Distributed Leader Election Algorithms in Synchronous Networks

Mitsou Valia

National Technical University of Athens

School of Applied Mathematics and Physics

Distributed Computing

Distributed computing is decentralised and parallel computing, using two or more computers communicating over a network to accomplish a common task.

The collaborating processes are often identical. One of the central problems is…

Leader Election

Given a network of processes, exactly one process should output the decision that it is the leader.

It is usually required that all non-leader processes are informed of the leader's election.

Networks

- The timing model:
  - Synchronous
  - Asynchronous
  - Partially synchronous
- The failure model:
  - Completely reliable
  - Partly faulty
    - Stopping failure
    - Byzantine failure

The Synchronous Network Model

- Directed Graph G(V,E), |V|=n
- Nodes represent processes
- Edges represent (directed) communication channels

The Synchronous Network Model (formal)

- Alphabet M (null indicates the absence of a message)
- On every i Є V we have a process which consists:
  - $states_i$ (a not necessarily finite set of states)
  - $start_i$ (the initial state)
  - $msgs_i$ (a message generation function)

$$msgs_i : states_i \times out-nbrs_i \rightarrow M \bigcup \{null\}$$

  - $trans_i$ (a state transition function)

$$trans_i : states_i \times \underbrace{M \bigcup \{null\} \times M \bigcup \{null\} \times \ldots \times M \bigcup \{null\}}_{in-nbrs_i} \rightarrow states_i$$

- With each edge i, j there is a link that can hold at most a single message in M.

Complexity measures

- Time complexity: the number of the rounds until all outputs are produced or all the processes halt.
- Communication complexity: the number of non-null messages that are sent during the execution.

Leader Election in a Synchronous Ring

Setting

- The network graph is a directed ring (unidirected or bi-directed) consisting of n nodes (n may be unknown to the processes).
- Processes run the same deterministic algorithm
- The only piece of information supplied to the processes is a unique integer identifier (UID).
- UIDs may be used
  - In comparisons only (comparison-based algorithms)
  - In comparisons and other calculations (non-comparison-based).

Related Work and Important Results

Algorithm	Time Complexity	Msg Complexity	Restrictions
LCR ('79)	O(n)	$O(n^2)$	-
HS ('80)	O(n)	O(nlogn)	Bidirectional
ML ('06)	O(n)	O(nlogn) (better constant)	-
TimeSlice	$O(n \cdot u_{min})$	O(n)	Non-comparison based
Lower bound	Ω(n) (trivial)	Ω(nlogn) FL ('87)	

The LCR Algorithm

- Comparison-based Algorithm
- The size of the ring is unknown to the processes
- Unidirectional Ring
- It elects the process with the maximum UID

The LCR Algorithm

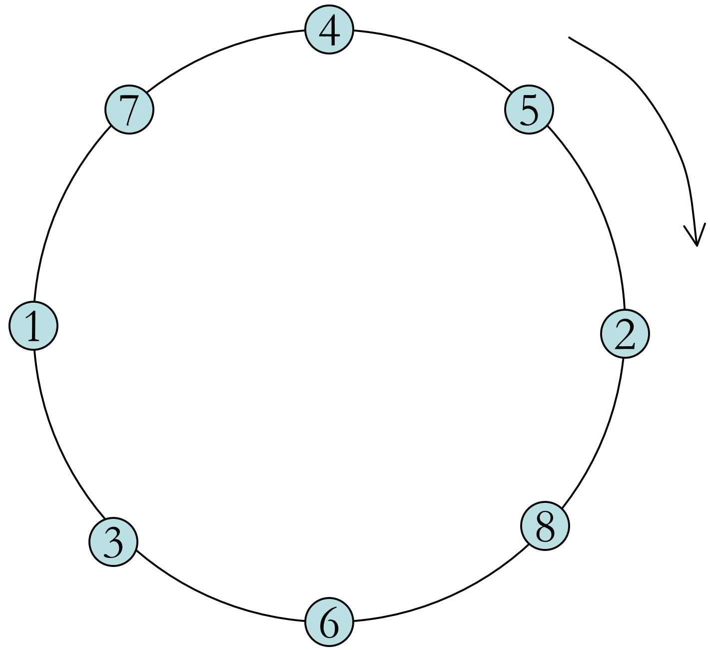
Description

Each process sends its UID around the ring.

When a process receives a UID, it compares this one to its own.

- If the incoming UID is greater, then it passes this UID to the next process.
- If the incoming UID is smaller, then it discards it.
- If it is equal, then the process declares itself the leader.

LCR Example



LCR Complexity Analysis

- Time Complexity: $O(n)$
- Message Complexity: $O(n^2)$ – worst case
  $O(n\log n)$ – average case

The HS Algorithm

- Comparison-based Algorithm
- The size of the ring is unknown to the processes
- Bi-directional Ring
- It elects the process with the maximum UID

The HS Algorithm

Description

Each process operates in phases 0, 1, 2...

In each phase k, process i sends tokens with its UID in both directions to travel distance 2^k and return back to it.

If both tokens return then process i continues in phase k+1.

