Lecture 11

Disjoint-Set Data Structure

Source: Introduction to Algorithms, CLRS

Finding Friend Groups on Facebook

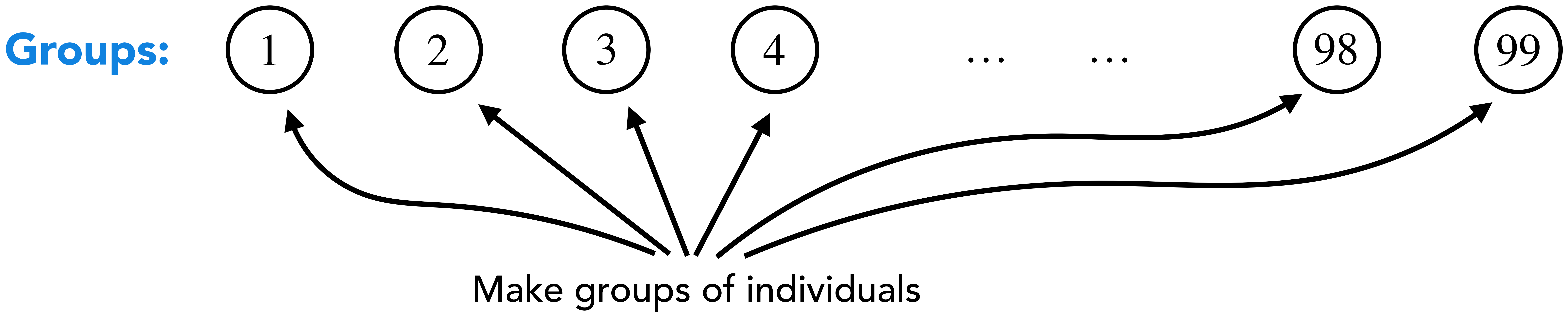
Finding Friend Groups on Facebook

Users:

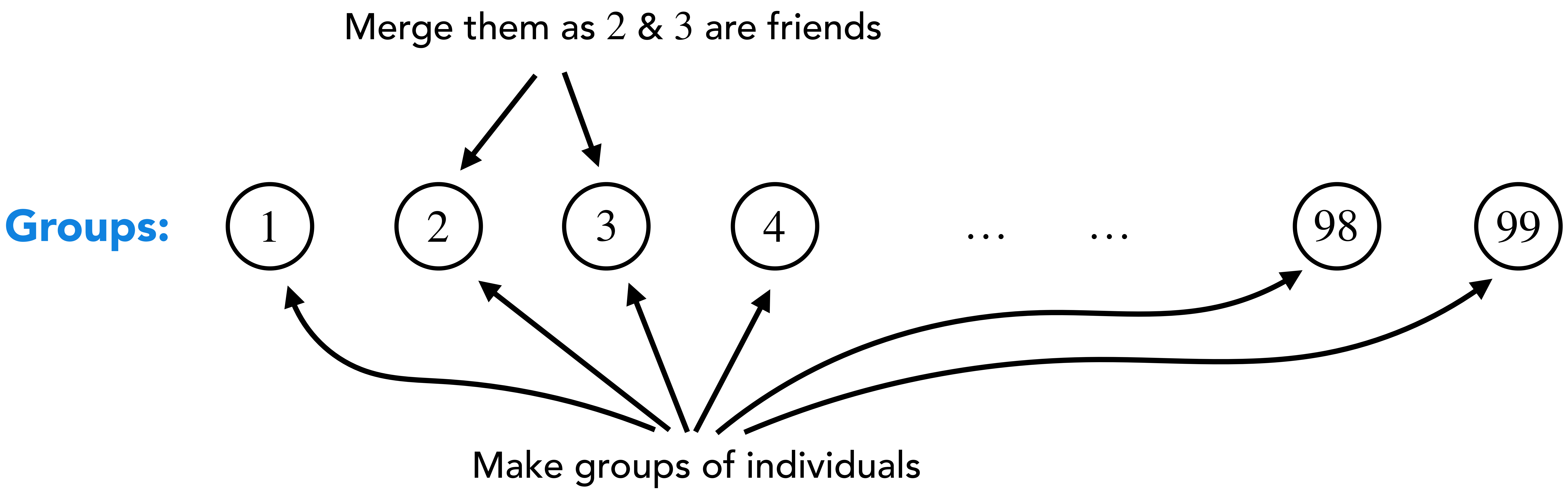
Finding Friend Groups on Facebook

Users: 1 2 3 4 … … 98 99

Finding Friend Groups on Facebook



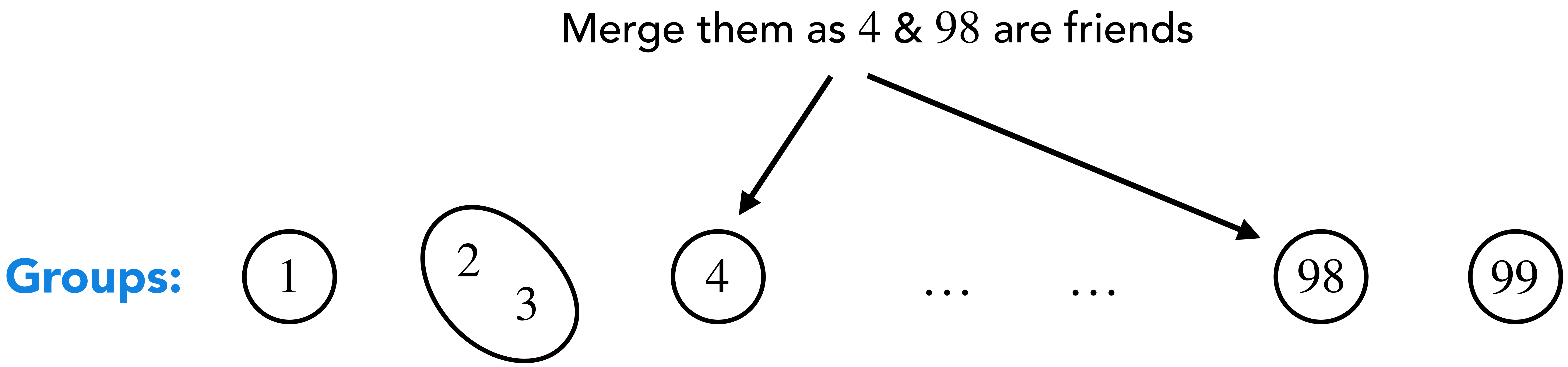
Finding Friend Groups on Facebook



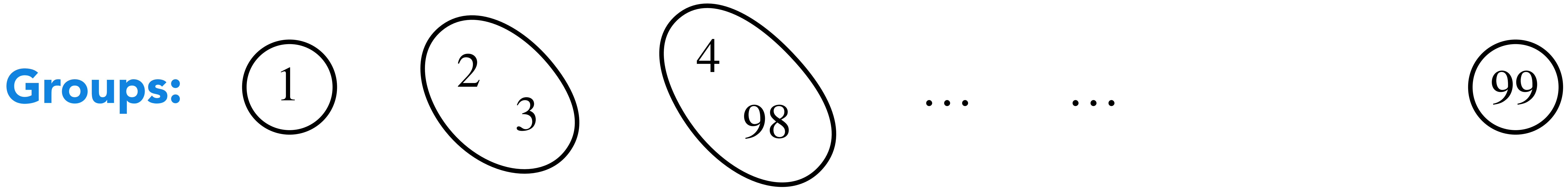
Finding Friend Groups on Facebook

Groups:

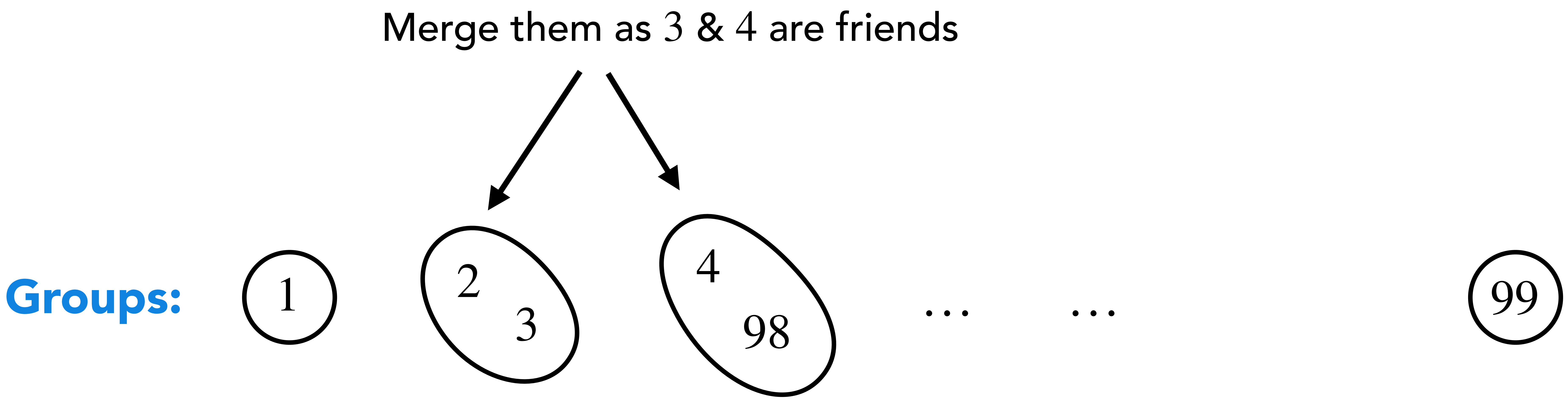
Finding Friend Groups on Facebook



Finding Friend Groups on Facebook

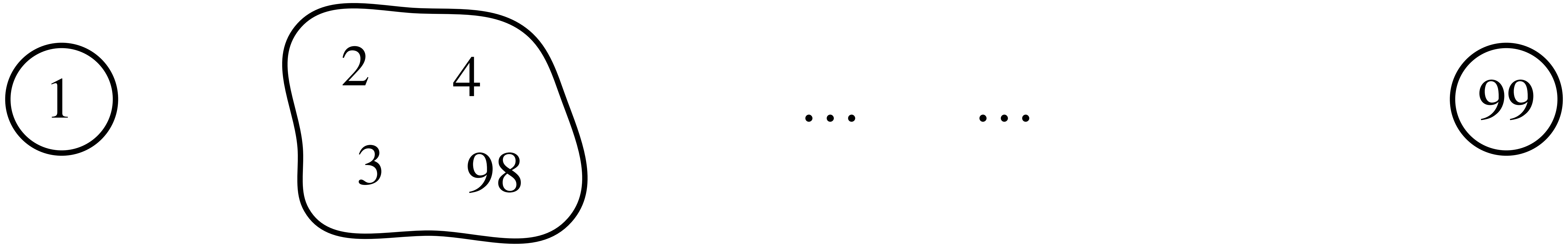


Finding Friend Groups on Facebook

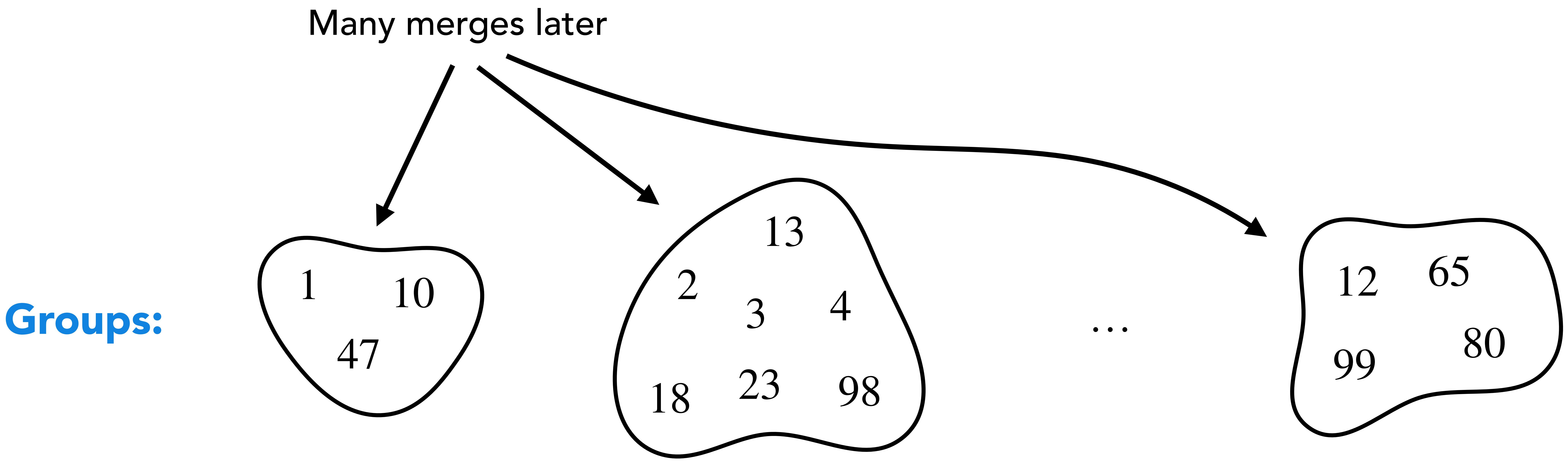


Finding Friend Groups on Facebook

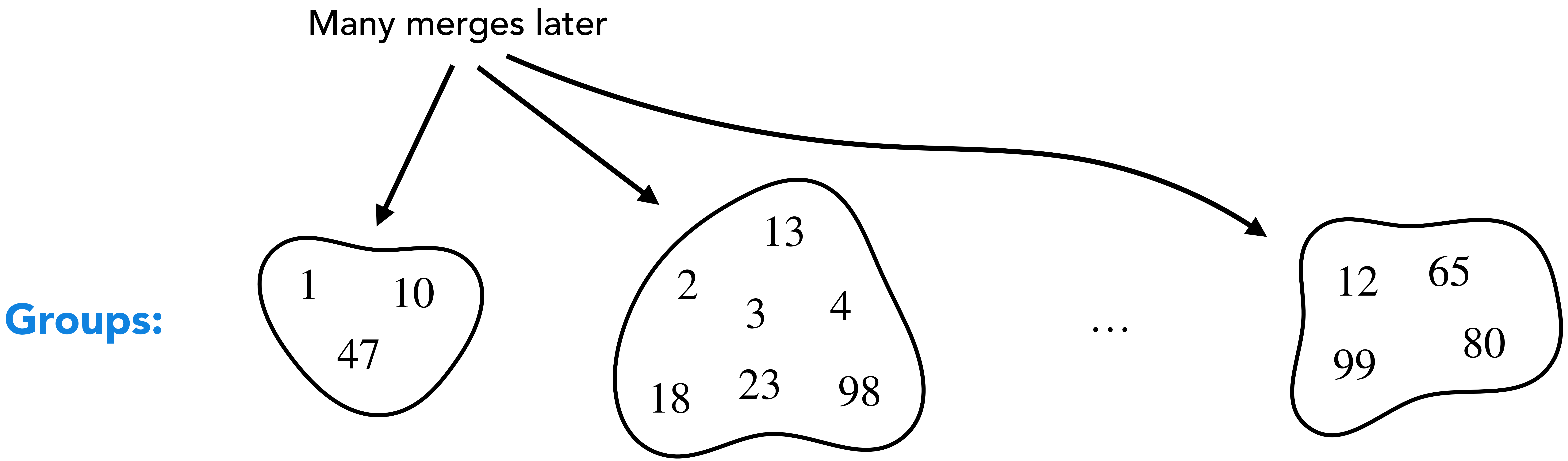
Groups:



Finding Friend Groups on Facebook

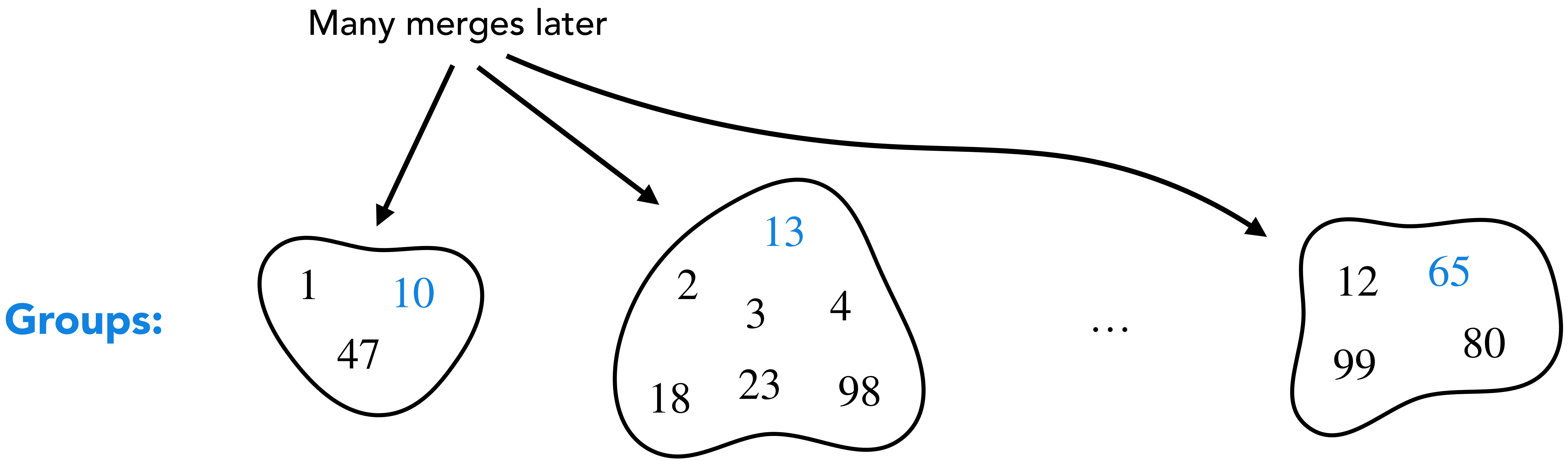


Finding Friend Groups on Facebook



There should be a way to tell to which group a user belongs.

Finding Friend Groups on Facebook

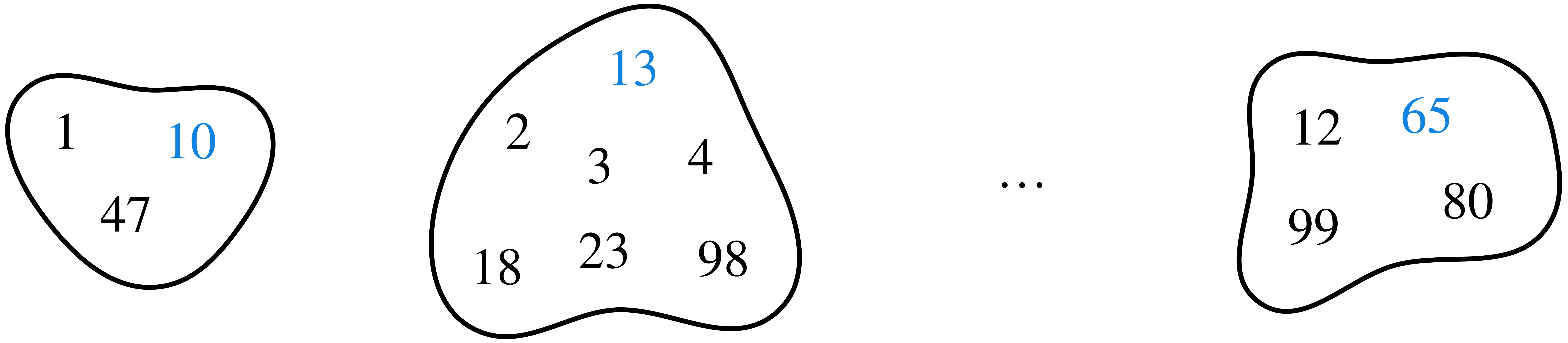


There should be a way to tell to which group a user belongs.

We achieve that by having a representative for every group.

Finding Friend Groups on Facebook

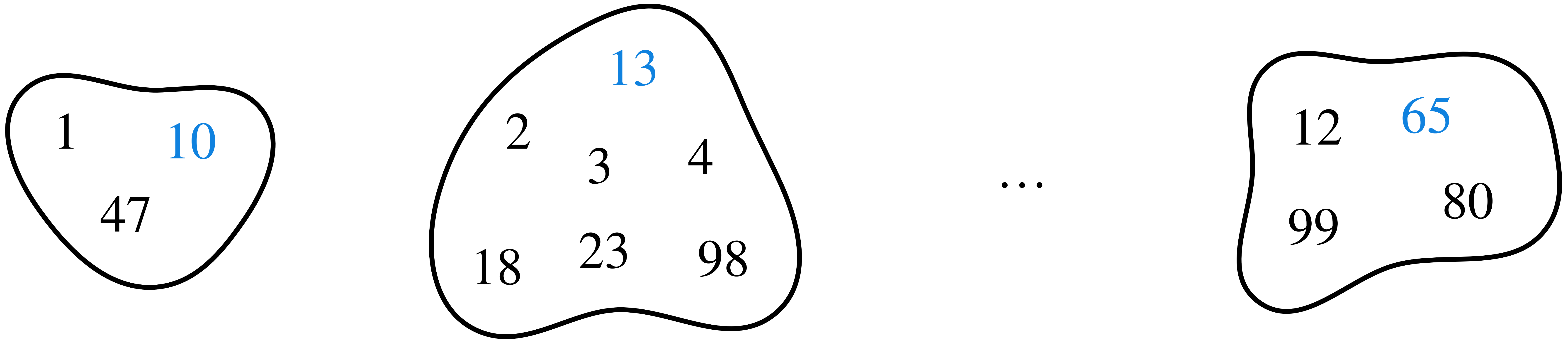
Groups:



Goal: Design a data-structure so that:

Finding Friend Groups on Facebook

Groups:

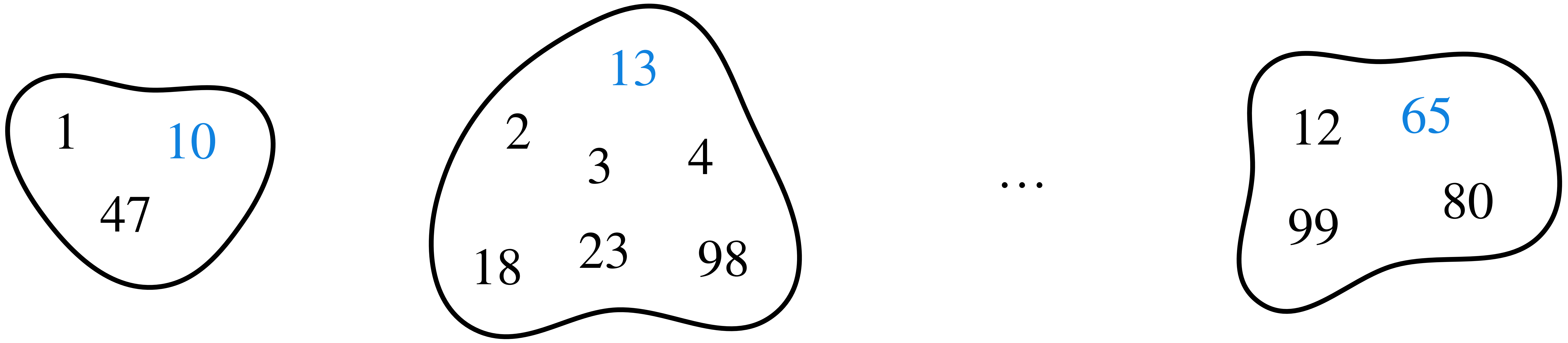


Goal: Design a data-structure so that:

- Merging is fast.

