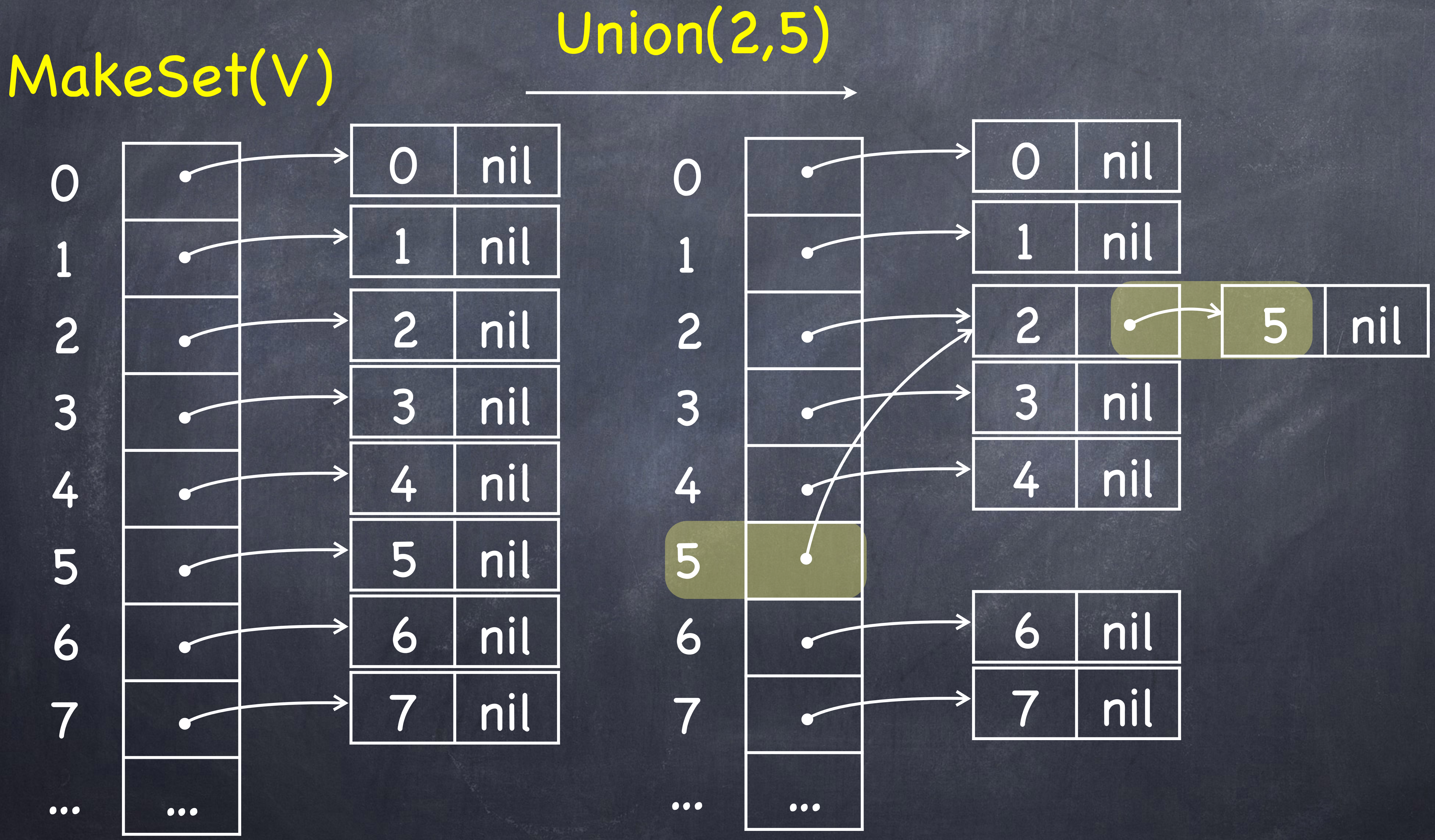
MST : Union-Find or Disjoint Sets Data Structure

- A Disjoint set data structure implements three functions:
  - MakeSet(x) : creates a set with one element x in it.
    MakeSet(V), creates n sets each with one element of V.
  - Find(x) : returns a set-canonical member for set-of(x).
  - Union(x,y) : merges the set containing x and the set containing y into one new set with one canonical element.

- Let V={1,2,3,4,5}
  MakeSet(V) = { {1}, {2}, {3}, {4}, {5} }
  Find(4) = 4
  Union(1,2) changes the sets to { {1,2}, {3}, {4}, {5} }
  Find(2) = 1.
  Union(2,5) changes the sets to { {1,2,5}, {3}, {4} }
  Find(5) = 1

Union-Find DS : Union-Find or Disjoint Sets 3

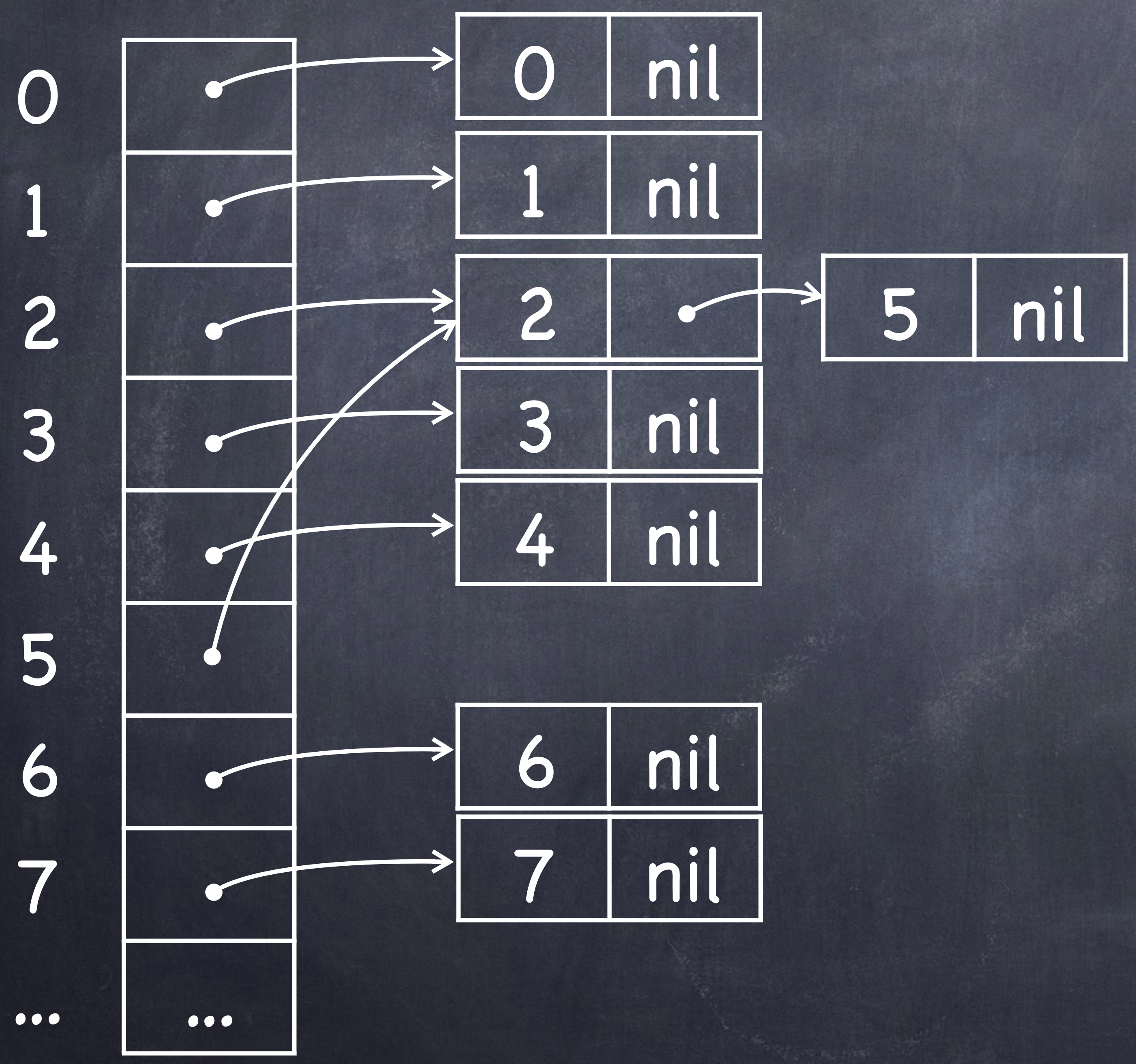
- A second try using array and linked list



