Asynchronous concurrent-initiator spanning tree algorithm using flooding

Design 1

- When two concurrent initiations are detected by two adjacent nodes that have sent a QUERY from different initiations to each other, the two partially computed spanning trees can be merged. However, this merging cannot be done based only on local knowledge or there might be cycles.

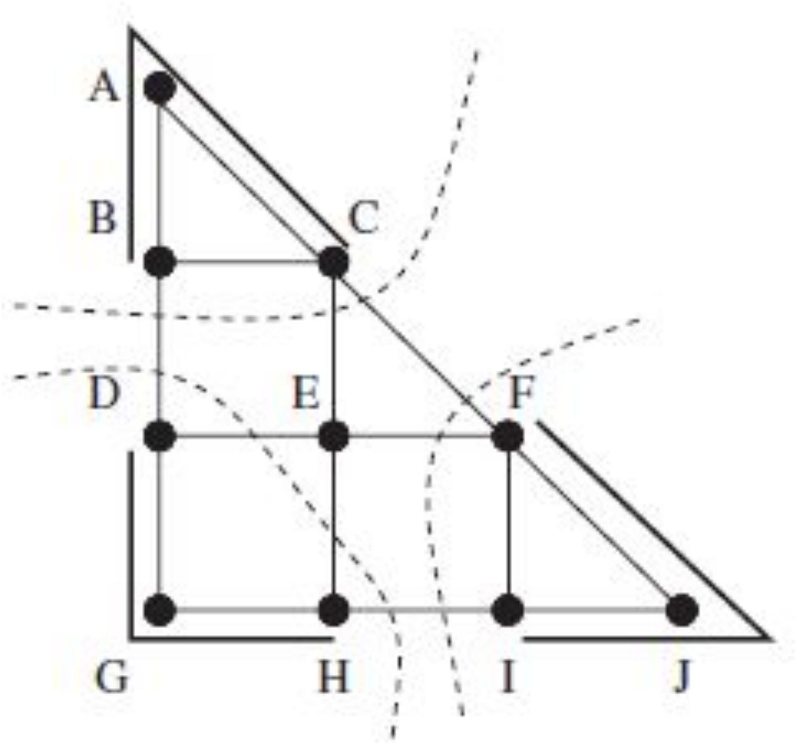
QUERY, ACCEPT, REJECT

```
(1)   When the node wants to initiate the algorithm as a root:
(1a)  if ($parent = \perp$) then
(1b)      send QUERY($i$) to all neighbors;
(1c)      $parent, myroot \longleftarrow i$.

(2)   When QUERY($newroot$) arrives from $j$:
(2a)  if $myroot < newroot$ then    // discard earlier partial execution due
                                    // to its lower priority
(2b)      $parent \longleftarrow j$; $myroot \longleftarrow newroot$; $Children, Unrelated \longleftarrow \emptyset$;
(2c)      send QUERY($newroot$) to all neighbors except $j$;
(2d)      if $Neighbors = \{j\}$ then
(2e)            send ACCEPT($myroot$) to $j$; terminate.    // leaf node
(2f)  else send REJECT($newroot$) to $j$.
          // if $newroot = myroot$ then $parent$ is already identified.
          // if $newroot < myroot$ ignore the QUERY. $j$ will update its root
          // when it receives QUERY($myroot$).

(3)   When ACCEPT($newroot$) arrives from $j$:
(3a)  if $newroot = myroot$ then
(3b)      $Children \longleftarrow Children \cup \{j\}$;
(3c)      if ($Children \cup Unrelated$) = ($Neighbors / \{parent\}$) then
(3d)            if $i = myroot$ then
(3e)                  terminate.
(3f)            else send ACCEPT($myroot$) to $parent$.
        // if $newroot < myroot$ then ignore the message. $newroot > myroot$
        // will never occur.

(4)   When REJECT($newroot$) arrives from $j$:
(4a)  if $newroot = myroot$ then
(4b)      $Unrelated \longleftarrow Unrelated \cup \{j\}$;
(4c)      if ($Children \cup Unrelated$) = ($Neighbors / \{parent\}$) then
(4d)            if $i = myroot$ then
(4e)                  terminate.
(4f)            else send ACCEPT($myroot$) to $parent$.
```



Design 2

Suppress the instance initiated by one root and continue the instance initiated by the other root, based on some rule such as tie-breaking using the processor identifier. Again, it must be ensured that the rule is correct.

