Chapitre 4
Algorithmique Répartie

Accord et coordination

Une famille de problèmes en algorithmique distribuée

Accord et coordination

Catégories de problèmes de cette famille :

Accord sur l'accès à une ressource partagée

Exclusion mutuelle

Accord sur l'ordre d'envoi de messages à tous

Diffusion atomique

Accord sur un processus jouant un rôle particulier

Élection d'un maître

Accord sur une valeur commune

Consensus

Accord sur une action à effectuer par tous ou personne

Transaction

Accord et coordination

« Consensus »

Plan

- Consensus : principe général
- Conditions à valider
- Consensus dans différents environnements :
  - **Sans faute**
  - **Synchrone, panne franche**
  - **Asynchrone, panne franche**
  - **Asynchrone, fautes byzantines**
  - **Synchrone, fautes byzantines**
    - Problème des généraux byzantins
- Consensus : résumé

Principe général

- Des processus doivent se mettre d'accord sur une valeur commune.
- **Deux étapes :**
  - Chaque processus fait une mesure ou un calcul : **valeur locale** qui est proposée à tous les autres processus.
  - Les processus, à partir de toutes les valeurs proposées, doivent se décider sur **une valeur unique commune**.
  - Soit un processus initie le consensus.
  - Soit le consensus est lancé à des instants prédéterminés.

Un ensemble de processus p1, . . . , pn reliés par des canaux de communication.

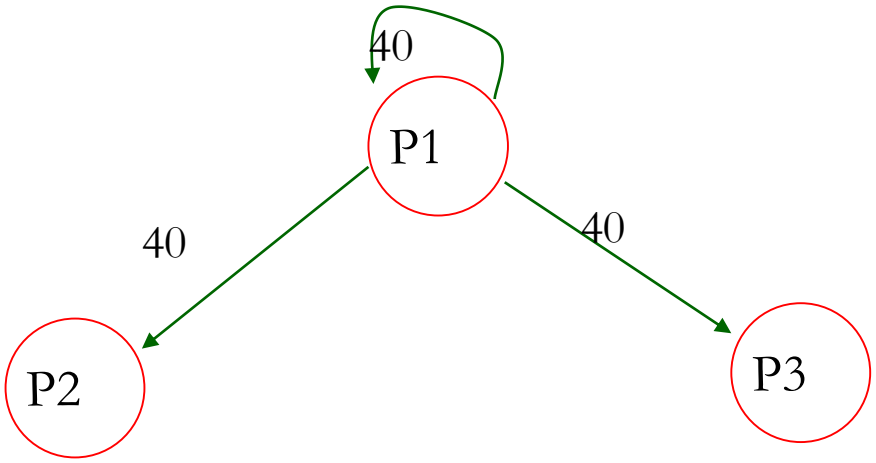
Initialement : chaque processus pi propose une valeur vi . A la terminaison de l'algorithme : les processus pi se mettre d'accord sur une valeur commune di.

Conditions à valider

- **Accord :**
  - La valeur décidée est la même pour tous les processus corrects.
- **Validité :**
  - La valeur choisie par un processus est une des valeurs proposées par l'ensemble des processus.
- **Intégrité :**
  - Un processus décide au plus une fois : pas de changement de choix de valeur.
- **Terminaison :**
  - La phase de décision se déroule en un temps fini : tout processus correct décide en un temps fini.

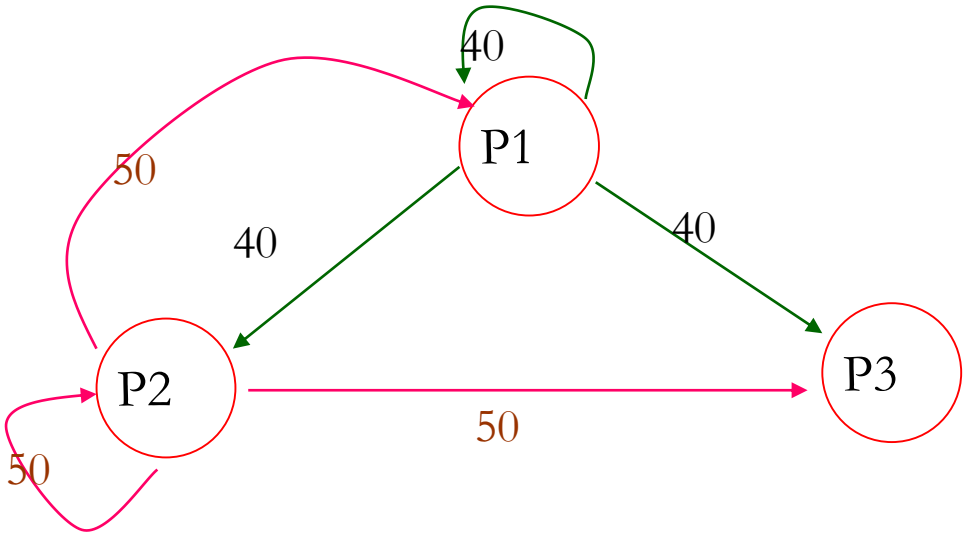
Consensus : sans faute

- **Solution simple :**
  - Après la mesure locale, un processus envoie sa valeur à tous les autres.
  - A la réception de toutes les valeurs (après un temps fini par principe), un processus applique une fonction donnée sur l'ensemble des valeurs.
  - La fonction est la même pour tous les processus.
    - Chaque processus est donc certain d'avoir récupéré la valeur commune qui sera la même pour tous.



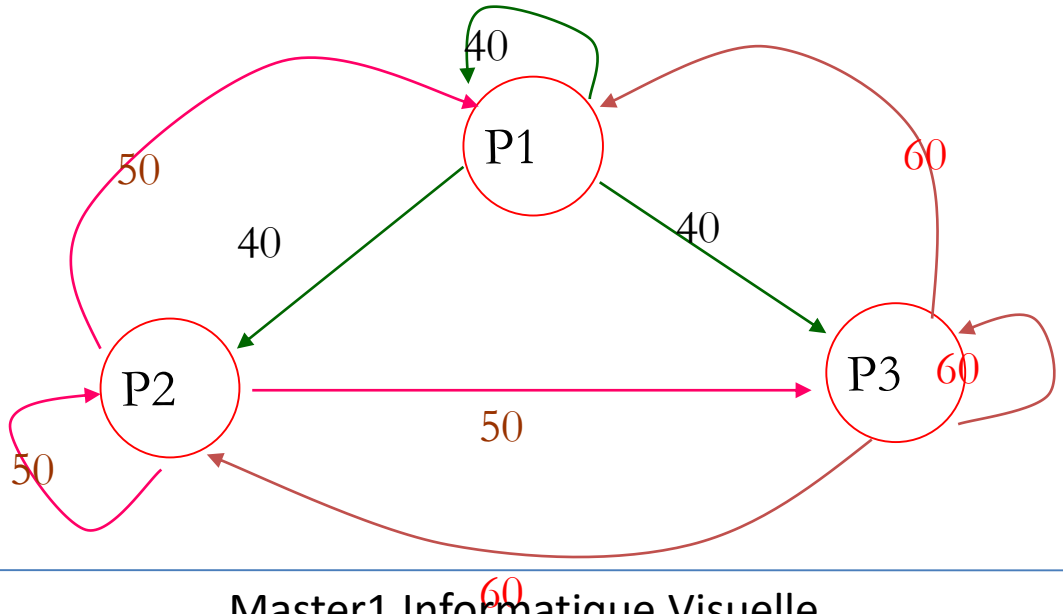
Consensus : sans faute

- **Solution simple :**
  - Après la mesure locale, un processus envoie sa valeur à tous les autres.
  - A la réception de toutes les valeurs (après un temps fini par principe), un processus applique une fonction donnée sur l'ensemble des valeurs.
  - La fonction est la même pour tous les processus.
    - Chaque processus est donc certain d'avoir récupéré la valeur commune qui sera la même pour tous.



