



Sistemas Distribuidos I (75.74)

Algoritmos de Consenso

Elección de Líder. Consenso. Generales Bizantinos.

Docentes

- Pablo D. Roca
- Ezequiel Torres Feyuk



- **Elección de Líder**
- Consenso
- Generales Bizantinos
- Paxos



- **Objetivo**

- Elegir a un proceso en un grupo para que desempeñe un rol particular
- Permitir reelecciones en caso de que proceso líder decida darse de baja
- Permitir reelecciones en caso de que proceso líder se encuentre caído

- **Características**

- Cualquier proceso puede comenzar una nueva elección de líder
- El resultado de la elección de un nuevo líder debe ser **única y repetible**



Elección de Líder | Propiedades

- Cada proceso debe tener un identificador único
- Todos los procesos poseen un array que indica el estado del algoritmo de elección de Líder
- Estados posibles: *Identificador (P)*, *indefinido (@)*
 - *Indefinido (@)* es el estado inicial que posee un proceso P_i al comenzar a participar del algoritmo de elección de líder
- **Safety**
 - Un proceso participante P_i posee el estado $elected_i = P$ o $elected_i = @$
- **Liveness**
 - Todos los procesos participan de la elección de líder y bien terminan con un estado $elected_i \neq @$ o mueren



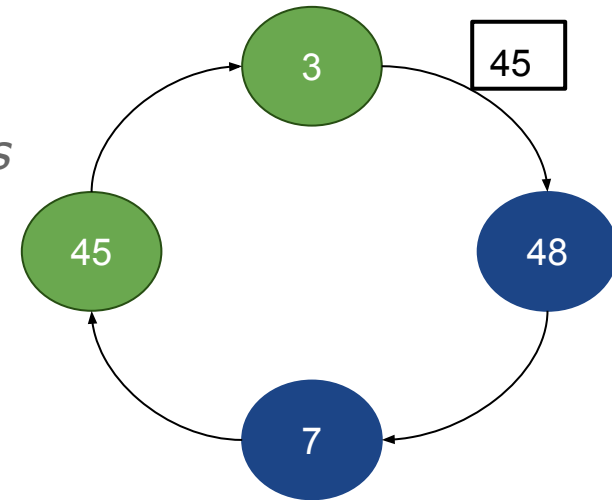
Elección de Líder | Algoritmo del anillo

- **Condiciones Iniciales**

- Cada proceso se comunica solamente con su vecino
- Mensajes son enviados siempre en la misma dirección (e.g. sentido horario)
- Al comienzo del algoritmo, todos los procesos son marcados como *no participantes*

- **Inicio de la elección**

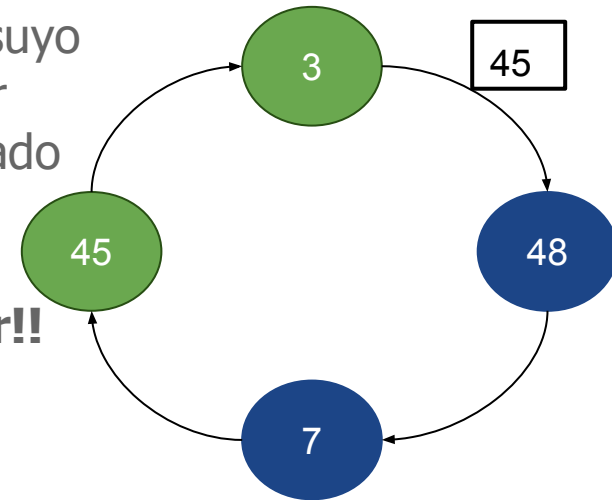
- Algún proceso P_i se marca como *participando* y envía un mensaje en sentido horario indicando que él es el líder





Elección de Líder | Algoritmo del anillo

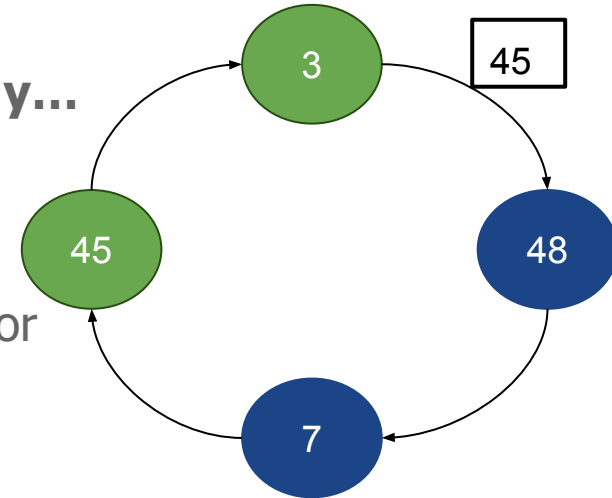
- **Proceso P_j recibe mensaje de elección de líder y...**
 - Reenvía el mensaje si se encuentra en estado *no participando*
 - Cambia su estado a *participando*
 - Compara el identificador del líder con el suyo y lo reemplaza en caso de que sea mayor
 - No reenvía el mensaje si se encuentra en estado *participando*
 - Si el identificador recibido es igual al Suyo, entonces **el proceso P_j es el líder!!**





Elección de Líder | Algoritmo del anillo

- **Proceso P_j reconoce que es el líder y...**
 - Se setea como *no participando*
 - Envía un mensaje de **líder elegido**
- **Proceso P_j recibe un mensaje de líder elegido y...**
 - Cambia su estado a *no participando*
 - Setea la variable elegido con el identificador del mensaje recibido
 - Retransmite el mensaje siempre el identificador Recibido sea distinto al suyo





- **Hipótesis**
 - Canales de comunicación *reliables*
 - Cualquier proceso puede morir de forma inesperada
 - Uso de timeouts para detectar que un proceso no está respondiendo
 - Todos los procesos pueden comunicarse entre sí
 - Cada proceso conoce el ID asociado a todos los procesos
 - Cada proceso conoce qué procesos **poseen un ID superior al suyo**



Elección de Líder | Bully Algorithm

- **Tipos de Mensajes**

- Election Message: Mensaje enviado para iniciar una elección de líder
- Answer Message: ACK de Election Messages
- Coordinator Message: Contiene información sobre quién fue el proceso líder elegido

