ÉCOLE POLYTECHNIQUE MONTRÉAL

Module 6:

Coordination et consensus

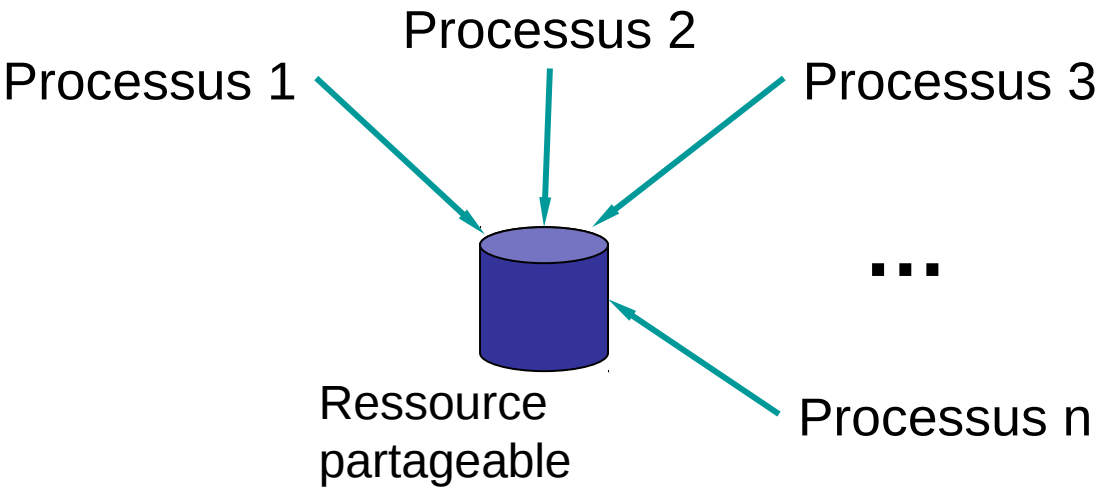
Introduction :

- **Un des problème fondamentaux des systèmes répartis : le consensus!**
  - Comment faire pour mettre d'accord plusieurs processus indépendants?
  - Comment faire pour coordonner leurs actions?

- **Comment faire pour résoudre ce problème?**
  - Une solution naïve est d'implémenter le modèle maître-esclave ;

- **Est-ce que le problème du consensus est le même pour les systèmes synchrones et asynchrones?**