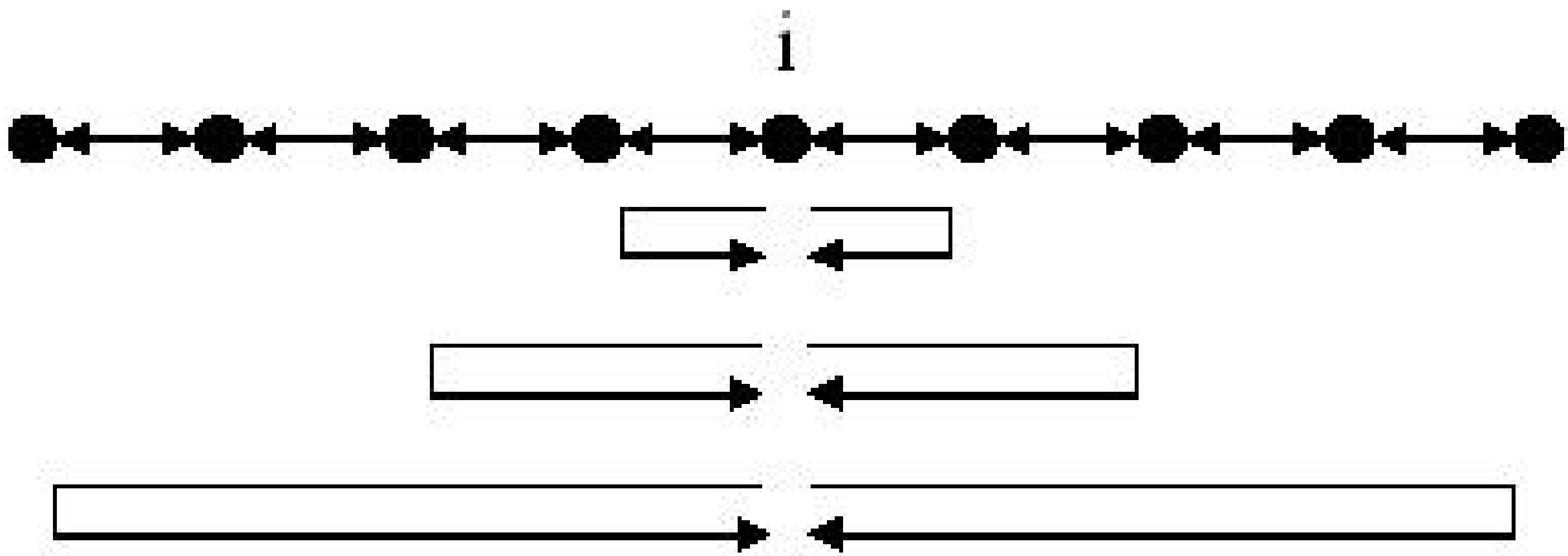


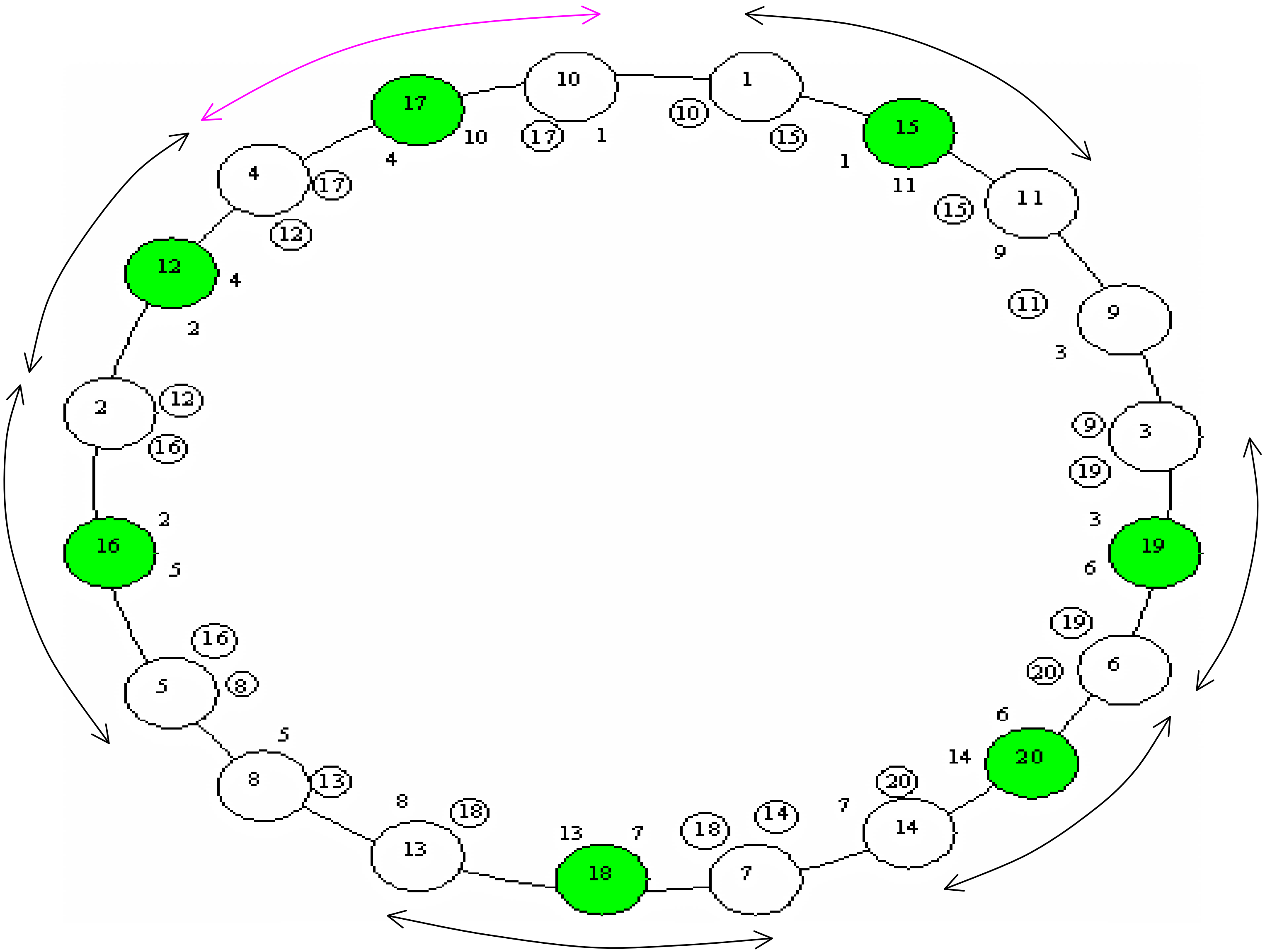
Figure: The execution of the HS

The HS Algorithm (continued)

When a process receives an outgoing UID, it compares this one with its own.

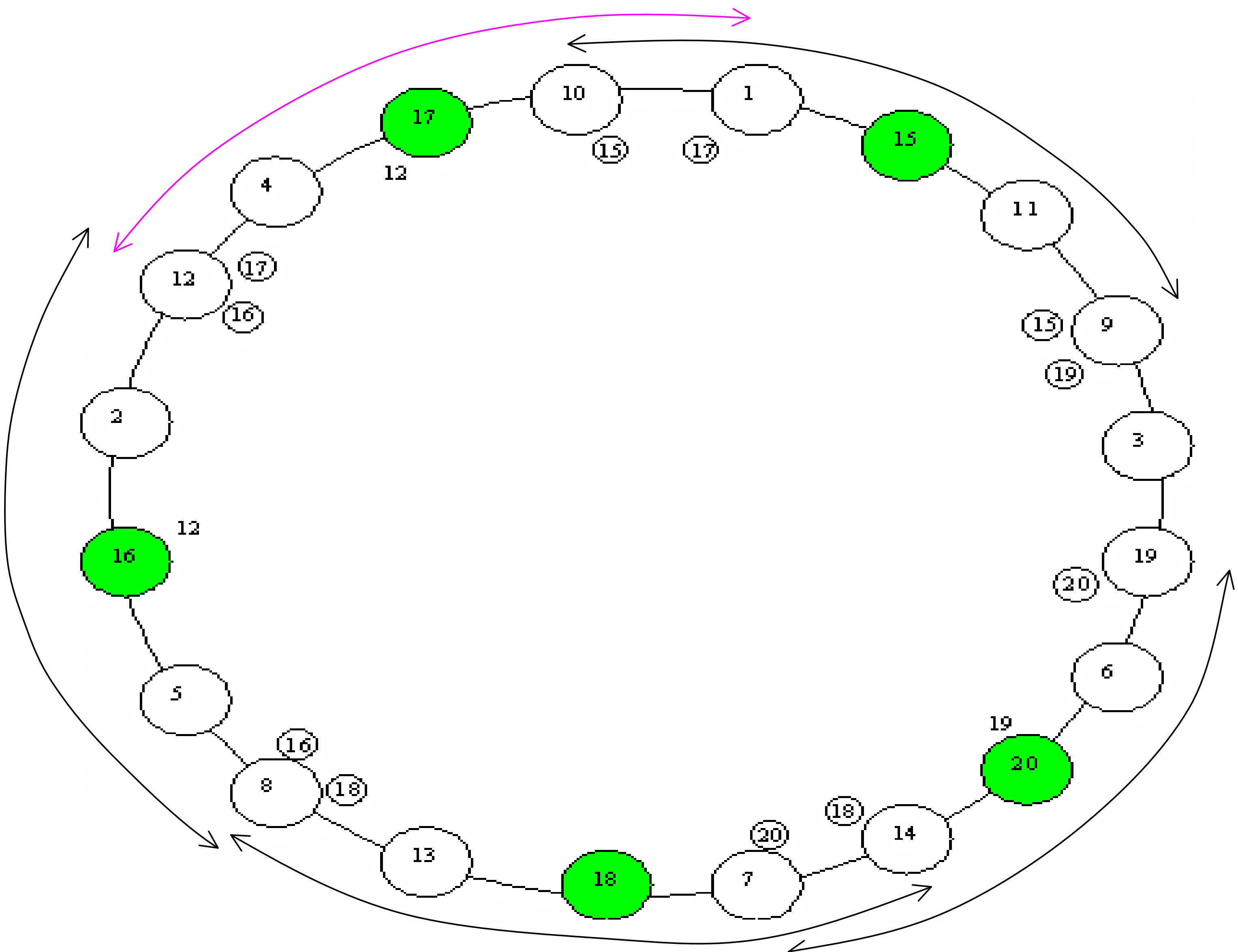
- If the received UID is smaller, then it discards it.
- If the received UID is greater then
  - it passes it to the next process, if it is not the end of its path,
  - else it returns it back to the previous one (to travel back to the originating process).
- If it is equal, then the process declares itself the leader.

HS Example (phase 0)



