Lélia Blin

Algorithmes distribués tolérant aux pannes

Algorithmique répartie
M1 Université d'Evry

Réseaux à grandes échelles

- Augmentation du nombre de composants dans les réseaux
  - Augmentation de la possibilité qu'un ou plusieurs de ces composants tombe en panne.
- Réduction du coût de fabrication des composants pour obtenir des économies d'échelle
  - Accroît également le taux de défauts potentiels.
- Les risques de pannes deviennent impossibles à négliger

Taxonomie des pannes

- Un premier critère pour classifier les pannes dans les systèmes répartis est la localisation dans le temps.
- On distingue généralement trois types de pannes possibles :
  - les pannes transitoires
  - les pannes définitives
  - les pannes intermittentes

Les pannes définitives

- Des pannes de nature arbitraire peuvent venir frapper le système,
- Il existe un point de l'exécution à partir duquel ces pannes anéantissent pour toujours ceux qui en sont frappés ;

Les pannes transitoires

- Pannes de nature arbitraire qui frappe le système
- Comportement aberrant temporaire du système
  - modification de variables
  - duplication de message
- Il existe un point de l'exécution à partir duquel ces comportements aberrant n'apparaissent plus

Les pannes intermittentes

- Pannes de nature arbitraire qui perturbe le système de manière régulières.

Les pannes d'état

- Le changement des variables d'un élément peut être dû à des perturbations dues à:
  - L'environnement
    - Ondes électromagnétiques
  - Des attaques
    - Dépassement de tampon
  - Défaillances du matériel utilisé.
- Les variables peuvent prendre des valeurs erronées
  - Des valeurs qu'elles ne sont pas sensées prendre lors d'une exécution normale du système.

Pannes de code

- Les pannes crash :
  - A un point donné de l'exécution, un élément cesse définitivement son exécution et n'effectue plus aucune action ;
- Les omissions :
  - A divers instants de l'exécution, un élément peut omettre de communiquer avec les autres éléments du système, soit en émission, soit en réception ;

Pannes de code

- Les duplications :
  - A divers instants de l'exécution, un élément peut effectuer une action plusieurs fois, quand bien même son code stipule que cette action doit être exécutée une fois ;
- Les désequencements :
  - A divers instants de l'exécution, un élément peut effectuer des actions correctes, mais dans le désordre ;
- Les pannes byzantines :
  - Elles correspondent simplement à un type arbitraire de pannes, et sont donc les pannes les plus malicieuses possibles.

Classes d'algorithmes tolérants aux pannes

- Il existe deux classes principales d'algorithmes tolérants aux pannes:
  - Algorithmes robustes
  - Algorithmes auto-stabilisants

Algorithmes robustes

- Utilisation de la redondance à plusieurs niveaux:
  - des informations,
  - des communications,
  - des nœuds du système,
  - des algorithmes.

Algorithmes robustes

- Hypothèse:
  - Seul un nombre limité de fautes peut frapper le système
    - Conservation d'une majorité d'éléments corrects
- Algorithmes masquants:
  - les algorithmes robustes cachent à l'utilisateur l'occurrence des fautes

Algorithmes auto-stabilisants

- Hypothèse:
  - Les pannes sont transitoires
    - Autrement dit limitées dans le temps
- Aucune supposition n'est faite quant à l'étendue des fautes qui peuvent concerner tous les éléments du système.

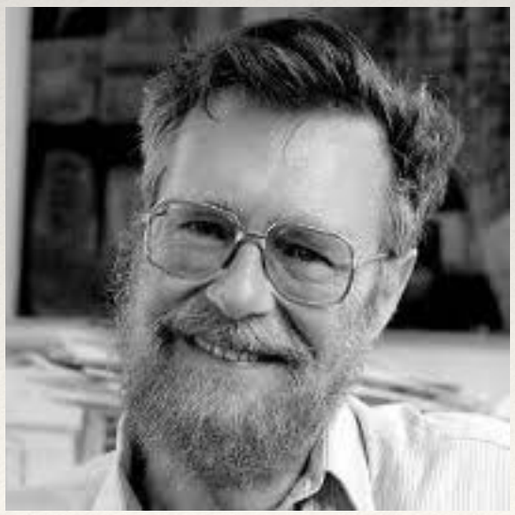
Algorithmes auto-stabilisants

- Un algorithme est auto-stabilisant si:
  - il parvient en temps fini, à exhiber un comportement correct indépendamment de l'état initial de ses éléments
- Non masquants:
  - car entre le moment le comportement aberrant cesse et le moment où le système est stabilisé sur un comportement correct, l'exécution peut être erratique

Définition

- Un système est dit auto-stabilisant, indépendamment de son état initial, si il garanti d'arriver en un état légitime en un nombre fini d'étapes.

