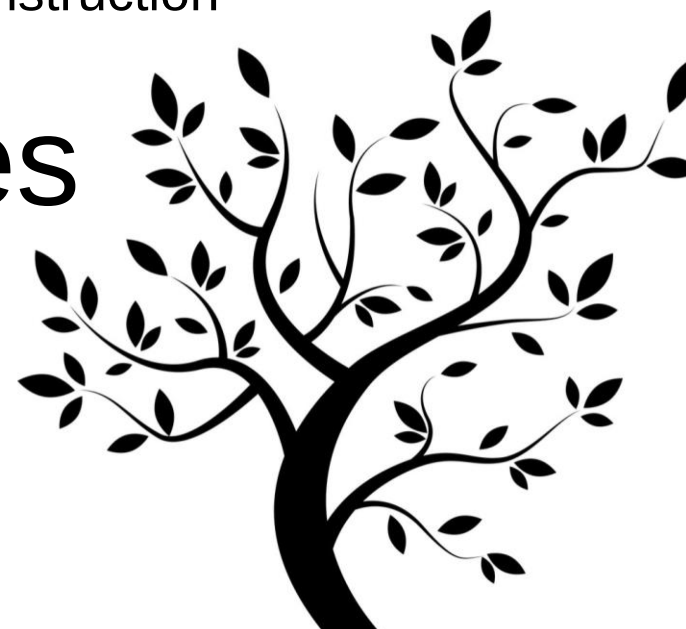
Week 5 Announcements

- Reminder from last week (W4):
  - Survey 1 (weeks 4 and 5).
  - Pointing to W3 reminders:
    - Video Assignment #1 → no late submission (hard deadline).
    - Group Project.
  - UML Exercise.
  - Control Flow Graph Exercise.
- Industry talks: 18 & 25 September.
- Android content is now available (in prep for W6 labs).
- COMP2100 are you okay? (zero e-mails) 😧 😕
- How are you doing? Is everything okay?
  - If you consider yourself at risk, please reach out!

Australian National University

COMP2100/6442
Software Design Methodologies / Software Construction

Tree Data Structures



Continuation…

Bernardo Pereira Nunes and
Sergio J. Rodríguez Méndez

Australian National University

Outline

- ~~Data Structures~~
- ~~Tree Data Structures~~
  - B-Tree
- Exercises

Australian National University

B-Tree

B-Tree

Bayer and McCreight never explained what, if anything, the *B* stands for: *Boeing*, *balanced*, *between*, *broad*, *bushy*, and *Bayer* have been suggested.[4][5][6] McCreight has said that "the more you think about what the B in B-trees means, the better you understand B-trees."[5]

The **B** stands for Bayer*

> Generalization of Binary Search Trees

- Not binary trees
- Overcome BST Limitation: each node is read individually (and access to secondary memory is slow)

> Designed for searching data stored on block-oriented devices

- Map tree nodes into blocks

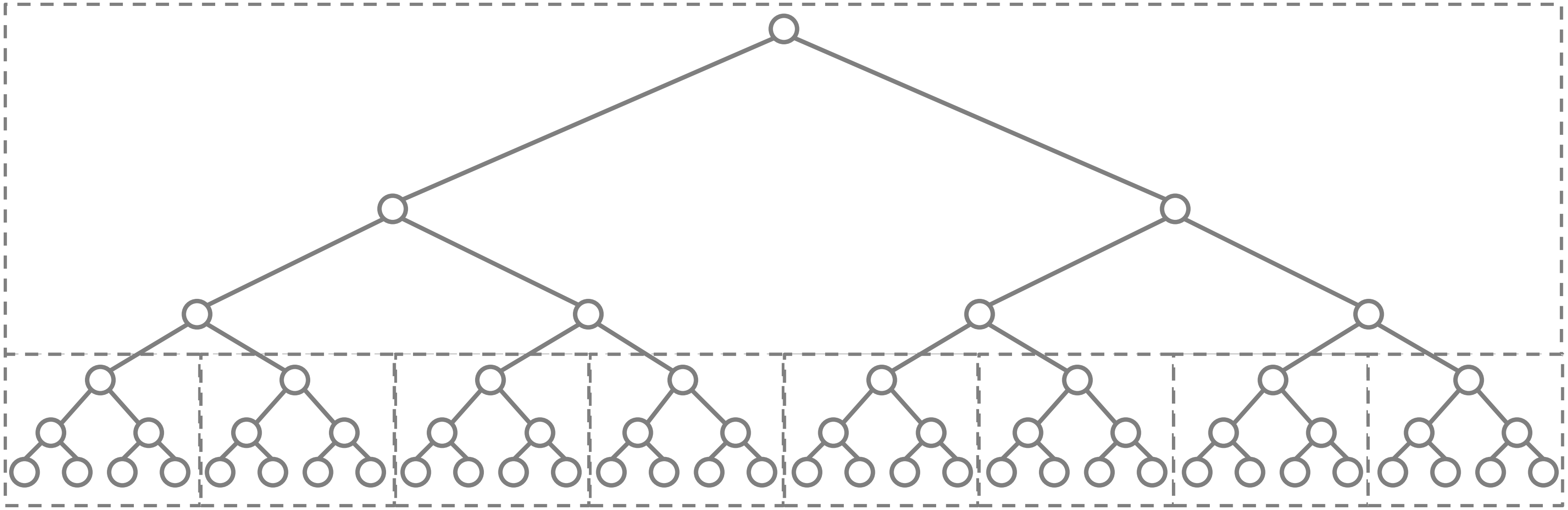
Finally, a balanced tree!