- **Sincronismo**

- T_{max}: Tiempo máximo de transmisión
- T_{process}: Tiempo máximo que proceso tarda en procesar un mensaje
- **$T = 2 * T_{max} + T_{process}$** (timeout para detectar procesos caídos)



Elección de Líder | Bully Algorithm

- **Algoritmo (Parte I)**

- Proceso con mayor ID puede identificarse como leader y mandar un *Coordinator Message* a todos los procesos del sistema
- Si un proceso detecta que el líder está caído, envía *Election Messages* a procesos que tengan **un ID mayor al suyo**
- Si un proceso recibe un *Election Message*, responde con un *Answer Message* y **comienza una nueva elección**
- Si un proceso recibe un *Coordinator Message*, elige al proceso que envió el mensaje como líder
- Si un proceso que comenzó una elección no recibe *Answer Messages* después de transcurrido tiempo T , **el proceso se autoproclama líder**



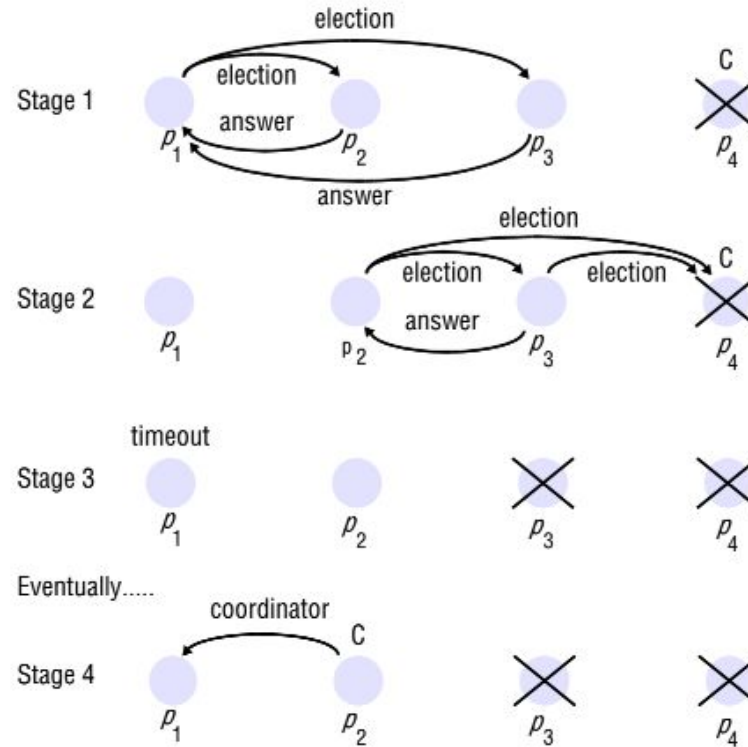
Elección de Líder | Bully Algorithm

- **Algoritmo (Parte II)**

- Si un proceso recibe un *Coordinator Message*, elige al proceso que envió el mensaje como líder
- Si un proceso que comenzó una elección no recibe *Answer Messages* después de transcurrido tiempo T , **el proceso se autoproclama líder**
- Si un proceso caído vuelve a la vida, **comienza una nueva elección de líder**
 - Si este proceso es el que posee mayor ID, será elegido como el nuevo líder
 - Por esta razón este algoritmo es llamado **Bully Algorithm**



Elección de Líder | Bully Algorithm



Agenda



- ☐ Elección de Líder
- ☒ **Consenso**
- ☐ Generales Bizantinos
- ☐ Paxos



- Dado un conjunto de procesos distribuidos y un punto de decisión, todos los procesos deben acordar en el mismo valor
- **Problema complejo, require acotar variables:**
 - Canales de comunicación son *reliables*
 - Todos los procesos pueden comunicarse entre sí
 - Única falla a considerar es la caída de un proceso
 - Caída de un proceso no puede ocasionar la caída de otro
- Propiedades necesarias de los algoritmos de consenso:
 - *Agreement*
 - *Integrity*
 - *Termination*



Consenso | Definición y Requerimientos del Problema

- **Definición**

- Conjuntos de P_i ($i = 1 \dots N$) procesos desean llegar a un acuerdo
- Cada proceso comienza en el estado ***undecided***
- Cada proceso posee una ***decision variable*** d_i ($i = 1 \dots N$)
- Cada proceso propone un valor v_i
- Procesos se comunican entre sí a través de mensajes
- Luego de haber recibido mensajes de otros procesos, proceso P_i setea su ***decision variable*** d_i y cambia su estado a ***decided***

