Distributed Systems
COMP 212

Lecture 6

Othon Michail

UNIVERSITY OF LIVERPOOL

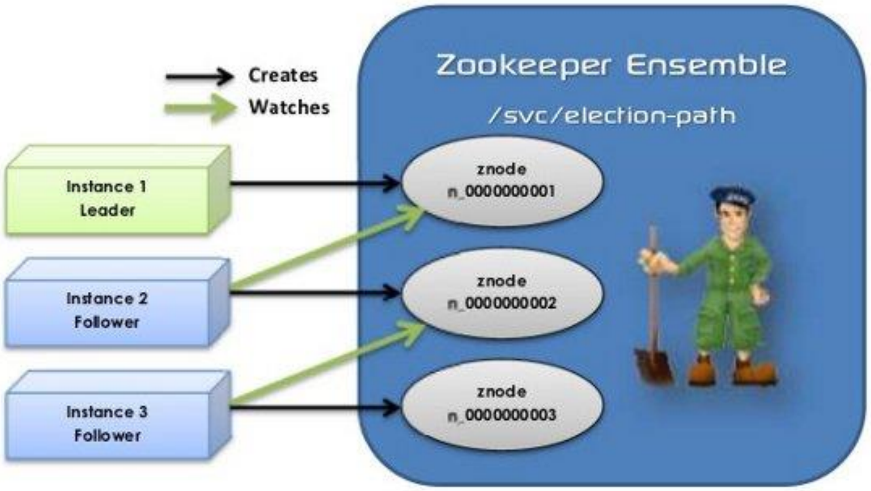
Leader Election

Election for Coordination

- Many Distributed Systems require a single process to act as the coordinator (for various reasons)
  - Time server
  - Coordinator in commit protocols
  - Master process in distributed computations
  - Master database server
- Coordinator may fail → the distributed group of processes must execute an election algorithm to determine a new coordinator process

Applications

- Zookeeper
  - distributed, open-source coordination service for distributed applications
  - provides a centralized infrastructure and services that enable synchronization across an Apache Hadoop cluster



  - uses a leader server
  - elects a new one if needed
  - Hadoop: processing of datasets of big data using the MapReduce programming model
- Chubby lock service
  - Google File System and MapReduce use it to elect a master

Problem Statement

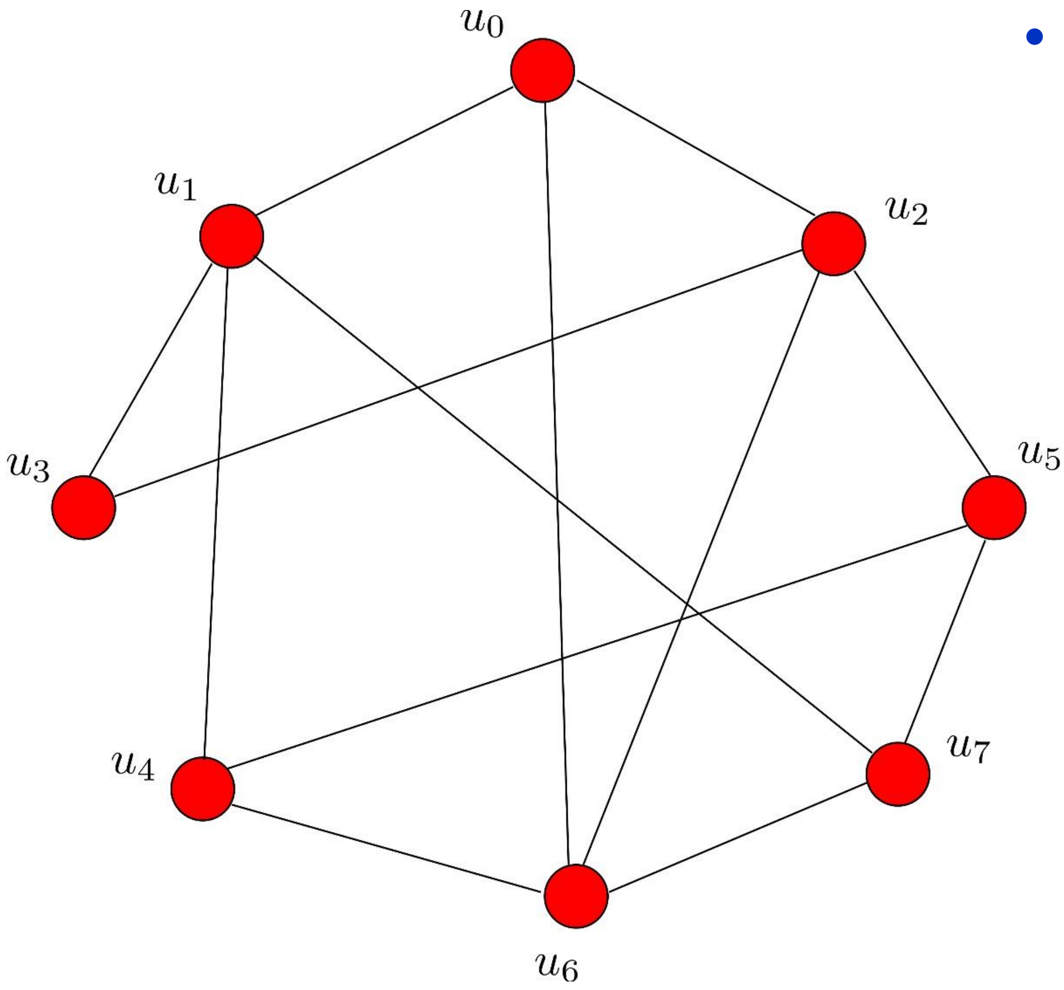
- *Elect a unique leader processor from among all the processors in the distributed system*
- Leader to be interpreted as:
  - coordinator
  - master processor
- Special case of consensus/agreement
- Processors should agree eventually on who they elect

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

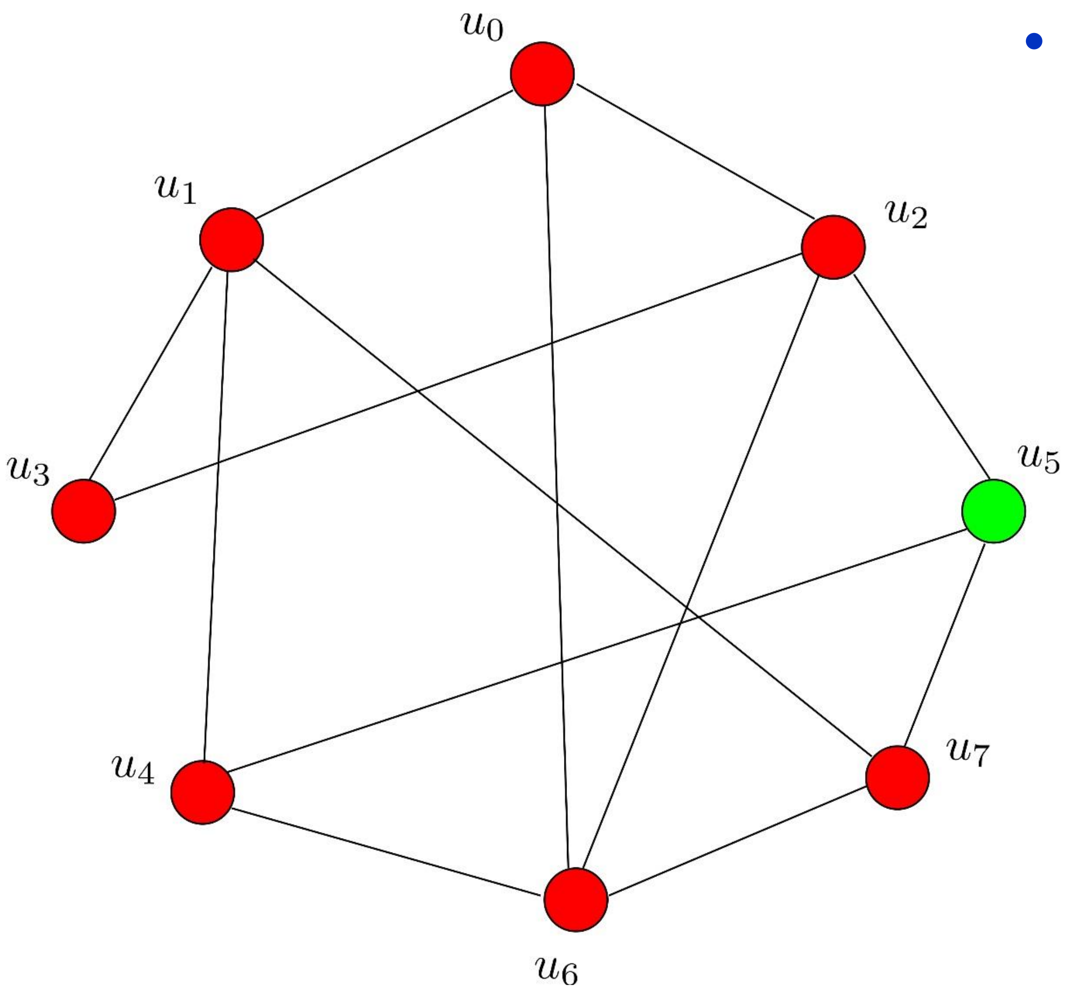
Important remark: Even slightly different assumptions can completely change the algorithmic solutions required or even make a problem impossible to solve

An Illustration



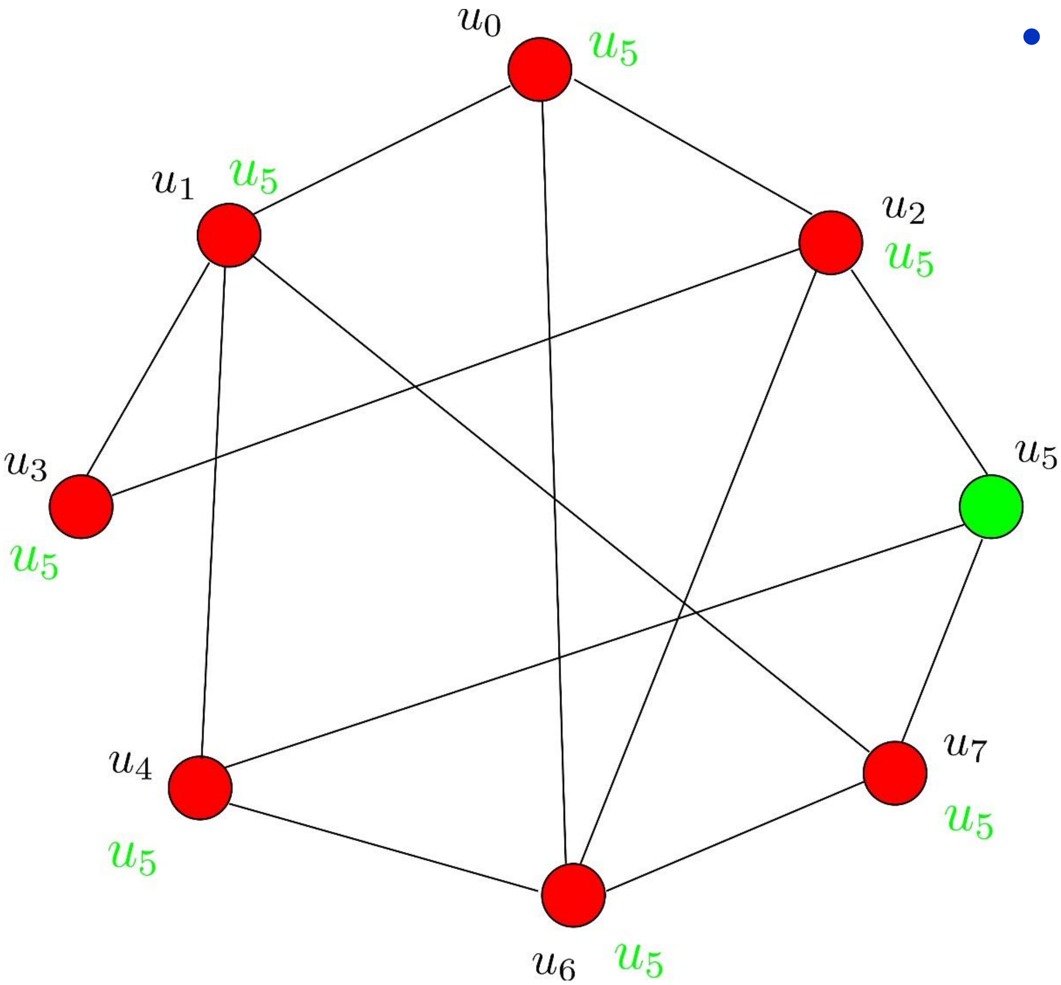
- All processors are initially non-leaders
  - no processor elected yet

An Illustration



- Eventually a unique leader will be elected
  - node u_5 in this case

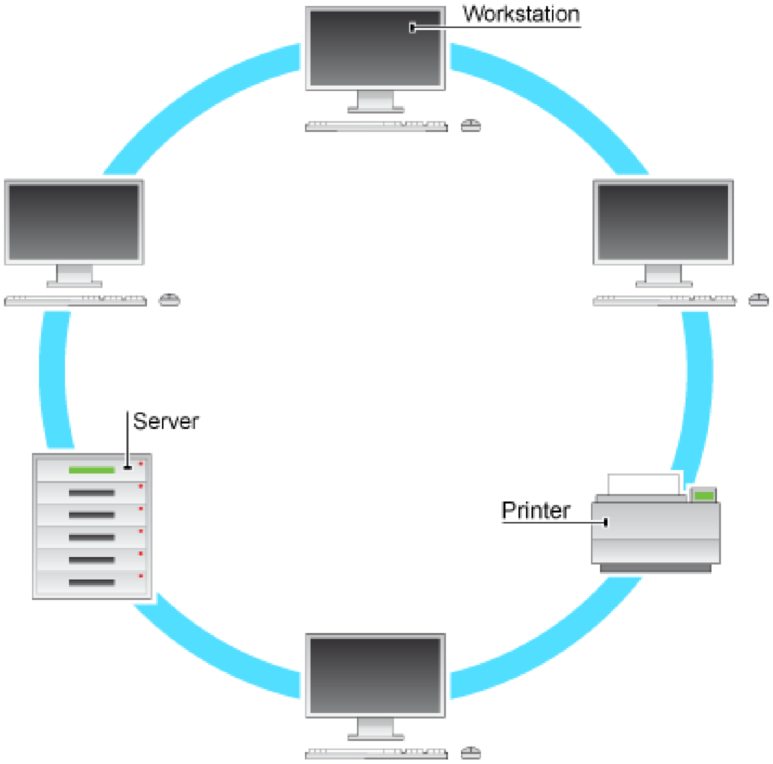
An Illustration



- Possibly all nodes could eventually learn who is the elected leader

Leader Election in a Ring

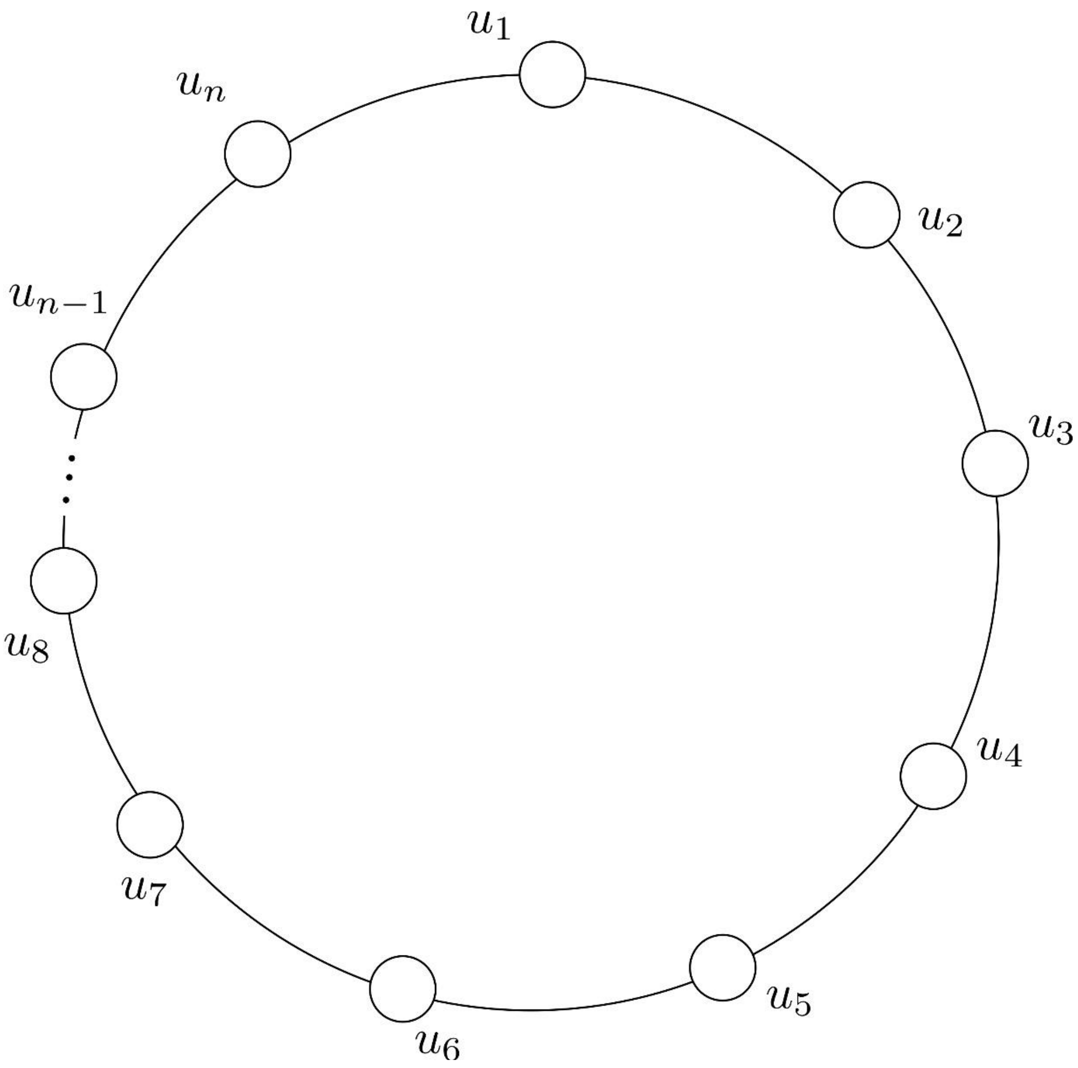
- First appeared in local area token ring networks
- Single token circulates the network
- The current owner of the token has sole right to initiate computation
  - e.g., send jobs to a printer in the network



- If 2 or more nodes were to attempt simultaneously to communicate they would interfere
- But occasionally the token is lost
- Processors must execute an algorithm to regenerate the lost token
  - Equivalent to leader election

