Elección de Líder

- Cualquier proceso puede iniciar elección.
- Nunca más de un lider.
- Resultado de elección **único y repetible**.
- Estado: Identificador (`P`), indefinido (`@`).

Ring

- Inicio: marcarse como `participando`, mandar un msj con su ID.
- Al recibir mensaje de elección:
  - Si estado es `no participando`: se marca como `participando`, y manda el msj con el max ID entre el suyo y el que venía.
  - Si estado es `participando`: reenvía el mensaje si el ID es mayor al suyo.
    - Si el ID es el suyo, **se identifica como líder**.
- Al reconocerse como lider: se marca como `no participando` y envía msj de `lider elegido`.
- Al recibir msj de `líder elegido`: se marca como `no participando`, settea el líder y lo retransmite si el ID no es el suyo.

Bully

Hipótesis

- Canales **reliable**.
- Cada proceso **conoce el ID** asociado del resto de procesos.
- Uso de **timeouts** para detectar procesos muertos.
  - `T = 2 * T_max_transmisión + T_max_process`
- **Todos** se pueden **comunicar** entre sí.

Mensajes

- `Election`: iniciar elección.
- `Answer`: ACK.
- `Coordinator`: quién fue elegido.

Algoritmo

- Al detectar que el líder se cayó, se manda `Election` a procesos con ID mayor.
- Si un proceso recibe `Election`, responde con `Answer` y comienza una nueva elección.
- Si un proceso recibe `Coordiator`, setea a su emisor como líder.
- Si un proceso que comenzó no recibe `Answer` tras `T` tiempo, se autoproclama líder.

Consenso

Dado un **conjunto de procesos distribuidos** y un punto de **decisión**, todos deben **acordar** en el mismo valor.

Propiedades necesarias

- **Agreement.** El valor de la variable `decided` es el mismo en todos los procesos correctos.
- **Integrity.** Si procesos correctos propusieron el mismo valor, entonces tienen la misma `decisón variable`.
- **Termination.** Todos los procesos activos setean eventualmente su `decisión variable`.

Generales Bizantinos

- Un proceso líder y varios followers.
  - Líder emite un valor.
  - Followers lo reenvían al resto.
- Procesos *traicioneros*: pueden enviar valores incorrectos.
- `N >= 3f + 1`

Paxos

- Objetivo: **consensuar valor** aunque hayan **diferentes propuestas**.
- **Tolerante a fallos.** Progresa si hay mayoría de procesos vivos.
  - Quorum: `N >= 2f + 1`
- Posible rechazar propuestas.
- Asegura **órden consistente en un cluster**.
  - Eventos almacenados incrementalmente por ID.

Actores

- **Proposer.** Recibe request e inicia protocolo.
- **Acceptor.**
  - Mantiene estado del protocolo en **almacenamiento estable**.
  - Quorum si la mayoría están vivos.
- **Learner.** Cuando hay acuerdo, se ejecuta el request.

Algoritmo

0. Request del cliente.
1. Fase 1: primera propuesta.
   1. **Prepare.**
      - Proposer envía propuesta `#N`.
      - `N` mayor a cualquier propuesta previa.
   2. **Promise.**
      - Si ID es mayor al último recibido, se rechaza cualquier request con `ID < N`.
      - Envía `Promise` al proposer.
2. Fase 2: si recibí promise de la mayoría.
   1. **Propose.**
      - Rechaza requests con `ID < N`.
      - Envía `Propose` con el `N` recibido y un valor `v`.
   2. **Accept.**
      - Si la propuesta aun es válida, anuncia nuevo valor `v`.
      - Envia `accepts` a todos los learners y al proposer.
   3. El **learner responde** al cliente.