01 – Trees 1

Advanced Algorithms and Data Structures

UNIVERSITY OF ZAGREB
Faculty of Electrical Engineering and Computing

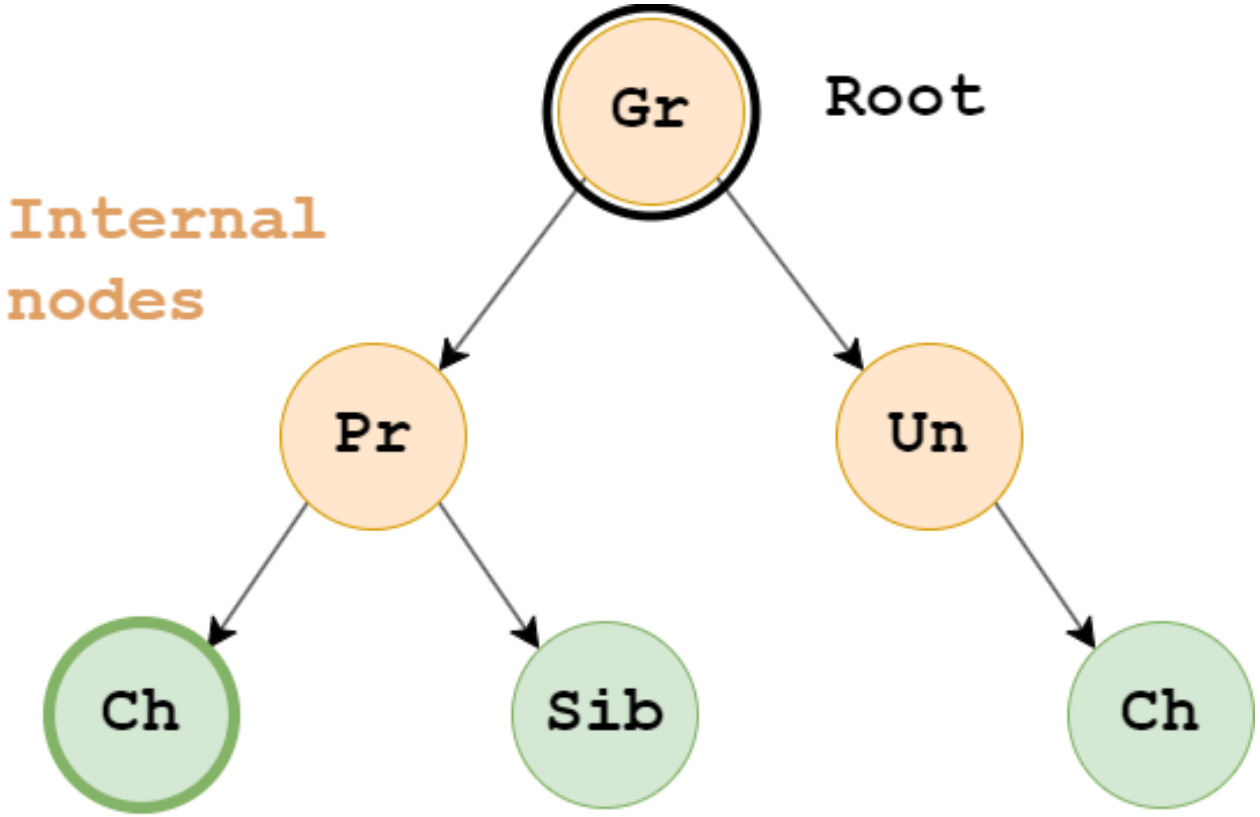
Creative Commons

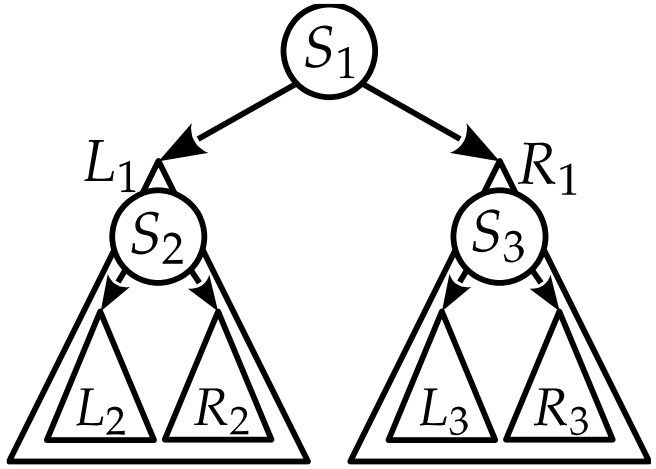
- You are free to:
  - share — multiply, distribute, and publicly communicate the work
  - adapt the work
- under the following conditions:
  - **Attribution:** You must acknowledge and indicate the authorship of the work in the manner specified by the author or license provider (but not in a way that suggests you or your use of the work have their direct support).
  - **Non-commercial:** You may not use this work for commercial purposes.
  - **Share alike:** If you modify, transform, or build upon this work, you may only distribute the modified work under the same or a similar license.

Binary trees (1)

- Trees are **acyclic graphs**
- **Binary trees** are a special case
  - Maximum of 2 children
- Recursive definition:

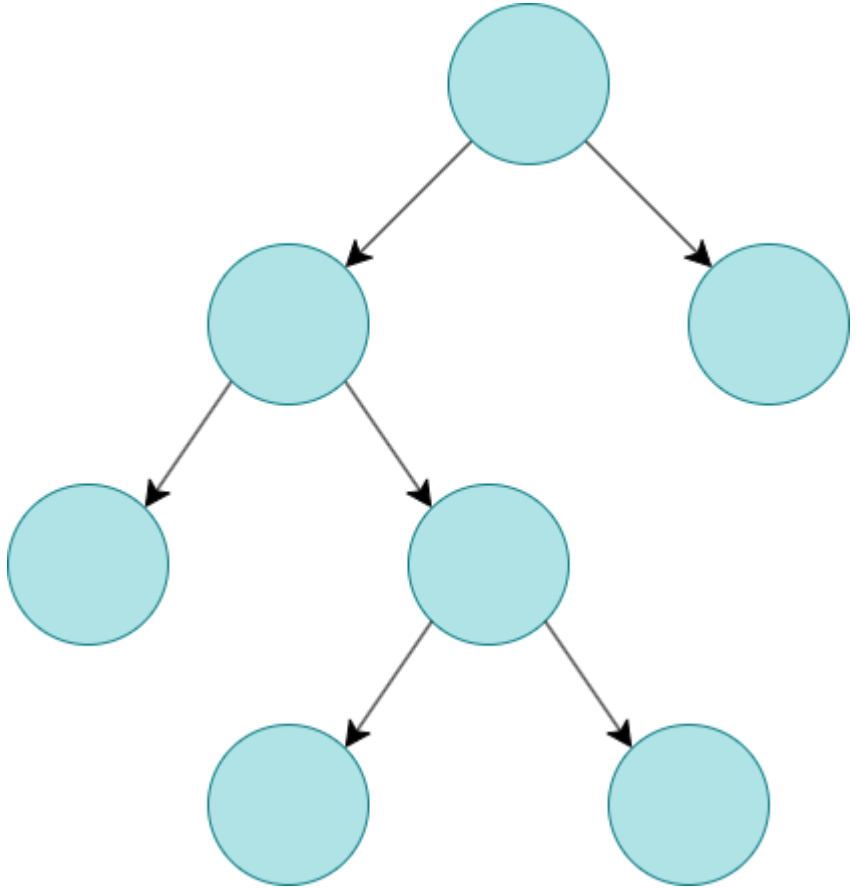
$B = (L, S, R)$





Binary trees (2)

- **Full** binary tree
  - Nodes either have 0 or 2 children
  - *Internal nodes have 2 children*



Binary trees (3)

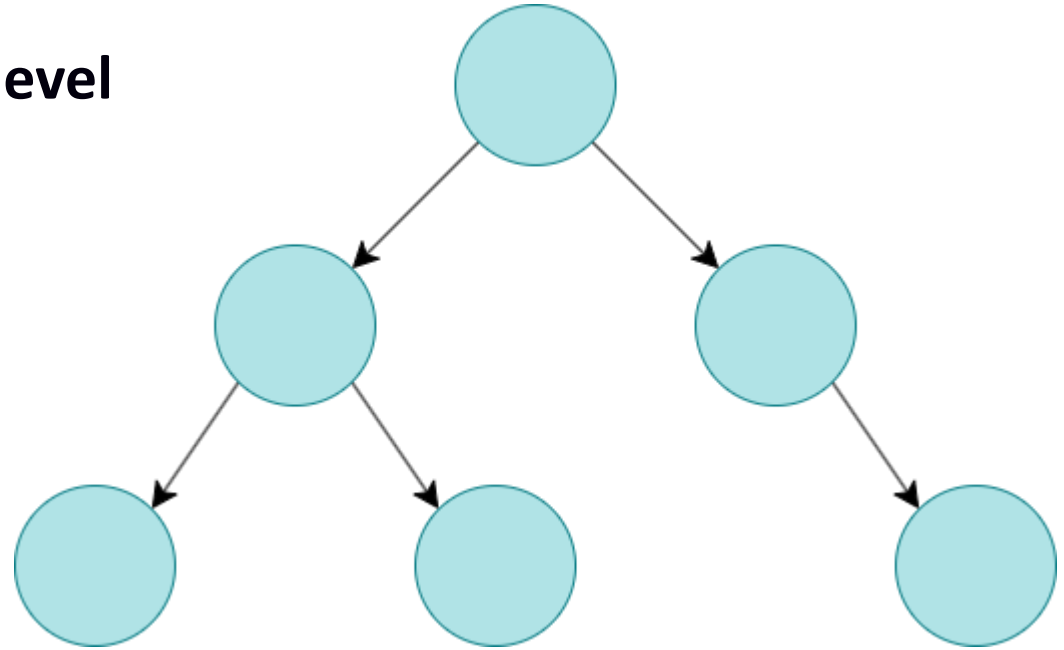
- **Complete** binary tree
  - All levels filled, **except for the lowest level**

$$N_{nodes} \leq 2^h - 1$$

$$N_{leafs\ (h=level)} \leq 2^{h-1}$$

$$N_{internal\ nodes} \leq 2^{h-1} - 1$$

$$H = \lceil log_2(n+1) \rceil$$



Binary trees (4)

- **Perfect** binary tree
  - All levels are full

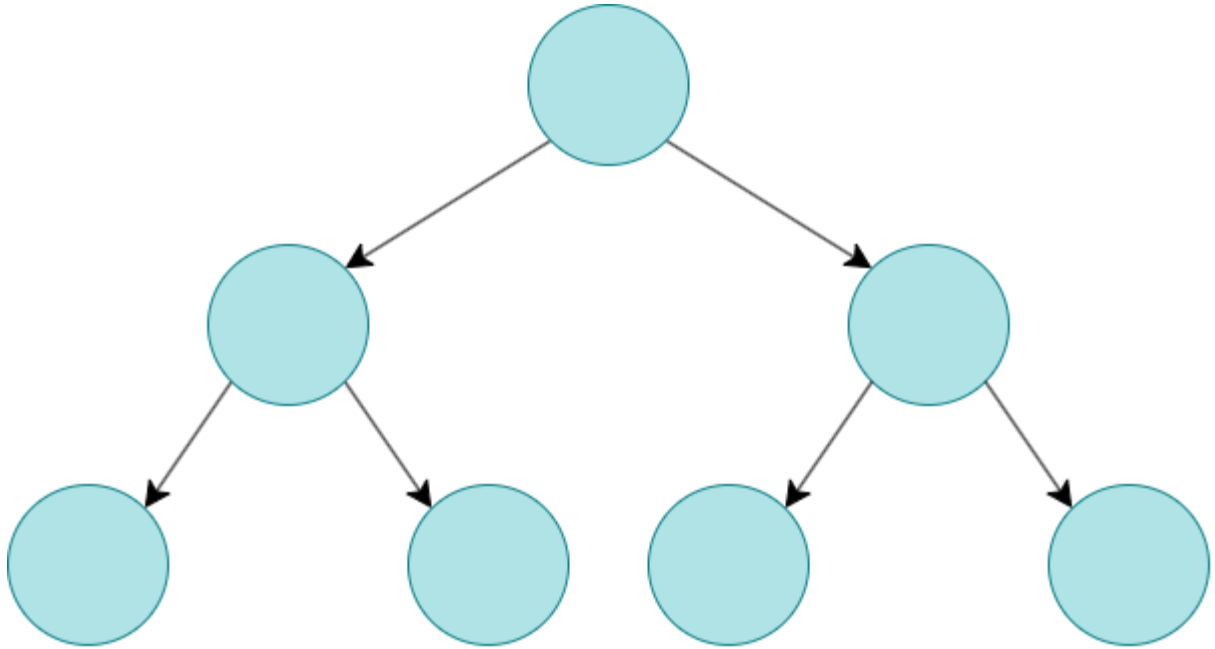
- Important formulae:

$$N_{nodes} = 2^h - 1$$

$$N_{leafs\ (h=level)} = 2^{h-1}$$

$$N_{internal\ nodes} = 2^{h-1} - 1$$

$$H = \log_2(n+1)$$



Binary tree (5)

- **Sorted** binary tree

$$B = (L, S, R)$$

  - No duplicates:

$$val\big(S(L)\big) < val(S) < val\big(S(R)\big)$$

  - With duplicates:

$$val\big(S(L)\big) \leq val(S) < val\big(S(R)\big)$$
$$val\big(S(L)\big) < val(S) \leq val\big(S(R)\big)$$

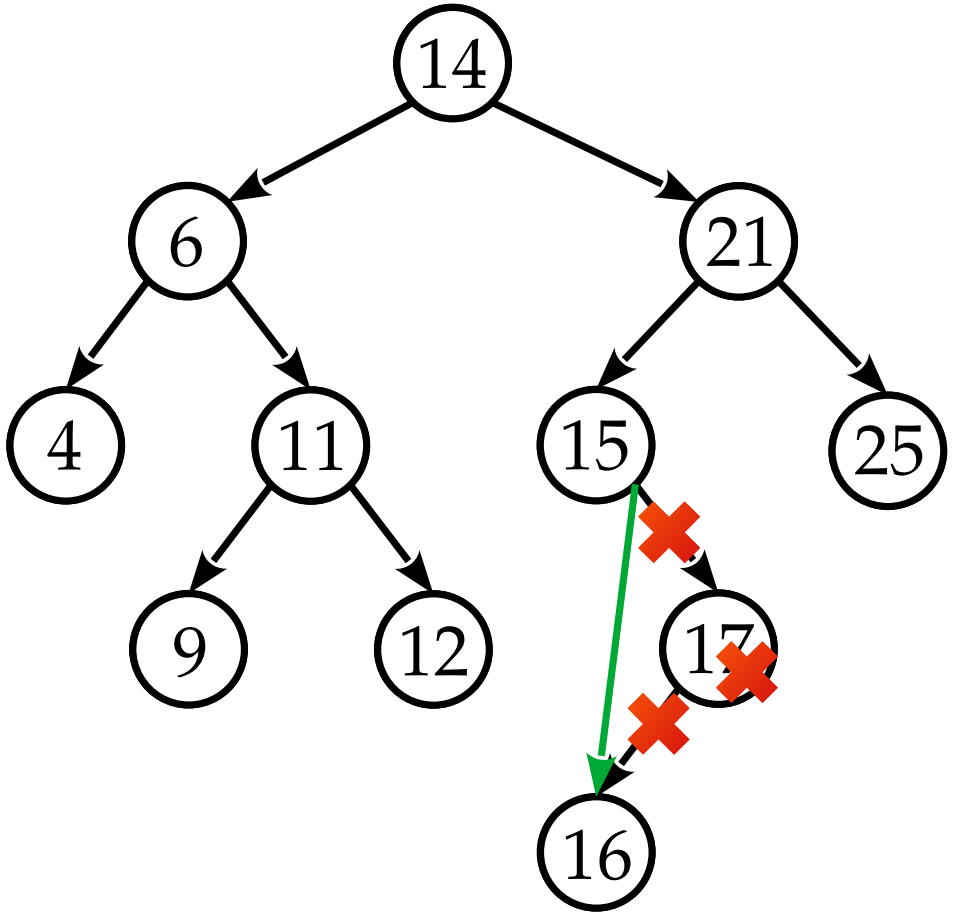
  - Lesser valued nodes go LEFT
  - Greater valued nodes go RIGHT

Operations - adding a node (SBT)

- Search from root to leaves
  - Left if lesser
  - Right if greater
- At leaf, place node:
  - As left child if lesser
  - As right child if greater

Operations - deleting a node (SBT)

- Search from root to node
  - …
- **Sc. 1** The node **is a leaf** – REMOVE
- **Sc. 2** The node **has 1 child**
  - Remove node
  - Recombine Gr -> Ch
  - Side is kept according to Gr „pointer”
- 2 children??



Operations – Deleting by copying (1)

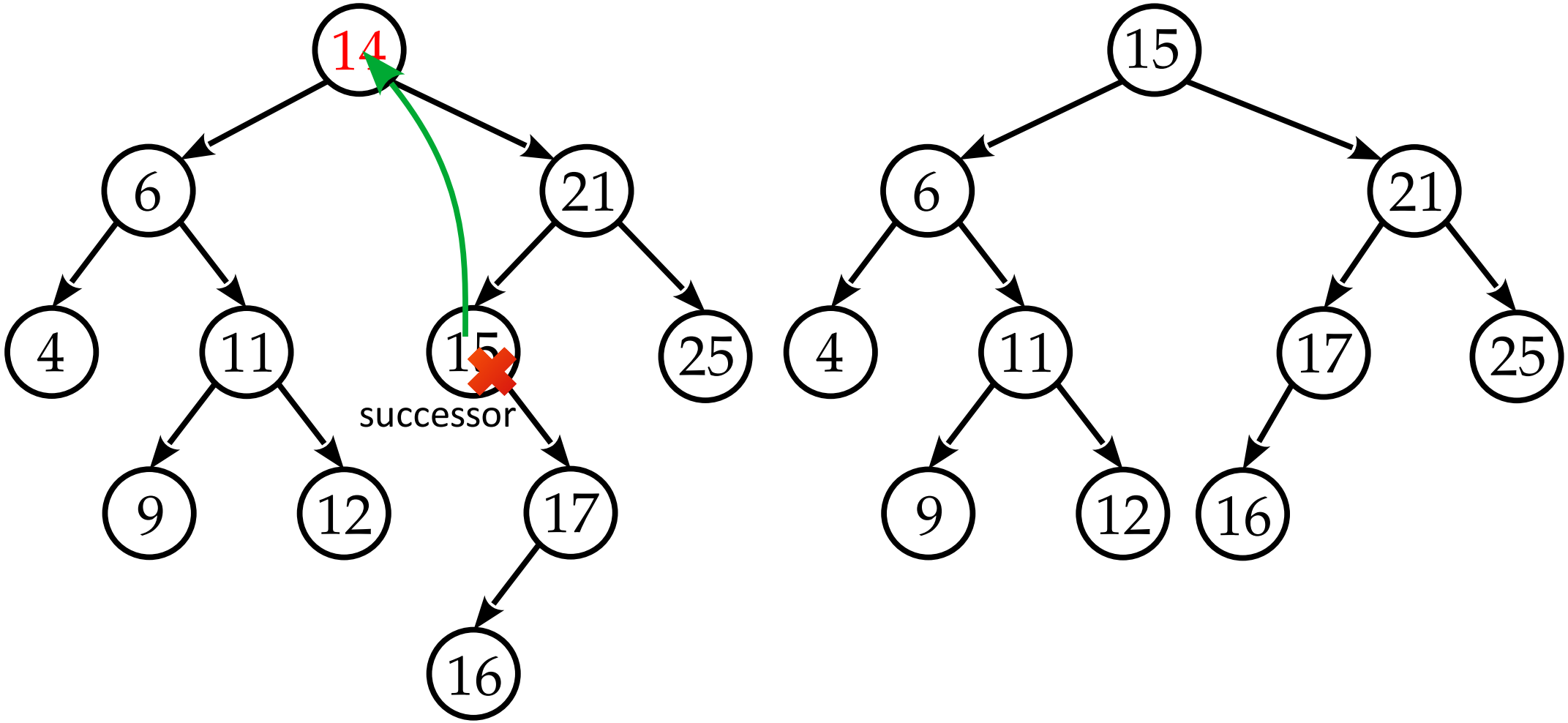
- **Strategy 1: By copying**
  - Structural changes are minimal
  - Comes down to Sc. 1 and 2
- Find the rightmost node in the left subtree – predecessor

  or …
- Find the leftmost node in the right subtree – successor

- The …essors have 0 or 1 child

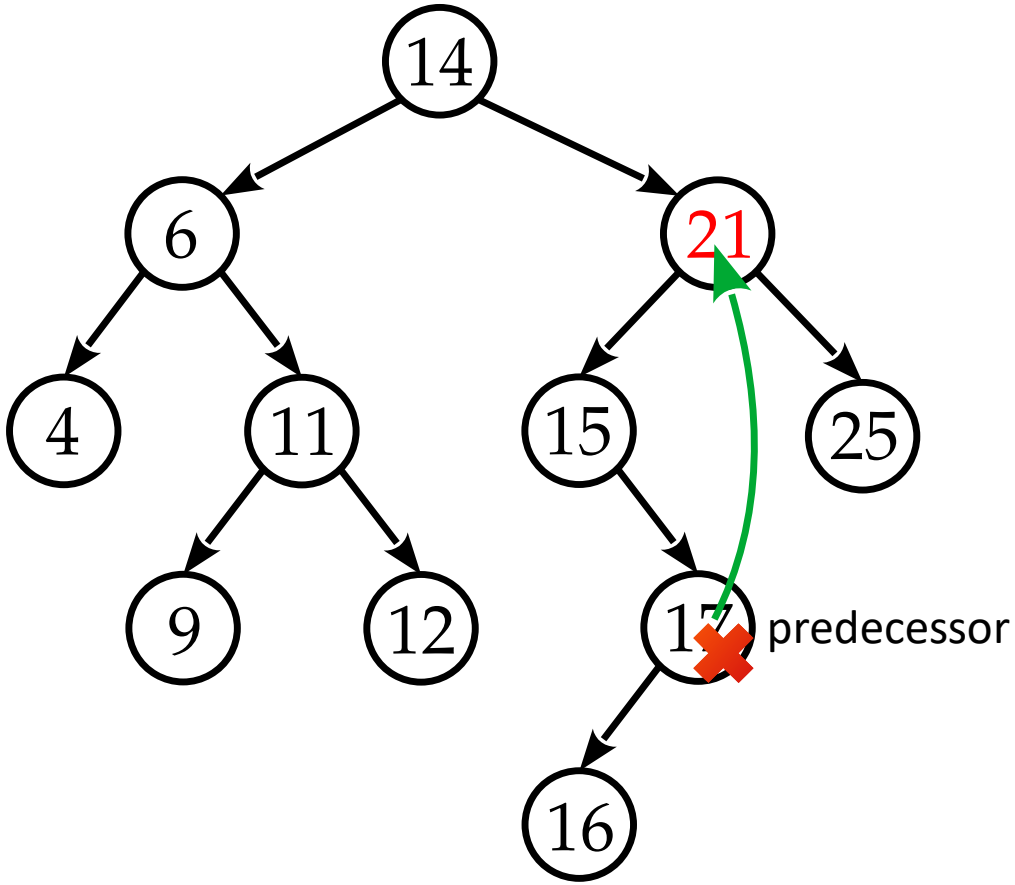
Operations – Deleting by copying (2)

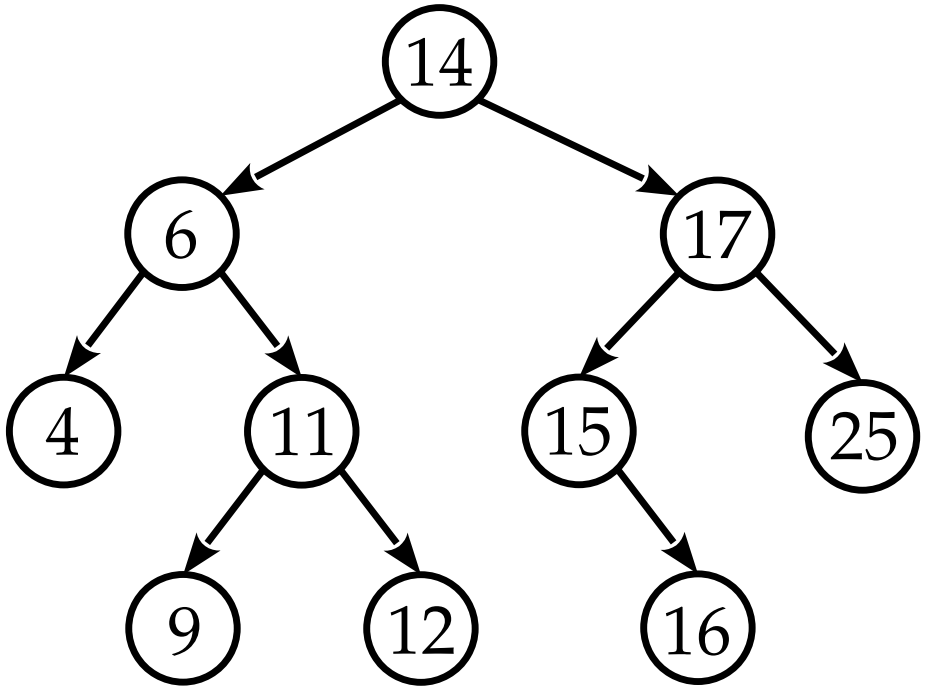
- Delete 14



Operations – Deleting by copying (3)

- Delete 21

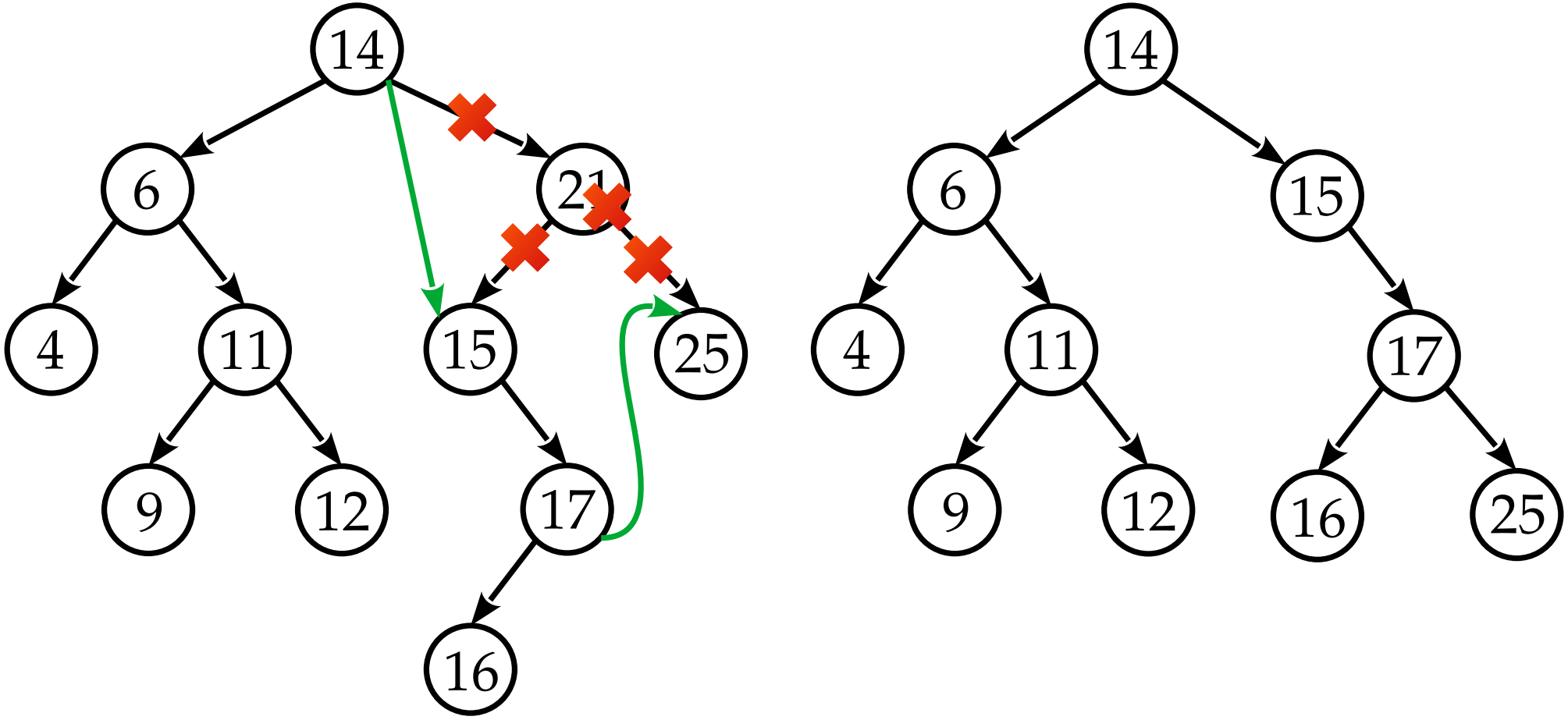




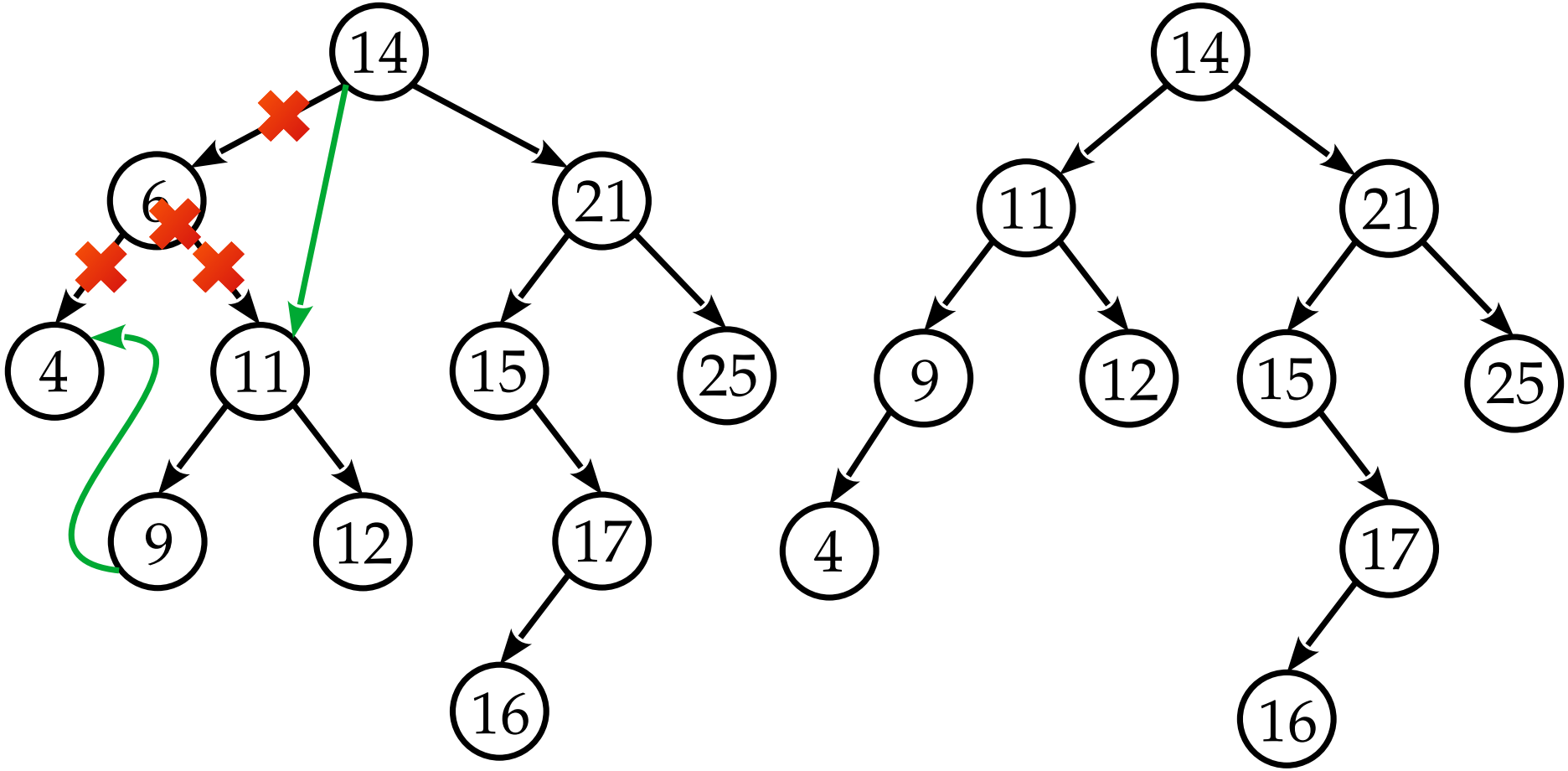
Operations - Deleting by merging (1)

- **Strategy 2: By merging**
  - Structural changes are considerable
  - The idea is to merge the subtrees of a node
- Find the *target* and *Pr*
  - Determine the side of the *target* in relation to the *Pr*
  - If left subtree – the merge subtree is *target*->right
    - target-> left goes to the leftmost of the merge subtree
  - If right subtree – the merge subtree is *target*->left
    - *target* -> right goes to the rightmost of the merge subtree

Operations - Deleting by merging (2)

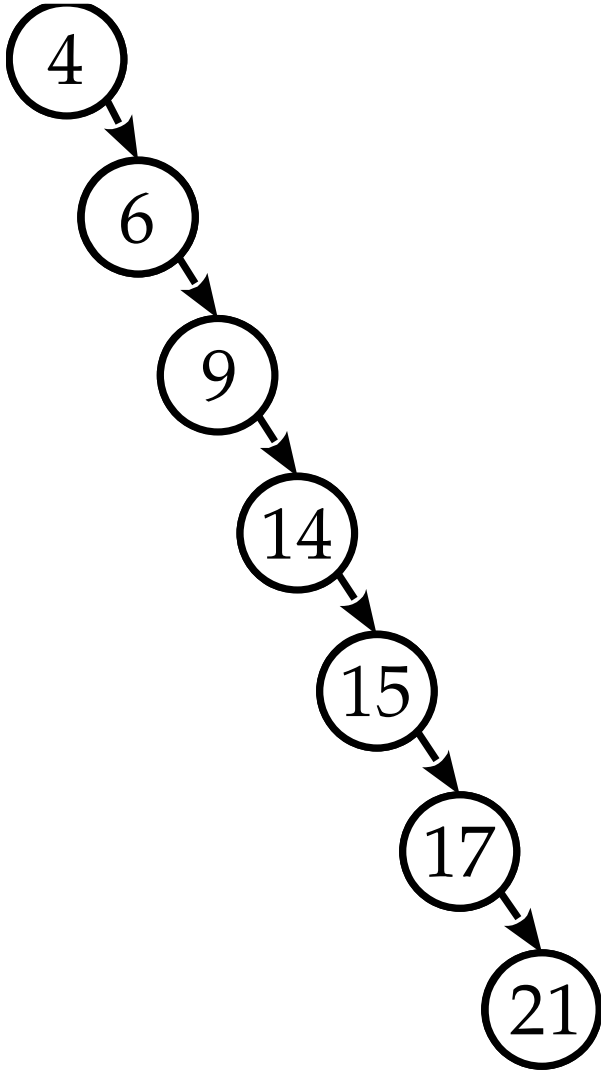


Operations - Deleting by merging (3)



Outcome of operations?

- Degenerate tree (backbone)
  - Left or right skewed
  - Spine
  - Linked list
- Search complexity (time)?
- How to fix this?



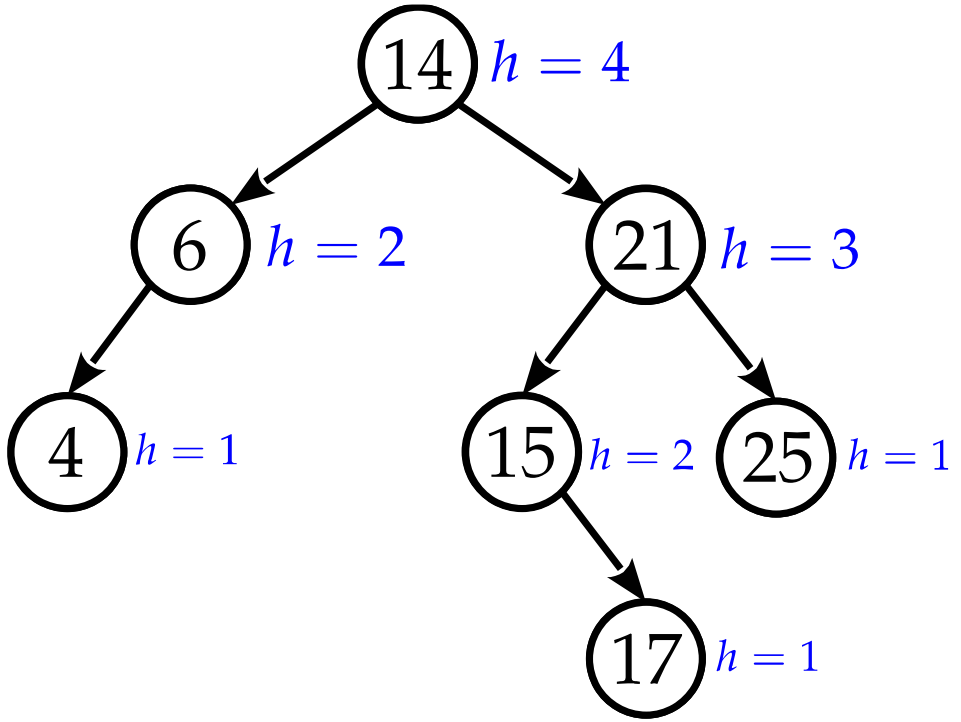
Balanced tree

- For a binary tree $B = (L, S, R)$ a balanced tree has the property

$$\forall S \quad |h(L) - h(R)| \leq 1$$

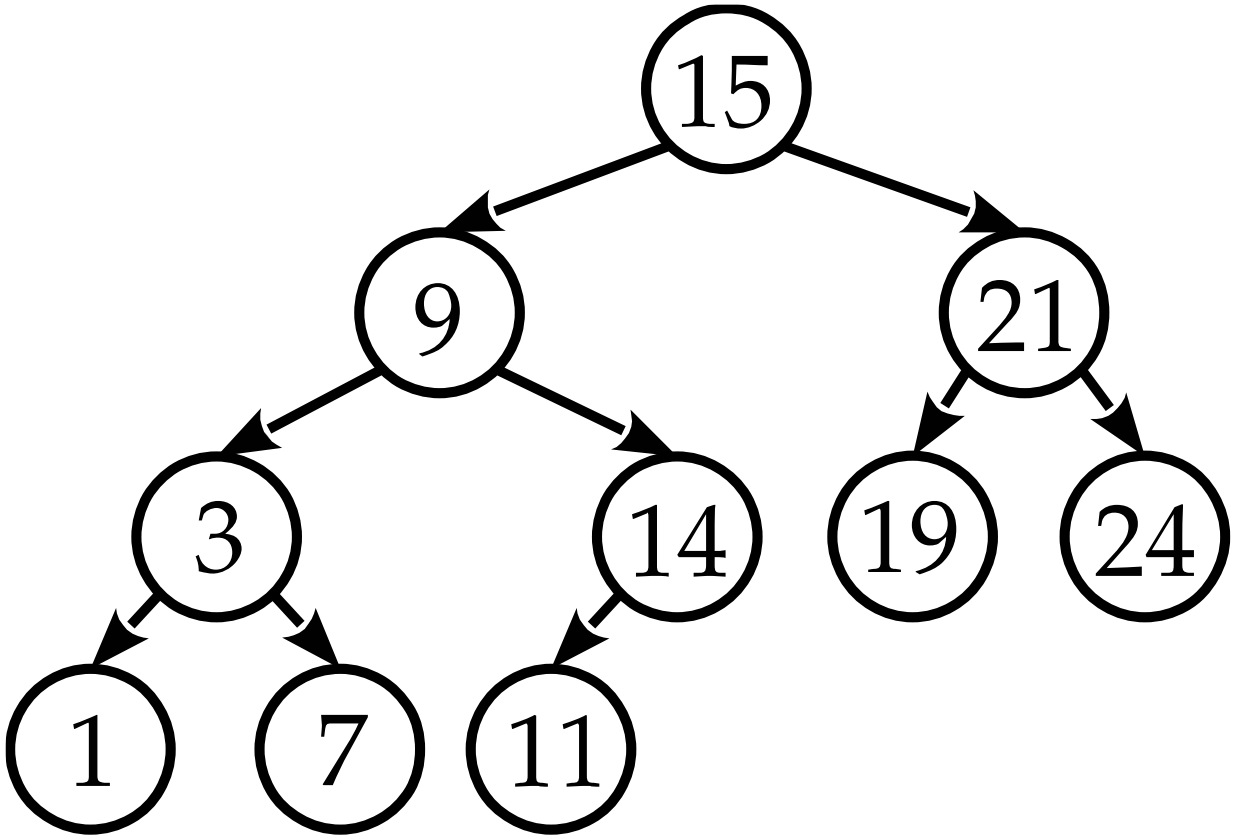
- **ALL** left and right subtrees have to be approximately the same height

- **Perfectly balanced tree**



Balanced tree?

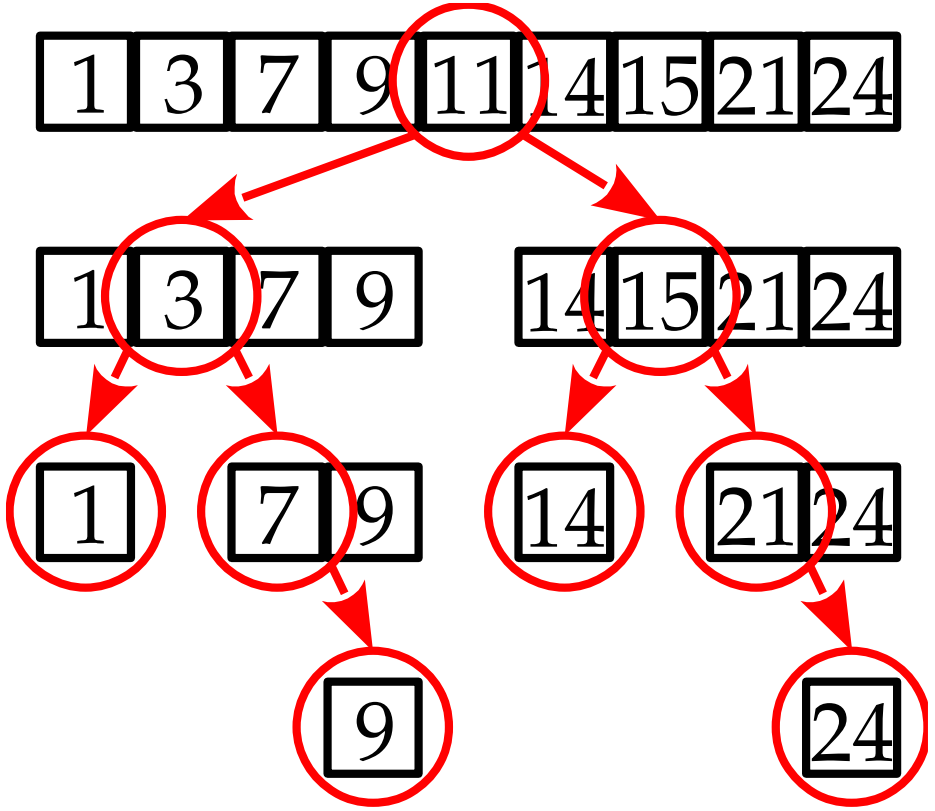
- Minimum deletions to disbalance?



Creating a balanced tree (sorted array)

- Get an array of values
- Sort the array
- Split the array
  - Positional centers become roots

$V_{sorted} = [1, 3, 7, 9, 11, 14, 15, 21, 24]$

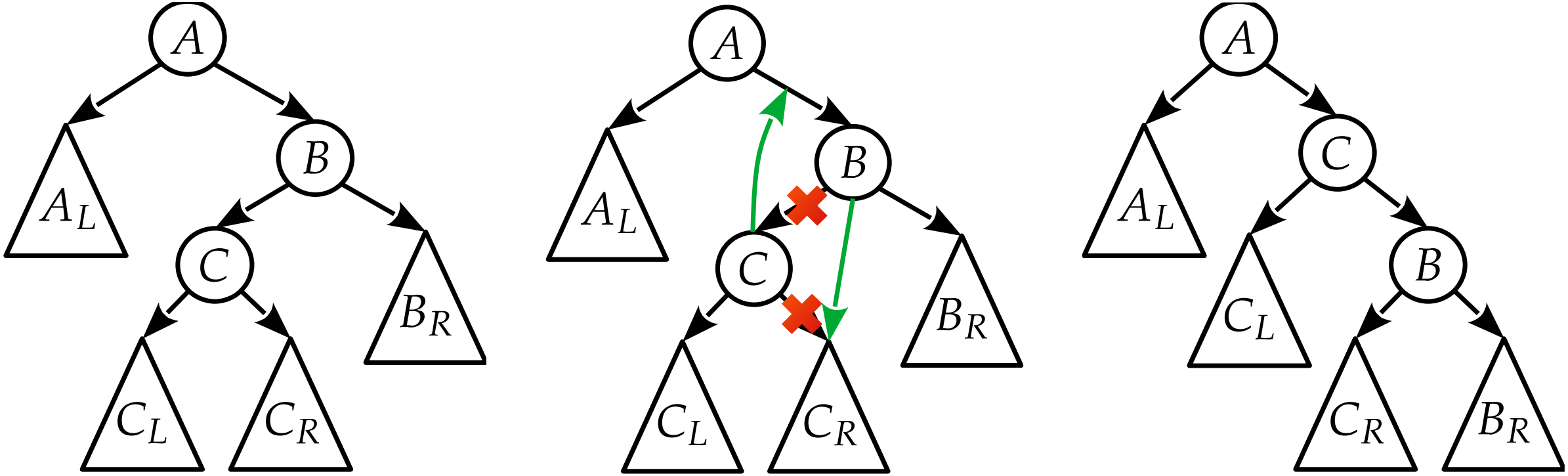


$O(n\ log_2 n + n)$

sort + pass through array

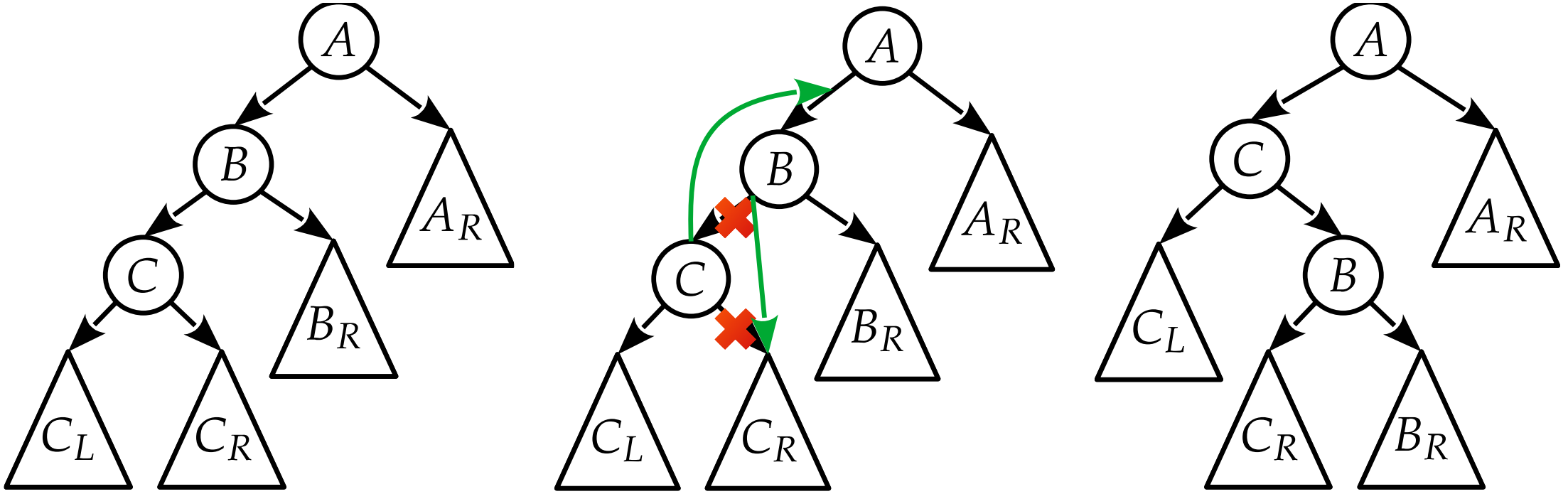
Rotations – right (1)

- C over B
- Get B as C->right



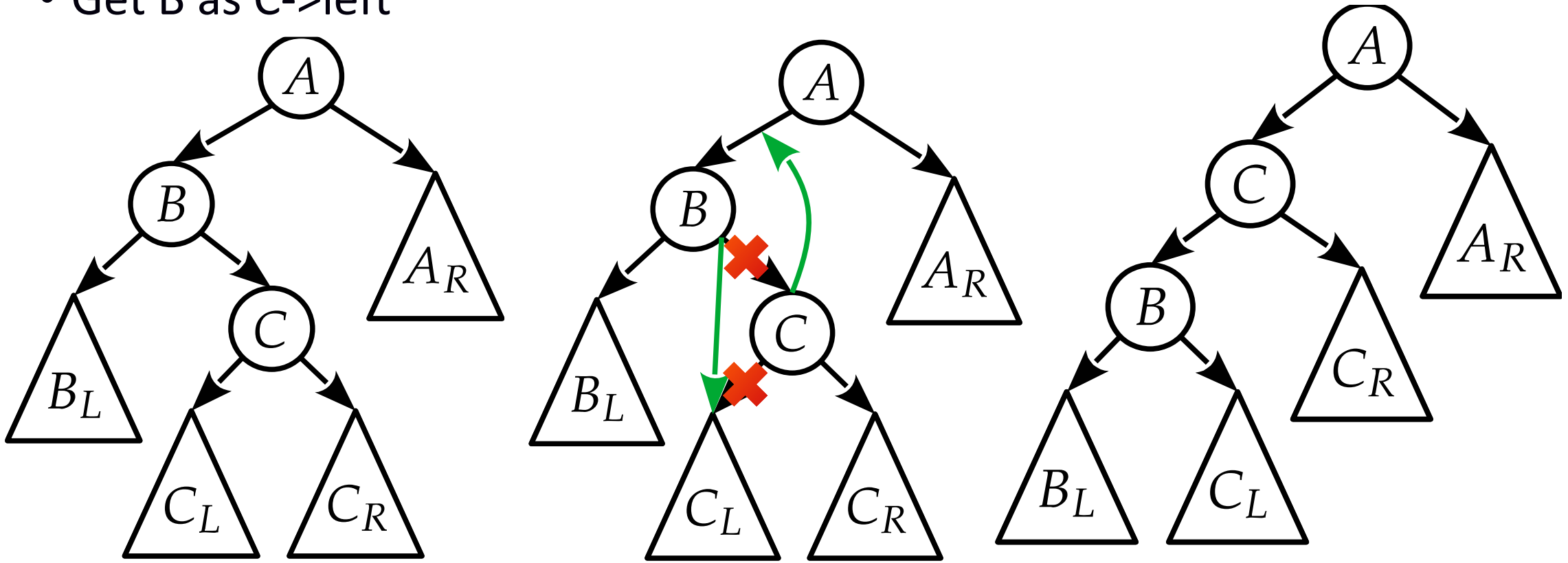
Rotations – right (2)

- C over B
- Get B as C->right



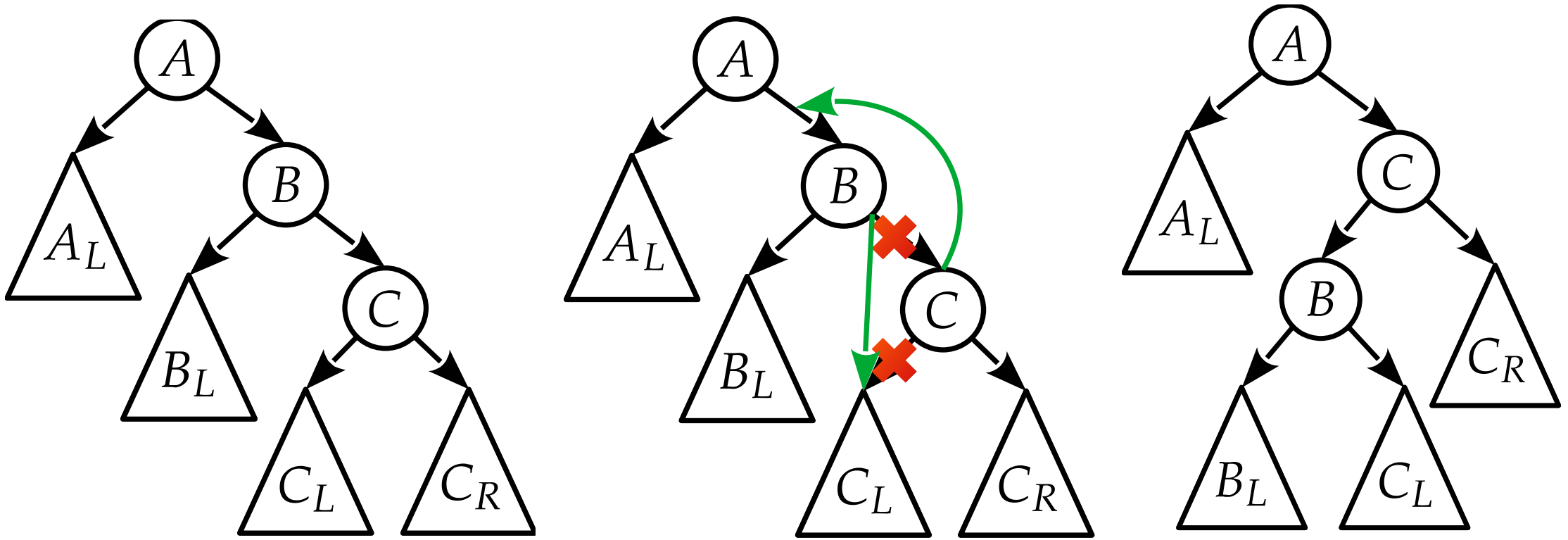
Rotations – left (1)

- C over B
- Get B as C->left



Rotations – left (2)

- C over B
- Get B as C->left



Day-Stout-Warren algoritam (DSW)

1. Create a backbone
   - Continuous rotations to L or R
2. Recursively break the backbone to get a complete tree

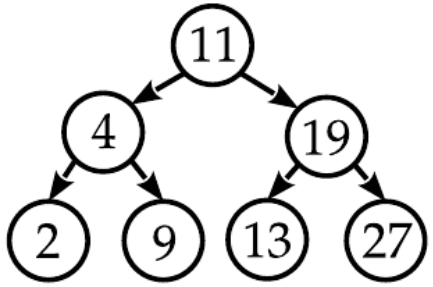
DSW – backbone creation (1)

- Depending on the desired backbone:
  - Left children are rotated right
  - Right children are rotated left
- Repeat until backbone
  - No left children
  - No right children
- `RightRotate(C, B)`
  - Missing something?