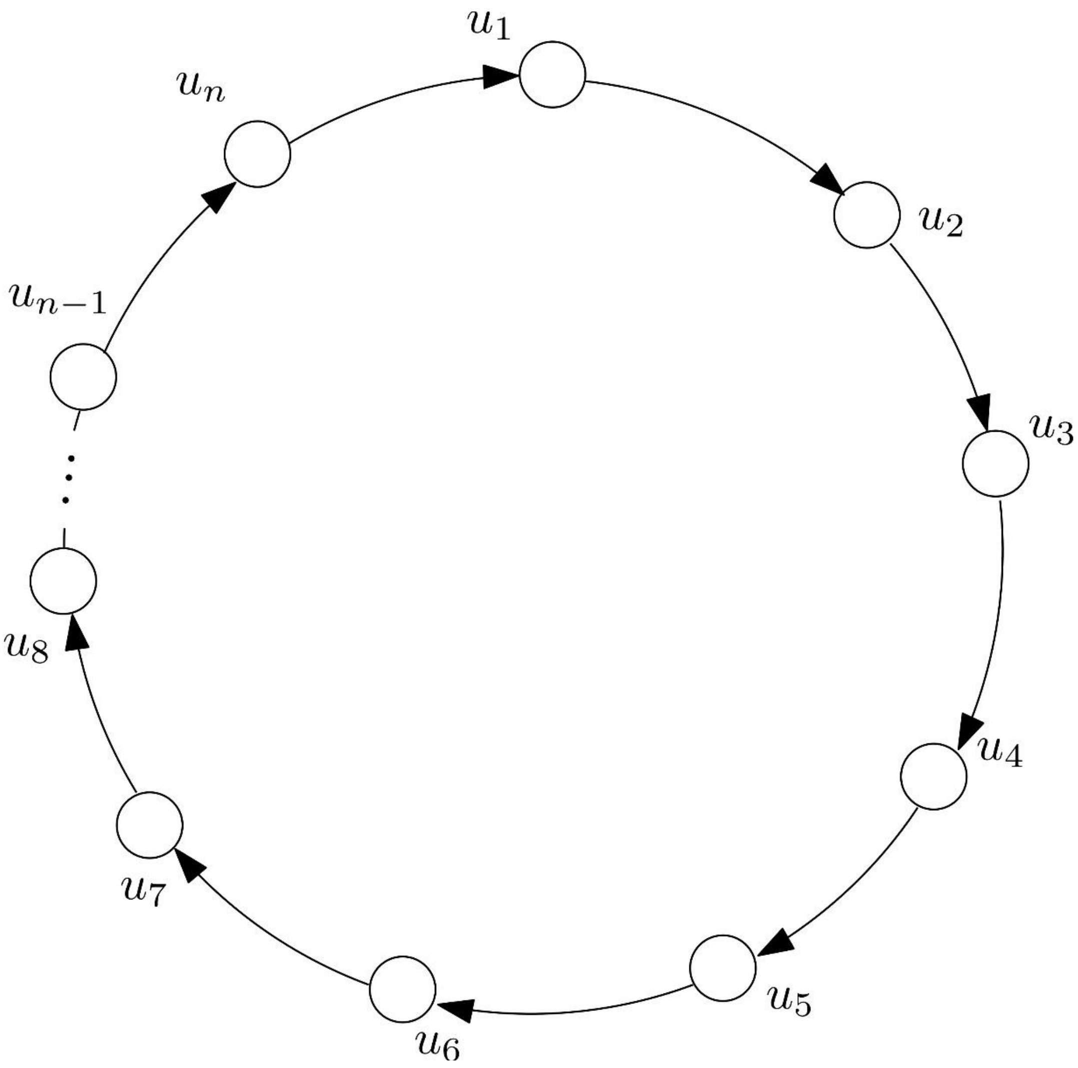
The Ring Network

- Bidirectional (or undirected)



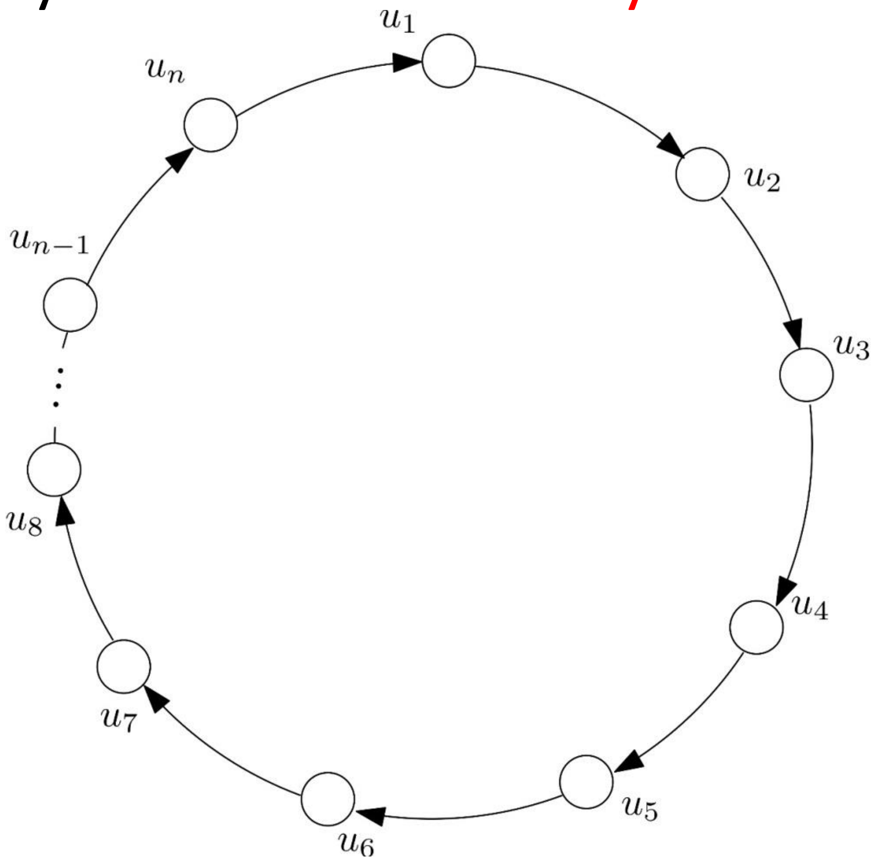
The Ring Network

- Unidirectional (or directed)



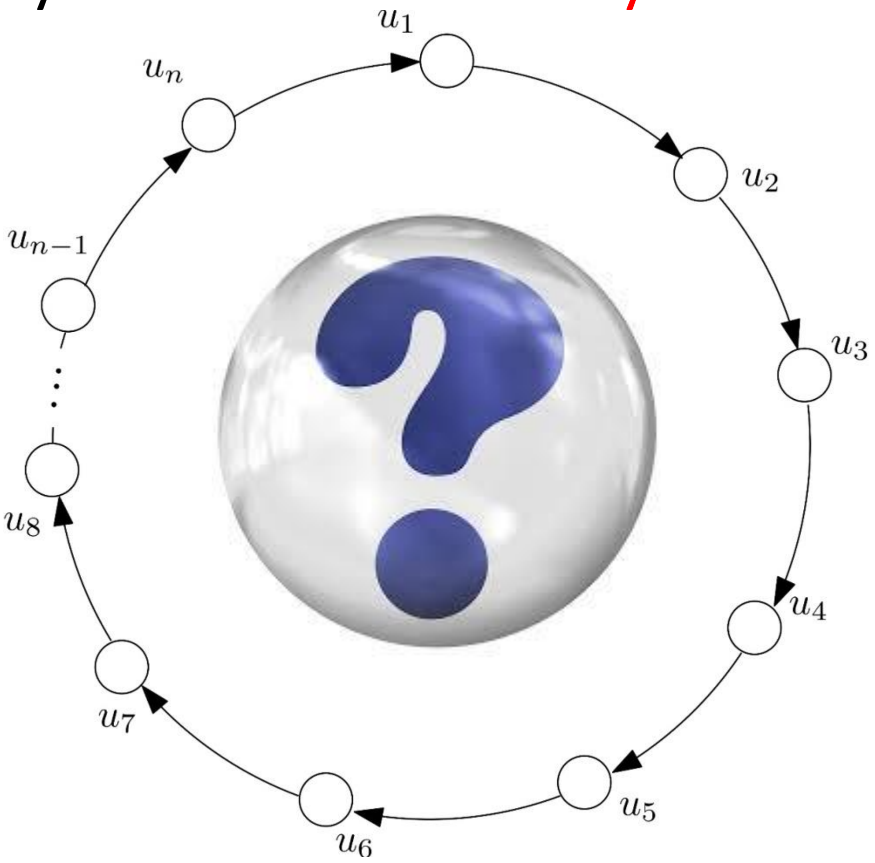
A First Minimal Setting

- Directed ring
- All processors are initially identical
  - Meaning here that they all start from exactly the same initial state



A First Minimal Setting

- Directed ring
- All processors are initially identical
  - Meaning here that they all start from exactly the same initial state



Question:

Is there an algorithm that solves leader election?

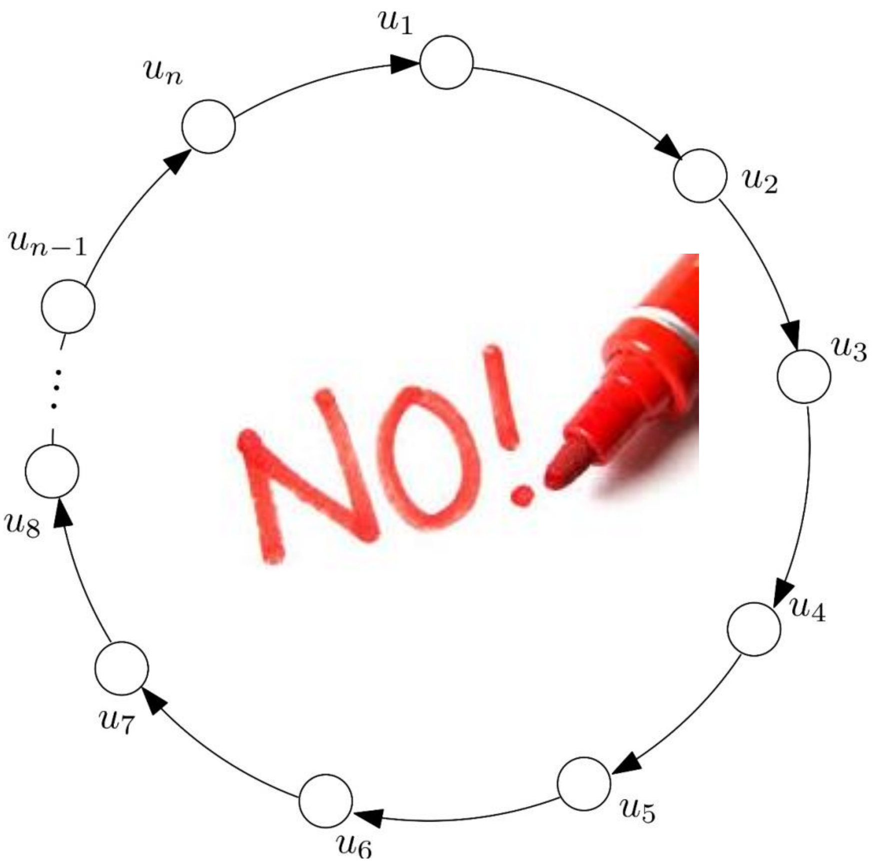
Our First Impossibility Result

Question: *Is there an algorithm that solves leader election?*

Answer: No

Very general:

No matter which algorithm you try, it will fail!



Impossibility of Leader Election with Identical Processors

Theorem. Let *G* be a directed ring of *n* processors. Take any algorithm *A* (to be executed in *G*), in which all processors are initially in exactly the same state. Then *A* does not solve the leader election problem.

Proof Idea.

Observation 1: To solve leader election, at some point one processor must be in a different state than the rest

Impossibility of Leader Election with Identical Processors

Proof Idea (continued).

Observation 2: If all processors are identical, given that

(i) they also have identical neighbourhoods and

(ii) they operate synchronously

we expect them all to do identical things

Summing-up: If we prove that all processors must forever remain identical (no matter which algorithm they execute), then we can conclude that they cannot elect a leader

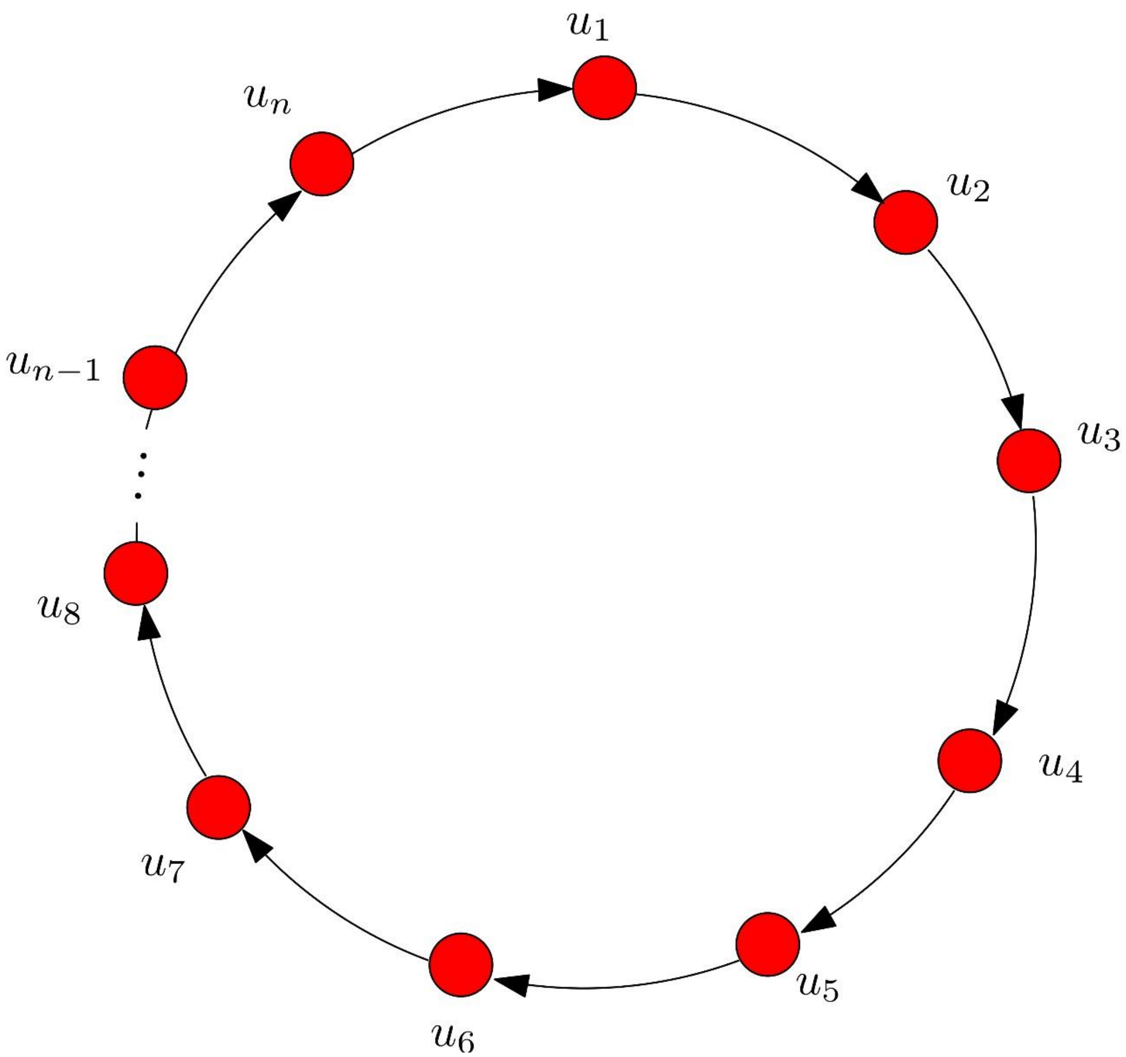
- Simply because a leader should be non-identical to the rest

Impossibility of Leader Election with Identical Processors

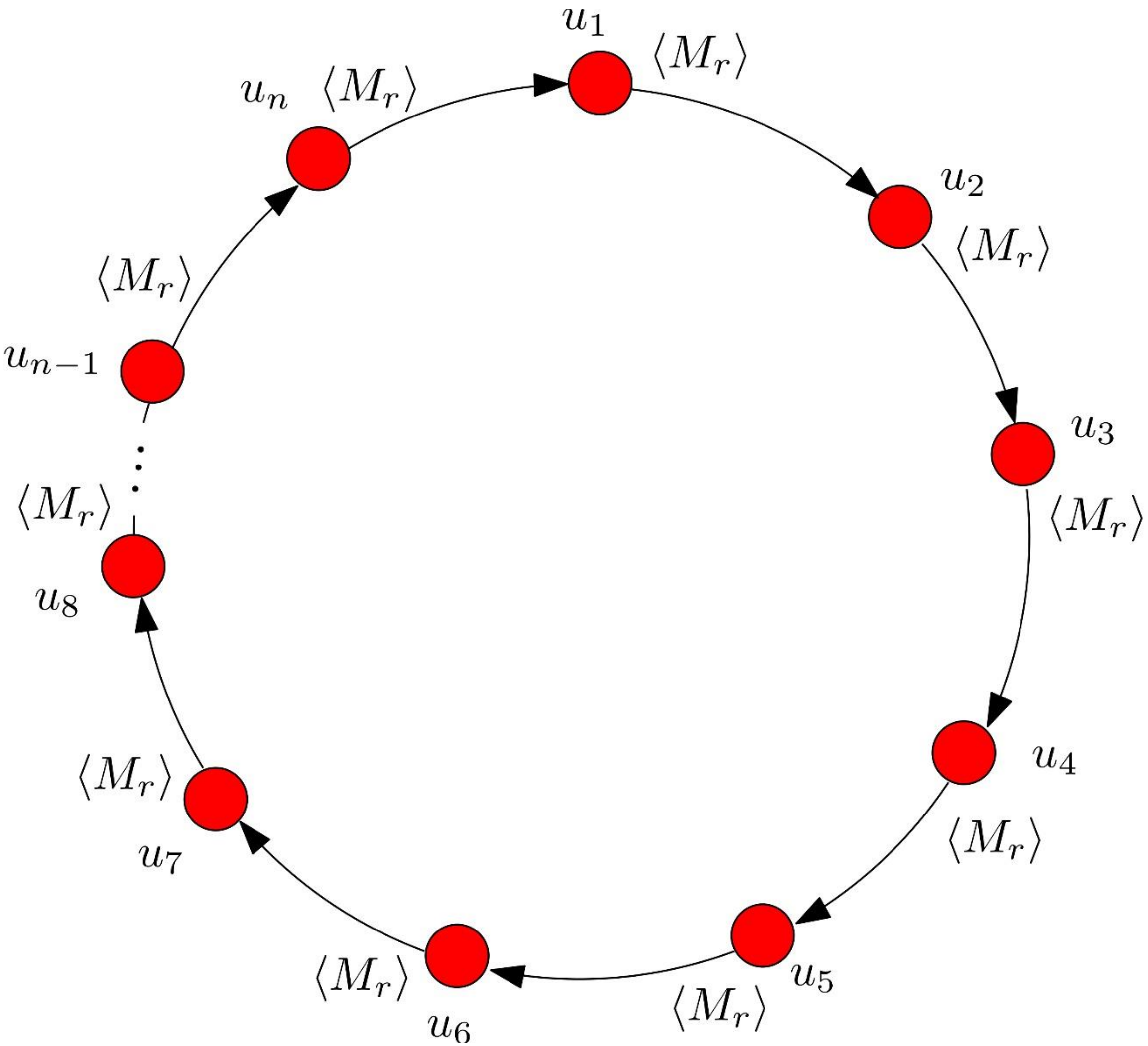
Proof. Take any algorithm *A*. We will show that in every round *r* all processors are in identical states.

