

Disjoint Sets Analysis

CS 225 - Fall 2026

Mattox Beckman

Use the up-tree data structure to model disjoint sets

Examine the time complexity of find and union

Use rank to prove the $\mathcal{O}(\log^* n)$ bound

Disjoint Sets

- `makeSet(vector<T> items)`
- `find(T key)`
- `union(T key1, T key2)`

Disjoint Sets

- `makeSet(vector<T> items)`
- `find(T key)`
- `union(T key1, T key2)`

Key Ideas

- Each item exists in exactly one set
- Each set has a different representative
- Items in the same set have the same representative

Use an integer vector `ds` to record the representatives

- `ds[i] == -1` means i is a singleton
- `ds[i] == n,  $n \geq 0$` means i 's parent is n
- `ds[i] == n,  $n < 0$` means i is root
- ... and that the set contains $|n|$ elements

Find(n)

- Keep following parents until you reach a root.

```
1 int find(vector<int> &ds, int n) {  
2     if (ds[n] < 0)  
3         return n;  
4     else  
5         return find(ds, ds[n]);  
6 }
```

Find(n)

- Keep following parents until you reach a root.

```
1 int find(vector<int> &ds, int n) {  
2     if (ds[n] < 0)  
3         return n;  
4     else  
5         return find(ds, ds[n]);  
6 }
```

$n < 0$ means we found the root

Find(n)

- Keep following parents until you reach a root.

```
1 int find(vector<int> &ds, int n) {  
2     if (ds[n] < 0)  
3         return n;  
4     else  
5         return find(ds, ds[n]);  
6 }
```

$n < 0$ means we found the root
return the index of the representative

Find(n)

- Keep following parents until you reach a root.

```
1 int find(vector<int> &ds, int n) {  
2     if (ds[n] < 0)  
3         return n;  
4     else  
5         return find(ds, ds[n]);  
6 }
```

n < 0 means we found the root
return the index of the representative

recurse...

Rank

rank(x) = the height of x *if we had never compressed*

rank(x) = the height of x *if we had never compressed*

- Never stored: it only exists for the proof
- **ds** still holds sizes; size decides who links under whom

rank(x) = the height of x if we had never compressed

- Never stored: it only exists for the proof
- **ds** still holds sizes; size decides who links under whom

We can track it as we go: when root a links under root b ,

- $\text{rank}(b) = \max(\text{rank}(b), \text{rank}(a) + 1)$

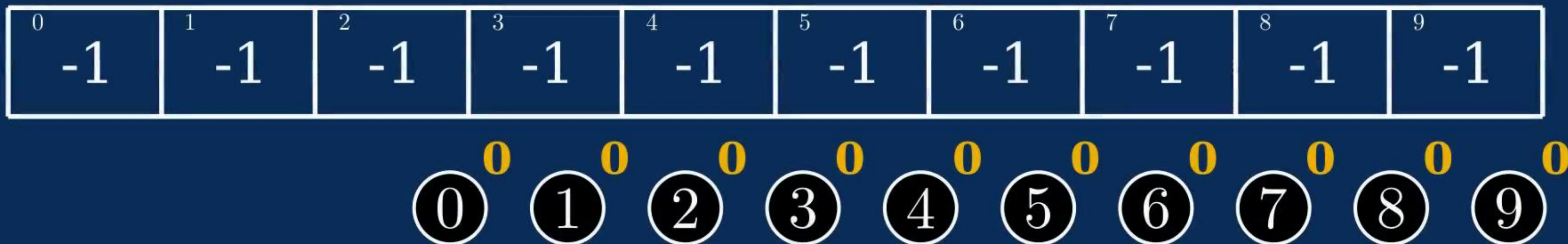
rank(x) = the height of x if we had never compressed

- Never stored: it only exists for the proof
- **ds** still holds sizes; size decides who links under whom

We can track it as we go: when root a links under root b ,

- $\text{rank}(b) = \max(\text{rank}(b), \text{rank}(a) + 1)$

Compression never changes rank



Gold number = rank

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

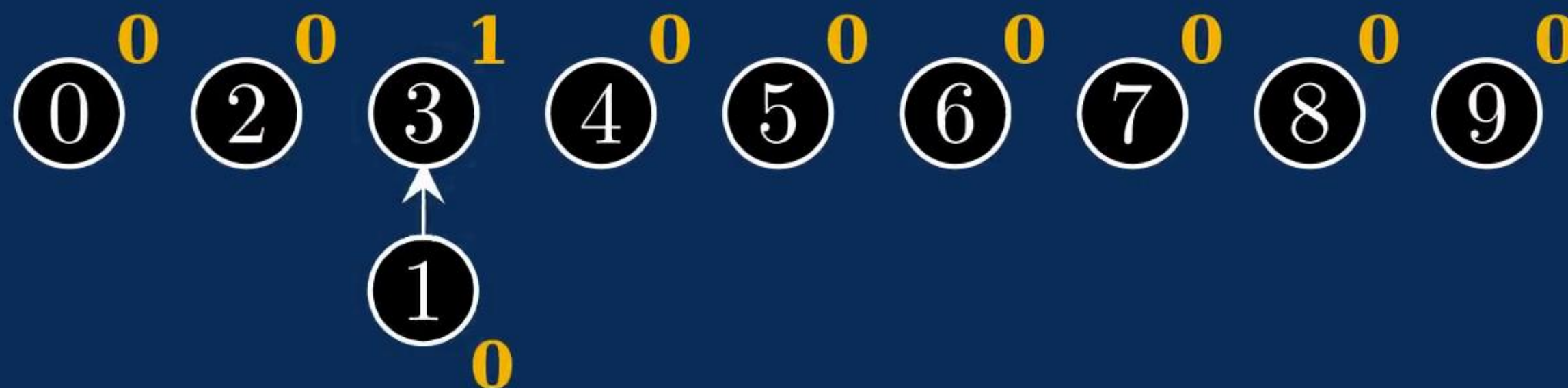
union(1,3)

⁰⁰ ¹⁰ ²⁰ ³⁰ ⁴⁰ ⁵⁰ ⁶⁰ ⁷⁰ ⁸⁰ ⁹⁰
(0) (1) (2) (3) (4) (5) (6) (7) (8) (9)

Gold number = rank

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

union(1,3)



Gold number = rank

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

union(1,3)

union(1,4)



Gold number = rank

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

union(1,3)

union(1,4)



Gold number = rank

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

union(1,3)

union(1,4)

union(5,9)

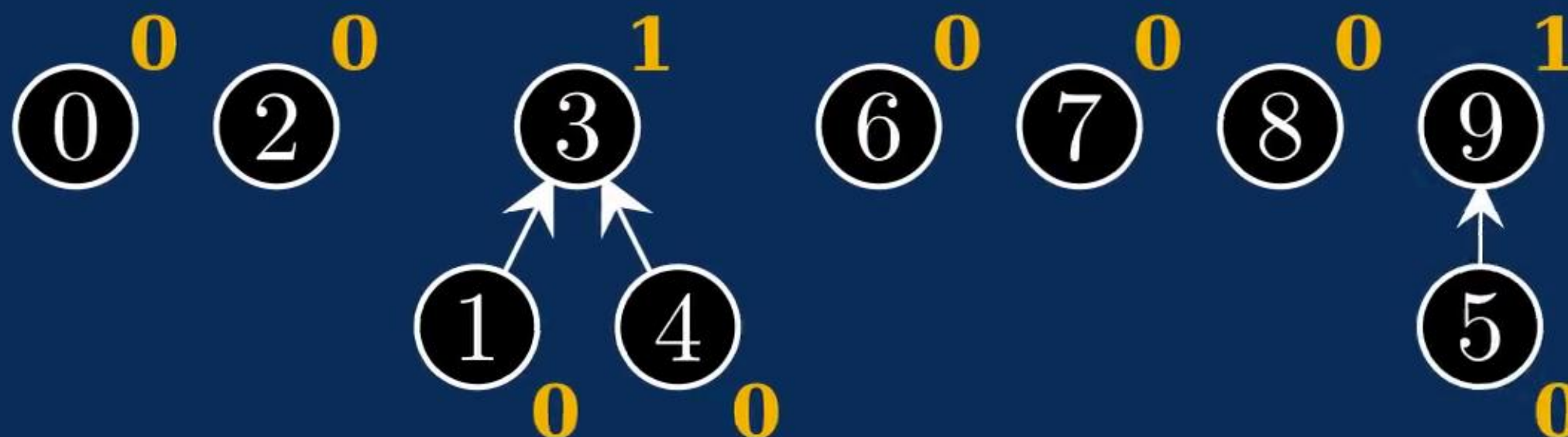


⁰ -1	¹ 3	² -1	³ -3	⁴ 3	⁵ 9	⁶ -1	⁷ -1	⁸ -1	⁹ -2
--------------------	-------------------	--------------------	--------------------	-------------------	-------------------	--------------------	--------------------	--------------------	--------------------

union(1,3)

union(1,4)

union(5,9)



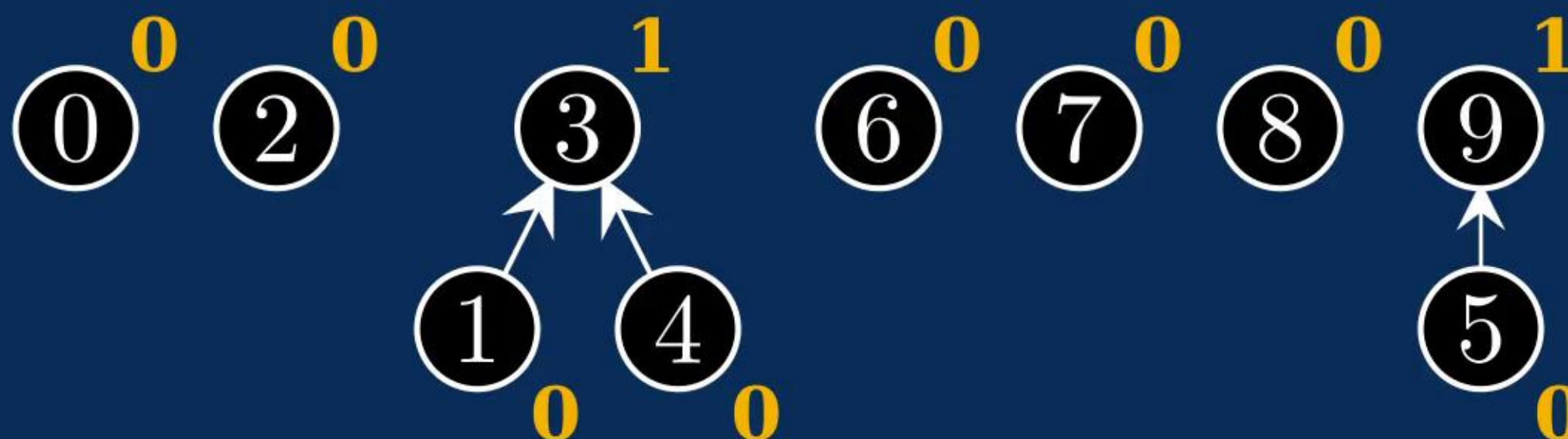
⁰ -1	¹ 3	² -1	³ -3	⁴ 3	⁵ 9	⁶ -1	⁷ -1	⁸ -1	⁹ -2
--------------------	-------------------	--------------------	--------------------	-------------------	-------------------	--------------------	--------------------	--------------------	--------------------

union(1,3)

union(1,4)

union(5,9)

union(8,6)



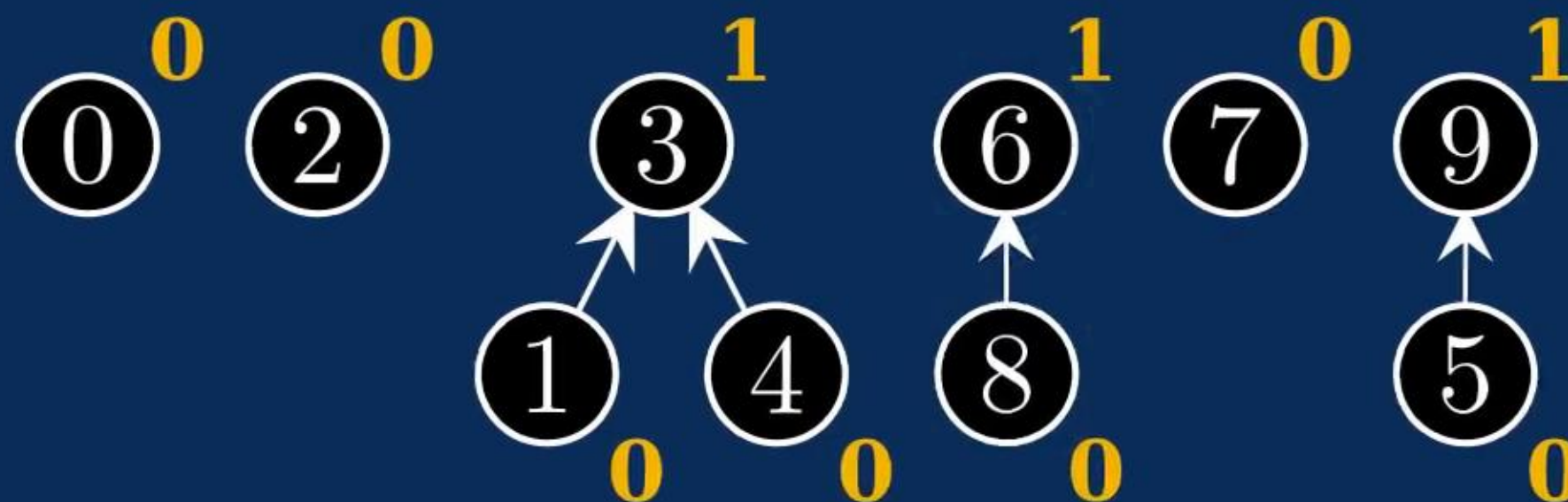
⁰ -1	¹ 3	² -1	³ -3	⁴ 3	⁵ 9	⁶ -2	⁷ -1	⁸ 6	⁹ -2
--------------------	-------------------	--------------------	--------------------	-------------------	-------------------	--------------------	--------------------	-------------------	--------------------

union(1,3)

union(1,4)

union(5,9)

union(8,6)



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

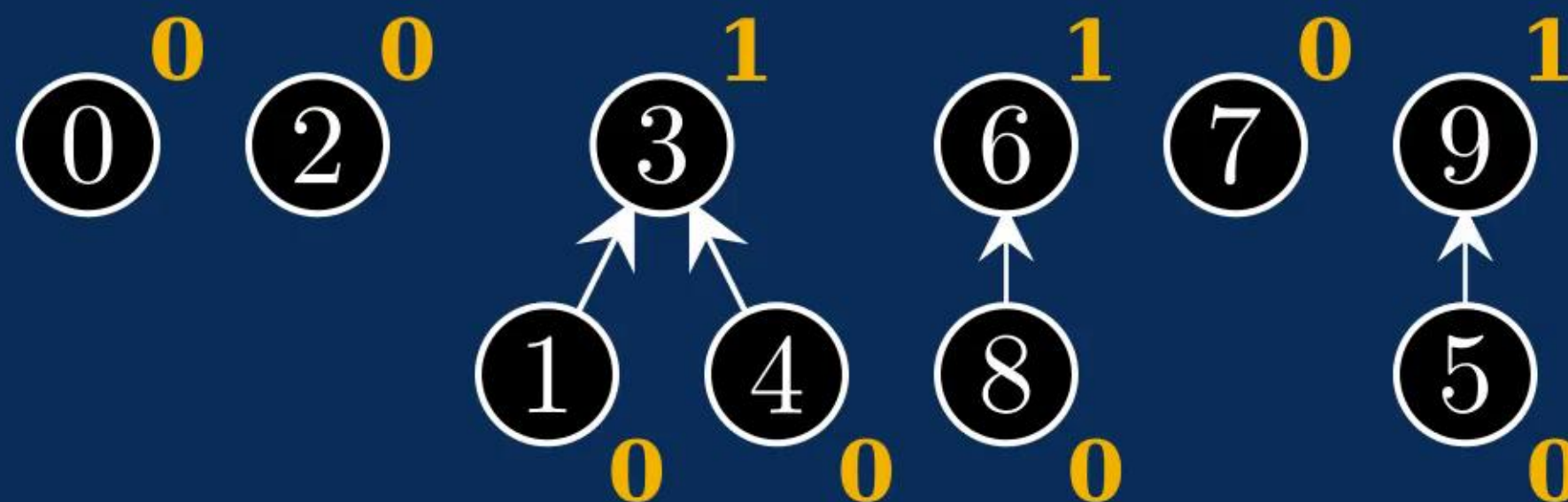
union(1,3)

union(1,4)

union(5,9)

union(8,6)

union(8,5)



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

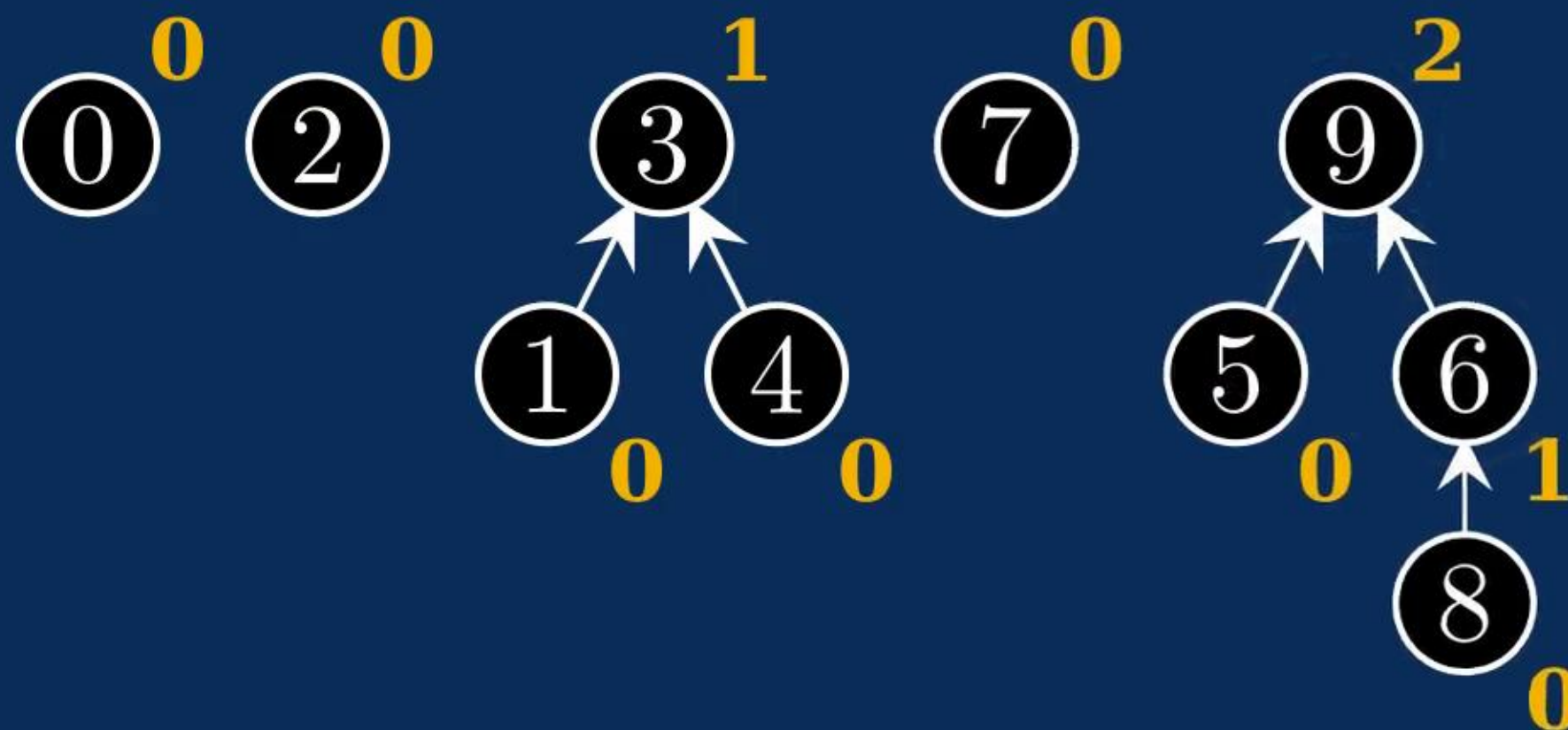
union(1,3)

union(1,4)

union(5,9)

union(8,6)

union(8,5)



Gold number = rank

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

union(1,3)

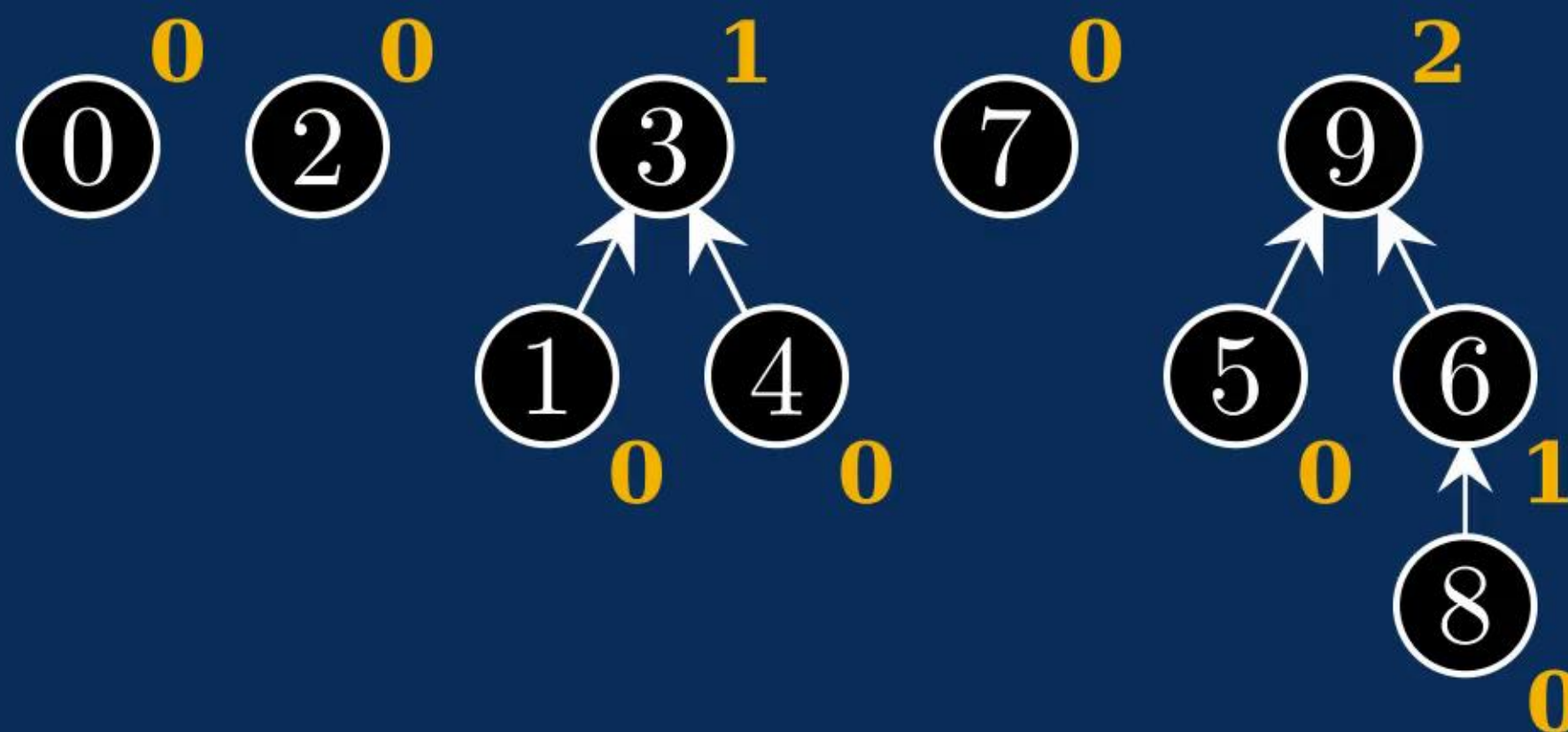
union(1,4)

union(5,9)

union(8,6)

union(8,5)

union(8,1)



Gold number = rank

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

union(1,3)