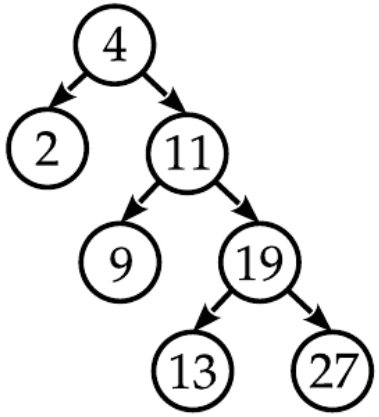
```
procedure RightBackbone(root)
    B ← root
    A ← nil
    while B ≠ nil do
        C ← leftChild(B)
        if C ≠ nil then
            RightRotate(C, B)
            if A = nil then
                root ← C
            B ← C
        else
          ▷ Descending right to the first node that has the left child
            A ← B
            B ← rightChild(B)
```

DSW – backbone creation (2)

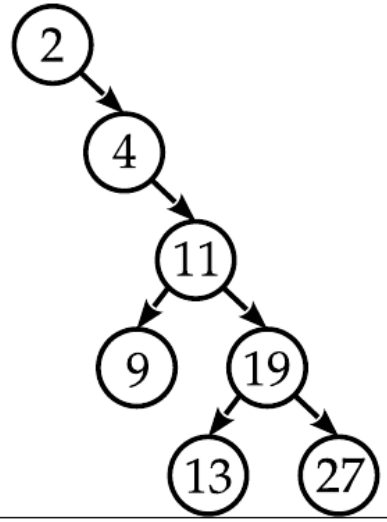
$A = nil, B = 11, C = 4$



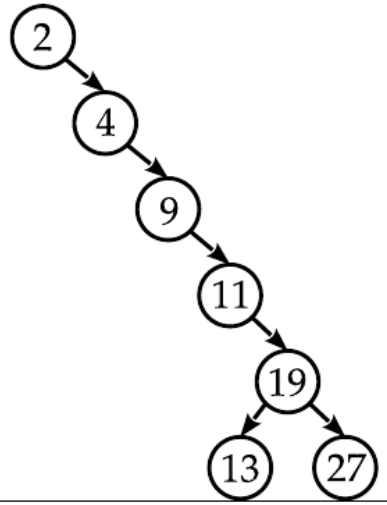
$A = nil, B = 4, C = 2$



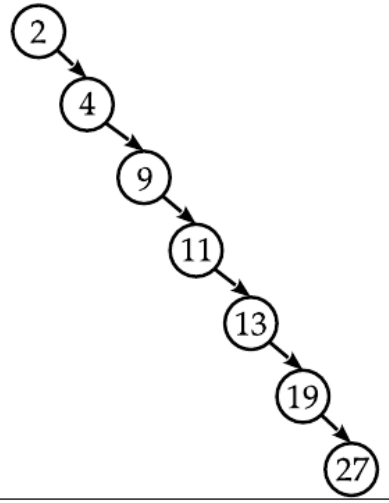
$A = 4, B = 11, C = 9$



$A = 11, B = 19, C = 13$



The final right backbone



DSW – backbone breaking

- Strategically placed left rotations
- Complexity is $O(n)$
- n is known
- Determine number or nodes in the closest complete tree:

$$m = 2^{\lfloor \log_2(n+1) \rfloor} - 1$$

```
createPerfectTree(n):
        m = pow(2,floor(log2(n+1)))-1;
        make n-m rotations from the top of the backbone; //overflowing nodes
        while (m > 1):
                m = m/2
                make m rotations from the top of the backbone;
```

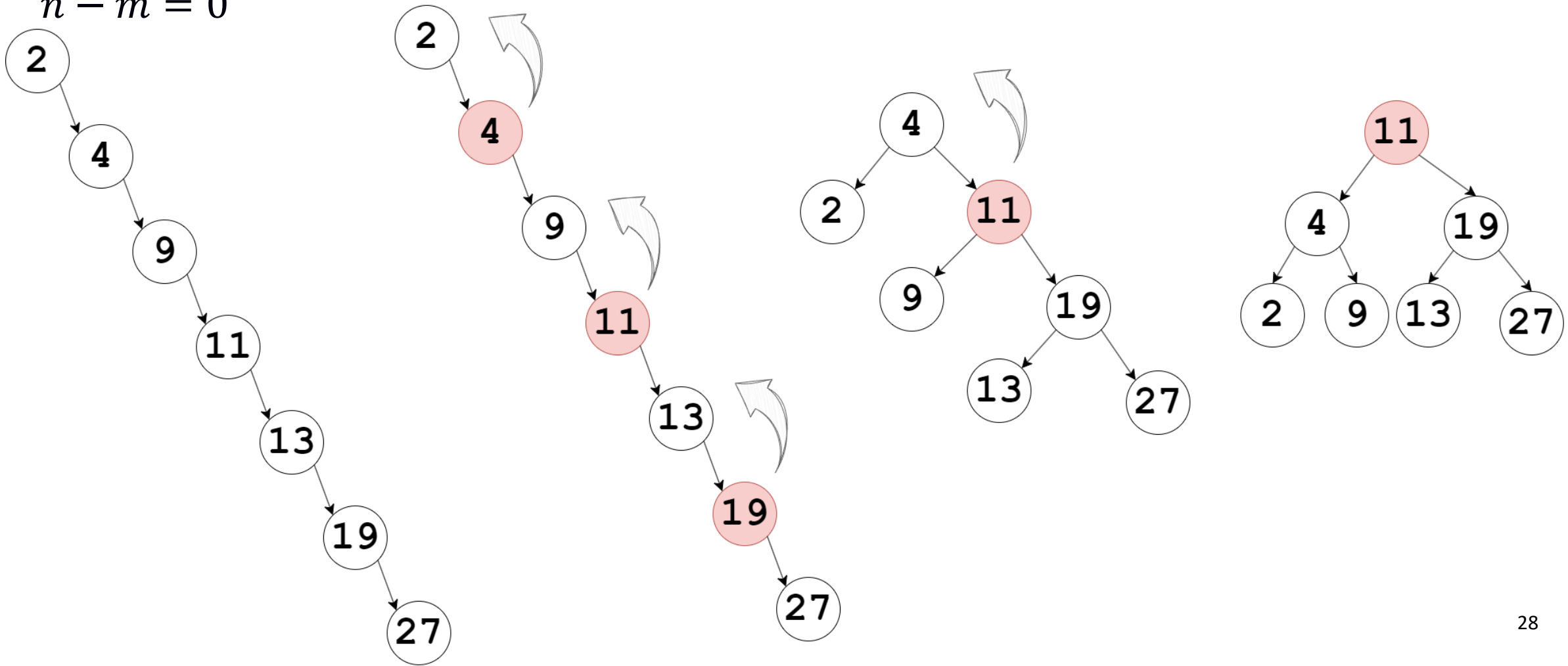
https://liacs.leidenuniv.nl/~rijnjnvan//ds2013/assets/opdrachten/dsw.pdf

DSW – backbone breaking example 1

$n = 7$
$m = 7$
$n - m = 0$

$m = \frac{m}{2} = 3$

$m = \frac{m}{2} = 1$

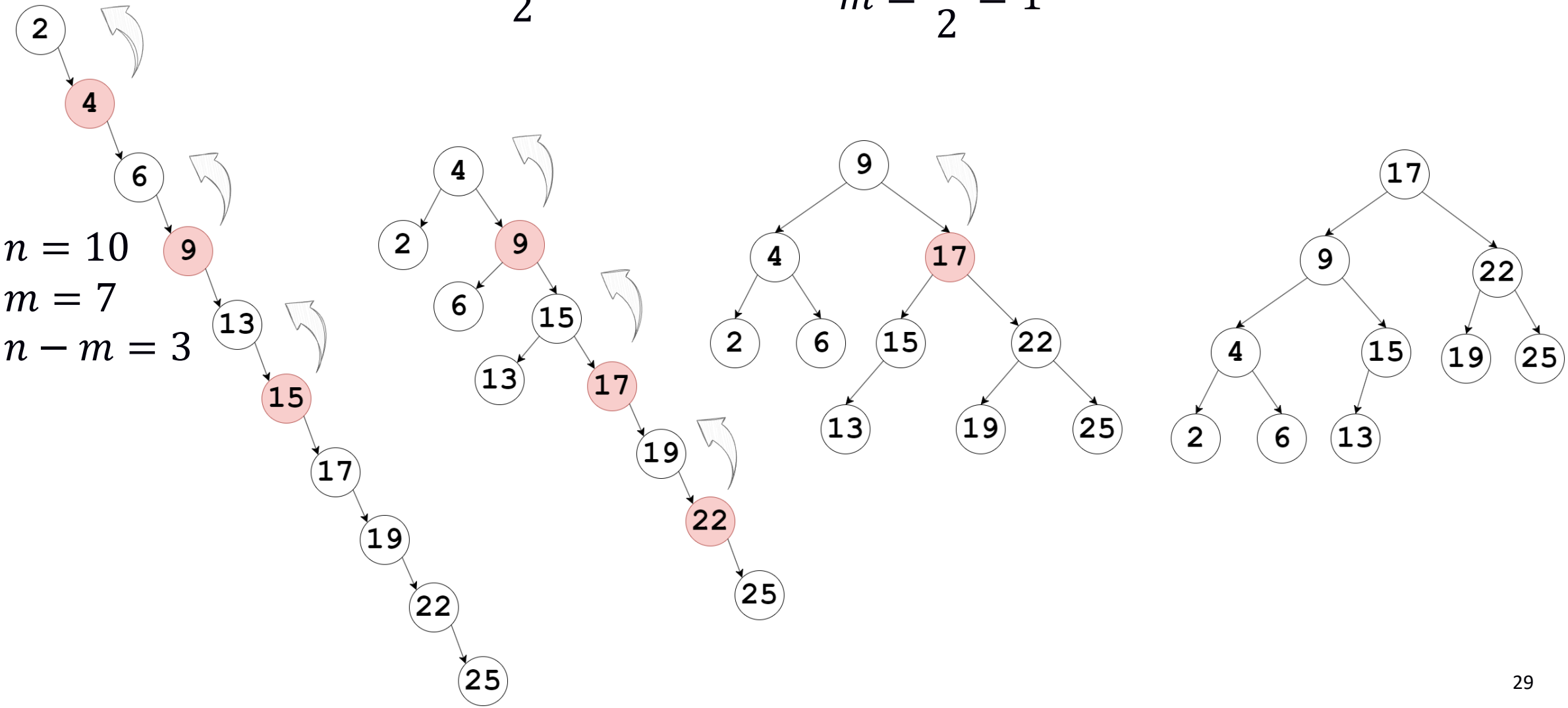


DSW – backbone breaking example 2

$$m = \frac{m}{2} = 3$$

$$m = \frac{m}{2} = 1$$

$n = 10$
$m = 7$
$n - m = 3$



Adelson-Velski-Landis binarno stablo (AVL)

- DSW is offline
- AVL is online
- **Online balancing**
  - Continually check for disbalance after operations
  - Fix the disbalance
- Balances with a minimal impact on structure -> local balancing
- Balances by height

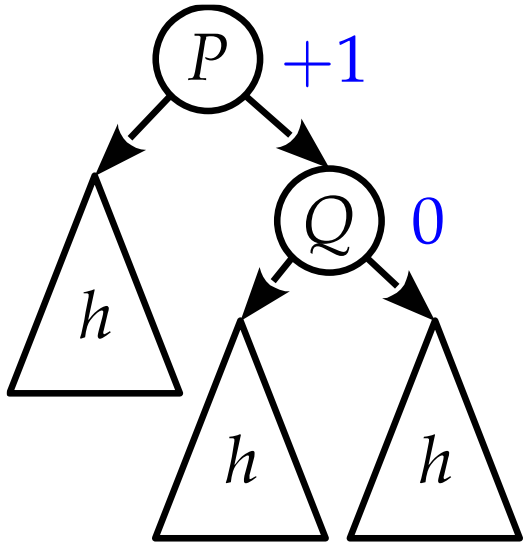
AVL (2)

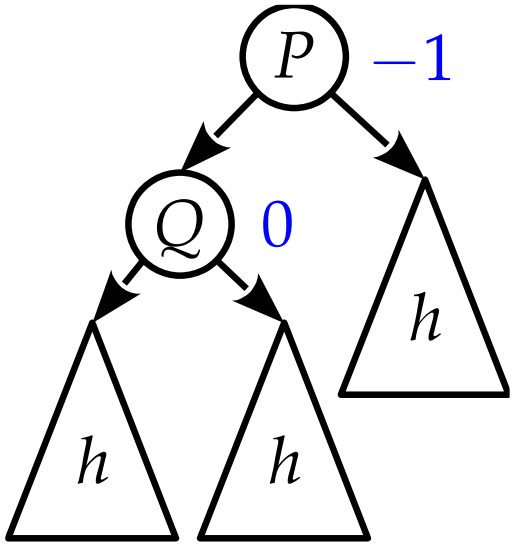
- We check a **balancing factor (BF)** for each subtree
  - Recursive

$$BF(S) = h(R) - h(L)$$

- We consider a subtree balanced if:

$$-1 \leq BF(S) \leq 1$$





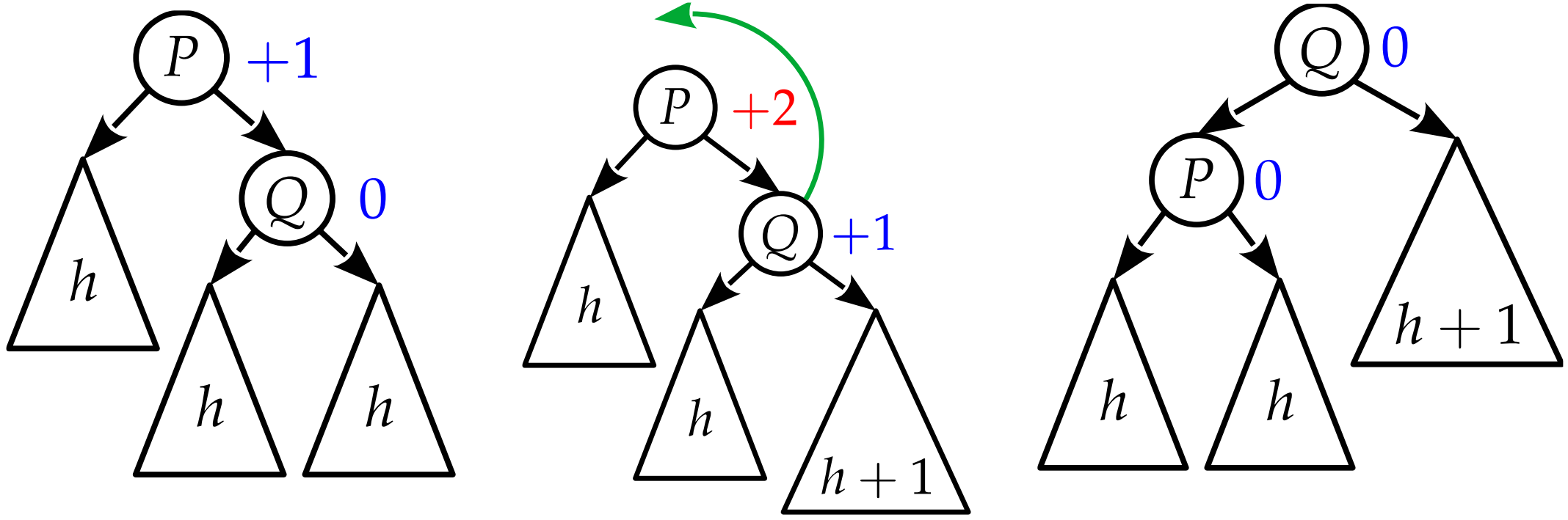
AVL (1)

- After each operation, we update the BF
- If a $BF(S) = -2$ or $BF(S) = 2$ appear, we balance the tree
  - We usually don't let it go far over
- 4 scenarios

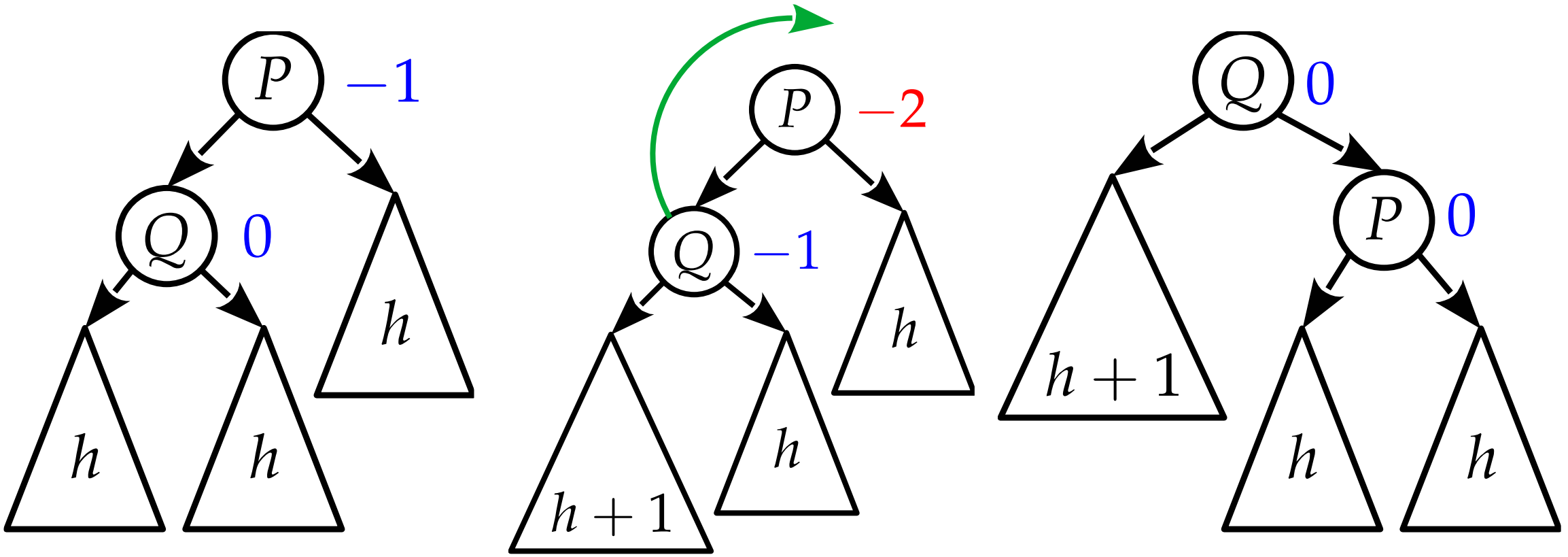
AVL – Balancing scenarios (2)

Scenario	Condition	Operation
Right Right	higher subtree Z is to the **right** and has **BF(Z) ≥ 0**	rotate left X
Left Left	higher subtree Z is to the **left** and has **BF(Z) < 0**	rotate right X
Right Left	higher subtree Z is to the **right** and has **BF(Z) < 0**	rotate right left X
Left Right	higher subtree Z is to the **left** and has **BF(Z) ≥ 0**	rotate left right X

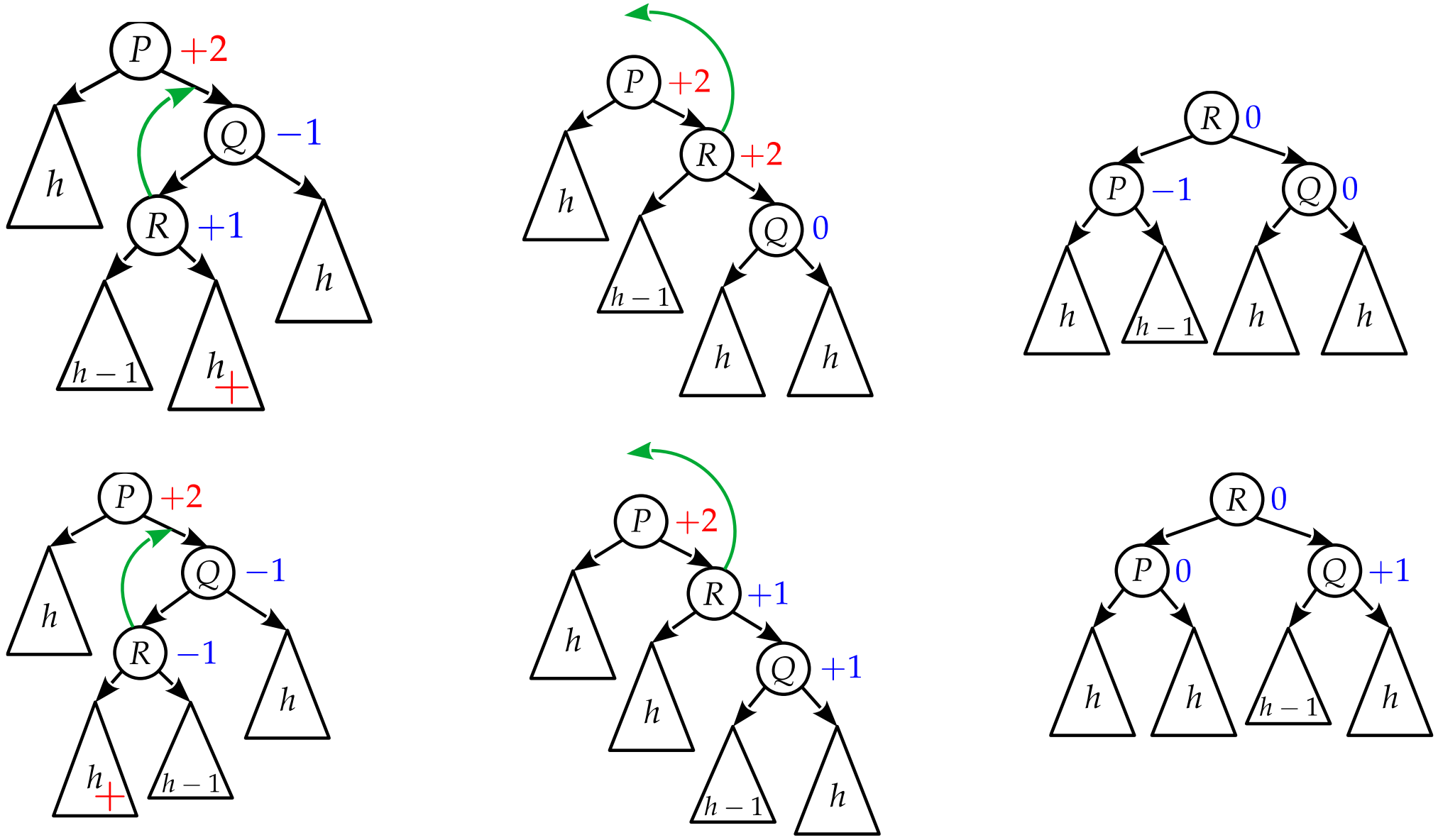
AVL – Right Right (3)



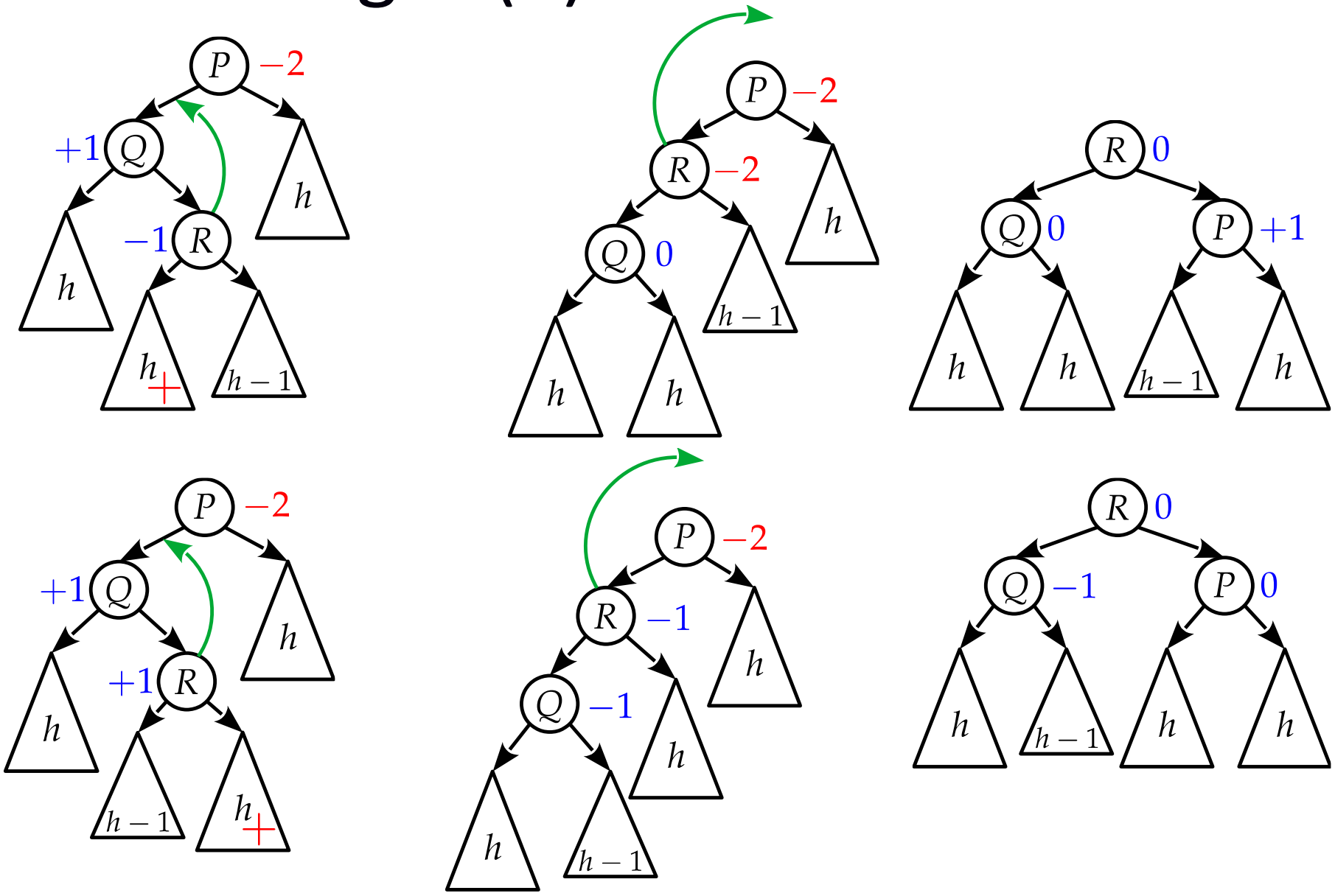
AVL – Left Left (4)



AVL – Right Left (6)



AVL – Left Right (6)



AVL (7)

- **Always copy to delete**
  - Least structural change
- Search: $O(log_2 n)$
- Insert or delete: $O(2log_2 n)$

- Theoretical height of an AVL tree:

$$log_2(n+1) \leq h \leq \mathbf{1.44}\, log_2(n+2) - 0.328$$

https://dl.acm.org/doi/10.5555/525639

02 – Trees 2

Advanced Algorithms and Data Structures

UNIVERSITY OF ZAGREB
Faculty of Electrical Engineering and Computing

Creative Commons

- You are free to:
  - share — multiply, distribute, and publicly communicate the work
  - adapt the work
- under the following conditions:
  - **Attribution:** You must acknowledge and indicate the authorship of the work in the manner specified by the author or license provider (but not in a way that suggests you or your use of the work have their direct support).
  - **Non-commercial:** You may not use this work for commercial purposes.
  - **Share alike:** If you modify, transform, or build upon this work, you may only distribute the modified work under the same or a similar license.

Red-Black Tree (1)

- Binary tree with 2 types of nodes:
  - **RED**
  - **BLACK**
- Constrained by the colors on paths from root to leaf
  - Balanced tree
- Balances horizontally
- Metrics
  - (max.) height: $h \leq 2log_2(n+1)$
  - search complexity: $O(log_2 n)$

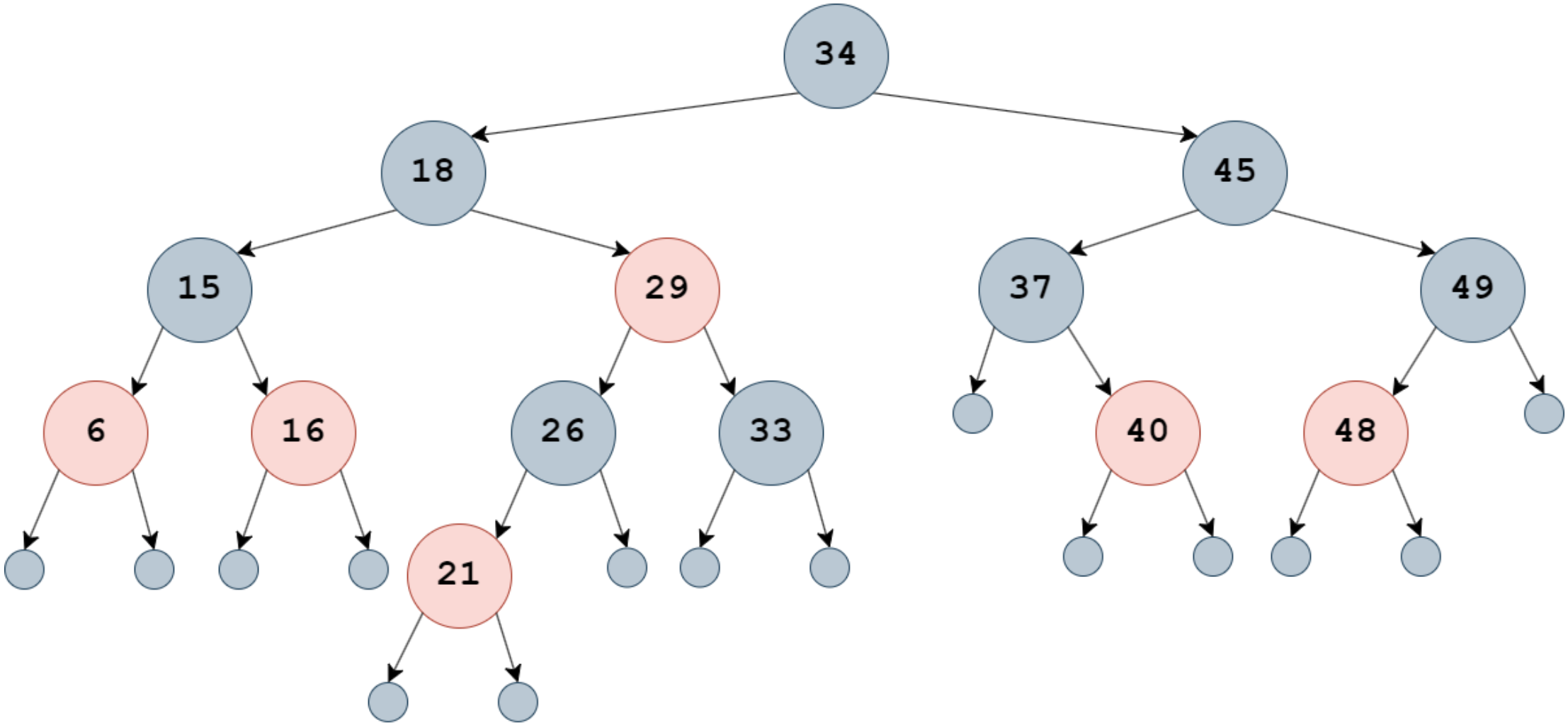
Red-Black Tree – definition rules

1. Every node is **red** or **black**
2. Root is **black** (not mandatory; usually switched from red to black)
3. Every leaf – **NIL** – is **black**
4. If a node is **red**, its children are **black**
5. For each node, all simple paths from the node to leaves contain the same number of black nodes.
   - Red and black heights $h_R(x)$ and $h_B(x)$ for node x

- The longest path root->leaf is at most twice as long as the shortest path root->leaf

*Note: NIL leaves don't contain keys so they don't have to *really* exist*

Red-Black Tree – example (3)



Red-Black Tree - adding a node

1. Add the node like with any sorted binary tree, **but make it Red**
2. Restructure the tree to adhere to definition rules (DRs)
   - Rotations + Recoloring

- Adding a node doesn't break DRs 1, 3, and 5

1. DR 2 is broken if the new node is a root
2. DR 4 is broken if the parent of a new node is **Red**

} **Restructuring!**

RB Tree – restructuring

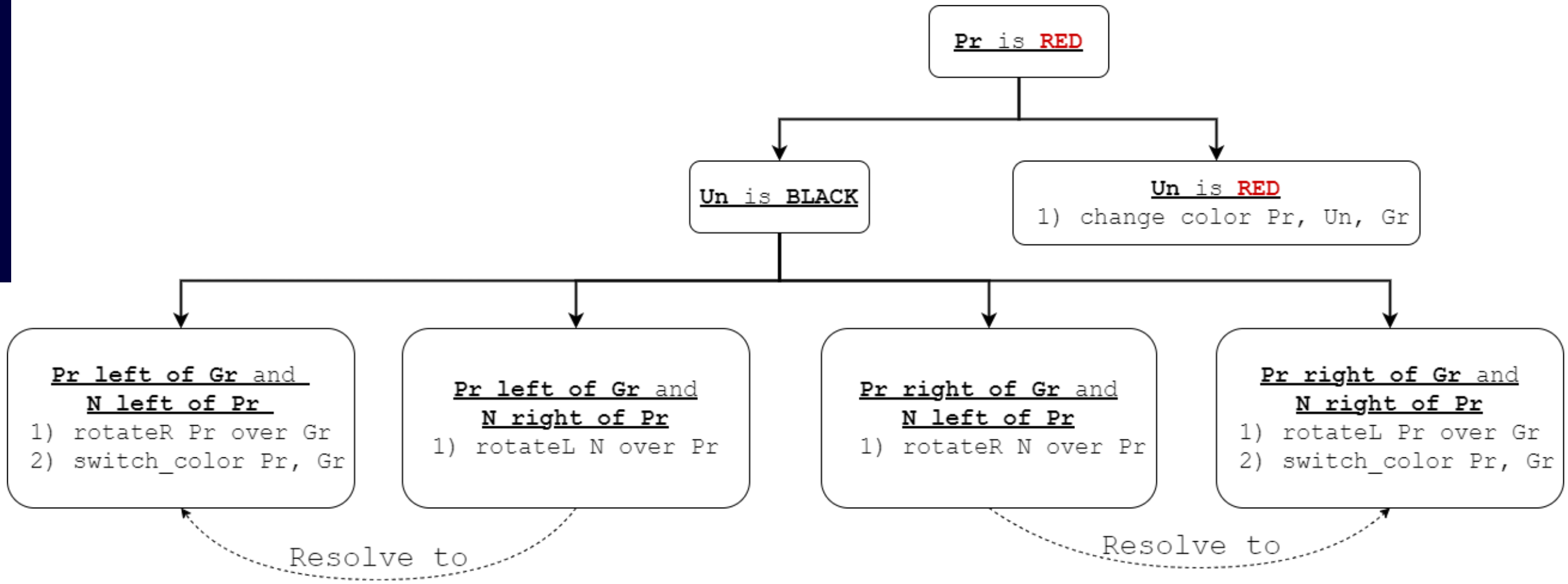
Sc. 1 New node is a root

- DR 2 is broken
- Change the color to **Black**

Sc. 2 Parent of the new node is **Red**

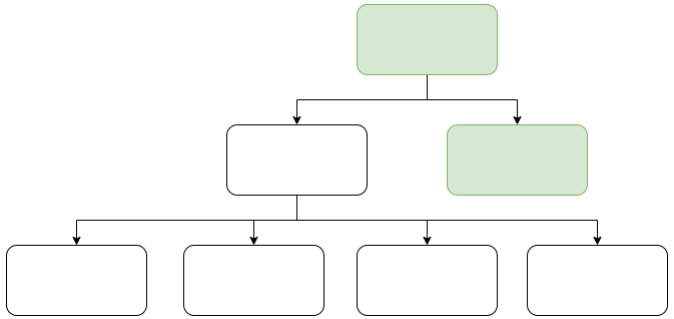
- Decision tree!

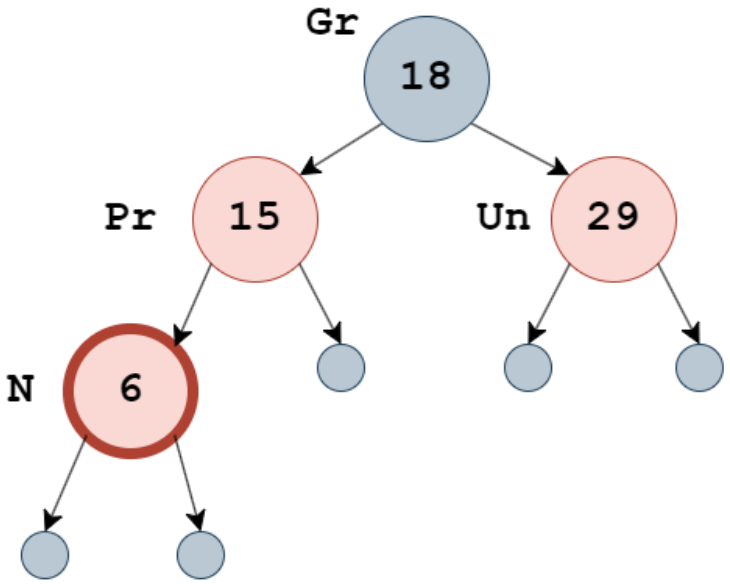
RB Tree – Sc. 2 restructuring

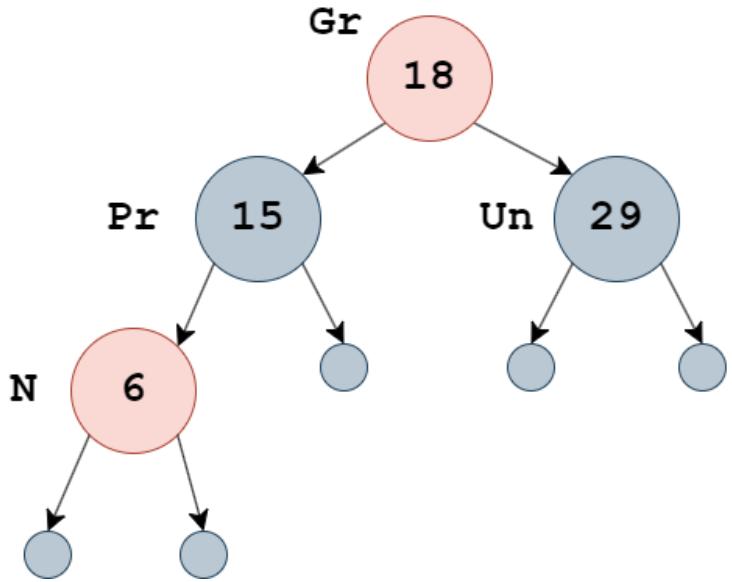


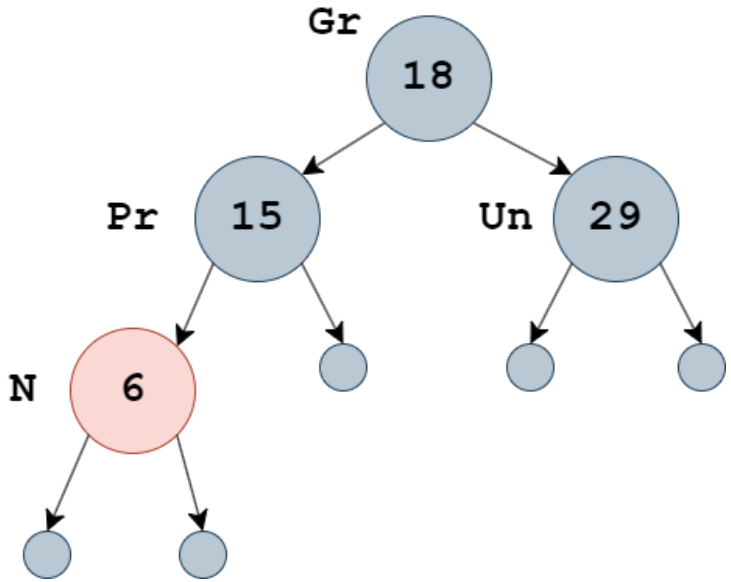
RB Tree – restructuring (1)

1. Added 6 as **red**
2. Switch colors of Pr, Un, and Gr
3. Switch back root to be **black**

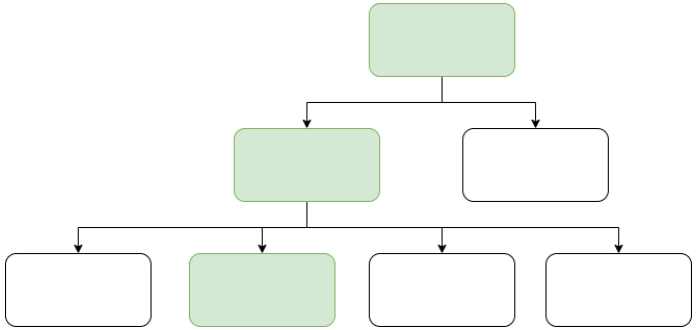




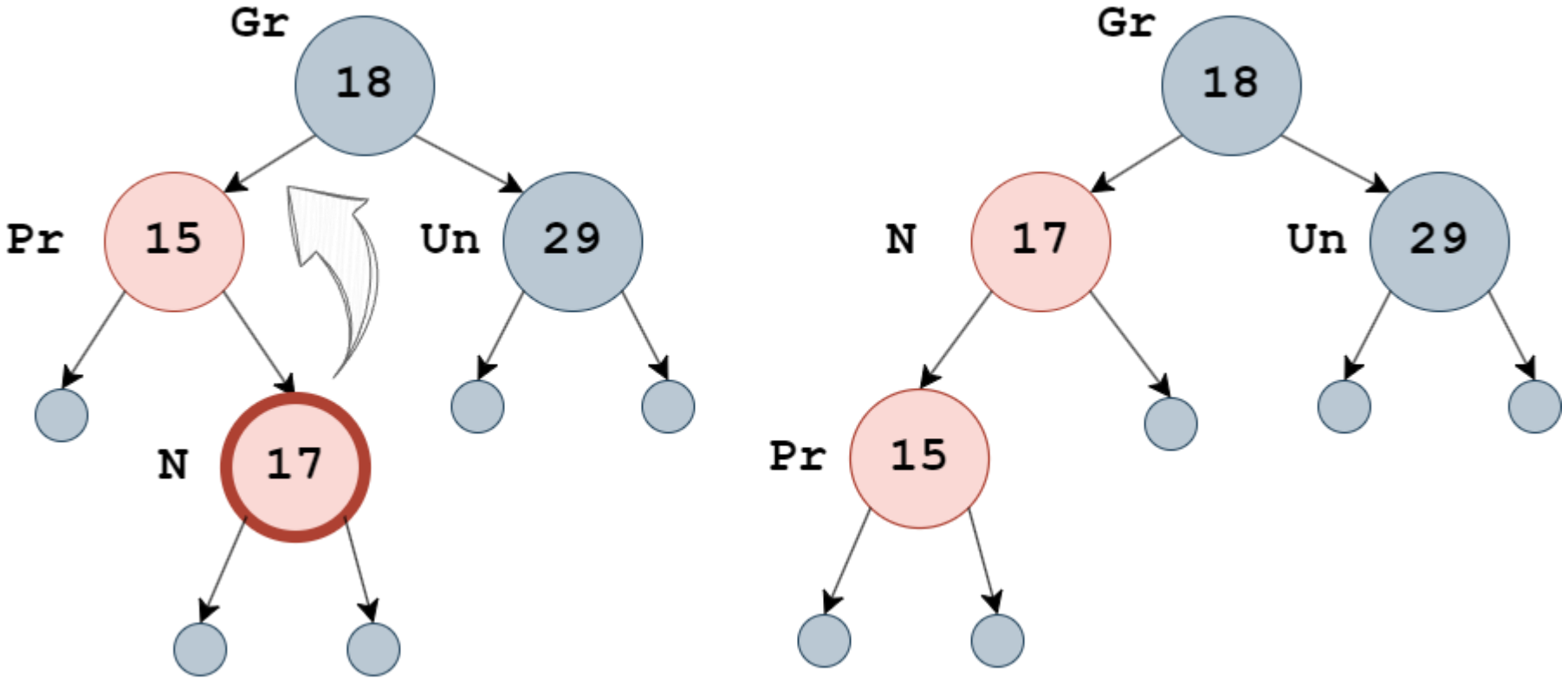




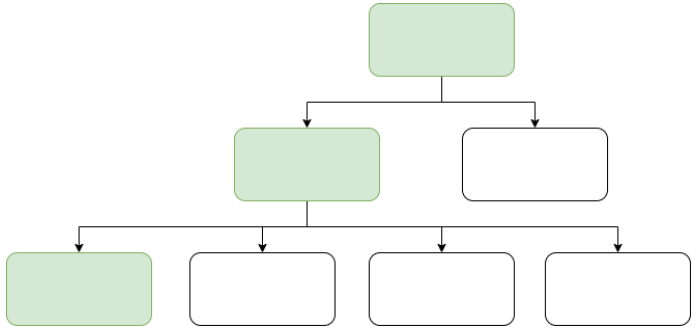
RB Tree – restructuring (2)

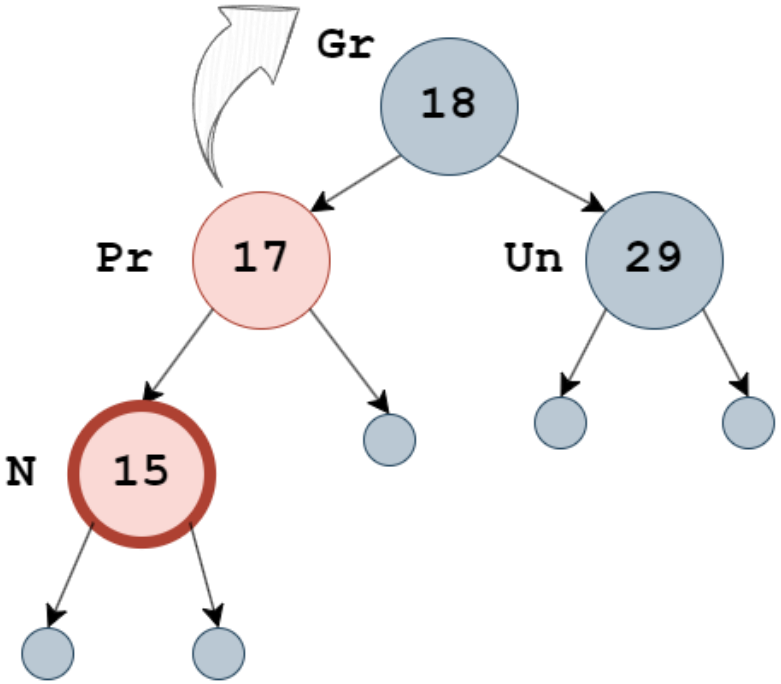


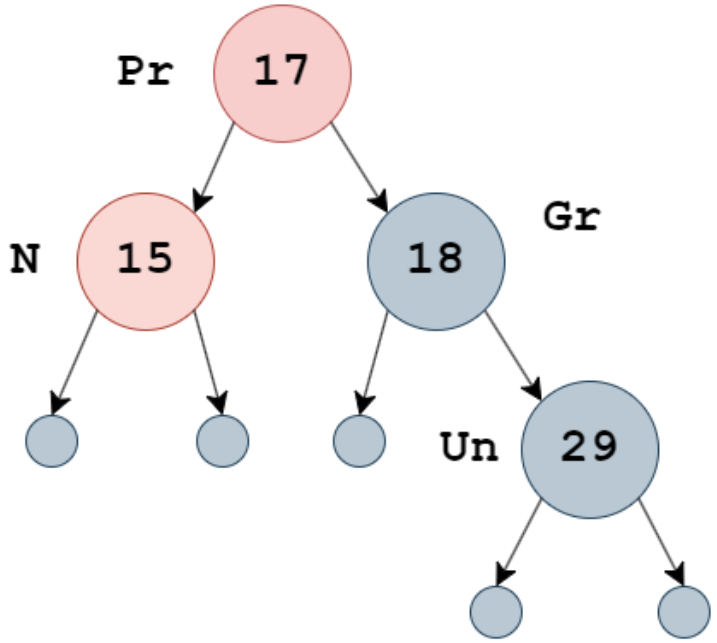
New case!

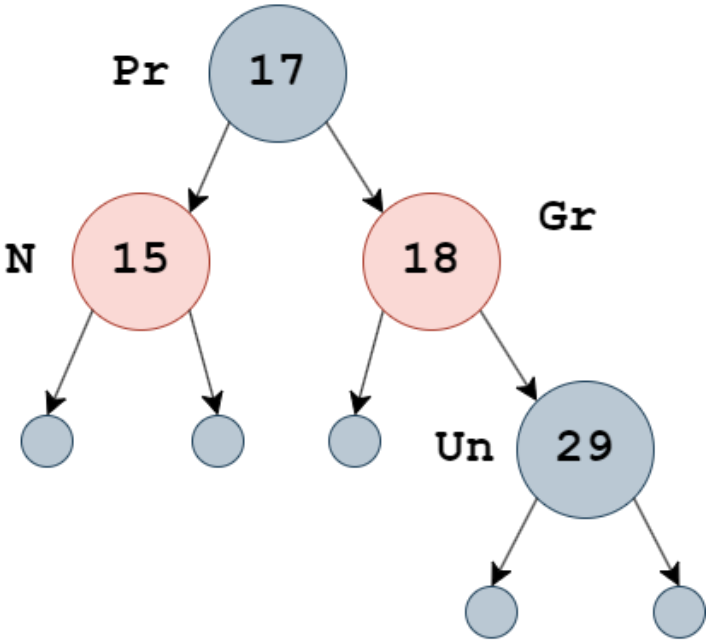


RB Tree – restructuring cont. (3)



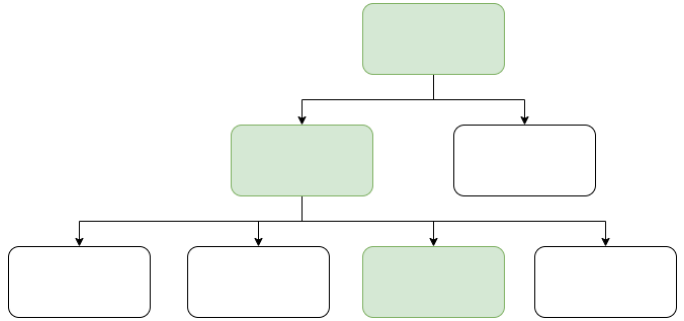




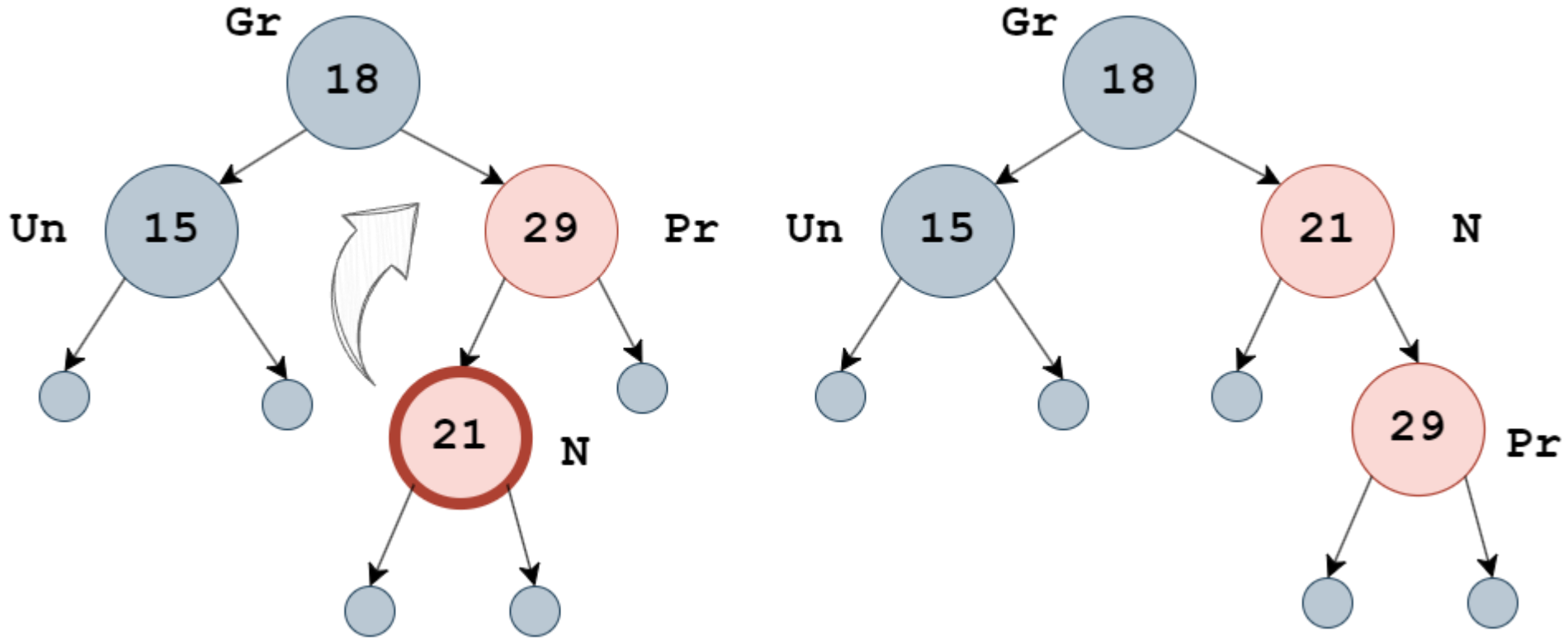


**Note change in relation names*

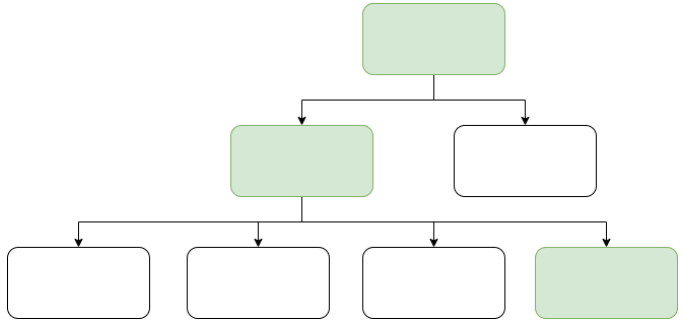
RB Tree – restructuring (4)

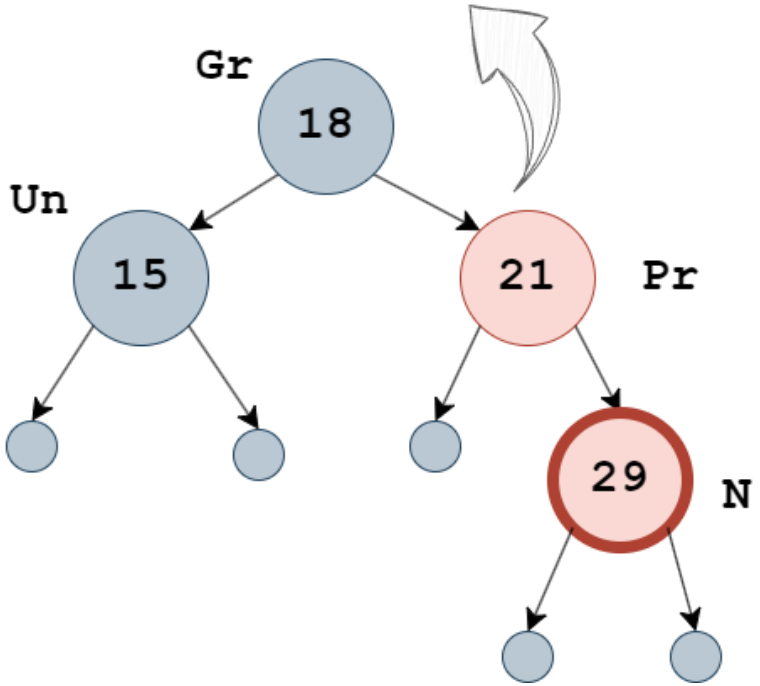


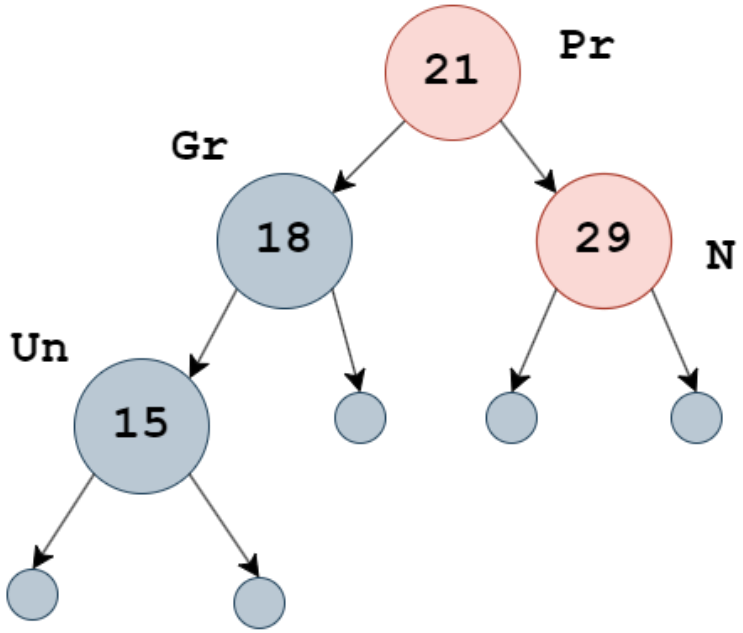
Symmetric case!

