Consensus tolérant aux fautes

Rohan Fossé - rohan.fosse@labri.fr

Défaillances des processeurs dans le passage des messages

- **Crash :** à un moment donné, le processeur cesse de prendre des mesures.
  - À l'étape finale du processeur, il peut réussir à envoyer seulement un sous-ensemble des messages qu'il est censé envoyer.

Problème de consensus

- Chaque processeur a une entrée.
- ***Terminaison*** : Finalement, chaque processeur non défaillant doit décider d'une valeur.
  - la décision est irrévocable !
- ***Accord*** : Toutes les décisions des transformateurs non fautifs doivent être identiques.
- ***Validité*** : la décision d'un processeur non défaillant doit être une valeur d'entrée.

Exemples de consensus

- Entrées binaires :
  - vecteur d'entrée 1,1,1,1,1,1
    - la décision doit être 1
  - vecteur d'entrée 0,0,0,0,0,0
    - la décision doit être 0
  - vecteur d'entrée 1,0,0,1,0
    - la décision peut être soit 0 soit 1
- Entrées à valeurs multiples :
  - vecteur d'entrée 1,2,3,2,1
    - la décision peut être 1 ou 2 ou 3

Modélisation des défaillances de l'accident

- Modifier les définitions sans défaillance de l'exécution **admissible** pour tenir compte des défaillances par crash :
- Tous les processeurs, à l'exception d'un ensemble d'au plus ***f*** processeurs (ceux qui sont **défectueux**), ont pris un nombre infini de mesures.
  - Dans le cas *synchrone* : dès qu'un processeur défectueux ne parvient pas à effectuer une étape dans un tour, il n'effectue plus aucune étape.
- Lors de la dernière étape d'un processeur défectueux, un sous-ensemble arbitraire des messages sortants du processeur se retrouve dans les canaux.

Algorithme de consensus synchrone tolérant aux pannes (pseudo-code suivant)

- L'algorithme est destiné aux systèmes synchrones avec au plus *f* processeurs défectueux.
- Chaque processeur conserve dans une variable V l'ensemble des valeurs qu'il sait exister dans le système. Initialement, sa valeur d'entrée.
- Chaque processeur met à jour V en le joignant aux ensembles reçus par les autres processeurs à chaque tour (ligne 3,4) et diffuse tout nouvel ajout à l'ensemble aux autres processeurs (ligne 2). L'ensemble diffusé par p_i est noté S_i .
- Au dernier tour, il décide de la plus petite valeur dans l'ensemble V

Algorithme de consensus synchrone tolérant aux pannes (pseudo-code suivant)

Algorithm 15 Consensus algorithm in the presence of crash failures: code for processor p_i, $0 \leq i \leq n-1$.

Initially $V = \{x\}$ // V contains p_i's input

1: round k, $1 \leq k \leq f+1$:
2: send $\{v \in V : p_i$ has not already sent $v\}$ to all processors
3: receive S_j from p_j, $0 \leq j \leq n-1$, $j \neq i$
4: $V := V \cup \bigcup_{j=0}^{n-1} S_j$
5: if $k = f+1$ then $y := \min(V)$ // decide

Exécution de l'algorithme

- round 1 :
  - envoyer mon entrée
  - recevoir les messages du 1er tour
  - calculer la valeur de *v*
- round 2 :
  - envoyer *v* (si c'est une nouvelle valeur)
  - recevoir les messages de la deuxième série
  - calculer la valeur de *v* calculer les événements
- ...
- round $f + 1$:
  - envoyer *v* (si c'est une nouvelle valeur)
  - recevoir le cycle $f + 1$ msgs
  - calculer la valeur de *v*
  - decide *v*

Validité de l'algorithme de consensus tolérant aux pannes

Nous supposons que le graphe de communication est une clique

Terminaison : Par le code, il se termine au tour $f+1$.

Validité : elle se maintient puisque les processeurs n'introduisent pas de messages parasites : seules les valeurs d'entrée circulent.

Validité de l'algorithme de consensus tolérant aux pannes

Consensus :

- Dans chaque exécution, à la fin du tour f+1, l'ensemble des valeurs V_i et V_j collectées par deux processeurs non défaillants p_i et p_j est le même
  - Nous prouvons que si x est dans V_i à la fin du tour f+1 alors x est dans V_j

Preuve

- Soit r le premier tour dans lequel x est ajouté à V_i pour tout processeur non défectueux p_i
  - r=0 si v est p_i valeur d'entrée
- Si $r \leq f$, alors au tour $r+1 \leq f+1$, p_i envoie x à chaque p_j et p_j ajoute x à V_j s'il n'est pas déjà présent
- Sinon, r=f+1

Performances de l'algorithme de consensus tolérant aux pannes

- Nombre de processeurs $n > f$
- $f + 1$ tours (c'est optimal)
- au plus $n^2 \times |V|$ messages, chacun de taille $\log |V|$ bits, où V est l'ensemble d'entrée.

Consensus asynchrone tolérant aux pannes

Consensus asynchrone tolérant aux pannes

- Il n'existe pas d'algorithme de consensus **asynchrone**, même en présence d'un seul processeur en panne.
  - Impossible en Message Passing (MP)
  - Impossible en lecture/écriture Mémoire partagée (SM)

Modélisation de systèmes asynchrones avec défaillances de type "crash".

- Soit f le nombre maximum de processeurs défectueux.
- Pour SM et MP : Tous les processeurs sauf f doivent prendre un nombre infini de pas dans une exécution admissible.
- Pour MP : exiger également que tous les messages envoyés à un processeur non défaillant soient finalement délivrés, à l'exception de ceux envoyés par un processeur défaillant lors de sa dernière étape, qui peuvent ou non être délivrés.

Impossibilité du consensus asynchrone tolérant aux pannes

1. Impossible dans une mémoire partagée en lecture/écriture avec *n* processeurs et *n* – *1* fautes.
   - implique qu'il n'y a pas d'algorithme 2-proc pour 1 défaut
2. Impossible dans une mémoire partagée en lecture/écriture avec *n* processeurs et 1 faute.
   - Prouvé par réduction au cas 1
3. Impossible dans le passage de messages asynchrones
   - Prouvé par la réduction à l'impossibilité dans la mémoire partagée en lecture/écriture.