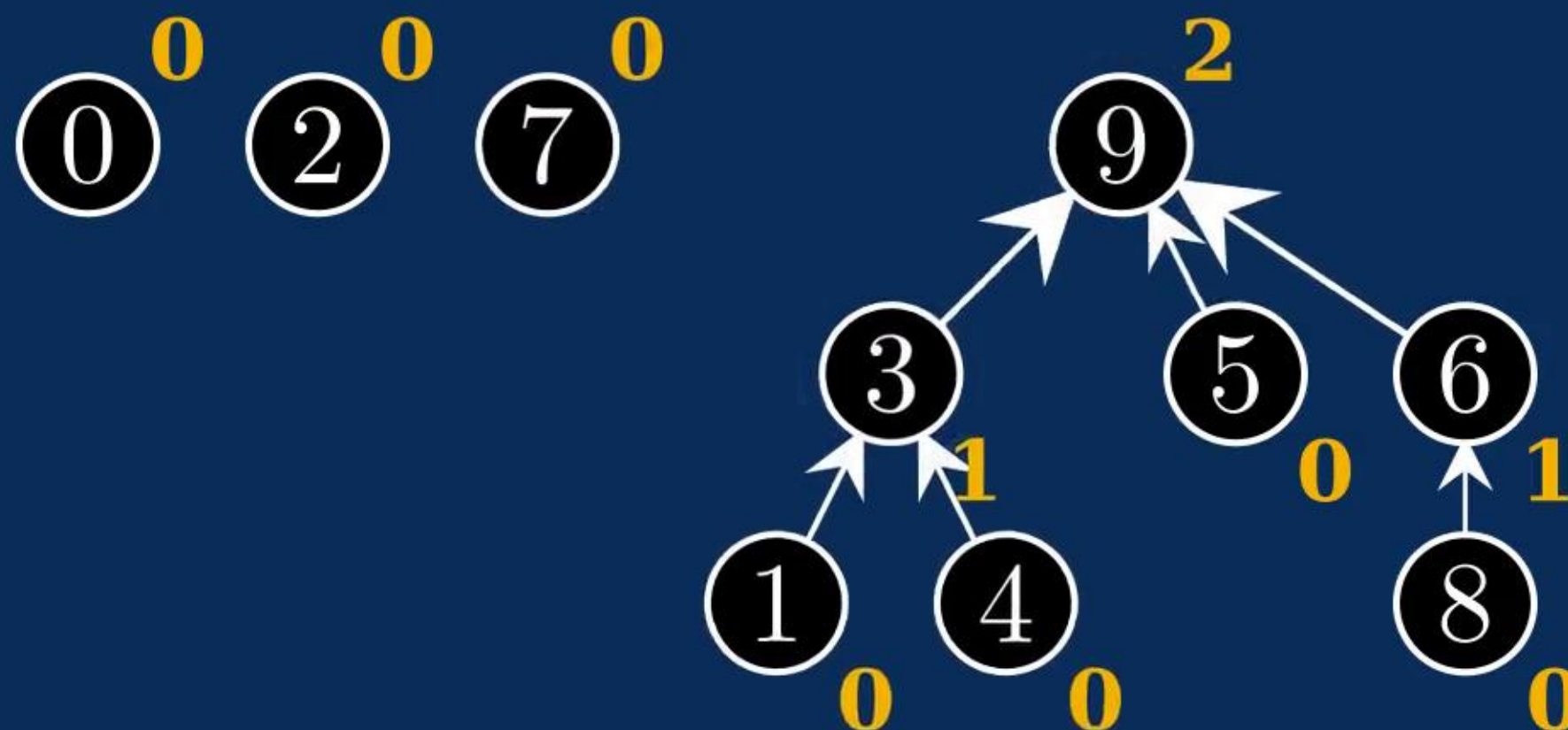
union(1,4)

union(5,9)

union(8,6)

union(8,5)

union(8,1)



Gold number = rank

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

union(1,3)

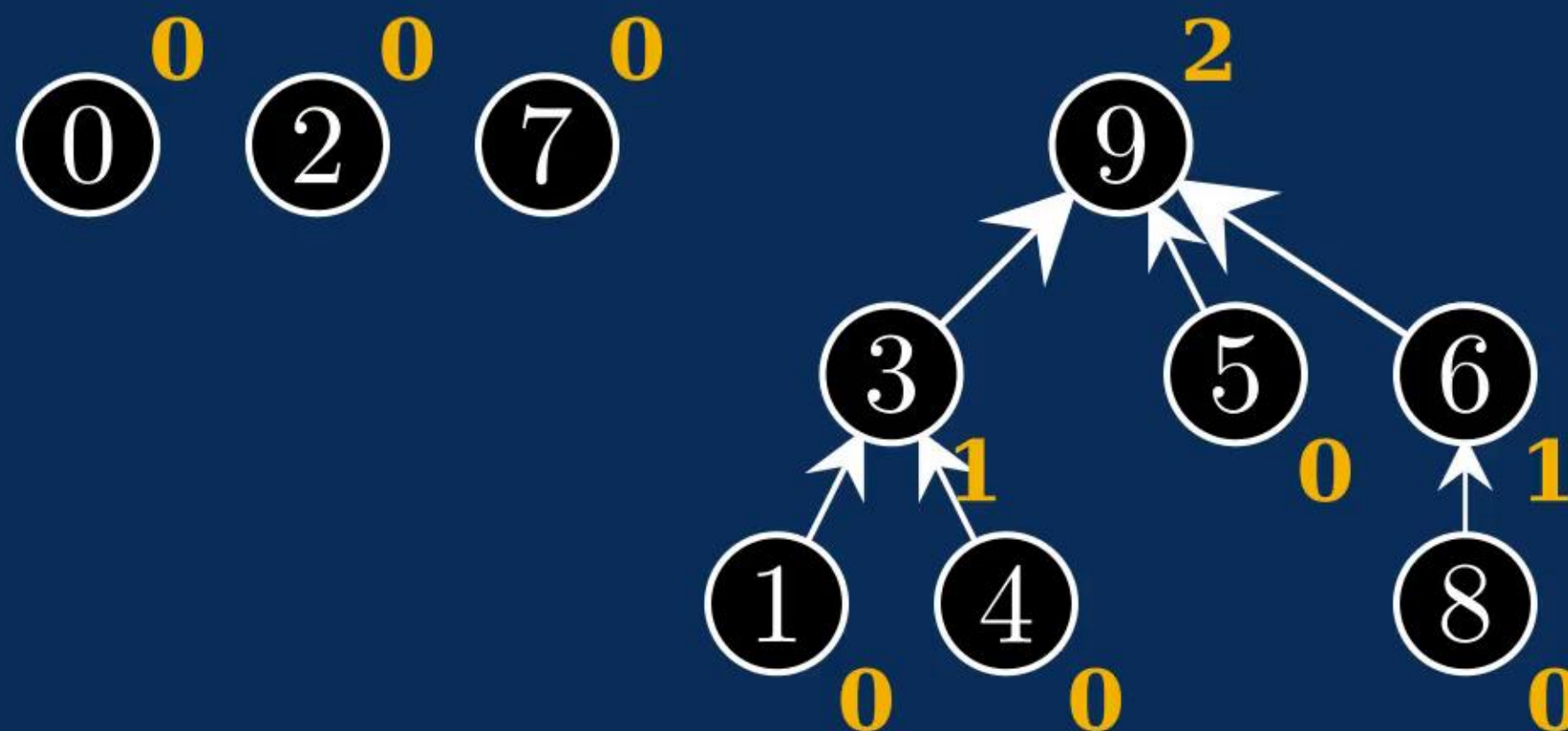
union(1,4)

union(5,9)

union(8,6)

union(8,5)

union(8,1)



No compression: rank = height

Gold number = rank

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

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

union(1,3)

⁰⁰ ⁰⁰ ⁰⁰ ⁰⁰ ⁰⁰ ⁰⁰ ⁰⁰ ⁰⁰ ⁰⁰ ⁰⁰

① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨

Gold number = rank

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

union(1,3)



Gold number = rank

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

union(1,3)

union(1,4)



Gold number = rank

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

union(1,3)

union(1,4)



Gold number = rank

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

union(1,3)

union(1,4)

union(5,9)

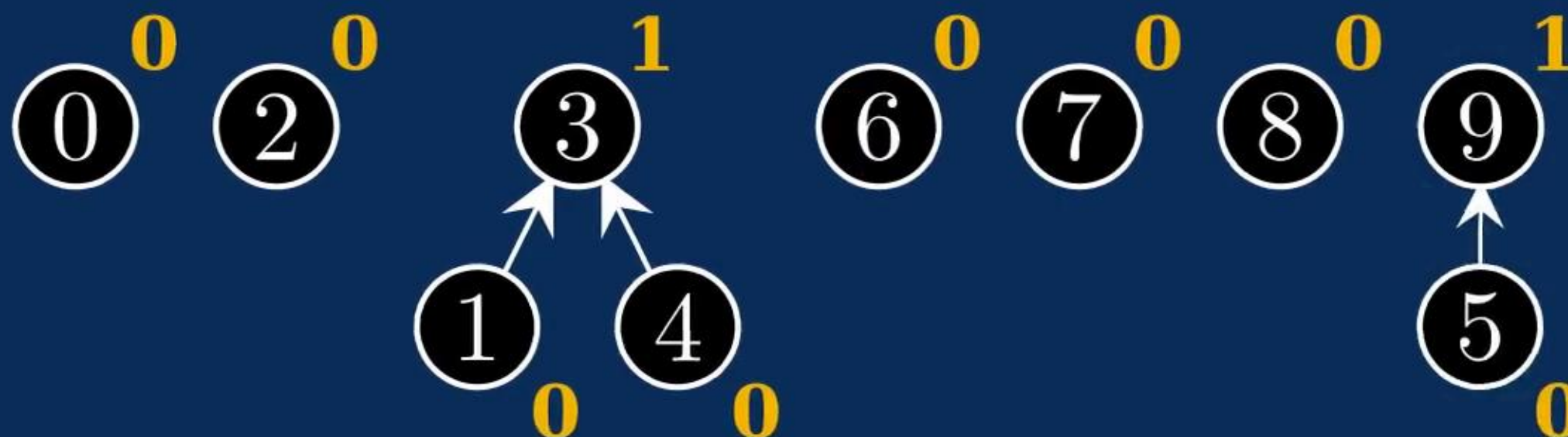


⁰ -1	¹ 3	² -1	³ -3	⁴ 3	⁵ 9	⁶ -1	⁷ -1	⁸ -1	⁹ -2
--------------------	-------------------	--------------------	--------------------	-------------------	-------------------	--------------------	--------------------	--------------------	--------------------

union(1,3)

union(1,4)

union(5,9)



Gold number = rank

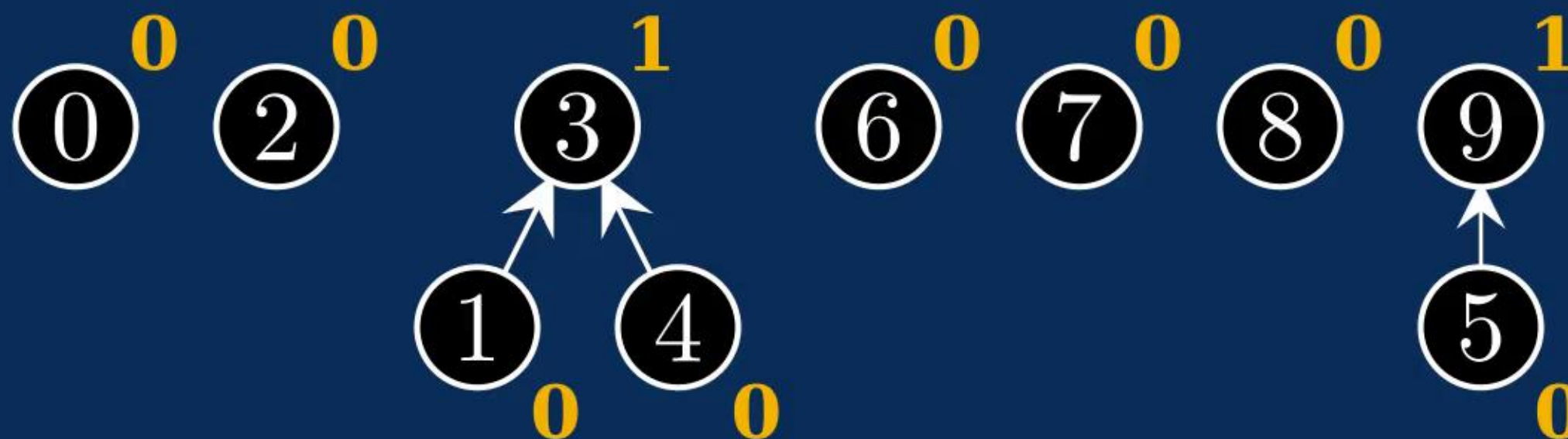
⁰ -1	¹ 3	² -1	³ -3	⁴ 3	⁵ 9	⁶ -1	⁷ -1	⁸ -1	⁹ -2
--------------------	-------------------	--------------------	--------------------	-------------------	-------------------	--------------------	--------------------	--------------------	--------------------

union(1,3)

union(1,4)

union(5,9)

union(8,6)



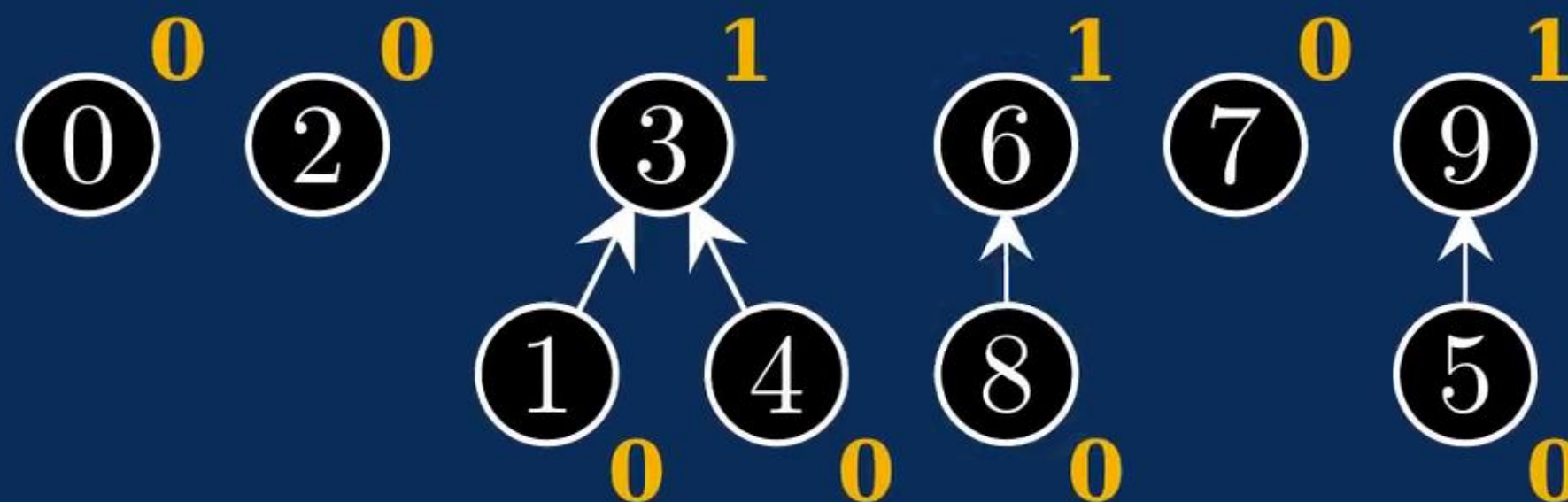
⁰ -1	¹ 3	² -1	³ -3	⁴ 3	⁵ 9	⁶ -2	⁷ -1	⁸ 6	⁹ -2
--------------------	-------------------	--------------------	--------------------	-------------------	-------------------	--------------------	--------------------	-------------------	--------------------

union(1,3)

union(1,4)

union(5,9)

union(8,6)



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

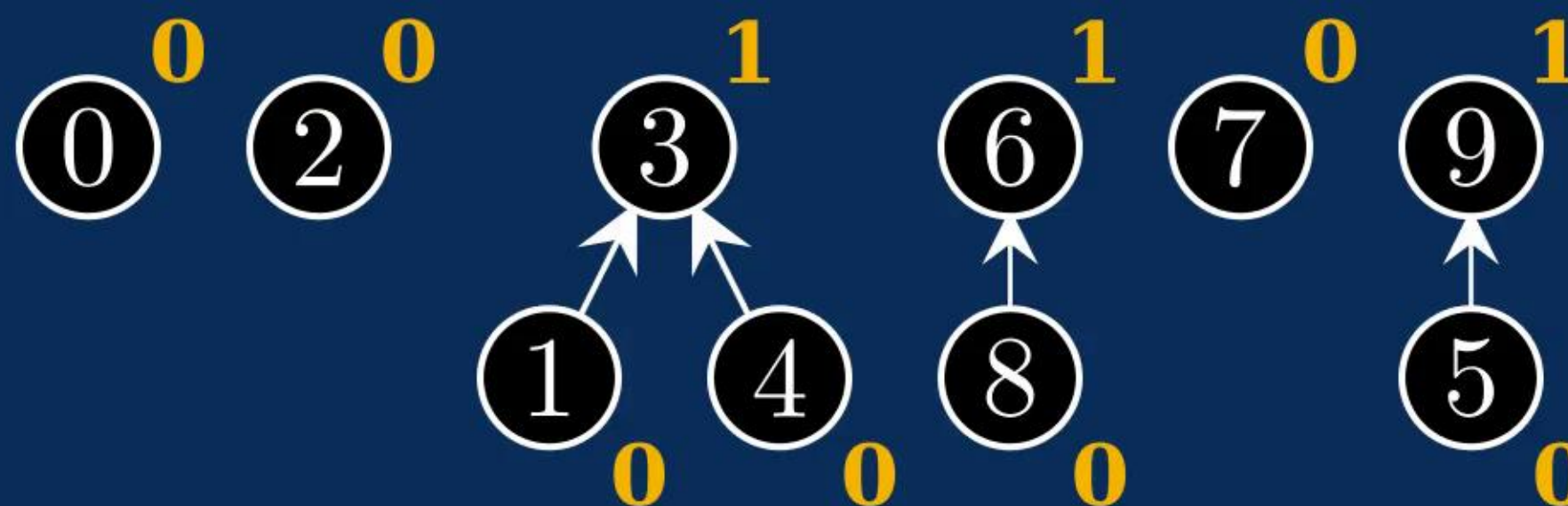
union(1,3)

union(1,4)

union(5,9)

union(8,6)

union(8,5)



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

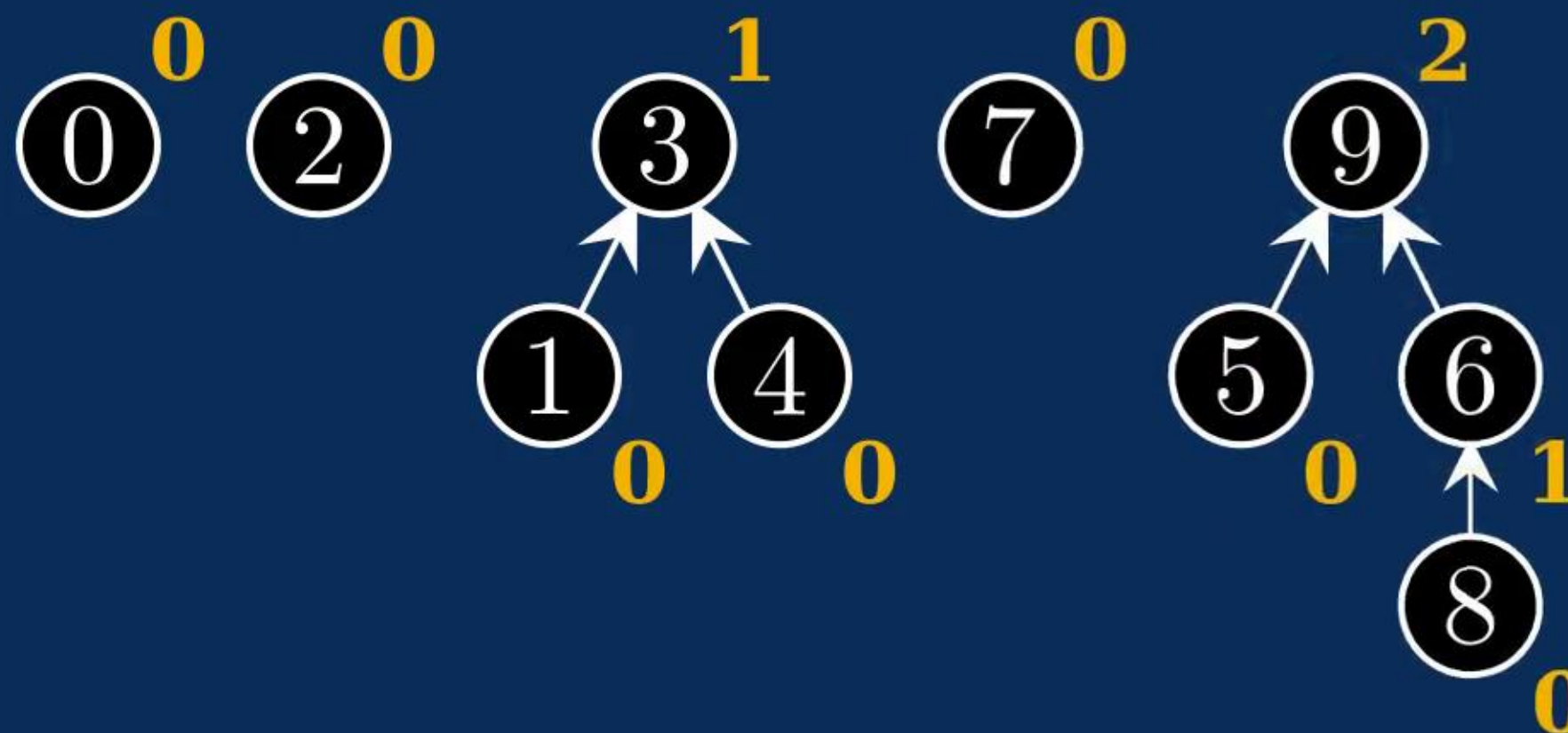
union(1,3)

union(1,4)

union(5,9)

union(8,6)

union(8,5)



Gold number = rank

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

union(1,3)

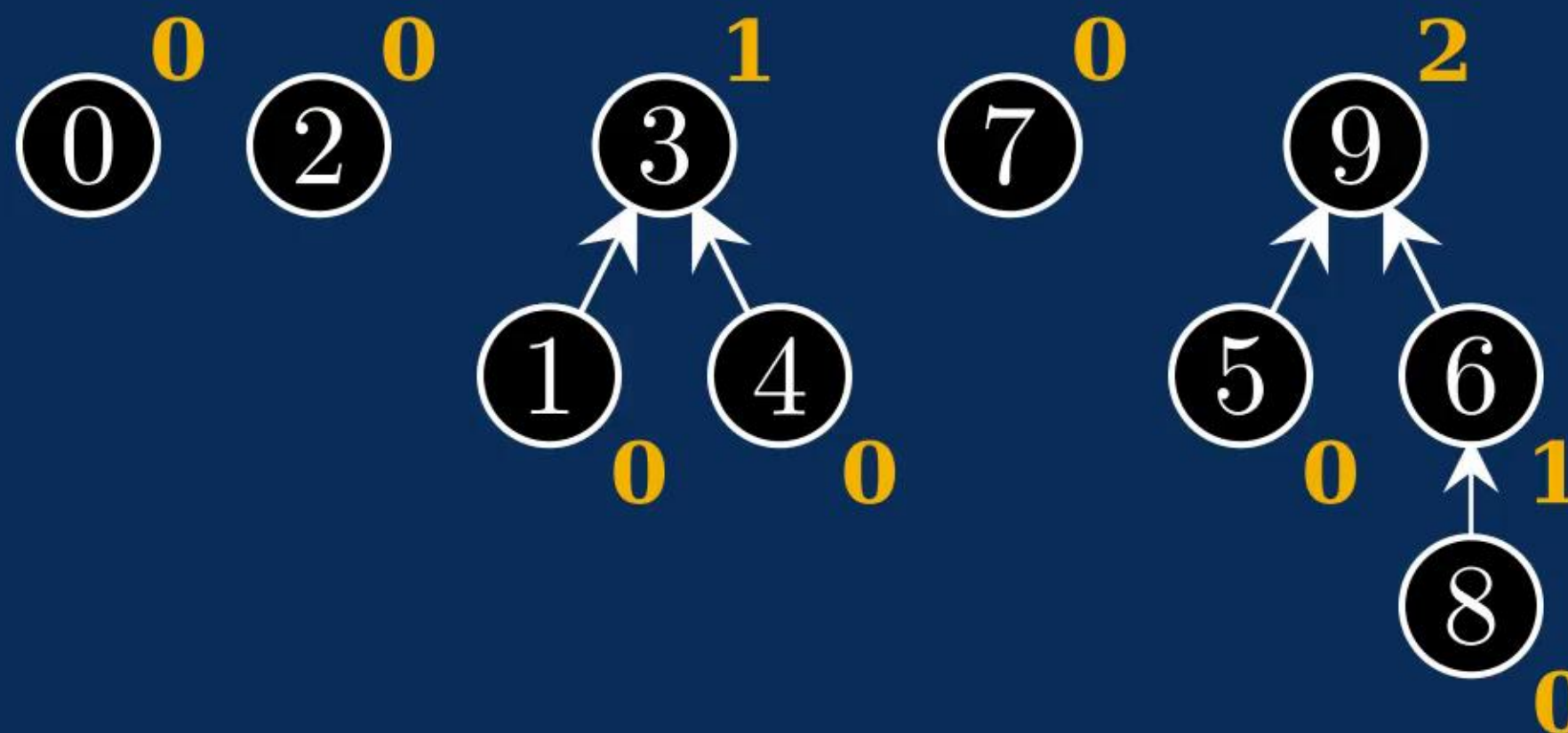
union(1,4)

union(5,9)

union(8,6)

union(8,5)

union(8,1)



