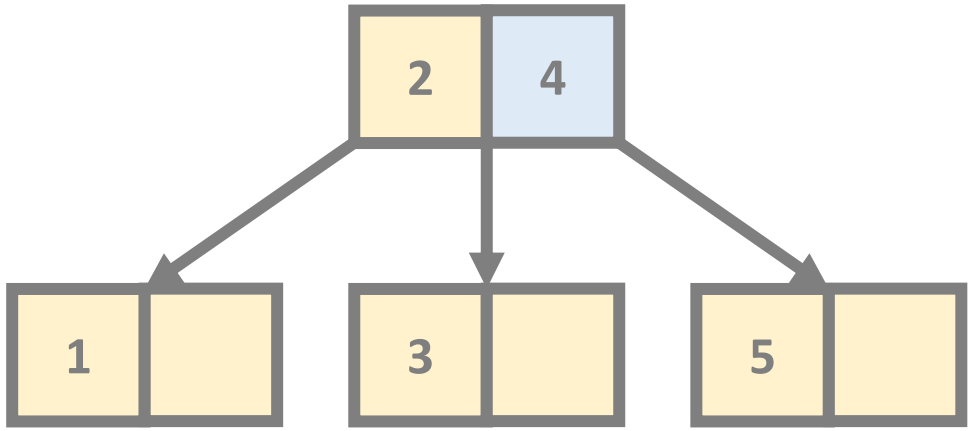
B-Trees

INSERT(6)



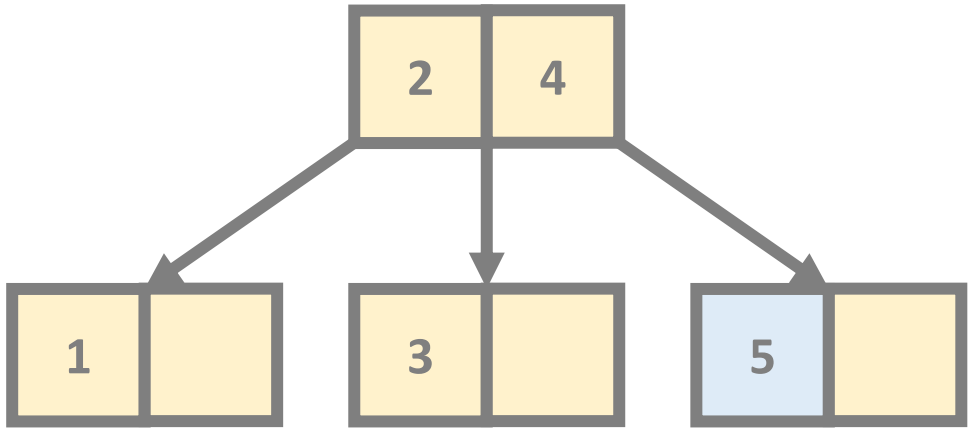
B-Trees

INSERT(6)



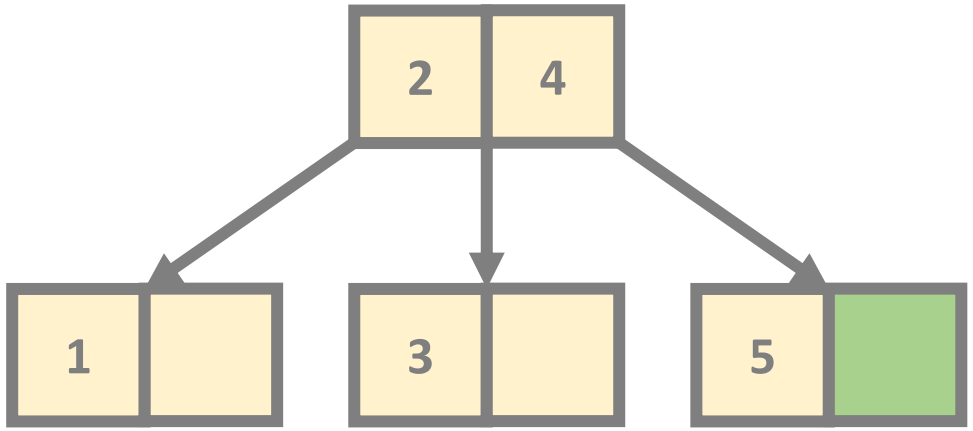
B-Trees

INSERT(6)

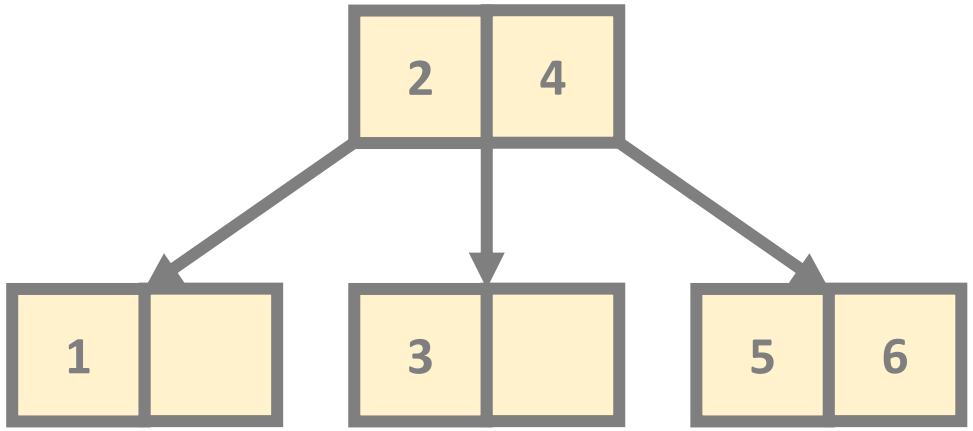


B-Trees

INSERT(6)

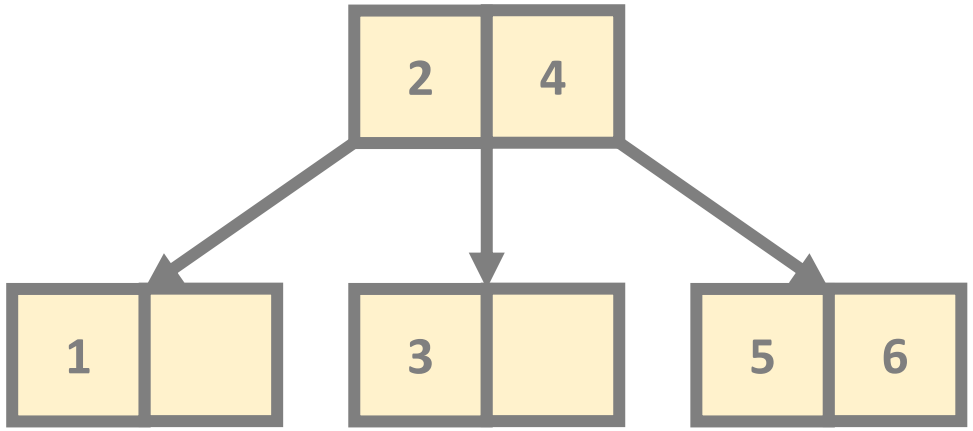


B-Trees



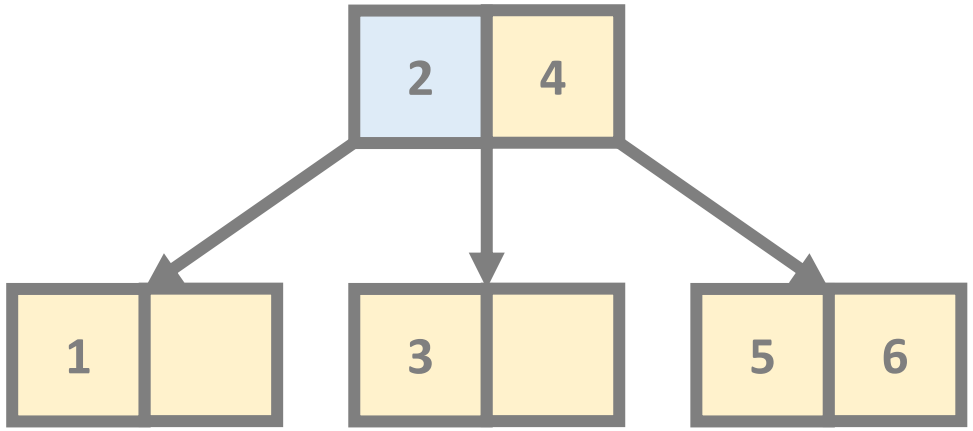
B-Trees

INSERT(7)



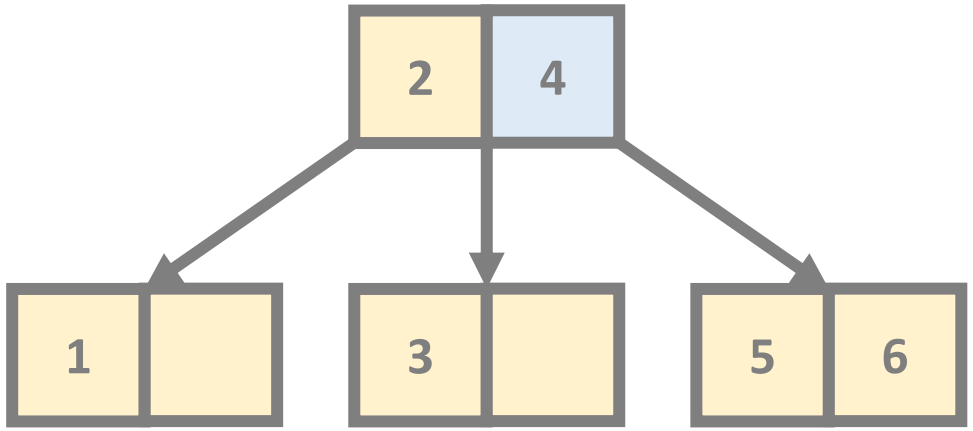
B-Trees

INSERT(7)



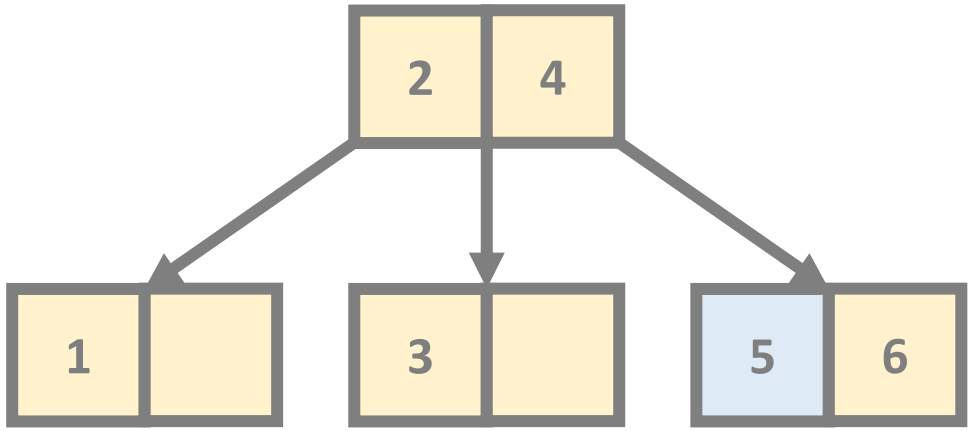
B-Trees

INSERT(7)



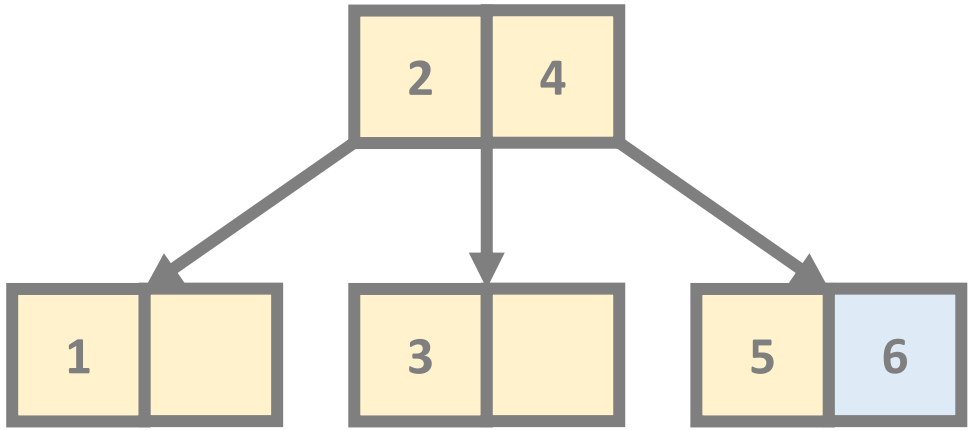
B-Trees

INSERT(7)



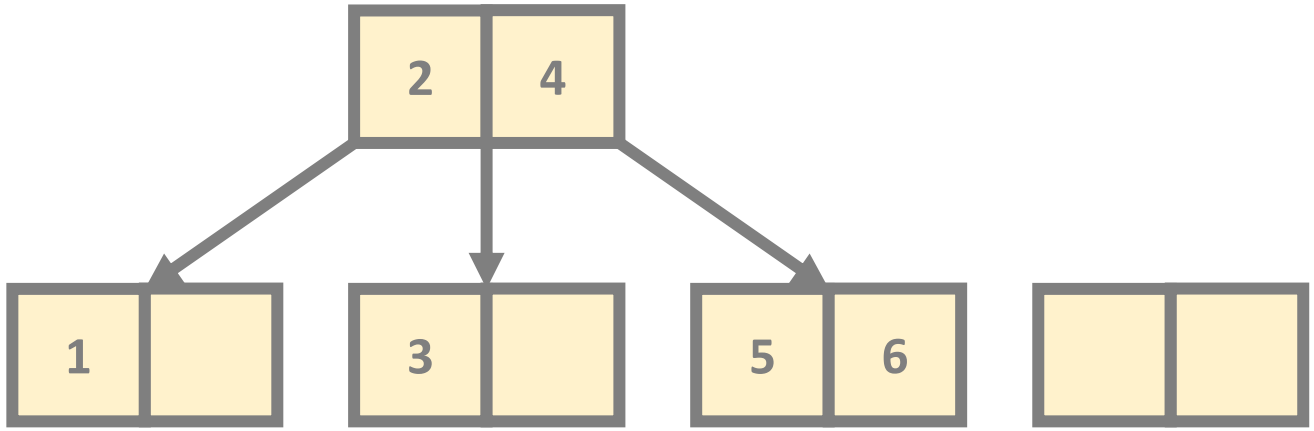
B-Trees

INSERT(7)



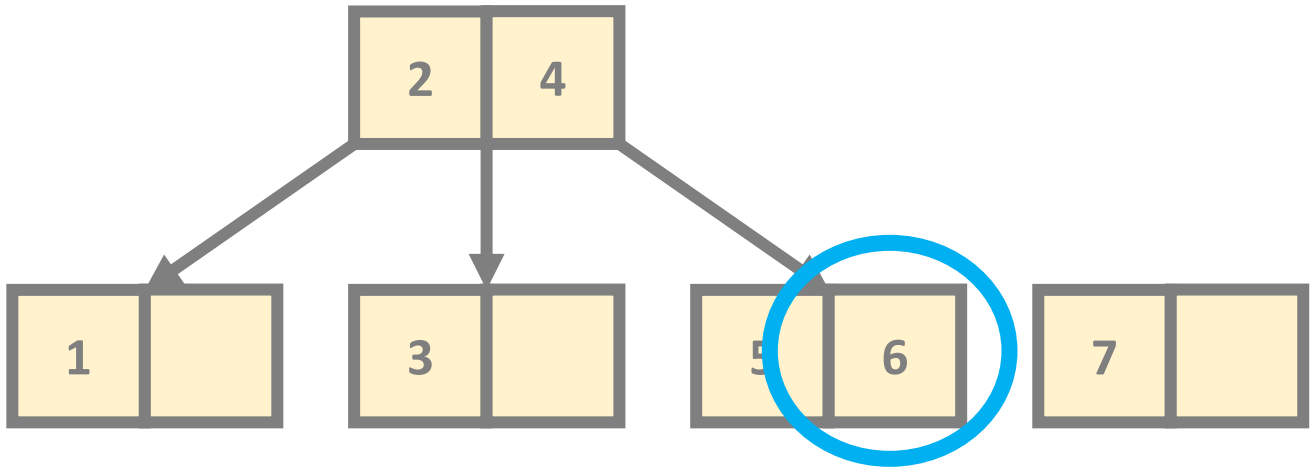
B-Trees

INSERT(7)



B-Trees

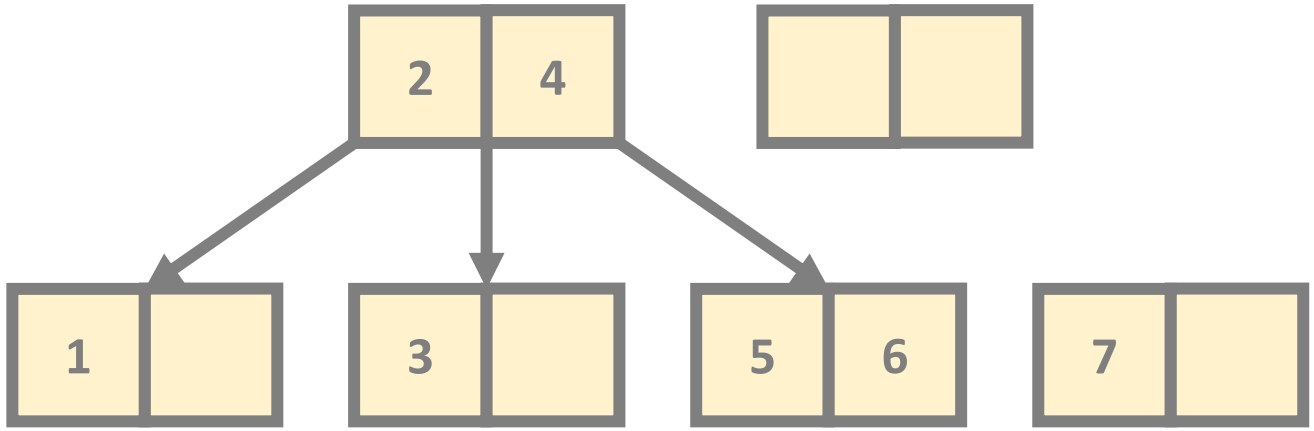
INSERT(7)



we always promote the
middle value *in these cases*

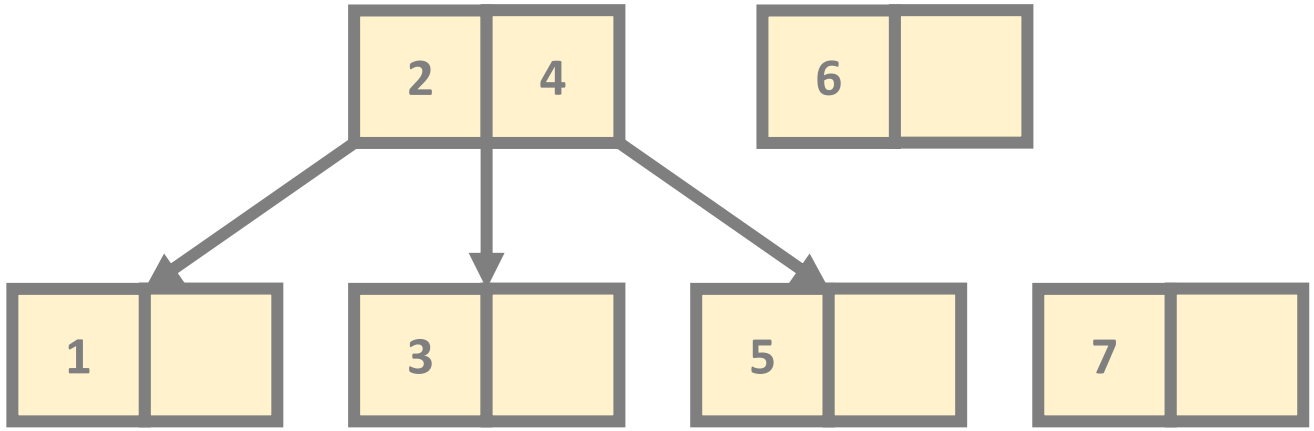
B-Trees

INSERT(7)



B-Trees

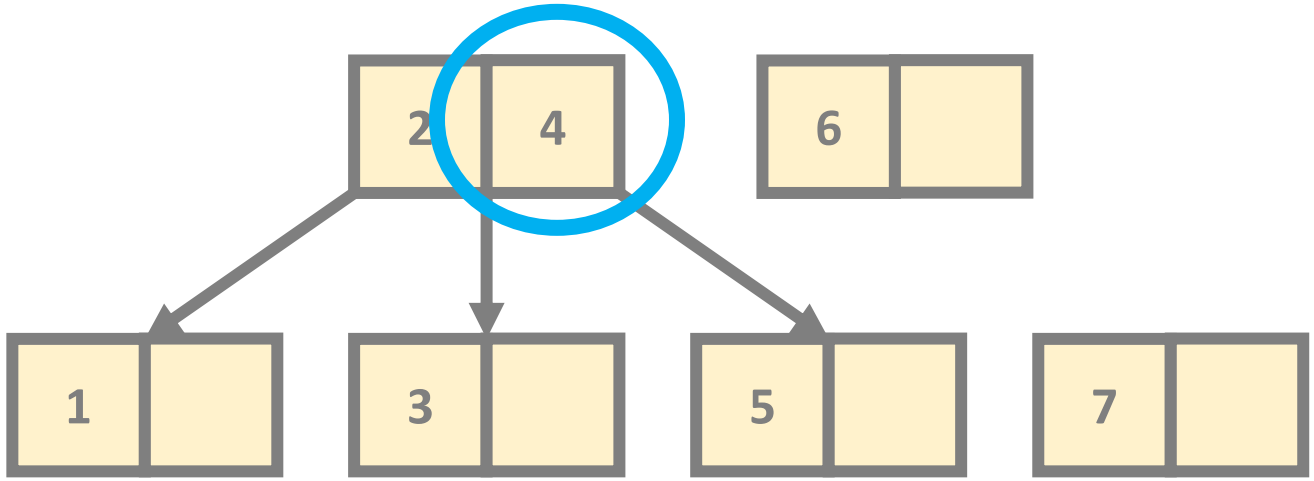
INSERT(7)



B-Trees

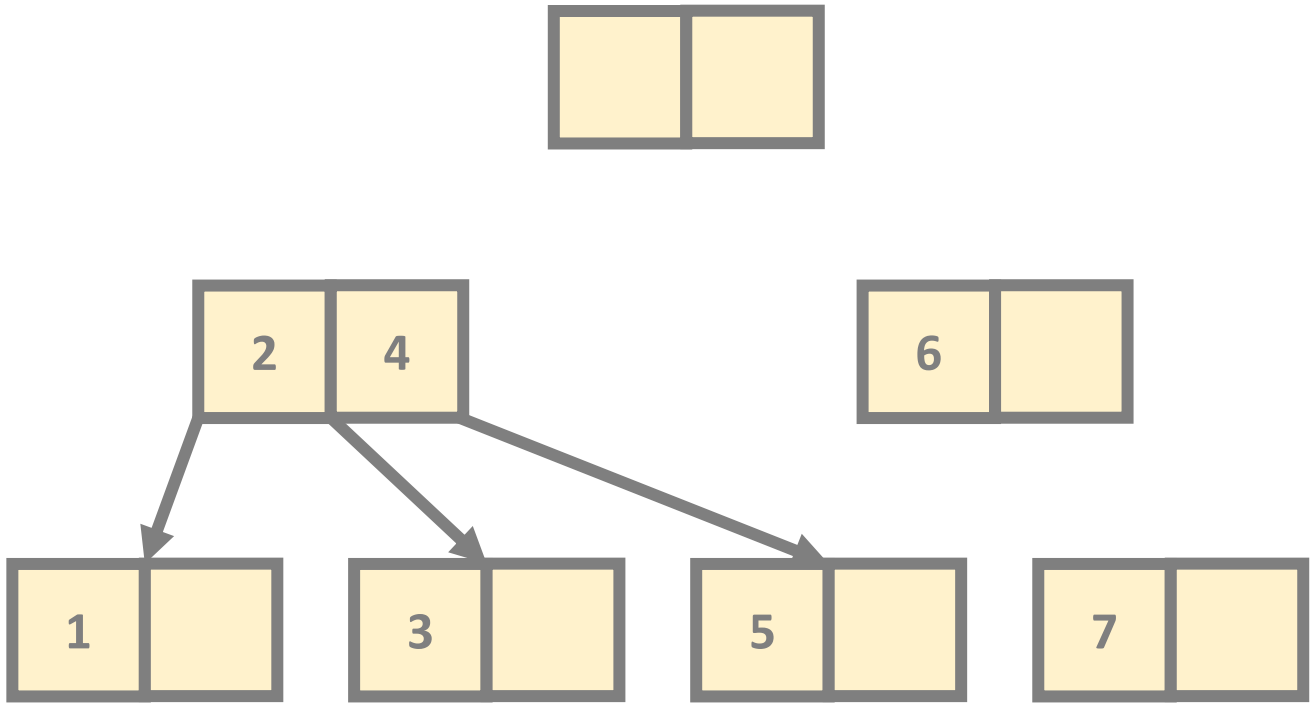
INSERT(7)

we always promote the
middle value *in these cases*



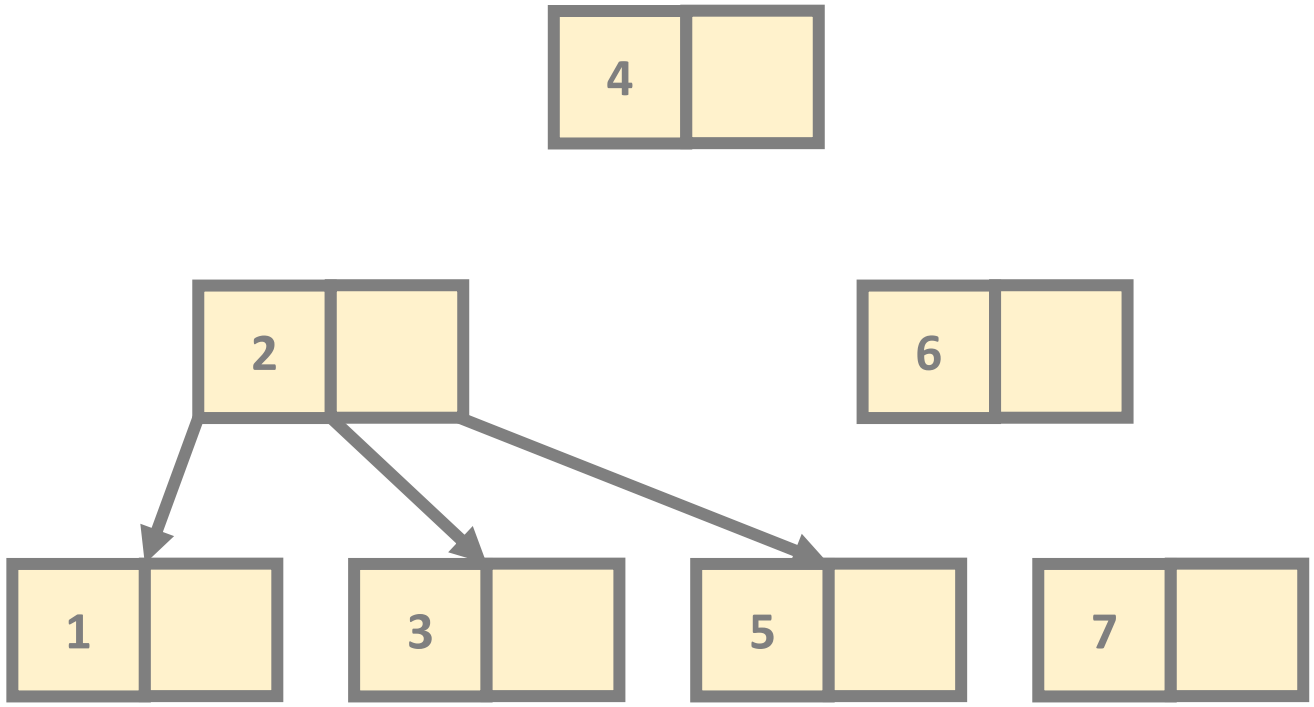
B-Trees

INSERT(7)



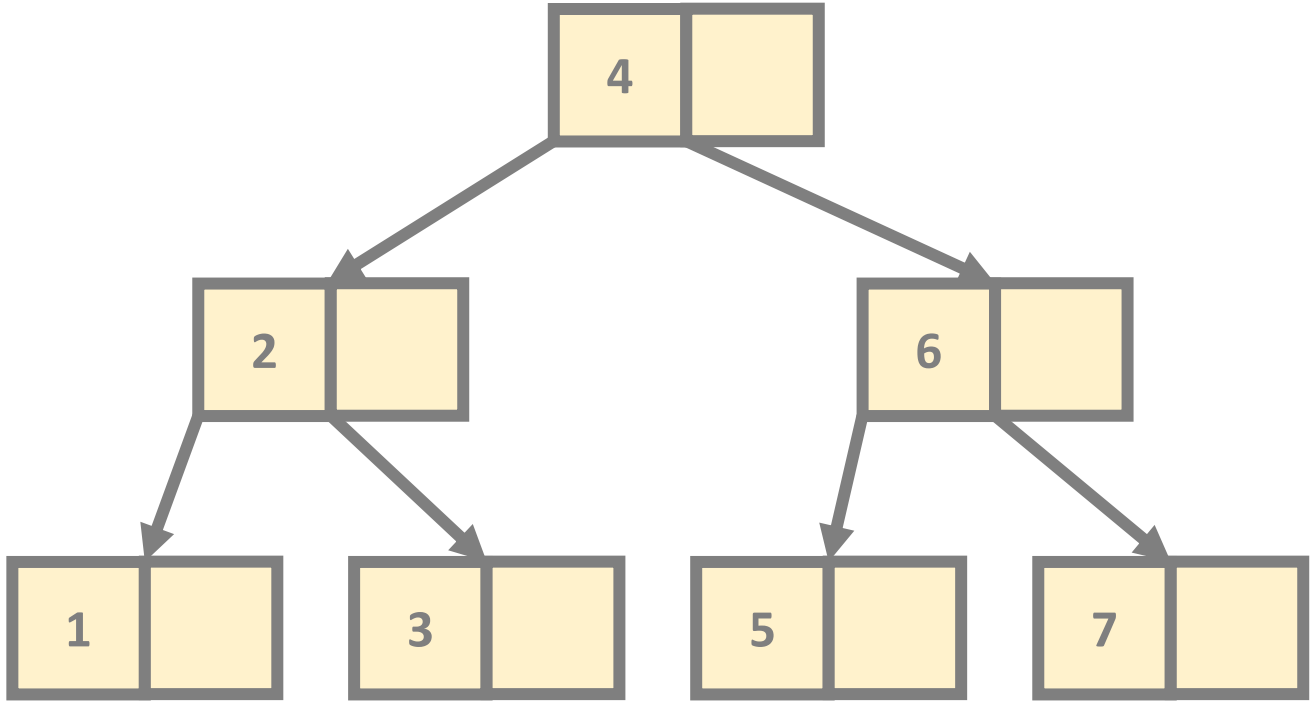
B-Trees

INSERT(7)



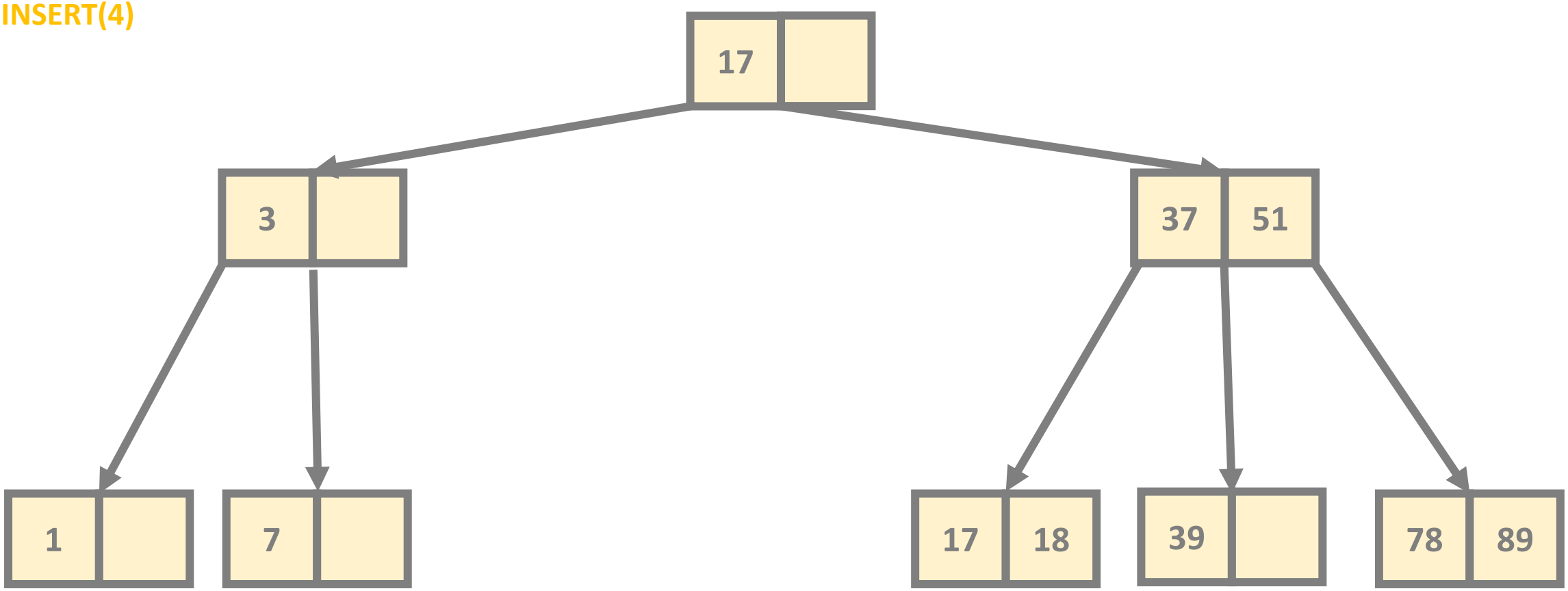
B-Trees

INSERT(7)



