# Systèmes et algorithmes répartis

## Consensus, détecteur de défaillances

Philippe Quéinnec

Département Informatique et Mathématiques Appliquées
ENSEEIHT

10 juillet 2018

# plan



# Plan



# Le consensus

## Définition

Soit un ensemble de processus $p_1, \ldots, p_n$ reliés par des canaux de communication.
Initialement : chaque processus $p_i$ propose une valeur $v_i$.
À la terminaison de l'algorithme : chaque $p_i$ décide d'une valeur $d_i$.

- Accord : la valeur décidée est la même pour tous les processus corrects
- Intégrité : tout processus décide au plus une fois (sa décision est définitive)
- Validité : la valeur décidée est l'une des valeurs proposées
- Terminaison : tout processus correct décide au bout d'un temps fini

1. *The Byzantine Generals Problem*, Leslie Lamport, Robert Shostak and Marshall Pease. ACM Trans. on Programming Languages and Systems. 1982.

# Remarques

## valeur décidée

- Les valeurs proposées ne sont pas nécessairement distinctes. 0/1 comme seules valeurs est équivalent
- La valeur décidée n'est pas nécessairement une valeur majoritaire, ni celle d'un processus correct

## Démarrage

Quand un processus démarre-t-il l'algorithme de consensus ?

- Démarrage simultané (à une heure donnée)
- Démarrage initié par l'un des processus : diffusion d'un message d'initialisation de l'algorithme.
  (Attention aux propriétés de cette diffusion : fiable, ordonnée)
- Sur réception d'un 1$^{er}$ message de l'algorithme : ça complique !

Les trois formes sont équivalentes.

# Variantes

## Consensus uniforme

- Accord uniforme : la valeur décidée est la même pour tous les processus (corrects ou ultérieurement fautifs) qui décident

Un processus peut décider puis devenir incorrect.

## $k$-consensus

- $k$ Accord : au plus $k$ valeurs distinctes sont décidées pour l'ensemble des processus corrects

Consensus basique : $k = 1$

## Consensus approximatif

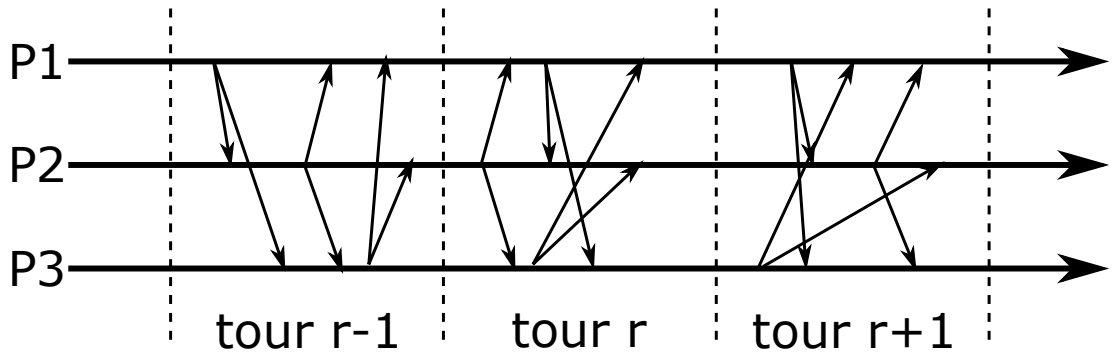
- $\epsilon$-Accord : les valeurs décidées par les processus corrects doivent être à distance maximale $\epsilon$ l'une de l'autre.

# Modèle temporel

## Synchrone

borne supérieure connue sur le temps de transmission et sur l'avancement des processus.

Usuellement, algorithmes fonctionnant par tours, synchronisés sur tous les processus.



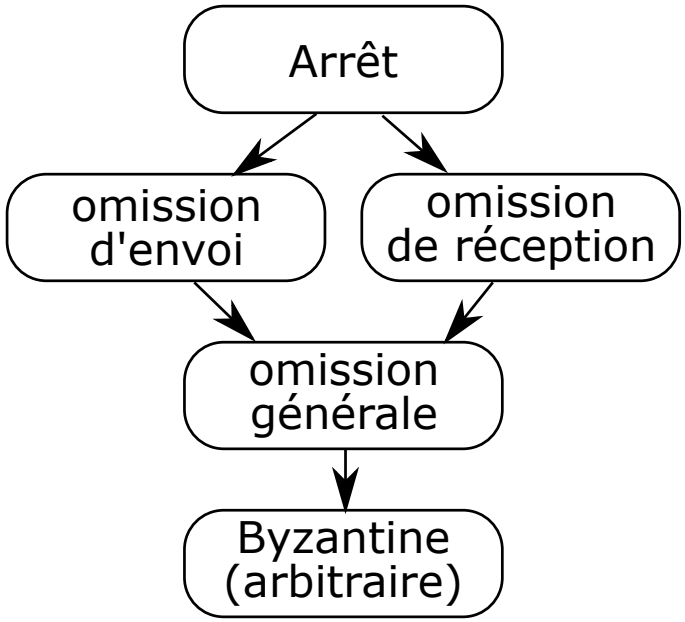

## Asynchrone

Pas de borne connue : avancement arbitrairement lent des processus et du réseau.

Modèle moins contraint, plus réaliste (mais plus difficile)

# Défaillances d'un processus

- Arrêt (*crash failure* ou panne franche) : le processus fonctionne correctement jusqu'à un point où il cesse définitivement d'agir.
- Omission
  - omission en émission : le processus omet certaines émissions qu'il aurait dû faire, ou cesse définitivement.
  - omission en réception : le processus ignore certains messages en réception, ou cesse définitivement.
- Arbitraire (*byzantine failure*) : le processus ment (par omission ou par contenu arbitraire des messages envoyés)




1. *Fault-Tolerant Broadcasts and Related Problems*, Vassos Hadzilacos and Sam Toueg. In *Distributed Systems*. 1993.

