Prefix Sum
Fenwick Tree

Outline

- Range Sum
  - 1D
- Prefix Sum
  - 1D
  - 2D
- Fenwick Tree (Binary Indexed Tree)
  - 1D

ADT: 1D Range Sum

- Objects: a sequence $S=(s_1,\ldots,s_n)$ of summable elements
- Operations:
  - Create(S,A[1,..,n]): initial a sequence S containing A[1], ..., A[n].
  - Update(S,i,x): update the i^{th} element S to x.
  - RangeSum(S,i,j): return $s_i+s_{i+1}+\ldots+s_j$.

Range Sum

1	2	3	4	5	6	7	8	9
0	1	1	1	0	2	1	1	0

1	2	3	4	5	6	7	8	9	
0	1	2	3	3	5	6	7	7	1
	1	2	3	3	5	6	7	7	2
		1	2	2	4	5	6	6	3
			1	1	3	4	5	5	4
				0	2	3	4	4	5
					2	3	4	4	6
						1	2	2	7
							1	1	8
								0	9

Straightforward: Store all range sums in a 2D table.

Space: $O(n^2)$

Query: $O(1)$

Update?

Range Sum: Update $s_5=1$

1	2	3	4	5	6	7	8	9
0	1	1	1	1	2	1	1	0

1	2	3	4	5	6	7	8	9	
0	1	2	3	4	6	7	8	8	1
	1	2	3	4	6	7	8	8	2
		1	2	3	5	6	7	7	3
			1	2	4	5	6	6	4
				1	3	4	5	5	5
					2	3	4	4	6
						1	2	2	7
							1	1	8
								0	9

Straightforward: Updating an entry takes $O(n^2)$ time.

Prefix Sums

- Let $p_0=0$ and $p_i=s_1+s_2+\ldots+s_i$. We say $(p_0,\ldots,p_n)$ is the prefix sums of $(s_1,\ldots,s_n)$.
  - Note: $p_i=p_{i-1}+s_i$. This fact allow us to evaluate $(p_0,\ldots,p_n)$ in $O(n)$.
- Compute $s_i+\ldots+s_j$: use p_j-p_{i-1}.
- Build time: $O(n)$
- Query: $O(1)$
- Update?

Prefix Sum: Update $s_5=1$

	0	1	2	3	4	5	6	7	8	9
A[i]		0	1	1	1	0	2	1	1	0
P[i]	0	0	1	2	3	3	5	6	7	7

	0	1	2	3	4	5	6	7	8	9
A[i]		0	1	1	1	1	2	1	1	0
P[i]	0	0	1	2	3	4	6	7	8	8

Prefix Sums: 2D

- Consider an n-by-n matrix $A=(a_{i,j})$.
- Let $p_{0,0}=0$ and $p_{i,j}=\Sigma_{0<x\leq i}\Sigma_{0<y\leq j}a_{i,j}$ for $i>0$ or $j>0$. We say $P=(p_{i,j})$ is the prefix sums of A.
  - Note: $p_{i,j}=p_{i-1,j}+p_{i,j-1}+a_{i,j}-p_{i-1,j-1}$. This fact allow us to evaluate P in $O(n^2)$.
- How to compute $\Sigma_{L\leq x\leq R}\Sigma_{B\leq y\leq U}a_{i,j}$?
  - $(p_{R,U})-(p_{L-1,U})-(p_{R,B-1})+(p_{L-1,B-1})$

Prefix Sums: 2D

A	1	2	3	4	5
1	0	0	0	0	1
2	0	1	1	0	0
3	0	0	0	2	0
4	1	0	1	0	0
5	0	1	0	0	1

P	1	2	3	4	5
1					
2					
3					
4					
5					

Prefix Sums: 2D

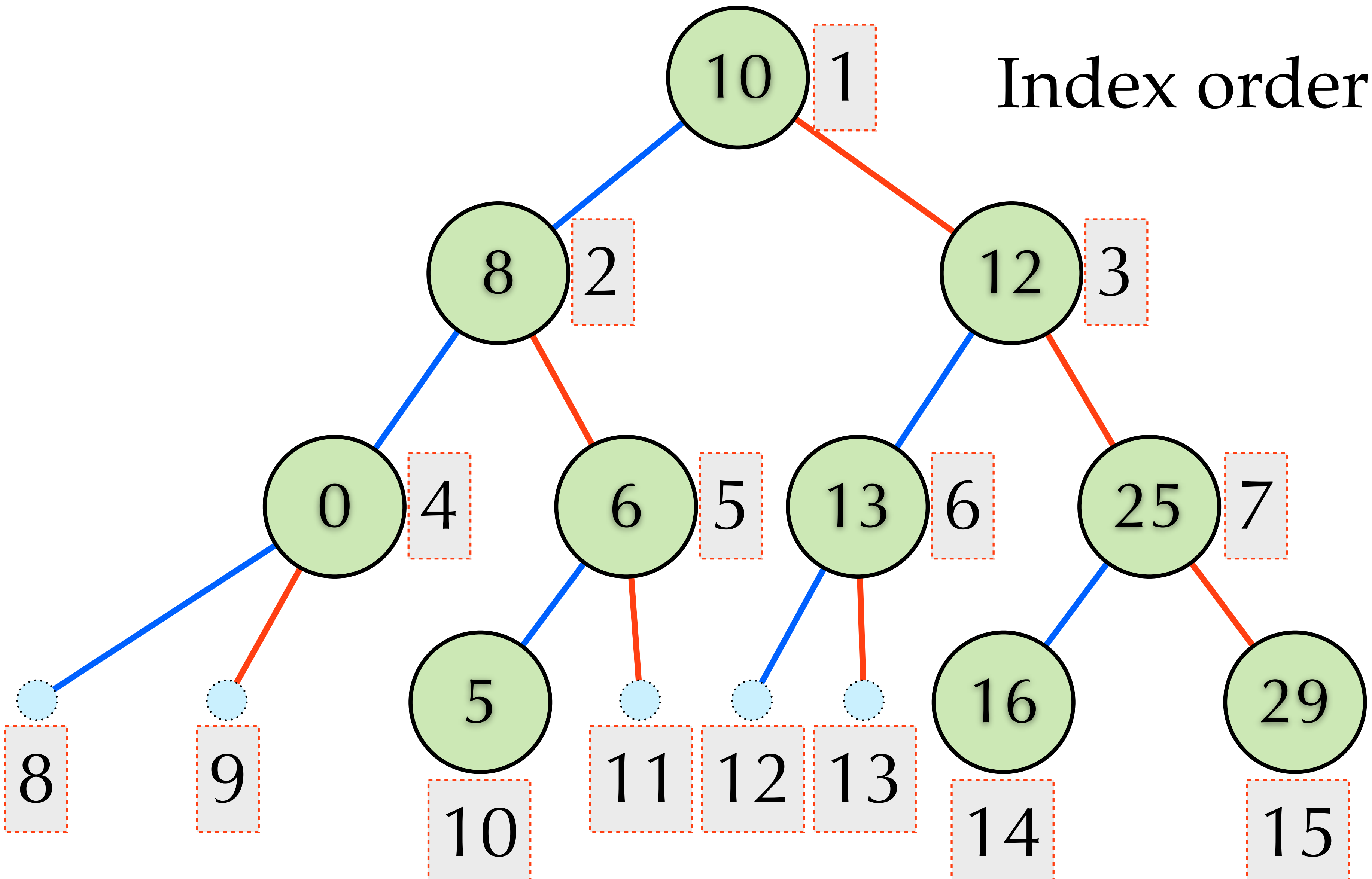
A	1	2	3	4	5
1	0	0	0	0	1
2	0	1	1	0	0
3	0	0	0	2	0
4	1	0	1	0	0
5	0	1	0	0	1

P	1	2	3	4	5
1	0	0	0	0	1
2	0	1	2	2	3
3	0	1	2	4	5
4	1	2	4	6	7
5	1	3	5	7	9

Fenwick Trees

- Provide efficient prefix sum query and update for an array with an easy implementation.
- Initial build: O(nlogn) or O(n)?
- Update s_i: O(logn)
- Prefix Sum $s_1+\ldots+s_i$: O(logn)
- Range Sum $s_i+\ldots+s_j$: O(logn)

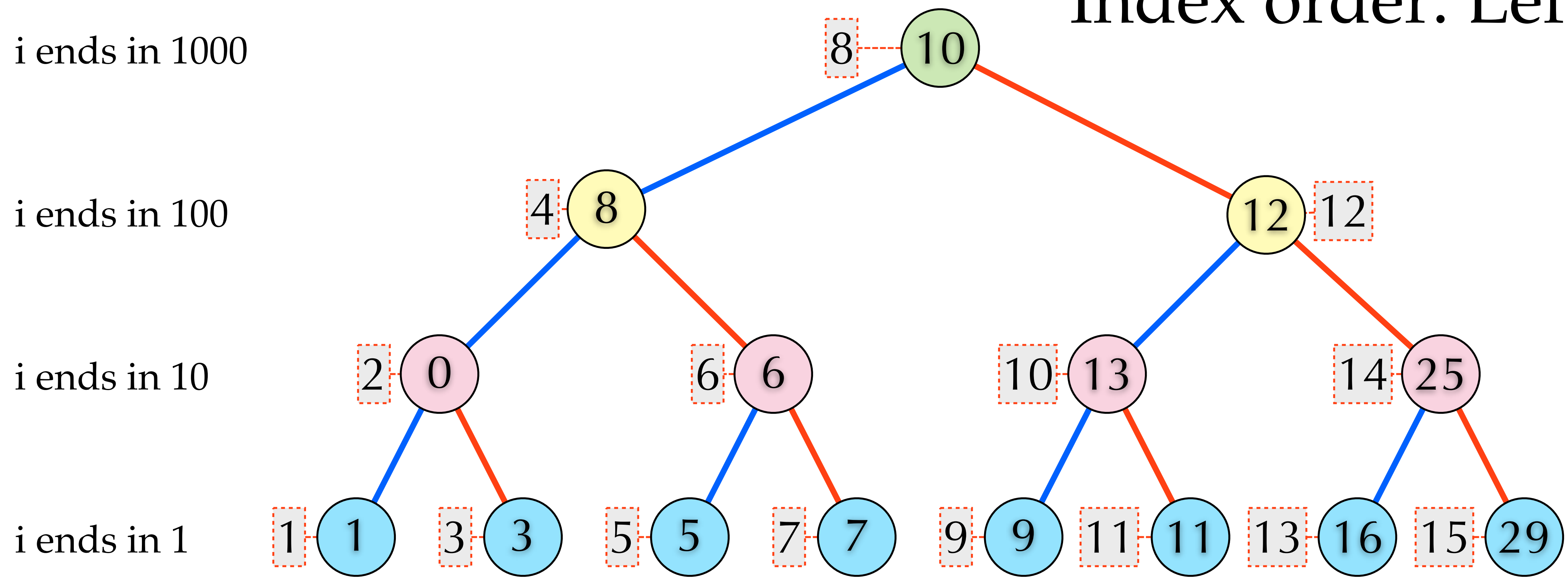
Binary Tree: Array



Index order: Top-down

Fenwick Tree (Binary Indexed Tree)

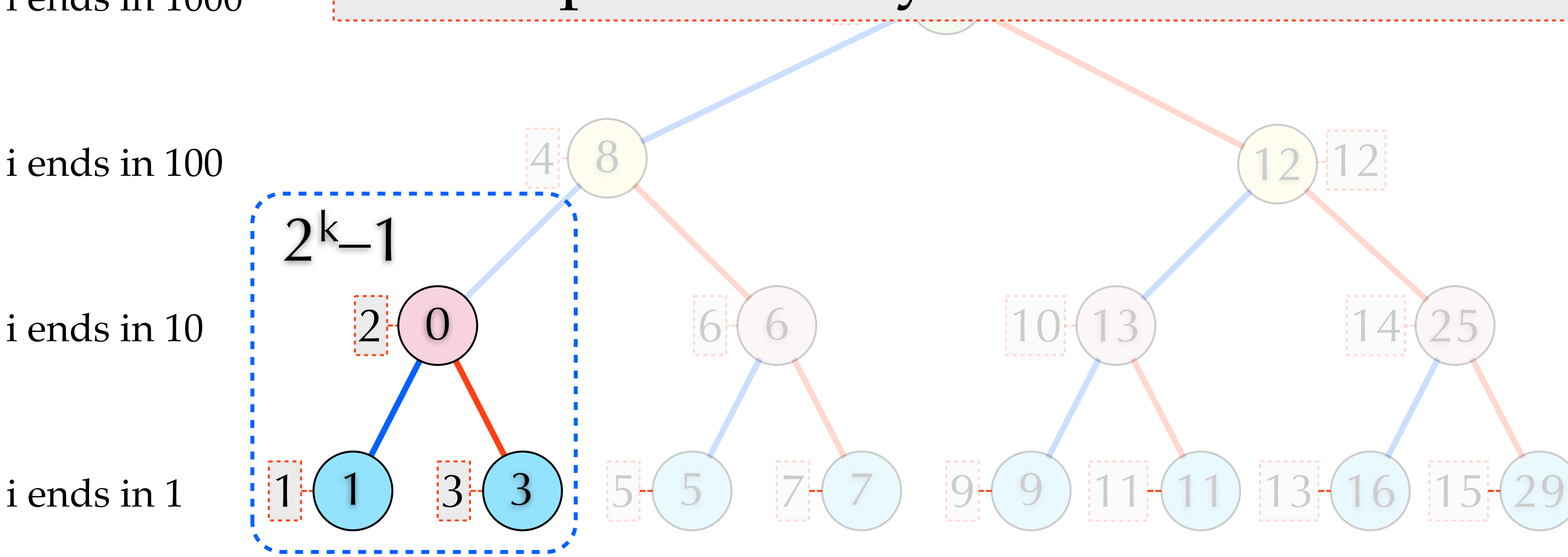
Index order: Left-right



i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
s_i	1	0	3	8	5	6	7	10	9	13	11	12	16	25	29

