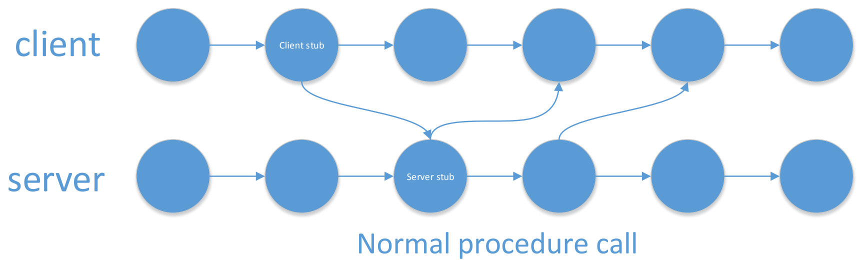

Figure 15: Remote procedure call

- reference parameters
  - passing pointers in extremely difficult
  - create array and pass as value, how to handle linked list, graph?

RPC protocol implementation needs interfaces (=stubs). Interface definition language (IDL) defines interfaces for stub implementation.

Asynchronous RPC hides communication $\Rightarrow$ communication transparency.

10 Leader Election Algorithms (week 8)

Remember that we work in

- decentralized structures and
- may want to elect a coordinator in distributed system.
- each node has two boolean variables:

  done: initially false

  leader: initially false

10.1 Properties of distributed algorithms:

Safety: not two nodes declare themselves leader

$$\forall i, j : \quad i \neq j : \neg(leader(i) \wedge \ leader(j))$$

liveness: every node eventually terminates

$$\forall I : \quad done(i) = true, \ \exists! i : leader(i) = true$$

fairness: (for Resource allocation) requests are granted in the order they are issued

10.2 Leader Election in synchronous ring

This is material taken from the book “Distributed Algorithms” by Nancy Lynch.

Network in a graph G consisting of N nodes connected by unidirected links. Use $n = 1, \dots, N$ as name of the node and (mod N) for label arithmetics.

- elected node is ”leader”
- processes have unique UID

The last is a very important point. In a distributed systemsnodes are *symmetric*, which means they are identical from the outside and cannot be distinguished. Leader election must break symmetry, by at least distinguishing one node (the leader) from all other nodes. This is possible only if nodes have unique names.

10.2.1 Basic algorithm, LCR algorithm (LeLann, Chang, Roberts)

This is a very classical and simple algorithm. We need some assumptions on the system model.

- unidirectional communication
- no messages get lost (reliable network)
- ring size unknown

The following properties can be observed”

- only leader produces output
- algorithm compares UIDs

The algorithm works as follows: Each process sends its UID around the ring. When a node receives a message (containing a UID) it compares the UID with its own. If the incoming UID is larger than the node’s ID it passes the UID on to its neighbour. If the incoming UID is smaller than the node’s ID it is dropped. If the UID is equal to the nodes ID the node declares itself *leader*.

LCR algorithm

```
The state of each node can be described by the triple:
(u, send, status), where:

u = a UID, initially is UID
send = a UID or NULL, initially is UID
status = values in{unknown, leader}, initially unknown

message generation:
send = current value of send to node i+1

state transition:
send := NULL

if incoming message is v(a UID) then
 case
   v > u : send:= v
   v = u : status := leader (leader = true)
   v < u : do nothing
 end case
```

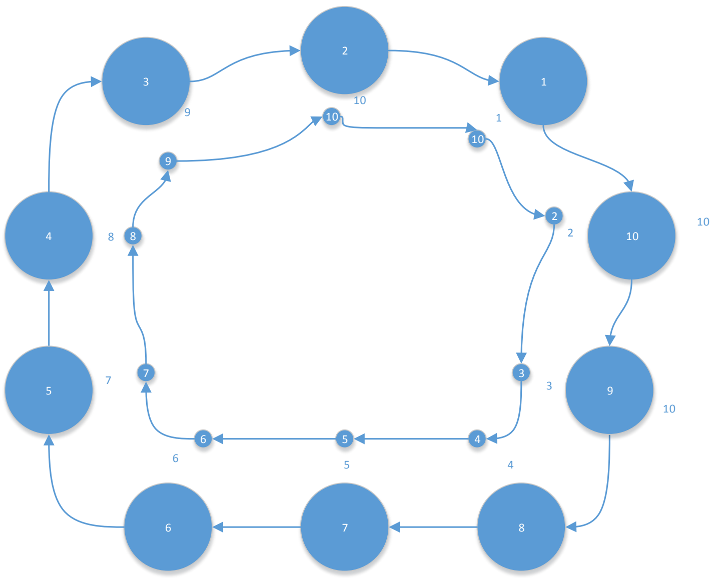
Figure 16: Ring with possible labels

In the first round each node sends its UID to its neighbour. After that in each round each node

- can receive a message
- forwards message if UID is larger than its own
- drops the message and does nothing if the UID is smaller than its own
- if own UID comes in $\Rightarrow$ declares itself leader

The two extreme cases are that nodes are aligned in increasing or in decreasing order. If nodes are arranged in decreasing order, every node will send its UID to the neighbour that has a smaller UID (except for the node with the smallest UID). The smalles UID will be immediately dropped by the node with the largest UID. Hence, all UIDs except for the smallest will be kept and passed on until they reach the node with the largest UID, which 'swallows' them all one by one.

If, however, nodes are arranged such that they are in increasing order, then all UIDs will be passed to the neighbour with larger UID, who will straight away discard the UID it has just received. Hence, only the largest UID will be passed around the ring.