https://en.wikipedia.org/wiki/B-tree

B-Tree

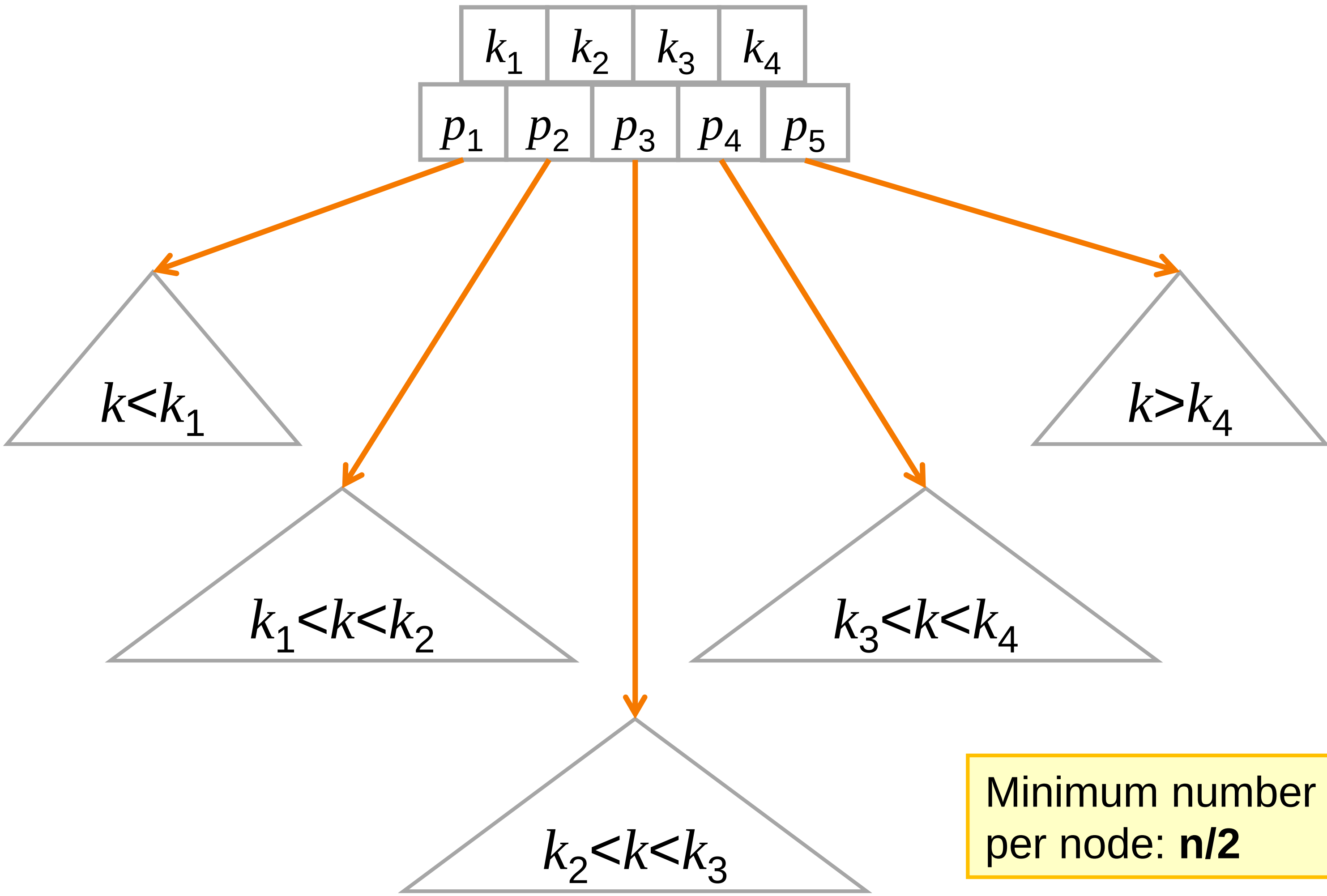
> Each access to secondary memory brings a group of elements

> Subtrees are divided into blocks (**pages**)