Union-Find DS : Union-Find or Disjoint Sets 4

- A second try using array and linked list

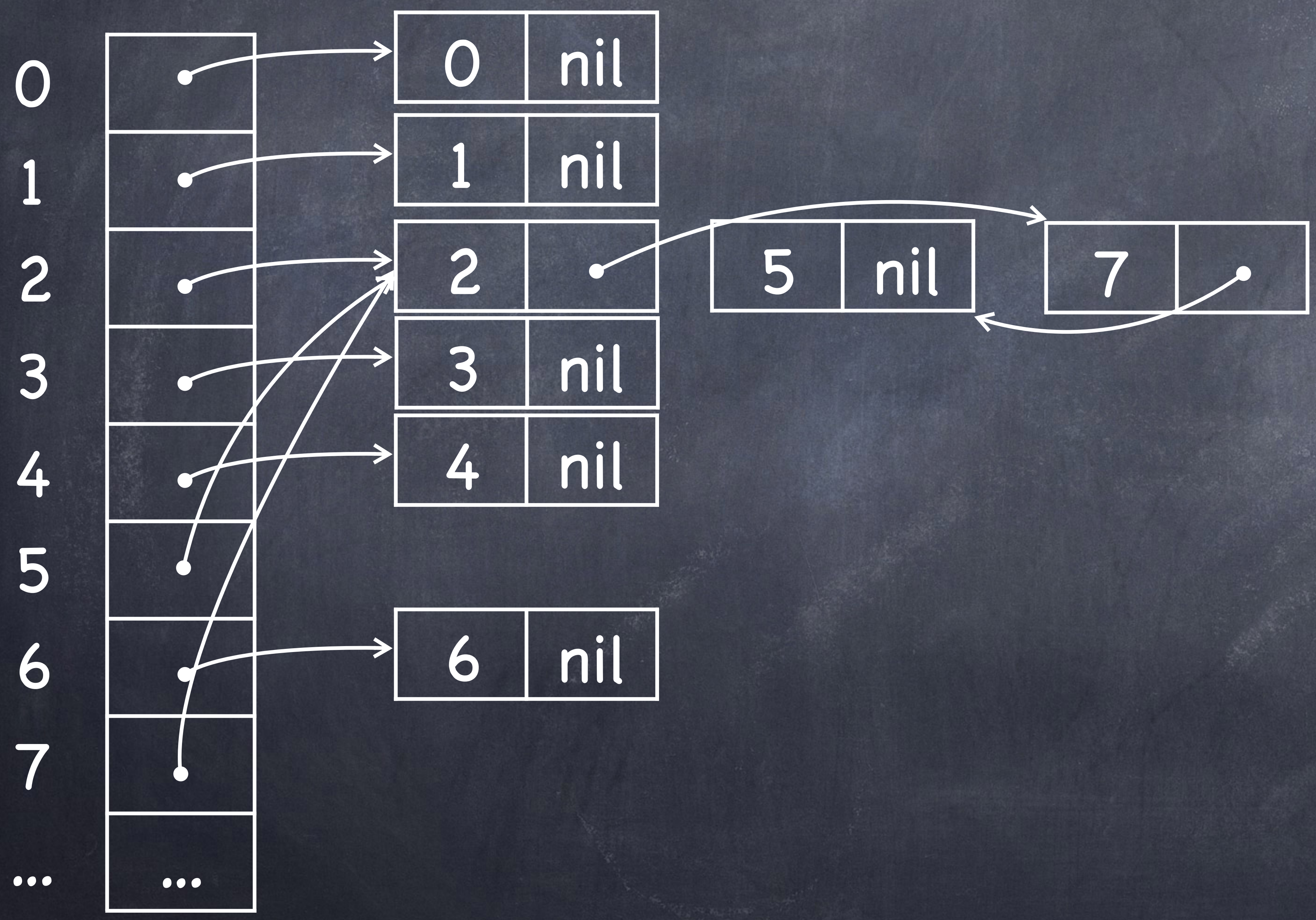
Union(5,7)



Union-Find DS : Union-Find or Disjoint Sets 4

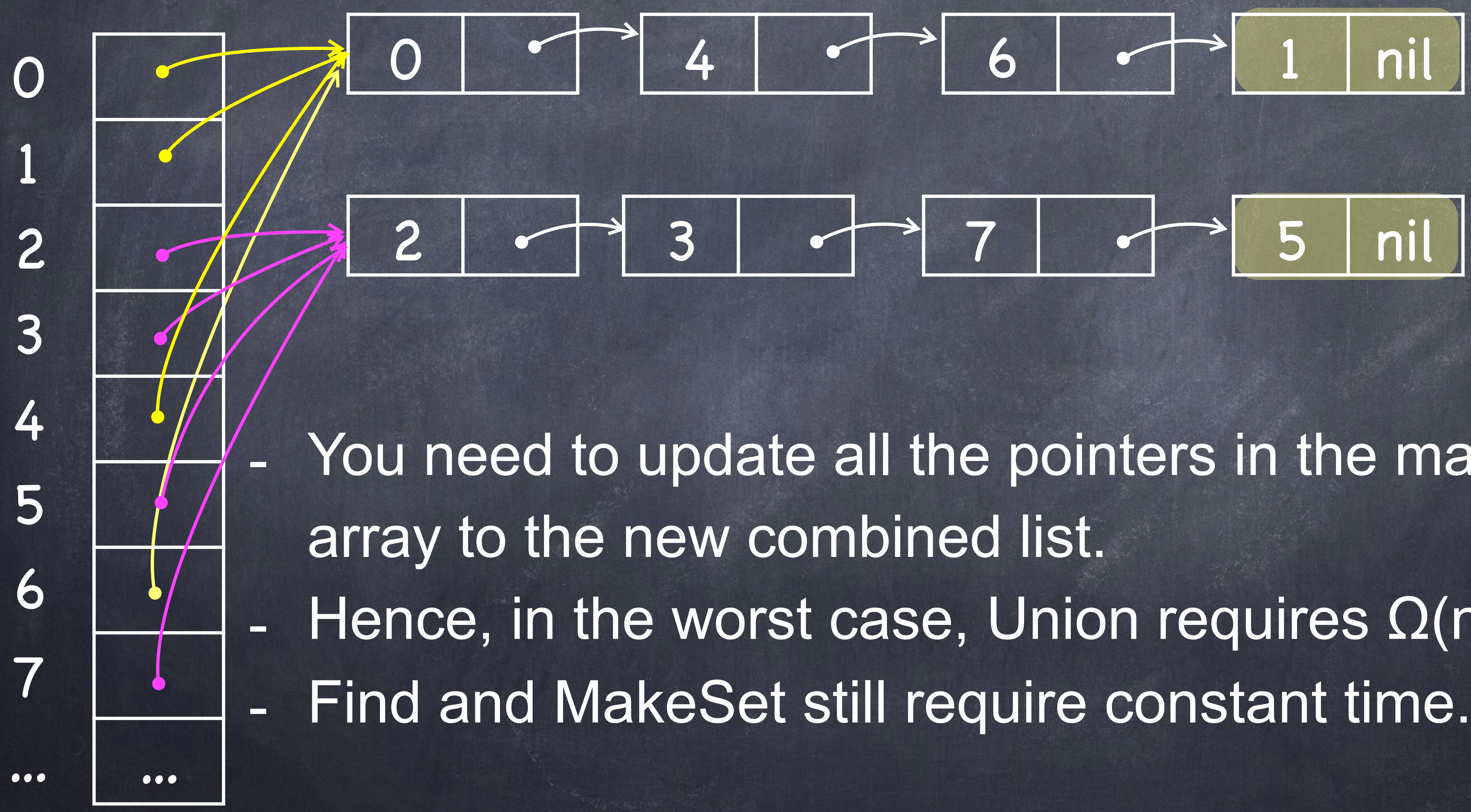
- A second try using array and linked list

Union(5,7)