- By induction on *r*
- *r* = 0: We know that in our setting all processors start from identical initial states
- Assume it holds for *r*
- *r* + 1: By assumption we know that it holds for *r*
  - That is, in round r all processors are in identical states
  - This means that they all produce the same message to be forwarded to their right neighbour (because the messages produced by a processor depend only on its state)
  - Thus, in *r* + 1 all processors are in identical states and receive the same incoming message
  - Therefore, they will perform the same state-update and will again obtain identical states in *r* + 1 (because the new state of a processor depends only on its previous state and the messages received) □

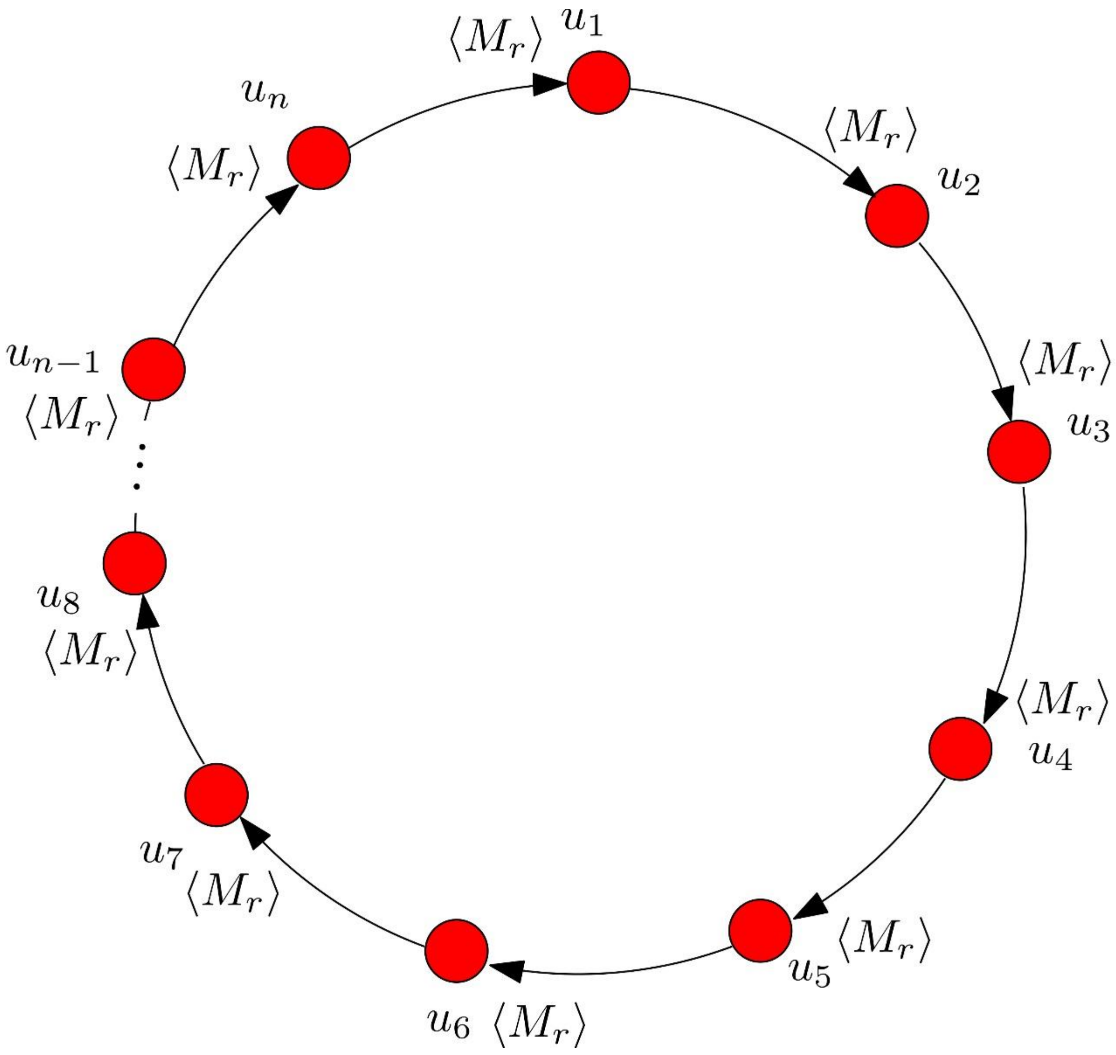
An Illustration



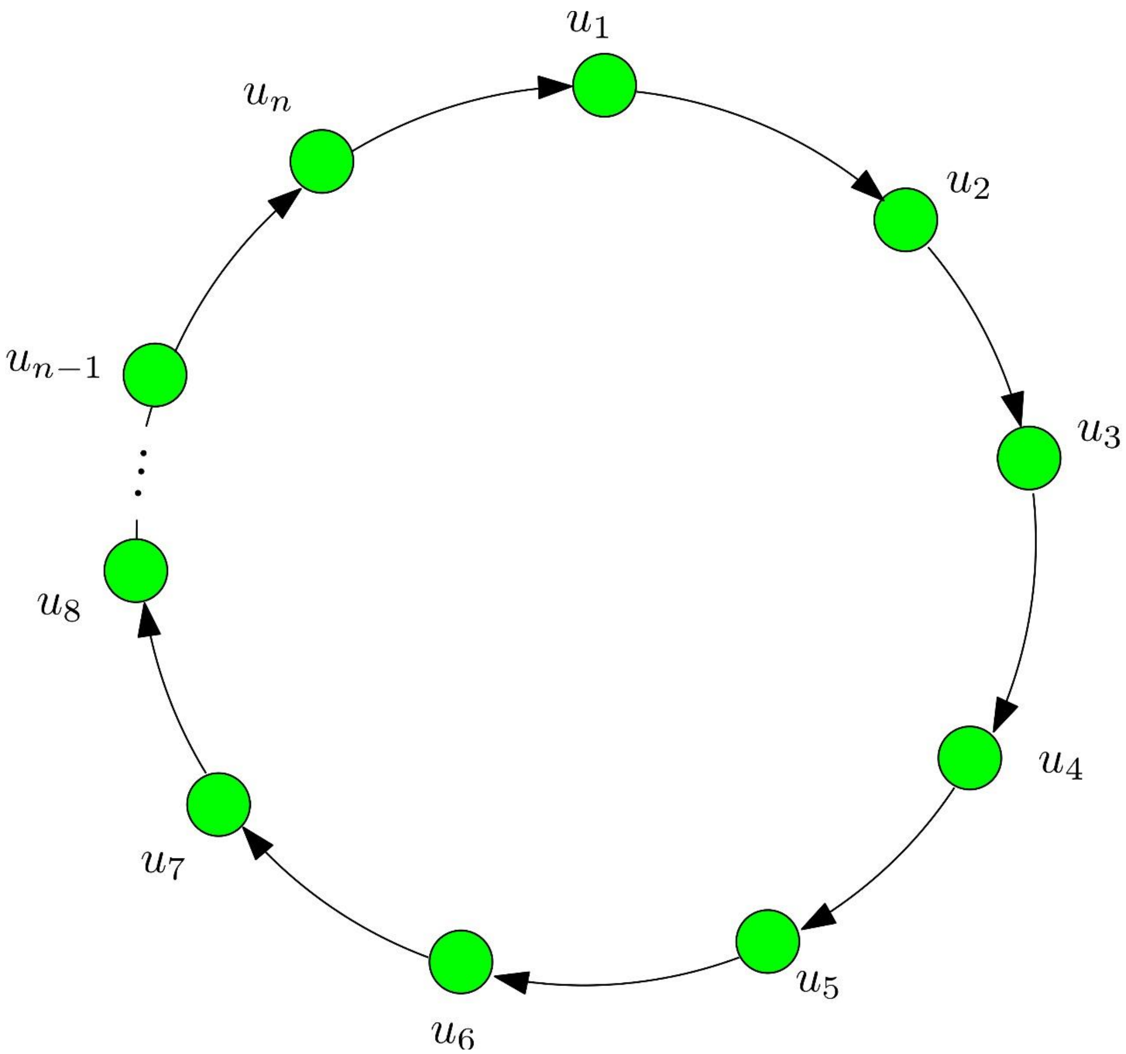
An Illustration



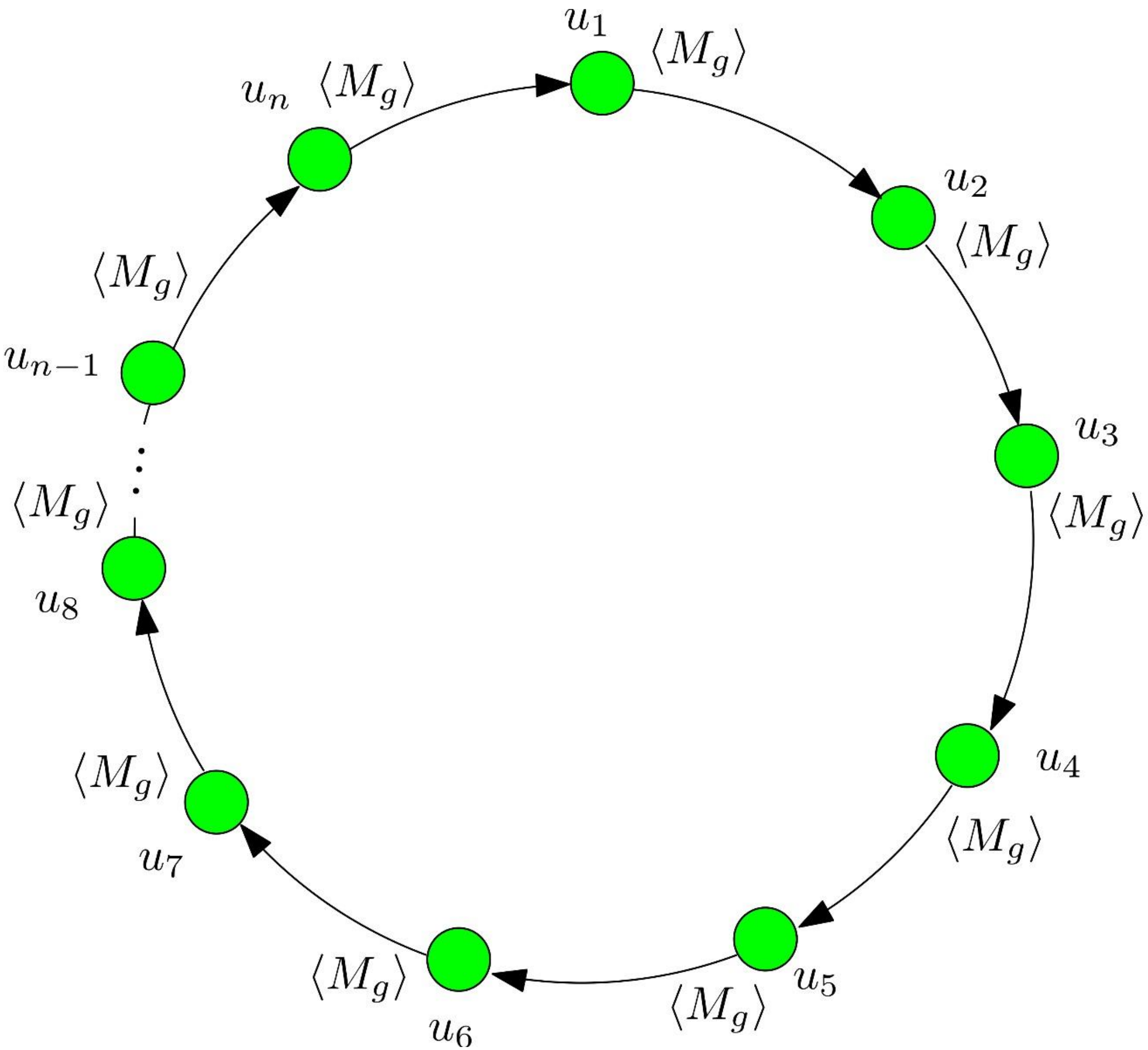
An Illustration



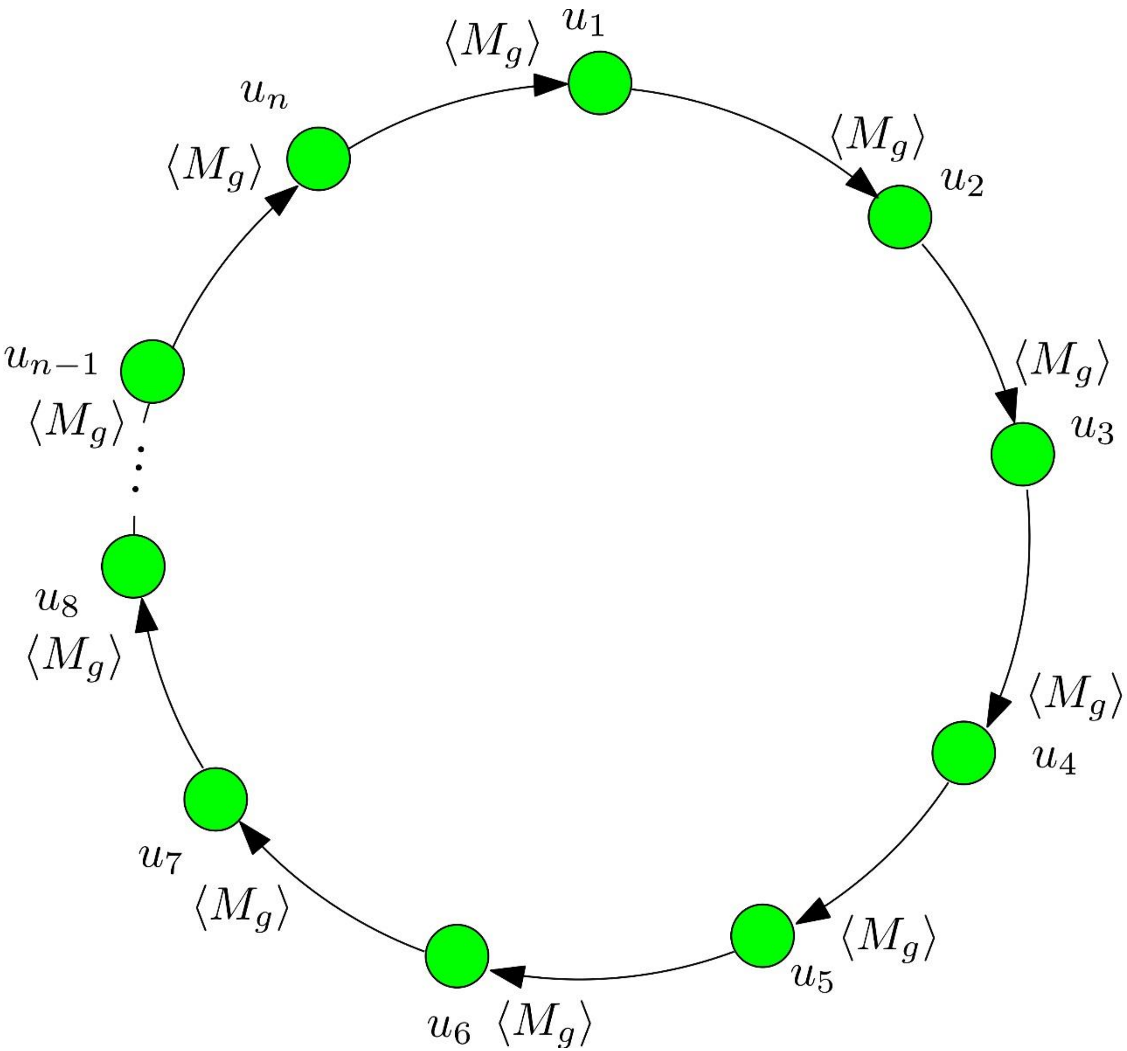
An Illustration



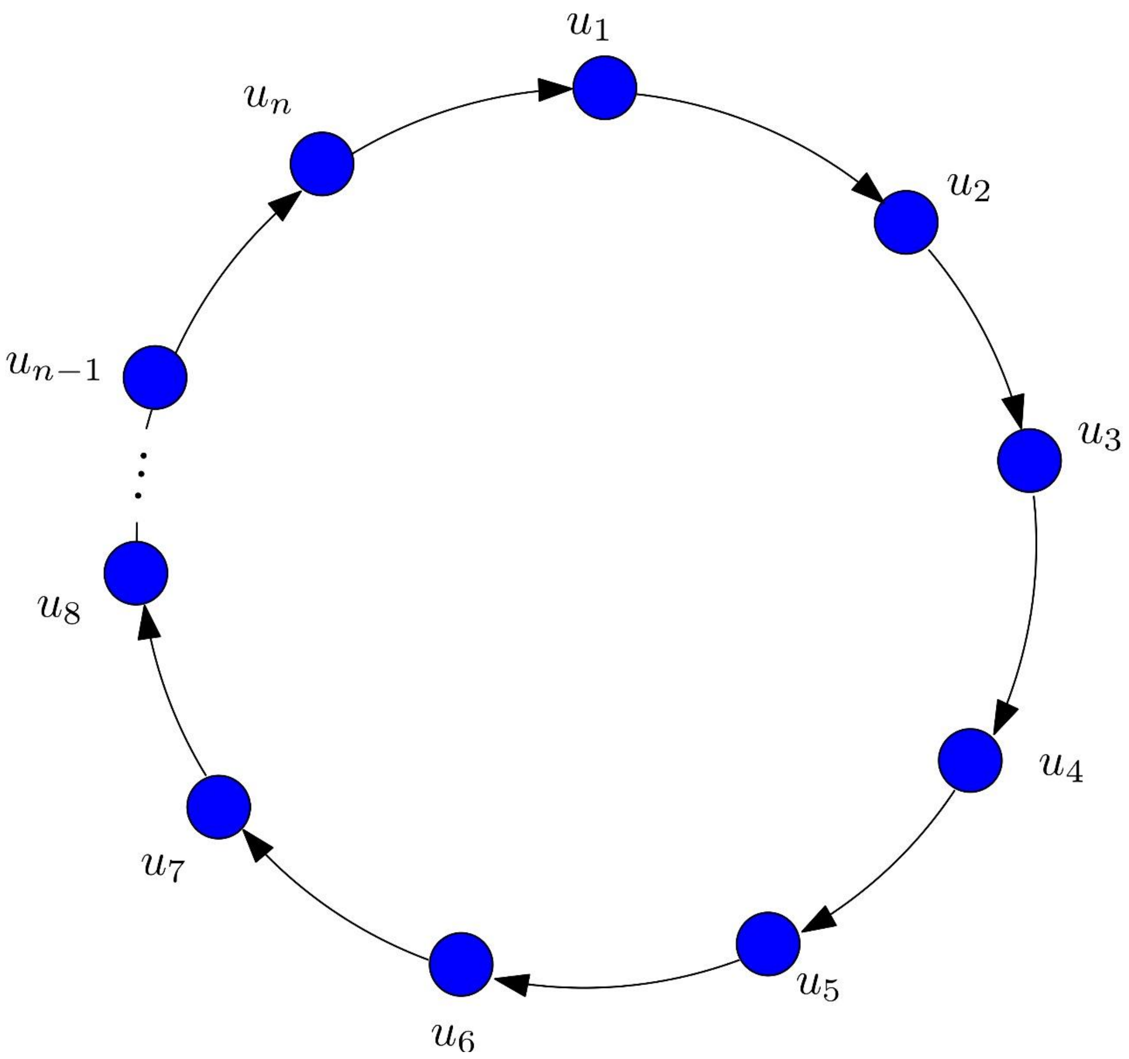
An Illustration



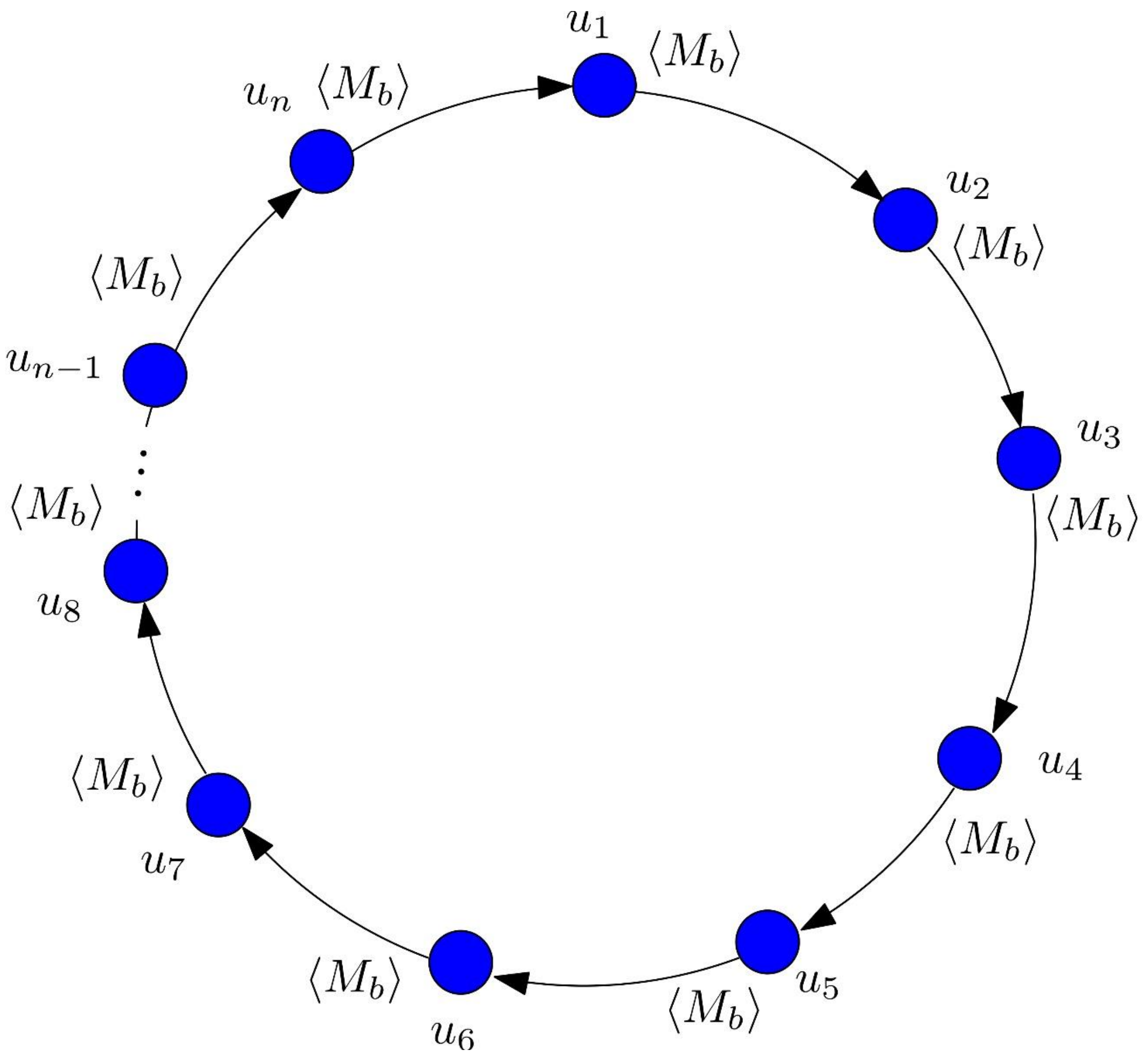
An Illustration



An Illustration



An Illustration



Interpretation

- Just proved that all algorithms will fail
- Therefore, not a drawback of a specific algorithm rather an inherent limitation in the setting under consideration
- Reveals the difficulties related to symmetry in distributed systems
  - Not always possible to break symmetry
  - Due to the independent/autonomous nature of processors they may very well be doing identical things
  - e.g., many of them regenerating the lost token even though only 1 is needed

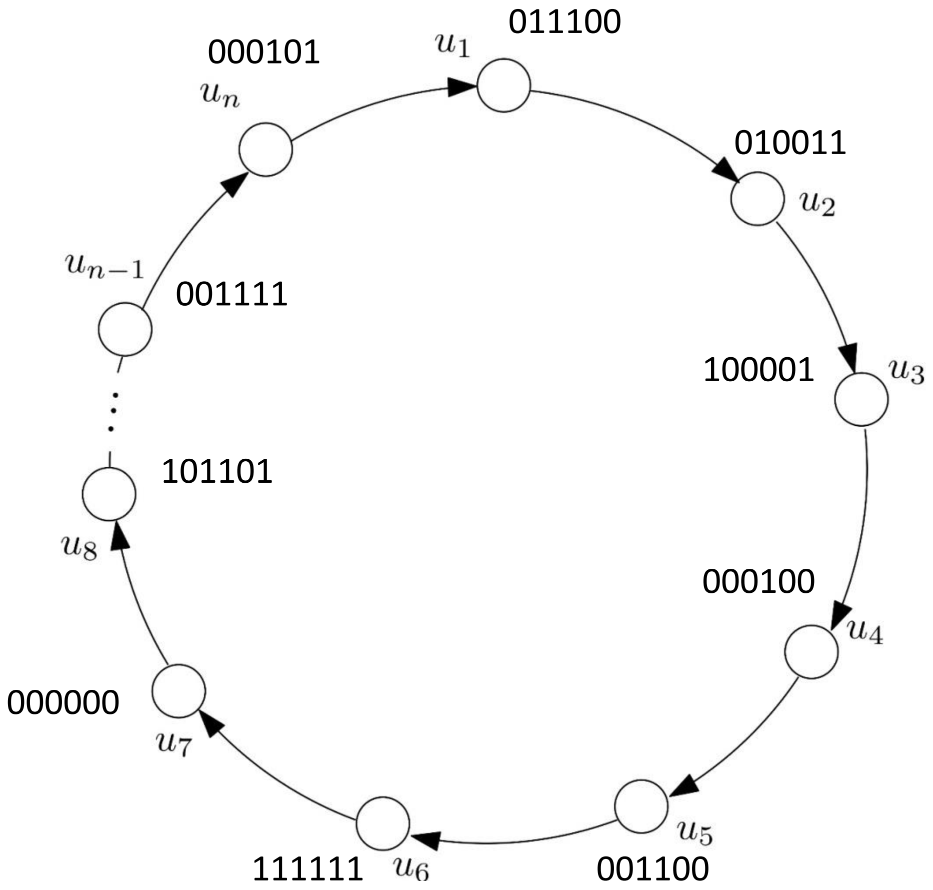
