Autour de l'Auto-Stabilisation

Partie 2 : Les versions fortes de l'auto-stabilisation

Alain Cournier & Stéphane Devismes

Université De Picardie

Cours de Master 2 Informatique

Plan de l'exposé

1. Auto-stabilisation (rappel)
2. Motivations
3. Tolérer des fautes de différents types
4. FTSS
5. Améliorer la convergence
6. La stabilisation instantanée
7. Conclusion

Auto-Stabilisation (rappel)

- **[Dijkstra 1974]**

- Une approche Générale pour réparer un système des effets de *toute* faute transitoire

Motivations

- L'auto-Stabilisation présente des *avantages*:

1. Tolérance à *toutes* les fautes transitoires :
   - Aucune hypothèse sur la nature de la faute transitoire
   - Retour à un état légitime indépendemment de la nature de la faute
2. Pas d'initialisation :
   - Passage à l'échelle
3. Dynamicité :
   - Auto-organisation dans les reseaux de capteurs et les réseaux ad hoc

Motivations

- Quelques inconvénients :

1. Résultats d'impossibilité
   - Problèmes fondamentaux sans solution auto-stabilisante
2. Surcoût
   - Des protocoles auto-stabilisants gourmands en ressources

Formes faibles

3. Intolérance aux autres formes de fautes
4. Sécurité à terme
   - Aucune garantie lors de la stabilisation

Formes fortes

Motivation

- Utile pour :
  - **Tolérer plus de type de fautes**
  - **Améliorer la propriété de convergence**
    - Converger rapidement dans certains cas (frequent)
    - Assurer des propriétés de sureté minimales lors des fautes

Tolérer d’autres fautes

- **Type de fautes :**
  - Transitoires
  - intermittentes
  - Pannes déf.
  - Byzantine

Tolérer d'autres fautes

- **Fautes transitoires** :
  - Prises en compte par **Self-Stabilisation**
  - Durée : *finie*
  - Fréquence : *rare*
  - Effet : *altération du contenu des composant(s) du réseau (processus et/ou liens)*
  - E.g., corruption de *mémoire/message, retour de panne, perte de messages…*

Tolérer d'autres fautes

- **Intermittent Faults**:
  - Duration: *finite*
  - Periodicity: ***frequent***
  - Effect: *alter the contain of some component(s) of the network (processes and/or links)*
  - E.g., *memory/message corruption, crash-recover, lose of messages…*
  - Some paper deals with both self-stabilization and certain types of intermittent fault, e.g., **[Delaët and Tixeuil, JPDC'02]**
    - Fair lose of message + finite number of message corruption

Tolérer d'autres fautes

- **Panne définitive**:
  - Durée : ***définitive***
  - Effet : un/*des composant(s) du reseau (processus et/ou liens) cessent définitivement de fonctionner*
  - E.g., panne de processus (hard), suppression de liens
  - *Autostabilisation* tolérante aux fautes (*FTSS*) **[Gopal and Perry, PODC'93]**
    - Seuls les pannes de processus sont étutiées (en general).

Tolérer d'autres fautes

- **Fautes Byzantine** :
  - Durée : ***pas de limite***
  - Effet : *des composant(s) du réseau (généralement les processus) fonctionnent de façon arbitraire*
  - E.g., processus subissant une attaque
  - *Autostabilisation tolérante aux byzantins* **[Dolev and Welch, PODC'95]**
    - Restriction sur le nombre de processus Byzantins et/ou
    - Des hypotheses de synchronisation

LIAFA

Election Robuste

Carole Delporte-Gallet (LIAFA)