Union-Find DS : Union-Find or Disjoint Sets 5

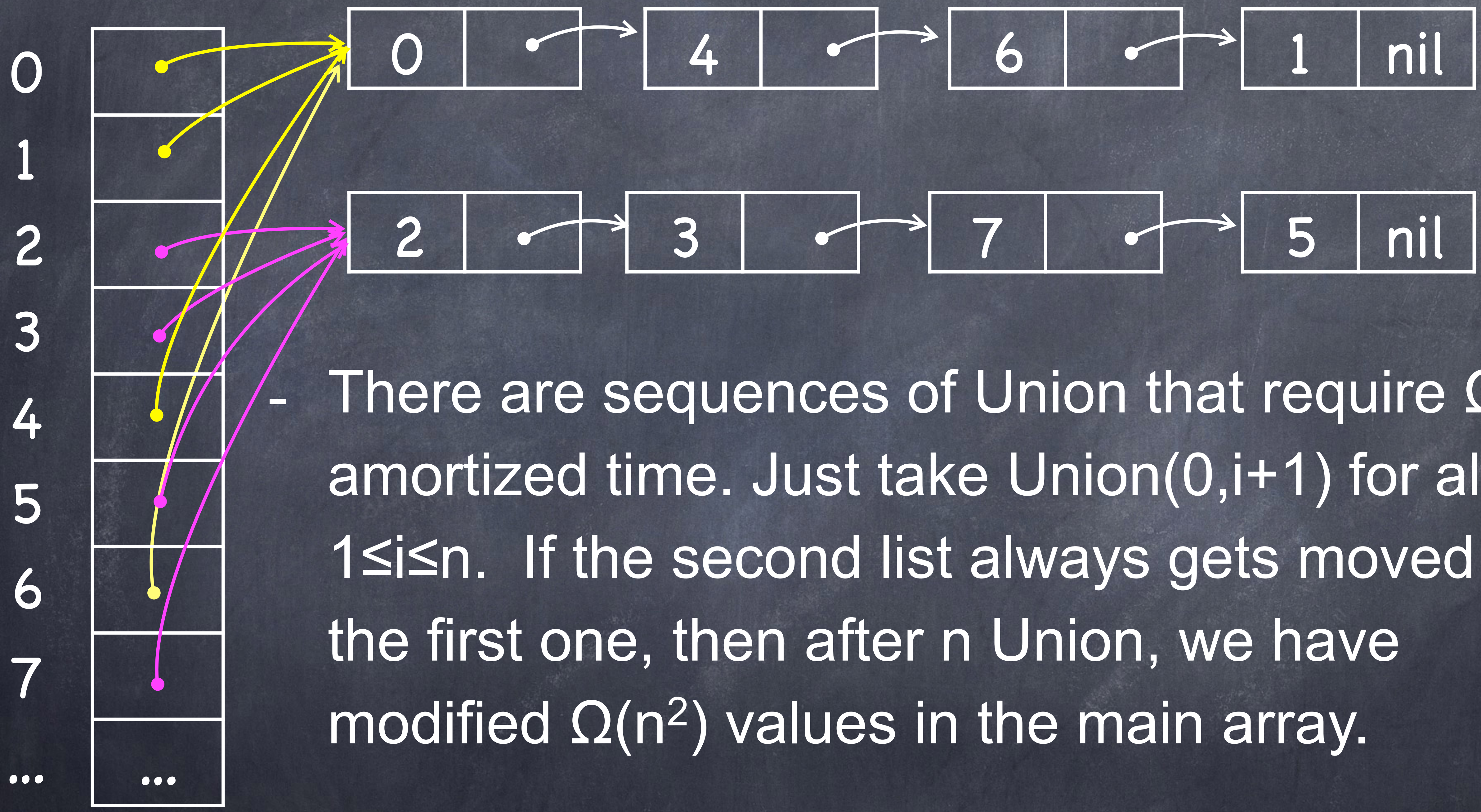
- A second try using array and linked list
- How long does it take to do Union(5,1) ?



- You need to update all the pointers in the main array to the new combined list.
- Hence, in the worst case, Union requires Ω(n) time
- Find and MakeSet still require constant time.

Union-Find DS : Union-Find or Disjoint Sets 6

- A second try using array and linked list



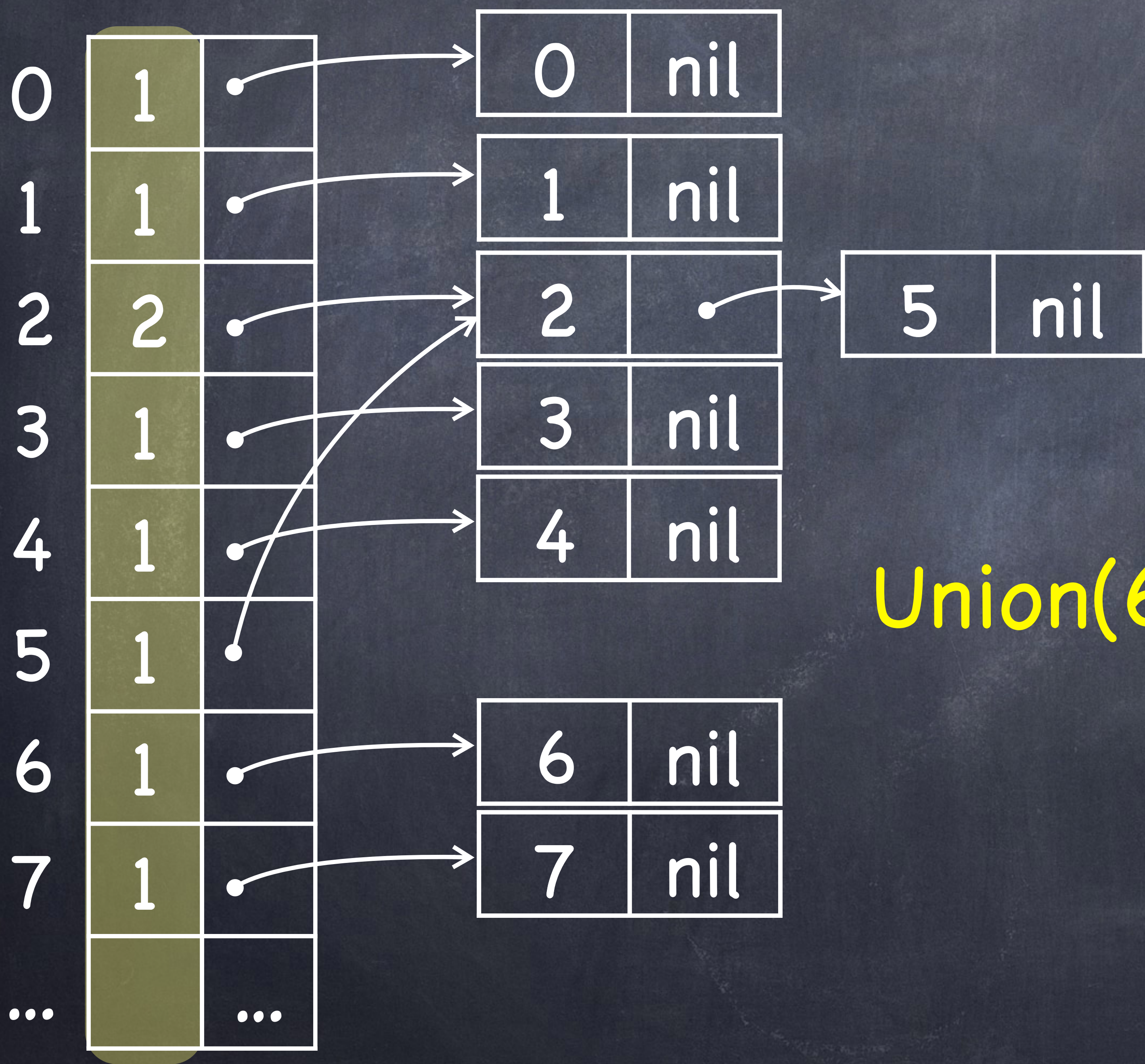
- There are sequences of Union that require $\Omega(n^2)$ amortized time. Just take Union(0,i+1) for all $1 \le i \le n$. If the second list always gets moved into the first one, then after n Union, we have modified $\Omega(n^2)$ values in the main array.

- But there is a heuristic that we can use to counter this bad side effect. It is called Union-by-rank.

Union-Find DS : Union-by-rank

- A second try using array and linked list

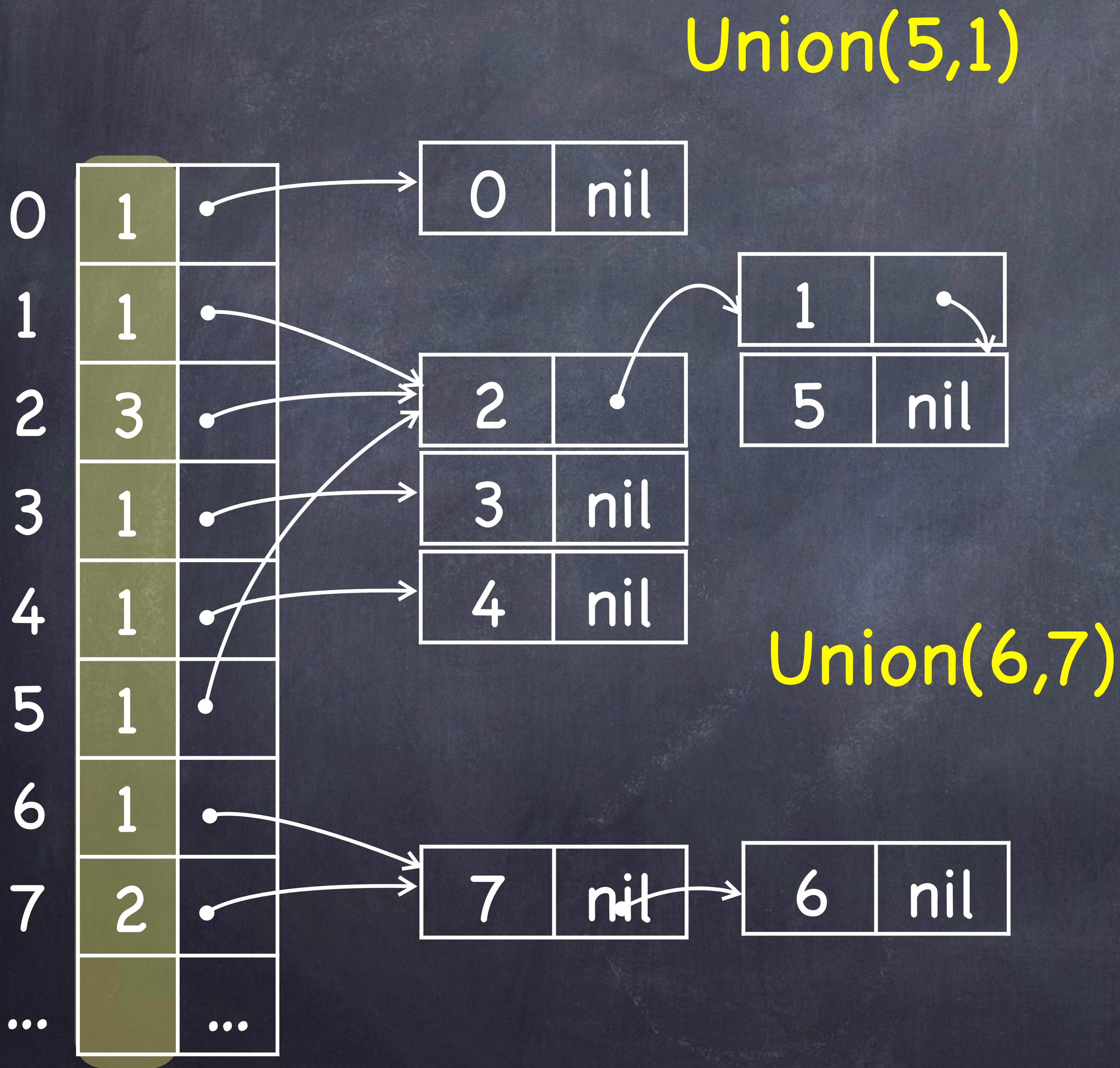
Union(5,1)



