Dependable Distributed Systems
Exercise week 2
October 9th, 2024

Exercise 1

With reference to the synchronization of physical clocks, provide the definition of internal and external clock synchronization. In addition, consider a system composed of two processes p1 and p2 and one UTC server p_S. Let us assume that:

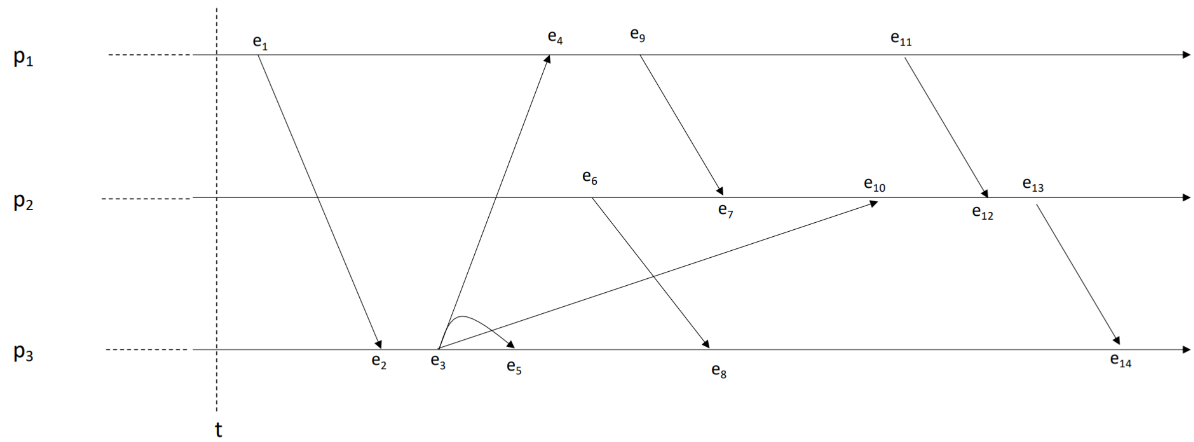
- p_1 and p_2 communicate with p_S by using perfect point-to-point links;
- the maximum latency of the channel between p_S and p_1 is 1 ms;
- the maximum latency of the channel between p_s and p_2 is 2 ms.

In addition, let us assume that p_1 and p_2 start a clocks synchronization procedure at a certain time t by running the Christian algorithm. Answer to the following questions:

1. How much is the accuracy bound D_{ext} of the external synchronization obtained by p_1 and p_2 at the end of the synchronization?
2. Is the current system internally synchronized? If yes, determine the internal synchronization bound D_{int} obtained at the end of the procedure.

Exercise 2

Let us consider the execution history depicted in the figure

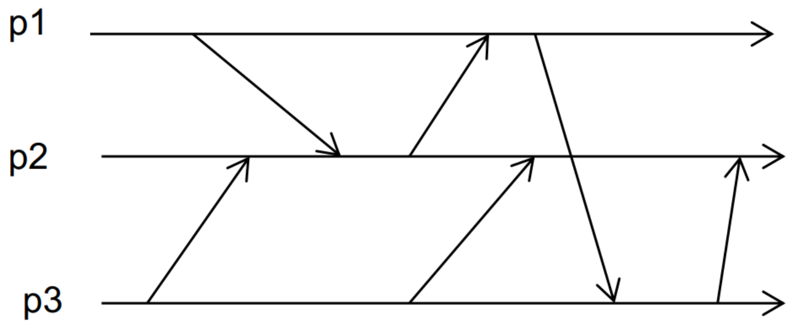


Given the run depicted in the figure state the truthfulness of the following sentences:			
a	According to the happened-before relation, $e_5 \rightarrow e_7$	T	F
b	According to the happened-before relation, $e_4 \parallel e_5$	T	F
c	Let CK_i be the variable storing the scalar logical clock of process p_i. Let us assume that at time t, $CK_i = 0$ for each process p_i. The logical clock CK_2 associated to e_6 is strictly larger than the logical clock CK_1 associated to e_1	T	F
d	Let CK_i be the variable storing the scalar logical clock of process p_i. If at time t $CK_3 = 0$ then the logical clock associated to e_8 is $CK_3=3$	T	F
e	Let CK_i be the variable storing the vector logical clock of process p_i. If at time t $CK_3 = [0, 0, 0]$ then the logical clock associated to e_8 is $CK_3=[3, 0, 3]$	T	F