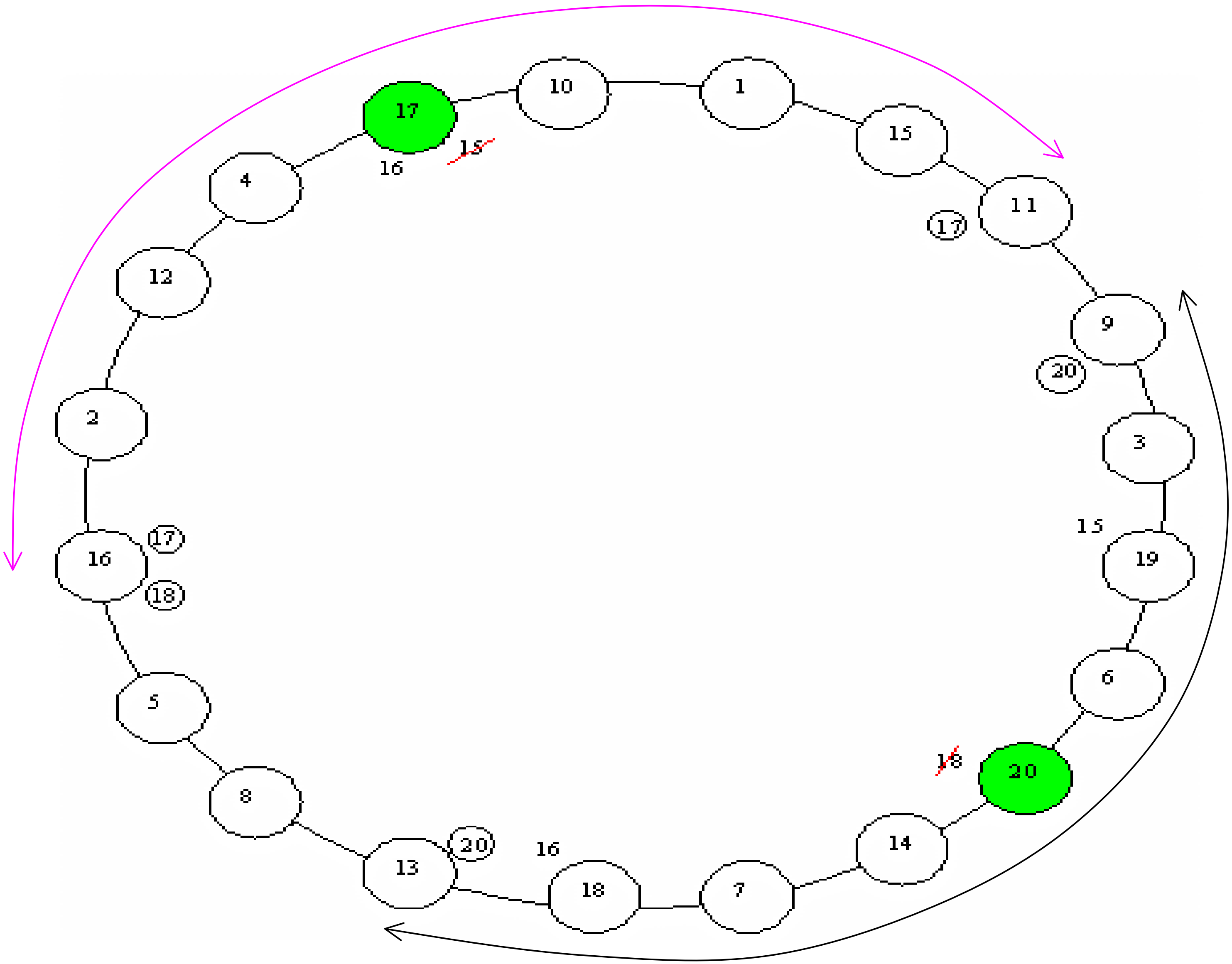
All the 20 nodes

HS Example (phase 1)



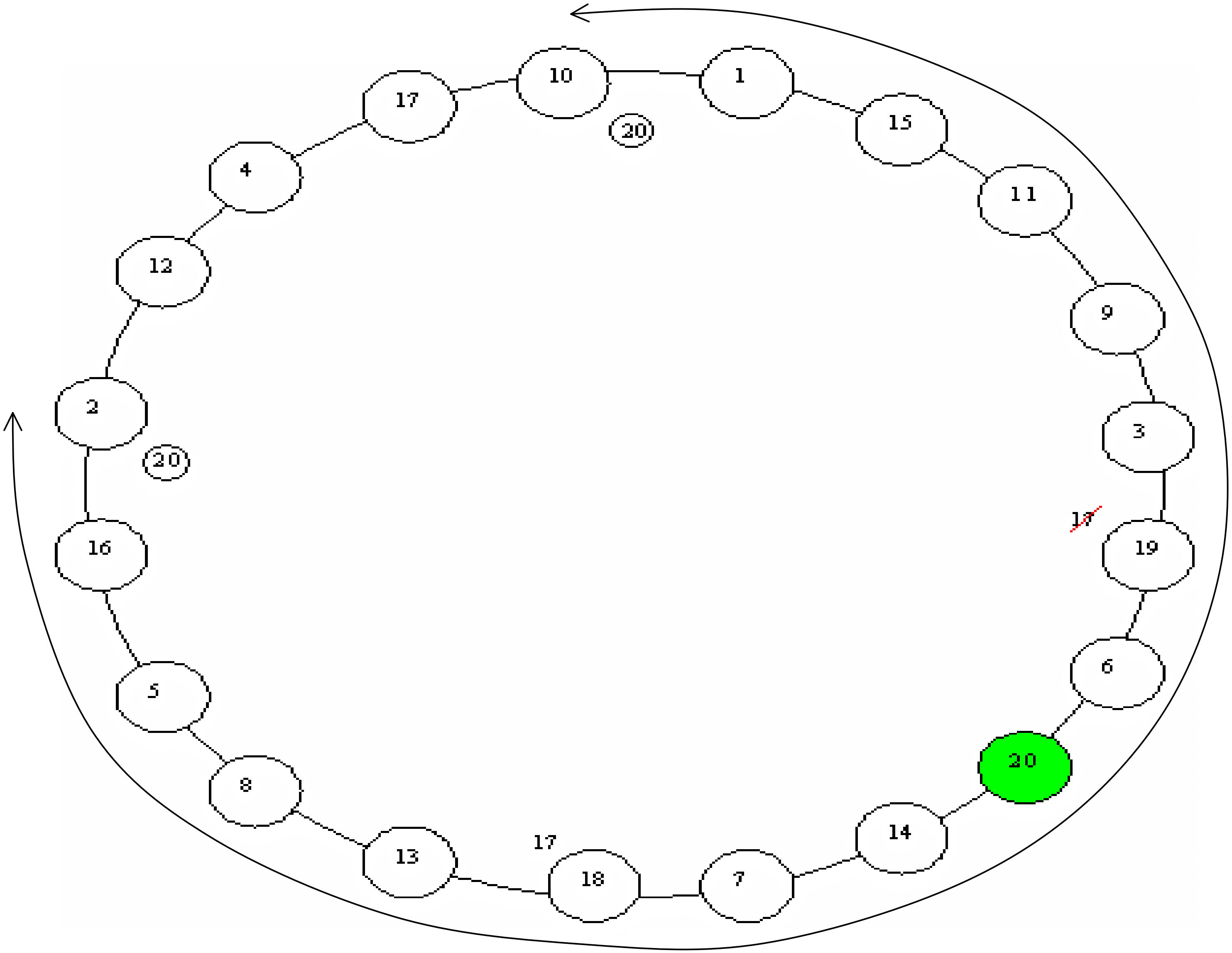
- 15
- 19
- 20
- 18
- 16
- 12
- 17

HS Example (phase 2)



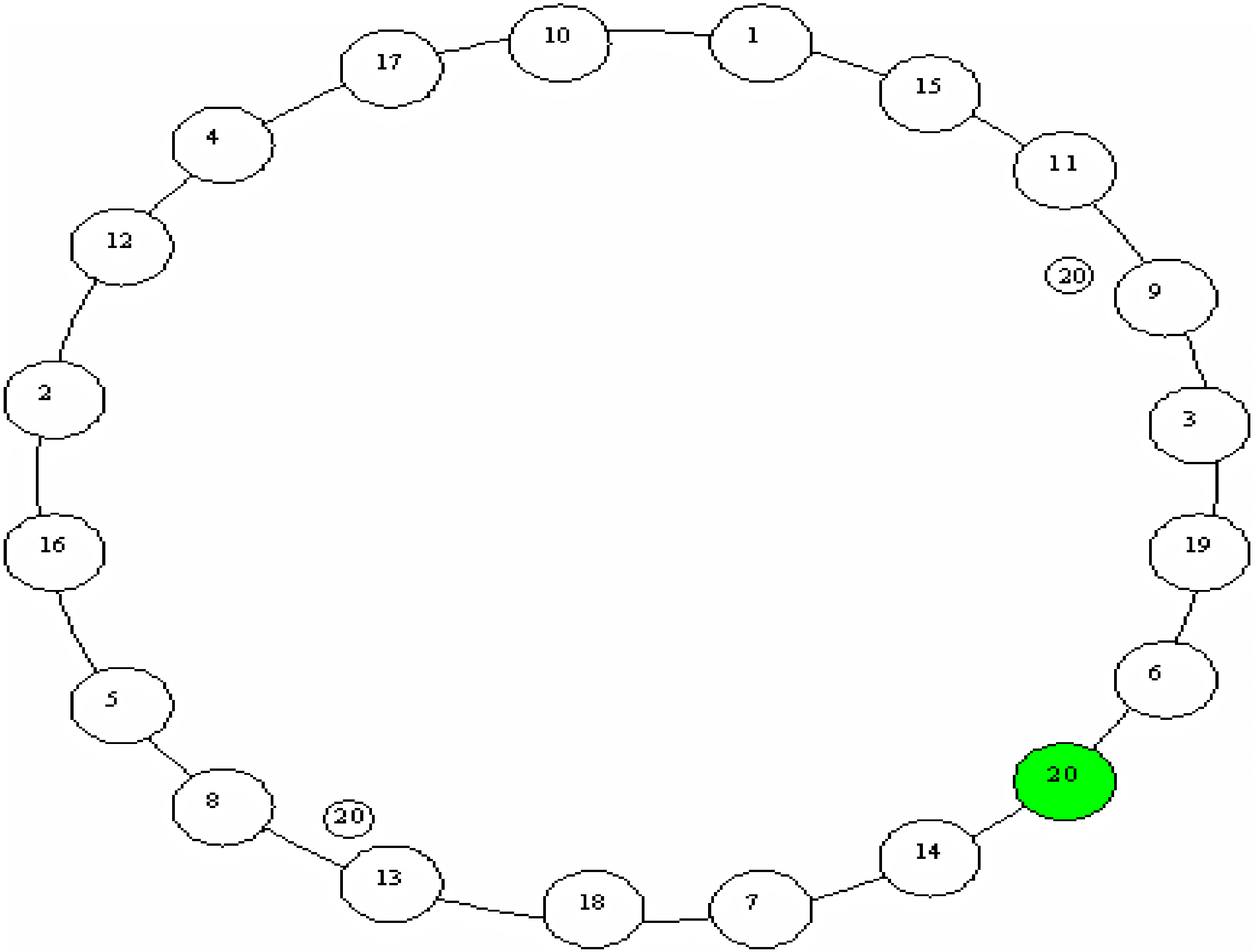
- 15
- 20
- 18
- 16
- 17

HS Example (phase 3)