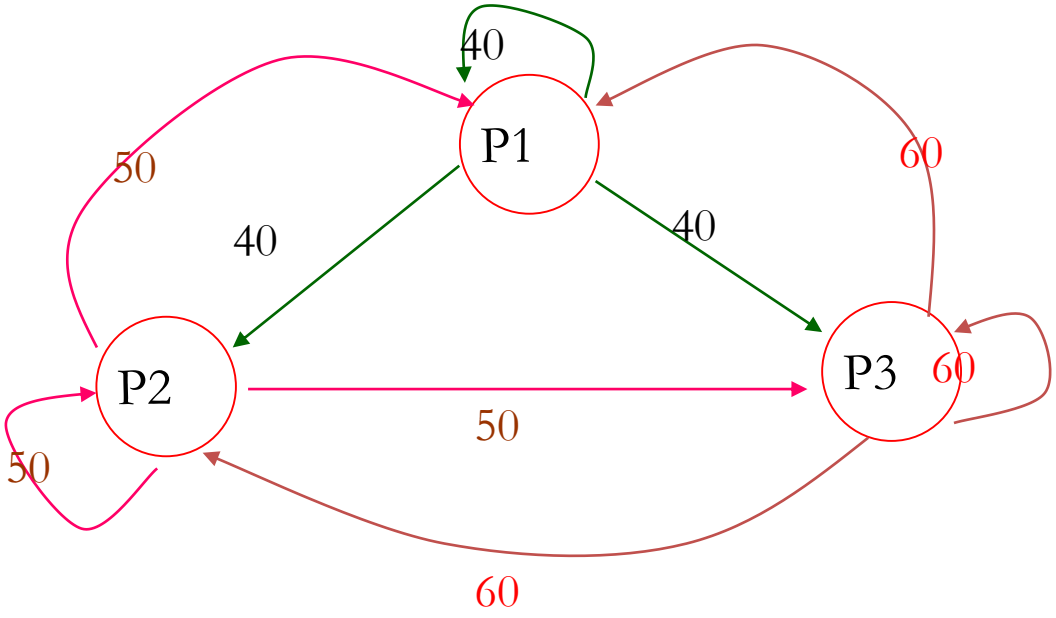
Consensus : sans faute

- **Solution simple :**
  - Après la mesure locale, un processus envoie sa valeur à tous les autres.
  - A la réception de toutes les valeurs (après un temps fini par principe), un processus applique une fonction donnée sur l'ensemble des valeurs.
  - La fonction est la même pour tous les processus.
    - Chaque processus est donc certain d'avoir récupéré la valeur commune qui sera la même pour tous.



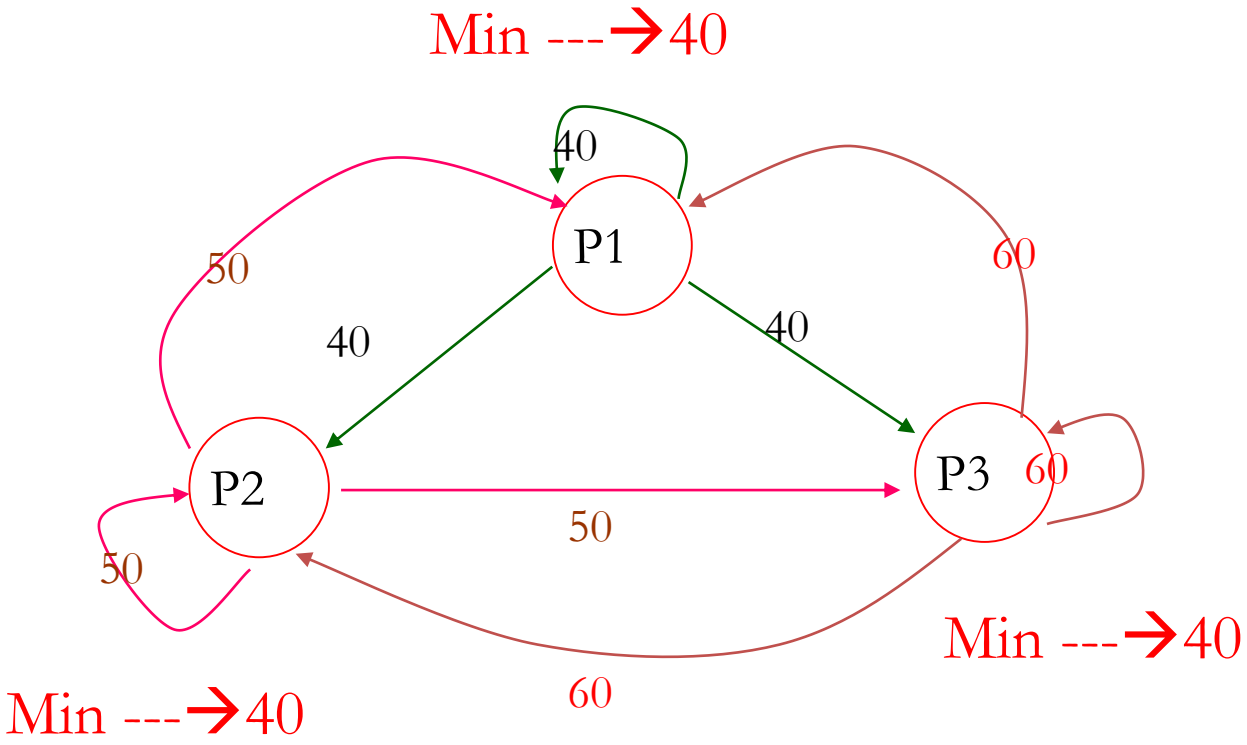
Consensus : sans faute

- Les processus appliquent la même fonction sur l'ensemble des valeurs.



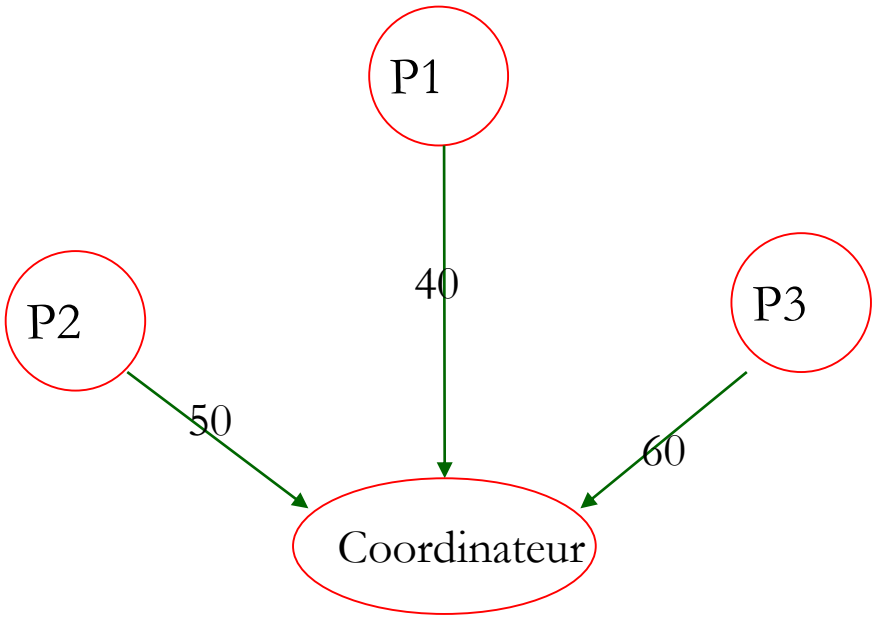
Consensus : sans faute

- Les processus appliquent la même fonction sur l'ensemble des valeurs.