B-Tree: General Idea

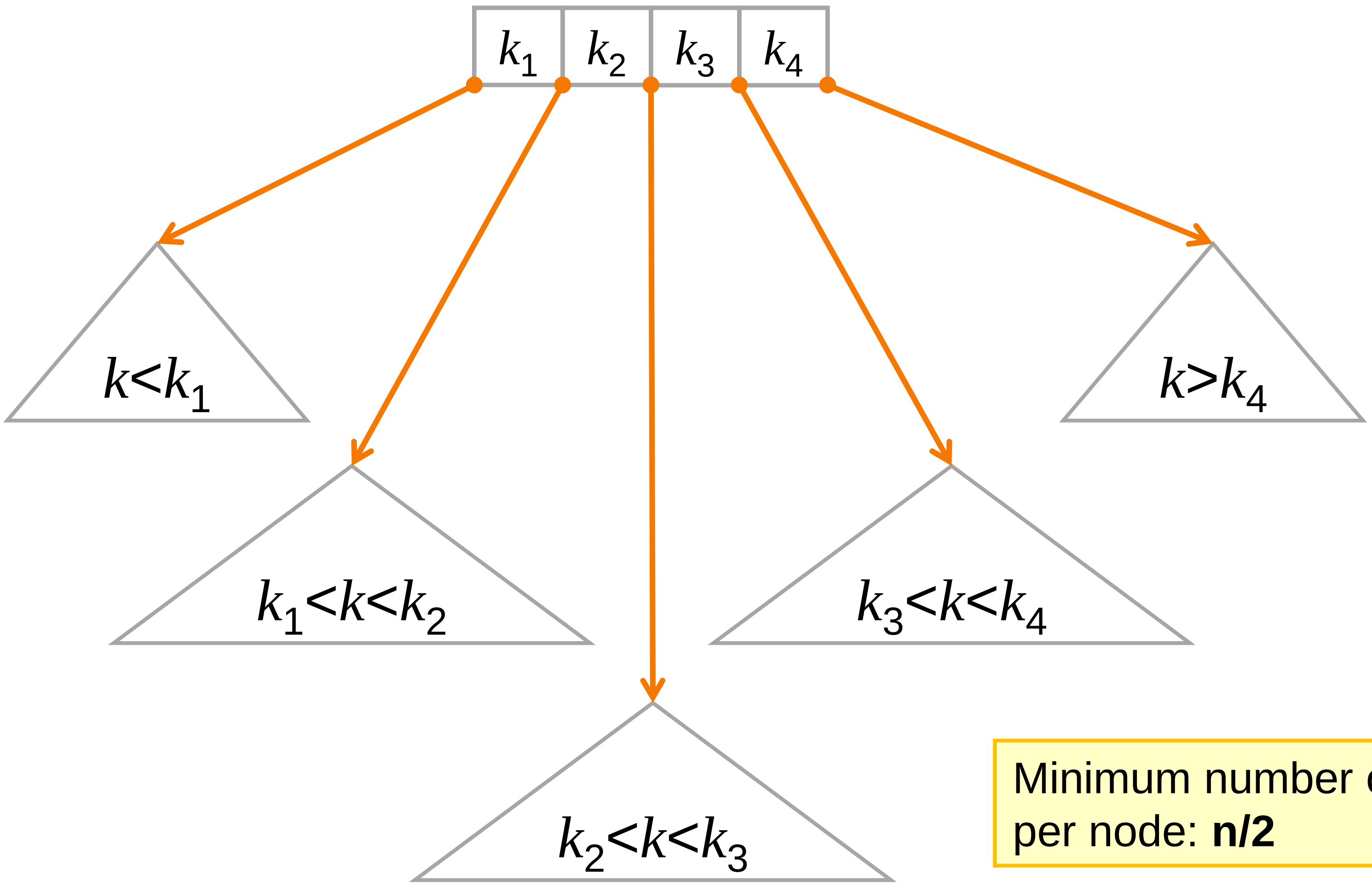
N-ary tree with search keys on nodes



Minimum number of keys per node: **n/2**

B-Tree: General Idea

N-ary tree with search keys on nodes

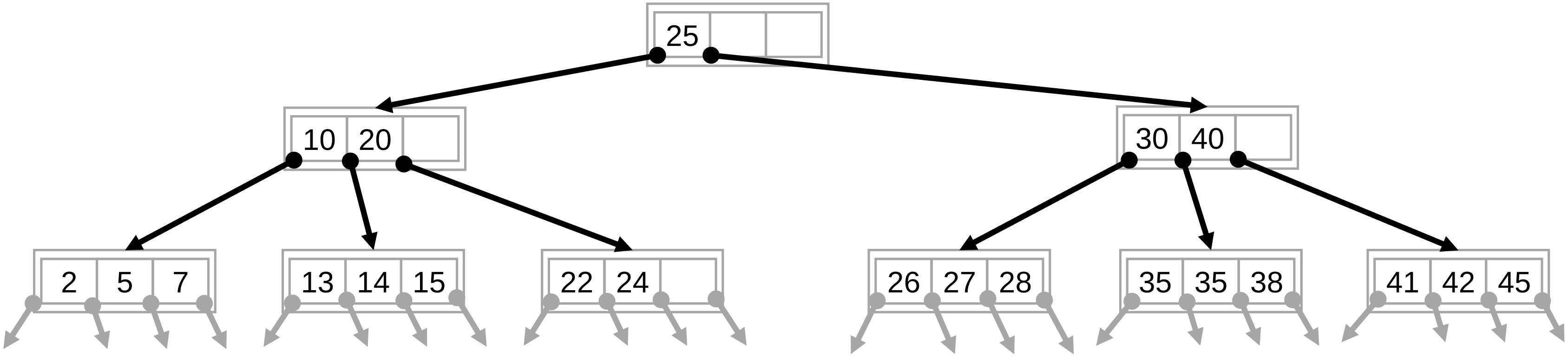


Minimum number of keys per node: **n/2**

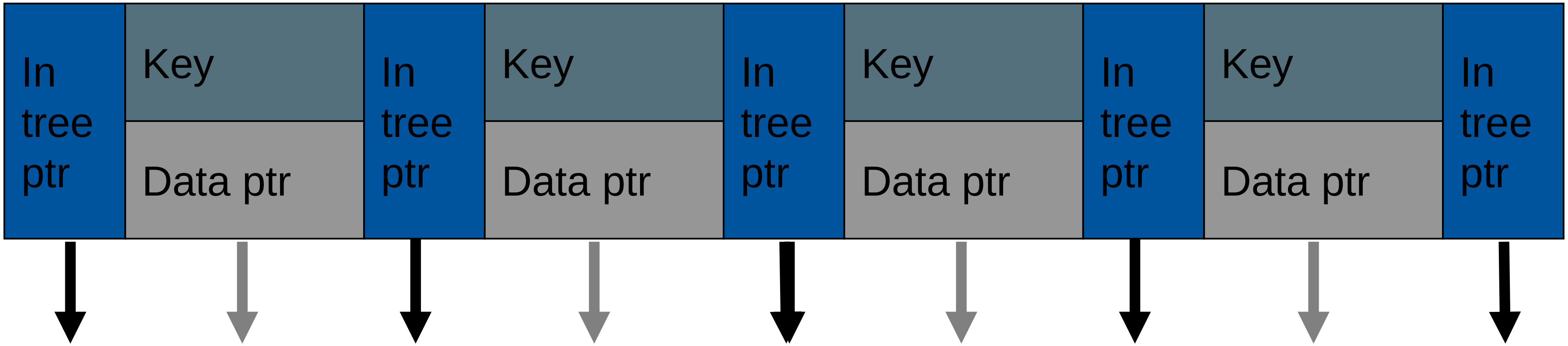
B-Tree: Definition (Knuth, 1997)

A B-tree of order *m* satisfies the following properties:

1. Every node (page) has at most *m* children
2. Each node (except the root and leaves) has at least $\lceil m/2 \rceil$ children
3. The root has at least 2 children (unless it is a leaf)
4. A non-leaf node with *k* children contains *k*-1 keys
5. All leaves appear in the same level



In reality


In tree ptr
Key
Data ptr
In tree ptr
Key
Data ptr
In tree ptr
Key
Data ptr
In tree ptr
Key
Data ptr
In tree ptr

B-Tree: Definition (Knuth, 1997)

> Maximum number of keys of a B-Tree of order *m* and height *h*:

number of keys

number of children

$$N = (m-1)*(1 + m + m^2 + m^3 + \ldots + m^h) =$$

$$= (m-1)\,\frac{1-m^{h+1}}{1-m} = m^{h+1} - 1$$

> The sum of the first terms of a geometric series is given by:

$$a + ar + ar^2 + ar^3 + \ldots + ar^{n-1} = a\,\frac{1-r^n}{1-r}$$

B-Tree: Definition (Knuth, 1997)

> Minimum number of keys of a B-Tree of order *m* and height *h*

>> The root has at least 1 key and 2 children
>> All internal nodes have at least $\lceil m/2 \rceil$ children and $\lceil m/2 \rceil -1$ keys
>> All leaves have at least ($\lceil m/2 \rceil -1$) keys

>> Therefore, the minimum number of keys will be:

$$k = 2 * \lceil m/2 \rceil^h -1$$

B-Tree: Definition (Knuth, 1997)

root has 2 children (minimum)

> Minimum number of keys of a B-Tree of order *m* and height *h*

$$k = 1 + (\lceil m/2 \rceil - 1) * (2 + 2*\lceil m/2 \rceil + 2*\lceil m/2 \rceil^2 + \ldots + 2*\lceil m/2 \rceil^{(h-1)})$$

root has 1 key (minimum)

$$= 1 + (\lceil m/2 \rceil - 1) * 2 * (1 + \lceil m/2 \rceil + \lceil m/2 \rceil^2 + \ldots + \lceil m/2 \rceil^{(h-1)})$$

each children can have m/2 children (minimum)

$$= 1 + (\lceil m/2 \rceil - 1) * 2 * (\textstyle\sum_{i=0}^{h-1} \lceil m/2 \rceil^i)$$

$$= 1 + (\lceil m/2 \rceil - 1) * 2 * \frac{\lceil m/2 \rceil^h - 1}{\lceil m/2 \rceil - 1}$$

minimum number of keys

$$= 1 + 2 * (\lceil m/2 \rceil^h - 1)$$

$$= 2 * (\lceil m/2 \rceil^h) - 1$$

B-Tree: Search

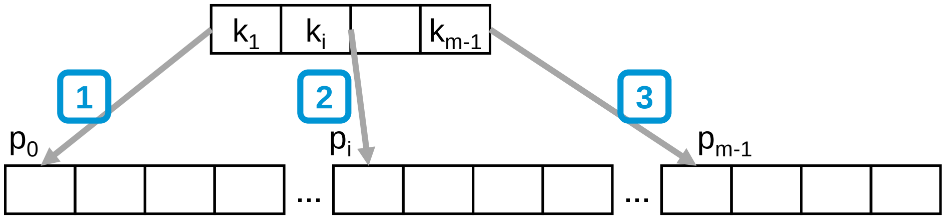
a. Search between keys on a page

k_1 ... k_{m-1} (if *m* is large: binary search)

b. if not found on the page:

1. $x < k_1$: search must continue on page p_0
2. $k_i < x < k_{i+1}$ for $1 \leq i < m-1$: search must continue on page p_i
3. $k_{m-1} < x$: search must continue on page p_{m-1}