For each point, **provide a justification for your answer**

Exercise 3 (NOTA: assegnare e per una più facile correzione)

Describe timestamping techniques based on scalar logical clocks and vector logical clocks. In addition, considering the execution reported in Figure, answer to the following questions:



1. Apply the scalar clock timestamping technique to the execution assigning a timestamp to each event
2. Apply the vector clock timestamping technique to the execution assigning a timestamp to each event
3. List all pairs of concurrent events in the proposed execution

Exercise 4

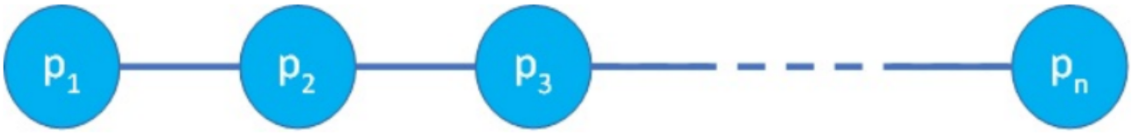
Consider an asynchronous message passing system that uses vectors clock to implement some causal consistency check. The message passing system is composed by 4 processes with IDs 1 to 4, and, as usual, the ID is used as displacement in the vector clock (i.e., the locations are (p1, p2, p3, p4)). Vector clocks are updated increasing before send. Processes communicate by point2point links. You start debugging process p1 (the process with id 1) in the middle of the algorithm execution. You see the following stream of messages exiting and entering the ethernet card of process p1:

- `Time 00:00 - EXITING: Send Message [MSG CONTENT] Vector Clock: (1, 0, 0, 0)`
- `Time 00:05 - ENTERING: Rcvd Message [MSG CONTENT] Vector Clock: (1, 2, 3, 1)`

1. Draw an execution that justifies the vectors clock you are seeing. Is such an execution unique?
2. There exists an execution that justifies the vector clocks and where there exists at least a process that does not send any message? Justify your answer.

Exercise 5

Let us consider a distributed system composed of N processes p1, p2, … pn, each one having a unique integer identifier. Processes are arranged in line topology as in the following figure, with consecutive identifiers.



Let us assume that there are no failures in the system (i.e., processes are always correct) and that topology links are implemented through perfect point-to-point links.

Write the pseudo-code of a distributed algorithm that builds the abstraction of a perfect point-to-point link between <u>any</u> pair of processes (i.e., also between those that are not directly connected, such as p1 and p3).

Exercise 1

With reference to the synchronization of physical clocks, provide the definition of internal and external clock synchronization. In addition, consider a system composed of two processes p1 and p2 and one UTC server p_S. Let us assume that:

- p_1 and p_2 communicate with p_S by using perfect point-to-point links;
- the maximum latency of the channel between p_S and p_1 is 1 ms;
- the maximum latency of the channel between p_s and p_2 is 2 ms.

In addition, let us assume that p_1 and p_2 start a clocks synchronization procedure at a certain time t by running the Christian algorithm. Answer to the following questions:

1. How much is the accuracy bound D_{ext} of the external synchronization obtained by p_1 and p_2 at the end of the synchronization?
2. Is the current system internally synchronized? If yes, determine the internal synchronization bound D_{int} obtained at the end of the procedure.