# Communications

## Défaillance

- réseau fiable : tout message finit par arriver
- perte : certains messages n'arrivent jamais
- ordre : respect de l'ordre d'émission ou d'un autre ordre
- arbitraire : duplication, modification du contenu...

## Hypothèse de réseau fiable

Les défaillances réseau en asynchrone peuvent être modélisées par des défaillances de site $\Rightarrow$ on suppose le réseau fiable.

# Utilité du consensus

Le consensus est un outil générique pour la tolérance aux fautes :

- Un système informatique est une machine à état :
  *(nouvel état, sorties)* $\leftarrow$ *fonction(état courant, entrées)*
- Assurer la disponibilité = réplication en $n$ copies
- Transparence = équivalence avec une seule copie
- Consensus pour ordonner identiquement les entrées
  + si non déterministe, consensus pour décider identiquement des réponses sur les $n$ copies

1. *The Implementation of Reliable Distributed Multiprocess Systems*, Leslie Lamport. Computer Networks. 1978.

# Universalité

## Spécification séquentielle

Un objet possède une spécification séquentielle si ses comportements corrects sont exprimables par des séquences (= des traces) de ses opérations.

## Universalité du consensus

Le consensus suffit pour implanter n'importe quel objet possédant une spécification séquentielle.

- En communication par message : consensus + diffusion générale (anonyme)

1. *Wait-Free Synchronization*, Maurice Herlihy. ACM Trans. on Programming Languages and Systems. 1991.

# Problèmes réalisables avec le consensus

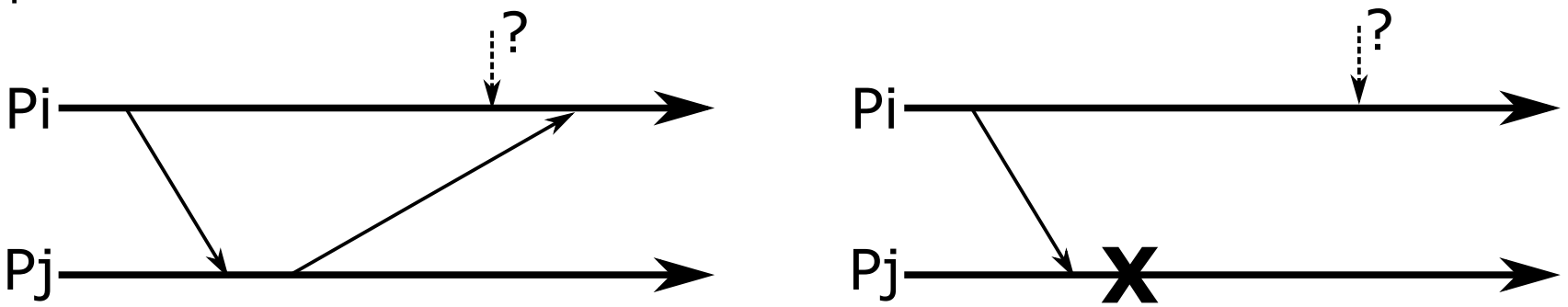
- Élection d'un leader = accord de tous sur un processus
- Diffusion fiable avec terminaison = tous les processus corrects délivrent un même message (éventuellement vide si l'émetteur s'est arrêté)
- Diffusion uniforme = tout les processus (corrects ou pas) délivrent ou aucun
- Construction de groupes
- Commit (validation) de transaction distribuée
- Calcul d'une fonction globale portant sur l'ensemble des sites

# Impossibilité du consensus en asynchrone avec arrêt

## Résultat d'impossibilité (FLP85)

Le consensus est impossible à réaliser dans un système asynchrone où un seul processus peut subir une défaillance d'arrêt.

Intuitivement, il est impossible de distinguer un processus lent d'un processus arrêté.




1. *Impossibility of Distributed Consensus with One Faulty Process*, Michael Fischer, Nancy Lynch and Michael Paterson. Journal of the ACM. April 1985.

# Impossibilité du consensus en synchrone avec perte de message

**Résultat d'impossibilité (Gray, 1978)**

Le consensus est impossible à réaliser dans un système synchrone où les messages peuvent être arbitrairement perdus.

Intuition de la preuve : le dernier message avant décision peut être perdu sans changer la décision $\Rightarrow$ il était inutile. On le supprime, et on recommence le raisonnement.

(Piège : la perte de message peut être modélisée par une défaillance d'omission de *n'importe quel* processus. Le consensus est faisable en synchrone avec omission s'il y a $< n/2$ sites défaillants.)

