Logical Time in Distributed Systems

Sistemi di Calcolo (II semestre) – Roberto Baldoni

Logical clock

- Physical clock synchronization algorithms try to coordinate distributed clocks to reach a common value
  - Based on the estimation of transmission times
    - It can be hard to find a good estimation.

  - In several applications it is not important when things happened but in which order they happened

- Reliable way of ordering events is required!

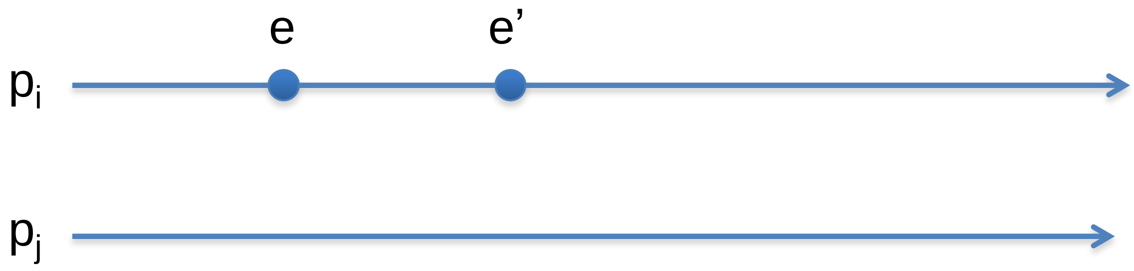
Notes:

1. Two events occurred at some process p_i happened in the same order as p_i observes them
2. When p_i sends a message to p_j the *send* event happens before the *receive* event

- Lamport introduced the relation that captures the causal dependencies between events (*causal order relation*)
  - We denote with $\rightarrow_i$ the ordering relation between events in a process p_i
  - We denote with $\rightarrow$ the happened-before relation between any pair of events

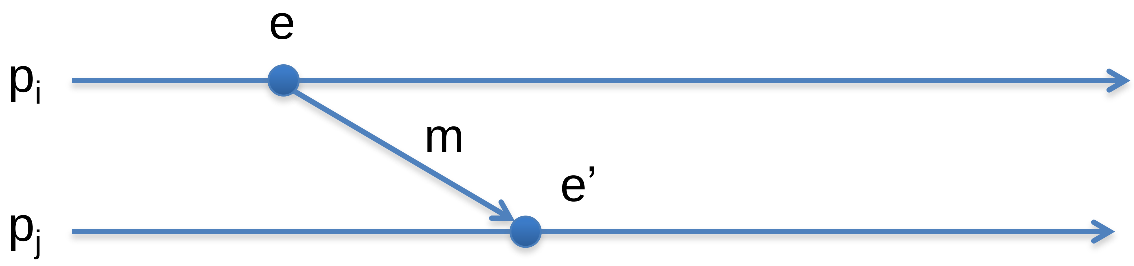
Happened-Before Relation: Definition

- Two events *e* and *e'* are related by happened-before relation ($e \rightarrow e'$) if:
  - $\exists\ p_i \mid e \rightarrow_i e'$



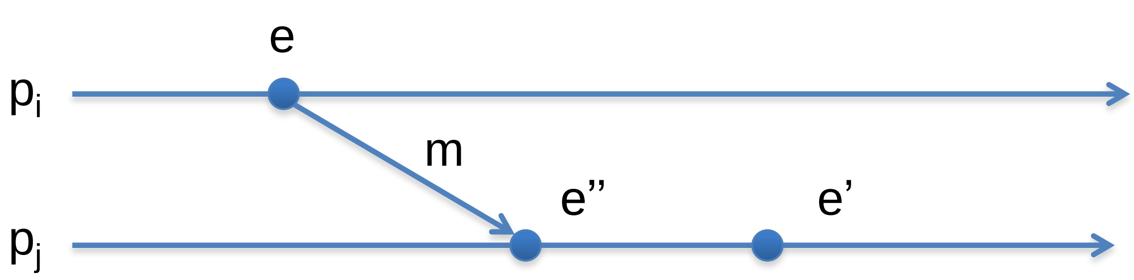
Happened-Before Relation: Definition

- Two events *e* and *e'* are related by happened-before relation ($e \rightarrow e'$) if:
  - $\exists\, p_i \mid e \rightarrow_i e'$
  - $\forall$ message m send(m) $\rightarrow$ receive(m)
    - send(m) is the event of sending a message m
    - receive(m) is the event of receipt of the same message m



Happened-Before Relation: Definition

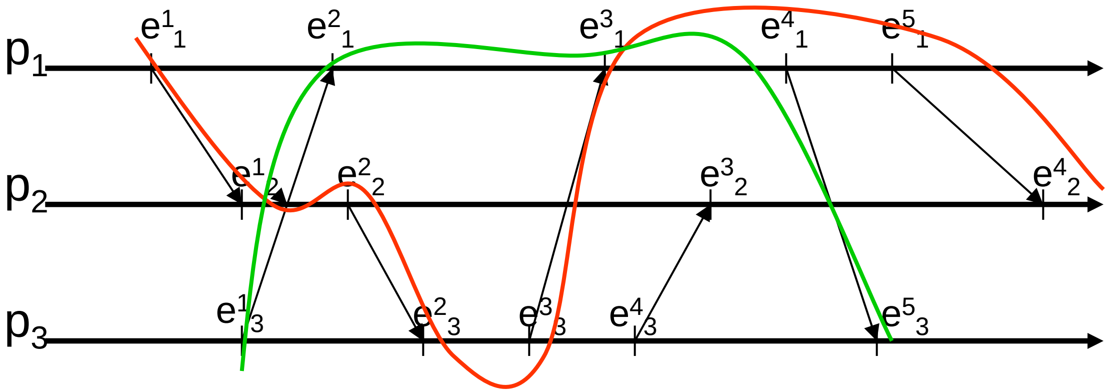
- Two events *e* and *e'* are related by happened-before relation ($e \rightarrow e'$) if:
  - $\exists\ p_i \mid e \rightarrow_i e'$
  - $\forall$ message m send(m) $\rightarrow$ receive(m)
    - send(m) is the event of sending a message m
    - receive(m) is the event of receipt of the same message m
  - $\exists\ e, e', e'' \mid (e \rightarrow e'') \wedge (e'' \rightarrow e')$
    (happened-before relation is transitive)



