

# Distributed Systems

## COMP 212

Lecture 7

Othon Michail

# Leader Election

# Problem Statement

- *Elect a **unique leader processor** from among all the processors in the distributed system*
- Leader to be interpreted as:
  - coordinator
  - master processor
- Special case of **consensus/agreement**
- Processors should agree eventually on who they elect

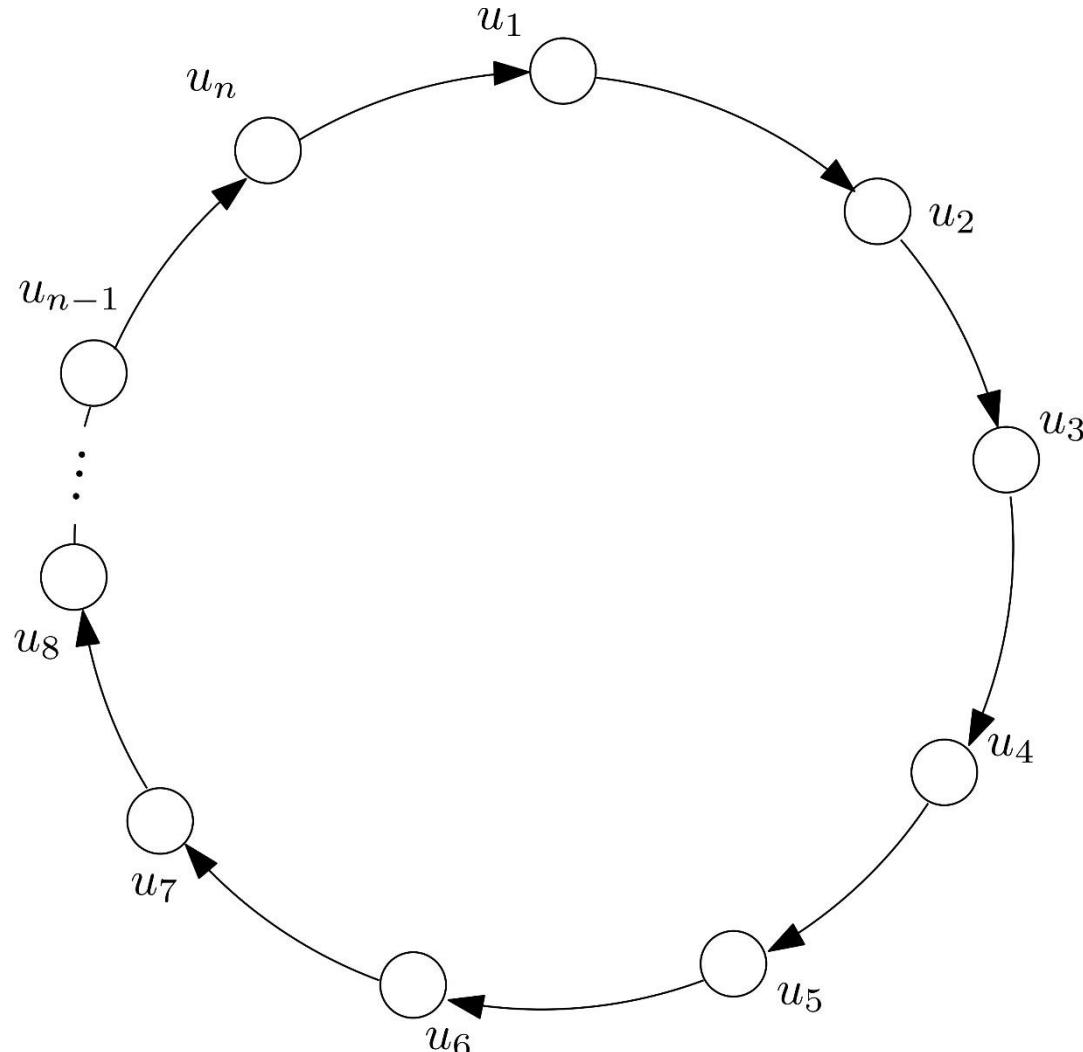
# Variants of Leader Election

- General network or special type of network, e.g., a ring
- Processors can be **identical** or have pre-assigned **unique ids**
- All processors may be required to
  - **know** the **elected** processor
  - to **output** the fact that themselves were **not elected**
  - to **terminate**
- Processors may possess in advance some **information about the network**
  - e.g., the number of processors  **$n$**  in the network

**Important remark:** Even slightly different assumptions can completely change the algorithmic solutions required or even make a problem impossible to solve

# The Ring Network

- Unidirectional (or directed)



# Leader Election with ids

- Processors have **unique ids** and **do not know  $n$**  in advance
- **LCR algorithm:** solves the problem
  - Le Lann, Chang, and Roberts [1977, 1979]
- Uses only **transmission** and **comparison** of ids
- **Simplest version:** Only the elected processor gives “output” and terminates
  - e.g., “I am the leader”
  - The other processors **never produce any output** and **do not terminate**

# LCR: Pseudocode

## Algorithm LCR

State of processor  $u_i$ :

- $myID_i$ : holds the processor's unique id
- $sendID_i$ : holds a unique id to be sent or *null*
- $status_i \in \{\text{"unknown"}, \text{"leader"}\}$ : indicates whether  $u_i$  has been elected ("leader") or not ("unknown")

# LCR: Pseudocode

## Algorithm LCR

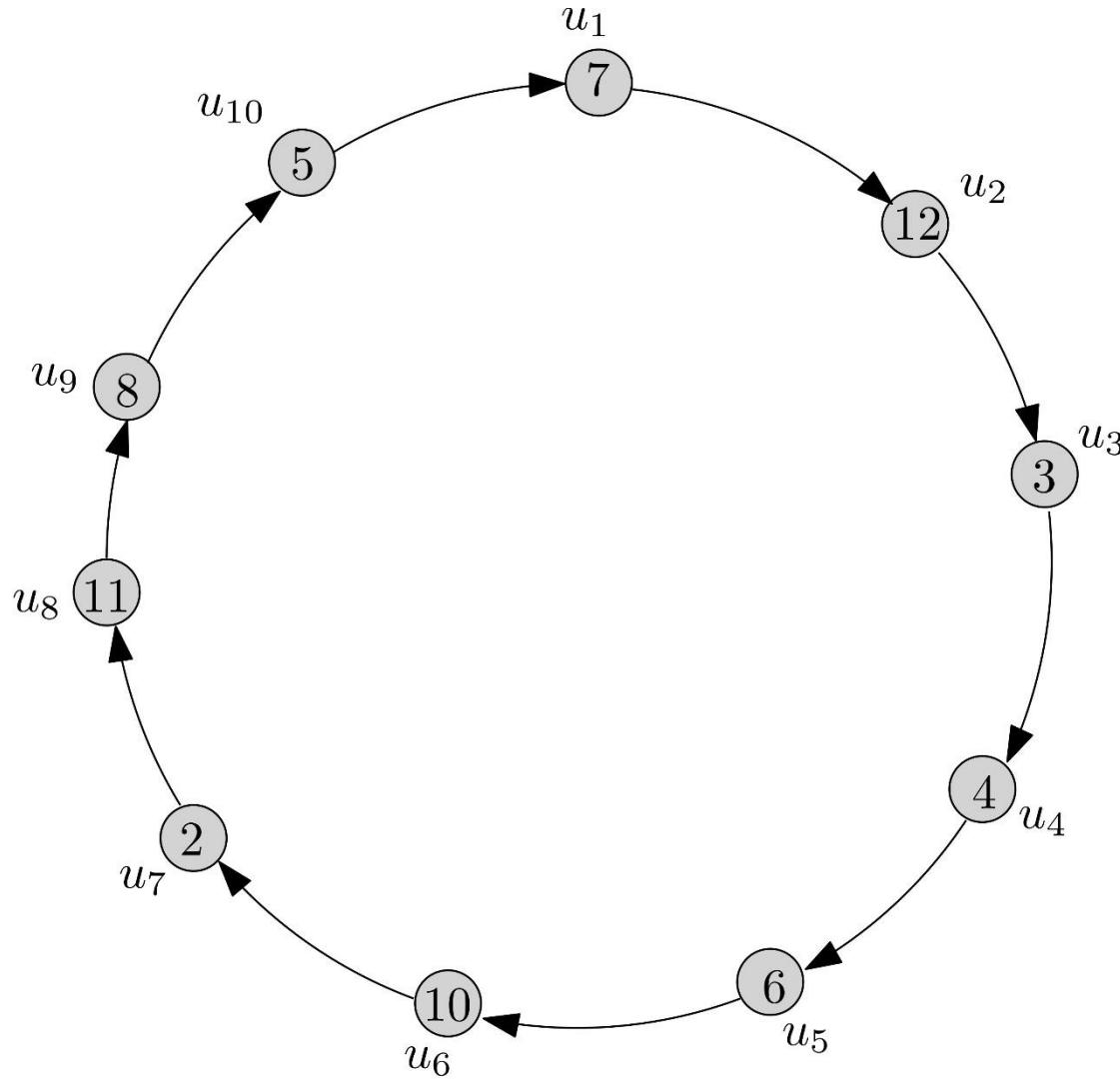
Code for processor  $u_i$ ,  $i \in \{1, 2, \dots, n\}$ :

Initially:

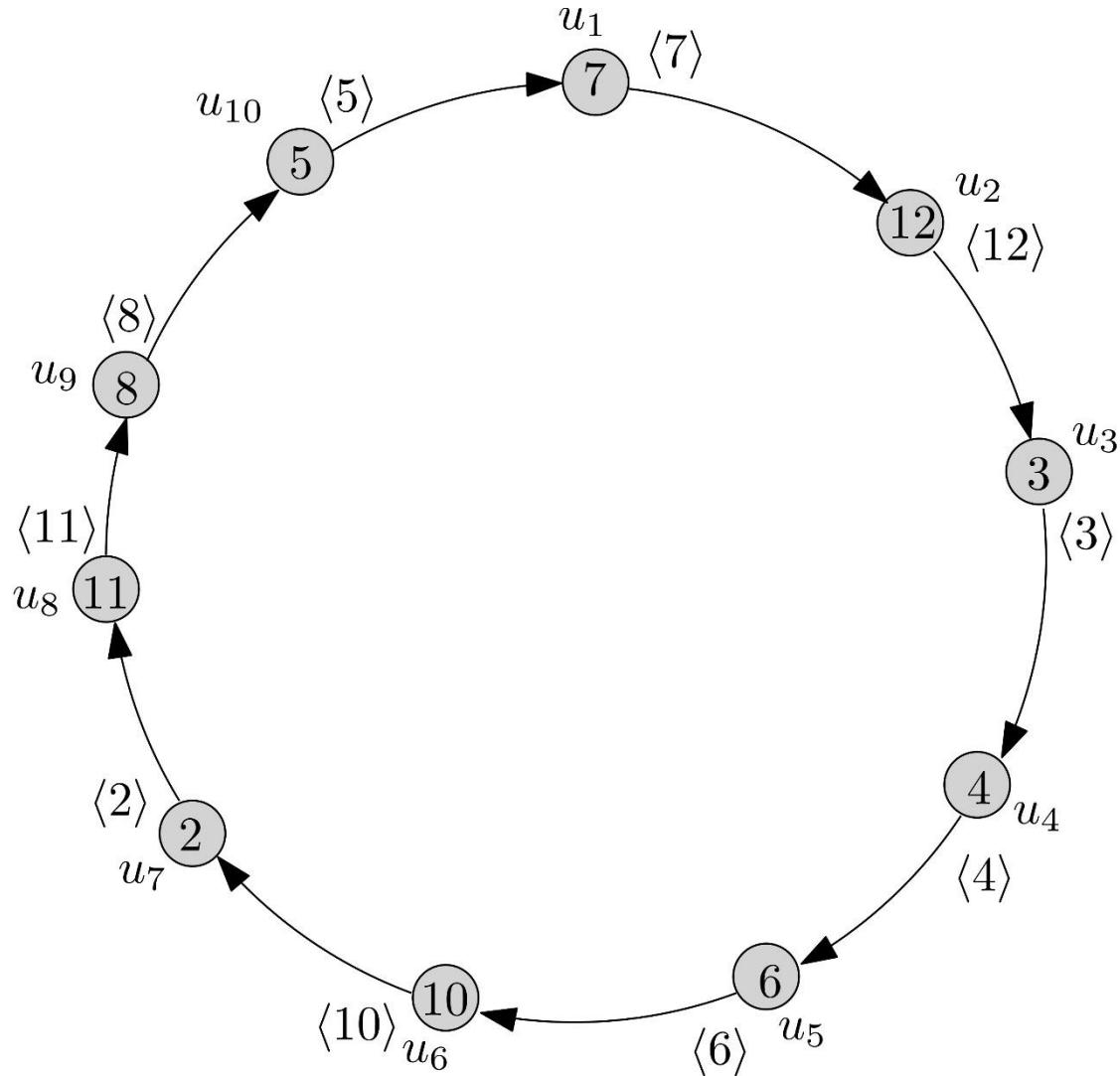
$u_i$  knows its own unique id stored in  $myID_i$ ,  
 $sendID_i := myID_i$ ,  
 $status_i := "unknown"$

```
if round = 1 then
    send ⟨sendIDi⟩ to unique out-neighbour
else // round > 1
    upon receiving ⟨inID⟩ from unique in-neighbour      // an id arriving from the left
        if inID > myIDi then                         // if greater than your own
            sendIDi := inID                           // forward it
            send ⟨sendIDi⟩ to unique out-neighbour
        else if inID = myIDi then                  // if equal to your own, your id managed a complete turn
            statusi := "leader"                      // therefore, elect yourself a leader
        else if inID < myIDi then                  // if smaller than own
            do nothing                                // ignore (discard) it
```

