Advanced Algorithms – COMS31900

Lowest Common Ancestor

Raphaël Clifford

Slides by Benjamin Sach

Advanced Algorithms – COMS31900

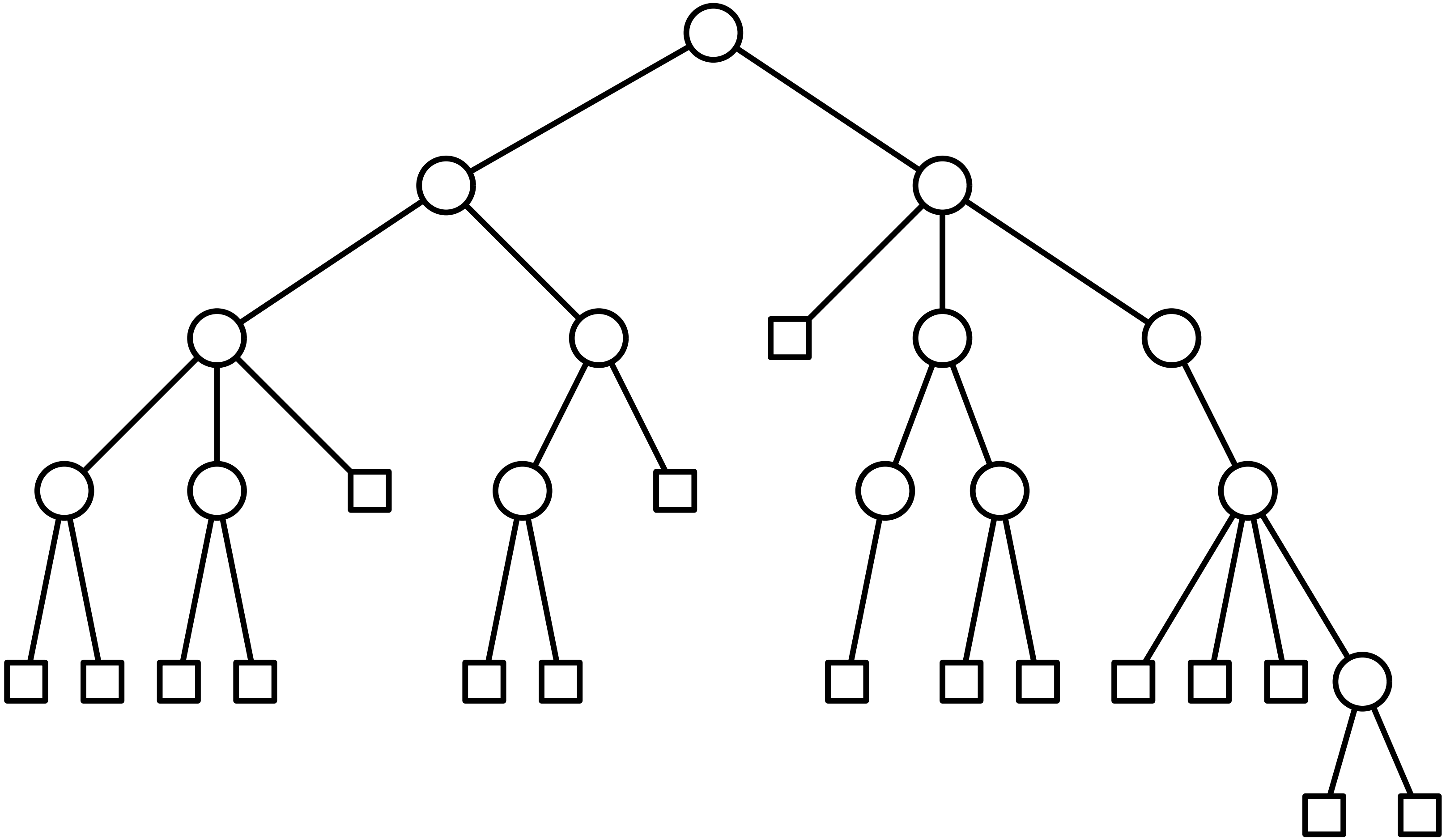
Lowest Common Ancestor

(with a bit on on Range Minimum Queries)

Raphaël Clifford

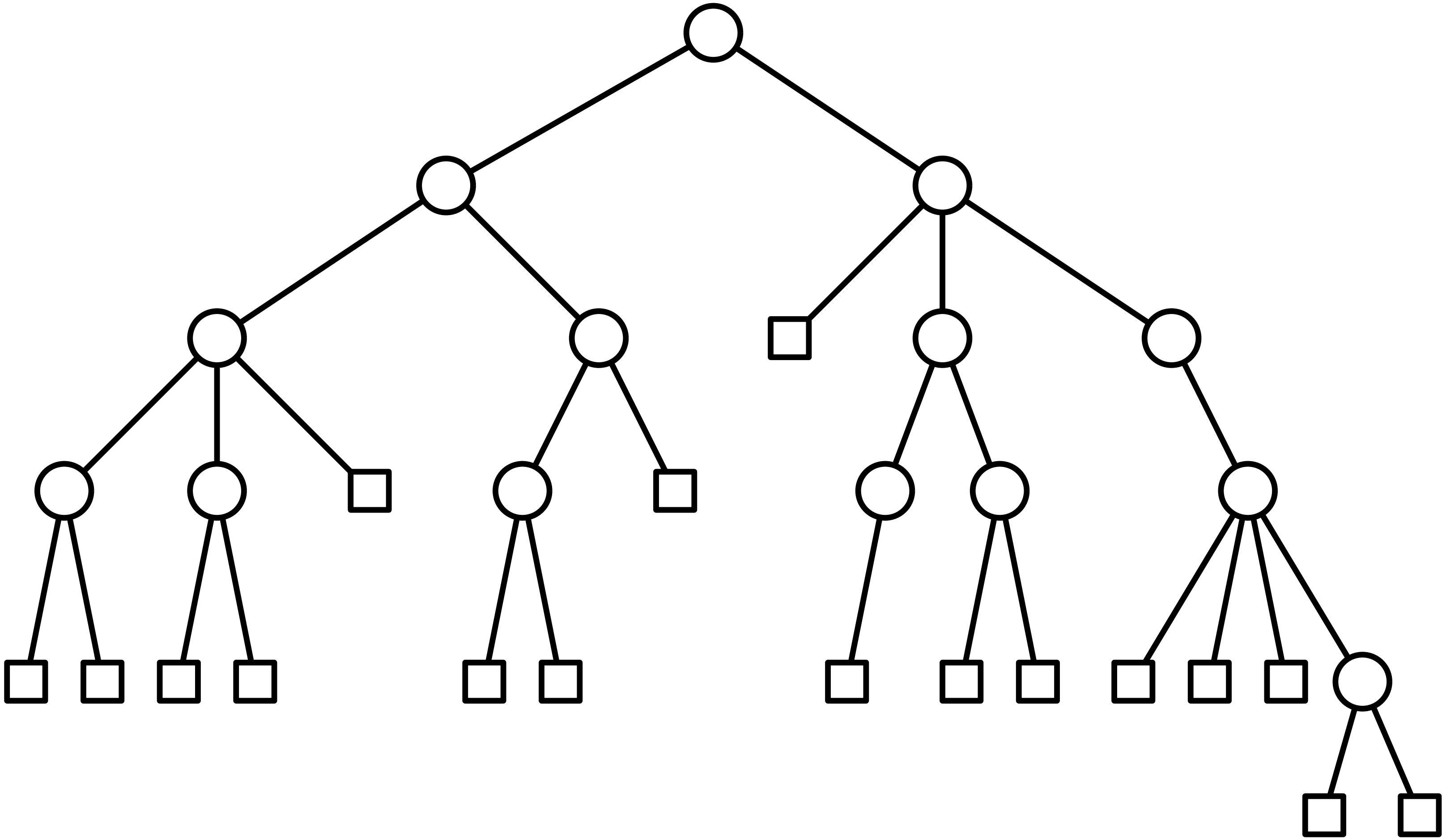
Lowest common ancestor

Preprocess a tree T (with n nodes) to answer lowest common ancestor queries. . .



Lowest common ancestor

Preprocess a tree T (with n nodes) to answer lowest common ancestor queries. . .

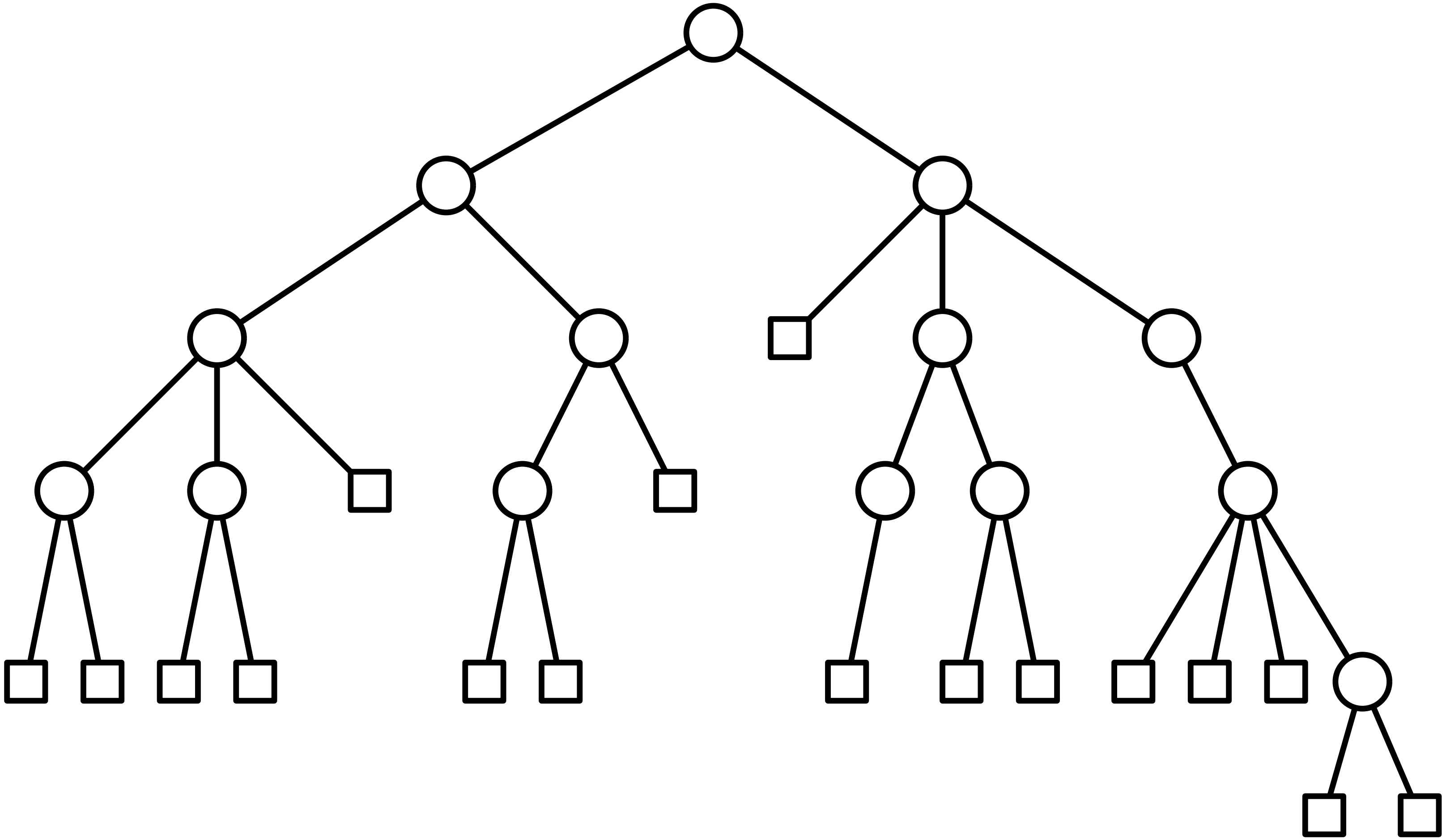


After preprocessing,

the output to a query LCA(i, j) is the lowest common ancestor of nodes i and j

Lowest common ancestor

Preprocess a tree T (with n nodes) to answer lowest common ancestor queries. . .

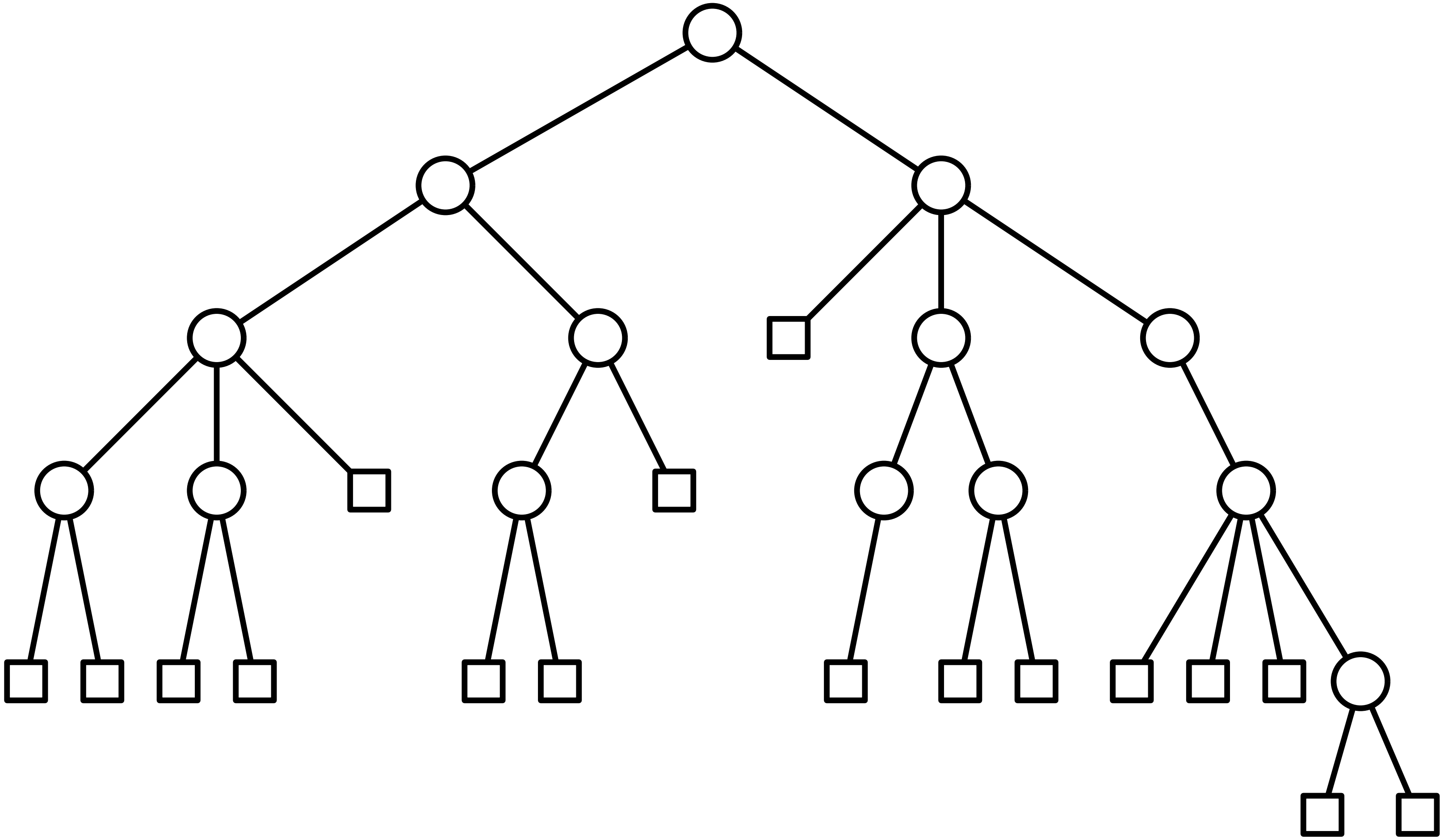


After preprocessing,

the output to a query LCA(i, j) is the lowest common ancestor of nodes i and j

Lowest common ancestor

Preprocess a tree T (with n nodes) to answer lowest common ancestor queries...

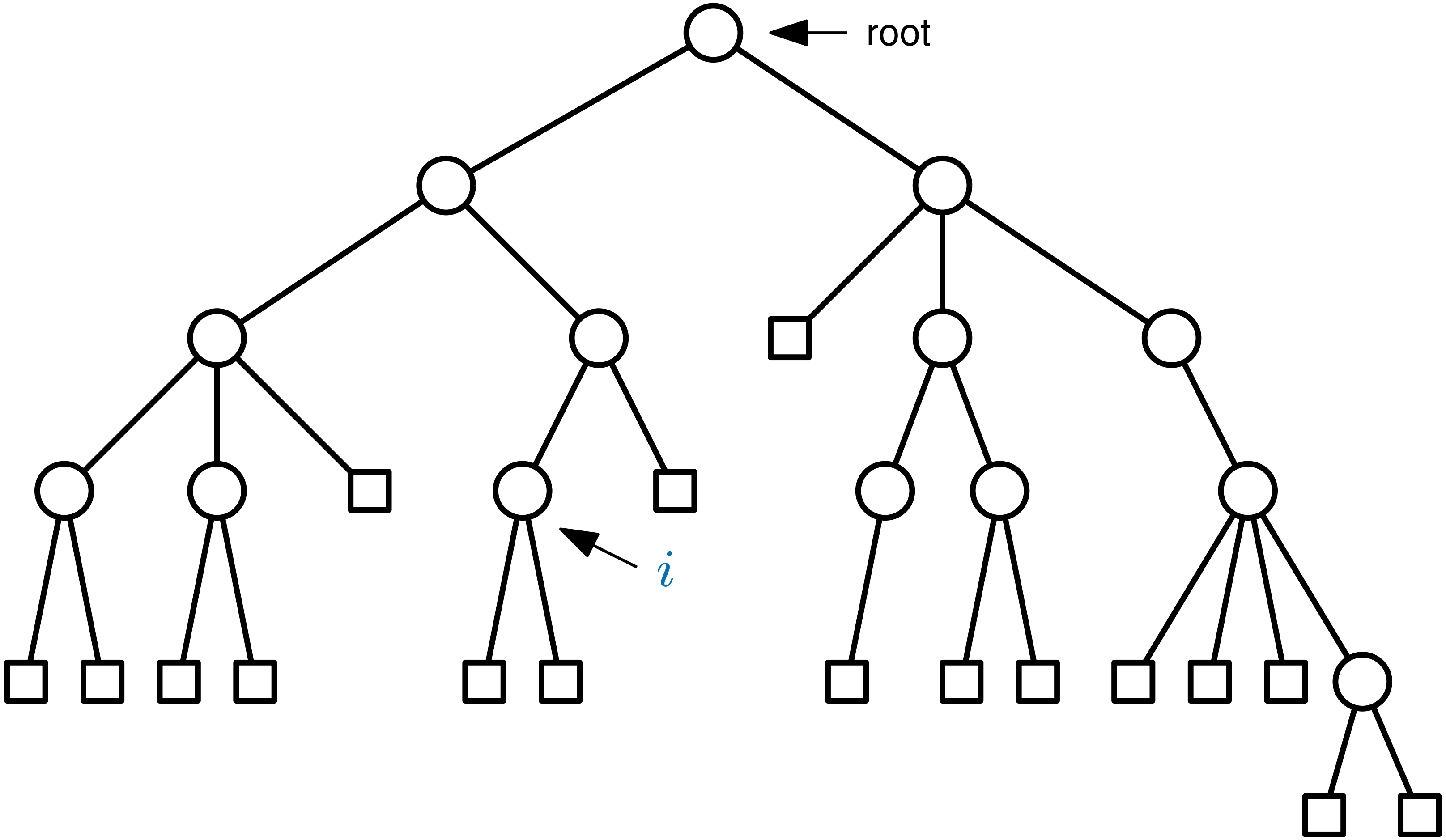


After preprocessing,

the output to a query LCA(i, j) is the lowest common ancestor of nodes i and j

Lowest common ancestor

Preprocess a tree T (with n nodes) to answer lowest common ancestor queries. . .

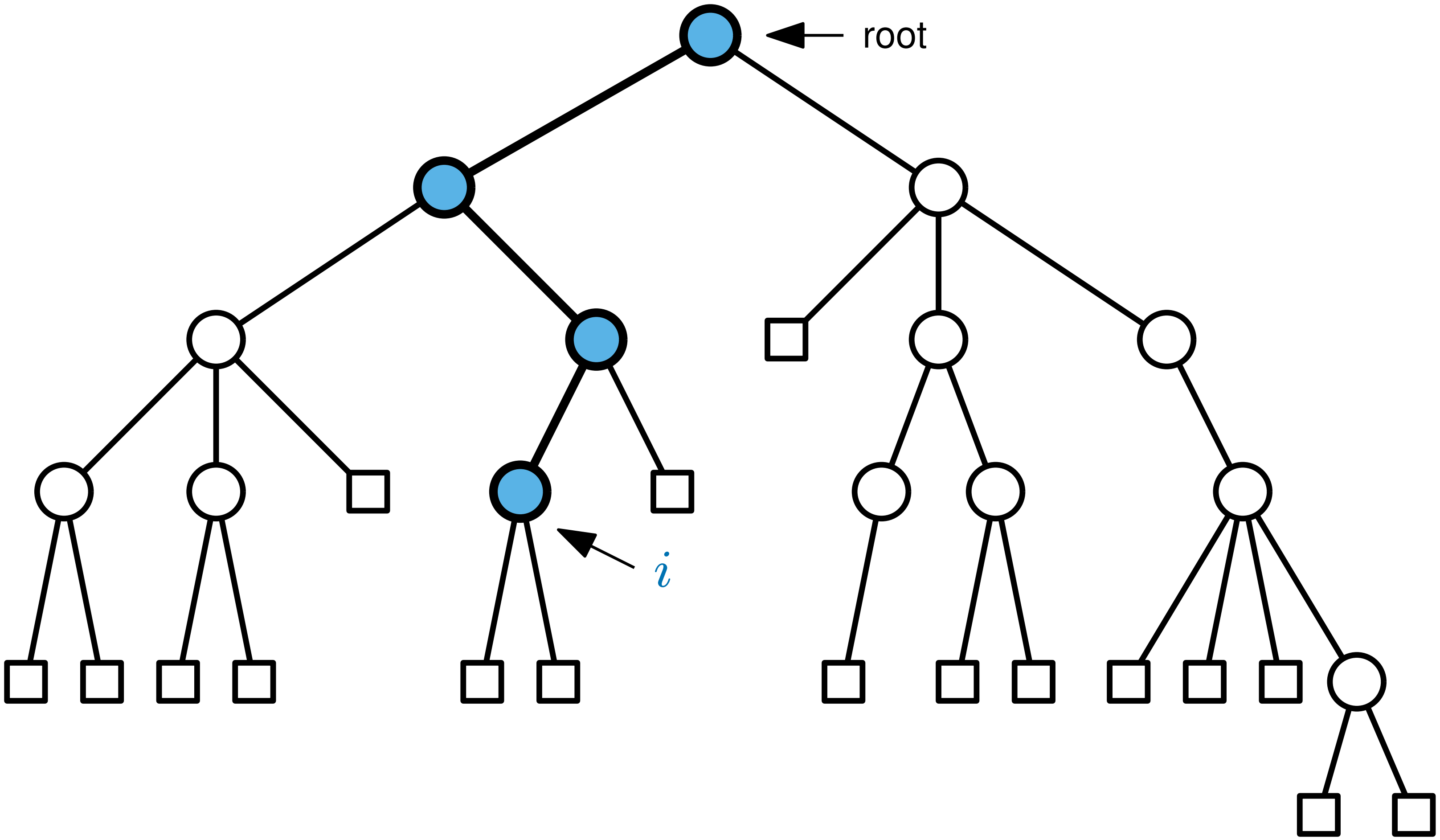


After preprocessing,

the output to a query LCA(i, j) is the lowest common ancestor of nodes i and j

Lowest common ancestor

Preprocess a tree T (with n nodes) to answer lowest common ancestor queries...

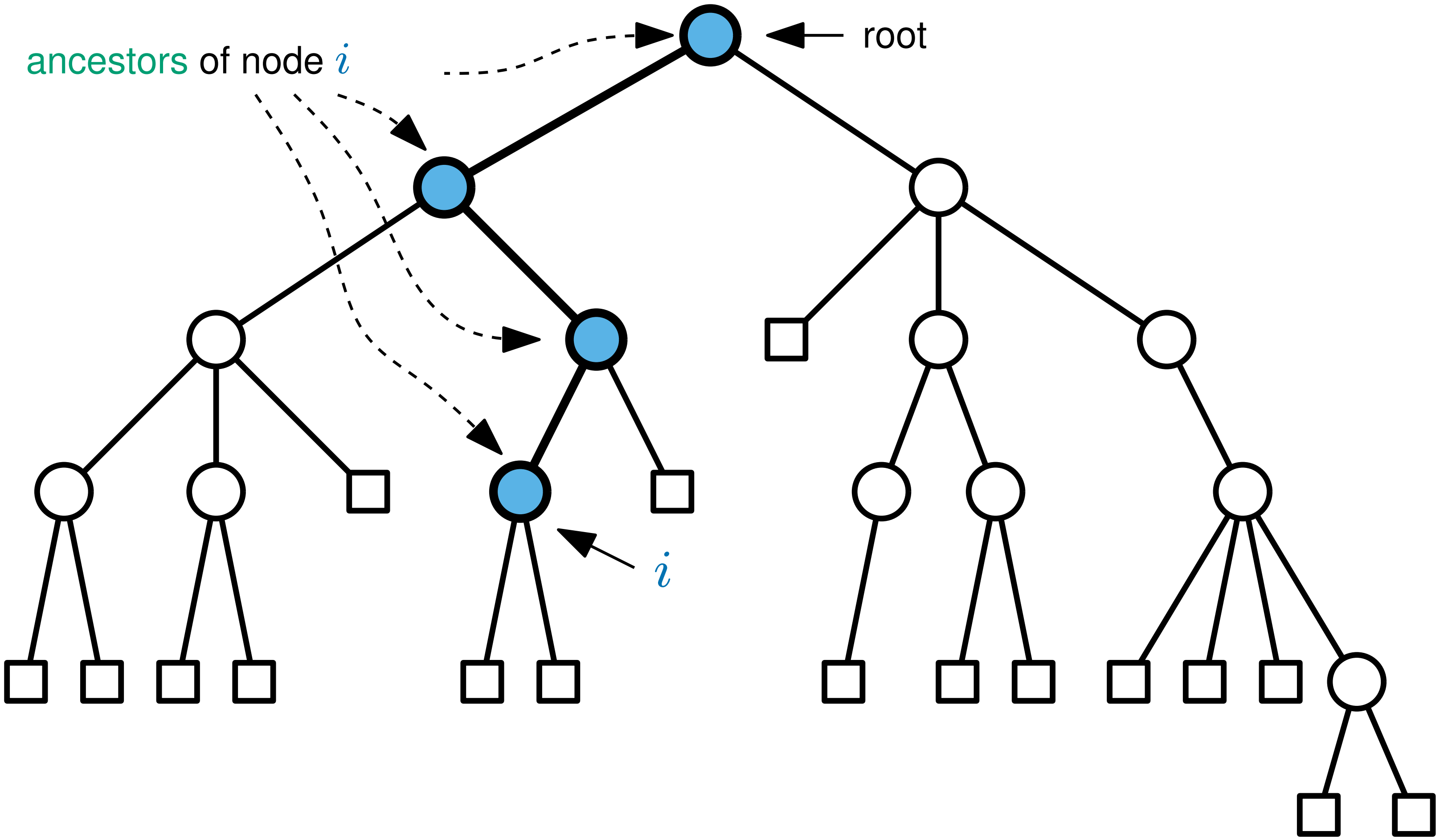


After preprocessing,

the output to a query LCA(i, j) is the lowest common ancestor of nodes i and j

Lowest common ancestor

Preprocess a tree T (with n nodes) to answer lowest common ancestor queries. . .

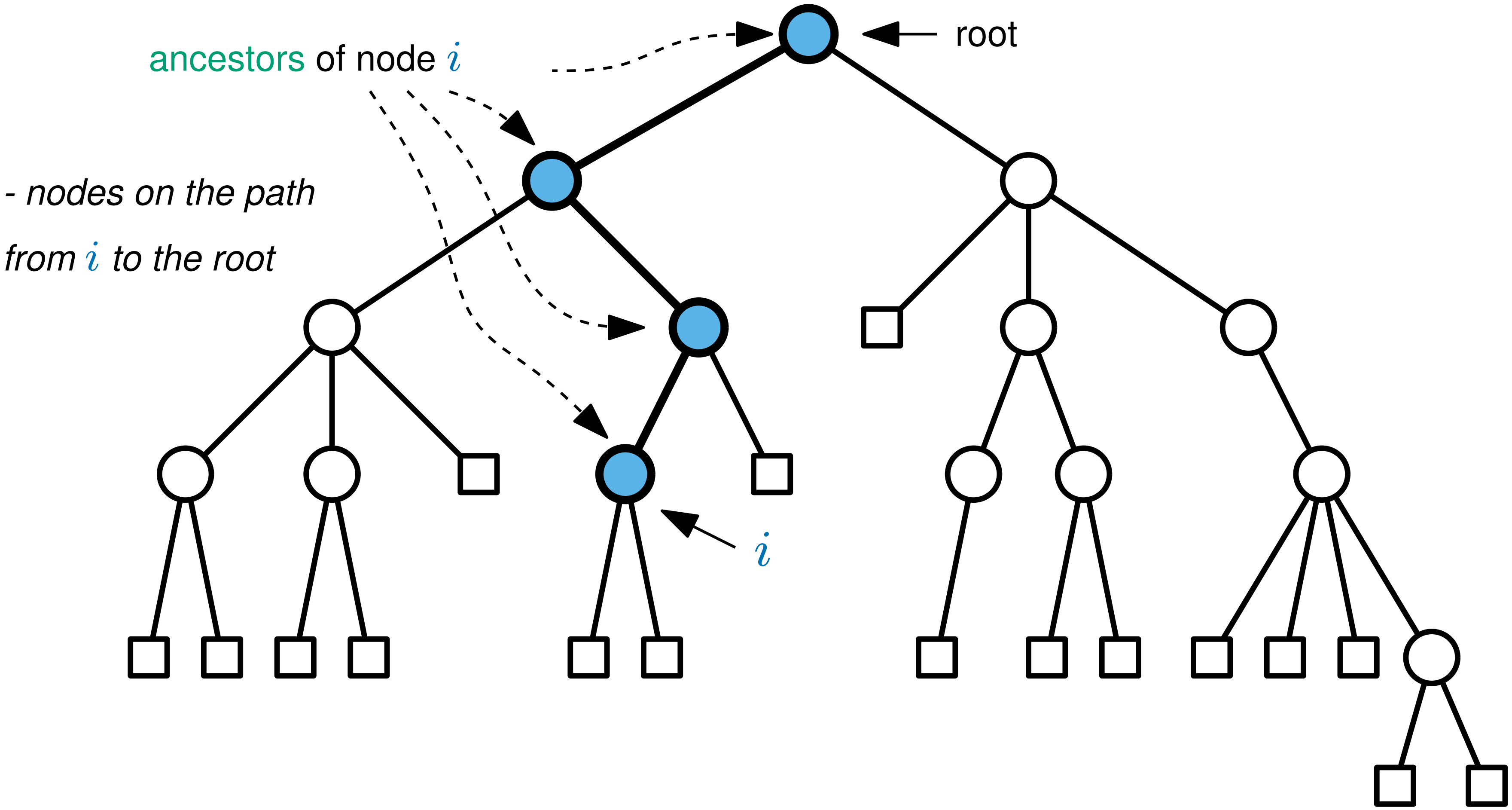


After preprocessing,

the output to a query LCA(i, j) is the lowest common ancestor of nodes i and j

Lowest common ancestor

Preprocess a tree T (with n nodes) to answer lowest common ancestor queries. . .

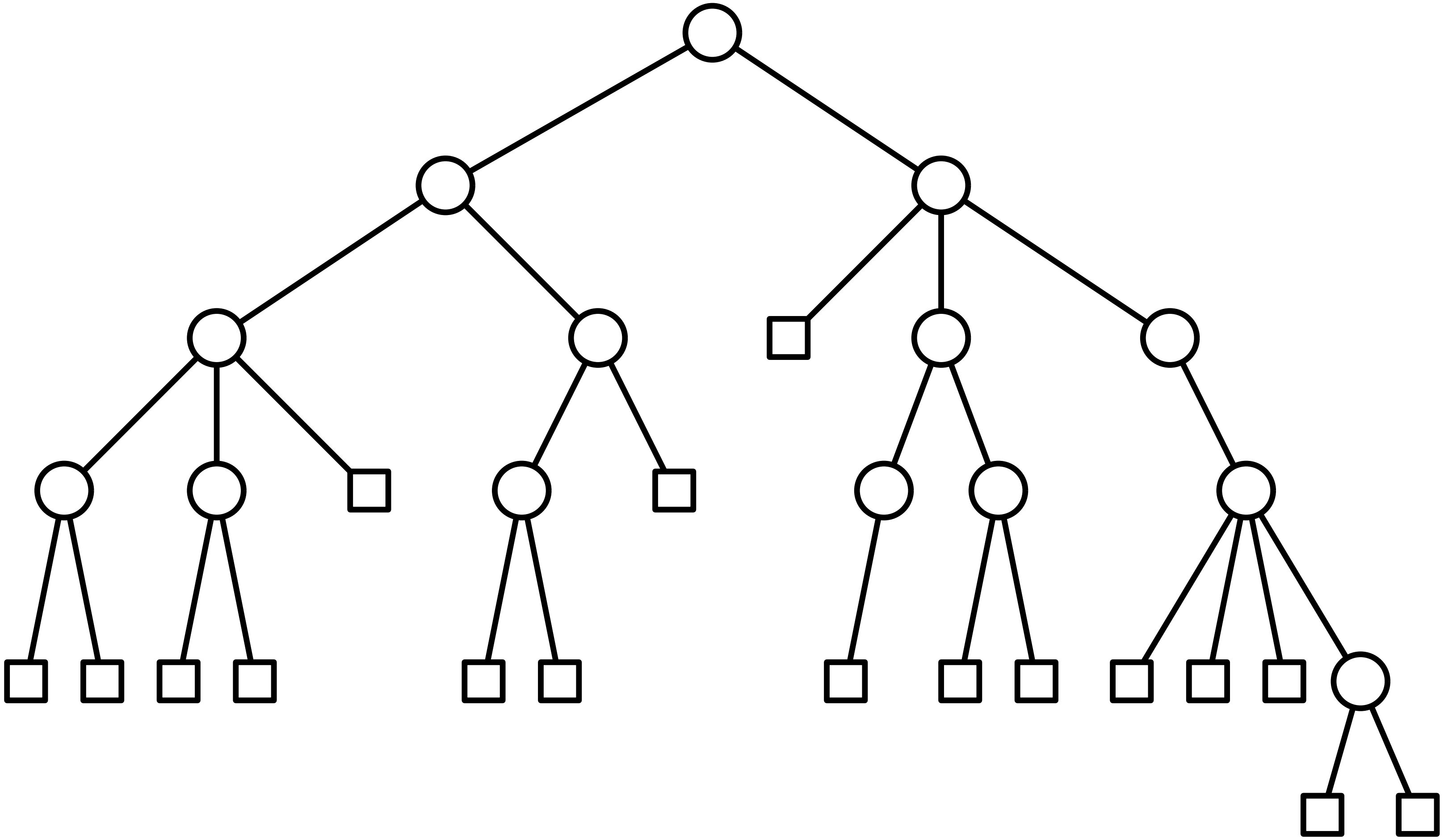


After preprocessing,

the output to a query LCA(i, j) is the lowest common ancestor of nodes i and j

Lowest common ancestor

Preprocess a tree T (with n nodes) to answer lowest common ancestor queries. . .

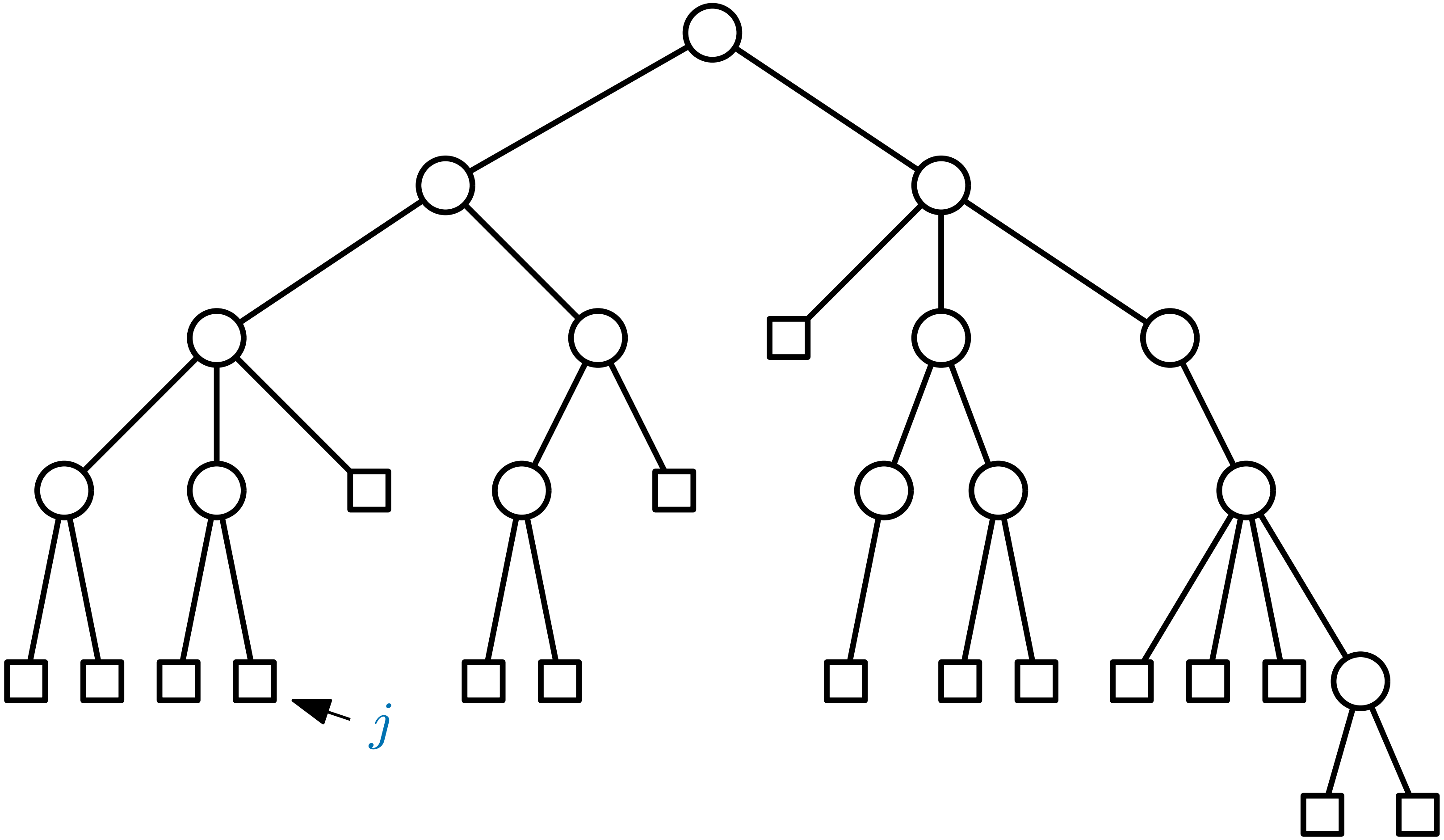


After preprocessing,

the output to a query LCA(i, j) is the lowest common ancestor of nodes i and j

Lowest common ancestor

Preprocess a tree T (with n nodes) to answer lowest common ancestor queries...

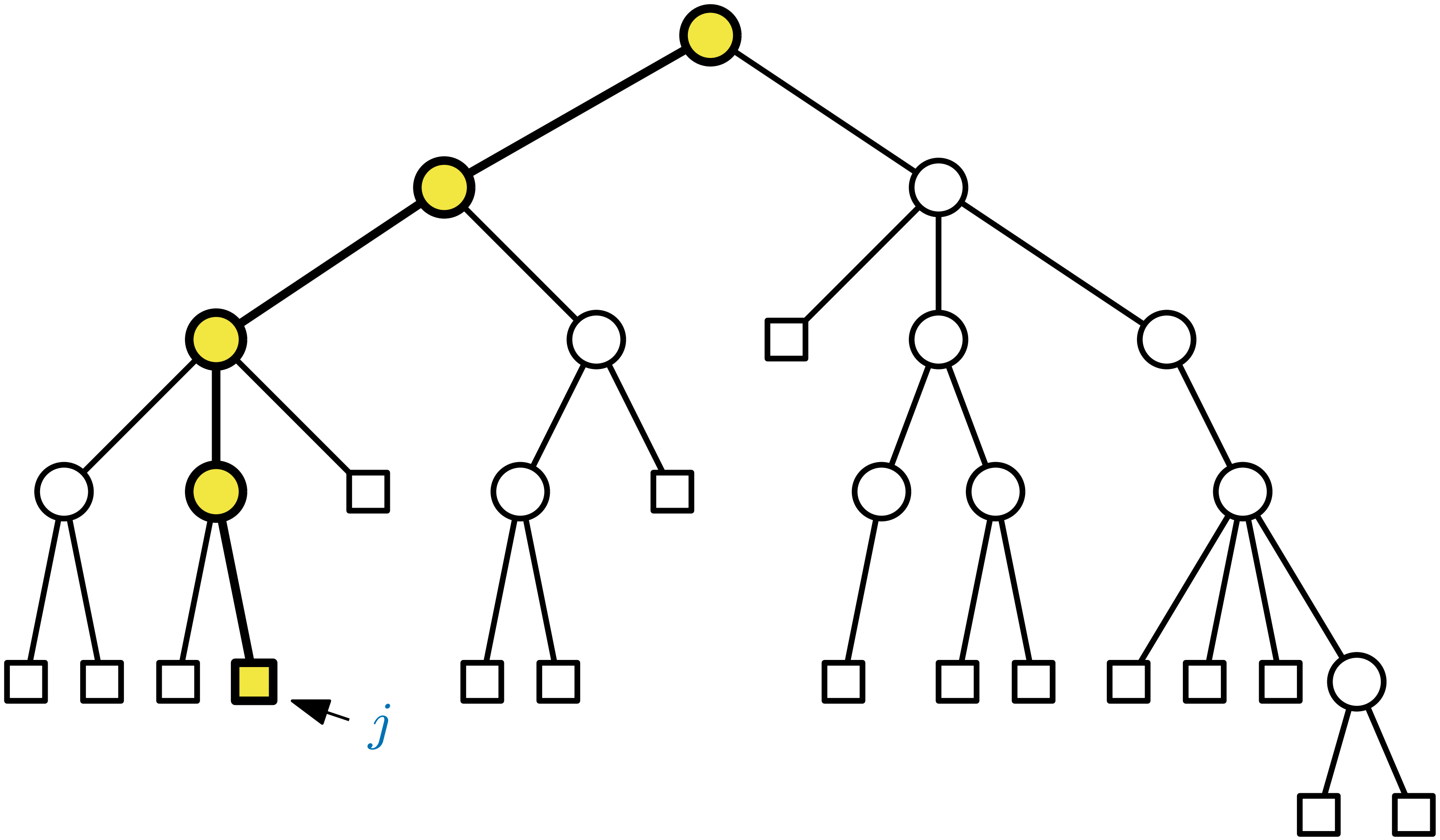


After preprocessing,

the output to a query LCA(i, j) is the lowest common ancestor of nodes i and j

Lowest common ancestor

Preprocess a tree T (with n nodes) to answer lowest common ancestor queries. . .

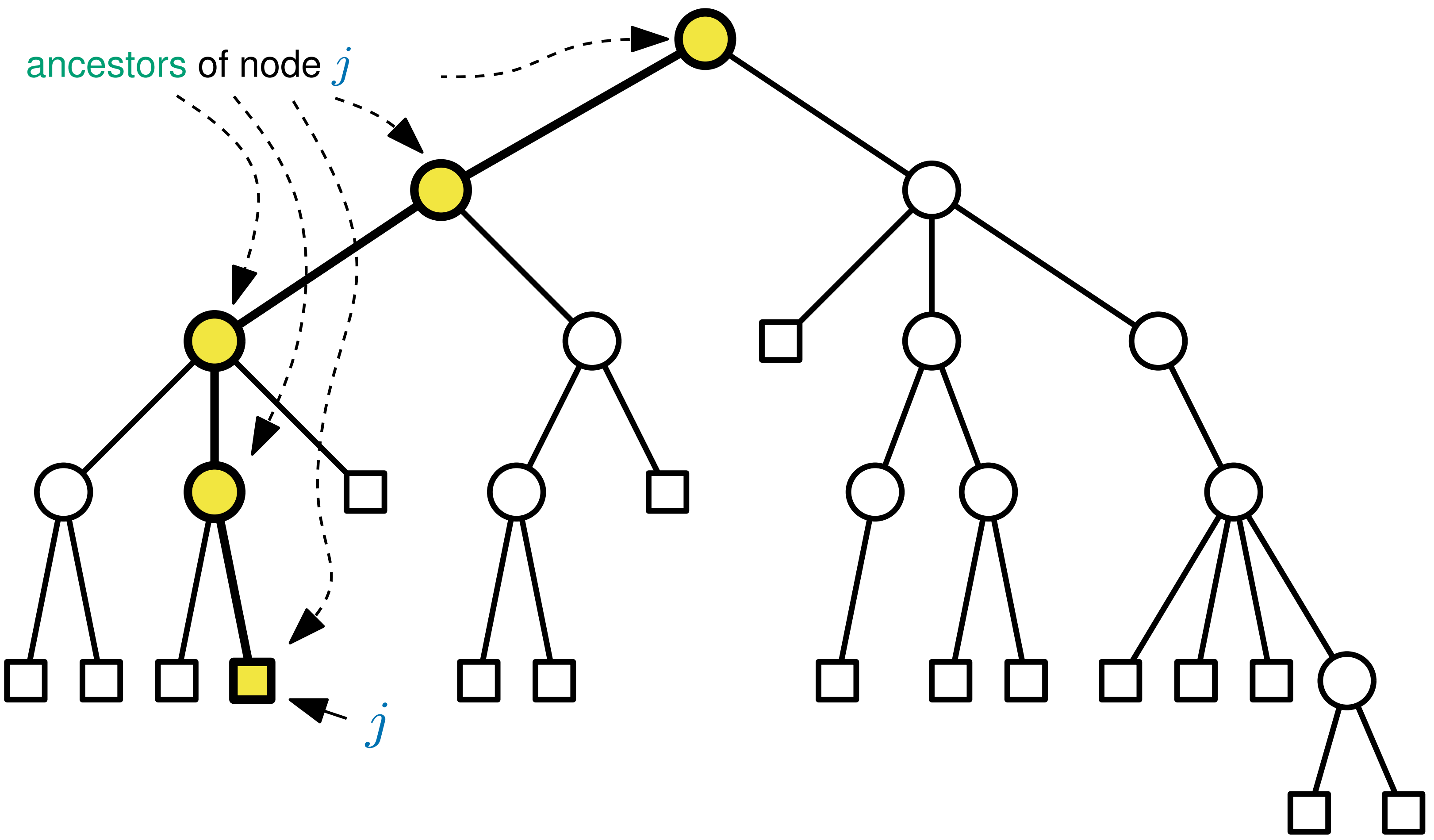


After preprocessing,

the output to a query LCA(i, j) is the lowest common ancestor of nodes i and j

Lowest common ancestor

Preprocess a tree T (with n nodes) to answer lowest common ancestor queries...

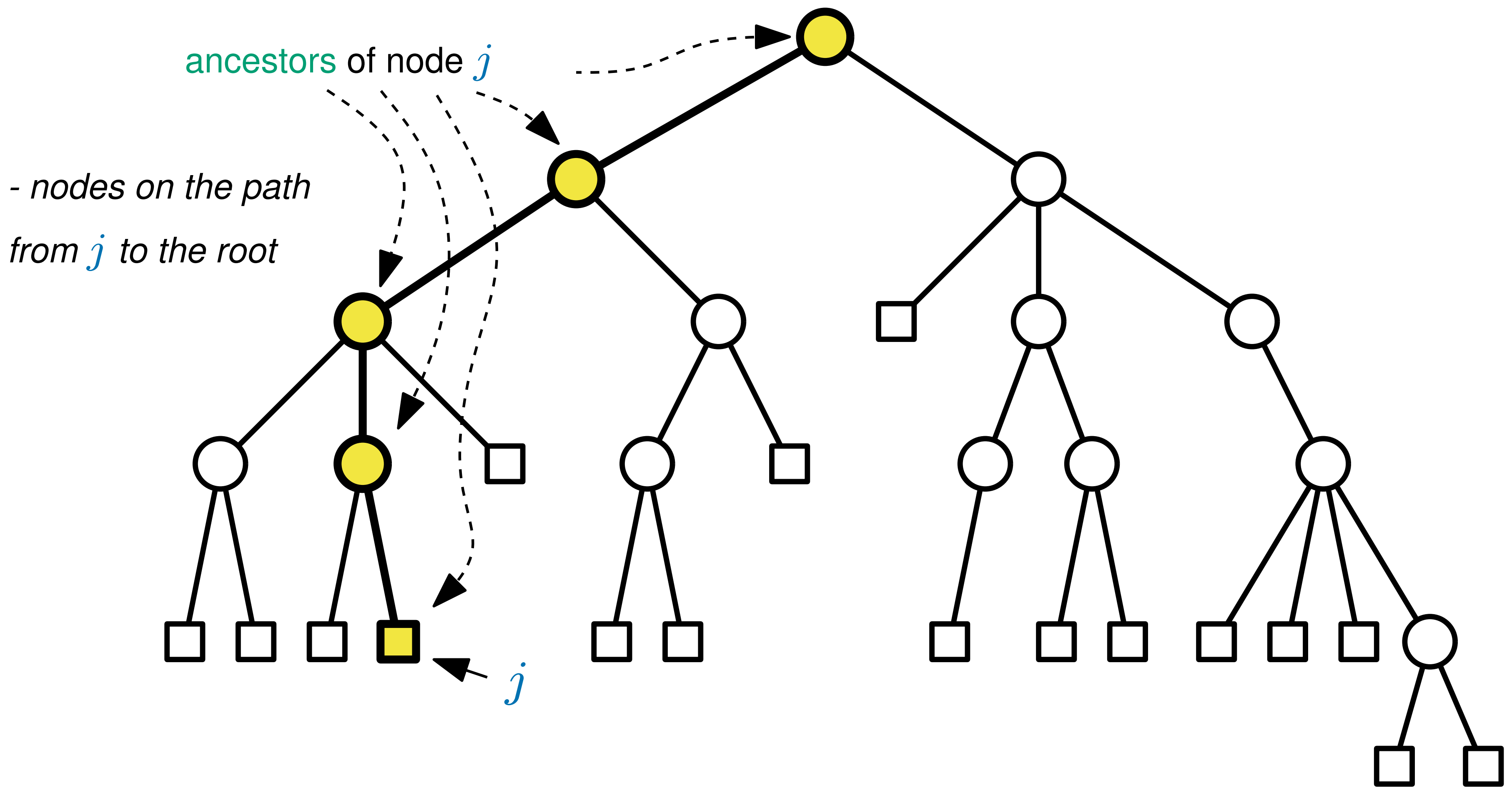


After preprocessing,

the output to a query $\text{LCA}(i, j)$ is the lowest common ancestor of nodes i and j

Lowest common ancestor

Preprocess a tree T (with n nodes) to answer lowest common ancestor queries. . .

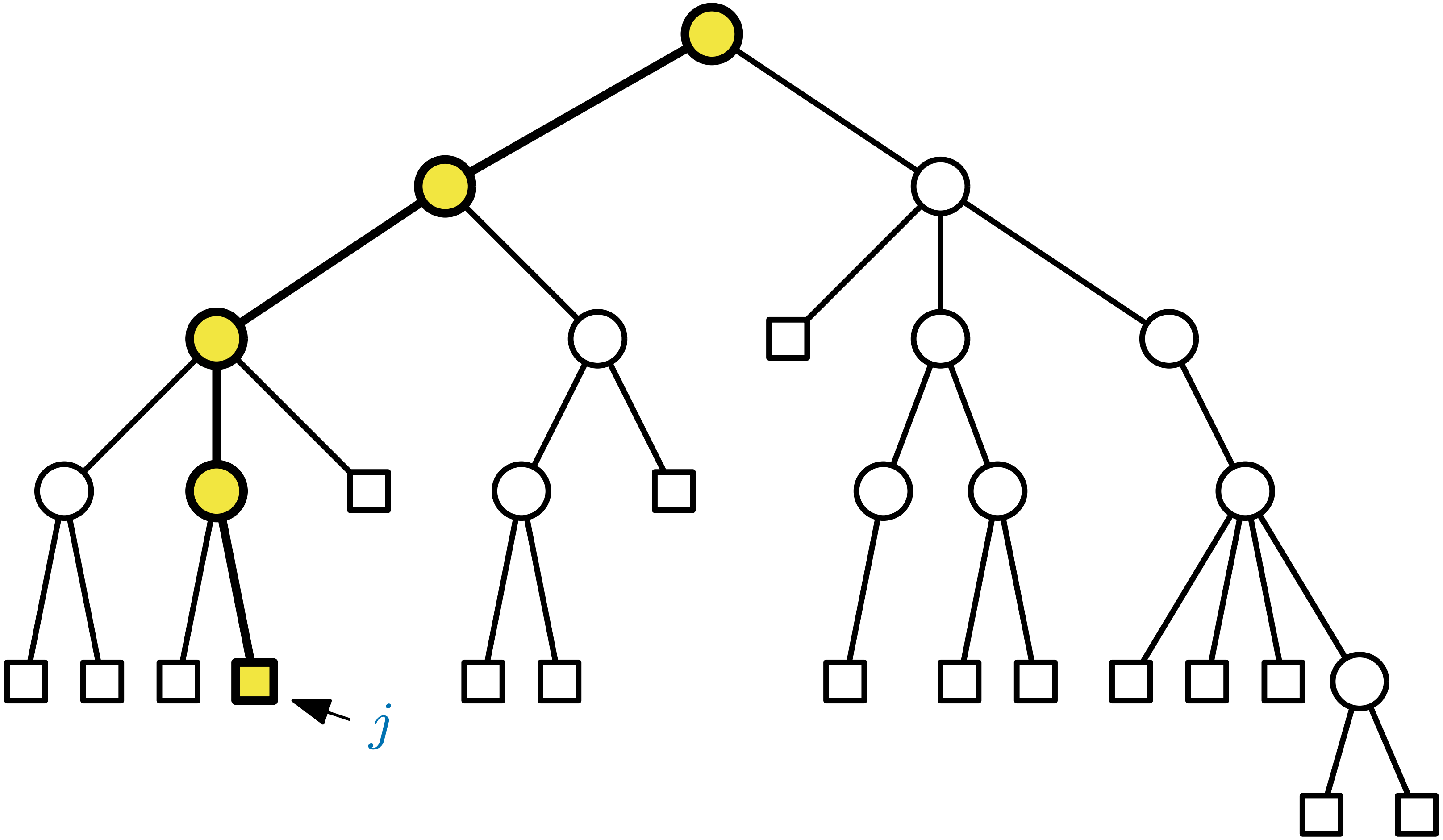


After preprocessing,

the output to a query $\text{LCA}(i, j)$ is the lowest common ancestor of nodes i and j

Lowest common ancestor

Preprocess a tree T (with n nodes) to answer lowest common ancestor queries. . .

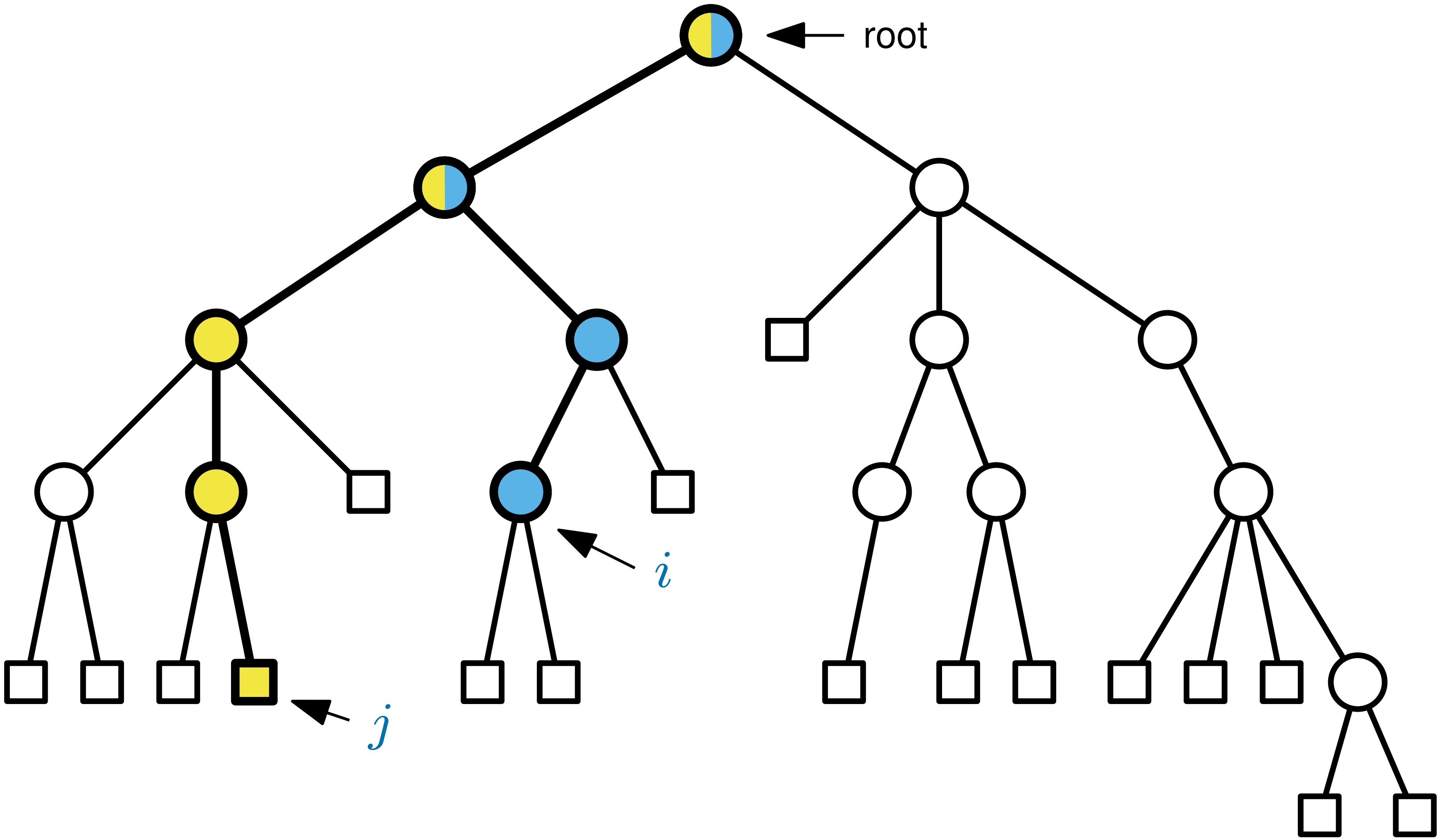


After preprocessing,

the output to a query LCA(i, j) is the lowest common ancestor of nodes i and j

Lowest common ancestor

Preprocess a tree T (with n nodes) to answer lowest common ancestor queries...

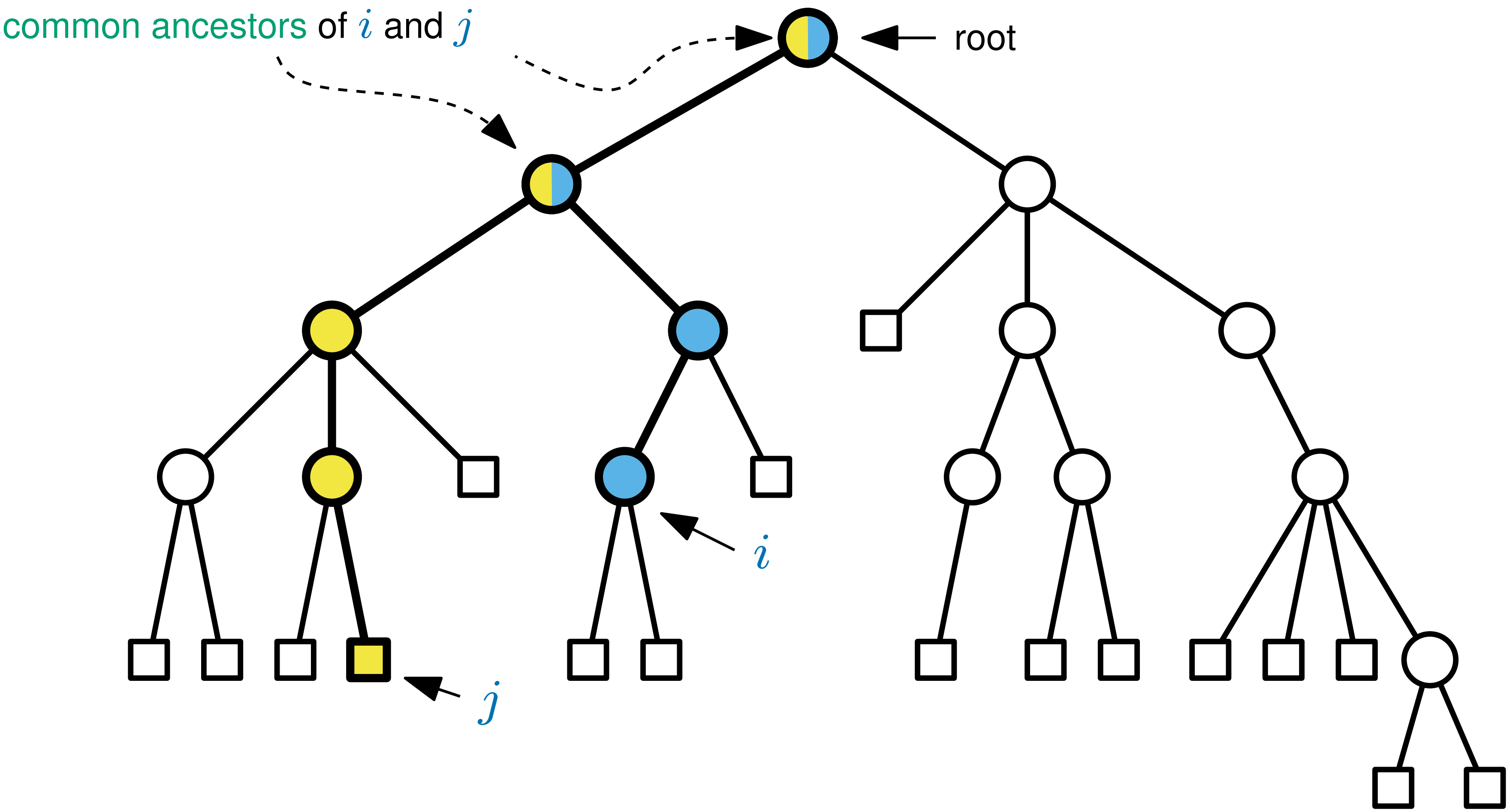


After preprocessing,

the output to a query LCA(i, j) is the lowest common ancestor of nodes i and j

Lowest common ancestor

Preprocess a tree T (with n nodes) to answer lowest common ancestor queries. . .

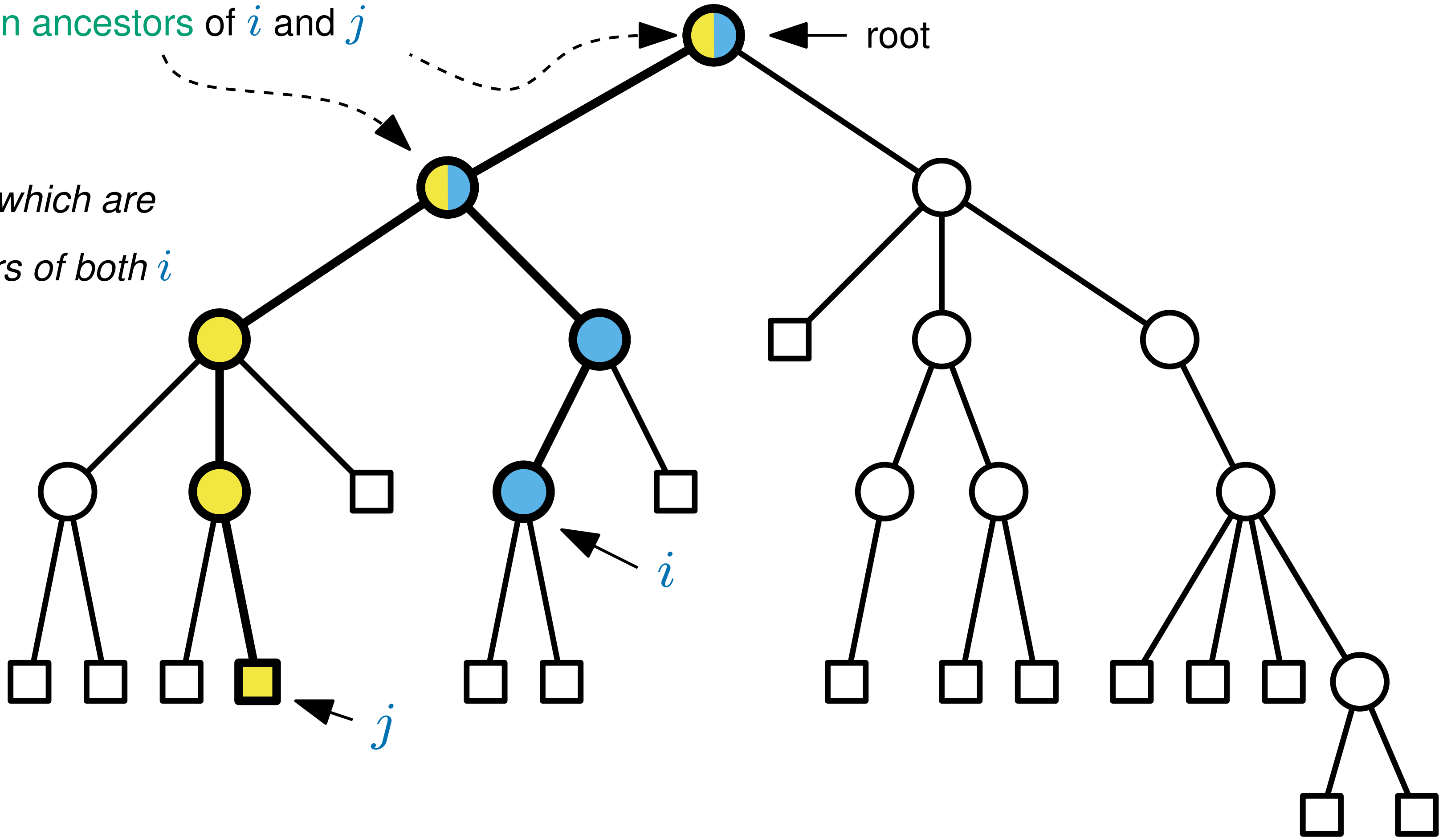


After preprocessing,

the output to a query LCA(i, j) is the lowest common ancestor of nodes i and j

Lowest common ancestor

Preprocess a tree T (with n nodes) to answer lowest common ancestor queries. . .



common ancestors of i and j

- nodes which are ancestors of both i and j

root

i

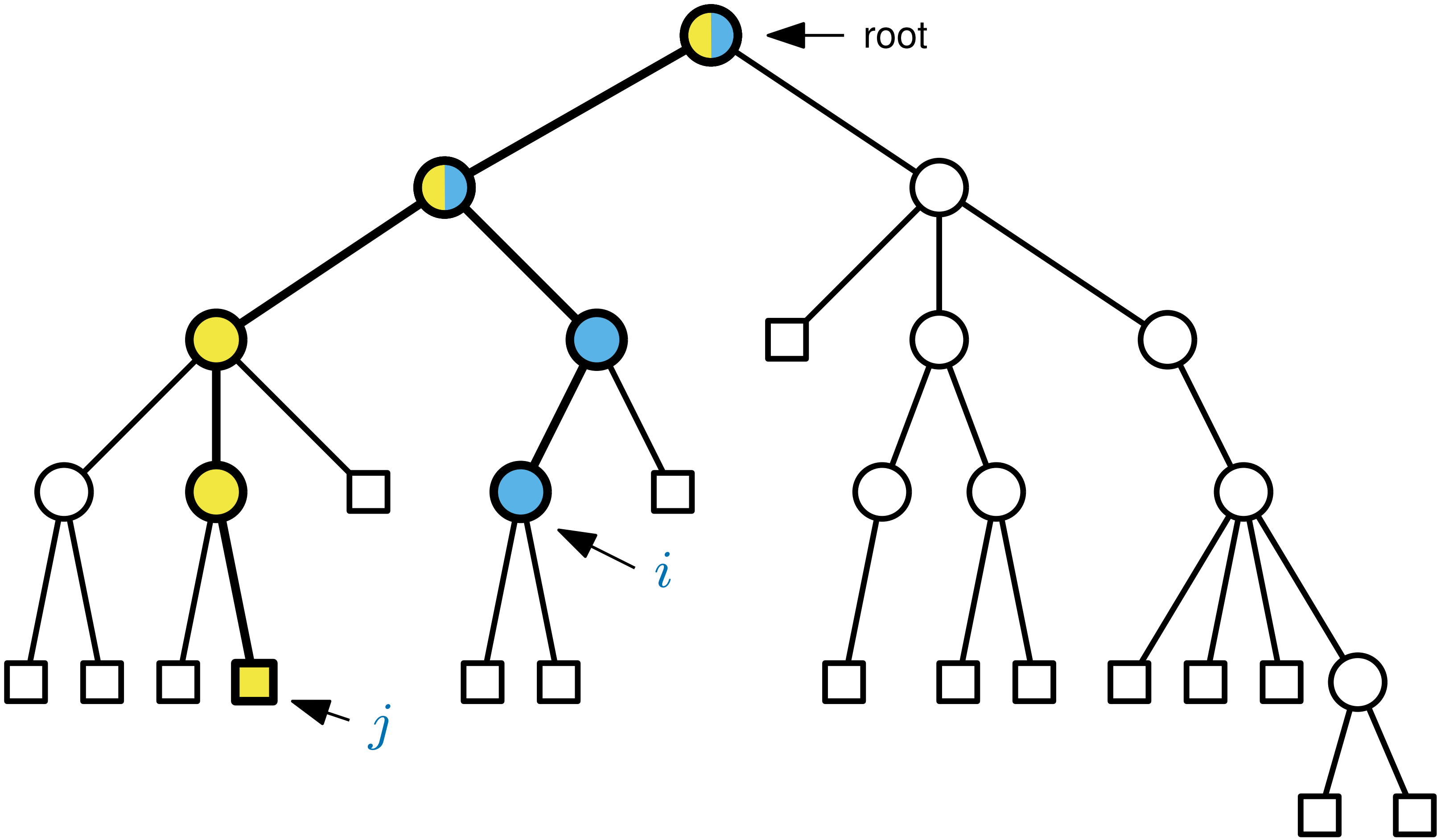
j

After preprocessing,

the output to a query LCA(i, j) is the lowest common ancestor of nodes i and j

Lowest common ancestor

Preprocess a tree T (with n nodes) to answer lowest common ancestor queries. . .

