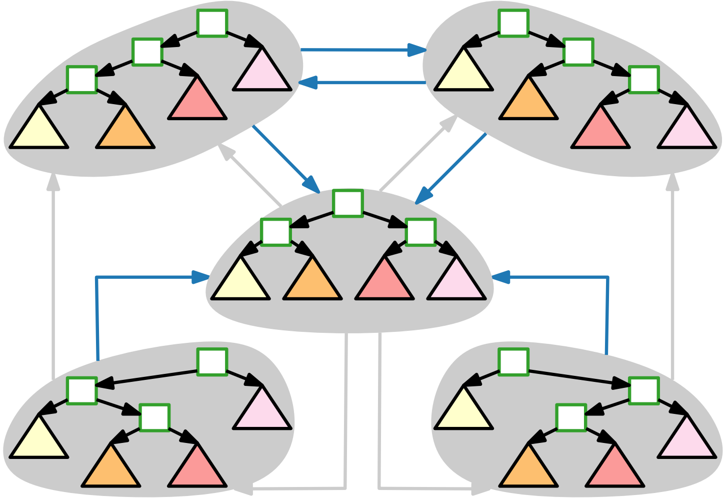
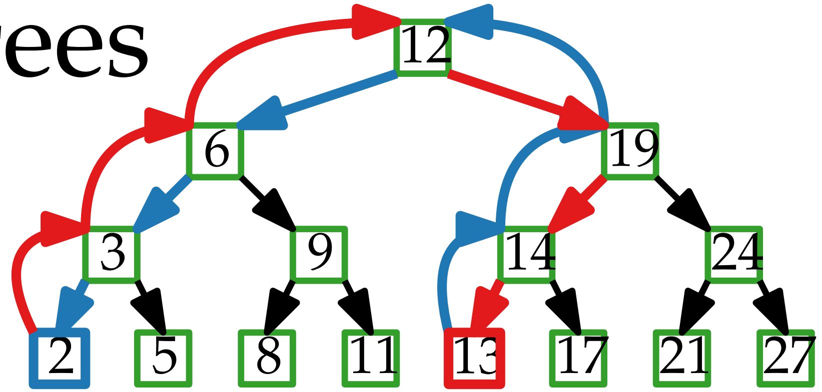
Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Advanced Algorithms

Optimal Binary Search Trees

Splay Trees



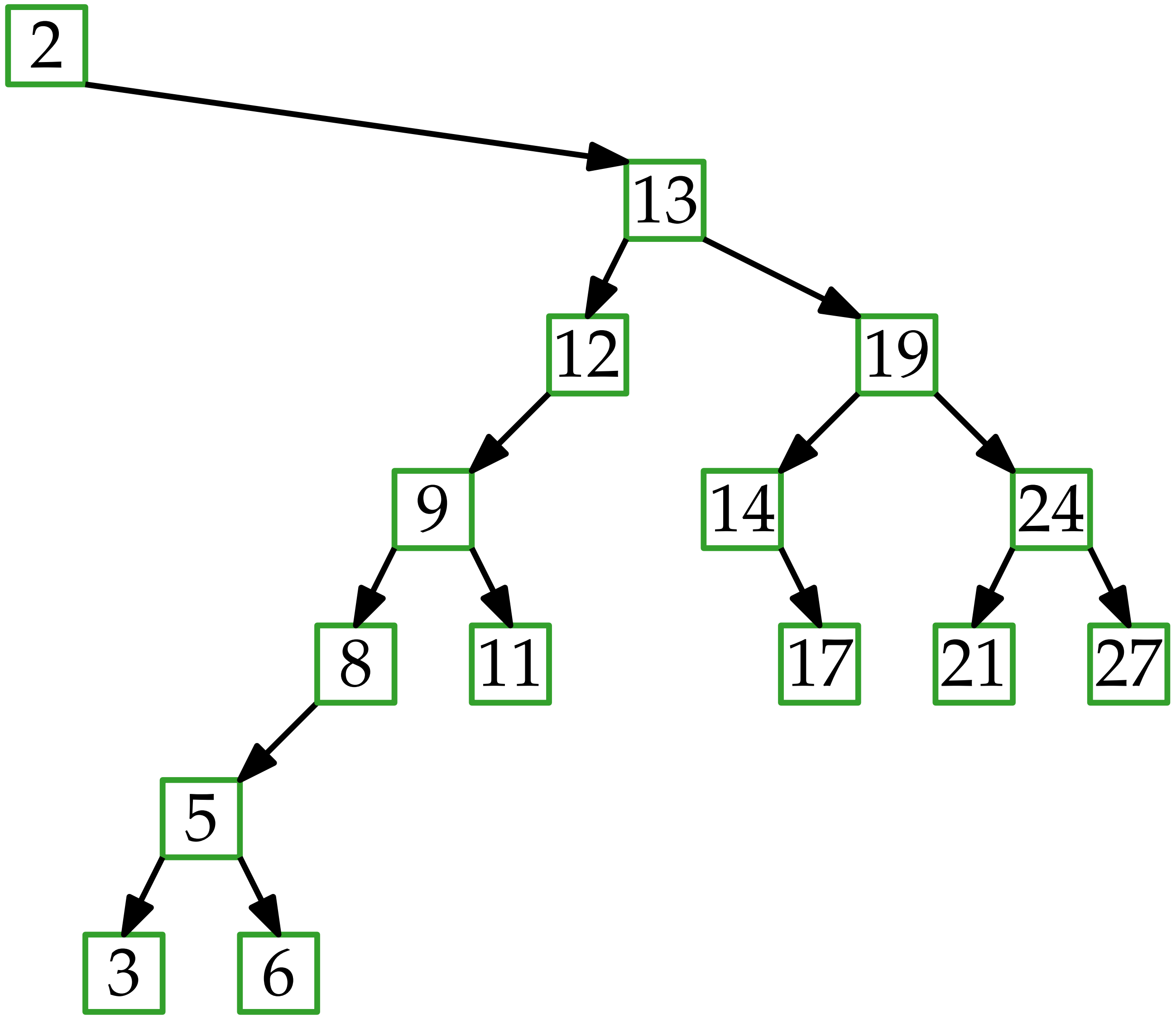


Philipp Kindermann · WS20

Part I:
How Good is a Binary Search Tree?

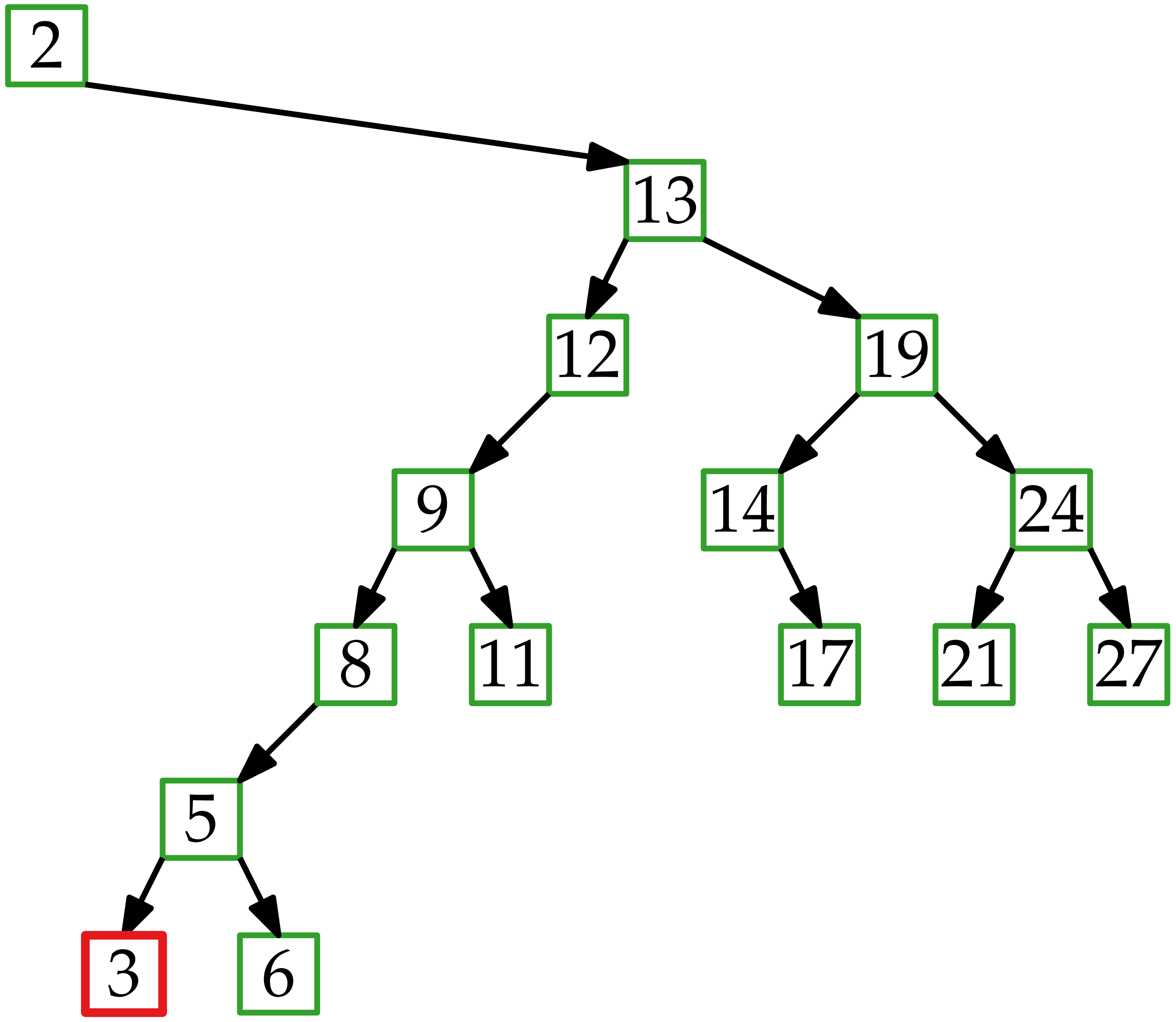
How Good is a Binary Search Tree?

Binary search tree:



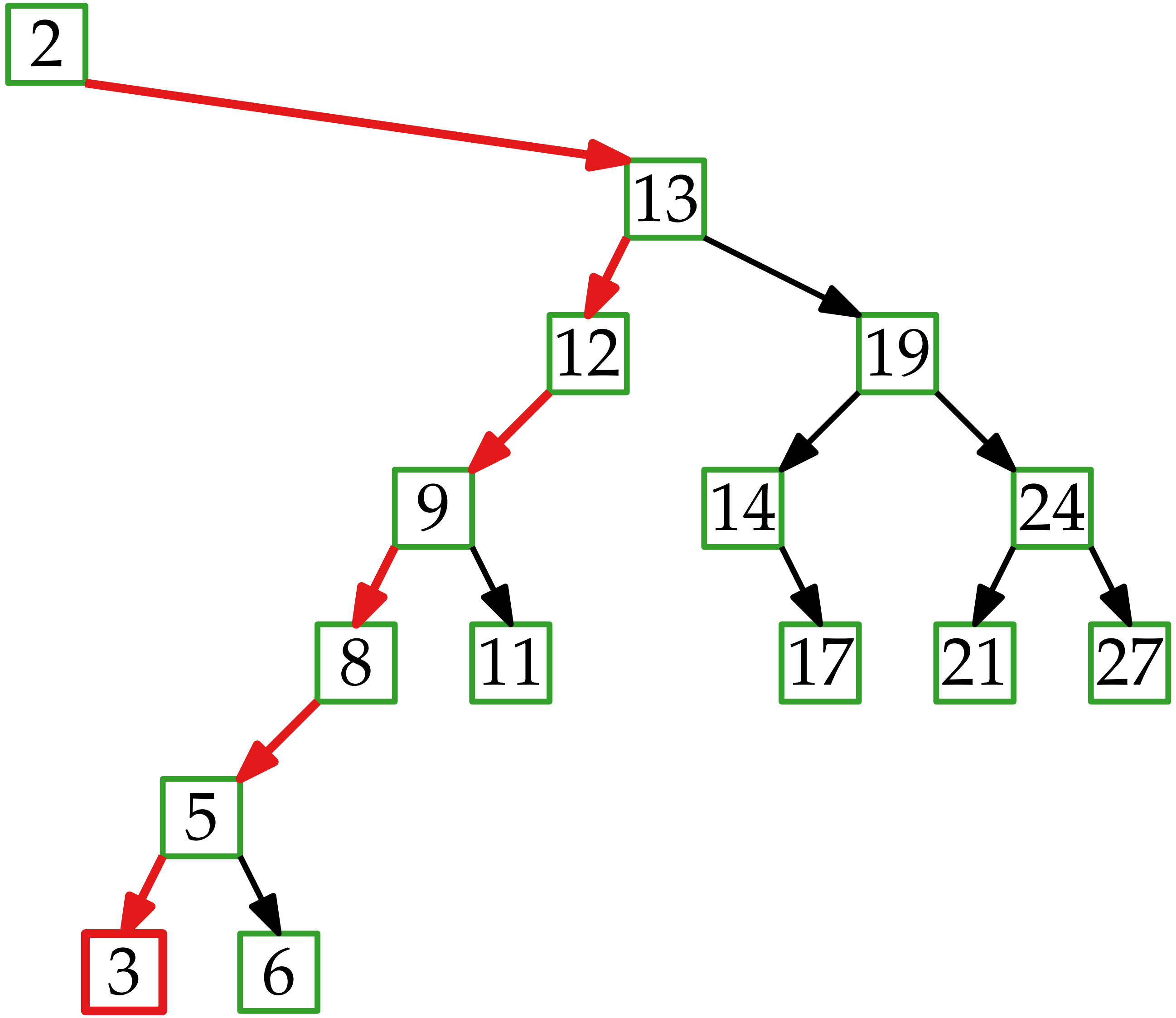
How Good is a Binary Search Tree?

Binary search tree:



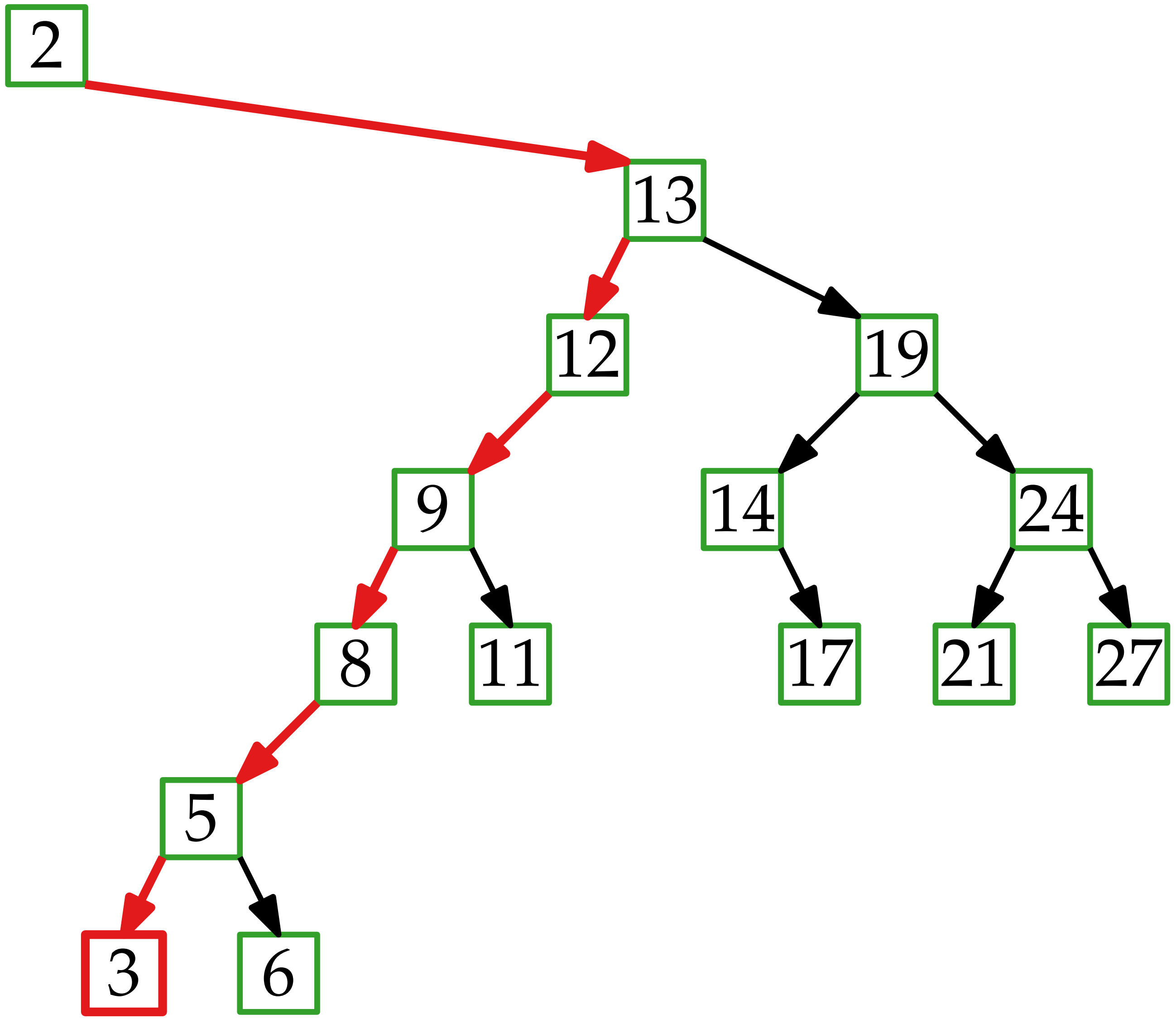
How Good is a Binary Search Tree?

Binary search tree:



How Good is a Binary Search Tree?

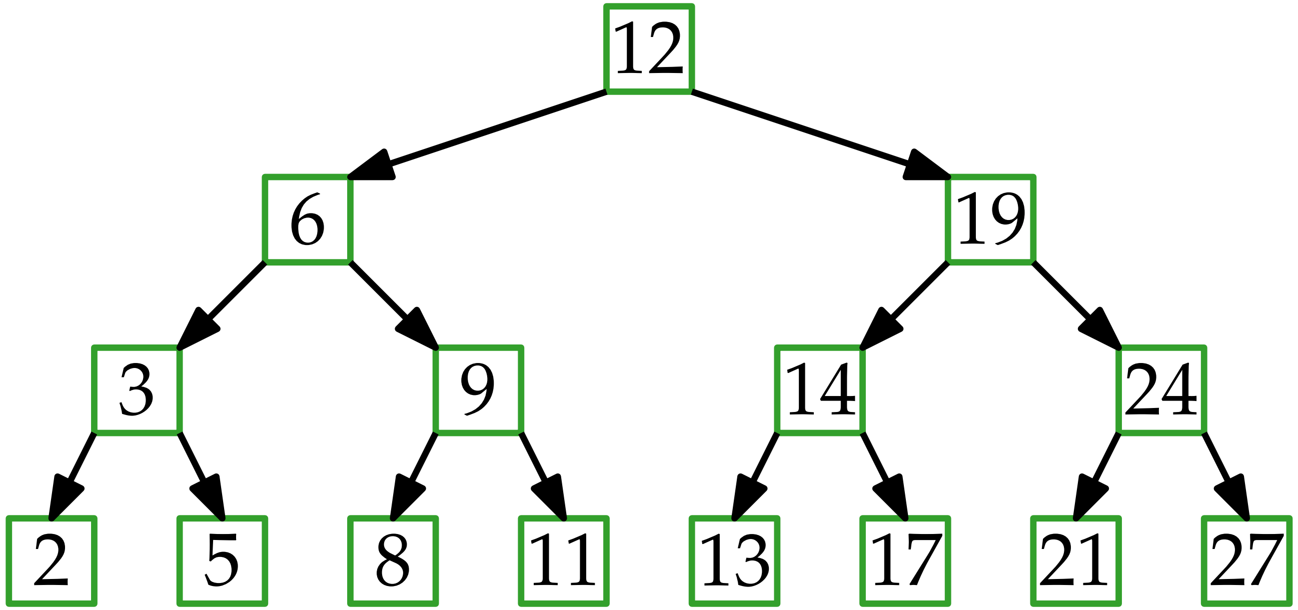
Binary search tree: w.c. query time $\Theta(n)$



How Good is a Binary Search Tree?

Binary search tree: w.c. query time $\Theta(n)$

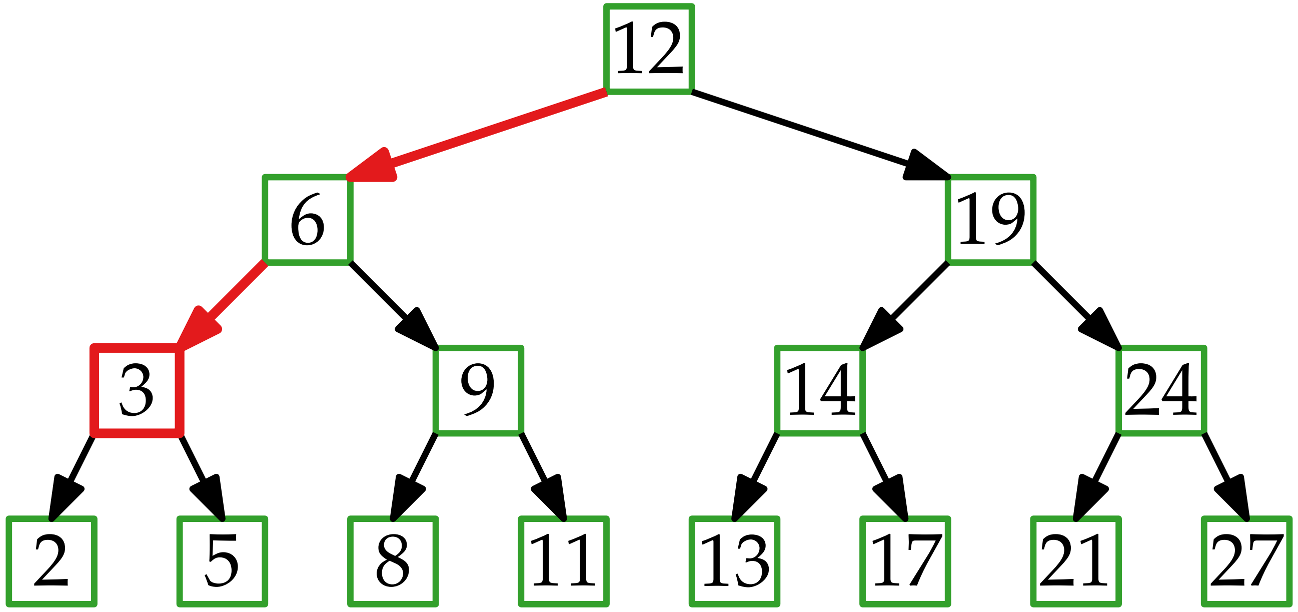
Balanced binary search tree:
(e.g. Red-Black-Tree)



How Good is a Binary Search Tree?

Binary search tree: w.c. query time $\Theta(n)$

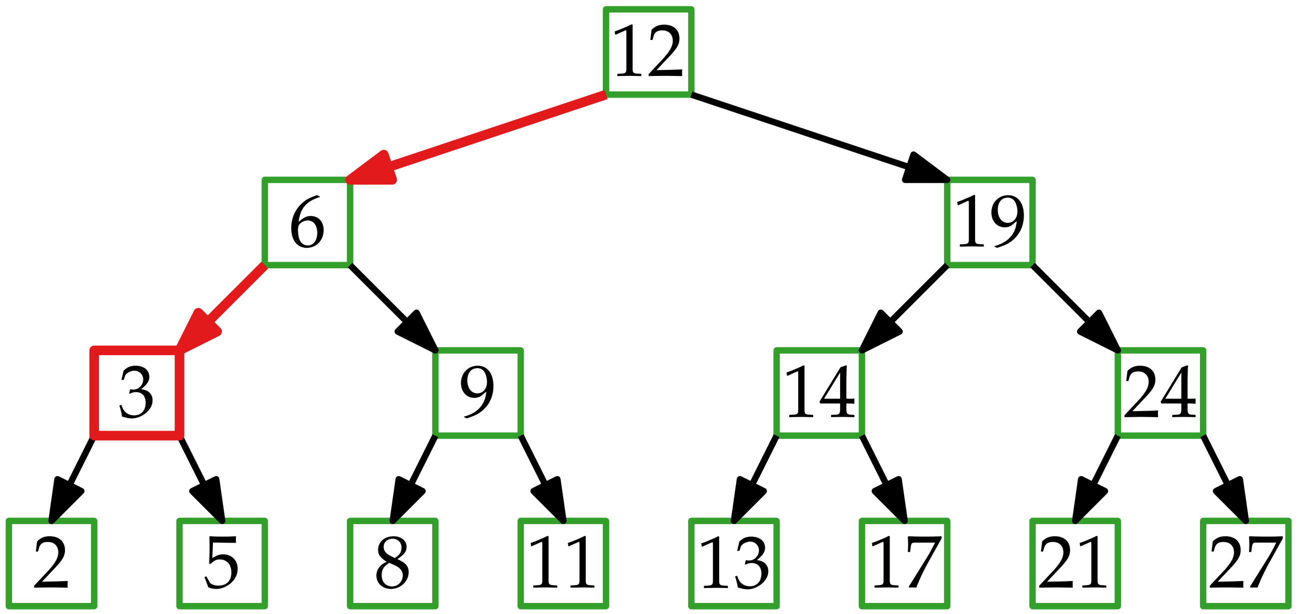
Balanced binary search tree:
(e.g. Red-Black-Tree)



How Good is a Binary Search Tree?

Binary search tree: w.c. query time $\Theta(n)$

Balanced binary search tree: w.c. query time $\Theta(\log n)$
(e.g. Red-Black-Tree)

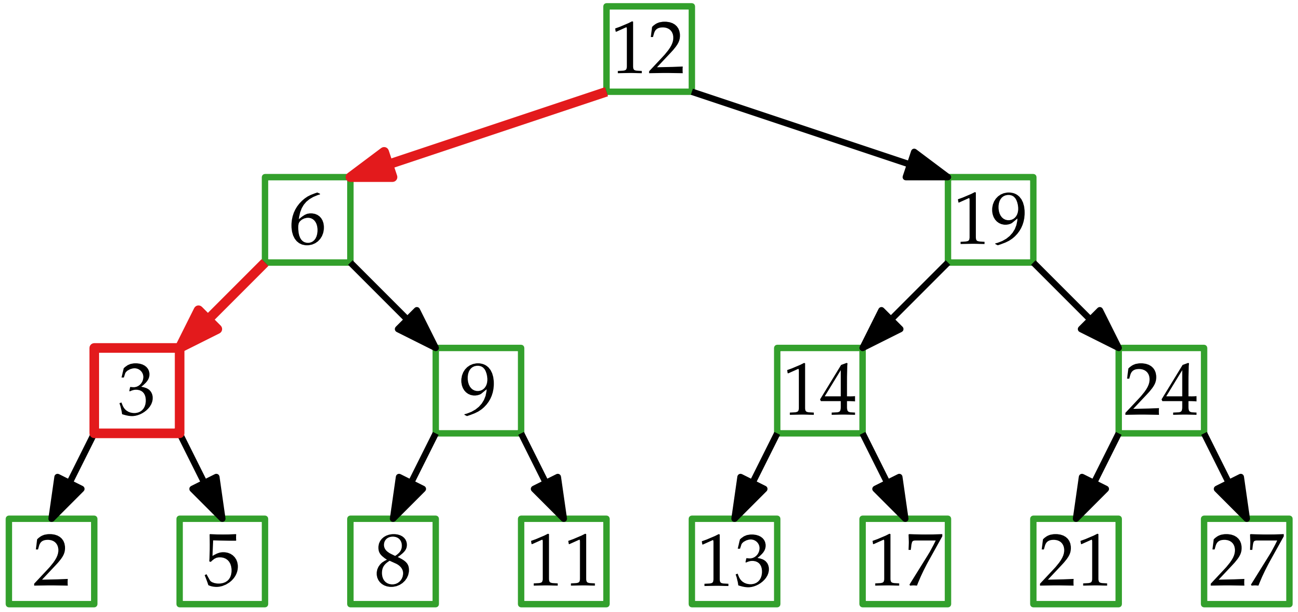


How Good is a Binary Search Tree?

optimal

Binary search tree: w.c. query time $\Theta(n)$

Balanced binary search tree: w.c. query time $\Theta(\log n)$
(e.g. Red-Black-Tree)



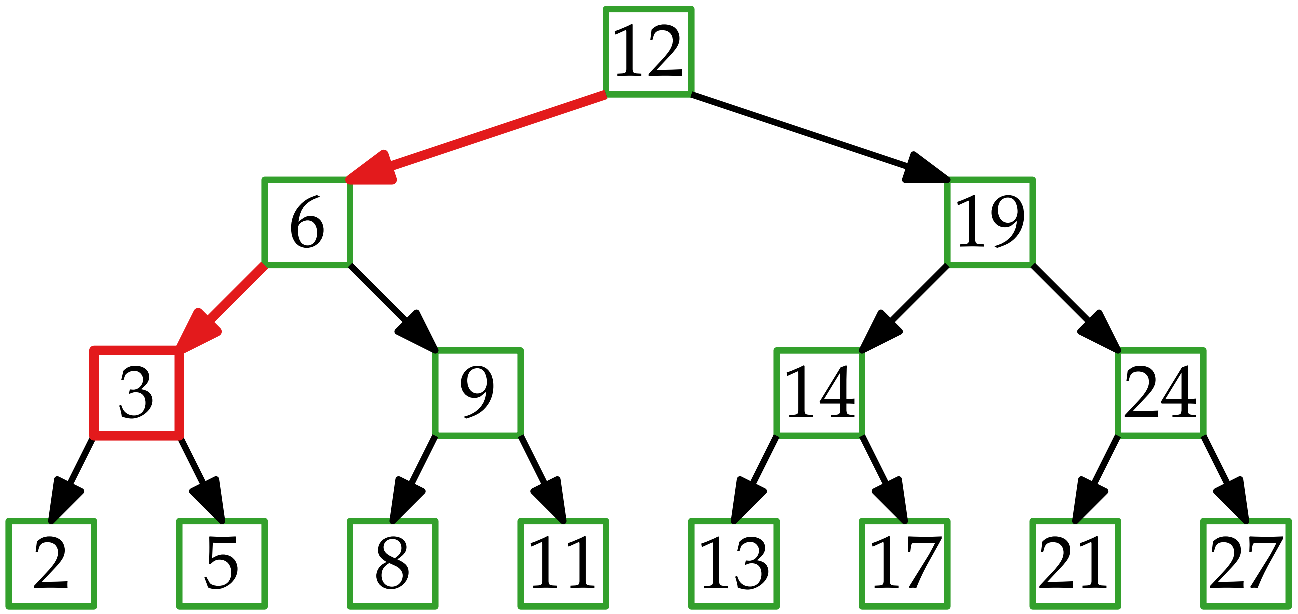
How Good is a Binary Search Tree?

Binary search tree: w.c. query time $\Theta(n)$

optimal

Balanced binary search tree: w.c. query time $\Theta(\log n)$
(e.g. Red-Black-Tree)

What if we *know* the query before?



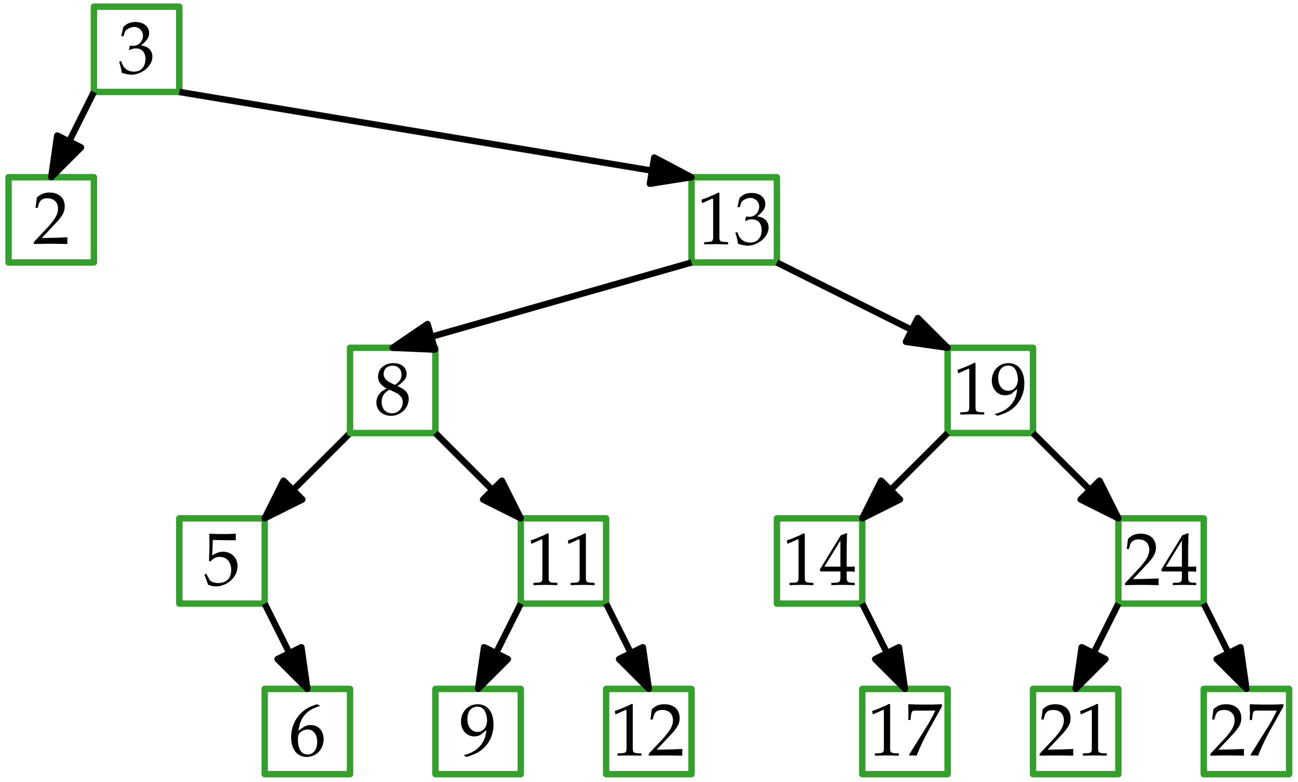
How Good is a Binary Search Tree?

Binary search tree: w.c. query time $\Theta(n)$

optimal

Balanced binary search tree: w.c. query time $\Theta(\log n)$
(e.g. Red-Black-Tree)

What if we *know* the query before?



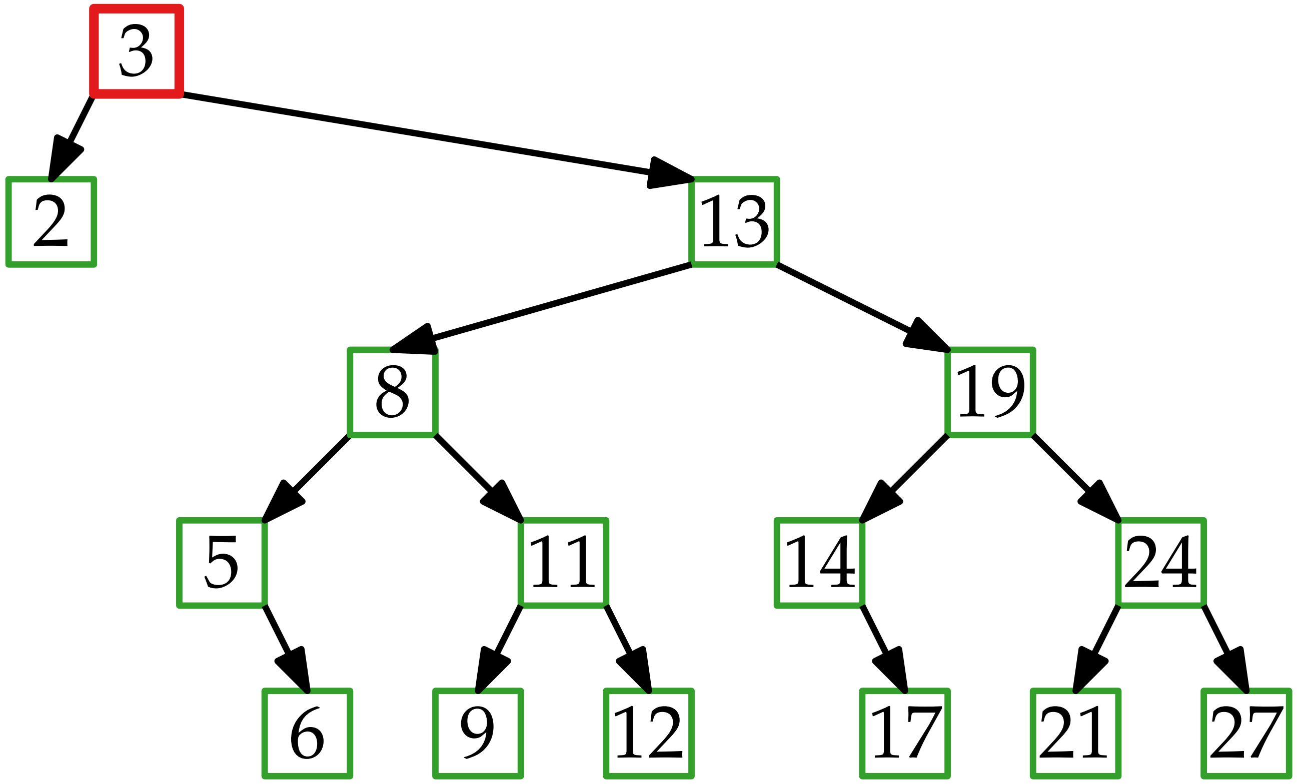
How Good is a Binary Search Tree?

optimal

Binary search tree: w.c. query time $\Theta(n)$

Balanced binary search tree: w.c. query time $\Theta(\log n)$
(e.g. Red-Black-Tree)

What if we *know* the query before?



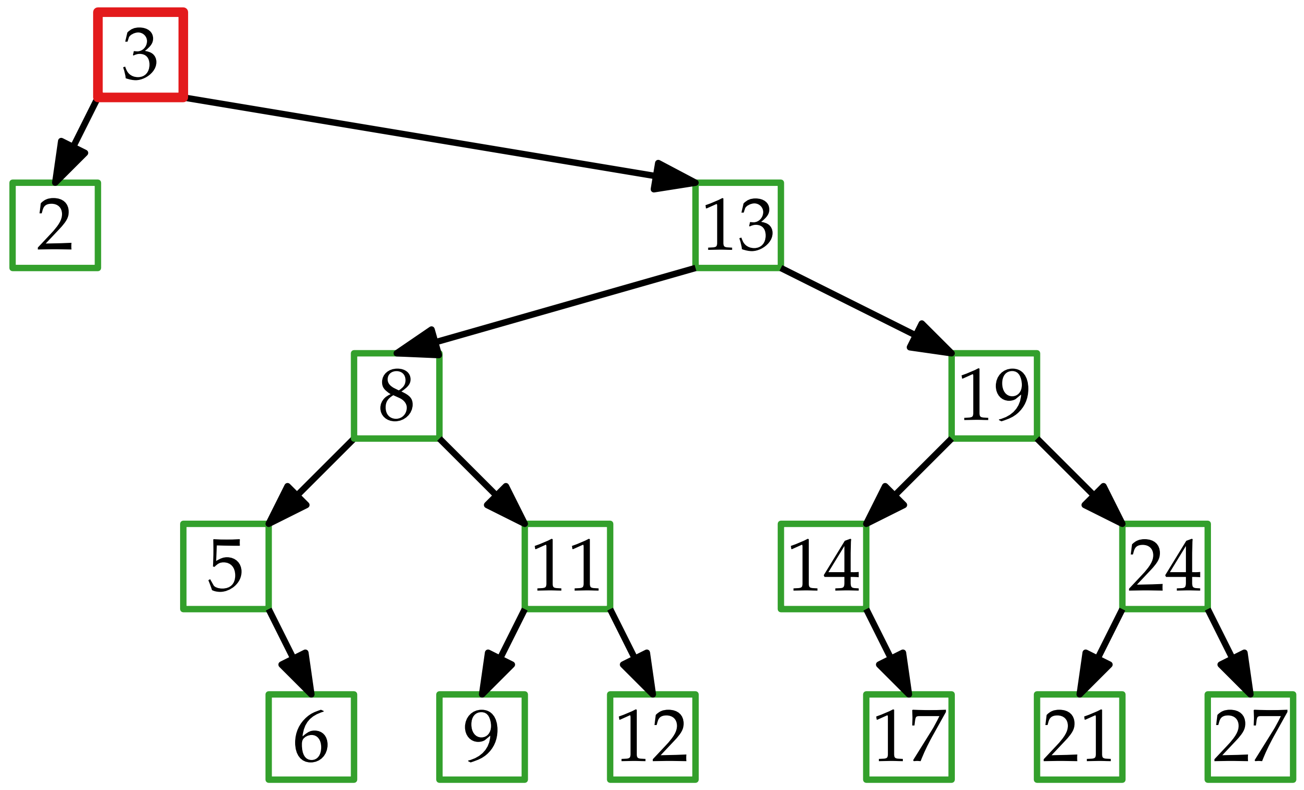
How Good is a Binary Search Tree?

optimal

Binary search tree: w.c. query time $\Theta(n)$

Balanced binary search tree: w.c. query time $\Theta(\log n)$
(e.g. Red-Black-Tree)

What if we *know* the query before? w.c. query time 1



How Good is a Binary Search Tree?

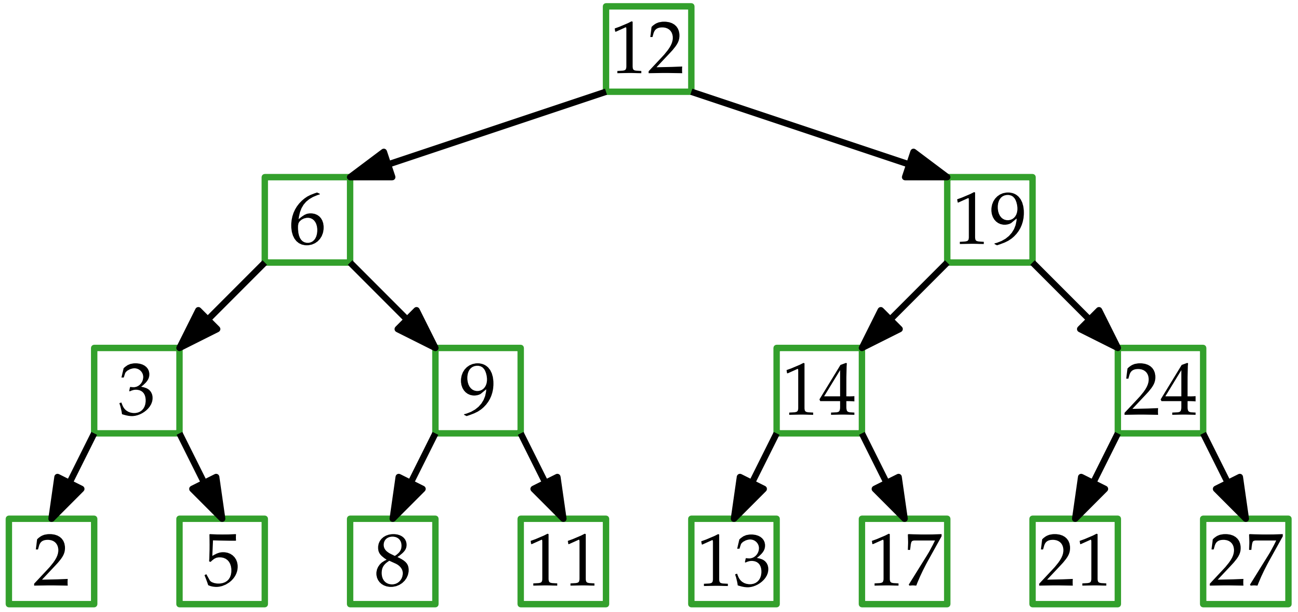
optimal

Binary search tree: w.c. query time $\Theta(n)$

Balanced binary search tree: w.c. query time $\Theta(\log n)$
(e.g. Red-Black-Tree)

What if we *know* the query before? w.c. query time 1

Sequence of queries?



How Good is a Binary Search Tree?

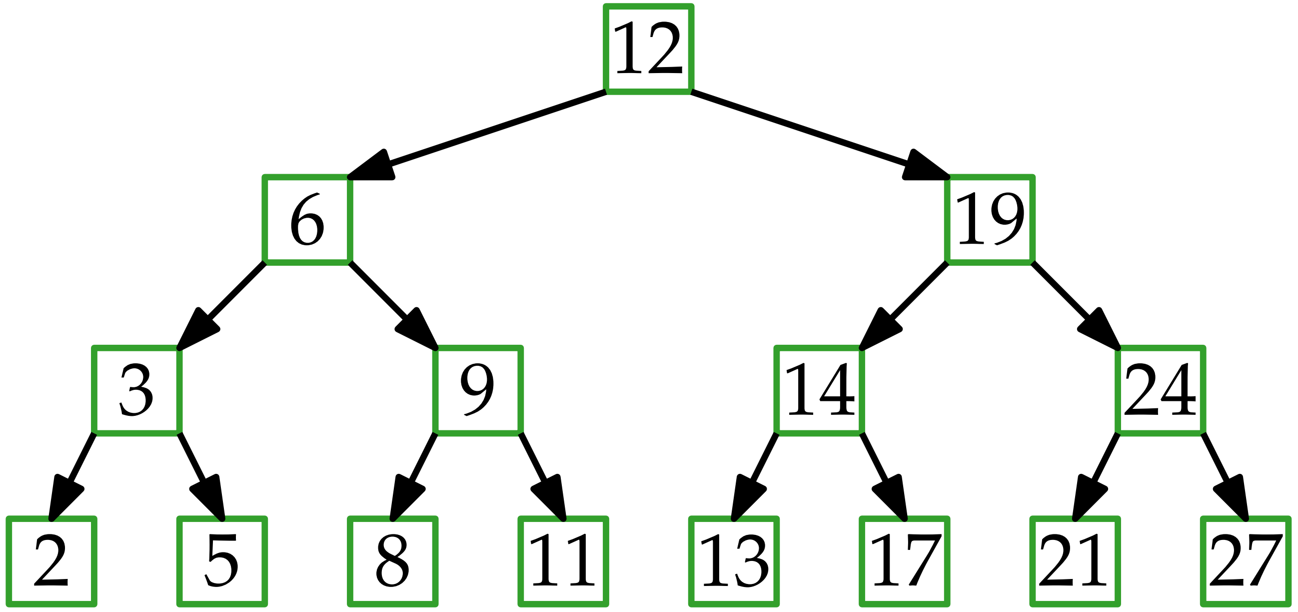
optimal

Binary search tree: w.c. query time $\Theta(n)$

Balanced binary search tree: w.c. query time $\Theta(\log n)$
(e.g. Red-Black-Tree)

What if we *know* the query before? w.c. query time 1

Sequence of queries?
e.g. 2—13—5

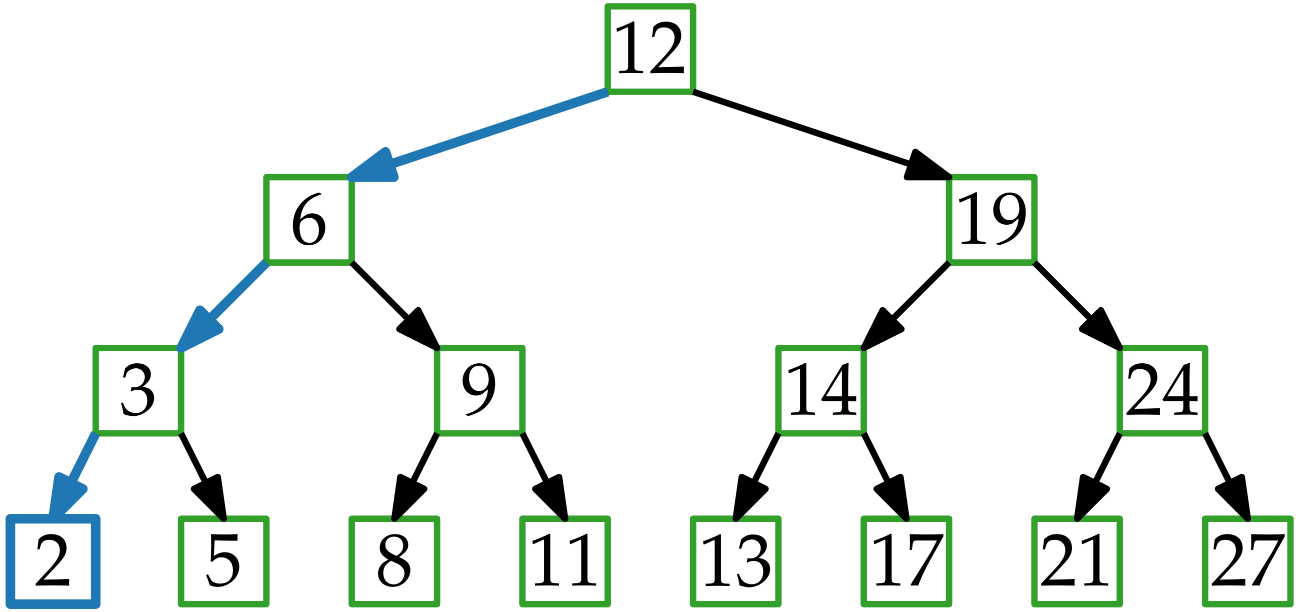


How Good is a Binary Search Tree?

optimal

Binary search tree:	w.c. query time $\Theta(n)$
Balanced binary search tree: (e.g. Red-Black-Tree)	w.c. query time $\Theta(\log n)$
What if we *know* the query before?	w.c. query time 1

Sequence of queries?
e.g. 2—13—5



How Good is a Binary Search Tree?

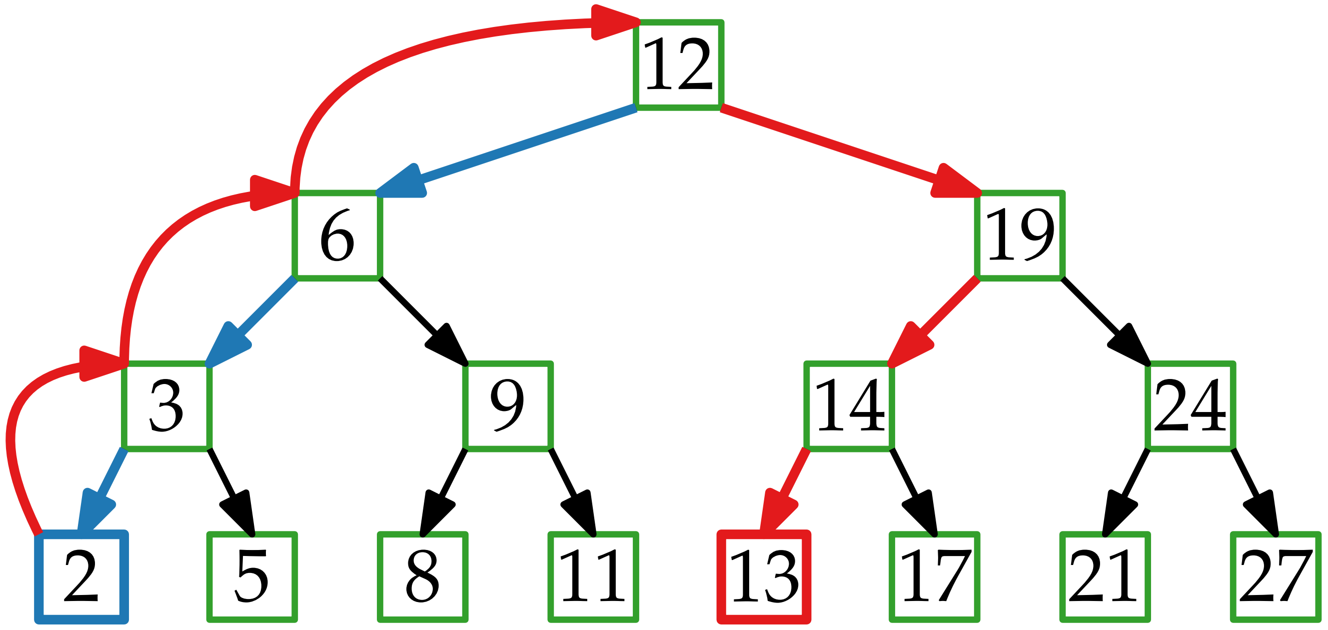
optimal

Binary search tree: w.c. query time $\Theta(n)$

Balanced binary search tree: w.c. query time $\Theta(\log n)$
(e.g. Red-Black-Tree)

What if we *know* the query before? w.c. query time 1

Sequence of queries?
e.g. 2—13—5



How Good is a Binary Search Tree?

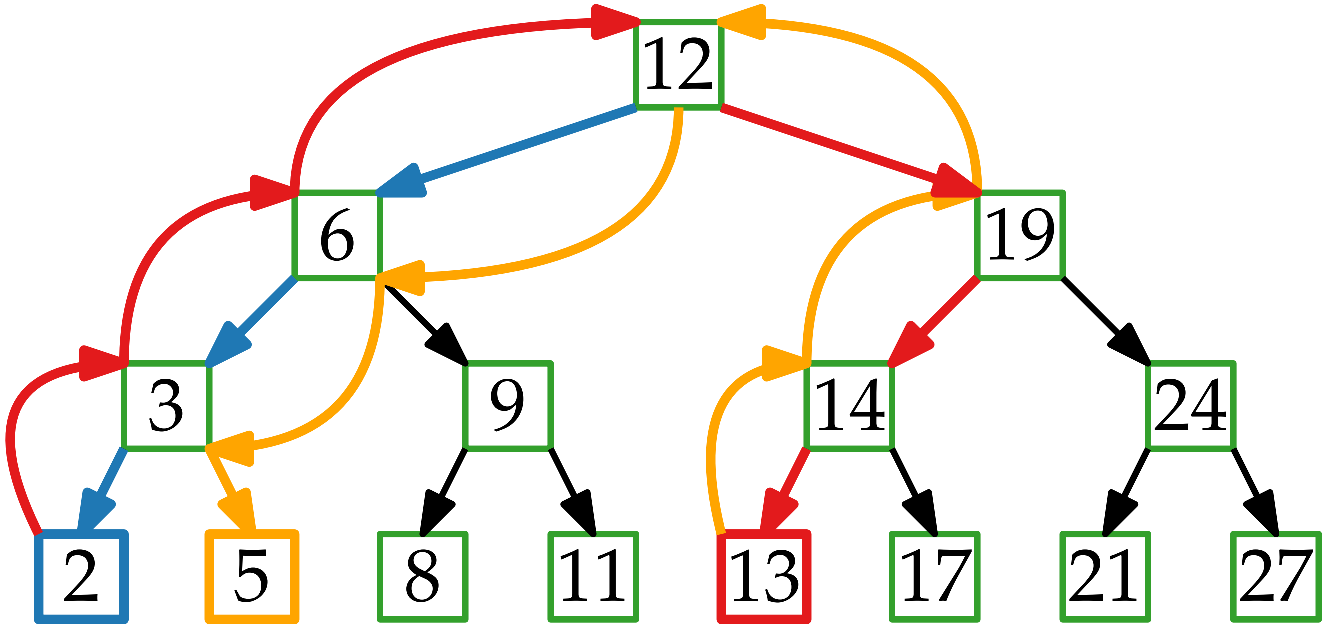
optimal

Binary search tree: w.c. query time $\Theta(n)$

Balanced binary search tree: w.c. query time $\Theta(\log n)$
(e.g. Red-Black-Tree)

What if we *know* the query before? w.c. query time 1

Sequence of queries?
e.g. 2—13—5



How Good is a Binary Search Tree?

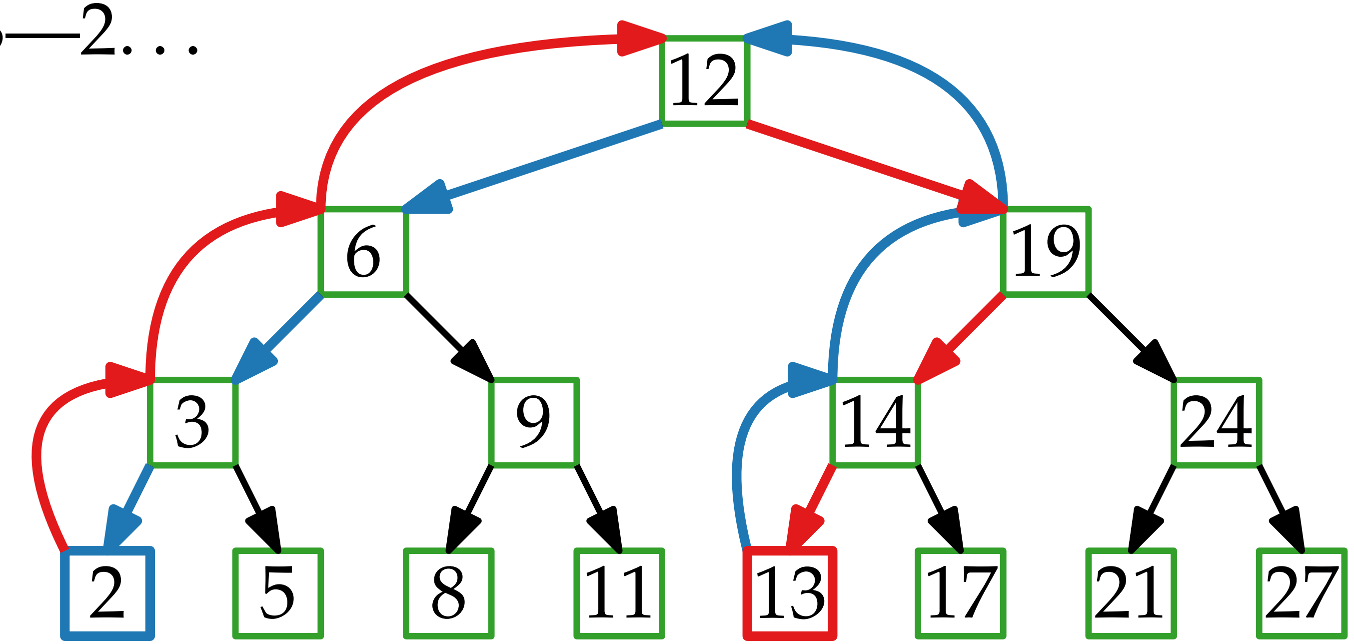
optimal

Binary search tree: w.c. query time $\Theta(n)$

Balanced binary search tree: w.c. query time $\Theta(\log n)$
(e.g. Red-Black-Tree)

What if we *know* the query before? w.c. query time 1

Sequence of queries?
e.g. 2—13—5
or 2—13—2—13—2. . .



How Good is a Binary Search Tree?

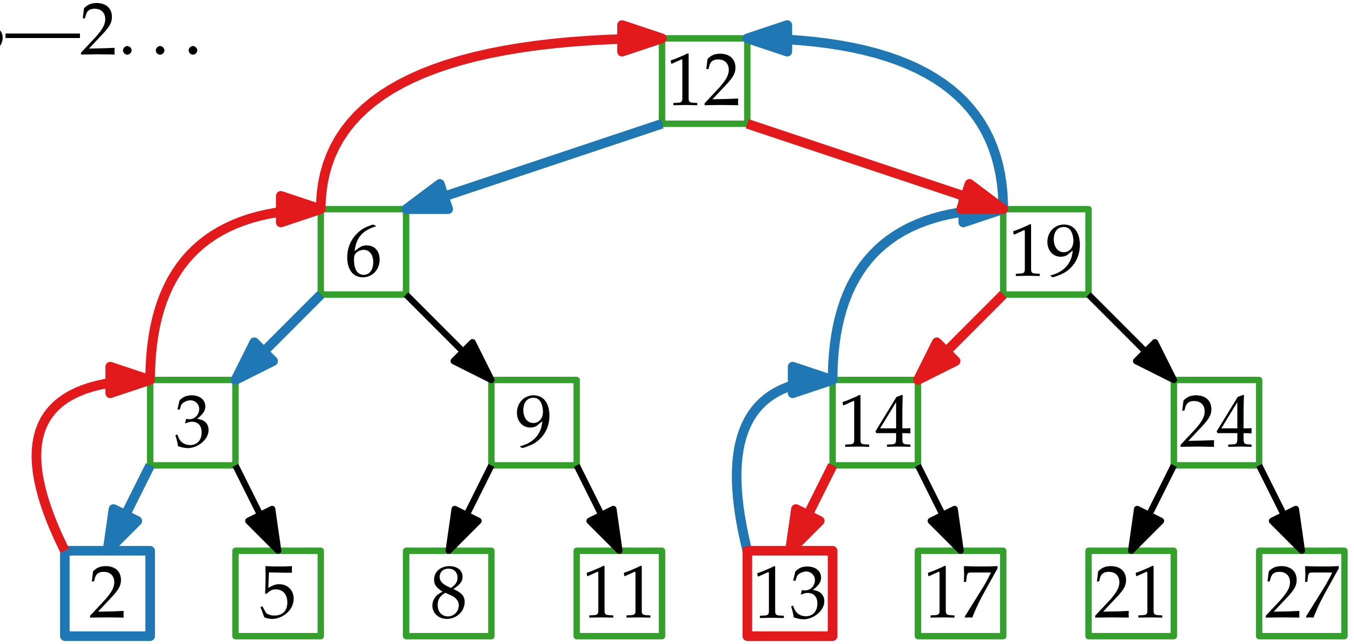
optimal

Binary search tree: w.c. query time $\Theta(n)$

Balanced binary search tree: w.c. query time $\Theta(\log n)$
(e.g. Red-Black-Tree)

What if we *know* the query before? w.c. query time 1

Sequence of queries? $O(\log n)$ per query
e.g. 2—13—5
or 2—13—2—13—2. . .



How Good is a Binary Search Tree?

optimal

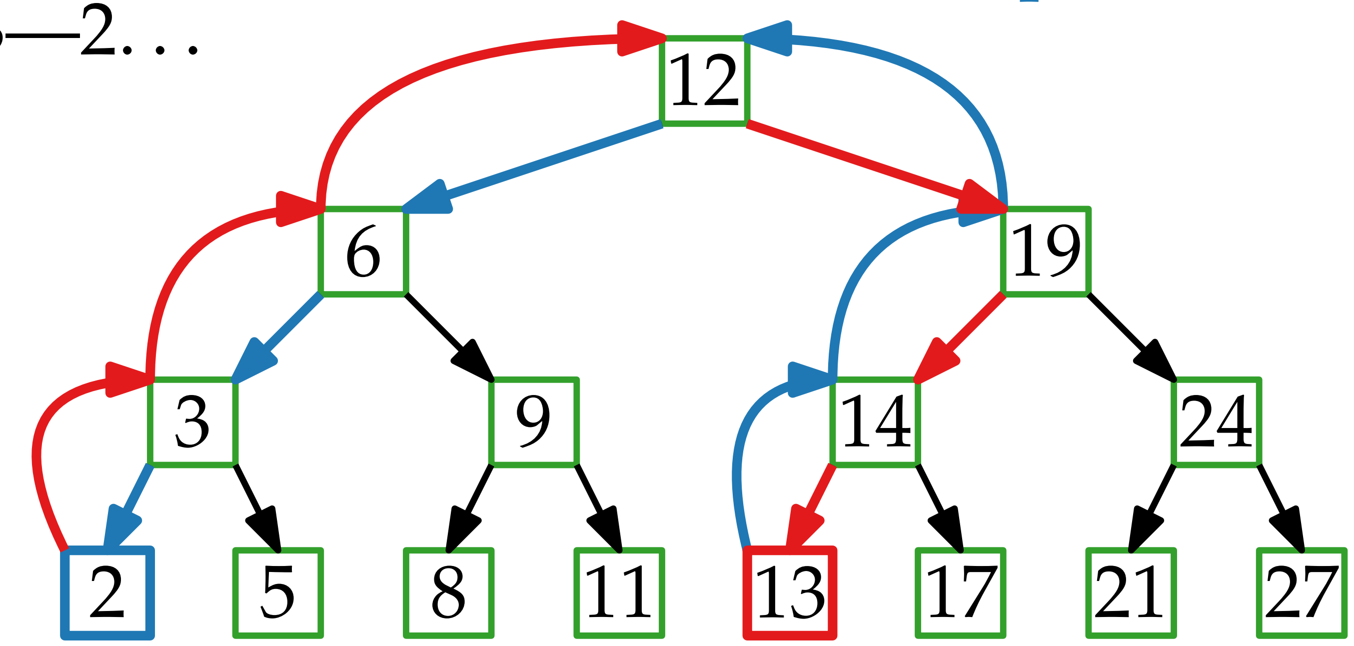
Binary search tree: w.c. query time $\Theta(n)$

Balanced binary search tree: w.c. query time $\Theta(\log n)$
(e.g. Red-Black-Tree)

What if we *know* the query before? w.c. query time 1

Sequence of queries? $O(\log n)$ per query
e.g. 2—13—5
or 2—13—2—13—2...

optimal?



How Good is a Binary Search Tree?

optimal

Binary search tree: w.c. query time $\Theta(n)$

Balanced binary sea)

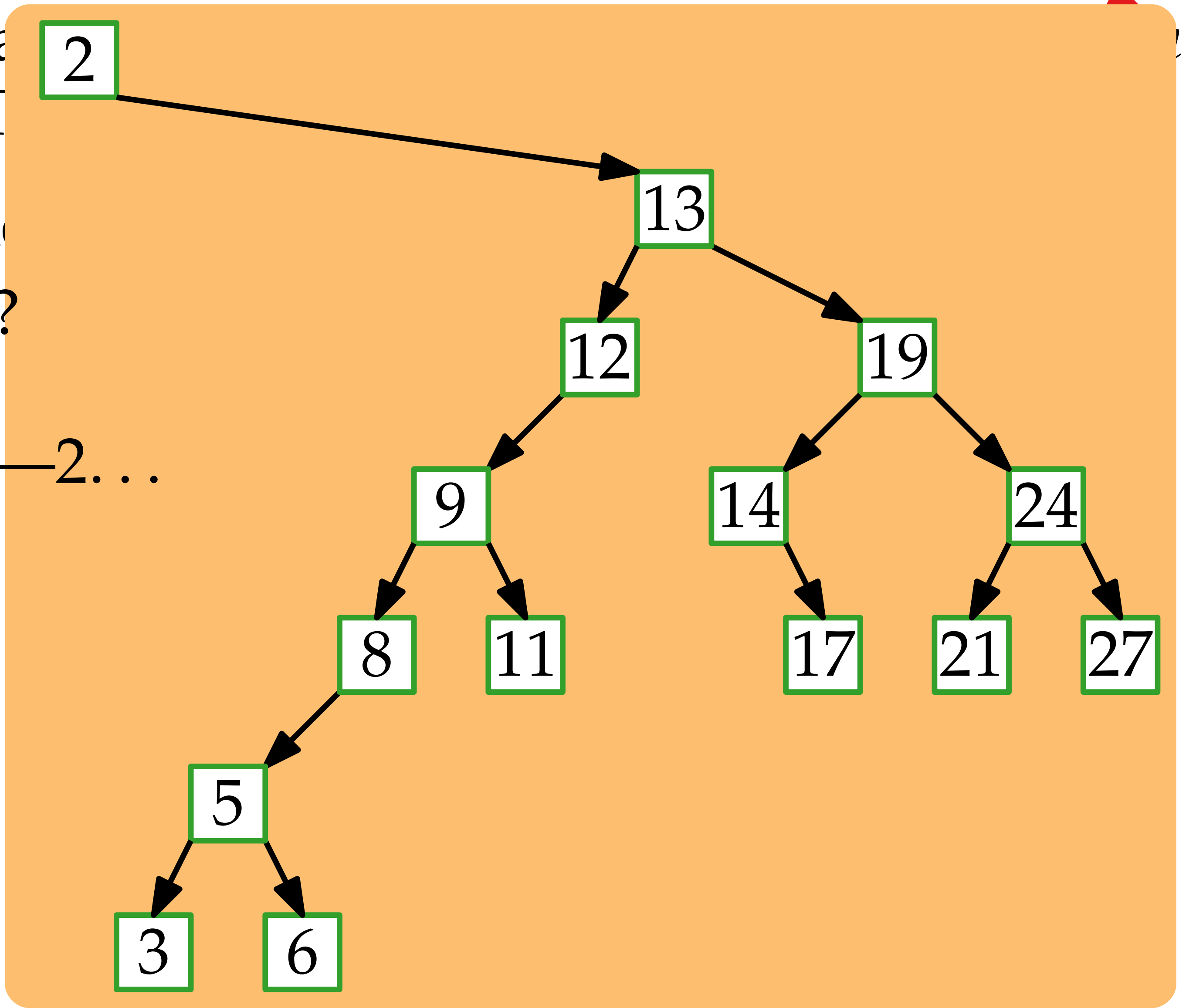
(e.g. Red-Black-T

What if we *know* th

Sequence of queries?

e.g. 2—13—5

or 2—13—2—13—2. . .



How Good is a Binary Search Tree?

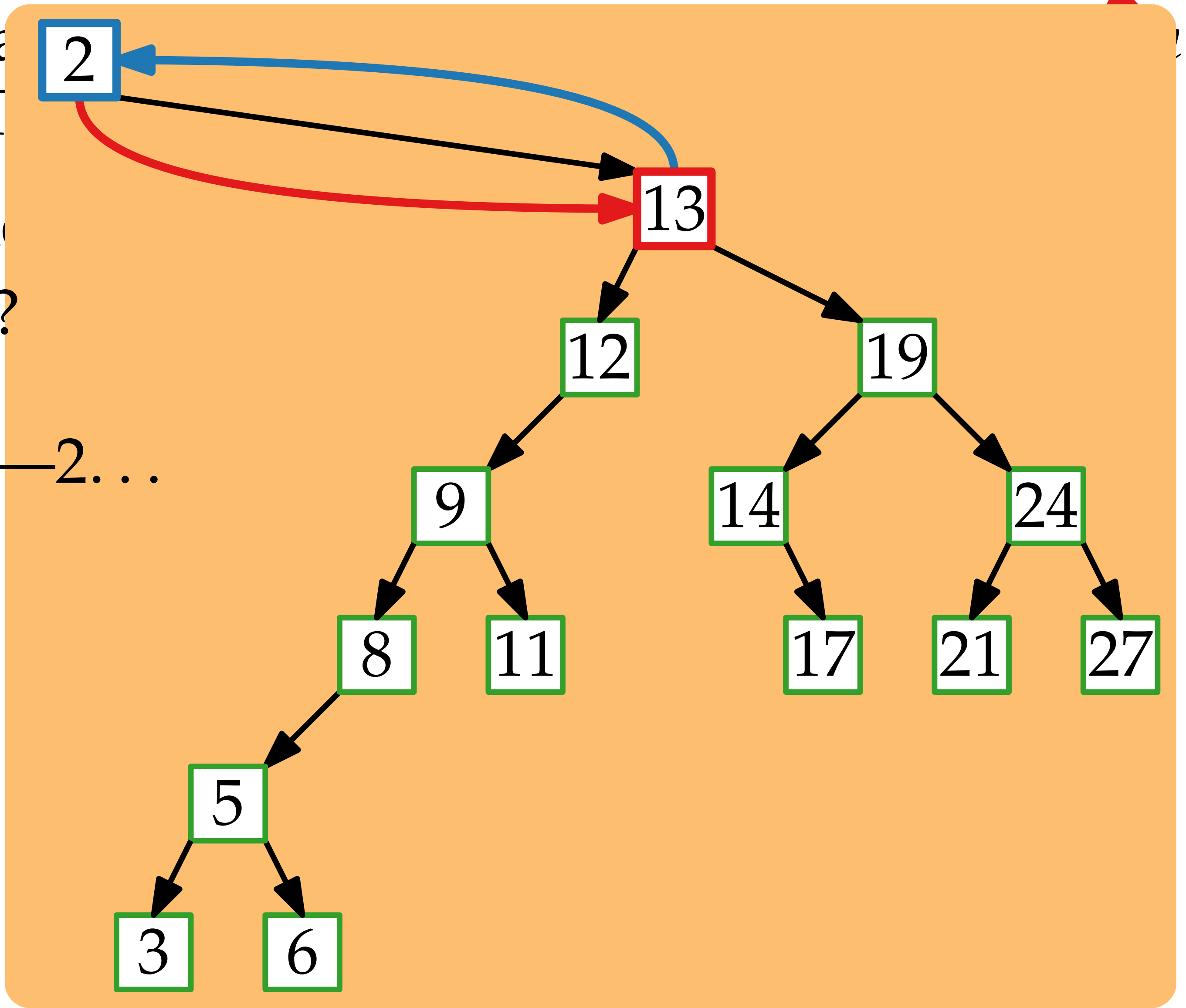
Binary search tree: w.c. query time $\Theta(n)$

optimal

Balanced binary sea[...] [...]$)$
(e.g. Red-Black-T[...]

What if we *know* th[...]

Sequence of queries?
e.g. 2—13—5
or 2—13—2—13—2. . .



How Good is a Binary Search Tree?

optimal

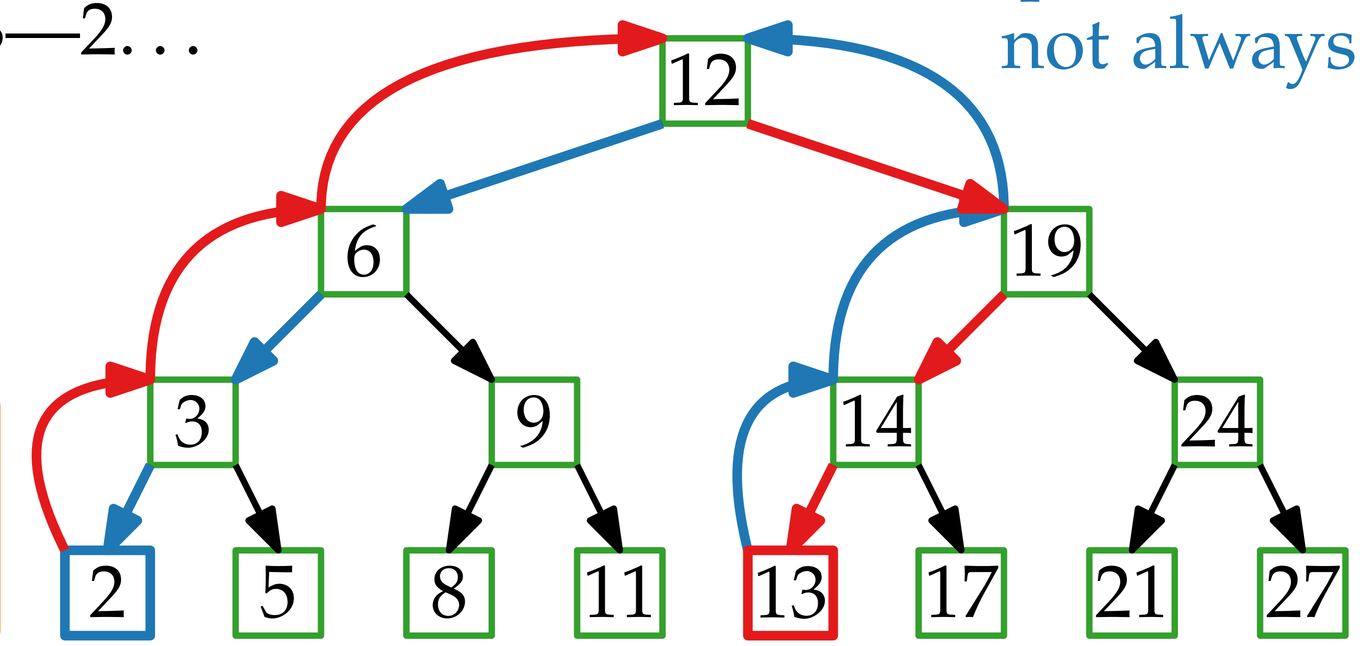
Binary search tree: w.c. query time $\Theta(n)$

Balanced binary search tree: w.c. query time $\Theta(\log n)$
(e.g. Red-Black-Tree)

What if we *know* the query before? w.c. query time 1

Sequence of queries? $O(\log n)$ per query
e.g. 2—13—5
or 2—13—2—13—2...

optimal?
not always!



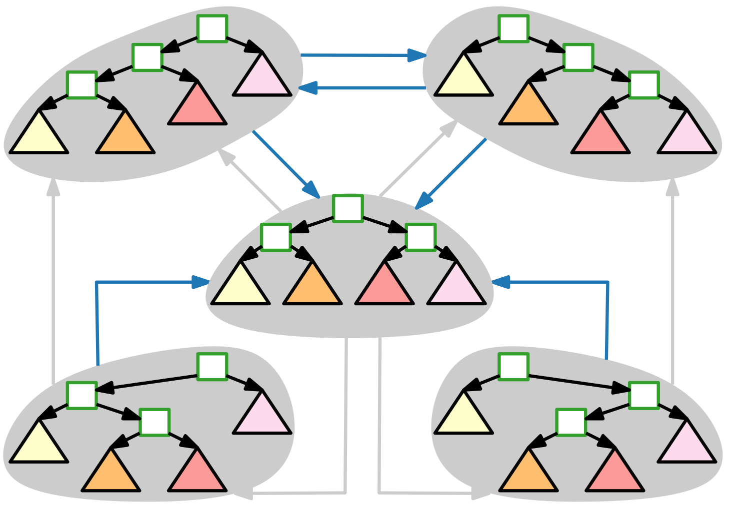
The performance of a BST depends on the model!

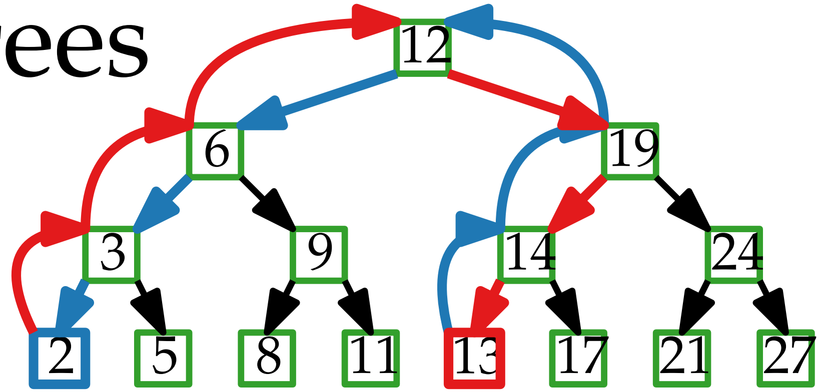
Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Advanced Algorithms

Optimal Binary Search Trees

Splay Trees





Philipp Kindermann · WS20

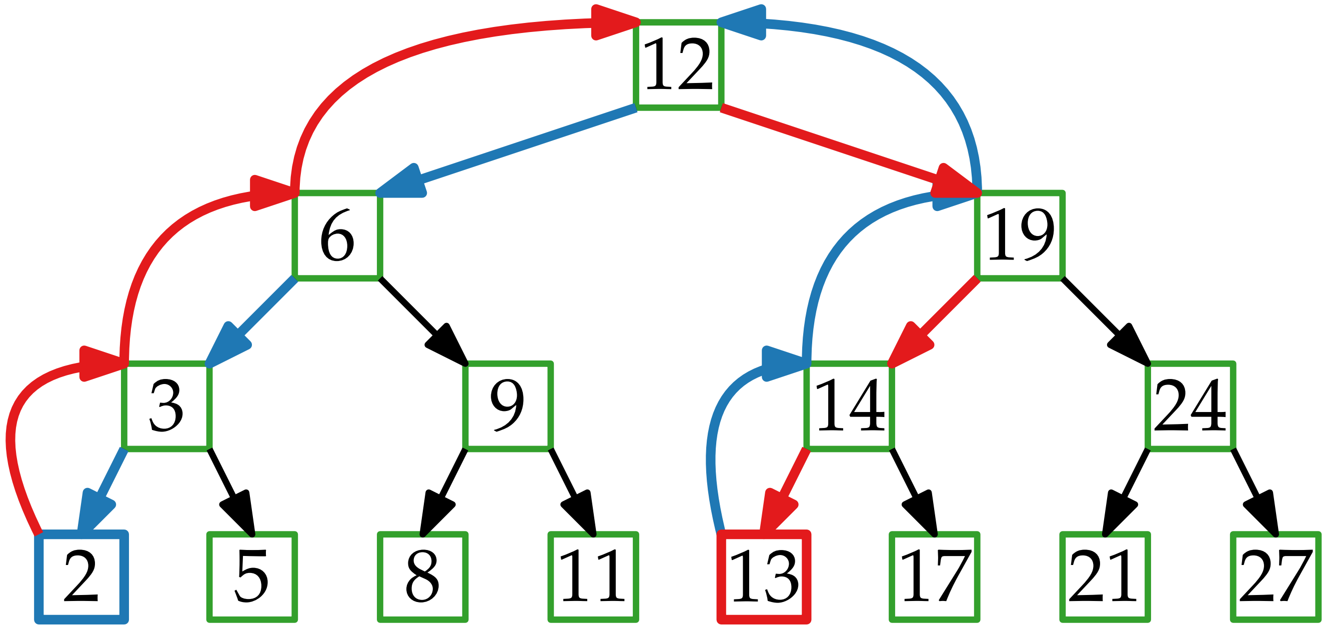
Part II:
Models of Optimality

Model 1: Malicious Queries

Given a BST, what is the worst sequence of queries?

Model 1: Malicious Queries

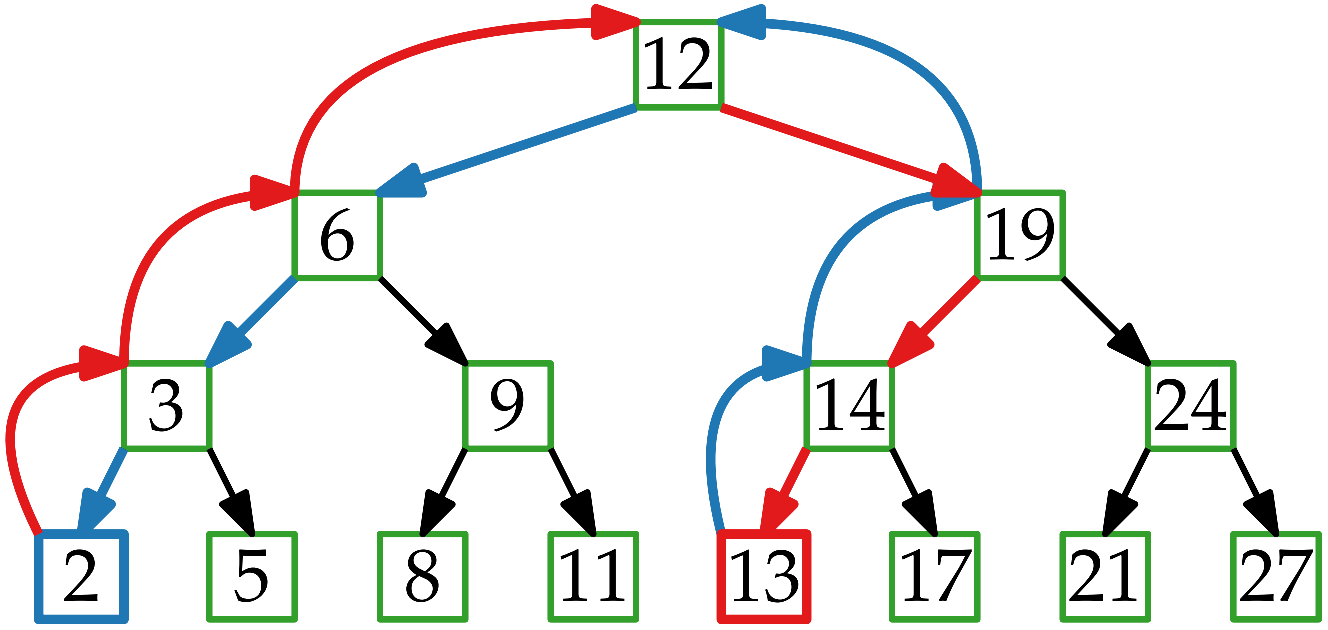
Given a BST, what is the worst sequence of queries?