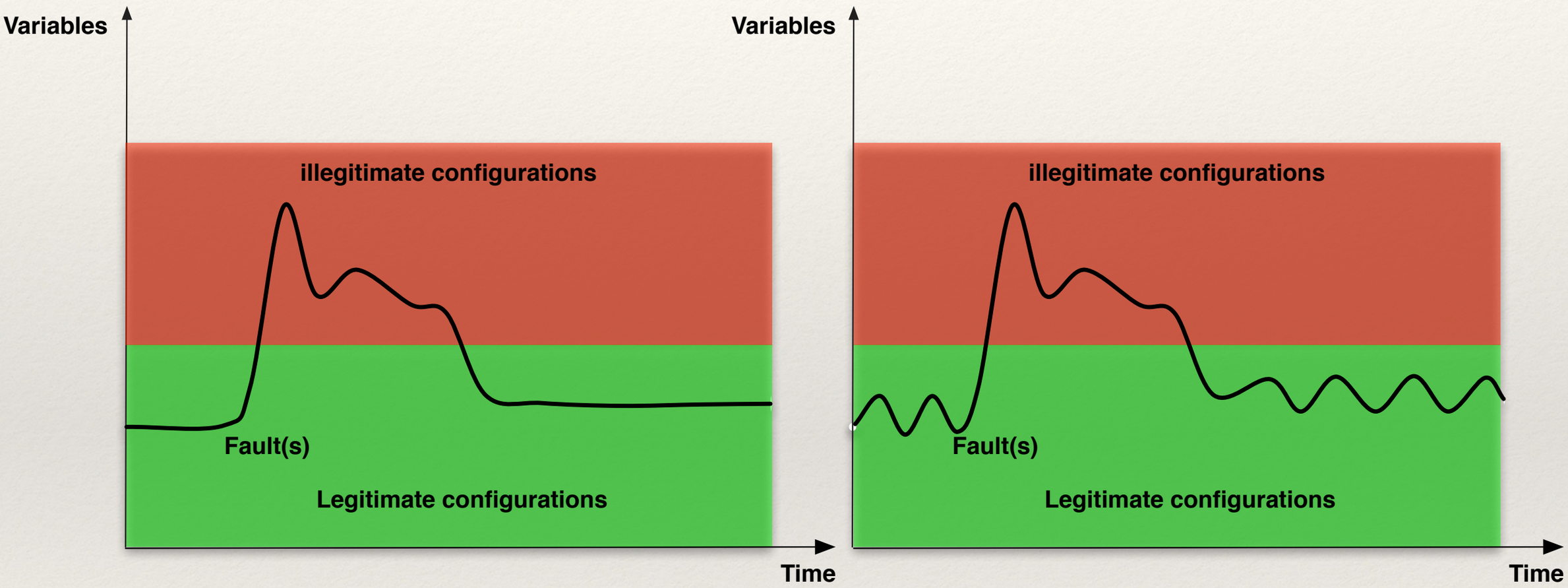
- Edsger W. Dijkstra, Self-stabilizing systems in spite of distributed control, Communications of the ACM, v.17 n.11, p.643-644, Nov. 1974



Convergence et clôture

- Un système est dit auto-stabilisant si il assure la :
  - Convergence:
    - Partant d'une configuration arbitraire il est garanti de rejoindre une configuration légitime en un nombre fini d'étapes.
  - Clôture:
    - Une fois dans une configuration légume le système reste dans une configuration légitime.