EXTERNAL SYNC: CLOCKS OF ALL PROCESS ARE SYNC WITH AN EXTERNAL SOURCE OF TIME (UTC). THE CLOCKS C_i ARE EXT SYNC WITH A TIME SOURCE S (UTC) IF FOR EACH TIME INTERVAL I:

$$|S(\tau) - C_i(\tau)| < D$$

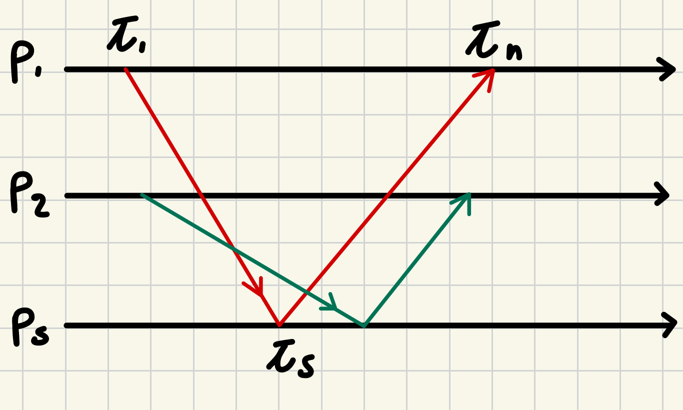
$i = 1,..,n$
D= BOUND
τ = REAL TIME IN I

INTERNAL SYNC: CLOCKS OF ALL PROCESS ARE SYNC BETWEEN THEM. CLOCKS ARE INT SYNC IN A TIME INTERVAL I IF:

C_i CLOCKS OF P_i
C_j CLOCKS OF P_j

$$|C_i(\tau) - C_j(\tau)| < D$$

$i,j = 1,..,n$
D= BOUND
τ = REAL TIME IN I



a) P_1: RTT MAX = $2 \cdot 1 = 2ms$
RTT MIN = $\tau_{min} = 0ms$

$$D_{EXT_1} = \pm\left(\frac{RTT}{2} - \tau_{min}\right) = \pm 1ms$$

P_2: RTT MAX = $2 \cdot 2 = 4ms$
RTT MIN = $\tau_{min} = 0ms$

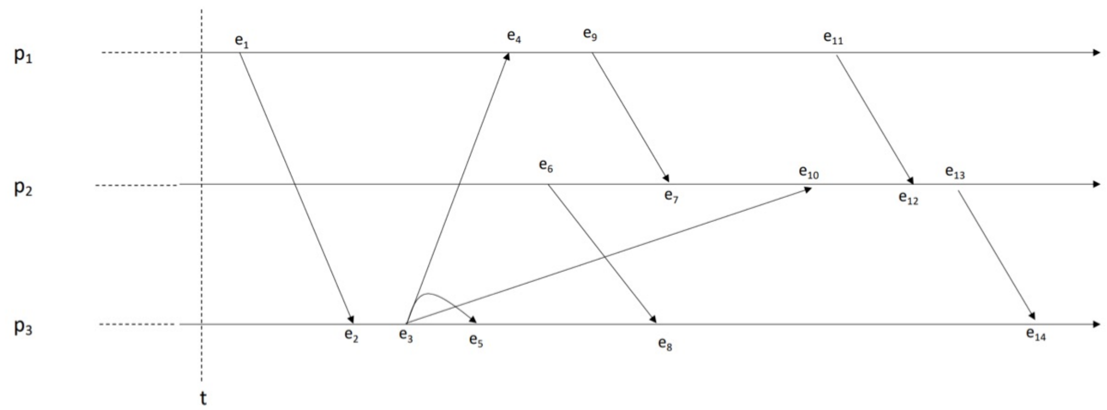
$$D_{EXT_2} = \pm\left(\frac{RTT}{2} - \tau_{min}\right) = \pm 2ms$$