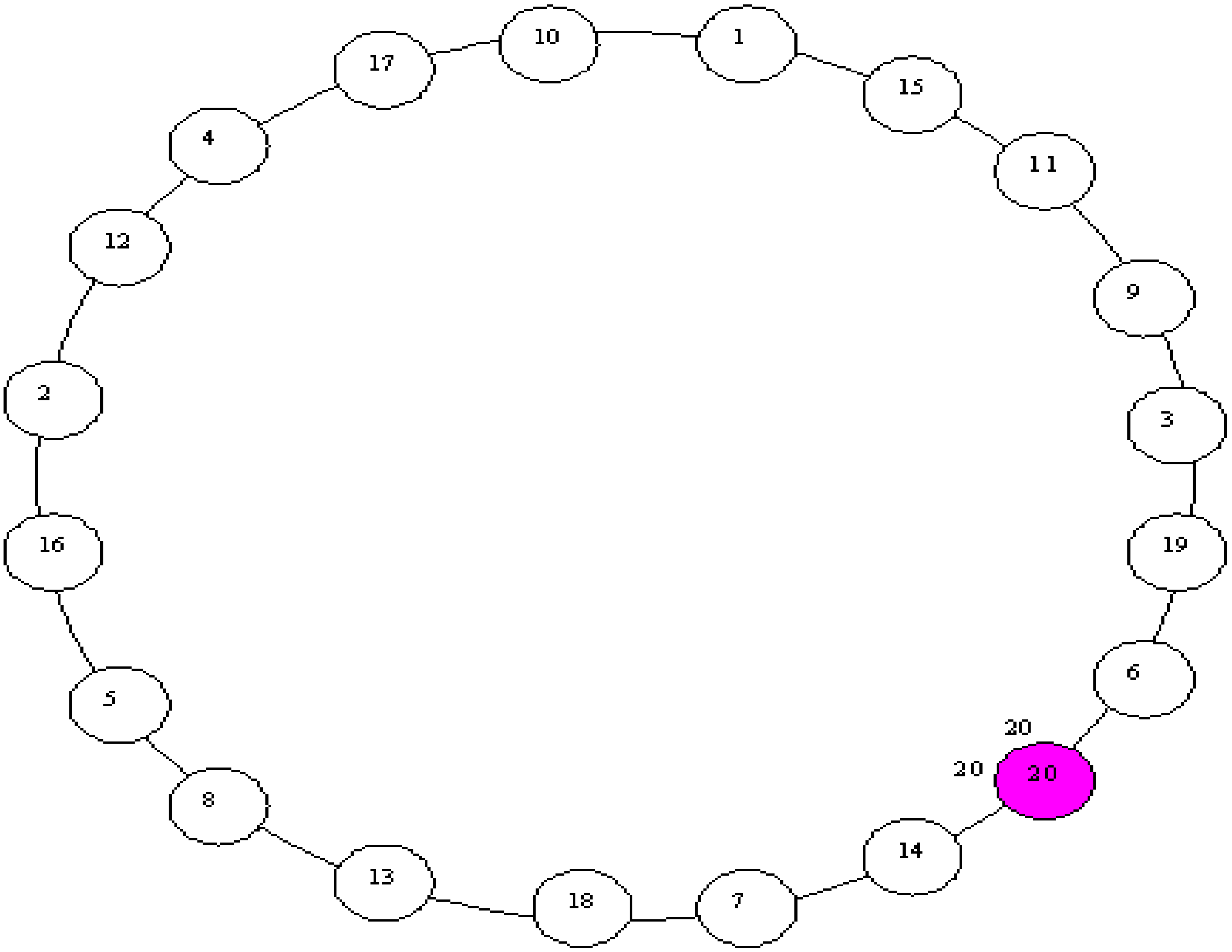
- 20
- 17

HS Example (phase 4)



- 20

HS Example (phase 5)



20 is the LEADER

Complexity Analysis

- Time Complexity: $O(n)$
- Message Complexity: $O(n\log n)$

Distributed Algorithms in a General Synchronous Network

Leader Election in a General Network - The FloodMax Algorithm

- The diam of the graph is known.
- Causes both leader and non-leaders to identify themselves.
- It elects the process with the maximum UID.

FloodMax Algorithm

- Every process keeps the maximum UID it has seen so far (initially its own).
- At each round, each process sends this maximum value to every outgoing neighbor.
- After diam rounds if the maximum value is the process's UID then it elects itself the leader, otherwise it is a non-leader.

Complexity Analysis

- Time Complexity: diam rounds
- Communication Complexity: $\text{diam}\cdot|E|$
  ($|E|$ messages in every round).

Minimum Spanning Tree

Spanning tree of a graph G(V,E): a tree that consists entirely of edges in E and contains every vertex of G.

The problem: Given an undirected graph G(V,E) find a minimum weight (undirected) spanning tree for the network.

Distributed output: Each process should determine which of its incident edges belong to the tree.

- Processes know n
- Processes have UIDs

Minimum Spanning Tree (continued)

General Strategy for MST:

- Start with the trivial spanning forest.
- For every connected component C select a minimum weight outgoing edge e.
- Combine C with the component at the other end of e, including e.
- Stop when the forest has a single component.

Minimum Spanning Tree (continued)

Several well-known sequential MST algorithms are special cases of this general strategy:

- Prim (add minimum-weight outgoing edge from the current component attaching a new single node)
- Kruskal (add minimum-weight edge that joins two separated parts)

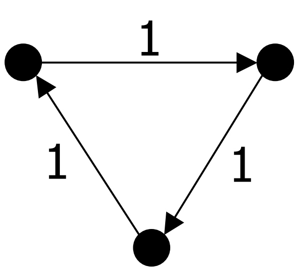
Minimum Spanning Tree (continued)

A distributed version could be:

Each component determines a minimum-weight outgoing edge and all these edges are added to the forest causing combinations of components all at once.

The above strategy is false in general!!!

Example: A cycle could be created.



Lemma: If all edges of G have distinct weights, then there is exactly one MST.

Minimum Spanning Tree (continued)

The SynchGHS algorithm

(Based on an asynchronous algorithm developed by Gallager, Humblet and Spira in 1983.)

The strategy mentioned before is used.

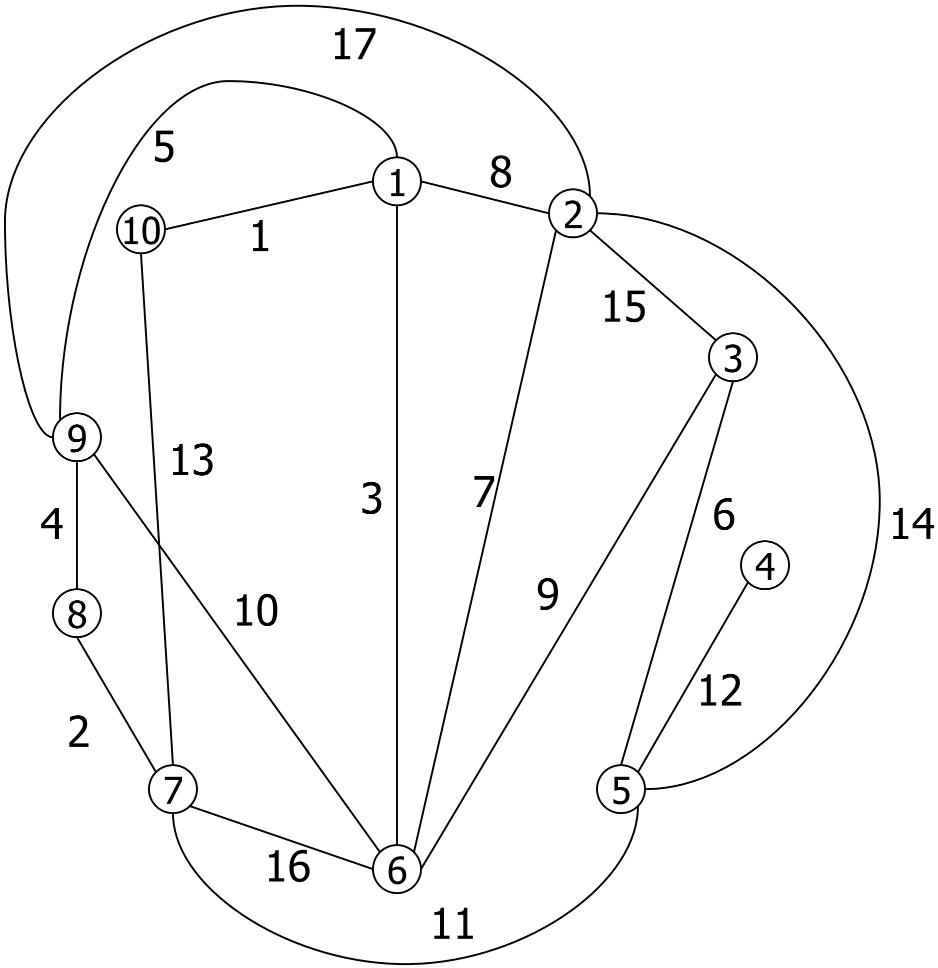
Assumption: Edge weights are all distinct.