Finding Friend Groups on Facebook

Groups:

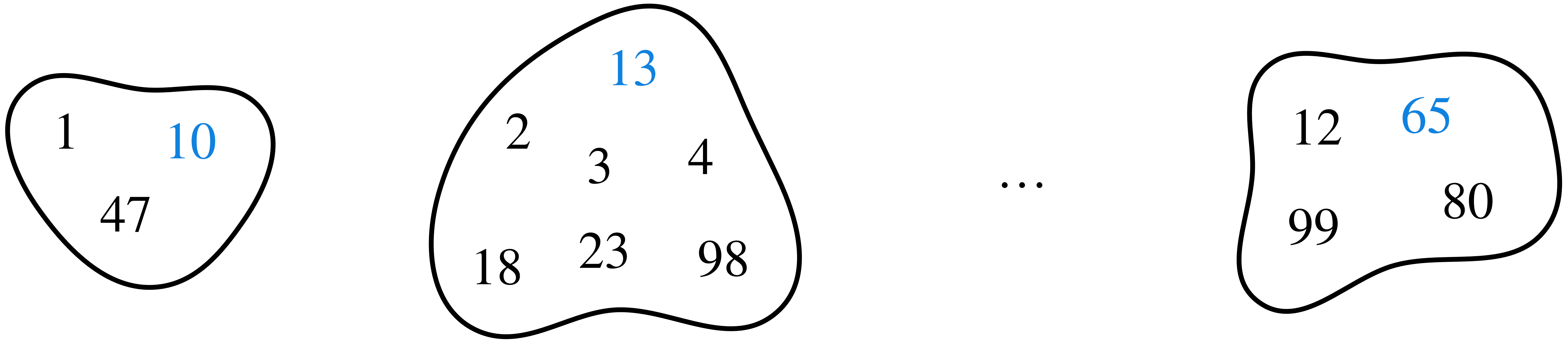


Goal: Design a data-structure so that:

- Merging is fast.
- Finding representative is fast.

Finding Friend Groups on Facebook

Groups:



Goal: Design a data-structure so that:

- Merging is fast.
- Finding representative is fast.

Instead of optimising the cost of individual operations we will optimise a sequence of such operations.

Disjoint-set Data Structure

Disjoint-set Data Structure

Disjoint-set data structure maintains:

Disjoint-set Data Structure

Disjoint-set data structure maintains:

- A collection $S = \{S_1, S_2, \ldots, S_k\}$ of disjoint dynamic sets.

Disjoint-set Data Structure

Disjoint-set data structure maintains:

- A collection $S = \{S_1, S_2, \ldots, S_k\}$ of disjoint dynamic sets.
- A representative for each set which is a member of the set.

Disjoint-set Data Structure

Disjoint-set data structure maintains:

- A collection $S = \{S_1, S_2, \ldots, S_k\}$ of disjoint dynamic sets.
- A representative for each set which is a member of the set.

Operations of disjoint-set data structure:

Disjoint-set Data Structure

Disjoint-set data structure maintains:

- A collection $S = \{S_1, S_2, \ldots, S_k\}$ of disjoint dynamic sets.
- A representative for each set which is a member of the set.

Operations of disjoint-set data structure:

- **Make-Set**(x): Creates a new set with x as the only member. Make x its own representative.

Disjoint-set Data Structure

Disjoint-set data structure maintains:

- A collection $S = \{S_1, S_2, \ldots, S_k\}$ of disjoint dynamic sets.
- A representative for each set which is a member of the set.

Operations of disjoint-set data structure:

- **Make-Set**(x): Creates a new set with x as the only member. Make x its own representative.
- **Union**(x, y): Adds $S_x \cup S_y$ to the collection, where S_x and S_y contain x and y, respectively.

Disjoint-set Data Structure

Disjoint-set data structure maintains:

- A collection $S = \{S_1, S_2, \ldots, S_k\}$ of disjoint dynamic sets.
- A representative for each set which is a member of the set.

Operations of disjoint-set data structure:

- **Make-Set**(x): Creates a new set with x as the only member. Make x its own representative.
- **Union**(x, y): Adds $S_x \cup S_y$ to the collection, where S_x and S_y contain x and y, respectively.
  Choose a representative for $S_x \cup S_y$.

Disjoint-set Data Structure

Disjoint-set data structure maintains:

- A collection $S = \{S_1, S_2, \ldots, S_k\}$ of disjoint dynamic sets.
- A representative for each set which is a member of the set.

Operations of disjoint-set data structure:

- **Make-Set**(x): Creates a new set with x as the only member. Make x its own representative.
- **Union**(x, y): Adds $S_x \cup S_y$ to the collection, where S_x and S_y contain x and y, respectively.
  Choose a representative for $S_x \cup S_y$. Destroys S_x and S_y.

Disjoint-set Data Structure

Disjoint-set data structure maintains:

- A collection $S = \{S_1, S_2, \ldots, S_k\}$ of disjoint dynamic sets.
- A representative for each set which is a member of the set.

Operations of disjoint-set data structure:

- **Make-Set**(x): Creates a new set with x as the only member. Make x its own representative.
- **Union**(x, y): Adds $S_x \cup S_y$ to the collection, where S_x and S_y contain x and y, respectively.
  Choose a representative for $S_x \cup S_y$. Destroys S_x and S_y.
- **Find-Set**(x): Gives the representative of the unique set that contains x.

Applications of Disjoint-set Data Structure

Applications of Disjoint-set Data Structure

Disjoint-set data structure is useful in:

Applications of Disjoint-set Data Structure

Disjoint-set data structure is useful in:

- Finding friend groups on social networks.

Applications of Disjoint-set Data Structure

Disjoint-set data structure is useful in:

- Finding friend groups on social networks.
- Finding connected components in graphs.

Applications of Disjoint-set Data Structure

Disjoint-set data structure is useful in:

- Finding friend groups on social networks.
- Finding connected components in graphs.
- Kruskal's algorithms to find minimum spanning tree.

Applications of Disjoint-set Data Structure

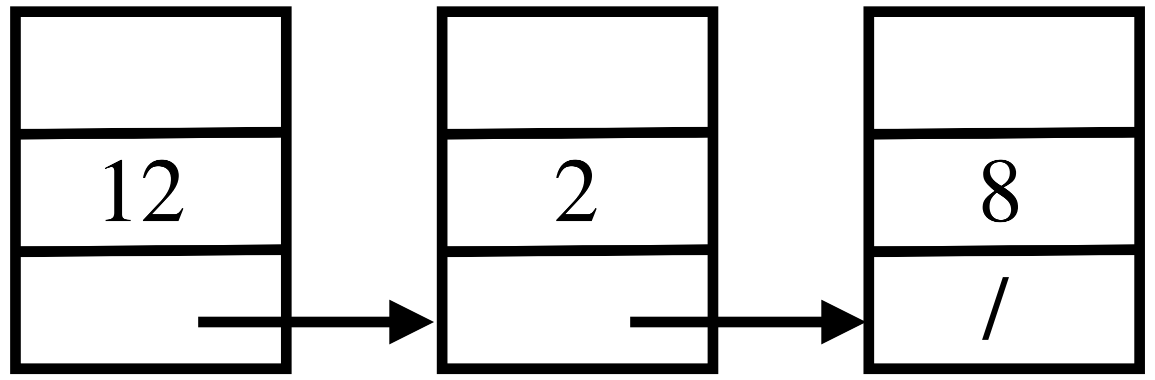
Disjoint-set data structure is useful in:

- Finding friend groups on social networks.
- Finding connected components in graphs.
- Kruskal's algorithms to find minimum spanning tree.
- Finding systems on the same network, etc.