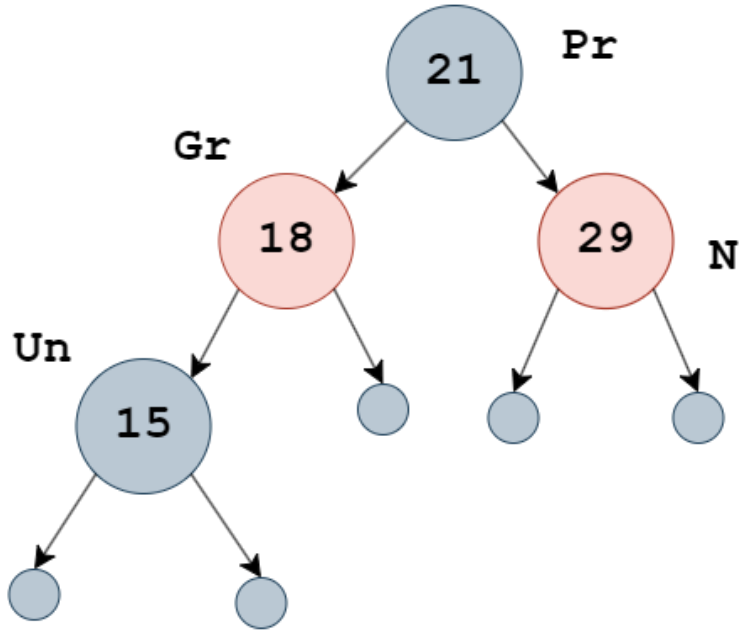


RB Tree – restructuring cont. (5)









**Note change in relation names*

RB Tree – deleting a node (1)

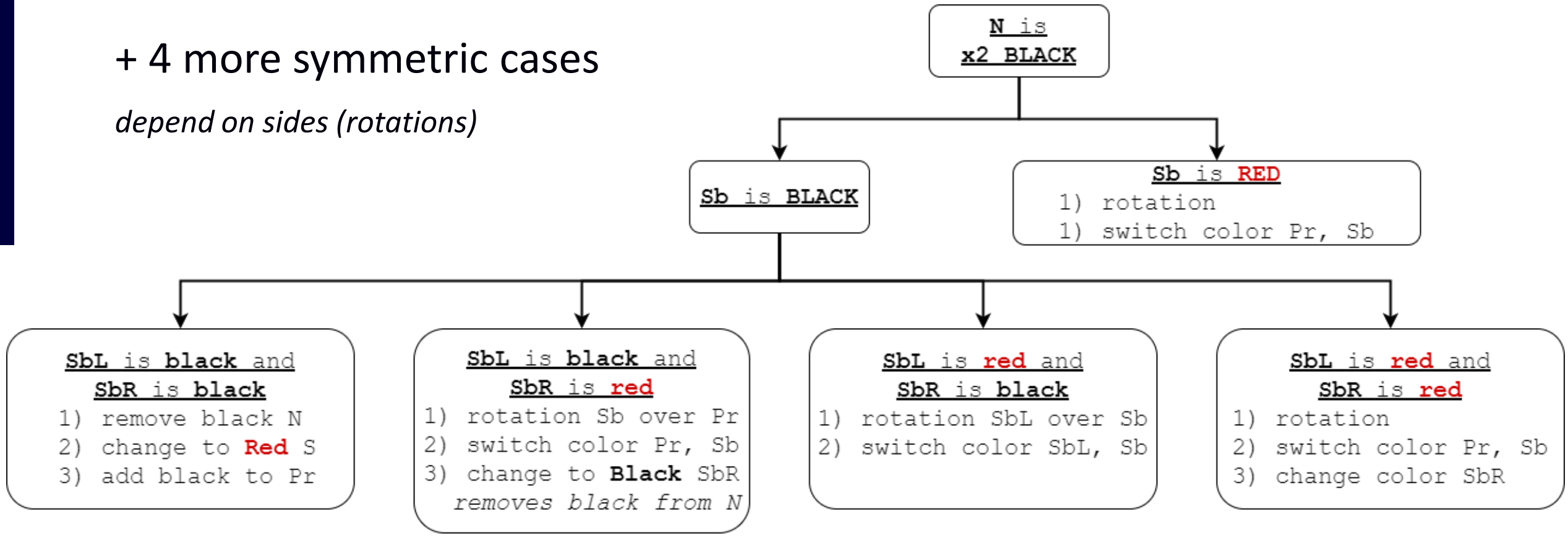
1. Delete by copying
2. Remove replacement node
   - If **Red** – done!
   - If **Black** – a complex procedure…

- **Double black** marking
  - Root node get **x2 black** removed automatically

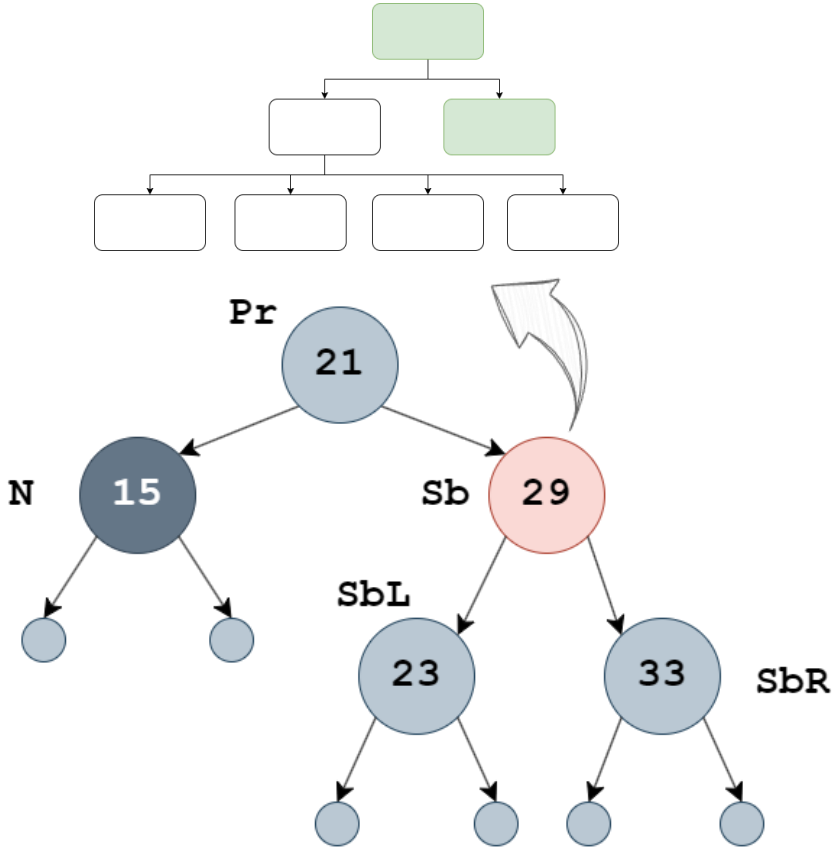
RB Tree – deleting a node (2)

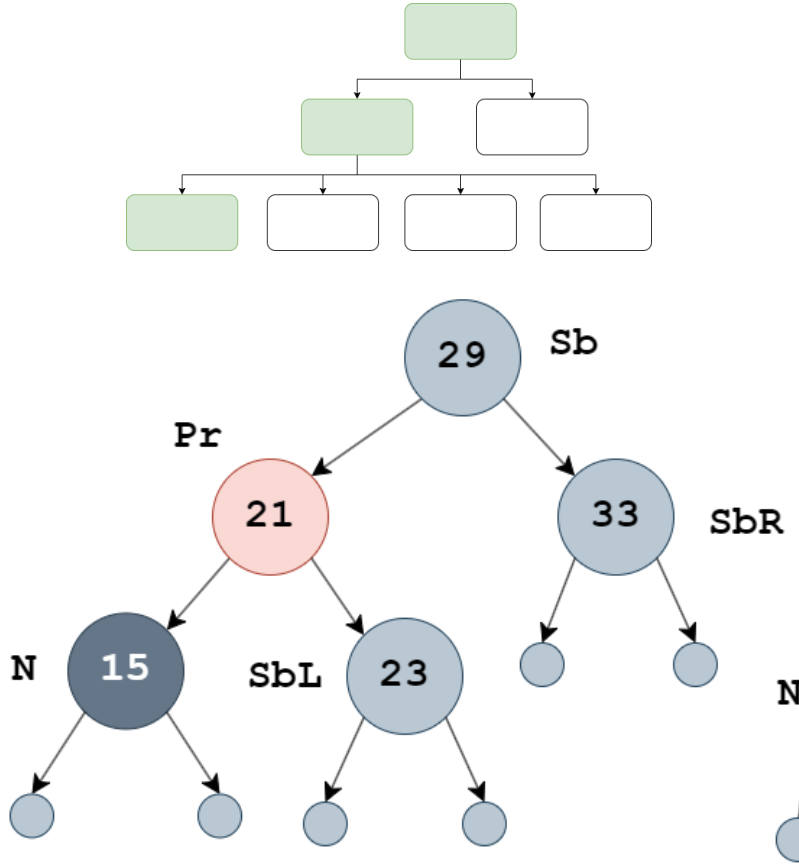
+ 4 more symmetric cases

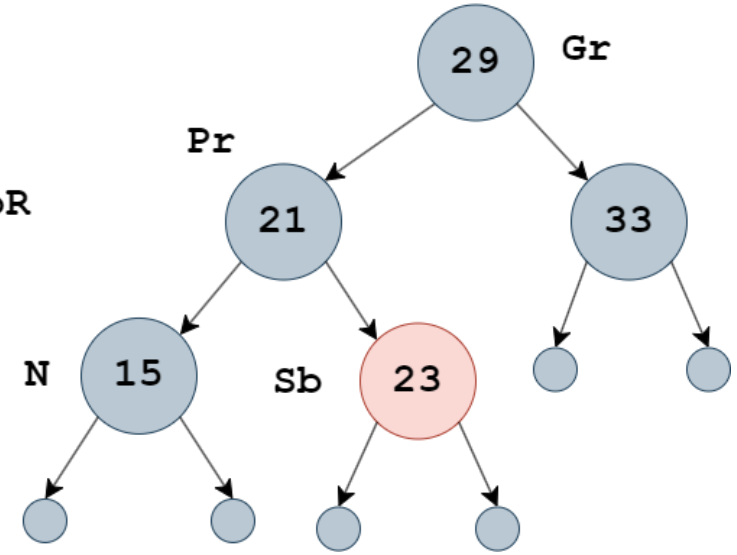
depend on sides (rotations)



RB Tree – deleting a node (3)



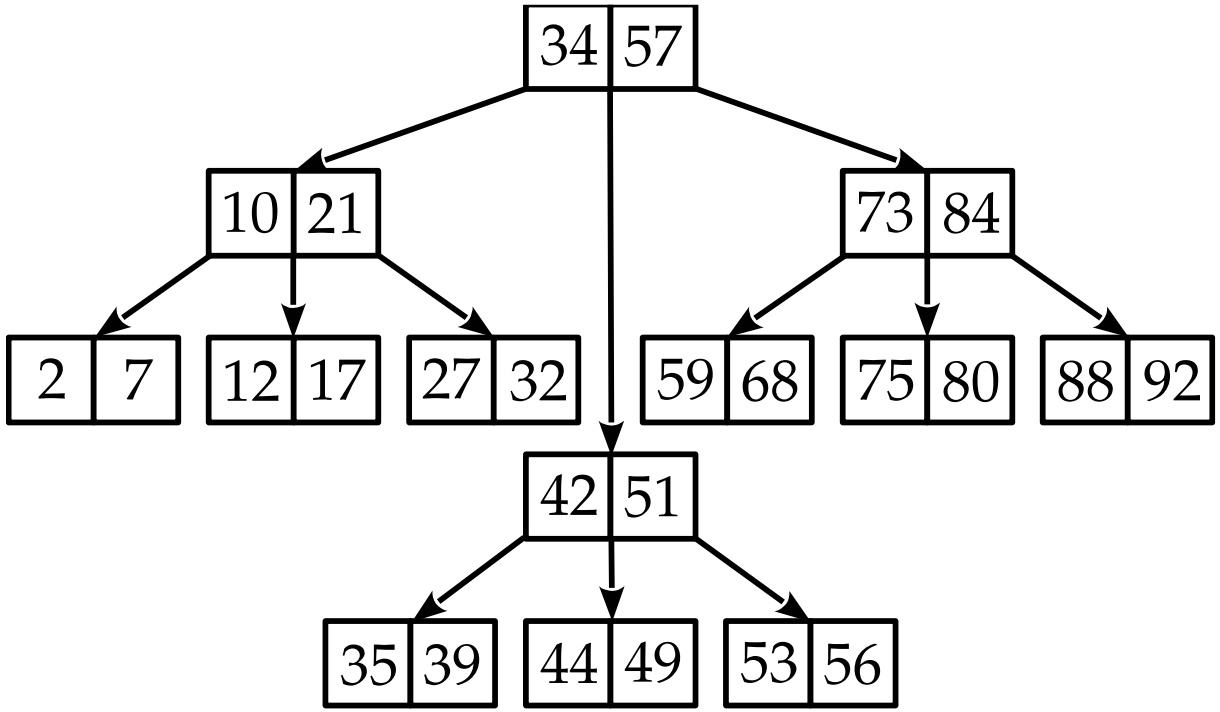




Note: renaming after step 2

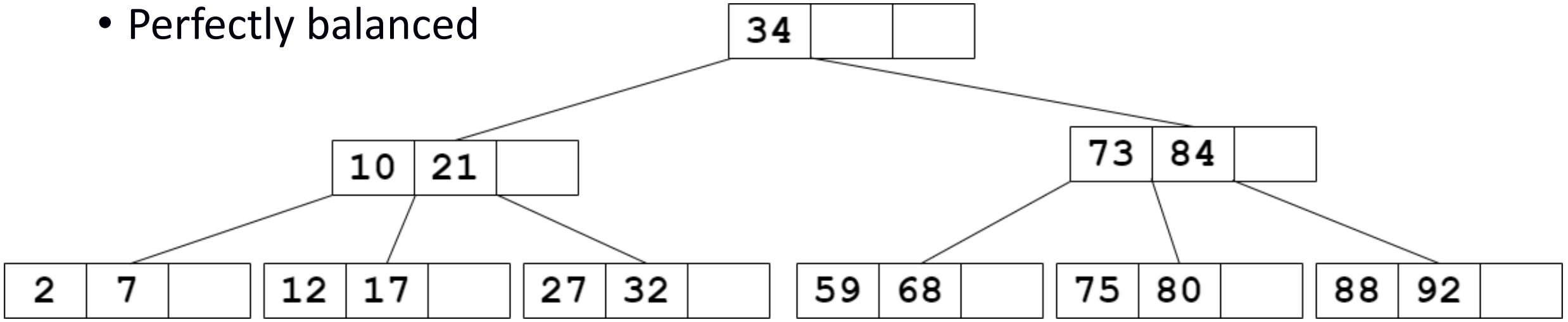
Multiway tree

- Mth order of children
- M children
- M-1 data



B-tree (1)

- Special case of M-tree
- Good for sequential memory read
  - Database indexes
- At least 50% capacity filled – not including the *root*
- Perfectly balanced



B-tree (2)

- M-tree with added features:

1. Root has at least 2 children; unless root = leaf
2. Every node has x-1 keys and x sub-tree pointers, while:

$$\lceil m/2 \rceil \leq x \leq m$$

3. All leaves have at least x-1 keys, while:

$$\lceil m/2 \rceil \leq x \leq m$$

4. All leaves are at the same level – perfectly balanced

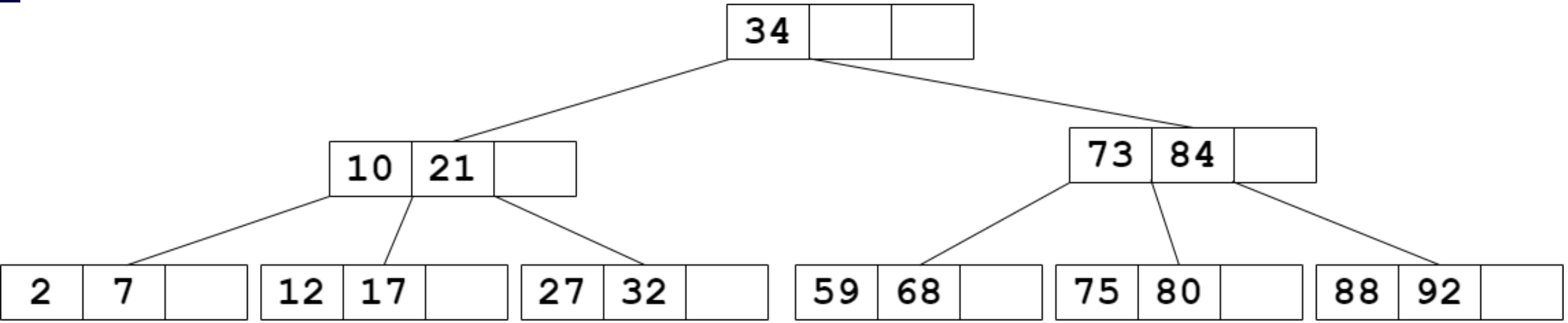
B-tree – search (1)

Searching for a key for a value

1. Enter a node and seq. read keys while:
   - key > value AND there are unread keys
   - HALT if key == value
2. If value > key OR end of the node is reached
   - Descend to the lower level
   - If there is no lower level – KEY NOT FOUND

B-tree – search (2)

- Find 17, 59, 84, 18

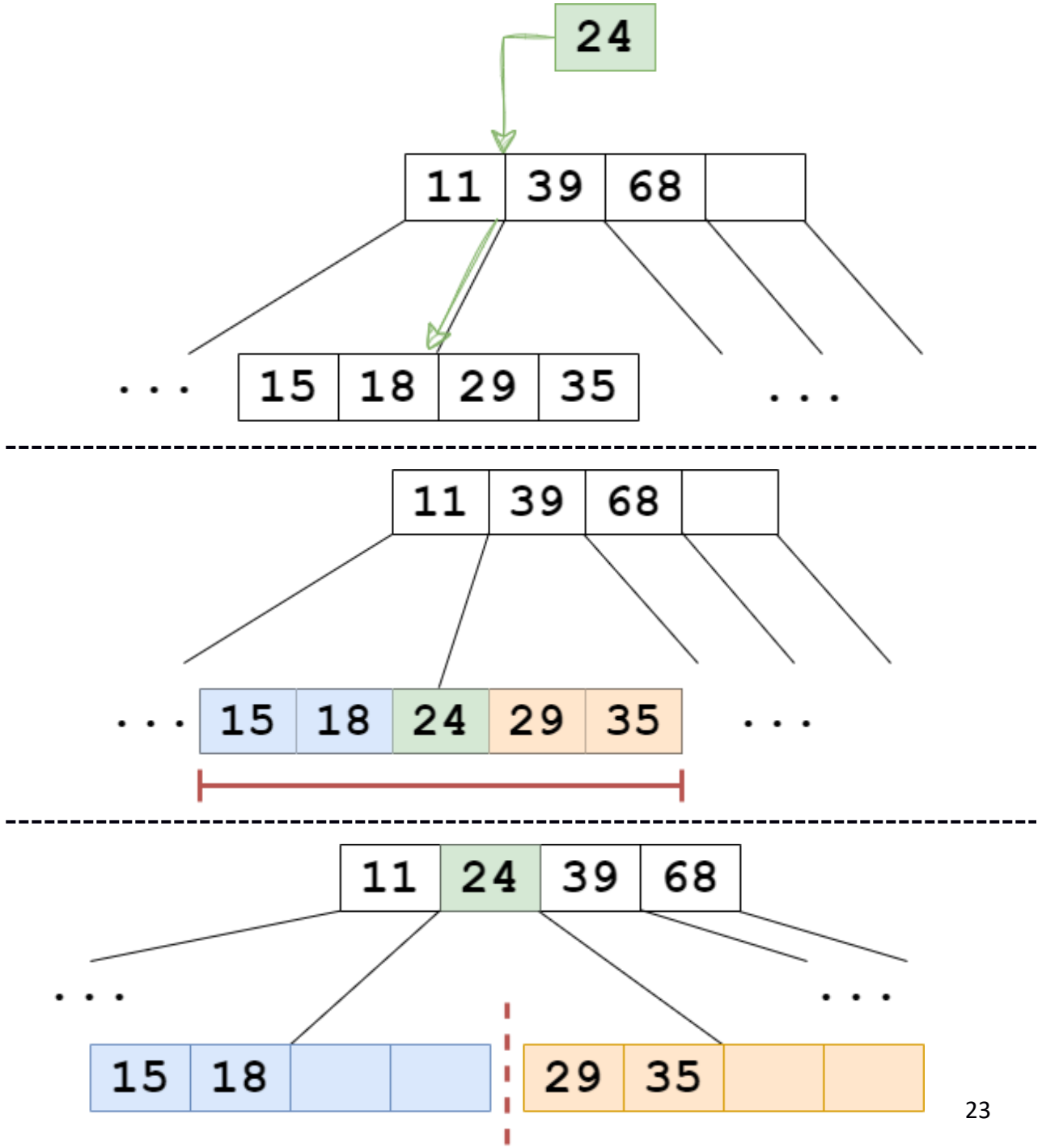


B-tree – insertion

1. Search for the appropriate leaf node
2. If there is space in the leaf, place the key
3. If there is no space in the leaf, split the node equally into two nodes
   - The middle key goes into the parent node of the leaf node
4. If the parent node has no space, continue the splitting upward

B-tree – splitting while inserting (1)

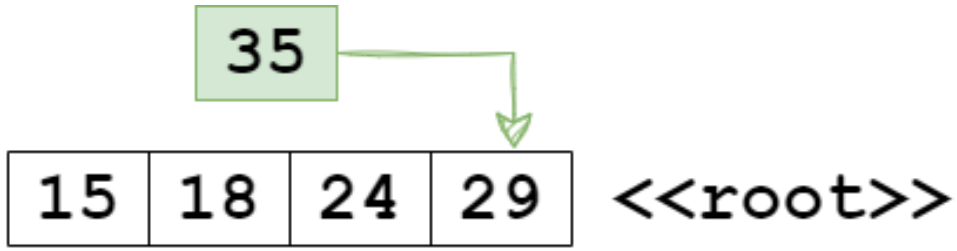
- m = 5 -> m-1 keys and k pointers
- Add 24
- n > m-1 ?
  - Split!

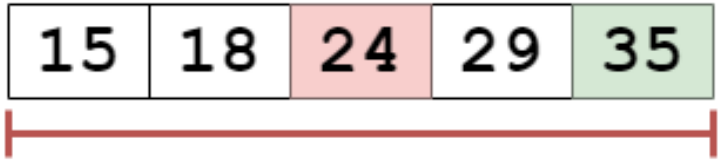


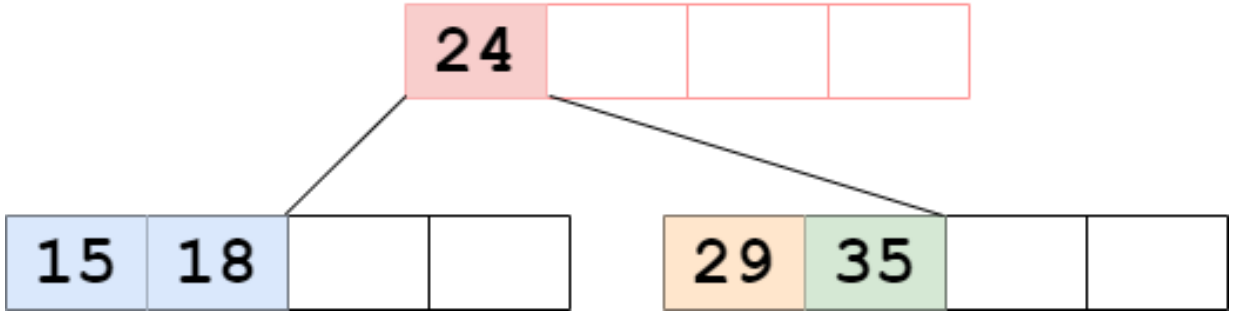
B-tree – splitting while inserting (2)

- Splitting a root node

- Preemptive splitting







B-tree – deletion (1)

Sc. 1 – target is a leaf key

- Delete leaf key

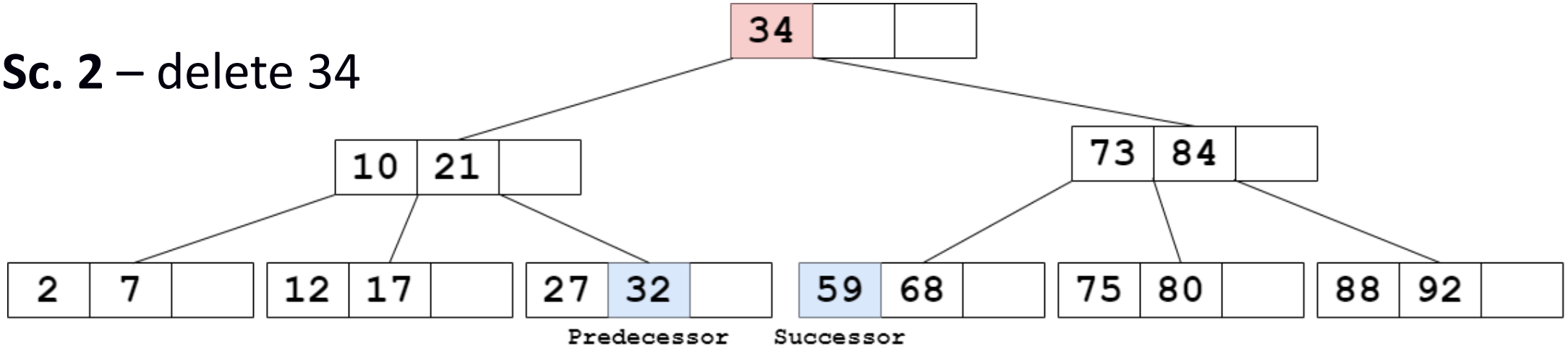
Sc. 2 – target is a root key

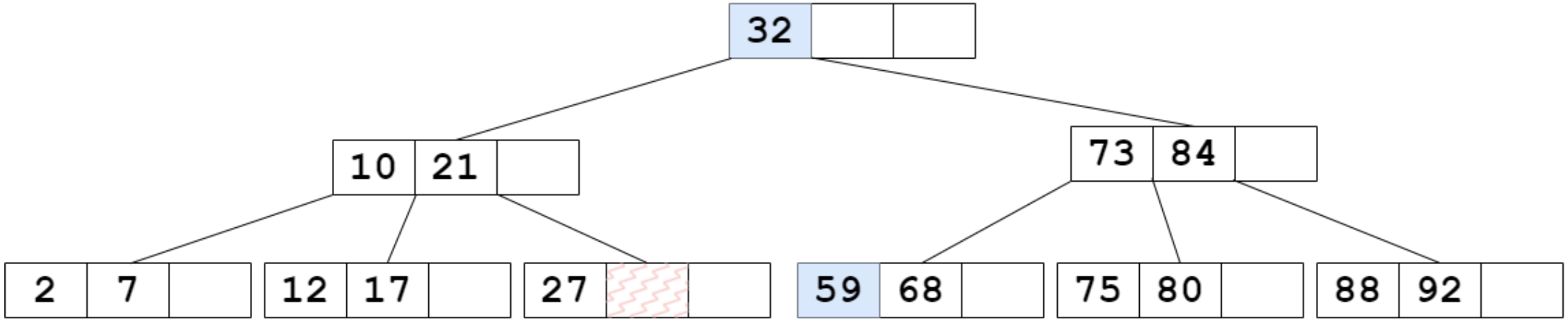
- Find a predecessor (closest value in a leaf)
- Delete the leaf key and place the predecessor key into it (eff. delete by copy)

- Preemptive merging
  - While searching – “steal” greatest key from ChL or least key from ChR

B-tree – deletion (2)

Sc. 2 – delete 34



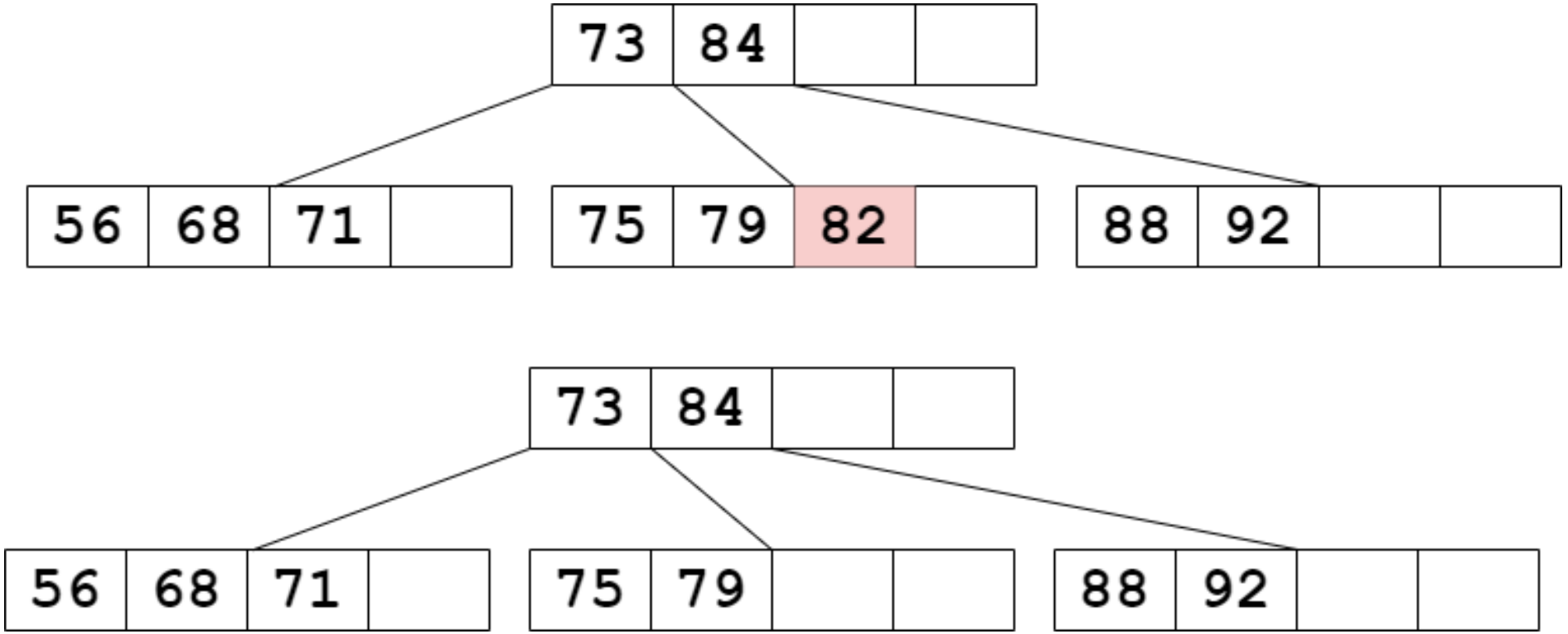


B-tree – state after deletion

1. Leaf still has N keys ≥ ⌈m/2⌉ - 1 -> **HALT**
2. Leaf has N keys < ⌈m/2⌉ - 1
   - **IF** SbL or SbR with N keys > ⌈m/2⌉ - 1
     - Union of target leaf keys, Pr key, and Sb keys(order is important)
     - Separate into new equally filled SbL' and SbR' and a (new) Pr key -> **HALT**
   - **ELSE** Union of target leaf, Sb, and Pr key
     - They form a new leaf -> **EVALUATE Pr**
   - **ELSE IF** target is *root*
     - **IF** root has N keys > 1: Union of target and Sb -> **HALT**
     - **ELSE** Union of target, sb, and root into *new root* node -> **HALT**

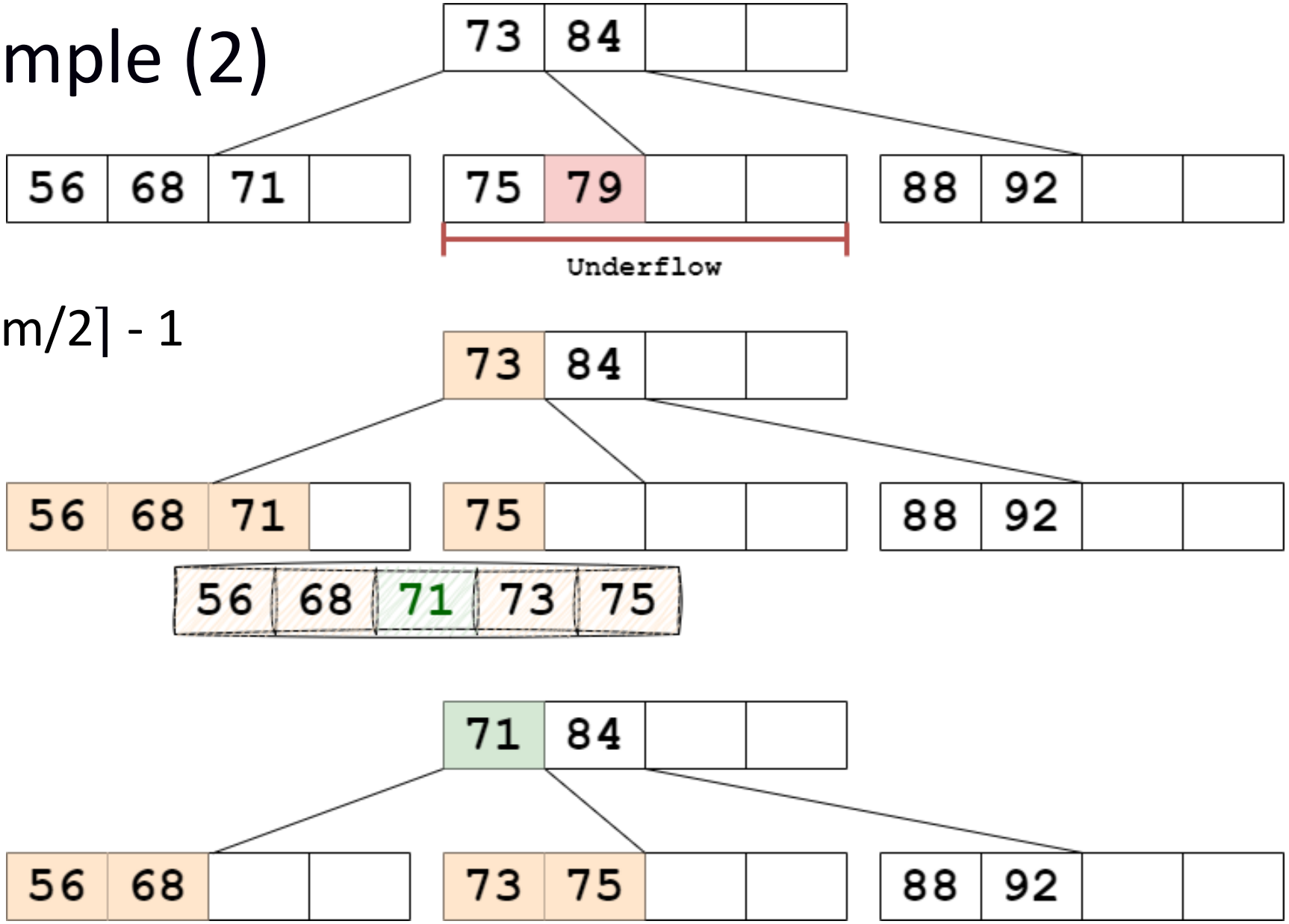
B-tree – deletion example (1)

- Delete 82
- Leaf still has N keys ≥ ⌈m/2⌉ - 1



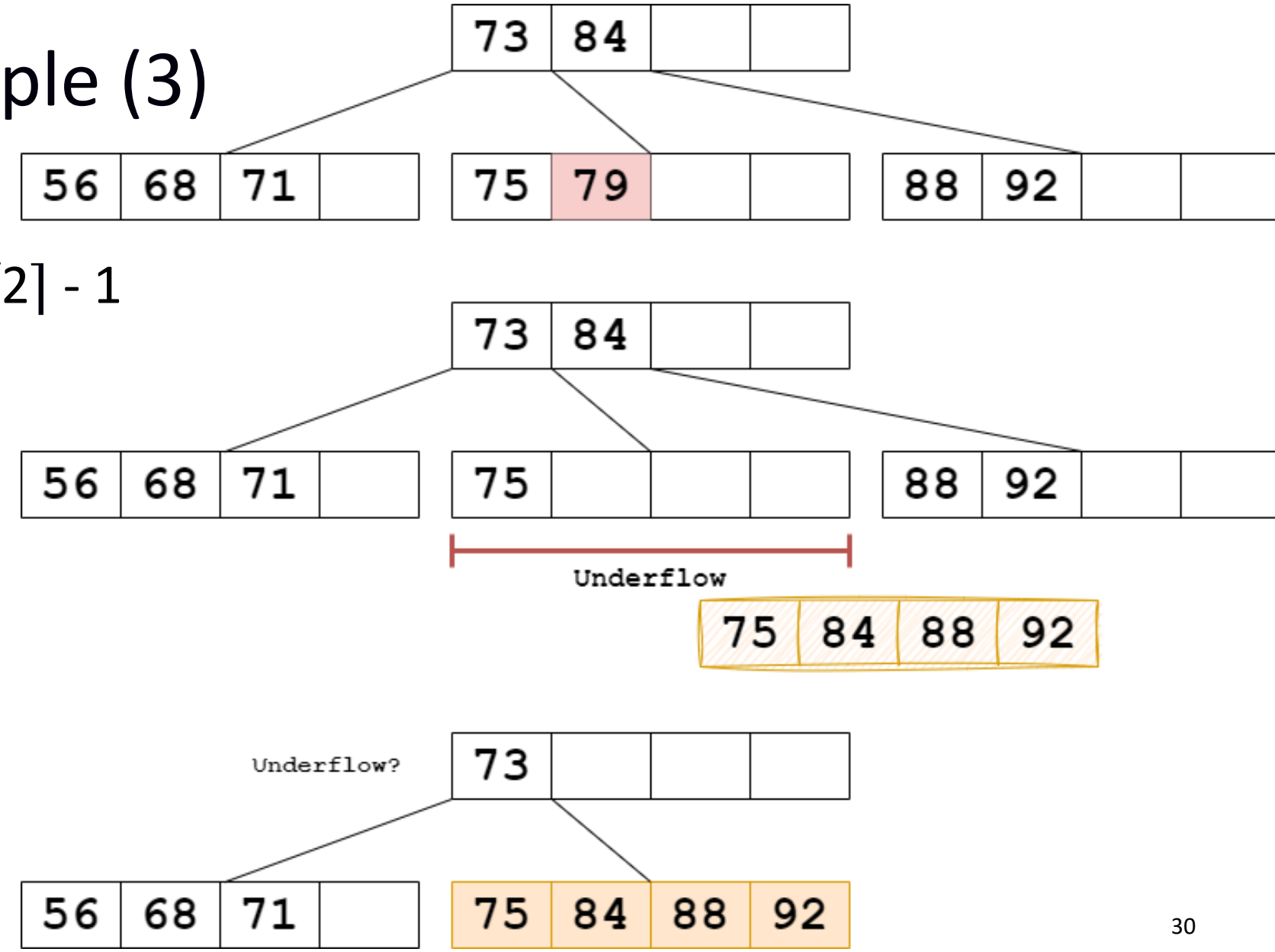
B-tree
– deletion example (2)

- Delete 79
- Leaf has N keys < [m/2] - 1
- SbL obs. case



B-tree
– deletion example (3)

- Delete 79
- Leaf has N keys < ⌈m/2⌉ - 1
- SbR obs. case
  - Not enough keys
- Solve root?



B+-tree?

- All keys are in the leaf nodes
  - They contain pointers to the data
- Parent keys are found in **both** internal and leaf nodes
- Leaf nodes have pointers to the adjacent leaf node
  - Sequential reading

03 – String Based Algorithms

Advanced Algorithms and Data Structures

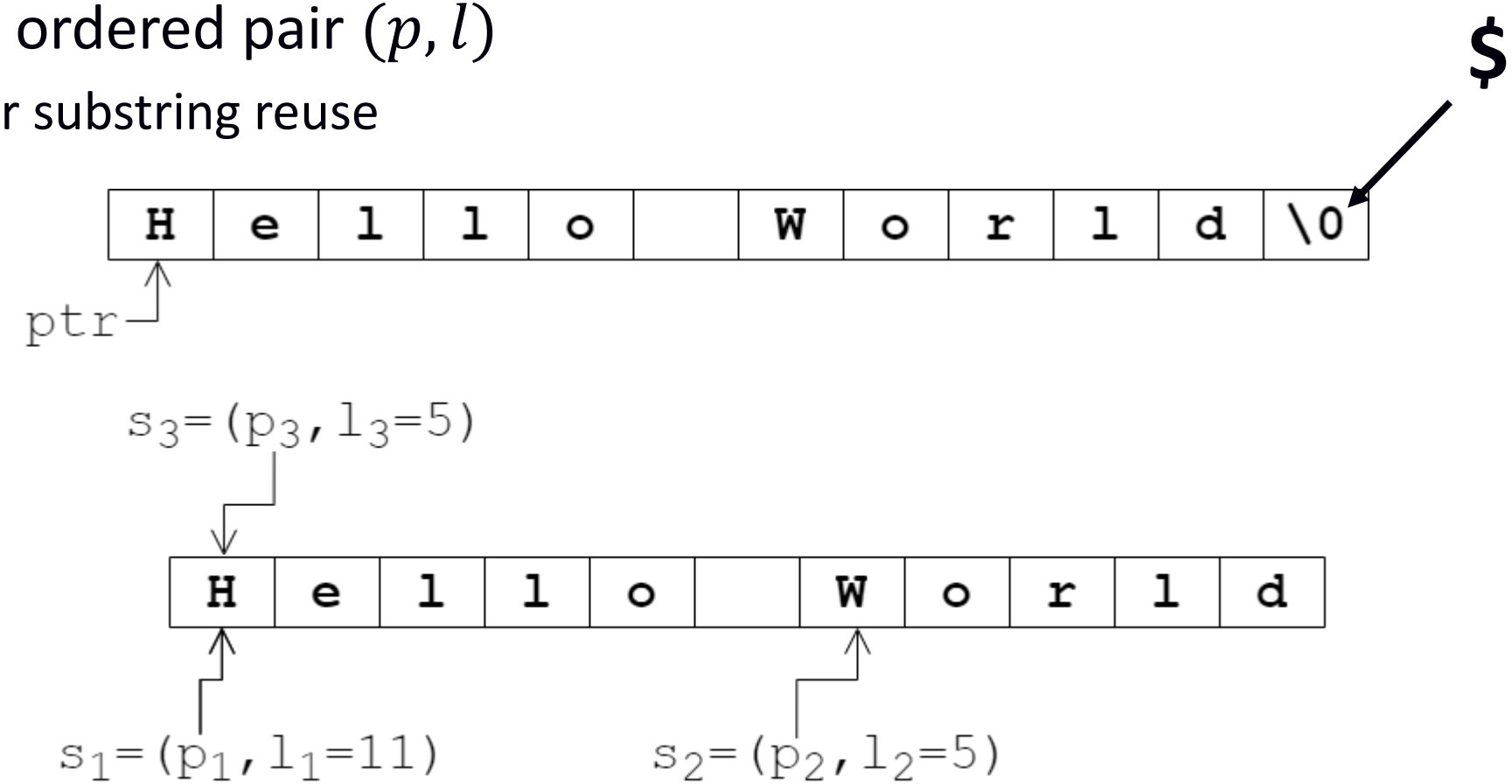
UNIVERSITY OF ZAGREB
Faculty of Electrical Engineering and Computing

Creative Commons

- You are free to:
  - share — multiply, distribute, and publicly communicate the work
  - adapt the work
- under the following conditions:
  - **Attribution:** You must acknowledge and indicate the authorship of the work in the manner specified by the author or license provider (but not in a way that suggests you or your use of the work have their direct support).
  - **Non-commercial:** You may not use this work for commercial purposes.
  - **Share alike:** If you modify, transform, or build upon this work, you may only distribute the modified work under the same or a similar license.

Representation of a string

- With a pointer and a NULL terminator
- With ordered pair (p, l)
  - for substring reuse



Searching a text

- Search for the existence of strings in a text corpus
  - Words in a text

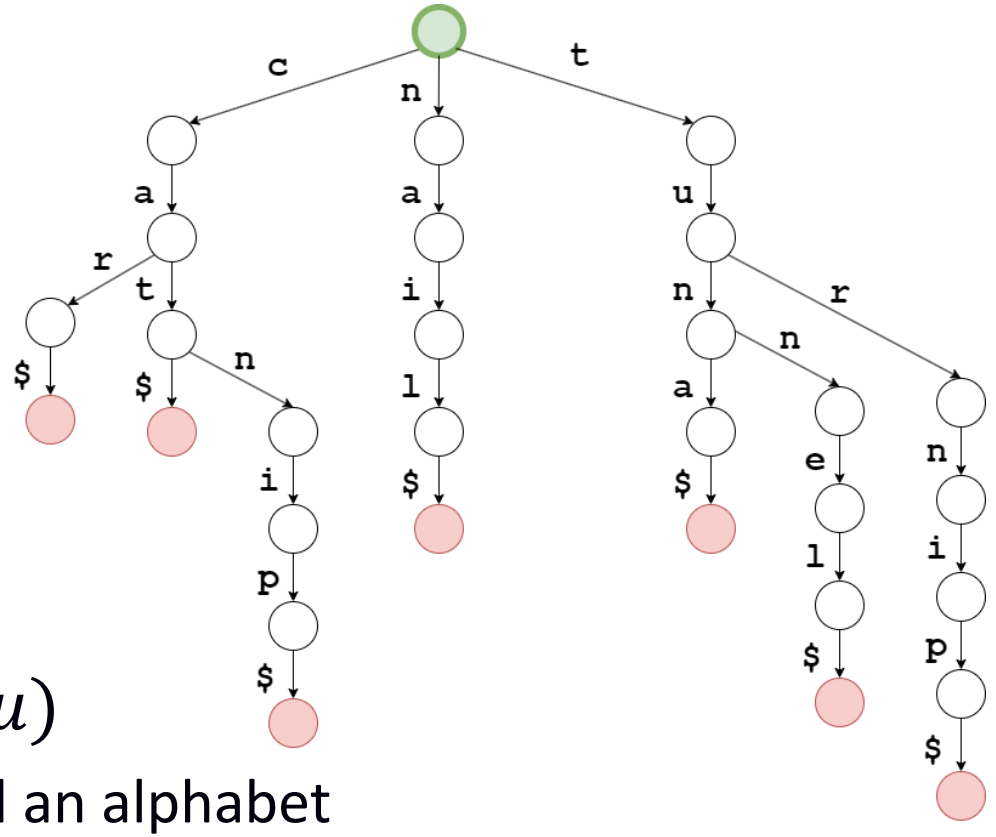
$$S = \{s_i : 0 < i \leq N\}$$

- Naively – search for q in S

$$O(\sum_{s_i \in S} |s_i|)$$

- Each symbol in each string

Prefix tree – Trie (1)



- Reads as *try*
- M-tree
- Ordered triple

$$T = (N, E, \mu)$$

With a mapping function between edges and an alphabet

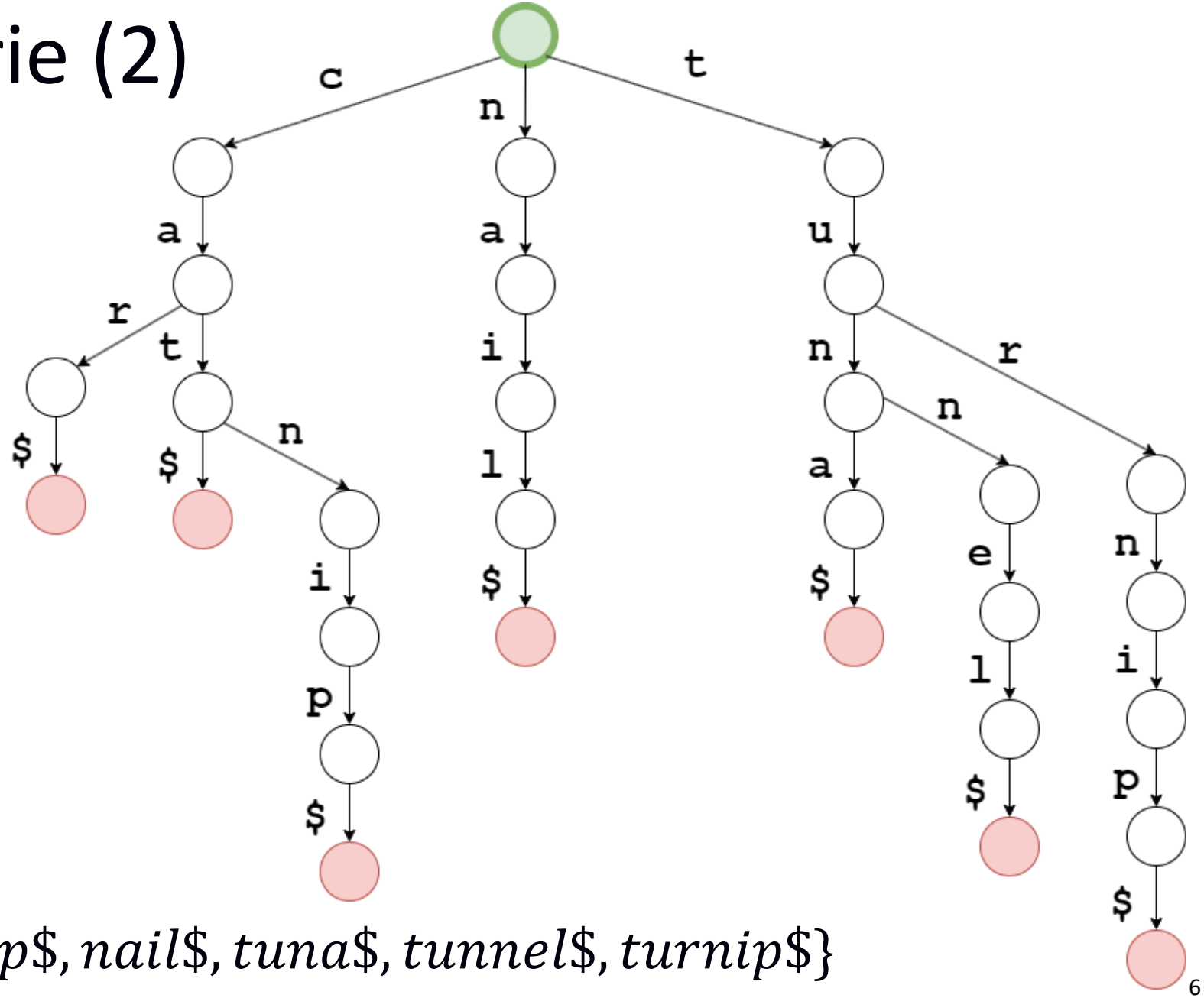
$$\mu: E \to \Sigma$$

- Each node can have at most $|\Sigma|$ Ch

Note: Deterministic finite automata from Introduction to Theoretical Computer Science
*with a few caveats

Prefix tree – Trie (2)

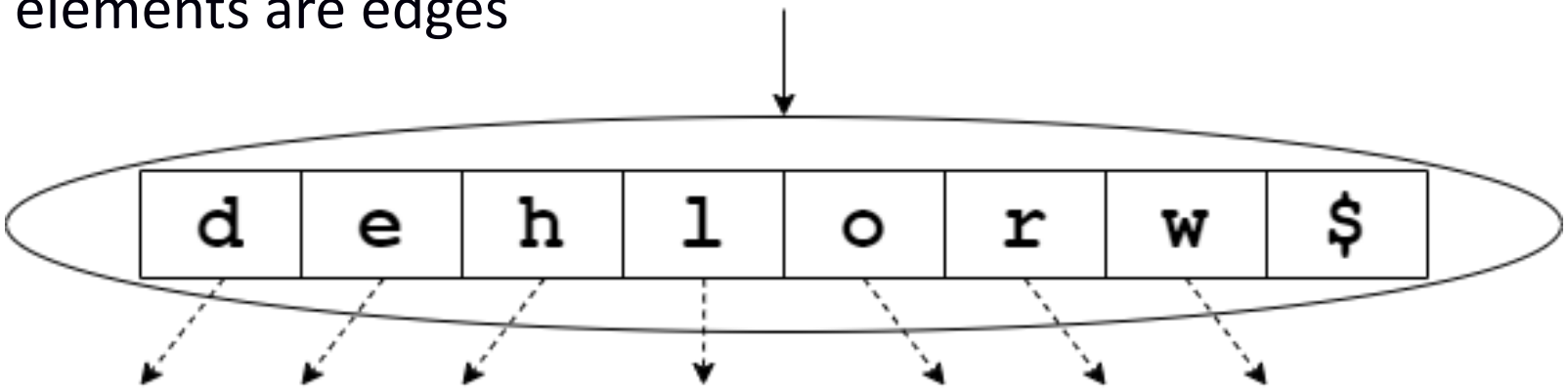
- Snail?
- Reuse -nip?



$S = \{car\$, cat\$, catnip\$, nail\$, tuna\$, tunnel\$, turnip\$\}$

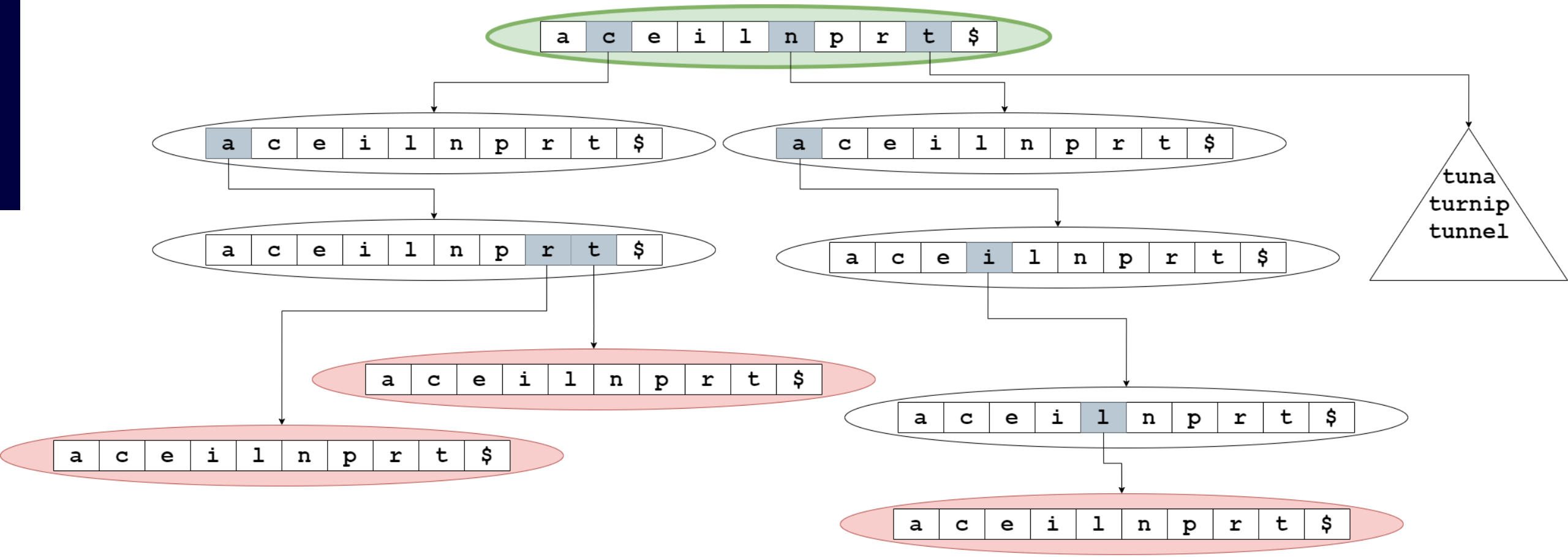
Trie – node structure

- Array of sorted alphabet chars and a pointer for each
- Pointers lead to next char
  - Array elements are edges



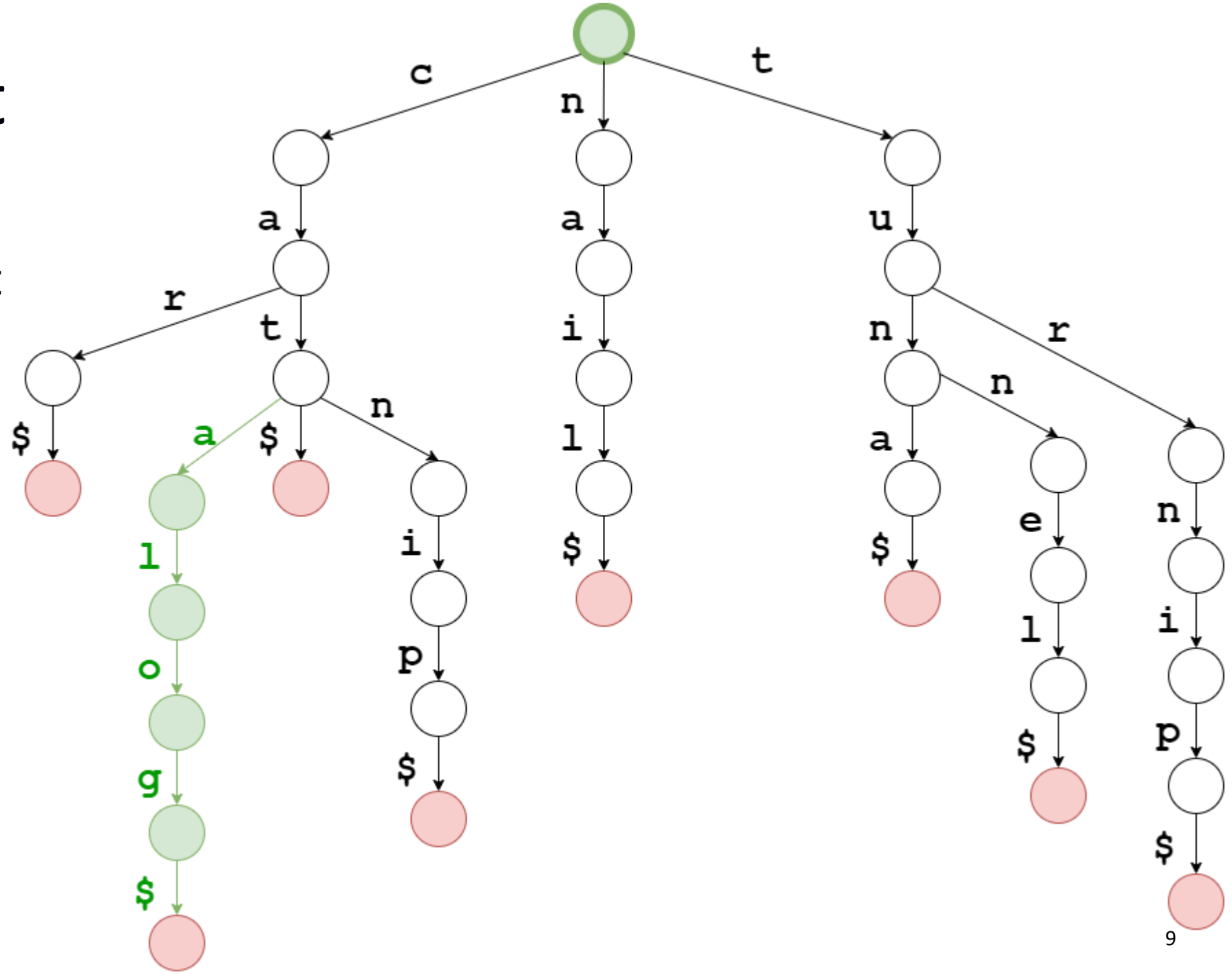
e.g. Hello World

Trie – structured example



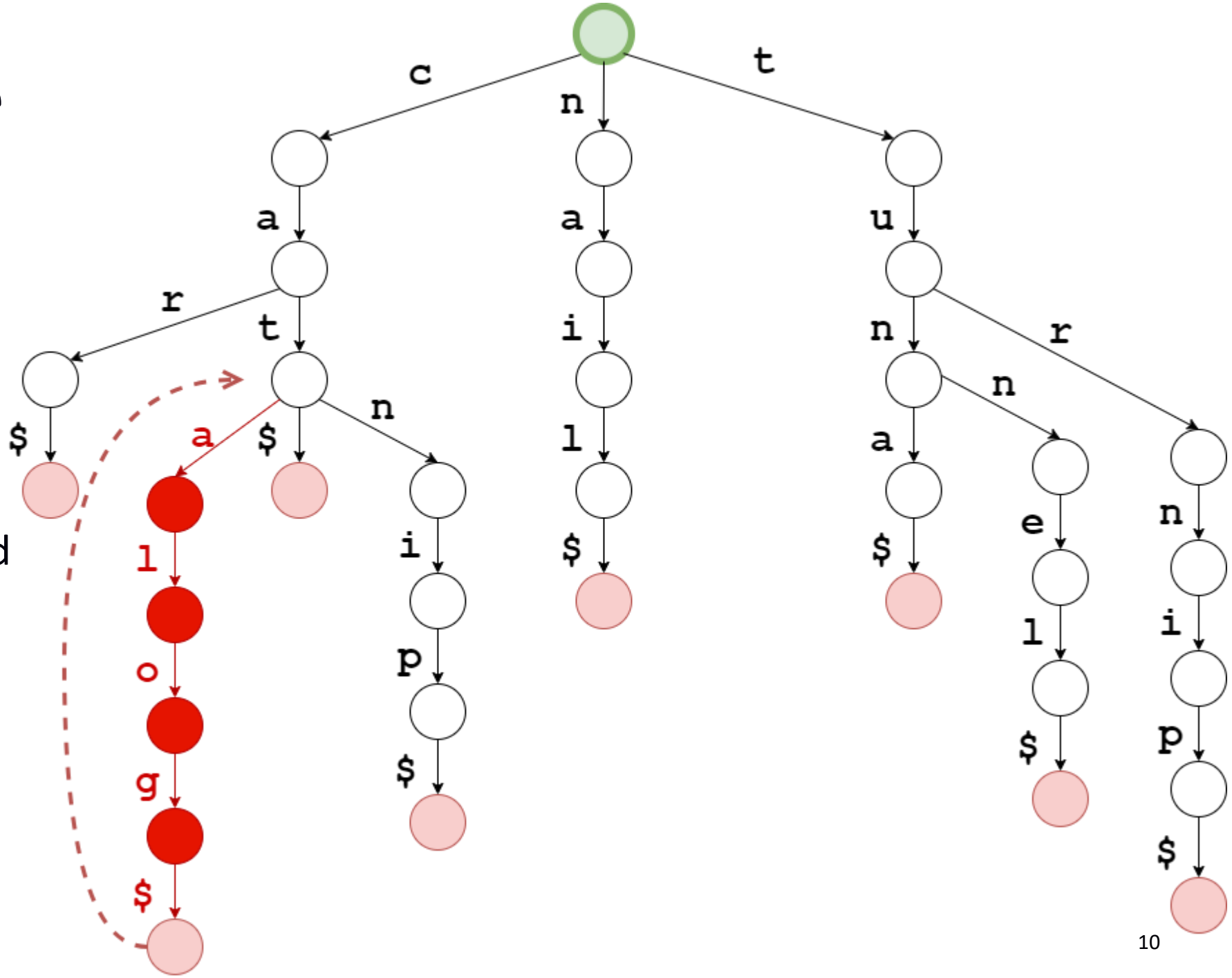
Trie – insert

- Update alphabet
- Search until no transition
- Add children for suffix
- e.g. catalog



Trie – delete

- Search until no transition
- Remove backwards
  - While single child
- e.g. catalog

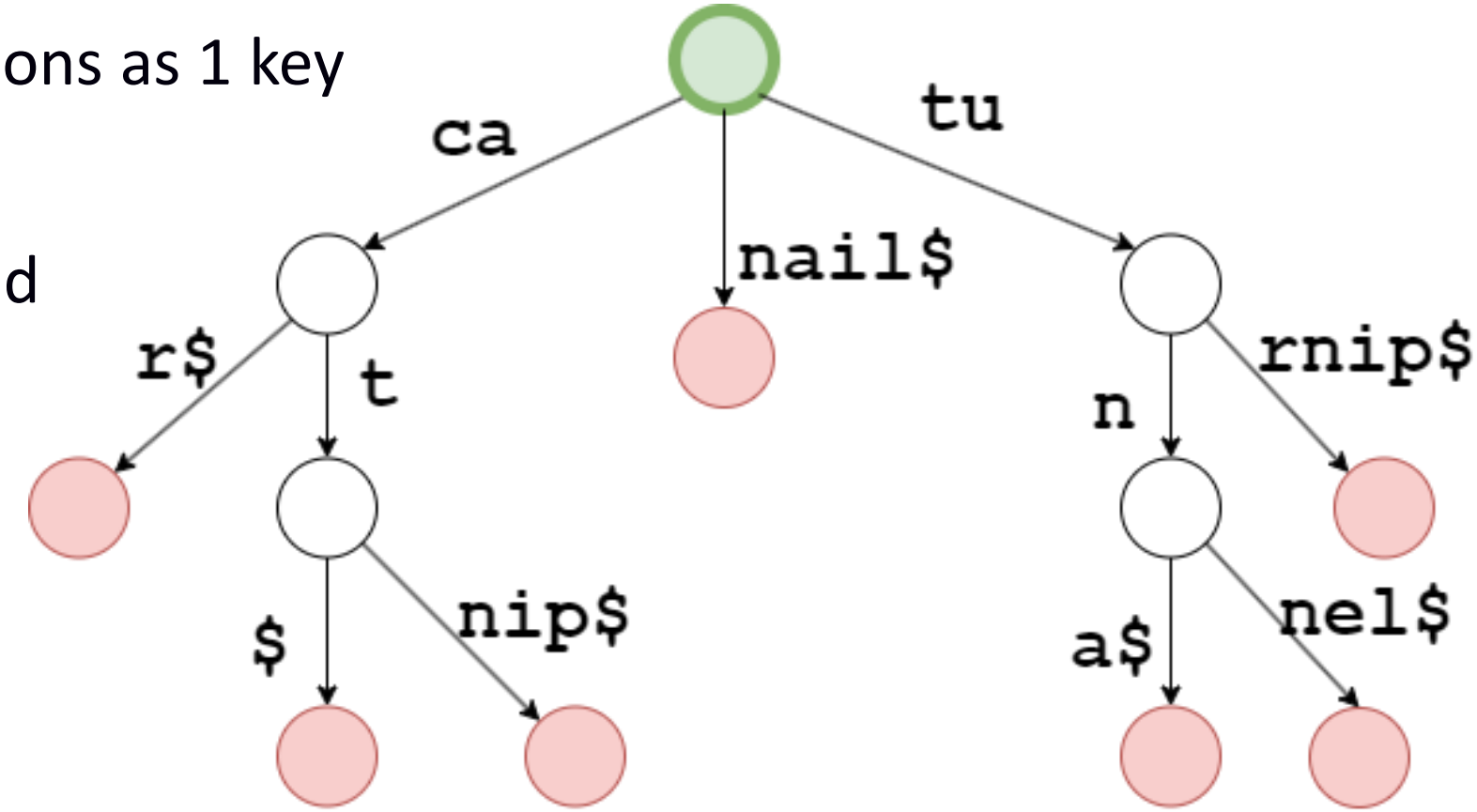


Trie – complexity

- Space: $O\left(\sum_{s_i \in S} |s_i| * |\Sigma|\right)$
  - All letters for each word
- Search time: $O(|q|)$
  - Length of the query string
- A tradeoff for including the array
- Insert: $O(|q|)$
  - Find prefix and complete for suffix
- Delete: $O(|q|)$
  - Find terminal for the word and rollup the unique branch
    - Only child or one of…

Patricia tree (trie?)