Convergence et clôture



Un algorithme auto-stabilisant d’arbre couvrant

Problème

- Soit un graphe connecté connexe non orienté G=(V,E) et un noeud distingué r (racine).
- Construire un BFS auto-stabilisant.
- Autrement dit:
  - Absence de cycle
  - Connexe
  - Couvrant tout les noeuds du systèmes

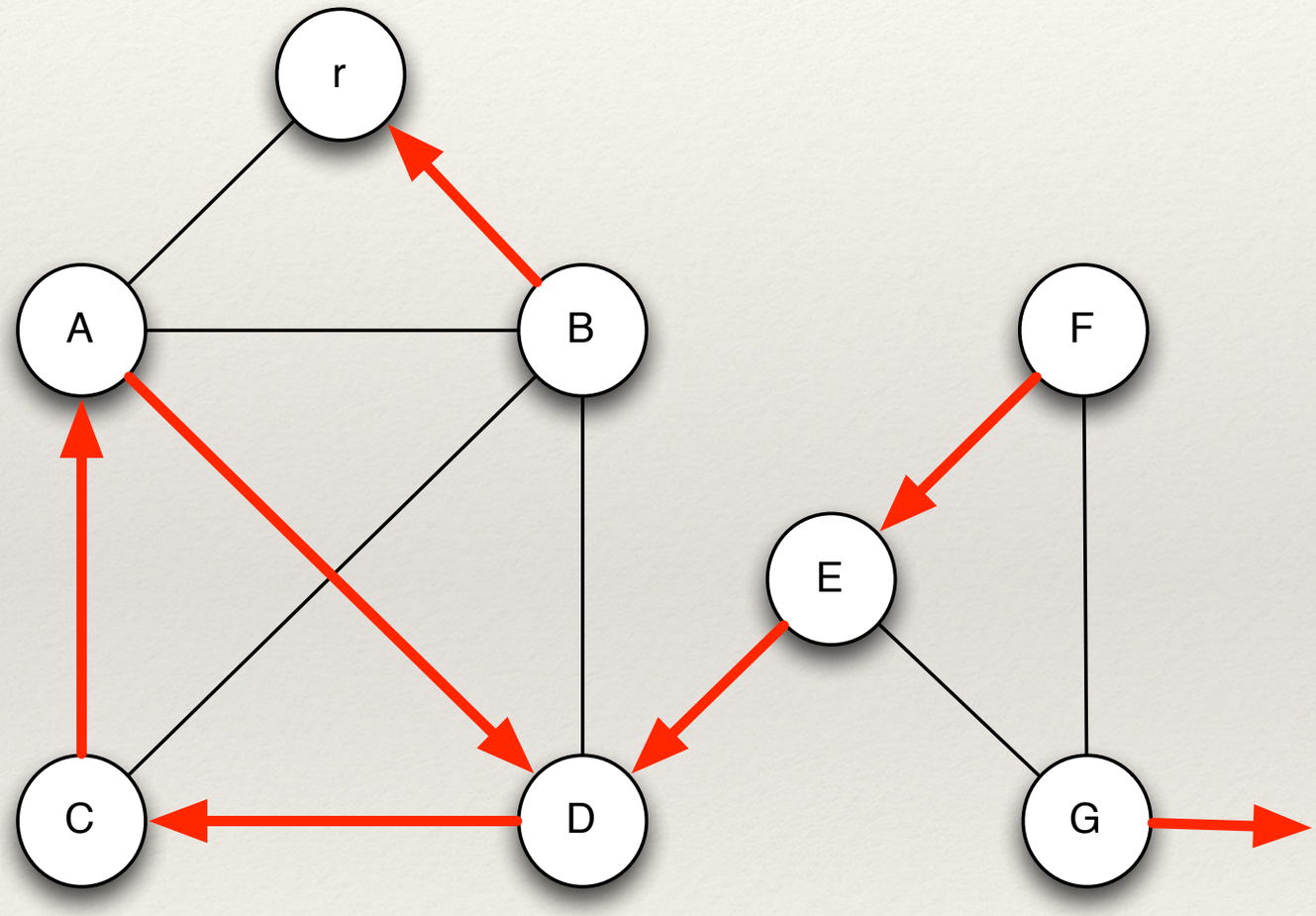
Variables locales

- d_i: distance du noeud i à la racine
- p_i: identifiant du parent du noeud i
- Remarques:
  - $0 \leq d_i \leq n$ (où $n = |V|$)

Etat légitime

- Pour la racine r:
  - d_i=0
  - p_i=NIL
- Pour les noeuds non racine:
  - d_i=d_{pi}+1
  - p_i=identifiant