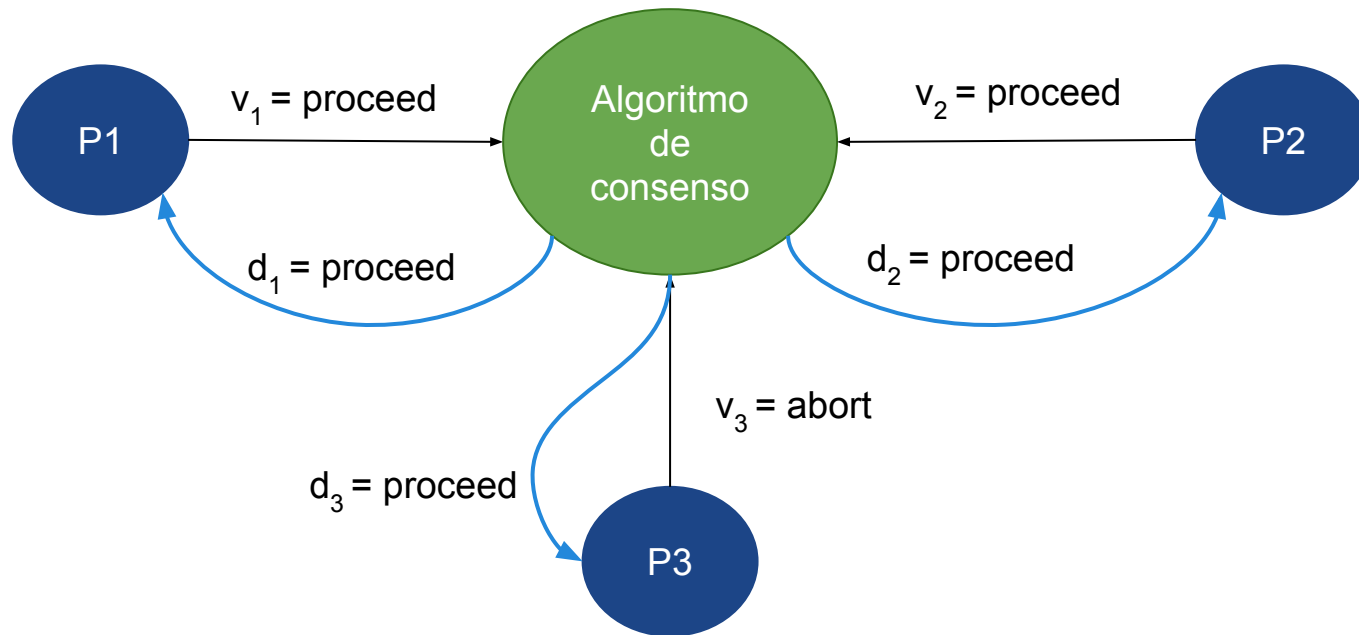


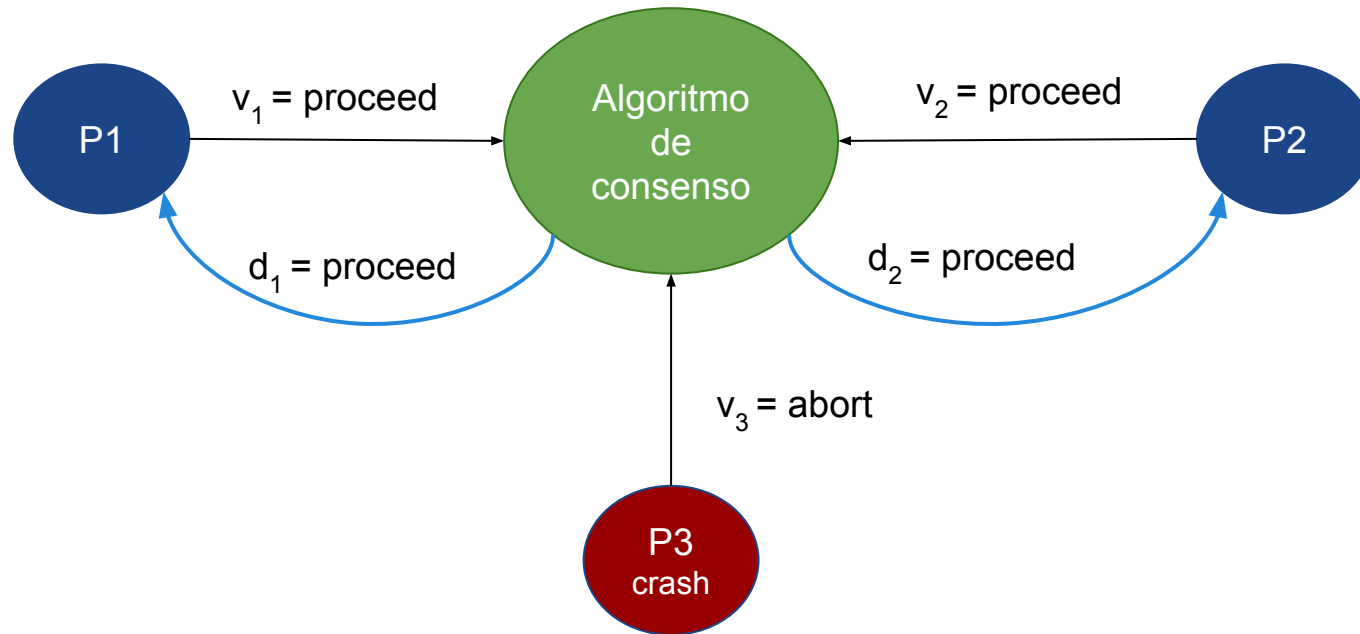
- **Requerimientos**

- *Agreement*
 - El valor de la variable ***decided*** es el mismo en todos los procesos correctos
 - Si P_i y P_j son procesos correctos entonces $d_i = d_j$ cuando su estado es ***decided***
- *Integrity*
 - Si los procesos correctos propusieron el mismo valor v_i , entonces el valor de su ***decision variable*** es la misma
- *Termination*
 - Eventualmente todos los procesos activos setean su ***decision variable***



Consenso | Definición y Requerimientos del Problema







Consenso | Algoritmo sincrónico

Init

Values(i , 0) = {}, Values(i , 1) = { v }

For each round r , $1 < r \leq f+1$

 broadcast Values(i , r) - Values(i , $r-1$)

 Values(i , $r+1$) = Values(i , r)

 While round r still open

 receive Values from j in Values(j , r)

 Values(i , $r + 1$) = Values(i , $r + 1$) $\cup$ Values(j , r)

After $f+1$ rounds

 decide d = aggregation over Values(i , $f+1$)

Agenda



- ☐ Elección de Líder
- ☐ Consenso
- ☒ **Generales Bizantinos**
- ☐ Paxos



Generales Bizantinos | Definición y *Requerimientos*

- **Definición**

- Tres o más **generales** deben decidir si atacan o se retiran
- Un **comandante** envía la orden de ataque/retirada
- **Generales** pueden ser *traicioneros*
 - Le indica a los otros generales que deben atacar si el comandante le indicó que se retiren y viceversa
- **Comandante** puede ser *traicionero*
 - Envía diferentes órdenes a diferentes generales
- **Comandante / General** *traicionero* viene a emular a un proceso que posee fallas