Disjoint-Sets as Linked Lists

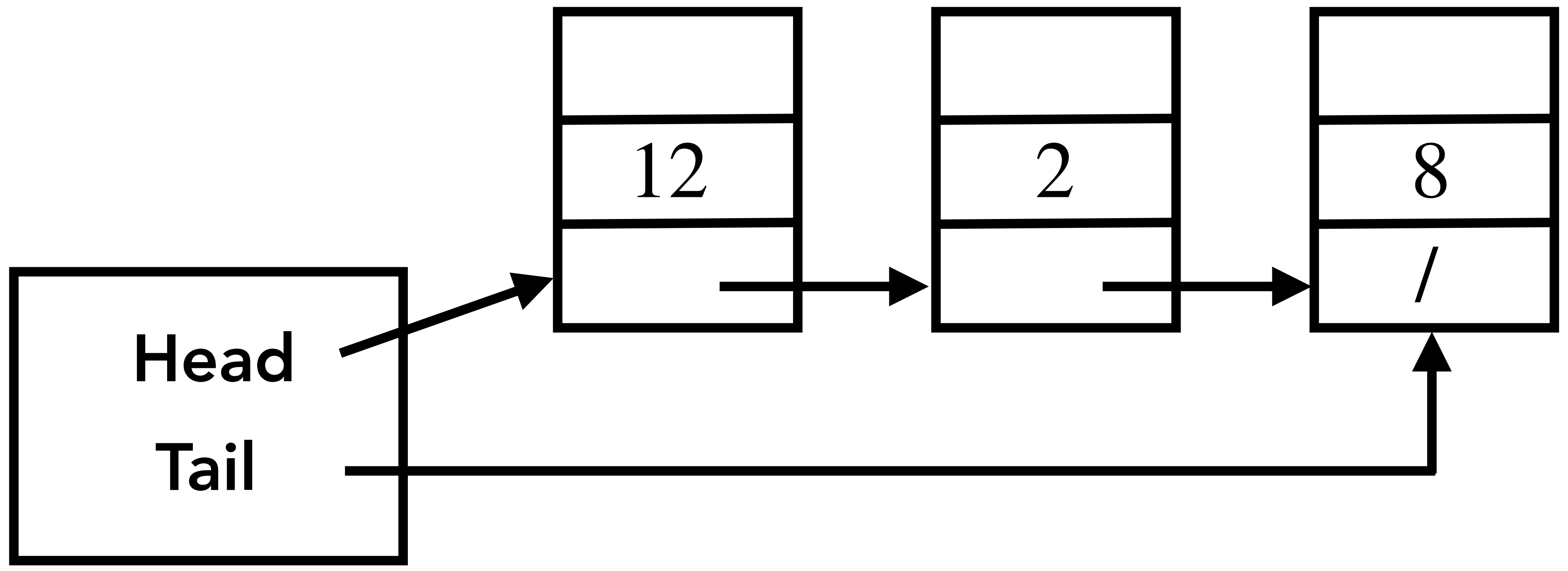
Disjoint-Sets as Linked Lists

S_1



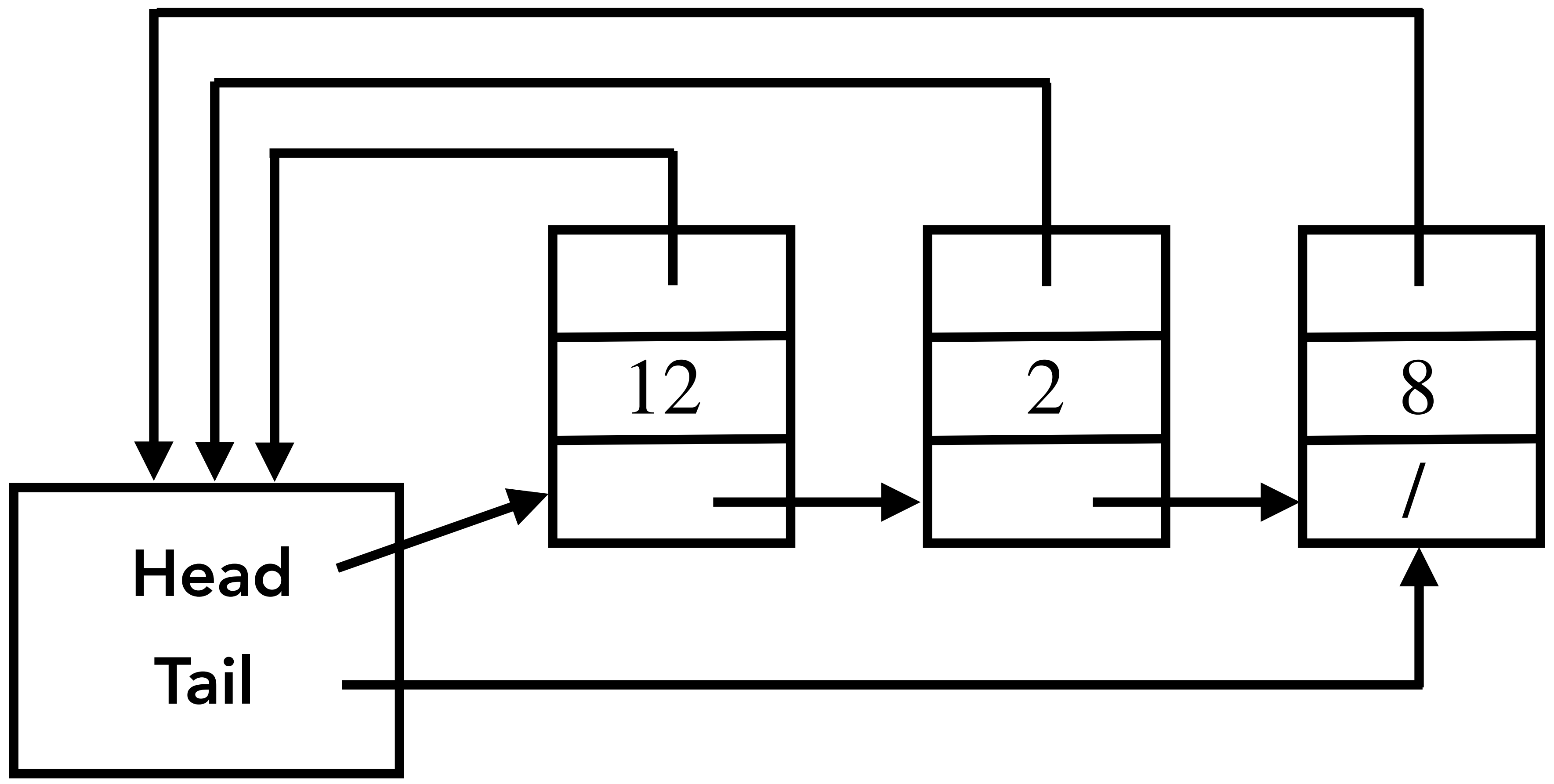
Disjoint-Sets as Linked Lists

S_1

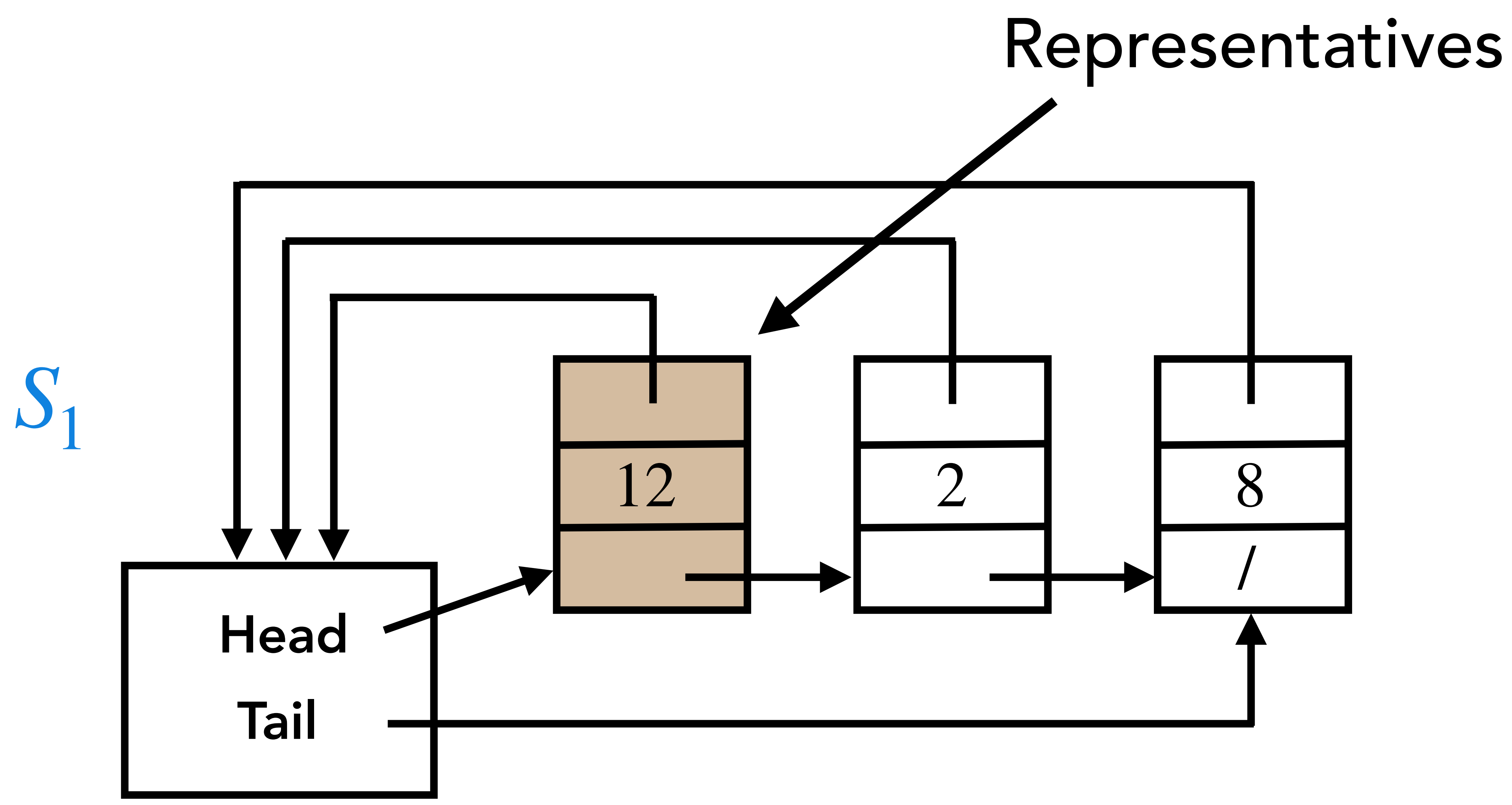


Disjoint-Sets as Linked Lists

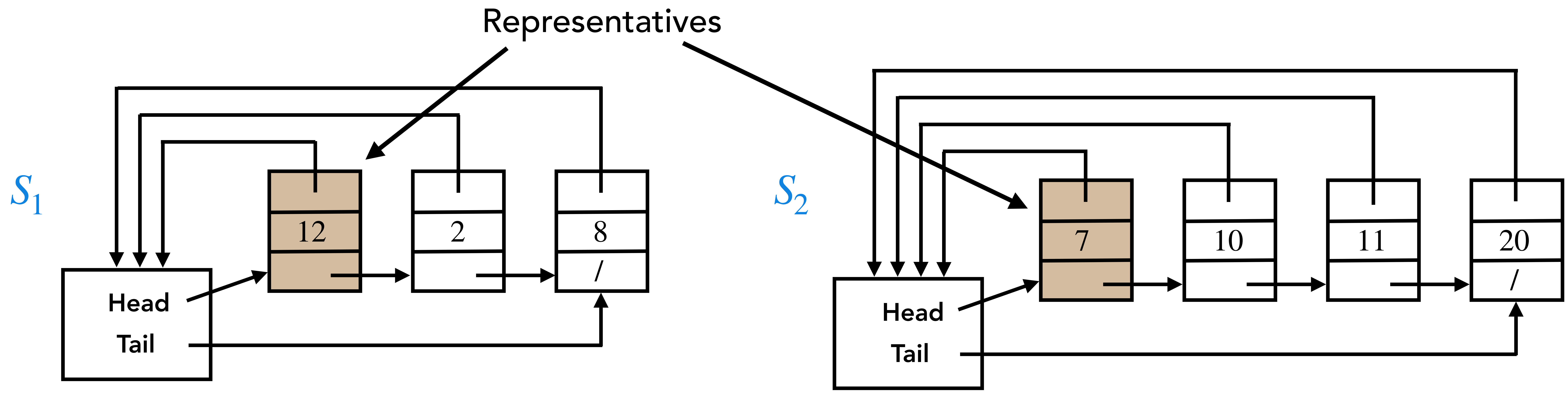
S_1



Disjoint-Sets as Linked Lists

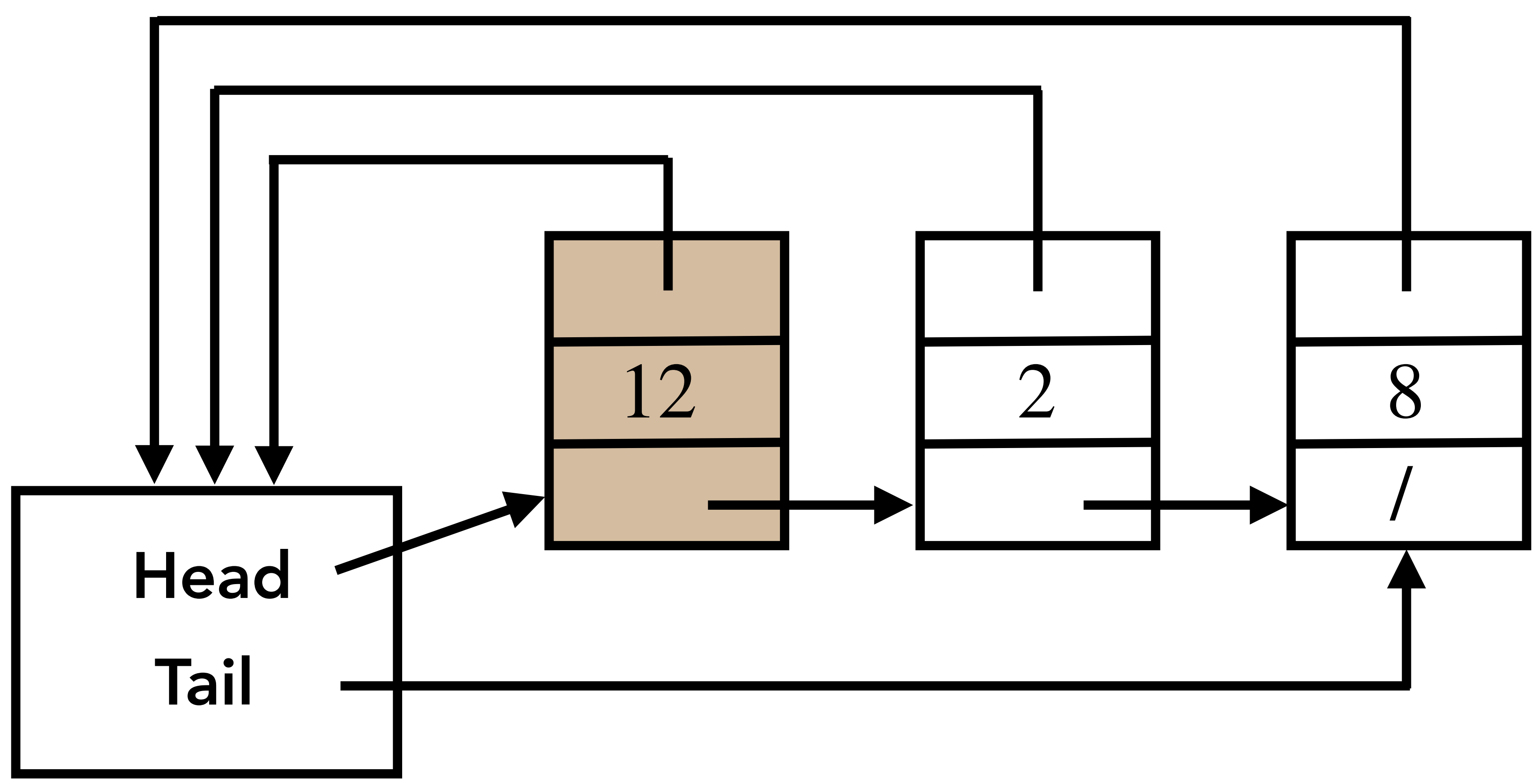


Disjoint-Sets as Linked Lists

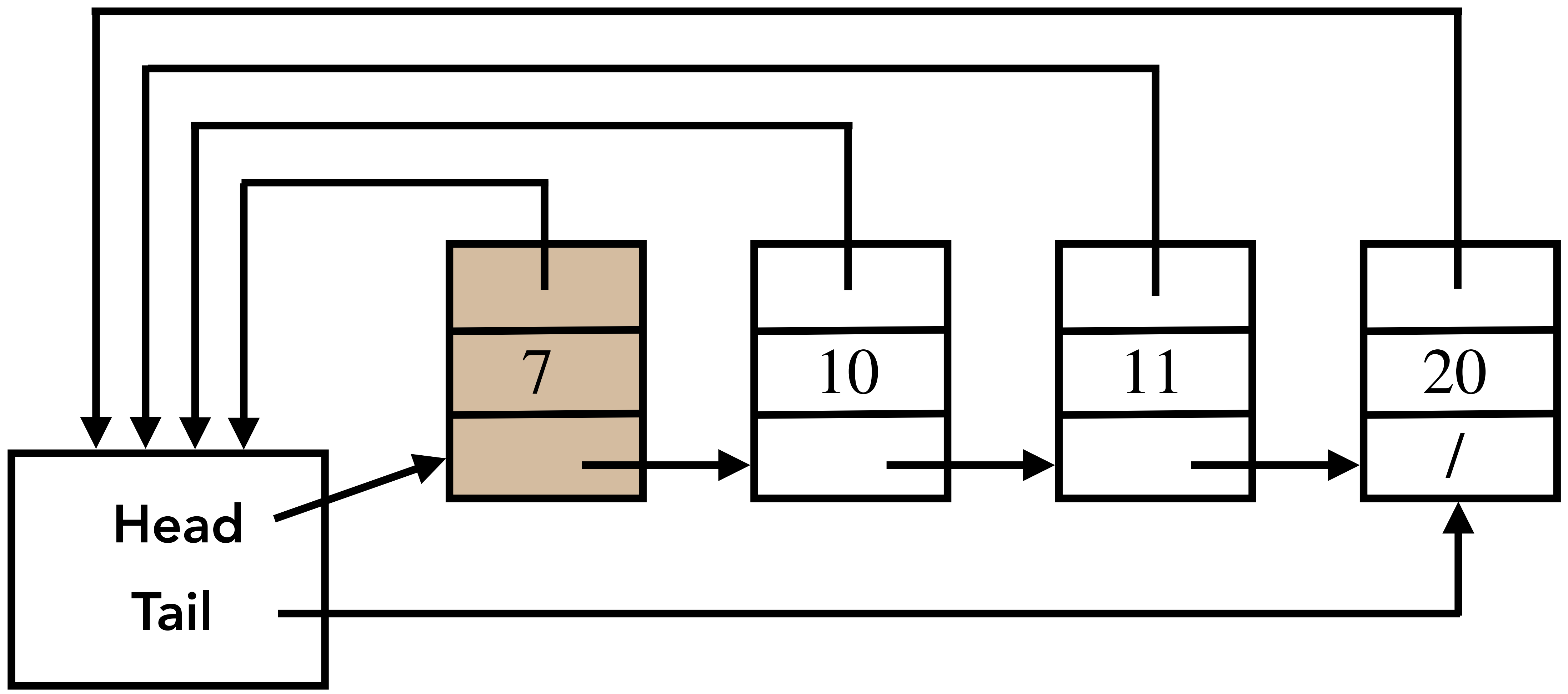


Disjoint-Sets as Linked Lists

S_1

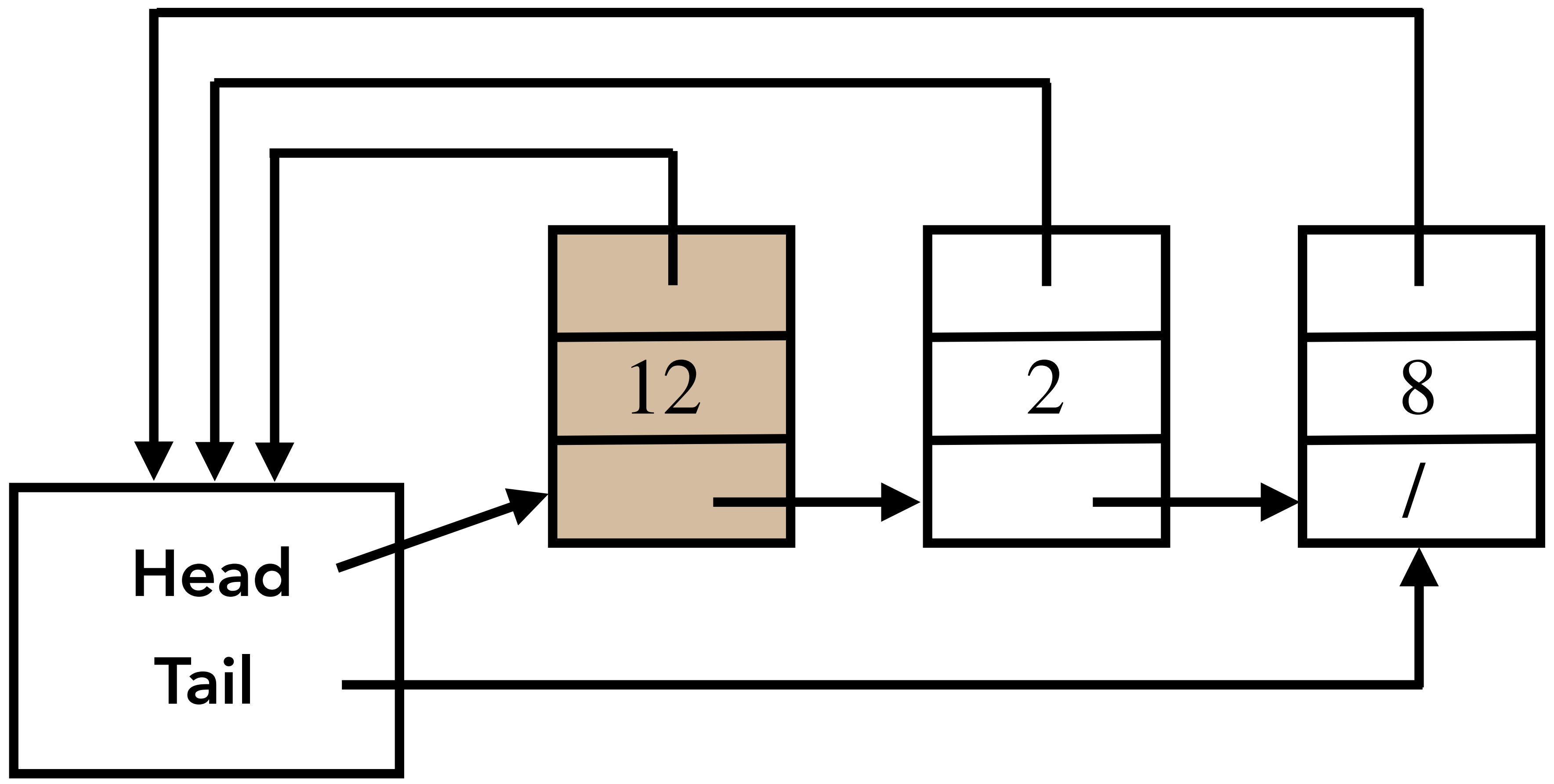


S_2

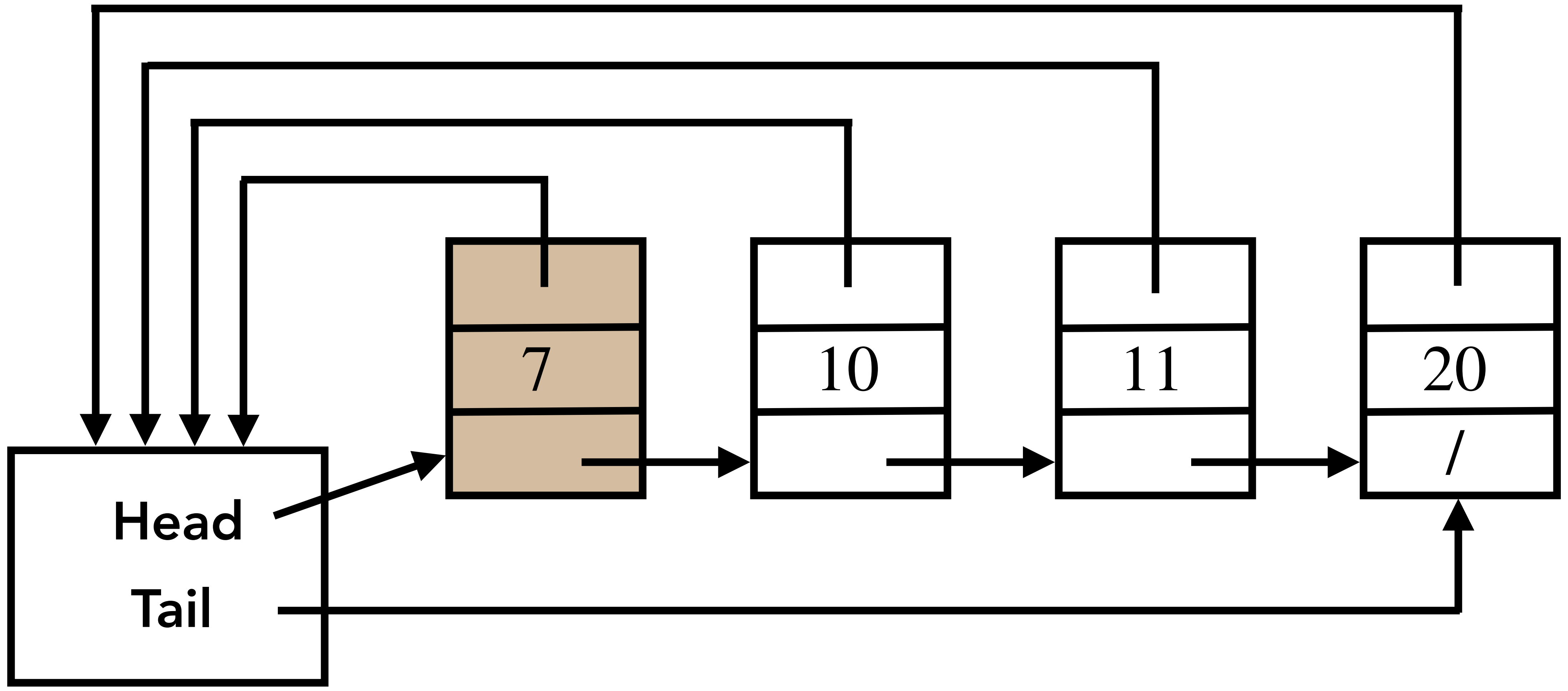


Disjoint-Sets as Linked Lists

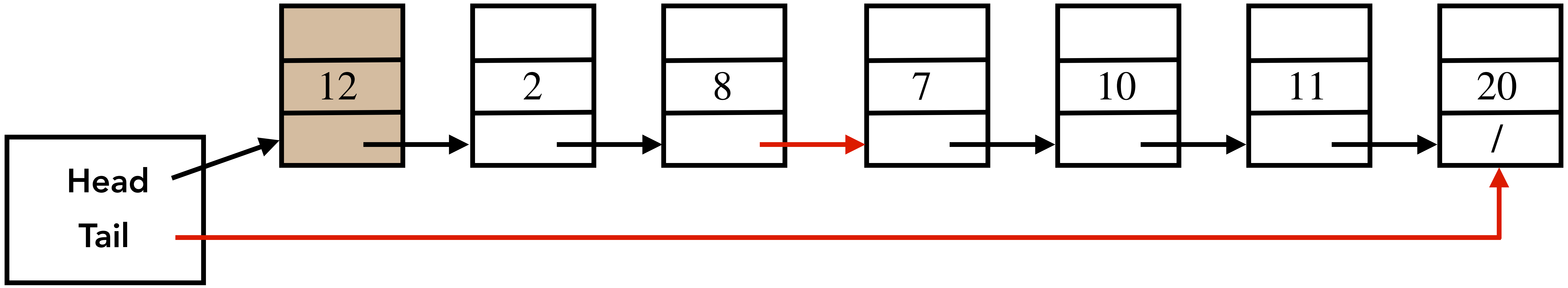
S_1



S_2

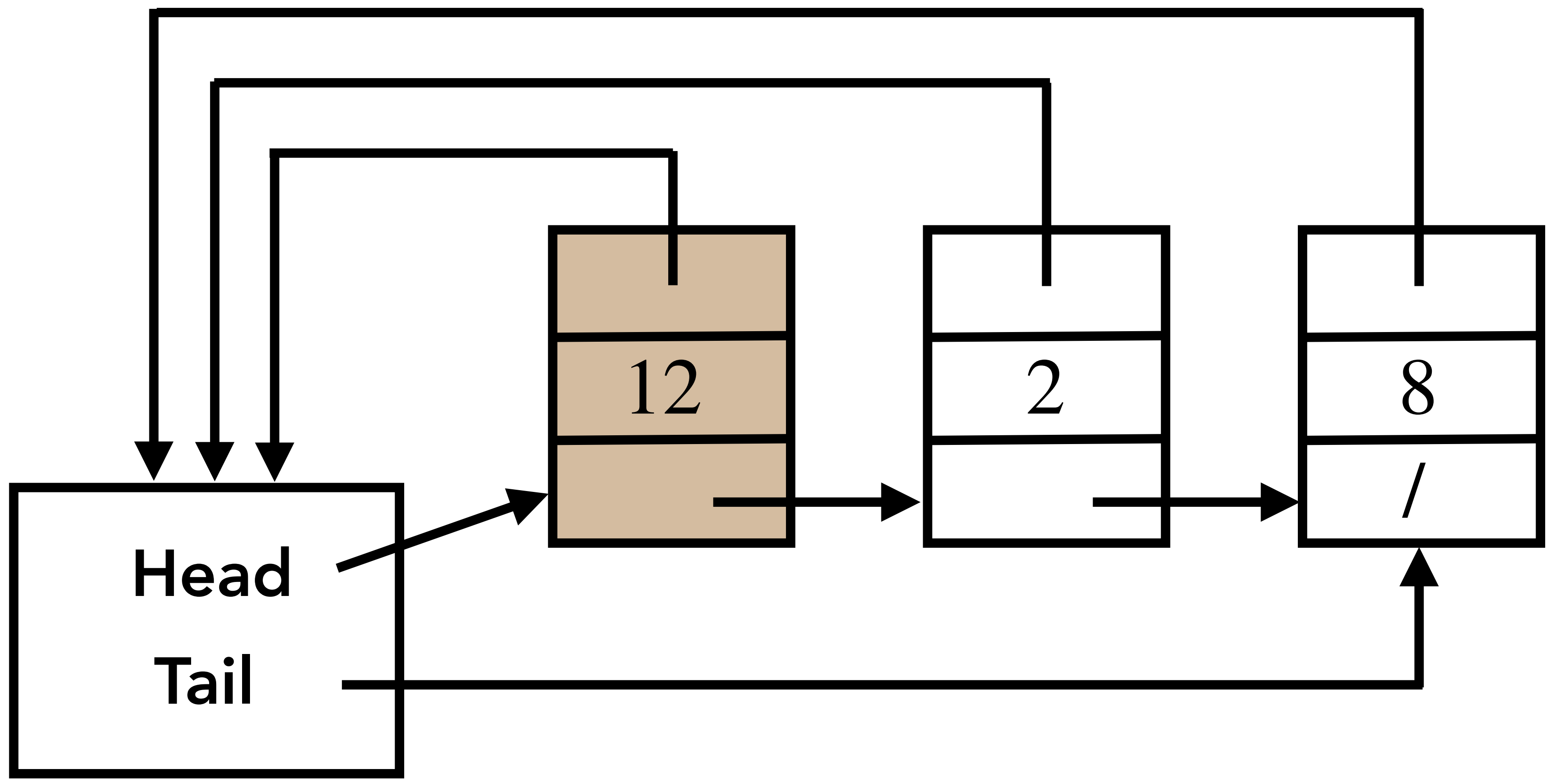


$S_1 \cup S_2$

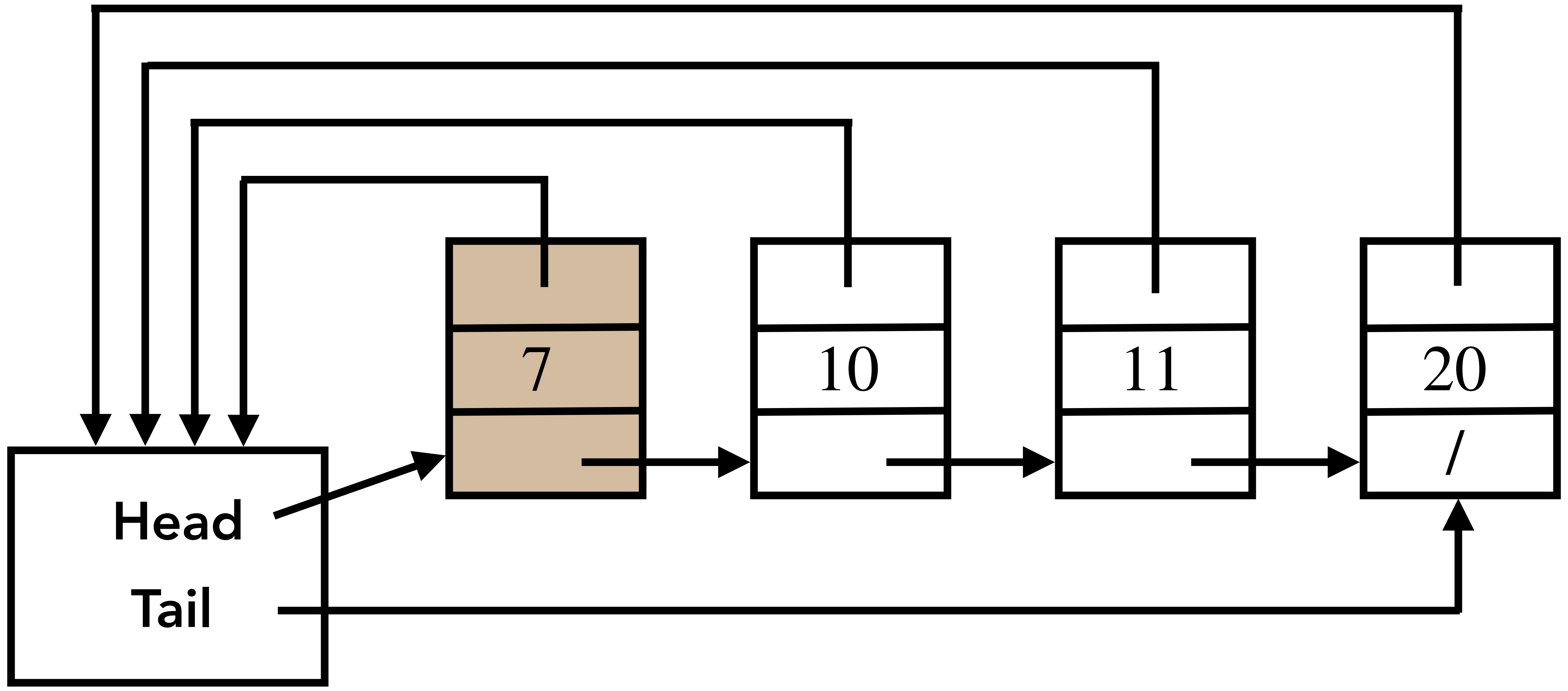


Disjoint-Sets as Linked Lists

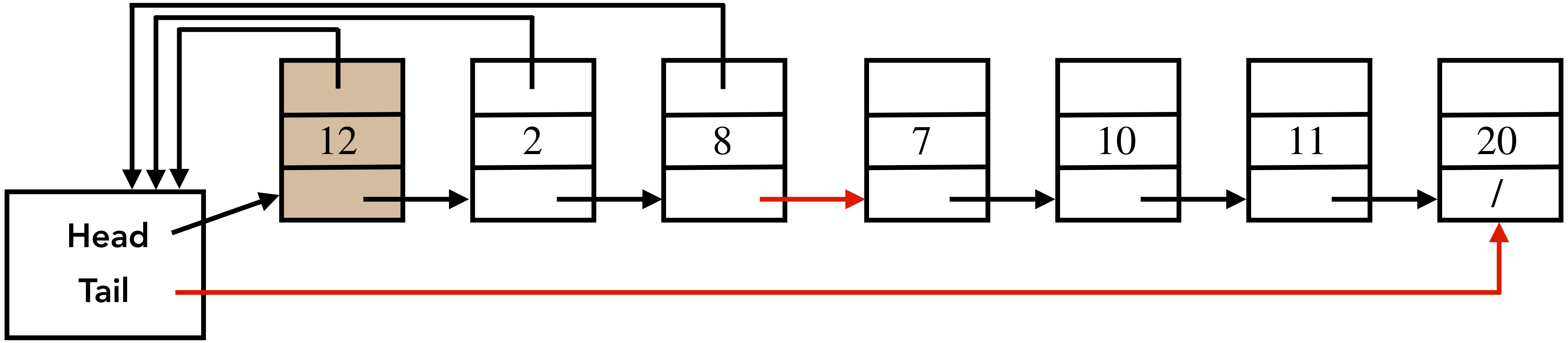
S_1



S_2

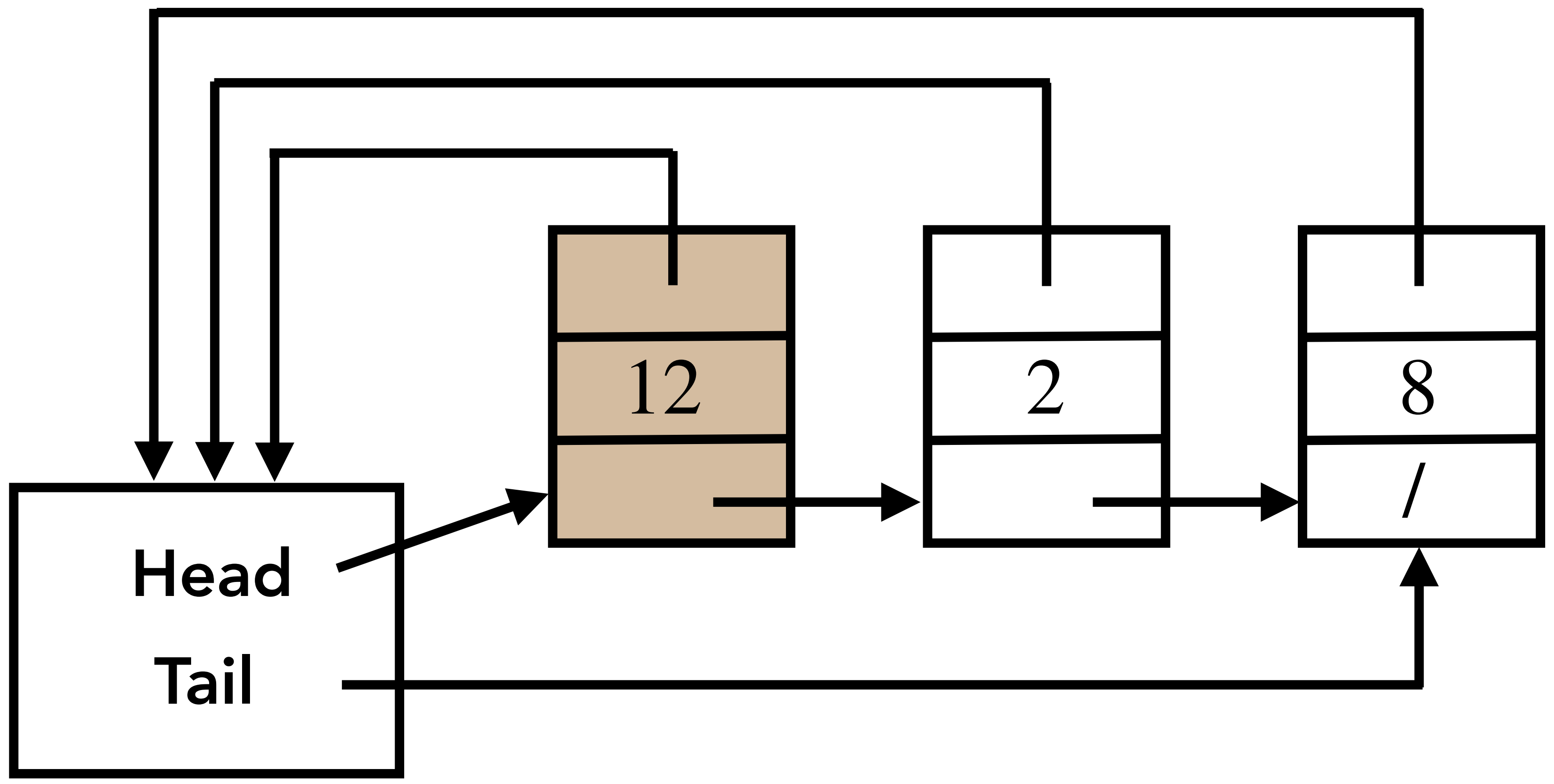


$S_1 \cup S_2$

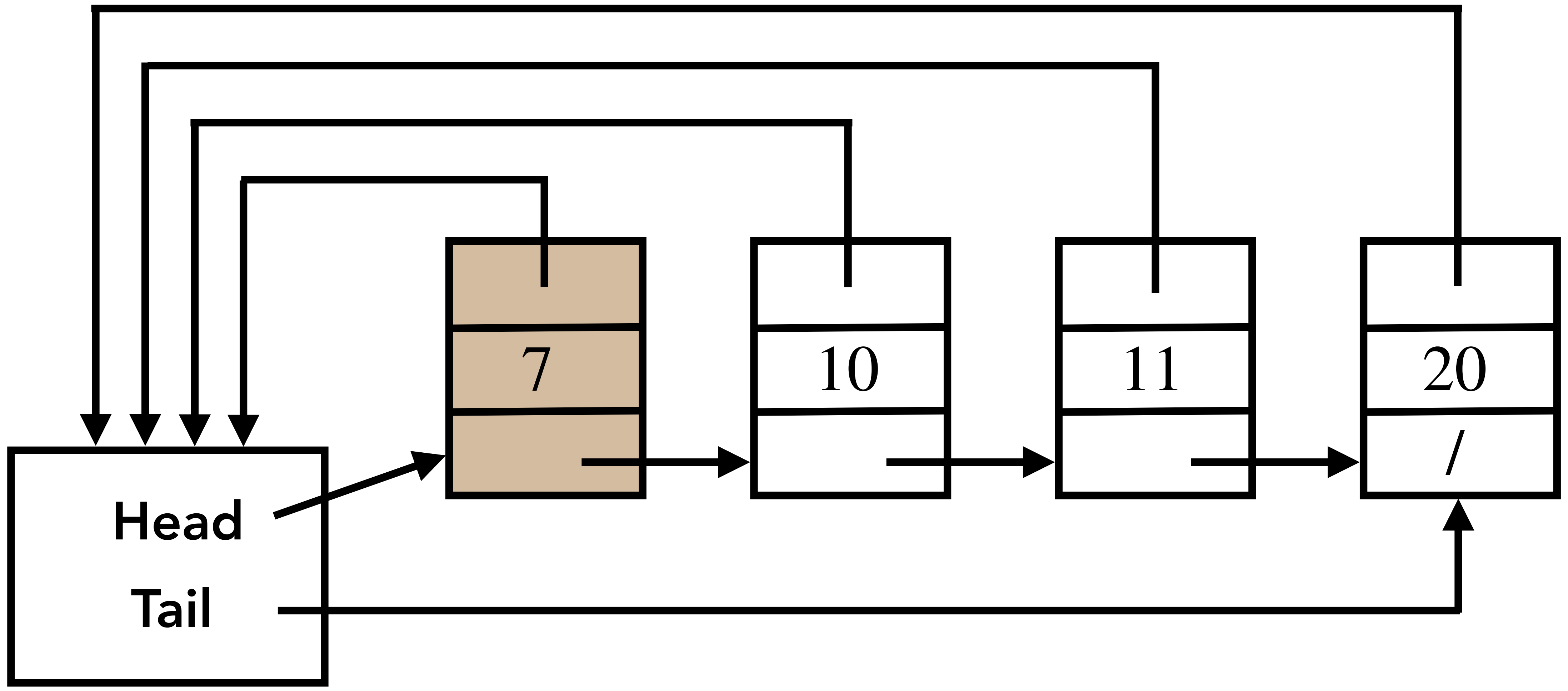


Disjoint-Sets as Linked Lists

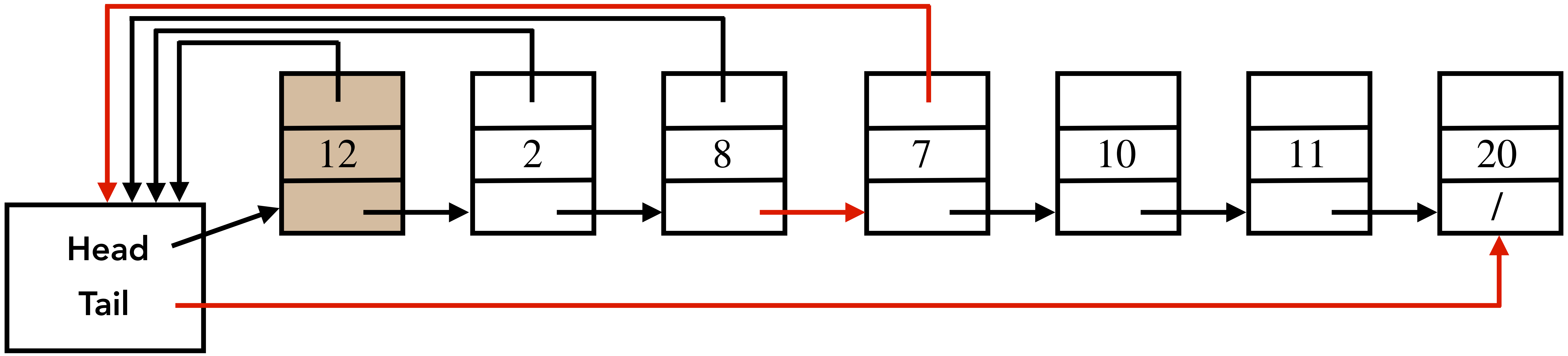
S_1



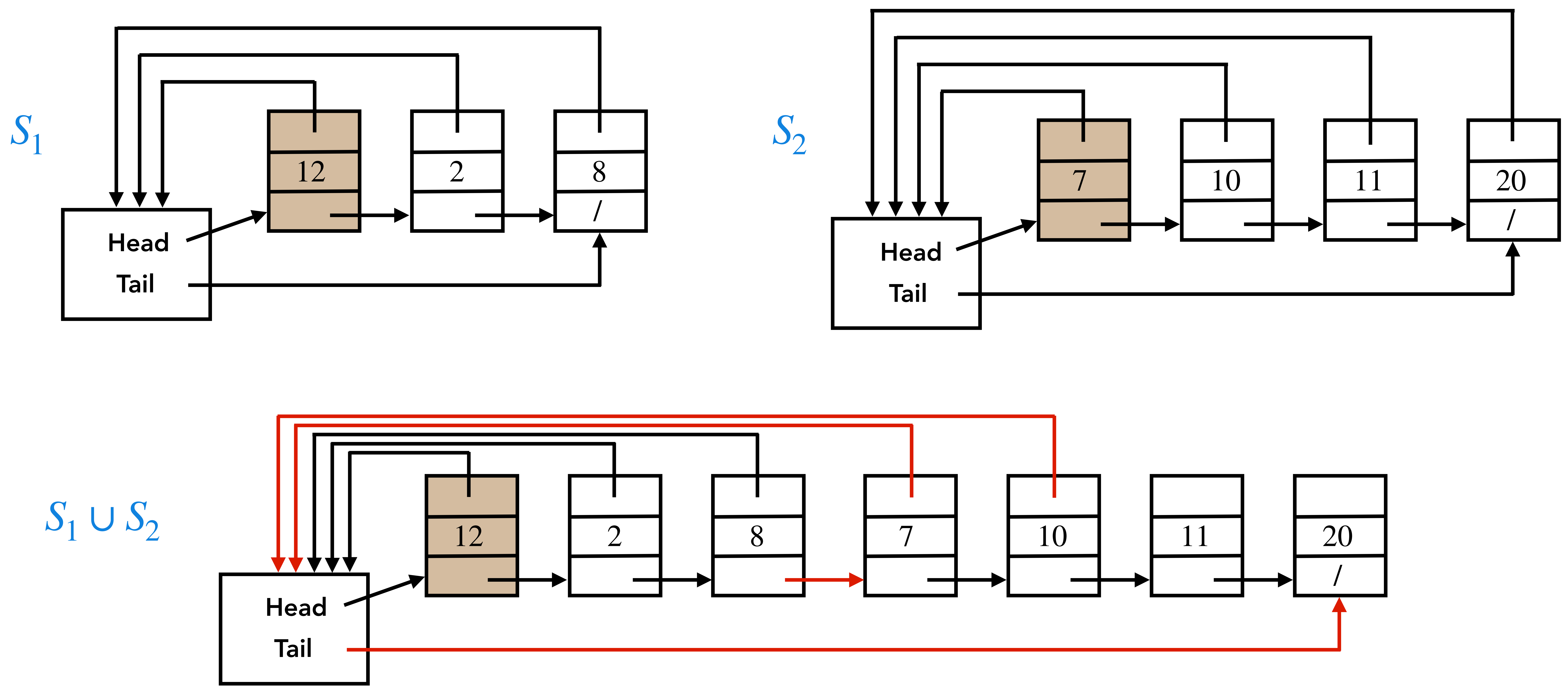
S_2



$S_1 \cup S_2$

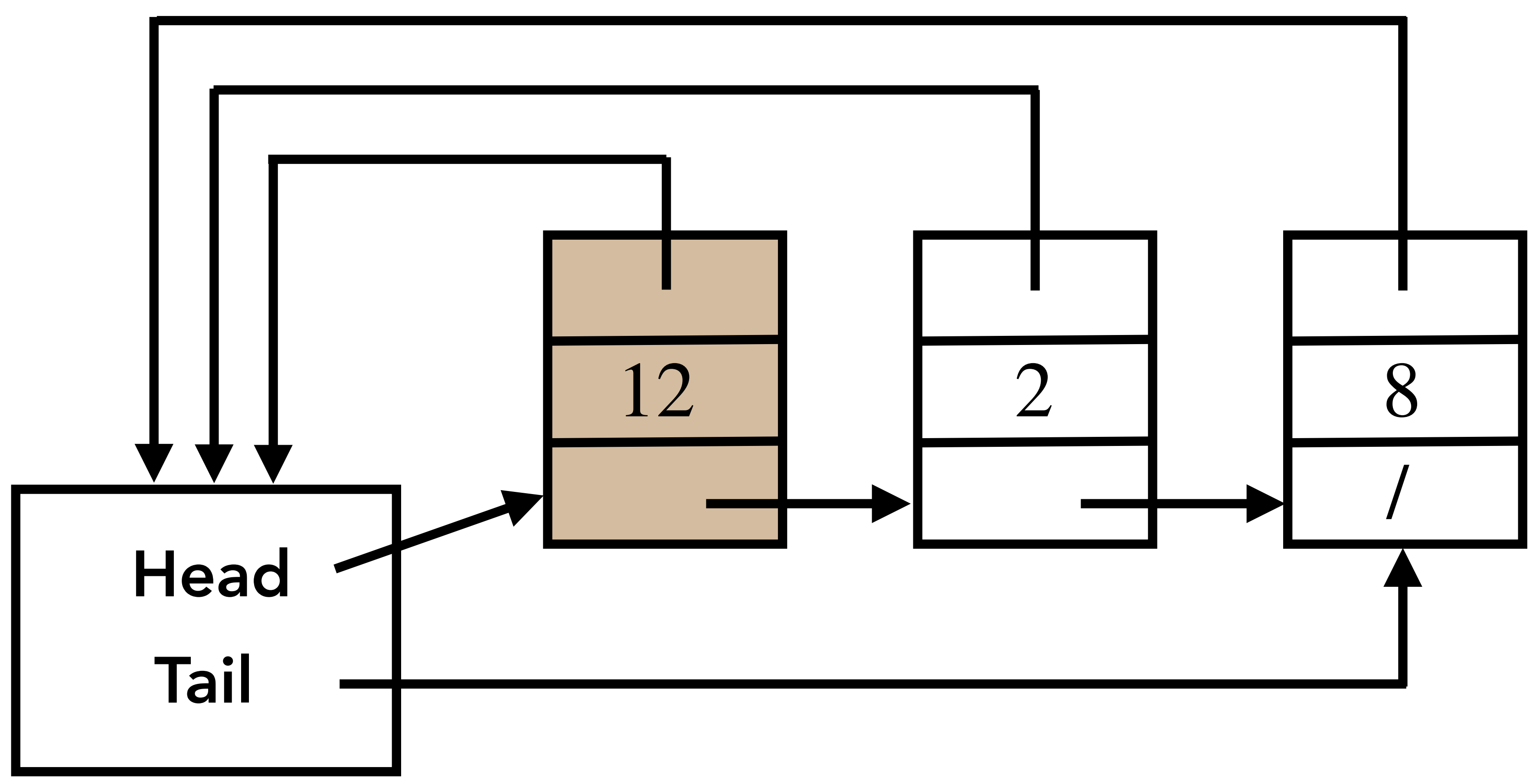


Disjoint-Sets as Linked Lists

