CS636: Parallel Prefix Scan

Swarnendu Biswas

Semester 2018-2019-II

CSE, IIT Kanpur

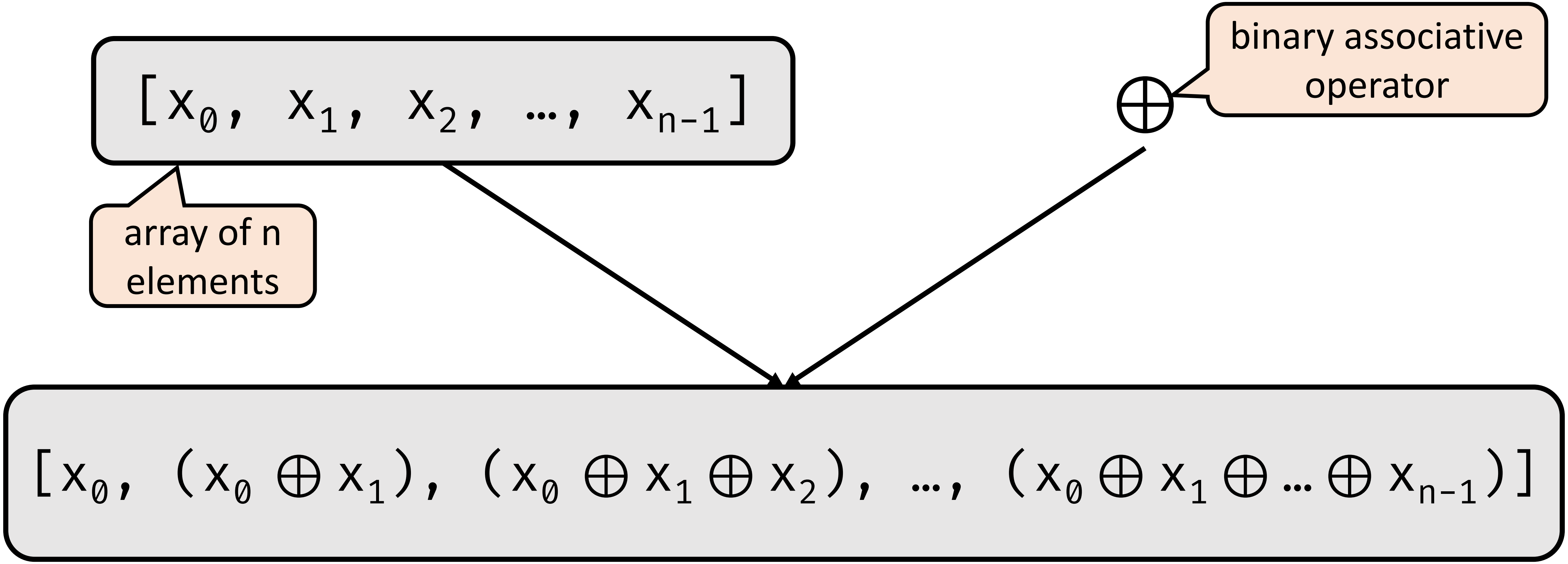
Content influenced by many excellent references, see References slide for acknowledgements.

sum_arr = f(arr)

```
int arr[8] = {10, 1, 4, 2, 9, 5, 7, 8}
```

```
int sum_arr[8] = {10, 11, 15, 17, 26, 31, 38, 46}
```

Definition of Inclusive Prefix Scan



Exclusive Prefix Scan

$[x_0,\ x_1,\ x_2,\ \ldots,\ x_{n-1}]$

$\oplus$

I

Identity

$[I,\ x_0,\ (x_0 \oplus x_1),\ \ldots,\ (x_0 \oplus x_1 \oplus \ldots \oplus x_{n-2})]$

A Problem

- Assume we have a 100-inch sandwich to feed ten people
- We know how many inches each person wants

```
3, 5, 2, 7, 28, 4, 3, 0, 8, 1
```

- How do we cut the sandwich quickly and distribute?

- **Method 1**: Cut the sandwich sequentially starting from say left

Yong Cao. Parallel Prefix Sum – Scan, ECE 408/498AL, UIUC.

Method 2

- Calculate prefix sum and cut in **parallel**

```
3, 8, 10, 17, 45, 49, 52, 52, 60, 61
```

Prefix Sum

- Inclusive Sum

$$Output_i = \sum_{j=0}^{i} arr_j$$

- Exclusive Sum

$$Output[0] = 0 \;\; \wedge \;\; Output_i = \sum_{j=0}^{i-1} arr_j, \;\; i > 0$$

Sequential Inclusive Prefix Sum

```
output[0] = arr[0]
for (int i = 1; i < n; i++) {
  output[i] = output[i-1] + arr[i];
}
```

Work done?
Span?

O(n)

Analysis of Parallel Algorithms

- T_p = Execution time of a parallel program with p processors
- **Work**
  - Total number of computation operations performed by the p processors
  - Time to run on a single processor (T_1)
- **Span**
  - Length of the longest series of sequential operations or the critical path
  - Time taken to run on infinite processors (T_∞)

Analysis of Parallel Algorithms

- **Cost**
  - Total time spent by **all** processors in computation (pT_p)