b) EXT $\Rightarrow$ INT WITH BOUND 2D

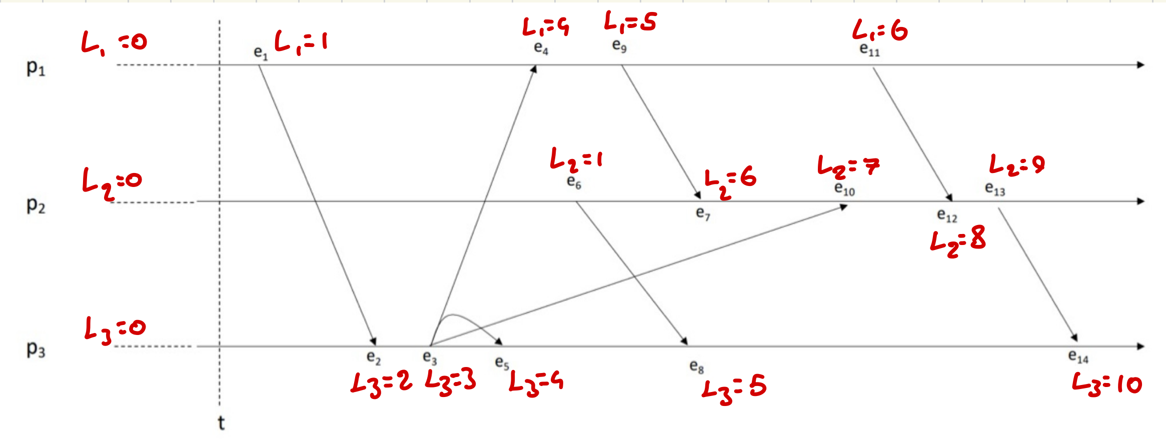
MAX (1 ms, 2ms) = 2ms = D $\rightarrow$ INT SYNC = 2D = 4ms

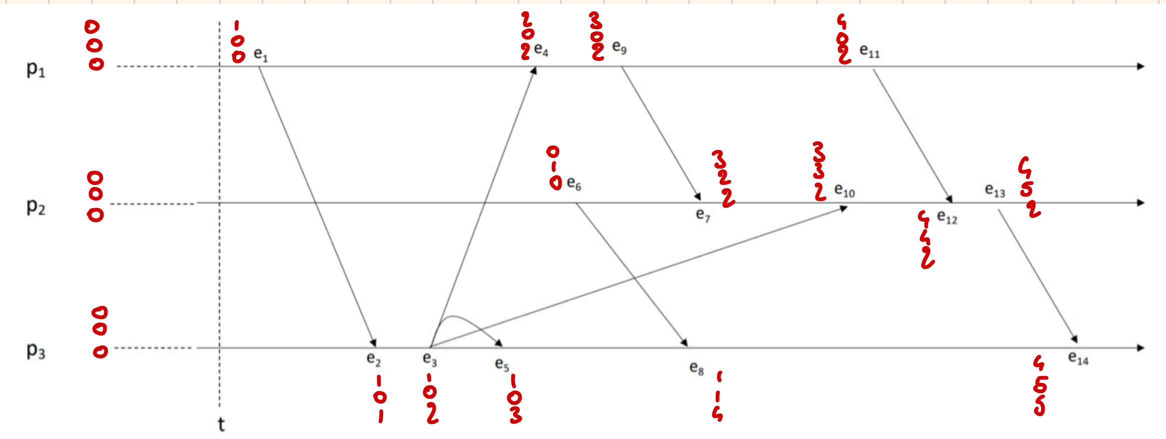
Exercise 2

Let us consider the execution history depicted in the figure



Given the run depicted in the figure state the truthfulness of the following sentences:			
a	According to the happened-before relation, $e_5 \rightarrow e_7$	T	X
b	According to the happened-before relation, $e_4 \parallel e_5$	X	F
c	Let CK_i be the variable storing the scalar logical clock of process p_i. Let us assume that at time t, $CK_i = 0$ for each process p_i. The logical clock CK_2 associated to e_6 is strictly larger than the logical clock CK_1 associated to e_1	T	X
d	Let CK_i be the variable storing the scalar logical clock of process p_i. If at time t $CK_3 = 0$ then the logical clock associated to e_8 is $CK_3=3$	T	X
e	Let CK_i be the variable storing the vector logical clock of process p_i. If at time t $CK_3 = [0, 0, 0]$ then the logical clock associated to e_8 is $CK_3=[3, 0, 3]$	T	X