- A compressed Trie
- A sequence of transitions as 1 key
- Less memory is utilized



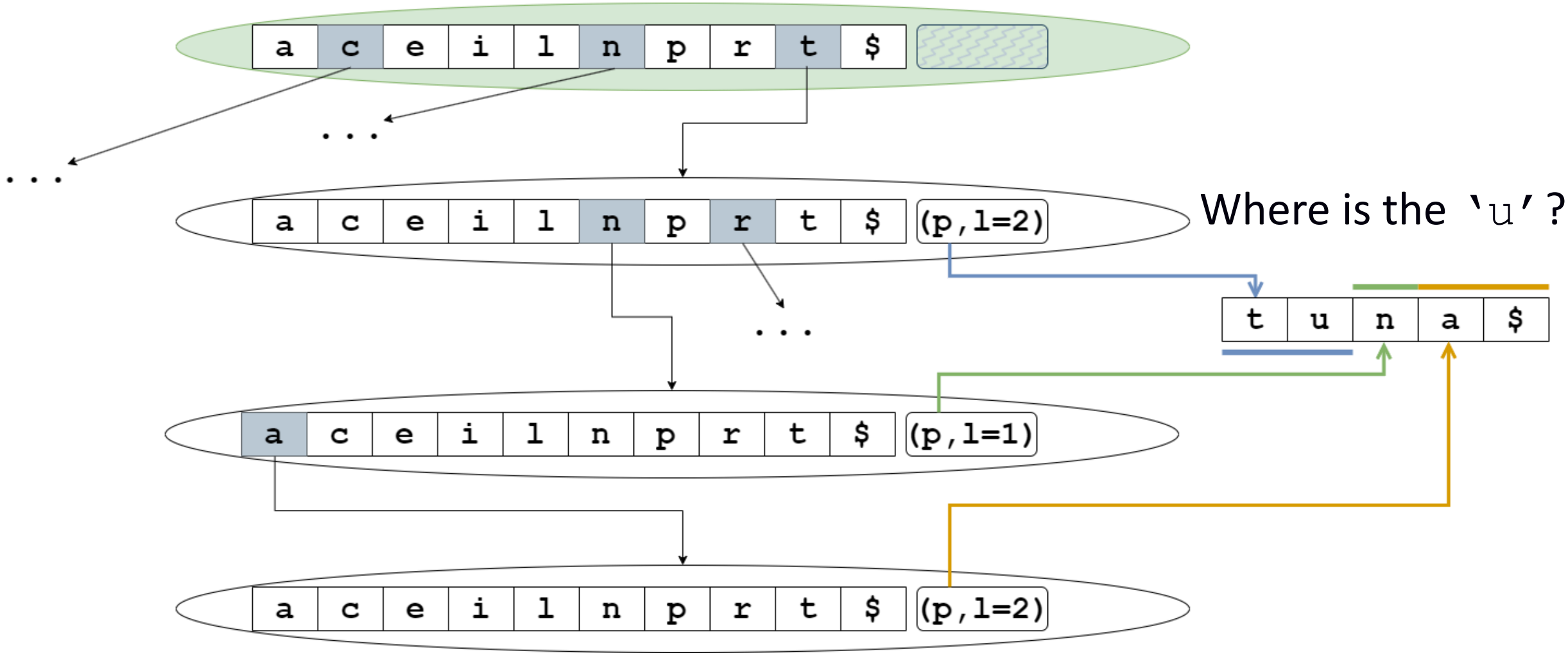
*Practical Algorithm To Retrieve Information Coded In Alphanumeric

Patricia tree – node structure (1)

- The nodes are references that contain:
  - (p, l) to an instance of a string
    - First created
  - Pointers to the Ch substrings
- Internal nodes have at least 2 Ch

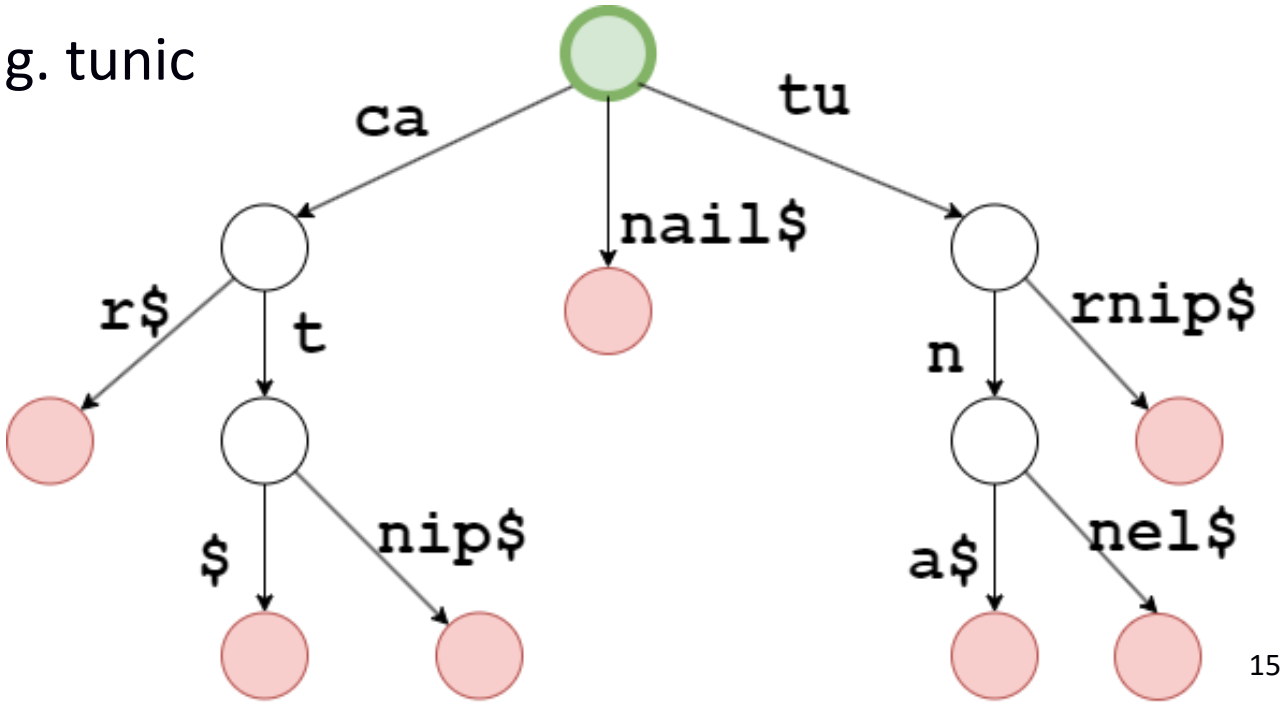
Note: „references” as in index nodes of a database

Patricia tree – node structure (2)



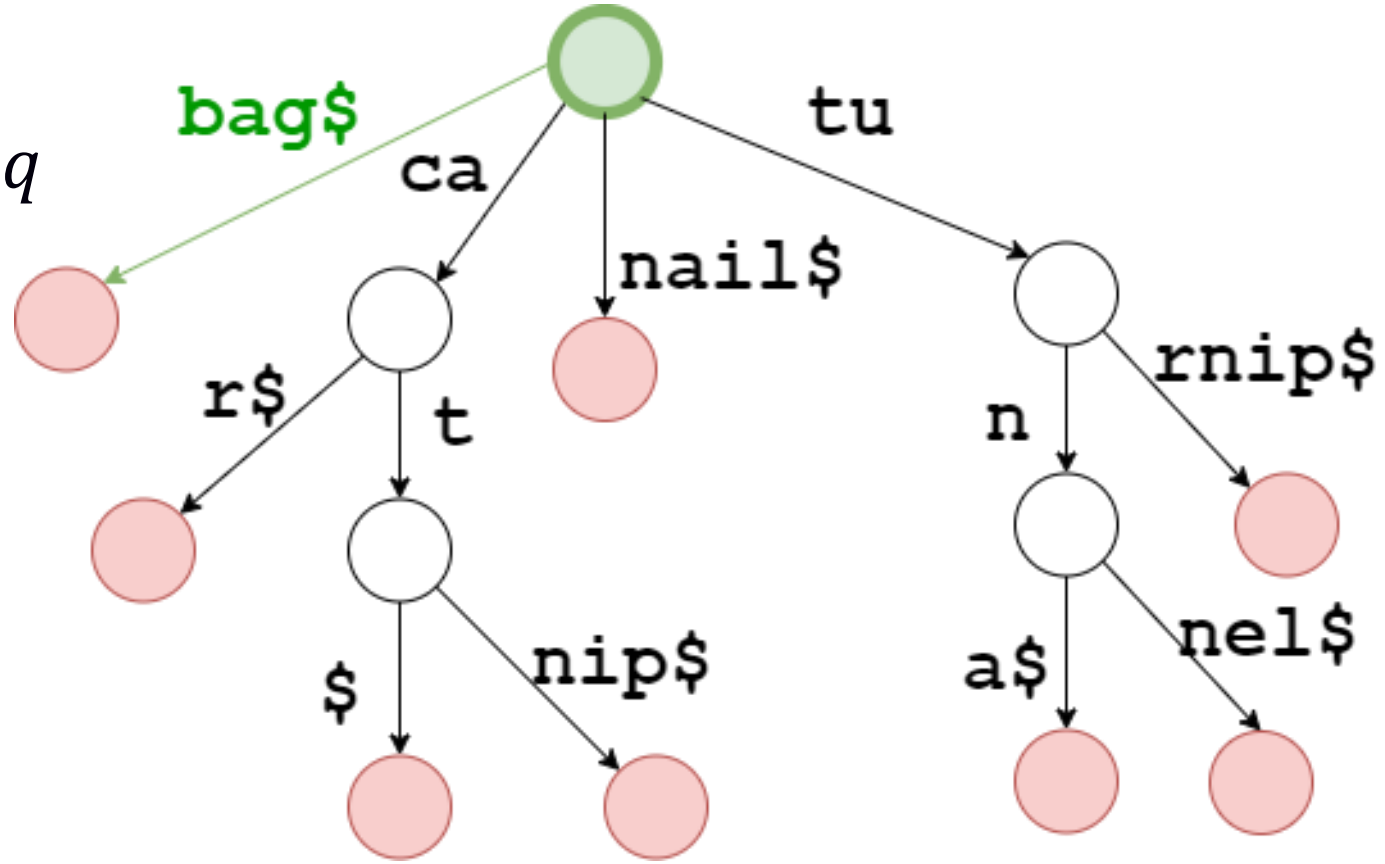
Patricia tree – insert (1)

- Starts with search
- 3 scenarios:
  1. No transition found – e.g. bag
  2. Partial transition found – e.g. name
  3. Complete transition found – e.g. tunic
- Adds string to memory
- Complexity: $O(|s| + |\Sigma|)$
  - New word after entire alphabet is matched



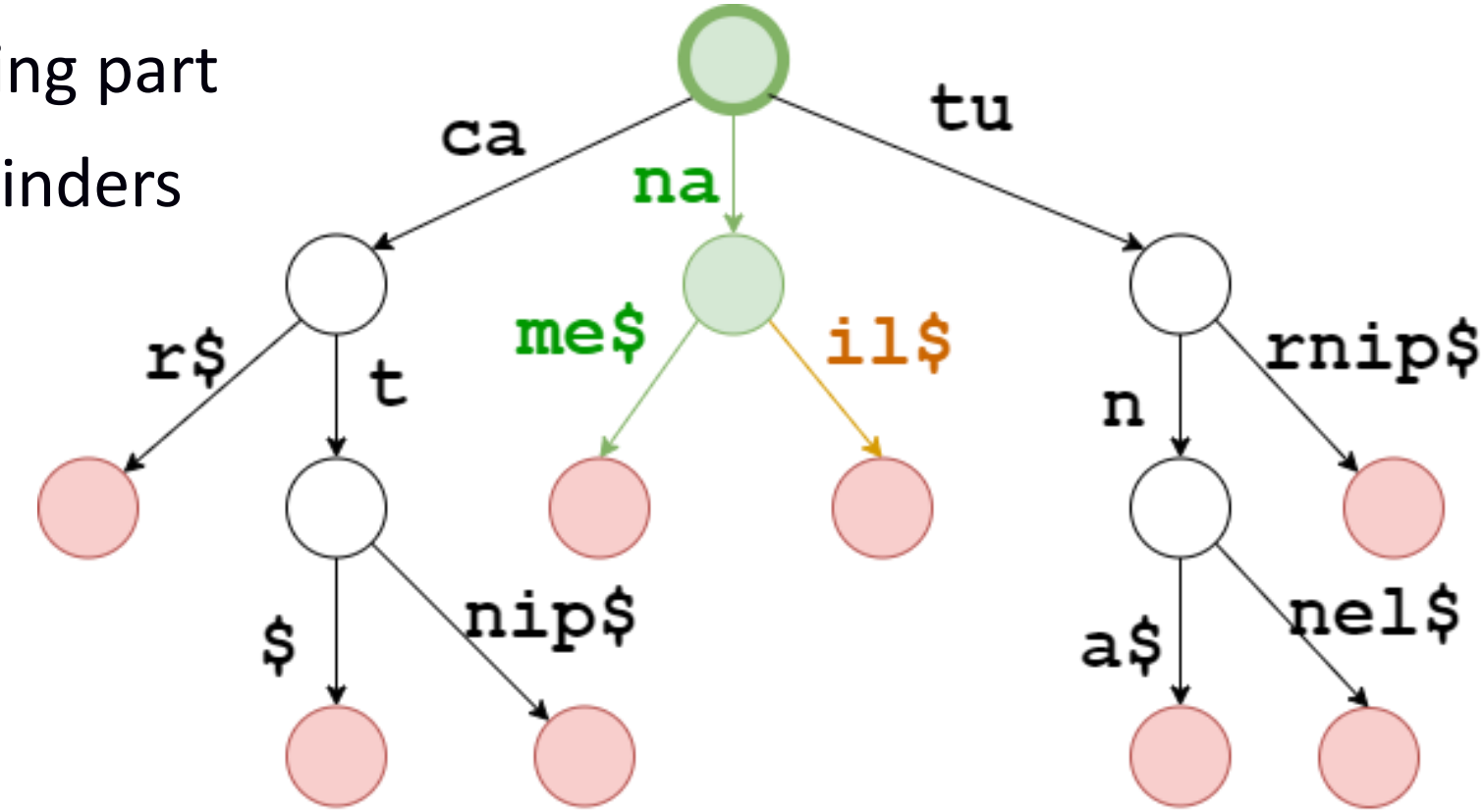
Patricia tree – insert (2)

- No transition
- Create new node with entire q
- “bag”



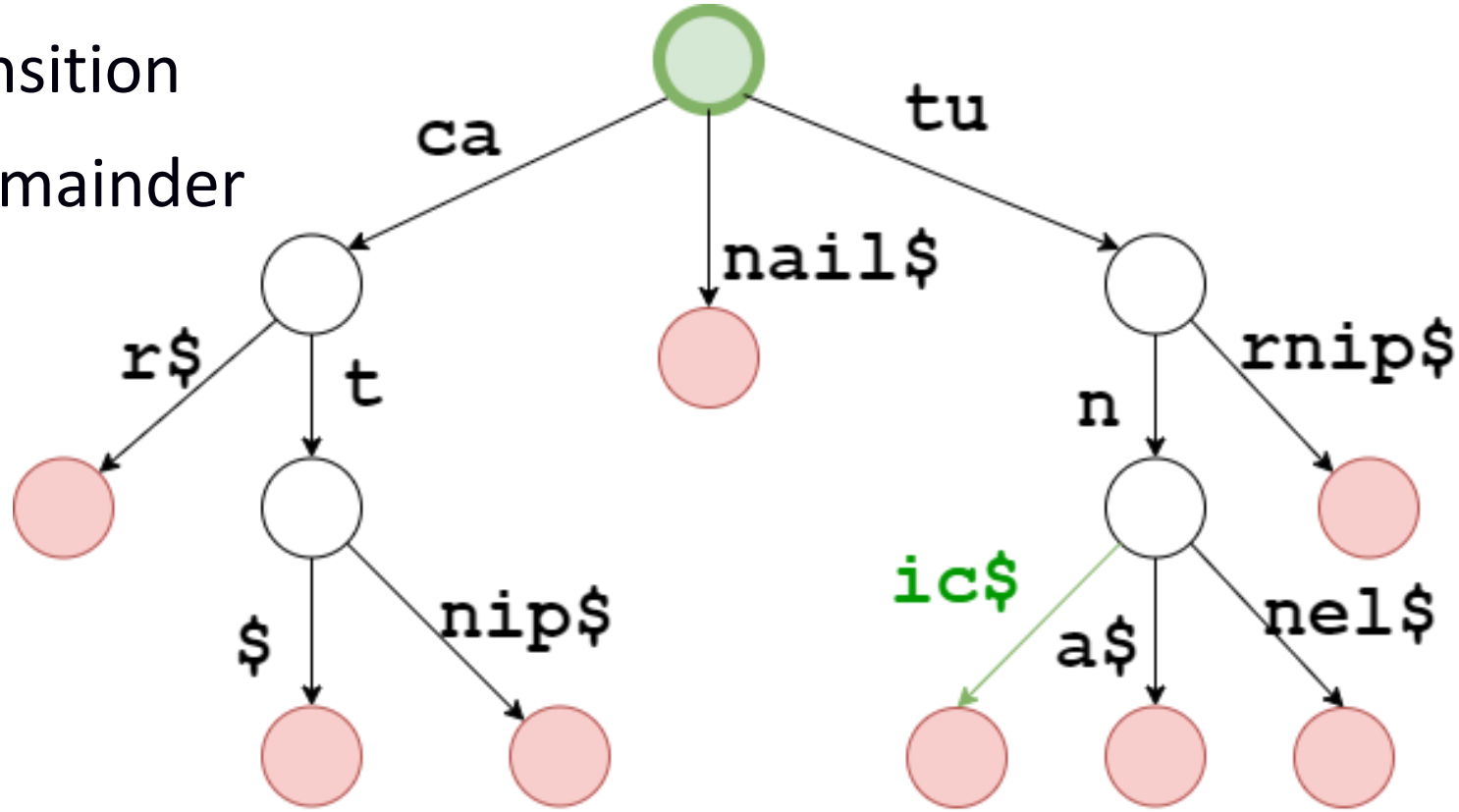
Patricia tree – insert (3)

- Partial transition
- Create Pr with the matching part
- Create Chs with the remainders
- “name”



Patricia tree – insert (4)

- Full transition
- Find the last matching transition
- Create new Ch with the remainder
- “tunic”

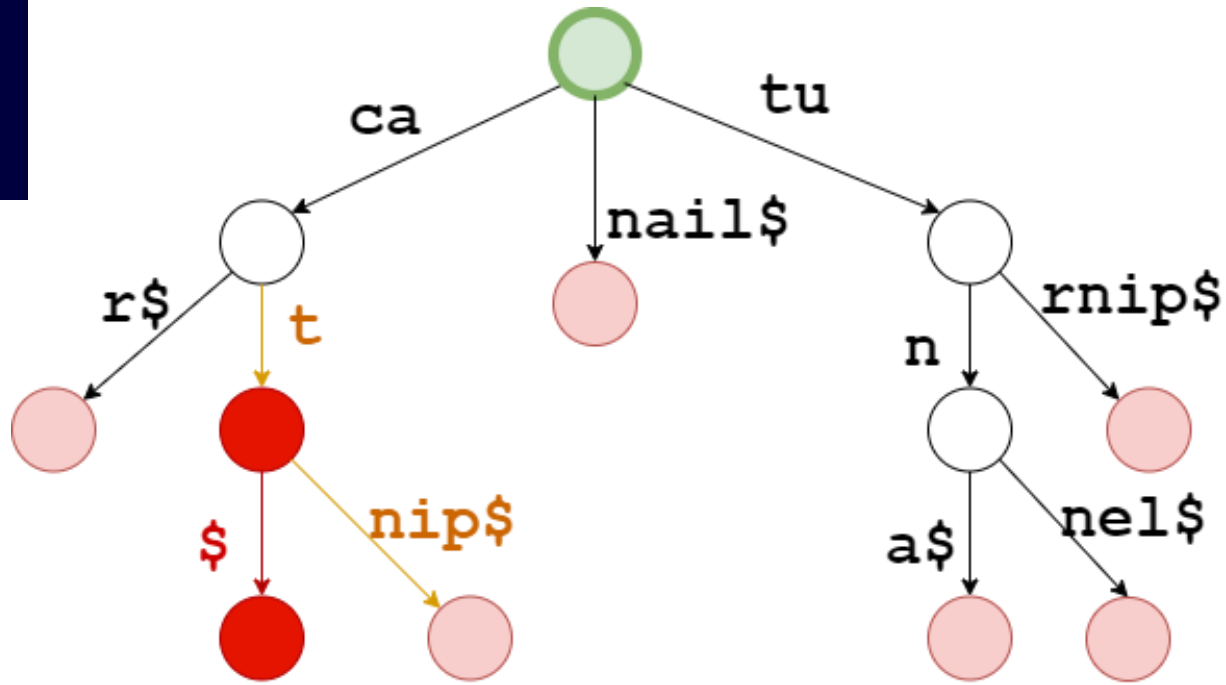


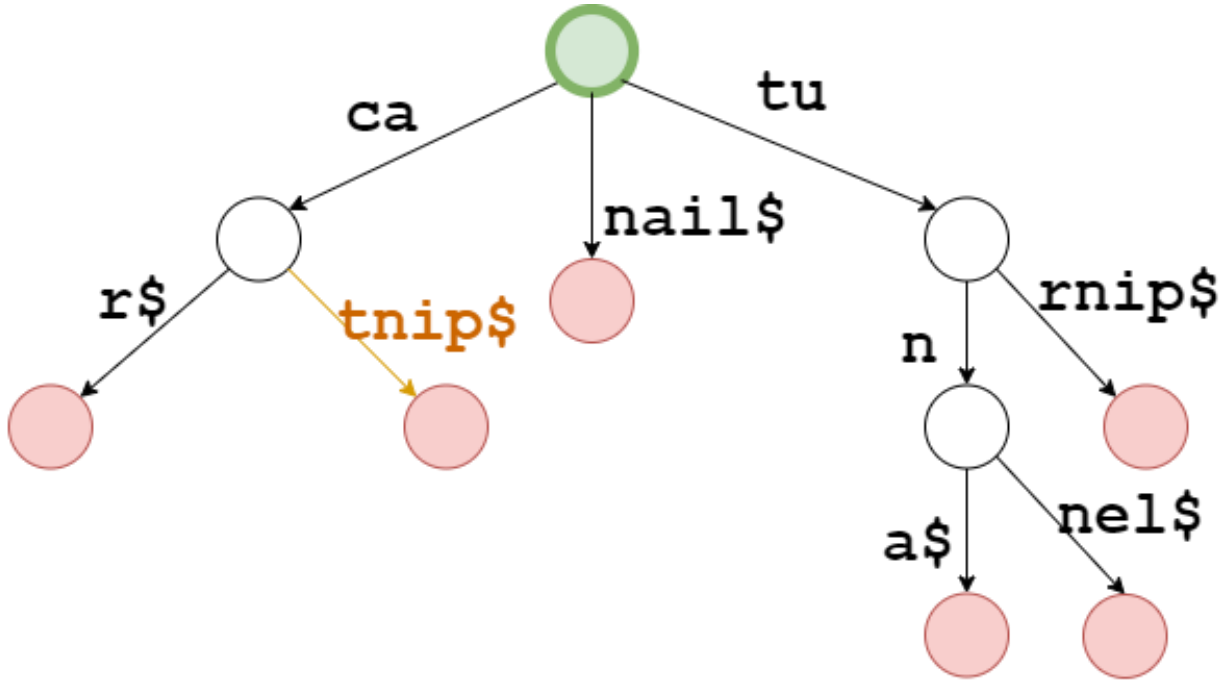
Patricia tree – delete (1)

- Search
- Remove leaf nodes that match
- Compress internal nodes that have one child after deletion
- Remove string from memory

Patricia tree – delete (2)

- Delete “cat”





Prefix trees in general

- Use in:
  - Associative arrays – e.g. dictionaries
    - Haskell
  - Sorting
  - IP routing
  - Inverted indexes
  - Proofs about inductive data structures or defining recursive functions
    - Coq
  - Symbol tables
    - Lexing

Suffix arrays
Suffix trees

Array suffix

- All suffixes of an array
- e.g. program$:

  program$
  rogram$
  ogram$
  gram$
  ram$
  am$
  m$
  $

- Best way to store it?
- Naively, a lot of copies
- (p, l) -> |s| ordered pairs

Suffix array (1)

- A sorted array of suffixes

i	A_i	t_{A_i}							
1	8	$							
2	6	a	m	$					
3	4	g	r	a	m	$			
4	7	m	$						
5	3	o	g	r	a	m	$		
6	1	p	r	o	g	r	a	m	$
7	5	r	a	m	$				
8	2	r	o	g	r	a	m	$	

- Creation is costly: $O(n^2 \log n)$
  - Sort $O(n \log n)$ and char comparison $O(n)$ in sorting

Suffix array (2)

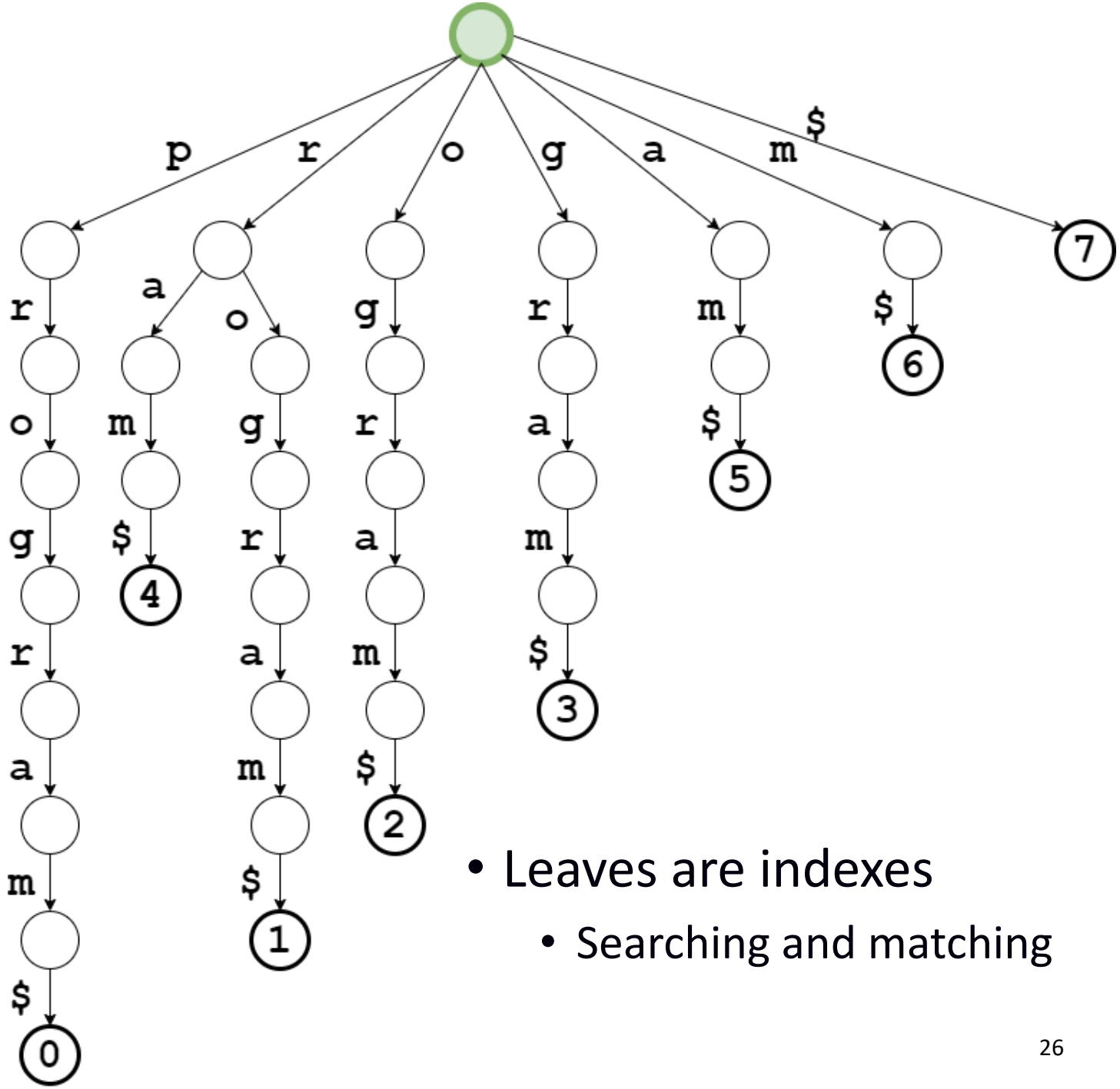
- Goal: Check for the existence of a q in a in a string t
- Search naively $O(|q| * n)$
- Search with binary alg. $O(|q| \log n)$

**Enhancements – AADS script*

Suffix trie (1)

- What if we use the Trie to index the suffix array?

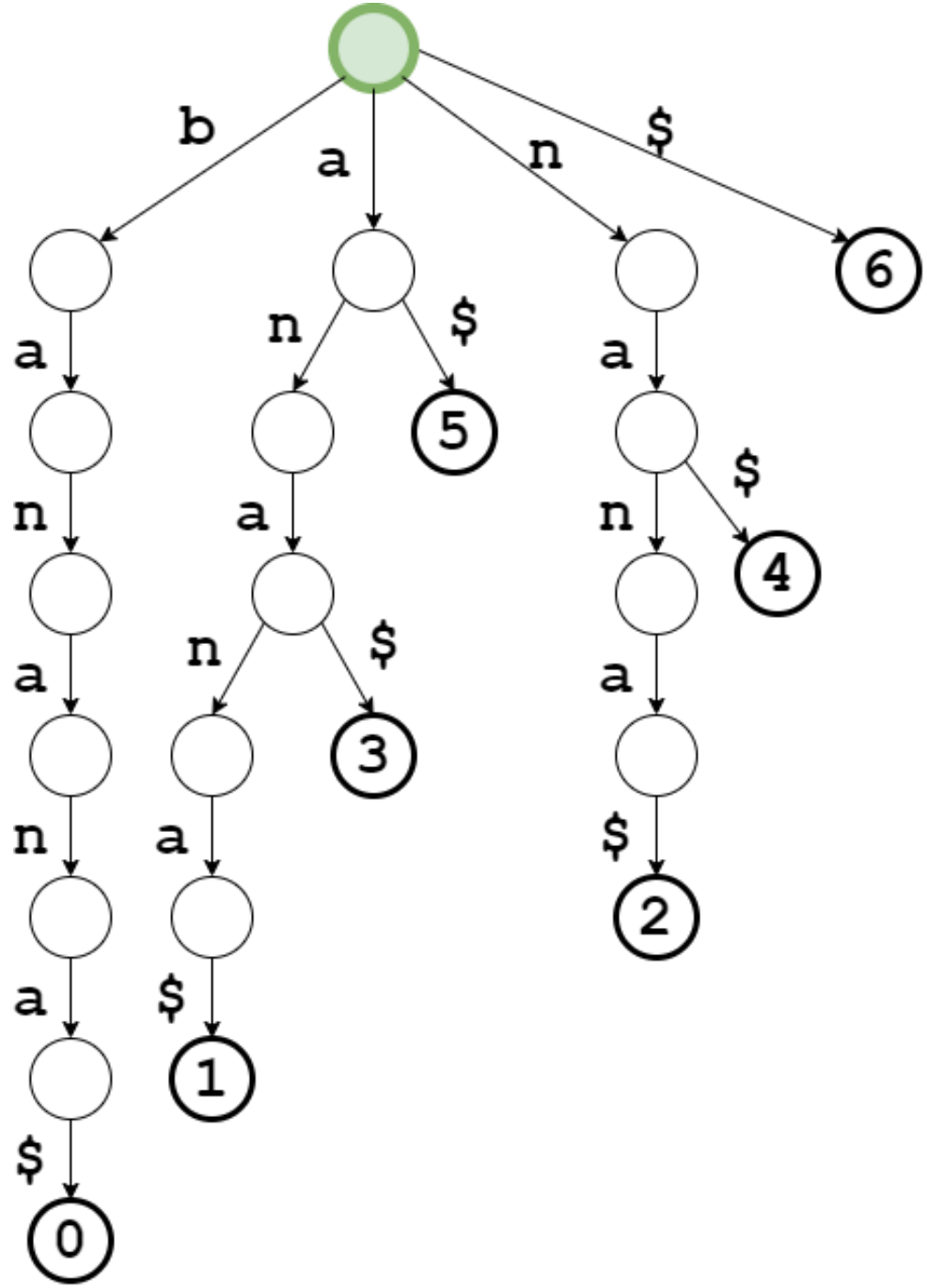
p	r	o	g	r	a	m	$
r	o	g	r	a	m	$	
o	g	r	a	m	$		
g	r	a	m	$			
r	a	m	$				
a	m	$					
m	$						
$							



- Leaves are indexes
  - Searching and matching

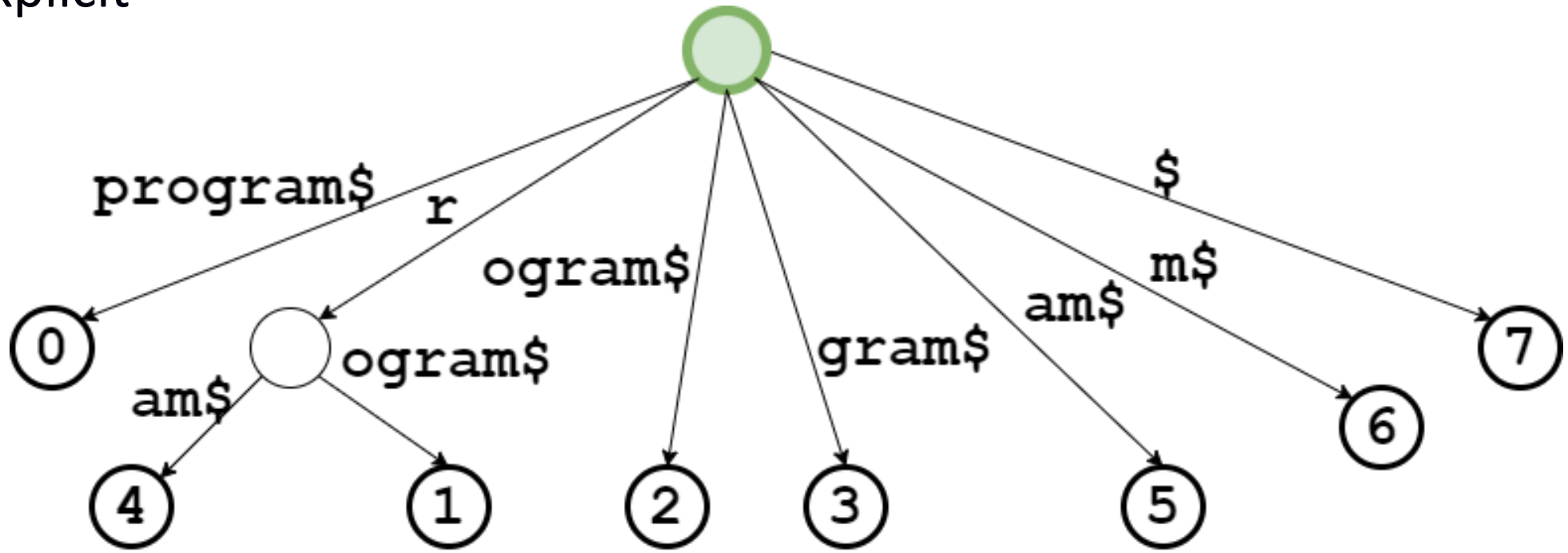
Suffix trie (2)

- A more repetitive example



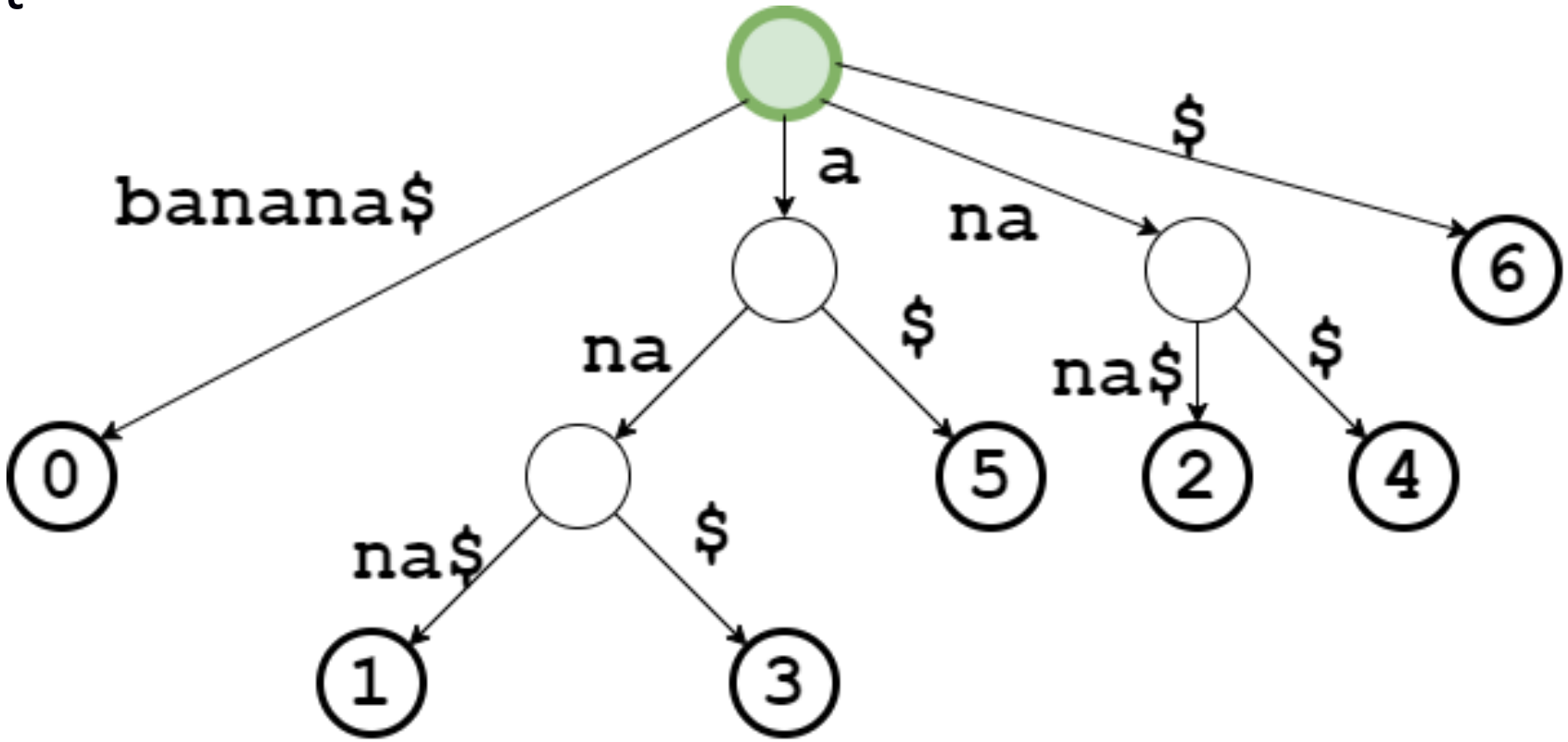
Compressed suffix trie – Patricia tree (1)

- Explicit



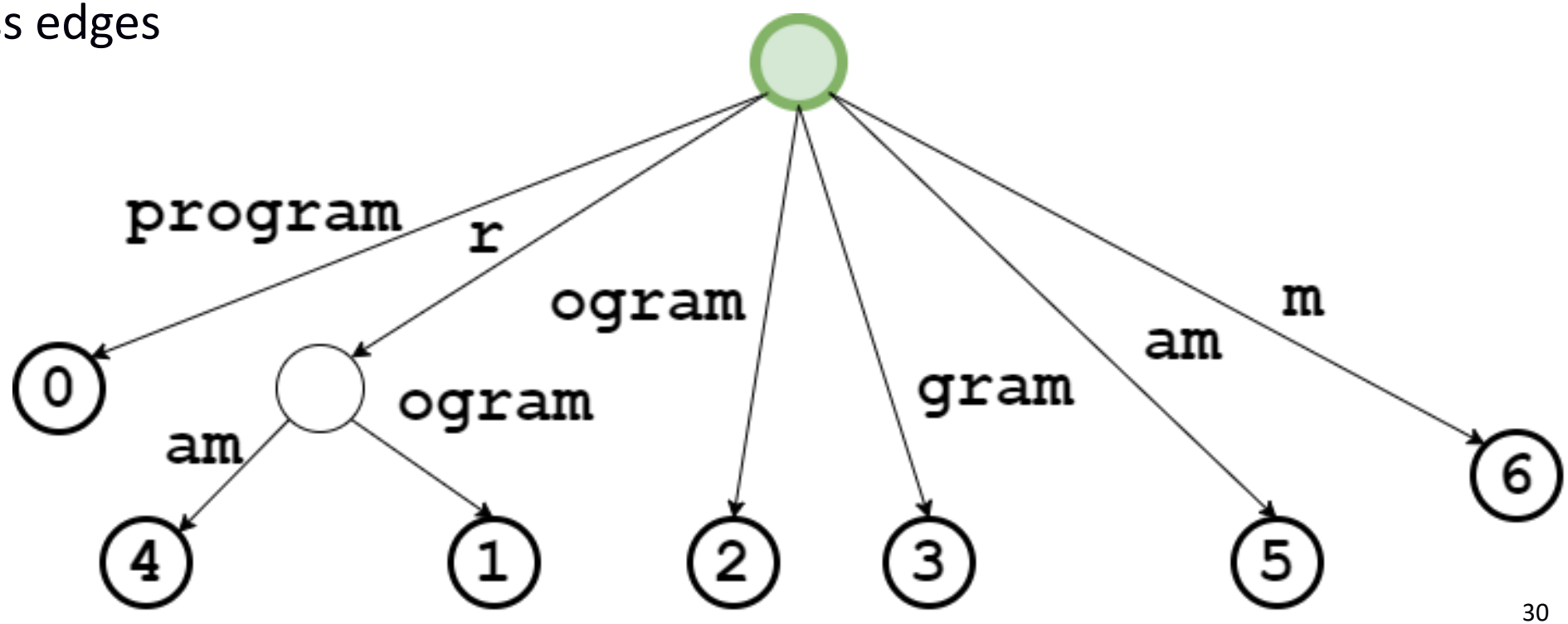
Compressed suffix trie – Patricia tree (2)

- Explicit



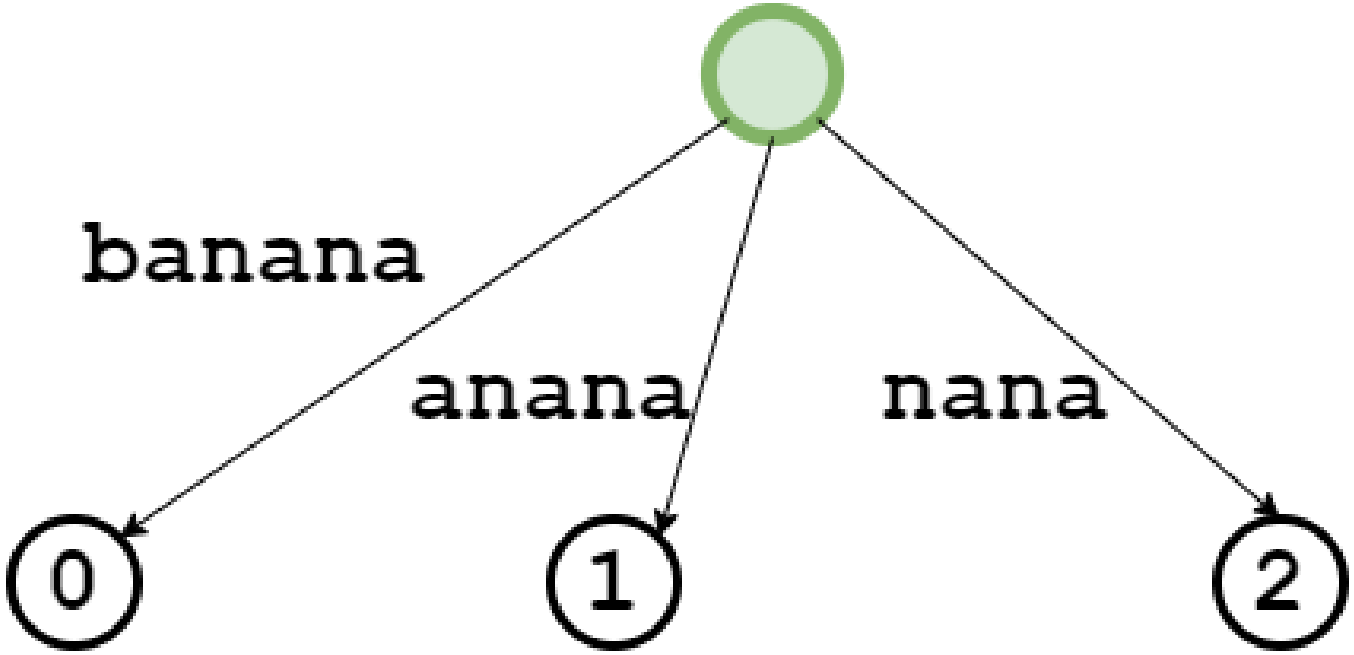
Compressed suffix trie – Patricia tree (3)

- Implicit suffix tree:
  1. Remove all terminators from edges
  2. Remove all empty edges; including subtrees
  3. Compress edges



Compressed suffix trie – Patricia tree (4)

- Implicit suffix tree:
  1. Remove all terminators from edges
  2. Remove all empty edges; including subtrees
  3. Compress edges



Ukkonen's algorithm (1)

- Online creation of implicit compressed suffix tries
- Creation: $O(|S|)$

- Contains steps and substeps
  - Steps analyze each prefix ($i \in [0, |S| - 1]$)
  - Substeps analyze each suffix of a prefix ($j \in [0, i + 1]$)
- Each step we add $\mathrm{S}[i + 1]$
  - There is a step $i = -1$ where we just add $S[0]$ to the tree
- Each substep we use $\beta = S[j..i]$ to traverse the existing tree

Ukkonen’s algorithm (2)

- 3 cases when adding:

  C1: β traverses to a leaf

  -> Add S[i+1] to the leaf edge

  C2: β stopped mid-traversal AND the next char is not S[i+1]

  -> Split at stop (create splitting node)

  -> Add leaf edge with S[i+1]

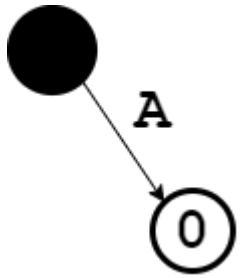
  -> Add current j as leaf

  C3: β stopped mid-traversal AND the next char is S[i+1]

  -> Do nothing

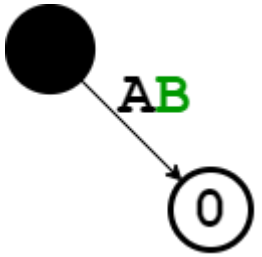
Ukkonen’s algorithm – example (1)

Step -1	S=ABABABC$	



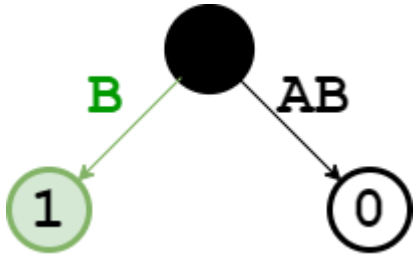
Ukkonen’s algorithm – example (2)

Step i=0	S=ABABABC$	S[i+1]=B
j=0	β=S[j..i]=S[0..0]=A	Case: C1



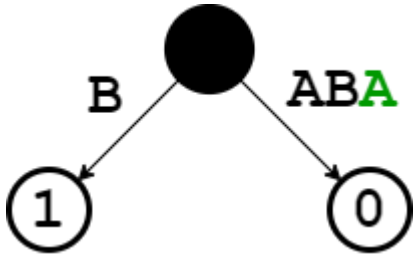
Ukkonen’s algorithm – example (3)

Step i=0	S=ABABABC$	S[i+1]=B
j=1	β=S[j..i]=S[1..0]=ε	Case: C2



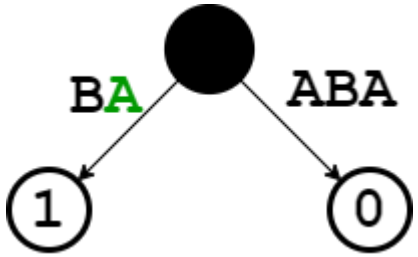
Ukkonen’s algorithm – example (4)

Step i=1	S=ABABABC$	S[i+1]=A
j=0	β=S[j..i]=S[0..1]=AB	Case: C1



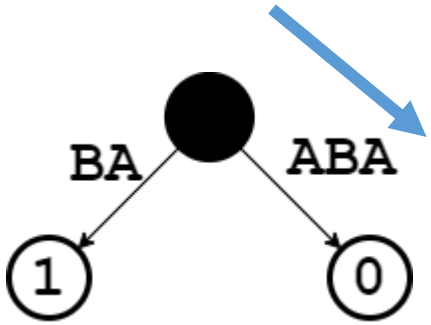
Ukkonen’s algorithm – example (5)

Step i=1	S=ABABABC$	S[i+1]=A
j=1	β=S[j..i]=S[1..1]=B	Case: C1



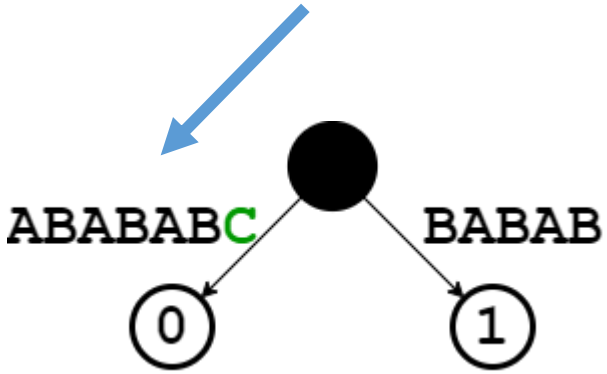
Ukkonen’s algorithm – example (6)

Step i=1	S=ABABABC$	S[i+1]=A
j=2	β=S[j..i]=S[2..1]=ε	Case: C3



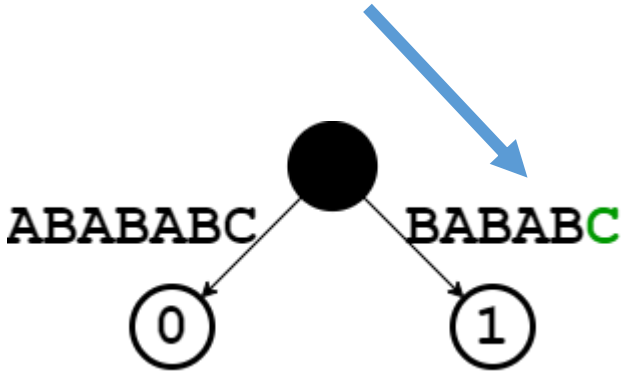
Ukkonen’s algorithm – example (7)

Step i=5	S=ABABABC$	S[i+1]=C
j=0	β=S[j..i]=S[0..5]=ABABAB	Case: C1



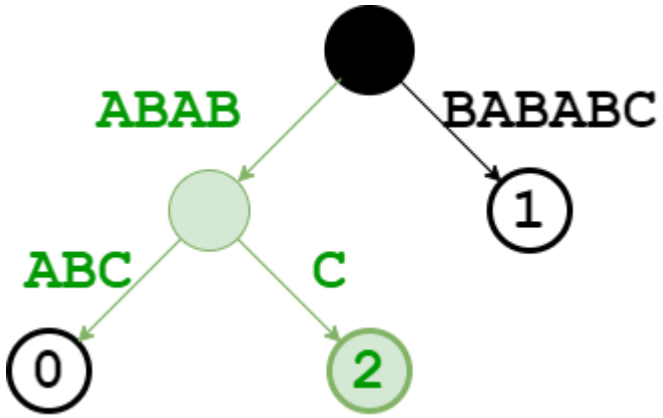
Ukkonen’s algorithm – example (8)

Step i=5	S=ABABABC$	S[i+1]=C
j=1	β=S[j..i]=S[1..5]=BABAB	Case: C1



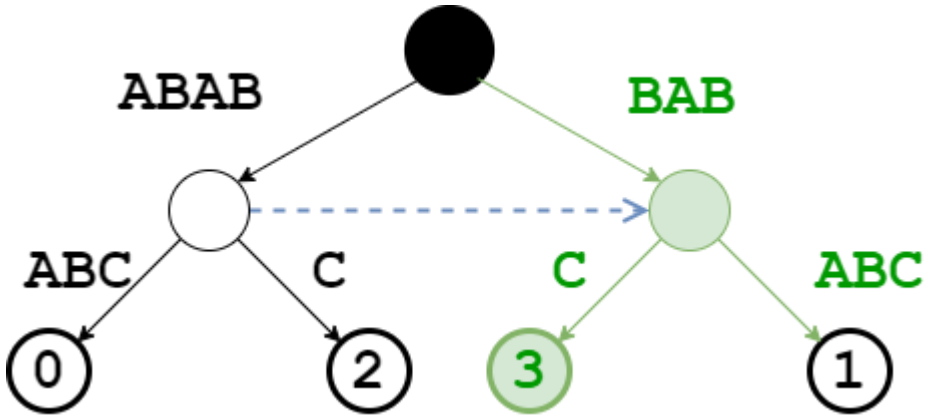
Ukkonen’s algorithm – example (9)

Step i=5	S=ABABABC$	S[i+1]=C
j=2	β=S[j..i]=S[2..5]=ABAB	Case: C2



Ukkonen’s algorithm – example (10)

Step i=5	S=ABABABC$	S[i+1]=C
j=3	β=S[j..i]=S[3..5]=BAB	Case: C2

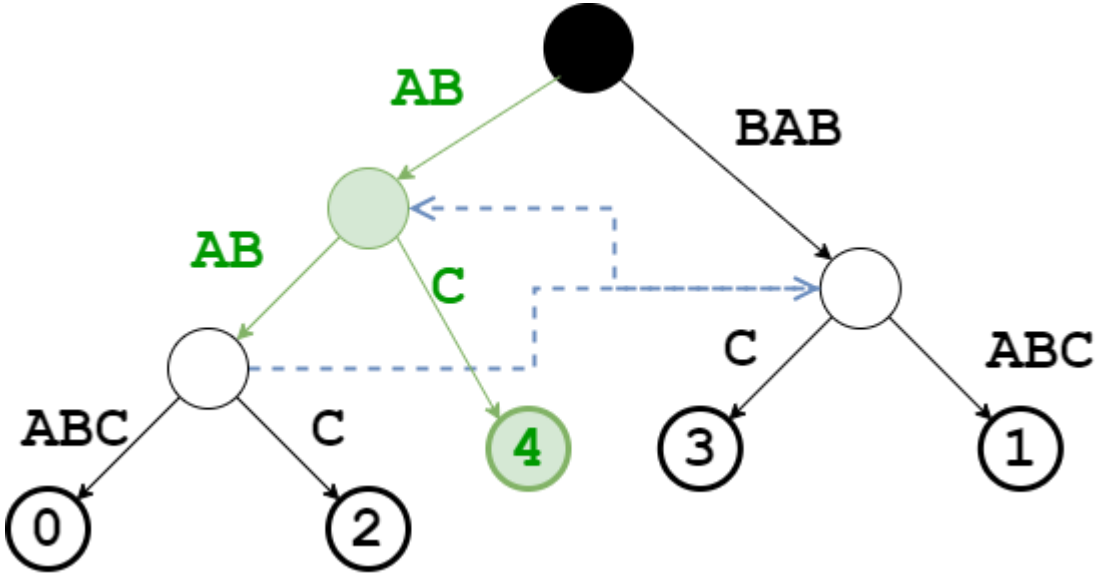


Suffix links are added only on C2 when an internal node is added

-> Link toward and from the created node with Xα -> α

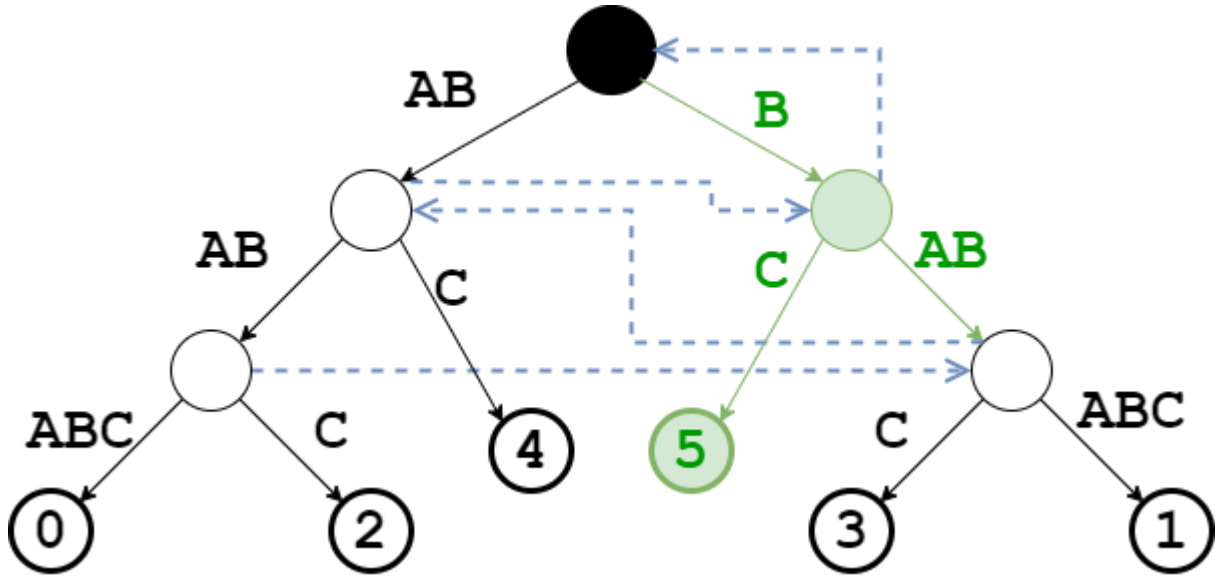
Ukkonen’s algorithm – example (11)

Step i=5	S=ABABABC$	S[i+1]=C
j=4	β=S[j..i]=S[4..5]=AB	Case: C2



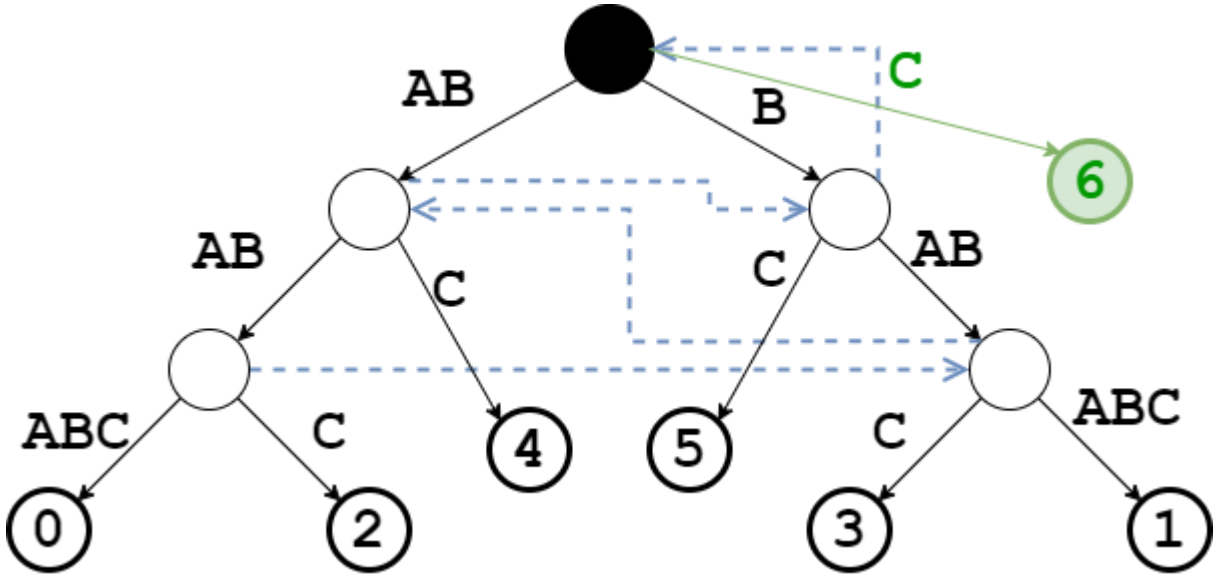
Ukkonen’s algorithm – example (12)

Step i=5	S=ABABABC$	S[i+1]=C
j=5	β=S[j..i]=S[5..5]=B	Case: C2



Ukkonen’s algorithm – example (13)

Step i=5	S=ABABABC$	S[i+1]=C
j=6	β=S[j..i]=S[6..5]=ε	Case: C2



Trees aren’t your thing?