Back to Optimism

- That we cannot solve a problem in a particular setting does not mean that we cannot solve it in all settings
  - Particularly true for distributed systems: even small modifications of the setting may greatly affect solvability/efficiency
- Immediate thought: *What if there is no such inconvenient symmetry initially?*
  - *Still allow some components of the processors states to be identical*
  - *But now all processors have distinct/unique initial states…*
  - *Any ideas?*

A Less Symmetrical Setting

Every processor has a unique id

- Largely available in distributed systems
- e.g., a MAC address
- Requires only *O*(log *n*) bits in local memory
- Not necessarily consecutive
- A processor only knows its own id initially



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: Informal description

- All processors send initially their id clockwise
- Upon receiving an incoming id, compare it to your own
  - If *incoming id* > *own id*, forward the received
  - if *incoming id* < *own id*, discard the received
  - if *incoming id* = *own id*, declare yourself the leader

- Intuitively:
  - The maximum id will manage to perform a complete turn and return to its origin
  - Any other id will at some point meet a processor with greater own id and will be discarded before making a complete turn

LCR: Pseudocode

Algorithm LCR

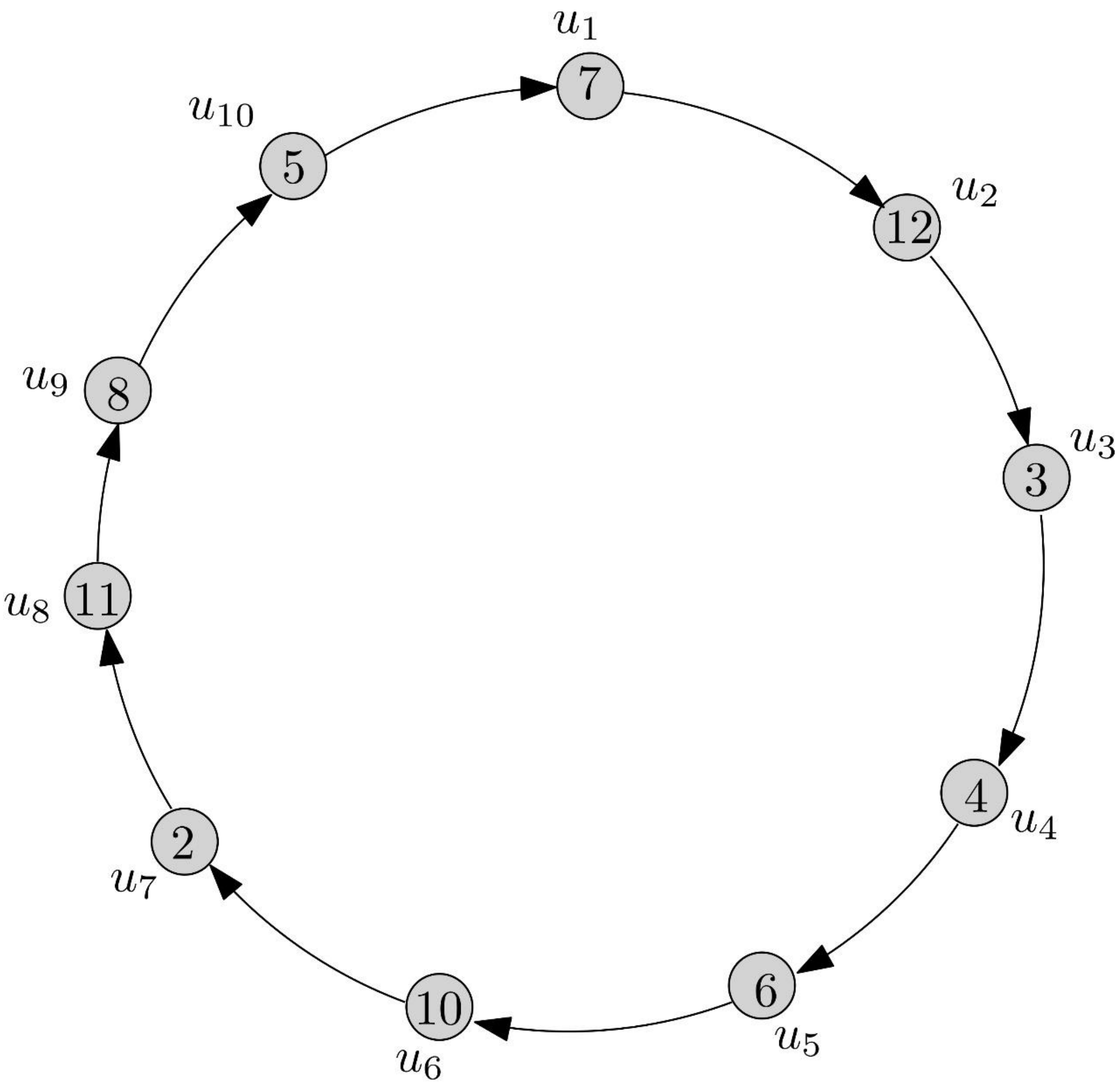
State of processor u_i :

- *myID*$_i$: holds the processor's unique id
- sendID$_i$: holds a unique id to be sent or *null*
- *status*$_i$ ∈ {"unknown", "leader"}: indicates whether u_i has been elected ("leader") or not ("unknown")