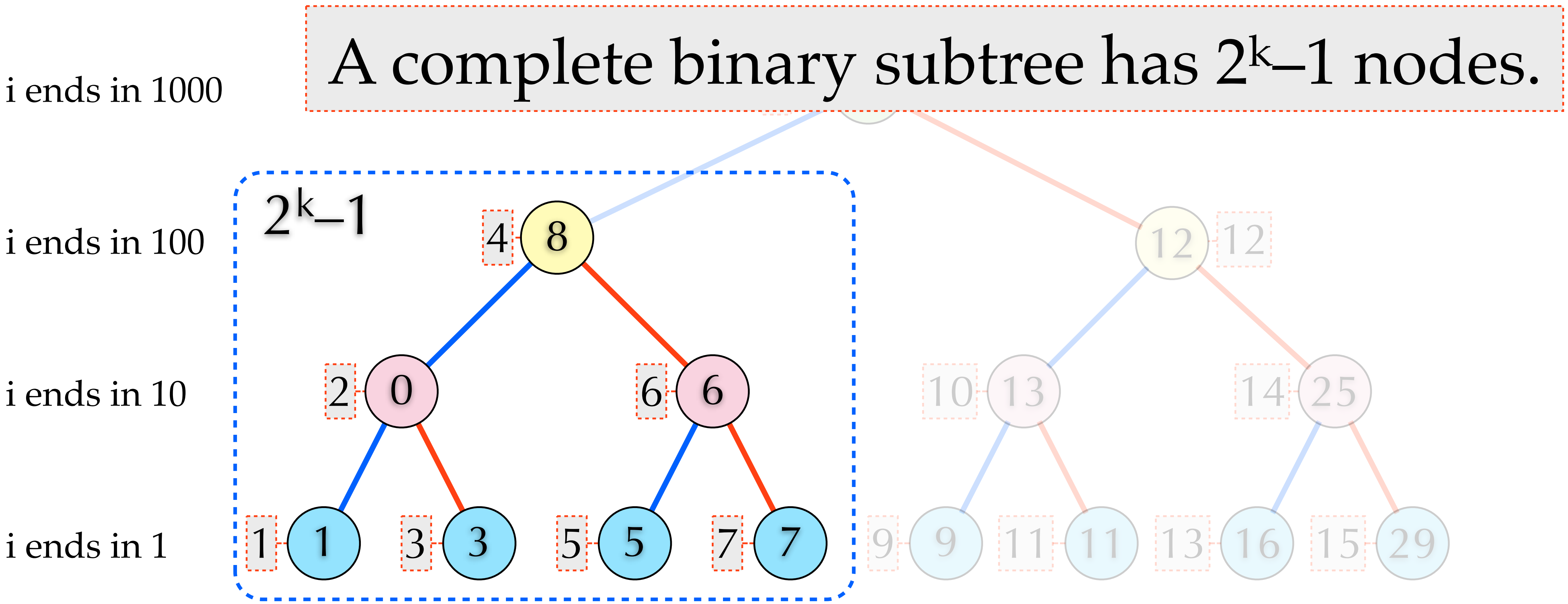
Fenwick Tree

A complete binary subtree has $2^k–1$ nodes.



i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
s_i	1	0	3	8	5	6	7	10	9	13	11	12	16	25	29

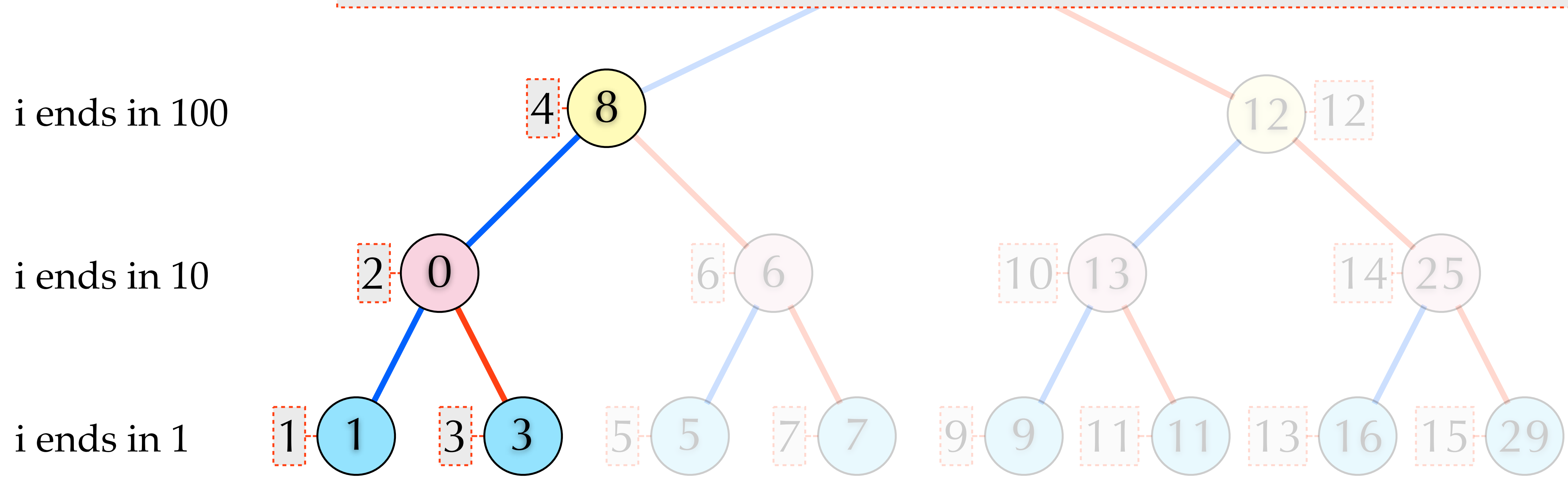
Fenwick Tree



i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
s_i	1	0	3	8	5	6	7	10	9	13	11	12	16	25	29

Fenwick Tree

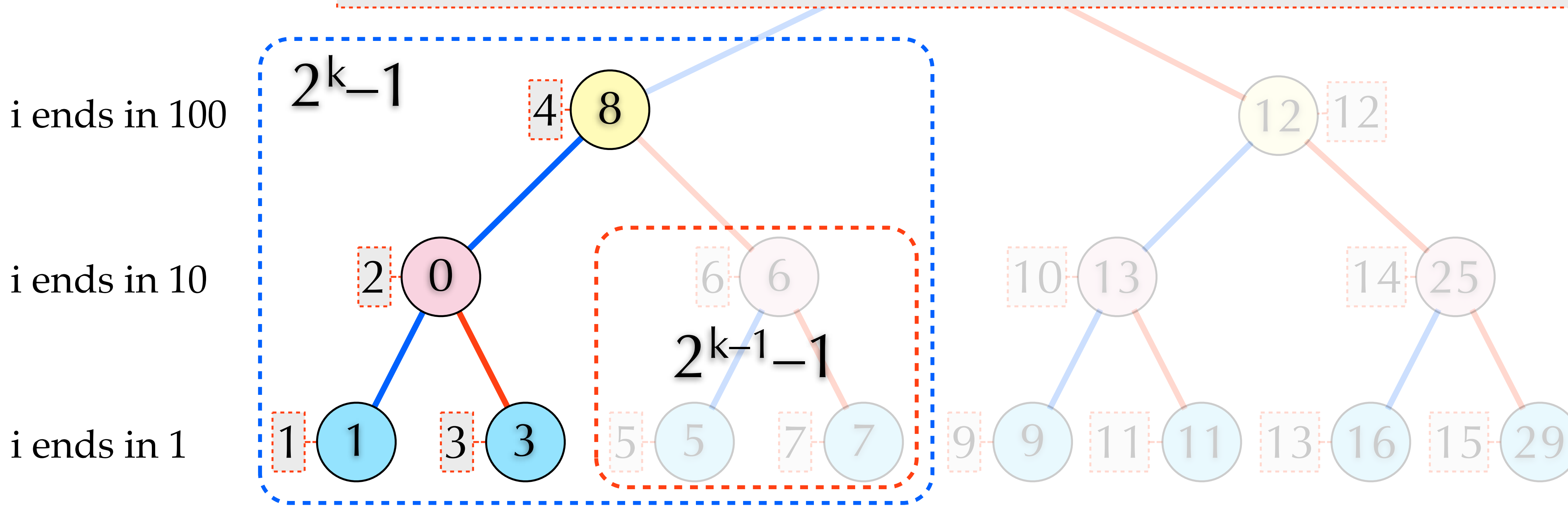
If a complete binary tree has 2^k-1 nodes, then its root has index 2^{k-1} and its left half has 2^{k-1} nodes.



i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
s_i	1	0	3	8	5	6	7	10	9	13	11	12	16	25	29

Fenwick Tree

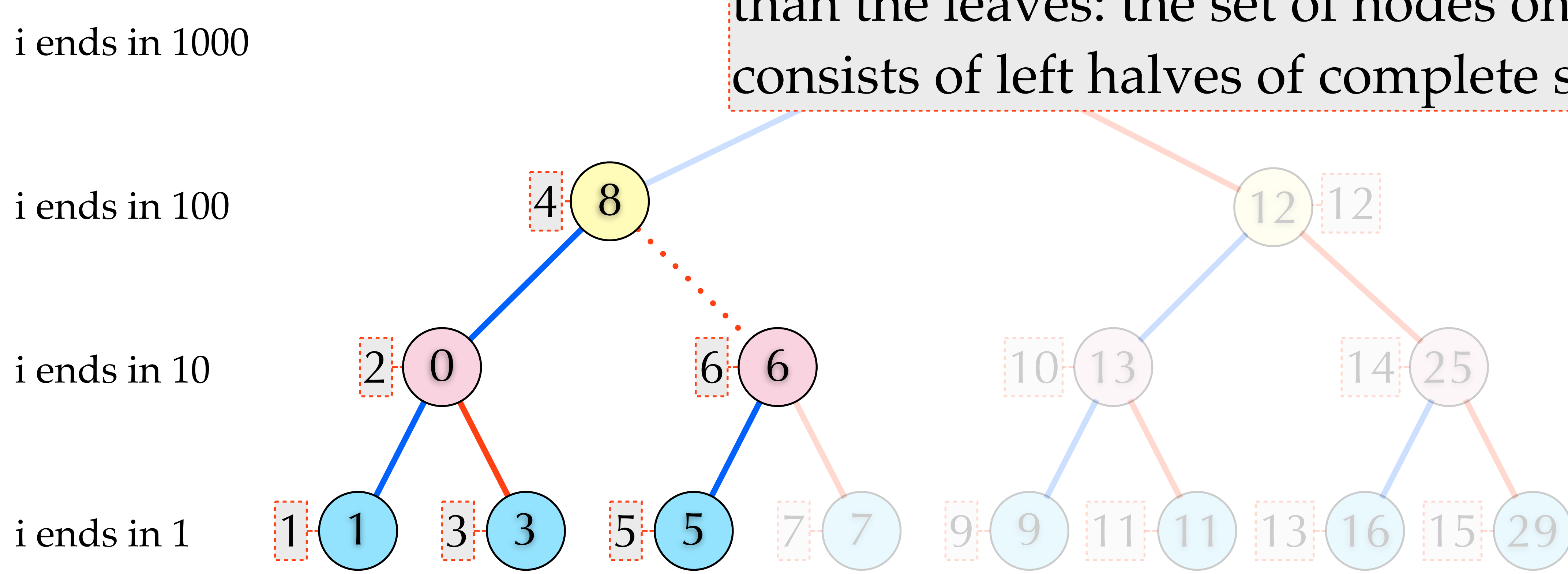
If a complete binary tree has $2^k–1$ nodes, then its root has index $2^{k–1}$ and its left half has $2^{k–1}$ nodes.



i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
s_i	1	0	3	8	5	6	7	10	9	13	11	12	16	25	29

Fenwick Tree

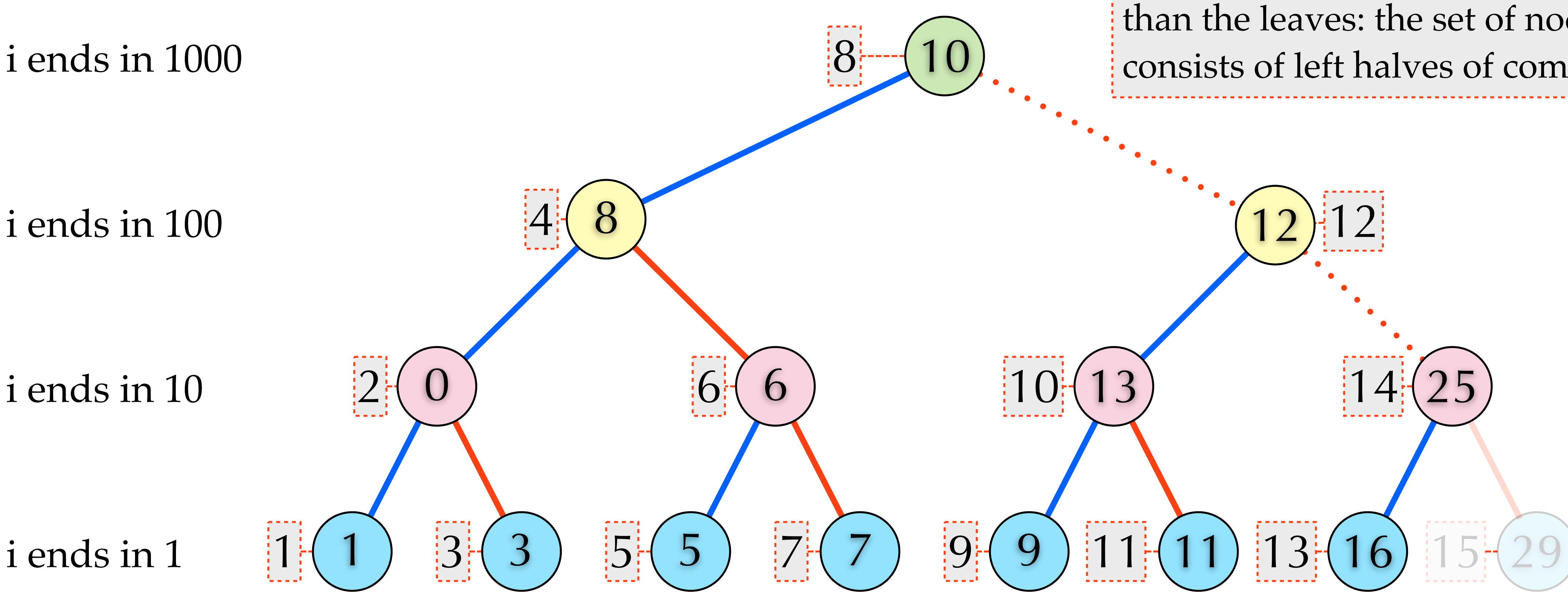
A node has index x10^i if it is i level higher than the leaves: the set of nodes on its left consists of left halves of complete subtrees.