- https://cp-algorithms.com/string/suffix-automaton.html

04 – Geometric algorithms

Advanced Algorithms and Data Structures

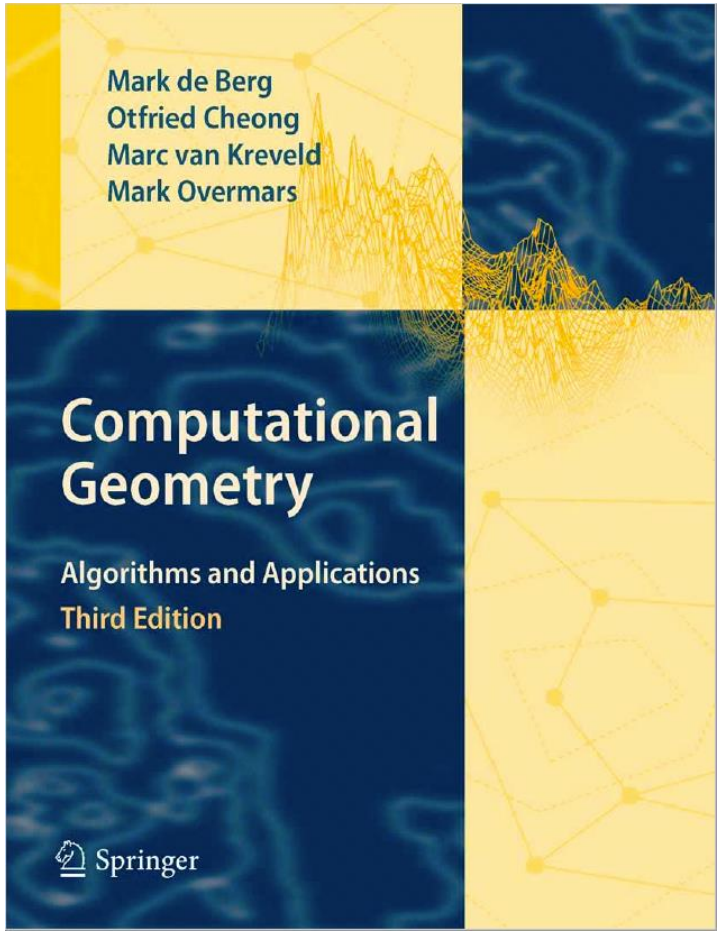
UNIVERSITY OF ZAGREB
Faculty of Electrical Engineering and Computing

Creative Commons

- You are free to:
  - share — multiply, distribute, and publicly communicate the work
  - adapt the work
- under the following conditions:
  - **Attribution:** You must acknowledge and indicate the authorship of the work in the manner specified by the author or license provider (but not in a way that suggests you or your use of the work have their direct support).
  - **Non-commercial:** You may not use this work for commercial purposes.
  - **Share alike:** If you modify, transform, or build upon this work, you may only distribute the modified work under the same or a similar license.

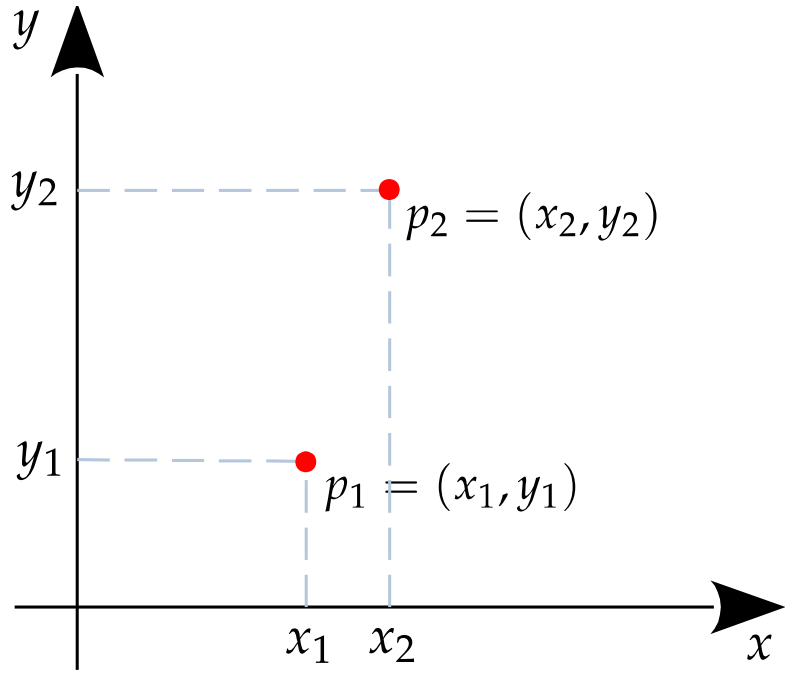
Geometric algorithms

- Uses in:
  - Computer graphics
  - Spatial product design (CAD)
  - Robotics
  - Geographic information systems
- Not just 2D and 3D
  - Consider everything as dimensionally expandable
  - $\mathbb{R} \times \mathbb{R} \times \mathbb{R} \times \cdots \times \mathbb{R}$



Basic concepts – point (1)

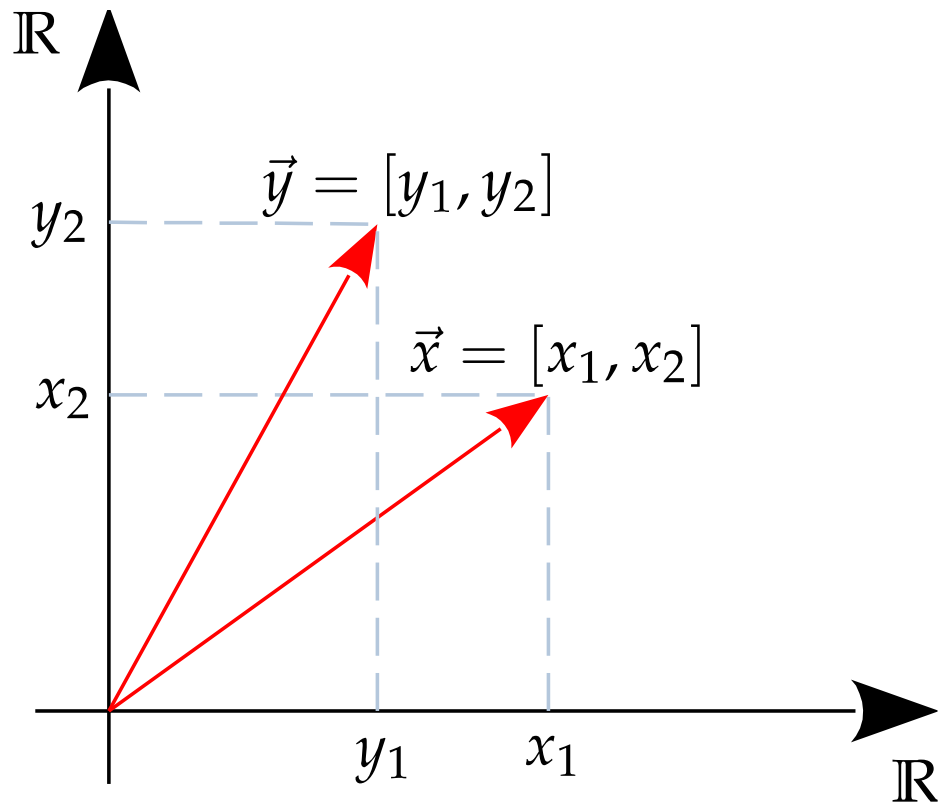
- Point
  - Infinitesimal element in $\mathbb{R}^n$
- Cartesian coordinate system



Basic concepts – point (2)

- For multiple dimensions
- As a vector: $\vec{x} = [x_1\ x_2\ x_3\ \dots\ x_n]$

- Dimensions as axes
  - Now numbered 1, 2



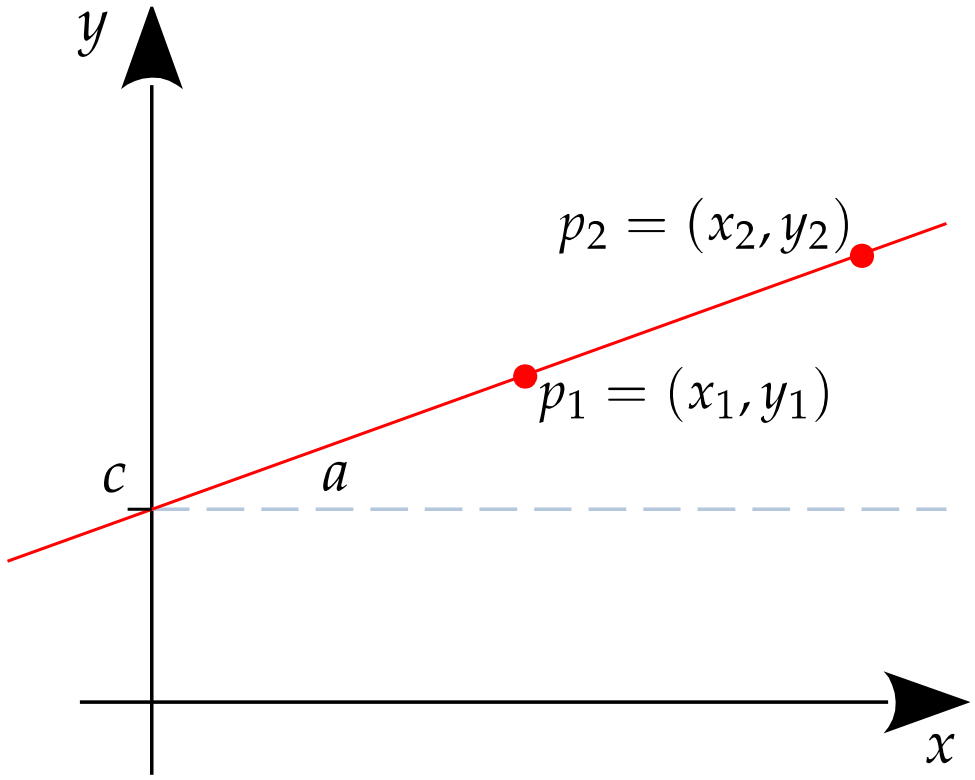
Basic concepts - line

- Infinite amount of points:

$$L = \{(x, y): (x, y) \in \mathbb{R}^2, ax + by = c\}$$

- Simply put: $y = \frac{c - ax}{b}$

- + other arithmetic expressions



Basic concepts – line segments

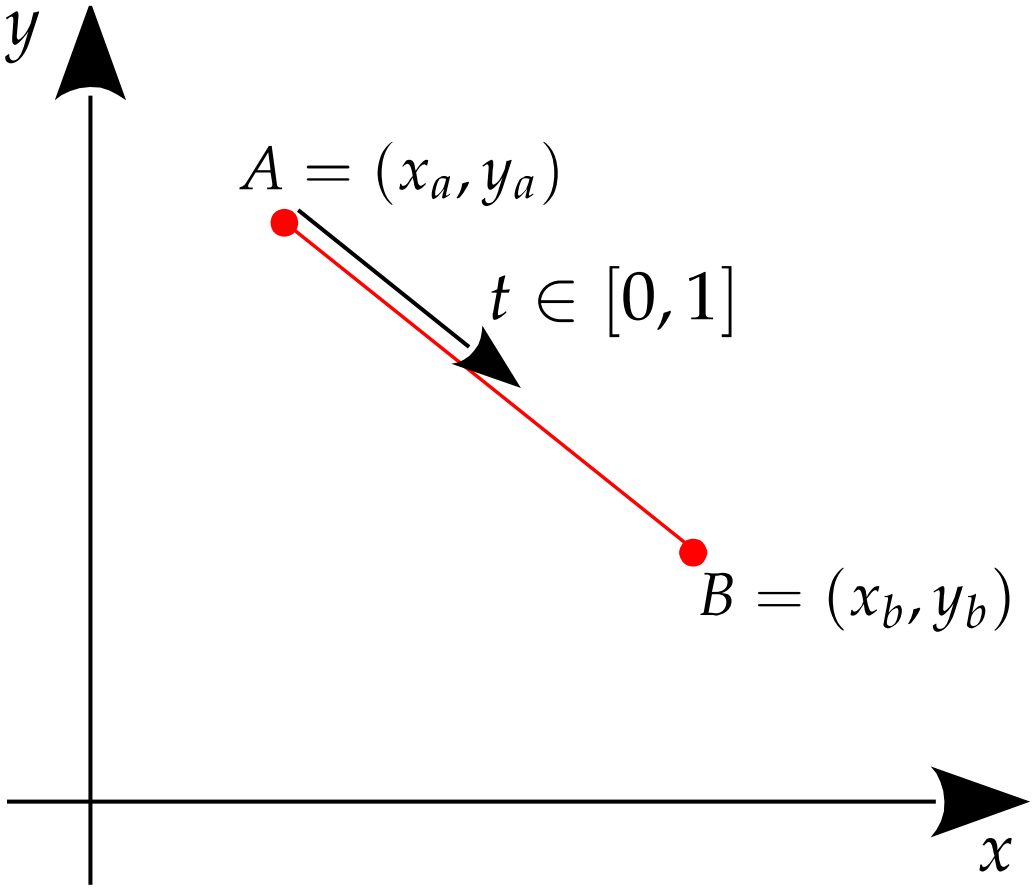
- A line can be parametrized by $t \in [0,1]$:

$$\Delta y = y_2 - y_1$$

$$\underline{\Delta x = x_2 - x_1}$$

$$y = y_1 + t\Delta y$$

$$x = x_1 + t\Delta x$$

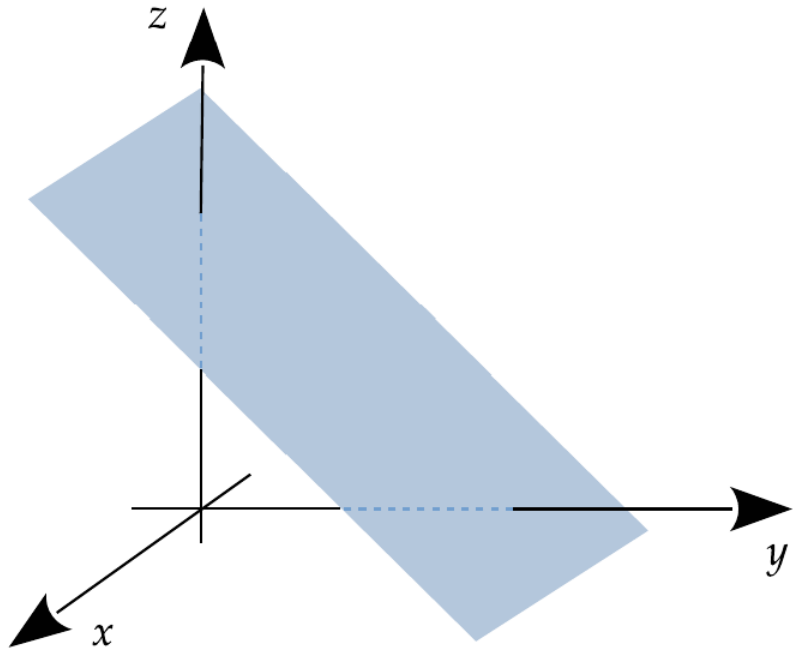


Basic concepts – plane

- Definition of a plane in 3D:

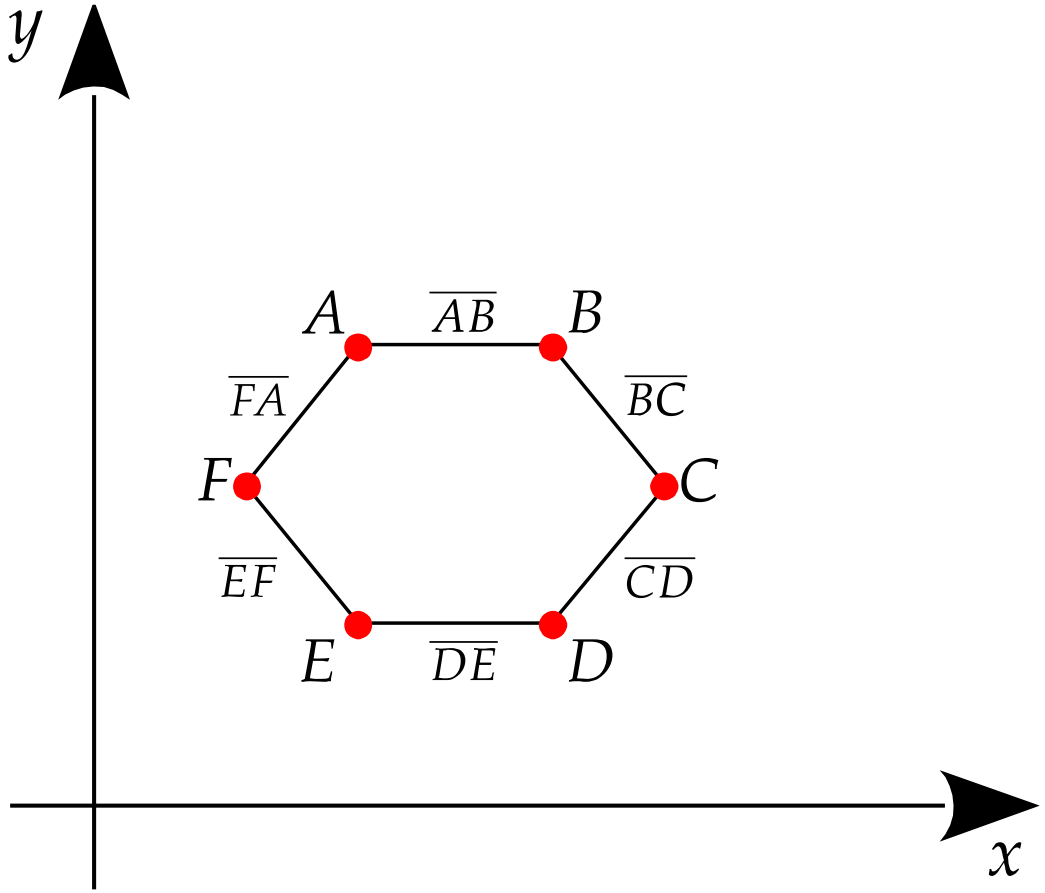
$$P = \{(x, y, z): (x, y, z) \in \mathbb{R}^3, ax + by + cz = d\}$$

- Geometric shape is a subset of points on a plane



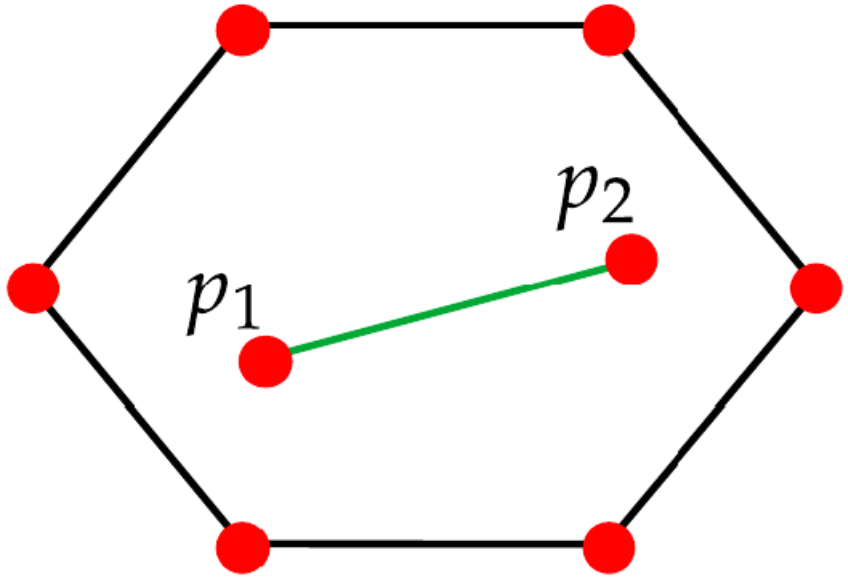
Basic concepts – polygon (1)

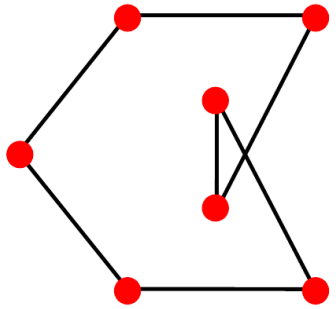
- What's a polygon?
  - Bounded by a subset of points on a plane
  - Bounded by line segments
- Body – internal points
- Chain or edge – line segments



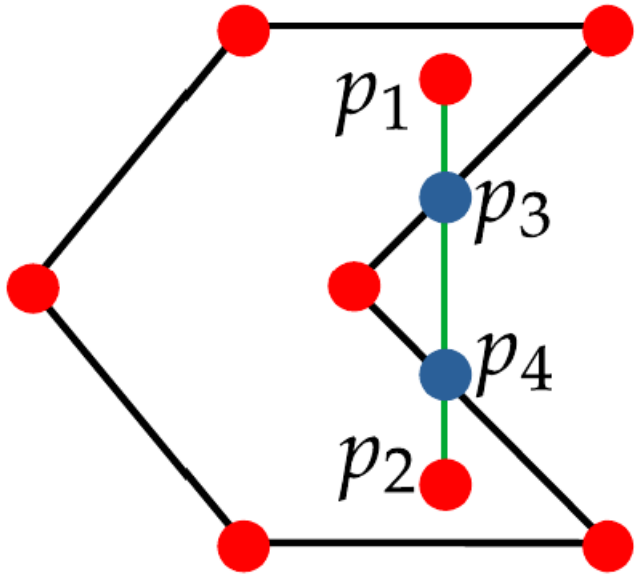
Basic concepts – polygon (2)

- Convex
  - All line segments of all $p_1, p_2 \in \mathcal{P}$ are inside the polygon
  - $p_1, p_2 \in \mathcal{P}: \overline{p_1 p_2} \in \mathcal{P}$
- Simple
  - Their edges don't cross
- Regular
  - Equiangular
  - Triangle, square, pentagon…





Not simple



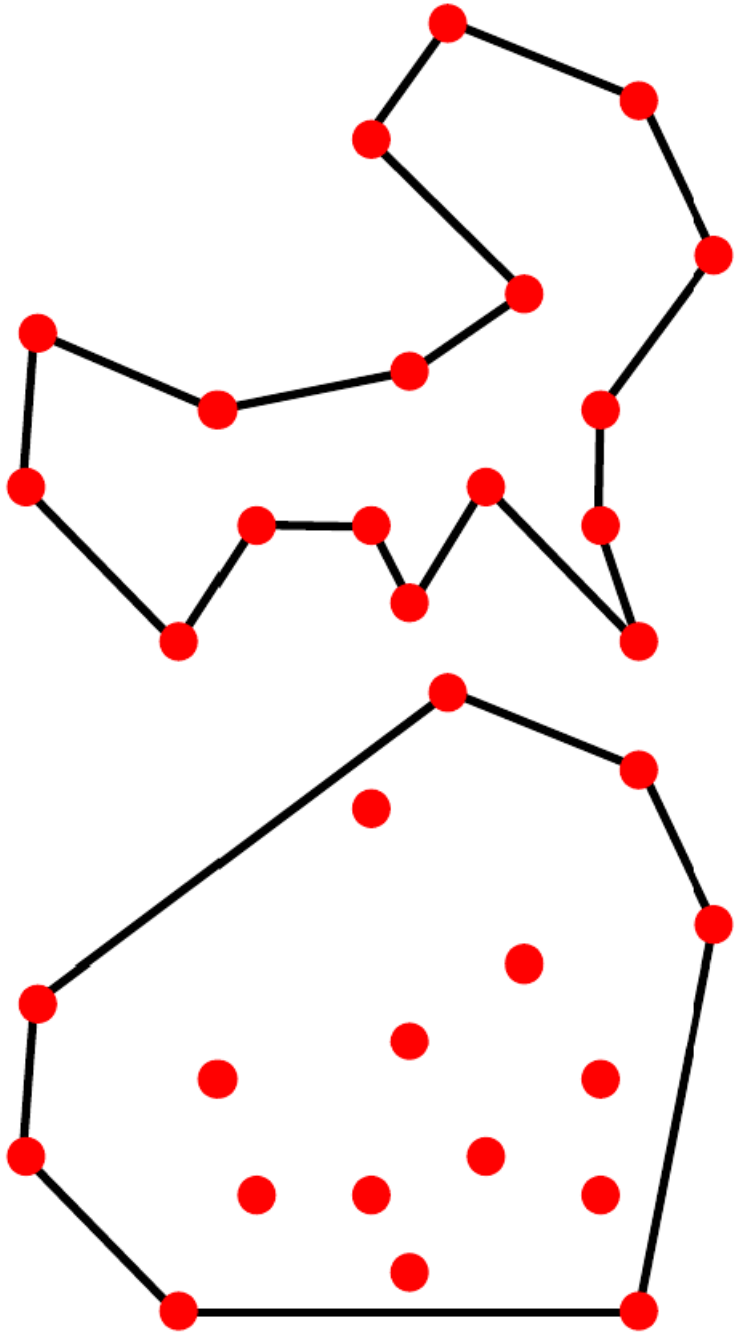
Convex hull I (1)

- A set of points $P_p = \{p_1, p_2, \dots, p_n\}$ in $\mathbb{R}^2$

- **Task:** Find as subset of P_p such that they make the edge of a polygon $\mathcal{P}$ that encompasses all points from P_p

$$\nexists p \in P_p : p \notin \mathcal{P}$$

- How do we get a CH?
- How do we check if we have it?



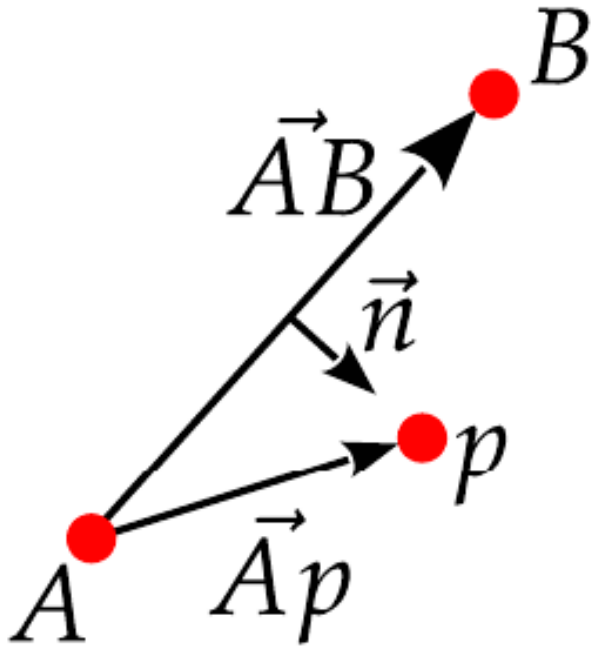
Convex hull I (2) – how to check?

- Pair of points on the hull:

$$\overrightarrow{AB} = [x_b - x_a \quad y_b - y_a]$$

- Set of equations:

$$\overrightarrow{AB} \cdot \vec{n}^T = 0$$
$$D = \vec{n} \cdot \overrightarrow{Ap}^T$$



- Solved to:

$$D = [y_b - y_a \quad x_a - x_b] \begin{bmatrix} x - x_a \\ y - y_a \end{bmatrix}$$
$$D = (x - x_a)(y_b - y_a) - (y - y_a)(x_b - x_a)$$

- $D > 0$ -> p is to the right of $\overrightarrow{AB}$

Convex hull I (3)

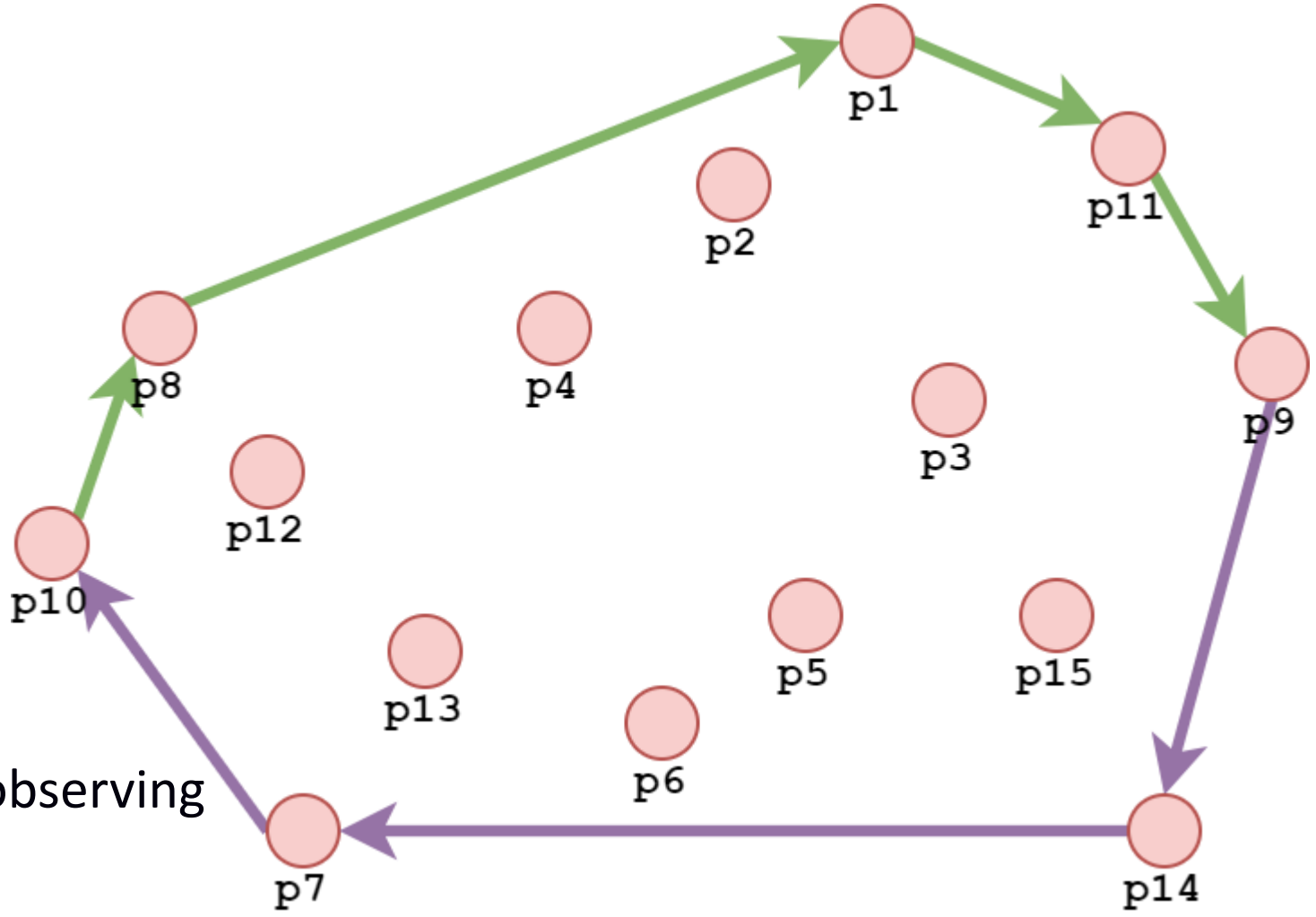
- Go through all pairs of points and find those that have nothing on the left
  - They make the lines of the convex hull

```
for each pair of points (p_i, p_j) in P_p:
    convex_hull = {}
    for each point p_k in (P_p \ {p_i, p_j}):
        if p_k left_of line(p_i, p_j):
            break #inner loop
    convex_hull.add(line(p_i, p_j))
```

- $O(n^3)$
  - Can we make this faster?

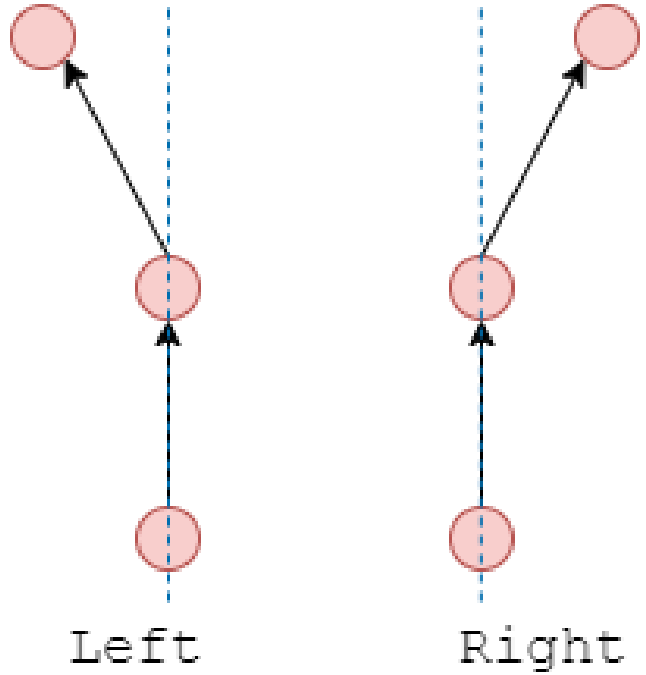
Convex hull II (1)

- **Strategy**: Split the hull into lower and upper "hull"
- Horizontally sort P_p
  - Get $h(P_p)$
  - $h(P_p) = \{p_{10}, \dots, p_9\}$
- Do inverted passes
  - Avoid left inclination when observing the last three points



Convex hull II (2)

- Both hulls must have the same incline
  - E.g. **right incline**
- We need at least 3 points to determine an incline



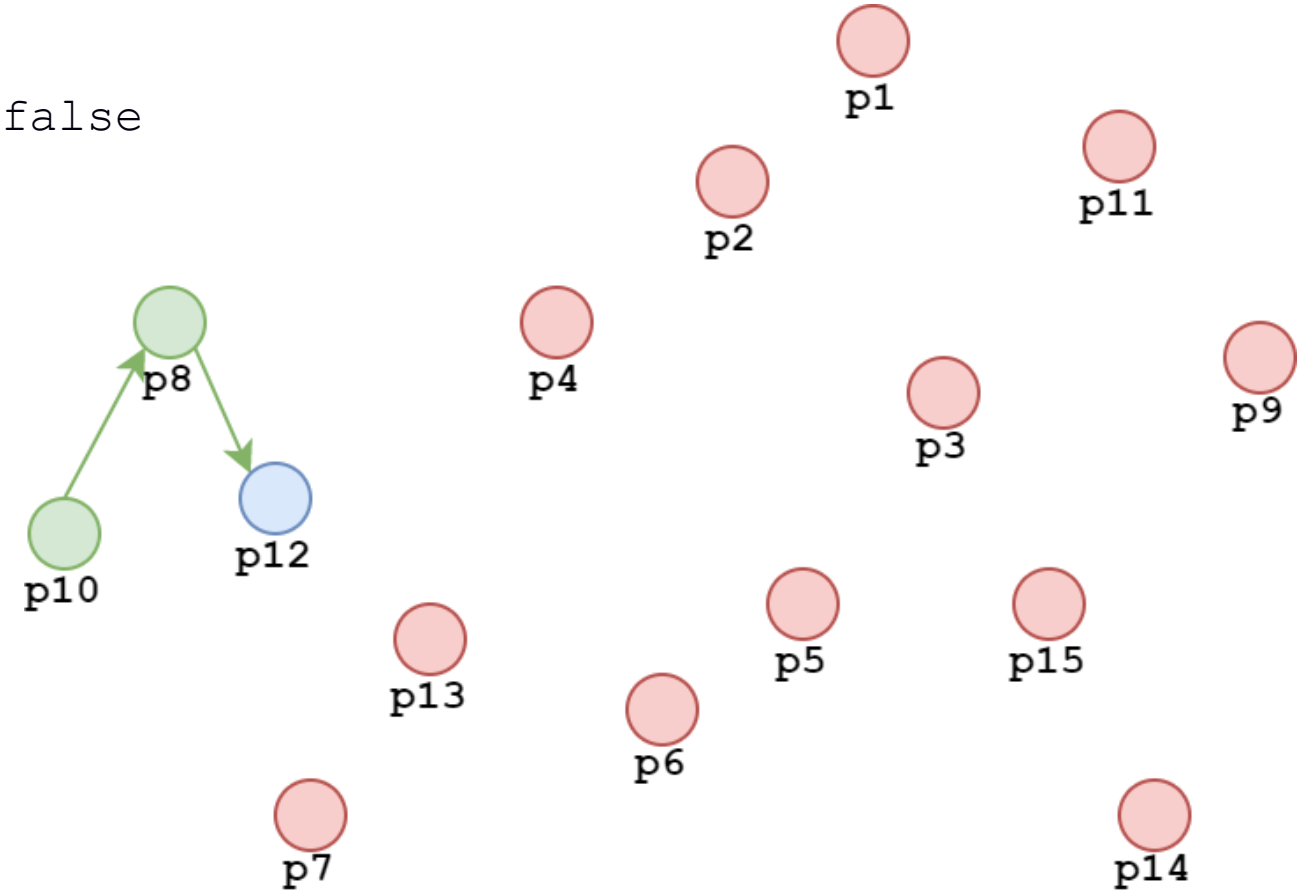
Convex hull II (3)

```
fun create_convex_hull(points):
    sorted_points = sort(points)
    n = sorted_points.count()
    upper_hull = {sorted_points[0], sorted_points[1]}
    for i in [2 .. n-1]:
        upper_hull.add(sorted_points[i])
        while upper_hull.count() >= 3 and are_inclined_left(p[i-2], p[i-1], p[i]):
            upper_hull.remove(i-1)
    lower_hull = {sorted_points[n-1], sorted[n-2]}
    for i in [n-3 .. 0]:
        lower_hull.add(sorted_points[i])
        while lower_hull.count() >= 3 and are_inclined_left(p[i], p[i-1], p[i-2]):
            lower_hull.remove(i+1)
    upper_hull + lower_hull
```

- Sort: $O(n \log n)$
- Passthrough: $O(n)$
- Completely: $O(n \log n + n) = O(n \log n)$

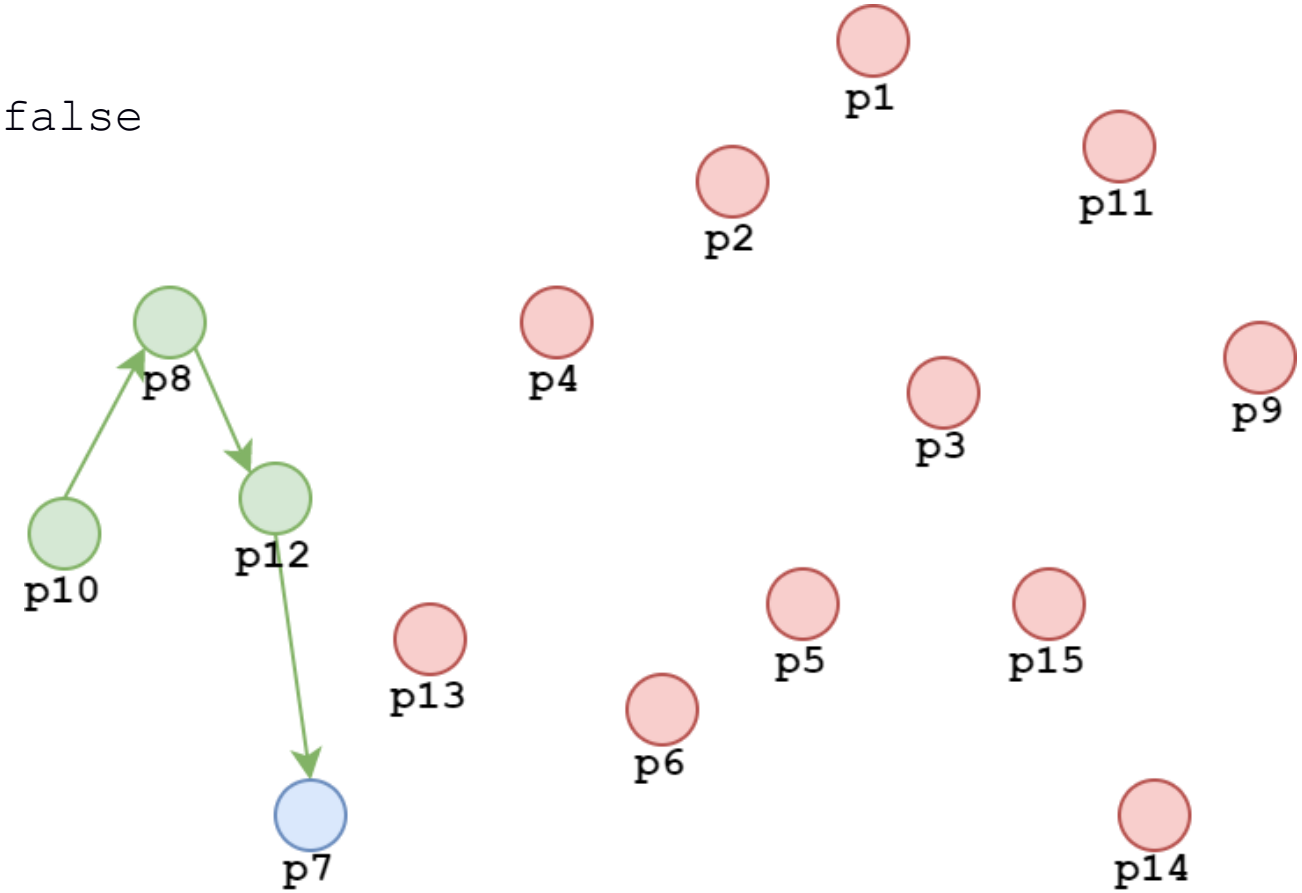
Convex hull II (4)

```
sorted_points = {p10, p8, p12, p7, p13, p4, p6, p2, p5, p1, p3, p15, p11, p14, p9}
upper_hull = {p10, p8, p12}
i = 2
is_left_incline = false
```



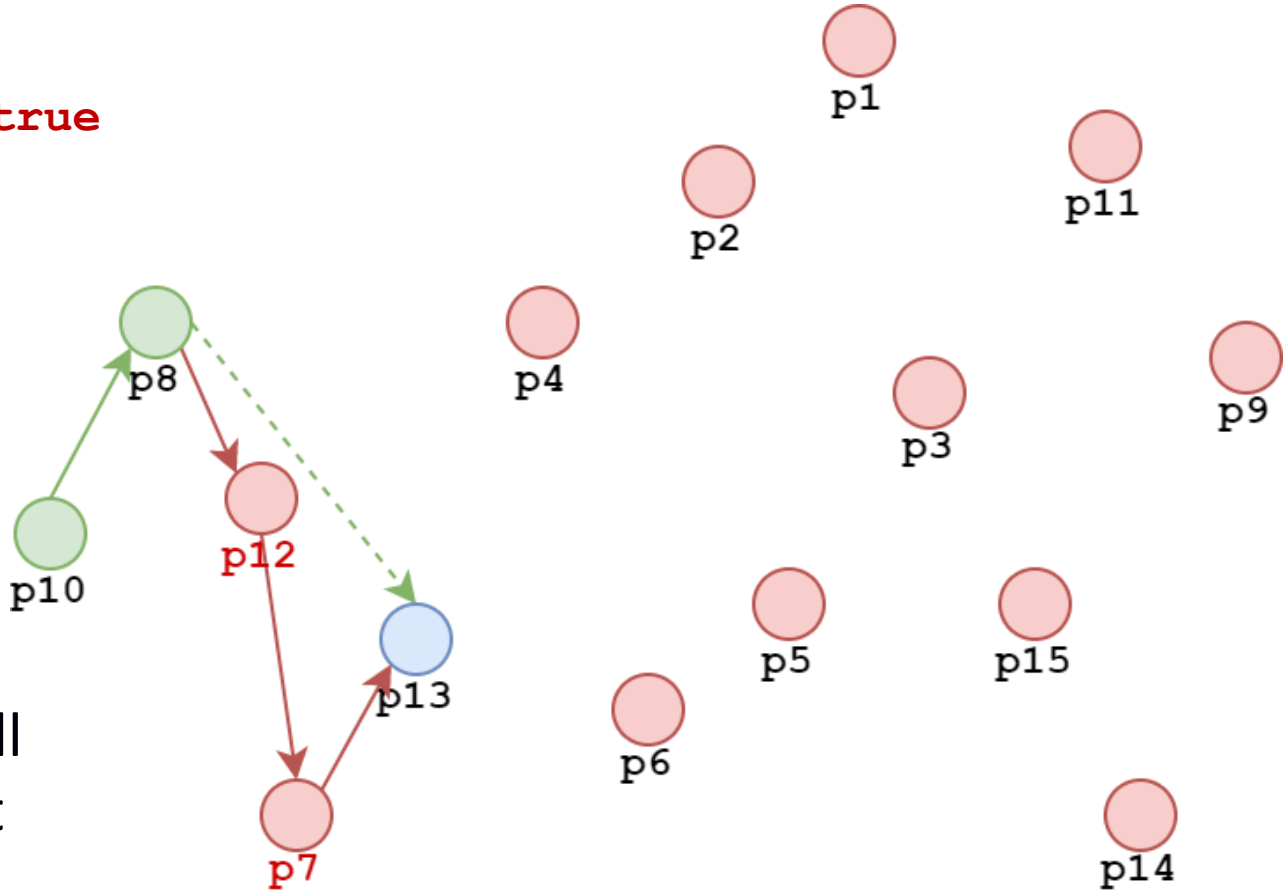
Convex hull II (5)

```
sorted_points = {~~p10, p8, p12,~~ p7, p13, p4, p6, p2, p5, p1, p3, p15, p11, p14, p9}
upper_hull = {p10, p8, p12, p7}
i = 3
is_left_incline = false
```



Convex hull II (6)

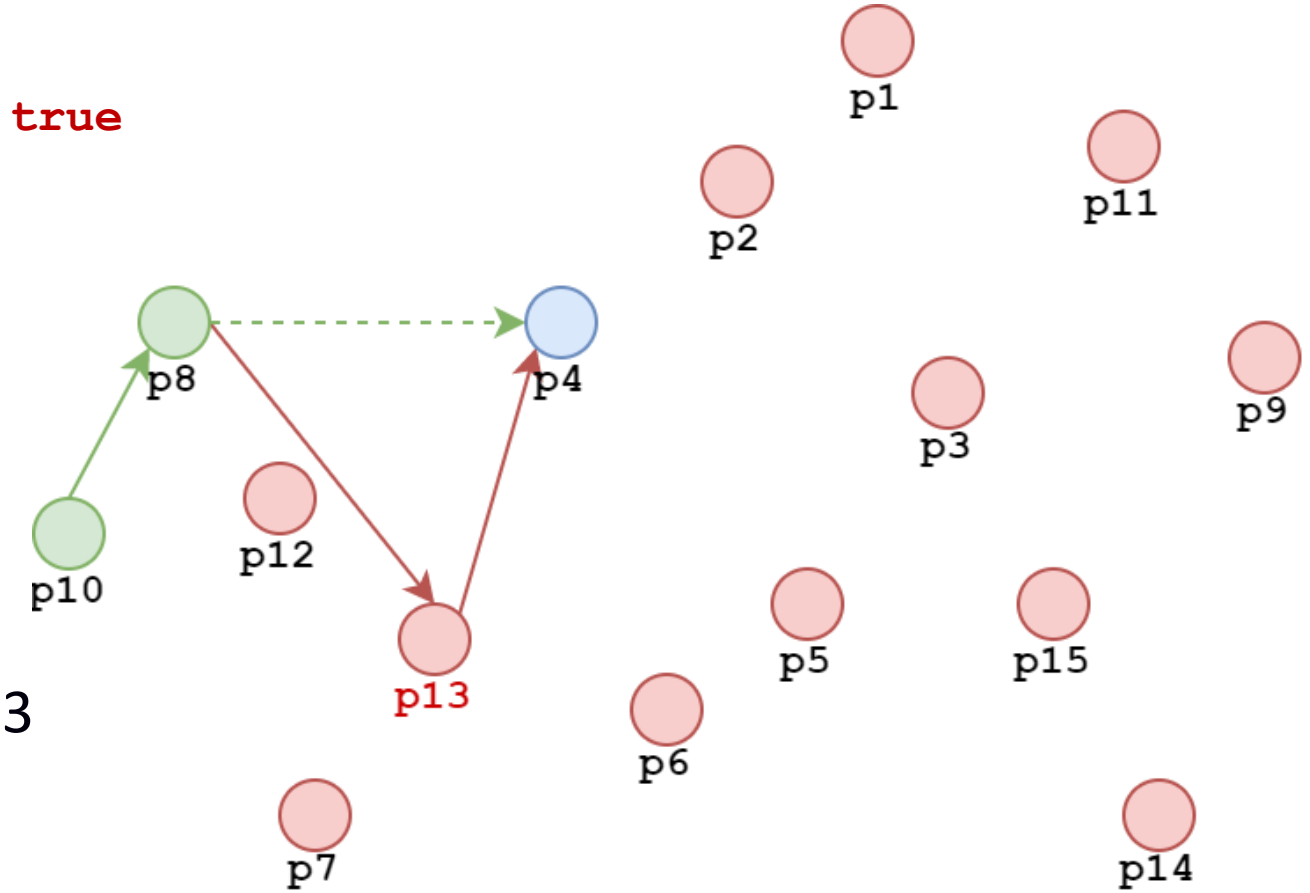
```
sorted_points = {p10, p8, p12, p7, p13, p4, p6, p2, p5, p1, p3, p15, p11, p14, p9}
upper_hull = {p10, p8, p12, p7, p13}
i = 4
is_left_incline = true
```



Keep removing the second last in the hull **`while`** there is a left incline

Convex hull II (7)

```
sorted_points = {p10, p8, p12, p7, p13, p4, p6, p2, p5, p1, p3, p15, p11, p14, p9}
upper_hull = {p10, p8, p13, p4}
i = 5
is_left_incline = true
```



Second last is now 13

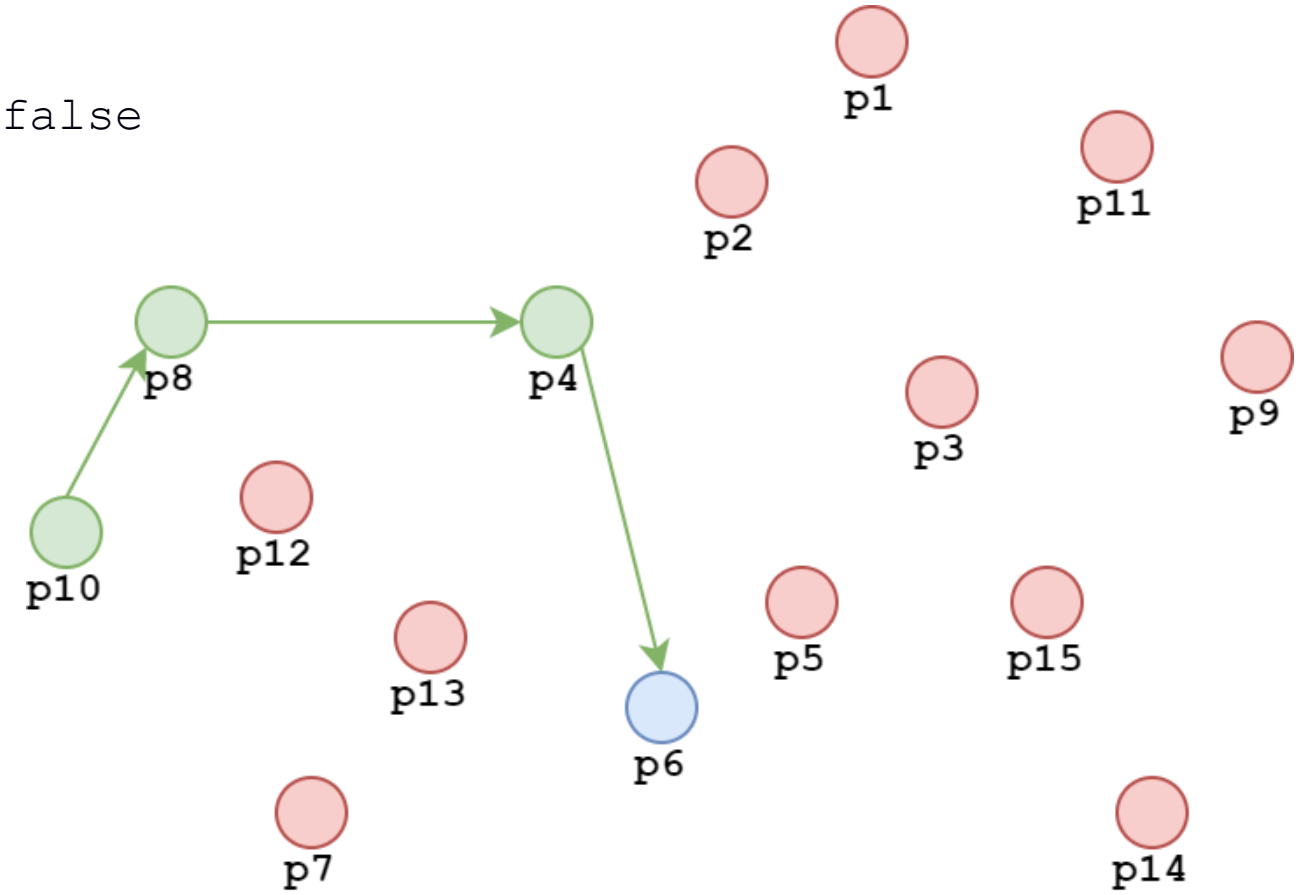
Convex hull II (8)

sorted_points = {~~p10, p8, p12, p7, p13, p4,~~ p6, p2, p5, p1, p3, p15, p11, p14, p9}

upper_hull = {p10, p8, p4, p6}

i = 6

is_left_incline = false



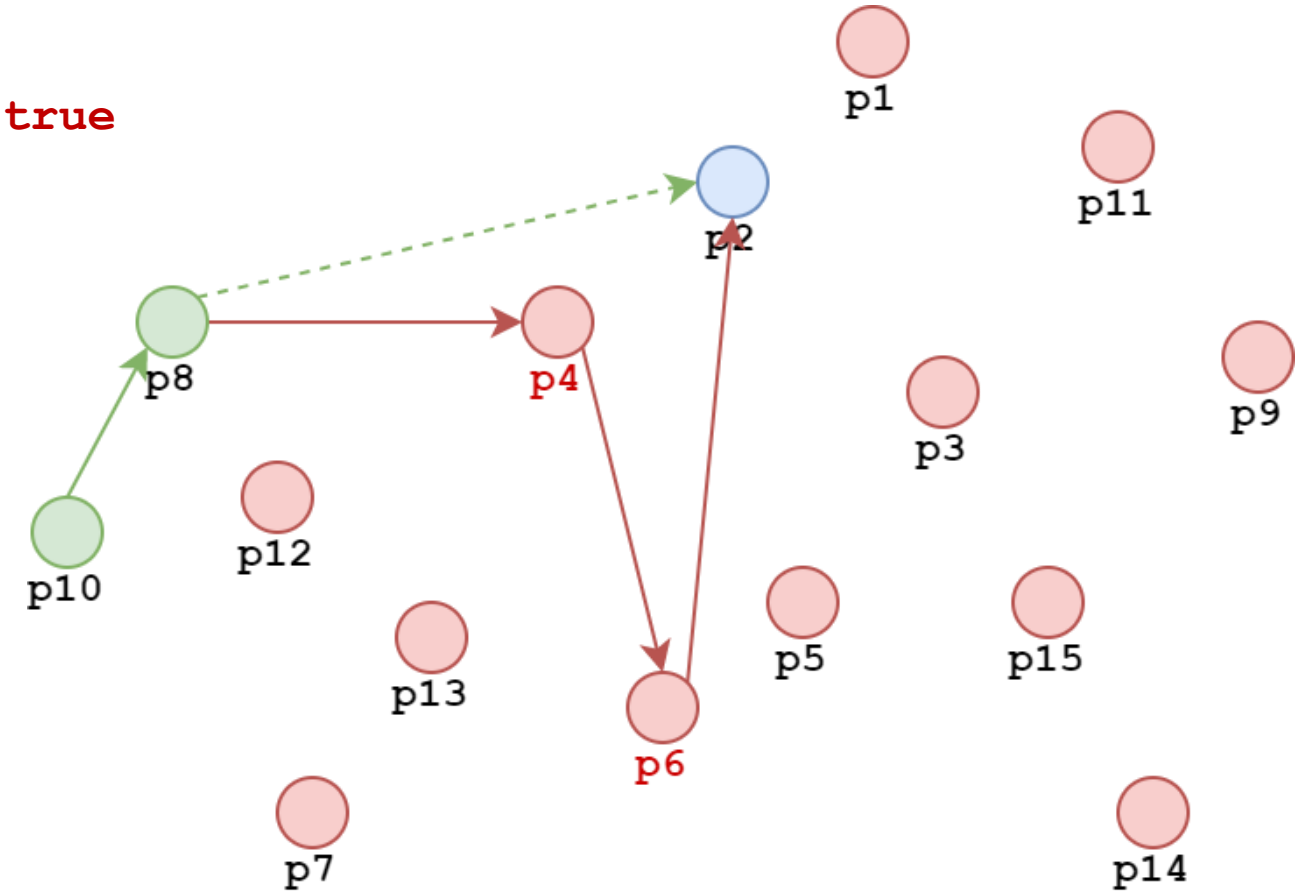
Convex hull II (9)

sorted_points = {~~p10, p8, p12, p7, p13, p4, p6,~~ p2, p5, p1, p3, p15, p11, p14, p9}

upper_hull = {p10, p8, **p4**, **p6**, p2}

i = 7

is_left_incline = **true**



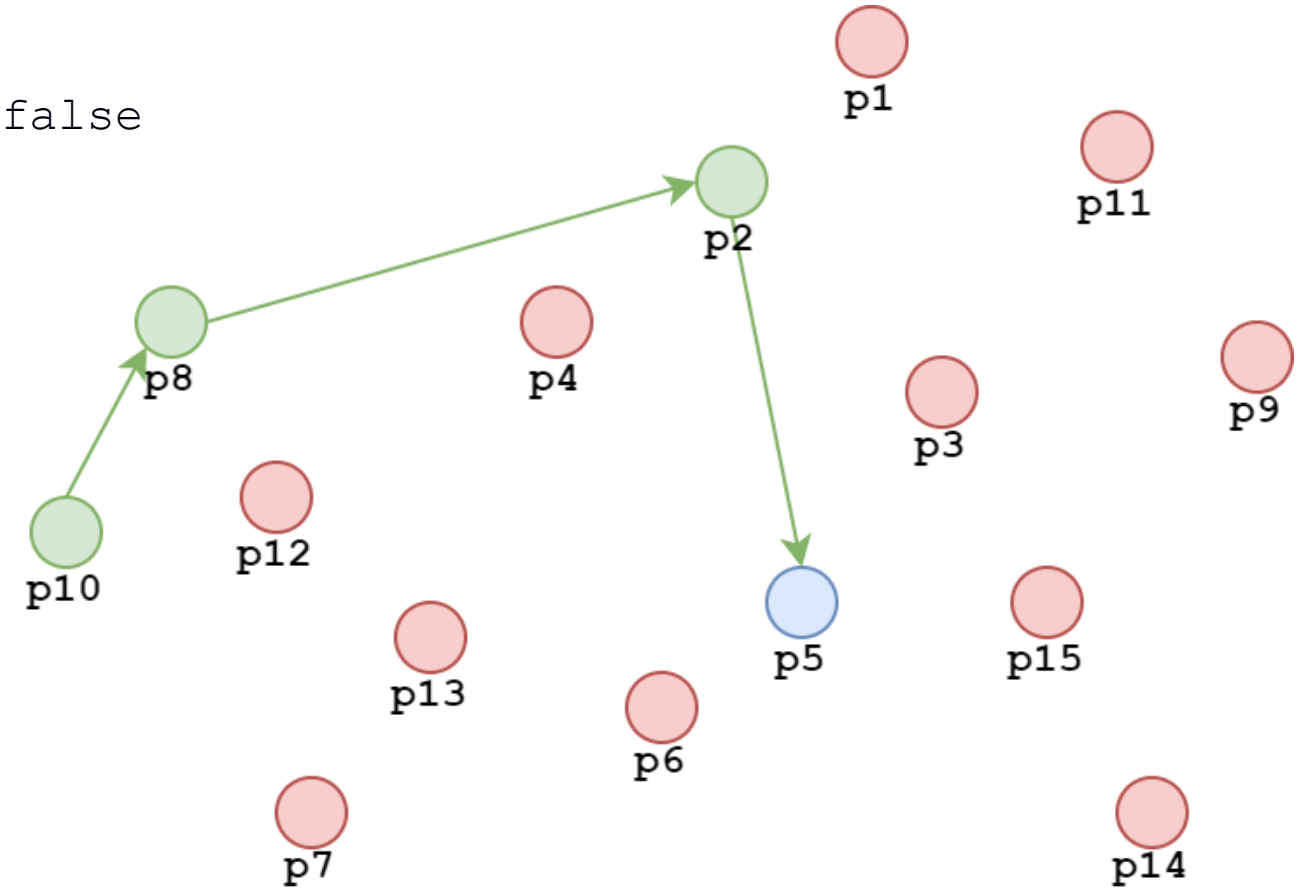
Convex hull II (10)