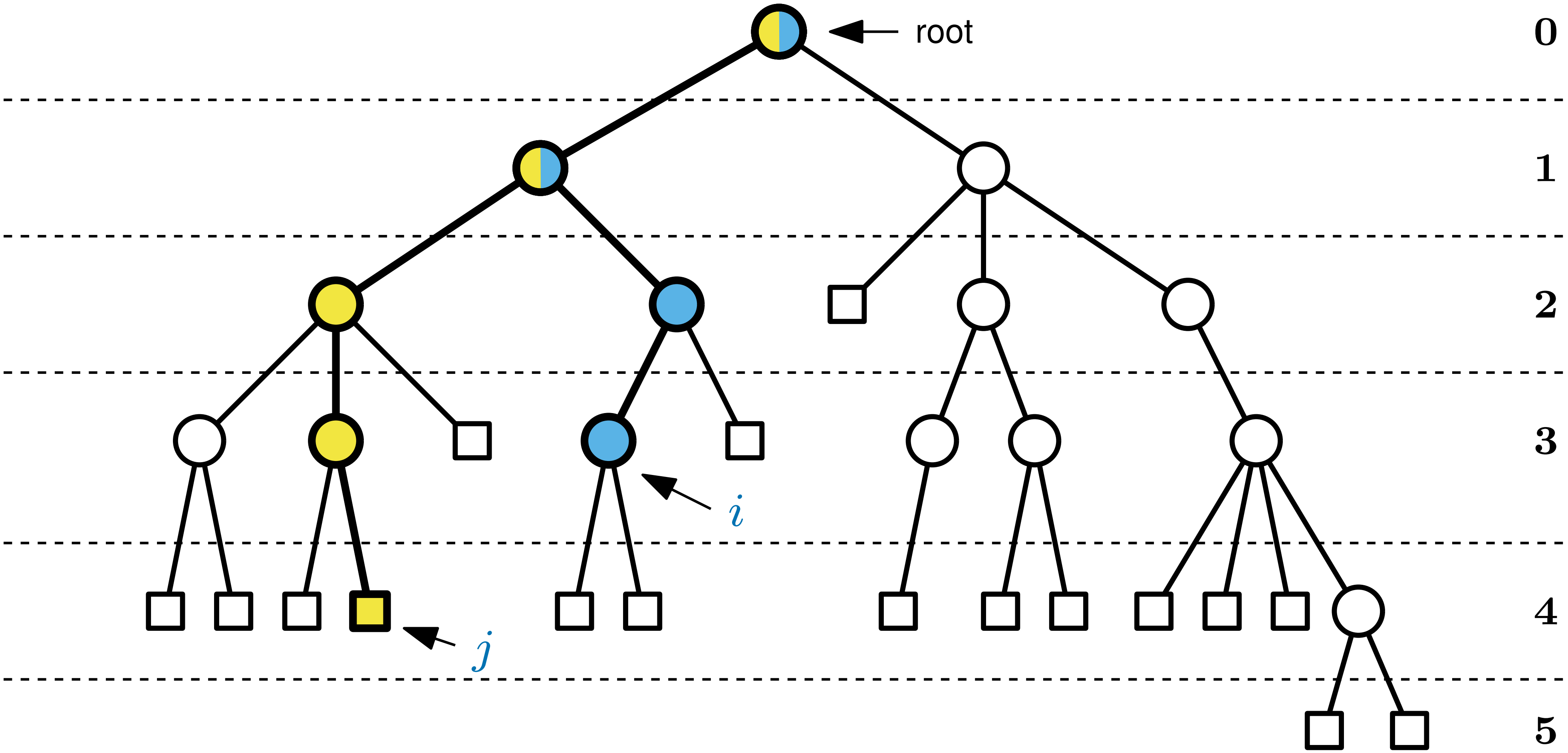


After preprocessing,

the output to a query LCA(i, j) is the lowest common ancestor of nodes i and j

Lowest common ancestor

Preprocess a tree T (with n nodes) to answer lowest common ancestor queries. . .

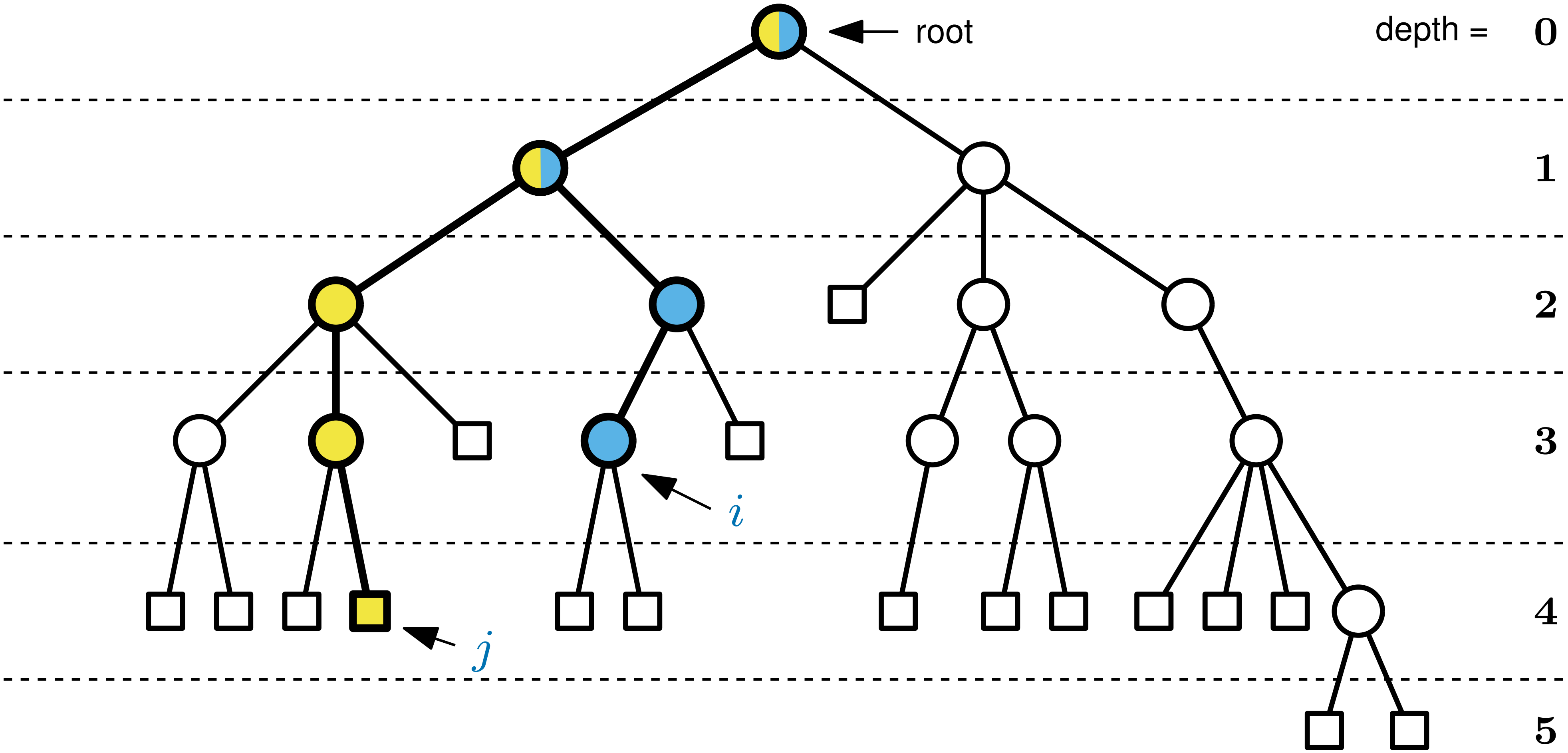


After preprocessing,

the output to a query $\text{LCA}(i, j)$ is the lowest common ancestor of nodes i and j

Lowest common ancestor

Preprocess a tree T (with n nodes) to answer lowest common ancestor queries. . .

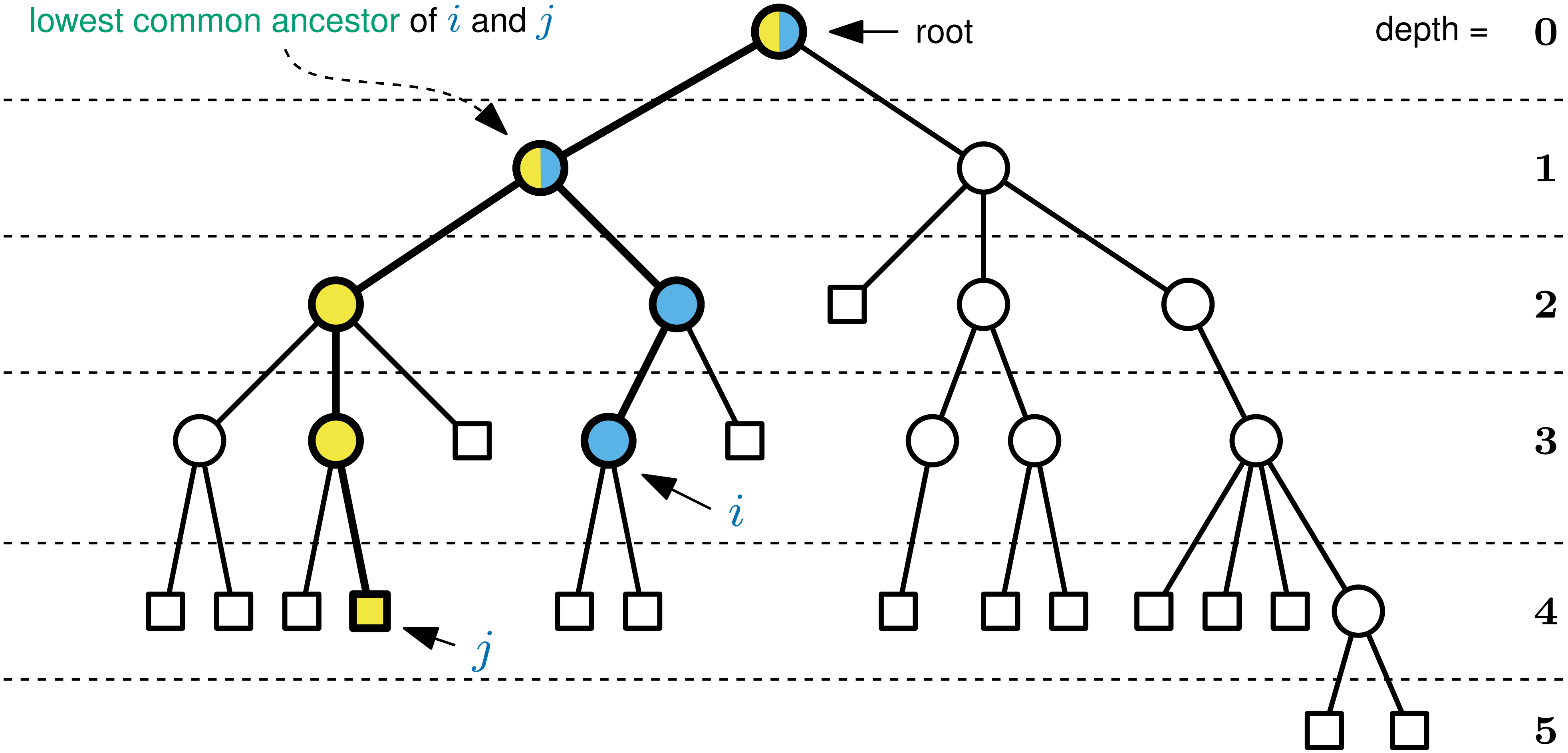


After preprocessing,

the output to a query LCA(i, j) is the lowest common ancestor of nodes i and j

Lowest common ancestor

Preprocess a tree T (with n nodes) to answer lowest common ancestor queries...

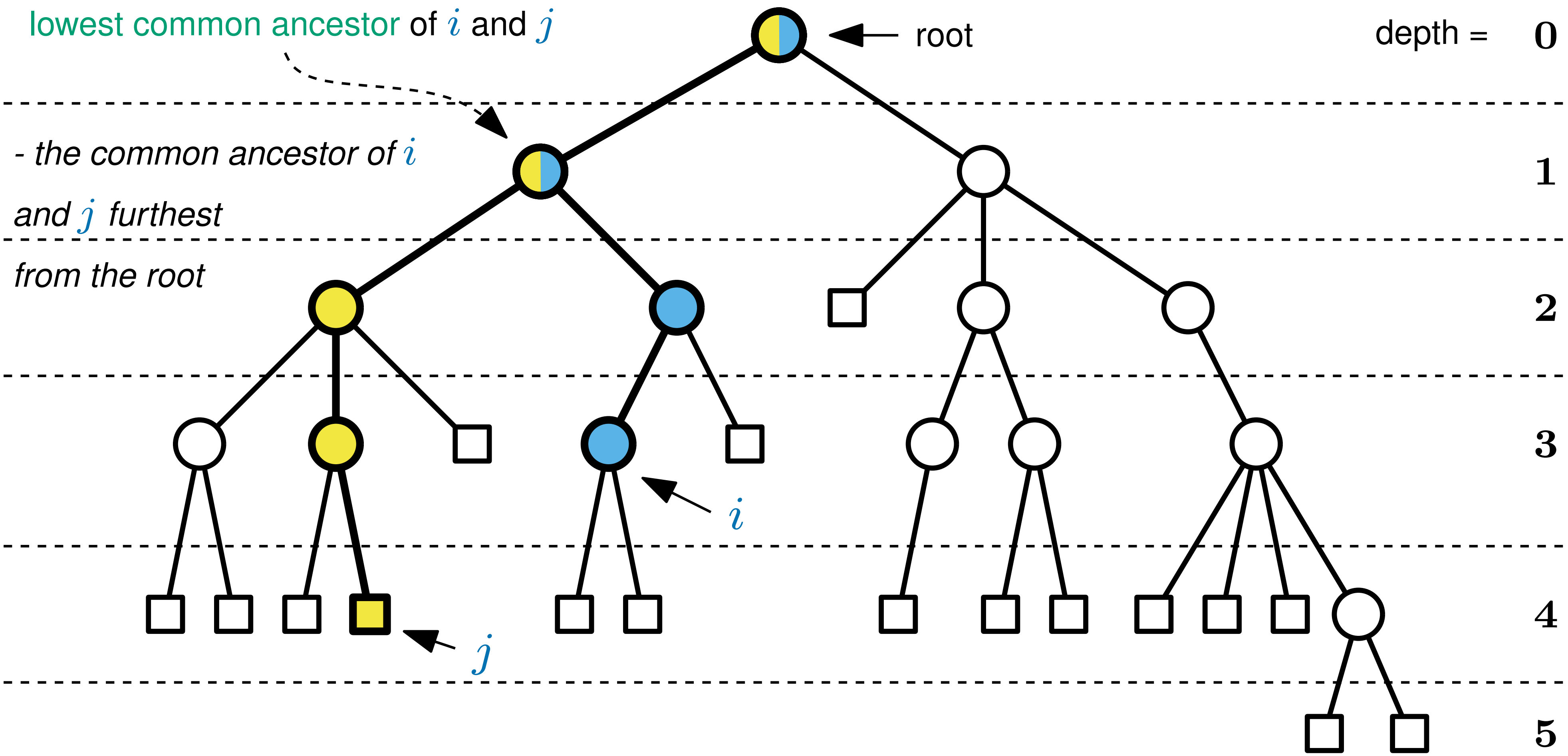


After preprocessing,

the output to a query LCA(i, j) is the lowest common ancestor of nodes i and j

Lowest common ancestor

Preprocess a tree T (with n nodes) to answer lowest common ancestor queries. . .

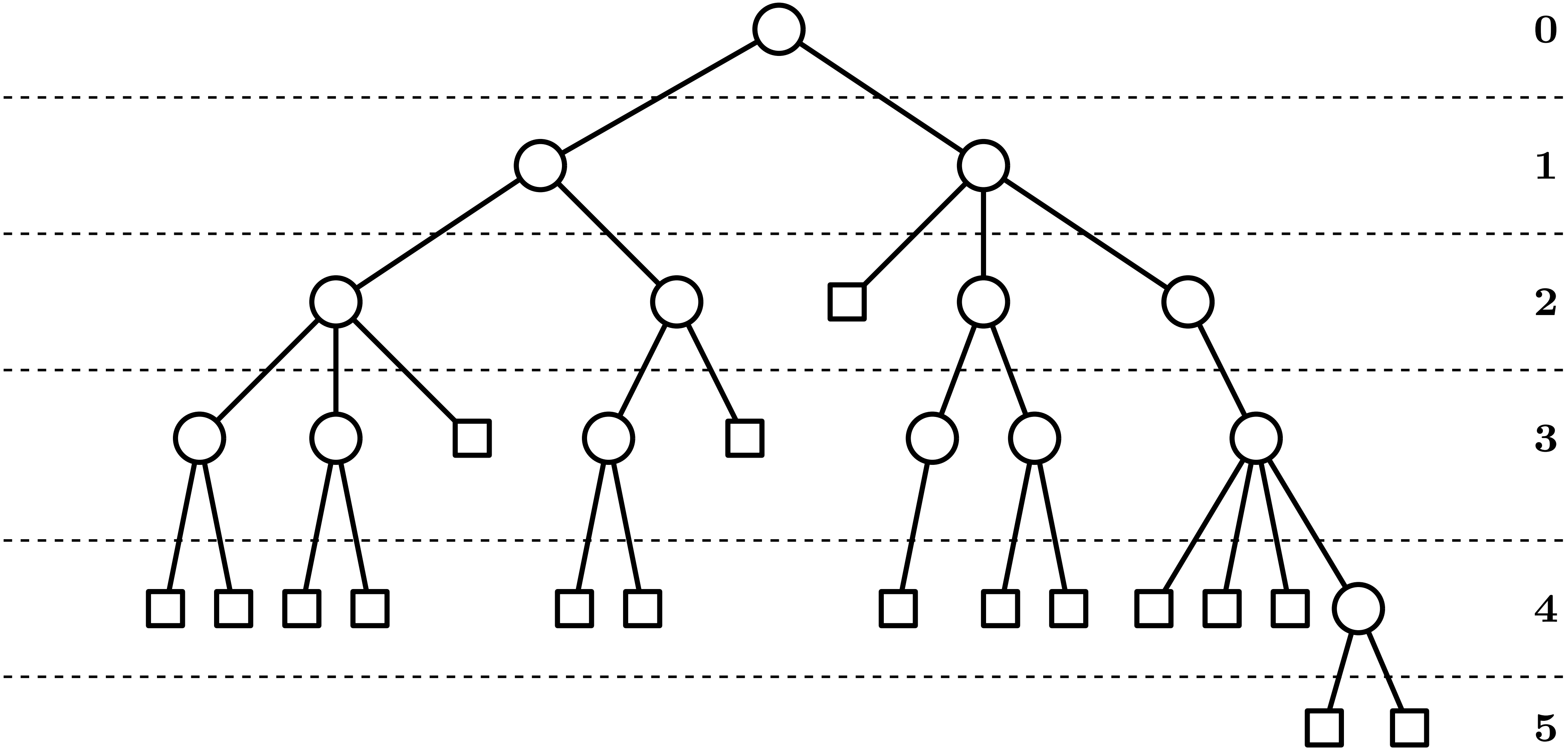

lowest common ancestor of i and j
- the common ancestor of i
and j furthest
from the root
root
depth =
0
1
2
3
4
5
i
j

After preprocessing,

the output to a query LCA(i, j) is the lowest common ancestor of nodes i and j

Lowest common ancestor

Preprocess a tree T (with n nodes) to answer lowest common ancestor queries...

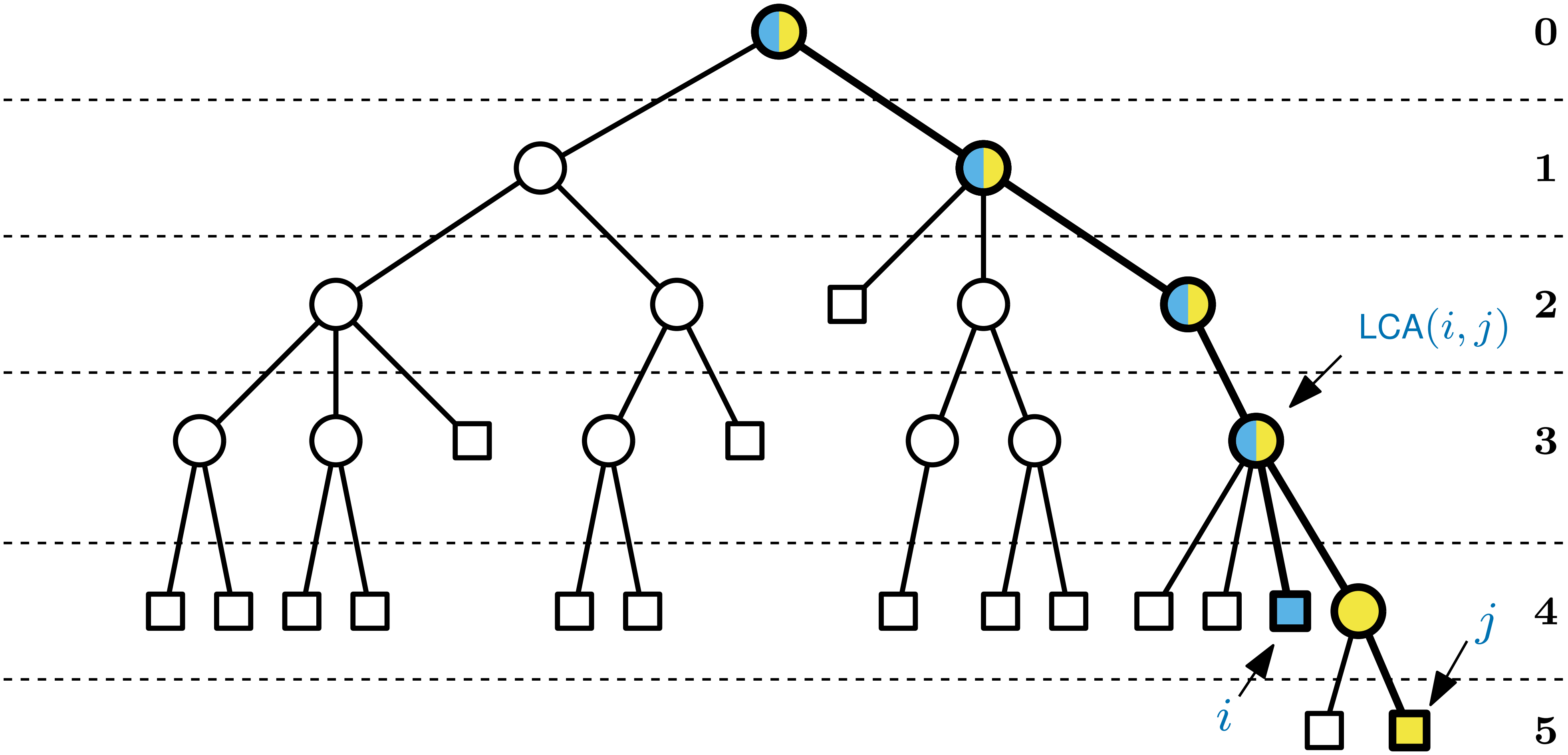


After preprocessing,

the output to a query LCA(i, j) is the lowest common ancestor of nodes i and j

Lowest common ancestor

Preprocess a tree T (with n nodes) to answer lowest common ancestor queries. . .

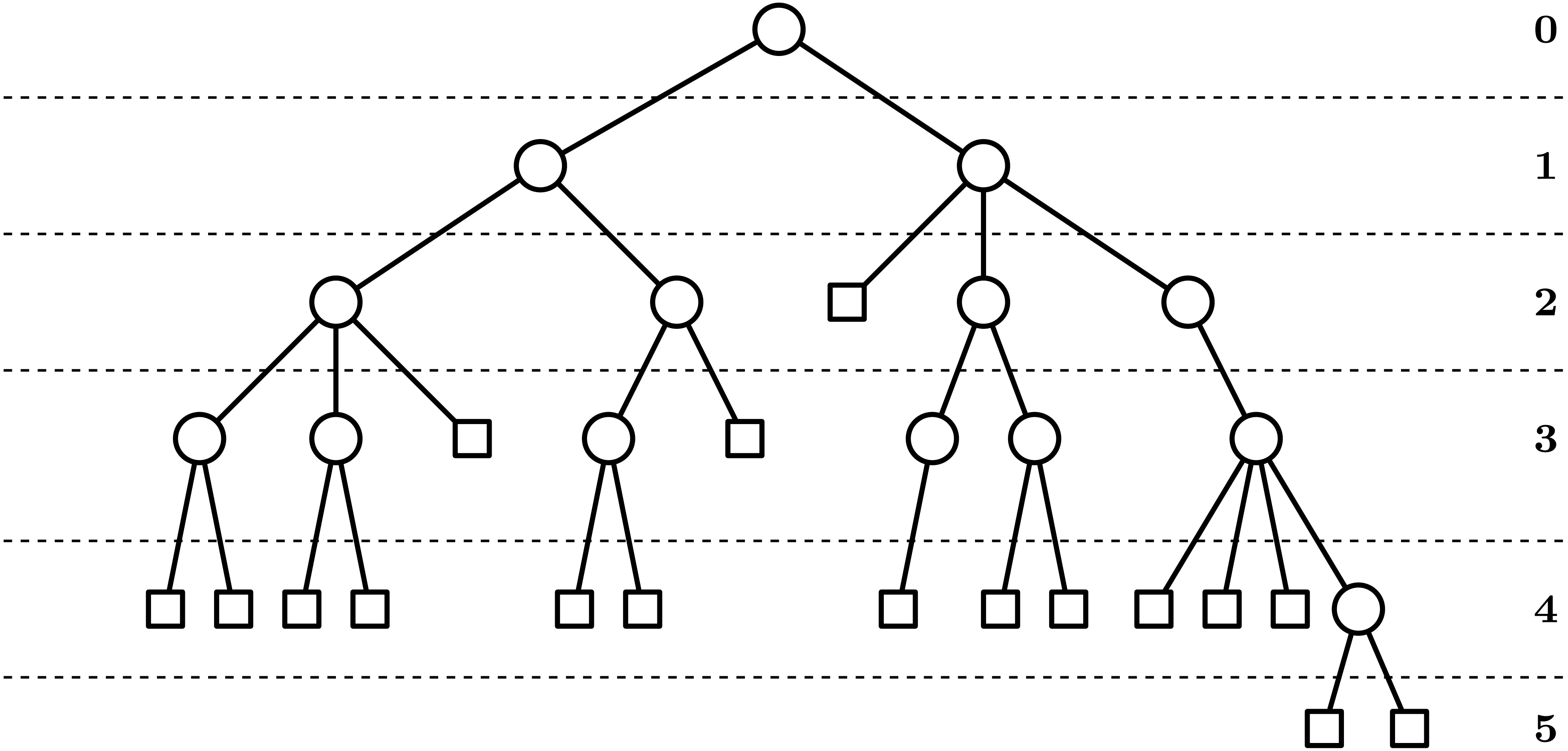


After preprocessing,

the output to a query LCA(i,j) is the lowest common ancestor of nodes i and j

Lowest common ancestor

Preprocess a tree T (with n nodes) to answer lowest common ancestor queries. . .

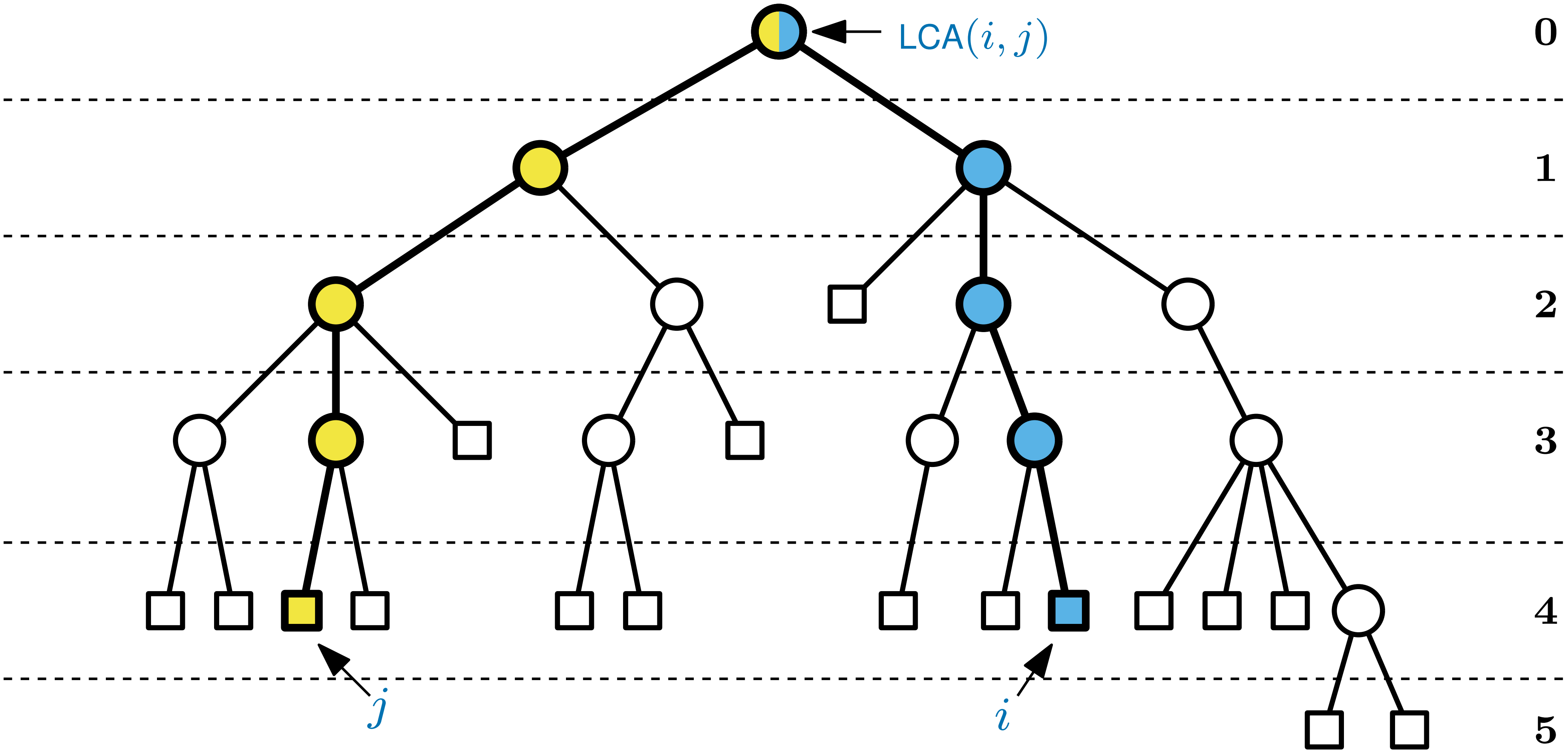


After preprocessing,

the output to a query LCA(i, j) is the lowest common ancestor of nodes i and j

Lowest common ancestor

Preprocess a tree T (with n nodes) to answer lowest common ancestor queries. . .

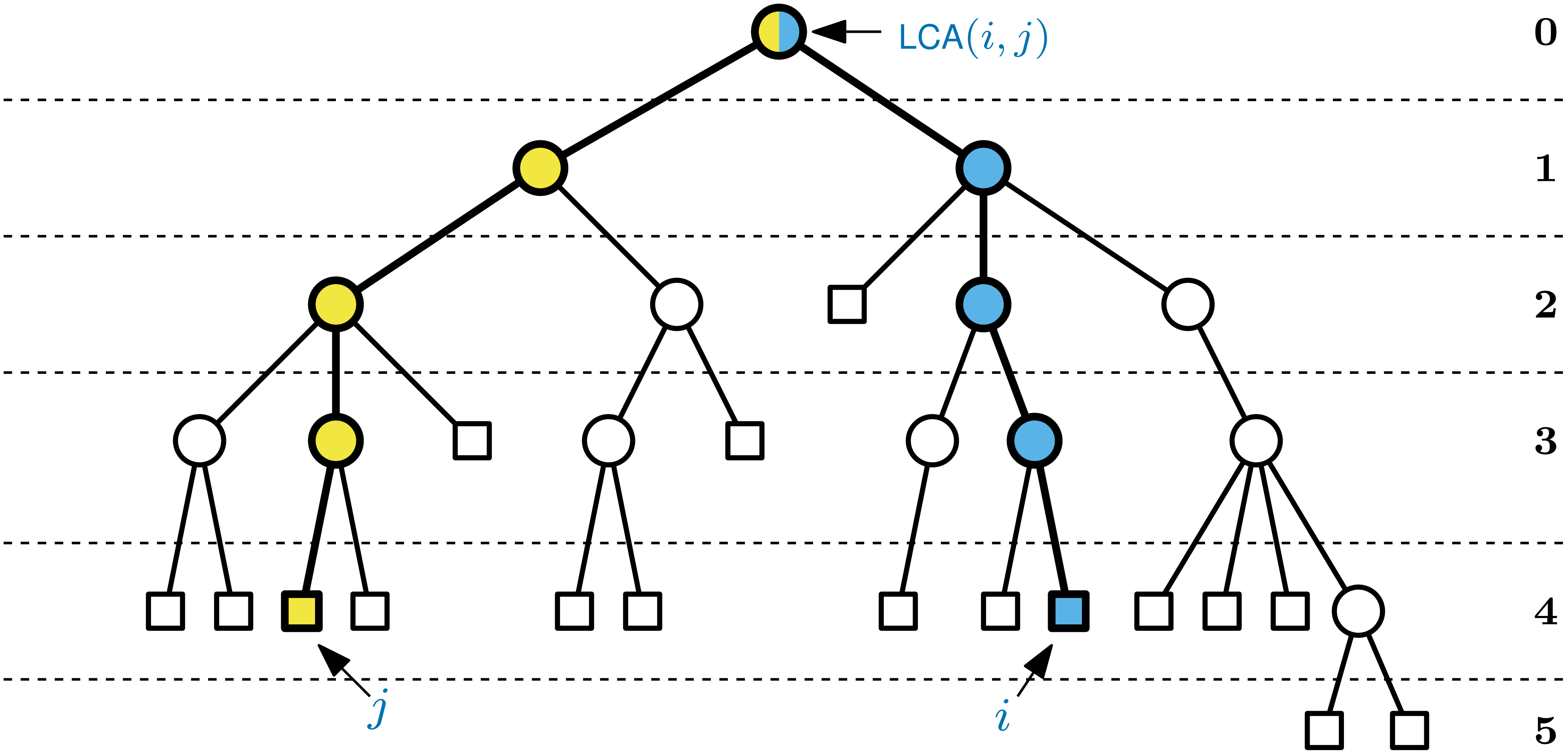


After preprocessing,

the output to a query LCA(i, j) is the lowest common ancestor of nodes i and j

Lowest common ancestor

Preprocess a tree T (with n nodes) to answer lowest common ancestor queries. . .

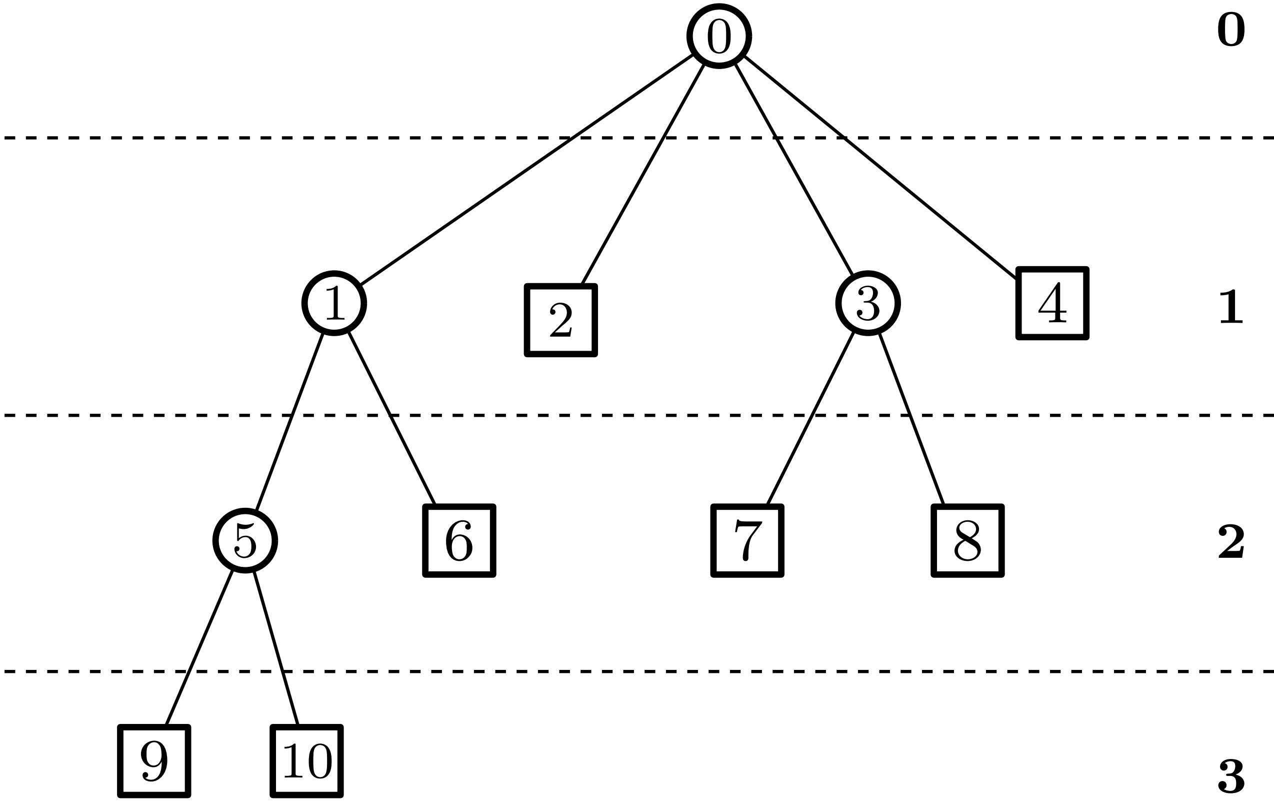


After preprocessing,

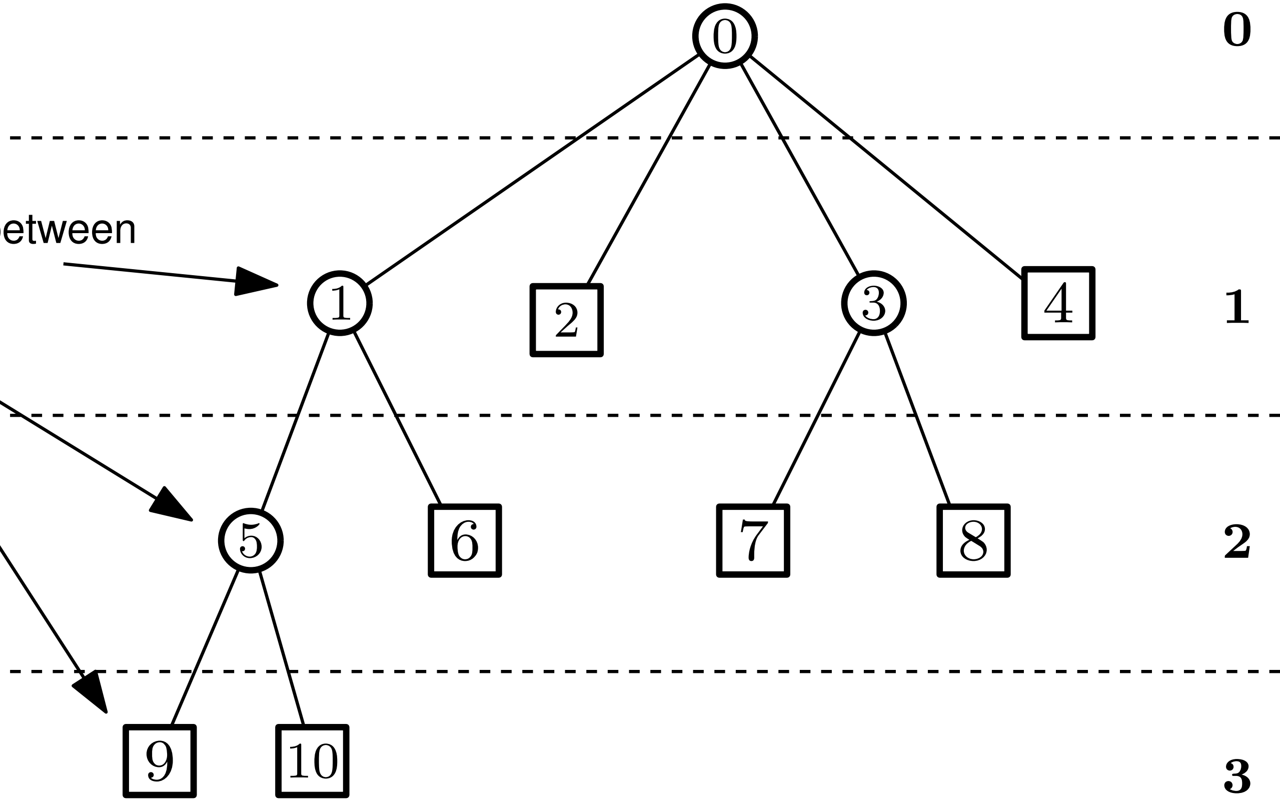
the output to a query LCA(i, j) is the lowest common ancestor of nodes i and j

- Ideally, we would like $O(n)$ space, $O(n)$ prep. time and $O(1)$ query time

Solving LCAs using RMQs



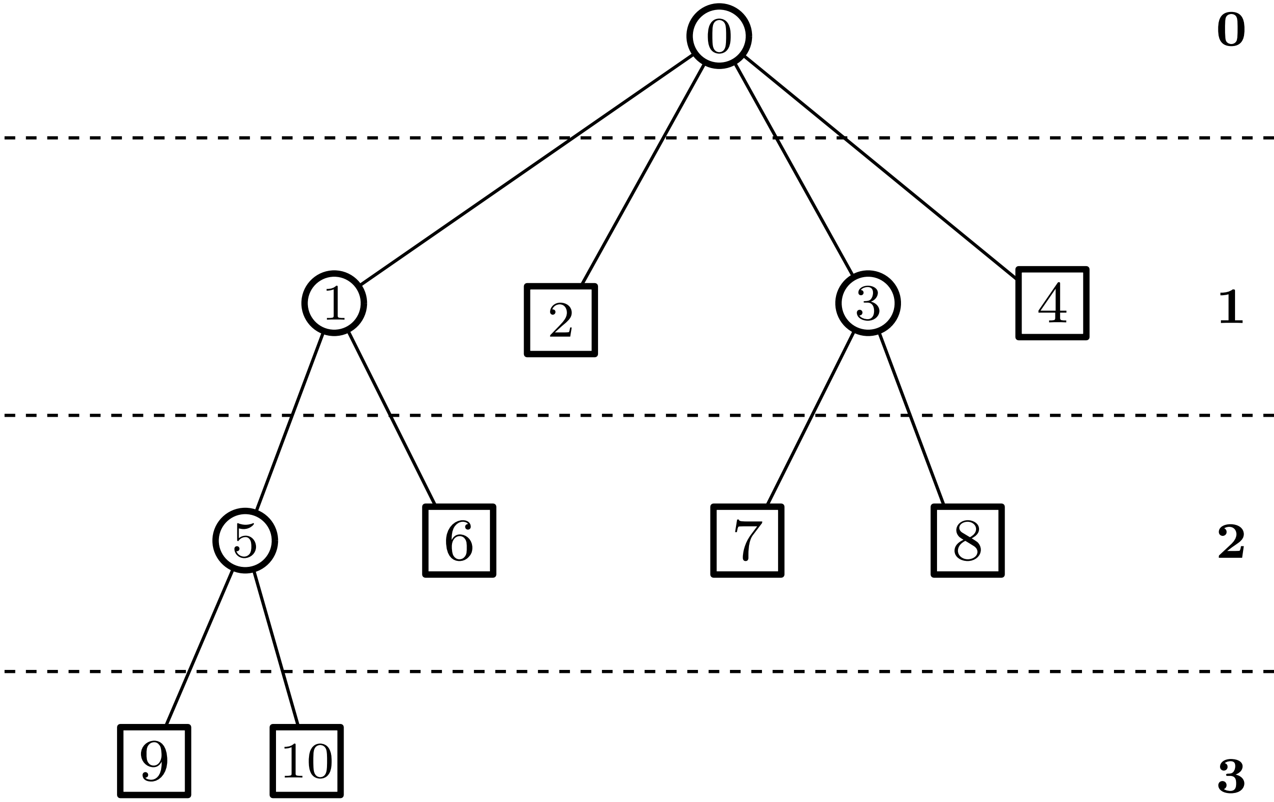
Solving LCAs using RMQs



the nodes are numbered between

0 and $(n-1)$

Solving LCAs using RMQs



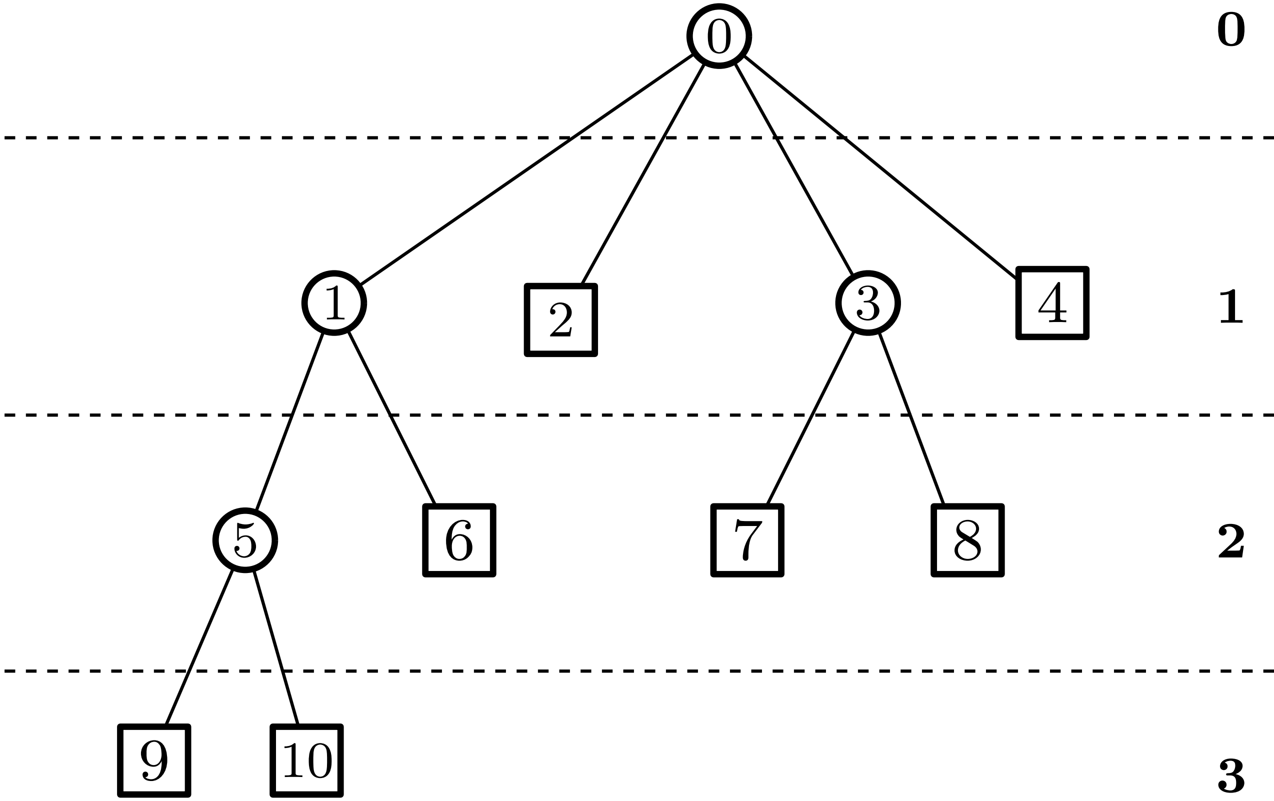
Solving LCAs using RMQs

Compute an Euler tour of T...

(a depth first search with repeats)

Write down every node you visit

...and its depth



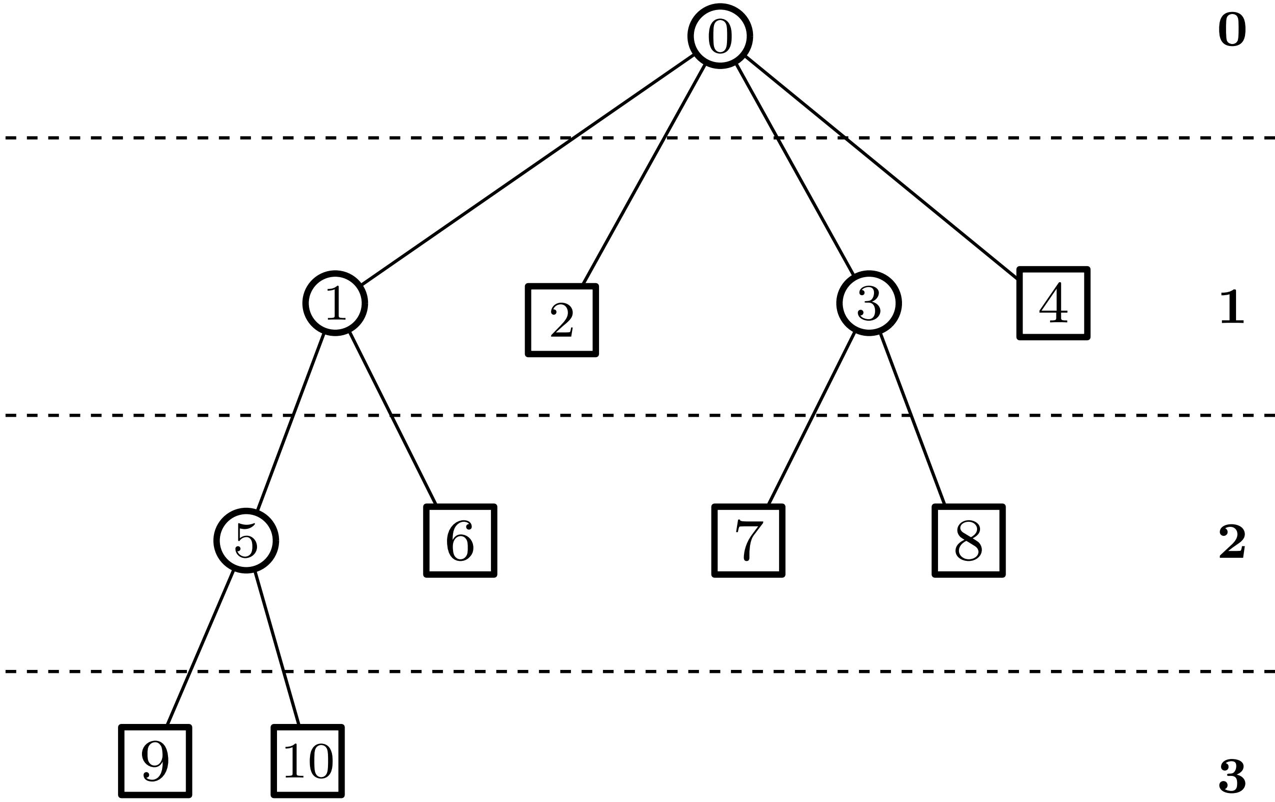
Solving LCAs using RMQs

Compute an Euler tour of T...

(a depth first search with repeats)

Write down every node you visit

...and its depth



(node) N $\boxed{0}$

(depth) D $\boxed{\mathbf{0}}$

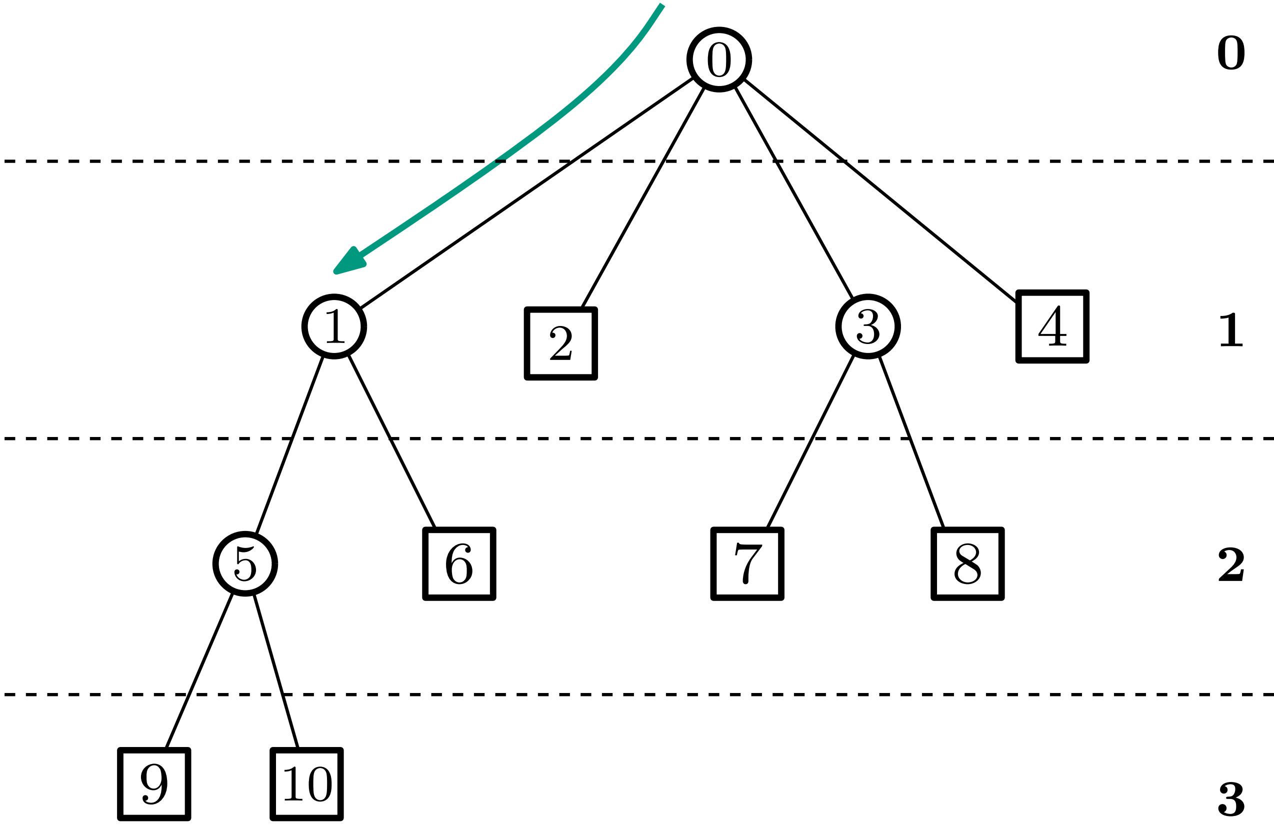
Solving LCAs using RMQs

Compute an Euler tour of T...

(a depth first search with repeats)

Write down every node you visit

...and its depth



(node) N	0	1
(depth) D	**0**	**1**

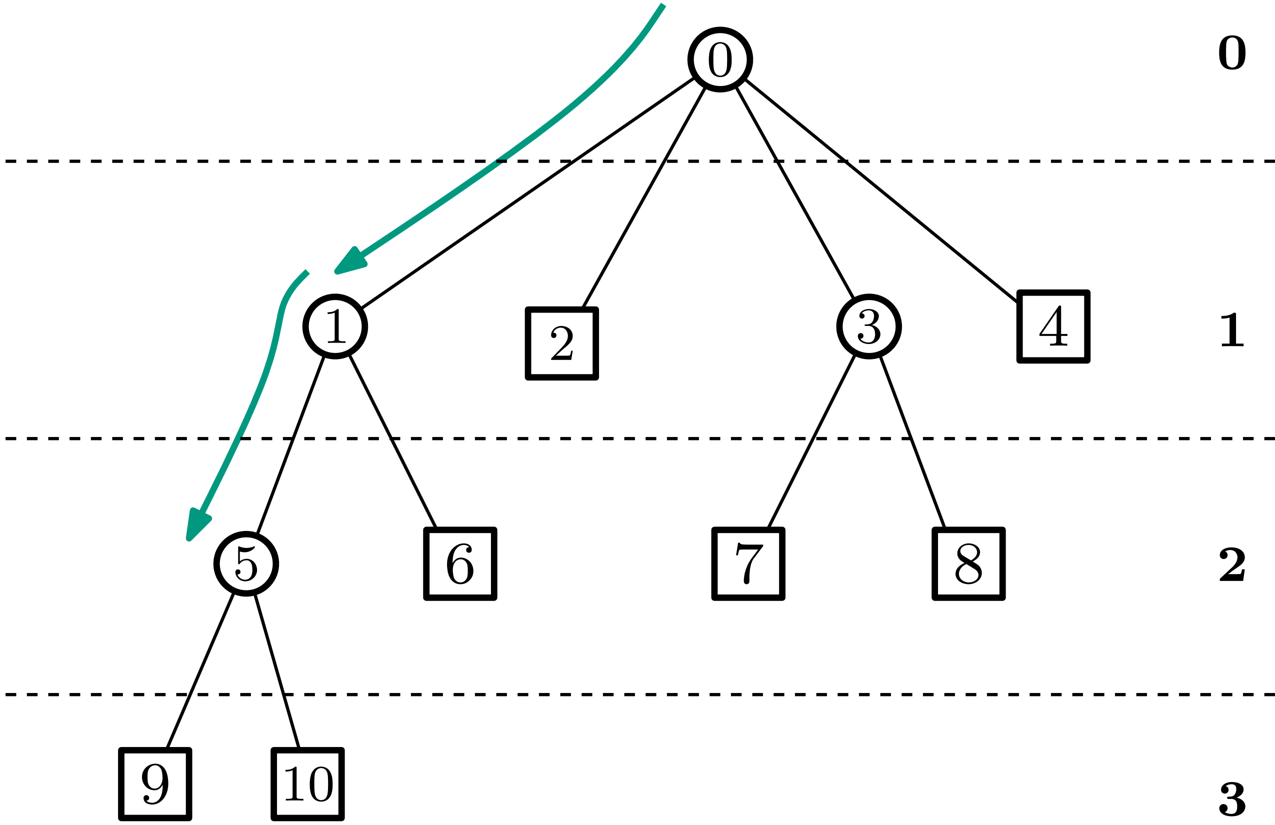
Solving LCAs using RMQs

Compute an Euler tour of T...

(a depth first search with repeats)

Write down every node you visit

...and its depth



(node) N	0	1	5
(depth) D	**0**	**1**	**2**

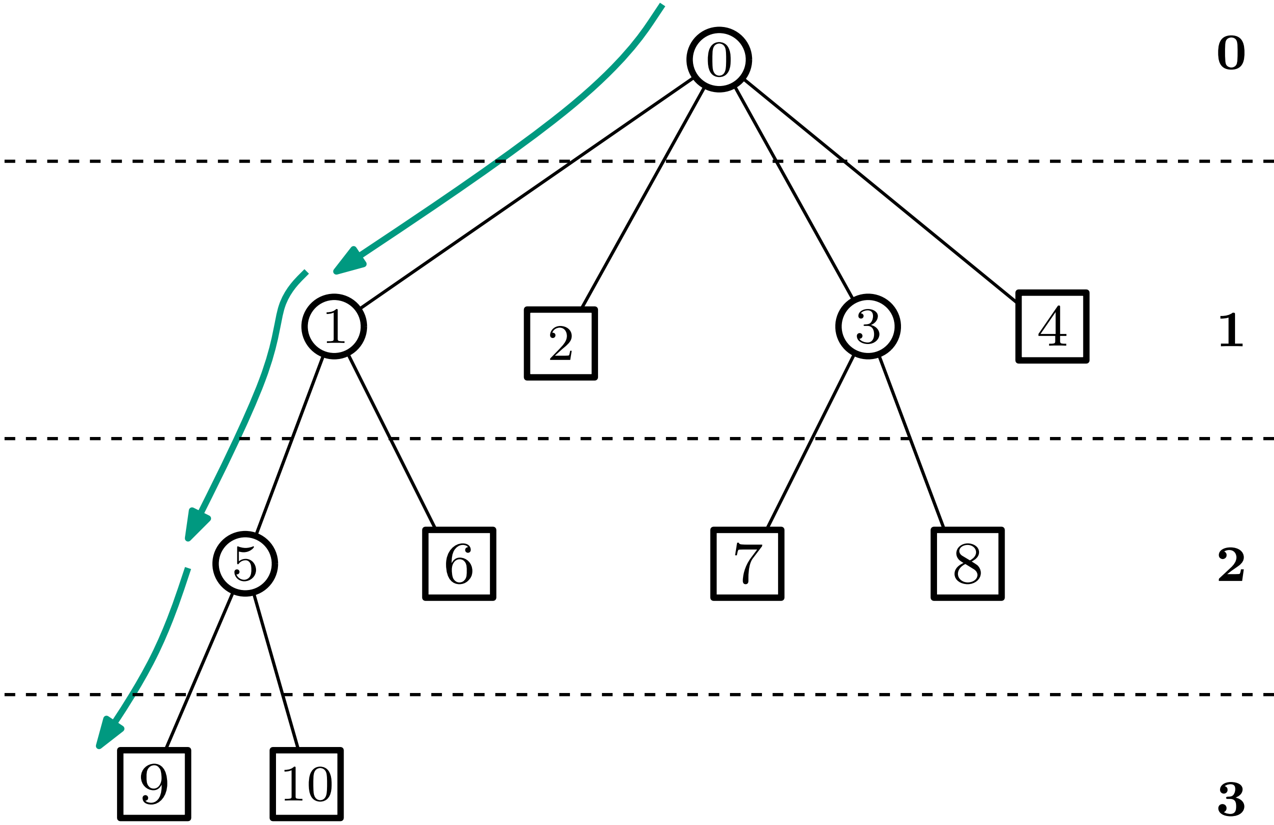
Solving LCAs using RMQs

Compute an Euler tour of T . . .

(a depth first search with repeats)

Write down every node you visit

. . . and its depth



(node) N

0	1	5	9

(depth) D

0	**1**	**2**	**3**

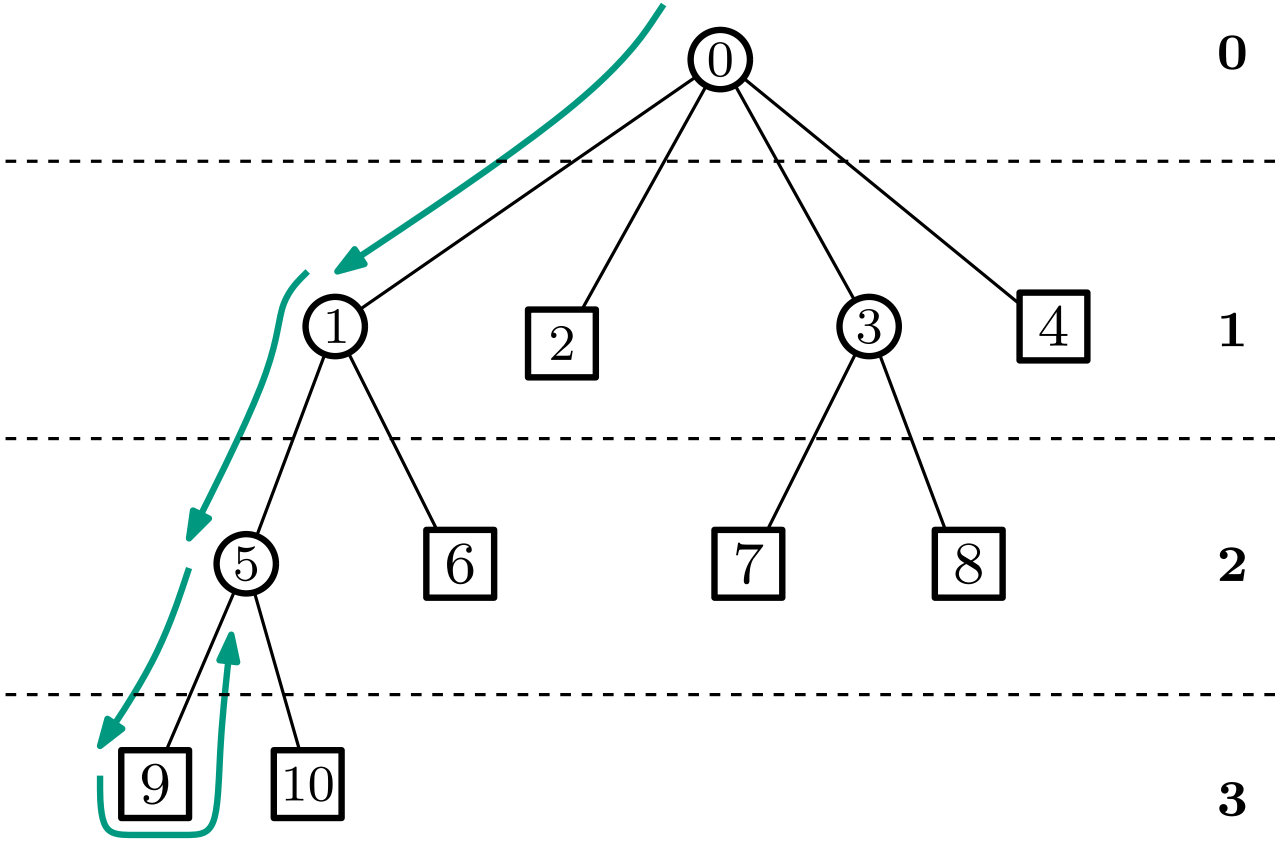
Solving LCAs using RMQs

Compute an Euler tour of T...

(a depth first search with repeats)

Write down every node you visit

...and its depth



(node) N	0	1	5	9	5
(depth) D	**0**	**1**	**2**	**3**	**2**

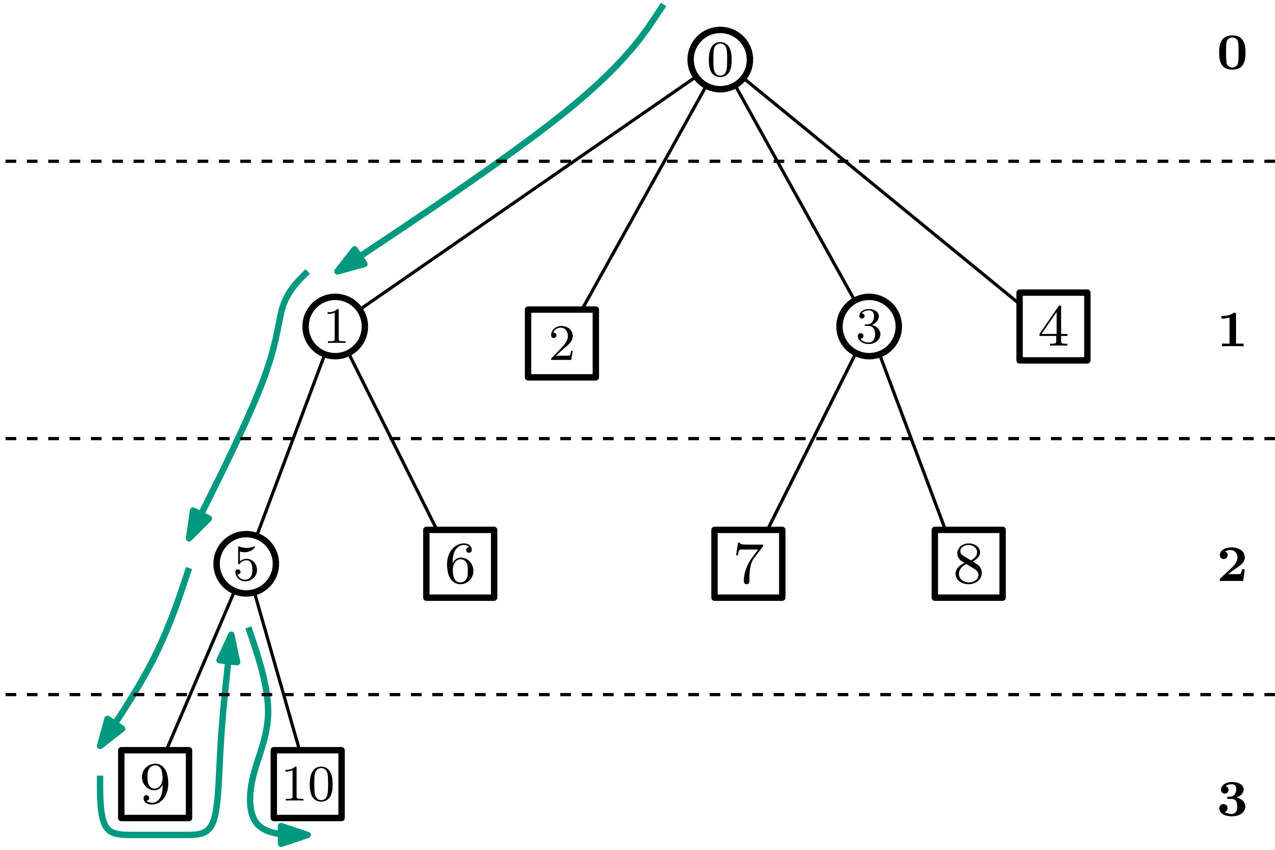
Solving LCAs using RMQs

Compute an Euler tour of T . . .

