

NPTEL MOOC, JAN-FEB 2015  
Week 5, Module 2

# **DESIGN AND ANALYSIS OF ALGORITHMS**

**Union-Find data structure using pointers**

**MADHAVAN MUKUND, CHENNAI MATHEMATICAL INSTITUTE**  
**<http://www.cmi.ac.in/~madhavan>**



# Union-Find data structure

- \* A set of elements  $S$  **partitioned** into subsets, or **components**,  $\{C_1, C_2, \dots, C_k\}$ 
  - \* Each  $s$  in  $S$  belongs to exactly one  $C_j$
- \* Support the following operations
  - \* **MakeUnionFind( $S$ )** — set up initial components, each  $s$  in  $S$  is a separate singleton component  $\{s\}$
  - \* **Find( $s$ )** — returns the component containing  $s$
  - \* **Union( $C, C'$ )** — merges the components  $C$  and  $C'$



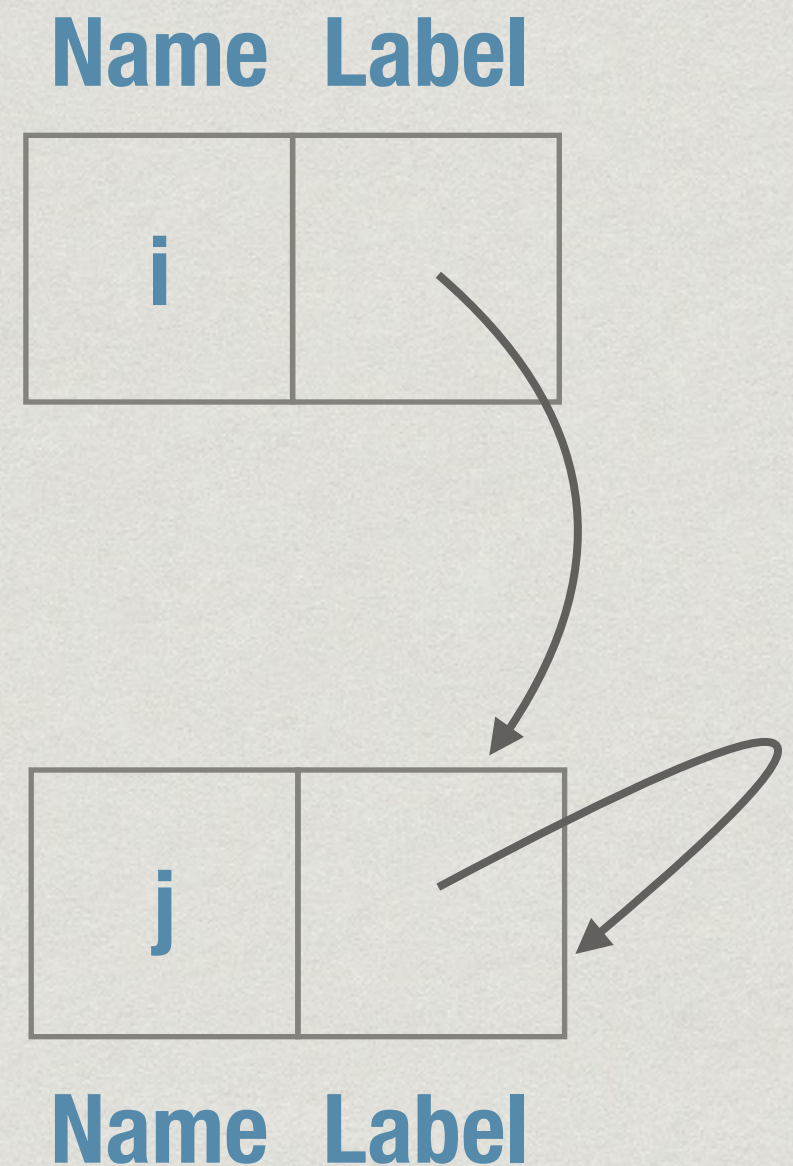
# Implement with arrays/lists

- \* Implement Union-Find using array **Component[1..n]**, lists **Member[1..n]** and array **Size[1..n]**
- \* **MakeUnionFind(S)** is  $O(n)$
- \* **Find(s)** is  $O(1)$
- \* Amortized complexity of each **Union(k,k')** is  $O(\log m)$  over a sequence of  $m$  operations



# Implementing with pointers

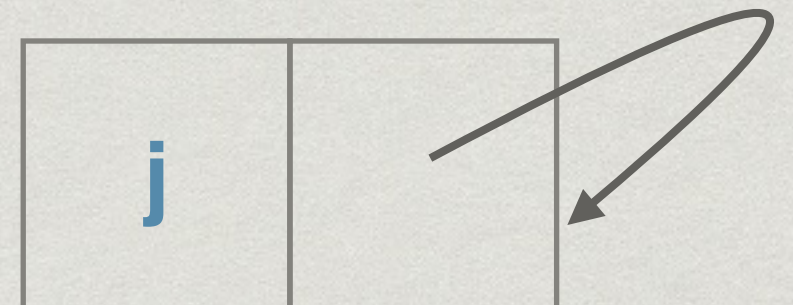
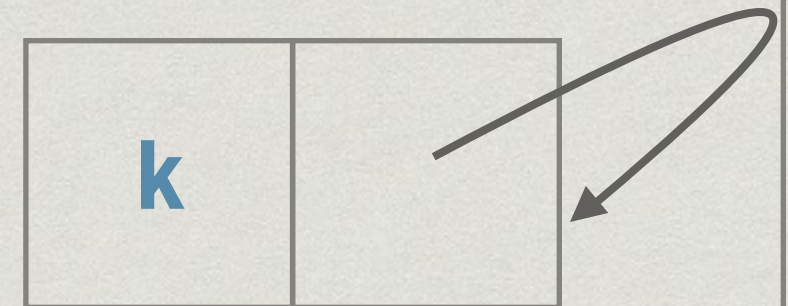
- \* Each element of the set is a node with two fields
  - \* Name: the name of the element
  - \* Label: pointer to the set containing the element
- \* Recall that we use same set S for component labels