Union(6,7)

Union-Find DS : Union-by-rank

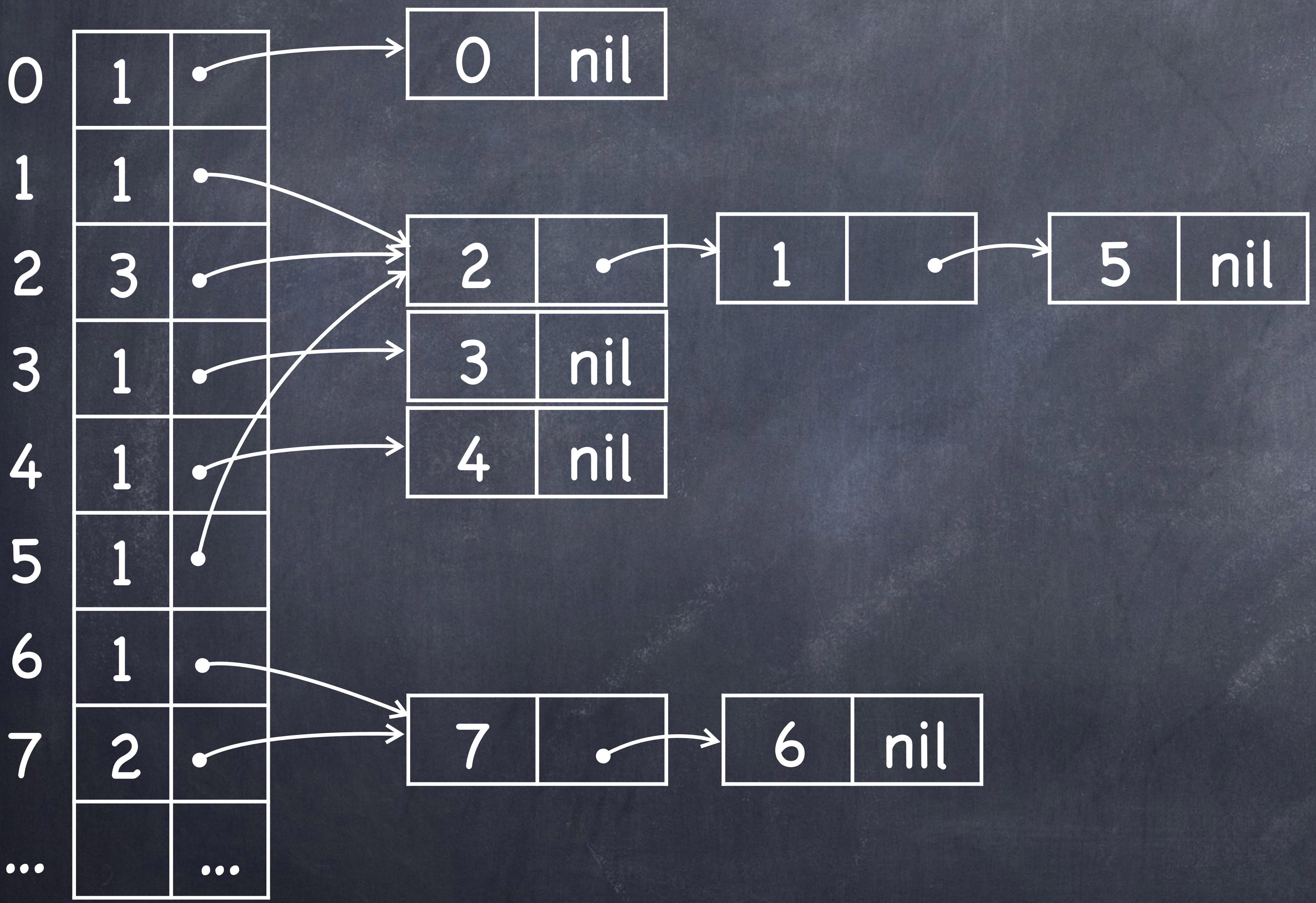
- A second try using array and linked list



Union-Find DS : Union-by-rank 2

- A second try using array and linked list

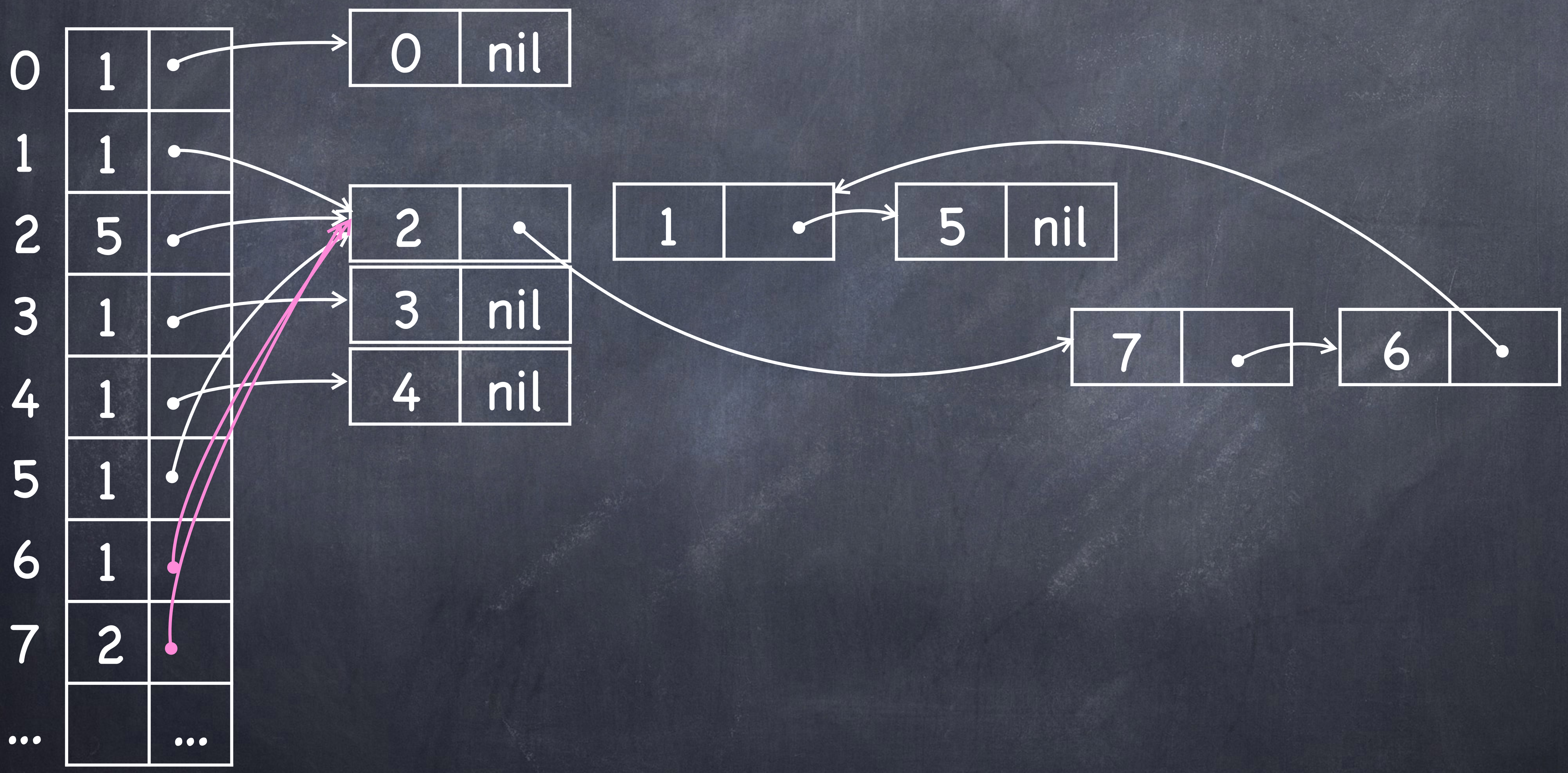
Union(5,6)