Happened-Before Relation

- Using these three rules it is possible to define a causal-ordered sequence of events e_1, e_2, … , e_n
- **Notes**:
  - The sequence e_1, e_2, …, e_n may not be unique
  - It may exists a pair of events $<e_1,e_2>$ such that e_1 and e_2 are not in happened-before relation
  - If e_1 and e_2 are not in happened-before relation then they are *concurrent* ($e_1||e_2$)
  - For any two events e_1 and e_2 in a distributed system, either
    - $e_1 \rightarrow e_2$
    - $e_2 \rightarrow e_1$
    - $e_1||e_2$

happened-before: example



e^j_i is j-th event of process p_i

$S_1 = < e^1_1, e^1_2, e^2_2, e^2_3, e^3_3, e^3_1, e^4_1, e^5_1, e^4_2 >$

$S_2 = < e^1_3, e^2_1, e^3_1, e^4_1, e^5_3 >$

Note: e^1_3 and e^1_2 are concurrent

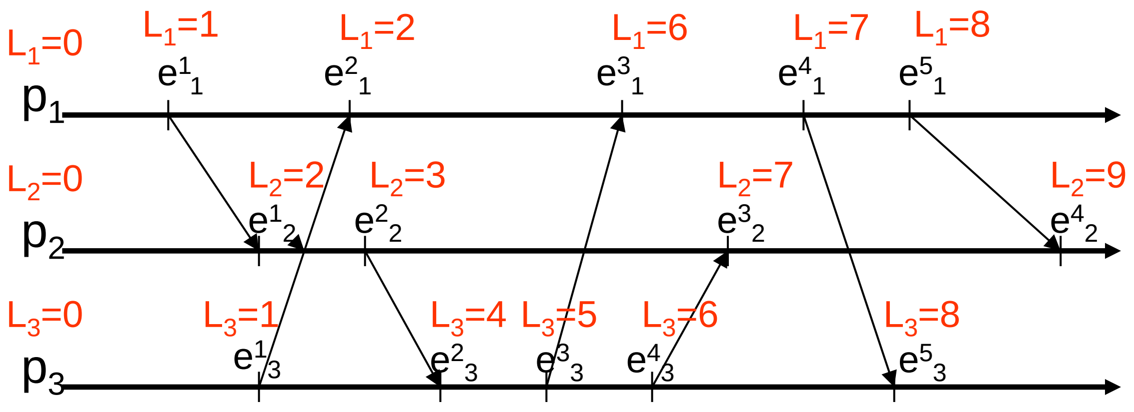
Logical Clock

- The Logical Clock, introduced by Lamport, is a software counting register *monotonically* increasing its value
  - Logical clock is not related to physical clock
- Each process p_i employs its logical clock L_i to apply a *timestamp* to events
- $L_i(e)$ is the "logical" timestamp assigned, using the logical clock, by a process p_i to event e
- **Property**:
  - If $e \rightarrow e'$ then $L(e) < L(e')$
- **Observation**:
  - The ordering relation obtained through logical timestamps is only a partial order. Consequently, timestamps could not be sufficient to relate two events

Scalar Logical Clock: an implementation

- Each process p_i initializes its logical clock $L_i=0$ ($\forall$ i = 1….N)
- p_i increases L_i of 1 when it generates an event (either *send* or *receive*)
  - $L_i = L_i + 1$
- When p_i sends a message *m*
  - creates an event *send(m)*
  - increases L_i
  - timestamps *m* with $t = L_i$
- When p_i receives a message *m* with timestamp *t*
  - Updates its logical clock $L_i = \max(t, L_i)$
  - Produces an event *receive(m)*
  - Increases L_i

Scalar Logical Clock: example



- e^j_i is j-th event of process pi
- L_i is the logical clock of p_i
- Note:
  - $e^1_1 \rightarrow e^2_1$ and timestamps reflect this property
  - $e^1_1 \| e^1_3$ and respective timestamps have the same value
  - $e^1_2 \| e^1_3$ but respective timestamps have different values

Limits of Scalar Logical Clock

- Scalar logical clock can guarantee the following property
  - IF $e \rightarrow e'$ then $L(e) < L(e')$
- But it is not possible to guarantee
  - IF $L(e) < L(e')$ then $e \rightarrow e'$
- **Consequently**:
  - It is not possible to determine, by analysing only scalar clocks, if two events are concurrent or correlated by the happened-before relation
- Mattern [1989] and Fridge [1991] proposed an improved version of logical clock where events are time-stamped with local logical clock and node identifier
  - ***Vector Clock***

Logical Time and Ricart-Agrawala Mutual Exclusion Algorithm

Logical clock in distributed algorithms

<u>Scalar Clock</u> can be used to solve
Lamport’s Mutual Exclusion problem
in a distributed setting

Ricart-Agrawala's algorithm: implementation (see also lecture notes)

- Local variables
  - #replies (*initially* 0)
  - State $\in$ {Requesting, CS, NCS} (*initially* NCS)
  - Q pending requests queue (*initially* empty)
  - Last_Req (*initially* MAX_INT)
  - Num (*initially* 0)
- Algorithm:

begin

1. State = Requesting
2. Num = Num + 1; Last_Req = num
3. $\forall$i = 1…N, send REQUEST to p_i
4. Wait until #replies == N - 1
5. State = CS
6. CS
7. $\forall r \in Q$, send REPLY to r
8. Q = $\emptyset$; State = NCS; #replies = 0; Last_Req = MAX_INT