- **Requerimientos**

- *Agreement*

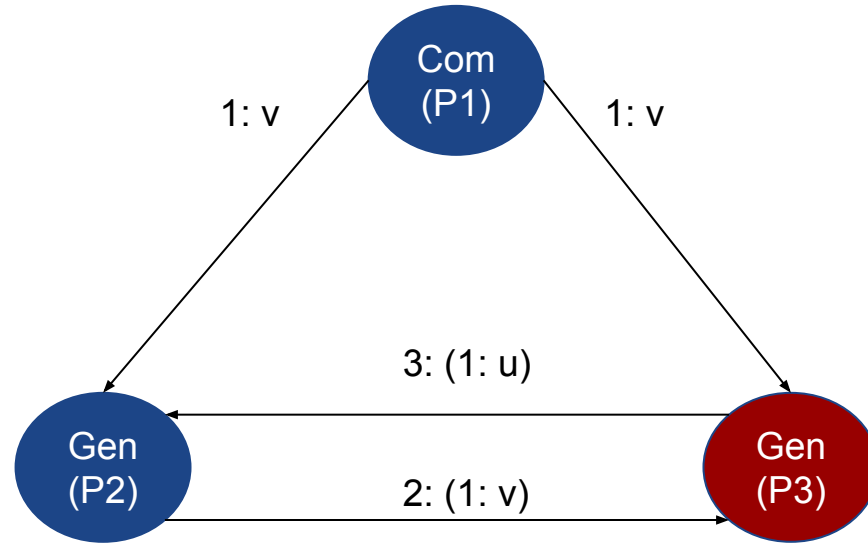
- El valor de la variable ***decided*** es el mismo en todos los procesos activos
 - Si P_i y P_j son procesos activos entonces $d_i = d_j$ cuando su estado es ***decided***

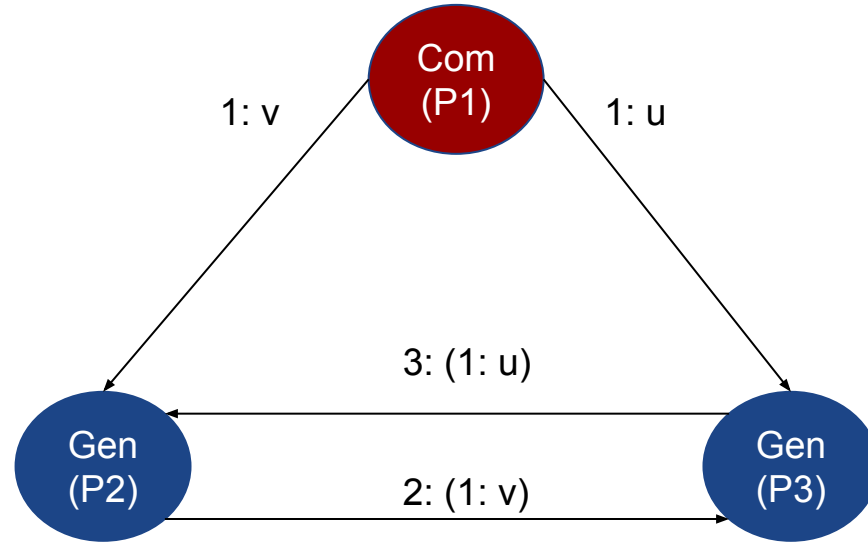
- *Integrity*

- Si el comandante *no es traicionero*, todos los procesos deben setear su ***decision variable*** al valor enviado por el comandante

- *Termination*

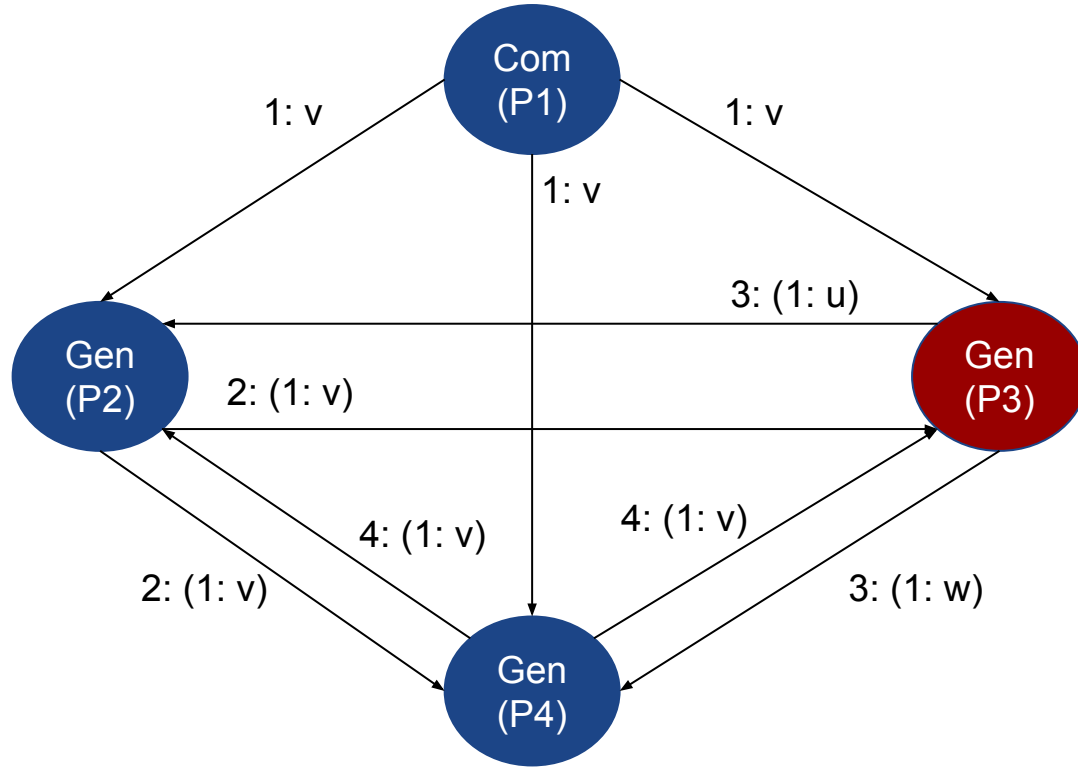
- Eventualmente todos los procesos activos setean su ***decision variable***





