Lecture 12

Augmenting Data Structures, Disjoint-Set Data Structure

Source: Introduction to Algorithms, CLRS

Maintaining Subtree Sizes: Insertion

Maintaining Subtree Sizes: Insertion

Insertion phase:

Maintaining Subtree Sizes: Insertion

Insertion phase:

Keep adding 1 to the **sizes** of **every node** we visit

Maintaining Subtree Sizes: Insertion

Insertion phase:

Keep adding 1 to the **sizes** of **every node** we visit while **searching** for the correct leaf

Maintaining Subtree Sizes: Insertion

Insertion phase:

Keep adding 1 to the **sizes** of **every node** we visit while **searching** for the correct leaf where new node can be inserted.

Maintaining Subtree Sizes: Insertion

Insertion phase:

Keep adding 1 to the **sizes** of **every node** we visit while **searching** for the correct leaf where new node can be inserted.

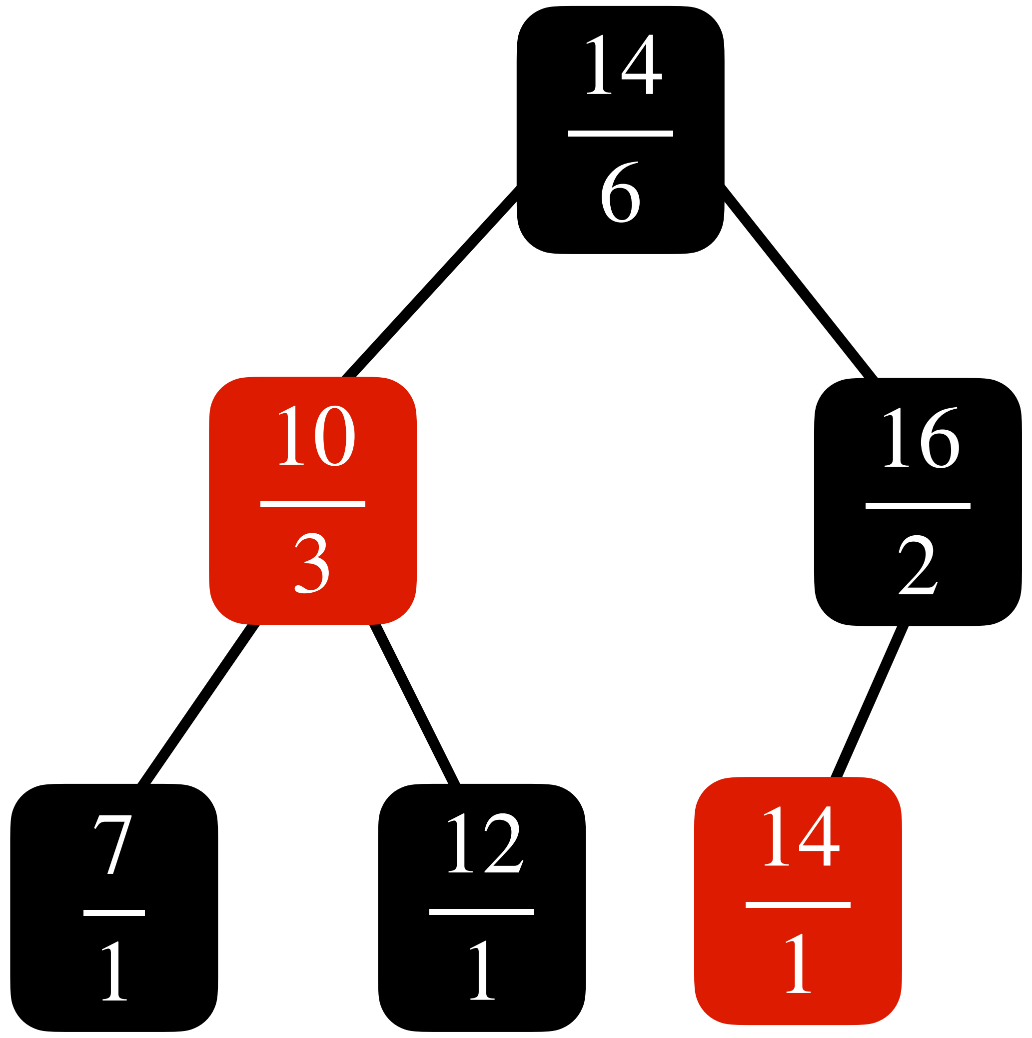
Insert 15 in this tree

Maintaining Subtree Sizes: Insertion

Insertion phase:

Keep adding 1 to the **sizes** of **every node** we visit while **searching** for the correct leaf where new node can be inserted.

Insert 15 in this tree

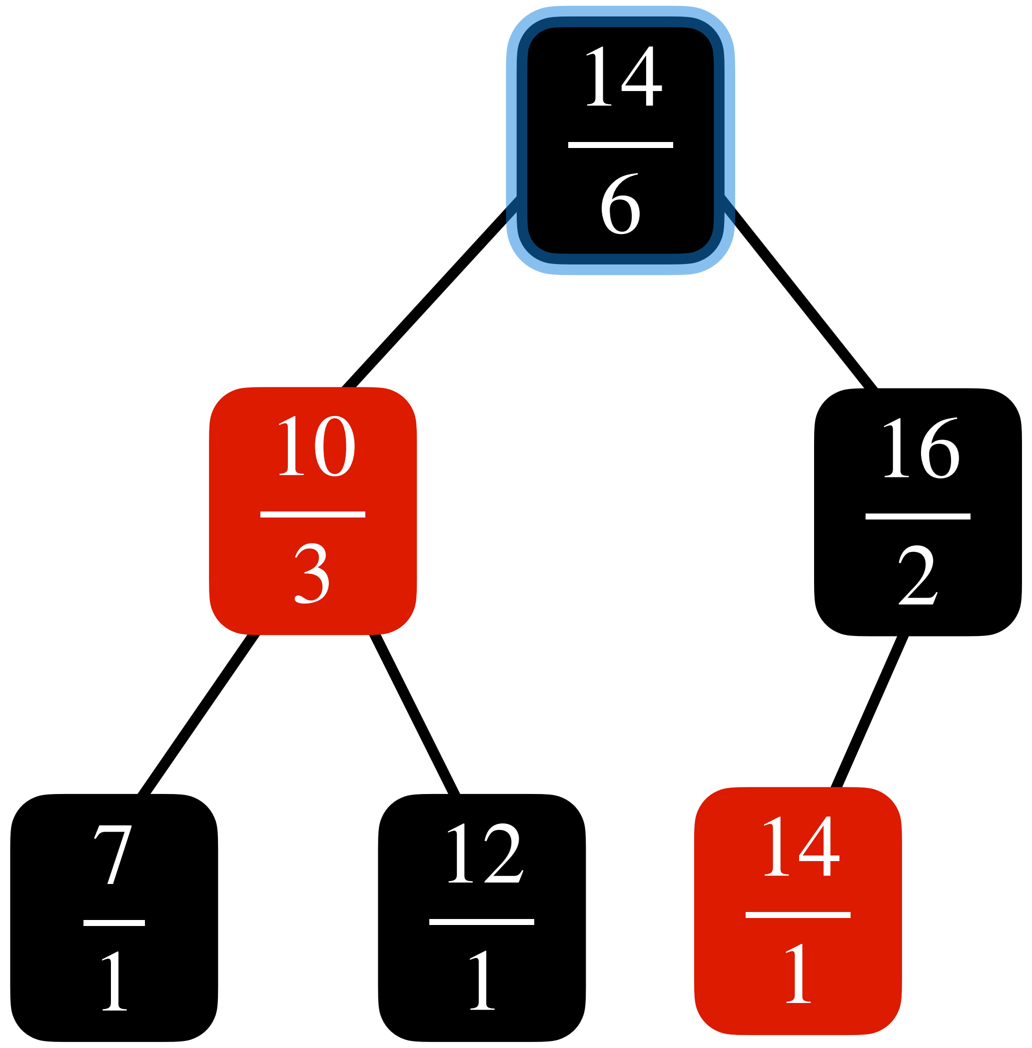


Maintaining Subtree Sizes: Insertion

Insertion phase:

Keep adding 1 to the **sizes** of **every node** we visit while **searching** for the correct leaf where new node can be inserted.

Insert 15 in this tree

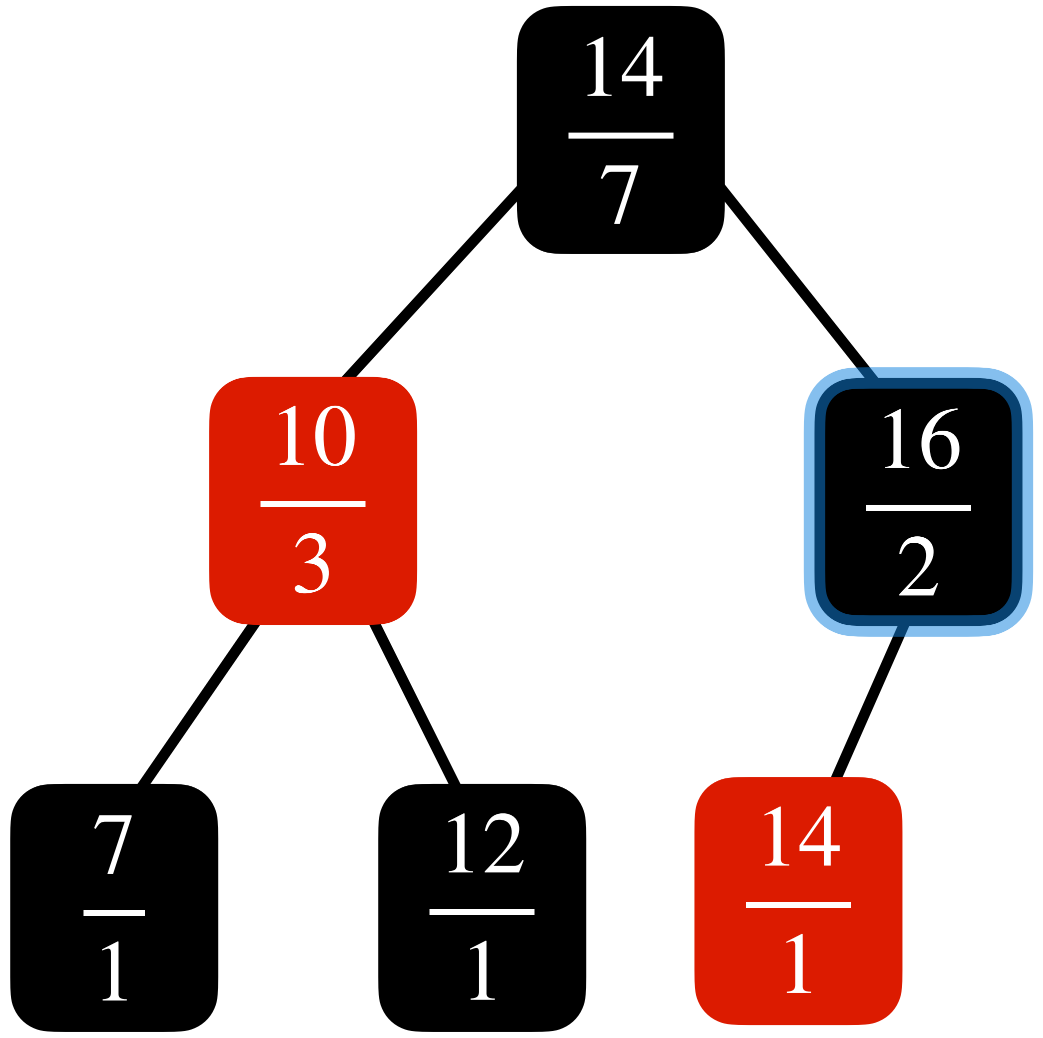


Maintaining Subtree Sizes: Insertion

Insertion phase:

Keep adding 1 to the **sizes** of **every node** we visit while **searching** for the correct leaf where new node can be inserted.

Insert 15 in this tree

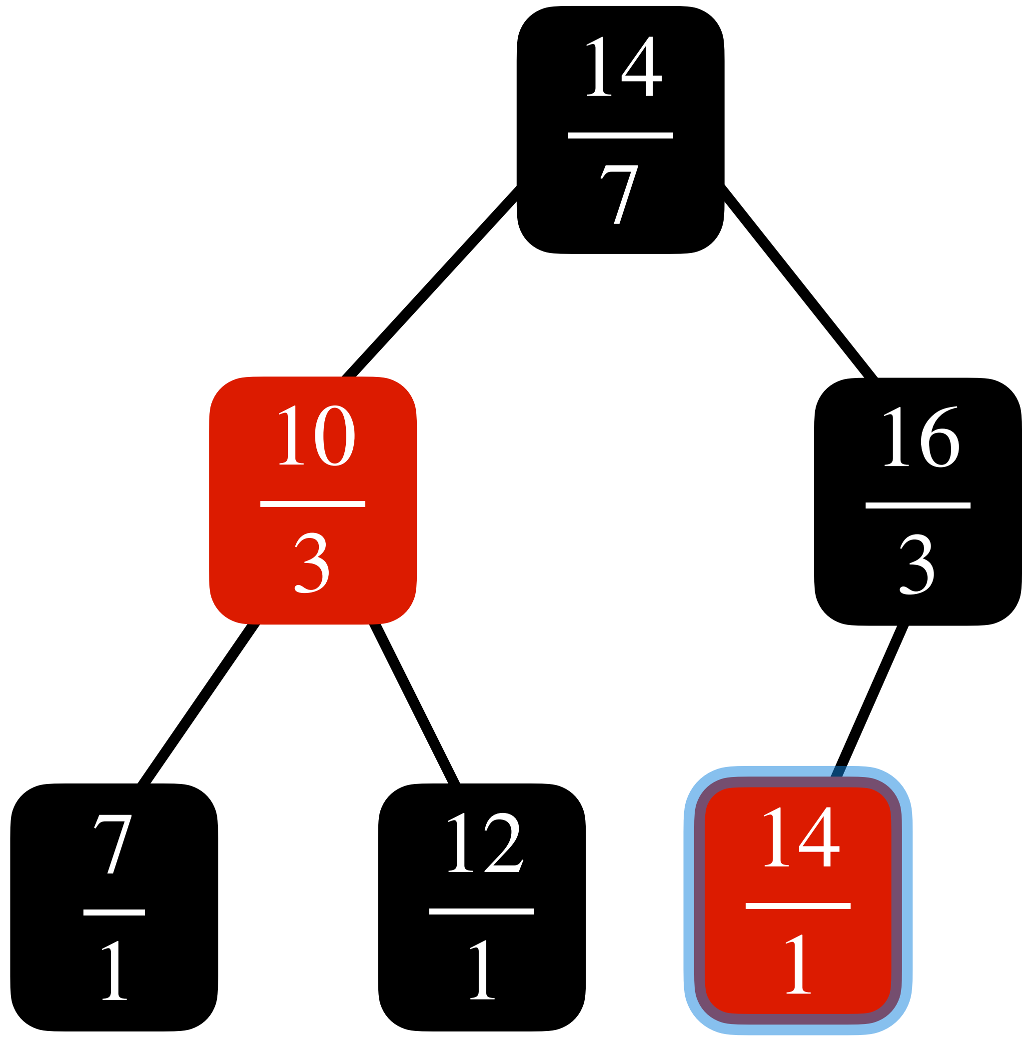


Maintaining Subtree Sizes: Insertion

Insertion phase:

Keep adding 1 to the **sizes** of **every node** we visit while **searching** for the correct leaf where new node can be inserted.