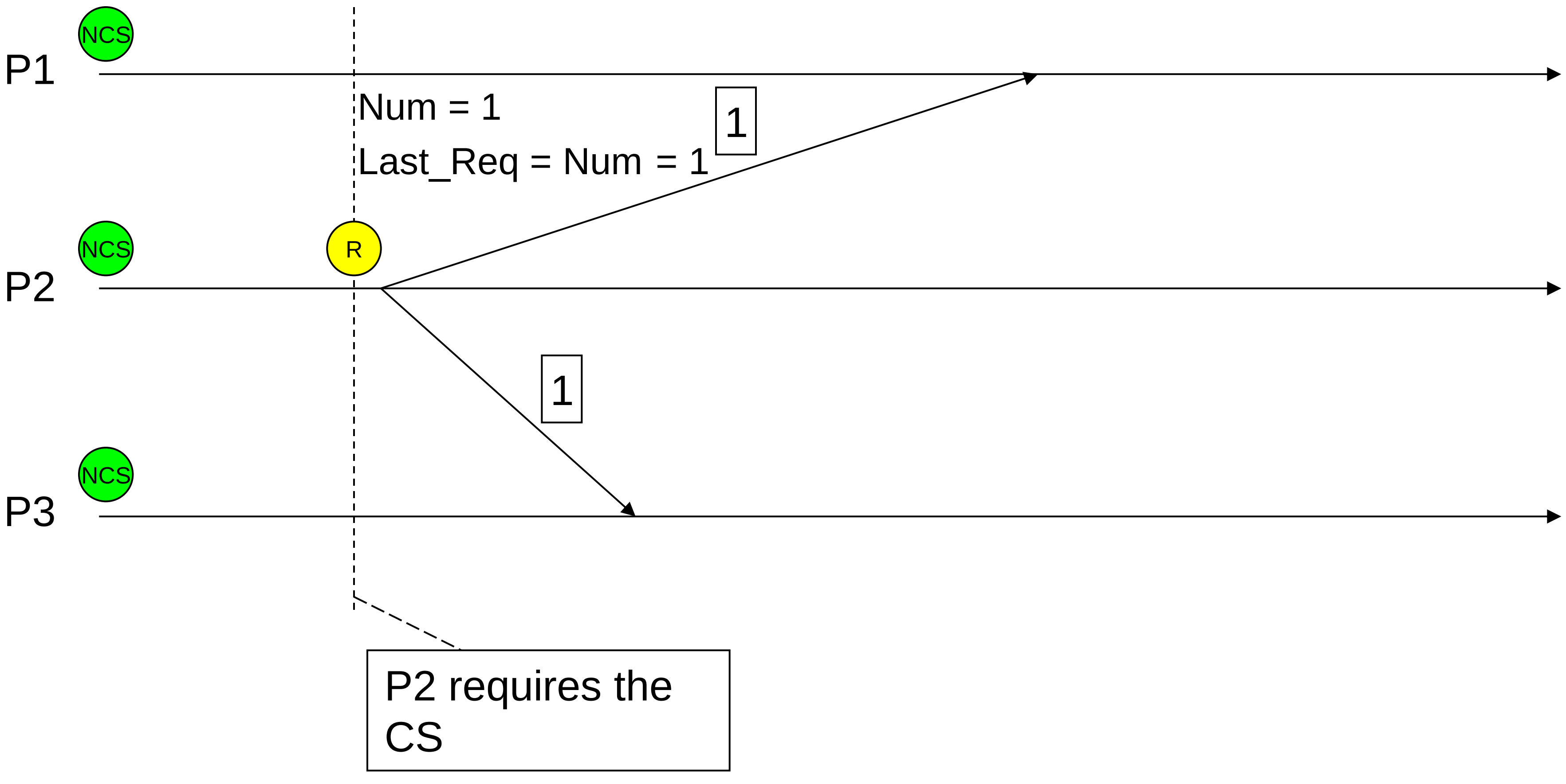
Upon receipt REQUEST(t) from p_j

1. Num = max(t, Num)
2. If State == CS or (State == Requesting and {Last_Req,i} < {t,j})
3. Then insert in Q{t, j}
4. Else send REPLY to p_j

