Implementing A Heap

Implementing a Heap

- Although it is helpful to think of a heap as a linked structure when visualizing the enqueue and dequeue operations, it is often implemented with an array by using remarkable properties of heaps.

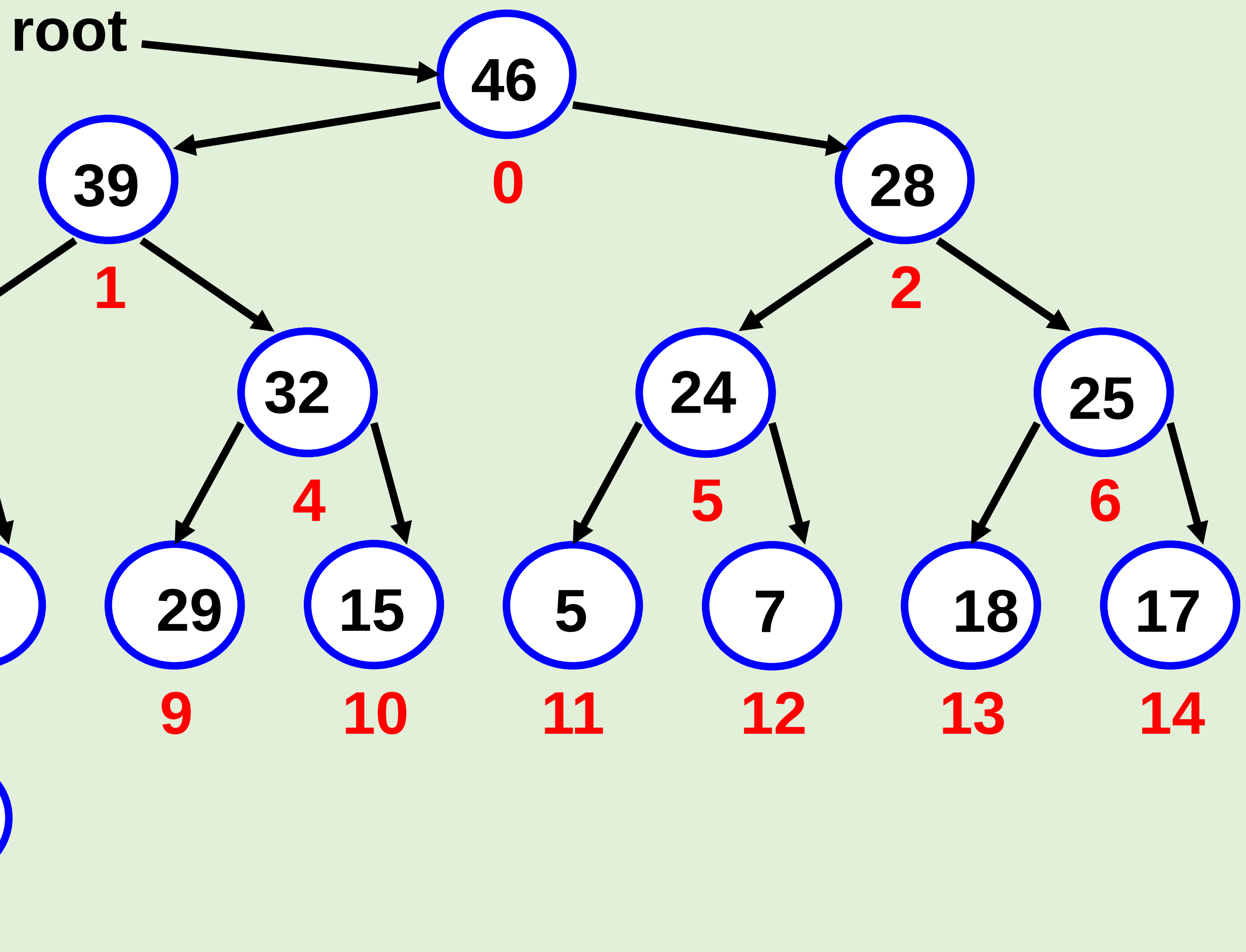
  Left child [index] = 2 * parent[index] + 1

  Right child [index\ = 2 * parent[indes] + 2

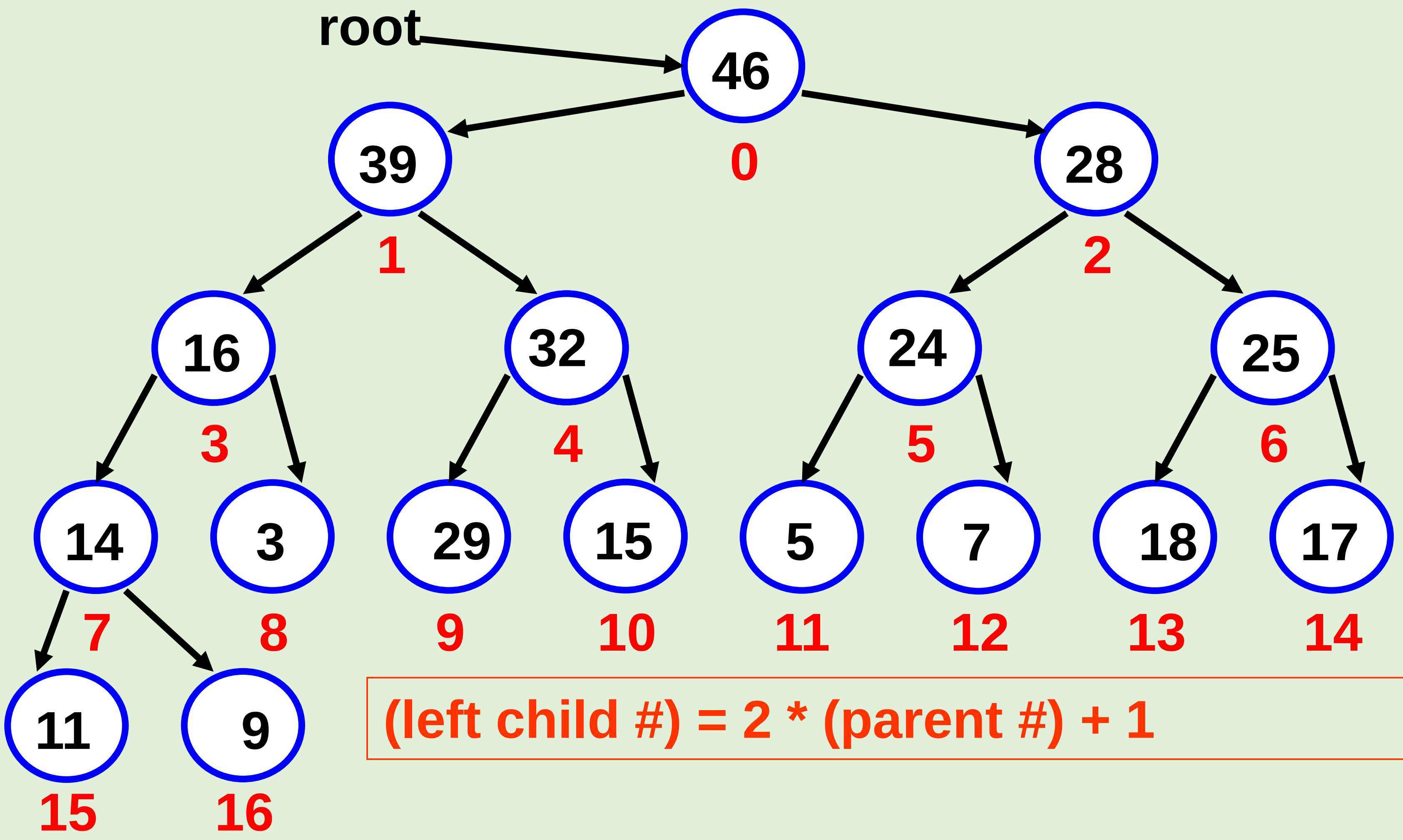
  Parent[index] = (child [index] – 1) / 2

- Let’s number the nodes of a heap, starting with 0, going top to bottom and left to right…

Implementing a Heap (cont.)