Exemple de configuration arbitraire



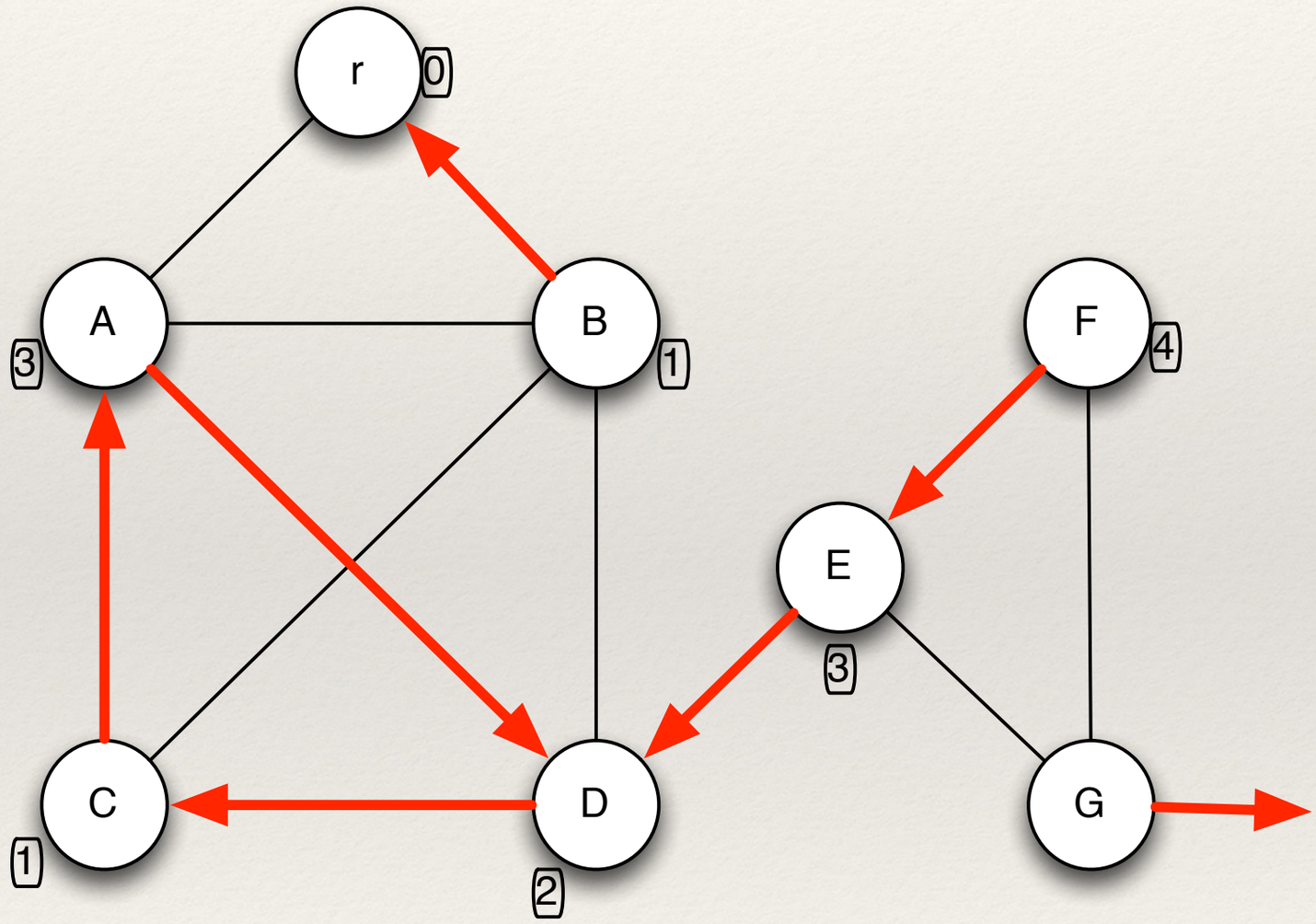
BFS Auto-stabilisant

Variables de v : R_v racine de l'arbre
d_v distance du noeud à la racine
p_v parent de v dans l'arbre

Algorithme

- Si $p_v \in N(v) \longrightarrow p_v := \emptyset$
- Si $p_v = \emptyset$ faire envoyer <Demande> à $N(v)$
  sinon envoyer <Demande> à p_v
- A la reception de <Demande> envoyer par u
  envoyer $<R_v, d_v>$ à u
- A la reception de $<R, D>$ envoyer par u
  si $R < R_v$ faire $R_v := R$; $d_v = D+1$; $p_v := u$
  si $R = R_v$
  si $p_v = u \wedge D > d_v + 1$ faire $d_v := \infty$; $p_v := \infty$
  sinon si $D < d_v + 1$ faire $d_v := D + 1$; $p_v := u$

Exemple de configuration arbitraire



Exemple d'exécution