Model 1: Malicious Queries

Given a BST, what is the worst sequence of queries?

Lemma. The worst-case malicious query cost in any BST with n nodes is at least $\Omega(\log n)$ per query.

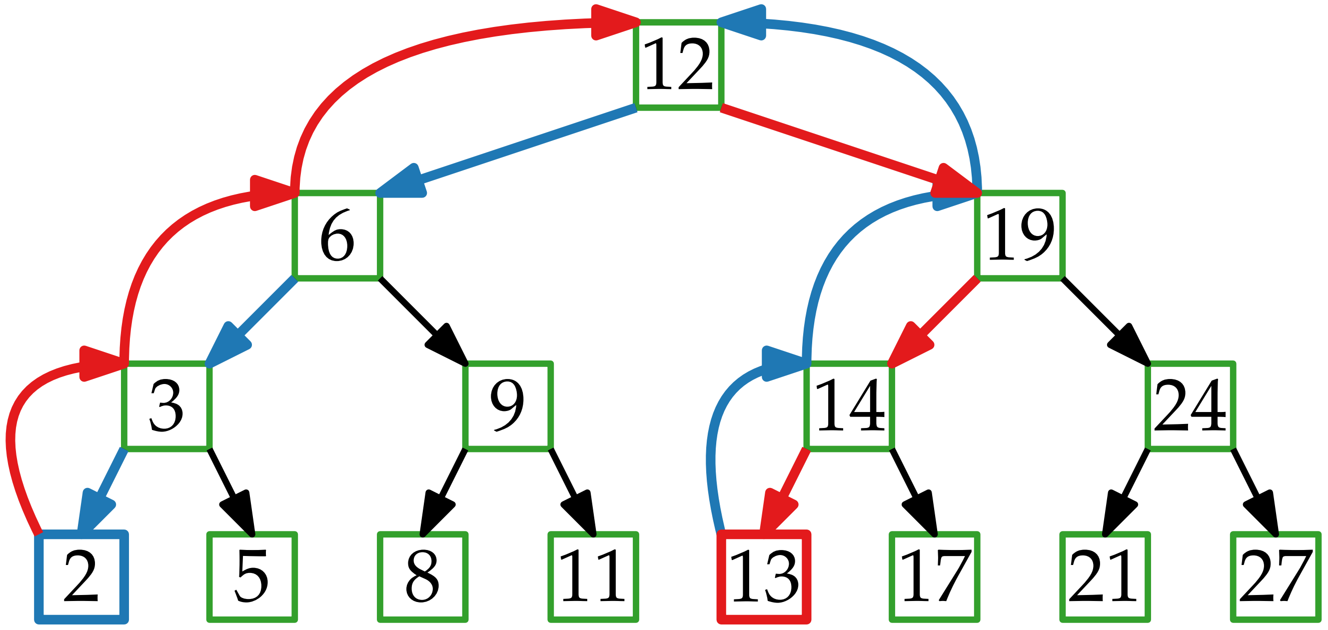


Model 1: Malicious Queries

Given a BST, what is the worst sequence of queries?

Lemma. The worst-case malicious query cost in any BST with n nodes is at least $\Omega(\log n)$ per query.

Definition. A BST is **balanced** if the cost of *any* sequence of m queries is $O(m \log n + n \log n)$.

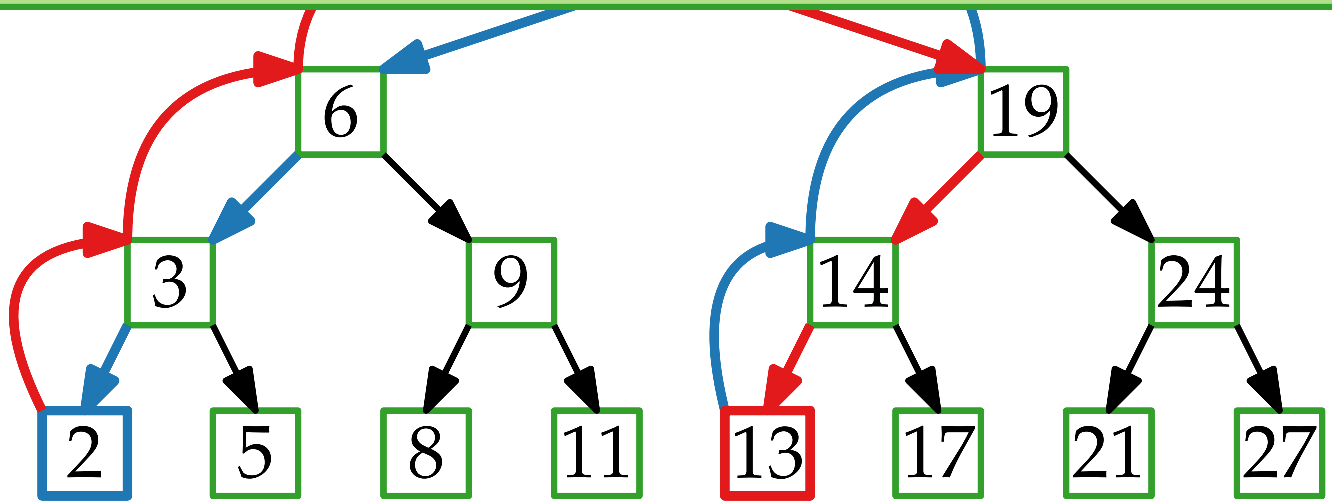


Model 1: Malicious Queries

Given a BST, what is the worst sequence of queries?

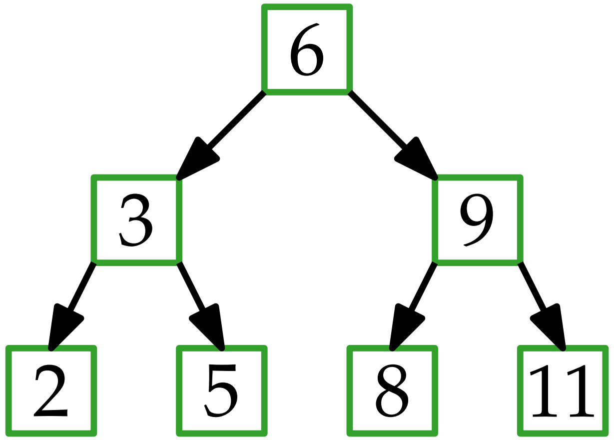
Lemma. The worst-case malicious query cost in any BST with n nodes is at least $\Omega(\log n)$ per query.

Definition. A BST is **balanced** if the cost of *any* sequence of m queries is $O(m \log n + n \log n)$.
$\Rightarrow$ the (amortized) cost of each query is $O(\log n)$ (for at least n queries)



Model 2: Known Probability Distribution

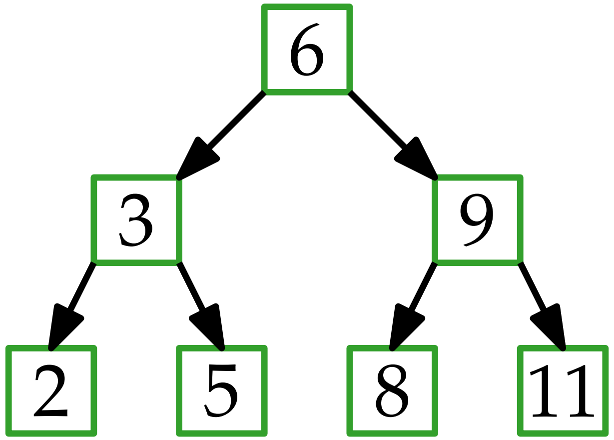
Model 2: Known Probability Distribution



Model 2: Known Probability Distribution

Access Probabilities:

2	3	5	6	8	9	11
2%	20%	30%	8%	20%	15%	5%

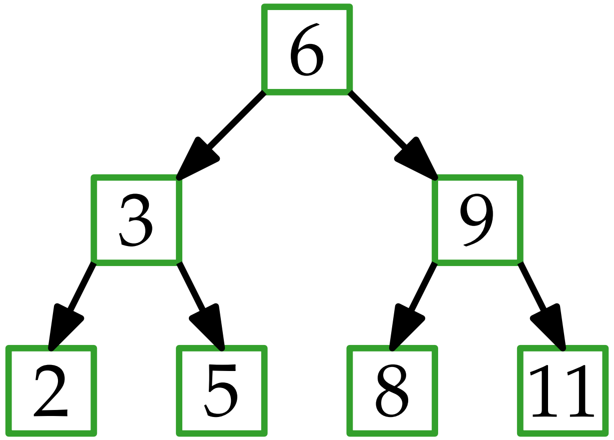


Model 2: Known Probability Distribution

Access Probabilities:

2	3	5	6	8	9	11
2%	20%	30%	8%	20%	15%	5%

Idea: Place nodes with higher probability higher in the tree.

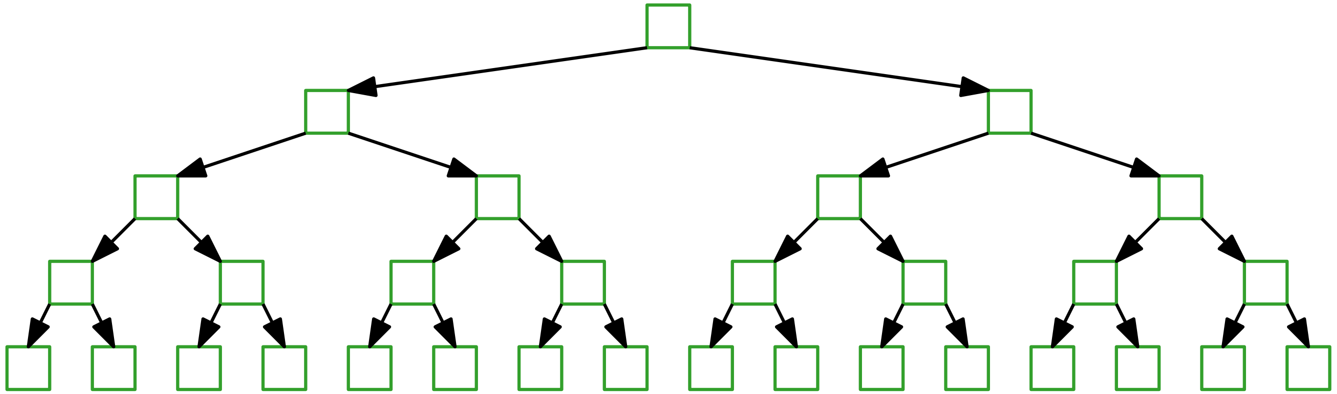


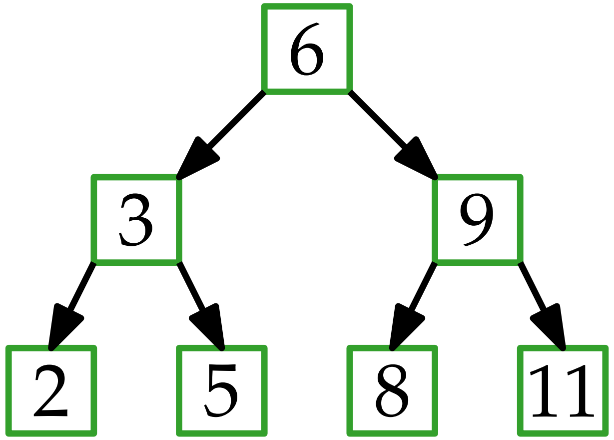
Model 2: Known Probability Distribution

Access Probabilities:

2	3	5	6	8	9	11
2%	20%	30%	8%	20%	15%	5%

Idea: Place nodes with higher probability higher in the tree.



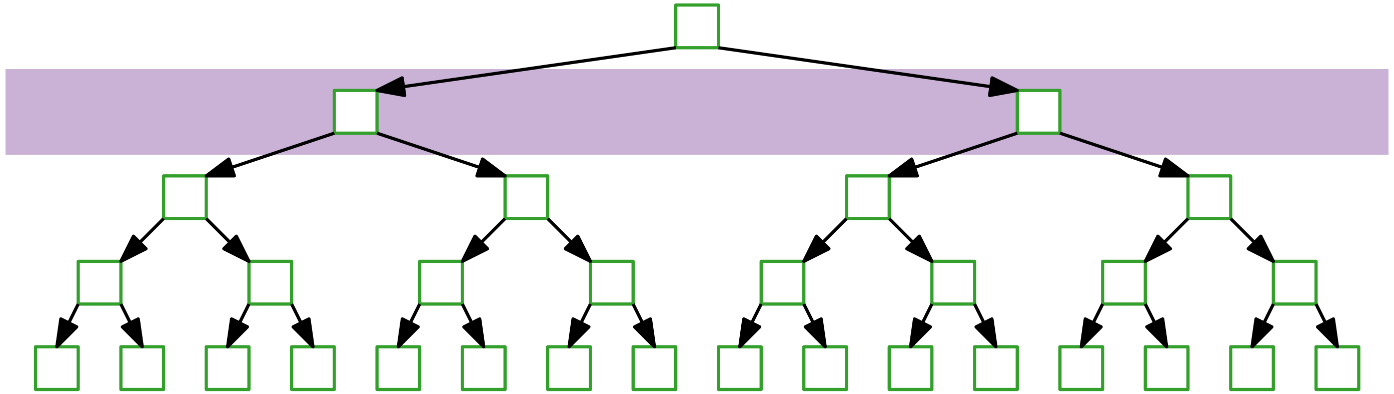


Model 2: Known Probability Distribution

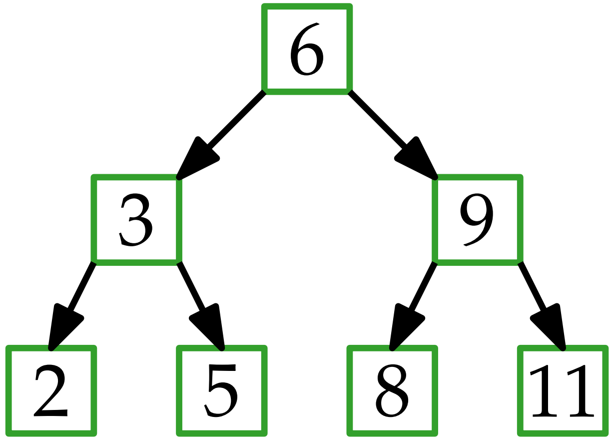
Access Probabilities:

2	3	5	6	8	9	11
2%	20%	30%	8%	20%	15%	5%

Idea: Place nodes with higher probability higher in the tree.



prob. $\leq 1/2$

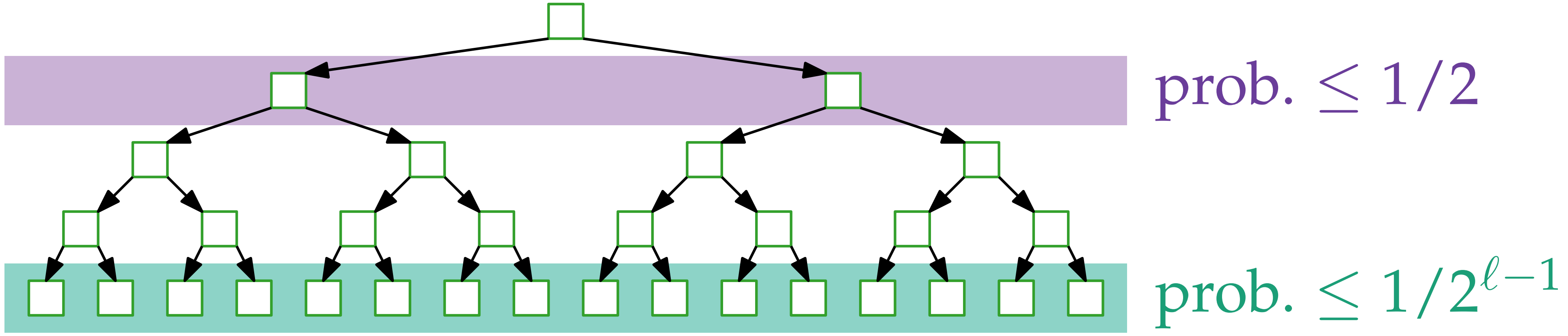


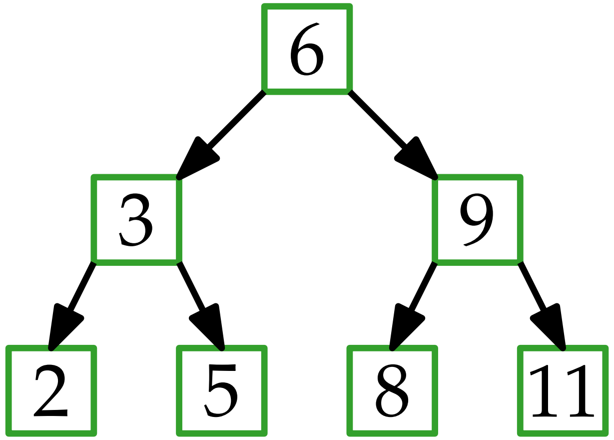
Model 2: Known Probability Distribution

Access Probabilities:

2	3	5	6	8	9	11
2%	20%	30%	8%	20%	15%	5%

Idea: Place nodes with higher probability higher in the tree.



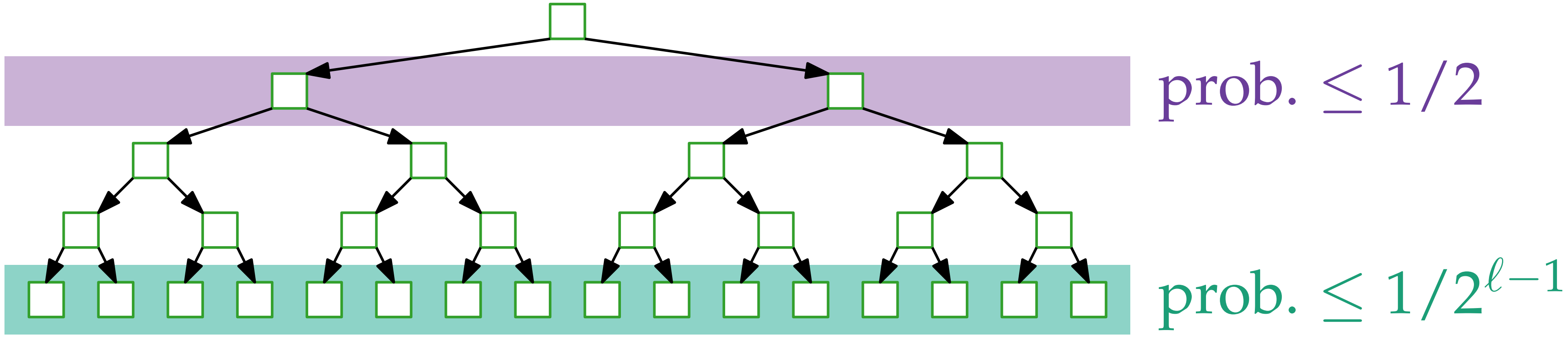


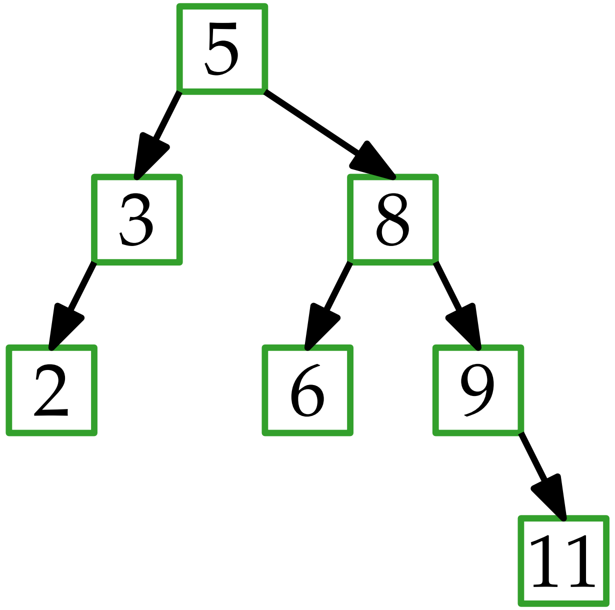
Model 2: Known Probability Distribution

Access Probabilities:

2	3	5	6	8	9	11
2%	20%	30%	8%	20%	15%	5%

Idea: Place nodes with higher probability higher in the tree.



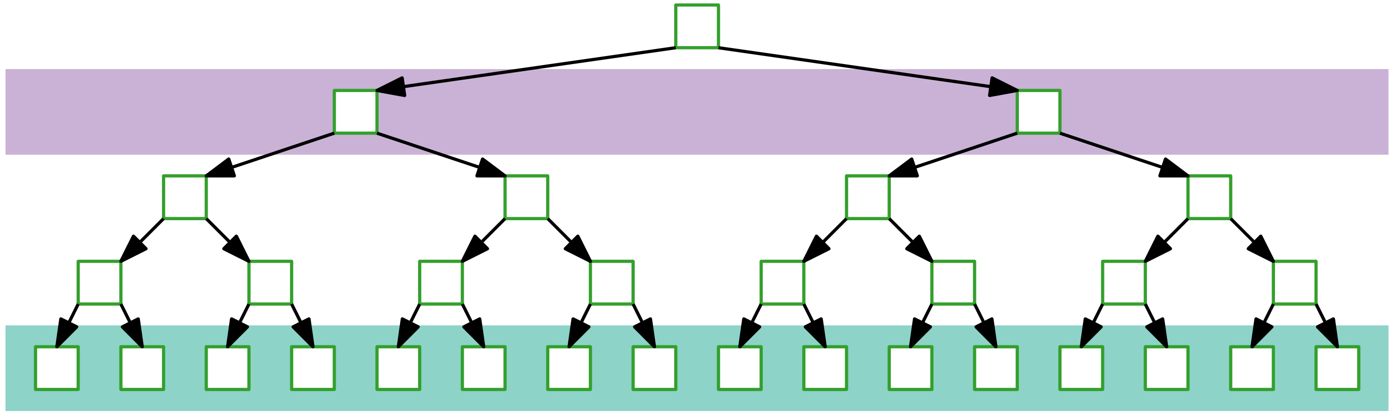


Model 2: Known Probability Distribution

Access Probabilities:

2	3	5	6	8	9	11
2%	20%	30%	8%	20%	15%	5%

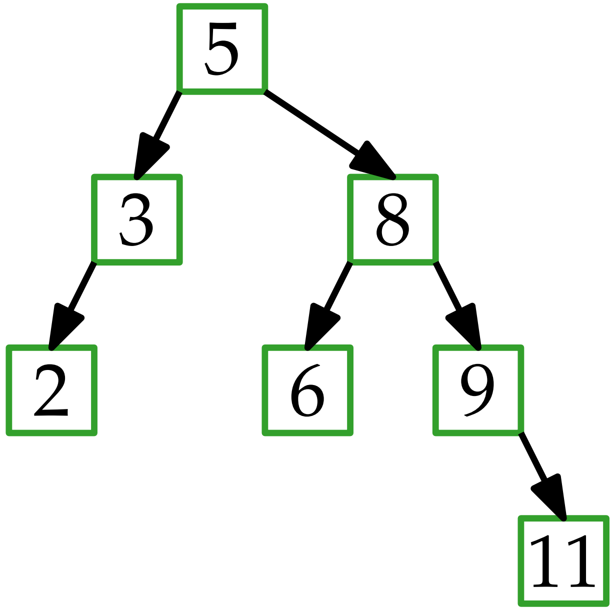
Idea: Place nodes with higher probability higher in the tree.



prob. $\leq 1/2$

OPT: prob. $p \Rightarrow$ level

prob. $\leq 1/2^{\ell-1}$

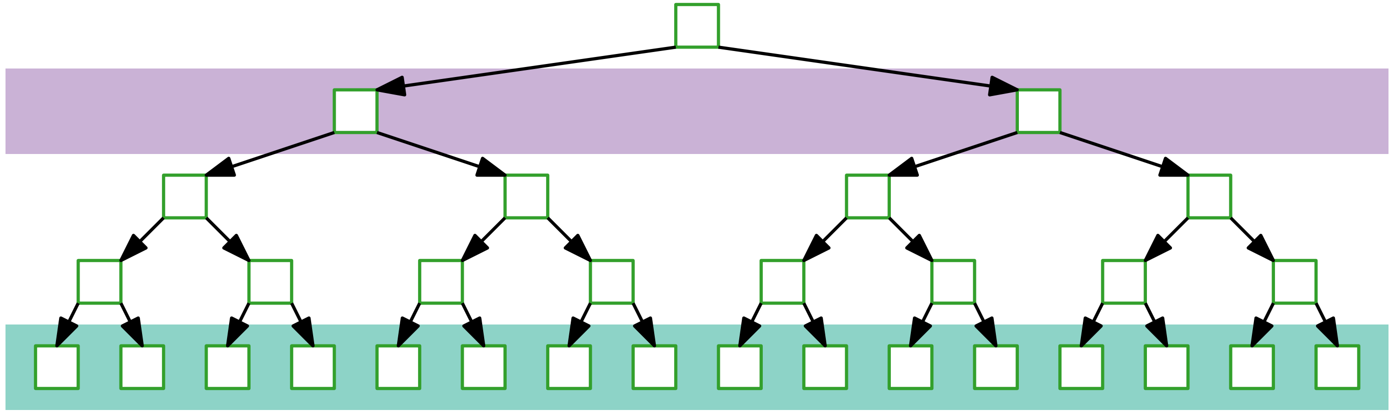


Model 2: Known Probability Distribution

Access Probabilities:

2	3	5	6	8	9	11
2%	20%	30%	8%	20%	15%	5%

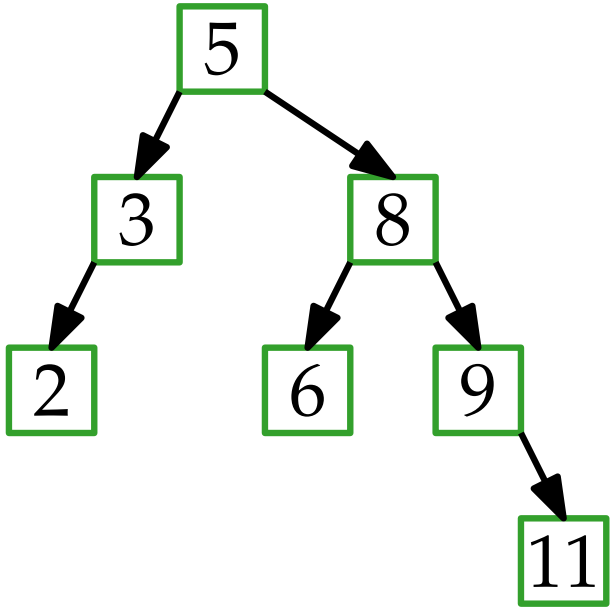
Idea: Place nodes with higher probability higher in the tree.



prob. $\leq 1/2$

OPT: prob. $p \Rightarrow$ level $\log(1/p)$

prob. $\leq 1/2^{\ell-1}$

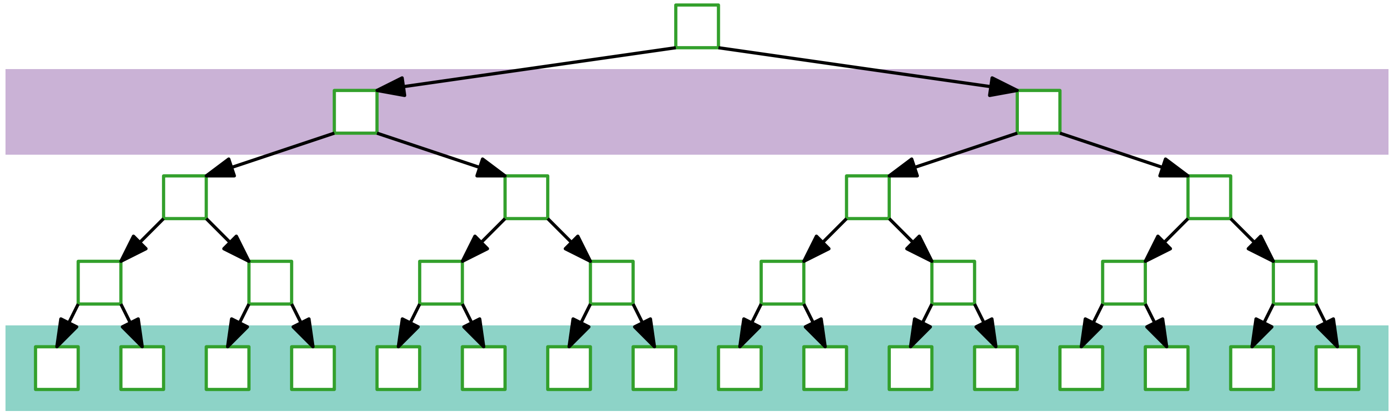


Model 2: Known Probability Distribution

Access Probabilities:

2	3	5	6	8	9	11
2%	20%	30%	8%	20%	15%	5%

Idea: Place nodes with higher probability higher in the tree.



prob. $\leq 1/2$

OPT: prob. $p \Rightarrow$ level $\log(1/p)$

prob. $\leq 1/2^{\ell-1}$

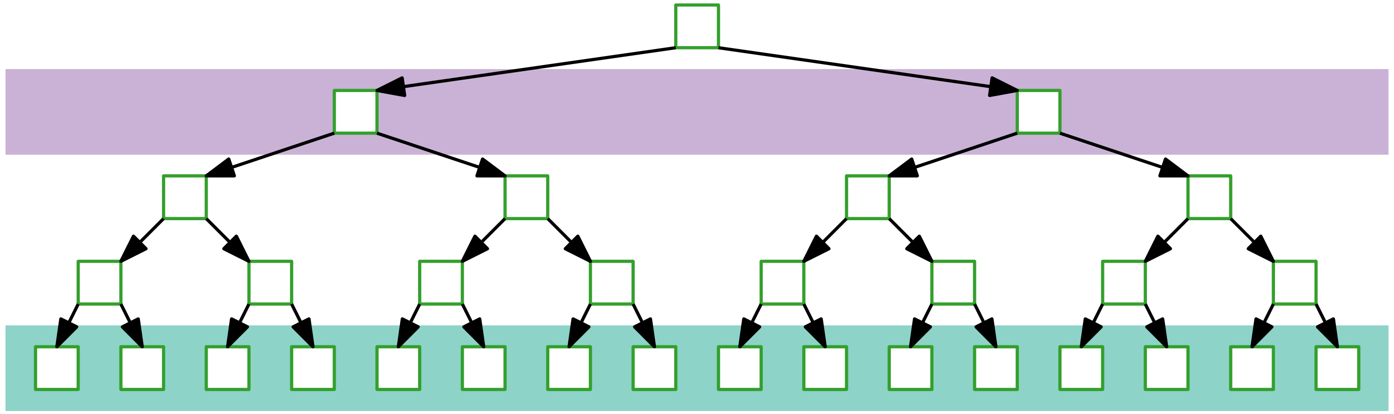
Lemma. The expected query cost in any BST is at least $\Omega(1 + H)$ per query with $H = \sum_{i=1}^{n} p_i \log(1/p_i)$.

Model 2: Known Probability Distribution

Access Probabilities:

2	3	5	6	8	9	11
2%	20%	30%	8%	20%	15%	5%

Idea: Place nodes with higher probability higher in the tree.



prob. $\leq 1/2$

OPT: prob. $p \Rightarrow$ level $\log(1/p)$

prob. $\leq 1/2^{\ell-1}$

Lemma. The expected query cost in any BST is at least $\Omega(1 + H)$ per query with $H = \sum_{i=1}^{n} p_i \log(1/p_i)$.

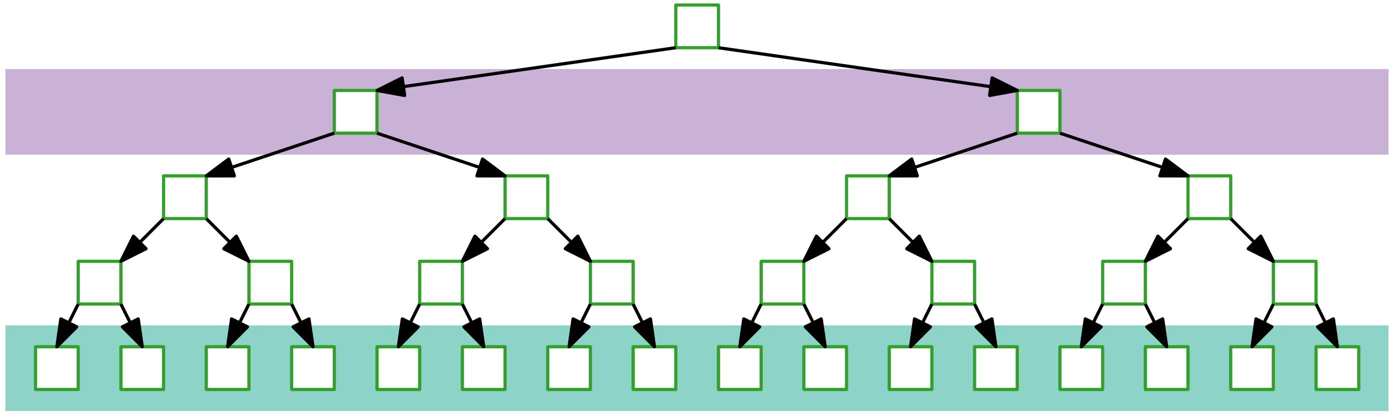
Definition. A BST has the **entropy property** if it reaches this bound.

Model 2: Known Probability Distribution

Access Probabilities:

2	3	5	6	8	9	11
2%	20%	30%	8%	20%	15%	5%

Idea: Place nodes with higher probability higher in the tree.



prob. $\leq 1/2$

OPT: prob. $p \Rightarrow$ level $\log(1/p)$

prob. $\leq 1/2^{\ell-1}$

Lemma. The expected query cost in any BST is at least $\Omega(1 + H)$ per query with $H = \sum_{i=1}^{n} p_i \log(1/p_i)$.

Definition. A BST has the **entropy property** if it reaches this bound.

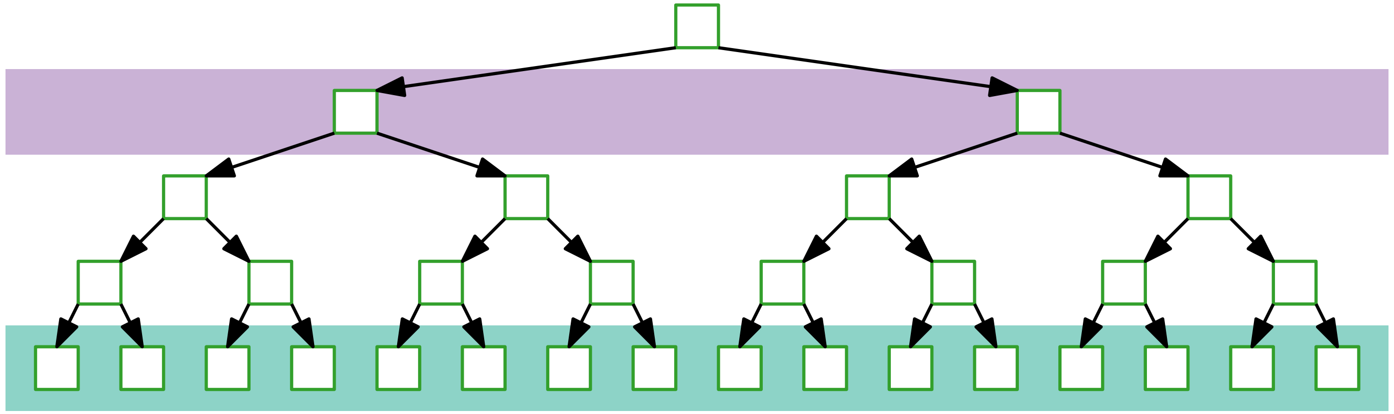
$p_i = 1/n$

Model 2: Known Probability Distribution

Access Probabilities:

2	3	5	6	8	9	11
2%	20%	30%	8%	20%	15%	5%

Idea: Place nodes with higher probability higher in the tree.



prob. $\leq 1/2$

OPT: prob. $p \Rightarrow$ level $\log(1/p)$

prob. $\leq 1/2^{\ell-1}$

Lemma. The expected query cost in any BST is at least $\Omega(1+H)$ per query with $H = \sum_{i=1}^{n} p_i \log(1/p_i)$.

Definition. A BST has the **entropy property** if it reaches this bound.

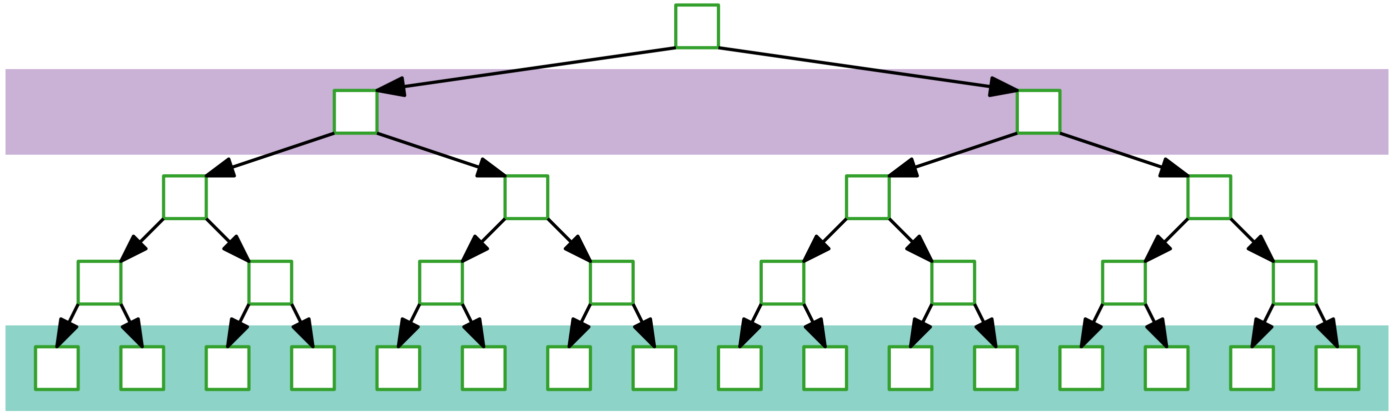
$$p_i = 1/n \Rightarrow H = \sum_{i=1}^{n} 1/n \cdot \log(n) =$$

Model 2: Known Probability Distribution

Access Probabilities:

2	3	5	6	8	9	11
2%	20%	30%	8%	20%	15%	5%

Idea: Place nodes with higher probability higher in the tree.



prob. $\leq 1/2$

OPT: prob. $p \Rightarrow$ level $\log(1/p)$

prob. $\leq 1/2^{\ell-1}$

Lemma. The expected query cost in any BST is at least $\Omega(1 + H)$ per query with $H = \sum_{i=1}^{n} p_i \log(1/p_i)$.

Definition. A BST has the **entropy property** if it reaches this bound.

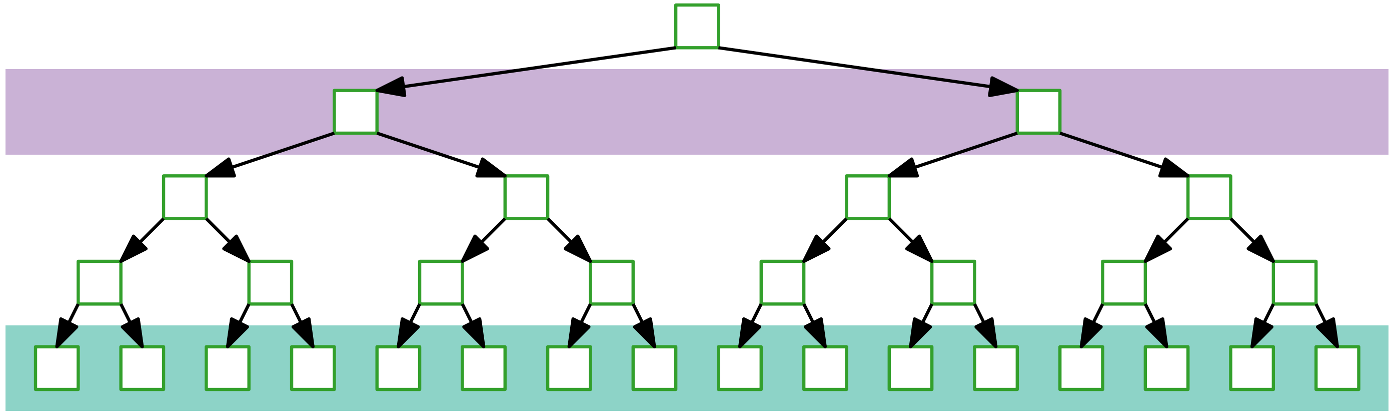
$$p_i = 1/n \Rightarrow H = \sum_{i=1}^{n} 1/n \cdot \log(n) = \log n$$

Model 2: Known Probability Distribution

Access Probabilities:

2	3	5	6	8	9	11
2%	20%	30%	8%	20%	15%	5%

Idea: Place nodes with higher probability higher in the tree.



prob. $\leq 1/2$

OPT: prob. $p \Rightarrow$ level $\log(1/p)$

prob. $\leq 1/2^{\ell-1}$

Lemma. The expected query cost in any BST is at least $\Omega(1+H)$ per query with $H = \sum_{i=1}^{n} p_i \log(1/p_i)$.

Definition. A BST has the **entropy property** if it reaches this bound.

$$p_i = 1/n \Rightarrow H = \sum_{i=1}^{n} 1/n \cdot \log(n) = \log n$$

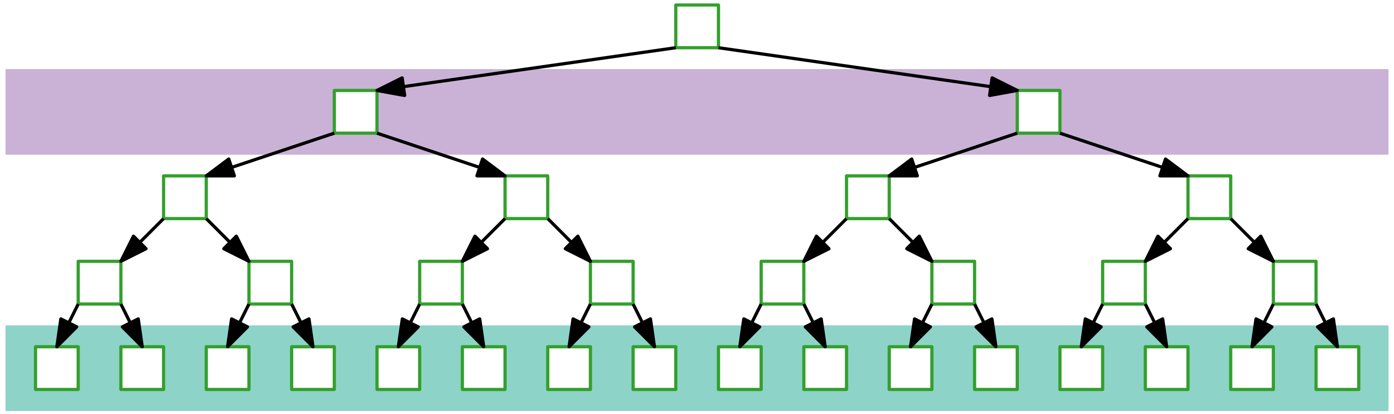
$$p_1 = 1, p_i = 0$$

Model 2: Known Probability Distribution

Access Probabilities:

2	3	5	6	8	9	11
2%	20%	30%	8%	20%	15%	5%

Idea: Place nodes with higher probability higher in the tree.



prob. $\leq 1/2$

OPT: prob. $p \Rightarrow$ level $\log(1/p)$

prob. $\leq 1/2^{\ell-1}$

Lemma. The expected query cost in any BST is at least $\Omega(1+H)$ per query with $H = \sum_{i=1}^{n} -p_i \log p_i$.

Definition. A BST has the **entropy property** if it reaches this bound.

$$p_i = 1/n \Rightarrow H = \sum_{i=1}^{n} 1/n \cdot \log(n) = \log n$$

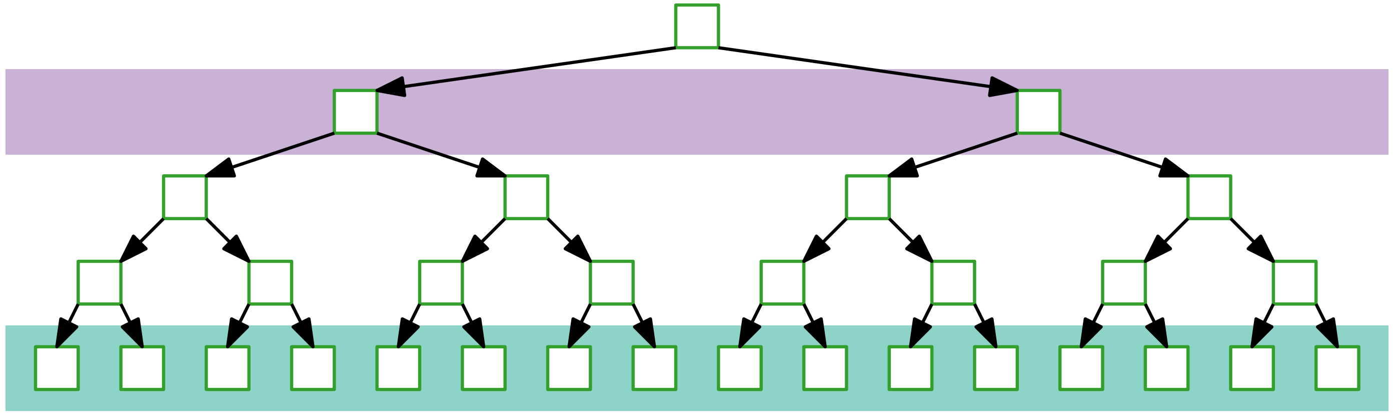
$$p_1 = 1, p_i = 0$$

Model 2: Known Probability Distribution

Access Probabilities:

2	3	5	6	8	9	11
2%	20%	30%	8%	20%	15%	5%

Idea: Place nodes with higher probability higher in the tree.



prob. $\leq 1/2$

OPT: prob. $p \Rightarrow$ level $\log(1/p)$

prob. $\leq 1/2^{\ell-1}$

Lemma. The expected query cost in any BST is at least $\Omega(1+H)$ per query with $H = \sum_{i=1}^{n} -p_i \log p_i$.

Definition. A BST has the **entropy property** if it reaches this bound.

$$p_i = 1/n \Rightarrow H = \sum_{i=1}^{n} 1/n \cdot \log(n) = \log n$$

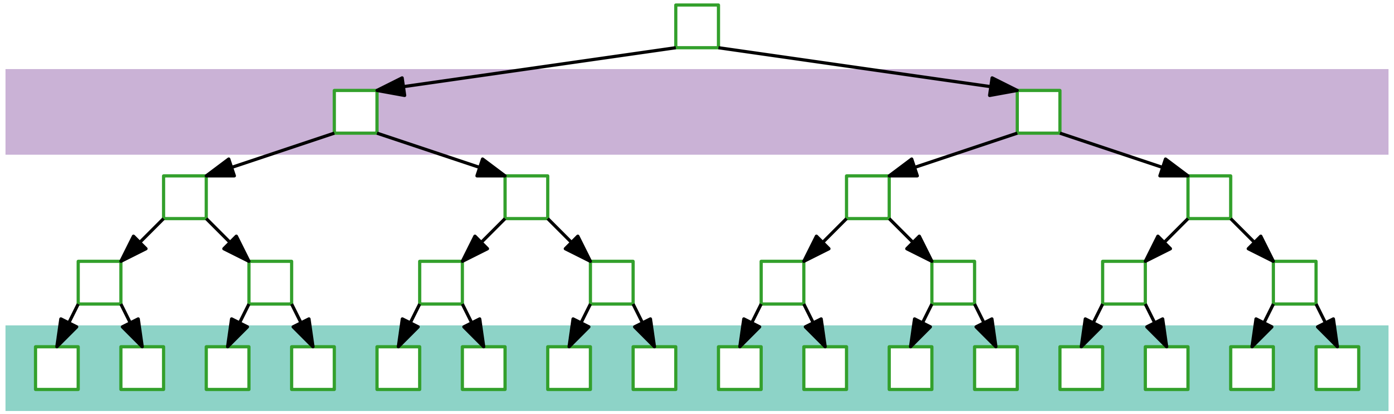
$$p_1 = 1, p_i = 0 \Rightarrow H = -\log 1$$

Model 2: Known Probability Distribution

Access Probabilities:

2	3	5	6	8	9	11
2%	20%	30%	8%	20%	15%	5%

Idea: Place nodes with higher probability higher in the tree.



prob. $\leq 1/2$

OPT: prob. $p \Rightarrow$ level $\log(1/p)$

prob. $\leq 1/2^{\ell-1}$

Lemma. The expected query cost in any BST is at least $\Omega(1+H)$ per query with $H = \sum_{i=1}^{n} -p_i \log p_i$.

Definition. A BST has the **entropy property** if it reaches this bound.

$$p_i = 1/n \Rightarrow H = \sum_{i=1}^{n} 1/n \cdot \log(n) = \log n$$

$$p_1 = 1, p_i = 0 \Rightarrow H = -\log 1 = 0$$

Model 3: Spacial Locality

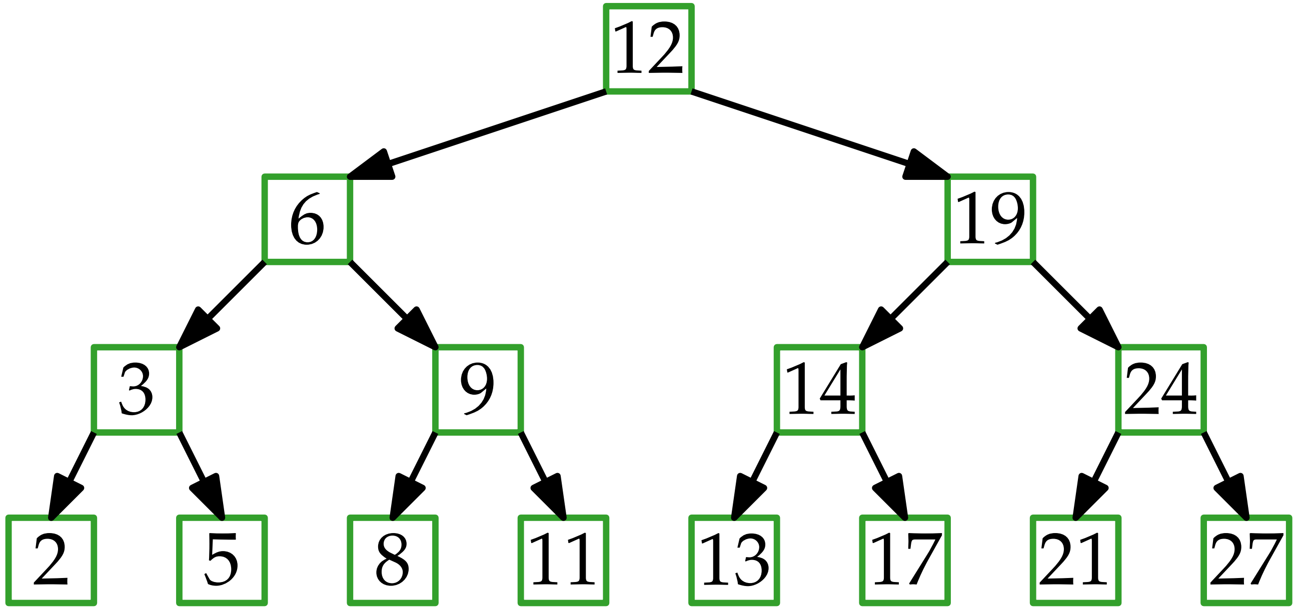
If a key is queried, then keys with nearby values are more likely to be queried.

Model 3: Spacial Locality

If a key is queried, then keys with nearby values are more likely to be queried.
Suppose we queried key x_i and want to query key x_j next.

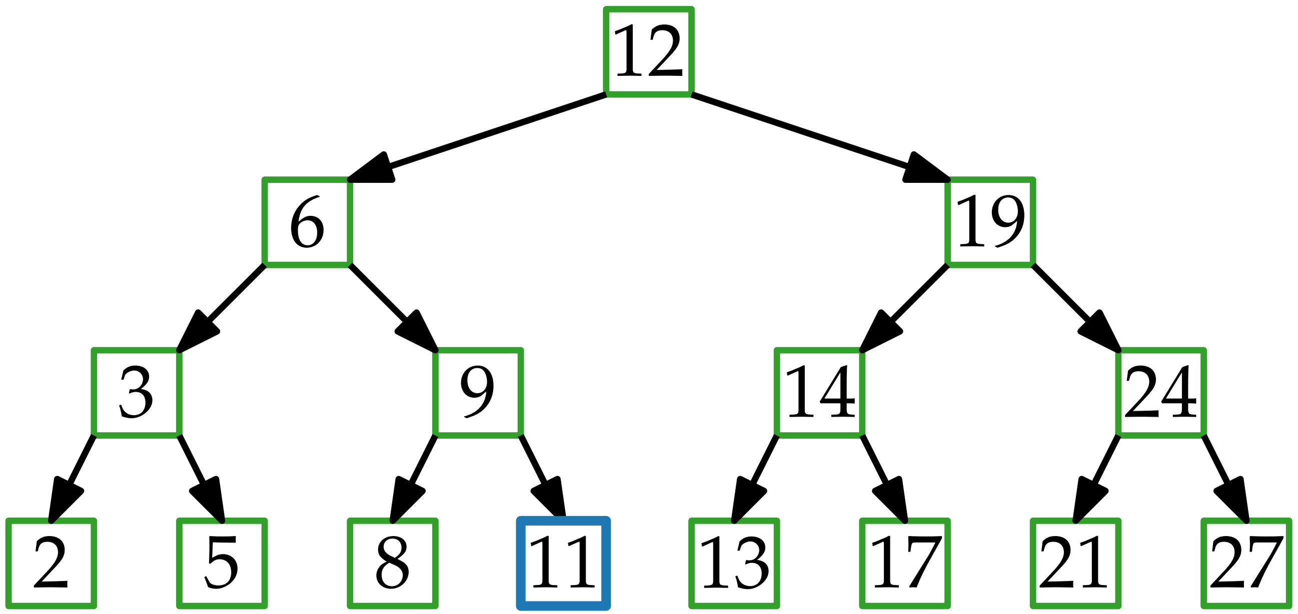
Model 3: Spacial Locality

If a key is queried, then keys with nearby values are more likely to be queried.
Suppose we queried key x_i and want to query key x_j next.



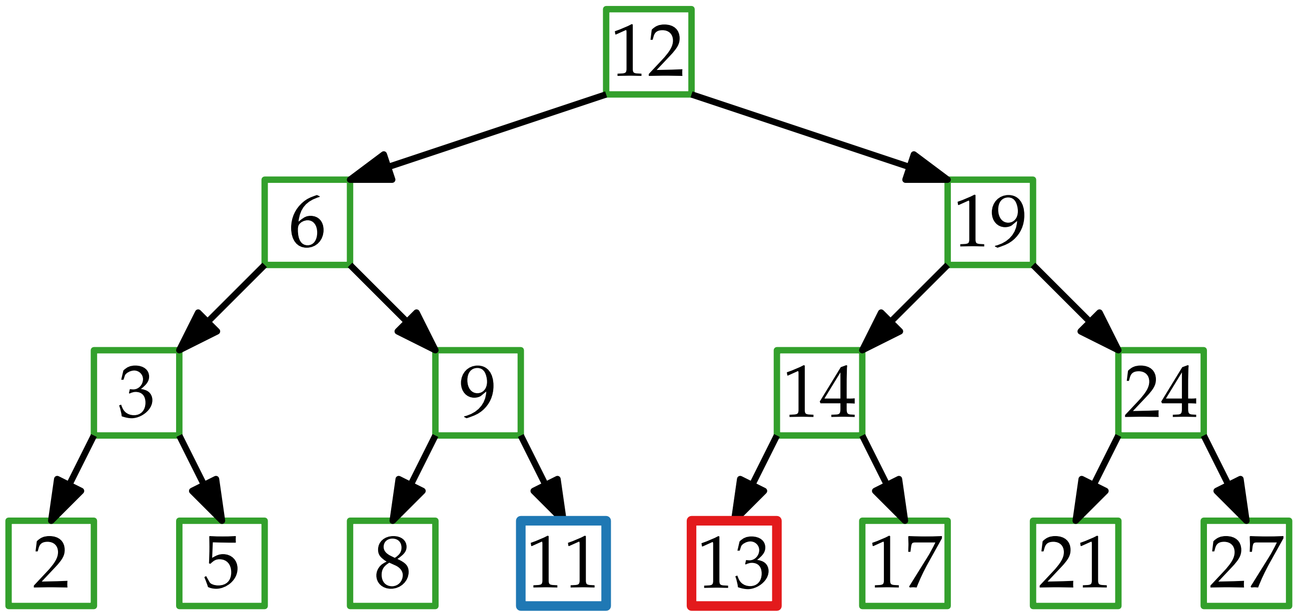
Model 3: Spacial Locality

If a key is queried, then keys with nearby values are more likely to be queried.
Suppose we queried key x_i and want to query key x_j next.



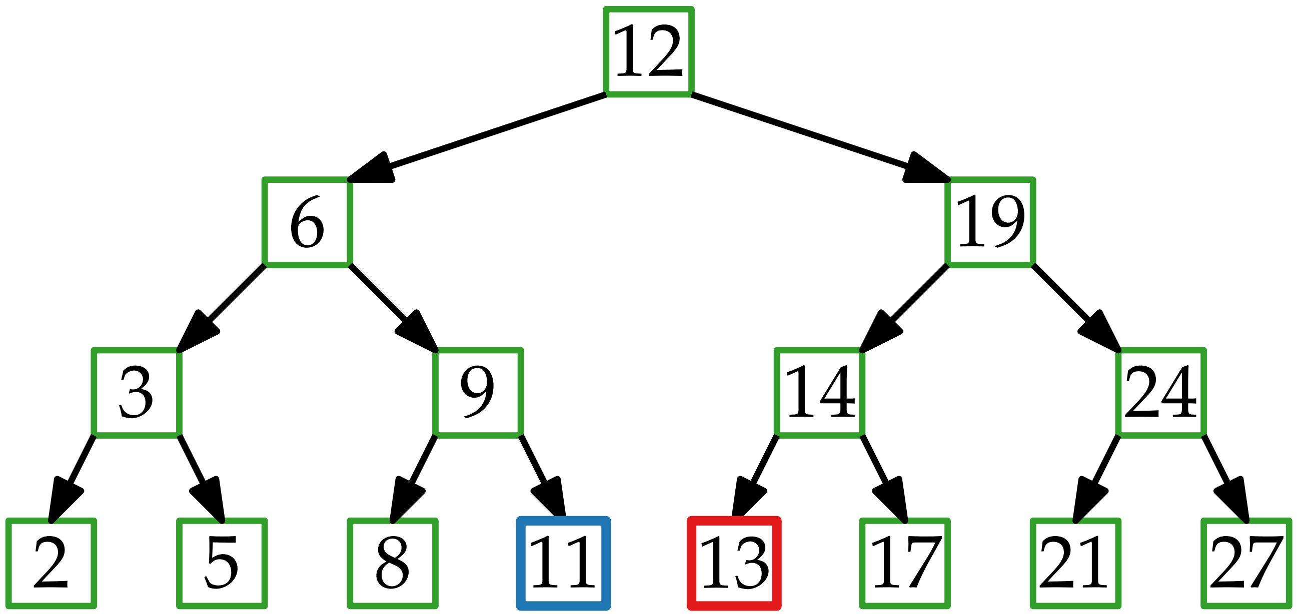
Model 3: Spacial Locality

If a key is queried, then keys with nearby values are more likely to be queried.
Suppose we queried key x_i and want to query key x_j next.



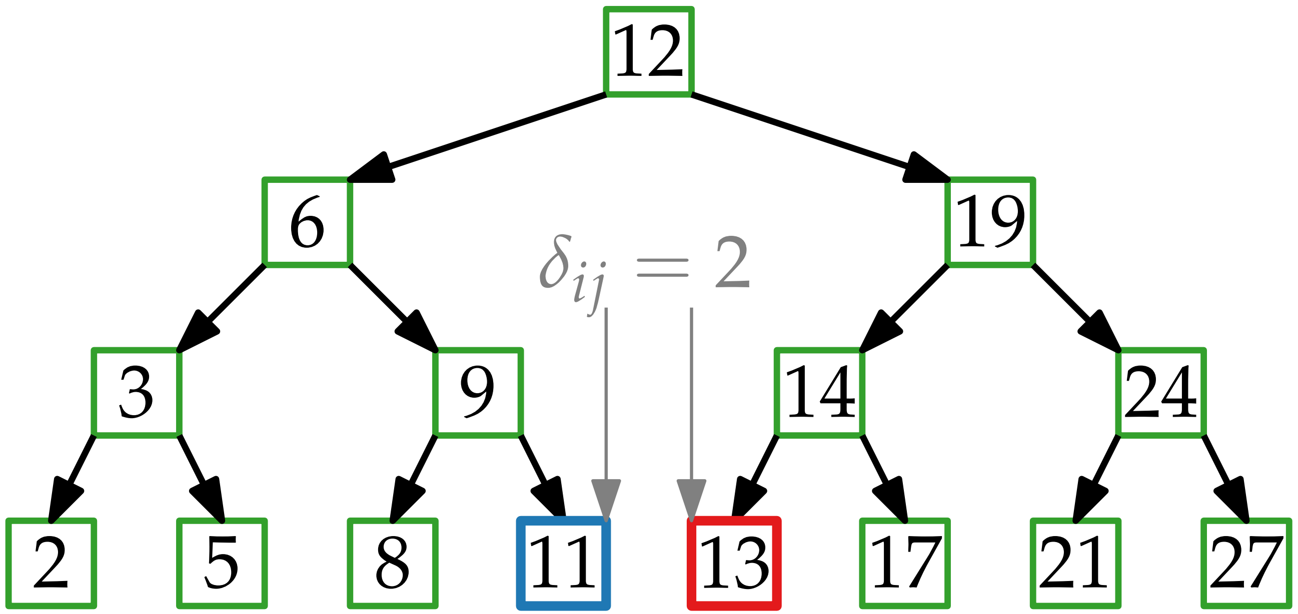
Model 3: Spacial Locality

If a key is queried, then keys with nearby values are more likely to be queried.
Suppose we queried key x_i and want to query key x_j next.
Let $\delta_{ij} = |\operatorname{rank}(x_j) - \operatorname{rank}(x_i)|$.



Model 3: Spacial Locality

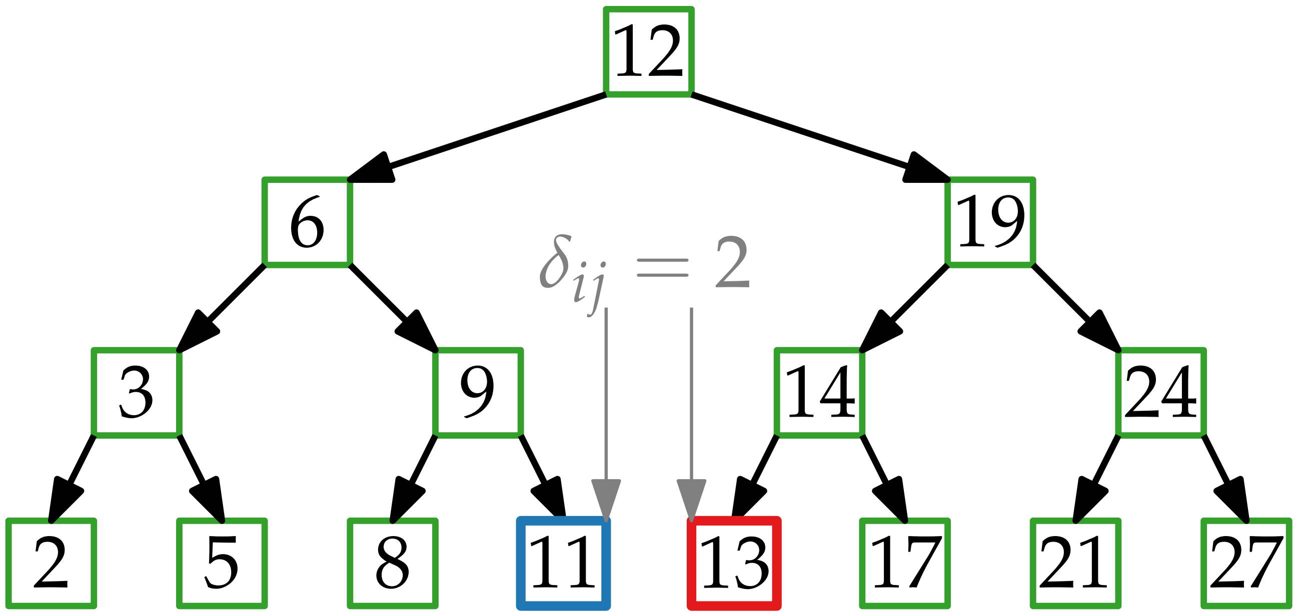
If a key is queried, then keys with nearby values are more likely to be queried.
Suppose we queried key x_i and want to query key x_j next.
Let $\delta_{ij} = |\,\mathrm{rank}(x_j) - \mathrm{rank}(x_i)|$.



Model 3: Spacial Locality

If a key is queried, then keys with nearby values are more likely to be queried.
Suppose we queried key x_i and want to query key x_j next.
Let $\delta_{ij} = |\,\mathrm{rank}(x_j) - \mathrm{rank}(x_i)|$.

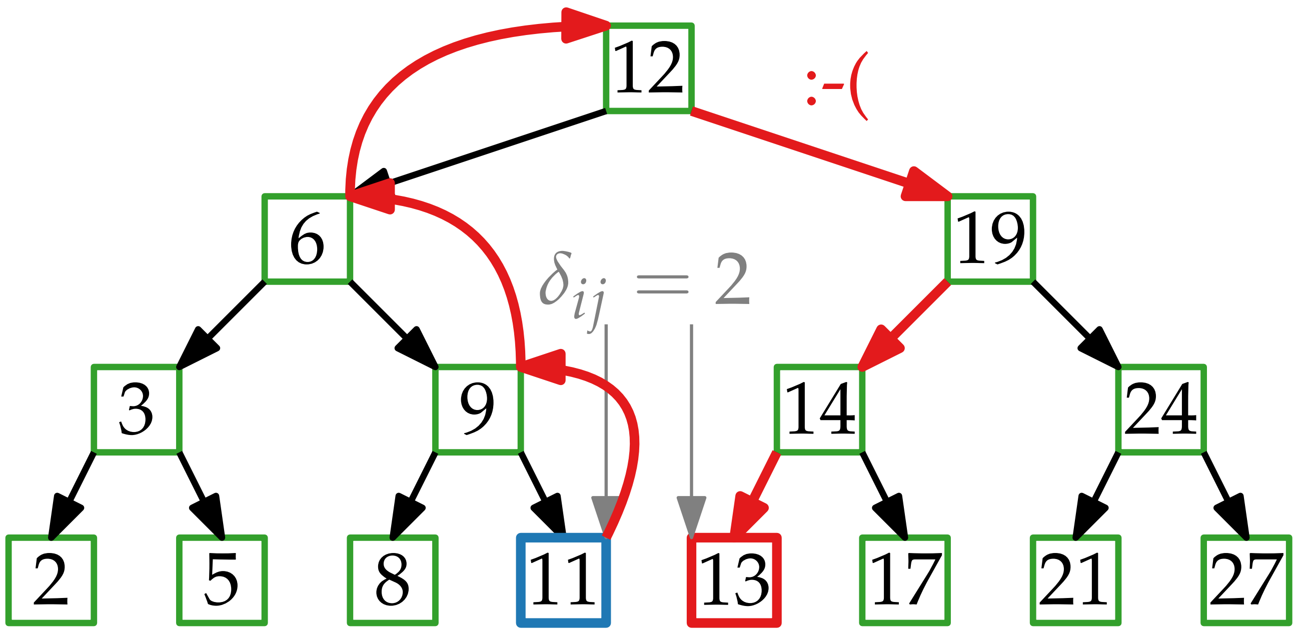
Definition. A BST has the **dynamic finger property** if the (amortized) cost of queries are $O(\log \delta_{ij})$.



Model 3: Spacial Locality

If a key is queried, then keys with nearby values are more likely to be queried.
Suppose we queried key x_i and want to query key x_j next.
Let $\delta_{ij} = |\operatorname{rank}(x_j) - \operatorname{rank}(x_i)|$.

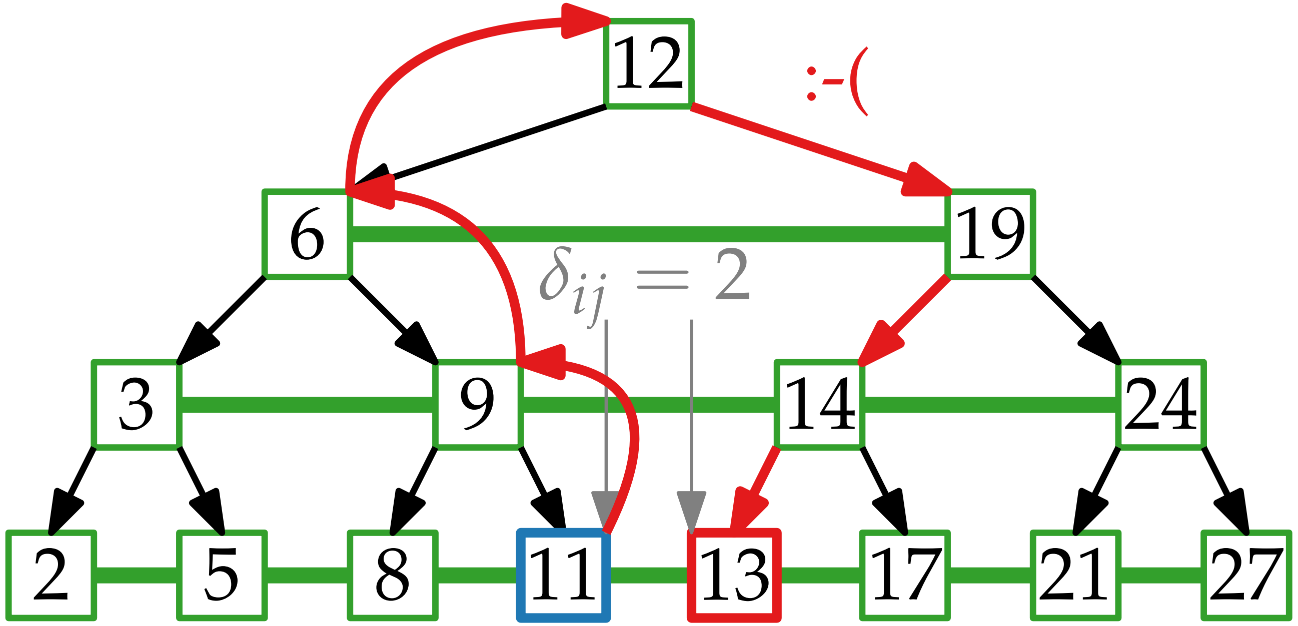
Definition. A BST has the **dynamic finger property** if the (amortized) cost of queries are $O(\log \delta_{ij})$.



Model 3: Spacial Locality

If a key is queried, then keys with nearby values are more likely to be queried.
Suppose we queried key x_i and want to query key x_j next.
Let $\delta_{ij} = |\,\mathrm{rank}(x_j) - \mathrm{rank}(x_i)|$.

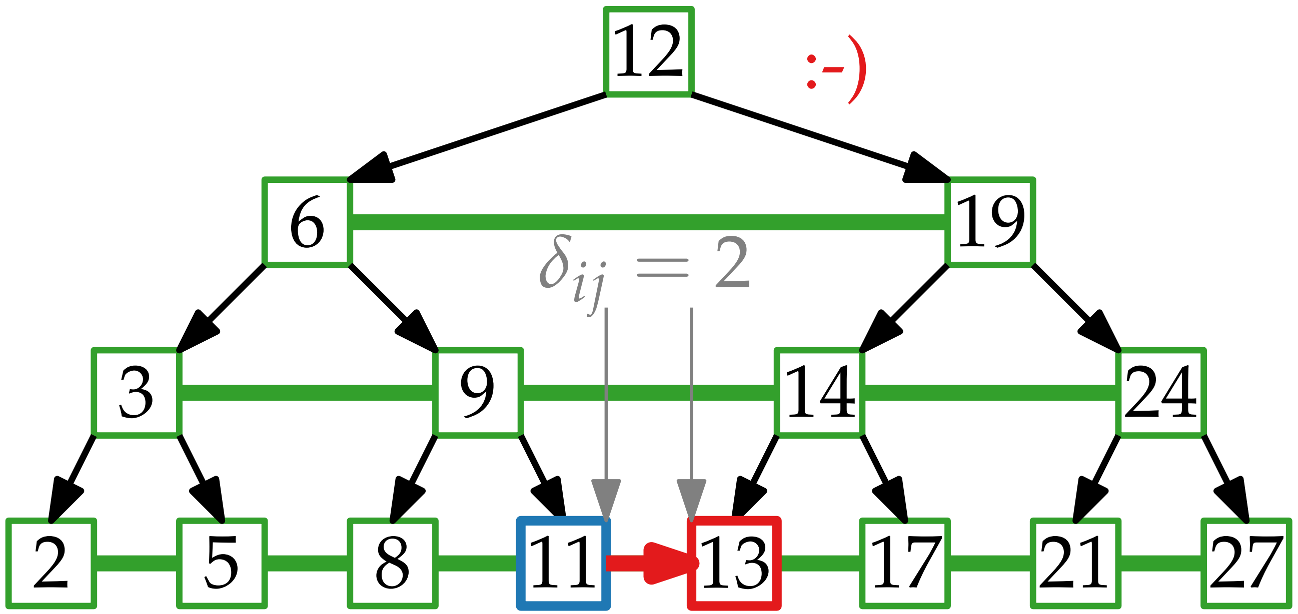
Definition. A BST has the **dynamic finger property** if the (amortized) cost of queries are $O(\log \delta_{ij})$.



Model 3: Spacial Locality

If a key is queried, then keys with nearby values are more likely to be queried.
Suppose we queried key x_i and want to query key x_j next.
Let $\delta_{ij} = |\,\mathrm{rank}(x_j) - \mathrm{rank}(x_i)|$.

Definition. A BST has the **dynamic finger property** if the (amortized) cost of queries are $O(\log \delta_{ij})$.

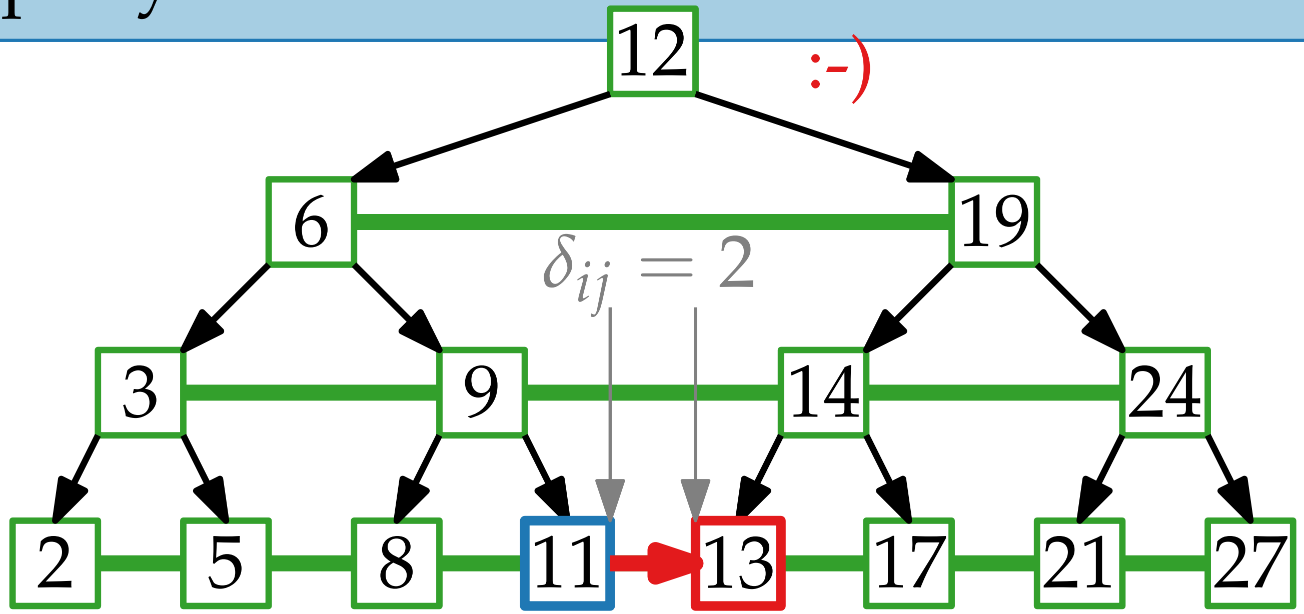


Model 3: Spacial Locality

If a key is queried, then keys with nearby values are more likely to be queried.
Suppose we queried key x_i and want to query key x_j next.
Let $\delta_{ij} = |\operatorname{rank}(x_j) - \operatorname{rank}(x_i)|$.

Definition. A BST has the **dynamic finger property** if the (amortized) cost of queries are $O(\log \delta_{ij})$.

Lemma. A level-linked Red-Black-Tree has the dynamic finger property.



Model 4: Temporal Locality

If a key is queried, then it's likely to be queried again soon.

Model 4: Temporal Locality

If a key is queried, then it's likely to be queried again soon.
A static tree will have a hard time...

Model 4: Temporal Locality

If a key is queried, then it’s likely to be queried again soon.

A static tree will have a hard time...
What if we can move elements?

Model 4: Temporal Locality

If a key is queried, then it's likely to be queried again soon.

A static tree will have a hard time...
What if we can move elements?

Idea: Use a sequence of trees

Model 4: Temporal Locality

If a key is queried, then it’s likely to be queried again soon.

A static tree will have a hard time...
What if we can move elements?

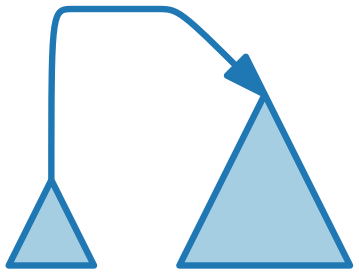
Idea: Use a sequence of trees

Model 4: Temporal Locality

If a key is queried, then it’s likely to be queried again soon.

A static tree will have a hard time...
What if we can move elements?

Idea: Use a sequence of trees

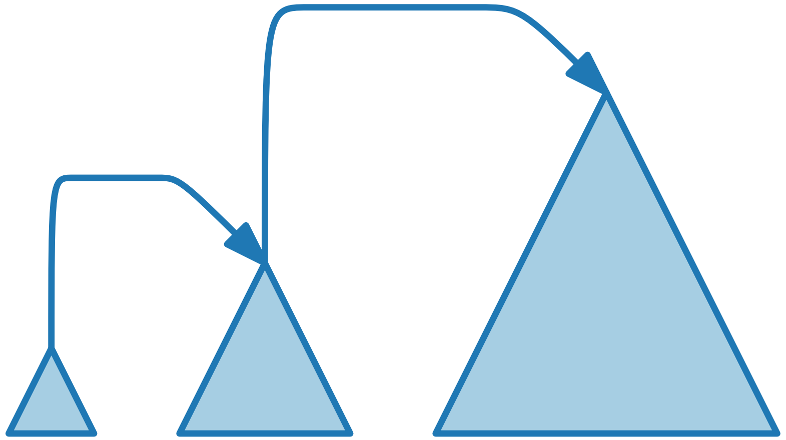


Model 4: Temporal Locality

If a key is queried, then it's likely to be queried again soon.

A static tree will have a hard time...
What if we can move elements?