a) F: THE EVENTS ARE CONCURRENTY AND THEY ARE IN A DIFFERENT LOCAL HISTORY.

b) T: e_4 AND e_6 ARE CONCURRENT BECAUSE THEY ARE IN A DIFFERENT LOCAL HISTORY BUT HAVE THE SAME LOGICAL CLOCK (4).

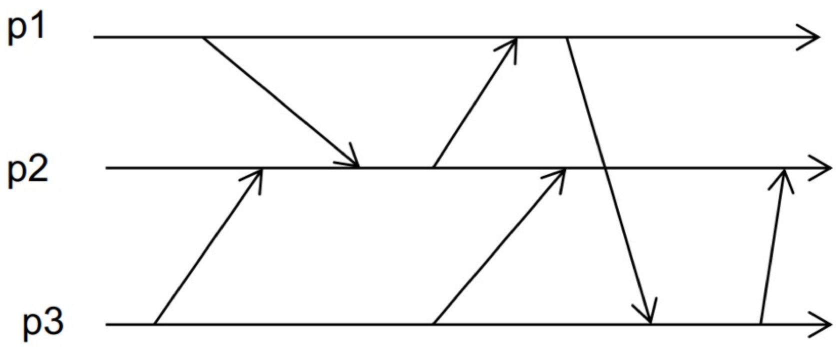
c) F: THEY HAVE THE SAME LOGICAL CLOCK (1).

d) F: e_8 HAS A $L_3 = 5$.

e) F: e_8 HAS A VECTOR CLOCK EQUAL TO $\begin{bmatrix} 1 \\ 1 \\ 4 \end{bmatrix}$.

Exercise 3 (NOTA: assegnare e per una più facile correzione)

Describe timestamping techniques based on scalar logical clocks and vector logical clocks. In addition, considering the execution reported in Figure, answer to the following questions:

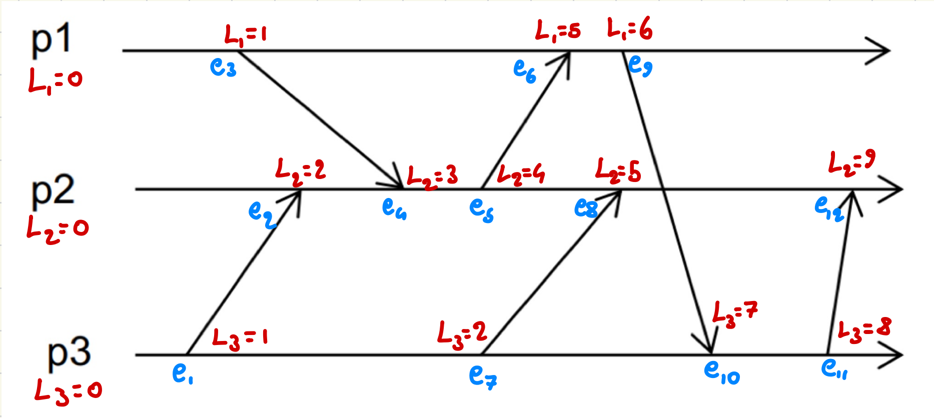


1. Apply the scalar clock timestamping technique to the execution assigning a timestamp to each event
2. Apply the vector clock timestamping technique to the execution assigning a timestamp to each event
3. List all pairs of concurrent events in the proposed execution

