Algorithmic Approach to D.S.

2020/2021

20/09/21

LEADER ELECTION

COMPLETE GRAPH LB

- WORST CASE
  - Process i receive IDs in increasing order
  - it receive n-1 msgs
  - it broadcast (n-1 msgs) for each one he receive
  - there are n processors

  ⇒ #MSGS = (n-1)(n-1)n = $O(n^3)$
- BEST CASE:
  - Process i receives IDs in non-increasing order
  - it receive n-1 msgs
  - there are n processors

  ⇒ #MSGS = (n-1) n = $O(n^2)$

CHANG/ROBERTS

- ASSUME that exists a UNIDIRECTIONAL RING of non faulty processors
- PROCLAMATION is needed (<End> msg)

ALGO

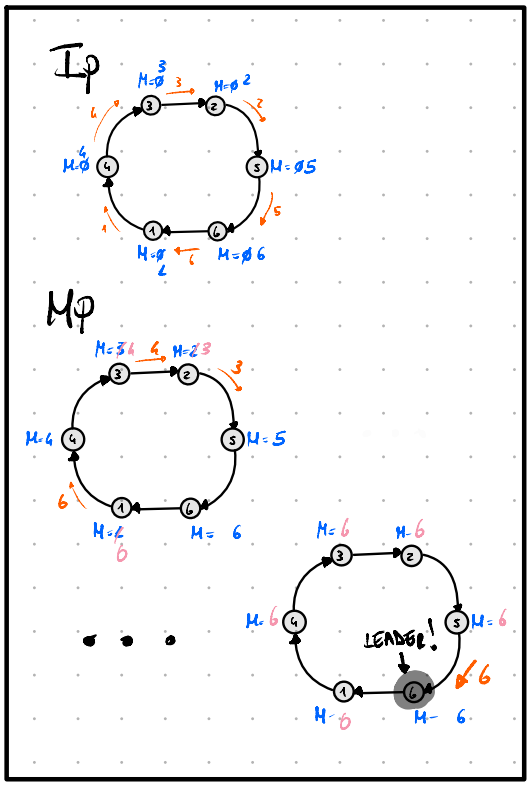
Ip

```
{M = ∅}
    M = P
    SEND M
```

// P is PROCESSOR ID

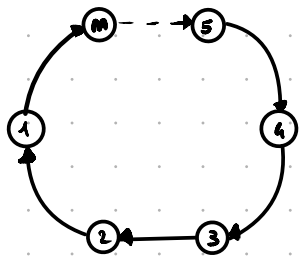
Mp

```
{RECEIVING J}
    IF (J > M)
        M = J
        SEND M
    IF (M = P)
        PROCLAMATION
        STOP
```



NOTE → IGNORE PROCLAMATION

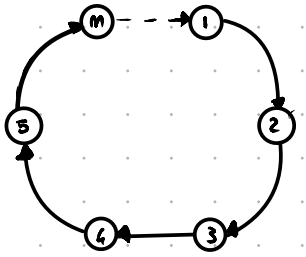
WORST CASE



- #MSGs $\underbrace{m}_{\#\text{MSGs of LEADER}} + (m-1) + (m-2) + \dots + 1 =$

$$= \sum_{i=1}^{m} i = \frac{m(m+1)}{2} \cdot \Theta(m^2)$$

BEST CASE



- #MSGs $\underbrace{m}_{\#\text{MSGs of LEADER}} + \underbrace{1 + \dots + 1}_{m} = 2m \cdot \Theta(m)$

AVG CASE

- The PROB. for a MSG. to make one step is $1/m$, so to complete the circle is $\sum_{i=1}^{m} i \cdot \mathbb{P}(\text{MAKING ONE STEP}) \leq \Theta(m \log m)$

⇒ $O(m \log m)$

ROUND BASED LEADER ELECTION

- A node become PASSIVE if it's value p is less then the one it receave
- A passive node only forwards messages

FRANKLIN ALGO (ON BIDIRECTED G)

- Processes send msgs in 2 directions
- At each round at least half Procs become PASSIVE (for every 2 near nodes one should win) → #ROUNDS $O(\log n)$
- In each roound ACTIVE send 2 msgs, PASSIVE forward 2 msgs → #MSGs = $2m \cdot O(\log n) = O(m \log m)$

27/09/21

ELECTION ON FAULTY SYSTEMS

- We assume that links are faulty free.
- We assume a TIMEOUT. (BOUNDED SYNCHRONICITY)

⇒ It follows that we can DETECT FAULTY PROCESS

- Assume a fully-connected topology.