sorted_points = {~~p10, p8, p12, p7, p13, p4, p6, p2,~~ p5, p1, p3, p15, p11, p14, p9}

upper_hull = {p10, p8, p2, p5}

i = 8

is_left_incline = false



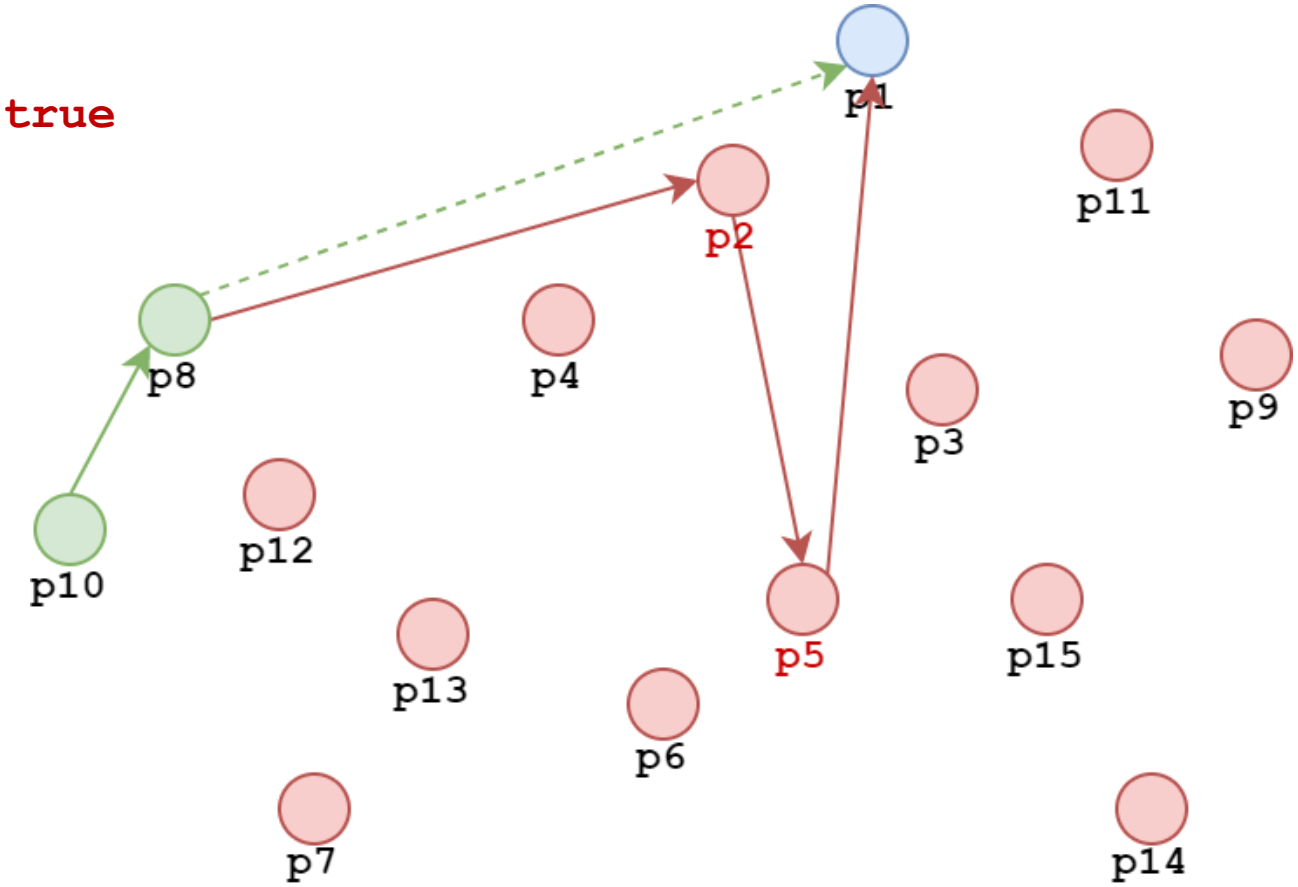
Convex hull II (11)

sorted_points = {~~p10, p8, p12, p7, p13, p4, p6, p2, p5,~~ p1, p3, p15, p11, p14, p9}

upper_hull = {p10, p8, **p2**, **p5**, p1}

i = 9

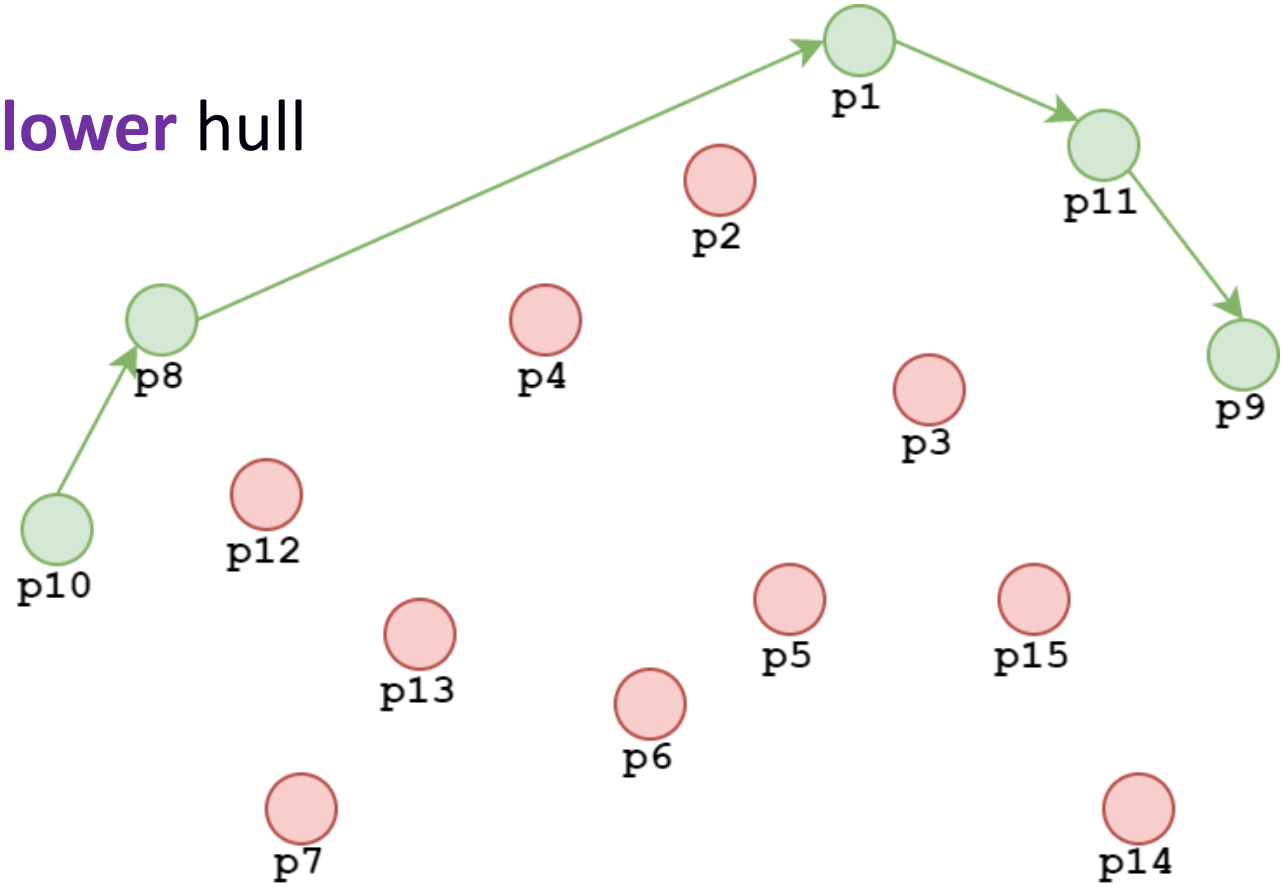
is_left_incline = **true**



Convex hull II (11)

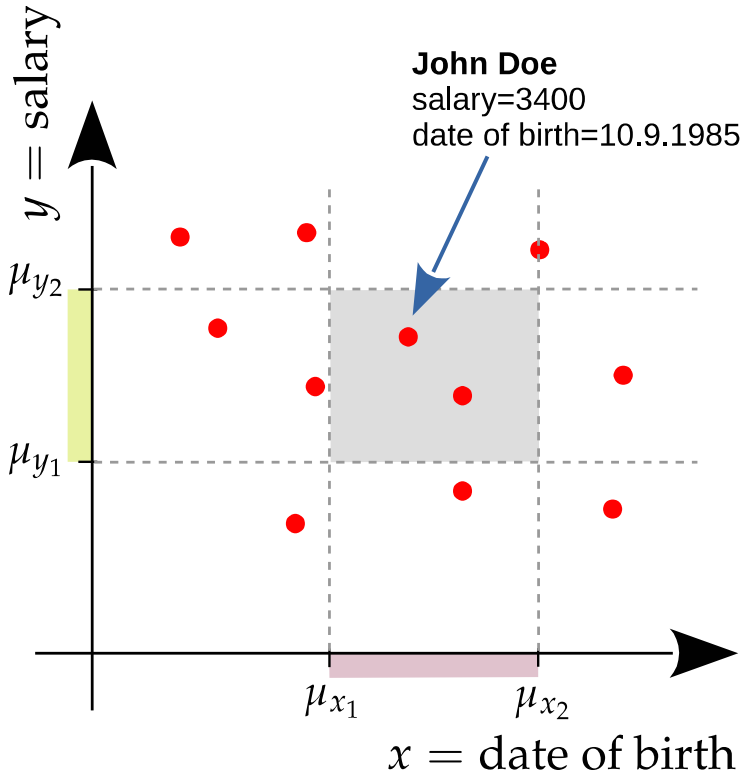
```
sorted_points = {p10, p8, p12, p7, p13, p4, p6, p2, p5, p1, p3, p15, p11, p14, p9}
upper_hull = {p10, p8, p1, p11, p9}
```

- And then the **lower** hull



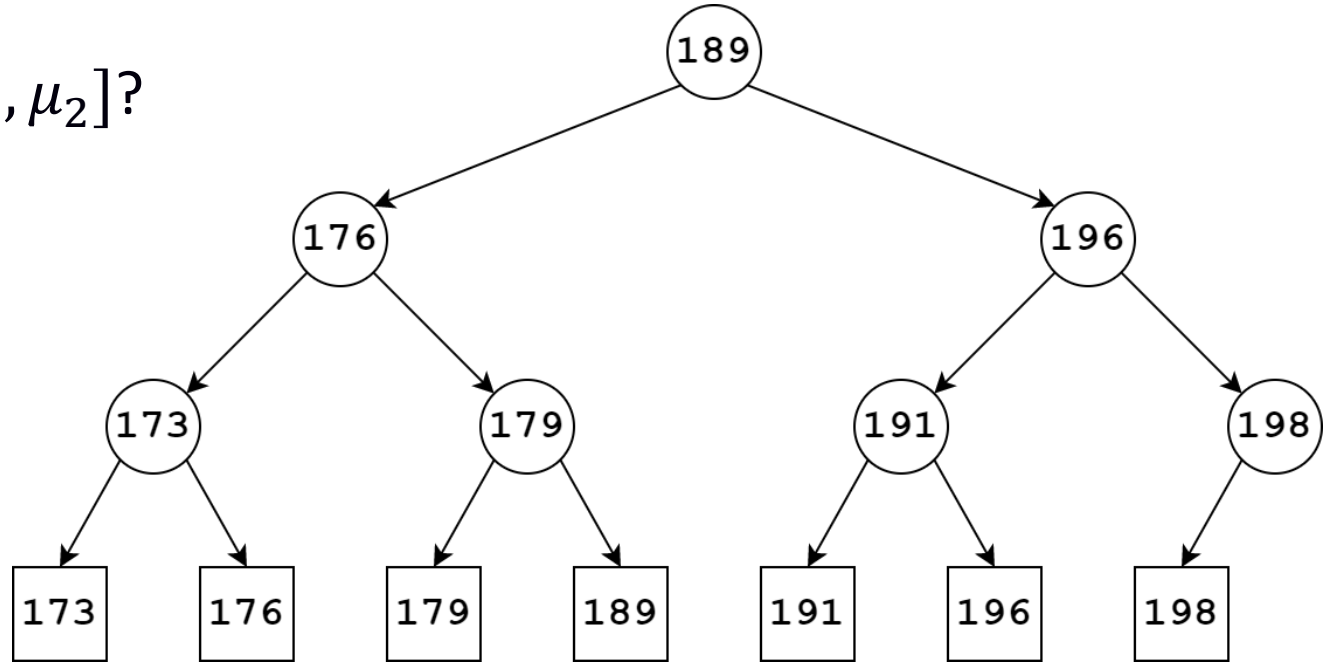
Geometric search

- N variables
  - N dimensional search space
- Height, weight, salary…
- Databases
  - B+-trees?



1D range search – Binary range trees

- Use-case: 1D range queries
  - What keys exist in range $[\mu_1, \mu_2]$?
- Children as leaves
- Different from B+ trees?



Binary range trees – search

- Find the splitting node n_{split} (ordinary traversal)
  - The first node within the range $[\mu_1, \mu_2]$
- Initiate left and right search
  - Skip redundant subtrees

n_{split} left subtree

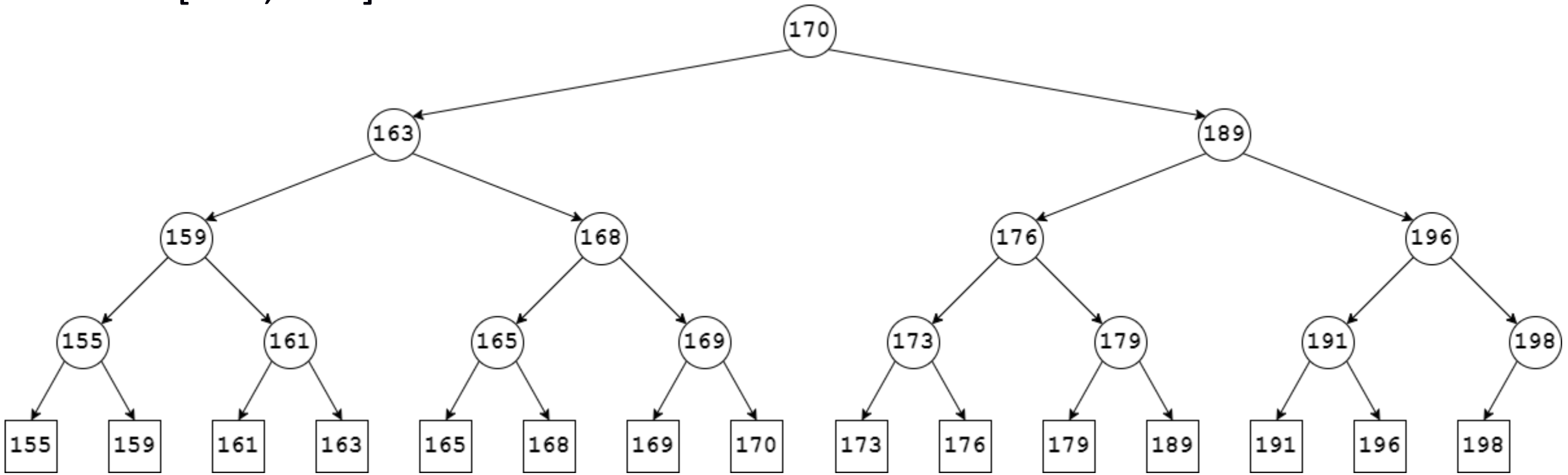
```
if μ1 ≤ n :
      take right subtree
      traverse to ChL(n)
if μ1 > n :
      skip
      traverse to ChR(n)
```

n_{split} right subtree

```
if μ2 ≥ n :
      take left subtree
      traverse to ChR(n)
if μ2 < n :
      skip
      traverse to ChL(n)
```

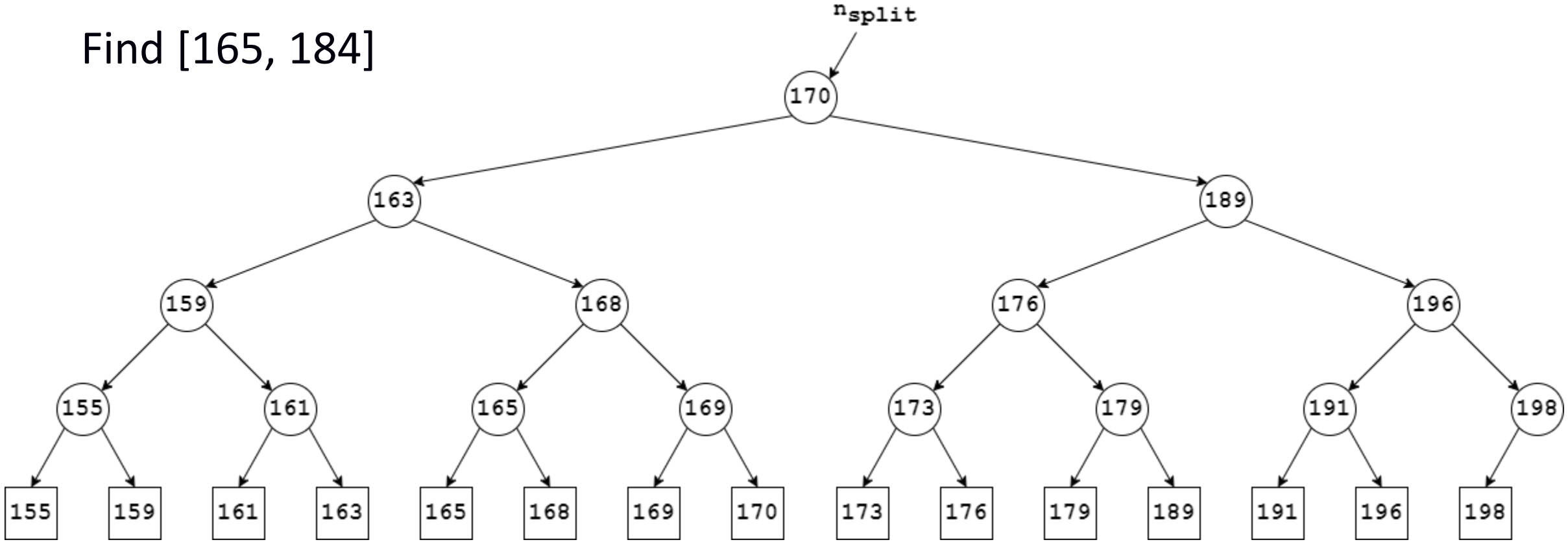
Binary range trees – search example (1)

Find [165, 184]



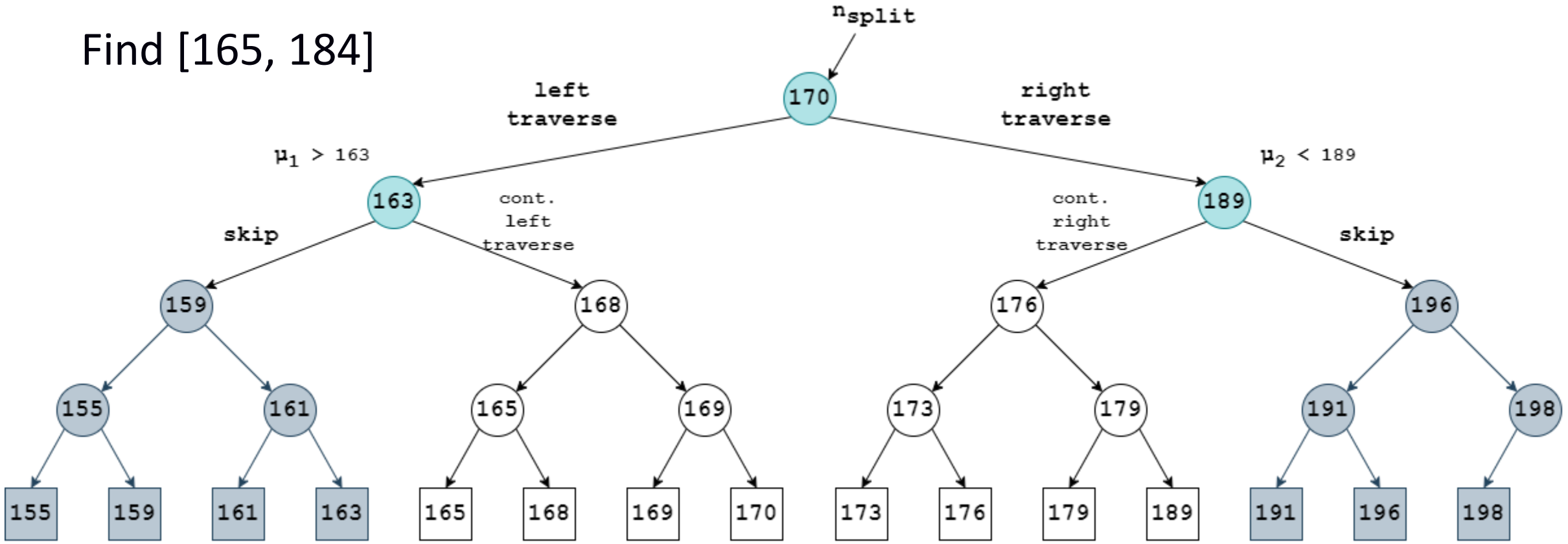
Binary range trees – search example (2)

Find [165, 184]



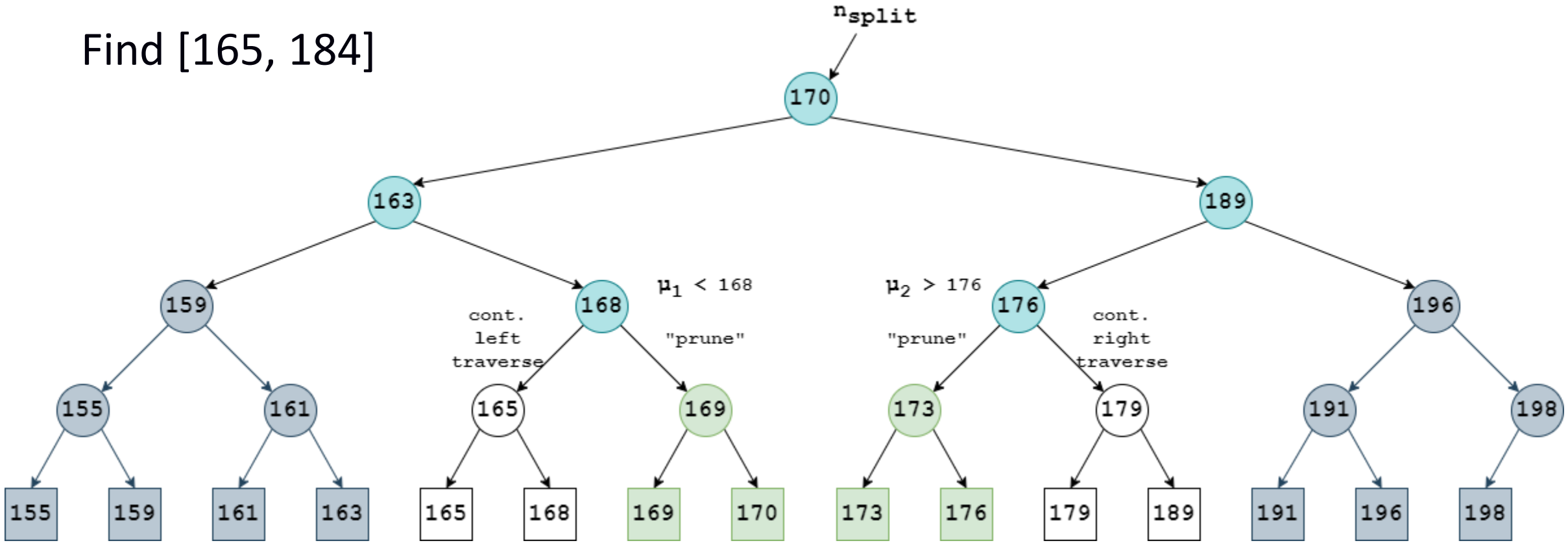
Binary range trees – search example (3)

Find [165, 184]



Binary range trees – search example (4)

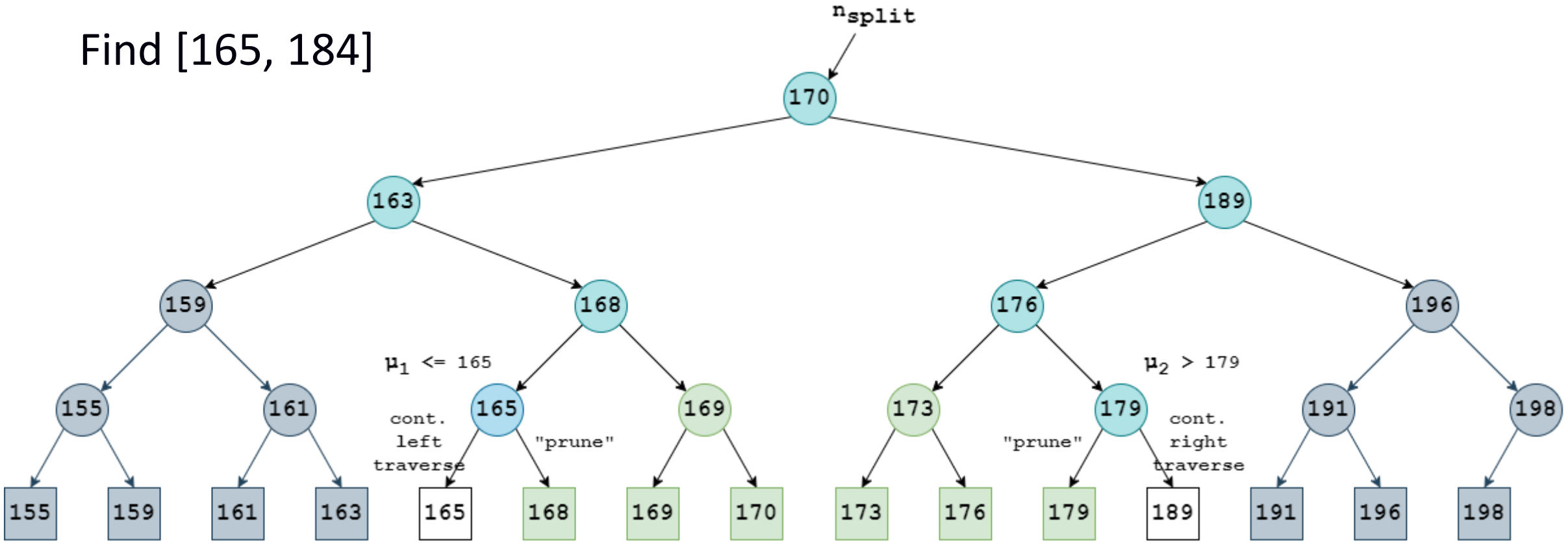
Find [165, 184]



*“prune” as in “just take everything”

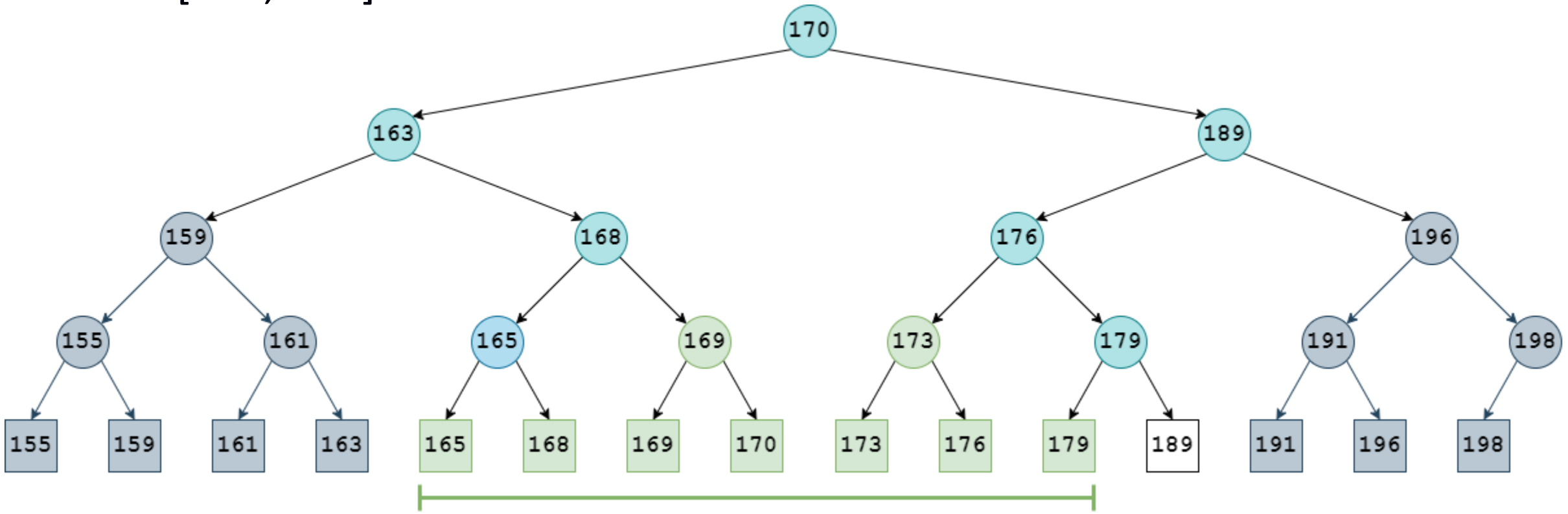
Binary range trees – search example (5)

Find [165, 184]



Binary range trees – search example (6)

Find [165, 184]



Leaves reached – DONE!

Binary range trees – search

- 2 parts (functions):
  - Find splitting node (root in example)
  - Right + Left traversal

- Complexity: $O(k + \log n)$
- k nodes pruned
- $\log n$ binary traversal search

Binary range trees – creation (1)

- Just like a sorted binary tree
- But we take **medians** as root and internal nodes' values

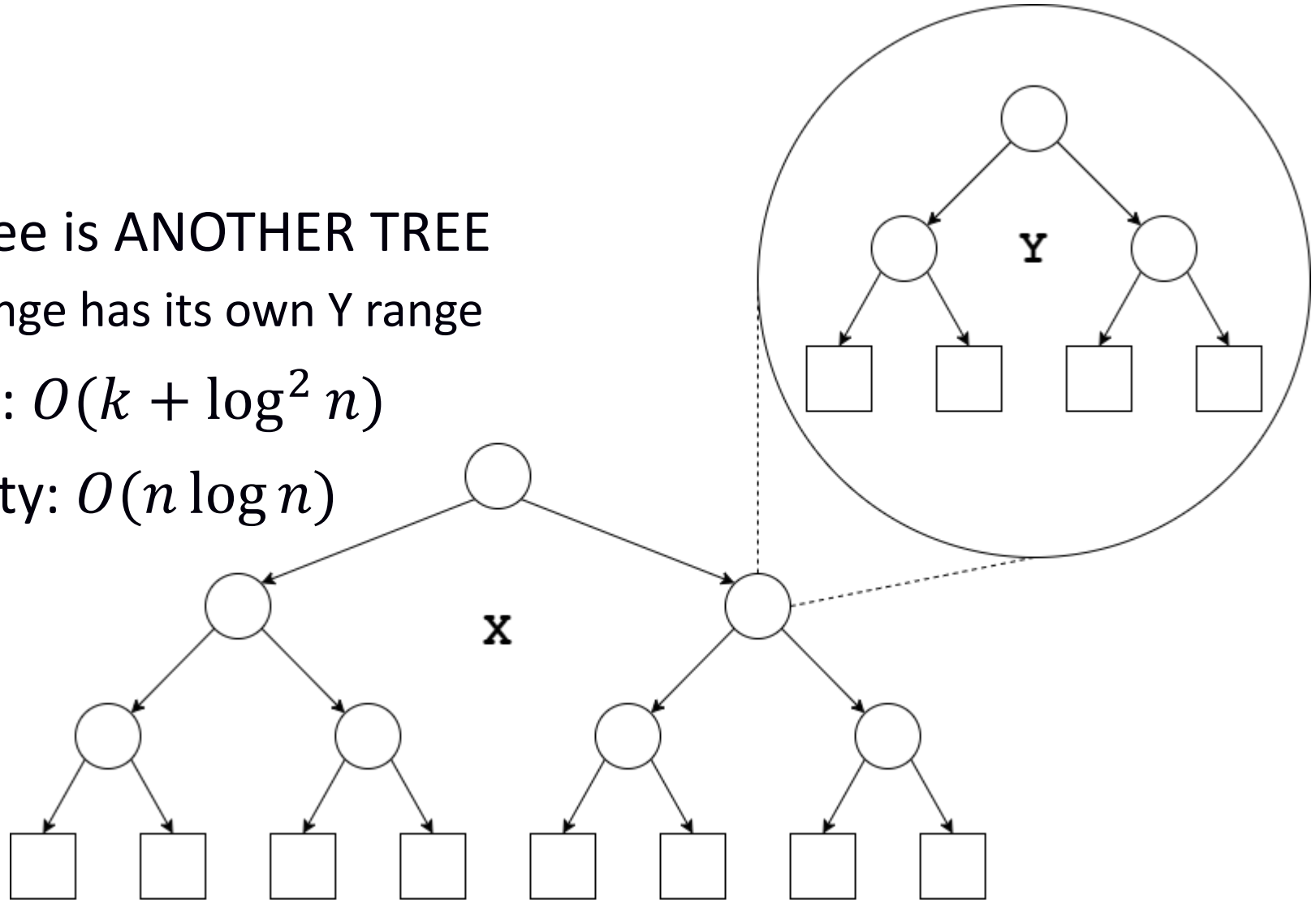
- Take an array and split it over medians
- Complexity: $O(n \log n)$

Binary range trees – creation (2)

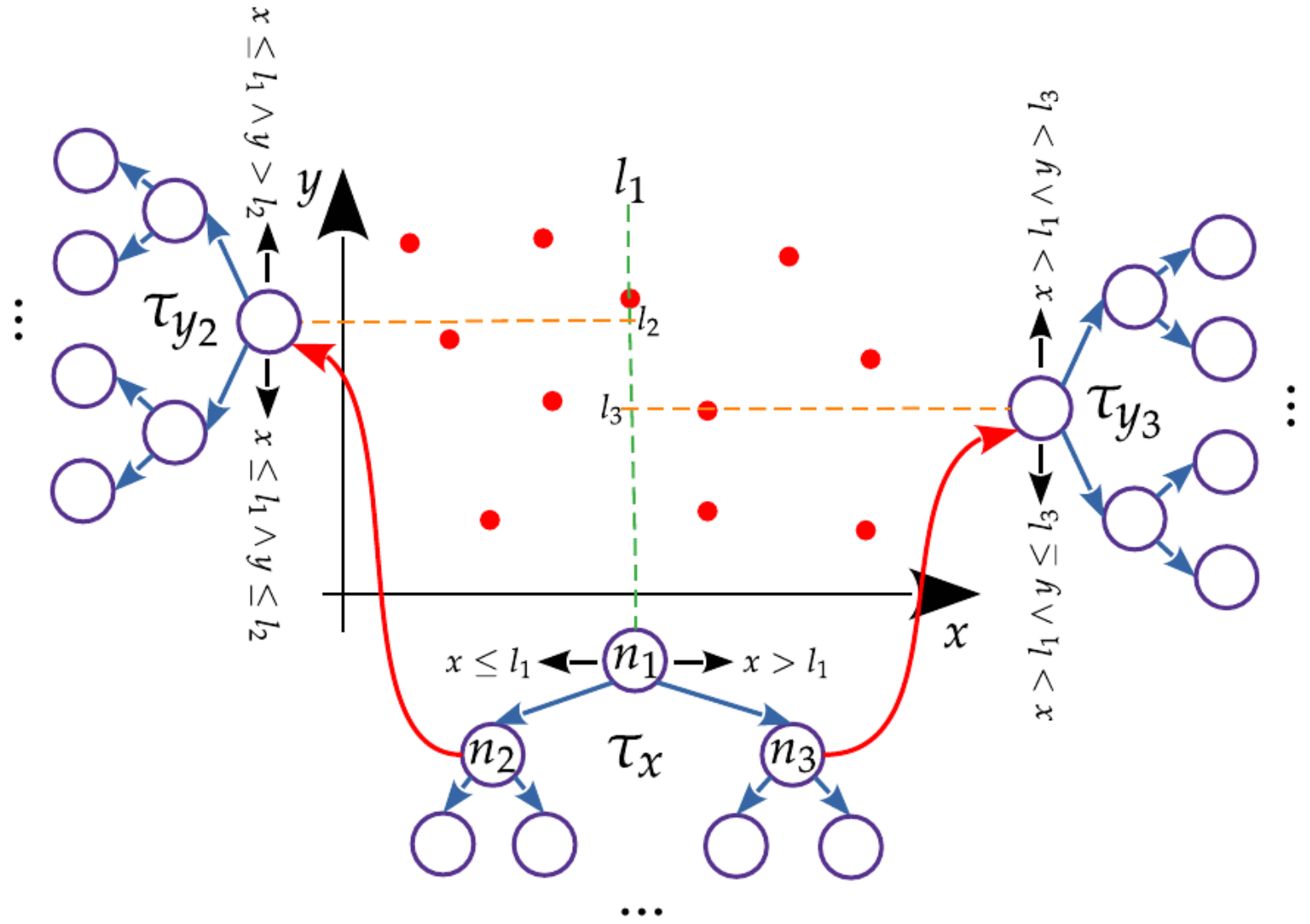
```
fun create_1d_range_tree(points):
    if points.count() == 1:
        points[0] # as only and root node
    else:
        curr_median = median(points)
        points_left = [for point in points if point <= curr_median]
        points_right = [for point in points if point > curr_median]
        left_subtree = create_1d_range_tree(points_left)
        right_subtree = create_1d_range_tree(points_right)
        curr_node = Node(curr_median)
        curr_node.left = left_subtree
        curr_node.right = right_subtree
        curr_node
```

2D range search

- 2D range tree
- Every node of a tree is ANOTHER TREE
  - Each x in the X range has its own Y range
- Search complexity: $O(k + \log^2 n)$
- Creation complexity: $O(n \log n)$



2D range search

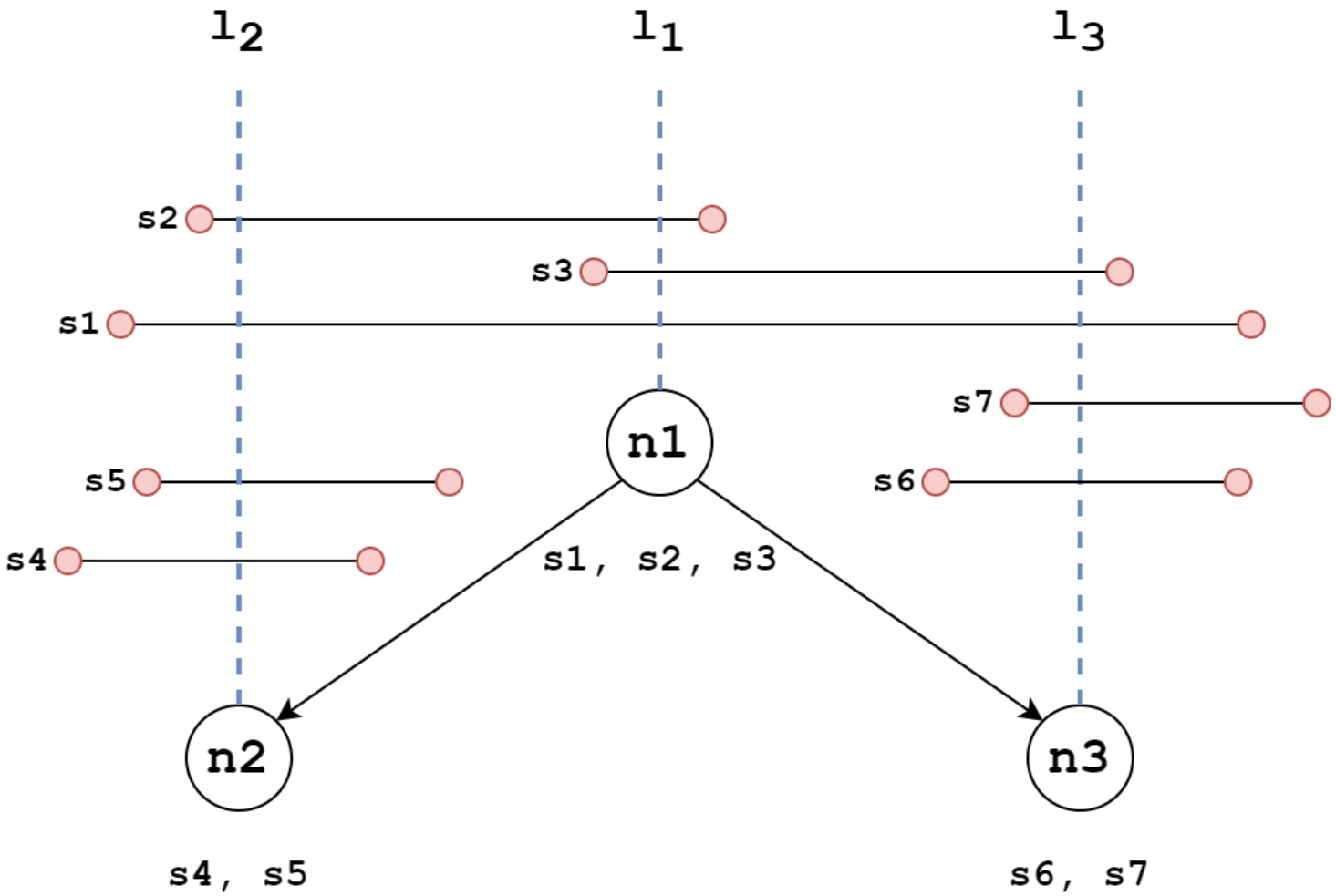


“1D” interval search

- A set of intervals: $I = \{s_i = ([x_{i_1}, x_{i_2}], y_i) : x_{i_1}, x_{i_2}, y_i \in \mathbb{R}\}$
  - Parallel with x-axis (y is const. per interval)
  - Not really 1D since, we can take into account the y-axis
- Each interval has 2 end points:
  - Left: (x_{i_1}, y_i)
  - Right: (x_{i_2}, y_i)
- Interval tree

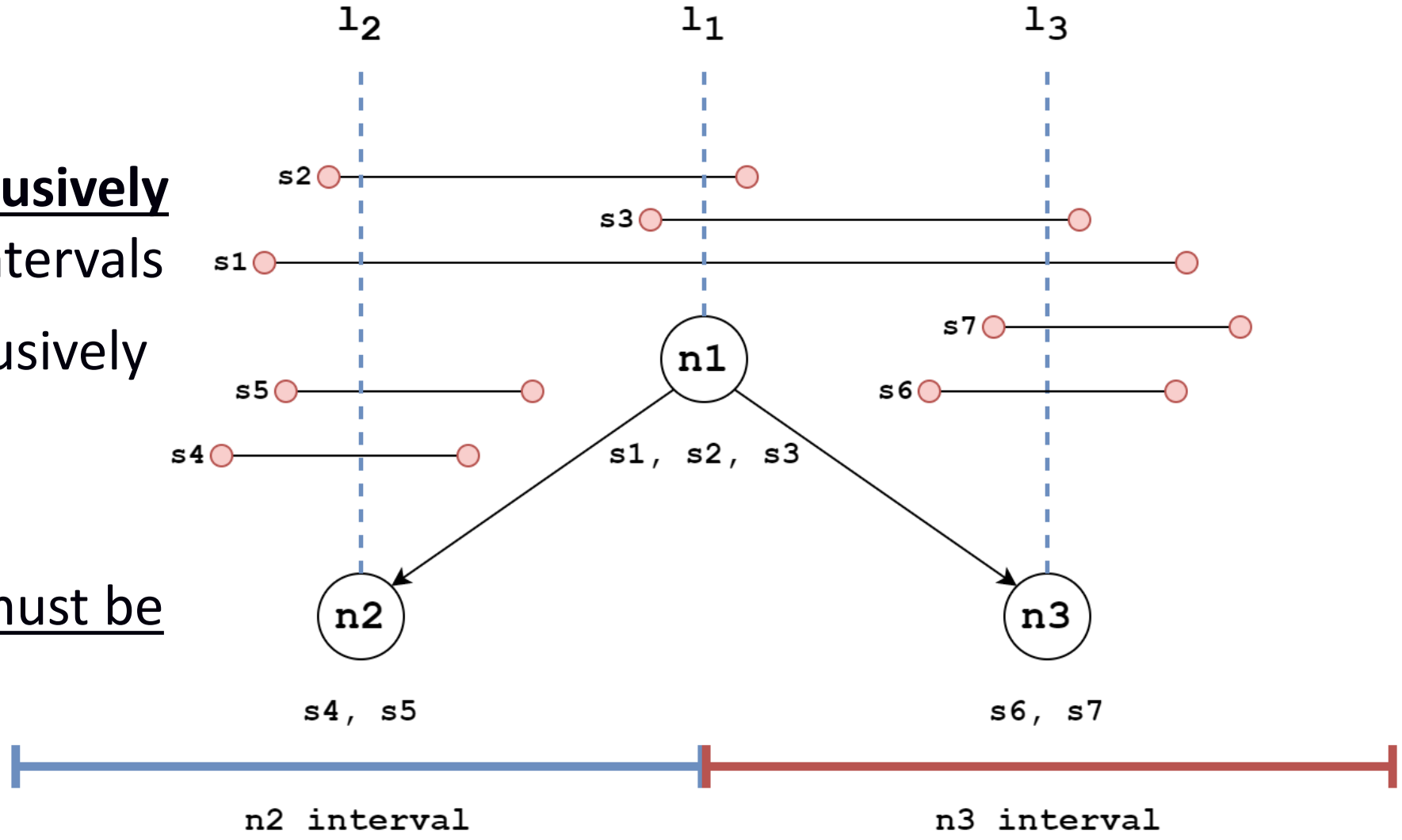
Interval tree (1)

- Sorted around medians:
  - $l_i = med(n_i)$
  - defined by the intervals in the node
- Why isn't s_3 in n_3?
- Why isn't s_2 in n_2?
- We sort by medians, but emulate ranges!



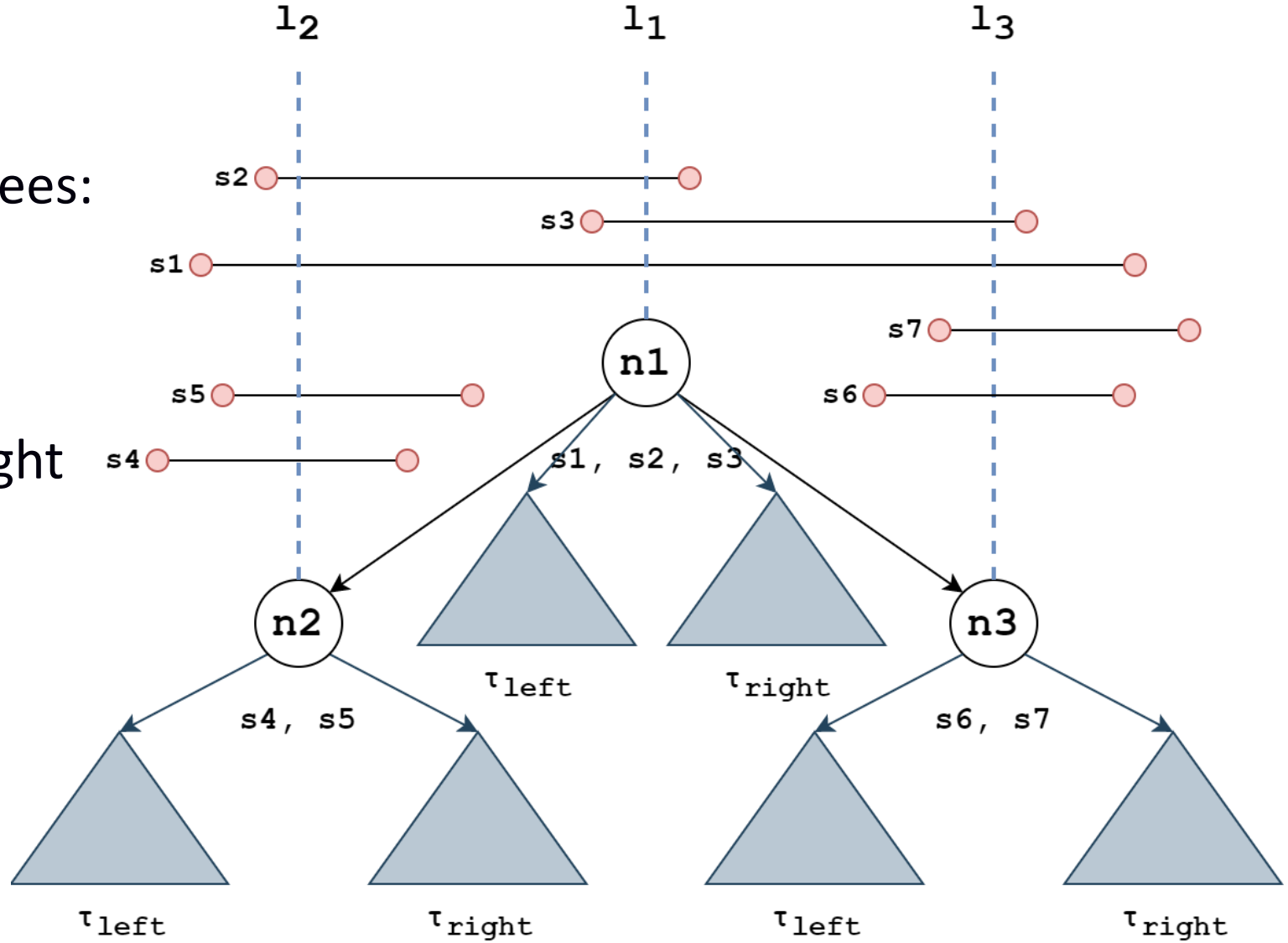
Interval tree (2)

- s_1, s_2, and s_3 don't belong **exclusively** to the n_2 or n_3 intervals
- They belong exclusively to n_1
- Both endpoints must be in the interval



Interval tree (3)

- Each node n_i has 2 range trees: τ_{left} and τ_{right}
  - Or any other trees
- They contain the left and right endpoints for the intervals
- e.g. n_2 has:
  - τ_{left}: $s_{4_{left}}, s_{5_{left}}$
  - τ_{right}: $s_{4_{right}}, s_{5_{right}}$



Interval tree – creation (1)

```
fun create_interval_tree(intervals):
    if intervals.count() == 0:
        Node() # empty
    else:
        curr_median = median([i.endpoints() for i in intervals])
        intervals_l = [i for i in intervals
                       if i.endp_l < curr_median and i.endp_r < curr_median] # sort
        intervals_r = [i for i in intervals
                       if i.endp_l > curr_median and i.endp_r > curr_median] # sort
        node_intervals = intervals \ (intervals_l + intervals_r) # leftovers
        curr_node = Node(curr_median)
        curr_node.ch_l = create_interval_tree(intervals_l)
        curr_node.ch_r = create_interval_tree(intervals_r)
        if node_intervals.count() != 0:
            endp_l = [i.endp_l for i in node_intervals if i.endp_l <= curr_median]
            endp_r = [i.endp_r for i in node_intervals if i.endp_r > curr_median]
            curr_node.tau_l = create_2d_range_tree(endp_l) # range tree
            curr_node.tau_r = create_2d_range_tree(endp_r) # range tree
        curr_node
```

Interval tree – creation (2)

- Sorting: $O(n \log n)$
- 2 range trees per node: $O\big((n \log n)(2 \log n)\big)$
- Completely: $O(n \log^2 n)$

Interval tree – search (1)

- Query window: $W_q = [q_{x_1}, q_{x_2}] \times [q_{y_1}, q_{y_2}]$
- Similar to range tree search
- 3 cases:

if $med(n_i) \in [q_{x_1}, q_{x_2}]$

-> Go to both τ_{left} and τ_{right}

if $med(n_i) > [q_{x_1}, q_{x_2}]$

-> Search τ_{right} for intervals right-ending in W_q

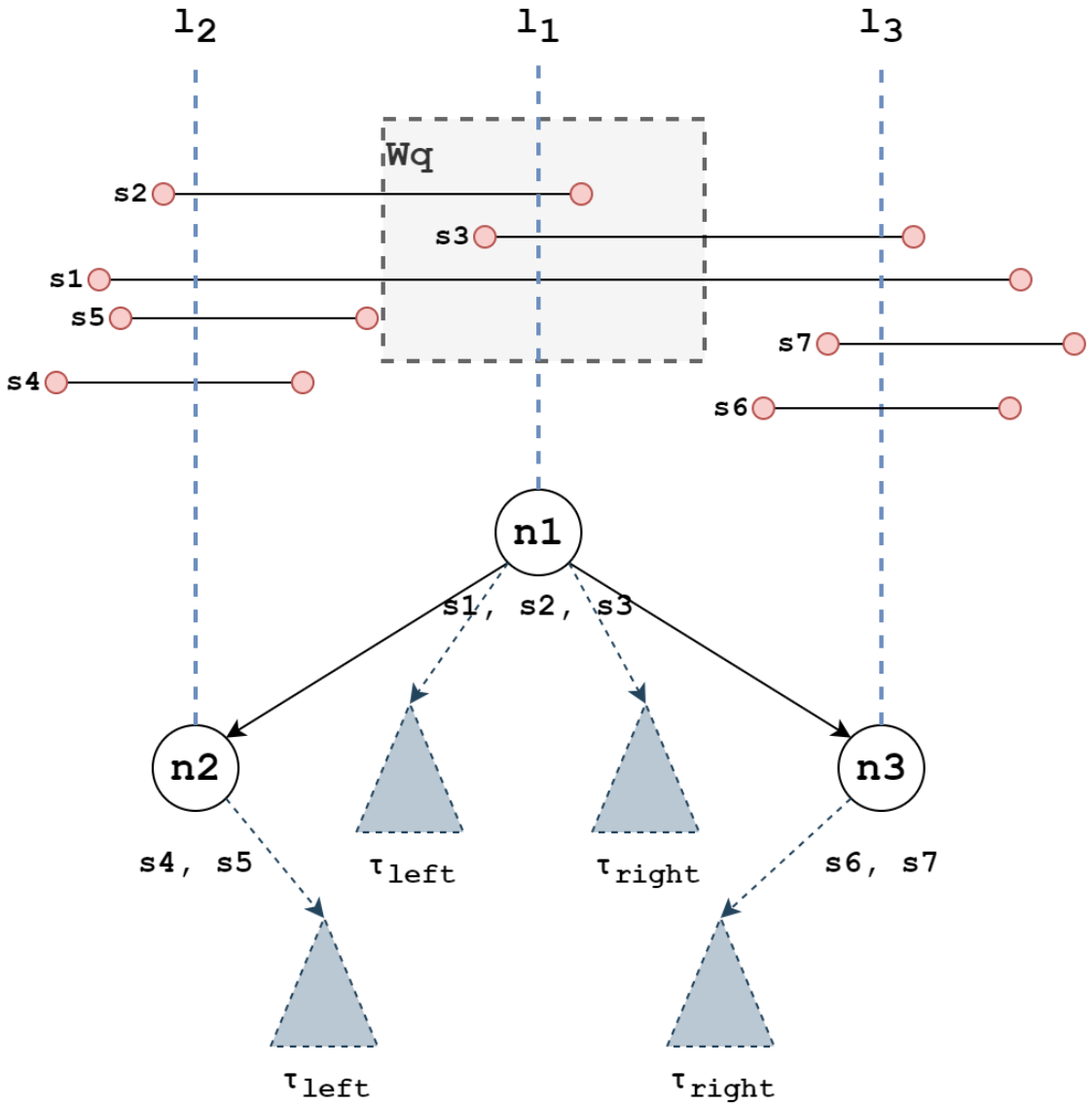
-> Skip τ_{left}

if $med(n_i) \leq [q_{x_1}, q_{x_2}]$

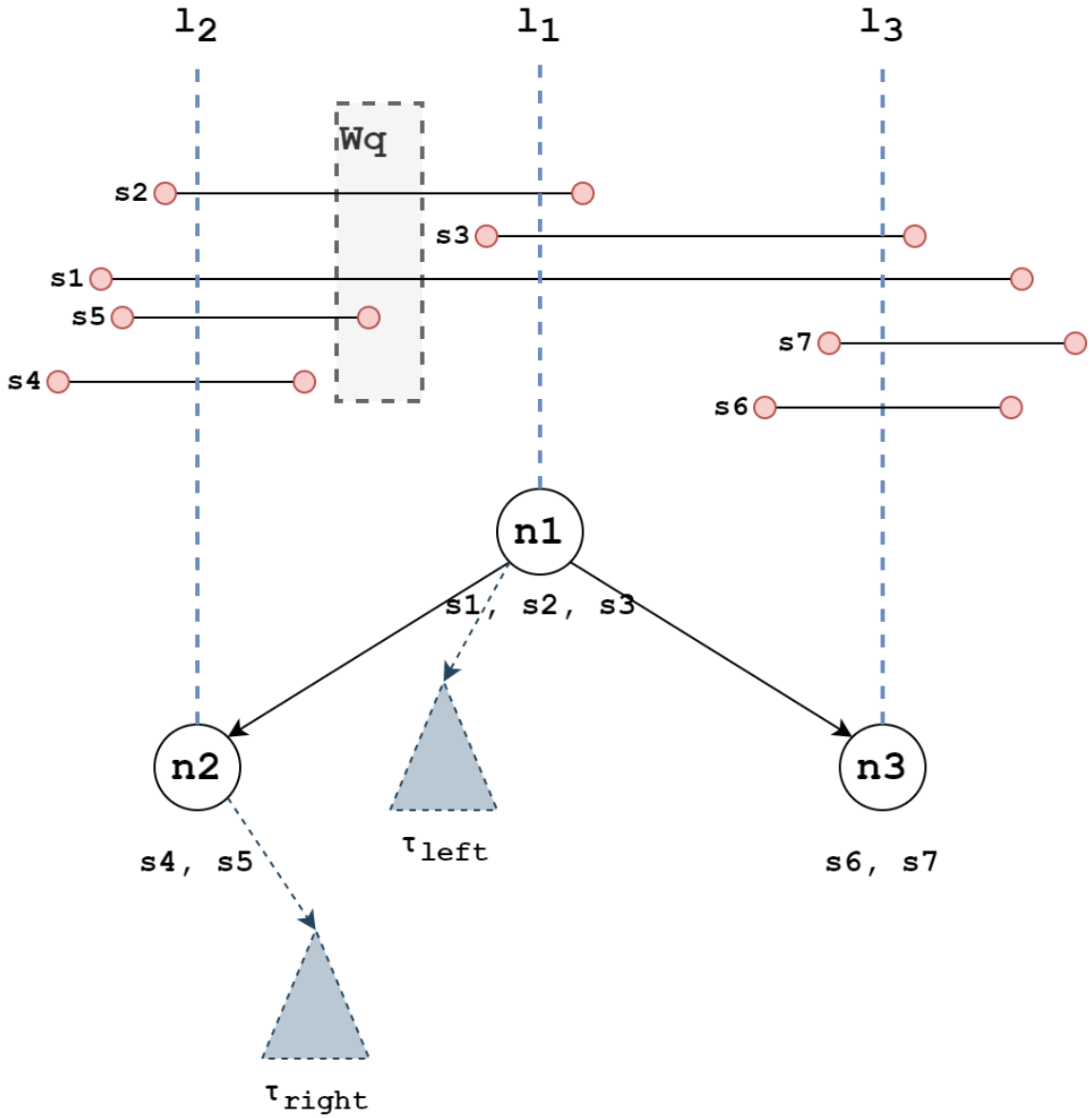
-> Search τ_{left} for intervals left-ending in W_q

-> Skip τ_{right}

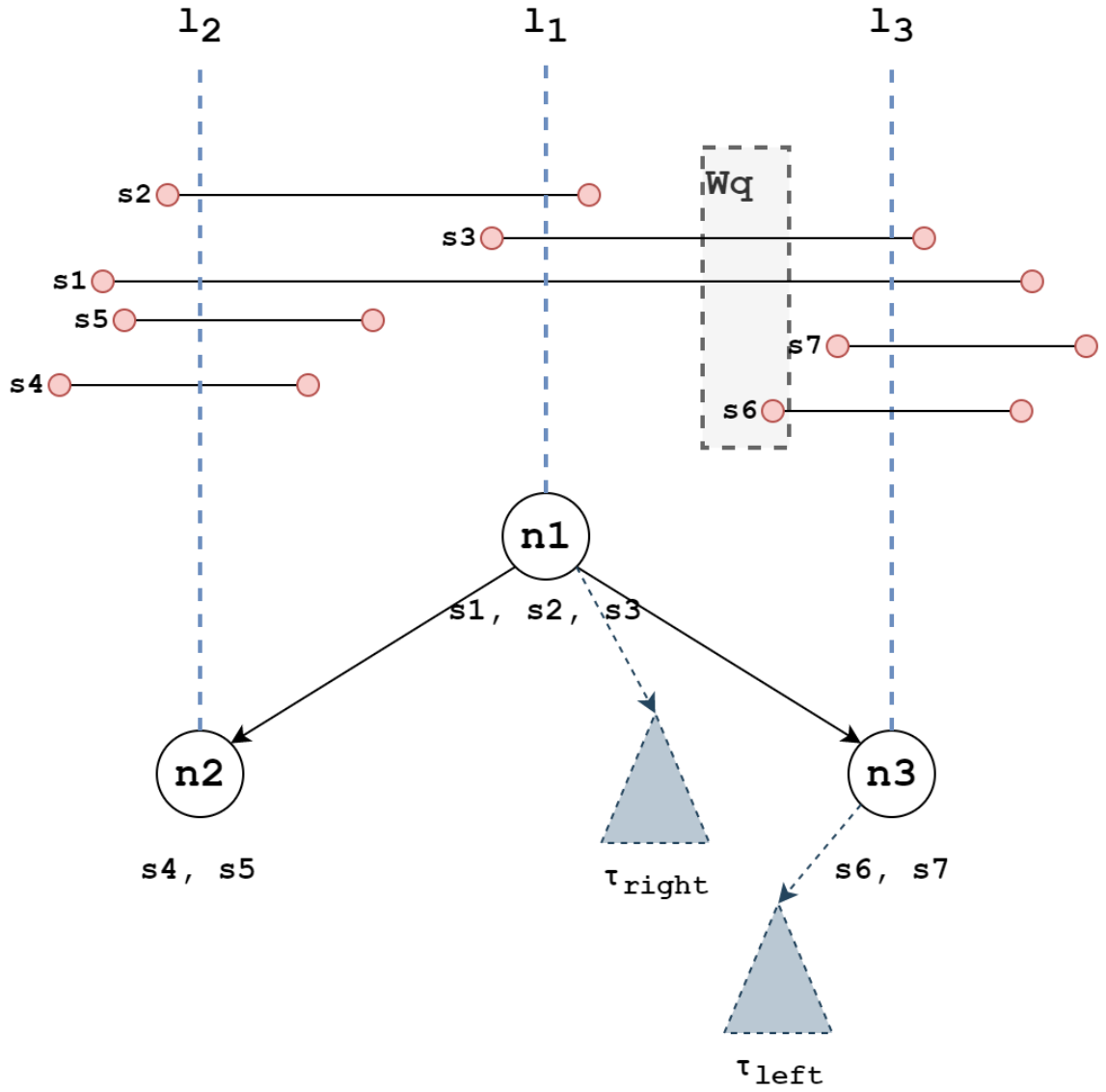
Interval tree – search (2)



Interval tree – search (3)

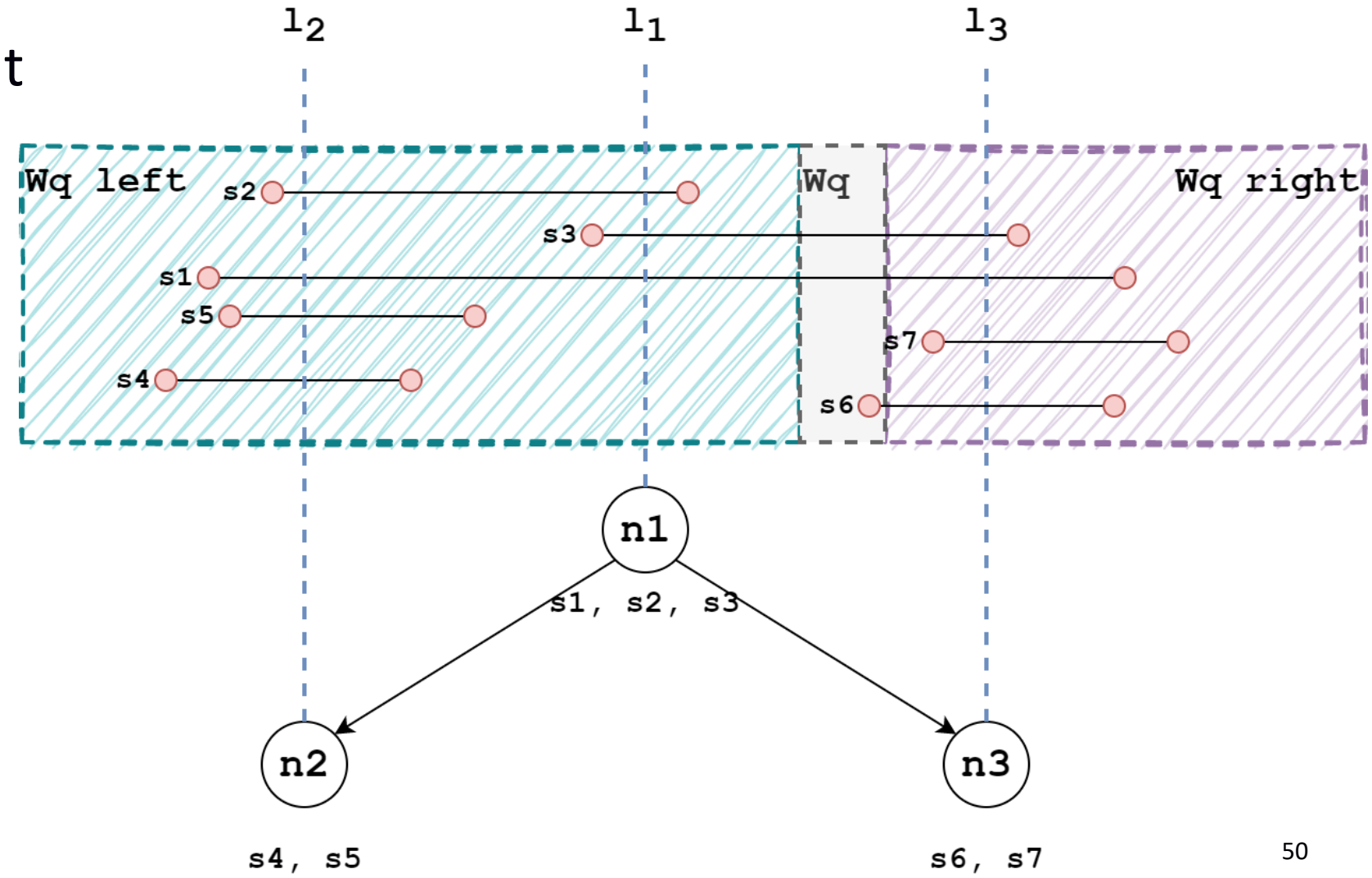


Interval tree – search (4)



Interval tree – search (5)

- Full intersect without endpoints?
- Search to infinity!