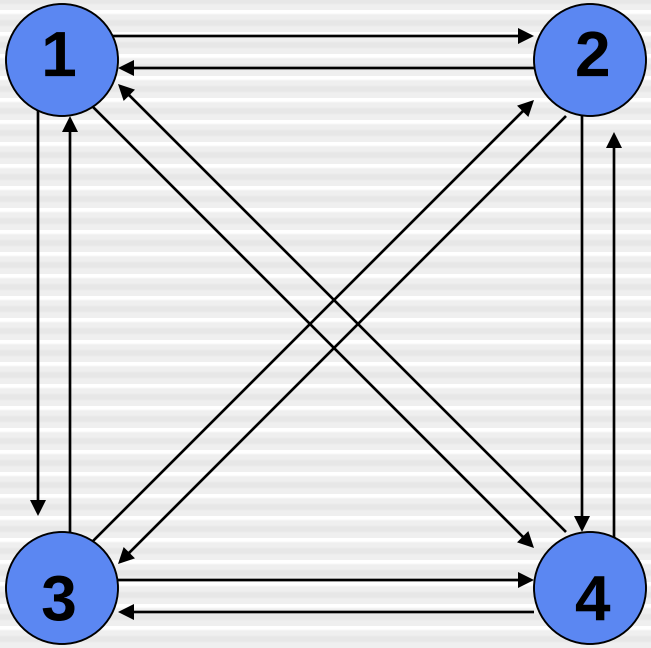
Stéphane Devismes (CNRS, MIS)

Hugues Fauconnier (LIAFA)

Sujet

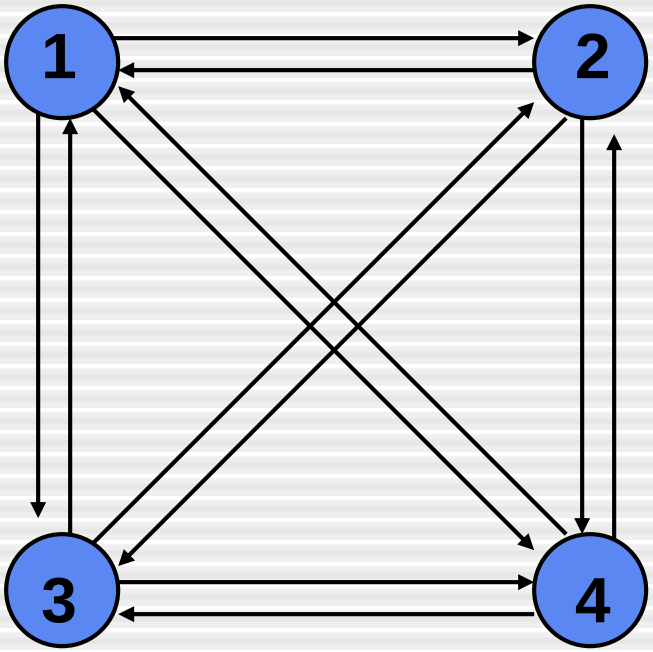
- Créer un algorithme d'*Election* dans le modèle à passage de messages qui soit
  1. Tolérant aux pannes définitives
  2. Autostabilisant
  3. Efficient pour le coût de Communications
  4. Avec d.es hypotheses de synchronisation faibles

Modèle



- Réseau en clique (complet)
- Communication par messages
- liens :
  - Unidirectionnels
  - Pas d'ordre sur la réception
  - Peut être synchrone
- Processus :
  - Synchrone ou en panne
  - Réseau identifié
  - Etat initial arbitraire

Efficacité en Communications



[Larrea, Fernandez, and Arevalo, 2000]:

« An algorithm is *communication-efficient* if it eventually only uses *n - 1* unidirectional links »

Un algorithme est efficace en communication s'il n'utilise à terme que n-1 liens unidirectionnels

Etat de l'art

- **[Gopal and Perry, PODC'93]**
- **[Anagnostou and Hadzilacos, WDAG'93]**
- **[Beauquier and Kekkonen-Moneta, JSS'97]**