Algorithmes sans attente

- Un algorithme pour *n* processeurs est **sans attente** s'il peut tolérer $n - 1$ échecs.
- L'intuition veut qu'un processeur non défaillant n'attende pas que d'autres processeurs fassent quelque chose : il ne le peut pas, car il pourrait être le seul processeur encore en vie.
- Nous allons montrer qu'il n'existe pas d'algorithme de consensus sans attente dans le modèle de mémoire partagée r/w asynchrone.

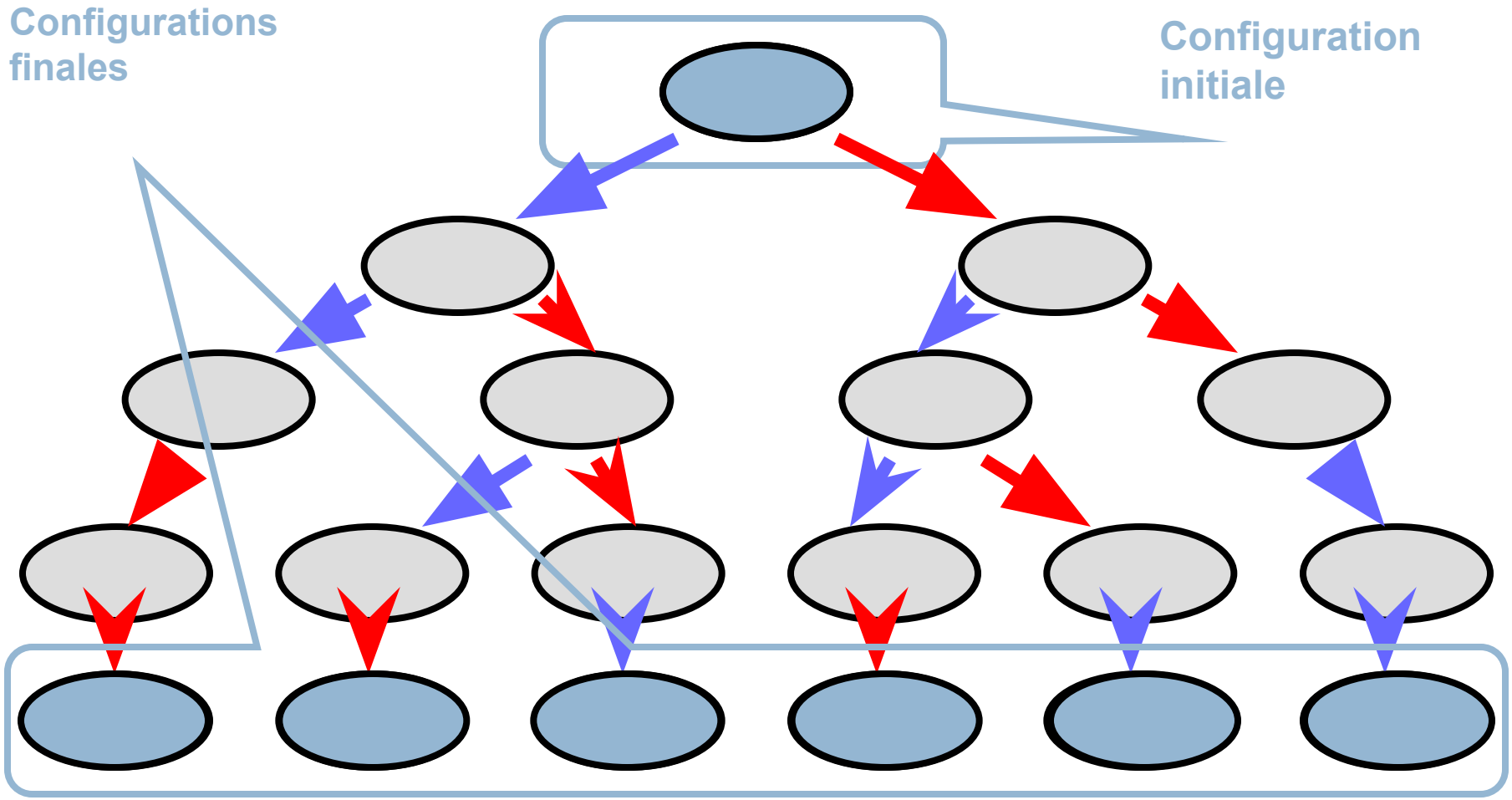
Impossibilité d'un consensus sans attente

- Supposons, par contradiction, qu'il existe un algorithme à *n processeurs* pour *n – 1* fautes dans le modèle de mémoire partagée à lecture/écriture asynchrone
- Nous considérons un consensus binaire : l'entrée d'un processeur est soit 0, soit 1.