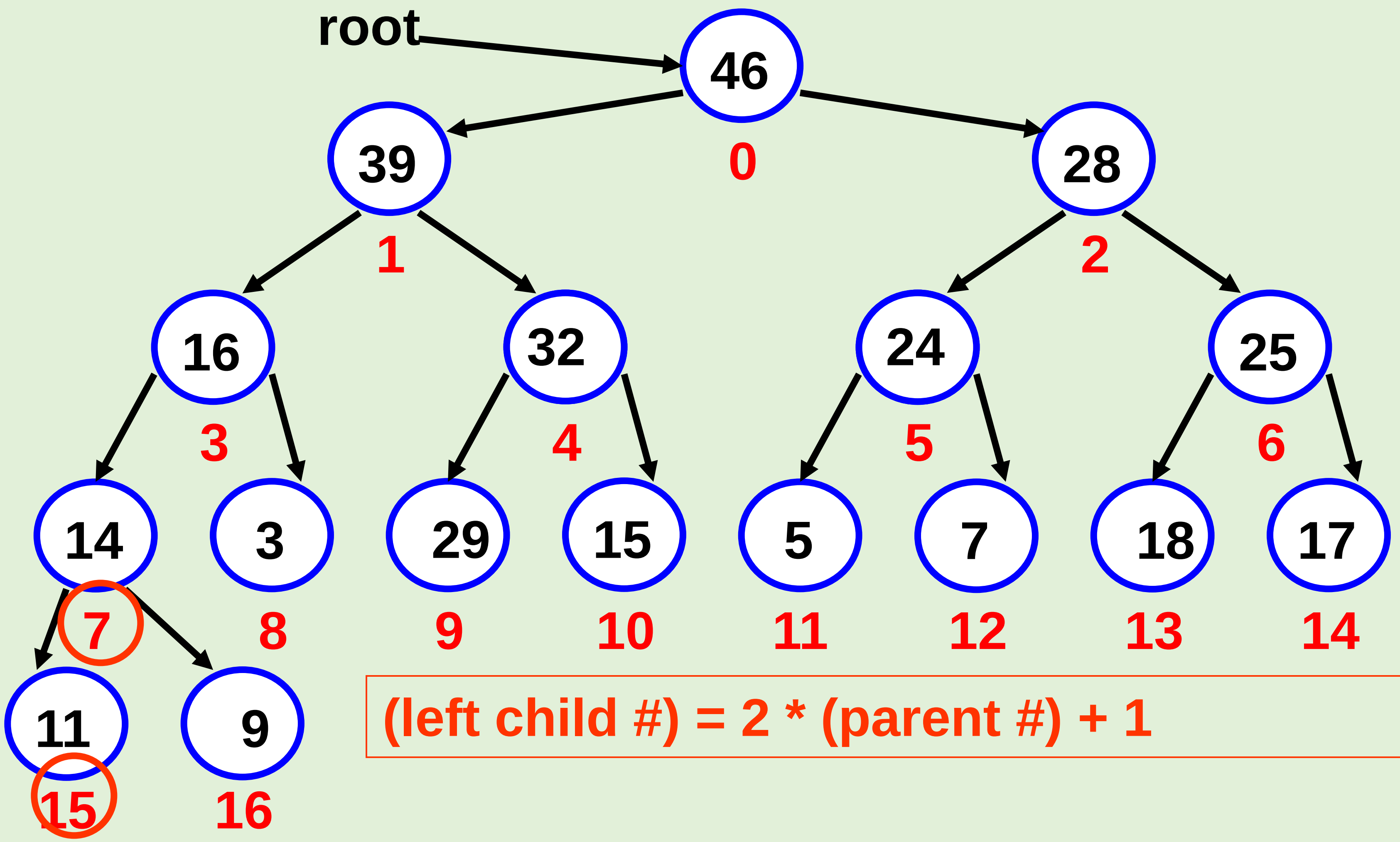


Heap Properties



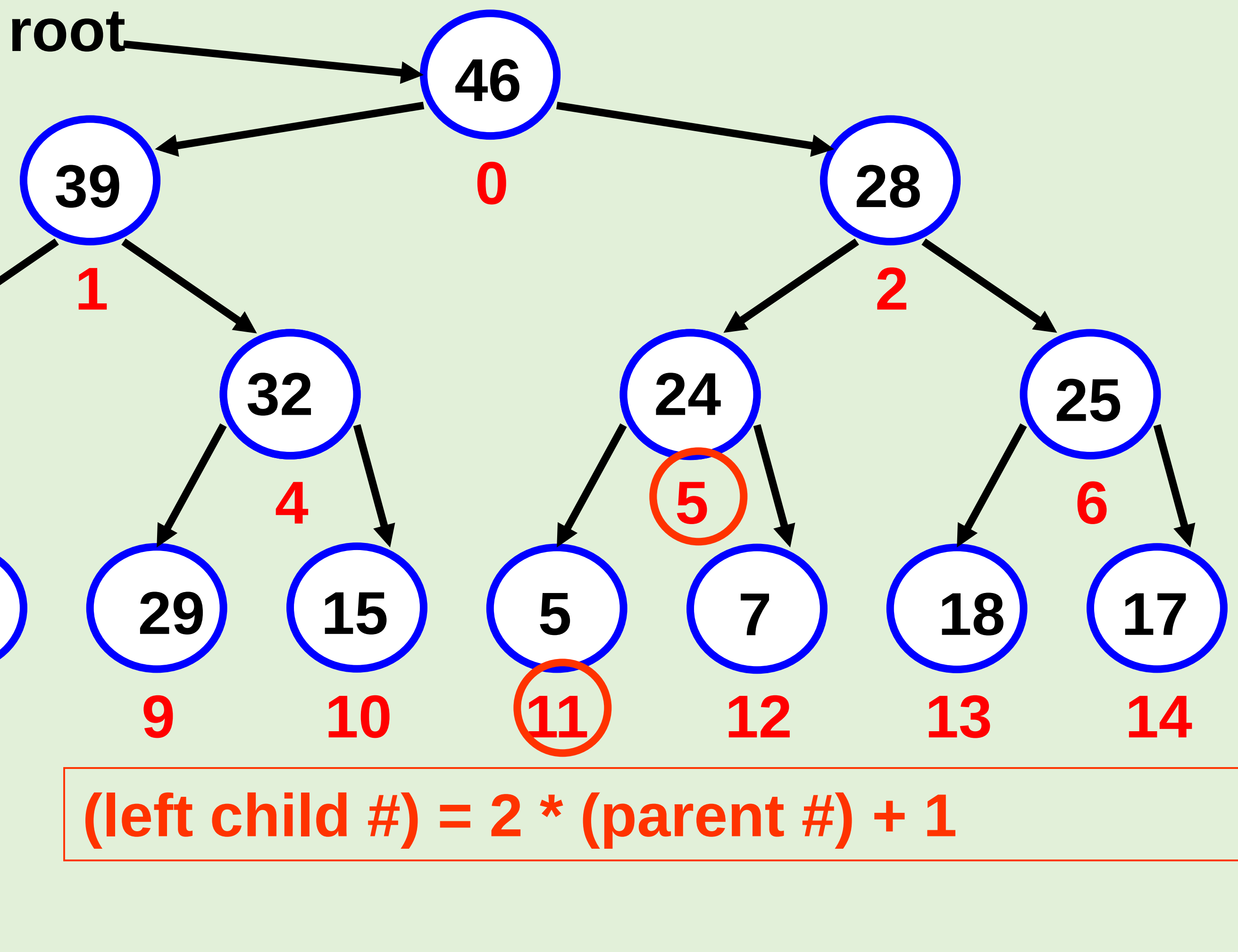
(left child #) = 2 * (parent #) + 1

Heap Properties (cont.)