i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
s_i	1	0	3	8	5	6	7	10	9	13	11	12	16	25	29

Fenwick Tree

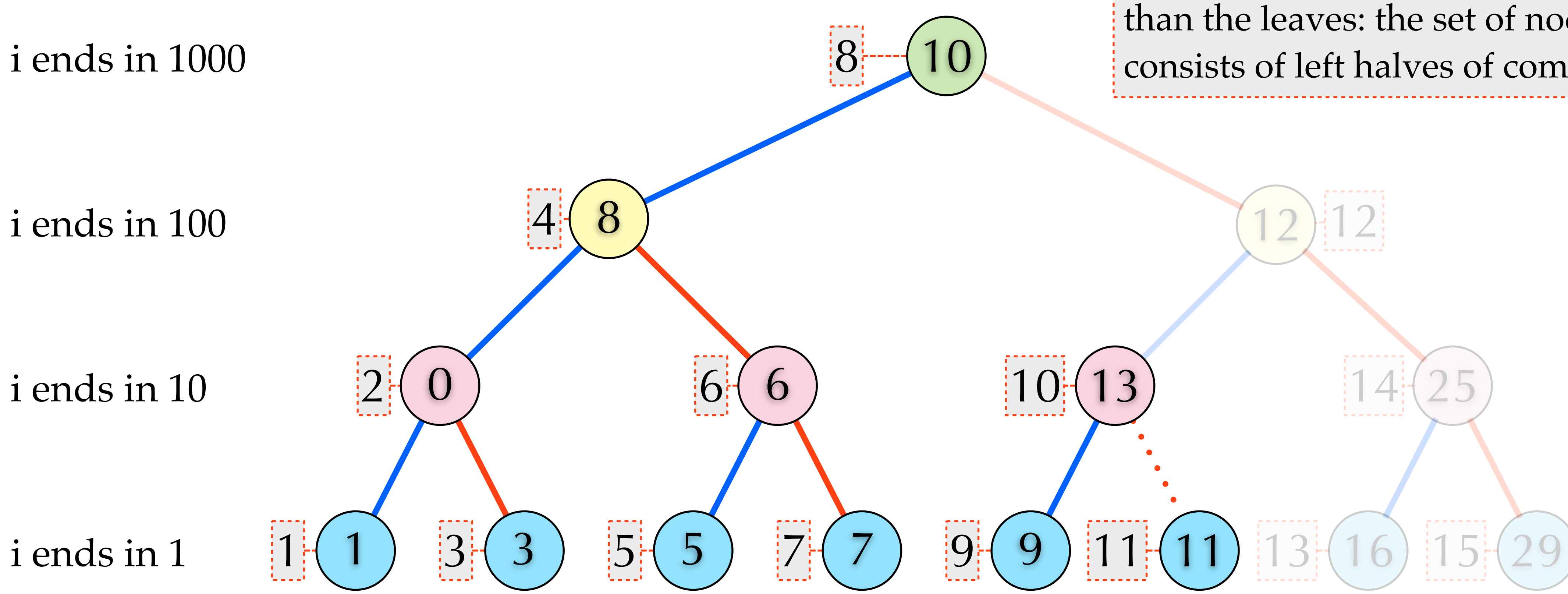
A node has index $x10^i$ if it is i level higher than the leaves: the set of nodes on its left consists of left halves of complete subtrees.



i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
s_i	1	0	3	8	5	6	7	10	9	13	11	12	16	25	29

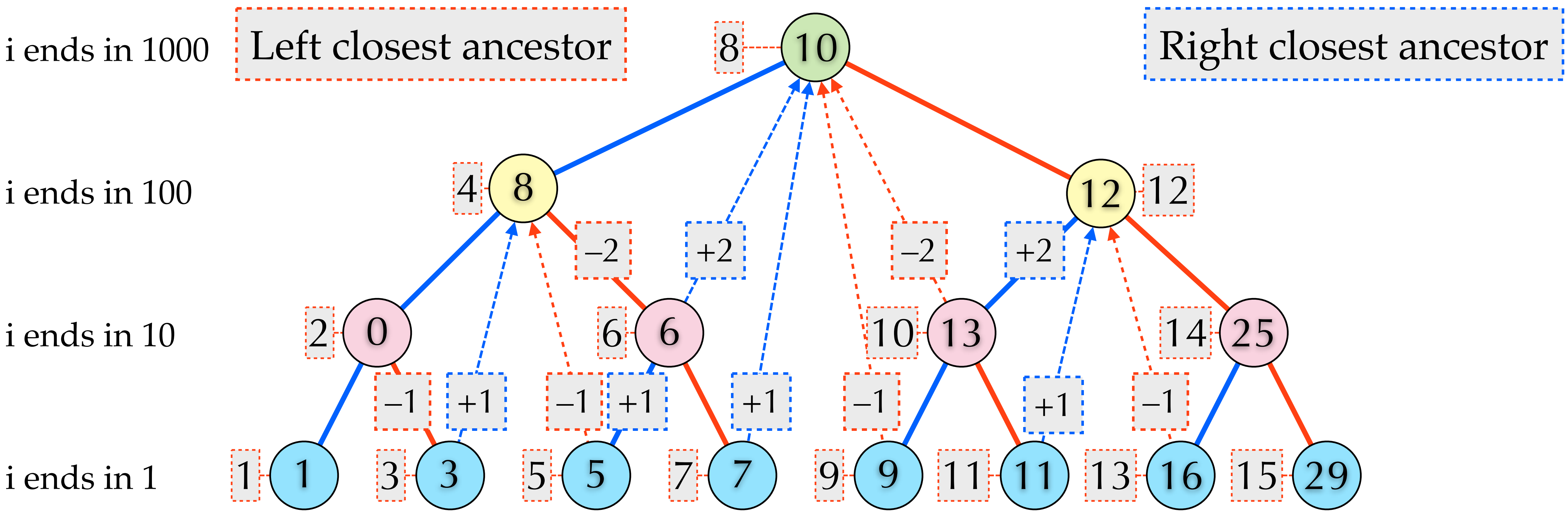
Fenwick Tree

A node has index x10^i if it is i level higher than the leaves: the set of nodes on its left consists of left halves of complete subtrees.



i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
s_i	1	0	3	8	5	6	7	10	9	13	11	12	16	25	29

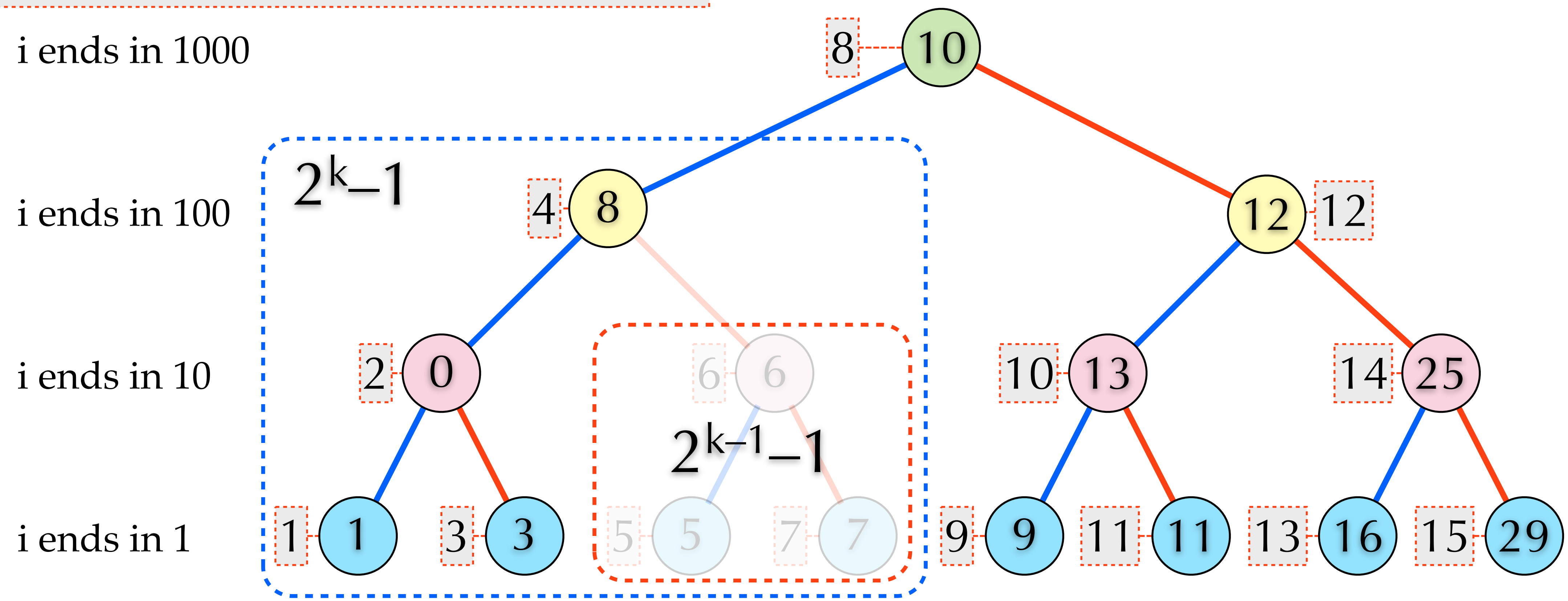
Closest Ancestors



i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
s_i	1	0	3	8	5	6	7	10	9	13	11	12	16	25	29

Parent & Child

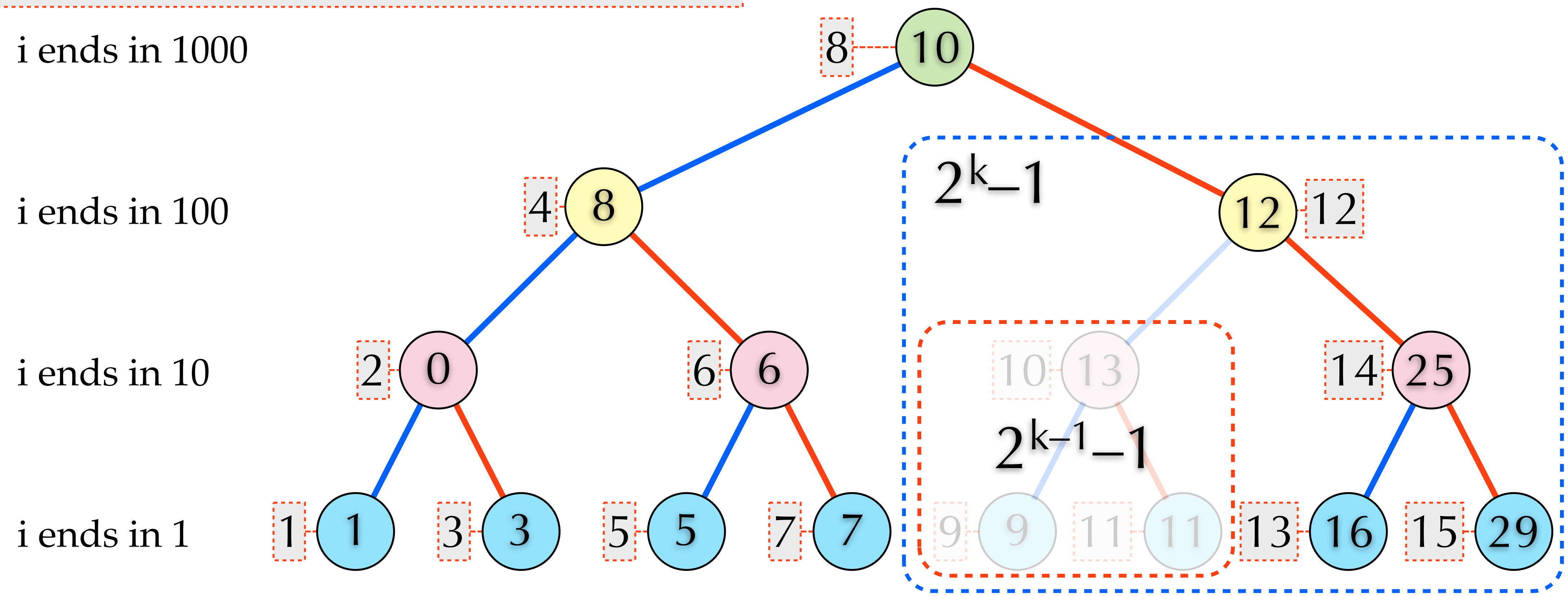
A left child x's parent: $x+2^{k-1}$



i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
s_i	1	0	3	8	5	6	7	10	9	13	11	12	16	25	29

Parent & Child

A right child x's parent: $x-2^{k-1}$



i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
s_i	1	0	3	8	5	6	7	10	9	13	11	12	16	25	29

Finding Parent

- Lowest non-zero bit of x:
  - $\text{lnzb}(x) = x \,\&\, (-x)$
- $y = x - \text{lnzb}(x)$:
  - The closest left ancestor of x.
- $z = x + \text{lnzb}(x)$:
  - The closest right ancestor of x.
- Parent: The closest ancestor!