Exclusion mutuelle répartie

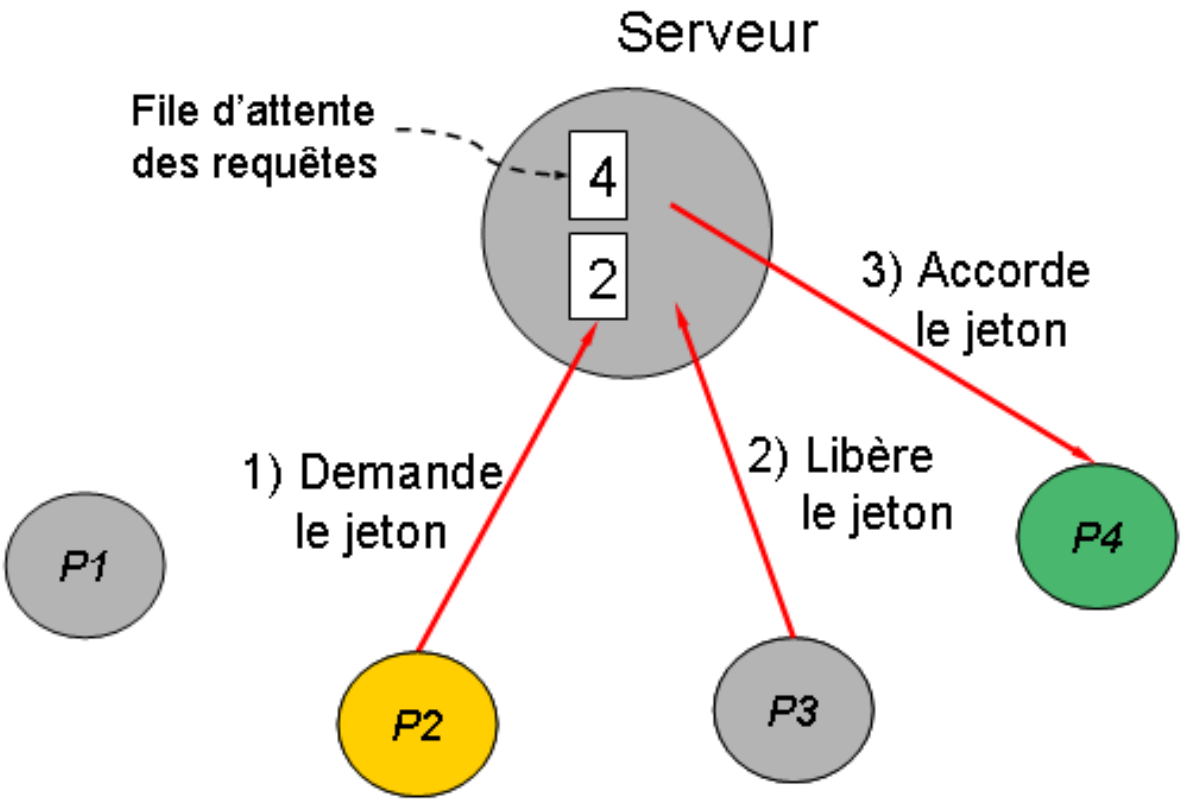


- **Exclusion mutuelle est nécessaire pour :**
  - Prévenir les interférences ;
  - Assurer la cohérence en cas d'accès simultanés aux ressources.

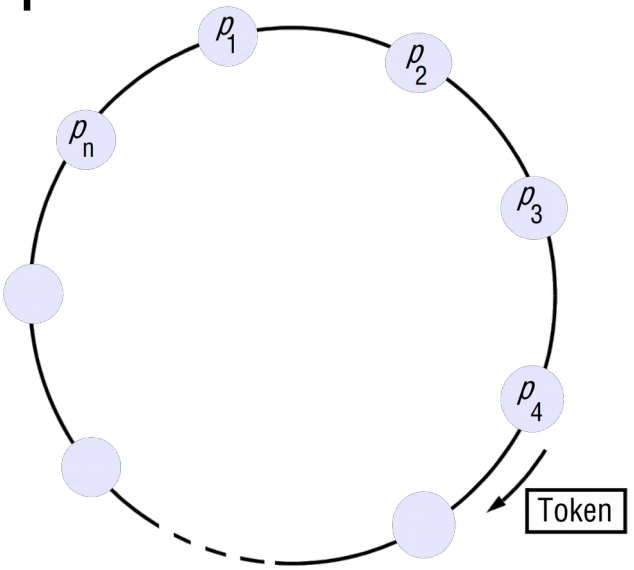
- **Synchronisation par exclusion mutuelle répartie**
  - Sûreté: un seul client à la fois obtient le verrou.
  - Vivacité: personne n'est laissé pour compte, chaque client voit éventuellement sa requête satisfaite.
  - Ordre: premier arrivé, premier servi, si un client A fait une demande, envoie un message à B, et B fait une demande, l'ordre logique dit que la demande de A arrive avant celle de B. Elle devrait être servie en premier.

- **Serveur central**
  - Un client envoie une demande de verrou, le serveur attend que le verrou soit libre, le donne au client, et le client le relâche éventuellement, ce qui permet au serveur de le donner à un autre client.
  - Si le serveur est en panne, élection d'un nouveau serveur (majorité de clients), et vérification de tous les clients pour voir qui possède des verrous ou a soumis une requête (problème si clients non rejoignables en raison de division du réseau).
  - Exemple: serveur *lockd* sous NFS.