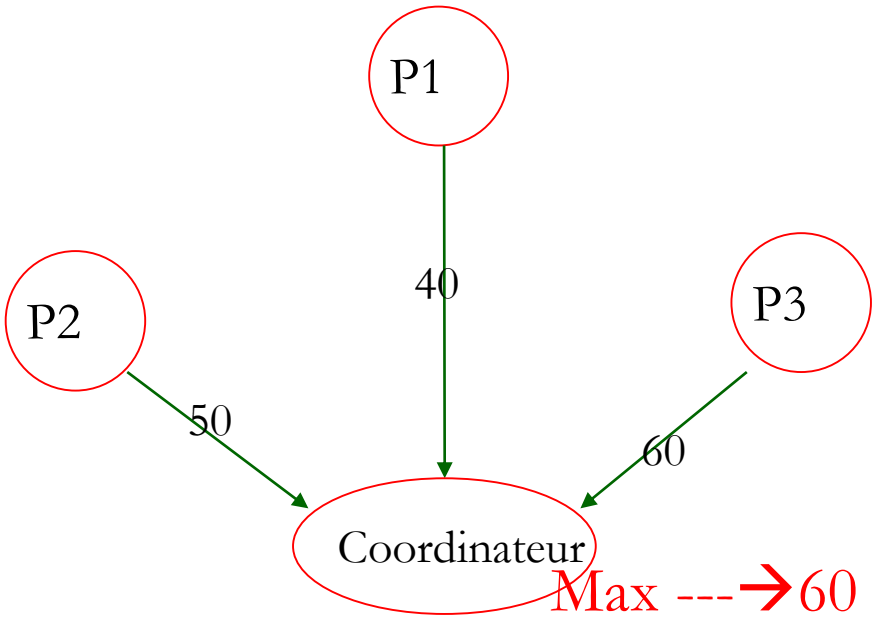
Consensus : sans faute

- **Autre solution simple :**
  - Passer par un processus **coordinateur** qui centralise la réception des valeurs proposées et fait la décision, puis la diffuse.
- Ces solutions fonctionnent pour les systèmes distribués **synchrones ou asynchrones.**
  - La différence est le temps de terminaison qui est borné ou non.



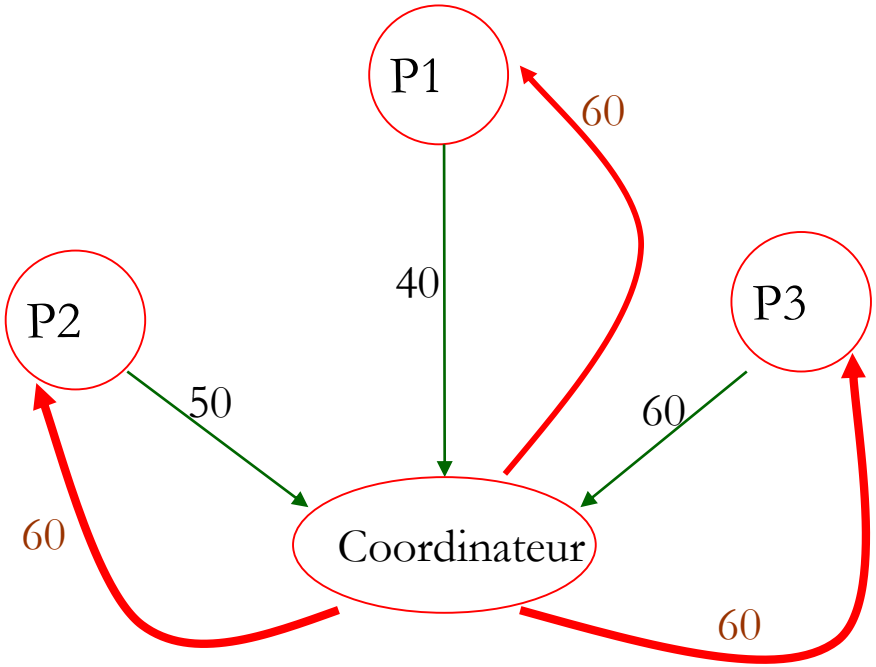
Consensus : sans faute

- **Autre solution simple :**
  - Passer par un processus **coordinateur** qui centralise la réception des valeurs proposées et fait la décision, puis la diffuse.
- Ces solutions fonctionnent pour les systèmes distribués **synchrones ou asynchrones.**
  - La différence est le temps de terminaison qui est borné ou non.



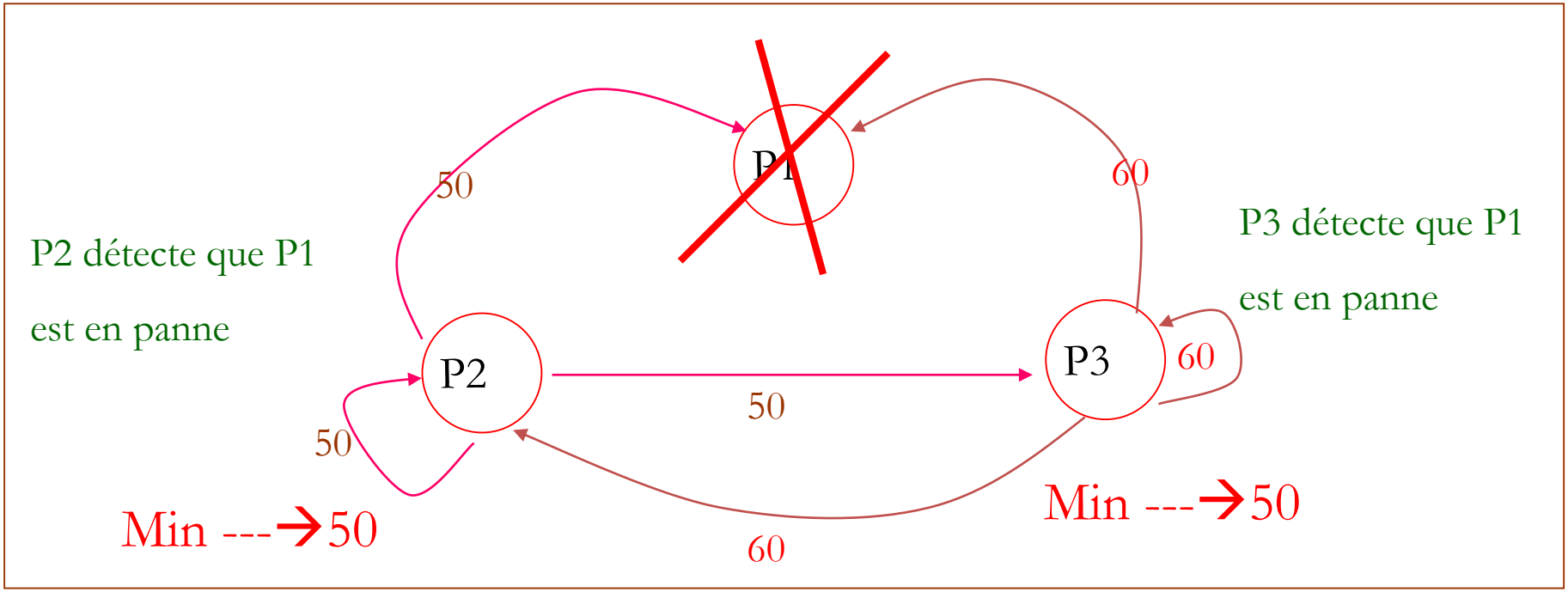
Consensus : sans faute

- **Autre solution simple :**
  - Passer par un processus **coordinateur** qui centralise la réception des valeurs proposées et fait la décision, puis la diffuse.
- Ces solutions fonctionnent pour les systèmes distribués **synchrones ou asynchrones.**
  - La différence est le temps de terminaison qui est borné ou non.



Consensus : synchrone, panne franche

- Hypothèse: communications **fiables** : pas de perte ni de duplication de messages.
- Peut déterminer si un processus n'a pas répondu, c'est-à-dire s'il est planté ou …. .
- **1er cas :** P1 plante avant l'envoi de sa valeur à P2 et P3.



- **2ème cas :** P1 plante pendant l'envoi de sa valeur, un seul processus reçoit la valeur de P1.
  - **Utiliser la diffusion fiable.**
    - Tous les processus recevront exactement les mêmes valeurs et pourront appliquer la même fonction de choix qui aboutira forcément au même choix.