(a depth first search with repeats)

Write down every node you visit

. . . and its depth



(node) N	0	1	5	9	5	10
(depth) D	**0**	**1**	**2**	**3**	**2**	**3**

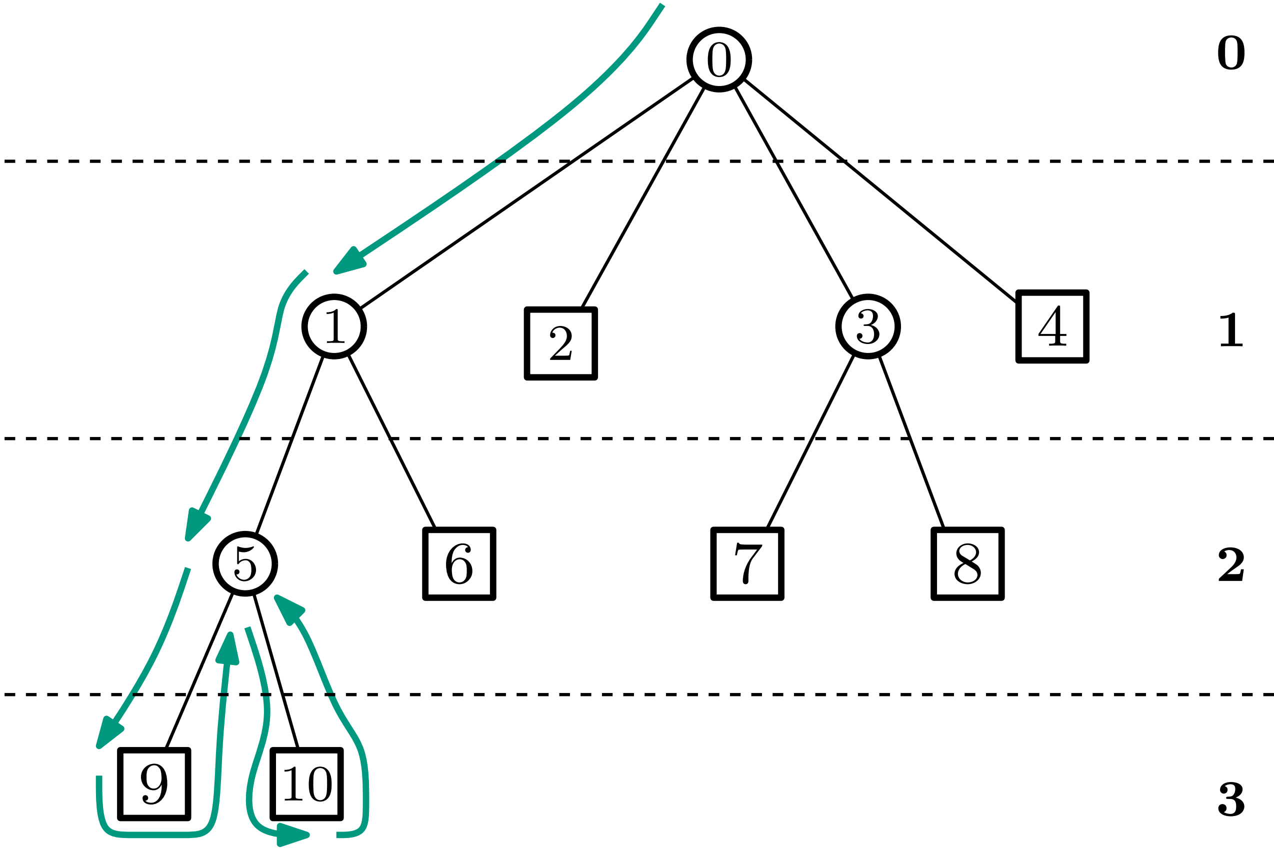
Solving LCAs using RMQs

Compute an Euler tour of T...

(a depth first search with repeats)

Write down every node you visit

...and its depth



(node) N	0	1	5	9	5	10	5
(depth) D	**0**	**1**	**2**	**3**	**2**	**3**	**2**

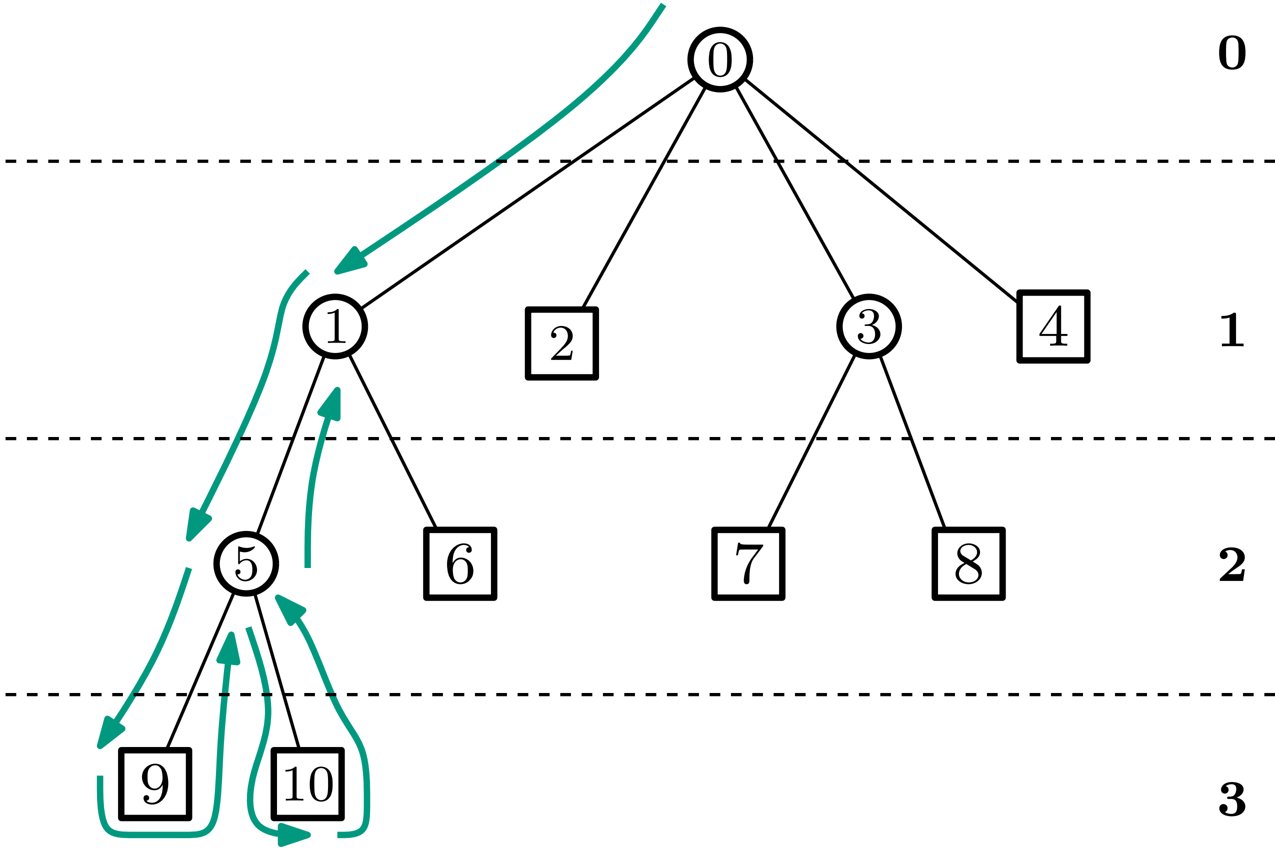
Solving LCAs using RMQs

Compute an Euler tour of T . . .

(a depth first search with repeats)

Write down every node you visit

. . . and its depth



(node) N	0	1	5	9	5	10	5	1
(depth) D	**0**	**1**	**2**	**3**	**2**	**3**	**2**	**1**

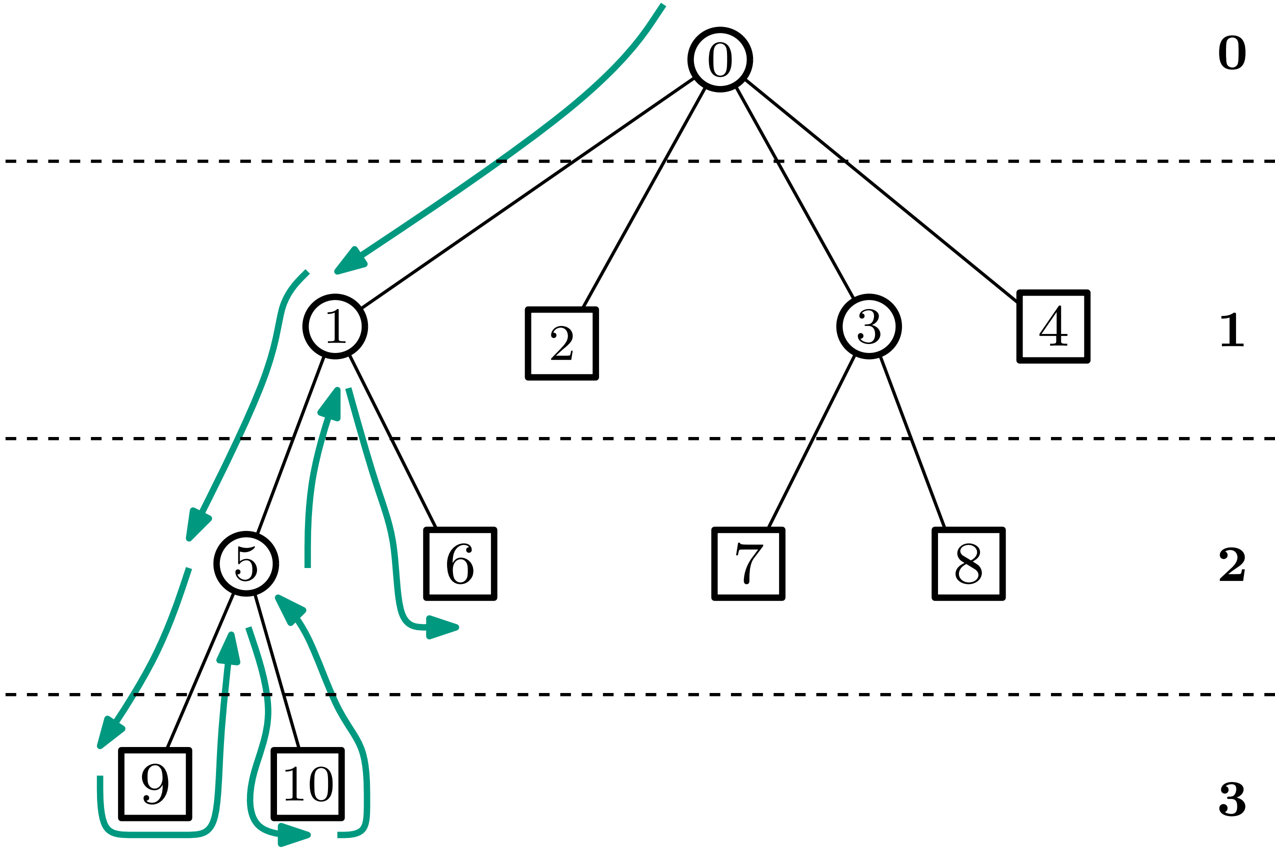
Solving LCAs using RMQs

Compute an Euler tour of T...

(a depth first search with repeats)

Write down every node you visit

...and its depth



(node) N	0	1	5	9	5	10	5	1	6
(depth) D	**0**	**1**	**2**	**3**	**2**	**3**	**2**	**1**	**2**

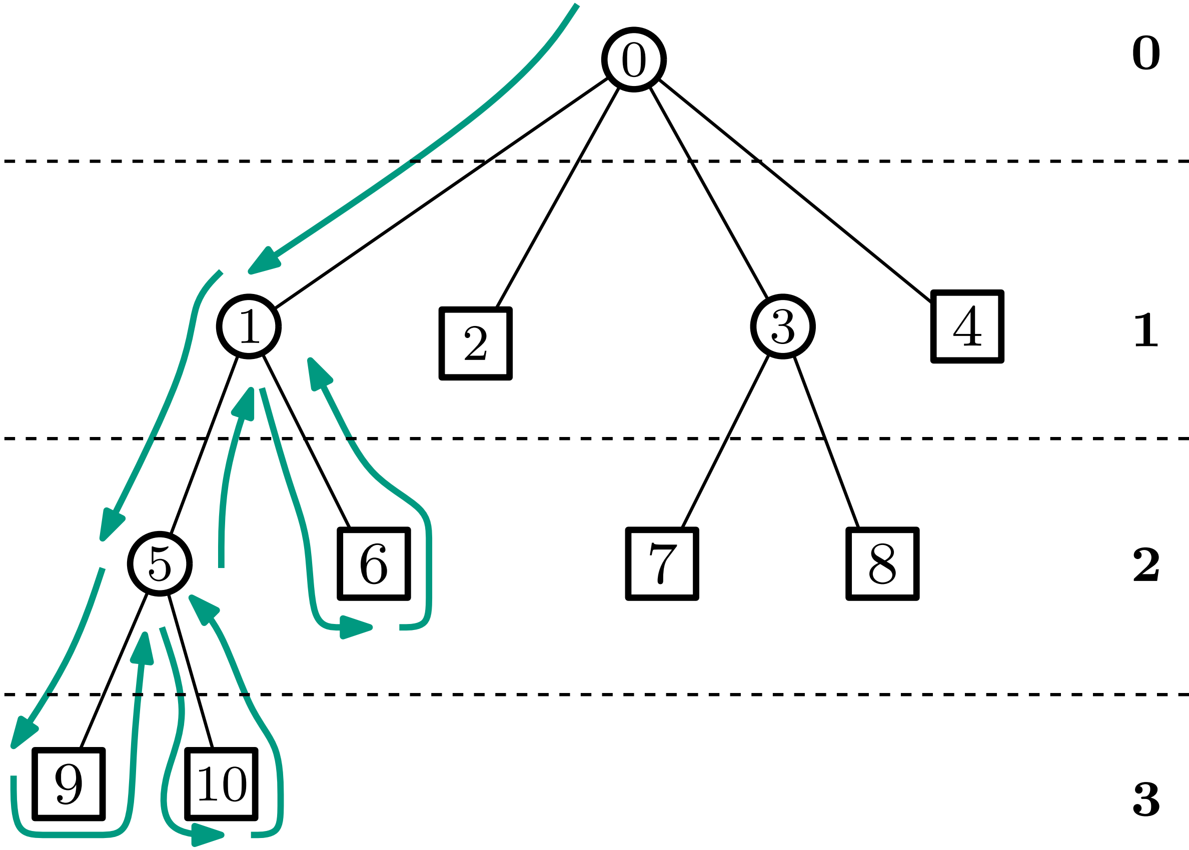
Solving LCAs using RMQs

Compute an Euler tour of T...

(a depth first search with repeats)

Write down every node you visit

...and its depth



(node) N	0	1	5	9	5	10	5	1	6	1
(depth) D	**0**	**1**	**2**	**3**	**2**	**3**	**2**	**1**	**2**	**1**

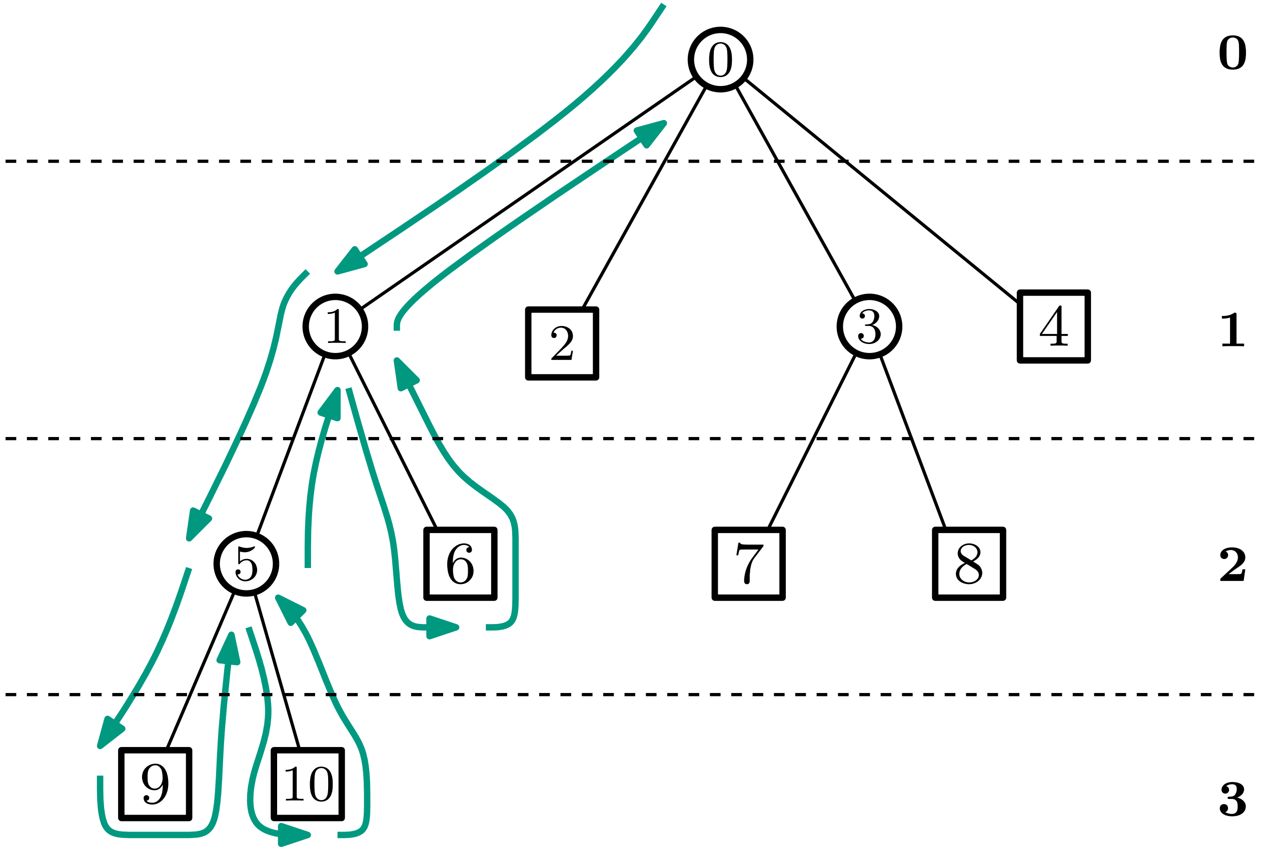
Solving LCAs using RMQs

Compute an Euler tour of T...

(a depth first search with repeats)

Write down every node you visit

...and its depth



(node) N	0	1	5	9	5	10	5	1	6	1	0
(depth) D	**0**	**1**	**2**	**3**	**2**	**3**	**2**	**1**	**2**	**1**	**0**

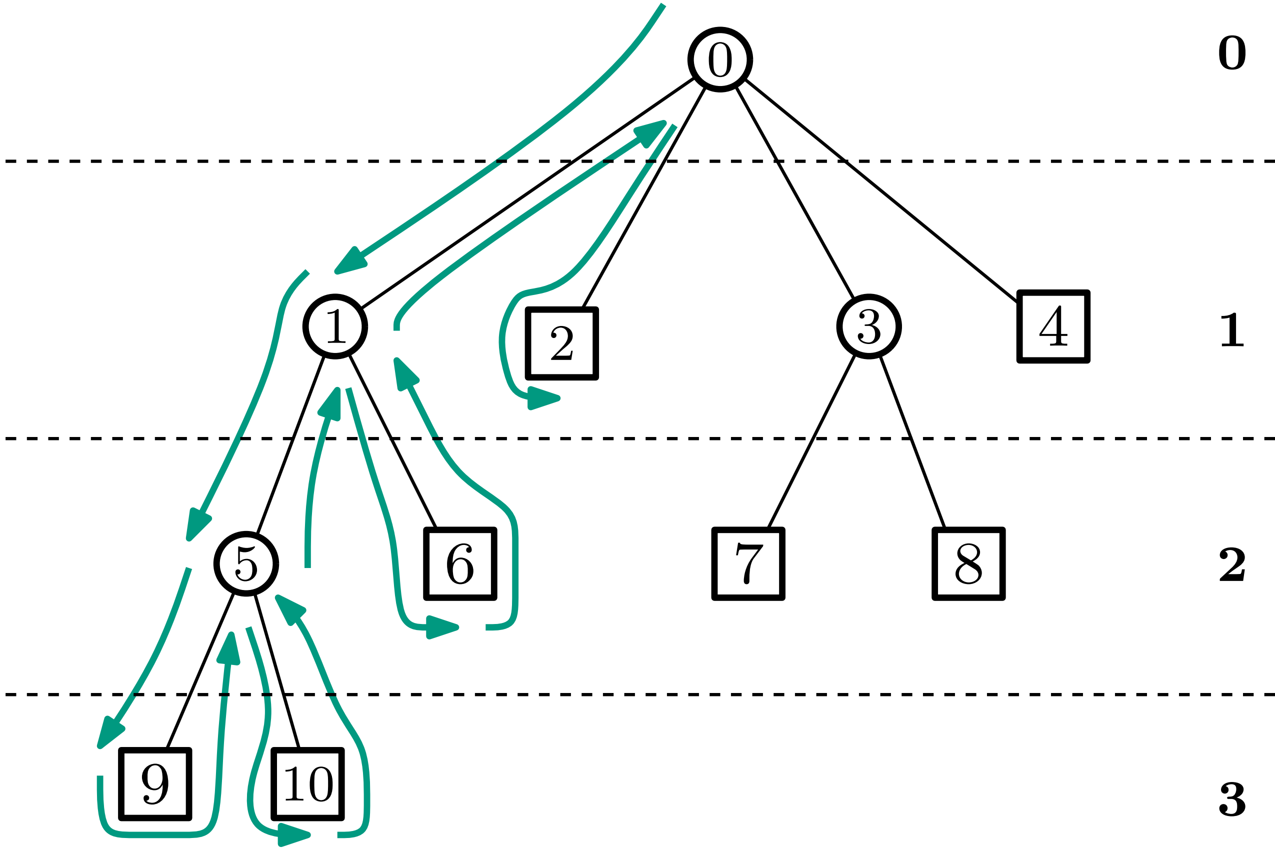
Solving LCAs using RMQs

Compute an Euler tour of T . . .

(a depth first search with repeats)

Write down every node you visit

. . . and its depth



(node) N	0	1	5	9	5	10	5	1	6	1	0	2
(depth) D	**0**	**1**	**2**	**3**	**2**	**3**	**2**	**1**	**2**	**1**	**0**	**1**

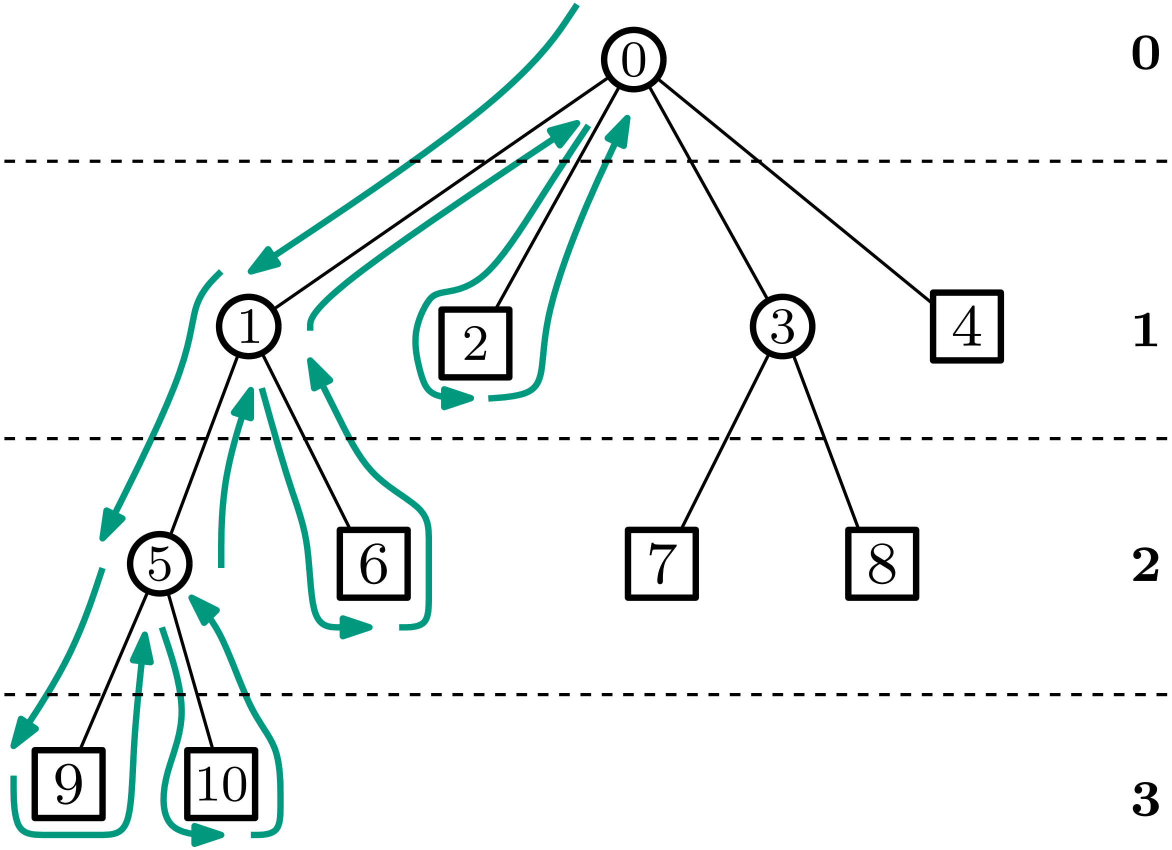
Solving LCAs using RMQs

Compute an Euler tour of T . . .

(a depth first search with repeats)

Write down every node you visit

. . . and its depth



(node) N	0	1	5	9	5	10	5	1	6	1	0	2	0
(depth) D	**0**	**1**	**2**	**3**	**2**	**3**	**2**	**1**	**2**	**1**	**0**	**1**	**0**

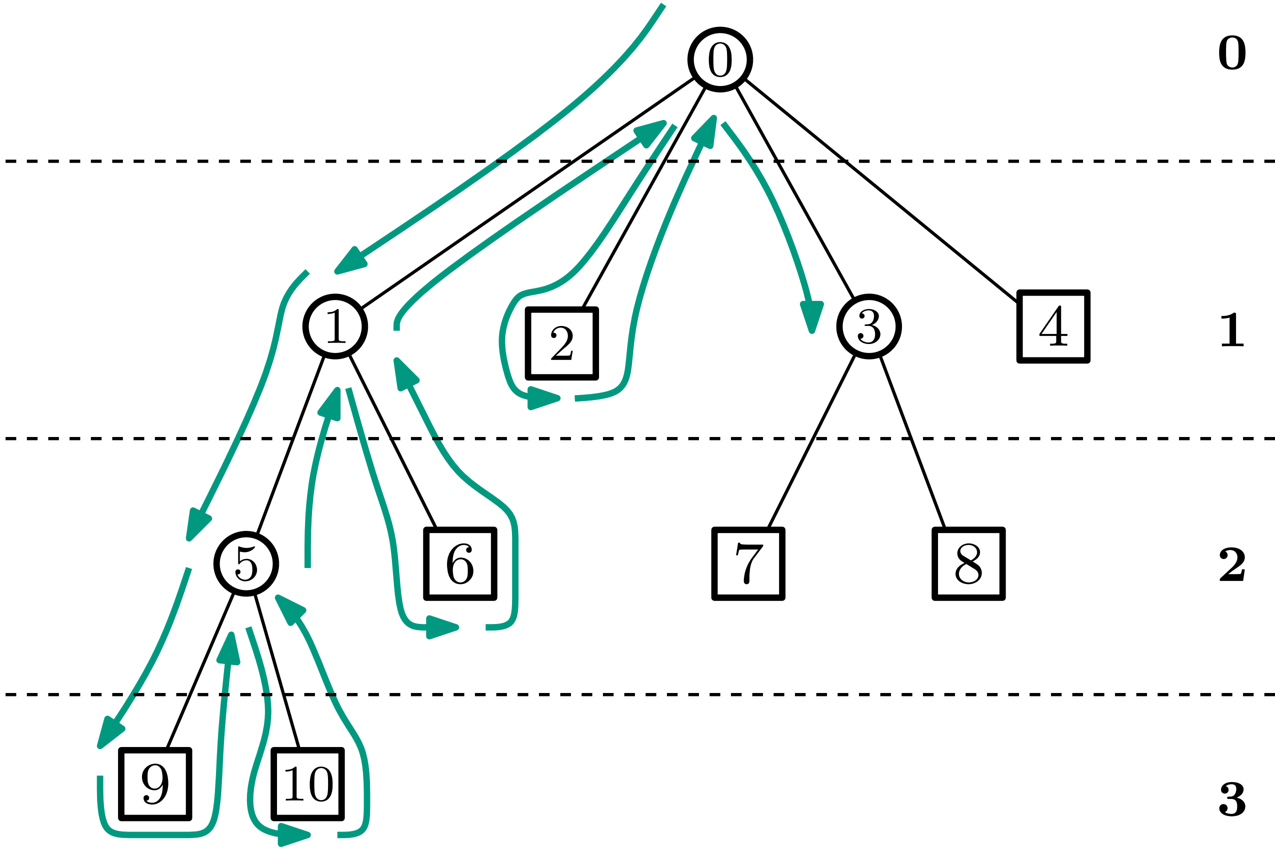
Solving LCAs using RMQs

Compute an Euler tour of T...

(a depth first search with repeats)

Write down every node you visit

...and its depth



(node) N	0	1	5	9	5	10	5	1	6	1	0	2	0	3
(depth) D	**0**	**1**	**2**	**3**	**2**	**3**	**2**	**1**	**2**	**1**	**0**	**1**	**0**	**1**

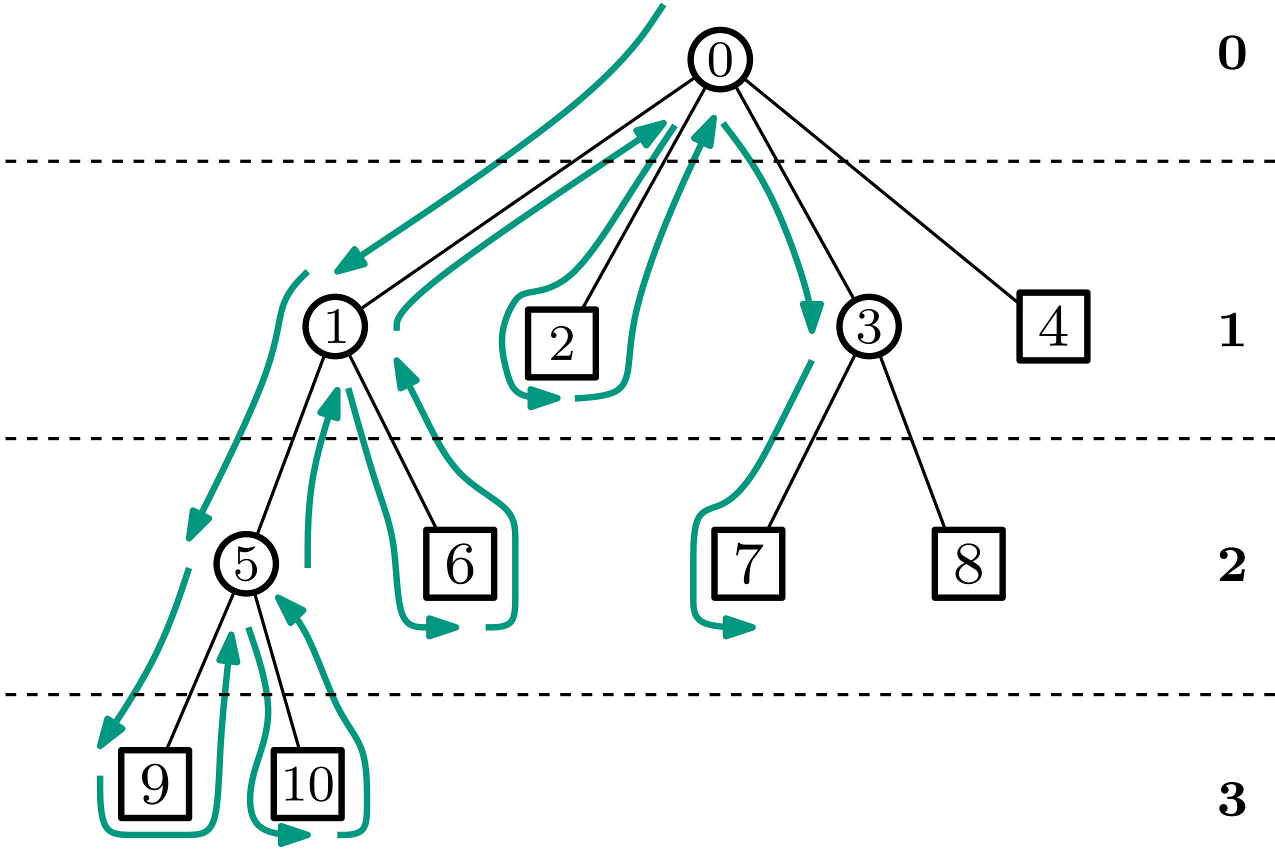
Solving LCAs using RMQs

Compute an Euler tour of T...

(a depth first search with repeats)

Write down every node you visit

...and its depth



(node) N	0	1	5	9	5	10	5	1	6	1	0	2	0	3	7
(depth) D	**0**	**1**	**2**	**3**	**2**	**3**	**2**	**1**	**2**	**1**	**0**	**1**	**0**	**1**	**2**

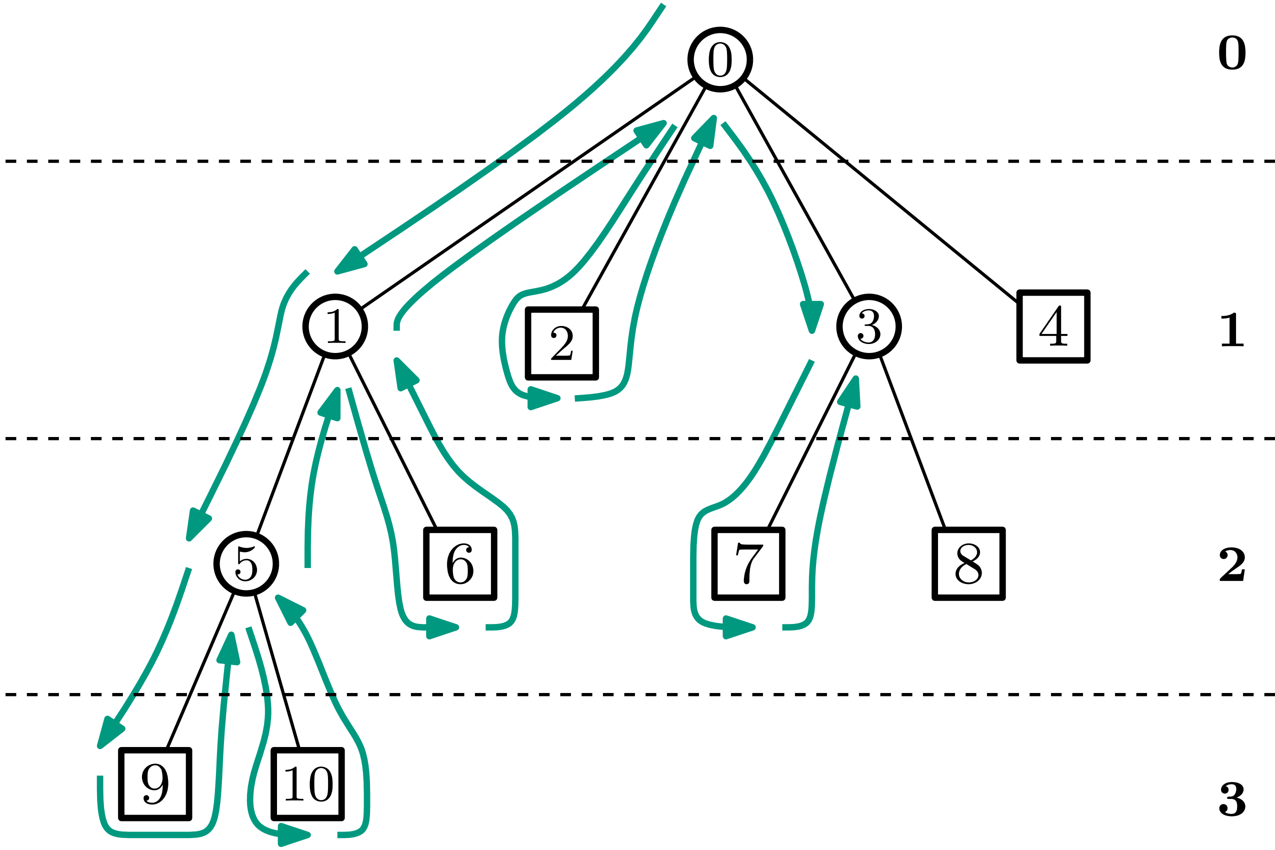
Solving LCAs using RMQs

Compute an Euler tour of T...

(a depth first search with repeats)

Write down every node you visit

...and its depth



(node) N	0	1	5	9	5	10	5	1	6	1	0	2	0	3	7	3
(depth) D	**0**	**1**	**2**	**3**	**2**	**3**	**2**	**1**	**2**	**1**	**0**	**1**	**0**	**1**	**2**	**1**

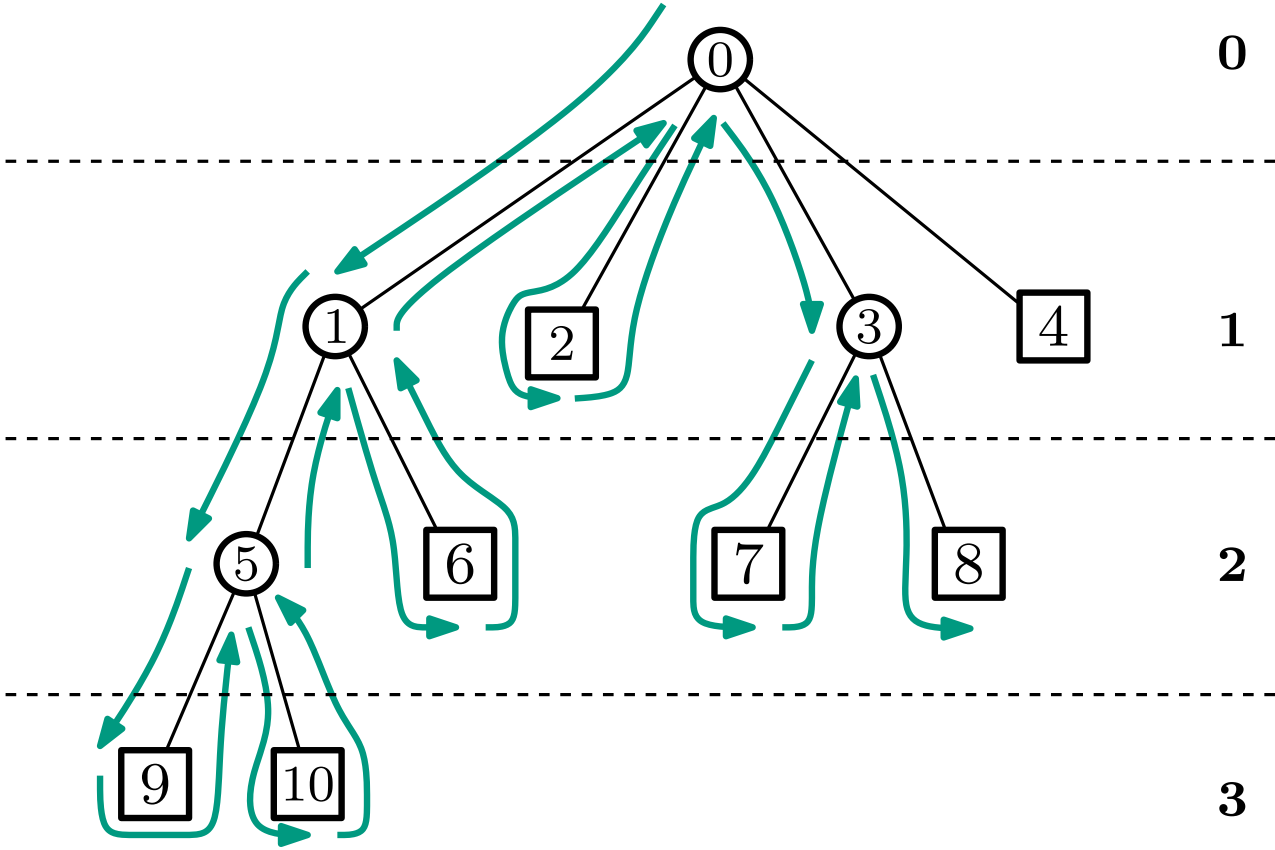
Solving LCAs using RMQs

Compute an Euler tour of T...

(a depth first search with repeats)

Write down every node you visit

...and its depth



(node) N	0	1	5	9	5	10	5	1	6	1	0	2	0	3	7	3	8
(depth) D	**0**	**1**	**2**	**3**	**2**	**3**	**2**	**1**	**2**	**1**	**0**	**1**	**0**	**1**	**2**	**1**	**2**

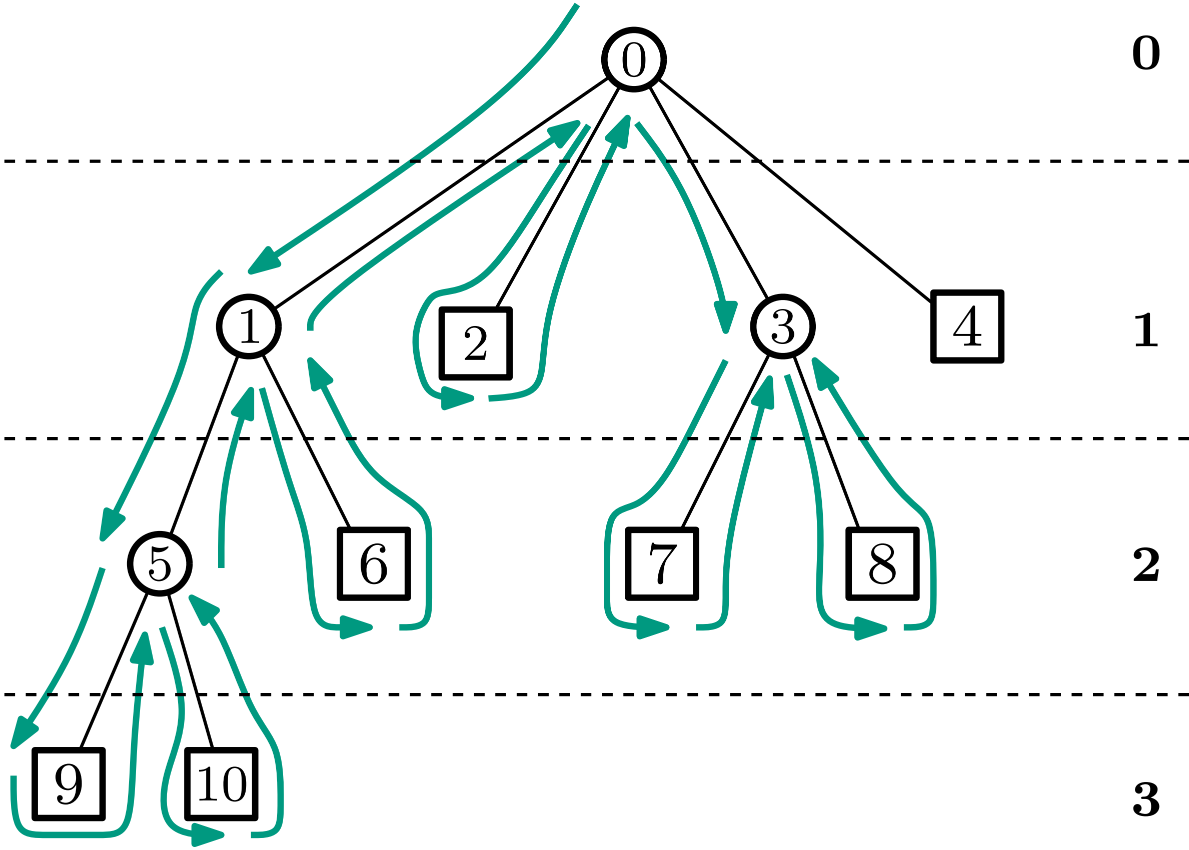
Solving LCAs using RMQs

Compute an Euler tour of T...

(a depth first search with repeats)

Write down every node you visit

...and its depth



(node) N	0	1	5	9	5	10	5	1	6	1	0	2	0	3	7	3	8	3
(depth) D	**0**	**1**	**2**	**3**	**2**	**3**	**2**	**1**	**2**	**1**	**0**	**1**	**0**	**1**	**2**	**1**	**2**	**1**

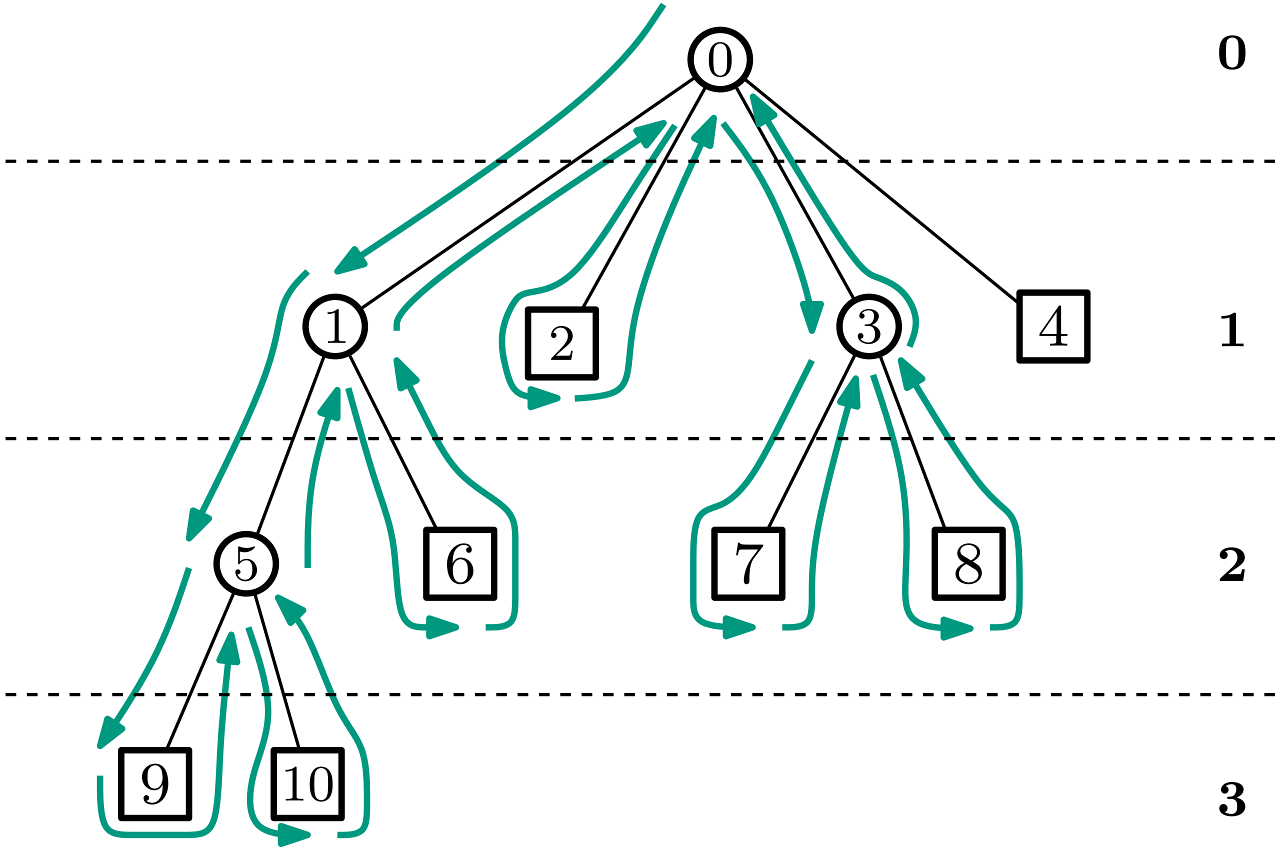
Solving LCAs using RMQs

Compute an Euler tour of T...

(a depth first search with repeats)

Write down every node you visit

...and its depth



(node) N	0	1	5	9	5	10	5	1	6	1	0	2	0	3	7	3	8	3	0
(depth) D	**0**	**1**	**2**	**3**	**2**	**3**	**2**	**1**	**2**	**1**	**0**	**1**	**0**	**1**	**2**	**1**	**2**	**1**	**0**

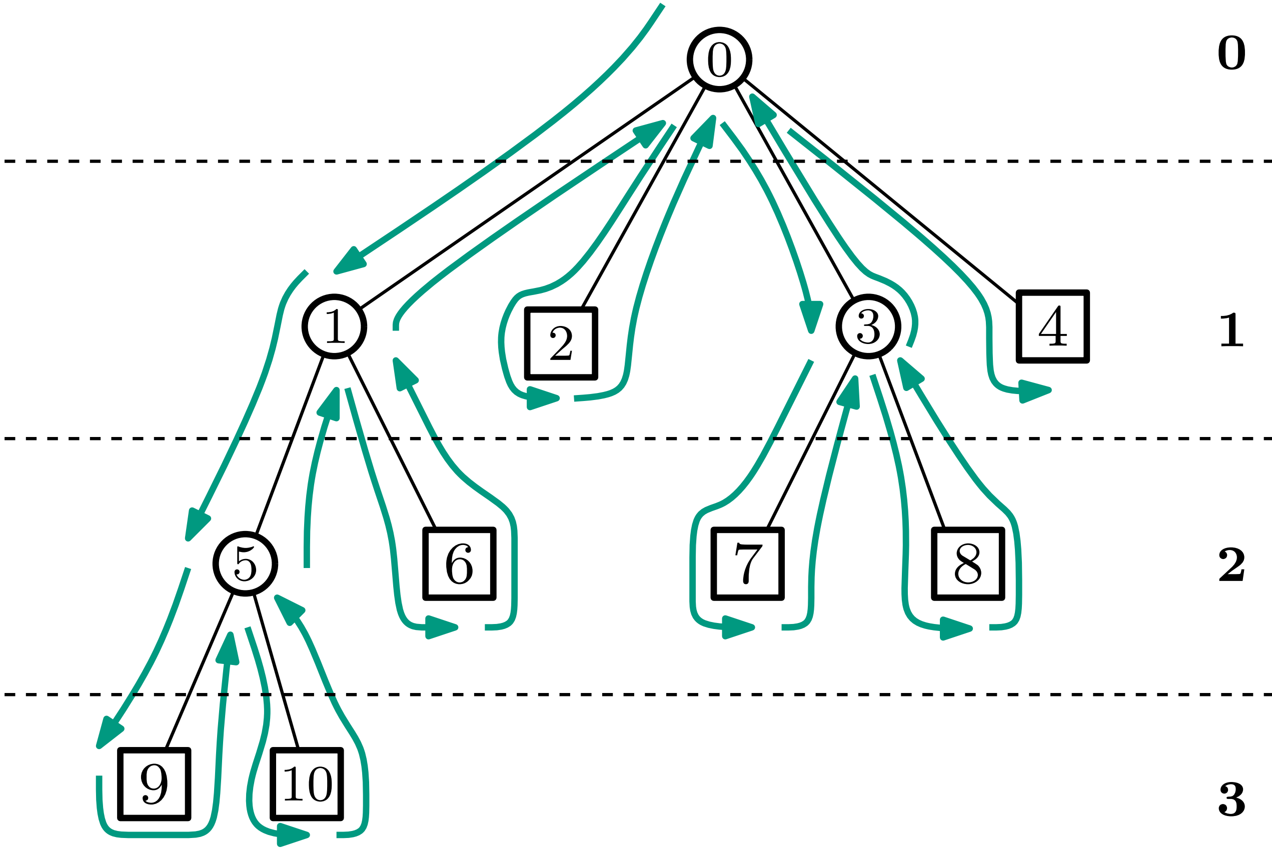
Solving LCAs using RMQs

Compute an Euler tour of T...

(a depth first search with repeats)

Write down every node you visit

...and its depth



(node) N	0	1	5	9	5	10	5	1	6	1	0	2	0	3	7	3	8	3	0	4
(depth) D	**0**	**1**	**2**	**3**	**2**	**3**	**2**	**1**	**2**	**1**	**0**	**1**	**0**	**1**	**2**	**1**	**2**	**1**	**0**	**1**

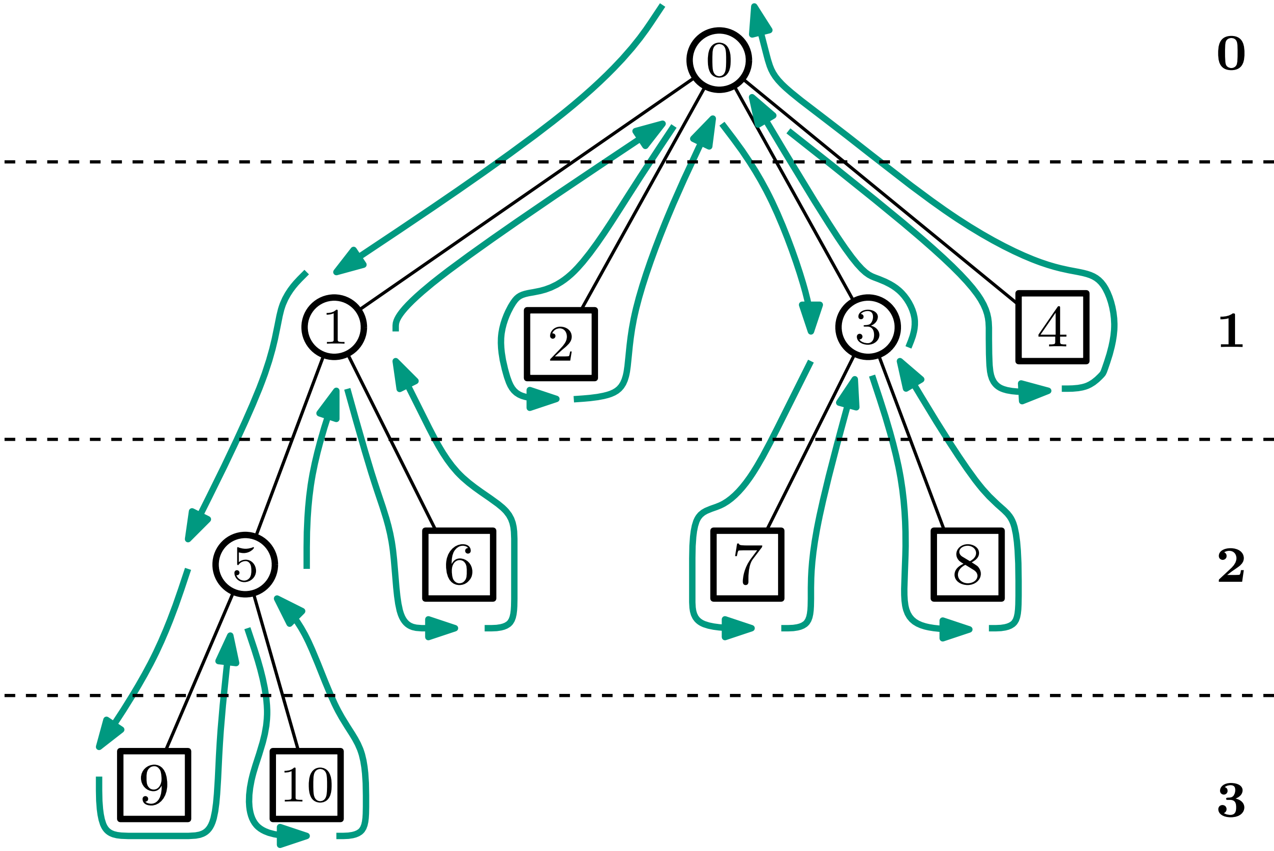
Solving LCAs using RMQs

Compute an Euler tour of T...

(a depth first search with repeats)

Write down every node you visit

...and its depth



(node) N	0	1	5	9	5	10	5	1	6	1	0	2	0	3	7	3	8	3	0	4	0
(depth) D	**0**	**1**	**2**	**3**	**2**	**3**	**2**	**1**	**2**	**1**	**0**	**1**	**0**	**1**	**2**	**1**	**2**	**1**	**0**	**1**	**0**

Solving LCAs using RMQs

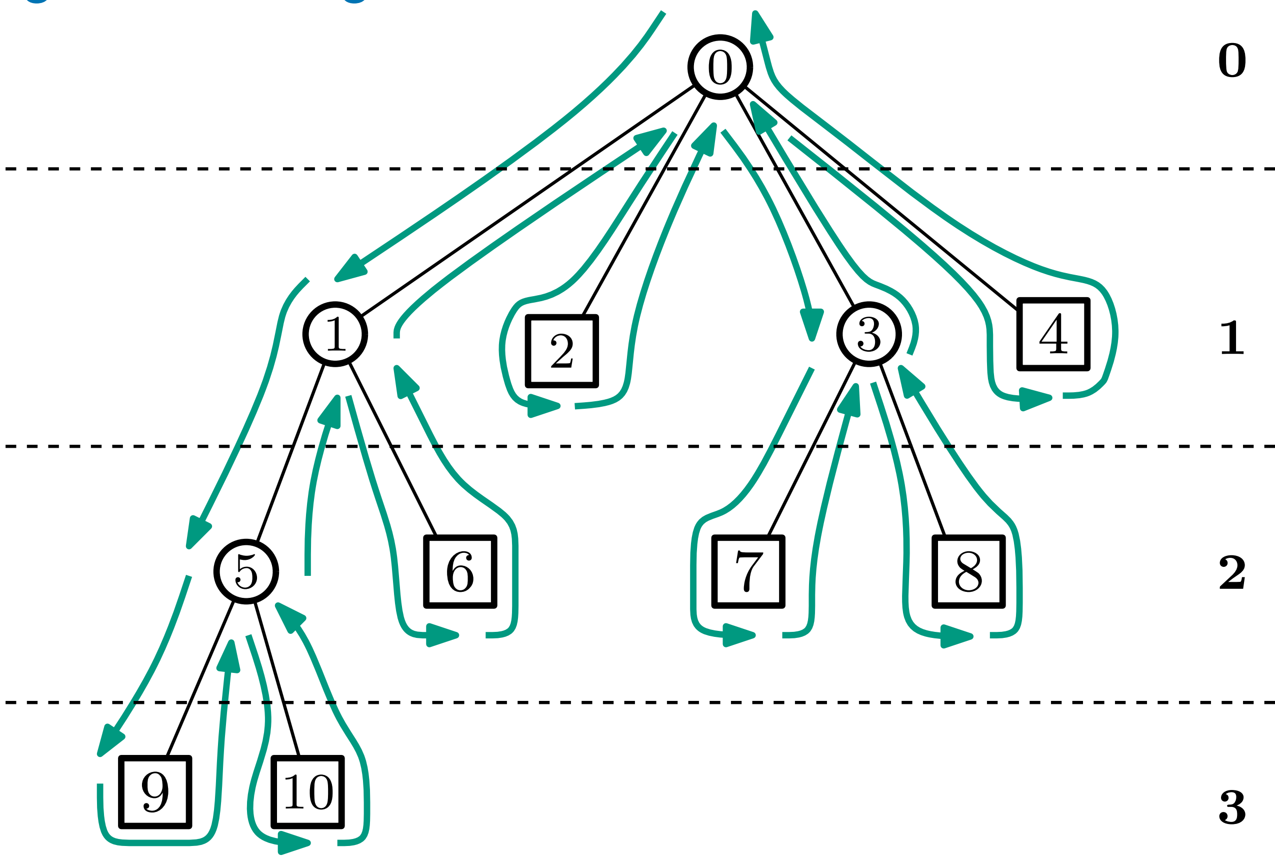
Compute an Euler tour of T...

(a depth first search with repeats)

Write down every node you visit

...and its depth

How long is the tour?



(node) N	0	1	5	9	5	10	5	1	6	1	0	2	0	3	7	3	8	3	0	4	0
(depth) D	**0**	**1**	**2**	**3**	**2**	**3**	**2**	**1**	**2**	**1**	**0**	**1**	**0**	**1**	**2**	**1**	**2**	**1**	**0**	**1**	**0**

Solving LCAs using RMQs

Compute an Euler tour of T...

(a depth first search with repeats)

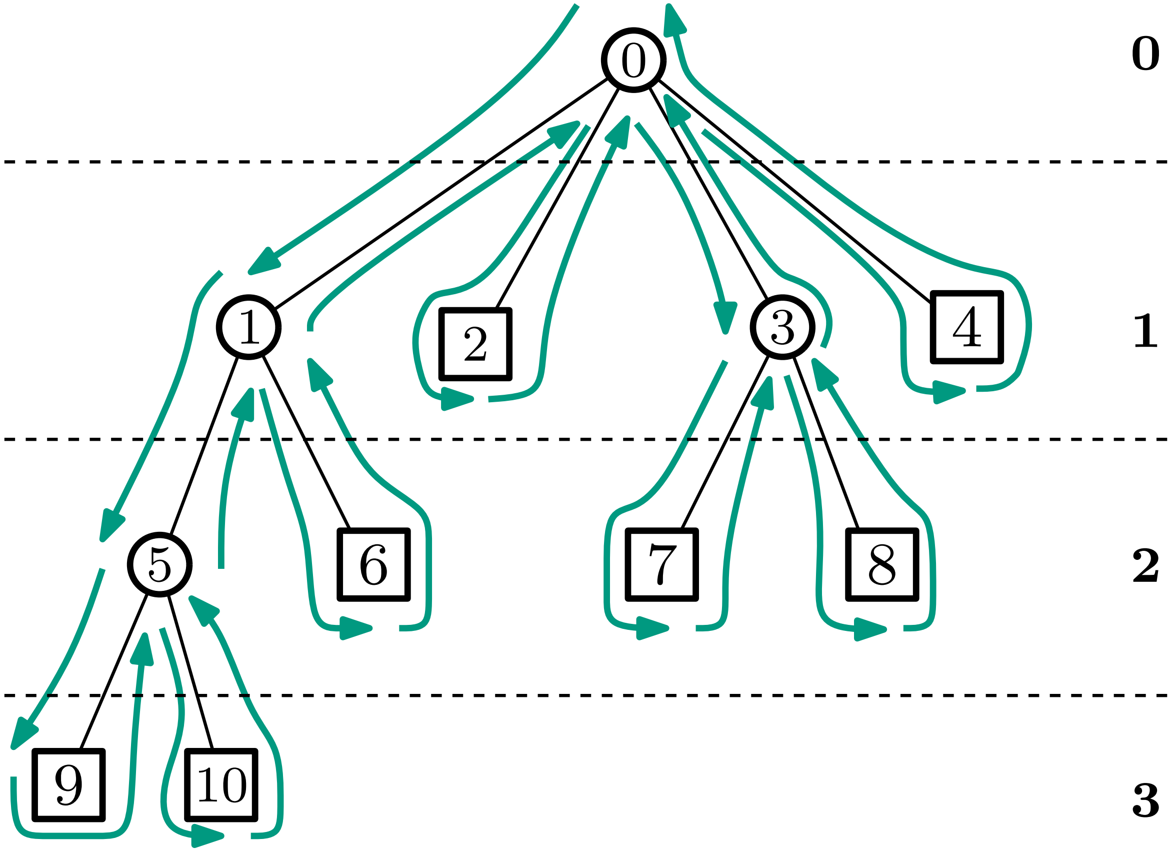
Write down every node you visit

...and its depth

How long is the tour?

We follow each edge twice...

and there are $(n-1)$ edges



(node) N	0	1	5	9	5	10	5	1	6	1	0	2	0	3	7	3	8	3	0	4	0
(depth) D	**0**	**1**	**2**	**3**	**2**	**3**	**2**	**1**	**2**	**1**	**0**	**1**	**0**	**1**	**2**	**1**	**2**	**1**	**0**	**1**	**0**

Solving LCAs using RMQs

Compute an Euler tour of T...

(a depth first search with repeats)

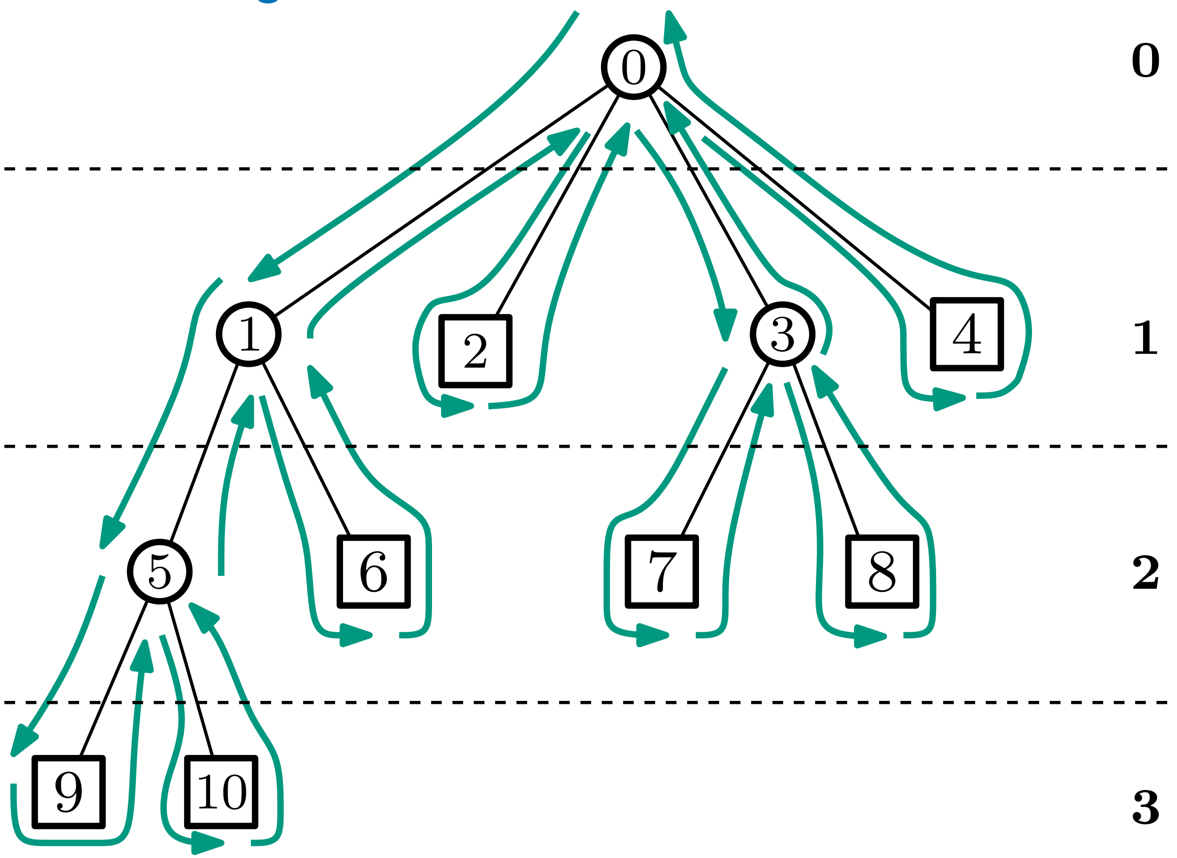
Write down every node you visit

...and its depth

How long is the tour?

We follow each edge twice...

and there are $(n-1)$ edges



(node) N	0	1	5	9	5	10	5	1	6	1	0	2	0	3	7	3	8	3	0	4	0

$2n-1$

(depth) D	**0**	**1**	**2**	**3**	**2**	**3**	**2**	**1**	**2**	**1**	**0**	**1**	**0**	**1**	**2**	**1**	**2**	**1**	**0**	**1**	**0**

Solving LCAs using RMQs

Compute an Euler tour of T...