Gold number = rank

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

union(1,3)

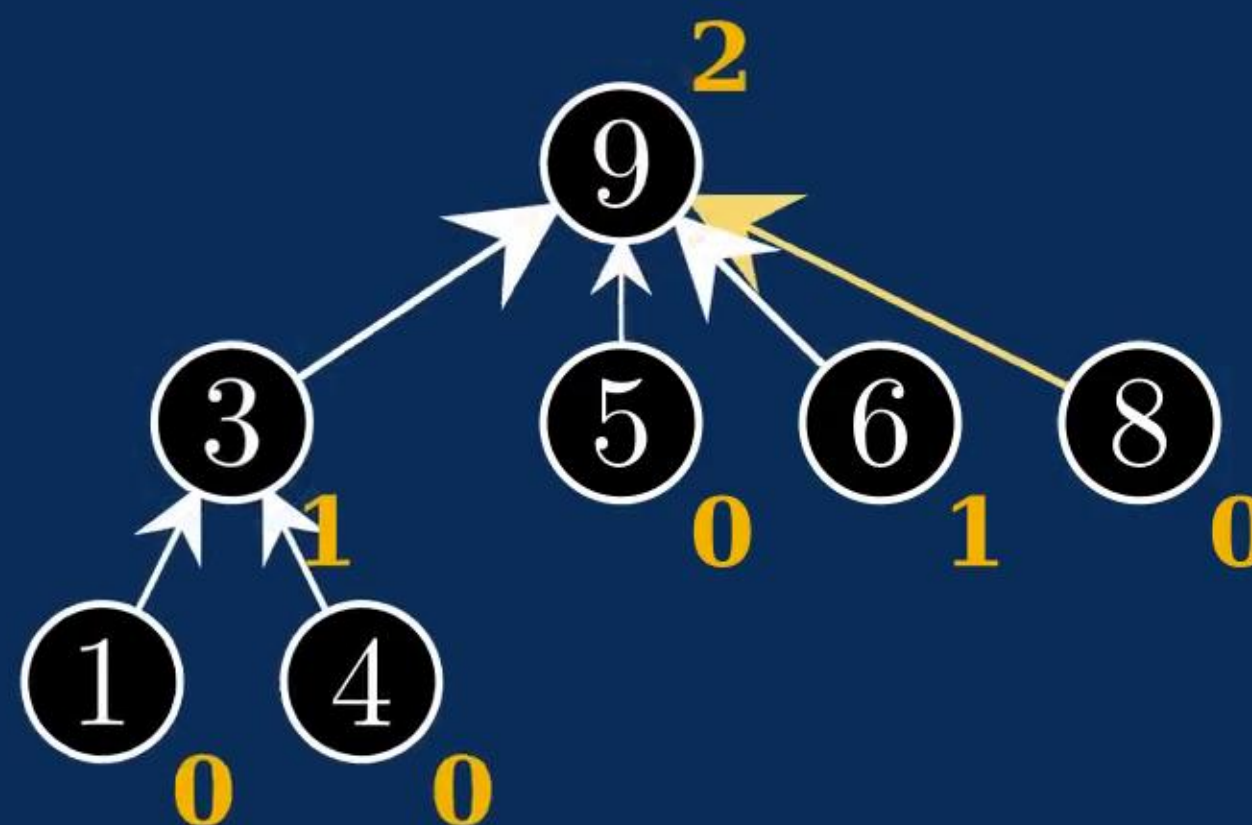
union(1,4)

union(5,9)

union(8,6)

union(8,5)

union(8,1)



Gold number = rank

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

union(1,3)

union(1,4)

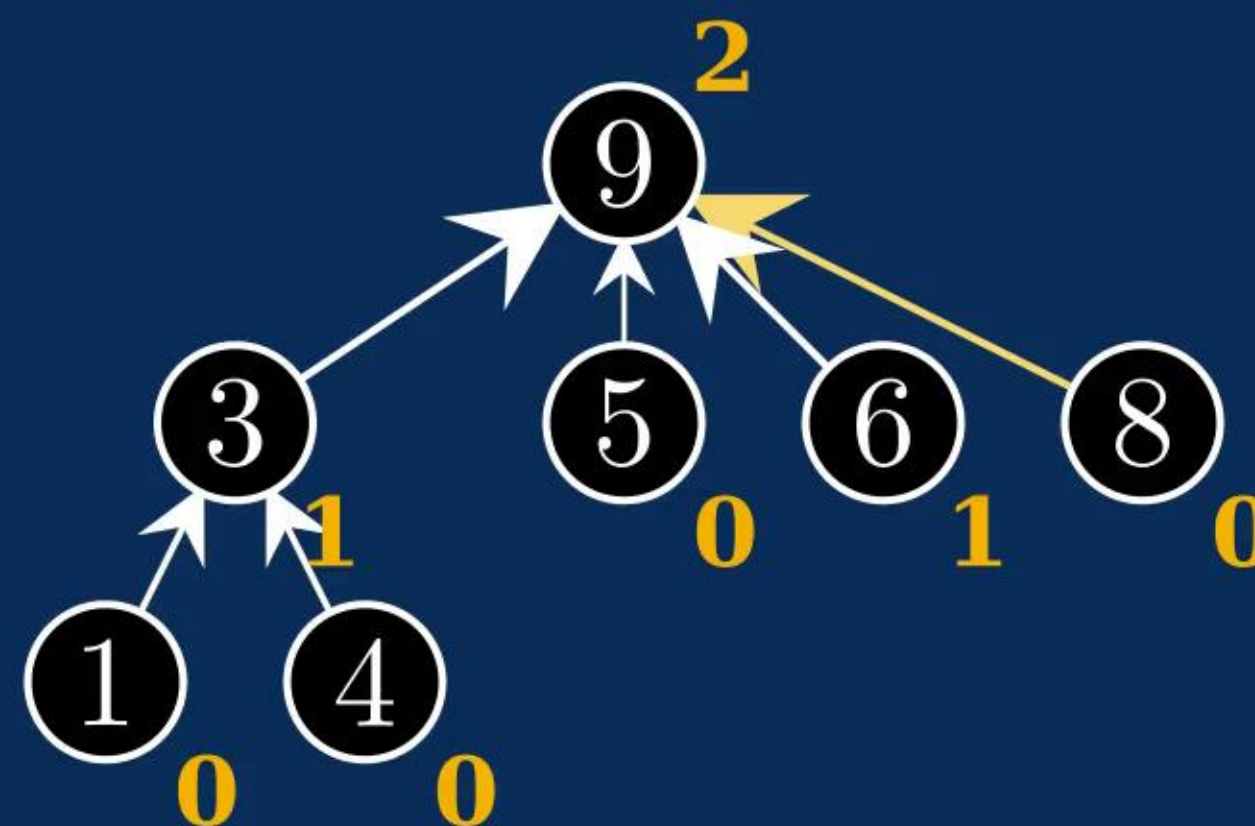
union(5,9)

union(8,6)

union(8,5)

union(8,1)

find(1)



Gold number = rank

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

union(1,3)

union(1,4)

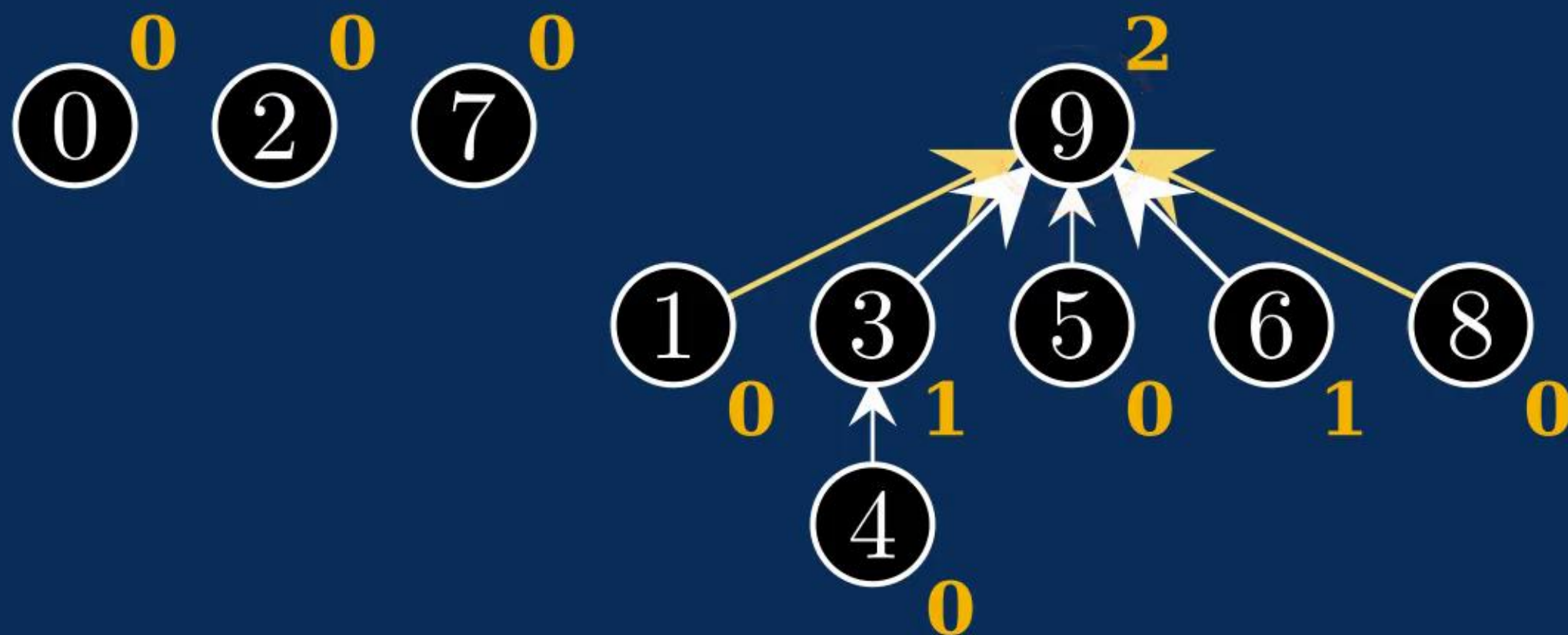
union(5,9)

union(8,6)

union(8,5)

union(8,1)

find(1)



Gold number = rank

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

union(1,3)

union(1,4)

union(5,9)

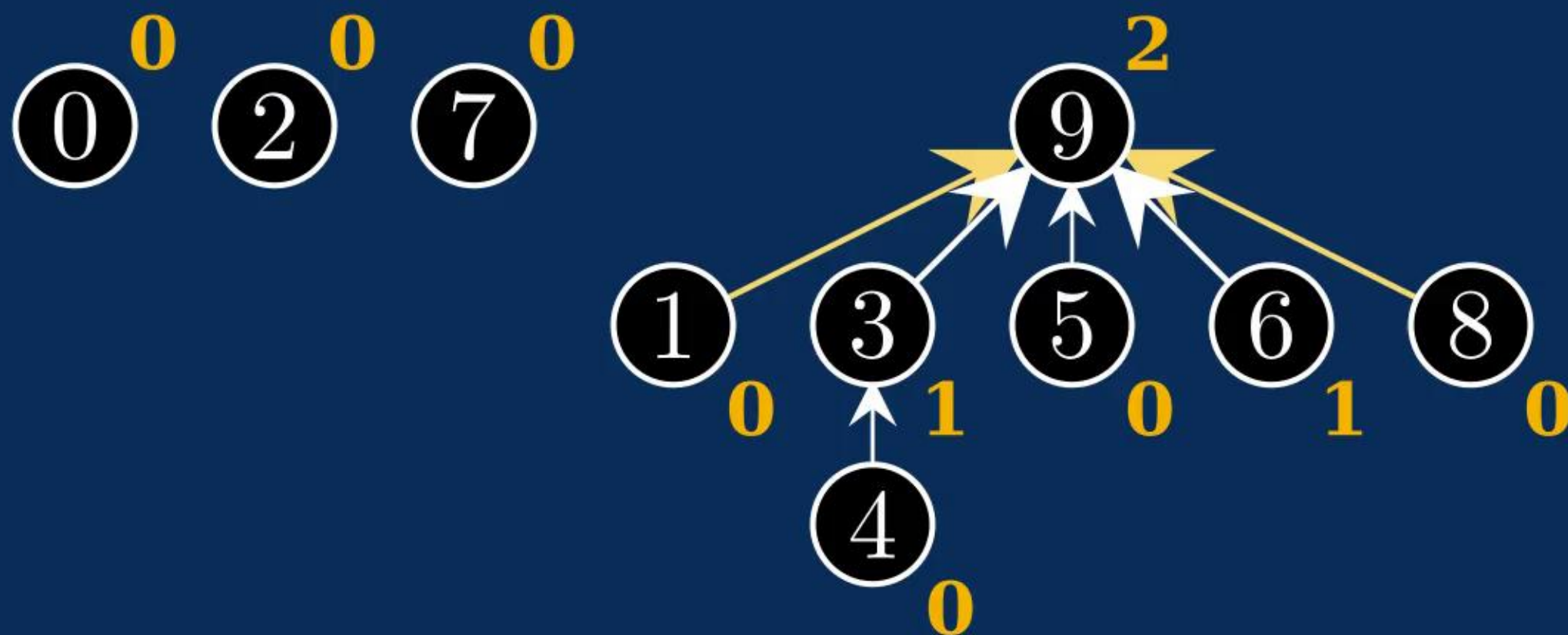
union(8,6)

union(8,5)

union(8,1)

find(1)

find(4)



Gold number = rank

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

union(1,3)

union(1,4)

union(5,9)

union(8,6)

union(8,5)

union(8,1)

find(1)

find(4)



Gold number = rank

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

union(1,3)

union(1,4)

union(5,9)

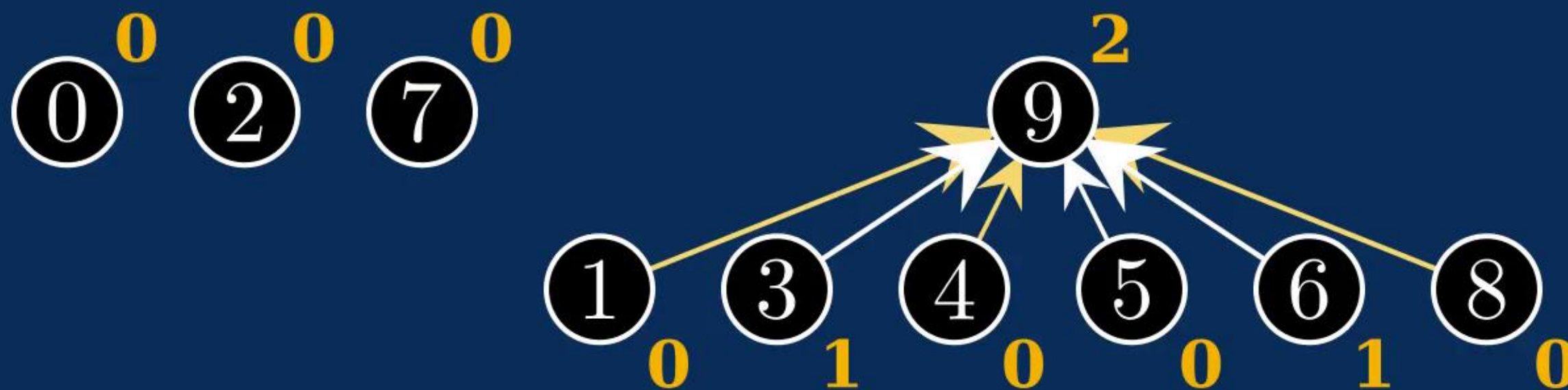
union(8,6)

union(8,5)

union(8,1)

find(1)

find(4)



3 and 6: height 0, but rank 1

Gold number = rank

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

union(1,3)

union(1,4)

union(5,9)

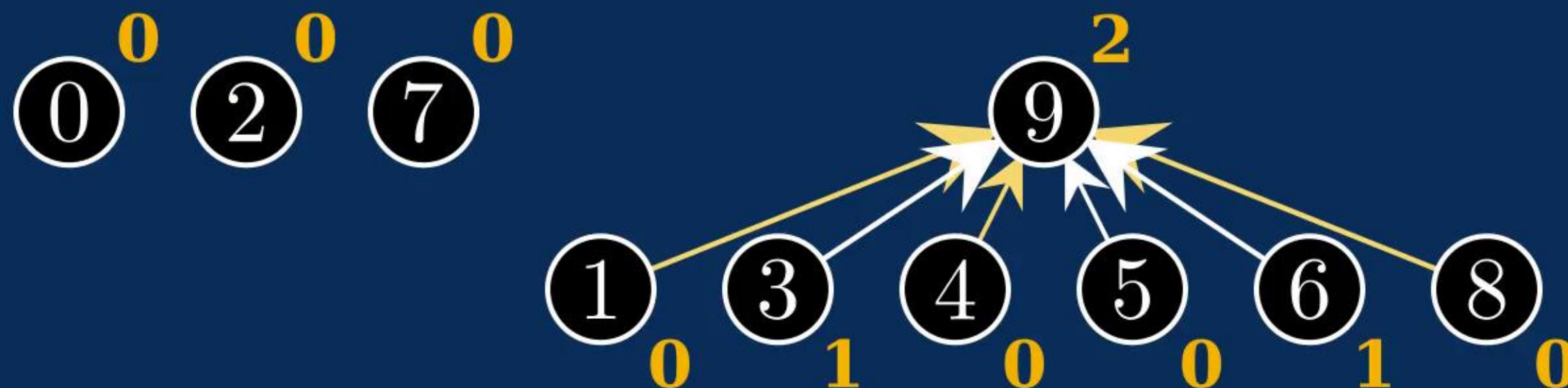
union(8,6)

union(8,5)

union(8,1)

find(1)

find(4)



3 and 6: height 0, but rank 1

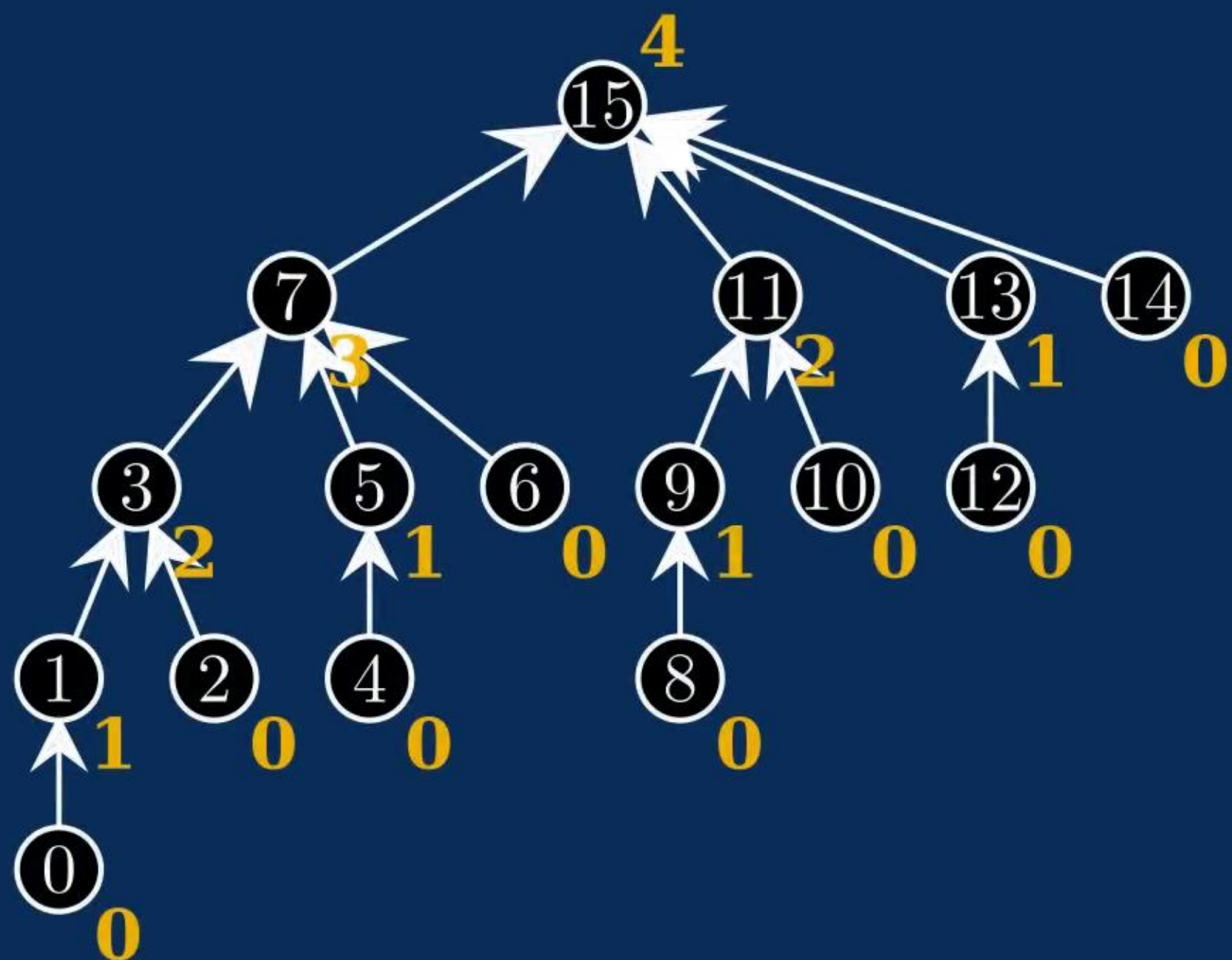
Rank is 'history'

Gold number = rank

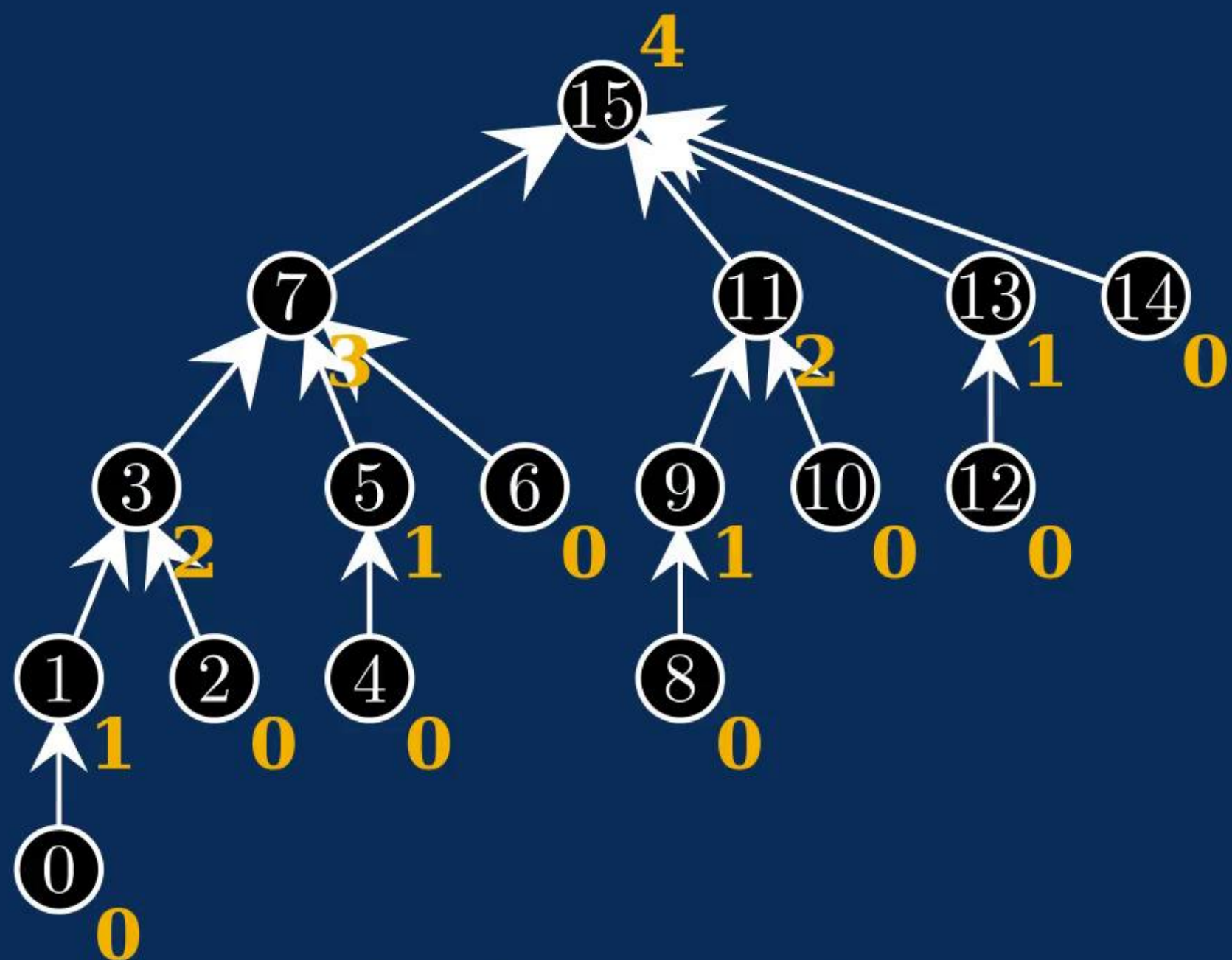
1. Rank only changes while x is a root

1. Rank only changes while x is a root
2. Ranks strictly increase along every path to the root
 - Even after compression: the new parent was an old ancestor

-
1. Rank only changes while x is a root
 2. Ranks strictly increase along every path to the root
 - Even after compression: the new parent was an old ancestor
 3. At most $n/2^r$ nodes have rank r
 - A rank- r node had $\geq 2^r$ nodes below it (doubling again)
 - A rank- r node can't have a rank- r parent or child
 - 'With \$2B you can only have 2 billionaires.'