IDEA

- If the leader fail, every process will try to replace it

BULLY ALGO [GARCIA-MOLINA]

Detection PH.

- TIMEOUT reached while waiting LEADER's ping reply.

Preparation PH.

- P_i sends "ELECTION" msg to every $P_j \mid j > i$

Election PH.

- If no P_j reply in-TIME then I'M THE LEADER ⇒ Proclamation PH
- If $P_j \mid j > i$ receive P_i msg ⇒ P_j reply "YOU CANNOT BE THE LEADER" ⇒ P_j (IF NOT YET) goes to PREPARATION PH. and P_i goes to PROCLAMATION PH. (will wait a LEADER)

Proclamation PH.

- LEADER P_i LEADER send it's "ID" to all $P_j \mid j < i$
- Other P_s waits for LEADER PROCLAMATION MSG; If TIMEOUT expires ⇒ goto PROCLAMATION PH.

LEADER FAILED DURING IT'S OWN PROCLAMATION

COMPLEXITY

- When P_i detects LEADER FAILURE it broadcast to his $(n-1)$ neighborhood, getting back at most $(n-1)$ replies
- WORST CASE:

$$\#MSGS_{WORST} = O(n^3)$$

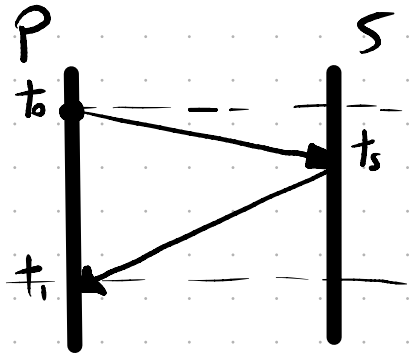
NOTE

- If a failed Processore is restarted it try to elect himself, so if it's the greatest ID proc., we will have 2 LEADERS, we should so consider EPOCHS to keep the latest elected one.

27/09/21

PHYSICAL CLOCKS

CHRISTIAN'S PHISICAL CLOCK



$$\Rightarrow t_1 = t_s + \frac{\text{ROUND-TRIP}}{2}$$

COMPUTER CLOCK

- Compute the average of the times of every processor; then each one set the average as it's TIME

BERKEG'S PHYSICAL CLOCK

- Same as COMPUTER CLOCK, but it consider ROUND-TRIP DELAYS TOO

LOGICAL CLOCKS

LAMPORT'S LOGICAL CLOCK

- Each P_i starts with a var. $L = \emptyset$
- BEFORE a LOCAL EVENT $L = L + 1$
- A msg m sent have L as TIMESTAMP
- When a msg m is received $L = \max(L, \text{TIMESTAMP}(m)) + 1$

RECEIVING m IS AN EVENT

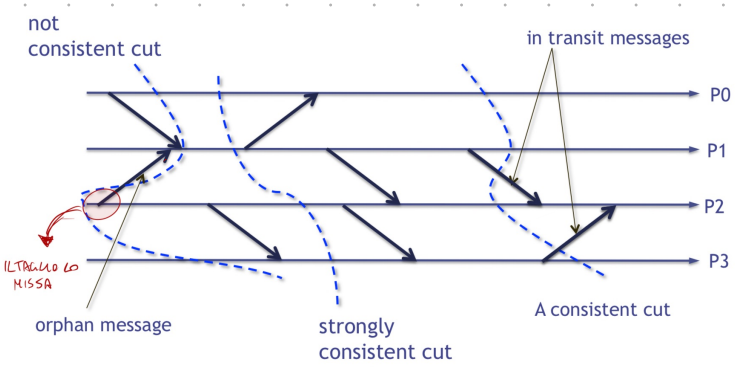
VECTOR CLOCKS

- P_i start with a vector $V = [V_1, \_\_, V_n]$ inizialized to $\emptyset$
  - $V[i]$ contains #events timestamped by P_i
  - $V[j]$ $\forall j \neq i$ contains #events of P_j which have interacted with P_i
- BEFORE P_i LOCAL event $V[i] = V[i] + 1$
- A msg sent has the whole vector V
- When a msg m is received from P_j $V[j] = \max(V[j], V_{RCV}[j])$ $\forall j$

6/10/21

FAULTS AND RECOVERY

CUT

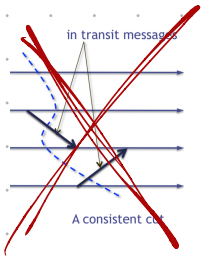


CHECKPOINT

- Restart from a CONSISTENT CUT; So in the cut we save the state of the system.