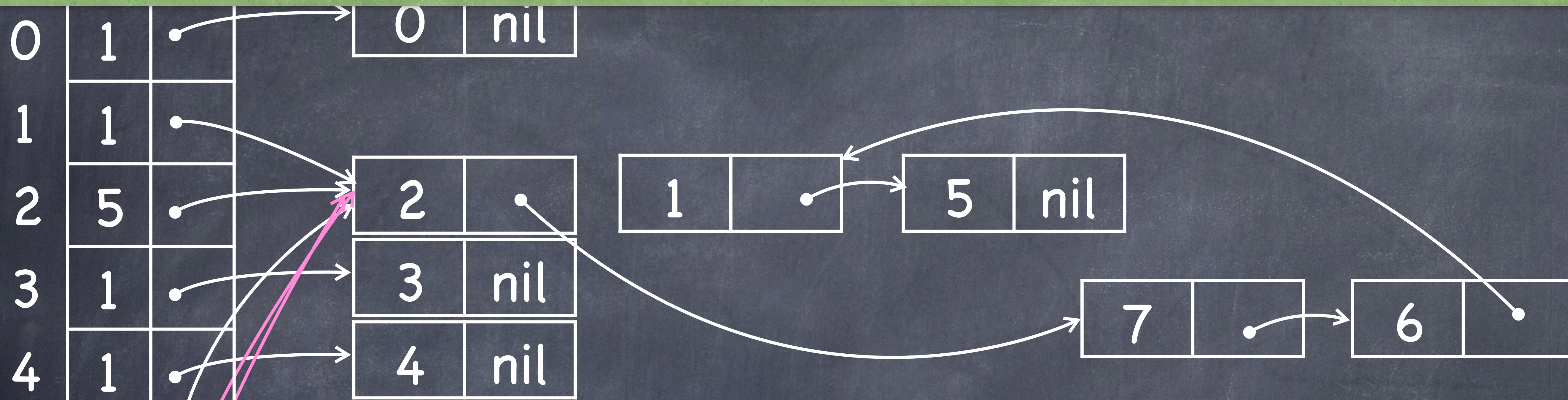
Union-Find DS : Union-by-rank 2

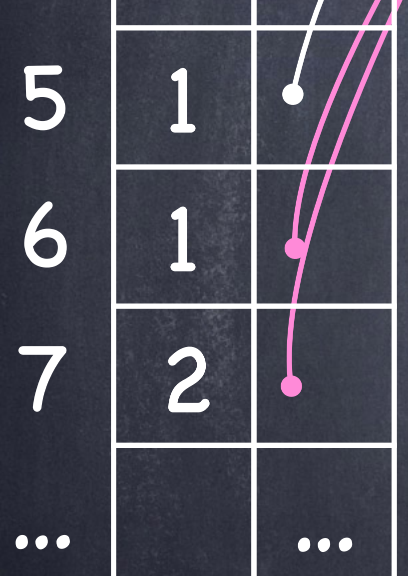
- A second try using array and linked list

Union(5,6)



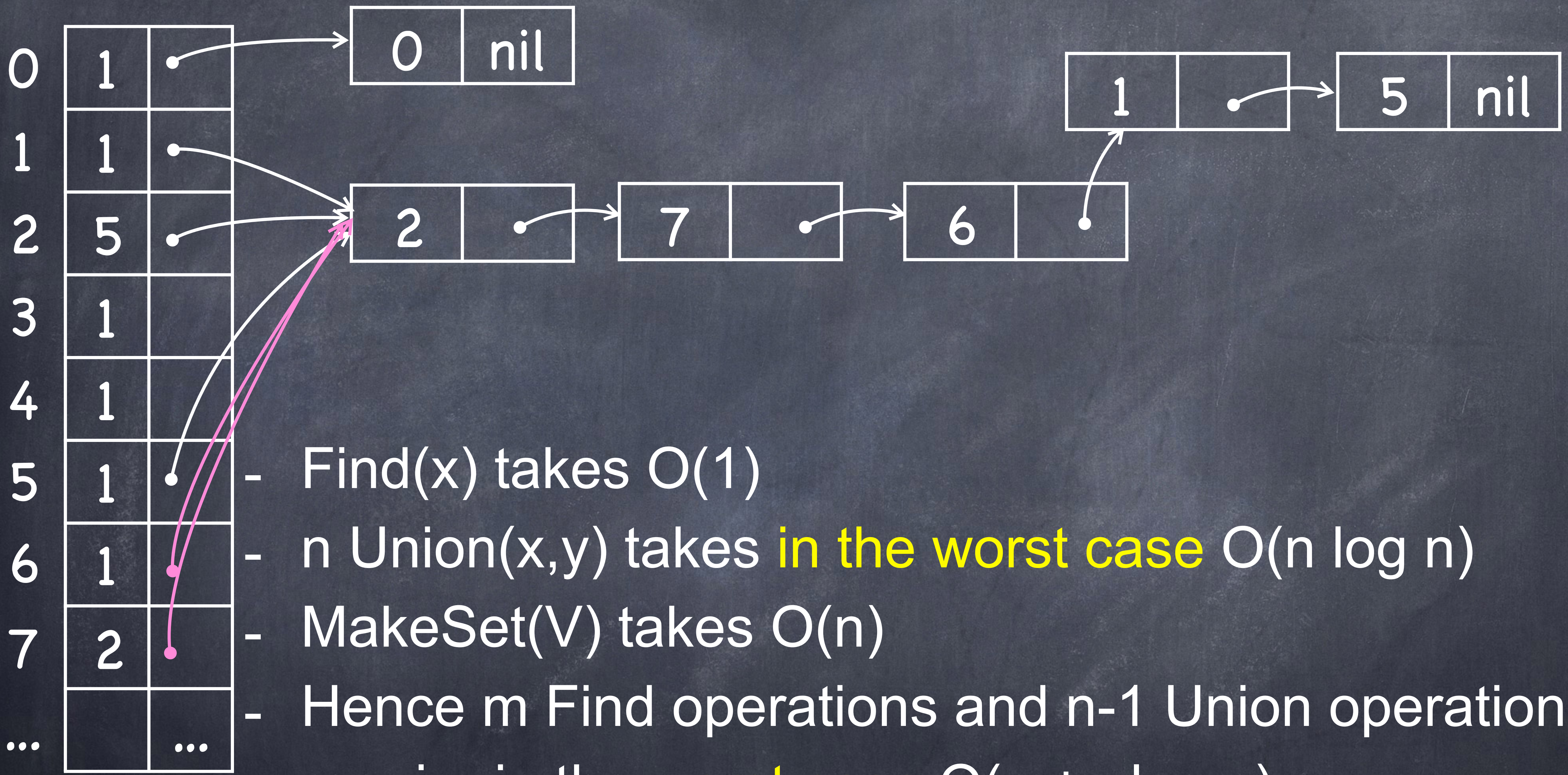
Union-Find DS : Union-by-rank 3





- Each time a list is added at the back of another list, its size doubles (the smallest).
- How many time can the size of list double if the maximum size it can achieve is n ?
- log(n) (no element can be updated more than)
- Yes, in the final step we work $\Omega(n)$, but we can work for only log(n) times.
- Hence the total work done to do n-1 Union and m Find is $O(m+n \log(n))$
- A definite improvement over $O(n^2)$.