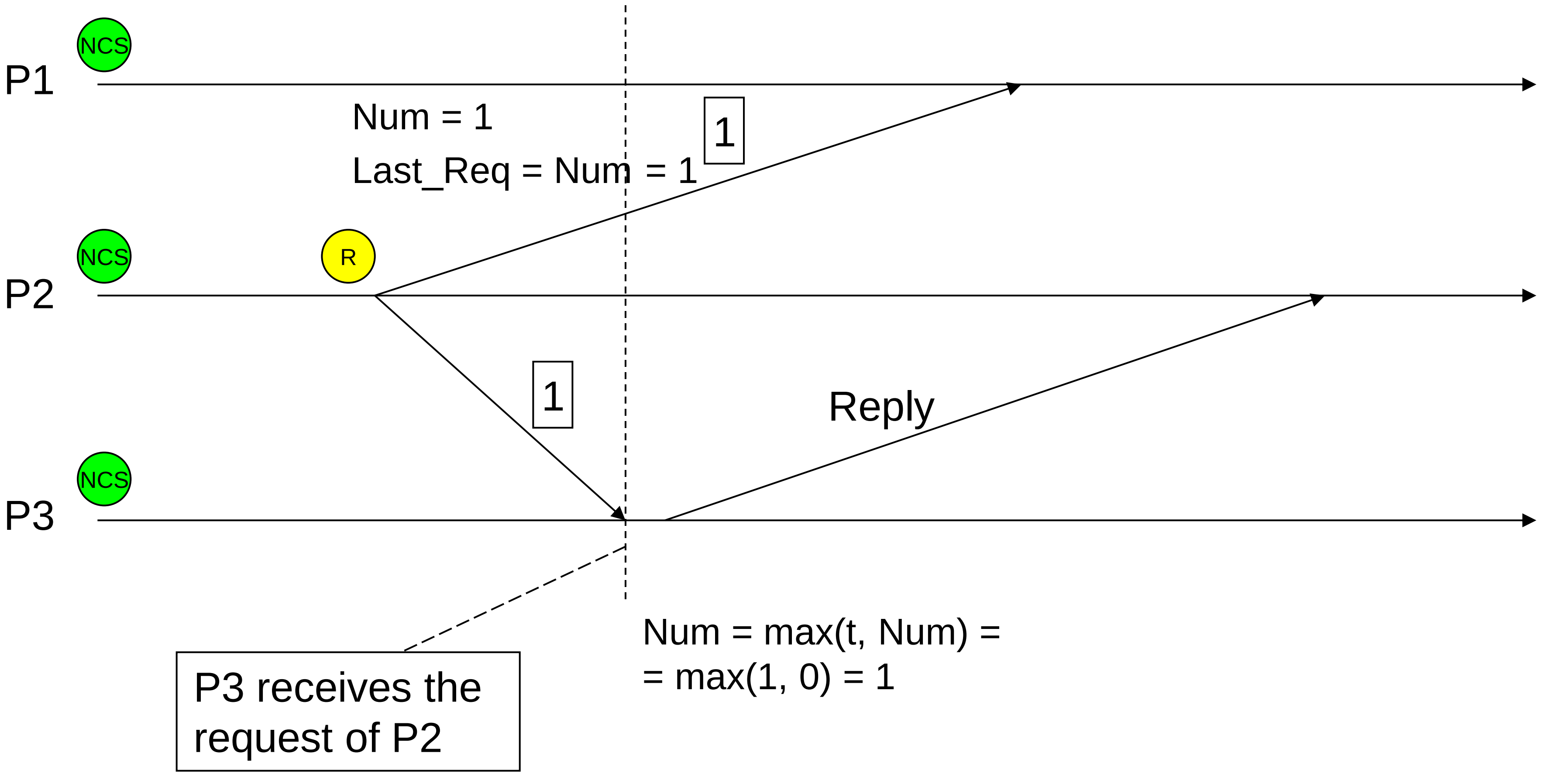
Upon receipt of REPLY from p_j

1. #replies = #replies + 1

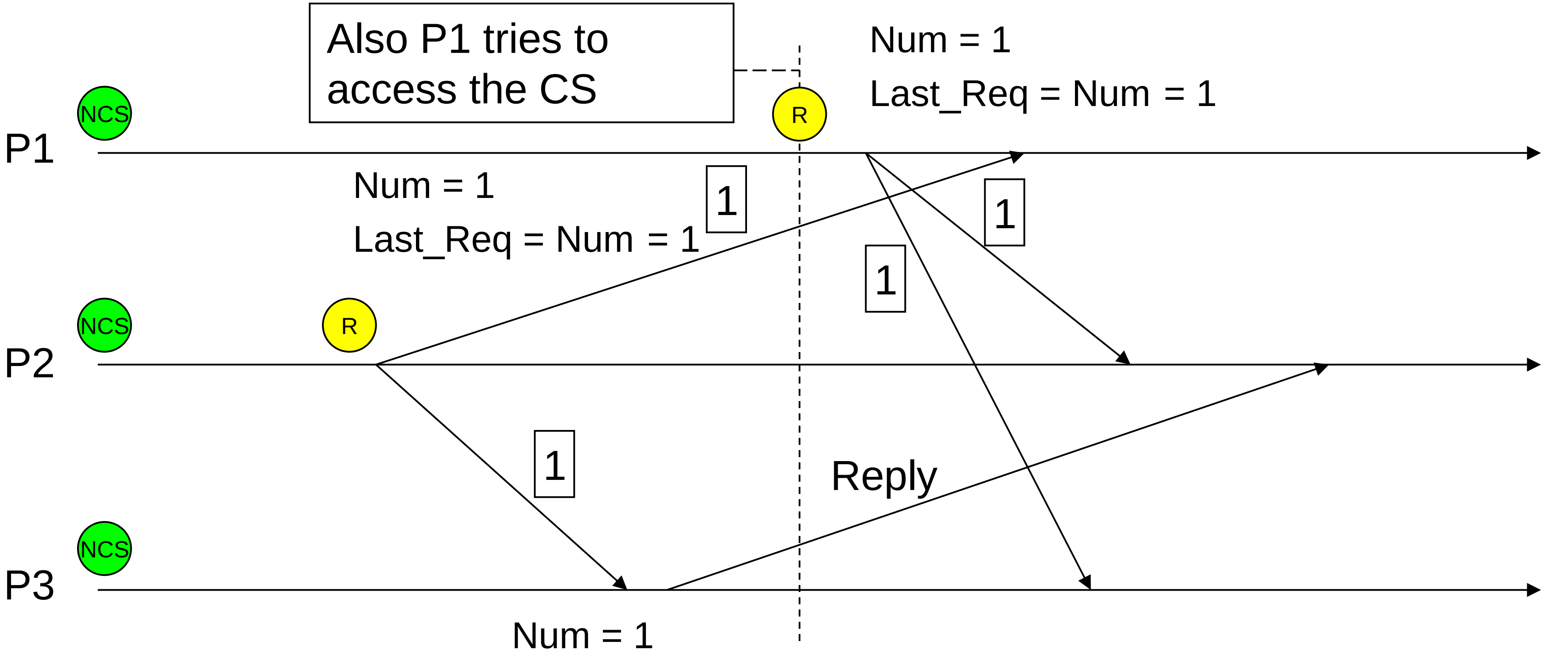
Ricart-Agrawala's algorithm: example



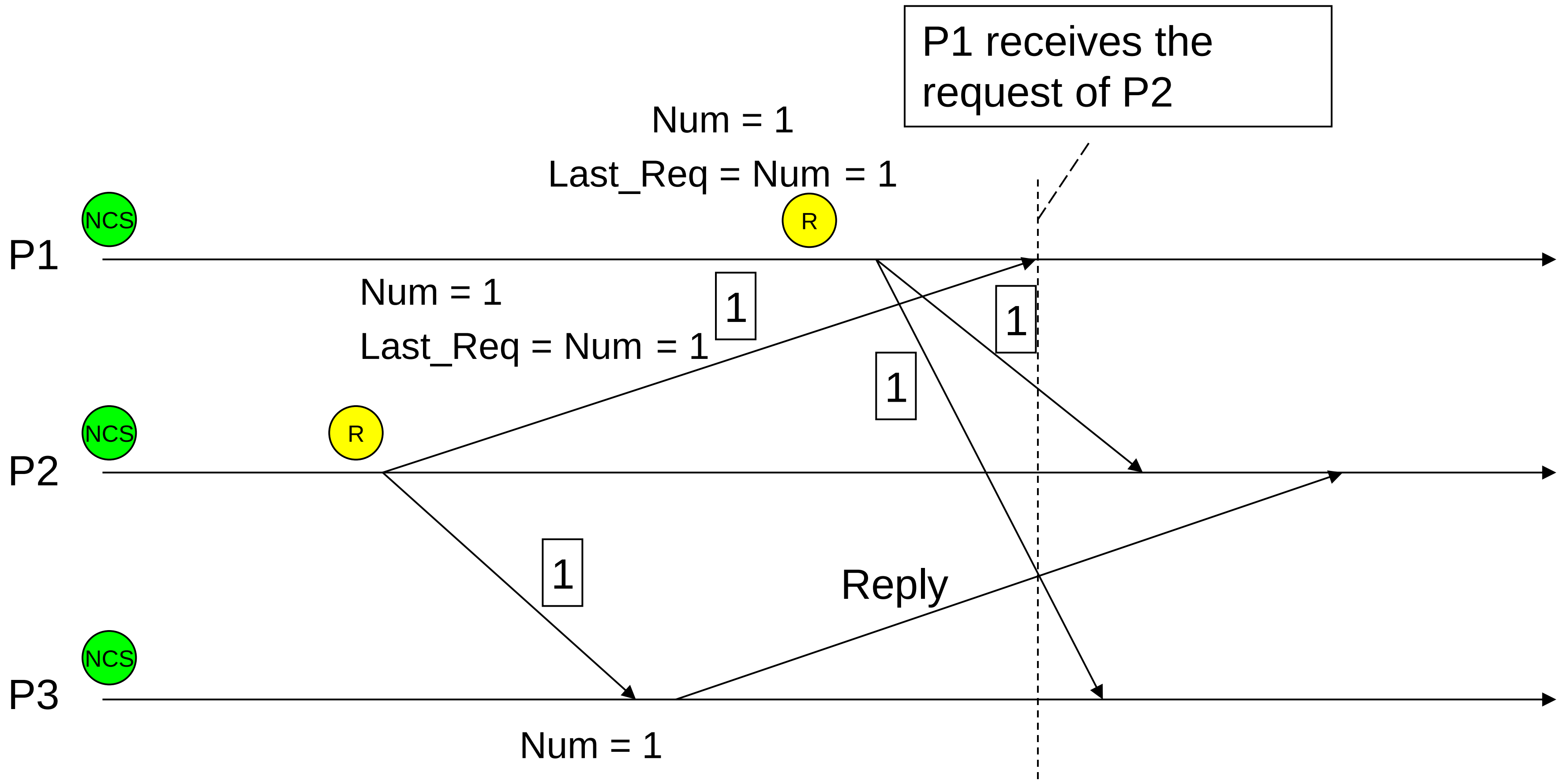