if there are no pages below the current one, the key does not exist



B-Tree: Insertion

Let p_i be the page where *x* should be inserted

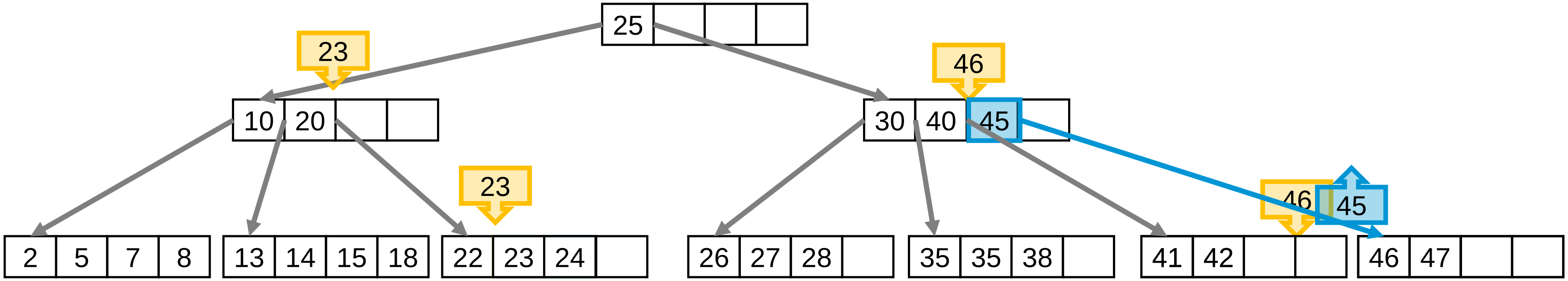
if p_i has less than m-1 elements then

insert in p_i, in the proper position

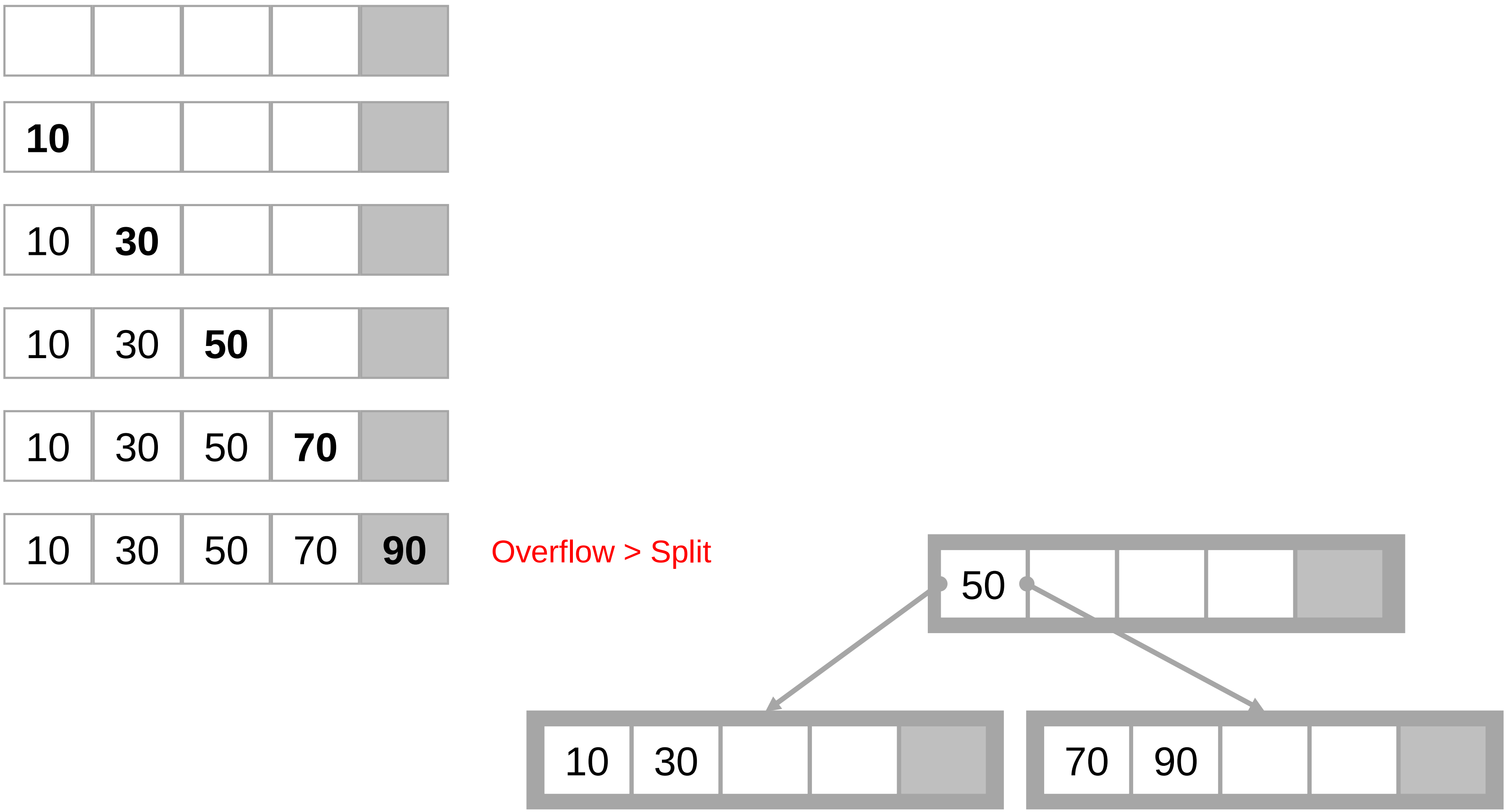
if page p_i is already full then

1. allocates a new page p_k
2. distributes the keys as follows:
    1. $\lceil m/2\text{-}1 \rceil$ smallest keys in p_i
    2. m-$\lceil m/2 \rceil$ biggest keys in p_k
    3. insert the median key (in $\lceil m/2 \rceil$) on the top page

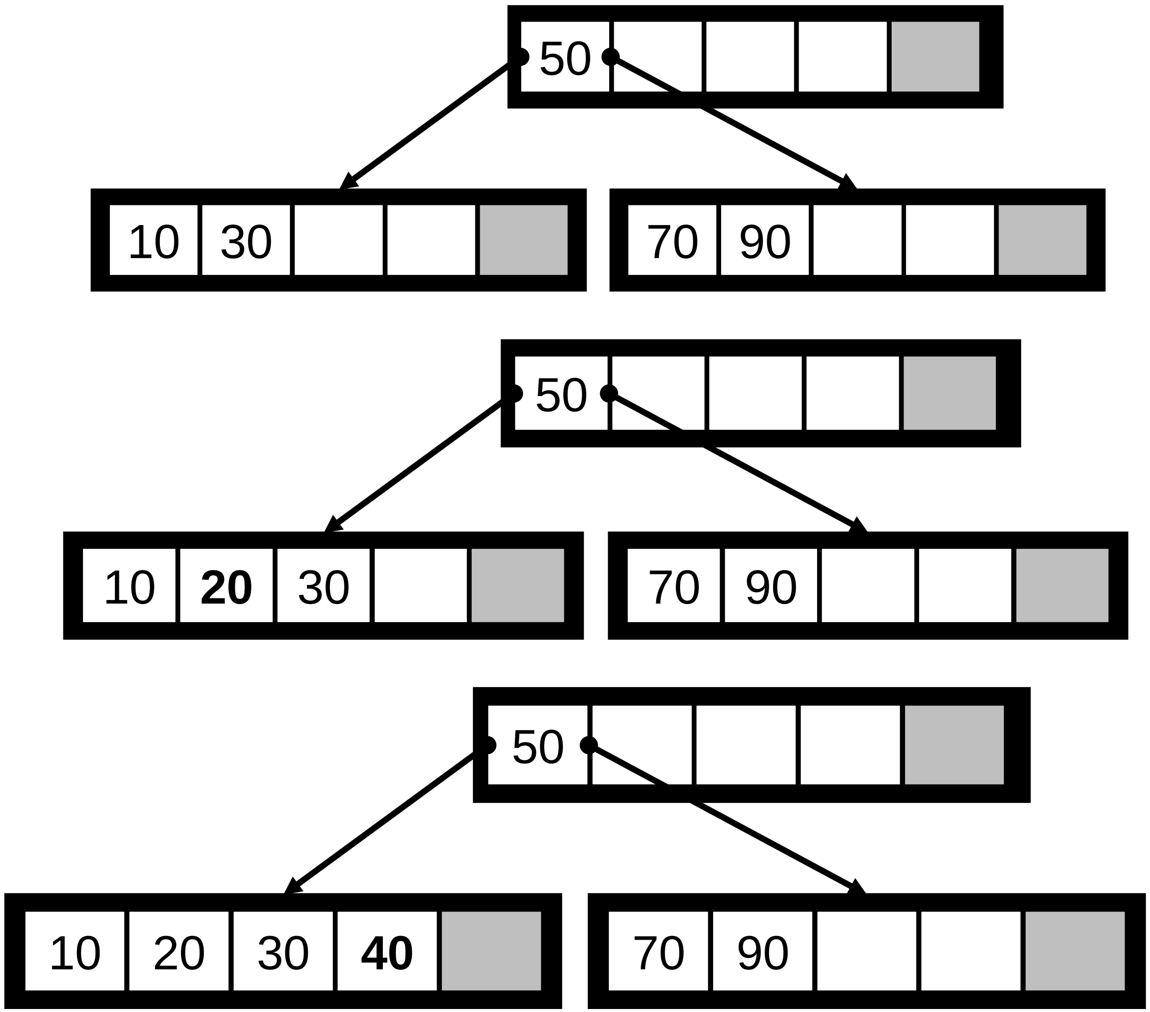
(if page p_i is root: create new root with median)