Consensus : asynchrone, panne franche

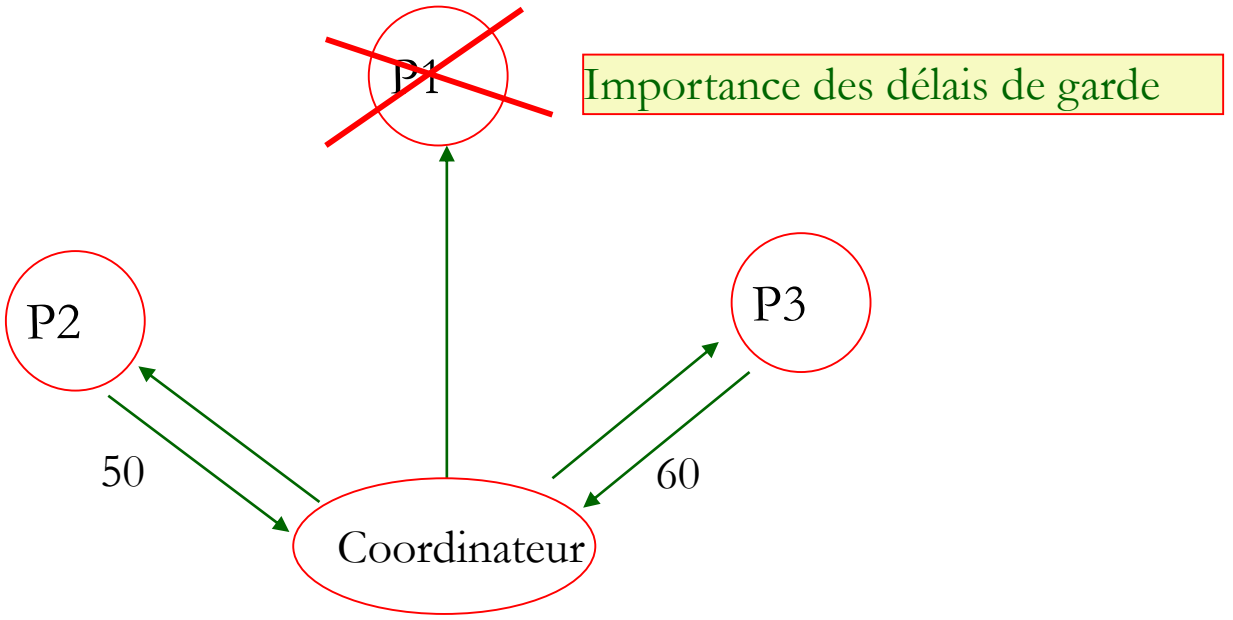
- **En 1983, Fischer, Lynch & Paterson (FLP)** ont montré que :
  - Dans un système asynchrone, même avec un seul processus fautif, **il est impossible d'assurer que l'on atteindra le consensus** (ne se termine pas toujours).
- **FLP précise qu'on atteindra pas toujours le consensus, mais pas qu'on ne l'atteindra jamais.**

- En théorie, le consensus n'est pas atteignable systématiquement en asynchrone. Mais, malgré ce résultat, il est possible en pratique d'atteindre des résultats satisfaisants.

  - **Définir des algorithmes non parfaits.**

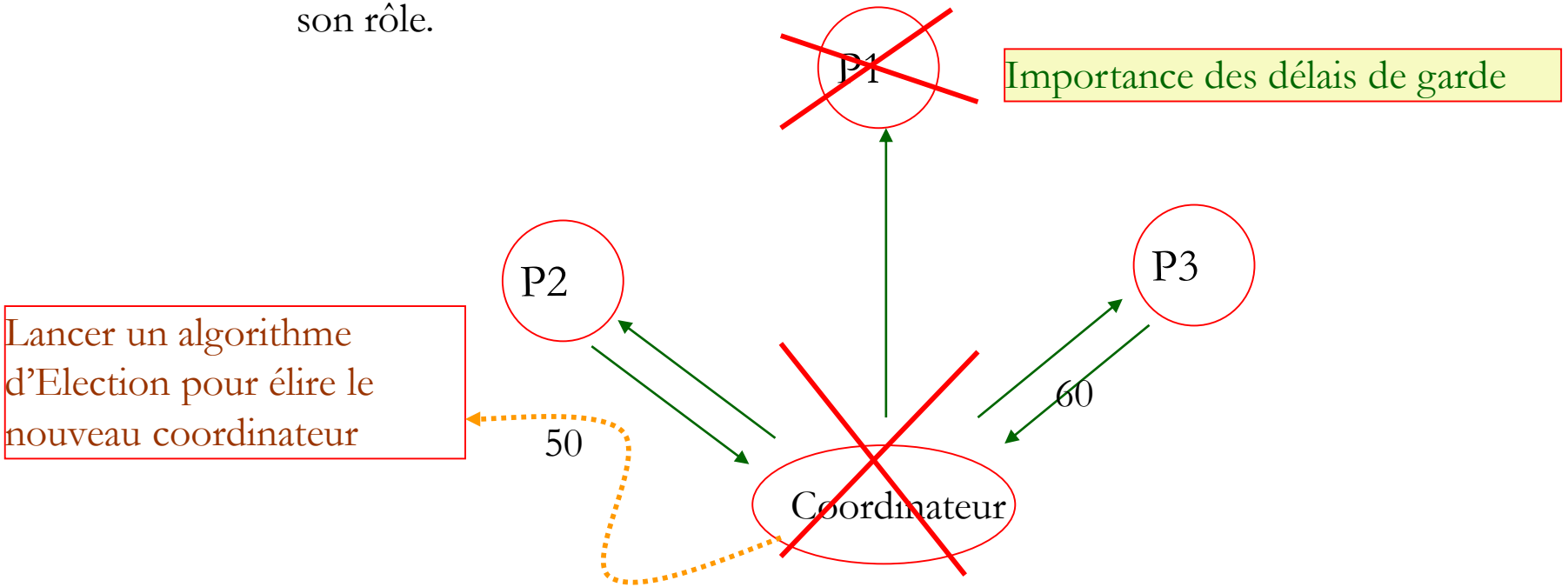
Consensus : asynchrone, panne franche

- **Algorithme de Paxos (Lamport, 89) :** algorithme non parfait de consensus en asynchrone, contexte de pannes franches.
  - **Principe très général :**
    - Un processus coordinateur tente d'obtenir les valeurs des processus et choisi la valeur majoritaire.
    - Si le processus coordinateur se plante, d'autres processus peuvent reprendre son rôle.



Consensus : asynchrone, panne franche

- **Algorithme de Paxos (Lamport, 89) :** algorithme non parfait de consensus en asynchrone, contexte de pannes franches.
  - **Principe très général :**
    - Un processus coordinateur tente d'obtenir les valeurs des processus et choisi la valeur majoritaire.
    - Si le processus coordinateur se plante, d'autres processus peuvent reprendre son rôle.



Consensus : asynchrone, fautes byzantines

- Contexte de fautes byzantines bien **plus complexe à gérer** que les pannes franches.

  - **Consensus pas toujours atteignable** en asynchrone, panne franche.

  - **Consensus pas toujours atteignable** en asynchrone, fautes byzantines.

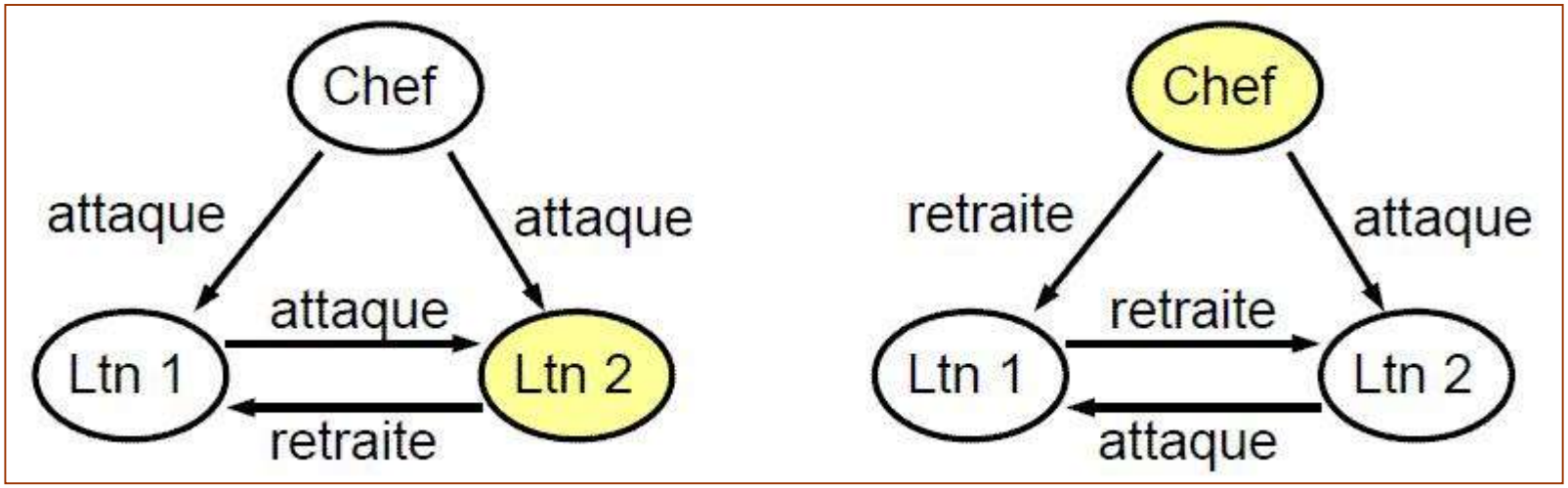
Consensus : synchrone, fautes byzantines

