Algorithmique répartie

Problèmes génériques

Gérard Padiou

Département Informatique et Mathématiques appliquées
ENSEEIHT

Octobre 2012

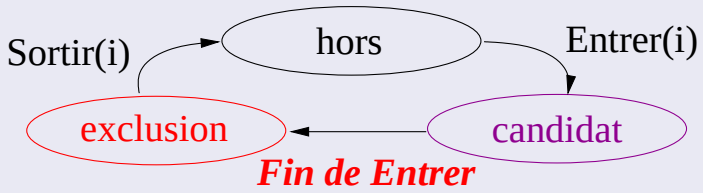
plan

Plan

Spécification du problème

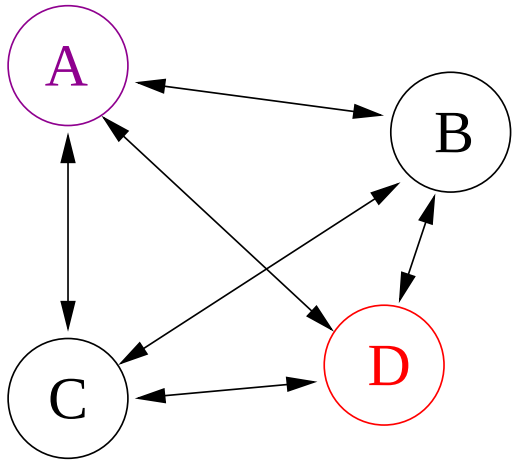
process P(int i) :
. . . ; Entrer(i) ; <SC> ; Sortir(i) ; . . .

Diagramme d'état



- Sûreté : Un processus au plus en exclusion
  $\forall i, j :: P_i.exclusion \wedge P_j.exclusion \Rightarrow i = j$;
- Sûreté : pas d'interblocage ;
- Vivacité : Tout processus finit par entrer ;
- Protocole : Tout processus finit par sortir ;

Réseau maillé fiable



D est en exclusion.
A est candidat

Principes de base

Règle de base

Tout processus candidat doit demander à d'autres processus la permission d'entrer en exclusion

- À tout P_i on associe donc un ensemble D_i contenant les processus à contacter ;
- Condition nécessaire : $\forall i \neq j : j \in D_i \vee i \in D_j$
- Deux types de permissions :
  - individuelles : un processus donne son autorisation selon son propre état ;
  - d'arbitre : les processus s'échangent des permissions préexistantes en nombre fixé.
- Objectif : Minimiser les ensembles D_i.

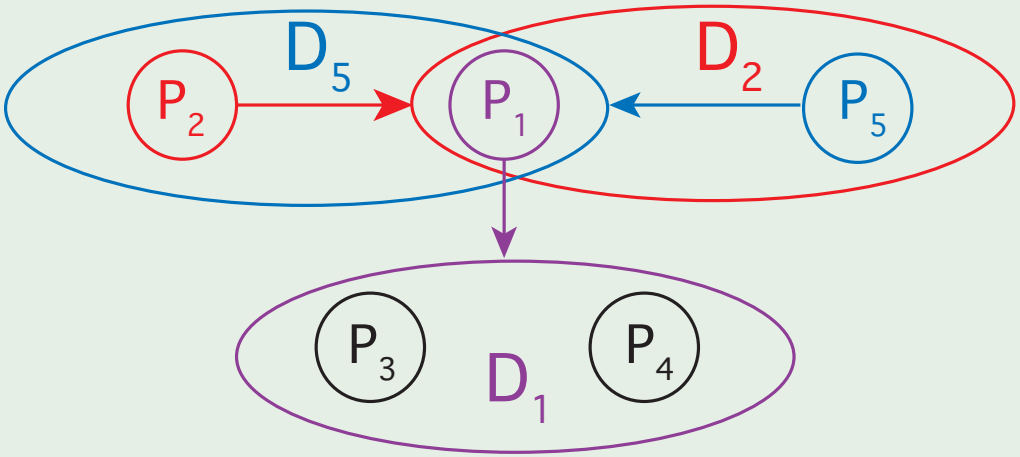
Permissions individuelles

Exemple

C'est bon pour P_1 car :

- P_1 contacte P_3 et P_4 : $P_3, P_4 \in D_1$,
- P_1 est contacté par P_2 et P_5 : $P_1 \in D_2 \wedge P_1 \in D_5$

Compléter : D_3, D_4?



Permissions individuelles : au pire ...

Ricart et Agrawala

Hypothèses

- Chaque processus sait qu'il existe N processus ;
- Réseau de communication fiable et maillé ;
- Pas de défaillance de processus.

Solution

- Utilisation de permissions individuelles avec :

$$\forall i : D_i = \{Tous\} - \{i\}$$

- Ordonnancement des requêtes par datation.