Minimum Spanning Tree (continued)

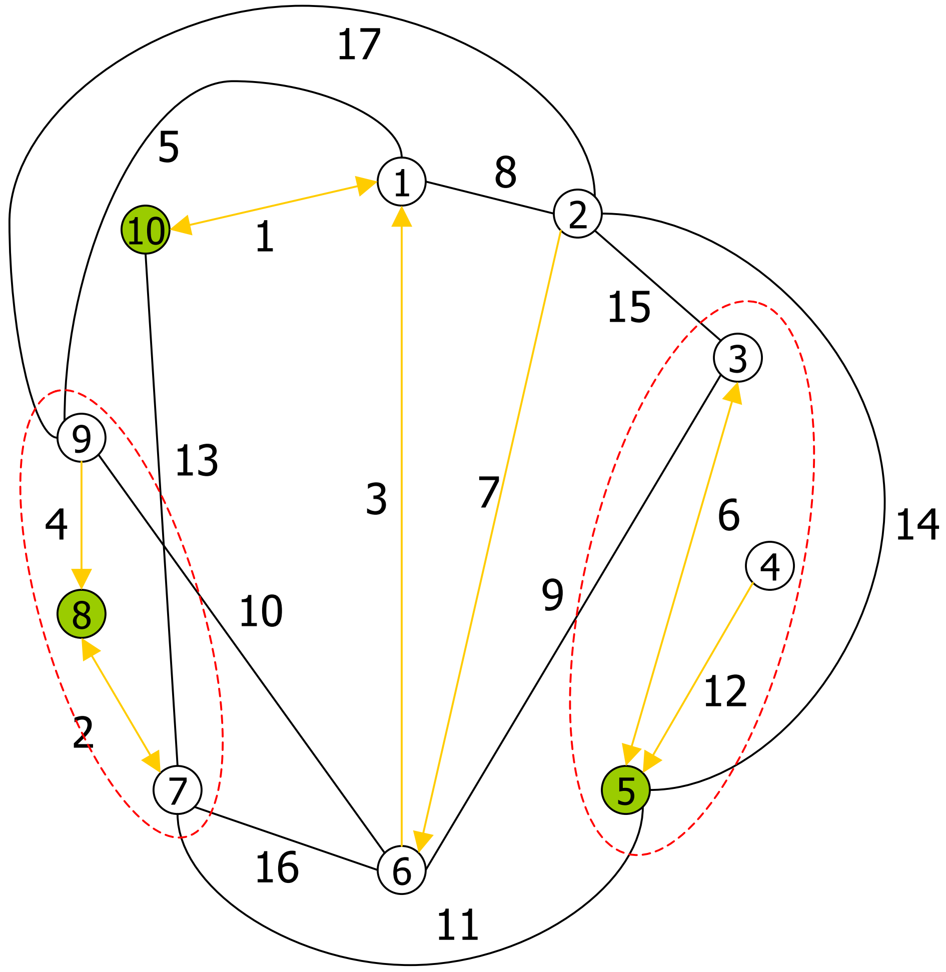
<u>The Algorithm:</u>

- The algorithm builds components in levels.
- For each level k, the level k components are subtrees of the MST that constitute a spanning forest.
- Each level k component has at least 2^k nodes.
- Every component at every level has a distinguished leader node.

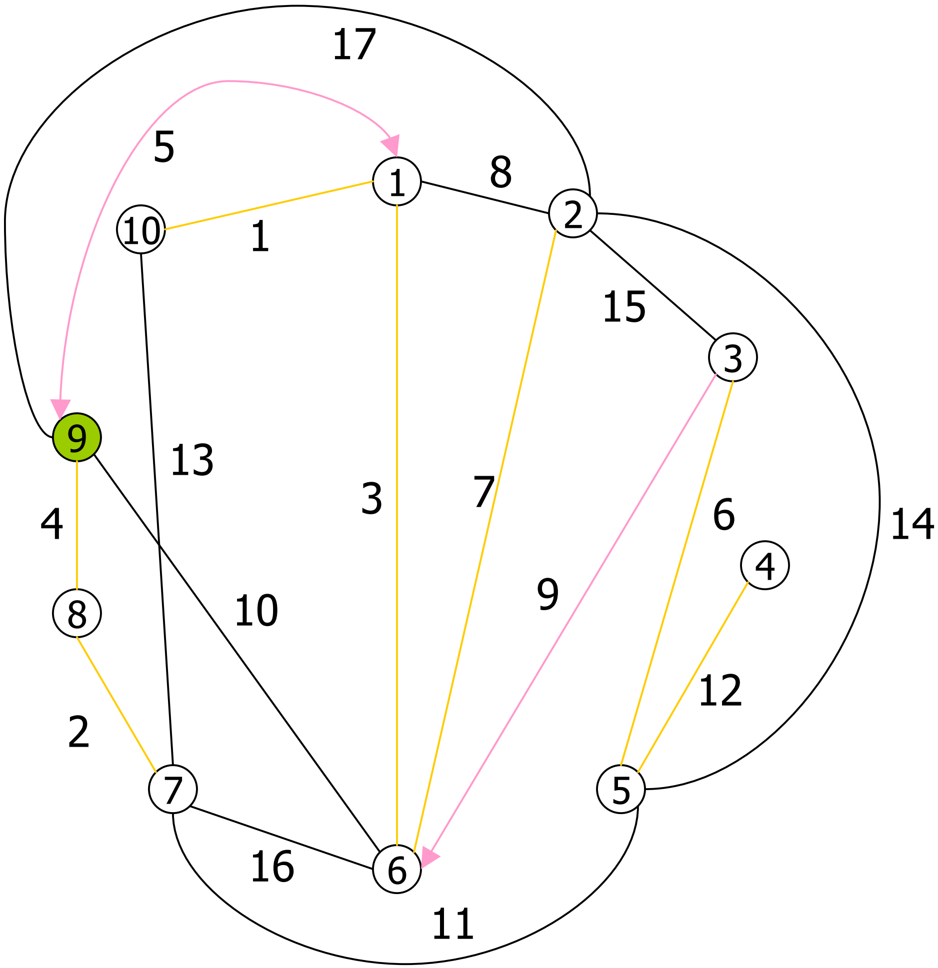
Minimum Spanning Tree (continued)



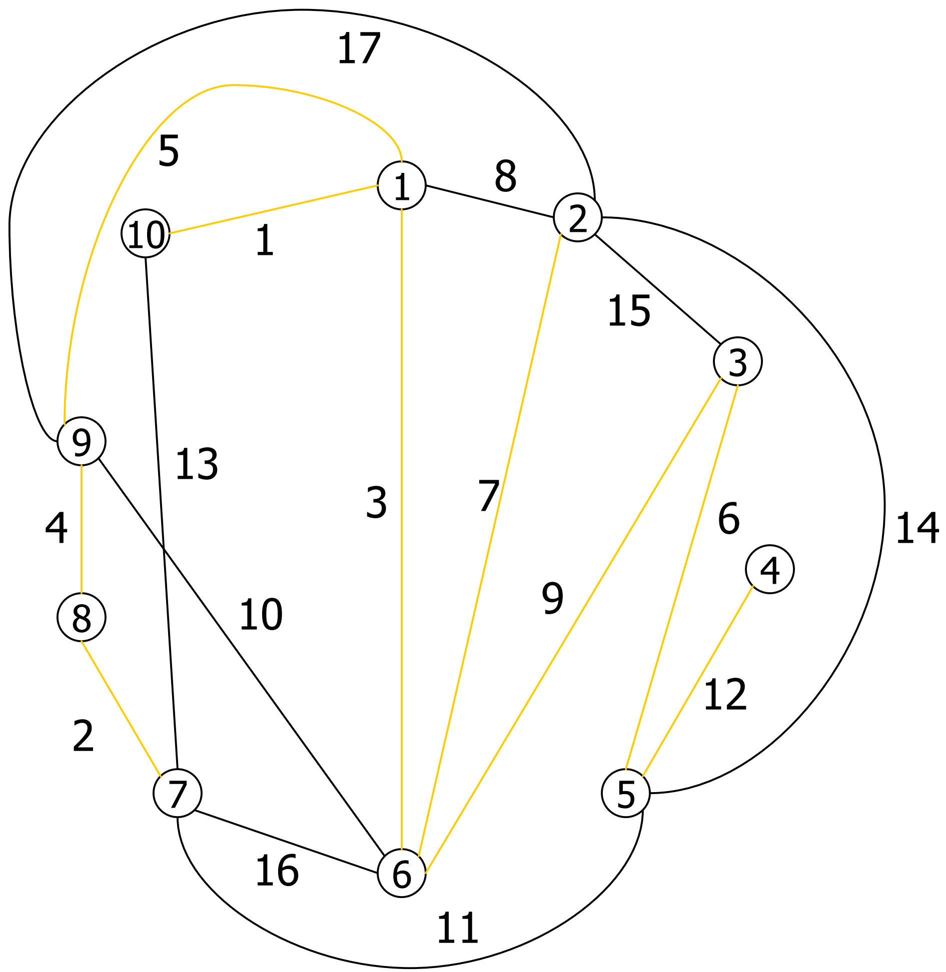
Minimum Spanning Tree (continued)