- En synchrone, contrairement aux pannes franches, il n'est pas possible d'assurer que le consensus sera toujours atteint dans un contexte de fautes byzantines.
  - **Lamport, Shostak et Pease** ont montré en 1982, via le problème des généraux byzantins, que : Si f est le nombre de processus fautifs, il faut au minimum $3f + 1$ processus (au total) pour que le consensus soit assuré.
    - En dessous, il n'est pas assuré.

- **Exemple :** avec 2 processus fautifs, il faut au minimum 7 processus au total (5 corrects) pour terminer le consensus.
- Les algorithmes à mettre en œuvre sont relativement lourds en nombre de messages.

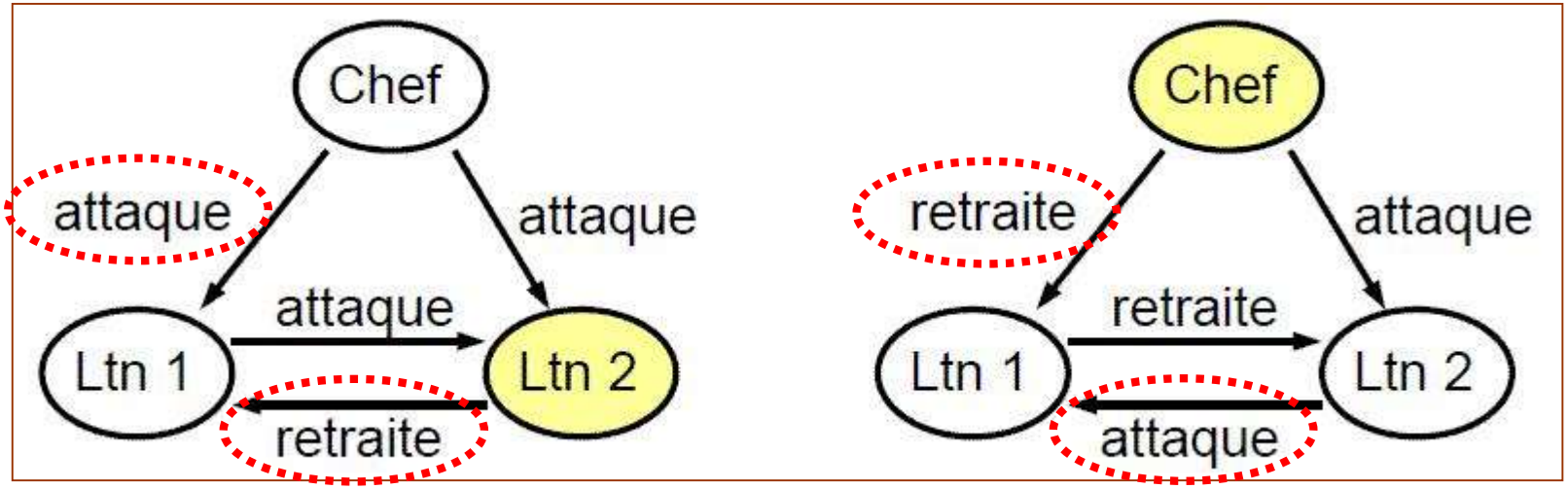
Problème des généraux byzantins

- Un ensemble de généraux doivent se coordonner pour mener une attaque, doivent tous faire la même chose (attaquer ou battre en retraite).
- Le général en chef prend l'ordre et l'envoie aux autres généraux (ses lieutenants). Ces derniers se renvoient l'ordre entre eux pour s'assurer qu'il est bien reçu et est bien le bon.
- **Problème :**
  - Un certain nombre de généraux peuvent être des traîtres.
  - Ils vont envoyer des ordres différents aux autres généraux.
  - Les généraux loyaux doivent détecter les traîtres et ignorer leurs ordres.



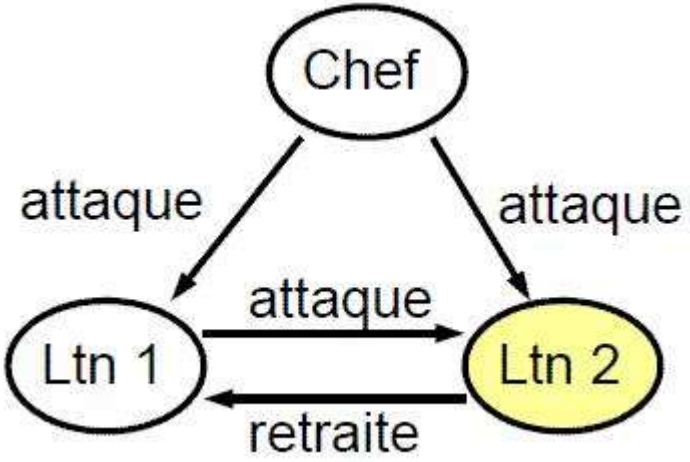
Problème des généraux byzantins

- Contexte avec 3 généraux : généralisable à des multiples de 3 d'où le « 3f + 1 ».
  - **Deux cas :**
    - Gauche : le lieutenant 2 est le traitre, il renvoie le mauvais ordre.
    - Droite : le chef est le traître, il envoie deux ordres différents.
    - Dans les 2 cas, le lieutenant 1 reçoit deux ordres contradictoires en étant incapable de savoir lequel est le bon.