Insert 15 in this tree

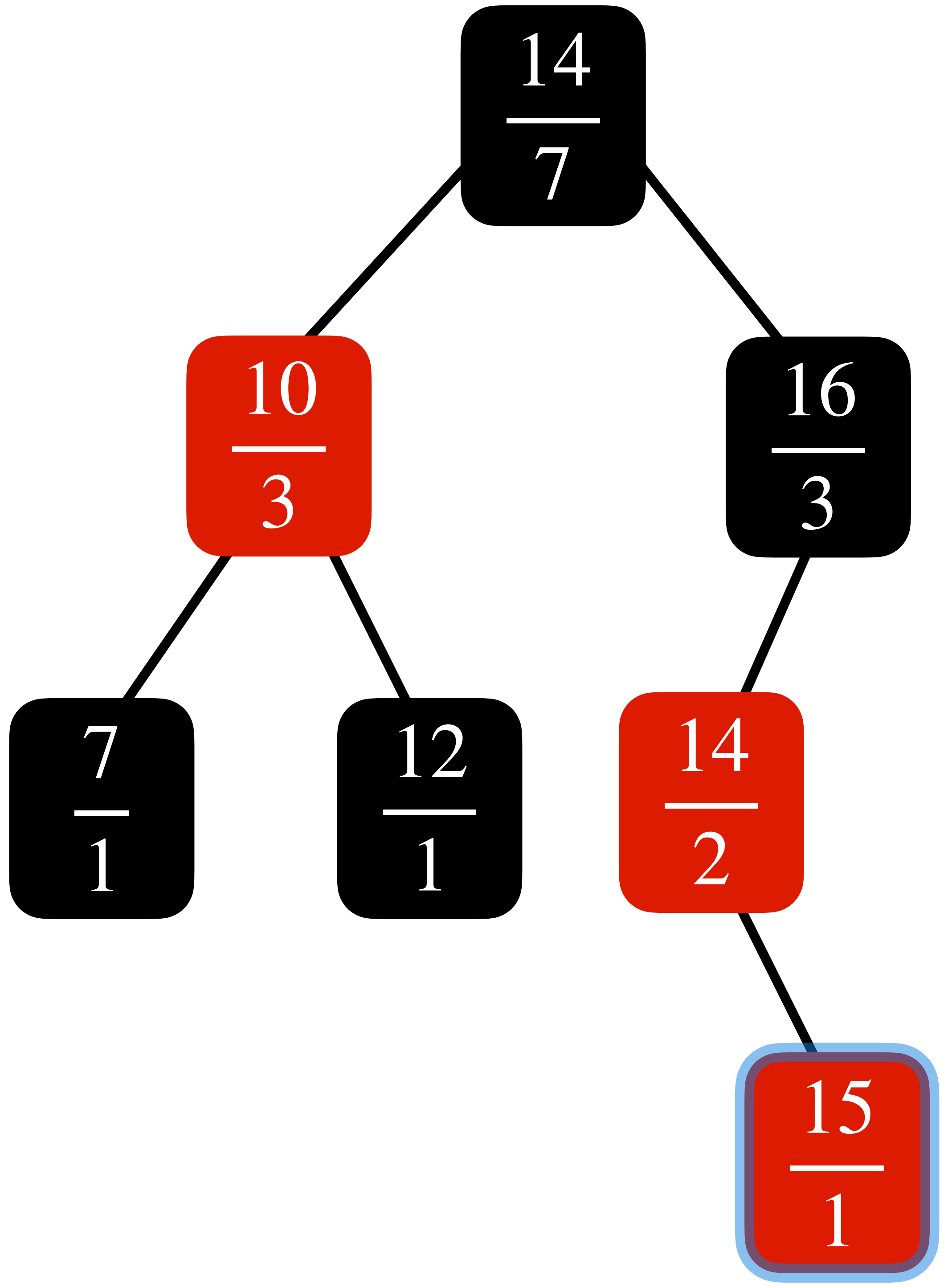


Maintaining Subtree Sizes: Insertion

Insertion phase:

Keep adding 1 to the **sizes** of **every node** we visit while **searching** for the correct leaf where new node can be inserted.

Insert 15 in this tree



Maintaining Subtree Sizes: Insertion

Maintaining Subtree Sizes: Insertion

Fix-up phase:

Maintaining Subtree Sizes: Insertion

Fix-up phase:

- Fix-ups involve only **rotations** and **recolouring**.

Maintaining Subtree Sizes: Insertion

Fix-up phase:

- Fix-ups involve only **rotations** and **recolouring**.
- **Recolouring** doesn't require changing sizes.

Maintaining Subtree Sizes: Insertion

Fix-up phase:

- Fix-ups involve only **rotations** and **recolouring**.
- **Recolouring** doesn’t require changing sizes.
- During **rotations** size changes are doable in constant time.

Maintaining Subtree Sizes: Deletion

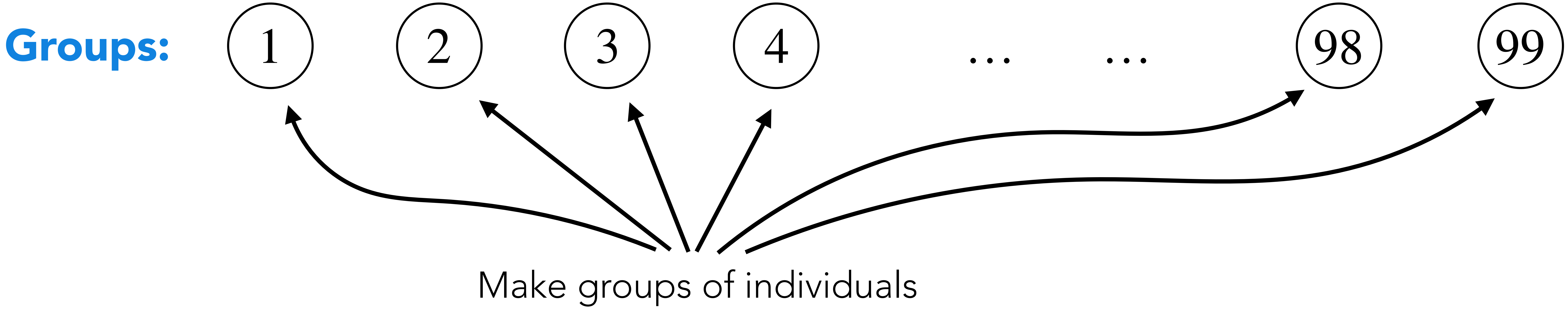
DIY.

Finding Friend Groups on Facebook

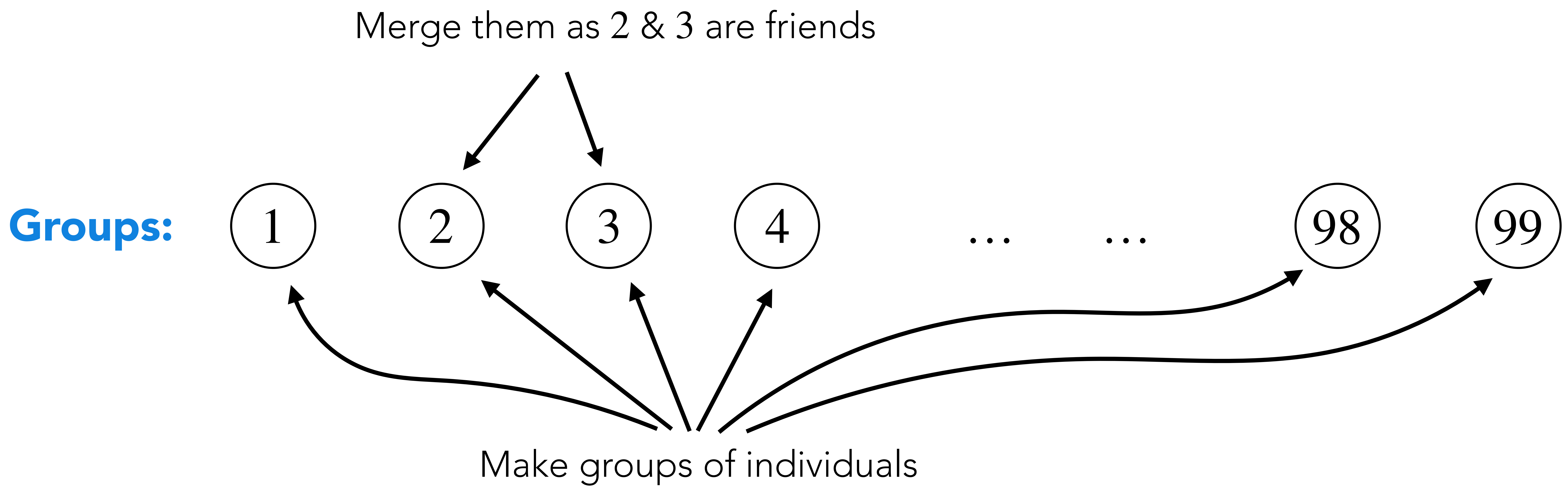
Finding Friend Groups on Facebook

Users: 1 2 3 4 … … 98 99

Finding Friend Groups on Facebook



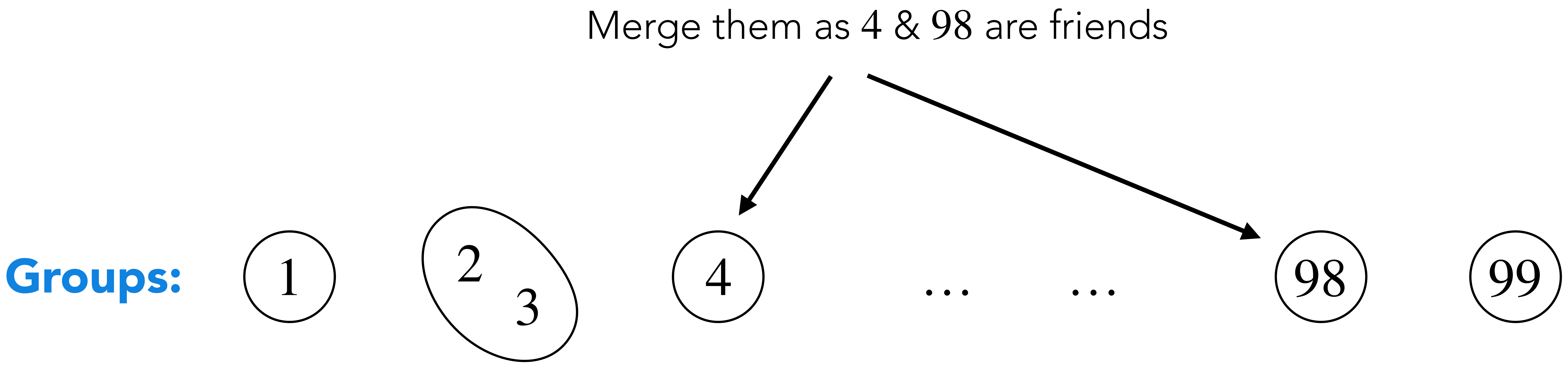
Finding Friend Groups on Facebook



Finding Friend Groups on Facebook

Groups:

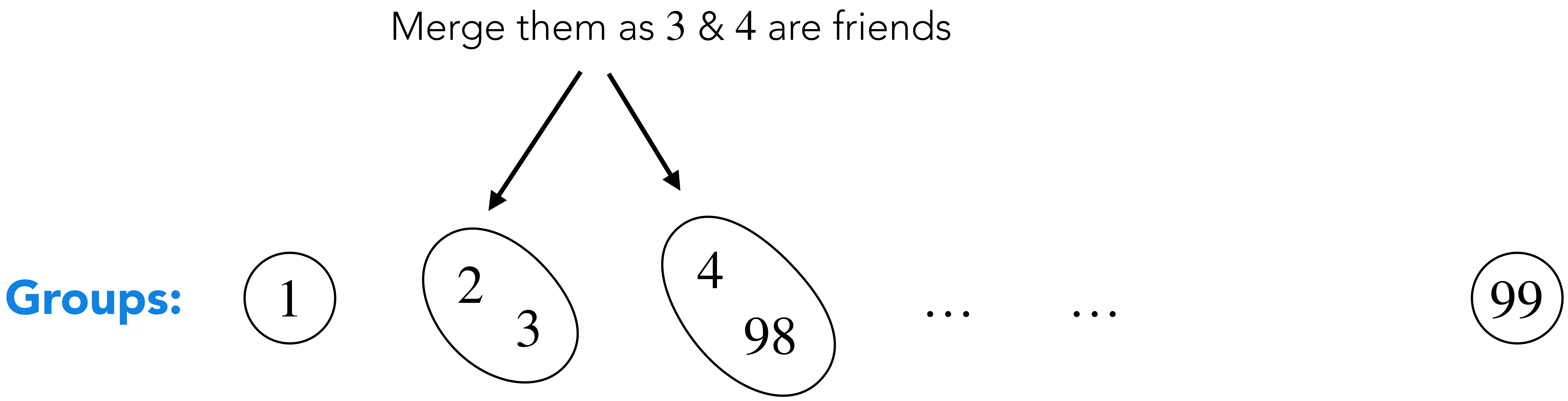
Finding Friend Groups on Facebook



Finding Friend Groups on Facebook

Groups: 