Algorithme de Ricart et Agrawala

Principes

- Requêtes d'entrée totalement ordonnées :
  ☞ Utilisation d'horloges de Lamport
- Chaque processus P_i candidat ou en exclusion connaît la date de sa requête courante $Date(R_i)$;
- Un candidat entre en exclusion s'il a obtenu les permissions de tous les autres ;
  ☞ il possède alors la requête la plus ancienne
- Revient à vérifier que la requête R_i d'un processus P_i est la plus vieille requête des processus candidats ou en exclusion :

$$\forall k : P_k.Etat \neq hors \Rightarrow Date(R_i) \leq Date(R_k)$$

Un algorithme de base

Algorithme de Ricart-Agrawala

```
// invariant : P_i.EC==exclusion ⇒ ∀p, P_p.EC==candidat : P_i.drLoc < P_p.drLoc
process P(i: 0..N-1) {
  type Etat = {hors,candidat,exclusion}; Etat EC = hors;
  Date hloc=new Date(i,1); Date drLoc;
  Set<int> Att=new EnumSet<int>();Set<int> D=EnumSet.range(0,N-1).remove(i);
  while (true) {
   select {
     when (EC==hors)⇒ // hors → candidat
       EC=candidat; drLoc=hloc.Top();
       for (int k: D) send Rq(i,drLoc) to P_k ;
   [] when (EC==exclusion)⇒ // exclusion → hors
       for (int k: Att) send Perm(i) to P_k;
       Att.clear(); D=EnumSet.range(0,N-1).remove(i); EC=hors;
   [] when (EC==candidat) ⇒ receive Perm(p) ; // candidat ? → exclusion
       D.remove(p); if(D.empty()) EC=exclusion;
   [] receive Rq(p,dr) ; hloc.Recaler(dr);
       if(EC!=hors && Date.pred(drloc,dr))Att.add(p);else send Perm(i) to P_p;
   } // select
  } // while
}
```

Permissions d'arbitres

Il faut obtenir la permission de tous les arbitres contactés : $\forall j : P_j \in D_i$

Definition

Condition nécessaire : $\exists$ arbitre commun : $\forall i \neq j : D_i \cap D_j \neq \emptyset$

Exemple

i prend pour arbitre :	1	2	3	4	5
1		•	•		
2			•	•	
3		•		•	
4		•		•	
5		•	•		

Note : 1 et 5 ne sont pas arbitres.

Permissions d'arbitres

Exemple

Optimisation :

- partir d'une matrice (ici 9 sites) :

$$\begin{array}{ccc} 1 & 2 & 3 \\ 4 & \boxed{5} & 6 \\ 7 & 8 & 9 \end{array}$$

$$D_5 = \{2, 4, 6, 8\}$$

- Tout processus utilise les arbitres de sa colonne et de sa ligne
- Tout processus utilise 4 arbitres : ici $|D_i| = 4$
- Tout arbitre appartient au même nombre d'ensembles D_i : ici 4

Permissions d'arbitres

ATTENTION : pas de miracle . . .

Modèle parallèle processus $\leftrightarrow$ ressources critiques

- Un processus doit collecter des jetons $\equiv$ ressources critiques ;
- Problème : risque d'interblocage ;
  - Solution préventive : ordonner les jetons et les demander en respectant cet ordre.
  - Solution dynamique :
    - ordonner les requêtes (approche transactionnelle) et appliquer l'algorithme « *wound-wait* » ou « *wait-die* ».
    - nécessite un ordre total sur les requêtes :
      ☞ usage d'horloges de Lamport

Un algorithme très original : Trehel-Naïmi

Un algorithme distribué d'exclusion mutuelle en log(n), 1987

Principes

- Approche jeton à circulation intelligente ;
- Implantation d'une file d'attente répartie ;
- La tête de file est là où se trouve le jeton ;
- Gestion d'indications (*hint*) ;
- Insertions parallèles possibles dans la file d'attente ;
- Maintien d'un arbre conduisant au dernier élément de la file d'attente ;
- Principe clé : inversion d'arcs orientés : un arc de $i \rightarrow j$ est remplacé par un arc de $j \rightarrow i$. (*Path reversal*)

Un algorithme très original : Trehel-Naïmi

Un algorithme distribué d'exclusion mutuelle en log(n), 1987

Particularités majeures

- Représentation répartie d'une file d'attente :
  - Un site sait s'il est tête de file ;
  - Un site connaît son suivant s'il existe ;
- Un site a seulement une indication du dernier site de la file ;
- Ces indications maintiennent un arbre dont la racine pointe de dernier site de la file (lorsqu'aucune opération n'est en cours) ;
- Autorise l'exécution répartie d'insertions en parallèle sans blocage