Minimum Spanning Tree (continued)



Minimum Spanning Tree (continued)



Minimum Spanning Tree (continued)

Complexity Analysis

- Time Complexity: O(nlogn)

  [logn levels] x [O(n) time for every level for synchronization].

- Communication Complexity: O((n+|E|)logn)

  [logn levels] x [O(n) messages along tree edges + O(|E|) messages for finding the local minimum weight outgoing edges].

→ It can be reduced to O(nlogn + |E|).

Minimum Spanning Tree (continued)

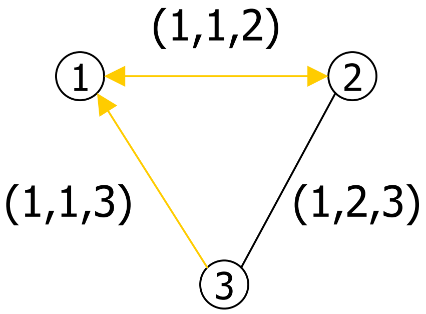
Non-unique weight edges:

edge identifier: a triple ($weight_{i,j}$, u, u')

where, u<u' the UIDs of i, j.

Thus, a total ordering is defined among the edge identifiers.

Example:



Minimum Spanning Tree (continued)

Leader Election:

- The leaves of the MST begin a convergecast along the paths of the tree.
- Internal nodes wait to receive messages from all but one neighbor. Then they send a message to the remaining neighbor.
- If a node receives messages from every neighbor without having itself send a message then becomes the leader.
- If two neighboring nodes receive messages from each other at the same round, then the one with the greatest UID becomes the leader.

Complexity: n-1 additional time and messages.

Leader Election in Anonymous Rings

General

Lemma: If the network is symmetric (i.e. a ring) and anonymous (the processes haven't UIDs) then it is impossible to elect a leader by a deterministic algorithm. [by Angluin (1980)]

Probabilistic algorithms are used to break symmetry.

Itai and Rodeh Algorithm

Assumption: Processes know n.

The Algorithm

- The algorithm proceeds in phases, each of them containing n rounds.
- At every phase, $a \leq n$ processes are active (initially everyone). During each phase some processes may become inactive.
- At the beginning of every phase, every active process decides with probability a^{-1} whether or not to become a candidate.

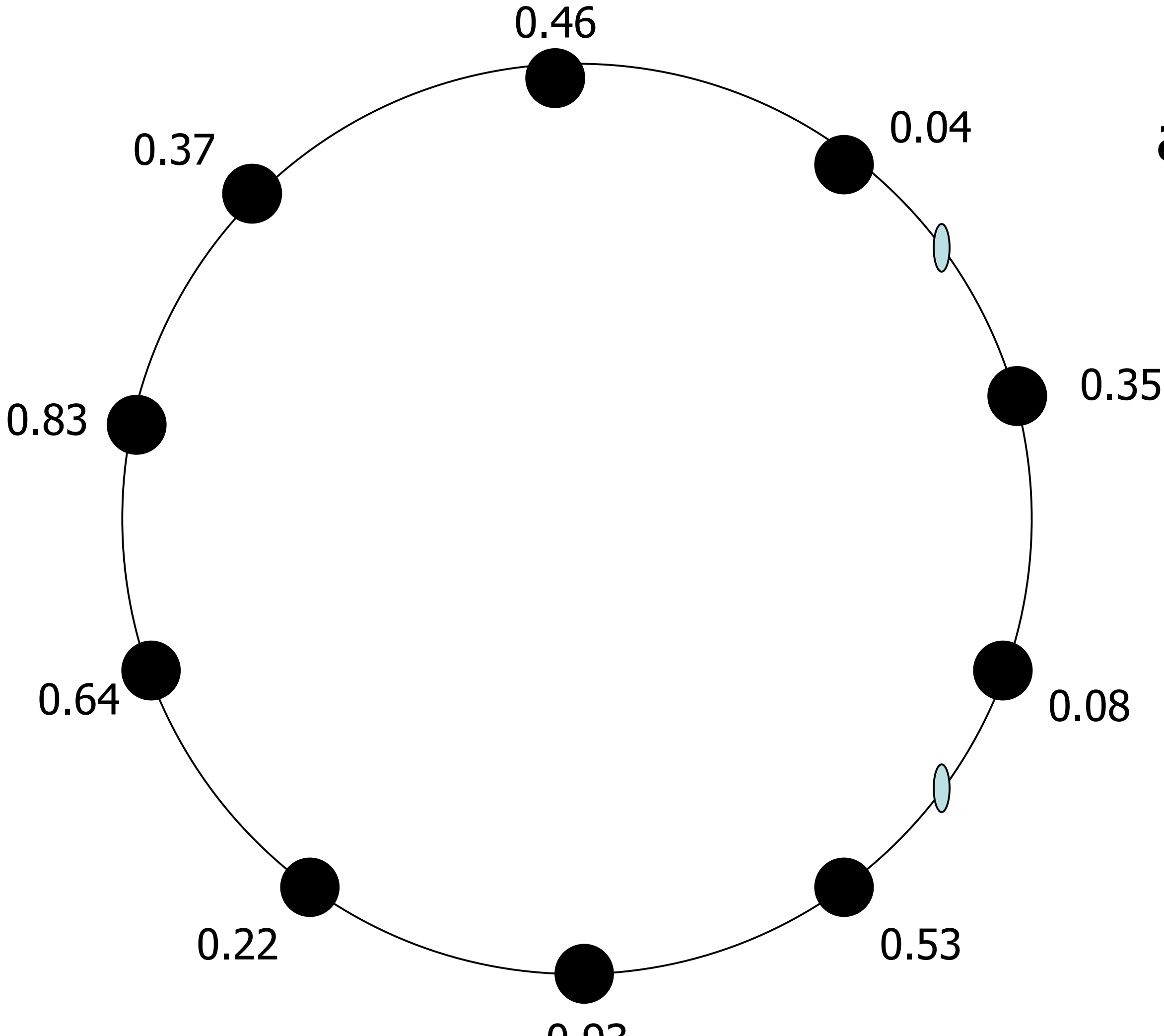
  To do that, it picks a random number r, $0<r<1$ and if $r<a^{-1}$, then it becomes a candidate and initiates a pebble to travel around the ring.

Itai and Rodeh Algorithm

- To compute the number of candidates (c), each process counts the pebbles it has seen.