(a depth first search with repeats)

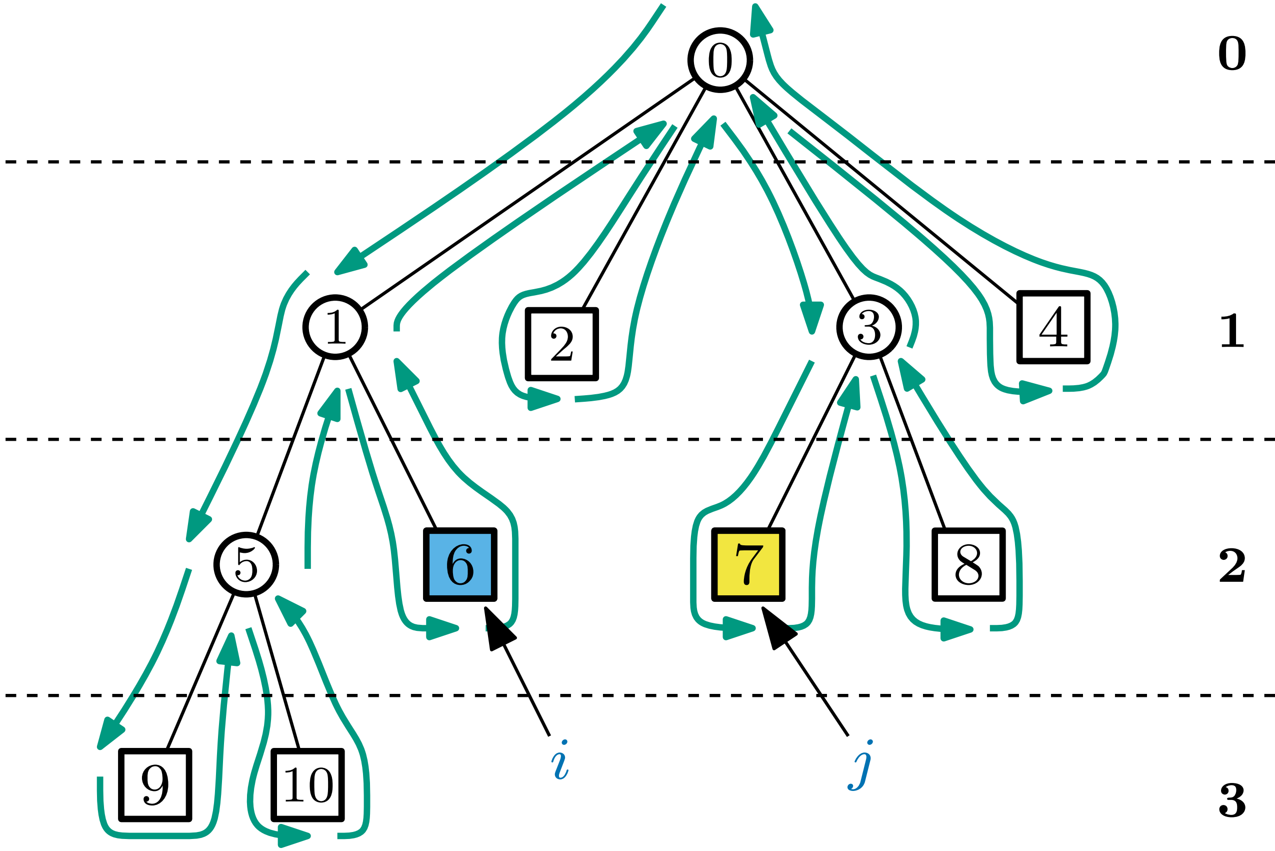
Write down every node you visit

...and its depth

How long is the tour?

We follow each edge twice...

and there are $(n-1)$ edges



(node) N	0	1	5	9	5	10	5	1	6	1	0	2	0	3	7	3	8	3	0	4	0

$2n-1$

(depth) D	**0**	**1**	**2**	**3**	**2**	**3**	**2**	**1**	**2**	**1**	**0**	**1**	**0**	**1**	**2**	**1**	**2**	**1**	**0**	**1**	**0**

how do we find LCA(i,j)?

Solving LCAs using RMQs

Compute an Euler tour of T...

(a depth first search with repeats)

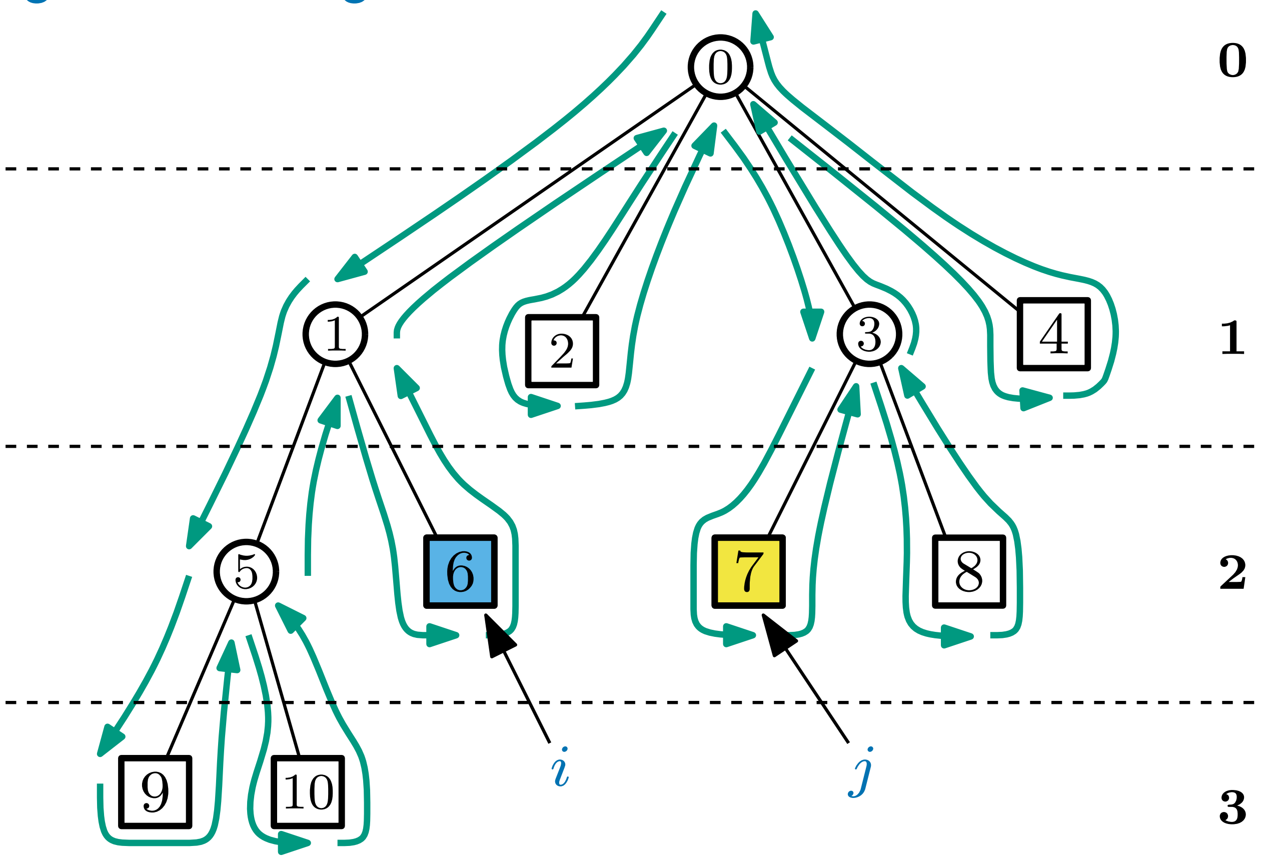
Write down every node you visit

...and its depth

How long is the tour?

We follow each edge twice...

and there are $(n-1)$ edges



Find i and j in N

(node) N	0	1	5	9	5	10	5	1	6	1	0	2	0	3	7	3	8	3	0	4	0
(depth) D	**0**	**1**	**2**	**3**	**2**	**3**	**2**	**1**	**2**	**1**	**0**	**1**	**0**	**1**	**2**	**1**	**2**	**1**	**0**	**1**	**0**

$2n-1$

how do we find LCA(i,j)?

Solving LCAs using RMQs

Compute an Euler tour of T...

(a depth first search with repeats)

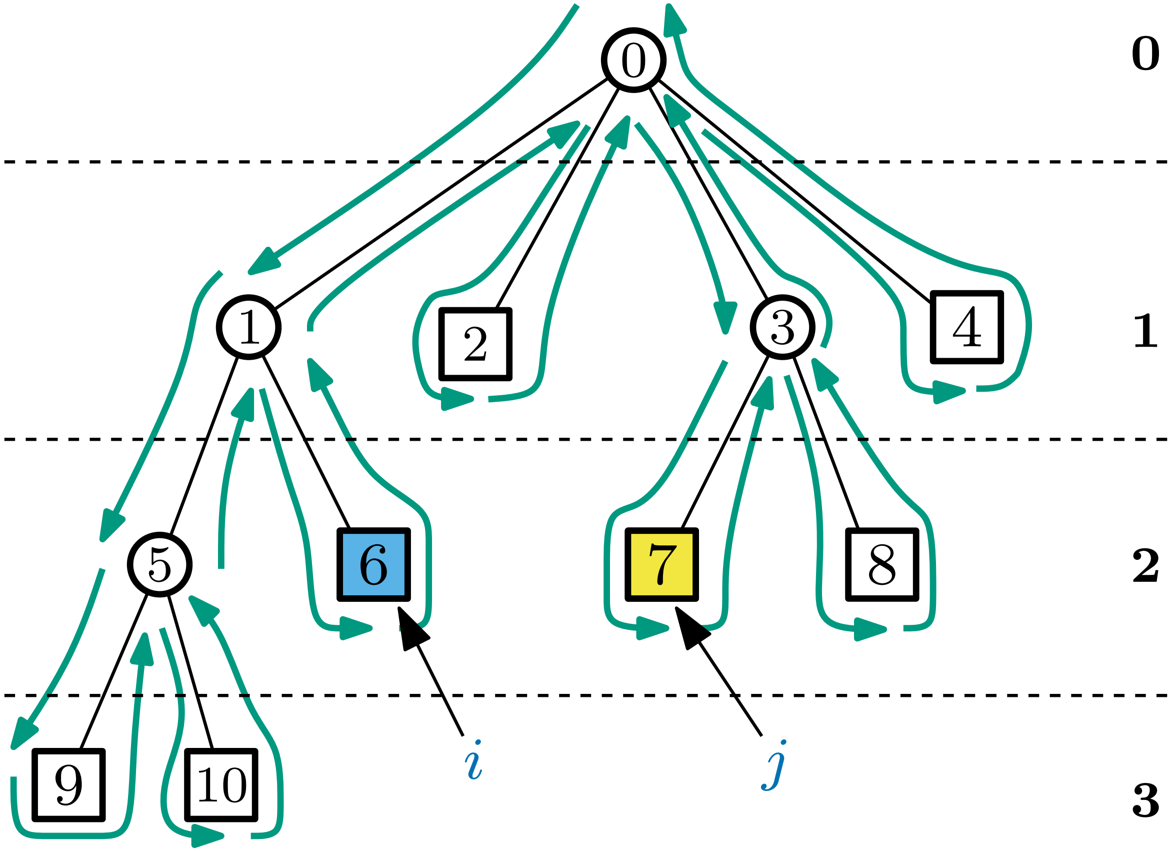
Write down every node you visit

...and its depth

How long is the tour?

We follow each edge twice...

and there are $(n-1)$ edges



Find i and j in N

(node) N	0	1	5	9	5	10	5	1	6	1	0	2	0	3	7	3	8	3	0	4	0
(depth) D	**0**	**1**	**2**	**3**	**2**	**3**	**2**	**1**	**2**	**1**	**0**	**1**	**0**	**1**	**2**	**1**	**2**	**1**	**0**	**1**	**0**

$2n-1$

i' j'

how do we find LCA(i,j)?

Solving LCAs using RMQs

Compute an Euler tour of T...

(a depth first search with repeats)

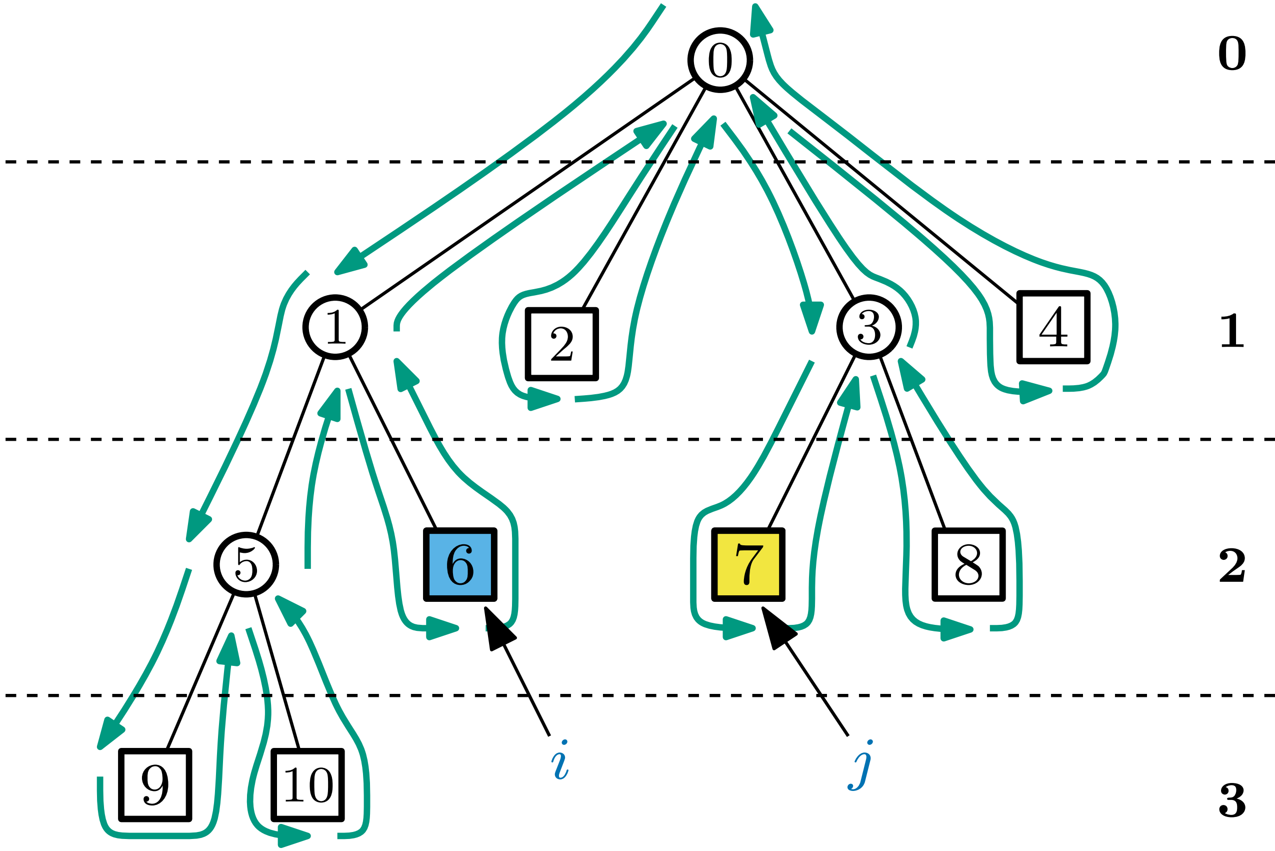
Write down every node you visit

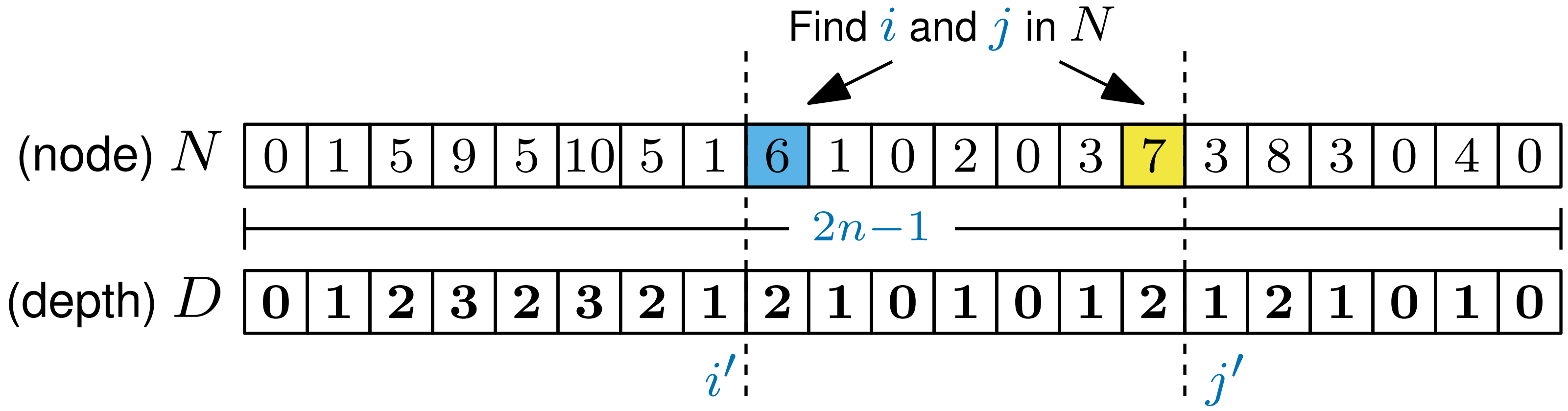
... and its depth

How long is the tour?

We follow each edge twice...

and there are $(n-1)$ edges





(node) N	0	1	5	9	5	10	5	1	6	1	0	2	0	3	7	3	8	3	0	4	0
(depth) D	**0**	**1**	**2**	**3**	**2**	**3**	**2**	**1**	**2**	**1**	**0**	**1**	**0**	**1**	**2**	**1**	**2**	**1**	**0**	**1**	**0**

how do we find LCA(i,j)?

Compute RMQ(i', j') in D

Solving LCAs using RMQs

Compute an Euler tour of T...

(a depth first search with repeats)

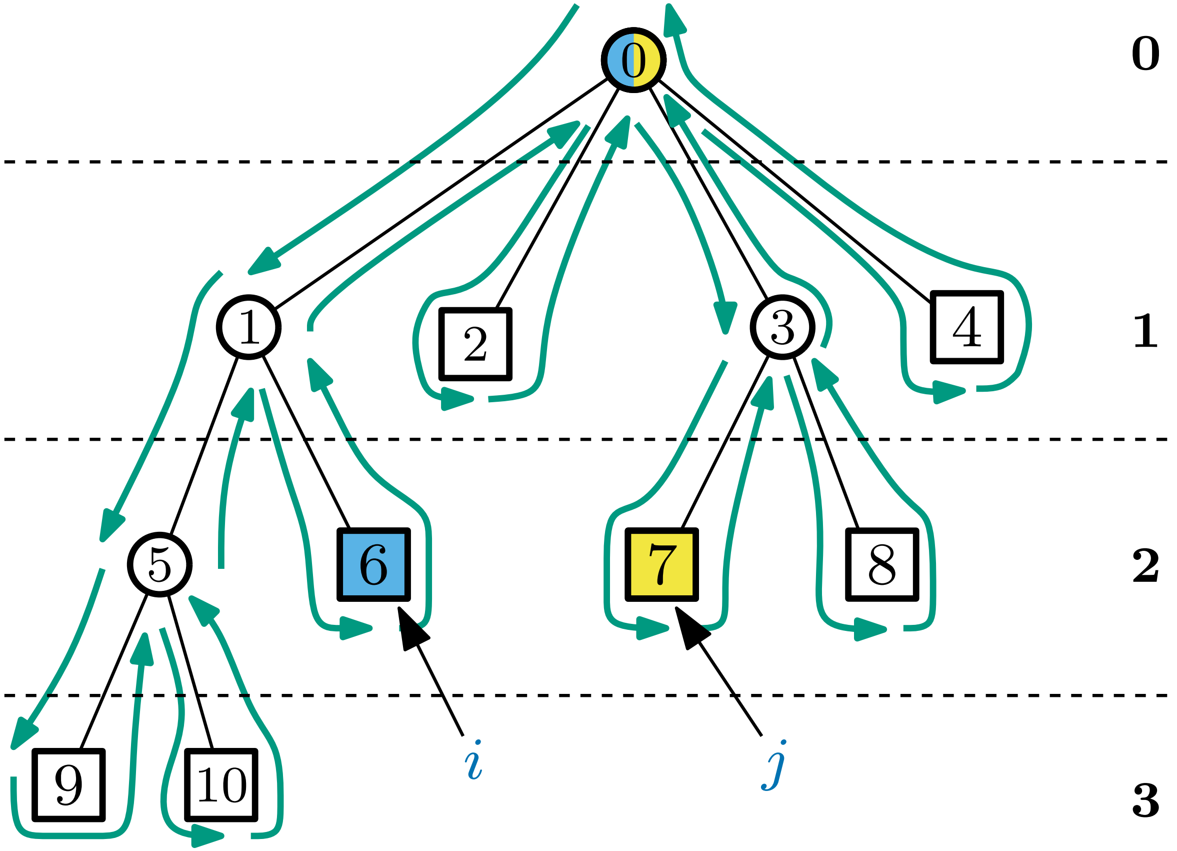
Write down every node you visit

...and its depth

How long is the tour?

We follow each edge twice...

and there are $(n-1)$ edges



Find i and j in N

(node) N	0	1	5	9	5	10	5	1	6	1	0	2	0	3	7	3	8	3	0	4	0
(depth) D	**0**	**1**	**2**	**3**	**2**	**3**	**2**	**1**	**2**	**1**	**0**	**1**	**0**	**1**	**2**	**1**	**2**	**1**	**0**	**1**	**0**

$2n-1$

i' j'

Compute RMQ(i', j') in D

how do we find LCA(i,j)?

Solving LCAs using RMQs

Compute an Euler tour of T...

(a depth first search with repeats)

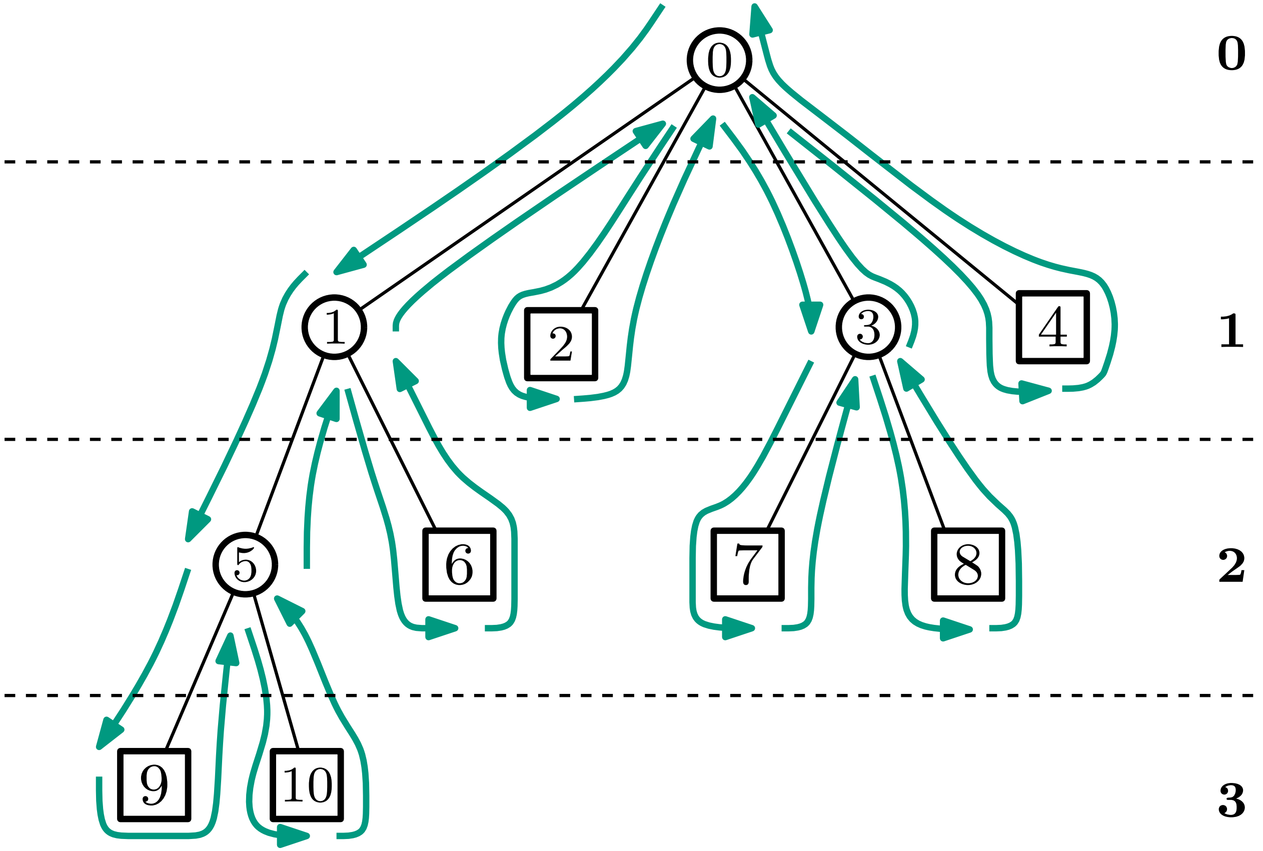
Write down every node you visit

...and its depth

How long is the tour?

We follow each edge twice...

and there are $(n-1)$ edges



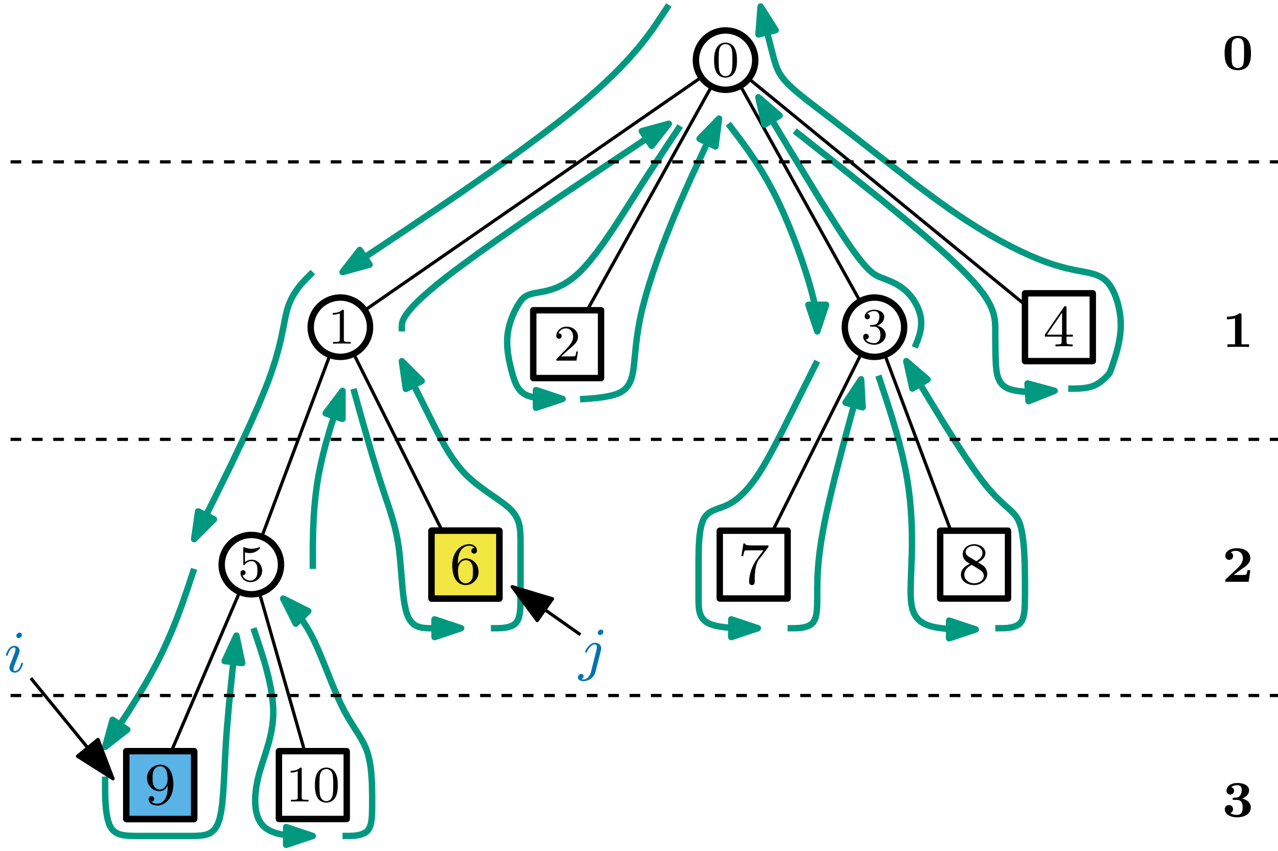
(node) N	0	1	5	9	5	10	5	1	6	1	0	2	0	3	7	3	8	3	0	4	0

$2n-1$

(depth) D	**0**	**1**	**2**	**3**	**2**	**3**	**2**	**1**	**2**	**1**	**0**	**1**	**0**	**1**	**2**	**1**	**2**	**1**	**0**	**1**	**0**

how do we find LCA(i,j)?

Solving LCAs using RMQs



Compute an Euler tour of T...

(a depth first search with repeats)

Write down every node you visit

...and its depth

How long is the tour?

We follow each edge twice...

and there are $(n-1)$ edges

(node) N	0	1	5	9	5	10	5	1	6	1	0	2	0	3	7	3	8	3	0	4	0

$2n-1$

(depth) D	**0**	**1**	**2**	**3**	**2**	**3**	**2**	**1**	**2**	**1**	**0**	**1**	**0**	**1**	**2**	**1**	**2**	**1**	**0**	**1**	**0**

how do we find LCA(i,j)?

Solving LCAs using RMQs

Compute an Euler tour of T...

(a depth first search with repeats)

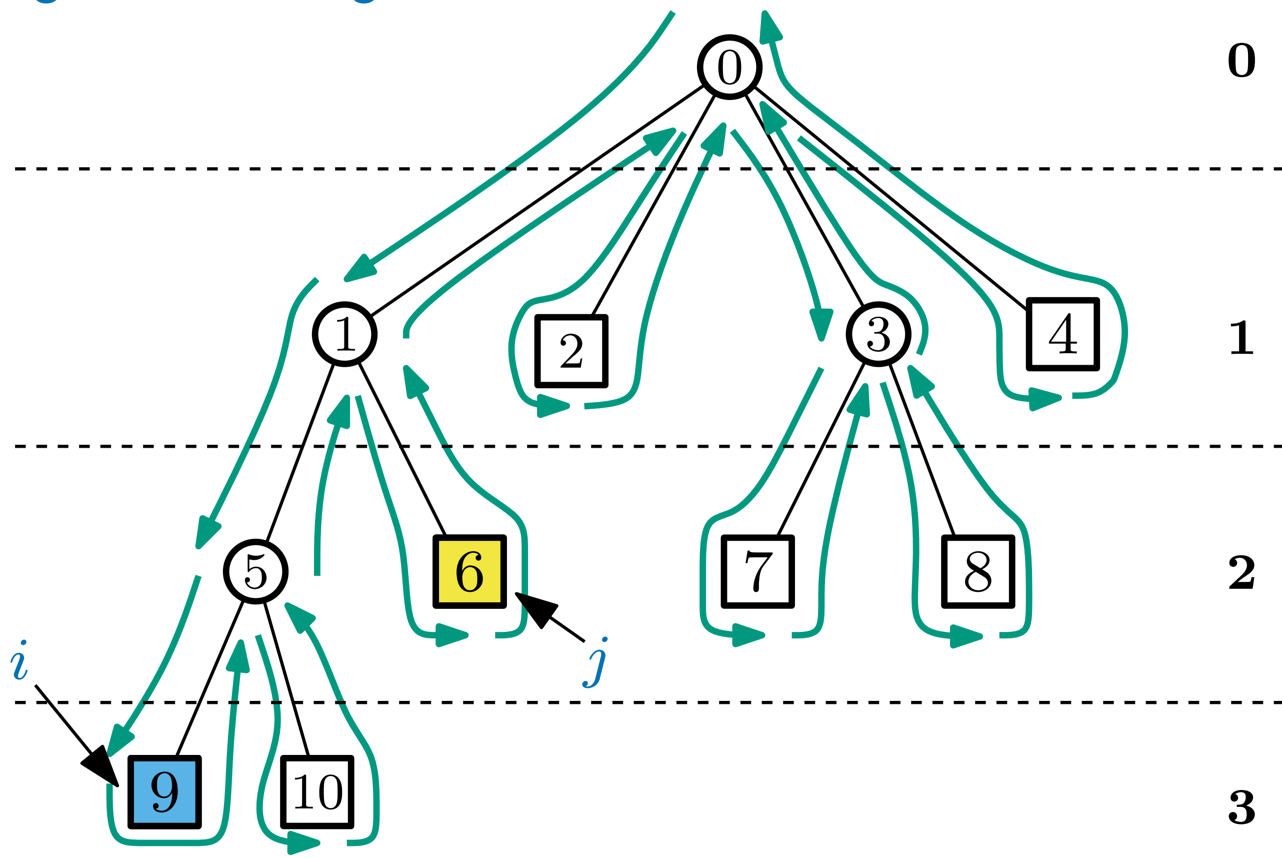
Write down every node you visit

...and its depth

How long is the tour?

We follow each edge twice...

and there are $(n-1)$ edges



Find i and j in N

(node) N	0	1	5	9	5	10	5	1	6	1	0	2	0	3	7	3	8	3	0	4	0

$2n-1$

(depth) D	**0**	**1**	**2**	**3**	**2**	**3**	**2**	**1**	**2**	**1**	**0**	**1**	**0**	**1**	**2**	**1**	**2**	**1**	**0**	**1**	**0**

how do we find LCA(i,j)?

Solving LCAs using RMQs

Compute an Euler tour of T...

(a depth first search with repeats)

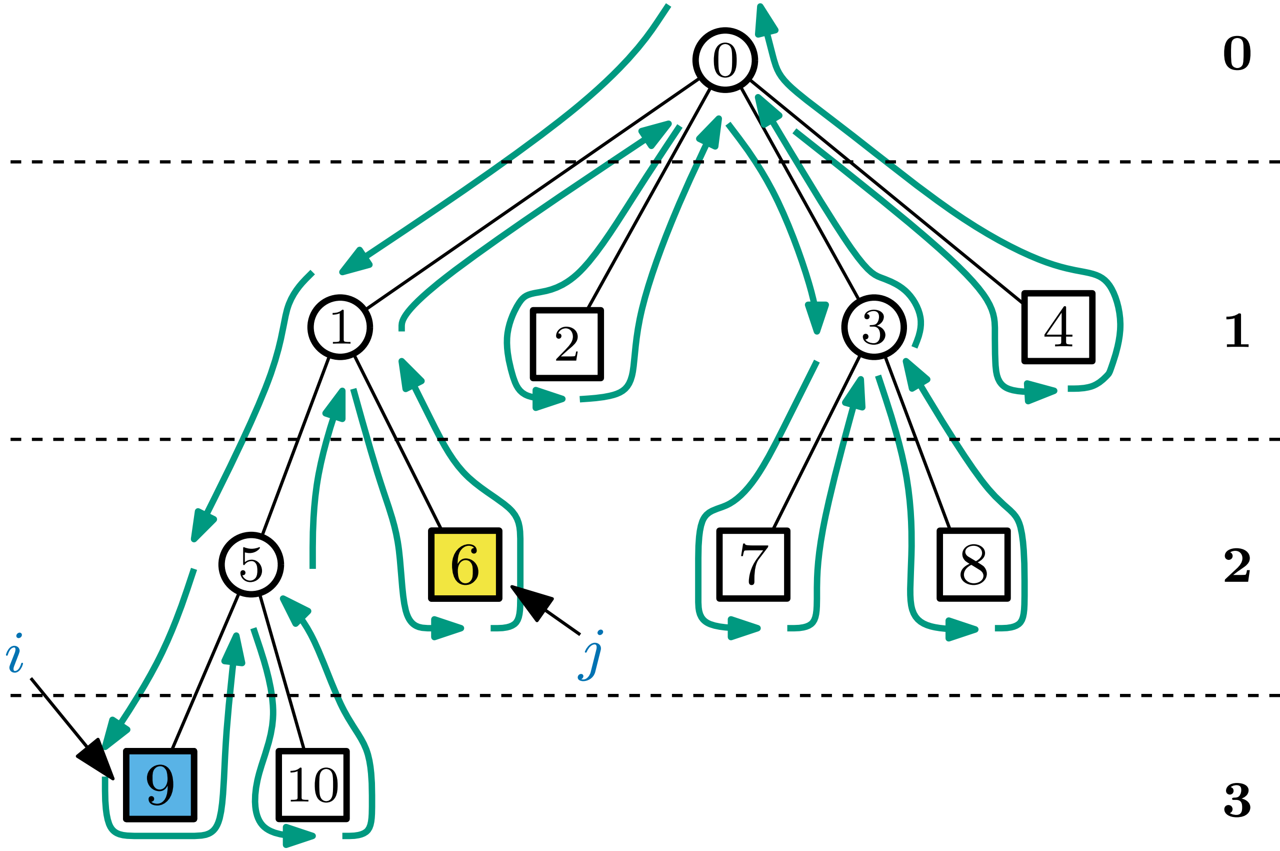
Write down every node you visit

. . . and its depth

How long is the tour?

We follow each edge twice. . .

and there are $(n-1)$ edges



Find i and j in N

(node) N	0	1	5	9	5	10	5	1	6	1	0	2	0	3	7	3	8	3	0	4	0
(depth) D	**0**	**1**	**2**	**3**	**2**	**3**	**2**	**1**	**2**	**1**	**0**	**1**	**0**	**1**	**2**	**1**	**2**	**1**	**0**	**1**	**0**

$2n-1$

i' j'

how do we find LCA(i,j)?

Solving LCAs using RMQs

Compute an Euler tour of T...

(a depth first search with repeats)

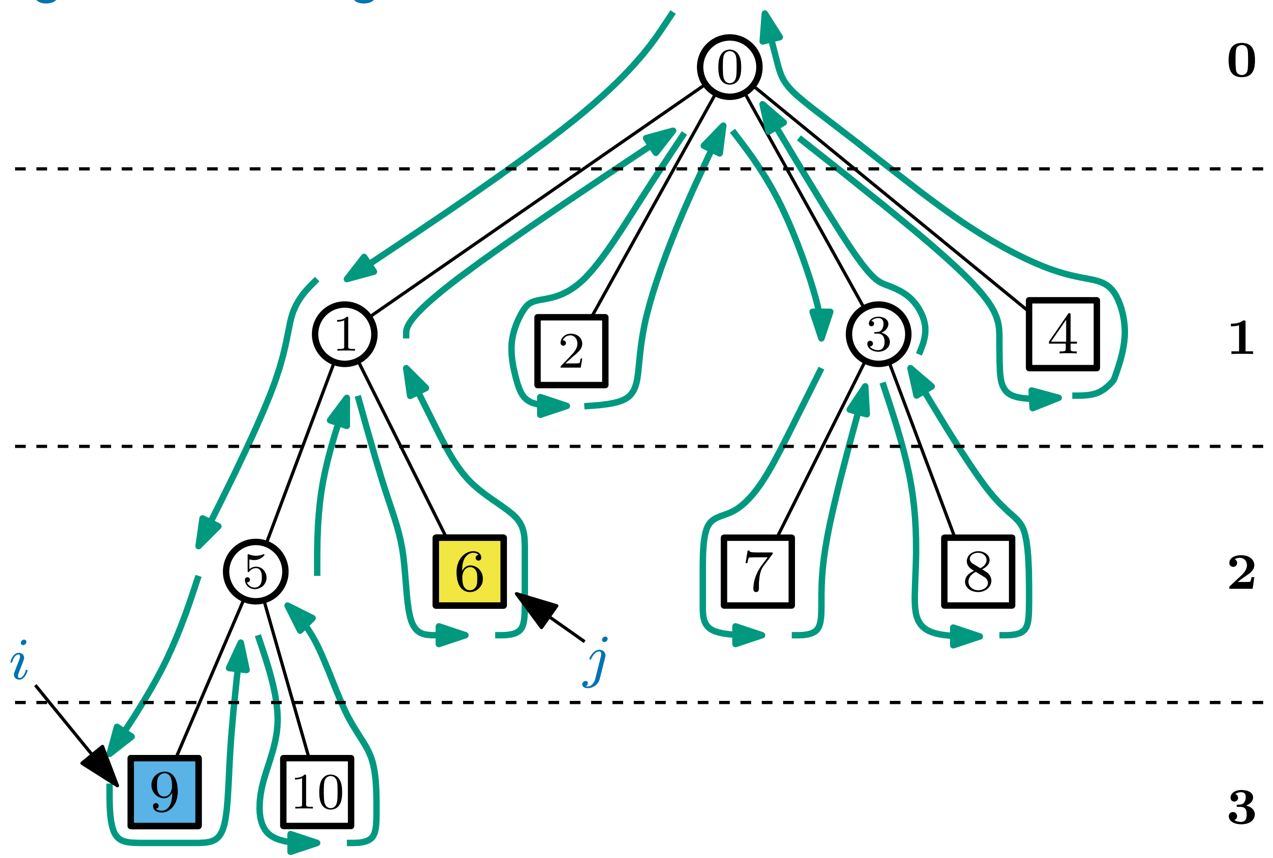
Write down every node you visit

...and its depth

How long is the tour?

We follow each edge twice...

and there are $(n-1)$ edges



Find i and j in N

(node) N	0	1	5	9	5	10	5	1	6	1	0	2	0	3	7	3	8	3	0	4	0
(depth) D	**0**	**1**	**2**	**3**	**2**	**3**	**2**	**1**	**2**	**1**	**0**	**1**	**0**	**1**	**2**	**1**	**2**	**1**	**0**	**1**	**0**

$2n-1$

i' j'

how do we find LCA(i,j)?

Compute RMQ(i', j') in D

Solving LCAs using RMQs

Compute an Euler tour of T...

(a depth first search with repeats)

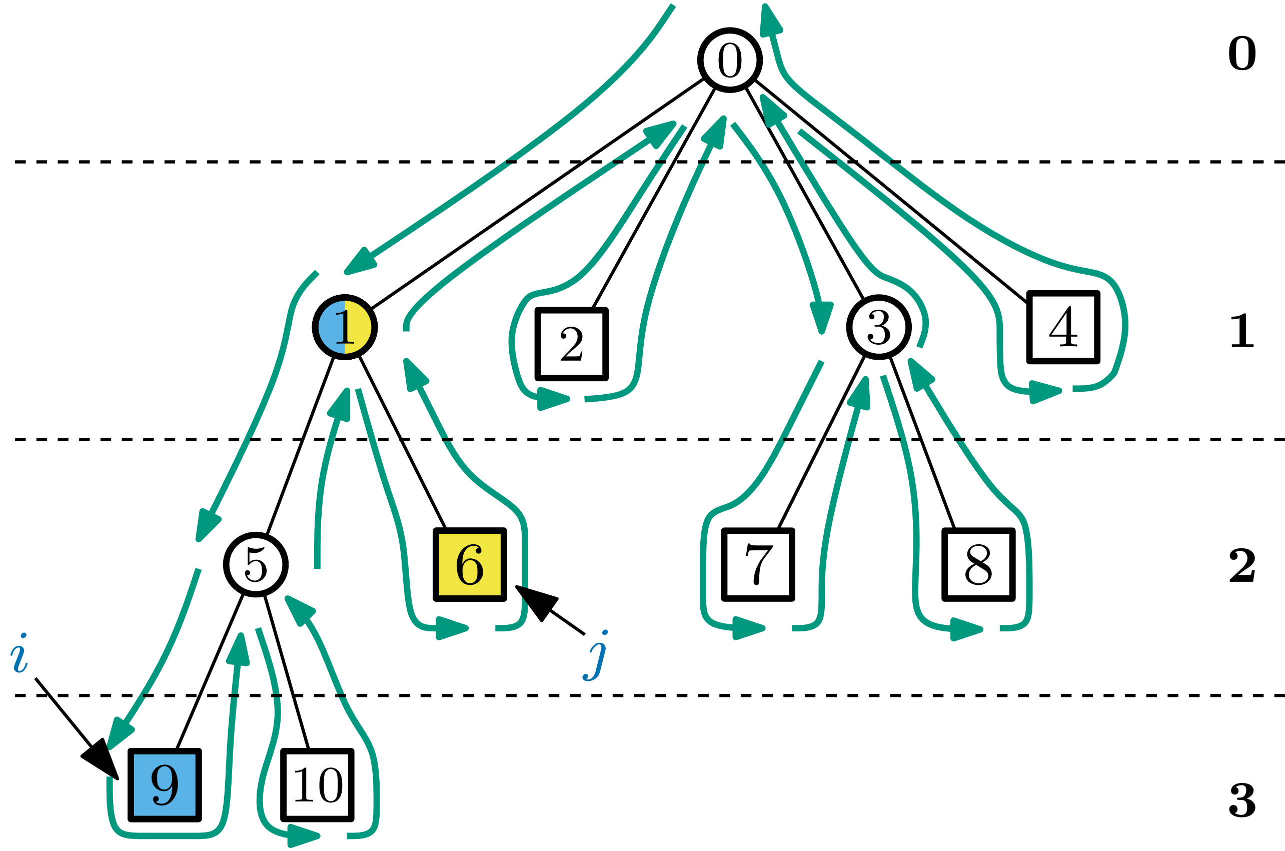
Write down every node you visit

...and its depth

How long is the tour?

We follow each edge twice...

and there are $(n-1)$ edges



Find i and j in N

(node) N	0	1	5	9	5	10	5	1	6	1	0	2	0	3	7	3	8	3	0	4	0
(depth) D	**0**	**1**	**2**	**3**	**2**	**3**	**2**	**1**	**2**	**1**	**0**	**1**	**0**	**1**	**2**	**1**	**2**	**1**	**0**	**1**	**0**

$2n-1$

i' j'

how do we find LCA(i,j)?

Compute RMQ(i', j') in D

Solving LCAs using RMQs

Compute an Euler tour of T...

(a depth first search with repeats)

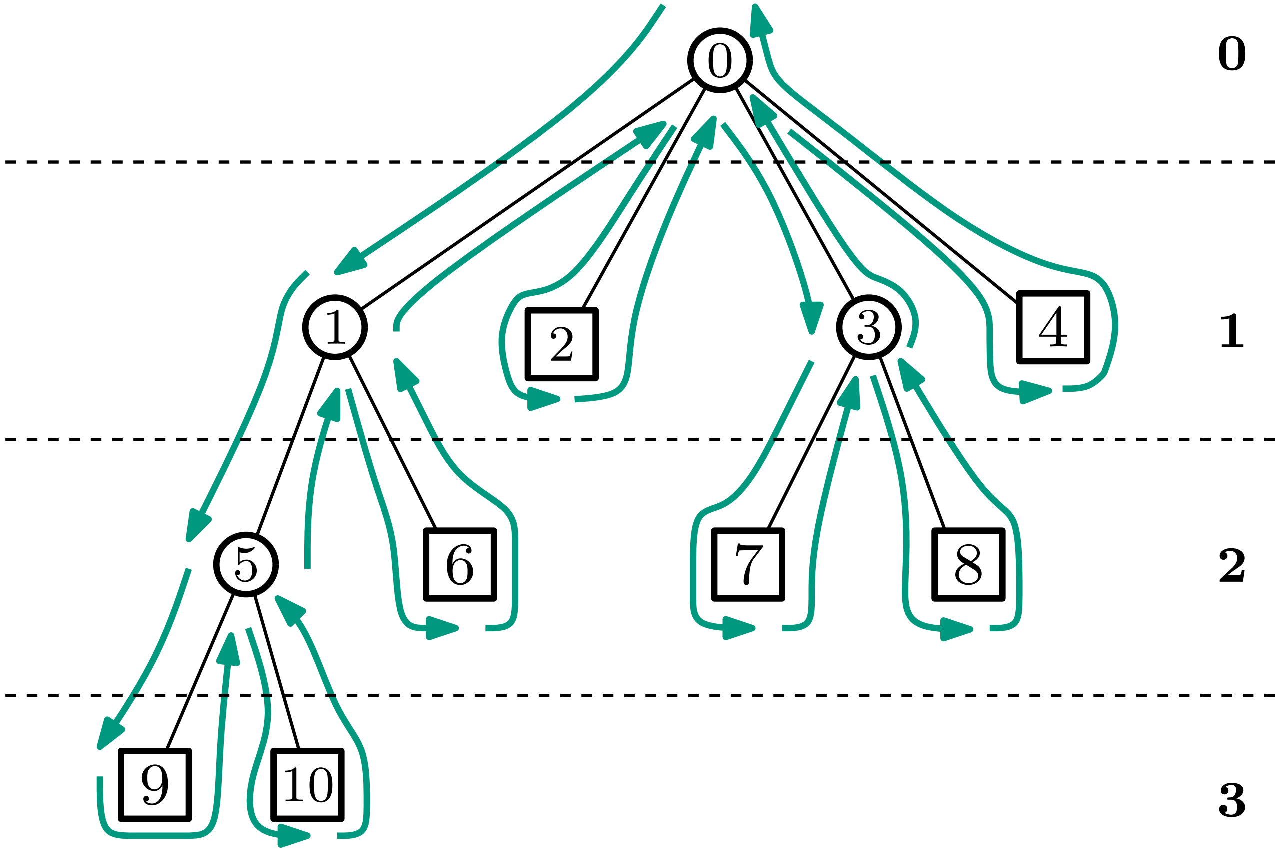
Write down every node you visit

...and its depth

How long is the tour?

We follow each edge twice...

and there are $(n-1)$ edges



(node) N	0	1	5	9	5	10	5	1	6	1	0	2	0	3	7	3	8	3	0	4	0

$2n-1$

(depth) D	**0**	**1**	**2**	**3**	**2**	**3**	**2**	**1**	**2**	**1**	**0**	**1**	**0**	**1**	**2**	**1**	**2**	**1**	**0**	**1**	**0**

how do we find LCA(i,j)?

Solving LCAs using RMQs

Compute an Euler tour of T...

(a depth first search with repeats)

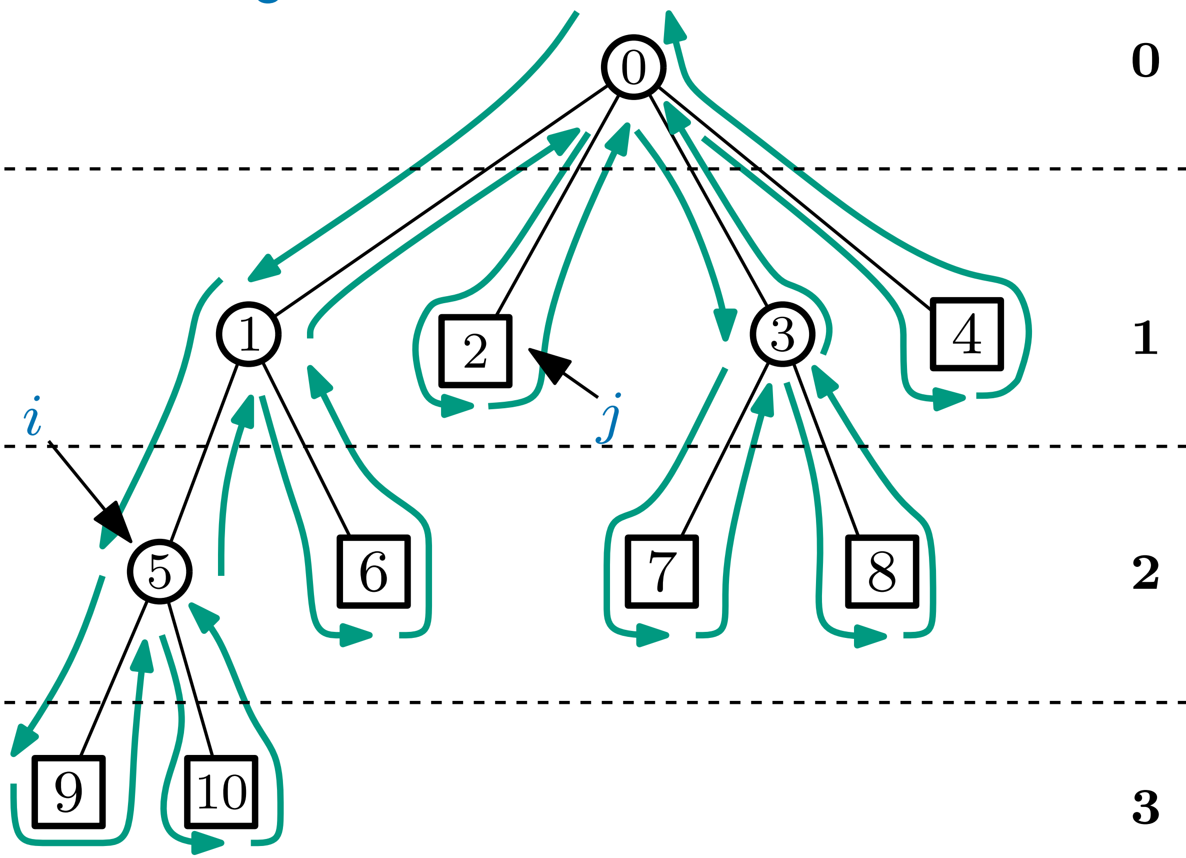
Write down every node you visit

... and its depth

How long is the tour?

We follow each edge twice...

and there are $(n-1)$ edges



(node) N	0	1	5	9	5	10	5	1	6	1	0	2	0	3	7	3	8	3	0	4	0

$2n-1$

(depth) D	**0**	**1**	**2**	**3**	**2**	**3**	**2**	**1**	**2**	**1**	**0**	**1**	**0**	**1**	**2**	**1**	**2**	**1**	**0**	**1**	**0**

how do we find LCA(i,j)?

Solving LCAs using RMQs

Compute an Euler tour of T...

(a depth first search with repeats)

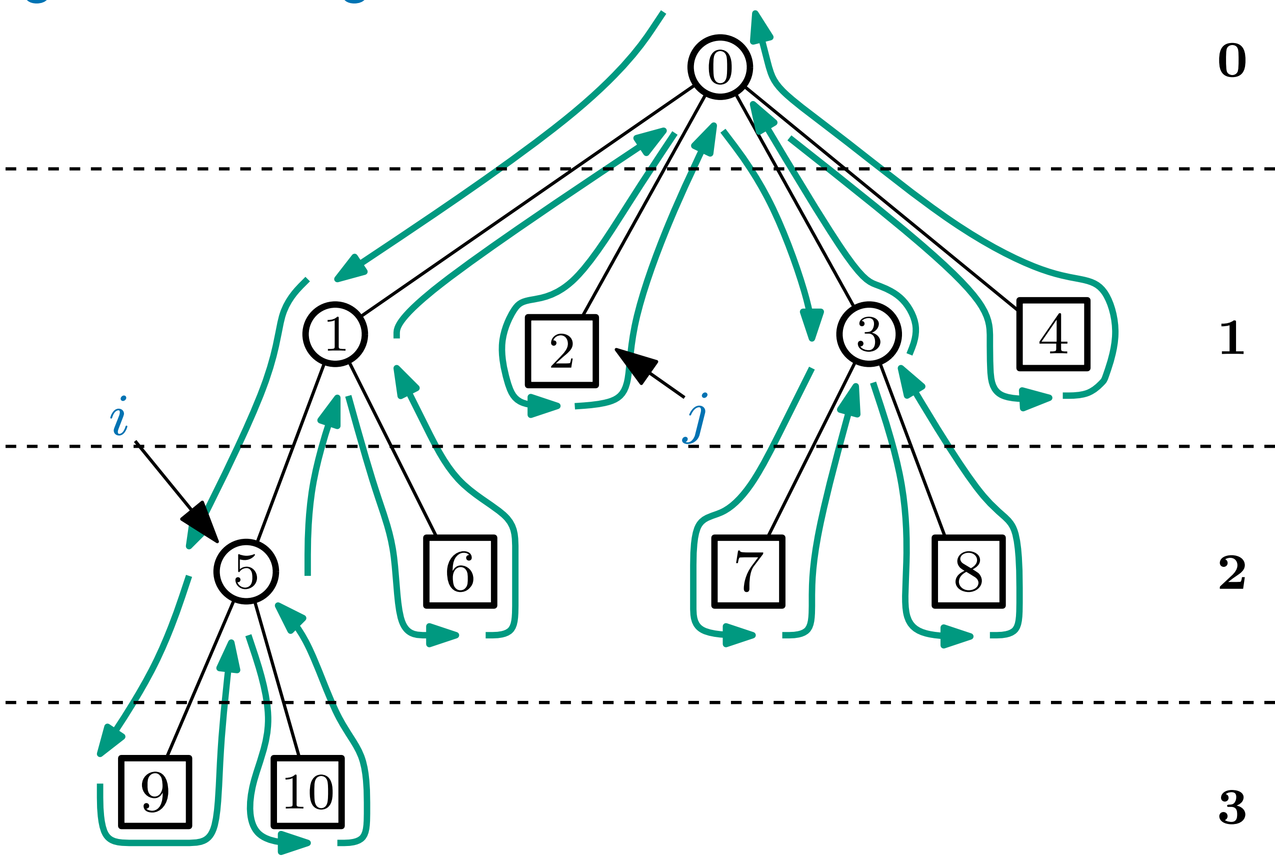
Write down every node you visit

...and its depth

How long is the tour?

We follow each edge twice...

and there are $(n-1)$ edges



Find i and j in N...

(node) N	0	1	5	9	5	10	5	1	6	1	0	2	0	3	7	3	8	3	0	4	0

$2n-1$

(depth) D	**0**	**1**	**2**	**3**	**2**	**3**	**2**	**1**	**2**	**1**	**0**	**1**	**0**	**1**	**2**	**1**	**2**	**1**	**0**	**1**	**0**

how do we find LCA(i,j)?

Solving LCAs using RMQs

Compute an Euler tour of T...

(a depth first search with repeats)

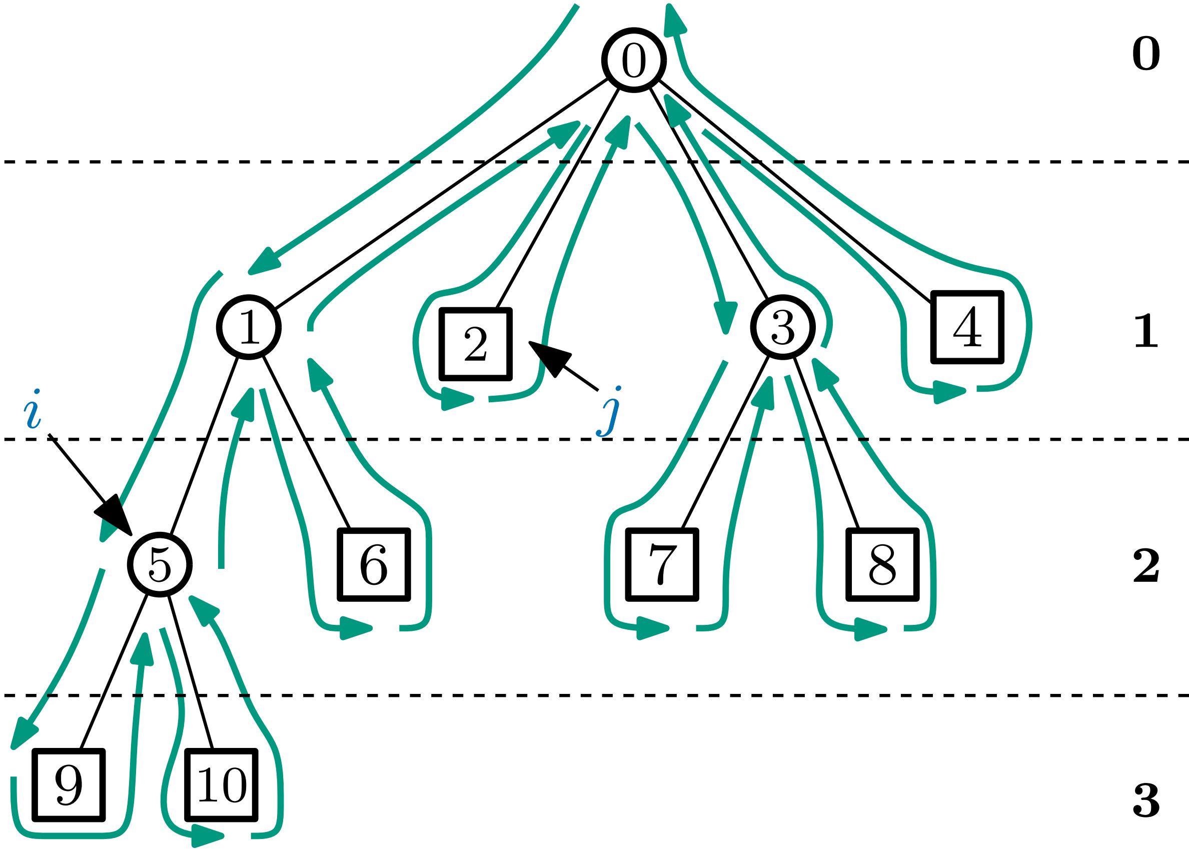
Write down every node you visit

...and its depth

How long is the tour?

We follow each edge twice...

and there are $(n-1)$ edges



Find i and j in N... which copy of i?

(node) N	0	1	5	9	5	10	5	1	6	1	0	2	0	3	7	3	8	3	0	4	0

$2n-1$

(depth) D	**0**	**1**	**2**	**3**	**2**	**3**	**2**	**1**	**2**	**1**	**0**	**1**	**0**	**1**	**2**	**1**	**2**	**1**	**0**	**1**	**0**

how do we find LCA(i,j)?

Solving LCAs using RMQs

Compute an Euler tour of T...

(a depth first search with repeats)

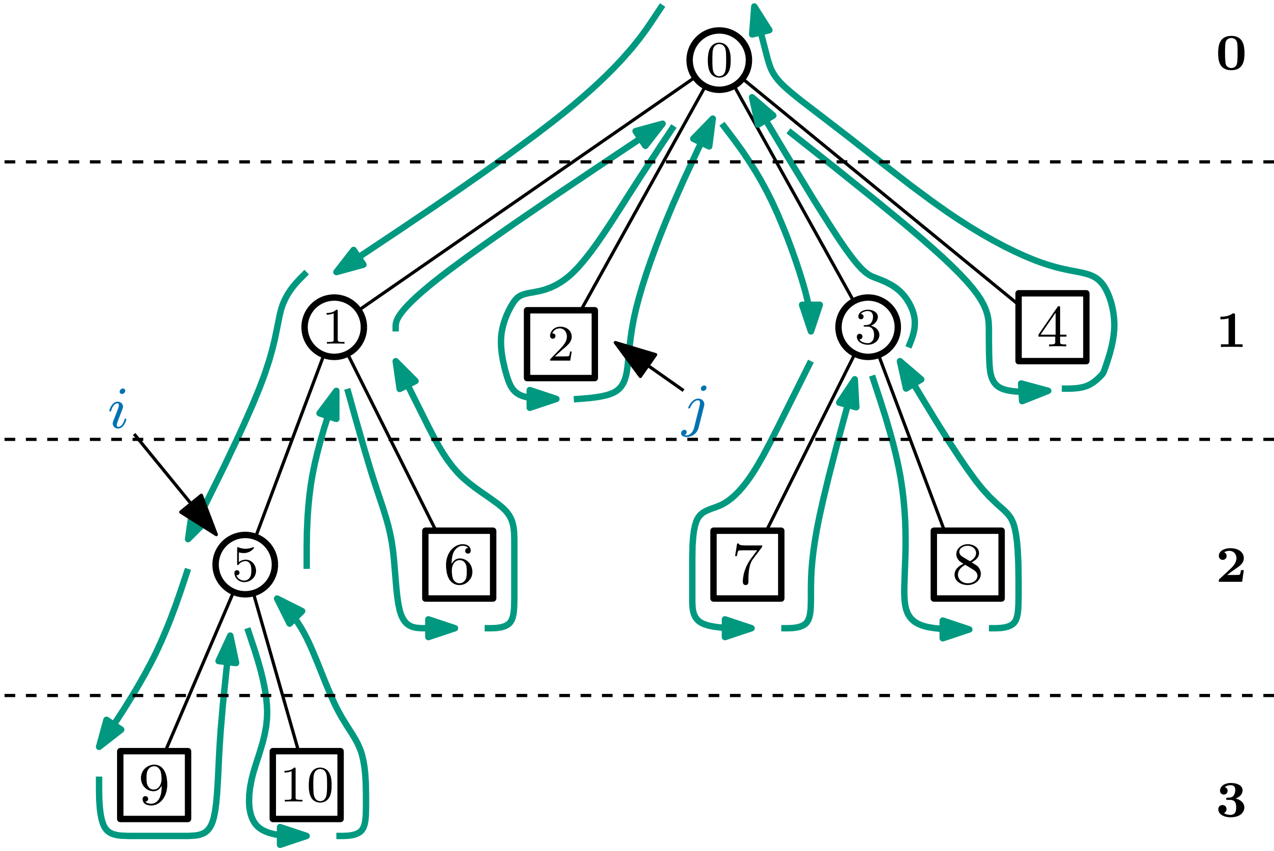
Write down every node you visit

...and its depth

How long is the tour?

We follow each edge twice...

and there are $(n-1)$ edges



Find i and j in N... which copy of i?

any copy is fine

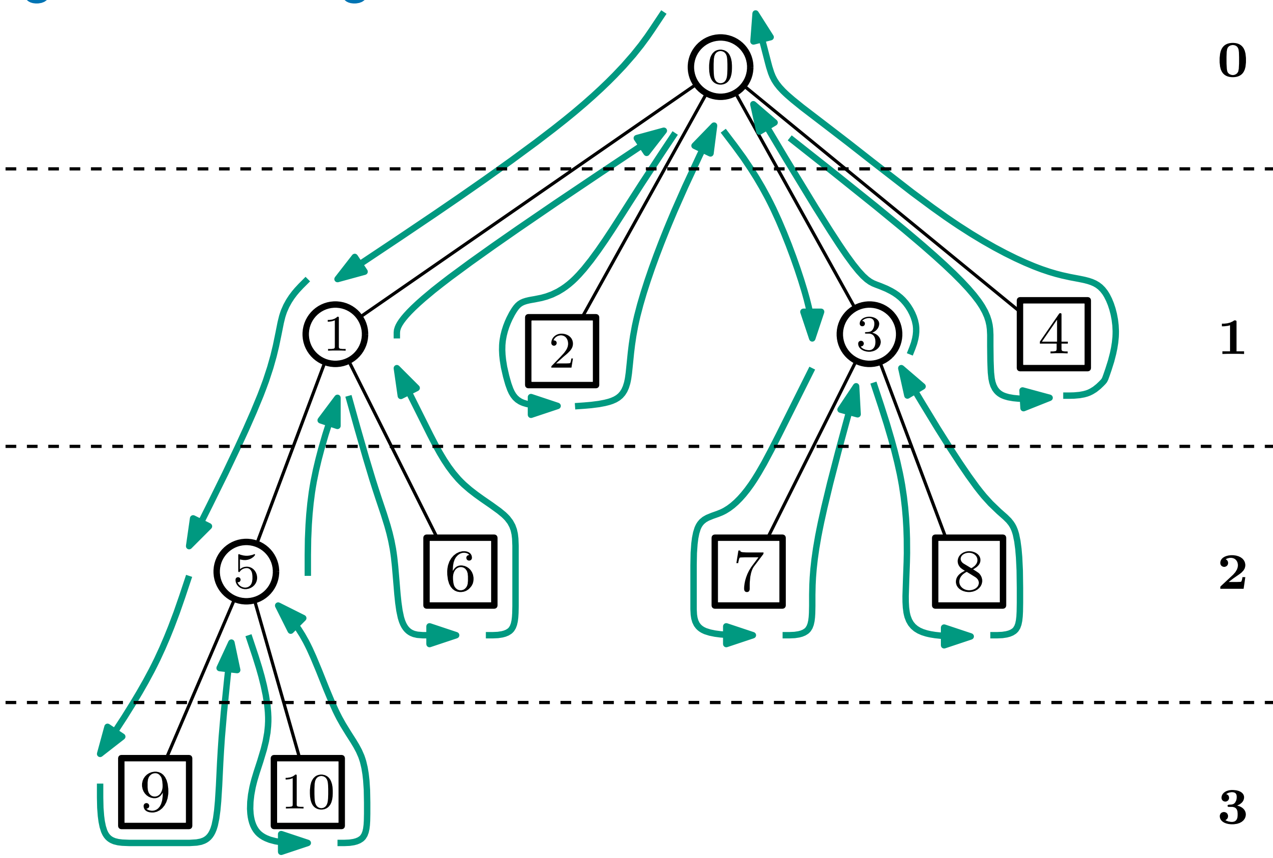
(node) N	0	1	5	9	5	10	5	1	6	1	0	2	0	3	7	3	8	3	0	4	0

$2n-1$

(depth) D	**0**	**1**	**2**	**3**	**2**	**3**	**2**	**1**	**2**	**1**	**0**	**1**	**0**	**1**	**2**	**1**	**2**	**1**	**0**	**1**	**0**

how do we find LCA(i,j)?