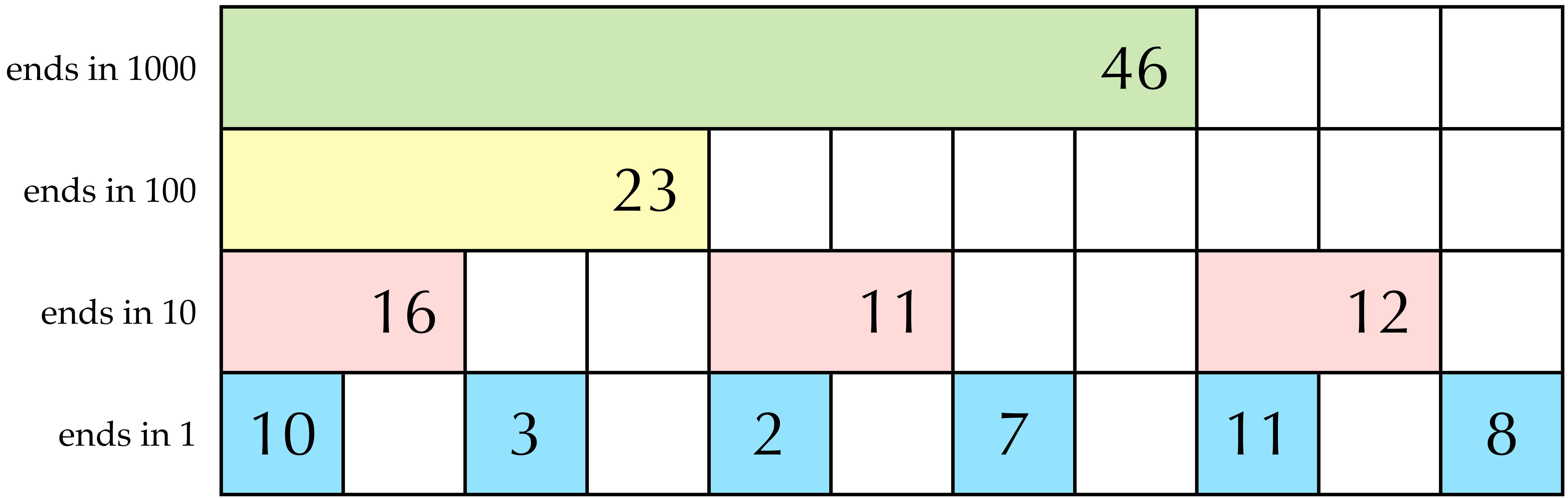
Lowest Non-Zero Bit

In C, we use 2's complement to represent a negative integer: $-x=(\sim x+1)$.

x	0	0	0	1	1	0	1	0	0	1	0	1	0	0	0
~x	1	1	1	0	0	1	0	1	1	0	1	0	1	1	1
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
–x	1	1	1	0	0	1	0	1	1	0	1	1	0	0	0
x	0	0	0	1	1	0	1	0	0	1	0	1	0	0	0
x&(–x)	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0

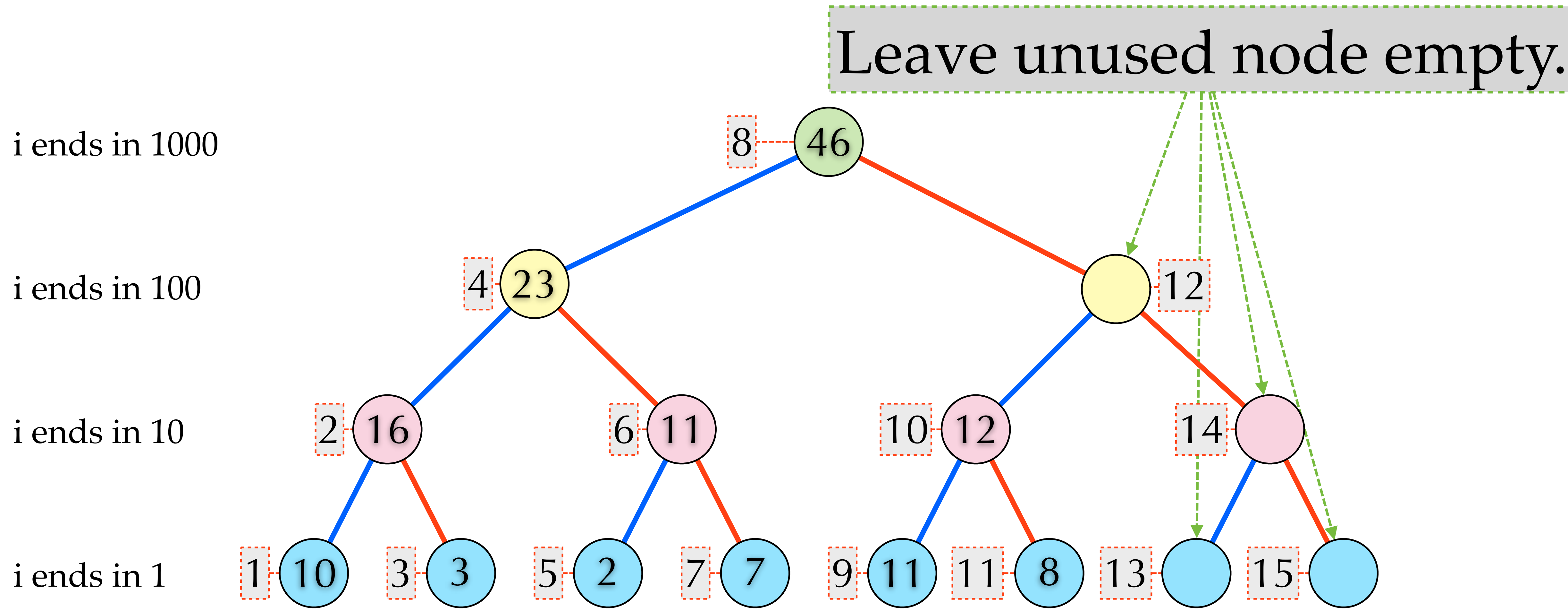
Fenwick Tree: Prefix Sum

Key idea: Store the sum of left half of a subtree on its root.