# Pointers ...

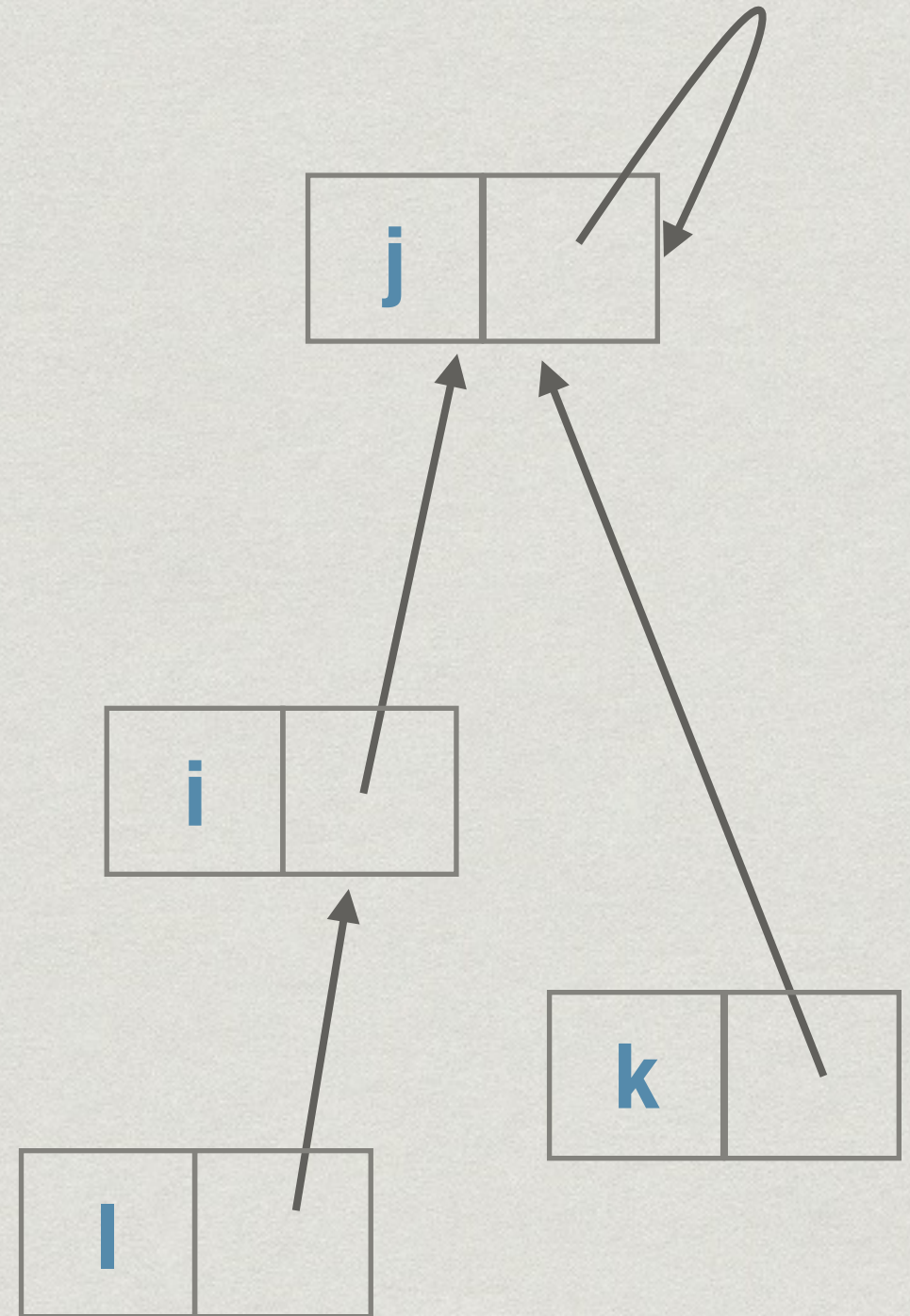
- \* Initially, each node points to itself
- \* Recall that we use same set  $S$  for component labels
- \* Initially, each element  $s$  is in component  $s$





# Pointers ...

- \* A component is a tree
- \* Root element names the component, points to itself
- \* For other elements, follow path to root to find the name of the component





# Auxiliary structures

- \* Node[1..n]