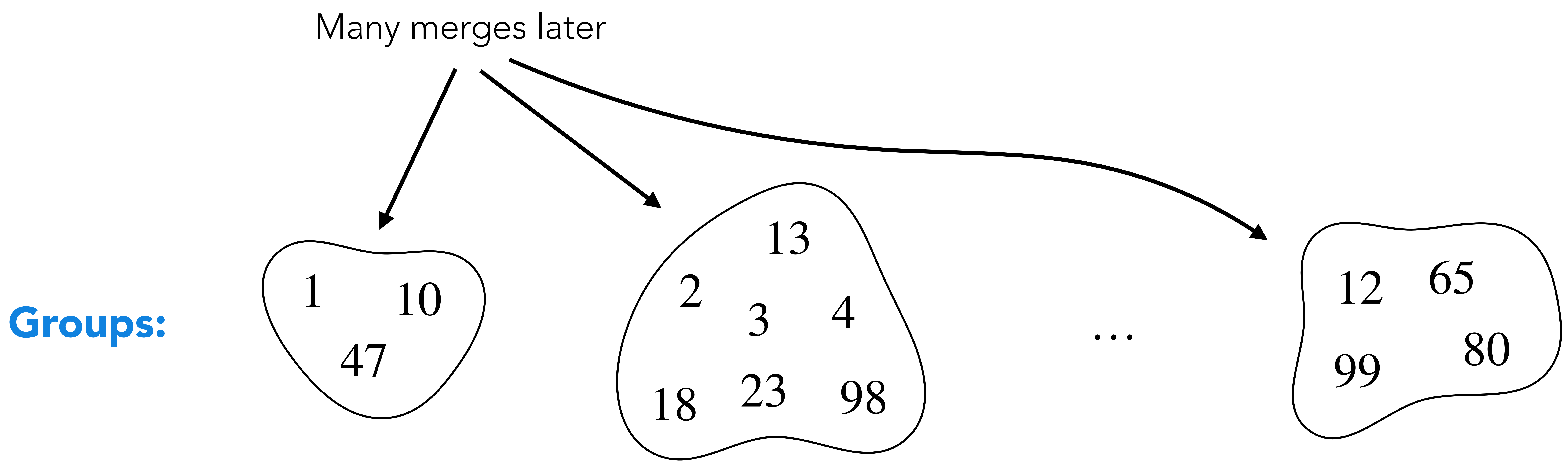
Finding Friend Groups on Facebook



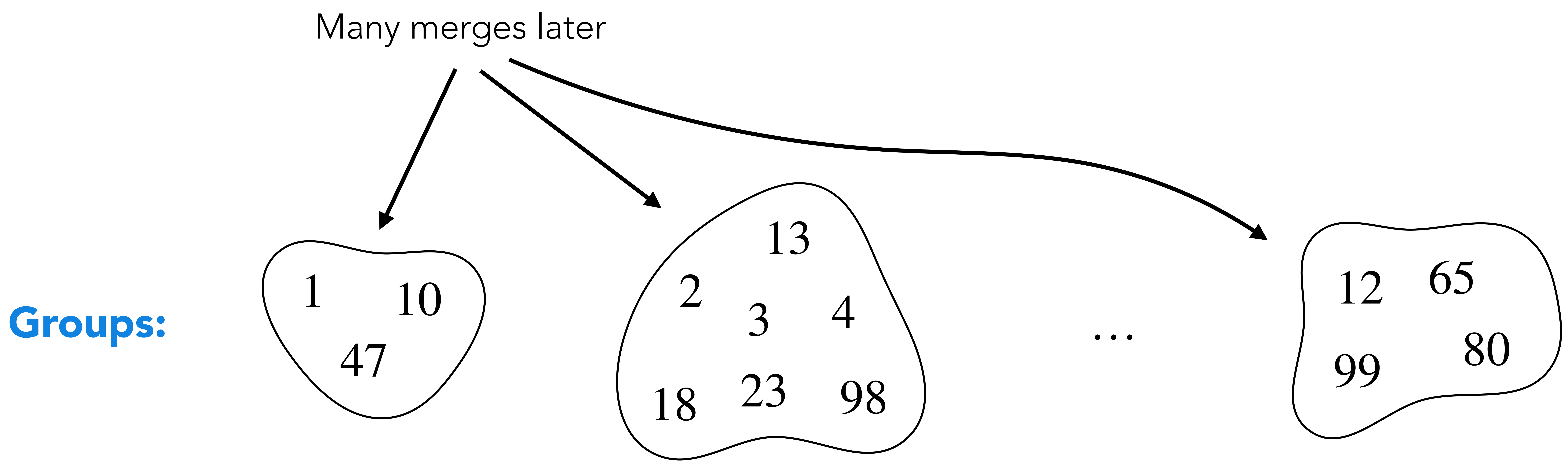
Finding Friend Groups on Facebook

Groups: 1   2 4 3 98   … …   99

Finding Friend Groups on Facebook

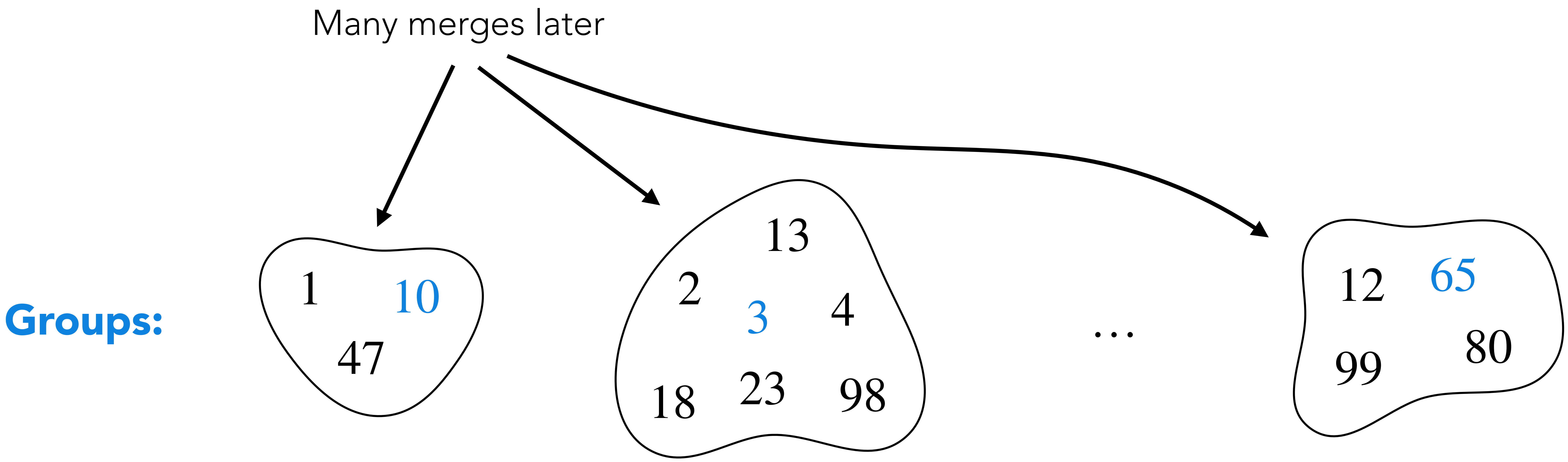


Finding Friend Groups on Facebook



New Feature: To identify groups keep a representative for each group.

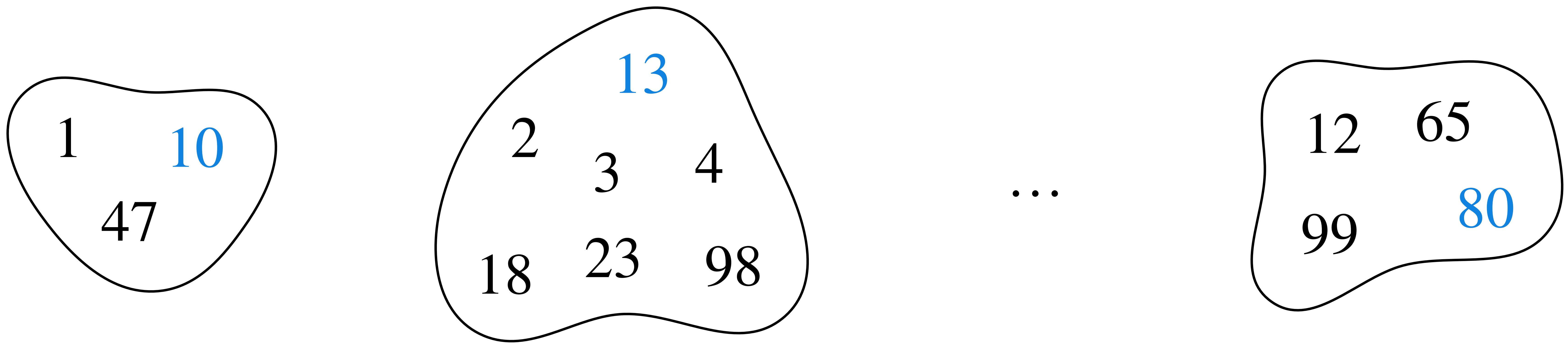
Finding Friend Groups on Facebook



New Feature: To identify groups keep a representative for each group.

Finding Friend Groups on Facebook

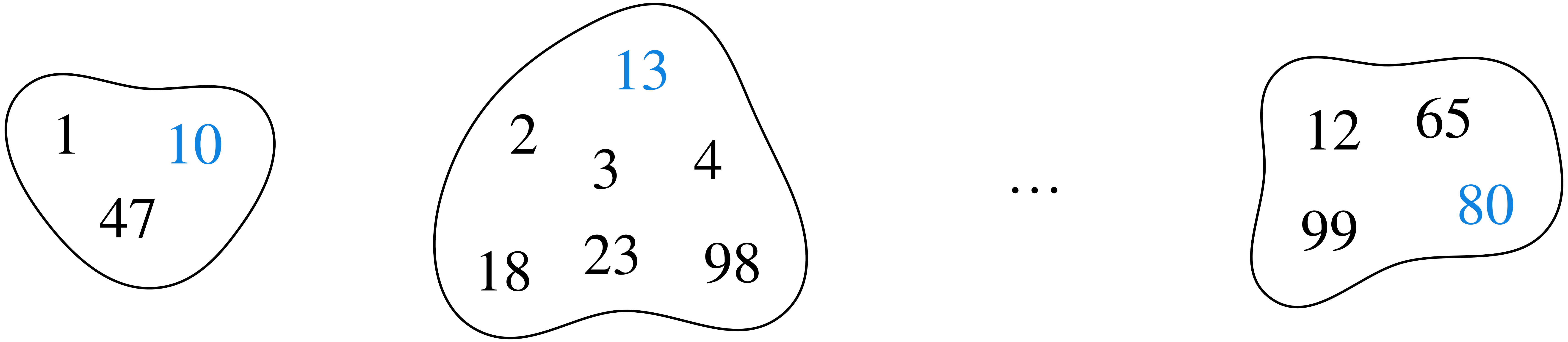
Groups:



Goal: Design a data-structure so that:

Finding Friend Groups on Facebook

Groups:

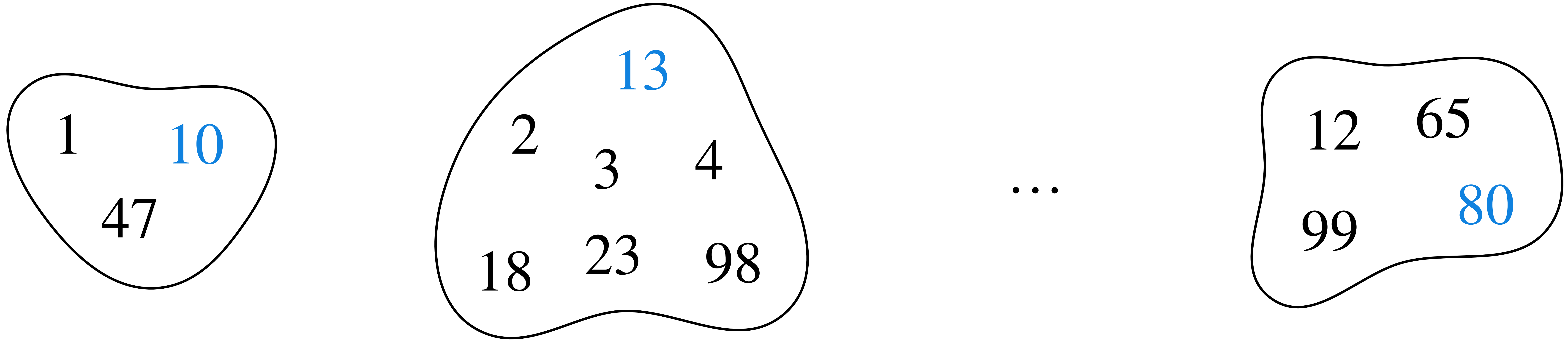


Goal: Design a data-structure so that:

1. Merging is fast.

Finding Friend Groups on Facebook

Groups:



Goal: Design a data-structure so that:

1. Merging is fast.
2. Finding representative is fast.

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.