16 nodes, after 15 unions by size



rank 4 (15)

$$1 \leq 16/2^4 = 1$$

rank 3 (7)

$$1 \leq 16/2^3 = 2$$

rank 2 (3) (11)

$$2 \leq 16/2^2 = 4$$

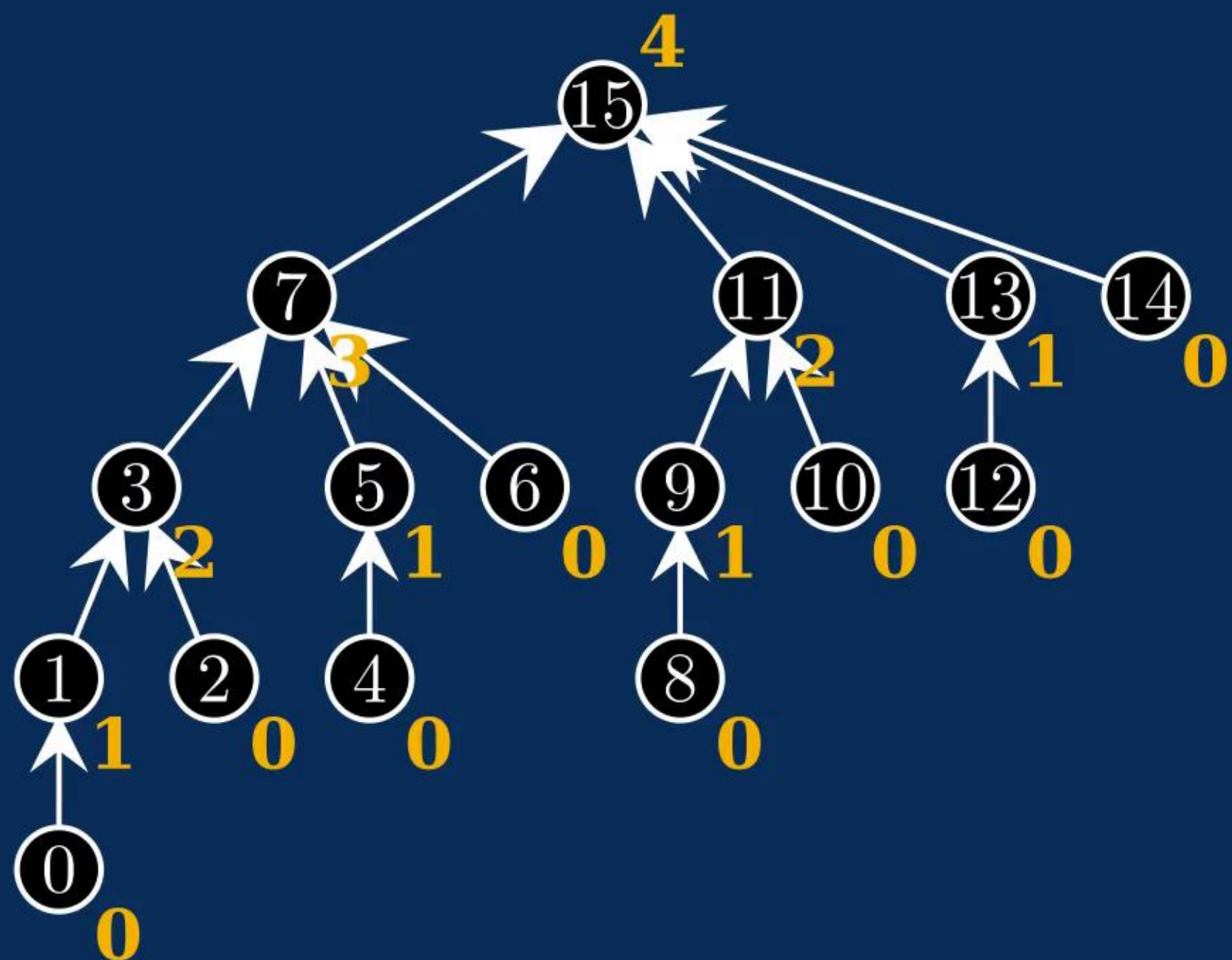
rank 1 (1) (5) (9) (13)

$$4 \leq 16/2^1 = 8$$

rank 0 (0) (2) (4) (6) (8) (10) (12) (14)

$$8 \leq 16/2^0 = 16$$

16 nodes, after 15 unions by size



rank 4 ⑮

$$1 \leq 16/2^4 = 1$$

rank 3 ⑦

$$1 \leq 16/2^3 = 2$$

rank 2 ③ ⑪

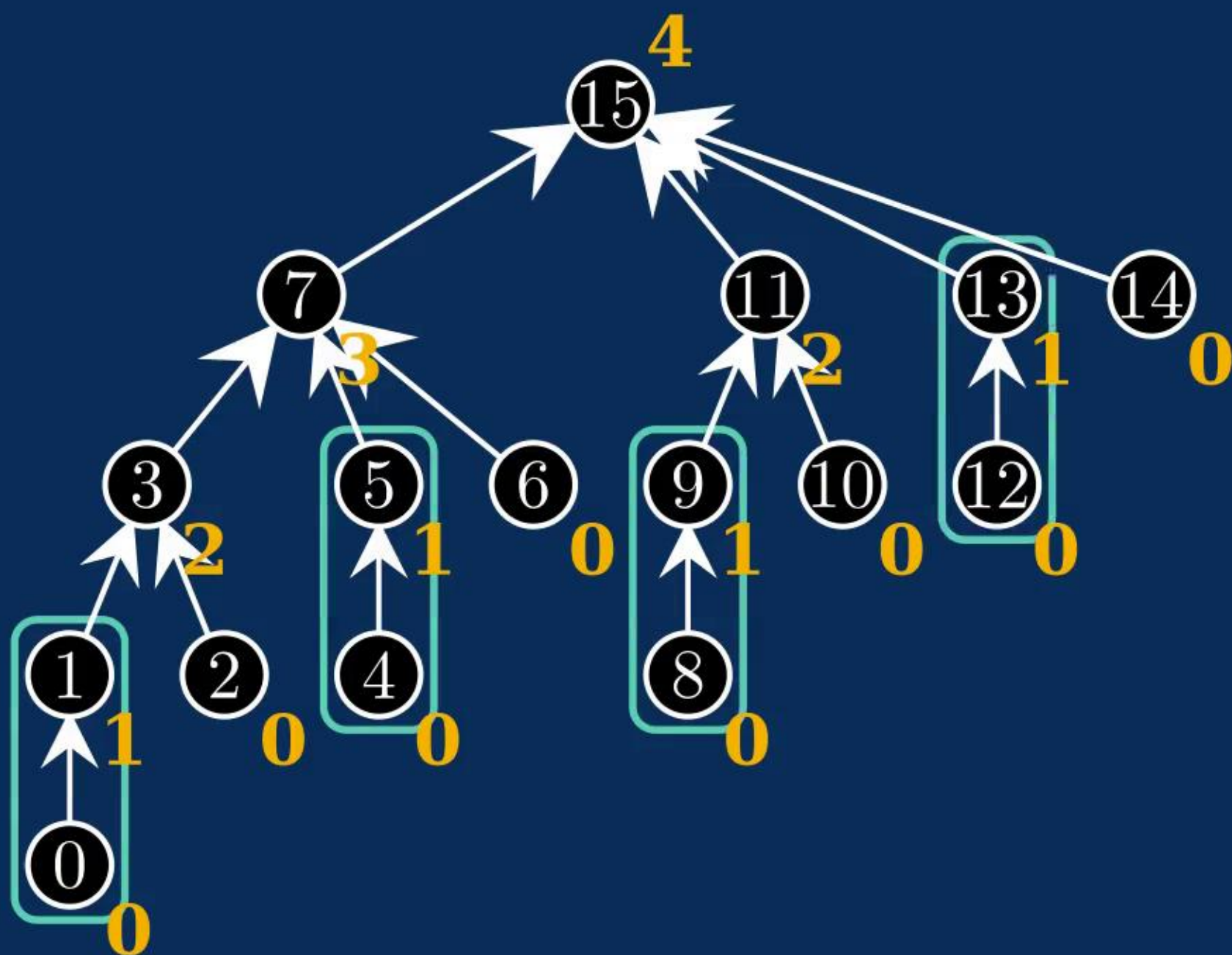
$$2 \leq 16/2^2 = 4$$

rank 1 ① ⑤ ⑨ ⑬

$$4 \leq 16/2^1 = 8$$

rank 0 ① ② ④ ⑥ ⑧ ⑩ ⑫ ⑭ $8 \leq 16/2^0 = 16$

Why? Each rank- r node has $\geq 2^r$ nodes in its subtree,
and two rank- r subtrees never overlap.



rank 4 ⑮

$$1 \leq 16/2^4 = 1$$

rank 3 ⑦

$$1 \leq 16/2^3 = 2$$

rank 2 ③ ⑪

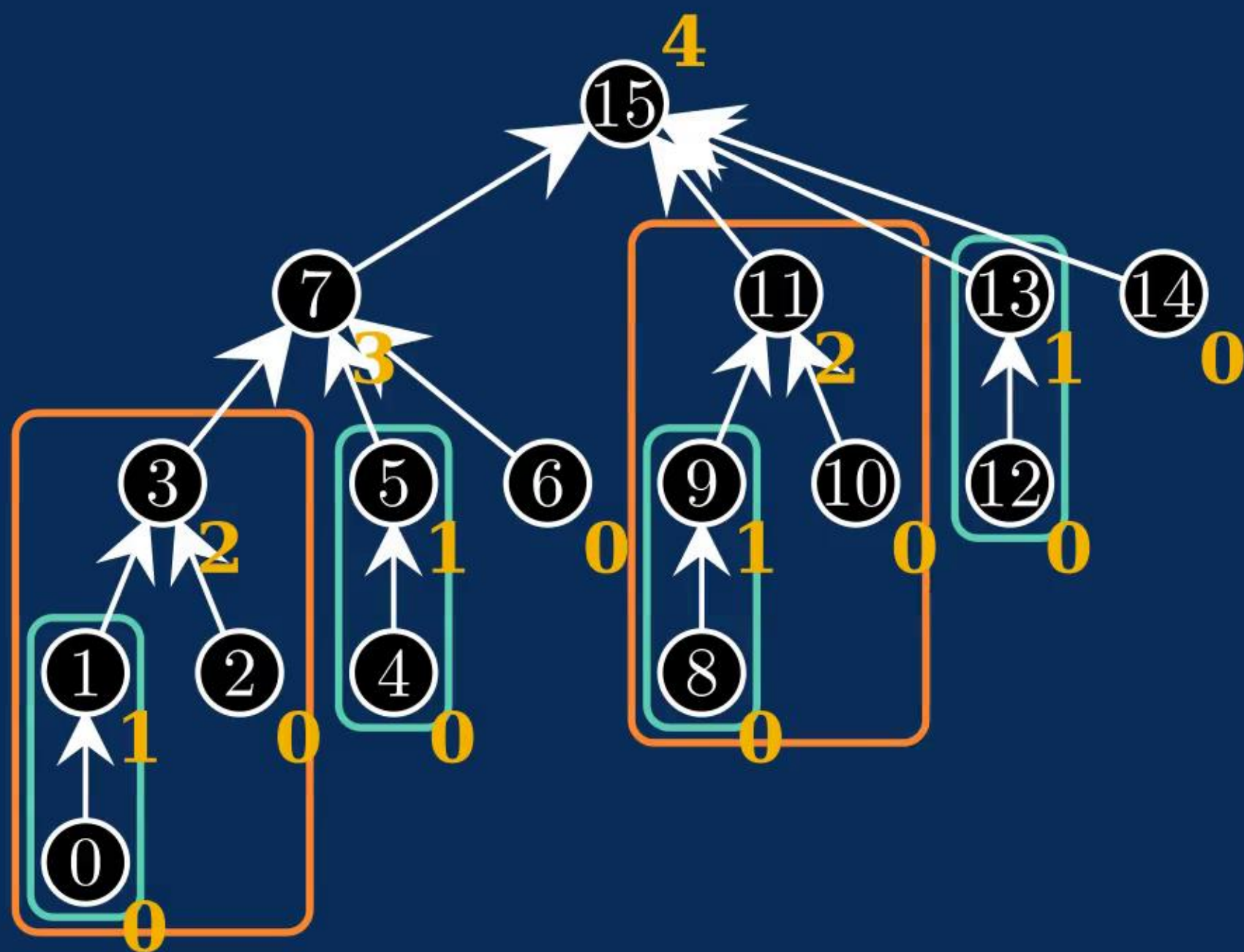
$$2 \leq 16/2^2 = 4$$

rank 1 ① ⑤ ⑨ ⑬

$$4 \leq 16/2^1 = 8$$

rank 0 ① ② ④ ⑥ ⑧ ⑩ ⑫ ⑭ $8 \leq 16/2^0 = 16$

Why? Each rank- r node has $\geq 2^r$ nodes in its subtree,
and two rank- r subtrees never overlap.



rank 4 ⑮

$$1 \leq 16/2^4 = 1$$

rank 3 ⑦

$$1 \leq 16/2^3 = 2$$

rank 2 ③ ⑪

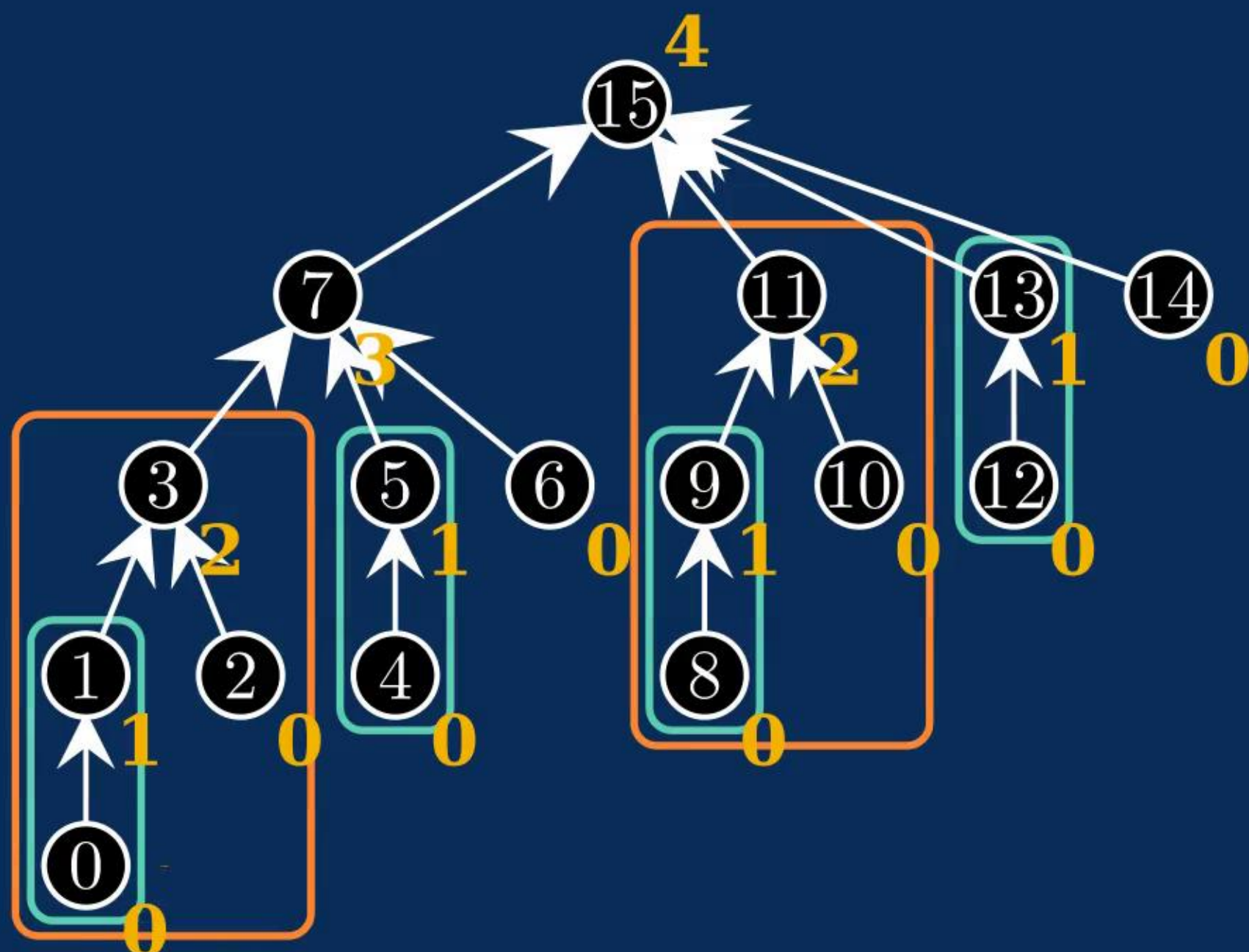
$$2 \leq 16/2^2 = 4$$

rank 1 ① ⑤ ⑨ ⑬

$$4 \leq 16/2^1 = 8$$

rank 0 ① ② ④ ⑥ ⑧ ⑩ ⑫ ⑭ $8 \leq 16/2^0 = 16$

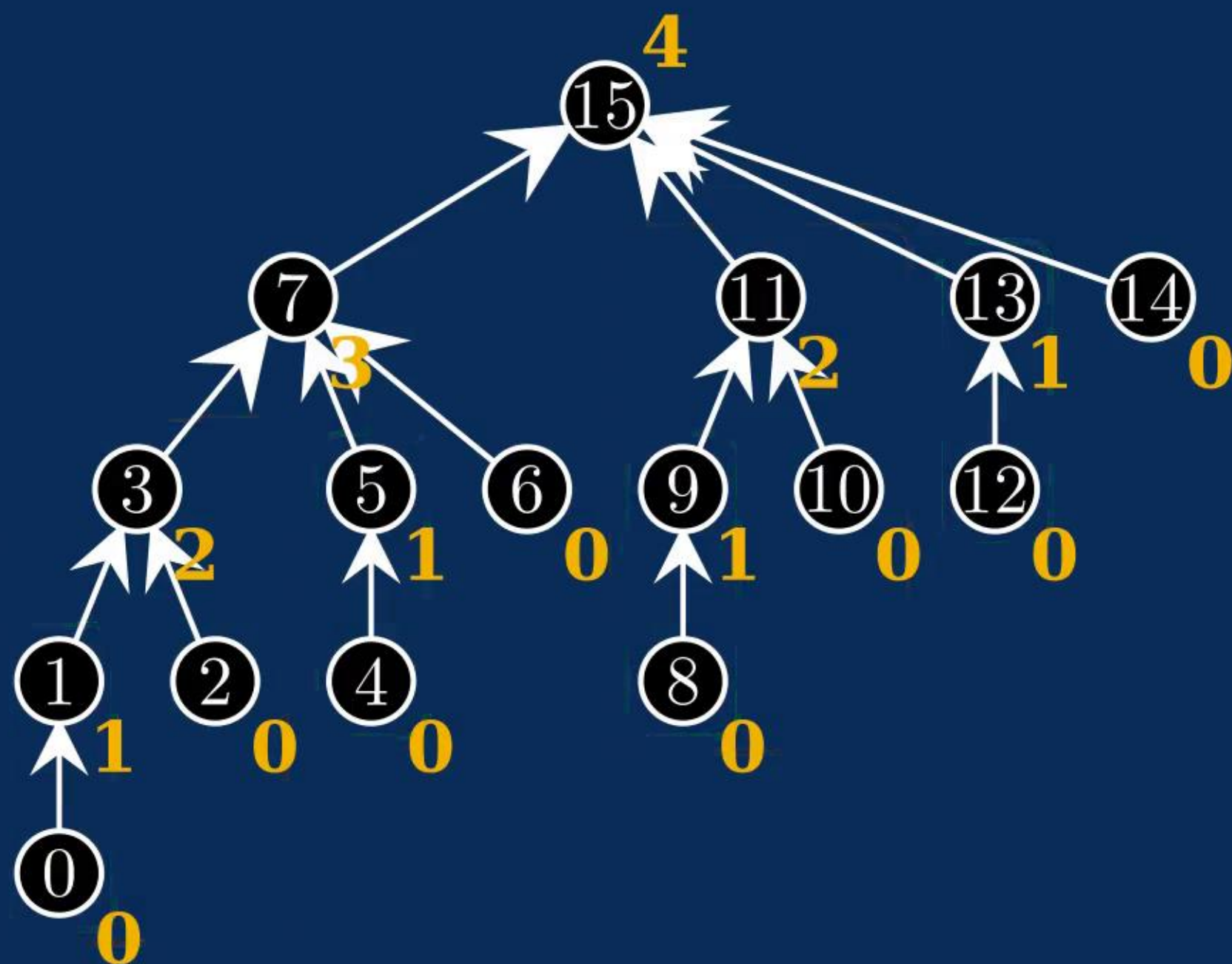
Why? Each rank- r node has $\geq 2^r$ nodes in its subtree,
and two rank- r subtrees never overlap.



Node 0 is in a teal box and an orange box:
that's allowed

rank 4	⑮	$1 \leq 16/2^4 = 1$
rank 3	⑦	$1 \leq 16/2^3 = 2$
rank 2	③ ⑪	$2 \leq 16/2^2 = 4$
rank 1	① ⑤ ⑨ ⑬	$4 \leq 16/2^1 = 8$
rank 0	① ② ④ ⑥ ⑧ ⑩ ⑫ ⑭	$8 \leq 16/2^0 = 16$

Why? Each rank- r node has $\geq 2^r$ nodes in its subtree,
and two rank- r subtrees never overlap.



rank 4 ⑮

$$1 \leq 16/2^4 = 1$$

rank 3 ⑦

$$1 \leq 16/2^3 = 2$$

rank 2 ③ ⑪

$$2 \leq 16/2^2 = 4$$

rank 1 ① ⑤ ⑨ ⑬

$$4 \leq 16/2^1 = 8$$

rank 0 ① ② ④ ⑥ ⑧ ⑩ ⑫ ⑭ $8 \leq 16/2^0 = 16$

Why? Each rank- r node has $\geq 2^r$ nodes in its subtree,
and two rank- r subtrees never overlap.