Idea: Use a sequence of trees

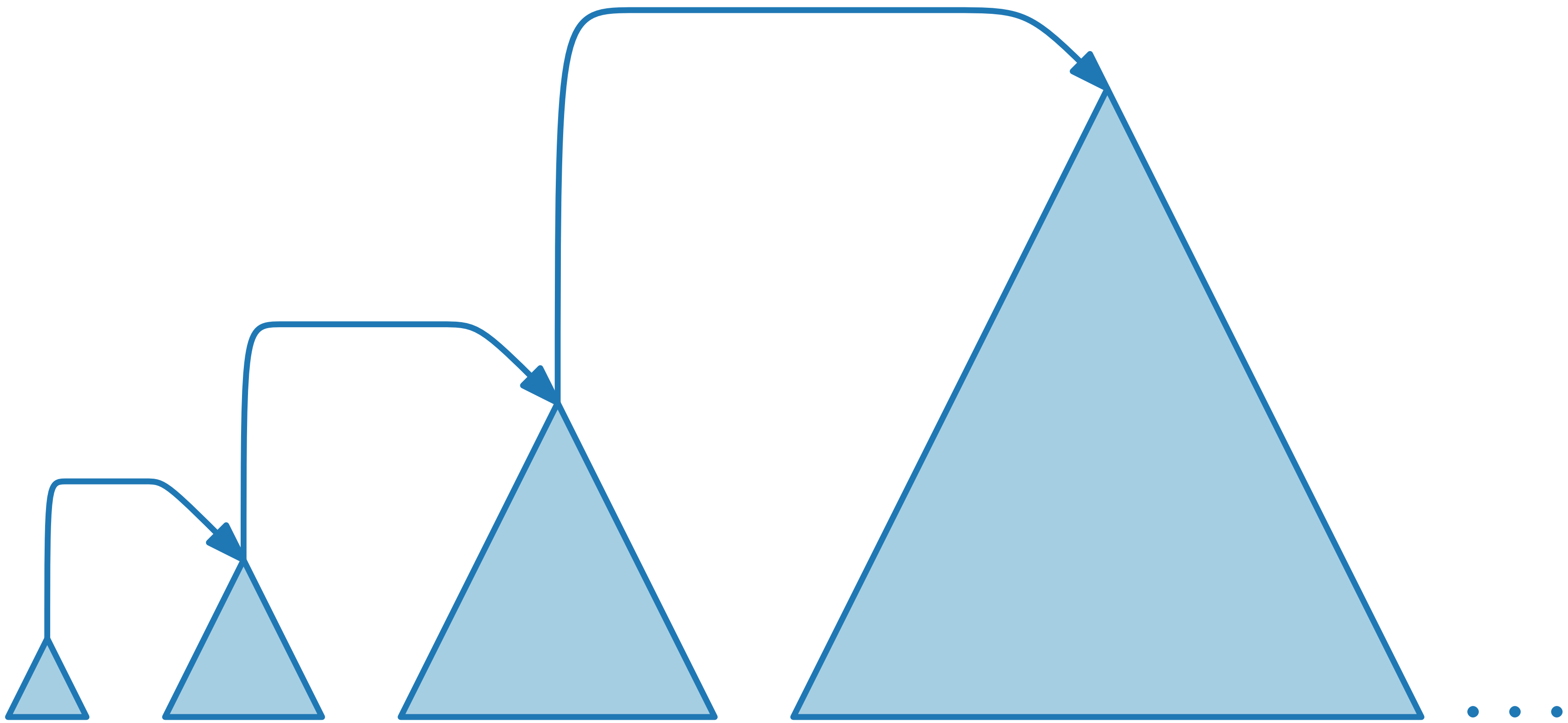


Model 4: Temporal Locality

If a key is queried, then it's likely to be queried again soon.

A static tree will have a hard time...
What if we can move elements?

Idea: Use a sequence of trees

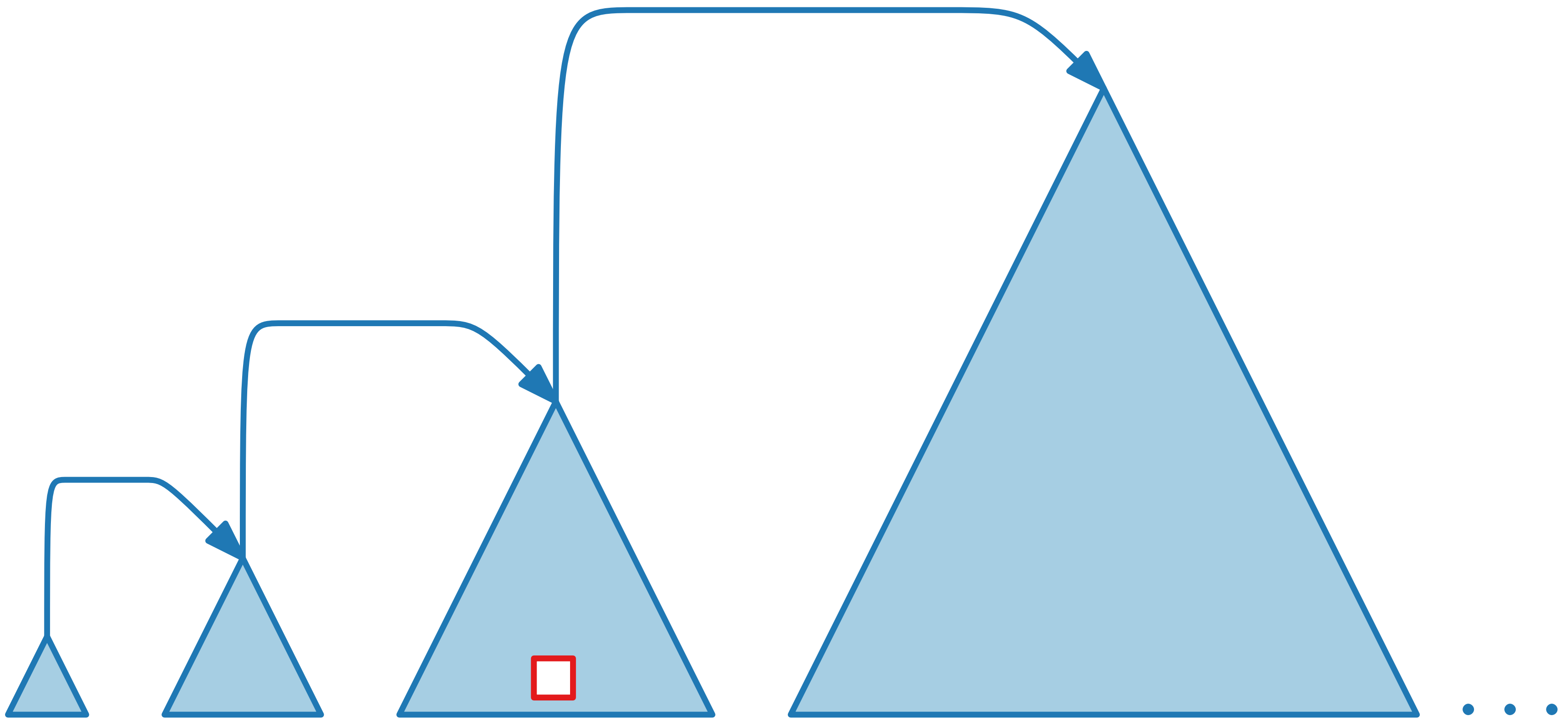


Model 4: Temporal Locality

If a key is queried, then it's likely to be queried again soon.

A static tree will have a hard time...
What if we can move elements?

Idea: Use a sequence of trees

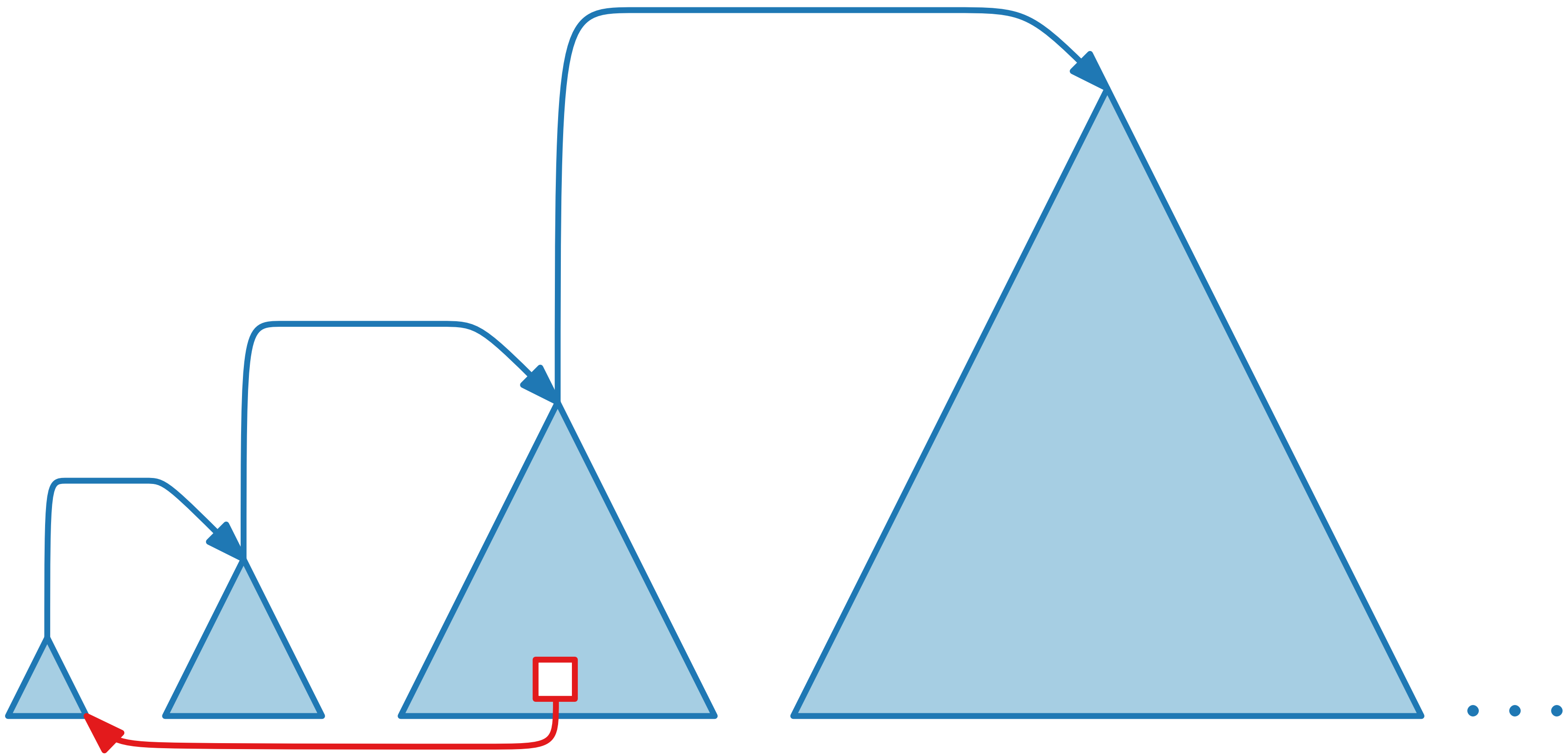


Model 4: Temporal Locality

If a key is queried, then it's likely to be queried again soon.

A static tree will have a hard time...
What if we can move elements?

Idea: Use a sequence of trees
Move queried key to first tree, then kick out oldest key.



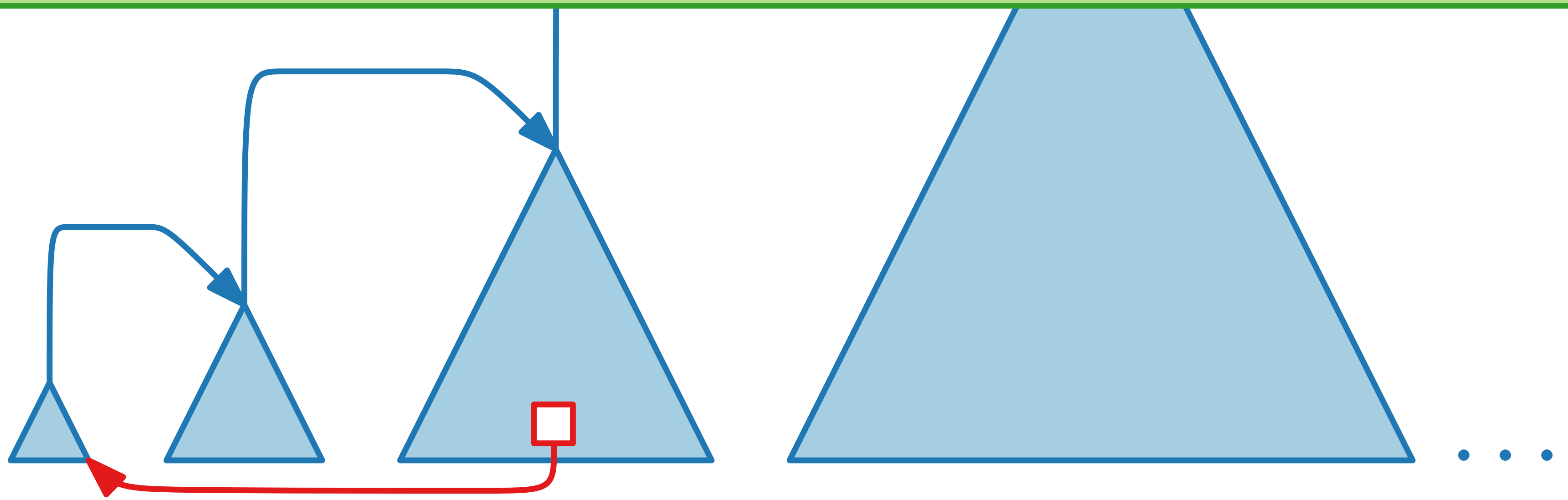
Model 4: Temporal Locality

If a key is queried, then it's likely to be queried again soon.

A static tree will have a hard time...
What if we can move elements?

Idea: Use a sequence of trees
Move queried key to first tree, then kick out oldest key.

Definition. A BST has the **working set property** if the (amortized) cost of a query for key x is $O(\log t)$, where t is the number of keys queried more recently than x.



Model 5: Static Optimality

Given a sequence S of queries.

Model 5: Static Optimality

Given a sequence S of queries.
Let T_S^* be the *optimal* static tree with the shortest query time OPT_S for S.

Model 5: Static Optimality

e.g. $S = 2, 5, 2, 5, 2, \ldots, 5$

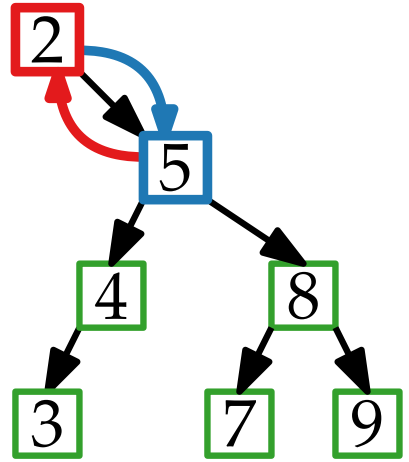
Given a sequence S of queries.
Let T^*_S be the *optimal* static tree with the shortest query time OPT_S for S.

Model 5: Static Optimality

Given a sequence S of queries.
Let T_S^* be the *optimal* static tree with the shortest query time OPT_S for S.

e.g. $S = 2, 5, 2, 5, 2, \ldots, 5$

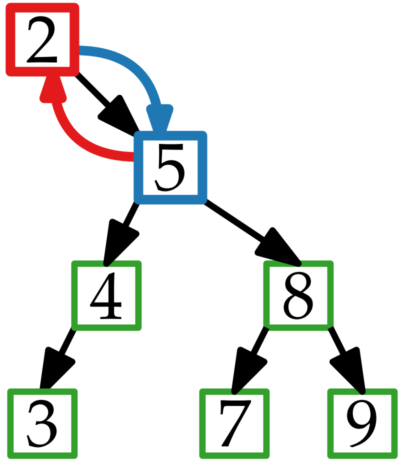
T^*:



Model 5: Static Optimality

Given a sequence S of queries.
Let T_S^* be the *optimal* static tree with the shortest query time OPT_S for S.

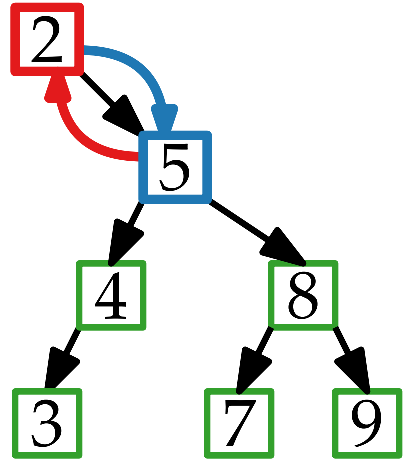
e.g. $S = 2, 5, 2, 5, 2, \ldots, 5$
T^*:
OPT: $|S|$



Model 5: Static Optimality

Given a sequence S of queries.
Let T_S^* be the *optimal* static tree with the shortest query time OPT_S for S.

e.g. $S = 2, 5, 2, 5, 2, \ldots, 5$
T^*:
OPT: $|S|$



Definition. A BST is **statically optimal** if queries take (amort.) $O(\mathrm{OPT}_S)$ time for every S.

All These Properties...

Balanced:	Queries take (amort.) $O(\log n)$ time
Entropy:	Queries take expected $O(1 + H)$ time
Dynamic Finger:	Queries take $O(\log \delta_i)$ time (δ_i: rank diff.)
Working Set:	Queries take $O(\log t)$ time (t: recency)
Static Optimality:	Queries take (amort.) $O(\mathrm{OPT}_S)$ time.

All These Properties...

Balanced: Queries take (amort.) $O(\log n)$ time

Entropy: Queries take expected $O(1 + H)$ time

Dynamic Finger: Queries take $O(\log \delta_i)$ time (δ_i: rank diff.)

Working Set: Queries take $O(\log t)$ time (t: recency)

Static Optimality: Queries take (amort.) $O(\mathrm{OPT}_S)$ time.

... is there one BST to rule them all?

All These Properties...

Balanced: Queries take (amort.) $O(\log n)$ time

Entropy: Queries take expected $O(1 + H)$ time

Dynamic Finger: Queries take $O(\log \delta_i)$ time (δ_i: rank diff.)

Working Set: Queries take $O(\log t)$ time (t: recency)

Static Optimality: Queries take (amort.) $O(\text{OPT}_S)$ time.

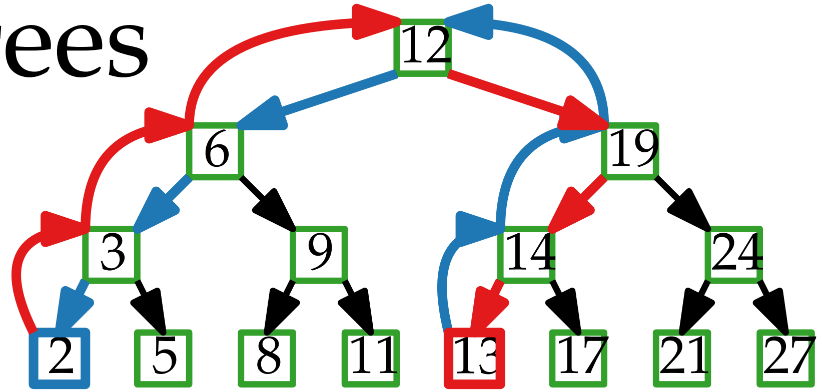
... is there one BST to rule them all?

Yes!

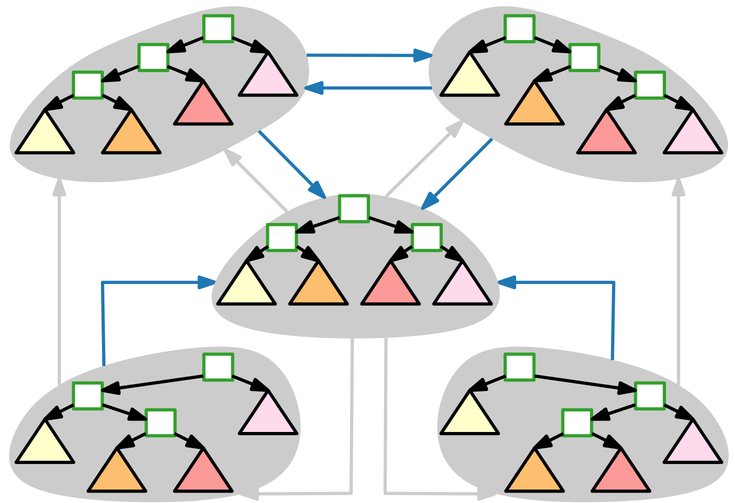
Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Advanced Algorithms

Optimal Binary Search Trees



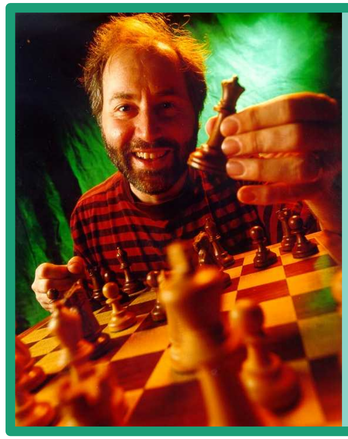
Splay Trees



Philipp Kindermann · WS20

Part III:
Splay Trees

Splay Trees

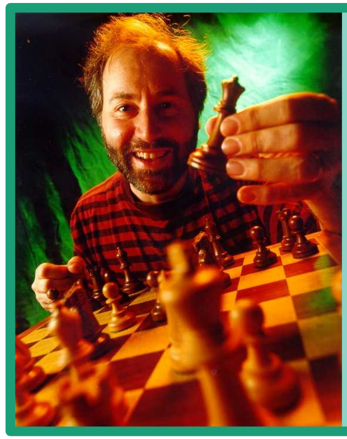


Daniel D. Sleator

Robert E. Tarjan

J. ACM 1985

Splay Trees



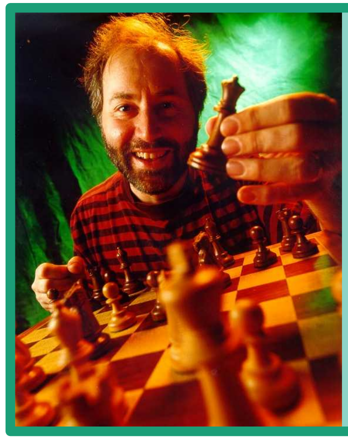
Daniel D. Sleator

Robert E. Tarjan

J. ACM 1985

Idea: Whenever we query a key, rotate it to the root.

Splay Trees



Daniel D. Sleator

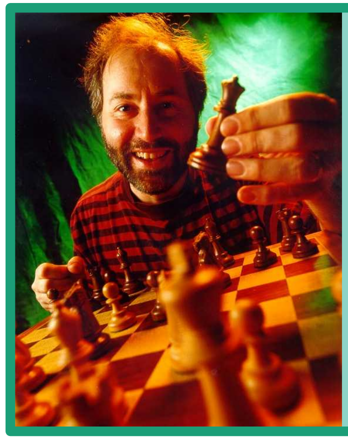
Robert E. Tarjan

J. ACM 1985

Idea: Whenever we query a key, rotate it to the root.

ADS:

Splay Trees

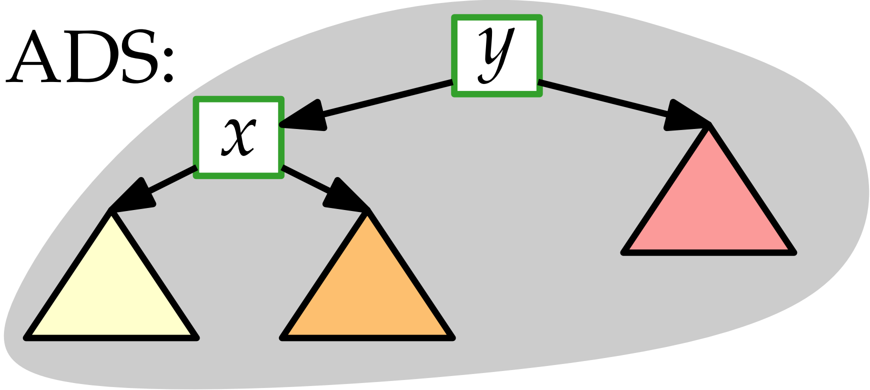


Daniel D. Sleator Robert E. Tarjan

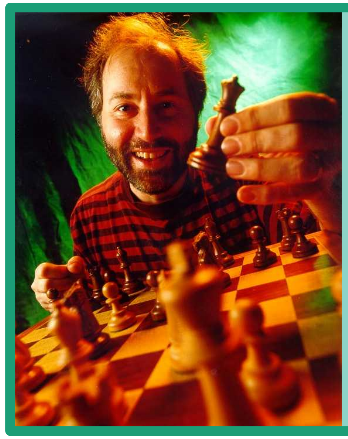
J. ACM 1985

Idea: Whenever we query a key, rotate it to the root.

ADS:



Splay Trees



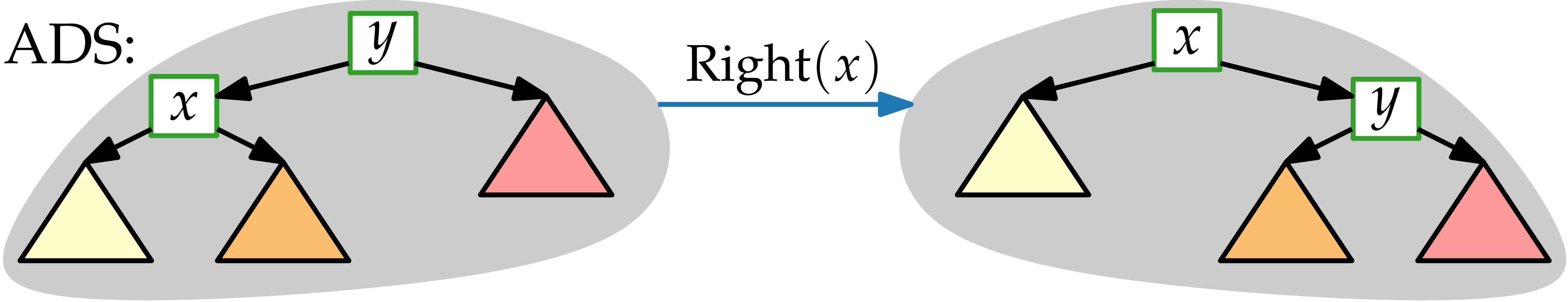
Daniel D. Sleator

Robert E. Tarjan

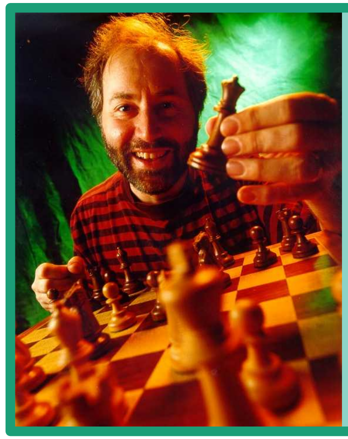
J. ACM 1985

Idea: Whenever we query a key, rotate it to the root.

ADS:



Splay Trees



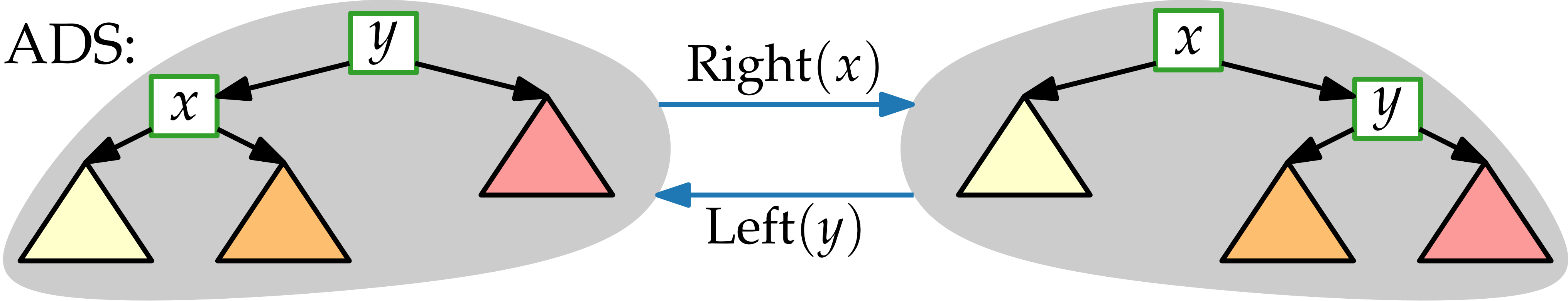
Daniel D. Sleator

Robert E. Tarjan

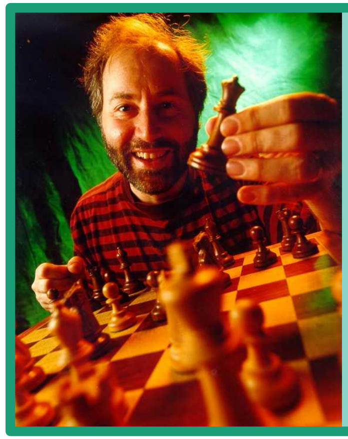
J. ACM 1985

Idea: Whenever we query a key, rotate it to the root.

ADS:



Splay Trees

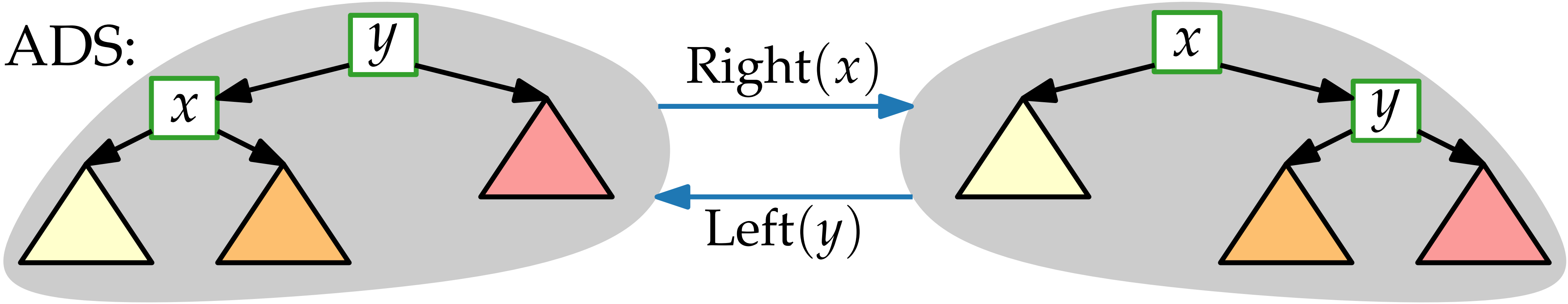


Daniel D. Sleator Robert E. Tarjan

J. ACM 1985

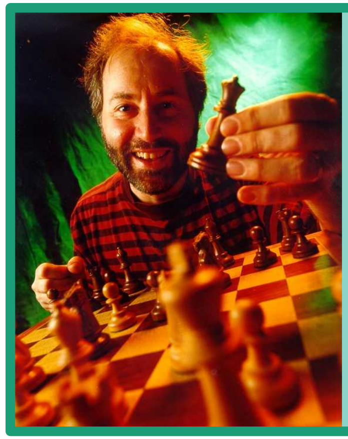
Idea: Whenever we query a key, rotate it to the root.

ADS:



Splay(x): Rotate x to the root

Splay Trees

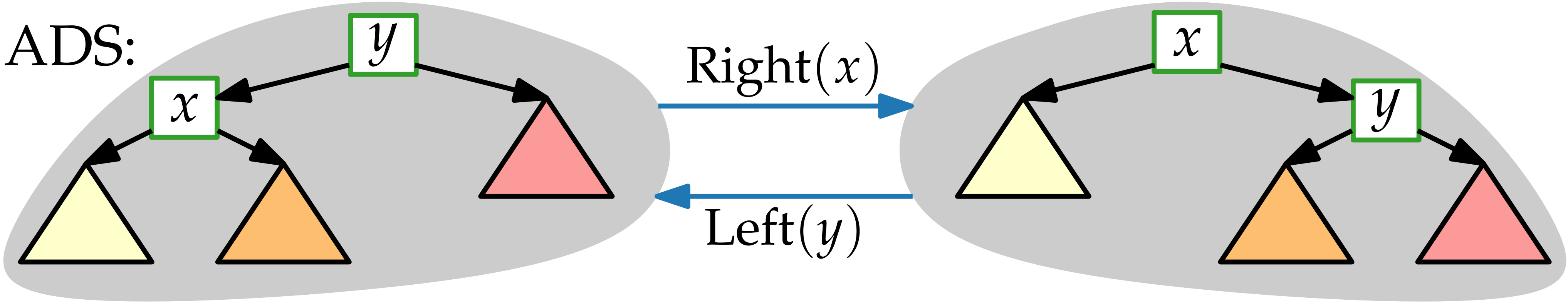


Daniel D. Sleator Robert E. Tarjan

J. ACM 1985

Idea: Whenever we query a key, rotate it to the root.

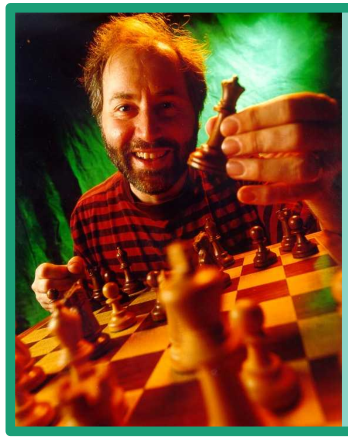
ADS:



Splay(x): Rotate x to the root

Query(x): Splay(x), then return root

Splay Trees

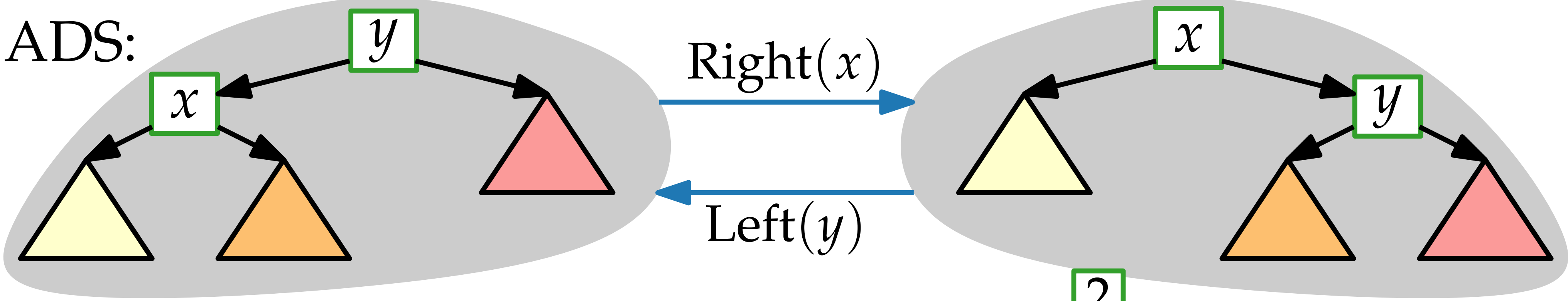


Daniel D. Sleator Robert E. Tarjan

J. ACM 1985

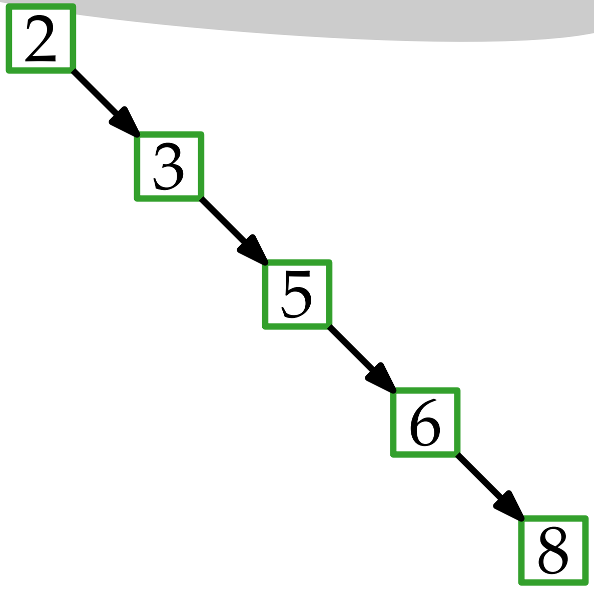
Idea: Whenever we query a key, rotate it to the root.

ADS:

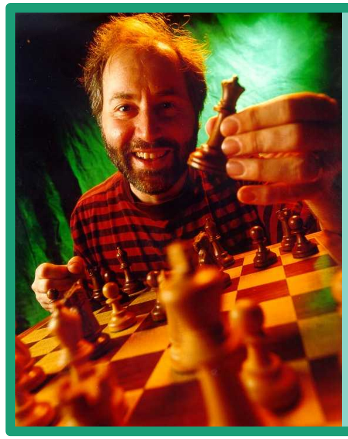


Splay(x): Rotate x to the root

Query(x): Splay(x), then return root



Splay Trees



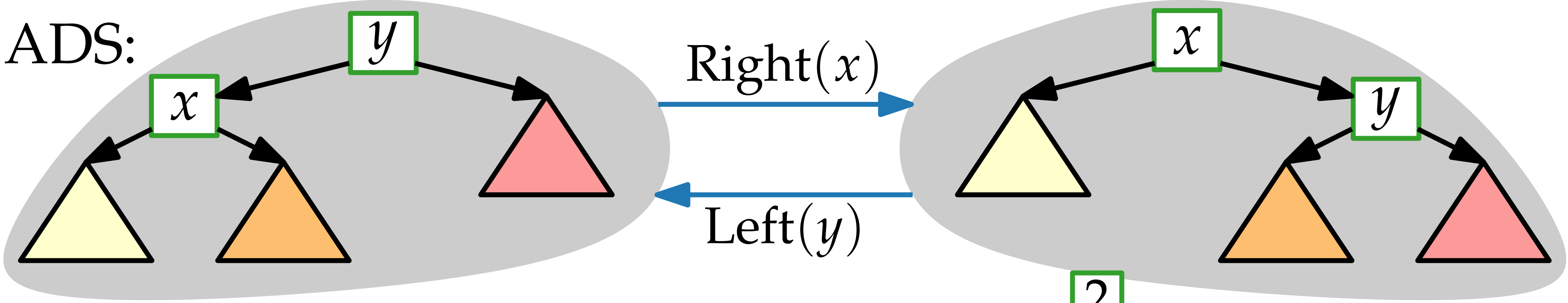
Daniel D. Sleator

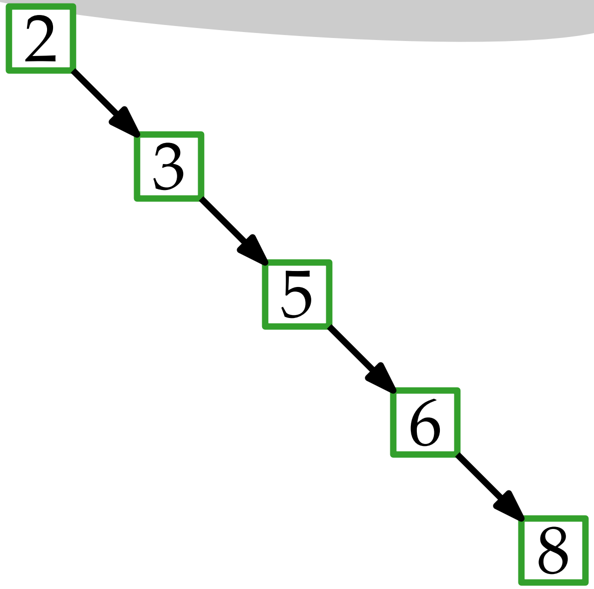
Robert E. Tarjan

J. ACM 1985

Idea: Whenever we query a key, rotate it to the root.

ADS:



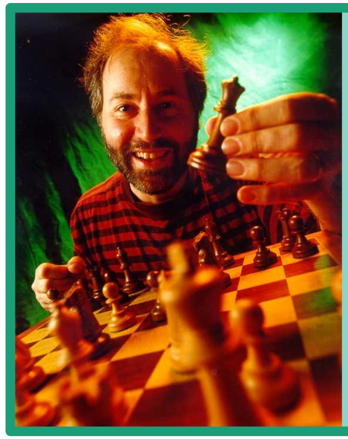


Splay(x): Rotate x to the root

Query(x): Splay(x), then return root

Query(8)

Splay Trees

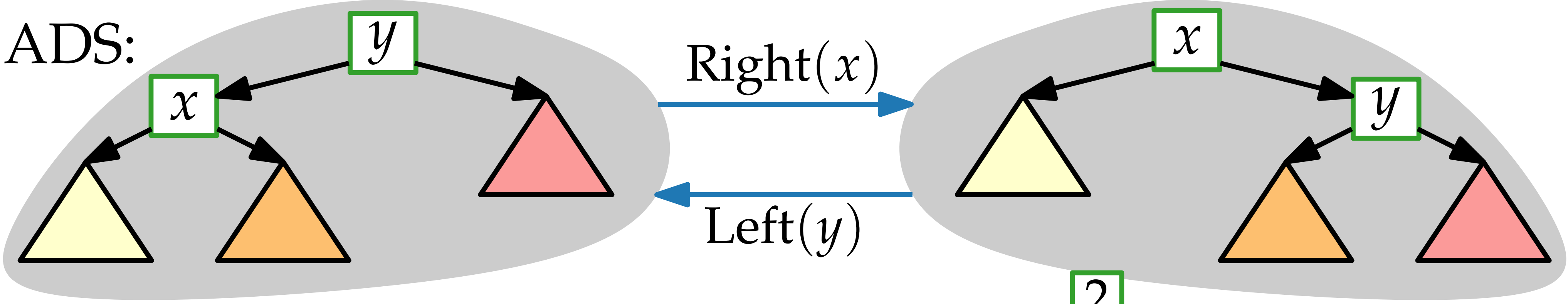


Daniel D. Sleator   Robert E. Tarjan

J. ACM 1985

Idea: Whenever we query a key, rotate it to the root.

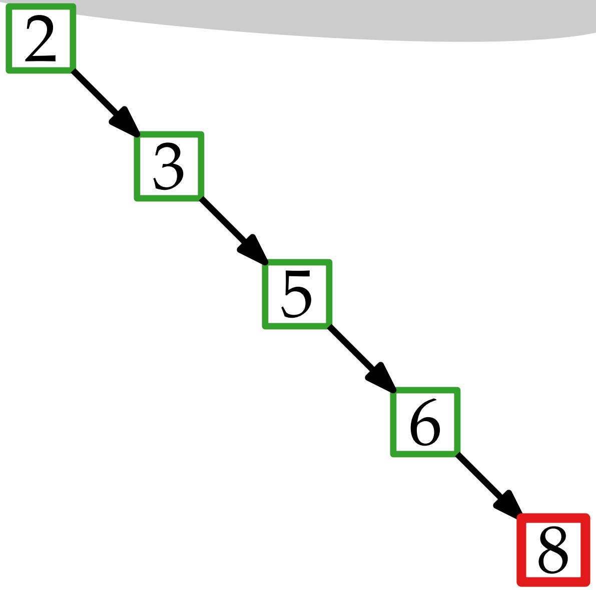
ADS:



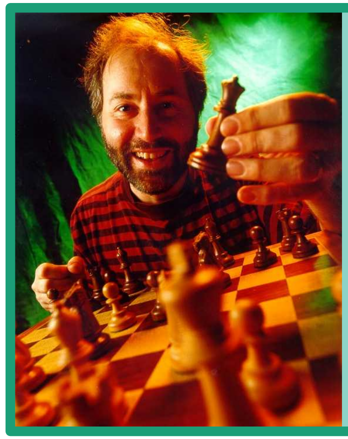
Splay(x): Rotate x to the root

Query(x): Splay(x), then return root

Query(8)



Splay Trees



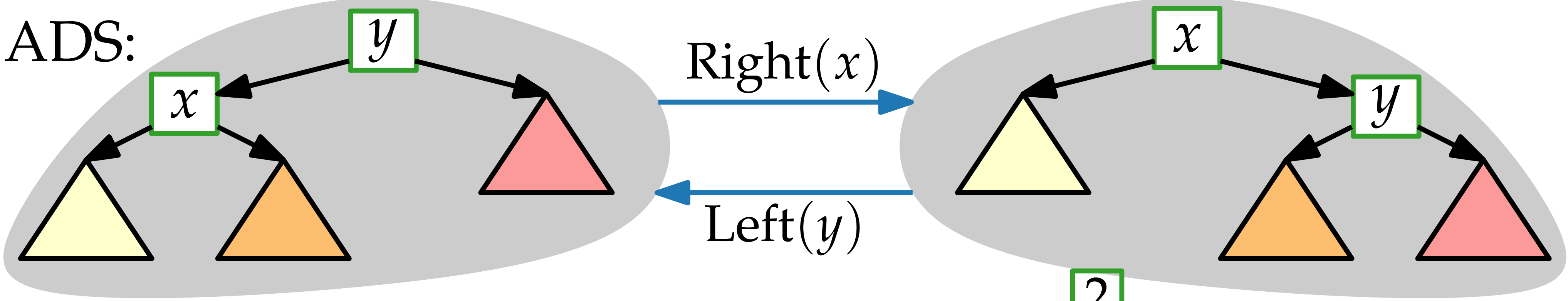
Daniel D. Sleator

Robert E. Tarjan

J. ACM 1985

Idea: Whenever we query a key, rotate it to the root.

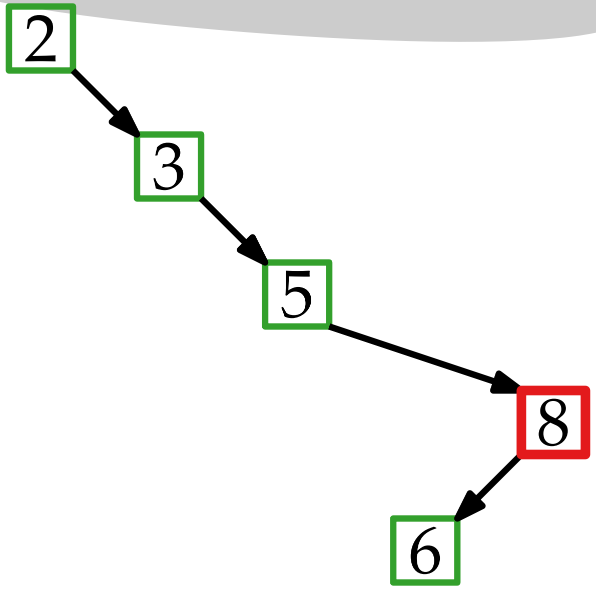
ADS:



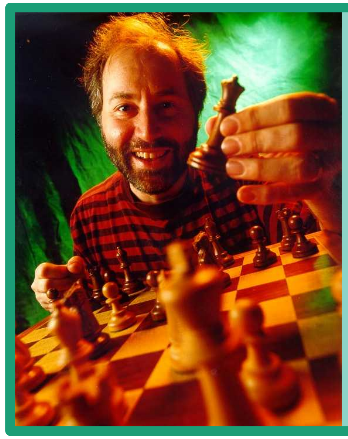
Splay(x): Rotate x to the root

Query(x): Splay(x), then return root

Query(8)



Splay Trees

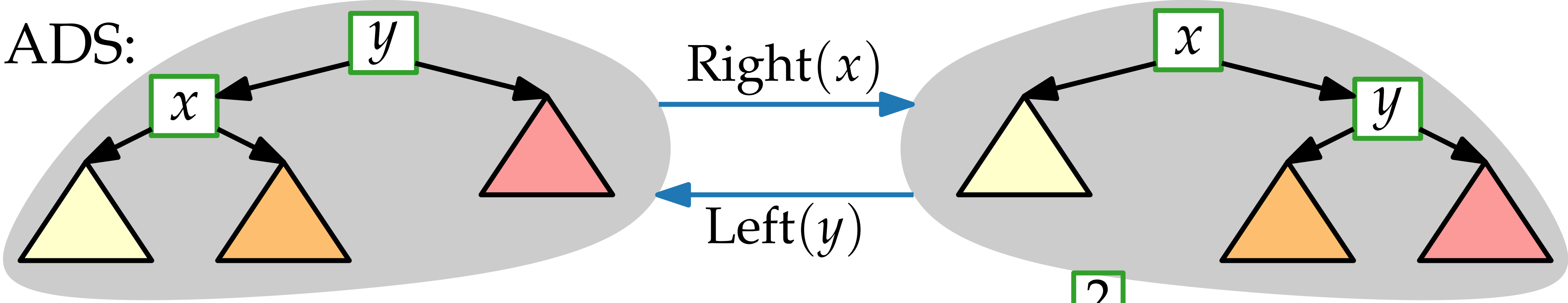


Daniel D. Sleator Robert E. Tarjan

J. ACM 1985

Idea: Whenever we query a key, rotate it to the root.

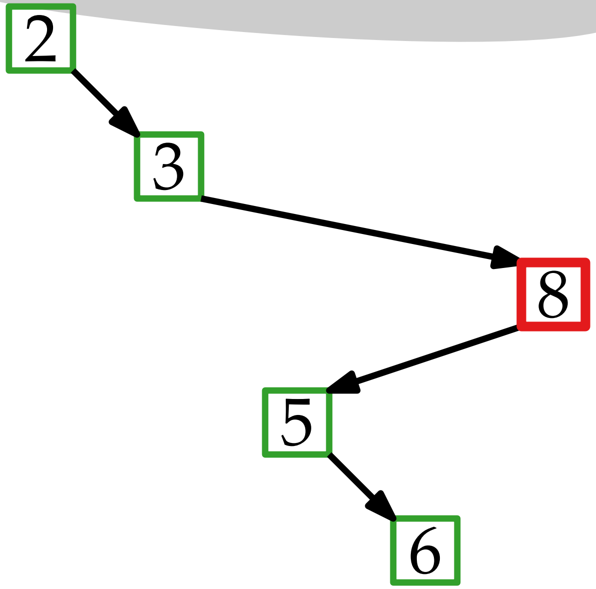
ADS:



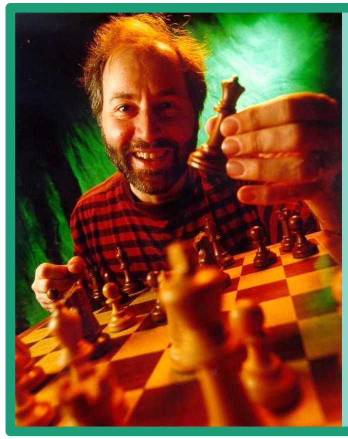
Splay(x): Rotate x to the root

Query(x): Splay(x), then return root

Query(8)



Splay Trees

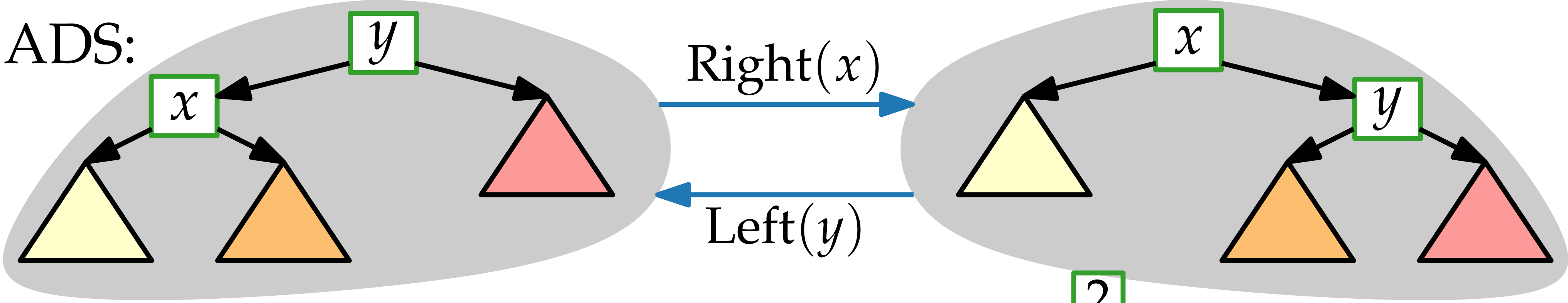


Daniel D. Sleator   Robert E. Tarjan

J. ACM 1985

Idea: Whenever we query a key, rotate it to the root.

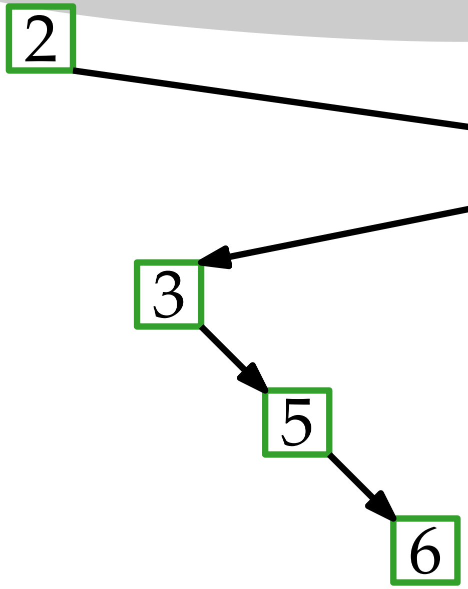
ADS:



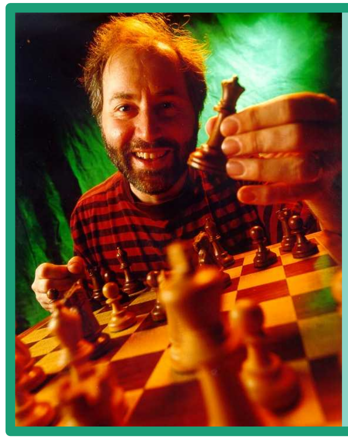
Splay(x): Rotate x to the root

Query(x): Splay(x), then return root

Query(8)



Splay Trees

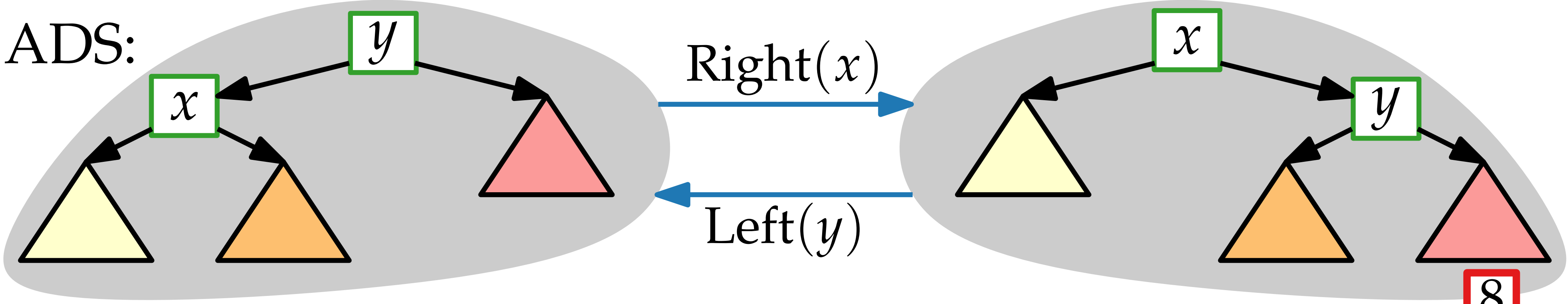


Daniel D. Sleator Robert E. Tarjan

J. ACM 1985

Idea: Whenever we query a key, rotate it to the root.

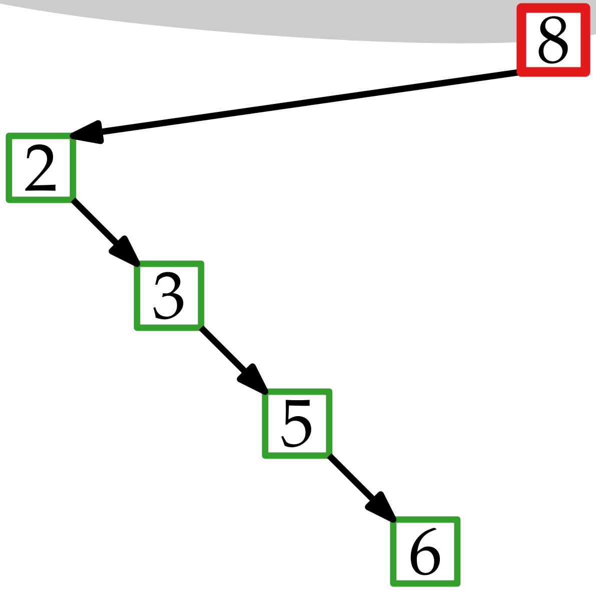
ADS:



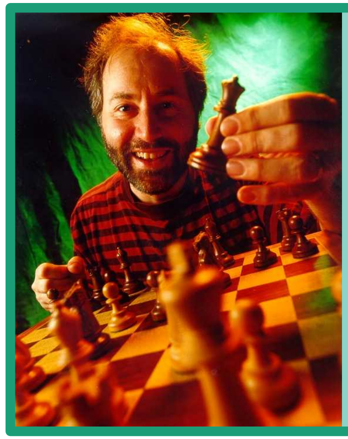
Splay(x): Rotate x to the root

Query(x): Splay(x), then return root

Query(8)



Splay Trees

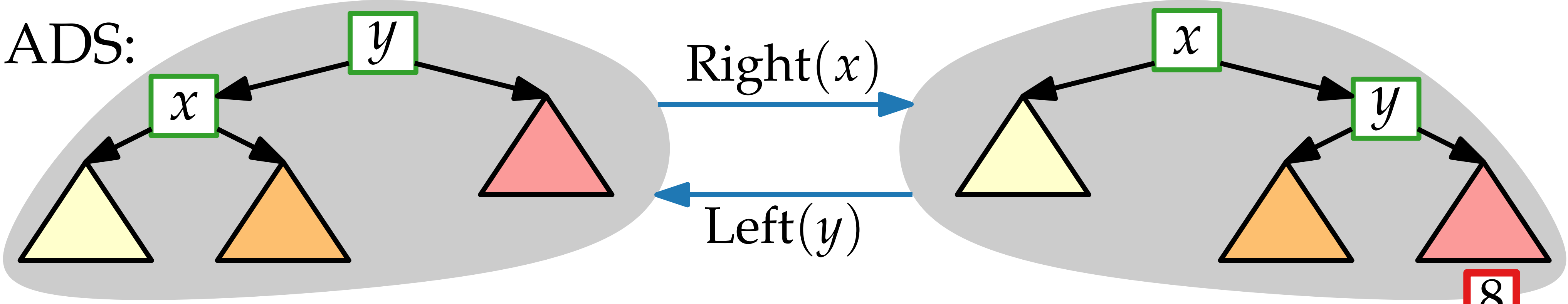


Daniel D. Sleator Robert E. Tarjan

J. ACM 1985

Idea: Whenever we query a key, rotate it to the root.

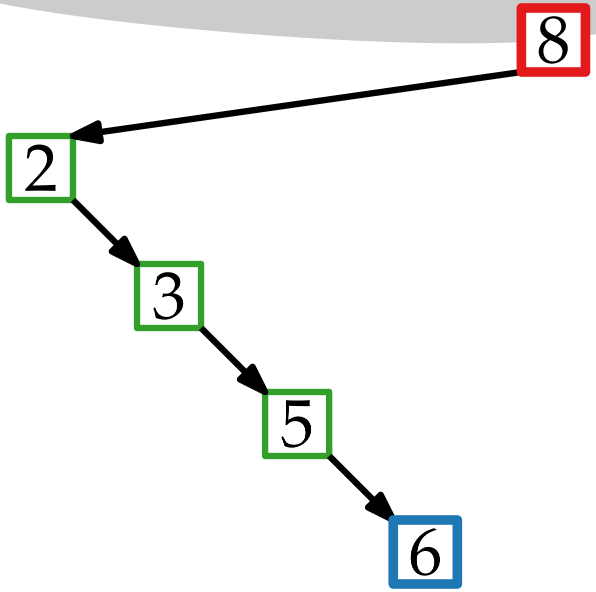
ADS:



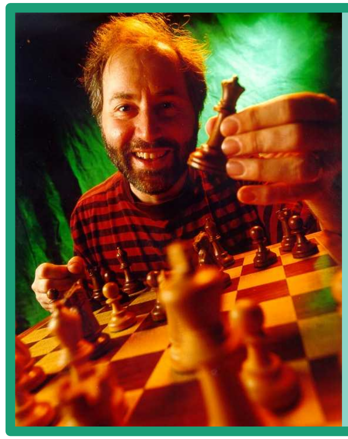
Splay(x): Rotate x to the root

Query(x): Splay(x), then return root

Query(8) Query(6)



Splay Trees

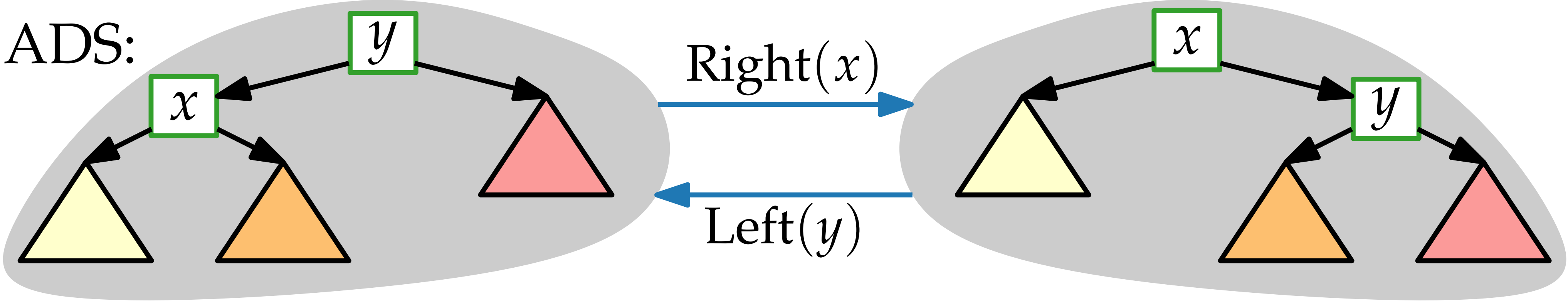


Daniel D. Sleator Robert E. Tarjan

J. ACM 1985

Idea: Whenever we query a key, rotate it to the root.

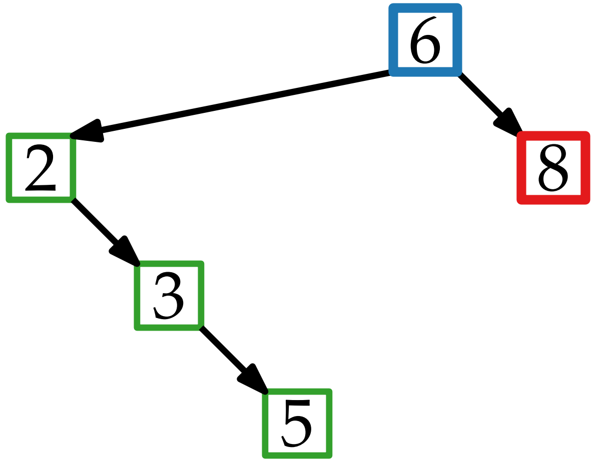
ADS:



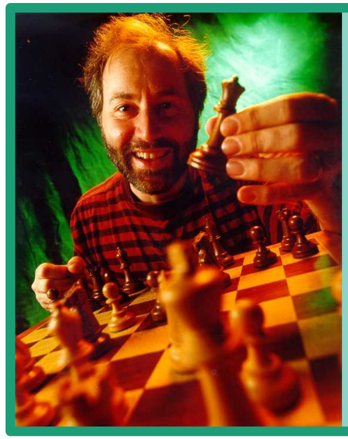
Splay(x): Rotate x to the root

Query(x): Splay(x), then return root

Query(8) Query(6)



Splay Trees

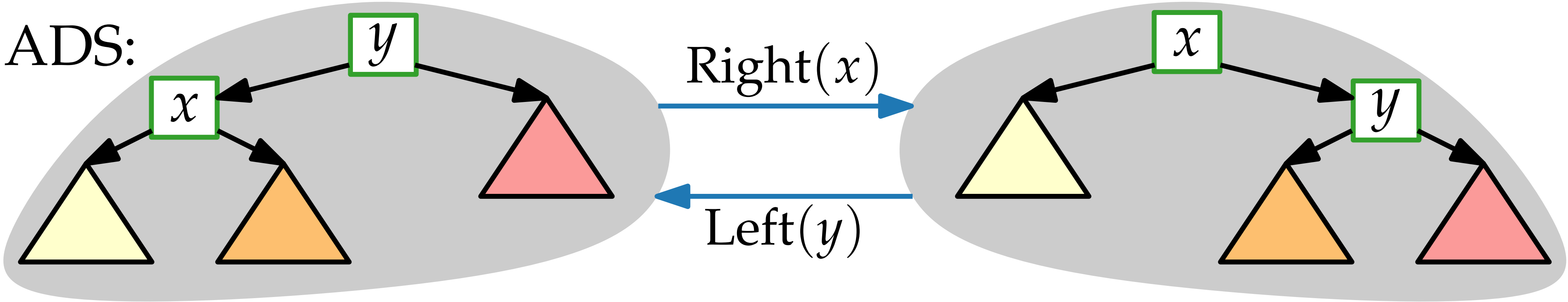


Daniel D. Sleator Robert E. Tarjan

J. ACM 1985

Idea: Whenever we query a key, rotate it to the root.

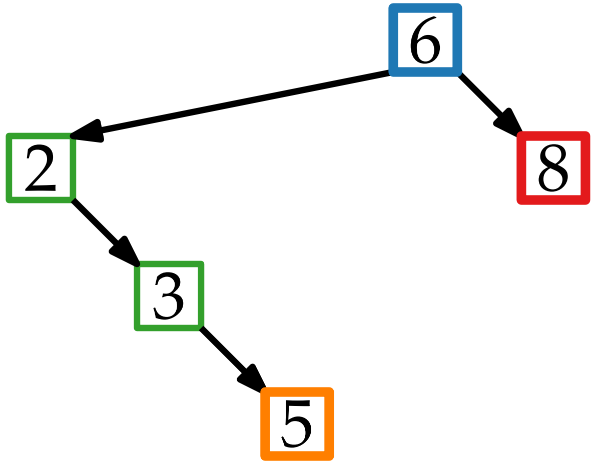
ADS:



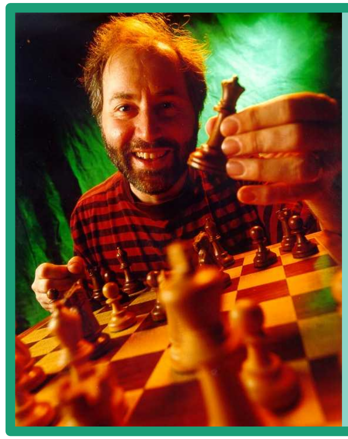
Splay(x): Rotate x to the root

Query(x): Splay(x), then return root

Query(8) Query(6) Query(5)



Splay Trees

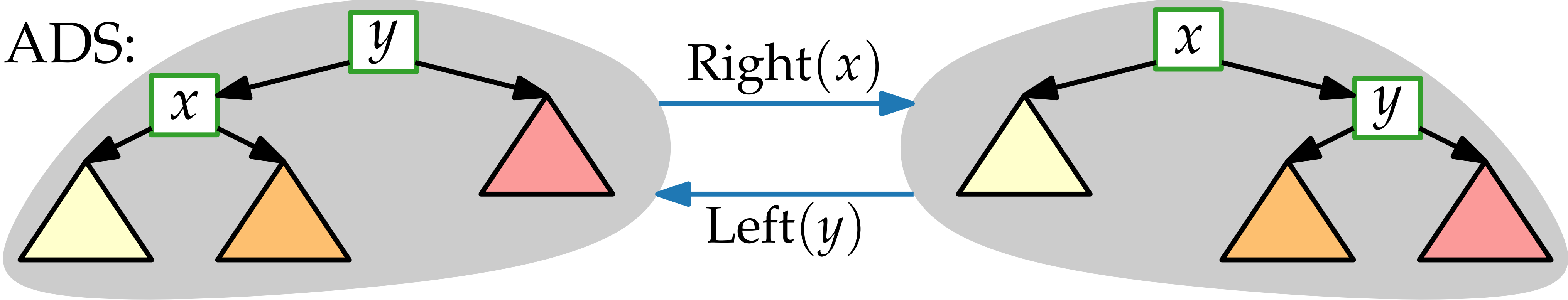


Daniel D. Sleator   Robert E. Tarjan

J. ACM 1985

Idea: Whenever we query a key, rotate it to the root.

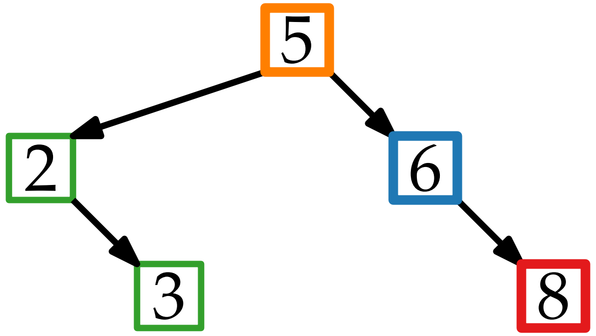
ADS:



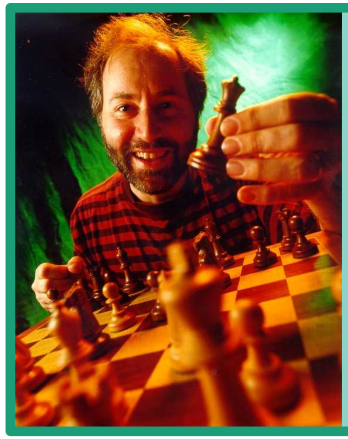
Splay(x): Rotate x to the root

Query(x): Splay(x), then return root

Query(8) Query(6) Query(5)



Splay Trees

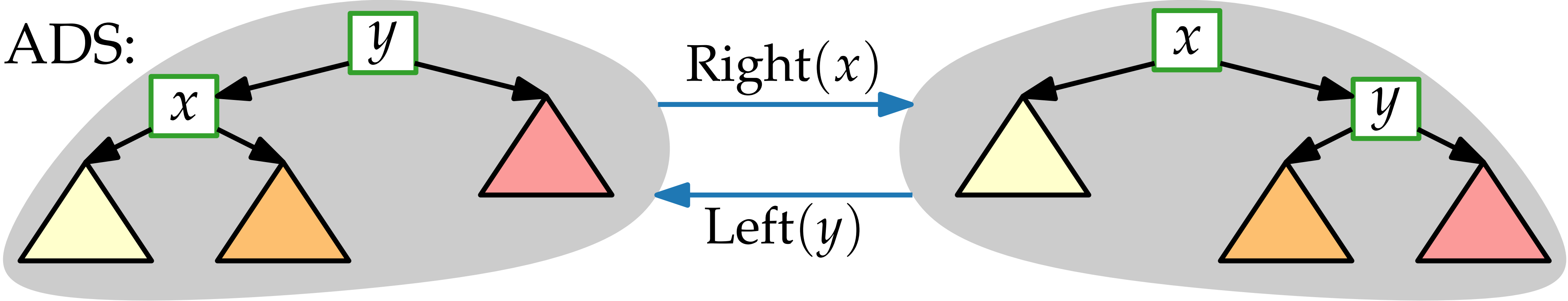


Daniel D. Sleator Robert E. Tarjan

J. ACM 1985

Idea: Whenever we query a key, rotate it to the root.

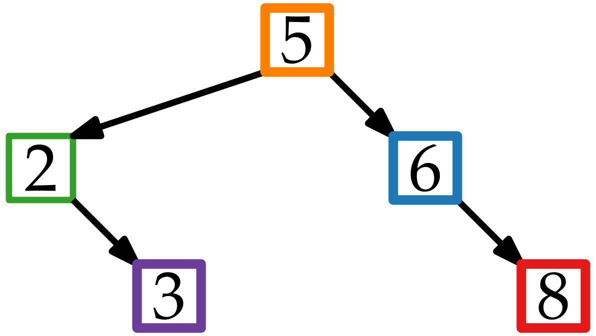
ADS:



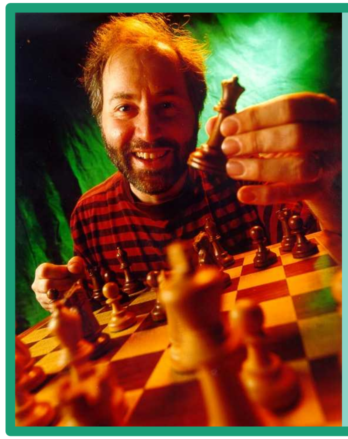
Splay(x): Rotate x to the root

Query(x): Splay(x), then return root

Query(8) Query(6) Query(5)
Query(3)



Splay Trees

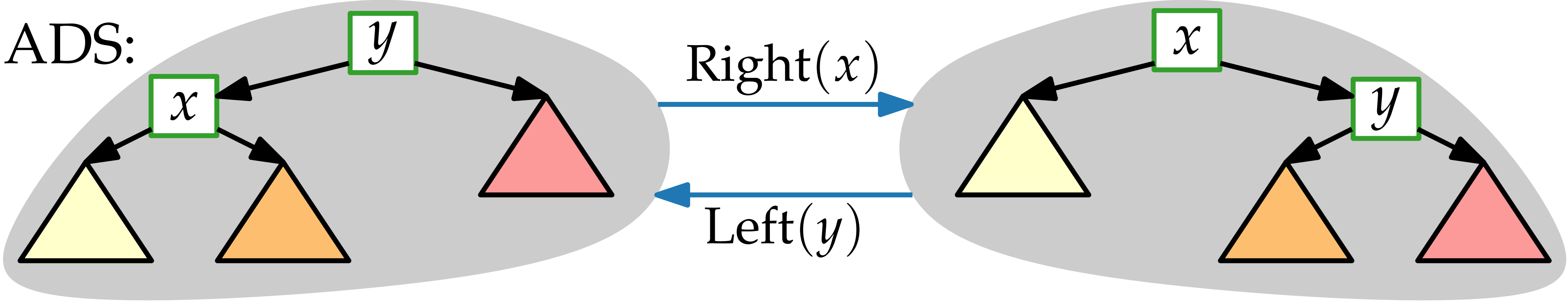


Daniel D. Sleator Robert E. Tarjan

J. ACM 1985

Idea: Whenever we query a key, rotate it to the root.

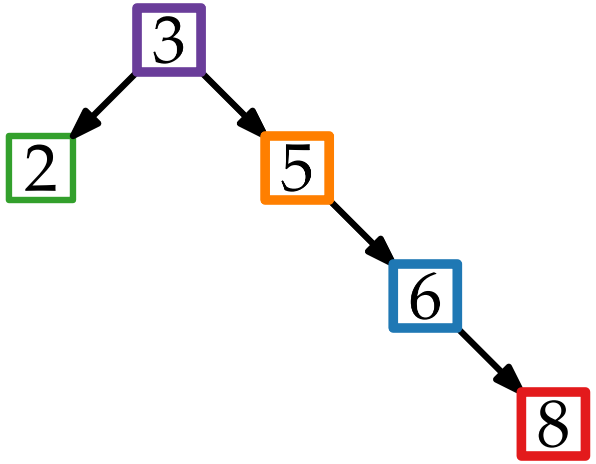
ADS:



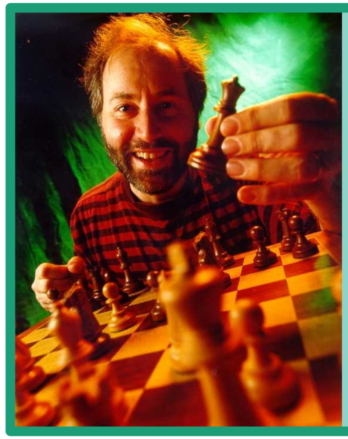
Splay(x): Rotate x to the root

Query(x): Splay(x), then return root

Query(8) Query(6) Query(5)
Query(3)



Splay Trees

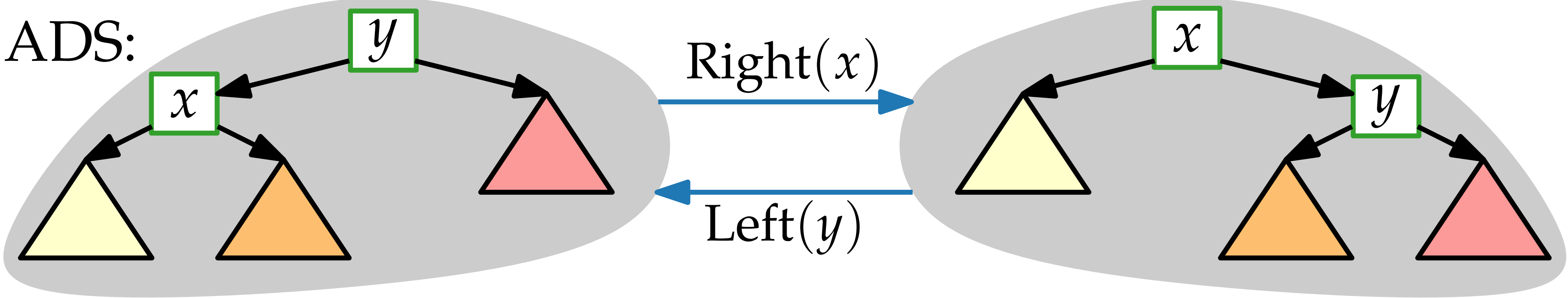


Daniel D. Sleator   Robert E. Tarjan

J. ACM 1985

Idea: Whenever we query a key, rotate it to the root.

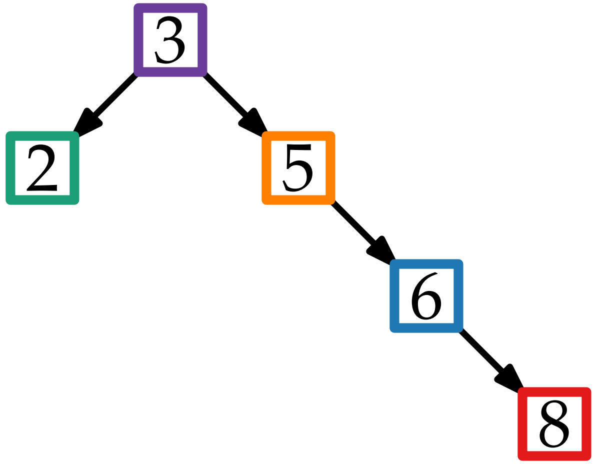
ADS:



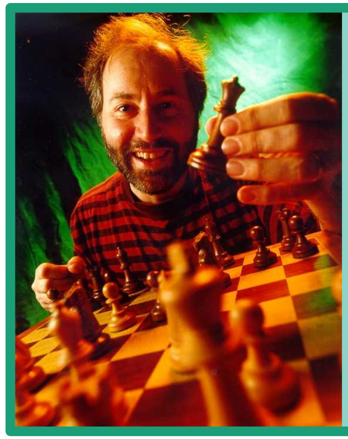
Splay(x): Rotate x to the root

Query(x): Splay(x), then return root

Query(8) Query(6) Query(5)
Query(3)
Query(2)



Splay Trees

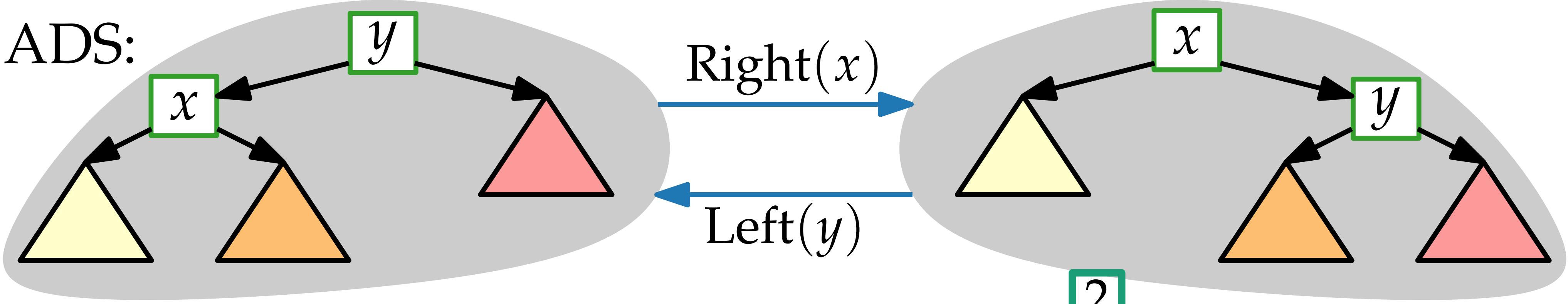


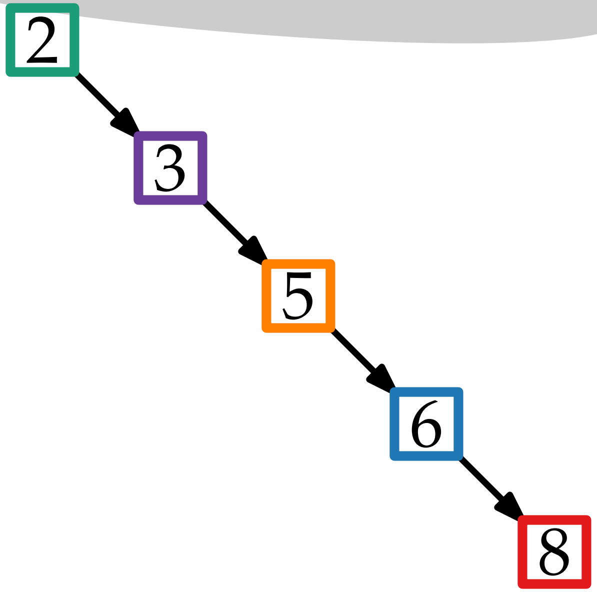
Daniel D. Sleator Robert E. Tarjan

J. ACM 1985

Idea: Whenever we query a key, rotate it to the root.

ADS:



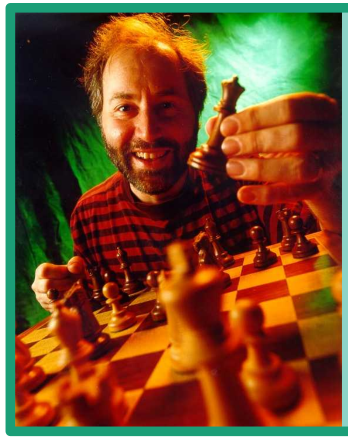


Splay(x): Rotate x to the root

Query(x): Splay(x), then return root

Query(8) Query(6) Query(5)
Query(3)
Query(2)

Splay Trees

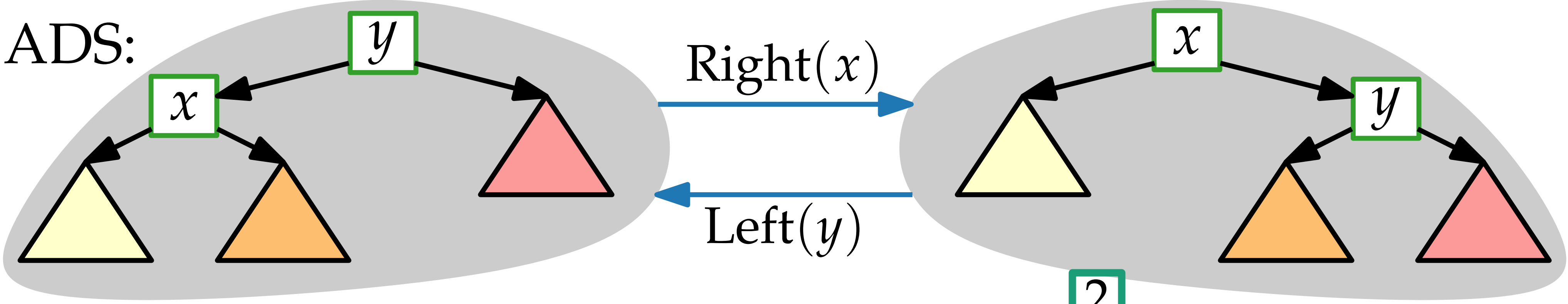


Daniel D. Sleator | Robert E. Tarjan

J. ACM 1985

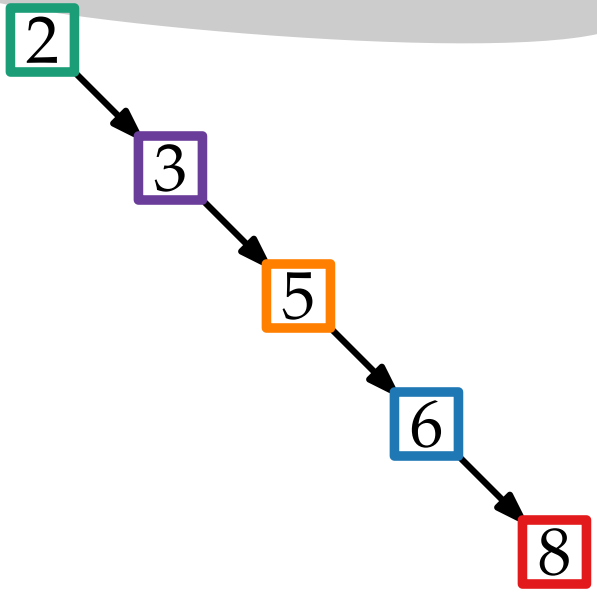
Idea: Whenever we query a key, rotate it to the root.