Interval tree – search (6)

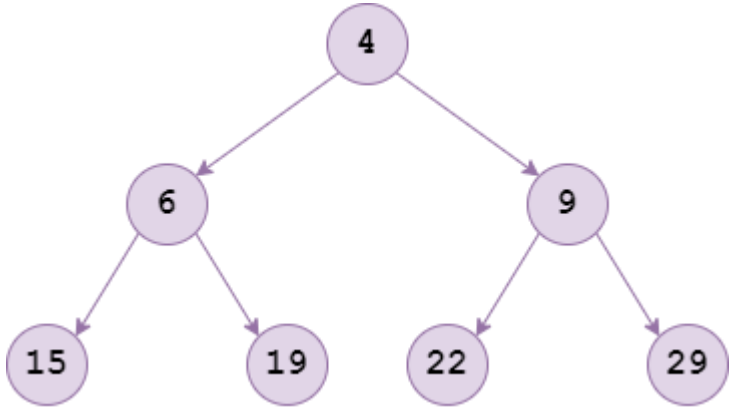
- 1st level tree: $(\log n)$
- 2nd level 2D range trees: $(2 \log^2 n)$
- Complete time complexity: $O\big((2 \log^2 n)(\log n)\big) = O(\log^3 n)$

- Not that efficient?

Priority tree (1)

- Use a heap
  - Sorted structure where the root is the greatest or least valued member

- Use an ascending heap
  - Sort the elements ascendingly
  - First element is the root
  - Left child of i^{th} node is at $(2i + 1)^{th}$ index.
  - Right child of i^{th} node is at $(2i + 2)^{th}$ index.
  - Parent of i^{th} node is at $(i-1)/2$ index.



Priority tree (2)

- Sort the points by p_x
- Store the points in a heap

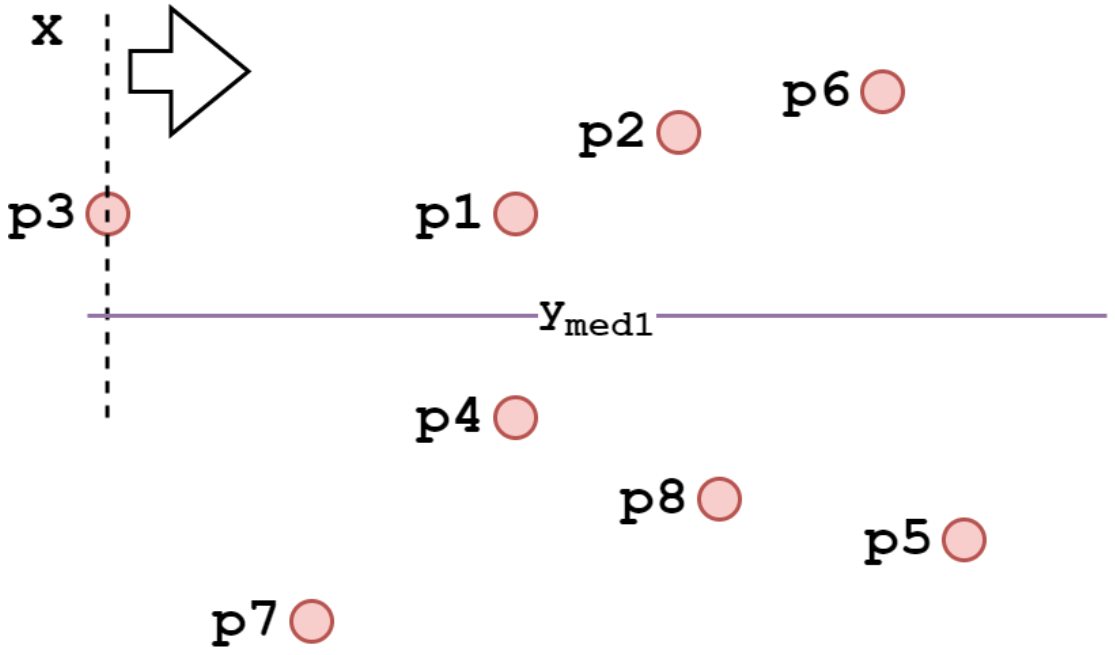
- Better searchability for $(-\infty, q_{x_2}]$ and $[q_{x_1}, +\infty)$
  - Depending on an ascending or descending heap
  - We hit "$-\infty$" or "$+\infty$" first – at the root

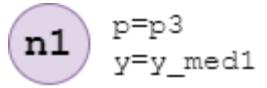
Priority tree – creation (1)

```
fun create_priority_tree(points):
    p_min = min_by_x(points)
    node = Node()
    node.point = p_min
    if points - p_min != []:
        y_med = median(points – p_min)
        points_left = [for p in points if p <= y_med]
        points_right = [for p in points if p > y_med]
        node.y = y_med
        if points_left != []:
            node.left = create_priority_tree(points_left)
        if points_right != []:
            node.right = create_priority_tree(points_right)
    node
```

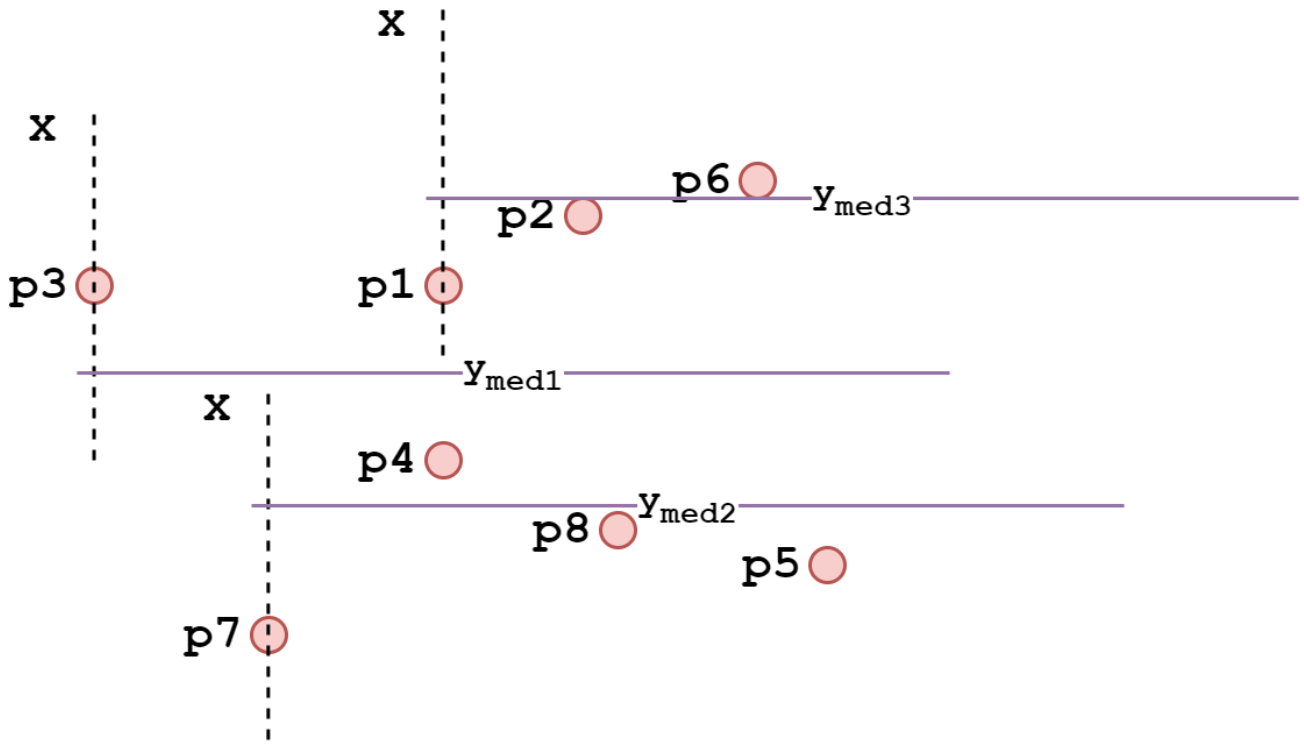
Priority tree – creation (2)

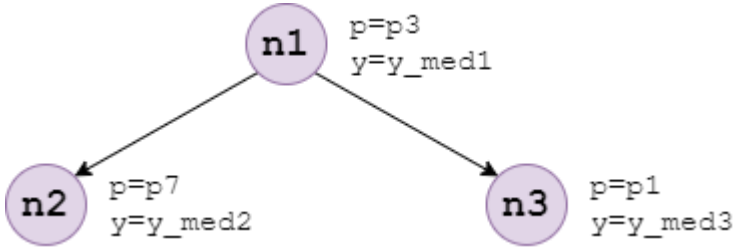
- We split the y-axis for every point along the points sorted by x



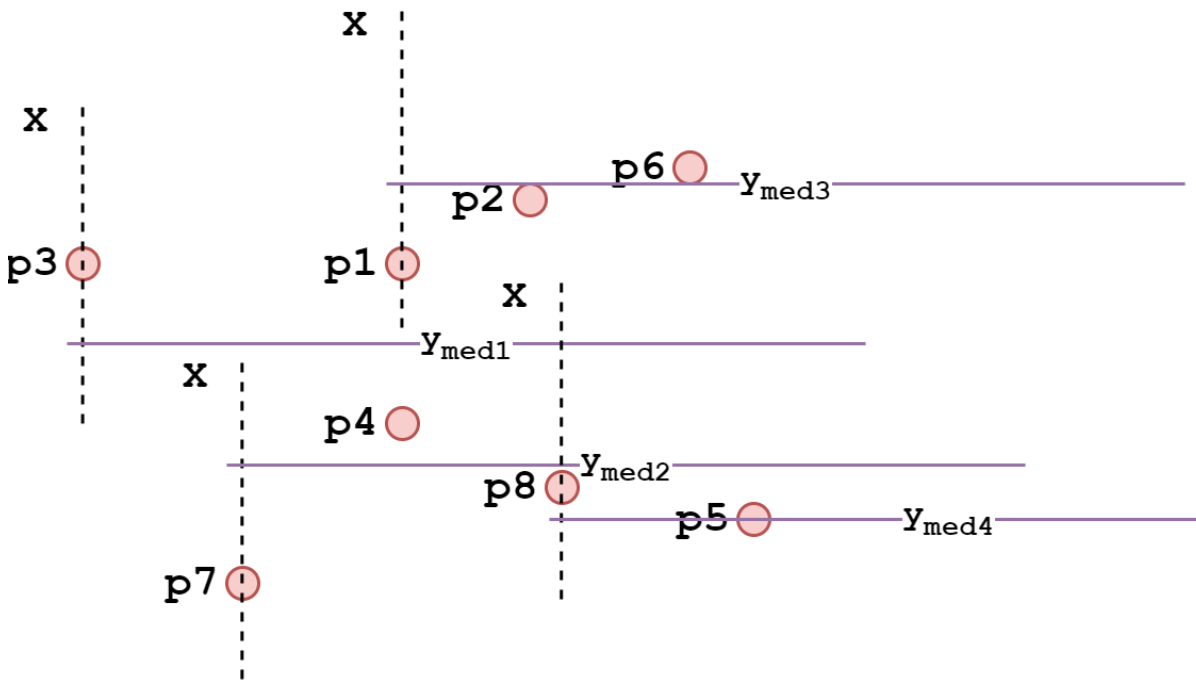


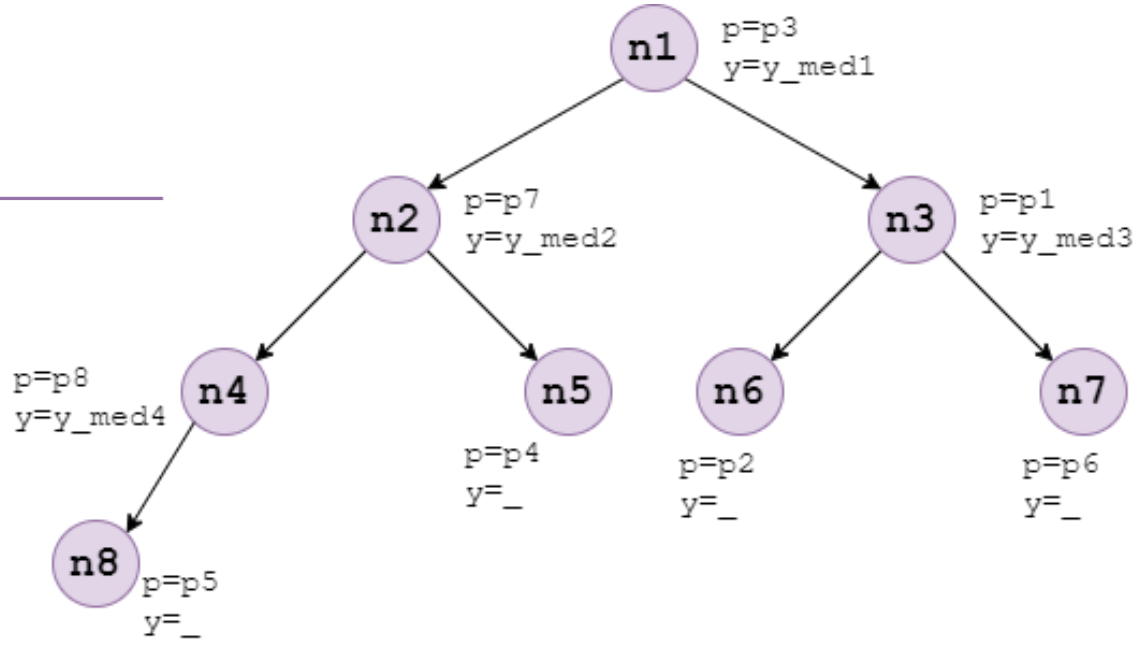
Priority tree – creation (3)





Priority tree – creation (4)





Priority tree – search (1)

- The main problem in the priority tree:

```
node.point.y != node.y_med
```

- They can differ greatly, meaning:

```
node.point.y in [q_y1, q_y2]
node.y_med not in [q_y1, q_y2]
```

- Path finding uses binary tree part of the priority tree: `node.y_med`
- Heap uses points stored in the tree nodes: `node.point`

Priority tree – search (2)

- The main problem in the priority tree:

```
node.point.y != node.y_med
```

- They can differ greatly, meaning:

```
node.point.y in [q_y1, q_y2]
node.y_med not in [q_y1, q_y2]
```

- Path finding uses binary tree part of the priority tree: `node.y_med`
- Heap uses points stored in the tree nodes: `node.point`

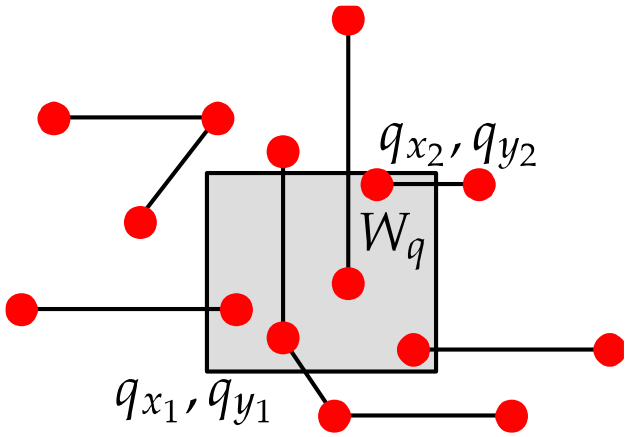
Priority tree – search (3)

- Search for the splitting node by `node.y_med` in $[q_{y1}, q_{y2}]$
  - 2 vertical tree traverses
- Continually check that `node.point.x` $\leq q_{x2}$

- We are checking for points in $(-\infty, q_{x_2}] \times [q_{y_1}, q_{y_2}]$
  - Ascending heap

Line segment search (1)

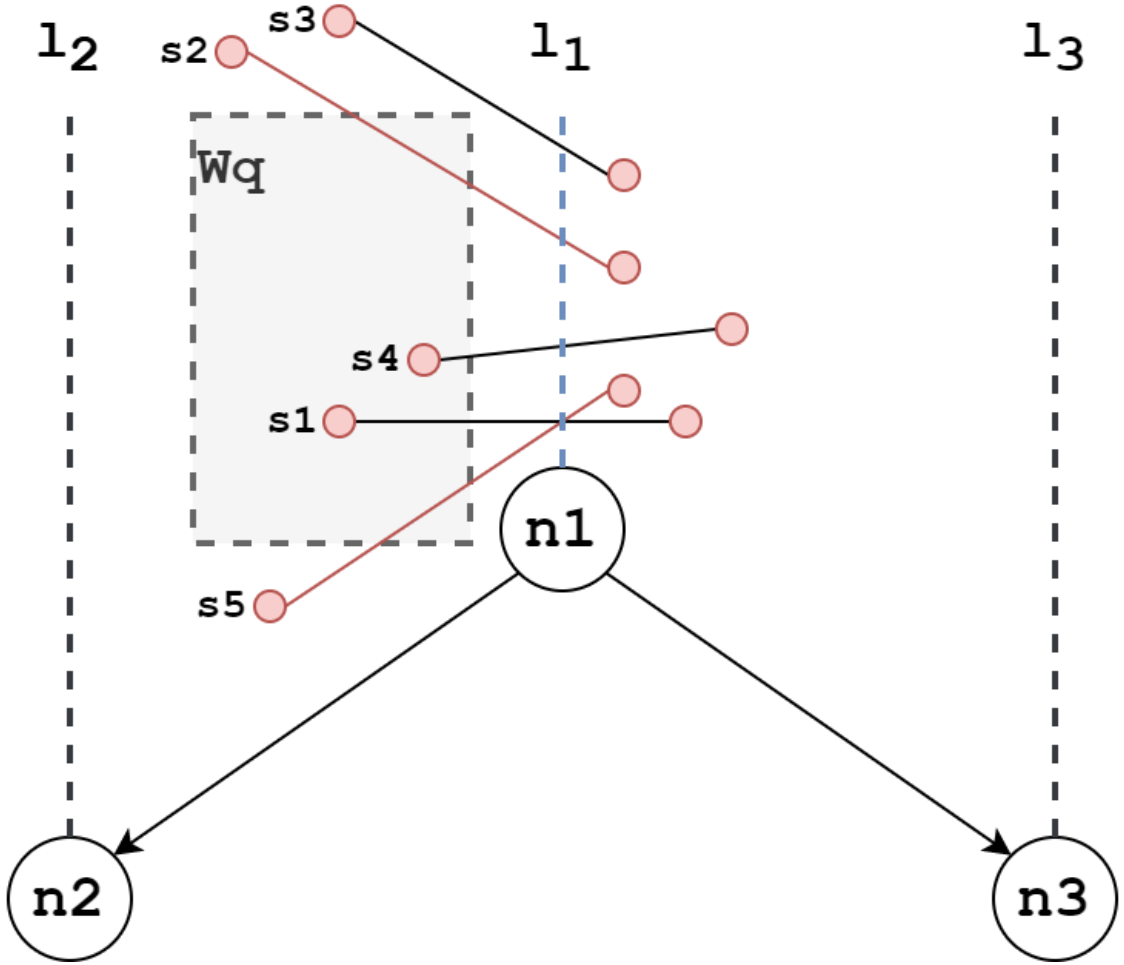
- Line segment set: $S = \left\{ s_i = \overline{(x_{i_1}, y_{i_1})(x_{i_2}, y_{i_2})} : (x_{i_k}, y_{i_k}) \in \mathbb{R}^2 \right\}$
- Search area: $W_q = [q_{x_1}, q_{x_2}] \times [q_{y_1}, q_{y_2}]$



Line segment search (2)

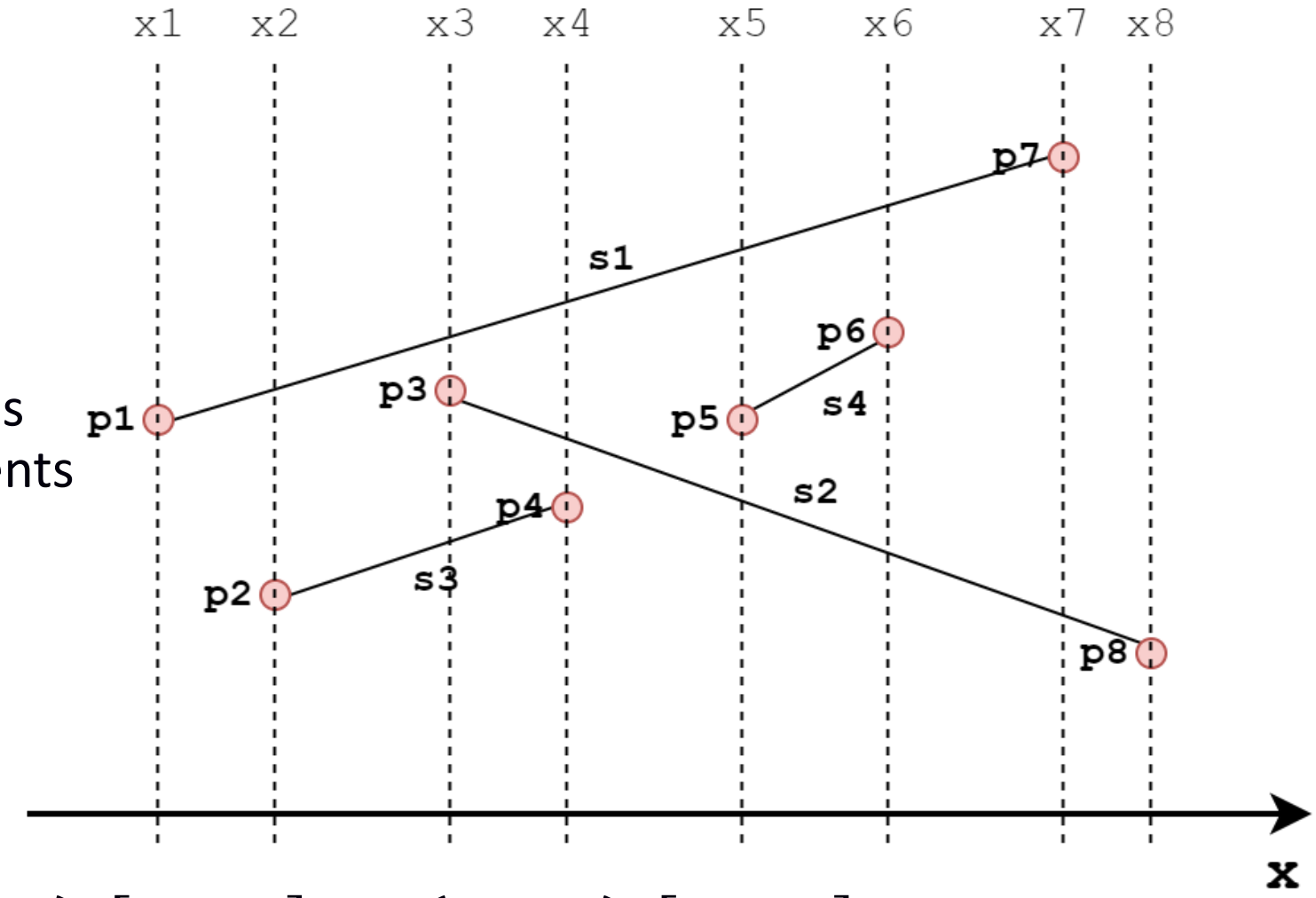
- Interval tree?
- What's wrong with **`s2`** and **`s5`**?

- False negative



Line segment search (3)

- Locus approach
- Mark elementary intervals
  - Each interval continually has the same number of segments



$$(-\infty, x_1), [x_1, x_1], (x_1, x_2), [x_2, x_2], \ldots, (x_7, x_8), [x_8, x_8], (x_8, \infty)$$

Line segment search (4)

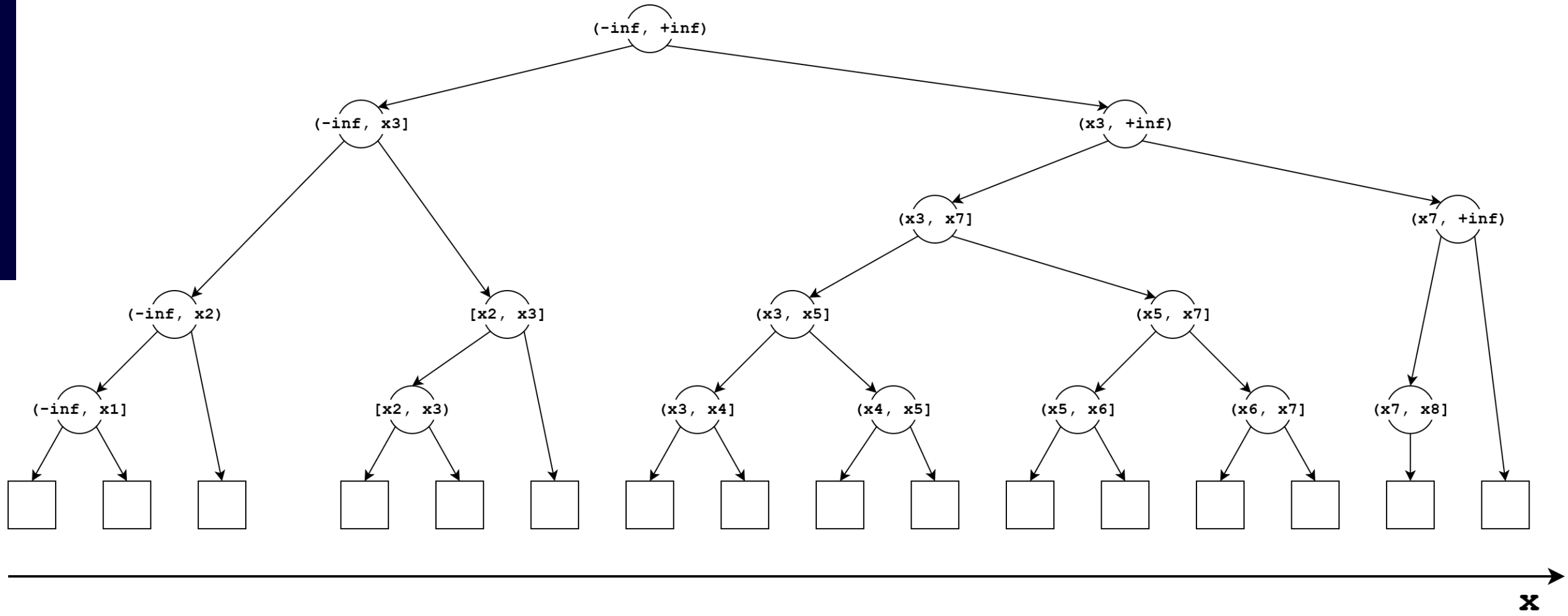
- Each interval becomes a tree node in a balanced index tree
  - We could use the B+ tree
- Leaves are sets of line segments passing through the interval
  - A line segment can be in multiple leaves (intervals)

- Parent nodes aggregate child node intervals:

$$I(n) = I(n_L) \cup I(n_R)$$

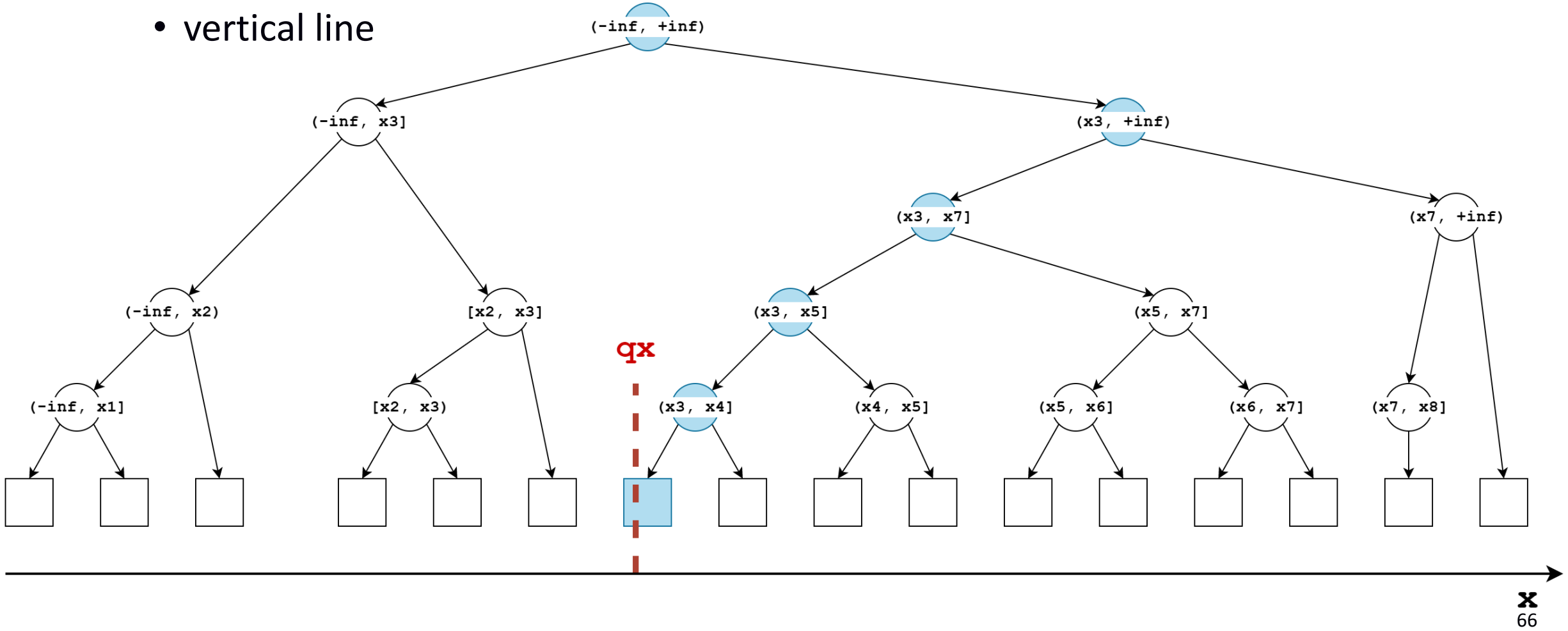
- Search complexity: $O(k + \log^2 n)$

Line segment search (5) – **segment tree**



Line segment search (6)

- We can search for segments going through $q_x \times [q_{y1}, q_{y2}]$
  - vertical line



Line segment search (7)

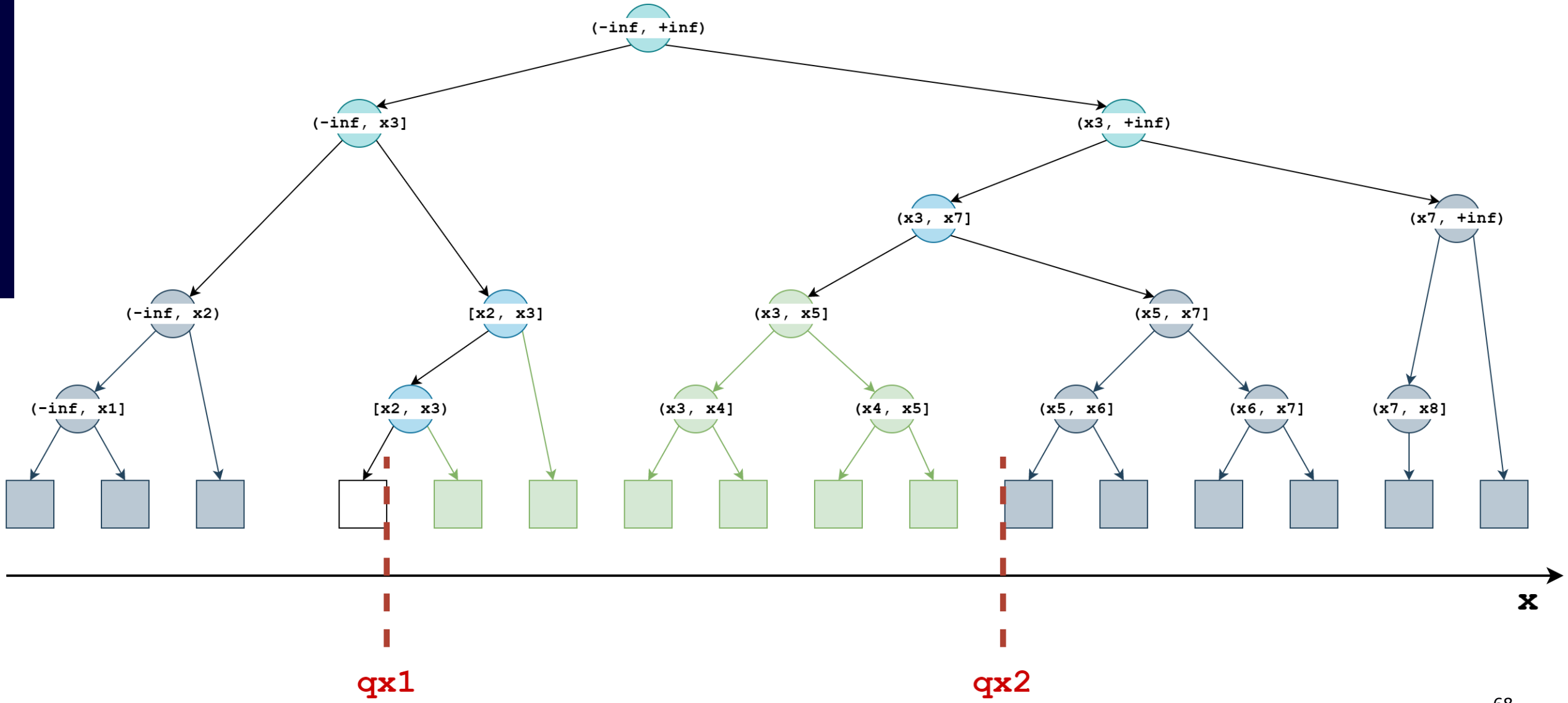
- We can search for segments going through

$$W_q = [q_{x_1}, q_{x_2}] \times [q_{y_1}, q_{y_2}]$$

- Again we are searching for n_{split}
- We proon or skip or traverse down the tree

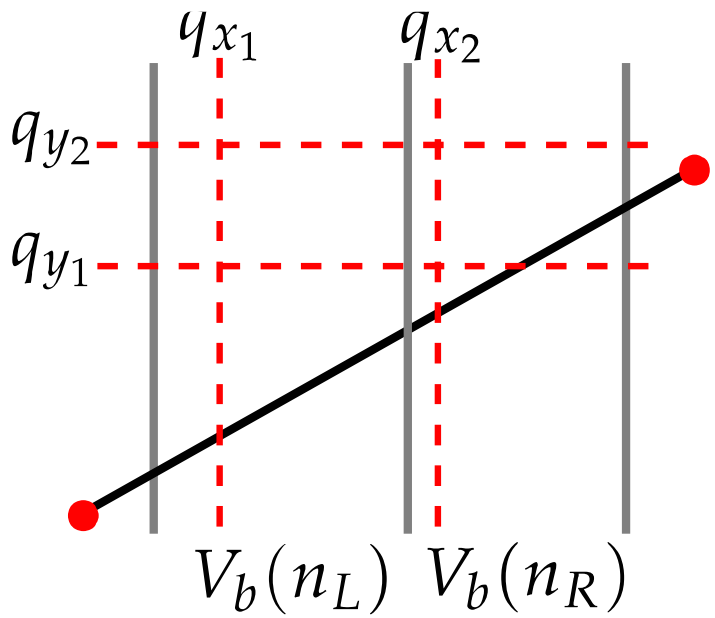
- The result is a set of vertical blocks (unbounded by y)

Line segment search (8)



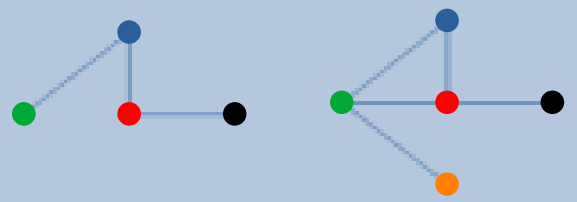
Line segment search (9)

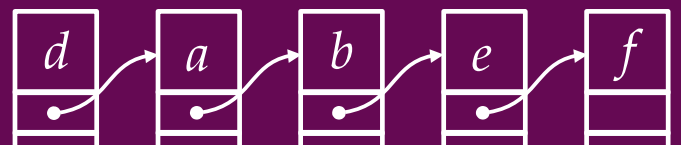
- What about $[q_{y_1}, q_{y_2}]$?
- Another embedded tree?
  - Range search?
- Crossed lines?
  - Problem for verticality
- Out of bounds segment?
  - Line segment does enter vertical box, but "after" the search box bounds
  - Possible *false positive*

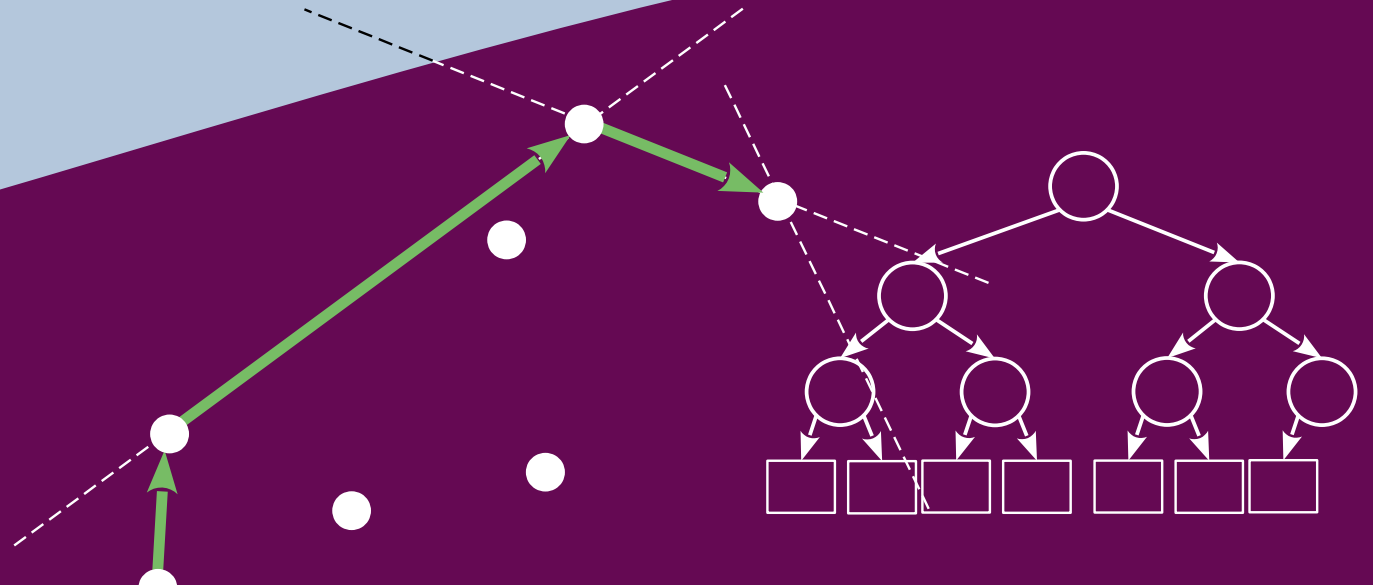


Napredni algoritmi i strukture podataka

Tjedan 5: Linearno programiranje







Što je linearni program (LP)?

- Dosta **generalna** klasa problema koja se može rješavati efikasno

- Brojne primjene u industriji
  - Lanci nabave
  - Raspoređivanje
  - Optimizacije u električnim mrežama

- Spada u kategoriju **konveksnih optimizacijskih problema**
  - LP je prva podkategorija koja je efikasno riješena (cca 1940tih)
  - Ostale kategorije su slijedile kasnije, otvoreno područje istraživanja

Literatura:

- Krleža, Brčić: „Advanced Algorithms and Data Structures”, Lecture book. (4. poglavlje)
- D. Bertsimas, and J. Tsitsiklis.: Introduction to linear optimization. Athena Scientific, 1997. (poglavlja 3.1-3.3, 3.5)

Što je linearni program (LP)?

- Rješavanje linearnih programa je dignuto na razinu industrijske pouzdanosti
  - Pouzdani i brzi alati
    - Gurobi – trenutno najbrži rješavač
    - Python – **scipy.optimize.linprog**
    - Čak integrirani u Excel

- Bitni i za dizajn i analizu algoritama, npr.
  - Algoritmi nad grafovima
  - Približni algoritmi

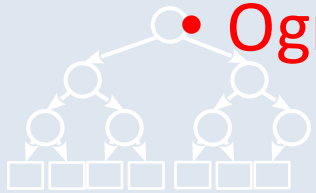
Linearni program (LP)?! Generalno…

$$\begin{aligned} &\text{minimizirati} && \mathbf{c}^T\mathbf{x} \\ &\text{uz uvjet} && \mathbf{A}\mathbf{x} \leq \mathbf{b} \\ & && \mathbf{D}\mathbf{x} = \mathbf{e} \end{aligned}$$

pri čemu je: $\mathbf{x} \in R^n, \mathbf{c} \in R^n, \mathbf{b} \in R^m, \mathbf{e} \in R^k,$

matrica $\mathbf{A} \in R^{mxn}$, matrica $\mathbf{D} \in R^{kxn}$

- Ciljna funkcija *f* (*objective function*, *cost function*)
- Varijable odluke ***x*** (*control*, *structural*, *decision variables*)
- Ograničenja sa parametrima ***A,b,D,e*** (*constraints*)



Primjer I – humanitarni transportni problem

- Pfizer proizvodi cjepiva za COVID-19 te ima proizvodne pogone u tri mjesta T1...3 s raspoloživim kapacitetima zadanima u tablici. Naručitelji su iz četiri mjesta O1...4 sa potrebama zadanima u zadnjem retku tablice. Jedinični transportni troškovi za sve kombinacije proizvodnih pogona i naručitelja su navedeni u tablici.
- Kako na najefikasniji način zadovoljiti potrebe naručitelja?

Transportni troškovi

	O1	O2	O3	O4	Kap.
T1	10	9	14	8	432
T2	7	11	9	11	138
T3	8	12	12	9	35
Nar.	500	200	115	100	

Primjer I – humanitarni transportni problem

x_{ij} - količina isporučenu iz *i*-te tvornice *j*-tom naručitelju

c_{ij} - trošak transporta po jedinici između *i* i *j*

$$\min\left(\sum_{i,j} x_{ij} c_{ij}\right)$$

uz uvjete $\sum_{j} x_{ij} \leq s_i$; $i = 1, 2, 3$

$$\sum_{i} x_{ij} = \mathrm{d}_j \quad ; \quad j = 1, 2, 3, 4$$

$x_{ij} \geq 0$

Primjer II – optimalno uparivanje u online datingu

Na dating siteu u hetero rubrici postoji M muškaraca i F žena. Na temelju popunjenih upitnika raznim modelima su izračunate kompatibilnosti za sve potencijalne parove. Site želi upariti sve u parove da bi ukupna suma normaliziranih kompatibilnosti bila što veća (utilitarizam).

kompatibilnosti

	M1	M2	M3	M4
F1	9	1	8	7
F2	1	2	1	7
F3	8	2	4	8
F4	2	4	6	4

Kombinatorni problem

Naivno rješenje: ispitati sve kombinacije F-M. Treba ispitati F! kombinacija (ako je F=M)

eharmony

Primjer II – optimalno uparivanje u online datingu

x_{ij} - 1 ako je ***i*** uparen sa ***j***, 0 inače

k_{ij} – kompatibilnost za uparivanje *(i,j)*

$$max\left(\sum_{i,j} x_{ij}k_{ij}\right)$$

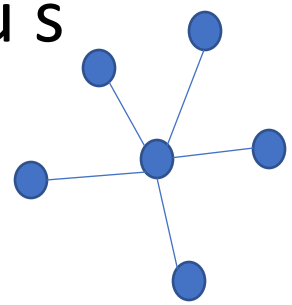
uz uvjete $\sum_{j} x_{ij} = 1, \forall i = 1, \dots, F$

$$\sum_{i} x_{ij} = 1, \forall j = 1, \dots, M$$

$x_{ij} \geq 0$

Primjer III – bežične raspodijeljene mreže

Centralna postaja treba ostvariti pouzdanu bežičnu komunikaciju s *N* raspodijeljenih senzora za praćenje klimatskih promjena.



Senzori se napajaju Sunčevom energijom i važno je smanjiti njihovu potrošnju. Signal *i*-tog senzora do centralne postaje stiže ***prigušen.*** Ako *i*-ta senzor emitira snagom p_i, centralna postaja prima signal snage $\lambda_i \cdot p_i$ ($\lambda<1$). Tijekom komunikacije s *i*-tim senzorom, signali svih drugih senzora koji dolaze u centralnu postaju predstavljaju smetnju i komunikacija je moguća samo ako je omjer signal/šum najmanje ρ_i.

Kolike trebaju biti emitivne snage p_i senzora kako bi se ostvarila pouzdana komunikacija uz najmanju moguću potrošnju energije?

Primjer III – bežične raspodijeljene mreže

Ukupna potrošnja je proporcionalna ukupnoj snazi pa ćemo to minimizirati:

$$\min\left(\sum_{i=1}^{N} p_i\right)$$

uz uvjete

$$\frac{\lambda_i p_i}{\sum_{j\neq i}\lambda_j p_j} \geq \rho_i \quad ; \quad i, j = 1, \ldots, N$$

$$p_k \geq 0 \quad ; \quad k = 1, \ldots, N \quad .$$

Budući da uvjetne (ne)jednadžbe moraju biti linearne po p_k, prevodimo ih u oblik

$$\lambda_i p_i - \rho_i \sum_{j\neq i}\lambda_j p_j \geq 0$$

LP formulacije

- Općenita formulacija je „neuredna”

- Dvije formulacije kojima težimo radi lakšeg rješavanja i pisanja algoritama
  - **Kanonska forma LP** – idealna za geometrijsku perspektivu
  - **Standardna forma LP** – idealna za algebarsku perspektivu

- Svi ostali LP se mogu prevesti u obje forme*

*pročitajte o transformacijama u skripti

Kanonska forma LP

$$\begin{aligned} &\text{minimizirati} && \mathbf{c}^T\mathbf{x} \\ &\text{uz uvjet} && \mathbf{Ax} \leq \mathbf{b} \\ & && \mathbf{x} \geq \mathbf{0} \end{aligned}$$

pri čemu je: $\mathbf{c} \in R^n$, $\mathbf{b} \in R^m$, matrica $\mathbf{A} \in R^{mxn}$.

Standardna forma LP

$$\begin{aligned} &\text{minimizirati} && \mathbf{c}^T\mathbf{x} \\ &\text{uz uvjet} && \mathbf{A}\mathbf{x} = \mathbf{b} \\ & && \mathbf{x} \geq \mathbf{0} \end{aligned}$$

pri čemu je: $\mathbf{c} \in R^n$, $\mathbf{b} \in R^m$ i $\mathbf{b} \geq 0$, matrica $\mathbf{A} \in R^{mxn}$

Nije dio definicije, ali pretpostavljat ćemo u sklopu predavanja da je $rang(\mathbf{A}) = m$ i $m < n$. Ako je rang manji, linearno-zavisna ograničenja se mogu ukloniti.

Grafička metoda - primjer

max $x_1 + 5 \cdot x_2$

uz uvjete $\begin{bmatrix} 5 & 6 \\ 3 & 2 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \leq \begin{bmatrix} 30 \\ 12 \end{bmatrix}$

$x \geq 0$

GeoGebra – interaktivna geometrija

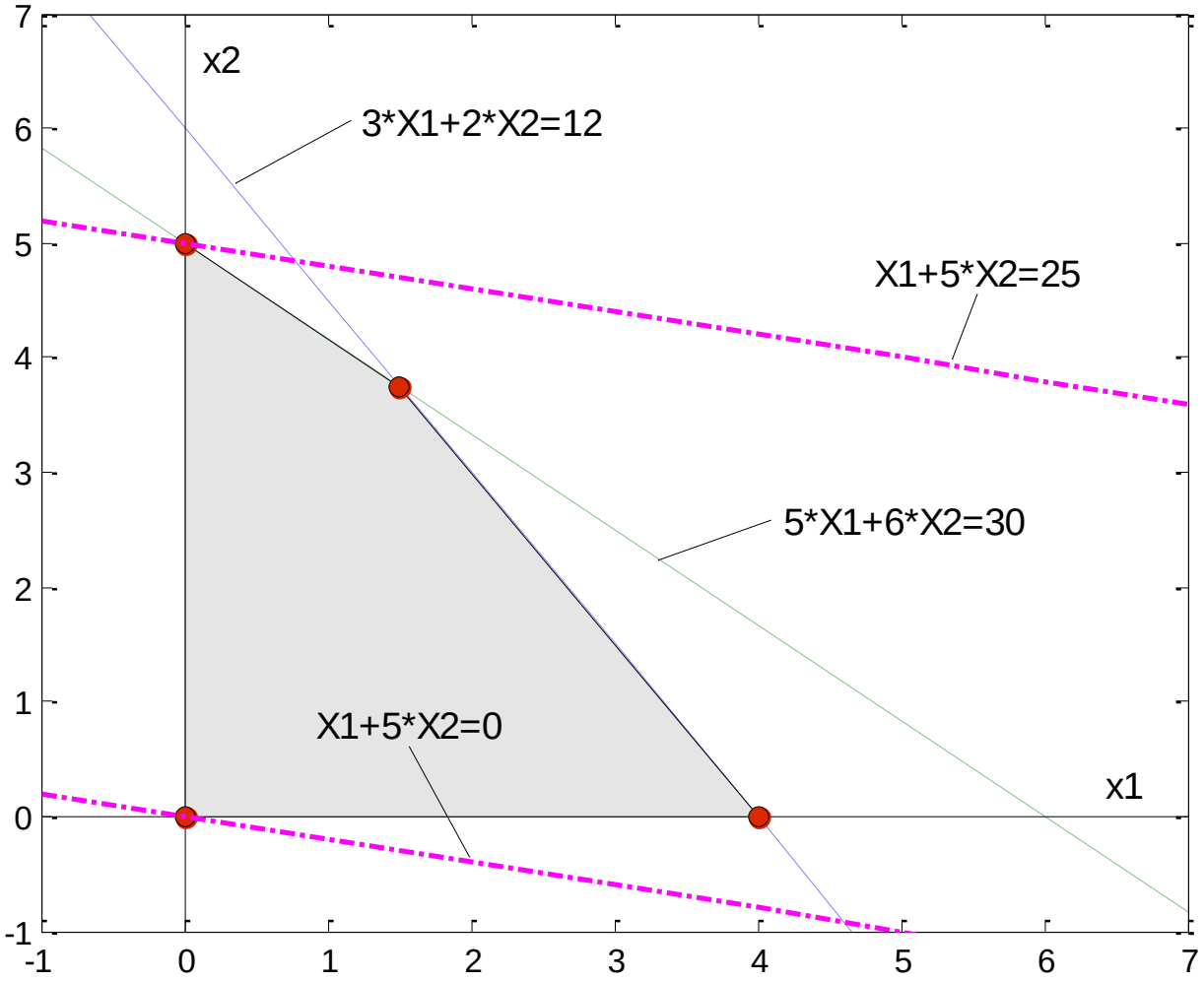
Grafička metoda - primjer

max $x_1 + 5 \cdot x_2$

uz uvjete $\begin{bmatrix} 5 & 6 \\ 3 & 2 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \leq \begin{bmatrix} 30 \\ 12 \end{bmatrix}$

$\mathbf{x \geq 0}$

Rješenje je sjecište pravca $x_1 + 5 \cdot x_2 = f$ s prostorom svih mogućih rješenja (sivi poligon na slici) za koje je f najveća.