# Résultats d'impossibilité de réalisation du consensus

## Processus communiquant par mémoire partagée

| Défaillance / modèle | Arrêt de processus |
|---|---|
| synchrone | ∃ solution |
| asynchrone | impossible |

## Processus communiquant par messages

| Défaillance / modèle | Arrêt de processus | Omission | Byzantine | Perte de message |
|---|---|---|---|---|
| synchrone | ∃ solution | ∃ solution | ∃ solution | impossible |
| asynchrone | impossible | impossible | impossible | impossible |

# Contourner FLP

## Affaiblir le problème

- Terminaison probabiliste
- $k$-consensus
- Consensus approximatif ($\epsilon$-consensus)
- *Best effort*

## Renforcer le système

- Système partiellement synchrone
- Système synchrone *suffisamment longtemps*
- Détecteur de défaillances

# Réalisabilité du consensus

| défaillance | synchrone | asynchrone |
|---|---|---|
| non | faisable | faisable |
| arrêt | faisable<br>$f$ pannes $< n$ processus<br>$\Omega(f+1)$ tours | impossible |
| omission | faisable<br>$f$ pannes $< n/2$ processus | impossible |
| byzantine | faisable<br>$f \leq \lfloor (n-1)/3 \rfloor$ processus<br>$\Omega(f+1)$ tours | impossible |

- $k$-consensus : faisable en asynchrone/arrêt avec $f < k < n$
- Consensus approximatif : faisable en asynchrone/arrêt avec $5f + 1 \leq n$

# Plan



# Réalisation en absence de pannes

## Principe

Tous les processus diffusent leur valeur, chacun garde la plus petite reçue.

Synchrone $\Rightarrow$ borne supérieure de communication $\Delta \Rightarrow$ décision en un tour et $n$ diffusions.

```
Processus P_i(v_i),  0 ≤ i < n
local V_i
on start :
    V_i ← {v_i}
    envoyer(v_i) à tous les autres
on réception(v) :
    V_i ← V_i ∪ {v}
on time Δ :
    décider min(V_i)   (toute fonction déterministe)
```

# Réalisation en défaillance d'arrêt

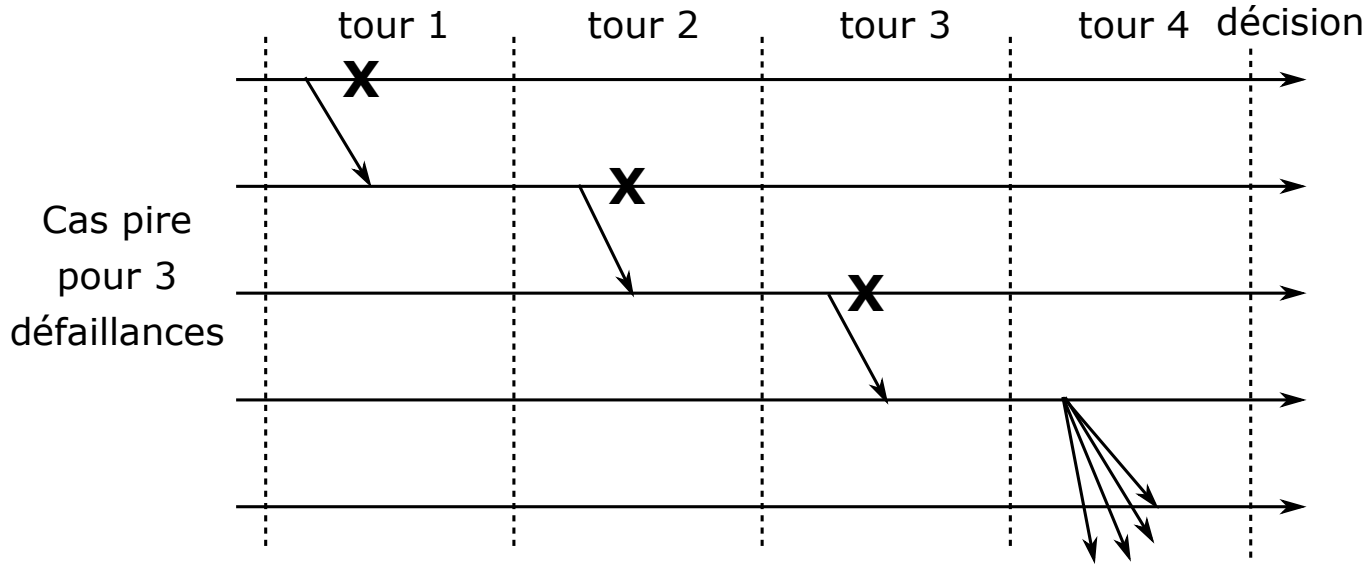
Tolérance à $f$ défaillances ($f < n$).

## Principe

Au $i$-ième tour, le processus $i$ diffuse sa valeur. Après $f+1$ tours, on est sûr qu'au moins un processus correct a diffusé une valeur reçue par au moins $n-f$ processus corrects.

```
Processus P_i(v_i),  0 ≤ i < n
local x
on start :
    x ← v_i
on réception(v) :
    x ← v
on time i × Δ, i ≤ f :     // au i-ième tour pour P_i
    envoyer(x) à tous les autres
on time (f + 1) × Δ :
    décider x
```

# Réalisation en défaillance d'arrêt




- $f + 1$ tours, $f + 1$ diffusions
- décision simultanée
- non équitable : les valeurs des processus $f + 1, \ldots, n$ ne sont pas considérées $\Rightarrow$ ajouter un tour préliminaire :

  `diffuser(`$v_i$`)`
  $x \leftarrow$ `l'une des valeurs reçues dans ce tour`

# Algorithme équitable à $f+1$ tours

## Principe

À chaque tour, chaque processus diffuse la plus petite valeur qu'il connaît (uniquement si elle a changé).

```
Processus P_i(v_i), 0 ≤ i < n
local x, prevx, received
on start :
    x ← v_i, prevx ← ⊥
on réception(v) :
    received ← received ∪ {v}
on time k × Δ, 0 ≤ k ≤ f :          // à chaque tour
    prevx ← x
    x ← min(received ∪ {x})
    received ← ∅
    si x ≠ prevx alors diffuser(x)
on time (f + 1) × Δ:
    décider x
```

# Nombre optimal de tours

## Borne inférieure

Il n'existe pas d'algorithme synchrone basé sur des tours qui résolve le consensus avec $f$ défaillances d'arrêt en moins de $f + 1$ tours.