ADS:



Splay(x): Rotate x to the root

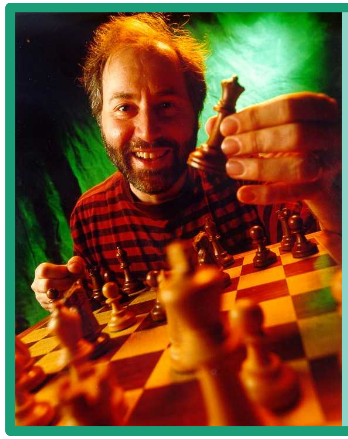
Query(x): Splay(x), then return root



Query(8) Query(6) Query(5)
Query(3)
Query(2)

We're back at the start...

Splay Trees



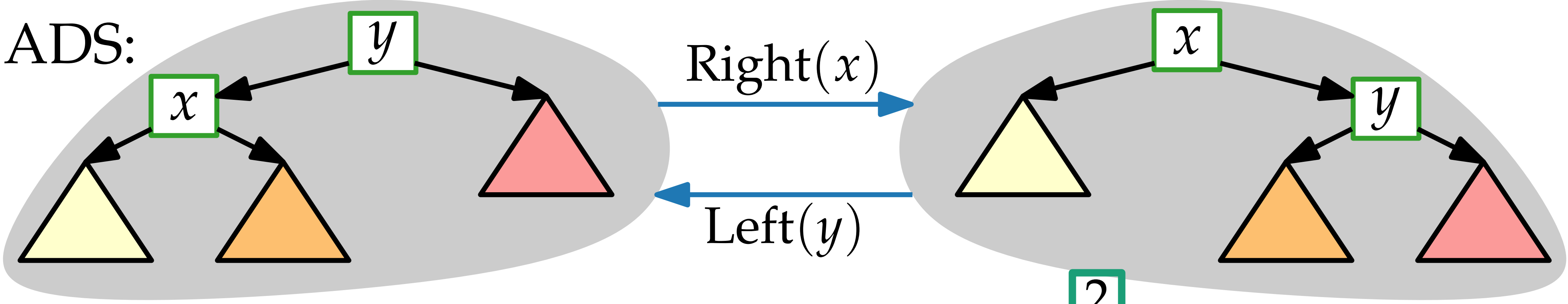
Daniel D. Sleator

Robert E. Tarjan

J. ACM 1985

Idea: Whenever we query a key, rotate it to the root.

ADS:

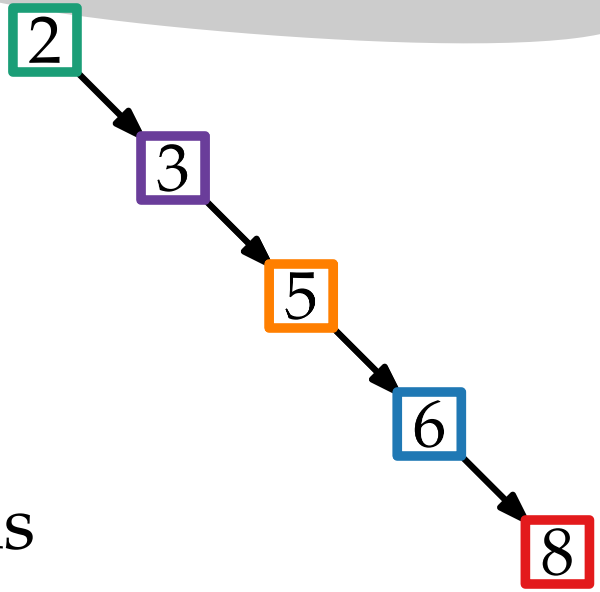


Splay(x): Rotate x to the root

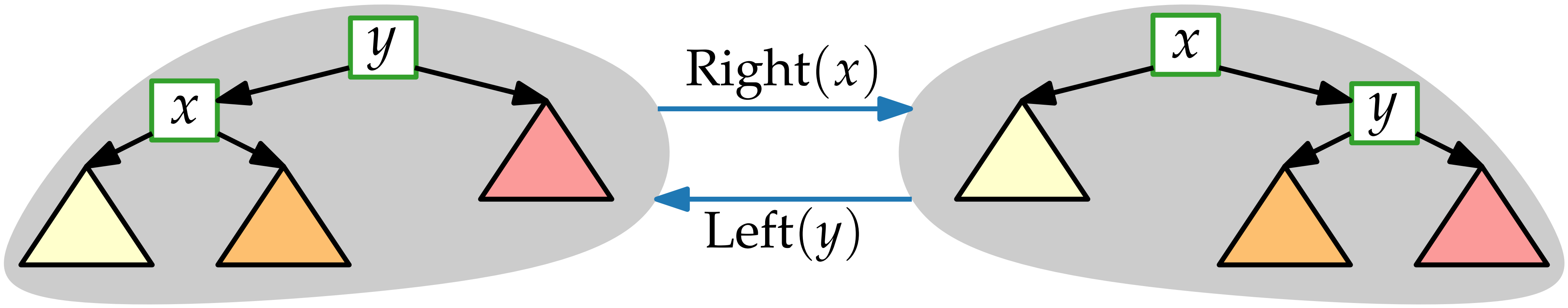
Query(x): Splay(x), then return root

Query(8) Query(6) Query(5)
Query(3)
Query(2)

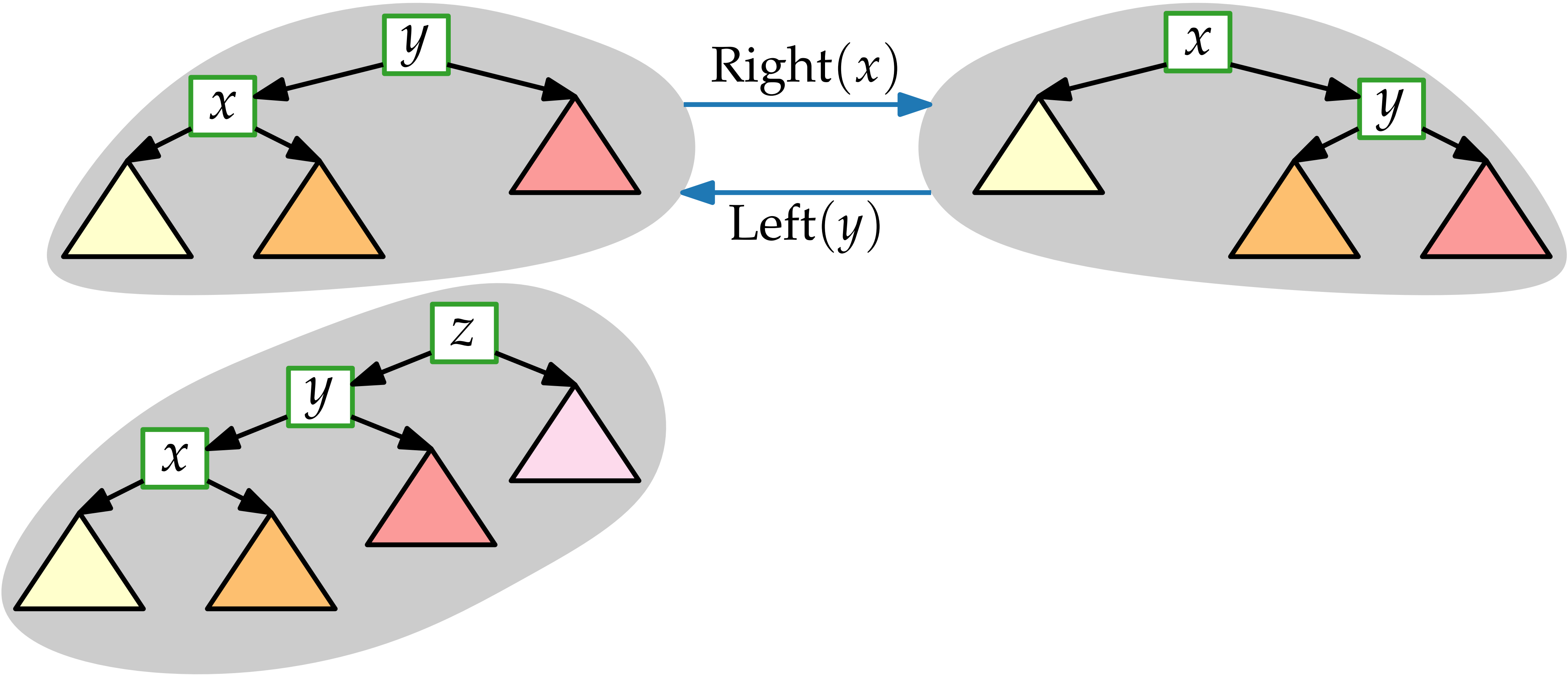
We're back at the start...
and we did $\Theta(n^2)$ rotations



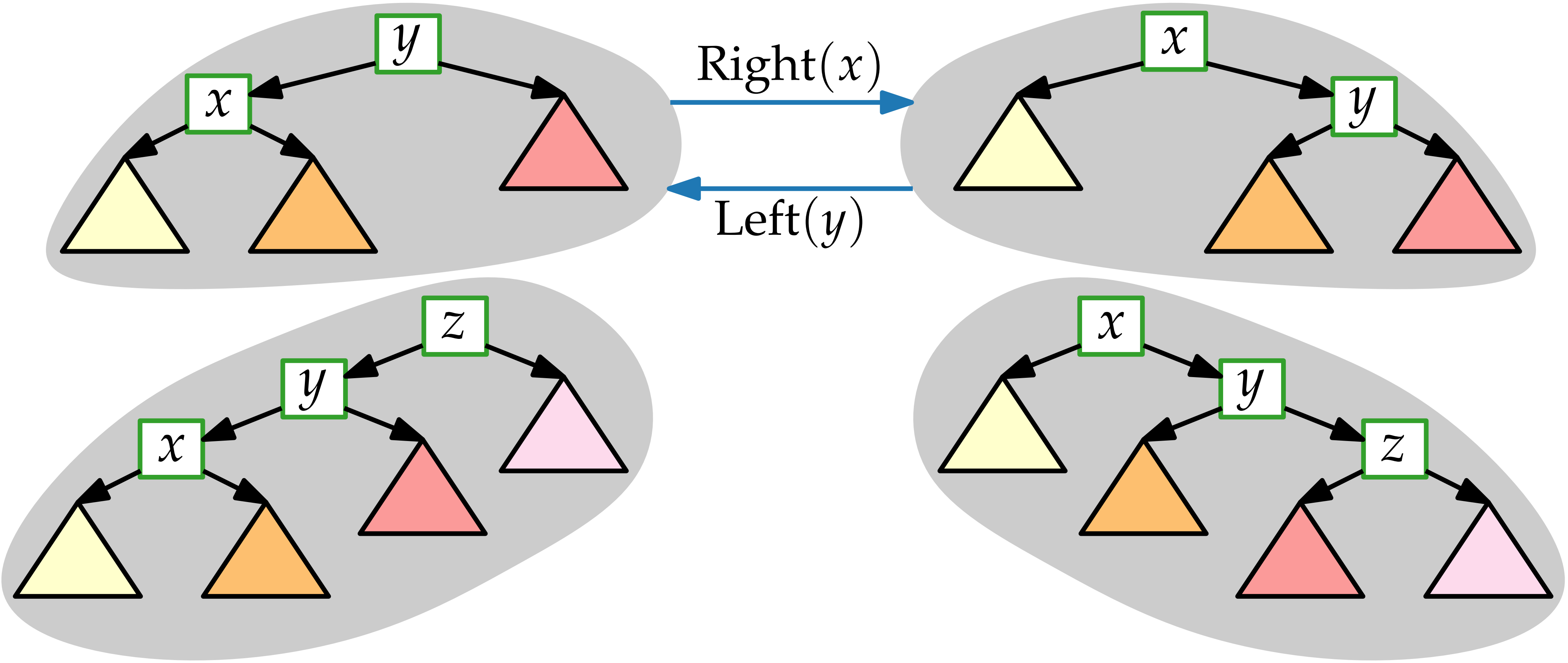
Rotations II



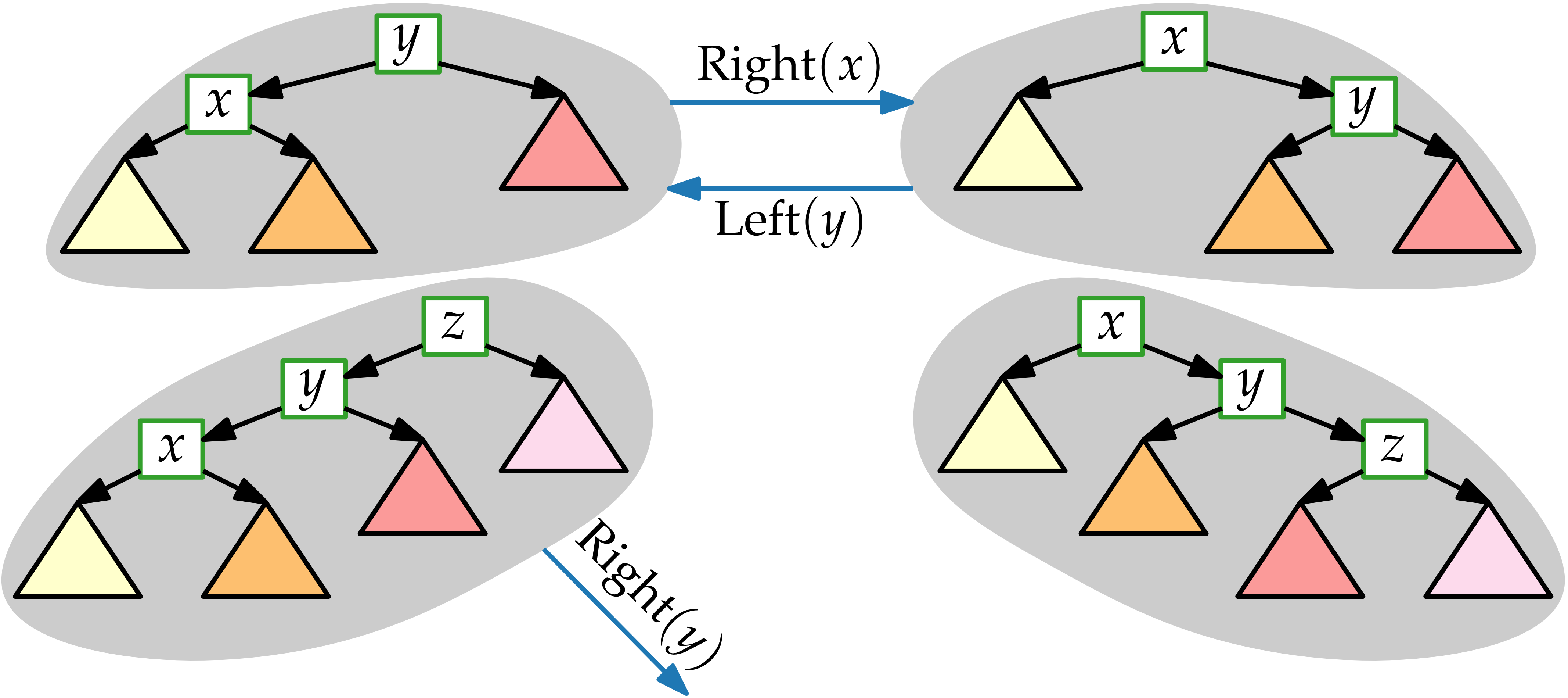
Rotations II



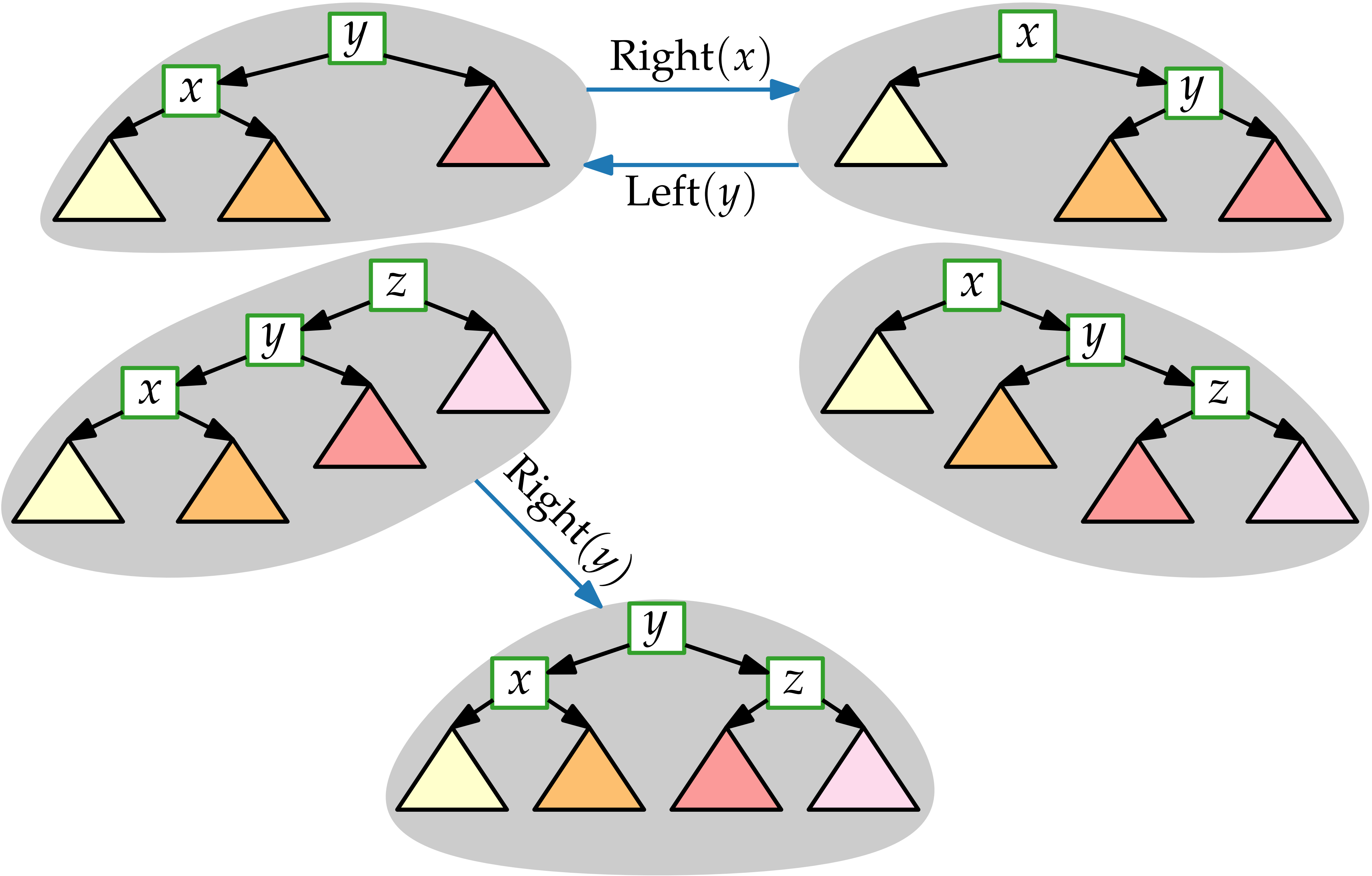
Rotations II



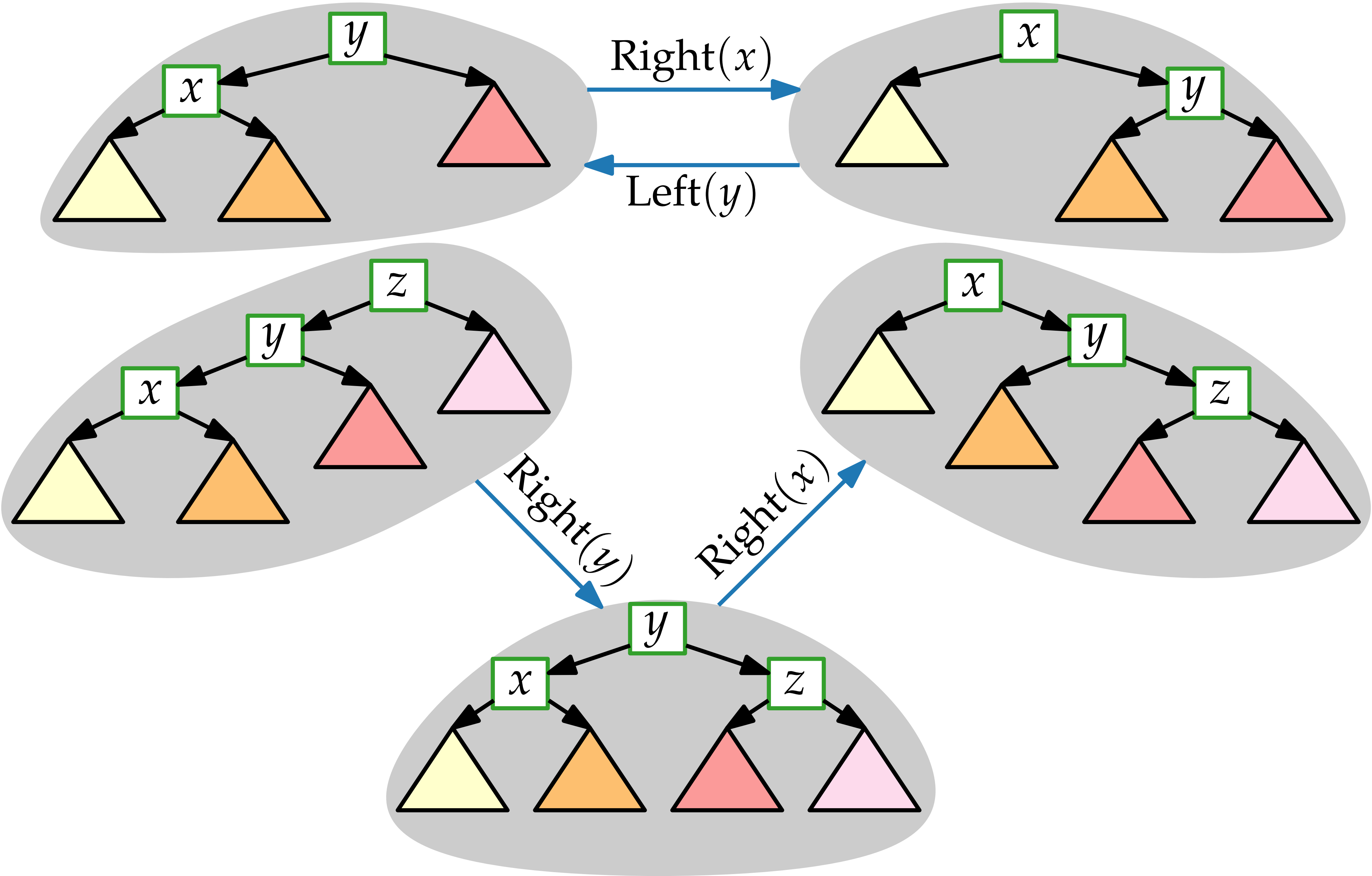
Rotations II



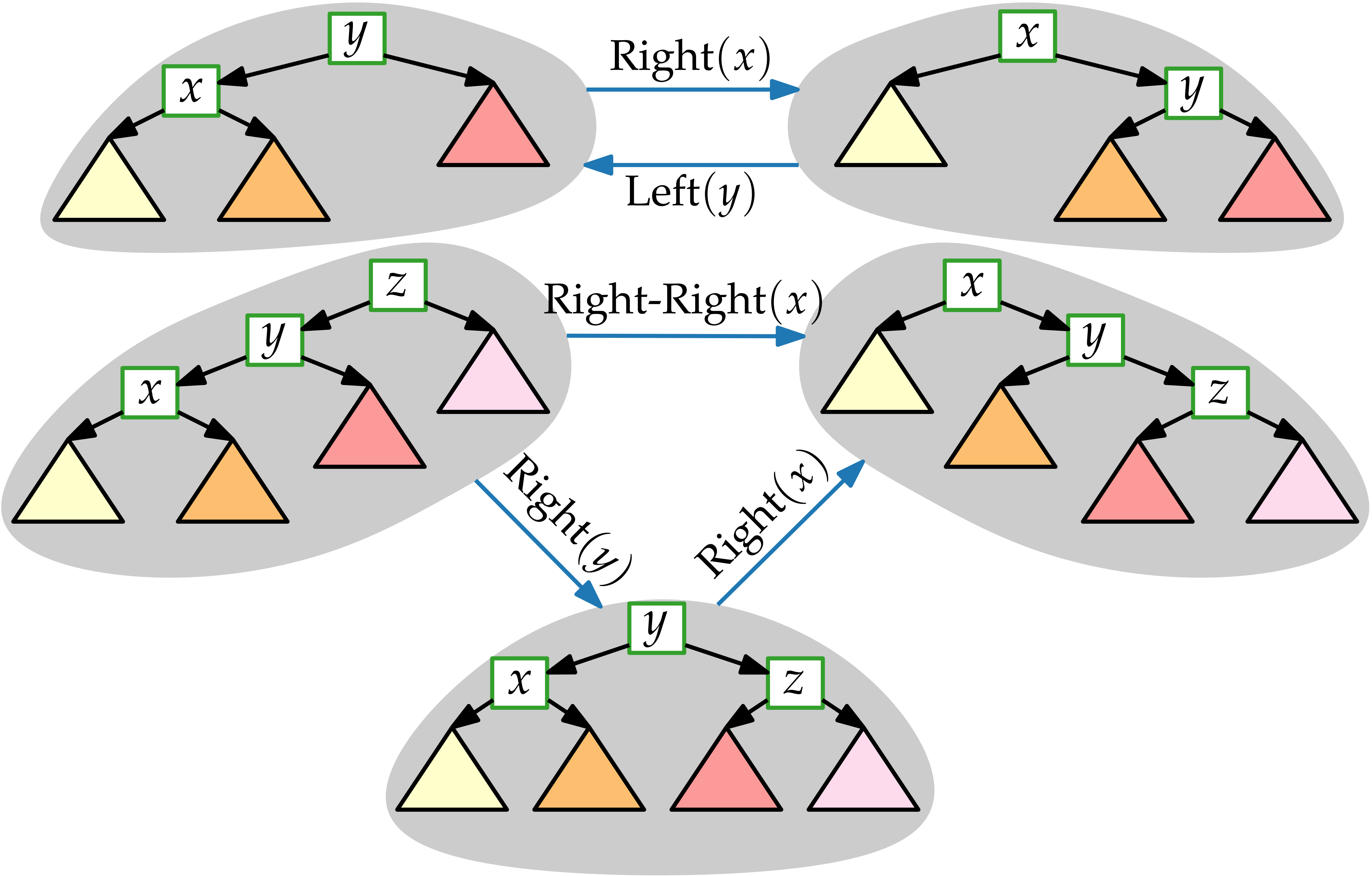
Rotations II



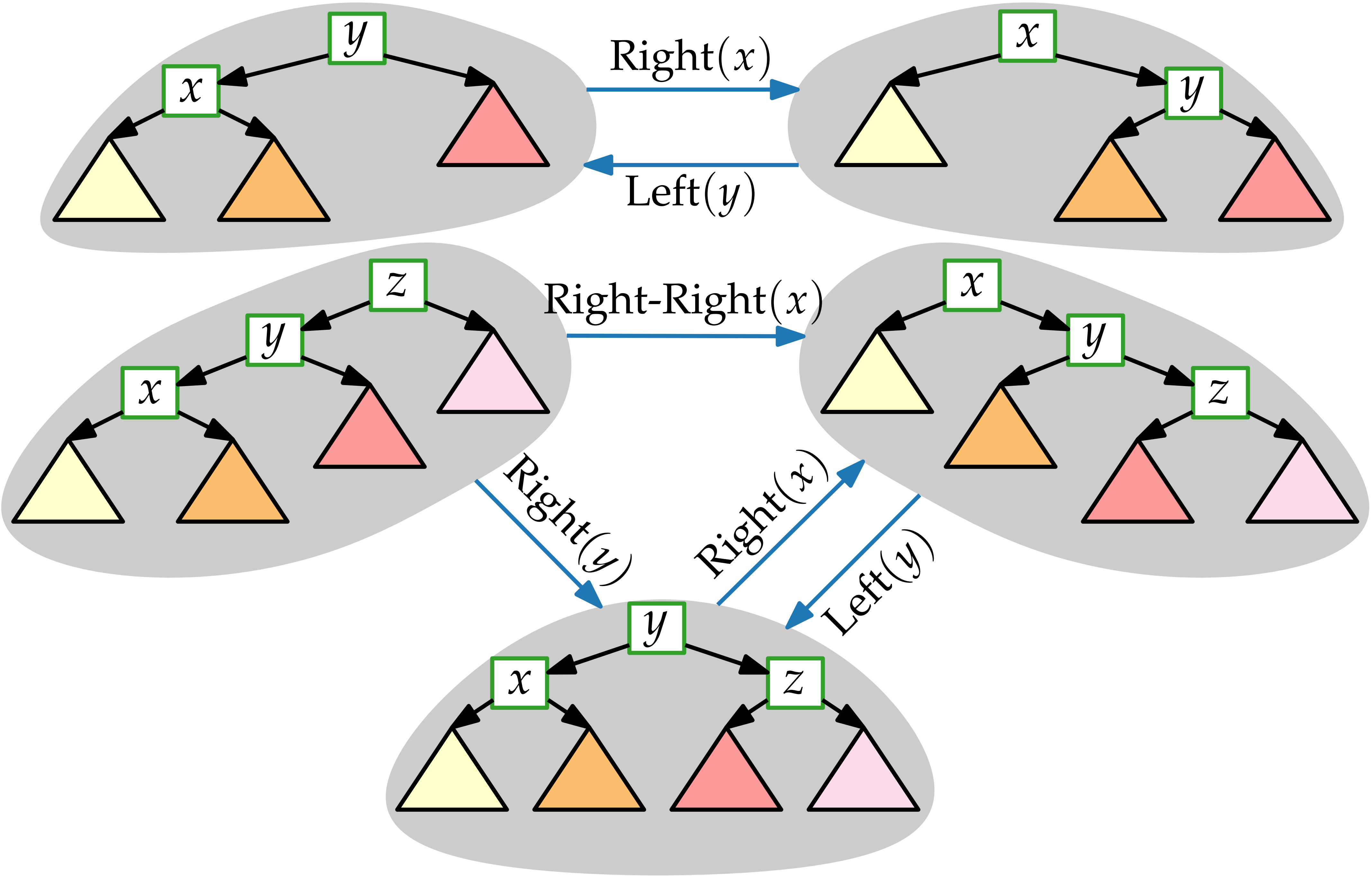
Rotations II



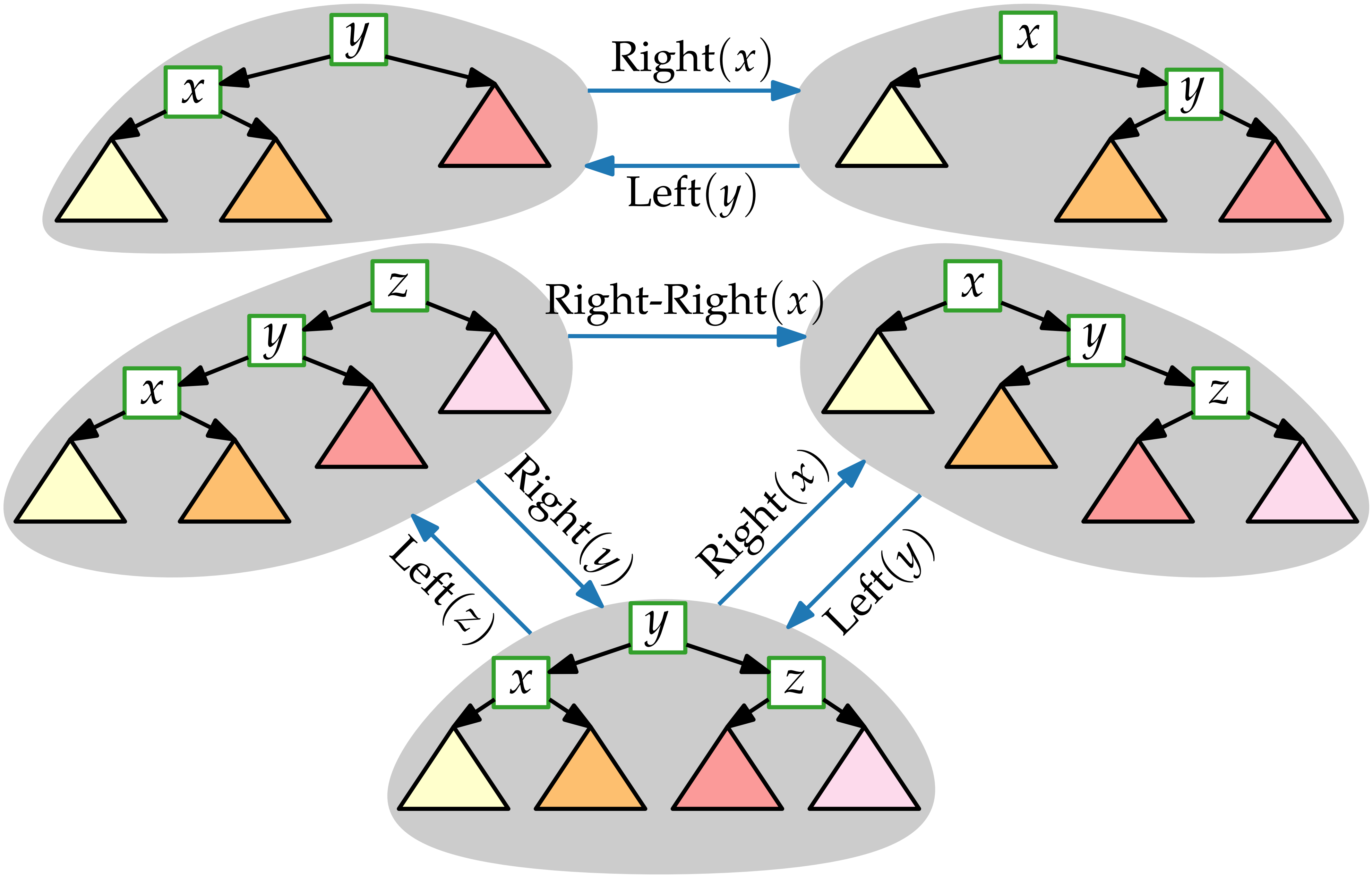
Rotations II



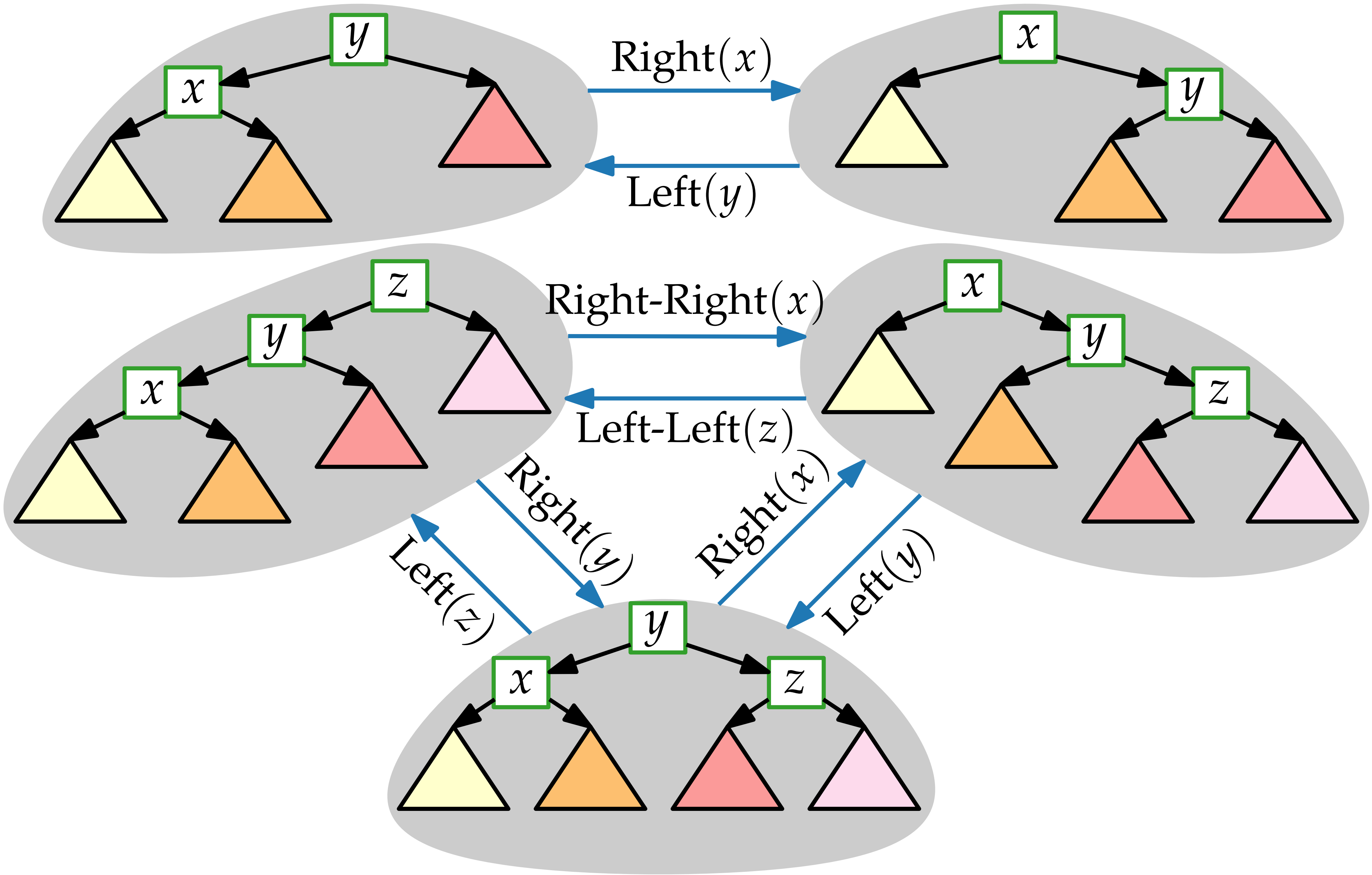
Rotations II



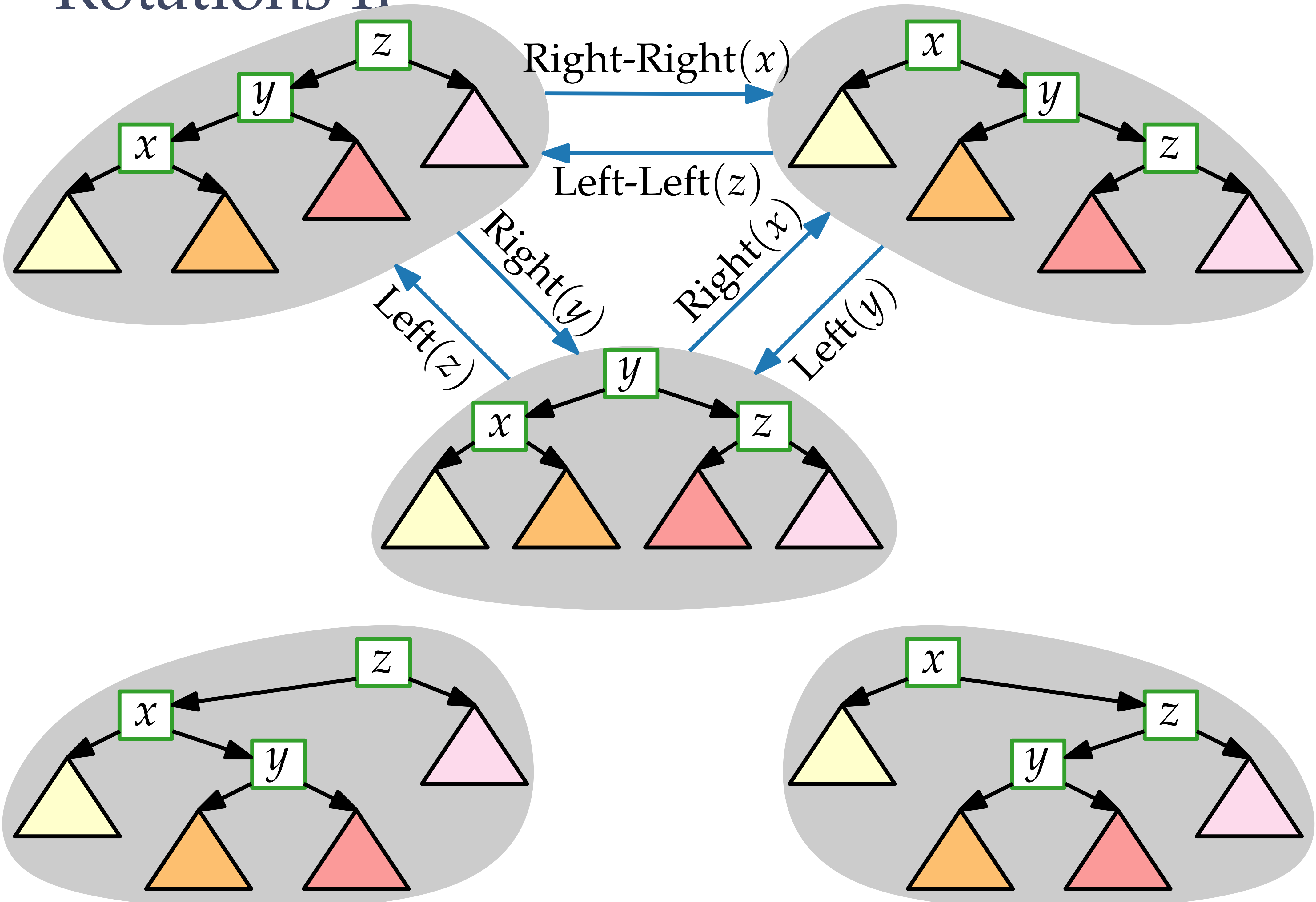
Rotations II



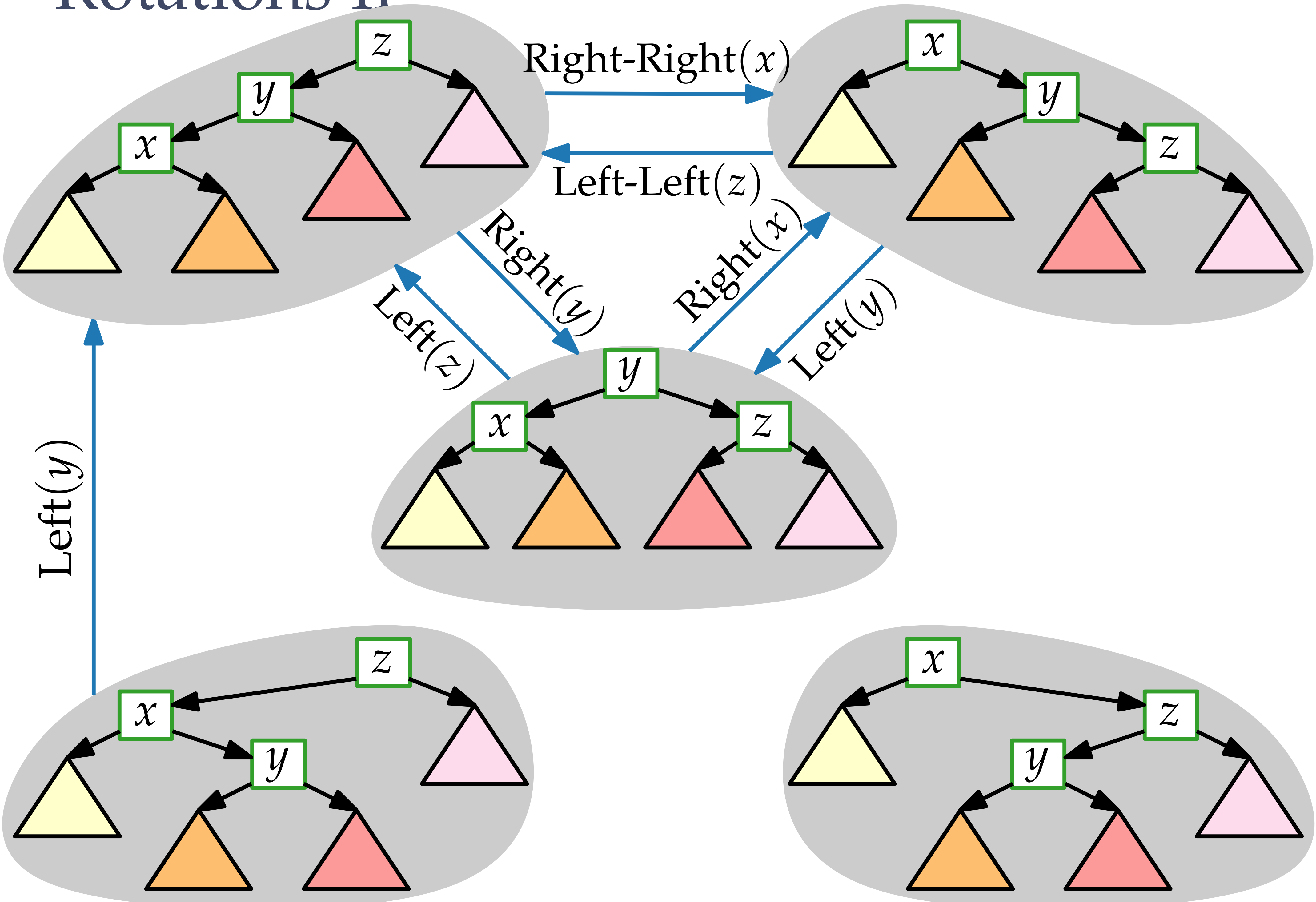
Rotations II



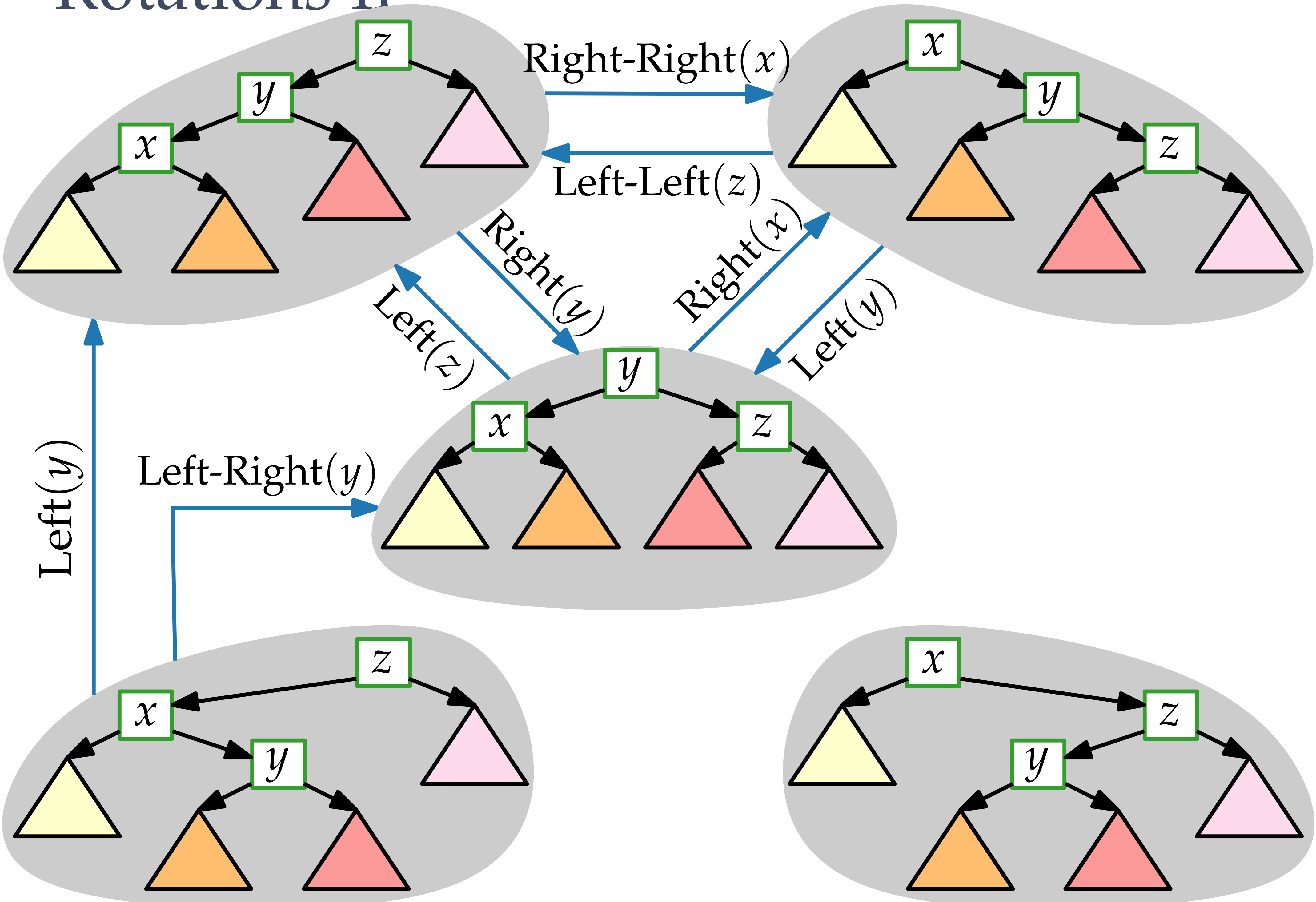
Rotations II



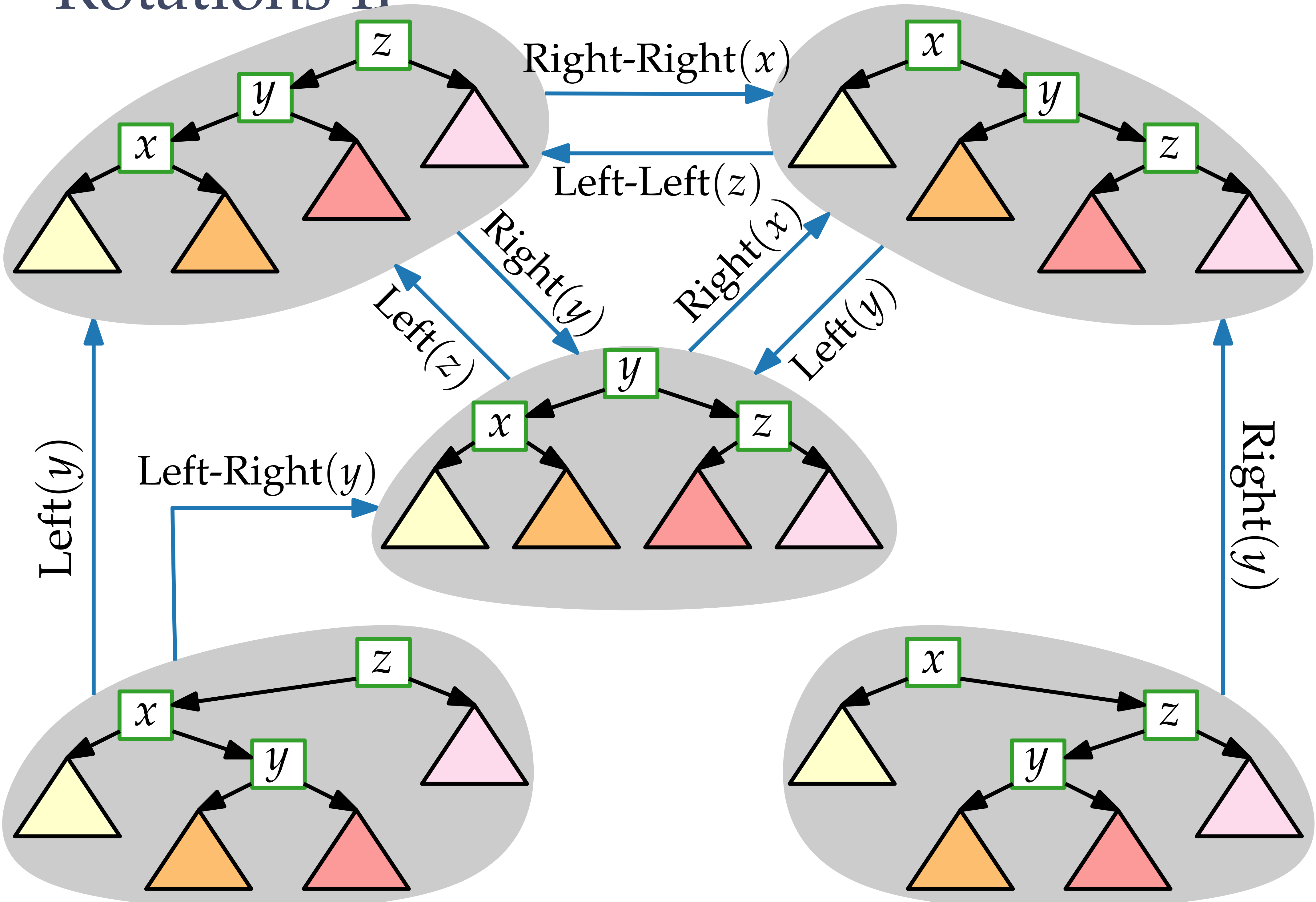
Rotations II



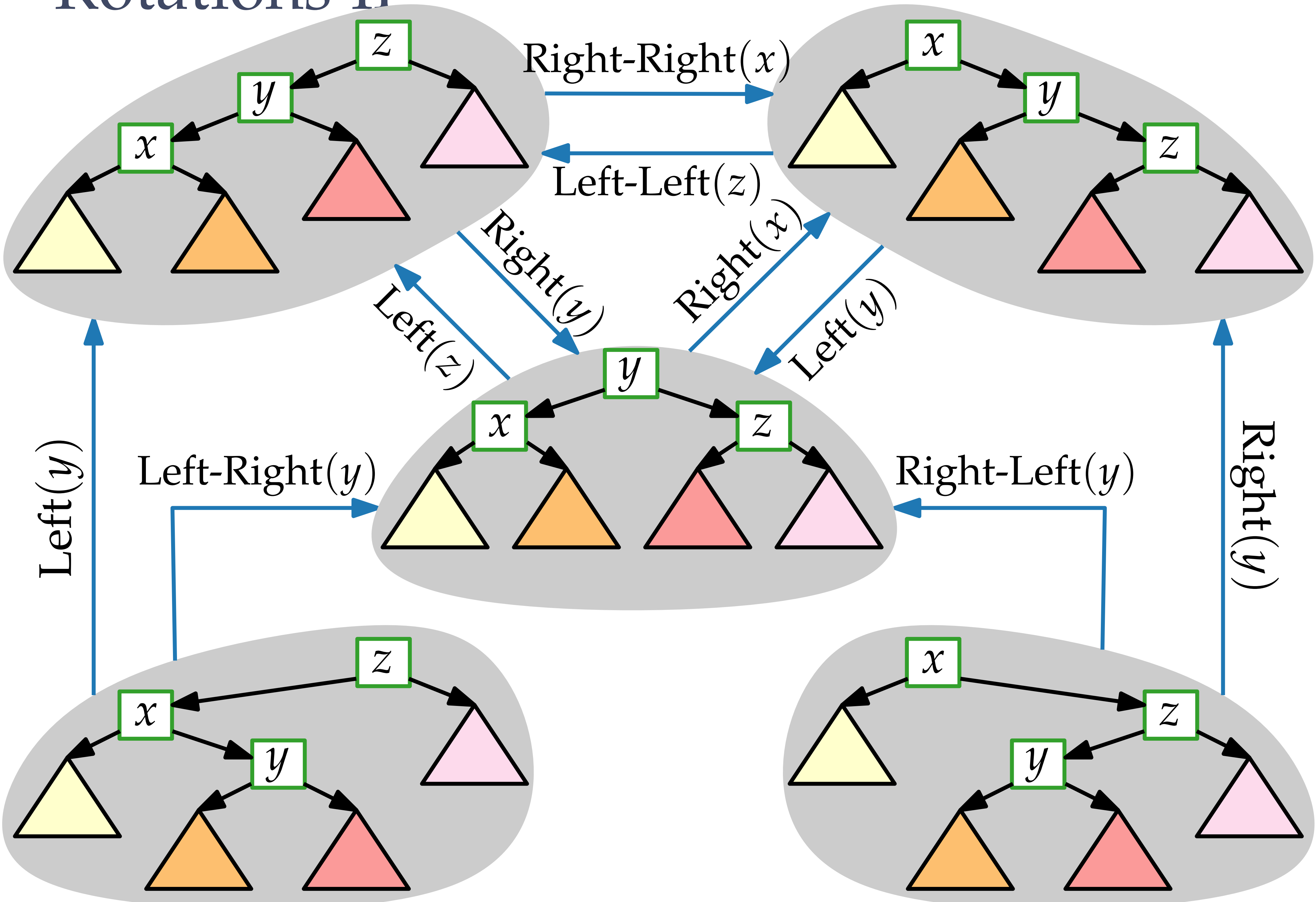
Rotations II



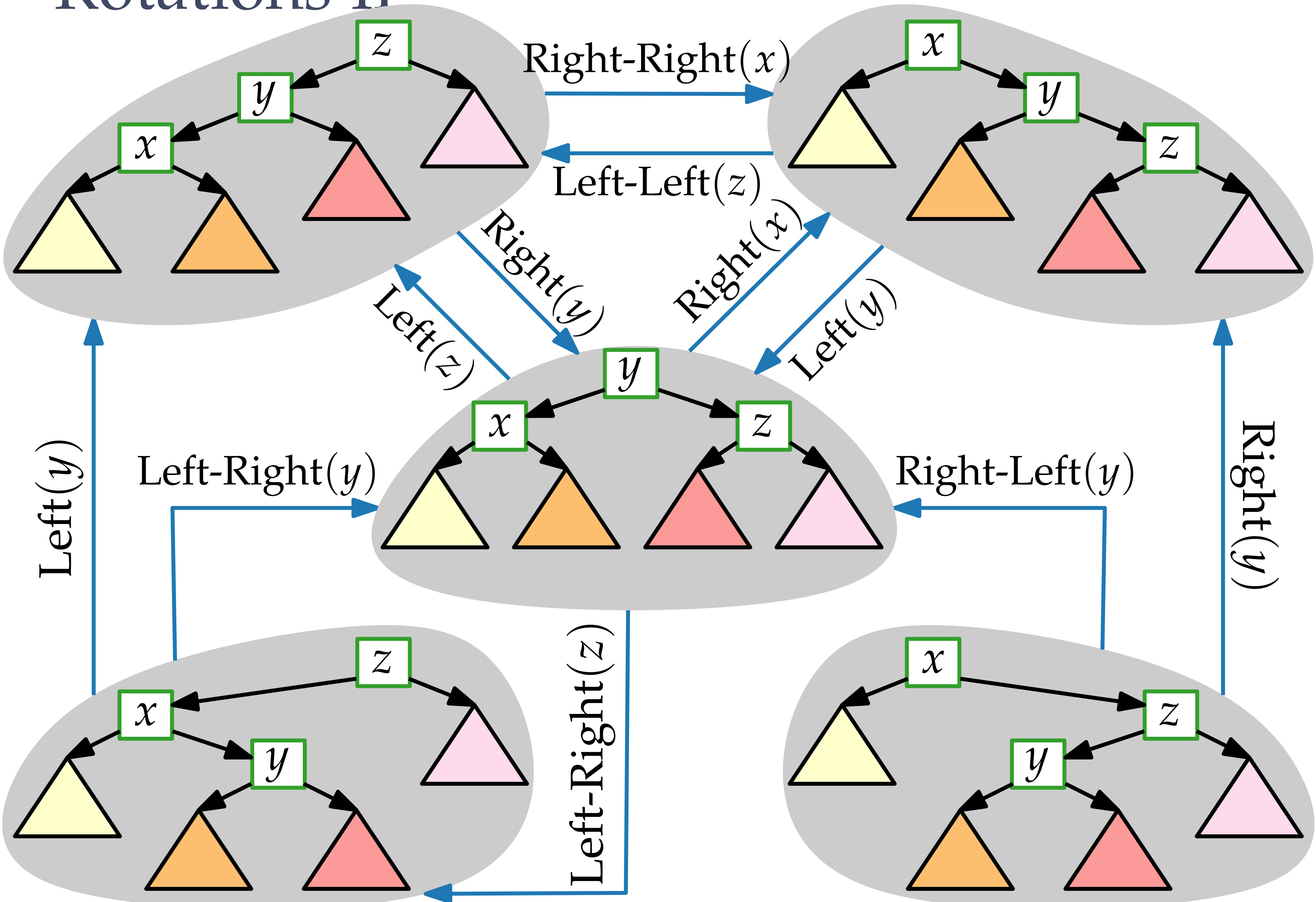
Rotations II



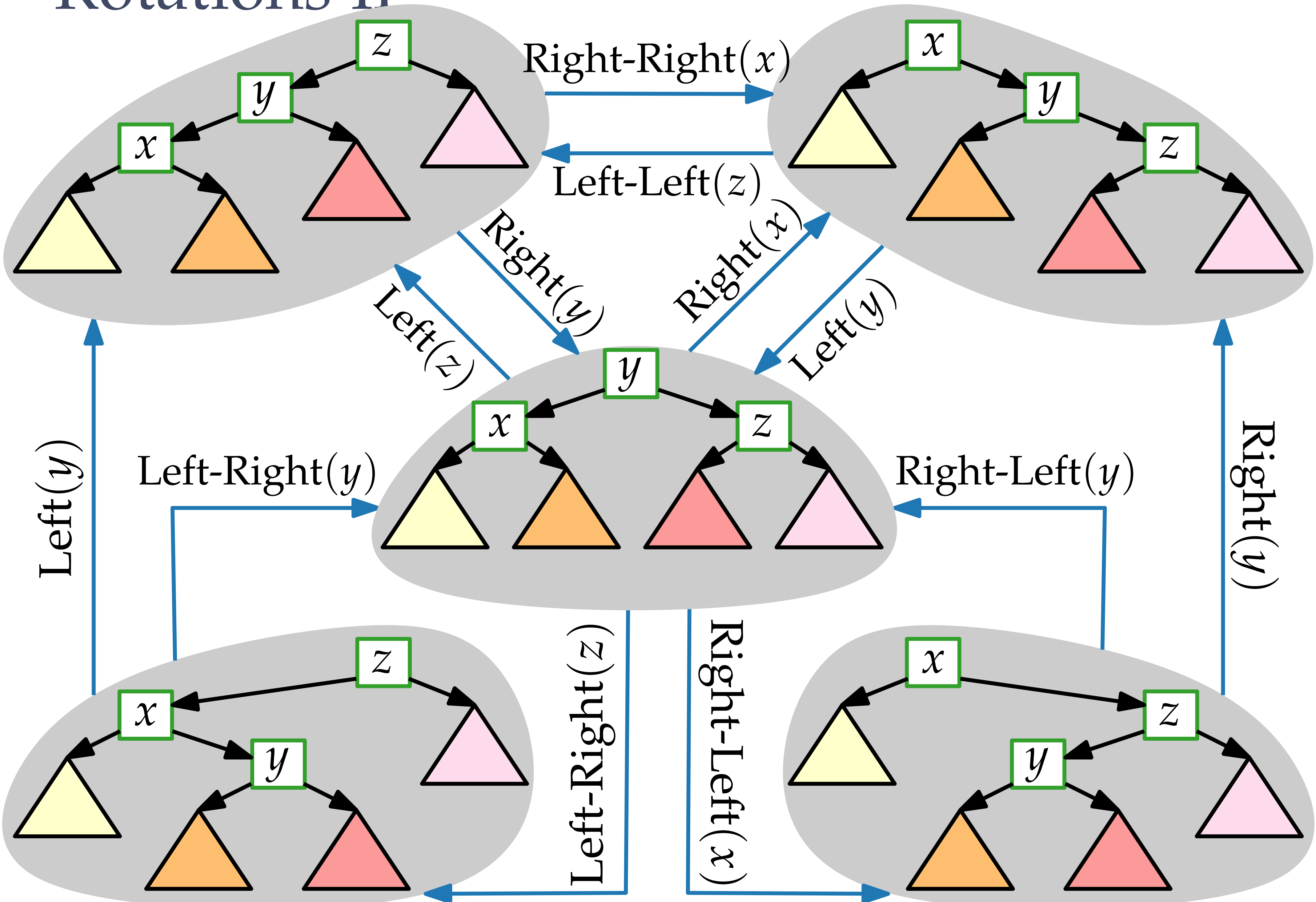
Rotations II



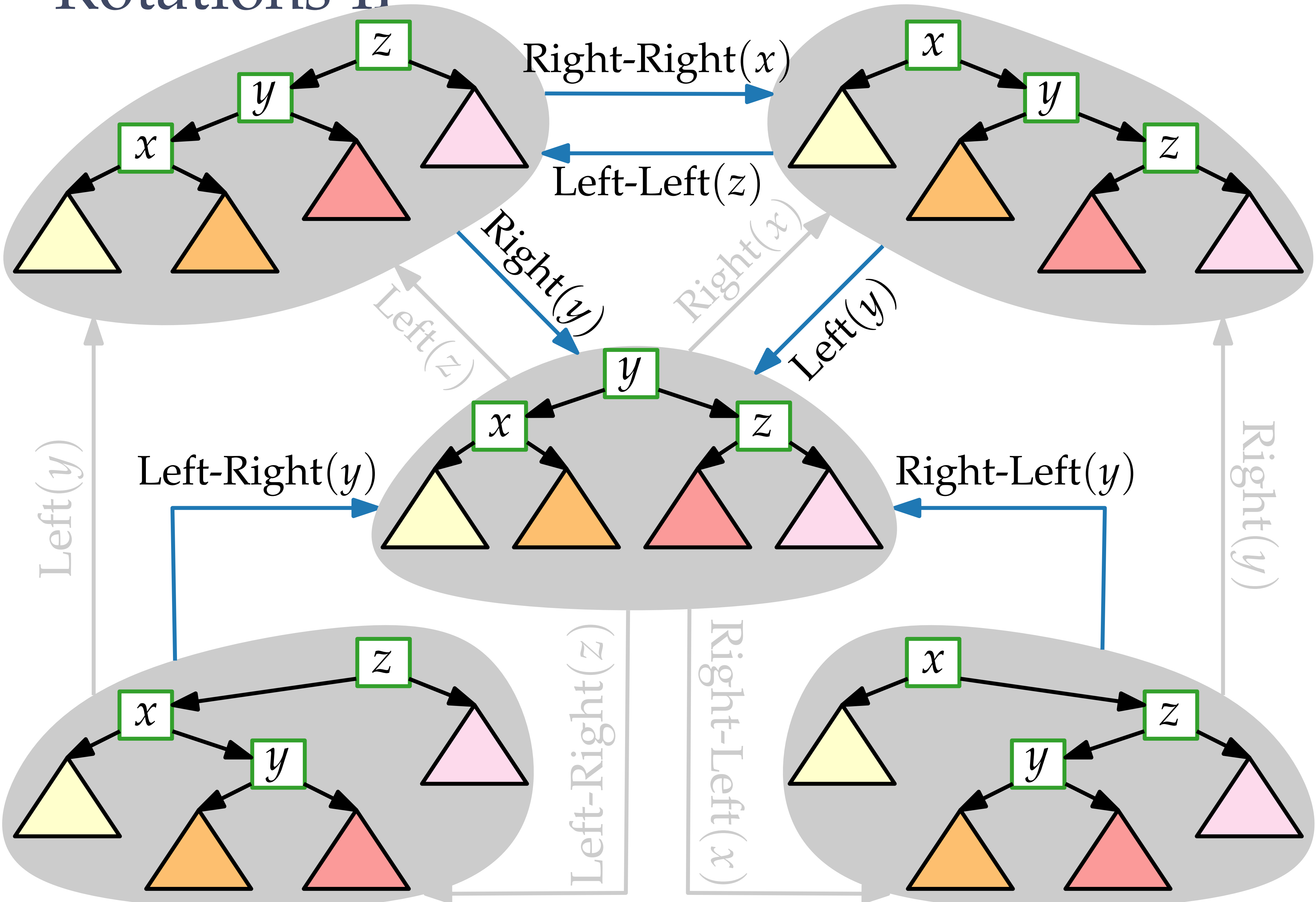
Rotations II



Rotations II



Rotations II



Splay

Algorithm: Splay(x)

Splay

Algorithm: Splay(x)

if $x \neq root$ **then**

Splay

Algorithm: Splay(x)

if $x \neq root$ **then**

  y = parent of x

Splay

Algorithm: Splay(x)

if $x \neq root$ **then**

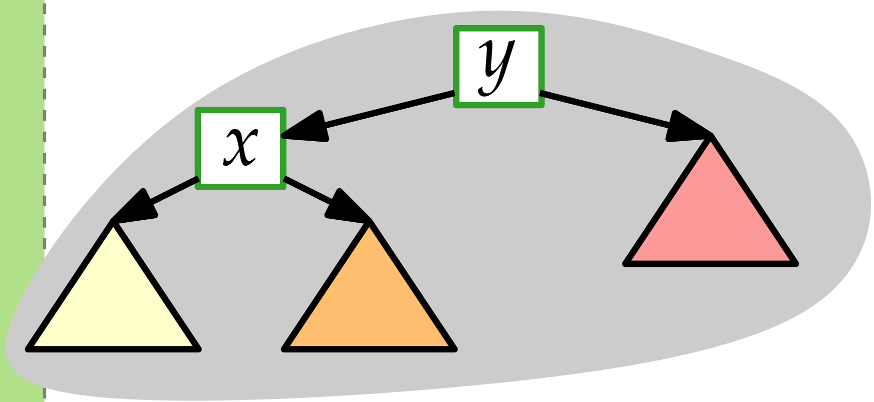
  y = parent of x

  if $y = root$ **then**

Splay

Algorithm: Splay(x)

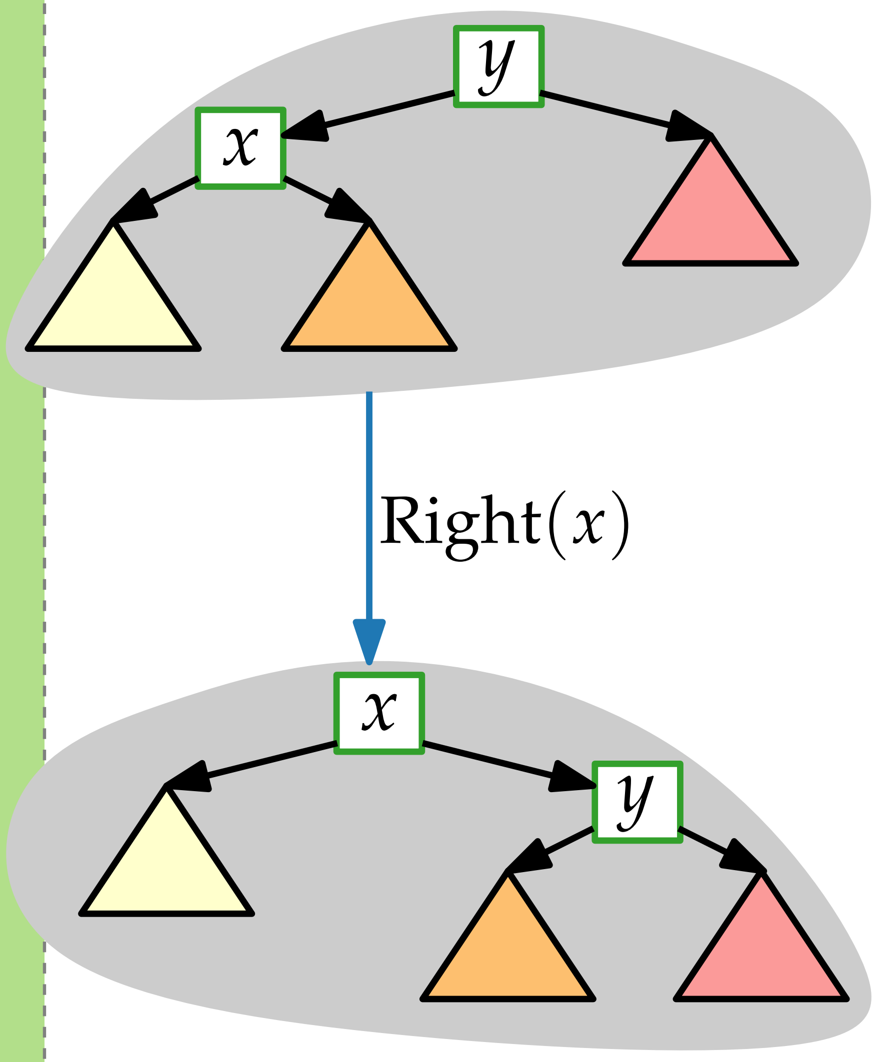
if $x \neq root$ **then**
  $y =$ parent of x
  if $y = root$ **then**
    if $x < y$ **then**



Splay

Algorithm: Splay(x)

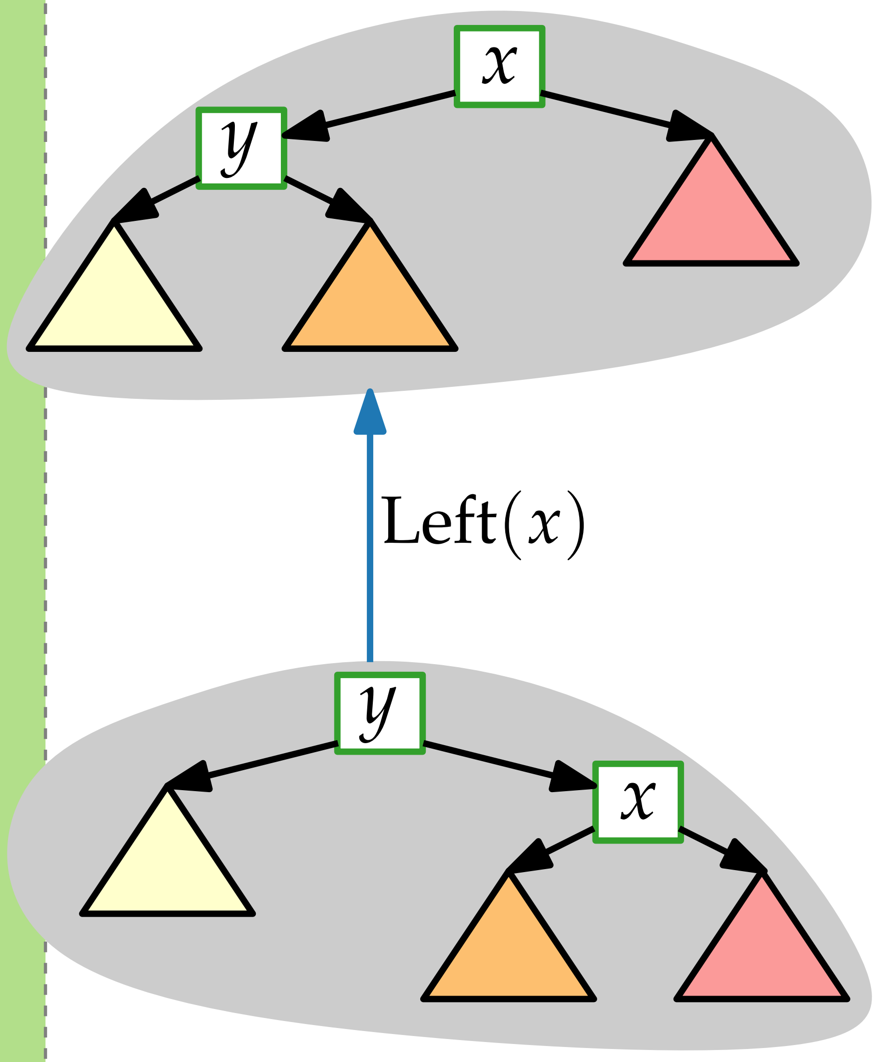
if $x \neq \mathit{root}$ **then**
  y = parent of x
  if $y = \mathit{root}$ **then**
    if $x < y$ **then** Right(x)



Splay

Algorithm: Splay(x)

if $x \neq \textit{root}$ **then**
- $y =$ parent of x
- **if** $y = \textit{root}$ **then**
  - **if** $x < y$ **then** Right(x)
  - **if** $y < x$ **then** Left(x)



Splay

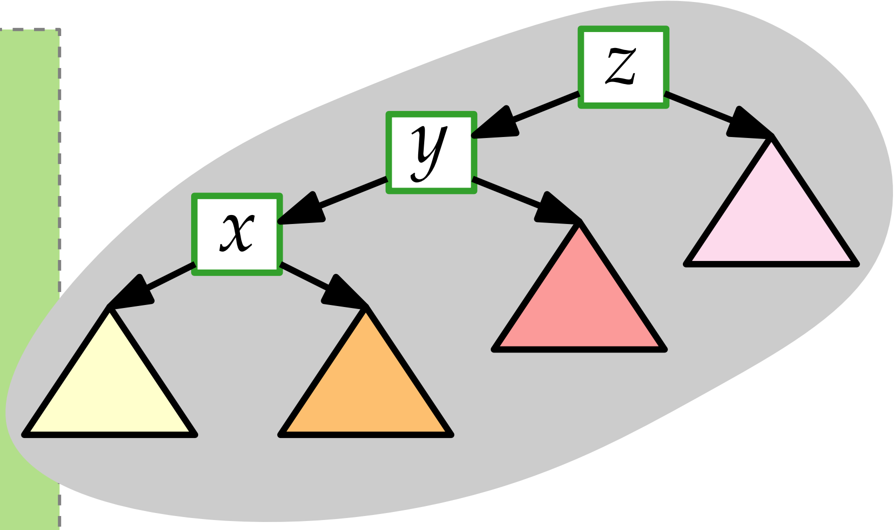
Algorithm: Splay(x)

if $x \neq \mathit{root}$ **then**
    y = parent of x
    if $y = \mathit{root}$ **then**
        if $x < y$ **then** Right(x)
        if $y < x$ **then** Left(x)
    else
        z = parent of y

Splay

Algorithm: Splay(x)

if $x \neq root$ **then**
    $y =$ parent of x
    if $y = root$ **then**
        if $x < y$ **then** Right(x)
        if $y < x$ **then** Left(x)
    else
        $z =$ parent of y
        if $x < y < z$ **then**



Splay

Algorithm: Splay(x)

if $x \neq$ *root* **then**

  $y =$ parent of x

  if $y =$ *root* **then**

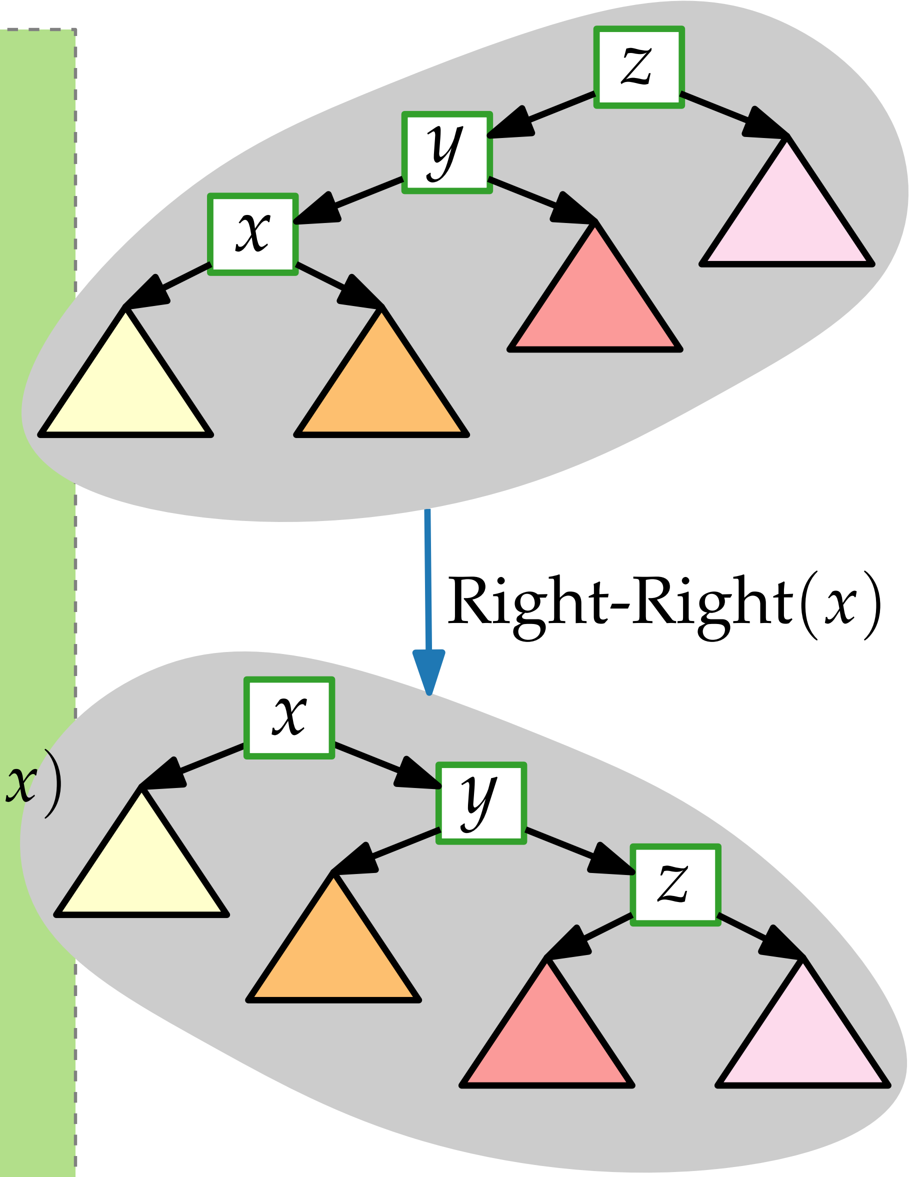
    if $x < y$ **then** Right(x)

    if $y < x$ **then** Left(x)

  else

    $z =$ parent of y

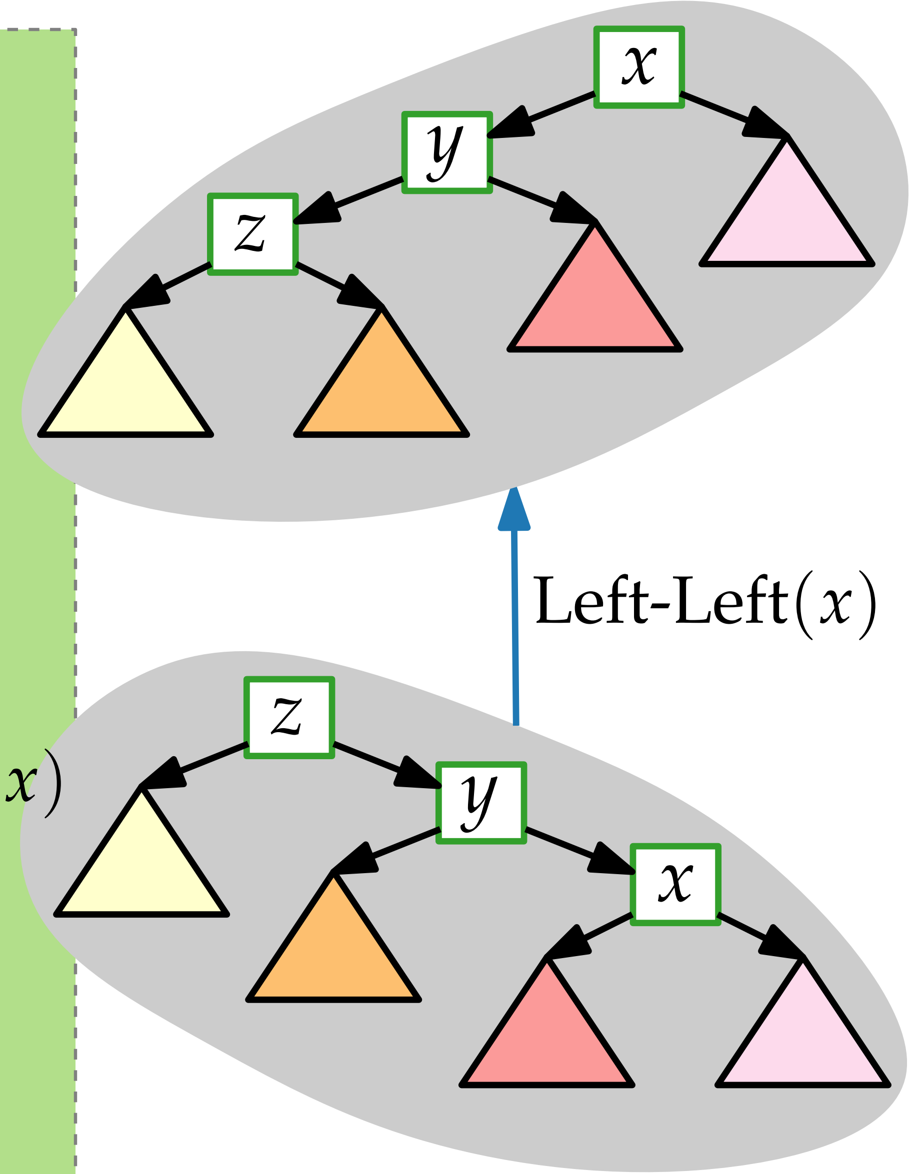
    if $x < y < z$ **then** Right-Right(x)