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.
- **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.
- **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.
- **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.
- **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

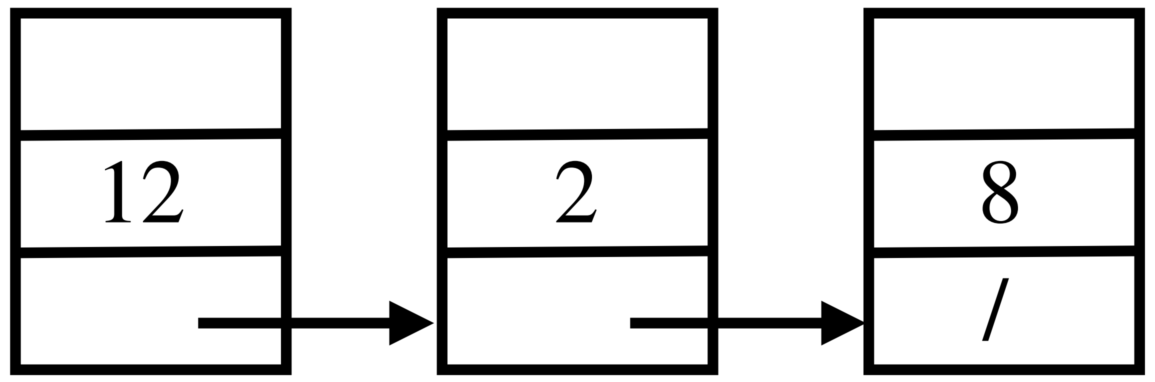
Disjoint-set data structure is useful in:

- 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

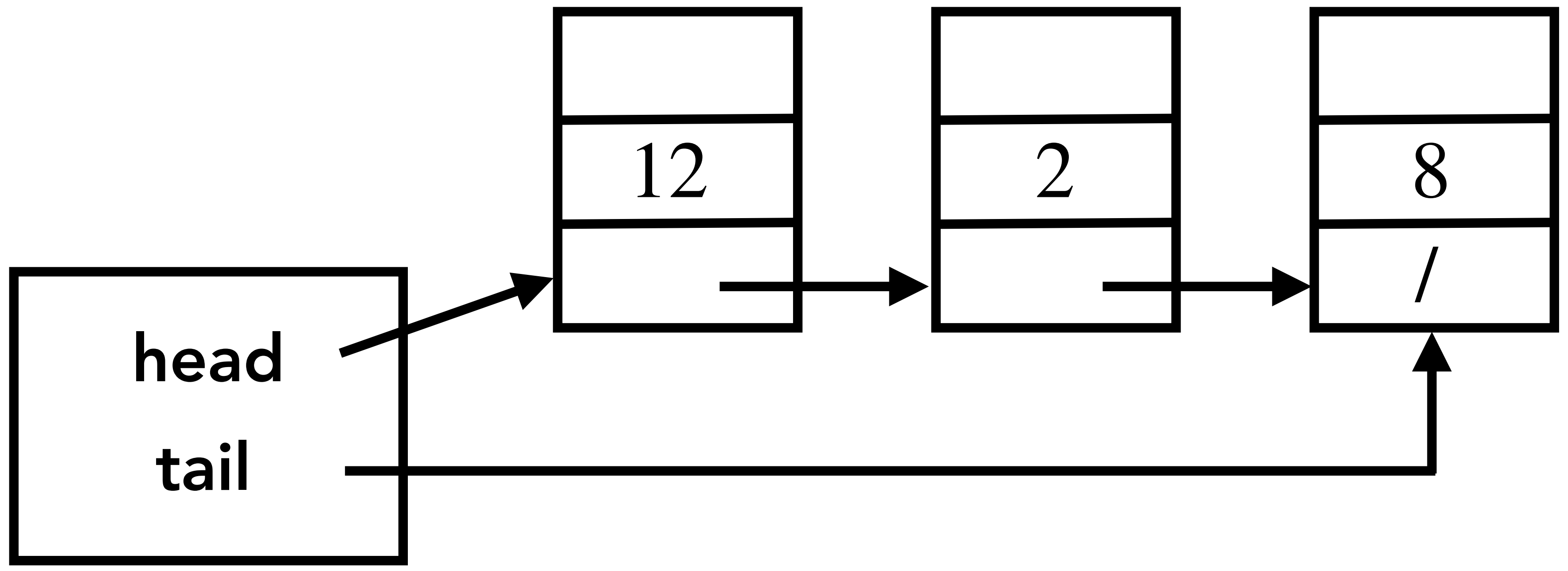
Disjoint-Sets as Linked Lists

S_1

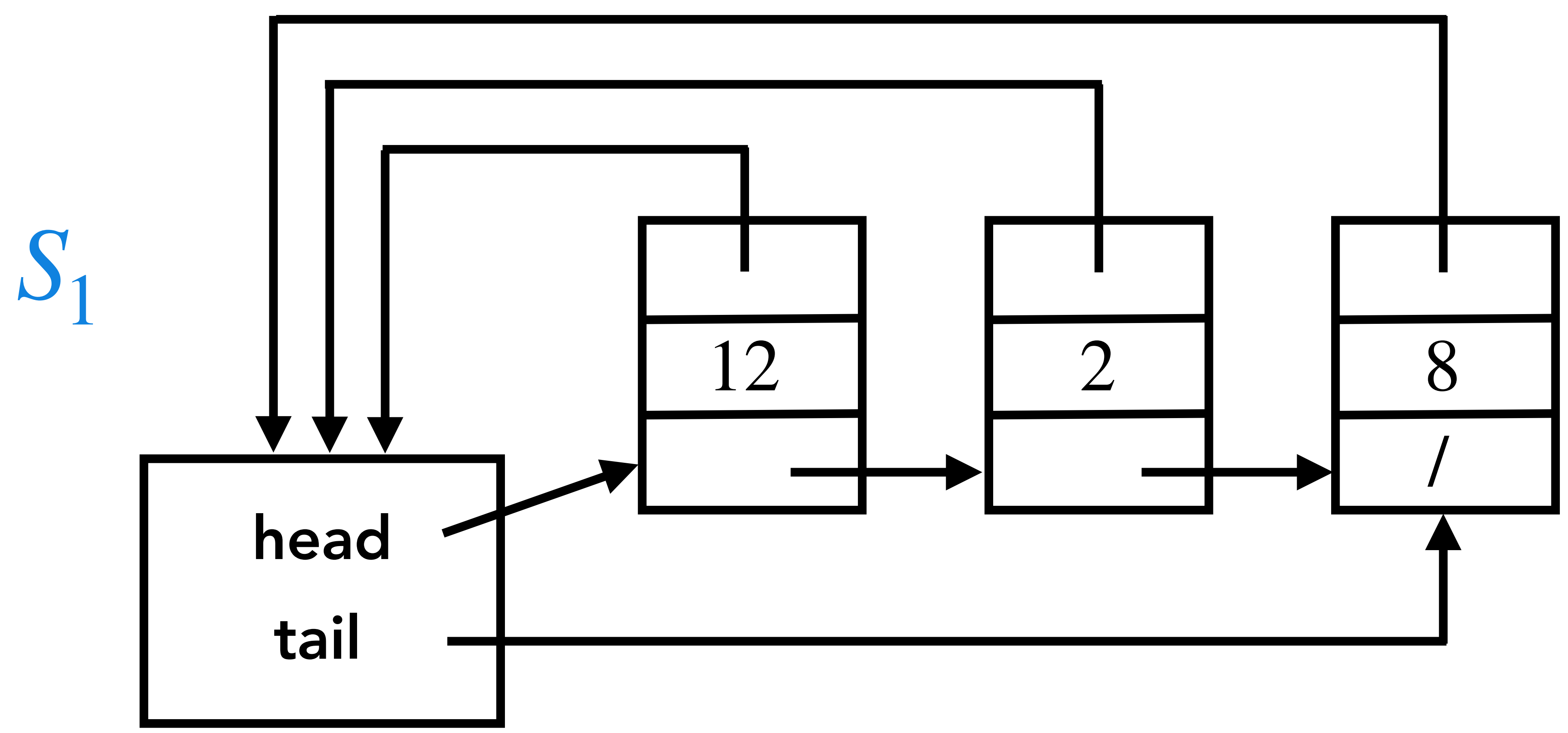


Disjoint-Sets as Linked Lists

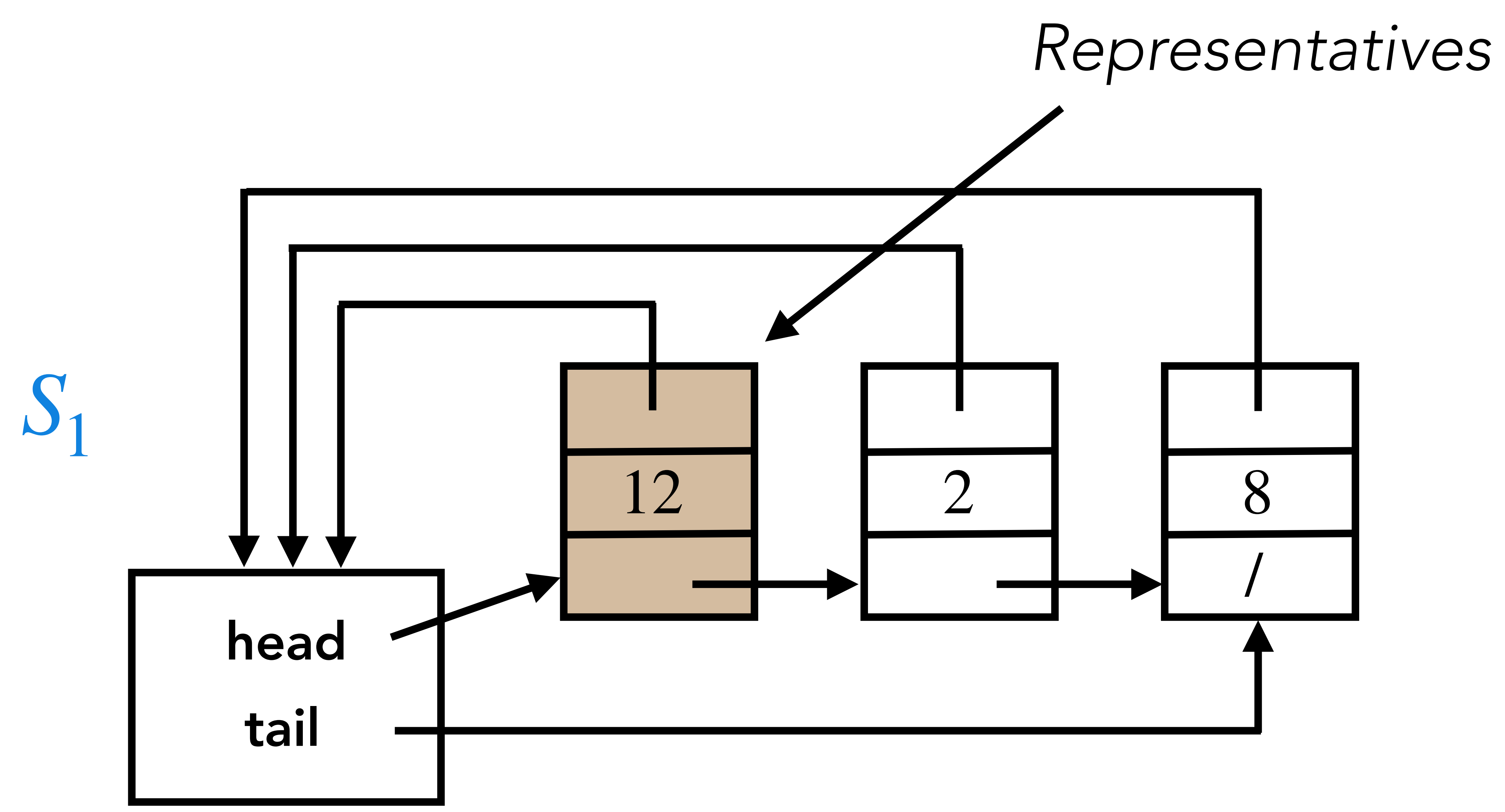
S_1



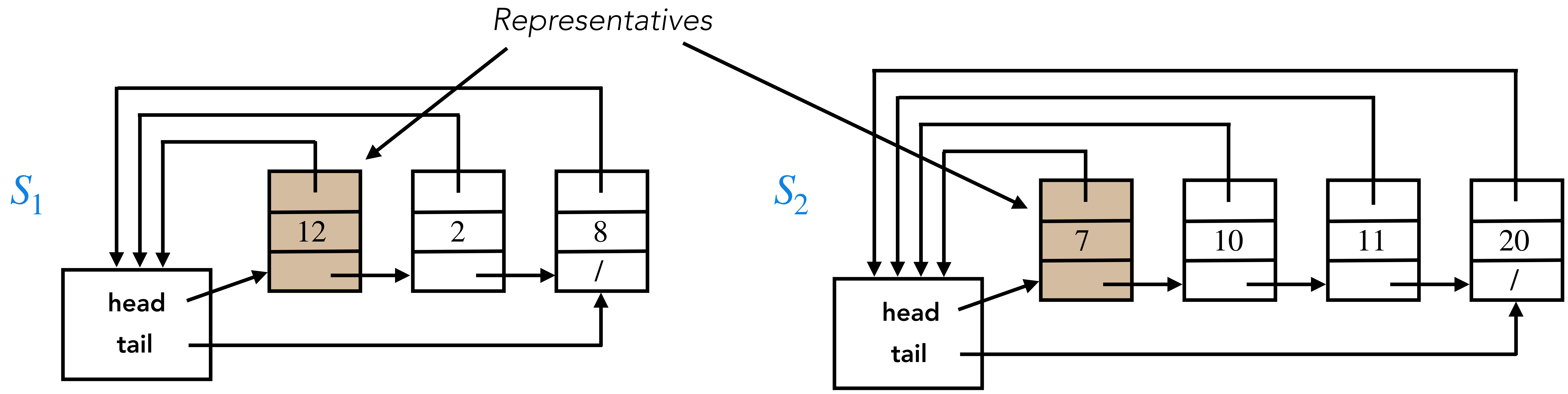
Disjoint-Sets as Linked Lists



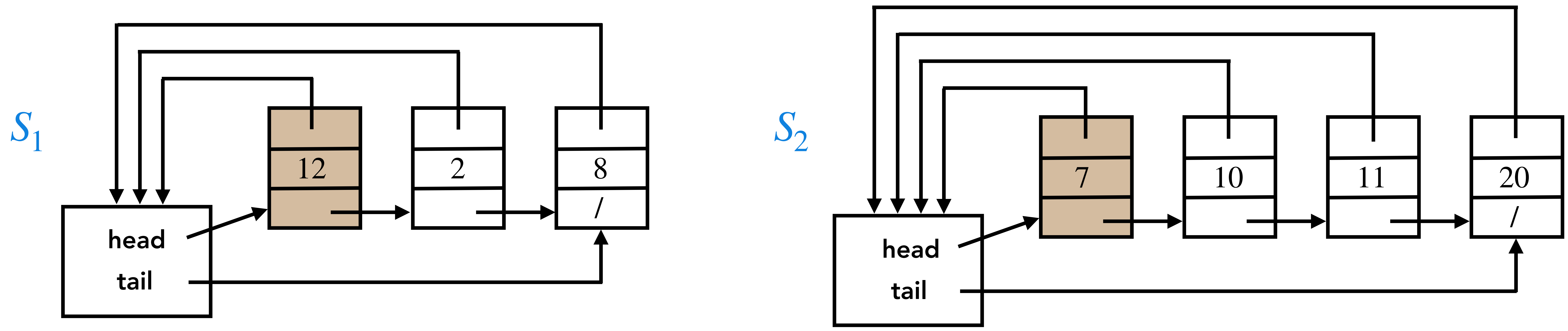
Disjoint-Sets as Linked Lists



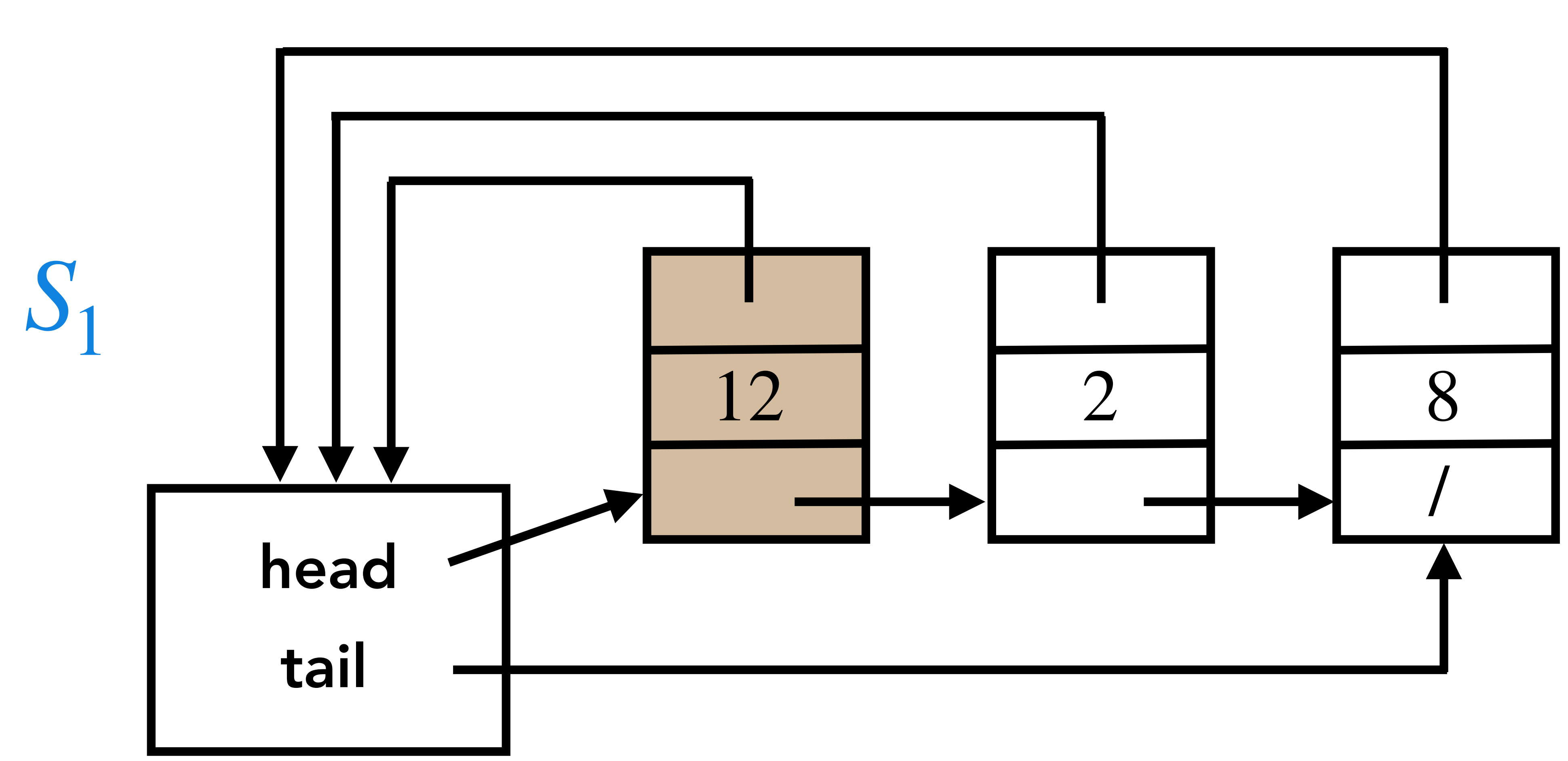
Disjoint-Sets as Linked Lists

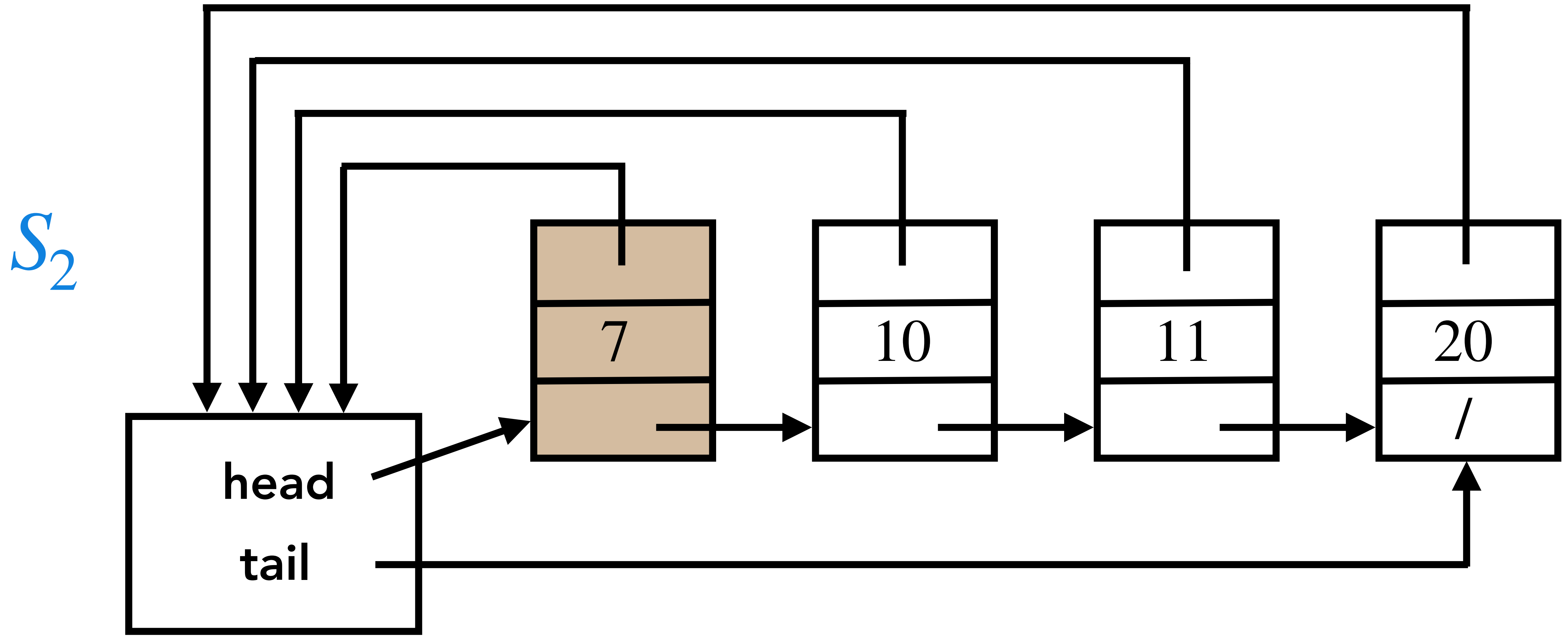


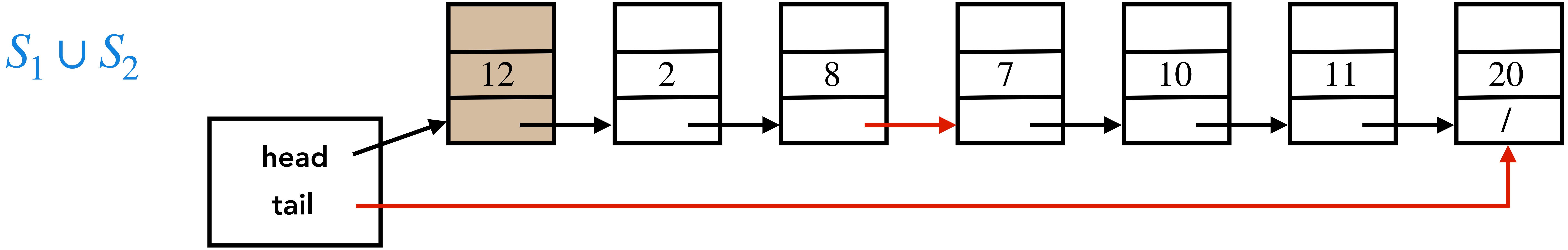
Disjoint-Sets as Linked Lists