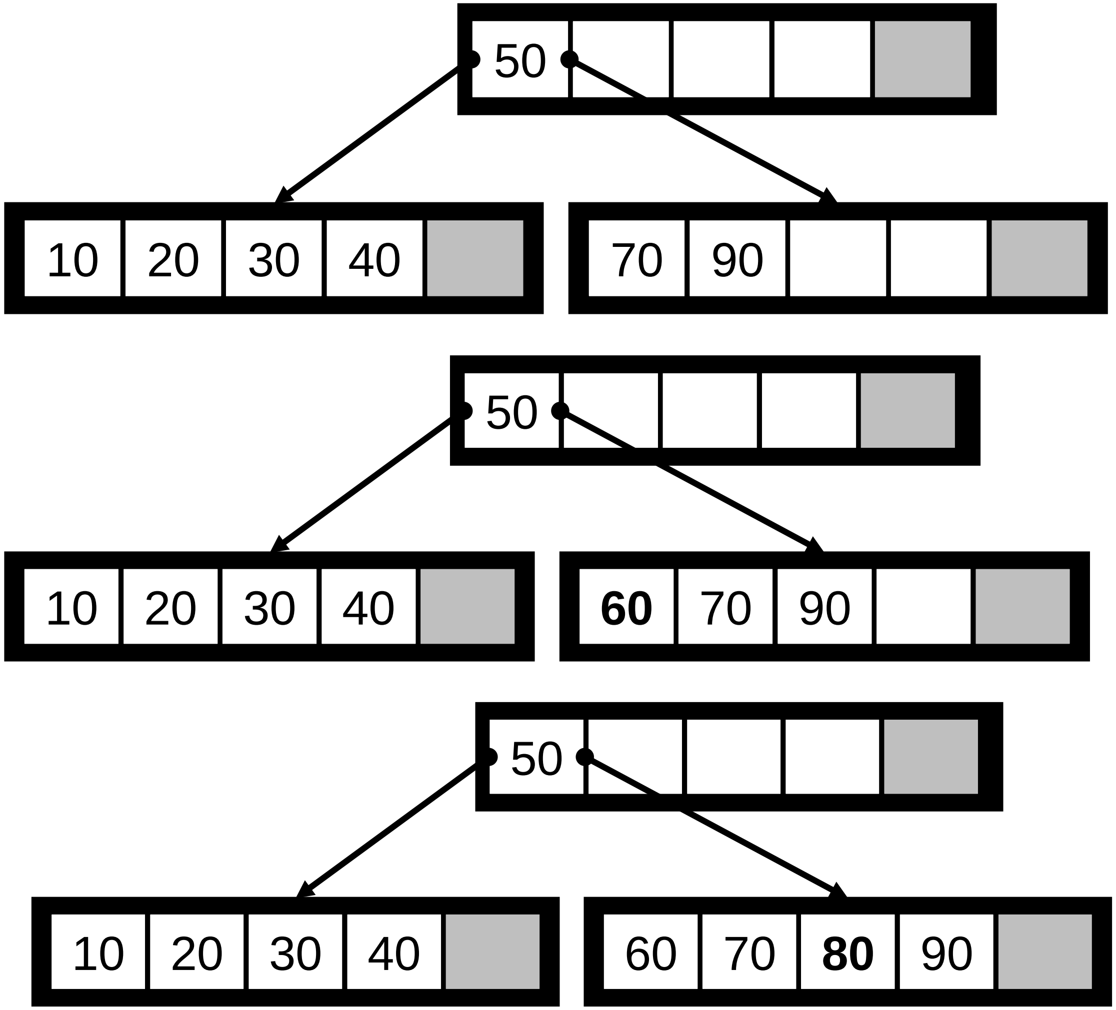
Insert 10, 30, 50, 70, 90 (order 5)



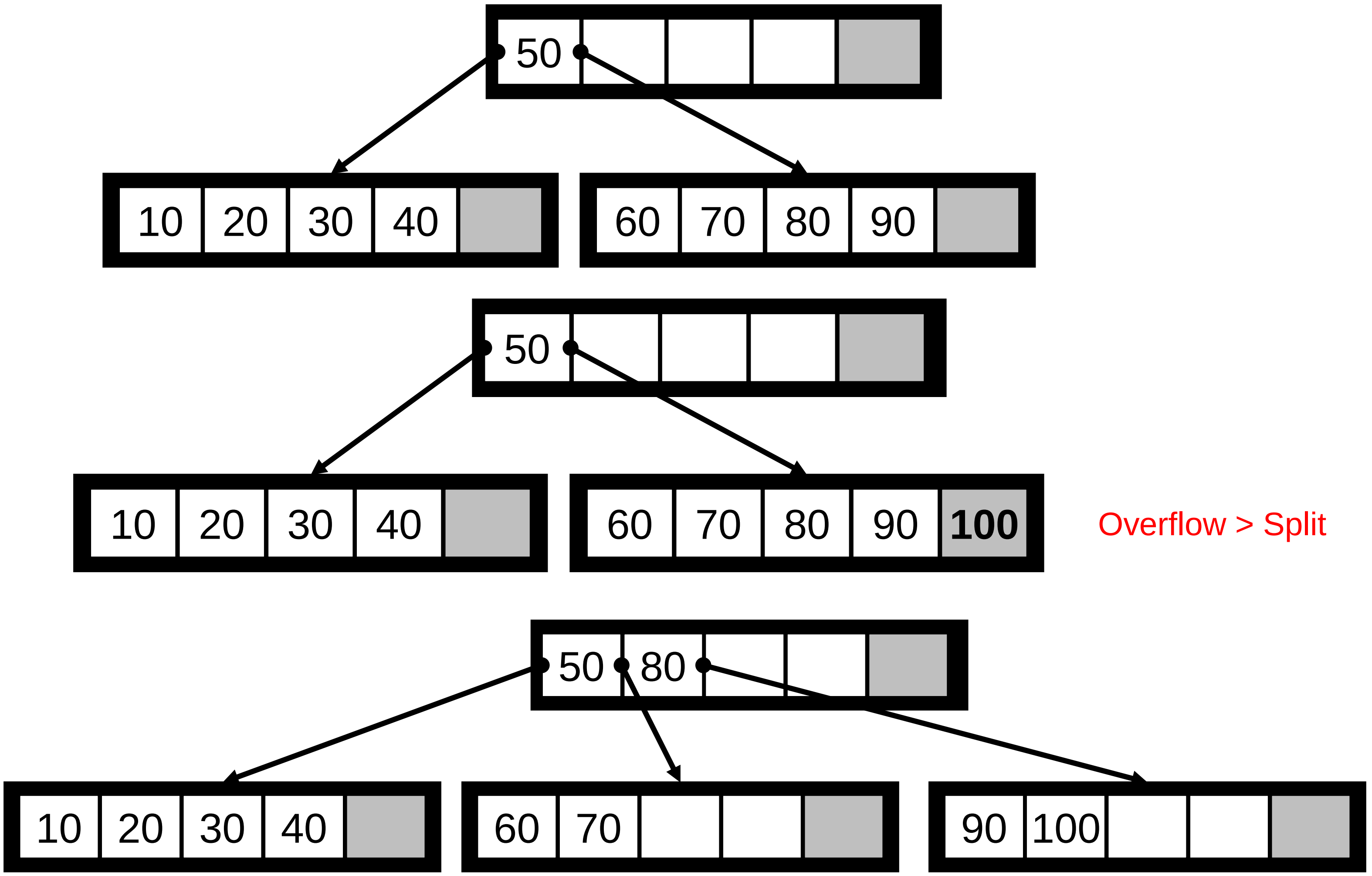
Insert 20, 40 (order 5)