①¹

rank 4

$$1 \leq 16/2^4 = 1$$

rank 3

$$1 \leq 16/2^3 = 2$$

rank 2

$$2 \leq 16/2^2 = 4$$

rank 1

$$4 \leq 16/2^1 = 8$$

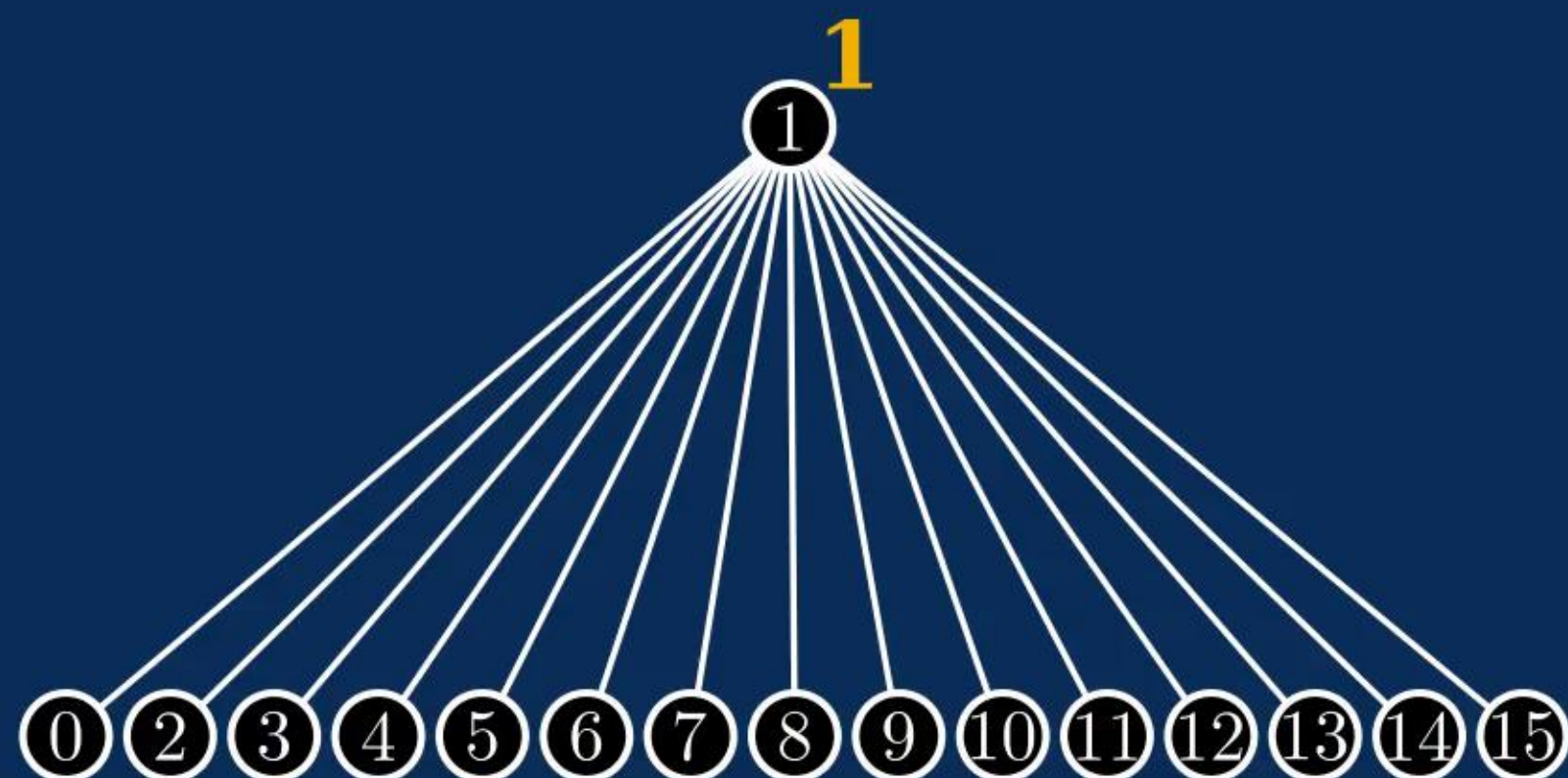
① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩ ⑪ ⑫ ⑬ ⑭ ⑮

rank 0

$$8 \leq 16/2^0 = 16$$

all 15 leaves: rank 0

16 nodes: `union(0,i)` for each i



all 15 leaves: rank 0

rank 4

$$0 \leq 16/2^4 = 1$$

rank 3

$$0 \leq 16/2^3 = 2$$

rank 2

$$0 \leq 16/2^2 = 4$$

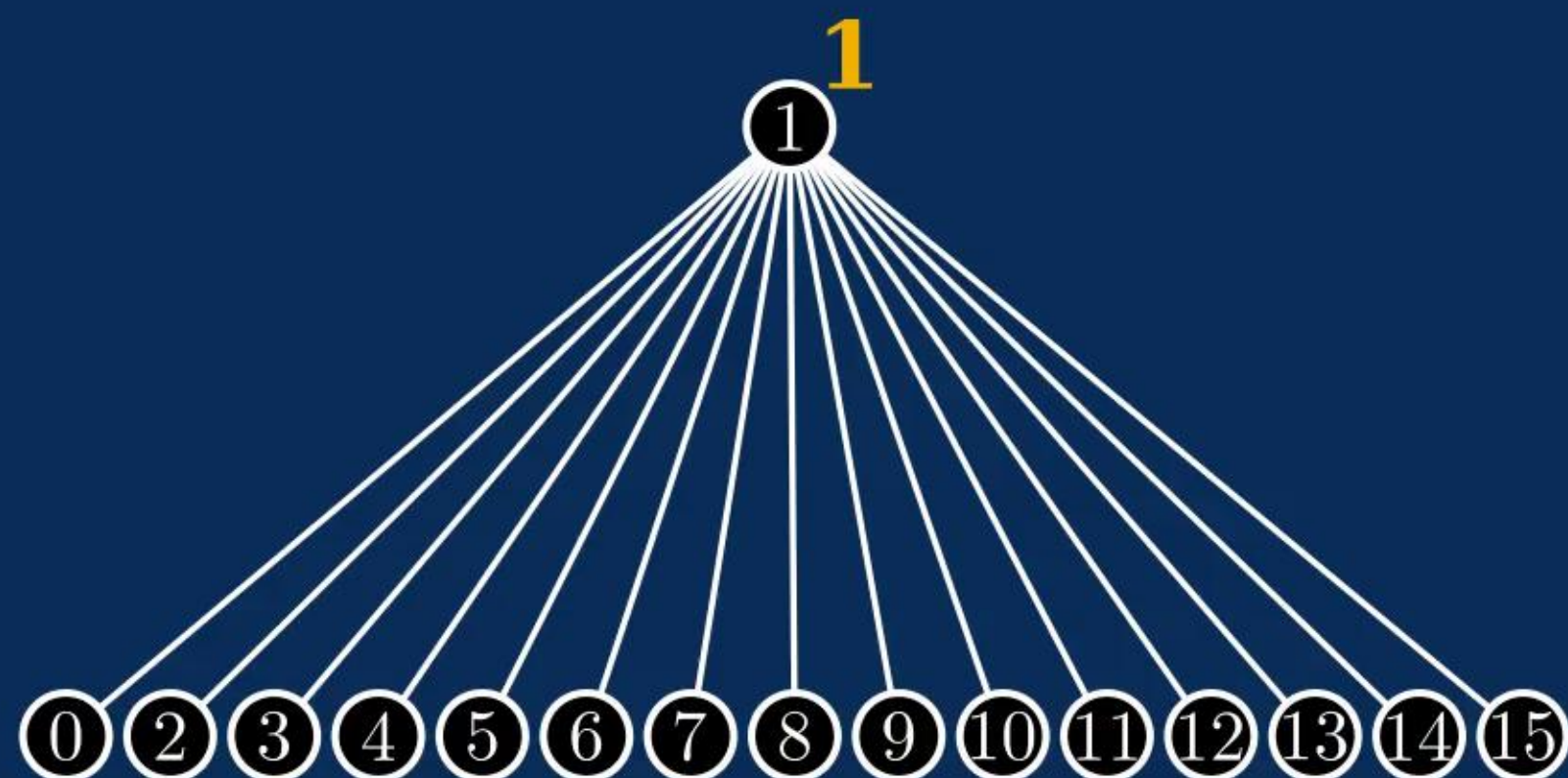
rank 1

$$1 \leq 16/2^1 = 8$$

rank 0

$$15 \leq 16/2^0 = 16$$

16 nodes: $\text{union}(0, i)$ for each i



all 15 leaves: rank 0

rank 4

$$0 \leq 16/2^4 = 1$$

rank 3

$$0 \leq 16/2^3 = 2$$

rank 2

$$0 \leq 16/2^2 = 4$$

rank 1

$$1 \leq 16/2^1 = 8$$

rank 0

$$15 \leq 16/2^0 = 16$$

16 nodes: `union(0,i)` for each i

The bound is a ceiling

Who Pays?

in 

out 

push / pop at the right end

moves so far: 0 elements so far: 0



enqueue(1,2,3,4)

cost: 4



push / pop at the right end

moves so far: 0 elements so far: 4



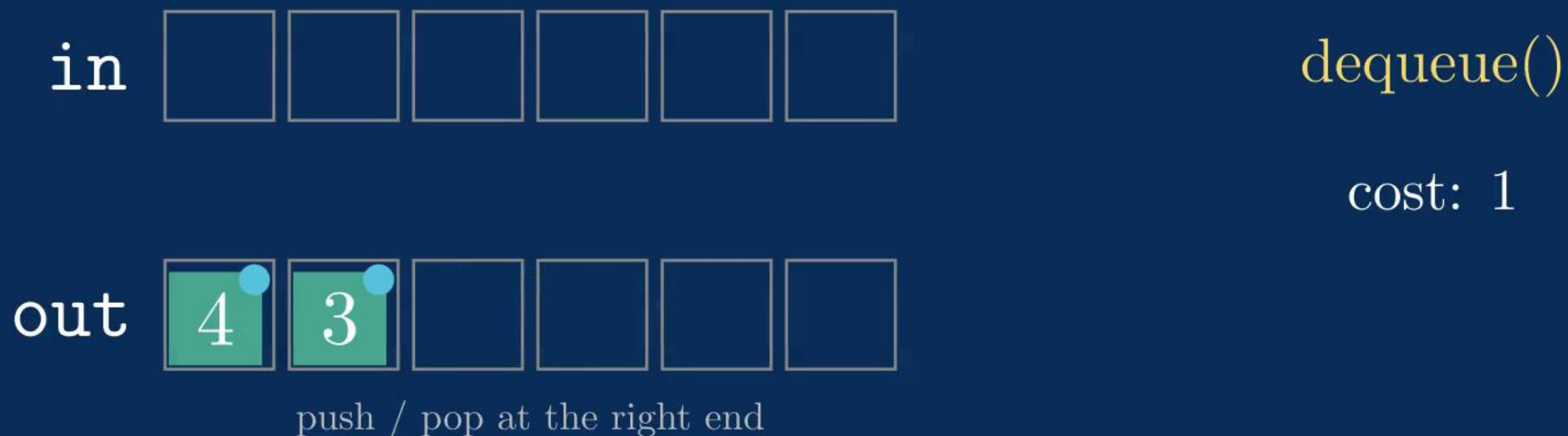
dequeue()

cost: 4 moves + 1



push / pop at the right end

moves so far: 4 elements so far: 4



moves so far: 4 elements so far: 4



enqueue(5,6)

cost: 2



push / pop at the right end

moves so far: 4 elements so far: 6



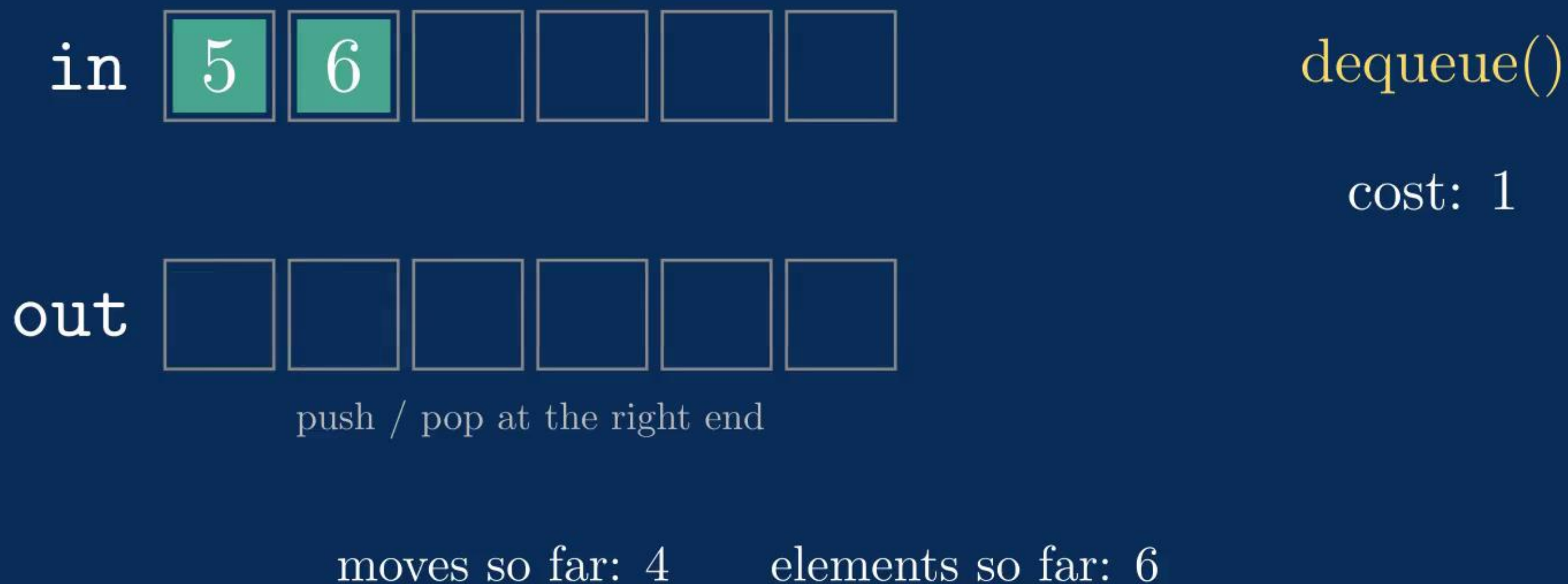
dequeue()

cost: 1



push / pop at the right end

moves so far: 4 elements so far: 6



in



dequeue()

cost: 2 moves + 1

out



push / pop at the right end

moves so far: 6 elements so far: 6

in 

out 

push / pop at the right end One dequeue can cost $n...$

moves so far: 6 elements so far: 6

in 

out 

push / pop at the right end

One dequeue can cost n ...

...but every element is moved *at most once*.

moves so far: 6

elements so far: 6

A dequeue can be long: each extra step **moves an element**.

A dequeue can be long: each extra step **moves an element**.

A find can be long: each extra step **walks past a node**.

A dequeue can be long: each extra step **moves an element**.

A find can be long: each extra step **walks past a node**.

Compression links that node to the root, so its parent's rank goes up.

A dequeue can be long: each extra step **moves an element**.

A find can be long: each extra step **walks past a node**.

Compression links that node to the root, so its parent's rank goes up.

Bill each extra step to the *element* or *node*, not to the operation.

Bill each extra step to the *element* or *node*, not to the operation.

	Two-array queue	Up-trees
Bill per operation		
Lifetime bill		

Bill each extra step to the *element* or *node*, not to the operation.

	Two-array queue	Up-trees
Bill per operation	$\mathcal{O}(1)$ per operation	
Lifetime bill	each element: moved ≤ 1 time	

Bill each extra step to the *element* or *node*, not to the operation.

	Two-array queue	Up-trees
Bill per operation	$\mathcal{O}(1)$ per operation	a find: $\leq \log^* n + 1$
Lifetime bill	each element: moved ≤ 1 time	each node: pays $\leq 2^k$ times

Bill each extra step to the *element* or *node*, not to the operation.

	Two-array queue	Up-trees
Bill per operation	$\mathcal{O}(1)$ per operation	a find: $\leq \log^* n + 1$
Lifetime bill	each element: moved ≤ 1 time	each node: pays $\leq 2^k$ times

(k : the node's rank block $(k, 2^k]$ — more on this next)

Block	Ranks
0	{0}
1	{1}
2	{2}
3	{3, 4}
4	{5..16}
5	{17..65536}
6	{65537..2 ⁶⁵⁵³⁶ }

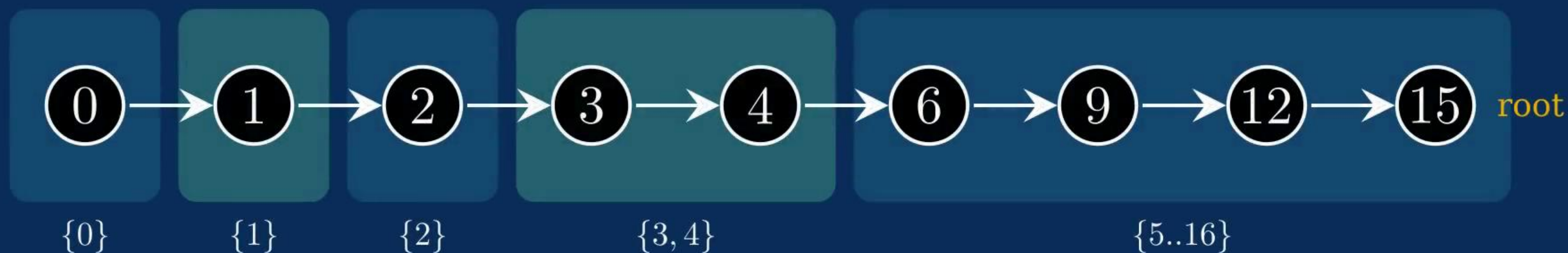
Block	Ranks
0	{0}
1	{1}
2	{2}
3	{3, 4}
4	{5..16}
5	{17..65536}
6	{65537.. 2^{65536} }

Each block ends at
 $2^{(\text{previous end})}$

Block	Ranks
0	{0}
1	{1}
2	{2}
3	{3, 4}
4	{5..16}
5	{17..65536}
6	{65537.. 2^{65536} }

Each block ends at
 $2^{(\text{previous end})}$

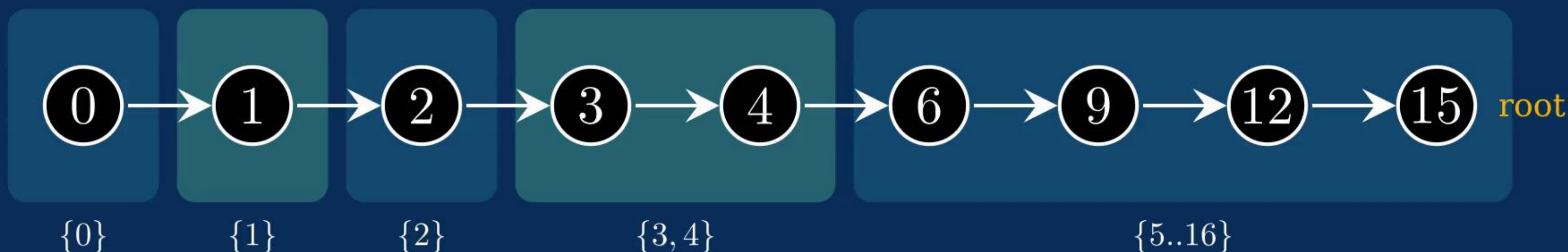
Ranks are $\leq \log_2 n$,
so there are only about
 $\log^* n$ blocks



Labels are ranks (a made-up path)

find pays: parent is the root, or in a different block

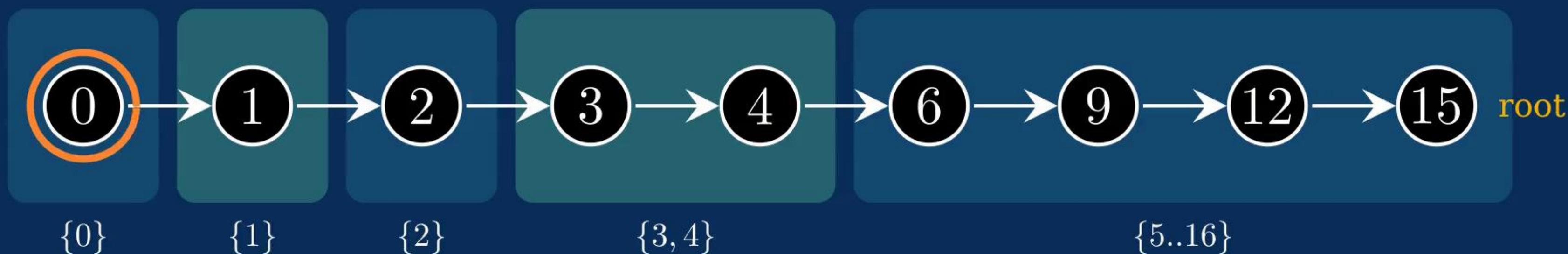
node pays: parent is in the same block



Labels are ranks (a made-up path)

find pays: parent is the root, or in a different block

node pays: parent is in the same block



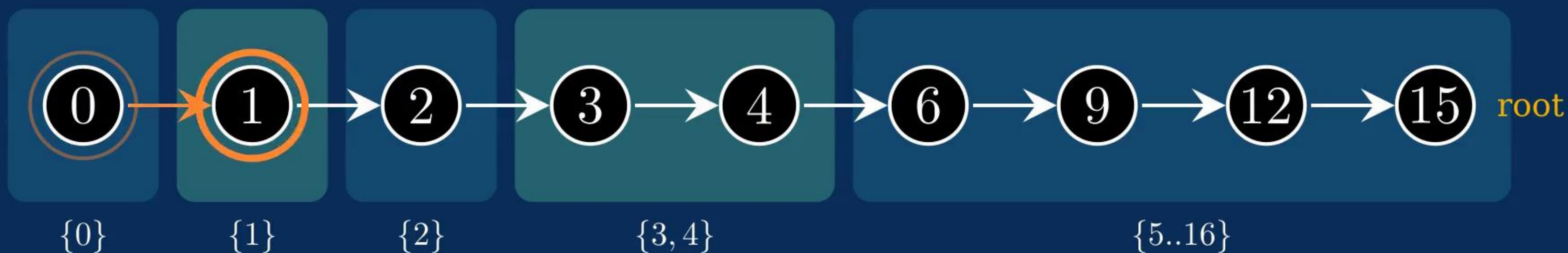
Labels are ranks (a made-up path)

find pays: 0

nodes pay: 0

find pays: parent is the root, or in a different block

node pays: parent is in the same block



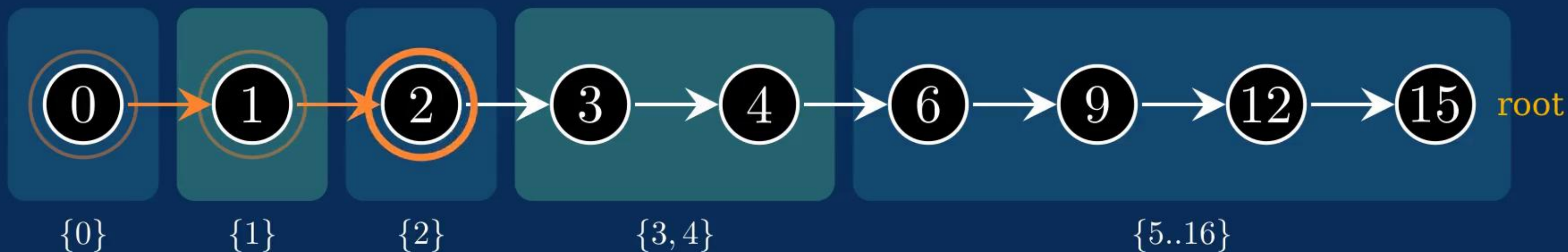
Labels are ranks (a made-up path)

find pays: 1

nodes pay: 0

find pays: parent is the root, or in a different block

node pays: parent is in the same block



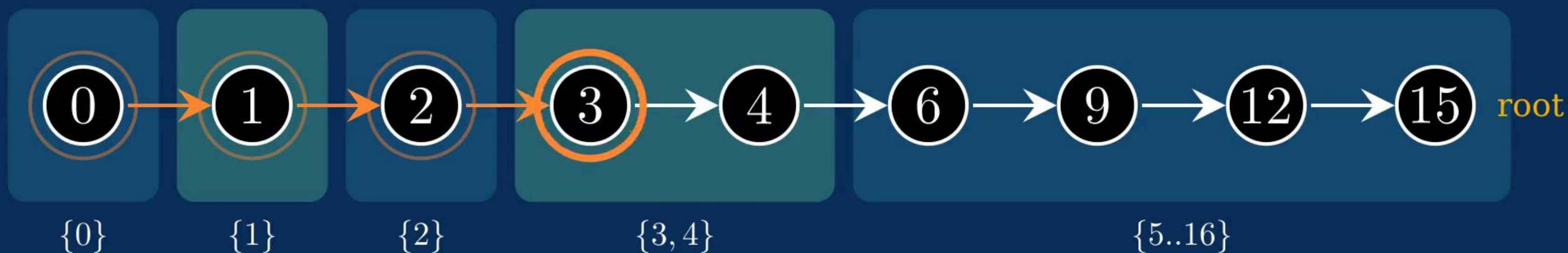
Labels are ranks (a made-up path)