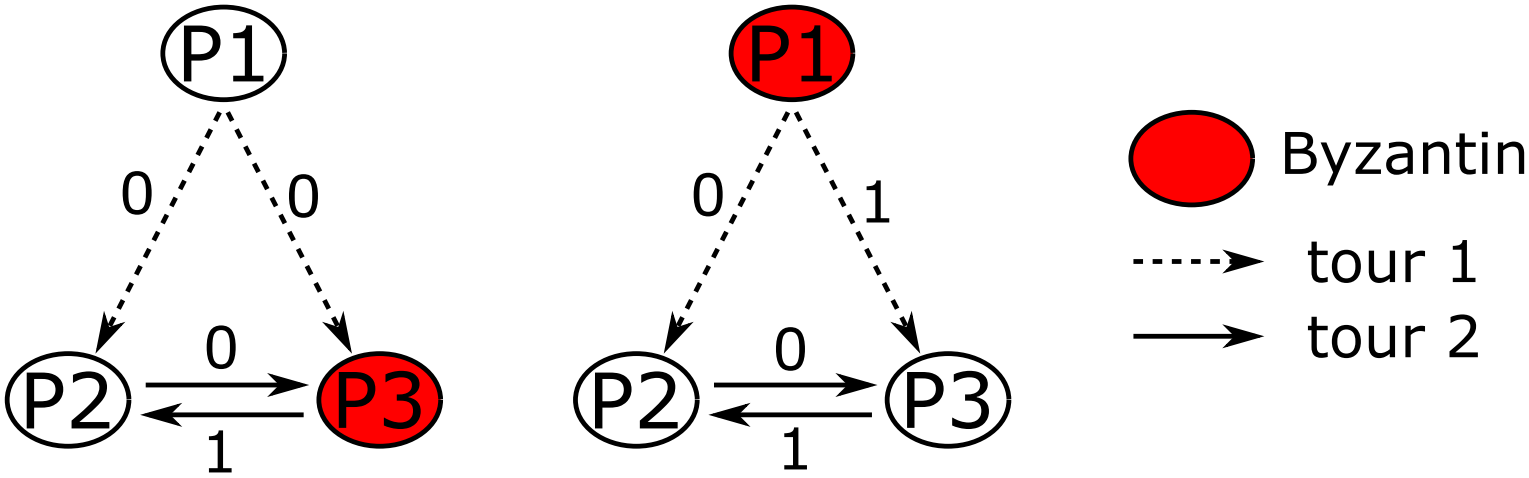
Le pire cas est qu'un seul processus défaille à chaque tour.

## Existence

La borne "$f + 1$ tours" est atteignable.

1. *A Lower Bound for the Time to Assure Interactive Consistency*, Michael Fischer and Nancy Lynch. Information Processing Letters. 1982.

# Impossibilité du consensus à 3 avec une panne byzantine




- Panne byzantine = le processus ment
- $P_2$ ne peut pas distinguer entre les deux scénarios
- Des échanges supplémentaires ne changent rien
- $P_2$ ne peut pas nécessairement décider identiquement à l'autre processus correct

# Algorithme avec défaillances byzantines

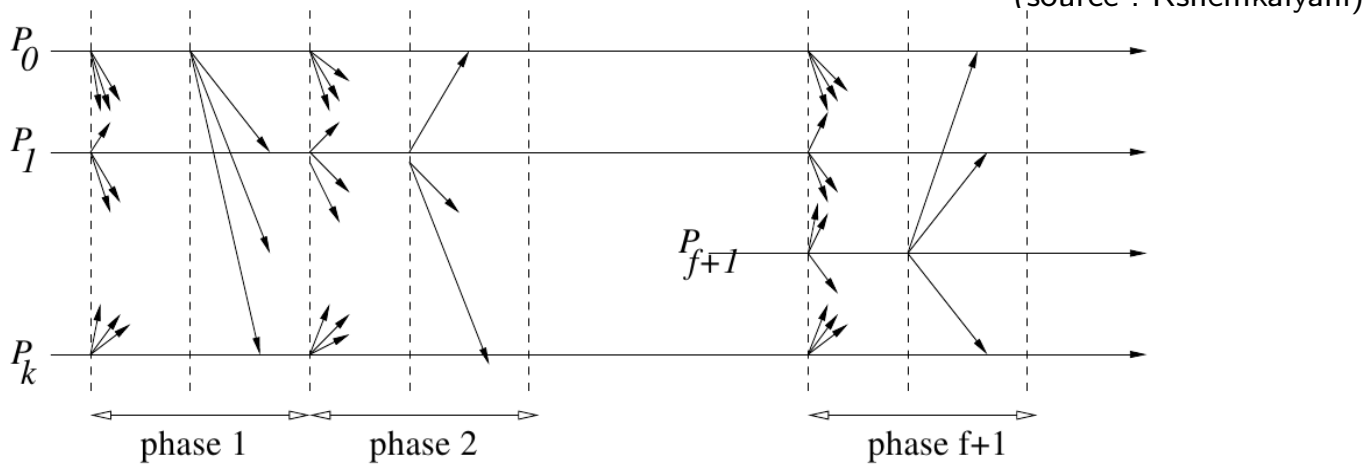
Le consensus en synchrone avec défaillance byzantine est impossible pour $n \leq 3f$.