Splay

Algorithm: Splay(x)

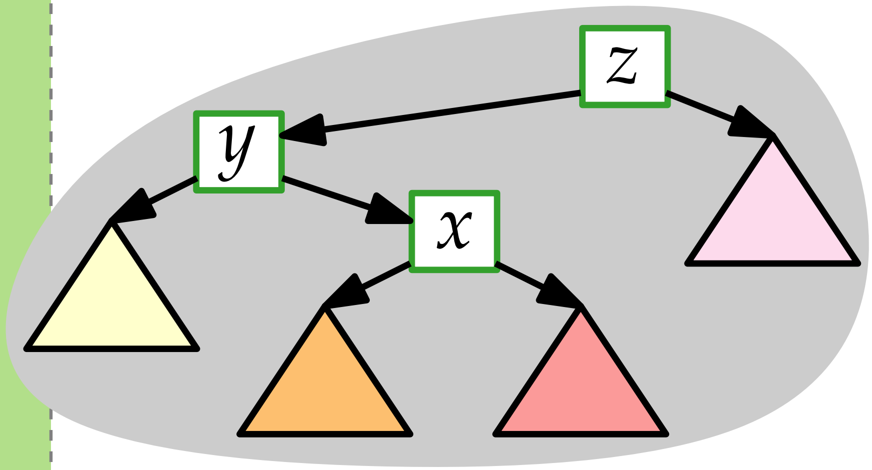
if $x \neq root$ **then**
  $y =$ parent of x
  if $y = root$ **then**
    if $x < y$ **then** Right(x)
    if $y < x$ **then** Left(x)
  else
    $z =$ parent of y
    if $x < y < z$ **then** Right-Right(x)
    if $z < y < x$ **then** Left-Left(x)



Splay

Algorithm: Splay(x)

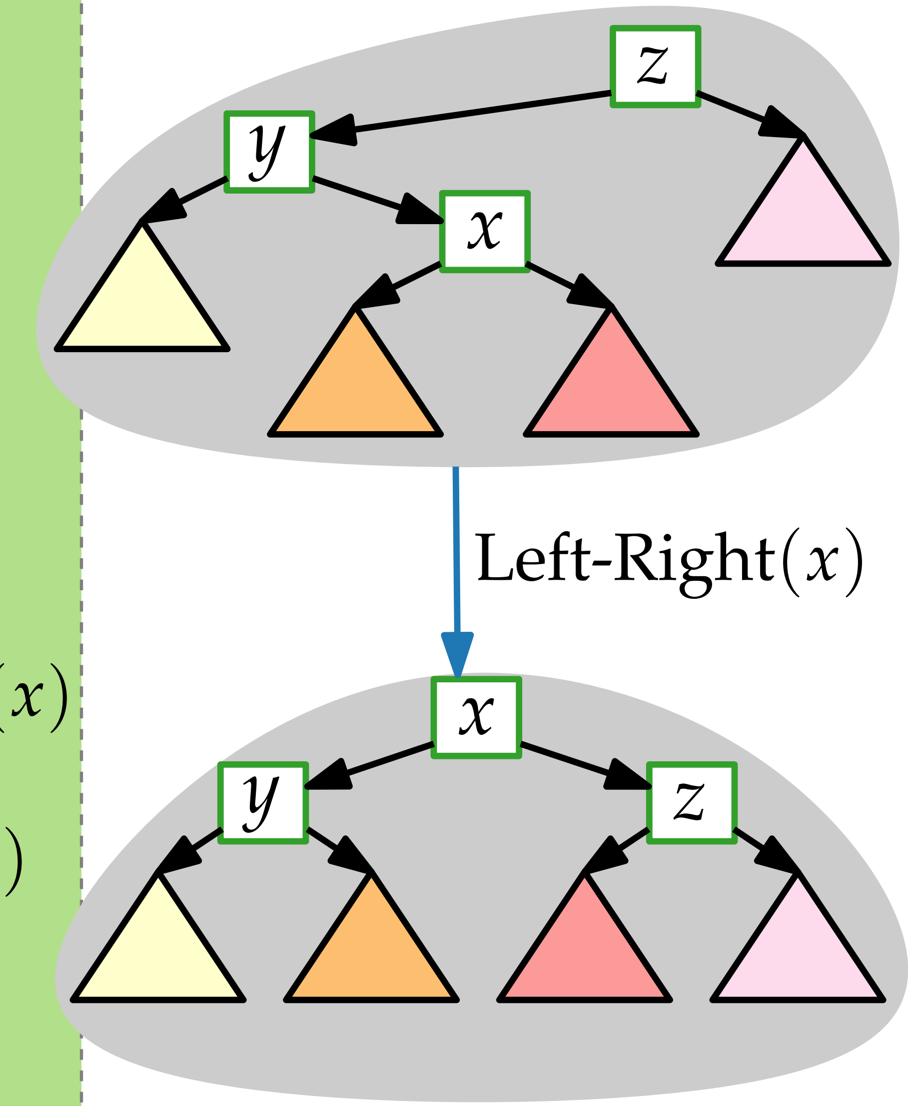
if $x \neq \mathit{root}$ **then**
  y = parent of x
  if $y = \mathit{root}$ **then**
    if $x < y$ **then** Right(x)
    if $y < x$ **then** Left(x)
  else
    z = parent of y
    if $x < y < z$ **then** Right-Right(x)
    if $z < y < x$ **then** Left-Left(x)
    if $y < x < z$ **then**



Splay

Algorithm: Splay(x)

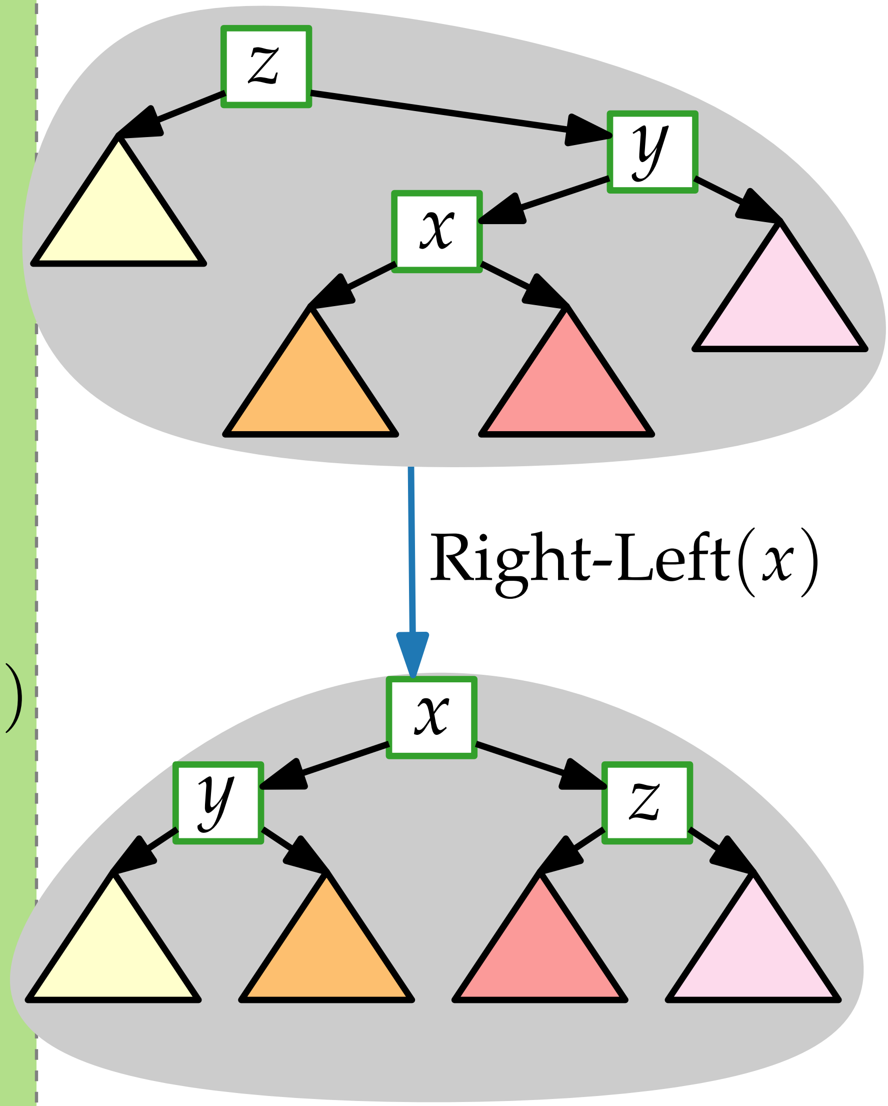
if $x \neq \mathit{root}$ **then**
  $y =$ parent of x
  if $y = \mathit{root}$ **then**
    if $x < y$ **then** Right(x)
    if $y < x$ **then** Left(x)
  else
    $z =$ parent of y
    if $x < y < z$ **then** Right-Right(x)
    if $z < y < x$ **then** Left-Left(x)
    if $y < x < z$ **then** Left-Right(x)



Splay

Algorithm: Splay(x)

if $x \neq root$ **then**
- y = parent of x
- **if** $y = root$ **then**
  - **if** $x < y$ **then** Right(x)
  - **if** $y < x$ **then** Left(x)
- **else**
  - z = parent of y
  - **if** $x < y < z$ **then** Right-Right(x)
  - **if** $z < y < x$ **then** Left-Left(x)
  - **if** $y < x < z$ **then** Left-Right(x)
  - **if** $z < x < y$ **then** Right-Left(x)



Splay

Algorithm: Splay(x)

if $x \neq \mathit{root}$ **then**
- $y =$ parent of x
- **if** $y = \mathit{root}$ **then**
  - **if** $x < y$ **then** Right(x)
  - **if** $y < x$ **then** Left(x)
- **else**
  - $z =$ parent of y
  - **if** $x < y < z$ **then** Right-Right(x)
  - **if** $z < y < x$ **then** Left-Left(x)
  - **if** $y < x < z$ **then** Left-Right(x)
  - **if** $z < x < y$ **then** Right-Left(x)
- Splay(x)

Splay

Algorithm: Splay(x)

if $x \neq root$ **then**

    $y =$ parent of x

    if $y = root$ **then**

        if $x < y$ **then** Right(x)

        if $y < x$ **then** Left(x)

    else

        $z =$ parent of y

        if $x < y < z$ **then** Right-Right(x)

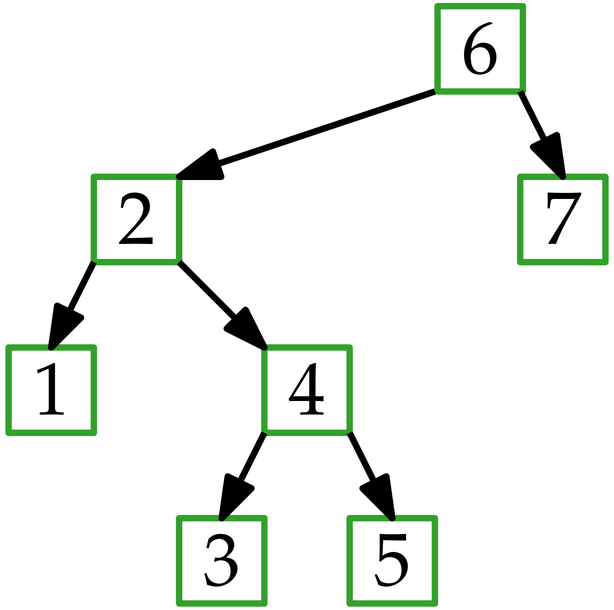
        if $z < y < x$ **then** Left-Left(x)

        if $y < x < z$ **then** Left-Right(x)

        if $z < x < y$ **then** Right-Left(x)

    Splay(x)

Splay(3):

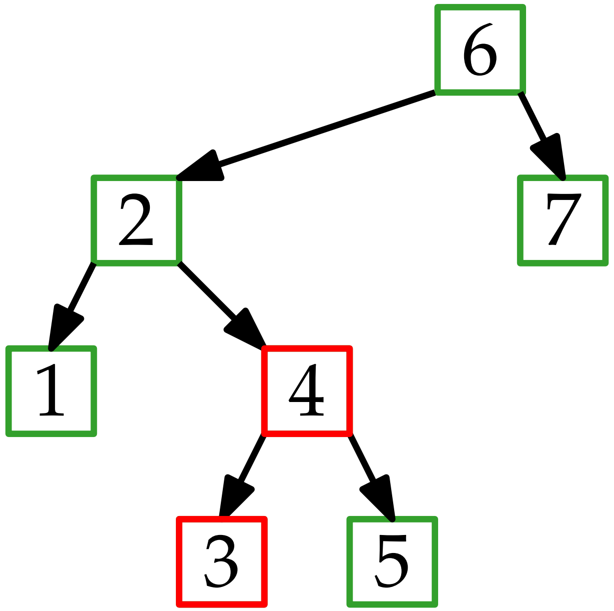


Splay

Algorithm: Splay(x)

if $x \neq root$ **then**
- y = parent of x
- **if** $y = root$ **then**
  - **if** $x < y$ **then** Right(x)
  - **if** $y < x$ **then** Left(x)
- **else**
  - z = parent of y
  - **if** $x < y < z$ **then** Right-Right(x)
  - **if** $z < y < x$ **then** Left-Left(x)
  - **if** $y < x < z$ **then** Left-Right(x)
  - **if** $z < x < y$ **then** Right-Left(x)
- Splay(x)

Splay(3):

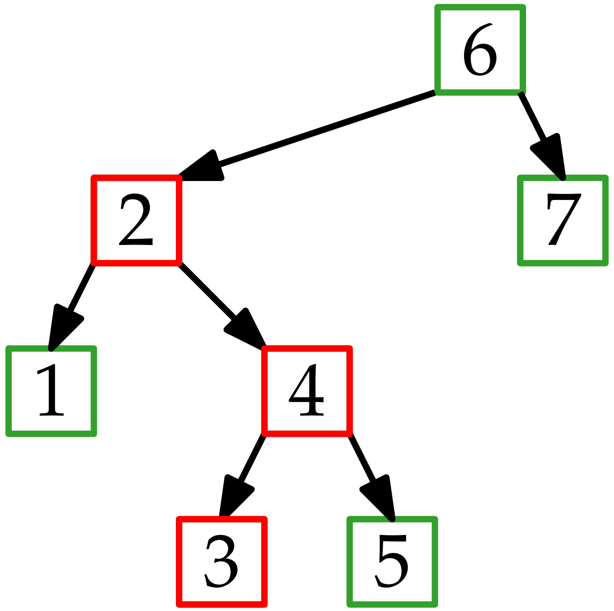


Splay

Algorithm: Splay(x)

if $x \neq root$ **then**
- y = parent of x
- **if** $y = root$ **then**
  - **if** $x < y$ **then** Right(x)
  - **if** $y < x$ **then** Left(x)
- **else**
  - z = parent of y
  - **if** $x < y < z$ **then** Right-Right(x)
  - **if** $z < y < x$ **then** Left-Left(x)
  - **if** $y < x < z$ **then** Left-Right(x)
  - **if** $z < x < y$ **then** Right-Left(x)
- Splay(x)

Splay(3):

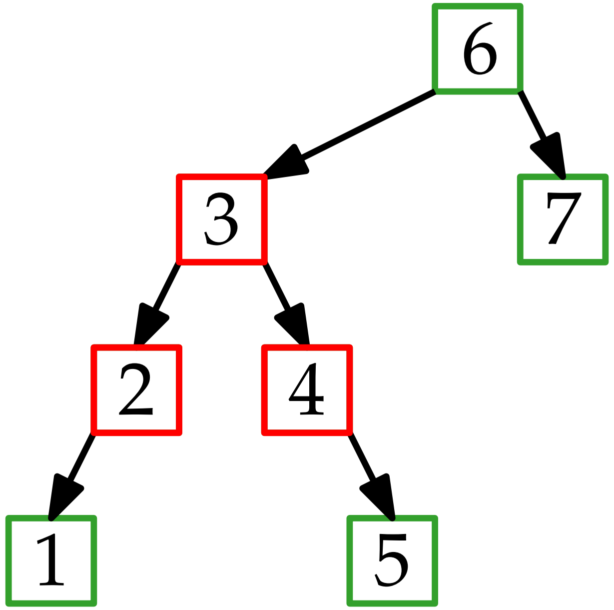


Splay

Algorithm: Splay(x)

if $x \neq \textit{root}$ **then**
- $y =$ parent of x
- **if** $y = \textit{root}$ **then**
  - **if** $x < y$ **then** Right(x)
  - **if** $y < x$ **then** Left(x)
- **else**
  - $z =$ parent of y
  - **if** $x < y < z$ **then** Right-Right(x)
  - **if** $z < y < x$ **then** Left-Left(x)
  - **if** $y < x < z$ **then** Left-Right(x)
  - **if** $z < x < y$ **then** Right-Left(x)
- Splay(x)

Splay(3):

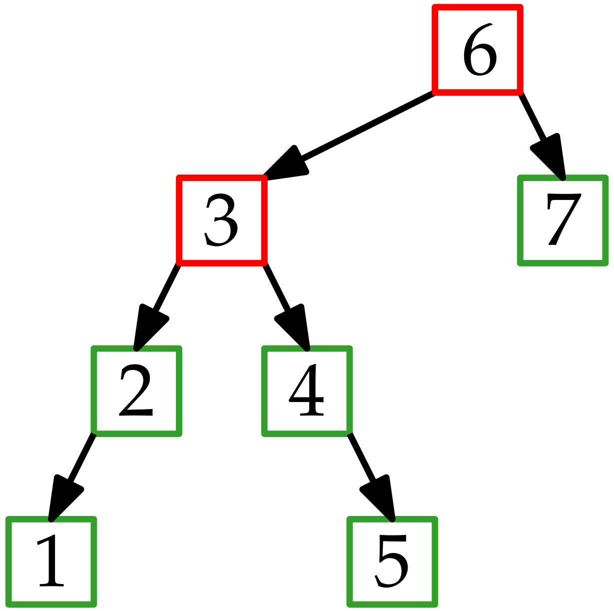


Splay

Algorithm: Splay(x)

if $x \neq root$ **then**
- $y =$ parent of x
- **if** $y = root$ **then**
  - **if** $x < y$ **then** Right(x)
  - **if** $y < x$ **then** Left(x)
- **else**
  - $z =$ parent of y
  - **if** $x < y < z$ **then** Right-Right(x)
  - **if** $z < y < x$ **then** Left-Left(x)
  - **if** $y < x < z$ **then** Left-Right(x)
  - **if** $z < x < y$ **then** Right-Left(x)
- Splay(x)

Splay(3):

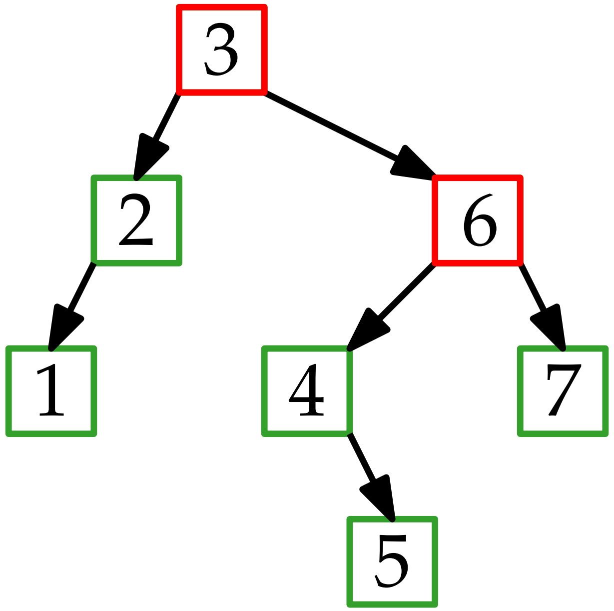


Splay

Algorithm: Splay(x)

if $x \neq root$ **then**
- y = parent of x
- **if** $y = root$ **then**
  - **if** $x < y$ **then** Right(x)
  - **if** $y < x$ **then** Left(x)
- **else**
  - z = parent of y
  - **if** $x < y < z$ **then** Right-Right(x)
  - **if** $z < y < x$ **then** Left-Left(x)
  - **if** $y < x < z$ **then** Left-Right(x)
  - **if** $z < x < y$ **then** Right-Left(x)
- Splay(x)

Splay(3):



Splay

Algorithm: Splay(x)

if $x \neq root$ **then**

  $y =$ parent of x

  if $y = root$ **then**

    if $x < y$ **then** Right(x)

    if $y < x$ **then** Left(x)

  else

    $z =$ parent of y

    if $x < y < z$ **then** Right-Right(x)

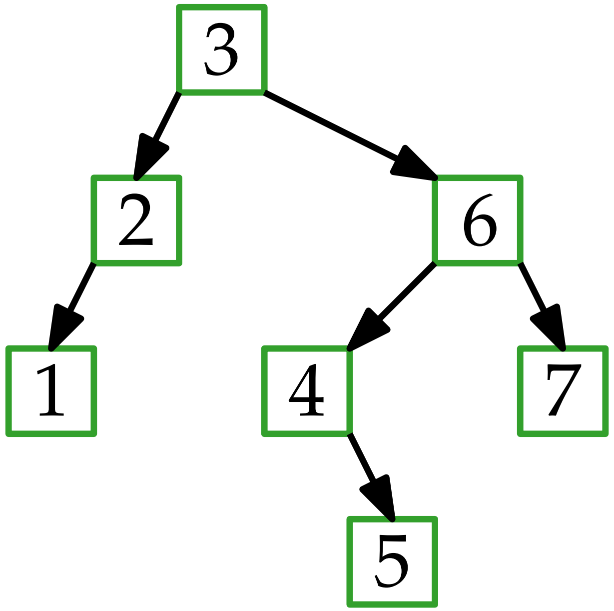
    if $z < y < x$ **then** Left-Left(x)

    if $y < x < z$ **then** Left-Right(x)

    if $z < x < y$ **then** Right-Left(x)

  Splay(x)

Splay(3):



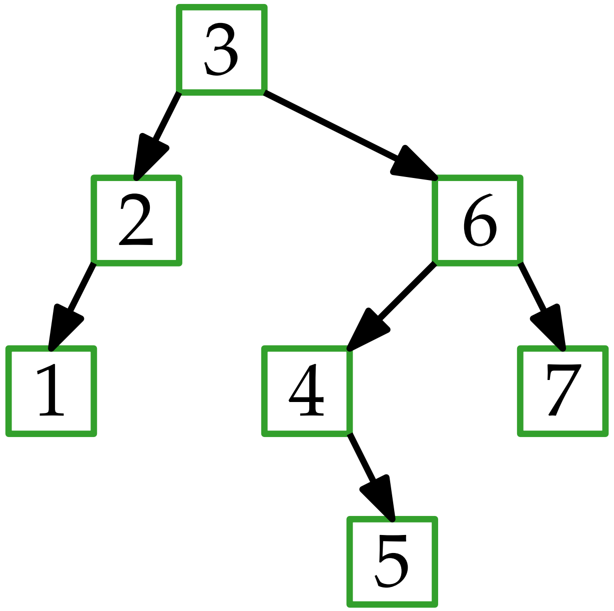
Call Splay(x):

Splay

Algorithm: Splay(x)

if $x \neq root$ **then**
- y = parent of x
- **if** $y = root$ **then**
  - **if** $x < y$ **then** Right(x)
  - **if** $y < x$ **then** Left(x)
- **else**
  - z = parent of y
  - **if** $x < y < z$ **then** Right-Right(x)
  - **if** $z < y < x$ **then** Left-Left(x)
  - **if** $y < x < z$ **then** Left-Right(x)
  - **if** $z < x < y$ **then** Right-Left(x)
- Splay(x)

Splay(3):



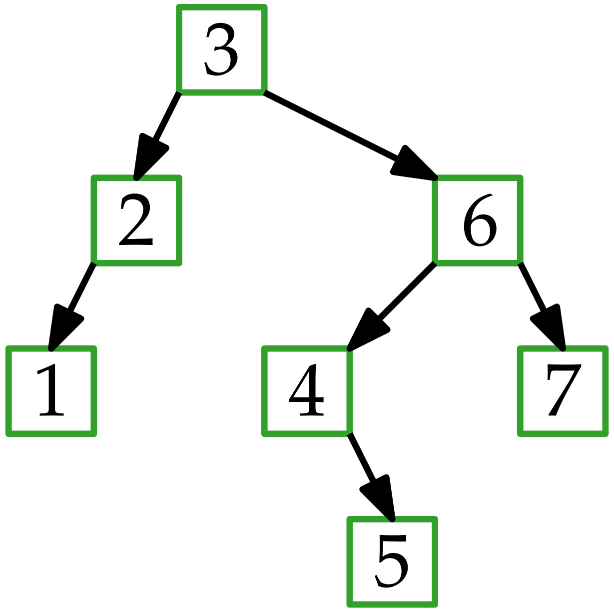
Call Splay(x):
- after Search(x)

Splay

Algorithm: Splay(x)

if $x \neq root$ **then**
- $y =$ parent of x
- **if** $y = root$ **then**
  - **if** $x < y$ **then** Right(x)
  - **if** $y < x$ **then** Left(x)
- **else**
  - $z =$ parent of y
  - **if** $x < y < z$ **then** Right-Right(x)
  - **if** $z < y < x$ **then** Left-Left(x)
  - **if** $y < x < z$ **then** Left-Right(x)
  - **if** $z < x < y$ **then** Right-Left(x)
- Splay(x)

Splay(3):



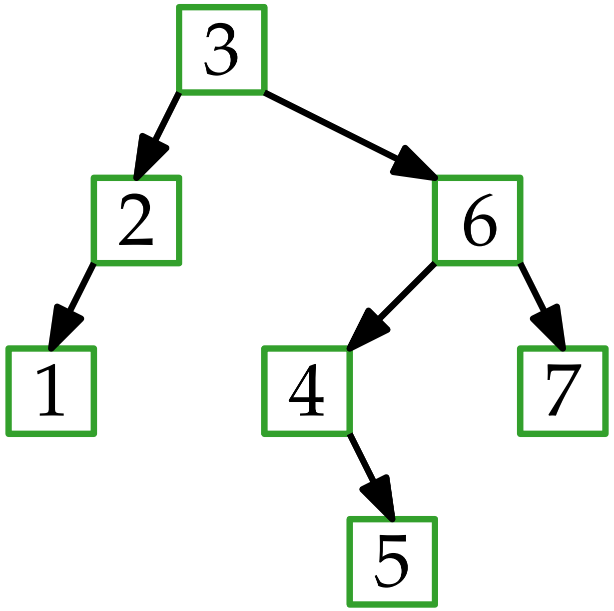
Call Splay(x):
- after Search(x)
- after Insert(x)

Splay

Algorithm: Splay(x)

if $x \neq root$ **then**
  $y =$ parent of x
  if $y = root$ **then**
    if $x < y$ **then** Right(x)
    if $y < x$ **then** Left(x)
  else
    $z =$ parent of y
    if $x < y < z$ **then** Right-Right(x)
    if $z < y < x$ **then** Left-Left(x)
    if $y < x < z$ **then** Left-Right(x)
    if $z < x < y$ **then** Right-Left(x)
  Splay(x)

Splay(3):



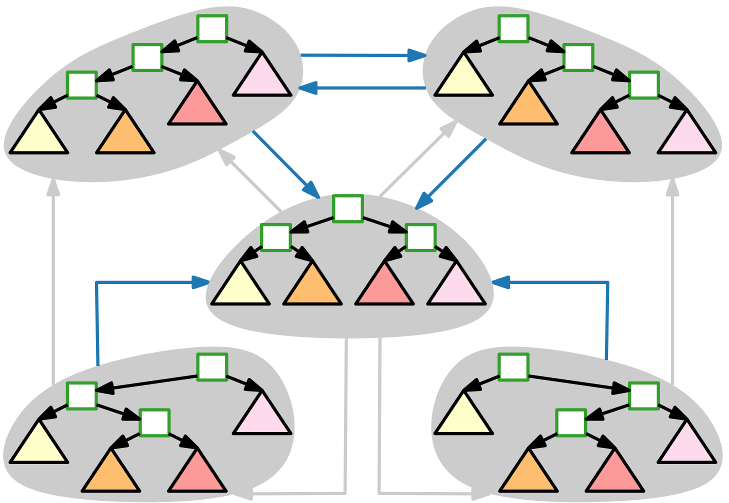
Call Splay(x):

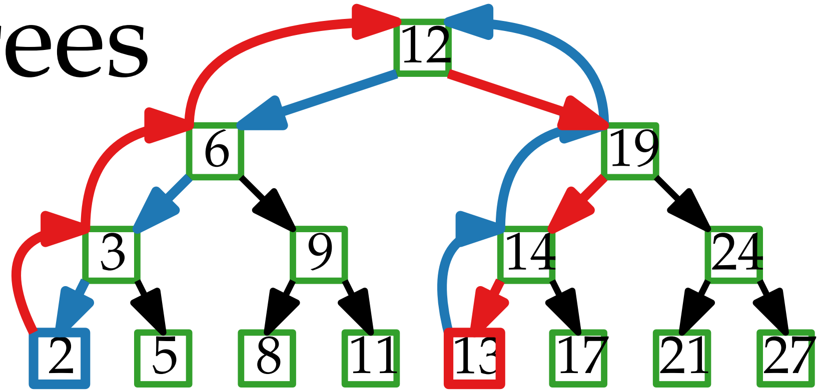
- after Search(x)
- after Insert(x)
- before Delete(x)

Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Advanced Algorithms

Optimal Binary Search Trees



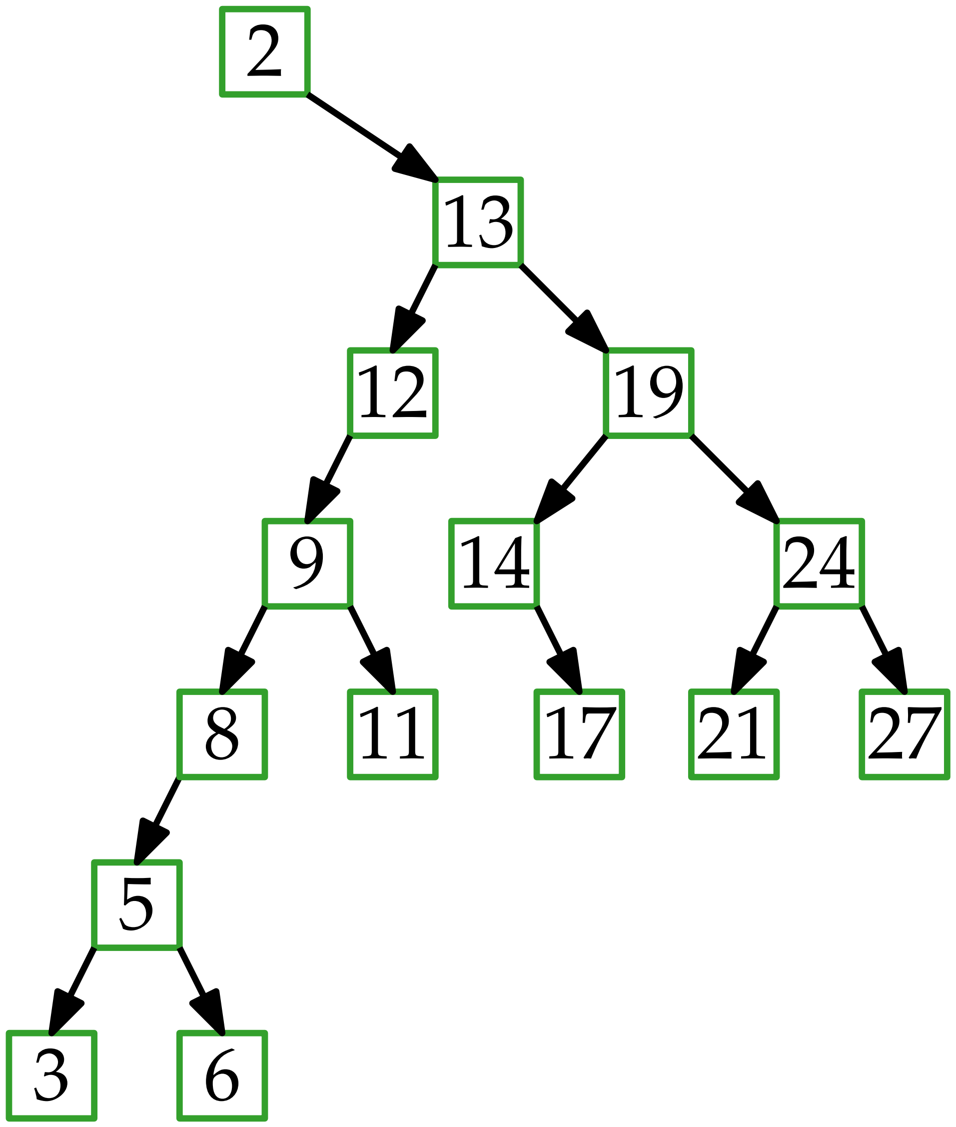


Splay Trees

Philipp Kindermann · WS20

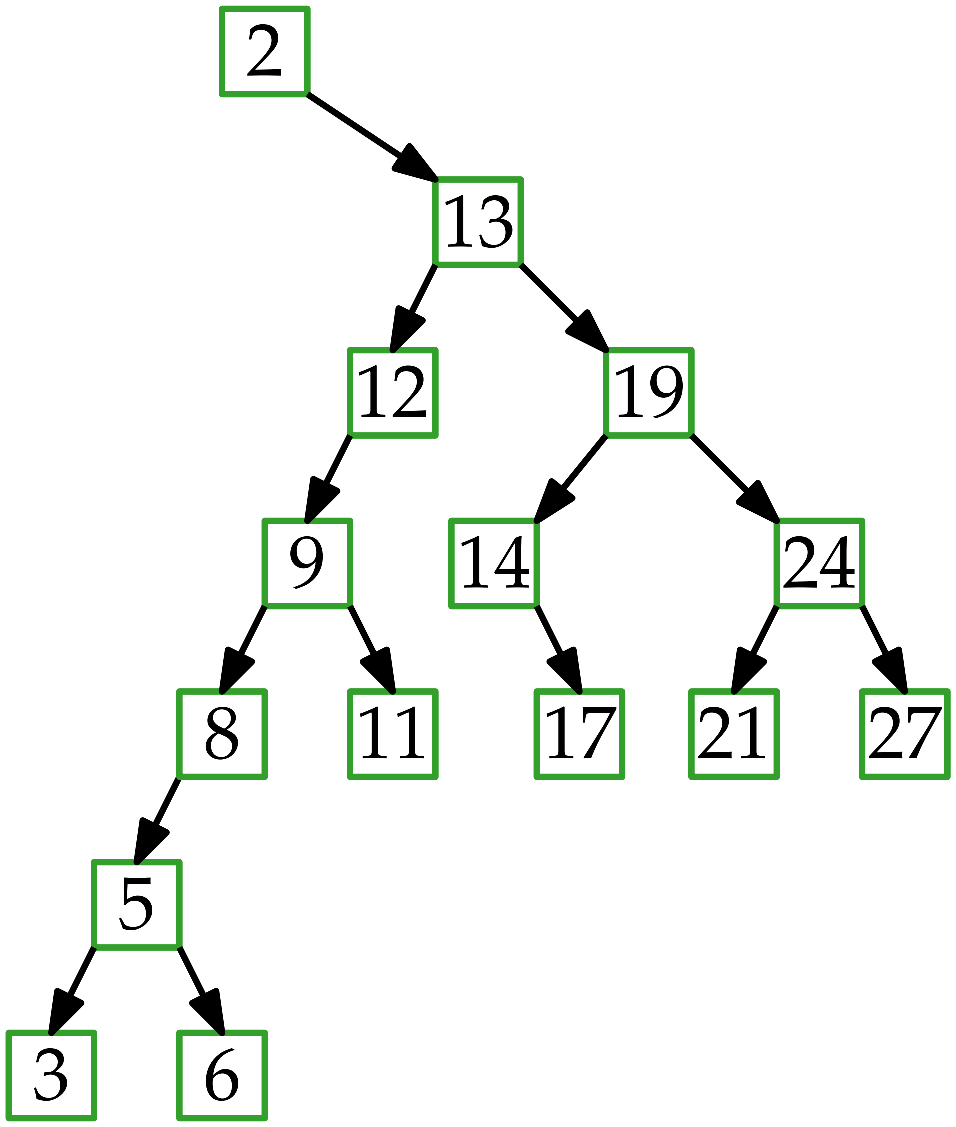
Part IV:
Potential

Why is Splay Fast?



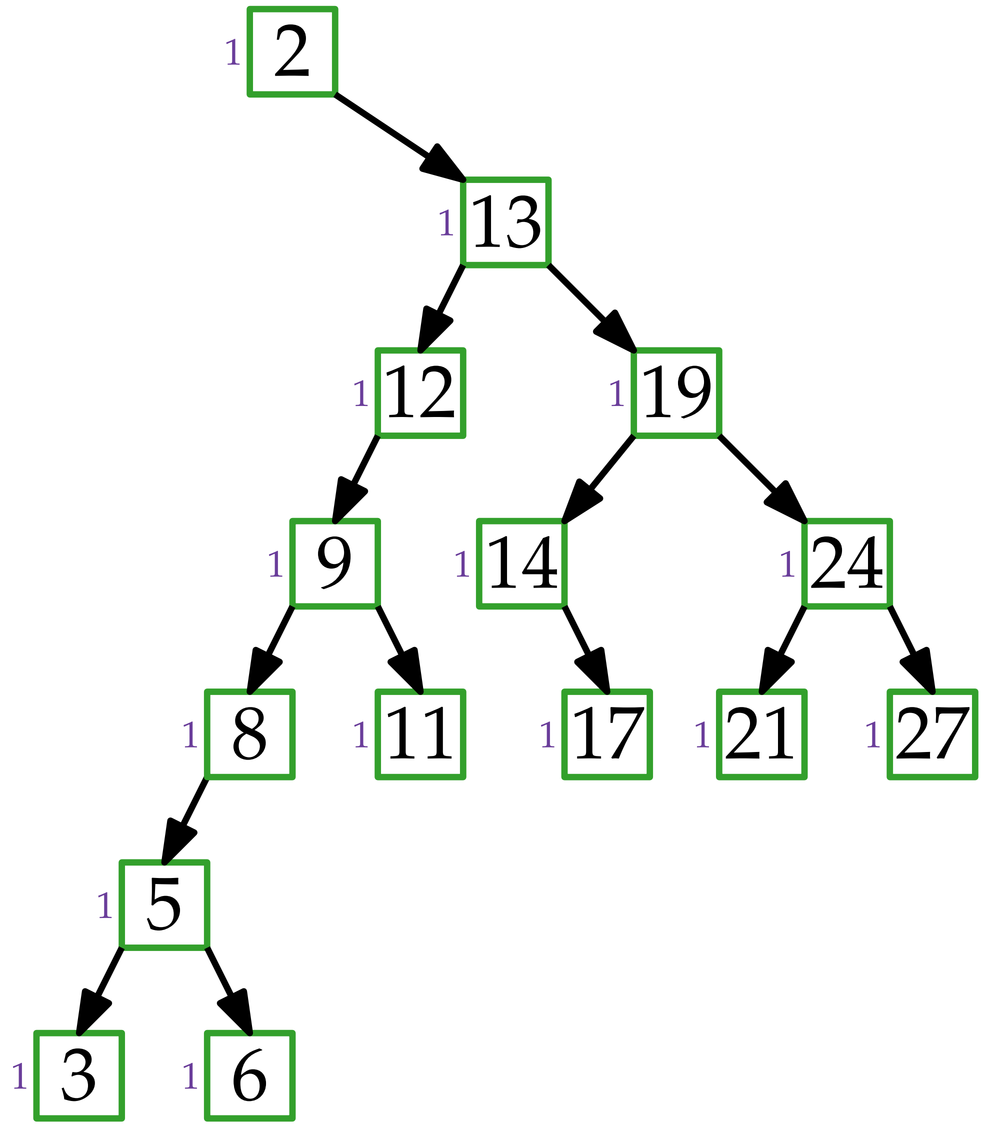
Why is Splay Fast?

$w(x)$: weight of x (here 1), $W = \sum w(x)$ (here n)



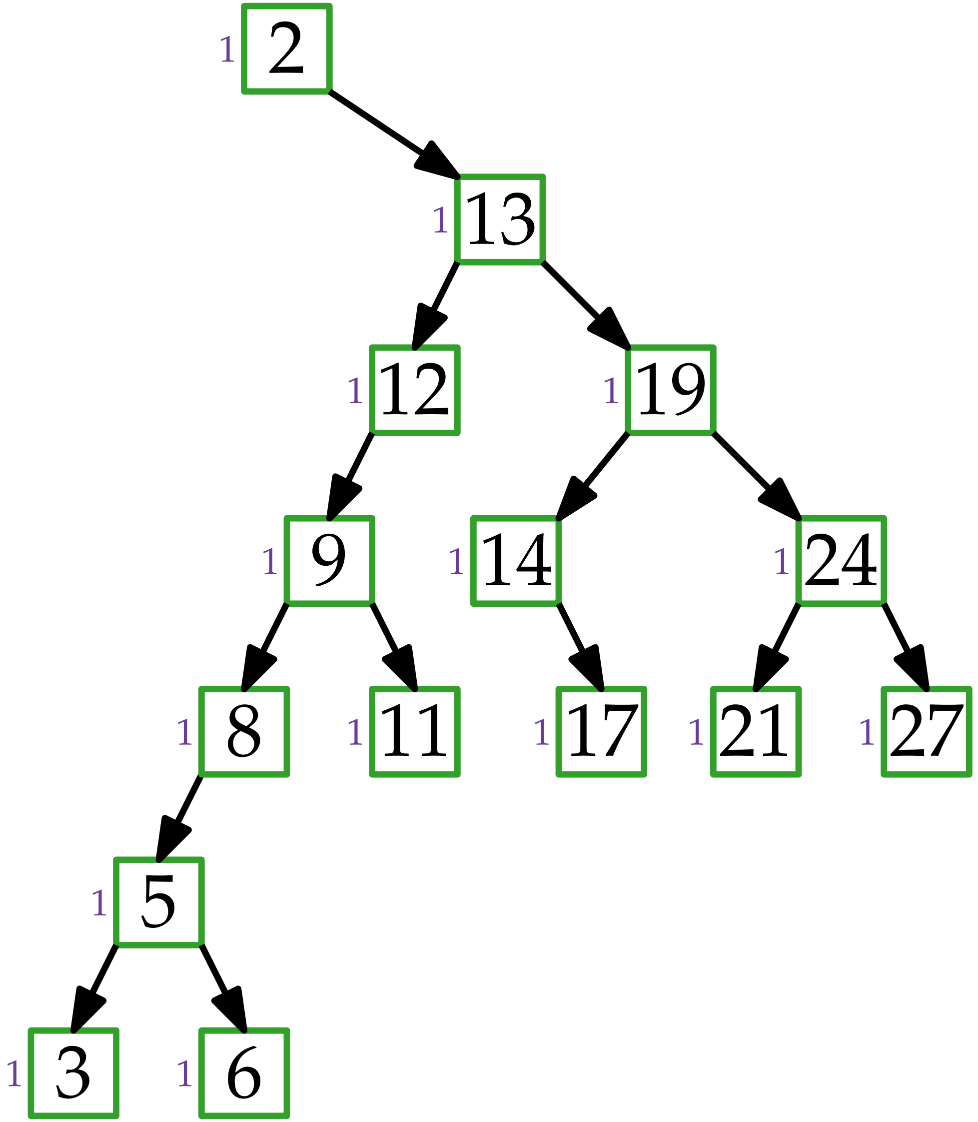
Why is Splay Fast?

$w(x)$: weight of x (here 1), $W = \sum w(x)$ (here n)



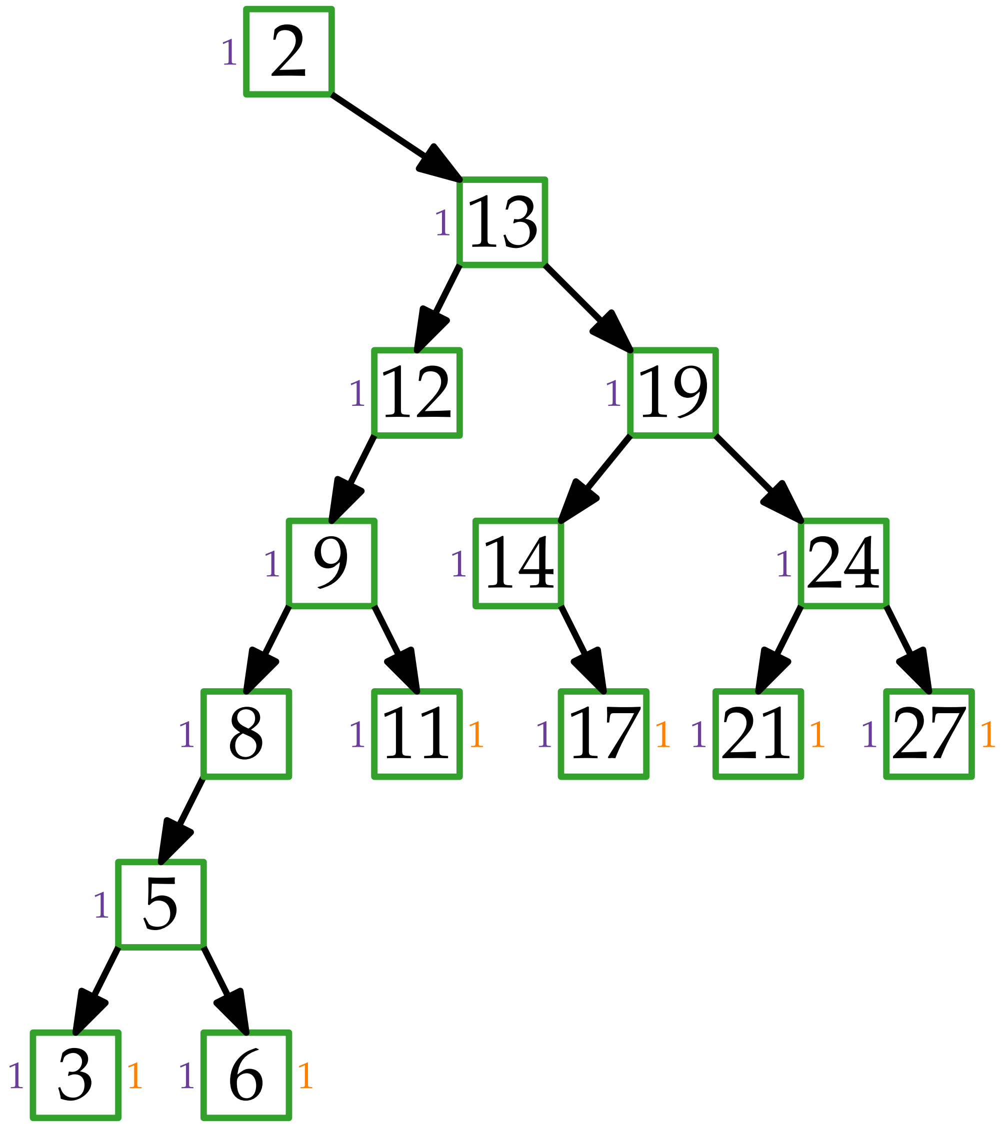
Why is Splay Fast?

$w(x)$: weight of x (here 1), $W = \sum w(x)$ (here n)
$s(x)$: sum of all $w(x)$ in subtree of x_i



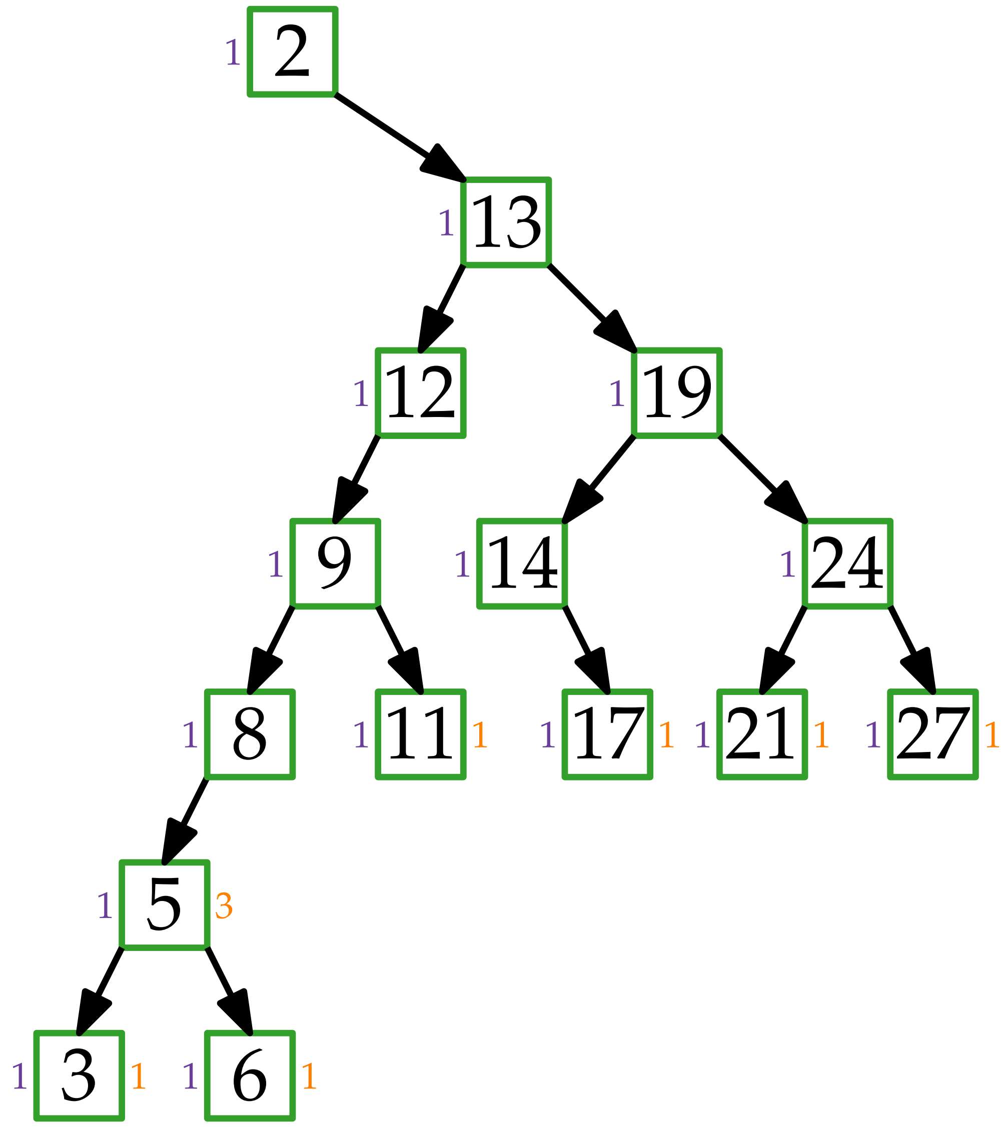
Why is Splay Fast?

$w(x)$: weight of x (here 1), $W = \sum w(x)$ (here n)
$s(x)$: sum of all $w(x)$ in subtree of x_i



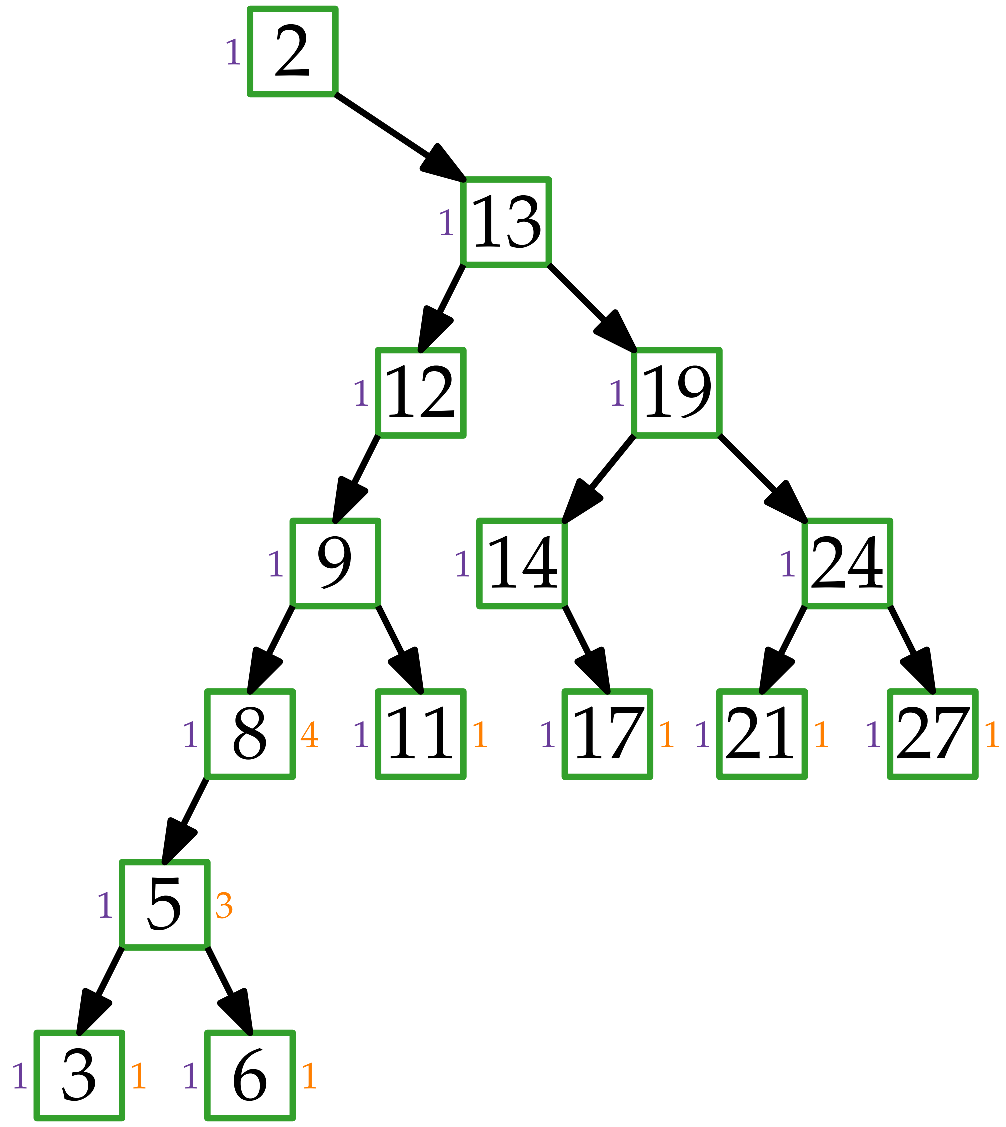
Why is Splay Fast?

$w(x)$: weight of x (here 1), $W = \sum w(x)$ (here n)
$s(x)$: sum of all $w(x)$ in subtree of x_i



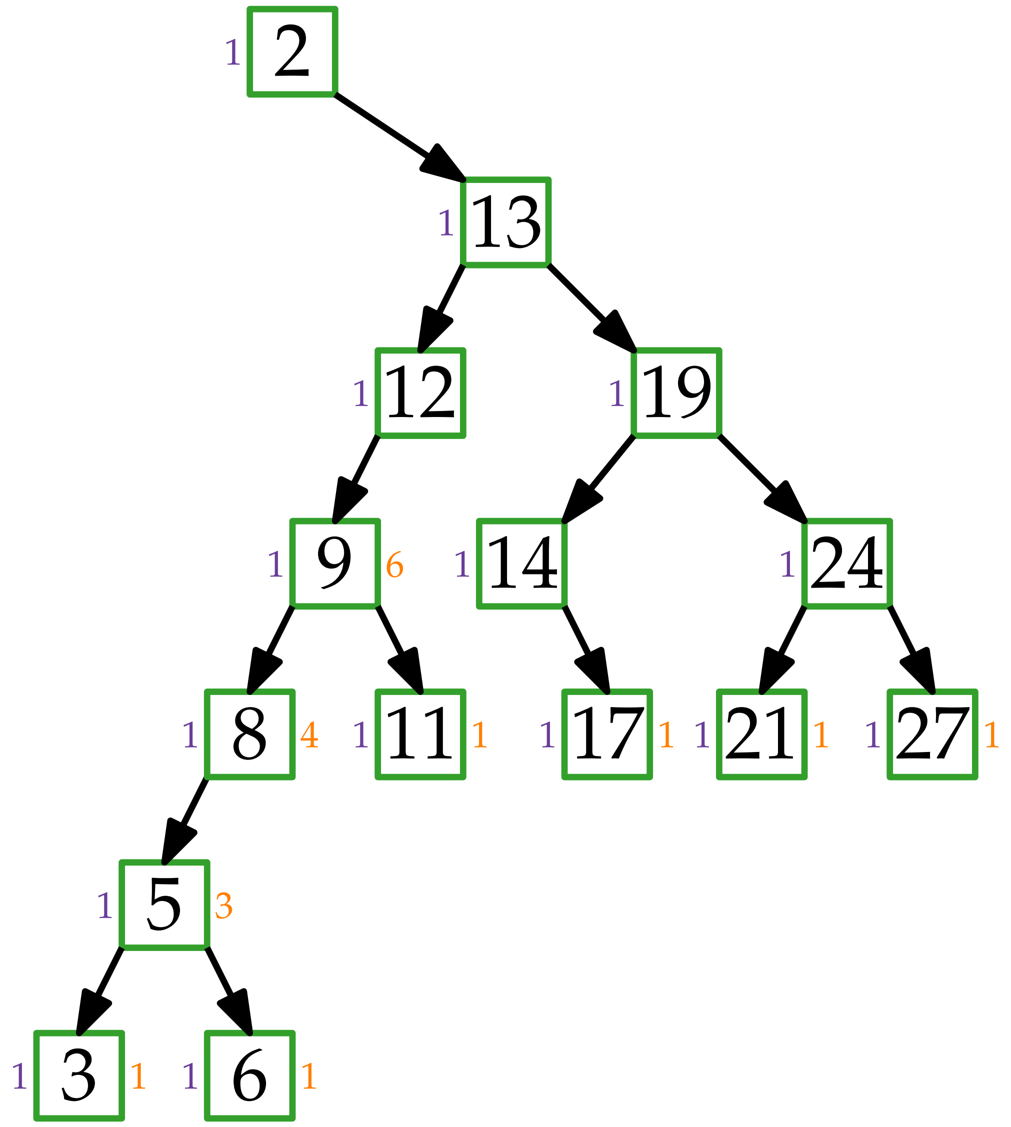
Why is Splay Fast?

$w(x)$: weight of x (here 1), $W = \sum w(x)$ (here n)
$s(x)$: sum of all $w(x)$ in subtree of x_i



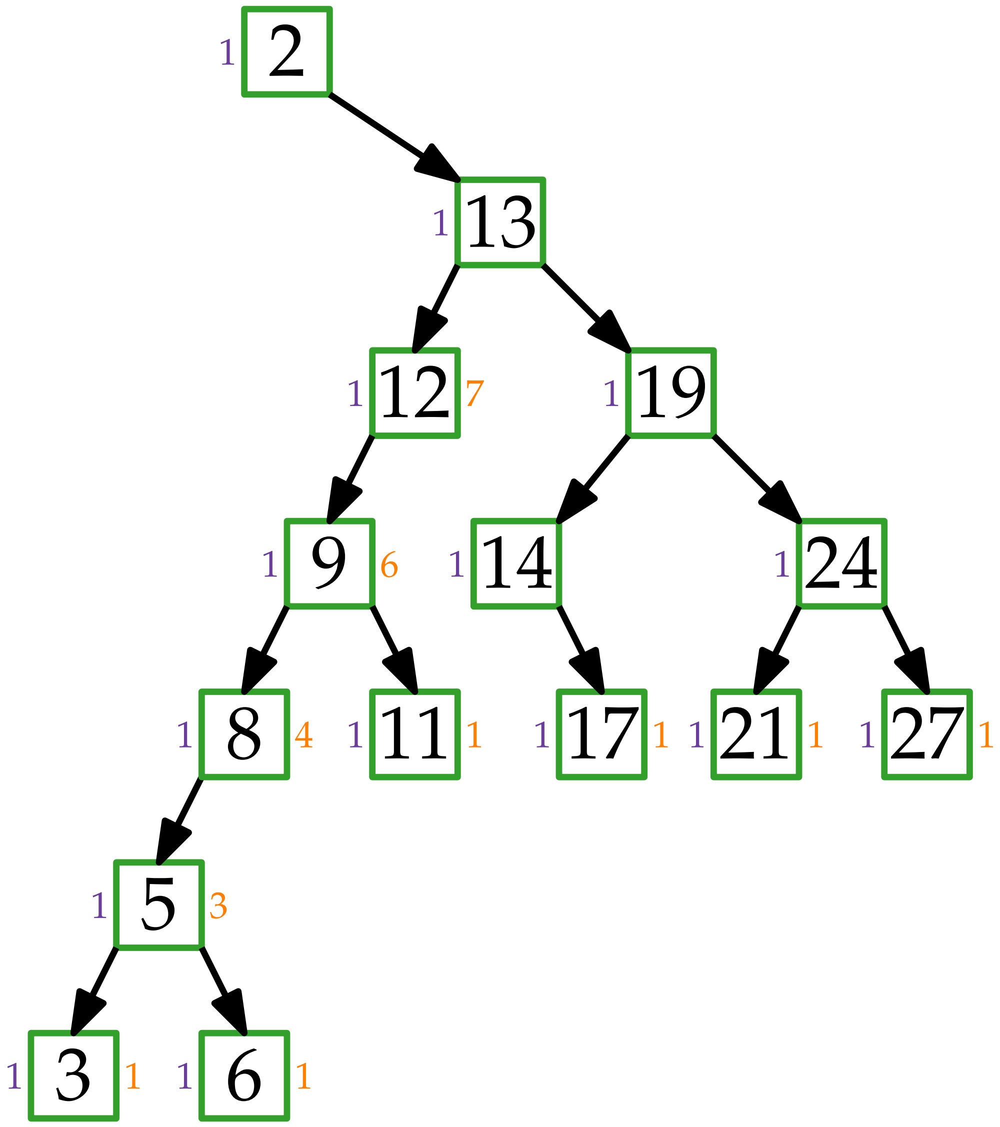
Why is Splay Fast?

$w(x)$: weight of x (here 1), $W = \sum w(x)$ (here n)
$s(x)$: sum of all $w(x)$ in subtree of x_i



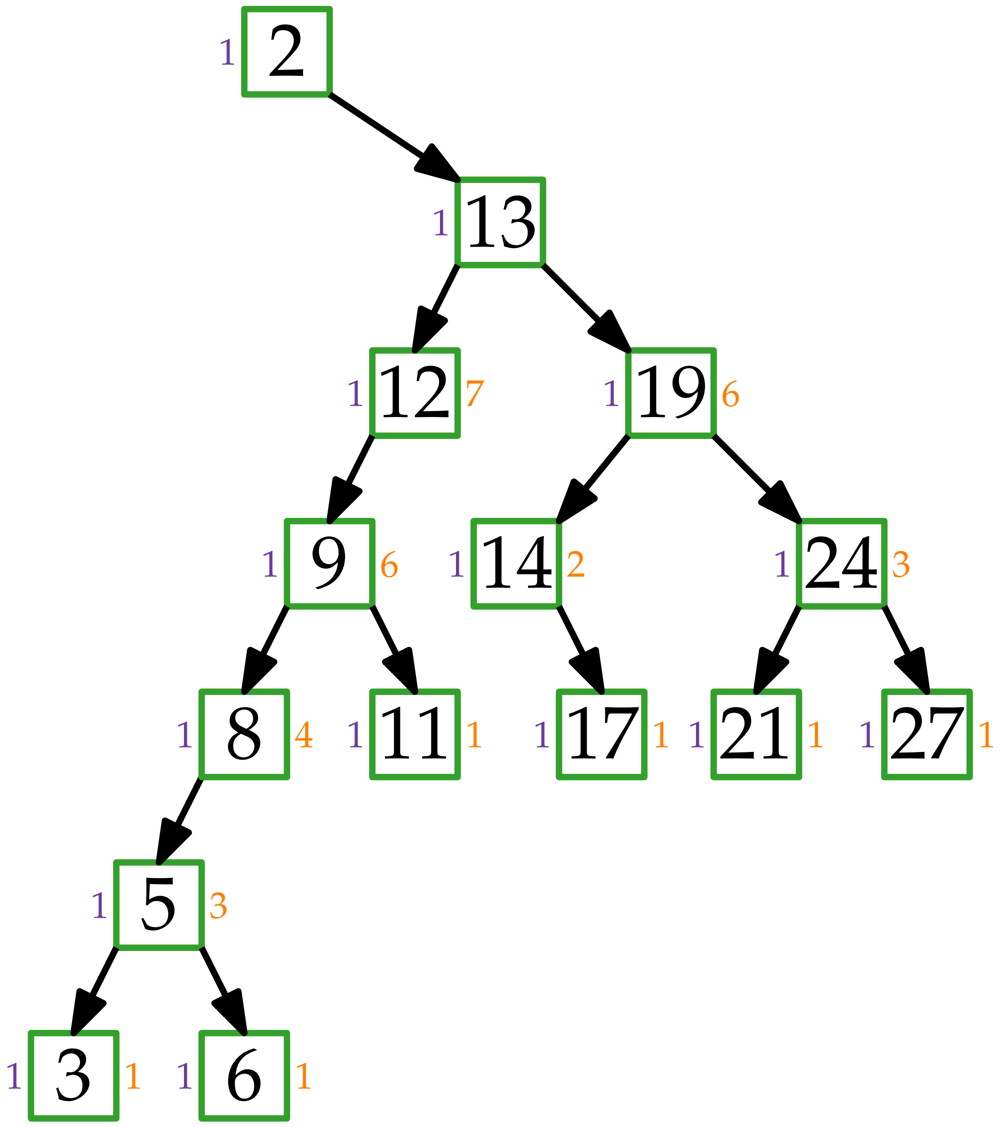
Why is Splay Fast?

$w(x)$: weight of x (here 1), $W = \sum w(x)$ (here n)
$s(x)$: sum of all $w(x)$ in subtree of x_i



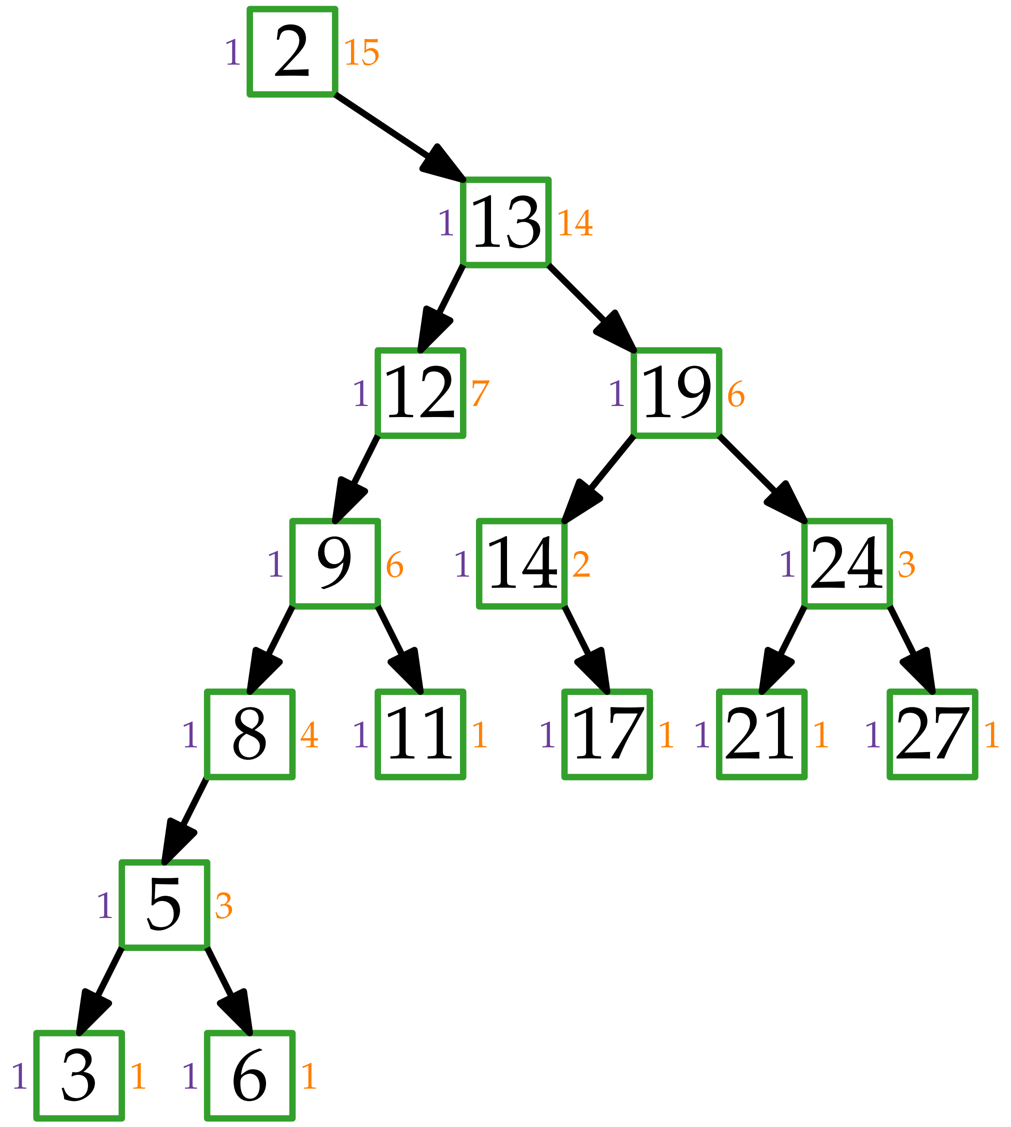
Why is Splay Fast?

$w(x)$: weight of x (here 1), $W = \sum w(x)$ (here n)
$s(x)$: sum of all $w(x)$ in subtree of x_i



Why is Splay Fast?

$w(x)$: weight of x (here 1), $W = \sum w(x)$ (here n)
$s(x)$: sum of all $w(x)$ in subtree of x_i

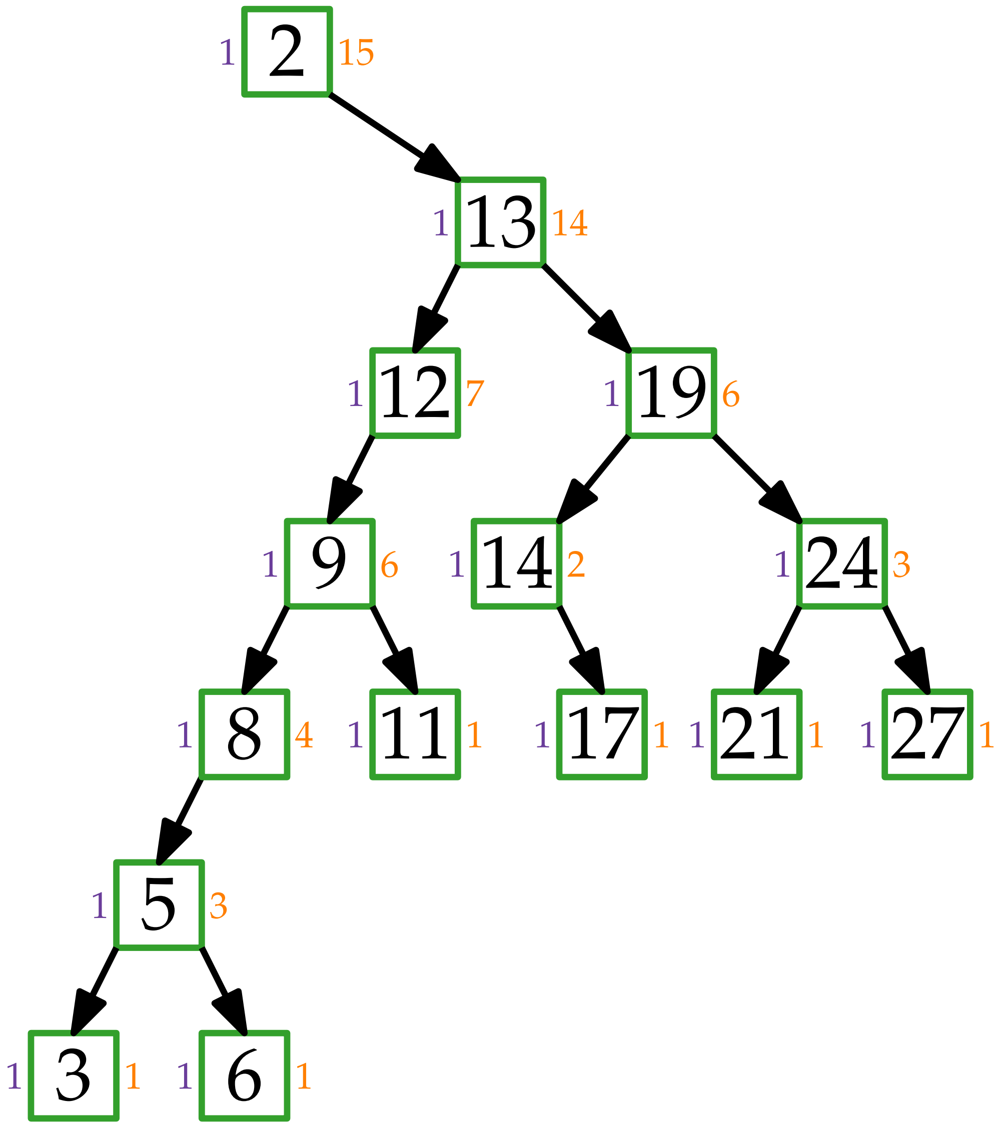


Why is Splay Fast?

$w(x)$: weight of x (here 1), $W = \sum w(x)$ (here n)

$s(x)$: sum of all $w(x)$ in subtree of x_i

mark edges:



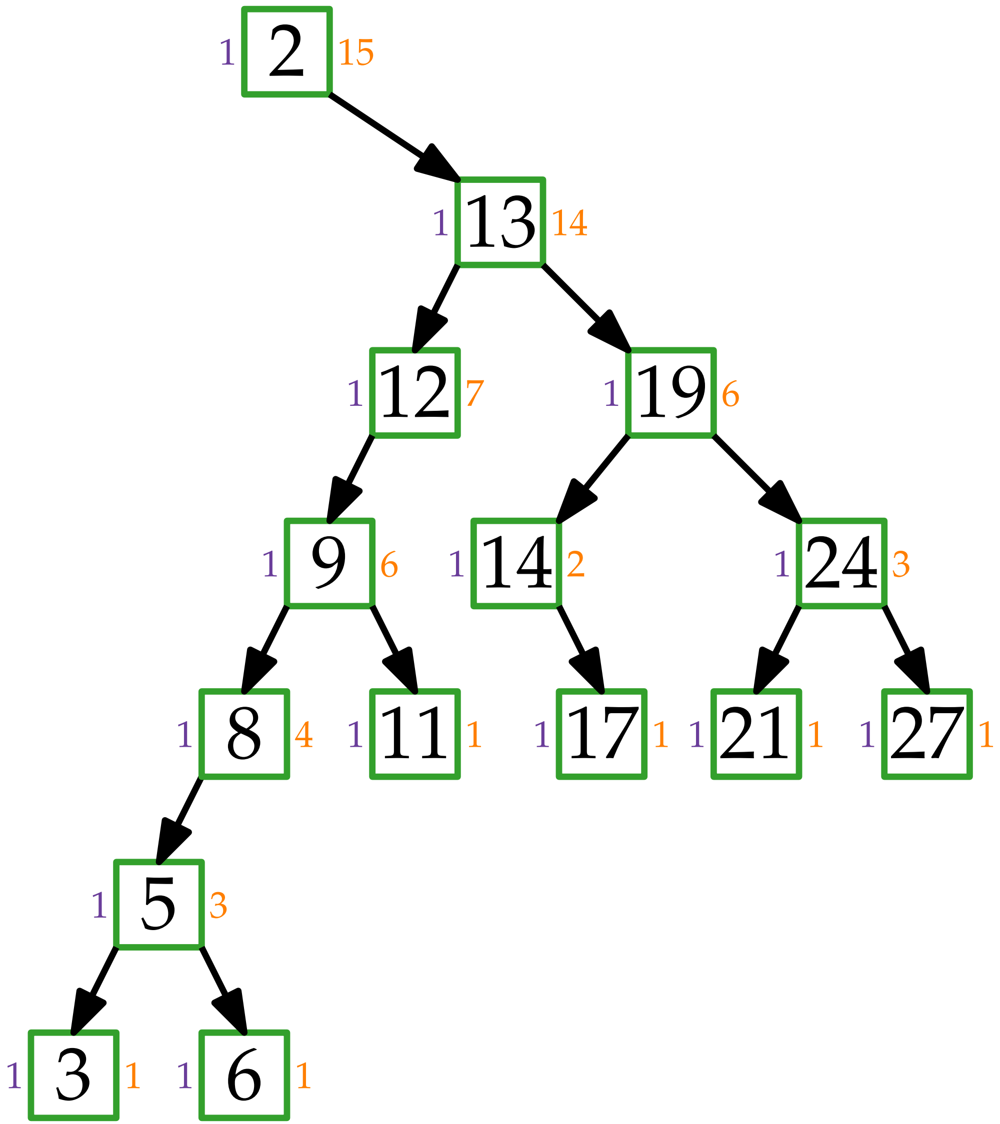
Why is Splay Fast?

$w(x)$: weight of x (here 1), $W = \sum w(x)$ (here n)

$s(x)$: sum of all $w(x)$ in subtree of x_i

mark edges:

$s(\text{child}) \leq s(\text{parent})/2$



Why is Splay Fast?