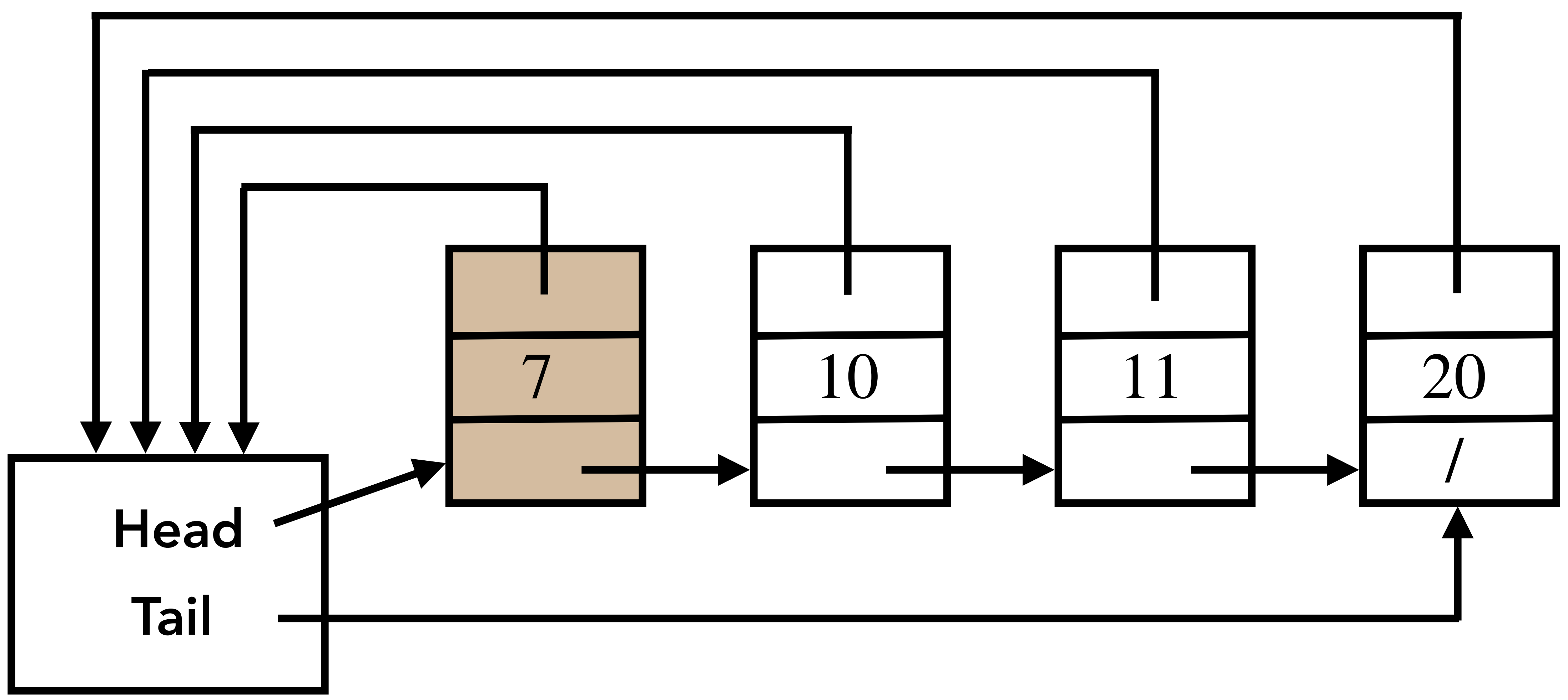


Disjoint-Sets as Linked Lists

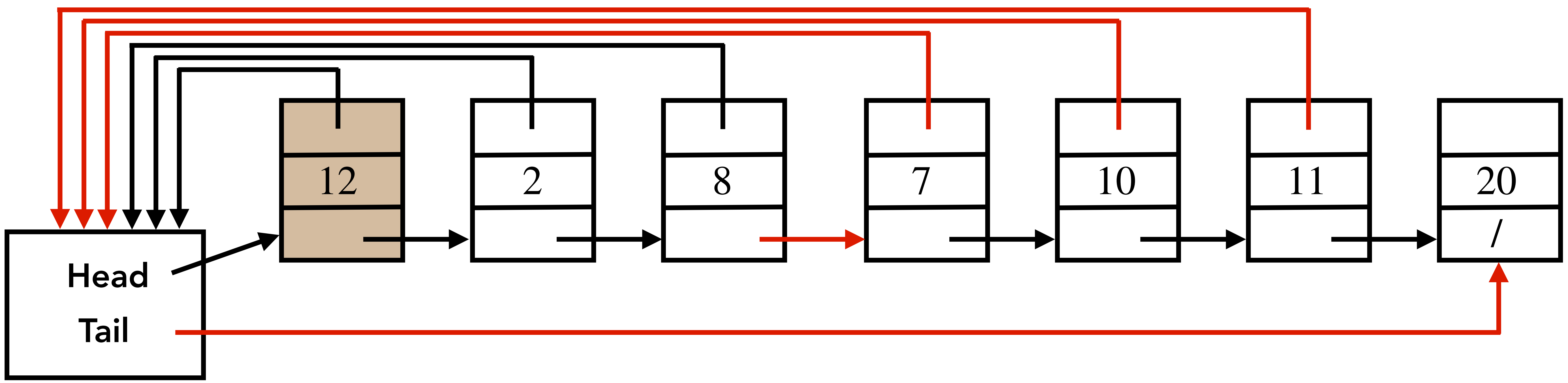
S_1



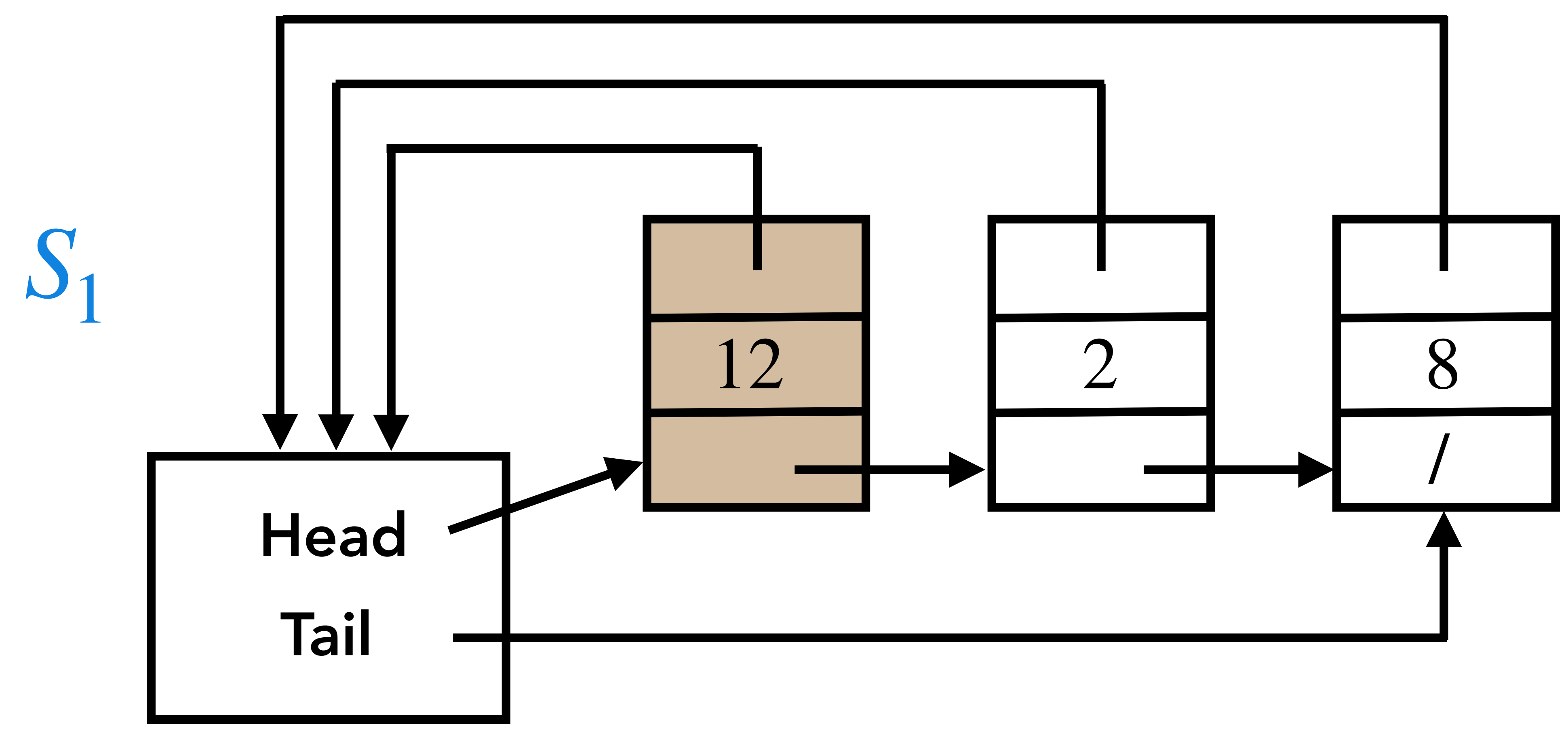
S_2

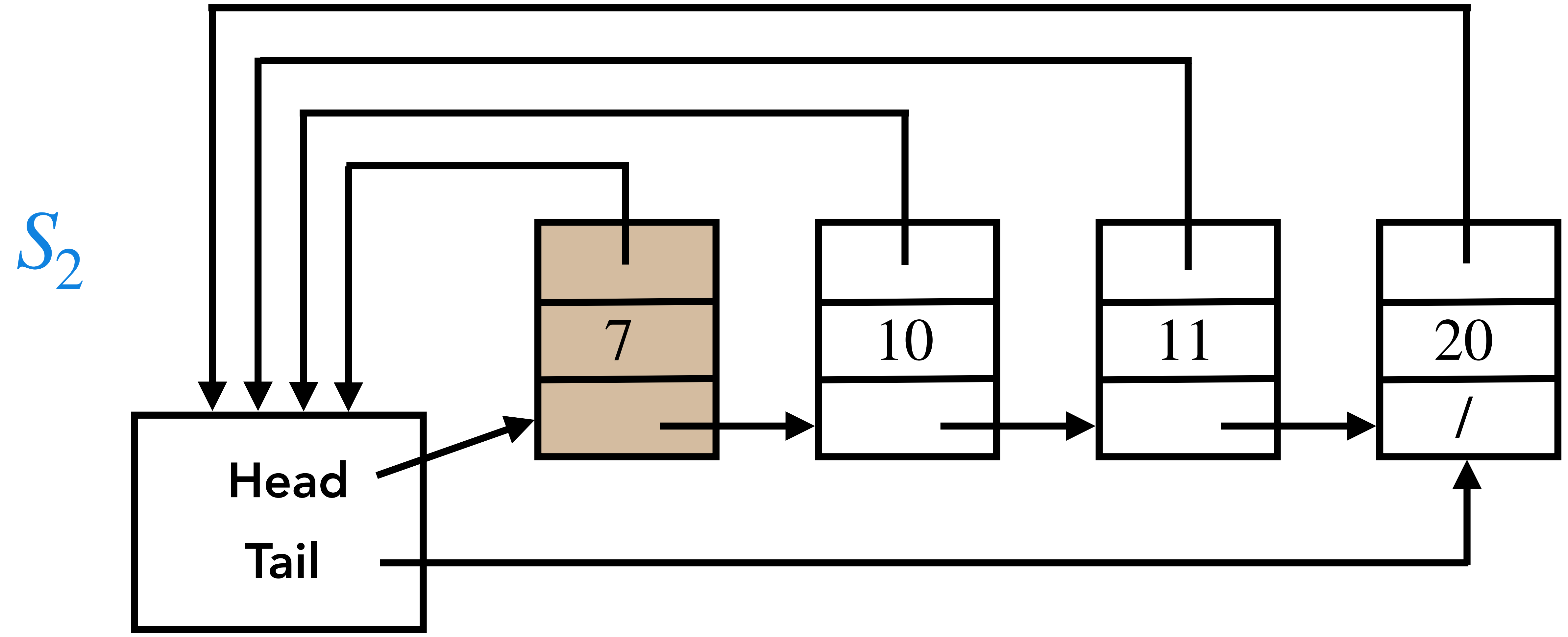


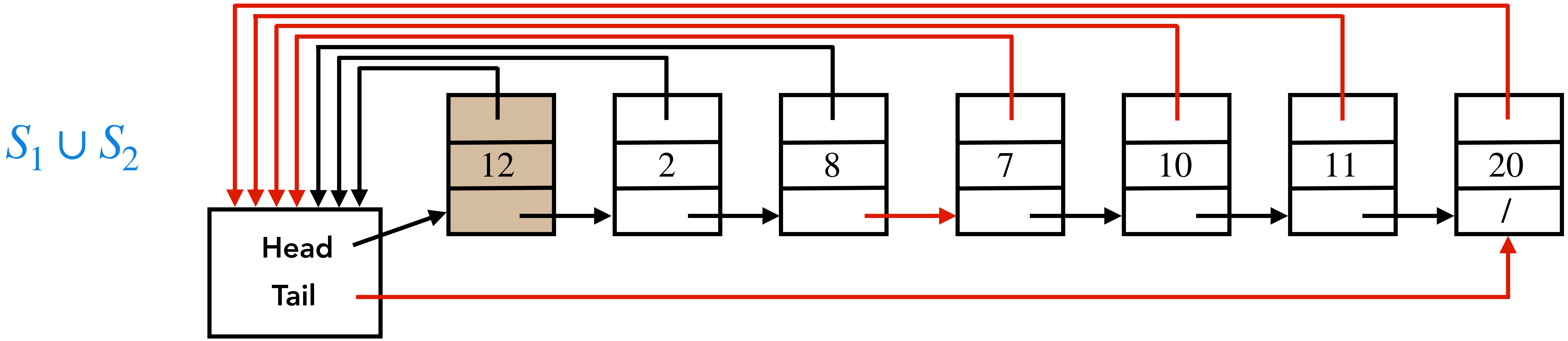
$S_1 \cup S_2$



Disjoint-Sets as Linked Lists

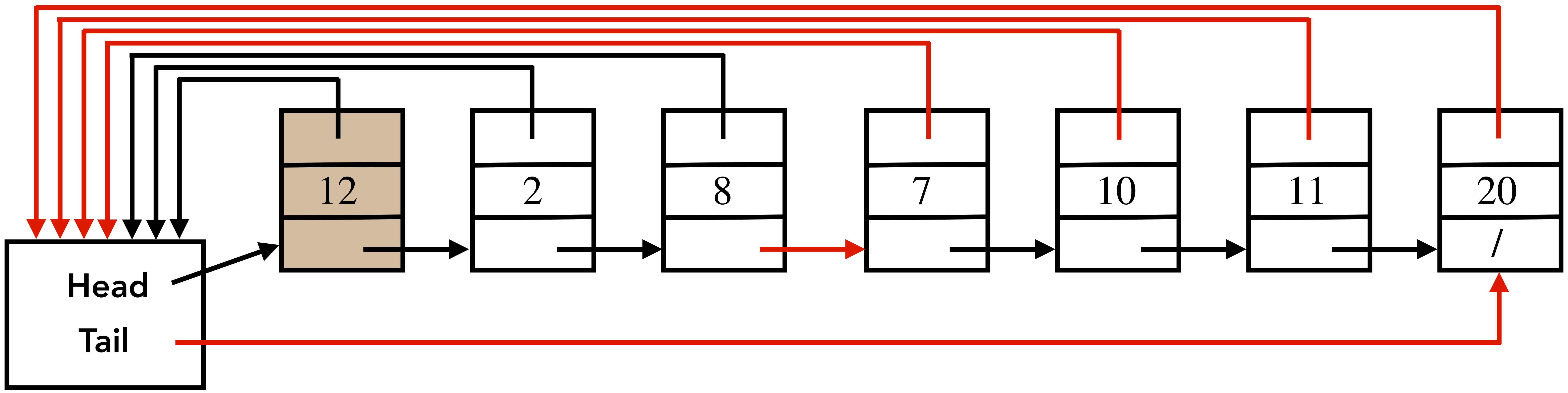






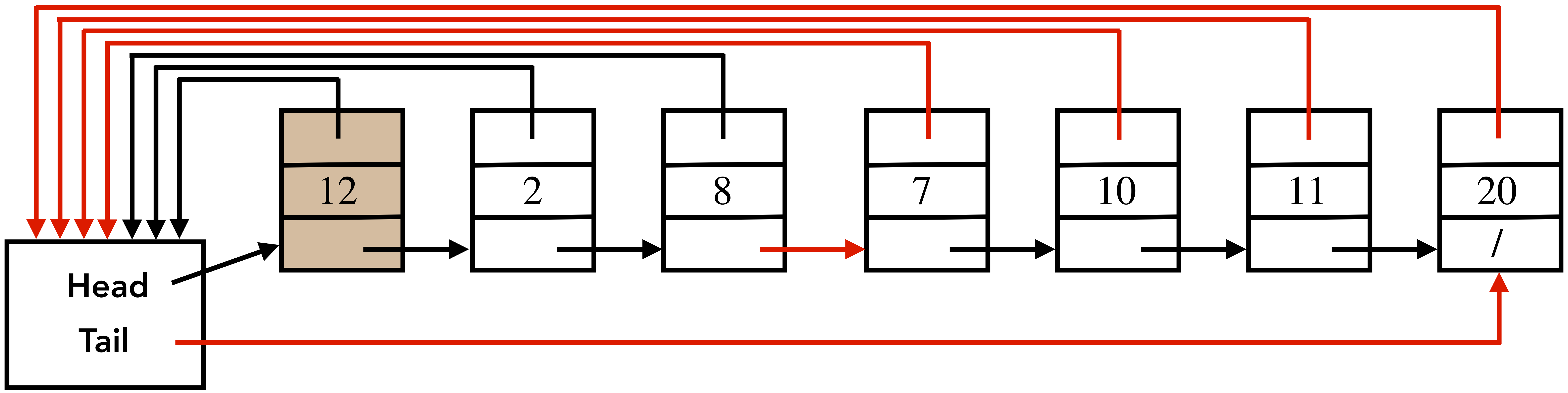
Disjoint-Sets as Linked Lists

$S_1 \cup S_2$



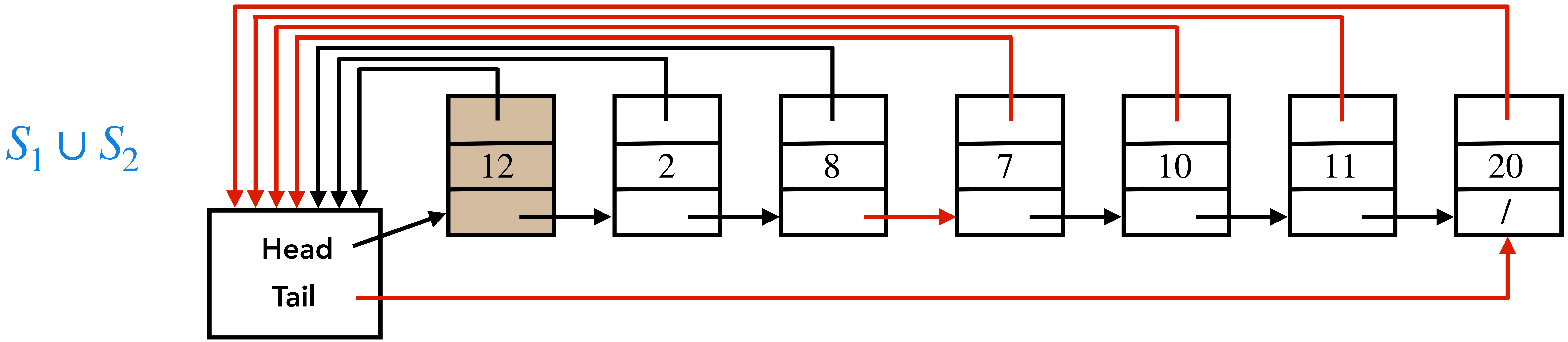
Disjoint-Sets as Linked Lists

$S_1 \cup S_2$



Heuristic: Appending the **shorter list** at the end of the **longer list**, will make **Union** faster.

Disjoint-Sets as Linked Lists

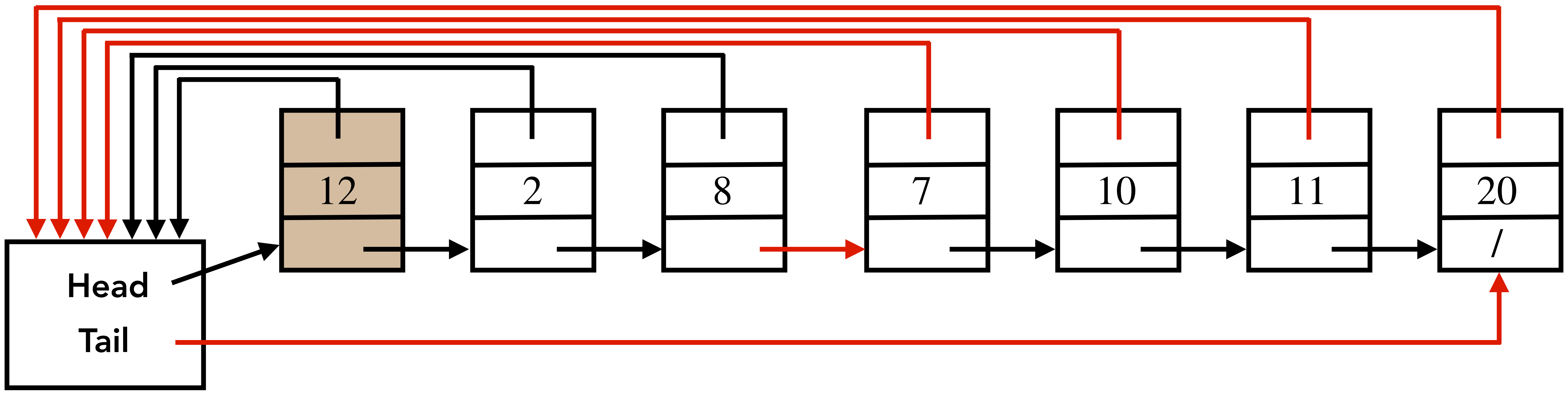


Heuristic: Appending the **shorter list** at the end of the **longer list**, will make **Union** faster.

Claim: A sequence of m **Make-Set**, **Union**, and **Find-Set** operations,

Disjoint-Sets as Linked Lists

$S_1 \cup S_2$

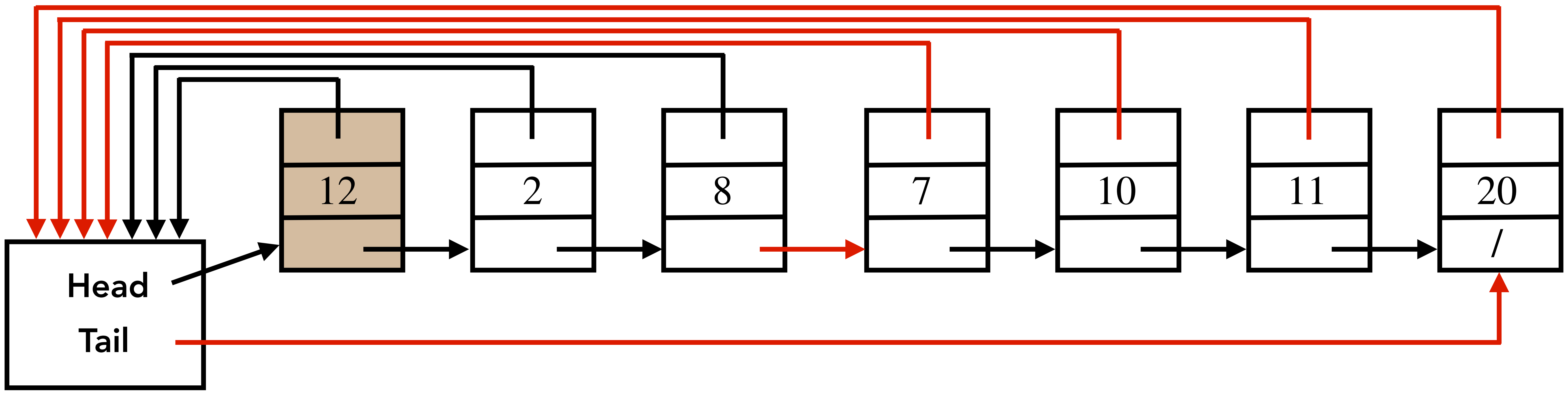


Heuristic: Appending the **shorter list** at the end of the **longer list**, will make **Union** faster.

Claim: A sequence of m **Make-Set**, **Union**, and **Find-Set** operations, first n of which are **Make-Set**