Le consensus en synchrone avec défaillance byzantine est possible si $f \leq \lfloor \frac{n-1}{3} \rfloor$.

# Algorithme avec “roi de phase” (*Phase King*)

- $f + 1$ phases, chaque phase a un unique roi, fixé a priori
- Chaque phase contient deux tours :

(source : Kshemkalyani)




  - Tour 1 : chaque site diffuse son estimation à tous.
    Chaque site reçoit les valeurs proposées puis détermine si une valeur est proposée par une majorité qualifiée ($> n/2 + f$), ou une majorité simple ($> n/2$)
  - Tour 2 : le roi fixe son estimation à sa valeur majoritaire (sa propre valeur si pas de majorité) et la diffuse.
    Chaque site fixe sa nouvelle estimation à la valeur reçue en tour 1 avec majorité qualifiée, ou sinon à la valeur reçue du roi.

# Algorithme avec “roi de phase” – justification

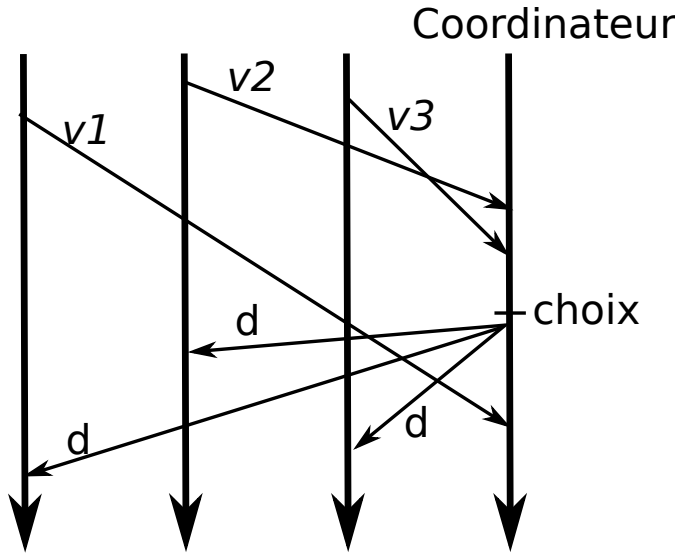
$f+1$ phases, $(f+1)(n+1)(n-1)$ messages,
$f < \lceil n/4 \rceil$ défaillances

- Parmi les $f+1$ phases, au moins une où le roi est correct
- Dans cette phase, les processus corrects obtiennent la même estimation que le roi : soit $p_i$ et $p_j$ corrects :
  - ou $p_i$ et $p_j$ ont chacun une majorité qualifiée ($> n/2 + f$) et le roi a alors aussi cette même estimation
  - ou $p_i$ a une majorité qualifiée ($> n/2 + f$) et $p_j$ utilise la valeur du roi ($> n/2$)
  - ou $p_i$ et $p_j$ utilisent l'estimation du roi
- Si tous les processus corrects ont la même estimation au début d'une phase, ils garderont cette même valeur à la fin (même si le roi est byzantin).

# Consensus avec coordinateur

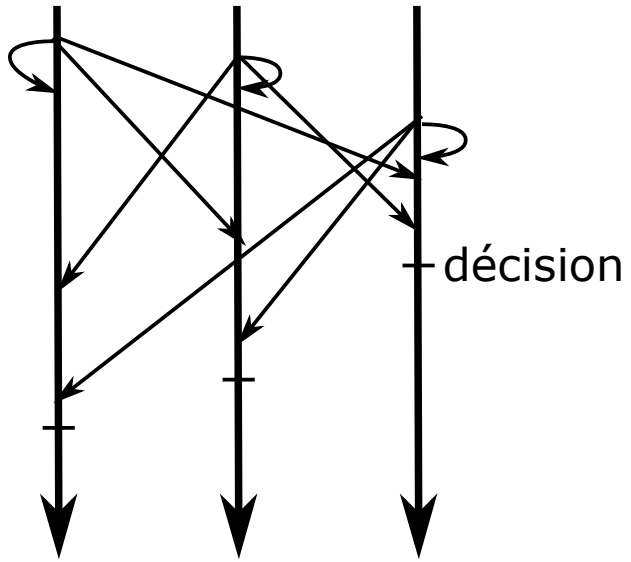
Hypothèse : système asynchrone fiable, pas de défaillance

- Chaque processus envoie sa valeur à un coordinateur désigné à l'avance
- Au bout d'un certain temps (après avoir reçu au moins une valeur), le coordinateur choisit une valeur $d$
- Le coordinateur envoie $d$ à tous les processus (diffusion fiable)
- Chaque processus décide $d$
- $\Rightarrow$ tous les processus décident identiquement en temps fini non borné



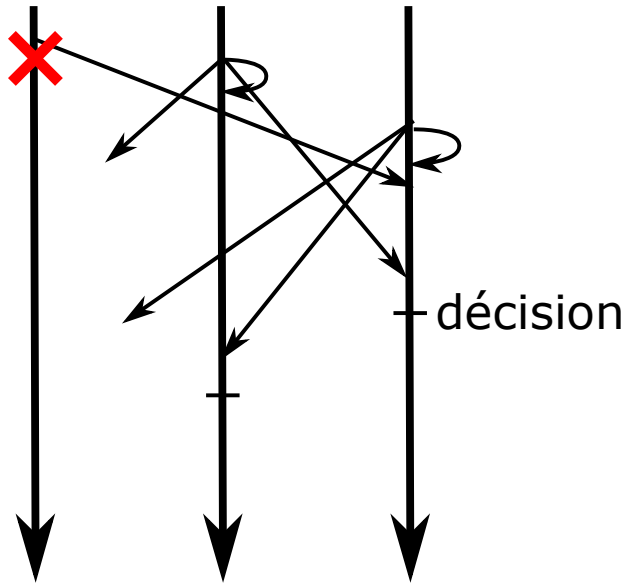
# Consensus symétrique

- Chaque processus diffuse sa valeur à tous
- Quand un processus a reçu toutes les valeurs, il applique un algorithme déterministe (p.e. min) pour choisir la valeur de décision
- Tous les processus utilisent le même algorithme
- $\Rightarrow$ tous les processus décident identiquement en temps fini non borné



# $k$-consensus avec arrêt de processus

- Chaque processus envoie sa valeur à tous les autres
- Quand un processus a reçu (au moins) $n - (k - 1)$ valeurs, il décide le min.