Cost $\geq$ Work

$$pT_p \geq T_1$$

Execution time $\geq$ Span

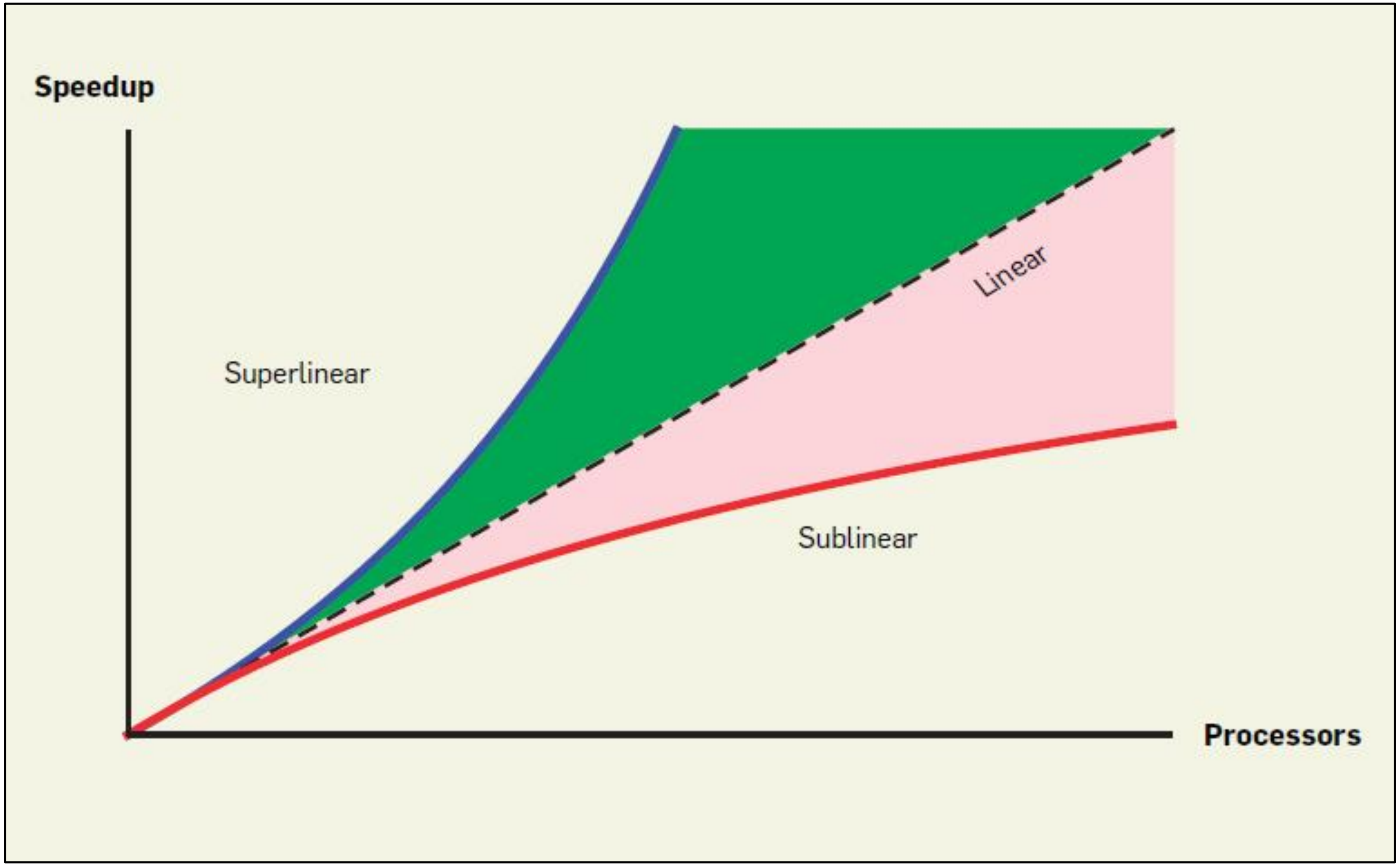
$$T_p \geq T_\infty$$

Analysis of Parallel Algorithms

- **Speedup (S_p)**
  - Total time spent by all processors in computation (pT_p)

$$\text{Speedup} = \frac{T_1}{T_p} \leq p$$

Speedup



Other Metrics

- **Efficiency**
  - Speedup per processor $\frac{S_p}{p}$
- **Parallelism**
  - Maximum possible speedup given any number of processors $\frac{T_1}{T_\infty}$

Sequential Inclusive Prefix Scan

```
output[0] = arr[0]
for (int i = 1; i < n; i++) {
  output[i] = output[i-1] + arr[i];
}
```

Asymptotic complexity O(n)

Work = O(n)

Span = O(n)

Parallel Prefix Sum

How can Inclusive Prefix Scan be Parallelized?

```
output[0] = arr[0]
for (int i = 1; i < n; i++) {
  output[i] = output[i-1] + arr[i];
}
```

loop-carried dependence

A Naïve Parallel Prefix Sum

- Use one thread to compute each output element
  - The thread adds up all the previous elements needed for the output

```
y0 = x0
y1 = x0 + x1
y2 = x0 + x1 + x2
…
```