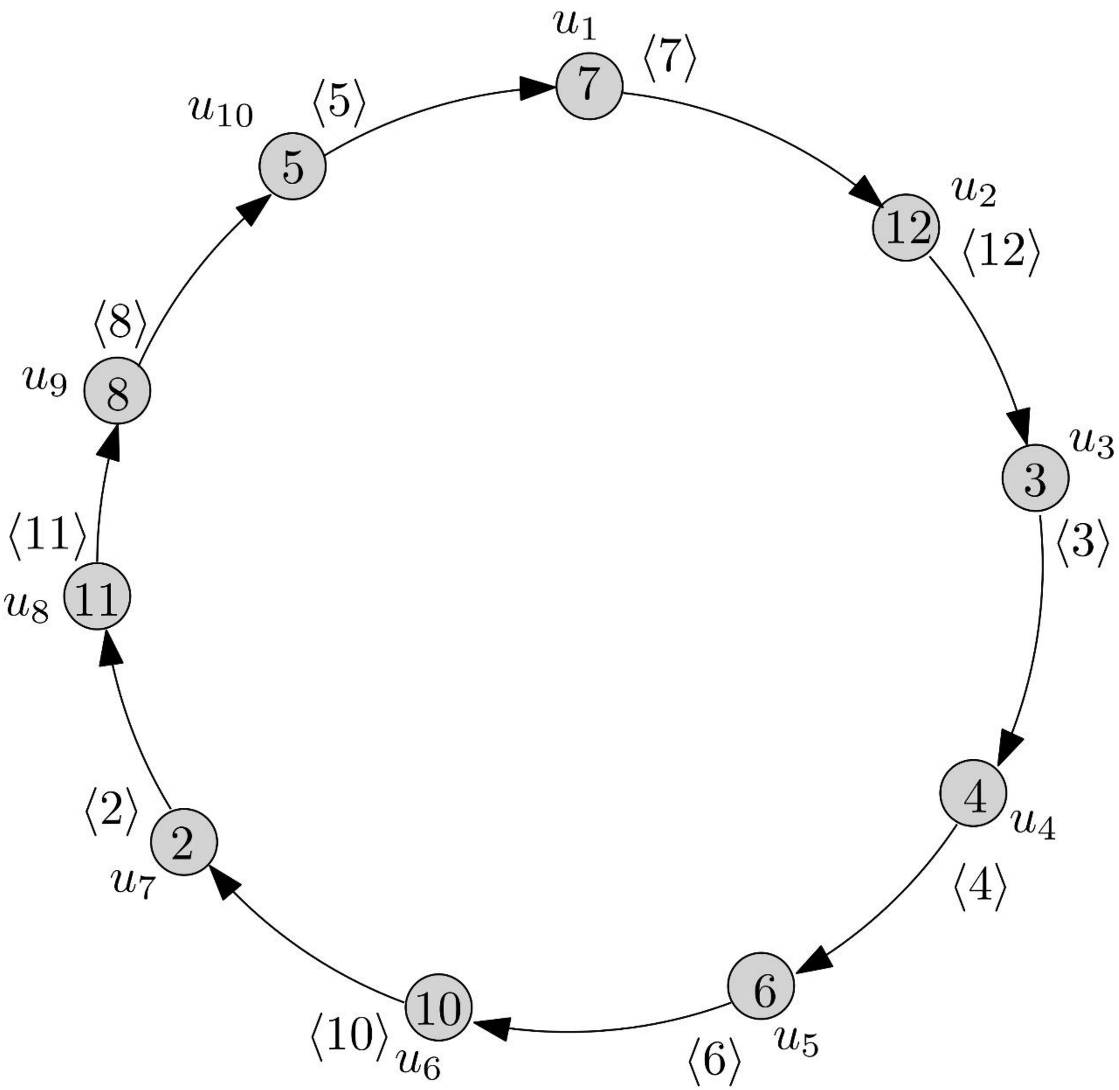
LCR: Pseudocode

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

Example Execution

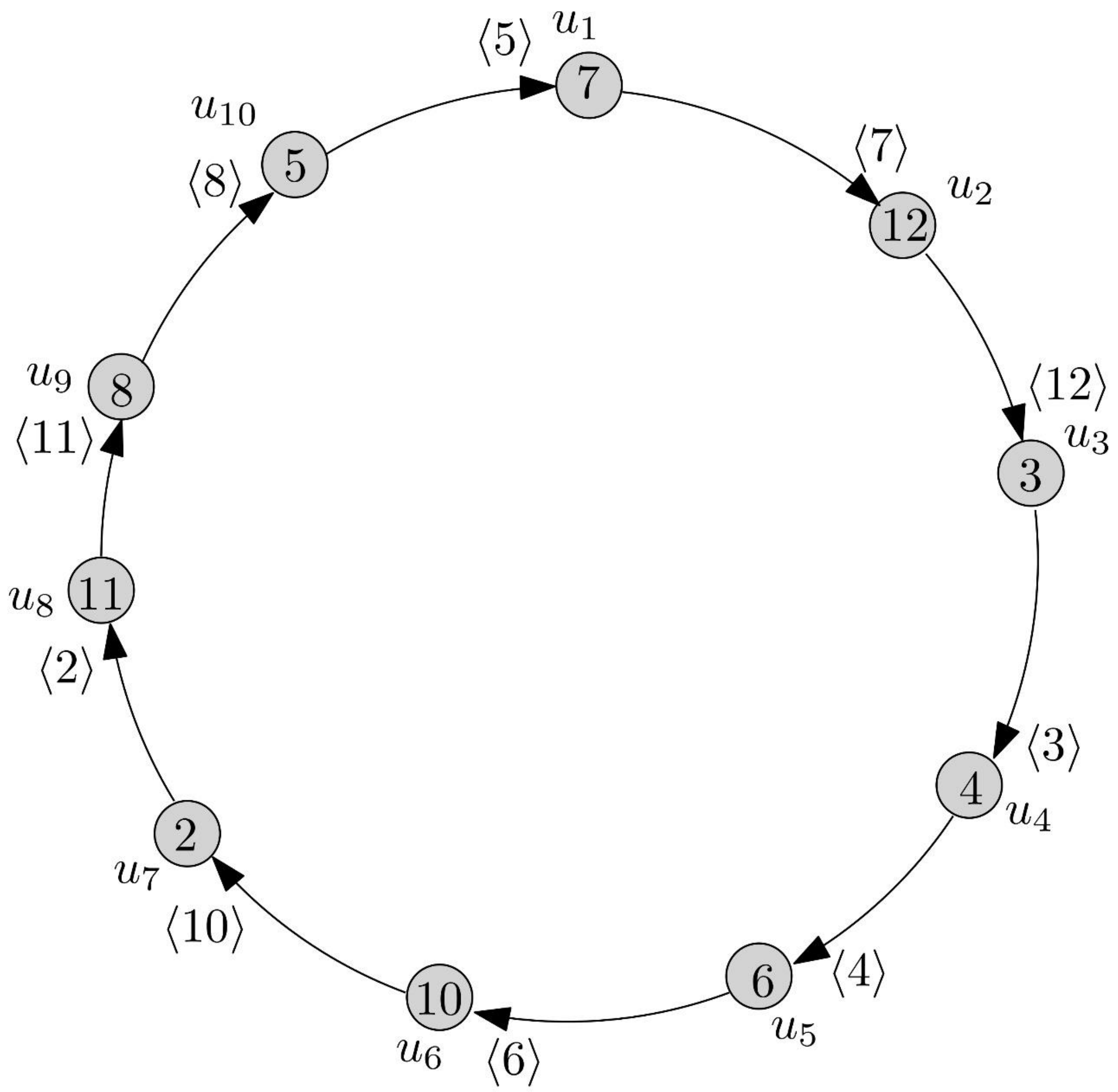


Example Execution



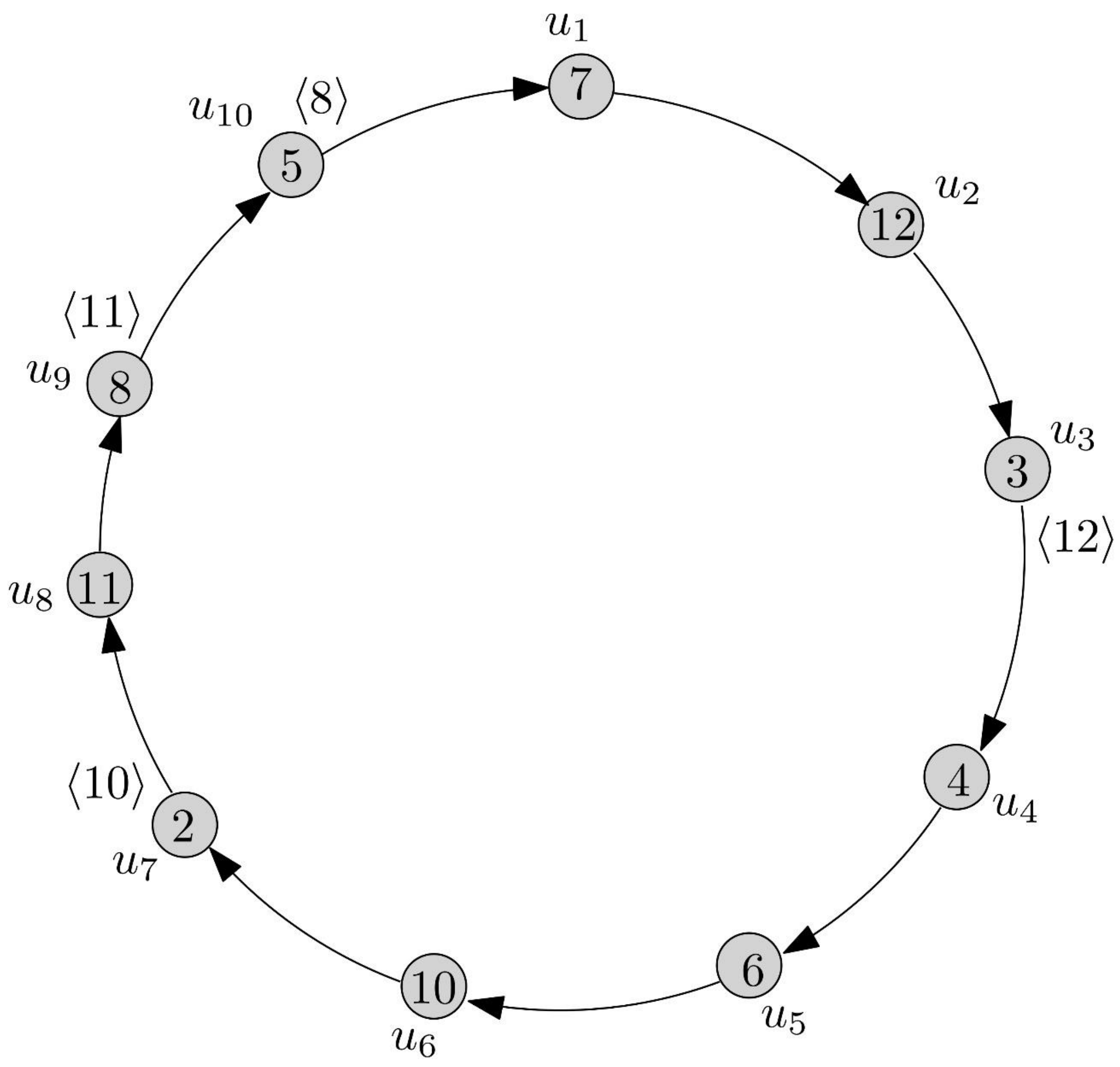
$round = 1$

Example Execution



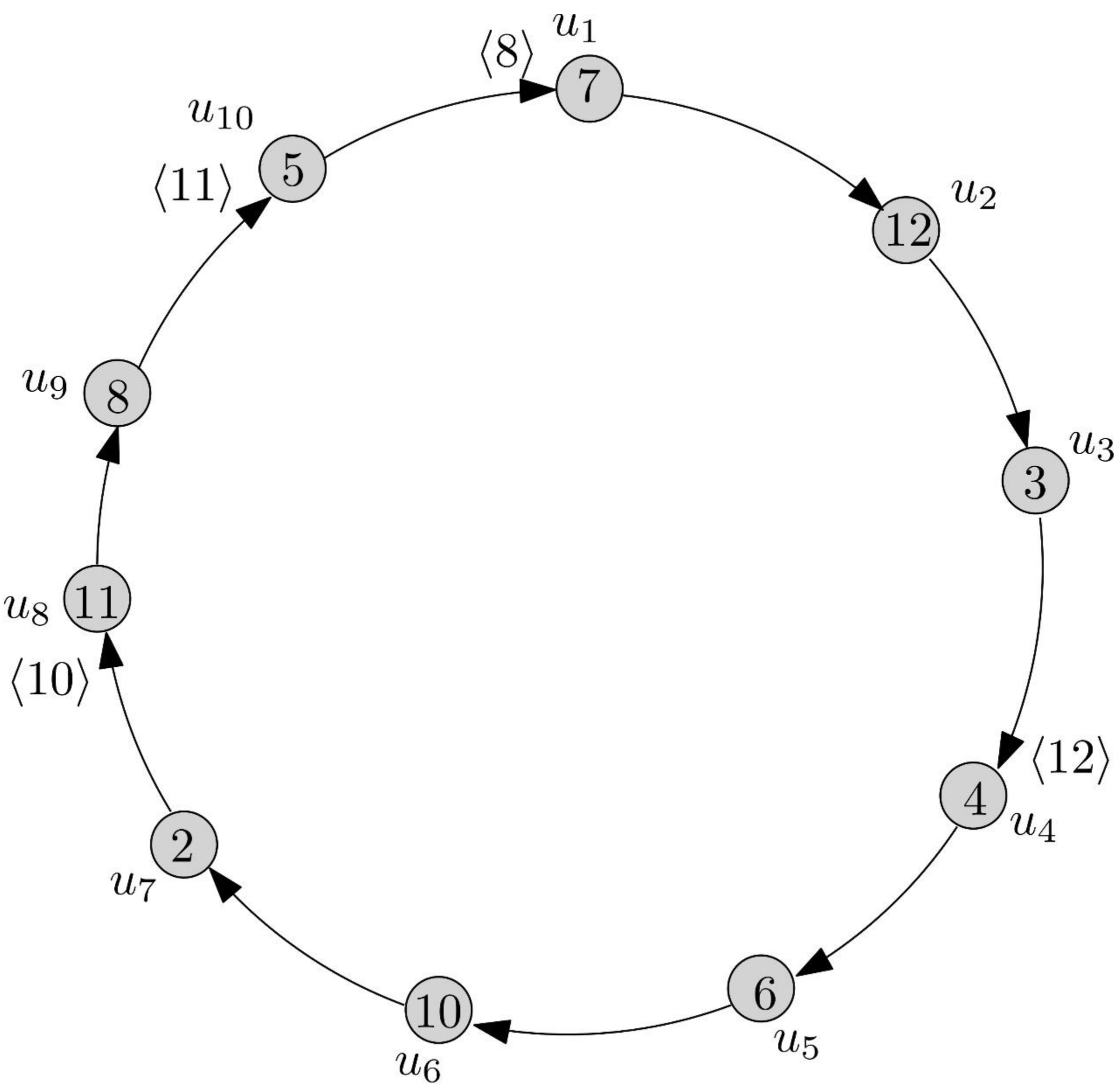
$round = 1$

Example Execution



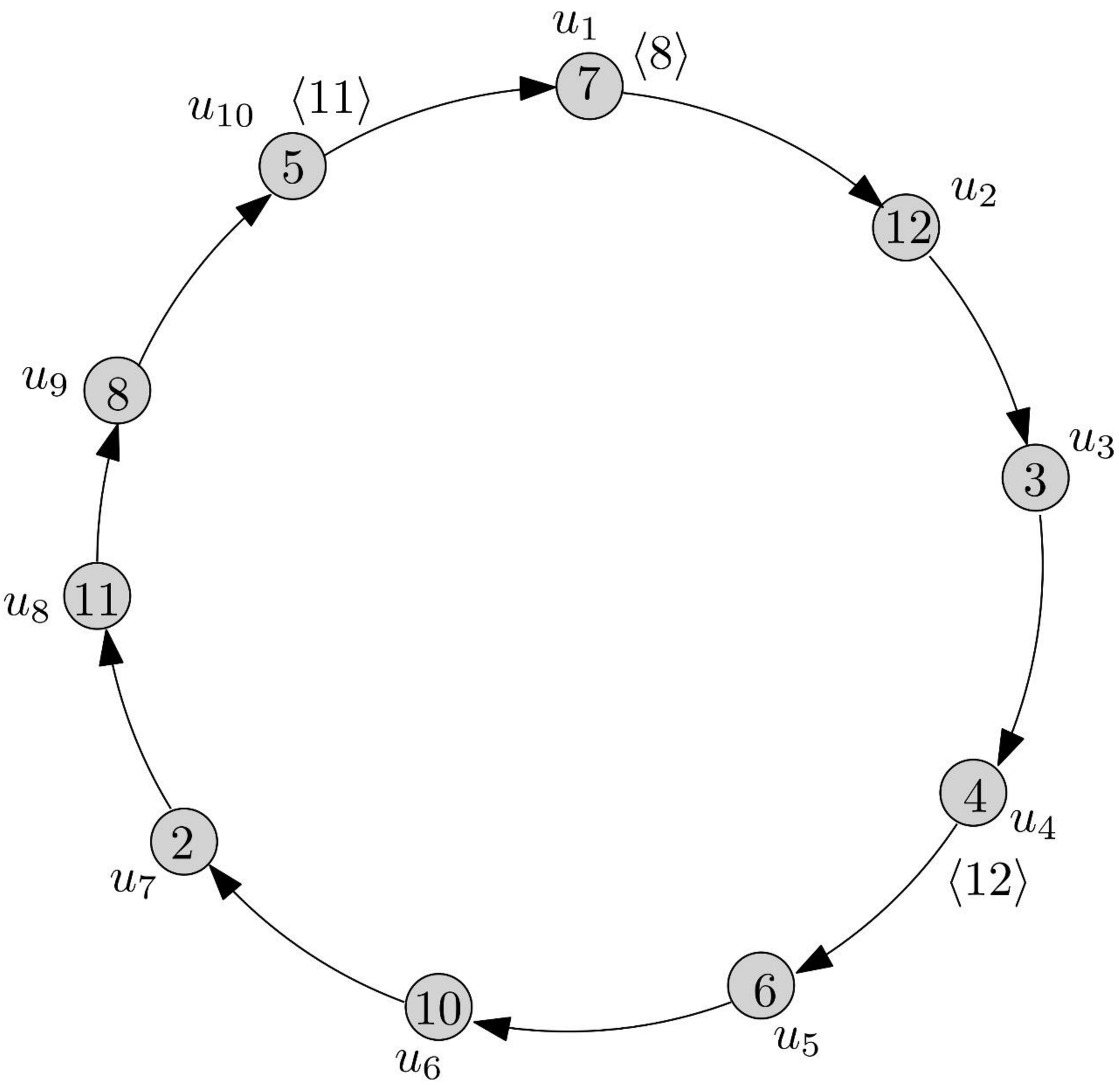
$round = 2$

Example Execution



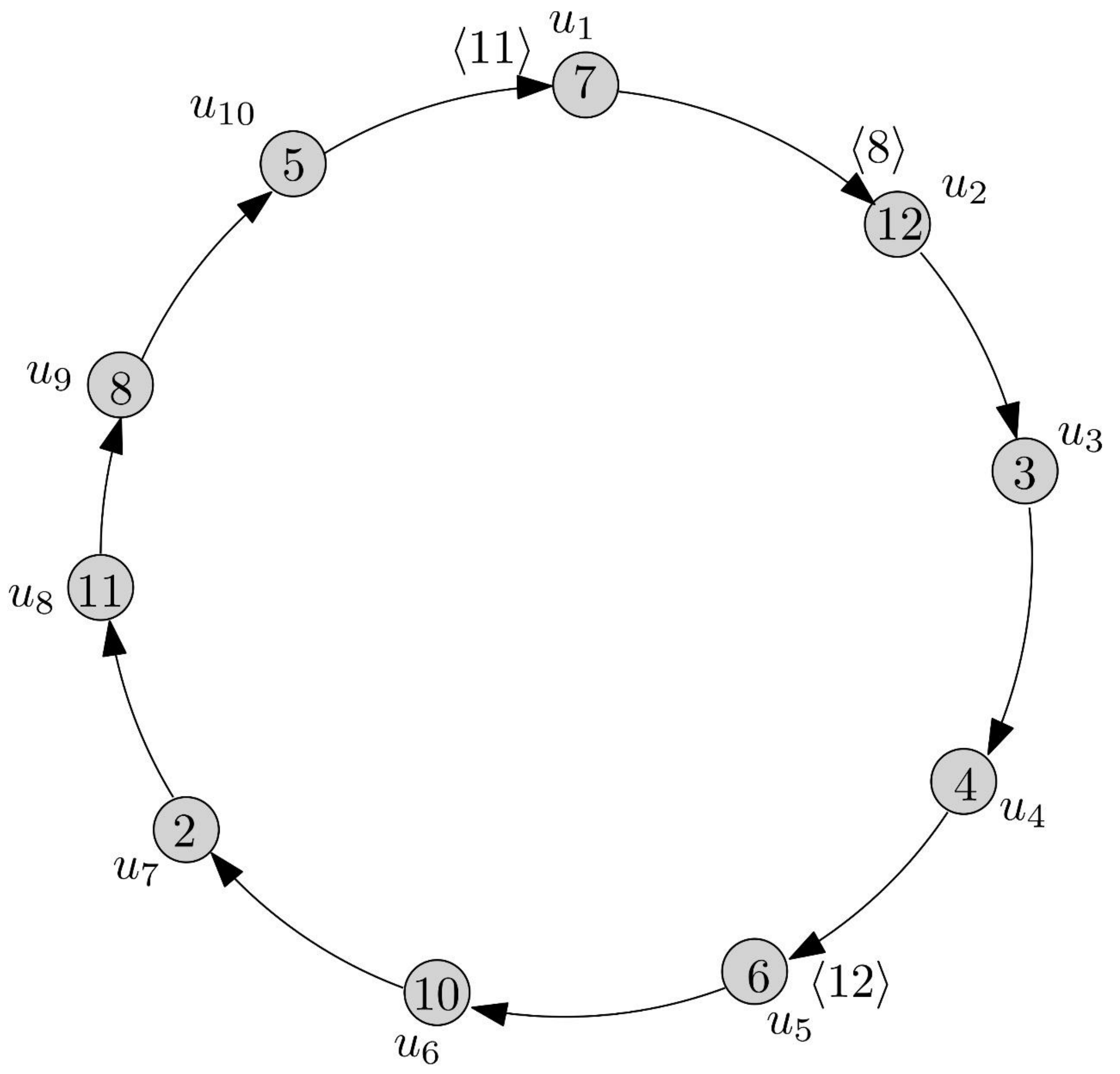
$round = 2$

Example Execution