Disjoint-Sets as Linked Lists

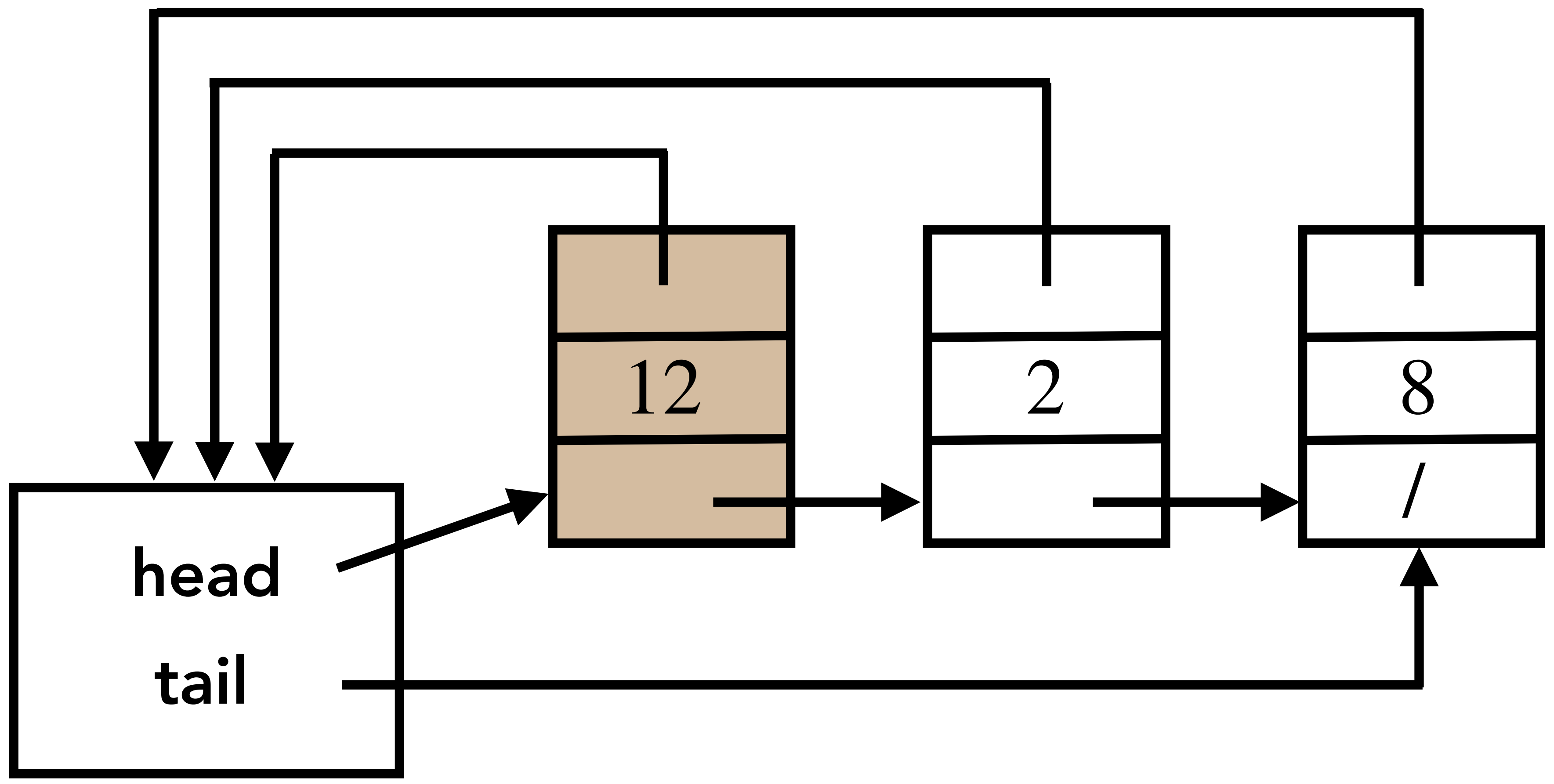




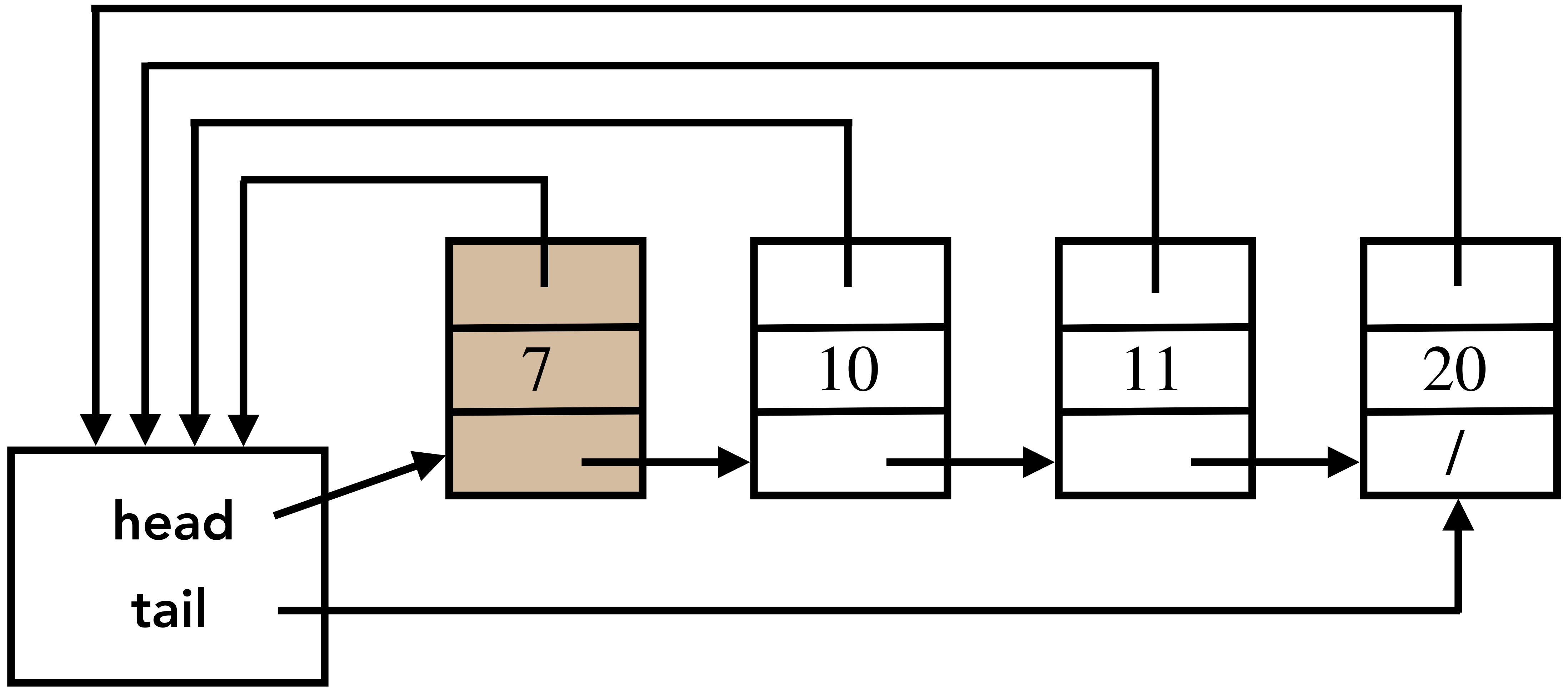


Disjoint-Sets as Linked Lists

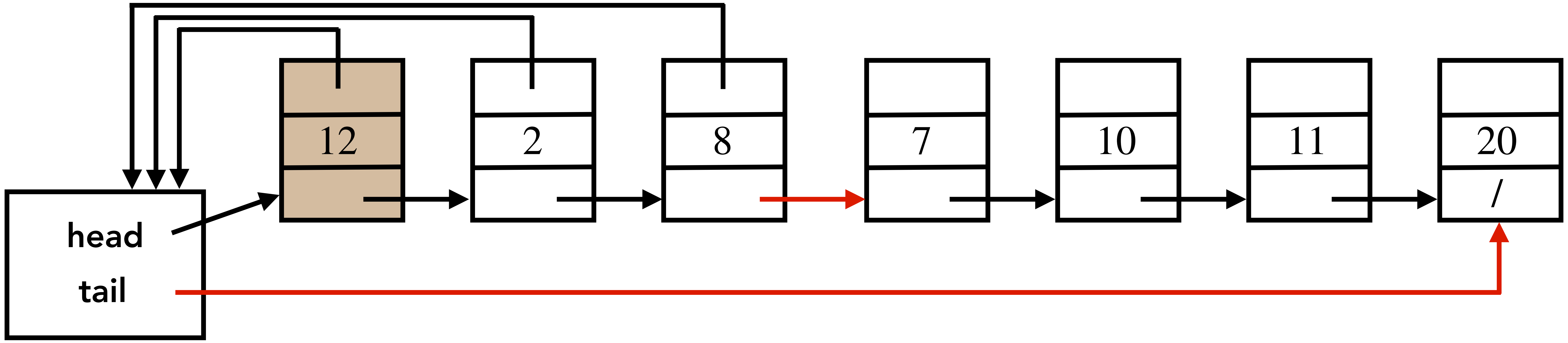
S_1



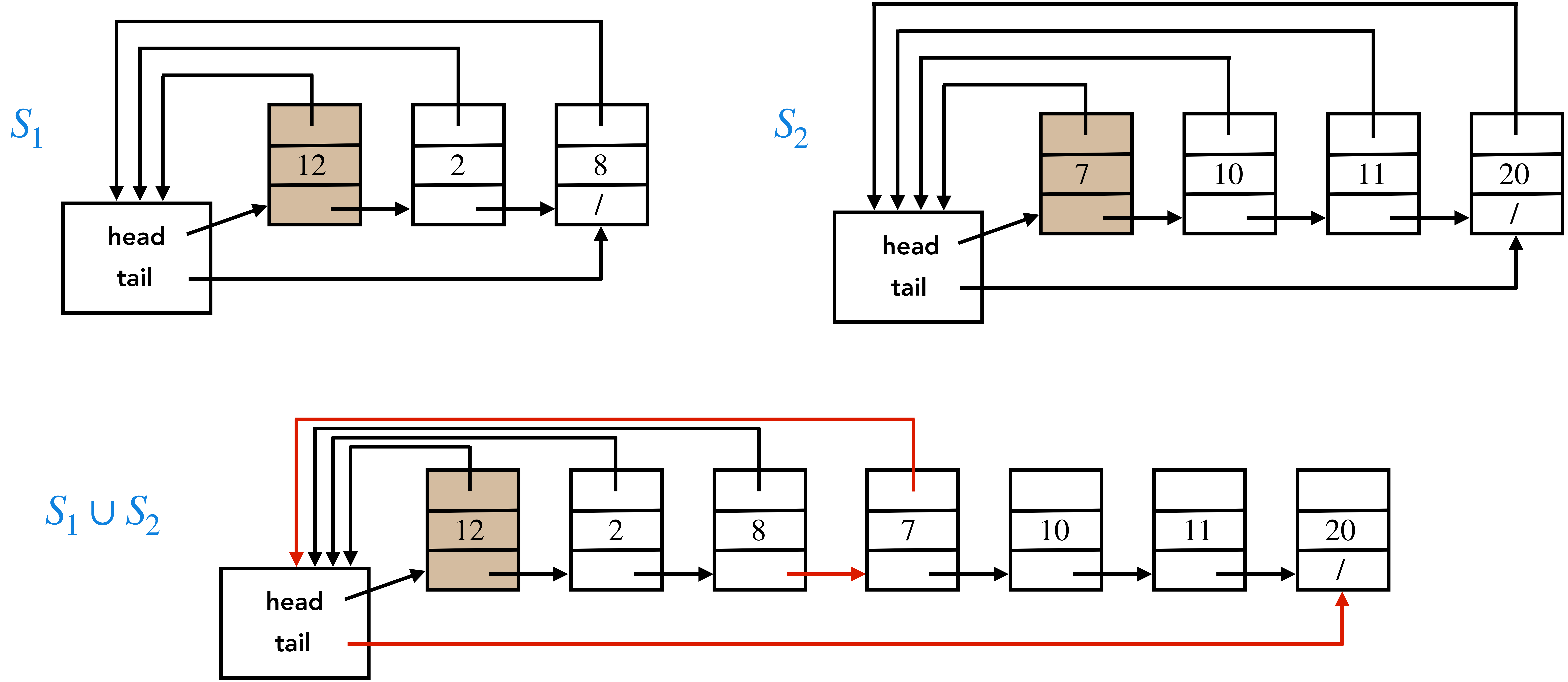
S_2



$S_1 \cup S_2$

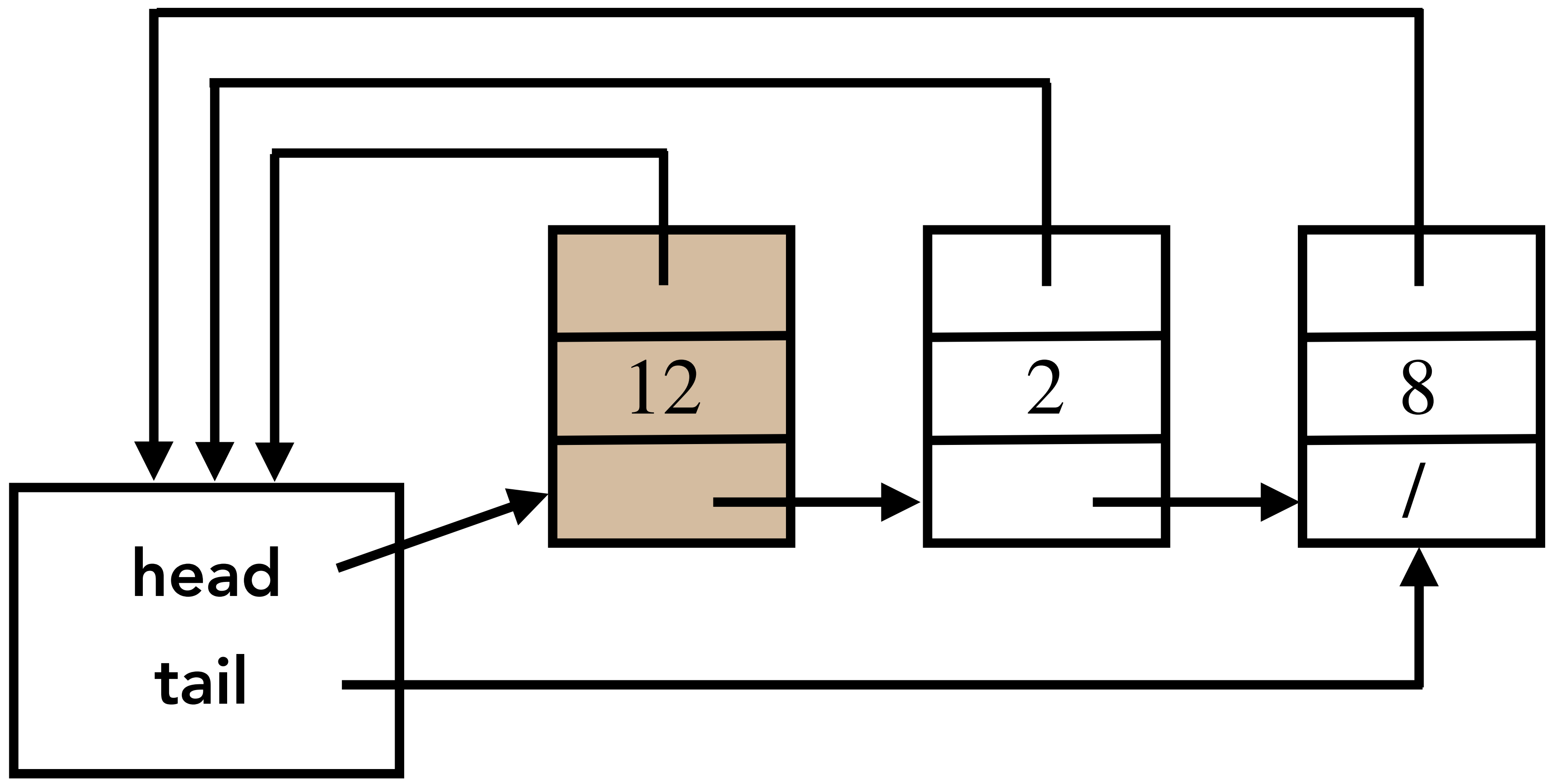


Disjoint-Sets as Linked Lists

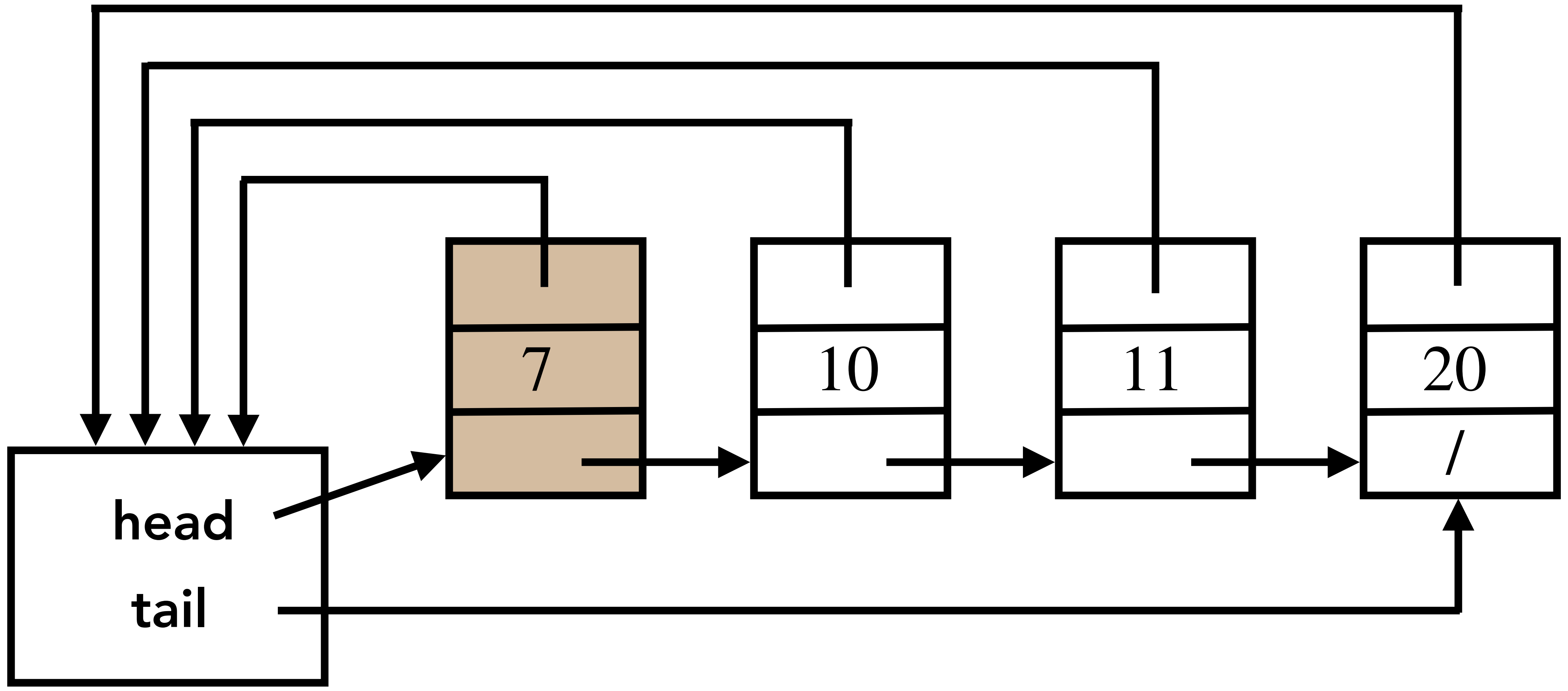


Disjoint-Sets as Linked Lists

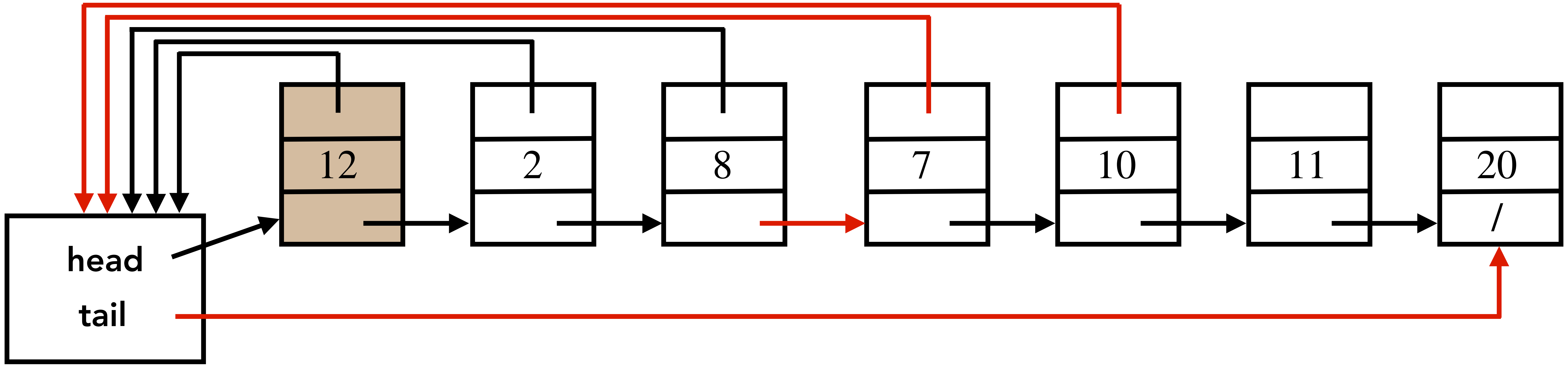
S_1



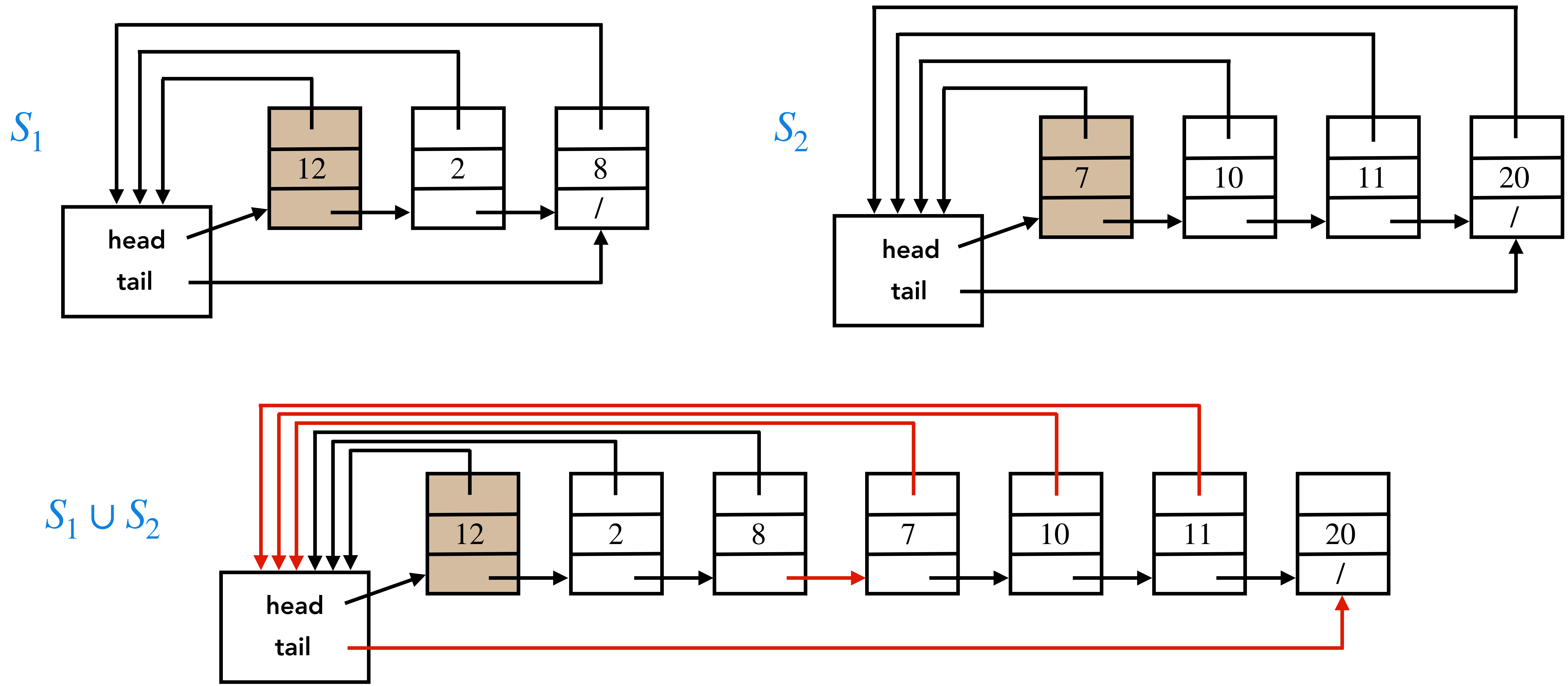
S_2



$S_1 \cup S_2$

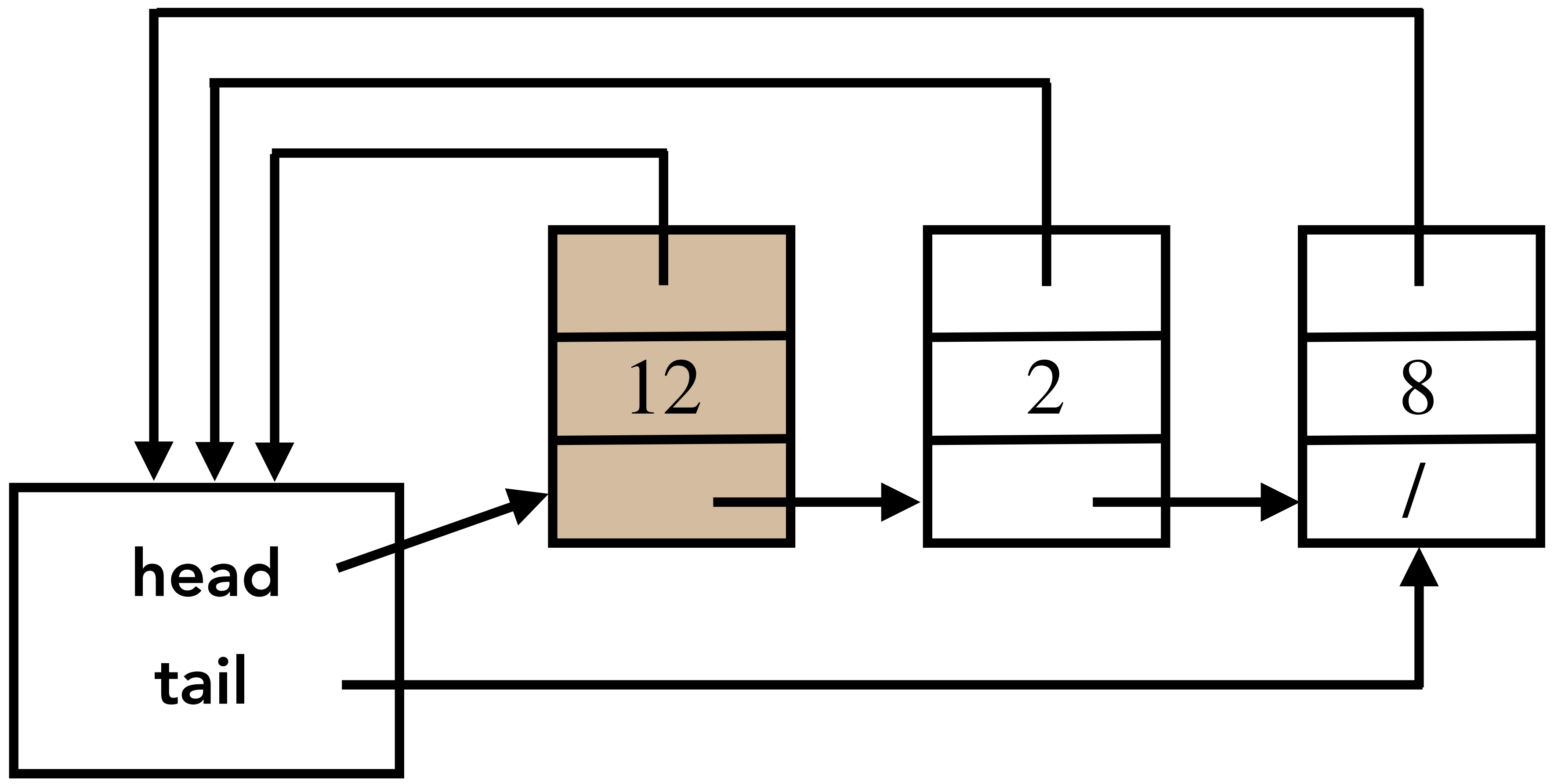


Disjoint-Sets as Linked Lists