find pays: 2

nodes pay: 0

find pays: parent is the root, or in a different block

node pays: parent is in the same block



Labels are ranks (a made-up path)

find pays: 3

nodes pay: 0

find pays: parent is the root, or in a different block

node pays: parent is in the same block



Labels are ranks (a made-up path)

find pays: 3

nodes pay: 1

find pays: parent is the root, or in a different block

node pays: parent is in the same block



Labels are ranks (a made-up path)

find pays: 4

nodes pay: 1

find pays: parent is the root, or in a different block

node pays: parent is in the same block



Labels are ranks (a made-up path)

find pays: 4

nodes pay: 2

find pays: parent is the root, or in a different block

node pays: parent is in the same block



Labels are ranks (a made-up path)

find pays: 4

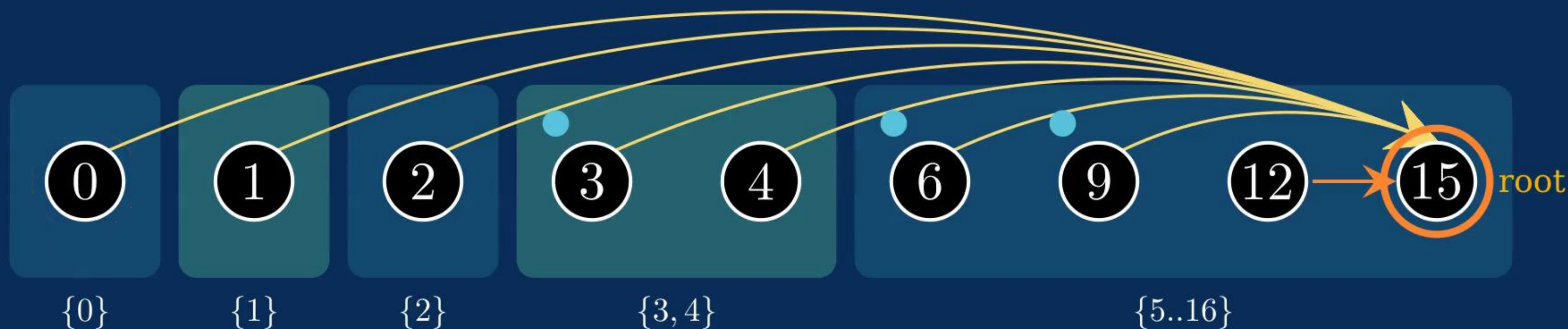
nodes pay: 3

node pays: parent is in the same block



find pays: 5

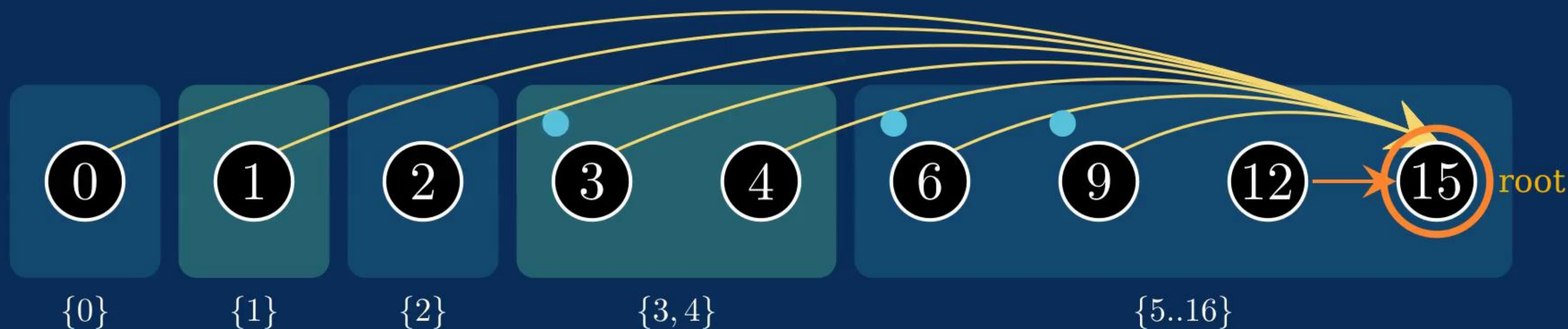
nodes pay: 3



find pays: 5

nodes pay: 3

Rank 3: parent went $4 \rightarrow 15$, left block $\{3, 4\}$: never pays again



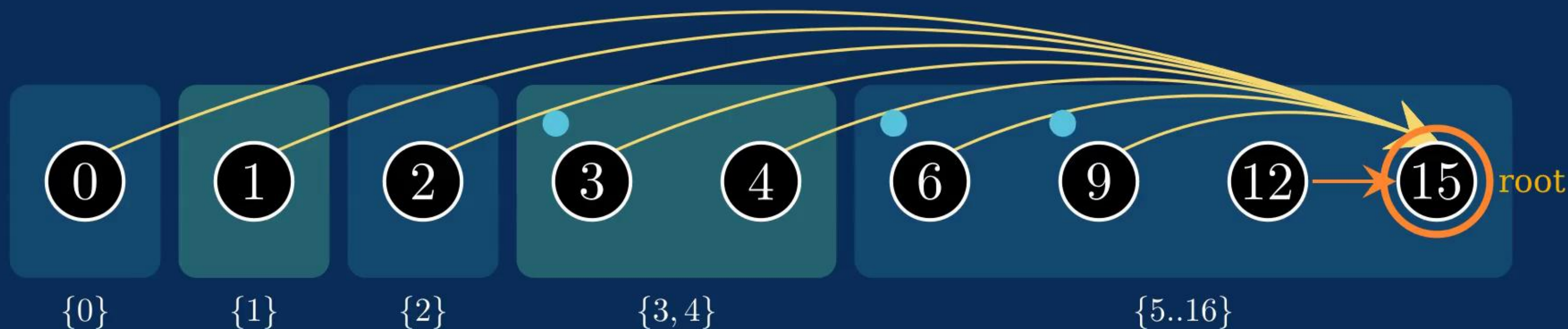
Labels are ranks (a made-up path)

find pays: 5

nodes pay: 3

Rank 3: parent went $4 \rightarrow 15$, left block $\{3, 4\}$: never pays again

Ranks 6, 9: parents went $9, 12 \rightarrow 15$, still in $\{5..16\}$: may pay again...



Labels are ranks (a made-up path)

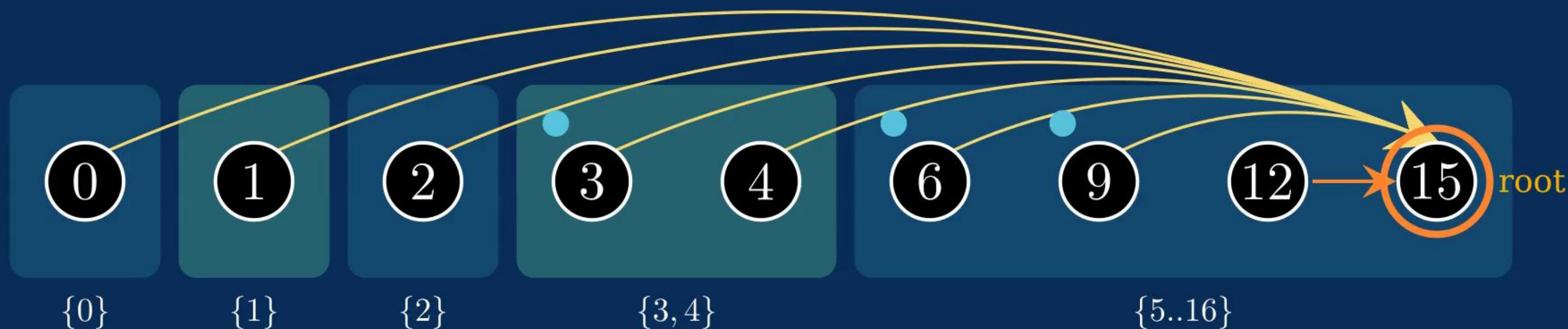
find pays: 5

nodes pay: 3

Rank 3: parent went $4 \rightarrow 15$, left block $\{3, 4\}$: never pays again

Ranks 6, 9: parents went $9, 12 \rightarrow 15$, still in $\{5..16\}$: may pay again...

...but every payment raises the parent's rank, so at most 16 times



Labels are ranks (a made-up path)

find pays: 5

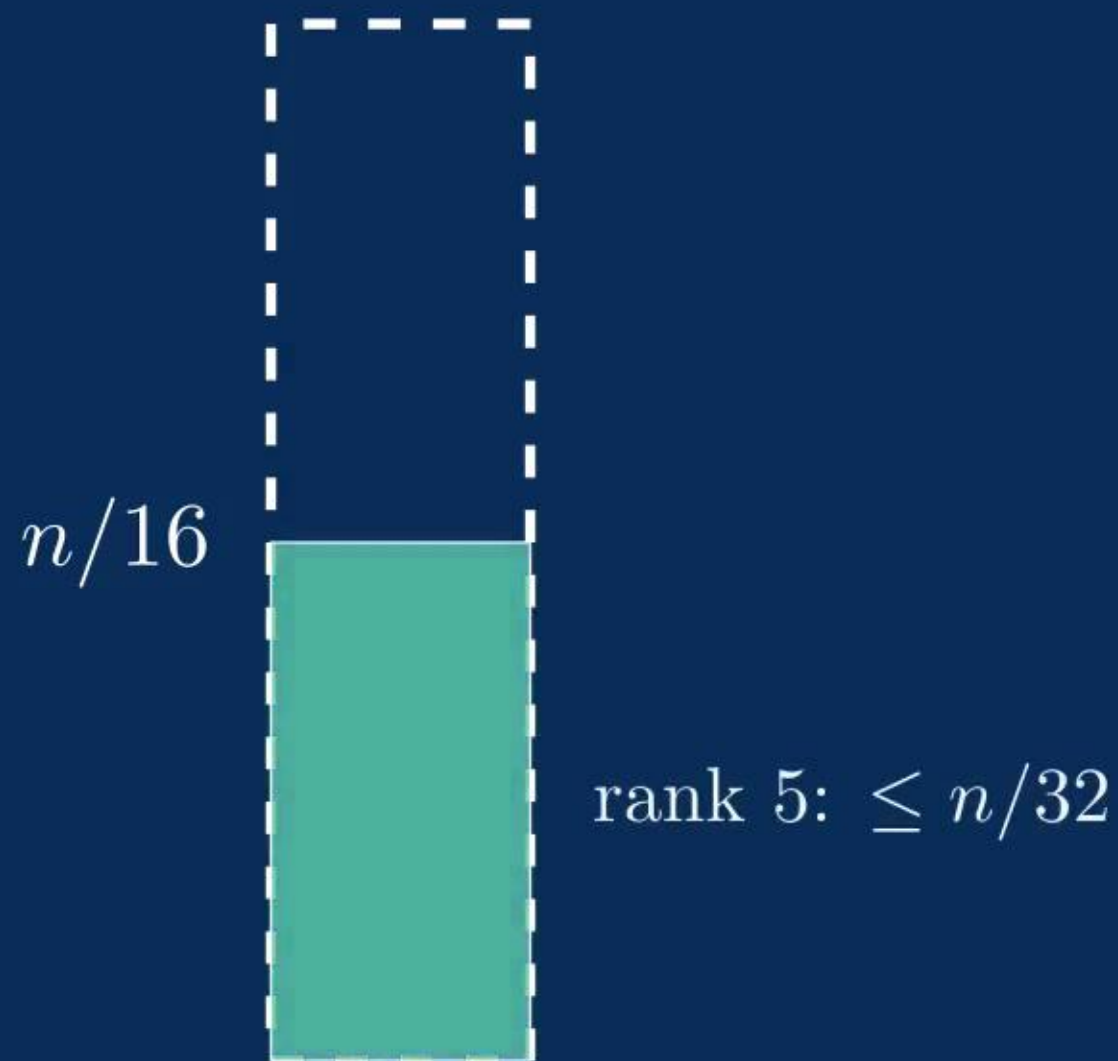
nodes pay: 3

How many nodes?

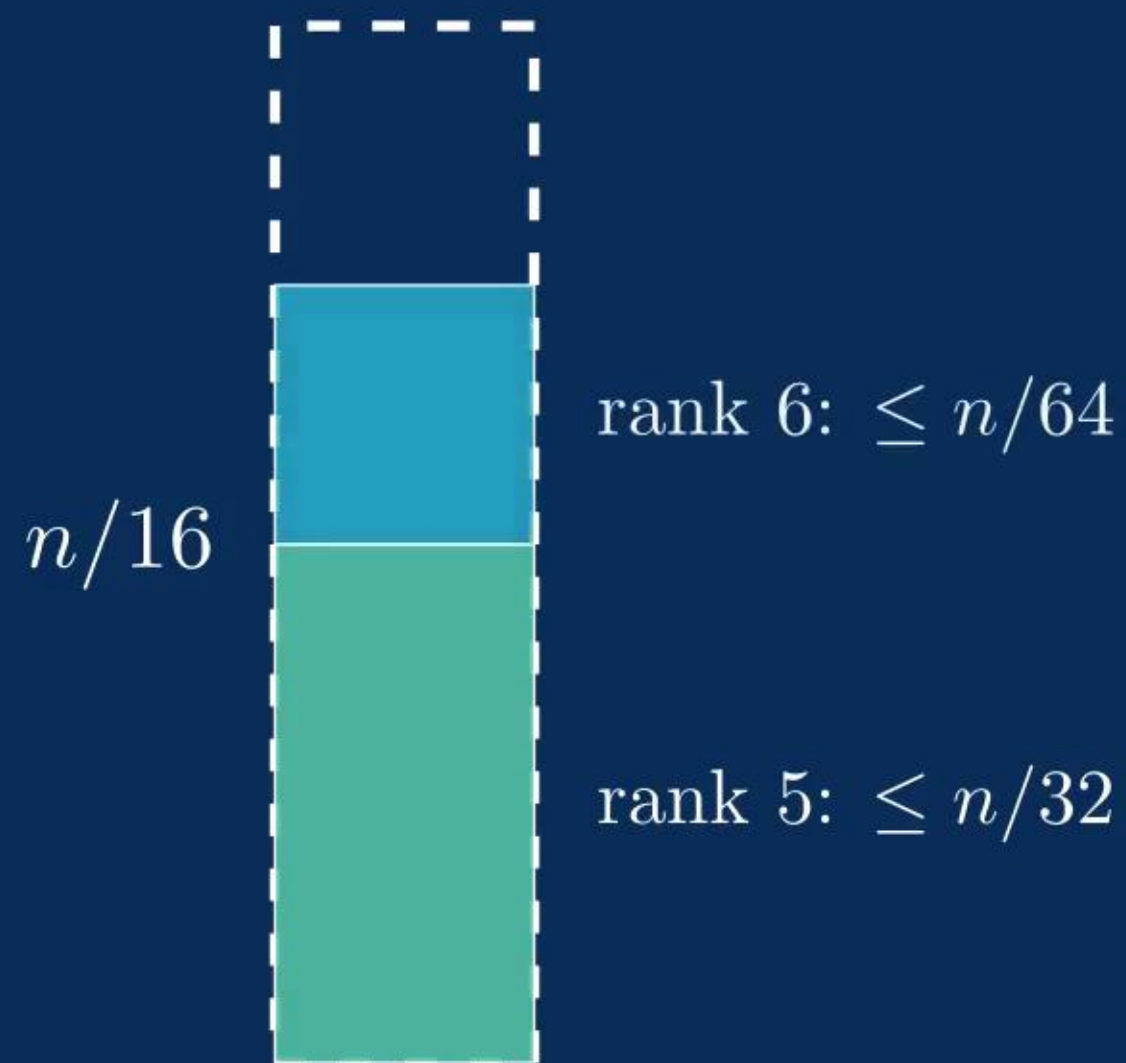


$n/16$

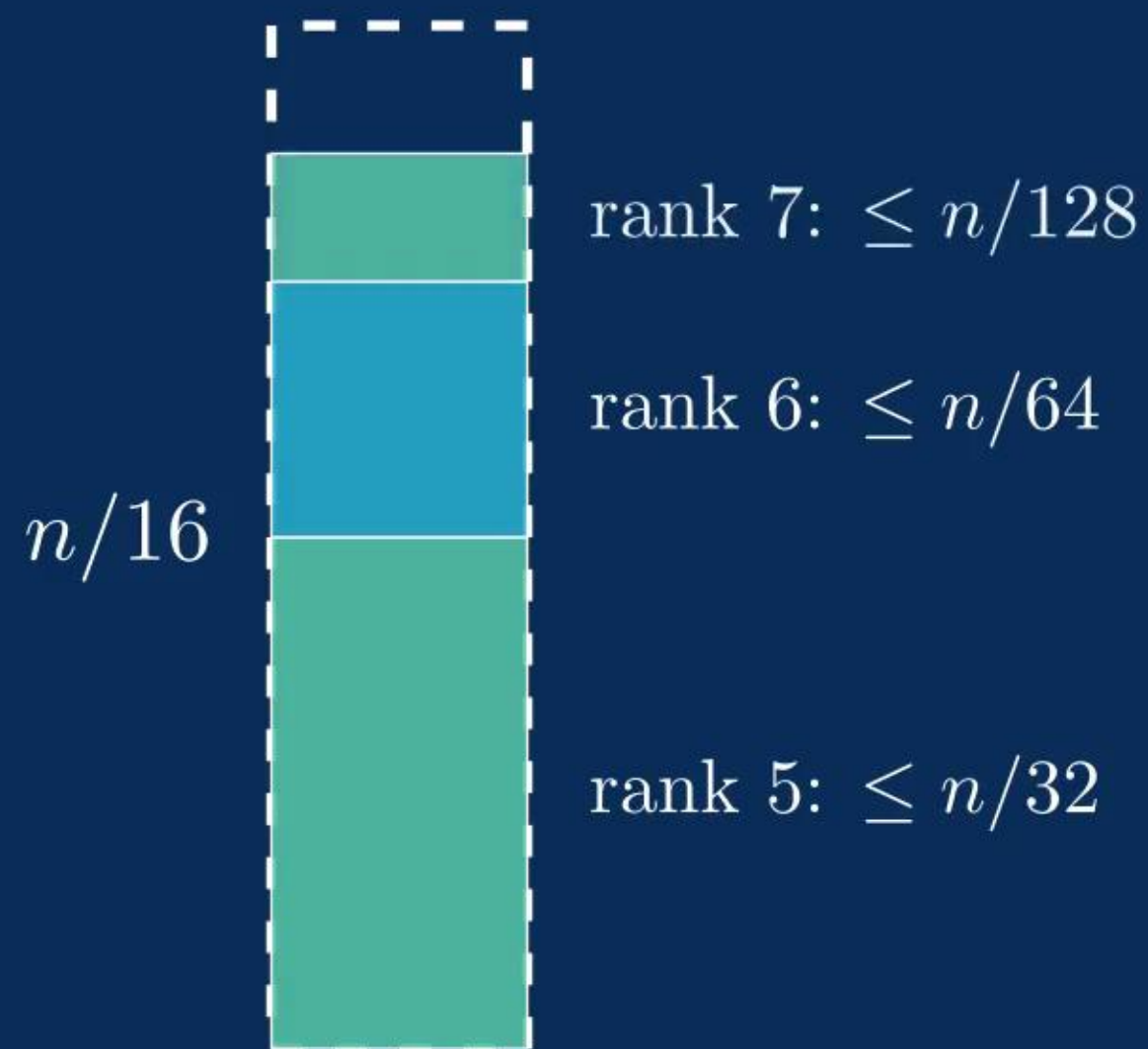
How many nodes?



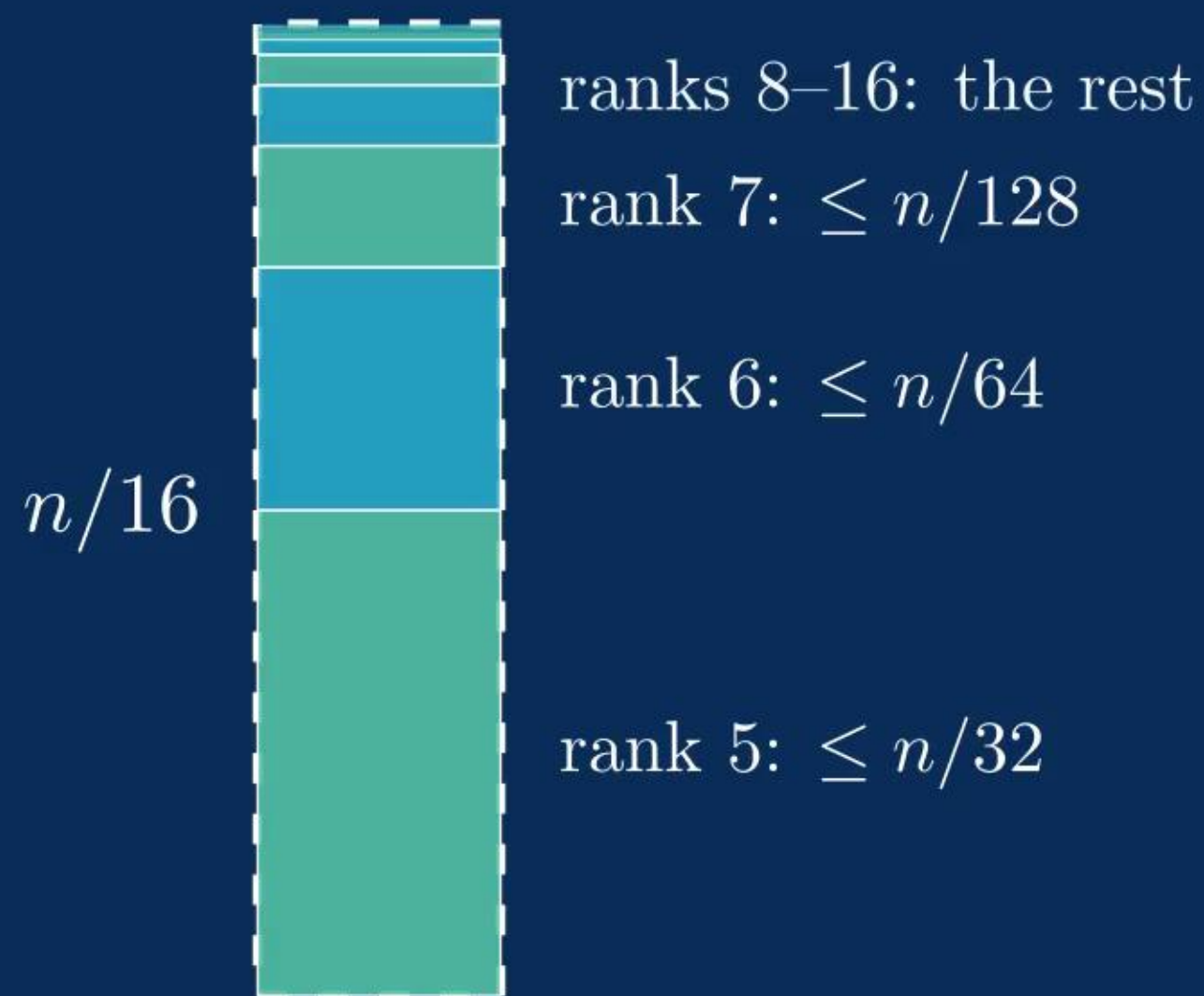
How many nodes?