- **Serveur central**



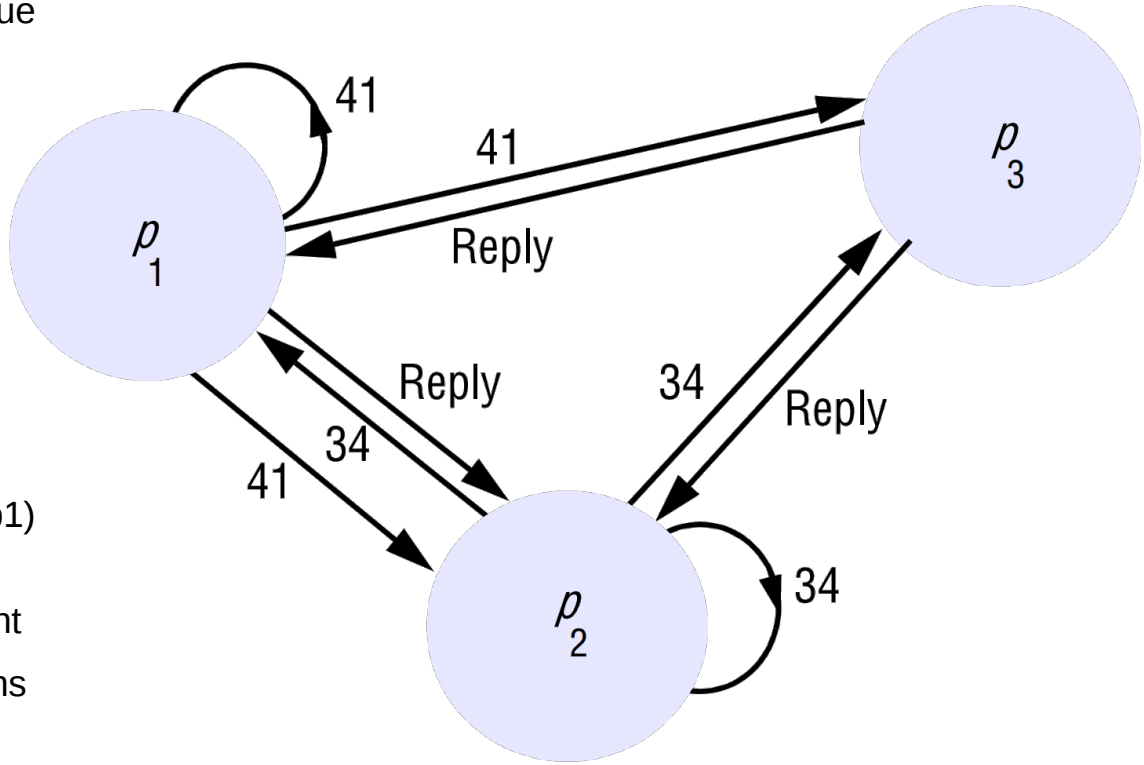
- **Anneau à jeton**
  - Le verrou passe de processus en processus constamment.
  - Inutile lorsque personne ne requiert le verrou.
  - Problème dès qu'un client fait défaut.
  - Ne respecte pas premier arrivé, premier servi.



- **Exclusion par envoi à tous**
  - Envoyer une requête à tous.
  - Recevoir le O.K. de tous.
  - Si on a le verrou, attendre d'en avoir terminé avant de répondre O.K.
  - Si on veut le verrou et notre demande est antérieure, attendre le verrou et d'en avoir fini avant de répondre O.K.
  - Demande n messages (ou 2n - 2 sans message à tous).
  - Tous les clients doivent être actifs.
  - Moins bon que serveur central.