# Example Execution

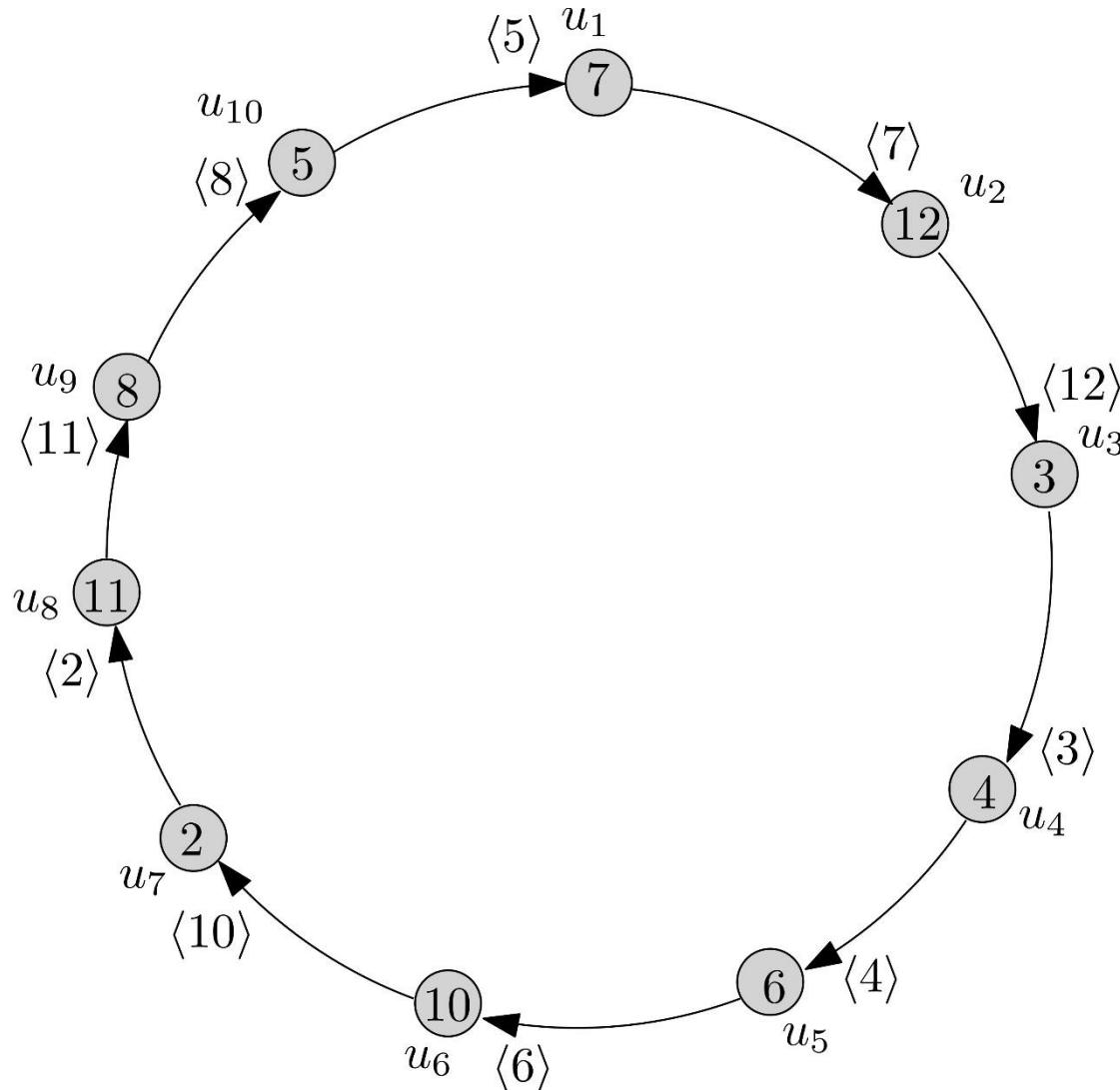


# Example Execution



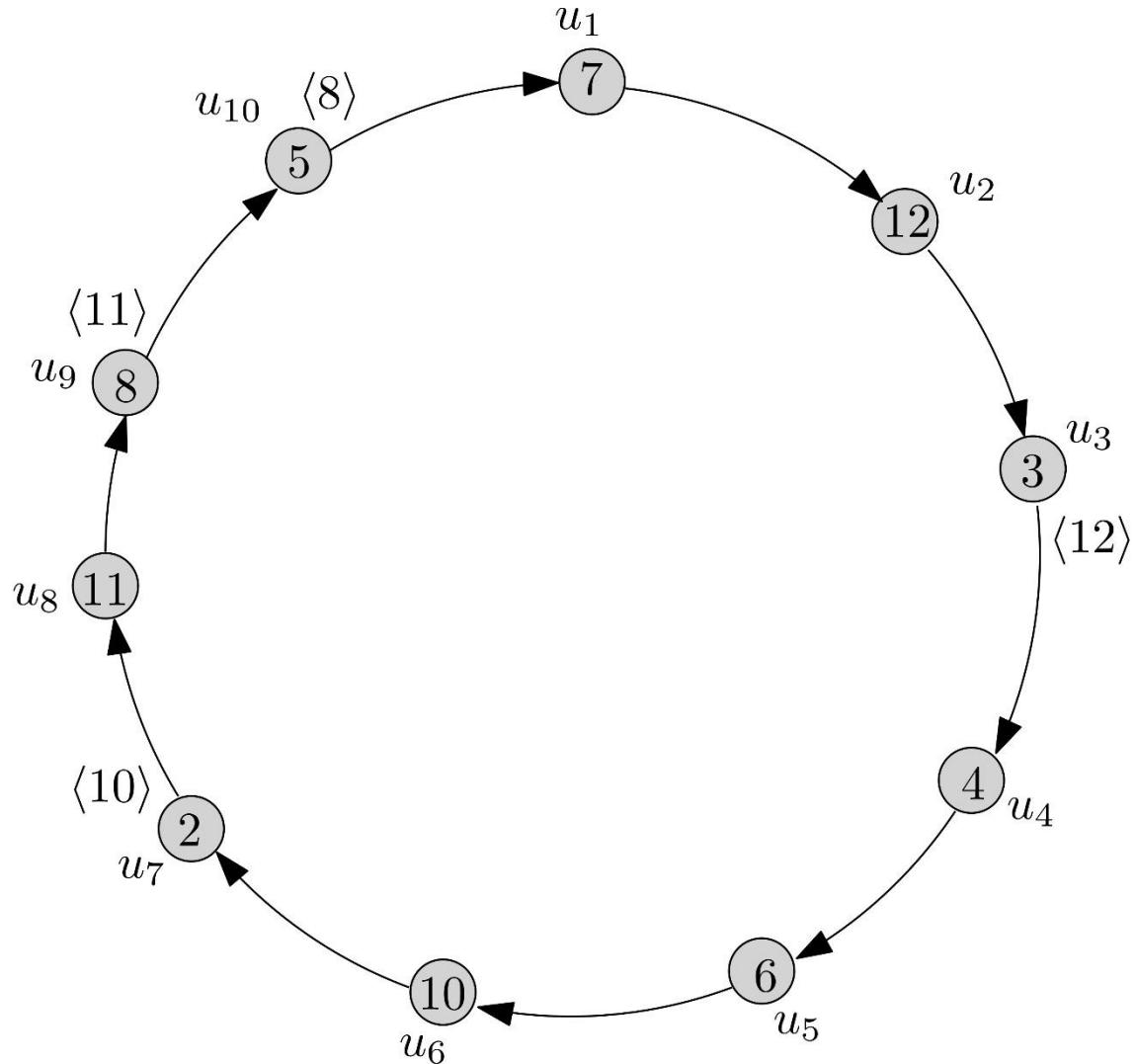
*round = 1*

# Example Execution



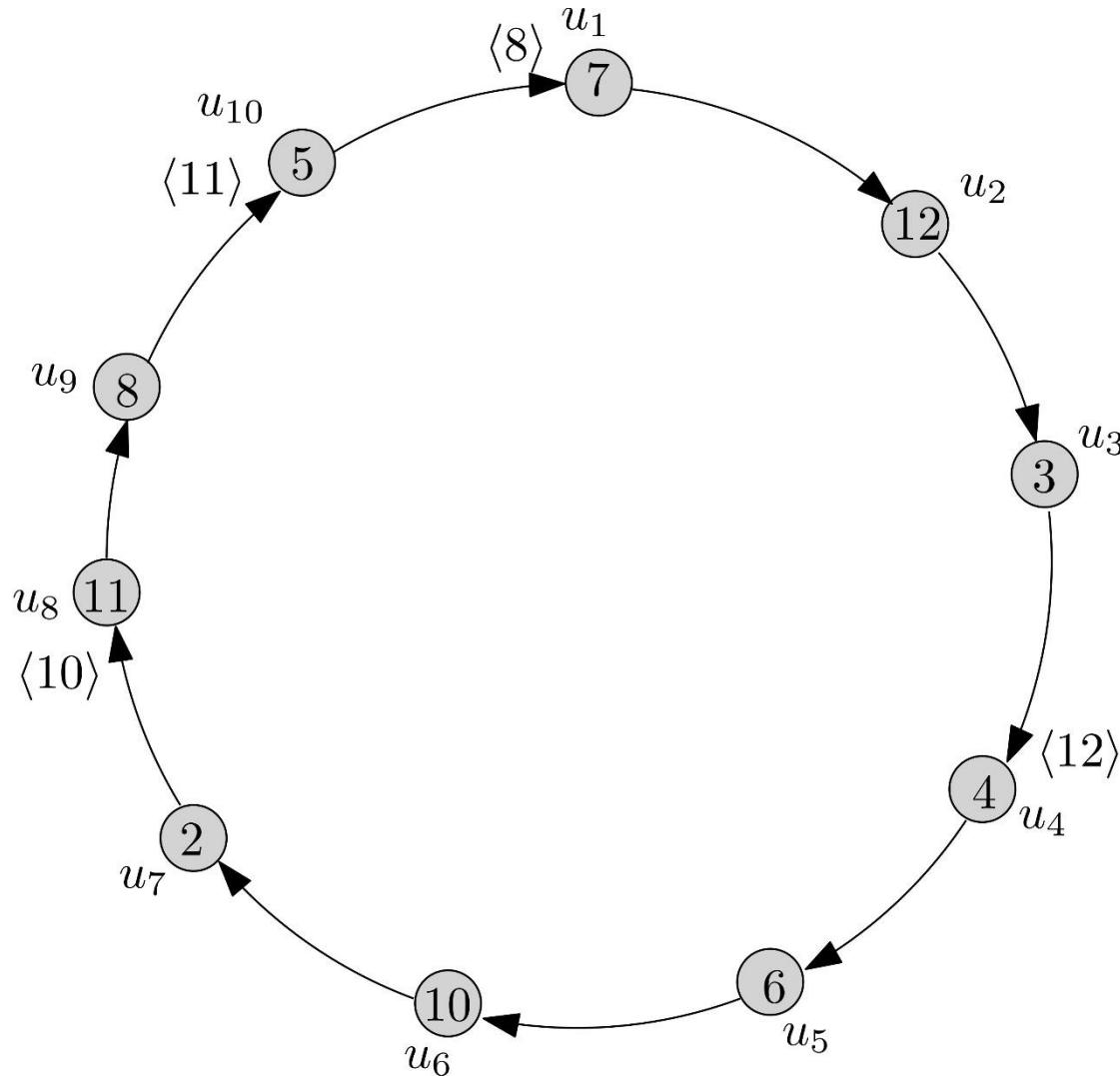
*round = 1*

# Example Execution



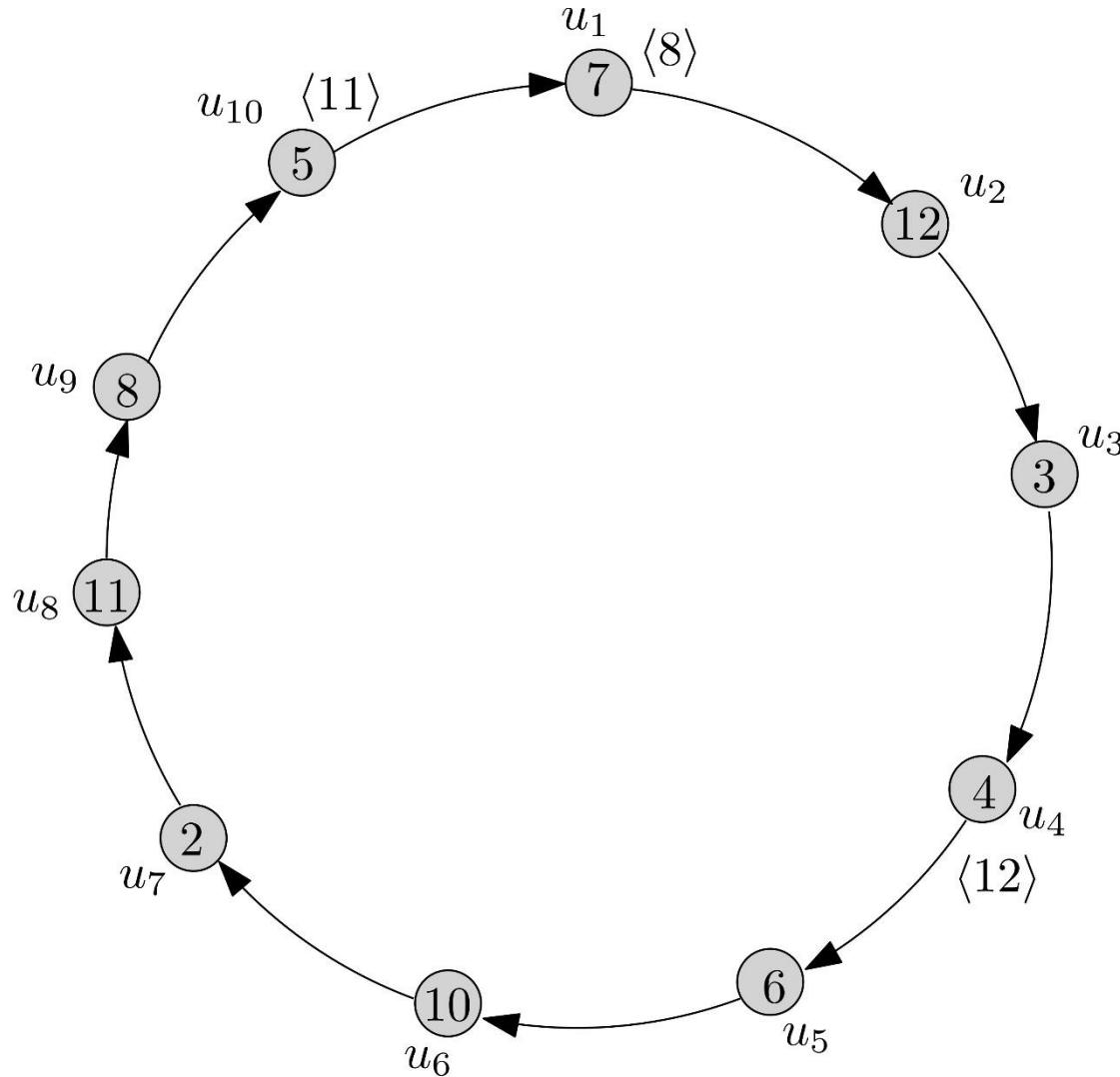
$round = 2$

# Example Execution



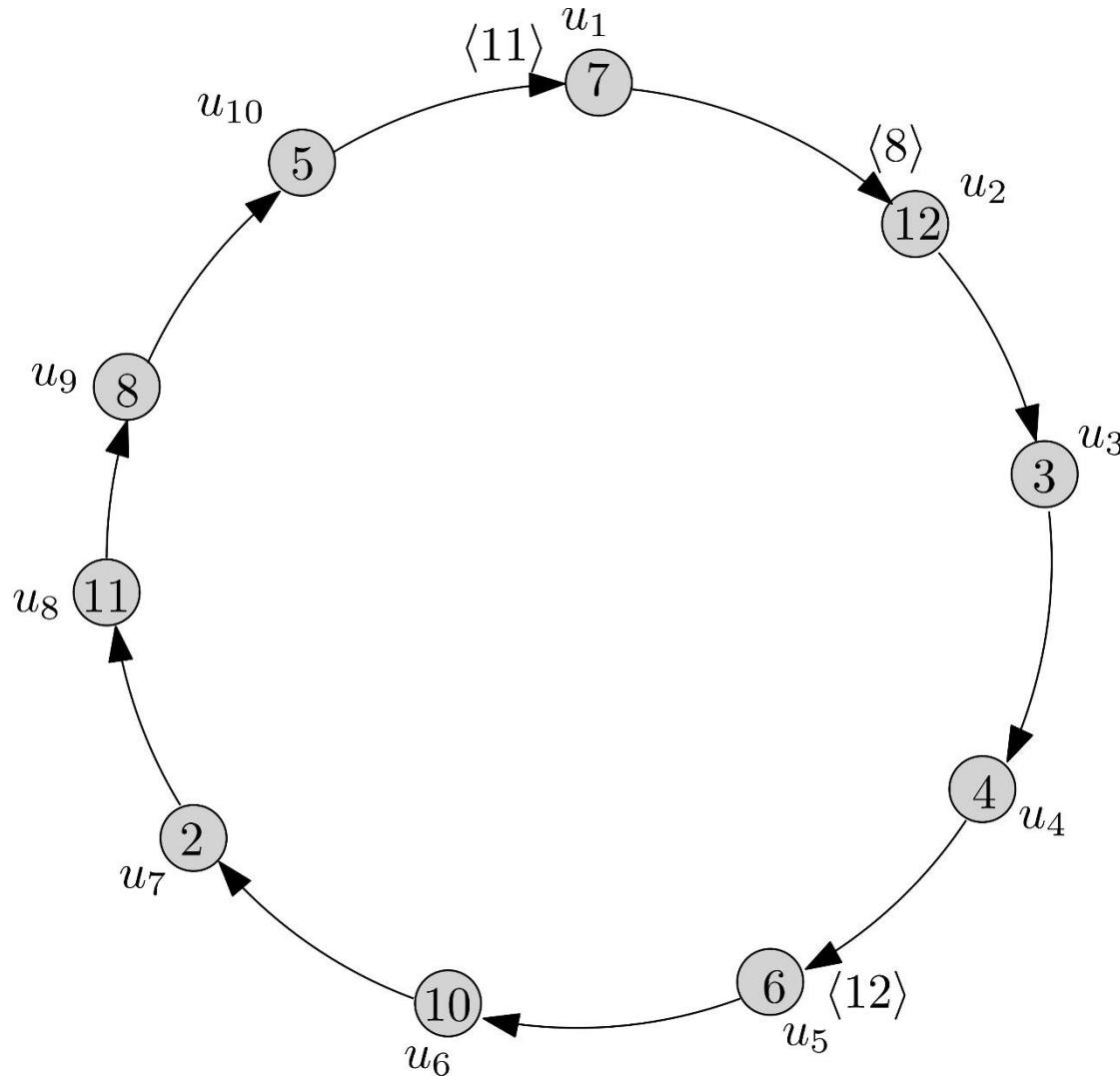
$round = 2$

# Example Execution



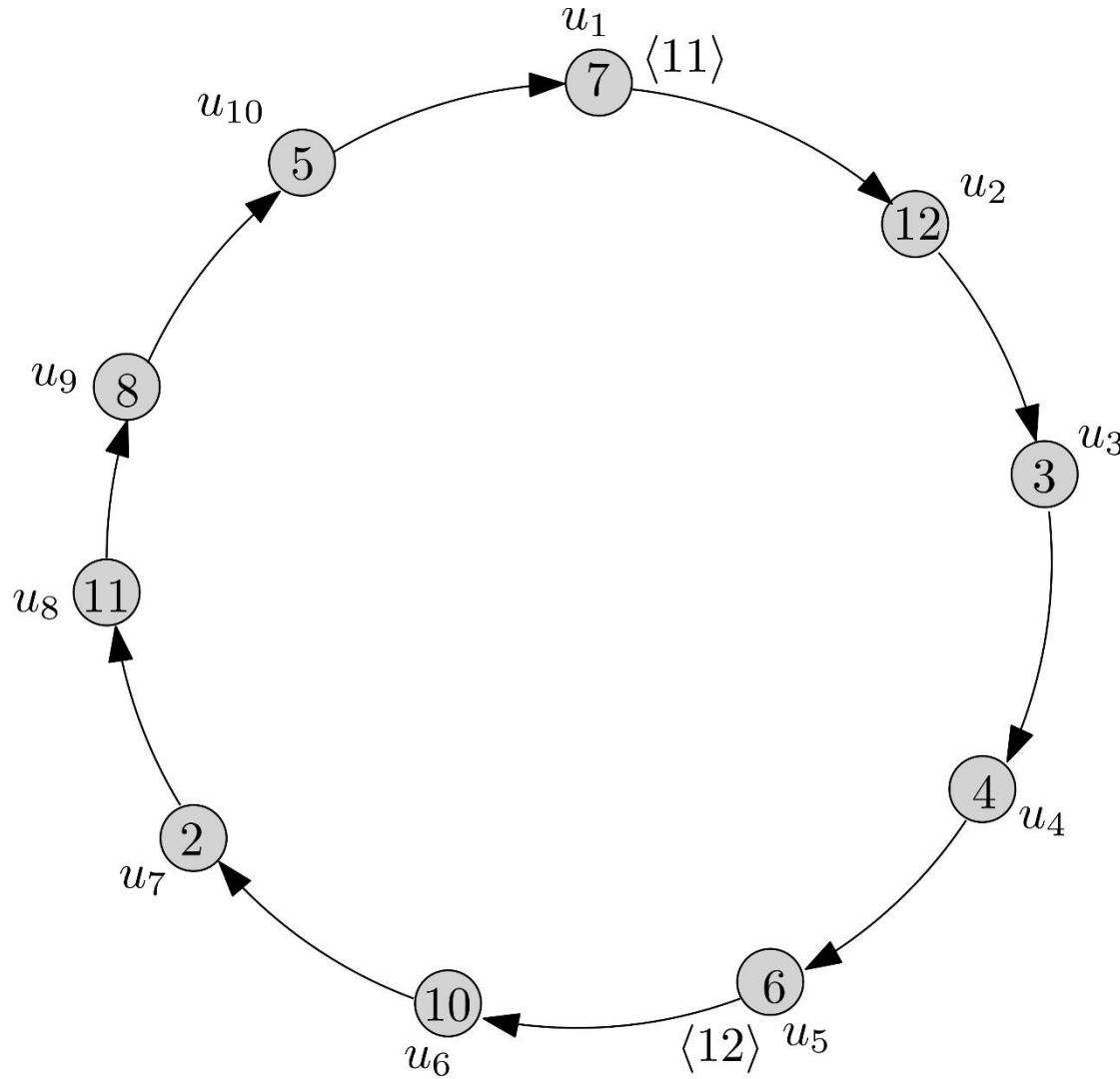
*round = 3*

# Example Execution



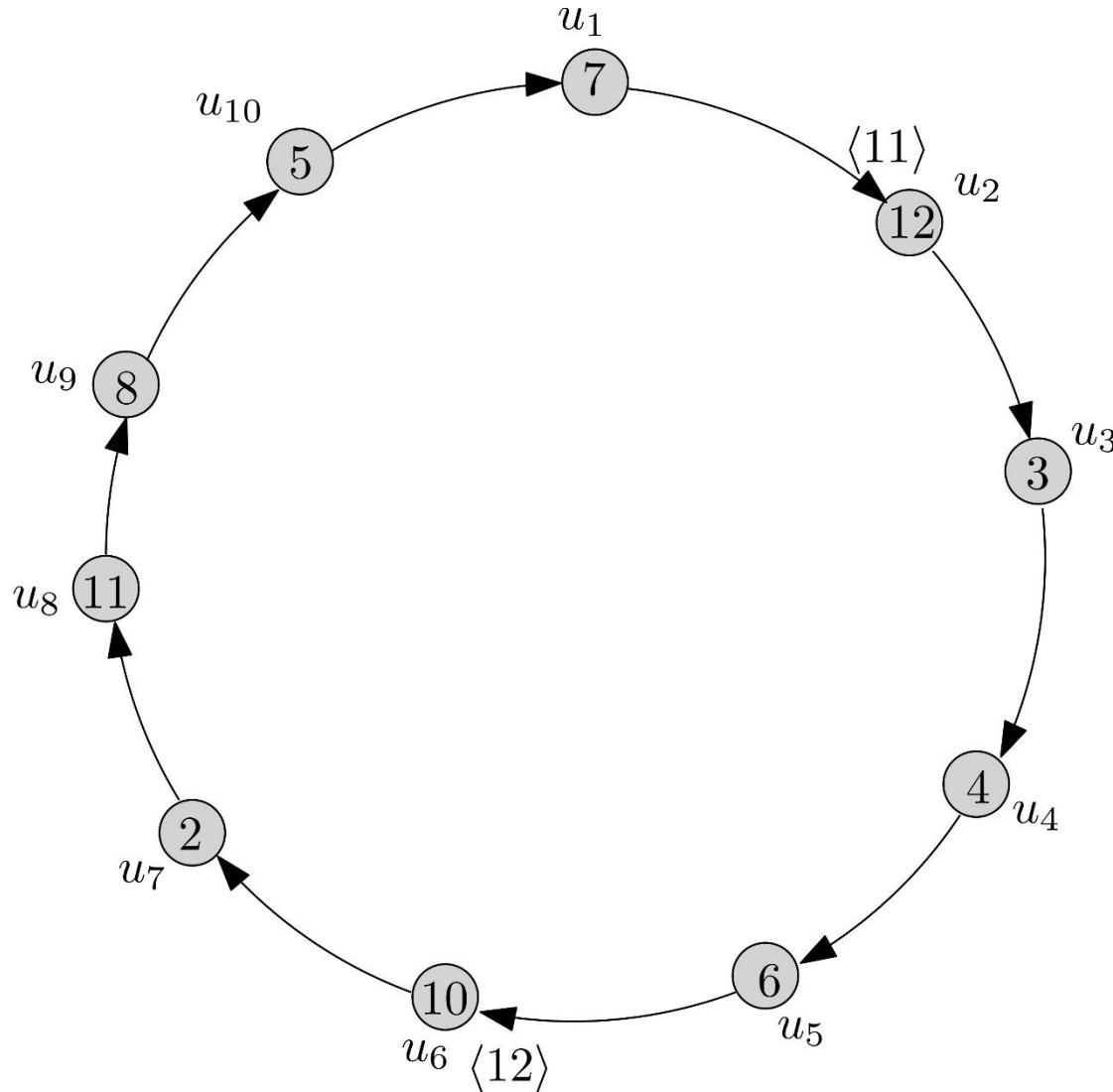
*round = 3*

# Example Execution



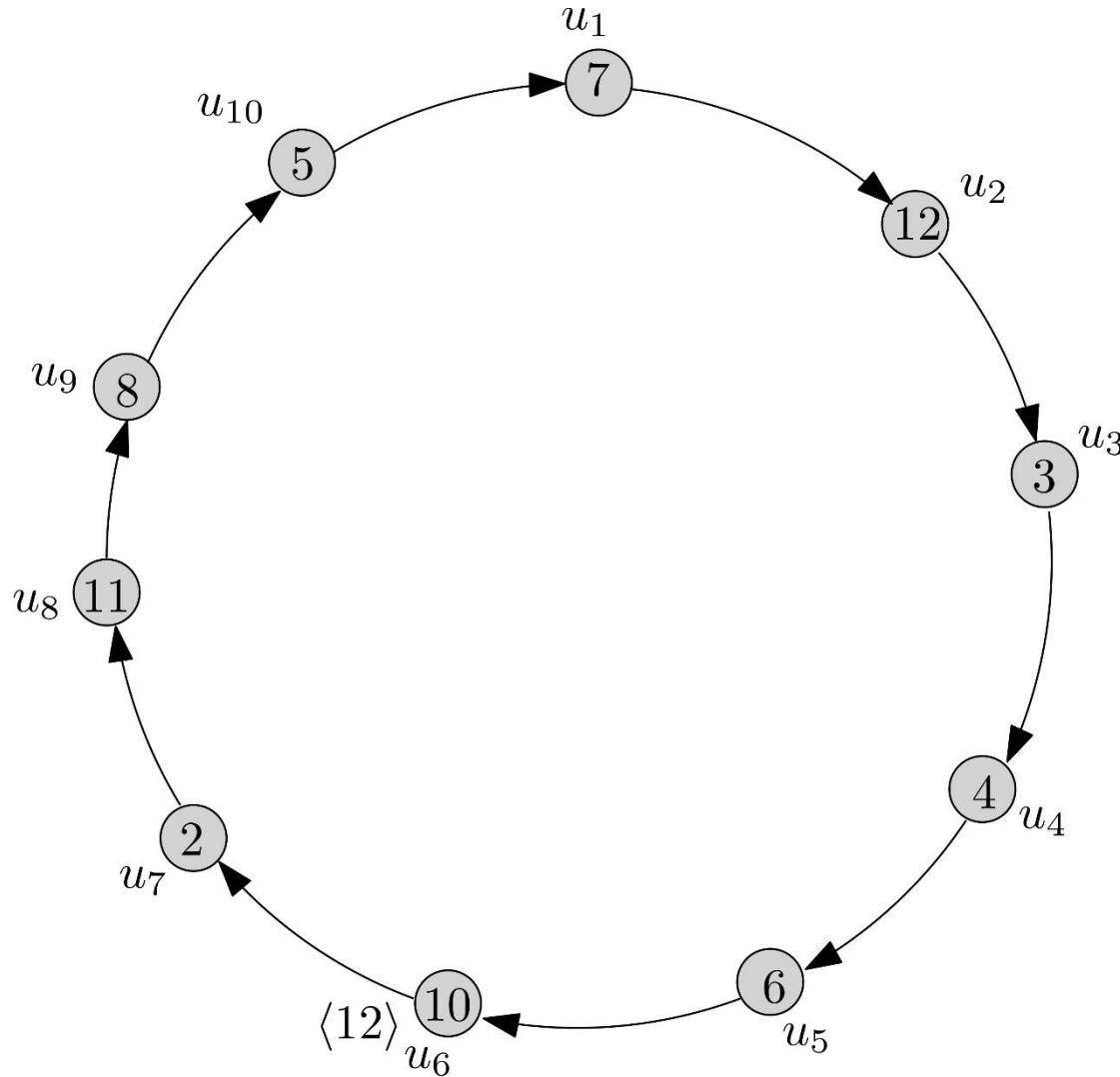
*round = 4*

# Example Execution



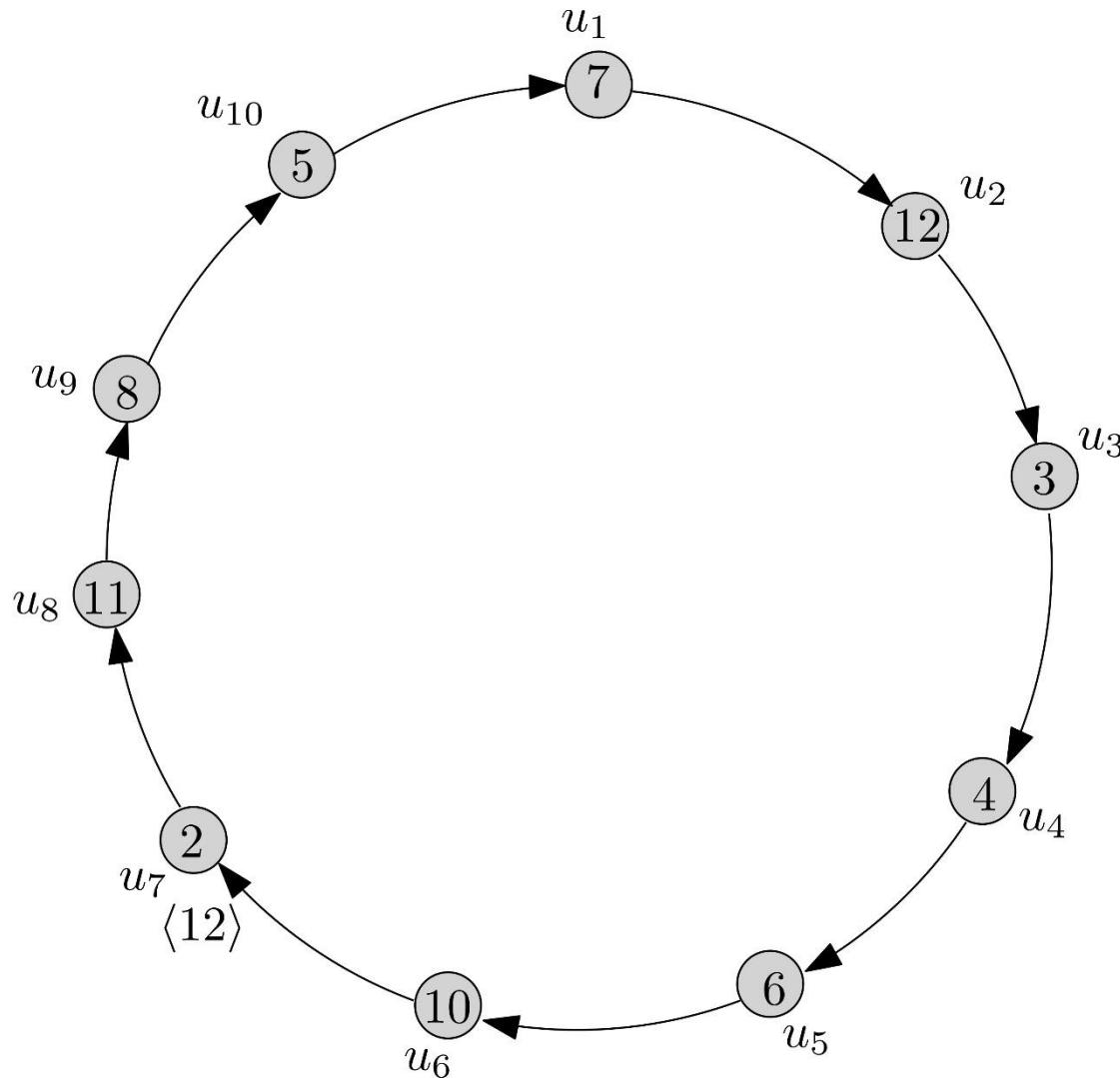
*round = 4*

# Example Execution



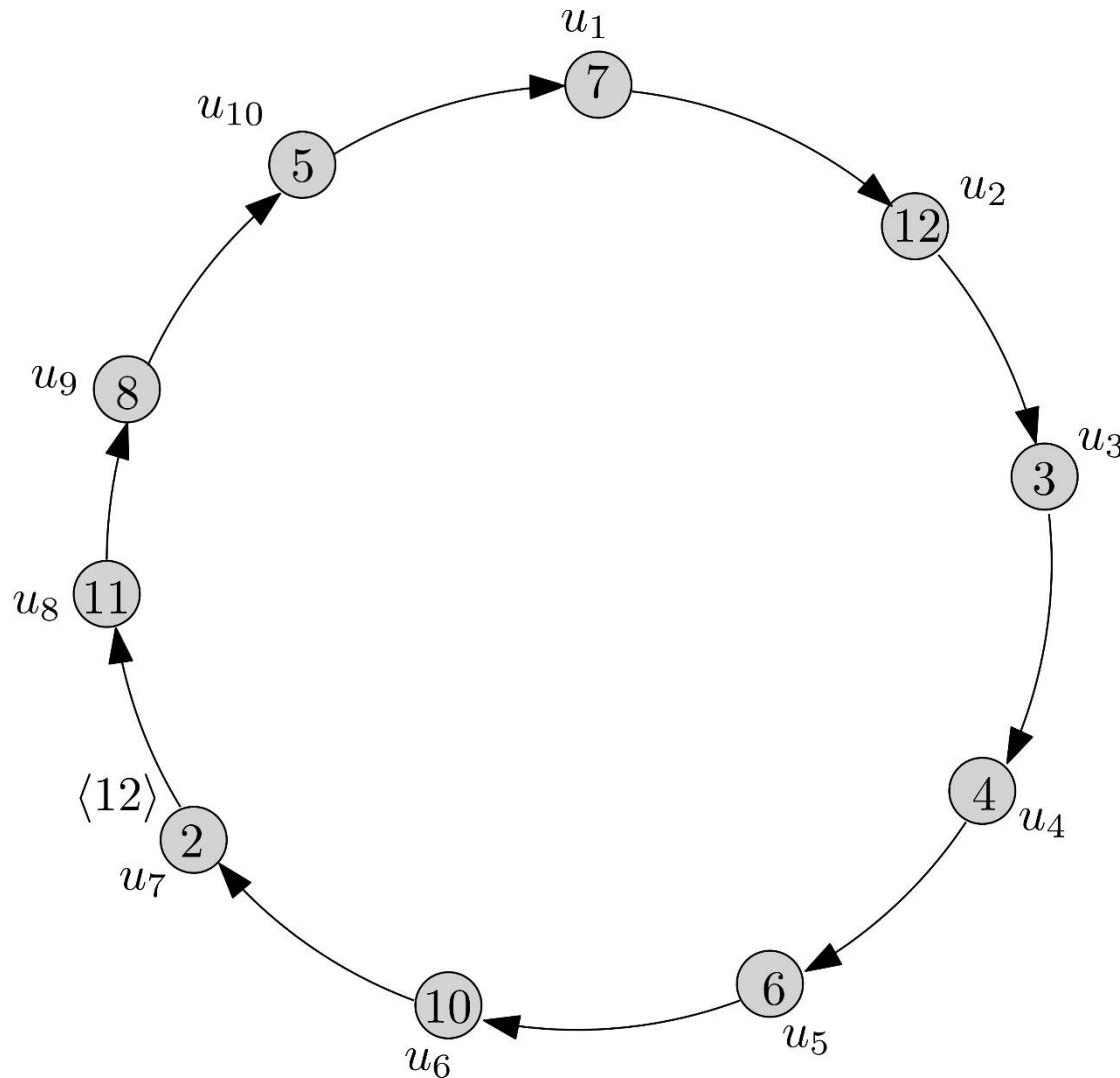
*round = 5*

# Example Execution



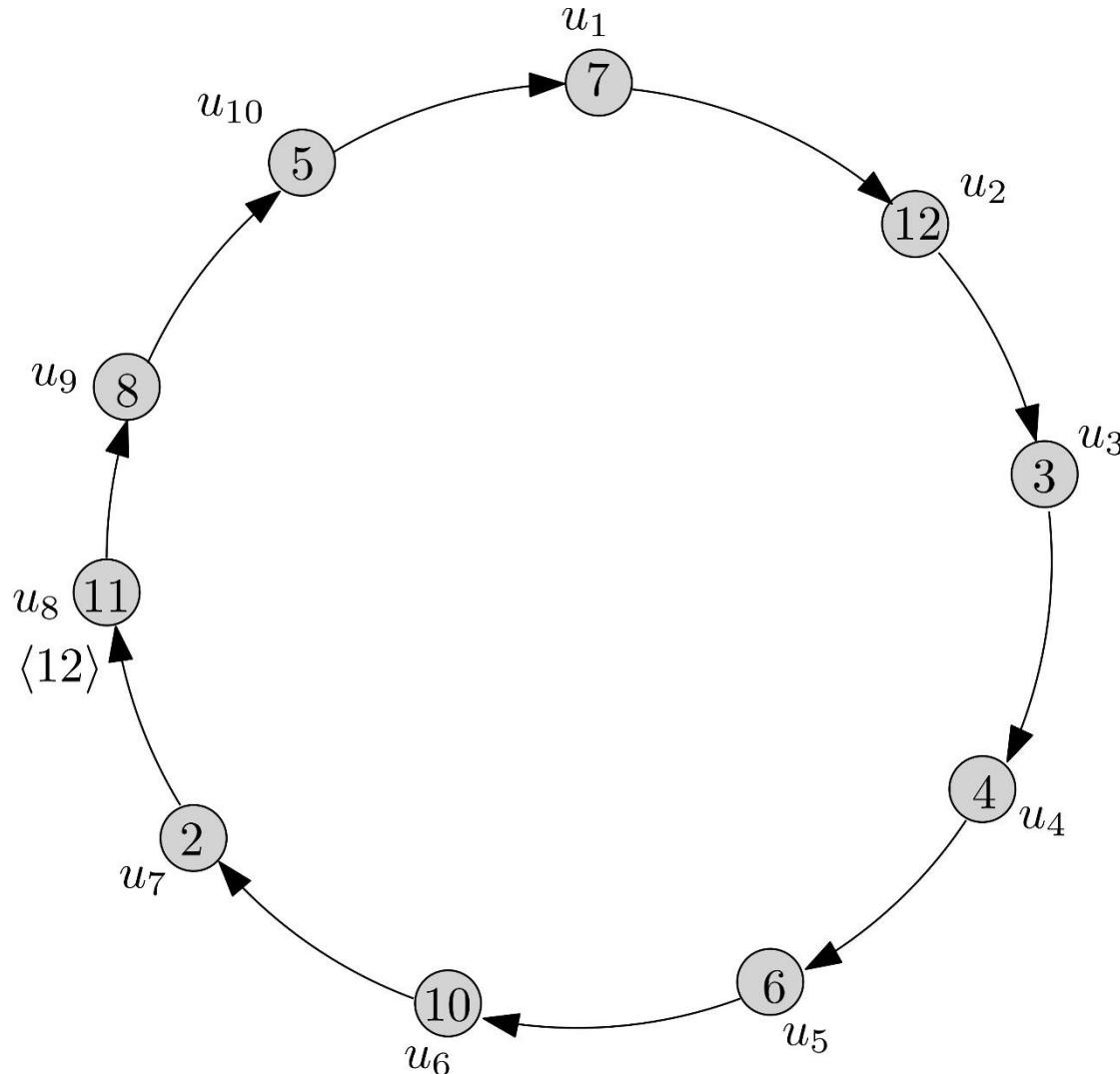
$round = 5$

# Example Execution



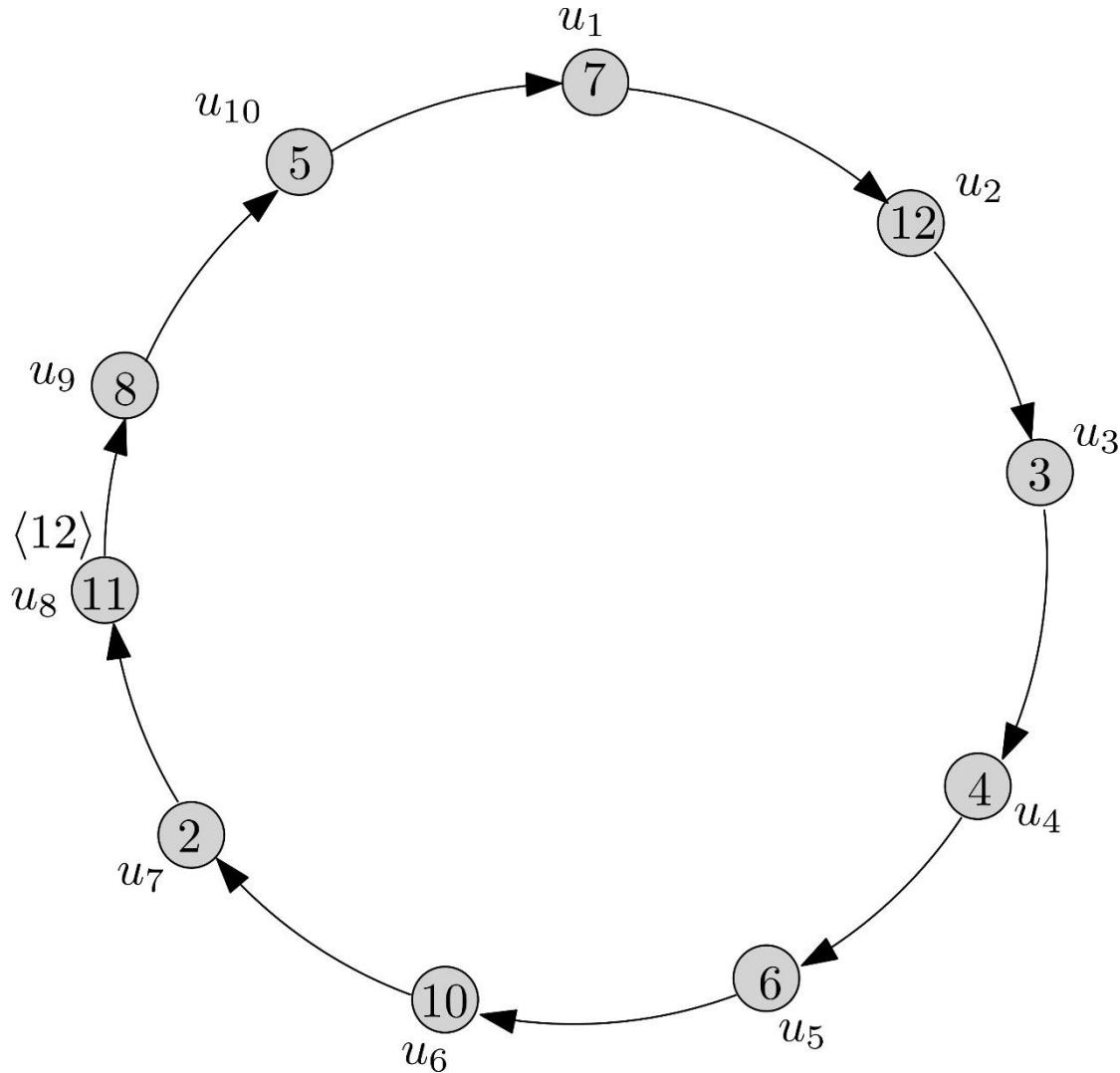
$round = 6$

# Example Execution



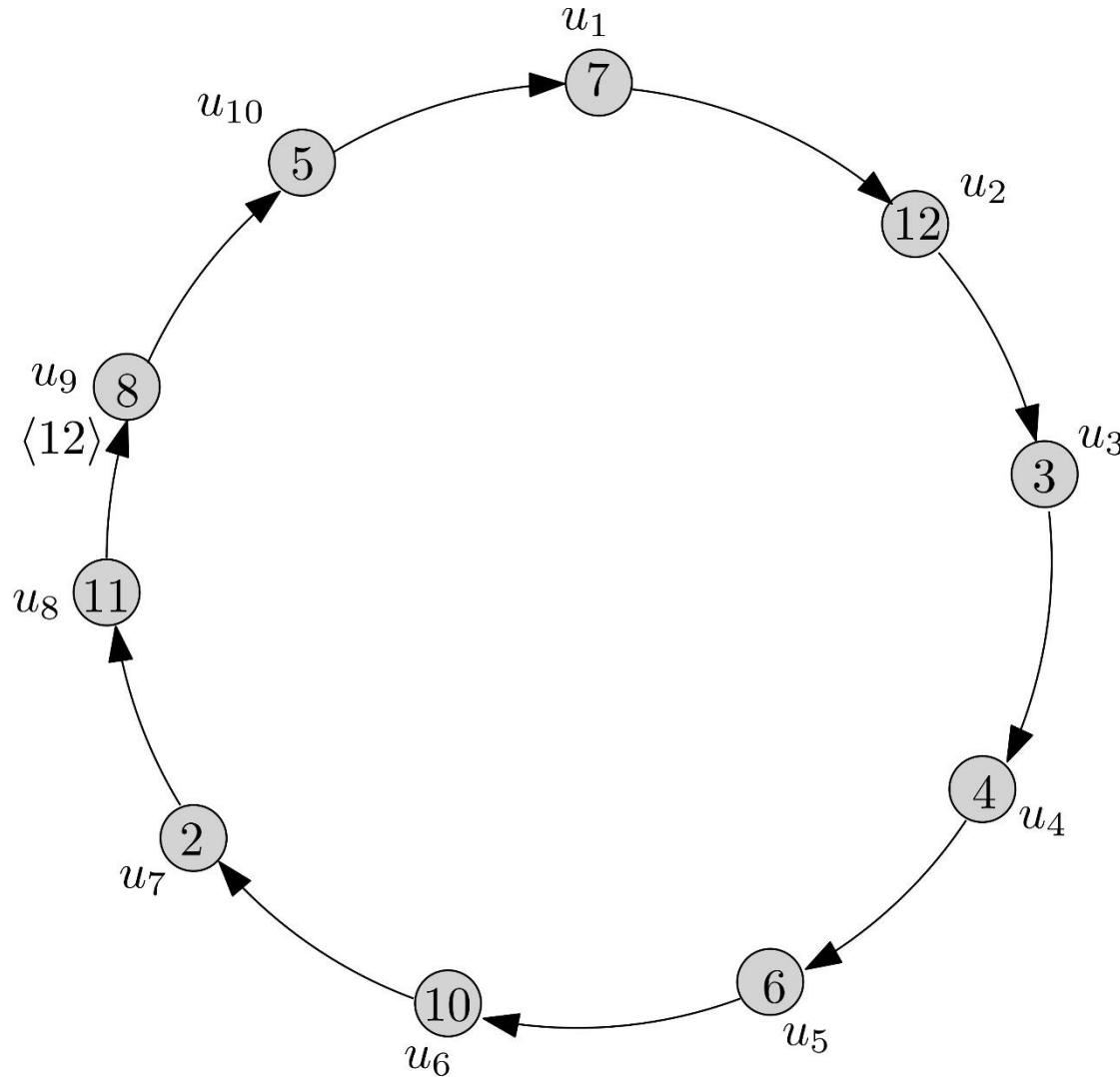
*round = 6*

# Example Execution



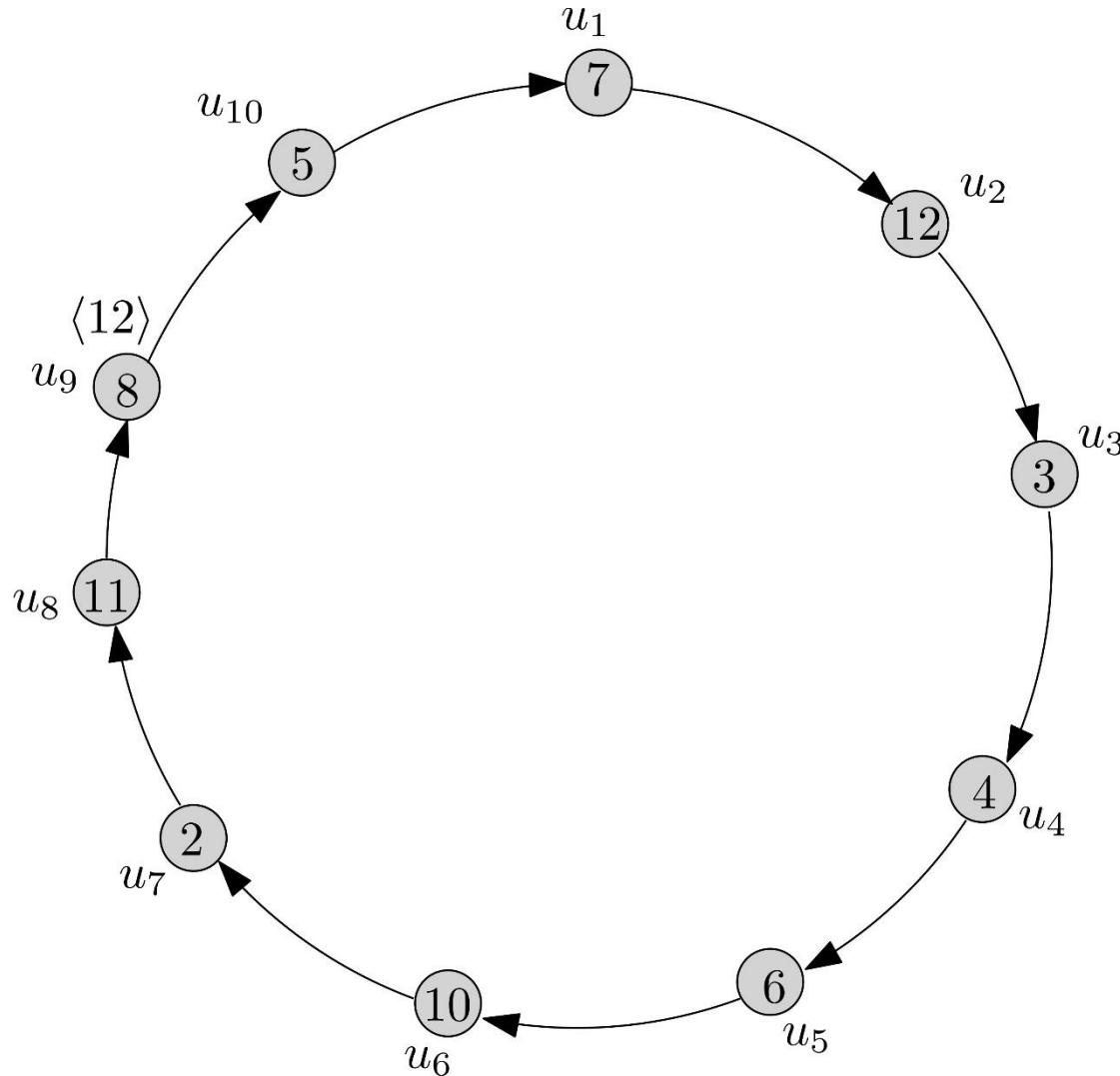
$round = 7$

# Example Execution



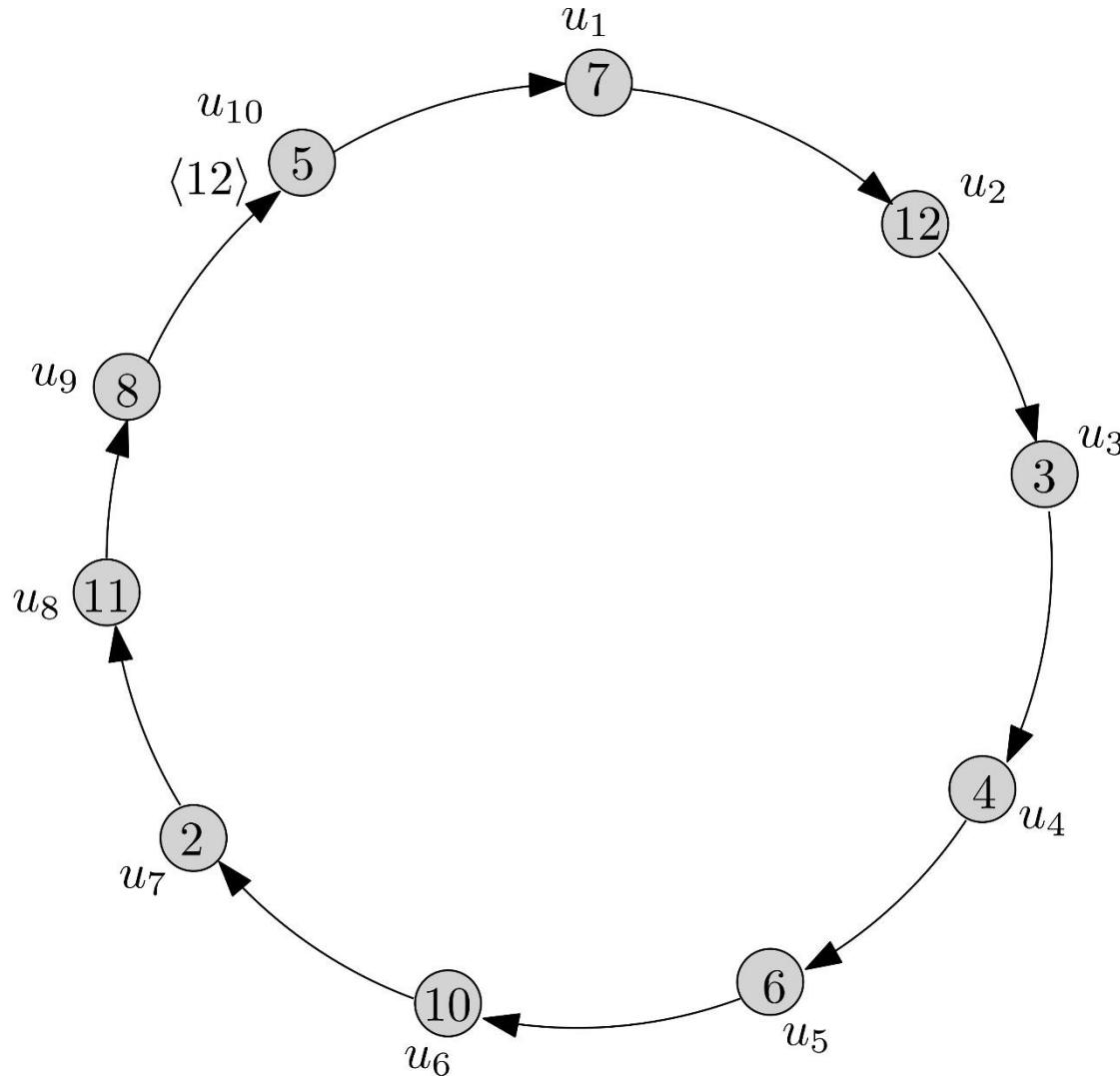
$round = 7$

# Example Execution



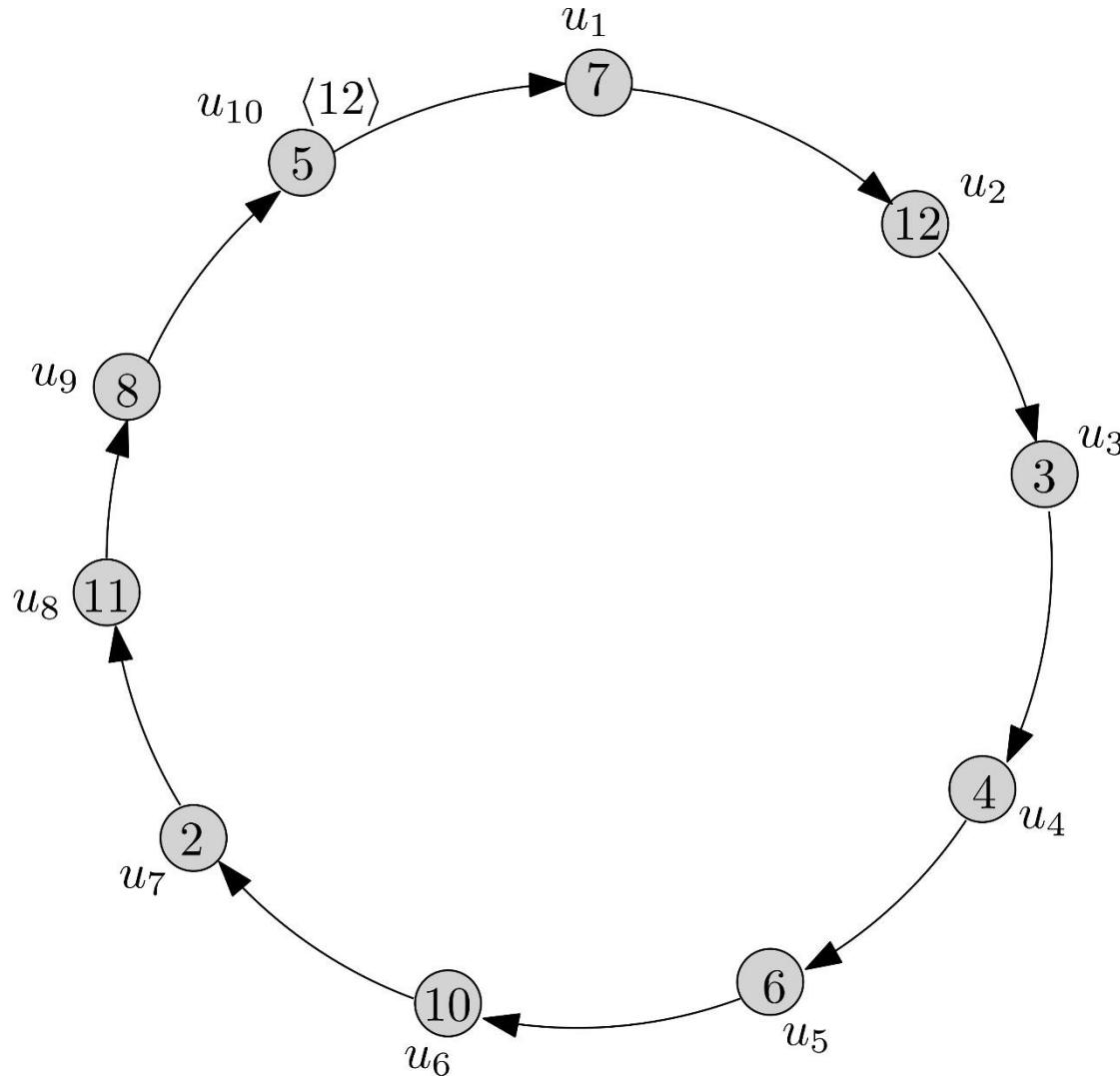