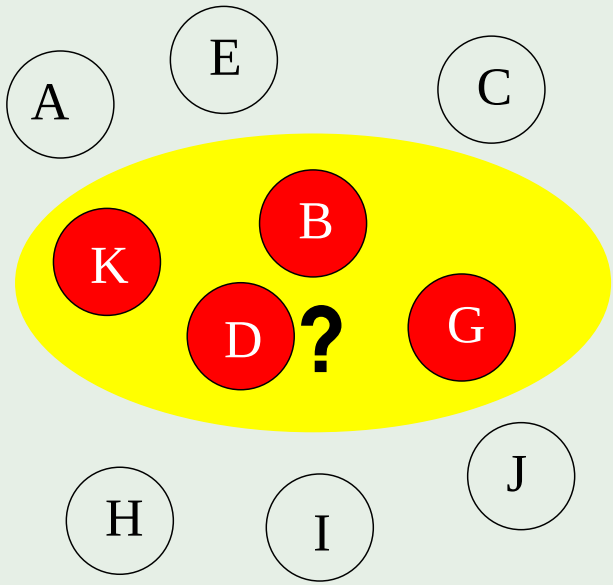
Plan

Spécification du problème

Détection d'une propriété stable

- Interblocage dû aux communications : attente de la réception d'un message
- Un processus est-il définitivement bloqué ?
- Sûreté : pas de fausse détection ;
- Vivacité : Un processus bloqué finit par le savoir ;

Exemple



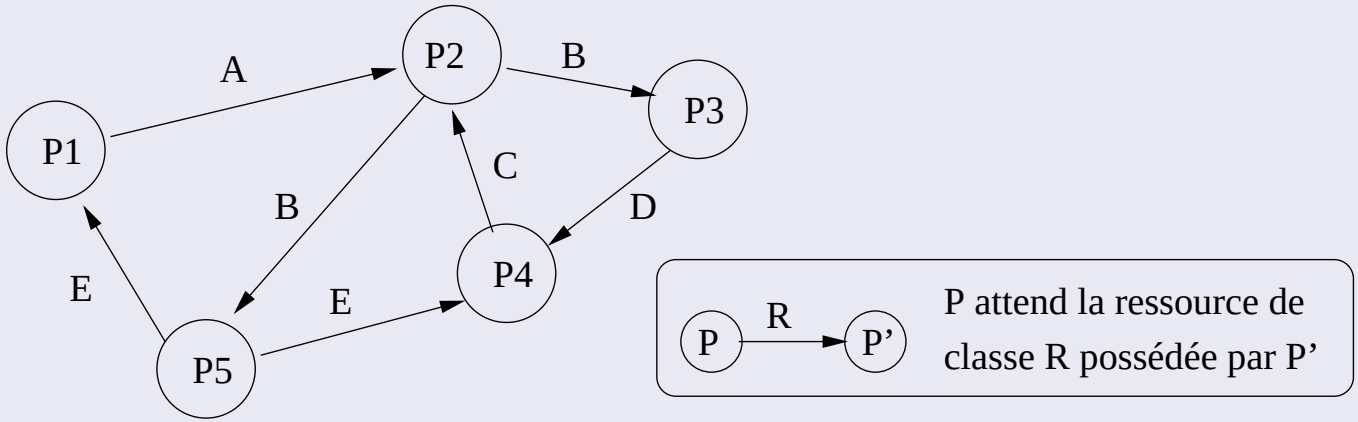
D s'interroge ?
K,B,D,G en interblocage

Caractérisation de l'état d'interblocage

Notion de composante fortement connexe terminale (CFCT)

Un sous-graphe G' d'un graphe $G = \{S, A\}$ est une CFCT [a] ssi il existe un chemin entre tout couple de sommets de G' et si tout sommet de G' a ses successeurs dans G' :