Disjoint-Sets as Linked Lists

$S_1 \cup S_2$

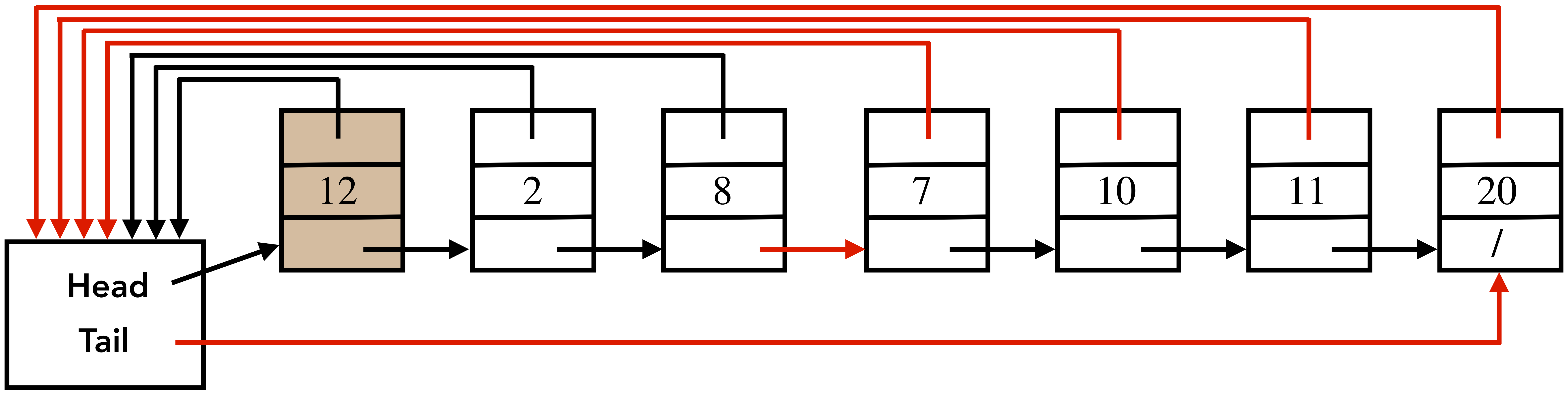


Heuristic: Appending the **shorter list** at the end of the **longer list**, will make **Union** faster.

Claim: A sequence of m **Make-Set**, **Union**, and **Find-Set** operations, first n of which are **Make-Set** operations, takes $O(m + n\log n)$ time under the above heuristic.

Disjoint-Sets as Linked Lists

$S_1 \cup S_2$



Heuristic: Appending the **shorter list** at the end of the **longer list**, will make **Union** faster.

Claim: A sequence of m **Make-Set**, **Union**, and **Find-Set** operations, first n of which are **Make-Set** operations, takes $O(m + n \log n)$ time under the above heuristic.

Proof: DIY.

Disjoint-Sets as Trees

Disjoint-Sets as Trees

Idea: We maintain the dynamic disjoint sets in the following way:

Disjoint-Sets as Trees

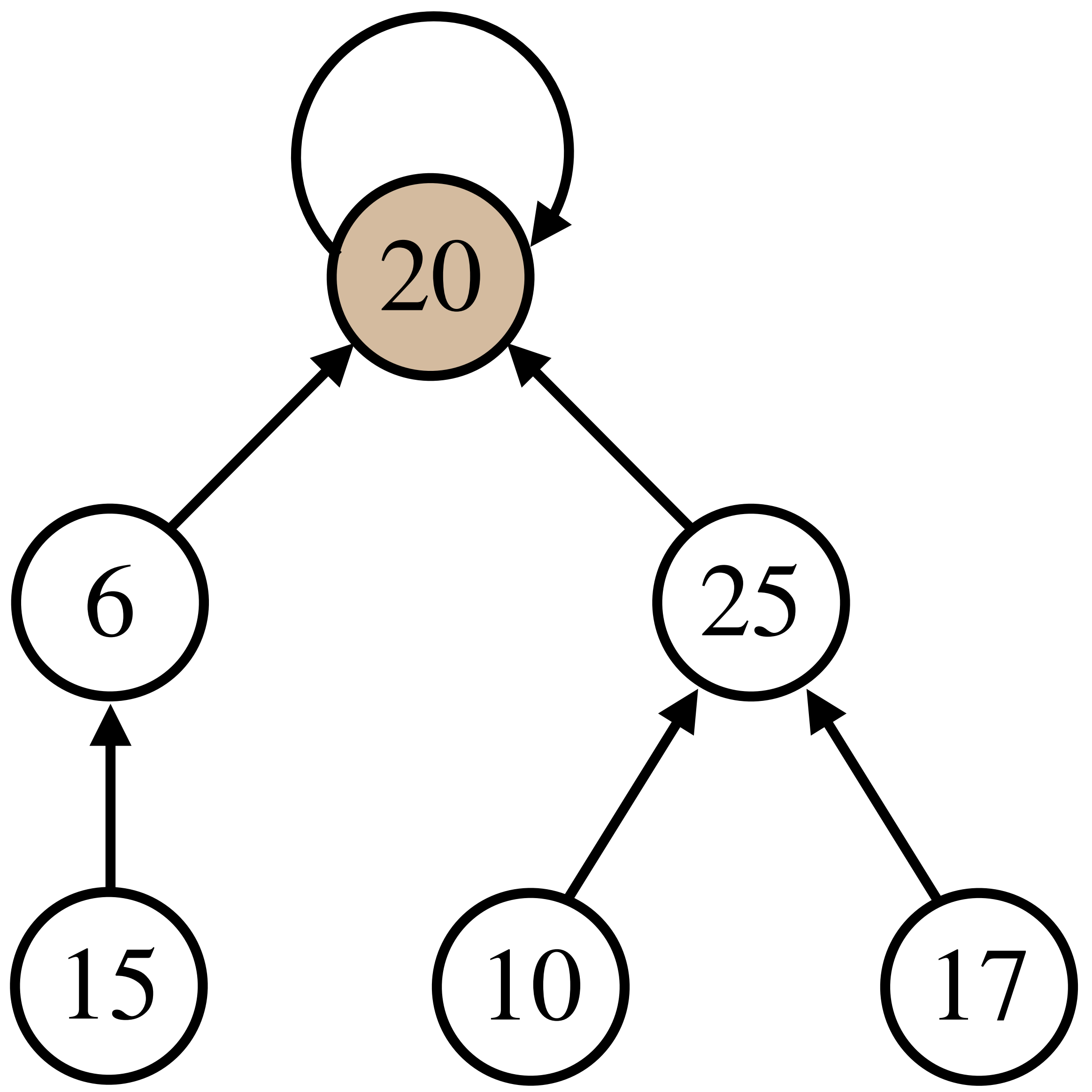
Idea: We maintain the dynamic disjoint sets in the following way:

- Keep sets as rooted trees.

Disjoint-Sets as Trees

Idea: We maintain the dynamic disjoint sets in the following way:

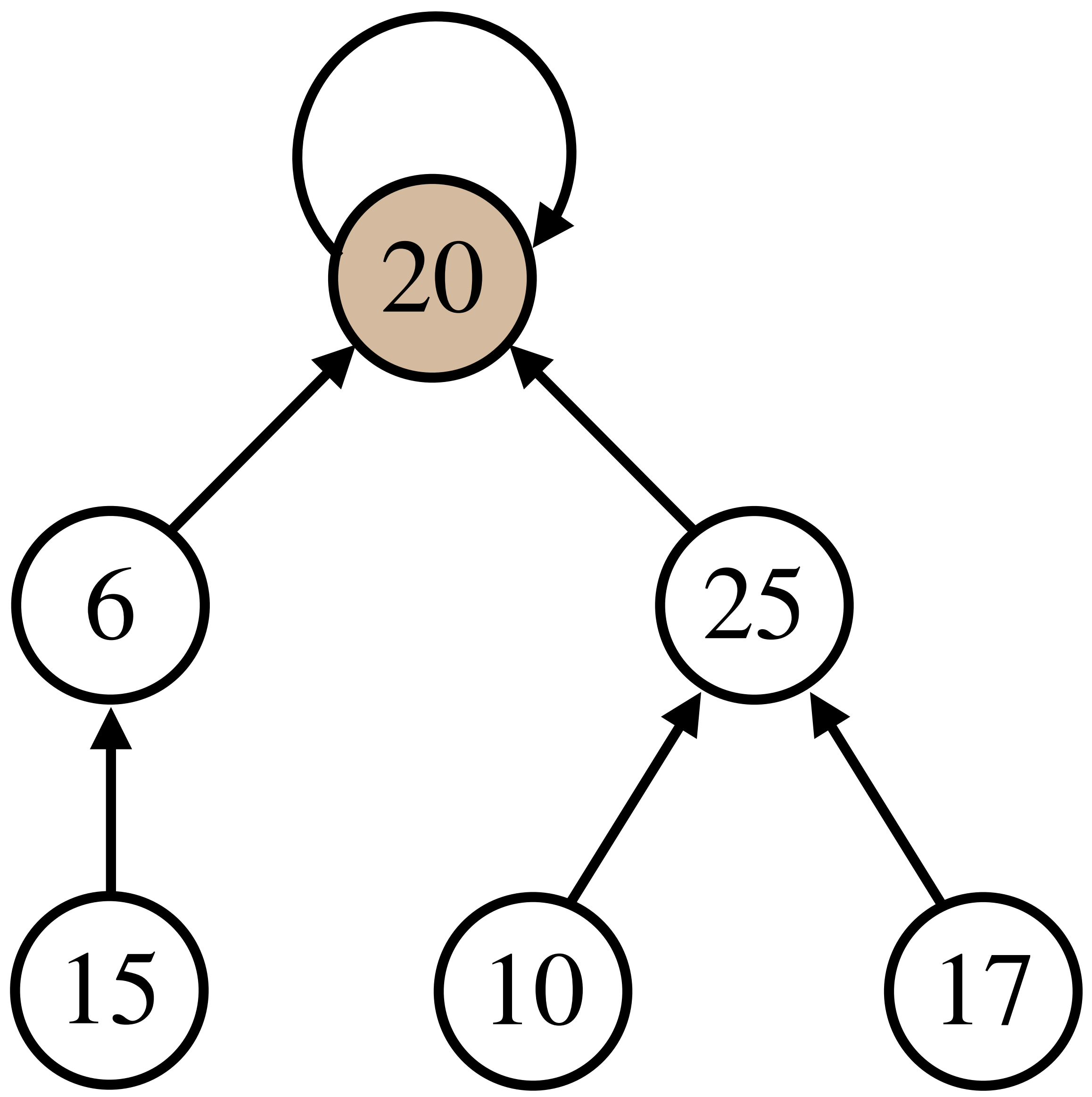
- Keep sets as rooted trees.