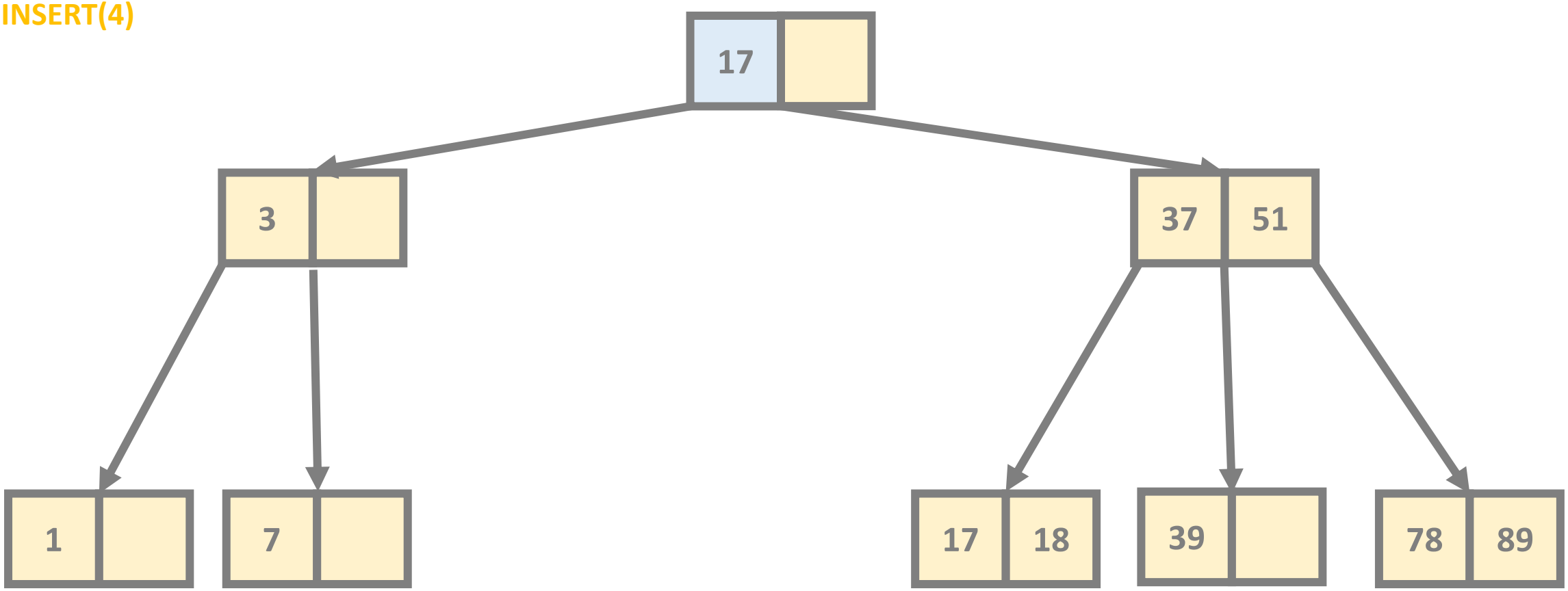
B-Trees

INSERT(4)



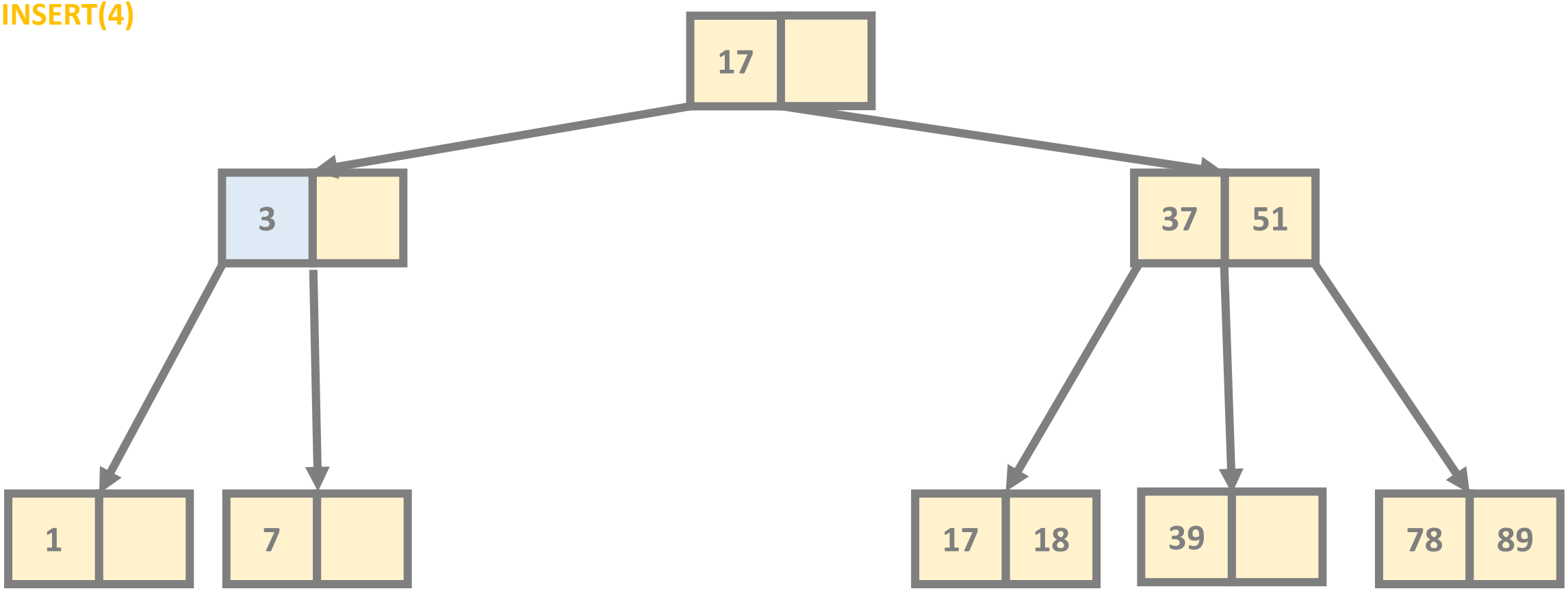
B-Trees

INSERT(4)



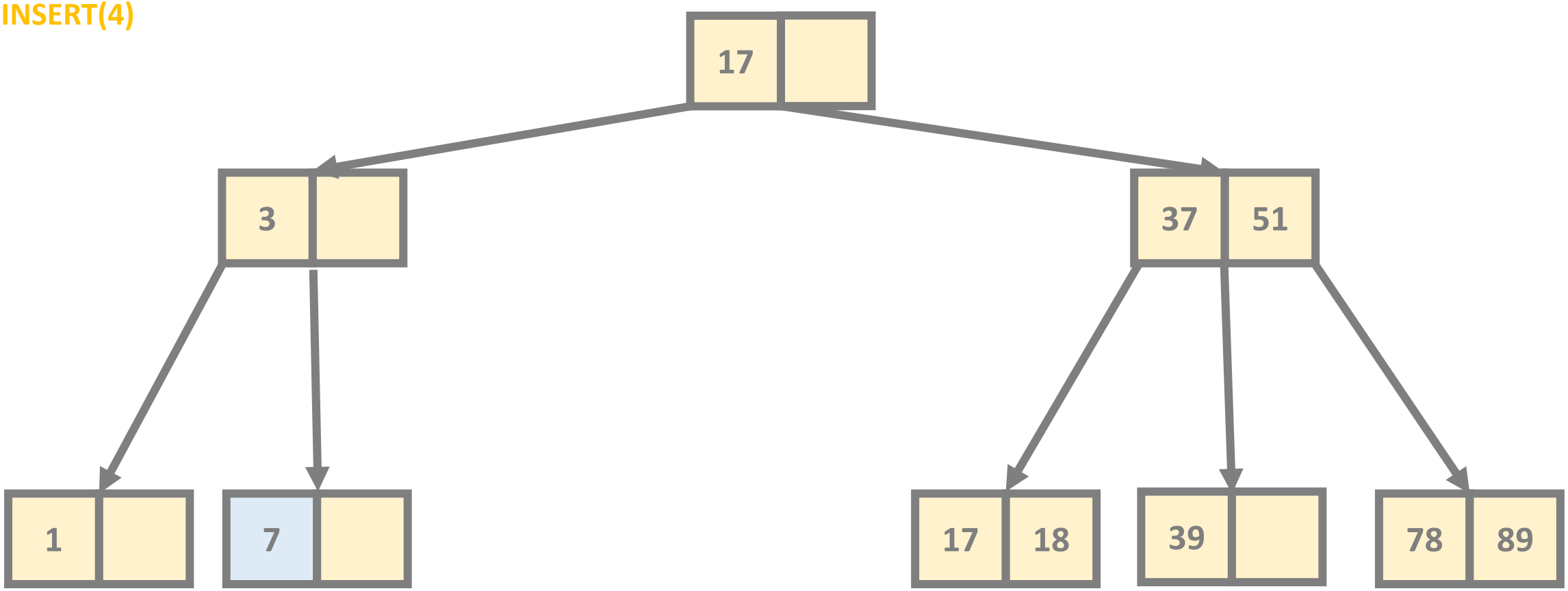
B-Trees

INSERT(4)



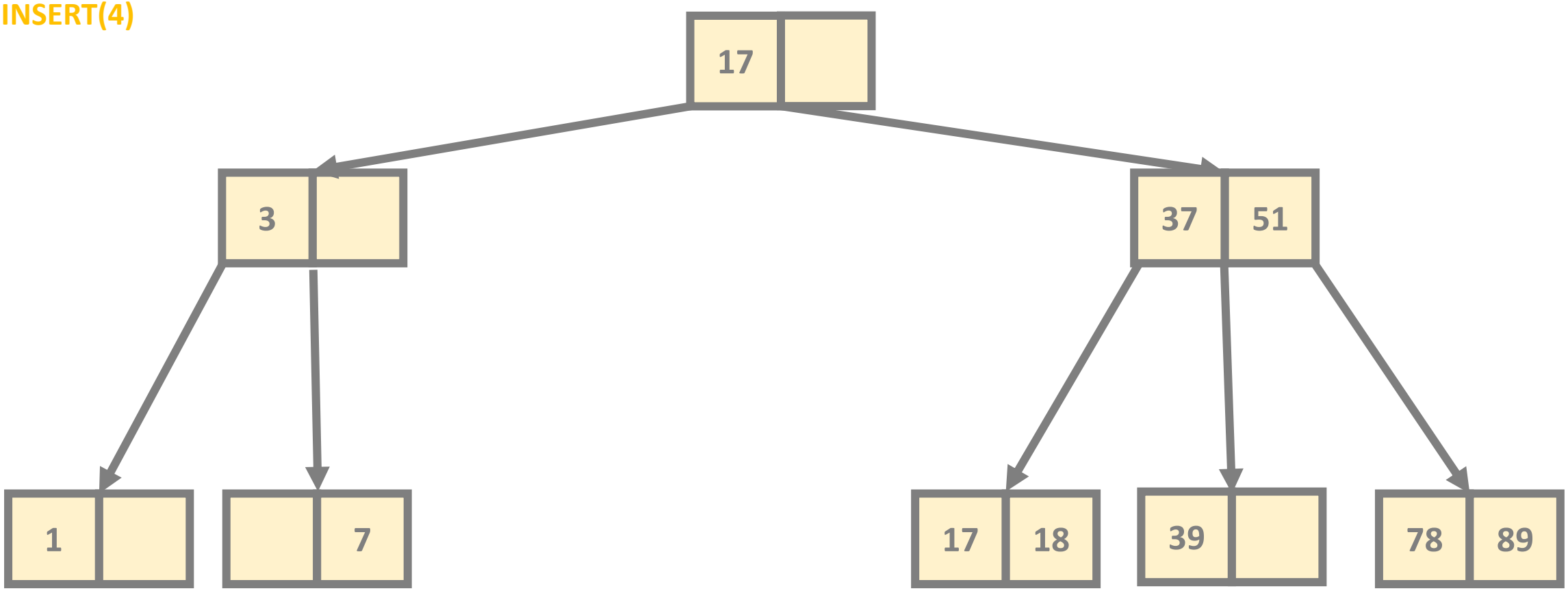
B-Trees

INSERT(4)



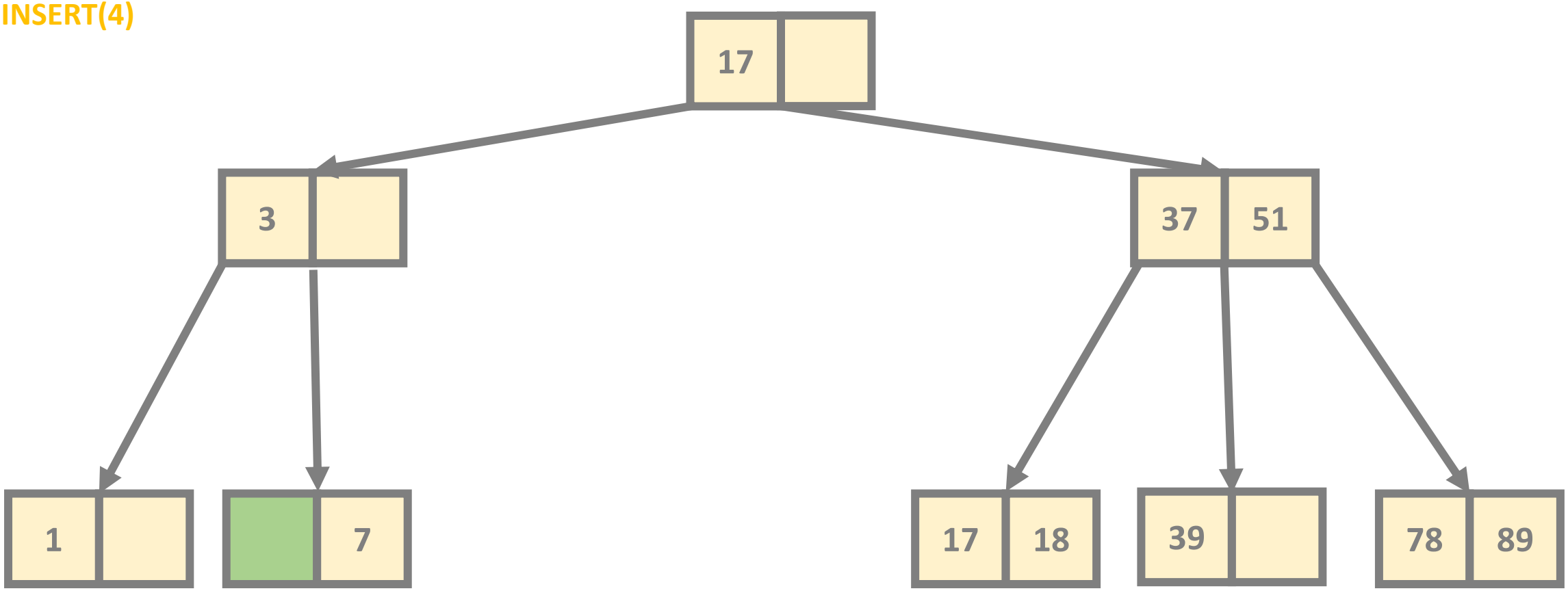
B-Trees

INSERT(4)



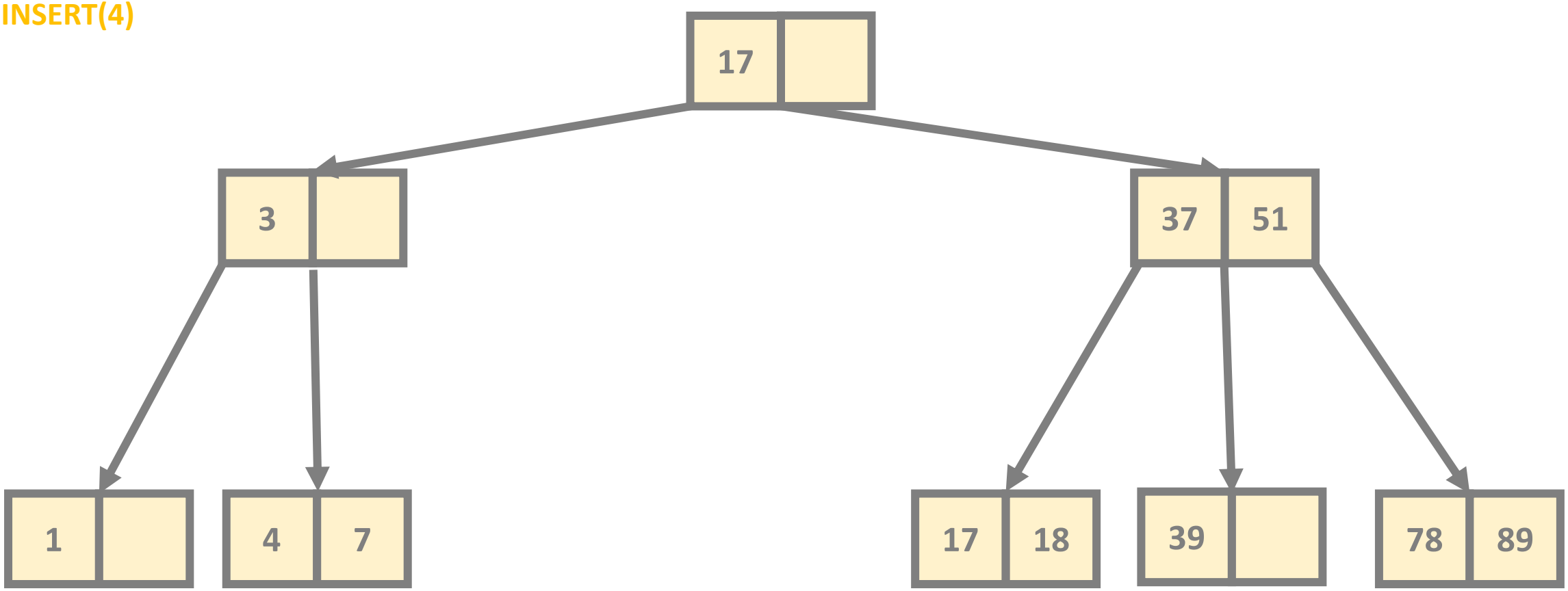
B-Trees

INSERT(4)

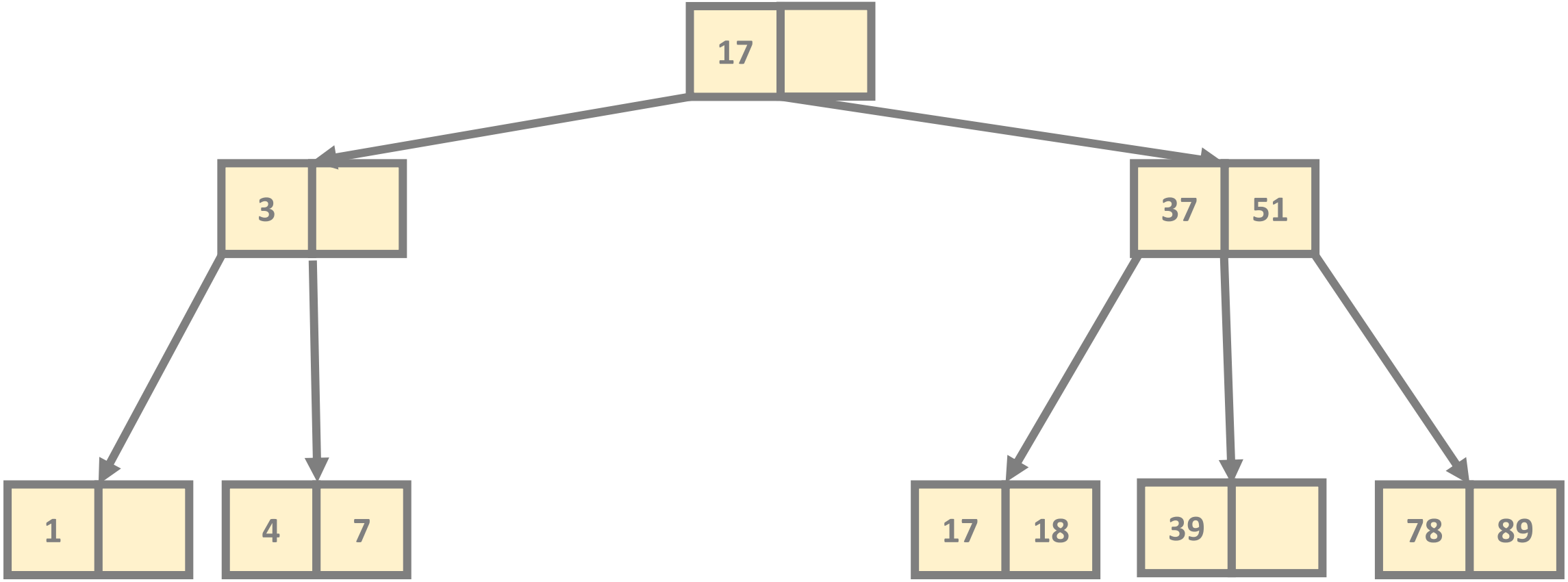


B-Trees

INSERT(4)

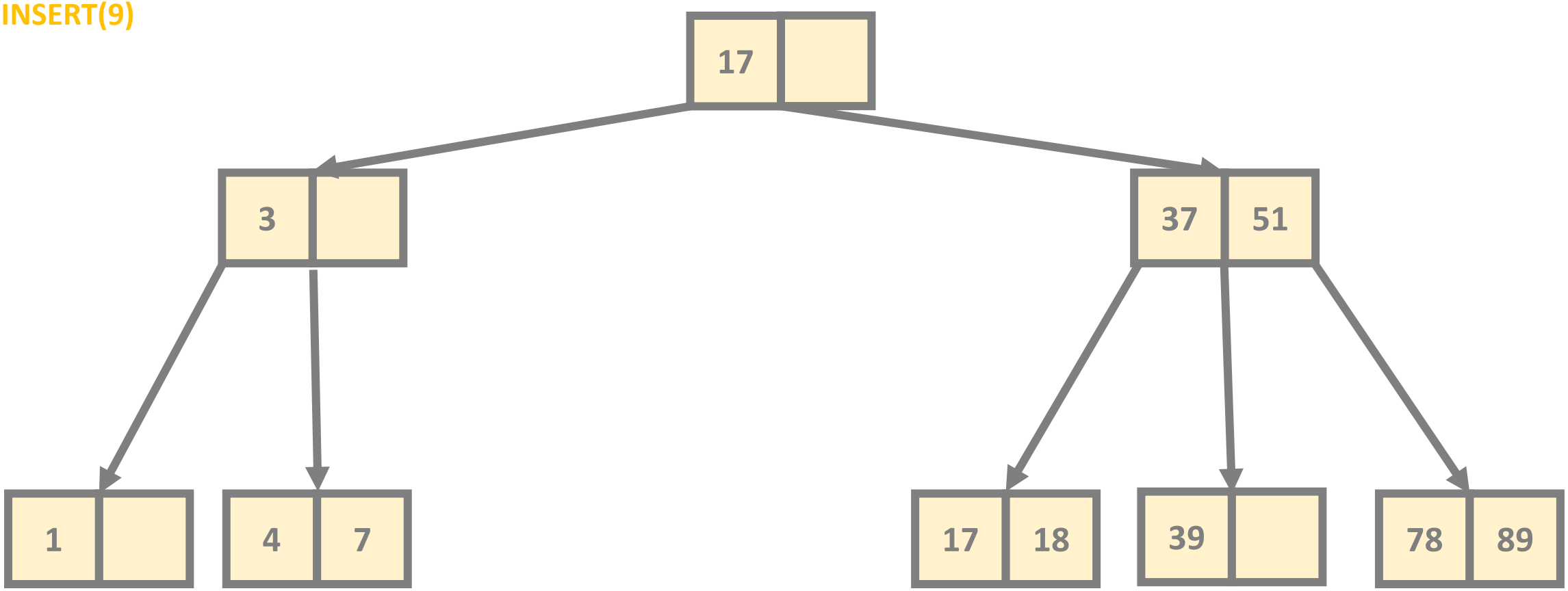


B-Trees



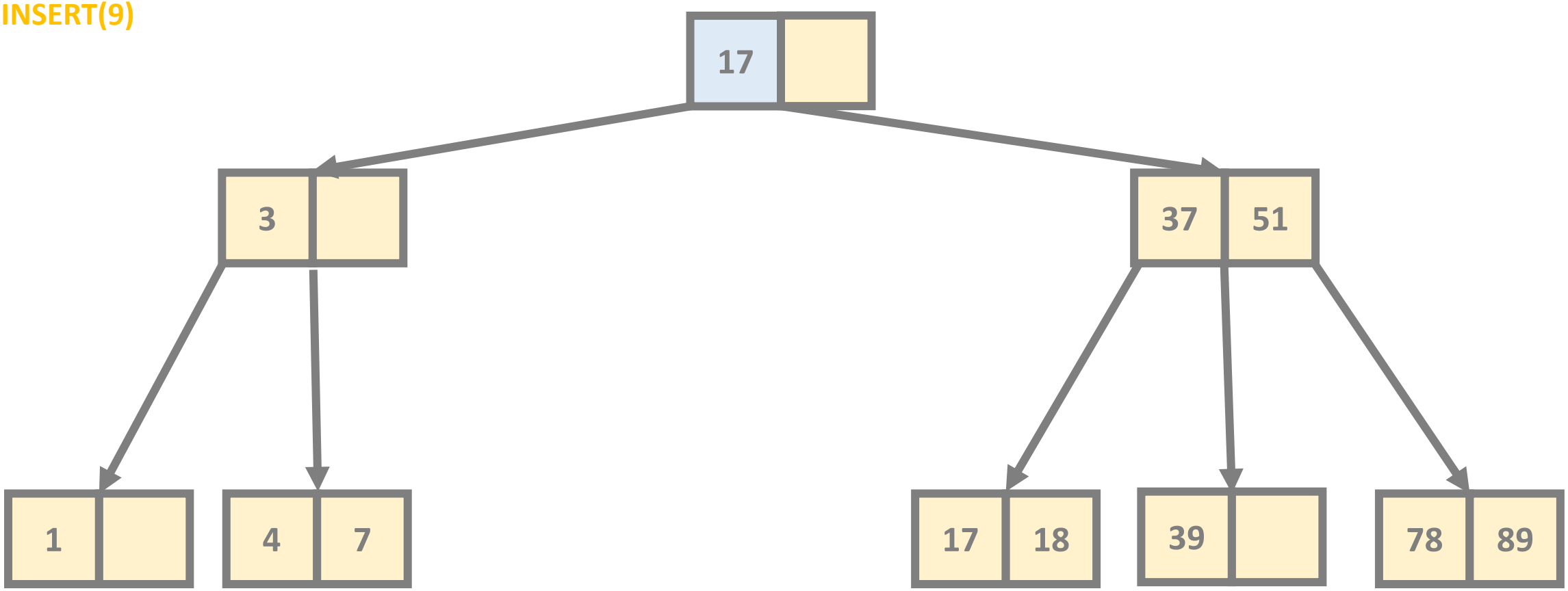
B-Trees

INSERT(9)



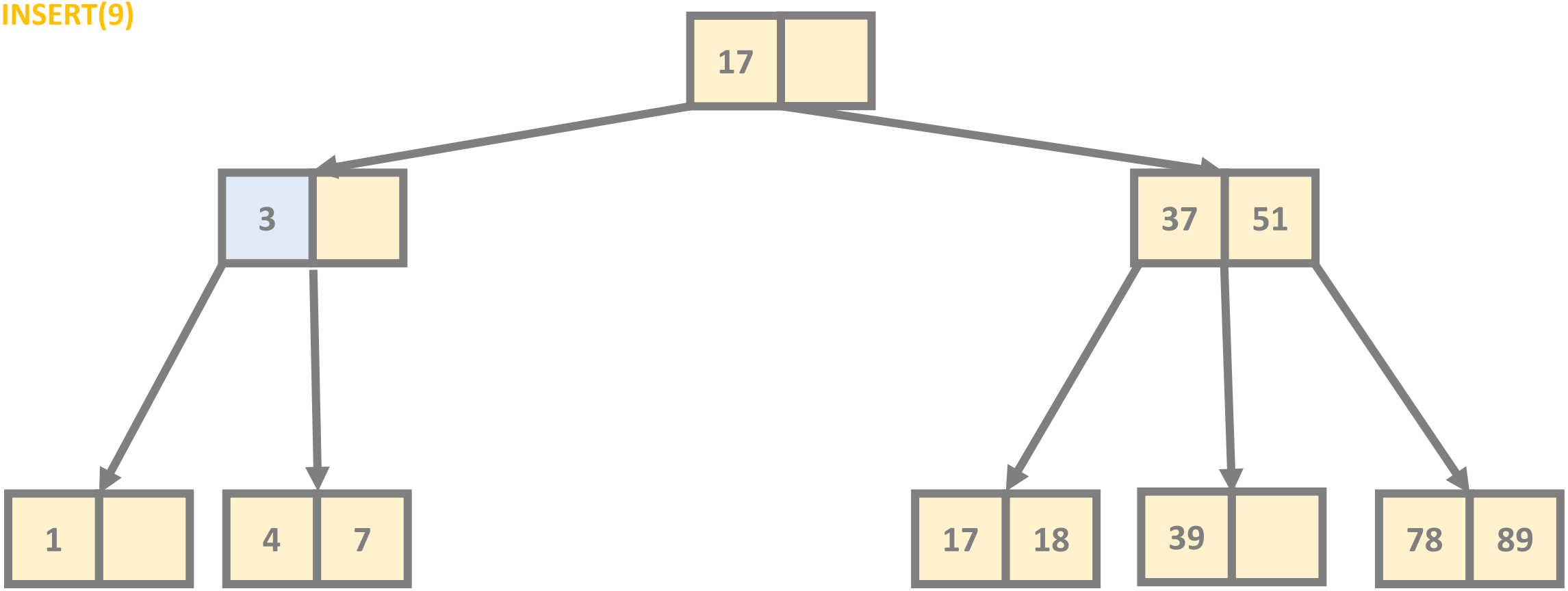
B-Trees

INSERT(9)



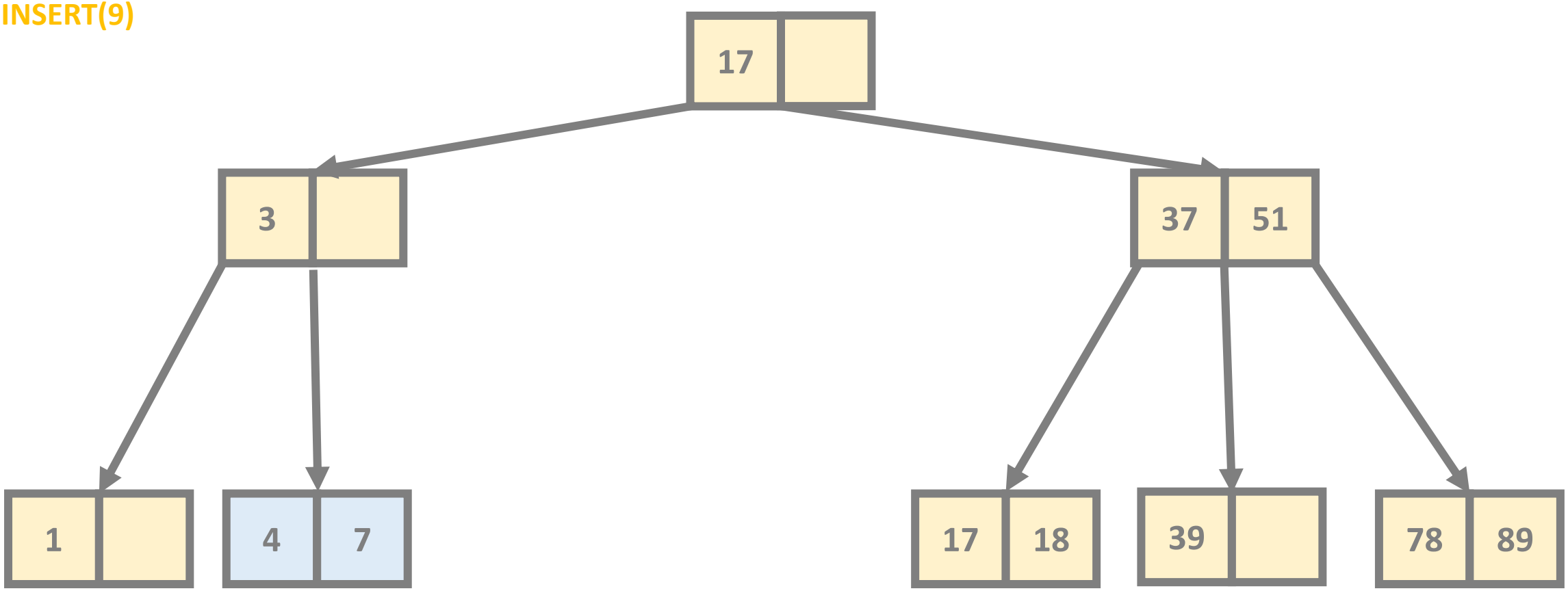
B-Trees

INSERT(9)