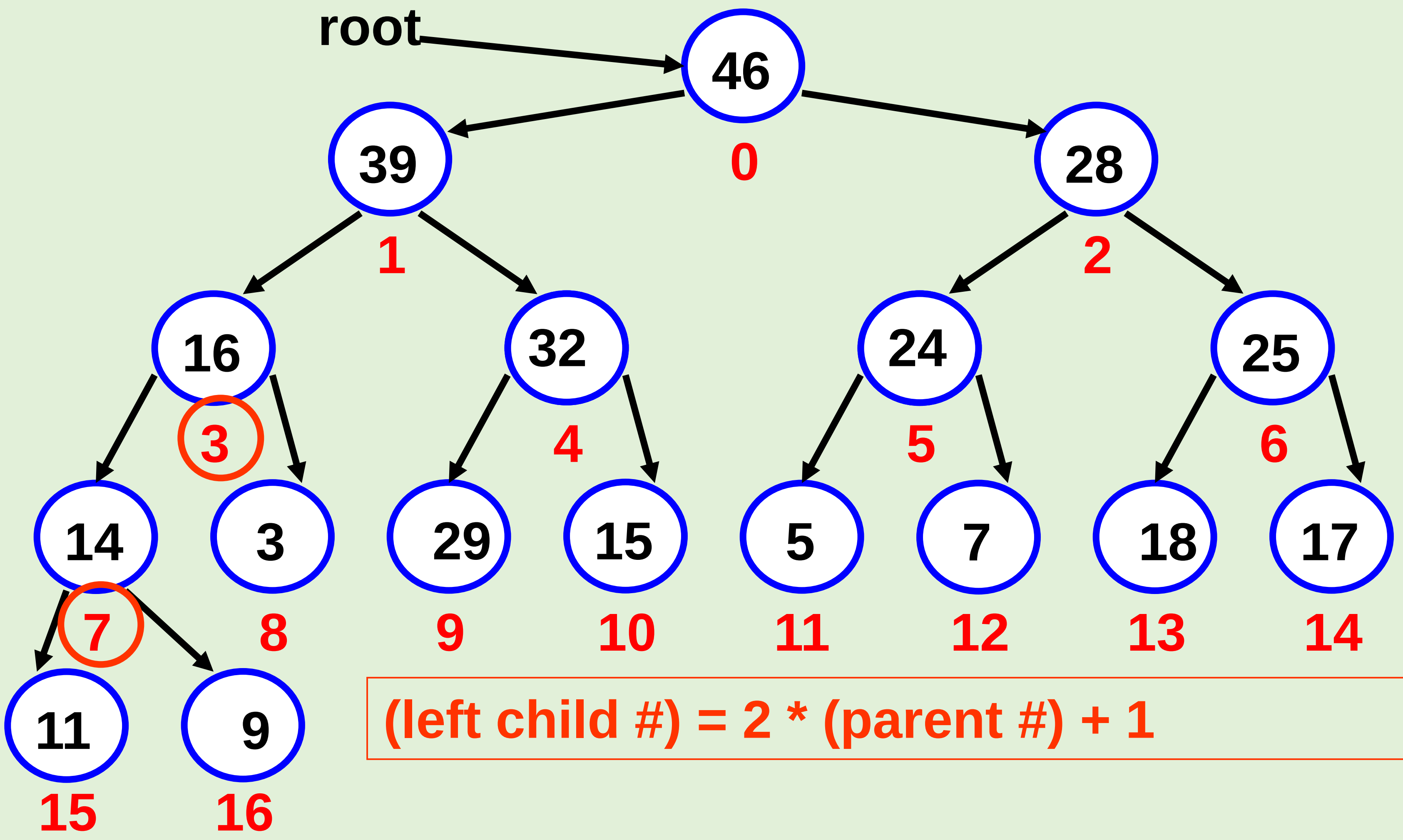
(left child #) = 2 * (parent #) + 1

Heap Properties (cont.)



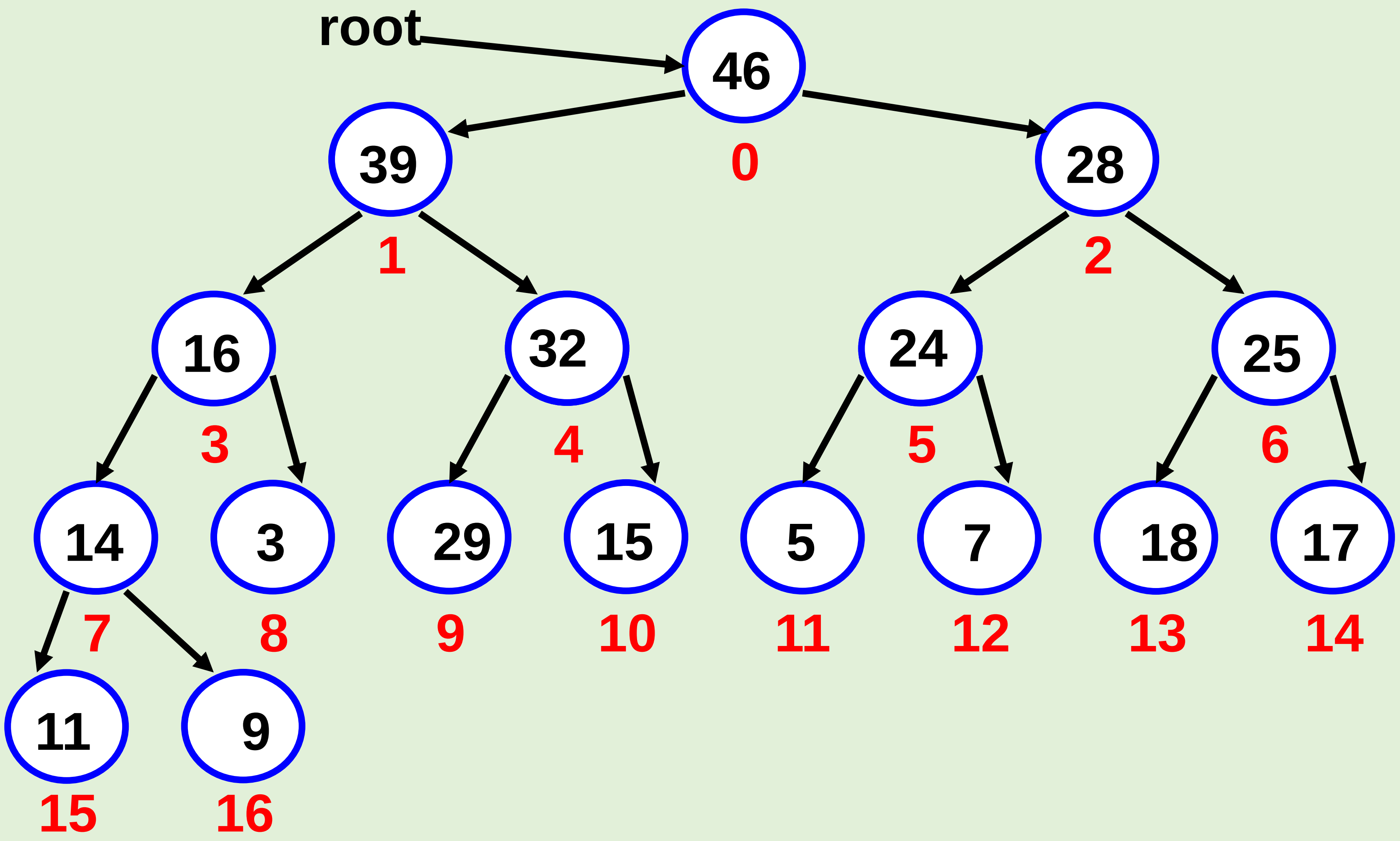
(left child #) = 2 * (parent #) + 1

Heap Properties (cont.)

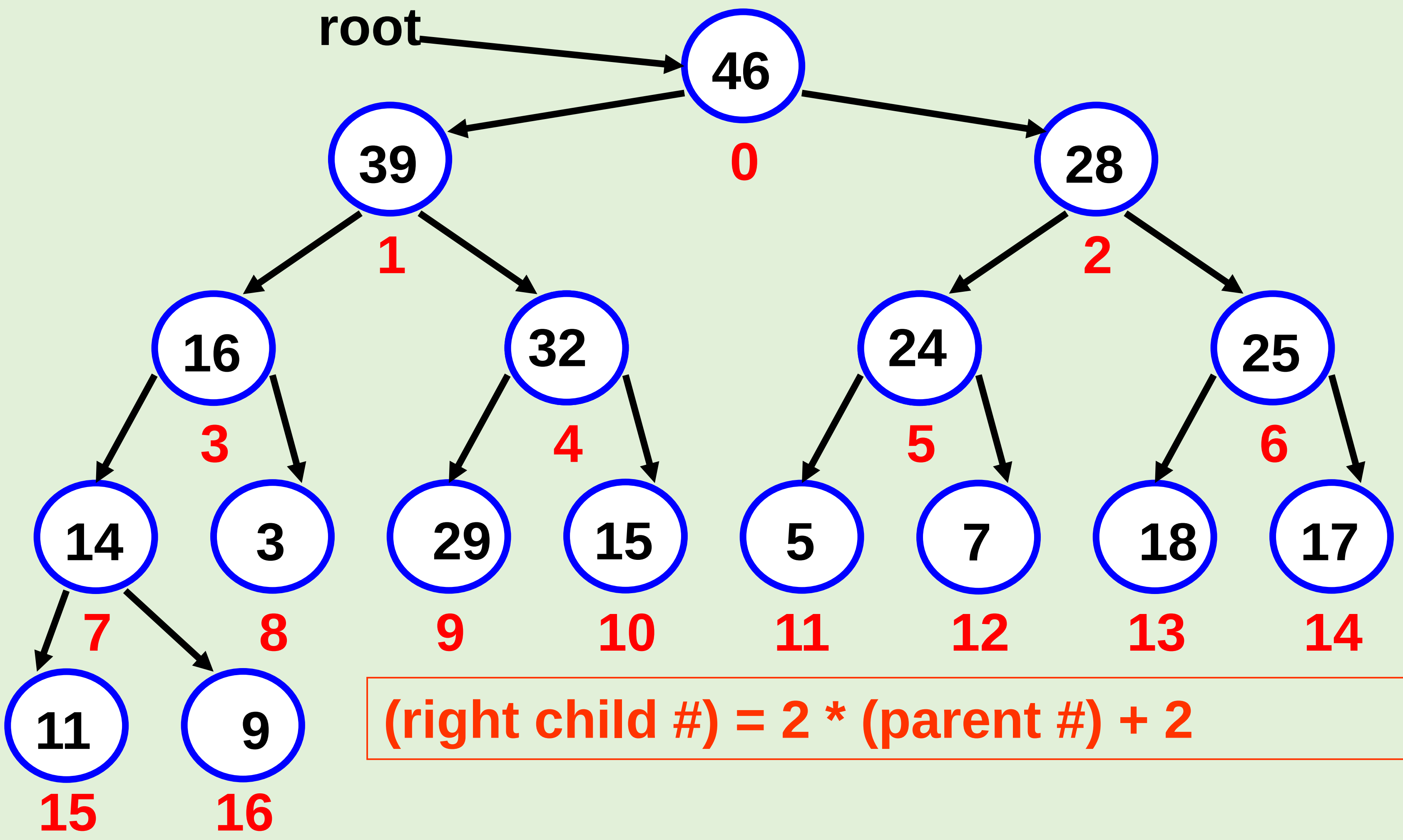


(left child #) = 2 * (parent #) + 1

Heap Properties (cont.)