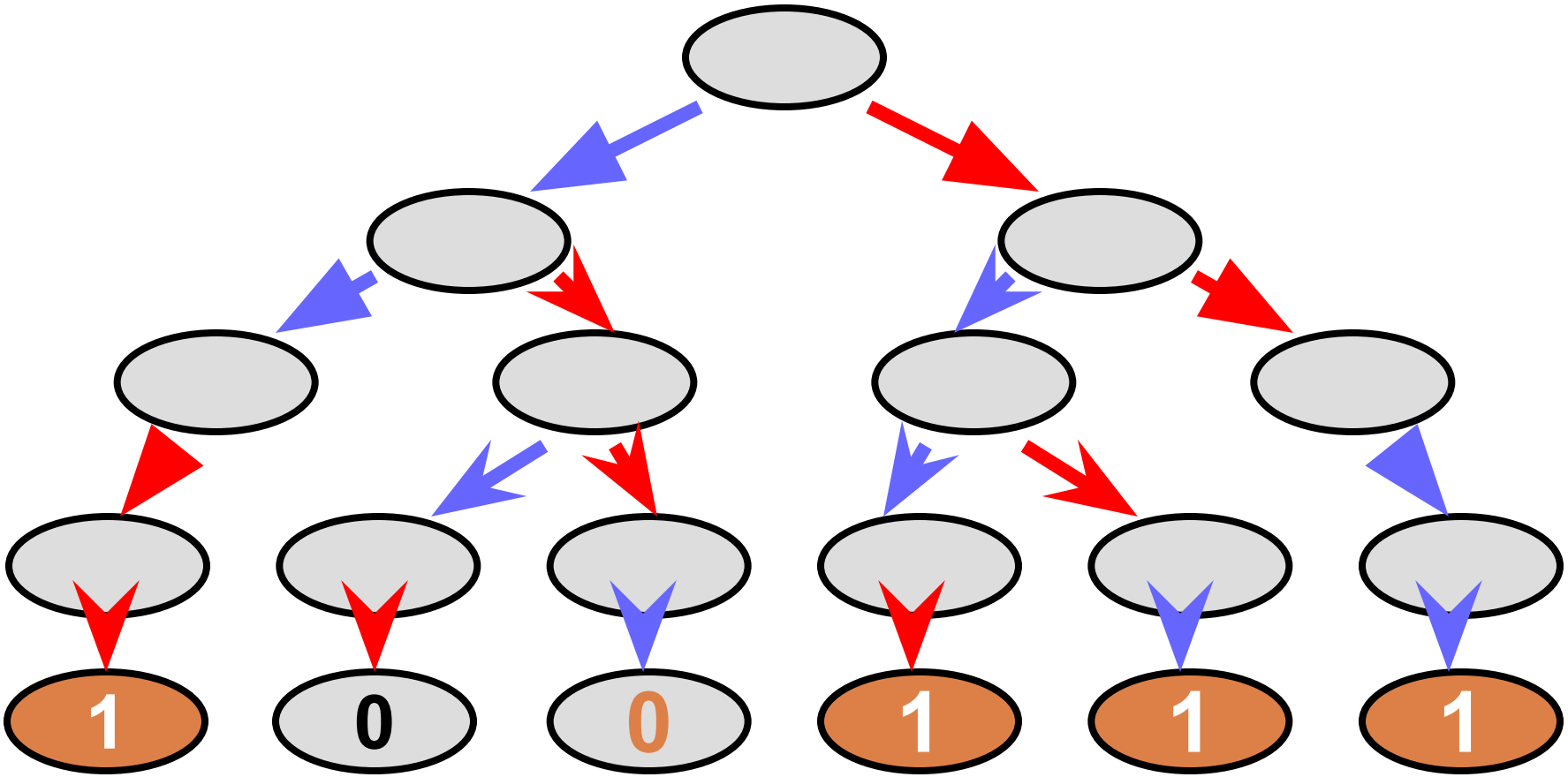
Valence d'une configuration

- La **valence** d'une configuration C est l'ensemble de toutes les valeurs décidées par un processeur non défaillant dans une configuration atteignable depuis C par une exécution admissible.
- **Bivalent :** l'ensemble contient 0 et 1.
- **Univalent :** l'ensemble ne contient qu'une seule valeur
  - *0-valent* ou *1-valent*