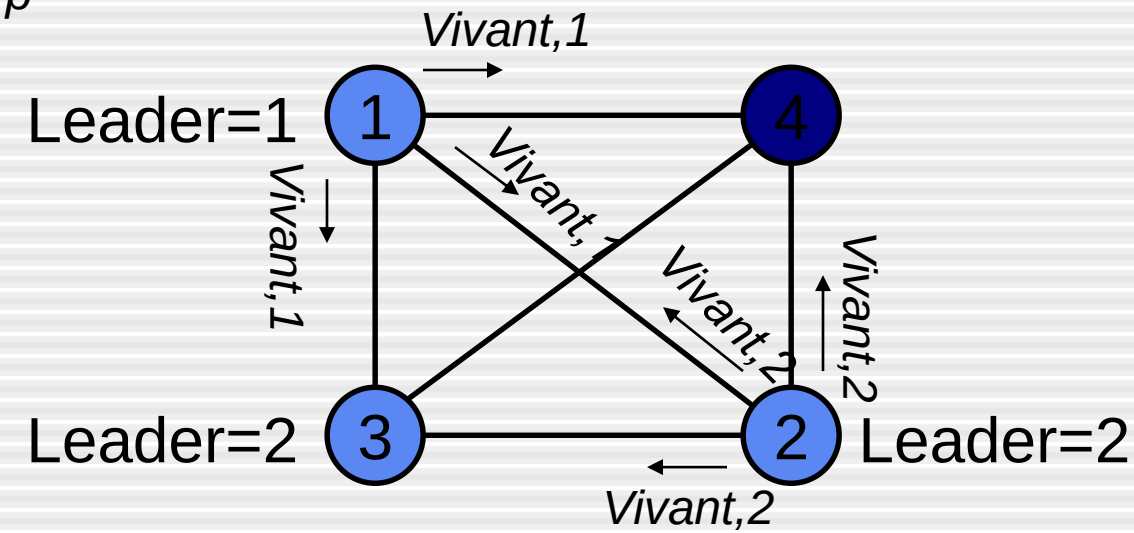
Aucune étude en efficacité de communications

Election autostabilisant dans un reseau complet ponctuel ?

Oui + efficacité en communication

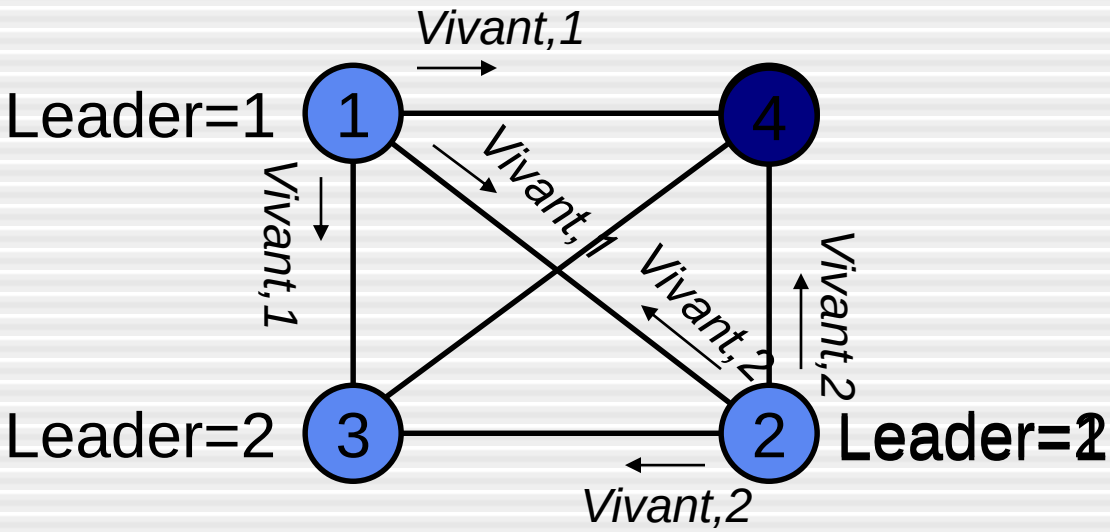
Algorithme (1/4)

- Chaque processus *p* envoie periodiquement *Vivant,p* à tous ses voisins si Leader = *p*