$round = 8$

# Example Execution



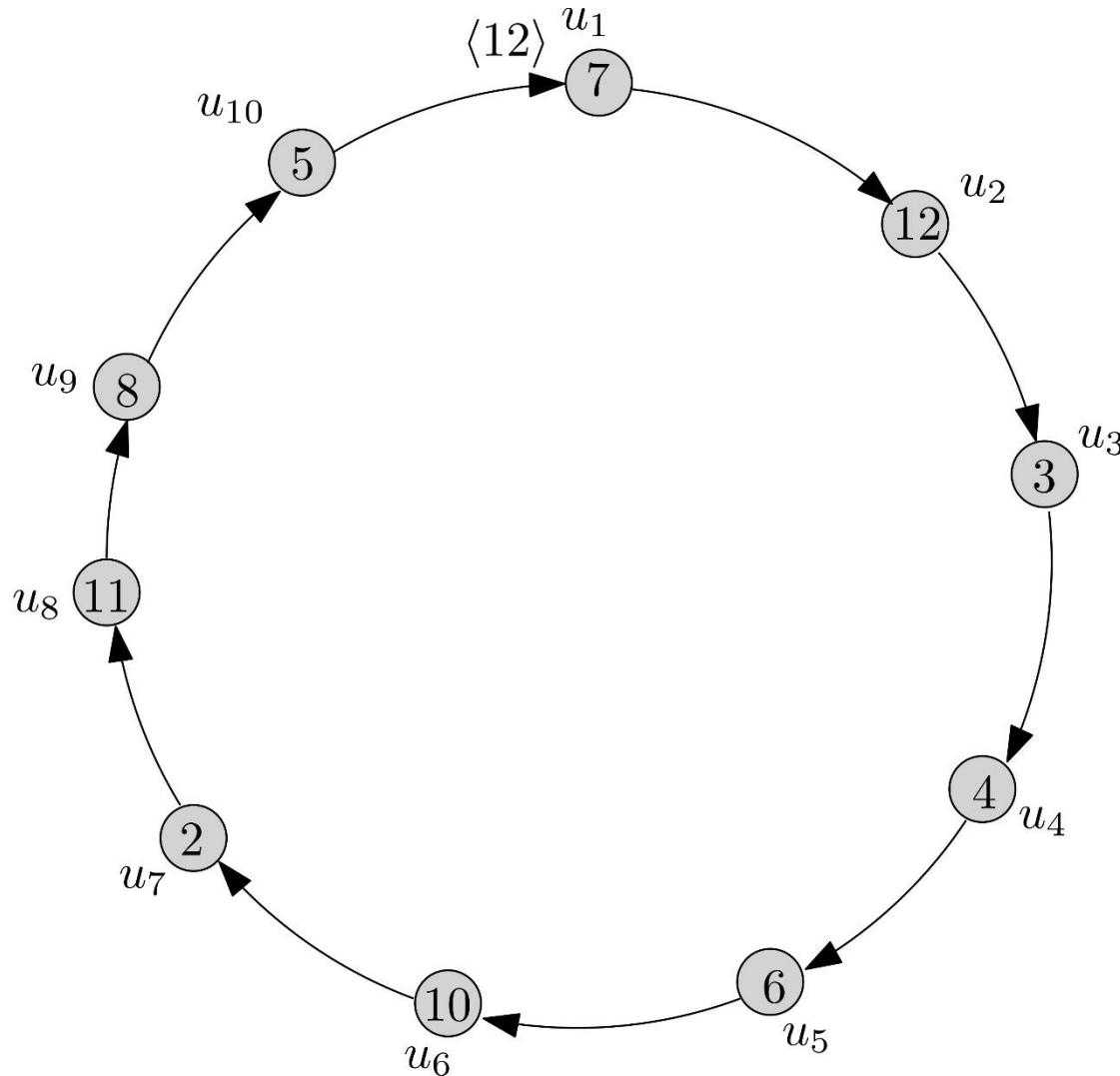
*round = 8*

# Example Execution



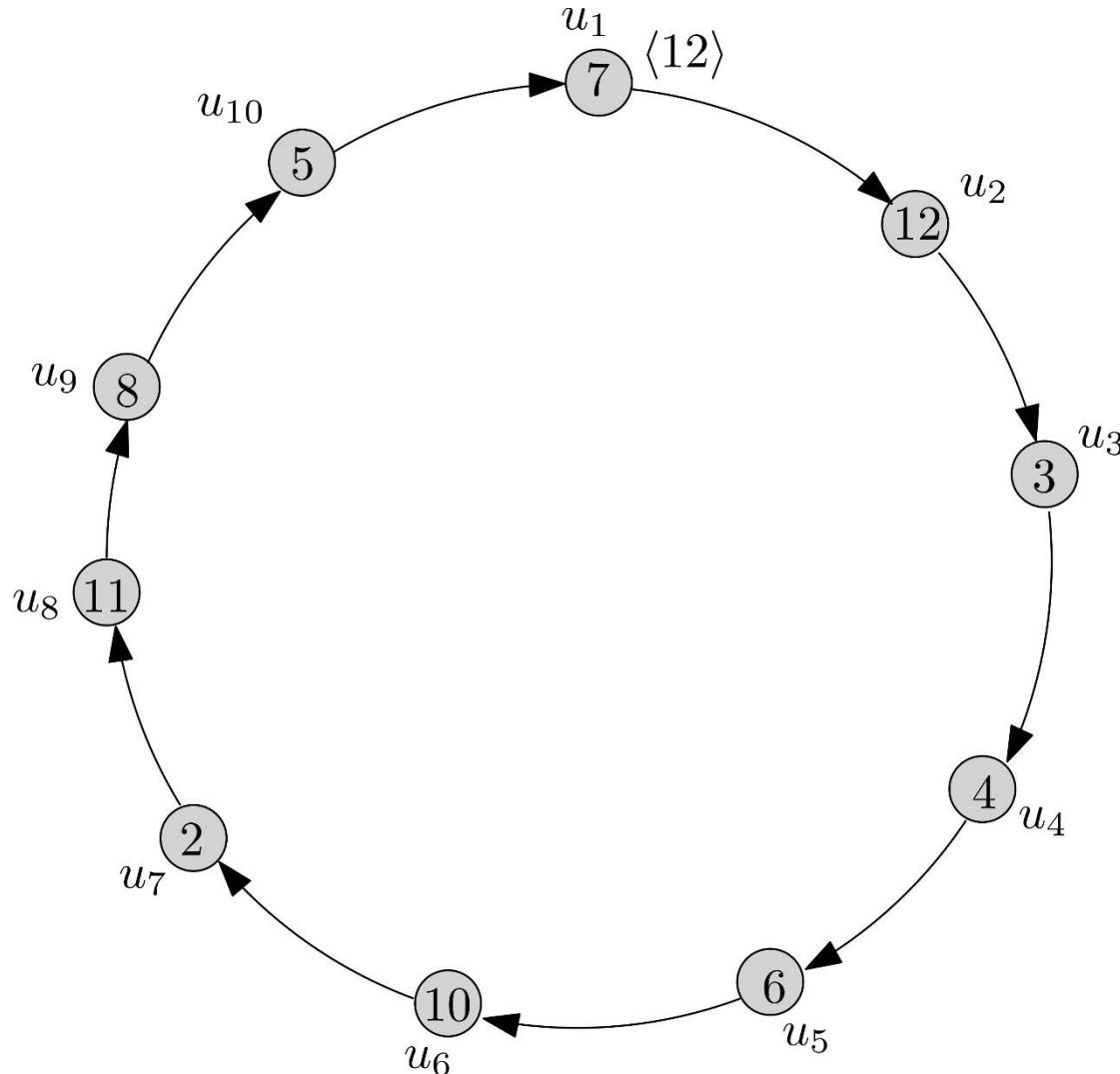
$round = 9$

# Example Execution



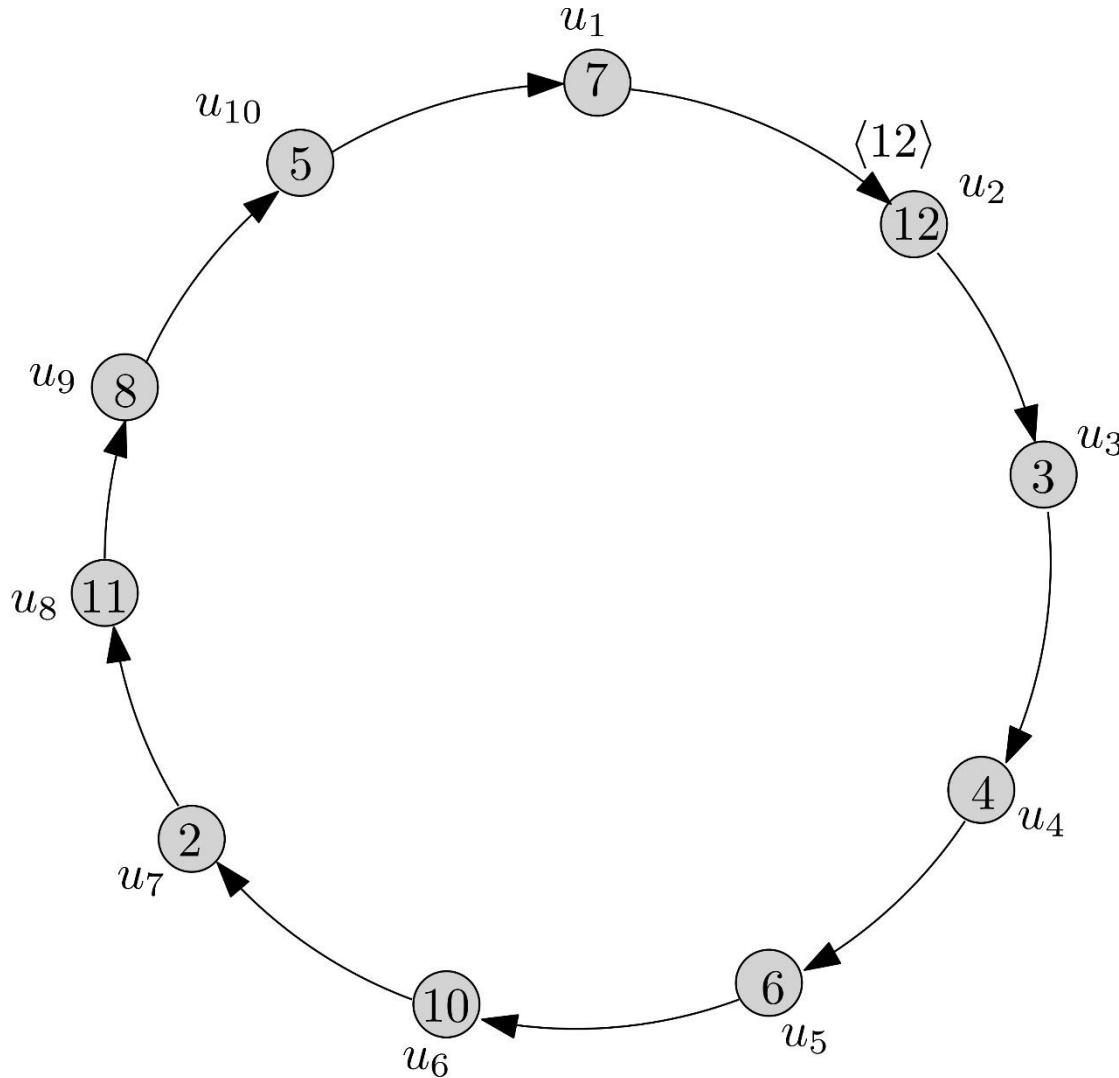
$round = 9$

# Example Execution



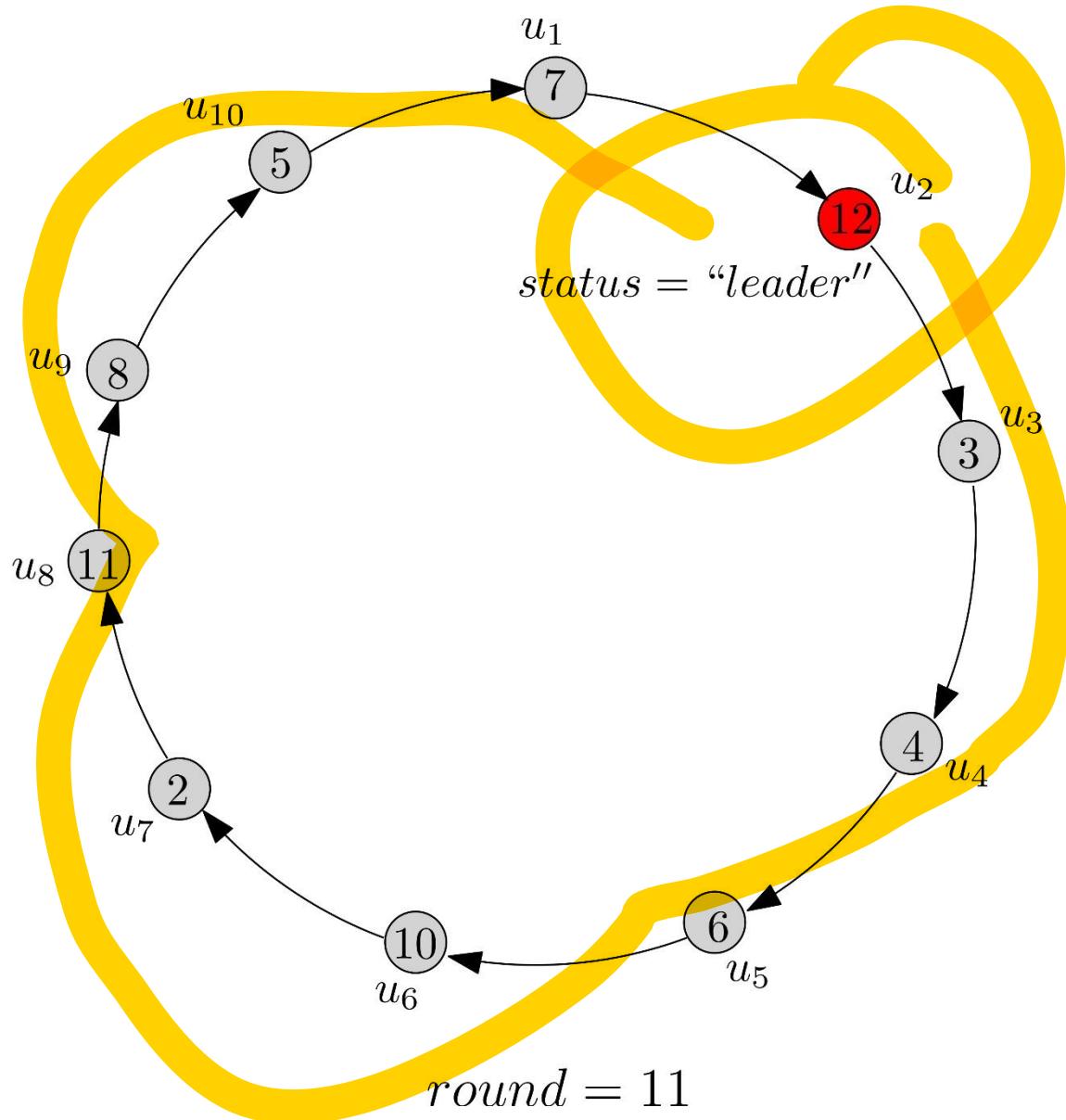
*round = 10*

# Example Execution



*round = 10*

# Example Execution



# Correctness and Complexity

- Correctness:
  - some processor is eventually elected
  - never 2 or more processors are elected
- Time complexity:
  - $n + 1$  rounds (or  $n$  depending on the round model)
- Communication complexity:
  - size of messages: encoding in bits of the maximum id
  - $O(n^2)$  messages in the worst case
    - *Can you think which is the worst case for this algorithm?*
- *How can we make all nodes terminate and know the elected leader and what will be the additional effect in performance?*

Think of these and we shall prove them in class

# LCR Correctness

- We have to show that:
  - Exactly one processor  $u_i$  eventually sets  $status_i := "leader"$
  - We know that the algorithm aims to elect the processor with the maximum id
    - Call it  $u_{max}$
- Suffices to show that:
  - $u_{max}$  eventually outputs “leader”
  - No other  $u_i$  ever does this

# LCR Correctness

**Lemma.**  $u_{max}$  outputs “leader” in round  $n + 1$ .

**Proof.** We observe that  $id_{max}$  (i.e., the id of  $u_{max}$ ) makes progress in every round.

- No processor with larger id to discard it

Therefore, in round 1 it is transmitted over the 1<sup>st</sup> clockwise edge, in round 2 over the 2<sup>nd</sup>, in round  $n$  over the  $n$ th, i.e., received back by  $u_{max}$ .

□

- Can be made completely formal by induction on the number of rounds  $r$

# LCR Correctness

**Lemma.** No processor  $u_i$  other than  $u_{max}$  ever outputs “*leader*”.

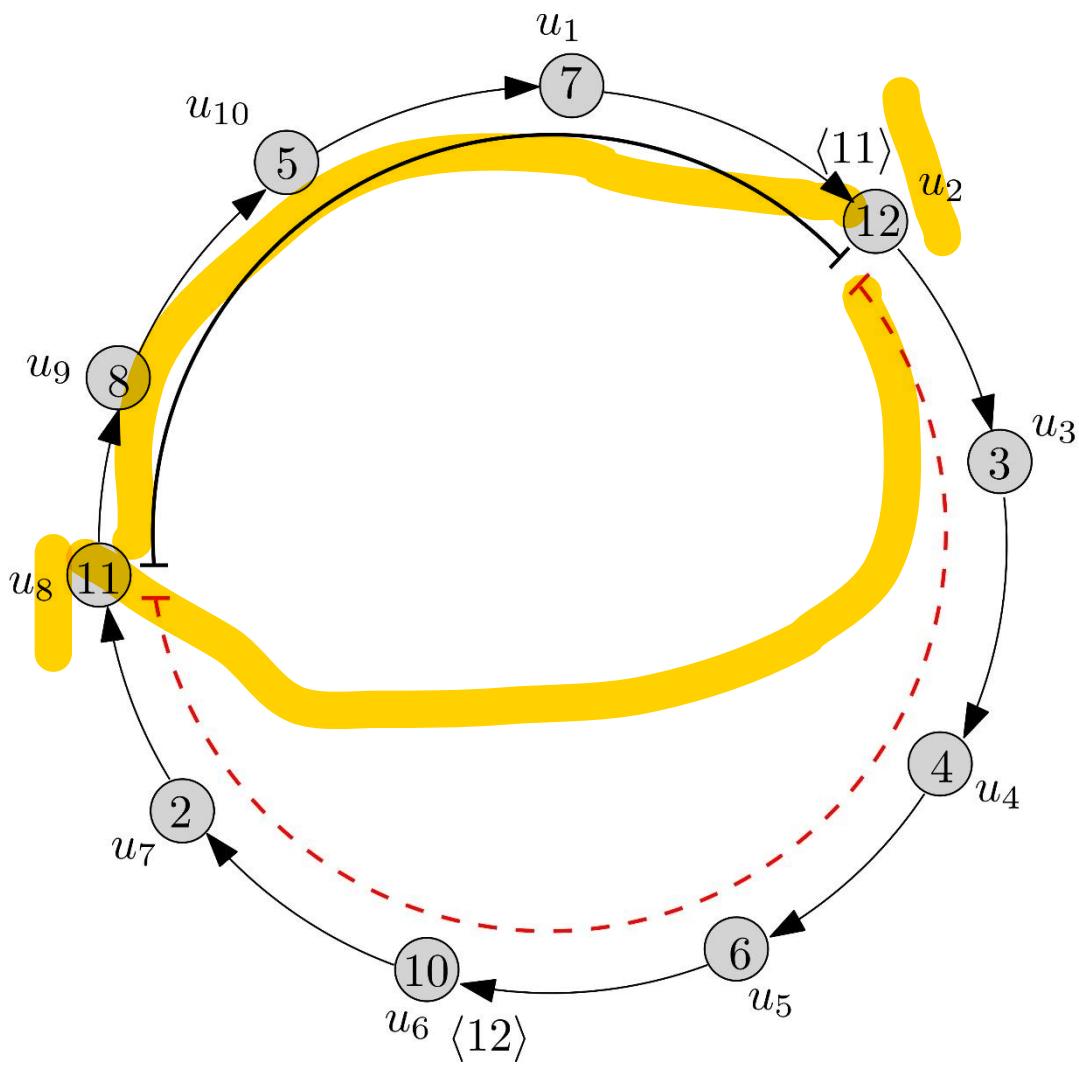
*Proof.*

- To output “*leader*”, the  $id_i$  of  $u_i$  must perform a complete turn and get back to  $u_i$ ;
- But  $u_{max}$  is always in the way and
- $id_i < id_{max}$
- Therefore,  $id_i$  will meet  $u_{max}$  before performing a complete turn and will be discarded by  $u_{max}$

□

- Can be made completely formal by induction on the number of rounds  $r$

# LCR Correctness



- $u_2$  is  $u_{max}$  in this case
- $u_8$ 's id is  $id_8 = 11$