f_i	10	16	3	23	2	11	7	46	11	12	8
s_i	10	6	3	4	2	9	7	5	11	1	8

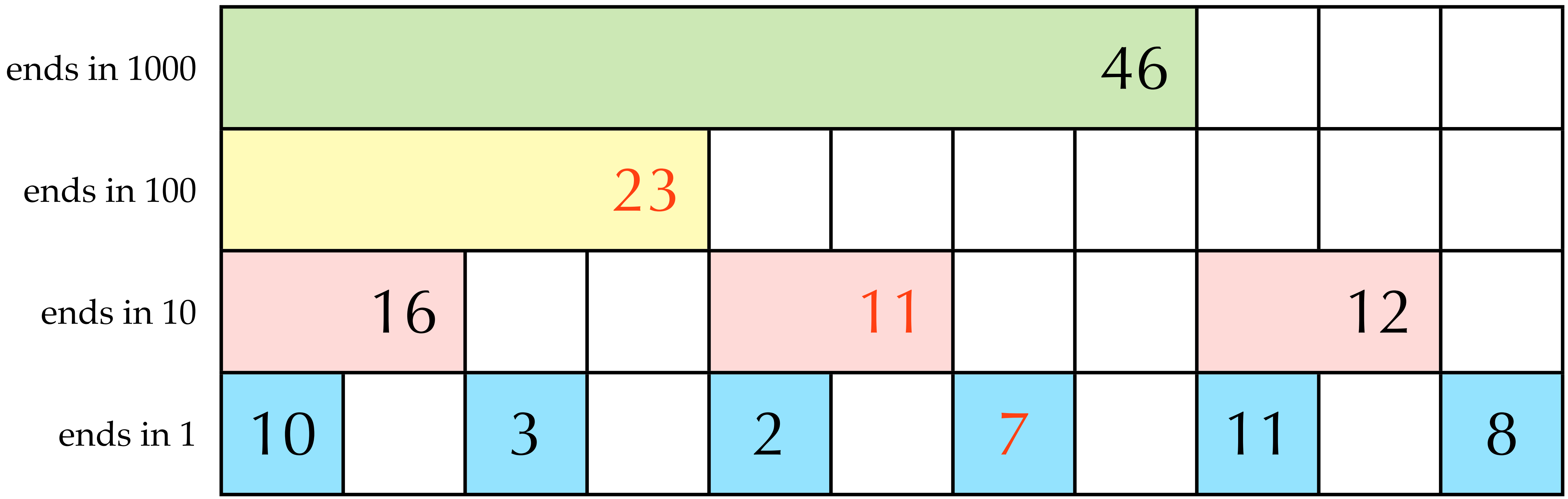
Fenwick Tree



f_i	10	16	3	23	2	11	7	46	11	12	8				
s_i	10	6	3	4	2	9	7	5	11	1	8				

Fenwick Tree: Query

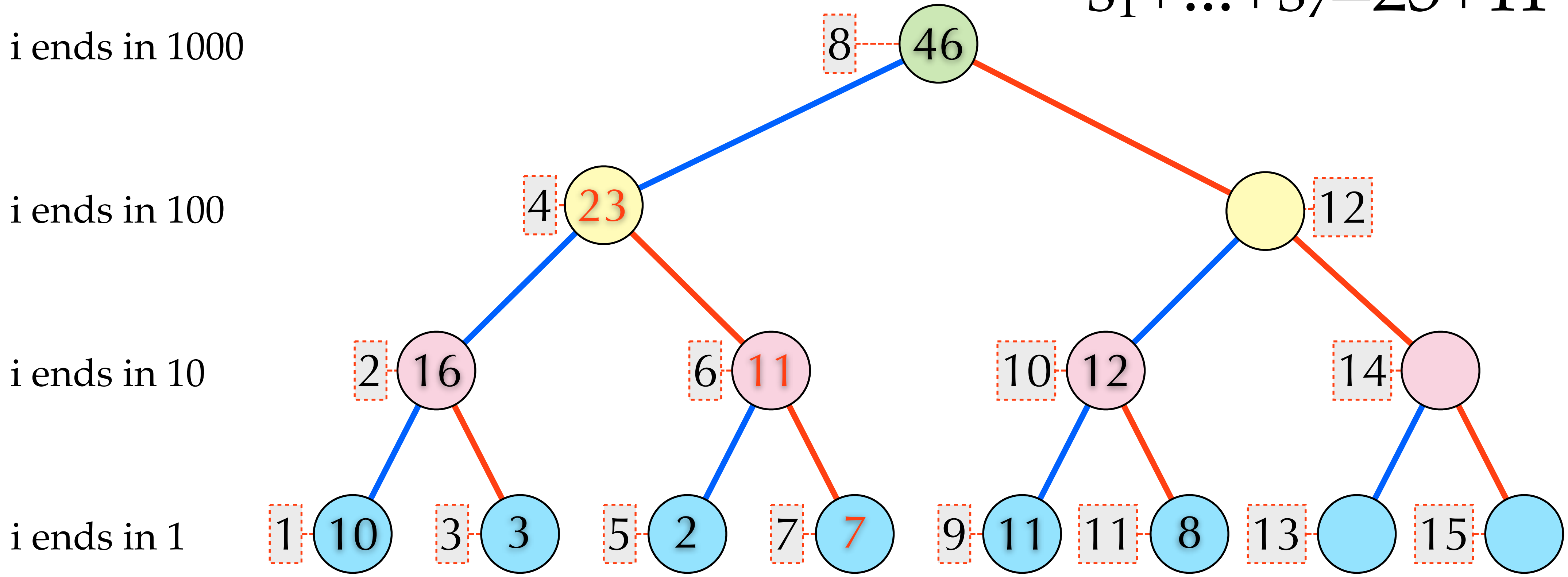
$s_1+\ldots+s_7=23+11+7=41$



f_i	10	16	3	23	2	11	7	46	11	12	8
s_i	10	6	3	4	2	9	7	5	11	1	8

Fenwick Tree: Query

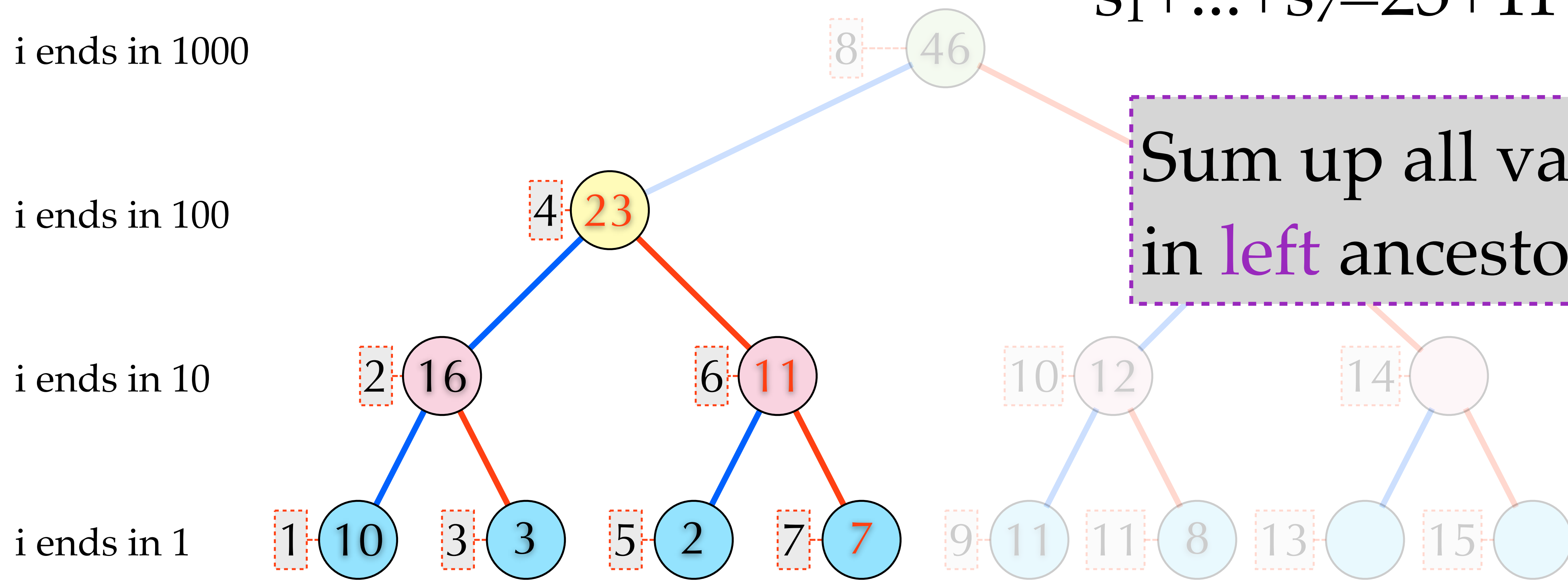
$s_1+\ldots+s_7=23+11+7=41$



f_i	10	16	3	23	2	11	7	46	11	12	8				
s_i	10	6	3	4	2	9	7	5	11	1	8				

Fenwick Tree: Query

$s_1+\ldots+s_7=23+11+7=41$

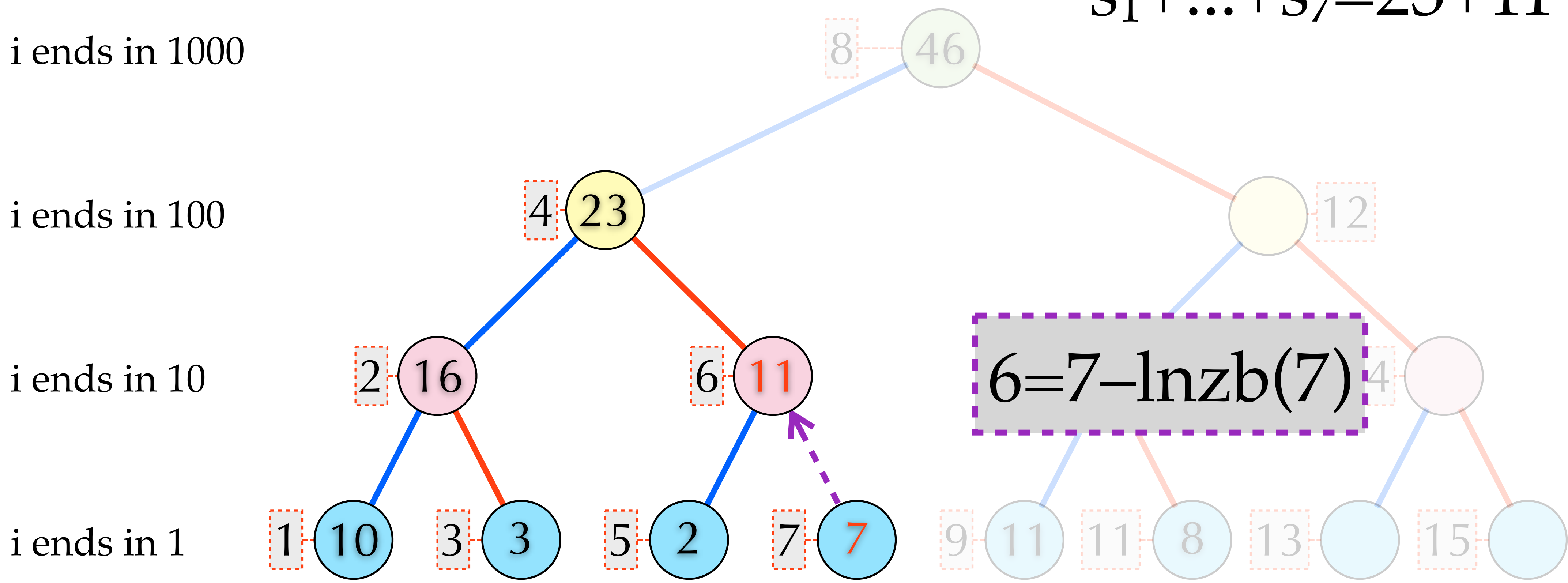


Sum up all values in left ancestors.

f_i	10	16	3	23	2	11	7	46	11	12	8				
s_i	10	6	3	4	2	9	7	5	11	1	8				

Fenwick Tree: Query

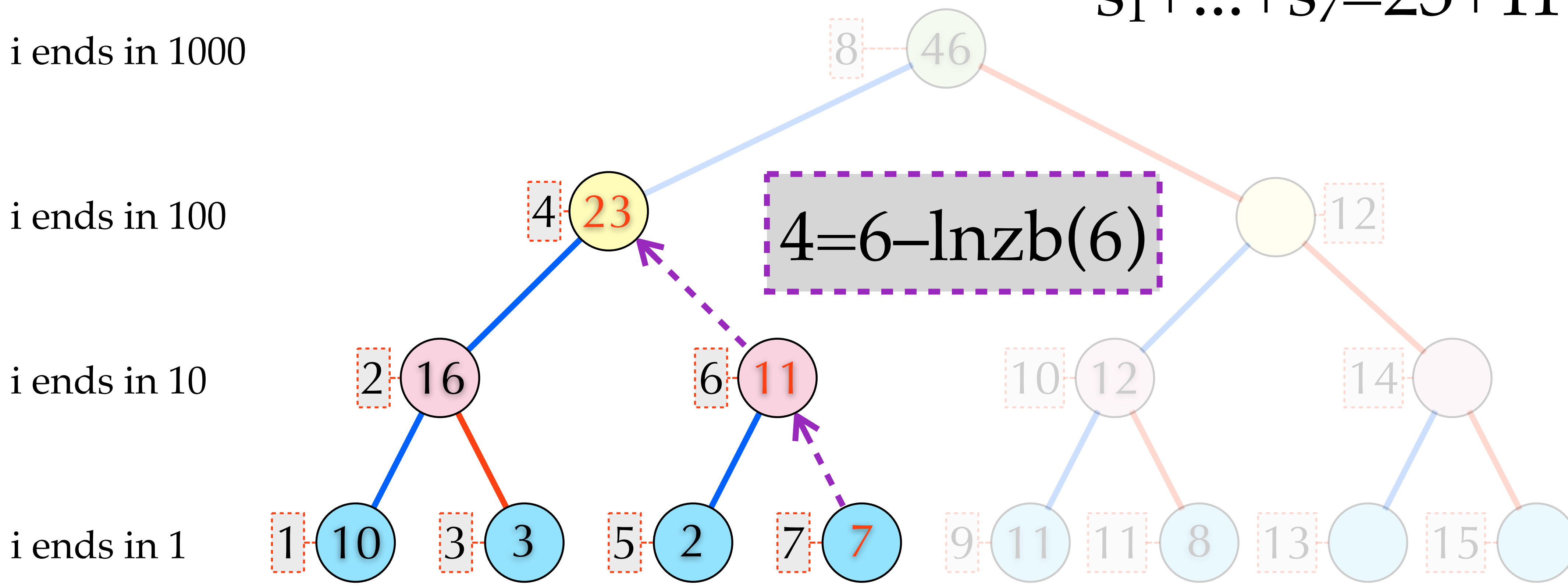
$s_1+\ldots+s_7=23+11+7=41$



f_i	10	16	3	23	2	11	7	46	11	12	8					
s_i	10	6	3	4	2	9	7	5	11	1	8					

Fenwick Tree: Query

$$s_1+\ldots+s_7=23+11+7=41$$



f_i	10	16	3	23	2	11	7	46	11	12	8					
s_i	10	6	3	4	2	9	7	5	11	1	8					

Fenwick Tree: Query

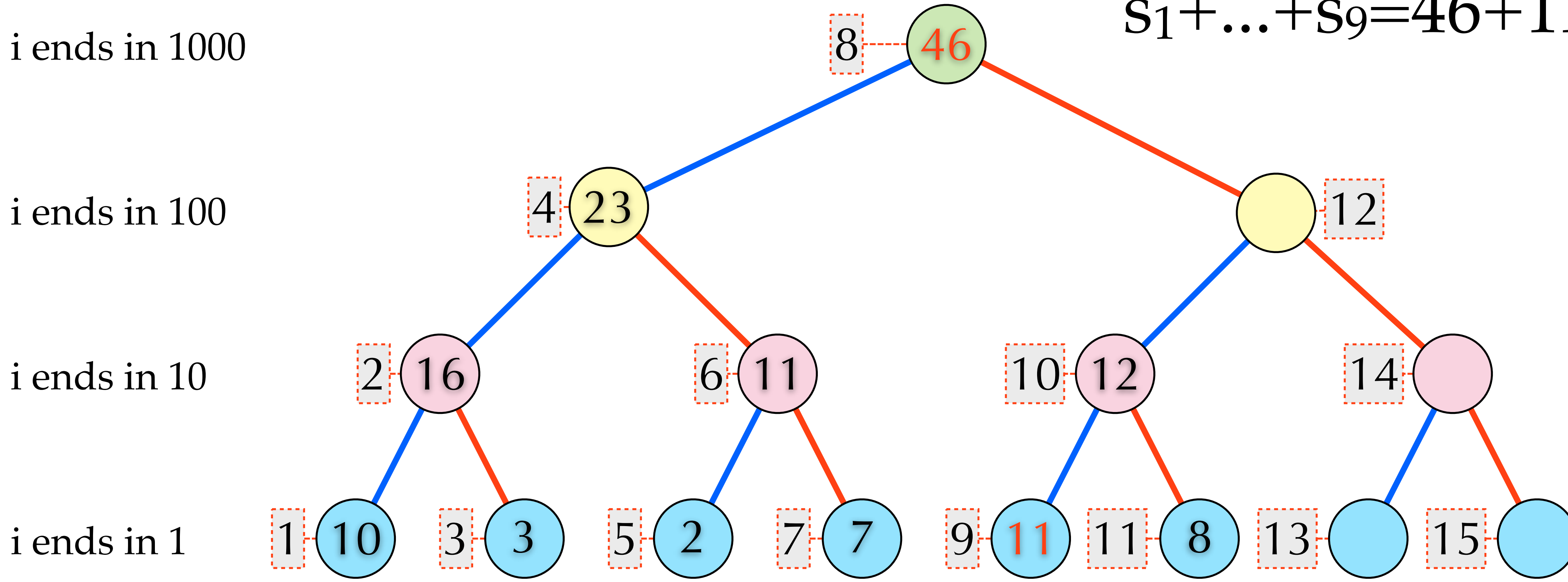
$s_1+\ldots+s_9=46+11=57$

ends in 1000								46			
ends in 100				23							
ends in 10		16				11				12	
ends in 1	10		3		2		7		11		8