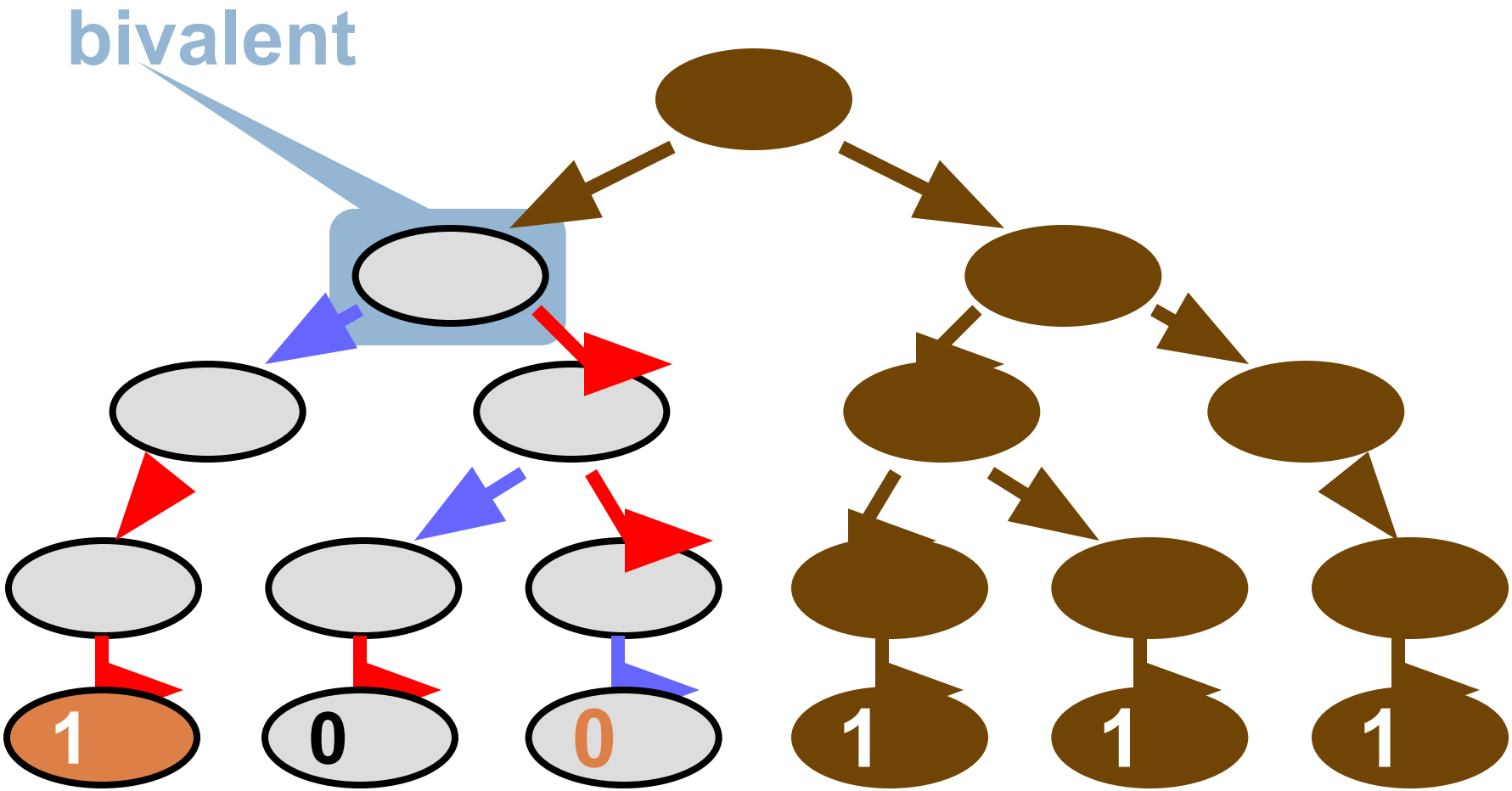
Pour un système avec deux processeurs



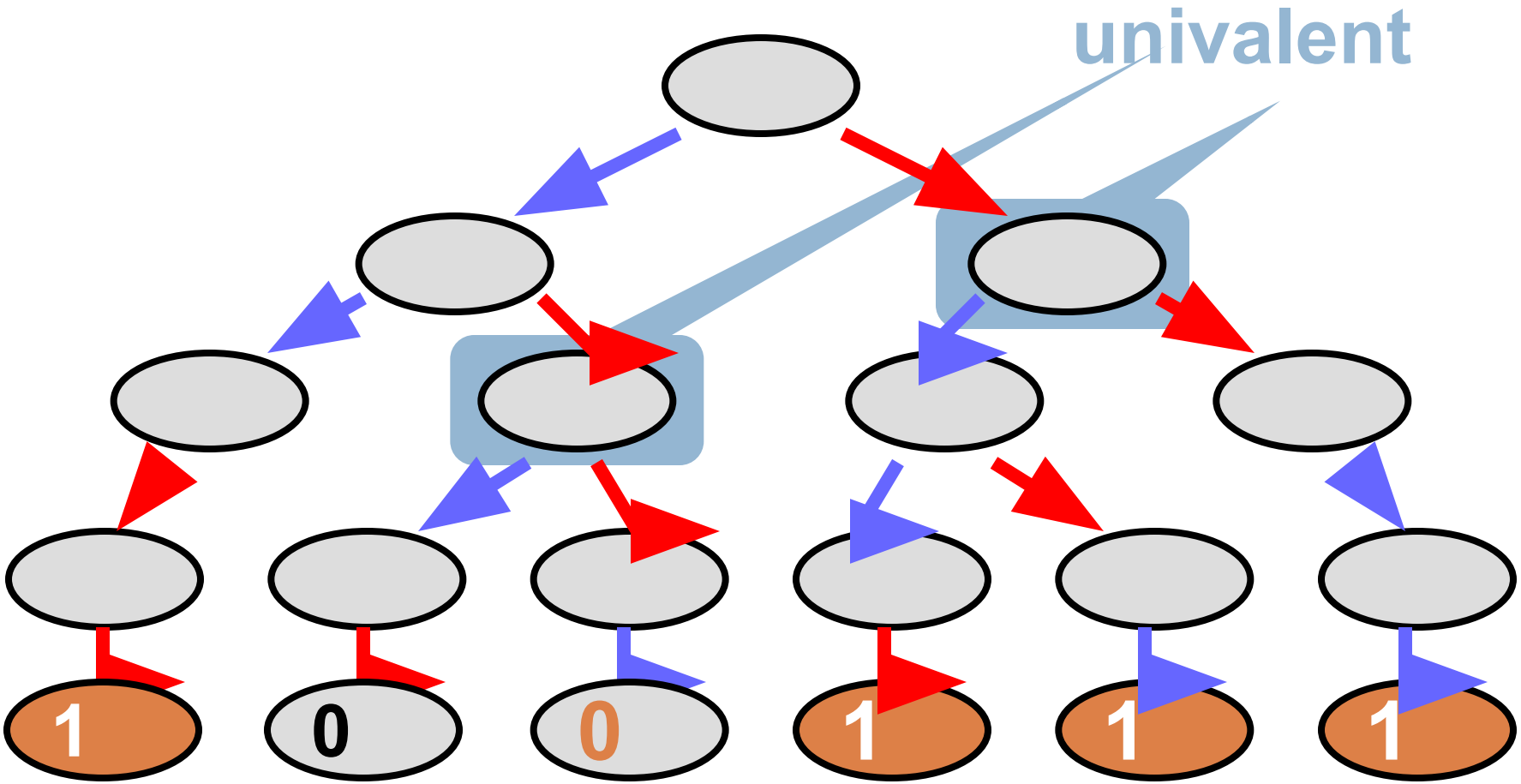
Consensus binaire : Valeurs de décision



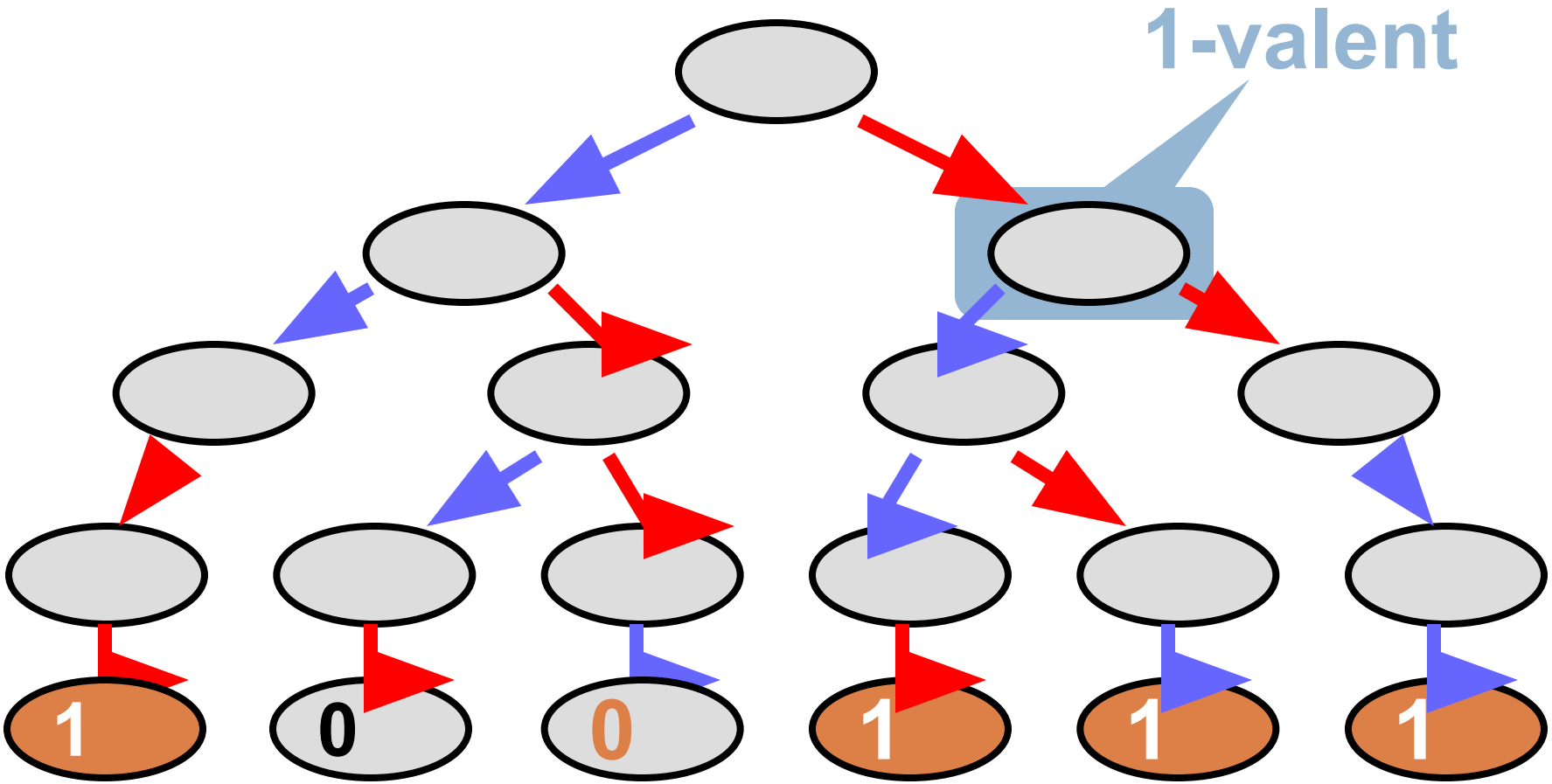
Bivalent : les deux sont possibles



Univalent : Une seule valeur possible



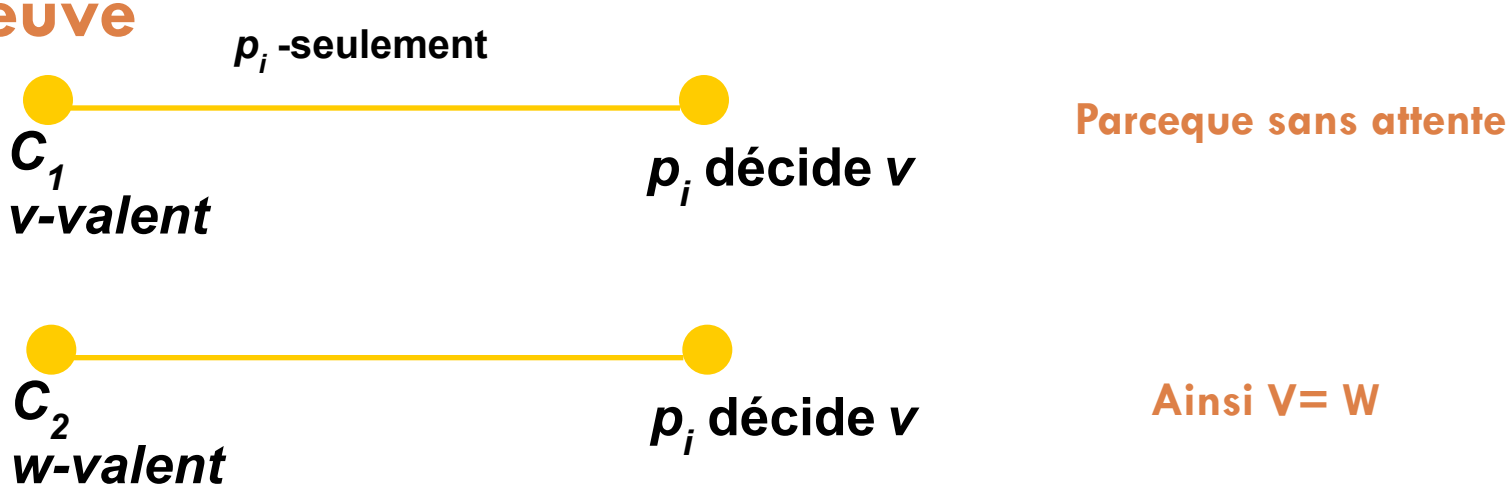
x-valent : x Seule décision possible



Similitude univalente

Lemme Si C_1 et C_2 sont tous deux univalents et qu'ils sont *similaires en ce qui concerne* p_i (l'état de la mémoire partagée est le même, l'état local de p_i est le même), alors ils ont la même valence.

Preuve



Configuration initiale bivalente

Lemme (5.16) : Il existe une configuration initiale bivalente.

Existence d'une configuration initiale bivalente.

- Supposons, par contradiction, que toutes les configurations initiales soient univalentes.

entrées	valence
000...00	0
000...01	?
000...11	?
...	
001...11	? 0
011...11	? 1
111...11	1

par condition de validité

Il existe 2 configurations adjacentes avec des valences différentes

Existence d'une configuration initiale bivalente.