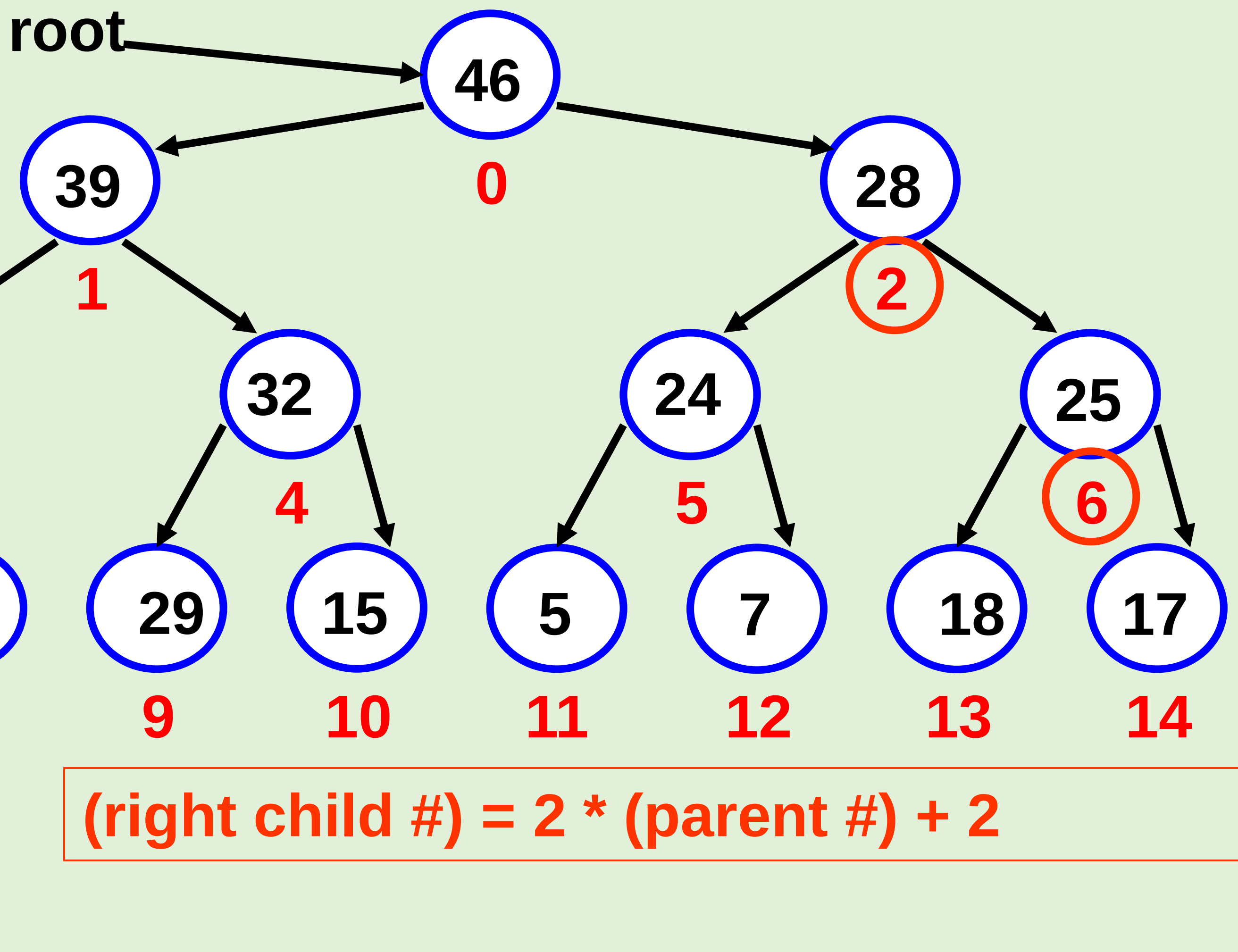


Heap Properties (cont.)



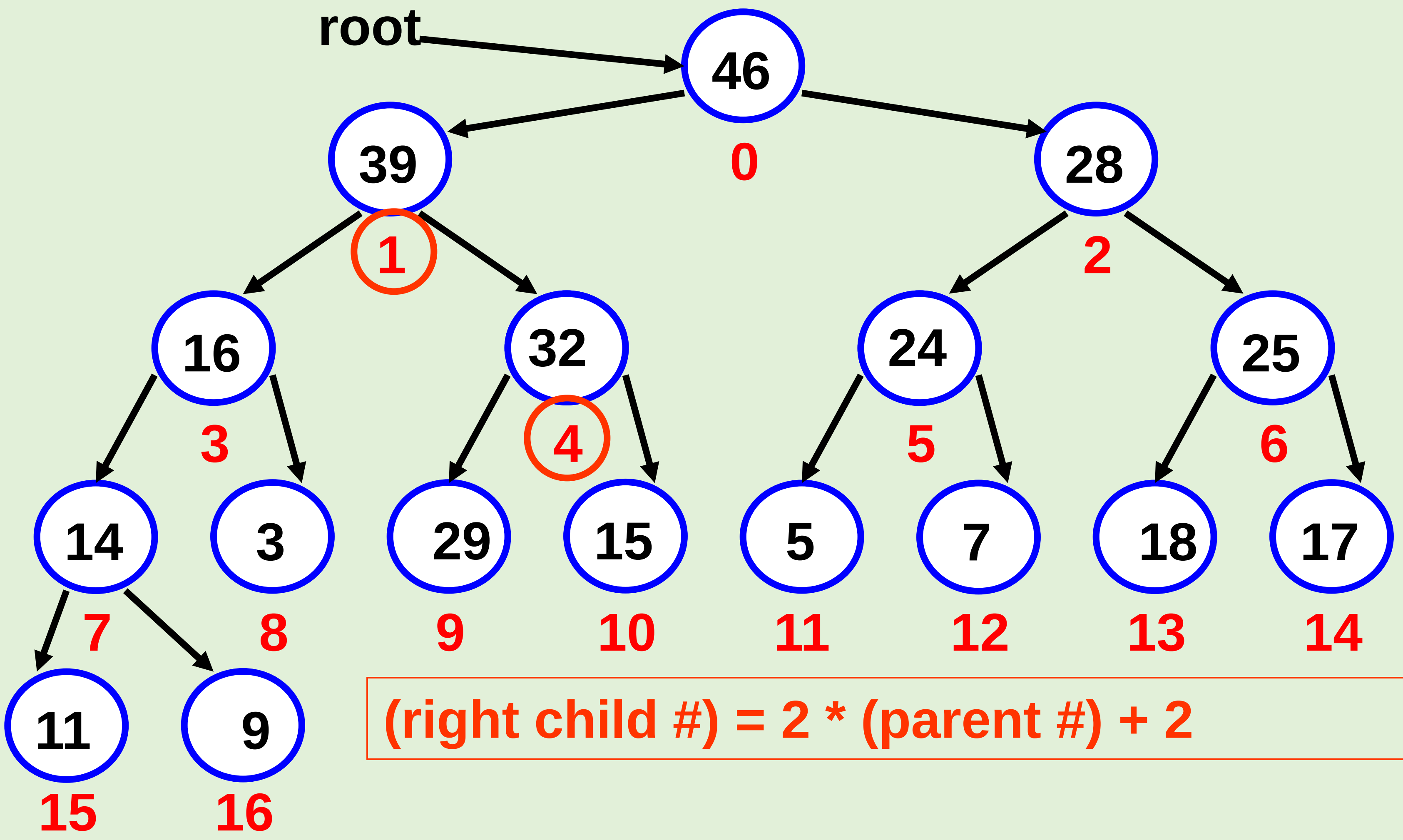
(right child #) = 2 * (parent #) + 2

Heap Properties (cont.)