- Work $= 1 + 2 + 3 + \ldots + n = \dfrac{n(n+1)}{2}$

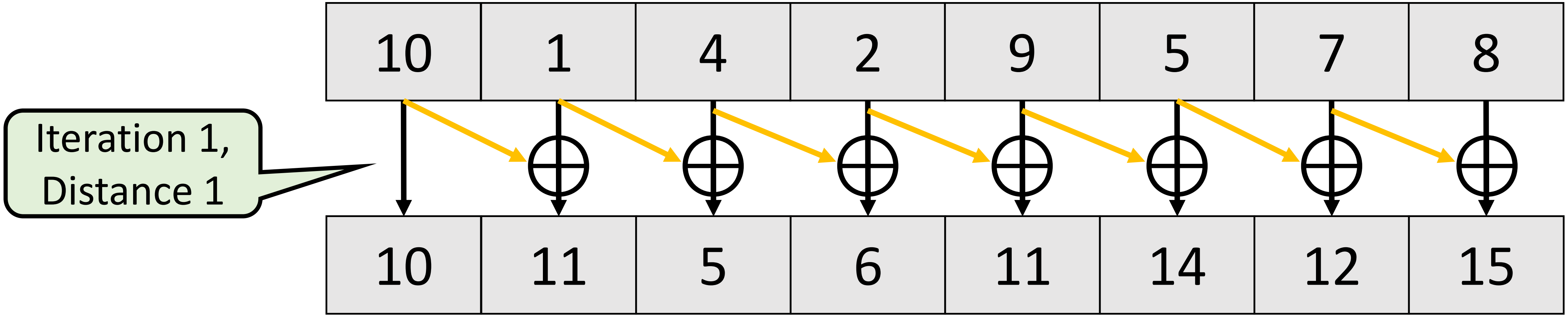
  $= O(n^2)$ operations

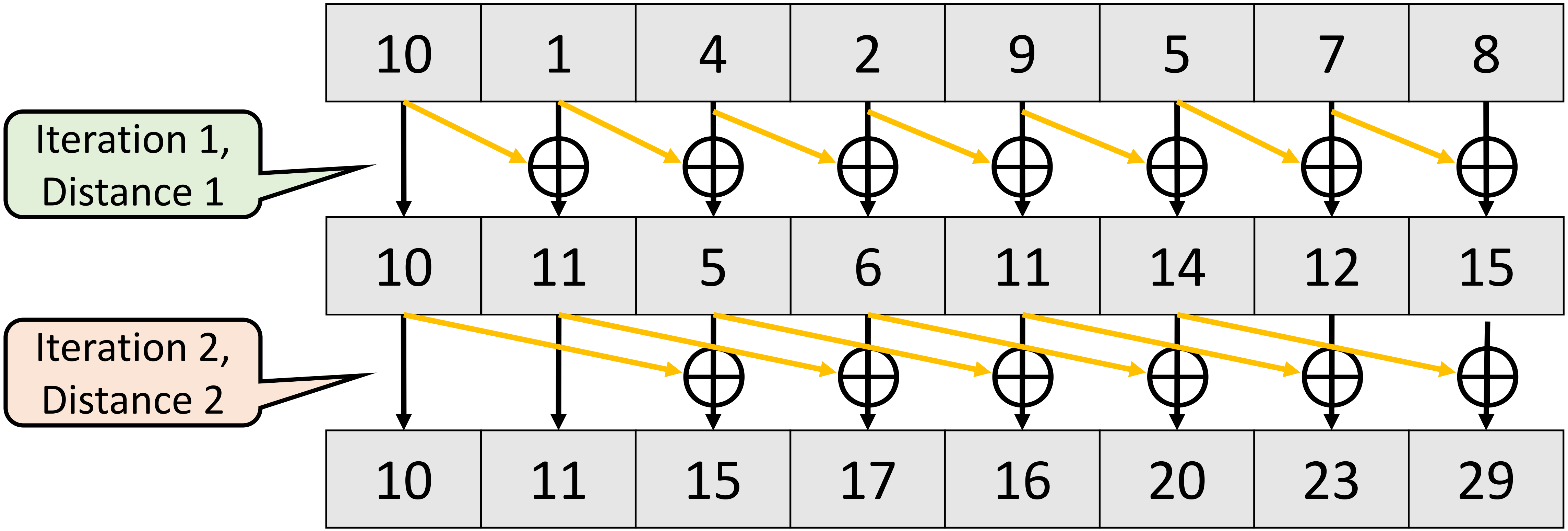
Parallel Inclusive Prefix Sum

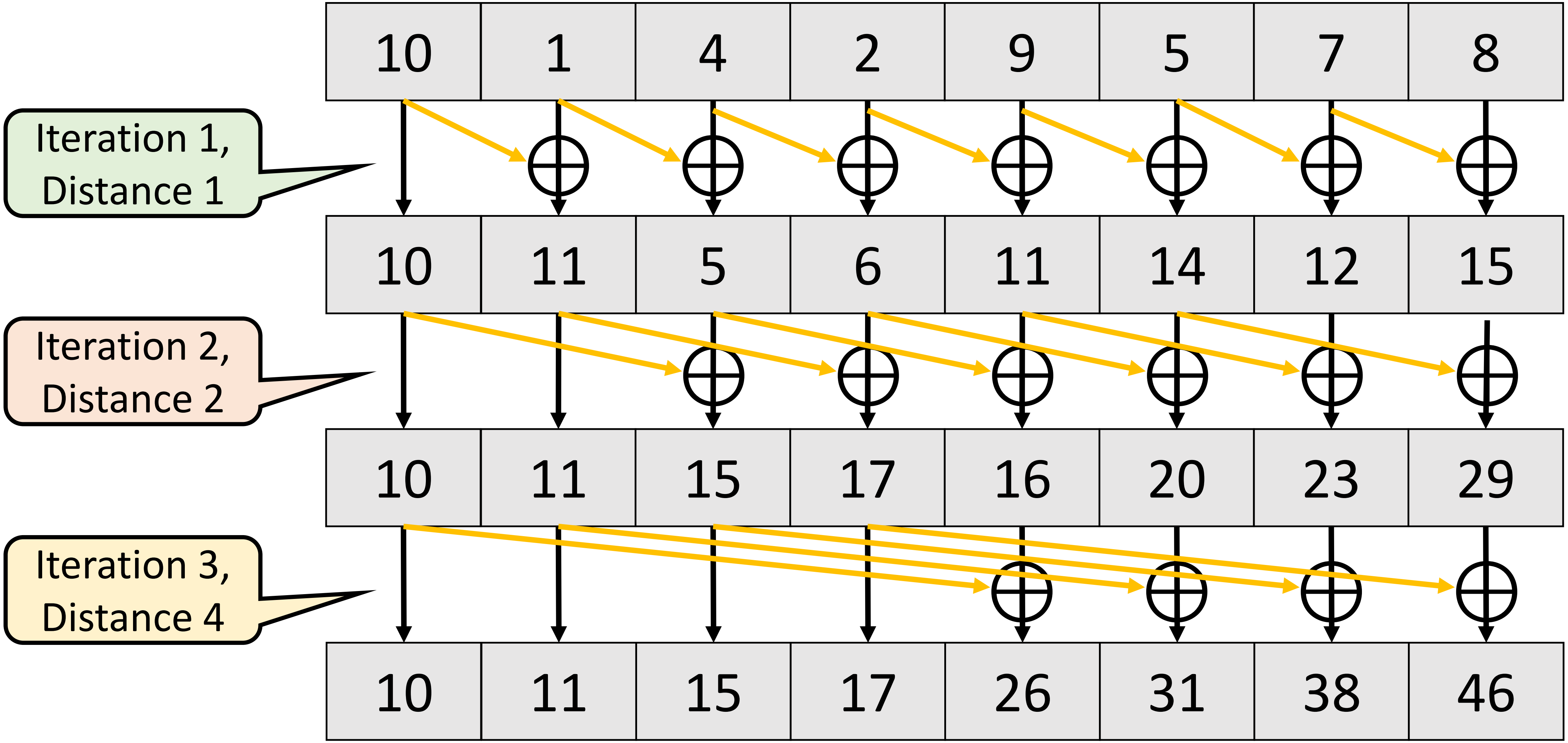
10	1	4	2	9	5	7	8

threads: p
(here p == n, and n = 8)

Ok, so what now?


10
1
4
2
9
5
7
8
Iteration 1,
Distance 1
10
11
5
6
11
14
12
15


10
1
4
2
9
5
7
8
Iteration 1,
Distance 1
10
11
5
6
11
14
12
15
Iteration 2,
Distance 2
10
11
15
17
16
20
23
29


10 1 4 2 9 5 7 8
Iteration 1, Distance 1
10 11 5 6 11 14 12 15
Iteration 2, Distance 2
10 11 15 17 16 20 23 29
Iteration 3, Distance 4
10 11 15 17 26 31 38 46

Algorithm Efficiency

- # of iterations: log n