- $u_8$ 's id can travel up to the point that it meets  $u_2$  but not further than this
- It will always be discarded by  $u_2$

# LCR Correctness

**Theorem.** The LCR algorithm solves the leader election problem in any directed ring network (provided the availability of unique ids).

- Works also for **undirected rings**
  - provided that all processors can distinguish their clockwise from their counterclockwise neighbour
- We could easily modify the algorithm to elect the processor with the **minimum id** instead
  - Try this at home

# LCR Time Complexity

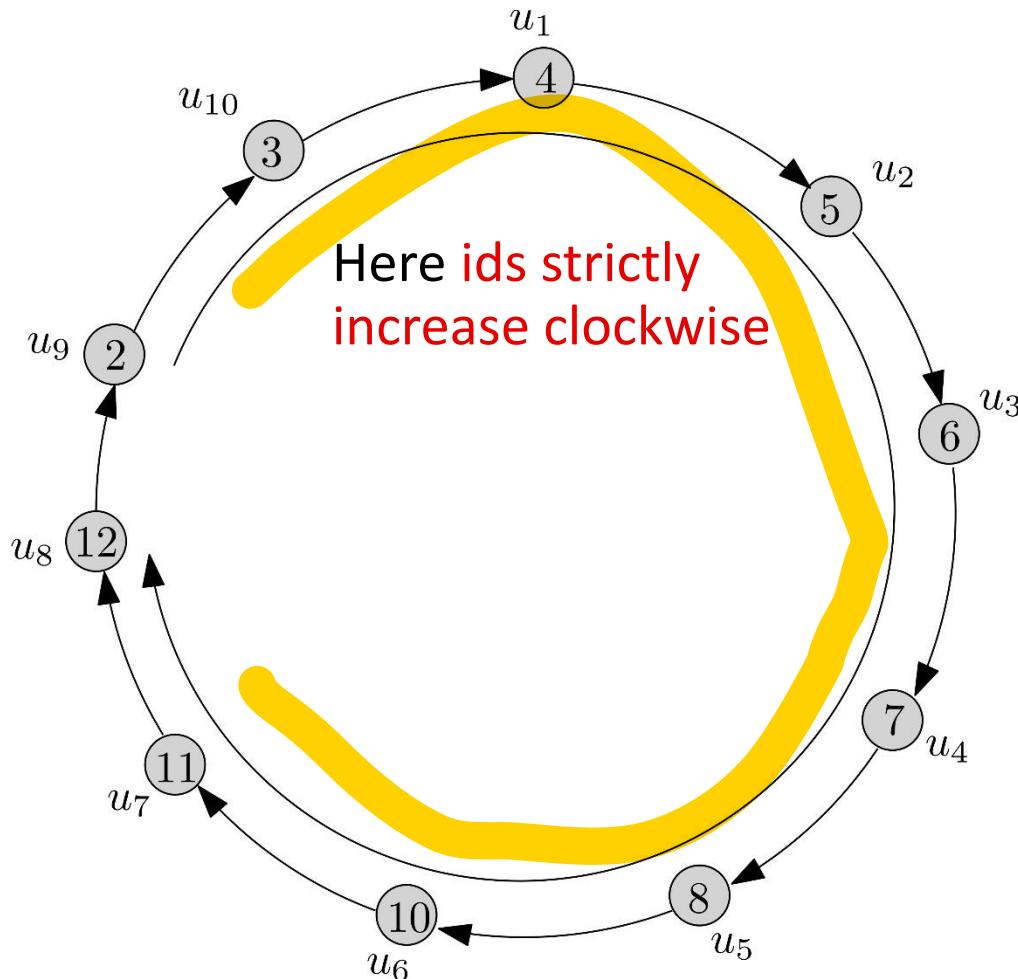
- $n + 1$  rounds
  - *Why?*
- How can we modify the algorithm so that all processors
  - eventually terminate and
  - know who the elected leader is
- ?
- *What will be the new time complexity then?*

# LCR Communication Complexity

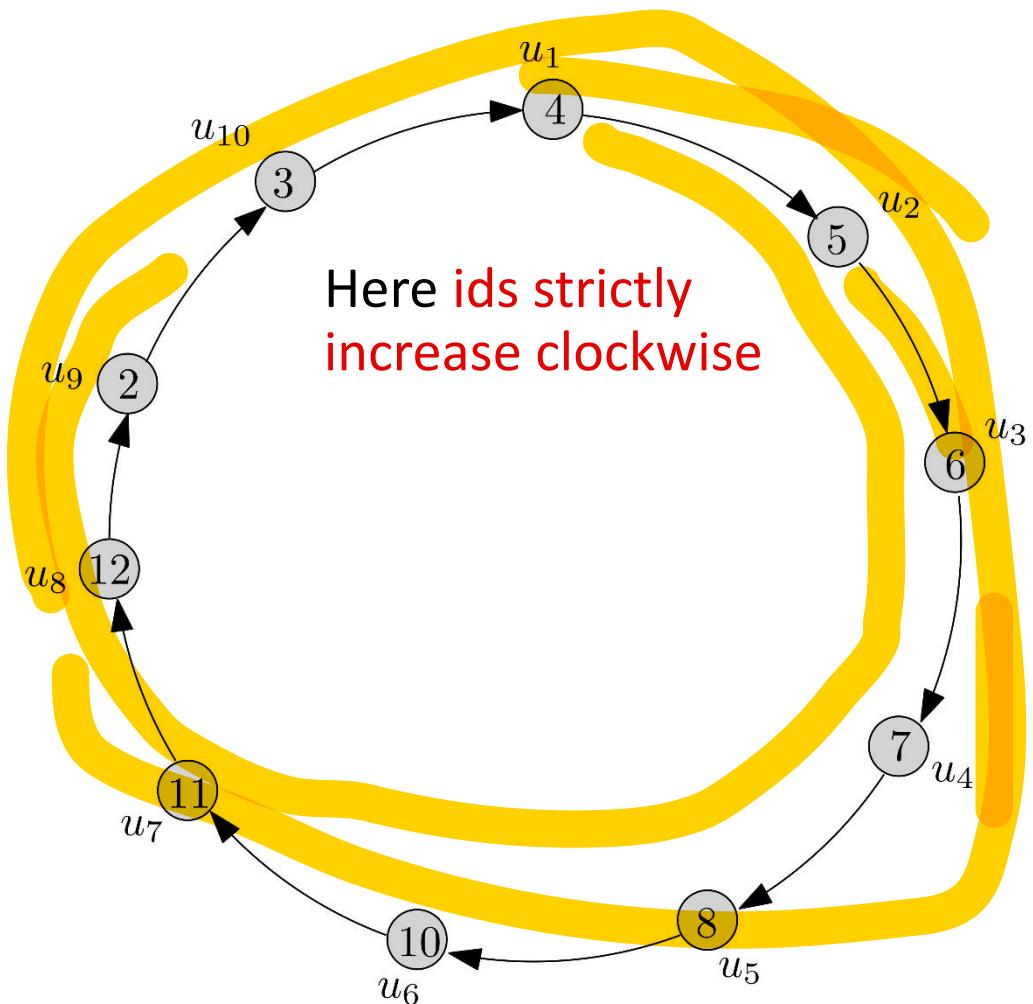
- $O(n^2)$  messages (in total) in the worst case
- The worst case is not the same for every problem and algorithm
  - *We have to think and find out the worst case of each specific algorithm*
- But what parameters it depends on?
  - i.e., what affects the #messages to be transmitted?
    1. Size  $n$  of the ring
    2. How ids are initially assigned