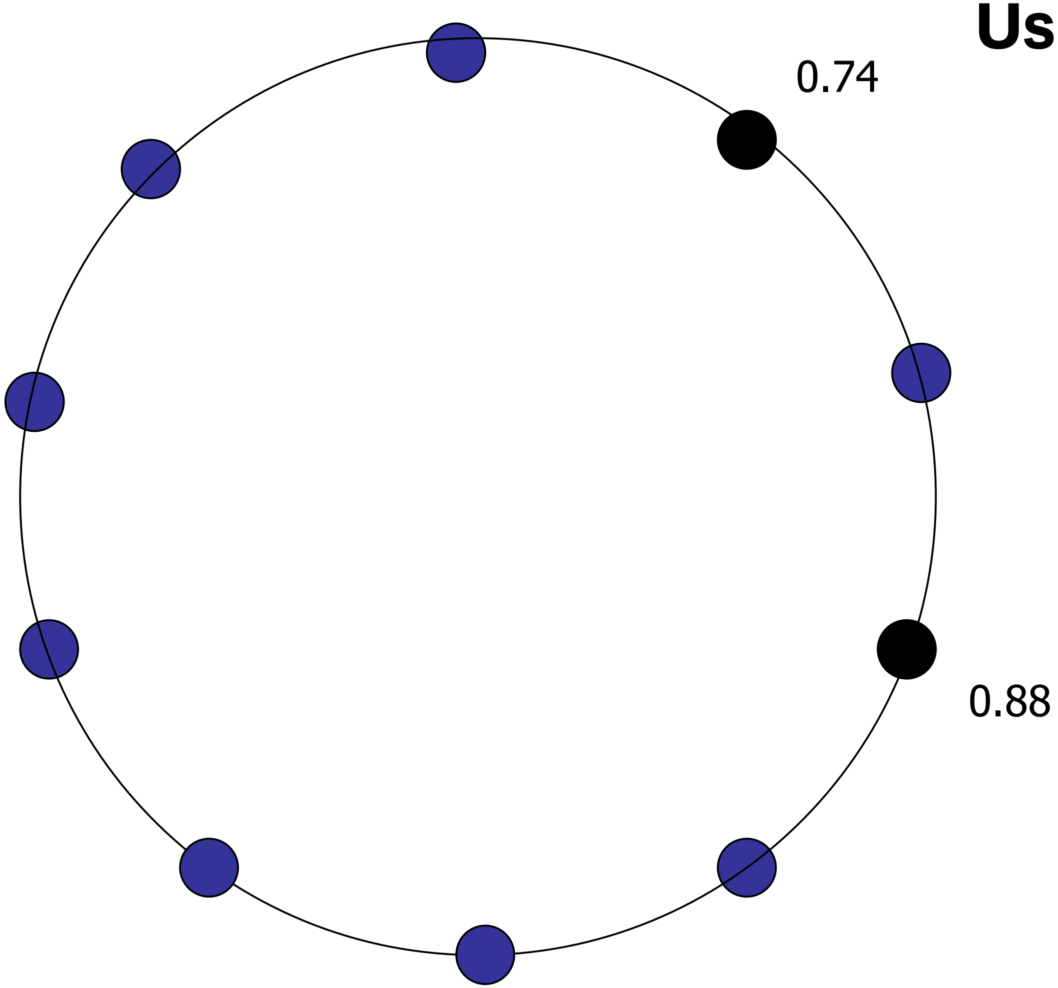
  Number of pebbles counted = Number of candidates.
- At the end of the phase, every process has calculated c.
- If c=1 then sole candidate becomes leader. If c>1 then a new phase begins with the new active processes (the candidates of the previous phase). If c=0 the phase was useless.

Itai and Rodeh Algorithm



$a^{-1} = 1/10$
$c = 2$

Itai and Rodeh Algorithm

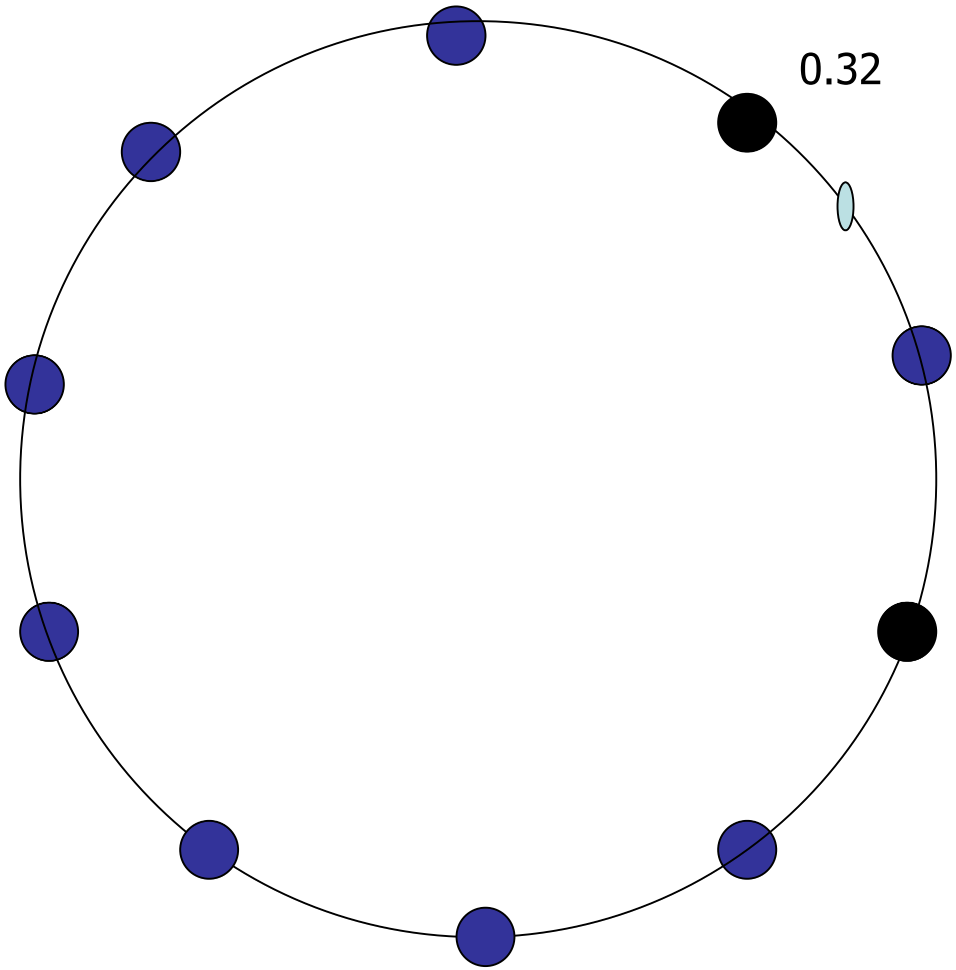


Useless phase

$a^{-1} = 1/2$

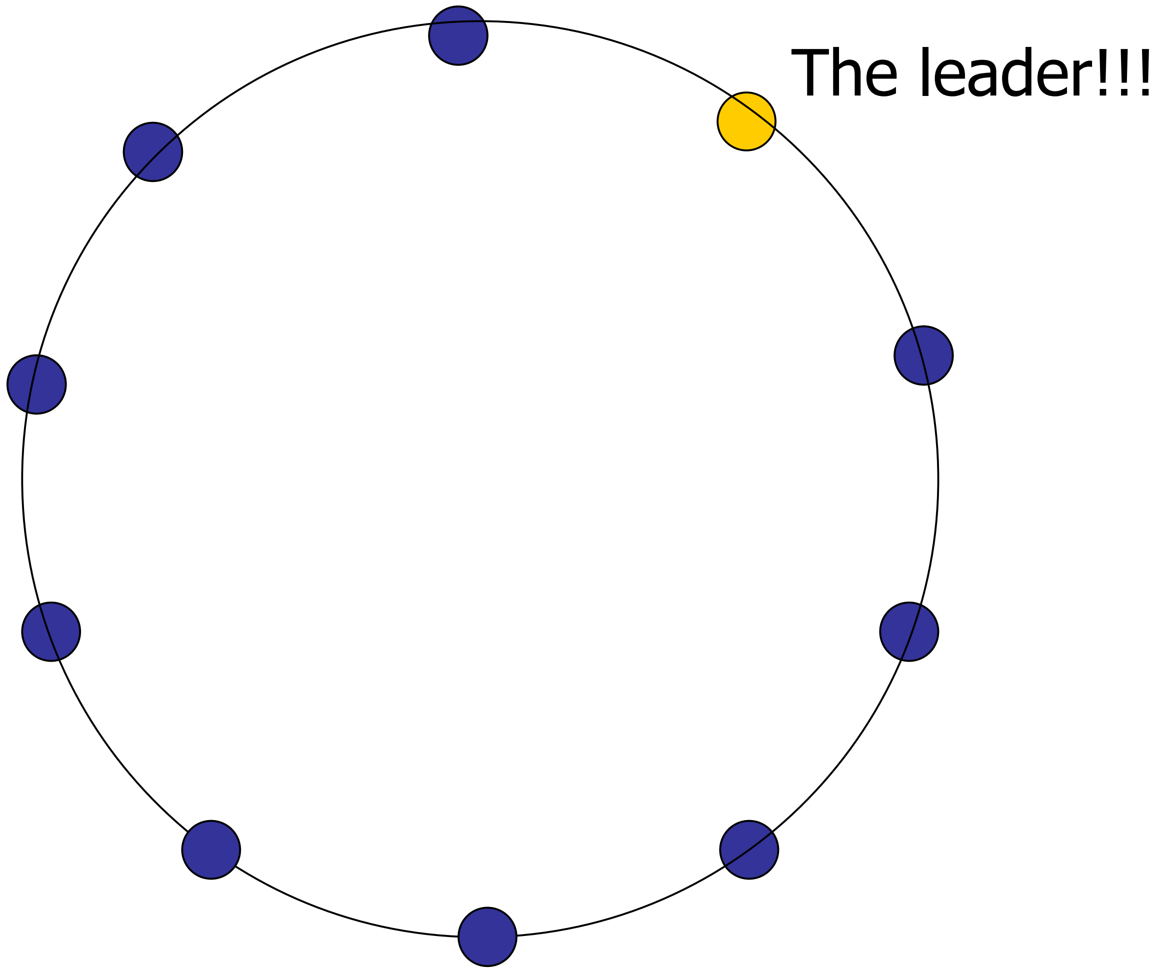
$c = 2$

Itai and Rodeh Algorithm



$a^{-1} = 1/2$
$c = 1$

Itai and Rodeh Algorithm



Itai and Rodeh Algorithm- Complexity Analysis

p(a,c) : the probability that c out of a active processes become candidates. Then

$$p(a,c) = \binom{a}{c} a^{-c} \left(1 - \frac{1}{a}\right)^{a-c}$$

Proof:

X_i a random variable, $X_i = \begin{cases} 1, & a^{-1} \\ 0, & 1 - a^{-1} \end{cases}$

X_i=1 if i becomes a candidate, else 0 (bernoulli trial)

Then $X=\Sigma^n X_i$= the number of processes become candidates. X~binomial distribution.