Insert 60, 80 (order 5)



Insert 100 (order 5)



Exercise

Draw a B-Tree step-by-step inserting the keys in the following order: **20, 11, 15, 3, 5, 7, 12, 15, 16, 19, 25, 30, 33, 37, 22, 23, 26, 31, 42, 35, 6, 47**

Use different orders ***m*={3, 5}**

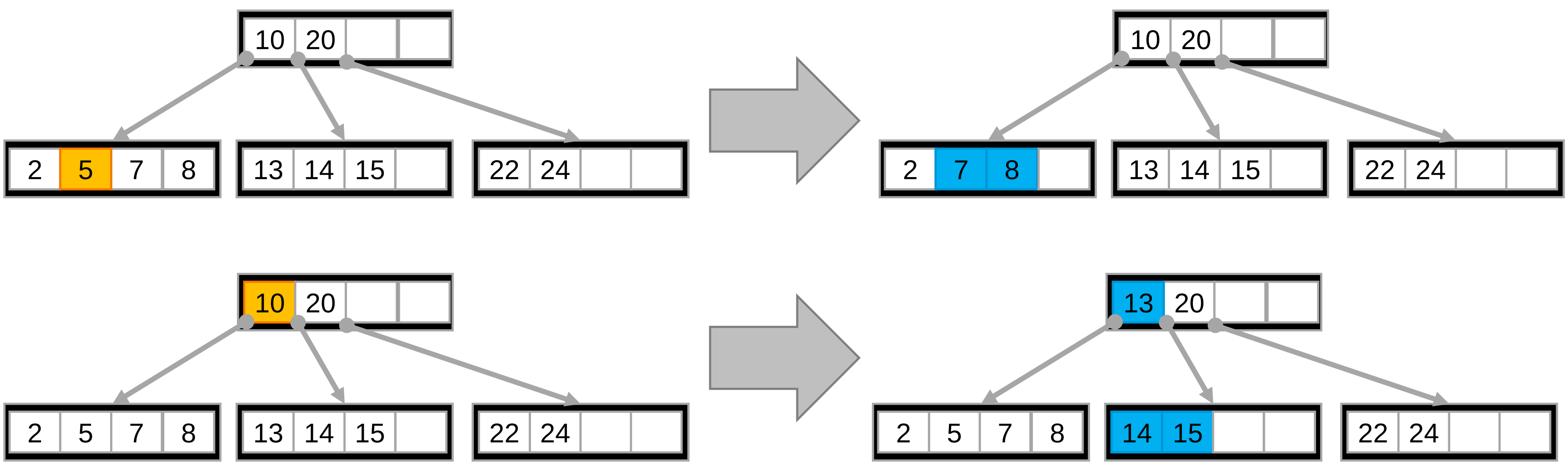
What is the height of the tree?

https://www.cs.usfca.edu/~galles/visualization/BTree.html

B-Tree: Remove

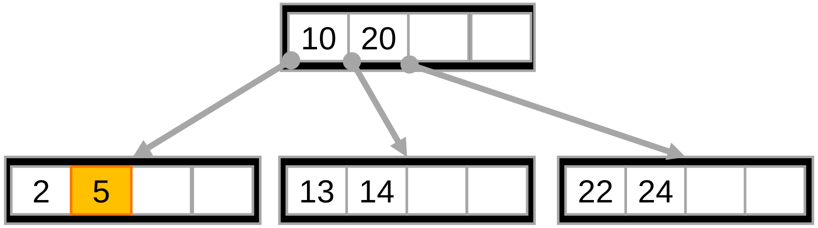
Removal must be performed on a leaf node

1. If the key to be removed is not in a leaf node then
   replace it with the largest key in its left subtree (predecessor)
   or the smallest key in its right subtree (successor)



B-Tree: Remove

2. if the key is on a leaf then
   it must be removed;
   if the leaf is **less than m/2 keys** then
   a **merge** or a **rebalancing** must be performed.



B-Tree: Remove

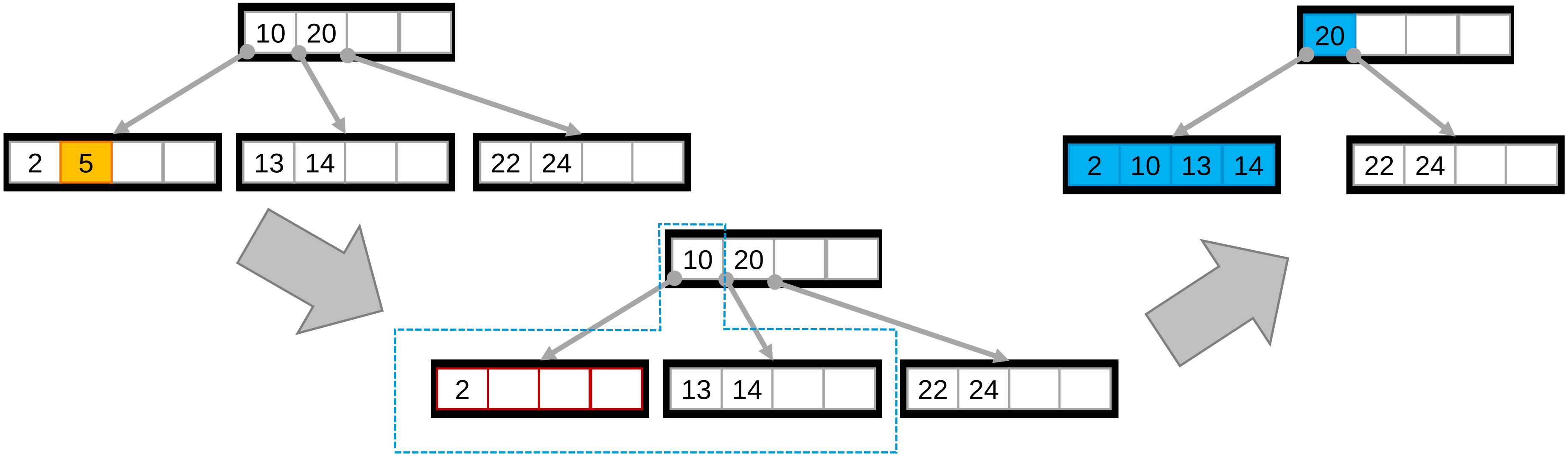
Merge:

> if, after removal, the page where the key was removed and its adjacent page have together less than m keys then

>> This page is merged with its adjacent one. The parent key that was between them goes to the page that was merged.