Ricart-Agrawala's algorithm: example



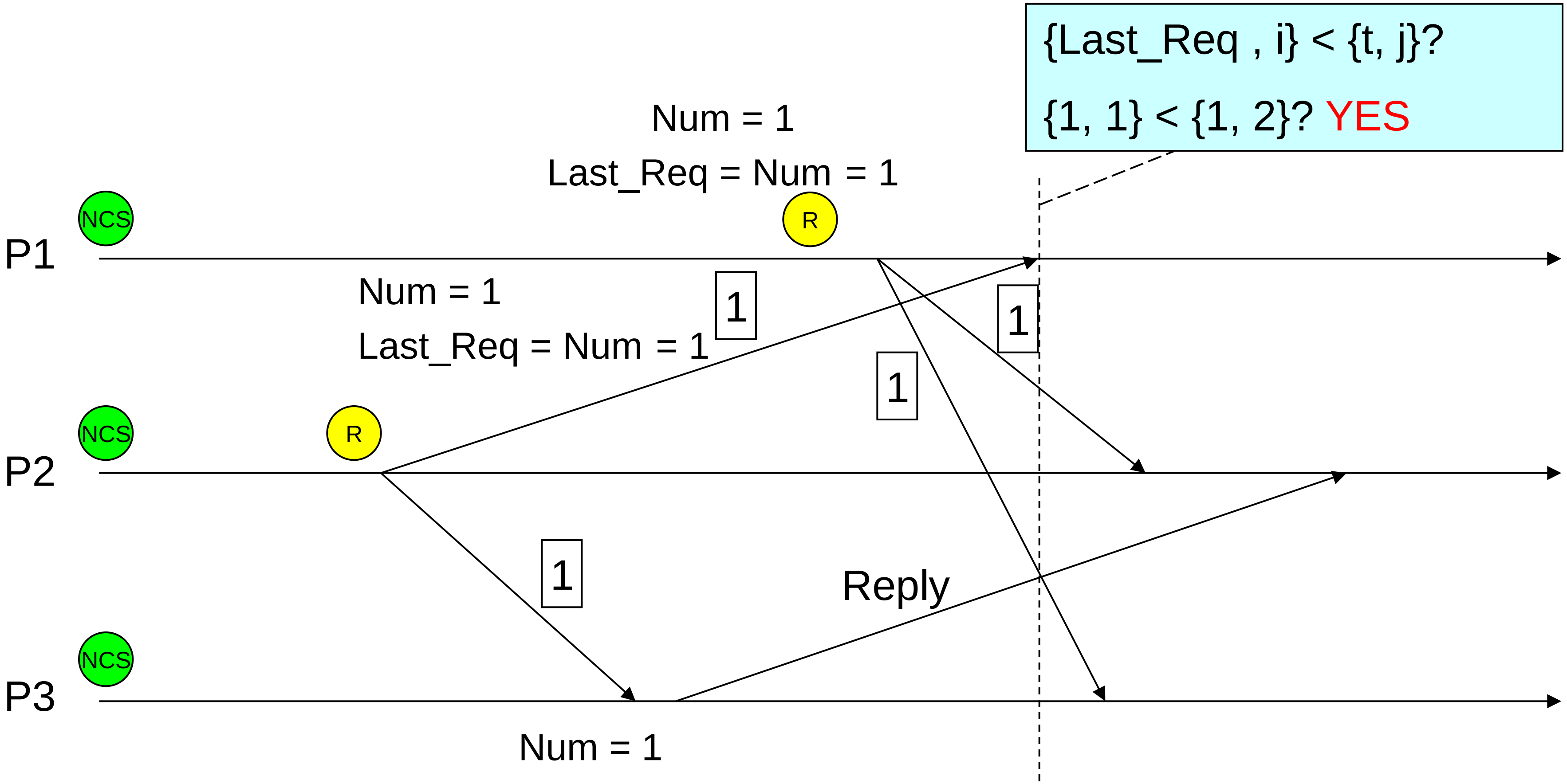
Ricart-Agrawala's algorithm: example



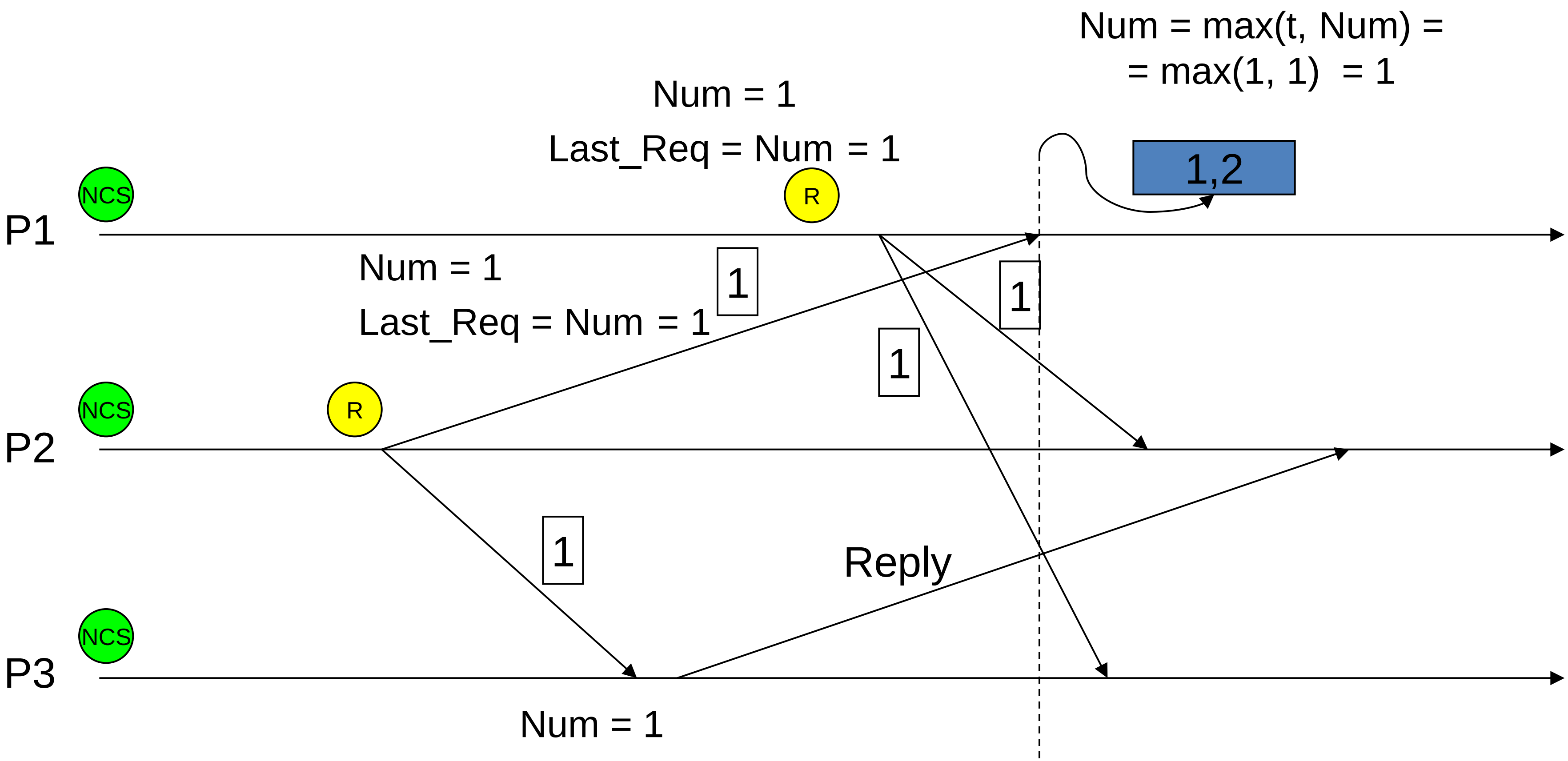
Ricart-Agrawala's algorithm: example