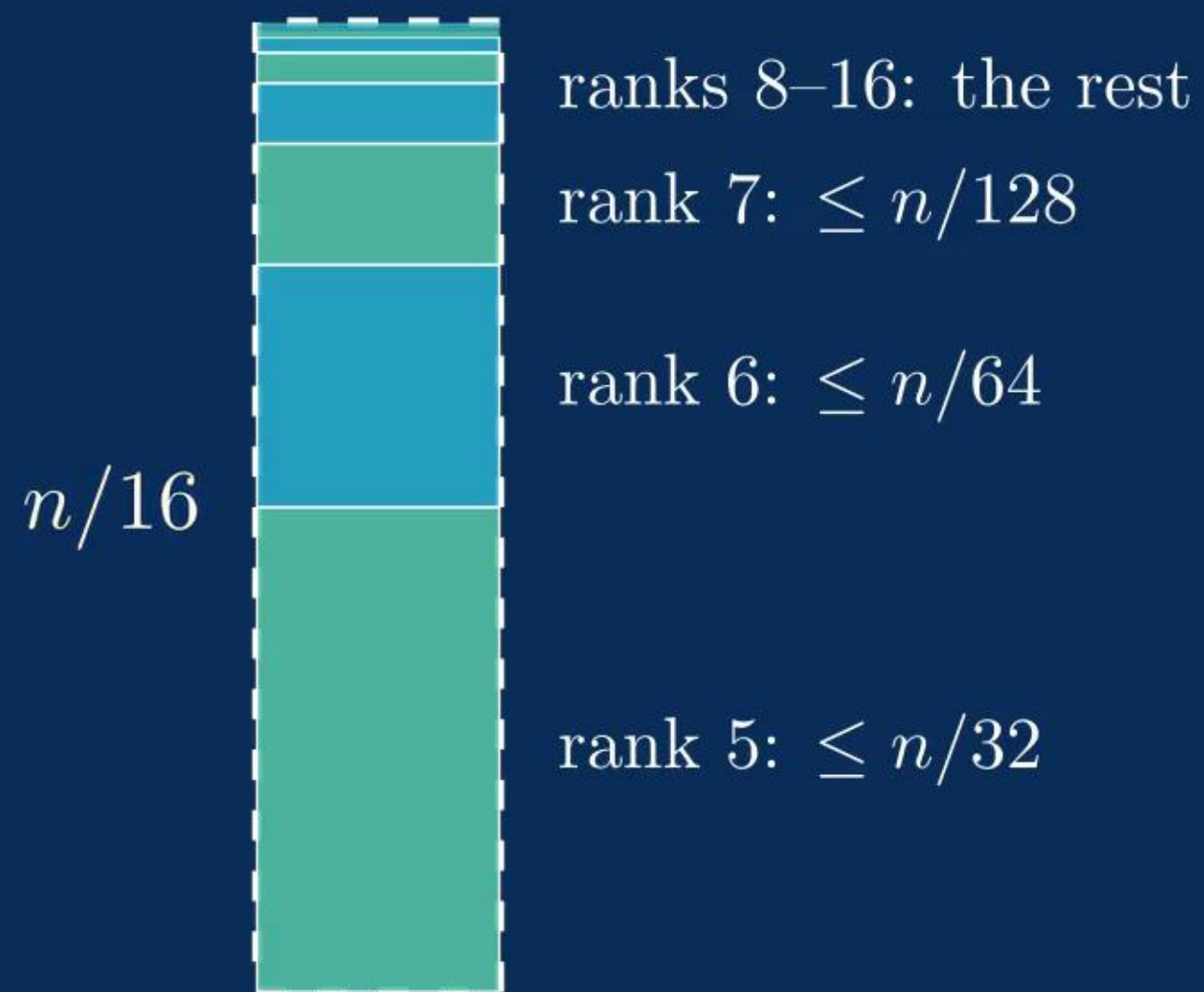
How many nodes?



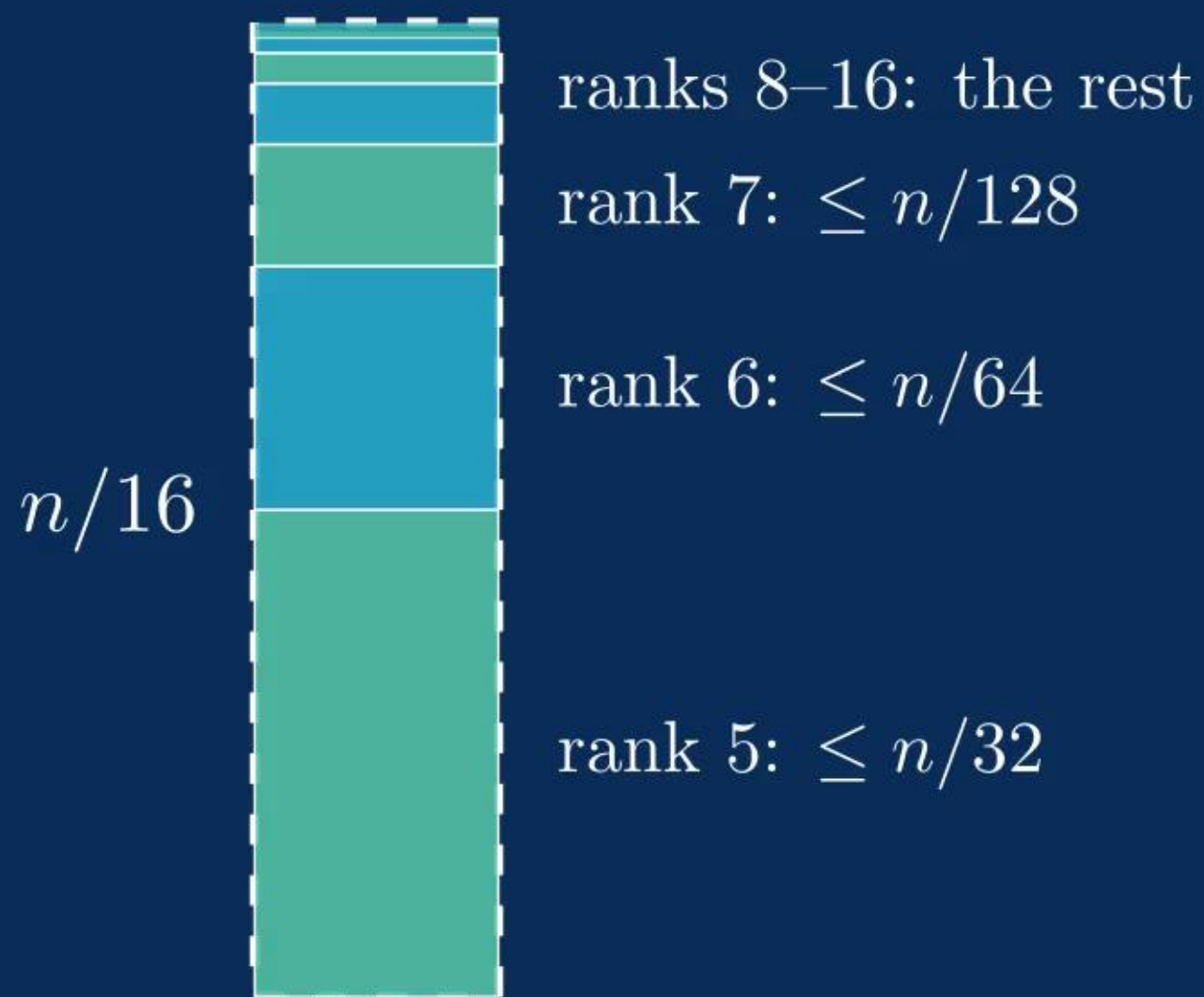
How many nodes?



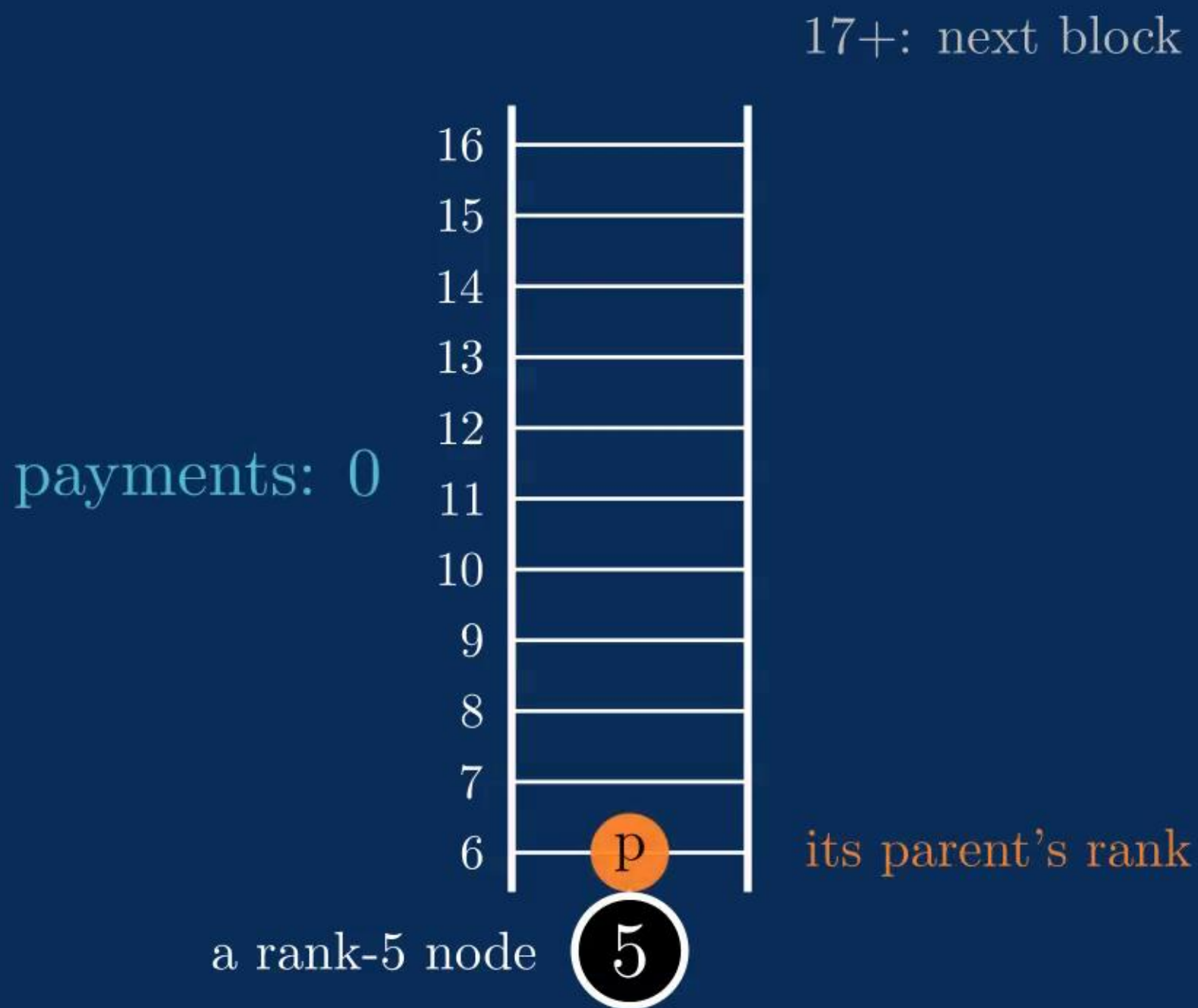
How many nodes?



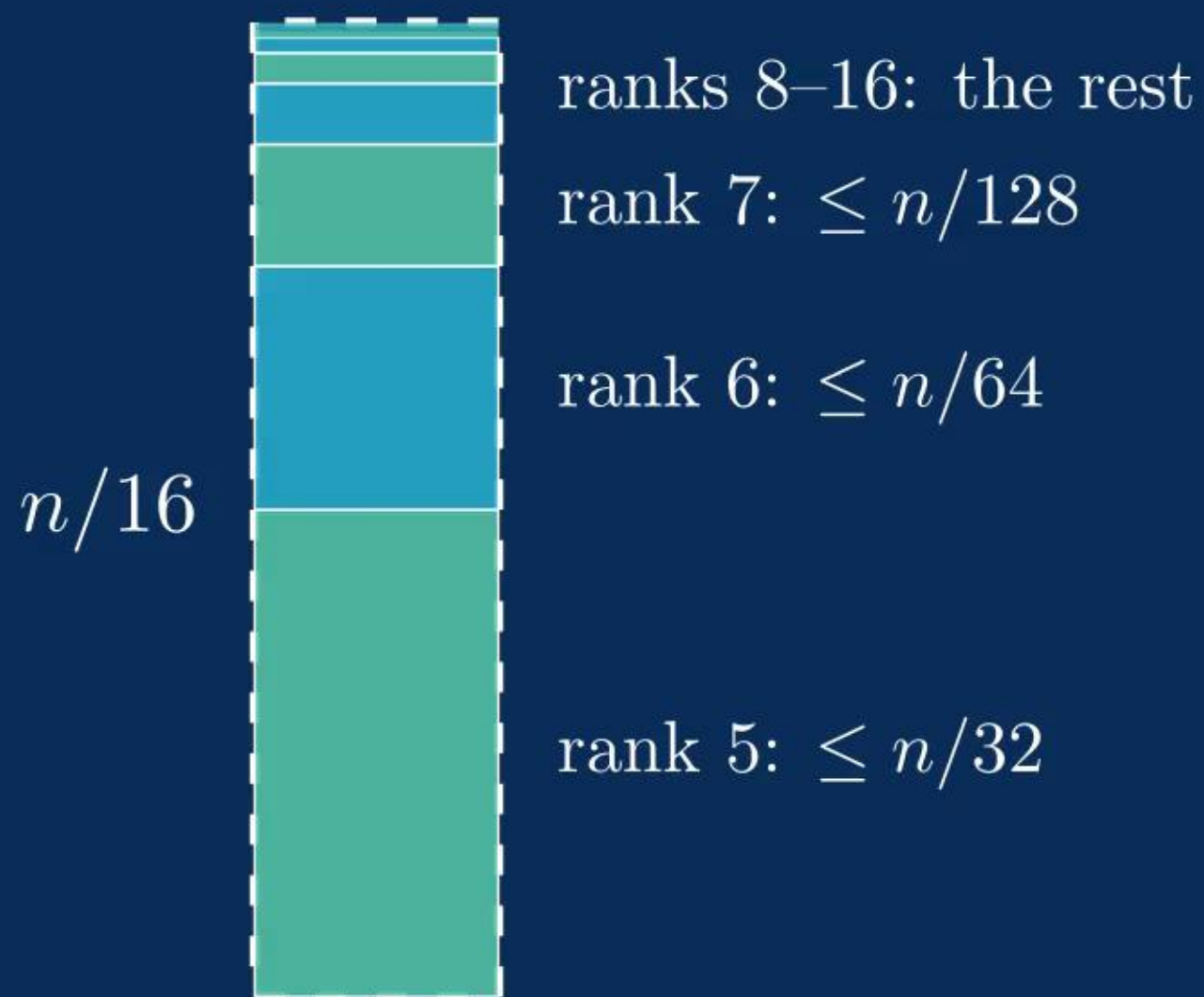
How many nodes?



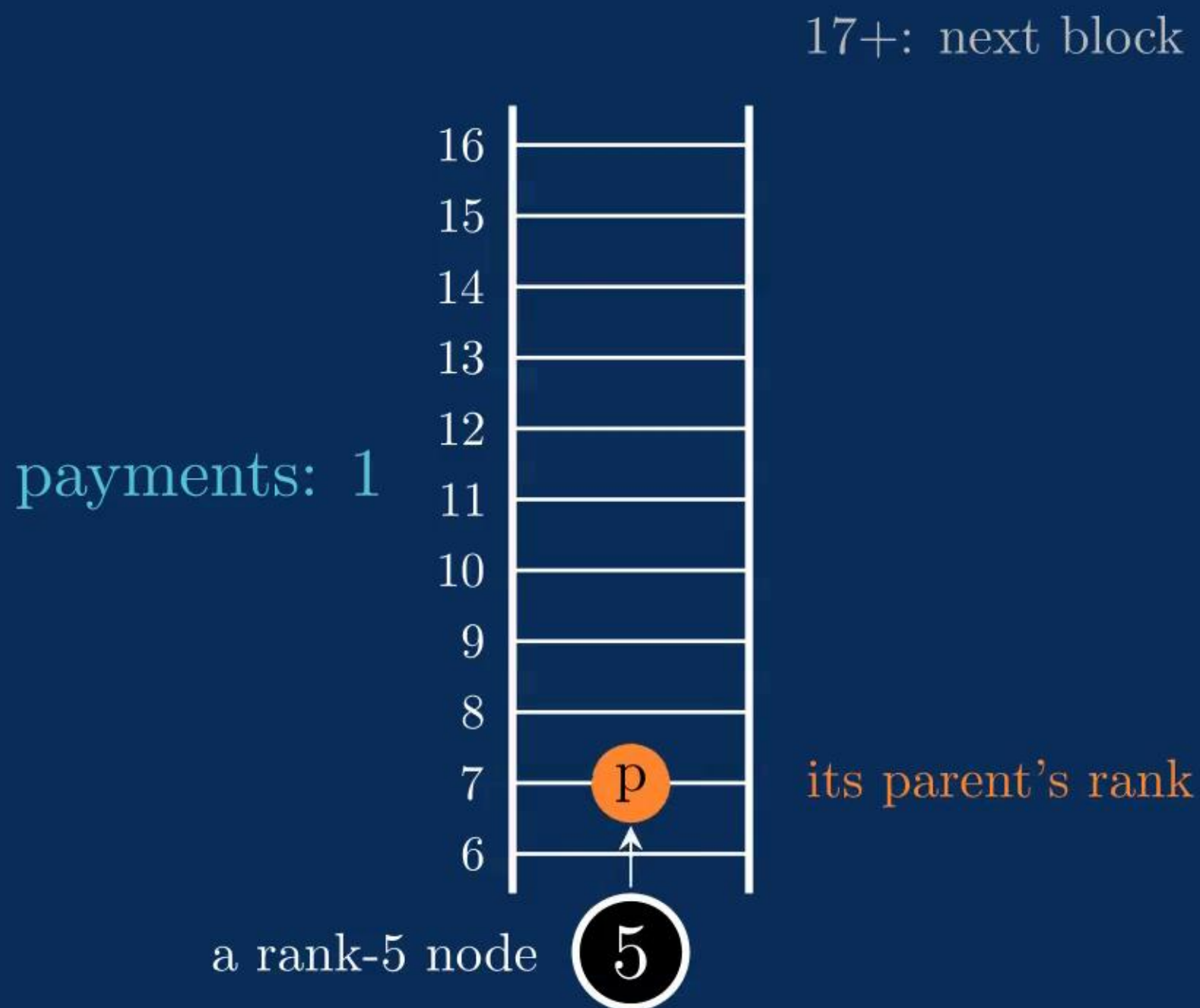
How many payments each?



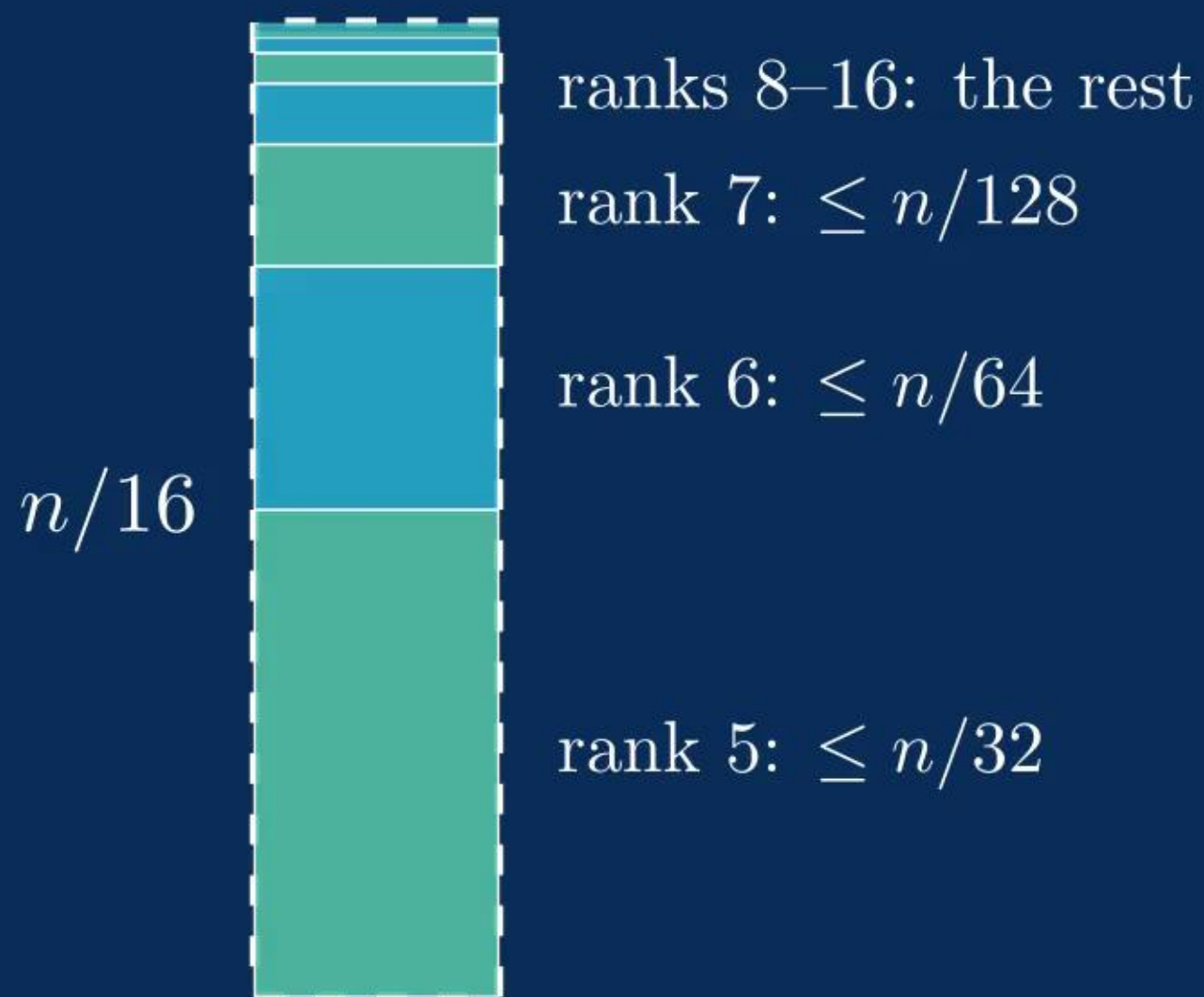
How many nodes?



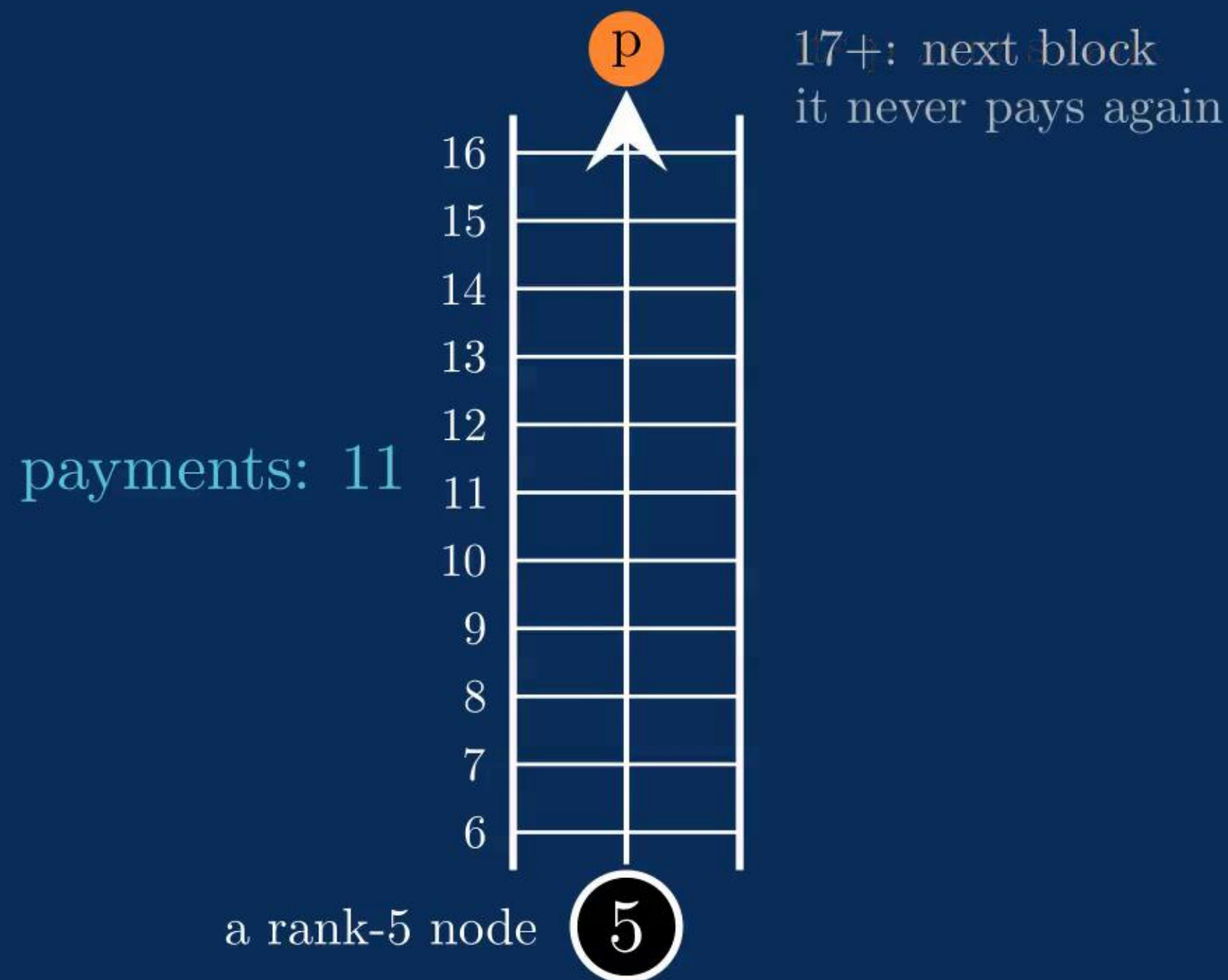
How many payments each?