> if the resulting page has less than $m/2$ keys then

>> This procedure is repeated until it reaches the root.



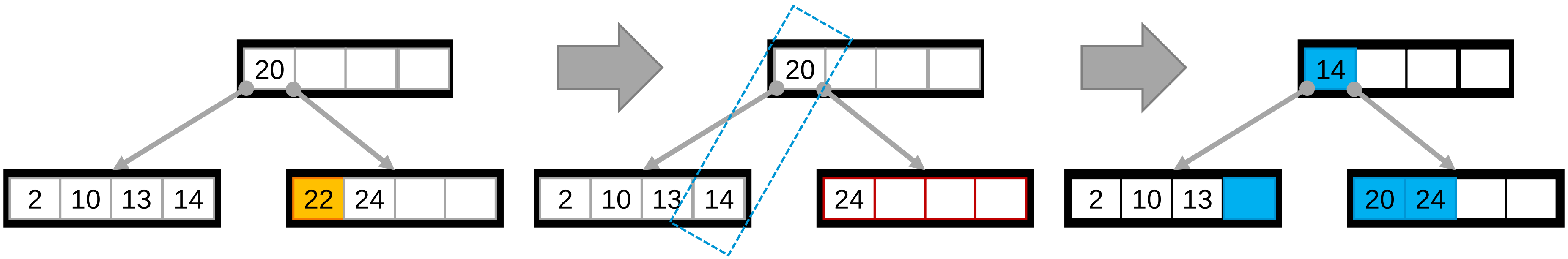
B-Tree: Remove

Rebalancing:

\> if, after removal, the page where the key was removed and its adjacent page together have *m* keys or more then

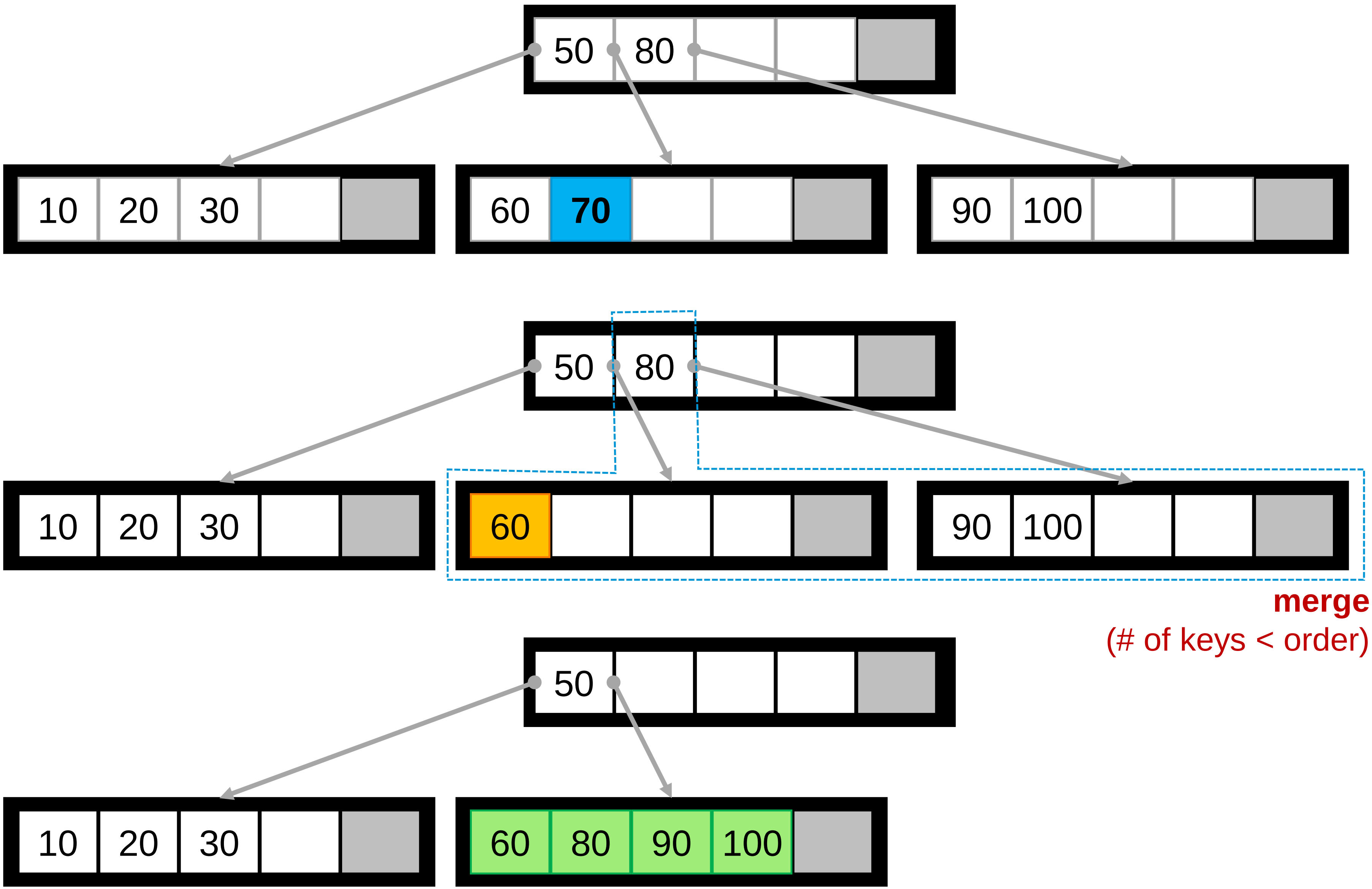
\>> Move the parent page key (the one "between" adjacent pages) to the page with less keys; then, move its adjacent page* key to the parent page.

\>There is no propagation as the number of parent keys does not change.