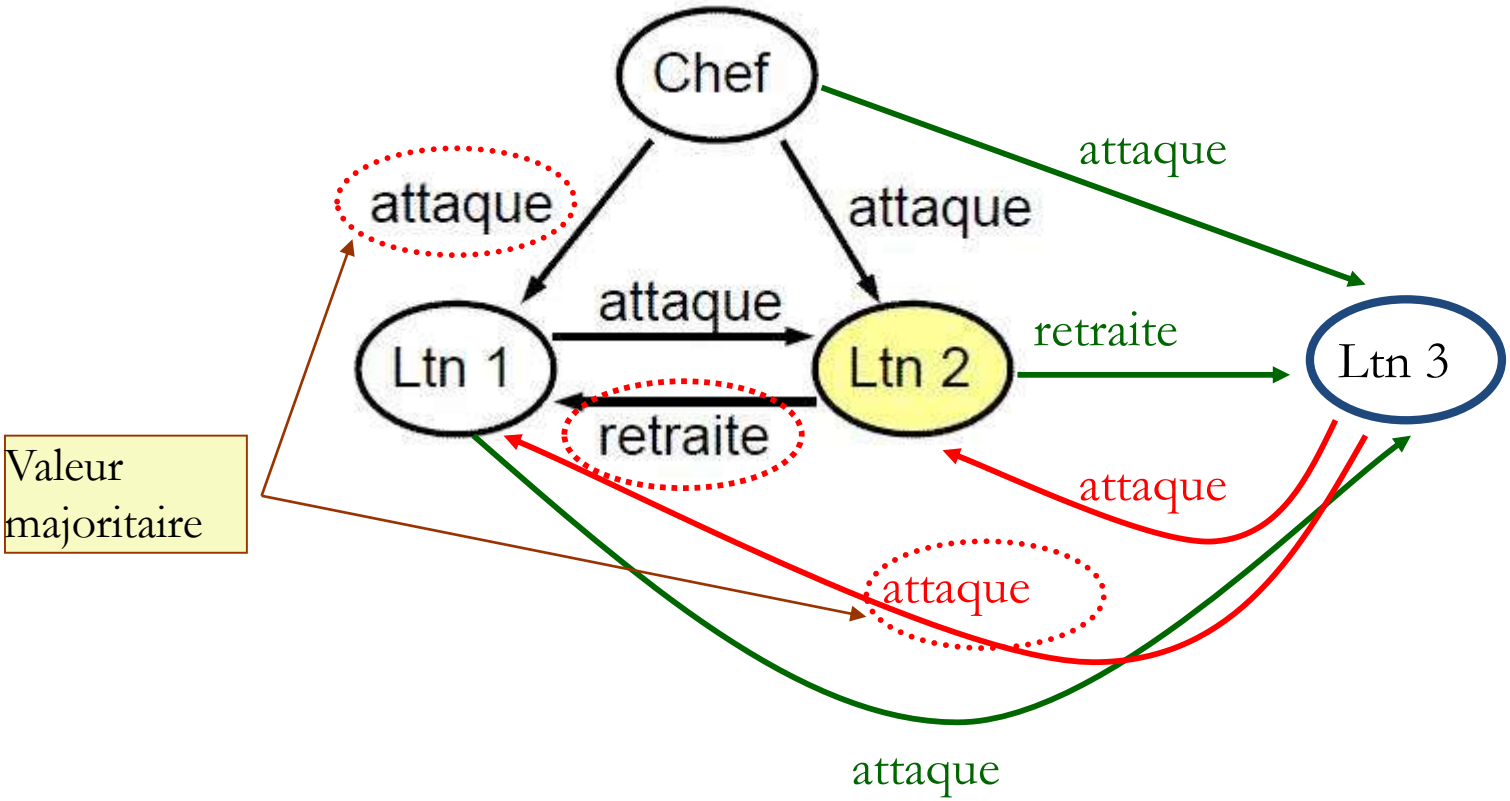
Problème des généraux byzantins

- Avec un troisième lieutenant (donc 4 généraux), le lieutenant 1 recevrait un message de plus qui permettrait de savoir quel est le bon ordre.



Problème des généraux byzantins

- Avec un troisième lieutenant (donc 4 généraux), le lieutenant 1 recevrait un message de plus qui permettrait de savoir quel est le bon ordre.



Consensus : synchrone, fautes byzantines

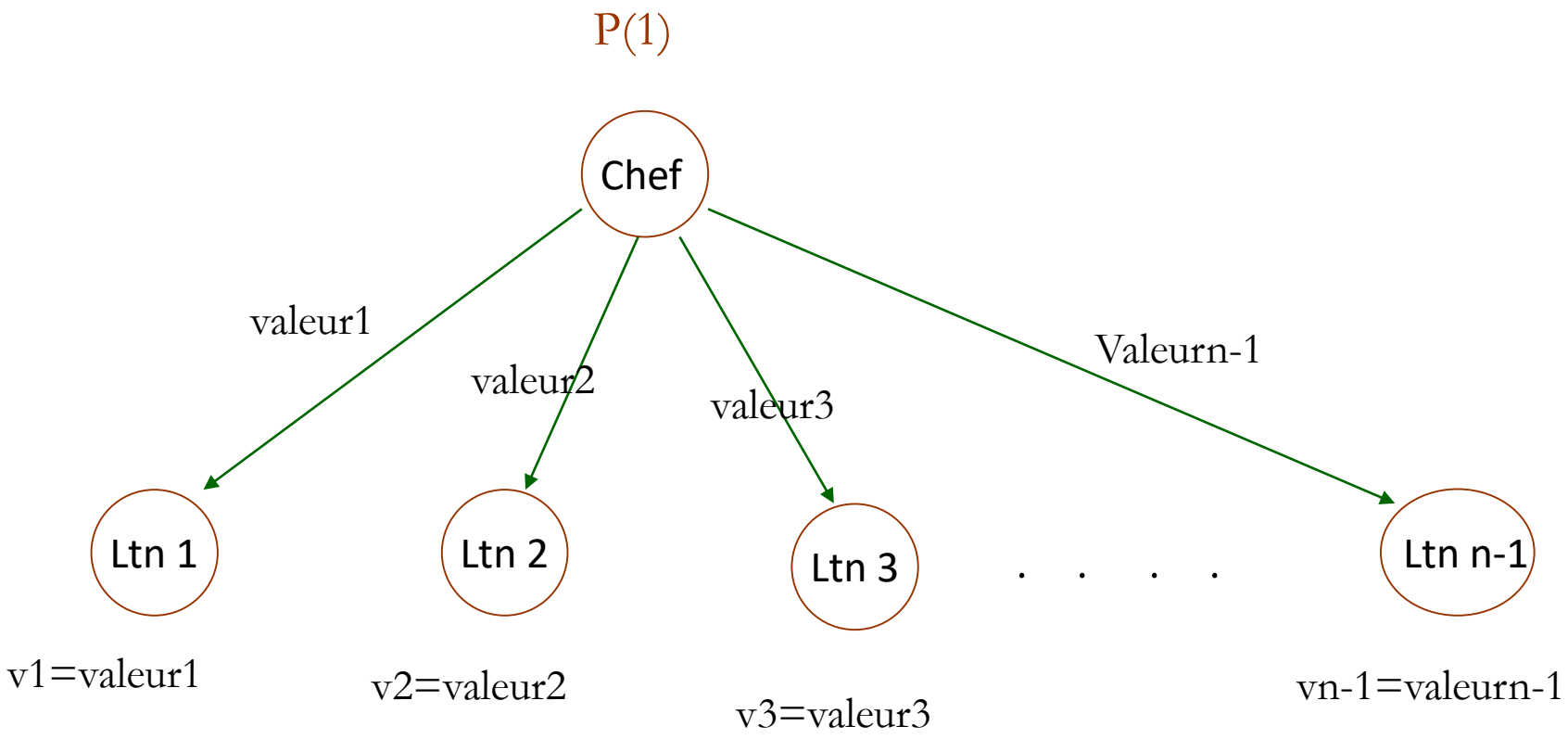
- **Algorithme des généraux byzantins** correspond à une réalisation de consensus dans un contexte de fautes byzantines.
- **Condition :** n = nombre de généraux, m = nombre de traîtres.
    - Il faut : **n >= 3m+1.**

- **Algorithme des généraux : P(m) : algorithme pour *m* traîtres au plus.**
    - Chaque lieutenant *Li* possède une valeur locale *vi* qui est l'ordre qu'il a décidé de suivre, déterminé en fonction de ce qu'il a reçu des autres généraux.
    - S'appliquera de manière récursive : **P(m–1), P(m–2)** jusqu'à atteindre **P(0).**
    - Pour un P(x), un processus joue le rôle de chef et diffuse sa valeur à tous les autres lieutenants.