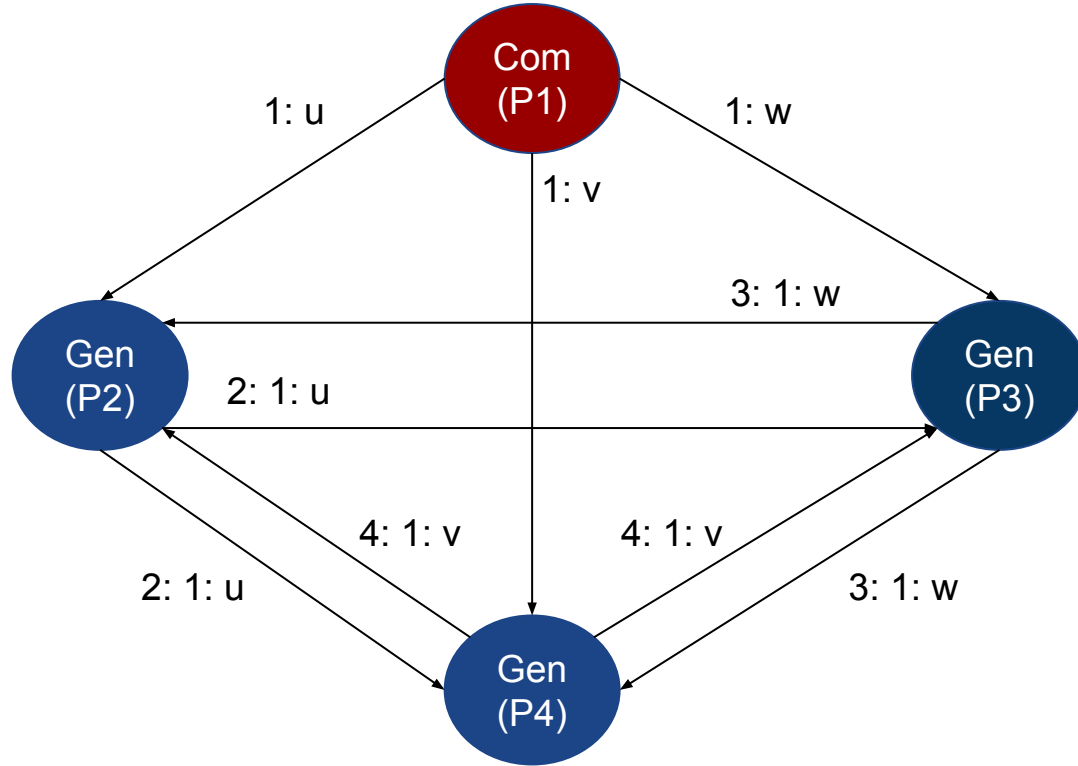


Generales Bizantinos | Solución $N \geq 3f + 1$





Generales Bizantinos | Solución $N \geq 3f + 1$



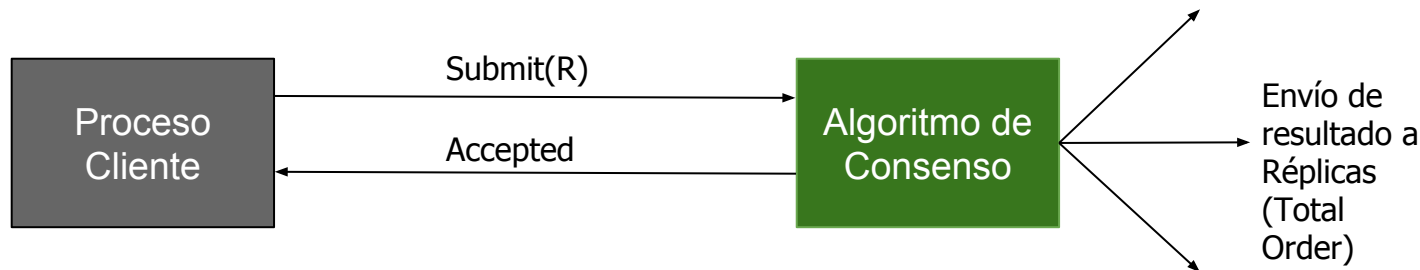
Agenda



- ☐ Elección de Líder
- ☐ Consenso
- ☐ Generales Bizantinos
- ☒ **Paxos**



- **Objetivo**
 - Consensuar un valor aunque múltiples procesos realicen diferentes propuestas
- **Características**
 - **Tolerante a Fallos**
 - Algoritmo progresa siempre que haya una mayoría de procesos vivos
 - $N \geq 2f + 1$
 - **Posible Rechazar Propuestas**
 - Pedido de un cliente puede ser rechazado
 - Cliente puede reintentar una propuesta las veces que desee



- Cliente intenta setear/modificar un un valor R (key/value)
 - Si puede hacerlo, recibe un OK
 - En caso contrario, vuelve a intentar de nuevo
- **Paxos asegura orden consistente en un cluster**
 - Eventos realizados por clientes son almacenados de forma incremental por ID