- Soit
  - I_0 une configuration initiale 0-valent
  - I_1 une configuration initiale 1-valent
  - tels qu'elles ne diffèrent que par l'entrée de p_i 's

I_0 p_i échoue initialement, pas d'autres échecs. Vu qu'il termine, c'est le reste qui décide.

tous sauf p_i décider 0

I_1 Cette exécution est la même que celle ci-dessus à tous les processeurs sauf p_i .

tous sauf p_i décider 0

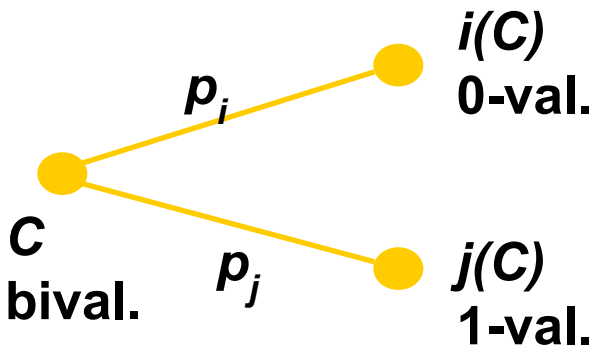
Contradiction

Processeurs critiques

Définition Si C est bivalent et la configuration *i*(C) (résultat de p_i en prenant une étape) est univalente, alors p_i est **critique** dans C.

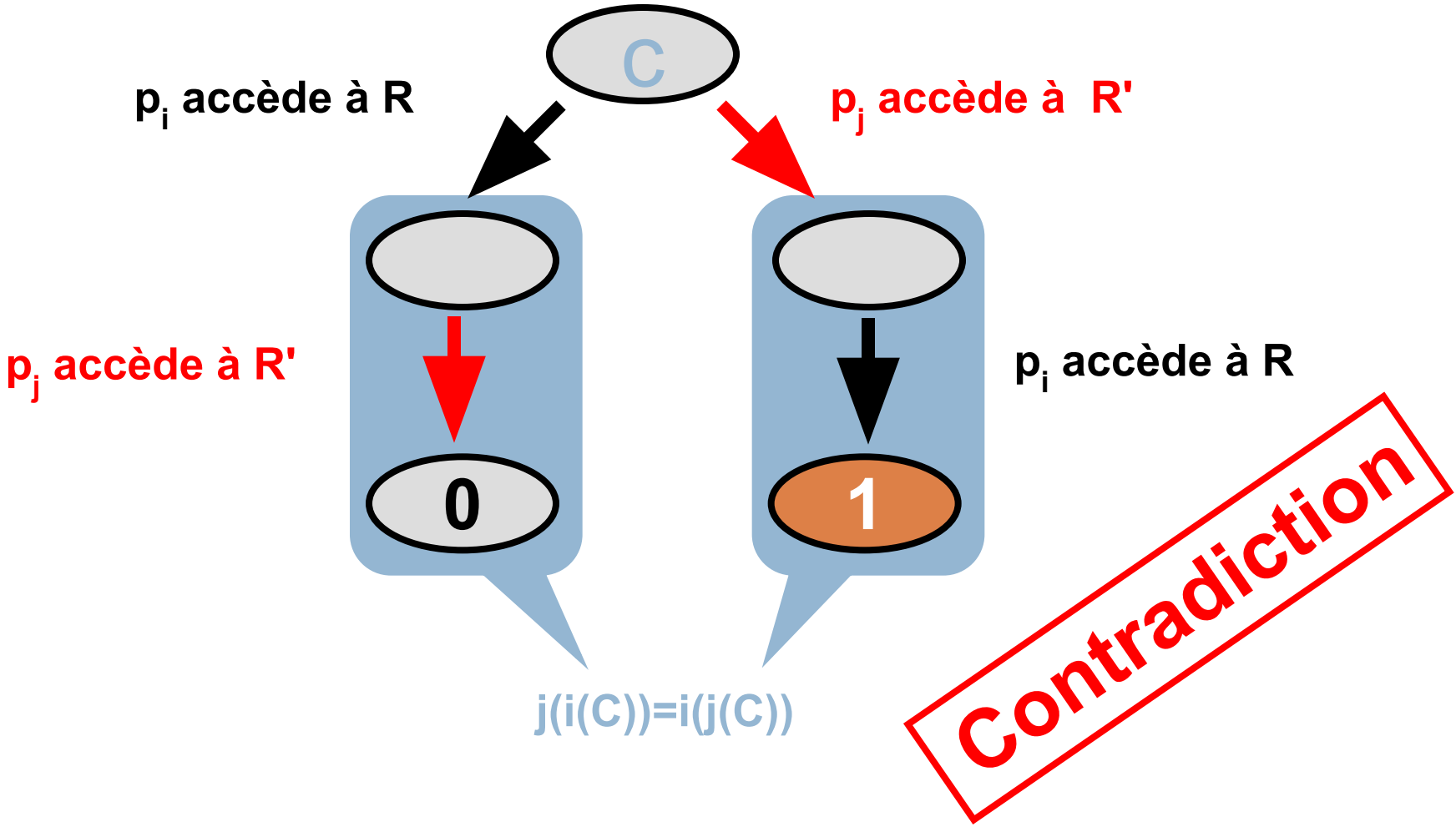
Lemme Si C est bivalent, alors au moins un processeur n'est pas critique dans C, c'est-à-dire qu'il existe une extension bivalente.

Preuve Supposons par contradiction que tous les processeurs sont critiques.