Disjoint-Sets as Trees

Idea: We maintain the dynamic disjoint sets in the following way:

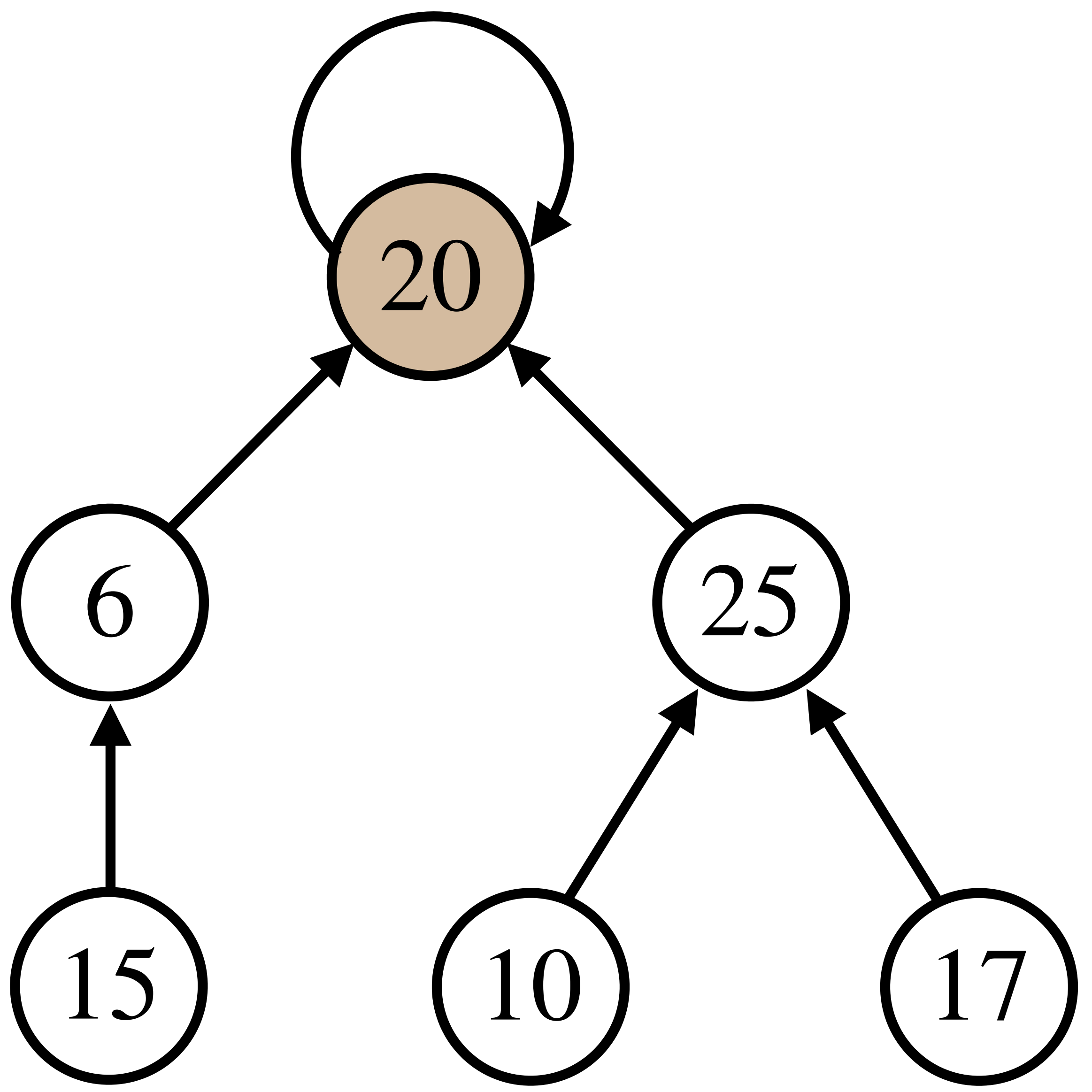
- Keep sets as rooted trees.
- Each node points to its parent.



Disjoint-Sets as Trees

Idea: We maintain the dynamic disjoint sets in the following way:

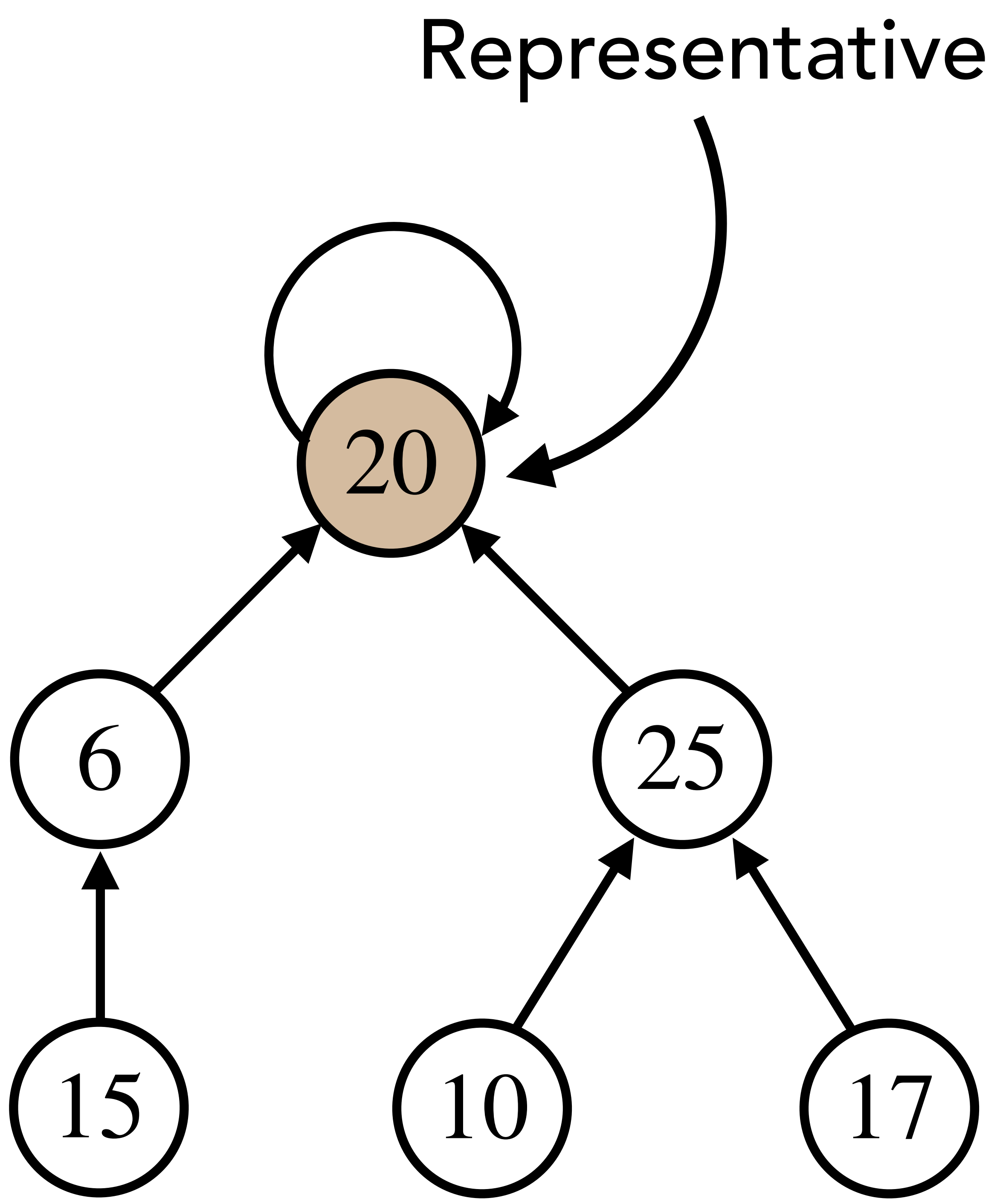
- Keep sets as rooted trees.
- Each node points to its parent.
- Root is its own parent and the representative of the set.



Disjoint-Sets as Trees

Idea: We maintain the dynamic disjoint sets in the following way:

- Keep sets as rooted trees.
- Each node points to its parent.
- Root is its own parent and the representative of the set.



Height in Disjoint-Sets as Trees

Height in Disjoint-Sets as Trees

Defn: Height of a node X in a disjoint set represented as a tree is the number of edges in the

Height in Disjoint-Sets as Trees

Defn: Height of a node X in a disjoint set represented as a tree is the number of edges in the longest path from a descendant leaf to x.

Height in Disjoint-Sets as Trees

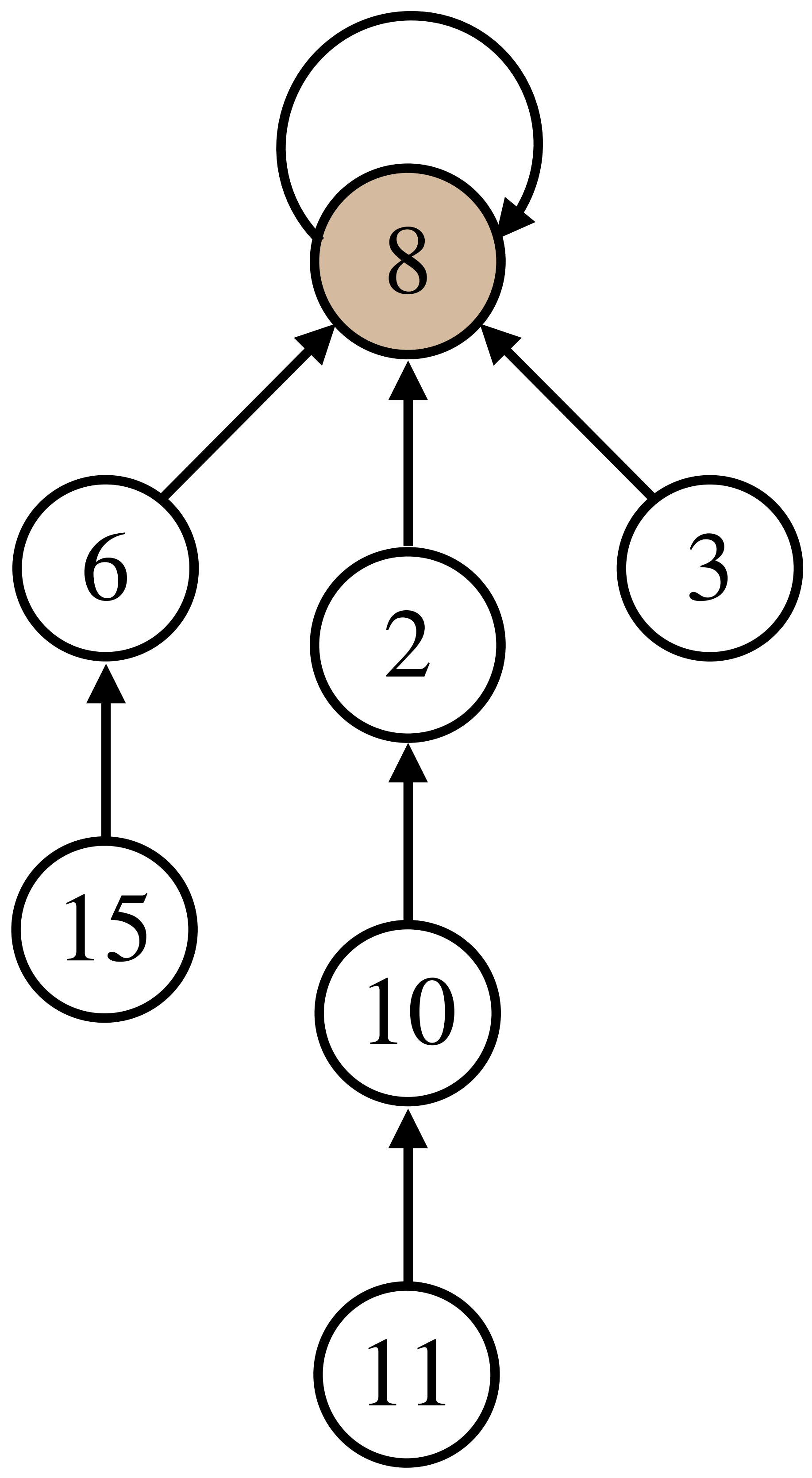
Defn: Height of a node X in a disjoint set represented as a tree is the number of edges in the longest path from a descendant leaf to x.

Example:

Height in Disjoint-Sets as Trees

Defn: Height of a node X in a disjoint set represented as a tree is the number of edges in the longest path from a descendant leaf to x.

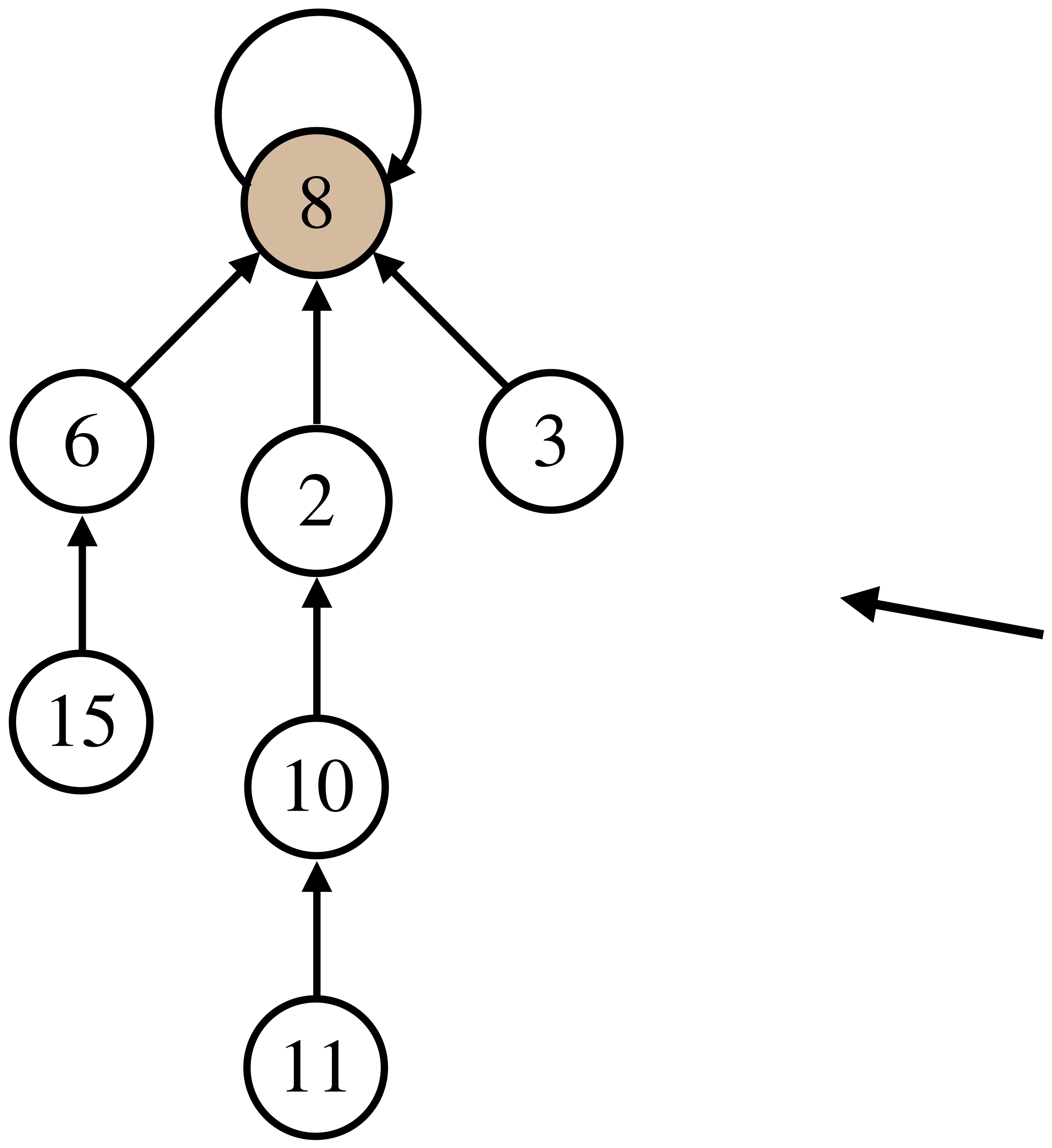
Example:



Height in Disjoint-Sets as Trees

Defn: Height of a node X in a disjoint set represented as a tree is the number of edges in the longest path from a descendant leaf to x.

Example:

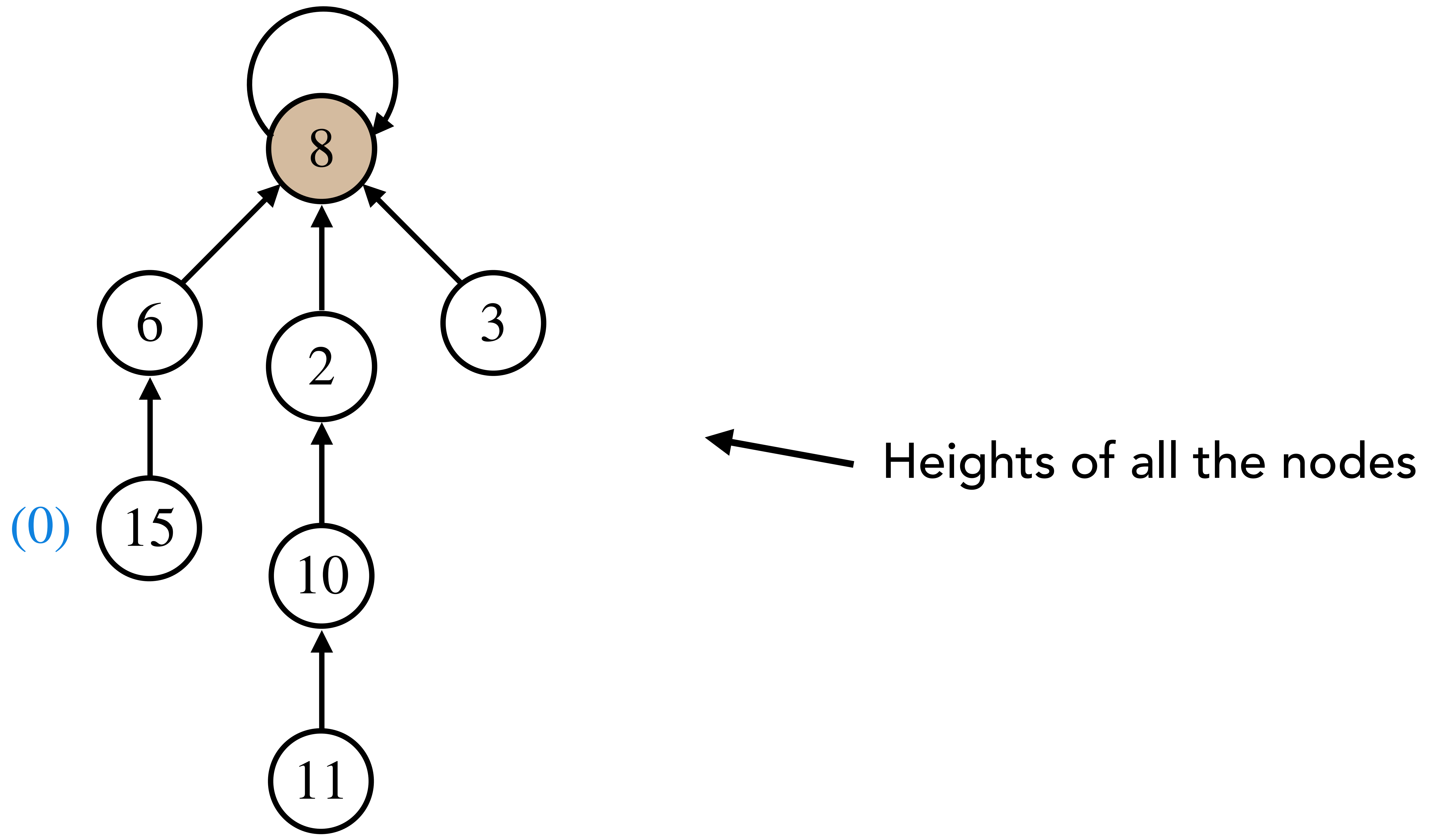


Heights of all the nodes

Height in Disjoint-Sets as Trees

Defn: Height of a node X in a disjoint set represented as a tree is the number of edges in the longest path from a descendant leaf to x.

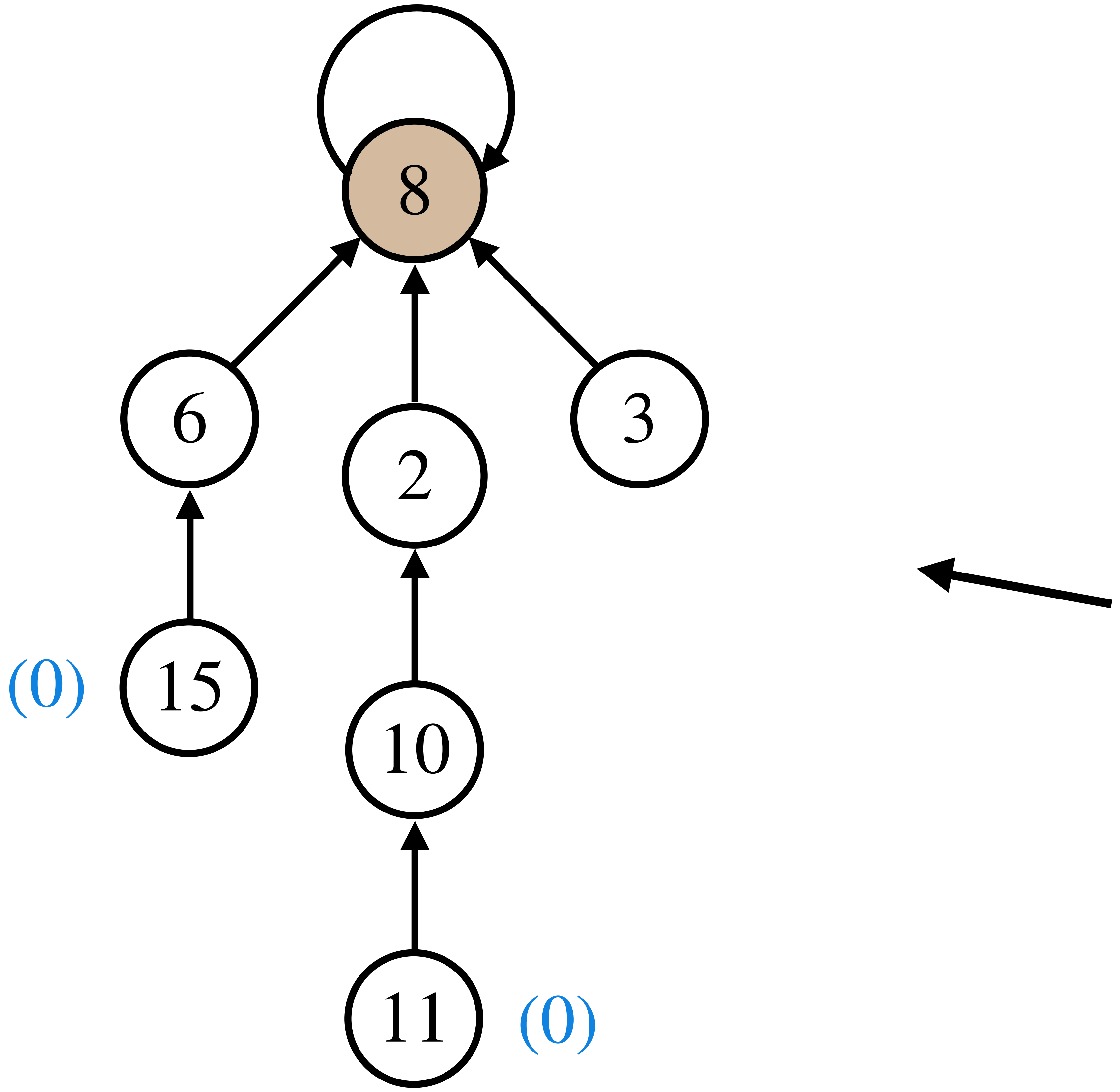
Example:



Height in Disjoint-Sets as Trees

Defn: Height of a node X in a disjoint set represented as a tree is the number of edges in the longest path from a descendant leaf to x.