- \* Node[k] points to node containing element k

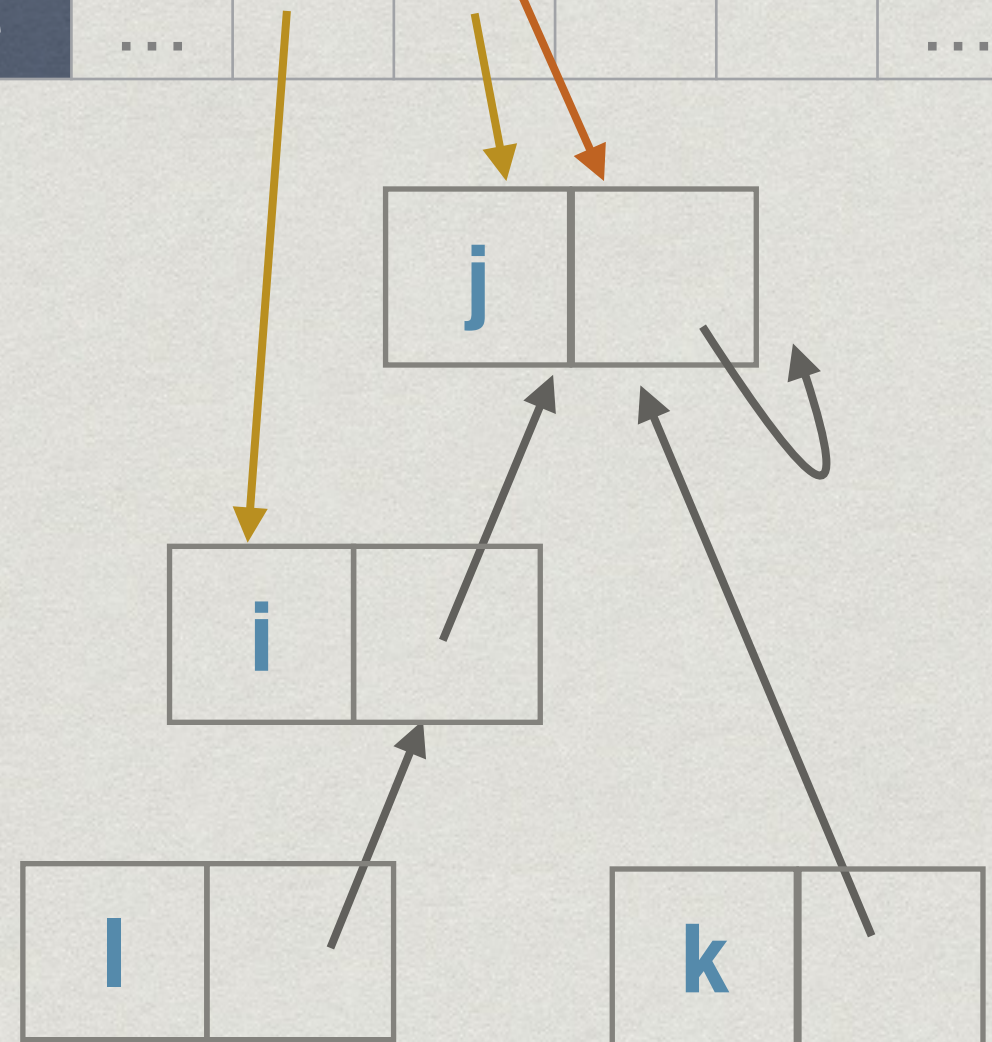
- \* Root[1..n]

- \* Root[k] points to root node of tree for component k

- \* Size[k]

- \* Size of component k

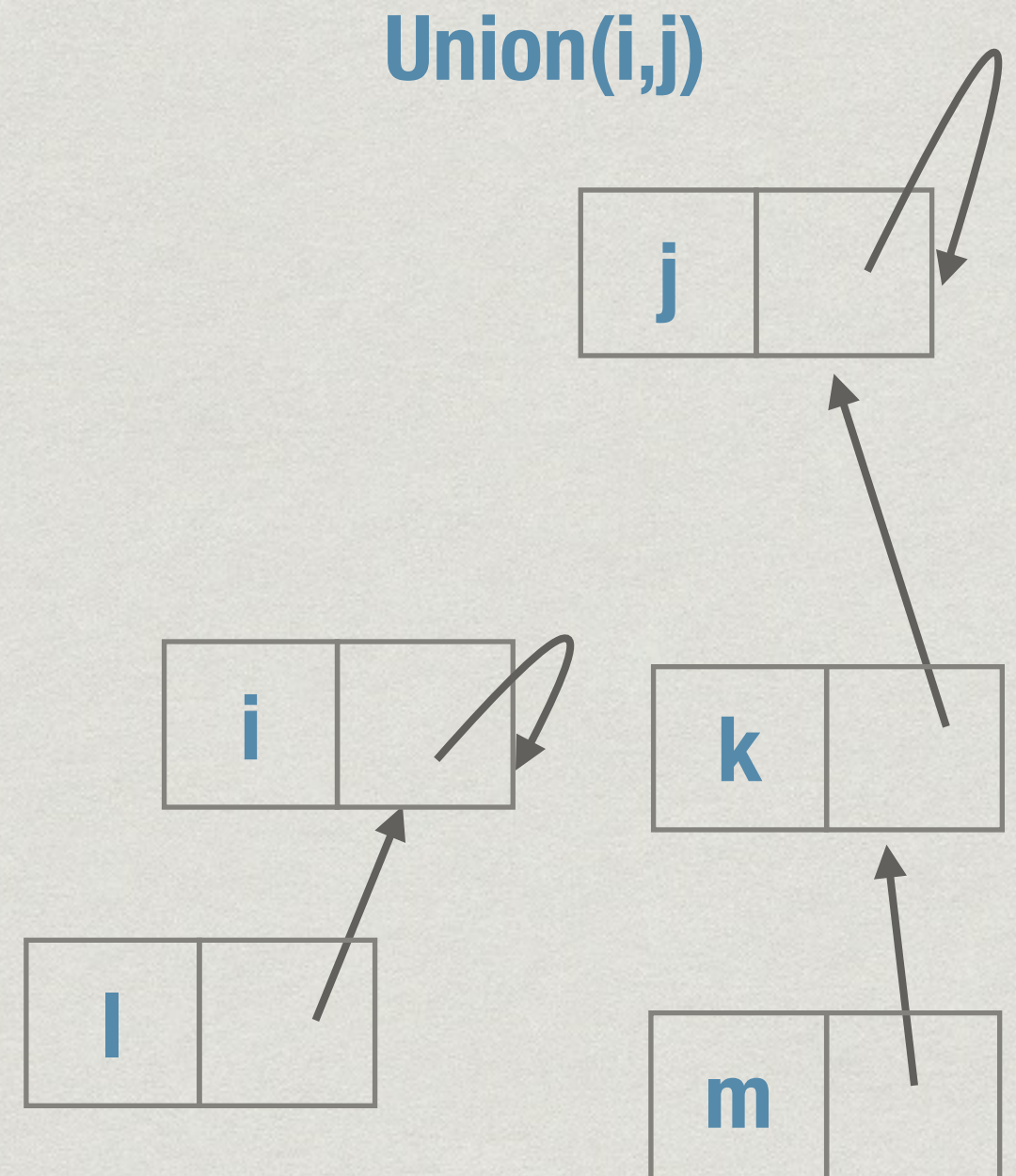
	...	i	j	k	l	...
Size	...	-	4	-	-	...
Root	...	-		-	-	...
Node	...					...