Termination

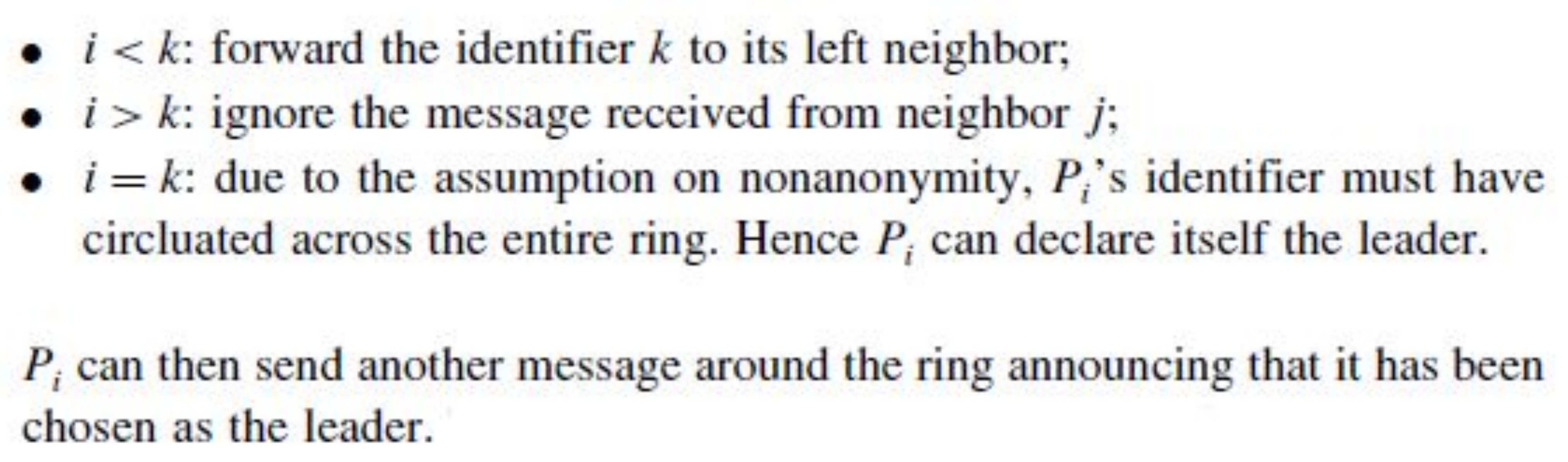
- A serious drawback of the algorithm is that only the root knows when its algorithm has terminated. To inform the other nodes, the root can send a special message along the newly constructed spanning tree edges.

Leader election

- A leader is required in many distributed systems because algorithms are typically not completely symmetrical, and some process has to take the lead in initiating the algorithm; another reason is that we would not want all the processes to replicate the algorithm initiation, to save on resources.

LCR algorithm

- The Lelang, Chang, and Roberts (LCR) algorithm assumes an asynchronous unidirectional ring.
- Each process in the ring sends its identifier to its left neighbor. When a process Pi receives the identifier k from its right neighbor Pj , it acts as follows:



- $i < k$: forward the identifier k to its left neighbor;
- $i > k$: ignore the message received from neighbor j;
- $i = k$: due to the assumption on nonanonymity, P_i's identifier must have circluated across the entire ring. Hence P_i can declare itself the leader.

P_i can then send another message around the ring announcing that it has been chosen as the leader.