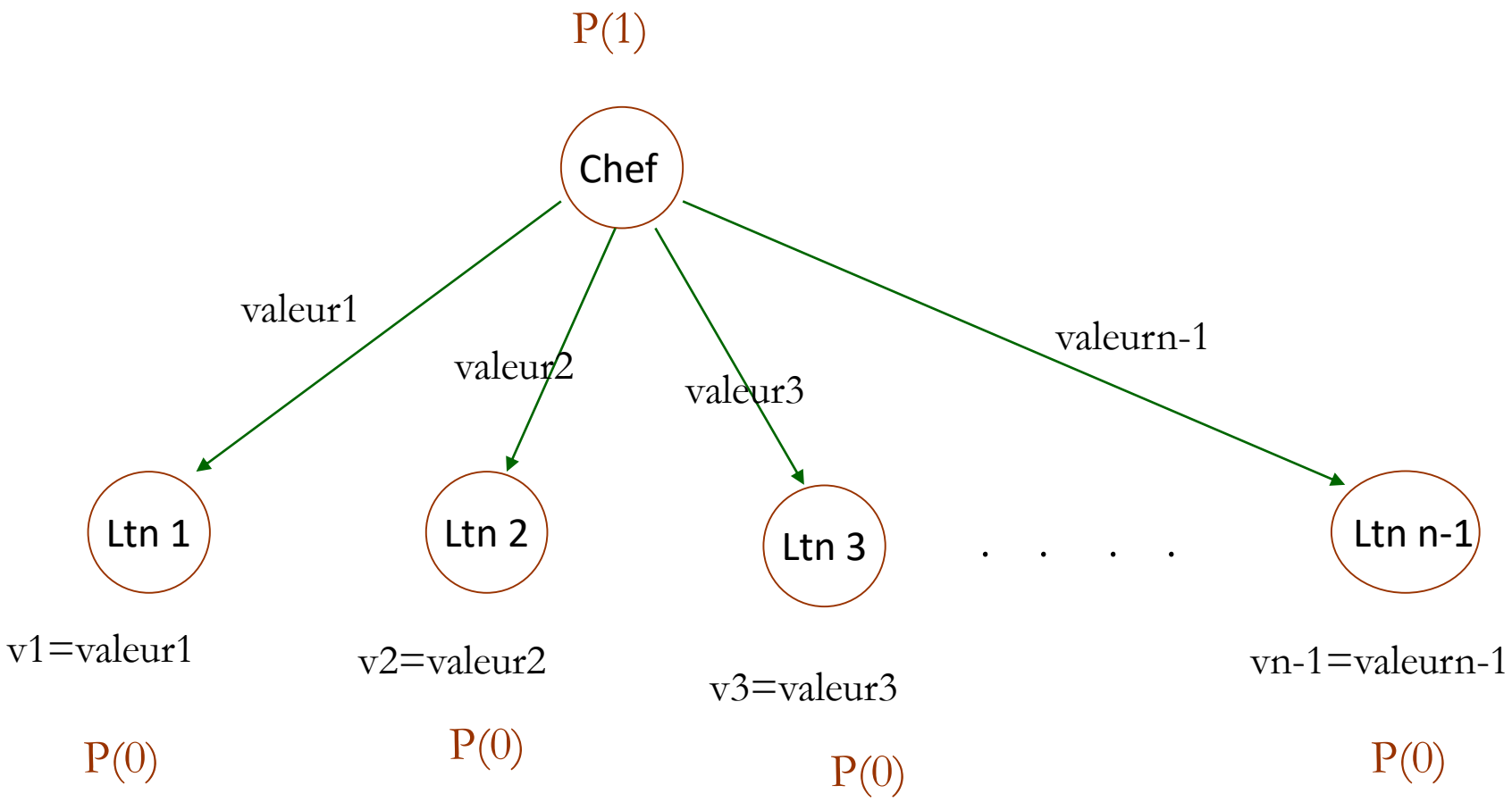
Consensus : synchrone, fautes byzantines

- **P(0) : pas de traître**
  - 1. Le chef diffuse sa valeur à tous les lieutenants *Li*.
  - 2. Pour tout *i*, à la réception de la valeur du chef, chaque lieutenant positionne *vi* à la valeur reçue.
    - Si ne reçoit rien, *vi* = DEF (absence de valeur).
- **P(m) : m traîtres (m > 0)**
  - 1. Le chef diffuse sa valeur à tous les lieutenants *Li*.
  - 2. Pour chaque *Li*, à la réception de la valeur du chef, chaque lieutenant positionne *vi* à la valeur reçue.
    - Si ne reçoit rien, *vi* = DEF (absence de valeur).
  - 3. Chaque lieutenant exécute **P(m–1)** en jouant le rôle du chef : diffusion de sa valeur aux (**n–2**) autres lieutenants.
  - 4. Une fois reçu la valeur *vj* de chacun des lieutenants (*vj* = DEF si rien reçu de *Lj*), détermine la valeur locale *vi*.
    - *vi* = **valeur majoritaire** parmi les *vj* ou DEF si pas de majorité.