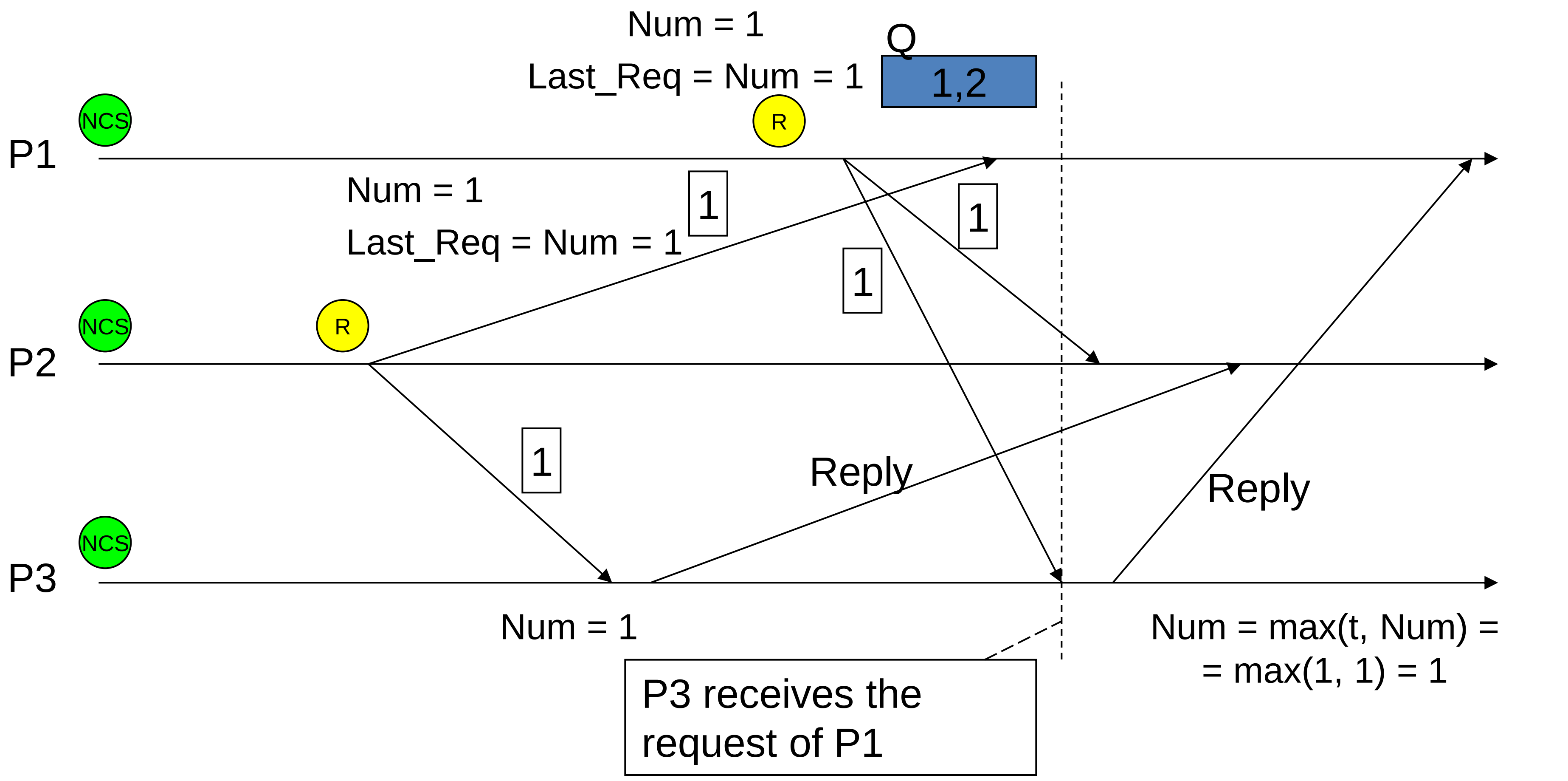
Ricart-Agrawala's algorithm: example



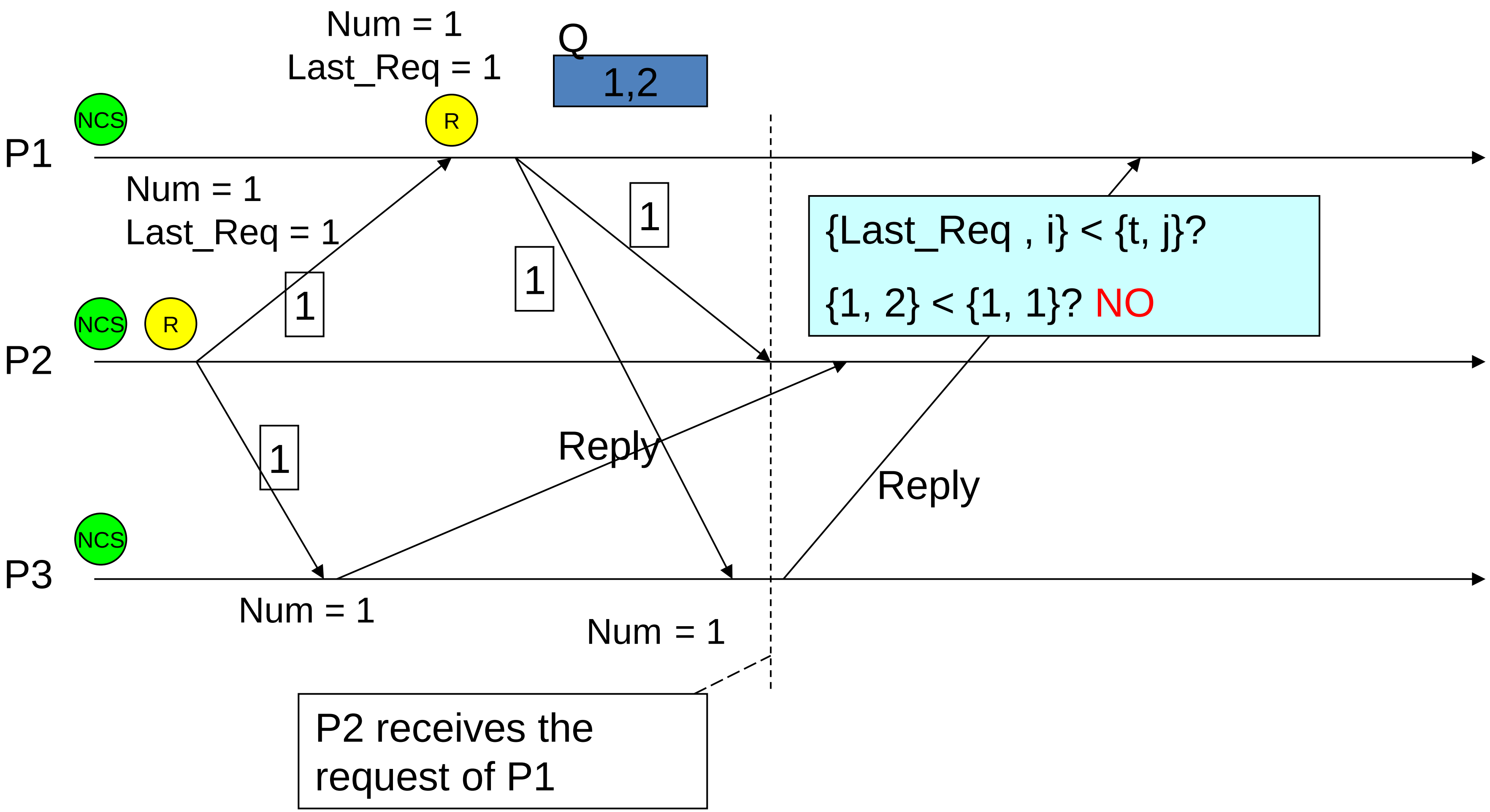
Ricart-Agrawala's algorithm: example



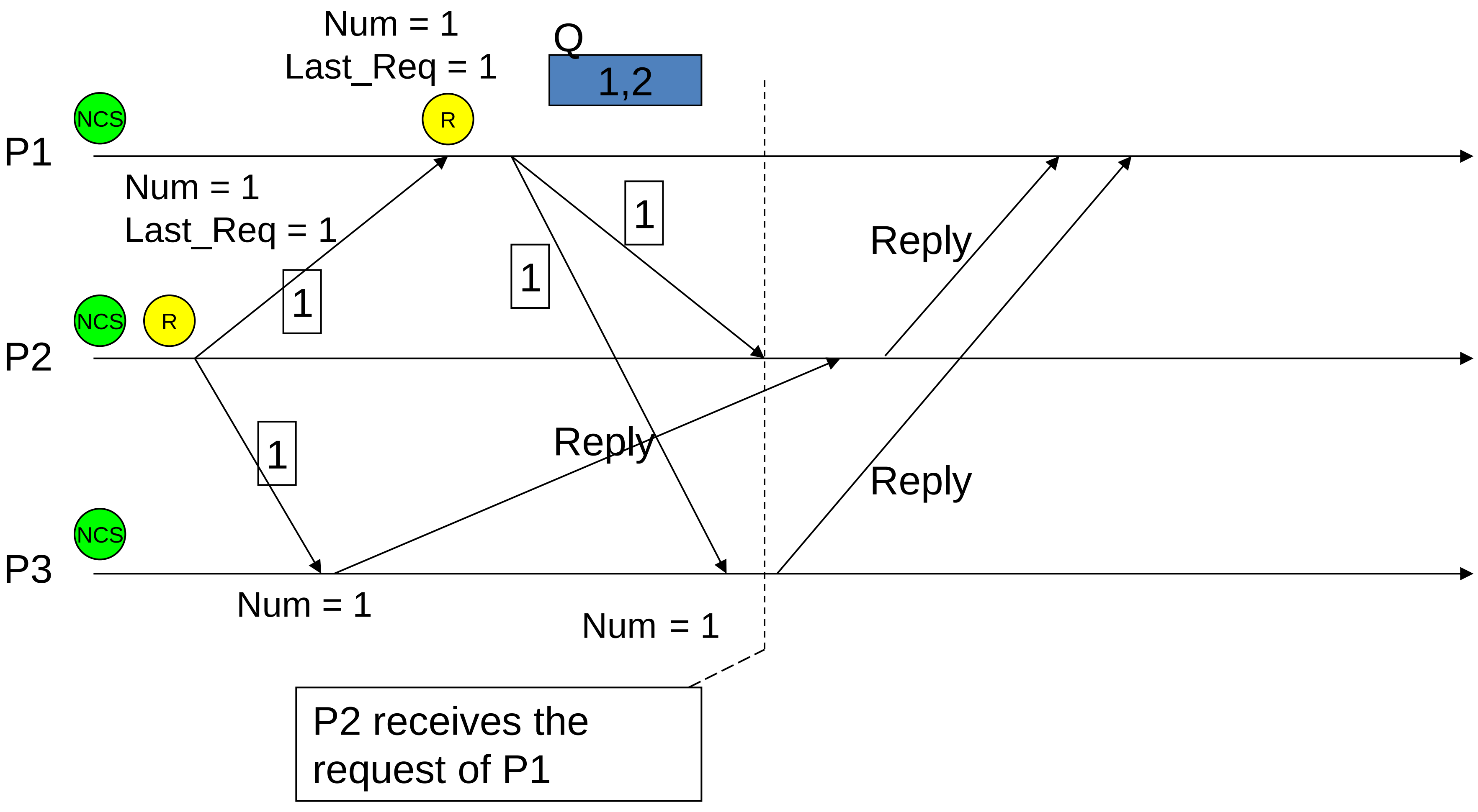