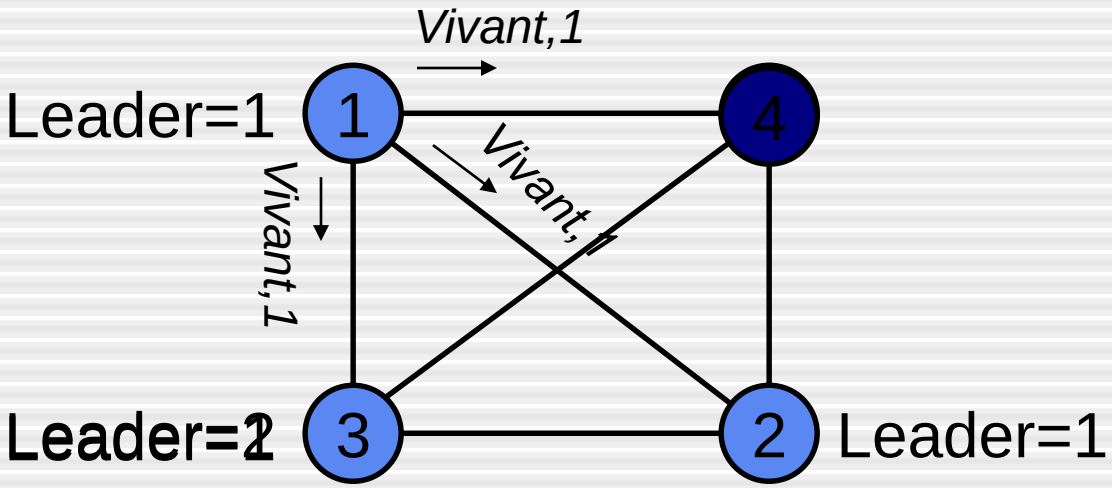
Algorithme (2/4)

- Quand un processus vivant *p* tel que *Leader* = *p* reçoit *Vivant* du processus *q*,
  - *Leader := q si q < p*



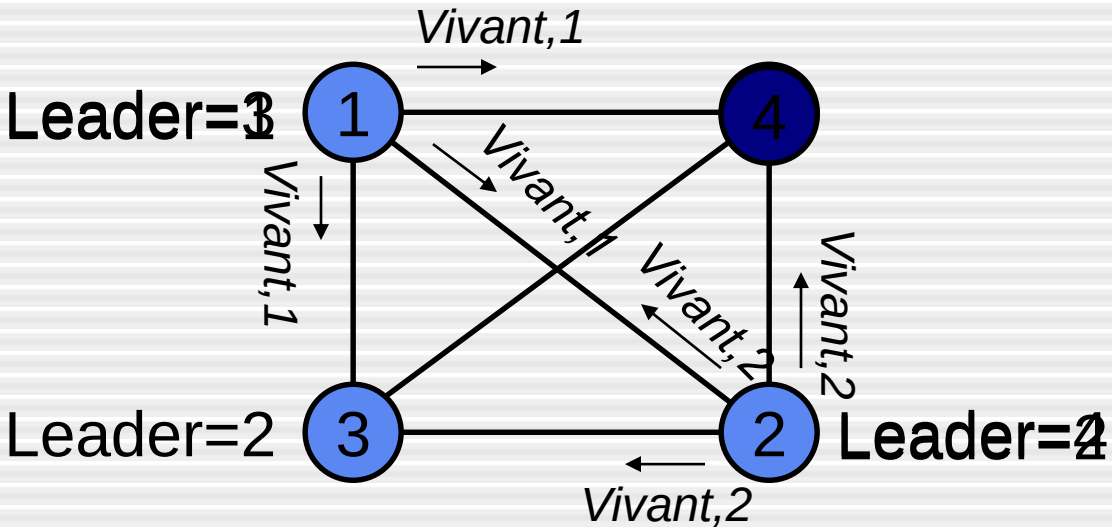
Algorithme (3/4)

- Chaque processus vivant *q* tel que *Leader* ≠ *q* Choisit tjs pour leader le processus dont il a reçu *Vivant* **le plus recemment**



Algorithm (4/4)

- ***A l'expiration du Time Out***, chaque processus vivant *p* met *Leader* à *p*



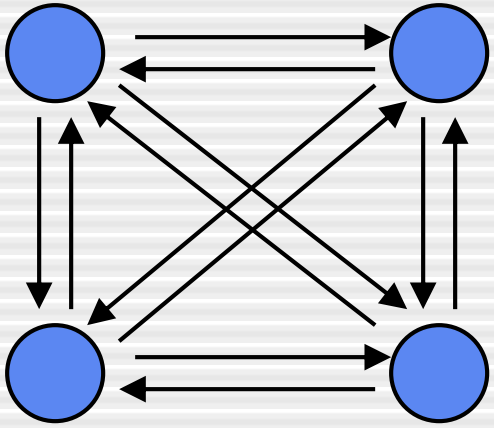
Election efficace en communication-Efficient et Autostab. dans un système où un lien est asynchrone?