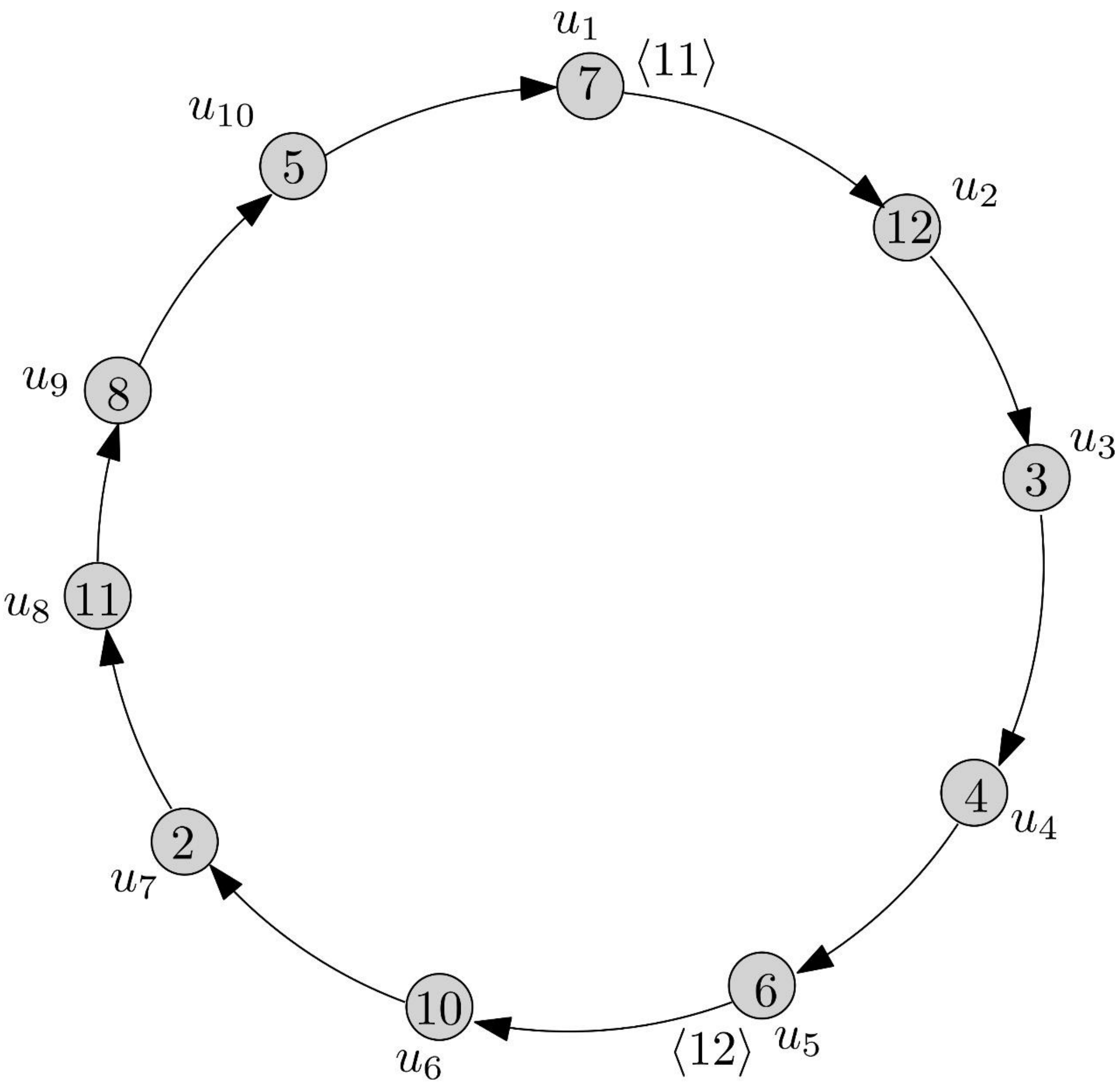
$round = 3$

Example Execution



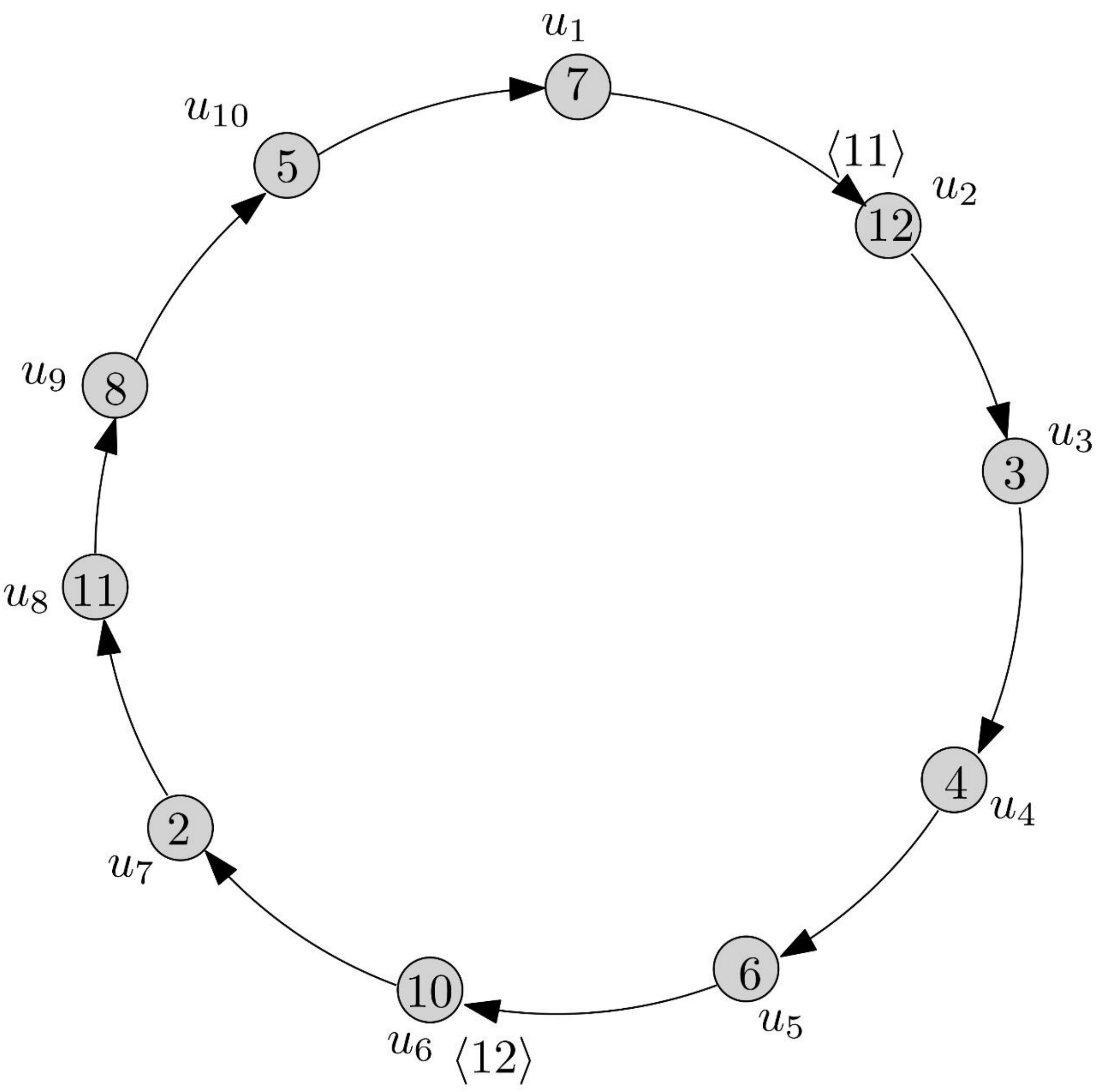
$round = 3$

Example Execution



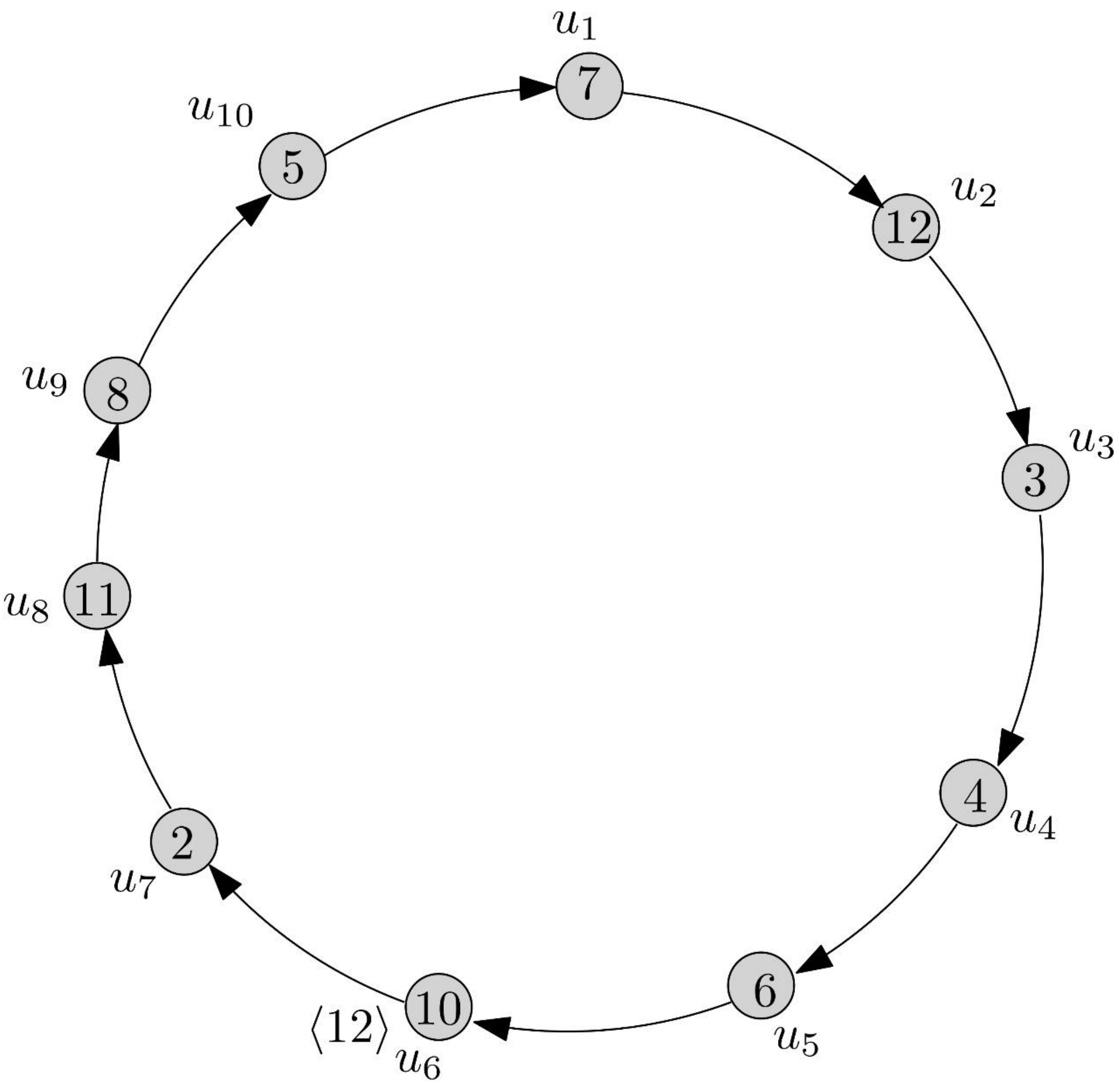
$round = 4$

Example Execution



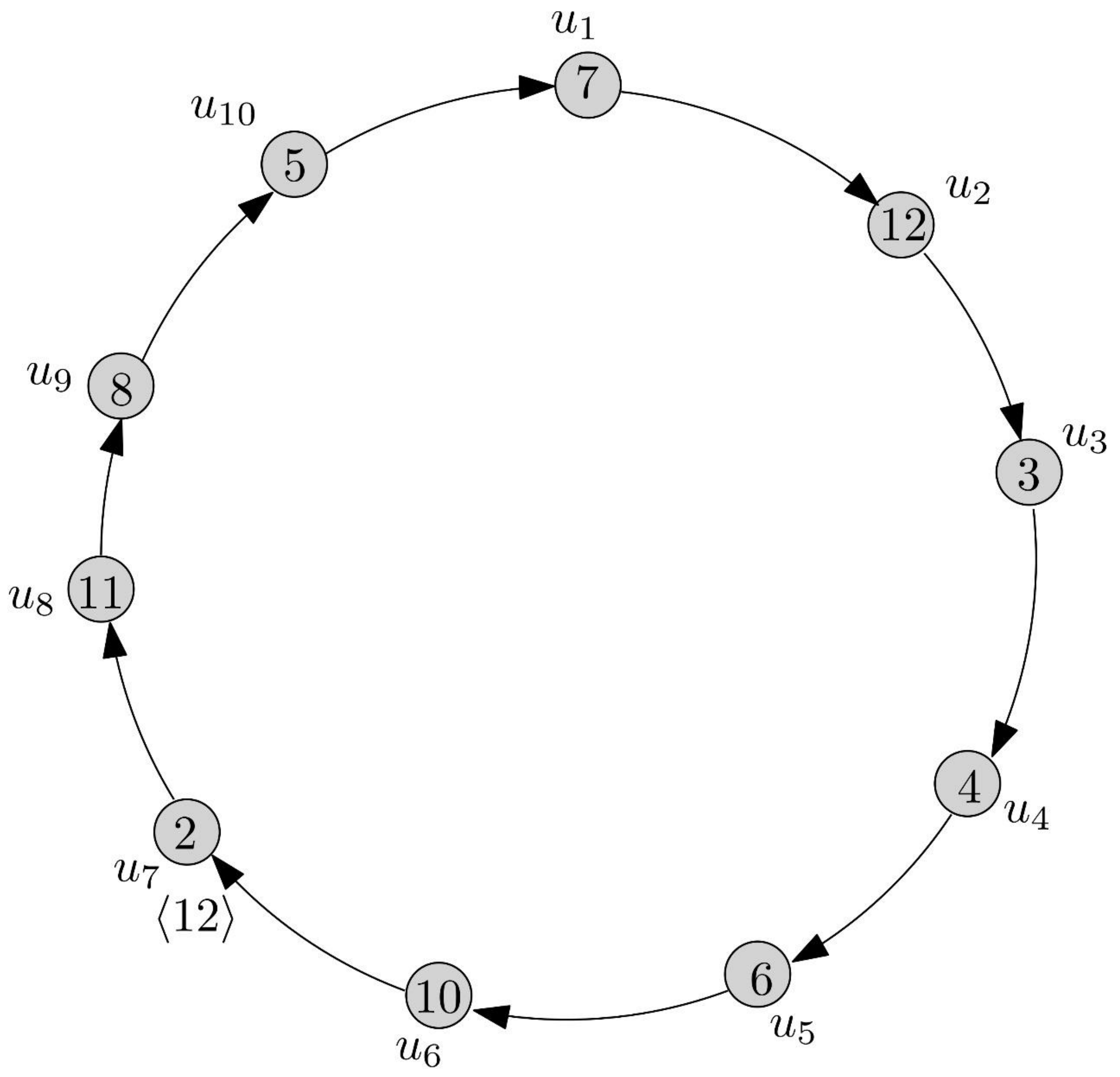
$round = 4$

Example Execution



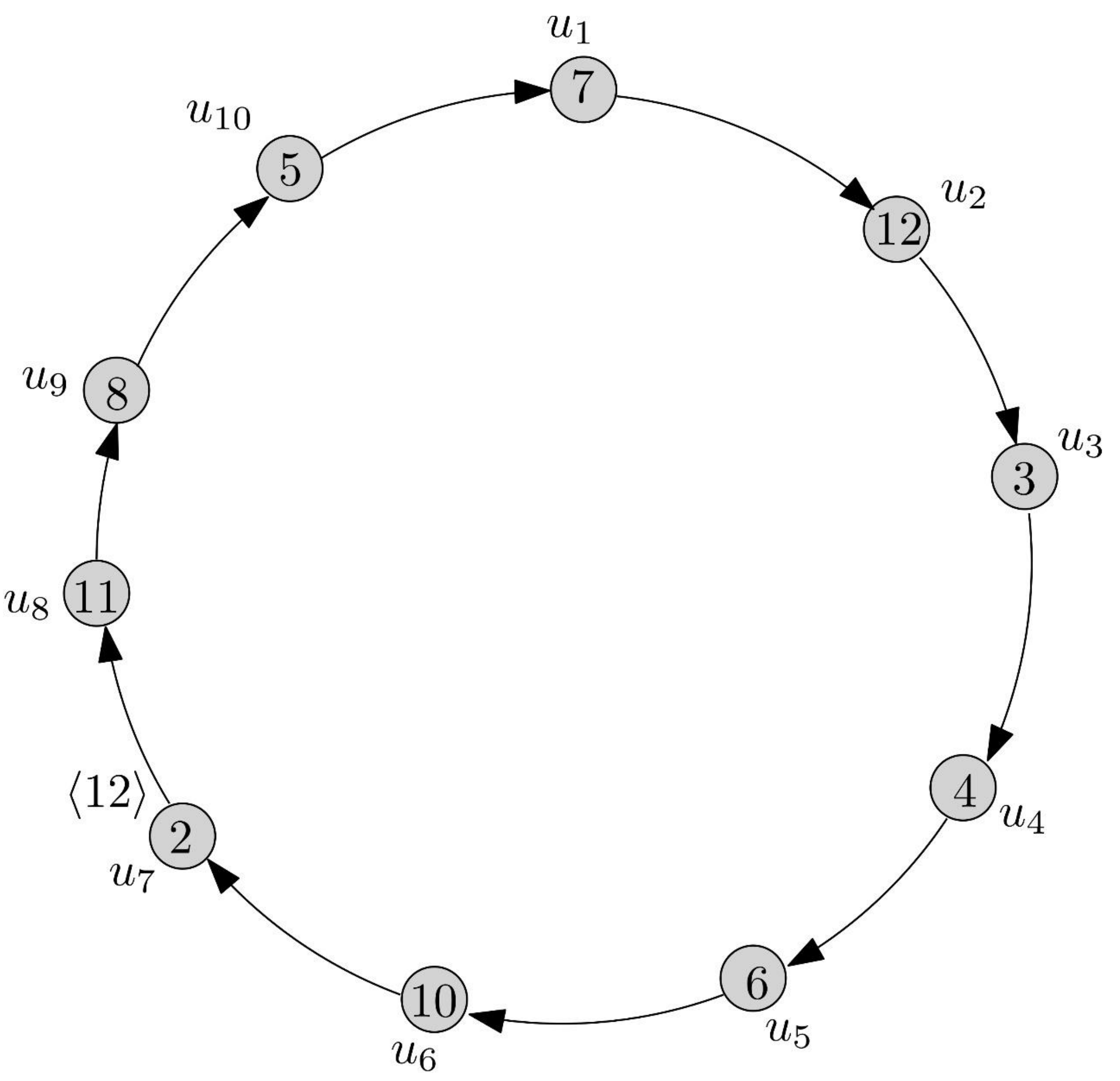
$round = 5$

Example Execution



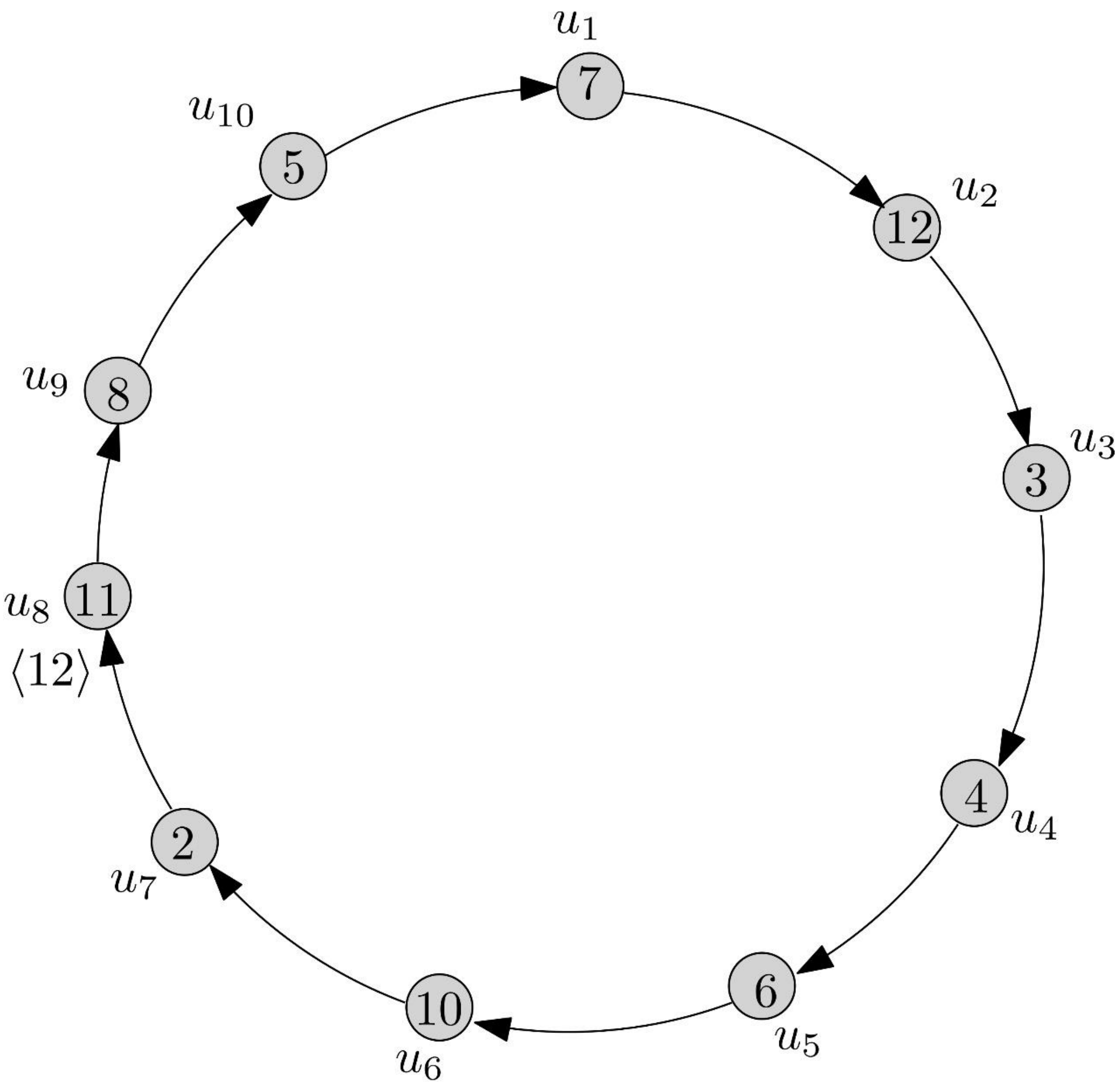
$round = 5$

Example Execution



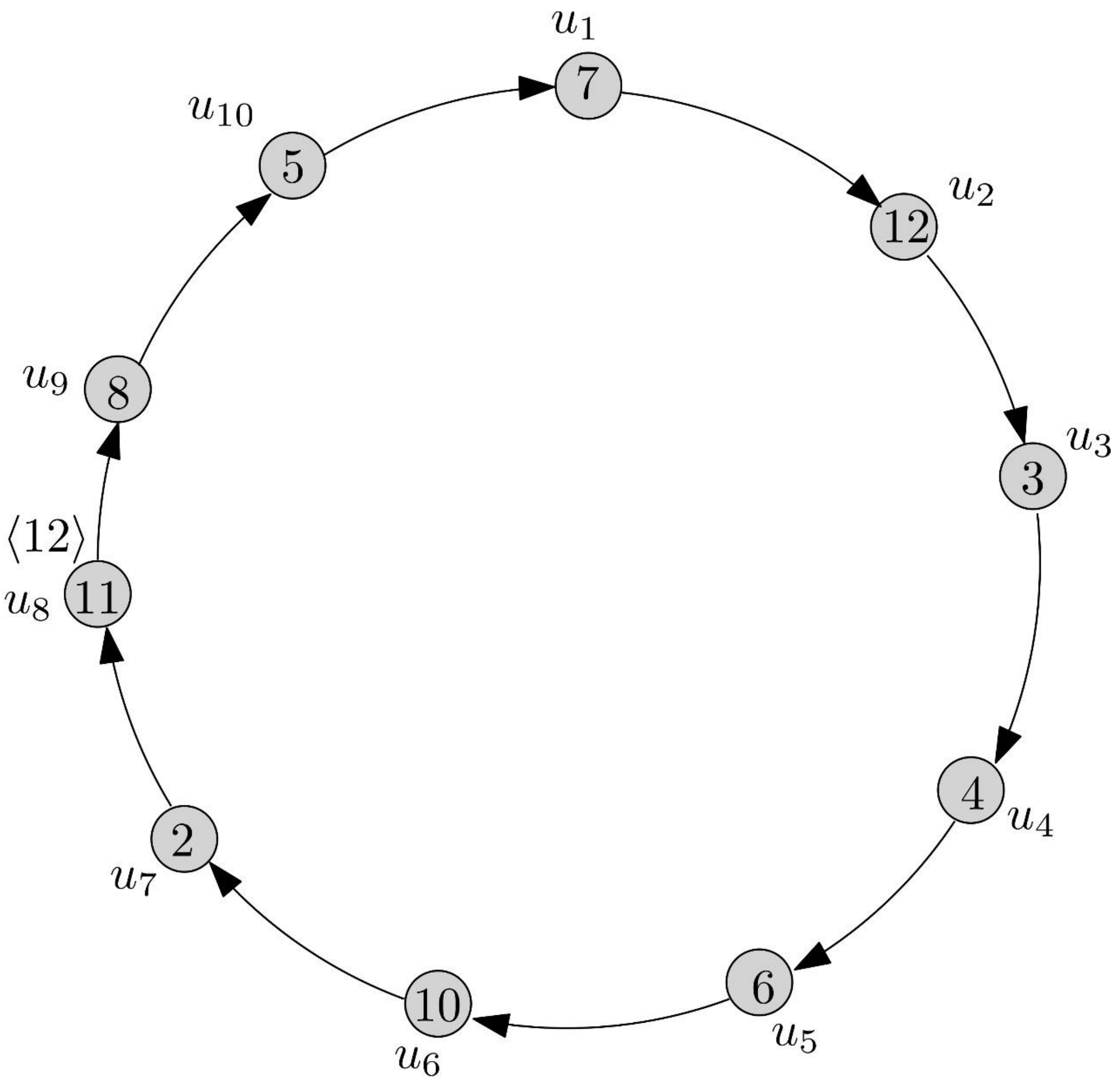
$round = 6$

Example Execution