Union-Find DS : Union-by-rank 4

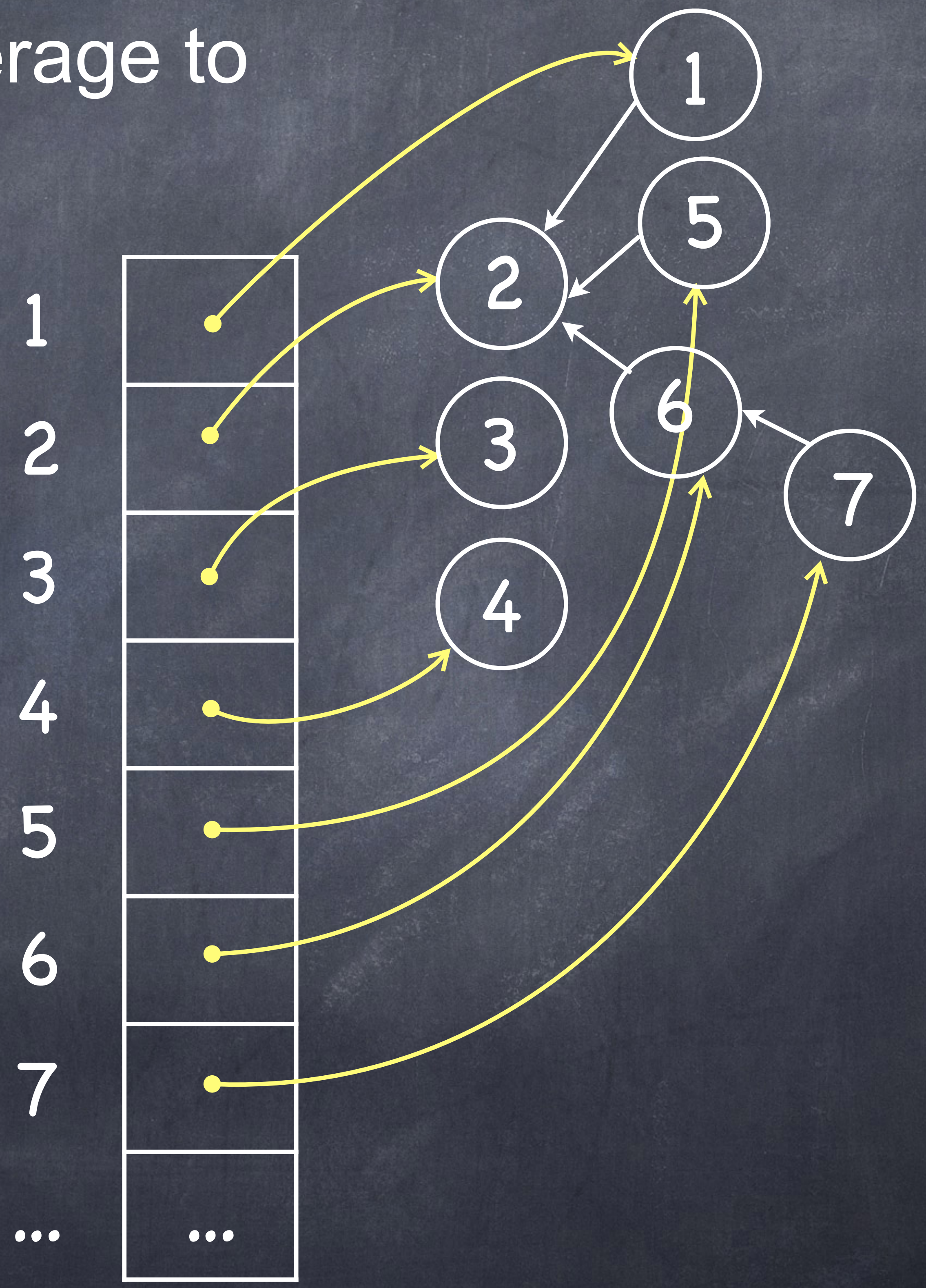


- Find(x) takes O(1)
- n Union(x,y) takes in the worst case O(n log n)
- MakeSet(V) takes O(n)
- Hence m Find operations and n-1 Union operations require in the worst case O(m+n log n)
- But we can still do better : Almost linear in the worst case for m Find operations and n-1 Unions

Union-Find DS : Path compression 3

- How long does it take on average to do a Find(x) operation ?
- Union(x,y) operation ?
- Makeset(V) ?

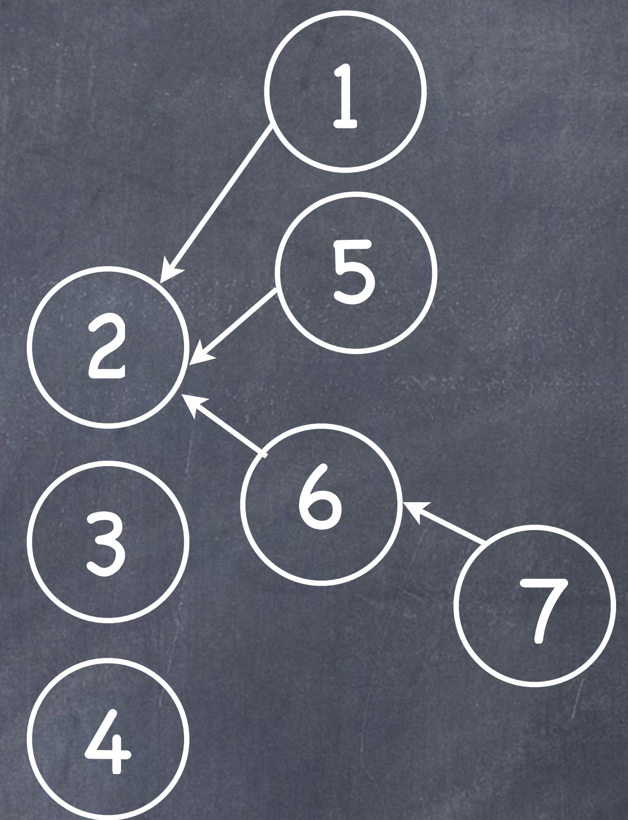
- We can use Union-by-rank.
- Just keep a pointer in the array or in each cell.



Union-Find DS : Path compression 4

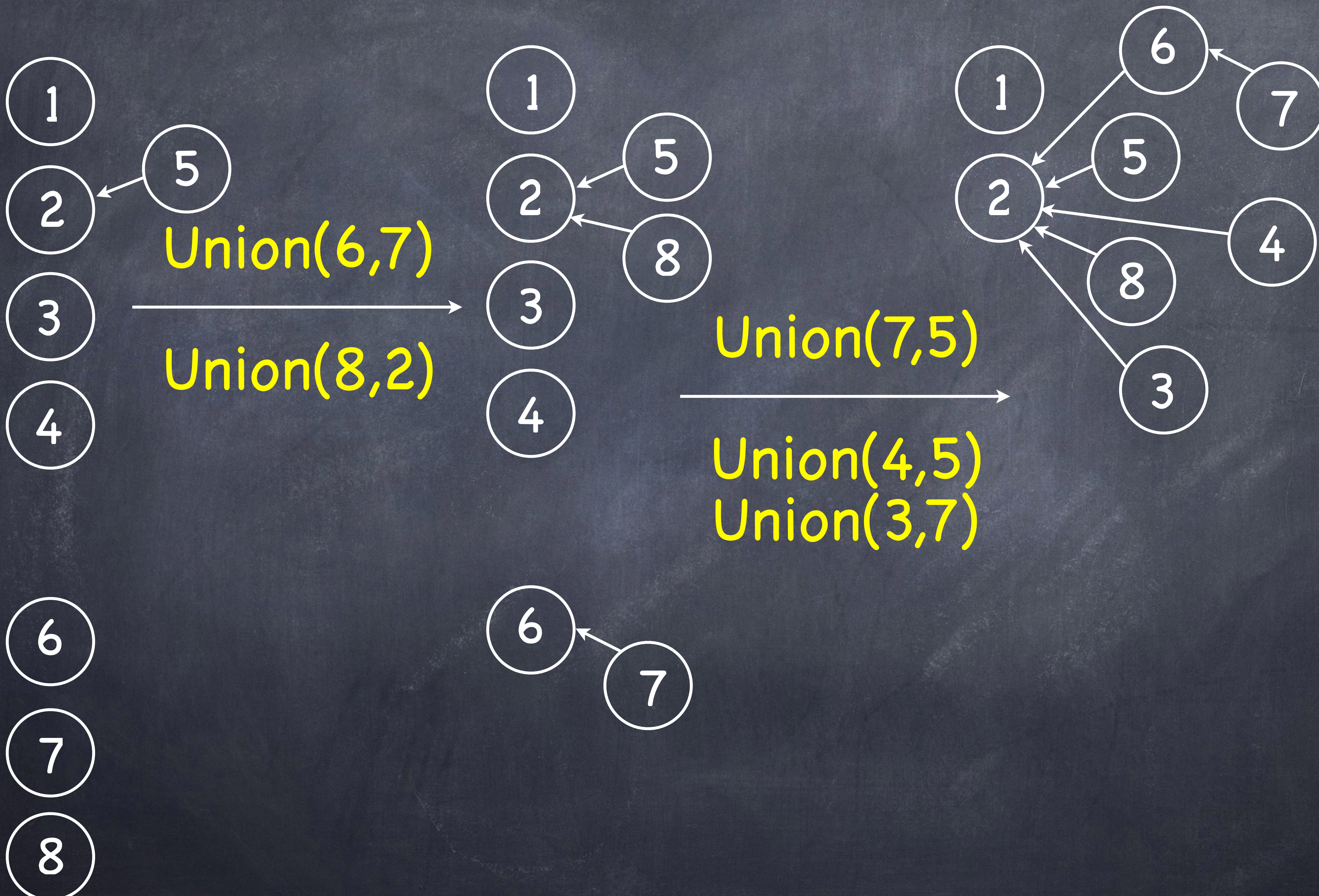
```
Find(x)::
while x ≠ prev(x)
 x=prev(x)
return x
```

```
Union(x,y)::
rx = Find(x)
ry = Find(y)
if rx==ry
 return
if rank(rx)>rank(ry)
 prev(ry)=rx
else
 prev(rx)=ry
 if rank(rx)==rank(ry)
  rank(ry)=rank(ry)+1
```