- First iteration: (n-1) additions
- Second iteration: (n-2) additions
- Third iteration: (n-4) additions
- Last iteration: (n – n/2) additions
- Total additions $= (n-1) + (n-2) + (n-4) + \ \ldots + \left(n\ - \frac{n}{2}\right)$

$$= n \log n\ - \left(1 + 2 + 4 + \cdots + \frac{\mathrm{n}}{2}\right)$$

$$= n \log n\ - (n-1)\ = \mathrm{O}\ (n \log n)$$

Algorithm Efficiency

- **Work** = O $(n \log n)$
  - Remember Work for the sequential algorithm was O(n)
  - For large n, $\log n$ can be a non-trivial factor

Hillis and Steele

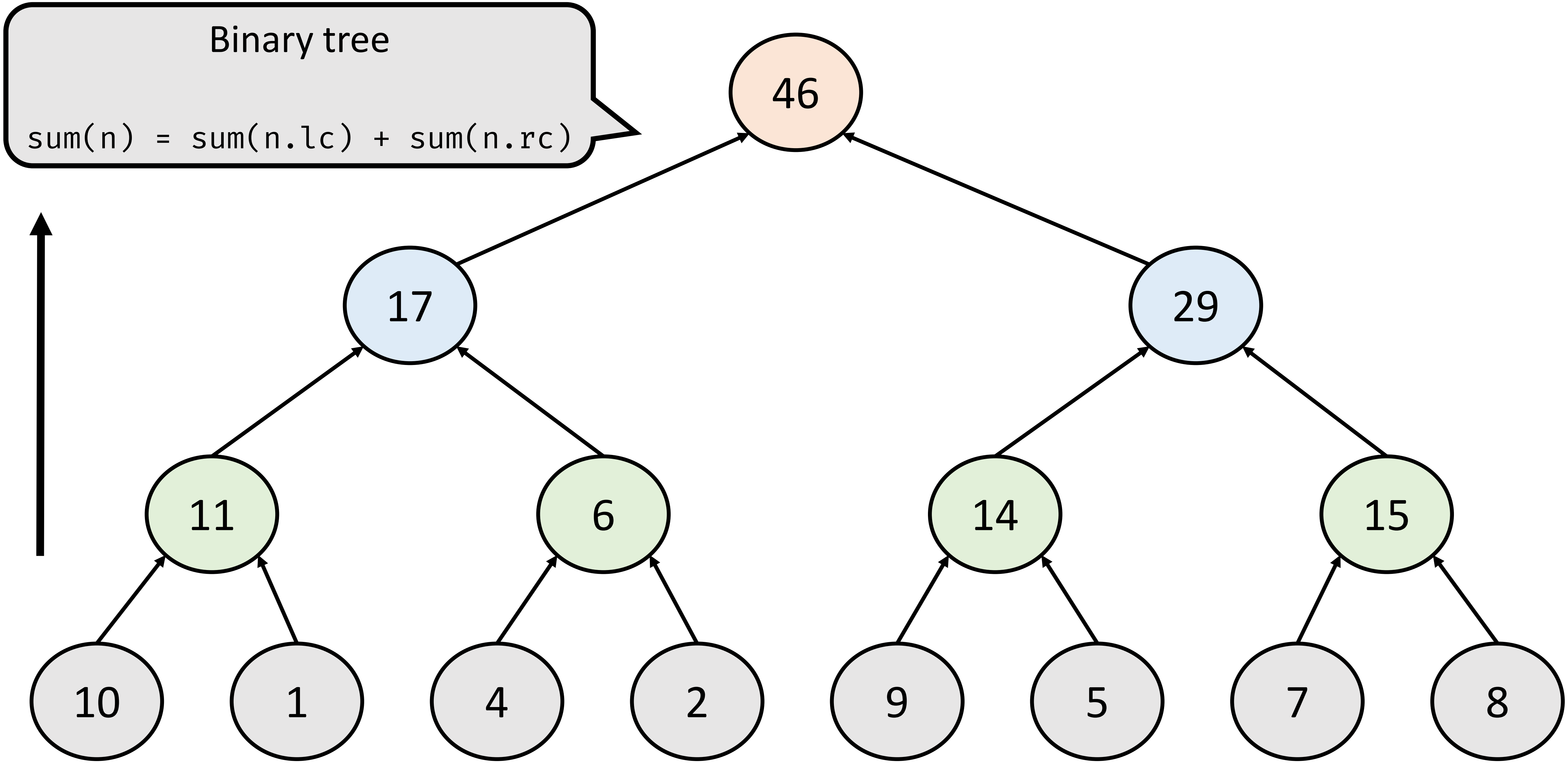
```
for i = 0 to ⌈log n-1⌉ do
    for j = 2^i to n-1 in parallel do
        A [j] = A[j] + A[j-2^i]
```