f_i	10	16	3	23	2	11	7	46	11	12	8
s_i	10	6	3	4	2	9	7	5	11	1	8

Fenwick Tree: Query

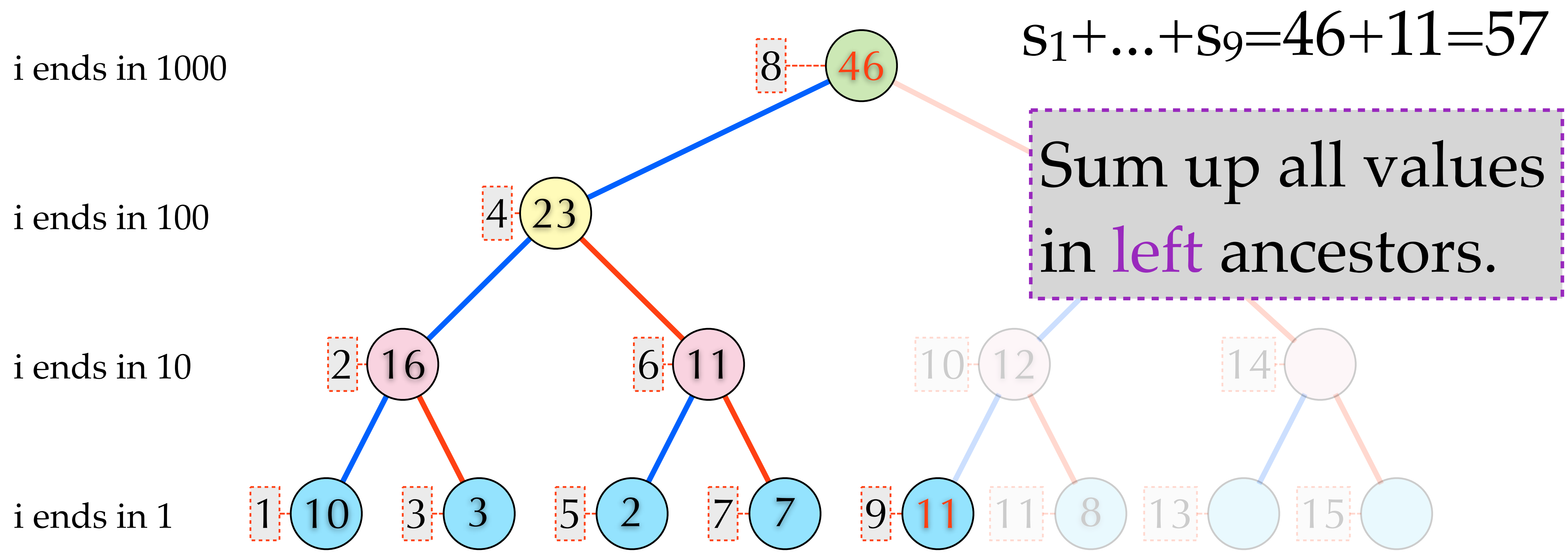
$s_1 + \ldots + s_9 = 46 + 11 = 57$



f_i	10	16	3	23	2	11	7	46	11	12	8				
s_i	10	6	3	4	2	9	7	5	11	1	8				

Fenwick Tree: Query

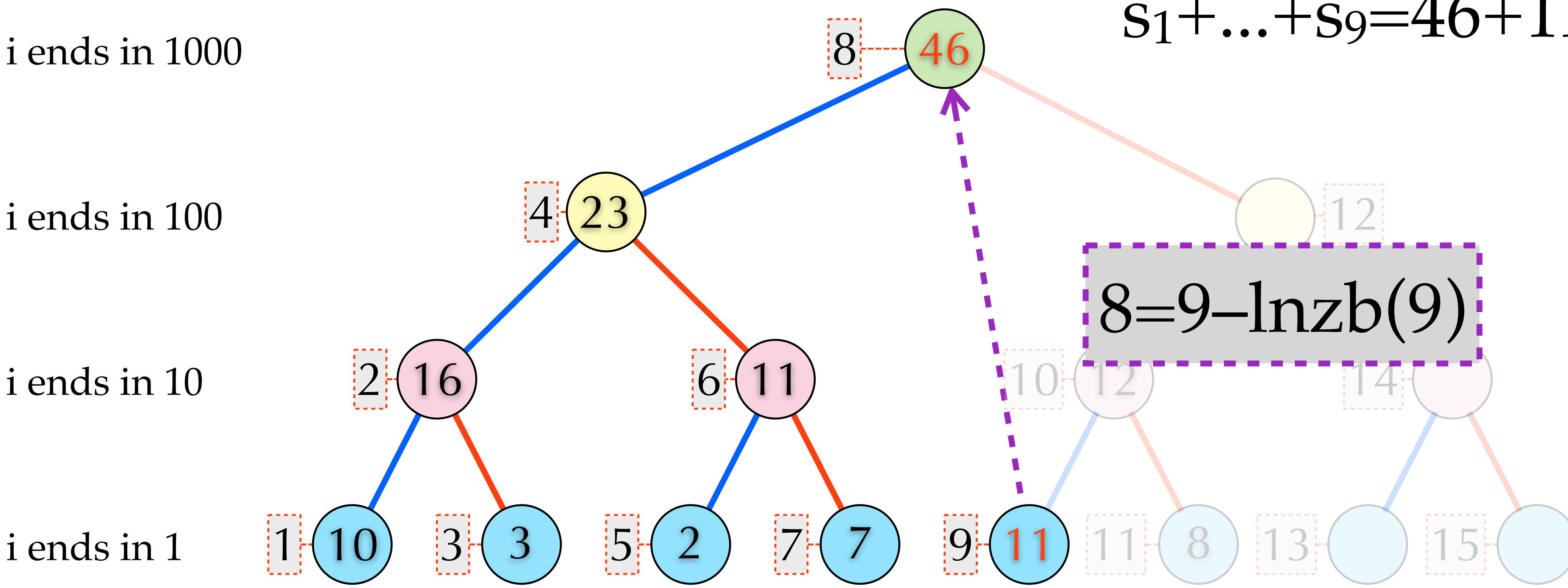
$s_1+\ldots+s_9=46+11=57$



f_i	10	16	3	23	2	11	7	46	11	12	8				
s_i	10	6	3	4	2	9	7	5	11	1	8				

Fenwick Tree: Query

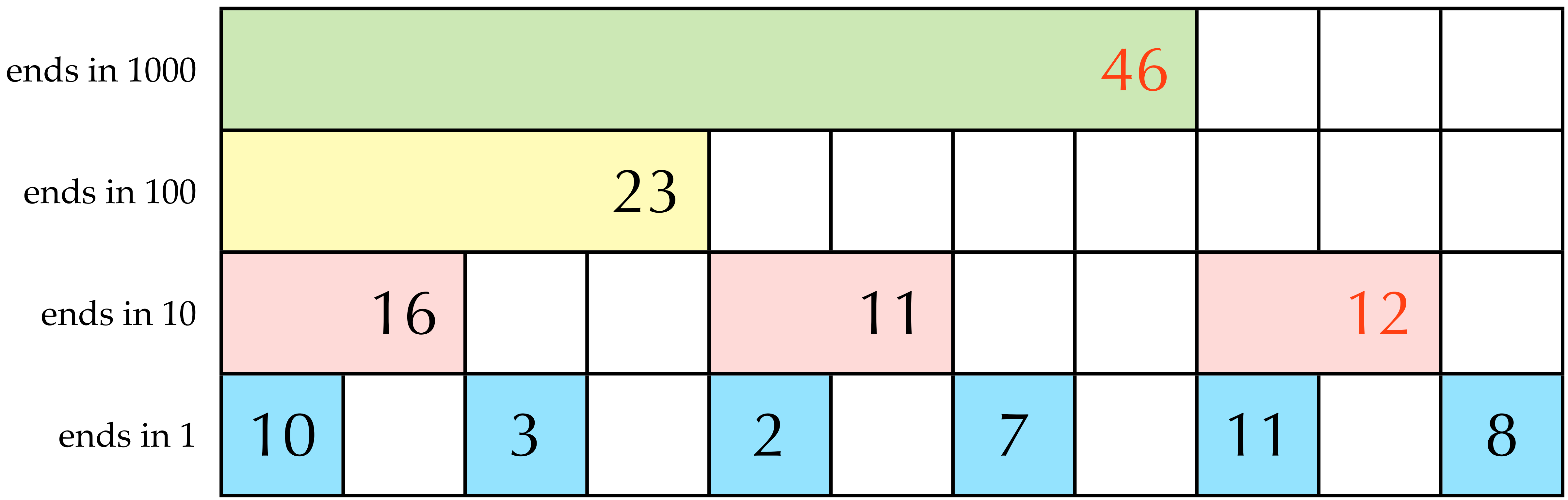
$s_1 + \ldots + s_9 = 46 + 11 = 57$



f_i	10	16	3	23	2	11	7	46	11	12	8				
s_i	10	6	3	4	2	9	7	5	11	1	8				

Fenwick Tree: Query

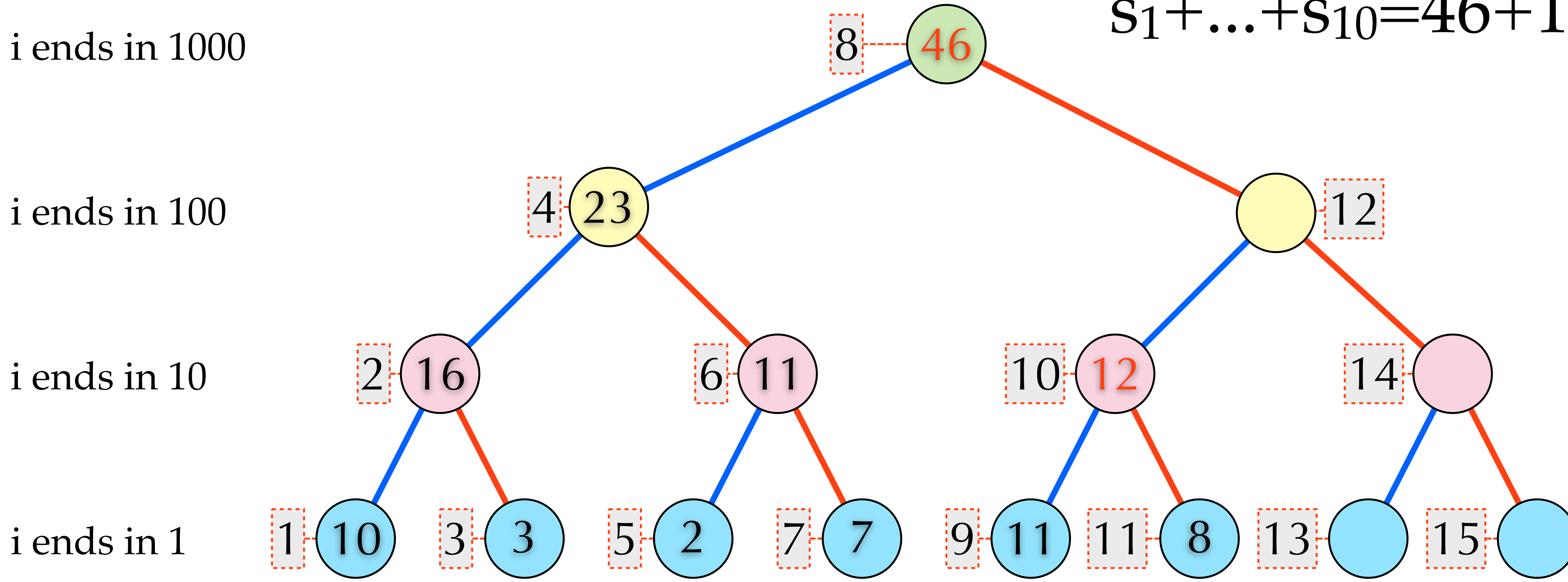
$s_1 + \ldots + s_{10} = 46 + 12 = 58$



f_i	10	16	3	23	2	11	7	46	11	12	8
s_i	10	6	3	4	2	9	7	5	11	1	8

Fenwick Tree: Query

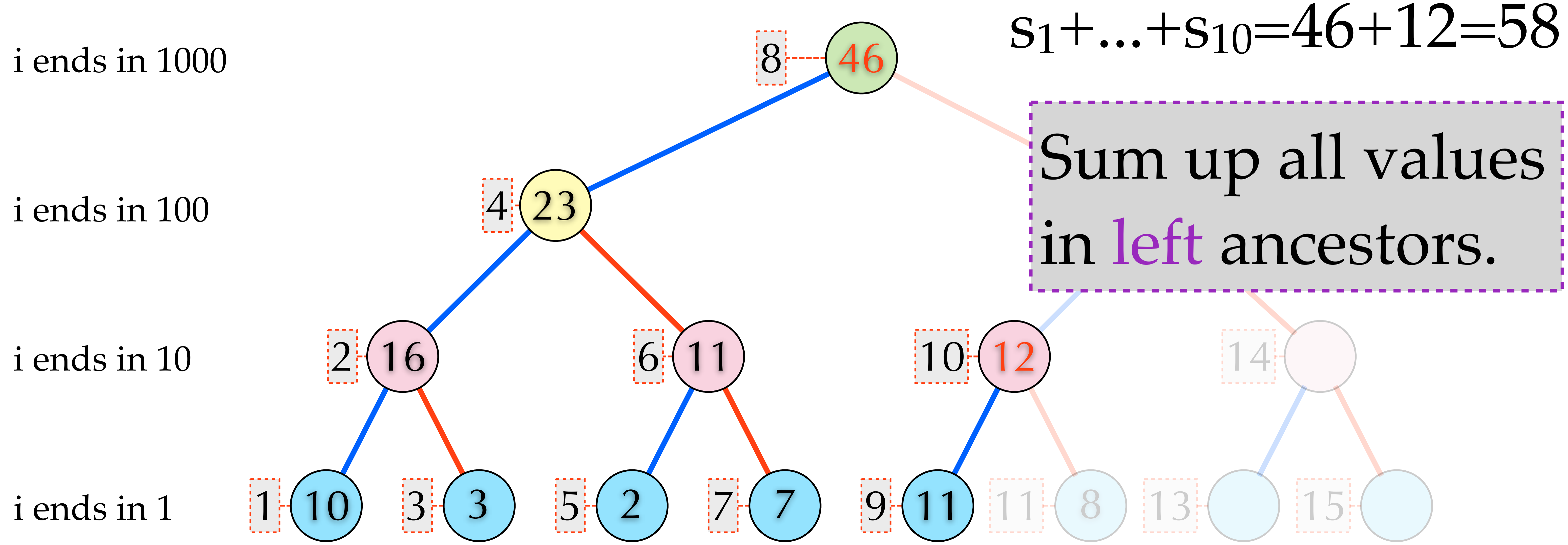
$s_1 + \ldots + s_{10} = 46 + 12 = 58$



f_i	10	16	3	23	2	11	7	46	11	12	8				