# Union( $k, k'$ )

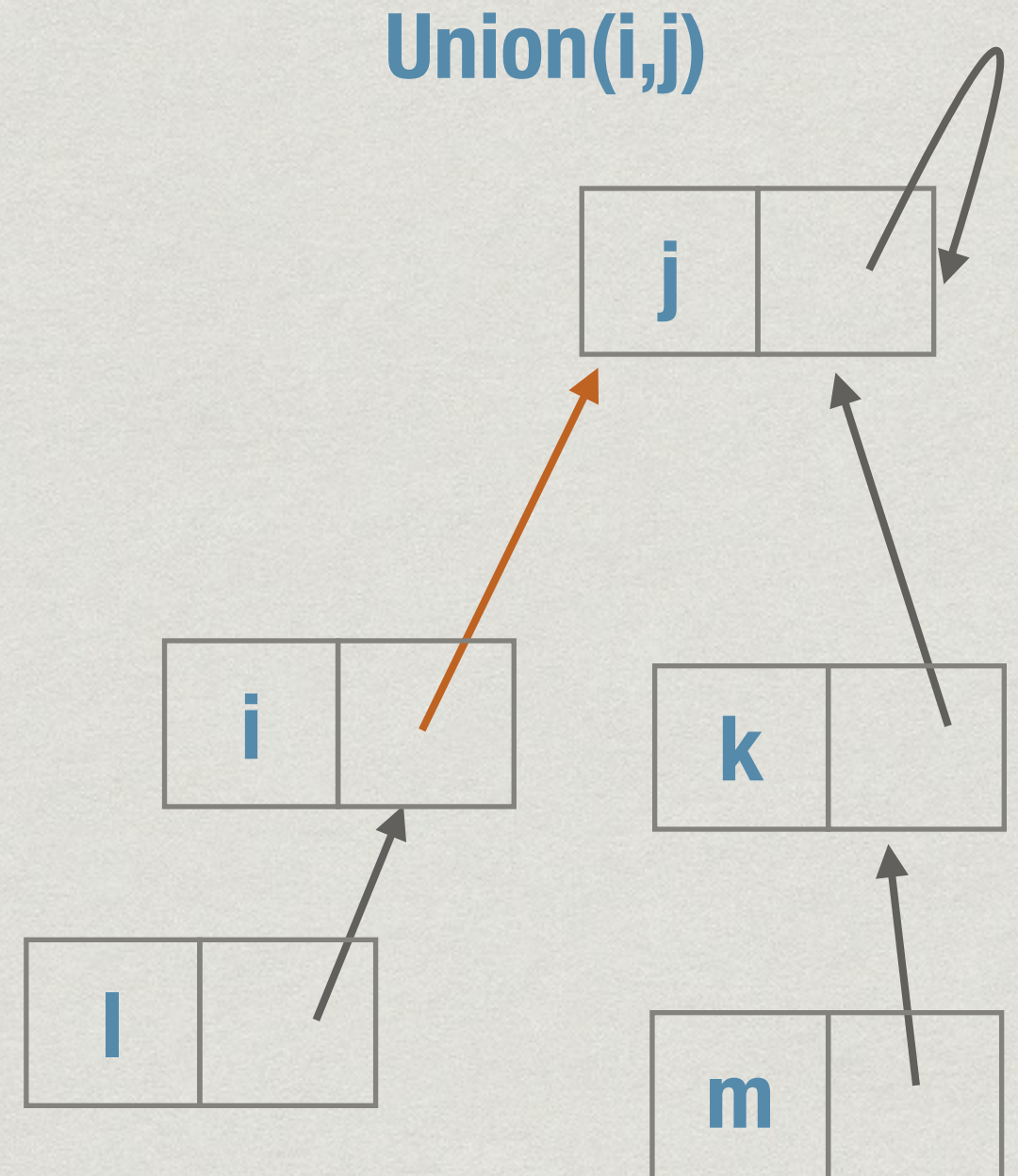
- \* Root of one component points to root of the other
- \* Becomes a direct child of the other root node
- \* As usual, merge smaller component into larger one
- \*  $O(1)$  operation
- \* Size information in  $\text{Size}[k]$
- \*  $\text{Root}[k]$  points to root of  $k$