Solving LCAs using RMQs



(node) N	0	1	5	9	5	10	5	1	6	1	0	2	0	3	7	3	8	3	0	4	0

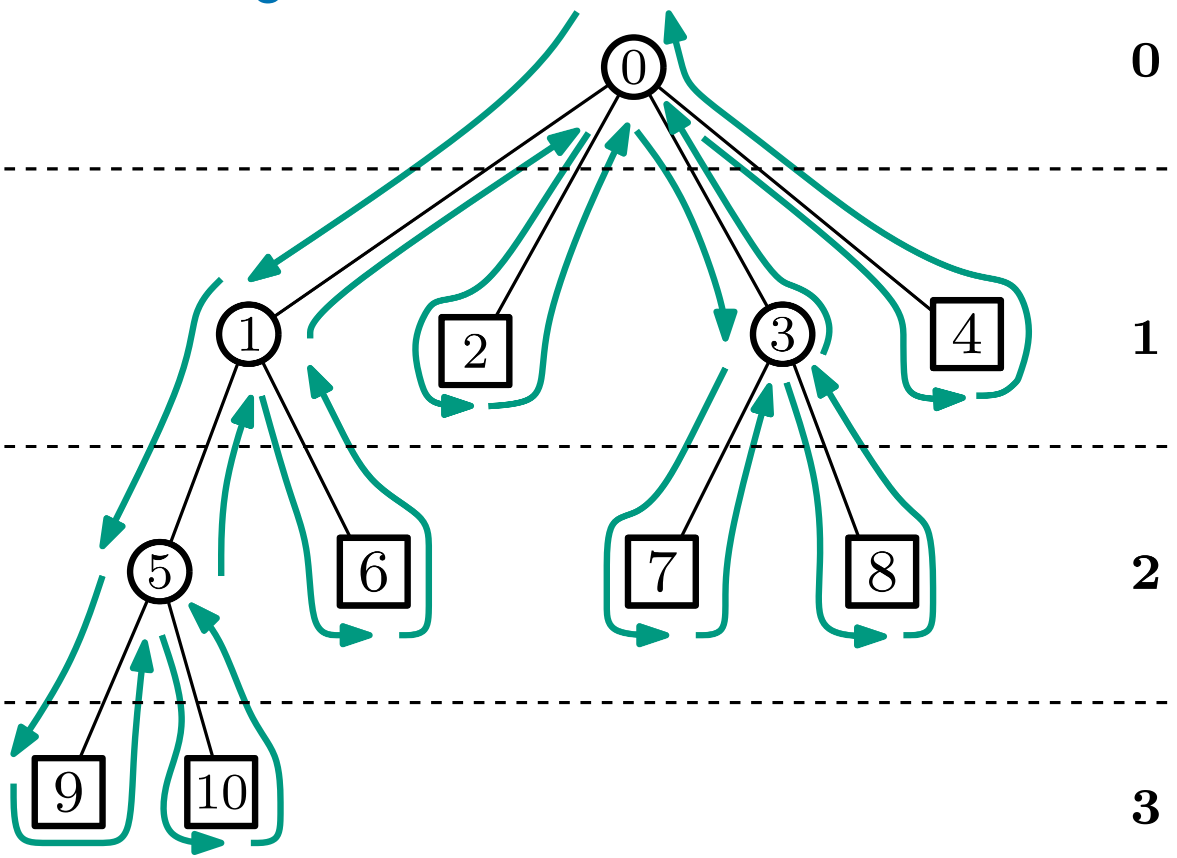
$2n-1$

(depth) D	**0**	**1**	**2**	**3**	**2**	**3**	**2**	**1**	**2**	**1**	**0**	**1**	**0**	**1**	**2**	**1**	**2**	**1**	**0**	**1**	**0**

Solving LCAs using RMQs

Preprocessing Summary

1. Construct N and D from T
2. Add a pointer from each node i to some $N[i'] = i$
3. Preprocess D for RMQs



(node) N	0	1	5	9	5	10	5	1	6	1	0	2	0	3	7	3	8	3	0	4	0

$2n-1$

(depth) D	**0**	**1**	**2**	**3**	**2**	**3**	**2**	**1**	**2**	**1**	**0**	**1**	**0**	**1**	**2**	**1**	**2**	**1**	**0**	**1**	**0**

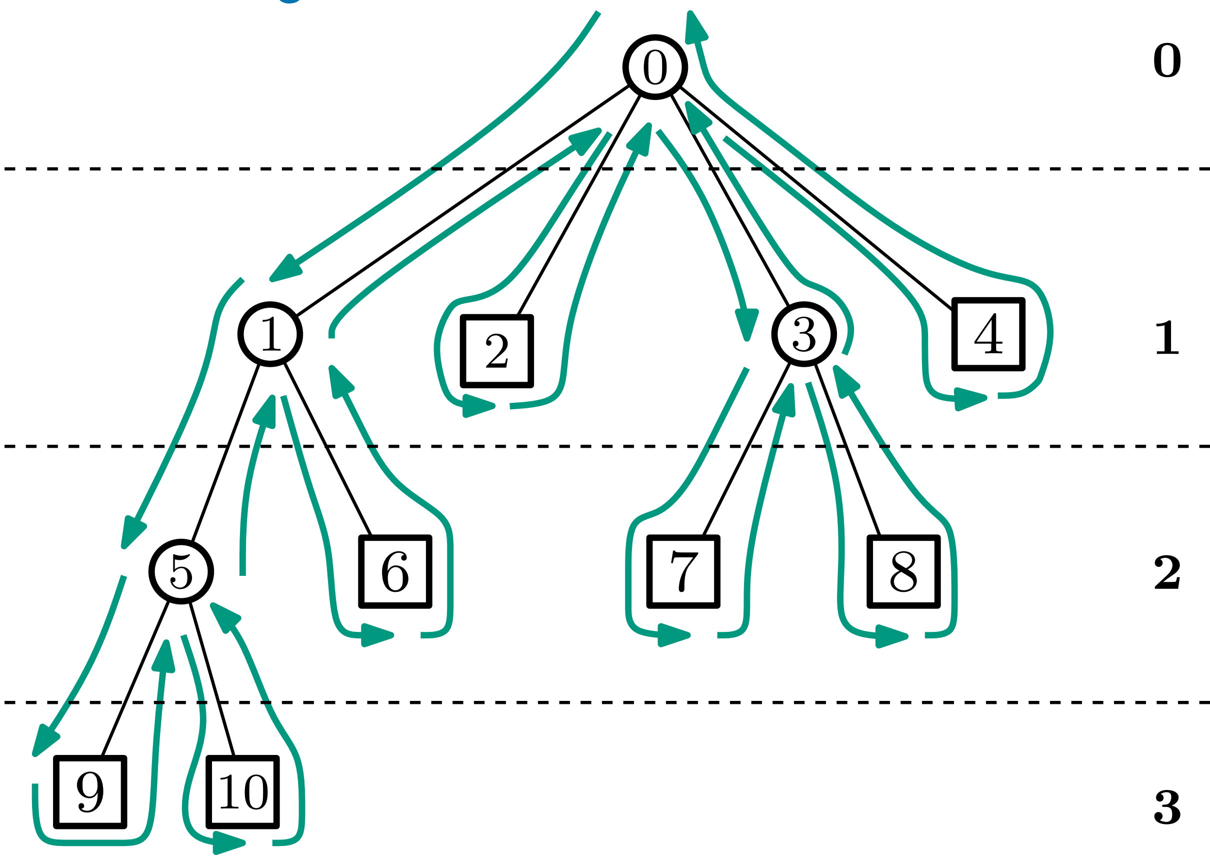
Solving LCAs using RMQs

Preprocessing Summary

1. Construct N and D from T
2. Add a pointer from each node i to some $N[i'] = i$
3. Preprocess D for RMQs

Query Summary - LCA(i,j)

1. Find (any) i' st. $N[i'] = i$
2. Find (any) j' st. $N[j'] = j$
3. Compute RMQ(i', j') in D
4. LCA$(i, j) = N[$RMQ$(i', j')]$



(node) N	0	1	5	9	5	10	5	1	6	1	0	2	0	3	7	3	8	3	0	4	0
(depth) D	**0**	**1**	**2**	**3**	**2**	**3**	**2**	**1**	**2**	**1**	**0**	**1**	**0**	**1**	**2**	**1**	**2**	**1**	**0**	**1**	**0**

$2n-1$

Solving LCAs using RMQs

Preprocessing Summary

$O(n)$ 1. Construct N and D from T

2. Add a pointer from each node i to some $N[i'] = i$

3. Preprocess D for RMQs

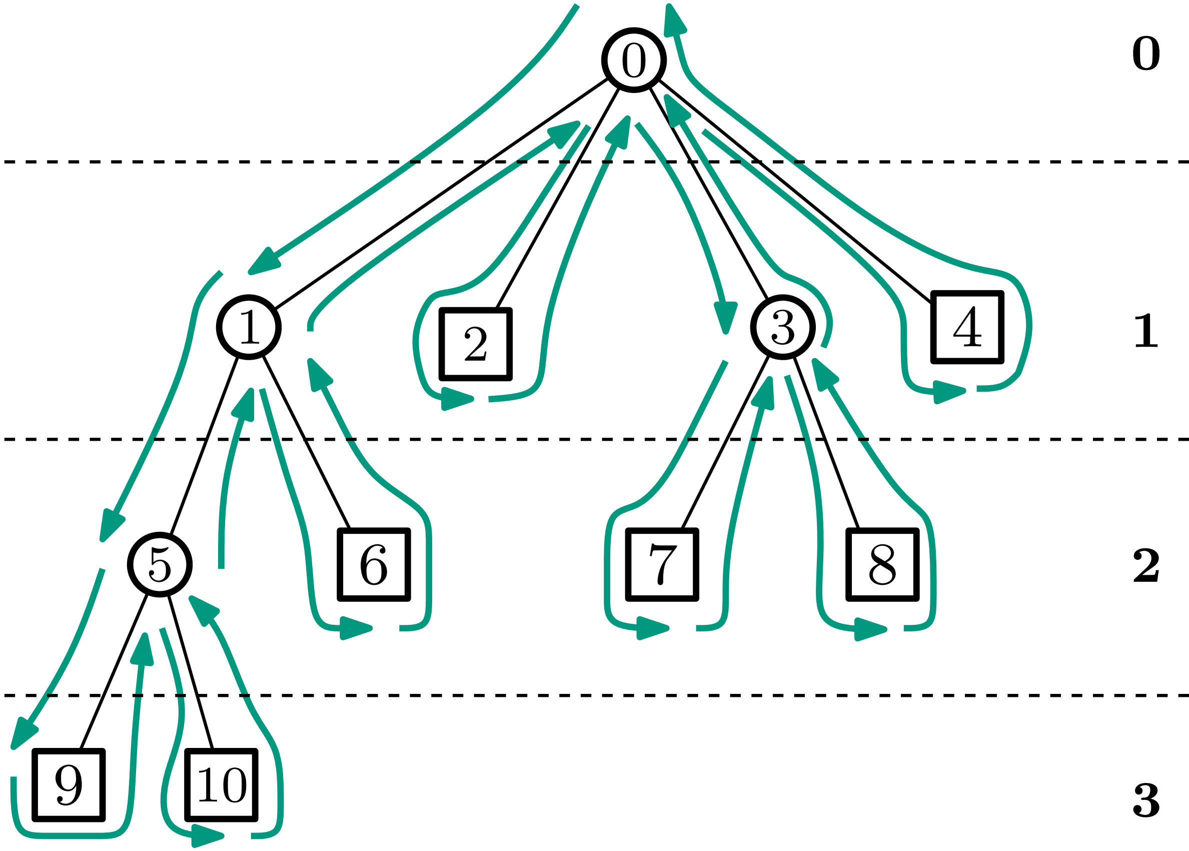
Query Summary - LCA(i,j)

1. Find (any) i' st. $N[i'] = i$

2. Find (any) j' st. $N[j'] = j$

3. Compute RMQ(i', j') in D

4. LCA$(i, j) = N[$RMQ$(i', j')]$



(node) N	0	1	5	9	5	10	5	1	6	1	0	2	0	3	7	3	8	3	0	4	0
(depth) D	**0**	**1**	**2**	**3**	**2**	**3**	**2**	**1**	**2**	**1**	**0**	**1**	**0**	**1**	**2**	**1**	**2**	**1**	**0**	**1**	**0**

$2n-1$

Solving LCAs using RMQs

Preprocessing Summary

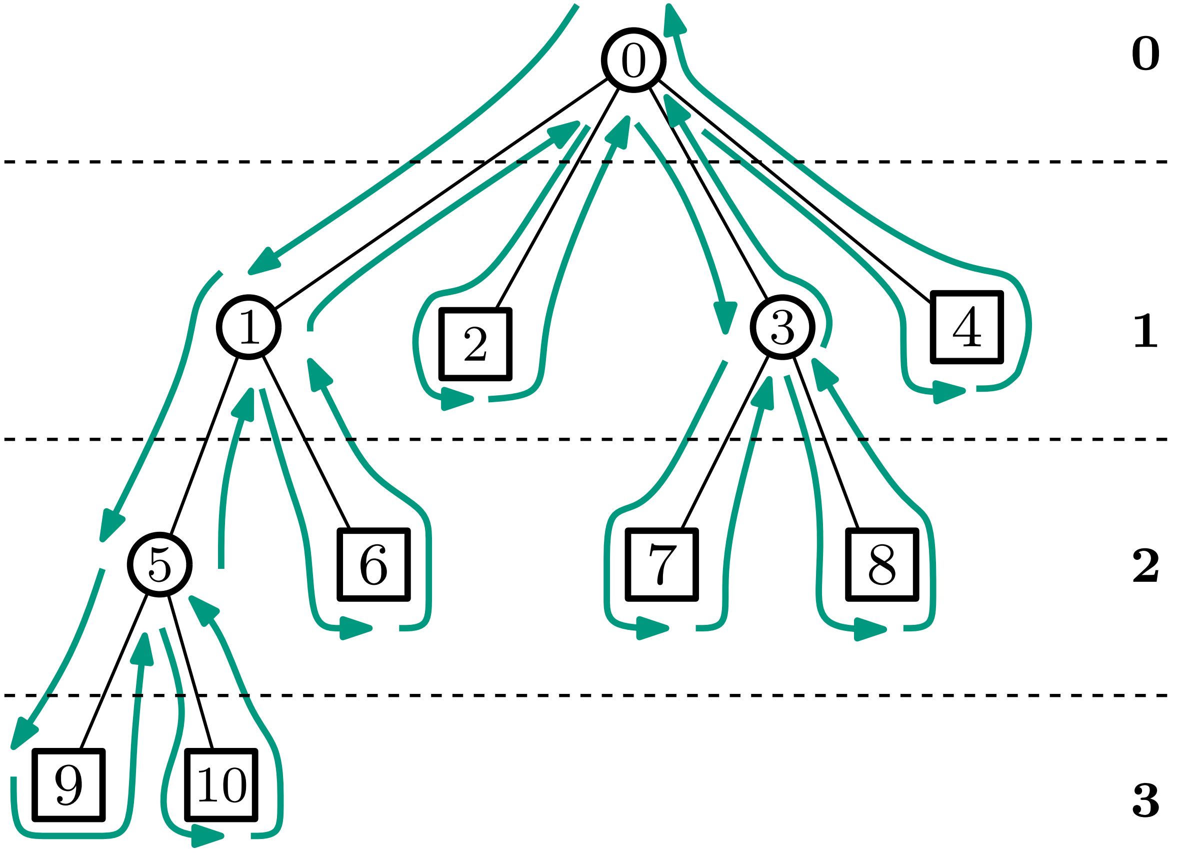
$O(n)$ 1. Construct N and D from T

$O(n)$ 2. Add a pointer from each node i to some $N[i'] = i$

3. Preprocess D for RMQs

Query Summary - LCA(i,j)

1. Find (any) i' st. $N[i'] = i$
2. Find (any) j' st. $N[j'] = j$
3. Compute RMQ(i', j') in D
4. LCA$(i, j) = N[$RMQ$(i', j')]$



(node) N	0	1	5	9	5	10	5	1	6	1	0	2	0	3	7	3	8	3	0	4	0
(depth) D	**0**	**1**	**2**	**3**	**2**	**3**	**2**	**1**	**2**	**1**	**0**	**1**	**0**	**1**	**2**	**1**	**2**	**1**	**0**	**1**	**0**

$2n-1$

Solving LCAs using RMQs

Preprocessing Summary

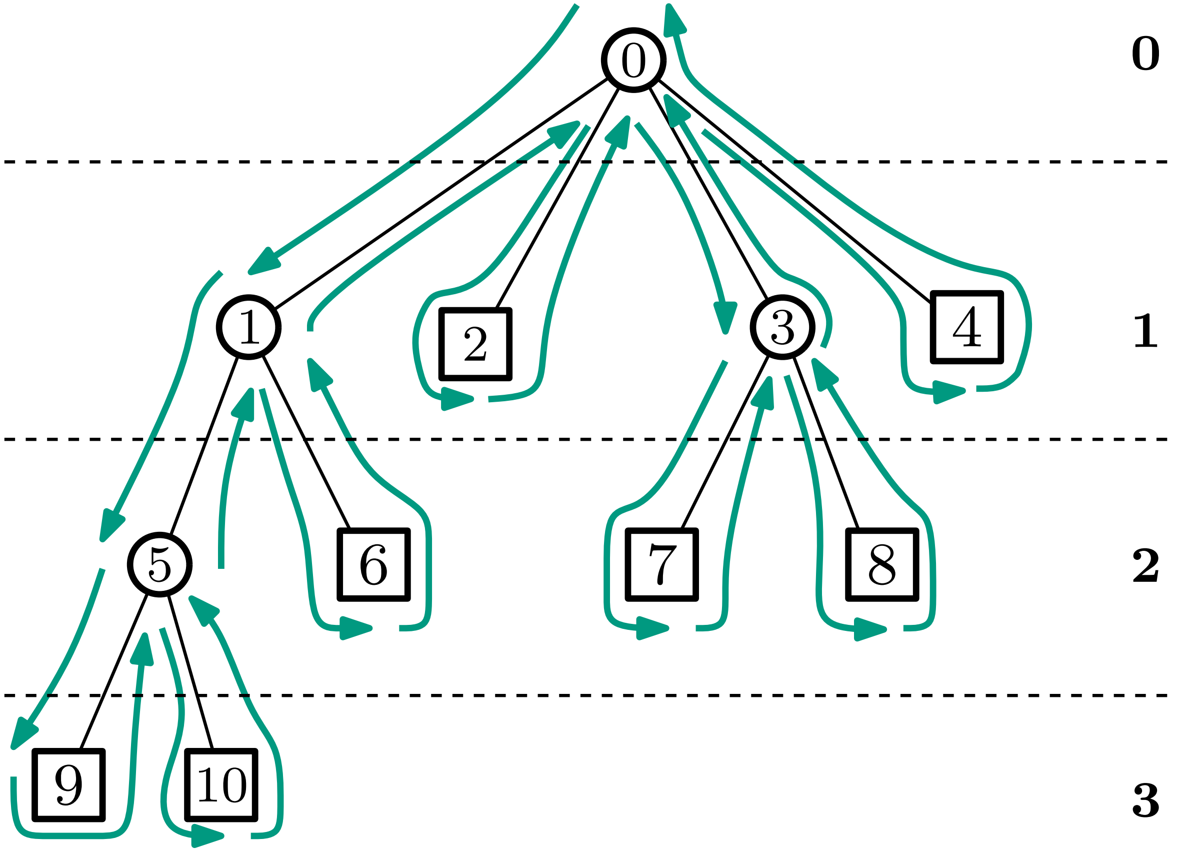
$O(n)$ 1. Construct N and D from T

$O(n)$ 2. Add a pointer from each node i to some $N[i'] = i$

$O(?)$ 3. Preprocess D for RMQs

Query Summary - LCA(i,j)

1. Find (any) i' st. $N[i'] = i$
2. Find (any) j' st. $N[j'] = j$
3. Compute RMQ(i', j') in D
4. LCA$(i, j) = N[$RMQ$(i', j')]$



(node) N	0	1	5	9	5	10	5	1	6	1	0	2	0	3	7	3	8	3	0	4	0

$2n-1$

(depth) D	**0**	**1**	**2**	**3**	**2**	**3**	**2**	**1**	**2**	**1**	**0**	**1**	**0**	**1**	**2**	**1**	**2**	**1**	**0**	**1**	**0**

Solving LCAs using RMQs

Preprocessing Summary

$O(n)$ 1. Construct N and D from T

$O(n)$ 2. Add a pointer from each node i to some $N[i'] = i$

$O(?)$ 3. Preprocess D for RMQs

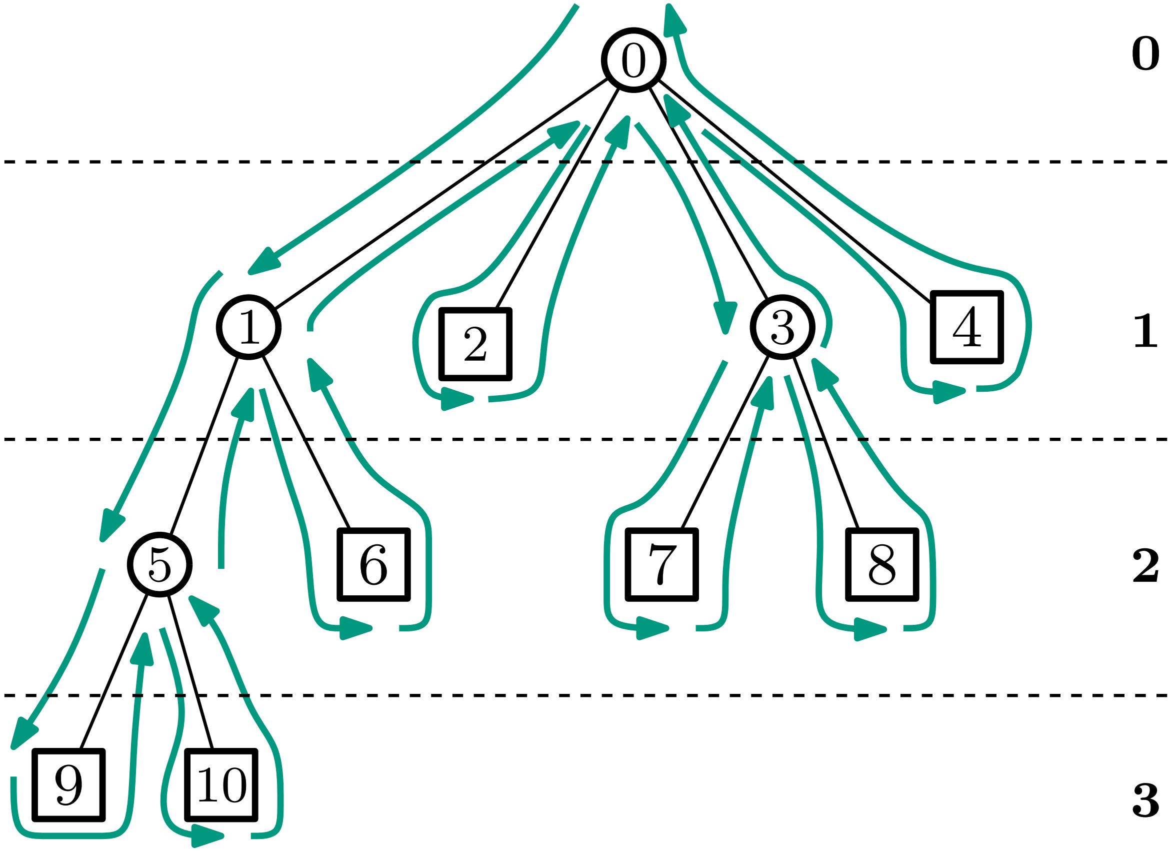
Query Summary - LCA(i,j)

$O(1)$ 1. Find (any) i' st. $N[i'] = i$

2. Find (any) j' st. $N[j'] = j$

3. Compute RMQ(i', j') in D

4. LCA$(i, j) = N[$RMQ$(i', j')]$



(node) N	0	1	5	9	5	10	5	1	6	1	0	2	0	3	7	3	8	3	0	4	0
(depth) D	**0**	**1**	**2**	**3**	**2**	**3**	**2**	**1**	**2**	**1**	**0**	**1**	**0**	**1**	**2**	**1**	**2**	**1**	**0**	**1**	**0**

$2n-1$

Solving LCAs using RMQs

Preprocessing Summary

$O(n)$ 1. Construct N and D from T

$O(n)$ 2. Add a pointer from each node i to some $N[i'] = i$

$O(?)$ 3. Preprocess D for RMQs

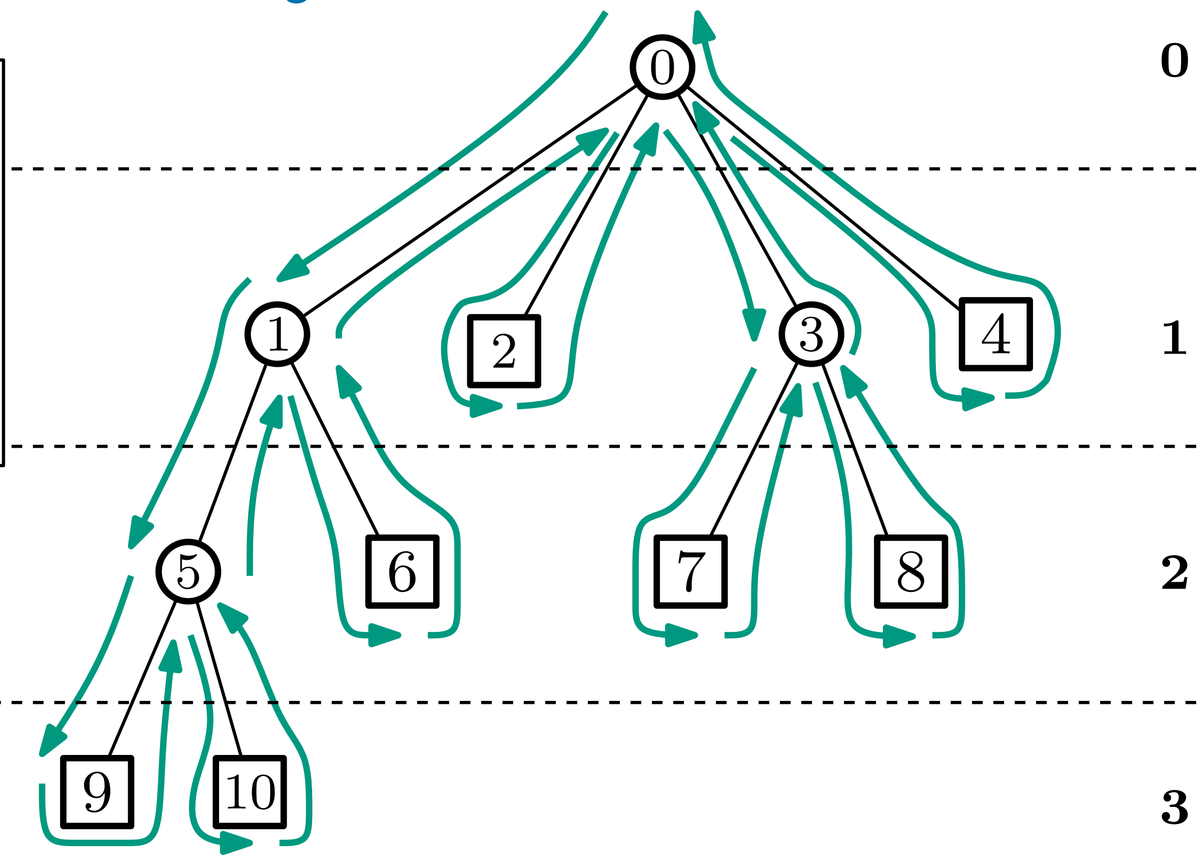
Query Summary - LCA(i,j)

$O(1)$ 1. Find (any) i' st. $N[i'] = i$

$O(1)$ 2. Find (any) j' st. $N[j'] = j$

3. Compute RMQ(i', j') in D

4. LCA$(i, j) = N[$RMQ$(i', j')]$



(node) N	0	1	5	9	5	10	5	1	6	1	0	2	0	3	7	3	8	3	0	4	0

$2n-1$

(depth) D	**0**	**1**	**2**	**3**	**2**	**3**	**2**	**1**	**2**	**1**	**0**	**1**	**0**	**1**	**2**	**1**	**2**	**1**	**0**	**1**	**0**

Solving LCAs using RMQs

Preprocessing Summary

$O(n)$ 1. Construct N and D from T

$O(n)$ 2. Add a pointer from each node i to some $N[i'] = i$

$O(?)$ 3. Preprocess D for RMQs

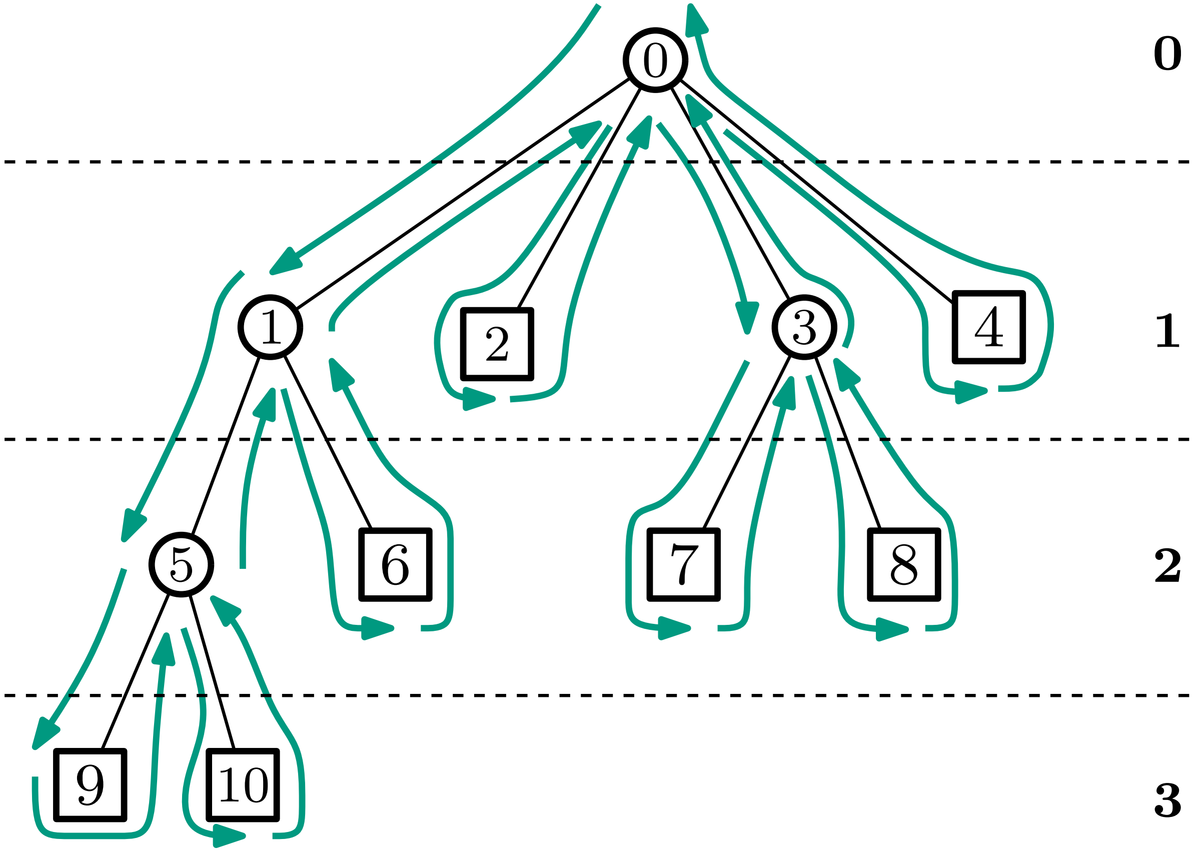
Query Summary - LCA(i,j)

$O(1)$ 1. Find (any) i' st. $N[i'] = i$

$O(1)$ 2. Find (any) j' st. $N[j'] = j$

$O(?)$ 3. Compute RMQ(i', j') in D

4. LCA$(i, j) = N[$RMQ$(i', j')]$



(node) N	0	1	5	9	5	10	5	1	6	1	0	2	0	3	7	3	8	3	0	4	0
(depth) D	**0**	**1**	**2**	**3**	**2**	**3**	**2**	**1**	**2**	**1**	**0**	**1**	**0**	**1**	**2**	**1**	**2**	**1**	**0**	**1**	**0**

$2n-1$

Solving LCAs using RMQs

Preprocessing Summary

$O(n)$ 1. Construct N and D from T

$O(n)$ 2. Add a pointer from each node i to some $N[i'] = i$

$O(?)$ 3. Preprocess D for RMQs

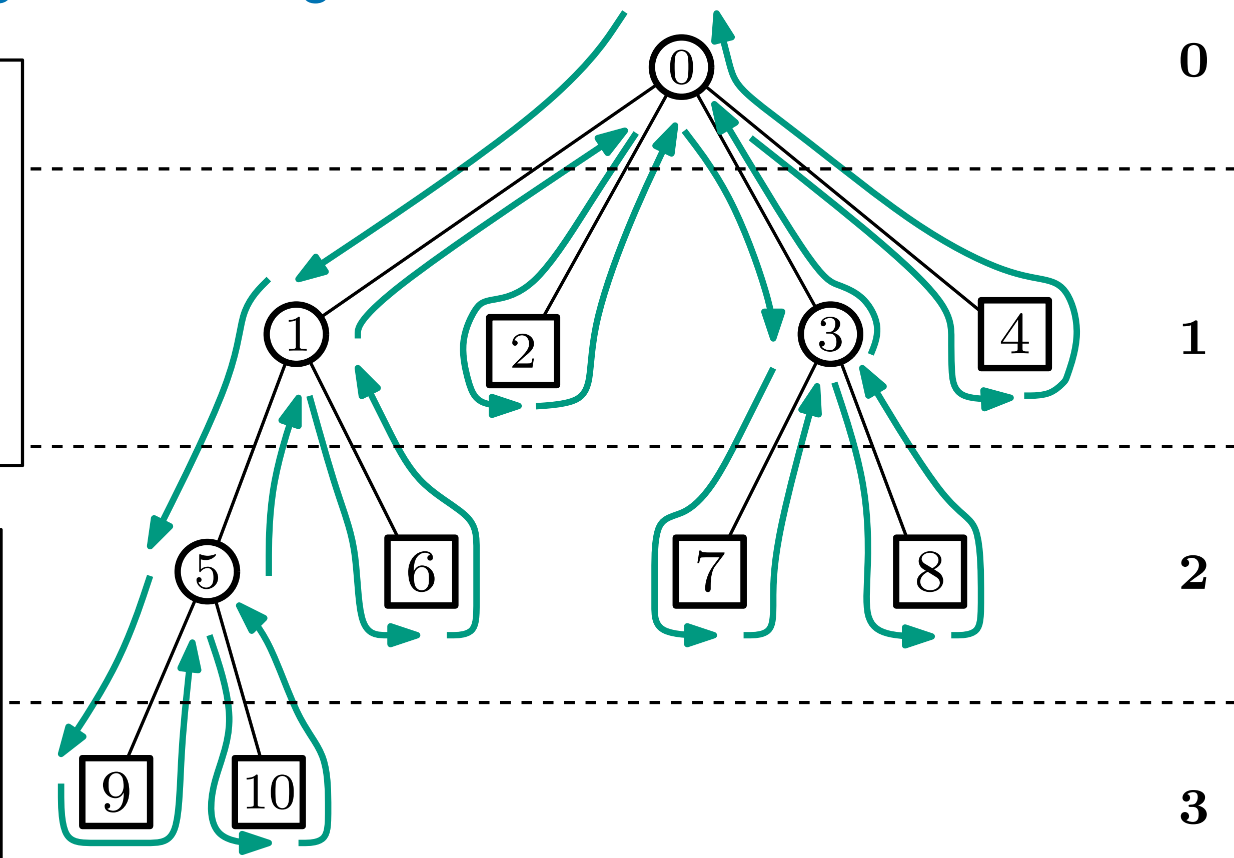
Query Summary - LCA(i,j)

$O(1)$ 1. Find (any) i' st. $N[i'] = i$

$O(1)$ 2. Find (any) j' st. $N[j'] = j$

$O(?)$ 3. Compute RMQ(i', j') in D

$O(1)$ 4. LCA$(i, j) = N[$RMQ$(i', j')]$



(node) N	0	1	5	9	5	10	5	1	6	1	0	2	0	3	7	3	8	3	0	4	0
(depth) D	**0**	**1**	**2**	**3**	**2**	**3**	**2**	**1**	**2**	**1**	**0**	**1**	**0**	**1**	**2**	**1**	**2**	**1**	**0**	**1**	**0**

$2n-1$

Solving LCAs using RMQs

Preprocessing Summary

$O(n)$ 1. Construct N and D from T

$O(n)$ 2. Add a pointer from each node i to some $N[i'] = i$

$O(?)$ 3. Preprocess D for RMQs

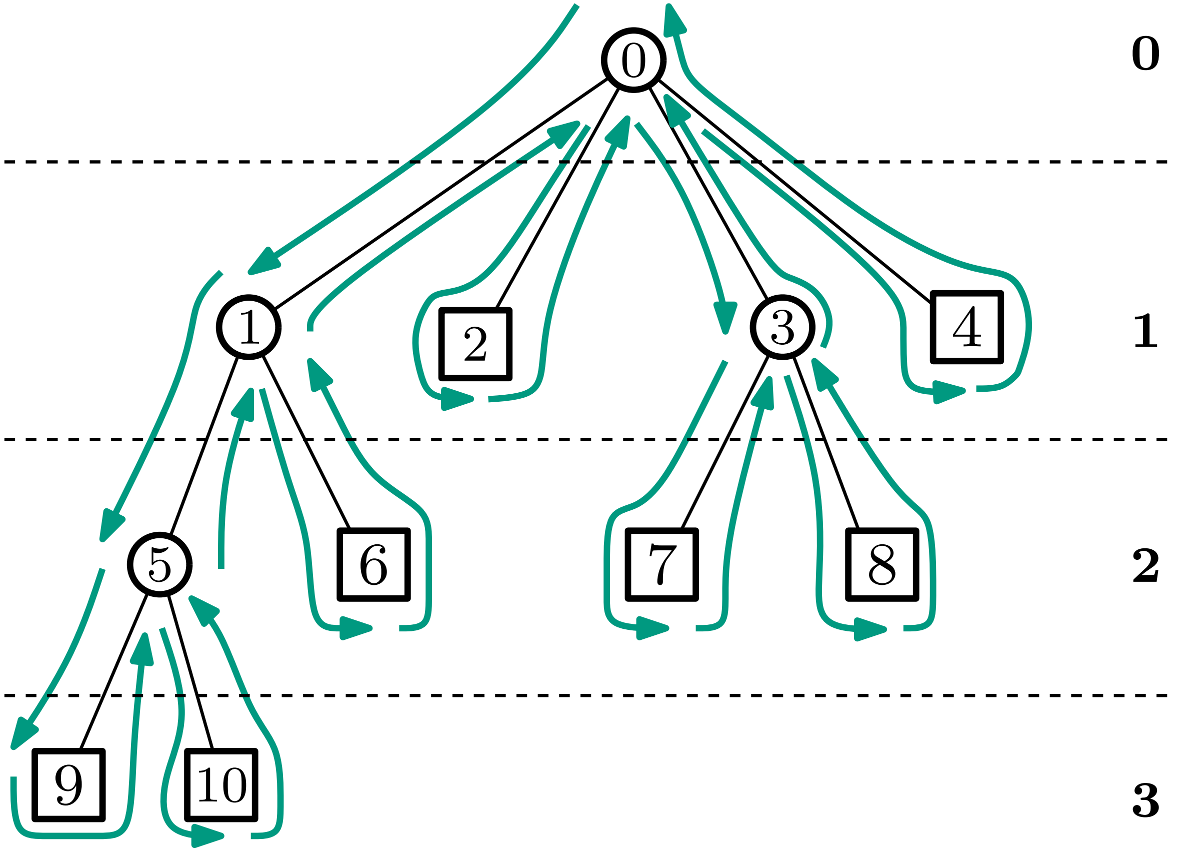
Query Summary - LCA(i,j)

$O(1)$ 1. Find (any) i' st. $N[i'] = i$

$O(1)$ 2. Find (any) j' st. $N[j'] = j$

$O(?)$ 3. Compute RMQ(i', j') in D

$O(1)$ 4. LCA$(i, j) = N[$RMQ$(i', j')]$



(node) N	0	1	5	9	5	10	5	1	6	1	0	2	0	3	7	3	8	3	0	4	0
(depth) D	**0**	**1**	**2**	**3**	**2**	**3**	**2**	**1**	**2**	**1**	**0**	**1**	**0**	**1**	**2**	**1**	**2**	**1**	**0**	**1**	**0**

$2n-1$

Prep. time $O(n + \text{prepRMQ}(n))$

Space $O(n + \text{spaceRMQ}(n))$

Query time $O(1 + \text{queryRMQ}(n))$

depends on the RMQ structure used

Solving LCAs using RMQs

Preprocessing Summary

$O(n)$ 1. Construct N and D from T

$O(n)$ 2. Add a pointer from each node i to some $N[i'] = i$

$O(?)$ 3. Preprocess D for RMQs

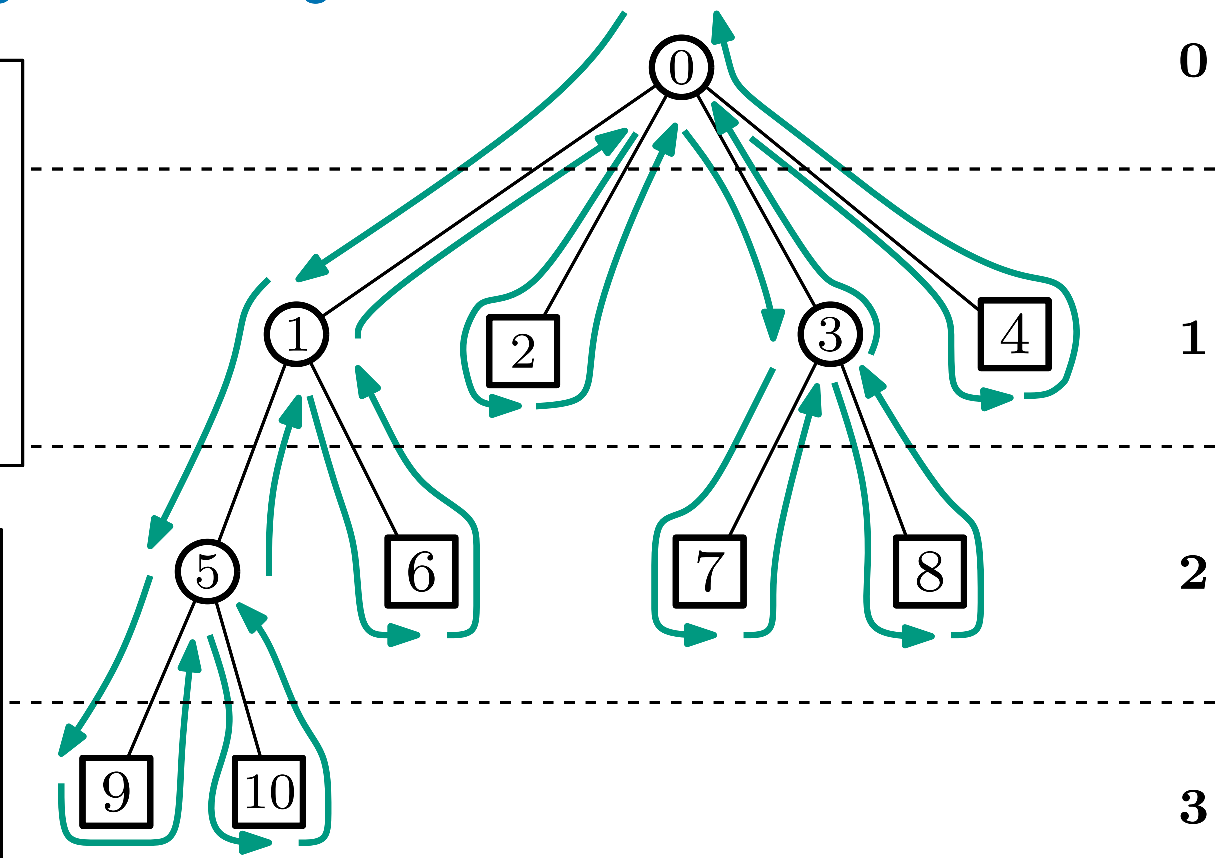
Query Summary - LCA(i,j)

$O(1)$ 1. Find (any) i' st. $N[i'] = i$

$O(1)$ 2. Find (any) j' st. $N[j'] = j$

$O(?)$ 3. Compute RMQ(i', j') in D

$O(1)$ 4. LCA$(i, j) = N[$RMQ$(i', j')]$



(node) N	0	1	5	9	5	10	5	1	6	1	0	2	0	3	7	3	8	3	0	4	0
(depth) D	**0**	**1**	**2**	**3**	**2**	**3**	**2**	**1**	**2**	**1**	**0**	**1**	**0**	**1**	**2**	**1**	**2**	**1**	**0**	**1**	**0**

$\underbrace{\qquad\qquad}_{2n-1}$

Prep. time $O(n \log\log n)$ Space $O(n \log\log n)$

Query time $O(1)$

using the best result from last lecture

Solving LCAs using RMQs

Preprocessing Summary

$O(n)$ 1. Construct N and D from T

$O(n)$ 2. Add a pointer from each node i to some $N[i'] = i$

$O(?)$ 3. Preprocess D for RMQs

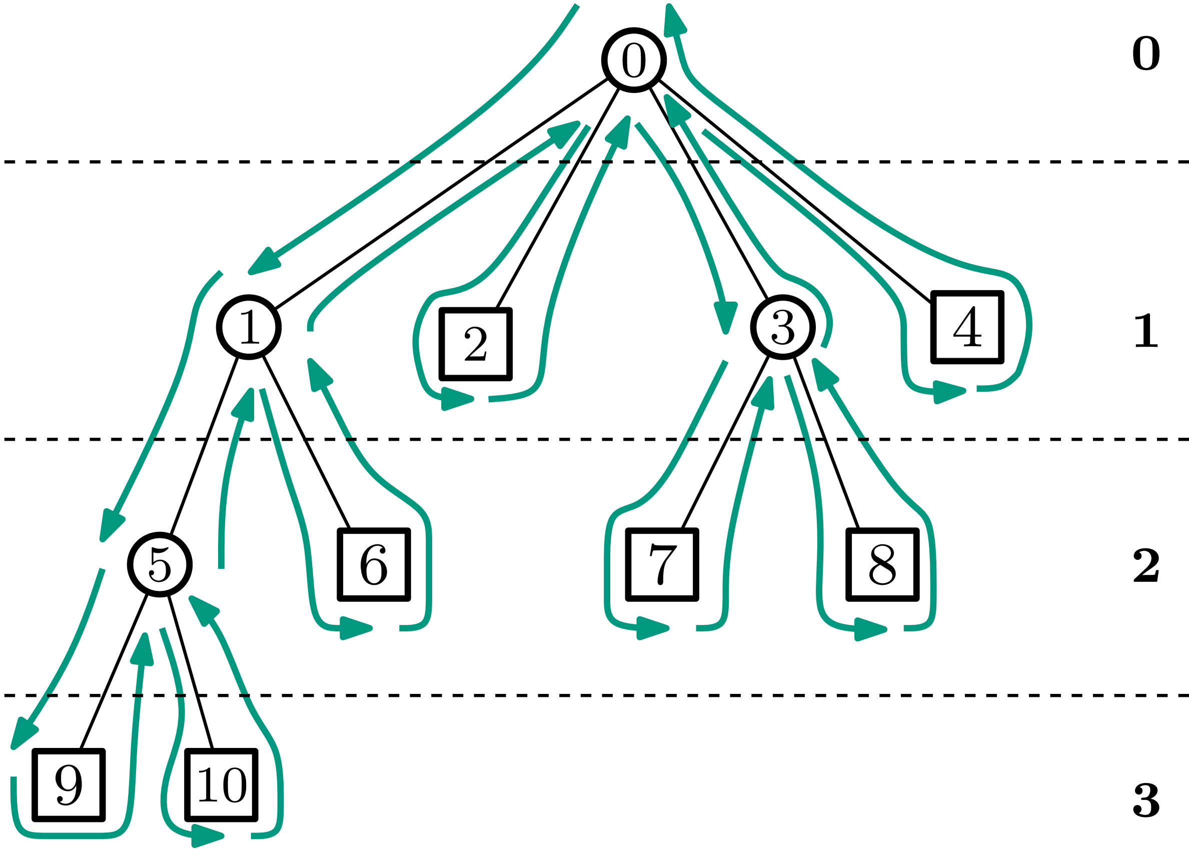
Query Summary - LCA(i,j)

$O(1)$ 1. Find (any) i' st. $N[i'] = i$

$O(1)$ 2. Find (any) j' st. $N[j'] = j$

$O(?)$ 3. Compute RMQ(i', j') in D

$O(1)$ 4. LCA$(i, j) = N[$RMQ$(i', j')]$



why does this work?

(node) N	0	1	5	9	5	10	5	1	6	1	0	2	0	3	7	3	8	3	0	4	0
(depth) D	**0**	**1**	**2**	**3**	**2**	**3**	**2**	**1**	**2**	**1**	**0**	**1**	**0**	**1**	**2**	**1**	**2**	**1**	**0**	**1**	**0**

$2n-1$

Prep. time $O(n \log\log n)$

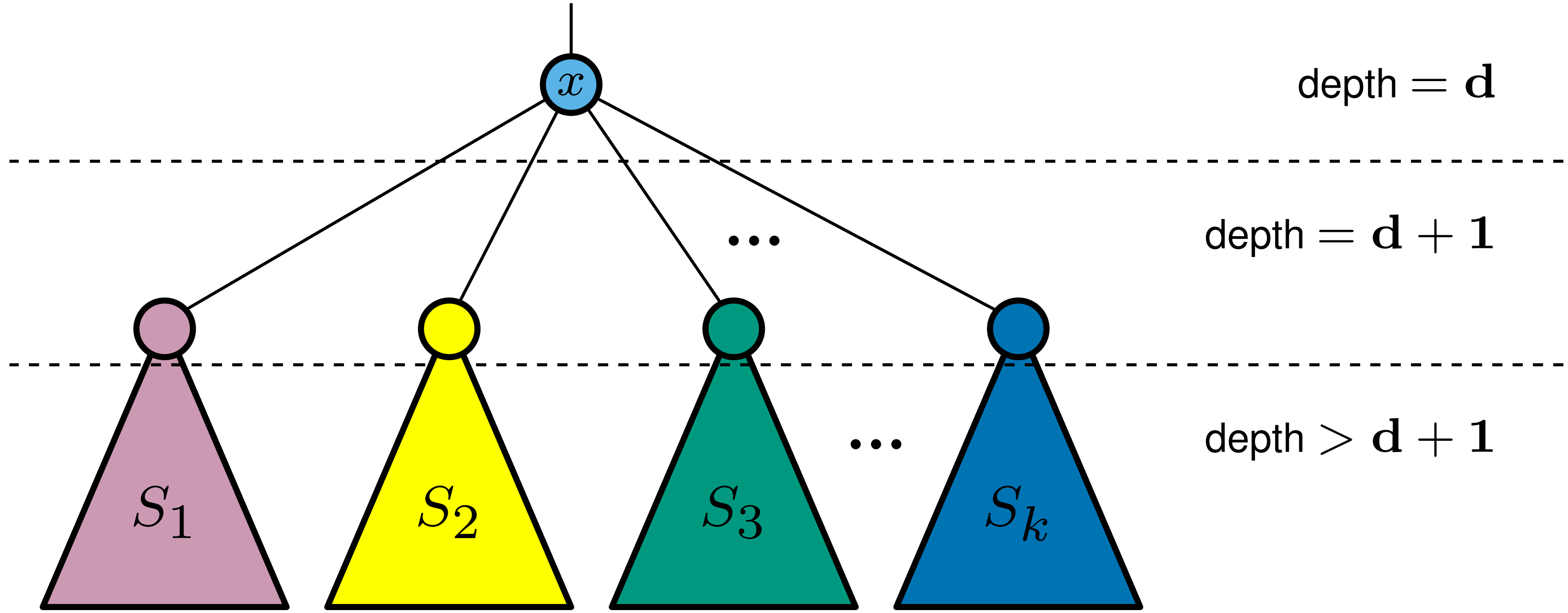
Space $O(n \log\log n)$

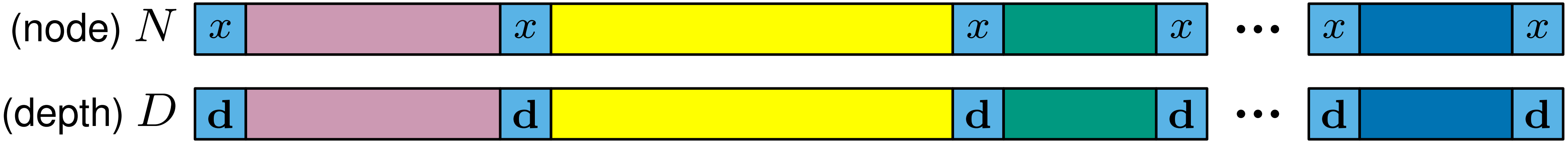
Query time $O(1)$

using the best result from last lecture

Solving LCA using RMQ - correctness

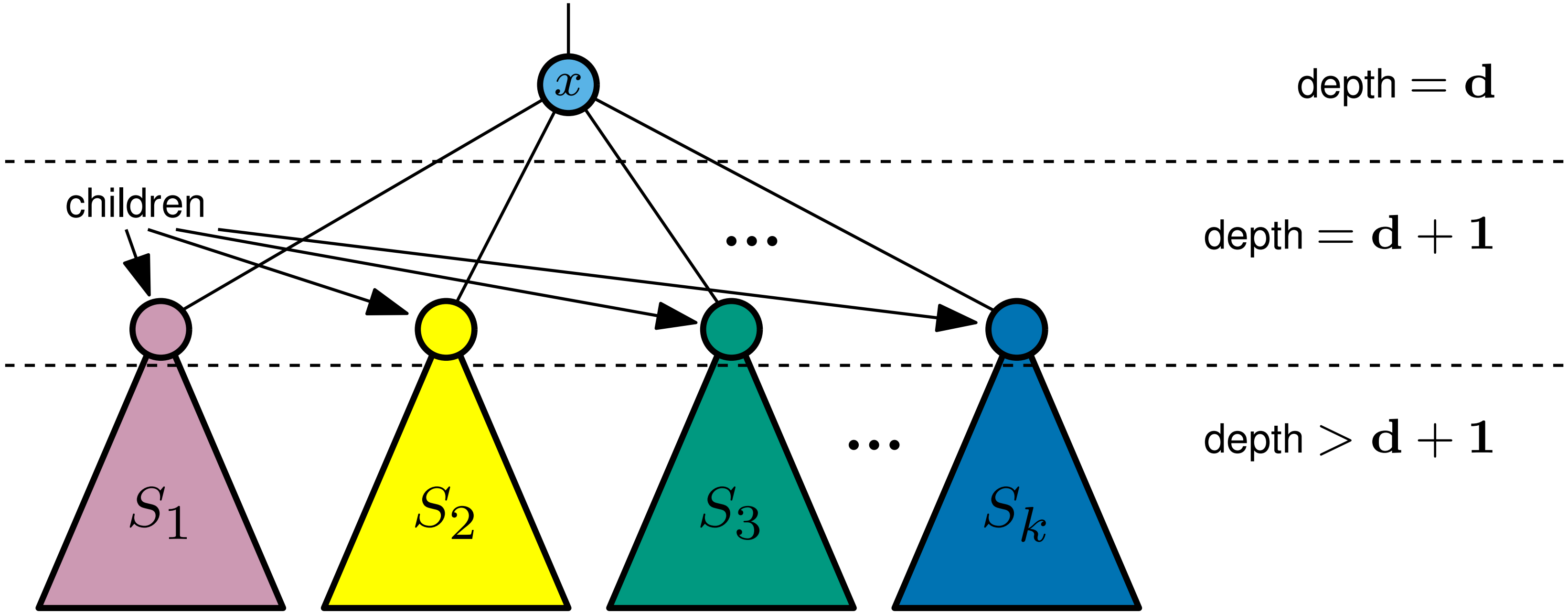
We can also define a Euler tour of T recursively. . .

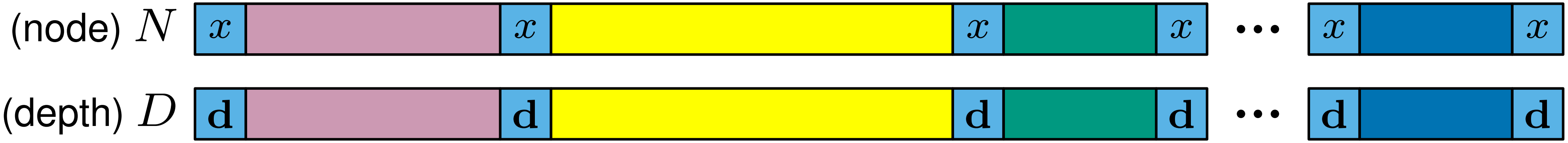




Solving LCA using RMQ - correctness

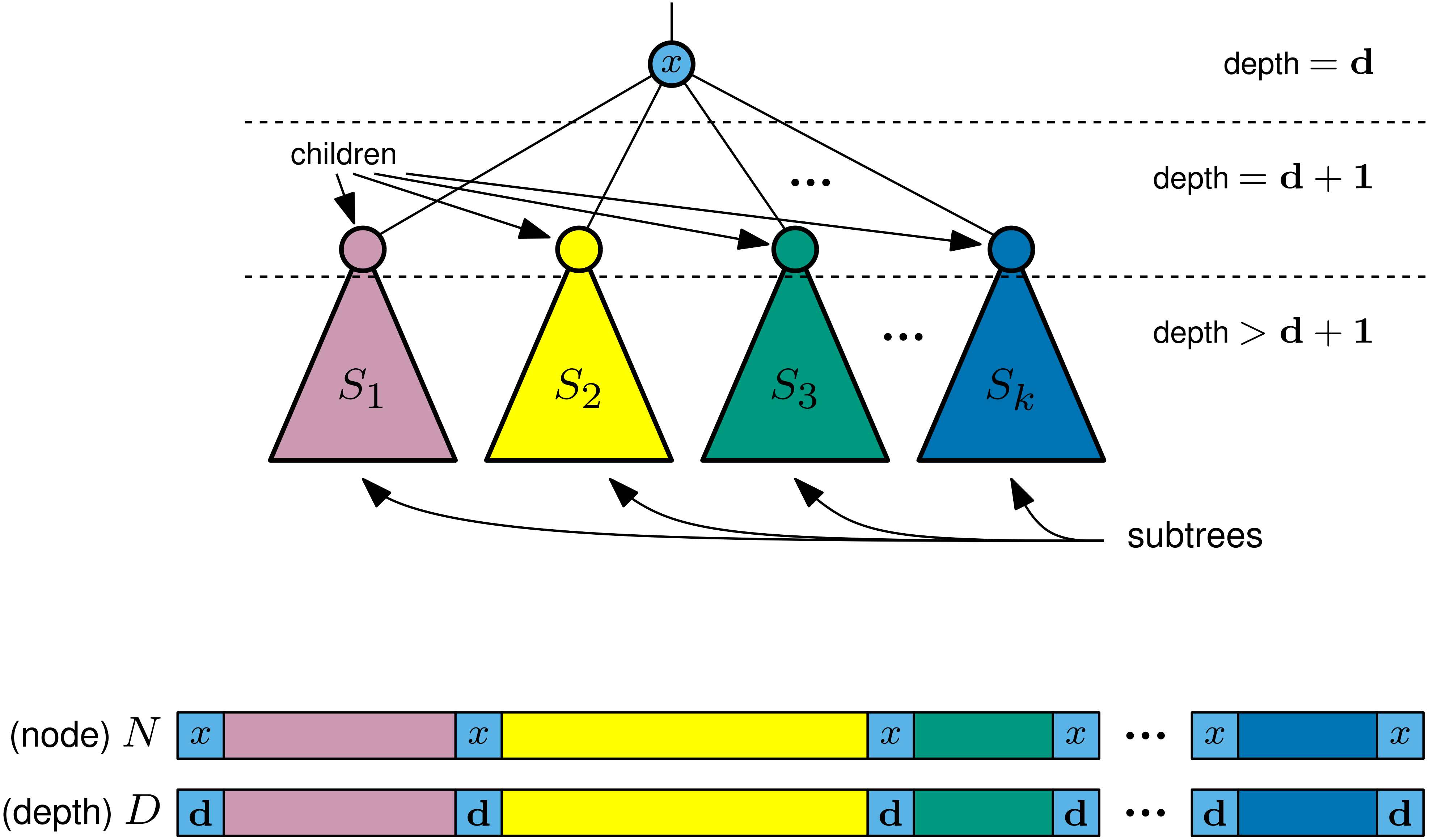
We can also define a Euler tour of T recursively. . .




(node) N
x x x x ... x x
(depth) D
$\mathbf{d}$ $\mathbf{d}$ $\mathbf{d}$ $\mathbf{d}$ ... $\mathbf{d}$ $\mathbf{d}$

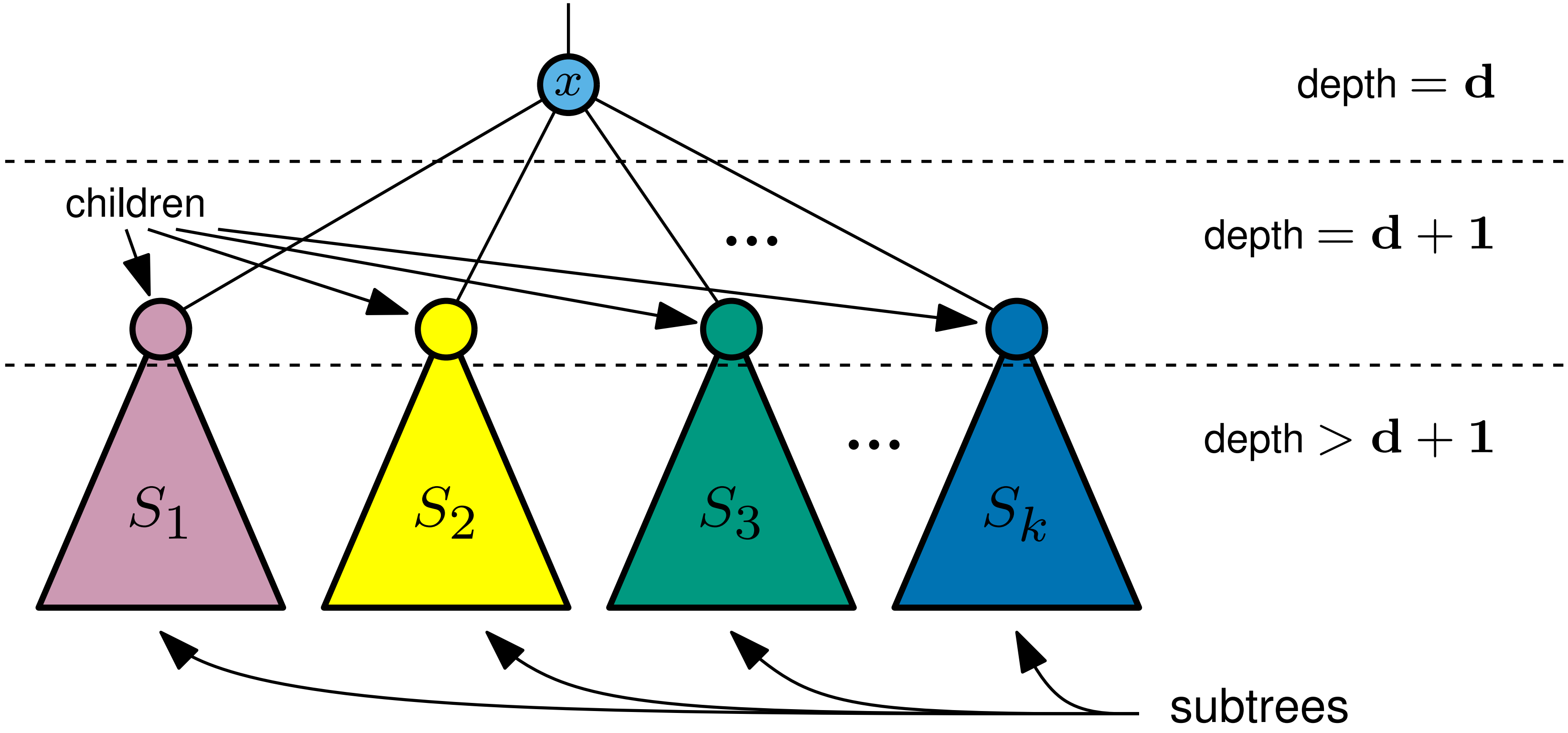
Solving LCA using RMQ - correctness

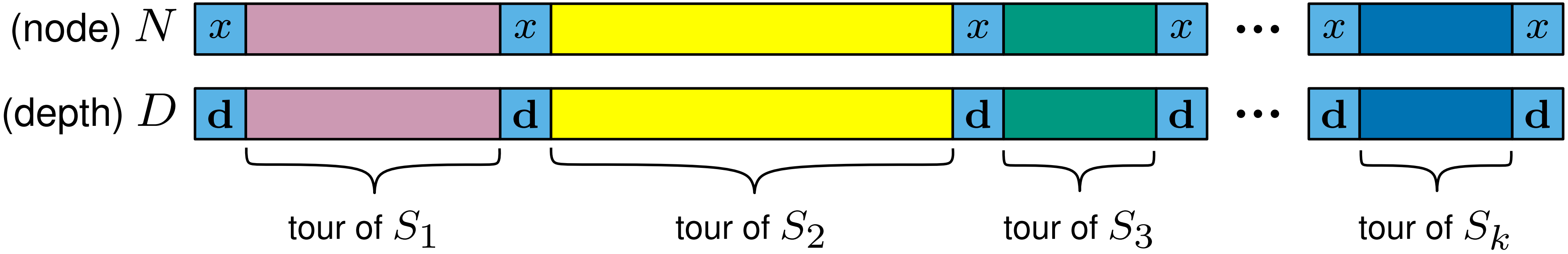
We can also define a Euler tour of T recursively. . .


x
depth $= \mathbf{d}$
children
$\cdots$
depth $= \mathbf{d}+\mathbf{1}$
S_1
S_2
S_3
$\cdots$
S_k
depth $> \mathbf{d}+\mathbf{1}$
subtrees
(node) N
x x x x $\cdots$ x x
(depth) D
$\mathbf{d}$ $\mathbf{d}$ $\mathbf{d}$ $\mathbf{d}$ $\cdots$ $\mathbf{d}$ $\mathbf{d}$

Solving LCA using RMQ - correctness

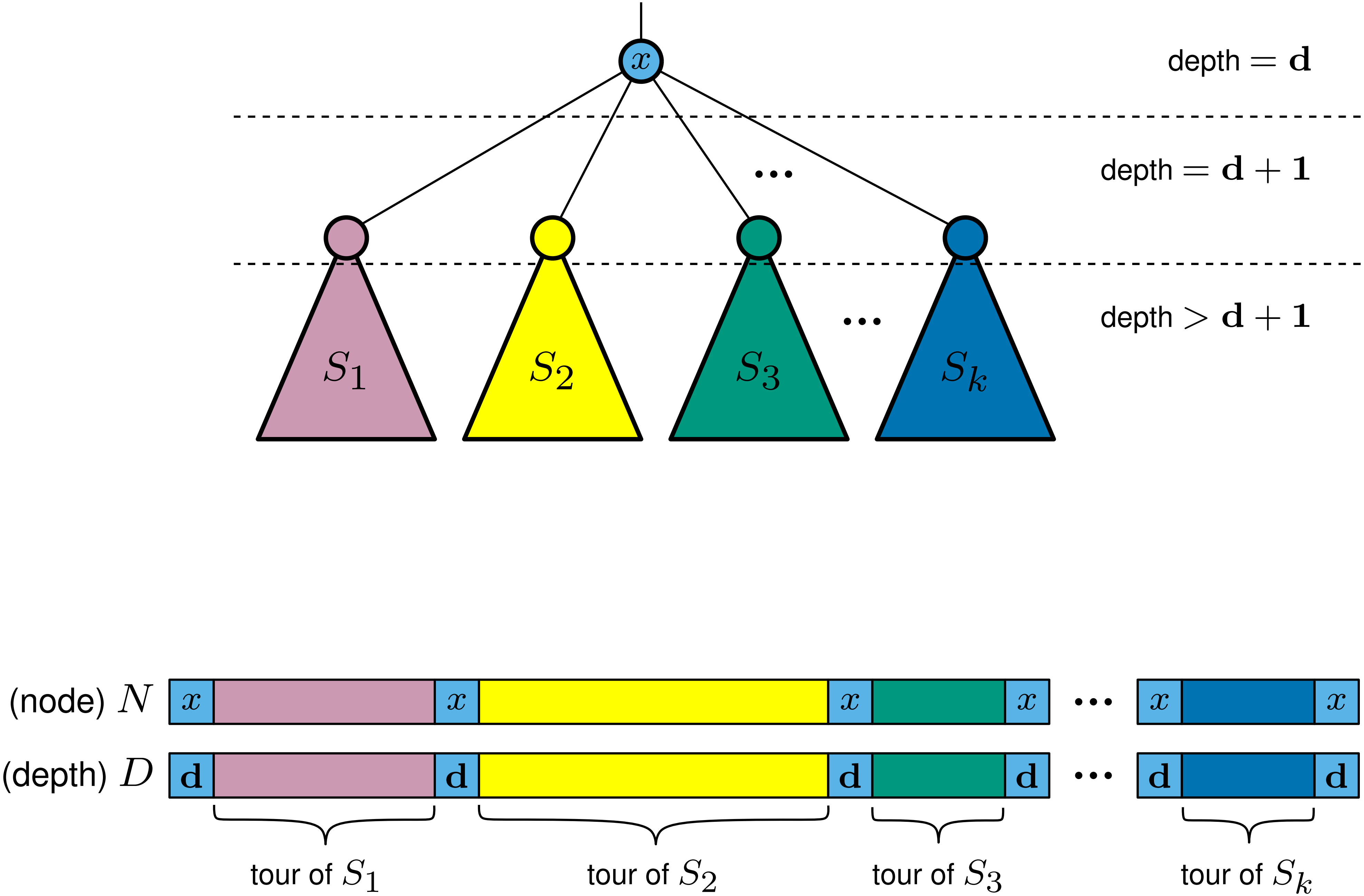
We can also define a Euler tour of T recursively...