s_i	10	6	3	4	2	9	7	5	11	1	8				

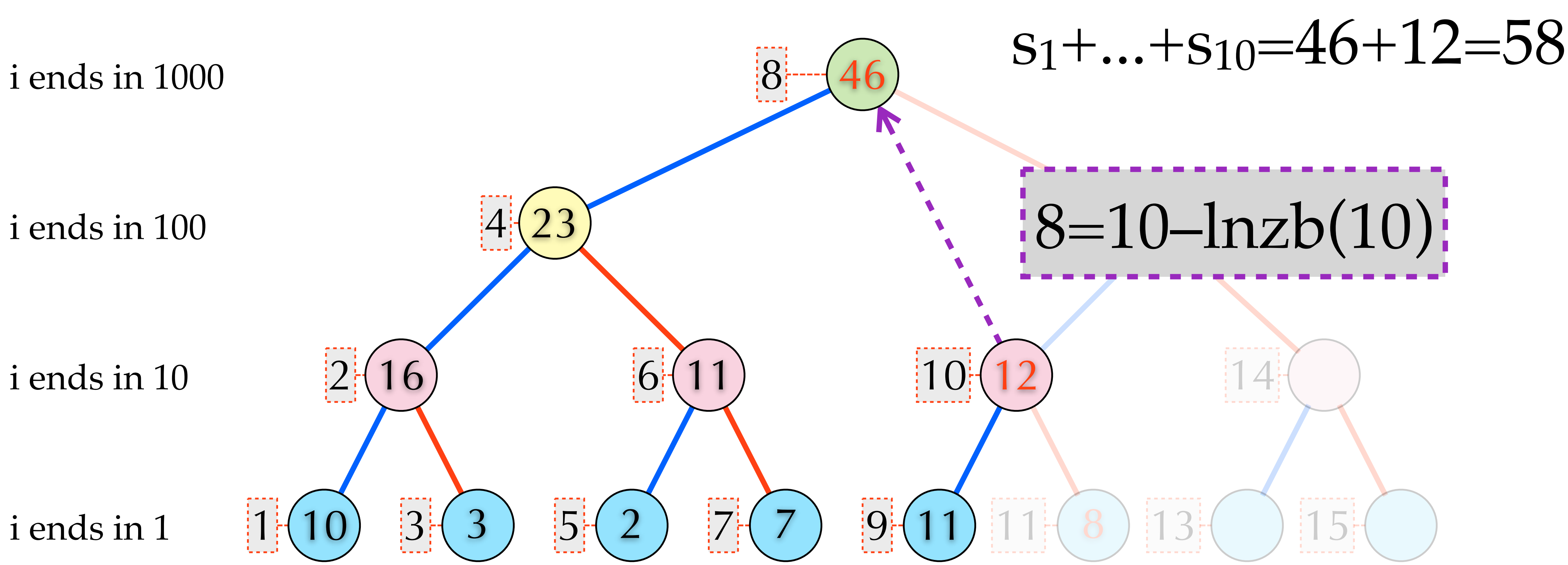
Fenwick Tree: Query

$s_1+...+s_{10}=46+12=58$



f_i	10	16	3	23	2	11	7	46	11	12	8				
s_i	10	6	3	4	2	9	7	5	11	1	8				

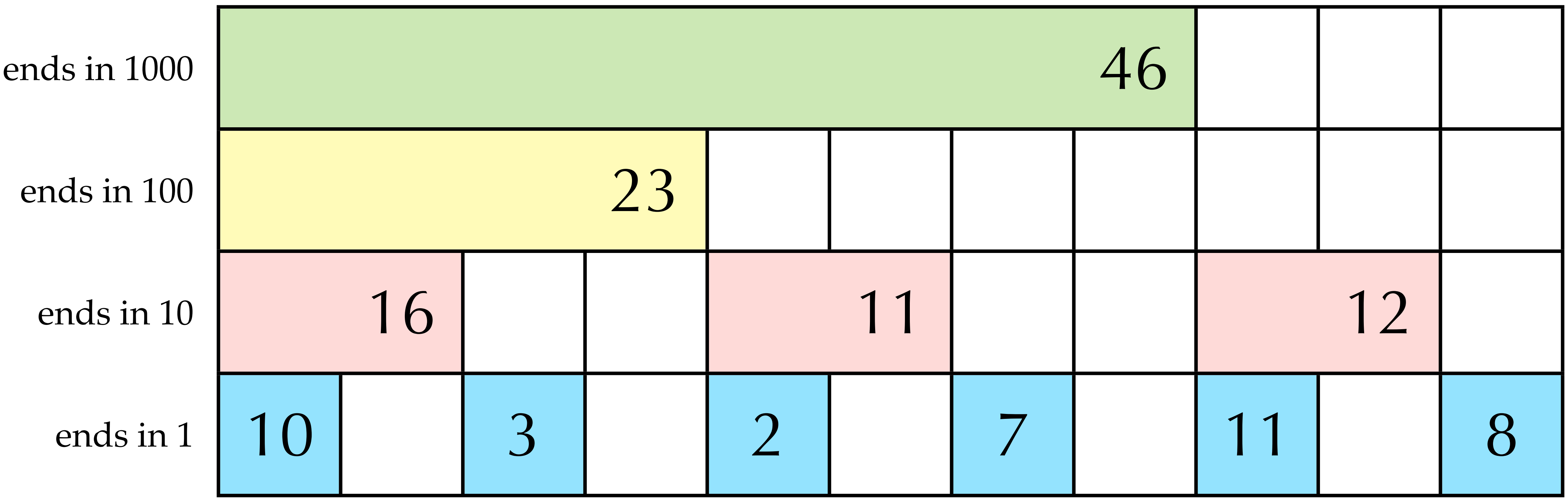
Fenwick Tree: Query



f_i	10	16	3	23	2	11	7	46	11	12	8				
s_i	10	6	3	4	2	9	7	5	11	1	8				

Fenwick Tree: Update

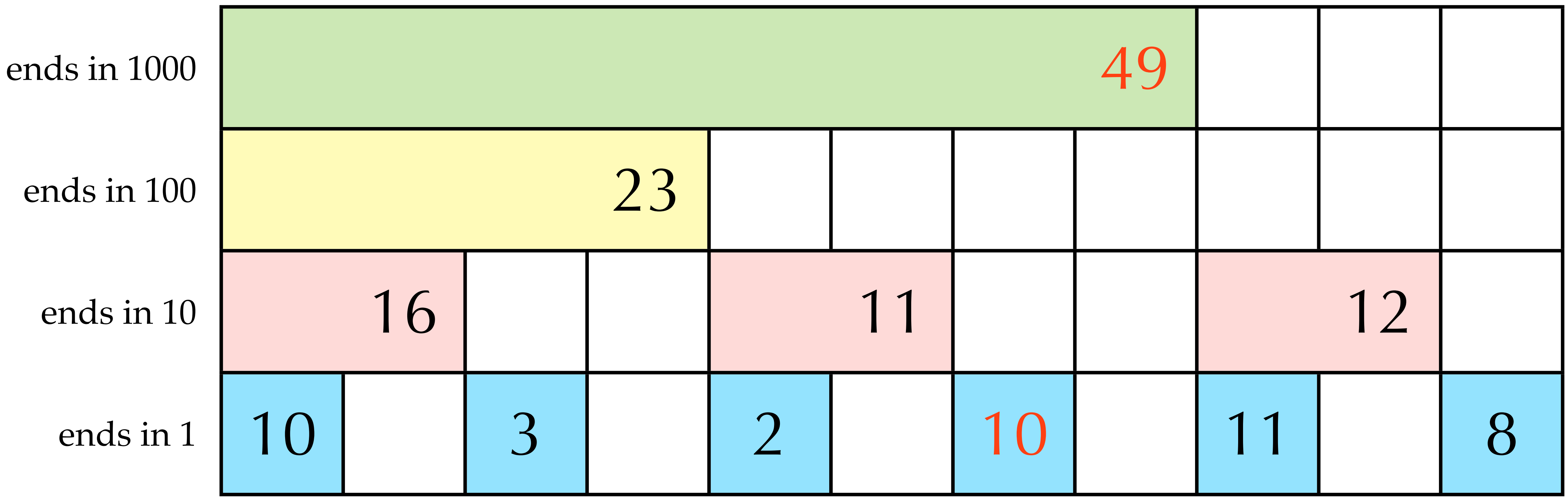
update s_7=10



f_i	10	16	3	23	2	11	7	46	11	12	8
s_i	10	6	3	4	2	9	7	5	11	1	8

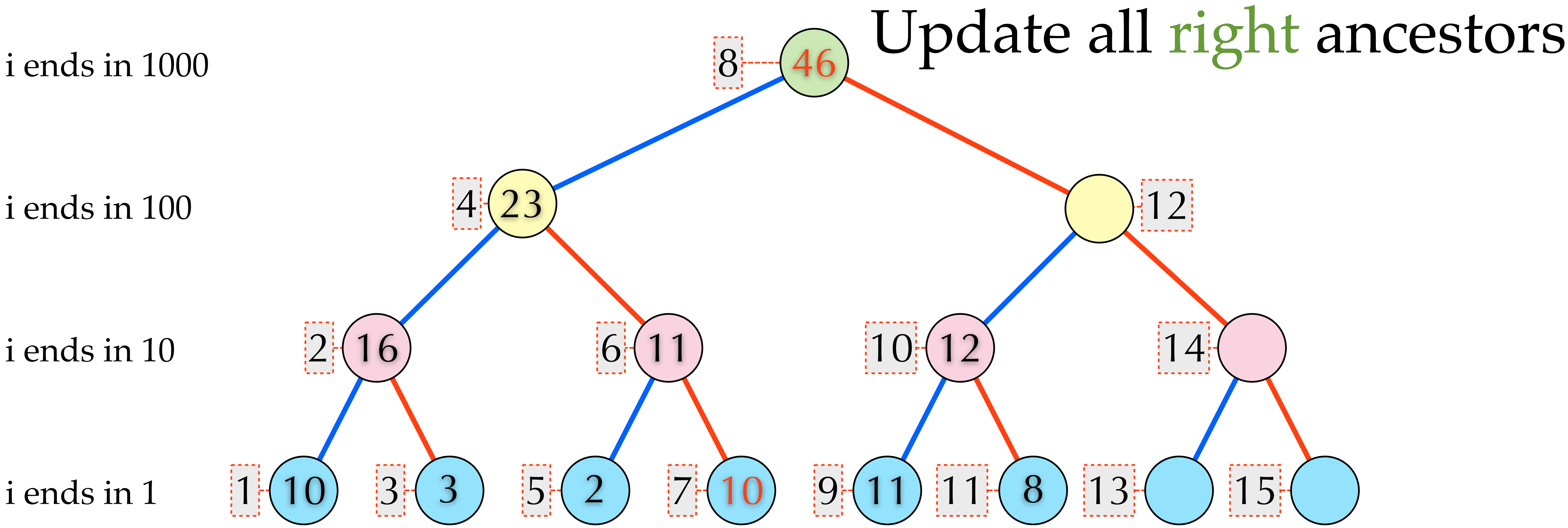
Fenwick Tree: Update

update s_7=10



f_i	10	16	3	23	2	11	10	49	11	12	8
s_i	10	6	3	4	2	9	10	5	11	1	8

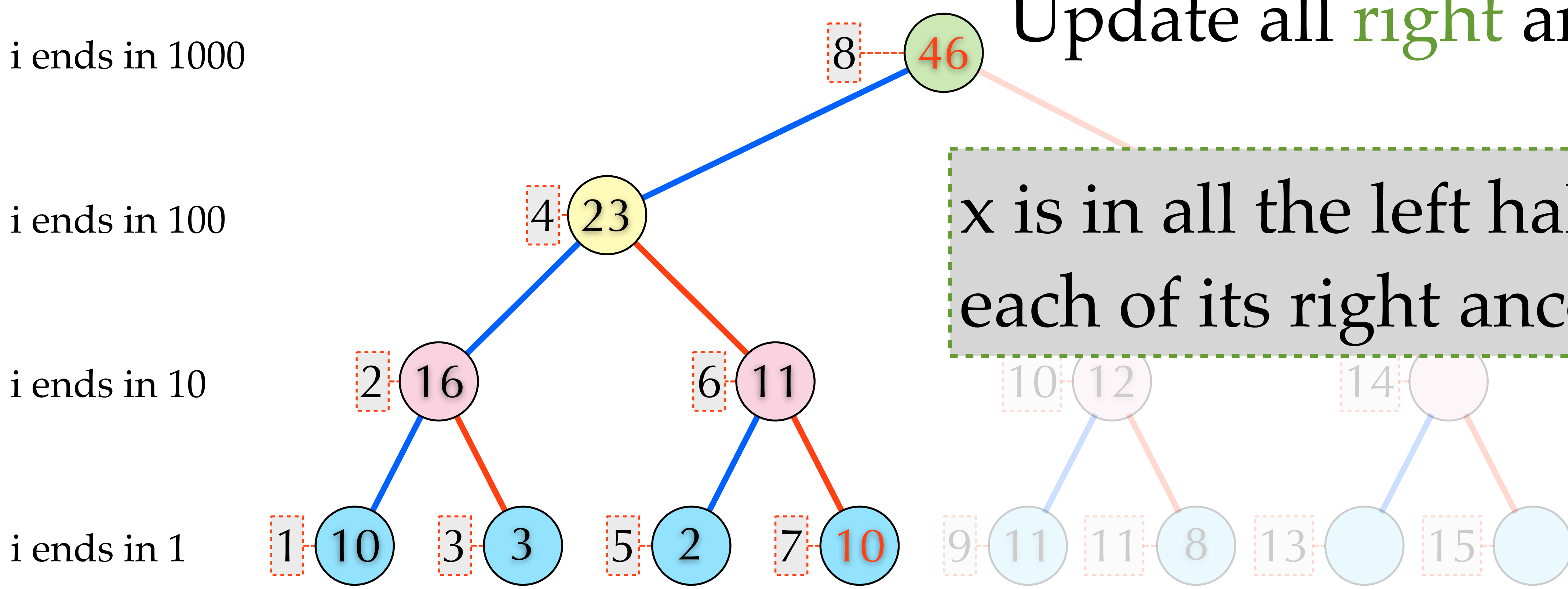
Fenwick Tree: Update



f_i	10	16	3	23	2	11	10	49	11	12	8				
s_i	10	6	3	4	2	9	10	5	11	1	8				

Fenwick Tree: Update

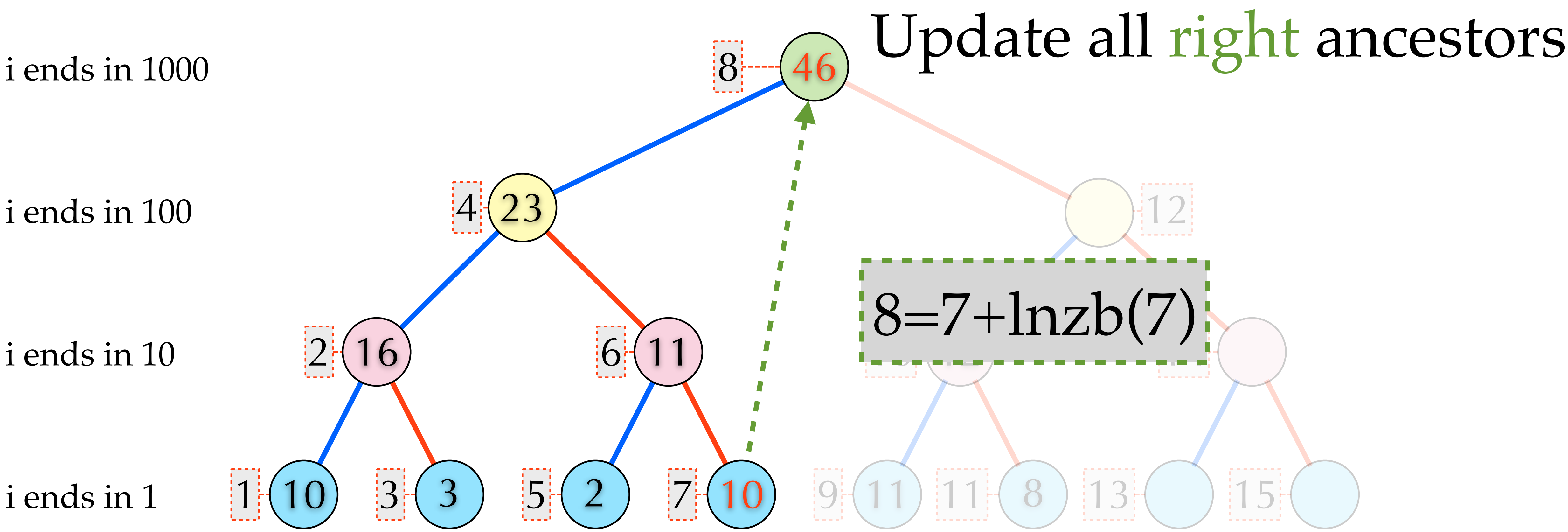
Update all right ancestors



x is in all the left half of each of its right ancestors!