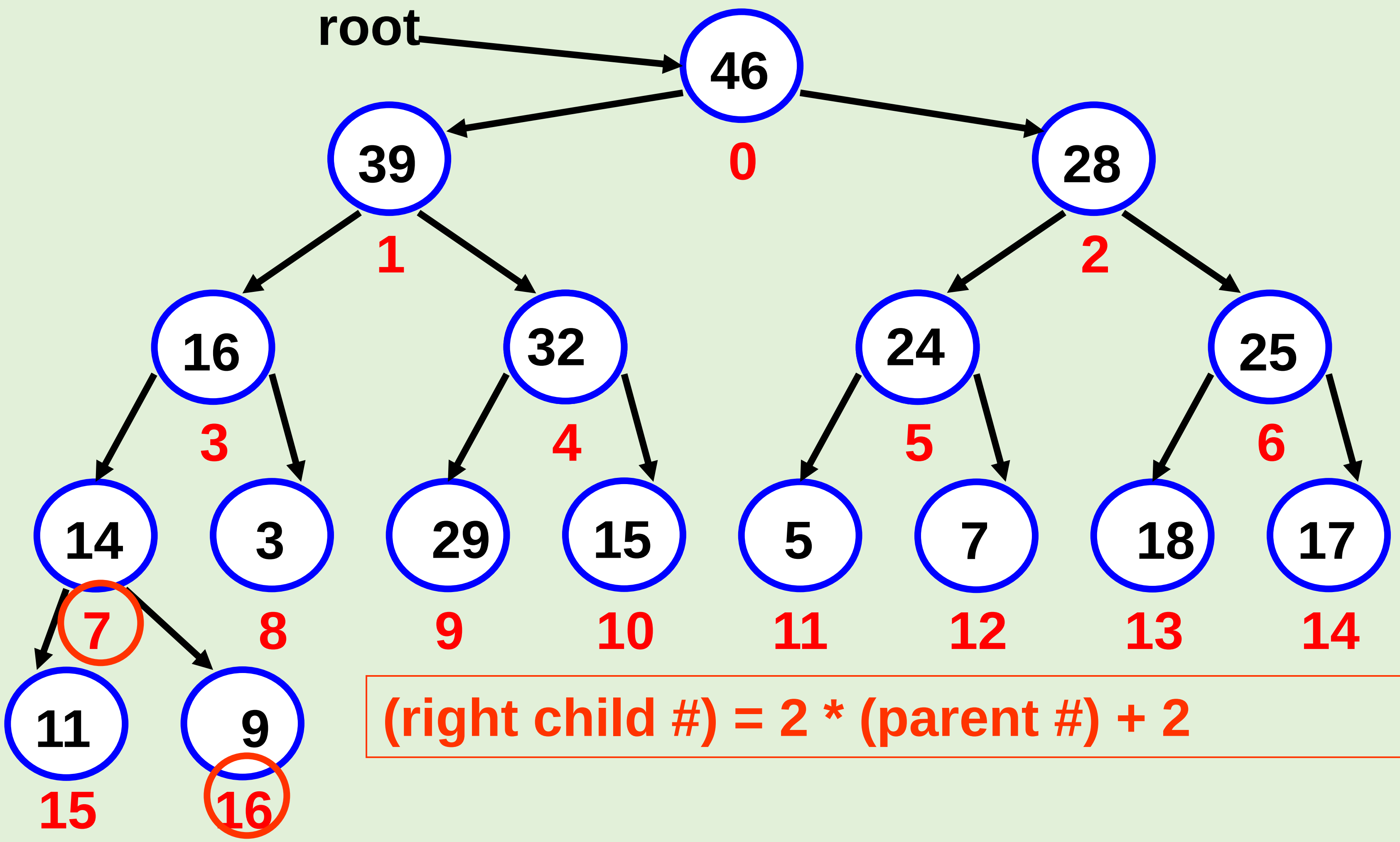
(right child #) = 2 * (parent #) + 2

Heap Properties (cont.)



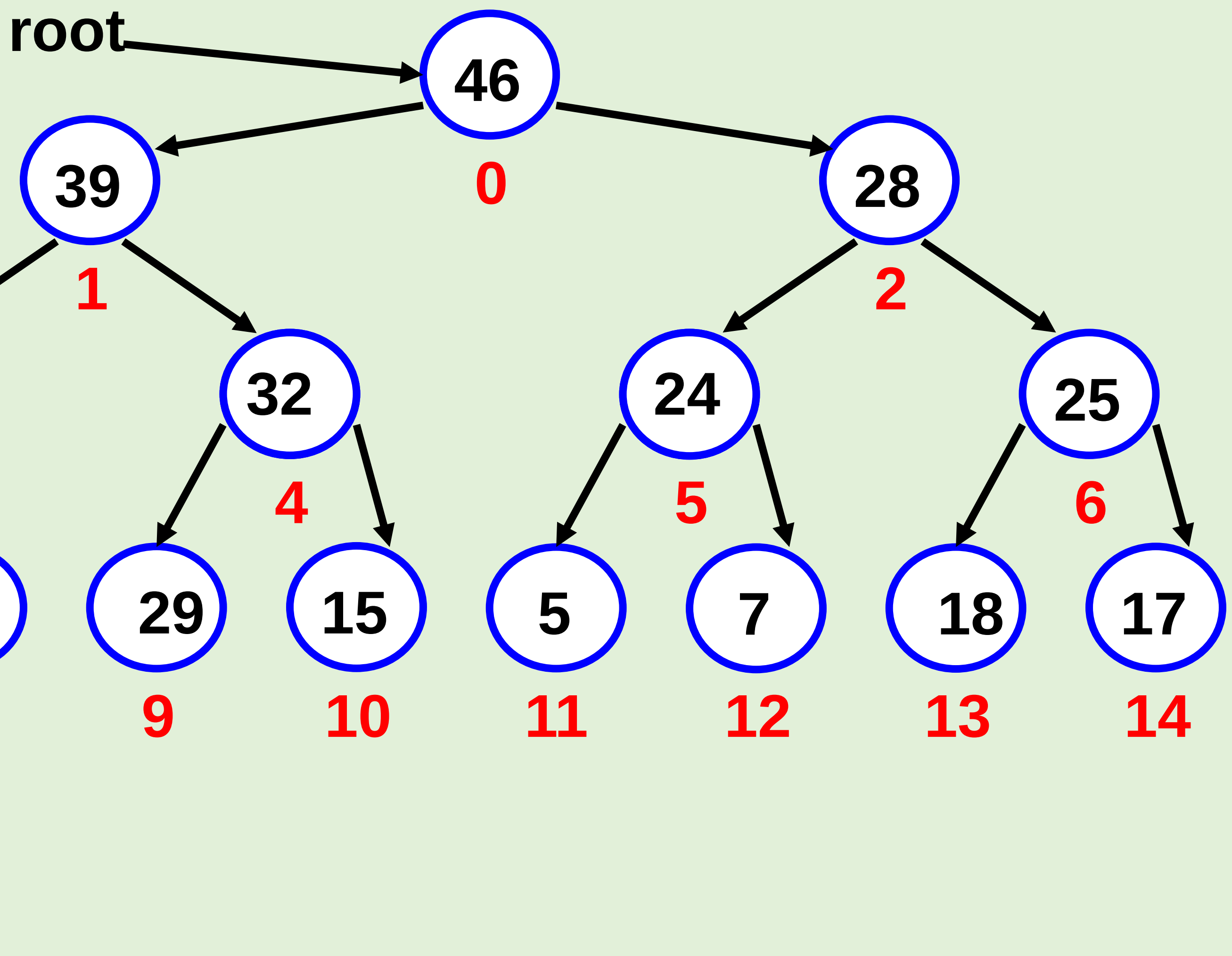
(right child #) = 2 * (parent #) + 2

Heap Properties (cont.)