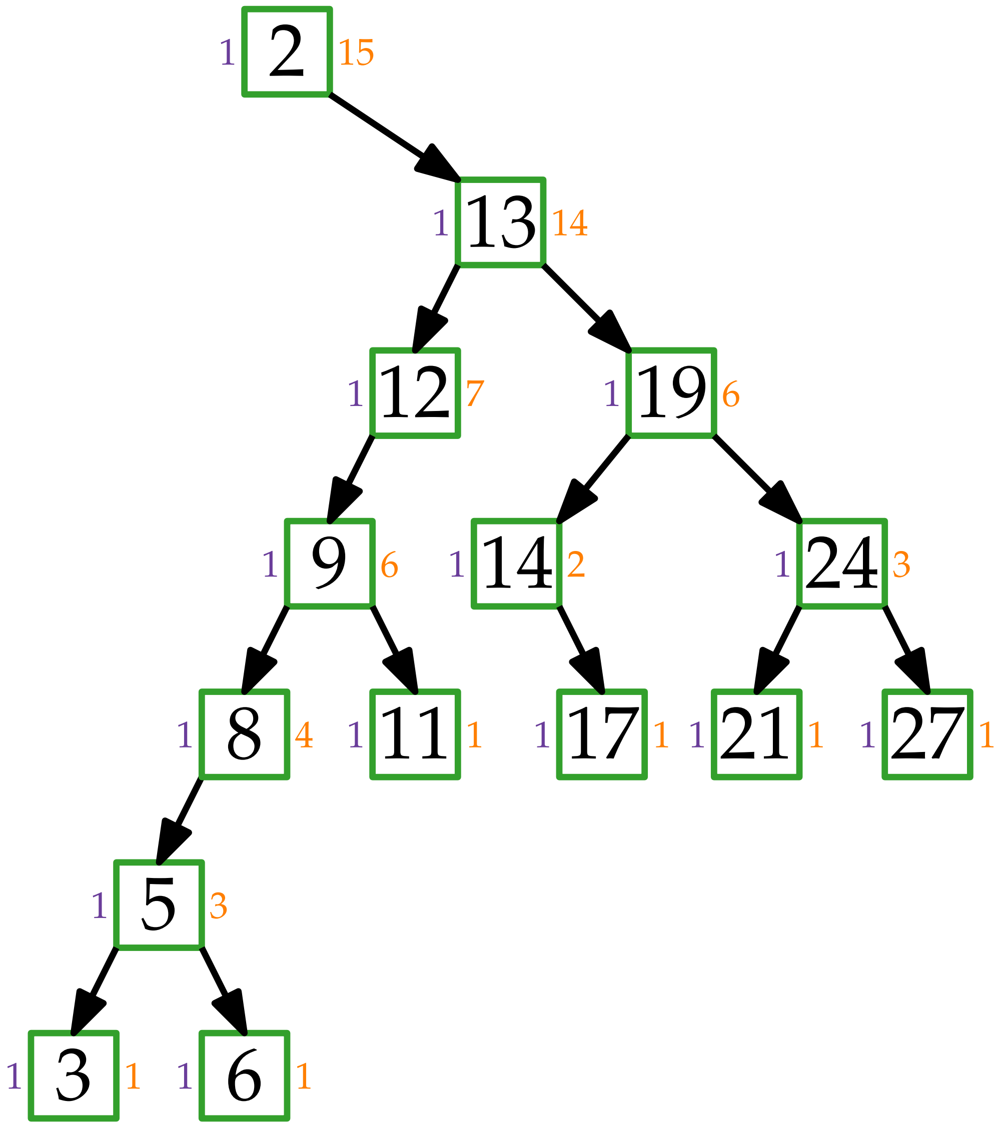
$w(x)$: weight of x (here 1), $W = \sum w(x)$ (here n)

$s(x)$: sum of all $w(x)$ in subtree of x_i

mark edges:

$s(\text{child}) \leq s(\text{parent})/2$

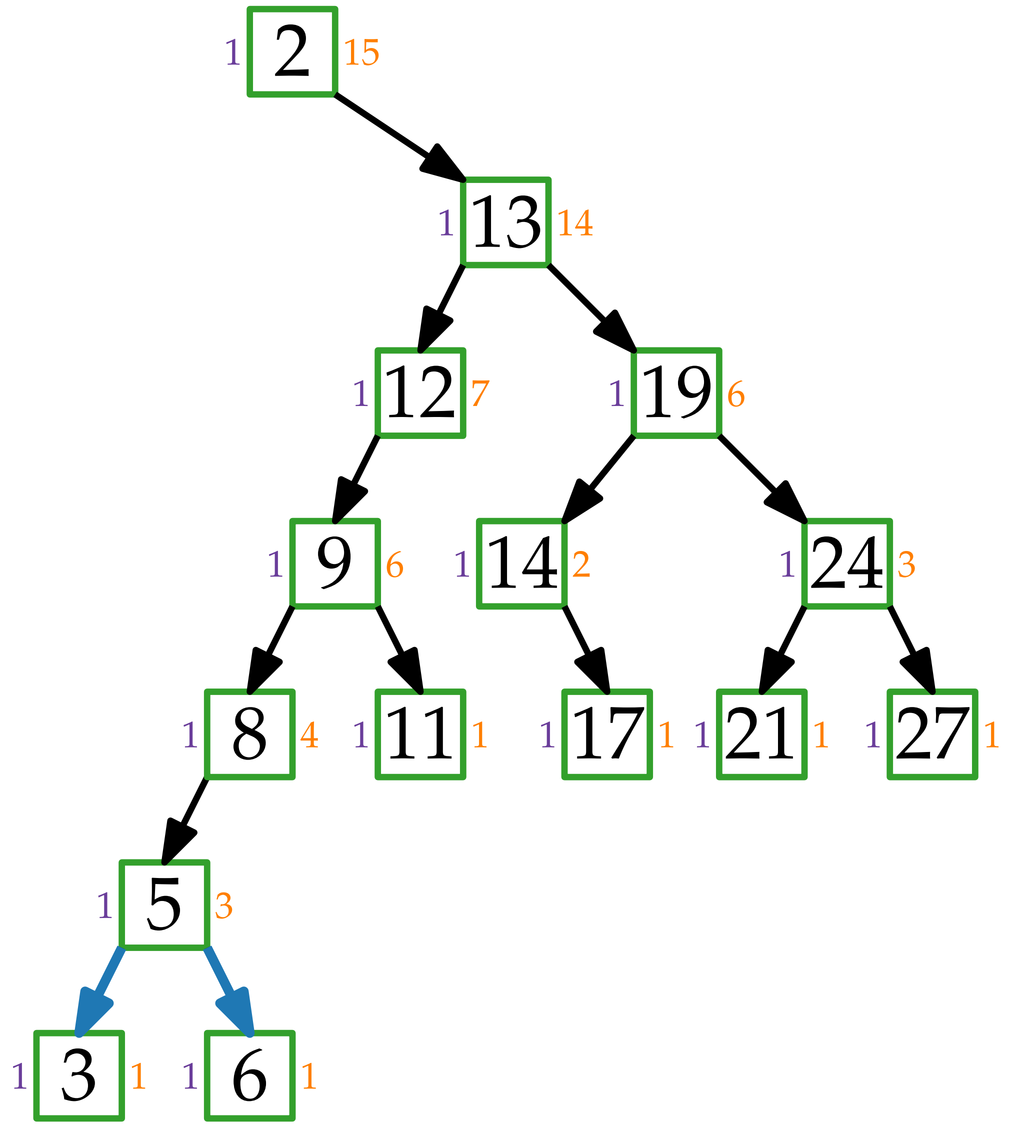
$s(\text{child}) > s(\text{parent})/2$



Why is Splay Fast?

$w(x)$: weight of x (here 1), $W = \sum w(x)$ (here n)
$s(x)$: sum of all $w(x)$ in subtree of x_i

mark edges:
→ $s(\text{child}) \leq s(\text{parent})/2$
→ $s(\text{child}) > s(\text{parent})/2$



Why is Splay Fast?

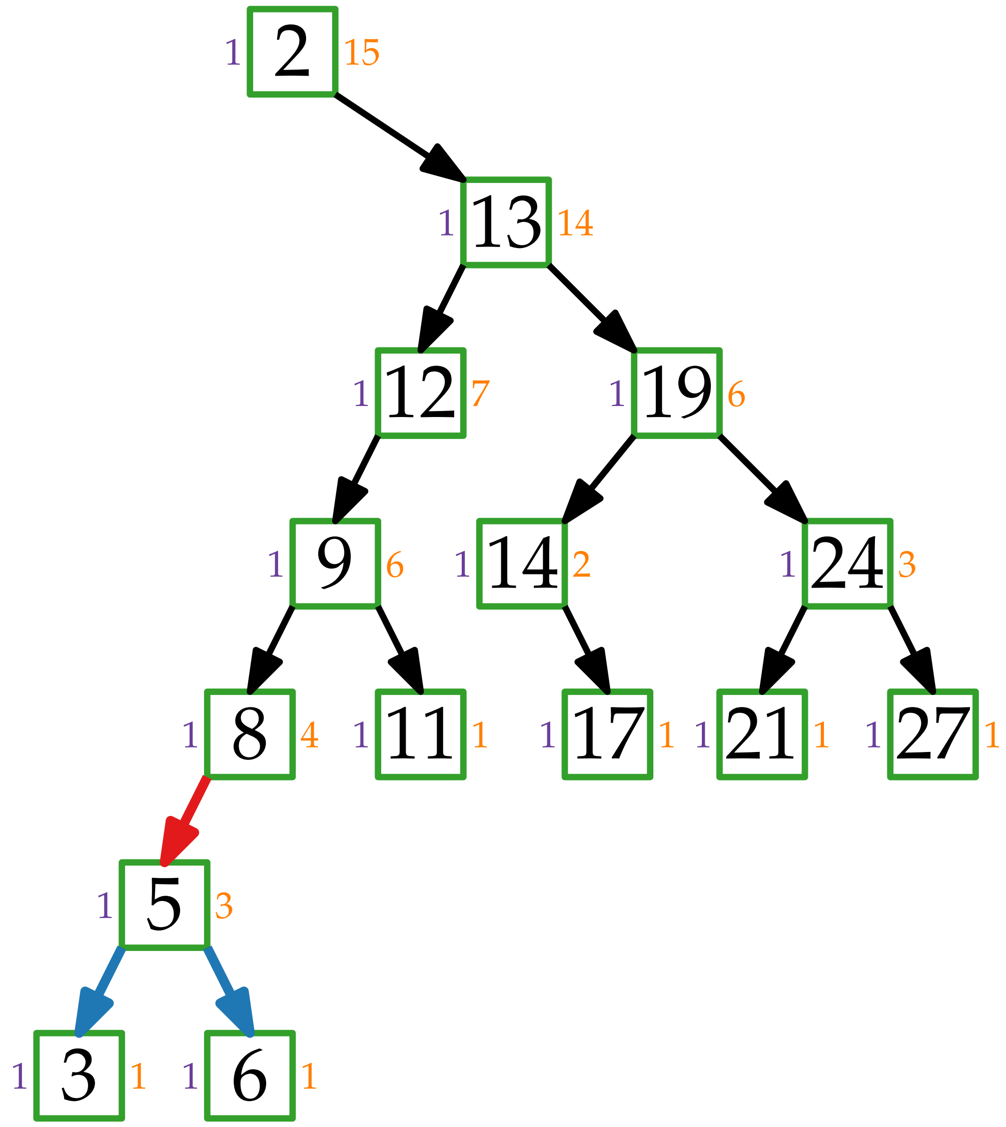
$w(x)$: weight of x (here 1), $W = \sum w(x)$ (here n)

$s(x)$: sum of all $w(x)$ in subtree of x_i

mark edges:

→ $s(\text{child}) \leq s(\text{parent})/2$

→ $s(\text{child}) > s(\text{parent})/2$



Why is Splay Fast?

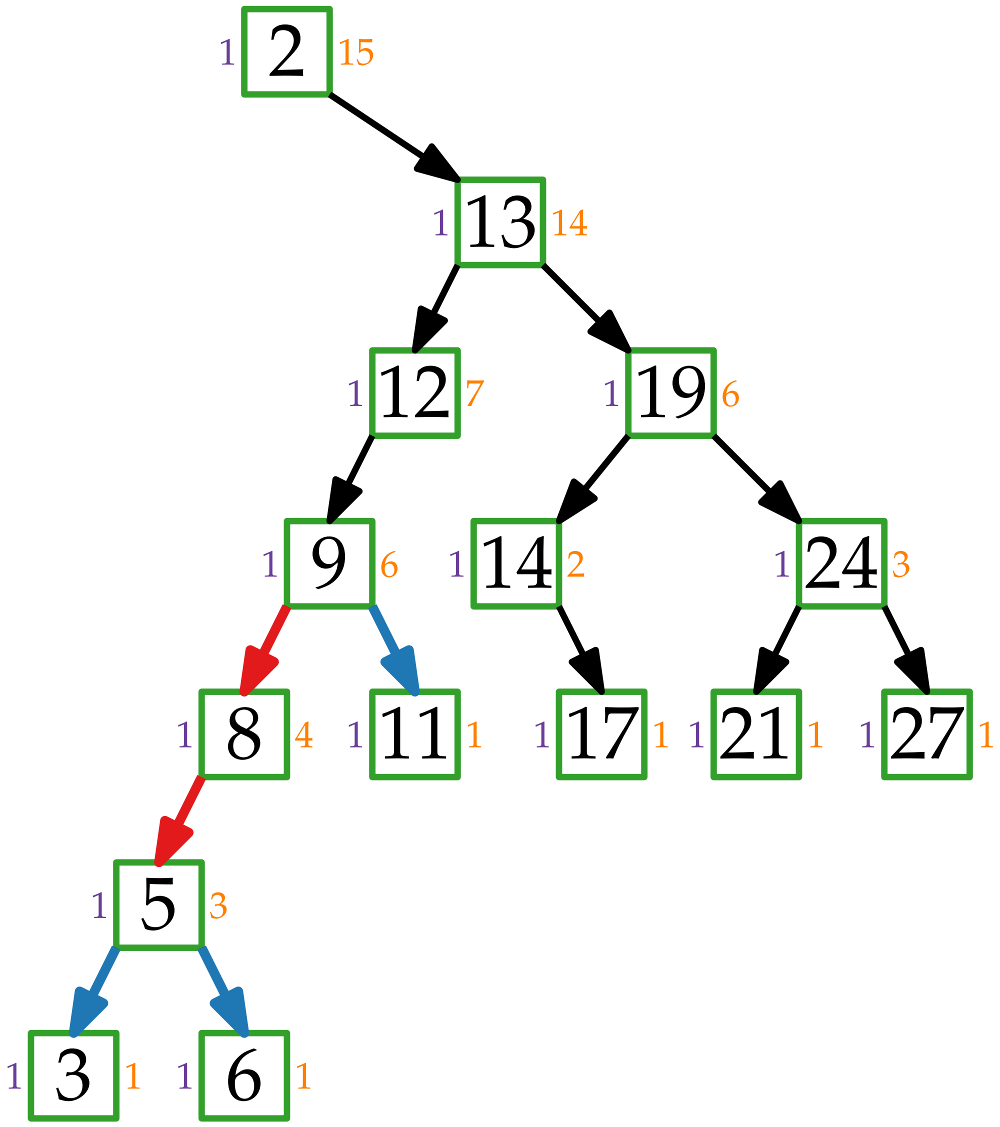
$w(x)$: weight of x (here 1), $W = \sum w(x)$ (here n)

$s(x)$: sum of all $w(x)$ in subtree of x_i

mark edges:

→ $s(\text{child}) \leq s(\text{parent})/2$

→ $s(\text{child}) > s(\text{parent})/2$



Why is Splay Fast?

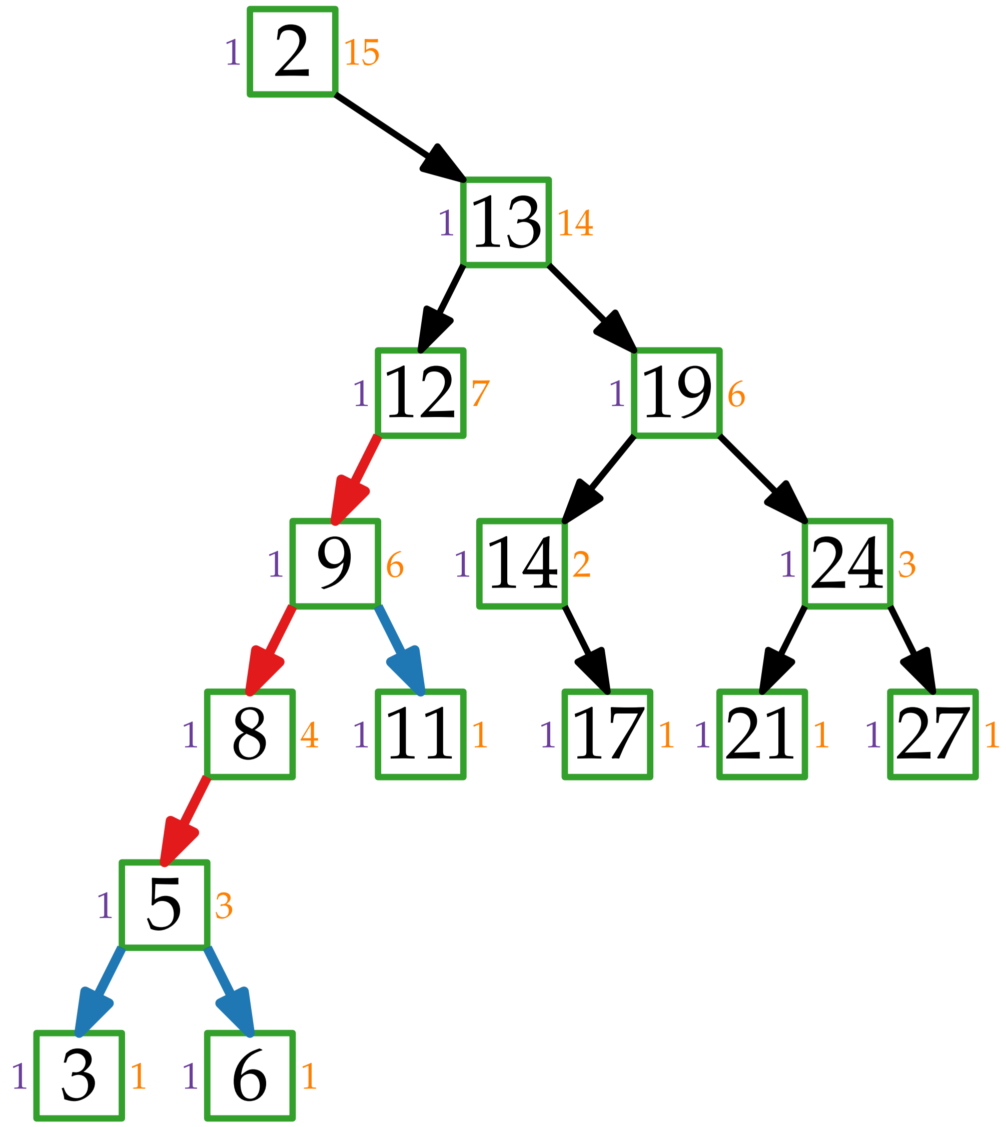
$w(x)$: weight of x (here 1), $W = \sum w(x)$ (here n)

$s(x)$: sum of all $w(x)$ in subtree of x_i

mark edges:

$s(\text{child}) \leq s(\text{parent})/2$

$s(\text{child}) > s(\text{parent})/2$



Why is Splay Fast?

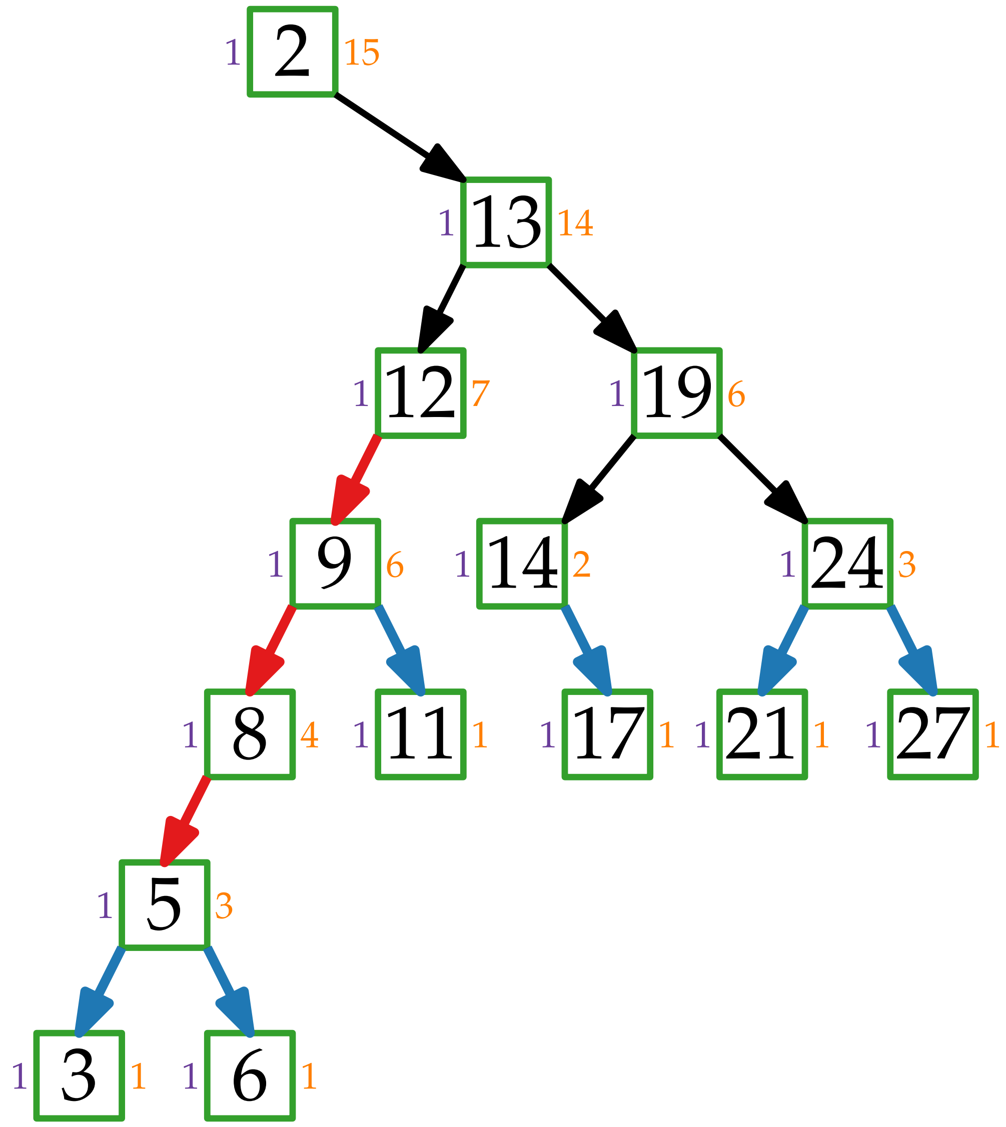
$w(x)$: weight of x (here 1), $W = \sum w(x)$ (here n)

$s(x)$: sum of all $w(x)$ in subtree of x_i

mark edges:

→ $s(\text{child}) \leq s(\text{parent})/2$

→ $s(\text{child}) > s(\text{parent})/2$



Why is Splay Fast?

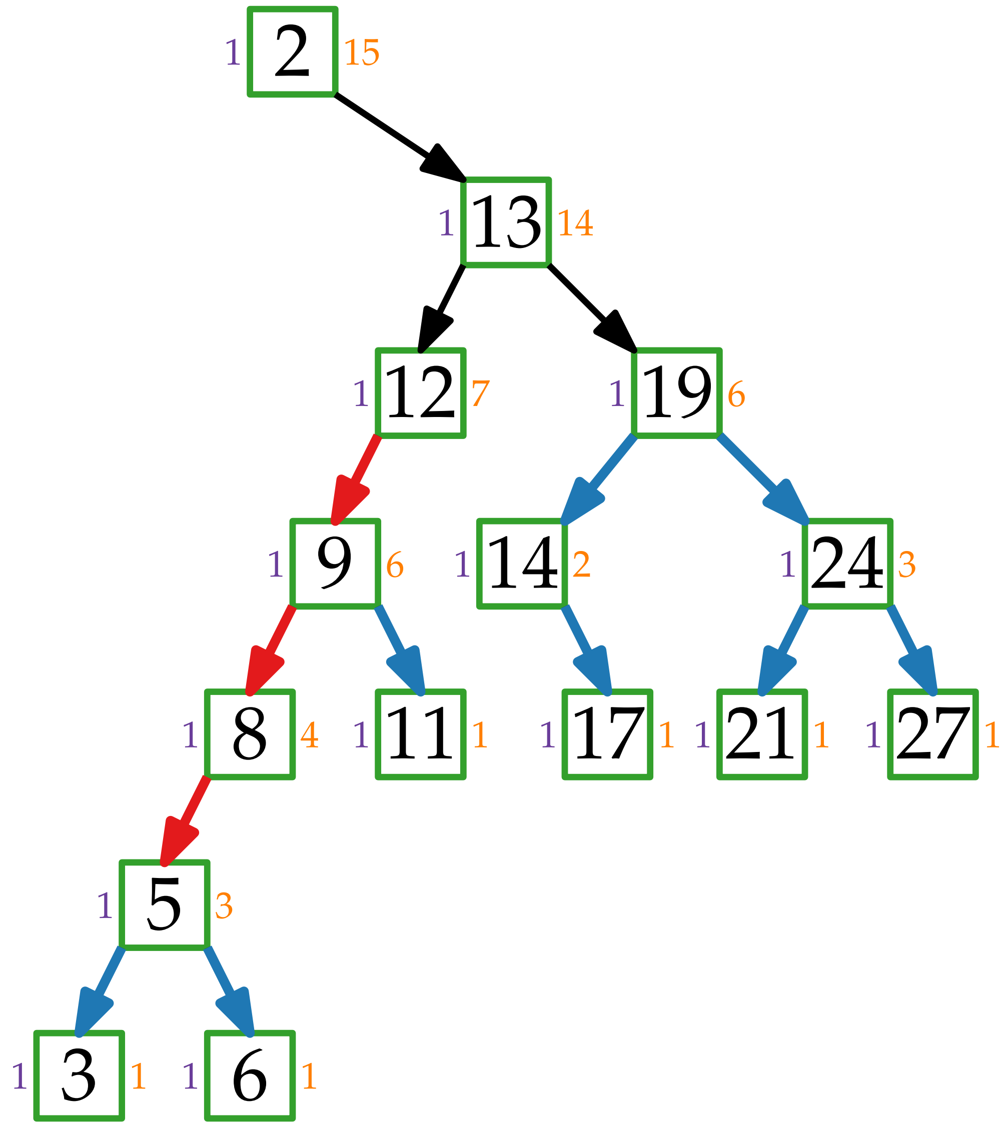
$w(x)$: weight of x (here 1), $W = \sum w(x)$ (here n)

$s(x)$: sum of all $w(x)$ in subtree of x_i

mark edges:

→ $s(\text{child}) \leq s(\text{parent})/2$

→ $s(\text{child}) > s(\text{parent})/2$



Why is Splay Fast?

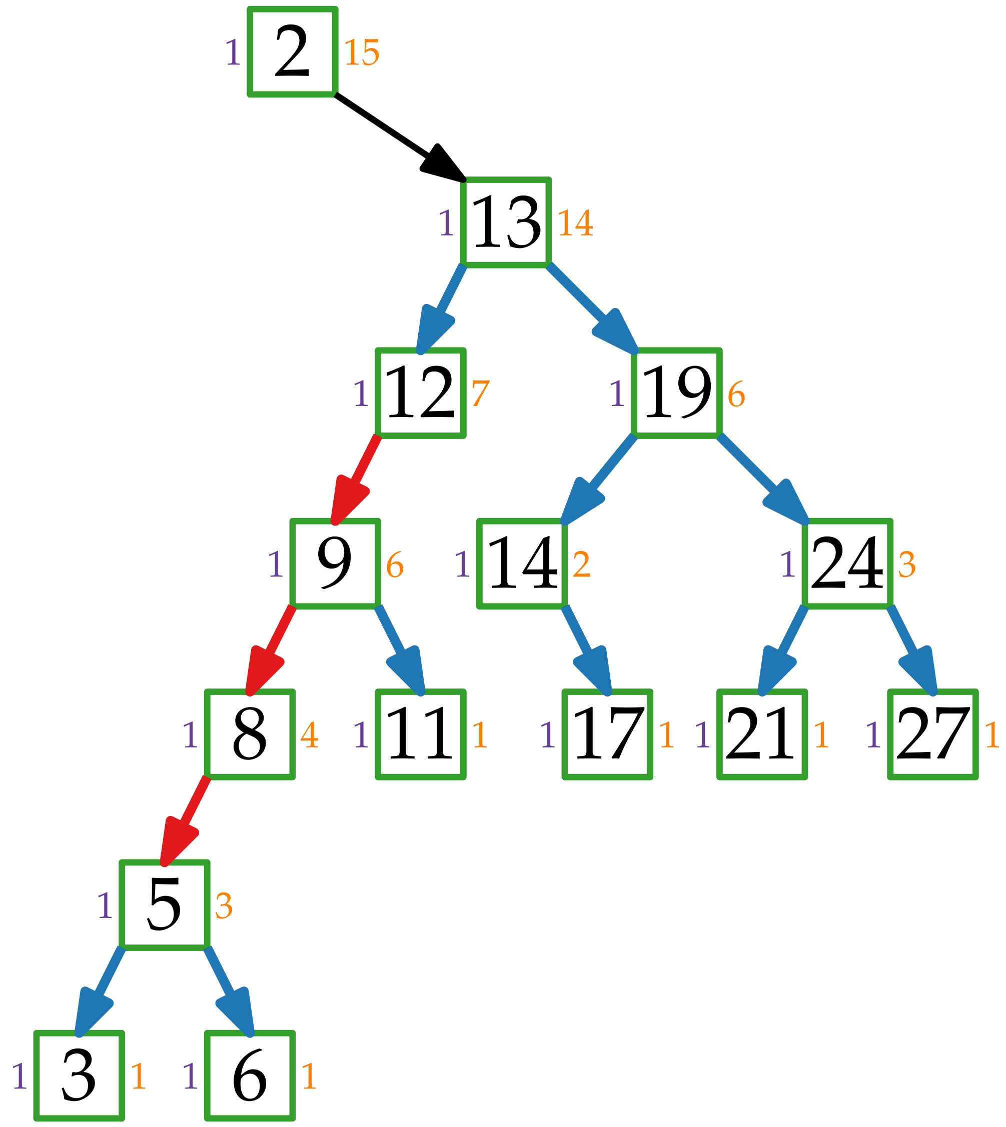
$w(x)$: weight of x (here 1), $W = \sum w(x)$ (here n)

$s(x)$: sum of all $w(x)$ in subtree of x_i

mark edges:

$\longrightarrow$ $s(\text{child}) \leq s(\text{parent})/2$

$\longrightarrow$ $s(\text{child}) > s(\text{parent})/2$



Why is Splay Fast?

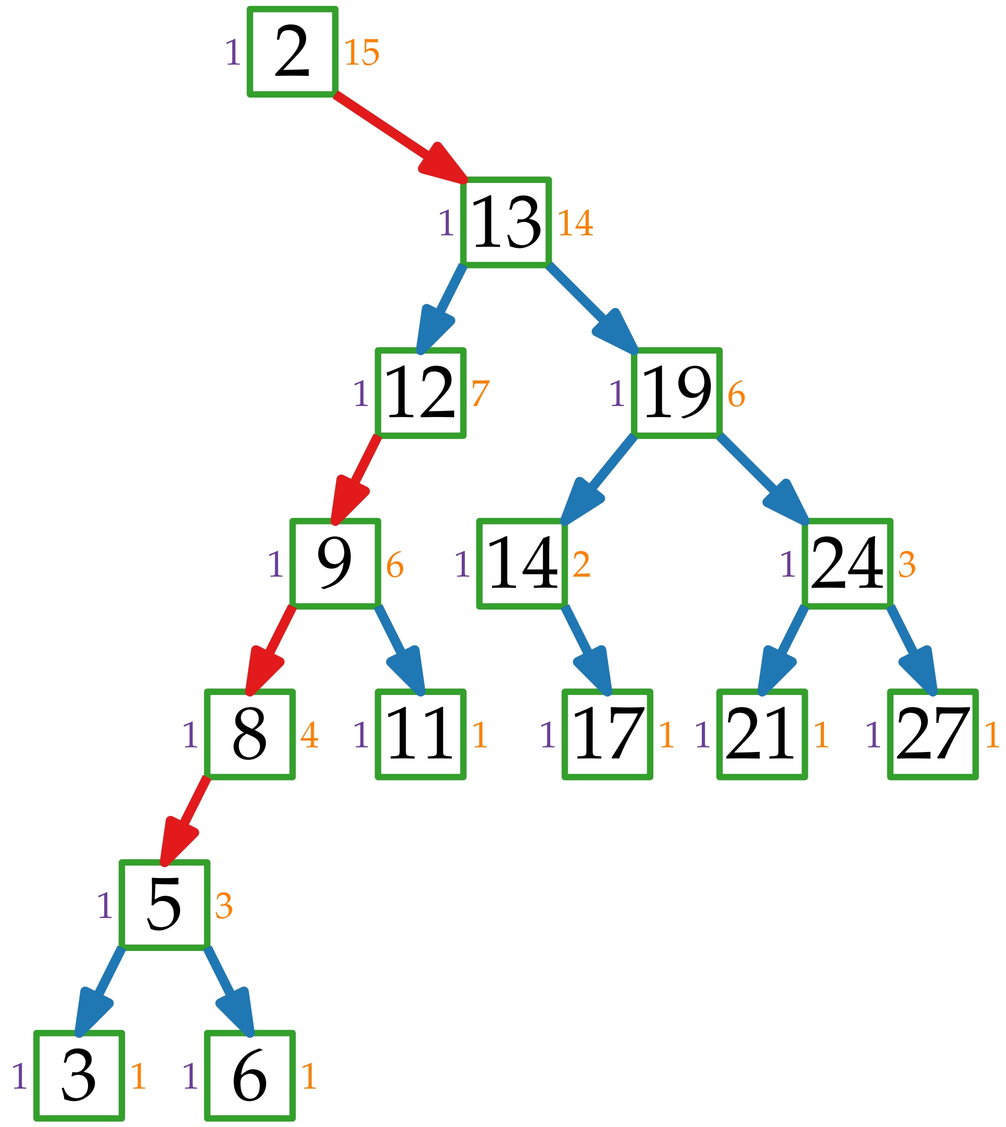
$w(x)$: weight of x (here 1), $W = \sum w(x)$ (here n)

$s(x)$: sum of all $w(x)$ in subtree of x_i

mark edges:

→ $s(\text{child}) \leq s(\text{parent})/2$

→ $s(\text{child}) > s(\text{parent})/2$



Why is Splay Fast?

$w(x)$: weight of x (here 1), $W = \sum w(x)$ (here n)

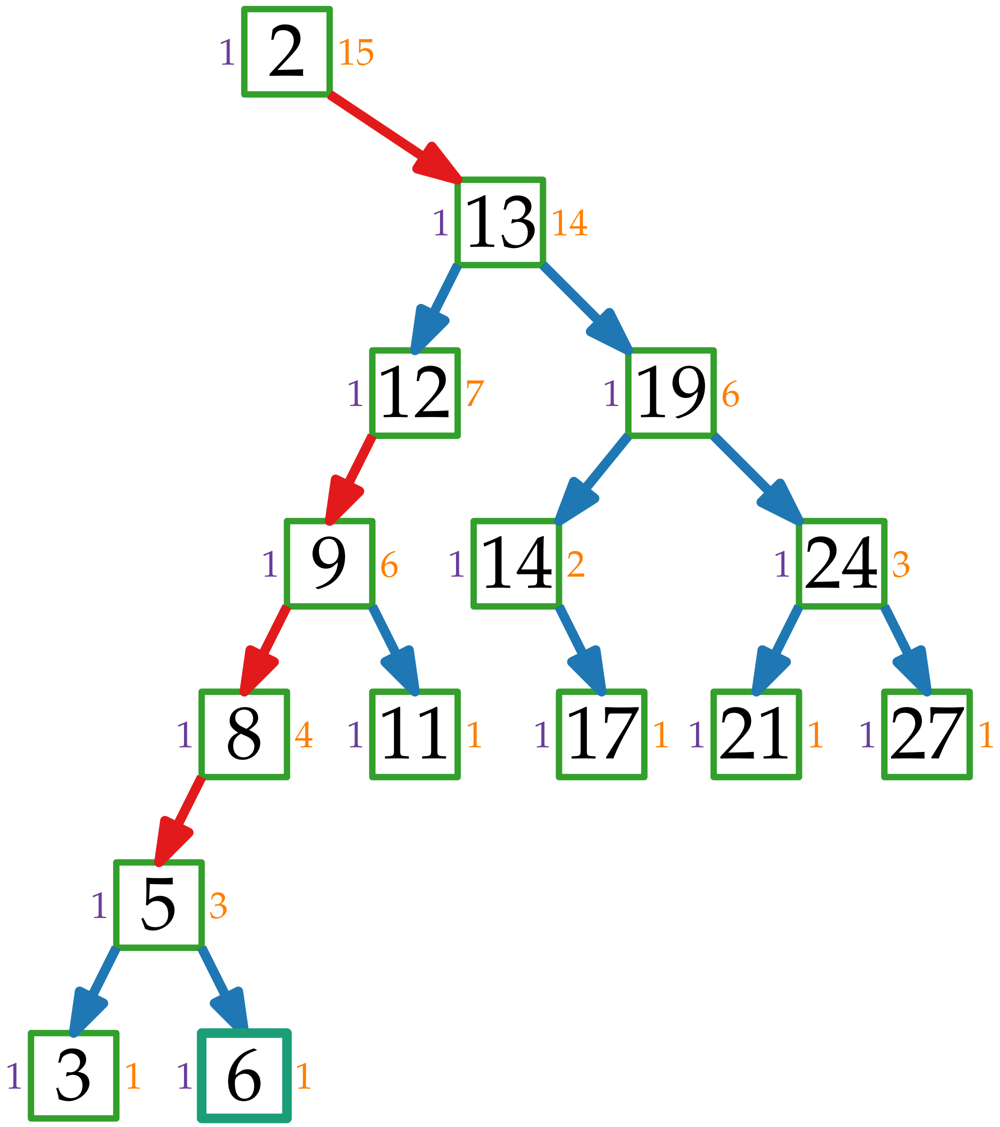
$s(x)$: sum of all $w(x)$ in subtree of x_i

mark edges:

→ $s(\text{child}) \leq s(\text{parent})/2$

→ $s(\text{child}) > s(\text{parent})/2$

Cost to query x_i:



Why is Splay Fast?

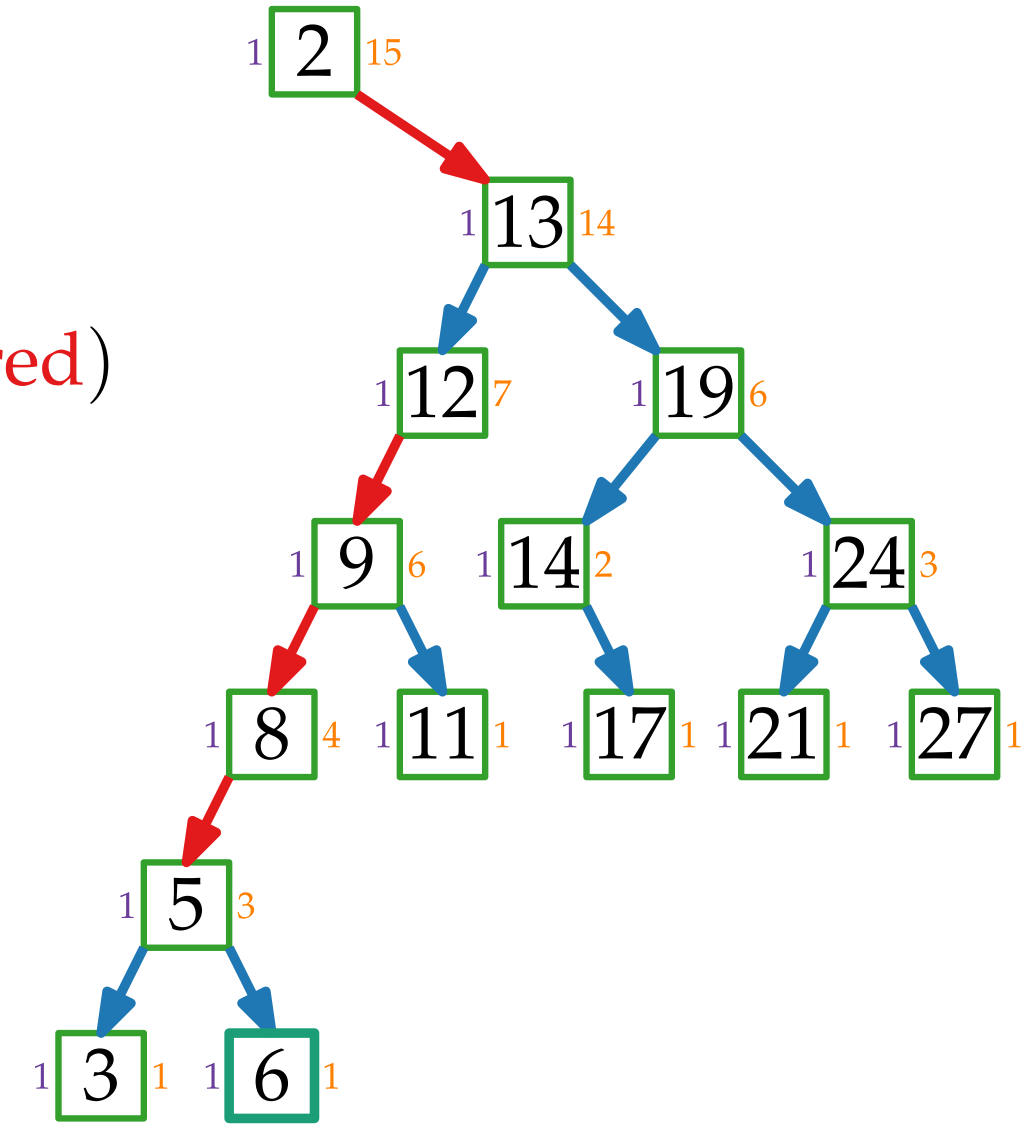
$w(x)$: weight of x (here 1), $W = \sum w(x)$ (here n)
$s(x)$: sum of all $w(x)$ in subtree of x_i

mark edges:

$s(\text{child}) \leq s(\text{parent})/2$

$s(\text{child}) > s(\text{parent})/2$

Cost to query x_i: $O(\#\text{blue} + \#\text{red})$



Why is Splay Fast?

$w(x)$: weight of x (here 1), $W = \sum w(x)$ (here n)

$s(x)$: sum of all $w(x)$ in subtree of x_i

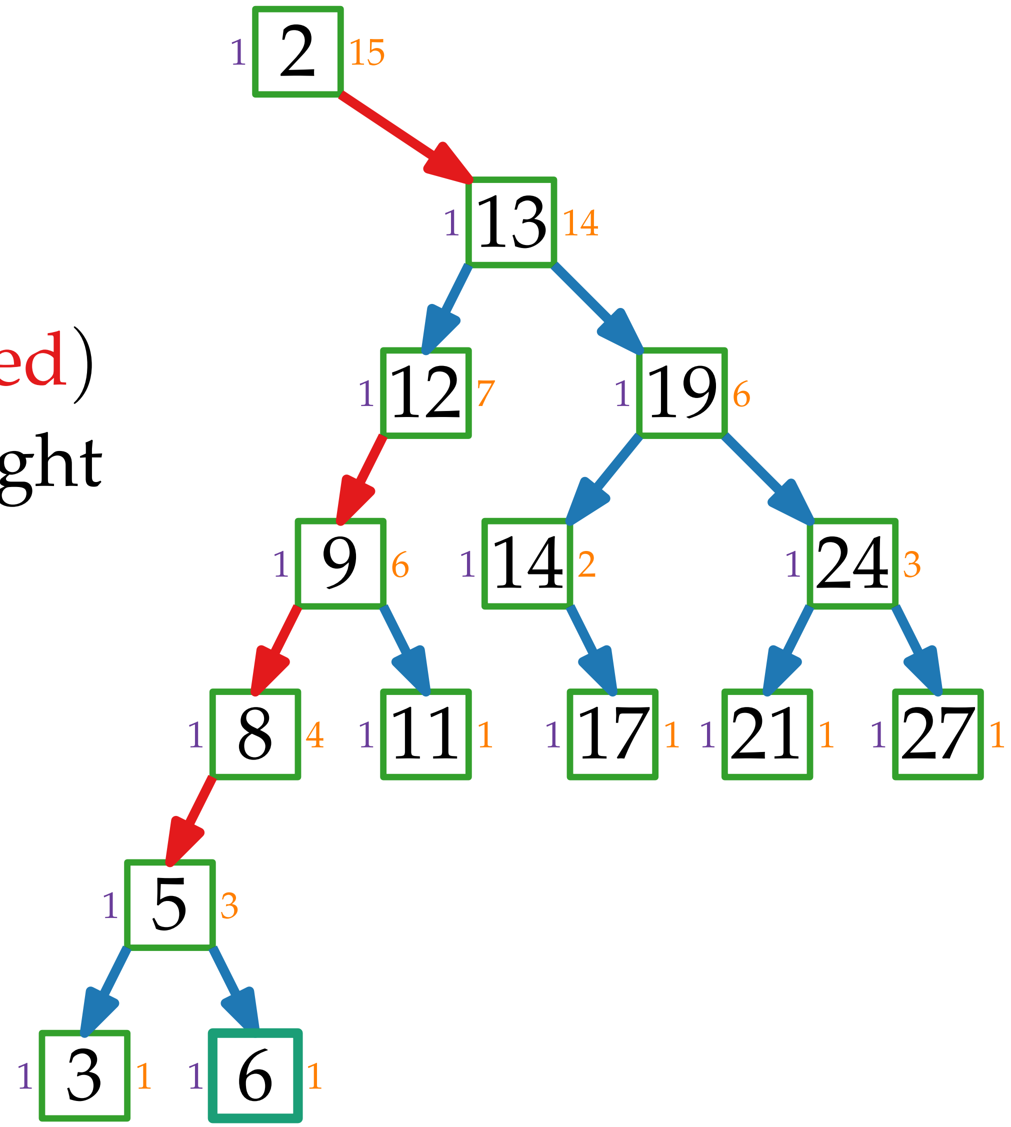
mark edges:

$s(\text{child}) \leq s(\text{parent})/2$

$s(\text{child}) > s(\text{parent})/2$

Cost to query x_i: $O(\#\text{blue} + \#\text{red})$

Idea: blue edges halve the weight



Why is Splay Fast?

$w(x)$: weight of x (here 1), $W = \sum w(x)$ (here n)

$s(x)$: sum of all $w(x)$ in subtree of x_i

mark edges:

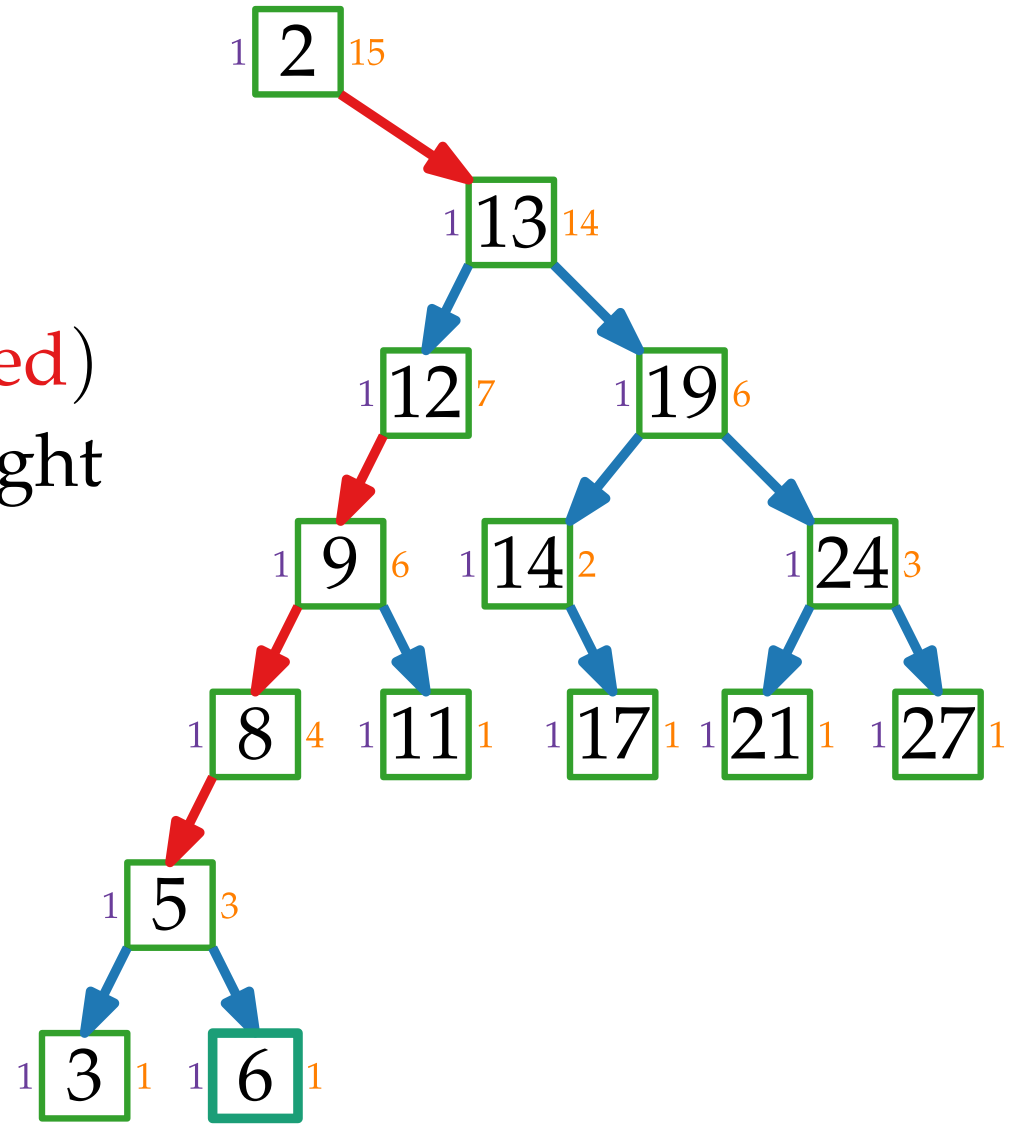
$s(\text{child}) \leq s(\text{parent})/2$

$s(\text{child}) > s(\text{parent})/2$

Cost to query x_i: $O(\#\text{blue} + \#\text{red})$

Idea: blue edges halve the weight

$\Rightarrow \#\text{blue} \in O(\log W)$



Why is Splay Fast?

$w(x)$: weight of x (here 1), $W = \sum w(x)$ (here n)

$s(x)$: sum of all $w(x)$ in subtree of x_i

mark edges:

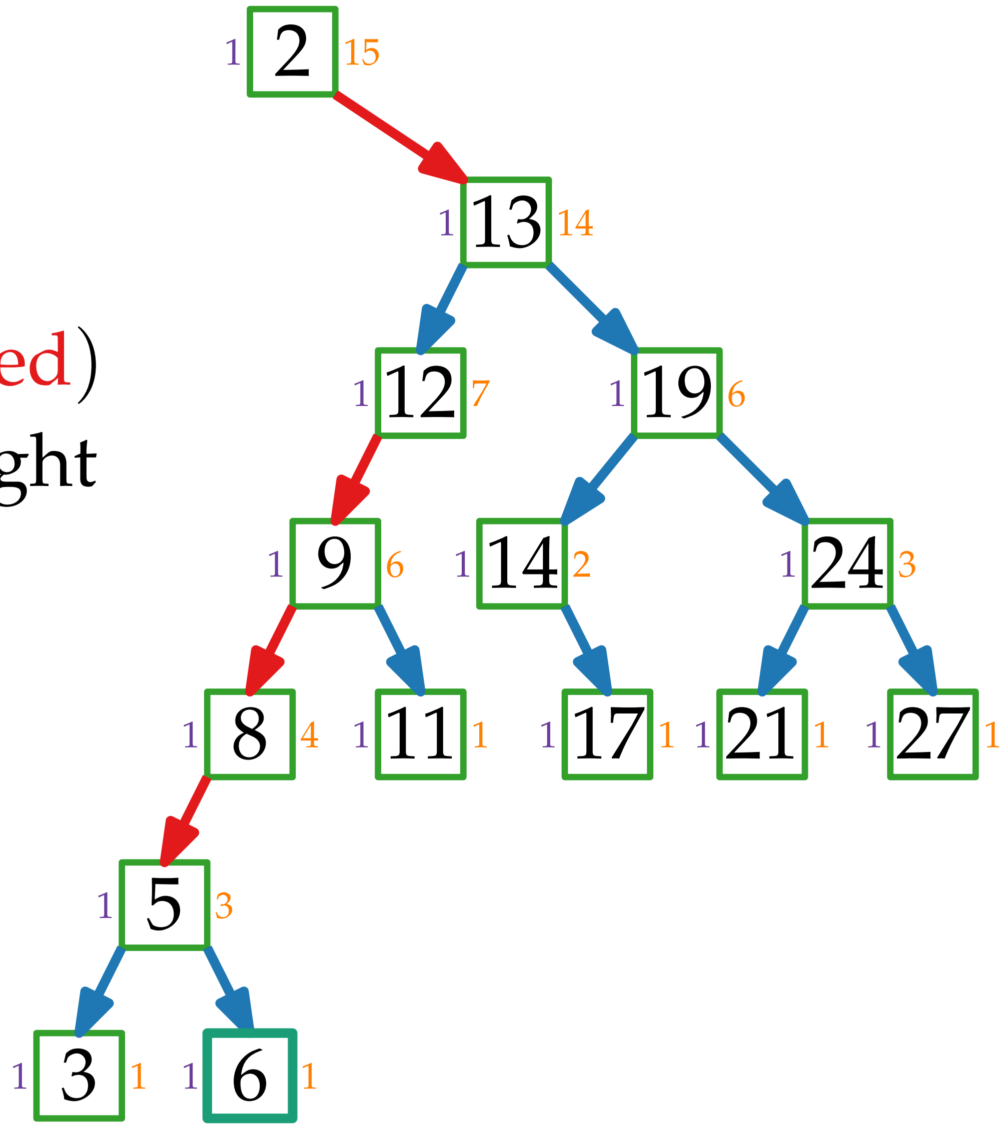
→ $s(\text{child}) \leq s(\text{parent})/2$

→ $s(\text{child}) > s(\text{parent})/2$

Cost to query x_i: $O(\log W + \#\text{red})$

Idea: blue edges halve the weight

$\Rightarrow \#\text{blue} \in O(\log W)$



Why is Splay Fast?

$w(x)$: weight of x (here 1), $W = \sum w(x)$ (here n)

$s(x)$: sum of all $w(x)$ in subtree of x_i

mark edges:

→ $s(\text{child}) \leq s(\text{parent})/2$

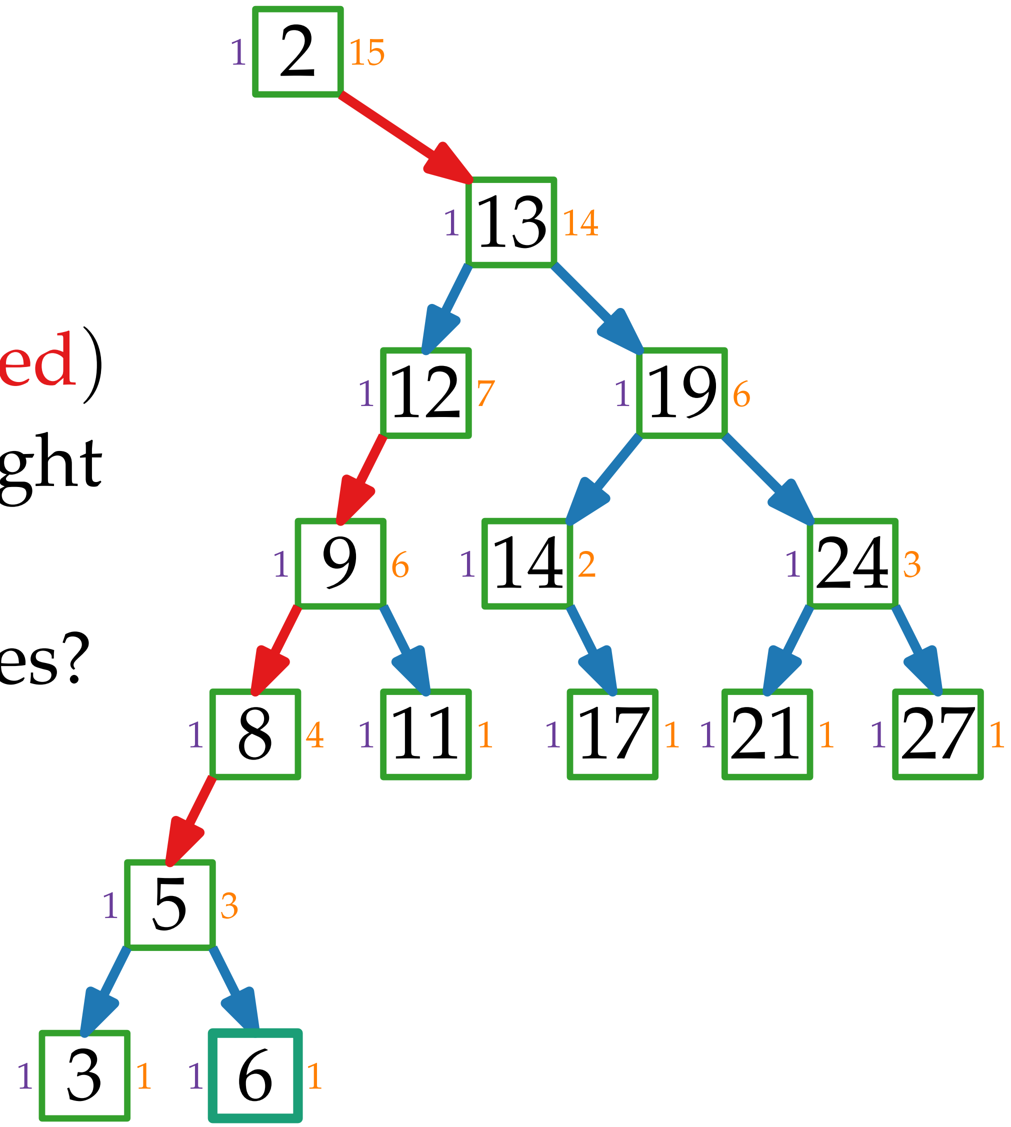
→ $s(\text{child}) > s(\text{parent})/2$

Cost to query x_i: $O(\log W + \#\text{red})$

Idea: blue edges halve the weight

$\Rightarrow \#\text{blue} \in O(\log W)$

How can we amortize red edges?



Why is Splay Fast?

$w(x)$: weight of x (here 1), $W = \sum w(x)$ (here n)

$s(x)$: sum of all $w(x)$ in subtree of x_i

mark edges:

$\longrightarrow$ $s(\text{child}) \leq s(\text{parent})/2$

$\longrightarrow$ $s(\text{child}) > s(\text{parent})/2$

Cost to query x_i: $O(\log W + \#\text{red})$

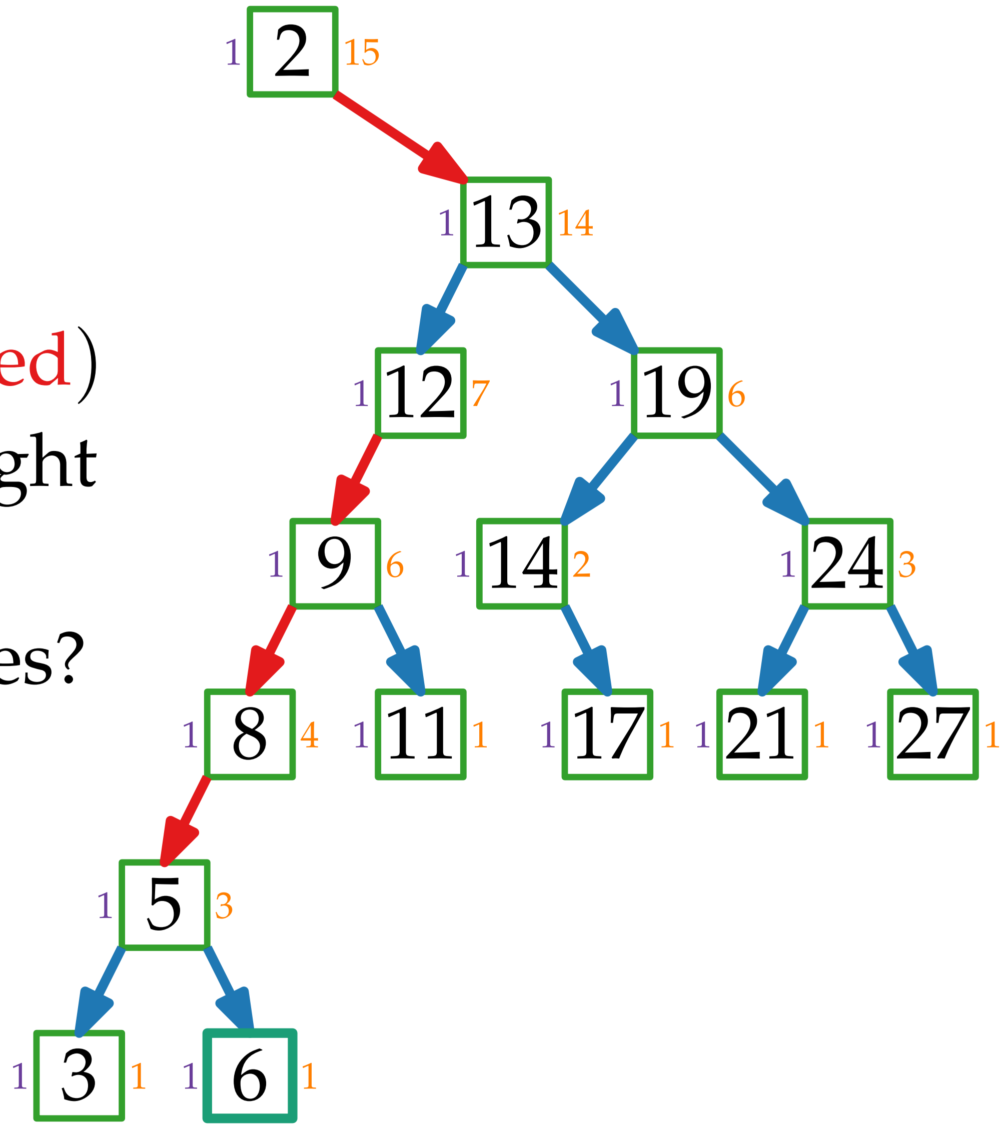
Idea: blue edges halve the weight

$\Rightarrow \#\text{blue} \in O(\log W)$

How can we amortize red edges?

Use sum-of-logs potential

$\Phi = \sum \log s(x)$



Why is Splay Fast?

$w(x)$: weight of x (here 1), $W = \sum w(x)$ (here n)

$s(x)$: sum of all $w(x)$ in subtree of x_i

mark edges:

→ $s(\text{child}) \leq s(\text{parent})/2$

→ $s(\text{child}) > s(\text{parent})/2$

Cost to query x_i: $O(\log W + \#\text{red})$

Idea: blue edges halve the weight

$\Rightarrow \#\text{blue} \in O(\log W)$

How can we amortize red edges?

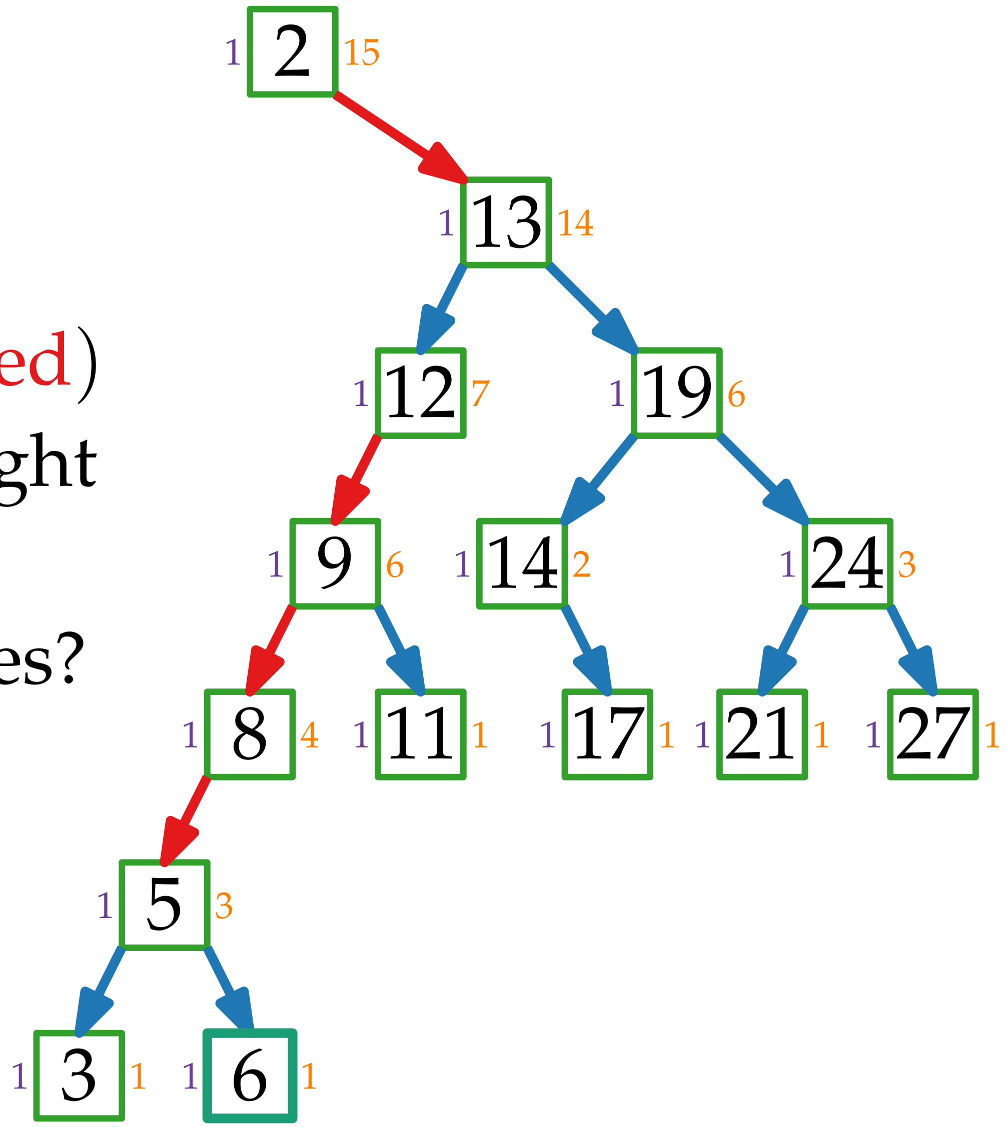
Use sum-of-logs potential

$\Phi = \sum \log s(x)$

Amortized cost:

real cost + $\Phi_+ - \Phi$

(potential after splay)



What is Potential?

Φ represent work that has been “paid for” but not yet performed.

What is Potential?

Φ represent work that has been "paid for" but not yet performed.

amortized cost per step: real cost $+ \Phi_{+} - \Phi$

What is Potential?

Φ represent work that has been "paid for" but not yet performed.

amortized cost per step: real cost $+ \Phi_+ - \Phi$

total cost $= \Phi_0 - \Phi_{\text{end}} + \sum$ amortized cost

What is Potential?

Φ represent work that has been "paid for" but not yet performed.

amortized cost per step: real cost $+ \Phi_{+} - \Phi$

total cost $= \Phi_0 - \Phi_{\text{end}} + \sum$ amortized cost

Example (ADS): Stack with multipop

What is Potential?

Φ represent work that has been "paid for" but not yet performed.

amortized cost per step: real cost $+\,\Phi_{+} - \Phi$

total cost $= \Phi_0 - \Phi_{\text{end}} + \sum$ amortized cost

Example (ADS): Stack with multipop

Φ :=size of the stack

What is Potential?

Φ represent work that has been “paid for” but not yet performed.

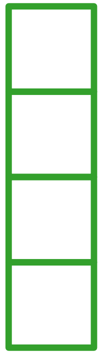
amortized cost per step: real cost $+\,\Phi_{+} - \Phi$

total cost $= \Phi_0 - \Phi_{\text{end}} + \sum$ amortized cost

Example (ADS): Stack with multipop

$\Phi :=$size of the stack

$\Phi = 0$



What is Potential?

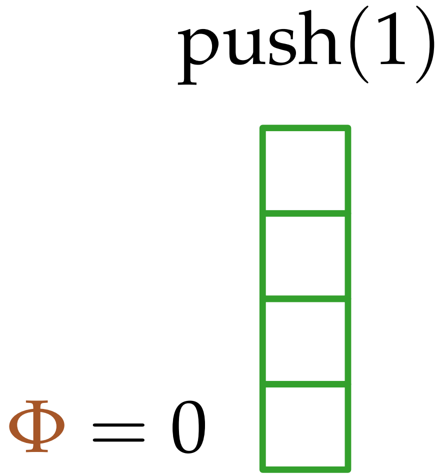
Φ represent work that has been "paid for" but not yet performed.

amortized cost per step: real cost $+ \Phi_+ - \Phi$

total cost $= \Phi_0 - \Phi_{\text{end}} + \sum$ amortized cost

Example (ADS): Stack with multipop

Φ :=size of the stack



What is Potential?

Φ represent work that has been “paid for” but not yet performed.

amortized cost per step: real cost $+ \Phi_+ - \Phi$

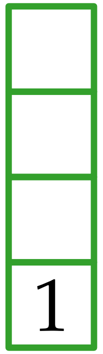
total cost $= \Phi_0 - \Phi_{\text{end}} + \sum$ amortized cost

Example (ADS): Stack with multipop

Φ :=size of the stack

push(1)

$\Phi = 1$



What is Potential?

Φ represent work that has been “paid for” but not yet performed.

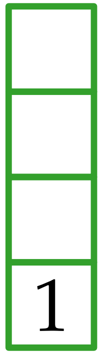
amortized cost per step: real cost $+ \Phi_+ - \Phi$

total cost $= \Phi_0 - \Phi_{\text{end}} + \sum$ amortized cost

Example (ADS): Stack with multipop

Φ :=size of the stack

push(2)



$\Phi = 1$

What is Potential?

Φ represent work that has been "paid for" but not yet performed.

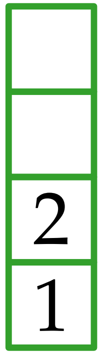
amortized cost per step: real cost $+ \Phi_+ - \Phi$

total cost $= \Phi_0 - \Phi_{\text{end}} + \sum$ amortized cost

push(2)

Example (ADS): Stack with multipop

Φ :=size of the stack



$\Phi = 2$

What is Potential?

Φ represent work that has been "paid for" but not yet performed.

amortized cost per step: real cost $+ \Phi_{+} - \Phi$

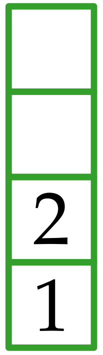
total cost $= \Phi_0 - \Phi_{\text{end}} + \sum$ amortized cost

push(3)

Example (ADS): Stack with multipop

Φ :=size of the stack

$\Phi = 2$



What is Potential?

Φ represent work that has been "paid for" but not yet performed.

amortized cost per step: real cost $+ \Phi_+ - \Phi$

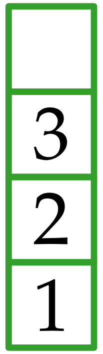
total cost $= \Phi_0 - \Phi_{\text{end}} + \sum$ amortized cost

push(3)

Example (ADS): Stack with multipop

Φ :=size of the stack

$\Phi = 3$



What is Potential?

Φ represent work that has been “paid for” but not yet performed.

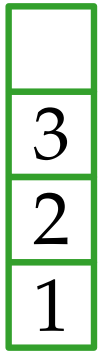
amortized cost per step: real cost $+ \Phi_{+} - \Phi$

total cost $= \Phi_0 - \Phi_{\text{end}} + \sum$ amortized cost

Example (ADS): Stack with multipop

Φ :=size of the stack

pop(2)



$\Phi = 3$

What is Potential?

Φ represent work that has been "paid for" but not yet performed.

amortized cost per step: real cost $+ \Phi_+ - \Phi$

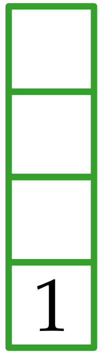
total cost $= \Phi_0 - \Phi_{\text{end}} + \sum$ amortized cost

Example (ADS): Stack with multipop

Φ :=size of the stack

pop(2)

$\Phi = 1$



What is Potential?

Φ represent work that has been "paid for" but not yet performed.

amortized cost per step: real cost $+ \Phi_+ - \Phi$

total cost $= \Phi_0 - \Phi_{\text{end}} + \sum$ amortized cost

Example (ADS): Stack with multipop

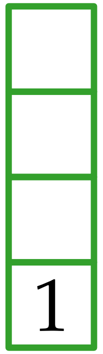
Φ :=size of the stack

push:

pop(k):

pop(2)

$\Phi = 1$



What is Potential?

Φ represent work that has been “paid for” but not yet performed.

amortized cost per step: real cost $+ \Phi_+ - \Phi$

total cost $= \Phi_0 - \Phi_{\text{end}} + \sum$ amortized cost

Example (ADS): Stack with multipop

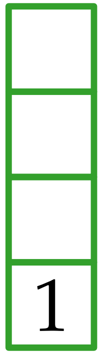
Φ :=size of the stack

push: $\quad 1 + \Phi_+ - \Phi$

pop(k):

pop(2)

$\Phi = 1$



What is Potential?

Φ represent work that has been "paid for" but not yet performed.

amortized cost per step: real cost $+ \Phi_+ - \Phi$

total cost $= \Phi_0 - \Phi_{\text{end}} + \sum$ amortized cost

pop(2)

Example (ADS): Stack with multipop

Φ :=size of the stack

$\Phi = 1$ $\boxed{1}$

push: $\quad 1 + \Phi_+ - \Phi \;= 2$

pop(k):

What is Potential?

Φ represent work that has been "paid for" but not yet performed.

amortized cost per step: real cost $+ \Phi_+ - \Phi$

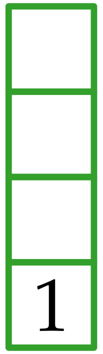
total cost $= \Phi_0 - \Phi_{\text{end}} + \sum$ amortized cost

pop(2)

Example (ADS): Stack with multipop

Φ :=size of the stack

$\Phi = 1$



push: $1 + \Phi_+ - \Phi \ = 2$

pop(k): $k + \Phi_+ - \Phi$

What is Potential?

Φ represent work that has been "paid for" but not yet performed.

amortized cost per step: real cost $+ \Phi_+ - \Phi$

total cost $= \Phi_0 - \Phi_{\text{end}} + \sum$ amortized cost

Example (ADS): Stack with multipop

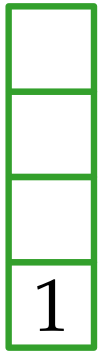
Φ :=size of the stack

push: $\quad 1 + \Phi_+ - \Phi \ = 2$

pop(k): $\quad k + \Phi_+ - \Phi \ = 0$

pop(2)

$\Phi = 1$



What is Potential?

Φ represent work that has been "paid for" but not yet performed.

amortized cost per step: real cost $+ \Phi_+ - \Phi$

total cost $= \Phi_0 - \Phi_{\text{end}} + \sum$ amortized cost

Example (ADS): Stack with multipop

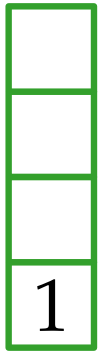
Φ :=size of the stack

push: $1 + \Phi_+ - \Phi \; = 2$

$\text{pop}(k)$: $k + \Phi_+ - \Phi \; = 0$

total cost $= \Phi_0 - \Phi_{\text{end}} +$ amortized cost

$\text{pop}(2)$



$\Phi = 1$

What is Potential?

Φ represent work that has been "paid for" but not yet performed.

amortized cost per step: real cost $+\,\Phi_+ - \Phi$

total cost $= \Phi_0 - \Phi_{\text{end}} + \sum$ amortized cost

Example (ADS): Stack with multipop

Φ :=size of the stack

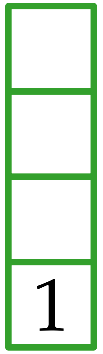
push: $\quad 1 + \Phi_+ - \Phi \ = 2$

pop(k): $\quad k + \Phi_+ - \Phi \ = 0$

$$\begin{aligned}\text{total cost} &= \Phi_0 - \Phi_{\text{end}} + \text{amortized cost}\\ &\leq \Phi_0 - \Phi_{\text{end}} + 2n\end{aligned}$$

pop(2)

$\Phi = 1$



What is Potential?

Φ represent work that has been "paid for" but not yet performed.

amortized cost per step: real cost $+ \Phi_+ - \Phi$

total cost $= \Phi_0 - \Phi_{\text{end}} + \sum$ amortized cost

Example (ADS): Stack with multipop

Φ :=size of the stack

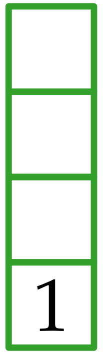
push: $\quad 1 + \Phi_+ - \Phi \; = 2$

pop(k): $\quad k + \Phi_+ - \Phi \; = 0$

$$\begin{aligned} \text{total cost} &= \Phi_0 - \Phi_{\text{end}} + \text{amortized cost} \\ &\leq \Phi_0 - \Phi_{\text{end}} + 2n \\ &\leq 2n \end{aligned}$$

pop(2)

$\Phi = 1$



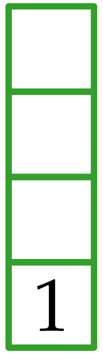
What is Potential?

Φ represent work that has been "paid for" but not yet performed.

amortized cost per step: real cost $+ \Phi_+ - \Phi$

total cost $= \Phi_0 - \Phi_{\text{end}} + \sum$ amortized cost

pop(2)



$\Phi = 1$

Example (ADS): Stack with multipop

Φ :=size of the stack

push: $\quad 1 + \Phi_+ - \Phi \;= 2$

pop(k): $\quad k + \Phi_+ - \Phi \;= 0$

$$
\begin{aligned}
\text{total cost} &= \Phi_0 - \Phi_{\text{end}} + \text{amortized cost} \\
&\leq \Phi_0 - \Phi_{\text{end}} + 2n \\
&\leq 2n \;= O(n)
\end{aligned}
$$

Why is Splay Fast?

$w(x)$: weight of x (here 1), $W = \sum w(x)$ (here n)

$s(x)$: sum of all $w(x)$ in subtree of x_i

mark edges:

→ $s(\text{child}) \leq s(\text{parent})/2$

→ $s(\text{child}) > s(\text{parent})/2$

Cost to query x_i: $O(\log W + \#\text{red})$

Idea: blue edges halve the weight

$\Rightarrow \#\text{blue} \in O(\log W)$

How can we amortize red edges?

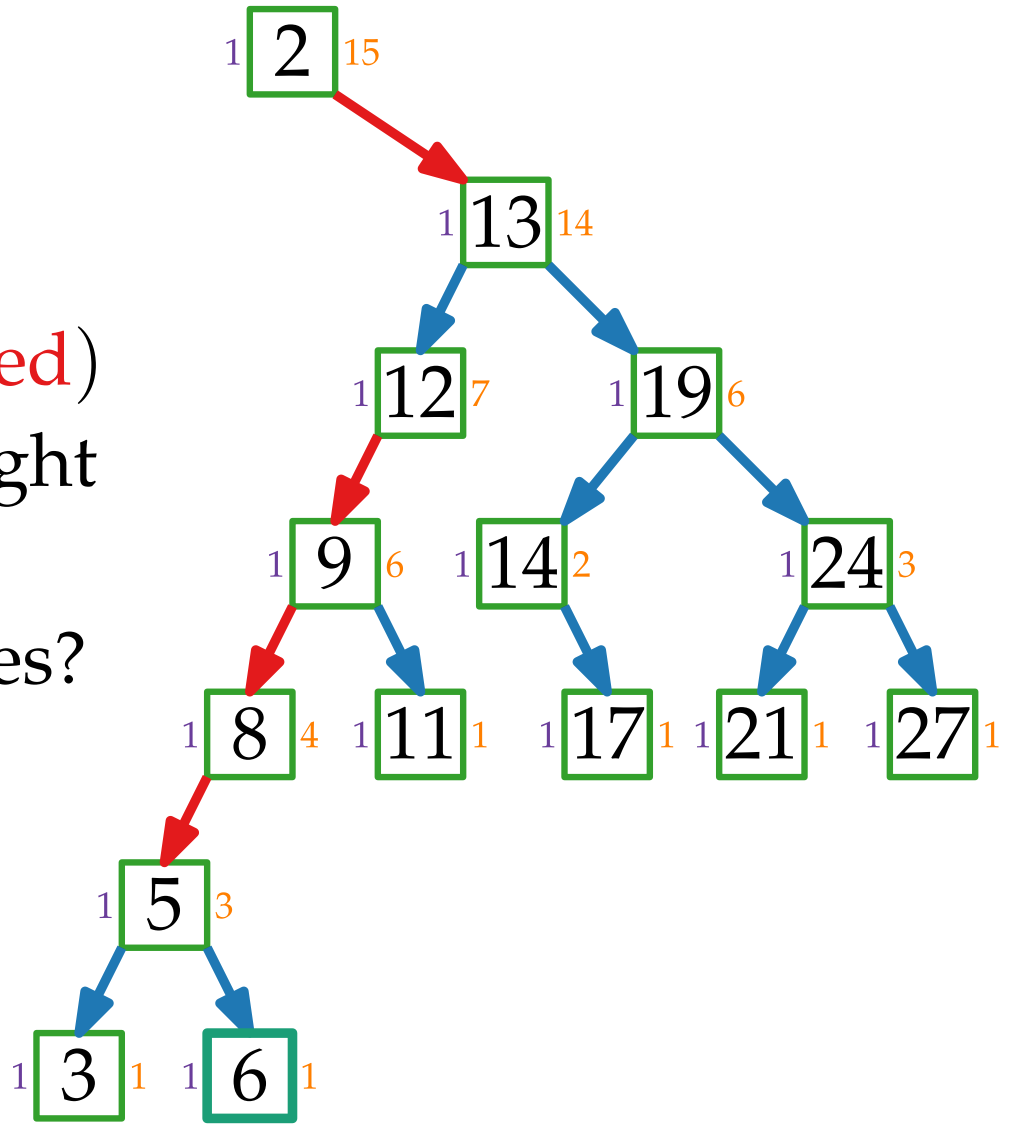
Use sum-of-logs potential

$\Phi = \sum \log s(x)$

Amortized cost:

real cost + $\Phi_+ - \Phi$

(potential after splay)



Why is Splay Fast?

$w(x)$: weight of x (here 1), $W = \sum w(x)$ (here n)

$s(x)$: sum of all $w(x)$ in subtree of x_i

mark edges:

→ $s(\text{child}) \leq s(\text{parent})/2$

→ $s(\text{child}) > s(\text{parent})/2$

Cost to query x_i: $O(\log W + \#\text{red})$

Idea: blue edges halve the weight

$\Rightarrow \#\text{blue} \in O(\log W)$

How can we amortize red edges?