<u>Correctness</u> :
Let i_{max} index of process with max(UID)
Let u_{max} its UID
show

i process i_{max} outputs "leader" after N rounds

ii no other process does have the same.
we clarify

iii After N rounds, status i_{max} = leader in addition

iv For $0 \leq r \leq N-1$, after r rounds
send$_{i_{max}+r} = u_{max}$
find max UID at distance r from i_{max}, as it has to go once around. Show(iv) by induction on r and (iii).

Complexity:

- time complexity is N rounds
- communications complexity ($\mathcal{O}(\mathbb{N}^2)$) messages in the worst case
- not too expensive in time
- many messages

10.2.2 Algorithm of Hirschberg and Sinclair(HS-algorithm)

Similar to the LCR algorithm also the HS algorithm elects the node with the largest ID as the leader. The main property is that

- it reduces messages to $\mathcal{O}(\mathbb{N} \log \mathbb{N})$

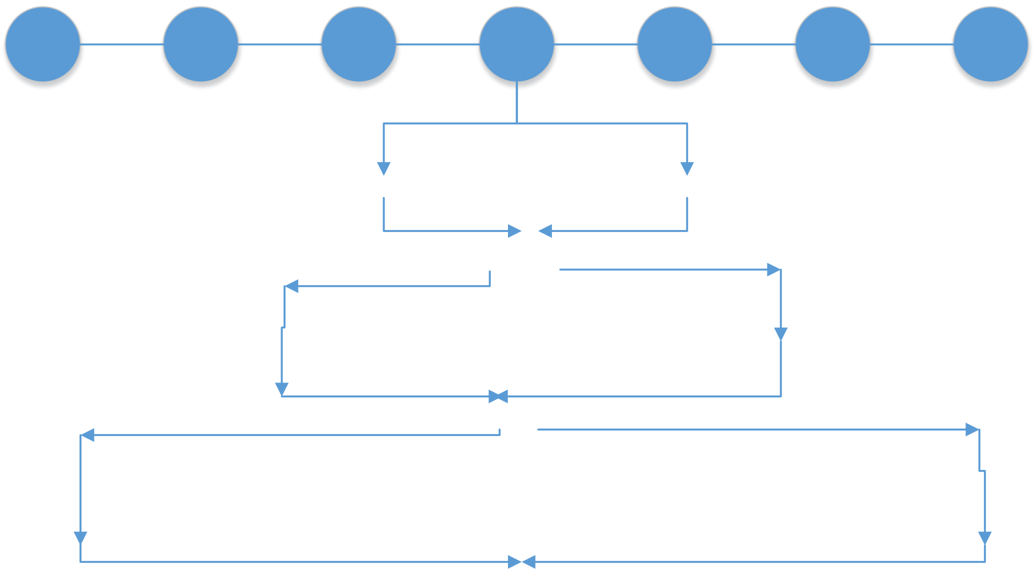

Figure 17: Hirschberg-Sinclair algorithm

The HS-algorithm is based on comparison of UIDs carried out in each node. In the HS-algorithm a node will always send out its UID to both neighbours up to a distance of 2^l in phase l. The outbound message will be checked by each node on the way. If the UID carried by the message is larger than the UID of the receiving node then the message with UID will be passed on, if the UID is smaller the message is discarded. If a node receives its own UID it has the largest UID and declares itself leader. At distance 2^l the message turns back and is returned as an inbound message back to the sender without being inspected by relaying nodes.

In consequence, a node receives its message back only if there has been no larger UID on the way. In this case it will double the distance for its messages to travel. All nodes continue until their message is swallowed. Only the largest UID will return to the sender at some point.

HS-algorithm

```
each process has states with componets
  u, type UID initially i's UID
  send+, containing NULL or (UID, flag{in, out}, hopcount)
      initially( i's UID, out, 1)
  send−, as send+
  status ∈ {unknown, leader}, initially unknown
  phase ∈ ℕ initially 0

  message generation
  send current value of send+ to process i+1
  send current value of send− to process i−1

  state transitions
     send+ := NULL
     send− := NULL
  if message from i−1 is (v, out, h) then
     v > u ∧ h > 1: send+ := (v, out, h−1)
     v > u ∧ h = 1: send− := (v, in, 1)
     v = u : status := leader
  if message from i+1 is (v, out, h) then
     v > u ∧ h > 1: send− := (v, out, h−1)
     v > u ∧ h = 1: send+ := (v, in, 1)
     v = u : status := leader
```

outgoing UIDs are compared, if u_i is smaller than any u_j on the way, it is discarded by node j. Otherwise it is passed on and if $u_i = u_j$ then node j has received its own ID and is leader. Inbound messages are always passed through.

HS-algorithm(continued)

```
if message from i=1 is(v, in, 1) and v ≠ u
then send+ =(v, in, 1)
if message from i+1 is(v, in, 1) and v ≠ u then
     send− = (v, in, 1)
ifboth messages from i−1 and i+1 are (u, in, 1) then
phase++,
  send+ = (u, out, $2^{phase}$))
  send− = (u, out, $2^{phase}$))
```

Complexity :

The number of messages sent is in the first phase 4 for each node. There is one message sent in each direction and one message is returned. In general, we can say that phase l is started only, if both message returned in phase $l-1$.

The total number of phases is at most

$$1 + \lceil \log N \rceil$$

the total number of messages is at most $8N(1 + \lceil \log N \rceil) \sim \mathcal{O}(N \log N)$ with constant factor of approx 8
Time Complexity is at most 3N if N is power of 2, otherwise 5N