Here rank means depth of the tree rooted at rx

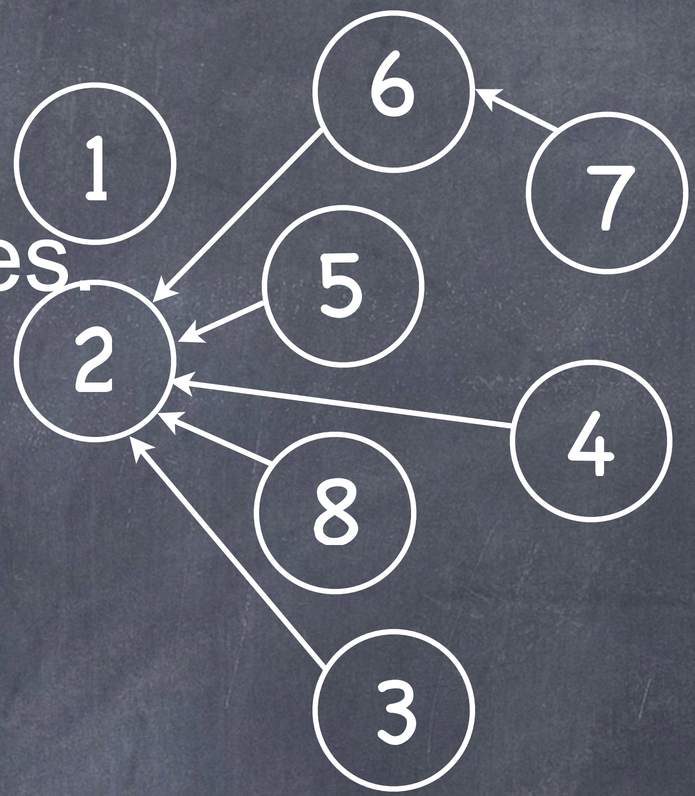
Union-Find DS : Path compression 5



Union-Find DS : Path compression 6

A few observations:

- rank(x)<rank(prev(x))
- Any sub-tree of rank k has at least 2^k nodes.



Pf: At the beginning all nodes have rank 0. $2^0=1$.

When does the rank change ?

When we unionize equal rank trees.

Hence if both trees had at least 2^k nodes, then the new tree has $2^k+2^k=2^{k+1}$ nodes and rank k+1.

Finally, over n nodes, only $n/2^k$ nodes can have rank k.

From this we conclude that the maximum depth of a tree is log(n) since $n/2^{\log n} = 1$.

Union-Find DS : Path compression 7

Since the maximum dept of a tree is at most log n, we can conclude that:

Find(x) takes at most O(log n) time,

n Union(x,y) takes at most O(n log n) time,

Makeset(V) is still linear.

But Find(x) for our linked list implementation used O(1) and we are not quicker for Union and Makeset.

What happened ?

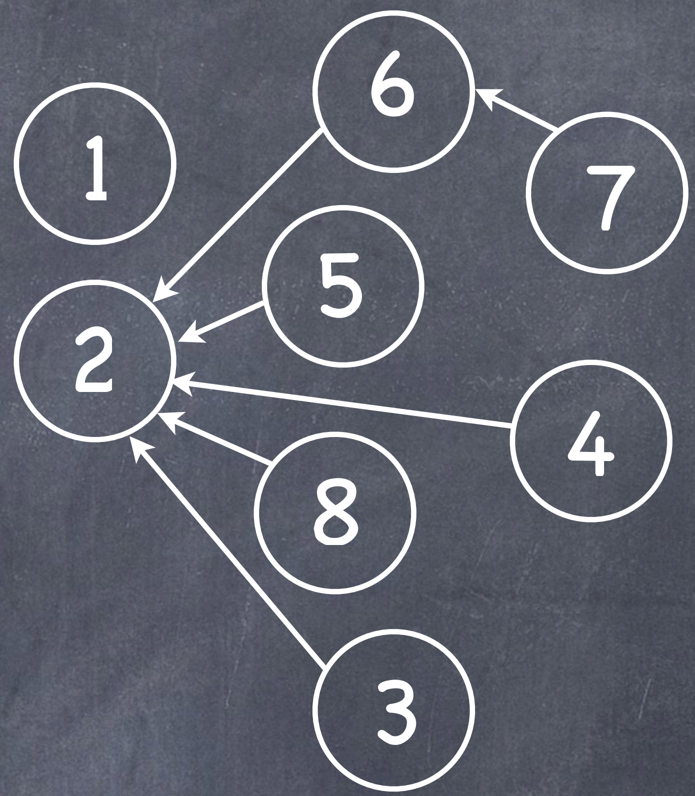
Well, we are not done.

This data structure needs another heuristic in order to be competitive.

Union-Find DS : Path compression 8

```
Find(x)::
if x ≠ prev(x) then
 prev(x) = Find(prev(x))
return prev(x)
```

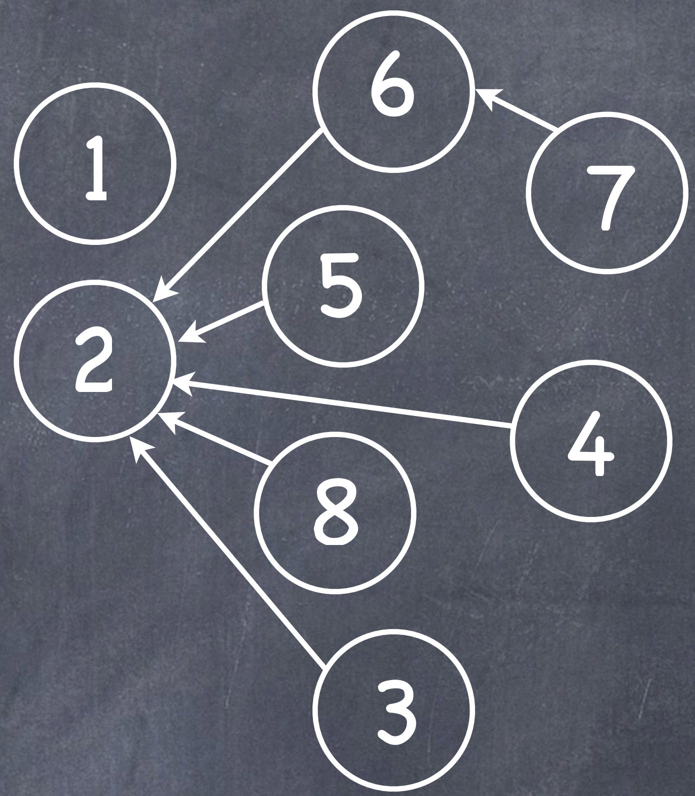
```
Find(x)::
r = x
while r ≠ prev(r)
  r = prev(r)
while x ≠ prev(x)
  x' = prev(x)
  prev(x) = r
  x = x'
return x
```



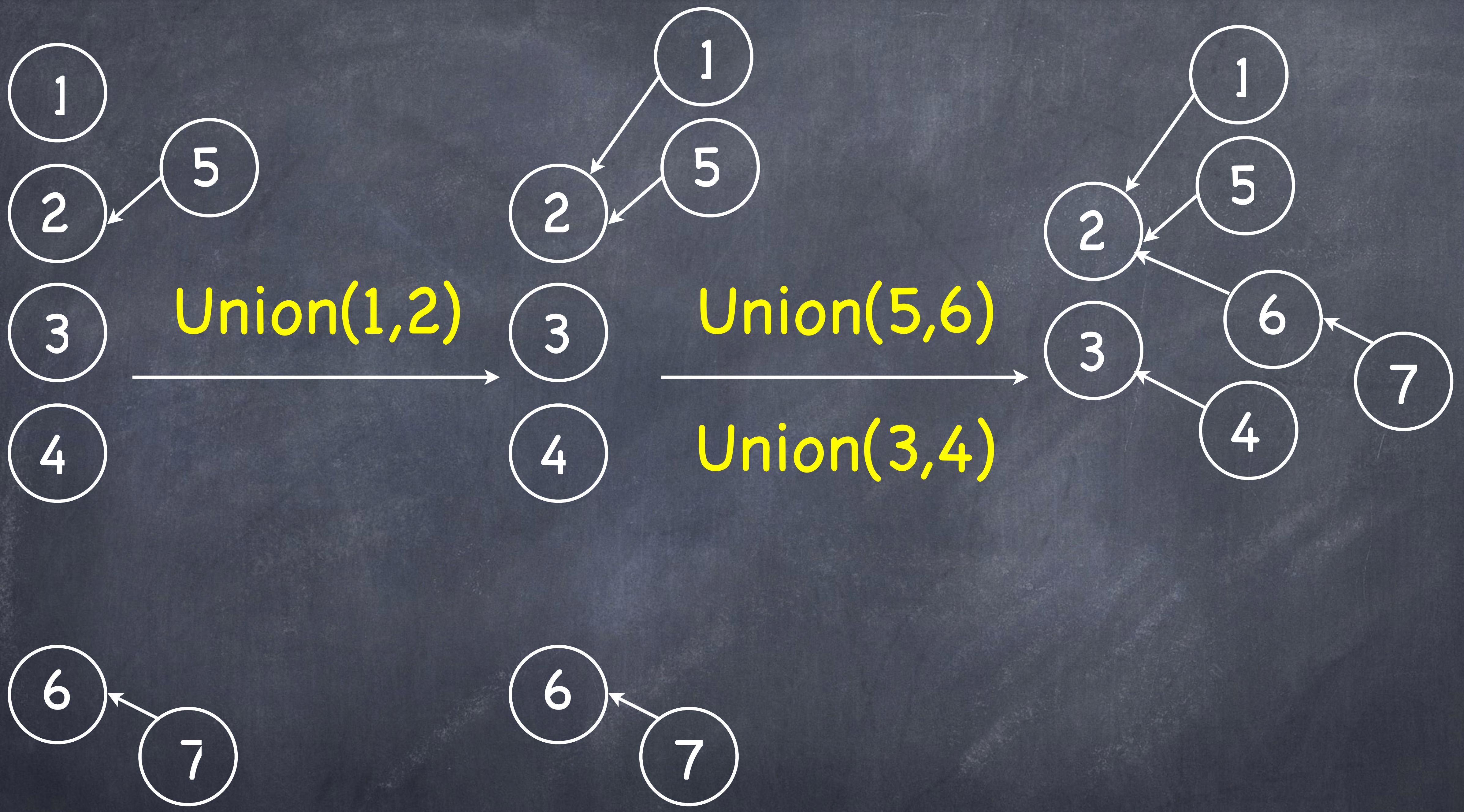
Union-Find DS : Path compression 8

```
Find(x)::
if x ≠ prev(x) then
 prev(x) = Find(prev(x))
return prev(x)
```

```
Union(x,y)::
rx = Find(x)
ry = Find(y)
if rx==ry
  return
if rank(rx)>rank(ry)
  prev(ry)=rx
else
  prev(rx)=ry
  if rank(rx)=rank(ry)
    rank(ry)=rank(ry)+1
```