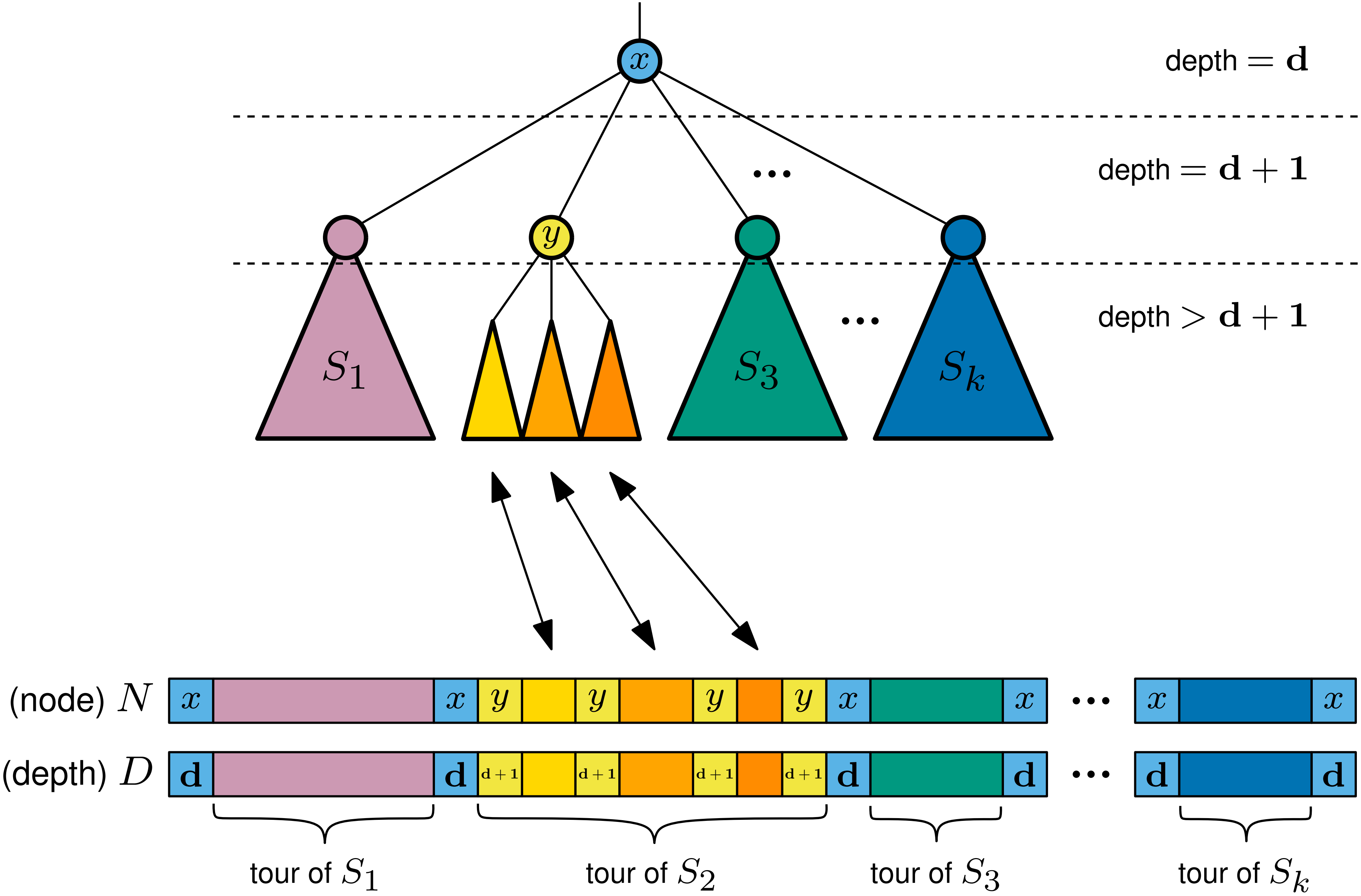
Solving LCA using RMQ - correctness

We can also define a Euler tour of T recursively...



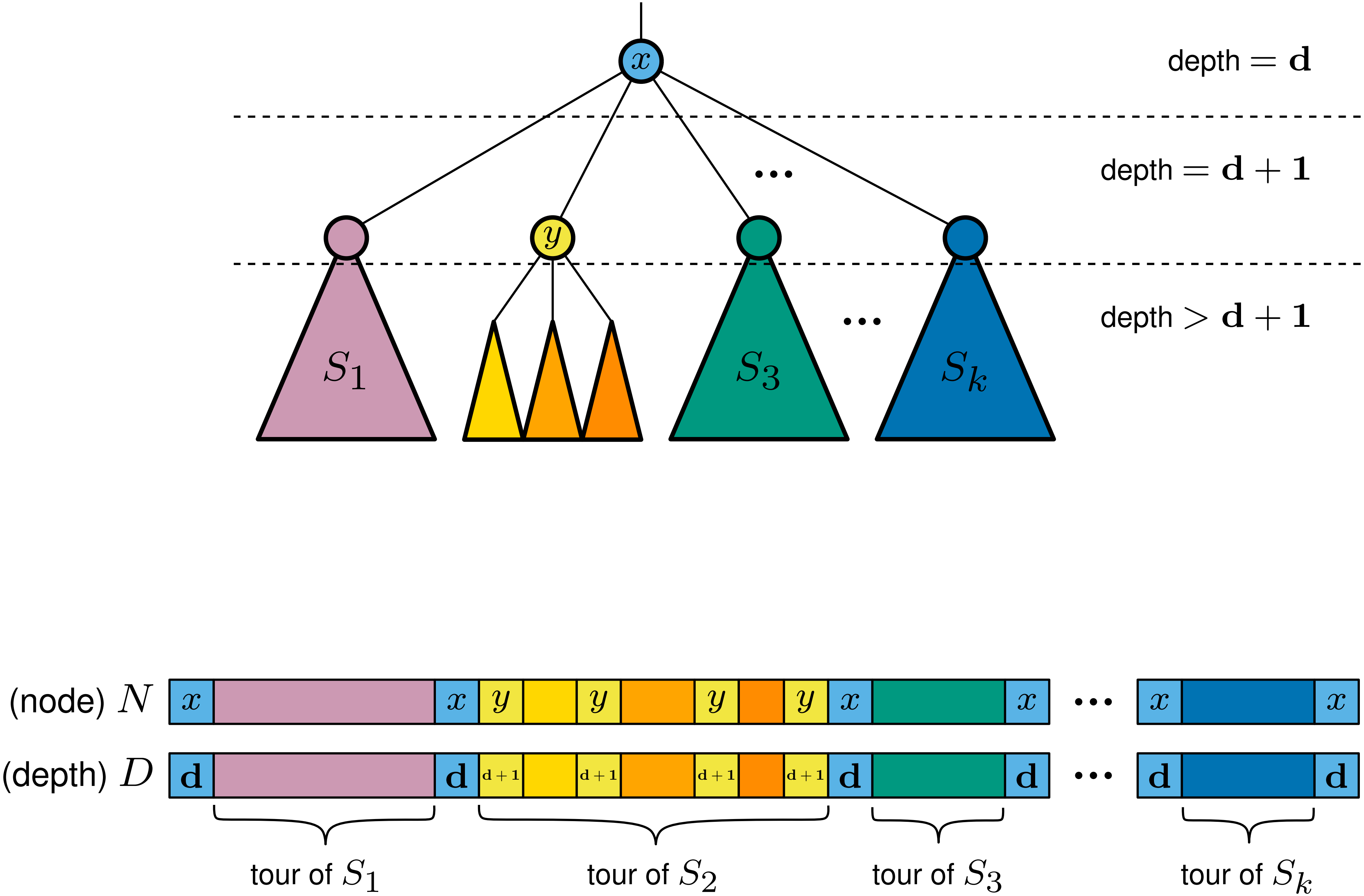
Solving LCA using RMQ - correctness

We can also define a Euler tour of T recursively...


x
depth $= \mathbf{d}$
depth $= \mathbf{d+1}$
y
...
depth $> \mathbf{d+1}$
S_1
S_3
...
S_k
(node) N
x x y y y y x x ... x x
(depth) D
$\mathbf{d}$ $\mathbf{d}$ $\mathrm{d+1}$ $\mathrm{d+1}$ $\mathrm{d+1}$ $\mathrm{d+1}$ $\mathbf{d}$ $\mathbf{d}$ ... $\mathbf{d}$ $\mathbf{d}$
tour of S_1
tour of S_2
tour of S_3
tour of S_k

Solving LCA using RMQ - correctness

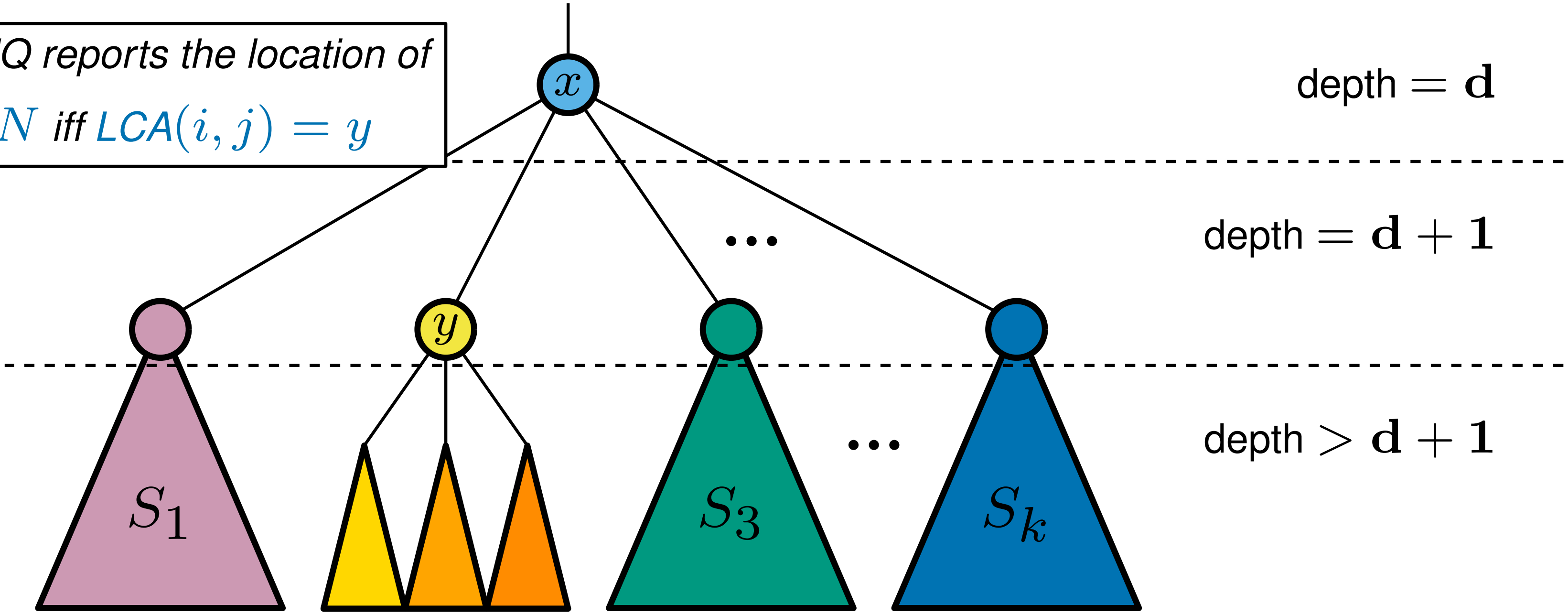
We can also define a Euler tour of T recursively. . .

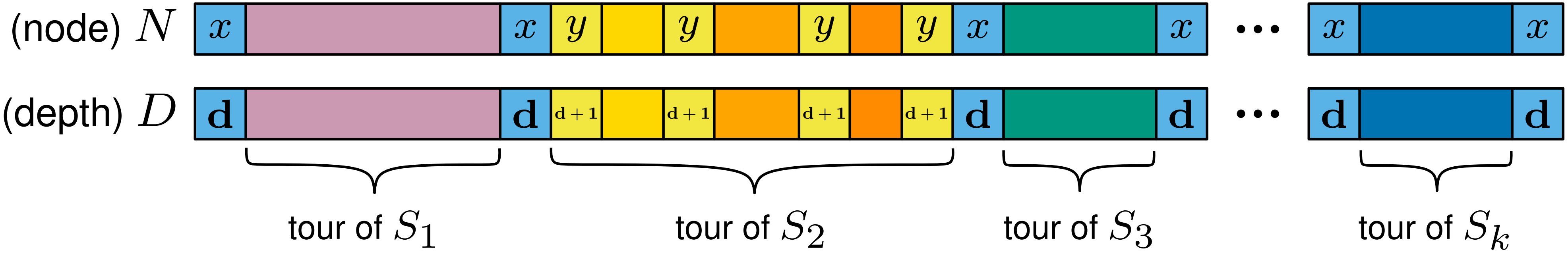

x
depth $= \mathbf{d}$
depth $= \mathbf{d+1}$
depth $> \mathbf{d+1}$
y
$\cdots$
S_1
S_3
$\cdots$
S_k
(node) N
x x y y y y x x $\cdots$ x x
(depth) D
$\mathbf{d}$ $\mathbf{d}$ $\mathrm{d+1}$ $\mathrm{d+1}$ $\mathrm{d+1}$ $\mathrm{d+1}$ $\mathbf{d}$ $\mathbf{d}$ $\cdots$ $\mathbf{d}$ $\mathbf{d}$
tour of S_1
tour of S_2
tour of S_3
tour of S_k

Solving LCA using RMQ - correctness

We can also define a Euler tour of T recursively...

Claim *the RMQ reports the location of some y in N iff LCA$(i,j) = y$*

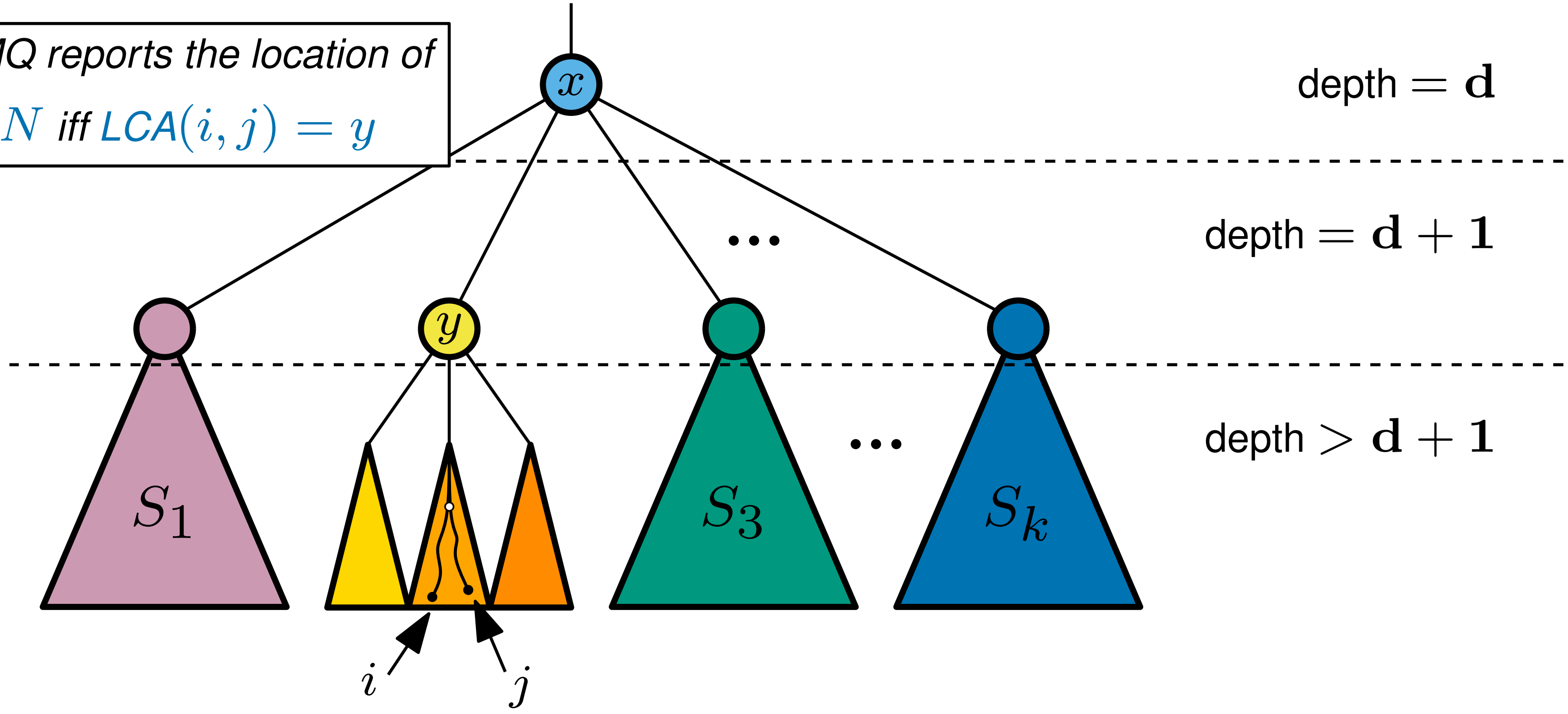




Solving LCA using RMQ - correctness

We can also define a Euler tour of T recursively. . .

Claim *the RMQ reports the location of some* y *in* N *iff* LCA$(i, j) = y$



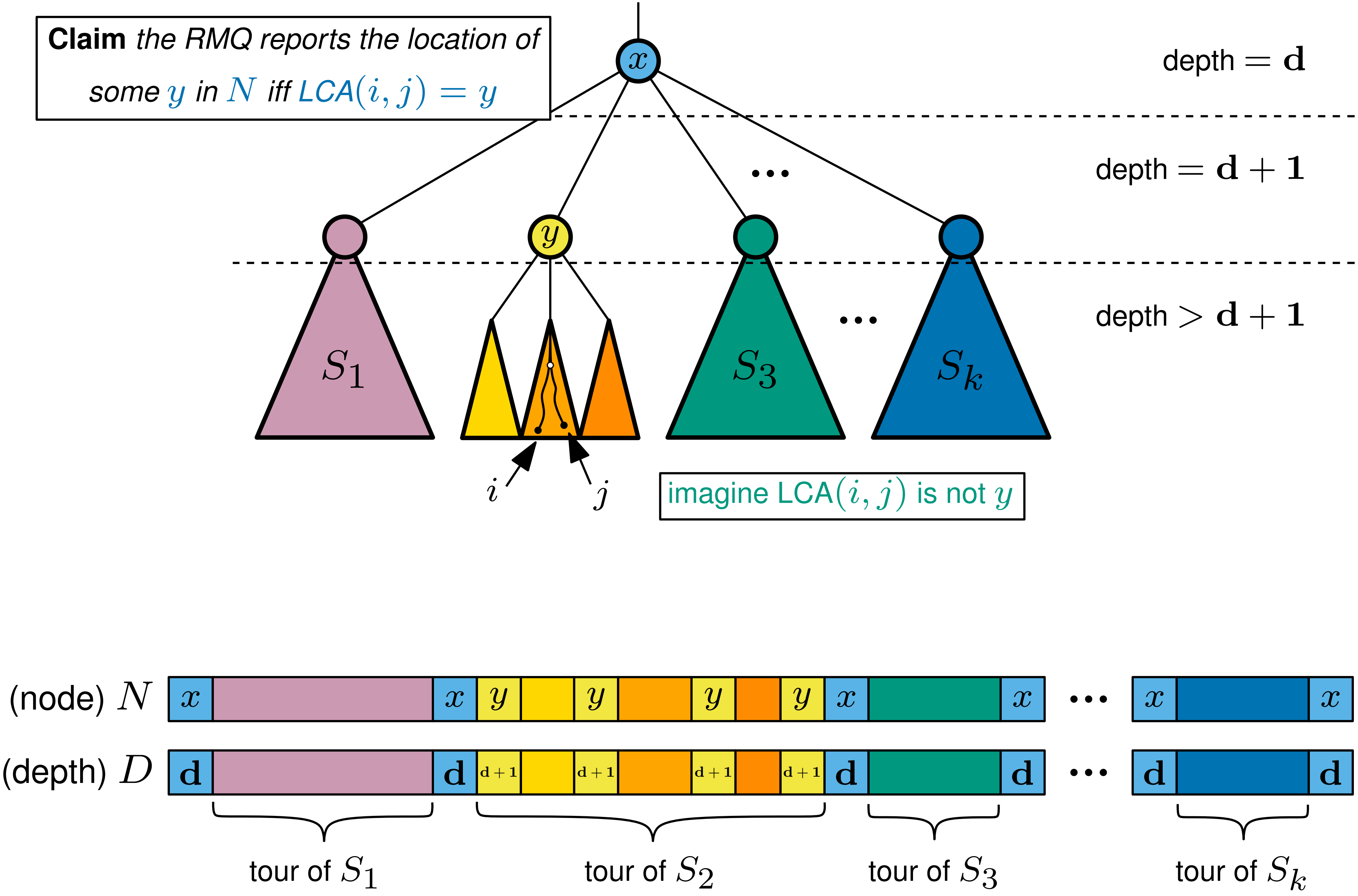
(node) N	x		x	y		y		y		y	x		x	$\cdots$	x		x
(depth) D	$\mathbf{d}$		$\mathbf{d}$	$\mathrm{d+1}$		$\mathrm{d+1}$		$\mathrm{d+1}$		$\mathrm{d+1}$	$\mathbf{d}$		$\mathbf{d}$	$\cdots$	$\mathbf{d}$		$\mathbf{d}$

tour of S_1 tour of S_2 tour of S_3 tour of S_k

Solving LCA using RMQ - correctness

We can also define a Euler tour of T recursively. . .

Claim *the RMQ reports the location of some y in N iff LCA$(i, j) = y$*

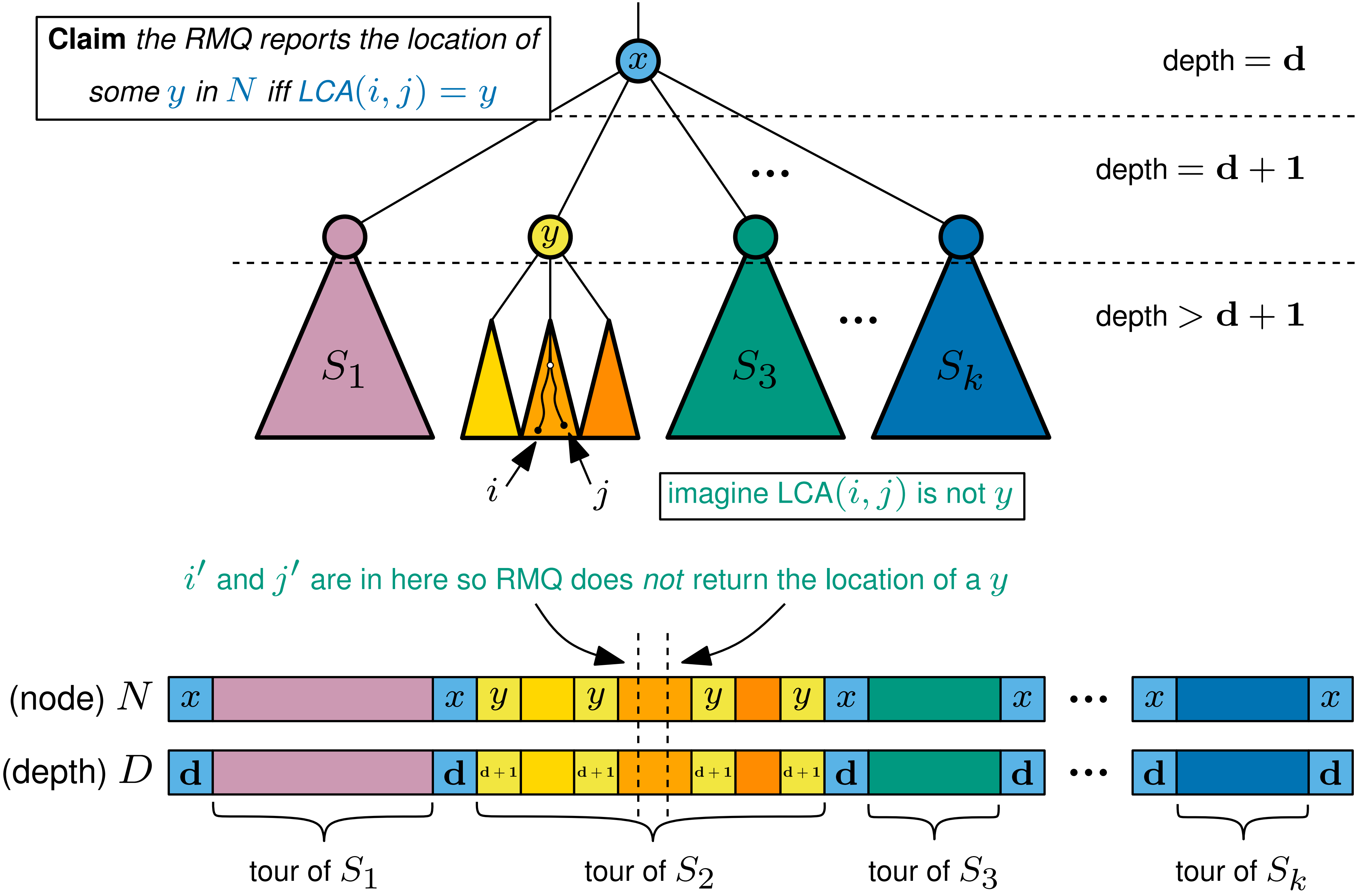


imagine LCA(i, j) is not y

Solving LCA using RMQ - correctness

We can also define a Euler tour of T recursively...

Claim *the RMQ reports the location of some y in N iff LCA$(i, j) = y$*


x
depth $= \mathbf{d}$
depth $= \mathbf{d+1}$
depth $> \mathbf{d+1}$
y
...
S_1
S_3
...
S_k
i
j
(node) N
x x y y y y x x ... x x
(depth) D
$\mathbf{d}$ $\mathbf{d}$ $\mathrm{d+1}$ $\mathrm{d+1}$ $\mathrm{d+1}$ $\mathrm{d+1}$ $\mathbf{d}$ $\mathbf{d}$ ... $\mathbf{d}$ $\mathbf{d}$
tour of S_1
tour of S_2
tour of S_3
tour of S_k

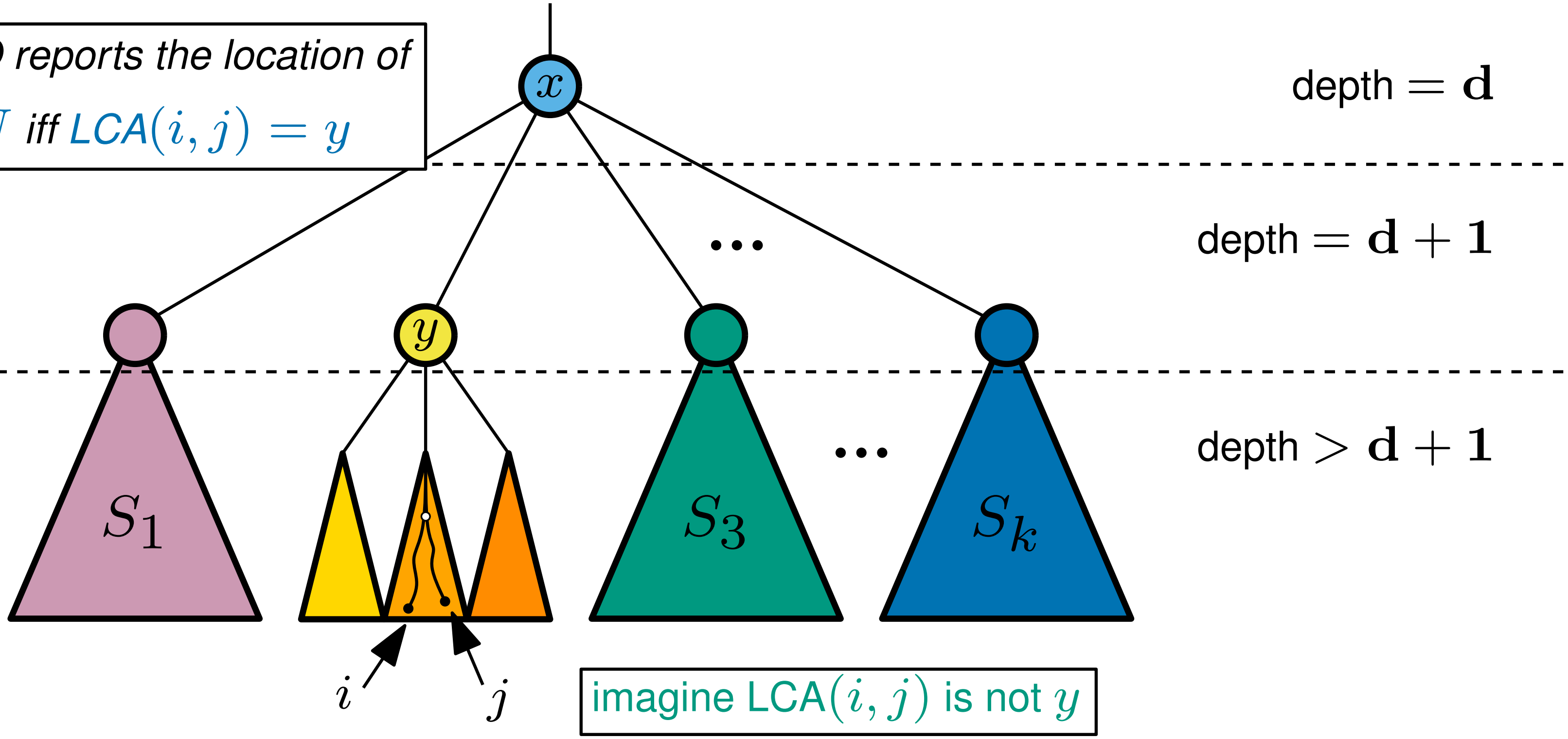
imagine LCA(i, j) is not y

i' and j' are in here so RMQ does *not* return the location of a y

Solving LCA using RMQ - correctness

We can also define a Euler tour of T recursively...

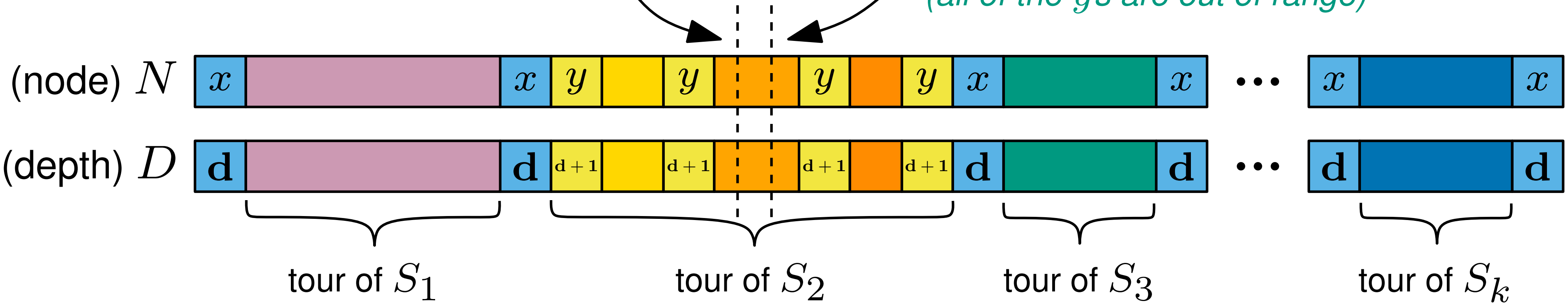
Claim *the RMQ reports the location of some y in N iff LCA$(i, j) = y$*



imagine LCA(i, j) is not y

i' and j' are in here so RMQ does *not* return the location of a y

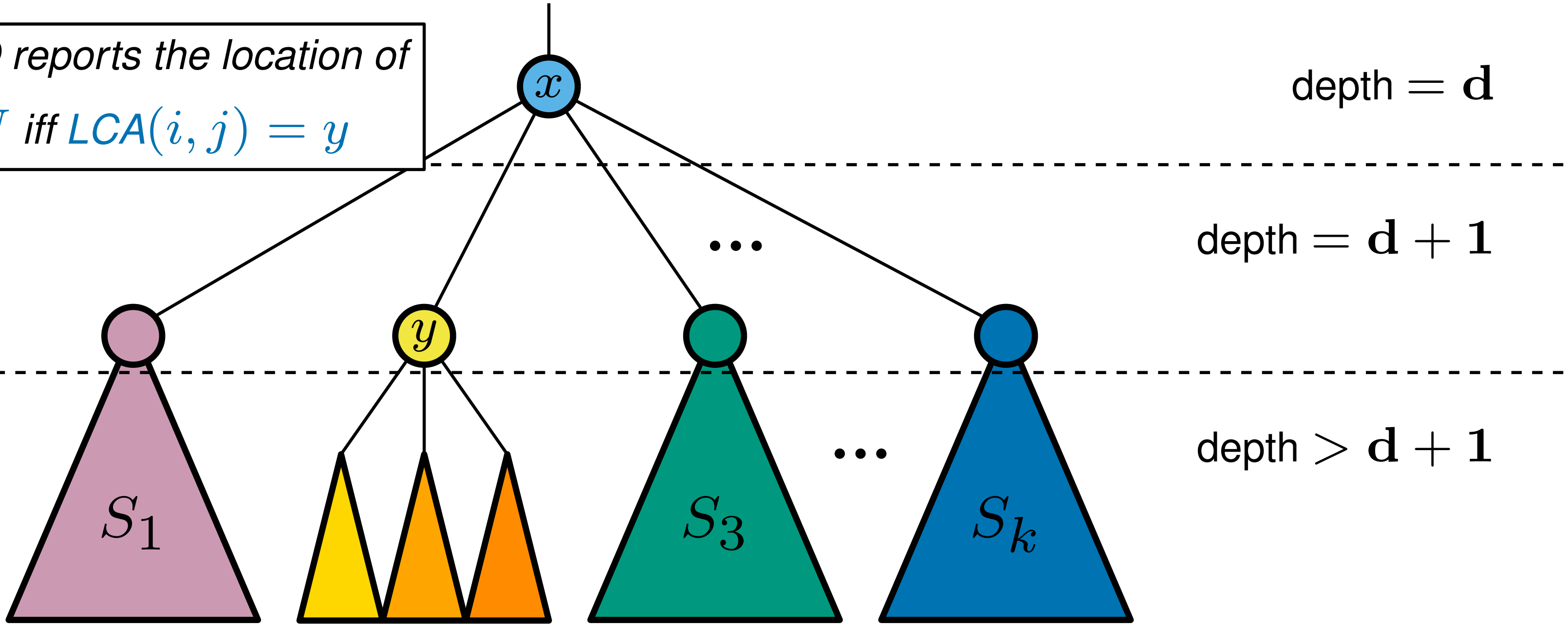
(all of the ys are out of range)

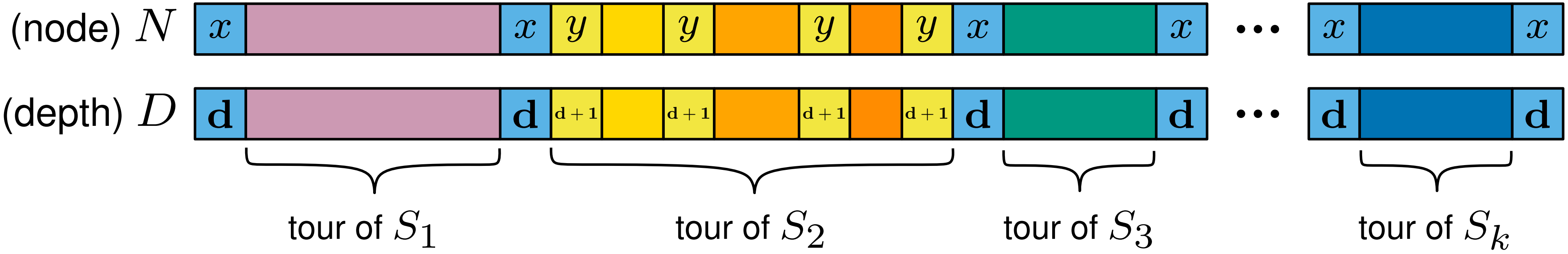


Solving LCA using RMQ - correctness

We can also define a Euler tour of T recursively...

Claim *the RMQ reports the location of some y in N iff LCA$(i, j) = y$*

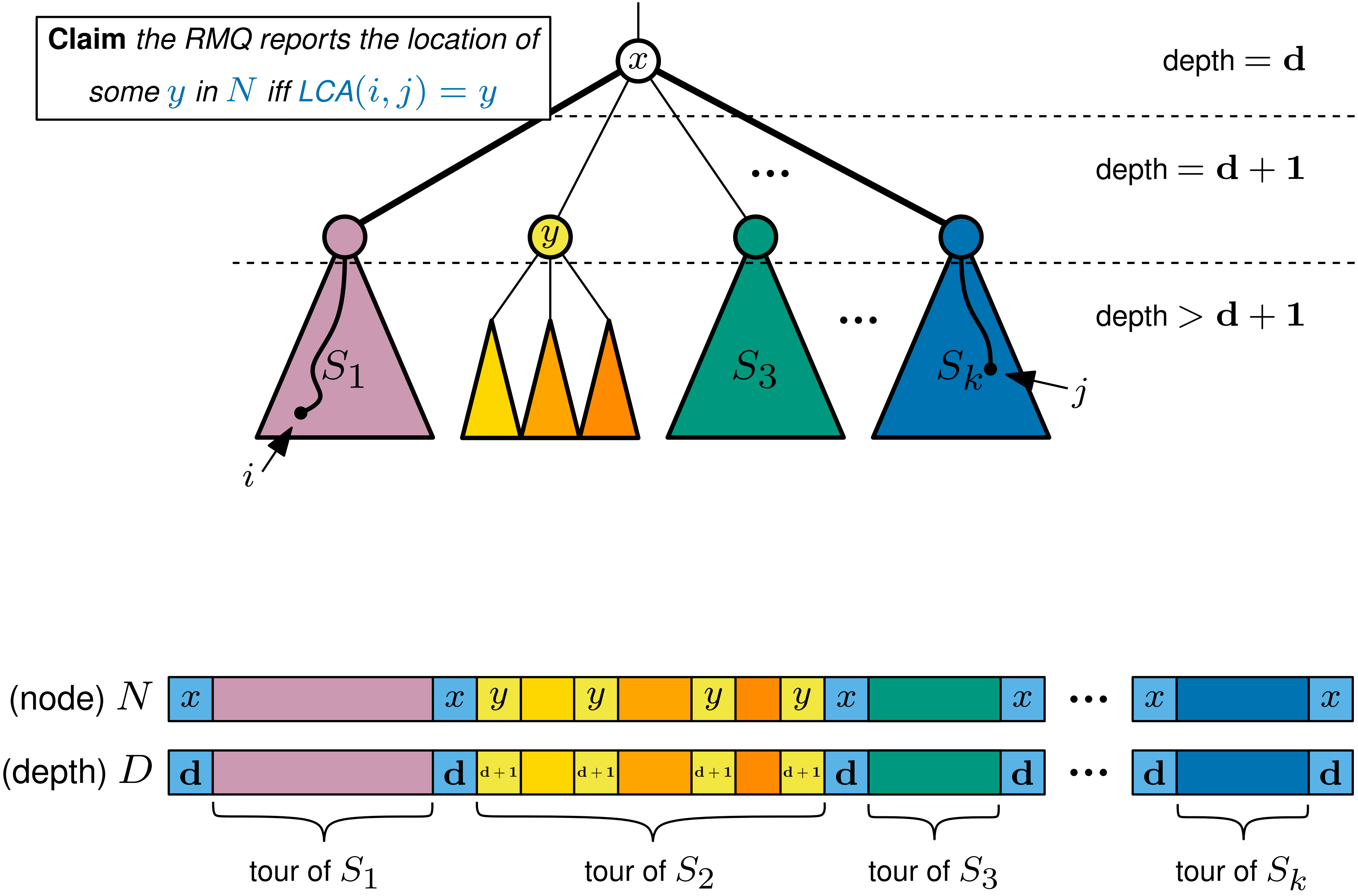




Solving LCA using RMQ - correctness

We can also define a Euler tour of T recursively...

Claim *the RMQ reports the location of some y in N iff LCA$(i, j) = y$*

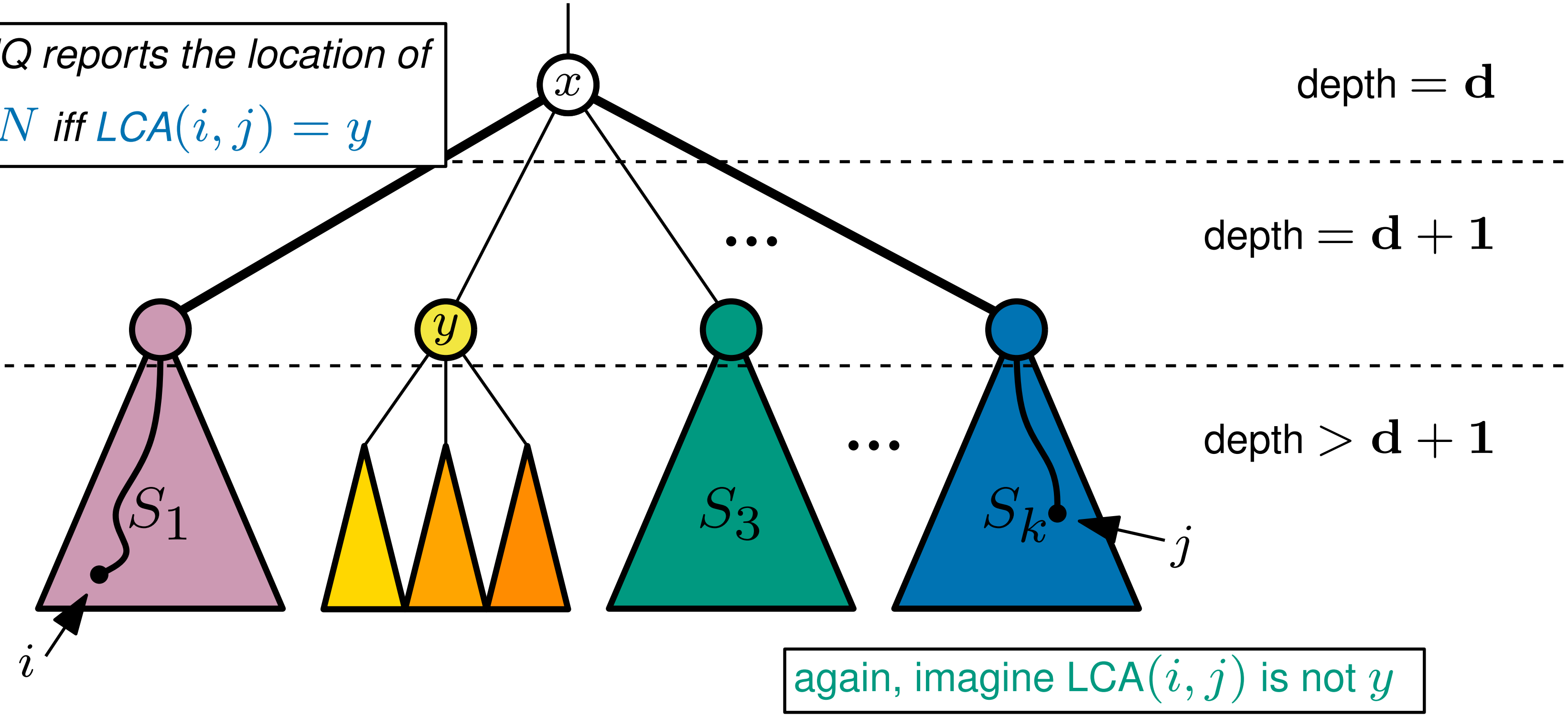


x
y
depth $= \mathbf{d}$
depth $= \mathbf{d+1}$
depth $> \mathbf{d+1}$
S_1
S_3
S_k
i
j
...
(node) N
x x y y y y x x ... x x
(depth) D
$\mathbf{d}$ $\mathbf{d}$ $\mathrm{d+1}$ $\mathrm{d+1}$ $\mathrm{d+1}$ $\mathrm{d+1}$ $\mathbf{d}$ $\mathbf{d}$ ... $\mathbf{d}$ $\mathbf{d}$
tour of S_1
tour of S_2
tour of S_3
tour of S_k

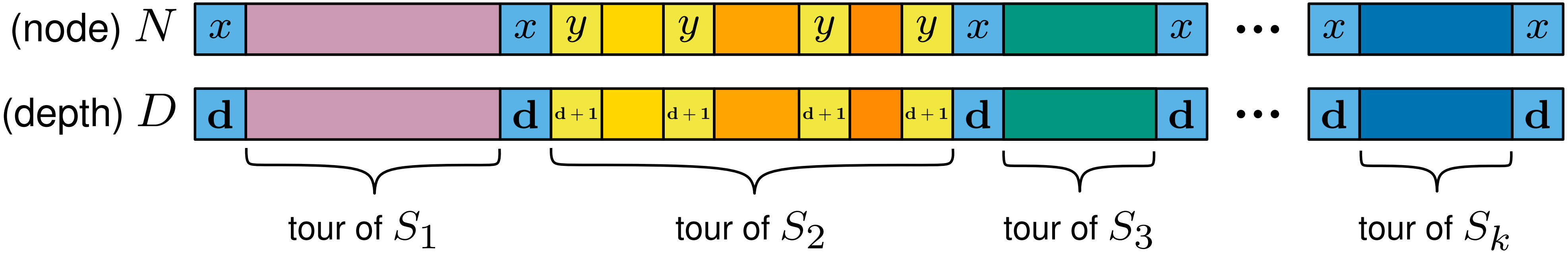
Solving LCA using RMQ - correctness

We can also define a Euler tour of T recursively. . .

Claim *the RMQ reports the location of some y in N iff LCA$(i,j) = y$*



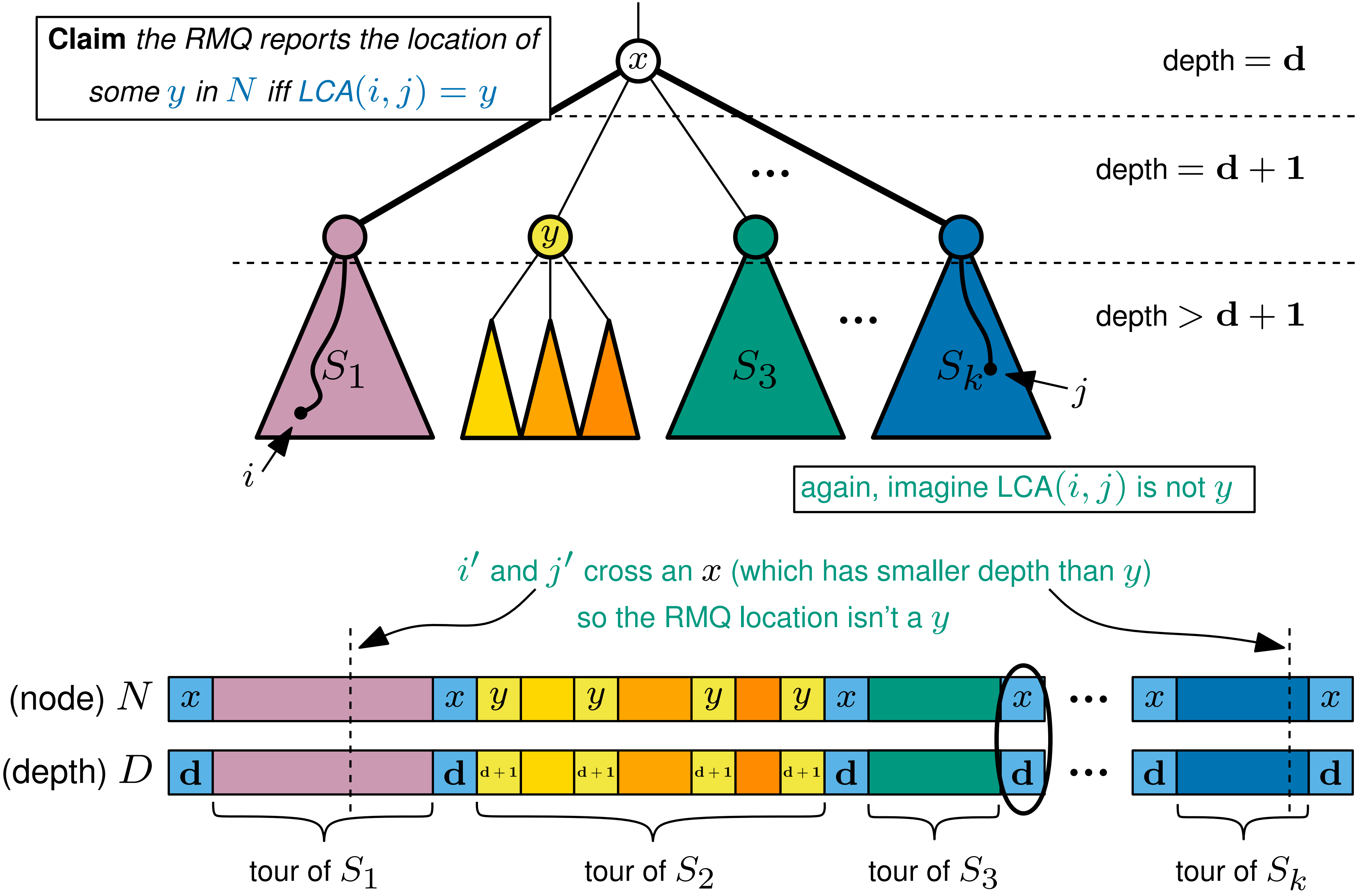
again, imagine LCA(i,j) is not y



Solving LCA using RMQ - correctness

We can also define a Euler tour of T recursively...

Claim *the RMQ reports the location of some y in N iff LCA$(i, j) = y$*



again, imagine LCA(i, j) is not y

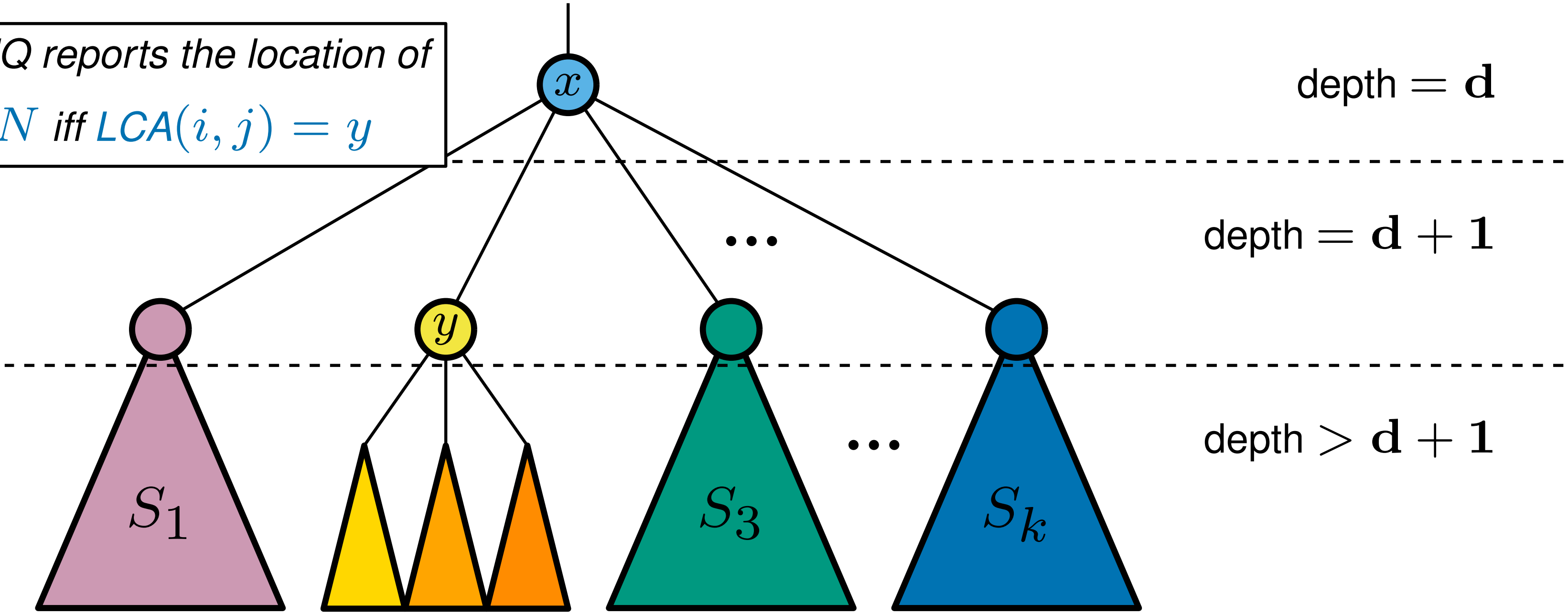
i' and j' cross an x (which has smaller depth than y)

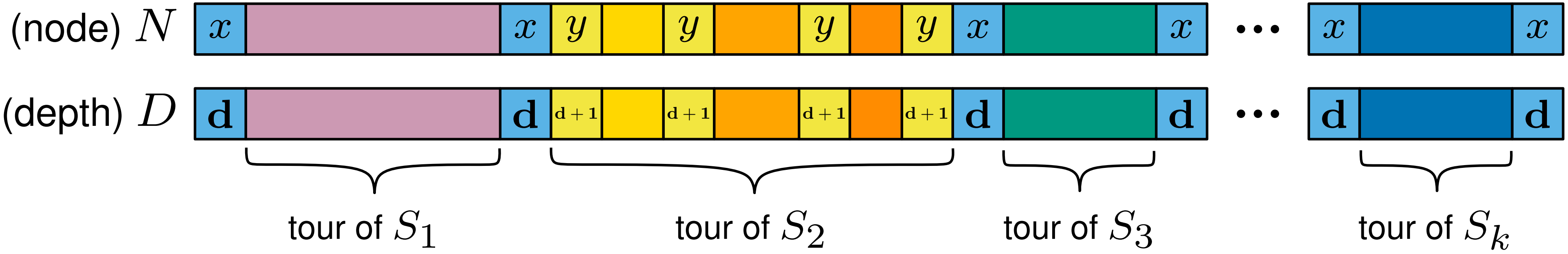
so the RMQ location isn't a y

Solving LCA using RMQ - correctness

We can also define a Euler tour of T recursively...

Claim *the RMQ reports the location of some* y *in* N *iff* LCA$(i,j) = y$



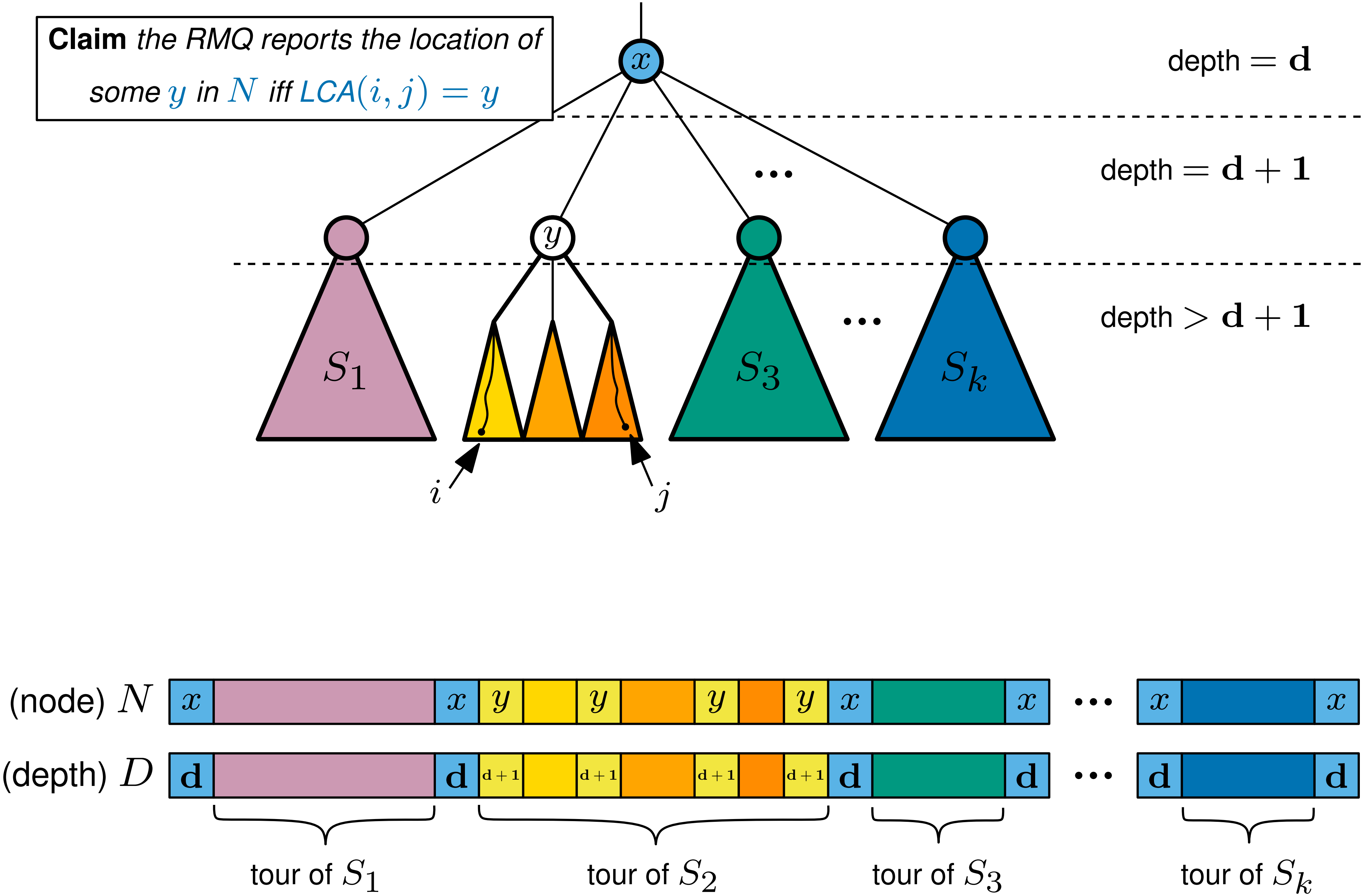


(node) N
x x y y y y x x ... x x
(depth) D
$\mathbf{d}$ $\mathbf{d}$ $\mathrm{d+1}$ $\mathrm{d+1}$ $\mathrm{d+1}$ $\mathrm{d+1}$ $\mathbf{d}$ $\mathbf{d}$ ... $\mathbf{d}$ $\mathbf{d}$
tour of S_1
tour of S_2
tour of S_3
tour of S_k

Solving LCA using RMQ - correctness

We can also define a Euler tour of T recursively...

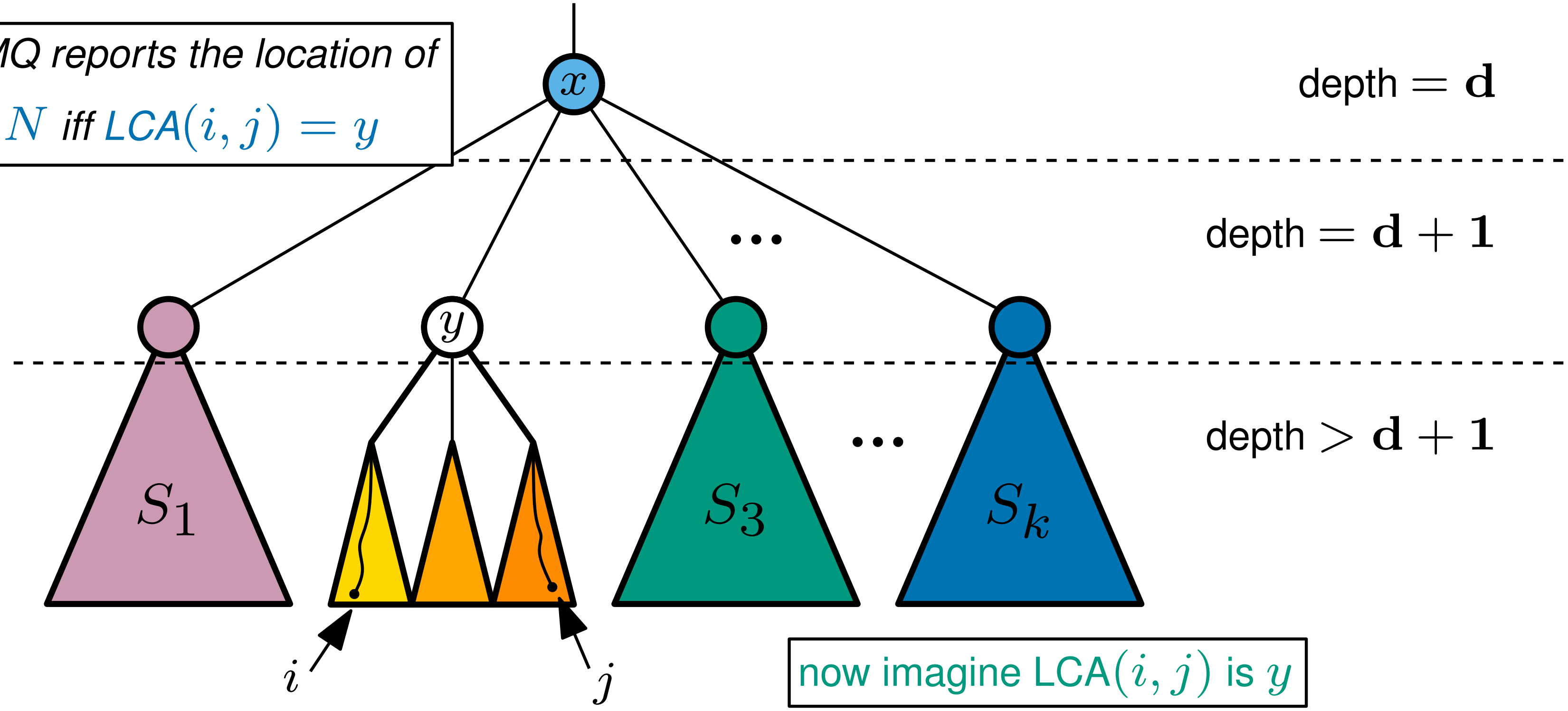
Claim *the RMQ reports the location of some y in N iff LCA$(i, j) = y$*


x
depth $= \mathbf{d}$
depth $= \mathbf{d+1}$
depth $> \mathbf{d+1}$
y
S_1
S_3
S_k
...
i
j
(node) N
(depth) D
x y y y y x x ... x x
$\mathbf{d}$ $\mathbf{d}$ $\mathrm{d+1}$ $\mathrm{d+1}$ $\mathrm{d+1}$ $\mathrm{d+1}$ $\mathbf{d}$ $\mathbf{d}$ ... $\mathbf{d}$ $\mathbf{d}$
tour of S_1
tour of S_2
tour of S_3
tour of S_k

Solving LCA using RMQ - correctness

We can also define a Euler tour of T recursively...

Claim *the RMQ reports the location of some y in N iff LCA$(i, j) = y$*



now imagine LCA(i, j) is y

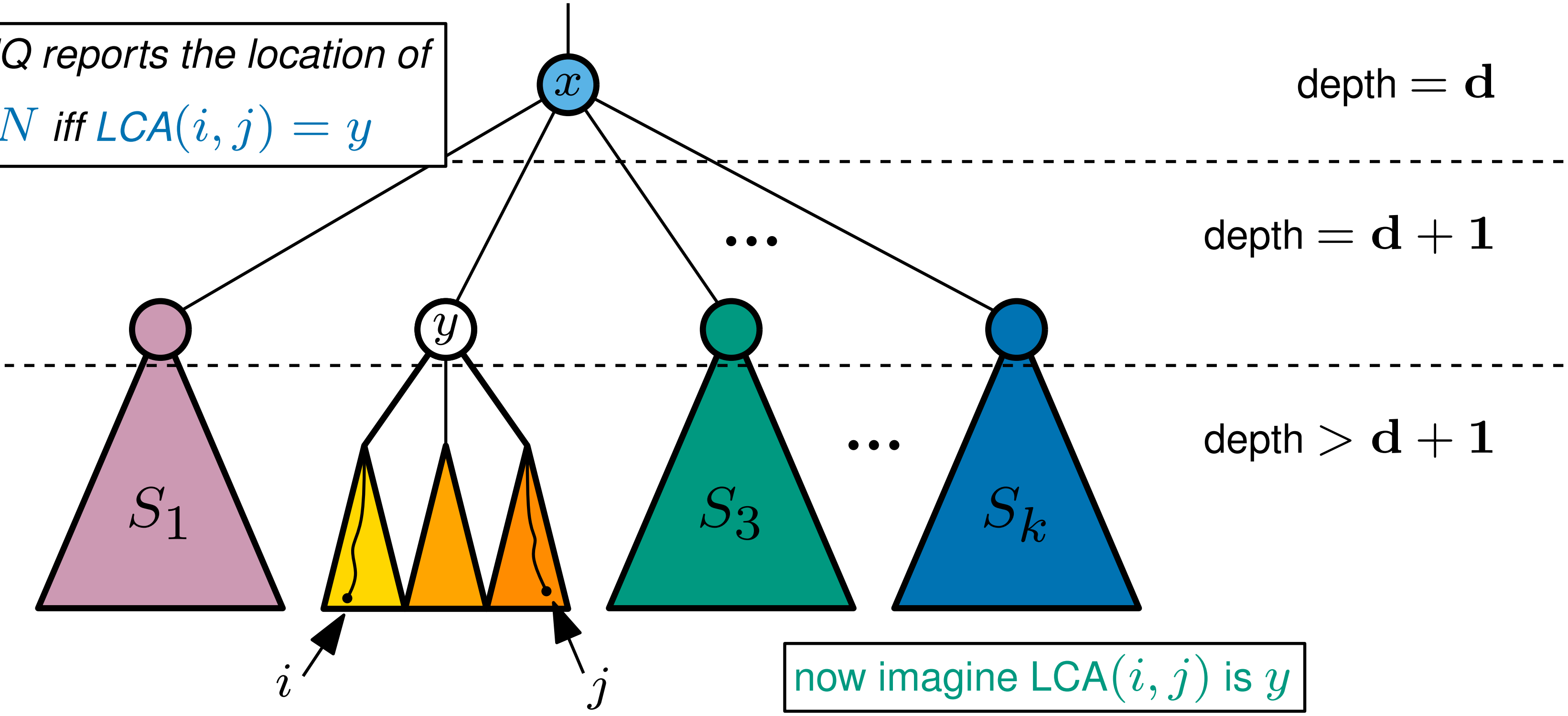
(node) N	x		x	y		y		y		y	x		x	...	x		x
(depth) D	$\mathbf{d}$		$\mathbf{d}$	$\mathrm{d+1}$		$\mathrm{d+1}$		$\mathrm{d+1}$		$\mathrm{d+1}$	$\mathbf{d}$		$\mathbf{d}$	...	$\mathbf{d}$		$\mathbf{d}$

tour of S_1 tour of S_2 tour of S_3 tour of S_k

Solving LCA using RMQ - correctness

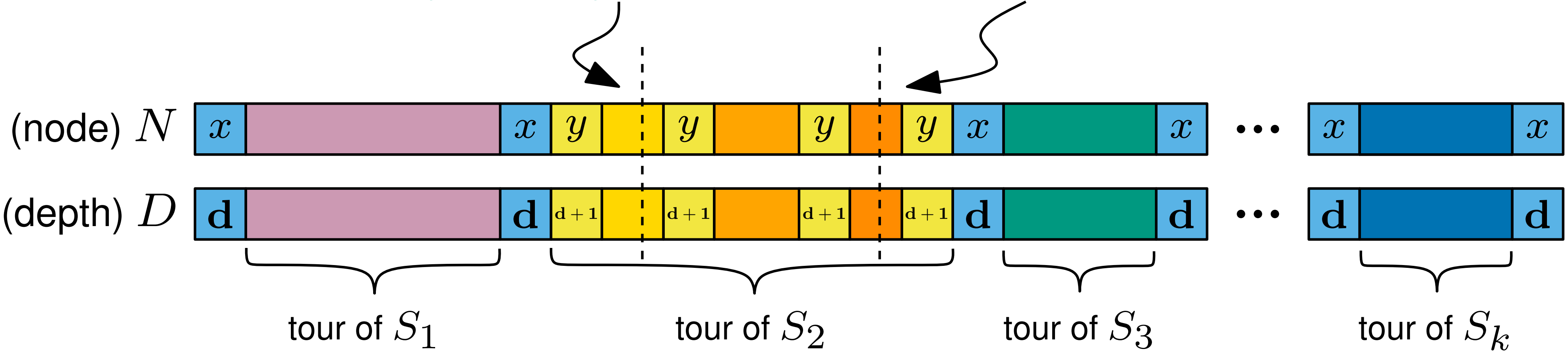
We can also define a Euler tour of T recursively...

Claim *the RMQ reports the location of some y in N iff LCA$(i,j) = y$*



now imagine LCA(i,j) is y

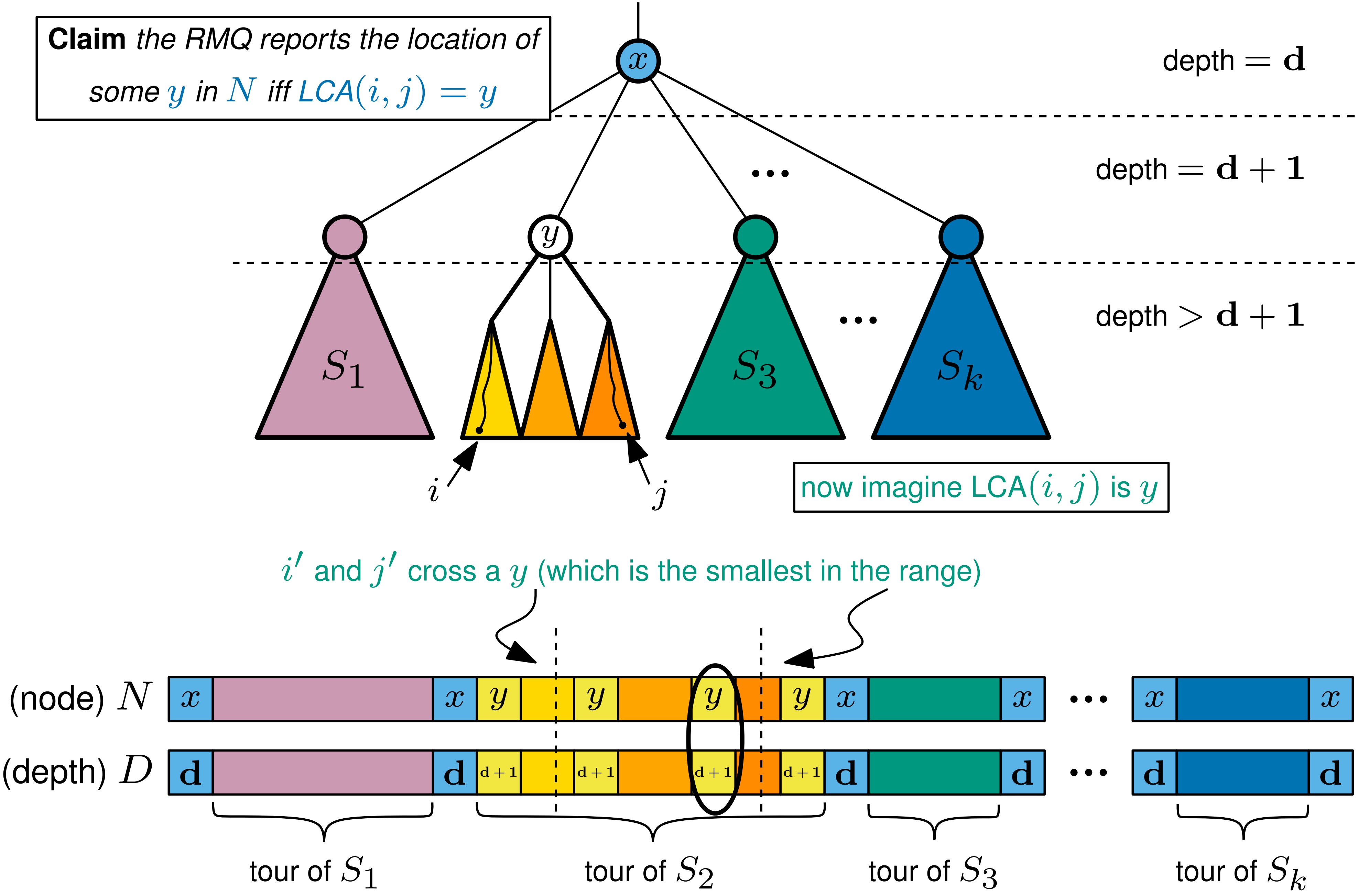
i' and j' cross a y (which is the smallest in the range)



Solving LCA using RMQ - correctness

We can also define a Euler tour of T recursively...