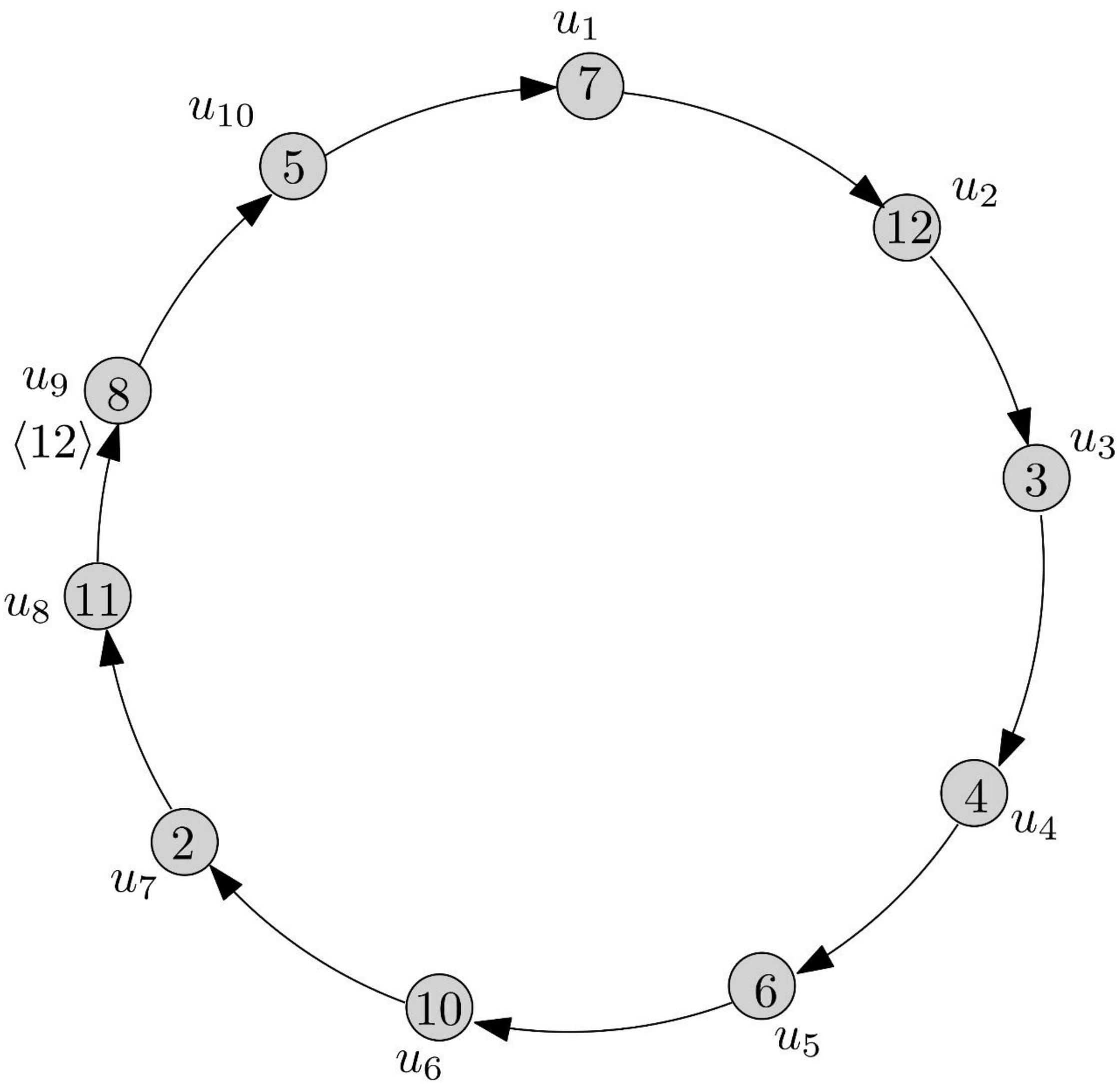
$round = 6$

Example Execution



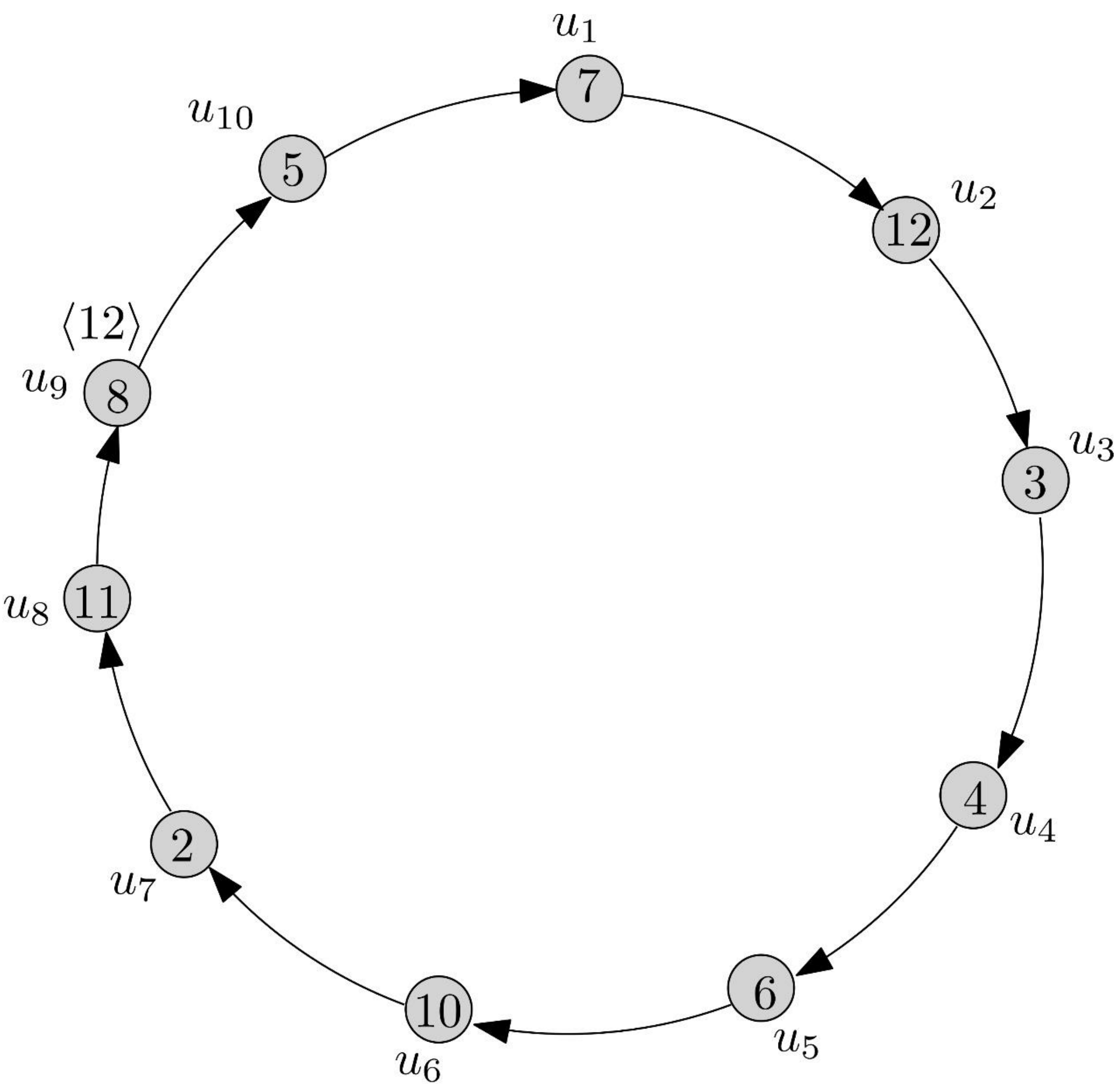
$round = 7$

Example Execution



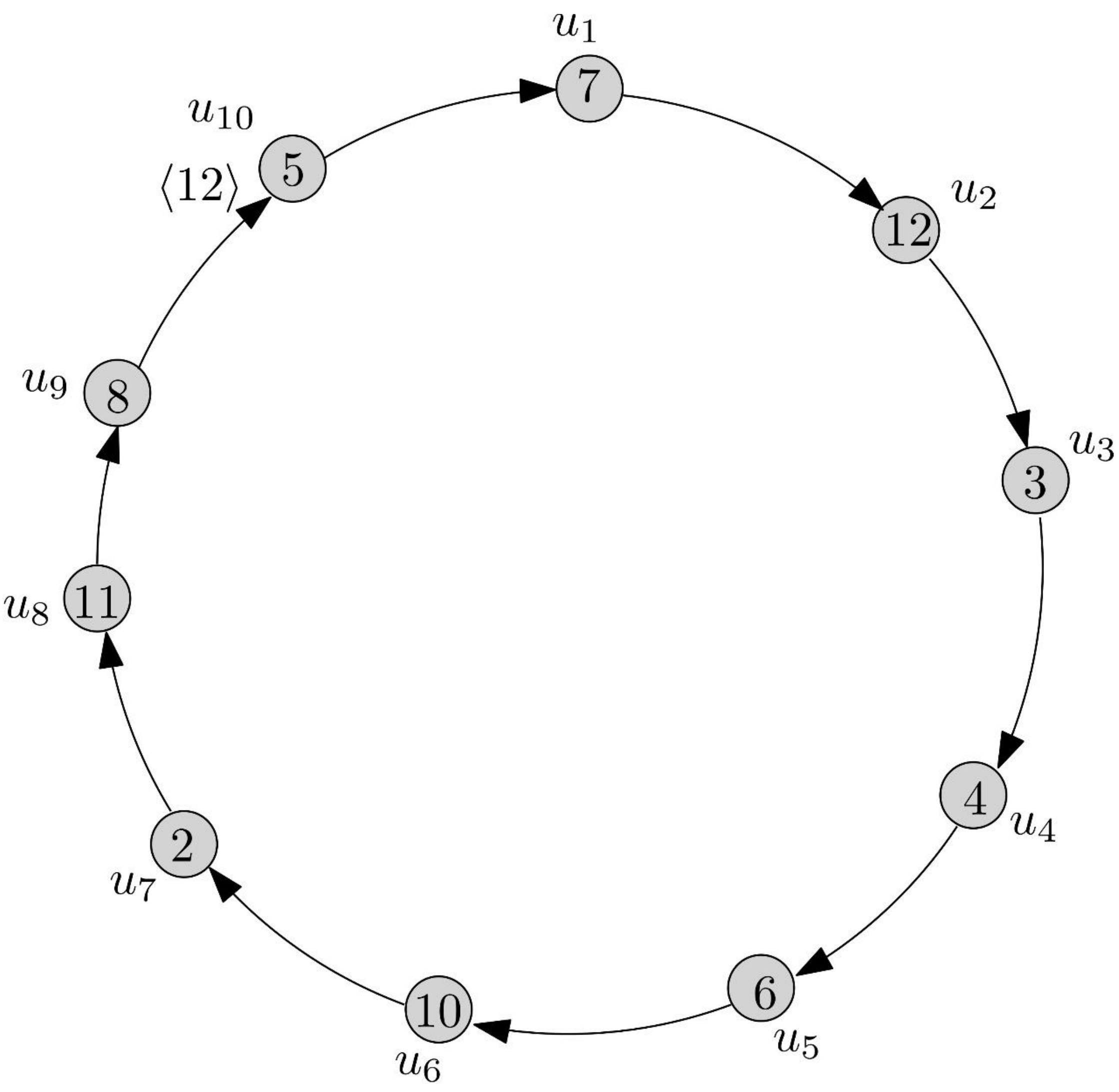
$round = 7$

Example Execution



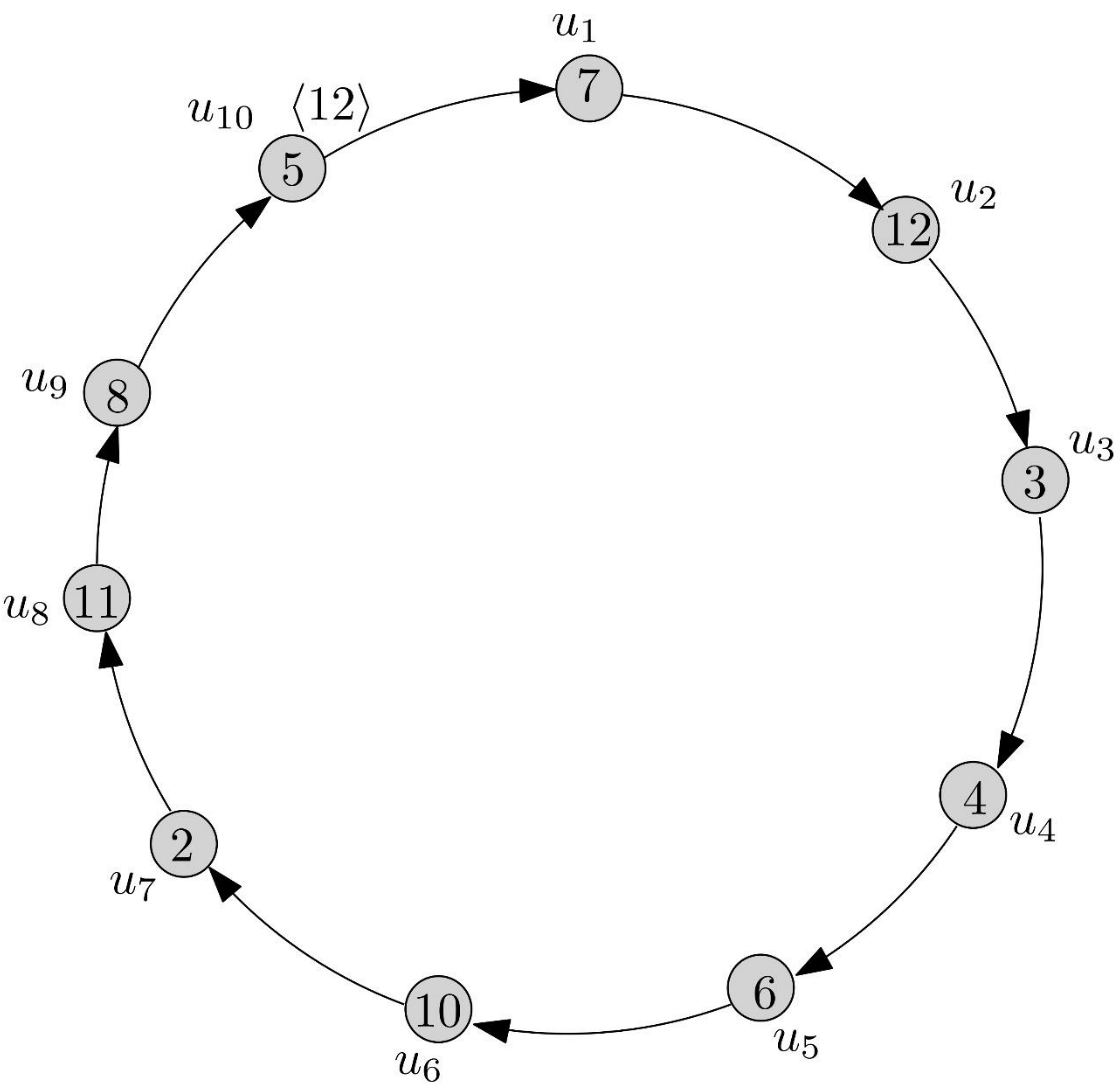
$round = 8$

Example Execution



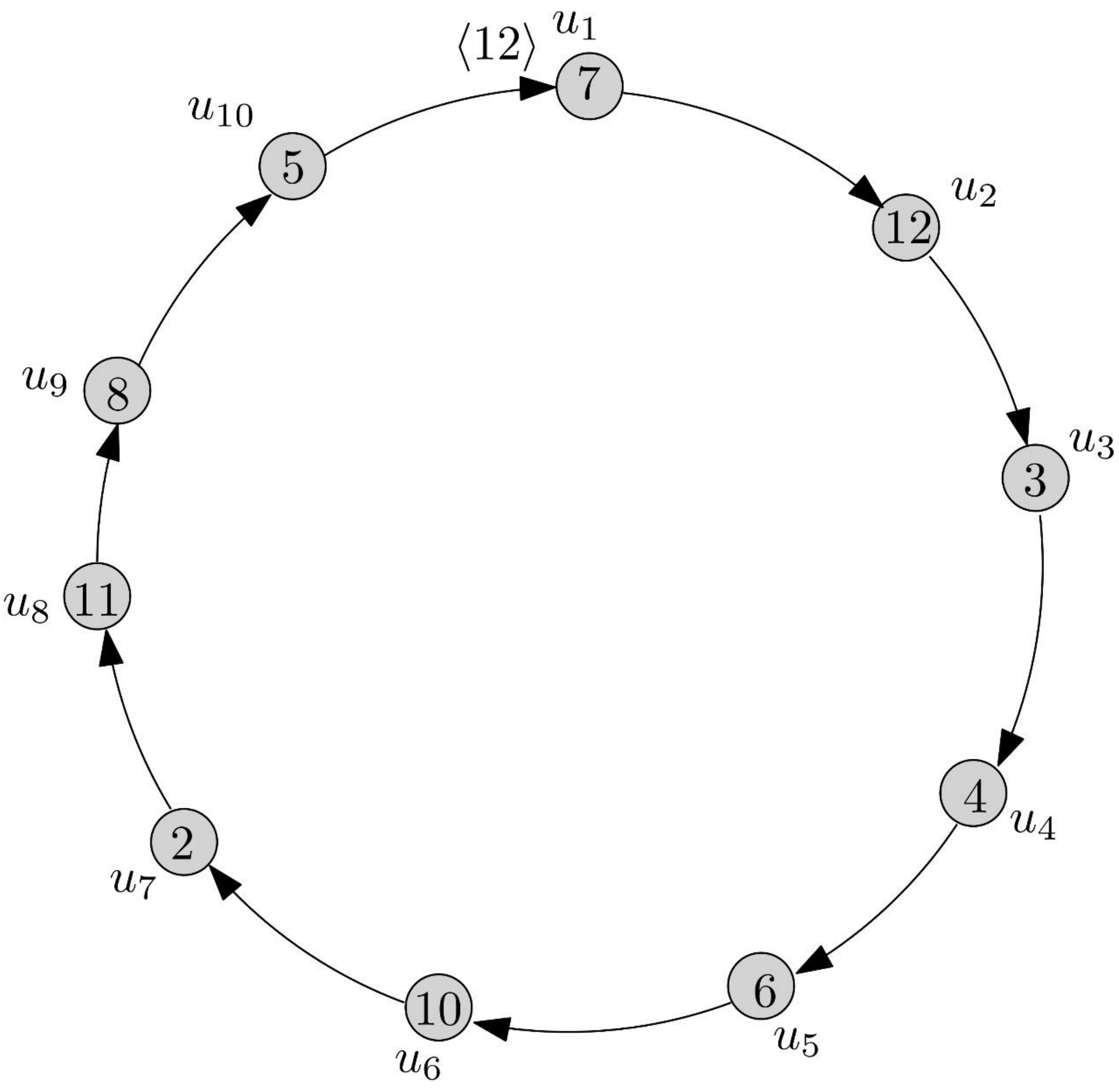
$round = 8$

Example Execution



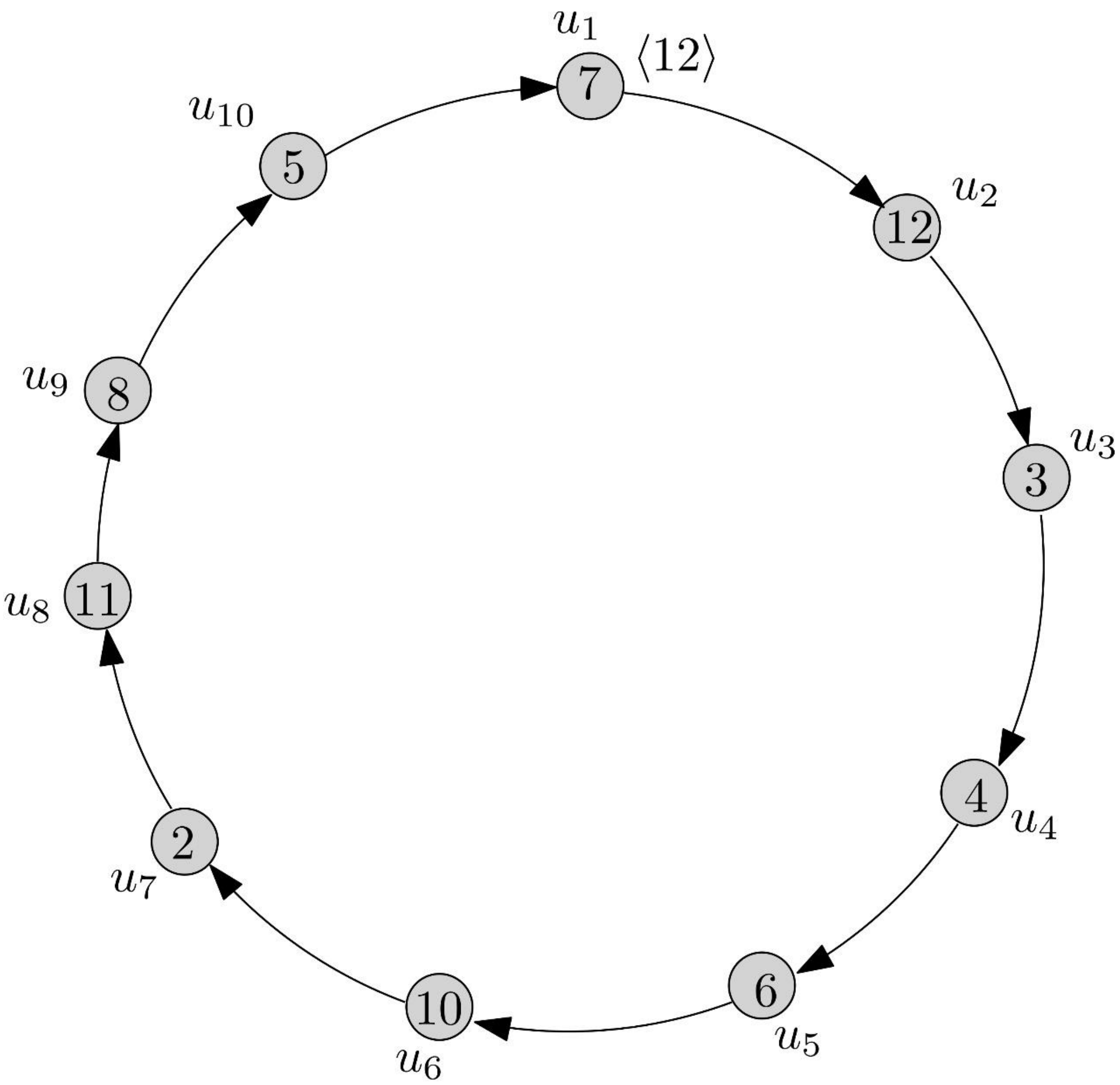
$round = 9$

Example Execution



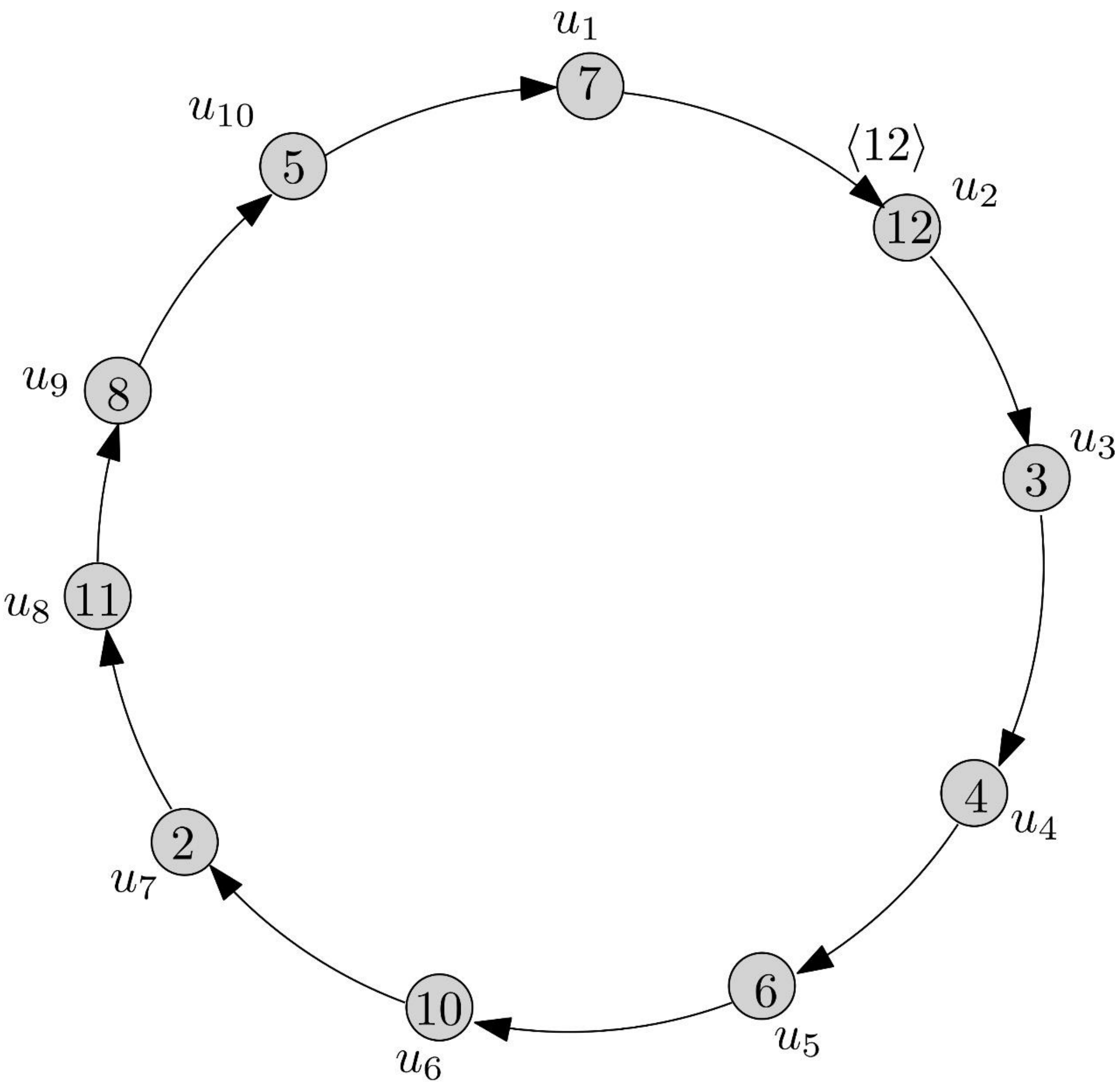
$$round = 9$$

Example Execution