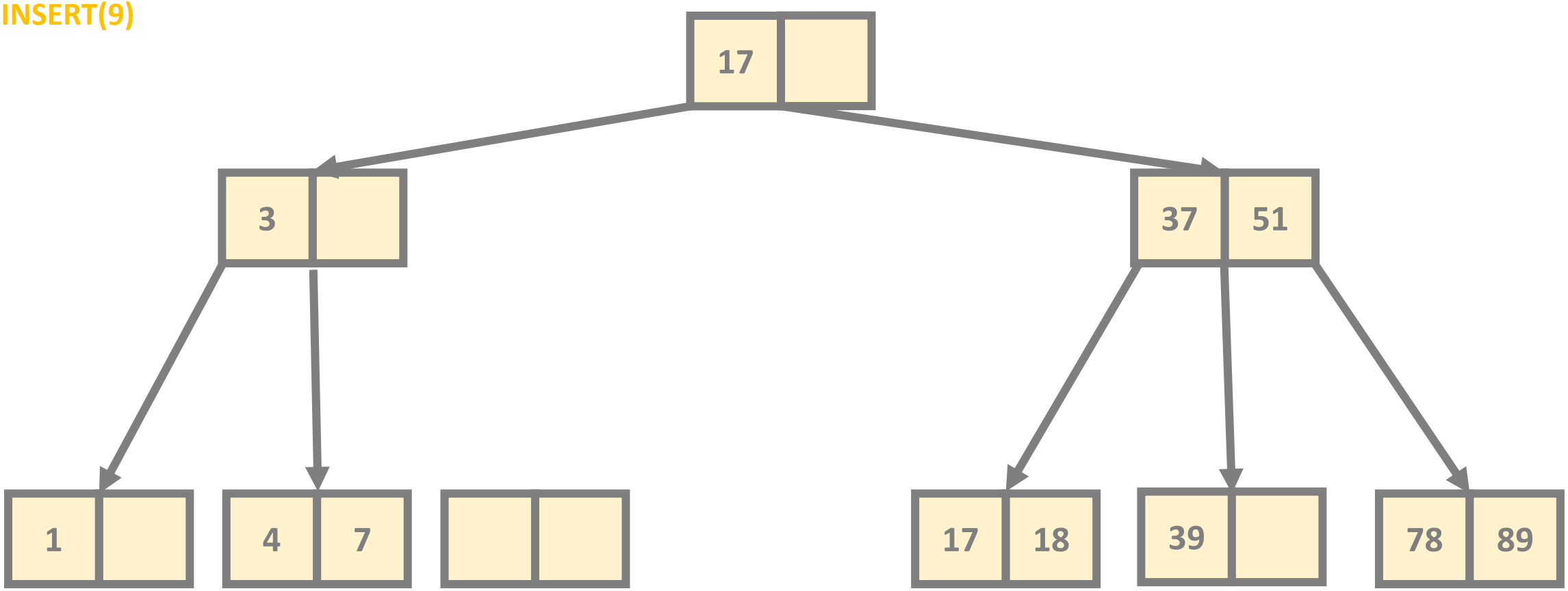
B-Trees

INSERT(9)



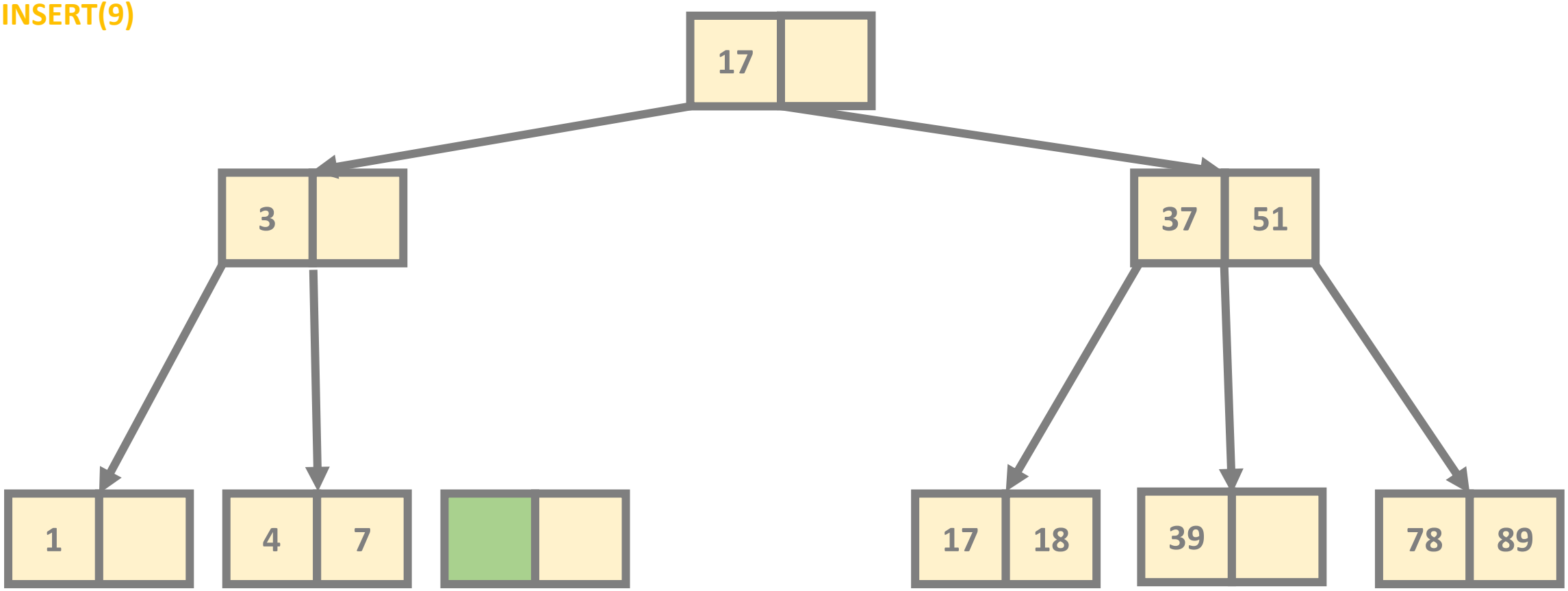
B-Trees

INSERT(9)



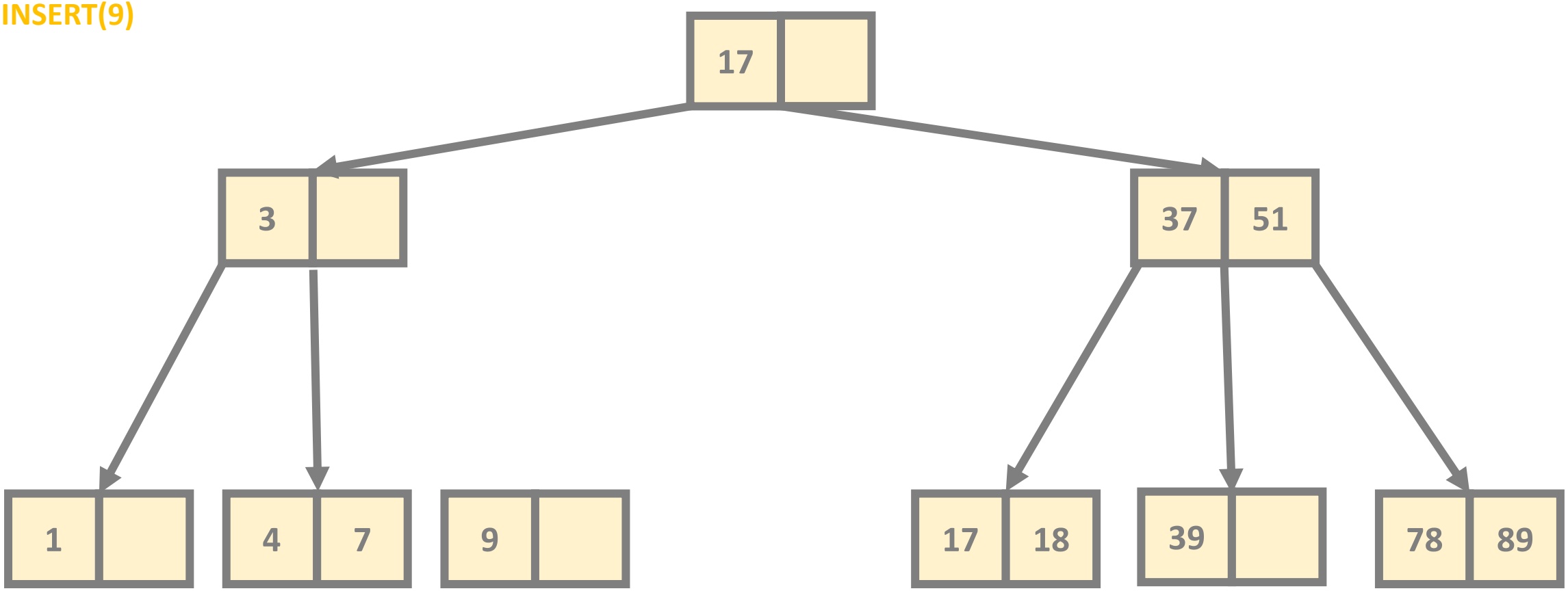
B-Trees

INSERT(9)



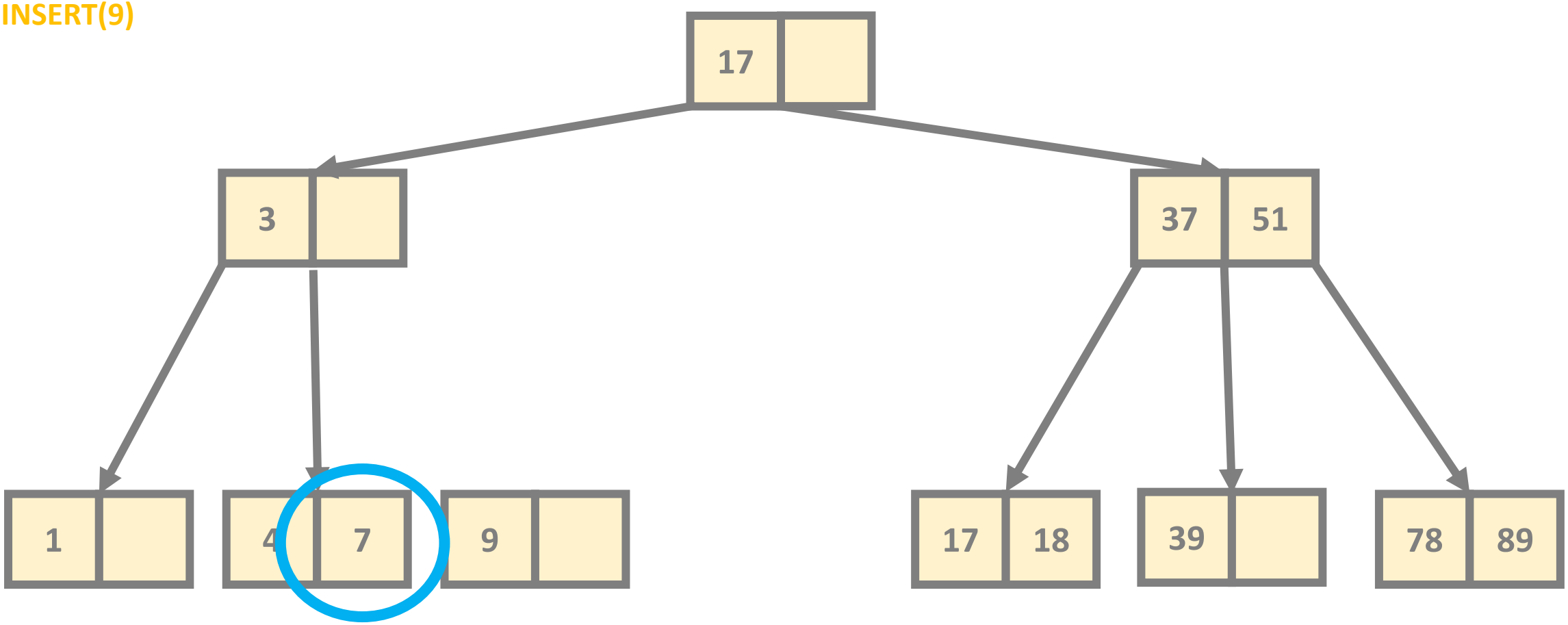
B-Trees

INSERT(9)



B-Trees

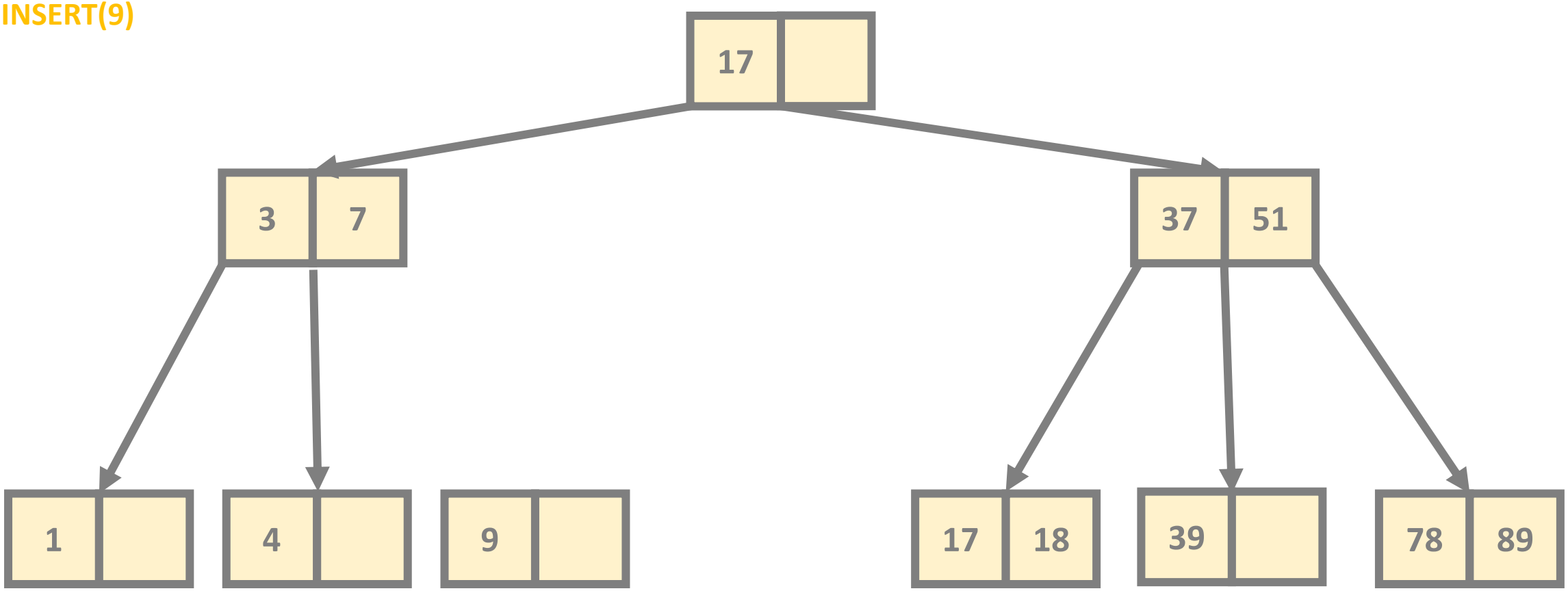
INSERT(9)



we always promote the
middle value *in these cases*

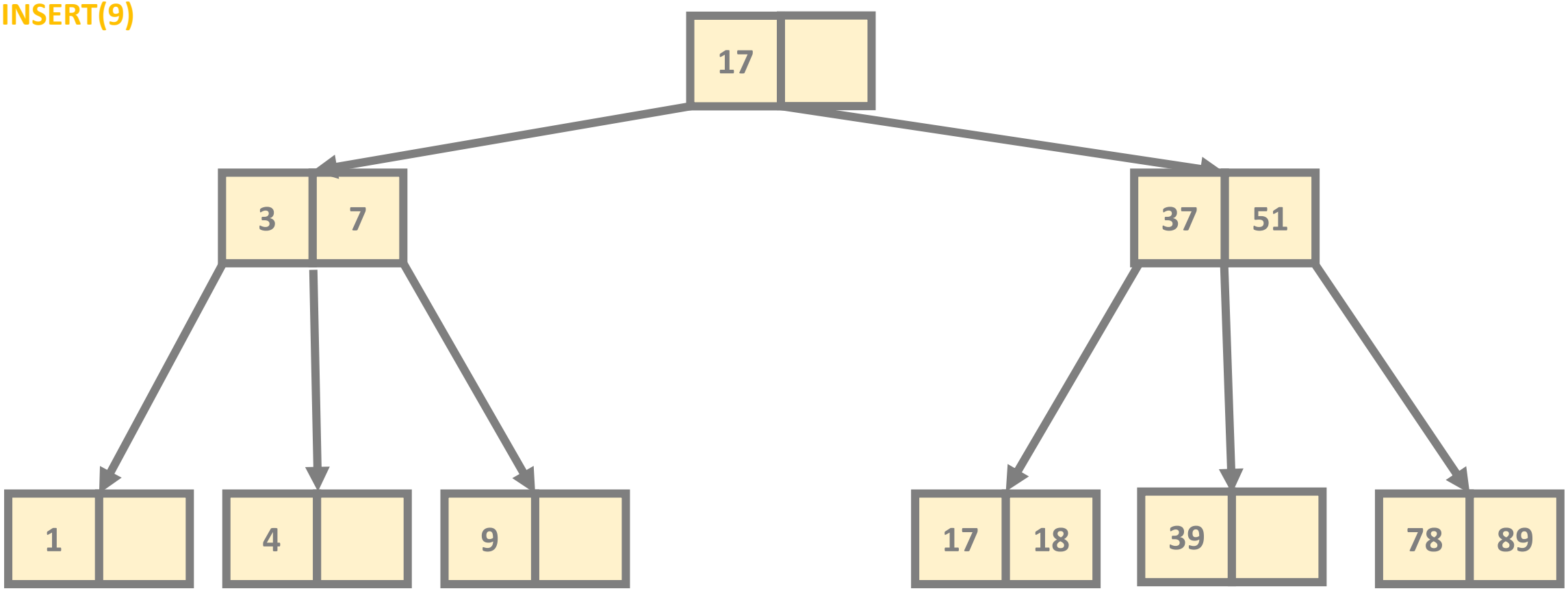
B-Trees

INSERT(9)

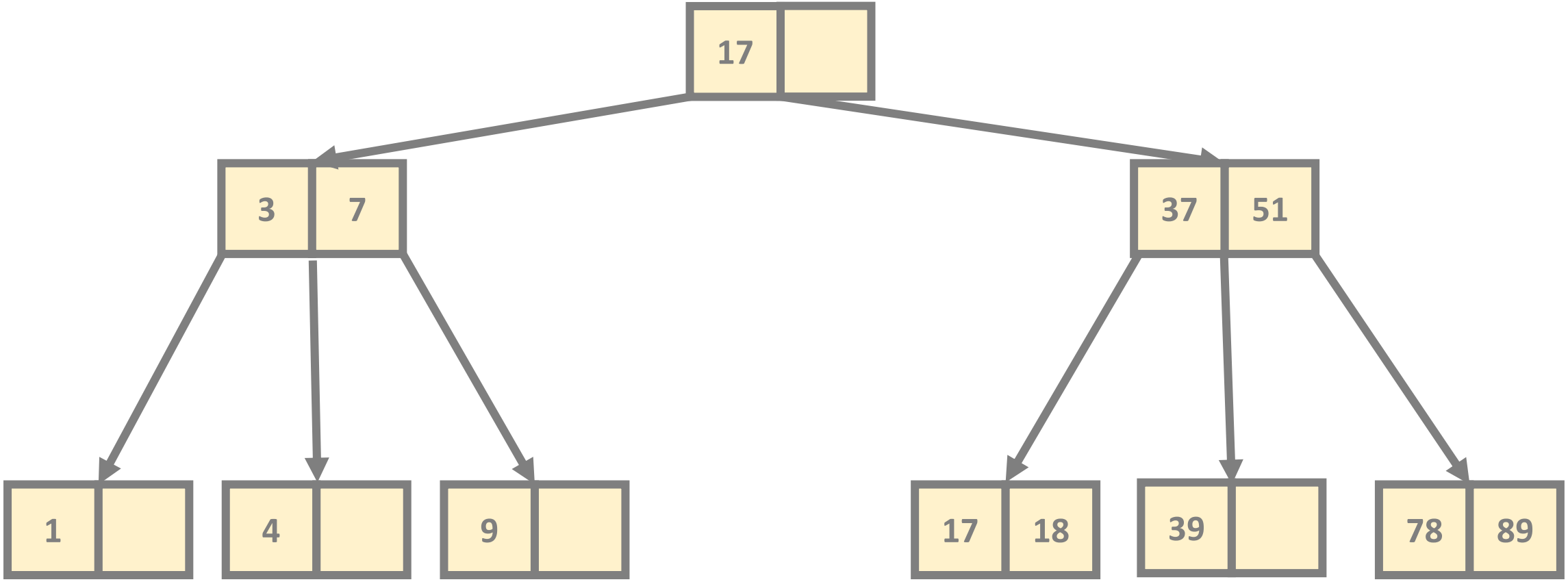


B-Trees

INSERT(9)

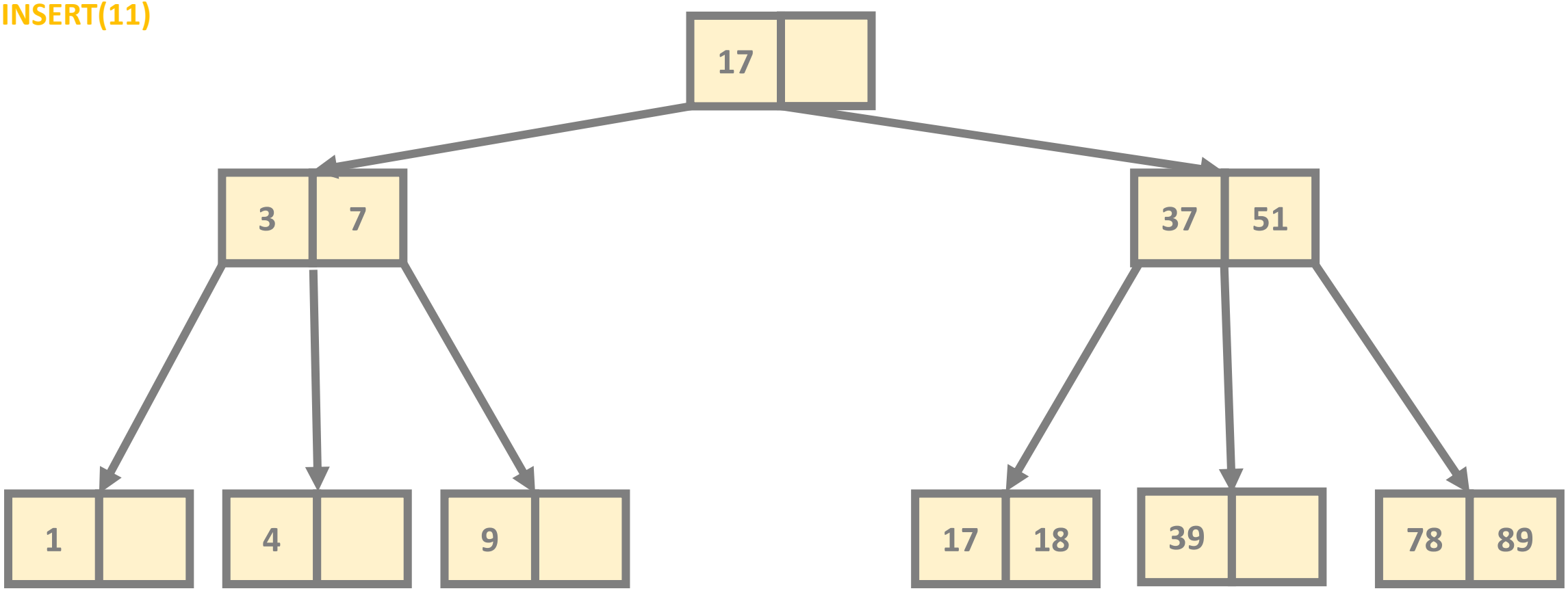


B-Trees



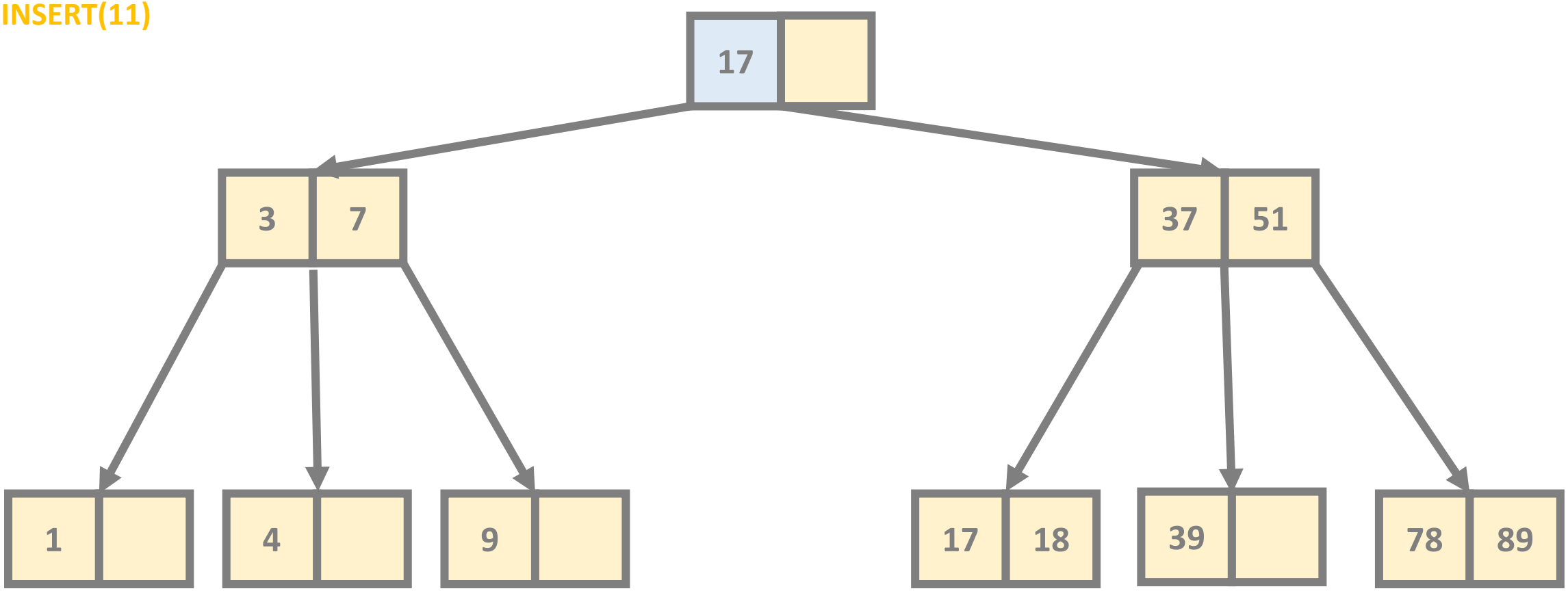
B-Trees

INSERT(11)



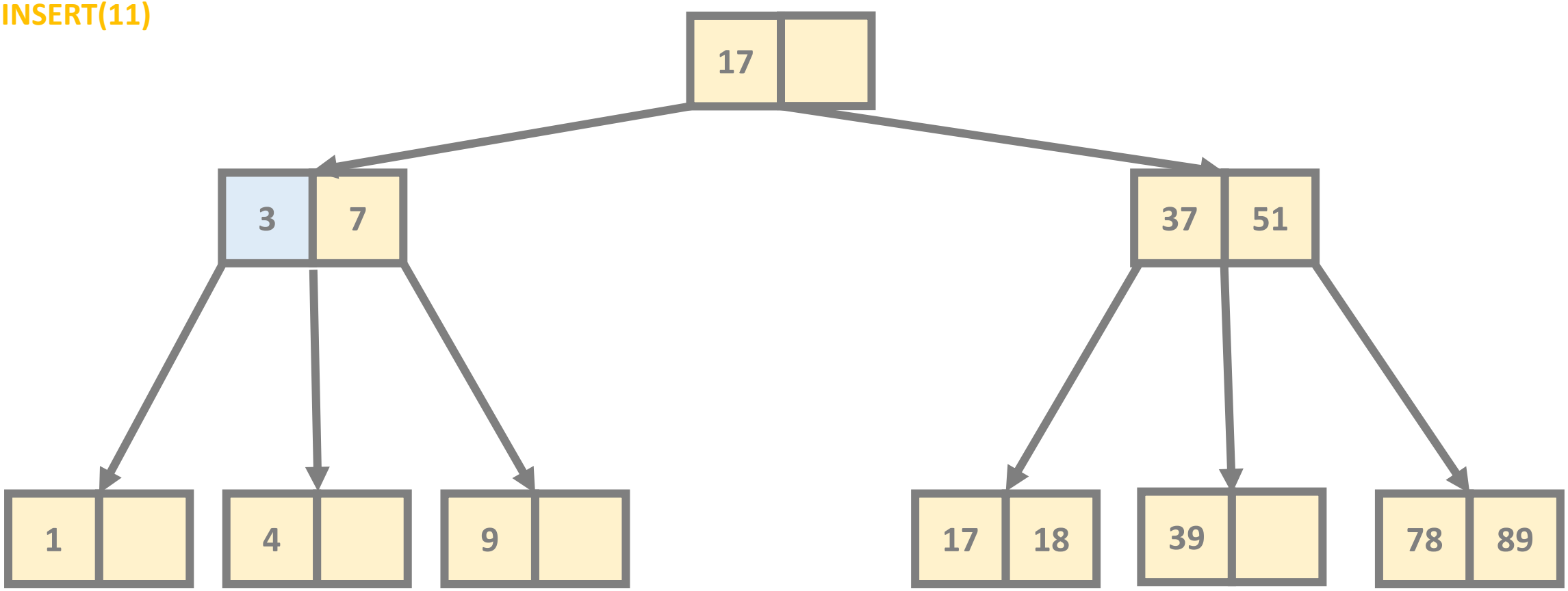
B-Trees

INSERT(11)



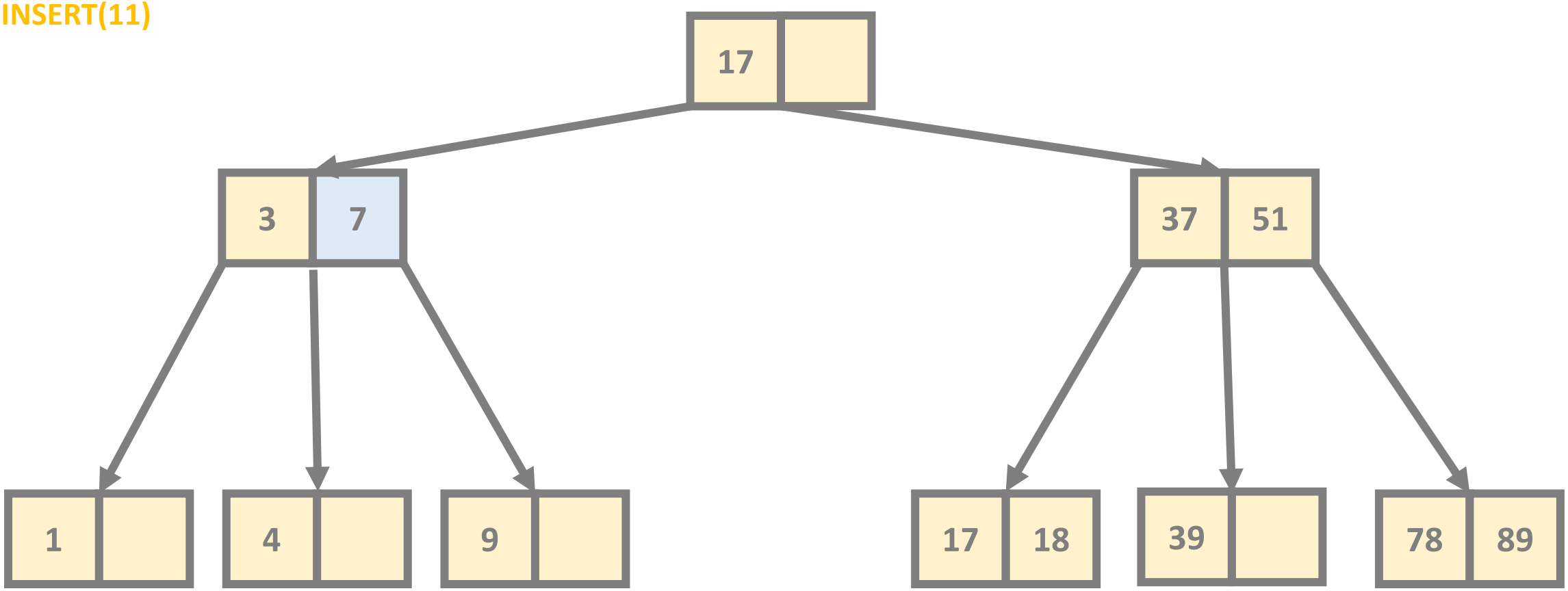
B-Trees

INSERT(11)