Claim *the RMQ reports the location of some y in N iff LCA$(i,j) = y$*



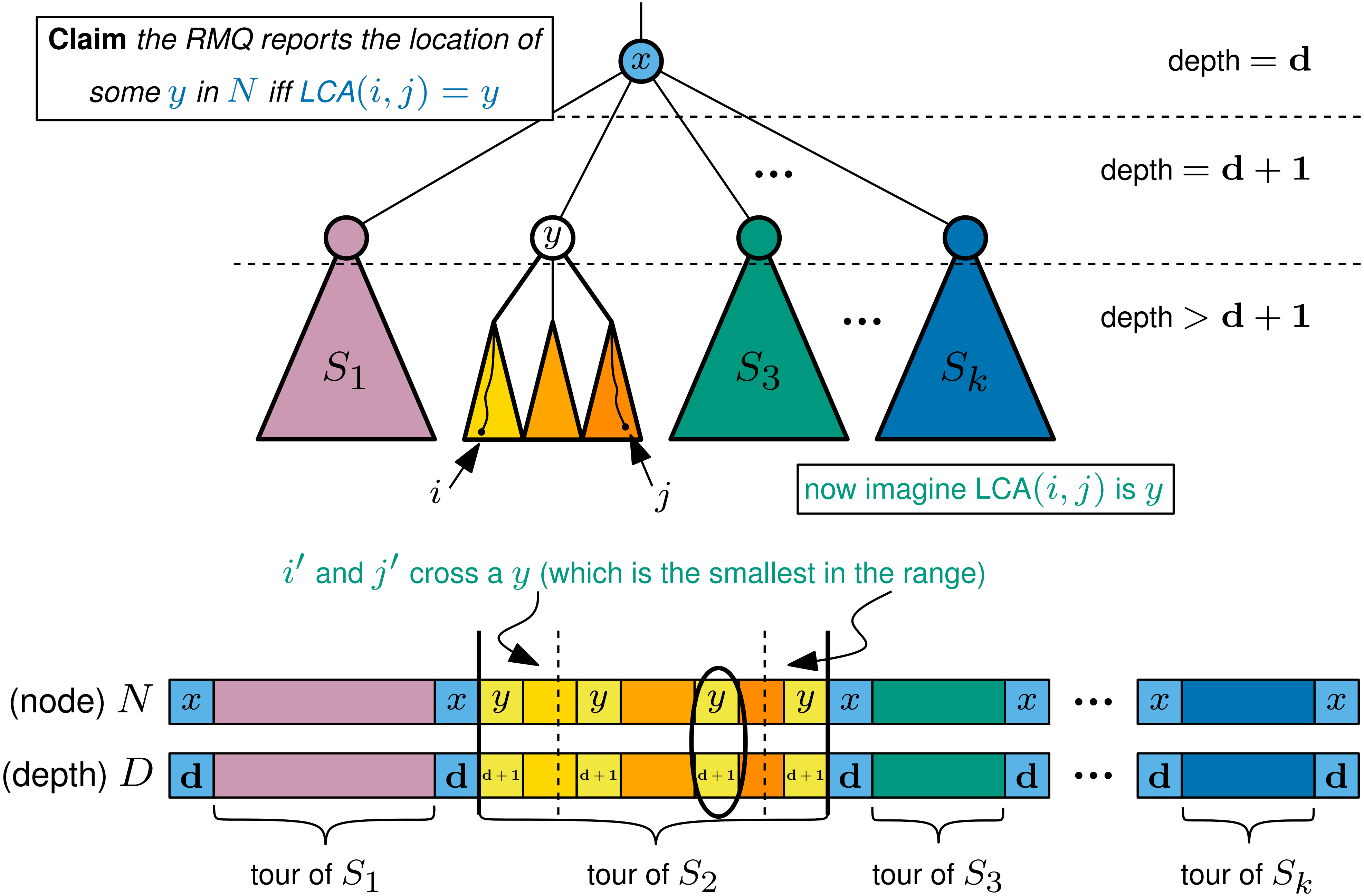
now imagine LCA(i,j) is y

i' and j' cross a y (which is the smallest in the range)

Solving LCA using RMQ - correctness

We can also define a Euler tour of T recursively...

Claim *the RMQ reports the location of some y in N iff LCA$(i, j) = y$*



x
depth $= \mathbf{d}$
depth $= \mathbf{d+1}$
depth $> \mathbf{d+1}$
y
S_1
S_3
S_k
i
j
(node) N
x x y y y y x x x x
(depth) D
$\mathbf{d}$ $\mathbf{d}$ $\mathbf{d+1}$ $\mathbf{d+1}$ $\mathbf{d+1}$ $\mathbf{d+1}$ $\mathbf{d}$ $\mathbf{d}$ $\mathbf{d}$ $\mathbf{d}$
tour of S_1
tour of S_2
tour of S_3
tour of S_k

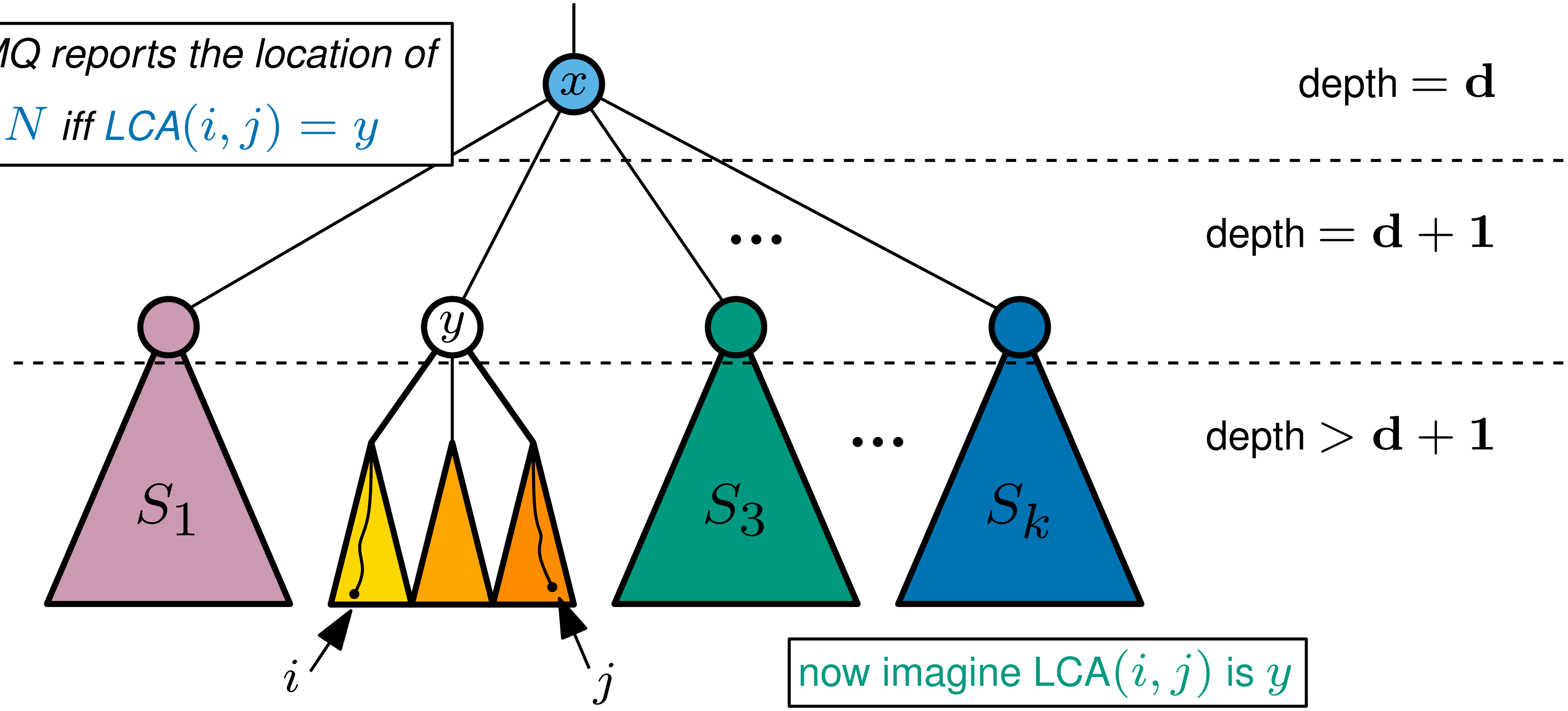
now imagine LCA(i, j) is y

i' and j' cross a y (which is the smallest in the range)

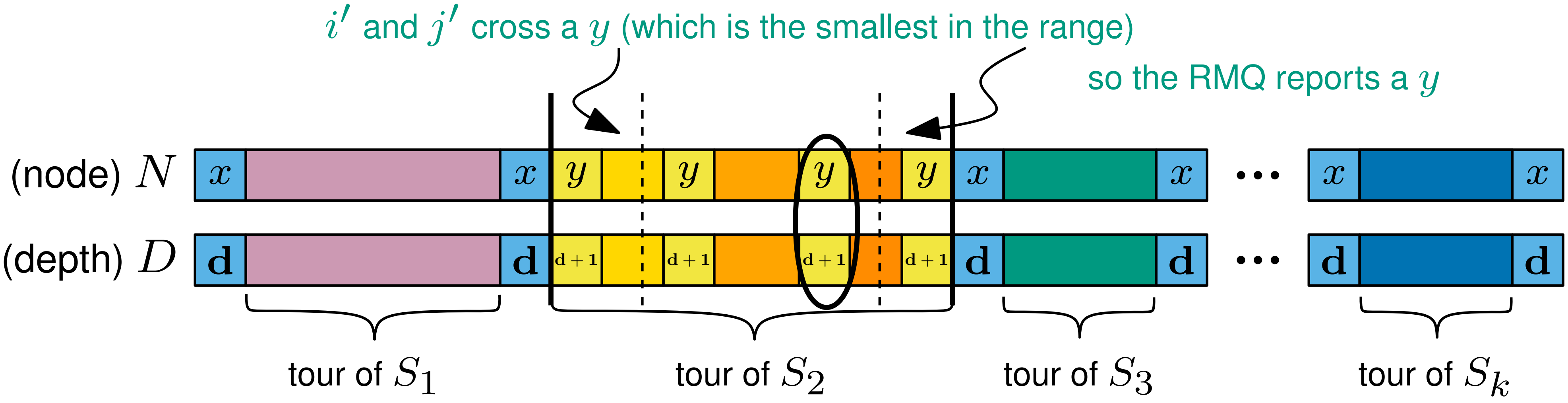
Solving LCA using RMQ - correctness

We can also define a Euler tour of T recursively...

Claim *the RMQ reports the location of some y in N iff LCA$(i,j) = y$*



now imagine LCA(i,j) is y



Ongoing Summary

We have seen an $O(n \log\log n)$ space, $O(n \log\log n)$ prep. time and $O(1)$ query time solution

for the Lowest Common Ancestor problem

which uses solution 3 for RMQ *from last lecture*

Ongoing Summary

We have seen an $O(n \log \log n)$ space, $O(n \log \log n)$ prep. time and $O(1)$ query time solution

for the Lowest Common Ancestor problem

which uses solution 3 for RMQ *from last lecture*

Can we do better?

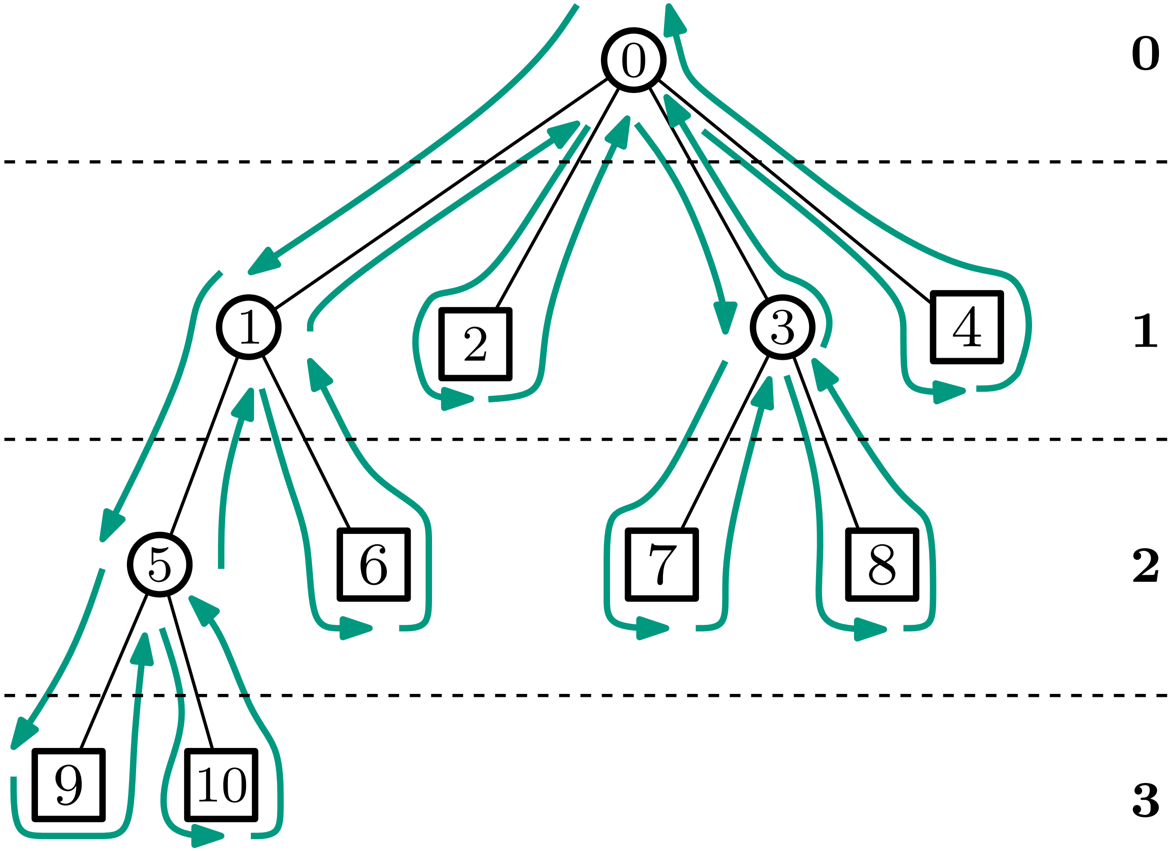
Solving LCAs using RMQs - efficiency

Preprocessing Summary

1. Construct N and D from T
2. Add a pointer from each node i to some $N[i'] = i$
3. Preprocess D for RMQs

Query Summary - LCA(i,j)

1. Find (any) i' st. $N[i'] = i$
2. Find (any) j' st. $N[j'] = j$
3. Compute RMQ(i', j') in D
4. LCA$(i, j) = N[$RMQ$(i', j')]$



(node) N	0	1	5	9	5	10	5	1	6	1	0	2	0	3	7	3	8	3	0	4	0
(depth) D	**0**	**1**	**2**	**3**	**2**	**3**	**2**	**1**	**2**	**1**	**0**	**1**	**0**	**1**	**2**	**1**	**2**	**1**	**0**	**1**	**0**

$2n-1$

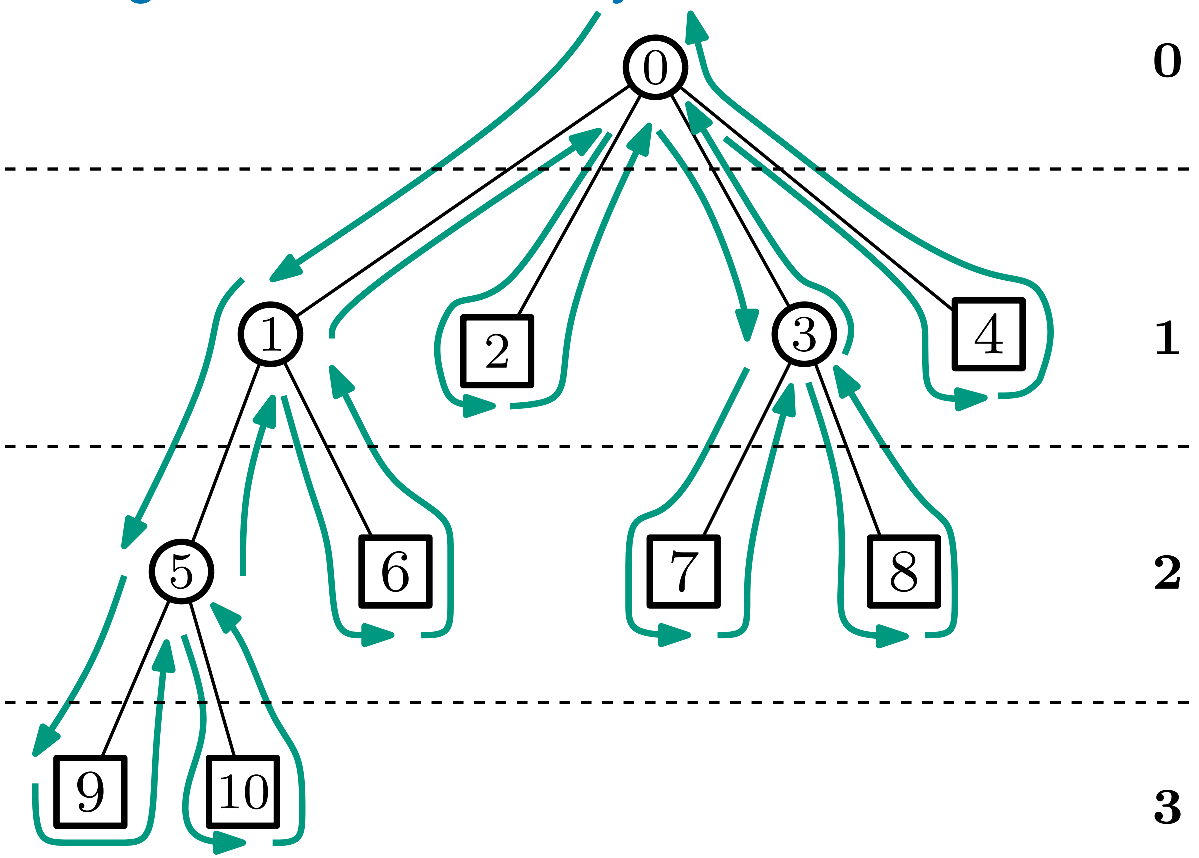
Solving LCAs using RMQs - efficiency

Preprocessing Summary

1. Construct N and D from T
2. Add a pointer from each node i to some $N[i'] = i$
3. Preprocess D for RMQs

Query Summary - LCA(i,j)

1. Find (any) i' st. $N[i'] = i$
2. Find (any) j' st. $N[j'] = j$
3. Compute RMQ(i', j') in D
4. LCA$(i, j) = N[$RMQ$(i', j')]$



(node) N	0	1	5	9	5	10	5	1	6	1	0	2	0	3	7	3	8	3	0	4	0
(depth) D	**0**	**1**	**2**	**3**	**2**	**3**	**2**	**1**	**2**	**1**	**0**	**1**	**0**	**1**	**2**	**1**	**2**	**1**	**0**	**1**	**0**

$2n-1$

Notice anything interesting about D?

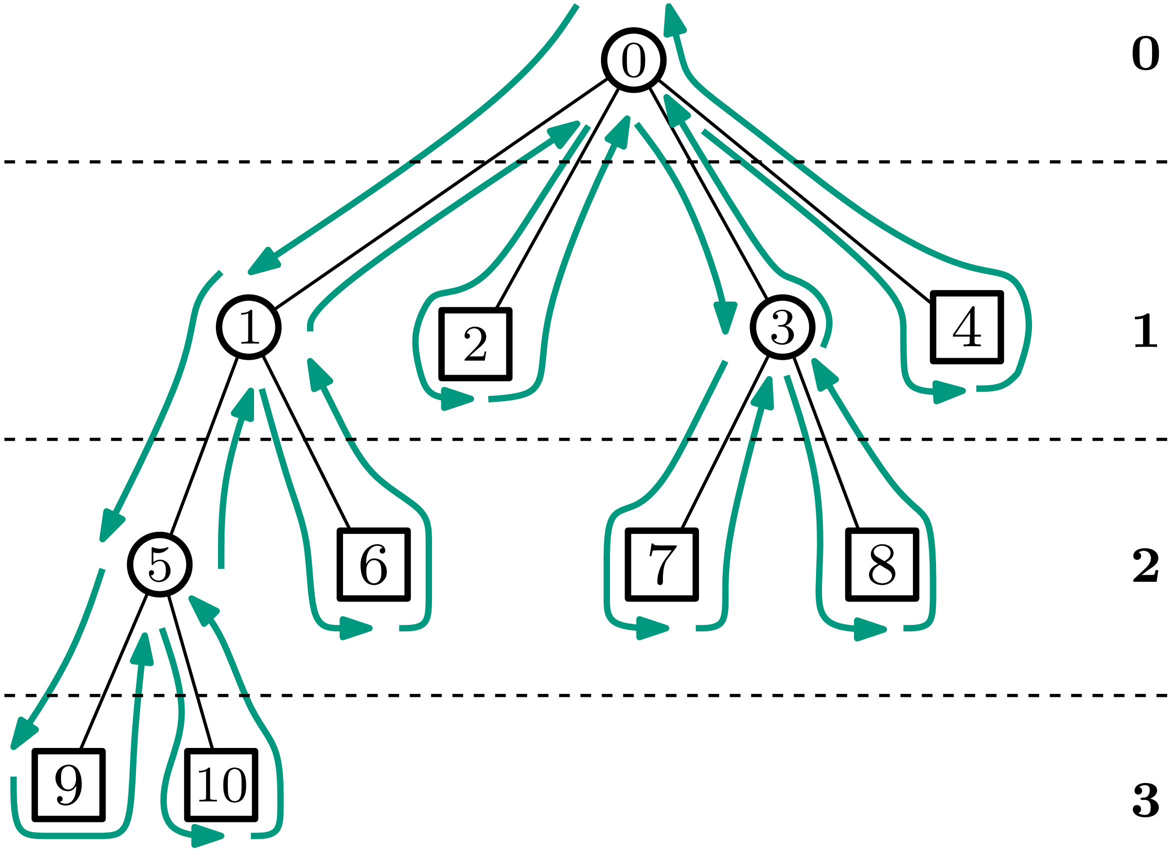
Solving LCAs using RMQs - efficiency

Preprocessing Summary

1. Construct N and D from T
2. Add a pointer from each node i to some $N[i'] = i$
3. Preprocess D for RMQs

Query Summary - LCA(i,j)

1. Find (any) i' st. $N[i'] = i$
2. Find (any) j' st. $N[j'] = j$
3. Compute RMQ(i', j') in D
4. LCA$(i, j) = N[$RMQ$(i', j')]$



(node) N	0	1	5	9	5	10	5	1	6	1	0	2	0	3	7	3	8	3	0	4	0
(depth) D	**0**	**1**	**2**	**3**	**2**	**3**	**2**	**1**	**2**	**1**	**0**	**1**	**0**	**1**	**2**	**1**	**2**	**1**	**0**	**1**	**0**

$2n-1$

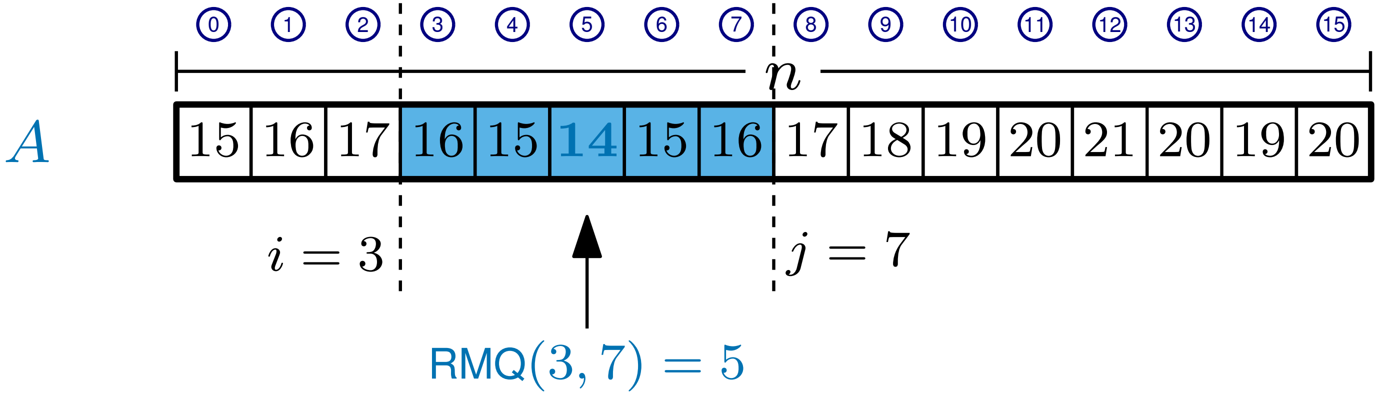
Notice anything interesting about D?

$$\boxed{D[i+1] = D[i] \pm 1}$$

± 1 Range minimum query

Preprocess an integer array A (length n) to answer range minimum queries...

where for all k, we have $A[k+1] = A[k] \pm 1$



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
15	16	17	16	15	14	15	16	17	18	19	20	21	20	19	20

After preprocessing, a **range minimum query** is given by RMQ(i, j)

the output is the location of the smallest element in $A[i, j]$

(in a tie, report the leftmost)

e.g. RMQ$(3, 7) = 5$, which is the location of the smallest element in $A[3, 7]$

- Can we exploit this ± 1 property to get a more efficient RMQ data structure?

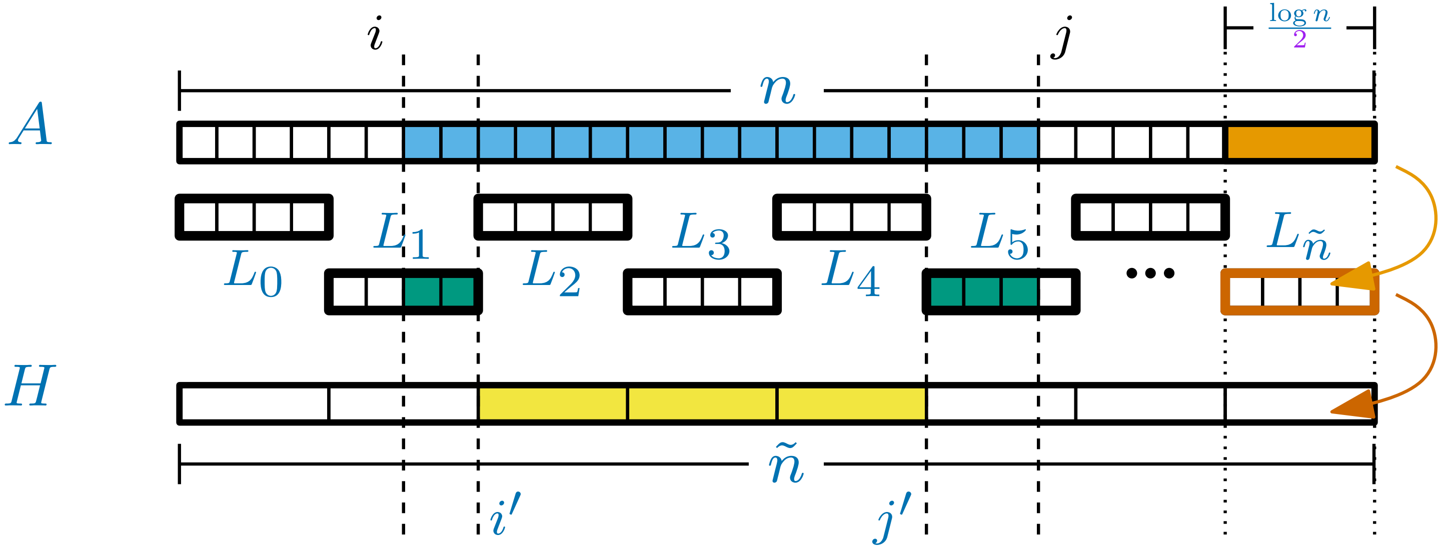
- Ideally we would like $O(n)$ space, $O(n)$ prep. time and $O(1)$ query time

Low-resolution RMQ (again)

$$\tilde{n} = \frac{2n}{\log n}$$

Key Idea replace A with a smaller, *'low resolution'* array H

and many small arrays $L_0, L_1, L_2 \ldots$ *'for the details'*



all of these go in here

min of **these** goes in here

Preprocess the array H $\left(\text{which has length } \tilde{n} = \frac{2n}{\log n}\right)$ to answer RMQs. . .

in $O(n)$ space/prep time

Preprocess each array L_i (which has length $(\log n)/2$) to answer RMQs. . .

in $O(\log n \log \log n)$ space/prep time

as there are $O(n/\log n)$ L_i arrays, we have $O(n \log \log n)$ total space/prep time

---

How do we answer a query in A in $O(1)$ time?

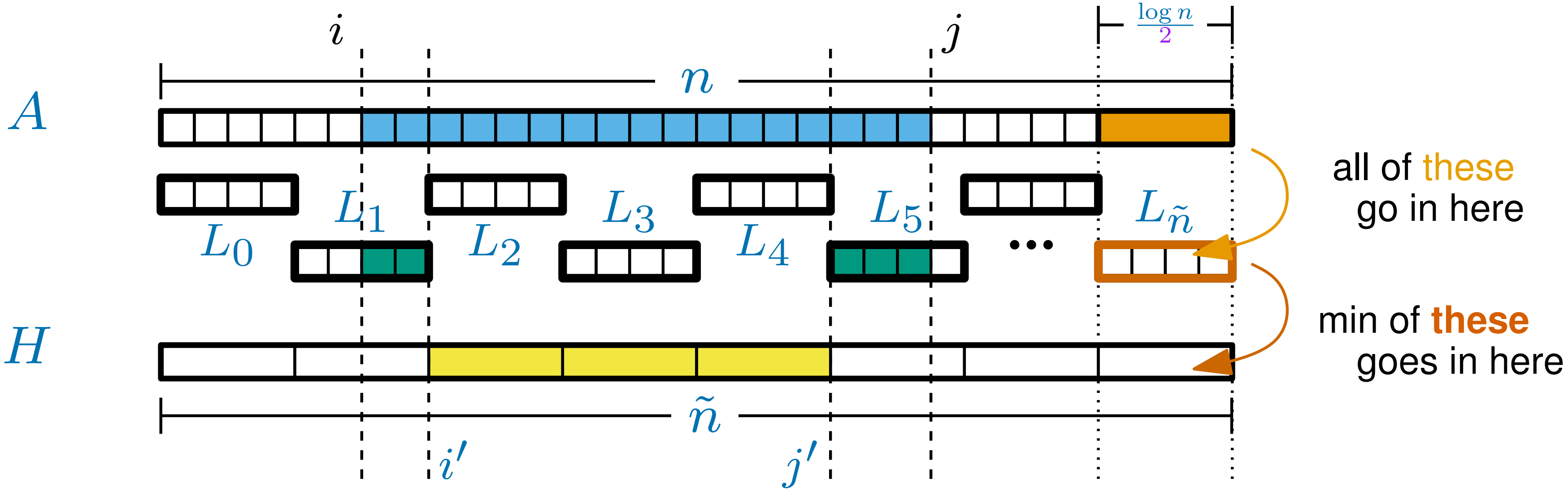
Do one query in H and one query in two different L_i and return the smallest

Low-resolution RMQ (again)

$$\boxed{\tilde{n} = \frac{2n}{\log n}}$$

Key Idea replace A with a smaller, *'low resolution'* array H

and many small arrays $L_0, L_1, L_2 \ldots$ *'for the details'*



Preprocess the array H $\left(\text{which has length } \tilde{n} = \frac{2n}{\log n}\right)$ to answer RMQs...

in $O(n)$ space/prep time

Preprocess each array L_i (which has length $(\log n)/2$) to answer RMQs...

too big and slow! → in $O(\log n \log \log n)$ space/prep time

as there are $O(n/\log n)$ L_i arrays, we have $O(n \log \log n)$ total space/prep time

How do we answer a query in A in $O(1)$ time?

Do one query in H and one query in two different L_i and return the smallest

Counting ± 1 RMQ arrays

How many different ± 1 RMQ arrays like this. . . $\overset{L}{\underbrace{\boxed{\quad\quad\quad}}_{\frac{\log n}{2}}}$ are there?

Counting ± 1 RMQ arrays

How many different ± 1 RMQ arrays like this… $\overset{L}{\underset{\vdash\ \frac{\log n}{2}\ \dashv}{\boxed{}}}$ are there?

We say that $\overset{L_x}{\underset{\vdash\ \frac{\log n}{2}\ \dashv}{\boxed{}}}$ is equivalent to $\overset{L_y}{\underset{\vdash\ \frac{\log n}{2}\ \dashv}{\boxed{}}}$ iff for all (i, j): $\text{RMQ}_x(i, j) = \text{RMQ}_y(i, j)$

Counting ± 1 RMQ arrays

How many different ± 1 RMQ arrays like this. . . $\underbrace{\overset{L}{\boxed{}}}_{\frac{\log n}{2}}$ are there?

We say that $\underbrace{\overset{L_x}{\boxed{}}}_{\frac{\log n}{2}}$ is equivalent to $\underbrace{\overset{L_y}{\boxed{}}}_{\frac{\log n}{2}}$ iff for all (i, j): RMQ$_x(i, j) =$ RMQ$_y(i, j)$

(remember these are the locations of the minimum)

Counting ± 1 RMQ arrays

How many different ± 1 RMQ arrays like this... $\overset{L}{\underbrace{\boxed{\qquad\qquad}}_{\frac{\log n}{2}}}$ are there?

We say that $\overset{L_x}{\underbrace{\boxed{\qquad\qquad}}_{\frac{\log n}{2}}}$ is equivalent to $\overset{L_y}{\underbrace{\boxed{\qquad\qquad}}_{\frac{\log n}{2}}}$ iff for all (i, j): RMQ$_x(i, j)$ = RMQ$_y(i, j)$

(remember these are the locations of the minimum)

L_x

0	1	2	3	4
16	15	14	15	14

is equivalent to

L_y

0	1	2	3	4
10	9	8	9	8

Counting ± 1 RMQ arrays

How many different ± 1 RMQ arrays like this... $\overset{L}{\underbrace{\boxed{\qquad\quad}}_{\frac{\log n}{2}}}$ are there?

We say that $\overset{L_x}{\underbrace{\boxed{\qquad\quad}}_{\frac{\log n}{2}}}$ is equivalent to $\overset{L_y}{\underbrace{\boxed{\qquad\quad}}_{\frac{\log n}{2}}}$ iff for all (i, j): RMQ$_x(i, j) =$ RMQ$_y(i, j)$

(remember these are the locations of the minimum)

L_x

0	1	2	3	4
16	15	14	15	14

is equivalent to

L_y

0	1	2	3	4
10	9	8	9	8

RMQ$_x(0, 2) =$ RMQ$_y(0, 2) = 2$

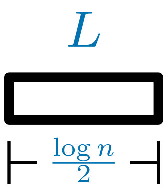
RMQ$_x(3, 4) =$ RMQ$_y(3, 4) = 4$

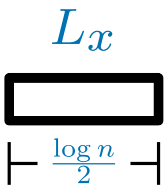
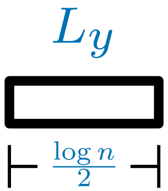
RMQ$_x(0, 4) =$ RMQ$_y(0, 4) = 2$

RMQ$_x(0, 1) =$ RMQ$_y(0, 1) = 1$

$\vdots$

Counting ± 1 RMQ arrays

How many different ± 1 RMQ arrays like this. . .  are there?

We say that  is equivalent to  iff for all (i, j): RMQ$_x(i, j) =$ RMQ$_y(i, j)$

(remember these are the locations of the minimum)

L_x

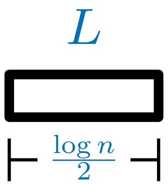
0	1	2	3	4
16	15	14	15	14

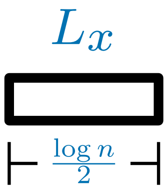
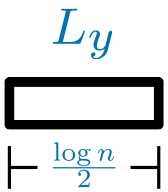
is equivalent to

L_y

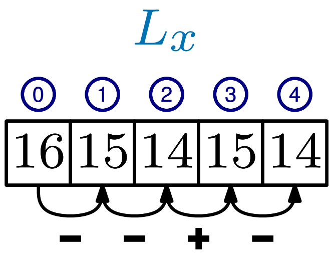
0	1	2	3	4
10	9	8	9	8

Counting ± 1 RMQ arrays

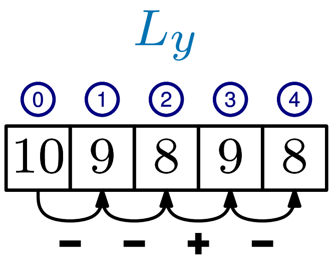
How many different ± 1 RMQ arrays like this...  are there?

We say that  is equivalent to  iff for all (i,j): RMQ$_x(i,j)$ = RMQ$_y(i,j)$

(remember these are the locations of the minimum)



is equivalent to

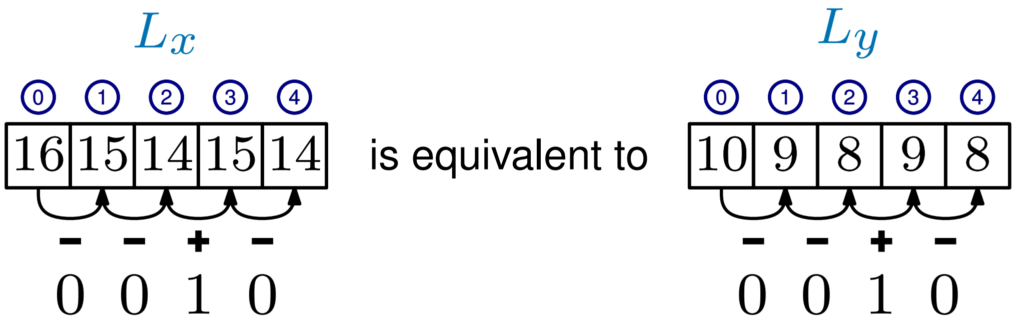


Counting ± 1 RMQ arrays

How many different ± 1 RMQ arrays like this... $\underset{\vdash \frac{\log n}{2} \dashv}{\overset{L}{\boxed{\quad\quad\quad}}}$ are there?

We say that $\underset{\vdash \frac{\log n}{2} \dashv}{\overset{L_x}{\boxed{\quad\quad\quad}}}$ is equivalent to $\underset{\vdash \frac{\log n}{2} \dashv}{\overset{L_y}{\boxed{\quad\quad\quad}}}$ iff for all (i, j): RMQ$_x(i, j) =$ RMQ$_y(i, j)$

(remember these are the locations of the minimum)



L_x

0	1	2	3	4
16	15	14	15	14

− − + −

0 0 1 0

is equivalent to

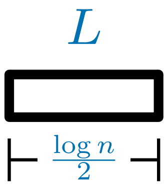
L_y

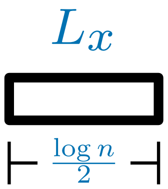
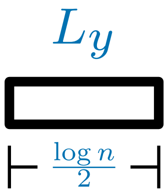
0	1	2	3	4
10	9	8	9	8

− − + −

0 0 1 0

Counting ± 1 RMQ arrays

How many different ± 1 RMQ arrays like this…  are there?

We say that  is equivalent to  iff for all (i, j): RMQ$_x(i, j) =$ RMQ$_y(i, j)$

(remember these are the locations of the minimum)

L_x

0	1	2	3	4
16	15	14	15	14

– – + –

$0 \quad 0 \quad 1 \quad 0$

is equivalent to

L_y

0	1	2	3	4
10	9	8	9	8

– – + –

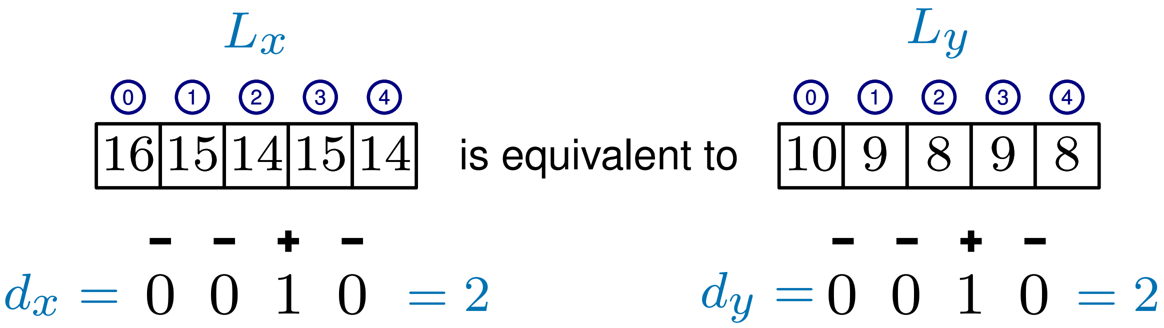
$0 \quad 0 \quad 1 \quad 0$

Counting ± 1 RMQ arrays

How many different ± 1 RMQ arrays like this... $\underset{\vdash \frac{\log n}{2} \dashv}{\overset{L}{\boxed{}}}$ are there?

We say that $\underset{\vdash \frac{\log n}{2} \dashv}{\overset{L_x}{\boxed{}}}$ is equivalent to $\underset{\vdash \frac{\log n}{2} \dashv}{\overset{L_y}{\boxed{}}}$ iff for all (i, j): RMQ$_x(i, j) =$ RMQ$_y(i, j)$

(remember these are the locations of the minimum)



L_x

0	1	2	3	4
16	15	14	15	14

is equivalent to

L_y

0	1	2	3	4
10	9	8	9	8

$d_x = \overset{-}{0}\ \overset{-}{0}\ \overset{+}{1}\ \overset{-}{0} = 2$

$d_y = \overset{-}{0}\ \overset{-}{0}\ \overset{+}{1}\ \overset{-}{0} = 2$

Counting ± 1 RMQ arrays

How many different ± 1 RMQ arrays like this... $\overset{L}{\underbrace{\boxed{}}_{\frac{\log n}{2}}}$ are there?

We say that $\overset{L_x}{\underbrace{\boxed{}}_{\frac{\log n}{2}}}$ is equivalent to $\overset{L_y}{\underbrace{\boxed{}}_{\frac{\log n}{2}}}$ iff for all (i, j): RMQ$_x(i, j) =$ RMQ$_y(i, j)$

(remember these are the locations of the minimum)

L_x

0	1	2	3	4
16	15	14	15	14

is equivalent to

L_y

0	1	2	3	4
10	9	8	9	8

Fact L_x is equivalent to L_y iff $d_x = d_y$

$d_x = 0 \;\; 0 \;\; 1 \;\; 0 \;\; = 2$

$d_y = 0 \;\; 0 \;\; 1 \;\; 0 \;\; = 2$

Counting ± 1 RMQ arrays

How many different ± 1 RMQ arrays like this... $\overset{L}{\underset{\vdash \frac{\log n}{2} \dashv}{\boxed{}}}$ are there?

We say that $\overset{L_x}{\underset{\vdash \frac{\log n}{2} \dashv}{\boxed{}}}$ is equivalent to $\overset{L_y}{\underset{\vdash \frac{\log n}{2} \dashv}{\boxed{}}}$ iff for all (i, j): RMQ$_x(i, j) =$ RMQ$_y(i, j)$

(remember these are the locations of the minimum)

L_x

0	1	2	3	4
16	15	14	15	14

is equivalent to

L_y

0	1	2	3	4
10	9	8	9	8

Fact L_x is equivalent to L_y
iff $d_x = d_y$

$d_x = 0\ \ 0\ \ 1\ \ 0\ \ = 2$

$d_y = 0\ \ 0\ \ 1\ \ 0\ \ = 2$

- We can precompute d_x for each L_x in $O(|L_x|) = O(\log n)$ time.

Counting ± 1 RMQ arrays

How many different ± 1 RMQ arrays like this… $\overset{L}{\underbrace{\boxed{}}_{\frac{\log n}{2}}}$ are there?

We say that $\overset{L_x}{\underbrace{\boxed{}}_{\frac{\log n}{2}}}$ is equivalent to $\overset{L_y}{\underbrace{\boxed{}}_{\frac{\log n}{2}}}$ iff for all (i, j): RMQ$_x(i, j) =$ RMQ$_y(i, j)$

(remember these are the locations of the minimum)

L_x

0	1	2	3	4
16	15	14	15	14

is equivalent to

L_y

0	1	2	3	4
10	9	8	9	8

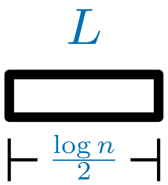
Fact L_x is equivalent to L_y iff $d_x = d_y$

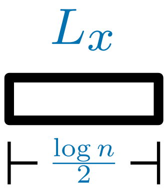
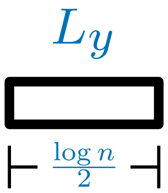
$d_x = 0\ \ 0\ \ 1\ \ 0\ \ = 2$

$d_y = 0\ \ 0\ \ 1\ \ 0\ \ = 2$

- We can precompute d_x for each L_x in $O(|L_x|) = O(\log n)$ time.
- How many different values of d are there?

Counting ±1 RMQ arrays

How many different ±1 RMQ arrays like this…  are there?

We say that  is equivalent to  iff for all (i, j): RMQ$_x(i, j) =$ RMQ$_y(i, j)$

(remember these are the locations of the minimum)

L_x

0	1	2	3	4
16	15	14	15	14

is equivalent to

L_y

0	1	2	3	4
10	9	8	9	8

$d_x = 0\ \ 0\ \ 1\ \ 0\ \ = 2$

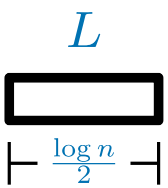
$d_y = 0\ \ 0\ \ 1\ \ 0\ \ = 2$

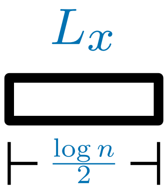
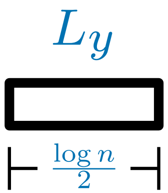
Fact L_x is equivalent to L_y iff $d_x = d_y$

- We can precompute d_x for each L_x in $O(|L_x|) = O(\log n)$ time.
- How many different values of d are there?

  d contains $(\log n)/2 - 1$ bits so …

Counting ± 1 RMQ arrays

How many different ± 1 RMQ arrays like this...  are there?

We say that  is equivalent to  iff for all (i,j): RMQ$_x(i,j) =$ RMQ$_y(i,j)$

(remember these are the locations of the minimum)

L_x

0	1	2	3	4
16	15	14	15	14

is equivalent to

L_y

0	1	2	3	4
10	9	8	9	8

Fact L_x is equivalent to L_y iff $d_x = d_y$

$d_x = 0\ \ 0\ \ 1\ \ 0\ \ = 2$ $\qquad$ $d_y = 0\ \ 0\ \ 1\ \ 0\ \ = 2$

- We can precompute d_x for each L_x in $O(|L_x|) = O(\log n)$ time.
- How many different values of d are there?

d contains $(\log n)/2 - 1$ bits so ... $\qquad$ at most $2^{(\log n)/2}$

Counting ± 1 RMQ arrays

How many different ± 1 RMQ arrays like this… $\underbrace{\overset{L}{\boxed{}}}_{\frac{\log n}{2}}$ are there?

We say that $\underbrace{\overset{L_x}{\boxed{}}}_{\frac{\log n}{2}}$ is equivalent to $\underbrace{\overset{L_y}{\boxed{}}}_{\frac{\log n}{2}}$ iff for all (i, j): RMQ$_x(i, j) =$ RMQ$_y(i, j)$

(remember these are the locations of the minimum)

L_x

0	1	2	3	4
16	15	14	15	14

is equivalent to

L_y

0	1	2	3	4
10	9	8	9	8

Fact L_x is equivalent to L_y
iff $d_x = d_y$

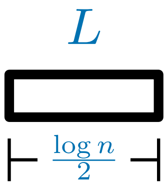
$d_x = 0\ \ 0\ \ 1\ \ 0\ \ = 2$

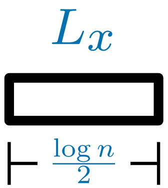
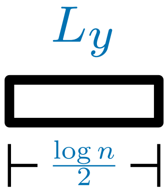
$d_y = 0\ \ 0\ \ 1\ \ 0\ \ = 2$

- We can precompute d_x for each L_x in $O(|L_x|) = O(\log n)$ time.
- How many different values of d are there?

d contains $(\log n)/2 - 1$ bits so … at most $2^{(\log n)/2} = \left(2^{\log n}\right)^{1/2}$

Counting ± 1 RMQ arrays

How many different ± 1 RMQ arrays like this...  are there?

We say that  is equivalent to  iff for all (i, j): RMQ$_x(i, j)$ = RMQ$_y(i, j)$

(remember these are the locations of the minimum)

L_x

0	1	2	3	4
16	15	14	15	14

is equivalent to

L_y

0	1	2	3	4
10	9	8	9	8

$d_x = 0\ \ 0\ \ 1\ \ 0\ = 2$

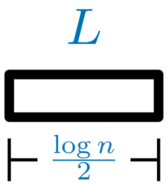
$d_y = 0\ \ 0\ \ 1\ \ 0\ = 2$

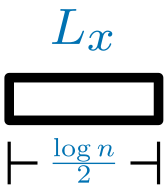
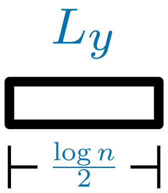
Fact L_x is equivalent to L_y iff $d_x = d_y$

- We can precompute d_x for each L_x in $O(|L_x|) = O(\log n)$ time.
- How many different values of d are there?

d contains $(\log n)/2 - 1$ bits so ... at most $2^{(\log n)/2} = \left(2^{\log n}\right)^{1/2} \leqslant \sqrt{n}$

Counting ± 1 RMQ arrays

How many different ± 1 RMQ arrays like this...  are there?

We say that  is equivalent to  iff for all (i,j): RMQ$_x(i,j) =$ RMQ$_y(i,j)$

(remember these are the locations of the minimum)

L_x

0	1	2	3	4
16	15	14	15	14

is equivalent to

L_y

0	1	2	3	4
10	9	8	9	8

Fact L_x is equivalent to L_y iff $d_x = d_y$

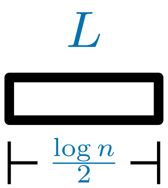
$d_x = 0\ \ 0\ \ 1\ \ 0\ \ = 2$

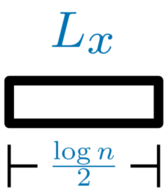
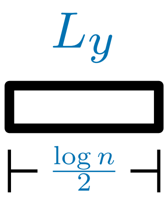
$d_y = 0\ \ 0\ \ 1\ \ 0\ \ = 2$

- We can precompute d_x for each L_x in $O(|L_x|) = O(\log n)$ time.
- How many different values of d are there?

  d contains $(\log n)/2 - 1$ bits so ... at most $2^{(\log n)/2} = \left(2^{\log n}\right)^{1/2} \leqslant \sqrt{n}$
- For each value of d we store RMQ(i,j) for all i, j

Counting ± 1 RMQ arrays

How many different ± 1 RMQ arrays like this...  are there?

We say that  is equivalent to  iff for all (i, j): RMQ$_x(i, j) =$ RMQ$_y(i, j)$

(remember these are the locations of the minimum)

L_x

0	1	2	3	4
16	15	14	15	14

is equivalent to

L_y

0	1	2	3	4
10	9	8	9	8

Fact L_x is equivalent to L_y iff $d_x = d_y$

$d_x = 0\ \ 0\ \ 1\ \ 0\ = 2$ $\qquad$ $d_y = 0\ \ 0\ \ 1\ \ 0\ = 2$

- We can precompute d_x for each L_x in $O(|L_x|) = O(\log n)$ time.
- How many different values of d are there?

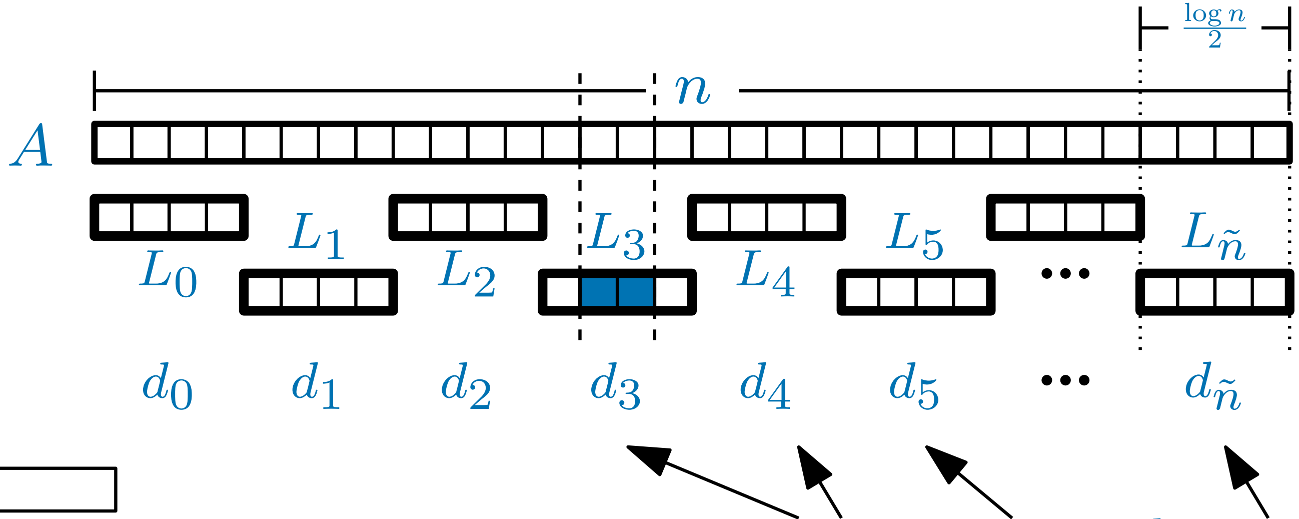
  d contains $(\log n)/2 - 1$ bits so ... $\qquad$ at most $2^{(\log n)/2} = \left(2^{\log n}\right)^{1/2} \leqslant \sqrt{n}$
- For each value of d we store RMQ(i, j) for all i, j

...this requires $O(\sqrt{n} \log^2 n) = O(n)$ total space and prep. time

RMQ on the L arrays in linear space

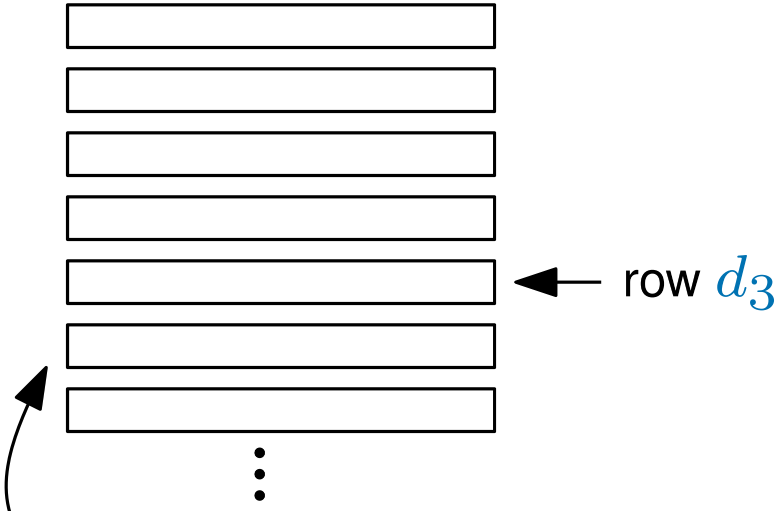
$$\boxed{\tilde{n} = \frac{2n}{\log n}}$$

Key Idea replace A with a smaller, *'low resolution'* array H



precompute the value of d_x for each L_x

in $O(n)$ total space and prep. time



Precompute all the RMQ answers for

every value $0 \leqslant d \leqslant \sqrt{n}$

in $O(n)$ total space and prep. time

To perform a query within some L_x

- Look up d_x
- Find the row d_x in the table
- Find the entry giving $\text{RMQ}_x(i, j)$

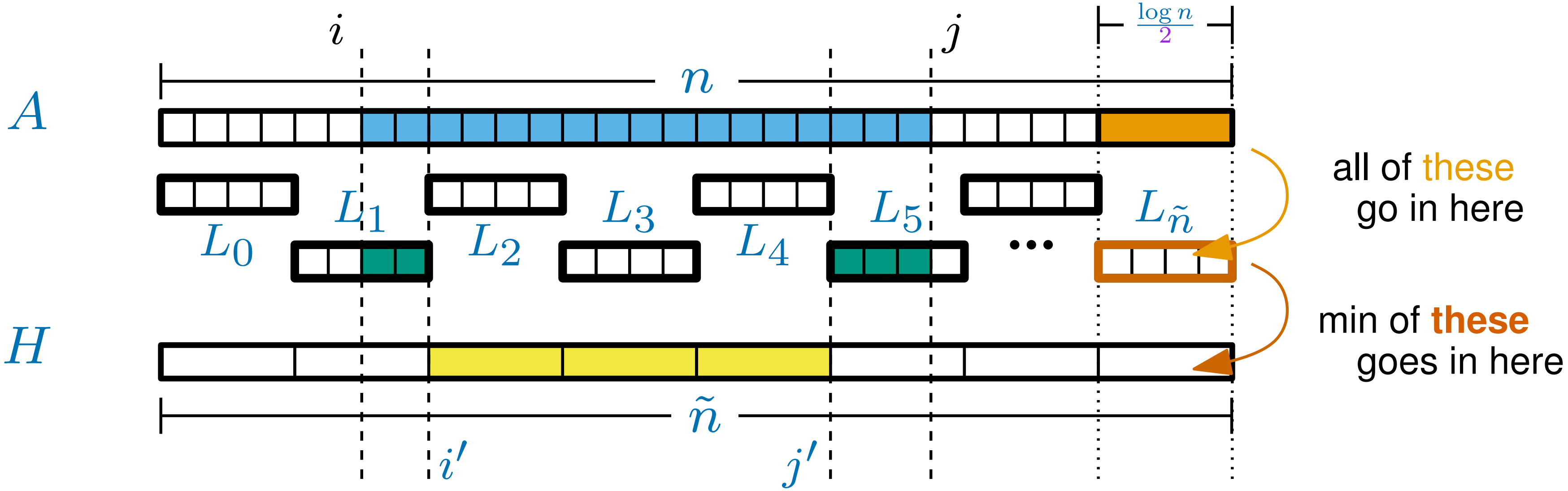
This takes $O(1)$ time

Optimal ± 1 RMQ

$$\tilde{n} = \frac{2n}{\log n}$$

Key Idea replace A with a smaller, *'low resolution'* array H

and many small arrays $L_0, L_1, L_2 \ldots$ *'for the details'*



Preprocess the array H to answer RMQs...

in $O(n)$ space/prep time

Preprocess each array L_i (which has length $(\log n)/2$) to answer RMQs...

build a complete table of answers

$O(n)$ total space/prep time

How do we answer a query in A in $O(1)$ time?

Do one query in H and one query in two different L_i and return the smallest

Ongoing Summary

We have seen an $O(n \log \log n)$ space, $O(n \log \log n)$ prep. time and $O(1)$ query time solution

for the Lowest Common Ancestor problem

which uses solution 3 for RMQ *from last lecture*

We have seen an $O(n)$ space, $O(n)$ prep. time and $O(1)$ query time solution

for the ± 1 Range Minimum Query problem

which improves solution 3 for RMQ *from last lecture*

(but only for ± 1 inputs)

Ongoing Summary

We have seen an $O(n \log \log n)$ space, $O(n \log \log n)$ prep. time and $O(1)$ query time solution

for the Lowest Common Ancestor problem

which uses solution 3 for RMQ *from last lecture*

We have seen an $O(n)$ space, $O(n)$ prep. time and $O(1)$ query time solution

for the ± 1 Range Minimum Query problem

which improves solution 3 for RMQ *from last lecture*

(but only for ± 1 inputs)

How does this affect our LCA solution?