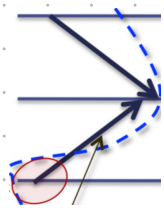
NB

- In case of RELIABLE system we must use only STRONGLY CONNECTED CUTS



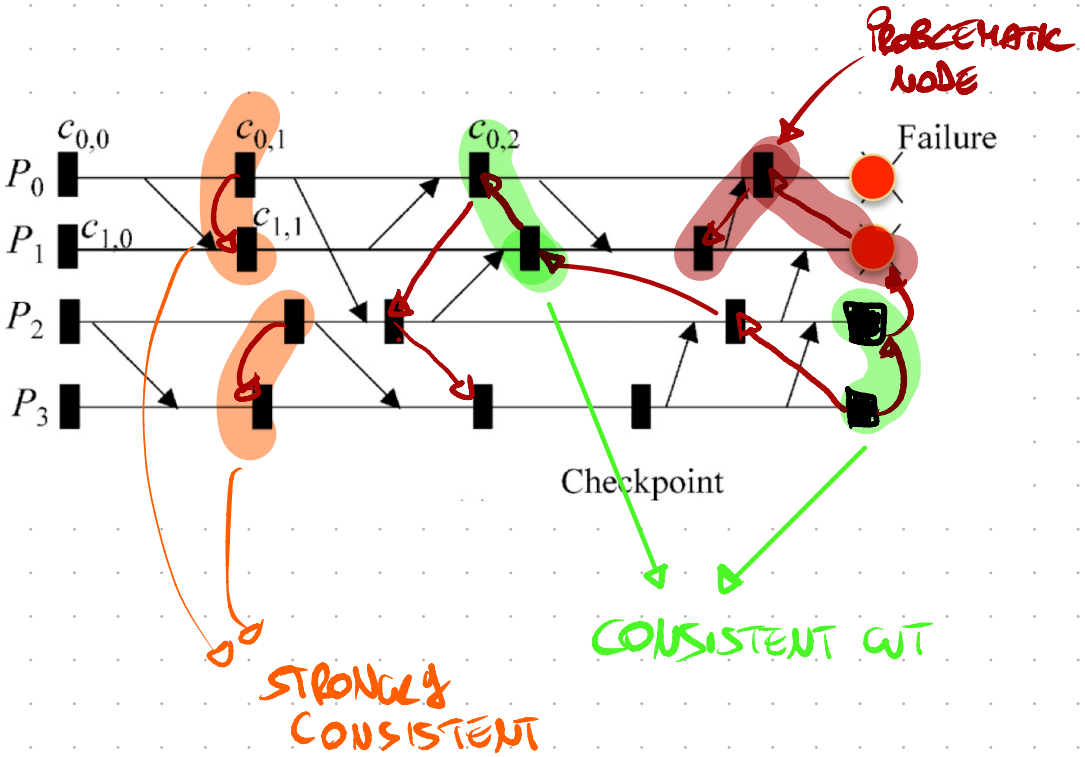
- In this case is not guaranteed that, after a restart will be sent the very same msgs.