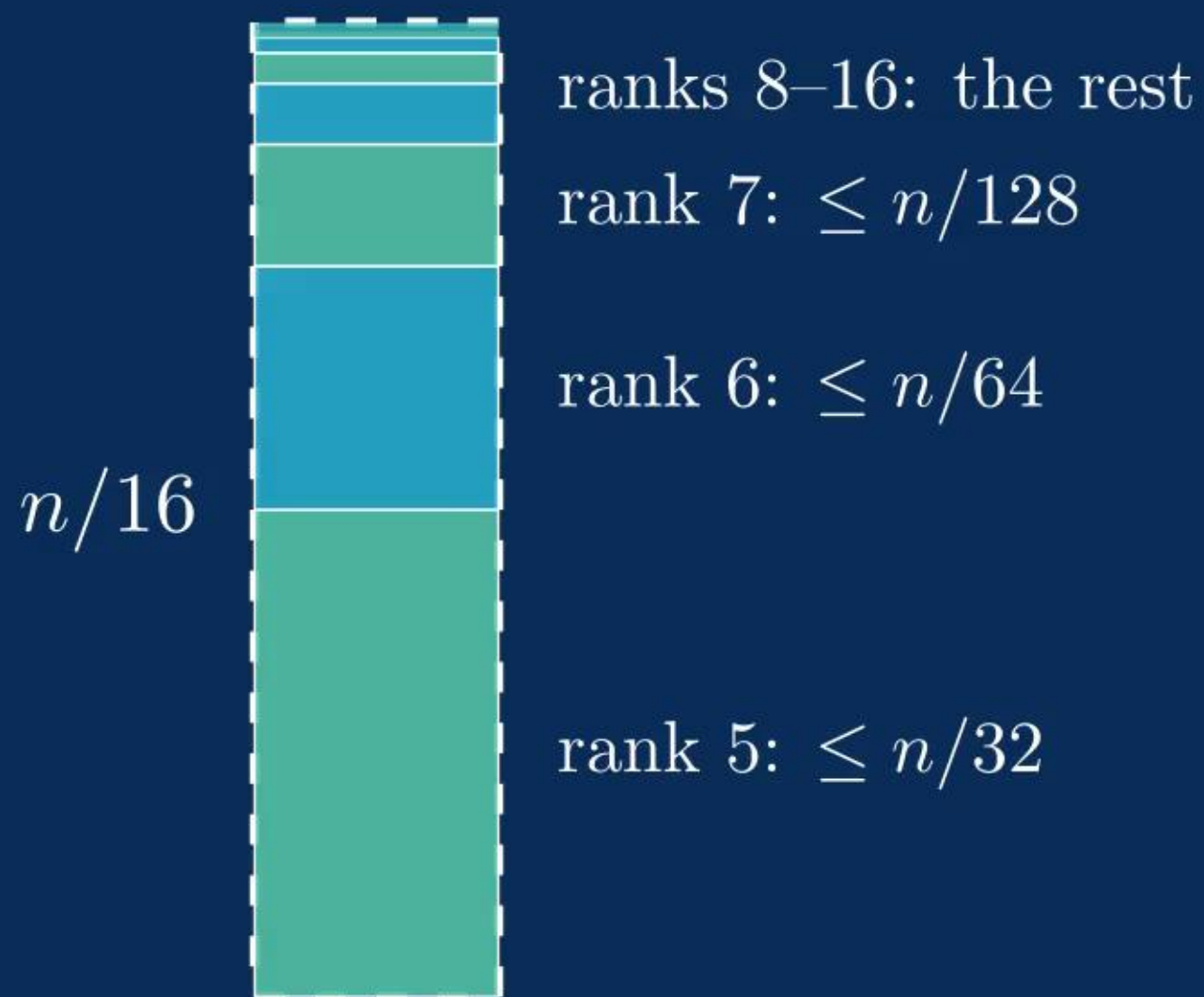
How many nodes?



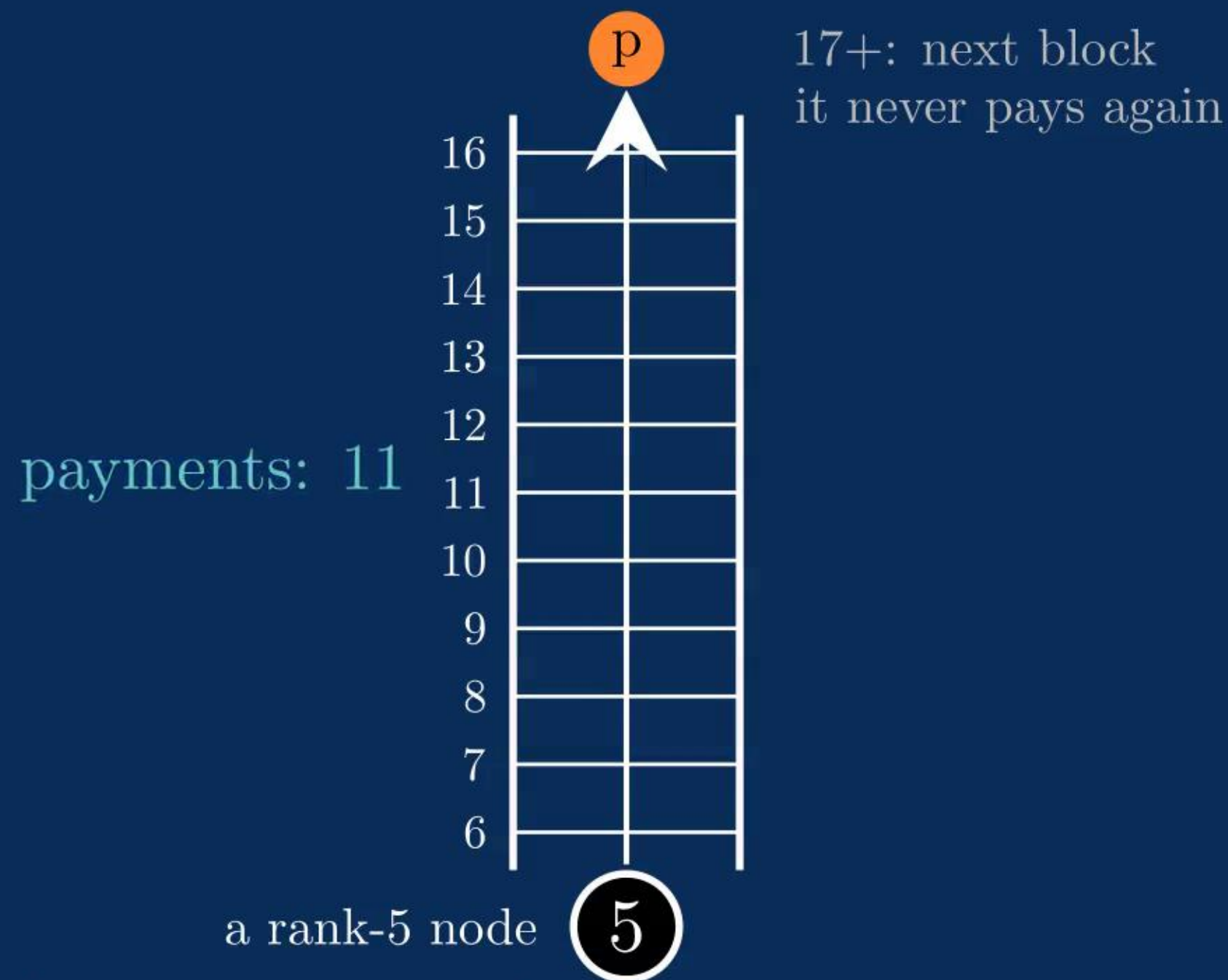
How many payments each?



How many nodes?

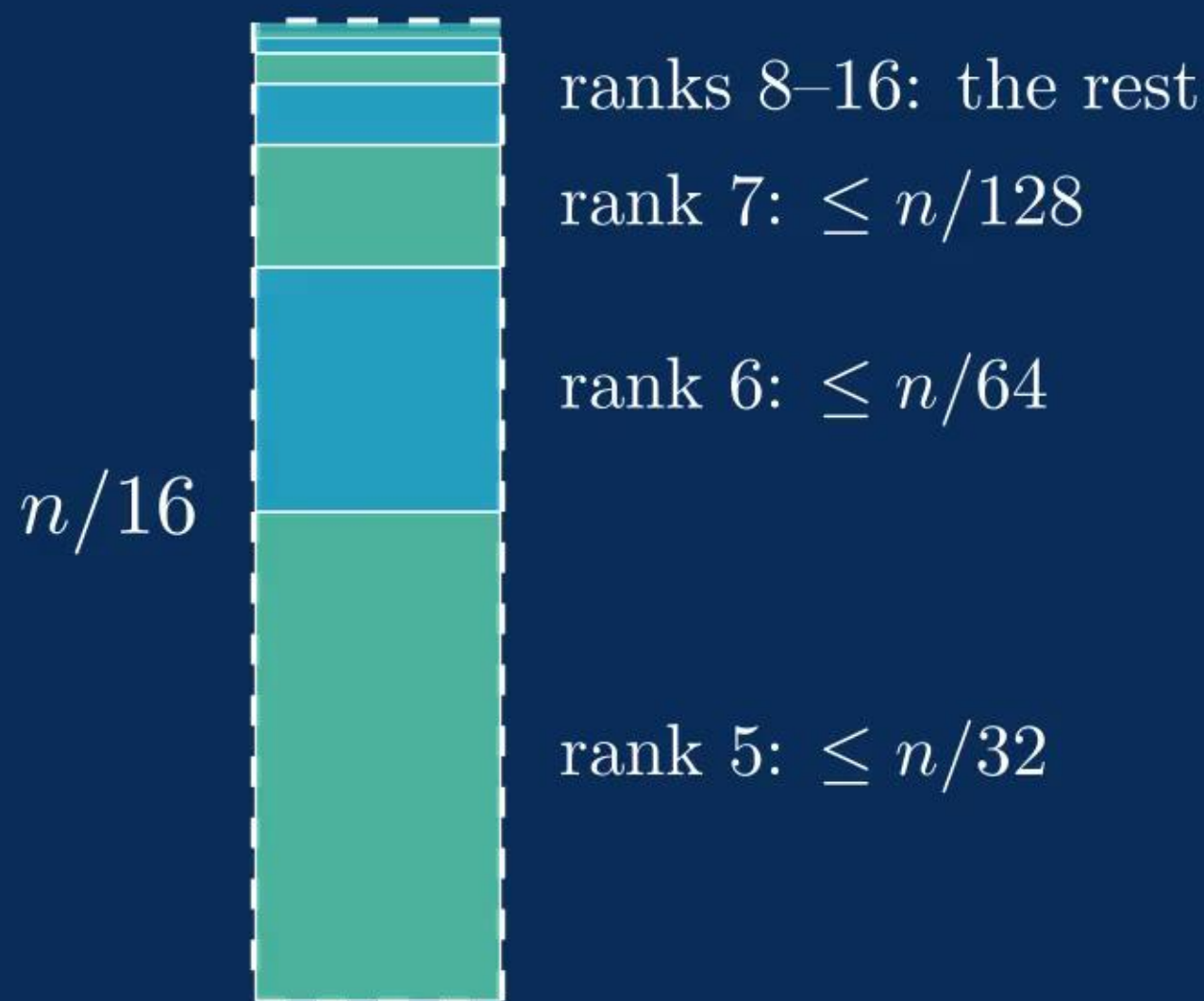


How many payments each?

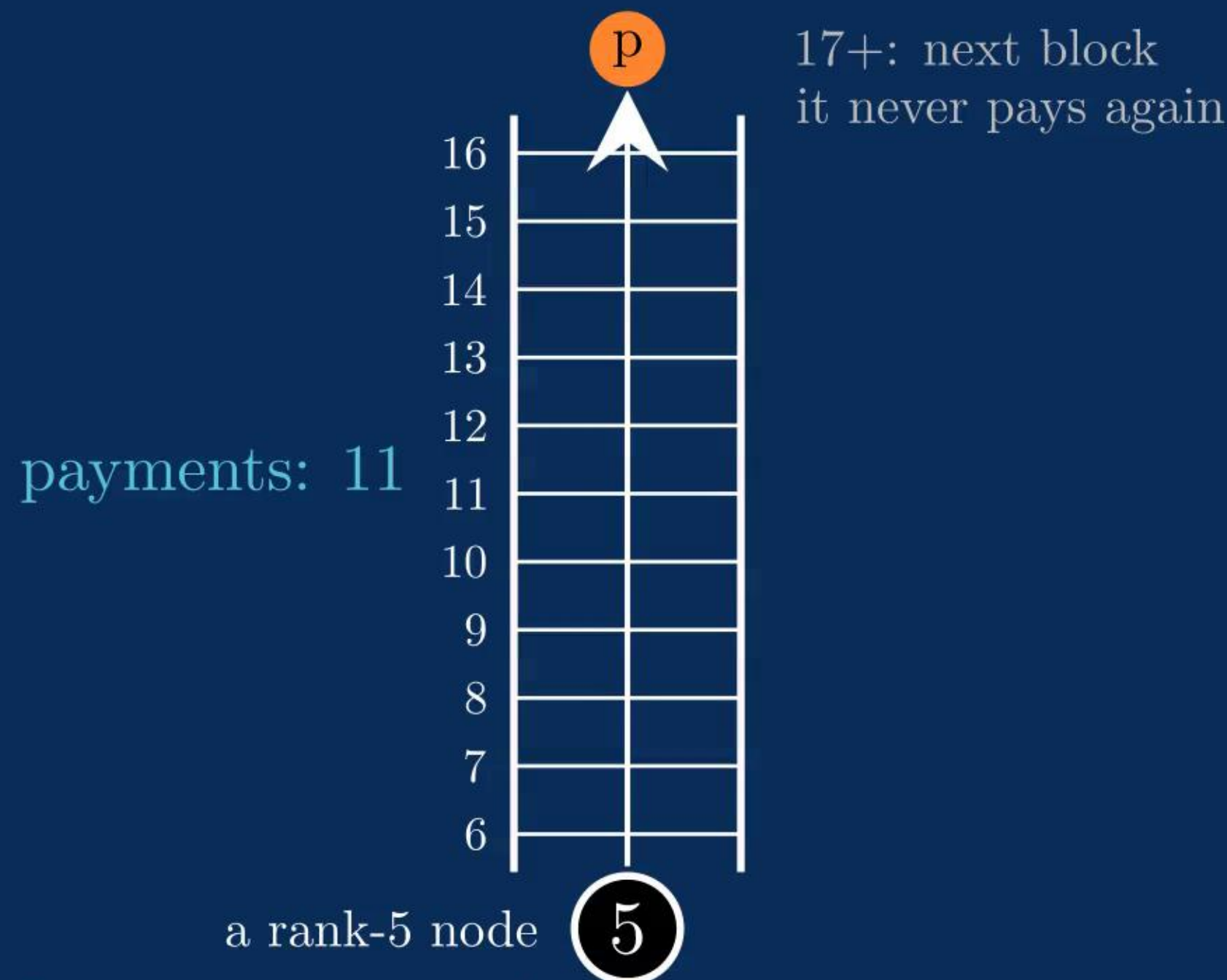


$< n/16$ nodes $\times \leq 16$ payments each

How many nodes?



How many payments each?



$$< n/16 \text{ nodes} \times \leq 16 \text{ payments each} < n$$

Finds pay: at most $\log^* n + 1$ per find (block boundaries, plus the root)

Finds pay: at most $\log^* n + 1$ per find (block boundaries, plus the root)

m finds: $\leq m(\log^* n + 1)$

Finds pay: at most $\log^* n + 1$ per find (block boundaries, plus the root)

m finds: $\leq m(\log^* n + 1)$

Nodes pay: less than n per block, over the *whole run*

{0}

{1}

{2}

{3, 4}

{5..16}

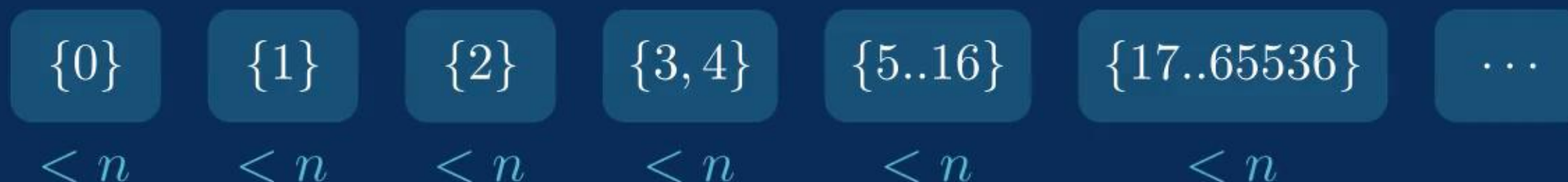
{17..65536}

...

Finds pay: at most $\log^* n + 1$ per find (block boundaries, plus the root)

m finds: $\leq m(\log^* n + 1)$

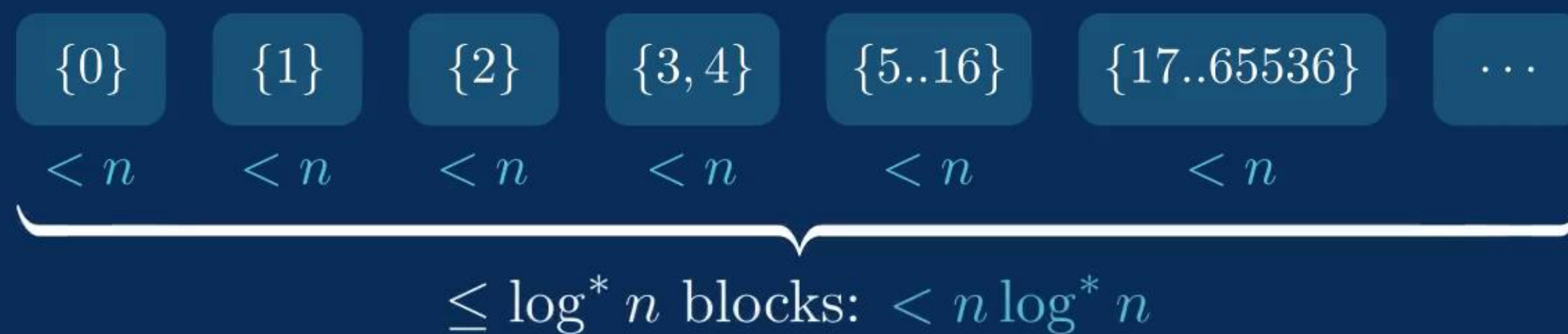
Nodes pay: less than n per block, over the *whole run*



Finds pay: at most $\log^* n + 1$ per find (block boundaries, plus the root)

m finds: $\leq m(\log^* n + 1)$

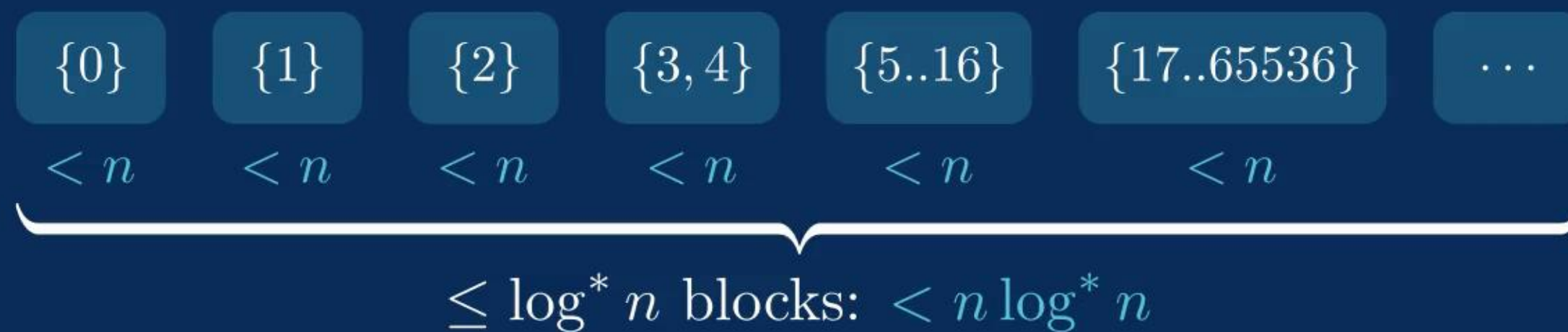
Nodes pay: less than n per block, over the *whole run*



Finds pay: at most $\log^* n + 1$ per find (block boundaries, plus the root)

m finds: $\leq m(\log^* n + 1)$

Nodes pay: less than n per block, over the *whole run*

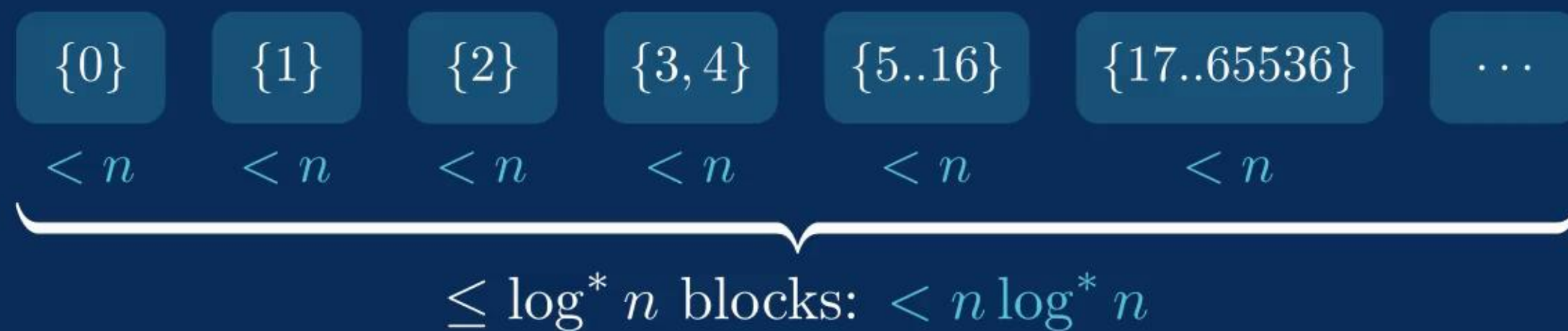


$$\text{Total} \leq m(\log^* n + 1) + n \log^* n$$

Finds pay: at most $\log^* n + 1$ per find (block boundaries, plus the root)

m finds: $\leq m(\log^* n + 1)$

Nodes pay: less than n per block, over the *whole run*

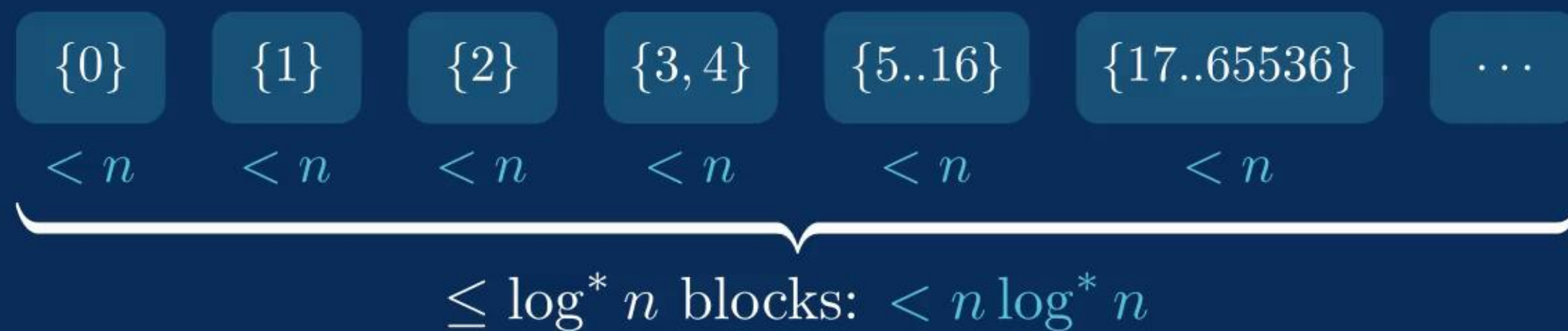


$$\text{Total} \leq m(\log^* n + 1) + n \log^* n = \mathcal{O}((m + n) \log^* n)$$

Finds pay: at most $\log^* n + 1$ per find (block boundaries, plus the root)

m finds: $\leq m(\log^* n + 1)$

Nodes pay: less than n per block, over the *whole run*



$$\text{Total} \leq m(\log^* n + 1) + n \log^* n = \mathcal{O}((m + n) \log^* n)$$

With $m \geq n$: $\mathcal{O}(\log^* n)$ amortized per operation

Next time: Graphs!