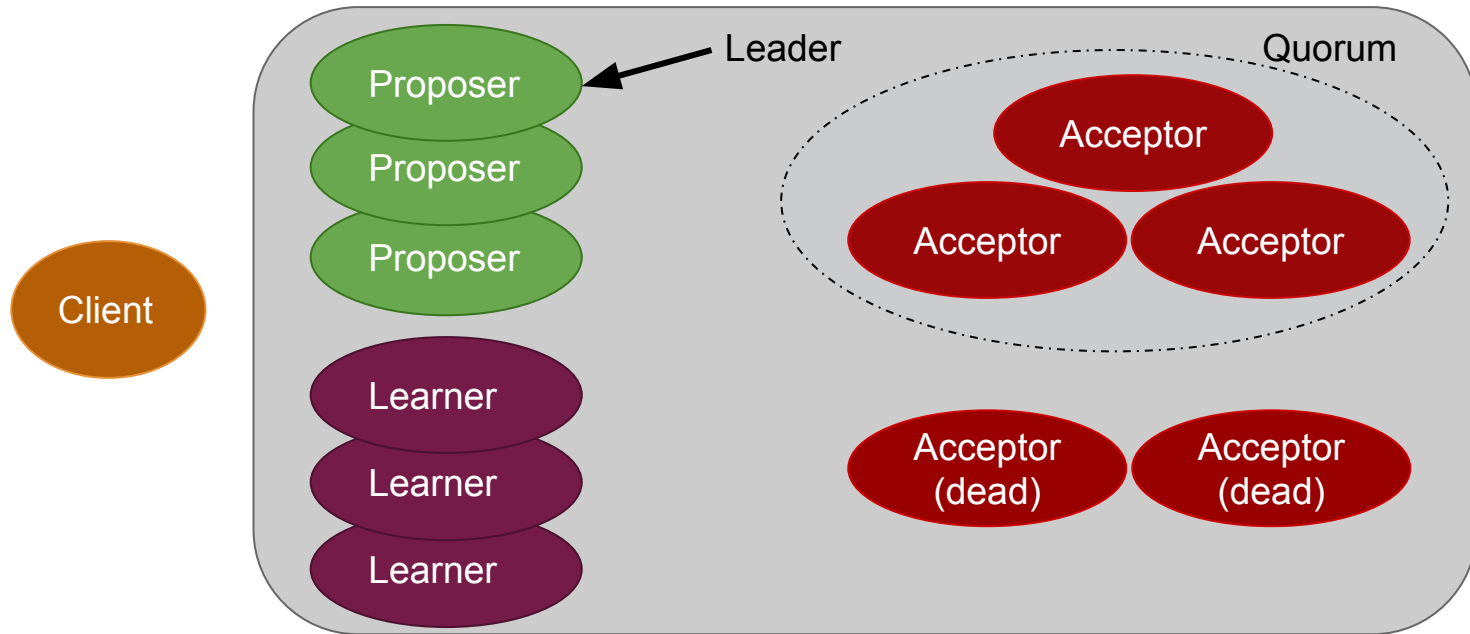


- **Client**
 - Realizan requests
- **Proposer**
 - Reciben requests de clientes y comienzan el protocolo
 - *Leader* debe ser elegido para evitar *starvation* (para más adelante...)
- **Acceptor**
 - Reciben propuestas/promesas de los Proposers
 - Mantienen el estado del protocolo en almacenamiento estable
 - Quorum: Existe cuando la mayoría de Acceptors se encuentran vivos
- **Learner**
 - Cuando el protocolo llega a un acuerdo, Learner ejecuta el request y envía respuesta al cliente



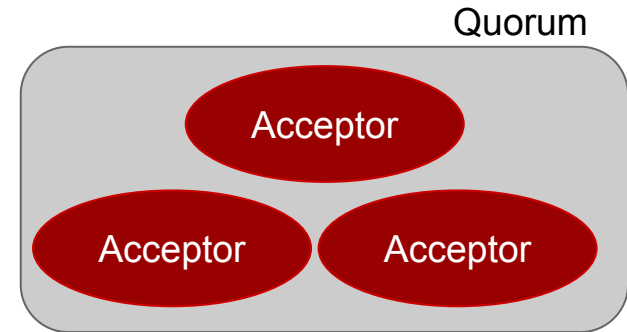
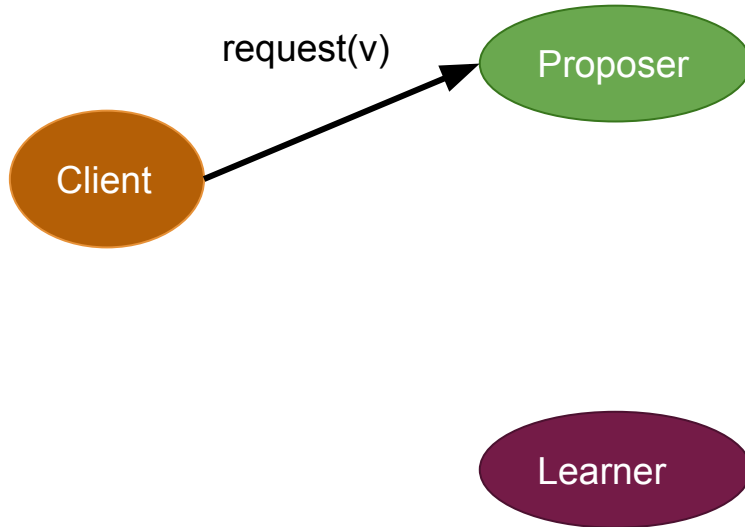
Paxos | Arquitectura

- **Objetivo:** Lograr que todos los **Acceptors** consensuen un valor **v** asociado a una propuesta realizada por un **Proposer**