# LCR Communication Complexity

- *Is this the worst case?*



# LCR Communication Complexity



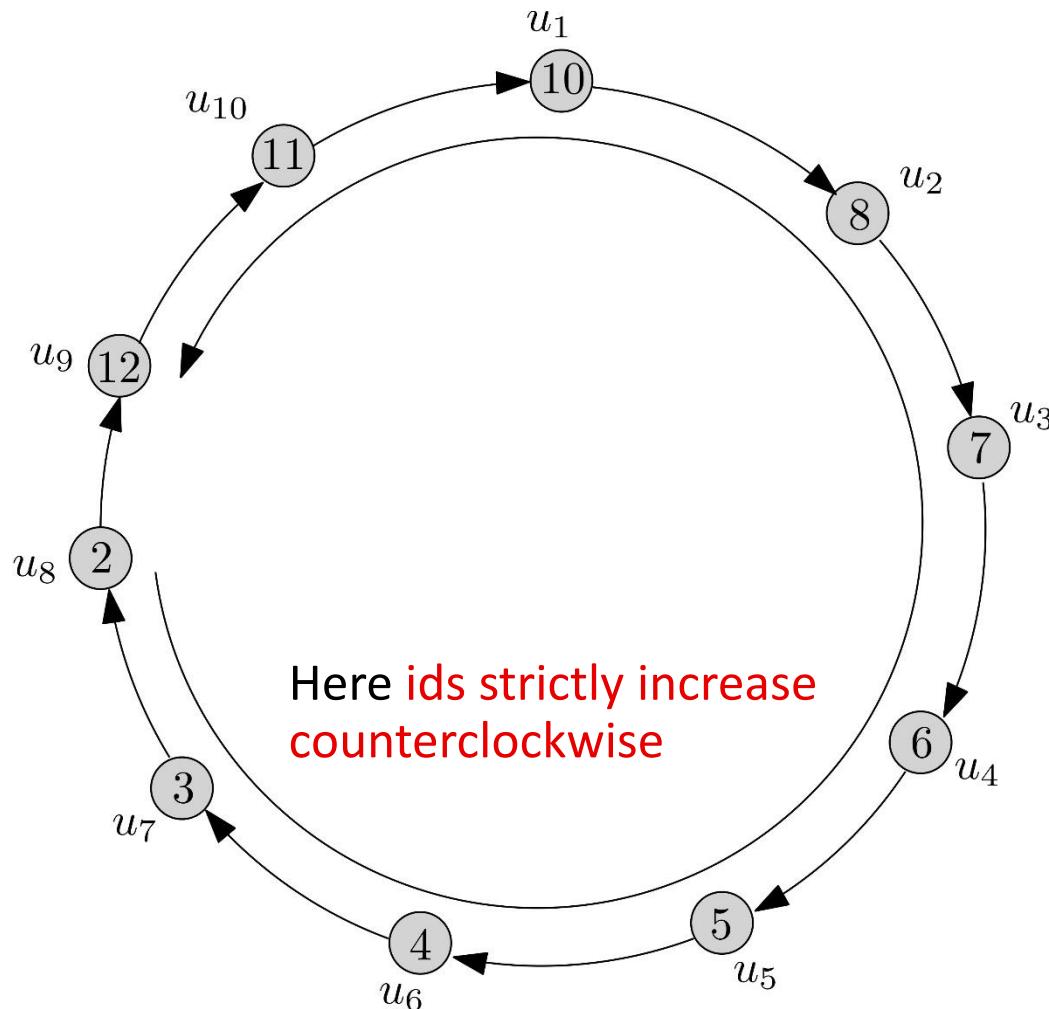
- Cannot tell yet if this is the worst case
- But we can analyse it
- Every  $id_i < id_{max}$  has a greater id to its right initially
  - So, transmitted once and then discarded
  - So,  $n - 1$  ids are only transmitted once
- $id_{max}$  is always transmitted  $n$  times (complete turn)

In total:

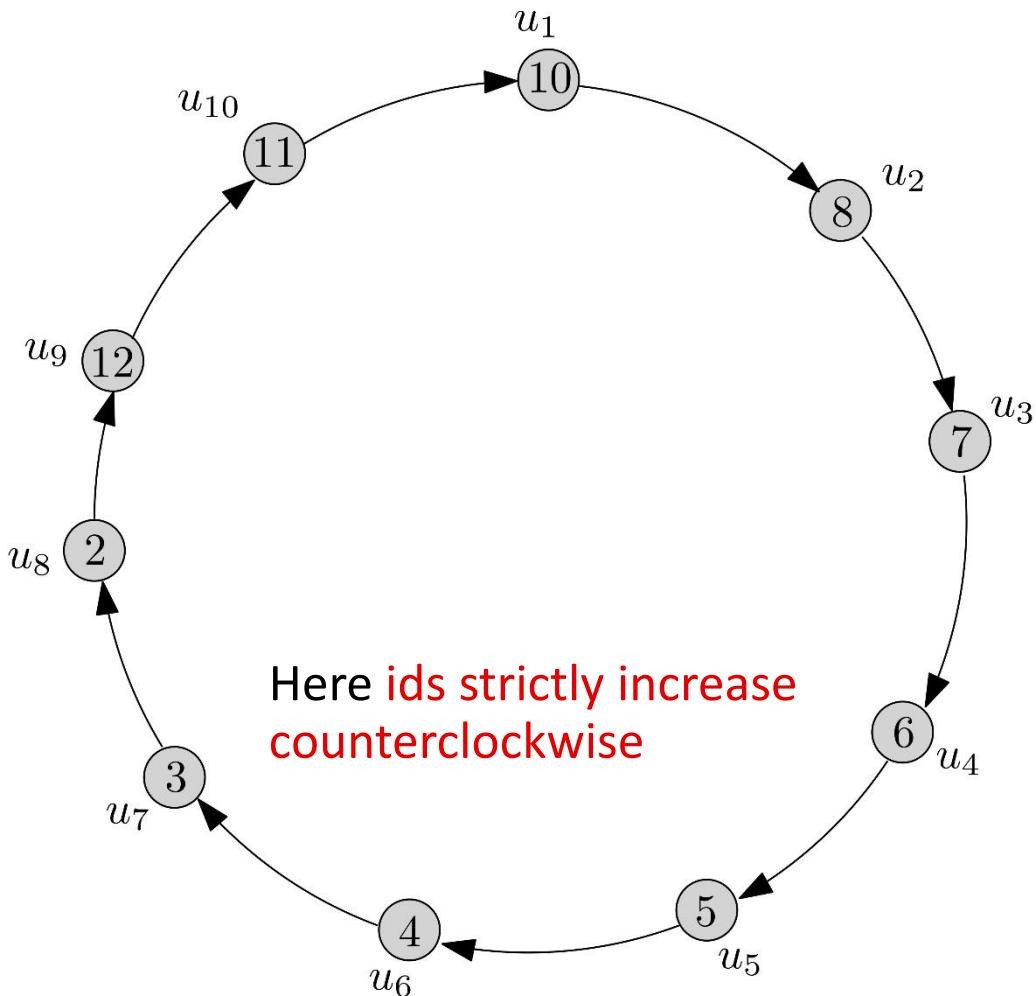
$$\text{\#messages} = n - 1 + n = 2n - 1$$

# LCR Communication Complexity

- *Is this the worst case?*



# LCR Communication Complexity



- Yes it is!
- Let's analyse it
- $id_{max}$  is always transmitted  **$n$  times** (complete turn)
- Every  $id_i < id_{max}$  will manage to reach  $u_{max}$ 
  - So, one id will travel  $n - 1$  edges, the next  $n - 2$ , ... the next 2, the next 1
  - This gives:

$$1 + 2 + \dots + (n - 2) + (n - 1)$$

In total:

$$\begin{aligned}\text{\#messages} &= 1 + 2 + \dots + n \\ &= n(n + 1)/2 \\ &= \Theta(n^2)\end{aligned}$$

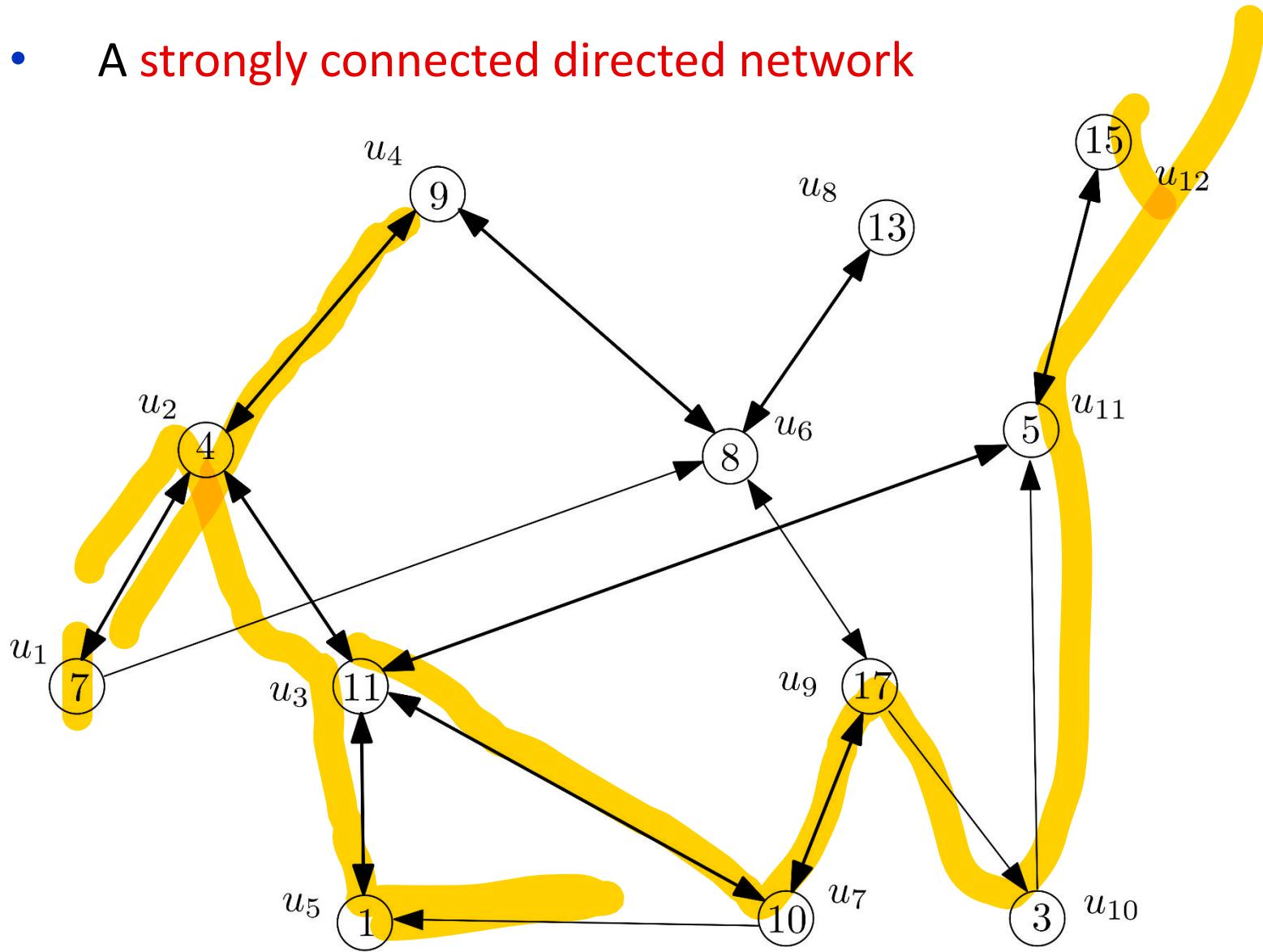
# Beyond Rings Leader Election in General Networks

# Leader Election in General Networks

- Elect a *unique leader processor* from among all the processors in the distributed system
- Now the network can be any **strongly connected directed network**
  - **Strongly connected:** For every processors  $u, v$  there is a path from  $u$  to  $v$  and a path from  $v$  to  $u$
  - e.g., the **directed ring** is just a special case
  - *Why can't we use LCR in this case?*
- Processors have **unique ids**

# Leader Election in General Networks

- A strongly connected directed network



# A Simple Algorithm based on Flooding

- Processors have unique ids, do not know  $n$  in advance, but do know the diameter  $D$  of the network
  - Diameter:
    - the distance between two nodes is given by the shortest path between them
    - Then the diameter of the network is determined by the pair of nodes at maximum distance (and is equal to that distance)
    - In other words, it is the maximum shortest path in the network
- FloodMax algorithm: solves the problem
- Uses transmission, comparison, and storage of ids
- Main idea: Flood the maximum id
  - LCR also does something like this but does not require knowledge of  $D$  and its termination condition works only for rings

# FloodMax: Informal description

- All processors know the diameter  $D$  and their own id in advance
- All processors remember the greatest id that they have “heard” so far (initially their own)
- In every round all processors send the greatest known to all their out-neighbours
- After  $D$  rounds compare the largest heard to your own
  - if greatest heard = own id, declare yourself the leader
  - otherwise, declare yourself non-leader
- Intuitively:
  - The maximum id will manage to reach the whole network
  - So everyone non-maximum will know that there is a greater id and  $u_{max}$  can never receive a larger id

# FloodMax: Pseudocode

## Algorithm FloodMax

State of processor  $u_i$ :

- $myID_i$ : holds the processor's unique id
- $maxID_i$ : holds the greatest id “heard” so far
- $status_i \in \{“unknown”, “leader”, “non-leader”\}$ : indicates whether  $u_i$  has been elected (“leader”), not elected (“non-leader”) or doesn’ know yet (“unknown”)

# FloodMax: Pseudocode

## Algorithm FloodMax

Code for processor  $u_i$ ,  $i \in \{1, 2, \dots, n\}$ :

Initially:

$u_i$  knows its own unique id stored in  $myID_i$

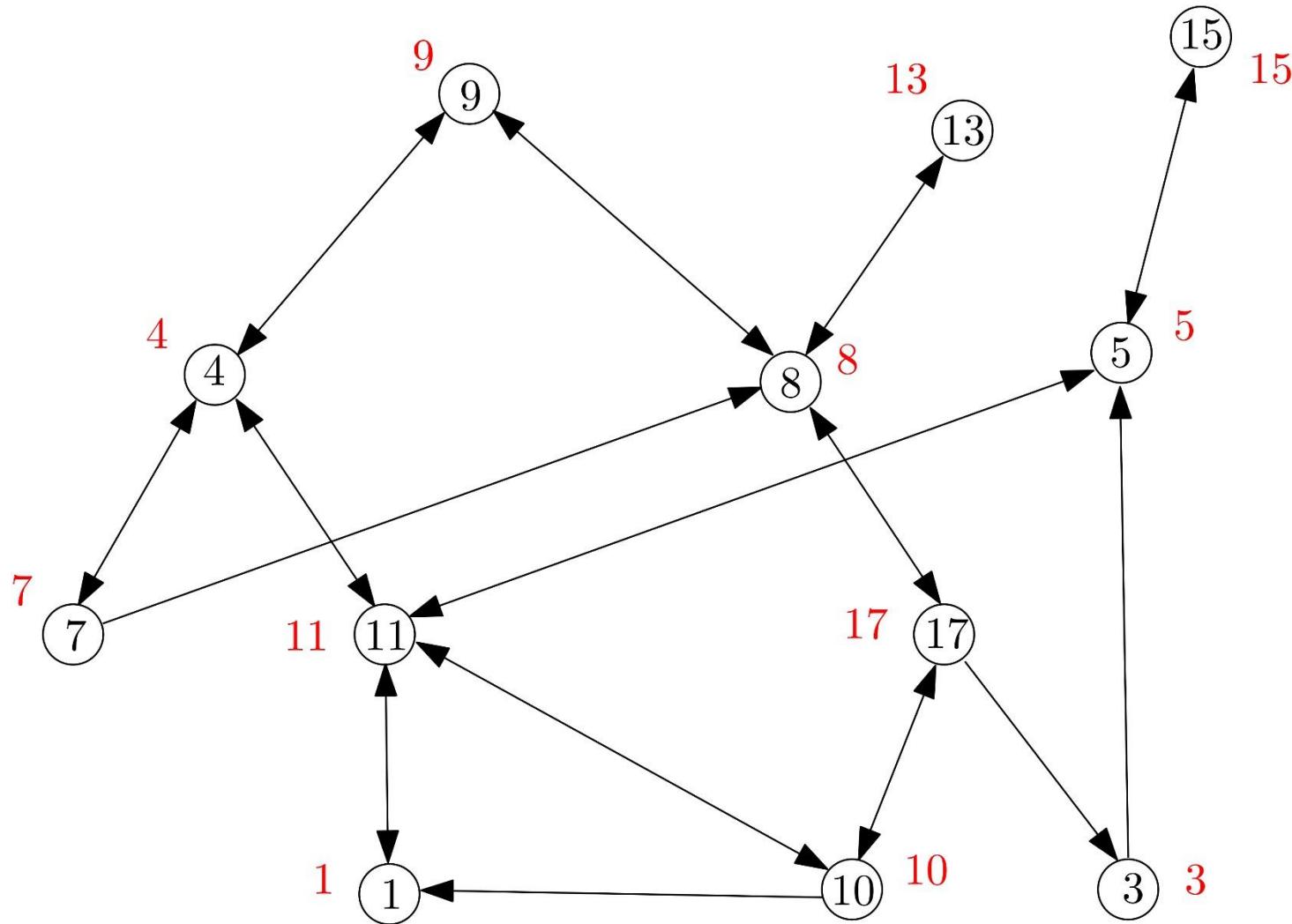
$maxID_i := myID_i$

$status_i := "unknown"$

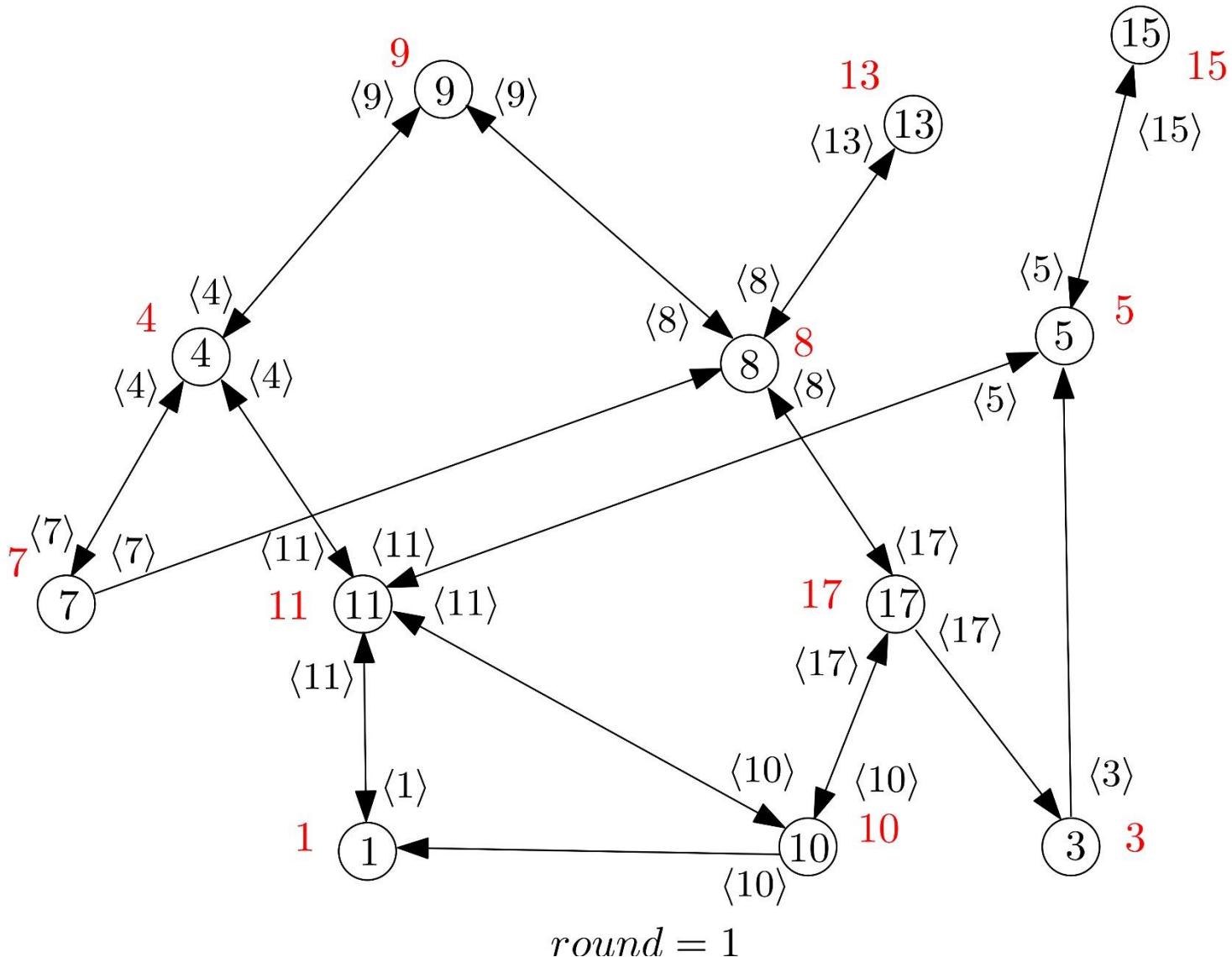
Also has access to the current round and knows the diameter  $D$

```
if round = 1 then
    send ⟨maxIDi⟩ to all out-neighbours
else
    upon receiving ⟨inIDs⟩ from in-neighbours           // one or more ids arriving from neighbours
        maxIDi := max({maxIDi} ∪ inIDs)           // remember only the maximum “heard” so far
        if round ≤ D then // 1 < round ≤ D
            send ⟨maxIDi⟩ to all out-neighbours
        else // round = D + 1
            if maxIDi = myIDi then          // if equal to your own, no greater id exists in the network
                statusi := "leader"           // therefore, elect yourself a leader
            else                           // greater than own
                statusi := "non-leader"      // therefore, declare yourself a non-leader
// observe that in the end all processors know the id of the elected leader, stored in their maxID,
// variable
```