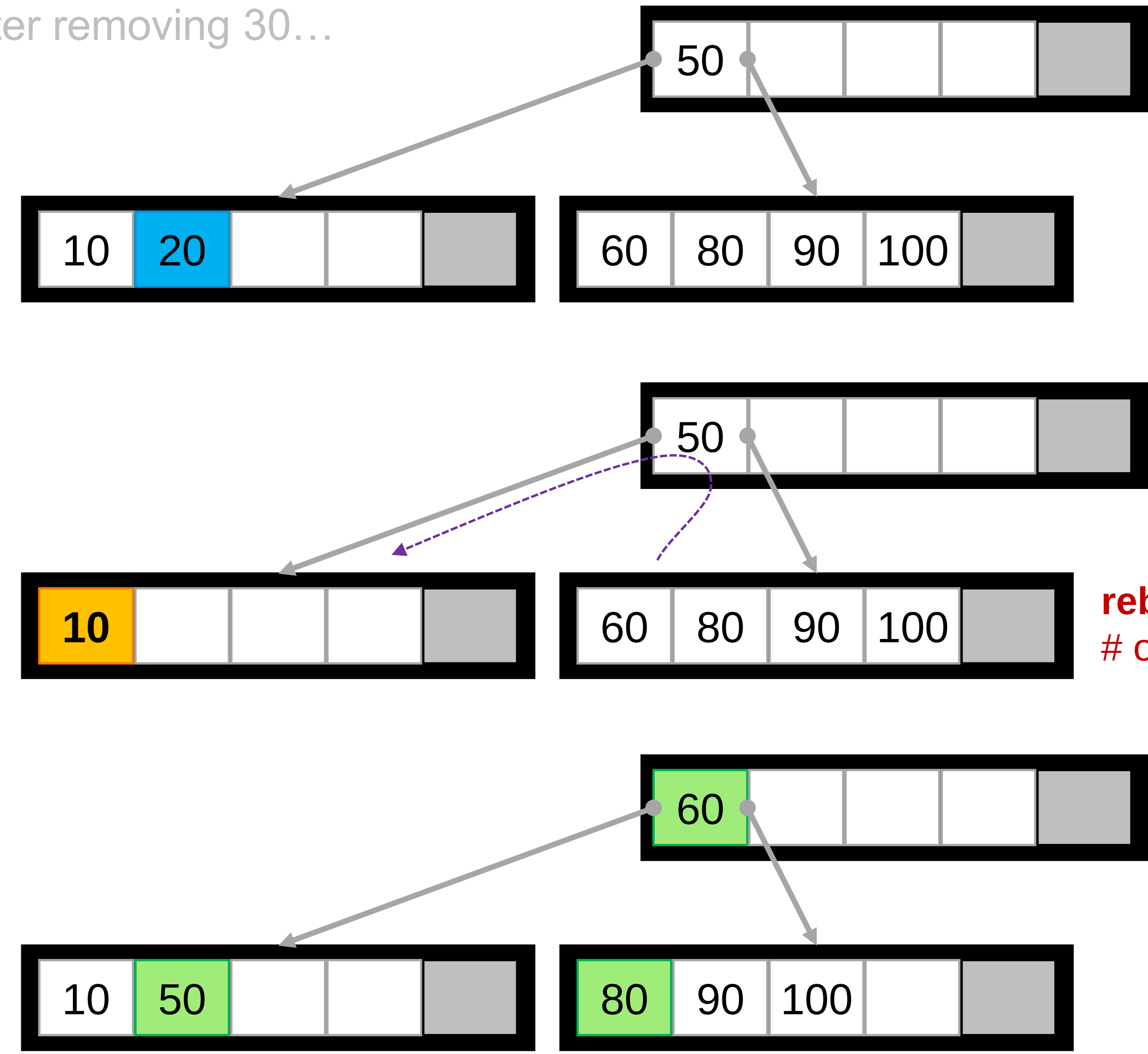
* if the adjacent page is to the left of the page with less keys, the moved key is the largest on that page (*borrow from left*). If the adjacent page is to the right, the key moved is the smallest of that page (*borrow from right*).

B-Tree: Remove 70 (order 5)



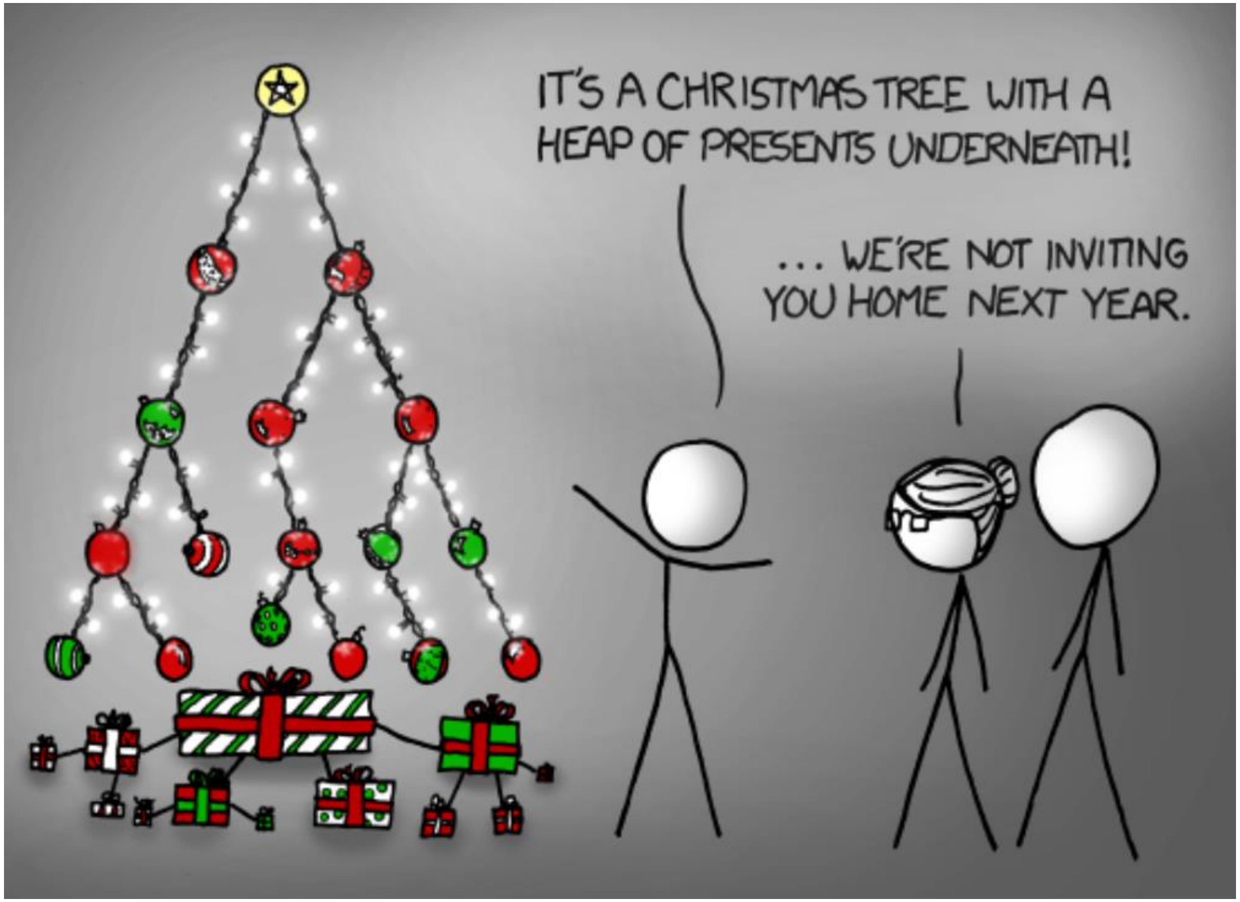
B-Tree: Remove 20 (order 5)

after removing 30…



rebalancing
# of keys on the right > order/2

Meme for today's lecture! Keep practicing!



It seems a good idea… no names on gifts anymore.

References

- Chapter 14 and 19 of Introduction to Algorithms (by Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest)
- Chapter 10 of Lee K.D., Hubbard S. (2015) Balanced Binary Search Trees. In: Data Structures and Algorithms with Python. Undergraduate Topics in Computer Science. Springer, Cham.
- Knuth, Donald (1998), Sorting and Searching, The Art of Computer Programming, Volume 3 (Second ed.), Addison-Wesley, ISBN 0-201-89685-0.