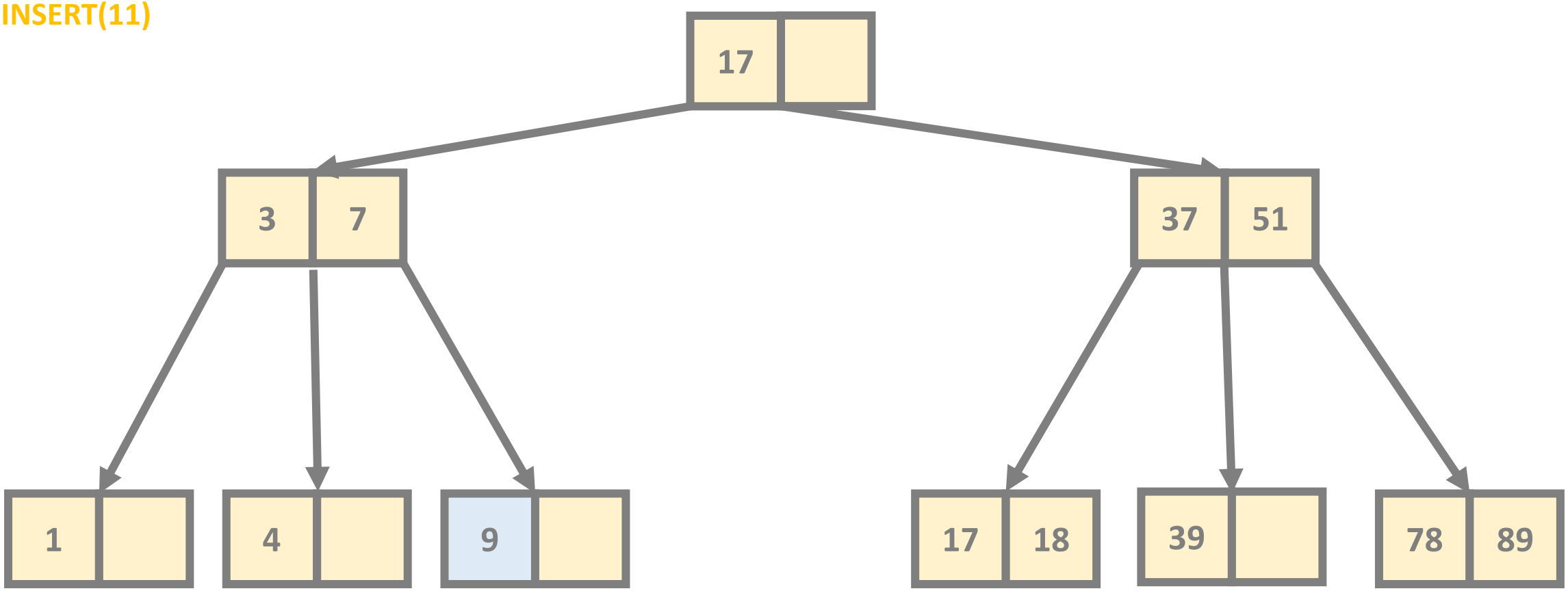
B-Trees

INSERT(11)



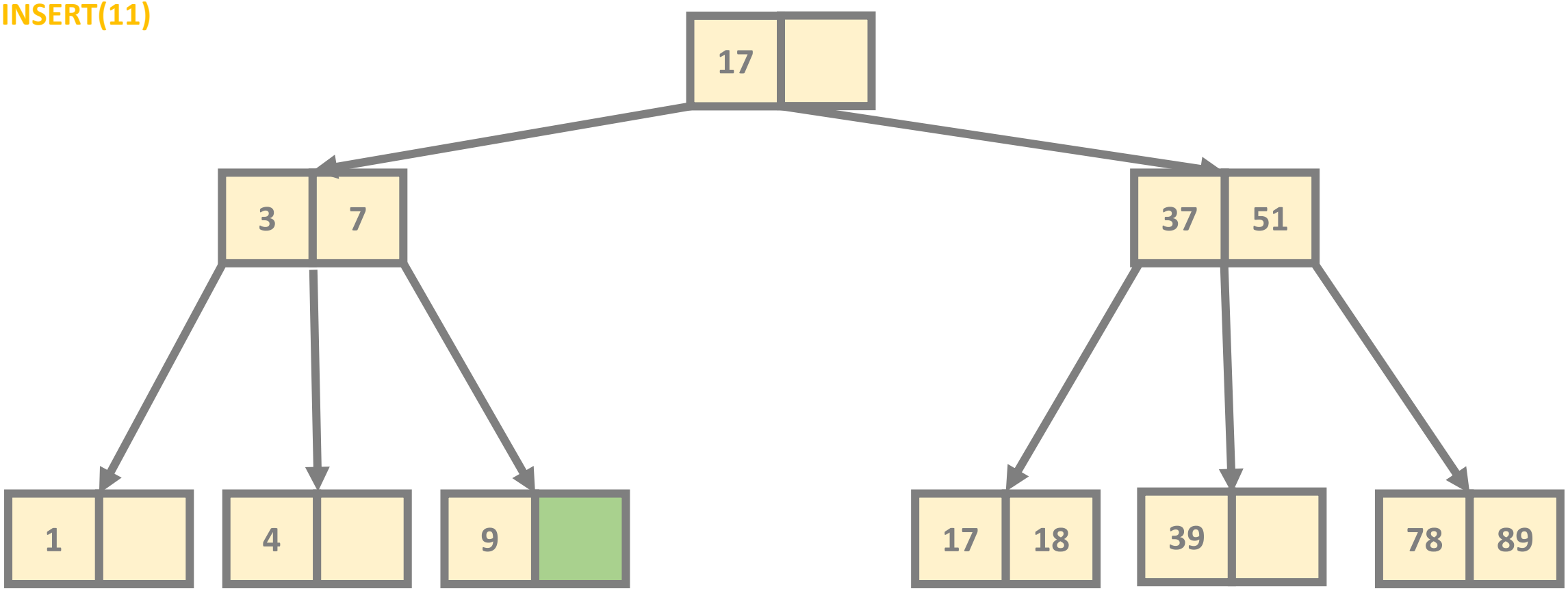
B-Trees

INSERT(11)



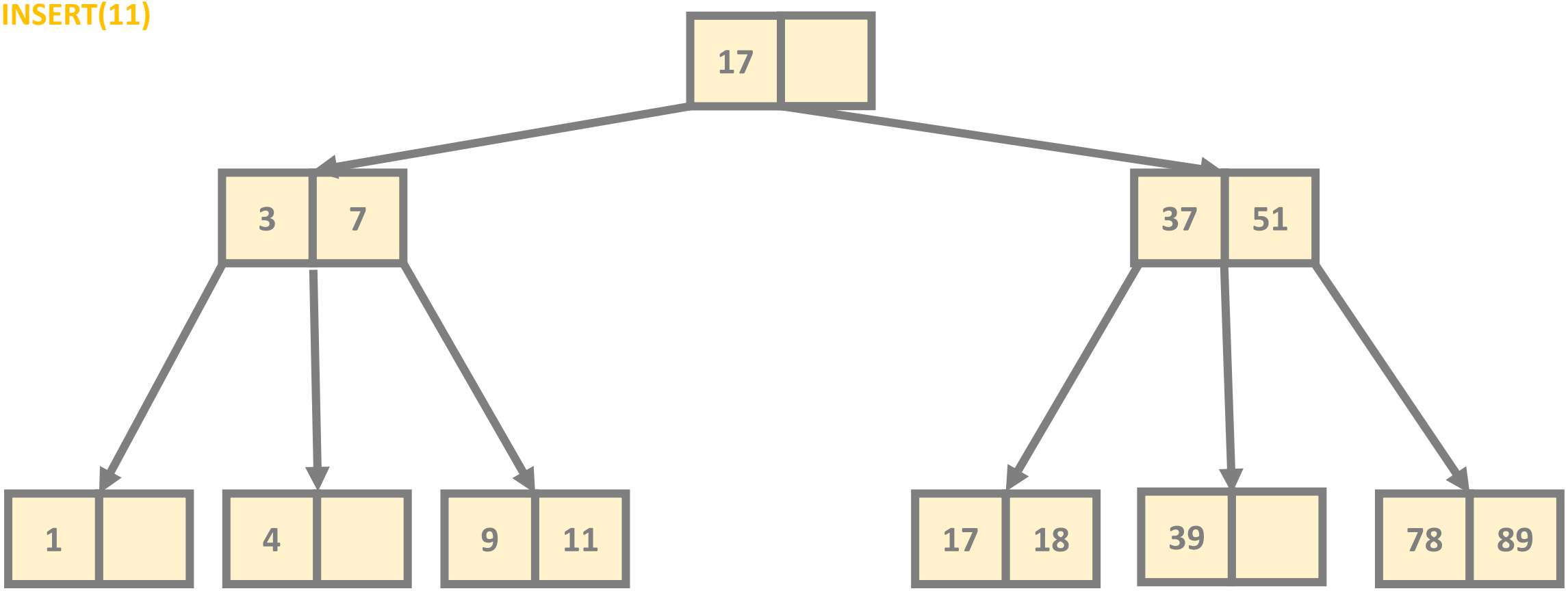
B-Trees

INSERT(11)

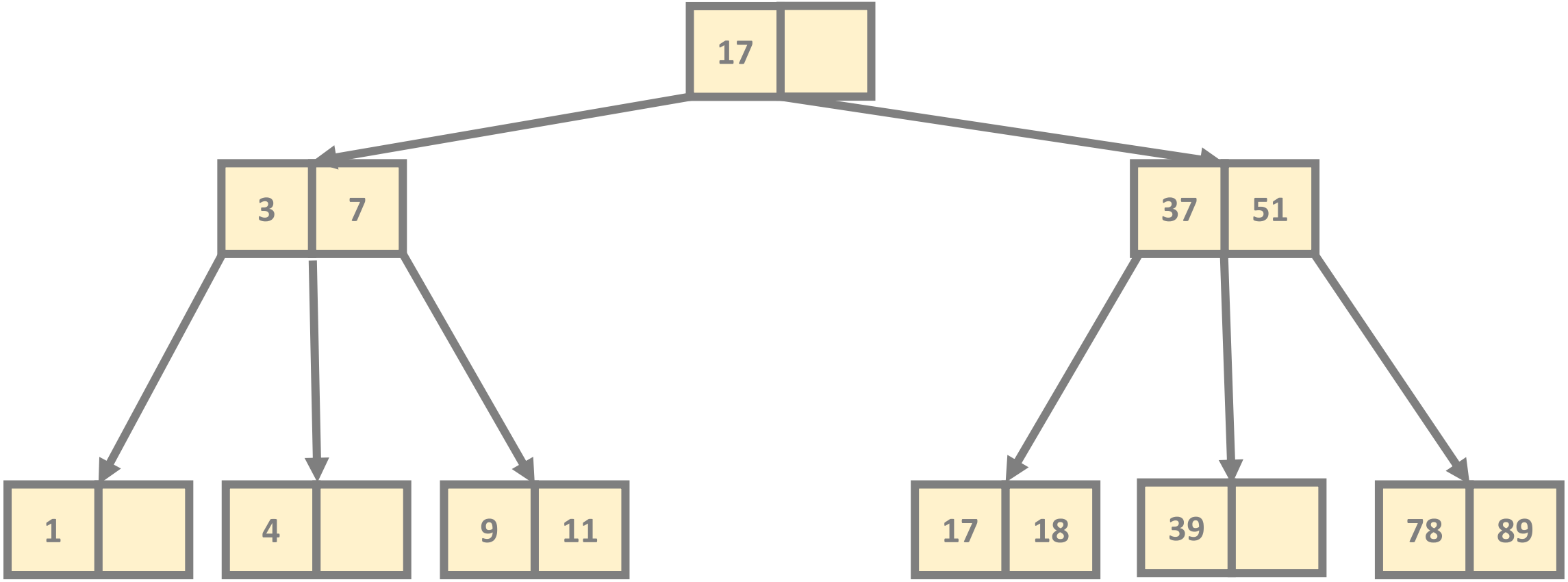


B-Trees

INSERT(11)

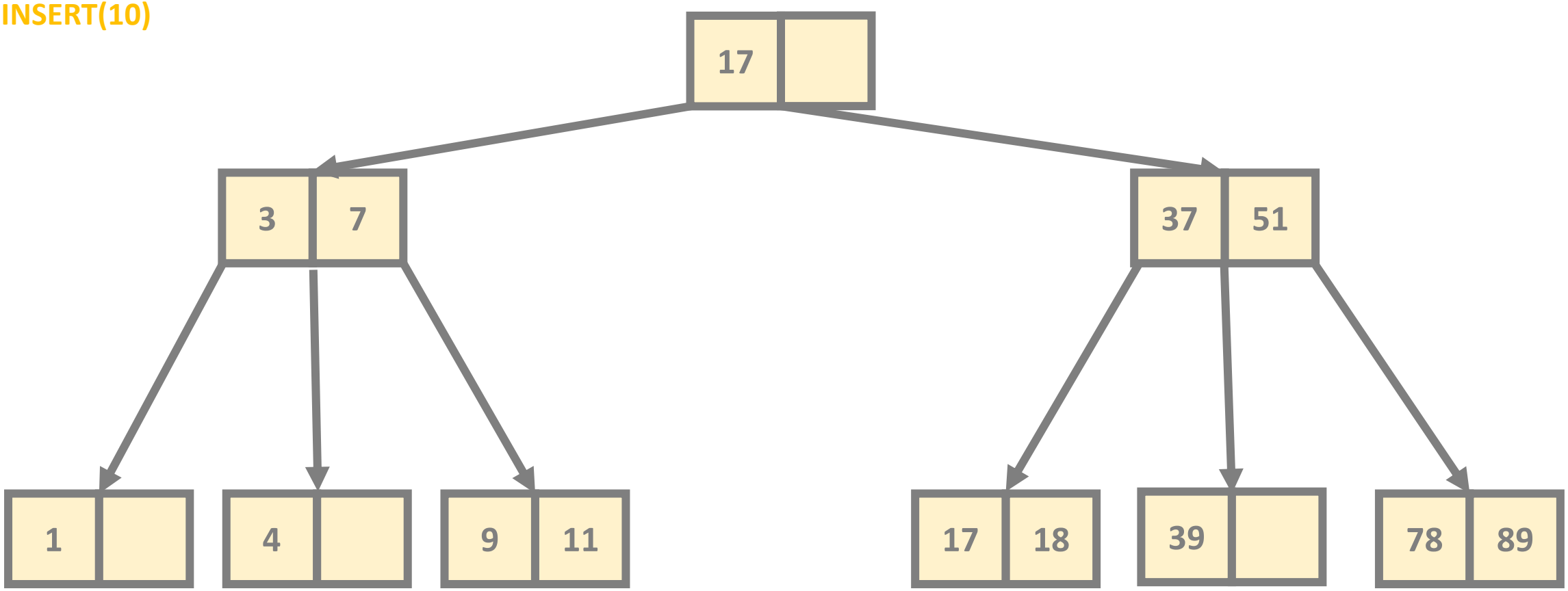


B-Trees



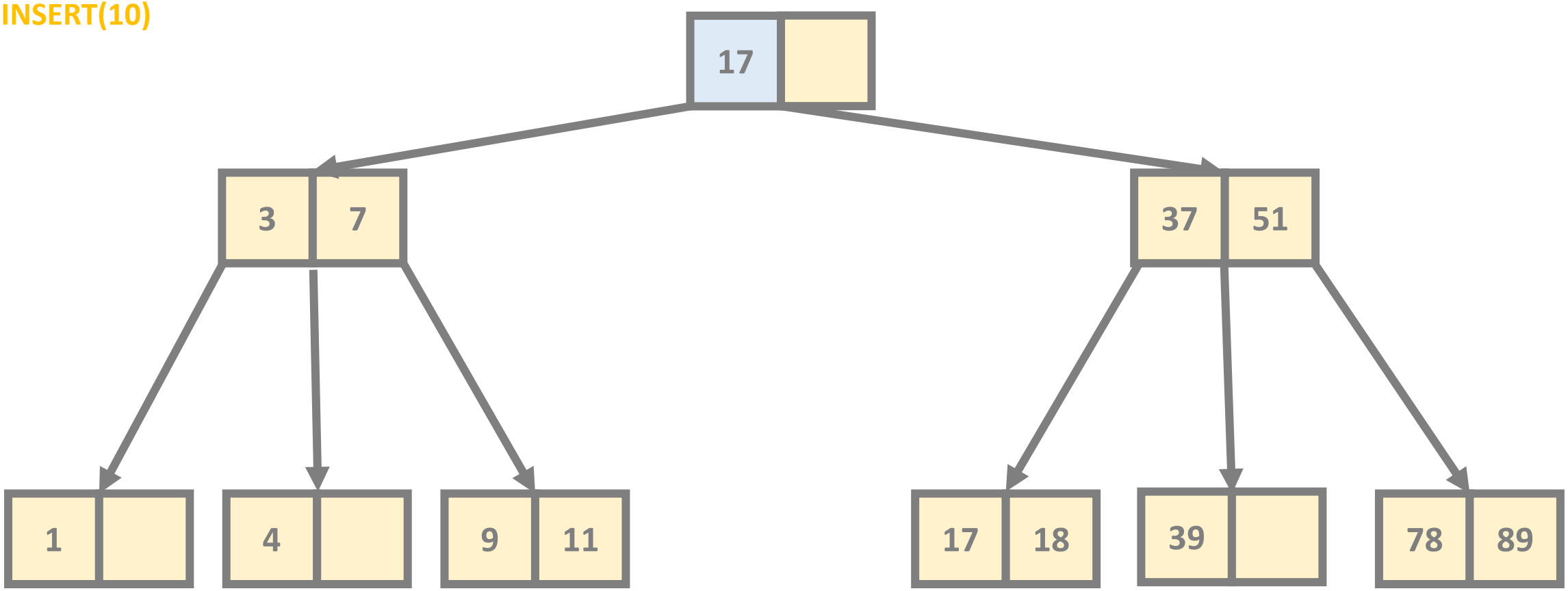
B-Trees

INSERT(10)



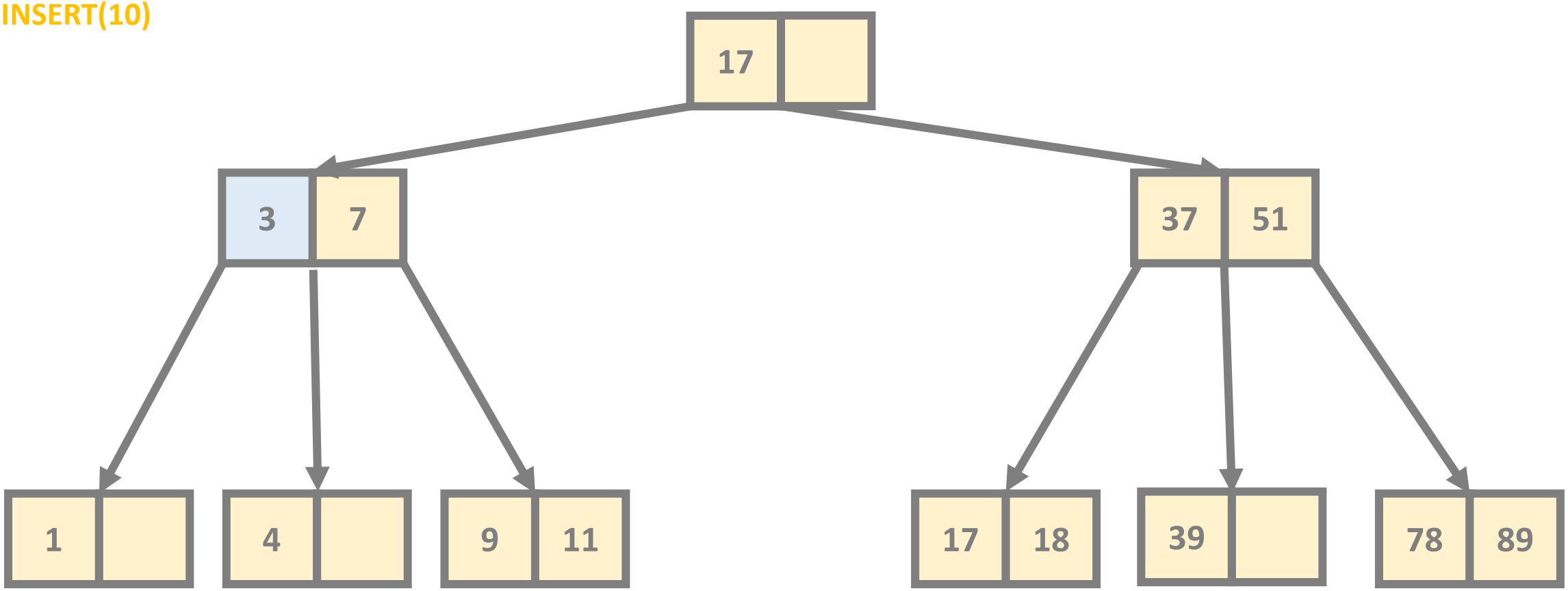
B-Trees

INSERT(10)



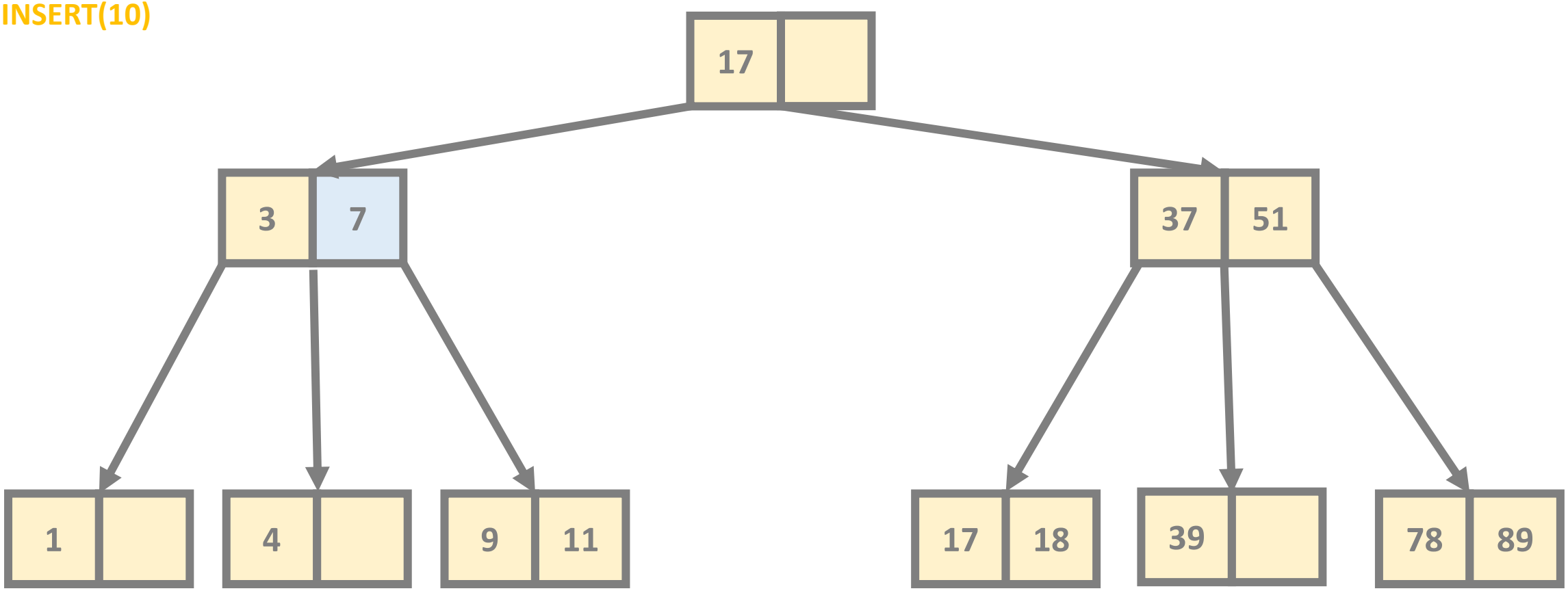
B-Trees

INSERT(10)



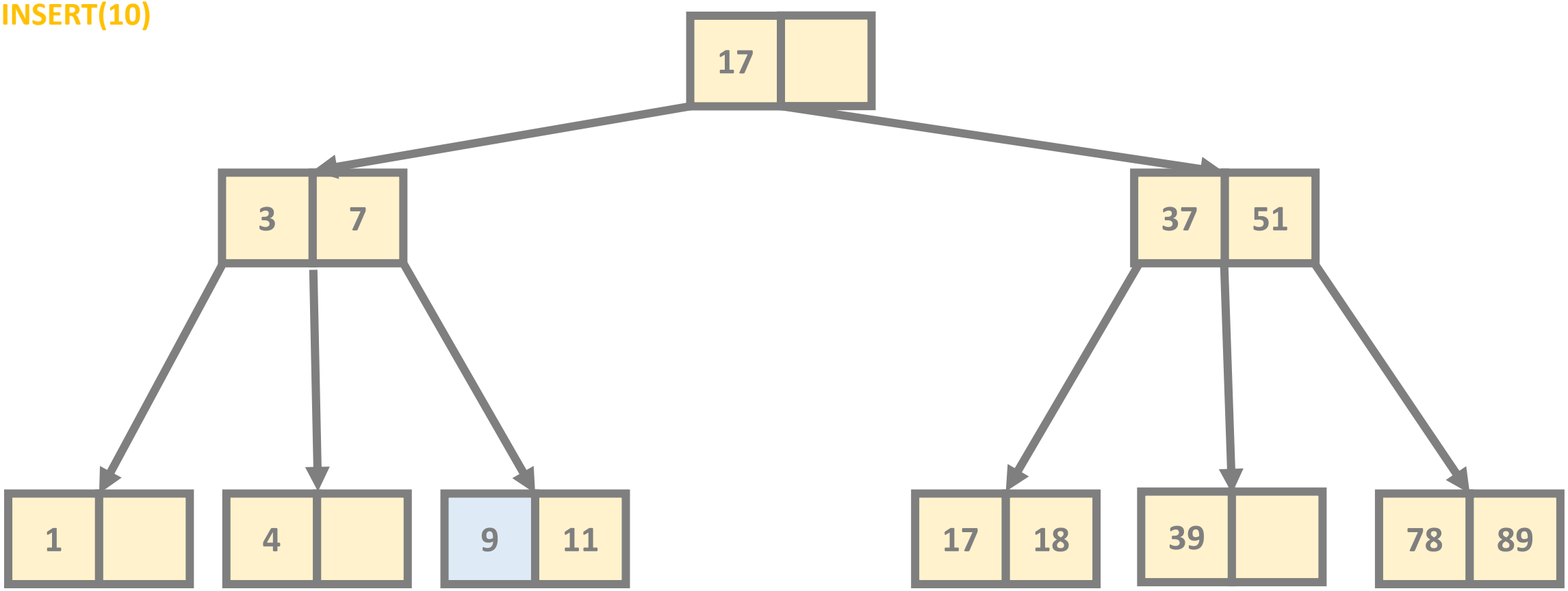
B-Trees

INSERT(10)



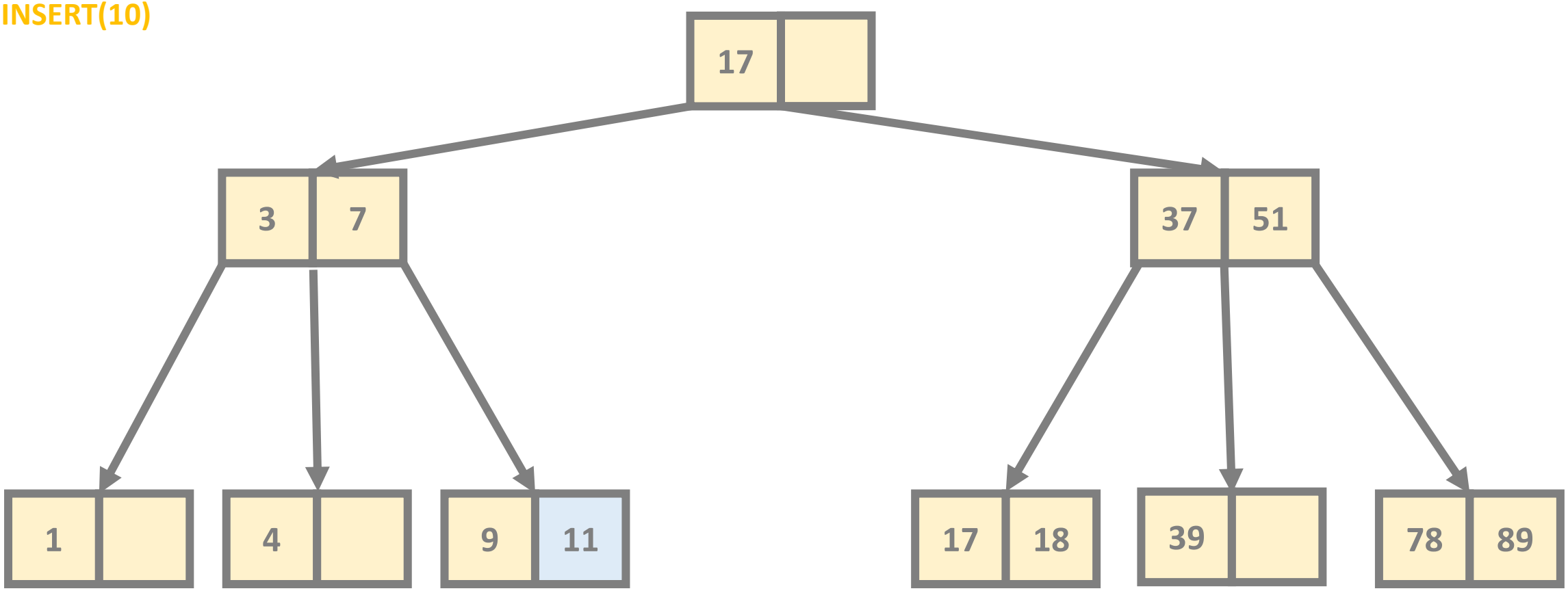
B-Trees

INSERT(10)



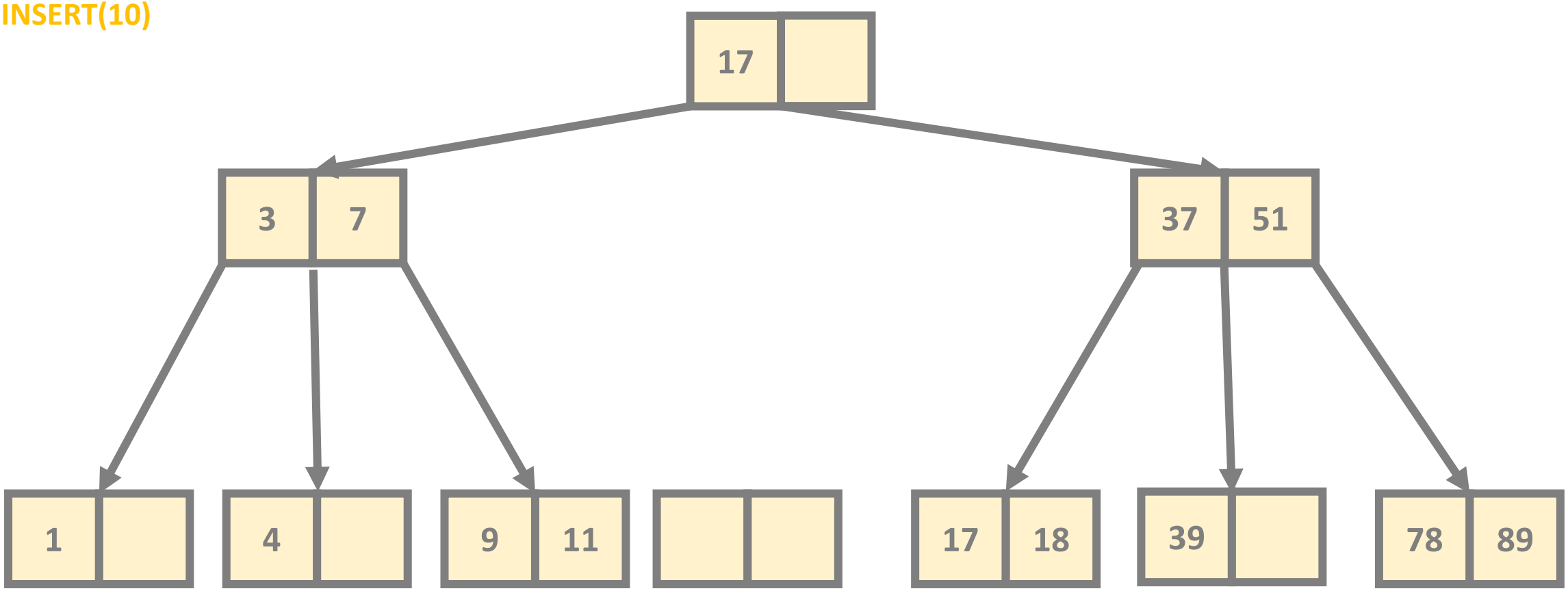
B-Trees

INSERT(10)