Ricart-Agrawala's algorithm: example



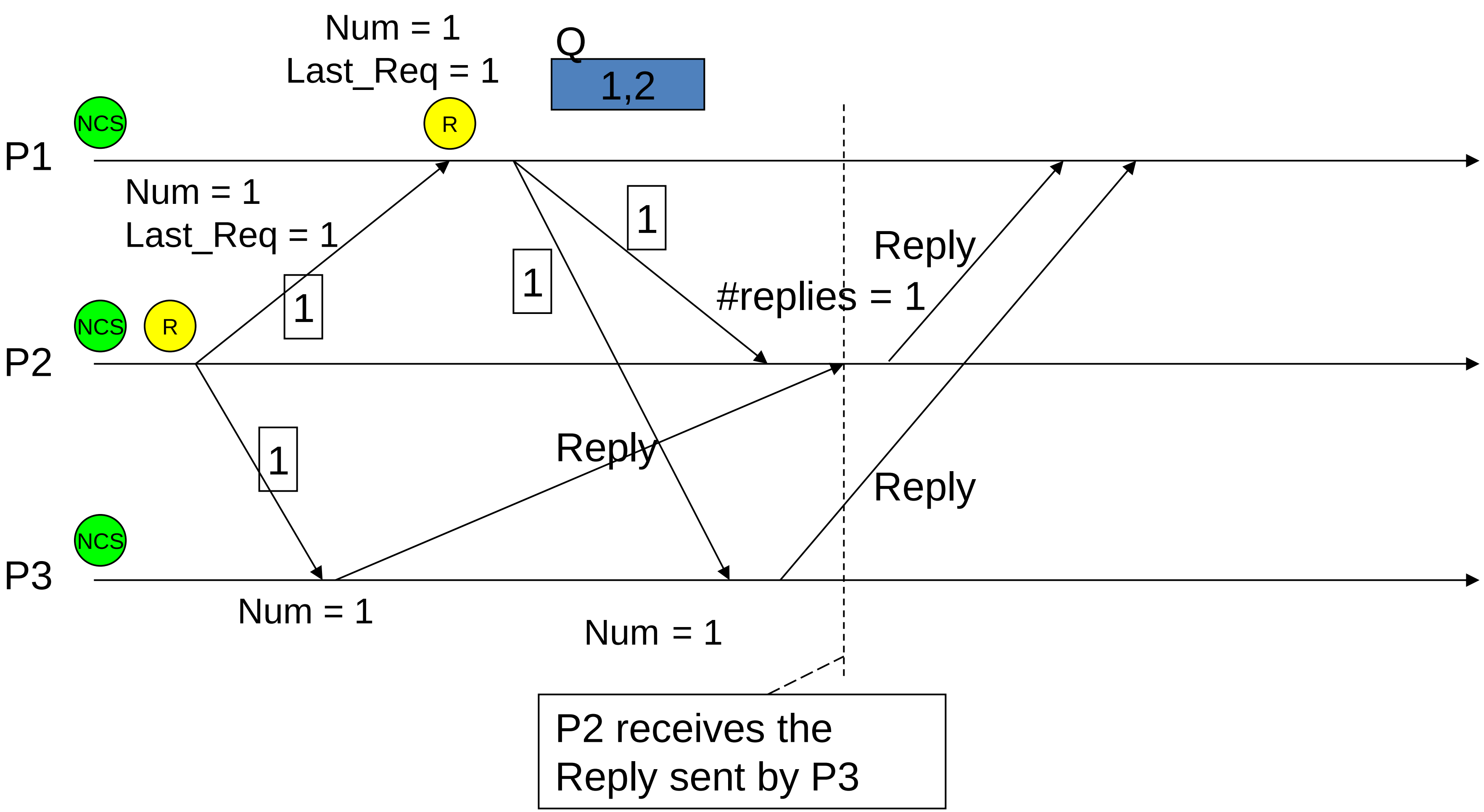
Ricart-Agrawala's algorithm: example



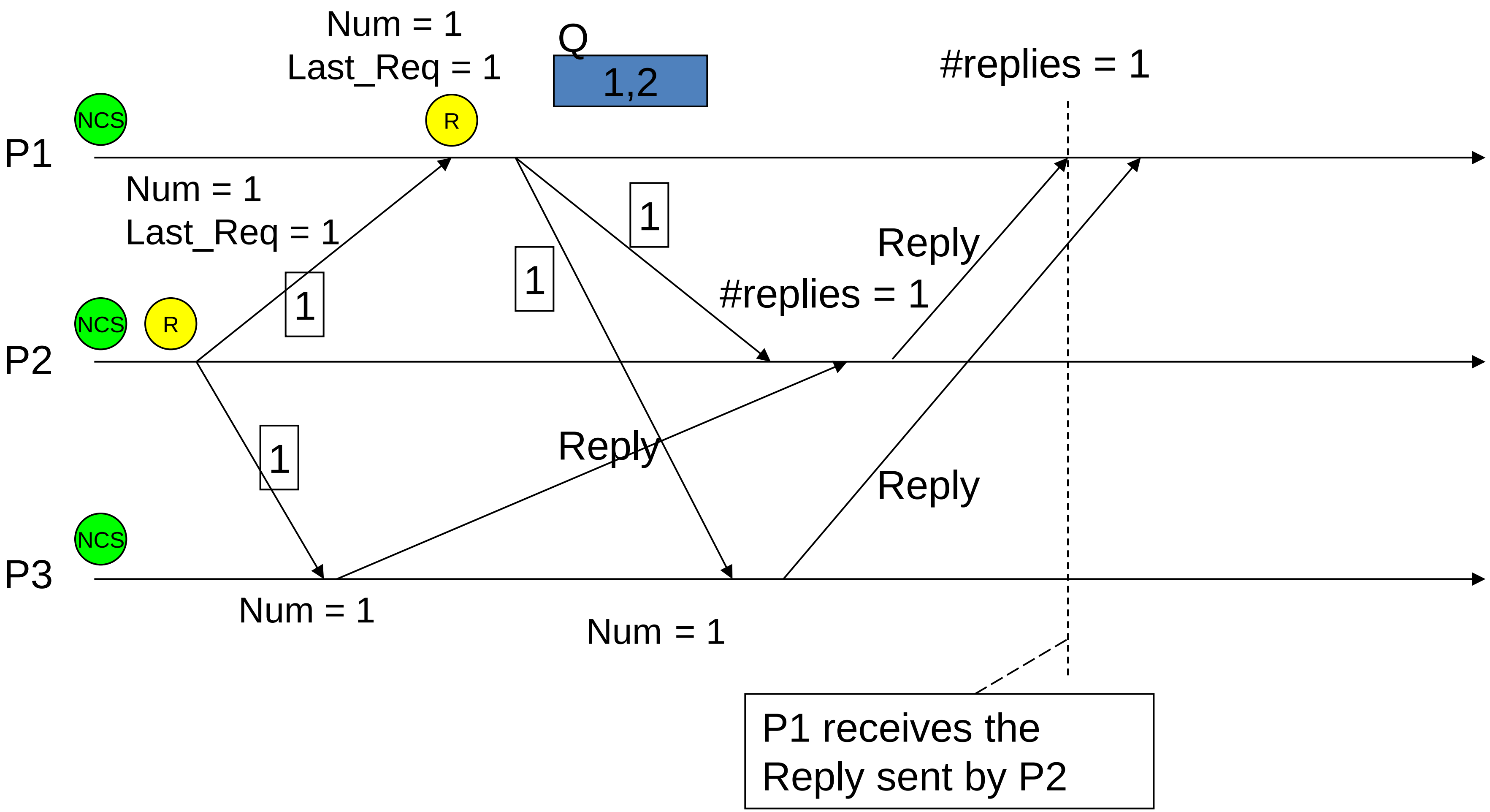
Ricart-Agrawala's algorithm: example