Rješenje

$x=[0, 5]^T$

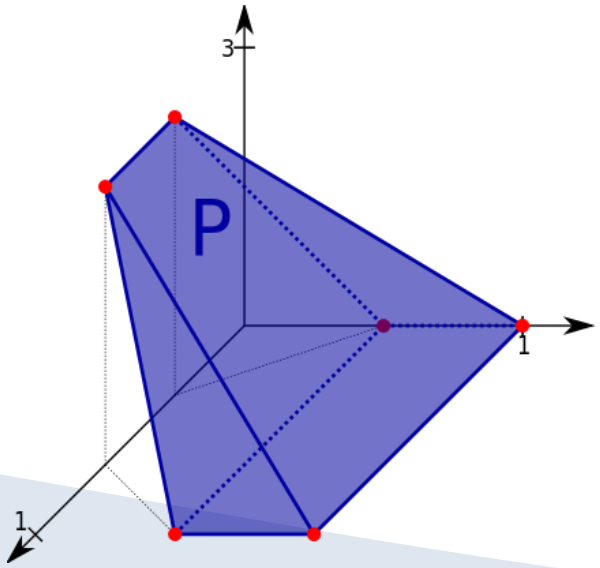
$f_{max} = 25$

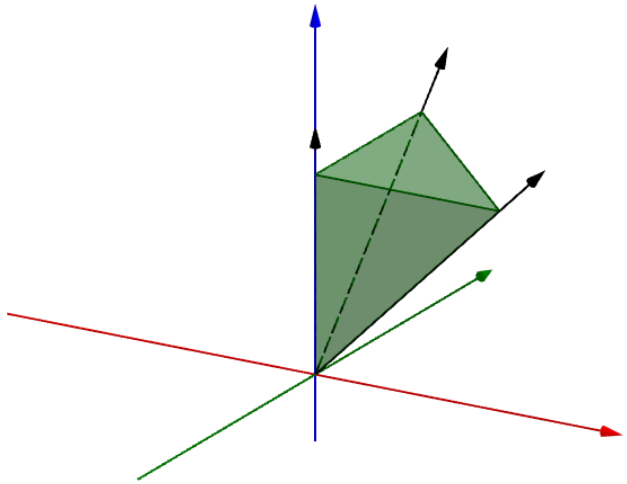
Geometrijska analiza

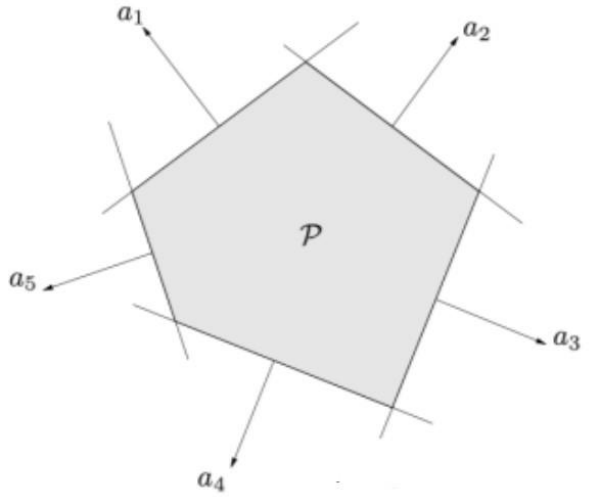
Definicija. ***Konveksni politop*** u n-dimenzionalnom prostoru jest skup vektora (točaka):

$$\{x \in \mathbb{R}^n \mid Ax \leq b\}$$

- Ograničenja linearnih programa!







Geometrijska analiza

Tijela opisana ograničenjima u LP su konveksni politopi P

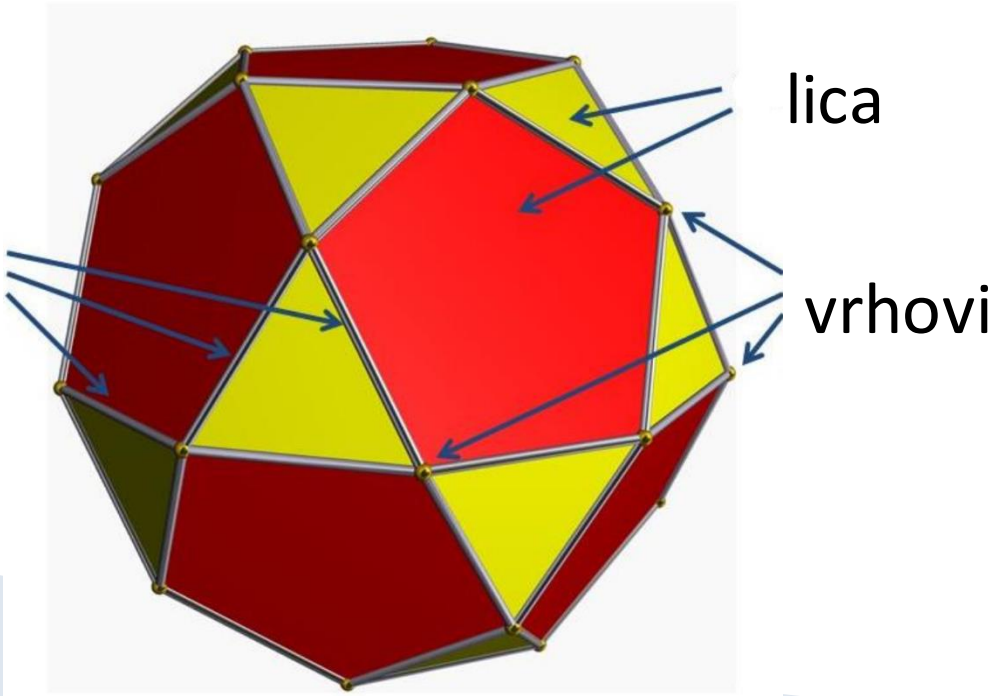
$$P = \{x \in \mathbb{R}^n \mid Ax \leq b\}$$

https://polyhedra.tessera.li/

Geometrijska analiza

Tijela opisana ograničenjima u LP su konveksni politopi P

$$P = \{x \in \mathbb{R}^n \mid Ax \leq b\}$$



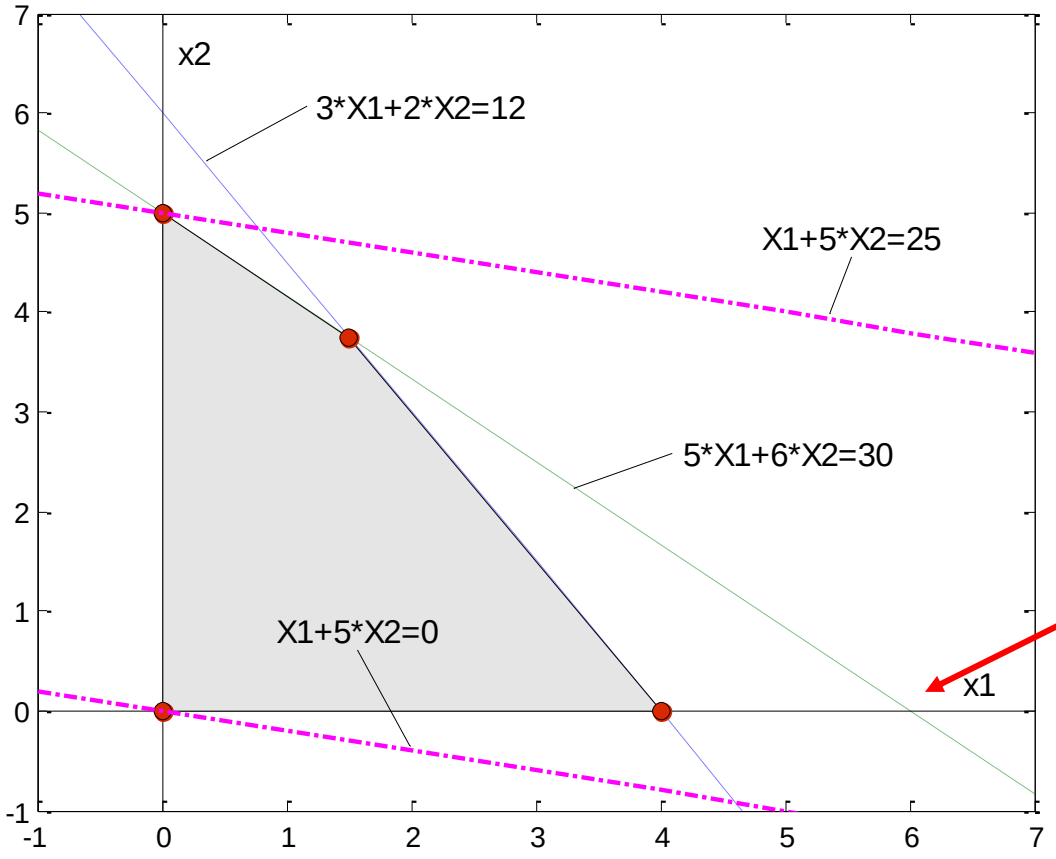
Definicija. Aktivno ograničenje u nekoj točki x je svako ograničenje koje je ispunjeno sa jednakosti u toj točki x.

Definicija. U ***n***-dimenzionalnom prostoru, **vrh politopa** je definiran kao presjecište barem ***n*** aktivnih linearno nezavisnih ograničenja pri čemu su ostala ograničenja zadovoljena.

Geometrijska analiza

Tijela opisana ograničenjima u LP su konveksni politopi P

$$P = \{x \in \mathbb{R}^n \mid Ax \leq b\}$$



Definicija. U ***n***-dimenzionalnom prostoru, **vrh politopa** je definiran kao presjecište barem ***n*** aktivnih linearno nezavisnih ograničenja pri čemu su ostala ograničenja zadovoljena.

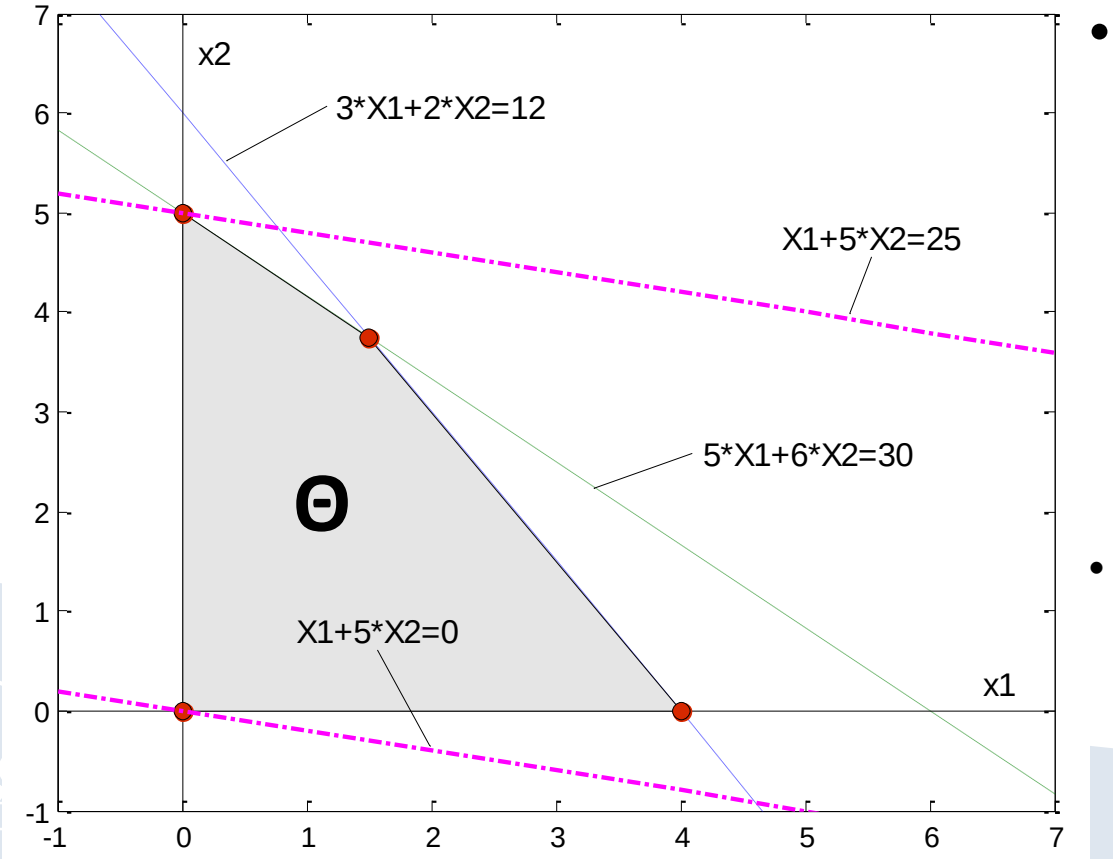
Oprez! Nije svako presjecište n aktivnih ograničenja **vrh**! Neaktivna ograničenja moraju biti ispoštovana da bismo bili u vrhu.

Susjedi?

Geometrijska analiza

- **Definicija.** Skup Θ je **konveksni skup** ako sadrži sve točke ravne spojnice između bilo kog para točaka iz skupa Θ.

  $\forall x,y \in \Theta, \forall \alpha \in (0,1): \alpha x+(1-\alpha)y \in \Theta$

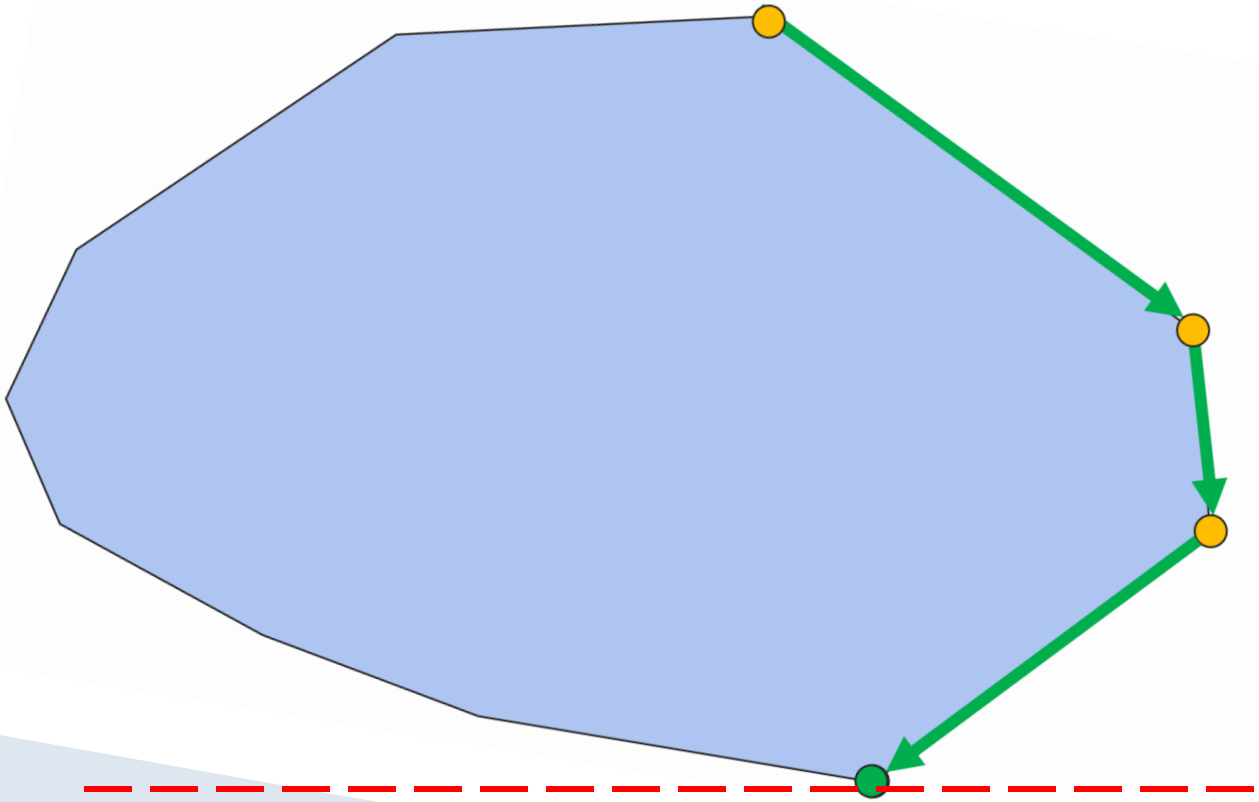


- **Definicija. Ekstrem konveksnog skupa** Θ je svaka točka $\boldsymbol{x \in \Theta}$ koja nije na ravnoj spojnici ikojih drugih dviju točaka skupa Θ.

$$(\nexists x_1, x_2 \in \Theta \backslash \{x\})\left(\nexists \alpha \in (0,1)\right)$$
$$x = \alpha x_1 + (1 - \alpha)x_2$$

- Ekstremi u politopu su **vrhovi** – geometrijski koncept

Simplex - ideja



Gradijent
fje cilja

Optimalna izolinija
grafičke metode

Simplex – ulazni problem

Za simpleks koristimo **LP u standardnoj formi**

minimizirati	$\mathbf{c}^T\mathbf{x}$
uz uvjet	$\mathbf{A}\mathbf{x} = \mathbf{b}$
	$\mathbf{x} \geq \mathbf{0}$

pri čemu je: $\mathbf{c} \in R^n$, $\mathbf{b} \in R^m$ i $\mathbf{b} \geq 0$, matrica $\mathbf{A} \in R^{mxn}$, $rang(\mathbf{A}) = m$ i $m < n$

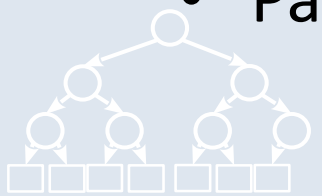
Imamo **m** ograničenja jednakosti, **n** ograničenja nejednakosti ($\mathbf{x} \geq \mathbf{0}$)

Nalazimo se u **n**-dimenzionalnom prostoru (n≥m)

Vrhovi određeni sa n aktivnih ograničenja (oprez!)

Simplex – jezgra metode

- Nalazimo se u **n**-dimenzionalnom prostoru (n≥m)
  - Vrhovi određeni sa **n aktivnih ograničenja**
  - m aktivnih ograničenja je već fiksirano
  - **„Proizvoljnih”** (n-m) biramo među nejednakostima
    - **One fiksiraju vrijednosti (n-m) varijabli na 0**
    - Te varijable ćemo nazivati **nebazičnima**
    - Kad ih uvrstimo u m ograničenja, dobijemo sustav m jednadžbi sa m nepoznanica! (znamo riješiti iz linearne algebre) – varijable koje rješavamo nazivamo **bazične**
- Particionira se skup svih varijabli na **bazične** i **nebazične**



Simplex – definicije

Definicija. Bazično rješenje sustava $\mathbf{Ax} = \mathbf{b}$, $\mathbf{x} \geq \mathbf{0}$ je vektor $\mathbf{x} = [\mathbf{x}_B^T \ \mathbf{0}]$, gdje je $\mathbf{x}_B = \mathbf{B}^{-1}\mathbf{b}$, a $\mathbf{B} \in R^{m \times m}$ odabrana baza (stupci u matrici $\mathbf{A}$) u sustavu od m jednadžbi s n nepoznanica, pri čemu je $m < n$ i $\det(\mathbf{B}) \neq 0$.

Teorem (osnovni teorem linearnog programiranja):

Promatrajmo linearni problem u standardnoj formi. Vrijedi sljedeće:

1. Ako postoji bilo kakvo rješenje, postoji i **izvedivo bazično rješenje**.
2. Ako postoji optimalno rješenje, postoji i bazično optimalno rješenje.

Tabličenje

Hajdemo staviti sve parametre u tablicu:

- m+1 redaka
- n+1 stupaca

c^T	0
A	b

Tabličenje

- Particioniranje A po stupcima na bazične B i nebazične N stupac-vektore

$c_B{}^T$	cN^T	0
B	N	b

Simplex tableau

Iterativno implicitno izračunavanje inverza B^{-1}

- Efikasnije izvođenje jer se susjedni vrhovi razlikuju samo u **jednom aktivnom ograničenju** (koje postavlja neku drugu varijablu na 0)

faktori redukcije po bridovima

-fja cilja

0^T	$c_N^T - c_B^T B^{-1} N$	$-c_B^T B^{-1} b$
I	$B^{-1} N$	$B^{-1} b$

bridovi do susjednih baz.rješ

Vrijednosti baz.var.

Simplex – pseudokod

1. **Početak** iz izvedivog bazičnog rješenja u simpleks tablici
2. Optimalno? Ako su svi faktori redukcije nenegativni, **STOP**
3. Tranzicija u boljeg susjeda:
   a) Odabir ulazne nebazične varijable koja odgovara stupcu ***q***
   b) Odabir izlazne bazične varijable koja odgovara retku ***p***. Ako ne postoji – problem je neograničen, **STOP**
   c) Gauss-Jordan eliminacija za pivot ***(p,q)***
4. Povratak na korak 2

Simplex – pseudokod

- Odabir ulazne nebazične varijable koja odgovara stupcu ***q***
  - Odabere se neka koja ima **NEGATIVNI** faktor redukcije

- Odabir izlazne bazične varijable $\boldsymbol{x_{[p]}}$ koja odgovara retku ***p***
  - $p = \text{argmin}_{i \in \{1,..,m\}} \{ x_{[i]} / B^{-1}A_{iq} \mid B^{-1}A_{iq} > 0 \}$

*[p] označava odabir varijable preko reference retkom

Simplex – primjer

$$\begin{aligned} \max \quad & 7x_1 + 6x_2 \\ \text{uz} \quad & 2x_1 + x_2 \leq 3 \\ & x_1 + 4x_2 \leq 4 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Uvođenjem dviju *slack* varijabli x_3 i x_4 prevodimo LP u standardnu formu

$$\begin{aligned} \min \quad & -7x_1 - 6x_2 + 0x_3 + 0x_4 \\ \text{uz} \quad & 2x_1 + x_2 + x_3 = 3 \\ & x_1 + 4x_2 + x_4 = 4 \\ & x_1, x_2, x_3, x_4 \geq 0 \end{aligned}$$

Simplex – primjer

Tablični zapis je

	a_1	a_2	a_3	a_4	b = RHS	
c^T	–7	–6	0	0	0	$= r^T$
	2	1	1	0	3	
	1	4	0	1	4	

- Tablica već valjana, vrijedi $r_i = c_i$
- najlakše je odabrati početnu bazu $\mathbf{B}_0 = [\boldsymbol{a}_3\ \boldsymbol{a}_4] = \mathbf{I}_2$
  - bazično rješenje $\mathbf{x}_{(0)} = [0\ 0\ 3\ 4]^T$, $f(\mathbf{x}_{(0)})=0$
- Nulti redak sadrži faktore redukcije
  - ima negativnih pa nije optimum

Simplex – primjer

Tablični zapis je

	$\boldsymbol{a}_1$	$\boldsymbol{a}_2$	$\boldsymbol{a}_3$	$\boldsymbol{a}_4$	**RHS**	
$\mathbf{c}^T$	**–7**	–6	0	0	0	$= \mathbf{r}^T$
	2	1	1	0	**3**	
	1	4	0	1	**4**	

Nulti redak ima negativne faktore redukcije pa nije optimum!

Odabiremo **q=1** (prvi stupčani vektor, a_1, ulazi u bazu)

Odabiremo redak p koji odgovara izlaznom vektoru

$p = \text{argmin}_{i \in \{1,2\}} \{x_{[i]} / B^{-1}A_{i1} \; ; B^{-1}A_{i1} > 0\} = \text{argmin}\{3/2, 4/1\} = 1$

Pivot (1,1): u bazu ulazi a_1, a izlazi a_3

Gauss-Jordanova eliminacija tako da $\boldsymbol{a_1 = [0,1,0]^T}$

Simplex – primjer

Tablični zapis je

	$\boldsymbol{a}_1$	$\boldsymbol{a}_2$	$\boldsymbol{a}_3$	$\boldsymbol{a}_4$	**RHS**	
$\mathbf{c}^T$	–7	–6	0	0	0	$= \mathbf{r}^T$
	2	1	1	0	3	
	1	4	0	1	4	

- Pivot (1,1): u bazu ulazi a_1, a izlazi a_3

	$\boldsymbol{a}_1$	$\boldsymbol{a}_2$	$\boldsymbol{a}_3$	$\boldsymbol{a}_4$	**RHS**
$\mathbf{r}^T$	0	–5/2	7/2	0	21/2
	1	1/2	1/2	0	3/2
	0	7/2	–1/2	1	5/2

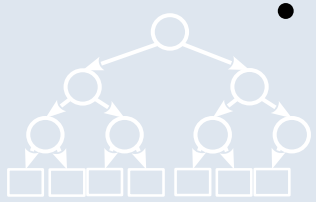
Nova baza $\mathbf{B}_1 = [\boldsymbol{a}_1\ \boldsymbol{a}_4] = \mathbf{I}_2$

- rješenje $\mathbf{x}_{(1)} = [3/2\ \ 0\ \ 0\ \ 5/2]^T$, $f(\mathbf{x}_{(1)})=-21/2$

Simplex – primjer

	a_1	a_2	a_3	a_4	RHS
$\mathbf{r}^T$	0	**–5/2**	7/2	0	21/2
	1	1/2	1/2	0	3/2
	0	7/2	–1/2	1	5/2

- Negativan faktor redukcije r_2 - nije optimum!
- Odabir stupca q=2
- Odabir retka
  - $p = \text{argmin}_{i \in \{1,2\}}\{x_{[i]} / B^{-1}A_{i2} \; ; B^{-1}A_{i2} > 0\} = \text{argmin}\{\mathbf{3}, \mathbf{5/7}\} = 2$
- Pivot (2,2)



Simplex – primjer

	a_1	a_2	a_3	a_4	RHS
$\mathbf{r}^T$	0	**–5/2**	7/2	0	21/2
	1	1/2	1/2	0	3/2
	0	7/2	–1/2	1	5/2

- Pivot (2,2): u bazu ulazi a_2, a izlazi a_4

	a_1	a_2	a_3	a_4	RHS
$\mathbf{r}^T$	0	0	22/7	5/7	86/7
	1	0	4/7	–1/7	8/7
	0	1	–1/7	2/7	5/7

Simplex – primjer

	$\boldsymbol{a}_1$	$\boldsymbol{a}_2$	$\boldsymbol{a}_3$	$\boldsymbol{a}_4$	**RHS**
$\mathbf{r}^T$	0	0	22/7	5/7	86/7
	1	0	4/7	–1/7	8/7
	0	1	–1/7	2/7	5/7

- nema negativnih faktora redukcije
  - Optimum!
- Baza $\mathbf{B}_2 = [\boldsymbol{a}_1\ \boldsymbol{a}_2] = \mathbf{I}_2$
  - Rješenje $\mathbf{x}^* = [8/7\ \ 5/7\ \ 0\ \ 0]^T$, $f(\mathbf{x}^*) = -86/7$

- Rješenje polaznog problema
  - $x_1 = 8/7$ i $x_2 = 5/7$
  - $f_{max} = 86/7$

Simplex – problem početne baze!

- Nekad se nakon pretvorbe u standardni oblik ne vidi odmah bazično rješenje!
- Dvofazni simpleks – rješavaju se 2 LPa u nizu
  - 1. FAZA – **pomoćni umjetni korak** za naći početno bazično rješenje
    - Uvijek ima svoju trivijalnu početnu bazu (tako je konstruiran)
  - 2. FAZA – zapravo rješava LP od interesa

Dvofazni simpleks – umjetni problem

- Pretp. da imamo problem u standardnoj formi

minimizirati	$\mathbf{c}^T\mathbf{x}$
uz uvjet	$\mathbf{Ax} = \mathbf{b}$
	$\mathbf{x} \geq \mathbf{0}$

pri čemu imamo m ograničenja jednakosti. Dodajemo ***m*** umjetnih varijabli da bismo stvorili jediničnu podmatricu

NOVI PROBLEM LP':

minimizirati	$\mathbf{1}^T\mathbf{x}_{n+1:n+m}$
uz uvjet	$[\mathbf{A} \mid \mathbf{I}_m]\,[\mathbf{x}_{1:n} \mid \mathbf{x}_{n+1:n+m}] = \mathbf{b}$
	$\mathbf{x}_{1:n+m} \geq \mathbf{0}$

Dvofazni simpleks – 1. FAZA

- Riješimo novi problem već definiranim postupkom
- Tri moguća ishoda:

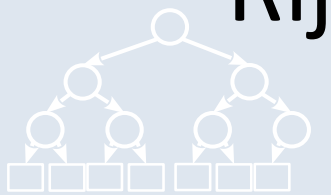
1. Optimum $f^*_{LP}, \neq 0 \Rightarrow originalni\ LP\ neizvediv$! **KRAJ!**
2. Optimum $f^*_{LP}, = 0$
   a) Sve umjetne varijable su nebazične. Adaptacija za 2. FAZU
   b) Neke umjetne varijable su bazične. Izvodi iteracije simpleksa dok ne izađu sve umjetne varijable iz baze. Adaptacija za 2. FAZU

Dvofazni simpleks – 2. FAZA

- Adaptacija tablice od LP’ (sadrži bazu za originalni LP)

1. Pobrisati kolone umjetnih varijabli
2. Zamijeniti fju cilja originalnom
3. Dovesti prvi red u faktore redukcije

- Riješiti LP od te točke nadalje



Dvofazni simpleks – primjer

$$\begin{aligned} \min\ & 2x_1 + 3x_2 \\ \text{uz}\ & 4x_1 + 2x_2 \geq 12 \\ & x_1 + 4x_2 \geq 6 \\ & x_1, x_2 \geq 0 \end{aligned}$$

standardna forma:

$$\begin{aligned} \min\ & 2x_1 + 3x_2 \\ \text{uz}\ & 4x_1 + 2x_2 - x_3 = 12 \\ & x_1 + 4x_2 - x_4 = 6 \\ & x_1, x_2, x_3, x_4 \geq 0 \end{aligned}$$

Dvofazni simpleks – primjer

Dodajemo dvije nove varijable i novu fju cilja

	$\boldsymbol{a}_1$	$\boldsymbol{a}_2$	$\boldsymbol{a}_3$	$\boldsymbol{a}_4$	$\boldsymbol{a}_5$	$\boldsymbol{a}_6$	**b = RHS**
$\mathbf{c}^T$	0	0	0	0	1	1	0
	4	2	–1	0	1	0	12
	1	4	0	–1	0	1	6

Dvofazni simpleks – primjer

Nakon nekoliko iteracija...

	$\boldsymbol{a}_1$	$\boldsymbol{a}_2$	$\boldsymbol{a}_3$	$\boldsymbol{a}_4$	$\boldsymbol{a}_5$	$\boldsymbol{a}_6$	**RHS**
$\mathbf{r}^T$	0	0	0	0	1	1	0
	1	0	–2/7	1/7	2/7	–1/7	18/7
	0	1	1/14	–2/7	–1/14	2/7	6/7

baza $\mathbf{B}_2 = [\boldsymbol{a}_1\ \boldsymbol{a}_2] = \mathbf{I}_2$

faktori redukcije su nenegativni -> **OPTIMUM** umjetnog problema

umjetne varijable su =0 i umjetna ciljna funkcija =0

gotova 1. FAZA

Dvofazni simpleks – primjer

2. faza – iz prethodne tablice se izbace stupci umjetnih varijabli i zamijeni se ciljna fja

	$\boldsymbol{a}_1$	$\boldsymbol{a}_2$	$\boldsymbol{a}_3$	$\boldsymbol{a}_4$	**RHS**
$\mathbf{c}^T$	2	3	0	0	0
	1	0	–2/7	1/7	18/7
	0	1	1/14	–2/7	6/7

Ispravak 0-tog retka - eliminacijom iznad baznih stupaca

Dvofazni simpleks – primjer

	a_1	a_2	a_3	a_4	**RHS**
$\mathbf{r}^T$	0	0	5/14	4/7	–54/7
	1	0	–2/7	1/7	18/7
	0	1	1/14	–2/7	6/7

- nema negativnih faktora redukcije
  - **OPTIMUM**
  - Rješenje proširenog izvornog problema je $\mathbf{x} = [18/7 \;\; 6/7 \;\; 0 \;\; 0]^T$
  - rješenje izvornog problema $\mathbf{x} = [18/7 \;\; 6/7]^T$, $f(\mathbf{x})=54/7$

Dualnost

- Teorija nastala poopćenjem metode Lagrangeovih množitelja

- Svaki LP (kojeg ćemo zvati „primal”) ima svoj povezani LP kojeg zovemo „dual”

Dualnost – kanonska forma

minimizirati $\mathbf{c}^T\mathbf{x}$

uz uvjet $\mathbf{Ax} \leq \mathbf{b}$

$\mathbf{x} \geq \mathbf{0}$

- Dual:

maksimizirati $\mathbf{b}^T\mathbf{y}$

uz uvjet $\mathbf{A}^T\mathbf{y} \geq \mathbf{c}$

$\mathbf{y} \geq \mathbf{0}$

Dual - izvođenje

primal	dual
broj ograničenja	broj varijabli
broj varijabli	broj ograničenja
rhs	funkcija cilja
funkcija cilja	rhs
A matrica koeficijenata	$\mathbf{A}^T$
jednakost	urs varijabla
urs varijabla	jednakost
<= ograničenje	>= varijabla
>= ograničenje	<= varijabla
>= varijabla	>= ograničenje
<= varijabla	<= ograničenje

Veze duala i primala - teoremi

- „Dual duala je primal”
- Slaba dualnost
- Jaka dualnost
- Komplementarnost

Veze duala i primala - teoremi

- Slaba dualnost
  - Ako je x izvedivo primalno rješenje i y je izvedivo dualno rješenje, tada je $y^T b \leq c^T x$

- Jaka dualnost
  - Ako linearni program ima optimalno rješenje, onda ga ima i dual i njihove vrijednosti su jednake.

Veze duala i primala - teoremi

- Komplementarnost
  - Ako su x i y izvediva rješenja primala i duala, onda su ***optimalna*** **ako i samo ako** vrijedi:

$$x_j\left(c_j - y^T A_{:,j}\right) = 0, \forall j$$

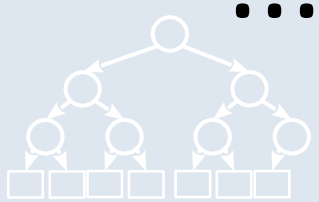
Faktor redukcije varijable x_j

$$y_i\left(A_{i,:}x - b_i\right) = 0, \forall i$$

Dopunjenje i-tog ograničenja primala

Dualnost - korisnost

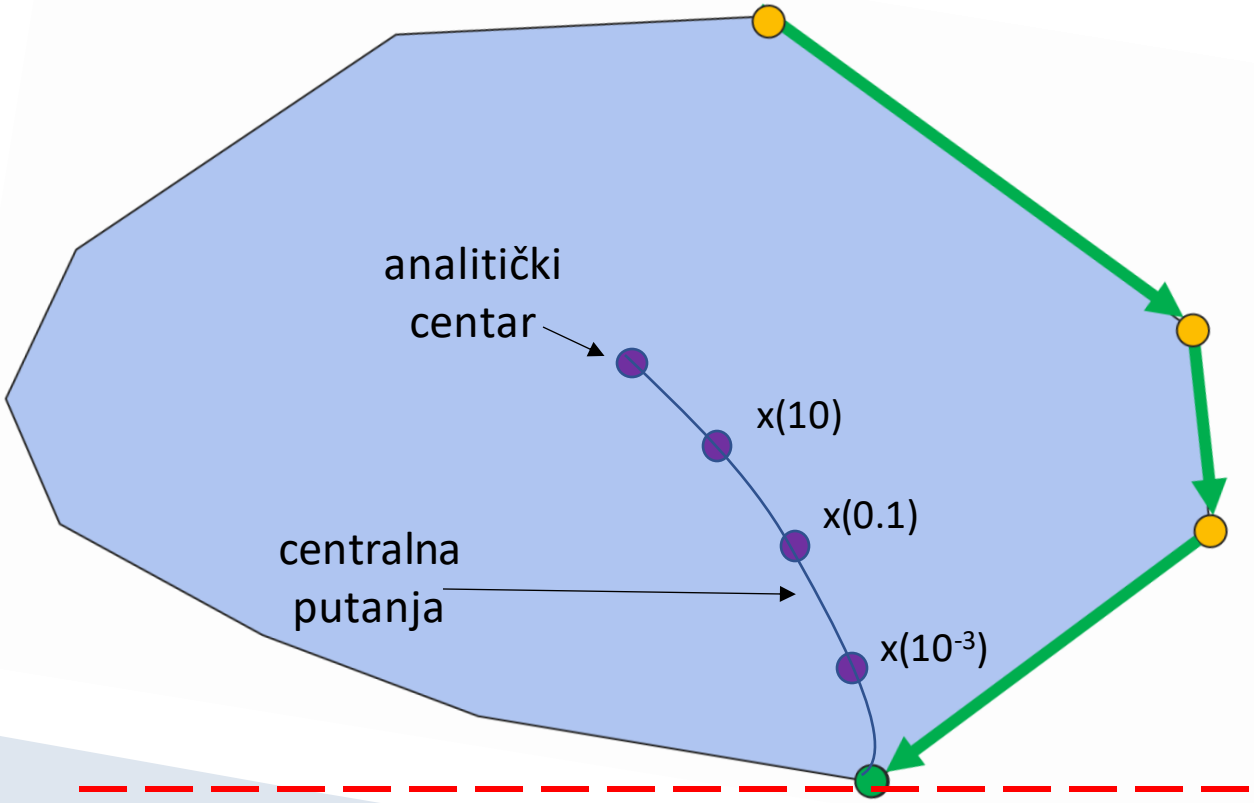
- Ekonomska interpretacija – cijene nad ograničenim resursima
- Minimax teorem u teoriji igara
- Analiza osjetljivosti
- Dualna simpleksna metoda
- …



Simplex – problem!

- Klee-Minty 1972. – konstrukcija perturbirane jedinične hiperkocke za popularna pravila biranja pivota
- Simplex u najgorem slučaju ima eksponencijalnu složenost
- LP je unutar klase problema P

Metoda unutarnjih točaka - ideja



Gradijent
fje cilja

Optimalna izolinija
grafičke metode

Metoda unutarnjih točaka – ideja 1/3

$$\begin{aligned} &\min && \mathbf{c}^T\mathbf{x} \\ &\text{uz uvjet} && \mathbf{Ax} = \mathbf{b} \\ & && \mathbf{x} \geq \mathbf{0} \end{aligned}$$

$$\begin{aligned} &\max && \mathbf{b}^T\mathbf{y} \\ &\text{uz uvjet} && \mathbf{A}^T\mathbf{y} + \mathbf{s} = \mathbf{c} \\ & && \mathbf{s} \geq \mathbf{0} \end{aligned}$$

Metoda unutarnjih točaka – ideja 2/3

$$\begin{aligned} &\min && \mathbf{c^T x - \mu\, 1^T \log(x)} \\ &\text{uz uvjet} && \mathbf{Ax = b} \end{aligned}$$

Barijerni problemi!

Logaritamska barijera

$$\begin{aligned} &\max && \mathbf{b^T y + \mu\, 1^T \log(s)} \\ &\text{uz uvjet} && \mathbf{A^T y + s = c} \end{aligned}$$

Metoda unutarnjih točaka – ideja 3/3

KKT uvjeti za μ-barijerne probleme

$$\mathbf{A}\mathbf{x}(\boldsymbol{\mu}) = \mathbf{b}$$

$$\mathbf{x}(\boldsymbol{\mu}) \geq \mathbf{0}$$

$$\mathbf{A}^T\mathbf{y}(\boldsymbol{\mu}) + \mathbf{s}(\boldsymbol{\mu}) = \mathbf{c}$$

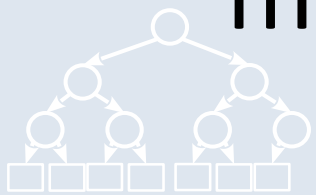
$$\mathbf{s}(\boldsymbol{\mu}) \geq \mathbf{0}$$

$$\mathbf{X}(\boldsymbol{\mu})\mathbf{S}(\boldsymbol{\mu})\mathbf{1} = \mathbf{1}\boldsymbol{\mu}$$

pri čemu $X(\mu)=\mathrm{diag}(x(\mu))$, $S(\mu)=\mathrm{diag}(s(\mu))$

Primalni algoritam praćenja putanje

- Barijerni problem „pretežak” iz KKT
- Taylorov raspis barijerne fje cilja do kvadratnog člana
- Optimizacija Taylorove aproksimacije metodom Lagrangeovih množitelja za pronalazak minimizirajućeg smjera iz trenutne točke



06 – Dynamic Programming

Advanced Algorithms and Data Structures

UNIVERSITY OF ZAGREB
Faculty of Electrical Engineering and Computing

Creative Commons

- You are free to:
  - share — multiply, distribute, and publicly communicate the work
  - adapt the work
- under the following conditions:
  - **Attribution:** You must acknowledge and indicate the authorship of the work in the manner specified by the author or license provider (but not in a way that suggests you or your use of the work have their direct support).
  - **Non-commercial:** You may not use this work for commercial purposes.
  - **Share alike:** If you modify, transform, or build upon this work, you may only distribute the modified work under the same or a similar license.

Fibonacci numbers (1)

- $F_1 = F_2 = 1, \;\; F_n = F_{n-1} + F_{n+2}$
- How many calculations do we make?
- How many are repeated?
- Tree of all calculation cases

```
def fib(n):
    if n == 1 or n == 2:
        return 1
    else:
        return fib(n-1) + fib(n-2)
```

Fibonacci numbers (2)

- Time complexity:

$$T_n = T_{n-1} + T_{n-2} + O(1) = \varphi^n$$

$$T_n \geq 2T_{n-1} \Rightarrow at\ least\ O(2^{\frac{n}{2}})$$

```
def fib(n):
    if n == 1 or n == 2:
        return 1
    else:
        return fib(n-1) + fib(n-2)
```

Dynamic programming (1)

- Strategy/paradigm for algorithm design
- Method in mathematics

- “Programming” – refers to strategy

- Bellman’s optimality principle

Dynamic programming (2)

- Bottom-up approach: combine a solution from less complex solutions
  - Unlike divide and conquer, which is top-down
- Applicable when there are **overlapping subproblems**
- Major idea – **reuse calculation results**
- **Memoization** - method of tabularly storing calculation results

Dynamic programming (3)

- Used for problems such as:
  - One-dimensional distribution
  - Two-dimensional distribution
  - Shortest path
  - Dynamics of equipment replacement

- Or to solve Advent of Code problems: https://adventofcode.com/2023/day/12

DP problem features

- **Optimal substructure**
  - Each problem's optimal solution contains the **optimal** solutions of **independent*** subproblems
  - Form a DAG
  - Not enough -> points to a greedy algorithm; e.g. Dijkstra
- **Overlapping subproblems**
  - The problem's solution requires (leads to) multiple solutions to **identical subproblems**

*Subproblems are independent regarding one another (when on the same level)

Solving a DP problem (1)

4 steps:

1. **Characterize the structure of an optimal solution**
   a) Does the problem meet the requirements for an optimal solution?
   b) Are the problems overlapping?
   - Solvability of the problem via DP – intuition
2. **Recursively define the value of an optimal solution**
   - "value" is from the function being optimized

Solving a DP problem (1)

3. **Compute the value of an optimal solution**
   - Using bottom-up and memoization
4. **Construct an optimal solution from the computed information**
   - From here on – the problem is solved
   - Packs the entire solution as a set of decisions
   - If we just need a value, then this is skipped

Fibonacci numbers – DP approach

- Implement memorization
- Dictionary to store calculated results
- Still recursive

```
def dyn_fib(n, memo: dict):
    if n == 1 or n == 2:
        memo[n] = 1
        return 1
    else:
        if n-1 not in memo:
            dyn_fib(n-1, memo)
        if n-2 not in memo:
            dyn_fib(n-2, memo)
        memo[n] = memo[n-1] + memo[n-2]
        return memo[n]
```

Example 2 - Factorials

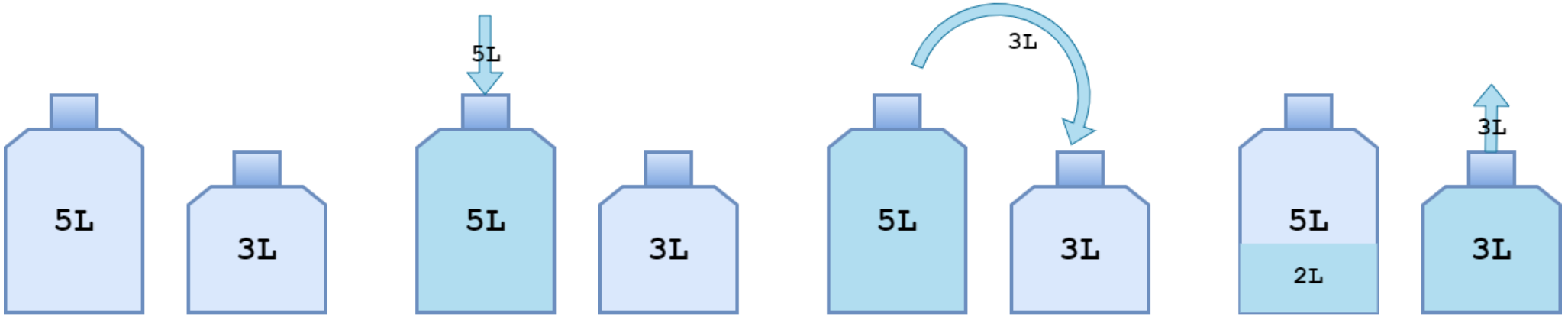
Calculate the factorial $n!$.

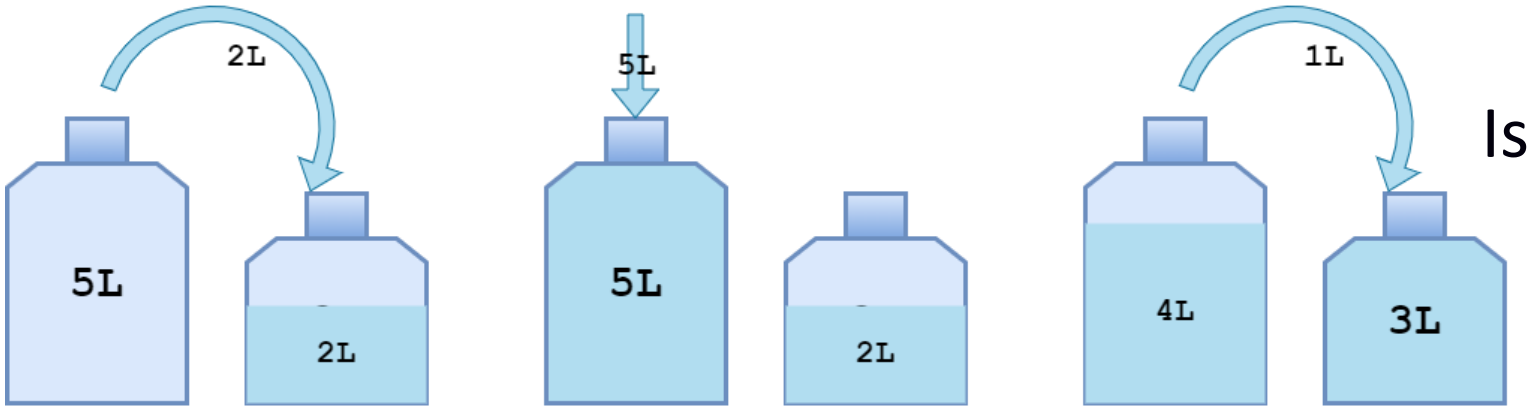
- $n! = 1 \cdot 2 \cdot 3 \cdot \cdots \cdot (n-1) \cdot n$
- We just need a value, not the decisions.

- What are the subproblems in relation to the problem?
  - Do they overlap?
- Can we define the solution recursively?
- Can we use memoization for computation?
- At which step do we stop?
  - Do we care about the decisions?

Example 3 – Jugs

- Problem from “Die Hard 2”
- Fill any of the jugs with 4L





Is the value 4L enough?

Knapsack problem (1)

Given a set of items, each with a **cost** (weight) and a **value**, determine which items to include in the collection so that the **total cost is less than or equal to a given limit** and the **total value is as large as possible**.

How many DP steps are we considering?

Knapsack problem (2)

Types of knapsack problems:

- **0-1 knapsack problem**
  - Finite set of items i, having one of each kind
  - We use “0” and “1” to denote that an item doesn’t exist (x_i)
- **Bounded knapsack problem**
  - Finite set of items, but each item can have up to c copies
- **Unbounded knapsack problem**
  - No upper bounds for each item

Knapsack problem (3)

Other types of knapsack problems:

- Multi-dimensional knapsack problem
- Multiple knapsack problem
- Quadratic knapsack problem

Knapsack problem – formalized (1)

x_i	i[th] item selection
$w_i\ (or\ c_i)$	i[th] item weight or cost
v_i	i[th] item value
W	Maximum weight or cost

0-1 knapsack problem

$$maximize \sum_{i=1}^{n} x_i v_i$$

$$subject\ to \sum_{i=1}^{n} x_i w_i \leq W$$

$$x_i \in \{0, 1\}$$

Knapsack problem – formalized (2)

Bounded knapsack problem

$$maximize \sum_{i=1}^{n} x_i v_i$$

$$subject\ to \sum_{i=1}^{n} x_i w_i \leq W$$

$$x_i \in [0, \mathrm{c}]$$

x_i	i[th] item selection
$w_i\ (or\ c_i)$	i[th] item weight or cost
v_i	i[th] item value
W	Maximum weight or cost
c	Maximum copies of each item

Knapsack problem – formalized (3)

x_i	i[th] item selection
$w_i\ (or\ c_i)$	i[th] item weight or cost
v_i	i[th] item value
W	Maximum weight or cost

Unbounded knapsack problem

$$maximize \sum_{i=1}^{n} x_i v_i$$

$$subject\ to \sum_{i=1}^{n} x_i w_i \leq W$$

$$x_i \in \mathbb{N}$$

Knapsack – a practical example (1)

- Studying for an exam on a tight schedule
- Topics
  - Time required to learn
  - The number of points it will bring on the exam

	Topic 1	**Topic 2**	**Topic 3**	**Topic 4**
Points	10	8	15	7
Cost in time	5	2	7	3

$c_{max} = 12$ (hours)

Knapsack – example (2)

Step 1: Characterize the structure of an optimal solution

- We assume the optimal solution has k topics and the entire time is used
- The optimal chosen topics set: Ω
- If we remove any topic, then we have Ω for that capacity and $k-1$ topics

- The optimal solution is reached through the optimal solutions on smaller sets of chosen topics – **optimal substructure**
- The solution for k topics is reached by expanding from the solution for $k-1$ topics, then that on $k-2$ topics… – a **overlapping problems**

Knapsack – example (3)

Step 2: Recursively define the value of an optimal solution

- We iteratively decide if we want to include a topic at some available amount of time

$$v_k(c) = \max\{\, v_{k-1}(c),\ v_{k-1}[c - cost(k)] + value(k) \,\}$$

$$v_0(\cdot) = 0$$

Knapsack – example (4)

Step 3: Compute the value of an optimal solution

- Create a table with expected points depending on the time capacity

Knapsack – example (5)

	Topic 1	Topic 2	Topic 3	Topic 4
Value	10	8	15	7
Cost	5	2	7	3

Cap	T1	T2	T3	T4
1	0			
2	0			
3	0			
4	0			
5	10			
6				
7				
8				
9				
10				
11				
12				

$$v_k(c) = \max\{ \; v_{k-1}(c), \; v_{k-1}[c - cost(k)] + value(k) \; \}$$

Knapsack – example (6)

	Topic 1	Topic 2	Topic 3	Topic 4
Value	10	8	15	7
Cost	5	2	7	3

Cap	T1	T2	T3	T4
1	0	0		
2	0	8		
3	0			
4	0			
5	10			
6	10			
7	10			
8	10			
9	10			
10	10			
11	10			
12	10			

$$v_k(c) = \max\{$$
$$v_{k-1}(c),$$
$$v_{k-1}[c - cost(k)] + value(k)$$
$$\}$$

We now observe both T1 and T2 values

Knapsack – example (7)

	Topic 1	Topic 2	Topic 3	Topic 4
Value	10	8	15	7
Cost	5	2	7	3

Cap	T1	T2	T3	T4
1	0	0		
2	0	8		
3	0	8		
4	0	8		
5	10	10		
6	10	10		
7	10			
8	10			
9	10			
10	10			
11	10			
12	10			

$$v_k(c) = \max\{$$
$$v_{k-1}(c),$$
$$v_{k-1}[c - cost(k)] + value(k)$$
$$\}$$

Knapsack – example (8)

	Topic 1	Topic 2	Topic 3	Topic 4
Value	10	8	15	7
Cost	5	2	7	3

Cap	T1	T2	T3	T4
1	0	0		
2	0	8		
3	0	8		
4	0	8		
5	10	10		
6	10	10		
7	10	18		
8	10	18		
9	10	18		
10	10	18		
11	10	18		
12	10	18		

$$v_k(c) = \max\{ v_{k-1}(c), v_{k-1}[c - cost(k)] + value(k) \}$$

Knapsack – example (9)

	Topic 1	Topic 2	Topic 3	Topic 4
Value	10	8	15	7
Cost	5	2	7	3

Cap	T1	T2	T3	T4
1	0	0	0	
2	0	8	8	
3	0	8	8	
4	0	8	8	
5	10	10	10	
6	10	10	10	
7	10	18	18	
8	10	18	18	
9	10	18		
10	10	18		
11	10	18		
12	10	18		

$$v_k(c) = \max\{ v_{k-1}(c), v_{k-1}[c - cost(k)] + value(k) \}$$

15 < 18 – keeping T1 and T2

Knapsack – example (10)

	Topic 1	Topic 2	Topic 3	Topic 4
Value	10	8	15	7
Cost	5	2	7	3

Cap	T1	T2	T3	T4
1	0	0	0	
2	0	8	8	
3	0	8	8	
4	0	8	8	
5	10	10	10	
6	10	10	10	
7	10	18	18	
8	10	18	18	
9	10	18	23	
10	10	18	23	
11	10	18	23	
12	10	18		

$$v_k(c) = \max\{ v_{k-1}(c), v_{k-1}[c - cost(k)] + value(k) \}$$

8+15 > 18 – keeping T2 and T3

Knapsack – example (11)

	Topic 1	Topic 2	Topic 3	Topic 4
Value	10	8	15	7
Cost	5	2	7	3

Cap	T1	T2	T3	T4
1	0	0	0	
2	0	8	8	
3	0	8	8	
4	0	8	8	
5	10	10	10	
6	10	10	10	
7	10	18	18	
8	10	18	18	
9	10	18	23	
10	10	18	23	
11	10	18	23	
12	10	18	25	

$$v_k(c) = \max\{ v_{k-1}(c), v_{k-1}[c - cost(k)] + value(k) \}$$

10+15 > 23 – keeping T1 and T3

Knapsack – example (12)

	Topic 1	Topic 2	Topic 3	Topic 4
Value	10	8	15	7
Cost	5	2	7	3

Cap	T1	T2	T3	T4
1	0	0	0	0
2	0	8	8	8
3	0	8	8	8
4	0	8	8	8
5	10	10	10	15
6	10	10	10	15
7	10	18	18	
8	10	18	18	
9	10	18	23	
10	10	18	23	
11	10	18	23	
12	10	18	25	

$$v_k(c) = \max\{ v_{k-1}(c), v_{k-1}[c - cost(k)] + value(k) \}$$

8+7 > 10 – keeping T2 and T4

Knapsack – example (13)

	Topic 1	Topic 2	Topic 3	Topic 4
Value	10	8	15	7
Cost	5	2	7	3

Cap	T1	T2	T3	T4
1	0	0	0	0
2	0	8	8	8
3	0	8	8	8
4	0	8	8	8
5	10	10	10	15
6	10	10	10	15
7	10	18	18	18
8	10	18	18	18
9	10	18	23	23
10	10	18	23	25
11	10	18	23	25
12	10	18	25	

$$v_k(c) = \max\{ v_{k-1}(c),\ v_{k-1}[c - cost(k)] + value(k) \}$$

T1 + T2 + T4

Knapsack – example (14)

	Topic 1	Topic 2	Topic 3	Topic 4
Value	10	8	15	7
Cost	5	2	7	3

Cap	T1	T2	T3	T4
1	0	0	0	0
2	0	8	8	8
3	0	8	8	8
4	0	8	8	8
5	10	10	10	15
6	10	10	10	15
7	10	18	18	18
8	10	18	18	18
9	10	18	23	23
10	10	18	23	25
11	10	18	23	25
12	10	18	25	**30**

$$v_k(c) = \max\{ v_{k-1}(c), v_{k-1}[c - cost(k)] + value(k) \}$$

T2 + T3 + T4

Knapsack – example (15)

	Topic 1	Topic 2	Topic 3	Topic 4
Value	10	8	15	7
Cost	5	2	7	3

<u>Step 4: Construct an optimal solution from the computed information</u>

- Change in row value right->left signifies an item was added
- Which items – determined by arrows
  - Impl. flag if the column item is included
- $O(C_{max} \cdot N)$

Cap	T1	T2	T3	T4
1	0	0	0	0
2	0	8	8	8
3	0	8	8	8
4	0	8	8	8
5	10	10	10	15
6	10	10	10	15
7	10	18	18	18
8	10	18	18	18
9	10	18	23	23
10	10	18	23	25
11	10	18	23	25
12	10	18	25	**30**

Knapsack – human optimization

- Sort by cost
- Ignore capacities < minimum cost

	Topic 1	Topic 2	Topic 3	Topic 4
Value	10	8	15	7
Cost	5	2	7	3

	Topic 2	Topic 4	Topic 1	Topic 3
Value	8	7	10	15
Cost	2	3	5	7

Knapsack – naïve recursive solution

```
def knapsack(capacity, costs, values, num_items):
# Base case: If no items are left or capacity is 0
    if num_items == 0 or capacity == 0:
        return 0

    # Check if there is enough capacity for the current item
    if costs[num_items-1] > capacity:
        return knapsack(capacity, costs, values, num_items-1)

    # Return the maximum of two cases:
    # (1) nth item included
    # (2) not included
    else:
        return max(
            knapsack(capacity, costs, values, num_items-1),
            values[num_items-1] + knapsack(capacity-costs[num_items-1], costs,
values, num_items-1)
            )
```

Knapsack – with memoization

```
def knapsack(capacity, costs, values, num_items, memo={}):
    # Base case: If no items are left or capacity is 0
    if num_items == 0 or capacity == 0:
        return 0

    # Check if the result is already in the memo dictionary
    if (num_items, capacity) in memo:
        return memo[(num_items, capacity)]
    # Capacity check
    if costs[num_items-1] > capacity:
        result = knapsack(capacity, costs, values, num_items-1, memo)
    else:
        # determine maximum
        result = max(
            knapsack(capacity, costs, values, num_items-1, memo),
            values[num_items-1] + knapsack(capacity-costs[num_items-1], costs, values,
num_items-1, memo)
        )
    # Store the result in the memo dictionary
    memo[(num_items, capacity)] = result
    return result
```

Knapsack – with decisions

```
def knapsack(capacity, costs, values, num_items, memo={}, items_taken=[]):

    # Base case: If no items are left or capacity is 0
    if num_items == 0 or capacity == 0:
        return 0, items_taken

    # Check if the result is already in the memo dictionary
    if (num_items, capacity) in memo:
        return memo[(num_items, capacity)], items_taken

    # determine if the item fits the capacity
    if costs[num_items-1] > capacity:
        result, items_taken = knapsack(capacity, costs, values, num_items-1, memo, items_taken)
    else:
        # calculate the two decision cases
          included_value, included_items = knapsack(capacity-costs[num_items-1], costs, values, num_items-1, memo,
items_taken + [num_items-1])
        excluded_value, excluded_items = knapsack(capacity, costs, values, num_items-1, memo, items_taken)
        # determine MAX
        if included_value + values[num_items-1] > excluded_value:
            result = included_value + values[num_items-1]
            items_taken = included_items
        else:
            result = excluded_value
            items_taken = excluded_items

    # Store the result in the memo dictionary
    memo[(num_items, capacity)] = result
    return result, items_taken
```

DP example task

- Traverse a matrix field by going down or right-down
- Determine the path with the maximum values passed