Solving LCAs using RMQs

Preprocessing Summary

$O(n)$ 1. Construct N and D from T

$O(n)$ 2. Add a pointer from each node i to some $N[i'] = i$

$O(n)$ 3. Preprocess D for RMQs

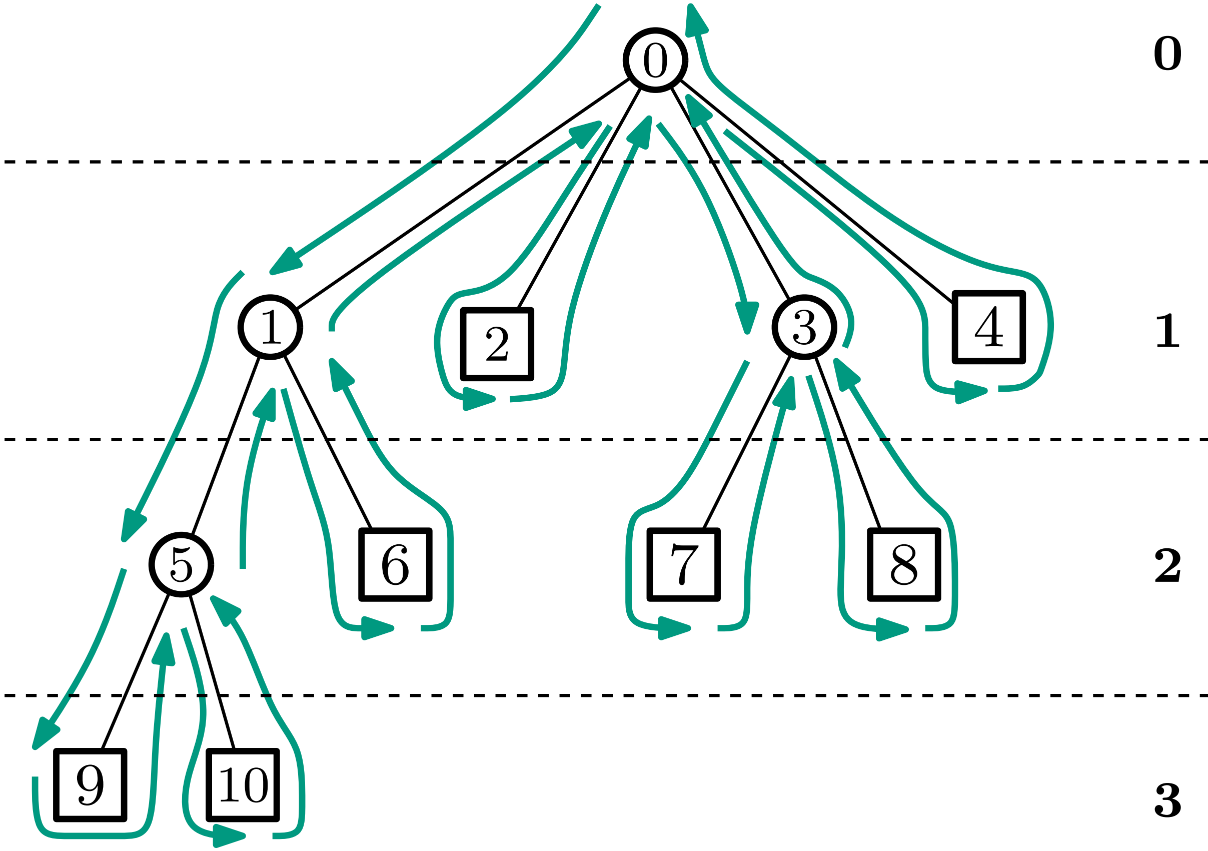
Query Summary - LCA(i,j)

$O(1)$ 1. Find (any) i' st. $N[i'] = i$

$O(1)$ 2. Find (any) j' st. $N[j'] = j$

$O(1)$ 3. Compute RMQ(i', j') in D

$O(1)$ 4. LCA$(i, j) = N[$RMQ$(i', j')]$



(node) N	0	1	5	9	5	10	5	1	6	1	0	2	0	3	7	3	8	3	0	4	0

$2n-1$

(depth) D	**0**	**1**	**2**	**3**	**2**	**3**	**2**	**1**	**2**	**1**	**0**	**1**	**0**	**1**	**2**	**1**	**2**	**1**	**0**	**1**	**0**

Solving LCAs using RMQs

Preprocessing Summary

$O(n)$ 1. Construct N and D from T

$O(n)$ 2. Add a pointer from each node i to some $N[i'] = i$

$O(n)$ 3. Preprocess D for RMQs

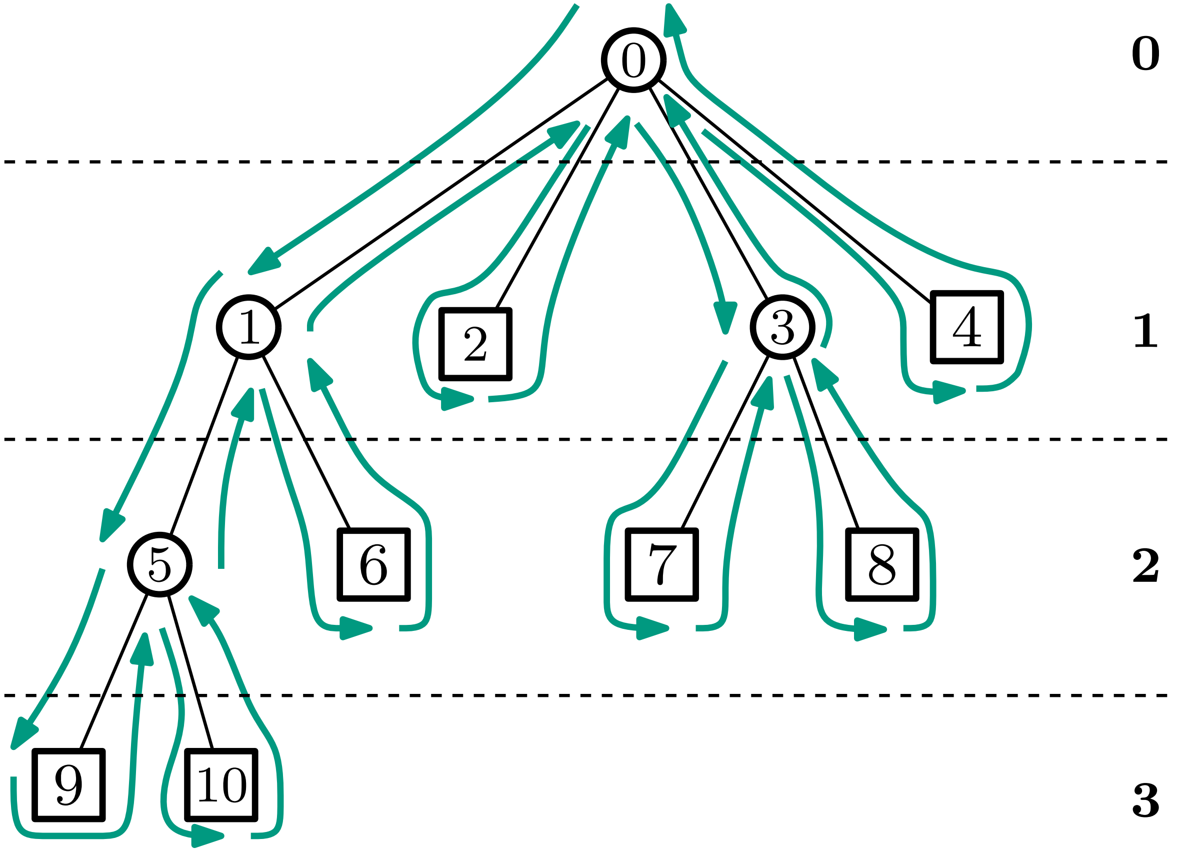
Query Summary - LCA(i,j)

$O(1)$ 1. Find (any) i' st. $N[i'] = i$

$O(1)$ 2. Find (any) j' st. $N[j'] = j$

$O(1)$ 3. Compute RMQ(i', j') in D

$O(1)$ 4. LCA$(i, j) = N[$RMQ$(i', j')]$



(node) N	0	1	5	9	5	10	5	1	6	1	0	2	0	3	7	3	8	3	0	4	0
(depth) D	**0**	**1**	**2**	**3**	**2**	**3**	**2**	**1**	**2**	**1**	**0**	**1**	**0**	**1**	**2**	**1**	**2**	**1**	**0**	**1**	**0**

$2n-1$

Solving LCAs using RMQs

Preprocessing Summary

$O(n)$ 1. Construct N and D from T

$O(n)$ 2. Add a pointer from each node i to some $N[i'] = i$

$O(n)$ 3. Preprocess D for RMQs

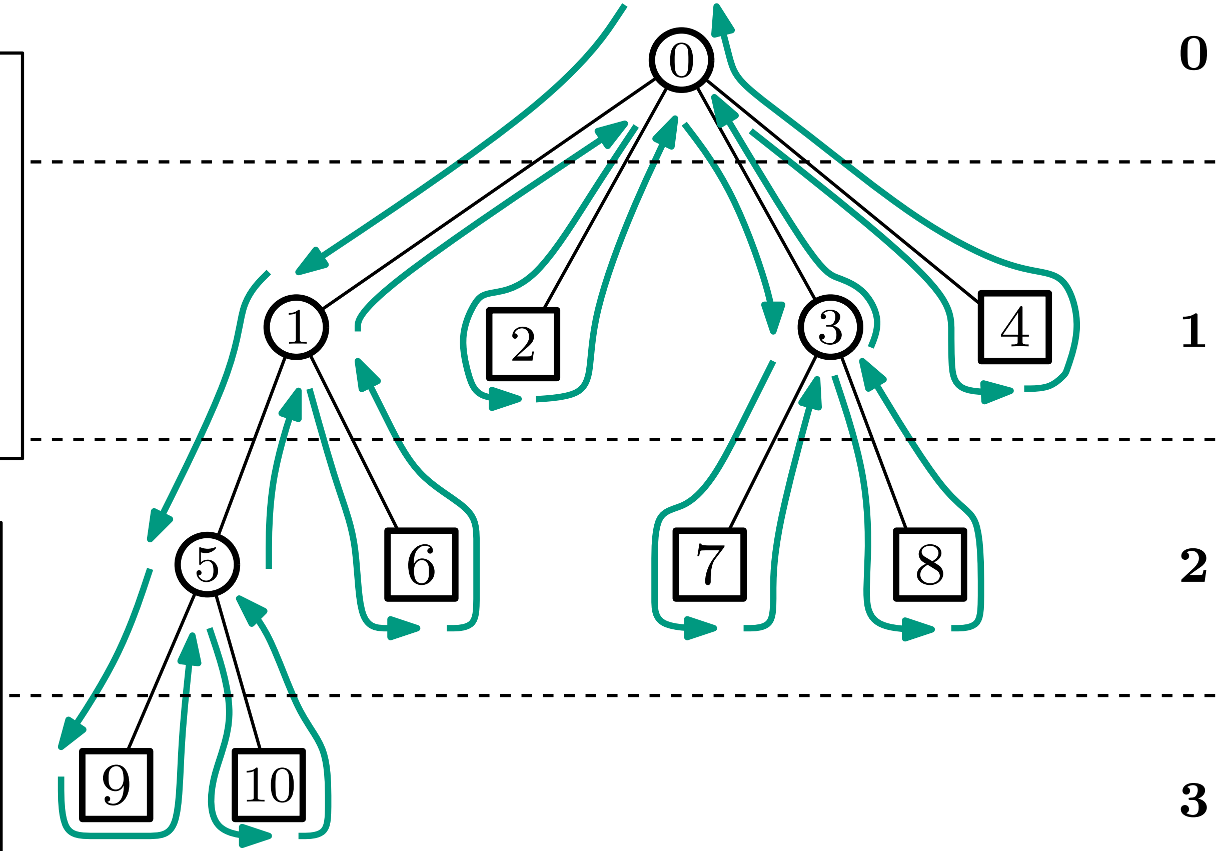
Query Summary - LCA(i,j)

$O(1)$ 1. Find (any) i' st. $N[i'] = i$

$O(1)$ 2. Find (any) j' st. $N[j'] = j$

$O(1)$ 3. Compute RMQ(i', j') in D

$O(1)$ 4. LCA$(i, j) = N[\text{RMQ}(i', j')]$



(node) N	0	1	5	9	5	10	5	1	6	1	0	2	0	3	7	3	8	3	0	4	0

$2n-1$

(depth) D	**0**	**1**	**2**	**3**	**2**	**3**	**2**	**1**	**2**	**1**	**0**	**1**	**0**	**1**	**2**	**1**	**2**	**1**	**0**	**1**	**0**

This gives us $O(n)$ space, $O(n)$ prep. time and $O(1)$ query time for the LCA problem

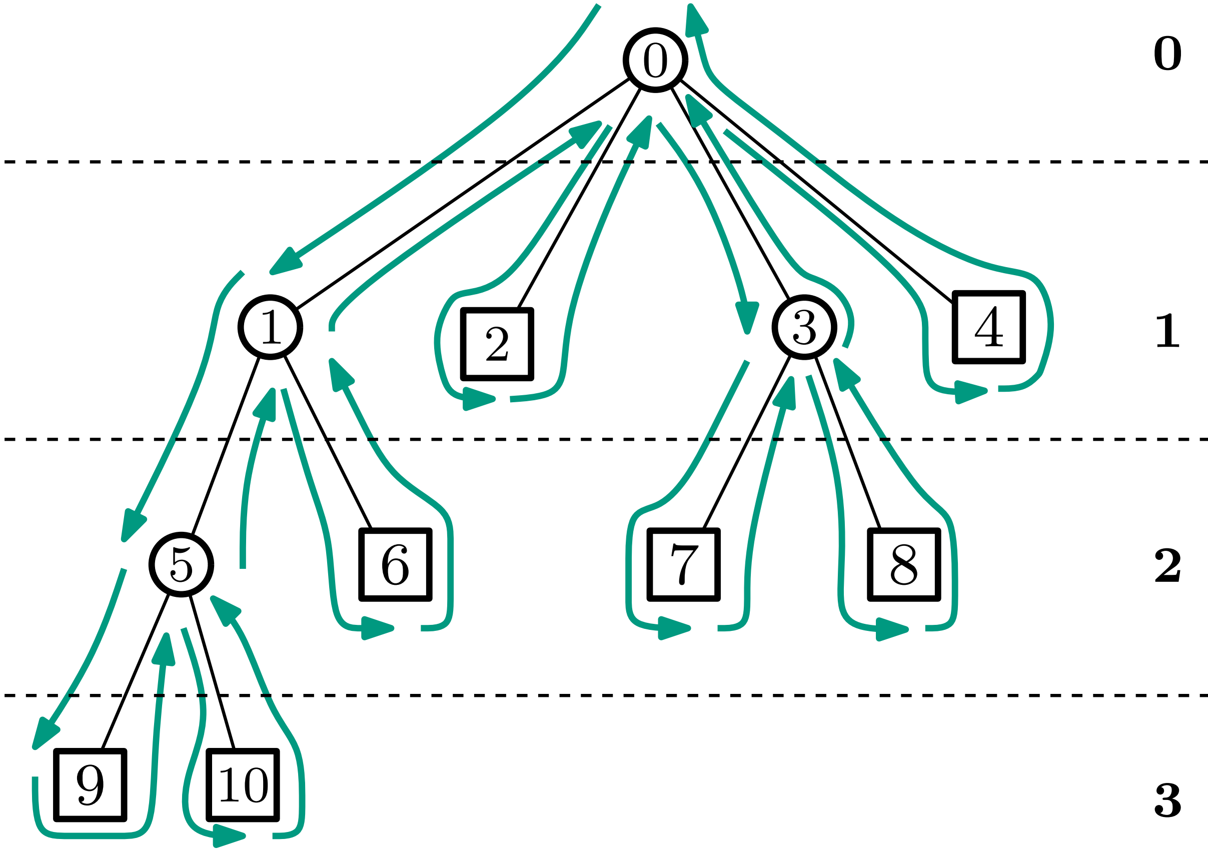
Solving LCAs using RMQs

Preprocessing Summary

$O(n)$ 1. Construct N and D from T

$O(n)$ 2. Add a pointer from each node i to some $N[i'] = i$

$O(n)$ 3. Preprocess D for RMQs



Query Summary - LCA(i,j)

$O(1)$ 1. Find (any) i' st. $N[i'] = i$

$O(1)$ 2. Find (any) j' st. $N[j'] = j$

$O(1)$ 3. Compute RMQ(i', j') in D

$O(1)$ 4. LCA$(i, j) = N[$RMQ$(i', j')]$

(node) N	0	1	5	9	5	10	5	1	6	1	0	2	0	3	7	3	8	3	0	4	0
(depth) D	**0**	**1**	**2**	**3**	**2**	**3**	**2**	**1**	**2**	**1**	**0**	**1**	**0**	**1**	**2**	**1**	**2**	**1**	**0**	**1**	**0**

$\longleftarrow 2n-1 \longrightarrow$

This gives us $O(n)$ space, $O(n)$ prep. time and $O(1)$ query time for the LCA problem

by using the solution to ±1RMQ

Ongoing Summary

We have seen an $O(n)$ space, $O(n)$ prep. time and $O(1)$ query time solution

for the ± 1 Range Minimum Query problem

which improves solution 3 for RMQ *from last lecture*

(but only for ± 1 inputs)

We have seen an $O(n)$ space, $O(n)$ prep. time and $O(1)$ query time solution

for the Lowest Common Ancestor problem

which uses the solution to ± 1RMQ

Ongoing Summary

We have seen an $O(n)$ space, $O(n)$ prep. time and $O(1)$ query time solution

for the ± 1 Range Minimum Query problem

which improves solution 3 for RMQ *from last lecture*

(but only for ± 1 inputs)

We have seen an $O(n)$ space, $O(n)$ prep. time and $O(1)$ query time solution

for the Lowest Common Ancestor problem

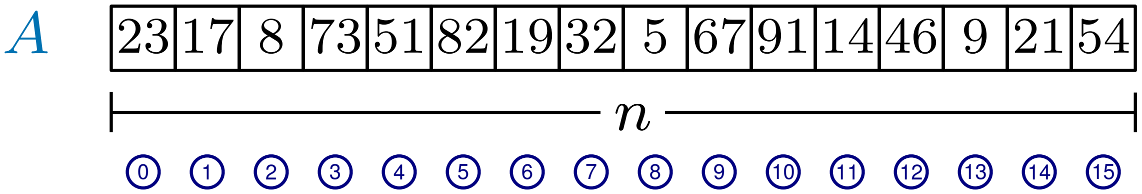
which uses the solution to ± 1RMQ

What about the general Range Minimum Query problem?

(when the inputs aren't ± 1)

Solving RMQs using LCAs

Build the Cartesian tree, T_A of the array A:

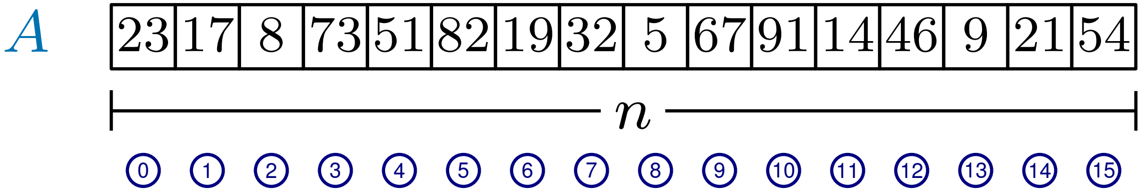


A															
23	17	8	73	51	82	19	32	5	67	91	14	46	9	21	54
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Solving RMQs using LCAs

Build the Cartesian tree, T_A of the array A:

- The root is the smallest value

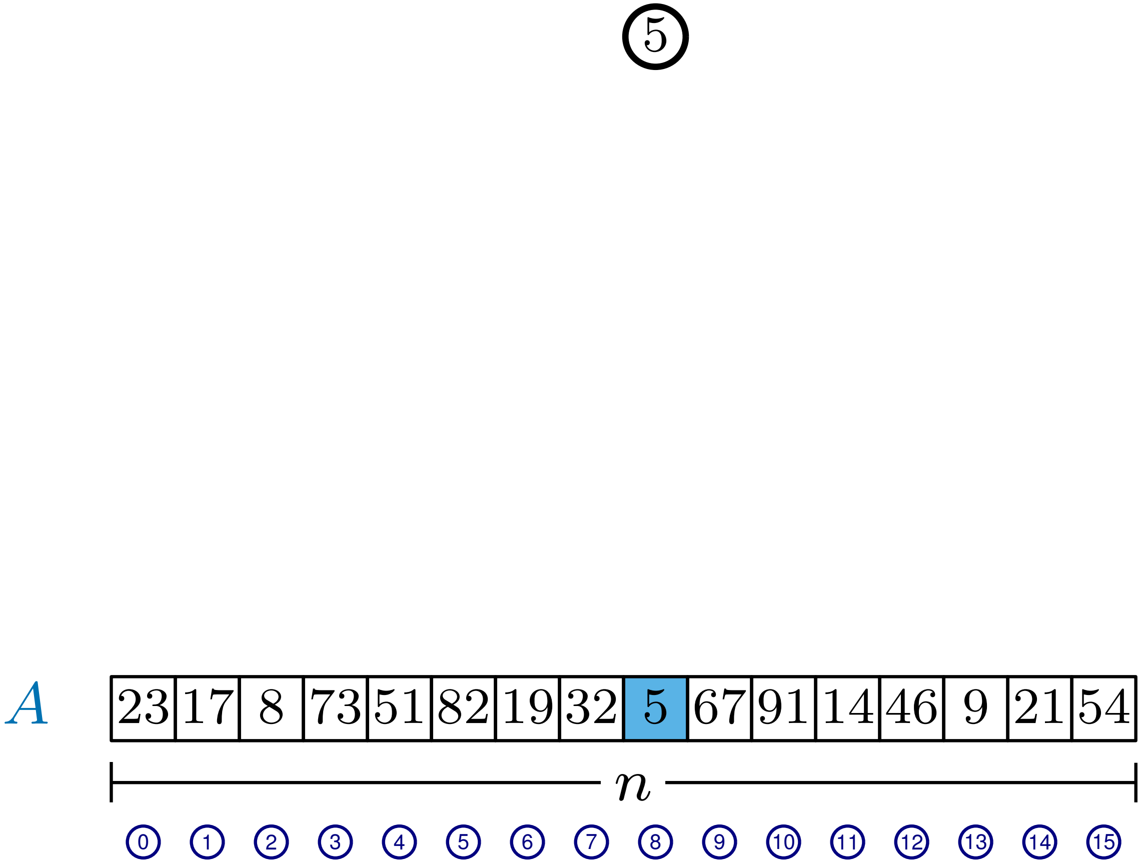


A	23	17	8	73	51	82	19	32	5	67	91	14	46	9	21	54
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Solving RMQs using LCAs

Build the Cartesian tree, T_A of the array A:

- The root is the smallest value

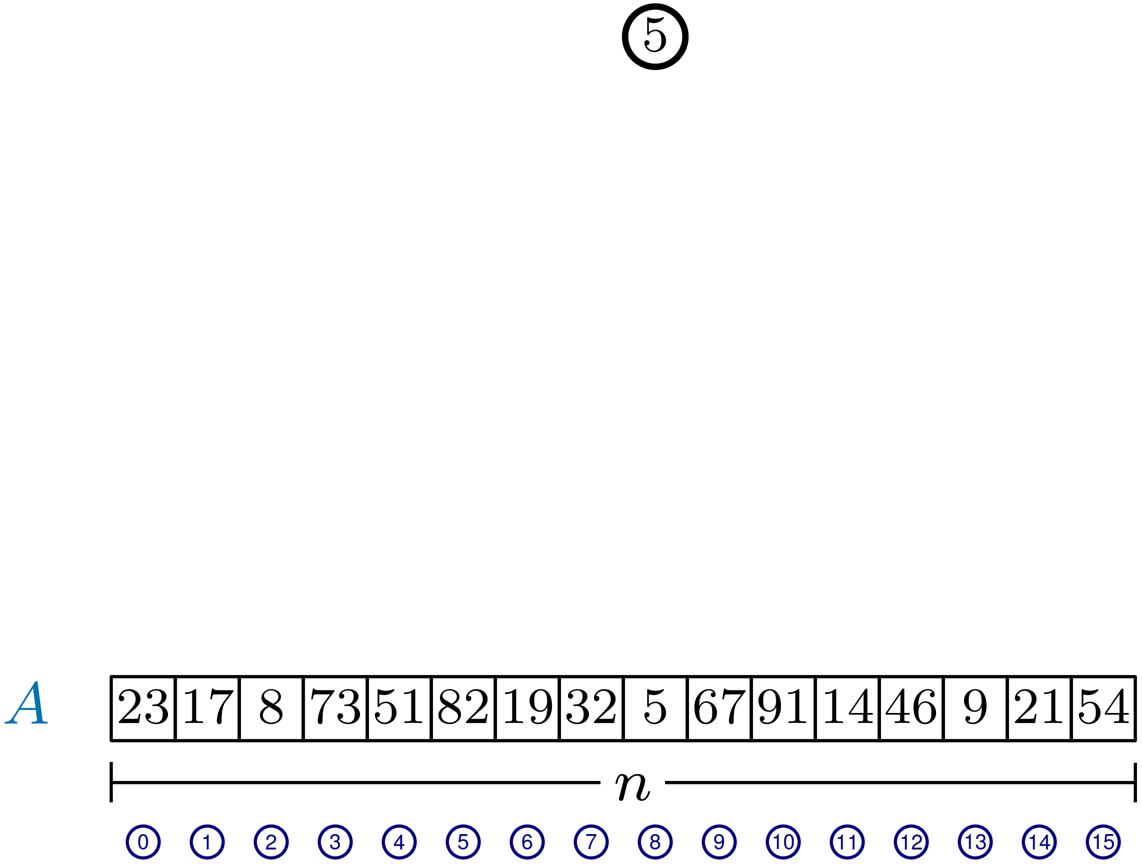


A	23	17	8	73	51	82	19	32	5	67	91	14	46	9	21	54
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Solving RMQs using LCAs

Build the Cartesian tree, T_A of the array A:

- The root is the smallest value
- The selected location
  partitions the array in two

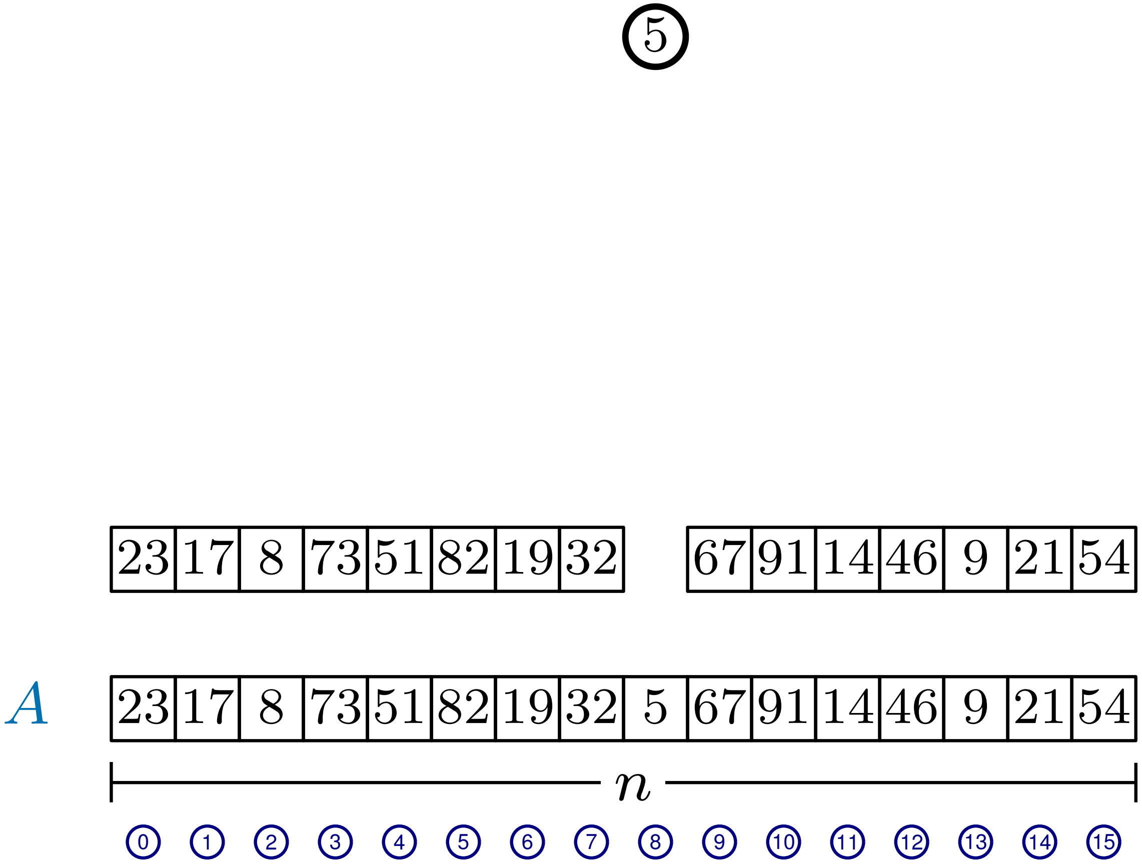


0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
23	17	8	73	51	82	19	32	5	67	91	14	46	9	21	54

Solving RMQs using LCAs

Build the Cartesian tree, T_A of the array A:

- The root is the smallest value
- The selected location partitions the array in two



5

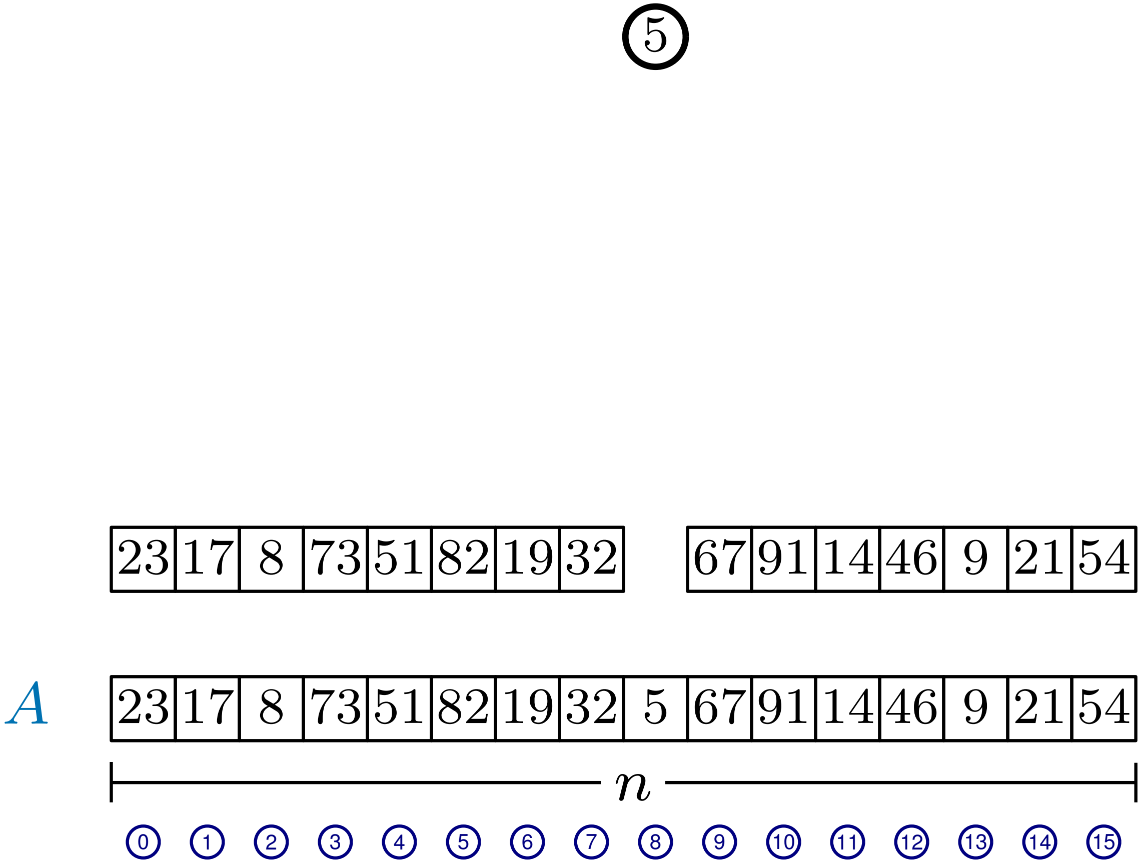
	23	17	8	73	51	82	19	32		67	91	14	46	9	21	54
A	23	17	8	73	51	82	19	32	5	67	91	14	46	9	21	54
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

n

Solving RMQs using LCAs

Build the Cartesian tree, T_A of the array A:

- The root is the smallest value
- The selected location partitions the array in two
- The rest of the tree is given by recursing left and right. . .



23	17	8	73	51	82	19	32

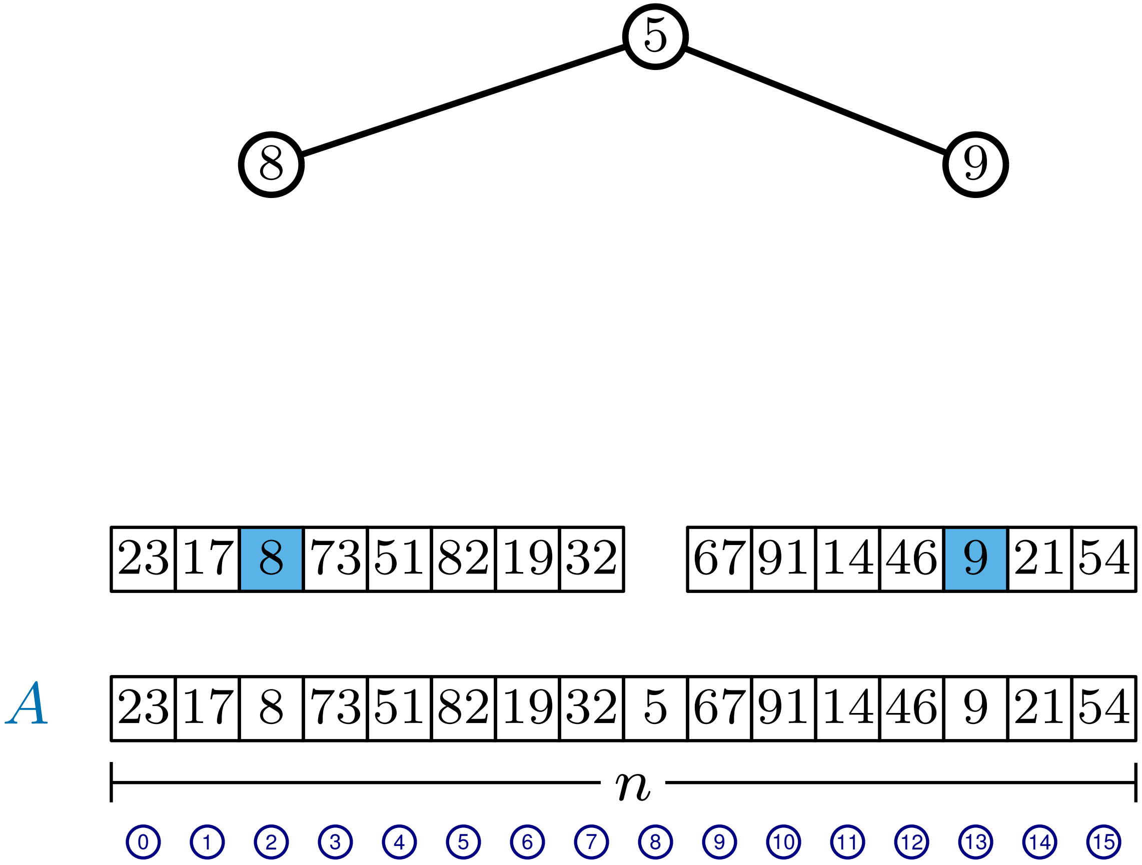
67	91	14	46	9	21	54

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
23	17	8	73	51	82	19	32	5	67	91	14	46	9	21	54

Solving RMQs using LCAs

Build the Cartesian tree, T_A of the array A:

- The root is the smallest value
- The selected location partitions the array in two
- The rest of the tree is given by recursing left and right. . .



23	17	8	73	51	82	19	32

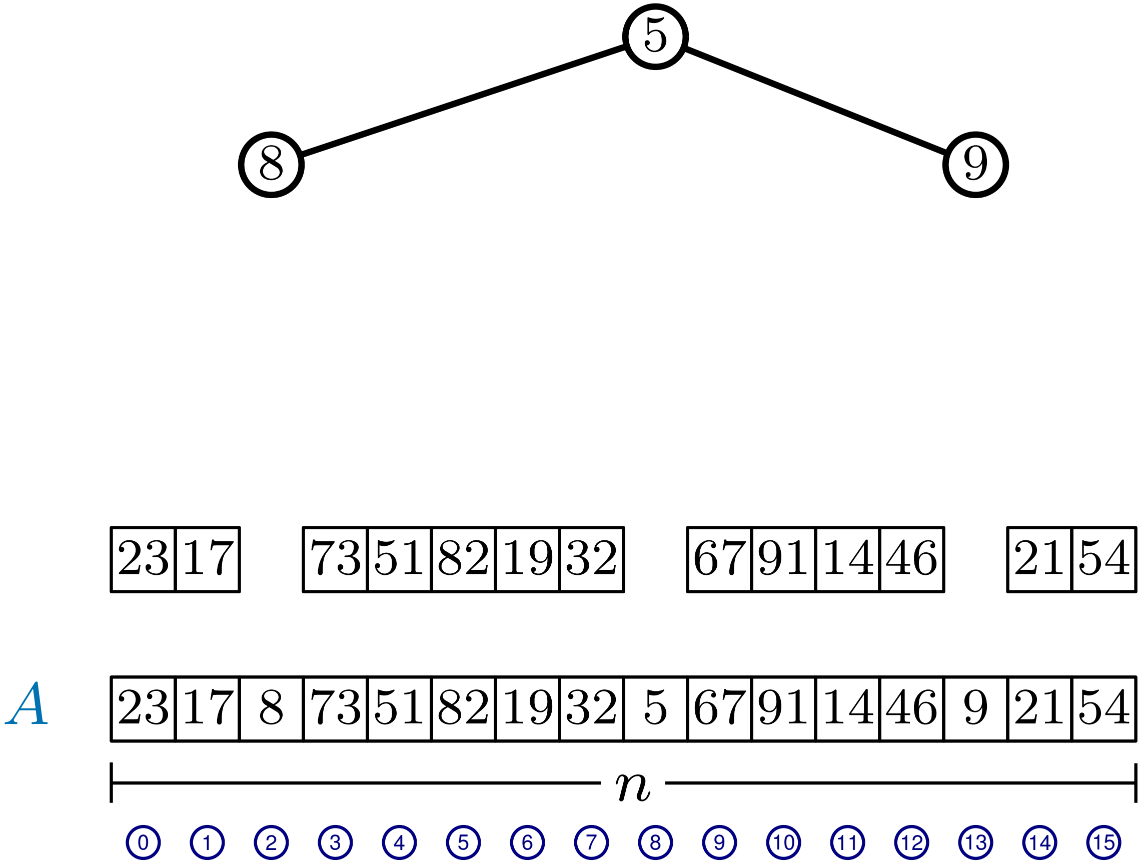
67	91	14	46	9	21	54

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
23	17	8	73	51	82	19	32	5	67	91	14	46	9	21	54

Solving RMQs using LCAs

Build the Cartesian tree, T_A of the array A:

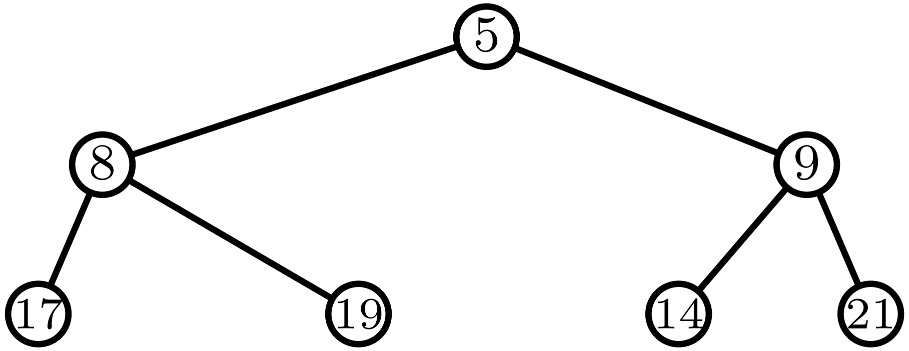
- The root is the smallest value
- The selected location partitions the array in two
- The rest of the tree is given by recursing left and right. . .



Solving RMQs using LCAs

Build the Cartesian tree, T_A of the array A:

- The root is the smallest value
- The selected location partitions the array in two
- The rest of the tree is given by recursing left and right. . .



23	**17**	73	51	82	**19**	32	67	91	**14**	46	**21**	54

A

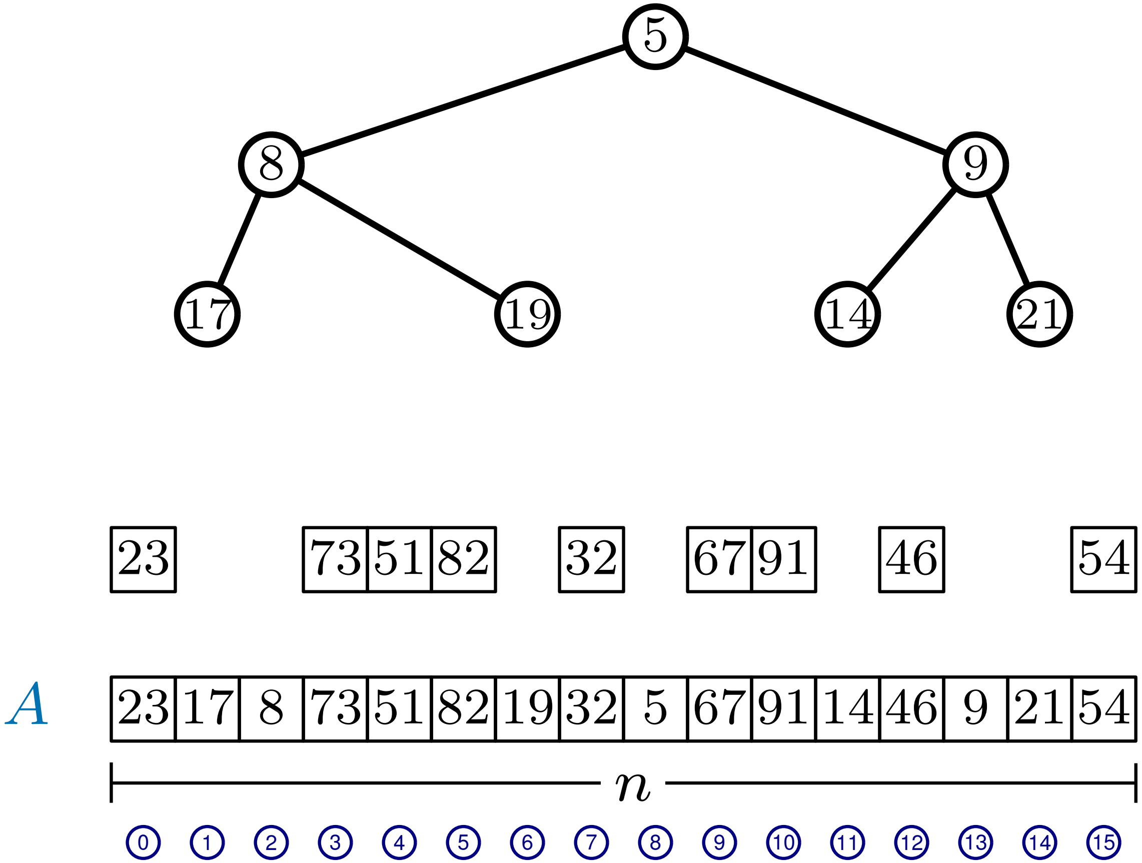
23	17	8	73	51	82	19	32	5	67	91	14	46	9	21	54
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

n

Solving RMQs using LCAs

Build the Cartesian tree, T_A of the array A:

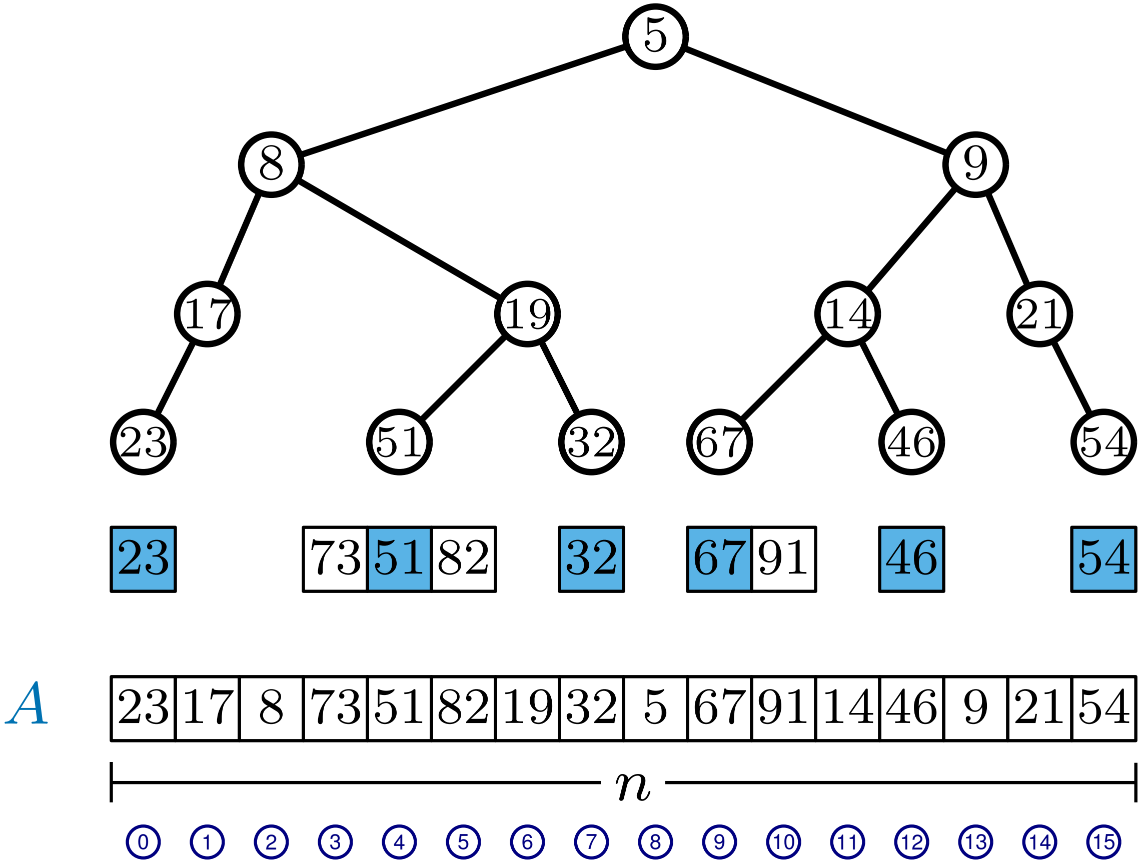
- The root is the smallest value
- The selected location partitions the array in two
- The rest of the tree is given by recursing left and right. . .



Solving RMQs using LCAs

Build the Cartesian tree, T_A of the array A:

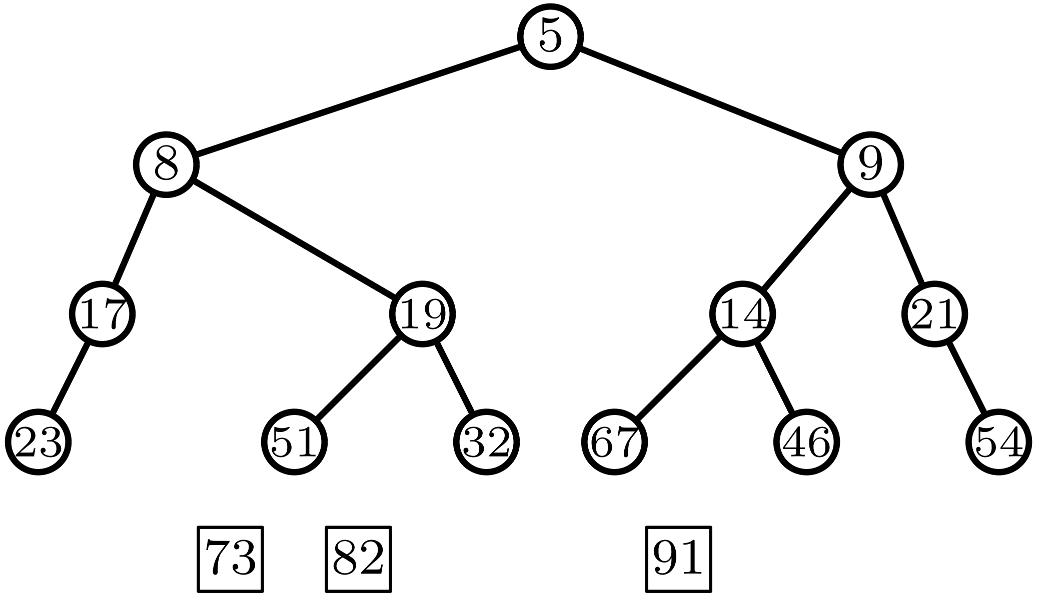
- The root is the smallest value
- The selected location partitions the array in two
- The rest of the tree is given by recursing left and right. . .



Solving RMQs using LCAs

Build the Cartesian tree, T_A of the array A:

- The root is the smallest value
- The selected location partitions the array in two
- The rest of the tree is given by recursing left and right. . .



A

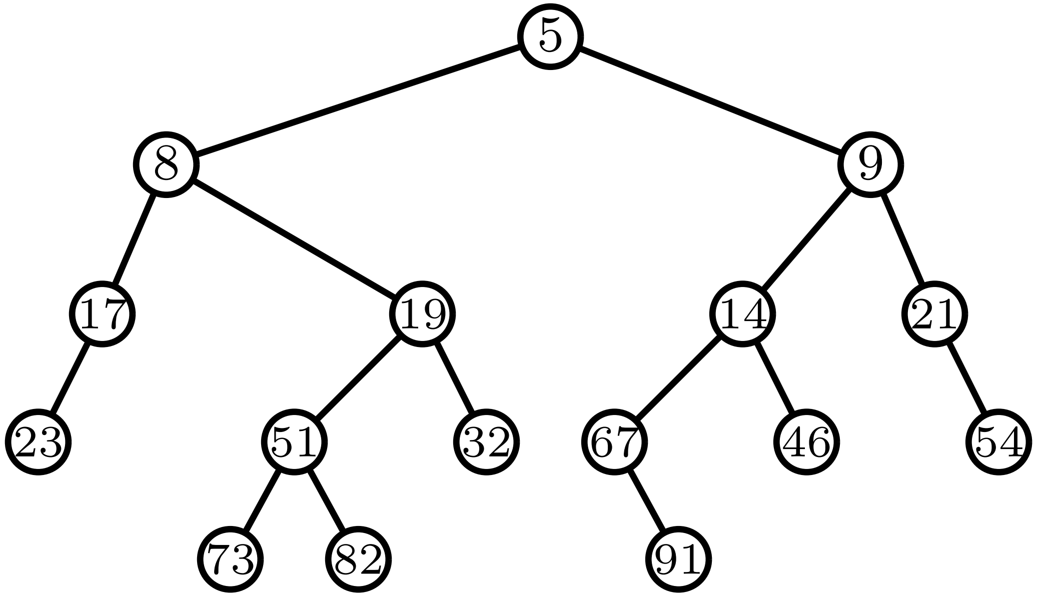
A	23	17	8	73	51	82	19	32	5	67	91	14	46	9	21	54
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

$\underbrace{\hspace{10em}}_{n}$

Solving RMQs using LCAs

Build the Cartesian tree, T_A of the array A:

- The root is the smallest value
- The selected location partitions the array in two
- The rest of the tree is given by recursing left and right. . .



A	23	17	8	73	51	82	19	32	5	67	91	14	46	9	21	54
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

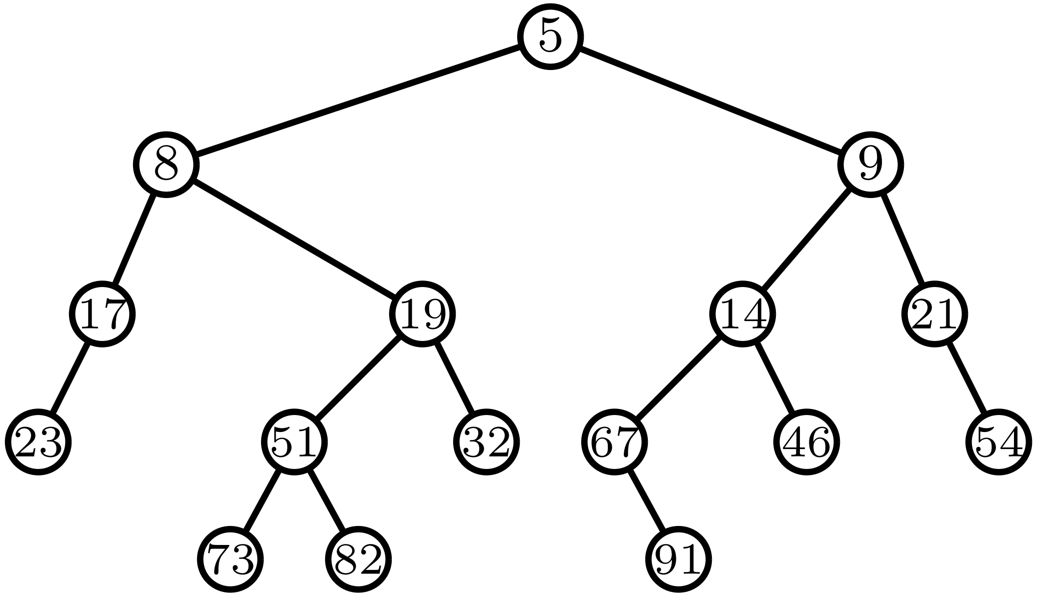
n

Solving RMQs using LCAs

Build the Cartesian tree, T_A of the array A:

- The root is the smallest value
- The selected location partitions the array in two
- The rest of the tree is given by recursing left and right. . .

This process isn't very efficient. . .
a better one takes $O(n)$ time



A	23	17	8	73	51	82	19	32	5	67	91	14	46	9	21	54
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

n

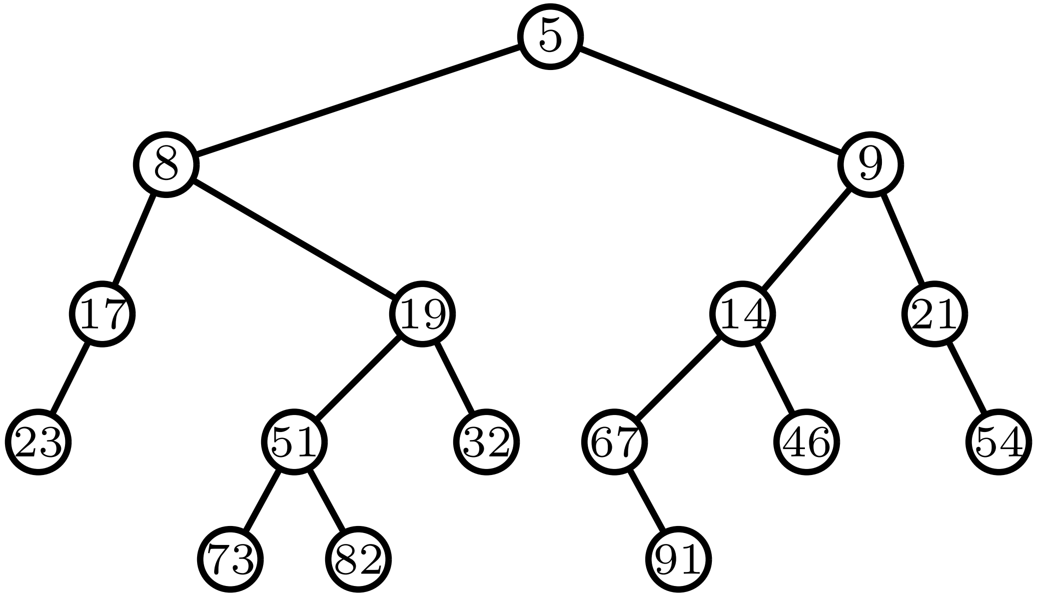
Solving RMQs using LCAs

Build the Cartesian tree, T_A of the array A:

- The root is the smallest value
- The selected location partitions the array in two
- The rest of the tree is given by recursing left and right...

This process isn't very efficient...
a better one takes $O(n)$ time

it's not tricky but we don't have time to cover it



A

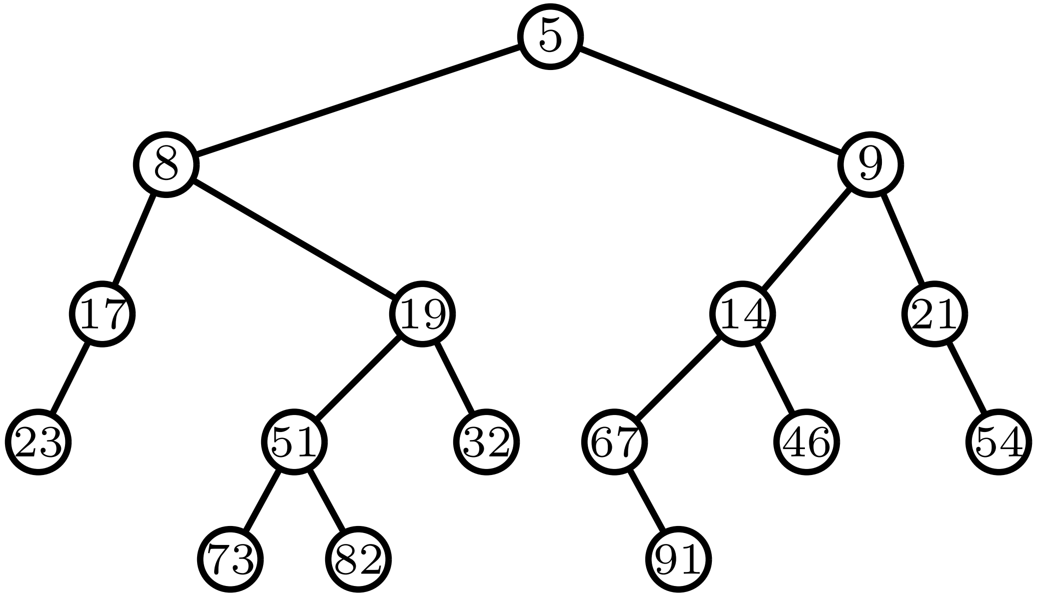
23	17	8	73	51	82	19	32	5	67	91	14	46	9	21	54
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

n

Solving RMQs using LCAs

Build the Cartesian tree, T_A of the array A:

- The root is the smallest value
- The selected location partitions the array in two
- The rest of the tree is given by recursing left and right. . .



This process isn't very efficient. . .
a better one takes $O(n)$ time

it's not tricky but we don't have time to cover it

A																
	23	17	8	73	51	82	19	32	5	67	91	14	46	9	21	54
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

n

Key Fact: The LCA in T_A equals the RMQ in A

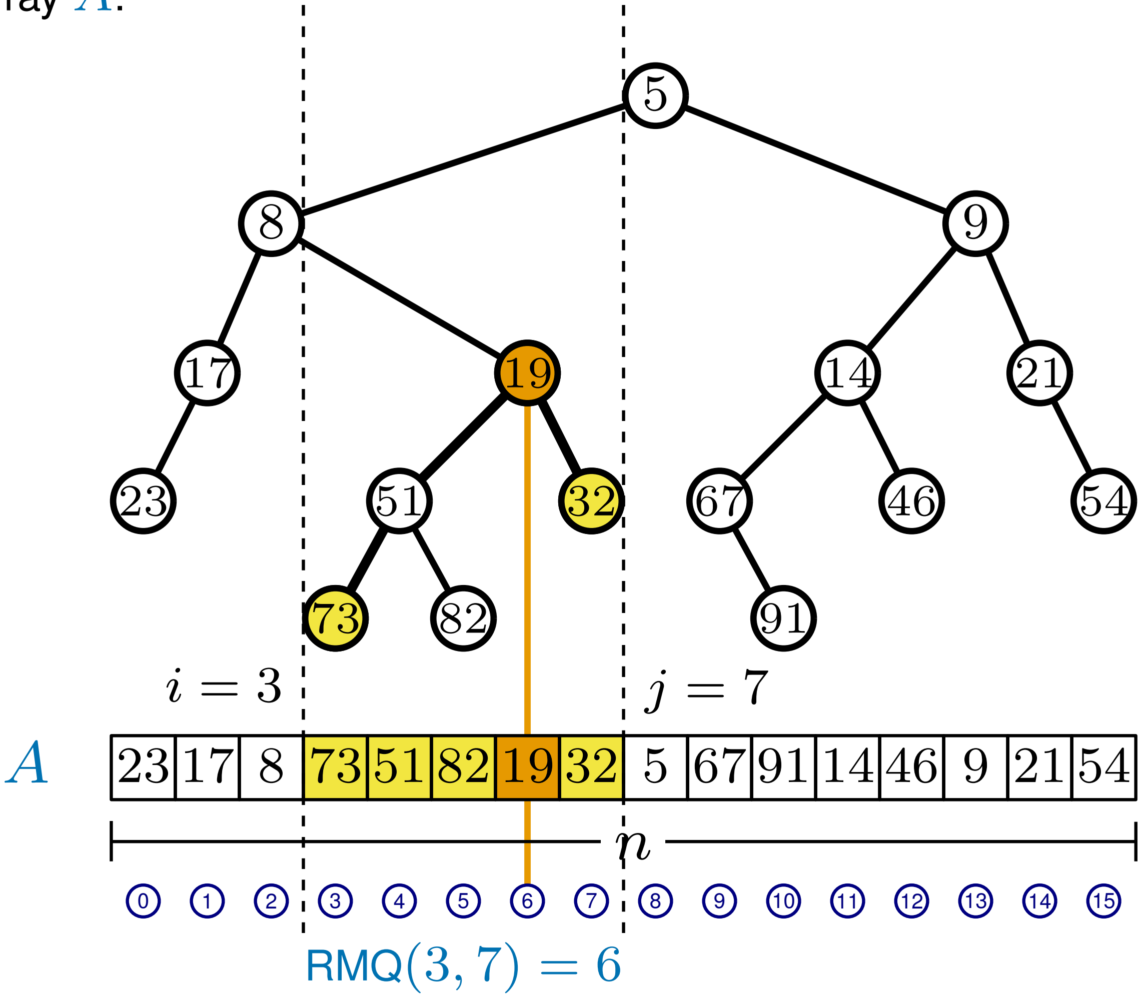
Solving RMQs using LCAs

Build the Cartesian tree, T_A of the array A:

- The root is the smallest value
- The selected location partitions the array in two
- The rest of the tree is given by recursing left and right. . .

This process isn't very efficient. . .
a better one takes $O(n)$ time

it's not tricky but we don't have time to cover it



Key Fact: The LCA in T_A equals the RMQ in A

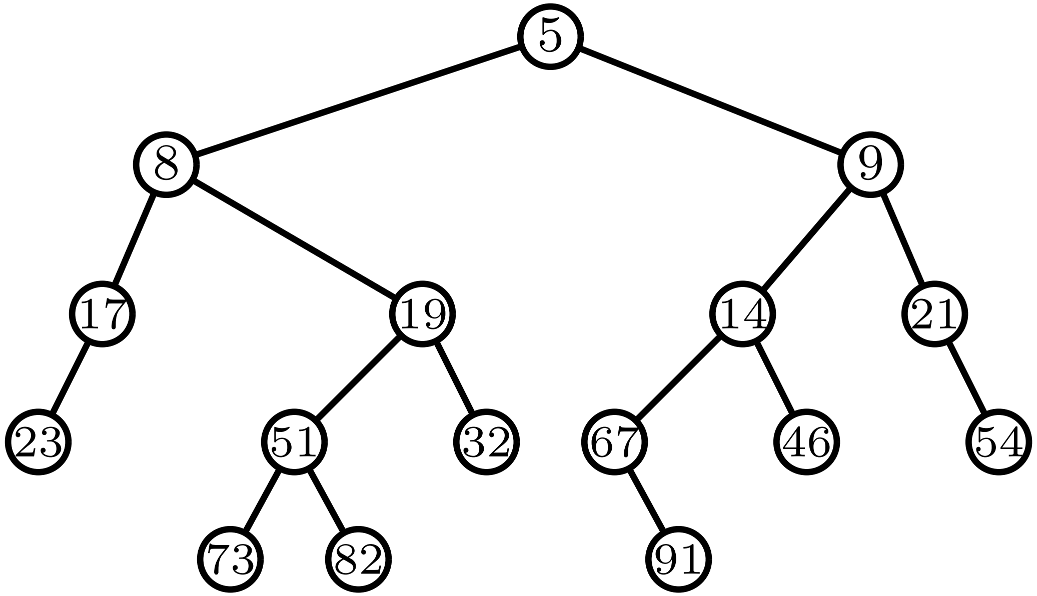
Solving RMQs using LCAs

Build the Cartesian tree, T_A of the array A:

- The root is the smallest value
- The selected location partitions the array in two
- The rest of the tree is given by recursing left and right. . .

This process isn't very efficient. . . a better one takes $O(n)$ time

it's not tricky but we don't have time to cover it



A	23	17	8	73	51	82	19	32	5	67	91	14	46	9	21	54
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

n

Key Fact: The LCA in T_A equals the RMQ in A

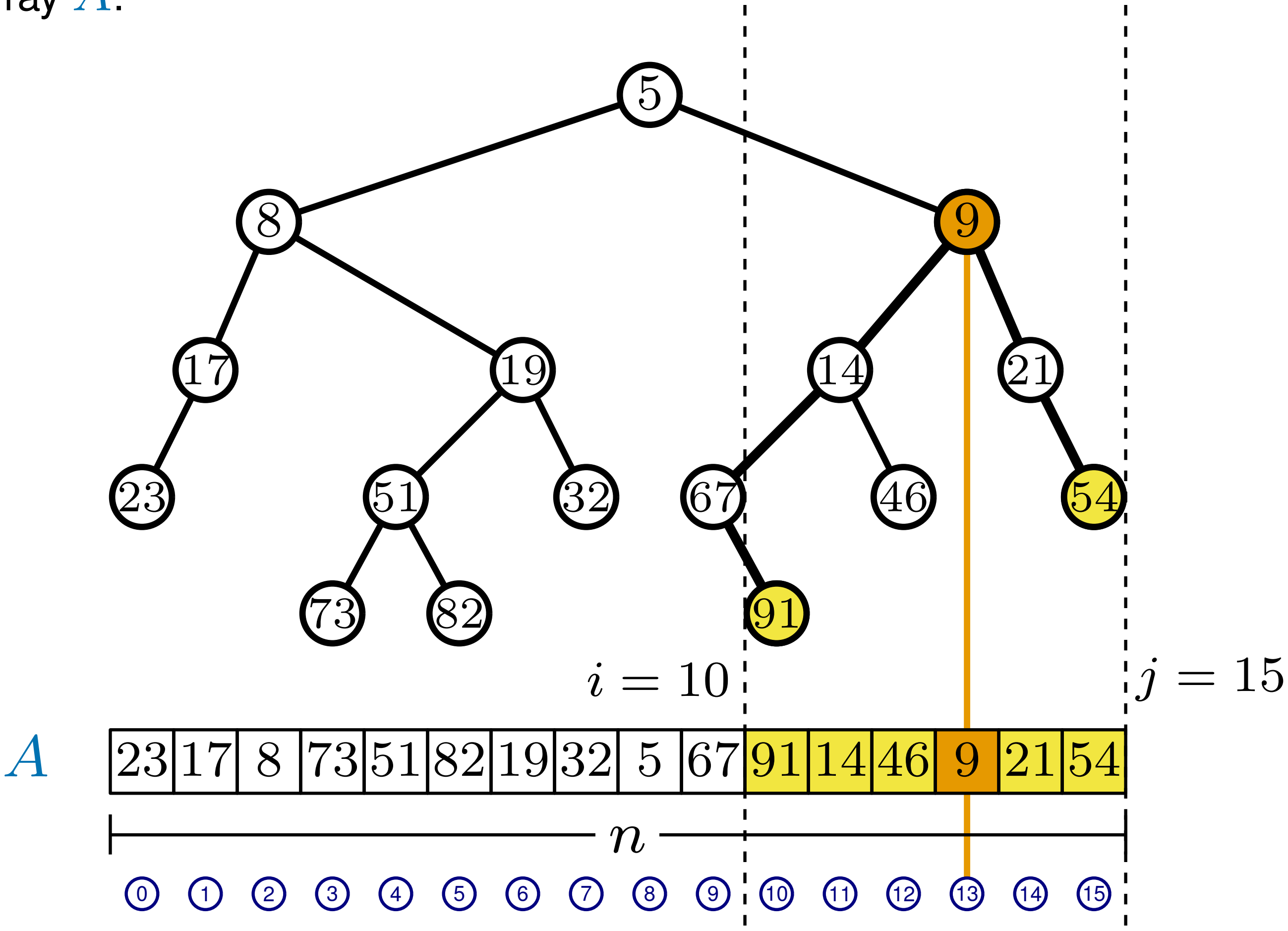
Solving RMQs using LCAs

Build the Cartesian tree, T_A of the array A:

- The root is the smallest value
- The selected location partitions the array in two
- The rest of the tree is given by recursing left and right. . .

This process isn't very efficient. . .
a better one takes $O(n)$ time

it's not tricky but we don't have time to cover it



A	23	17	8	73	51	82	19	32	5	67	91	14	46	9	21	54
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Key Fact: The LCA in T_A equals the RMQ in A

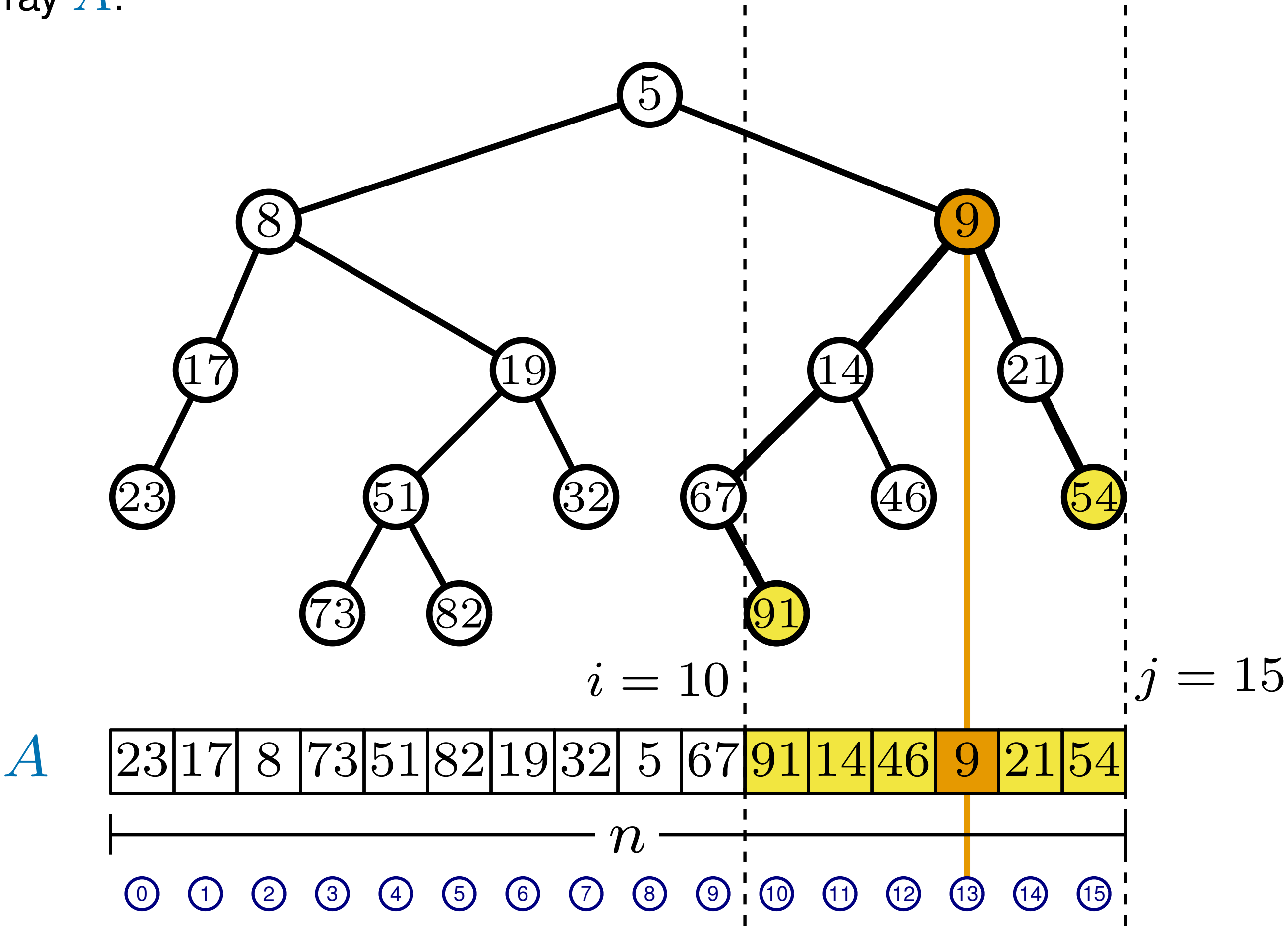
Solving RMQs using LCAs

Build the Cartesian tree, T_A of the array A:

- The root is the smallest value
- The selected location partitions the array in two
- The rest of the tree is given by recursing left and right...

This process isn't very efficient...
a better one takes $O(n)$ time

it's not tricky but we don't have time to cover it



Key Fact: The LCA in T_A equals the RMQ in A

This gives us $O(n)$ space, $O(n)$ prep. time and $O(1)$ query time for the RMQ problem

Solving RMQs using LCAs

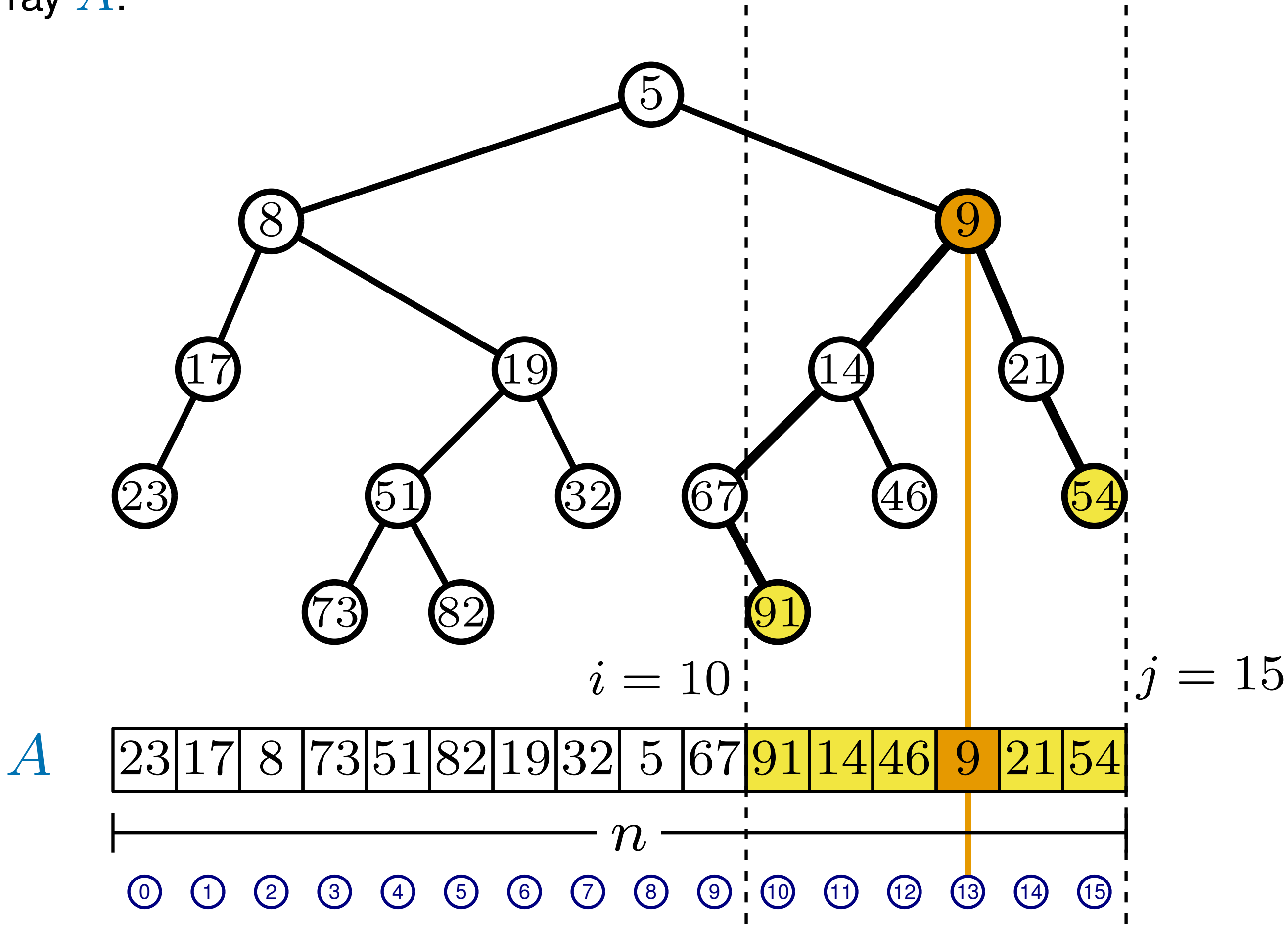
Build the Cartesian tree, T_A of the array A:

- The root is the smallest value
- The selected location partitions the array in two
- The rest of the tree is given by recursing left and right...

This process isn't very efficient...

a better one takes $O(n)$ time

it's not tricky but we don't have time to cover it



Key Fact: The LCA in T_A equals the RMQ in A

This gives us $O(n)$ space, $O(n)$ prep. time and $O(1)$ query time for the RMQ problem

by using the solution to LCA :)

Summary

We have seen an $O(n)$ space, $O(n)$ prep. time and $O(1)$ query time solution

for the ± 1 Range Minimum Query problem

which improves solution 3 for RMQ *from last lecture*

(but only for ± 1 inputs)

We have seen an $O(n)$ space, $O(n)$ prep. time and $O(1)$ query time solution

for the Lowest Common Ancestor problem

which uses the solution to ± 1RMQ

We have seen an $O(n)$ space, $O(n)$ prep. time and $O(1)$ query time solution

for the Range Minimum Query problem

which uses the solution to LCA

(which works for all inputs)