Non

Impossibilité d'être efficace en Communication quand un lien est asynchrone

- **Fait :** tout processus p tel que Leader ≠ p doit périodiquement recevoir un message en temps borné sans quoi il choisit un autre leader

le processus choisit un autre leader



Election autostabilisante (pas efficiente en communication) *avec des liens asynchrone* ?

Oui

Election autostabilisante dans un reseau avec acheminement en temps borné

- Pour chaque processus (*p*,*q*), Il existe au moins deux chemins ponctuels (acheminement en temps borne) :
  - De *p* à *q*
  - De *q* à *p*

Algorithme

- Chaque processus calcule l'ensemble des noeuds vivants et choisit comme leader le plus petit de cet ensemble
- Pour calculer cet ensemble :

1. Chaque processus *p* envoie **periodiquement** *Vivant,p* à tous les autres
2. Tous les messages *Vivant,p* sont répétés *n - 1* fois

(les autres processus reçoivent **periodiquement** un message)

Election autostabilisante dans un reseau sans acheminement en temps borné ?

Non

Conclusion

- Avoir des algorithmes qui sont autostabilisant et tolérant aux pannes est souhaitable

- Mais les solutions efficiente en communication demandent des contraintes de synchronisation forte même dans un reseau complet

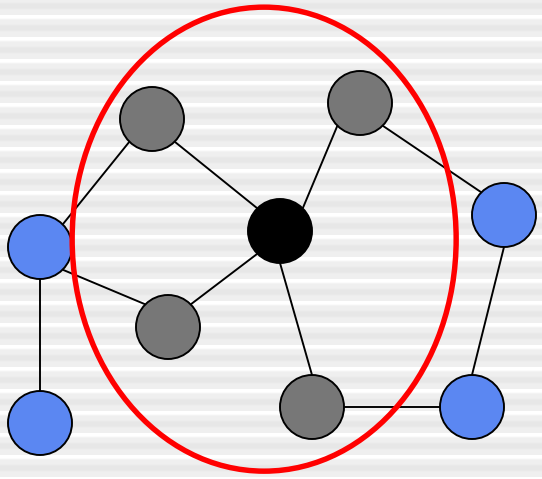
- **Solution:** *FTPS* (Fault-Tolerant Pseudo-Stabilization)

Améliorer la Convergence

- Autostabilisation + confinement de fautes
- Autostabilisation time-adaptive
- Autostabilisation avec garantie de propriété
- Superstabilisation
- Stabilisation instantanée

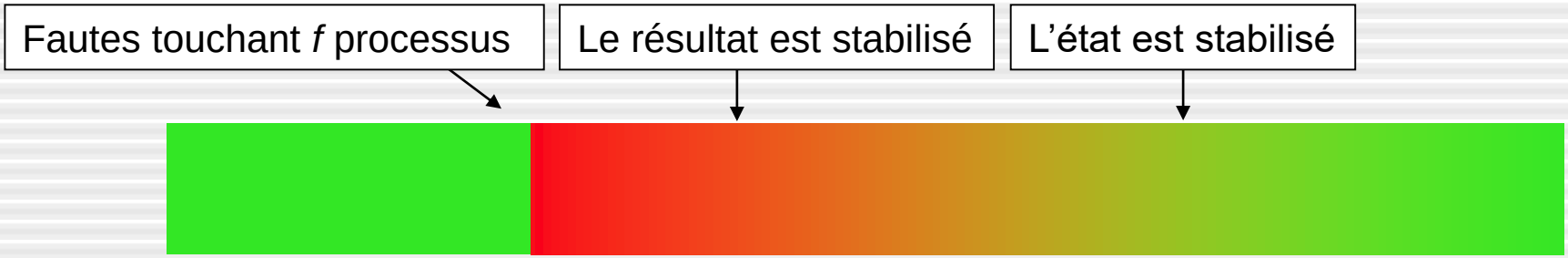
Confiner les fautes

- **[Ghosh et al, PODC'96]**
- Autostabilisation + s'il y a peu de fautes:
  - Confinement Spacial : un petit nombre de processus peuvent être influencés par les fautes
  - Convergence rapide (en temps)