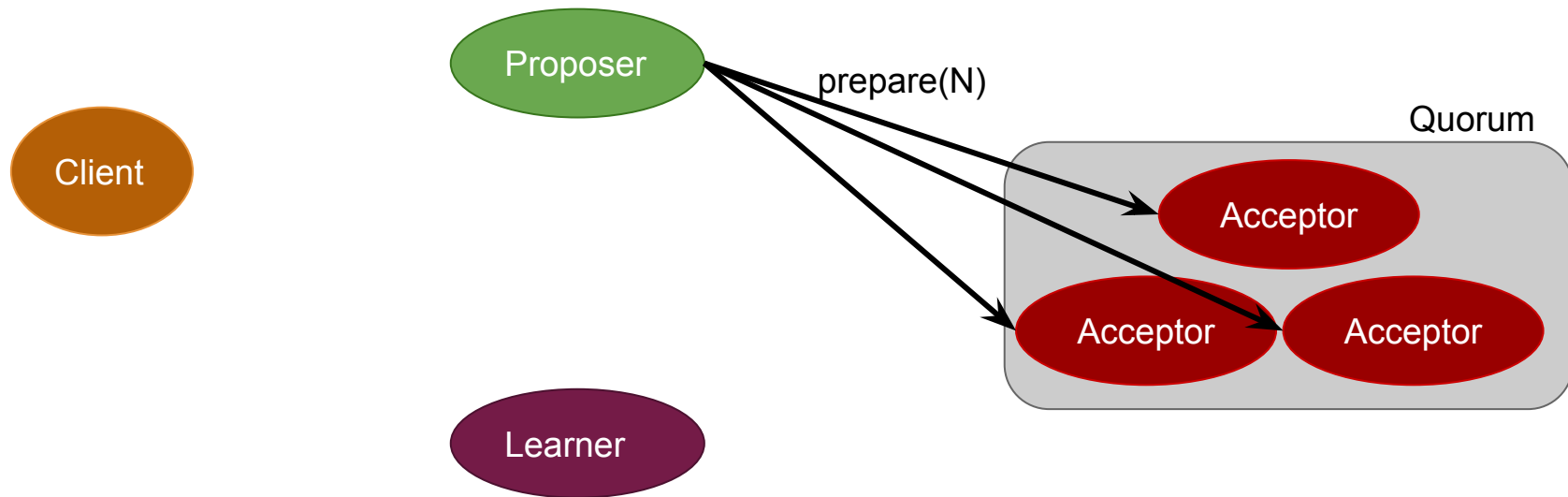
- **Client** realiza un Request





Paxos | Fase 1a - Prepare

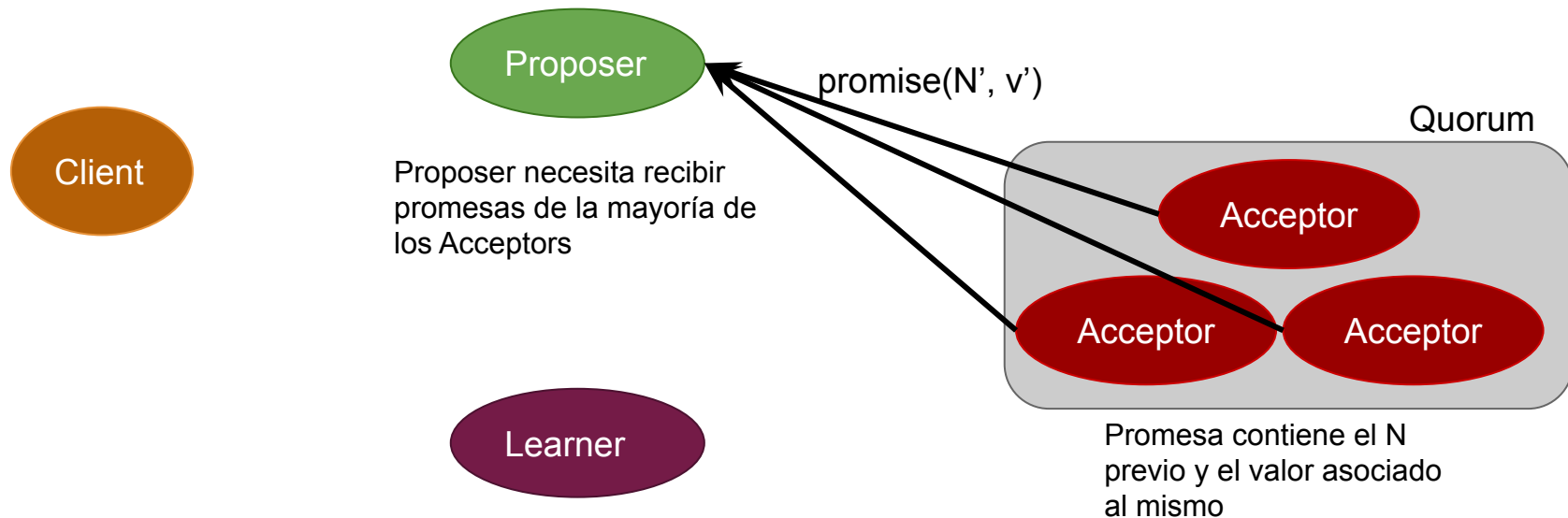
- **Proposer:** Crea una propuesta #N donde N es mayor a cualquier propuesta realizada previamente por el Proposer
- Enviar propuesta a Acceptors esperando obtener Quorum (mensaje llegue a la mayoría de Acceptors)





Paxos | Fase 1b - Promise

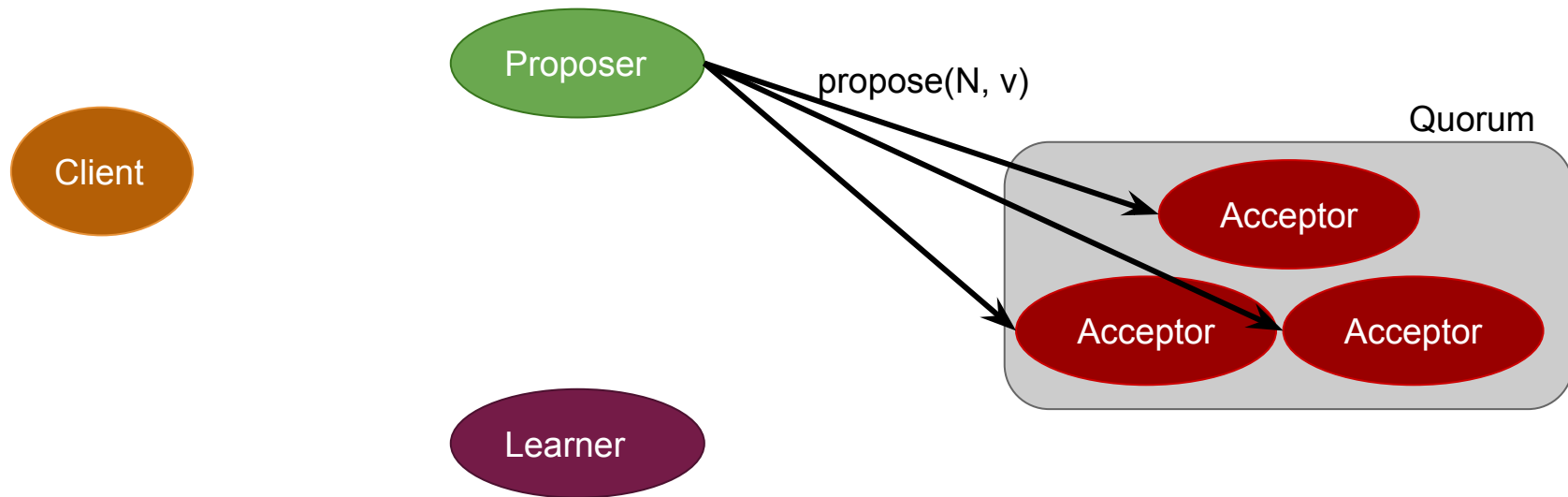
- Si ID recibido del Proposer es mayor al último recibido, Acceptors prometen rechazar cualquier Requests con $ID < N$
- Envío de Promesa a Proposer
- Acceptors no contestan si llega una propuesta no se cumpla que $N > N'$