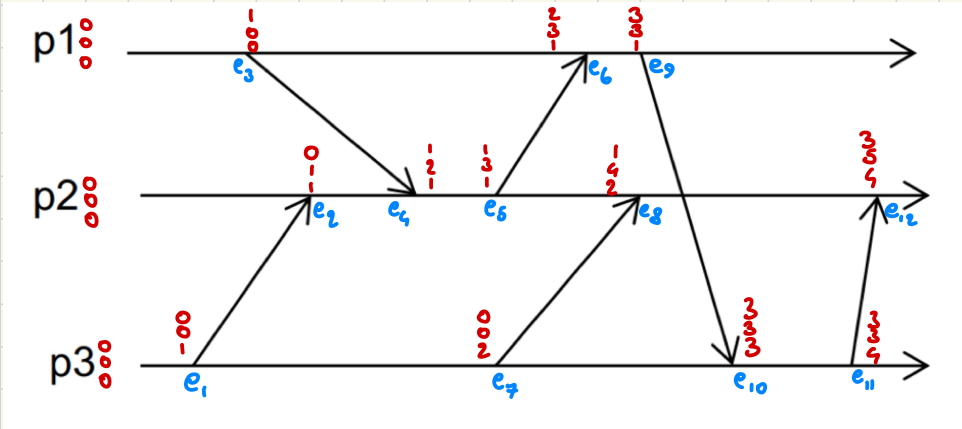
SCALAR LOGICAL CLOCKS TECHNIQUE: EACH PROCESS p_i INITIALIZES ITS LOGICAL CLOCK $L_i = 0$ AND INCREASE L_i OF 1 WHEN IT GENERATES AN EVENT ($L_i = L_i + 1$). IN PARTICULAR, WHEN p_i SENDS A MESSAGE m, INCREASES L_i AND TIMESTAMP m WITH $t_s = L_i$. INSTEAD, IF p_i RECEIVES A MESSAGE THE LOGICAL CLOCK IS UPDATED $L_i = max(t_s, L_i)$.

VECTOR CLOCKS TECHNIQUE: FOR A SET OF n PROCESSES, THE VECTOR CLOCK IS COMPOSED BY AN ARRAY OF n INTEGERS. SO EACH PROCESS p_i MAINTAIN A VECTOR CLOCK V_i AND TIMESTAMPS EVENTS BY MEAN OF ITS VECTOR CLOCK.

a)



b)



c) CONCURRENCY: ~~$e_2 \parallel e_7$, $e_1 \parallel e_3$, $e_8 \parallel e_6$, $e_2 \parallel e_3$, $e_8 \parallel e_{10}$~~

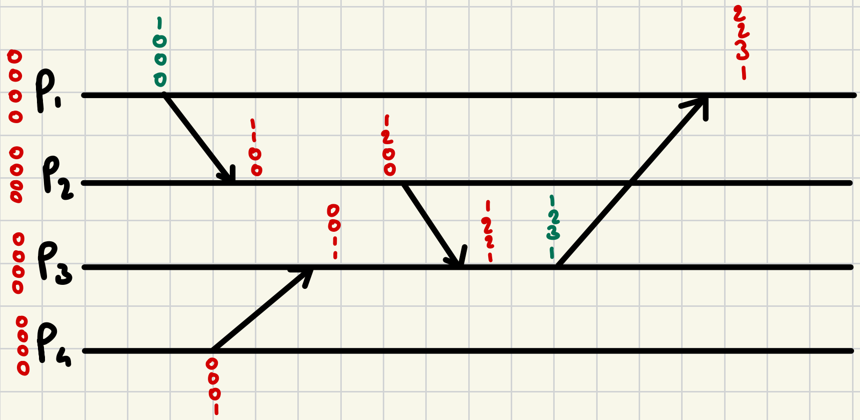
$e_1 \parallel e_3$, $e_2 \parallel e_3$, $e_3 \parallel e_7$, $e_5 \parallel e_7$, $e_6 \parallel e_7$, $e_7 \parallel e_9$, $e_3 \parallel e_{10}$, $e_8 \parallel e_{11}$, $e_8 \parallel e_9$, $e_6 \parallel e_8$