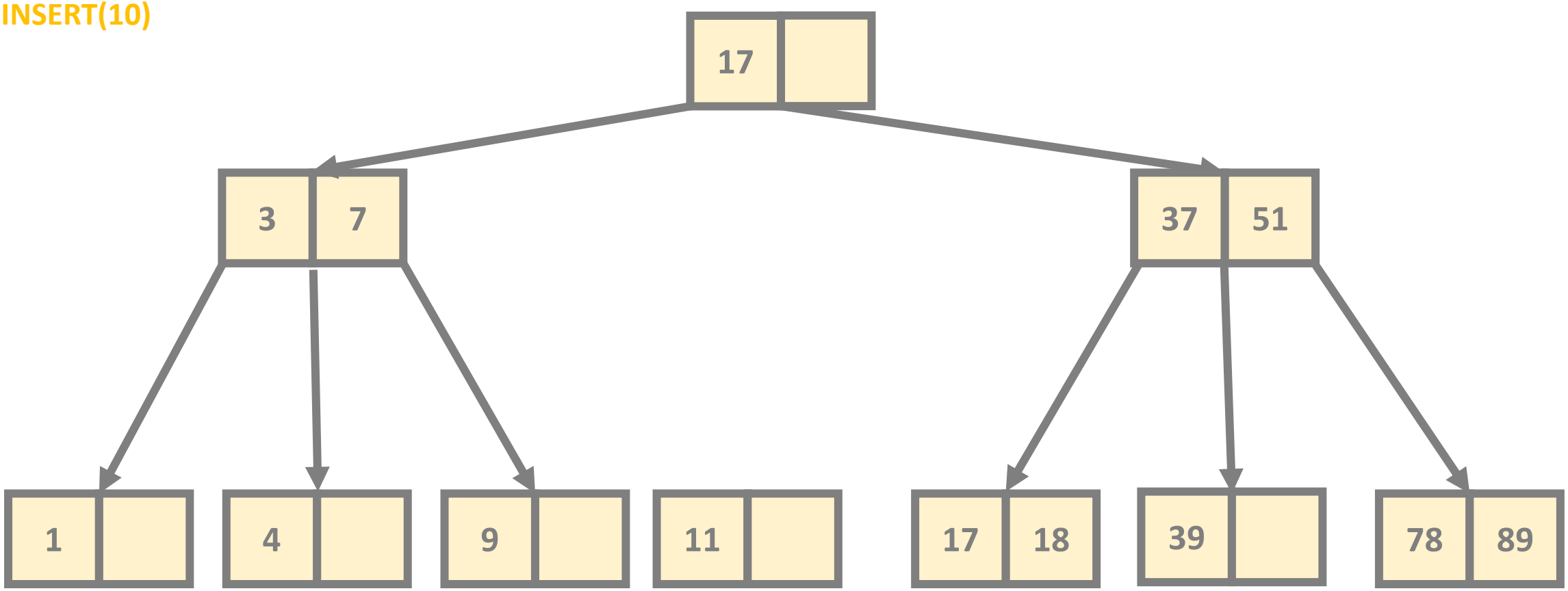
B-Trees

INSERT(10)



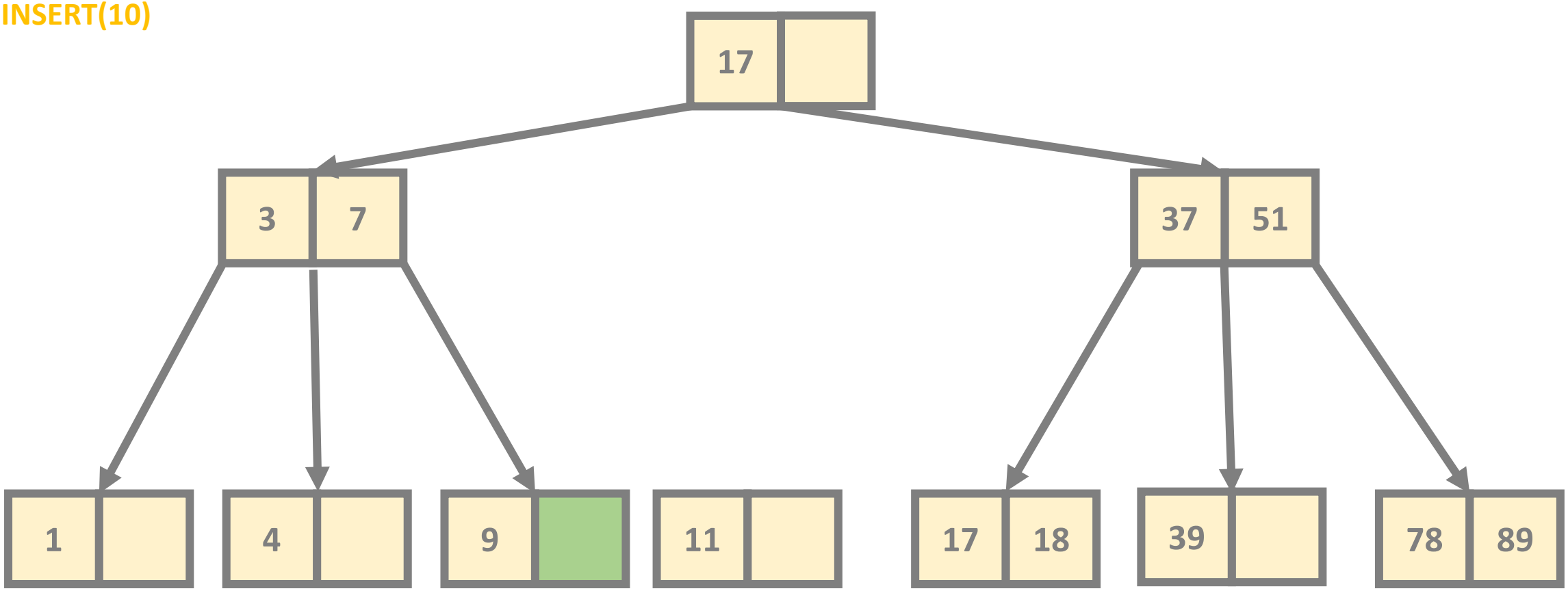
B-Trees

INSERT(10)



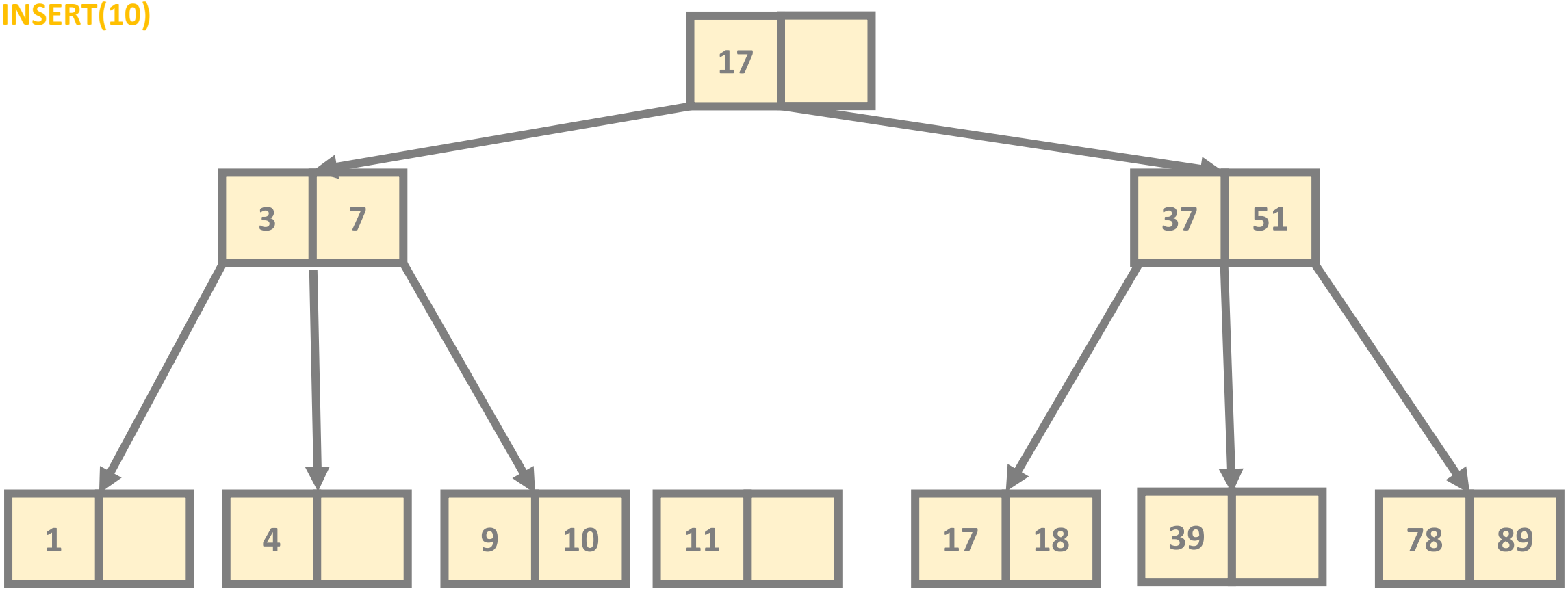
B-Trees

INSERT(10)



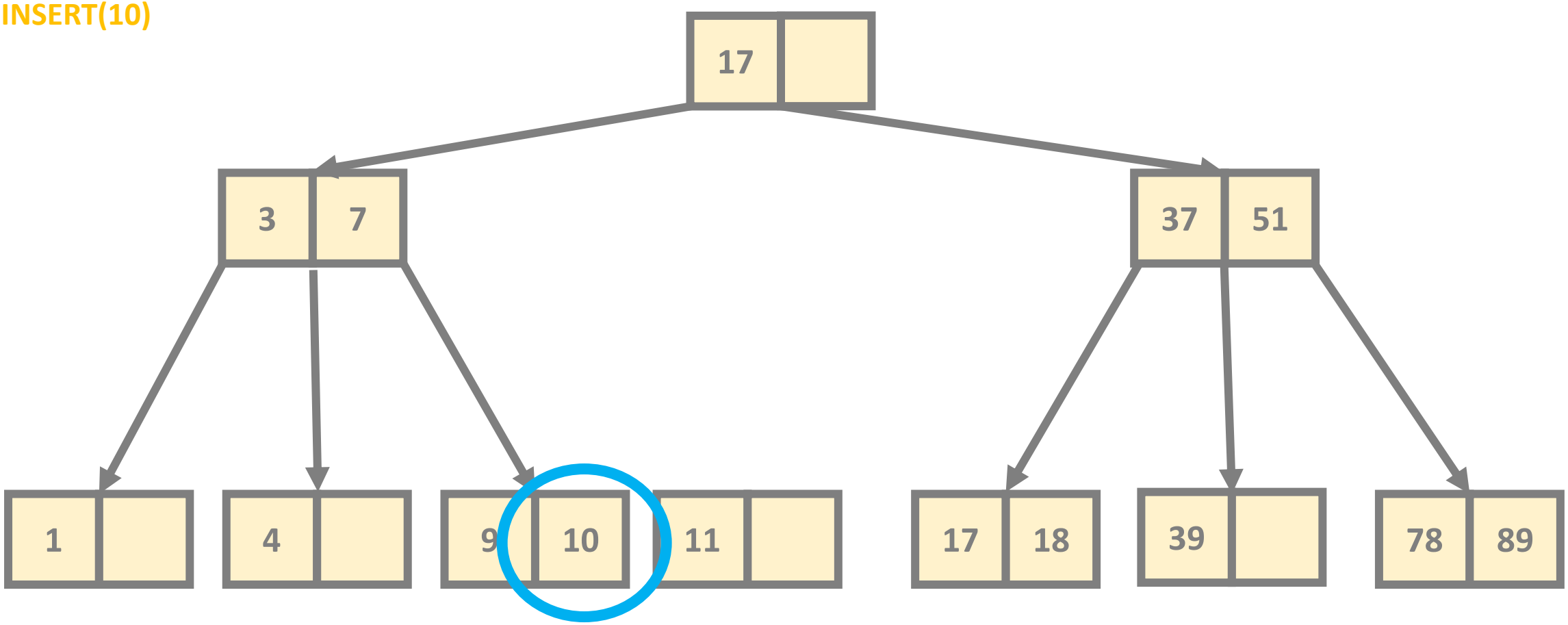
B-Trees

INSERT(10)



B-Trees

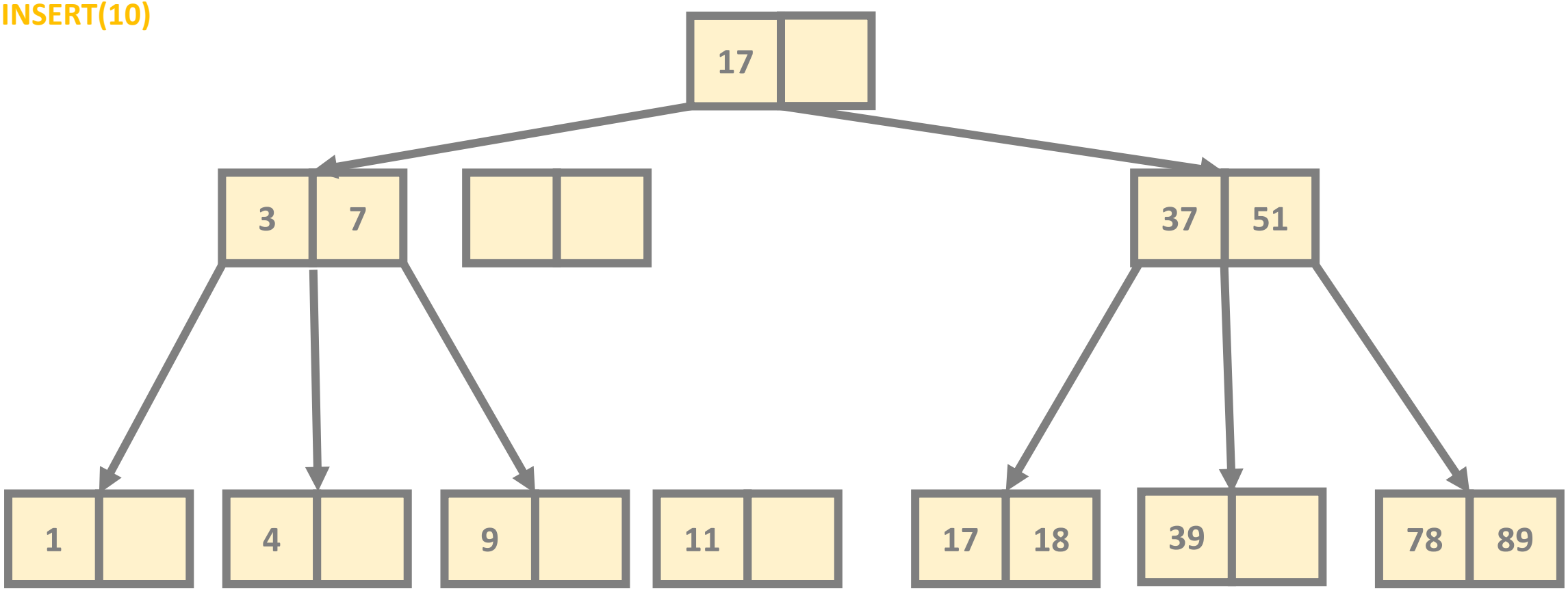
INSERT(10)



we always promote the
middle value *in these cases*

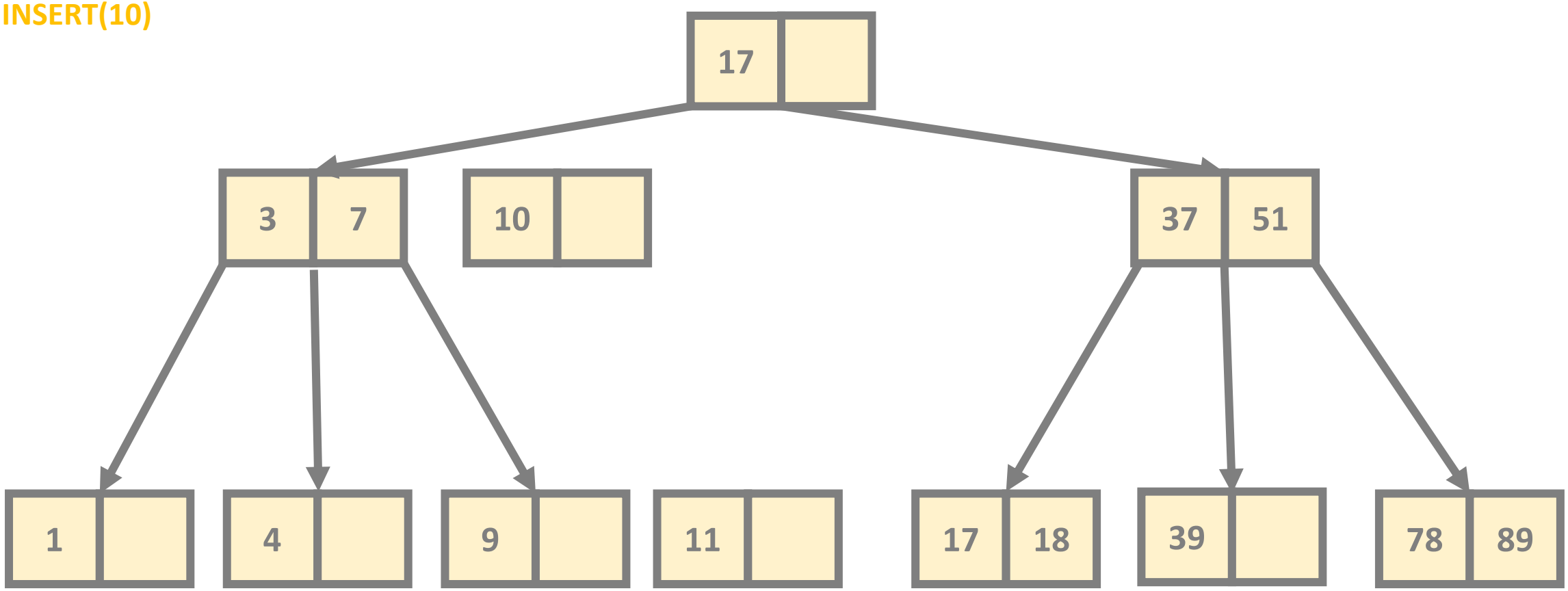
B-Trees

INSERT(10)



B-Trees

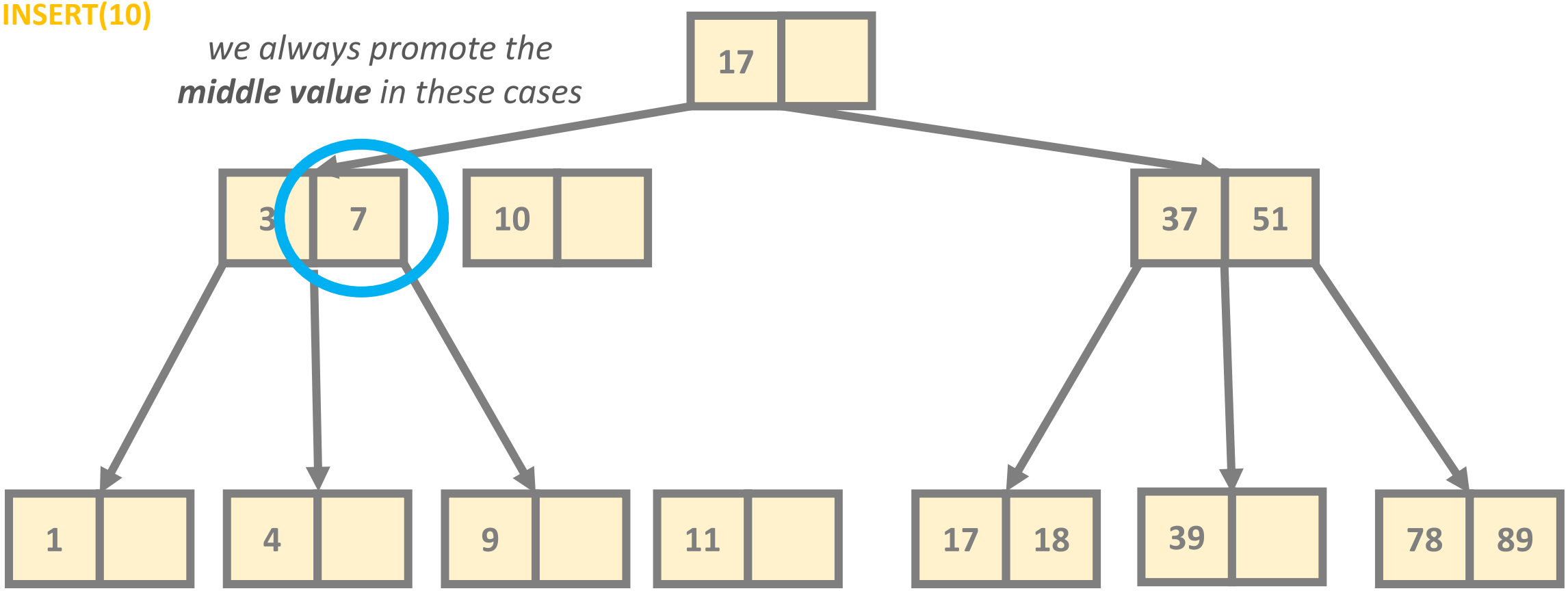
INSERT(10)



B-Trees

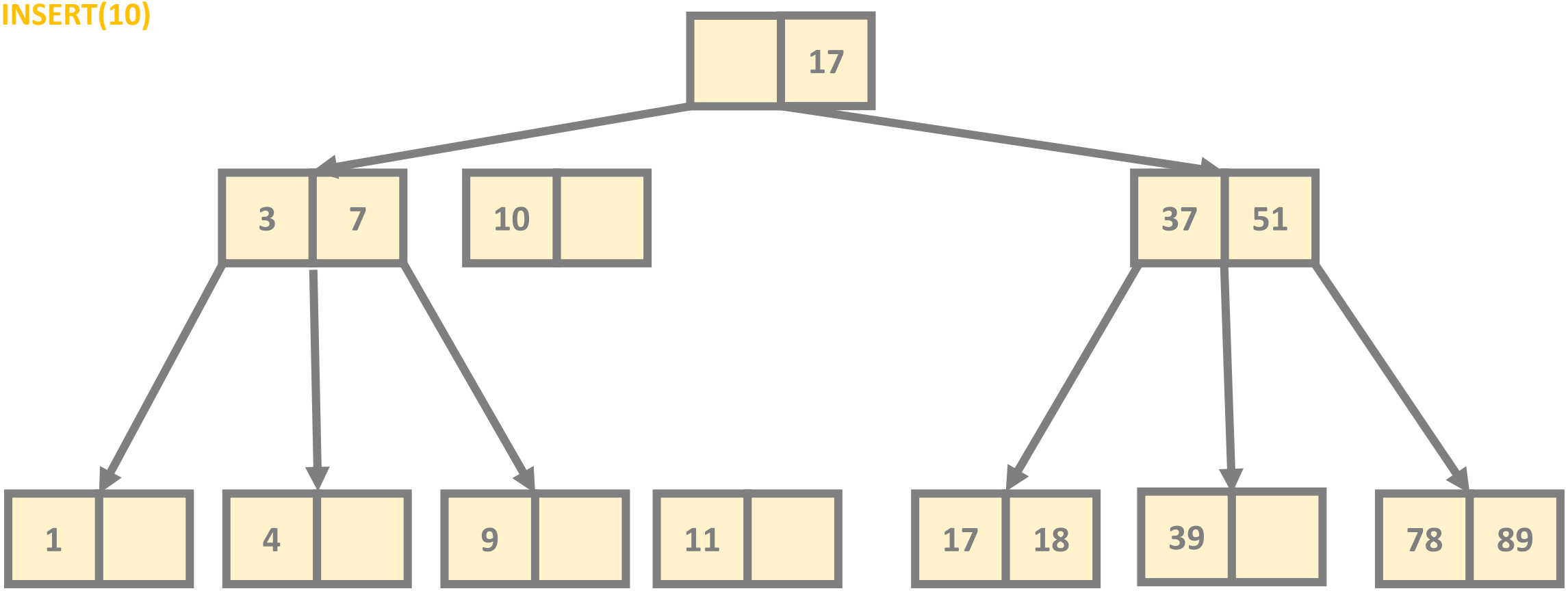
INSERT(10)

we always promote the ***middle value*** *in these cases*



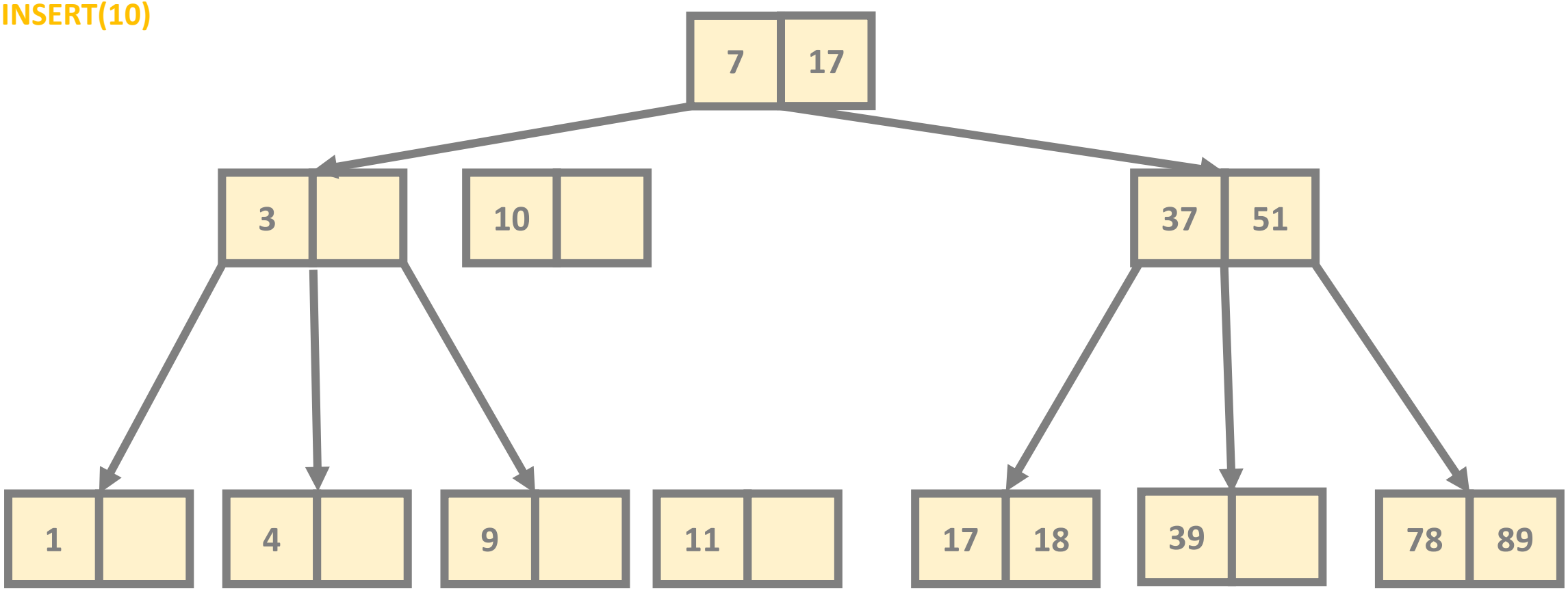
B-Trees

INSERT(10)



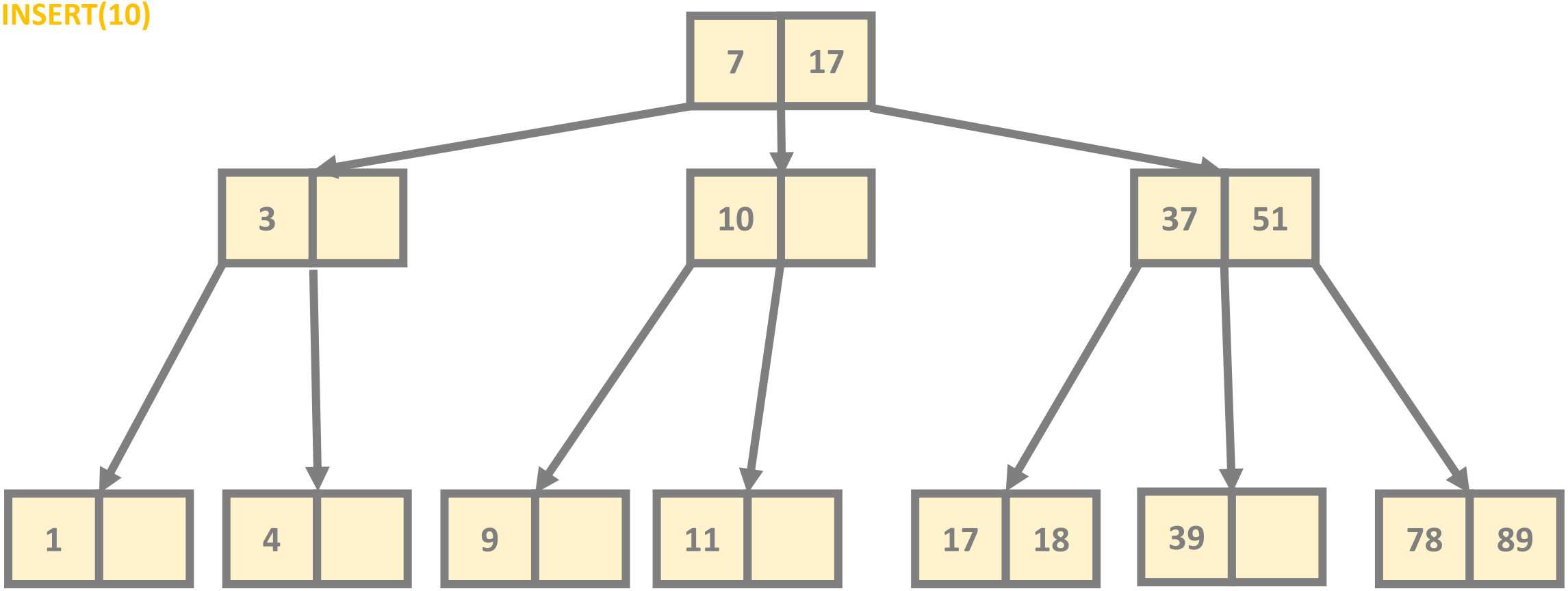
B-Trees

INSERT(10)



B-Trees

INSERT(10)



B-Trees Removal

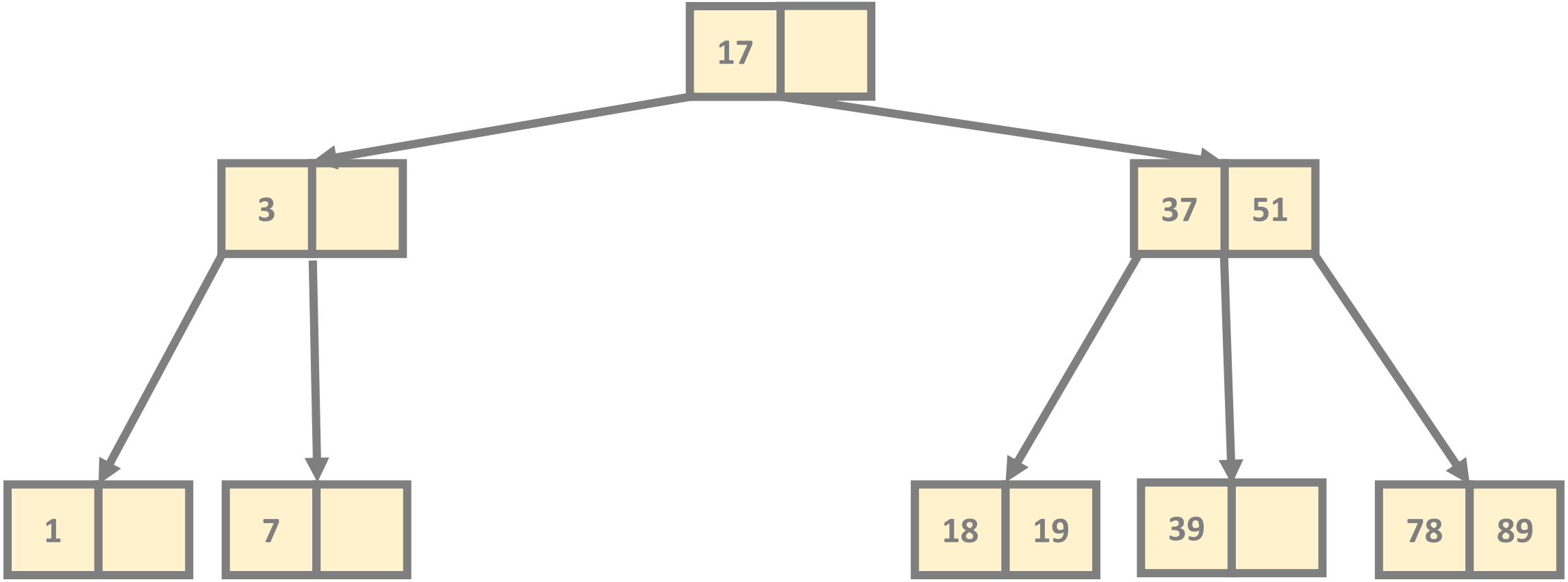
(Algorithms and Data Structures)