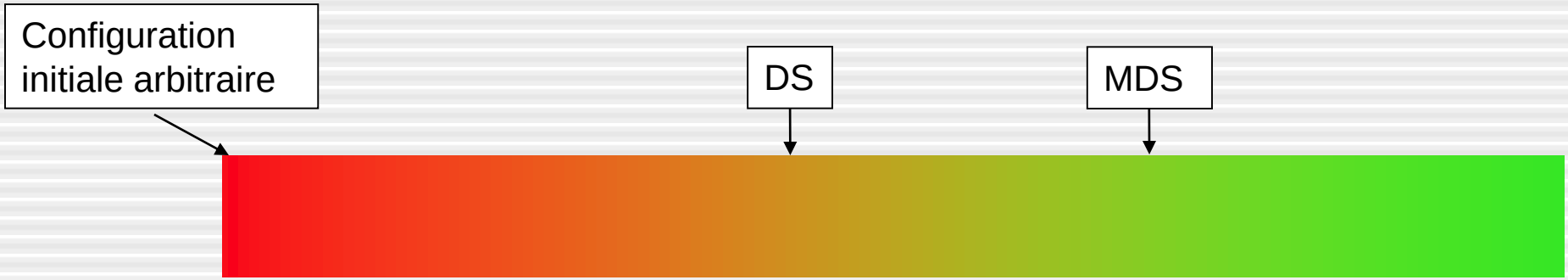
Time-Adaptive Self-Stabilization

- **[Kutten & Patt-Shamir, PODC'97]**
- Autostabilisation et si *f*<*k* processus fautifs:
  - Le résultat de l'algorithm se stabilise en *O(f)*



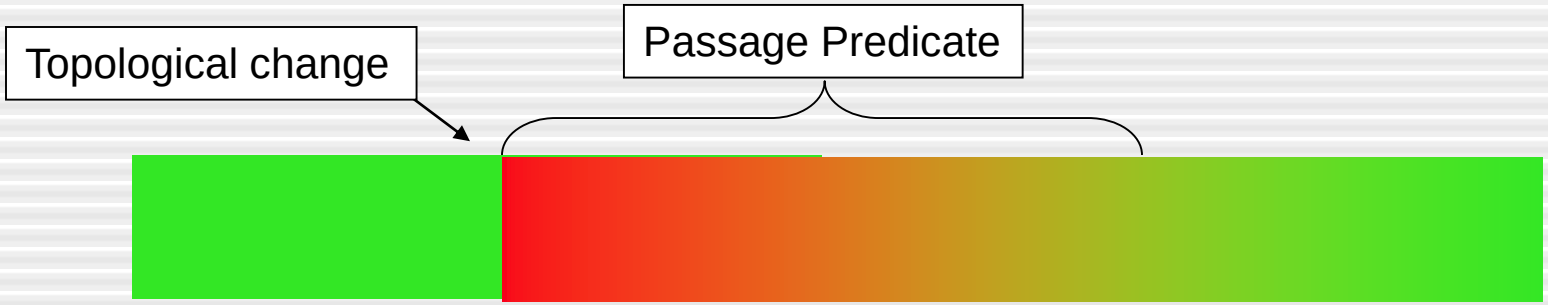
Safe-Converging Self-Stabilization

- **[Kakugawa & Masuzawa, IPDPS'06]**
- Autostabilisation + convergence rapide vers une propriété plus faible (mais utile)
- E.g. *Ensenble dominant Minimal (MDS)*:



Superstabilisation

- **[Dolev & Herman, CJTCS'97]**
- Un algorithme Superstabilisant
  - Dois être autostabilisant
  - Dois préserver un "*prédicat de transition*"
  - *Prédicat de transition* - Défini par rapport à la classe de changement de topologie (Le changement altère la legitimité donc le prédicat de transition doit être plus faible mais assez fort pour être utile).