ORPHAN MSGS

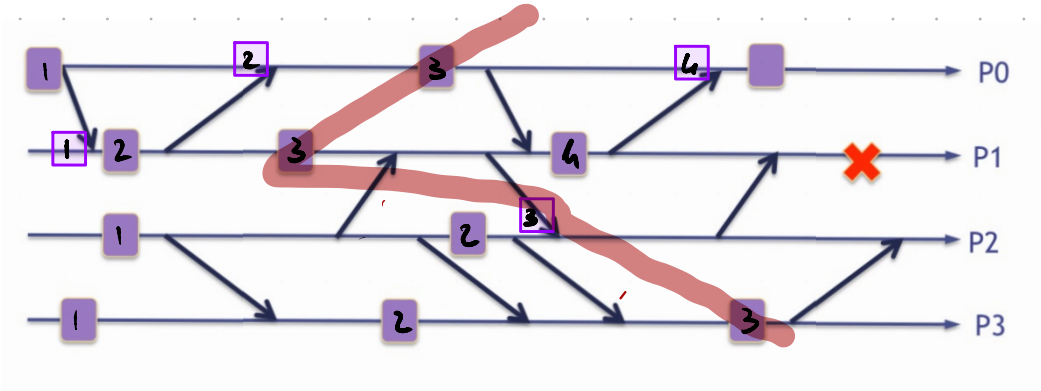


- If it restart the msg will be RECEIVED 2 times from the top processor

ROLLBACK DEPENDENCY GRAPH



CIC



GROUP COMUNICATION

TYPES OF EVENTS

1) REQUEST
2) INDICATION (RESPONSE)
3) CONFIRMATION (ACK

upon event ⟨ *Event1* | att_1^1, att_1^2, ... ⟩ do
    something
    trigger ⟨ *Event2* | att_2^1,att_2^2, ... ⟩; // *send some event*

trigger <*send* | *dest*, [data1, data2, ...] >

upon event <*send* | *src*, [data1, data2, ...] > do

TYPES OF NODE FAILURE

1) CRASH STOP
2) OMISSION
3) CRASH RECOVERY
4) BYZANTINE

TYPES OF CHANNEL FAILURE

1) FAIR LOSS LINK — NON-ZERO PROB. OF DELIVERY
2) STUBBORN LINK — MSGS DELIVERED INFINITE TIMES
3) PERFECT LINK — MSGS DELIVERED EXATLY ONCE

1) FAIR LOSS LINKS

- Request: ⟨flp2pSend | dest, m⟩
  - Request transmission of message m to node dest
- Indication: ⟨flp2pDeliver | src, m⟩
  - Deliver message m sent by node src

PROPERTIES
- FL1 (FAIR LOSS): IF m SENT ∞ w/ NO CRASH ⇒ m INFINITELY DELIVERED
- FL2 (& FINITE DUPLICATION): IF m SENT ∞ ⇒ m DELIVERED ∞
- FL3 (NO CREATION): m NOT DELIVERED IF NOT SENT

2) STUBBORN LINKS

- Request: ⟨sp2pSend | dest, m⟩
  - Request the transmission of message m to node dest
- Indication: ⟨sp2pDeliver src, m⟩
  - deliver message m sent by node src

PROPERTIES
- SL1 (STUBBORN DELIVER): IF m sent to P_j CORRECT, and NO CRASH of P_i ⇒ m DELIVERED ∞
- SL2 (NO CREATION): m DELIVERED ⇒ m SENT

Implementation
- Use the Lossy link
- Sender stores every message it sends in *sent*
- It periodically resends all messages in *sent*

3) PERFECT LINK

- Request: ⟨pp2pSend | dest, m⟩
  - Request the transmission of message m to node dest
- Indication: ⟨pp2pDeliver | src, m⟩
  - deliver message m sent by node src

PROPERTIES
- PL1 (RELIABLE DELIVERY): IF NO CRASH ⇒ m EVENTUALLY DELIVERED
- PL2 (NO DUPLICATION): m DELIVERED AT MOST ONCE
- PL3 (NO CREATION): m DELIVERED ⇒ m SENT

Implementation
- Use Stubborn links
- Receiver keeps log of all received messages in Delivered
  - Only deliver (call pp2pDeliver) messages that weren't delivered before

BES EFFORT BROADCAST

IDEA: ALL PERFECT LINKS SERVER CRASH

Request: < bebBroadcast | m >: Used to broadcast message m to all processes.
Indication: < bebDeliver | src, m > : Used to deliver message m broadcast by process src.

PROPERTIES
- BEB1 (BEST EFFORT VALIDITY): IF P_i, P_j CORREC ⇒ m EVENTUALLY DELIVERED
- BEB2: NO DUPLICATION
- BEB3: NO CREATION

CASUAL ORDER BROADCAST

- Request: ⟨coBroadcast | m⟩
- Indication: ⟨coDeliver | src, m⟩

COMON → CB (CASUAL DELIVERY): IF P_i DELIVERS m_i, MUST HAVE DELIVERED ALL MSGS CASUALLY PRECEDING m_i

TOTALLY ORDERED BROADCAST

- IF P_i AND P_j BOTH DELIVERS m_1, m_2, THEY DELIVER THEM IN SAME ORDER.

RELIABLE BEB

- BEB + RELIABILITY PROPERTY

Request: <rbBroadcast | m >: Used to broadcast message m.

Indication: < rbDeliver | src,m> : Used to deliver message m broadcast by process src.

RB1=Beb1: *Validity*: If a correct process p_i broadcasts a message m, then p_i eventually delivers m.