B-Tree Removal

1.) REMOVING AN INTERNAL ITEM

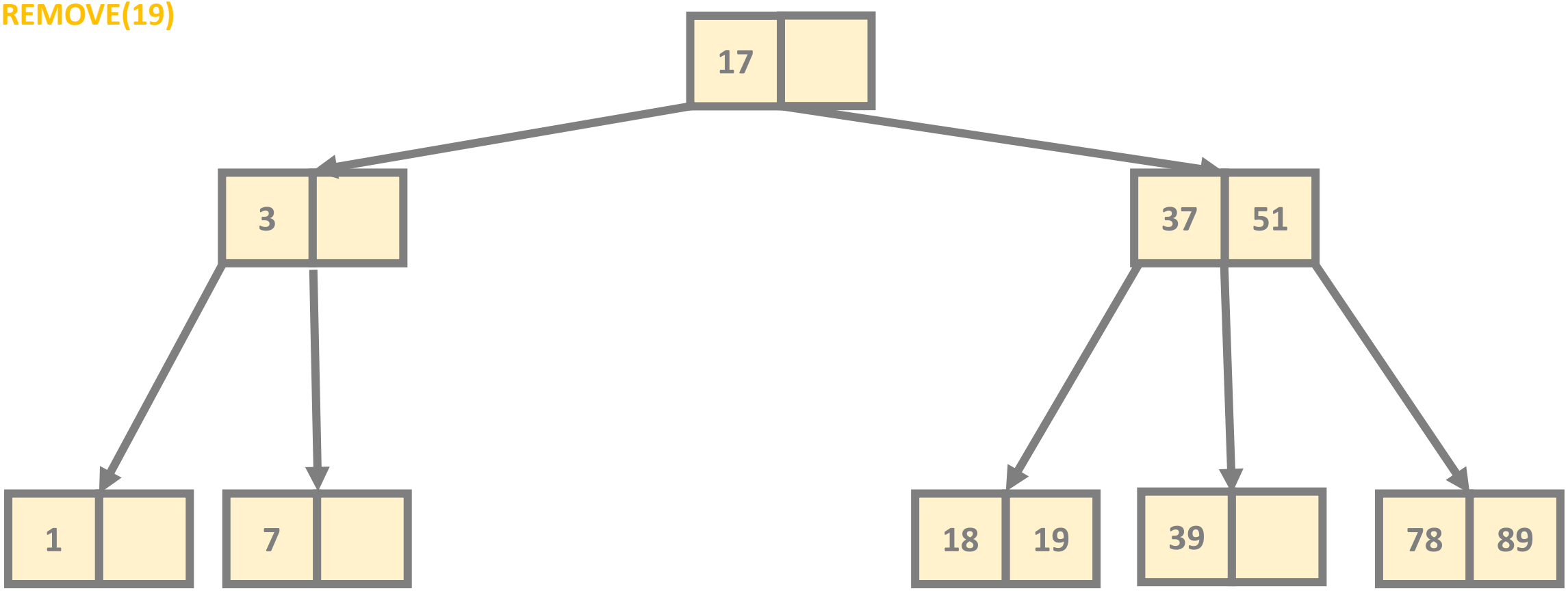
We remove an item from the node and does not violate the B-tree properties so the number of items remains in the range $\left[\frac{\mathbf{m}}{\mathbf{2}}, \mathbf{m}\right]$

B-Trees



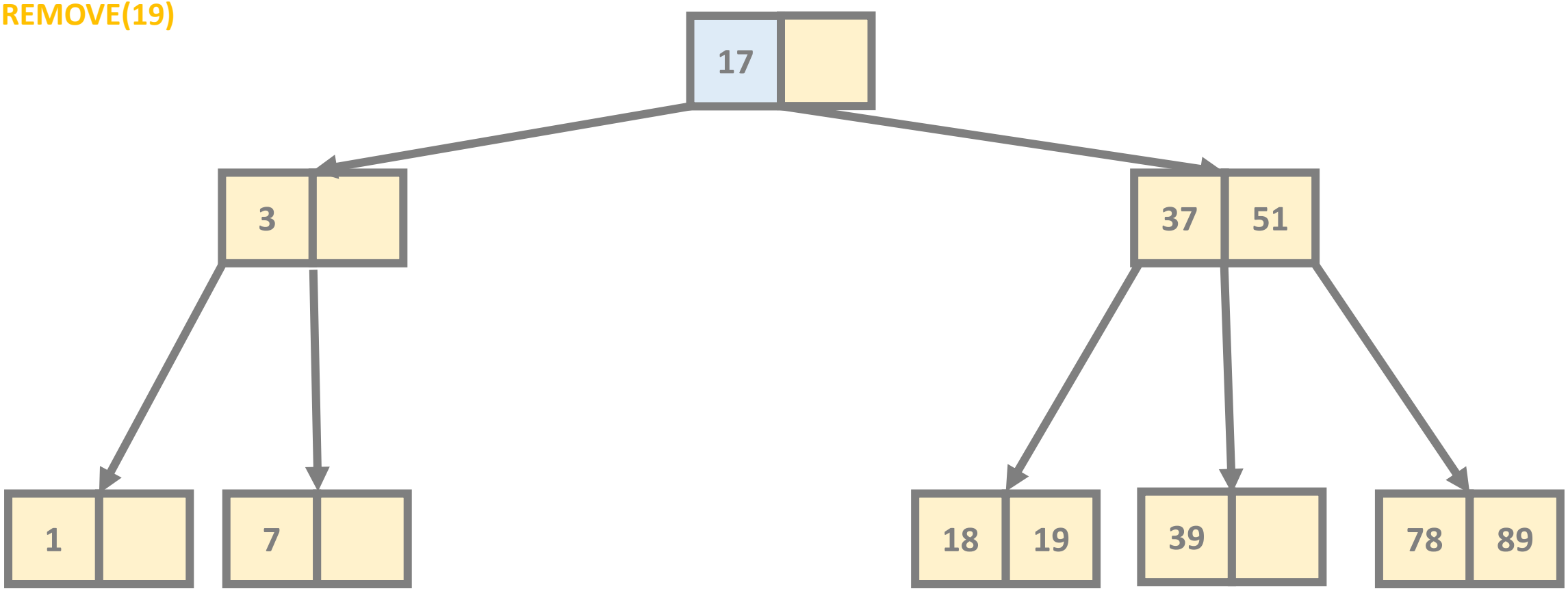
B-Trees

REMOVE(19)



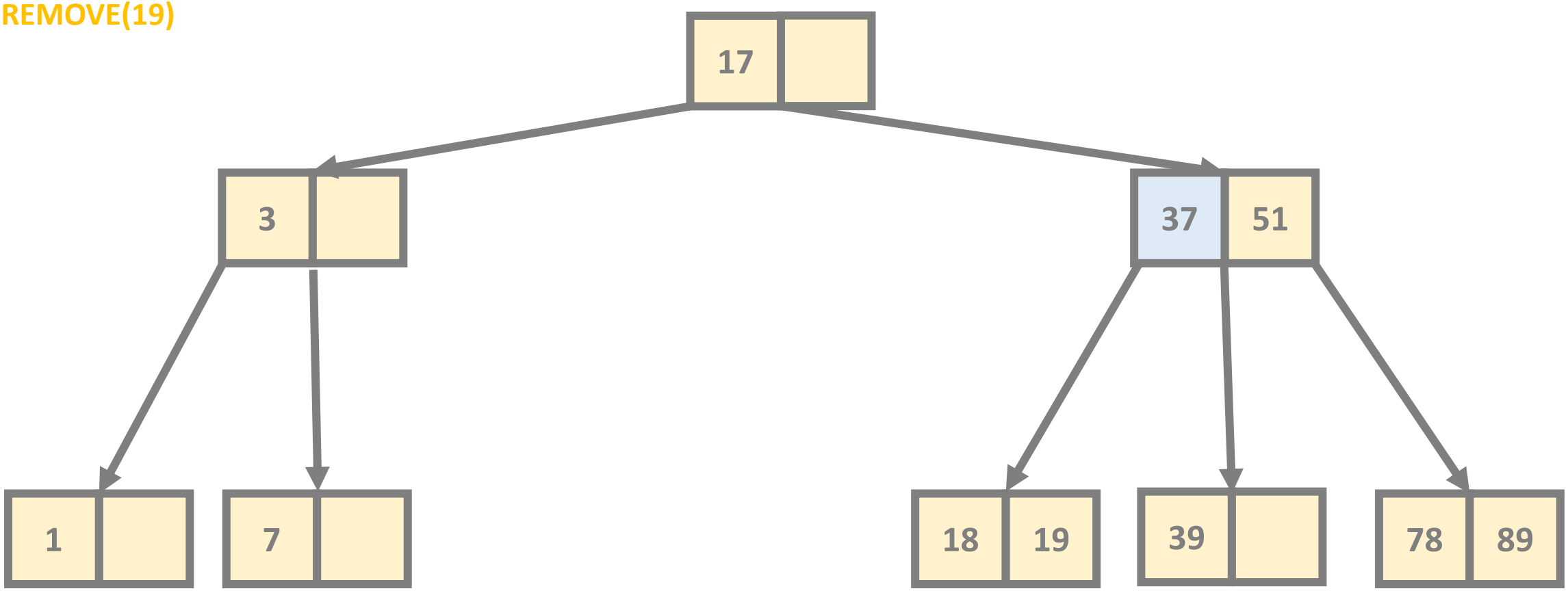
B-Trees

REMOVE(19)



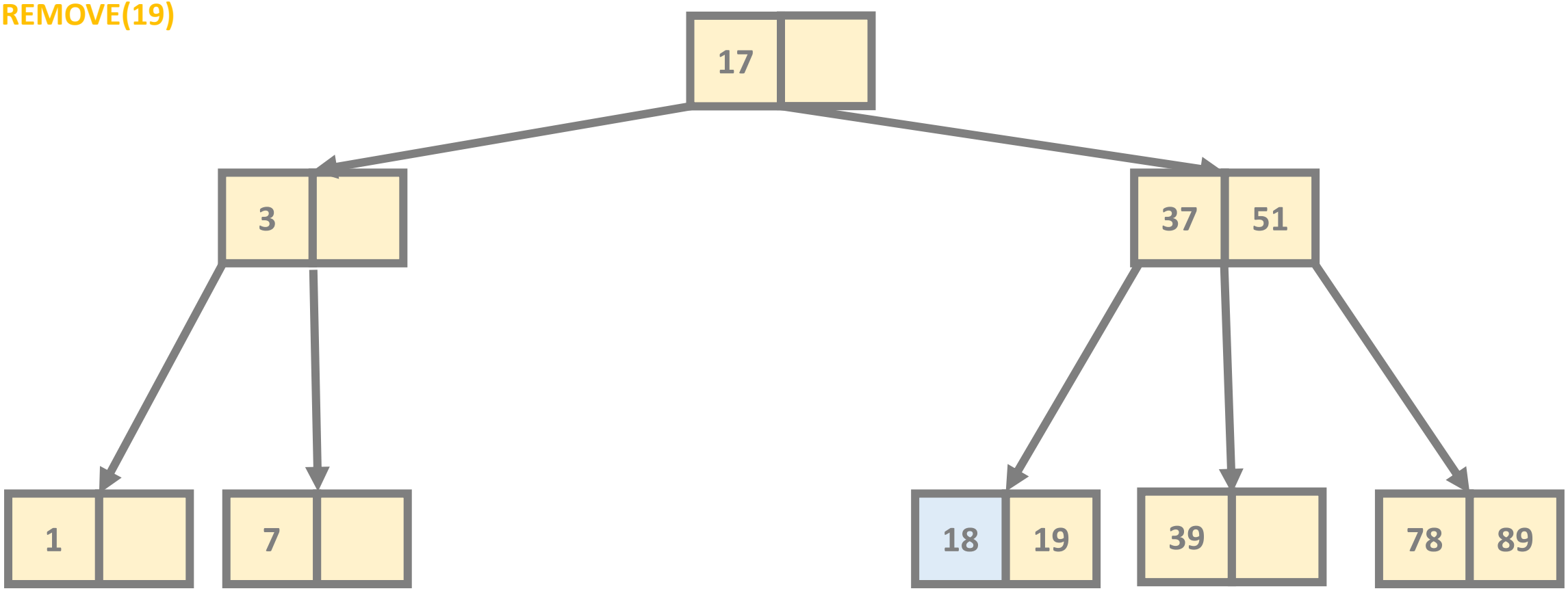
B-Trees

REMOVE(19)



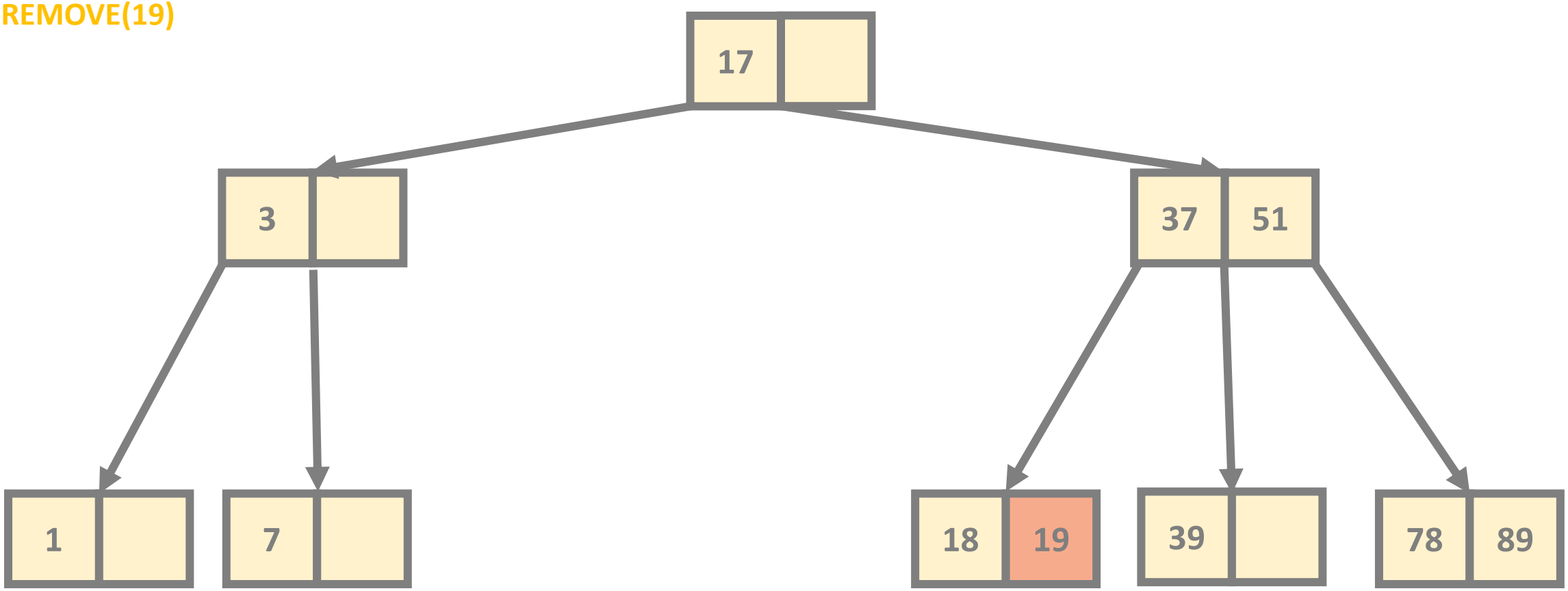
B-Trees

REMOVE(19)

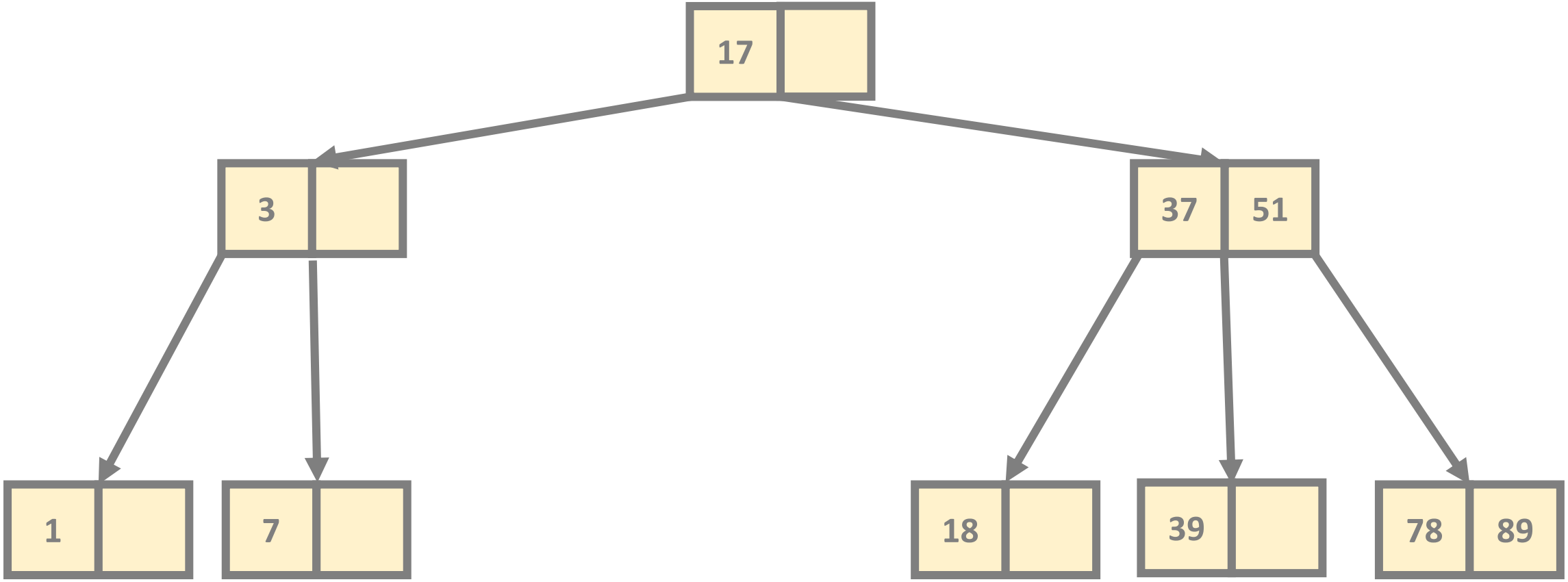


B-Trees

REMOVE(19)



B-Trees



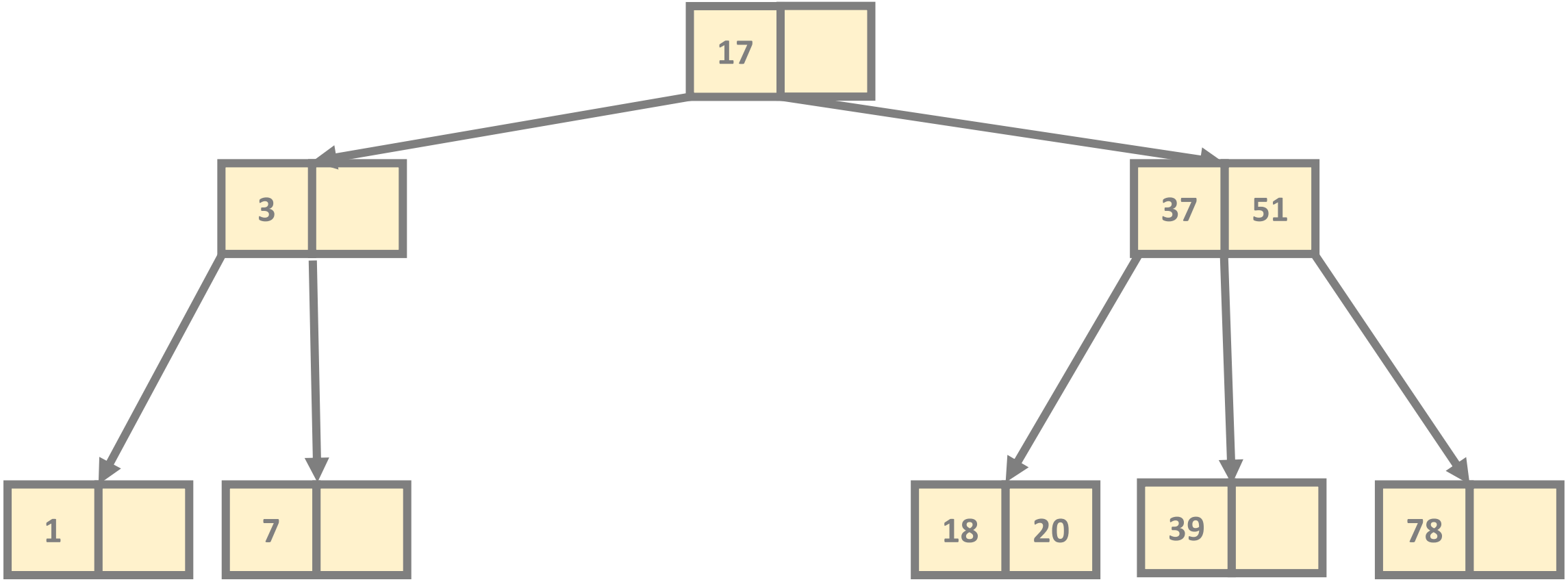
B-Tree Removal

2.) REMOVING A INTERNAL ITEM

We remove an item from the node and the **B-tree properties are violated** as there will be less than $\frac{\mathbf{m}}{\mathbf{2}}$ items in the given node

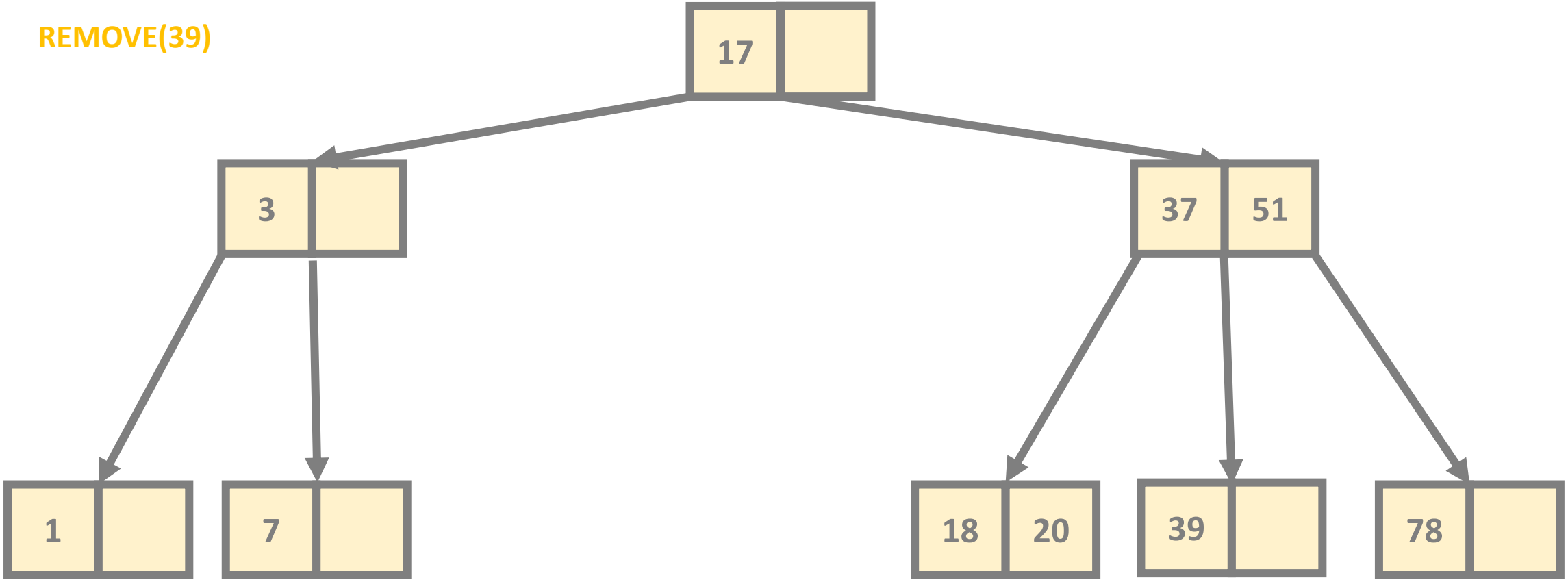
WE CAN GET AN ITEM FROM THE LEFT SIBLING !!!

B-Tree



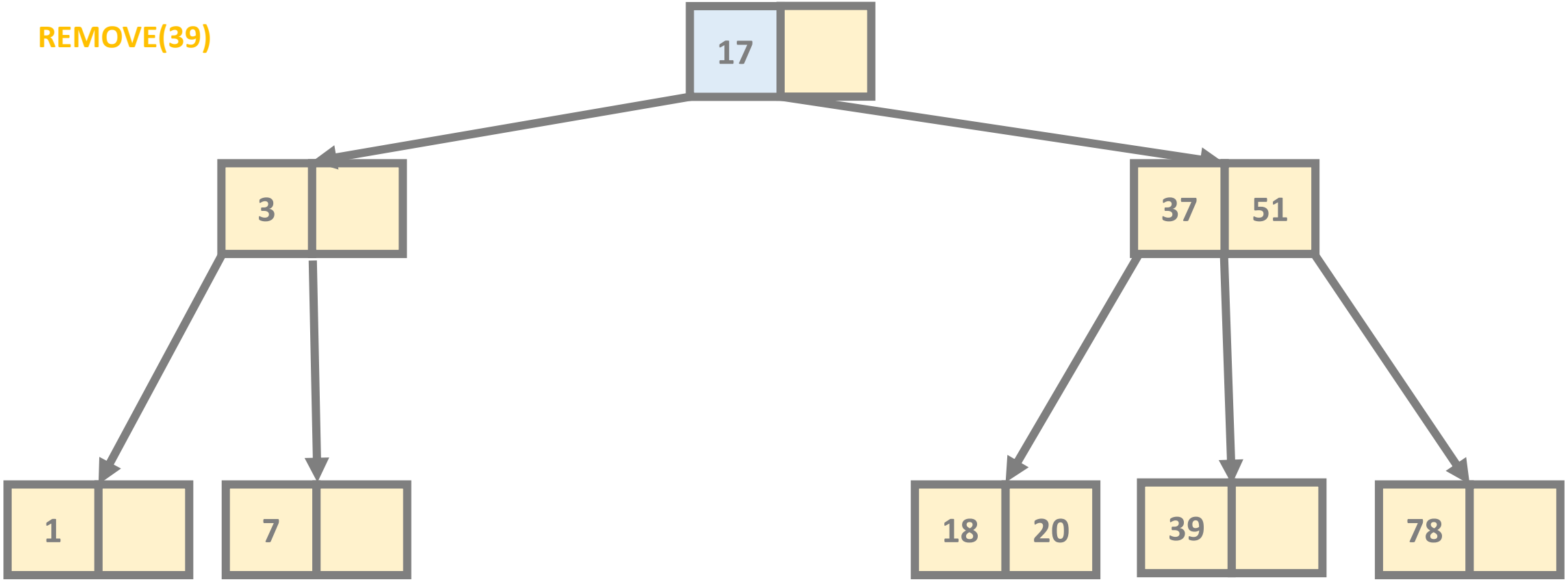
B-Tree

REMOVE(39)



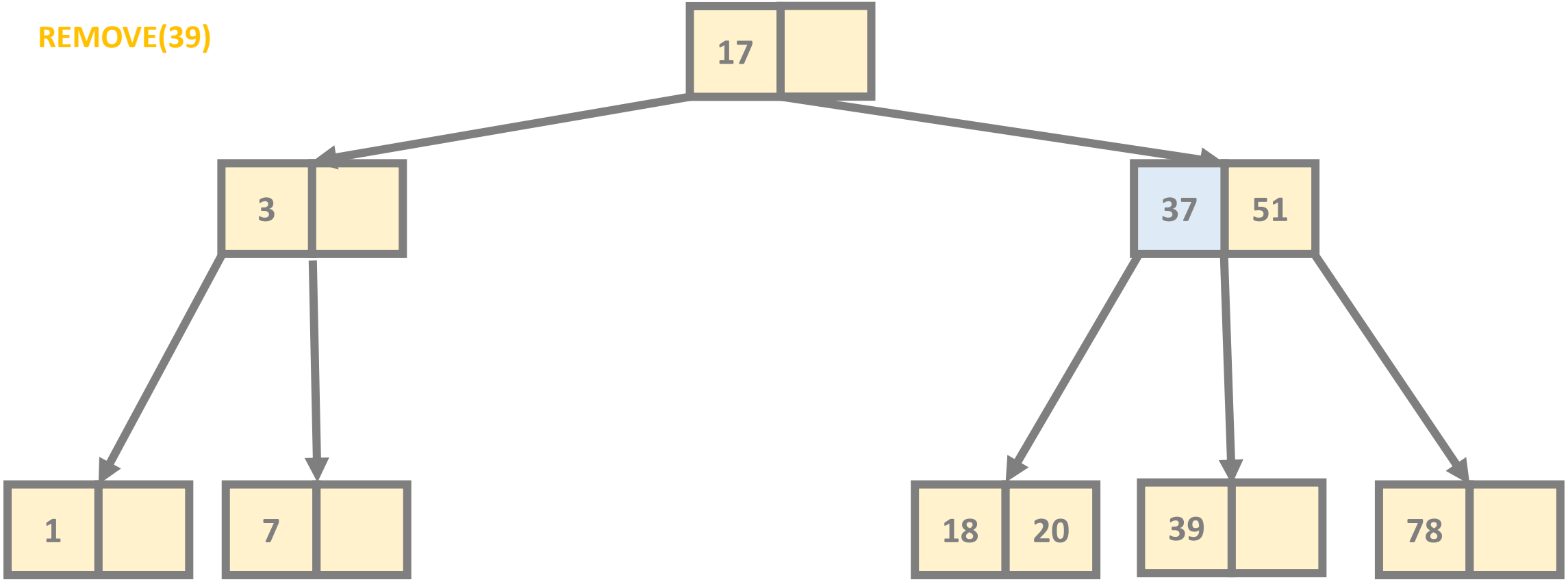
B-Tree

REMOVE(39)



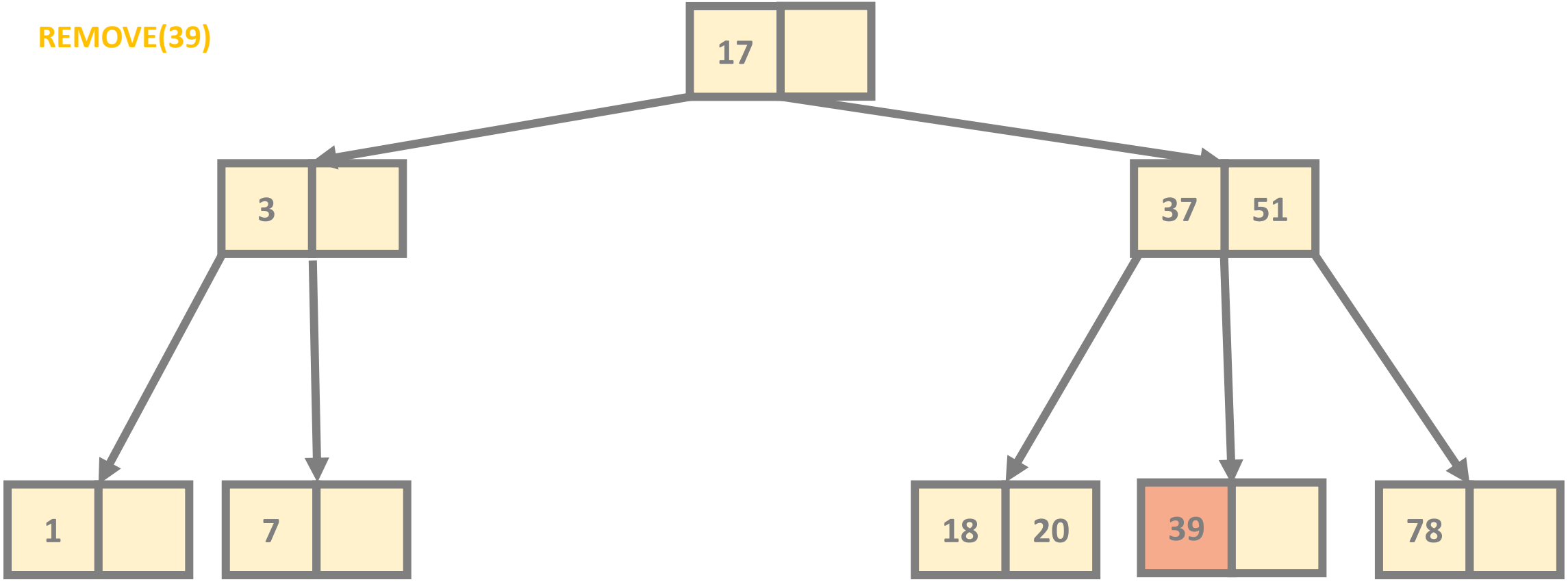
B-Tree

REMOVE(39)



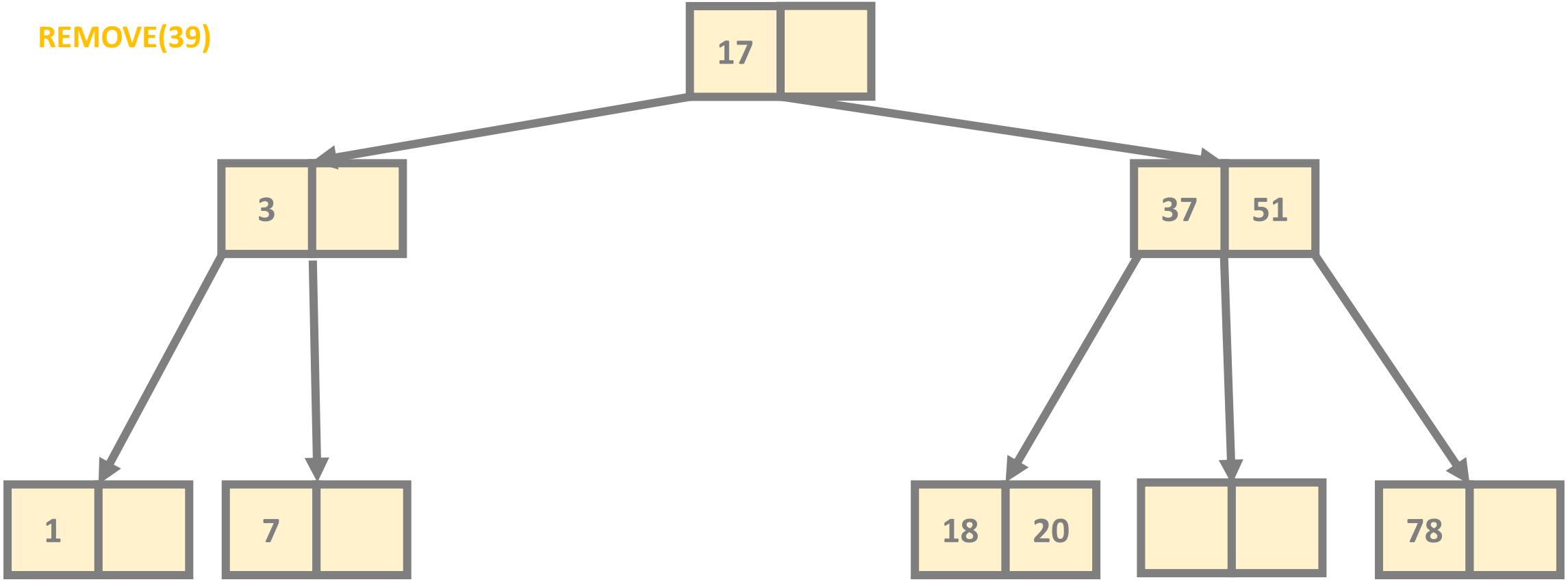
B-Tree

REMOVE(39)