Exemple de Predicat de transition

Jeton dans un anneau:

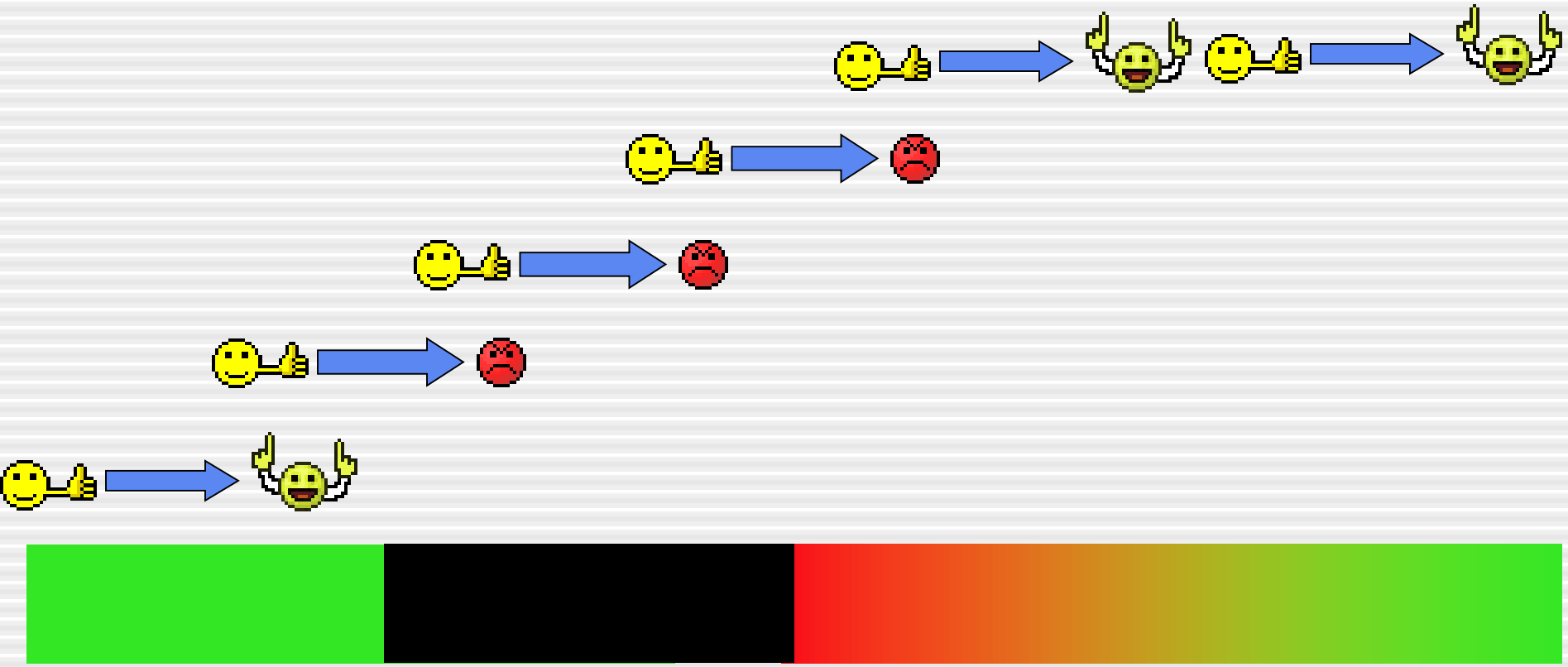
Prédicat de transition	Legitimate State
Au plus un jeton dans le système. (e.g. l'existence de 2 tokens est illégale)	**Exactement** un jeton dans le système.