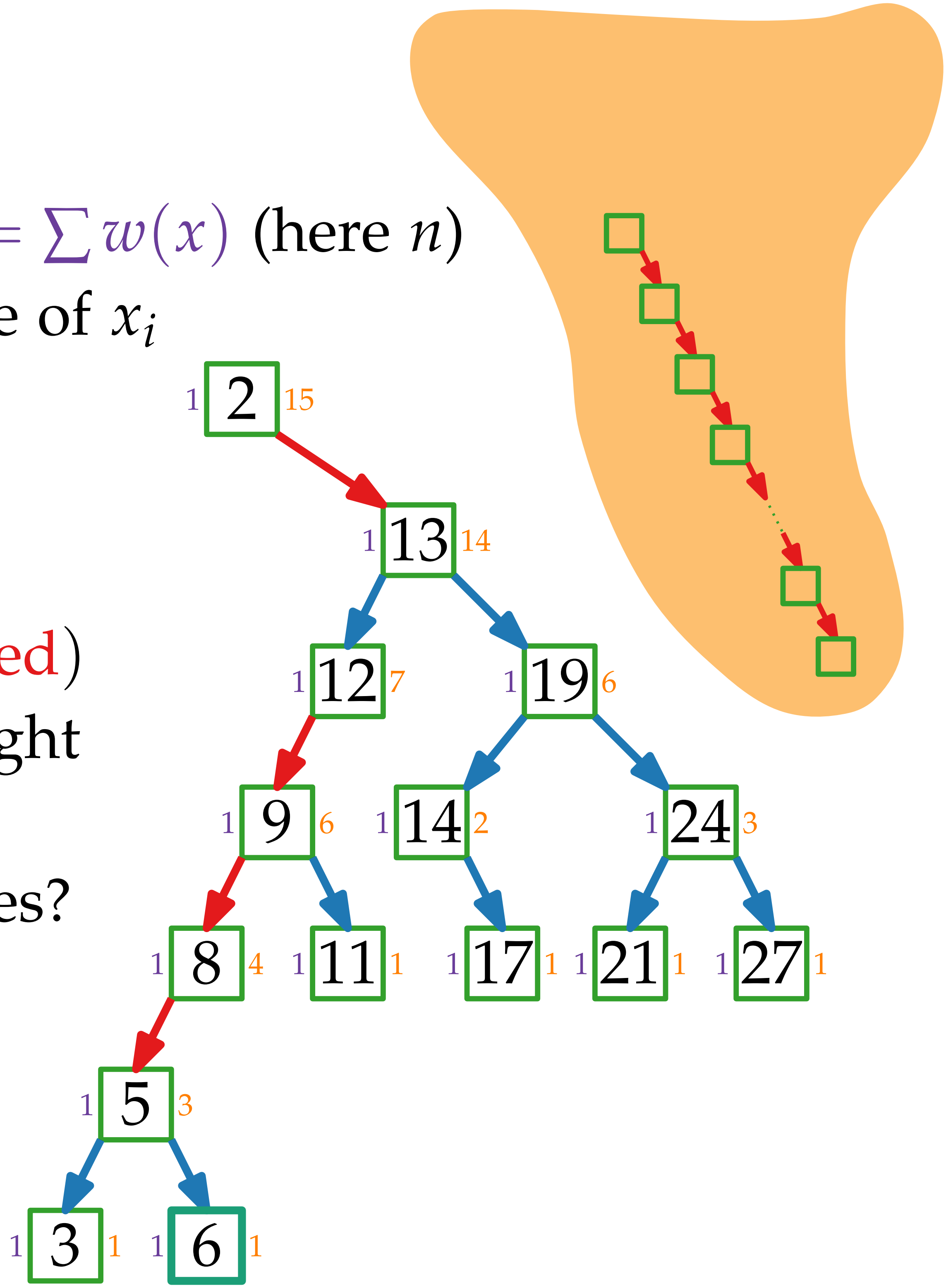
Use sum-of-logs potential

$\Phi = \sum \log s(x)$

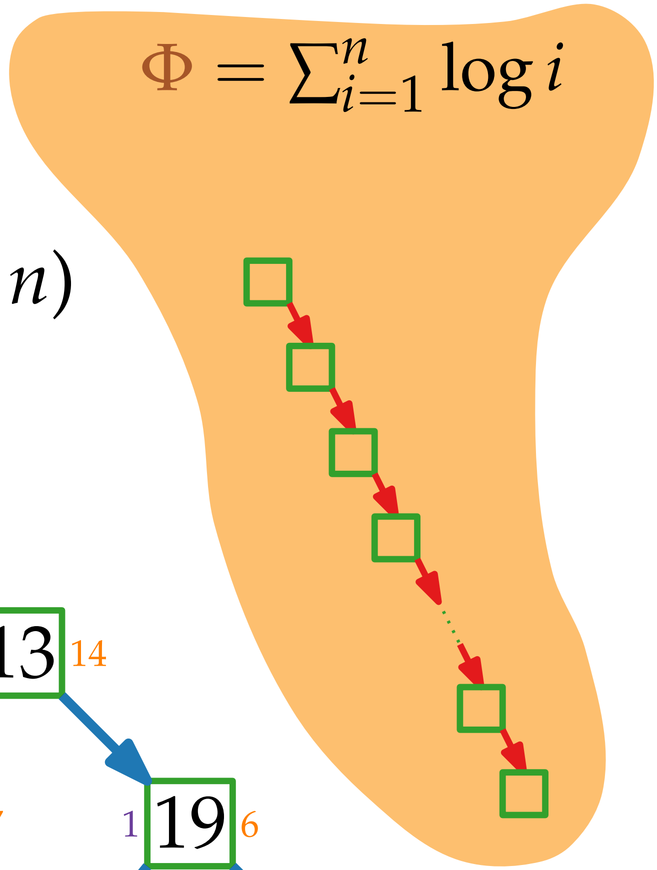
Amortized cost:

real cost + $\Phi_+ - \Phi$

(potential after splay)



Why is Splay Fast?



$w(x)$: weight of x (here 1), $W = \sum w(x)$ (here n)
$s(x)$: sum of all $w(x)$ in subtree of x_i

mark edges:
→ $s(\text{child}) \leq s(\text{parent})/2$
→ $s(\text{child}) > s(\text{parent})/2$

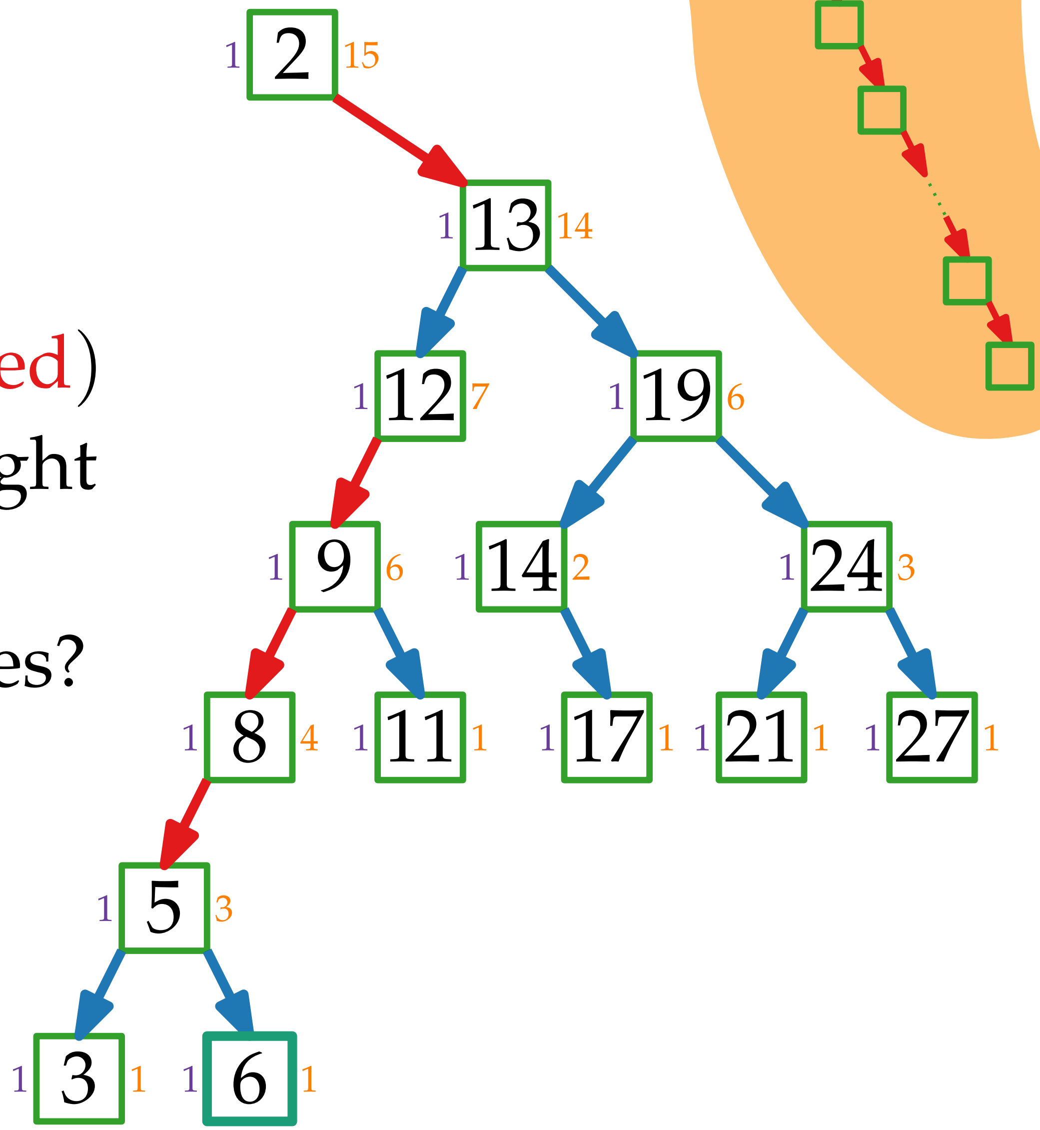
Cost to query x_i: $O(\log W + \#\text{red})$

Idea: blue edges halve the weight
$\Rightarrow \#\text{blue} \in O(\log W)$

How can we amortize red edges?

Use sum-of-logs potential
$\Phi = \sum \log s(x)$

Amortized cost:
real cost + $\Phi_+ - \Phi$
(potential after splay)



Why is Splay Fast?

$w(x)$: weight of x (here 1), $W = \sum w(x)$ (here n)
$s(x)$: sum of all $w(x)$ in subtree of x_i

mark edges:
$s(\text{child}) \leq s(\text{parent})/2$
$s(\text{child}) > s(\text{parent})/2$

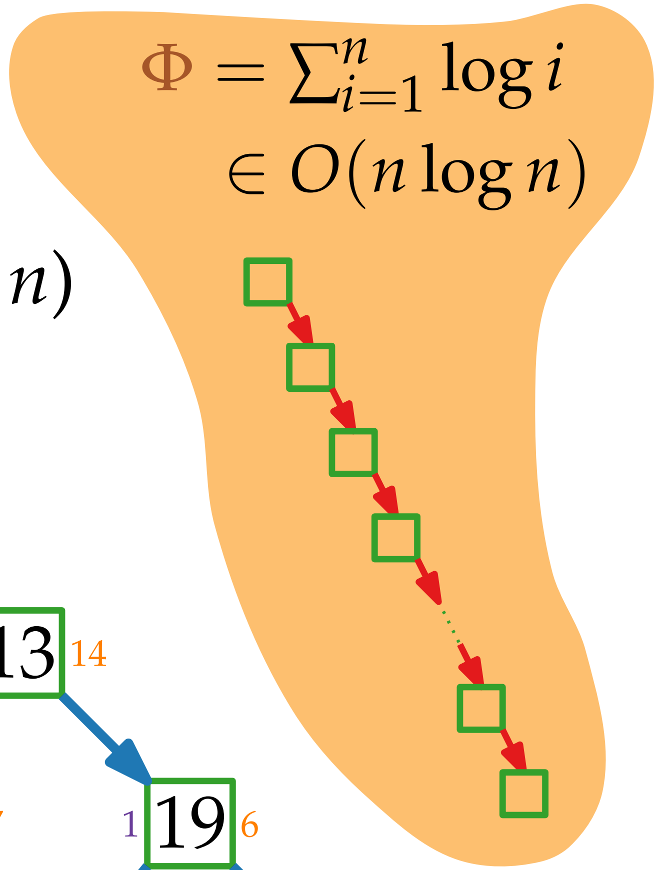
Cost to query x_i: $O(\log W + \#\text{red})$

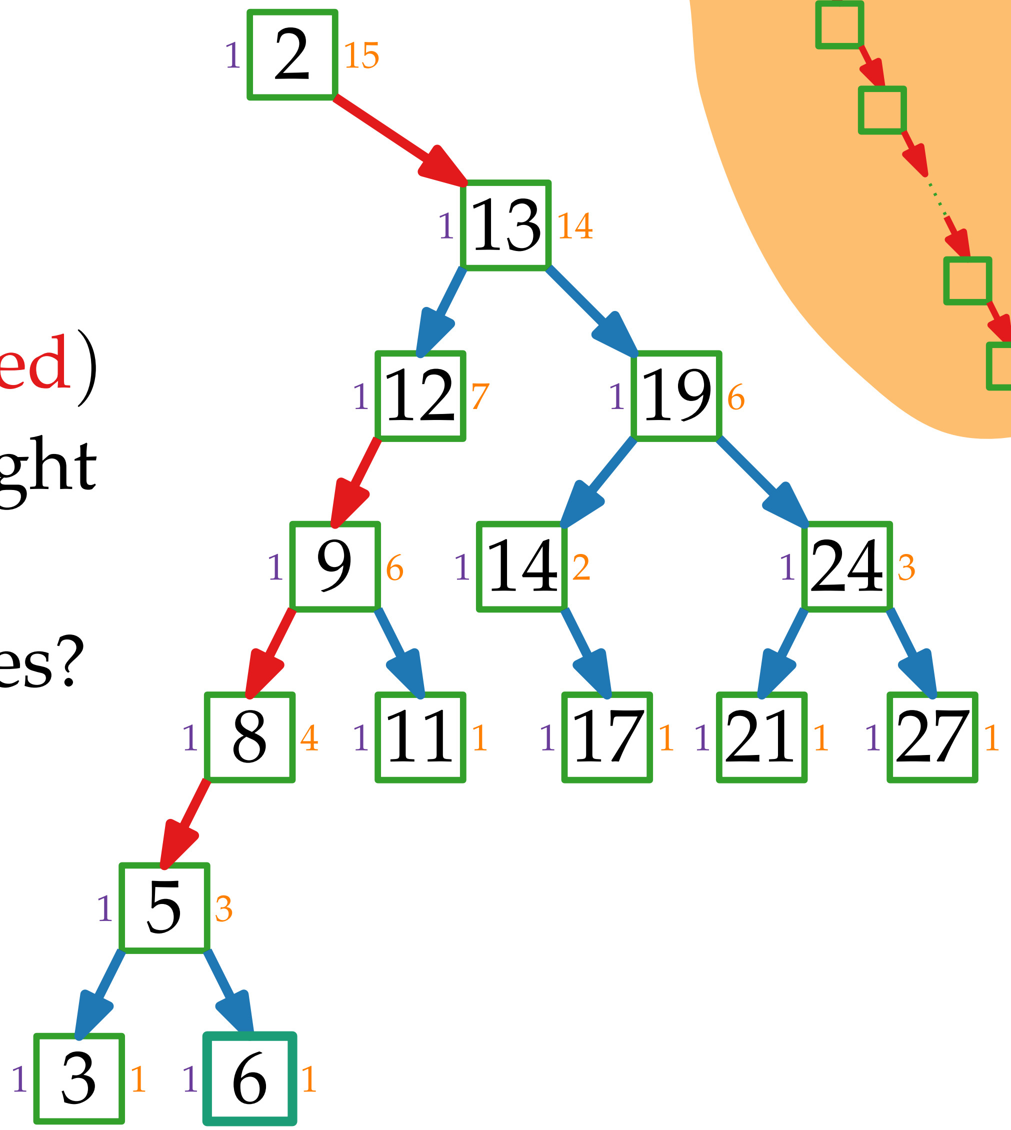
Idea: blue edges halve the weight
$\Rightarrow \#\text{blue} \in O(\log W)$

How can we amortize red edges?

Use sum-of-logs potential
$\Phi = \sum \log s(x)$

Amortized cost:
real cost + $\Phi_+ - \Phi$
(potential after splay)





Why is Splay Fast?

$w(x)$: weight of x (here 1), $W = \sum w(x)$ (here n)
$s(x)$: sum of all $w(x)$ in subtree of x_i

mark edges:

$\longrightarrow$ $s(\text{child}) \leq s(\text{parent})/2$

$\longrightarrow$ $s(\text{child}) > s(\text{parent})/2$

Cost to query x_i: $O(\log W + \#\text{red})$

Idea: blue edges halve the weight

$\Rightarrow \#\text{blue} \in O(\log W)$

How can we amortize red edges?

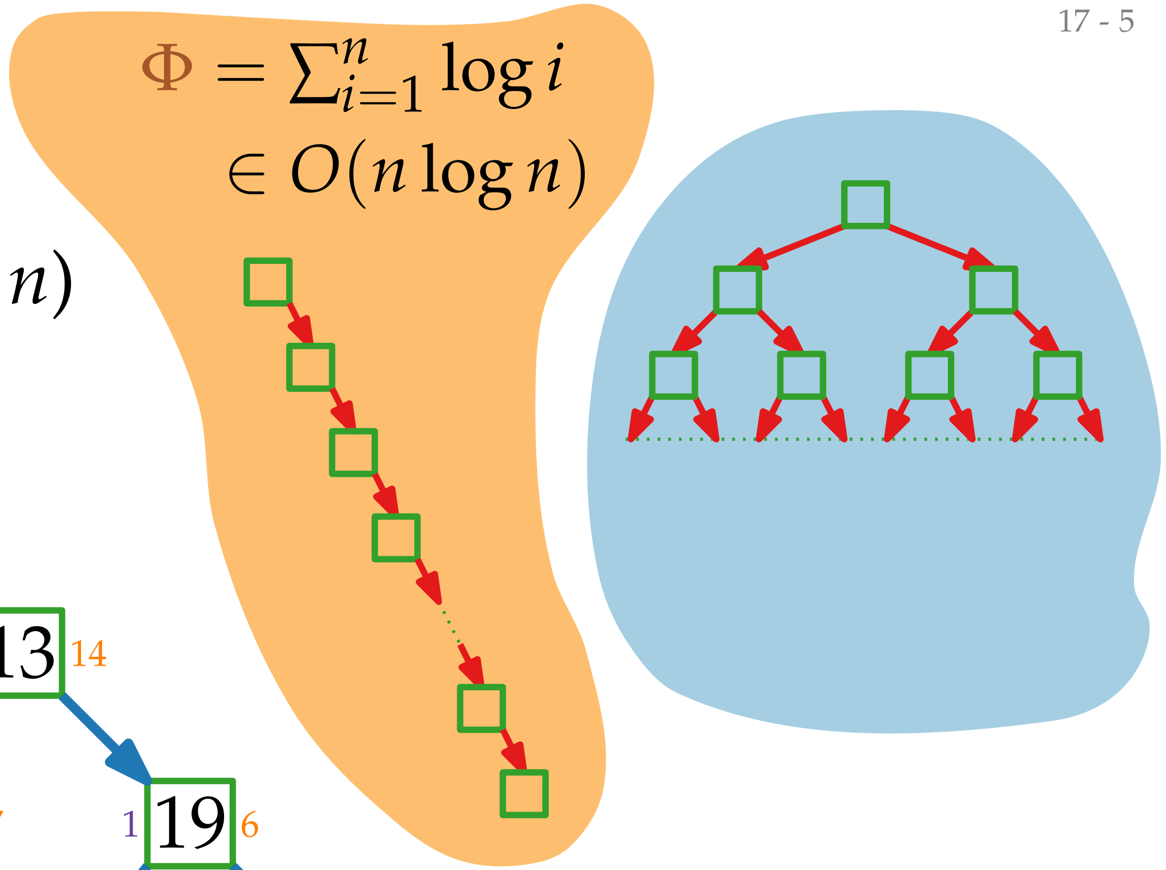
Use sum-of-logs potential

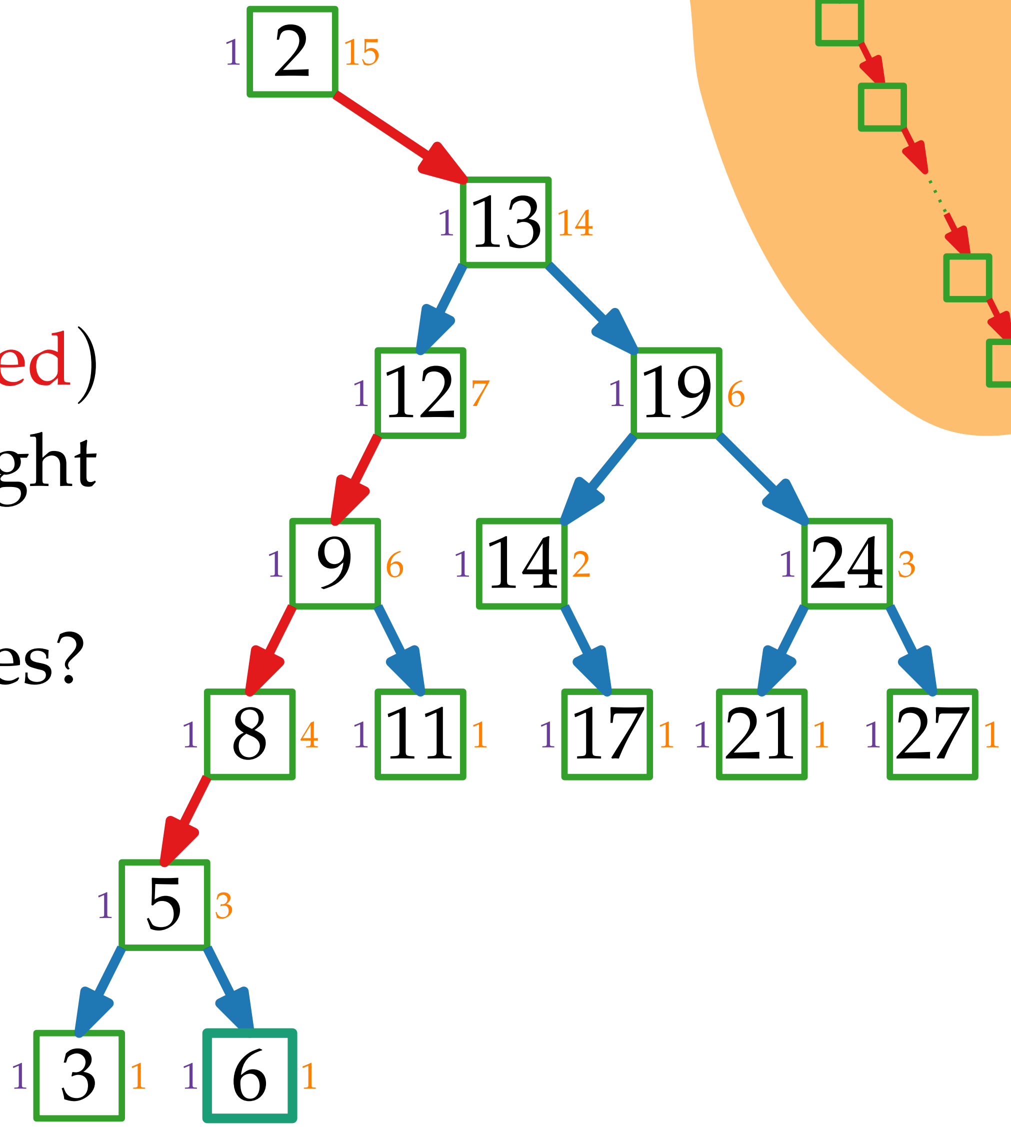
$\Phi = \sum \log s(x)$

Amortized cost:

real cost + $\Phi_+ - \Phi$

(potential after splay)





Why is Splay Fast?

$w(x)$: weight of x (here 1), $W = \sum w(x)$ (here n)

$s(x)$: sum of all $w(x)$ in subtree of x_i

mark edges:

→ $s(\text{child}) \leq s(\text{parent})/2$

→ $s(\text{child}) > s(\text{parent})/2$

Cost to query x_i: $O(\log W + \#\text{red})$

Idea: blue edges halve the weight

$\Rightarrow \#\text{blue} \in O(\log W)$

How can we amortize red edges?

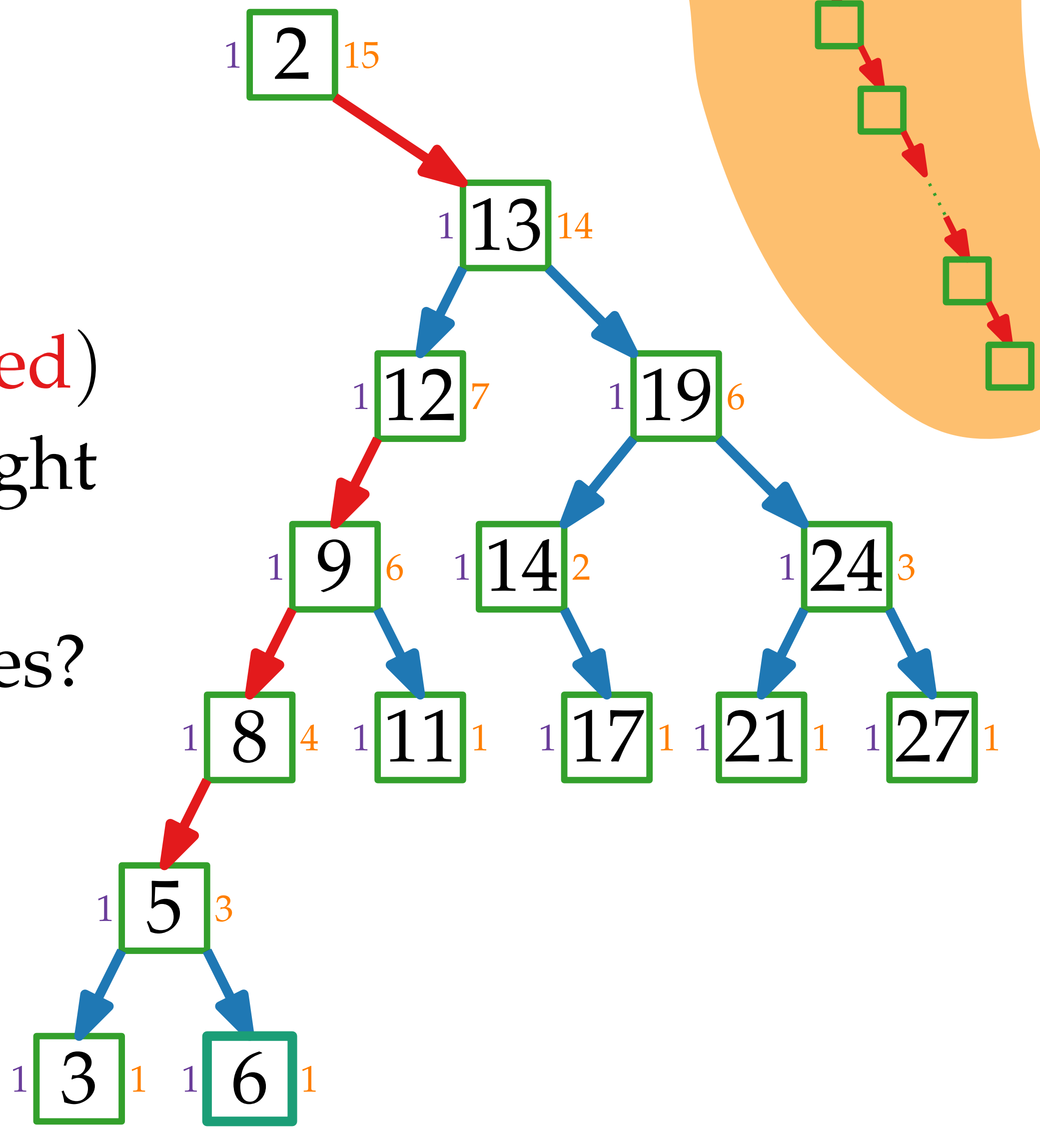
Use sum-of-logs potential

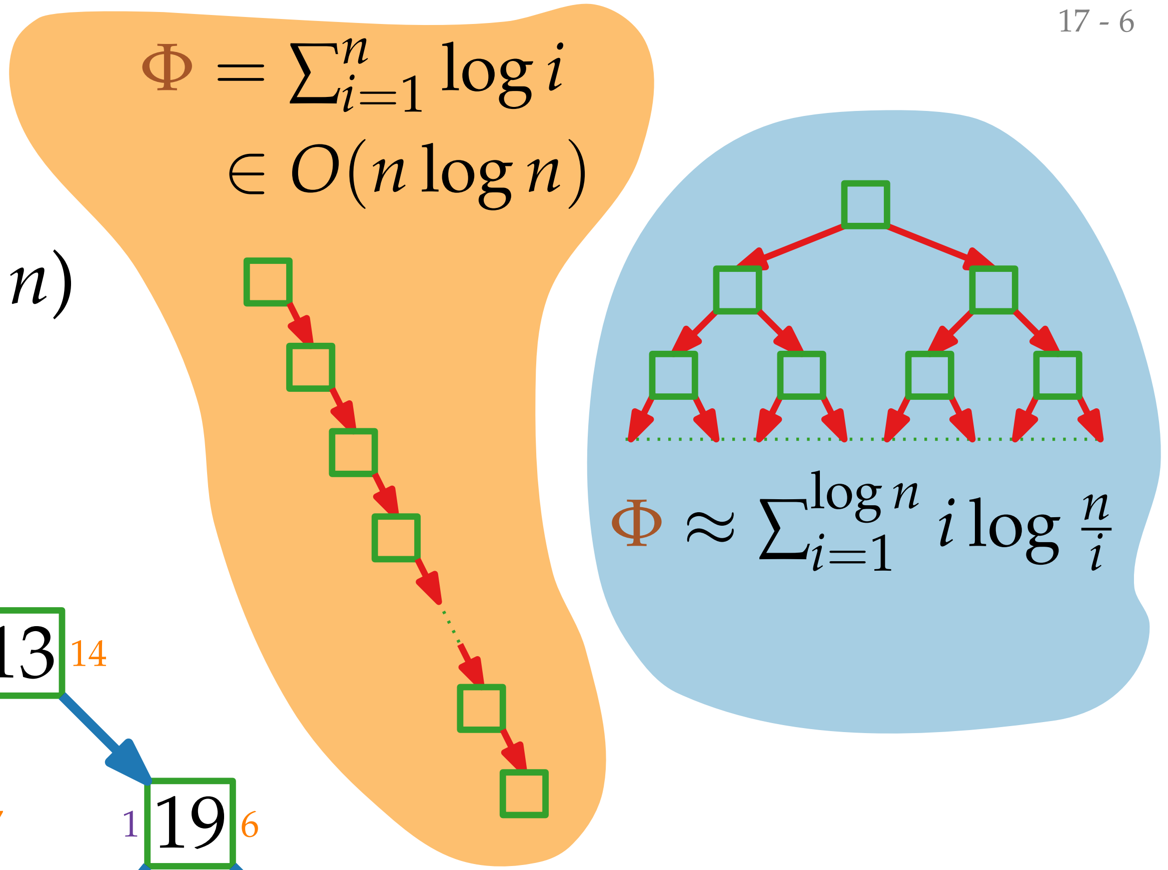
$\Phi = \sum \log s(x)$

Amortized cost:

real cost + $\Phi_+ - \Phi$

(potential after splay)





Why is Splay Fast?

$w(x)$: weight of x (here 1), $W = \sum w(x)$ (here n)

$s(x)$: sum of all $w(x)$ in subtree of x_i

mark edges:

→ $s(\text{child}) \leq s(\text{parent})/2$

→ $s(\text{child}) > s(\text{parent})/2$

Cost to query x_i: $O(\log W + \#\text{red})$

Idea: blue edges halve the weight

$\Rightarrow \#\text{blue} \in O(\log W)$

How can we amortize red edges?

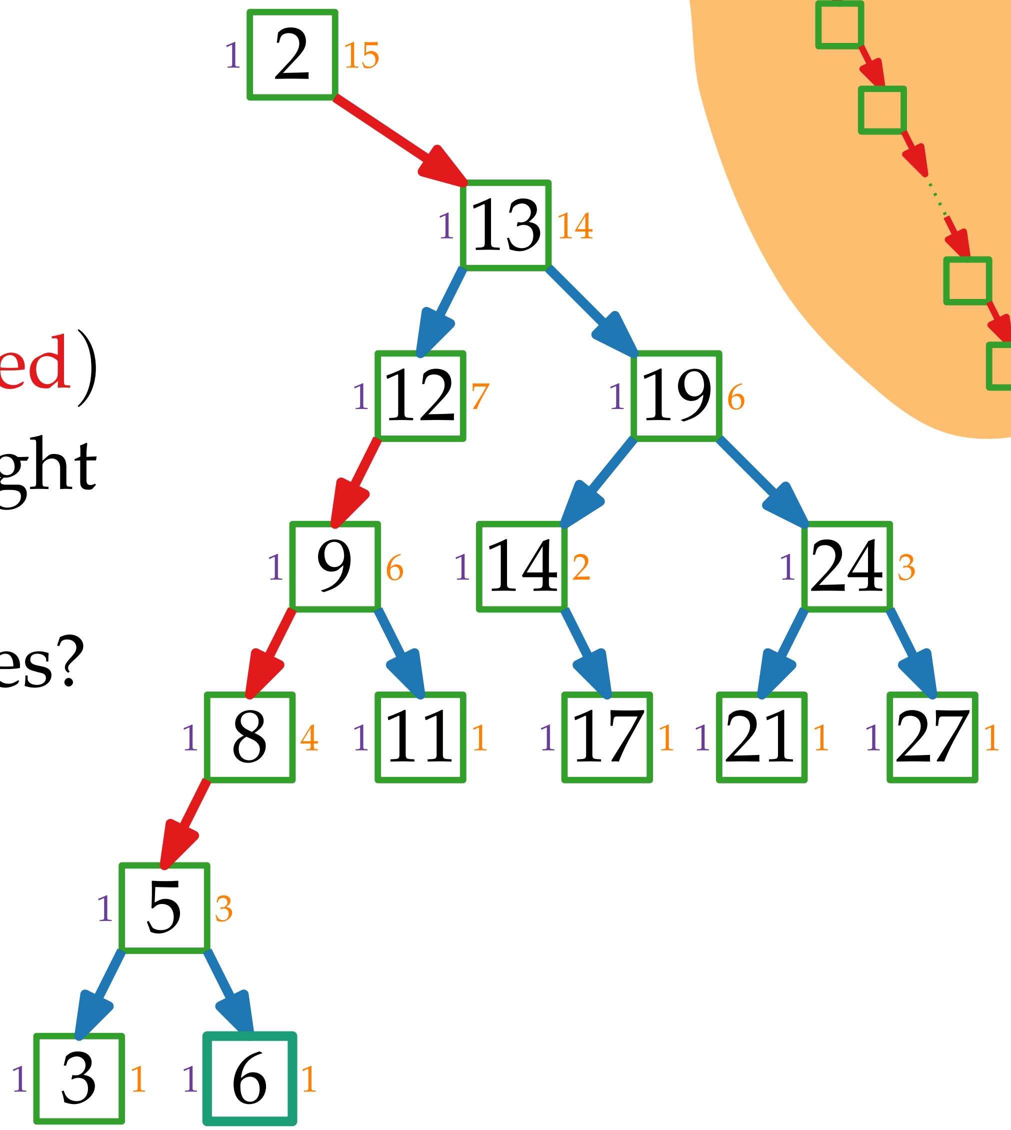
Use sum-of-logs potential

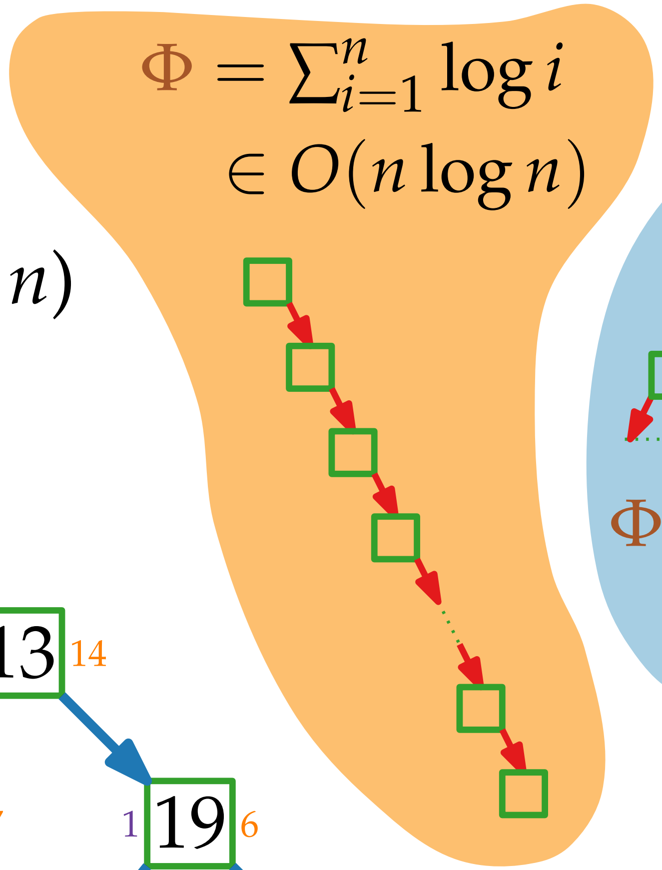
$\Phi = \sum \log s(x)$

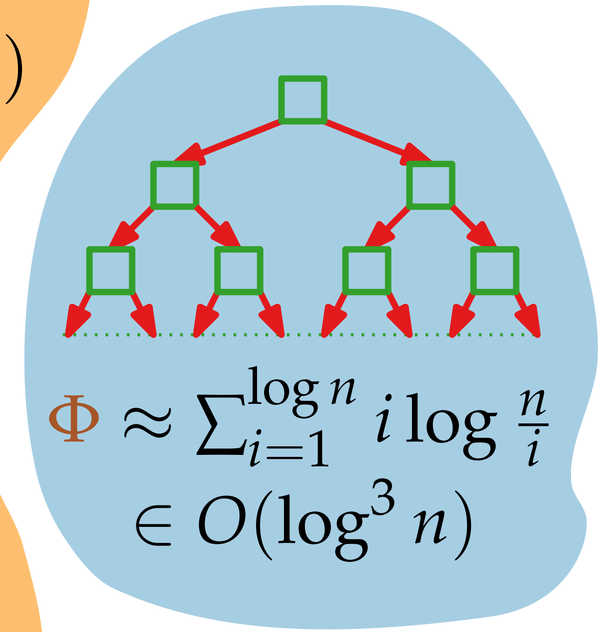
Amortized cost:

real cost + $\Phi_+ - \Phi$

(potential after splay)





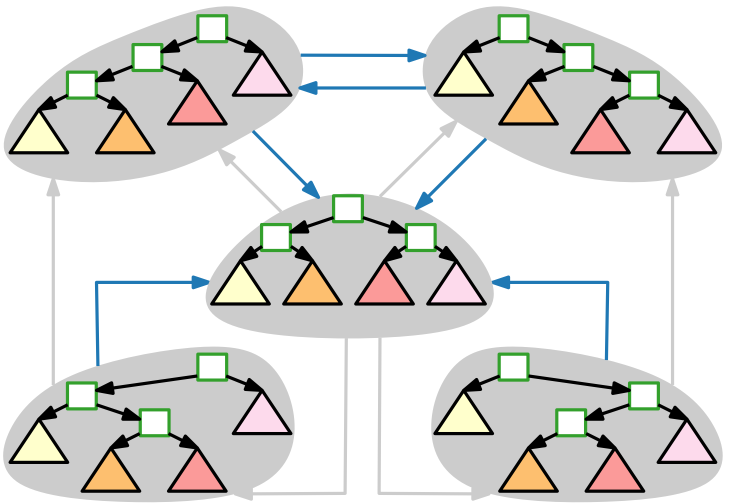


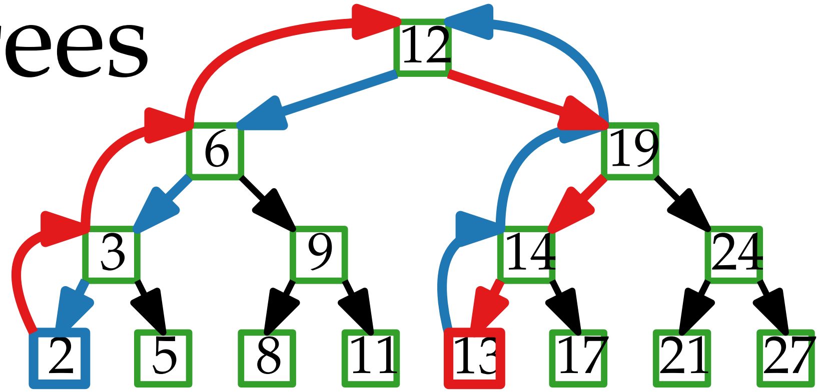
Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Advanced Algorithms

Optimal Binary Search Trees

Splay Trees





Philipp Kindermann · WS20

Part V:
Access Lemma and Running Time of Splay

Potential after Rotation

Consider any rotation; $s(x)$ before rotation, $s_+(x)$ afterwards

Potential after Rotation

Consider any rotation; $s(x)$ before rotation, $s_+(x)$ afterwards

Lemma. After a single rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x))$.

Potential after Rotation

Consider any rotation; $s(x)$ before rotation, $s_+(x)$ afterwards

Lemma. After a single rotation, the potential increases by $\leq 3\left(\log s_+(x) - \log s(x)\right)$.

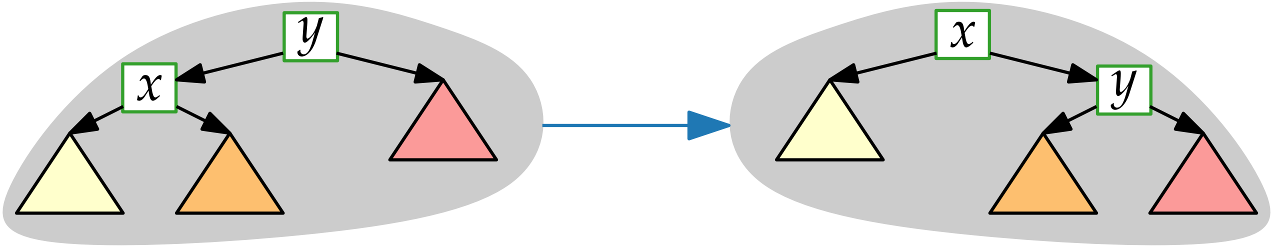
Proof. $\mathrm{Right}(x)$

Potential after Rotation

Consider any rotation; $s(x)$ before rotation, $s_+(x)$ afterwards

Lemma. After a single rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x))$.

Proof. Right(x)

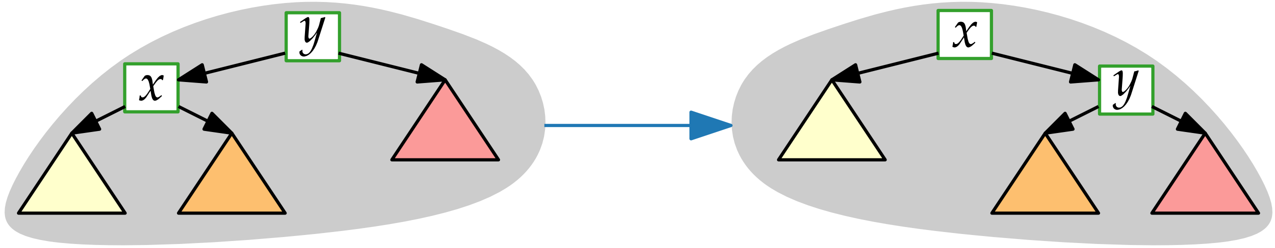


Potential after Rotation

Consider any rotation; $s(x)$ before rotation, $s_+(x)$ afterwards

Lemma. After a single rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x))$.

Proof. Right(x)



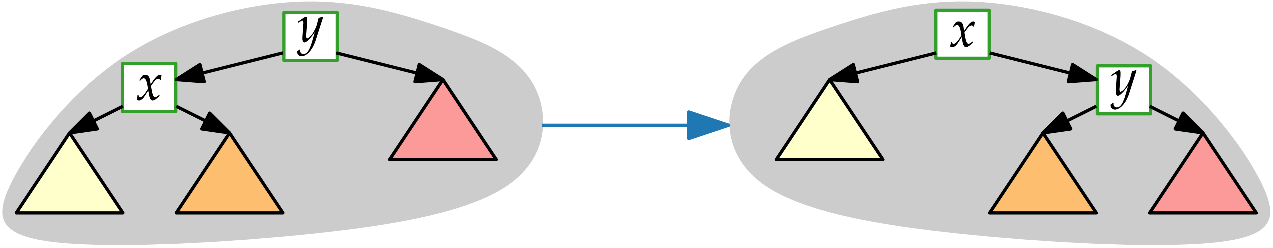
Observe: Only $s(x)$ and $s(y)$ change.

Potential after Rotation

Consider any rotation; $s(x)$ before rotation, $s_+(x)$ afterwards

Lemma. After a single rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x))$.

Proof. Right(x)



Observe: Only $s(x)$ and $s(y)$ change.

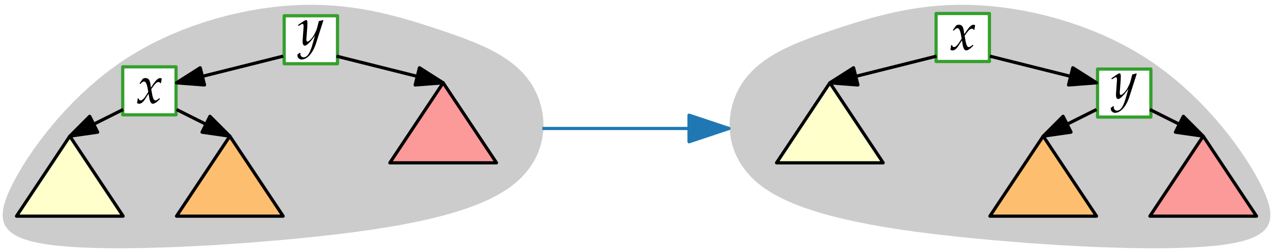
$$\text{pot. change} = \log s_+(x) + \log s_+(y) - \log s(x) - \log s(y)$$

Potential after Rotation

Consider any rotation; $s(x)$ before rotation, $s_+(x)$ afterwards

Lemma. After a single rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x))$.

Proof. Right(x)



Observe: Only $s(x)$ and $s(y)$ change.

$$\text{pot. change} \quad = \log s_+(x) + \log s_+(y) - \log s(x) - \log s(y)$$

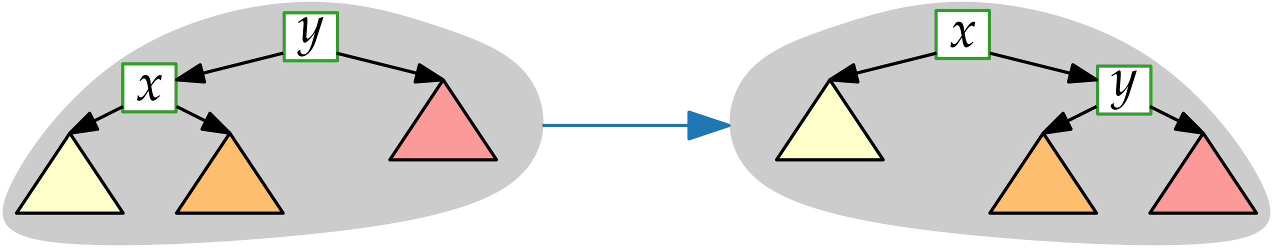
$(s_+(y) \leq s(y))$

Potential after Rotation

Consider any rotation; $s(x)$ before rotation, $s_+(x)$ afterwards

Lemma. After a single rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x))$.

Proof. $\mathrm{Right}(x)$



Observe: Only $s(x)$ and $s(y)$ change.

$$\text{pot. change} = \log s_+(x) + \log s_+(y) - \log s(x) - \log s(y)$$

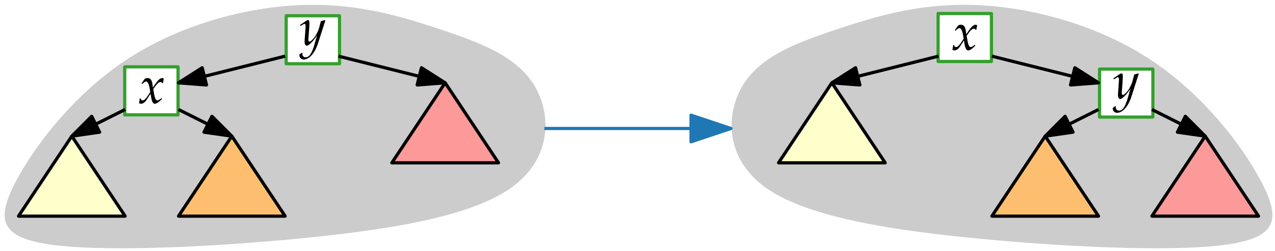
$$(s_+(y) \leq s(y)) \quad \leq \log s_+(x) - \log s(x)$$

Potential after Rotation

Consider any rotation; $s(x)$ before rotation, $s_+(x)$ afterwards

Lemma. After a single rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x))$.

Proof. Right(x)



Observe: Only $s(x)$ and $s(y)$ change.

pot. change $= \log s_+(x) + \log s_+(y) - \log s(x) - \log s(y)$

$(s_+(y) \leq s(y))$ $\leq \log s_+(x) - \log s(x)$

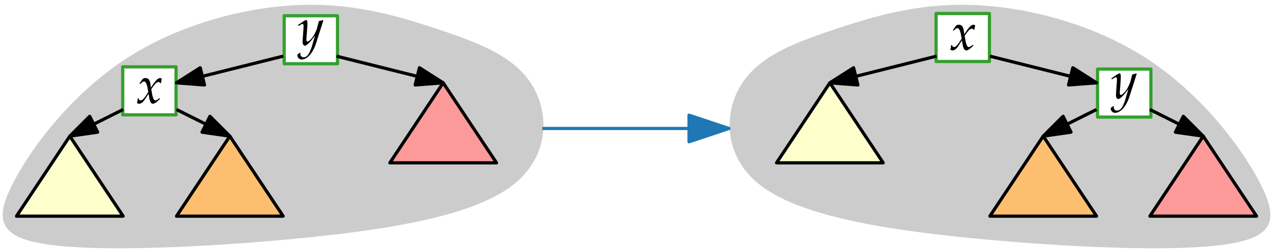
$(s_+(x) > s(x))$

Potential after Rotation

Consider any rotation; $s(x)$ before rotation, $s_+(x)$ afterwards

Lemma. After a single rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x))$.

Proof. Right(x)



Observe: Only $s(x)$ and $s(y)$ change.

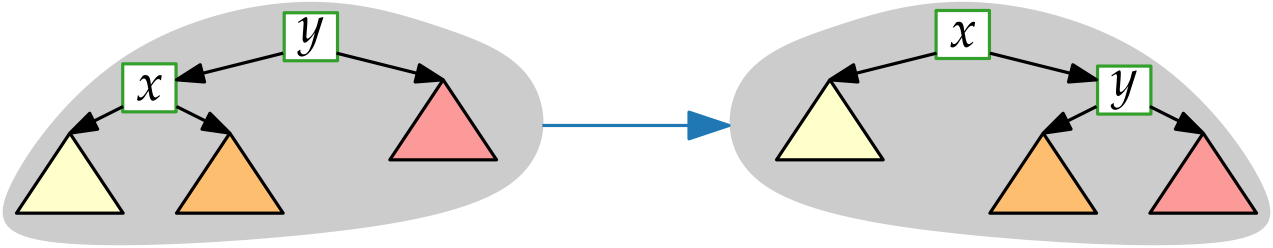
$$\begin{aligned}
\text{pot. change} \quad &= \log s_+(x) + \log s_+(y) \\
&\quad - \log s(x) - \log s(y) \\
(s_+(y) \leq s(y)) \quad &\leq \log s_+(x) - \log s(x) \\
(s_+(x) > s(x)) \quad &\leq 3\,(\log s_+(x) - \log s(x))
\end{aligned}$$

Potential after Rotation

Consider any rotation; $s(x)$ before rotation, $s_+(x)$ afterwards

Lemma. After a single rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x))$.

Proof. Right(x)



Observe: Only $s(x)$ and $s(y)$ change.

pot. change $= \log s_+(x) + \log s_+(y) - \log s(x) - \log s(y)$

$(s_+(y) \leq s(y))$ $\leq \log s_+(x) - \log s(x)$

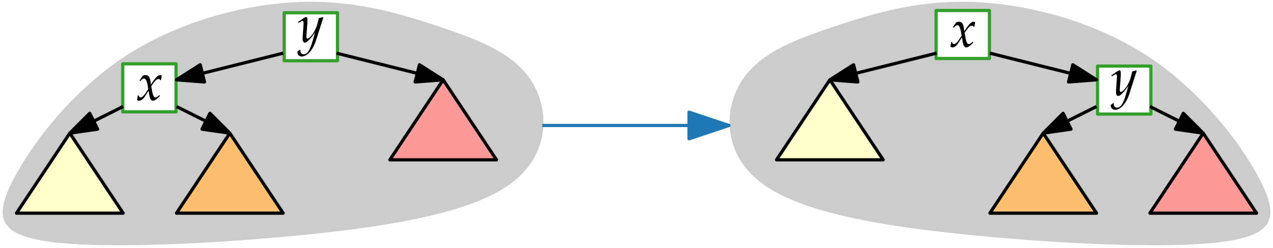
$(s_+(x) > s(x))$ $\leq 3\,(\log s_+(x) - \log s(x))$ ✓

Potential after Rotation

Consider any rotation; $s(x)$ before rotation, $s_+(x)$ afterwards

Lemma. After a single rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x))$.

Proof. $\mathrm{Right}(x)$



Observe: Only $s(x)$ and $s(y)$ change.

$$\begin{aligned}
\text{pot. change} \quad &= \log s_+(x) + \log s_+(y) \\
&\quad - \log s(x) - \log s(y) \\
(s_+(y) \leq s(y)) \quad &\leq \log s_+(x) - \log s(x) \\
(s_+(x) > s(x)) \quad &\leq 3\,(\log s_+(x) - \log s(x)) \quad \checkmark
\end{aligned}$$

$\mathrm{Left}(x)$ analogue $\checkmark$

Potential after Rotation

Consider any rotation; $s(x)$ before rotation, $s_+(x)$ afterwards

Lemma. After a double rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x)) - 2$.

Potential after Rotation

Consider any rotation; $s(x)$ before rotation, $s_+(x)$ afterwards

Lemma. After a double rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x)) - 2$.

Proof.

Potential after Rotation

Consider any rotation; $s(x)$ before rotation, $s_+(x)$ afterwards

Lemma. After a double rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x)) - 2$.

Proof.

Case 1. Right-Right(x)

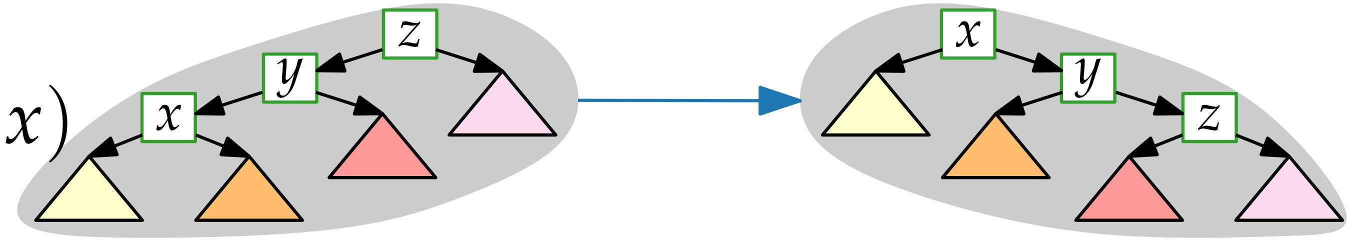
Potential after Rotation

Consider any rotation; $s(x)$ before rotation, $s_+(x)$ afterwards

Lemma. After a double rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x)) - 2$.

Proof.

Case 1. Right-Right(x)



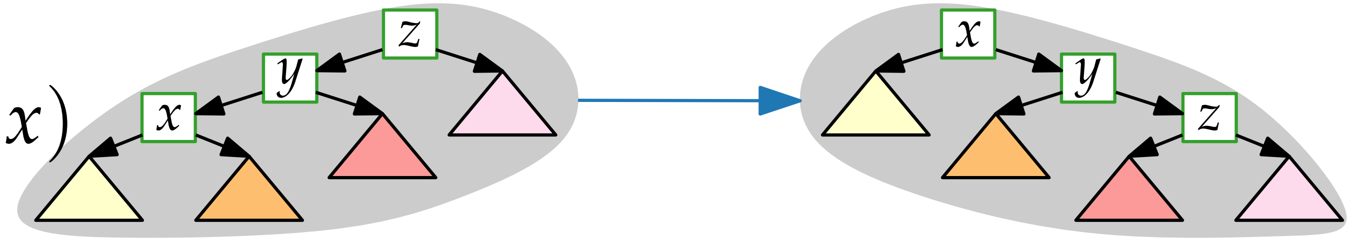
Potential after Rotation

Consider any rotation; $s(x)$ before rotation, $s_+(x)$ afterwards

Lemma. After a double rotation, the potential increases by $\leq 3(\log s_+(x) - \log s(x)) - 2$.

Proof.

Case 1. Right-Right(x)



$$\begin{aligned}\text{pot. change} \quad &= \log s_+(x) + \log s_+(y) + \log s_+(z) \\ &\quad - \log s(x) - \log s(y) - \log s(z)\end{aligned}$$

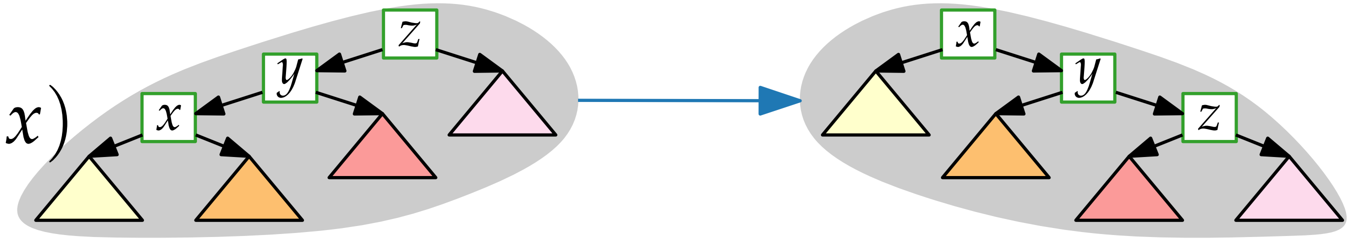
Potential after Rotation

Consider any rotation; $s(x)$ before rotation, $s_+(x)$ afterwards

Lemma. After a double rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x)) - 2$.

Proof.

Case 1. Right-Right(x)



pot. change $= \log s_+(x) + \log s_+(y) + \log s_+(z)$
$- \log s(x) - \log s(y) - \log s(z)$

$(s_+(x) = s(z))$

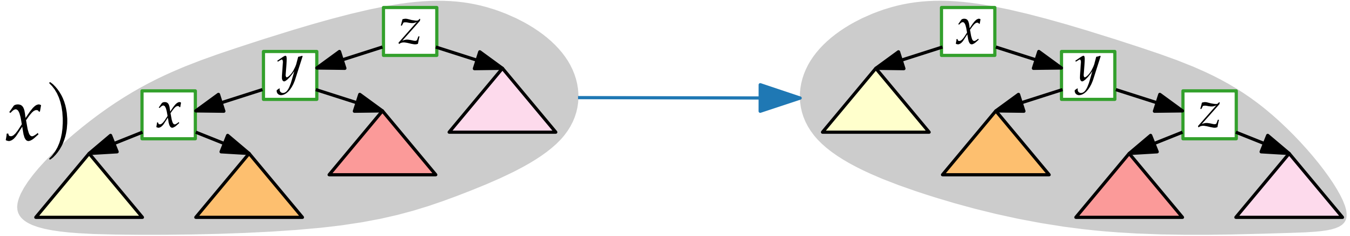
Potential after Rotation

Consider any rotation; $s(x)$ before rotation, $s_+(x)$ afterwards

Lemma. After a double rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x)) - 2$.

Proof.

Case 1. Right-Right(x)



$$
\begin{aligned}
\text{pot. change} \quad &= \log s_+(x) + \log s_+(y) + \log s_+(z) \\
&\quad - \log s(x) - \log s(y) - \log s(z) \\
(s_+(x) = s(z)) \quad &= \log s_+(y) + \log s_+(z) - \log s(x) - \log s(y)
\end{aligned}
$$

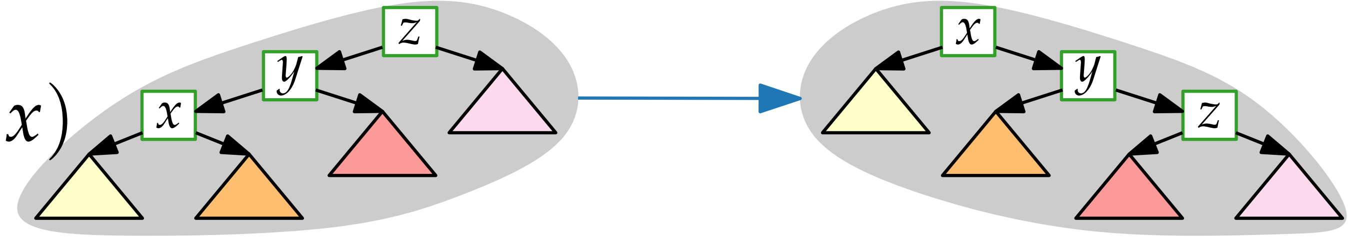
Potential after Rotation

Consider any rotation; $s(x)$ before rotation, $s_+(x)$ afterwards

Lemma. After a double rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x)) - 2$.

Proof.

Case 1. Right-Right(x)



$$\text{pot. change} = \log s_+(x) + \log s_+(y) + \log s_+(z) - \log s(x) - \log s(y) - \log s(z)$$

$$(s_+(x) = s(z)) \quad = \log s_+(y) + \log s_+(z) - \log s(x) - \log s(y)$$

$(s(x) \leq s(y))$

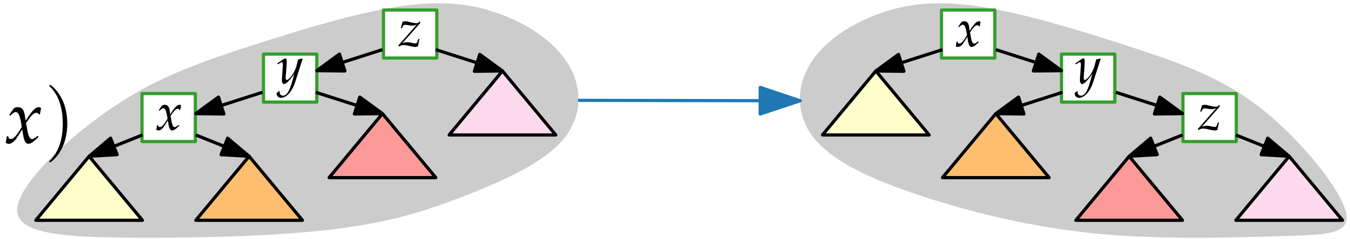
Potential after Rotation

Consider any rotation; $s(x)$ before rotation, $s_+(x)$ afterwards

Lemma. After a double rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x)) - 2$.

Proof.

Case 1. Right-Right(x)



$$
\begin{aligned}
\text{pot. change} \quad &= \log s_+(x) + \log s_+(y) + \log s_+(z) \\
&\quad - \log s(x) - \log s(y) - \log s(z) \\
(s_+(x) = s(z)) \quad &= \log s_+(y) + \log s_+(z) - \log s(x) - \log s(y) \\
(s(x) \leq s(y)) \quad &\leq \log s_+(y) + \log s_+(z) - 2\log s(x)
\end{aligned}
$$

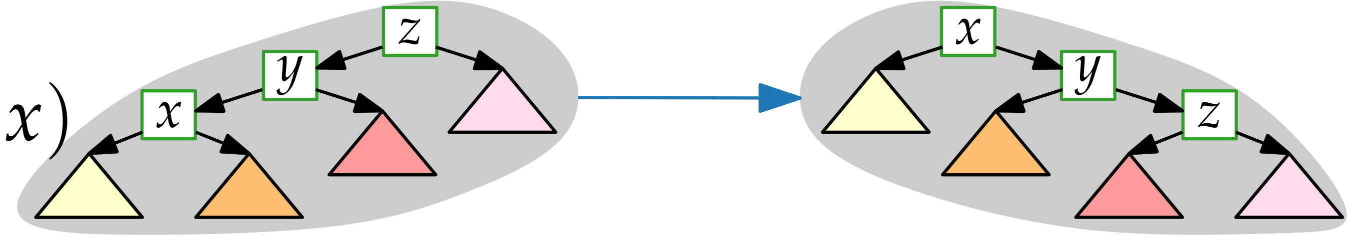
Potential after Rotation

Consider any rotation; $s(x)$ before rotation, $s_+(x)$ afterwards

Lemma. After a double rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x)) - 2$.

Proof.

Case 1. Right-Right(x)



$$\begin{aligned}
\text{pot. change} \quad &= \log s_+(x) + \log s_+(y) + \log s_+(z) \\
&\quad - \log s(x) - \log s(y) - \log s(z) \\
(s_+(x) = s(z)) \quad &= \log s_+(y) + \log s_+(z) - \log s(x) - \log s(y) \\
(s(x) \leq s(y)) \quad &\leq \log s_+(y) + \log s_+(z) - 2\log s(x) \\
(s_+(y) \leq s_+(x)) \quad &
\end{aligned}$$

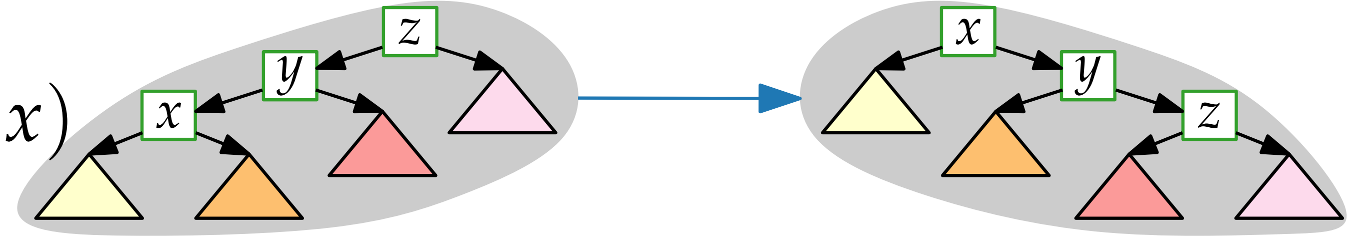
Potential after Rotation

Consider any rotation; $s(x)$ before rotation, $s_+(x)$ afterwards

Lemma. After a double rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x)) - 2$.

Proof.

Case 1. Right-Right(x)



$$
\begin{aligned}
\text{pot. change} &= \log s_+(x) + \log s_+(y) + \log s_+(z) \\
&\quad - \log s(x) - \log s(y) - \log s(z) \\
(s_+(x) = s(z)) \quad &= \log s_+(y) + \log s_+(z) - \log s(x) - \log s(y) \\
(s(x) \leq s(y)) \quad &\leq \log s_+(y) + \log s_+(z) - 2\log s(x) \\
(s_+(y) \leq s_+(x)) \quad &\leq \log s_+(x) + \log s_+(z) - 2\log s(x)
\end{aligned}
$$

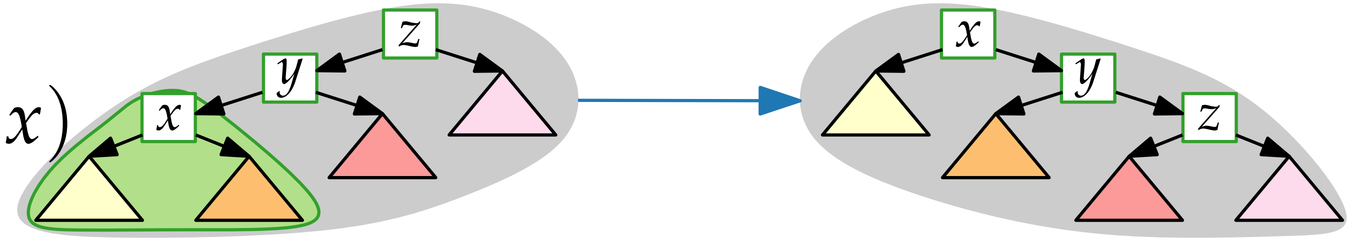
Potential after Rotation

Consider any rotation; $s(x)$ before rotation, $s_+(x)$ afterwards

Lemma. After a double rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x)) - 2$.

Proof.

Case 1. Right-Right(x)



$$
\begin{aligned}
\text{pot. change} \quad &= \log s_+(x) + \log s_+(y) + \log s_+(z) \\
&\quad - \log s(x) - \log s(y) - \log s(z) \\
(s_+(x) = s(z)) \quad &= \log s_+(y) + \log s_+(z) - \log s(x) - \log s(y) \\
(s(x) \leq s(y)) \quad &\leq \log s_+(y) + \log s_+(z) - 2\log s(x) \\
(s_+(y) \leq s_+(x)) \quad &\leq \log s_+(x) + \log s_+(z) - 2\log s(x)
\end{aligned}
$$

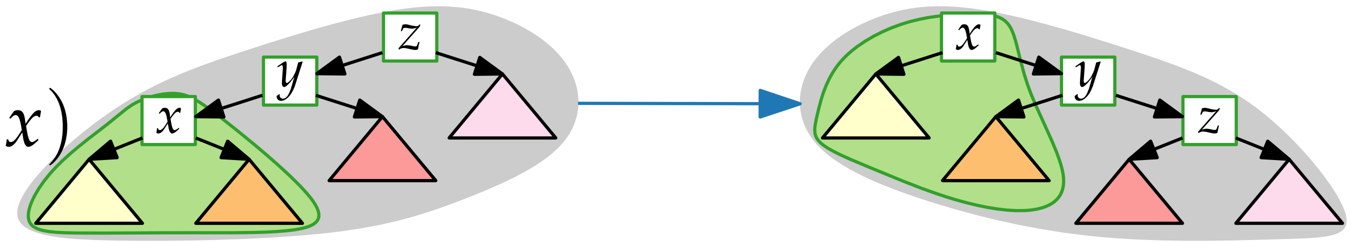
Potential after Rotation

Consider any rotation; $s(x)$ before rotation, $s_+(x)$ afterwards

Lemma. After a double rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x)) - 2$.

Proof.

Case 1. Right-Right(x)



pot. change $= \log s_+(x) + \log s_+(y) + \log s_+(z) - \log s(x) - \log s(y) - \log s(z)$

$(s_+(x) = s(z))$ $= \log s_+(y) + \log s_+(z) - \log s(x) - \log s(y)$

$(s(x) \leq s(y))$ $\leq \log s_+(y) + \log s_+(z) - 2\log s(x)$

$(s_+(y) \leq s_+(x))$ $\leq \log s_+(x) + \log s_+(z) - 2\log s(x)$

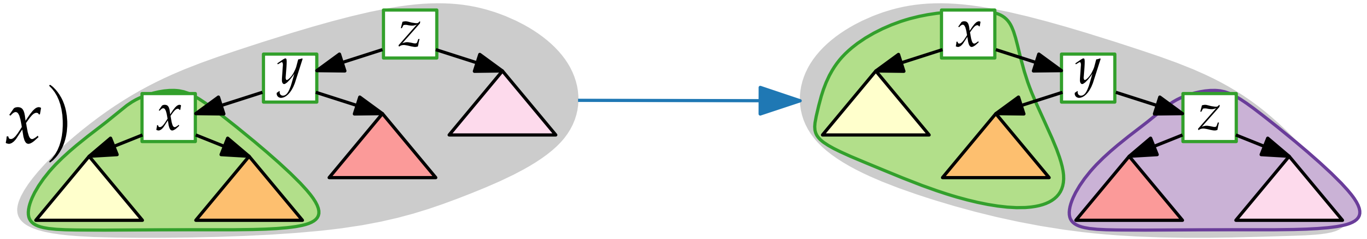
Potential after Rotation

Consider any rotation; $s(x)$ before rotation, $s_+(x)$ afterwards

Lemma. After a double rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x)) - 2$.

Proof.

Case 1. Right-Right(x)



$$
\begin{aligned}
\text{pot. change} \quad &= \log s_+(x) + \log s_+(y) + \log s_+(z) \\
&\quad - \log s(x) - \log s(y) - \log s(z) \\
(s_+(x) = s(z)) \quad &= \log s_+(y) + \log s_+(z) - \log s(x) - \log s(y) \\
(s(x) \leq s(y)) \quad &\leq \log s_+(y) + \log s_+(z) - 2\log s(x) \\
(s_+(y) \leq s_+(x)) \quad &\leq \log s_+(x) + \log s_+(z) - 2\log s(x)
\end{aligned}
$$

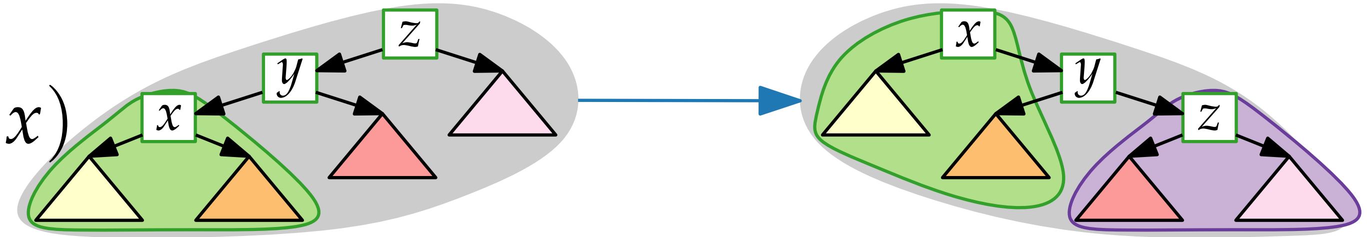
Potential after Rotation

Consider any rotation; $s(x)$ before rotation, $s_+(x)$ afterwards

Lemma. After a double rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x)) - 2$.

Proof.

Case 1. Right-Right(x)



$$
\begin{aligned}
\text{pot. change} \quad &= \log s_+(x) + \log s_+(y) + \log s_+(z) \\
&\quad - \log s(x) - \log s(y) - \log s(z) \\
(s_+(x) = s(z)) \quad &= \log s_+(y) + \log s_+(z) - \log s(x) - \log s(y) \\
(s(x) \leq s(y)) \quad &\leq \log s_+(y) + \log s_+(z) - 2\log s(x) \\
(s_+(y) \leq s_+(x)) \quad &\leq \log s_+(x) + \log s_+(z) - 2\log s(x)
\end{aligned}
$$

$s(x) + s_+(z) \leq s_+(x)$

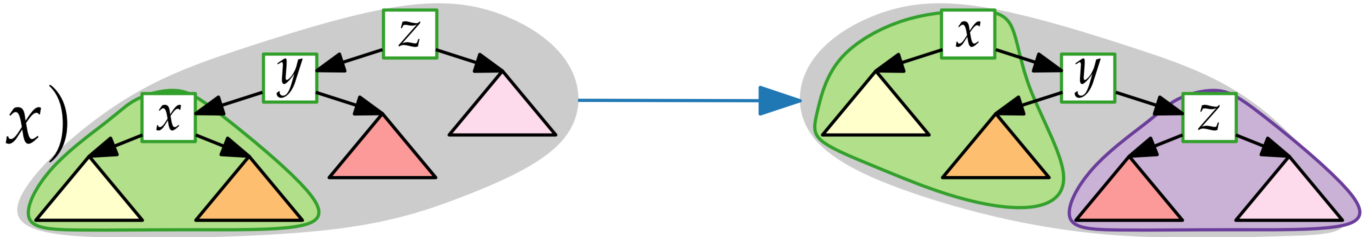
Potential after Rotation

Consider any rotation; $s(x)$ before rotation, $s_+(x)$ afterwards

Lemma. After a double rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x)) - 2$.

Proof.

Case 1. Right-Right(x)



$$
\begin{aligned}
\text{pot. change} \quad &= \log s_+(x) + \log s_+(y) + \log s_+(z) \\
&\quad - \log s(x) - \log s(y) - \log s(z) \\
(s_+(x) = s(z)) \quad &= \log s_+(y) + \log s_+(z) - \log s(x) - \log s(y) \\
(s(x) \leq s(y)) \quad &\leq \log s_+(y) + \log s_+(z) - 2\log s(x) \\
(s_+(y) \leq s_+(x)) \quad &\leq \log s_+(x) + \log s_+(z) - 2\log s(x)
\end{aligned}
$$

$s(x) + s_+(z) \leq s_+(x) \Rightarrow \log s(x) + \log s_+(z)$

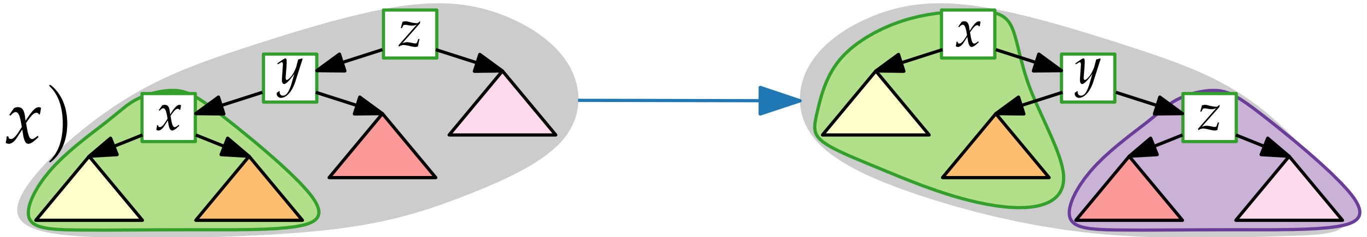
Potential after Rotation

Consider any rotation; $s(x)$ before rotation, $s_+(x)$ afterwards

Lemma. After a double rotation, the potential increases by $\leq 3\left(\log s_+(x) - \log s(x)\right) - 2$.

Proof.

Case 1. Right-Right(x)



$$\begin{aligned}
\text{pot. change} \quad &= \log s_+(x) + \log s_+(y) + \log s_+(z) \\
&\quad - \log s(x) - \log s(y) - \log s(z) \\
(s_+(x) = s(z)) \quad &= \log s_+(y) + \log s_+(z) - \log s(x) - \log s(y) \\
(s(x) \leq s(y)) \quad &\leq \log s_+(y) + \log s_+(z) - 2\log s(x) \\
(s_+(y) \leq s_+(x)) \quad &\leq \log s_+(x) + \log s_+(z) - 2\log s(x)
\end{aligned}$$

$$s(x) + s_+(z) \leq s_+(x) \Rightarrow \log s(x) + \log s_+(z) = \log s(x)s_+(z)$$

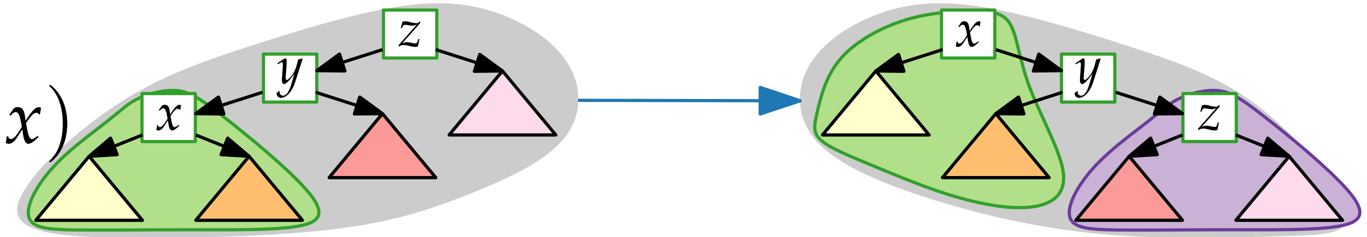
Potential after Rotation

Consider any rotation; $s(x)$ before rotation, $s_+(x)$ afterwards

Lemma. After a double rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x)) - 2$.

Proof.

Case 1. Right-Right(x)



$$\begin{aligned}
\text{pot. change} \quad &= \log s_+(x) + \log s_+(y) + \log s_+(z) \\
&\quad - \log s(x) - \log s(y) - \log s(z) \\
(s_+(x) = s(z)) \quad &= \log s_+(y) + \log s_+(z) - \log s(x) - \log s(y) \\
(s(x) \leq s(y)) \quad &\leq \log s_+(y) + \log s_+(z) - 2\log s(x) \\
(s_+(y) \leq s_+(x)) \quad &\leq \log s_+(x) + \log s_+(z) - 2\log s(x)
\end{aligned}$$

$$s(x) + s_+(z) \leq s_+(x) \Rightarrow \log s(x) + \log s_+(z) = \log s(x)s_+(z) \underset{\text{(AM-GM)}}{\leq} \log\,(s_+(x)/2)^2$$

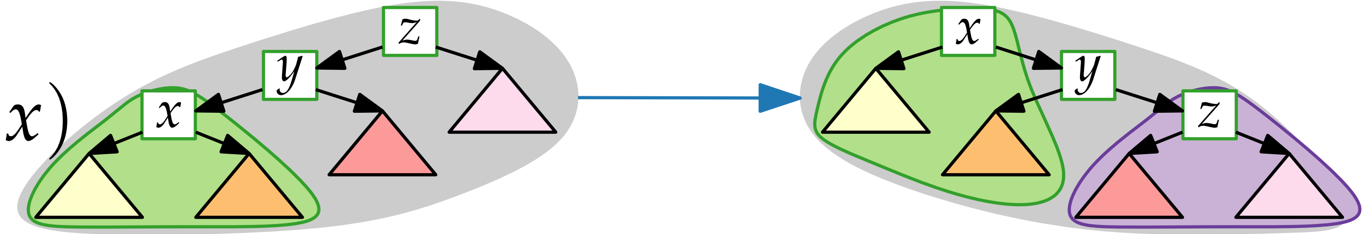
Potential after Rotation

Consider any rotation; $s(x)$ before rotation, $s_+(x)$ afterwards

Lemma. After a double rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x)) - 2$.

Proof.

Case 1. Right-Right(x)



$$
\begin{aligned}
\text{pot. change} \quad & = \log s_+(x) + \log s_+(y) + \log s_+(z) \\
& \quad - \log s(x) - \log s(y) - \log s(z) \\
(s_+(x) = s(z)) \quad & = \log s_+(y) + \log s_+(z) - \log s(x) - \log s(y) \\
(s(x) \leq s(y)) \quad & \leq \log s_+(y) + \log s_+(z) - 2\log s(x) \\
(s_+(y) \leq s_+(x)) \quad & \leq \log s_+(x) + \log s_+(z) - 2\log s(x)
\end{aligned}
$$

$$
s(x) + s_+(z) \leq s_+(x) \Rightarrow \log s(x) + \log s_+(z) = \log s(x) s_+(z)
$$
$$
\underset{\text{(AM-GM)}}{\leq} \log\left(s_+(x)/2\right)^2 \leq 2\log s_+(x) - 2
$$

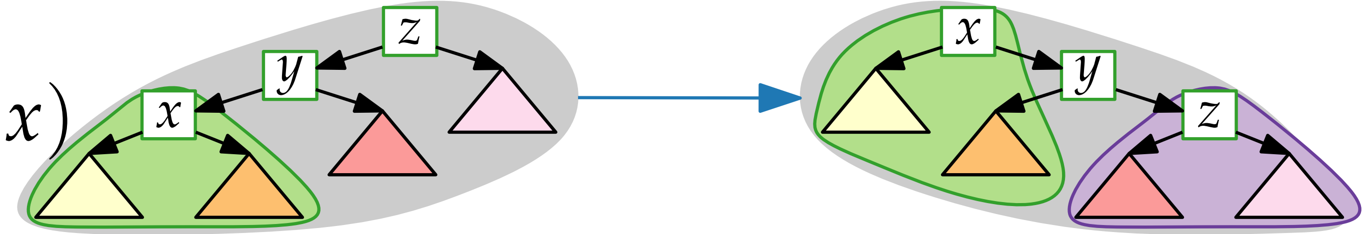
Potential after Rotation

Consider any rotation; $s(x)$ before rotation, $s_+(x)$ afterwards

Lemma. After a double rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x)) - 2$.