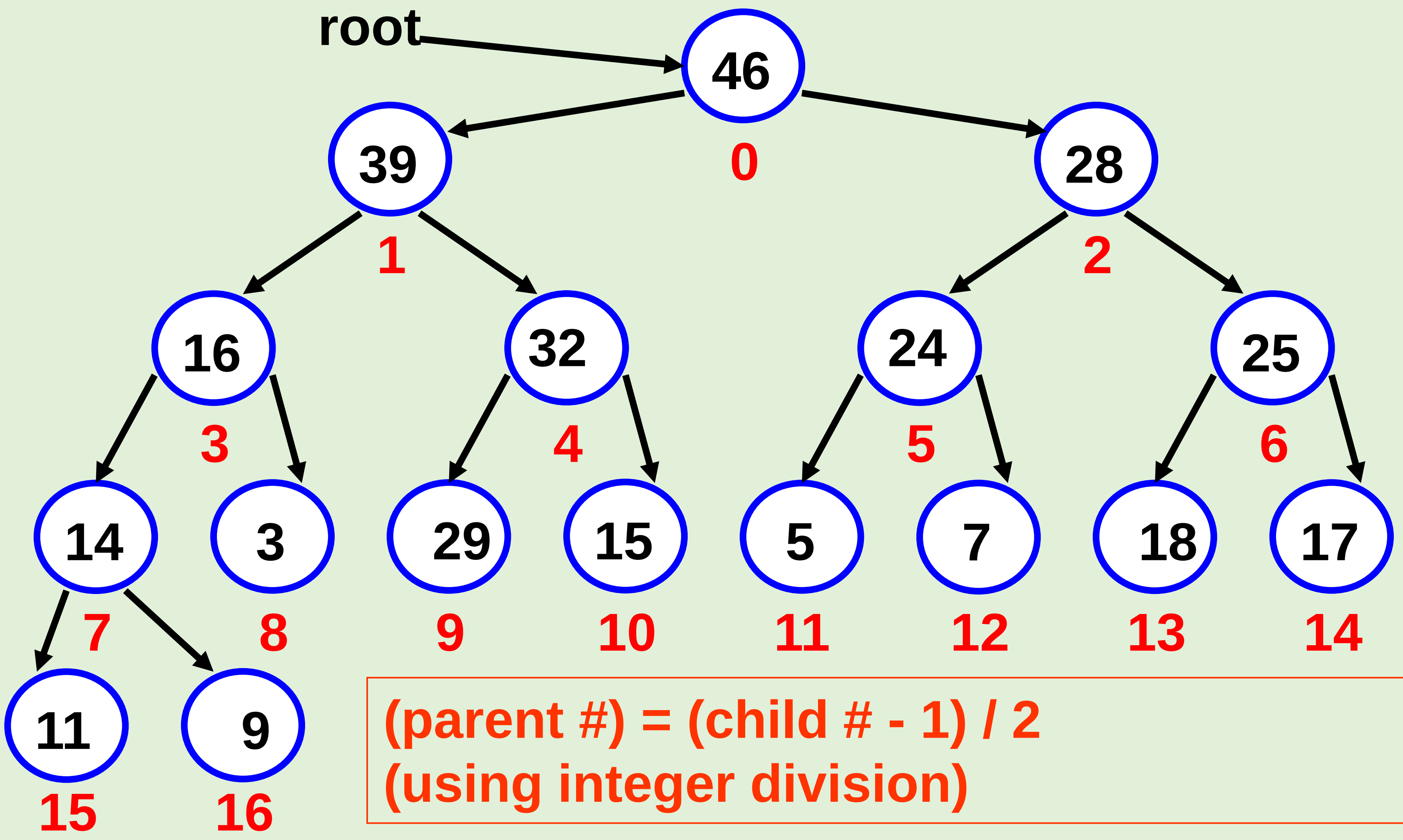


(right child #) = 2 * (parent #) + 2

Heap Properties (cont.)

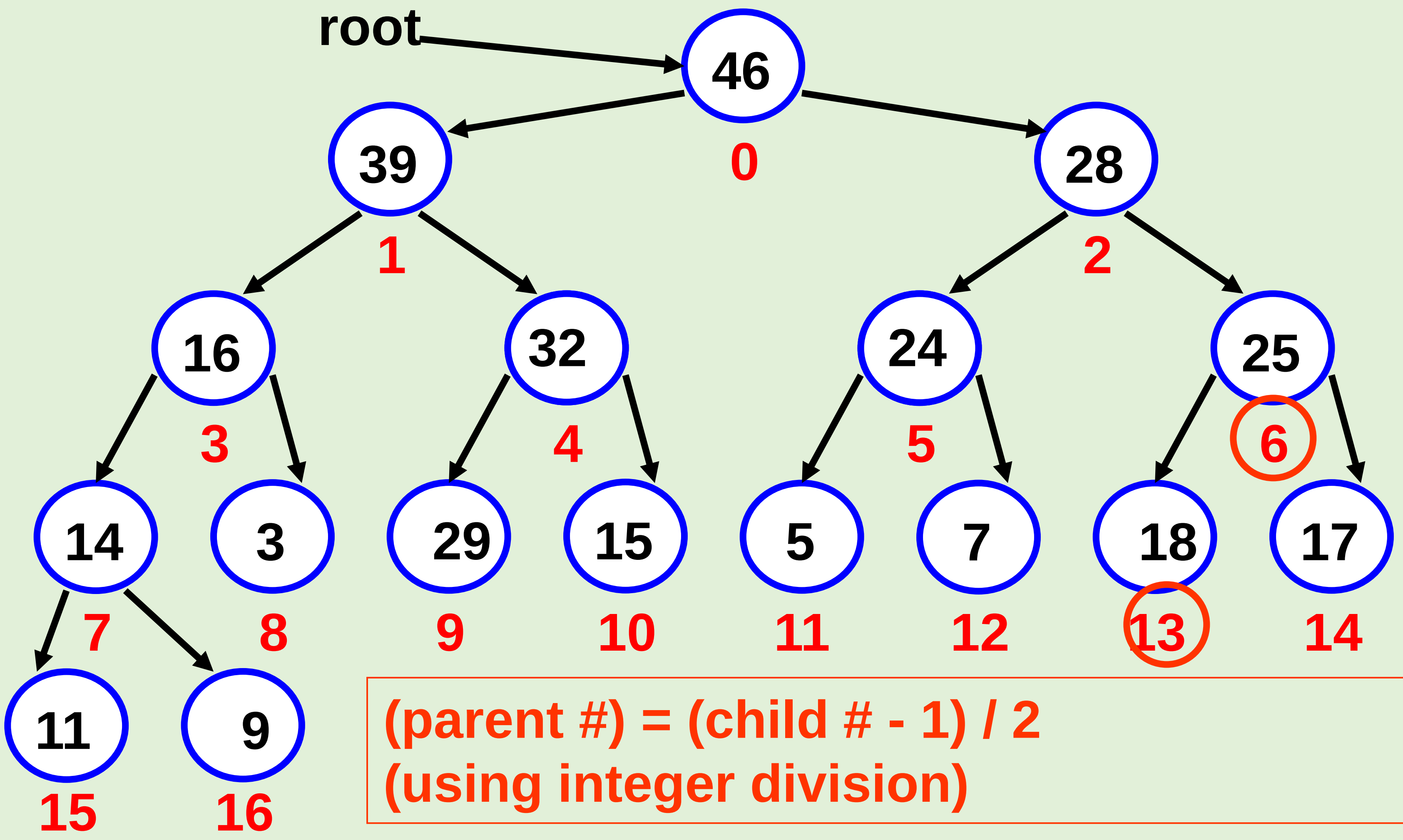


Heap Properties (cont.)