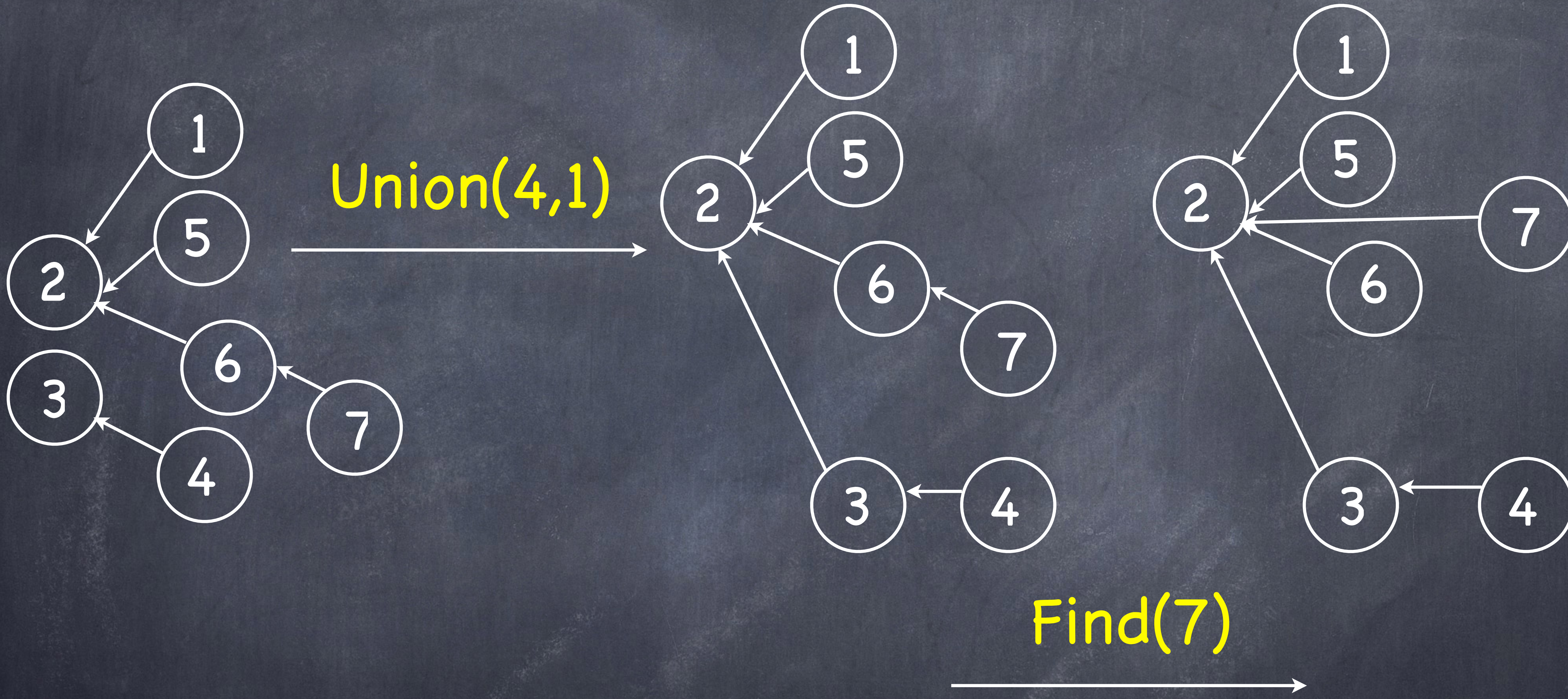


Union-Find DS : Path compression 10

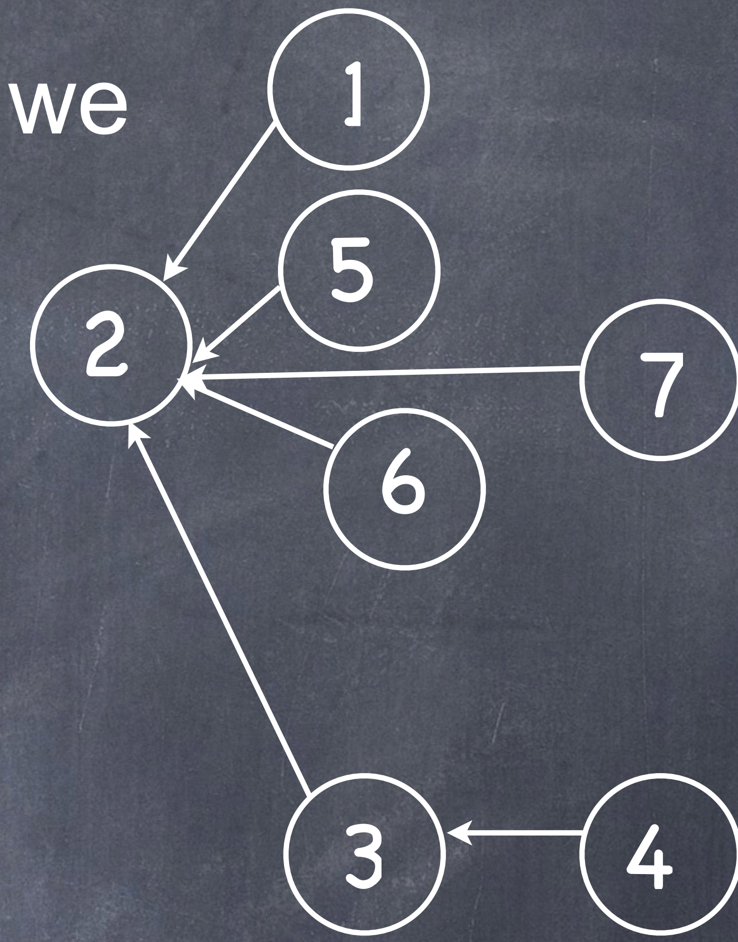


2 = Find(5) = Find(6)   2 = Find(7) ≠ Find(4) = 3

Union-Find DS : Path compression 11



Union-Find DS : Path compression 12



- Intuitively, the tree cannot get very deep as we constantly make it flatter.
- Makeset(V) takes O(n).
- In fact, any sequence of m operations (Makeset, Union and Find) n of which are Makeset(x) operations require

  $\Theta(m \cdot \alpha(m,n))$ operations.

- The function $\alpha(m,n)$ is the extremely slowly growing inverse of Ackermann's function. For all practical (and un-practical) problem instances, it is at most 4.
- So in Practice, it requires $\Theta(4m)$ time for m operations, n of which are Union.

In fact, $\alpha(n)$ is less than 5 for any practical input size n, since A(4, 4) is on the order of $2^{2^{2^{2^{16}}}}$.

Union-Find DS : Kruskal's algorithm

Input: $G=(V,E,cost(E))$
output: T

n Makeset + 2m Find + n-1 Union
2n+2m operations, n of which are Makeset.

```
for all u in V
  Makeset(u)

T={}
sort the edges E by cost(E)
for each edges (u,v) in (sorted) E
    if Find(u)≠Find(v)
        T=T+(u,v)
        Union(u,v)
        if(size(T)=|V|-1)
          done
```

n Makeset

O(1)

m log n

2m Find

n-1 Union

m O(1)

Hence, our data structure operations will require $\Theta((m+n)\cdot\alpha((m+n),n))$ or roughly $\Theta(m+n)$.