# Union( $k, k'$ )

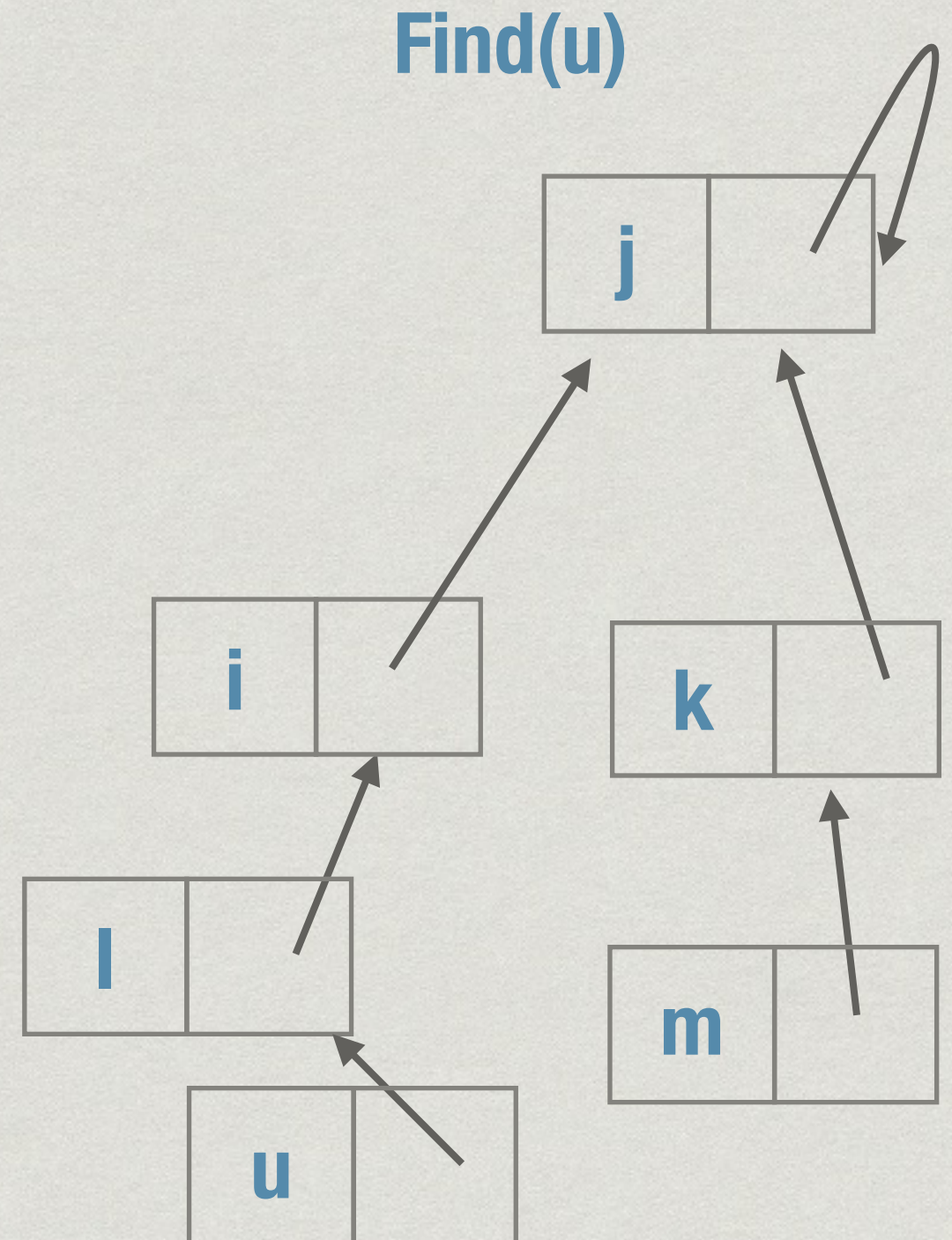
- \* Root of one component points to root of the other
- \* Becomes a direct child of the other root node
- \* As usual, merge smaller component into larger one
- \*  $O(1)$  operation
- \* Size information in  $\text{Size}[k]$
- \*  $\text{Root}[k]$  points to root of  $k$