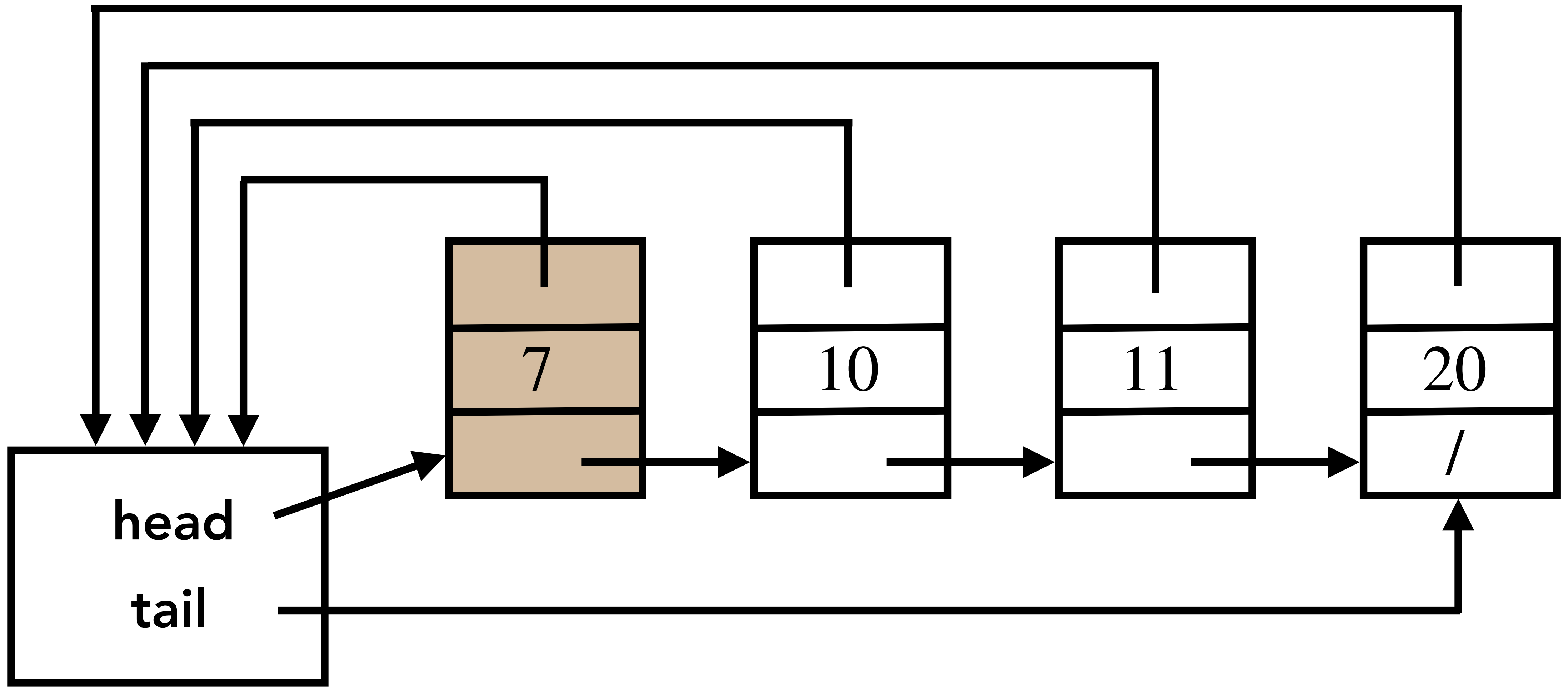


Disjoint-Sets as Linked Lists

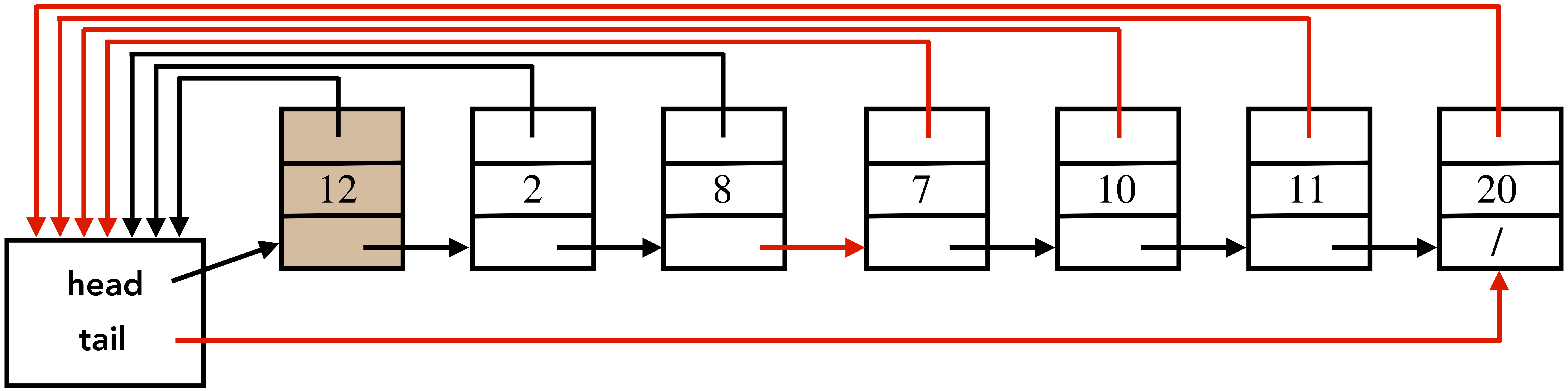
S_1



S_2

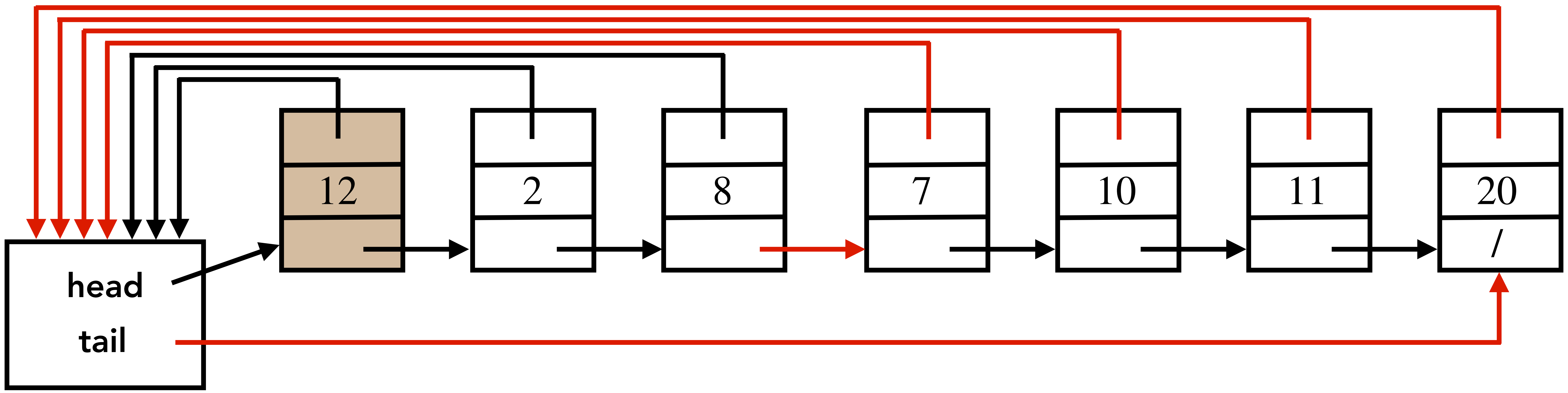


$S_1 \cup S_2$



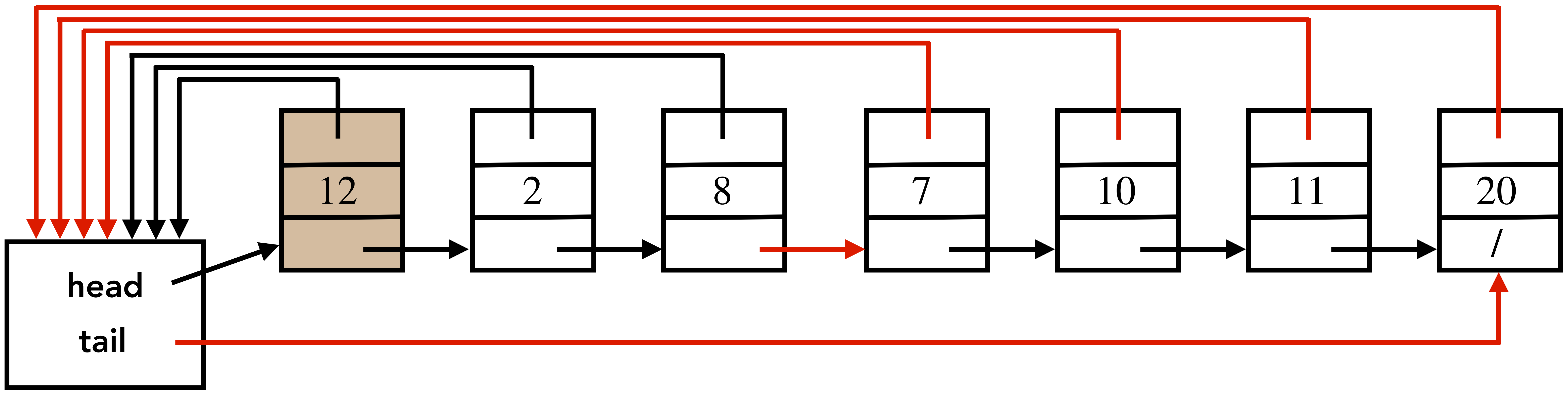
Disjoint-Sets as Linked Lists

$S_1 \cup S_2$



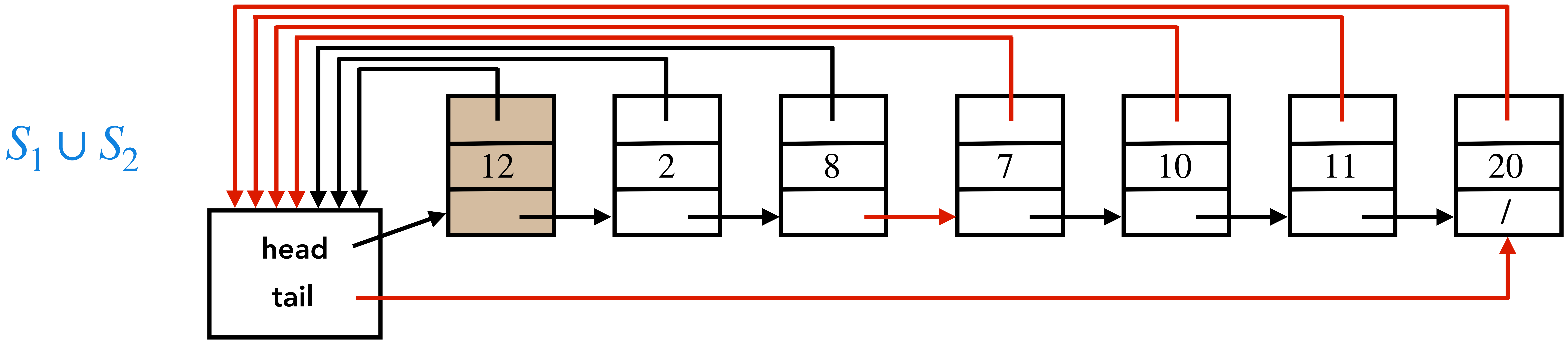
Disjoint-Sets as Linked Lists

$S_1 \cup S_2$



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

Disjoint-Sets as Linked Lists

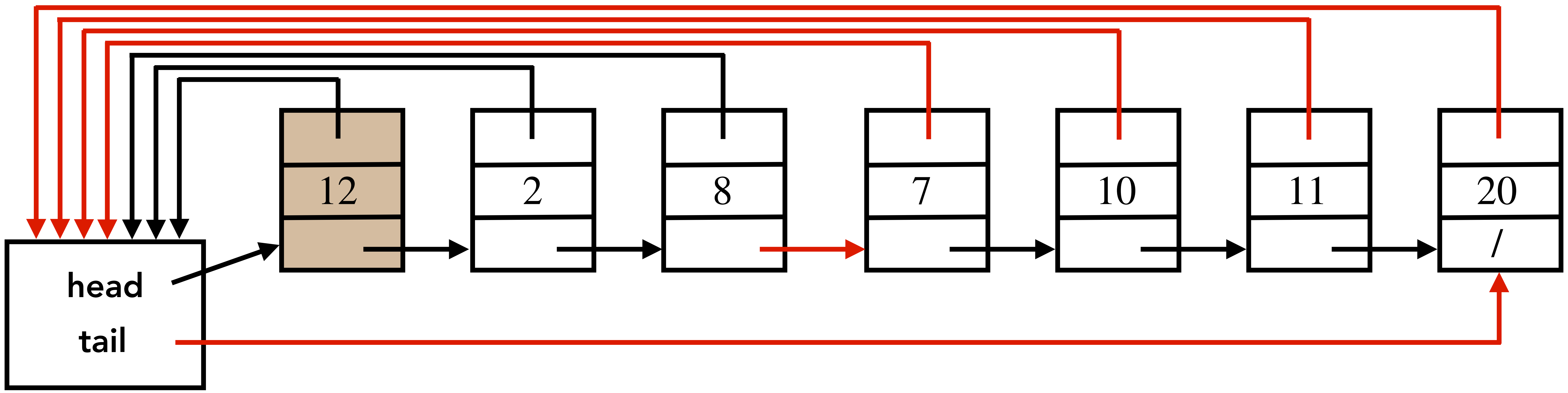


Heuristic: Appending the **shorter list** at the end of the **longer list**, will make **Union** faster.