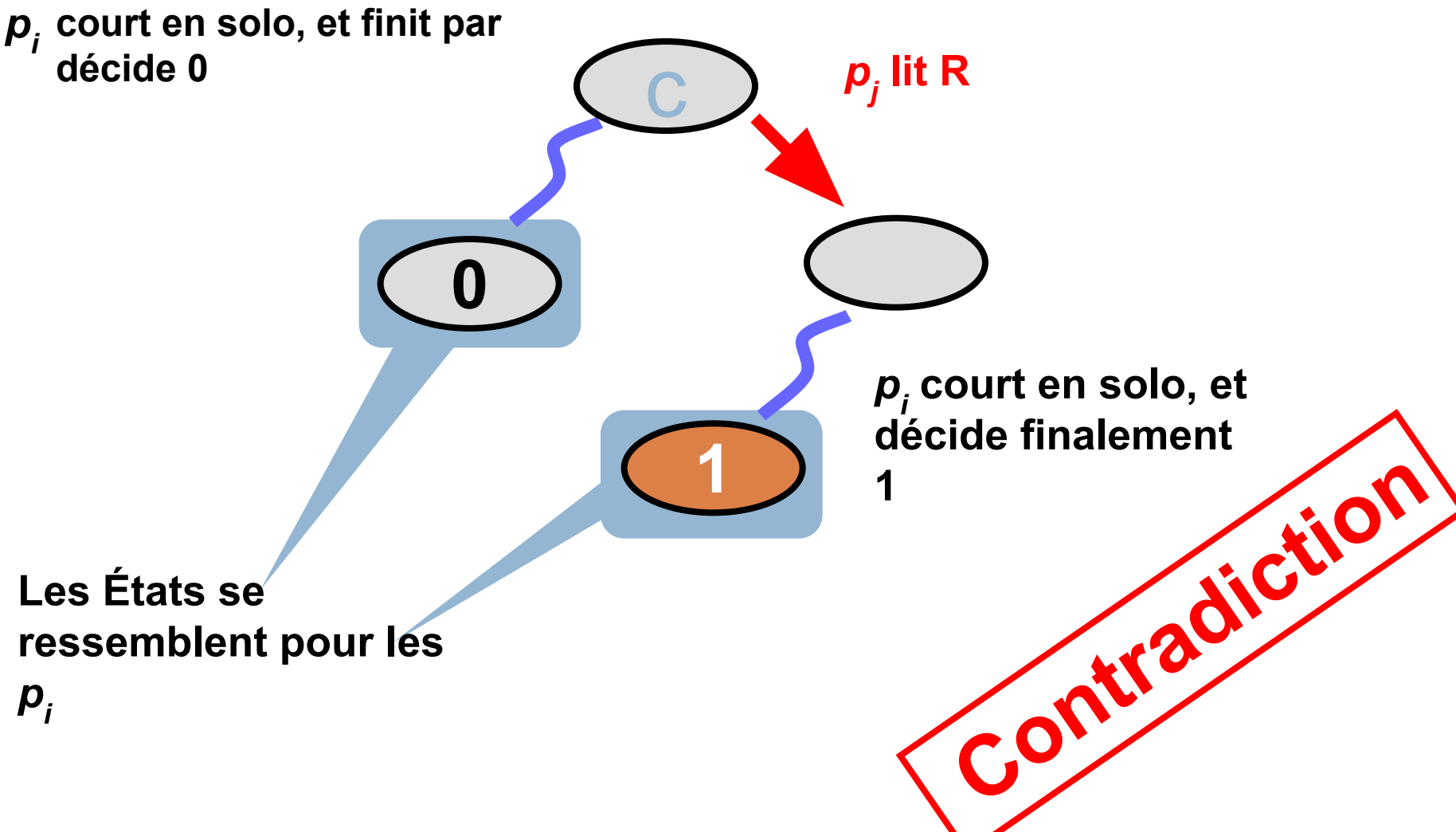


Le reste de la preuve est une analyse de cas de ce que p_i et p_j font dans leurs deux pas

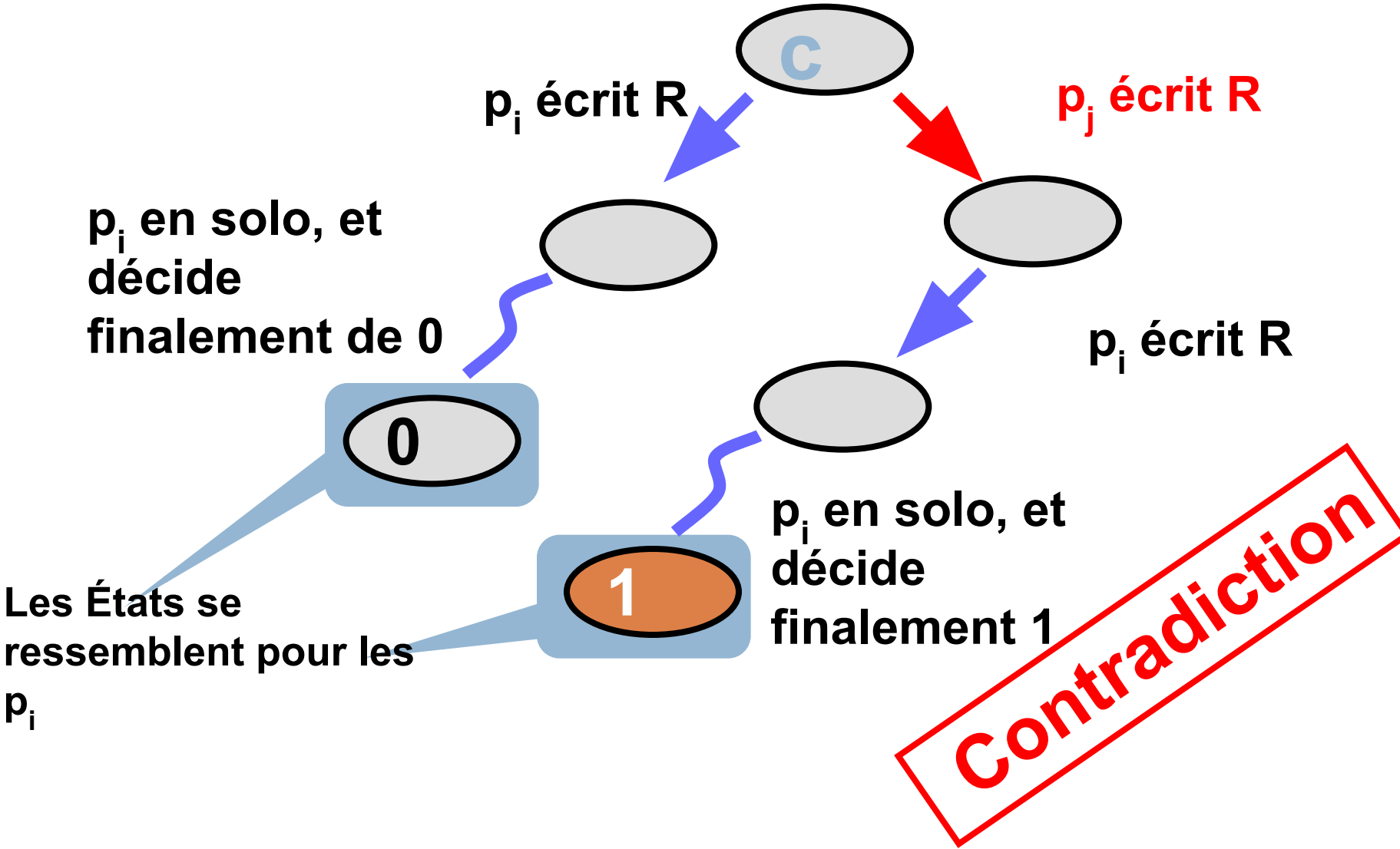
Cas 1 : p_i et p_j accèdent à des registres différents