Asymptotic complexity O(log n)

Algorithm With Improved Work-Efficiency

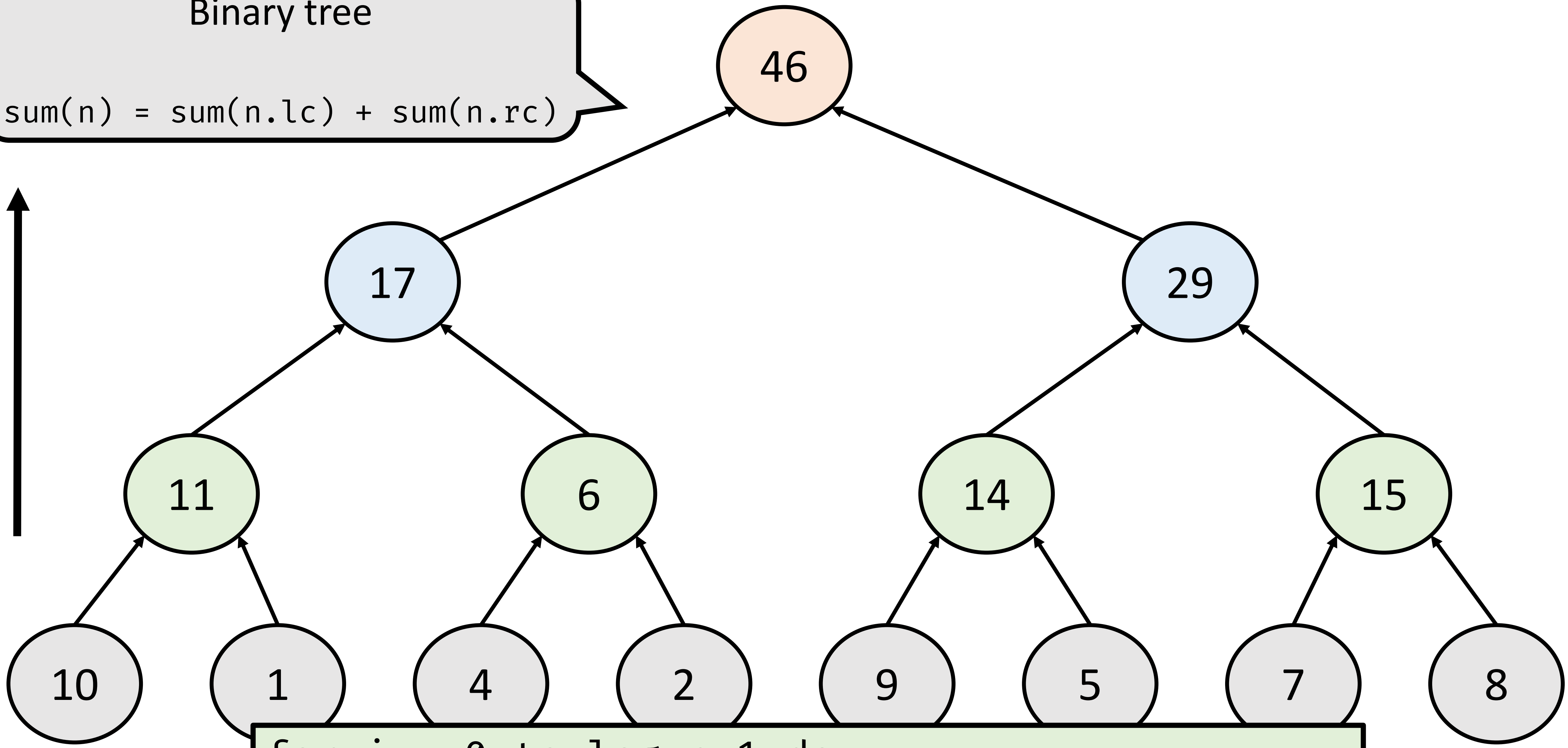
Guy Blelloch

10	1	4	2	9	5	7	8

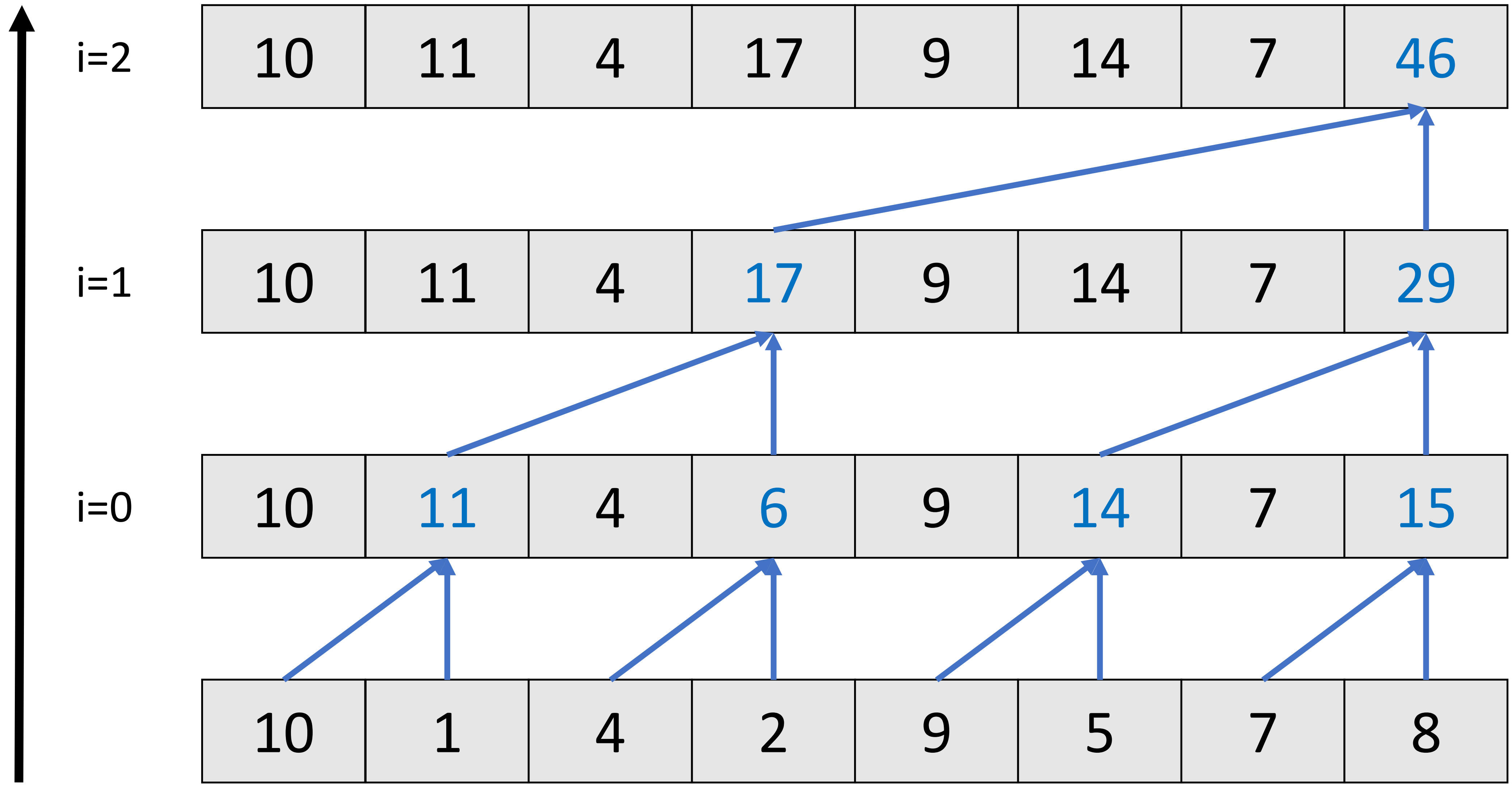

Binary tree
sum(n) = sum(n.lc) + sum(n.rc)
46
17
29
11
6
14
15
10
1
4
2
9
5
7
8

Binary tree