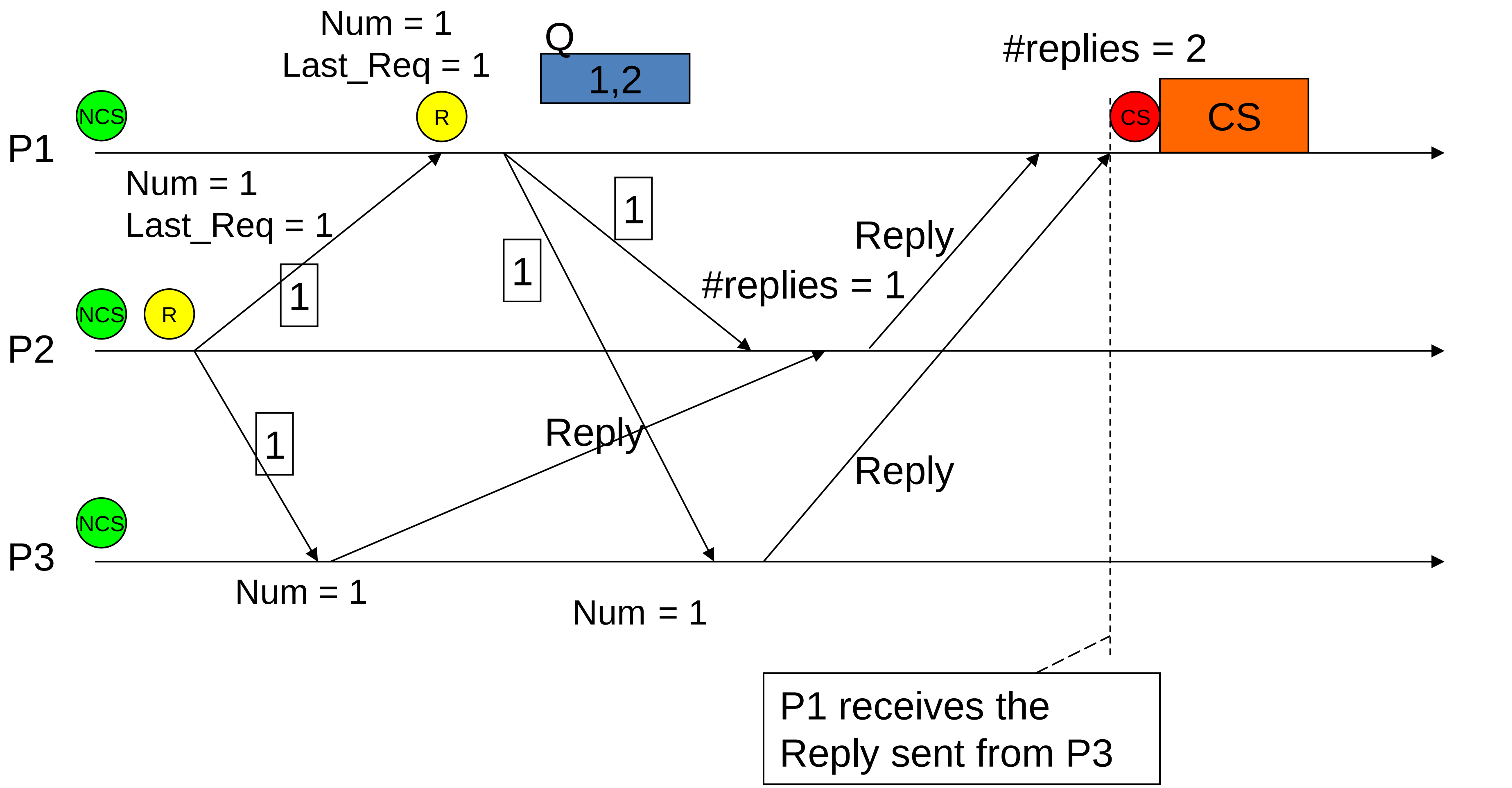
Ricart-Agrawala's algorithm: example



Ricart-Agrawala's algorithm: example



Ricart-Agrawala's algorithm: example



Ricart-Agrawala's algorithm: example