Thus $P[X = c] = \binom{a}{c} a^{-c} \left(1 - a^{-1}\right)^{a-c} = p(a,c)$

Itai and Rodeh Algorithm- Complexity Analysis

Average Case

- Time Complexity: $2.441716 \cdot n$
- Message Complexity: $2.441716 \cdot n$

The number of pebbles initialized per phase is X (the number of active processes that become candidates).
$E[X] = E[\Sigma^a X_i] = \Sigma^a(E[X_i]) = a \cdot a^{-1} = 1$
Thus, the expected message complexity per phase is n.

Leader Election Protocols with Cheating Processes

The Model

Full- or Perfect-Information Model [BL 90]**:**

- There is an adversary that controls t players
- The adversary has unlimited computational power.
- Communication between players is by broadcast.
- Reliable delivery of messages.
- The identity of the sender is protected.

The adversary has complete knowledge of the state of the protocol at any given moment.

The Network

We assume an asynchronous network with synchronization points :

- Computation proceeds in rounds.
- In each round processes send messages.
- During a round we can't force processes to act simultaneously.
- Messages of round i precede those of round i+1.

Within a round, all cheaters have the opportunity to wait until they receive messages from all honest players and then send their own.

Example

A Leader Election Protocol of n processes (Baton Passing [Saks 89]):

- In every round the baton is randomly passed to a process that hasn't yet received it.
- The last process left with the baton becomes the leader.

$$P(i) = \frac{n-1}{n} \cdot \frac{n-2}{n-1} \cdots \frac{2}{3} \cdot \frac{1}{2} = \frac{1}{n}$$

If there are cheaters, when they take the baton they give it to an honest process, in order to increase the probability of a cheater to be elected.

Baton Passing lasts n-2 rounds.

Failure probability

Let *P(n,t)* be a leader election protocol between n processes, t of which are corrupted.

<u>$fail_P(n,t)$:</u> the probability that one of the cheaters is elected.

Proposition: For any n, any $t \geq 1$ and any leader election protocol *P* :

1. $fail_P(n,t)$ is non-decreasing in t.
2. $fail_P(n,t) \geq t/n$
3. $fail_P(n, \lceil n/2 \rceil) = 1$

Resilience

<u>Resilience</u>: How many cheaters are allowed in order for the protocol to guarantee that an honest player can be elected with positive probability.

Definition: P is resilient for t=b(n) iff $\exists \varepsilon > 0$ such that for all suitably large n

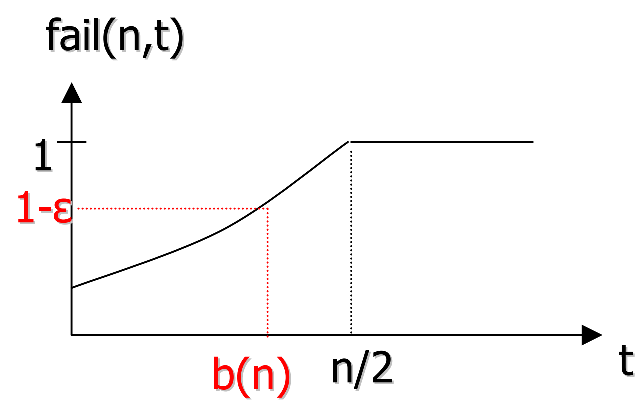
$$\text{fail}_P(n, b(n)) \leq 1 - \varepsilon.$$

However,

If $t \geq 1$ then $\text{fail}_P(n,t) > 1/4$

(P is the Lightest Bin protocol

which achieves optimal resilience)



Cheaters' Edge

<u>Cheaters' edge</u>: The factor that $fail_P(n,t)$ increases by cheating. (Antonakopoulos 2006)

Definition: $edge_P(n,t) = n/t \cdot fail_P(n,t) - 1 \ (\geq 0)$

Example: Assume a fair leader election protocol

If edge = 1 $\Rightarrow n/t \cdot fail_P(n,t) - 1 = 1 \Rightarrow$

$\Rightarrow n/t \cdot fail_P(n,t) = 2 \Rightarrow$

$\Rightarrow fail_P(n,t) = 2 \cdot t/n$

Zero Edge Protocols

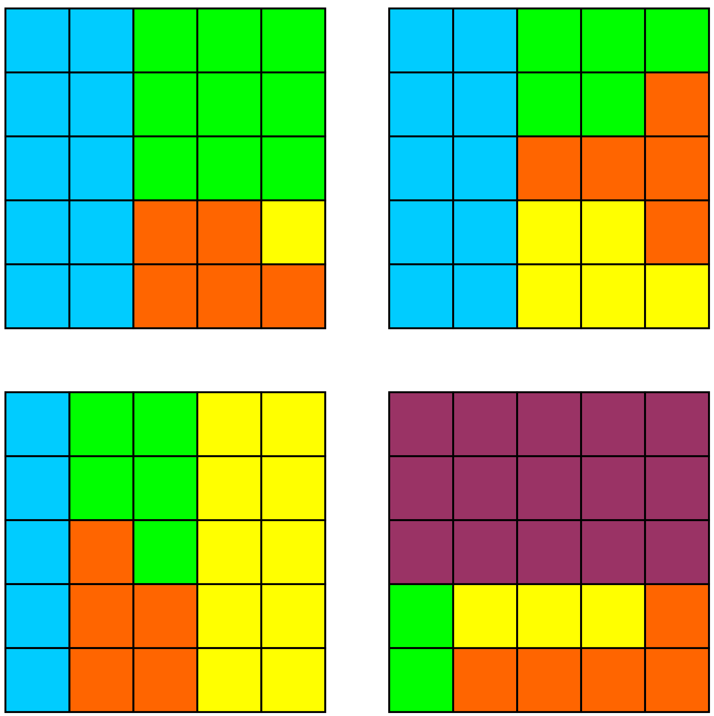
Zero Edge Protocols: Protocols where cheaters cannot increase their probability of election by cheating.

- These protocols exist only for t=1.
- For t>1 the adversary can find two players that can collude.

Example: Baton Passing (t=1)

Counter Example: Itai - Rodeh

Zero Edge Protocols



A picks a row

B picks a column

C picks a level

D, E are mute.

The selected square determines the leader.

A cannot increase the probability of his election. Same for B and C .

Related Work and Important Results

Probabilistic arguments have established that:

1. There exists a leader election protocol AN1 with bounded cheaters' edge for all $t \leq n$.
   [Alon, Naor, '93].
2. For any $\beta < 1/2$, there exists a protocol that is resilient for $t = \beta n$. [AN93, BN00] (In terms of resilience, this is optimal.)

Disadvantages

- Non-constructive. Exhaustive search may be attempted, but could take time $2^{2^{O(n)}}$.
- O(n) running time (very slow)

Related Work and Important Results - Reduction via Committees

Lemma: From a leader election protocol $P(n,t)$ executed in $r(n)$ rounds and constructed in $s(n)$ time, we can obtain a leader election protocol $cmt|P(\log^d n, (t/n + c/\log n) \log^d n)$, that lasts $r(\log^d n)+1$ rounds and is constructible in $s(\log^d n)+poly(n)$ time.

[Russel - Zuckerman '01]

Related Work and Important Results

General scheme to overcome this drawback:

1. Players pick a small committee.
2. Committee members pick a leader among them using a suitably "good" protocol, discovered via exhaustive search (so it doesn't have to be efficiently constructible).

   After long line of work, achieved ($\log^* n + O(1)$)-round protocols, with optimal resilience.
   [Russel, Zuckerman '01], [Feige '99].

None of them has bounded cheaters' edge.

Related Work and Important Results

Antonakopoulos (2006) presented three leader election protocols with bounded cheaters' edge that are poly(n)-time constructible:

Protocol	Condition	rounds
P_*	$t \leq \Theta(n/\log n)$	5
$P_\#$	$t \leq \Theta(n/\sqrt{(\log n \log\log n)})$	$5\log n$
P_+	-	polylogn

THANK YOU!