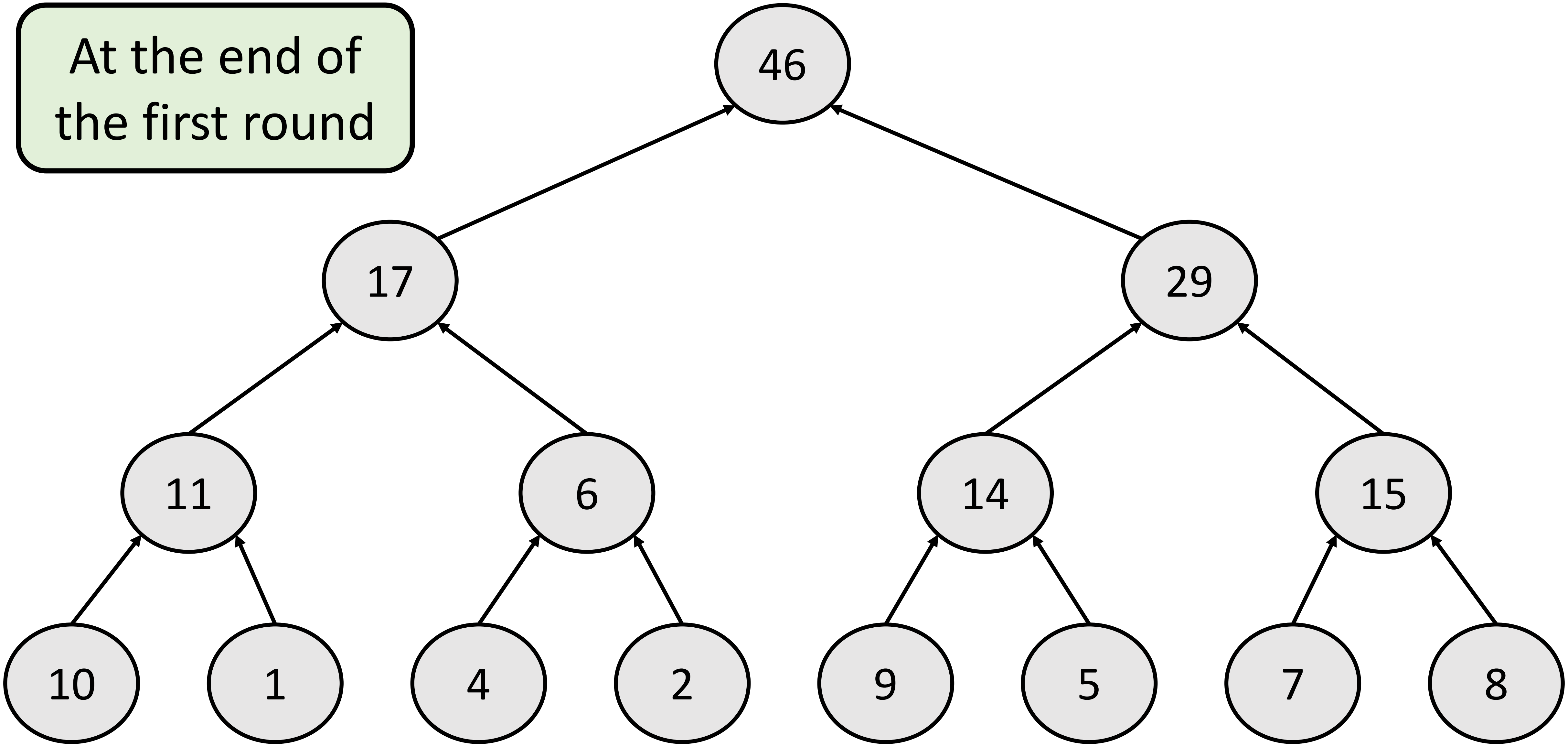
```
sum(n) = sum(n.lc) + sum(n.rc)
```

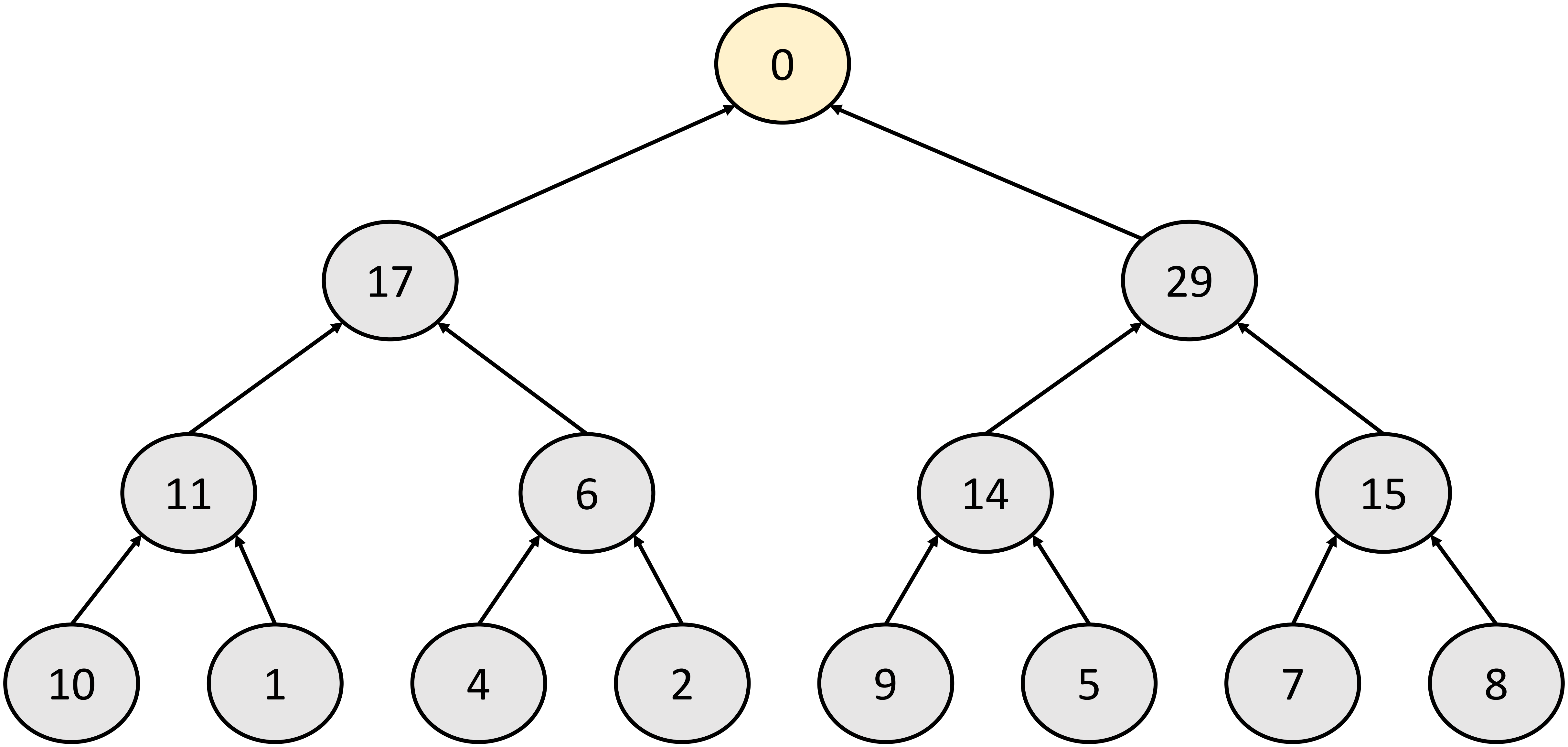


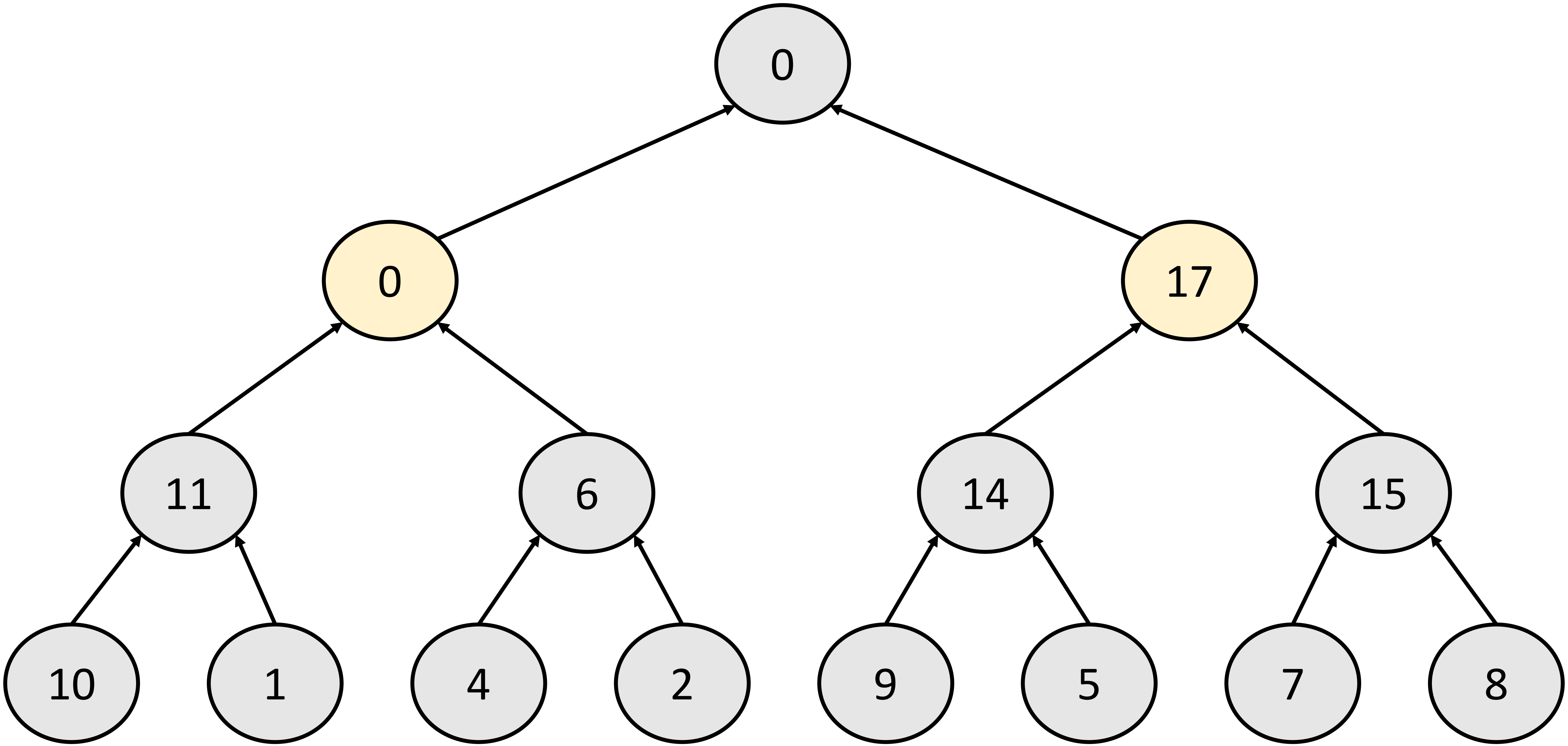

```
for i = 0 to log n-1 do
    for j = 0 to n-1 by 2^(i+1) in parallel do
        a[j+2^(i+1)-1] = a[j+2^i-1] + a[j+2^(i+1)-1]
```