La panne d'un processus peut faire disparaitre le jeton mais le prédicat de transition reste vrai

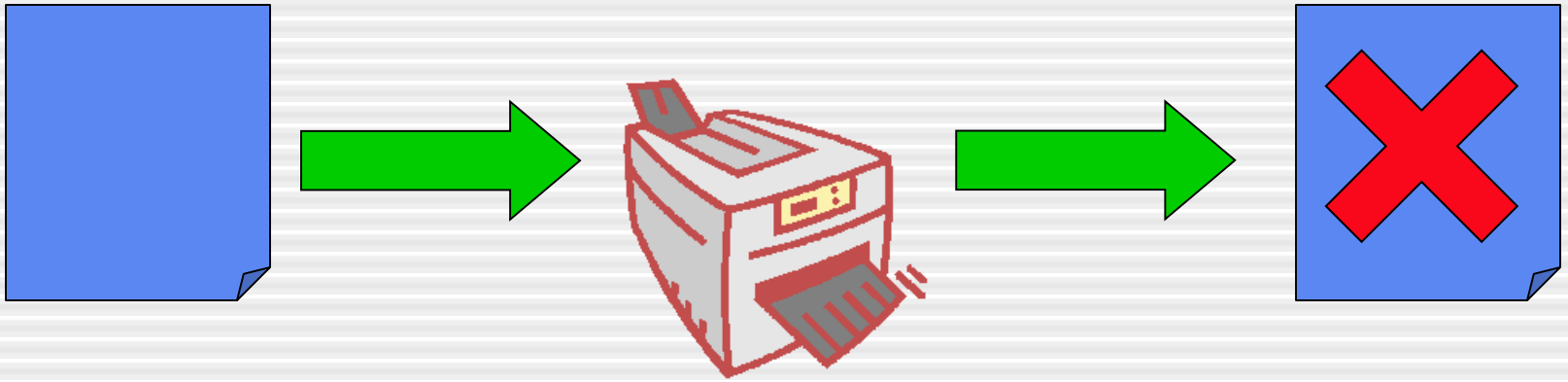
Stabilisation instantanée

- **[Bui *et al*, WSS'*99*]**
- Un algorithme instantanément stabilisant opère correctement dès la fin des fautes
- Algorithm à base de requête et vue centrée sur l'utilisateur :
  - Chaque requête initiée par l'utilisateur obtieint un résultat correct.

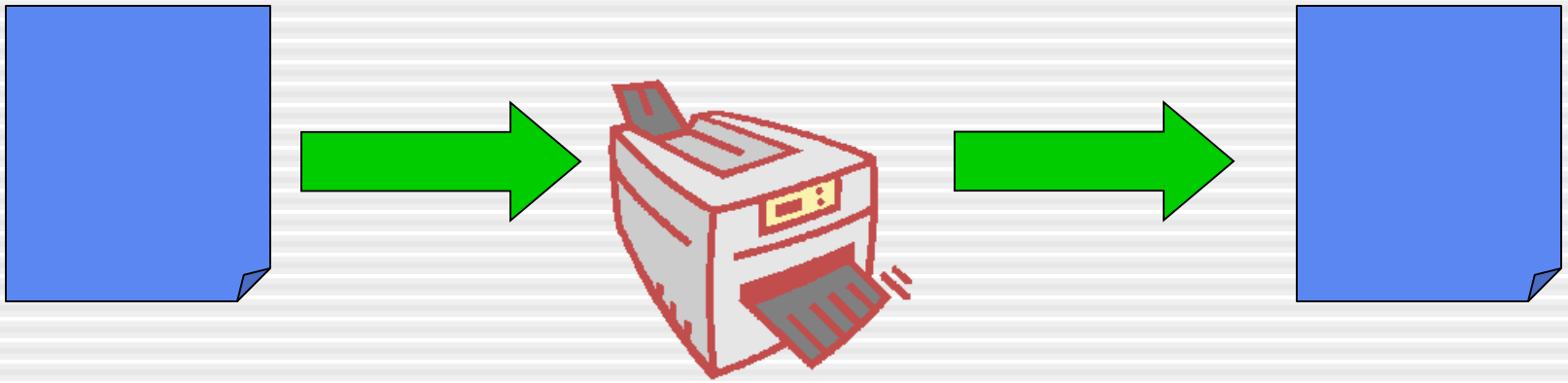
Snap-Stabilization



Self *vs.* Snap



Self *vs.* Snap



1.X

A suivre...