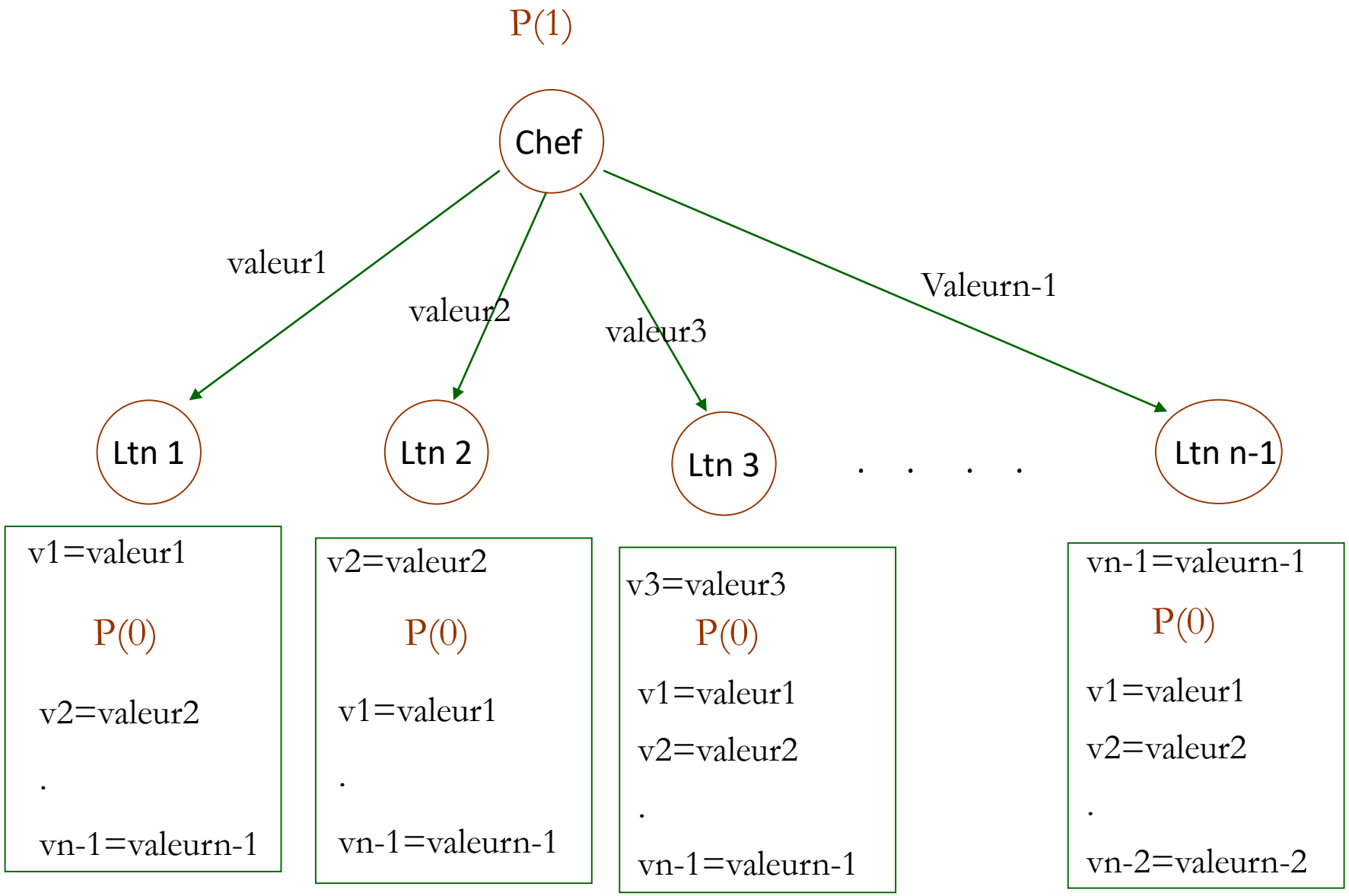
Avec un seul traitre : m=1



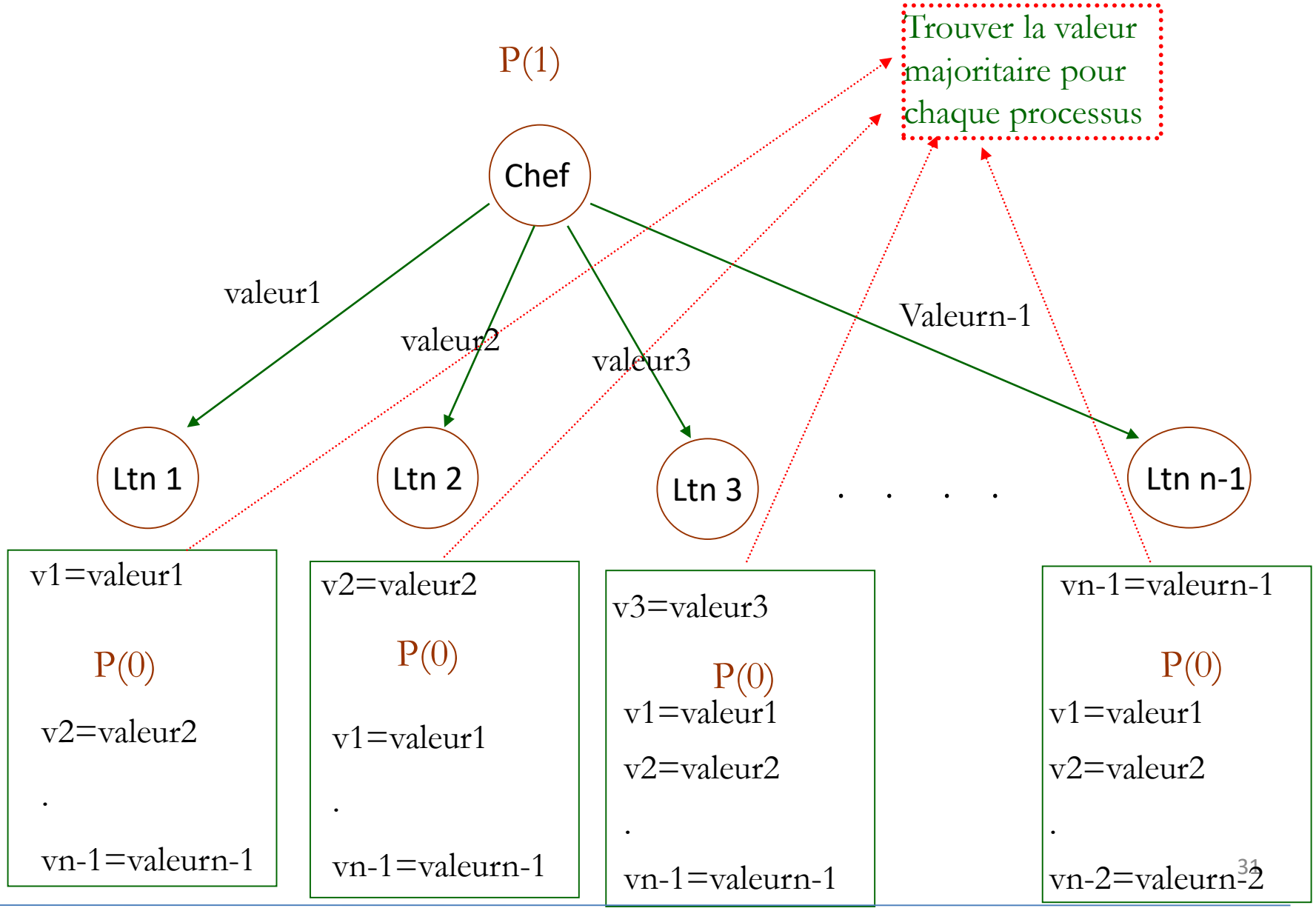
Avec un seul traitre : m=1



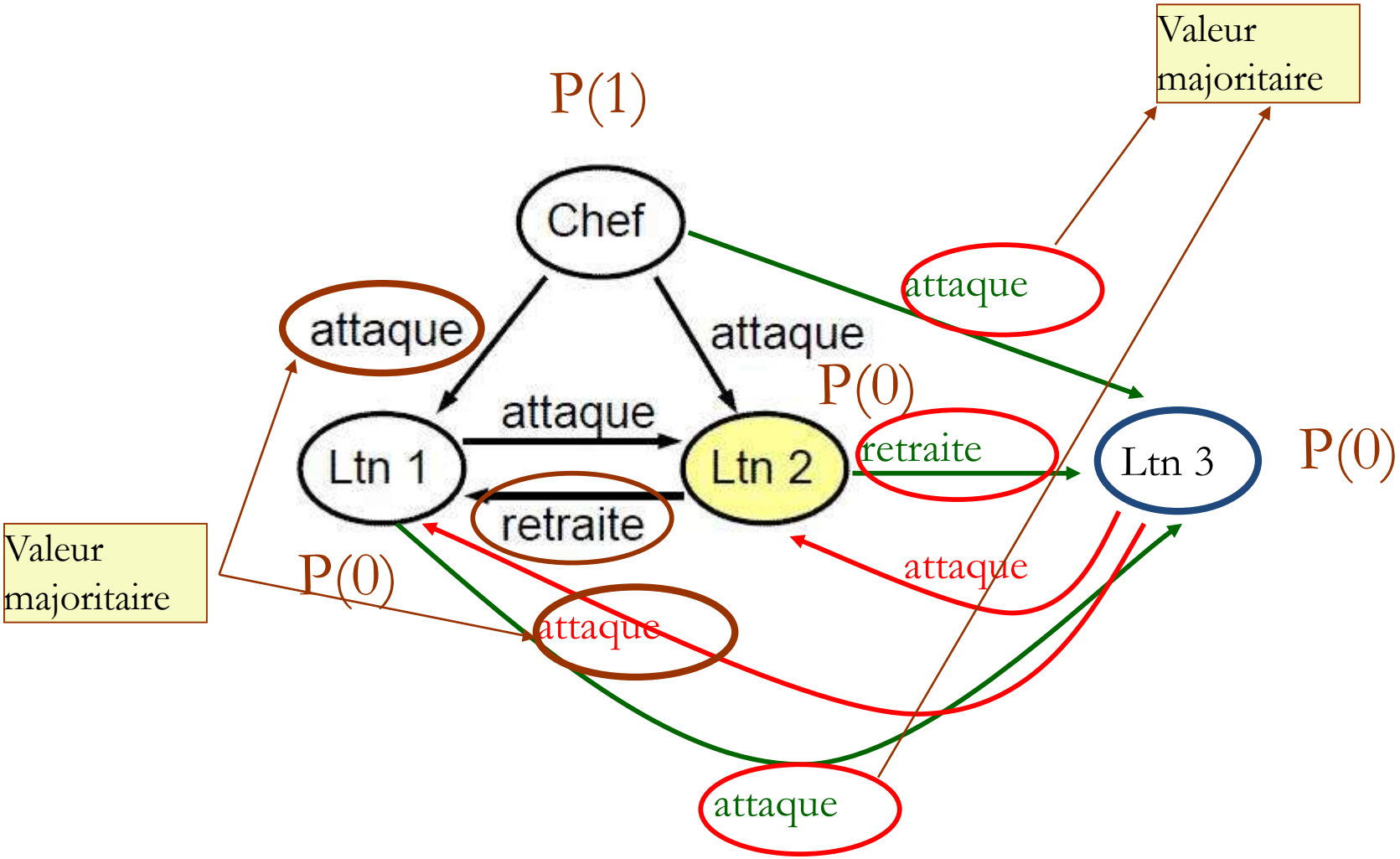
Avec un seul traitre : m=1



Avec un seul traitre : m=1



Avec un seul traitre : m=1



Consensus : résumé

- Selon le contexte de faute et le modèle temporel (avec communications fiables dans tous les cas) :
  - **Asynchrone :**
    - Impossibilité de définir des algorithmes assurant d'aboutir à un consensus dans tous les cas (franche ou byzantine).
      - Problème principal : impossibilité de différencier un processus planté d'un message lent.
  - **Synchrone :**
    - Panne franche : consensus atteignable systématiquement.
    - Panne byzantine : consensus assuré si on ne dépasse pas un seuil de processus au comportement byzantin en proportion du nombre global de processus.
      - n = nombre de processus, m = nombre de processus fautifs.
        - » Il faut : **n > 3m.**