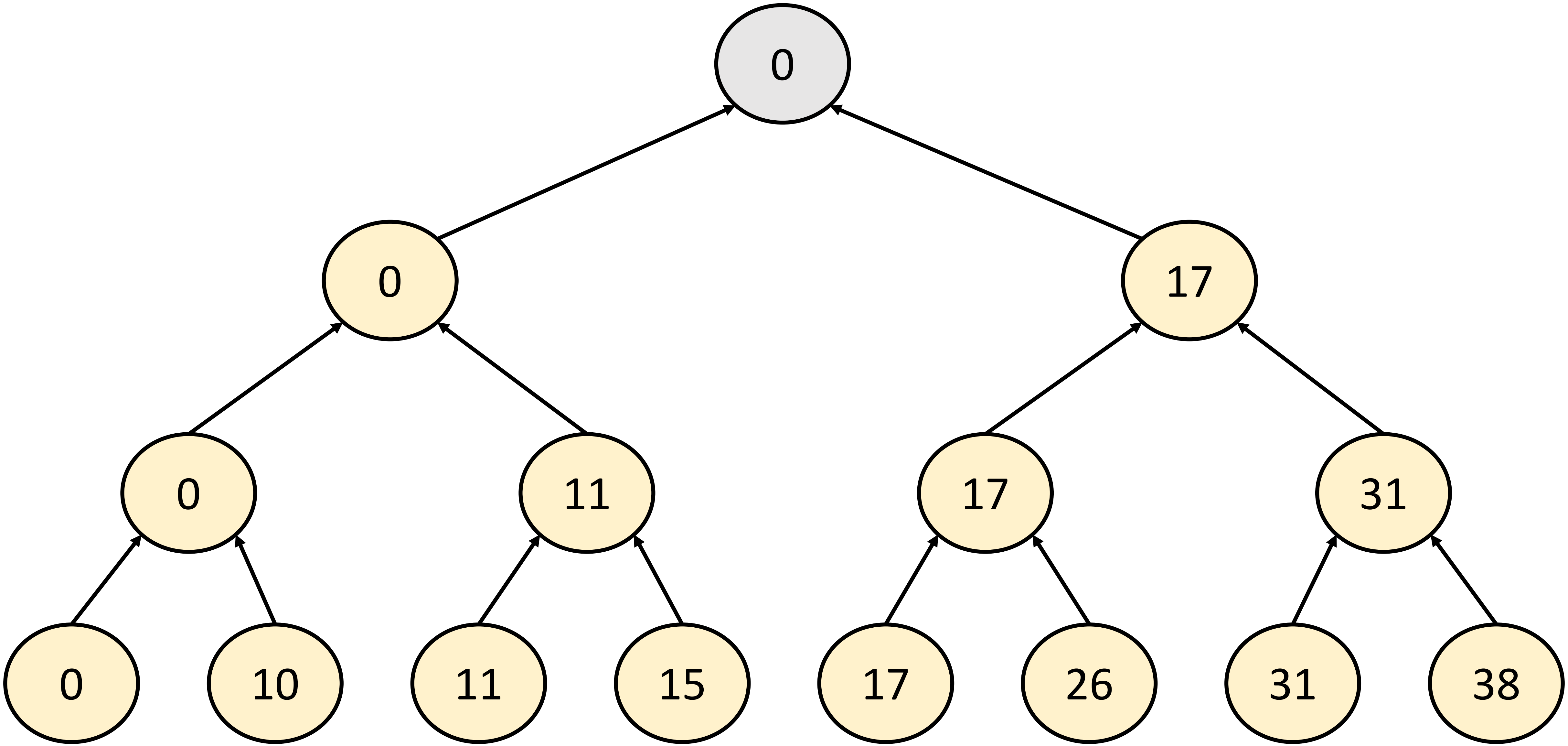

i=2
10 11 4 17 9 14 7 46
i=1
10 11 4 17 9 14 7 29
i=0
10 11 4 6 9 14 7 15
10 1 4 2 9 5 7 8