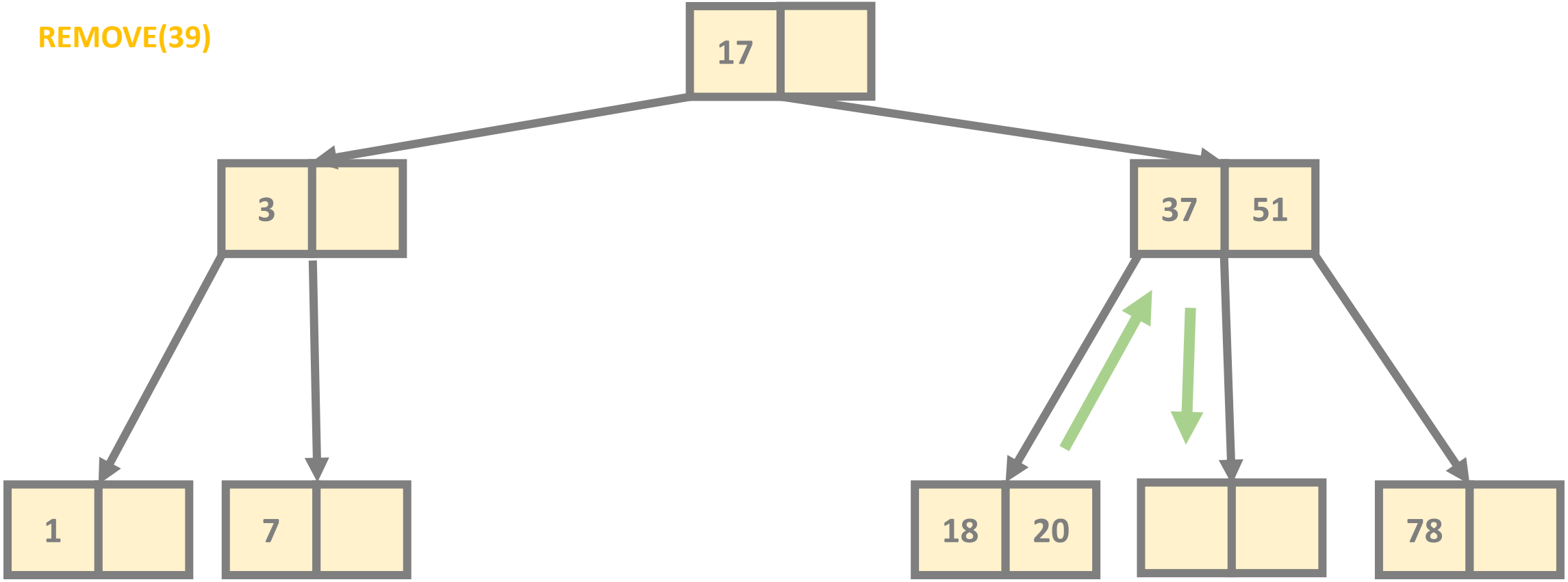
B-Tree

REMOVE(39)



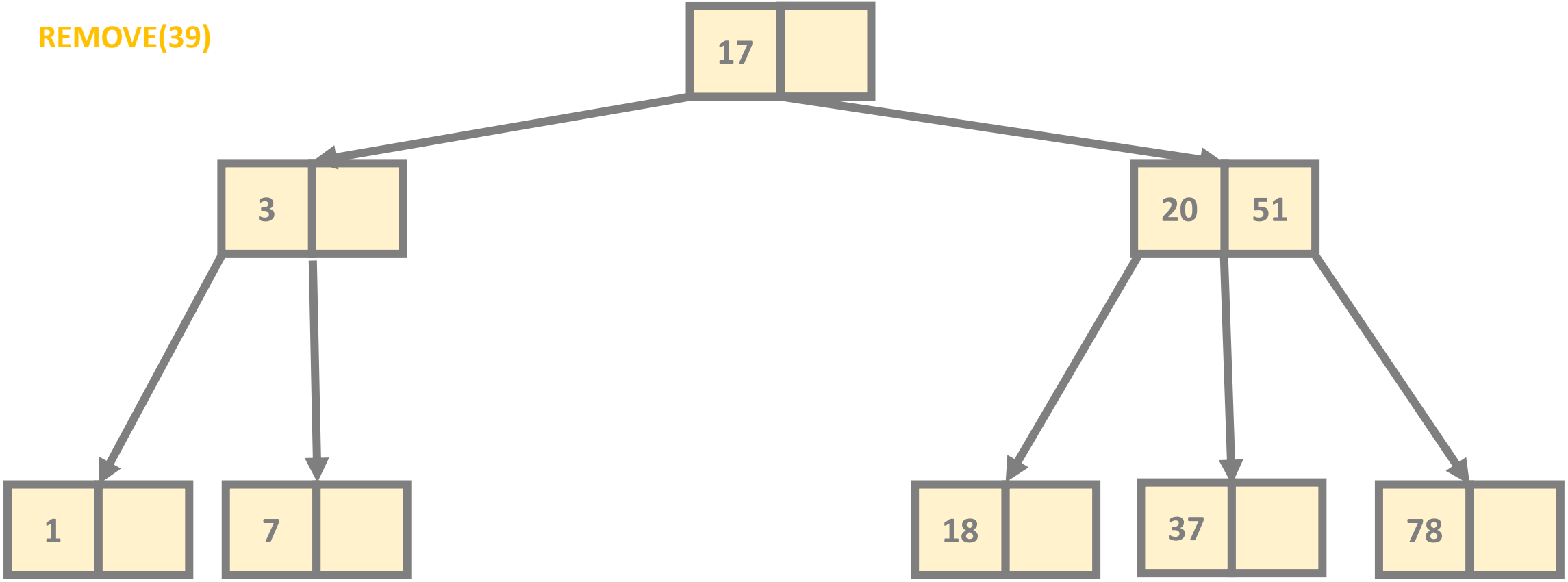
B-Tree

REMOVE(39)

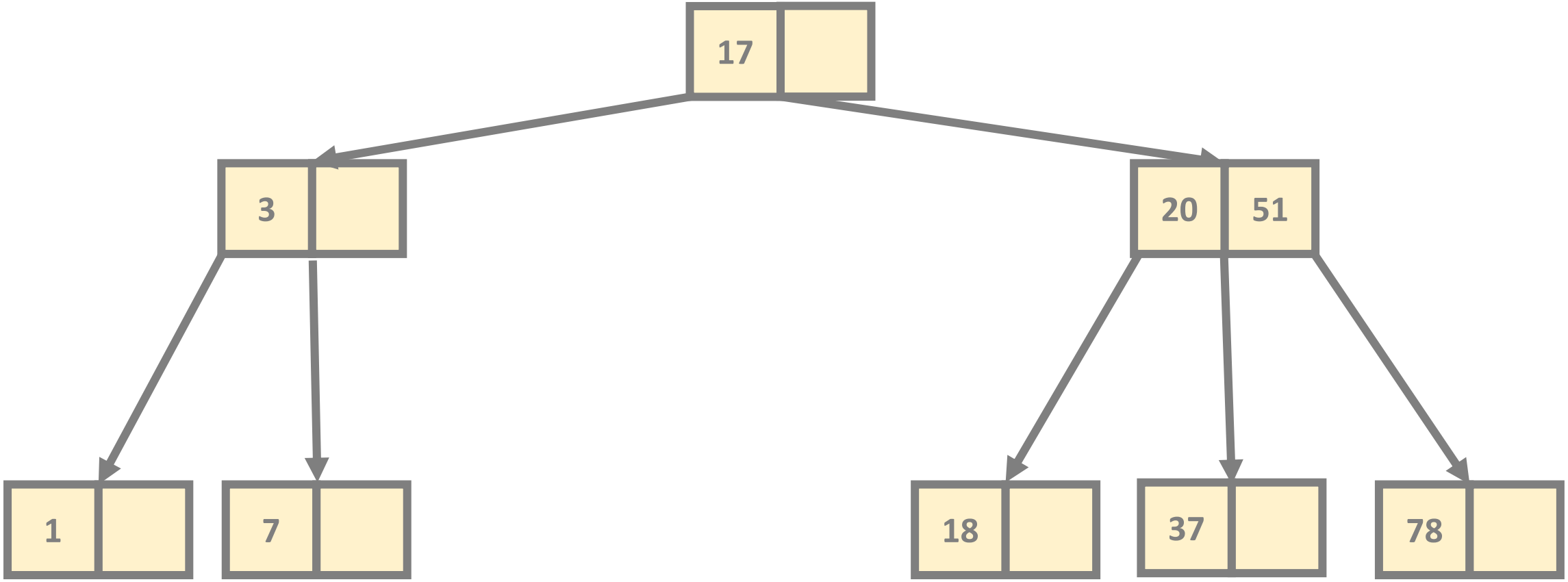


B-Tree

REMOVE(39)



B-Tree



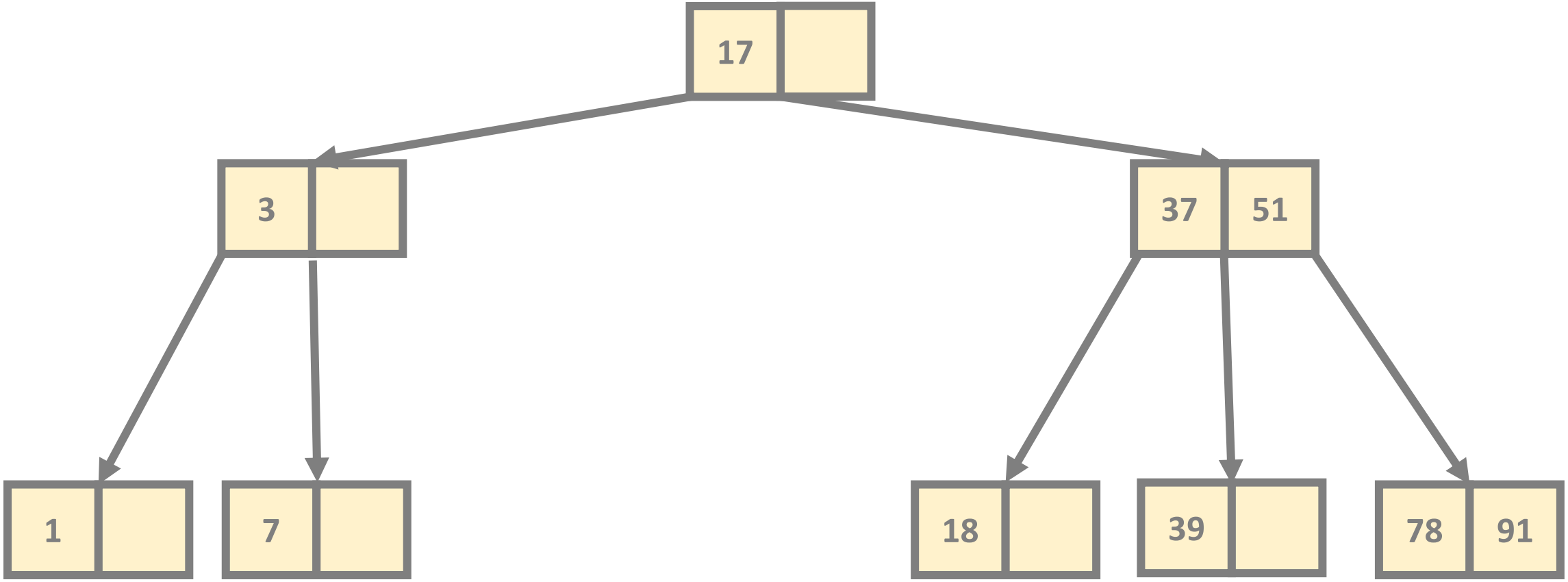
B-Tree Removal

3.) REMOVING A INTERNAL ITEM

We remove an item from the node and the **B-tree properties are violated** as there will be less than $\frac{\mathbf{m}}{\mathbf{2}}$ items in the given node

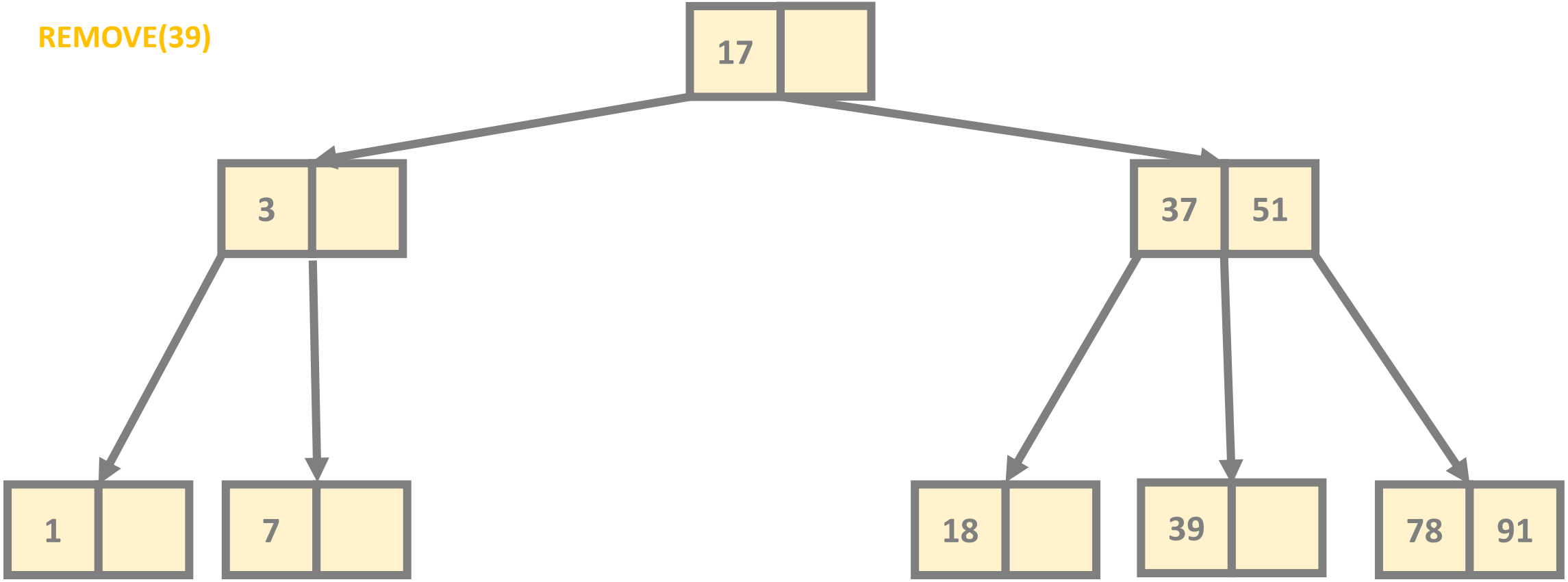
WE CAN GET AN ITEM FROM THE RIGHT SIBLING !!!

B-Tree



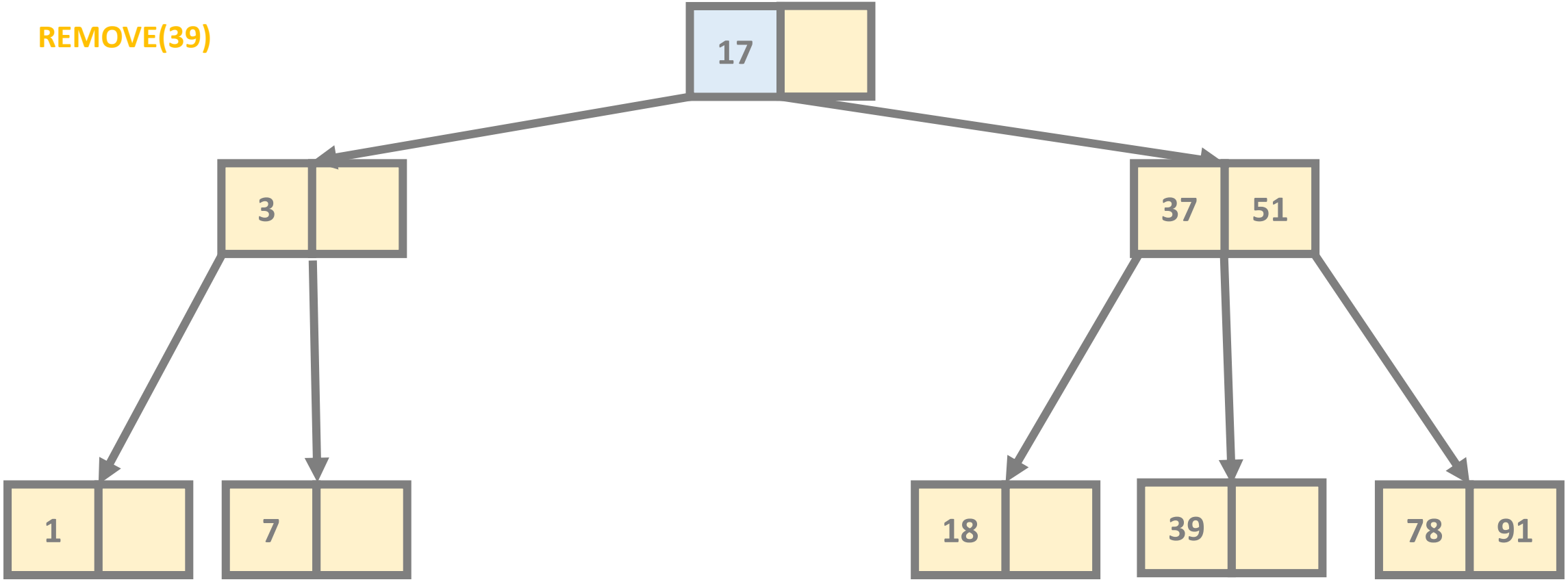
B-Tree

REMOVE(39)



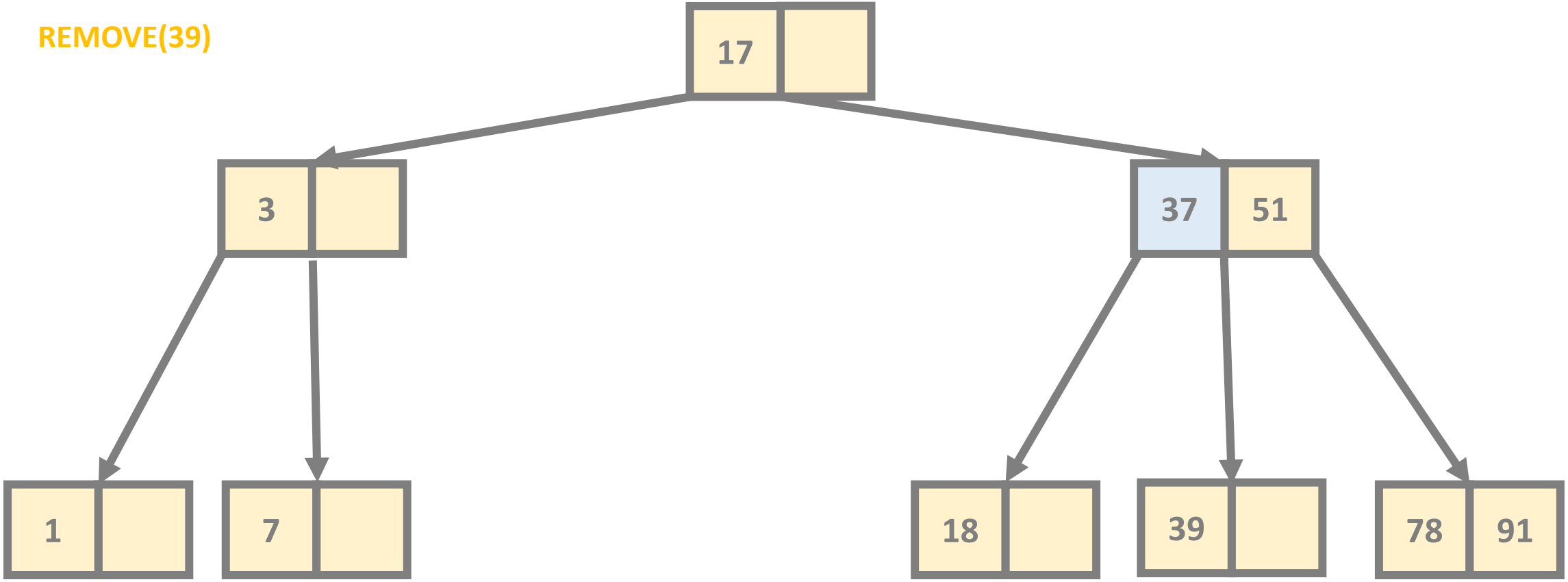
B-Tree

REMOVE(39)



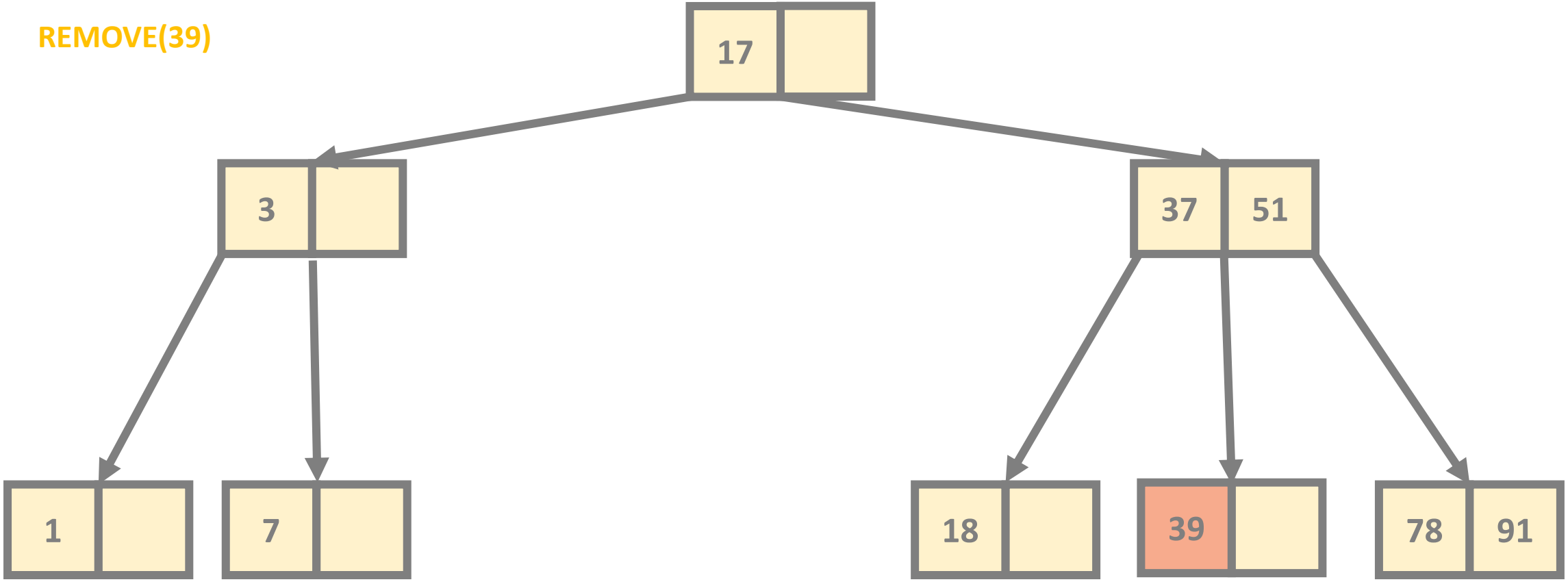
B-Tree

REMOVE(39)



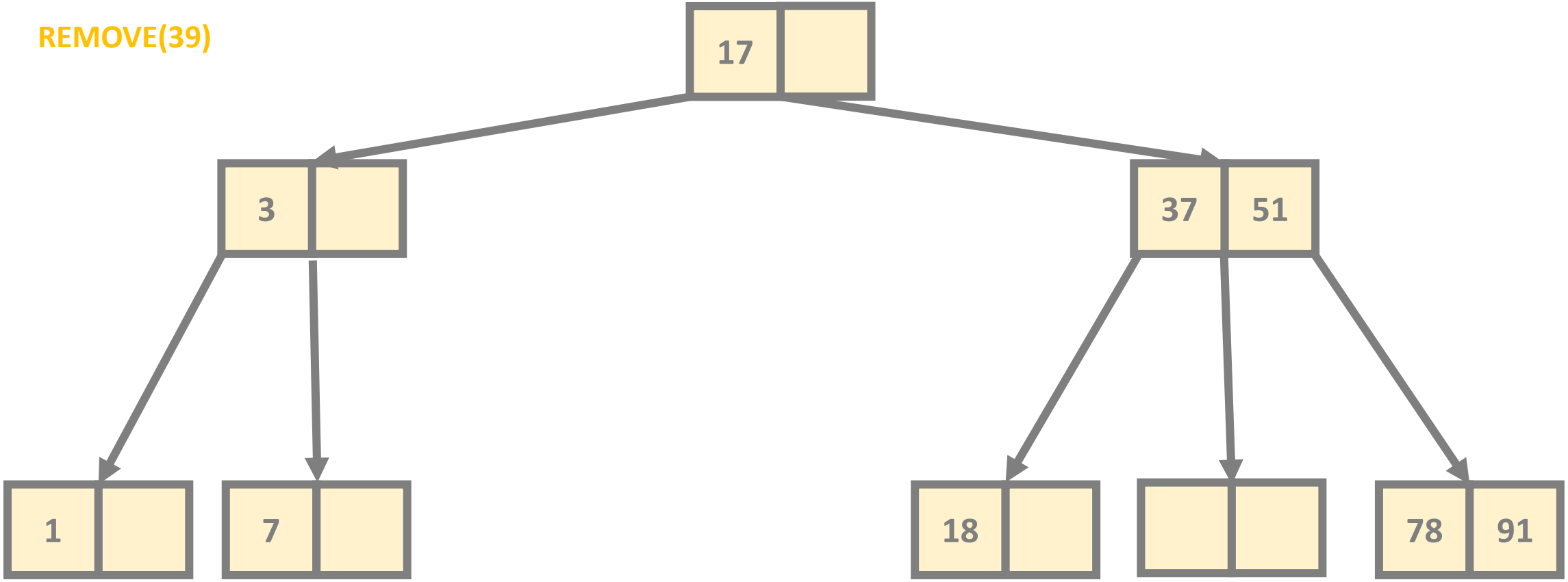
B-Tree

REMOVE(39)



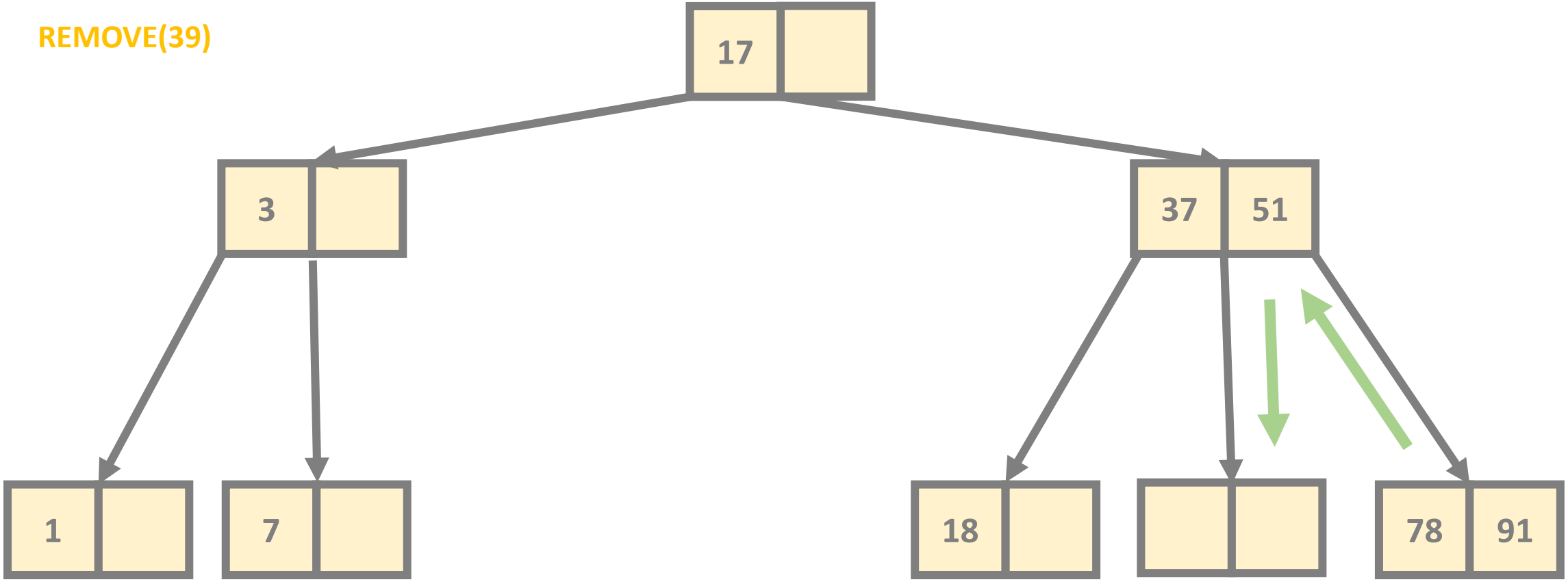
B-Tree

REMOVE(39)



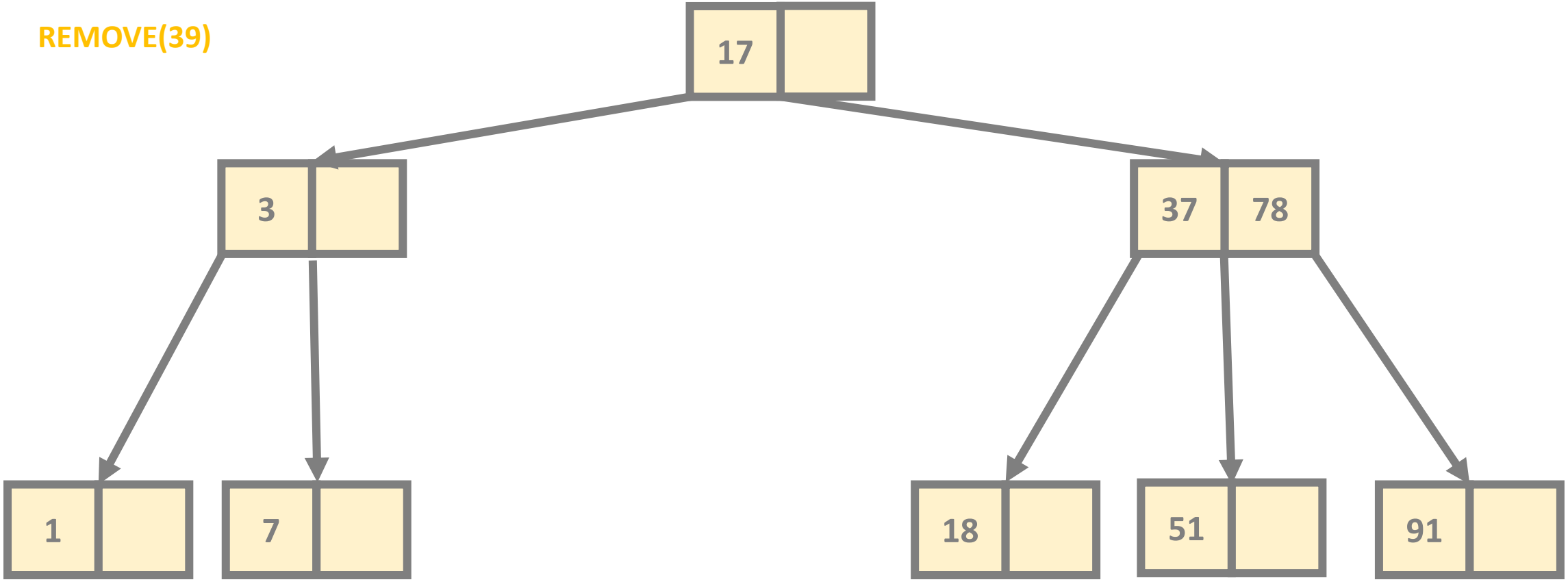
B-Tree

REMOVE(39)