Si $f < k < n$ alors :

- Terminaison en temps fini (non borné) pour les processus corrects
- Au plus $f$ valeurs non reçues par tous $\Rightarrow$ au plus $f + 1$ décisions distinctes

# Détecteur de défaillances : Motivation

## Résultat d'impossibilité (FLP85)

Le consensus est impossible à réaliser dans un système asynchrone où un seul processus peut subir une défaillance d'arrêt.

- Le consensus en asynchrone est impossible car on ne sait pas distinguer un processus défaillant (arrêté) d'un processus lent.
- On va supposer l'existence d'un détecteur de défaillances, qui indique si un processus est fautif ou non.
- FLP $\Rightarrow$ un détecteur parfait est impossible.
- Il existe des détecteurs imparfaits (i.e. qui peuvent se tromper) qui suffisent pour réaliser le consensus !
- FLP $\Rightarrow$ ces détecteurs imparfaits sont impossibles mais on peut en construire des approximations réalistes.

# Détecteur de défaillances

## Définition

- Un détecteur de défaillances est un service réparti composé de détecteurs locaux à chaque processus (site).
- Un détecteur fournit à son processus local une liste des processus qu'il suspecte d'être défaillants.
- Les détecteurs locaux coopèrent (ou pas) pour établir cette liste.

- Propriétés :
  - Complétude : peut-on ne pas suspecter un processus défaillant ?
  - Exactitude : peut-on suspecter un processus correct ?
- Équivalent à un oracle (éventuellement imparfait)

1. *Unreliable Failure Detectors for Reliable Distributed Systems*, Tushar Chandra and Sam Toueg. Journal of the ACM. March 1996.

# Complétude

## Complétude (*completeness*)

- Complétude forte : tout processus défaillant finit par être suspecté par tout processus correct
- Complétude faible : tout processus défaillant finit par être suspecté par un processus correct

Complétude faible et complétude forte sont équivalentes :

- En complétude faible, tout processus défaillant finit par être détecté par au moins un processus
- Périodiquement, chaque processus diffuse sa liste de processus suspectés
- Alors tous les processus finiront par obtenir l'information de suspicion = complétude forte
- (hypothèse : la diffusion est fiable, ce qui est réalisable en asynchrone avec défaillance d'arrêt)

# Exactitude

## Exactitude permanente (*accuracy*)

- Exactitude forte : aucun processus correct n'est jamais suspecté par aucun autre processus correct
- Exactitude faible : il existe un processus correct qui n'est jamais suspecté par aucun autre processus correct

## Exactitude inévitable

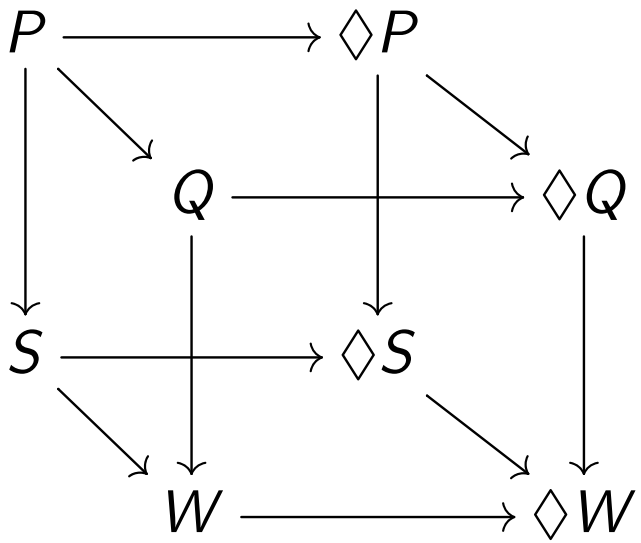
- Exactitude finalement forte : au bout d'un certain temps, aucun processus correct n'est plus jamais suspecté par aucun autre processus correct
- Exactitude finalement faible : au bout d'un certain temps, il existe un processus correct qui n'est plus jamais suspecté par aucun autre processus correct

Finalement = inévitablement = *eventually*

# Classes de détecteurs de défaillances

| | Exactitude | | | |
|---|---|---|---|---|
| | forte | faible | finalement forte | finalement faible |
| Complétude forte | P | S | $\Diamond P$ | $\Diamond S$ |
| Complétude faible | Q | W | $\Diamond Q$ | $\Diamond W$ |

P = perfect, S = strong, W = weak



# Consensus avec détecteur parfait $P$

## Données

- $f$ = nombre de défaillances d'arrêt à tolérer
- pour chaque processus $p_i$, un vecteur $V_i$ contenant les valeurs proposées par les autres processus et connues de $p_i$
- $V_i[j]$ = valeur proposée par $p_j$ telle que connue par $p_i$
- Initialement $V_i[i] = v_i$ (valeur proposée par $p_i$) et $V_i[j] = \perp (j \neq i)$