Exercise 4

Consider an asynchronous message passing system that uses vectors clock to implement some causal consistency check. The message passing system is composed by 4 processes with IDs 1 to 4, and, as usual, the ID is used as displacement in the vector clock (i.e., the locations are (p1, p2, p3, p4)). Vector clocks are updated increasing before send. Processes communicate by point2point links. You start debugging process p1 (the process with id 1) in the middle of the algorithm execution. You see the following stream of messages exiting and entering the ethernet card of process p1:

- `Time 00:00 - EXITING: Send Message [MSG CONTENT] Vector Clock: (1, 0, 0, 0)`
- `Time 00:05 - ENTERING: Rcvd Message [MSG CONTENT] Vector Clock: (1, 2, 3, 1)`

1. Draw an execution that justifies the vectors clocks you are seeing. Is such an execution unique?
2. There exists an execution that justifies the vector clocks and where there exists at least a process that does not send any message? Justify your answer.

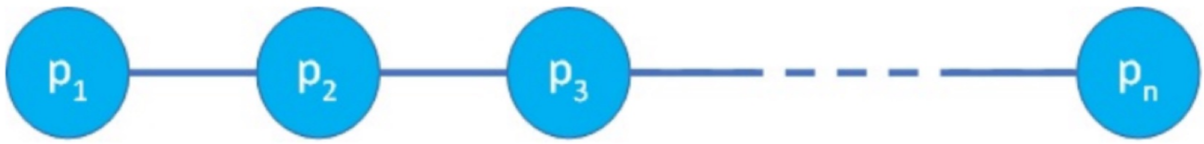


a) IT'S NOT UNIQUE BECAUSE WE CAN DESIGN ANOTHER EXECUTION

b) NO, BECAUSE P1 RECEIVED A MESSAGE THAT IS INCREASED AT LEAST ONE TIME BY EACH PROCESS.

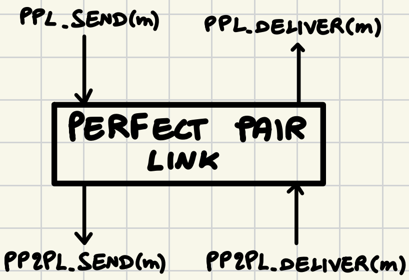
Exercise 5

Let us consider a distributed system composed of N processes p1, p2, … pn, each one having a unique integer identifier. Processes are arranged in line topology as in the following figure, with consecutive identifiers.



Let us assume that there are no failures in the system (i.e., processes are always correct) and that topology links are implemented through perfect point-to-point links.

Write the pseudo-code of a distributed algorithm that builds the abstraction of a perfect point-to-point link between any pair of processes (i.e., also between those that are not directly connected, such as p1 and p3).




```
INIT
  ID ← IDENTIFIER

UPON EVENT <PPL |SEND| DEST, m> DO
  IF (DEST > ID) THEN
    TRIGGER <PP2PL |SEND| ID+1 <ID, DEST, m>>
  ELSE IF (DEST = ID) THEN
    TRIGGER <PP2PL |SEND| ID <ID, DEST, m>>
  ELSE
    TRIGGER <PP2PL |SEND| ID-1 <ID, DEST, m>>

UPON EVENT <PP2PL |DELIVER| q <SOURCE, DEST, m>> DO
  IF (DEST > ID) THEN
    TRIGGER <PP2PL |SEND| ID+1 <SOURCE, DEST, m>>
  ELSE IF (DEST = ID) THEN
    TRIGGER <PPL |DELIVER| SOURCE, m>
  ELSE
    TRIGGER <PP2PL |SEND| ID-1 <SOURCE, DEST, m>>
```