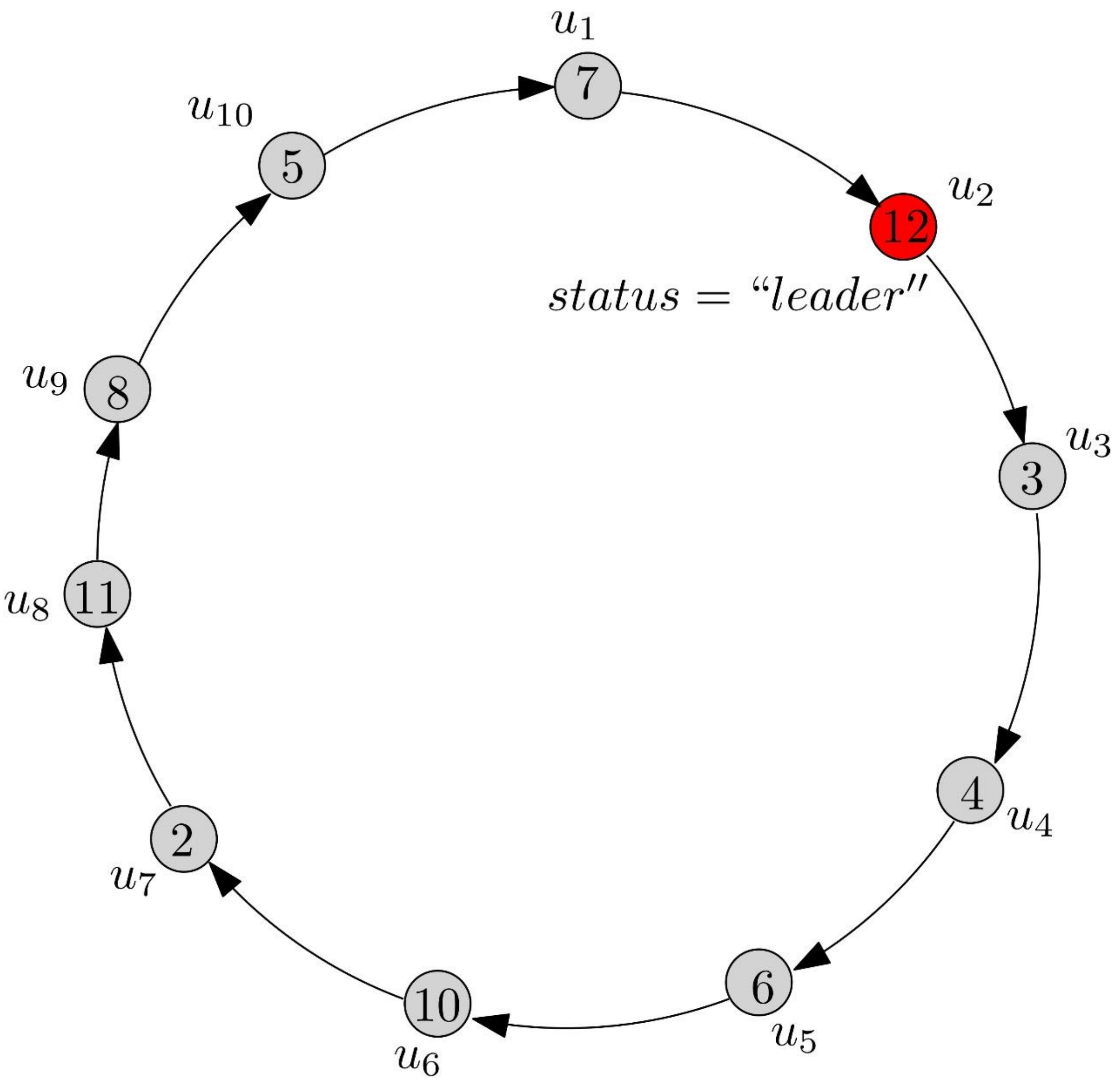
$round = 10$

Example Execution



$round = 10$

Example Execution



$round = 11$

Correctness and Complexity

- Correctness:
  - some processor is eventually elected
  - never 2 or more processors are elected
- Time complexity:
  - n rounds
- 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

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
  - $O(n^2)$ messages
  - At a small additional cost can be made terminating and inform all processors of the elected one