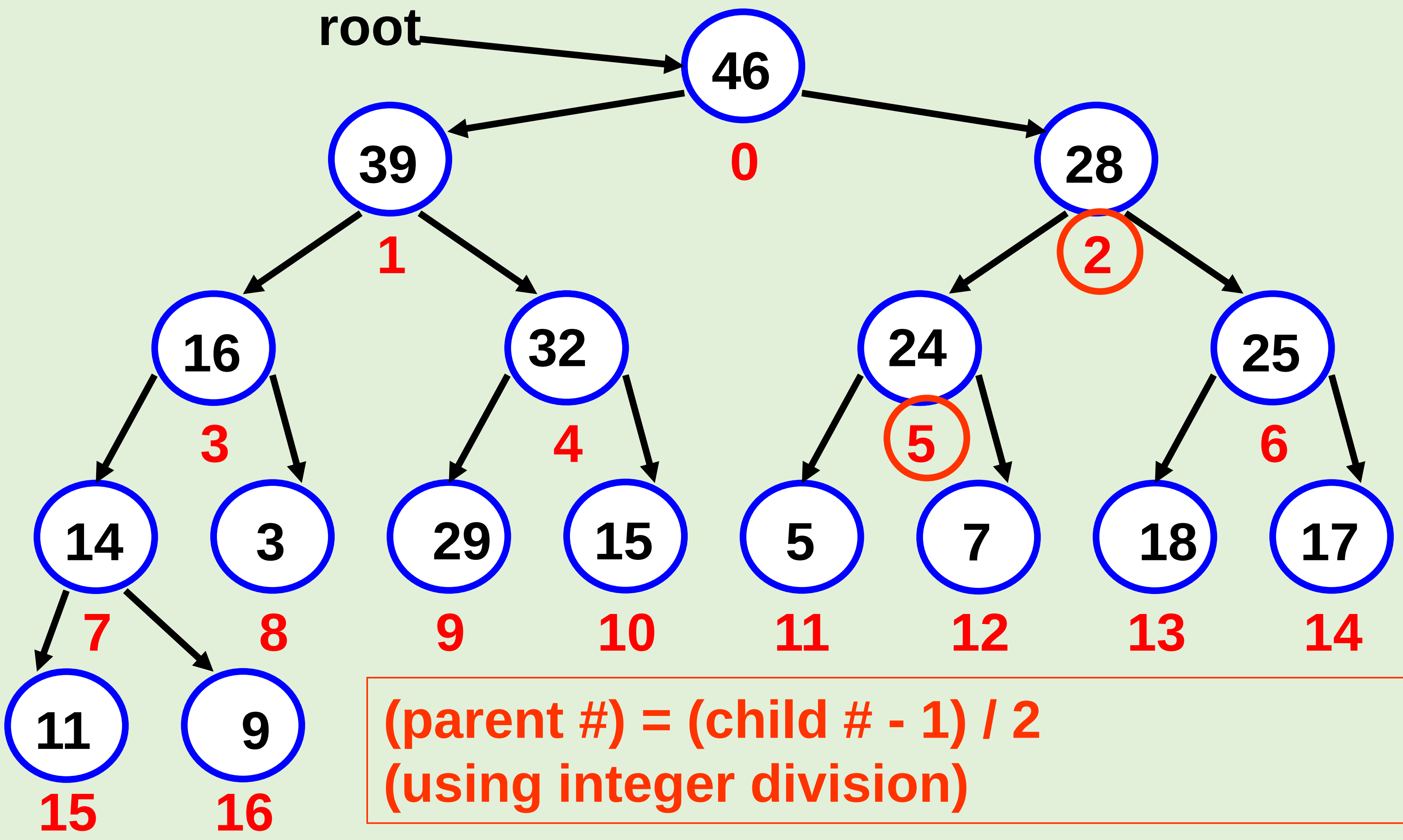
(parent #) = (child # - 1) / 2
(using integer division)

Heap Properties (cont.)



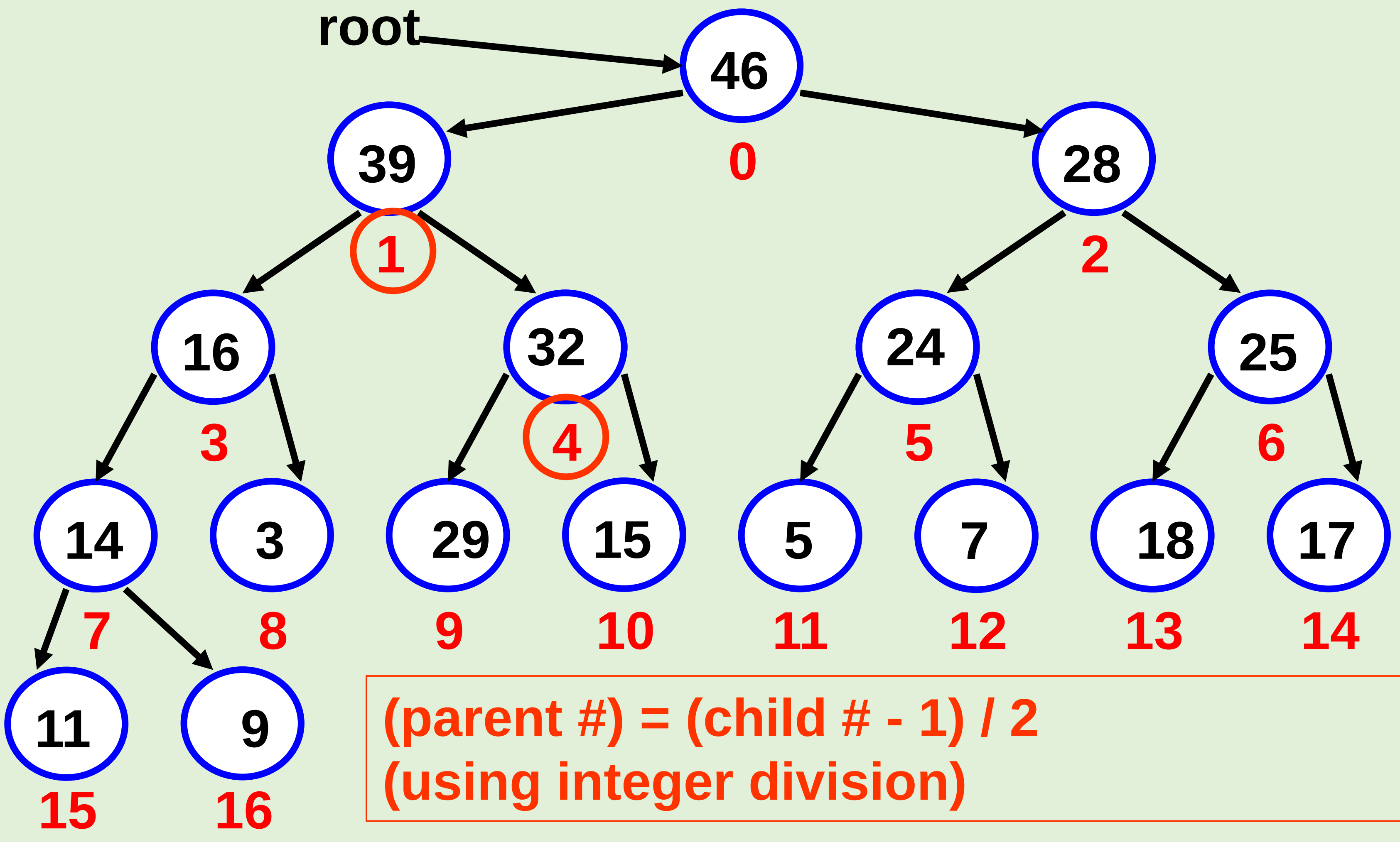
(parent #) = (child # - 1) / 2
(using integer division)

Heap Properties (cont.)



(parent #) = (child # - 1) / 2
(using integer division)

Heap Properties (cont.)



(parent #) = (child # - 1) / 2
(using integer division)

Array Implementation

- These remarkable properties of the heap allow us to place the elements into an array
- The red numbers on the previous slide correspond to the array indexes

Array Implementation (cont.)

46	39	28	16	32	24	25	14	3	29	15	5	7	18	17	11	9
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

So now, this is our heap. It has no linked nodes, so it is much easier to work with.

Let's dequeue an element.

The highest value is stored in the root node (index 0).

Array Implementation (cont.)

46	39	28	16	32	24	25	14	3	29	15	5	7	18	17	11	9
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

remElement: 46

Array Implementation (cont.)

46	39	28	16	32	24	25	14	3	29	15	5	7	18	17	11	9
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

remElement: 46

Now we need to move the object at the last node to the root node

We need to keep track of the object in the last position of the heap using a heapsize variable

Array Implementation (cont.)

46	39	28	16	32	24	25	14	3	29	15	5	7	18	17	11	9
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

remElement: 46

heapsize: 17

Now we need to move the object at the last node to the root node

We need to keep track of the object in the last position of the heap using a heapsize variable

Array Implementation (cont.)

46	39	28	16	32	24	25	14	3	29	15	5	7	18	17	11	9
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

remElement: 46

heapsize: 17

Now we can access the object in the last node using elements[heapsize – 1] and assign it to elements[0]

Array Implementation (cont.)

9	39	28	16	32	24	25	14	3	29	15	5	7	18	17	11	9
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

remElement: 46

heapsize: 17

Now we can access the object in the last node using elements[heapsize – 1] and assign it to elements[0]

Array Implementation (cont.)

9	39	28	16	32	24	25	14	3	29	15	5	7	18	17	11	9
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

remElement: 46

heapsize: 17

Next, decrement the heap size

Array Implementation (cont.)

9	39	28	16	32	24	25	14	3	29	15	5	7	18	17	11	9
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

remElement: 46

heapsize: 16

Next, decrement the heap size

Array Implementation (cont.)

9	39	28	16	32	24	25	14	3	29	15	5	7	18	17	11	9
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

remElement: 46

heapsize: 16

The value at index 16 can't be used anymore; it will be overwritten on the next enqueue

Array Implementation (cont.)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
9	39	28	16	32	24	25	14	3	29	15	5	7	18	17	11	9

remElement: 46

heapsize: 16

Now, we need to find the greatest child of node 0 and compare it to 9.

But how do we get the greatest child?

Array Implementation (cont.)