# Consensus avec détecteur parfait : principe

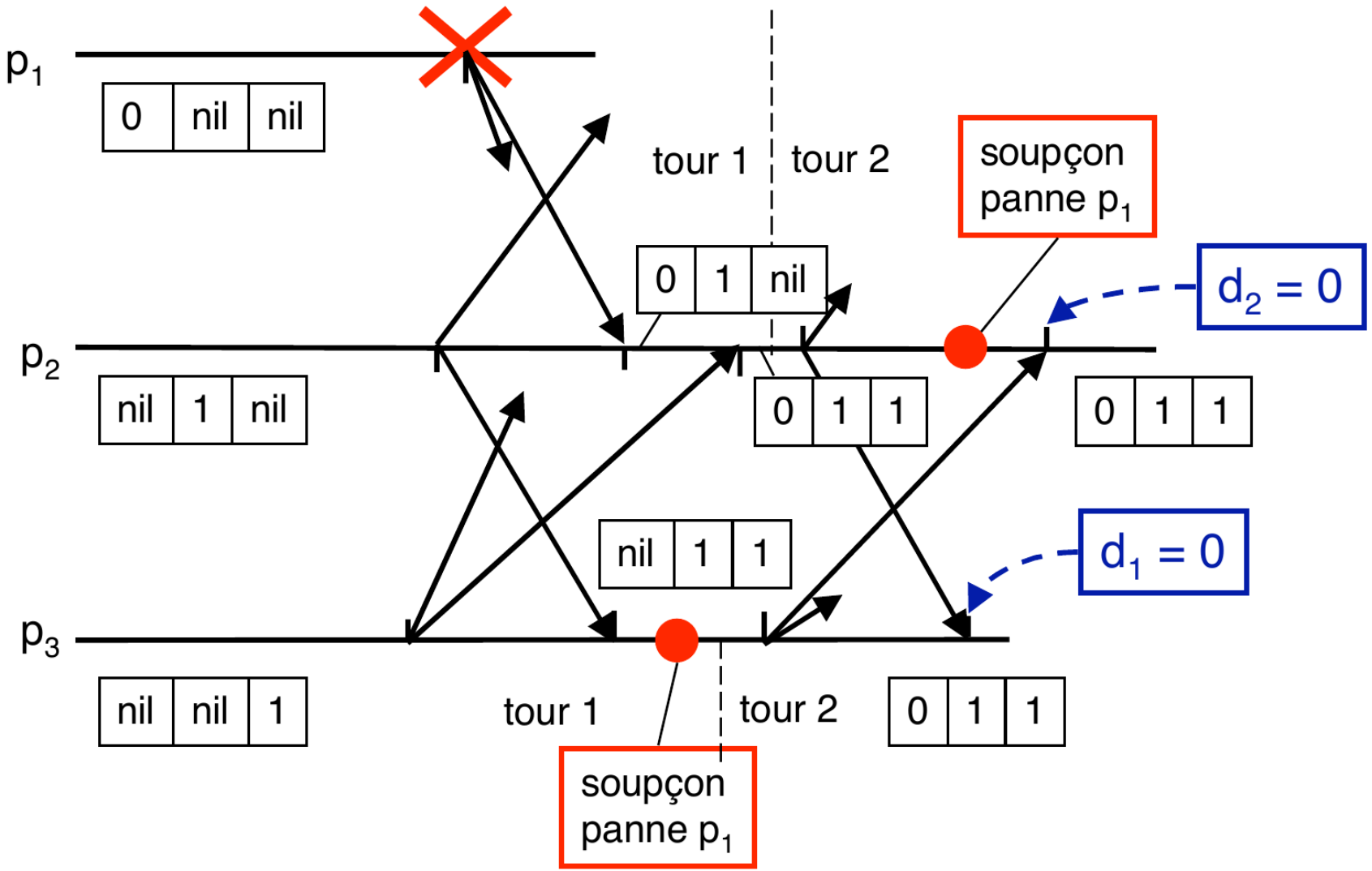
Deux phases :

- $f + 1$ tours asynchrones : à chaque tour :
  - chaque processus envoie aux autres son vecteur $V_i$
  - chaque processus met à jour son vecteur $V_i$ avec les nouvelles valeurs apprises (celles telles que $V_i[j] = \perp$)
  - quand un processus a reçu un message de tous les processus non suspectés par son détecteur $P$, il passe au tour suivant.
- Après $f + 1$ tours, le processus décide la première valeur non $\perp$ de son vecteur.

Remarques :

- il suffit de transmettre les nouvelles valeurs apprises au tour précédent $\Rightarrow$ un processus n'envoie une valeur qu'une seule fois.
- $V_i[j]$ est stable : mis à jour au plus une fois et plus jamais modifié si non $\perp$.

# Consensus avec détecteur parfait : exemple



# Consensus avec détecteur parfait : sûreté – Accord

*La valeur décidée est la même pour tous les processus corrects.*
Supposons que $p_i$ et $p_j$ décident deux valeurs différentes $\Rightarrow$ au tour $f+1$, $V_i$ et $V_j$ sont différents. Soit $x$ une valeur de $V_i$ non présente dans $V_j$.