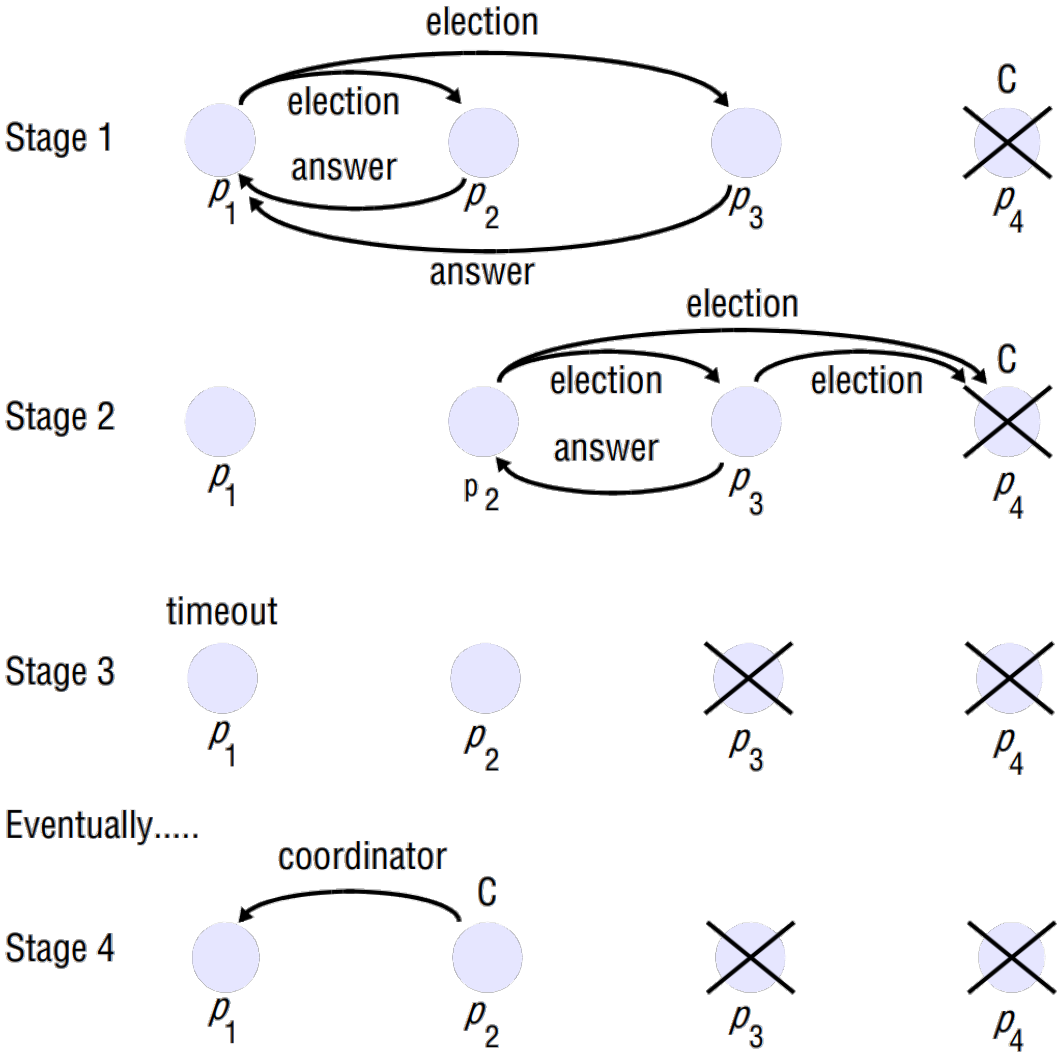
Exclusion par envoi à tous

- p3 ne veut pas entrer dans la section critique
- p1 et p2 demandent accès à la section critique en même temps:
  - requête de p1 T=41
  - requête de p2 T=34
- Processus:
  - p3 réponds immédiatement aux requêtes
  - p2 reçoit la requête de p1, il voit que T(p2) < T(p1) et ne répond pas (hold p1)
  - p1 reçoit la requête de p2, il voit que T(p2) < T(p1) et répond immédiatement
  - p2 reçoit la réponse de p1 et entre dans la section critique; quand il la quitte, il répond à p1
  - p1 reçoit la réponse de p2 et entre dans la section critique