Paxos | Fase 2a - Propose

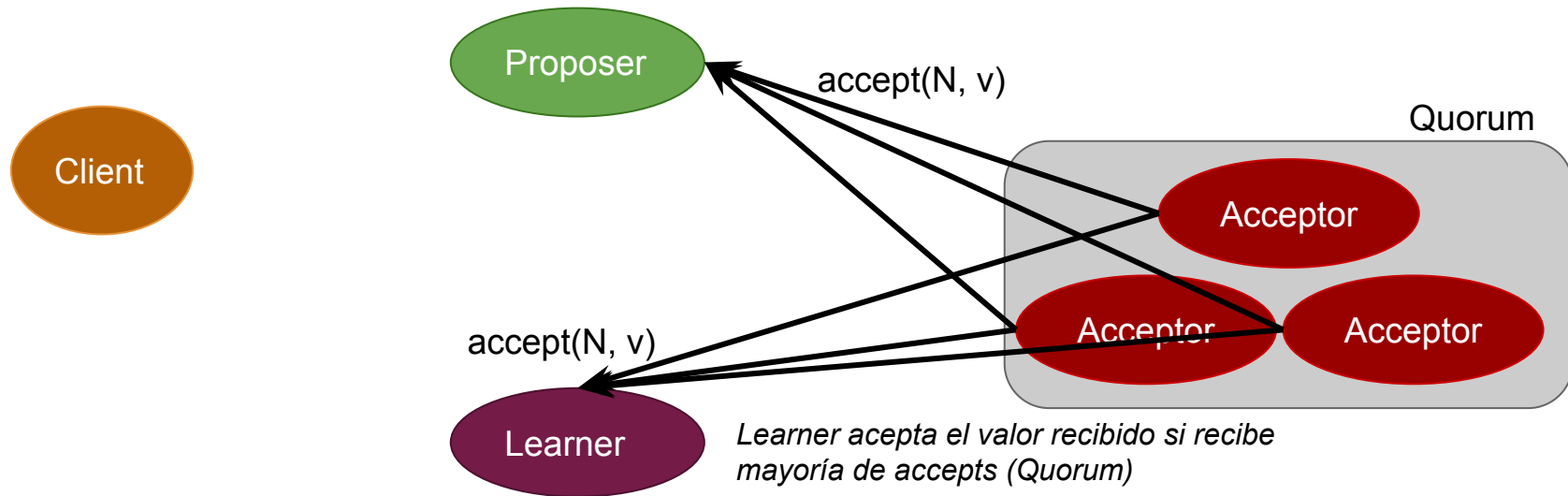
- **Proposer:** Si recibe Promesas de la mayoría:
 - Rechaza todos los requests con un ID $< N$
 - envía Propose con el N recibido por los acceptors y un valor v: $\text{Propose}(N, v)$





Paxos | Fase 2b - Accept

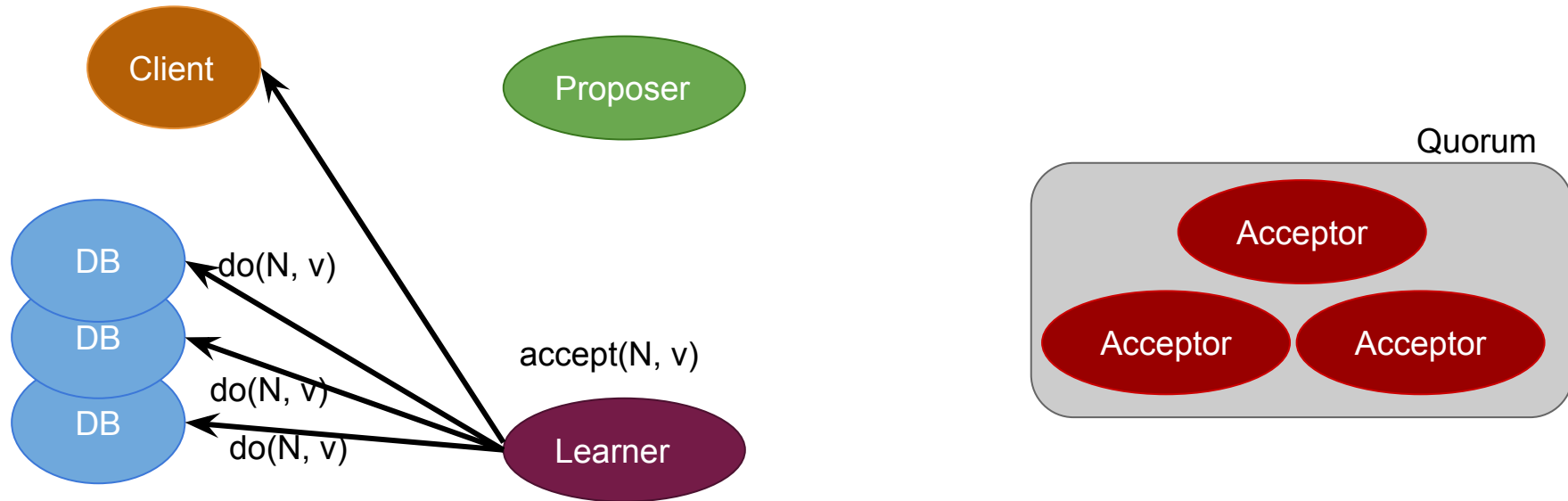
- **Acceptor:** Si la promesa aún es mantenida, anunciar el nuevo valor v
 - Enviar **accepts** a todos los Learners y al Proposer que envió el request inicial
 - No enviar **accepts** si un ID superior a N fue recibido





Paxos | Fase 2c - Accept

- **Learner:** Responde al cliente y se toman acciones respecto del valor acordado





- P. Verissimo, L. Rodriguez: Distributed Systems for Systems Architects, Kluwer Academic Publishers, 2001.
 - Capítulo 6: Fault-Tolerant System Foundations
 - Capítulo 7: Paradigms for Distributed Fault Tolerance
 - Capítulo 8: Models of Distributed Fault-Tolerant Computing
- G. Coulouris, J. Dollimore, t. Kindberg, G. Blair: Distributed Systems. Concepts and Design, 5th Edition, Addison Wesley, 2012.
 - Capítulo 15: Coordination And Agreement
 - Capítulo 17: Replication