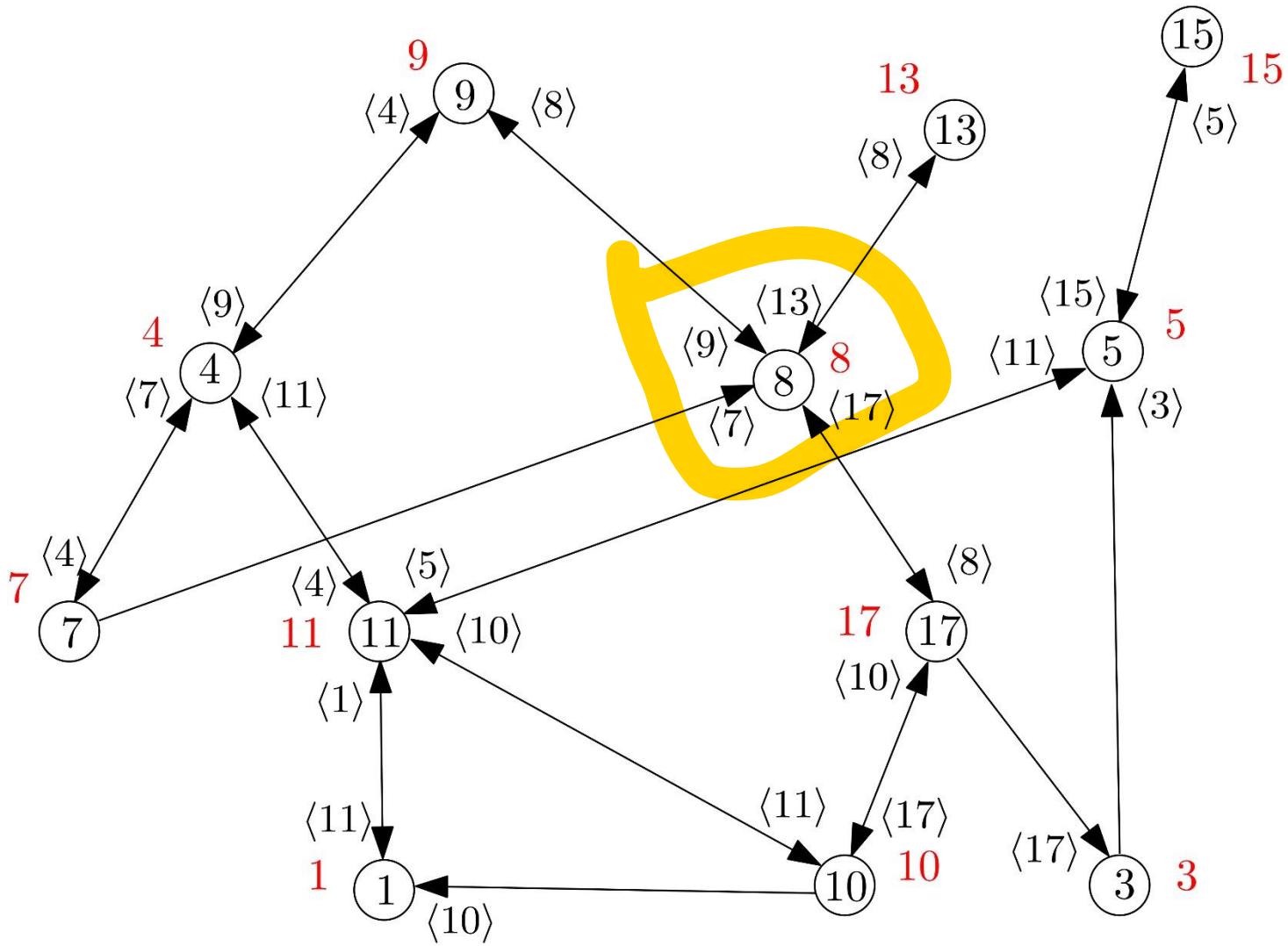
# Example Execution



# Example Execution

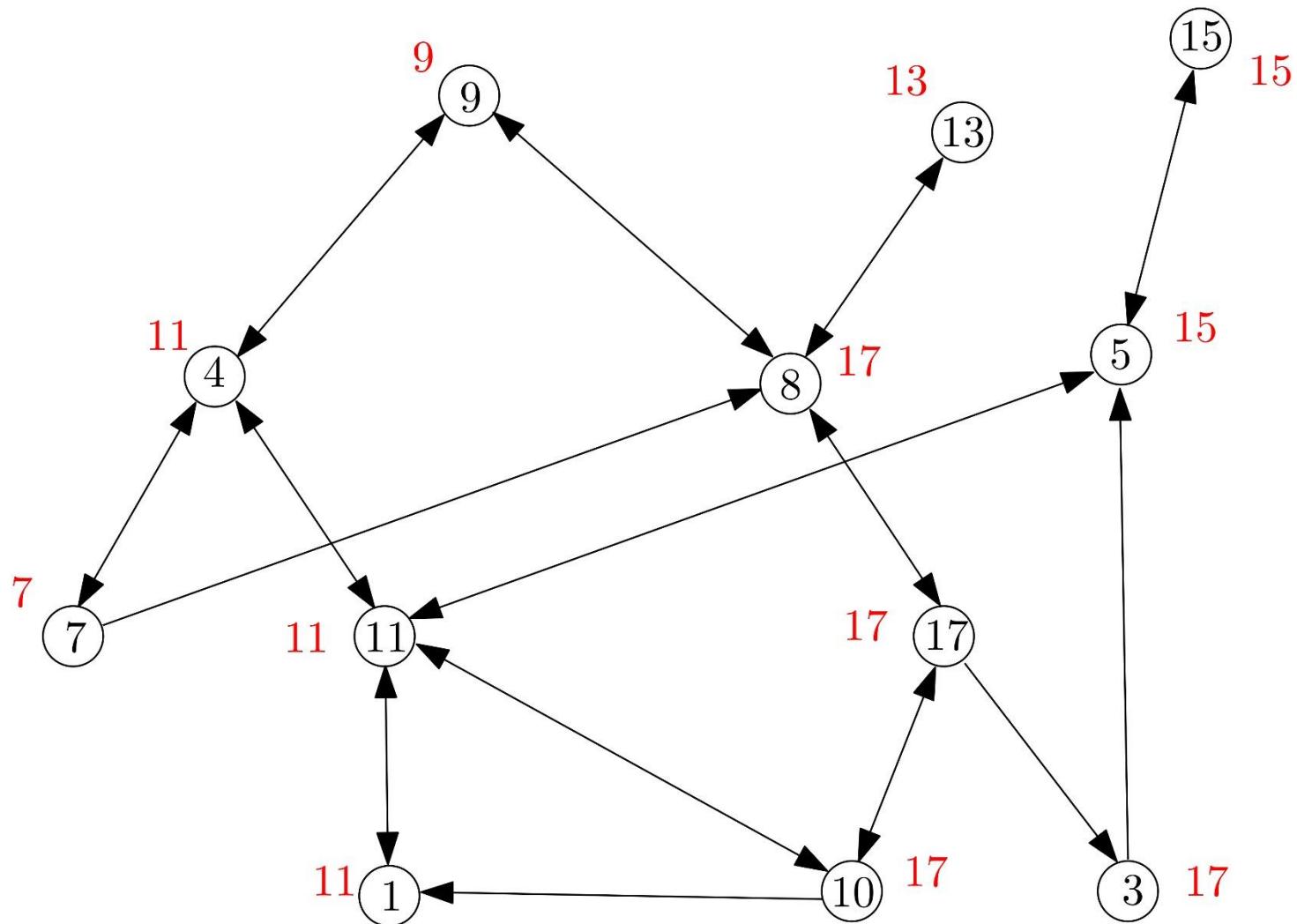


# Example Execution



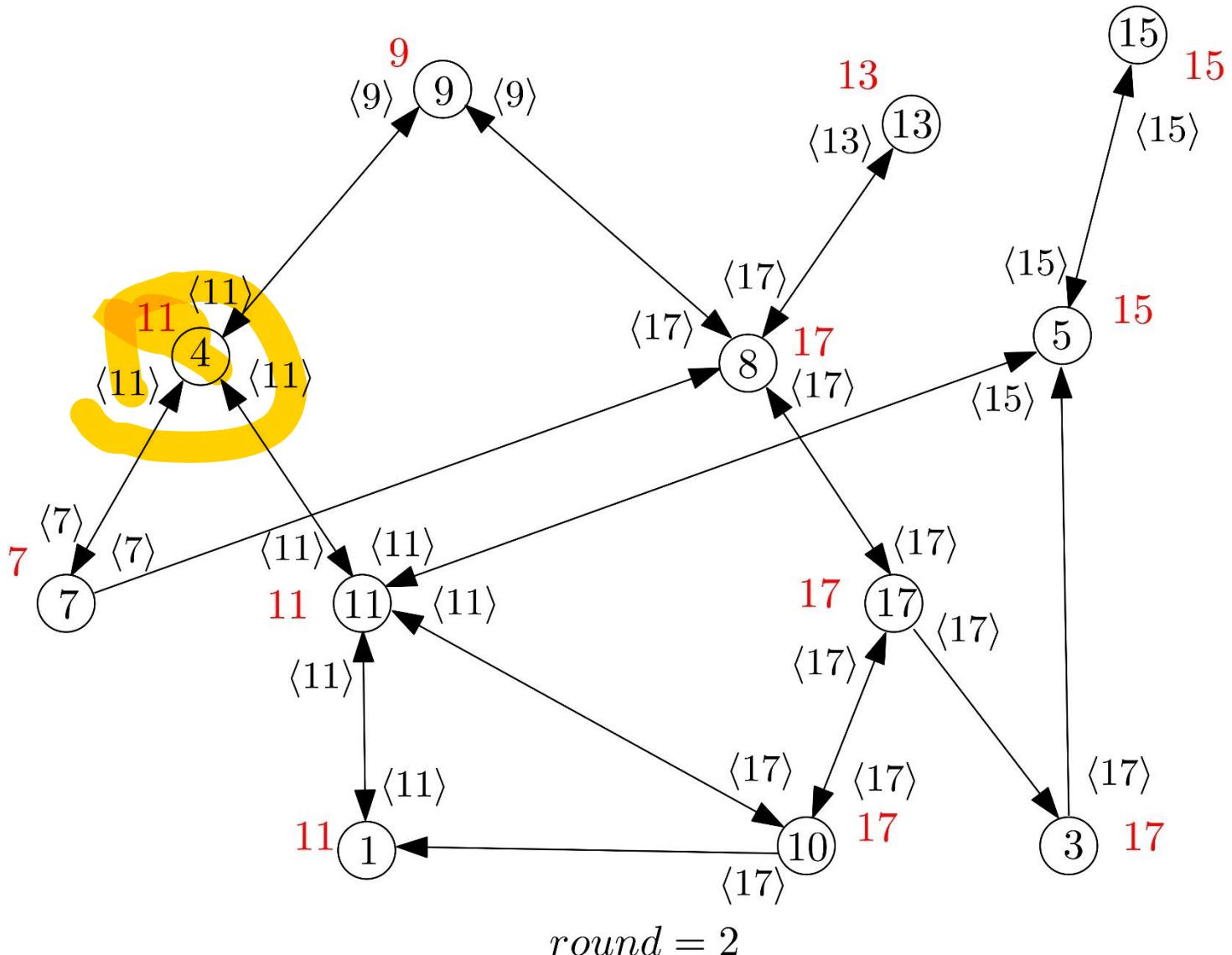
round = 1

# Example Execution

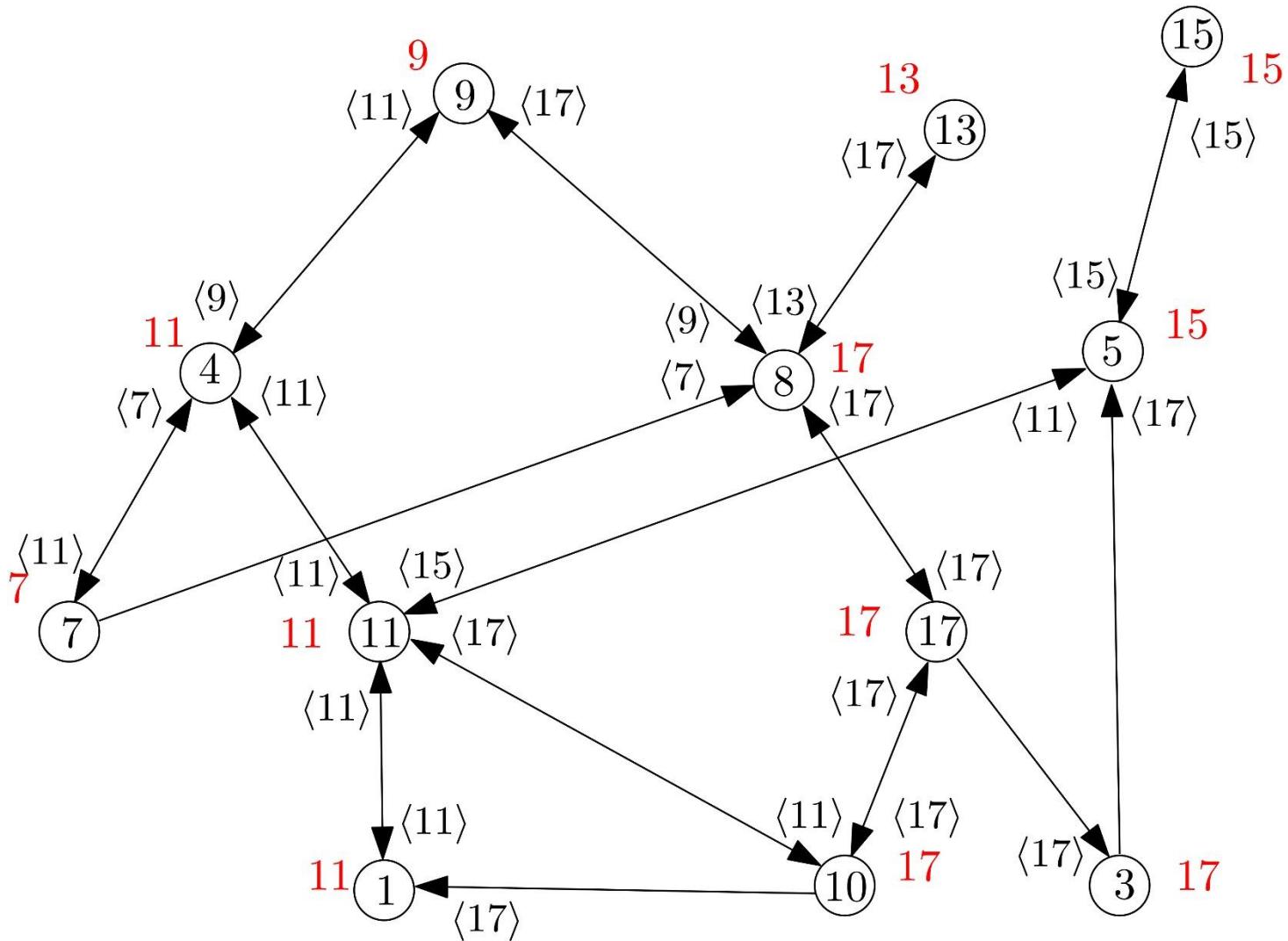


*round = 2*

# Example Execution

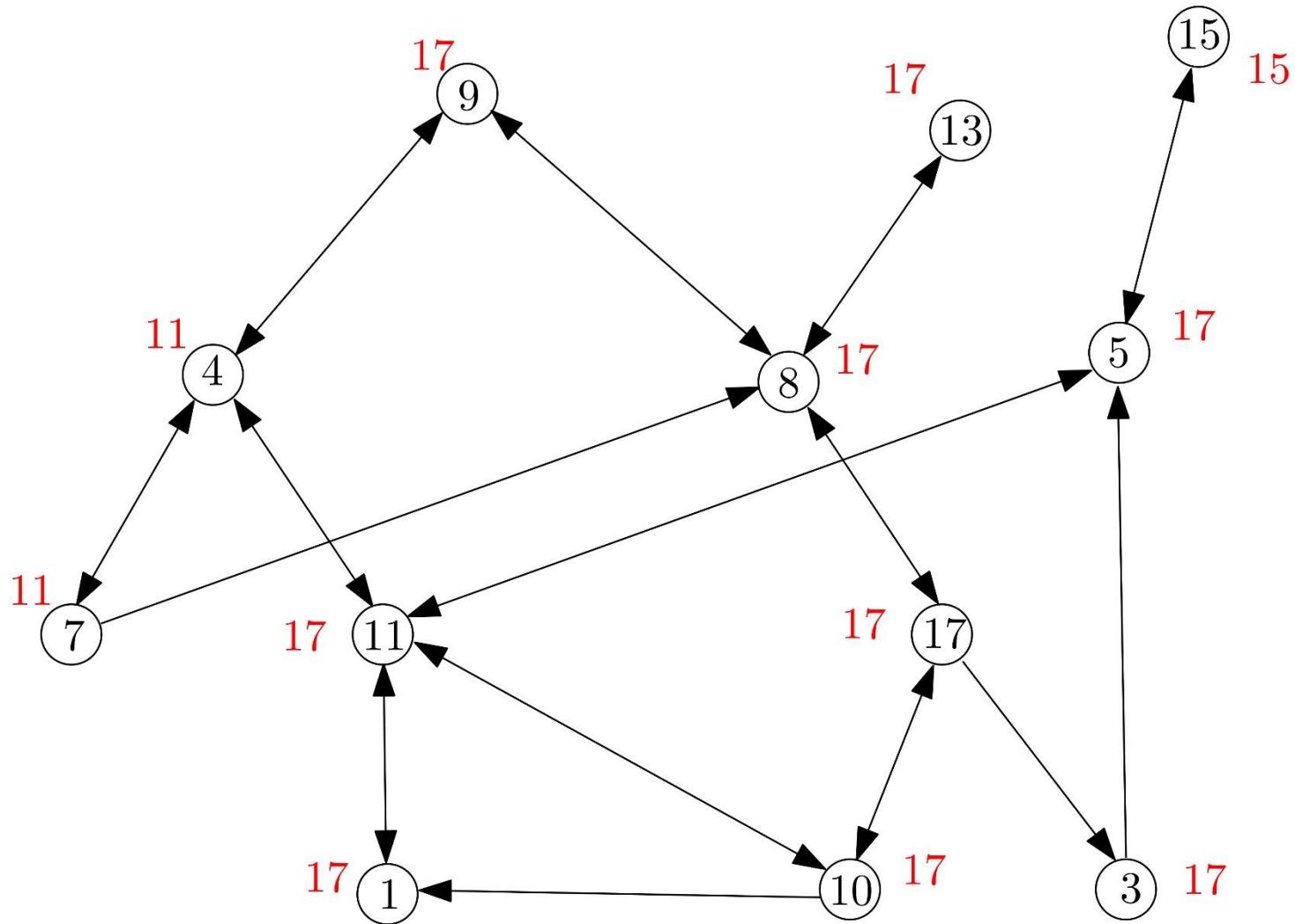


# Example Execution



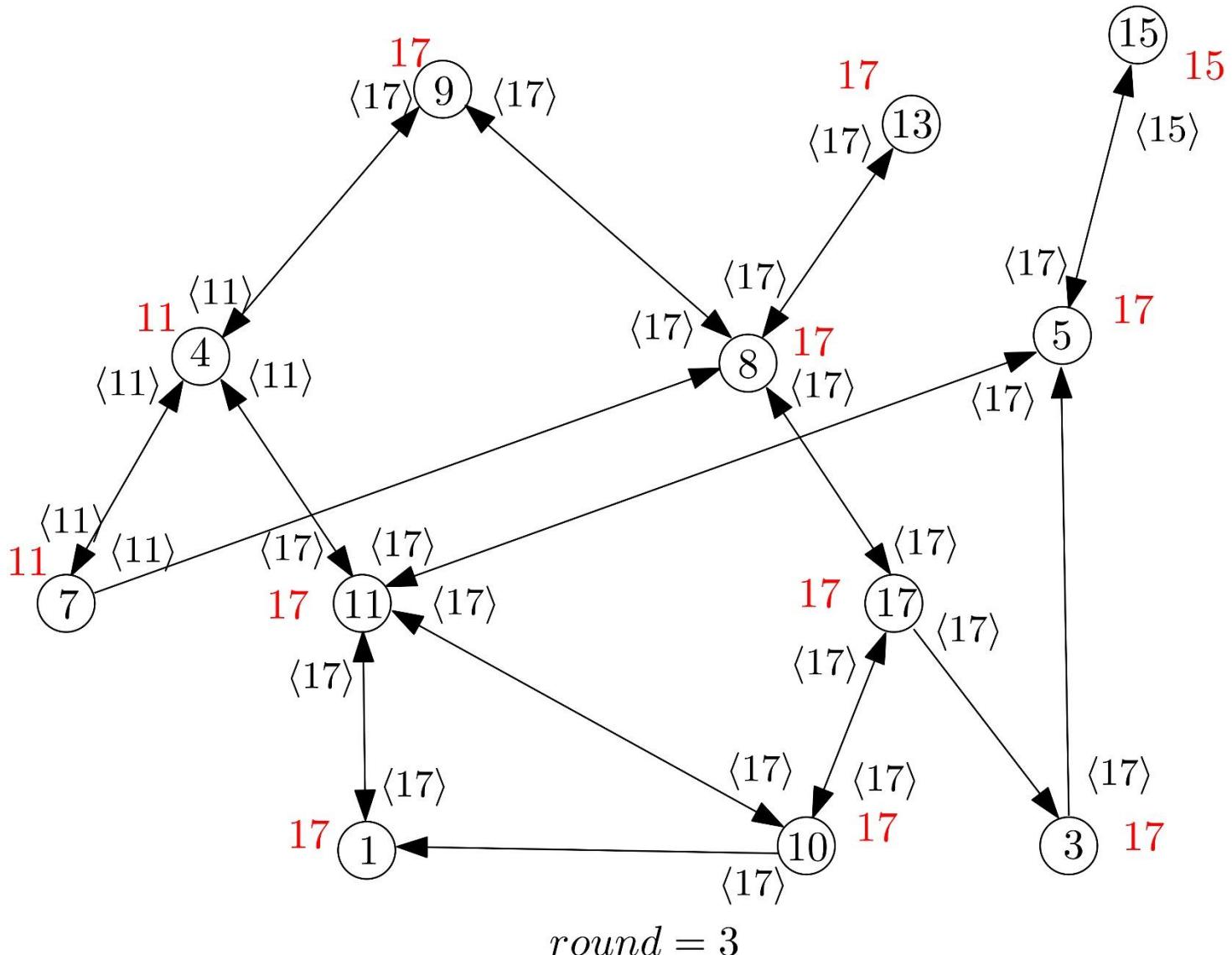
round = 2

# Example Execution

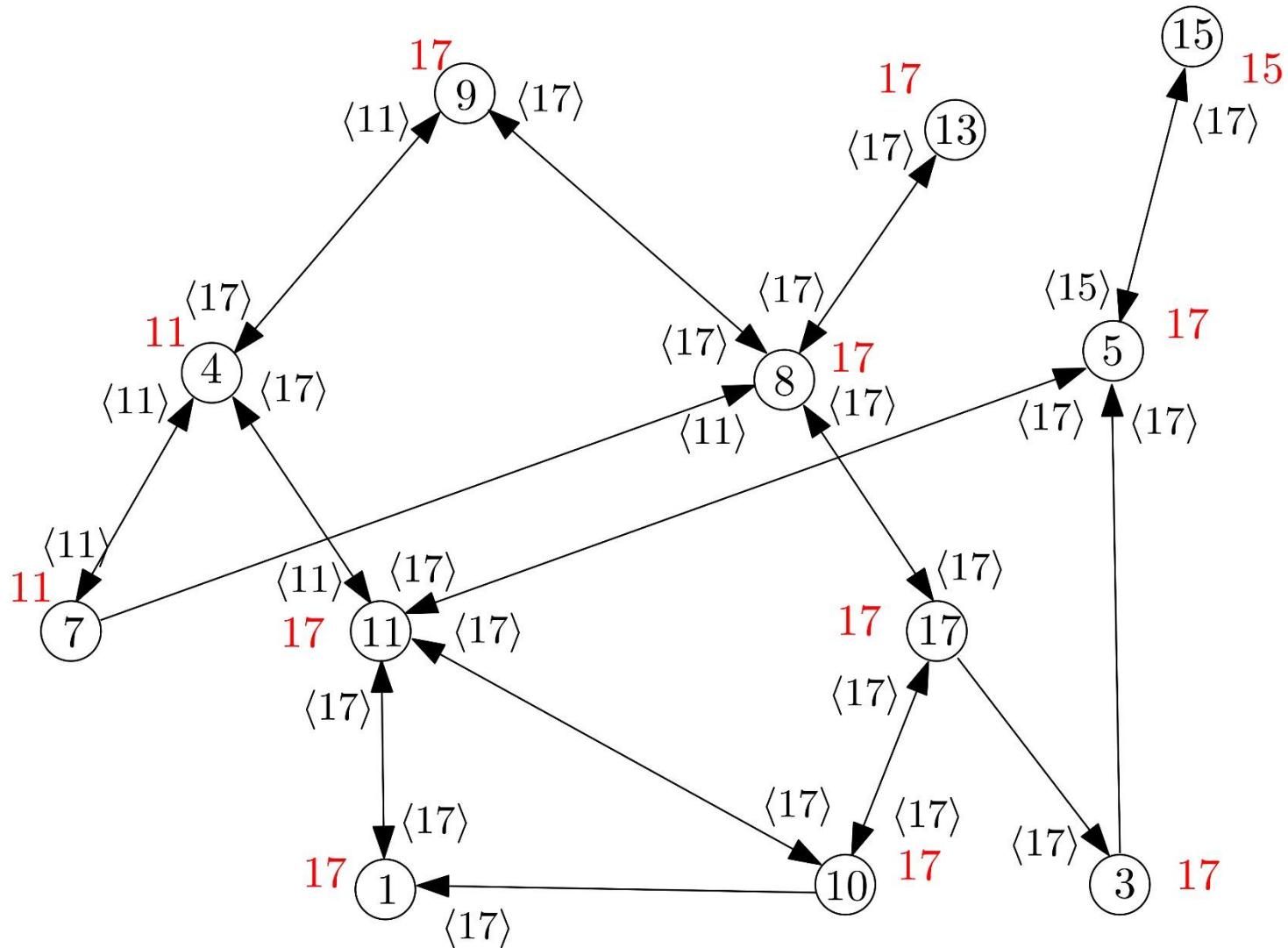


round = 3

# Example Execution

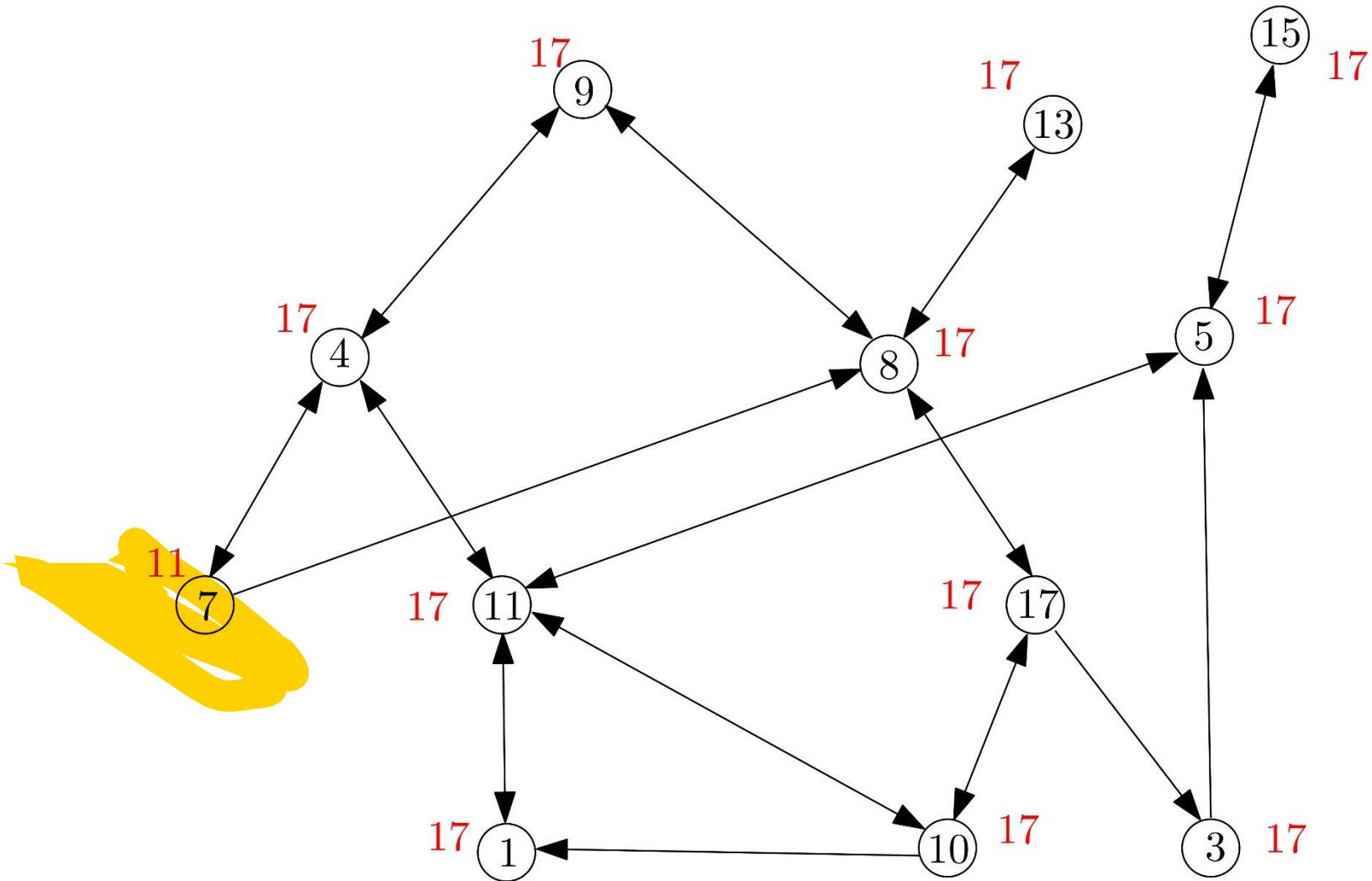


# Example Execution



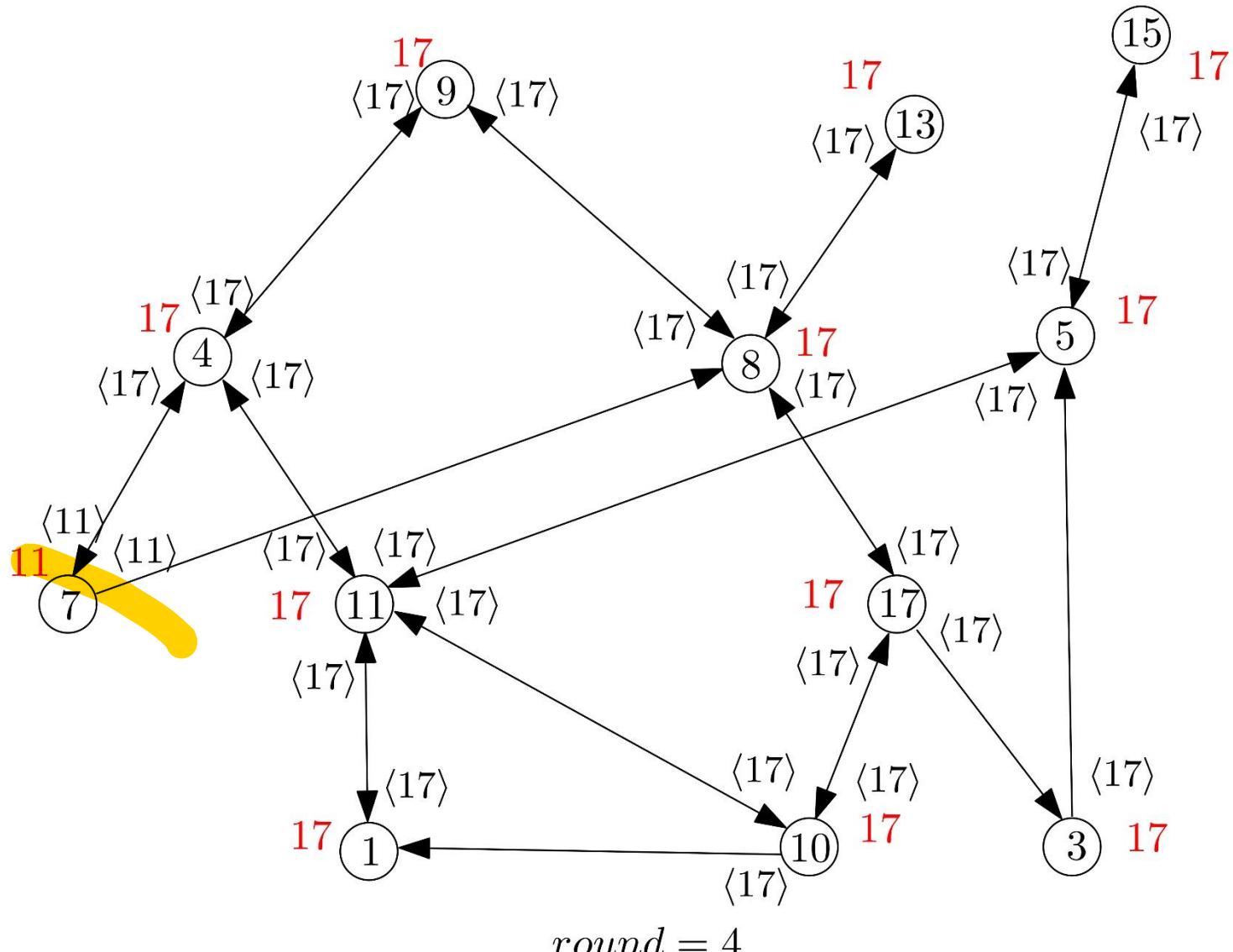
round = 3

# Example Execution

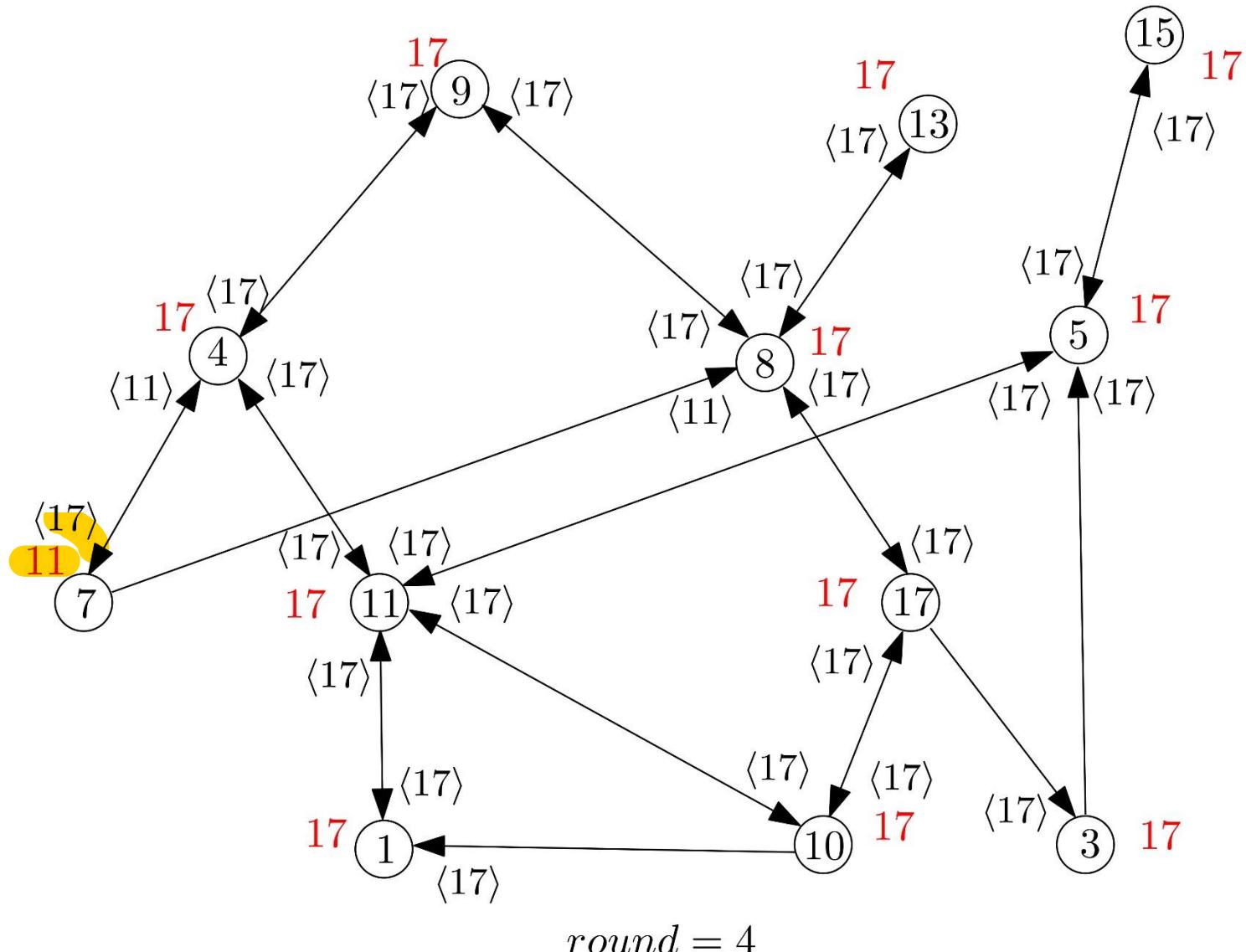


*round = 4*

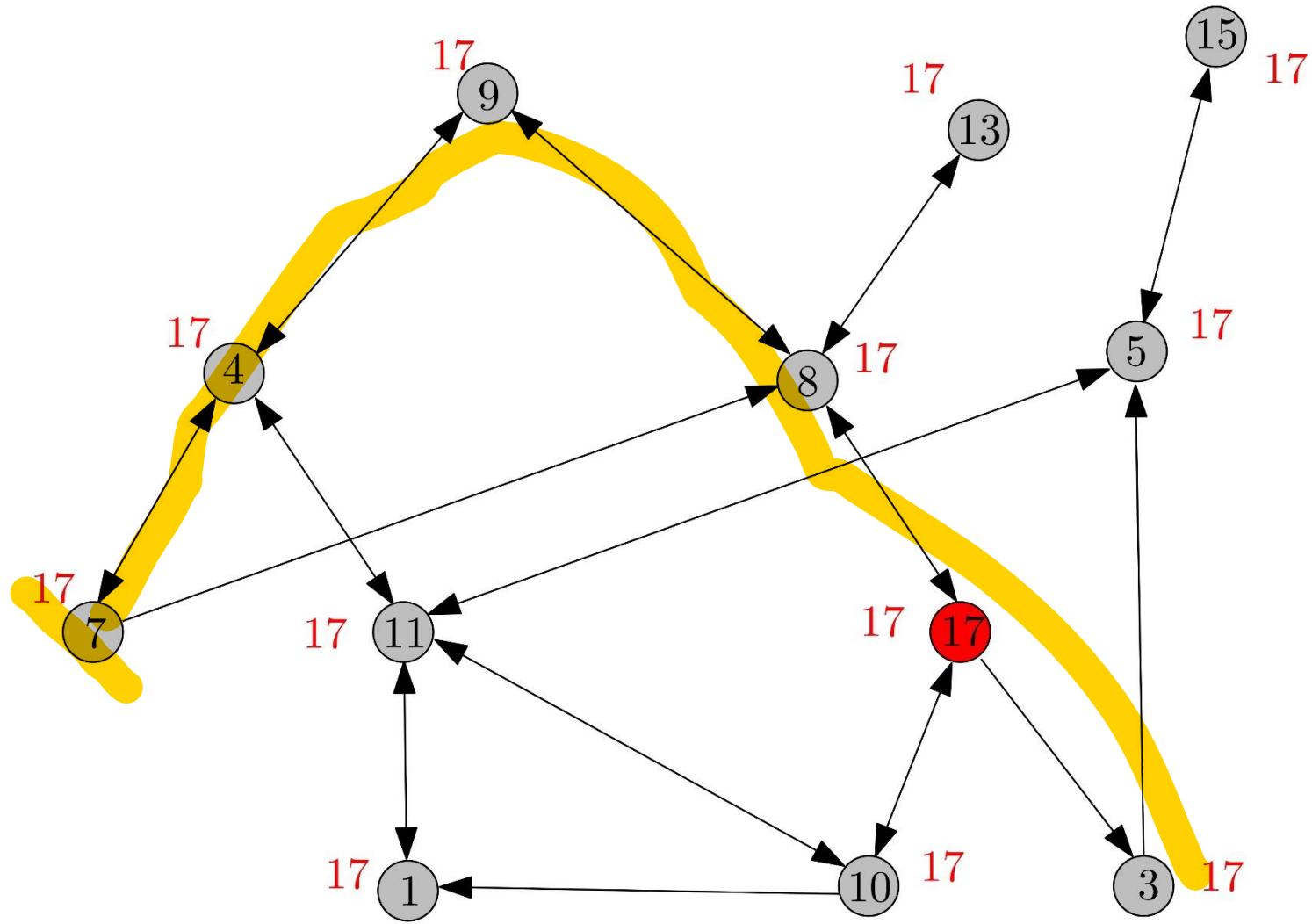
# Example Execution



# Example Execution



# Example Execution



round = 5

# Correctness and Complexity

- Correctness:
  - exactly one processor is elected in the last round
- Time complexity:
  - $D + 1$  rounds (or  $D$  depending on the round model)
- Communication complexity:
  - size of messages: encoding in bits of the maximum id
  - $D \cdot m$  messages always
    - $m$  is the number of directed links in the network
- *Can you see why?*

Think of these and we shall prove them in class

# Summary

- Leader election is crucial for distributed systems
  - breaks symmetry
  - allows for coordination
- If all processors are initially identical then
  - impossible to elect a leader even in very simple networks
  - e.g., a ring
- Adding unique ids breaks this inconvenient initial symmetry
- The LCR algorithm elects a leader in any ring network
  - simple conceptually, assumes unique ids
  - $n$  rounds (or  $2n$  for all to terminate),  $O(n^2)$  messages
- The FloodMax algorithm elects a leader in any strongly connected network
  - like a generalisation of LCR
  - simple, assumes unique ids and knowledge of the diameter  $D$
  - $D$  rounds,  $D \cdot m$  messages