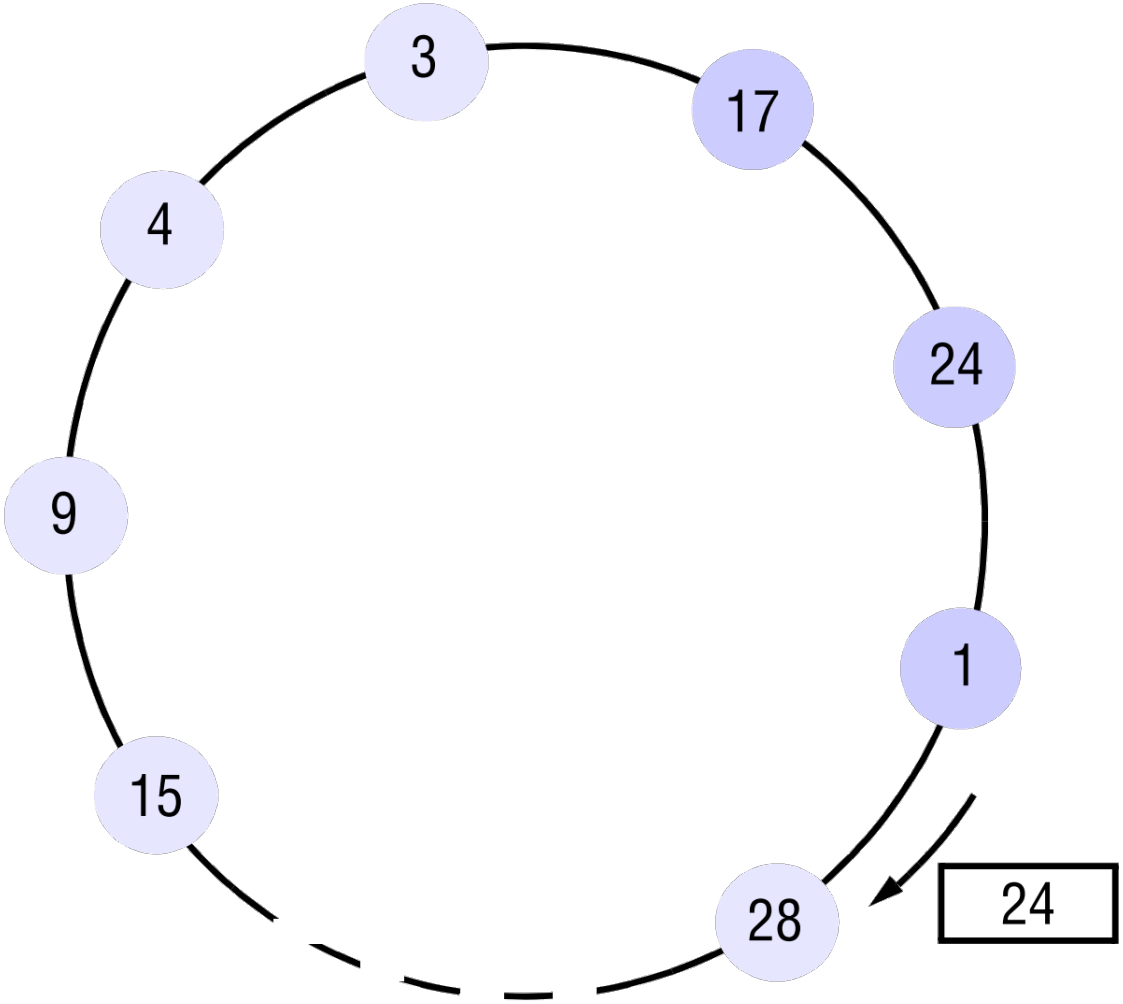
- **Election hiérarchique (Bully algorithm)**
  - Lorsqu'un nouvel ordinateur est connecté, ou si le coordonnateur ne peut être rejoint, déclencher une élection.
  - Envoyer un message d'élection à ceux de plus haute priorité.
  - Pas de réponse, il se proclame coordonnateur et le signale à tous.
  - Une réponse, il attend un message de proclamation qui devrait suivre.
  - Demande n-1 messages dans le meilleur des cas, n*n si tous les processus déclenchent une élection en même temps.

• Election hiérarchique



- **Election en anneau**
  - Un participant envoie son message d'élection avec son ID.
  - Le prochain participant envoie max(ID reçu, ID) à son voisin.
  - Si un participant reçoit un message avec son ID, il se déclare élu et propage la nouvelle.
  - Ceci prend un maximum de 3n - 1 messages (ID maximum n - 1, se découvrir élu n, propager la nouvelle n).