9	39	28	16	32	24	25	14	3	29	15	5	7	18	17	11	9
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

remElement: 46

heapsize: 16

By using the formulas we noted earlier (this is why an array can be used)…

Array Implementation (cont.)

9	39	28	16	32	24	25	14	3	29	15	5	7	18	17	11	9
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

remElement: 46

heapsize: 16

(left child #) =
2*(parent #) + 1 =
2 * 0 + 1 =
1

(right child #) =
2*(parent #) + 2 =
2 * 0 + 2 =
2

Array Implementation (cont.)

9	39	28	16	32	24	25	14	3	29	15	5	7	18	17	11	9
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

remElement: 46

heapsize: 16

(left child #) =
2*(parent #) + 1 =
2 * 0 + 1 =
1

(right child #) =
2*(parent #) + 2 =
2 * 0 + 2 =
2

Array Implementation (cont.)

Greatest Child

9	39	28	16	32	24	25	14	3	29	15	5	7	18	17	11	9
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

remElement: 46

heapsize: 16

(left child #) =
2*(parent #) + 1 =
2 * 0 + 1 =
1

(right child #) =
2*(parent #) + 2 =
2 * 0 + 2 =
2

Array Implementation (cont.)

Greatest Child **39 > 9, so swap**

9	39	28	16	32	24	25	14	3	29	15	5	7	18	17	11	9
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

remElement: 46

heapsize: 16

(left child #) =
2*(parent #) + 1 =
2 * 0 + 1 =
1

(right child #) =
2*(parent #) + 2 =
2 * 0 + 2 =
2

Array Implementation (cont.)

9	39	28	16	32	24	25	14	3	29	15	5	7	18	17	11	9
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

remElement: 46

heapsize: 16

Array Implementation (cont.)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
9	39	28	16	32	24	25	14	3	29	15	5	7	18	17	11	9

remElement: 46

heapsize: 16

Array Implementation (cont.)

39	9	28	16	32	24	25	14	3	29	15	5	7	18	17	11	9
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

remElement: 46

heapsize: 16

Array Implementation (cont.)

39	9	28	16	32	24	25	14	3	29	15	5	7	18	17	11	9
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

remElement: 46

heapsize: 16

Array Implementation (cont.)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
39	9	28	16	32	24	25	14	3	29	15	5	7	18	17	11	9

remElement: 46

heapsize: 16

If the greatest child of 9 is greater than 9, then swap

Array Implementation (cont.)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
39	9	28	16	32	24	25	14	3	29	15	5	7	18	17	11	9

remElement: 46

heapsize: 16

(left child #) =
2*(parent #) + 1 =
2 * 1 + 1 =
3

(right child #) =
2*(parent #) + 2 =
2 * 1 + 2 =
4

Array Implementation (cont.)

39	9	28	16	32	24	25	14	3	29	15	5	7	18	17	11	9
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

remElement: 46

heapsize: 16

(left child #) =
2*(parent #) + 1 =
2 * 1 + 1 =
3

(right child #) =
2*(parent #) + 2 =
2 * 1 + 2 =
4

Array Implementation (cont.)

Greatest Child

39	9	28	16	32	24	25	14	3	29	15	5	7	18	17	11	9
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

remElement: 46

heapsize: 16

(left child #) =
2*(parent #) + 1 =
2 * 1 + 1 =
3

(right child #) =
2*(parent #) + 2 =
2 * 1 + 2 =
4

Array Implementation (cont.)

Greatest Child 32 > 9, so swap

39	9	28	16	32	24	25	14	3	29	15	5	7	18	17	11	9
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

remElement: 46

heapsize: 16

(left child #) =
2*(parent #) + 1 =
2 * 1 + 1 =
3

(right child #) =
2*(parent #) + 2 =
2 * 1 + 2 =
4

Array Implementation (cont.)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
39	9	28	16	32	24	25	14	3	29	15	5	7	18	17	11	9

remElement: 46

heapsize: 16

Array Implementation (cont.)

9

32

39		28	16		24	25	14	3	29	15	5	7	18	17	11	9
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

remElement: 46

heapsize: 16

Array Implementation (cont.)

9
32

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
39		28	16		24	25	14	3	29	15	5	7	18	17	11	9

remElement: 46

heapsize: 16

Array Implementation (cont.)

9

32

39		28	16		24	25	14	3	29	15	5	7	18	17	11	9
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

remElement: 46

heapsize: 16

Array Implementation (cont.)

	32		9													
39		28	16		24	25	14	3	29	15	5	7	18	17	11	9
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

remElement: 46

heapsize: 16

Array Implementation (cont.)

39	32	28	16	9	24	25	14	3	29	15	5	7	18	17	11	9
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

remElement: 46

heapsize: 16

Array Implementation (cont.)

39	32	28	16	9	24	25	14	3	29	15	5	7	18	17	11	9
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

remElement: 46

heapsize: 16

If the greatest child of 9 is greater than 9, then swap

Array Implementation (cont.)

39	32	28	16	9	24	25	14	3	29	15	5	7	18	17	11	9
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

remElement: 46

heapsize: 16

(left child #) =
2*(parent #) + 1 =
2 * 4 + 1 =
9

(right child #) =
2*(parent #) + 2 =
2 * 4 + 2 =
10

Array Implementation (cont.)

39	32	28	16	9	24	25	14	3	29	15	5	7	18	17	11	9
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

remElement: 46

heapsize: 16

(left child #) =
2*(parent #) + 1 =
2 * 4 + 1 =
9

(right child #) =
2*(parent #) + 2 =
2 * 4 + 2 =
10

Array Implementation (cont.)

Greatest Child

39	32	28	16	9	24	25	14	3	29	15	5	7	18	17	11	9
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

remElement: 46

heapsize: 16

(left child #) =
2*(parent #) + 1 =
2 * 4 + 1 =
9

(right child #) =
2*(parent #) + 2 =
2 * 4 + 2 =
10

Array Implementation (cont.)

Greatest Child **29 > 9, so swap**

39	32	28	16	9	24	25	14	3	29	15	5	7	18	17	11	9
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

remElement: 46

heapsize: 16

(left child #) =
2*(parent #) + 1 =
2 * 4 + 1 =
9

(right child #) =
2*(parent #) + 2 =
2 * 4 + 2 =
10

Array Implementation (cont.)

39	32	28	16	9	24	25	14	3	29	15	5	7	18	17	11	9
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

remElement: 46

heapsize: 16

Array Implementation (cont.)

				9				29								
39	32	28	16		24	25	14	3		15	5	7	18	17	11	9
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

remElement: 46

heapsize: 16

Array Implementation (cont.)

9

29

39	32	28	16		24	25	14	3		15	5	7	18	17	11	9
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

remElement: 46

heapsize: 16

Array Implementation (cont.)

9
29

39	32	28	16		24	25	14	3		15	5	7	18	17	11	9
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

remElement: 46

heapsize: 16

Array Implementation (cont.)

9

29

39	32	28	16		24	25	14	3		15	5	7	18	17	11	9
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

remElement: 46

heapsize: 16

Array Implementation (cont.)

				29					9							
39	32	28	16		24	25	14	3		15	5	7	18	17	11	9
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

remElement: 46

heapsize: 16

Array Implementation (cont.)

39	32	28	16	29	24	25	14	3	9	15	5	7	18	17	11	9
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

remElement: 46

heapsize: 16

Array Implementation (cont.)

39	32	28	16	29	24	25	14	3	9	15	5	7	18	17	11	9
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

remElement: 46

heapsize: 16

If the greatest child of 9 is greater than 9, then swap

Array Implementation (cont.)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
39	32	28	16	29	24	25	14	3	9	15	5	7	18	17	11	9

remElement: 46

heapsize: 16

(left child #) =
2*(parent #) + 1 =
2 * 9 + 1 =
19

Array Implementation (cont.)

39	32	28	16	29	24	25	14	3	9	15	5	7	18	17	11	9
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

remElement: 46

heapsize: 16

19 > heapsize

(left child #) =
2*(parent #) + 1 =
2 * 9 + 1 =
19

Array Implementation (cont.)

39	32	28	16	29	24	25	14	3	9	15	5	7	18	17	11	9
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

so 9 must be a leaf node

(we can stop)

remElement: 46

heapsize: 16

(left child #) =
2*(parent #) + 1 =
2 * 9 + 1 =
19

Array Implementation (cont.)

39	32	28	16	29	24	25	14	3	9	15	5	7	18	17	11	9
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

heapsize: 16

An enqueue is done by placing the new element at elements[heapsize], then swapping upwards

Array Implementation (cont.)

39	32	28	16	29	24	25	14	3	9	15	5	7	18	17	11	9
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

heapsize: 16

When enqueuing, the parent is always found by using the parent formula:

(parent #) = (child # - 1) / 2