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

Disjoint-Sets as Linked Lists

$S_1 \cup S_2$

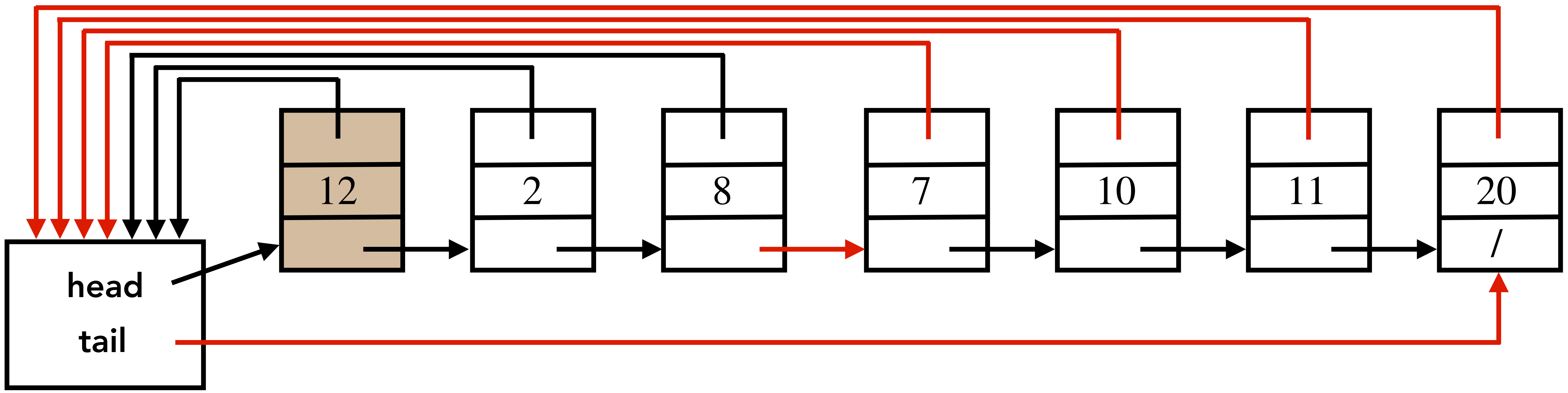


Heuristic: Appending the **shorter list** at the end of the **longer list**, will make **Union** faster.

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

Disjoint-Sets as Linked Lists

$S_1 \cup S_2$

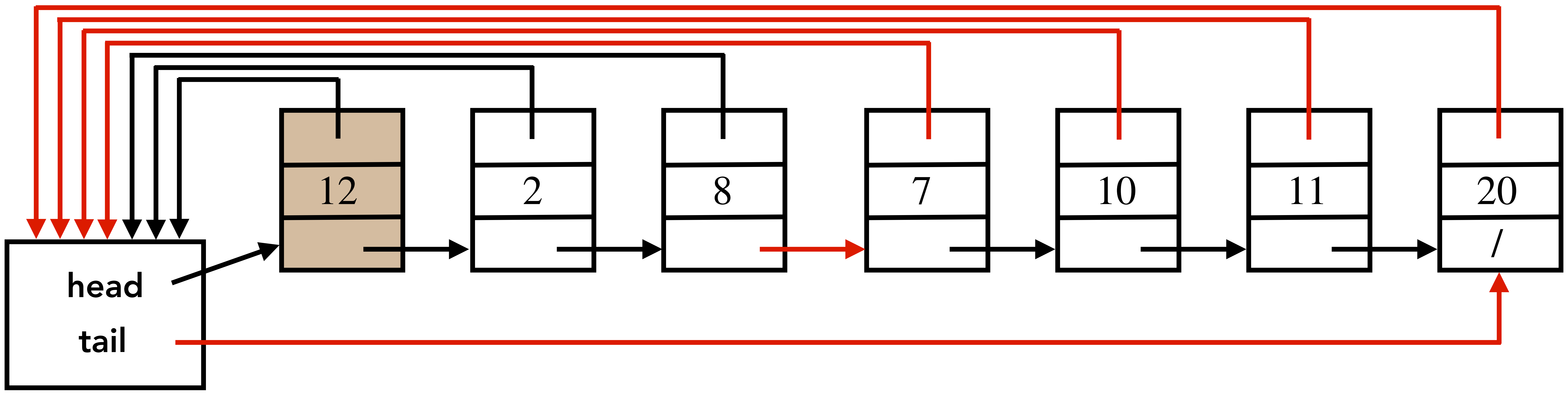


Heuristic: Appending the **shorter list** at the end of the **longer list**, will make **Union** faster.

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.