Cas 2 : p_i accède (lecture ou écriture) à un registre R et p_j lit dans R



Cas 3 : p_i et p_j écrivent le même registre



Pour compléter la preuve

- Créer inductivement une exécution admissible dans laquelle les configurations restent bivalentes pour toujours :
  - *à chaque configuration, le processeur qui n'est pas critique pour la configuration prend une étape*

Consensus avec Défaillances

Théorème (Fisher, Lynch, Paterson, '85)

> Le consensus est impossible pour des processus asynchrone s'il peut se produire au moins une défaillance de type crash

Cependant, que se passe-t-il si les fautes se produisent avant et pas pendant l'exécution de l'algorithme ?

Consensus avec fautes passées

Si toutes les fautes se produisent avant le commencement de l'algorithme :

- Il existe un algorithme *f-tolérant* pour $f < n/2$

Algorithme (au moins L= (n+1)/2 processus corrects)

- On envoie un message à tous (graphe complet)
- Attendre pour au moins L messages de voisins;
- Si reçu un message de w, on ajoute une arête dirigé vers w
- On calcule la plus grande composante fortement connexe H
- On effectue l'élection sur H et décide la valeur

Détecteur de fautes

Supposons qu'il existe un mécanisme permettant de détecter des fautes

Définition (Failure Detector (FD))

Un module qui donne une liste de processus qui sont suspectés d'être en panne.

- Trouver un algorithme qui utilise le FD pour éviter des fautes;
- Attendre pour un message de p_i si et seulement si p_i n'est pas dans la liste des suspects.

Types de FD

1. Complet: Si p est en panne => p est suspecté par chaque q.
2. Précise: Si p est correct, p $\notin$ Suspect(q) $\forall$ q
3. Faiblement précise: $\exists$p tel que p$\notin$Suspect(q) $\forall$ q
4. Parfaite : Complet + Précise

Consensus Asynchrone avec FD

Hypothèse: Réseau K_n avec faiblement précis FD

Algorithme

- Processus P_i avec valeur v_i : $X = v_i$
- Pour ronde r =1 à n:
  - Si r == i Envoyer vi à tous;
  - Si Pr ∉ Suspect, attendre pour message vr de Pr;
  - Si reçu vr, X = vr
- Décider la valeur de X.

Correction: Il y a un moins un processus qui n'est pas suspecté. Tous les processus va recevoir sa valeur.

Problèmes avec FD

- La liste de Suspecté n'est pas la même pour tous les processus;
- Si un processus p est suspecté, p n'est pas forcément en panne. Le résultat d'un algorithme dépend du nombre de processus suspectés.
- Difficile de construire un détecteur de fautes.
  - Il faut regarder les états des machines, les statistiques, les probabilités de défaillances, les temps de réponses, etc…

Types de Pannes

- Fautes Permanentes (Crash)
  - Défaillances de processus
  - Défaillances de liens
- Fautes Temporaires
  - Omission de messages
  - addition de messages
  - corruptions de messages

Fautes Dynamiques

- Défaillances de liens (**temporaires**)
- Pas localisés: peuvent apparaître n'importe où dans le réseau
- Il y a une borne sur le nombre de fautes par ronde.

Réseau synchrone: Message a envoyé à temps t, message b reçu à temps t +1 :

1. Omissions: $a \neq \varnothing$, $b = \varnothing$
2. Additions: $a = \varnothing$, $b \neq \varnothing$
3. Corruptions: $a,b \neq \varnothing$, $a \neq b$

Problème de k-consensus

Problème de k-Consensus:

Au moins k processus doivent être d'accord

1. Unaminité: $k = n$ (Consensus)
2. Majorité: $k = (n+1)/2$
   a. S'il y a Deg(G) fautes d'omissions par ronde, k-Consensus est impossible pour $k > n/2$
   b. S'il y a Deg(G) fautes d'additions ou corruptions par ronde, k-Consensus est impossible pour $k > n/2$
   c. S'il y a Deg(G)/2 fautes d'omissions, additions ou corruptions par ronde, k-Consensus est impossible pour $k > n/2$

Algorithme pour Unanimité

1. Fautes d'omissions:
   a. Possible de tolérer $f = c - 1$ fautes, si G est c-arete-connexe.
   b. Algorithme: Pour c(n-1) rondes, envoyer à tous les voisins v_i
2. Additions: Possible de tolérer f fautes
   a. Pour chaque f,Envoyer un message dans chaque ronde.
3. Corruptions: Possible de tolérer f fautes
   a. Pour chaque f, si vi = 0 envoyer 0, sinon ne rien faire.