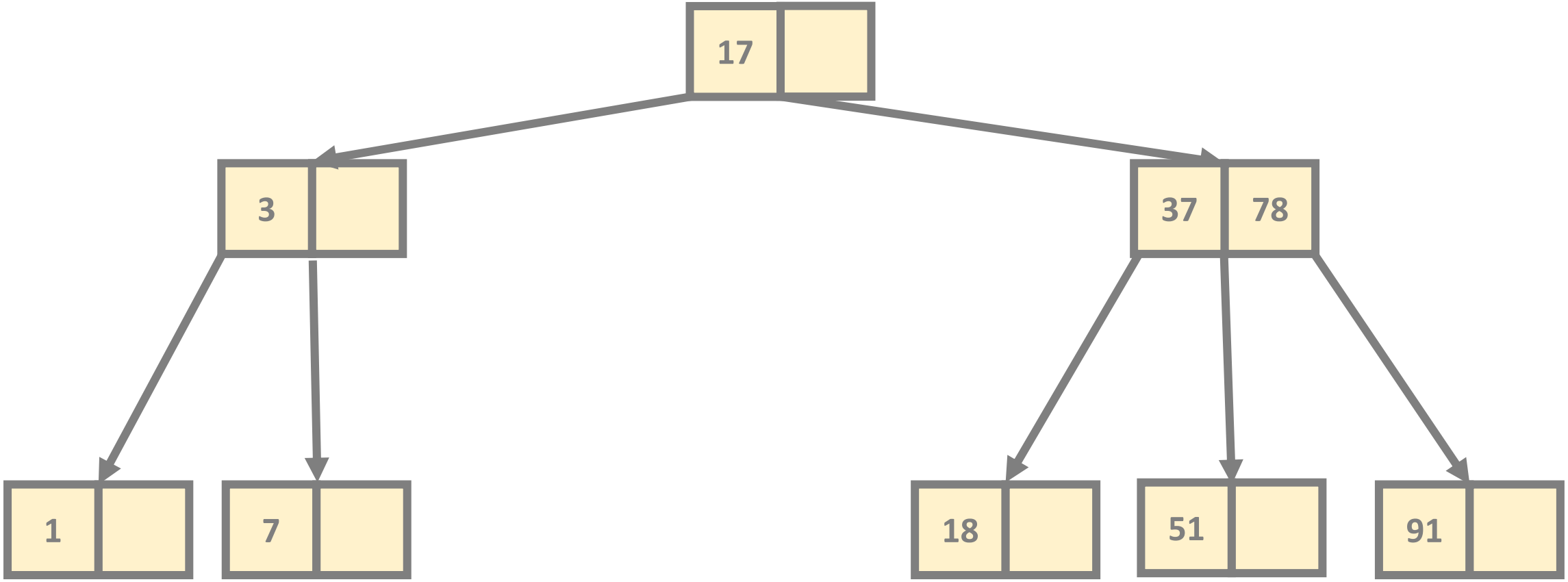


B-Tree

REMOVE(39)



B-Tree



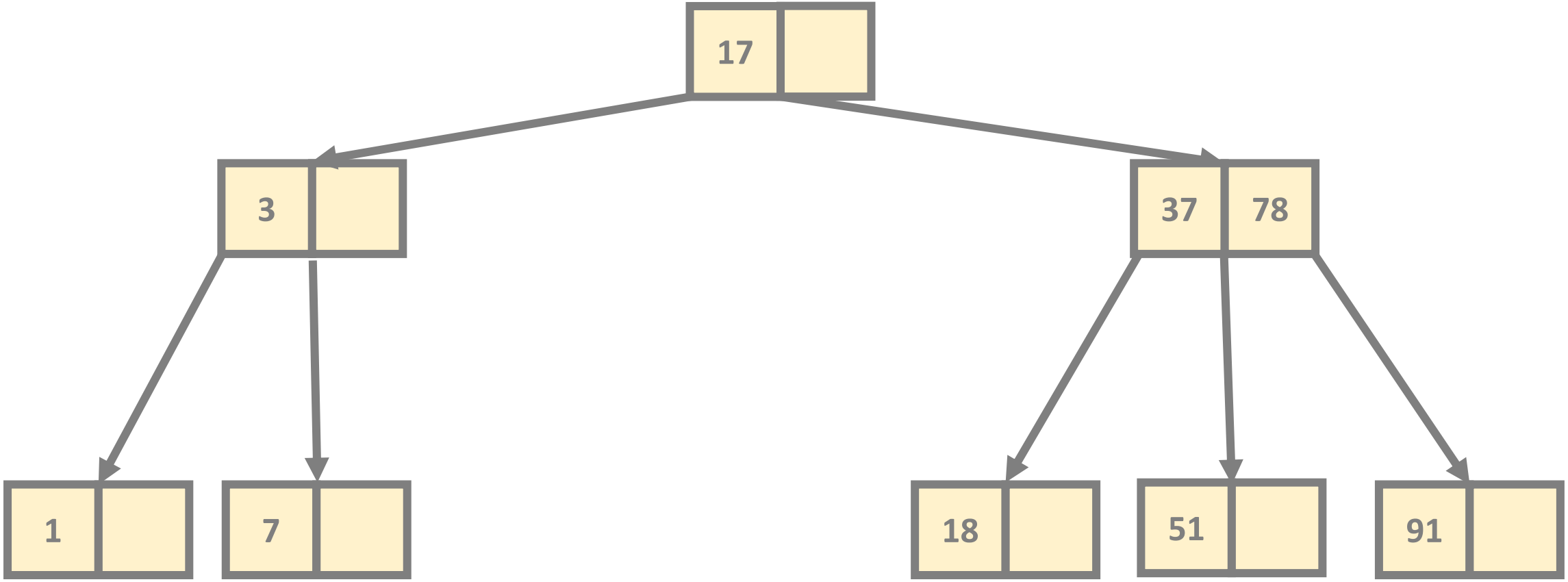
B-Tree Removal

4.) REMOVING A INTERNAL ITEM

We remove an item from the node and the **B-tree properties are violated** as there will be less than $\frac{\mathbf{m}}{\mathbf{2}}$ items in the given node

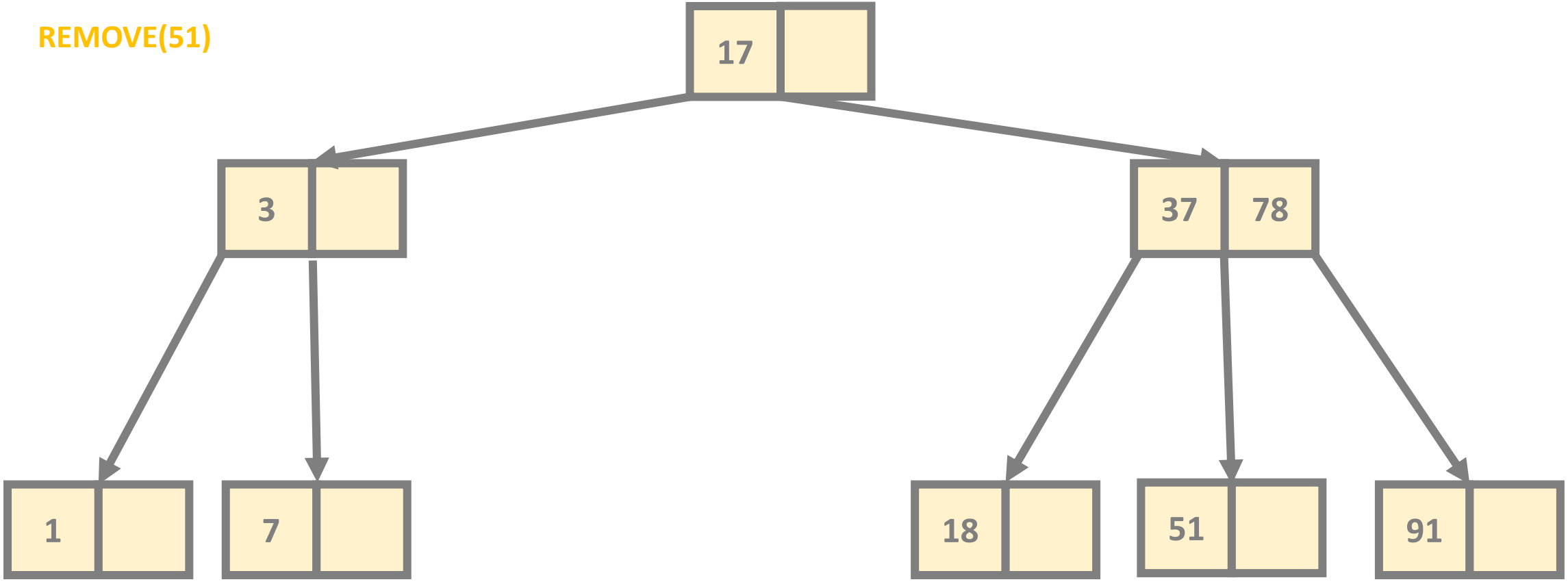
THERE IS NO ADDITIONAL ITEMS IN THE SIBLINGS SO WE HAVE NO OTHER OPTION BUT TO MERGE NODES !!!

B-Tree



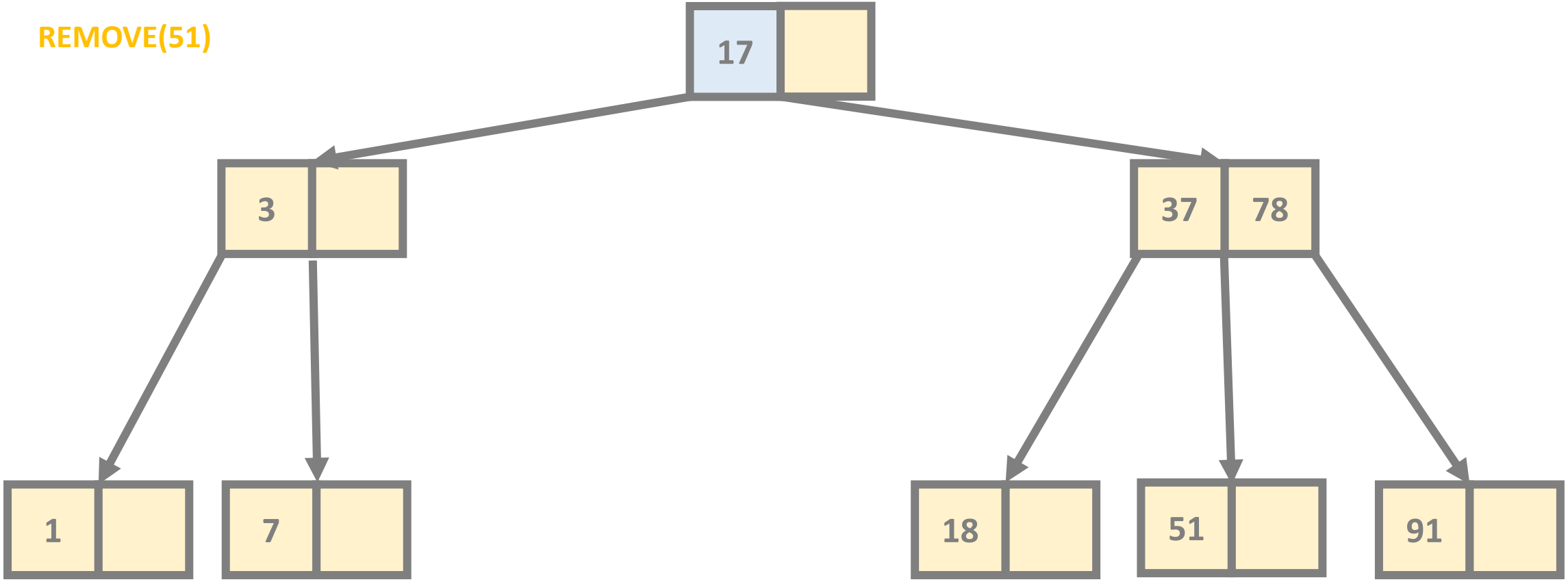
B-Tree

REMOVE(51)



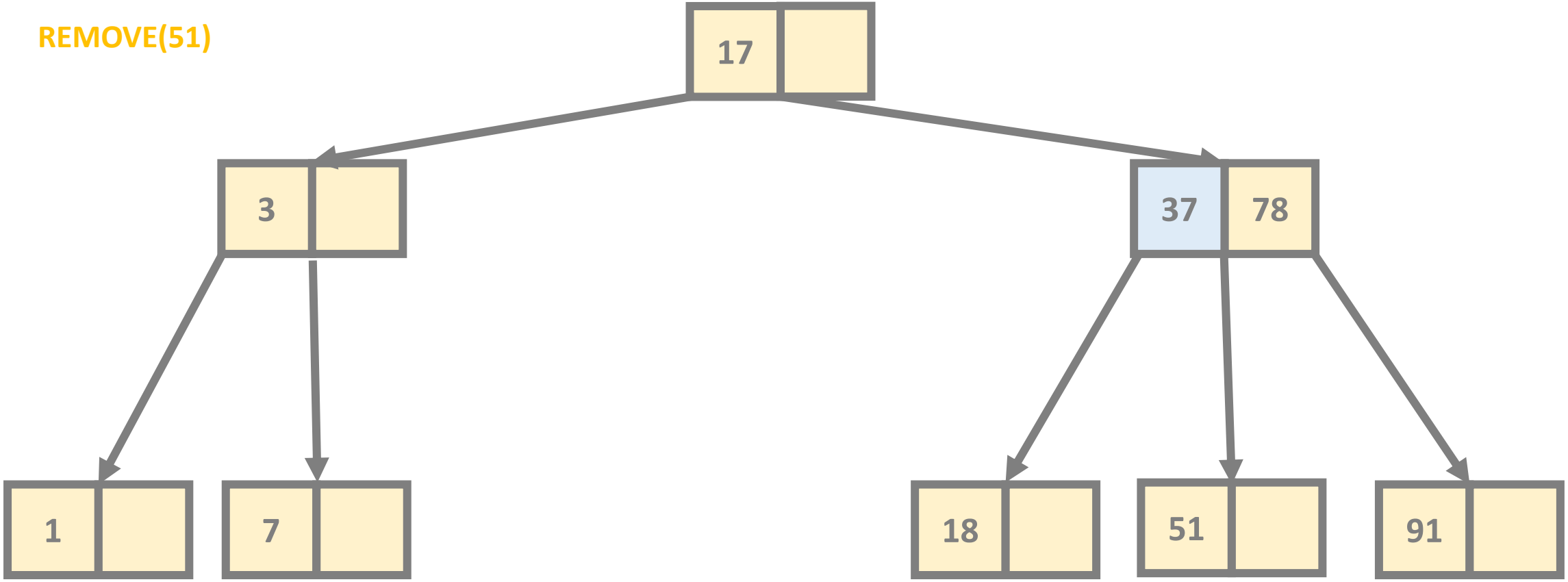
B-Tree

REMOVE(51)



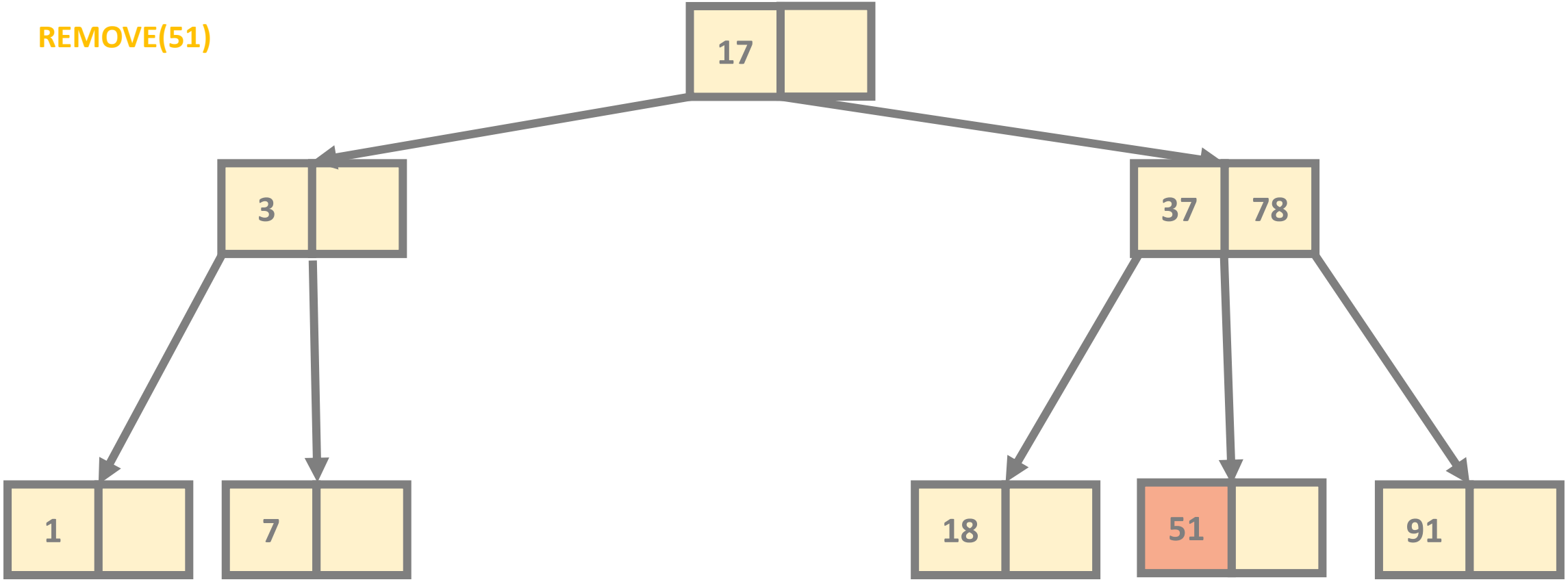
B-Tree

REMOVE(51)



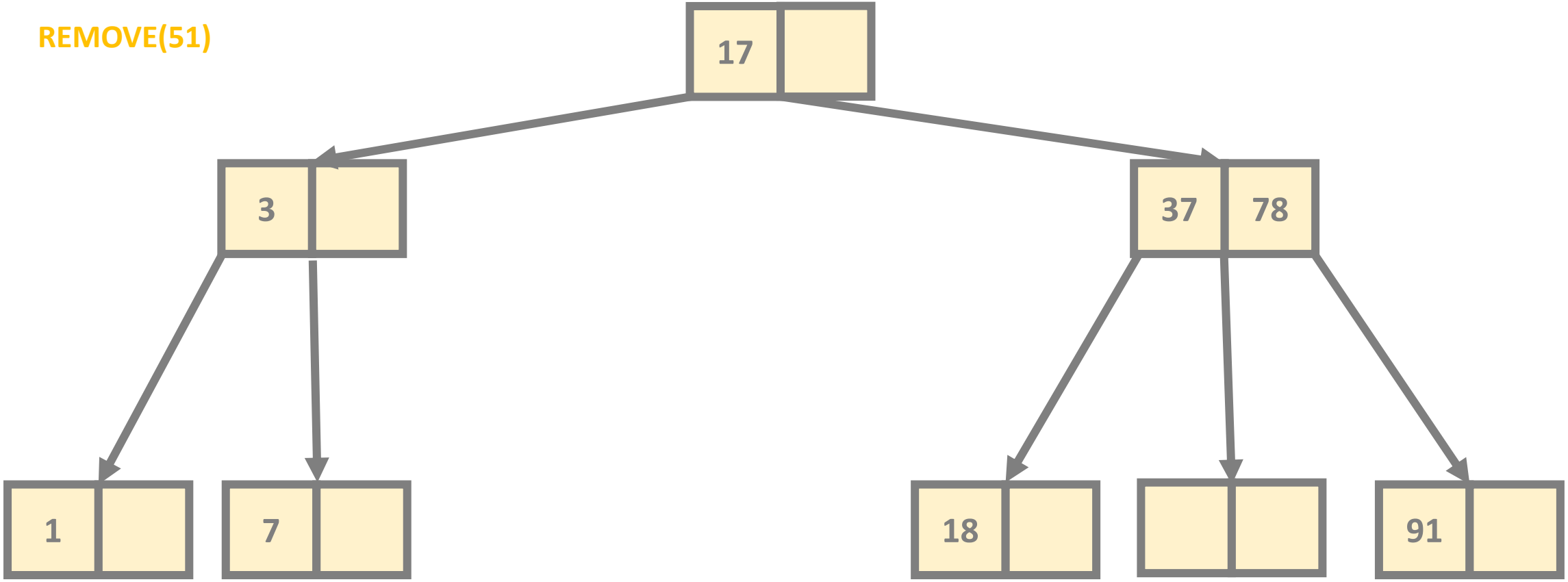
B-Tree

REMOVE(51)



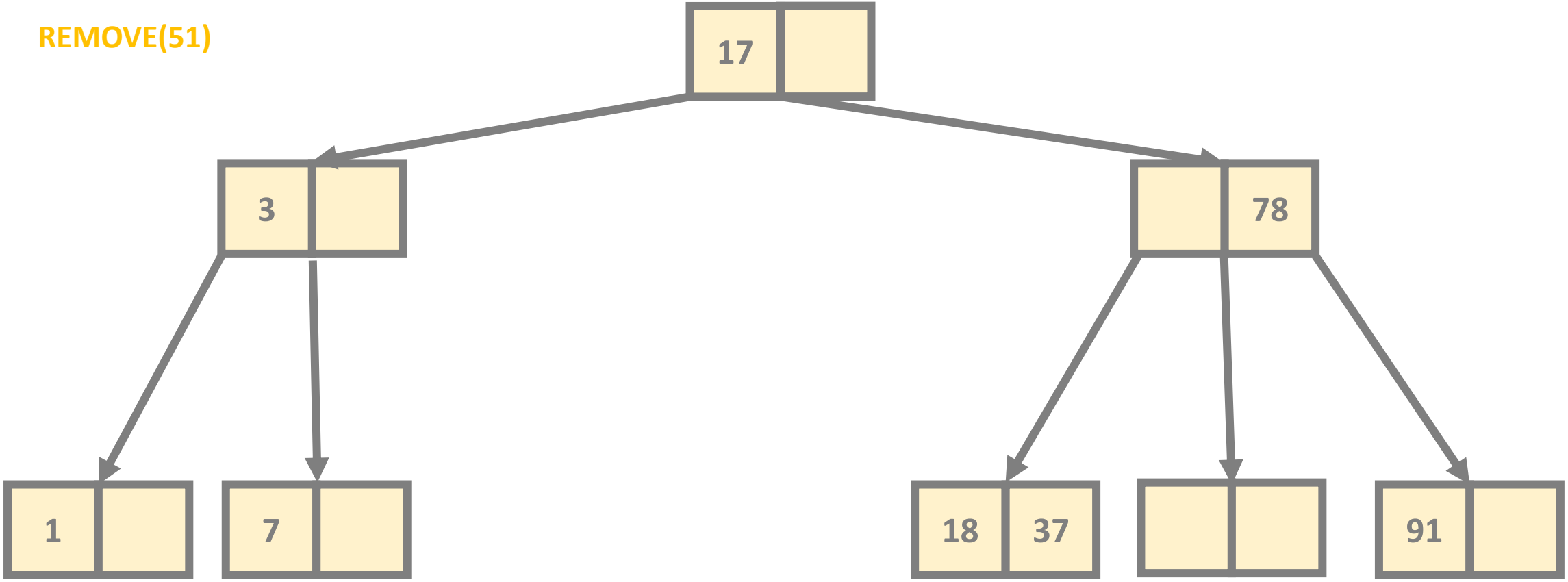
B-Tree

REMOVE(51)



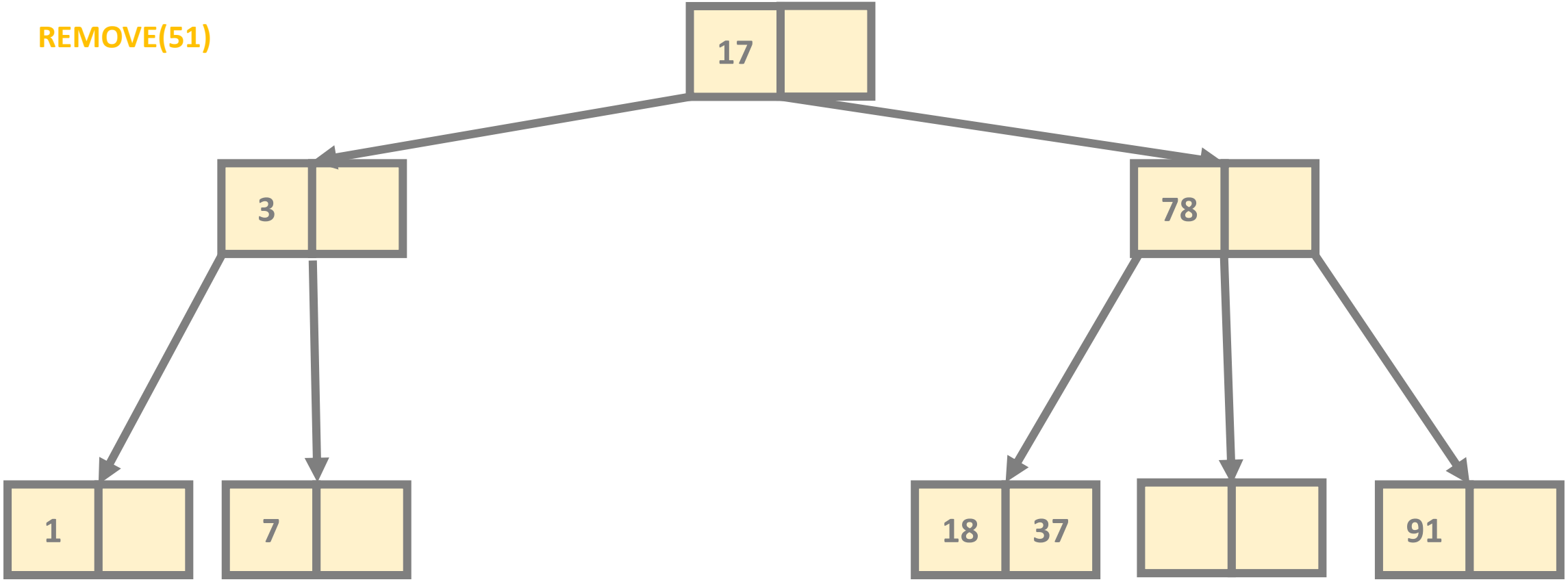
B-Tree

REMOVE(51)



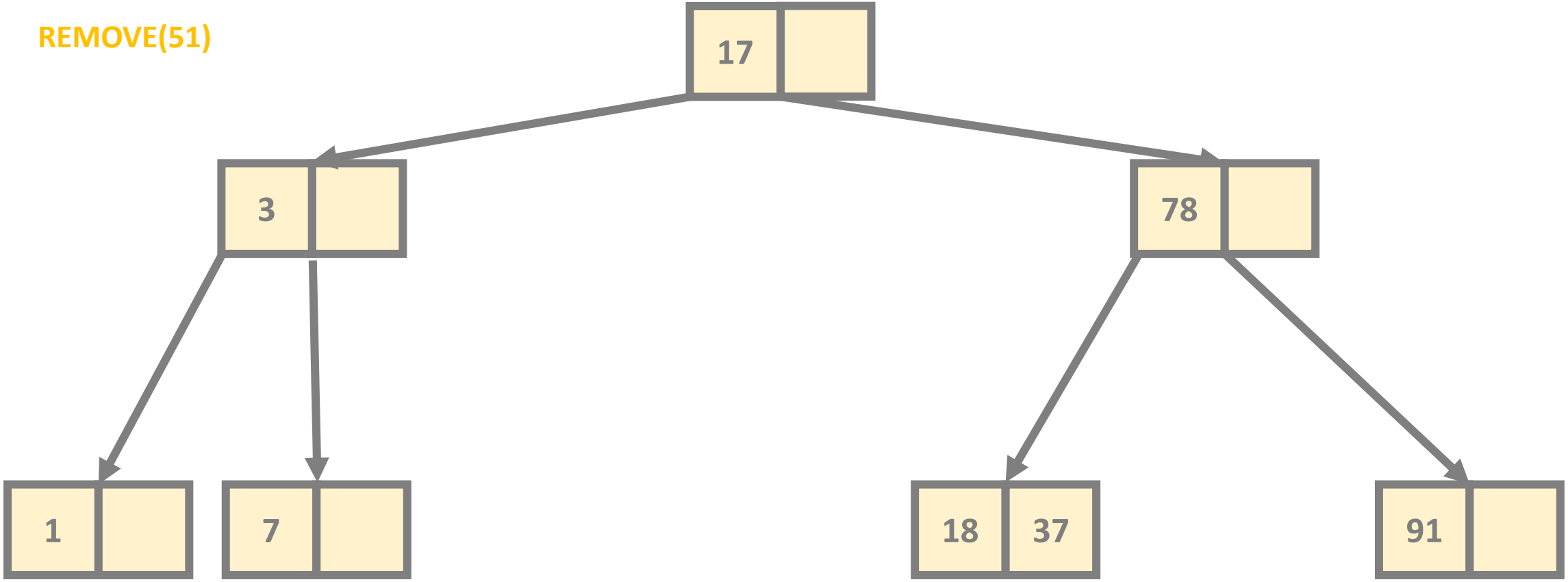
B-Tree

REMOVE(51)



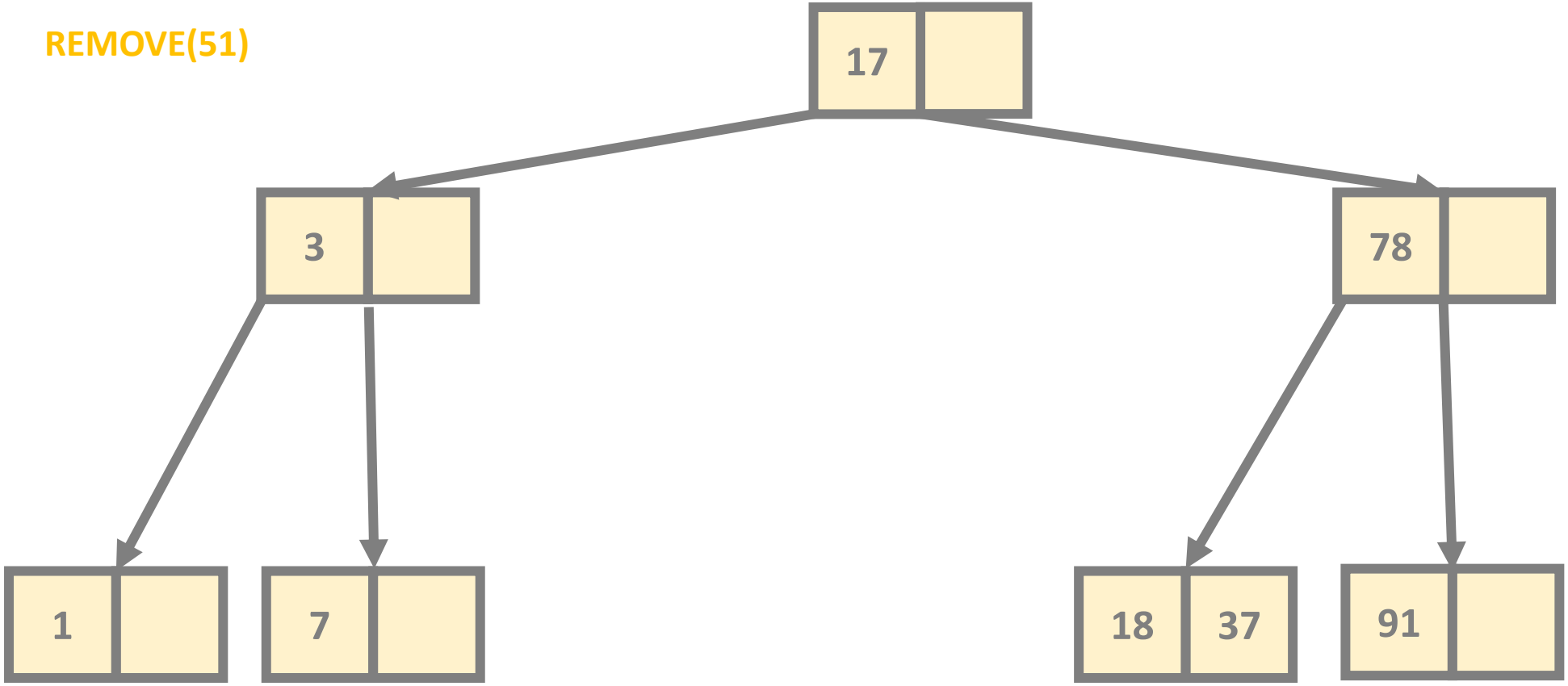
B-Tree

REMOVE(51)



B-Tree

REMOVE(51)



B-Tree Variants

(Algorithms and Data Structures)

B-Trees Advantages and Disadvantages

- there are several **advantages** of B-trees
- there is a guaranteed **O(logN)** logarithmic running time
- **BUT THERE MAY BE SEVERAL EMPTY CELLS**
- we can decrease the **h** height of the B-tree
- this is why better variants came to be

B* Trees

- B* trees keep each node at least $\frac{2}{3}$ full instead of just $\frac{1}{2}$ by redistributing keys until **2** child nodes are full
- then splitting the **2** full nodes into **3** nodes each $\frac{2}{3}$ full
- as a result nodes are generally fuller and trees are more shallow which means **faster searches**
- data can be shared between siblings or neighboring nodes this is why the implementation is quite complex

B+ Trees

- it is the original B-tree like tree structure
- but the **leaf nodes are connected** with references (pointers) in a linked list manner
- the primary value of a **B+ tree** is in storing data for efficient retrieval in a block-oriented storage context like file systems
- almost every operating system (*MacOS*, *Linux* or *Windows*) rely heavily on **B+ trees**
- **NTFS** file system is first constructed at **Microsoft**
- **APFS** (Apple File System) is the file system of **MacOS**