Example:

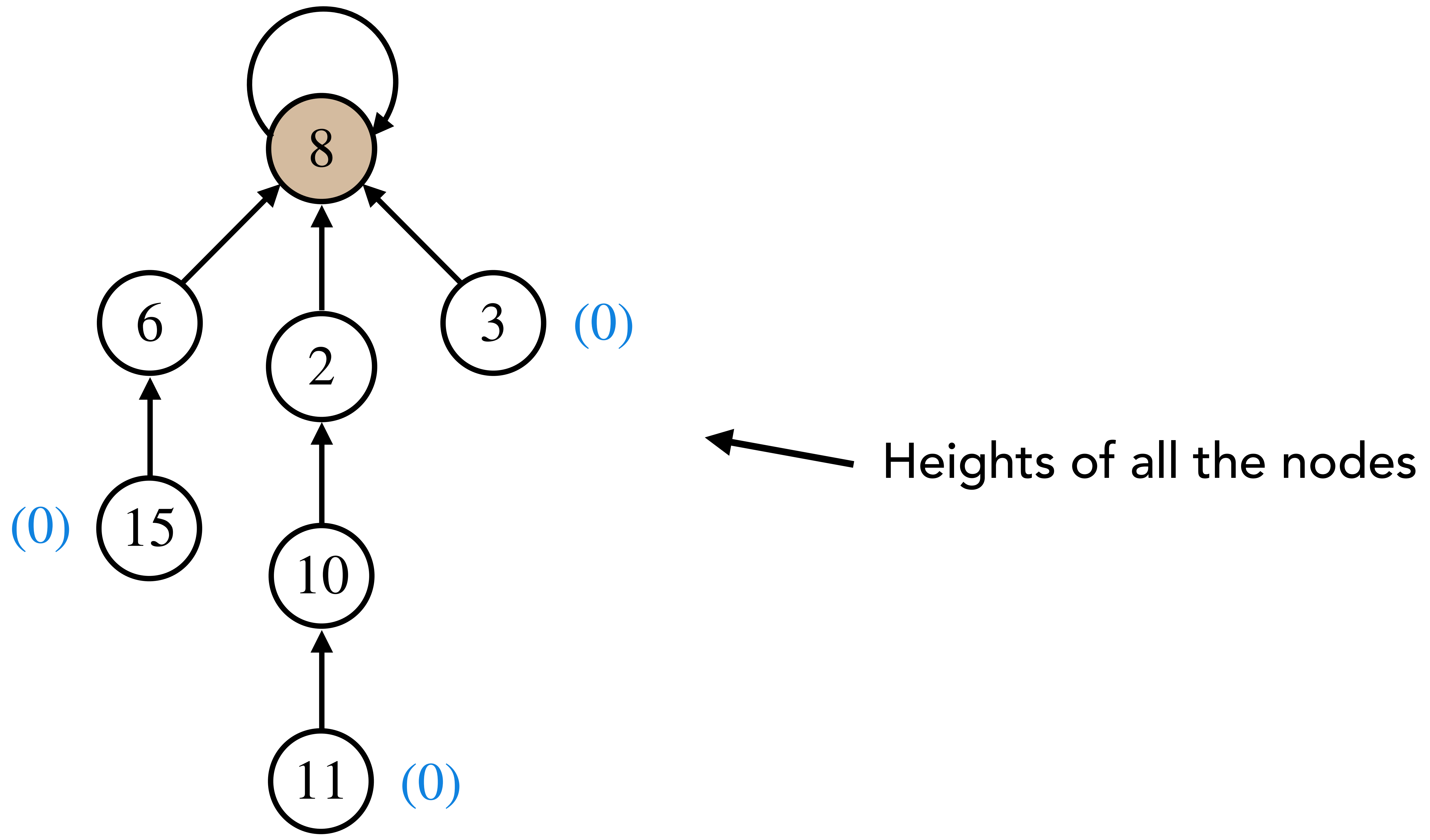


Heights of all the nodes

Height in Disjoint-Sets as Trees

Defn: Height of a node X in a disjoint set represented as a tree is the number of edges in the longest path from a descendant leaf to x.

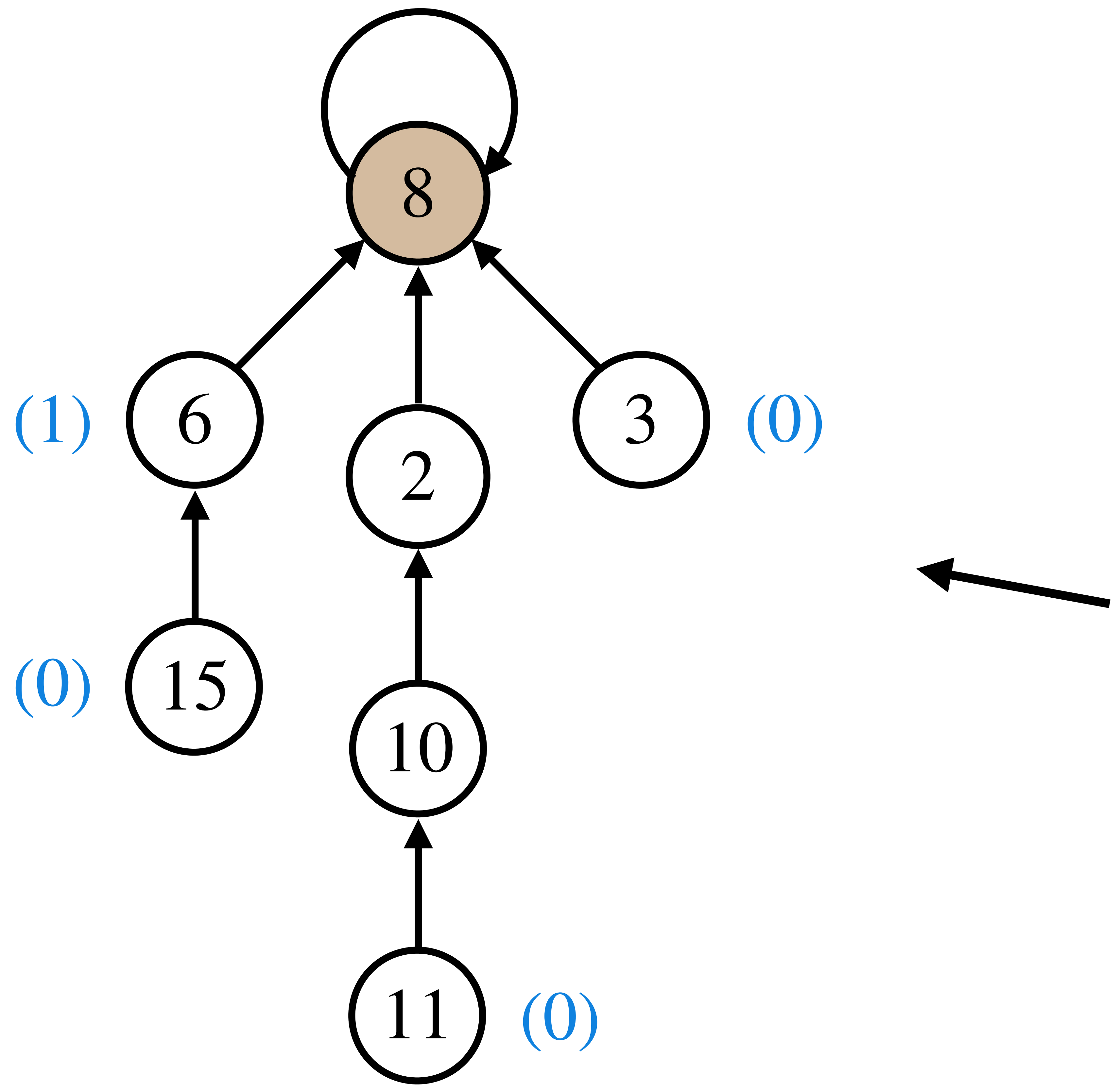
Example:



Height in Disjoint-Sets as Trees

Defn: Height of a node X in a disjoint set represented as a tree is the number of edges in the longest path from a descendant leaf to x.

Example:

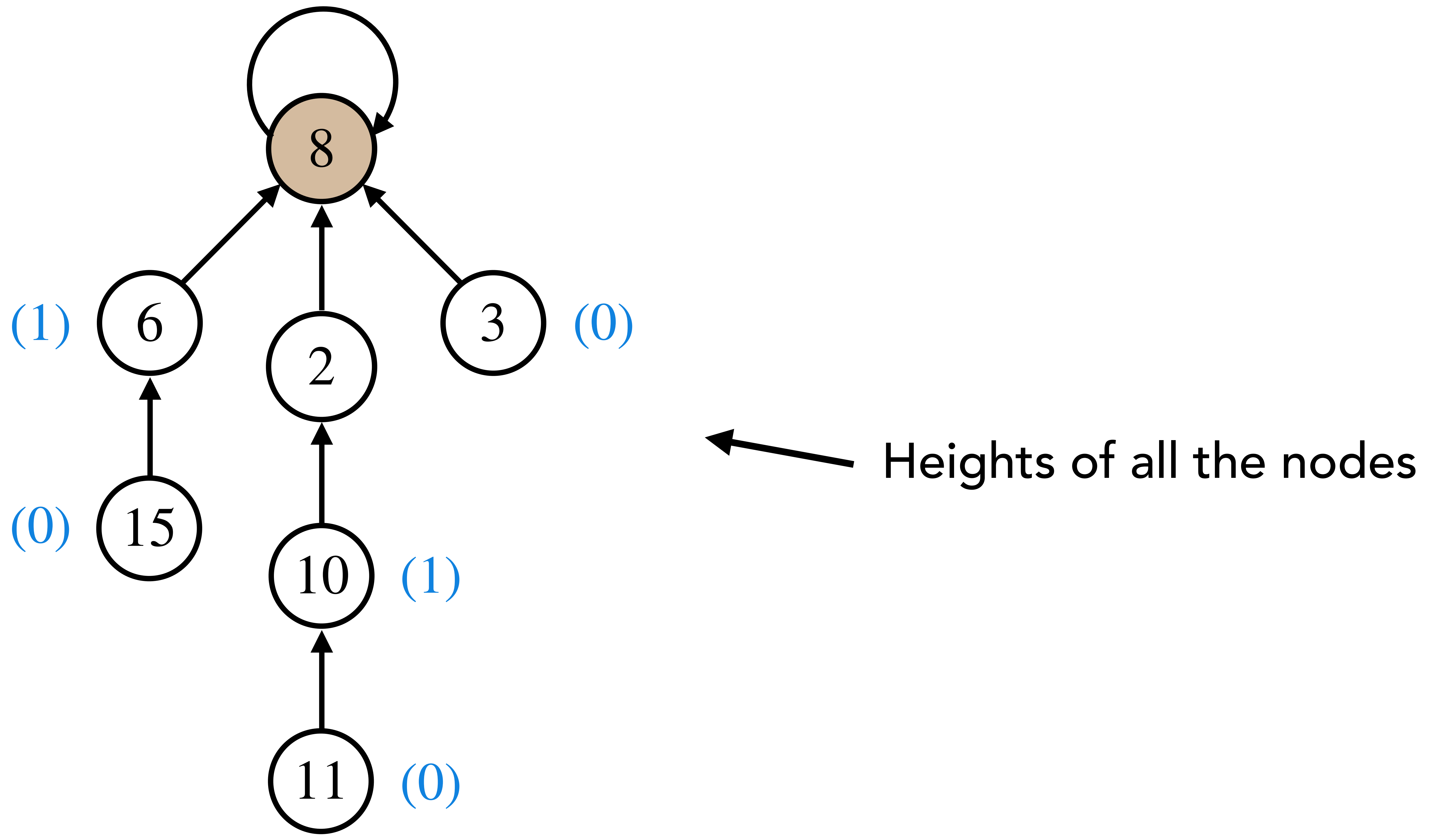


Heights of all the nodes

Height in Disjoint-Sets as Trees

Defn: Height of a node X in a disjoint set represented as a tree is the number of edges in the longest path from a descendant leaf to x.

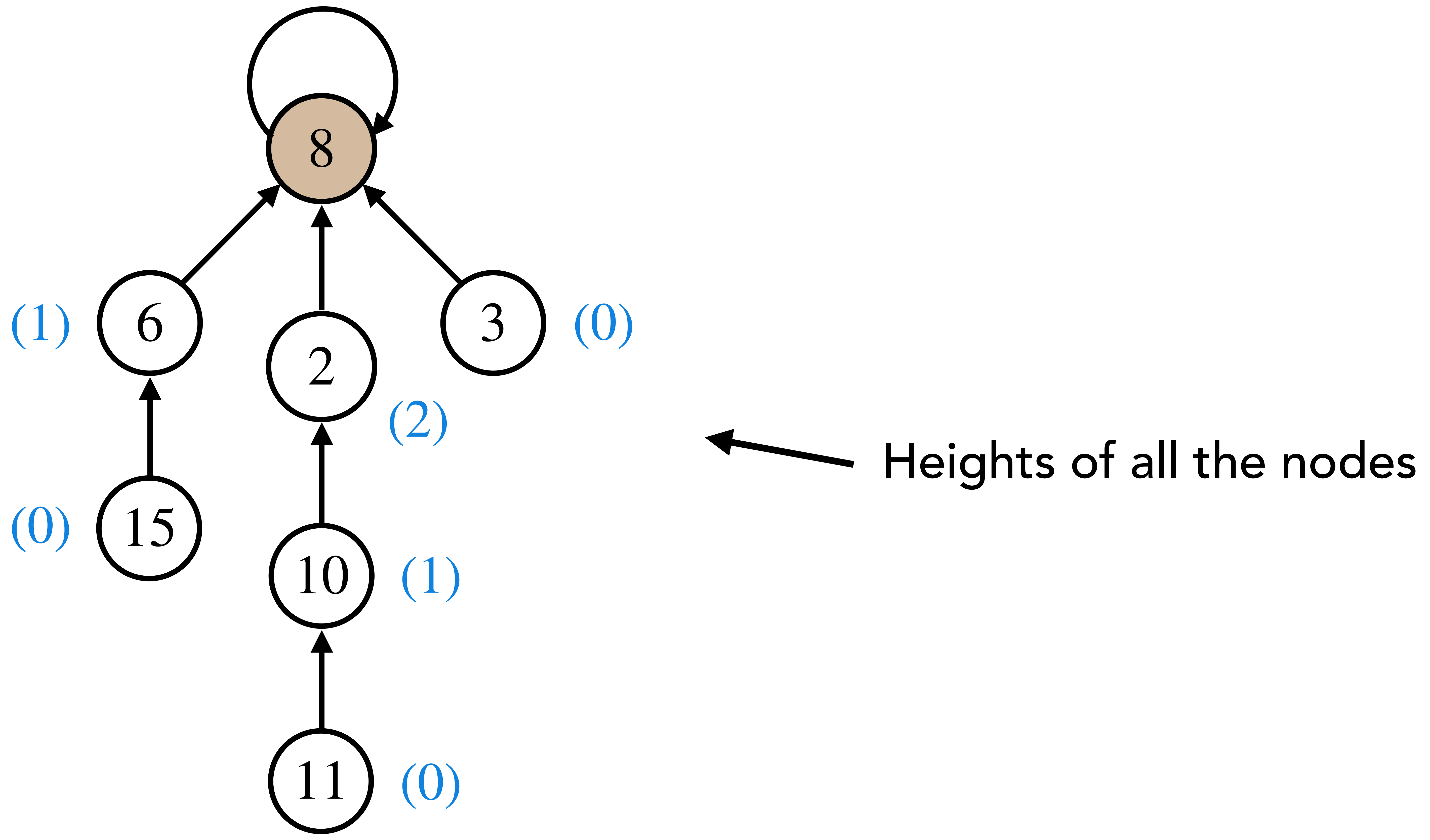
Example:



Height in Disjoint-Sets as Trees

Defn: Height of a node X in a disjoint set represented as a tree is the number of edges in the longest path from a descendant leaf to x.

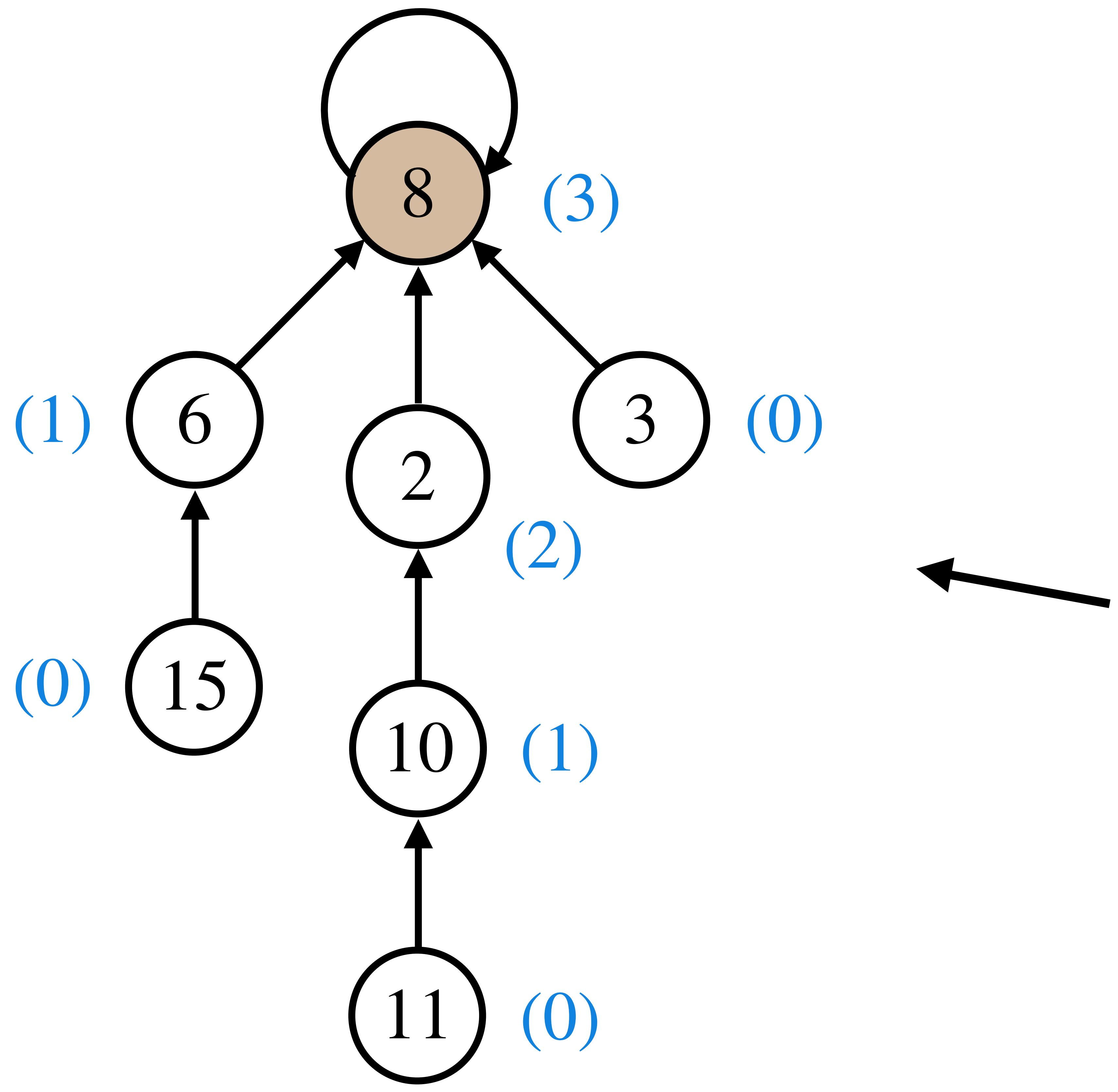
Example:



Height in Disjoint-Sets as Trees

Defn: Height of a node X in a disjoint set represented as a tree is the number of edges in the longest path from a descendant leaf to x.

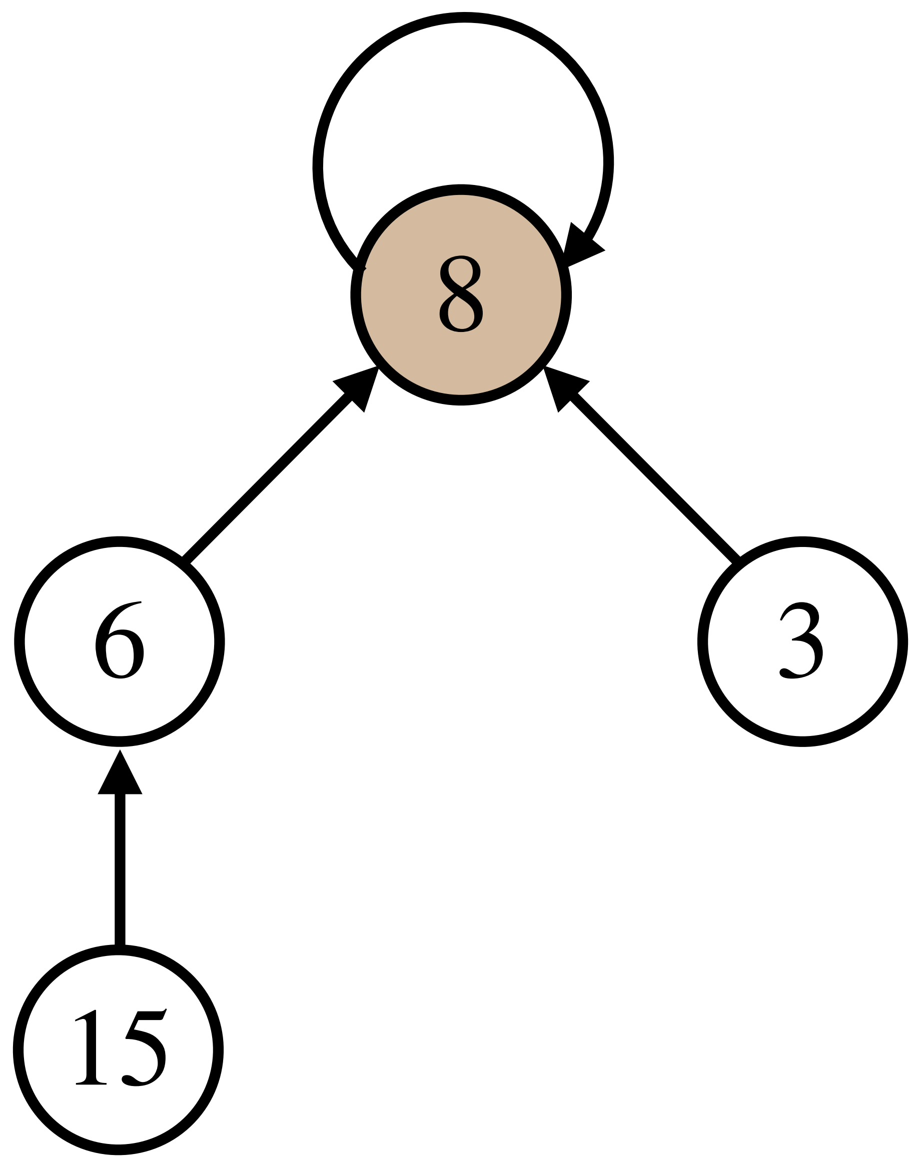
Example:



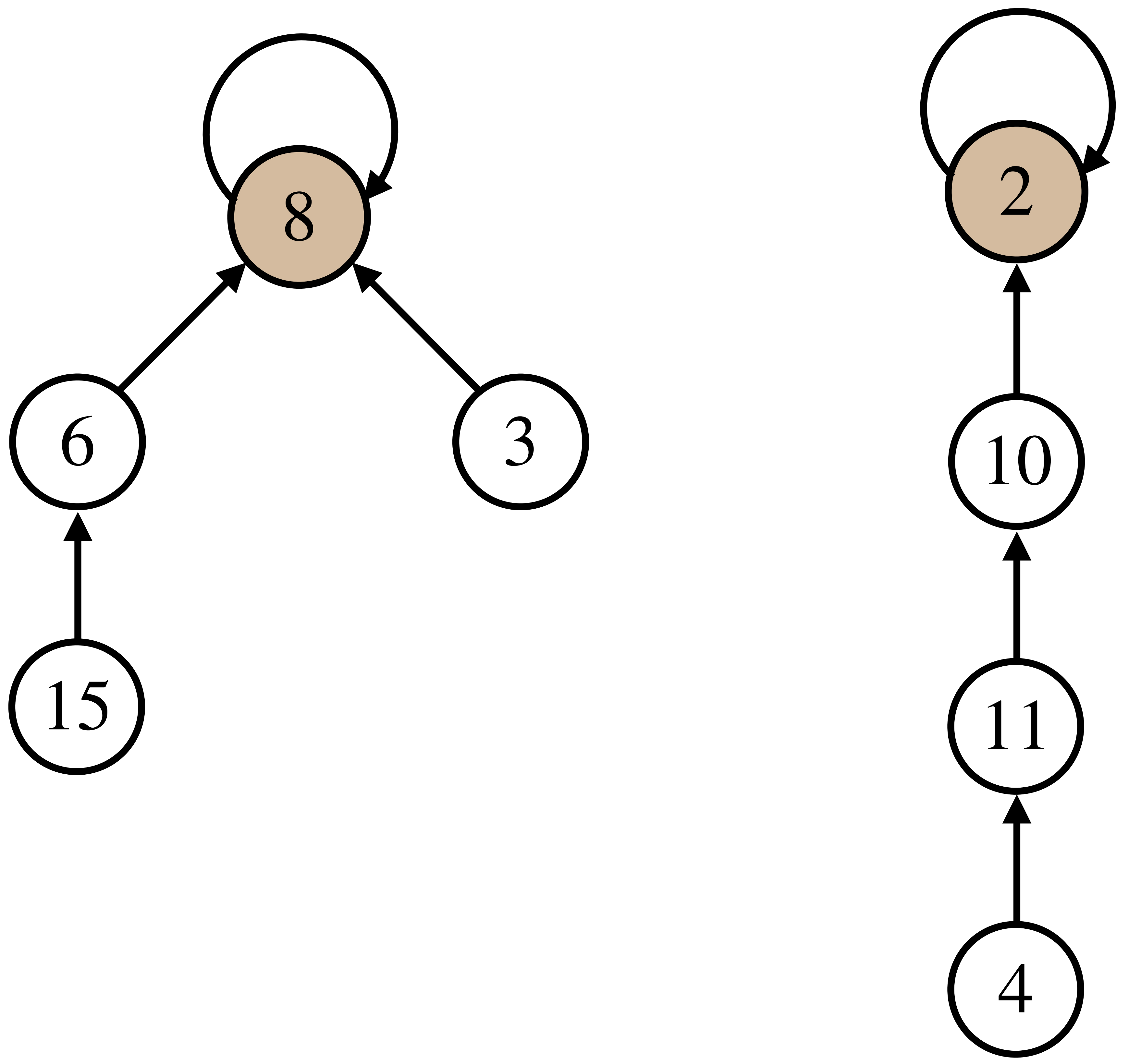
Heights of all the nodes

Union on Disjoint-Sets as Trees

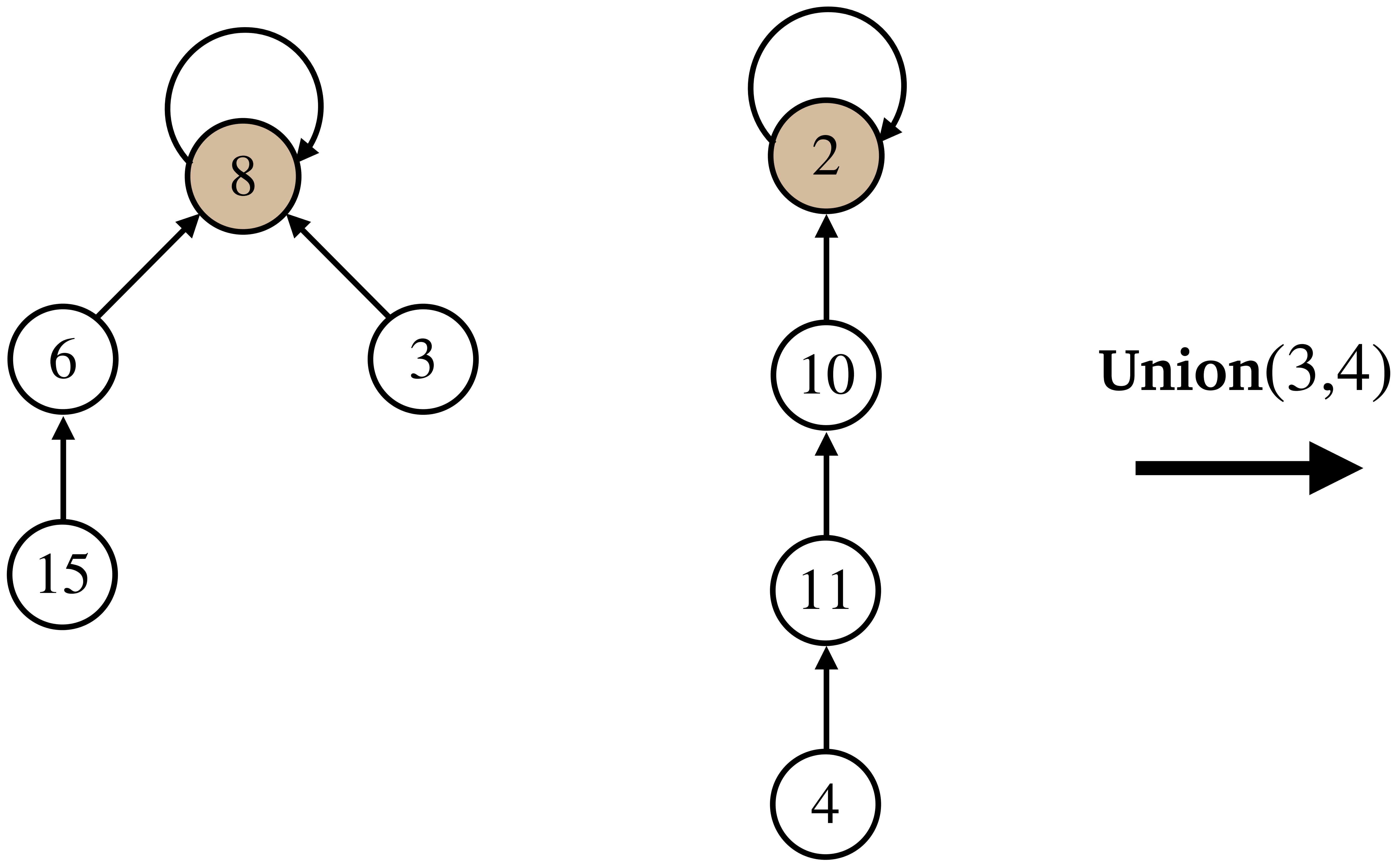
Union on Disjoint-Sets as Trees



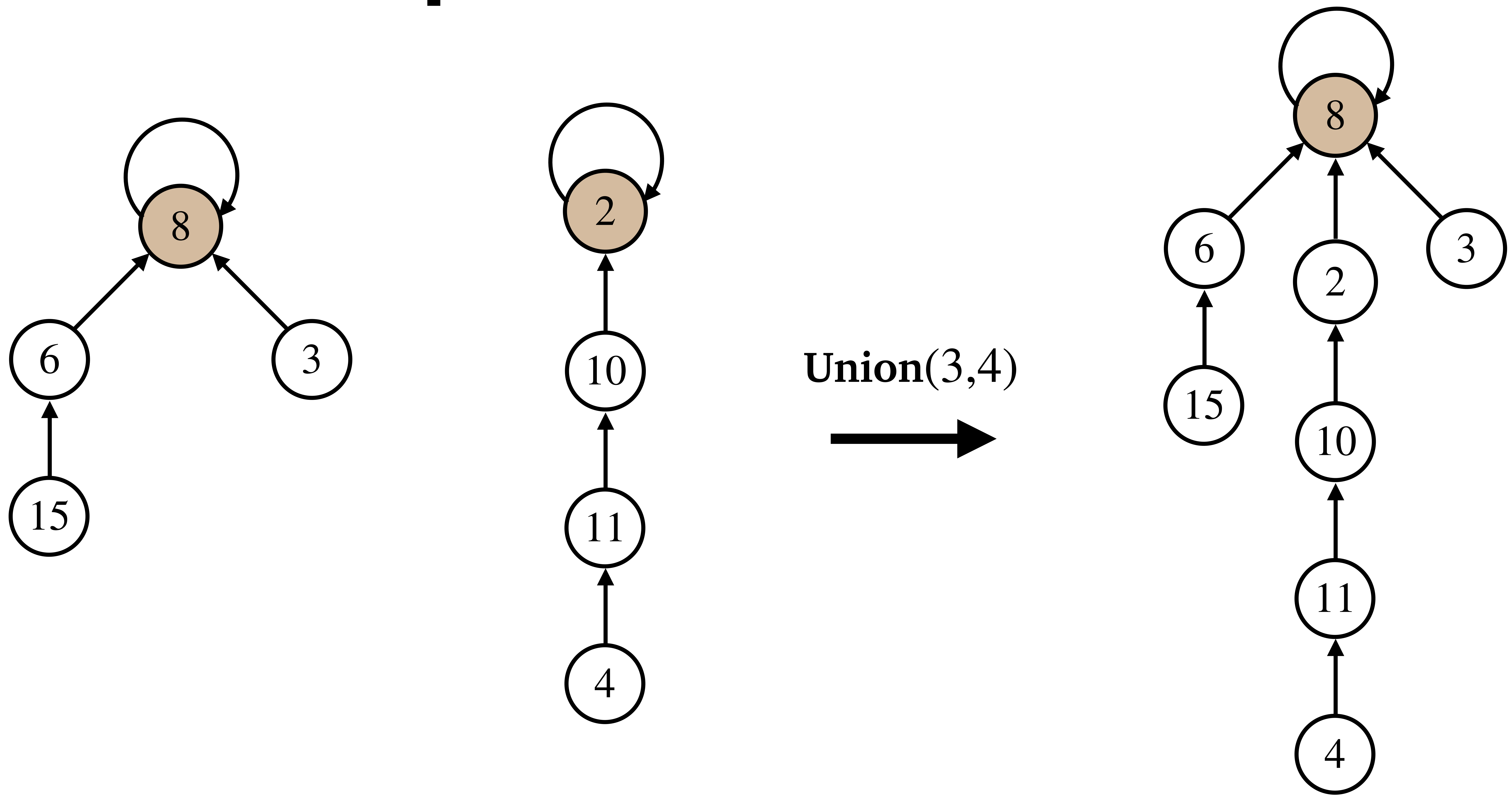
Union on Disjoint-Sets as Trees



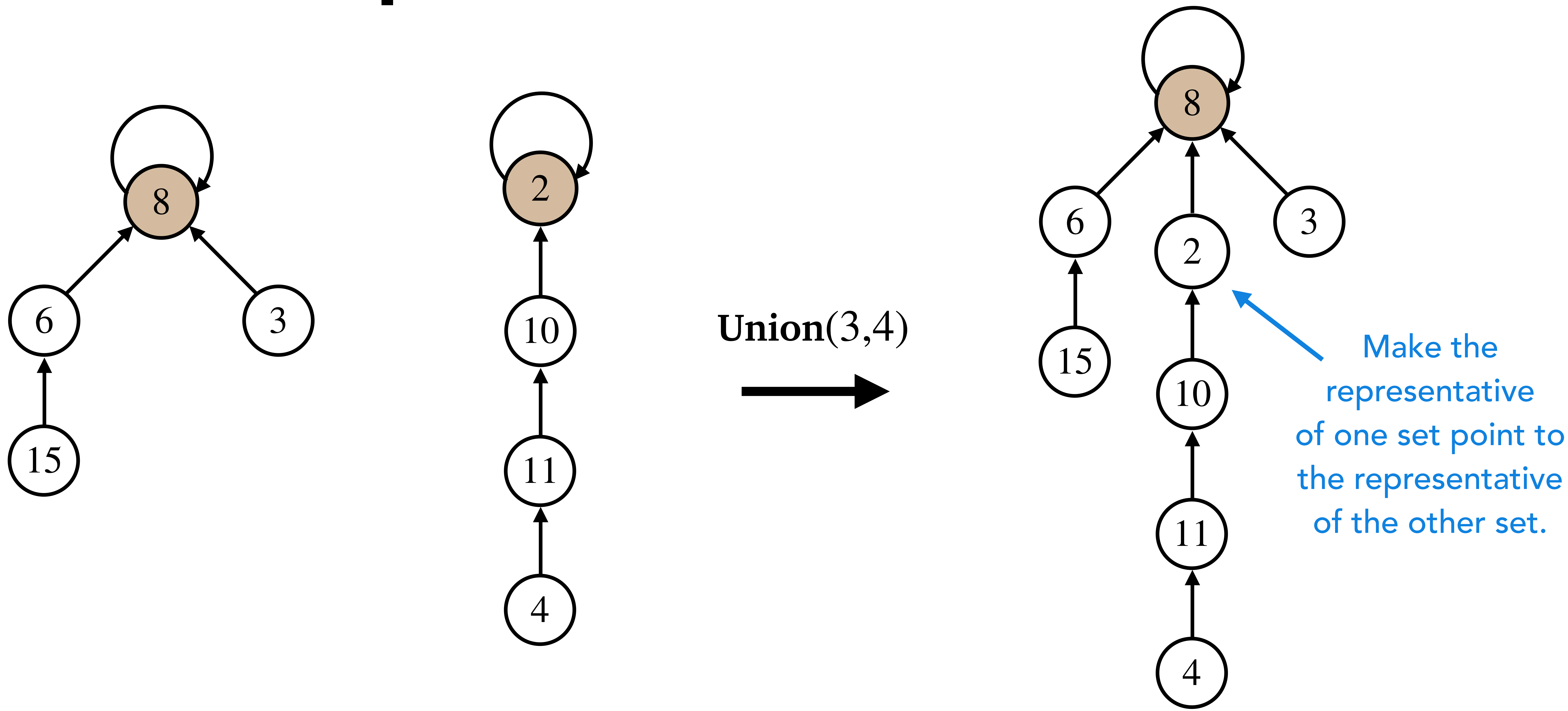
Union on Disjoint-Sets as Trees



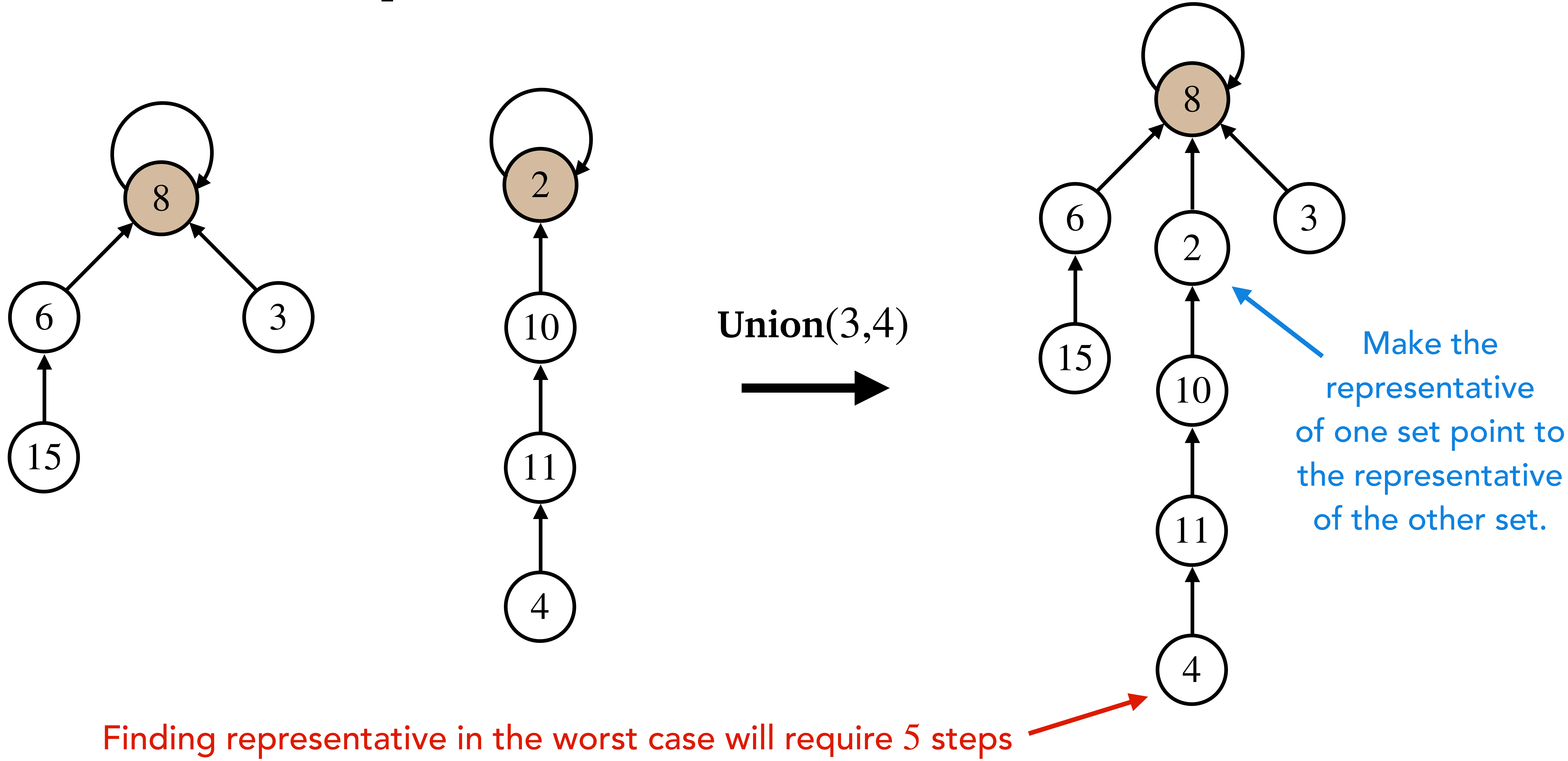
Union on Disjoint-Sets as Trees



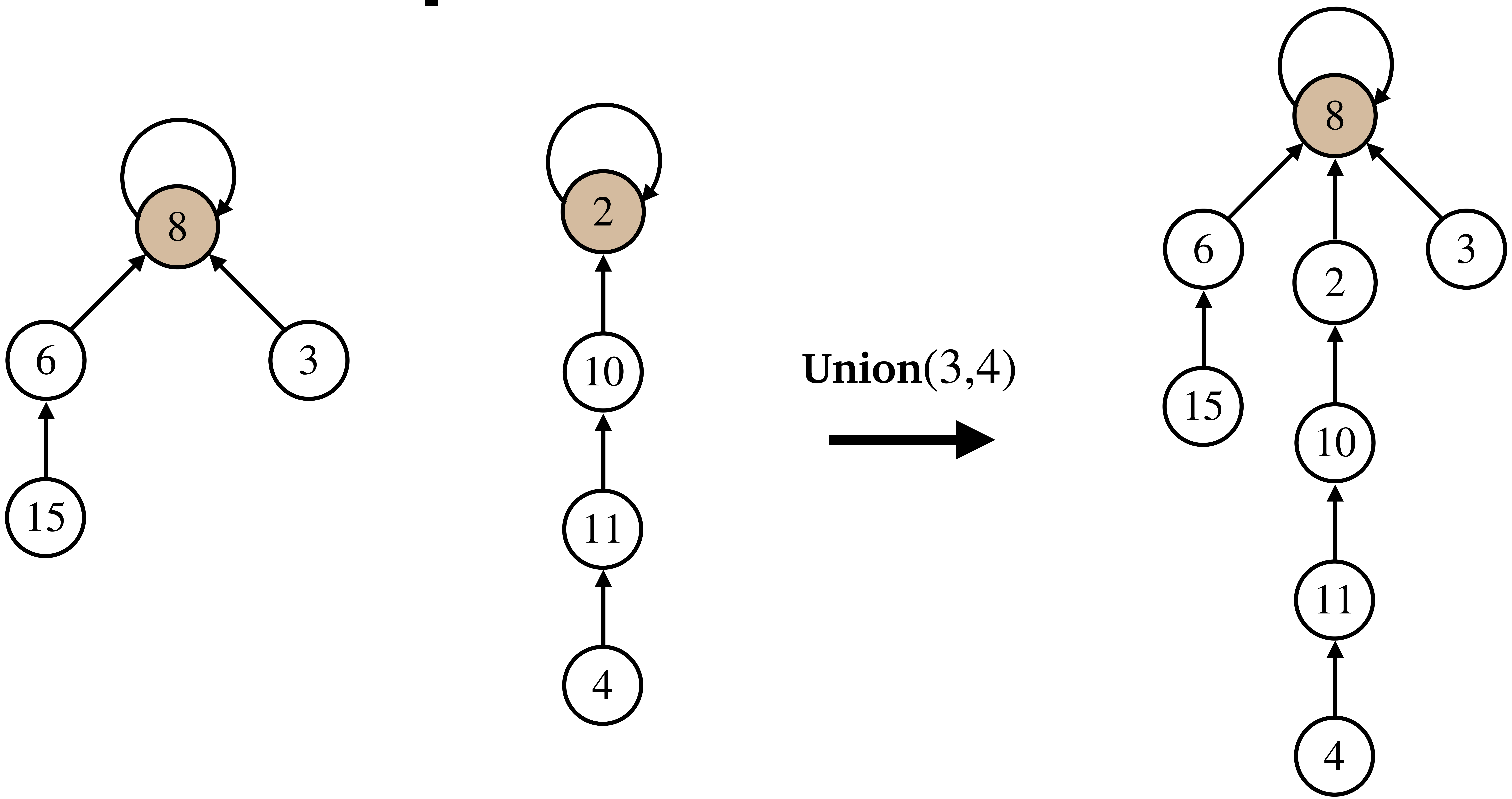
Union on Disjoint-Sets as Trees



Union on Disjoint-Sets as Trees

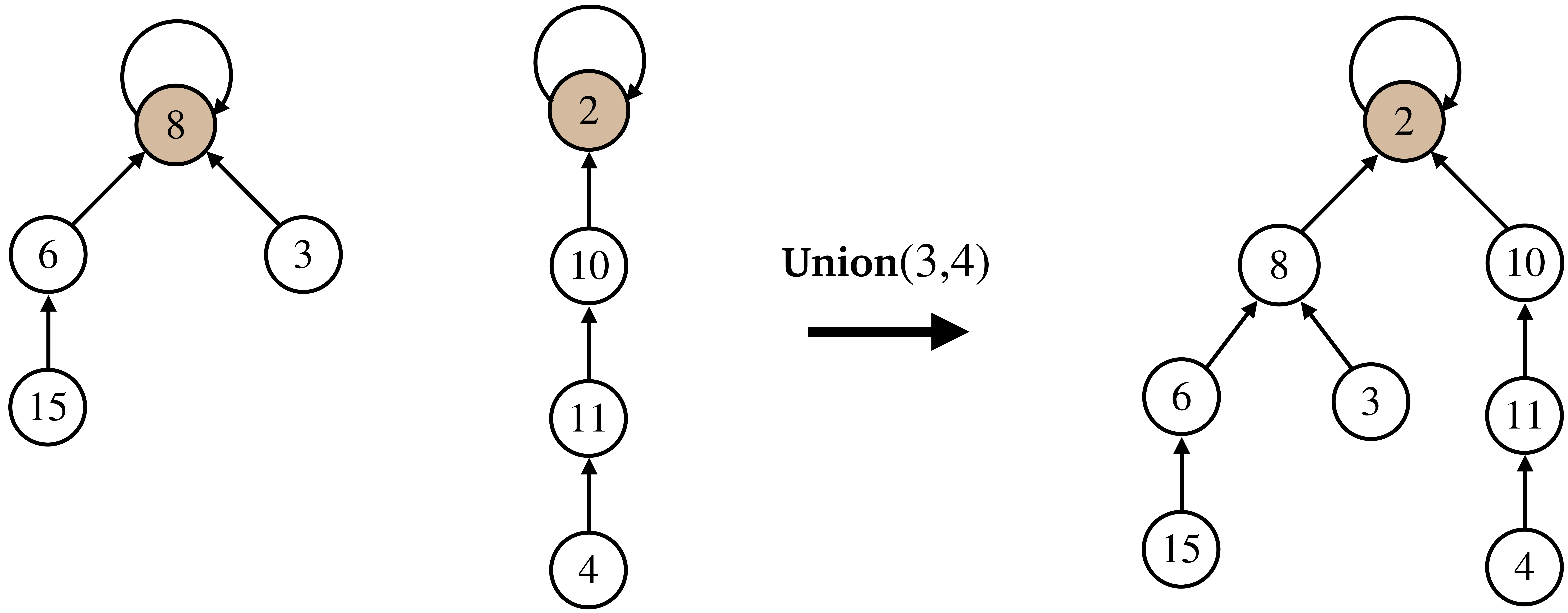


Union on Disjoint-Sets as Trees



Shouldn’t representative with smaller height point to representative with larger height?

Union on Disjoint-Sets as Trees



Union on Disjoint-Sets as Trees