f_i	10	16	3	23	2	11	10	49	11	12	8				
s_i	10	6	3	4	2	9	10	5	11	1	8				

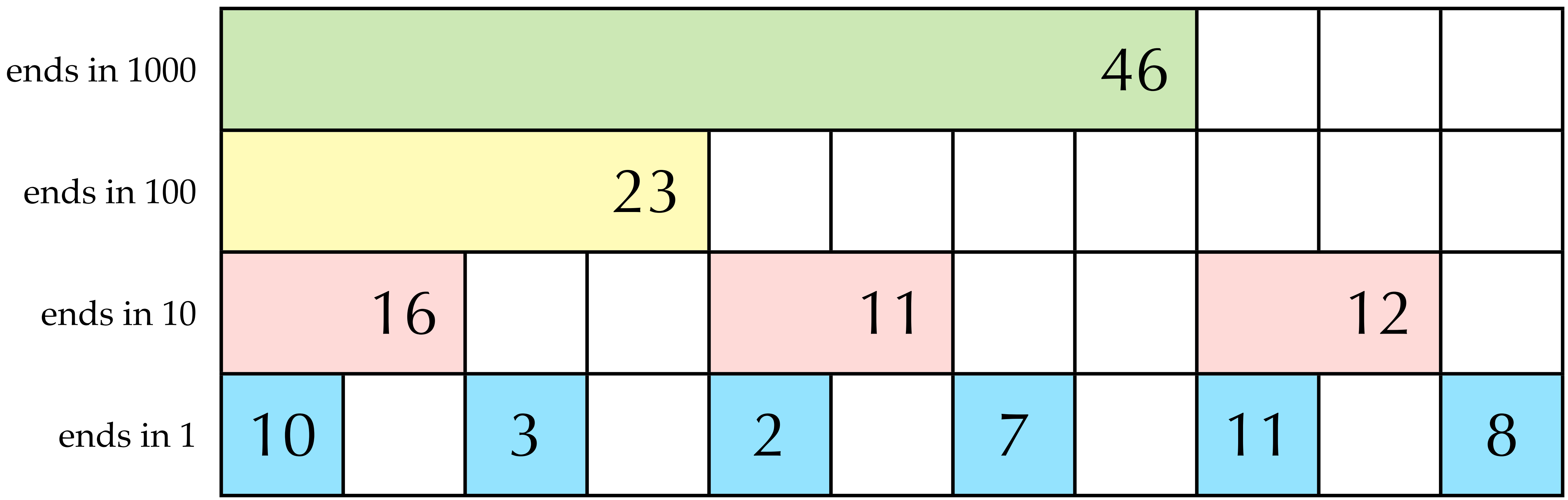
Fenwick Tree: Update



f_i	10	16	3	23	2	11	10	49	11	12	8				
s_i	10	6	3	4	2	9	10	5	11	1	8				

Fenwick Tree: Update

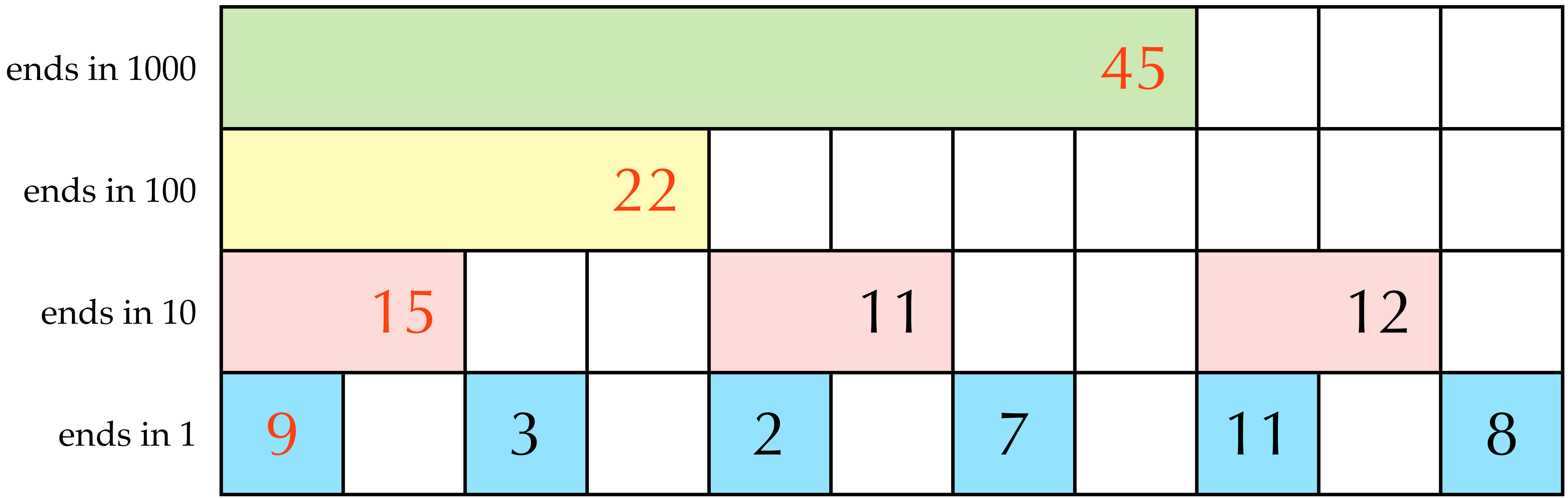
update s_1=9



f_i	10	16	3	23	2	11	7	46	11	12	8
s_i	10	6	3	4	2	9	7	5	11	1	8

Fenwick Tree: Update

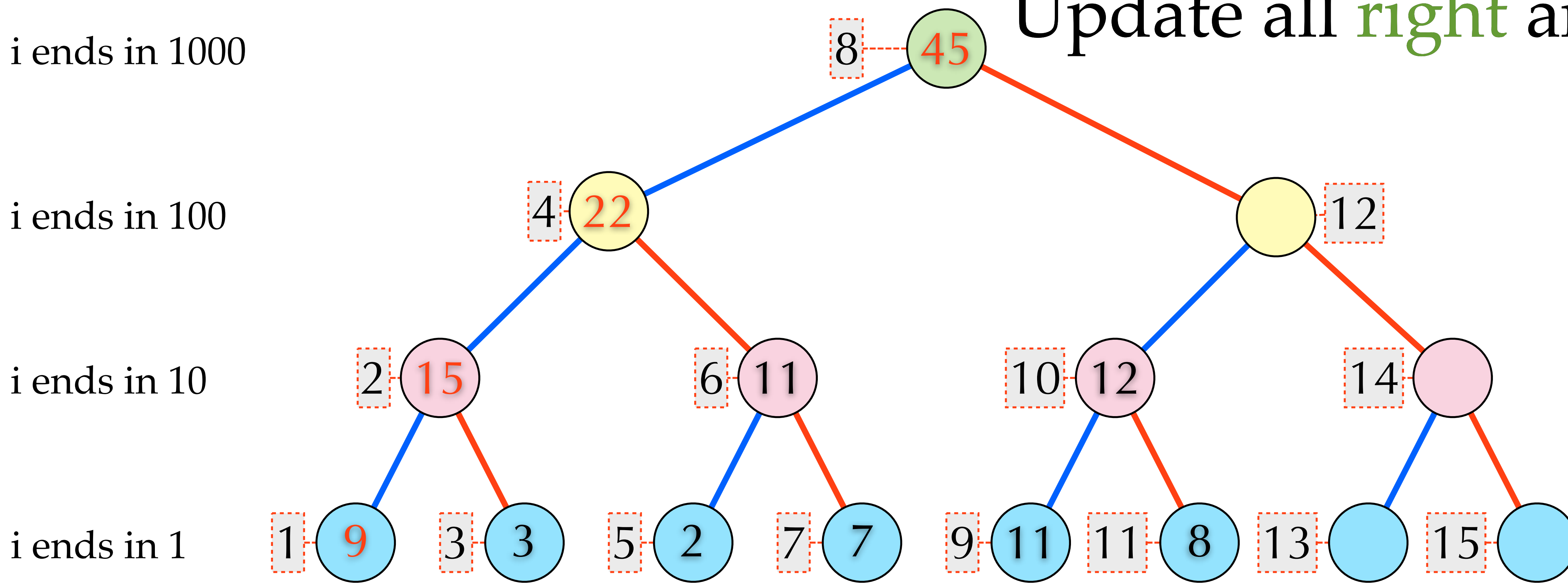
update $s_1=9$



f_i	9	15	3	22	2	11	7	45	11	12	8
s_i	9	6	3	4	2	9	7	5	11	1	8

Fenwick Tree: Update

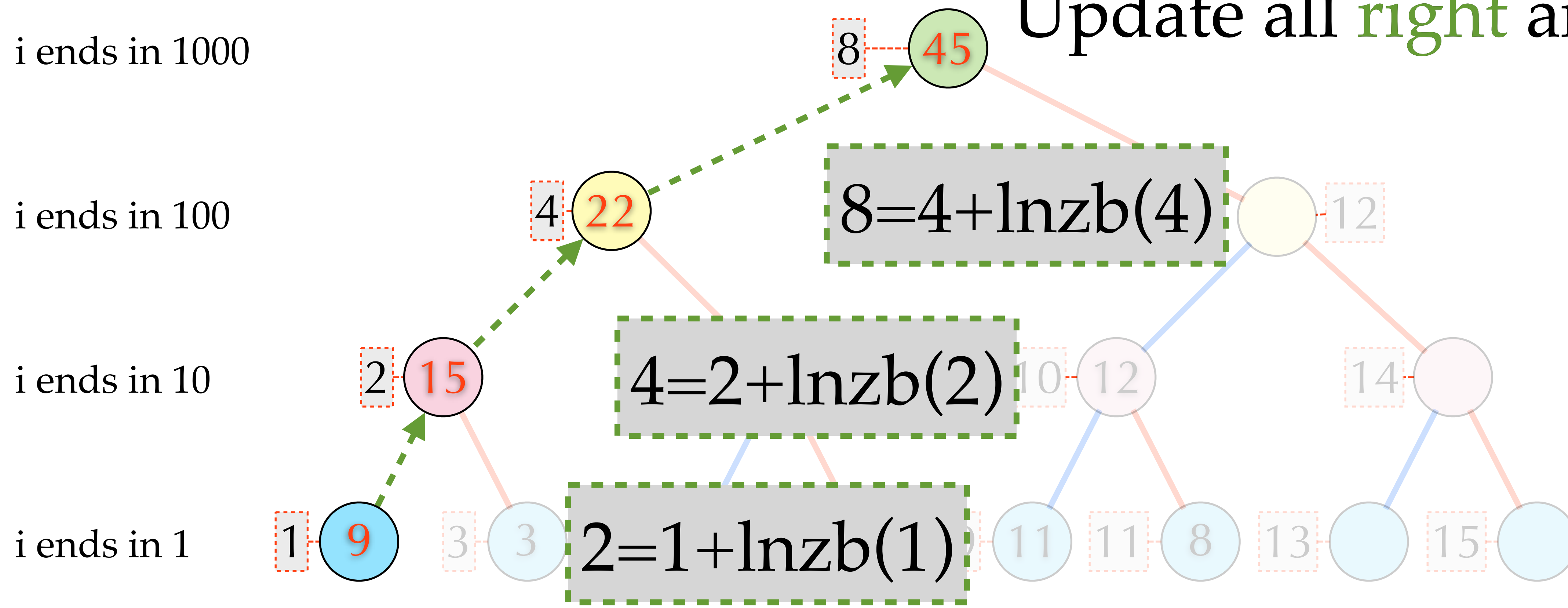
Update all right ancestors



f_i	9	15	3	22	2	11	7	45	11	12	8				
s_i	9	6	3	4	2	9	7	5	11	1	8				

Fenwick Tree: Update

Update all right ancestors



f_i	9	15	3	22	2	11	7	45	11	12	8					
s_i	9	6	3	4	2	9	7	5	11	1	8					

Homework 7.1

- a) Prove or disprove to construct a Fenwick tree from an array A[1..n] can be done in O(n).
- b) How to generalize Fenwick trees to support k-dimensional range sum query?
- c) How to use Fenwick trees to support range query without subtraction and negation?