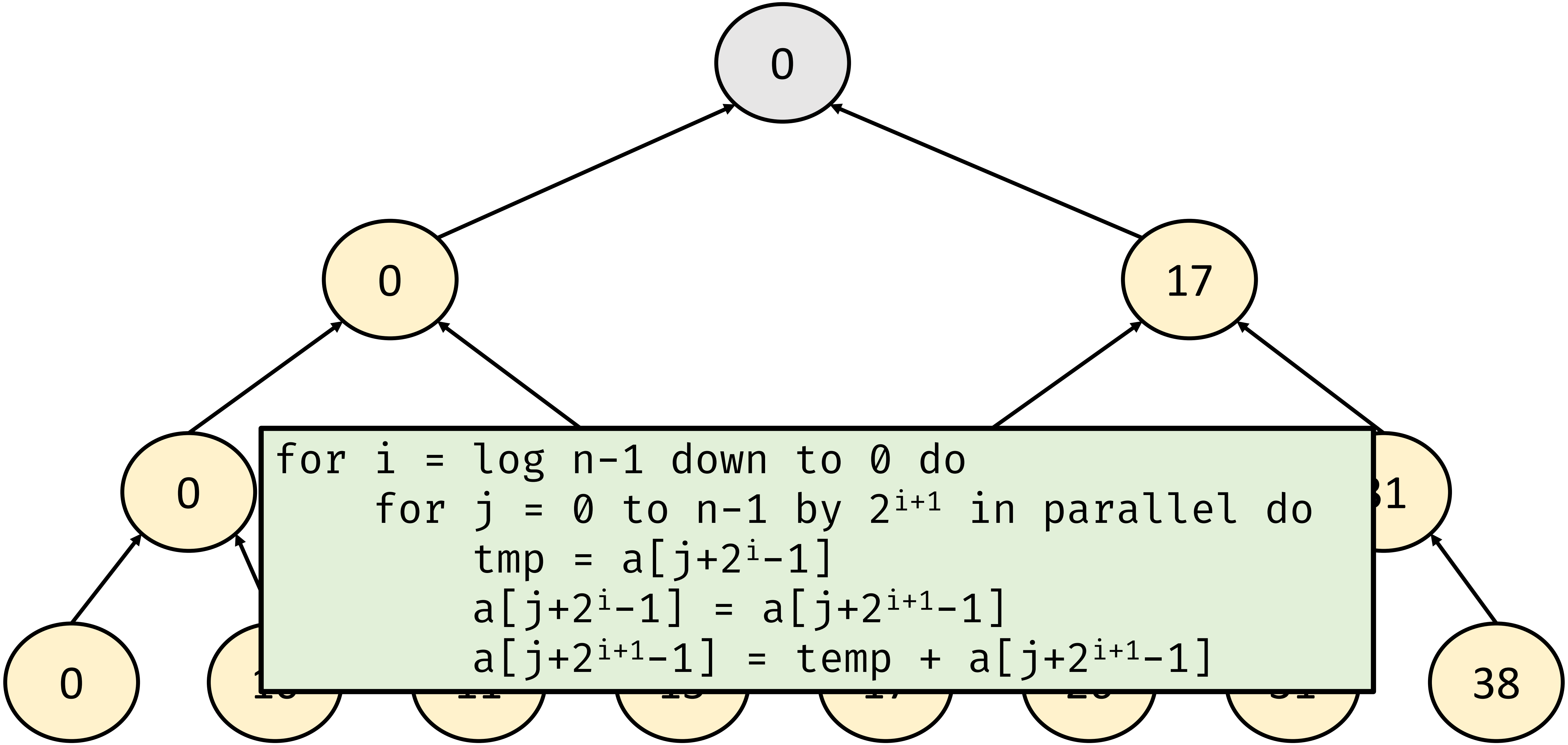
At the end of the first round



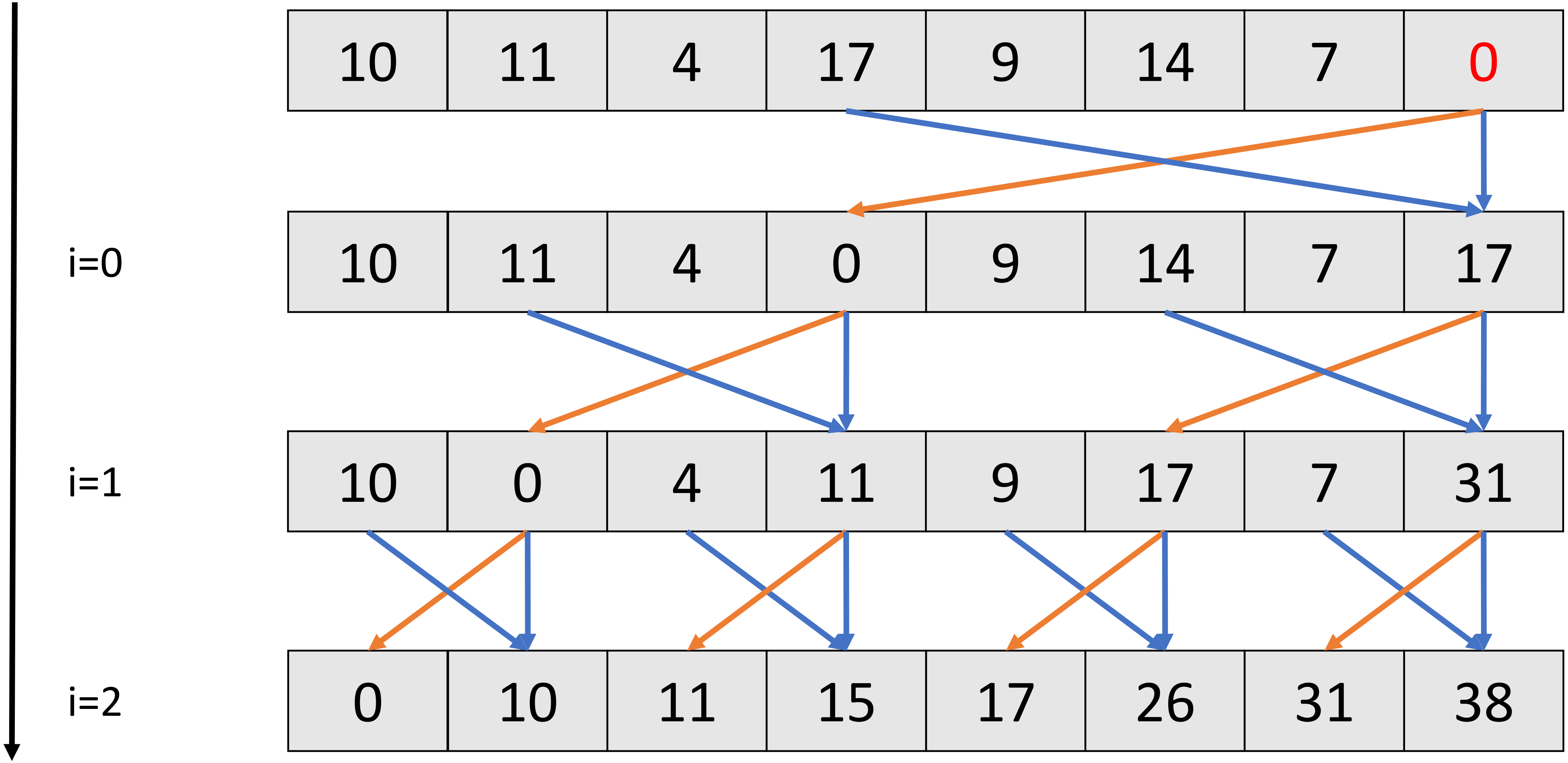

0
17
29
11
6
14
15
10
1
4
2
9
5
7
8


0
0
17
11
6
14
15
10
1
4
2
9
5
7
8

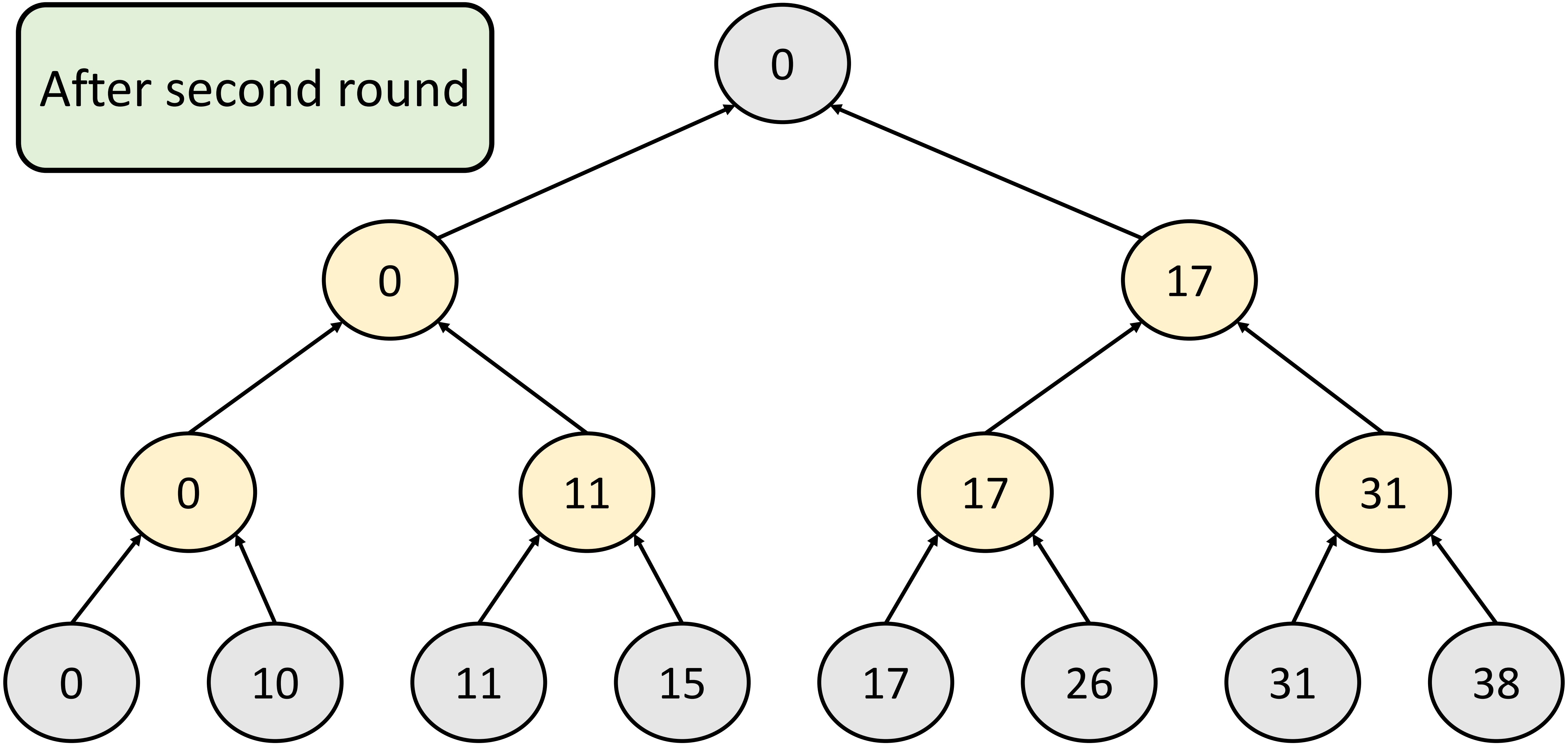

0
0
17
0
11
17
31
0
10
11
15
17
26
31
38




```
for i = log n-1 down to 0 do
    for j = 0 to n-1 by 2^(i+1) in parallel do
        tmp = a[j+2^i-1]
        a[j+2^i-1] = a[j+2^(i+1)-1]
        a[j+2^(i+1)-1] = temp + a[j+2^(i+1)-1]
```


10 11 4 17 9 14 7 0
i=0
10 11 4 0 9 14 7 17
i=1
10 0 4 11 9 17 7 31
i=2
0 10 11 15 17 26 31 38

After second round



Algorithm Efficiency

Asymptotic complexity O(log n)

```
for i = 0 to log n-1 do
    for j = 0 to n-1 by 2^i+1 in parallel do
        a[j+2^(i+1)-1] = a[j+2^i-1] + a[j+2^(i+1)-1]
```

⇩

```
for i = log n-1 down to 0 do
    for j = 0 to n-1 by 2^(i+1) in parallel do
        tmp = a[j+2^i-1]
        a[j+2^i-1] = a[j+2^(i+1)-1]
        a[j+2^(i+1)-1] = temp + a[j+2^(i+1)-1]
```

Algorithm Efficiency

- # of iterations: log n in each pass
- Number of addition operations in first pass: $\frac{n}{2} + \frac{n}{4} + \cdots + 2 + 1$
- Number of addition operations in second pass: $1 + 2 + \cdots + \frac{n}{2}$
- Total additions $= (\mathrm{n} - 1) + (\mathrm{n} - 1) = \mathbf{2}(\mathrm{n} - 1)$

  = O (n)

Benefits from parallelism can overcome the constant factor increase in computation

References

- Yong Cao. Parallel Prefix Sum – Scan.
- G. Blelloch. Prefix Sums and Their Applications.
- Th. Ottmann. Parallel Prefix Computation.