# Find(j)

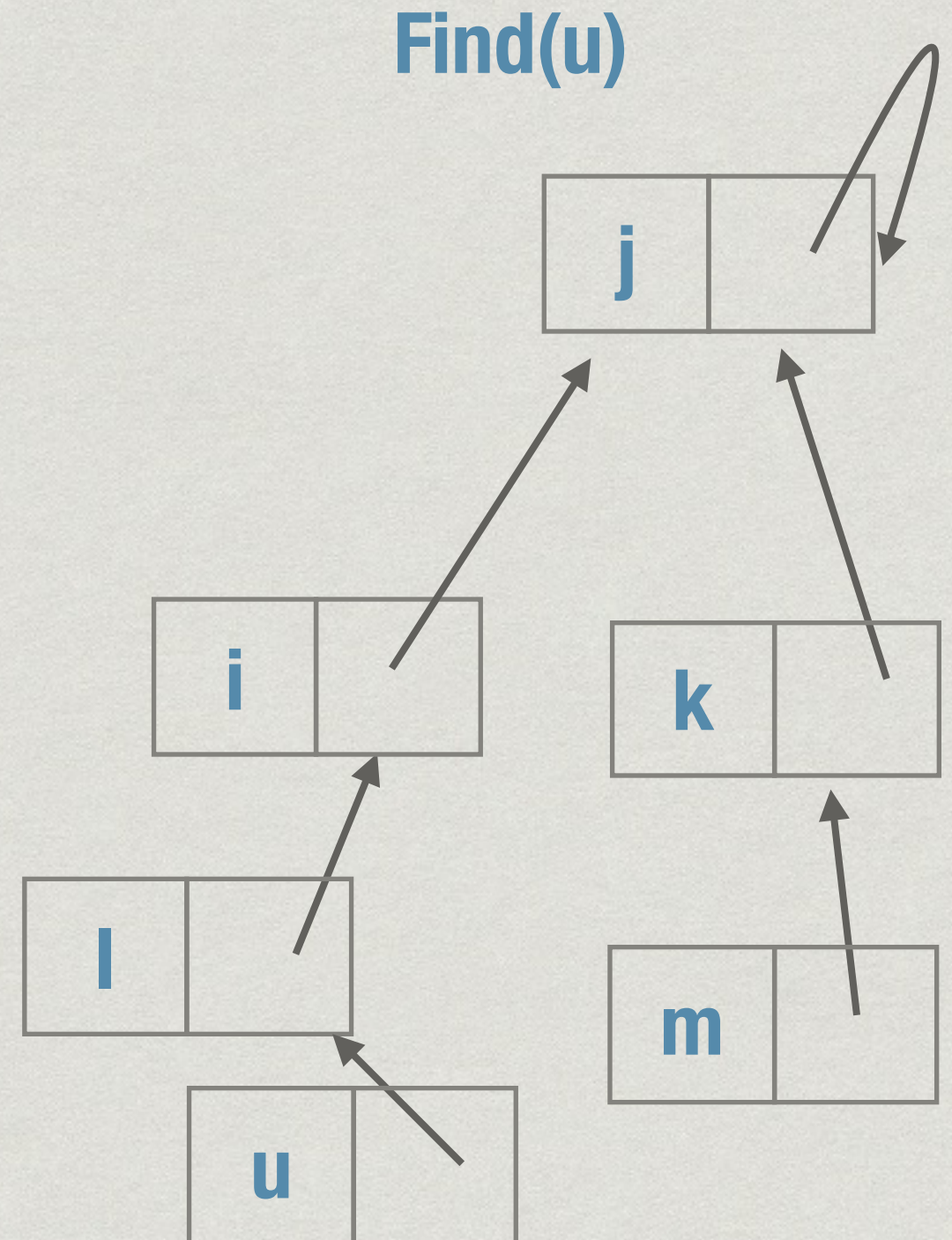
- \* Need to traverse the path from j to root
- \* Path increases by 1 each time label of j changes
- \* Component size doubles with each merge
- \* Max component size is n
- \* Path is at most  $\log n$