RB2=Beb2: *No duplication*: No message is delivered more than once.

RB3=Beb3: *No creation*: If a message m is delivered by some process p_j, then m was previously broadcast by some process p_i.

RB4: *Agreement*: If a message m is delivered by some correct process p_i,then m is eventually delivered by every correct process p_j .

MUTUAL ESCLUSION

- ME1 : Mutual Exclusion
  - At most one process can remain in CS at any time
  - Safety property
- ME2 : Freedom from deadlock
  - At least one process is eligible to enter CS
  - Liveness property
- ME3 : Fairness
  - Every process trying to enter must eventually succeed
  - Absence of starvation
- A measure of fairness: bounded waiting
  - Specifies an upper bound on the number of times a process waits for its turn to enter SC -> n-fairness (n is the MAXIMUM number of rounds)
  - FIFO fairness when n=0

LAMPORT ALGO

- "VOGLIO ENTRA"
- "Ack"
- "Ack"
- ⋮

} TRAPPI Ack HA FUNZIONA

RICART & AGRAWALA

- "VOGLIO ENTRA" + TIMESTAMP
- "Ack" SOLO SE NON VOGLIO ENTRARE O VOGLIO E HO TIMESTAMP PIÙ ALTO
- USCENDO MANDO "Ack"

TOKE

TOKEN RING

- NON SERVONO PRESENTAZIONI
- APPENA MI ARRIVA IL TOKEN ENTRO

NAIVE ALGO

("IL KING È CHI HA IL TOKEN")

- "VOGLIO ENTRA"
- IL KING METTE IN CODA
- KING ESCE DA CS E PASSA IL TOKEN AL PRIMO IN CODA

⇒ STARVATION (POSSO ENTRA DUE VOLTE DI FILA)

⇒ POSSO DIVENTARE KING SENZA AVERLO CHIESTO

SUKUZZI-KAZZAKI

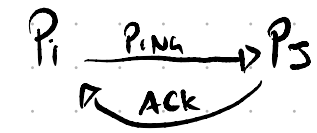
- OGNUNO TIENE LISTA RICHIESTE "VOGLIO ENTRA"
- IL KING HA LA CODA E LA "LAST" (LISTA DI CHI È STATO KING)
- USCENDO AGGIORNA LA LAST E PASSA TOKEN.

QUORUM BASED

- GRIGLIA
- DEADLOCK

FAILURE DETECTORS

PING ACK



HEARTBEATING

Failure Detectors ("hide/abstract time=delays")

- **Basic properties**
  - Completeness
    - Every crashed process is suspected
  - Accuracy
    - No correct process is suspected

Both properties comes in two flavours : Strong and Weak

- Strong Completeness
  - Every crashed process is eventually suspected by ***every*** correct process
- Weak Completeness
  - Every crashed process is eventually suspected by ***at least*** one correct process
- Strong Accuracy
  - No correct process is ***ever*** suspected
- Weak Accuracy
  - There is ***at least*** one correct process that is ***never*** suspected

13

PERFECT FAILURE DETECTOR (ASSUMING SYNCHRONOUS SYSTEM)

P

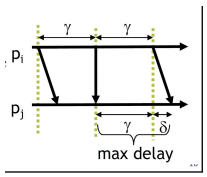
P: **set of processes**

```
Initialize HBTimeout and DetectTimeout
Upon event (HBTimeout)
   For all pi in P
        Send HeartBeat to pi
        startTimer (gamma, HBTimeout)

Upon event Receive HeartBeat from pj
   alive:=alive ∪ pj

Upon event (DetectTimeout)
   crashed := P \ alive
   for all pi in crashed Trigger (crashed, pi)
   alive := ∅
   startTimer (delta+gamma, DetectTimeout)
```

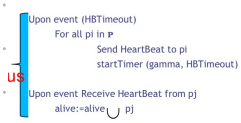
• MAX DELAY =>



EVENTUALLY PERFECT FAILURE DETECTOR (ASYNCHRONOUS SYSTEM)

• ASSUMING THERE IS A FINITE MAX DELAY (PARTIALLY SYNCHRONOUS)

susp
initia




```
Upon event (DetectTimeout)
   for all pi in P
        if pi not in alive and pi not in suspected
             suspected :=suspected ∪pi
             Trigger (suspected, pi)
        if pi in alive and pi in suspected
             suspected :=suspected \ pi
             Trigger (restore, pi)
             T:=T+delta
   alive :=∅
   startTimer (T, DetectTimeout)
```