Proof.

Case 1. Right-Right(x)



$$\begin{aligned}
\text{pot. change} \quad &= \log s_+(x) + \log s_+(y) + \log s_+(z) \\
&\quad - \log s(x) - \log s(y) - \log s(z) \\
(s_+(x) = s(z)) \quad &= \log s_+(y) + \log s_+(z) - \log s(x) - \log s(y) \\
(s(x) \leq s(y)) \quad &\leq \log s_+(y) + \log s_+(z) - 2\log s(x) \\
(s_+(y) \leq s_+(x)) \quad &\leq \log s_+(x) + \log s_+(z) - 2\log s(x)
\end{aligned}$$

$$s(x) + s_+(z) \leq s_+(x) \Rightarrow \log s(x) + \log s_+(z) = \log s(x)s_+(z)$$
$$\underset{\text{(AM-GM)}}{\leq} \log\,(s_+(x)/2)^2 \leq 2\log s_+(x) - 2$$

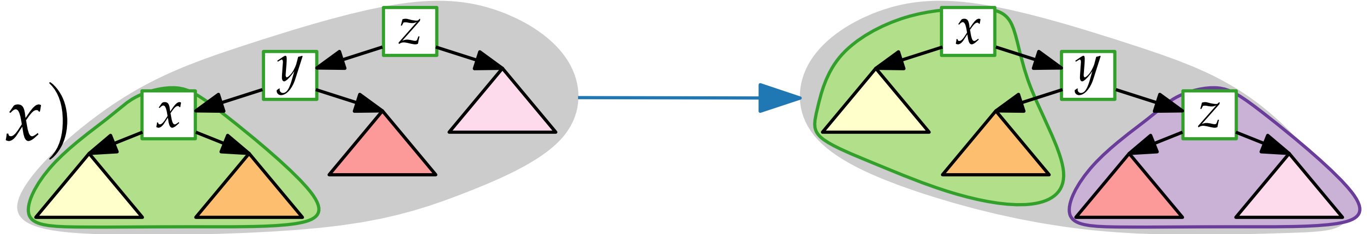
Potential after Rotation

Consider any rotation; $s(x)$ before rotation, $s_+(x)$ afterwards

Lemma. After a double rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x)) - 2$.

Proof.

Case 1. Right-Right(x)



$$\begin{aligned}
\text{pot. change} \quad &= \log s_+(x) + \log s_+(y) + \log s_+(z) \\
&\quad - \log s(x) - \log s(y) - \log s(z) \\
(s_+(x) = s(z)) \quad &= \log s_+(y) + \log s_+(z) - \log s(x) - \log s(y) \\
(s(x) \leq s(y)) \quad &\leq \log s_+(y) + \log s_+(z) - 2\log s(x) \\
(s_+(y) \leq s_+(x)) \quad &\leq \log s_+(x) + \log s_+(z) - 2\log s(x) \\
&\leq 3\log s_+(x) - 3\log s(x) - 2
\end{aligned}$$

$$s(x) + s_+(z) \leq s_+(x) \Rightarrow \log s(x) + \log s_+(z) = \log s(x)s_+(z)$$

$$\underset{\text{(AM-GM)}}{\leq} \log\,(s_+(x)/2)^2 \leq 2\log s_+(x) - 2$$

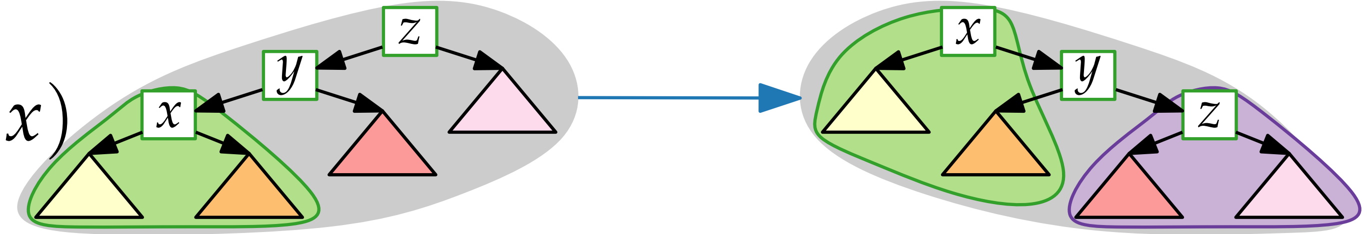
Potential after Rotation

Consider any rotation; $s(x)$ before rotation, $s_+(x)$ afterwards

Lemma. After a double rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x)) - 2$.

Proof.

Case 1. Right-Right(x)



$$
\begin{aligned}
\text{pot. change} \quad & = \log s_+(x) + \log s_+(y) + \log s_+(z) \\
& \quad - \log s(x) - \log s(y) - \log s(z) \\
(s_+(x) = s(z)) \quad & = \log s_+(y) + \log s_+(z) - \log s(x) - \log s(y) \\
(s(x) \leq s(y)) \quad & \leq \log s_+(y) + \log s_+(z) - 2\log s(x) \\
(s_+(y) \leq s_+(x)) \quad & \leq \log s_+(x) + \log s_+(z) - 2\log s(x) \\
& \leq 3\log s_+(x) - 3\log s(x) - 2 \quad \checkmark
\end{aligned}
$$

$$
s(x) + s_+(z) \leq s_+(x) \Rightarrow \log s(x) + \log s_+(z) = \log s(x)s_+(z)
$$

$$
\underset{\text{(AM-GM)}}{\leq} \log\left(s_+(x)/2\right)^2 \leq 2\log s_+(x) - 2
$$

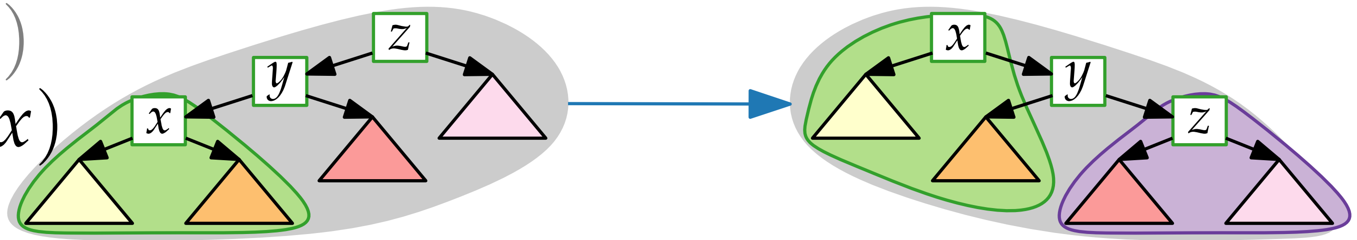
Potential after Rotation

Consider any rotation; $s(x)$ before rotation, $s_+(x)$ afterwards

Lemma. After a double rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x)) - 2$.

Proof. / Left-Left(x)

Case 1. Right-Right(x)



$$\begin{aligned}
\text{pot. change} \quad &= \log s_+(x) + \log s_+(y) + \log s_+(z) \\
&\quad - \log s(x) - \log s(y) - \log s(z) \\
(s_+(x) = s(z)) \quad &= \log s_+(y) + \log s_+(z) - \log s(x) - \log s(y) \\
(s(x) \leq s(y)) \quad &\leq \log s_+(y) + \log s_+(z) - 2\log s(x) \\
(s_+(y) \leq s_+(x)) \quad &\leq \log s_+(x) + \log s_+(z) - 2\log s(x) \\
&\leq 3\log s_+(x) - 3\log s(x) - 2 \quad \checkmark
\end{aligned}$$

$$s(x) + s_+(z) \leq s_+(x) \Rightarrow \log s(x) + \log s_+(z) = \log s(x)s_+(z)$$
$$\underset{\text{(AM-GM)}}{\leq} \log\,(s_+(x)/2)^2 \leq 2\log s_+(x) - 2$$

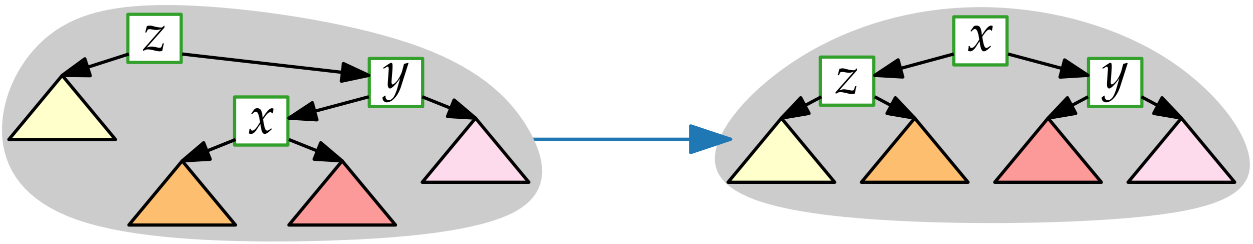
Potential after Rotation

Consider any rotation; $s(x)$ before rotation, $s_+(x)$ afterwards

Lemma. After a double rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x)) - 2$.

Proof.

Case 2. Right-Left(x)

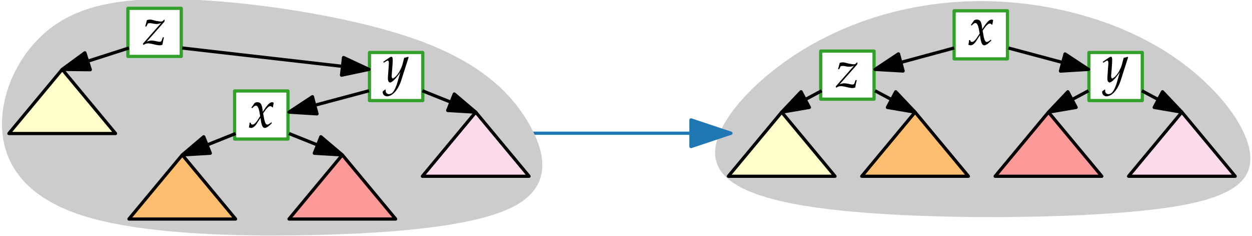


Potential after Rotation

Consider any rotation; $s(x)$ before rotation, $s_+(x)$ afterwards

Lemma. After a double rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x)) - 2$.

Proof.
Case 2. Right-Left(x)



pot. change $= \log s_+(x) + \log s_+(y) + \log s_+(z)$
$\quad - \log s(x) - \log s(y) - \log s(z)$

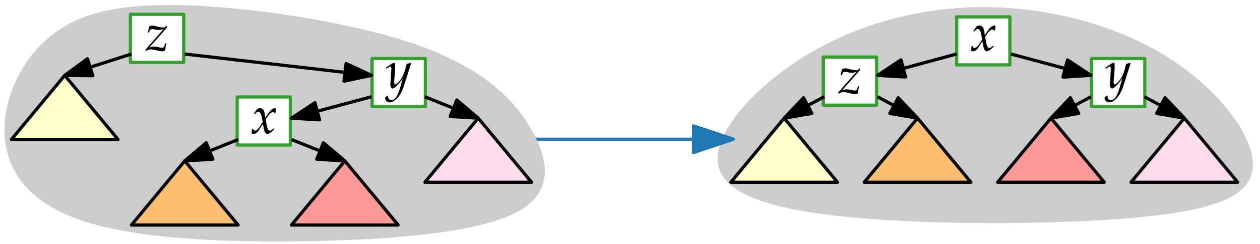
Potential after Rotation

Consider any rotation; $s(x)$ before rotation, $s_+(x)$ afterwards

Lemma. After a double rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x)) - 2$.

Proof.

Case 2. Right-Left(x)



pot. change $= \log s_+(x) + \log s_+(y) + \log s_+(z) - \log s(x) - \log s(y) - \log s(z)$

$(s_+(x) = s(z))$ $= \log s_+(y) + \log s_+(z) - \log s(x) - \log s(y)$

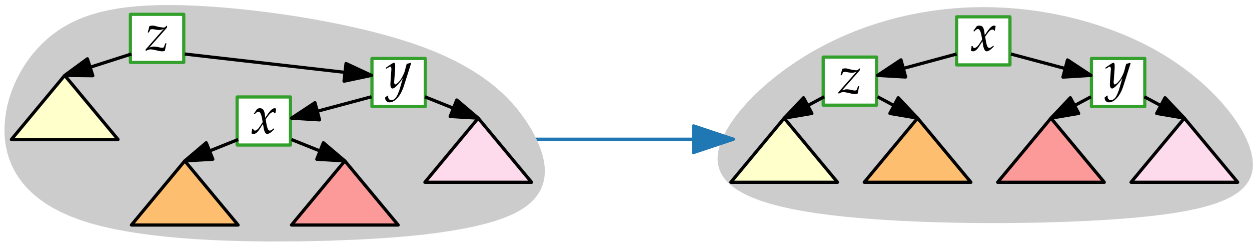
Potential after Rotation

Consider any rotation; $s(x)$ before rotation, $s_+(x)$ afterwards

Lemma. After a double rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x)) - 2$.

Proof.

Case 2. Right-Left(x)



$$
\begin{aligned}
\text{pot. change} \quad &= \log s_+(x) + \log s_+(y) + \log s_+(z) \\
&\quad - \log s(x) - \log s(y) - \log s(z) \\
(s_+(x) = s(z)) \quad &= \log s_+(y) + \log s_+(z) - \log s(x) - \log s(y) \\
(s(x) \leq s(y)) \quad &\leq \log s_+(y) + \log s_+(z) - 2\log s(x)
\end{aligned}
$$

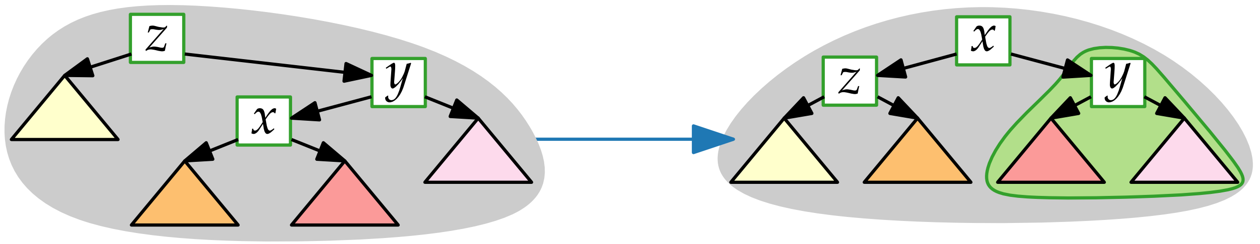
Potential after Rotation

Consider any rotation; $s(x)$ before rotation, $s_+(x)$ afterwards

Lemma. After a double rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x)) - 2$.

Proof.

Case 2. Right-Left(x)



$$\begin{aligned}
\text{pot. change} \quad &= \log s_+(x) + \log s_+(y) + \log s_+(z) \\
&\quad - \log s(x) - \log s(y) - \log s(z) \\
(s_+(x) = s(z)) \quad &= \log s_+(y) + \log s_+(z) - \log s(x) - \log s(y) \\
(s(x) \leq s(y)) \quad &\leq \log s_+(y) + \log s_+(z) - 2\log s(x)
\end{aligned}$$

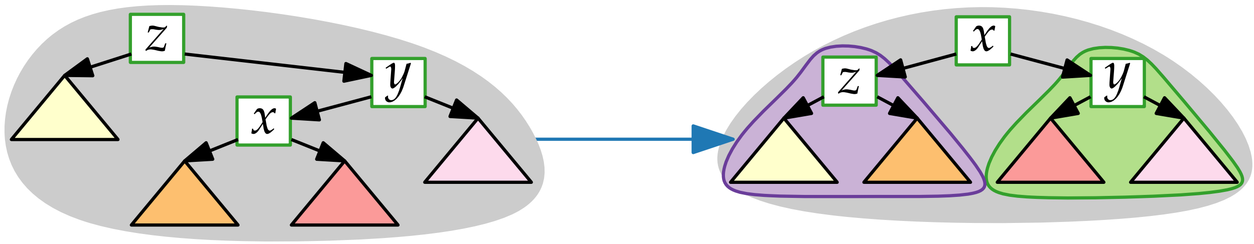
Potential after Rotation

Consider any rotation; $s(x)$ before rotation, $s_+(x)$ afterwards

Lemma. After a double rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x)) - 2$.

Proof.

Case 2. Right-Left(x)



pot. change $= \log s_+(x) + \log s_+(y) + \log s_+(z) - \log s(x) - \log s(y) - \log s(z)$

$(s_+(x) = s(z))$ $= \log s_+(y) + \log s_+(z) - \log s(x) - \log s(y)$

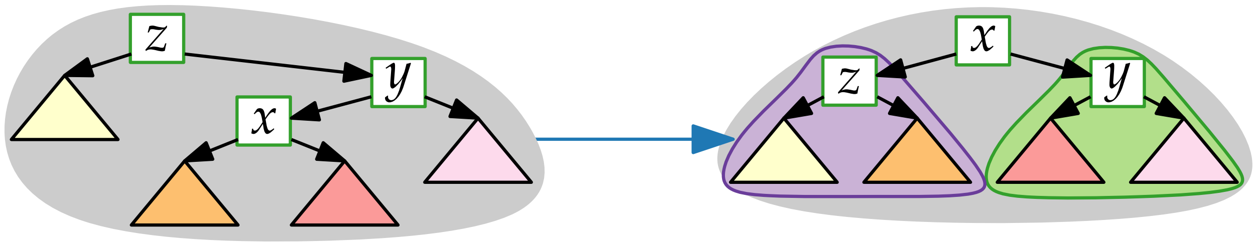
$(s(x) \leq s(y))$ $\leq \log s_+(y) + \log s_+(z) - 2\log s(x)$

Potential after Rotation

Consider any rotation; $s(x)$ before rotation, $s_+(x)$ afterwards

Lemma. After a double rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x)) - 2$.

Proof.
Case 2. Right-Left(x)



$$
\begin{aligned}
\text{pot. change} \quad &= \log s_+(x) + \log s_+(y) + \log s_+(z) \\
&\quad - \log s(x) - \log s(y) - \log s(z) \\
(s_+(x) = s(z)) \quad &= \log s_+(y) + \log s_+(z) - \log s(x) - \log s(y) \\
(s(x) \leq s(y)) \quad &\leq \log s_+(y) + \log s_+(z) - 2\log s(x)
\end{aligned}
$$

$s_+(y) + s_+(z) \leq s_+(x)$

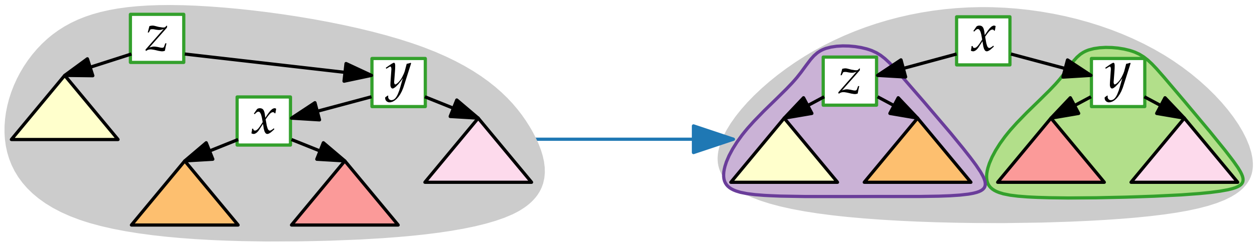
Potential after Rotation

Consider any rotation; $s(x)$ before rotation, $s_+(x)$ afterwards

Lemma. After a double rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x)) - 2$.

Proof.

Case 2. Right-Left(x)



$$\begin{aligned}
\text{pot. change} \quad &= \log s_+(x) + \log s_+(y) + \log s_+(z) \\
&\quad - \log s(x) - \log s(y) - \log s(z) \\
(s_+(x) = s(z)) \quad &= \log s_+(y) + \log s_+(z) - \log s(x) - \log s(y) \\
(s(x) \leq s(y)) \quad &\leq \log s_+(y) + \log s_+(z) - 2 \log s(x)
\end{aligned}$$

$$s_+(y) + s_+(z) \leq s_+(x) \quad \Rightarrow \log s_+(y) + \log s_+(z) \leq 2 \log s_+(x) - 2$$

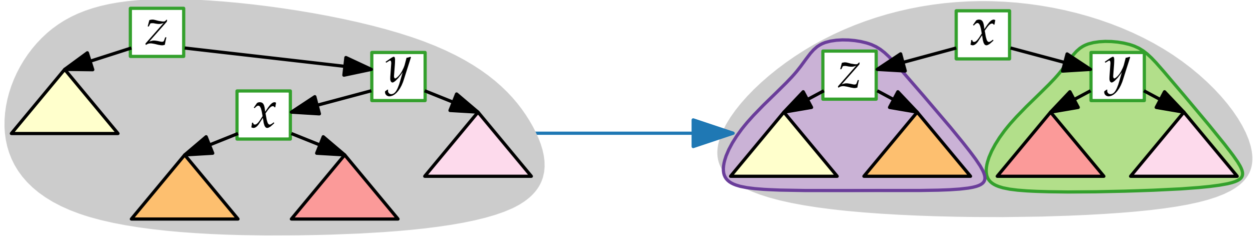
Potential after Rotation

Consider any rotation; $s(x)$ before rotation, $s_+(x)$ afterwards

Lemma. After a double rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x)) - 2$.

Proof.

Case 2. Right-Left(x)



$$\begin{aligned}
\text{pot. change} \quad &= \log s_+(x) + \log s_+(y) + \log s_+(z) \\
&\quad - \log s(x) - \log s(y) - \log s(z) \\
(s_+(x) = s(z)) \quad &= \log s_+(y) + \log s_+(z) - \log s(x) - \log s(y) \\
(s(x) \leq s(y)) \quad &\leq \log s_+(y) + \log s_+(z) - 2\log s(x)
\end{aligned}$$

$s_+(y) + s_+(z) \leq s_+(x) \quad \Rightarrow \log s_+(y) + \log s_+(z) \leq 2\log s_+(x) - 2$

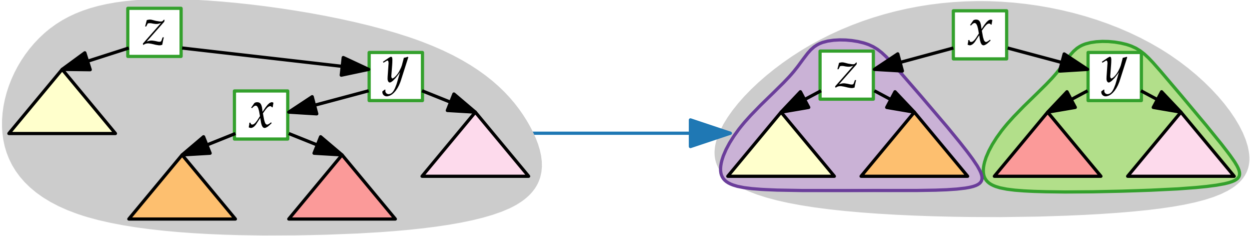
Potential after Rotation

Consider any rotation; $s(x)$ before rotation, $s_+(x)$ afterwards

Lemma. After a double rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x)) - 2$.

Proof.

Case 2. Right-Left(x)



$$\begin{aligned}
\text{pot. change} \quad &= \log s_+(x) + \log s_+(y) + \log s_+(z) \\
&\quad - \log s(x) - \log s(y) - \log s(z) \\
(s_+(x) = s(z)) \quad &= \log s_+(y) + \log s_+(z) - \log s(x) - \log s(y) \\
(s(x) \leq s(y)) \quad &\leq \log s_+(y) + \log s_+(z) - 2\log s(x) \\
&\leq 2\log s_+(x) - 2\log s(x) - 2
\end{aligned}$$

$s_+(y) + s_+(z) \leq s_+(x) \quad \Rightarrow \log s_+(y) + \log s_+(z) \leq 2\log s_+(x) - 2$

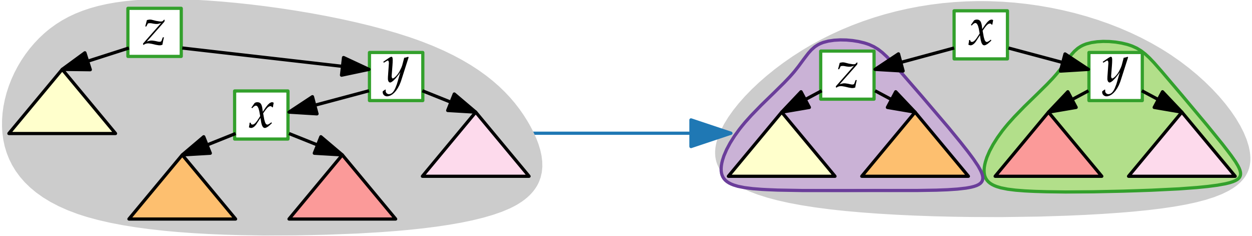
Potential after Rotation

Consider any rotation; $s(x)$ before rotation, $s_+(x)$ afterwards

Lemma. After a double rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x)) - 2$.

Proof.

Case 2. Right-Left(x)



pot. change $= \log s_+(x) + \log s_+(y) + \log s_+(z)$
$\quad - \log s(x) - \log s(y) - \log s(z)$

$(s_+(x) = s(z))$ $\quad = \log s_+(y) + \log s_+(z) - \log s(x) - \log s(y)$

$(s(x) \leq s(y))$ $\quad \leq \log s_+(y) + \log s_+(z) - 2\log s(x)$

$\quad \leq 2\log s_+(x) - 2\log s(x) - 2$

$(s_+(x) > s(x))$

$s_+(y) + s_+(z) \leq s_+(x) \;\Rightarrow\; \log s_+(y) + \log s_+(z) \leq 2\log s_+(x) - 2$

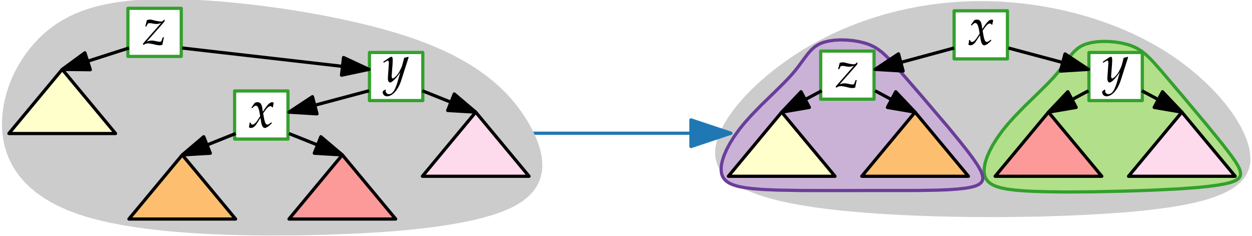
Potential after Rotation

Consider any rotation; $s(x)$ before rotation, $s_+(x)$ afterwards

Lemma. After a double rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x)) - 2$.

Proof.

Case 2. Right-Left(x)



$$\begin{aligned}
\text{pot. change} \quad &= \log s_+(x) + \log s_+(y) + \log s_+(z) \\
&\quad - \log s(x) - \log s(y) - \log s(z) \\
(s_+(x) = s(z)) \quad &= \log s_+(y) + \log s_+(z) - \log s(x) - \log s(y) \\
(s(x) \leq s(y)) \quad &\leq \log s_+(y) + \log s_+(z) - 2\log s(x) \\
&\leq 2\log s_+(x) - 2\log s(x) - 2 \\
(s_+(x) > s(x)) \quad &\leq 3\log s_+(x) - 3\log s(x) - 2
\end{aligned}$$

$$s_+(y) + s_+(z) \leq s_+(x) \quad \Rightarrow \log s_+(y) + \log s_+(z) \leq 2\log s_+(x) - 2$$

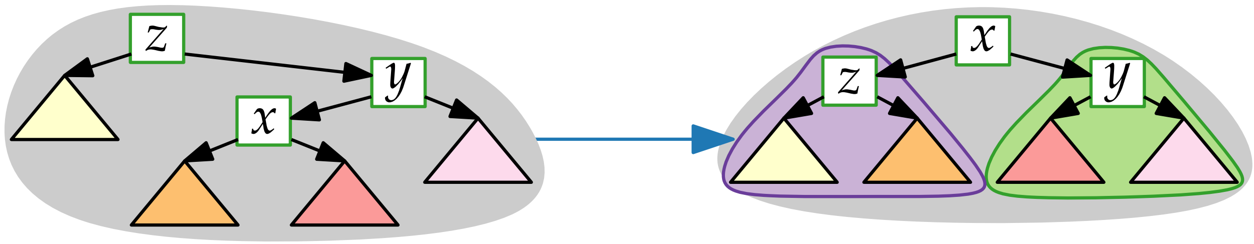
Potential after Rotation

Consider any rotation; $s(x)$ before rotation, $s_+(x)$ afterwards

Lemma. After a double rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x)) - 2$.

Proof.

Case 2. Right-Left(x)



$$
\begin{aligned}
\text{pot. change} \quad &= \log s_+(x) + \log s_+(y) + \log s_+(z) \\
&\quad - \log s(x) - \log s(y) - \log s(z) \\
(s_+(x) = s(z)) \quad &= \log s_+(y) + \log s_+(z) - \log s(x) - \log s(y) \\
(s(x) \leq s(y)) \quad &\leq \log s_+(y) + \log s_+(z) - 2\log s(x) \\
&\leq 2\log s_+(x) - 2\log s(x) - 2 \\
(s_+(x) > s(x)) \quad &\leq 3\log s_+(x) - 3\log s(x) - 2 \qquad \checkmark
\end{aligned}
$$

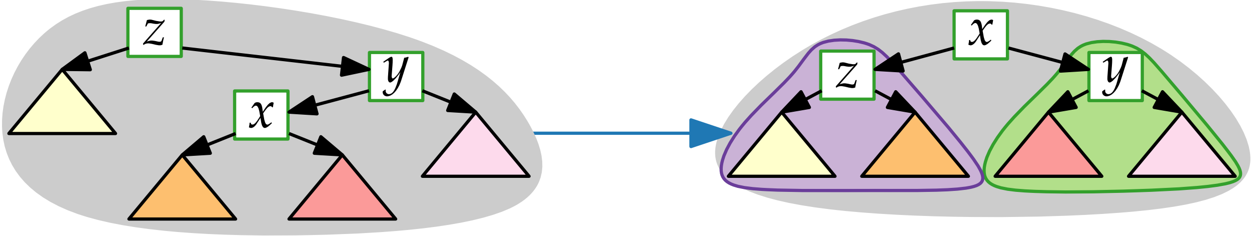
$s_+(y) + s_+(z) \leq s_+(x) \quad \Rightarrow \log s_+(y) + \log s_+(z) \leq 2\log s_+(x) - 2$

Potential after Rotation

Consider any rotation; $s(x)$ before rotation, $s_+(x)$ afterwards

Lemma. After a double rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x)) - 2$.

Proof. / Left-Right(x)
Case 2. Right-Left(x)



$$\begin{aligned}
\text{pot. change} \quad &= \log s_+(x) + \log s_+(y) + \log s_+(z) \\
&\quad - \log s(x) - \log s(y) - \log s(z) \\
(s_+(x) = s(z)) \quad &= \log s_+(y) + \log s_+(z) - \log s(x) - \log s(y) \\
(s(x) \leq s(y)) \quad &\leq \log s_+(y) + \log s_+(z) - 2\log s(x) \\
&\leq 2\log s_+(x) - 2\log s(x) - 2 \\
(s_+(x) > s(x)) \quad &\leq 3\log s_+(x) - 3\log s(x) - 2 \quad \checkmark
\end{aligned}$$

$$s_+(y) + s_+(z) \leq s_+(x) \;\Rightarrow\; \log s_+(y) + \log s_+(z) \leq 2\log s_+(x) - 2$$

Access Lemma

Lemma. After a single rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x))$.
After a double rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x)) - 2$.

Access Lemma

Lemma. After a single rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x))$.
After a double rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x)) - 2$.

Lemma. The (amortized) cost of Splay(x) is $\leq 1 + 3\log(W/w(x))$

Access Lemma

Lemma. After a single rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x))$.
After a double rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x)) - 2$.

Lemma. The (amortized) cost of Splay(x) is $\leq 1 + 3\log(W/w(x))$

Proof.

Access Lemma

Lemma. After a single rotation, the potential increases by $\leq 3\,(\log s_{+}(x) - \log s(x))$.
After a double rotation, the potential increases by $\leq 3\,(\log s_{+}(x) - \log s(x)) - 2$.

Lemma. The (amortized) cost of $\text{Splay}(x)$ is $\leq 1 + 3\log(W/w(x))$

Proof. W.l.o.g. k double rotations and 1 single rotation.

Access Lemma

Lemma. After a single rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x))$.
After a double rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x)) - 2$.

Lemma. The (amortized) cost of $\text{Splay}(x)$ is $\leq 1 + 3\log(W/w(x))$

Proof. W.l.o.g. k double rotations and 1 single rotation.
Let $s_i(x)$ be $s(x)$ after i single/double rotations.

Access Lemma

Lemma. After a single rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x))$.
After a double rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x)) - 2$.

Lemma. The (amortized) cost of Splay(x) is $\leq 1 + 3\log(W/w(x))$

Proof. W.l.o.g. k double rotations and 1 single rotation.
Let $s_i(x)$ be $s(x)$ after i single/double rotations.
Potential increases by at most

Access Lemma

Lemma. After a single rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x))$.
After a double rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x)) - 2$.

Lemma. The (amortized) cost of Splay(x) is $\leq 1 + 3\log(W/w(x))$

Proof. W.l.o.g. k double rotations and 1 single rotation.
Let $s_i(x)$ be $s(x)$ after i single/double rotations.
Potential increases by at most
$\sum_{i=1}^{k} (3\,(\log s_i(x) - \log s_{i-1}(x)) - 2)$

Access Lemma

Lemma. After a single rotation, the potential increases by $\leq 3\,(\log s_{+}(x) - \log s(x))$.
After a double rotation, the potential increases by $\leq 3\,(\log s_{+}(x) - \log s(x)) - 2$.

Lemma. The (amortized) cost of $\mathrm{Splay}(x)$ is $\leq 1 + 3\log(W/w(x))$

Proof. W.l.o.g. k double rotations and 1 single rotation.
Let $s_i(x)$ be $s(x)$ after i single/double rotations.
Potential increases by at most
$\sum_{i=1}^{k} (3\,(\log s_i(x) - \log s_{i-1}(x)) - 2)$
$+3\,(\log s_{k+1}(x) - \log s_k(x))$

Access Lemma

Lemma. After a single rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x))$.
After a double rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x)) - 2$.

Lemma. The (amortized) cost of Splay(x) is $\leq 1 + 3\log(W/w(x))$

Proof. W.l.o.g. k double rotations and 1 single rotation.
Let $s_i(x)$ be $s(x)$ after i single/double rotations.
Potential increases by at most

$$\sum_{i=1}^{k} (3\,(\log s_i(x) - \log s_{i-1}(x)) - 2)$$
$$+3\,(\log s_{k+1}(x) - \log s_k(x))$$
$$= 3\,(\log s_{k+1}(x) - \log s(x)) - 2k$$

Access Lemma

Lemma. After a single rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x))$.
After a double rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x)) - 2$.

Lemma. The (amortized) cost of Splay(x) is $\leq 1 + 3\log(W/w(x))$

Proof. W.l.o.g. k double rotations and 1 single rotation.
Let $s_i(x)$ be $s(x)$ after i single/double rotations.
Potential increases by at most

$$\sum_{i=1}^{k} \left(3\,(\log s_i(x) - \log s_{i-1}(x)) - 2\right)$$
$$+3\,(\log s_{k+1}(x) - \log s_k(x))$$

root!

$$= 3\,(\log s_{k+1}(x) - \log s(x)) - 2k$$

Access Lemma

Lemma. After a single rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x))$.
After a double rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x)) - 2$.

Lemma. The (amortized) cost of Splay(x) is $\leq 1 + 3\log(W/w(x))$

Proof. W.l.o.g. k double rotations and 1 single rotation.
Let $s_i(x)$ be $s(x)$ after i single/double rotations.
Potential increases by at most

$$\sum_{i=1}^{k} (3\,(\log s_i(x) - \log s_{i-1}(x)) - 2)$$
$$+3\,(\log s_{k+1}(x) - \log s_k(x))$$
$$= 3\,(\log s_{k+1}(x) - \log s(x)) - 2k$$
$$= 3\,(\log W - \log s(x)) - 2k$$

root!

Access Lemma

Lemma. After a single rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x))$.
After a double rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x)) - 2$.

Lemma. The (amortized) cost of Splay(x) is $\leq 1 + 3\log(W/w(x))$

Proof. W.l.o.g. k double rotations and 1 single rotation.
Let $s_i(x)$ be $s(x)$ after i single/double rotations.
Potential increases by at most
$\sum_{i=1}^{k} (3\,(\log s_i(x) - \log s_{i-1}(x)) - 2)$
$+3\,(\log s_{k+1}(x) - \log s_k(x))$

root!

$= 3\,(\log s_{k+1}(x) - \log s(x)) - 2k$
$= 3\,(\log W - \log s(x)) - 2k$

$(s(x) \geq w(x))$

Access Lemma

Lemma. After a single rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x))$.
After a double rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x)) - 2$.

Lemma. The (amortized) cost of Splay(x) is $\leq 1 + 3\log(W/w(x))$

Proof. W.l.o.g. k double rotations and 1 single rotation.
Let $s_i(x)$ be $s(x)$ after i single/double rotations.
Potential increases by at most

$$
\begin{aligned}
& \textstyle\sum_{i=1}^{k} \left(3\,(\log s_i(x) - \log s_{i-1}(x)) - 2\right) \\
& +3\,(\log s_{k+1}(x) - \log s_k(x)) \\
\text{root!} \quad & = 3\,(\log s_{k+1}(x) - \log s(x)) - 2k \\
& = 3\,(\log W - \log s(x)) - 2k \\
(s(x) \geq w(x)) \quad & \leq 3\,(\log W - \log w(x)) - 2k
\end{aligned}
$$

Access Lemma

Lemma. After a single rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x))$.
After a double rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x)) - 2$.

Lemma. The (amortized) cost of Splay(x) is $\leq 1 + 3\log(W/w(x))$

Proof. W.l.o.g. k double rotations and 1 single rotation.
Let $s_i(x)$ be $s(x)$ after i single/double rotations.
Potential increases by at most

$$\sum_{i=1}^{k} (3\,(\log s_i(x) - \log s_{i-1}(x)) - 2)$$
$$+3\,(\log s_{k+1}(x) - \log s_k(x))$$

root!

$$= 3\,(\log s_{k+1}(x) - \log s(x)) - 2k$$
$$= 3\,(\log W - \log s(x)) - 2k$$
$$(s(x) \geq w(x)) \quad \leq 3\,(\log W - \log w(x)) - 2k \;= 3\log(W/w(x)) - 2k$$

Access Lemma

Lemma. After a single rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x))$.
After a double rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x)) - 2$.

Lemma. The (amortized) cost of Splay(x) is $\leq 1 + 3\log(W/w(x))$

Proof. W.l.o.g. k double rotations and 1 single rotation.
Let $s_i(x)$ be $s(x)$ after i single/double rotations.
Potential increases by at most

$$\sum_{i=1}^{k} (3\,(\log s_i(x) - \log s_{i-1}(x)) - 2)$$
$$+3\,(\log s_{k+1}(x) - \log s_k(x))$$

root!

$$= 3\,(\log s_{k+1}(x) - \log s(x)) - 2k$$
$$= 3\,(\log W - \log s(x)) - 2k$$
$$(s(x) \geq w(x)) \quad \leq 3\,(\log W - \log w(x)) - 2k \; = 3\log(W/w(x)) - 2k$$

$2k+1$ rotations $\Rightarrow$ (amort.) cost

Access Lemma

Lemma. After a single rotation, the potential increases by $\leq 3\left(\log s_+(x) - \log s(x)\right)$.
After a double rotation, the potential increases by $\leq 3\left(\log s_+(x) - \log s(x)\right) - 2$.

Lemma. The (amortized) cost of Splay(x) is $\leq 1 + 3\log(W/w(x))$

Proof. W.l.o.g. k double rotations and 1 single rotation.
Let $s_i(x)$ be $s(x)$ after i single/double rotations.
Potential increases by at most

$$\sum_{i=1}^{k}\left(3\left(\log s_i(x) - \log s_{i-1}(x)\right) - 2\right)$$
$$+3\left(\log s_{k+1}(x) - \log s_k(x)\right)$$

root!

$$= 3\left(\log s_{k+1}(x) - \log s(x)\right) - 2k$$
$$= 3\left(\log W - \log s(x)\right) - 2k$$
$$(s(x) \geq w(x)) \quad \leq 3\left(\log W - \log w(x)\right) - 2k = 3\log(W/w(x)) - 2k$$

$2k+1$ rotations $\Rightarrow$ (amort.) cost $\leq 1 + 3\log(W/w(x))$

Access Lemma

Lemma. After a single rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x))$.
After a double rotation, the potential increases by $\leq 3\,(\log s_+(x) - \log s(x)) - 2$.

Lemma. The (amortized) cost of Splay(x) is $\leq 1 + 3\log(W/w(x))$

Proof. W.l.o.g. k double rotations and 1 single rotation.
Let $s_i(x)$ be $s(x)$ after i single/double rotations.
Potential increases by at most

$$\sum_{i=1}^{k} \left(3\,(\log s_i(x) - \log s_{i-1}(x)) - 2\right)$$
$$+3\,(\log s_{k+1}(x) - \log s_k(x))$$

root!

$$= 3\,(\log s_{k+1}(x) - \log s(x)) - 2k$$
$$= 3\,(\log W - \log s(x)) - 2k$$

$(s(x) \geq w(x))$ $\leq 3\,(\log W - \log w(x)) - 2k = 3\log(W/w(x)) - 2k$

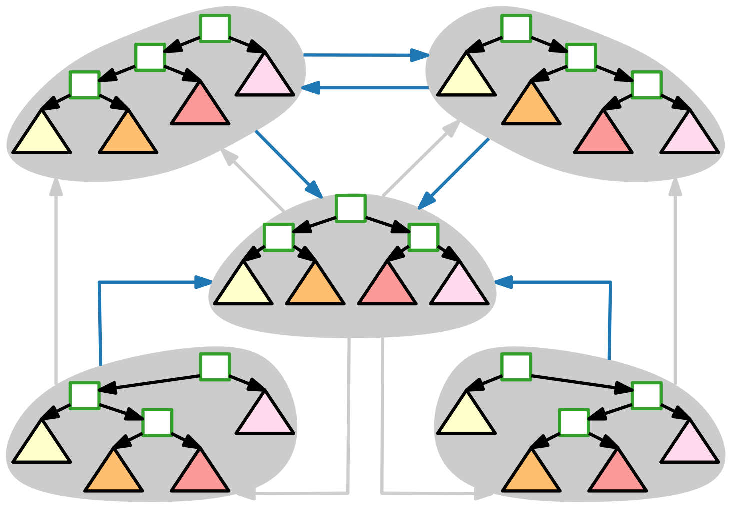
$2k+1$ rotations $\Rightarrow$ (amort.) cost $\leq 1 + 3\log(W/w(x))$ $\square$

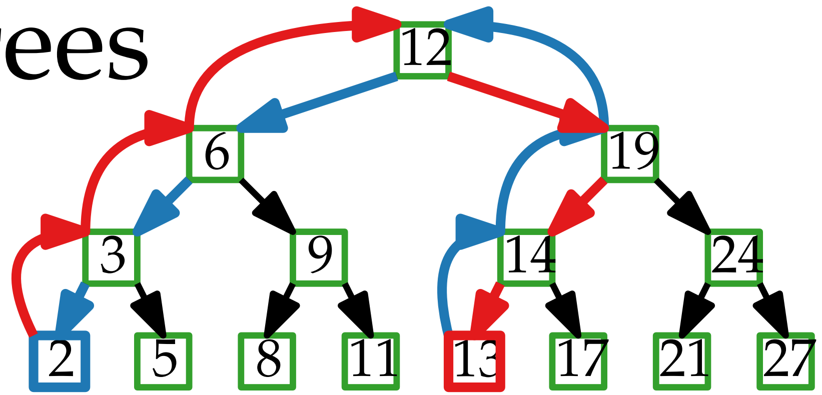
Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Advanced Algorithms

Optimal Binary Search Trees

Splay Trees





Philipp Kindermann · WS20

Part VI:
Properties of Splay Trees

All These Properties...

Balanced:	Queries take (amort.) $O(\log n)$ time
Entropy:	Queries take expected $O(1 + H)$ time
Dynamic Finger:	Queries take $O(\log \delta_i)$ time (δ_i: rank diff.)
Working Set:	Queries take $O(\log t)$ time (t: recency)
Static Optimality:	Queries take (amort.) $O(\mathrm{OPT}_S)$ time.

... is there one BST to rule them all?

Yes!

Querying a Sequence

Let S be a sequence of queries.

Querying a Sequence

Let S be a sequence of queries.

What is the *real* cost of querying S?

Querying a Sequence

Let S be a sequence of queries.

What is the *real* cost of querying S?

Let Φ_k be the potential after query k.

Querying a Sequence

Let S be a sequence of queries.

What is the *real* cost of querying S?

Let Φ_k be the potential after query k.

$\Rightarrow$ total cost $\Phi_0 - \Phi_{|S|} + \sum_{x \in S} \mathrm{Splay}(x)$

Querying a Sequence

Let S be a sequence of queries.

What is the *real* cost of querying S?

Let Φ_k be the potential after query k.

$\Rightarrow$ total cost $\Phi_0 - \Phi_{|S|} + \sum_{x \in S} \text{Splay}(x)$

How can we bound $\Phi_0 - \Phi_{|S|}$?

Querying a Sequence

Let S be a sequence of queries.

What is the *real* cost of querying S?

Let Φ_k be the potential after query k.

$\Rightarrow$ total cost $\Phi_0 - \Phi_{|S|} + \sum_{x \in S} \text{Splay}(x)$

How can we bound $\Phi_0 - \Phi_{|S|}$?

Reminder: $\Phi = \sum \log s(x)$

Querying a Sequence

Let S be a sequence of queries.

What is the *real* cost of querying S?

Let Φ_k be the potential after query k.

$\Rightarrow$ total cost $\Phi_0 - \Phi_{|S|} + \sum_{x \in S} \text{Splay}(x)$

How can we bound $\Phi_0 - \Phi_{|S|}$?

Reminder: $\Phi = \sum \log s(x)$

$s(x) \geq w(x)$

Querying a Sequence

Let S be a sequence of queries.

What is the *real* cost of querying S?

Let Φ_k be the potential after query k.

$\Rightarrow$ total cost $\Phi_0 - \Phi_{|S|} + \sum_{x \in S} \text{Splay}(x)$

How can we bound $\Phi_0 - \Phi_{|S|}$?

Reminder: $\Phi = \sum \log s(x)$

$s(x) \geq w(x)$ $\quad\quad\quad \Rightarrow \Phi_{|S|} \geq \sum_{x \in T} \log w(x)$

Querying a Sequence

Let S be a sequence of queries.

What is the *real* cost of querying S?

Let Φ_k be the potential after query k.

$\Rightarrow$ total cost $\Phi_0 - \Phi_{|S|} + \sum_{x \in S} \text{Splay}(x)$

How can we bound $\Phi_0 - \Phi_{|S|}$?

Reminder: $\Phi = \sum \log s(x)$

$s(x) \geq w(x)$ $\qquad \Rightarrow \Phi_{|S|} \geq \sum_{x \in T} \log w(x)$

$s(\text{root}) = \log W$

Querying a Sequence

Let S be a sequence of queries.

What is the *real* cost of querying S?

Let Φ_k be the potential after query k.

$\Rightarrow$ total cost $\Phi_0 - \Phi_{|S|} + \sum_{x \in S} \mathrm{Splay}(x)$

How can we bound $\Phi_0 - \Phi_{|S|}$?

Reminder: $\Phi = \sum \log s(x)$

$s(x) \geq w(x) \qquad \Rightarrow \Phi_{|S|} \geq \sum_{x \in T} \log w(x)$

$s(\mathrm{root}) = \log W \qquad \Rightarrow \Phi_0 \leq \sum_{x \in T} \log W$

Querying a Sequence

Let S be a sequence of queries.

What is the *real* cost of querying S?

Let Φ_k be the potential after query k.

$\Rightarrow$ total cost $\Phi_0 - \Phi_{|S|} + \sum_{x \in S} \text{Splay}(x)$

How can we bound $\Phi_0 - \Phi_{|S|}$?

Reminder: $\Phi = \sum \log s(x)$

$s(x) \geq w(x) \qquad \Rightarrow \Phi_{|S|} \geq \sum_{x \in T} \log w(x)$

$s(\text{root}) = \log W \qquad \Rightarrow \Phi_0 \leq \sum_{x \in T} \log W$

$\Rightarrow \Phi_0 - \Phi_{|S|} \leq \sum_{x \in T} (\log W - \log w(x))$

Querying a Sequence

Let S be a sequence of queries.

What is the *real* cost of querying S?

Let Φ_k be the potential after query k.

$\Rightarrow$ total cost $\Phi_0 - \Phi_{|S|} + \sum_{x \in S} \text{Splay}(x)$

How can we bound $\Phi_0 - \Phi_{|S|}$?

Reminder: $\Phi = \sum \log s(x)$

$s(x) \geq w(x) \qquad \Rightarrow \Phi_{|S|} \geq \sum_{x \in T} \log w(x)$

$s(\text{root}) = \log W \qquad \Rightarrow \Phi_0 \leq \sum_{x \in T} \log W$

$\Rightarrow \Phi_0 - \Phi_{|S|} \leq \sum_{x \in T} (\log W - \log w(x)) \leq \sum_{x \in T} O\left(\text{Splay}(x)\right)$

Querying a Sequence

Let S be a sequence of queries.

What is the *real* cost of querying S?

Let Φ_k be the potential after query k.

$\Rightarrow$ total cost $\Phi_0 - \Phi_{|S|} + \sum_{x \in S} \text{Splay}(x)$

How can we bound $\Phi_0 - \Phi_{|S|}$?

Reminder: $\Phi = \sum \log s(x)$

$s(x) \geq w(x) \qquad \Rightarrow \Phi_{|S|} \geq \sum_{x \in T} \log w(x)$

$s(\text{root}) = \log W \qquad \Rightarrow \Phi_0 \leq \sum_{x \in T} \log W$

$\Rightarrow \Phi_0 - \Phi_{|S|} \leq \sum_{x \in T} (\log W - \log w(x)) \leq \sum_{x \in T} O\left(\text{Splay}(x)\right)$

$\Rightarrow$ as long as every key is queried at least once, it doesn't change the asymptotic running time.

Balance

Lemma. The (amortized) cost of Splay(x) is $\leq 1 + 3\log(W/w(x))$

Definition. A BST is **balanced** if the (amortized) cost of *any* query is $O(\log n)$ (for at least n queries in total).

Balance

Lemma. The (amortized) cost of Splay(x) is $\leq 1 + 3\log(W/w(x))$

Definition. A BST is **balanced** if the (amortized) cost of *any* query is $O(\log n)$ (for at least n queries in total).

Theorem. Splay Trees are balanced.

Balance

Lemma. The (amortized) cost of $\text{Splay}(x)$ is $\leq 1 + 3\log(W/w(x))$

Definition. A BST is **balanced** if the (amortized) cost of *any* query is $O(\log n)$ (for at least n queries in total).

Theorem. Splay Trees are balanced.

Proof.

Balance

Lemma. The (amortized) cost of $\mathrm{Splay}(x)$ is $\leq 1 + 3\log(W/w(x))$

Definition. A BST is **balanced** if the (amortized) cost of *any* query is $O(\log n)$ (for at least n queries in total).

Theorem. Splay Trees are balanced.

Proof. Choose $w(x) = 1$ for each x

Balance

Lemma. The (amortized) cost of Splay(x) is $\leq 1 + 3\log(W/w(x))$

Definition. A BST is **balanced** if the (amortized) cost of *any* query is $O(\log n)$ (for at least n queries in total).

Theorem. Splay Trees are balanced.

Proof. Choose $w(x) = 1$ for each x $\Rightarrow W = n$

Balance

Lemma. The (amortized) cost of $\text{Splay}(x)$ is $\leq 1 + 3\log(W/w(x))$

Definition. A BST is **balanced** if the (amortized) cost of *any* query is $O(\log n)$ (for at least n queries in total).

Theorem. Splay Trees are balanced.

Proof. Choose $w(x) = 1$ for each x $\Rightarrow W = n$
$\text{Splay}(x)$ costs at least as much as finding x

Balance

Lemma. The (amortized) cost of $\text{Splay}(x)$ is $\leq 1 + 3\log(W/w(x))$

Definition. A BST is **balanced** if the (amortized) cost of *any* query is $O(\log n)$ (for at least n queries in total).

Theorem. Splay Trees are balanced.

Proof. Choose $w(x) = 1$ for each x $\Rightarrow W = n$
$\text{Splay}(x)$ costs at least as much as finding x
$\Rightarrow$ total time $= \Phi_0 - \Phi_{|S|} + \sum_{x \in S} \text{Splay}(x)$

Balance

Lemma. The (amortized) cost of Splay(x) is $\leq 1 + 3\log(W/w(x))$

Definition. A BST is **balanced** if the (amortized) cost of *any* query is $O(\log n)$ (for at least n queries in total).

Theorem. Splay Trees are balanced.

Proof. Choose $w(x) = 1$ for each x $\Rightarrow W = n$
Splay(x) costs at least as much as finding x
$\Rightarrow$ total time $= \Phi_0 - \Phi_{|S|} + \sum_{x \in S} \text{Splay}(x)$
$\leq \sum_{x \in T}(\log W - \log w(x)) + \sum_{x \in S} \text{Splay}(x)$

Balance

Lemma. The (amortized) cost of $\text{Splay}(x)$ is $\leq 1 + 3\log(W/w(x))$

Definition. A BST is **balanced** if the (amortized) cost of *any* query is $O(\log n)$ (for at least n queries in total).

Theorem. Splay Trees are balanced.

Proof. Choose $w(x) = 1$ for each $x \quad \Rightarrow W = n$
$\text{Splay}(x)$ costs at least as much as finding x
$$\begin{aligned}\Rightarrow \text{total time} &= \Phi_0 - \Phi_{|S|} + \textstyle\sum_{x\in S} \text{Splay}(x)\\ &\leq \textstyle\sum_{x\in T}(\log W - \log w(x)) + \sum_{x\in S} \text{Splay}(x)\\ &\leq n\log n + \textstyle\sum_{x\in S}(1 + 3\log(W/w(x)))\end{aligned}$$

Balance

Lemma. The (amortized) cost of $\text{Splay}(x)$ is $\leq 1 + 3\log(W/w(x))$

Definition. A BST is **balanced** if the (amortized) cost of *any* query is $O(\log n)$ (for at least n queries in total).

Theorem. Splay Trees are balanced.

Proof. Choose $w(x) = 1$ for each $x \quad \Rightarrow W = n$

$\text{Splay}(x)$ costs at least as much as finding x

$$\begin{aligned}
\Rightarrow \text{total time} &= \Phi_0 - \Phi_{|S|} + \textstyle\sum_{x \in S} \text{Splay}(x) \\
&\leq \textstyle\sum_{x \in T} (\log W - \log w(x)) + \sum_{x \in S} \text{Splay}(x) \\
&\leq n \log n + \textstyle\sum_{x \in S} \left(1 + 3\log(W/w(x))\right) \\
&\leq n \log n + |S| + 3|S| \log n
\end{aligned}$$

Balance

Lemma. The (amortized) cost of Splay(x) is $\leq 1 + 3\log(W/w(x))$

Definition. A BST is **balanced** if the (amortized) cost of *any* query is $O(\log n)$ (for at least n queries in total).

Theorem. Splay Trees are balanced.

Proof. Choose $w(x) = 1$ for each x $\Rightarrow W = n$
Splay(x) costs at least as much as finding x

$$\begin{aligned}\Rightarrow \text{total time} &= \Phi_0 - \Phi_{|S|} + \textstyle\sum_{x \in S} \text{Splay}(x) \\ &\leq \textstyle\sum_{x \in T}(\log W - \log w(x)) + \sum_{x \in S} \text{Splay}(x) \\ &\leq n \log n + \textstyle\sum_{x \in S} (1 + 3\log(W/w(x))) \\ &\leq n \log n + |S| + 3|S| \log n \;\in O(|S| \log n)\end{aligned}$$

Balance

Lemma. The (amortized) cost of Splay(x) is $\leq 1 + 3\log(W/w(x))$

Definition. A BST is **balanced** if the (amortized) cost of *any* query is $O(\log n)$ (for at least n queries in total).

Theorem. Splay Trees are balanced.

Proof. Choose $w(x) = 1$ for each $x \quad \Rightarrow W = n$
Splay(x) costs at least as much as finding x

$$\begin{aligned}
\Rightarrow \text{total time} &= \Phi_0 - \Phi_{|S|} + \textstyle\sum_{x \in S} \text{Splay}(x) \\
&\leq \textstyle\sum_{x \in T}(\log W - \log w(x)) + \sum_{x \in S} \text{Splay}(x) \\
&\leq n \log n + \textstyle\sum_{x \in S} (1 + 3\log(W/w(x))) \\
&\leq n \log n + |S| + 3|S| \log n \ \in O(|S| \log n)
\end{aligned}$$

$\Rightarrow$ Queries take (amort.) $O(\log n)$ time.

Balance

Lemma. The (amortized) cost of $\text{Splay}(x)$ is $\leq 1 + 3\log(W/w(x))$

Definition. A BST is **balanced** if the (amortized) cost of *any* query is $O(\log n)$ (for at least n queries in total).

Theorem. Splay Trees are balanced.

Proof. Choose $w(x) = 1$ for each $x \quad \Rightarrow W = n$

$\text{Splay}(x)$ costs at least as much as finding x

$$\begin{aligned}
\Rightarrow \text{total time} &= \Phi_0 - \Phi_{|S|} + \textstyle\sum_{x \in S} \text{Splay}(x) \\
&\leq \textstyle\sum_{x \in T}(\log W - \log w(x)) + \sum_{x \in S} \text{Splay}(x) \\
&\leq n \log n + \textstyle\sum_{x \in S} (1 + 3\log(W/w(x))) \\
&\leq n \log n + |S| + 3|S| \log n \ \in O(|S| \log n)
\end{aligned}$$

$\Rightarrow$ Queries take (amort.) $O(\log n)$ time. $\square$

Entropy

Lemma. The (amortized) cost of Splay(x) is $\leq 1 + 3\log(W/w(x))$

Definition. A BST has the **entropy property** if queries take expected $O(1 - \sum_{i=1}^{n} p_i \log p_i)$ time.

Entropy

Lemma. The (amortized) cost of Splay(x) is $\leq 1 + 3\log(W/w(x))$

Definition. A BST has the **entropy property** if queries take expected $O(1 - \sum_{i=1}^{n} p_i \log p_i)$ time.

Theorem. Splay Trees have the entropy property.

Entropy

Lemma. The (amortized) cost of Splay(x) is $\leq 1 + 3\log(W/w(x))$

Definition. A BST has the **entropy property** if queries take expected $O(1 - \sum_{i=1}^{n} p_i \log p_i)$ time.

Theorem. Splay Trees have the entropy property.

Proof.

Entropy

Lemma. The (amortized) cost of Splay(x) is $\leq 1 + 3\log(W/w(x))$

Definition. A BST has the **entropy property** if queries take expected $O(1 - \sum_{i=1}^{n} p_i \log p_i)$ time.

Theorem. Splay Trees have the entropy property.

Proof. Choose $w(x_i) = p_i$

Entropy

Lemma. The (amortized) cost of Splay(x) is $\leq 1 + 3\log(W/w(x))$

Definition. A BST has the **entropy property** if queries take expected $O(1 - \sum_{i=1}^{n} p_i \log p_i)$ time.

Theorem. Splay Trees have the entropy property.

Proof. Choose $w(x_i) = p_i \quad \Rightarrow W = 1$

Entropy

Lemma. The (amortized) cost of Splay(x) is $\leq 1 + 3\log(W/w(x))$

Definition. A BST has the **entropy property** if queries take expected $O(1 - \sum_{i=1}^{n} p_i \log p_i)$ time.

Theorem. Splay Trees have the entropy property.

Proof. Choose $w(x_i) = p_i \quad \Rightarrow W = 1$

Time to query x_i:

Entropy

Lemma. The (amortized) cost of Splay(x) is $\leq 1 + 3\log(W/w(x))$

Definition. A BST has the **entropy property** if queries take expected $O(1 - \sum_{i=1}^{n} p_i \log p_i)$ time.

Theorem. Splay Trees have the entropy property.

Proof. Choose $w(x_i) = p_i \quad \Rightarrow W = 1$

Time to query x_i:

$\leq 1 + 3\log(W/w(x_i))$

Entropy

Lemma. The (amortized) cost of Splay(x) is $\leq 1 + 3\log(W/w(x))$

Definition. A BST has the **entropy property** if queries take expected $O(1 - \sum_{i=1}^{n} p_i \log p_i)$ time.

Theorem. Splay Trees have the entropy property.

Proof. Choose $w(x_i) = p_i \quad \Rightarrow W = 1$

Time to query x_i:

$$\leq 1 + 3\log(W/w(x_i))$$
$$= 1 + 3\log(1/p_i)$$

Entropy

Lemma. The (amortized) cost of Splay(x) is $\leq 1 + 3\log(W/w(x))$

Definition. A BST has the **entropy property** if queries take expected $O(1 - \sum_{i=1}^{n} p_i \log p_i)$ time.

Theorem. Splay Trees have the entropy property.

Proof. Choose $w(x_i) = p_i \quad \Rightarrow W = 1$

Time to query x_i:

$$\begin{aligned} &\leq 1 + 3\log(W/w(x_i)) \\ &= 1 + 3\log(1/p_i) \\ &= 1 - 3\log p_i \end{aligned}$$

Entropy

Lemma. The (amortized) cost of Splay(x) is $\leq 1 + 3\log(W/w(x))$

Definition. A BST has the **entropy property** if queries take expected $O(1 - \sum_{i=1}^{n} p_i \log p_i)$ time.

Theorem. Splay Trees have the entropy property.

Proof. Choose $w(x_i) = p_i \quad \Rightarrow W = 1$

Time to query x_i:

$$\leq 1 + 3\log(W/w(x_i))$$
$$= 1 + 3\log(1/p_i)$$
$$= 1 - 3\log p_i$$

$\Rightarrow$ expected query time:

Entropy

Lemma. The (amortized) cost of Splay(x) is $\leq 1 + 3\log(W/w(x))$

Definition. A BST has the **entropy property** if queries take expected $O(1 - \sum_{i=1}^{n} p_i \log p_i)$ time.

Theorem. Splay Trees have the entropy property.

Proof. Choose $w(x_i) = p_i \quad \Rightarrow W = 1$

Time to query x_i:

$$\begin{aligned} &\leq 1 + 3\log(W/w(x_i)) \\ &= 1 + 3\log(1/p_i) \\ &= 1 - 3\log p_i \end{aligned}$$

$\Rightarrow$ expected query time:

$$O(\textstyle\sum_{i=1}^{n} p_i(1 - 3\log p_i))$$

Entropy

Lemma. The (amortized) cost of Splay(x) is $\leq 1 + 3\log(W/w(x))$

Definition. A BST has the **entropy property** if queries take expected $O(1 - \sum_{i=1}^{n} p_i \log p_i)$ time.

Theorem. Splay Trees have the entropy property.

Proof. Choose $w(x_i) = p_i \quad \Rightarrow W = 1$

Time to query x_i:

$$\leq 1 + 3\log(W/w(x_i))$$
$$= 1 + 3\log(1/p_i)$$
$$= 1 - 3\log p_i$$

$\Rightarrow$ expected query time:

$$O(\textstyle\sum_{i=1}^{n} p_i(1 - 3\log p_i))$$
$$= O(1 - \textstyle\sum_{i=1}^{n} p_i \log p_i)$$

Entropy

Lemma. The (amortized) cost of Splay(x) is $\leq 1 + 3\log(W/w(x))$

Definition. A BST has the **entropy property** if queries take expected $O(1 - \sum_{i=1}^{n} p_i \log p_i)$ time.

Theorem. Splay Trees have the entropy property.

Proof. Choose $w(x_i) = p_i \quad \Rightarrow W = 1$

Time to query x_i:

$$\leq 1 + 3\log(W/w(x_i))$$
$$= 1 + 3\log(1/p_i)$$
$$= 1 - 3\log p_i$$

$\Rightarrow$ expected query time:

$$O(\textstyle\sum_{i=1}^{n} p_i(1 - 3\log p_i))$$
$$= O(1 - \textstyle\sum_{i=1}^{n} p_i \log p_i) \qquad \square$$

Static Optimality

Given a sequence S of queries.

Static Optimality

Given a sequence S of queries.
Let T_S^* be the *optimal* static tree with the shortest query time OPT_S for S.

Static Optimality

Given a sequence S of queries.
Let T_S^* be the *optimal* static tree with the shortest query time OPT_S for S.

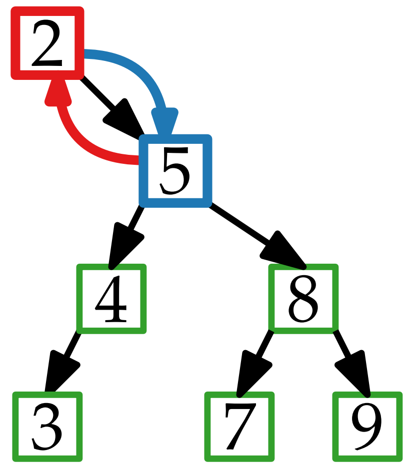
e.g. $S = 2, 5, 2, 5, 2, \ldots, 5$

Static Optimality

Given a sequence S of queries.
Let T^*_S be the *optimal* static tree with the shortest query time OPT_S for S.

e.g. $S = 2, 5, 2, 5, 2, \ldots, 5$

T^*:



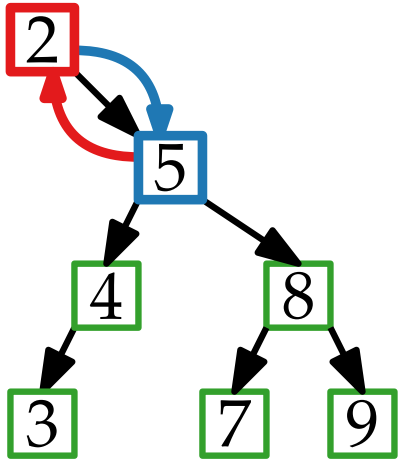
Static Optimality

Given a sequence S of queries.
Let T_S^* be the *optimal* static tree with the shortest query time OPT_S for S.

e.g. $S = 2, 5, 2, 5, 2, \ldots, 5$

T^*:

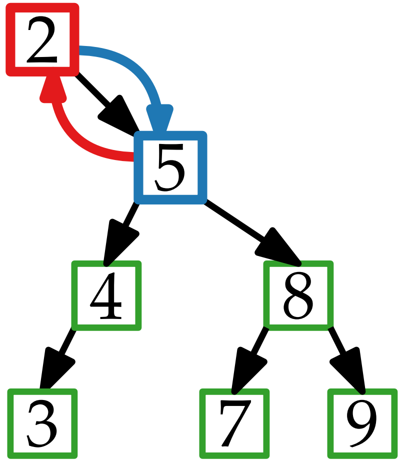
OPT: $|S|$



Static Optimality

Given a sequence S of queries.
Let T_S^* be the *optimal* static tree with the shortest query time OPT_S for S.

e.g. $S = 2, 5, 2, 5, 2, \ldots, 5$
T^*:
OPT: $|S|$

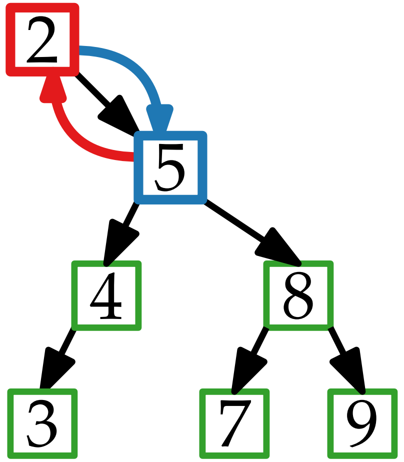


Definition. A BST is **statically optimal** if queries take (amort.) $O(\text{OPT}_S)$ time for every S.

Static Optimality

Given a sequence S of queries.
Let T_S^* be the *optimal* static tree with the shortest query time OPT_S for S.

e.g. $S = 2, 5, 2, 5, 2, \ldots, 5$
T^*:
OPT: $|S|$



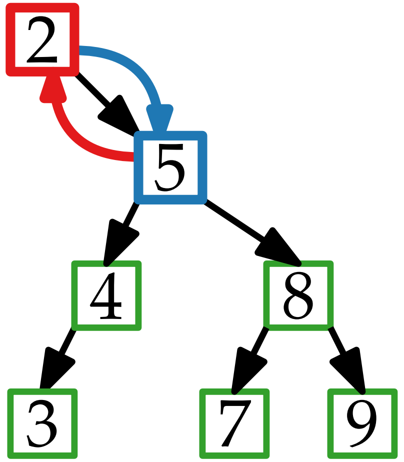
Definition. A BST is **statically optimal** if queries take (amort.) $O(\mathrm{OPT}_S)$ time for every S.

Theorem. Splay Trees are statically optimal.

Static Optimality

Given a sequence S of queries.
Let T_S^* be the *optimal* static tree with the shortest query time OPT_S for S.

e.g. $S = 2, 5, 2, 5, 2, \ldots, 5$
T^*:
OPT: $|S|$



Definition. A BST is **statically optimal** if queries take (amort.) $O(\mathrm{OPT}_S)$ time for every S.

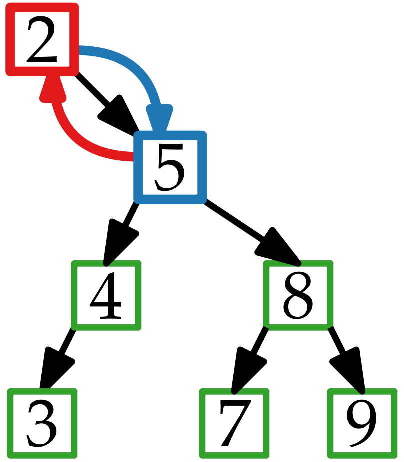
Theorem. Splay Trees are statically optimal.

Proof.

Static Optimality

Given a sequence S of queries.
Let T^*_S be the *optimal* static tree with the shortest query time OPT_S for S.

e.g. $S = 2, 5, 2, 5, 2, \ldots, 5$
T^*:
OPT: $|S|$



Definition. A BST is **statically optimal** if queries take (amort.) $O(\mathrm{OPT}_S)$ time for every S.

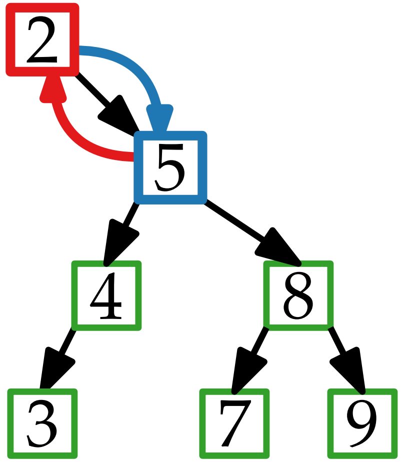
Theorem. Splay Trees are statically optimal.

Proof. Let f_i be the number of items on path to x_i in T^*.

Static Optimality

Given a sequence S of queries.
Let T^*_S be the *optimal* static tree with the shortest query time OPT_S for S.

e.g. $S = 2, 5, 2, 5, 2, \ldots, 5$
T^*:
OPT: $|S|$



Definition. A BST is **statically optimal** if queries take (amort.) $O(\mathrm{OPT}_S)$ time for every S.

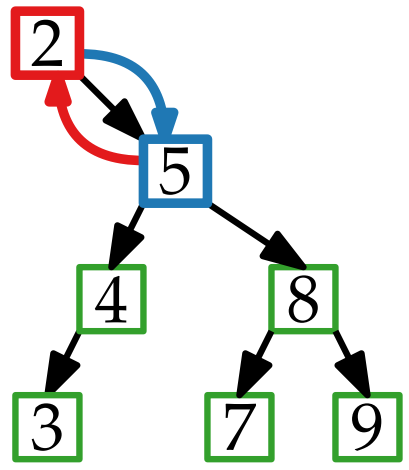
Theorem. Splay Trees are statically optimal.

Proof. Let f_i be the number of items on path to x_i in T^*.
Let $w_i := 3^{-f_i}$.

Static Optimality

Given a sequence S of queries.
Let T_S^* be the *optimal* static tree with the shortest query time OPT_S for S.

e.g. $S = 2, 5, 2, 5, 2, \ldots, 5$
T^*:
OPT: $|S|$



Definition. A BST is **statically optimal** if queries take (amort.) $O(\mathrm{OPT}_S)$ time for every S.

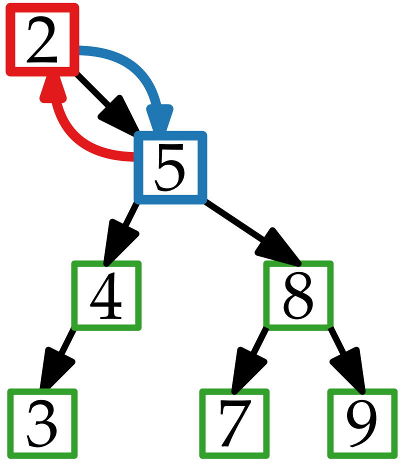
Theorem. Splay Trees are statically optimal.

Proof. Let f_i be the number of items on path to x_i in T^*.
Let $w_i := 3^{-f_i}$. $\Rightarrow W \leq 1$

Static Optimality

Given a sequence S of queries.
Let T^*_S be the *optimal* static tree with the shortest query time OPT_S for S.

e.g. $S = 2, 5, 2, 5, 2, \ldots, 5$
T^*:
OPT: $|S|$



Definition. A BST is **statically optimal** if queries take (amort.) $O(\text{OPT}_S)$ time for every S.

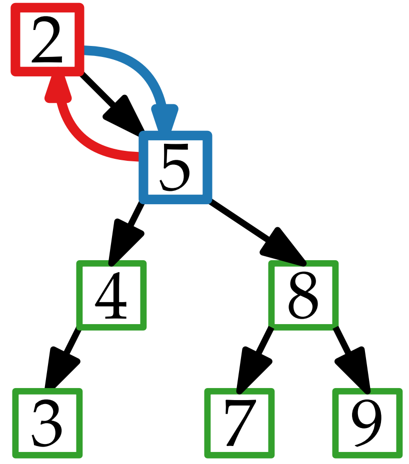
Theorem. Splay Trees are statically optimal.

Proof. Let f_i be the number of items on path to x_i in T^*.
Let $w_i := 3^{-f_i}$. $\Rightarrow W \leq 1$
$\Rightarrow \text{Splay}(x_i) = 1 + 3\log(W/w(x))$

Static Optimality

Given a sequence S of queries.
Let T_S^* be the *optimal* static tree with the shortest query time OPT_S for S.

e.g. $S = 2, 5, 2, 5, 2, \ldots, 5$
T^*:
OPT: $|S|$



Definition. A BST is **statically optimal** if queries take (amort.) $O(\mathrm{OPT}_S)$ time for every S.

Theorem. Splay Trees are statically optimal.

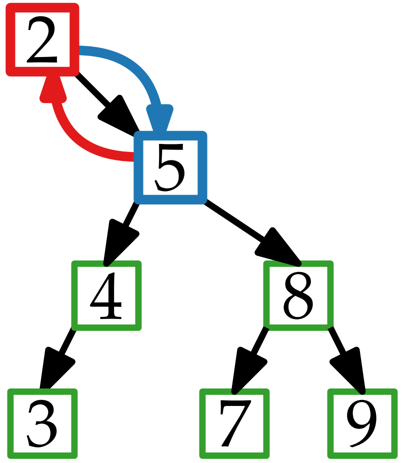
Proof. Let f_i be the number of items on path to x_i in T^*.
Let $w_i := 3^{-f_i}$. $\Rightarrow W \leq 1$

$$\Rightarrow \mathrm{Splay}(x_i) = 1 + 3\log(W/w(x))$$
$$\leq 1 + 3\log 3^{f_i}$$

Static Optimality

Given a sequence S of queries.
Let T_S^* be the *optimal* static tree with the shortest query time OPT_S for S.

e.g. $S = 2, 5, 2, 5, 2, \ldots, 5$
T^*:
OPT: $|S|$



Definition. A BST is **statically optimal** if queries take (amort.) $O(\mathrm{OPT}_S)$ time for every S.

Theorem. Splay Trees are statically optimal.

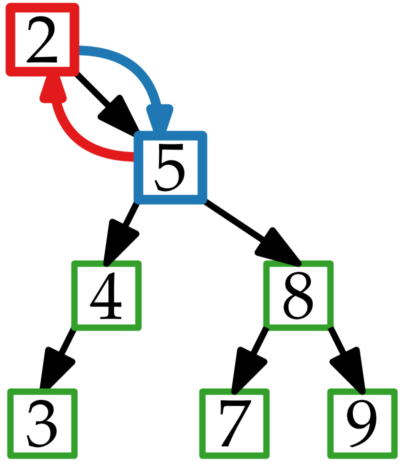
Proof. Let f_i be the number of items on path to x_i in T^*.
Let $w_i := 3^{-f_i}$. $\Rightarrow W \leq 1$

$$\Rightarrow \mathrm{Splay}(x_i) = 1 + 3\log(W/w(x))$$
$$\leq 1 + 3\log 3^{f_i} = O(f_i)$$

Static Optimality

Given a sequence S of queries.
Let T_S^* be the *optimal* static tree with the shortest query time OPT_S for S.

e.g. $S = 2, 5, 2, 5, 2, \ldots, 5$
T^*:
OPT: $|S|$



Definition. A BST is **statically optimal** if queries take (amort.) $O(\mathrm{OPT}_S)$ time for every S.

Theorem. Splay Trees are statically optimal.

Proof. Let f_i be the number of items on path to x_i in T^*.
Let $w_i := 3^{-f_i}$. $\Rightarrow W \leq 1$

$$\Rightarrow \mathrm{Splay}(x_i) = 1 + 3\log(W/w(x))$$
$$\leq 1 + 3\log 3^{f_i} \quad = O(f_i) \qquad \square$$

Dynamic Optimality

Given a sequence S of queries.

Dynamic Optimality

Given a sequence S of queries.
Let D_S^* be the optimal *dynamic* tree with the shortest query time OPT_S^* for S.

Dynamic Optimality

Given a sequence S of queries.
Let D_S^* be the optimal *dynamic* tree with the shortest query time OPT_S^* for S.
(That is, modifications are allowed, e.g. rotations)

Dynamic Optimality

Given a sequence S of queries.
Let D_S^* be the optimal *dynamic* tree with the shortest query time OPT_S^* for S.
(That is, modifications are allowed, e.g. rotations)

Definition. A BST is **dynamically optimal** if queries take (amort.) $O(\mathrm{OPT}_S^*)$ time for every S.

Dynamic Optimality

Given a sequence S of queries.
Let D_S^* be the optimal *dynamic* tree with the shortest query time OPT_S^* for S.
(That is, modifications are allowed, e.g. rotations)

Definition. A BST is **dynamically optimal** if queries take (amort.) $O(\mathrm{OPT}_S^*)$ time for every S.

Splay Trees: Queries take $O(\mathrm{OPT}_S^* \cdot \log n)$ time.

Dynamic Optimality

Given a sequence S of queries.
Let D_S^* be the optimal *dynamic* tree with the shortest query time OPT_S^* for S.
(That is, modifications are allowed, e.g. rotations)

Definition. A BST is **dynamically optimal** if queries take (amort.) $O(\mathrm{OPT}_S^*)$ time for every S.

Splay Trees: Queries take $O(\mathrm{OPT}_S^* \cdot \log n)$ time.
Tango Trees: Queries take $O(\mathrm{OPT}_S^* \cdot \log\log n)$ time.
[Demaine, Harmon, Iacono, Pătrașcu ’04]

Dynamic Optimality

Given a sequence S of queries.
Let D_S^* be the optimal *dynamic* tree with the shortest query time OPT_S^* for S.
(That is, modifications are allowed, e.g. rotations)

Definition. A BST is **dynamically optimal** if queries take (amort.) $O(\mathrm{OPT}_S^*)$ time for every S.

Splay Trees: Queries take $O(\mathrm{OPT}_S^* \cdot \log n)$ time.
Tango Trees: Queries take $O(\mathrm{OPT}_S^* \cdot \log\log n)$ time.
[Demaine, Harmon, Iacono, Pătrașcu '04]

Open Problem. Does a dynamically optimal BST exist?

Dynamic Optimality

Given a sequence S of queries.
Let D_S^* be the optimal *dynamic* tree with the shortest query time OPT_S^* for S.
(That is, modifications are allowed, e.g. rotations)

Definition. A BST is **dynamically optimal** if queries take (amort.) $O(\mathrm{OPT}_S^*)$ time for every S.

Splay Trees: Queries take $O(\mathrm{OPT}_S^* \cdot \log n)$ time.
Tango Trees: Queries take $O(\mathrm{OPT}_S^* \cdot \log\log n)$ time.
[Demaine, Harmon, Iacono, Pătrașcu '04]

Open Problem. Does a dynamically optimal BST exist?

This is one of the biggest open problems in algorithms.

Dynamic Optimality

Given a sequence S of queries.
Let D_S^* be the optimal *dynamic* tree with the shortest query time OPT_S^* for S.
(That is, modifications are allowed, e.g. rotations)

Definition. A BST is **dynamically optimal** if queries take (amort.) $O(\mathrm{OPT}_S^*)$ time for every S.

Splay Trees: Queries take $O(\mathrm{OPT}_S^* \cdot \log n)$ time.
Tango Trees: Queries take $O(\mathrm{OPT}_S^* \cdot \log\log n)$ time.
[Demaine, Harmon, Iacono, Pătrașcu '04]

Open Problem. Does a dynamically optimal BST exist?

This is one of the biggest open problems in algorithms.

Conjecture. Splay Trees are dynamically optimal.