- **Election en anneau**



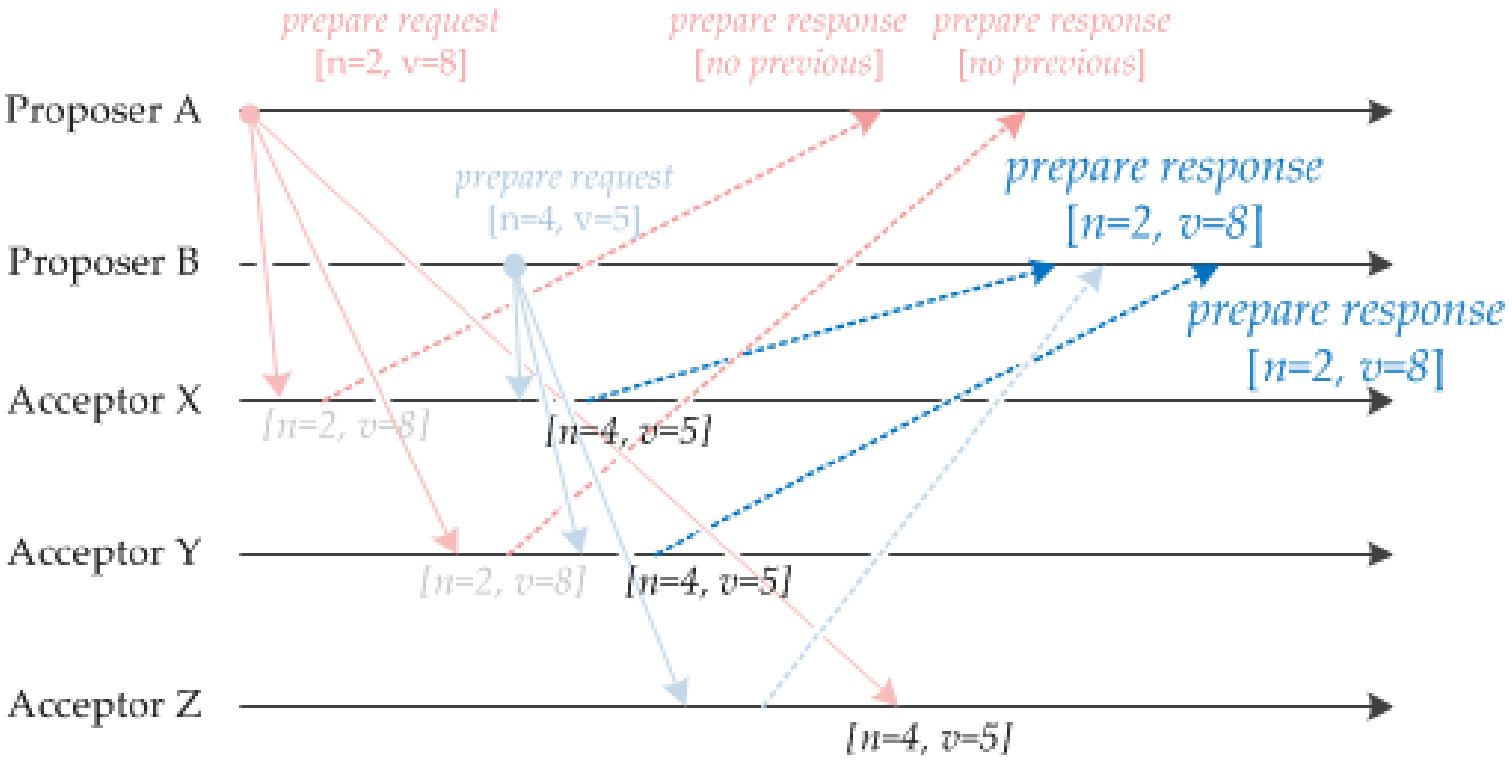
- **Le consensus en réparti**
  - Plusieurs processus, corrects ou fautifs, échangent des messages.
  - Chaque processus doit prendre une décision (e.g. qui est le serveur maître).
  - Terminaison: éventuellement tous les processus corrects arrivent à une décision.
  - Consensus: tous les processus corrects terminent avec la même décision.
  - Intégrité: si tous les processus corrects proposent la même décision, cette décision doit l'emporter.
  - Chaque processus correct communique sa proposition et celle déjà connue d'autres processus au groupe. Chaque processus accumule les propositions des autres et se soumet à la proposition majoritaire.

- **Algorithme de Paxos**
  - Le consensus en réparti est très difficile à assurer en présence de défaillances, messages asynchrones, partitionnement de réseau...
  - Proposé par Lamport en 1989 et publié dans une revue, après avoir été entièrement revisé seulement en 1998.
  - Complexe, utilisé surtout pour les questions fondamentales (e.g., élire un serveur maître de manière sécuritaire, ensuite le serveur maître peut coordonner le reste).

- **Algorithme de Paxos**
  - Par exemple, si on veut élire (établir un consensus) un serveur maître, il faut obtenir une majorité parmi les accepteurs.
  - Chaque proposition a un numéro de séquence et une valeur.
  - Première proposition reçue par un accepteur, il promet de ne pas accepter une proposition antérieure et mémorise cette proposition.
  - Proposition antérieures reçues par un accepteur, il promet mais retourne la plus récente proposition reçue.
  - Le proposeur choisit la proposition la plus récente déjà reçue par un accepteur et demande à chaque accepteur de l'accepter.
  - Si une majorité d'accepteurs acceptent, le consensus est obtenu.

Algorithme de Paxos : exemple

- A envoie “accept n=2, v=8” qui est ignoré et B envoie “accept n=4, v=8” qui est accepté par tous.



https://angus.nyc/2012/paxos-by-example/