• When $T > \gamma + \delta$ the system becomes synchronous

CONSENSUS

PROPERTIES

- **Termination**
  - **Every correct node eventually decides**
- **Agreement**
  - **No two correct processes decide differently**
- **Validity**
  - **Any value decided is a value proposed**
- **Integrity:**
  - **A node decides at most once**

- **A variant: UNIFORM CONSENSUS**
  - **Uniform agreement: No two processes decide differently**

algorithm I

- A P-based (fail-stop) consensus algorithm
- The processes exchange and update proposals in rounds and decide on the value of the non-suspected process with the smallest id [Gue95]

=)

- The processes go through rounds incrementally (1 to n): in each round, the process with the id corresponding to that round is the leader of the round
- The leader of a round decides its current proposal and broadcasts it to all
- A process that is not leader in a round waits (a) to deliver the proposal of the leader in that round to adopt it, or (b) to suspect the leader

Consensus algorithm II

- A P-based (i.e., fail-stop) uniform consensus algorithm
- The processes exchange and update proposal in rounds, and after n rounds decide on the current proposal value [Lyn96]

=)

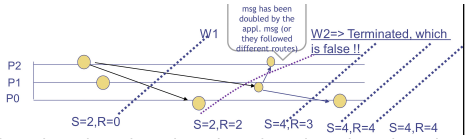
- The "Hierarchical Uniform Consensus" algorithm uses a **perfect failure-detector, a best-effort broadcast** to disseminate the proposal, **a perfect link abstraction** to acknowledge the receipt of a proposal, and **a reliable broadcast** to disseminate the decision
- Every process maintains a single proposal value that it broadcasts in the round corresponding to its **rank**. When it receives a proposal from a more importantly ranked process (so the hierarchical), it adopts the value
- In every round of the algorithm, the process whose **rank** corresponds to the **number of the round** is the leader.
- A round here consists of two communication steps: within the same round, the leader broadcasts a PROPOSAL message to all processes, trying to impose its value, and then expects to obtain an acknowledgment from all correct processes
- Processes that receive a proposal from the leader of the round adopt this proposal as their own and send an acknowledgment back to the leader of the round
- If the leader succeeds in collecting an acknowledgment from all processes except detected as crashed, the leader can decide. It disseminates the decided value using a reliable broadcast communication abstraction

GLOBAL STATE COLLECTION

- SOME GLOBAL STATES
  - TERMINATION STATE : useful to start next phase
  - DEADLOCK DETECTION
  - #MSGS EXANGED
  - #PROCS AT GIVEN MOMENT
- TERMINATION DETECTION
  - (1) #SENT == #RCVED_PROCESSES
  - IF THERE ARE ONLY PASSIVE PROCSS, IT SEEMS TO BE IN TERMINATION STATE (NO MSG EXANGE) BUT THOSE PASSIVE CAN BE PROCESSING

MATTERN SOL.

- Check if there is a consistent cut. If there is double check, the cut are called WAVES.



MISRA

- TO CHECK ALIVE PASSIVE: if P_i PASSIVE forward the msg then is ALIVE
  - initially each of the n process state is white, Initiator is process 0
  - On msg reception on any Process: state=black;
  - (On each Pi, i in [1..n-1]) On token reception:
    - if state= black token:=1 else token:=token+1
    - state = white; forward token
  - (On P0) On token reception:
    - if (state=white & token=n) "terminated", else state=white; forward token=1

Diikstra-Feiien
-van Gasteren
ALGO

- $0 \to m-1 \to m-2 \to 1 \to 0$
- Ci sono msg che non sono TOKEN
- SE MANDI A ID' > TUO_ID ⇒ DIVENTI NERO
- SE SEI NERO E MANDI IL TOKEN ⇒ TOKEN NERO
- SE SEI BIANCO E PORCUPRODI ⇒ RESTI BIANCO
- SE SEI NERO E MANDI MSG A ID' < ID_TUO ⇒ DIVENTI BIANCO
- SE 'INITIATOR BIANCO' RICEVE TOKEN BIANCO ⇒ TERMINATION

DEADLOCK

- **RES. DEADLOCK**: When there is a cycle in WAIT-FOR-RES-GRAPH
- **COMMUN. DEAD.**: Communication deadlock is detected if the initiator receives ack from all its successors (implying none of its successors can unblock it =>it is deadlocked)