# Path compression

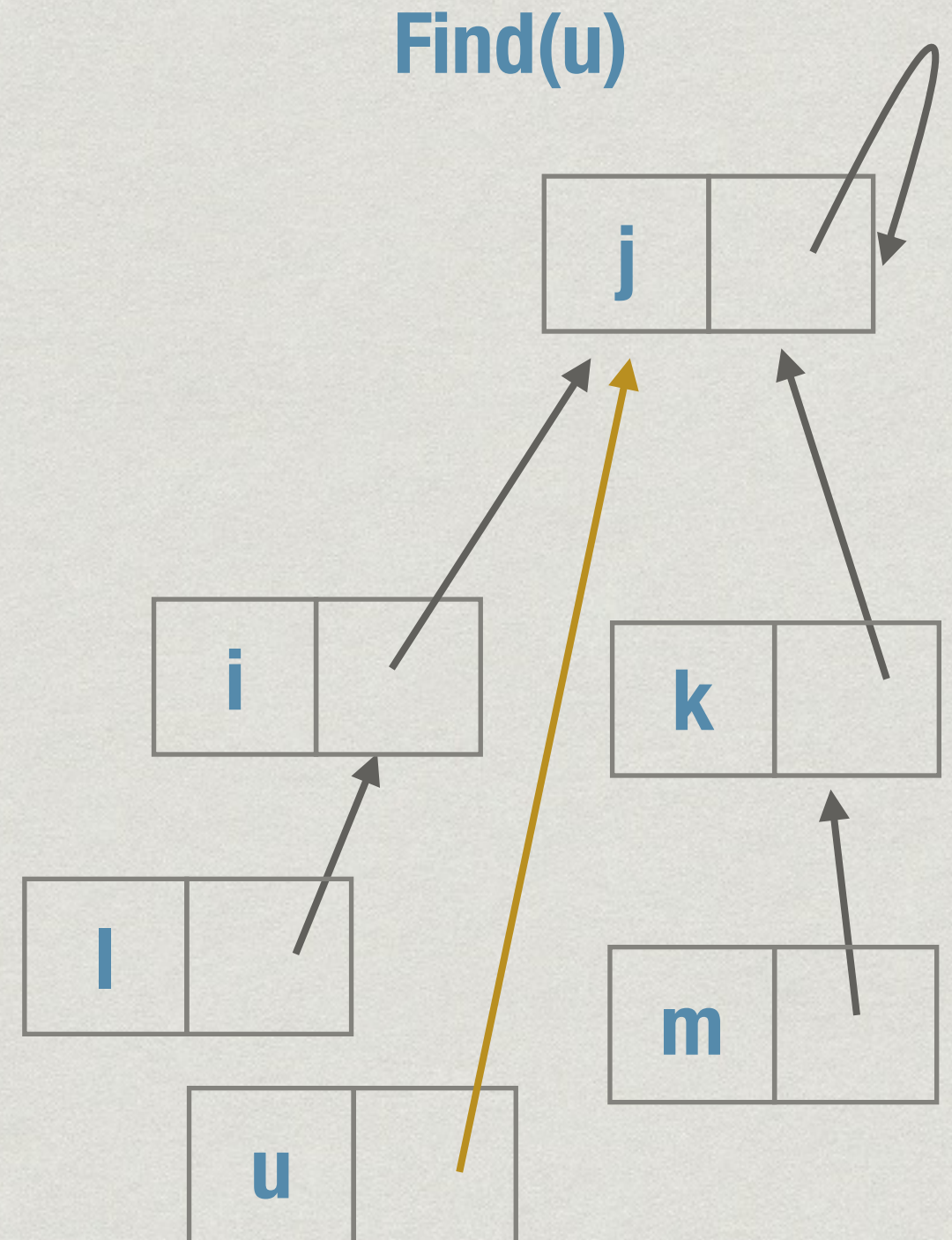
- \* Each Find(j) traverses path from j to root
- \* Remember the answer: make j point directly to root
- \* After each Find(j), retrace path and reset all pointers to point directly to root
- \* “Flatten” the tree





# Path compression

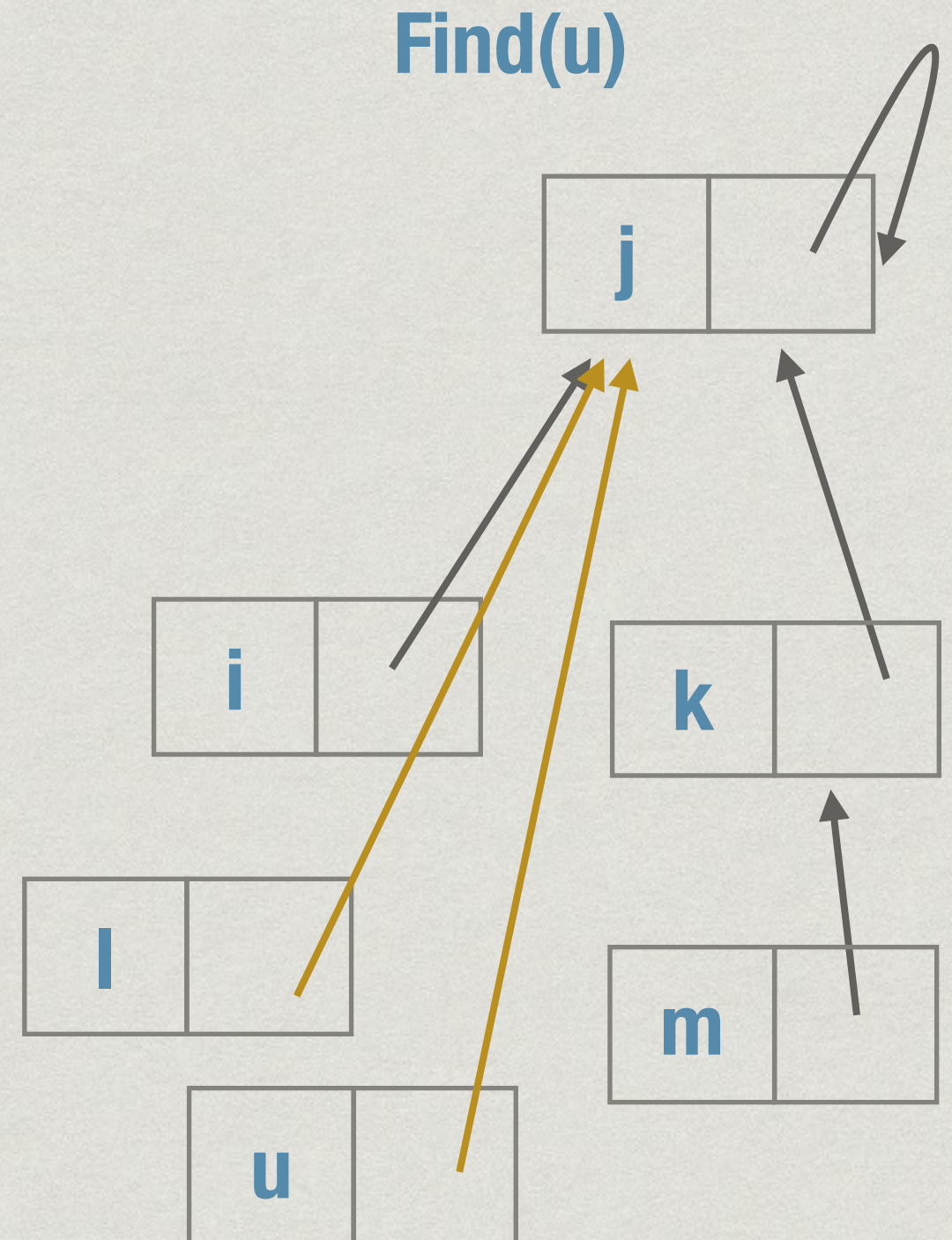
- \* Each Find(j) traverses path from j to root
- \* Remember the answer: make j point directly to root
- \* After each Find(j), retrace path and reset all pointers to point directly to root
- \* “Flatten” the tree