- $x$ a été transmis à $p_i$ par un processus correct $p_k$, mais $p_j$ n'a pas attendu ce message car il suspectait $p_k$.
  Impossible : exactitude du détecteur parfait.
- $x$ a été transmis à $p_i$ au dernier tour par un processus $p_k$ qui s'est arrêté avant de transmettre $x$ à $p_j$ (si pas au dernier, $p_i$ aurait transmis la valeur à $p_j$).
  $p_k$ connaissait cette valeur depuis un tour $t$. Nécessairement, $t$ = le tour précédent ($f$), car sinon $p_j$ aurait attendu et reçu un message de $p_k$ avec la valeur $x$ ($p_k$ correct aux tours $< f+1$).
  Par récurrence, puisque $p_j$ ne connaît pas $x$, le processus qui devait le lui envoyer s'est chaque fois arrêté avant.
  $\Rightarrow$ $f+1$ défaillances : impossible

# Consensus avec détecteur parfait : sûreté – intégrité, validité

- *Intégrité : tout processus décide au plus une fois.*
  Unique point de décision après $f + 1$ tours.

- *Validité : la valeur décidée est l'une des valeurs proposées.*
  Les vecteurs $V_i$ sont constitués de valeurs connues par les autres sites, et au départ il n'y a que les $V_i[i]$ qui contiennent les valeurs proposées (pas d'invention de valeurs).

# Consensus avec détecteur parfait : vivacité – terminaison

*Terminaison : tout processus correct décide au bout d'un temps fini*
Supposons qu'un processus $p_i$ ne décide pas. Alors il est bloqué dans un tour en attente d'un message provenant d'un processus $p_j$ :

- $p_j$ est correct. Alors son message finira par arriver (en temps fini non borné) ;
- $p_j$ est défaillant. Alors il finira par être suspecté (complétude du détecteur parfait) et $p_i$ ne l'attendra plus.

# Consensus avec détecteur $S$

On sait qu'un processus correct n'est jamais suspecté. Faire $n$ tours pour être sûr de le voir.

- $n-1$ tours identiques à l'algorithme précédent.
  Au $n-1$-ième tour, chaque processus a un vecteur, mais tous les vecteurs des processus corrects ne sont pas nécessairement identiques car des processus corrects ont pu être suspectés.
- tour supplémentaire : chaque processus diffuse son vecteur et attend un vecteur de chacun des autres processus qu'il ne suspecte pas. Les $\perp$ reçus effacent de son vecteur les valeurs non $\perp$.
  À la fin, tous les processus corrects ont le même vecteur : des valeurs proposées par des processus corrects peuvent ne pas y être (processus suspectés à tort) mais au moins la valeur du processus correct jamais suspecté est présente.
- décision comme précédemment.

# Consensus avec détecteur $\Diamond S$

Pour tolérer $f$ défaillances, il faut $2f + 1$ processus.
Principe :

- Un coordinateur tournant. Chaque processus sert de coordinateur à tour de rôle.
- Exactitude finalement faible : il existe un moment où un processus correct sera à la fois non suspecté et coordinateur.
- Un coordinateur (non suspecté) qui réunit une majorité de processus corrects (possible car $n \geq 2f + 1$) peut décider d'une valeur.
- Il diffuse alors cette valeur qui est retenue par tous les processus corrects.

# Synthèse

| détecteur | nombre de défaillances | nombre de tours |
|---|---|---|
| $P$ | $n-1$ | $f+1$ |
| $S$ | $n-1$ | $n$ |
| $\Diamond S$, $\Diamond W$ | $\frac{n-1}{2}$ | fini non borné |

## Plus faible détecteur de défaillances

Le détecteur de défaillances $\Diamond W$ (complétude faible, exactitude finalement faible) est le plus faible détecteur de défaillances permettant de résoudre le consensus en asynchrone avec défaillance d'arrêt.

1. *The Weakest Failure Detector for Solving Consensus*. Tushar Chandra, Vassos Hadzilacos and Sam Toueg. Journal of the ACM. July 1996.

# Implantation des détecteurs de défaillances

Les détecteurs de défaillances $P$, $S$, $\Diamond S$, $\Diamond W$ ne sont pas réalisables en asynchrone avec défaillances d'arrêt (FLP), mais ils sont réalisables en supposant assez de synchronisme.
On se donne $\delta$ = délai maximal de transmission d'un message.

## Détecteur actif (*ping*)

Périodiquement, $p$ envoie un message à $q$ et attend un acquittement. En absence de réponse après $2\delta$, $p$ suspecte $q$.

## Détecteur passif (*heartbeat*)

Périodiquement (période $T$), $q$ diffuse un message "je suis vivant". Si à $T + \delta$ après le message précédent, le processus $p$ n'a rien reçu, il suspecte $q$.

Nécessite des horloges synchronisées, réalisables sous l'hypothèse synchrone d'un délai maximal de communication.

# Implantation des détecteurs de défaillances

Estimation de $\delta$ :

- Trop petite : fausses détections
- Trop grande : temps trop long avant détection

Si le système est effectivement asynchrone, il n'existe pas de $\delta$ qui évite les fausses détections, et on ne peut pas garantir l'exactitude même faible (tous supposés défaillants par au moins un autre), sauf à tuer les processus suspectés...

# Détecteur de défaillances – bilan

- Apport théorique : identifier le minimum nécessaire pour réaliser le consensus ; cadre unique de comparaison
- Apport pratique : isoler la résolution d'un problème réparti et la gestion de la détection des défaillances

# Conclusion sur le consensus

## Universalité

Consensus $\rightarrow$ tout objet (ayant une spécification séquentielle)

## Communication par messages, synchrone

- Consensus réalisable même avec défaillances complexes (mais alors très coûteux)
- Modèle peu réaliste

## Communication par messages, asynchrone

- Consensus impossible même avec défaillance simple
- Contournement : détecteurs de défaillances