# Path compression

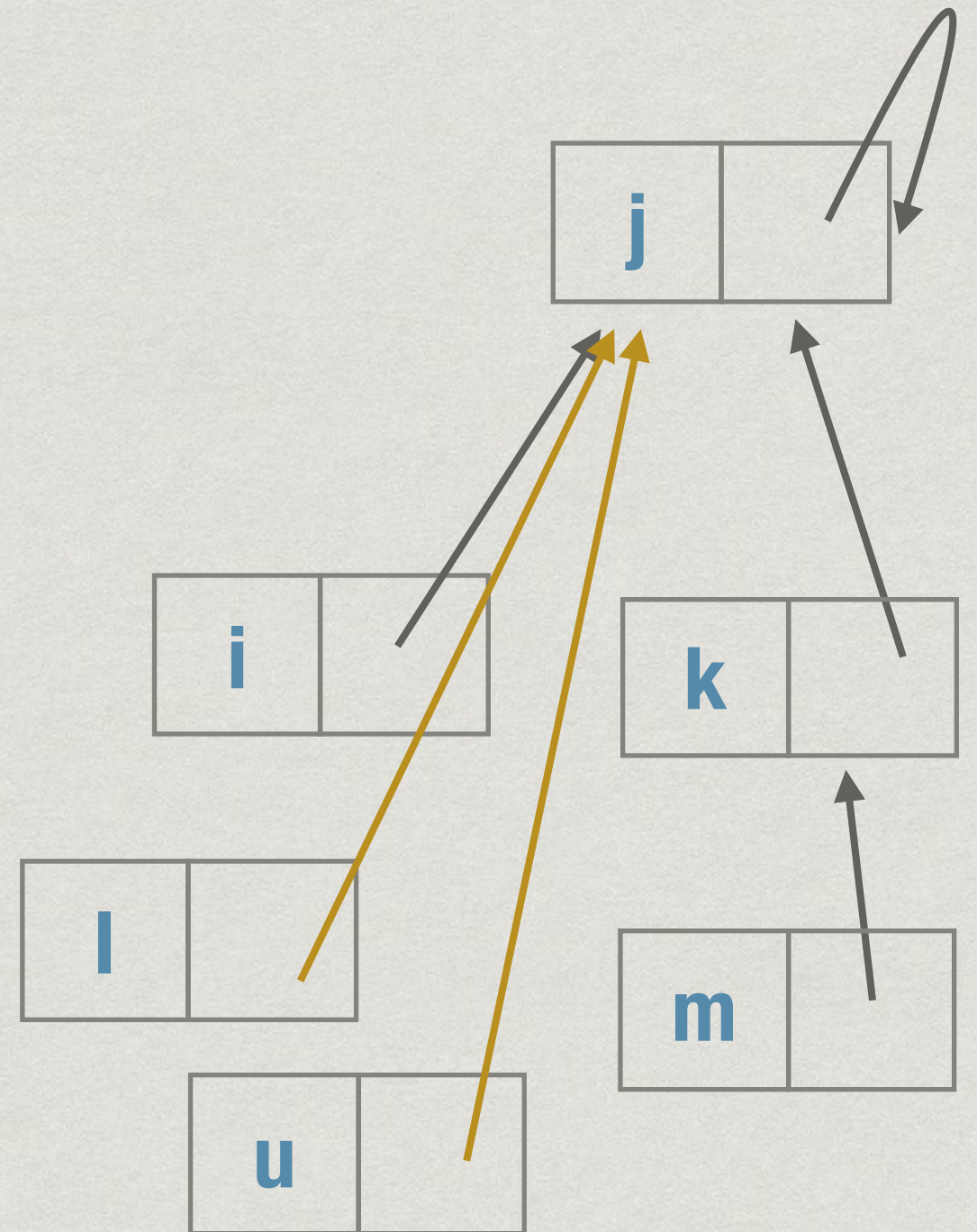
- \* Each Find(j) traverses path from j to root
- \* Remember the answer: make j point directly to root
- \* After each Find(j), retrace path and reset all pointers to point directly to root
- \* “Flatten” the tree





# Path compression

- \* First Find(j) takes  $\log n$
- \* Then  $O(1)$
- \* Overall,  $n$  finds using path compression take time almost linear in  $n$
- \*  $O(n \alpha(n))$
- \*  $\alpha(n)$  : inverse Ackermann function, grows very slowly





# Summary

- \* Implement Union-Find using nodes with pointers
- \* **MakeUnionFind(S)** is  $O(n)$
- \* **Union(k,k')** is  $O(1)$
- \* **Find(s)** is  $O(n \alpha(n))$ 
  - \* Use path compression to speed up repeated Find(s) operations