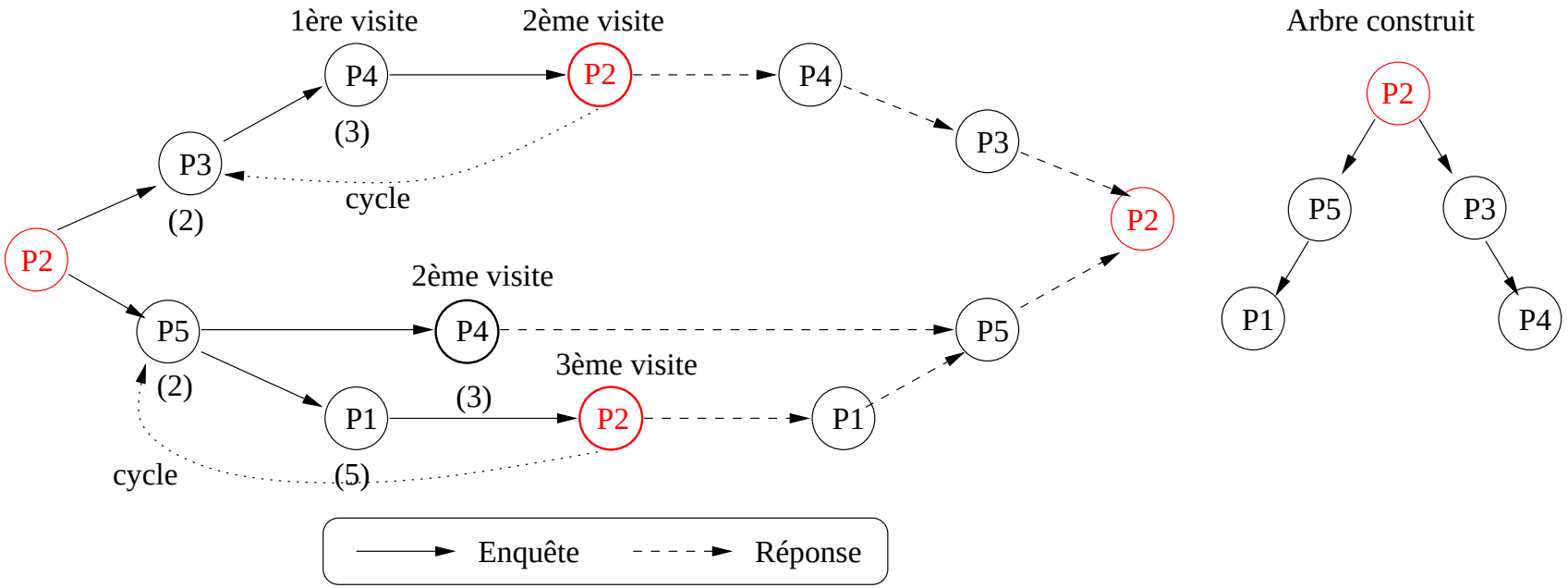
$$\forall s, s' \in G' : \exists s \xrightarrow{*} s' \wedge \forall s \in G' : suc(s) \neq \emptyset \wedge suc(s) \subset G'$$



a. *knot* en anglais

Algorithme : calcul diffusant et arbre de contrôle

de K.Mani Chandy, J. Misra, Laura M.Haas



Propriétés

- Terminaison de la construction de l'arbre $\Rightarrow$ P_2 est interbloqué
- Pas de terminaison : Il faudra recommencer...

Plan

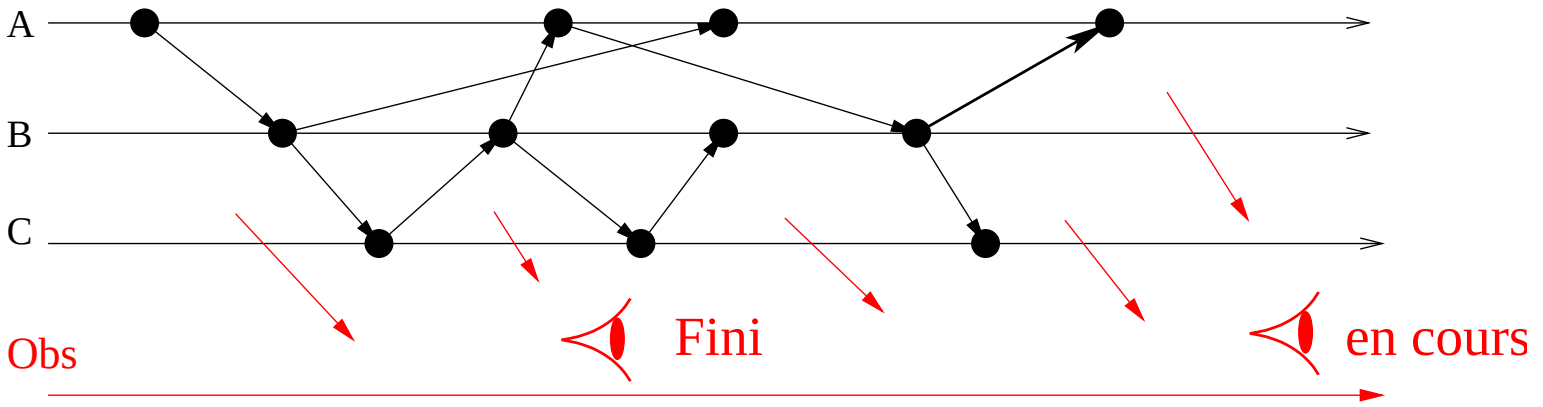
Détection d'un propriété stable

Spécification

- Propriété stable à détecter :

Tous les processus sont passifs ET pas de message en transit.

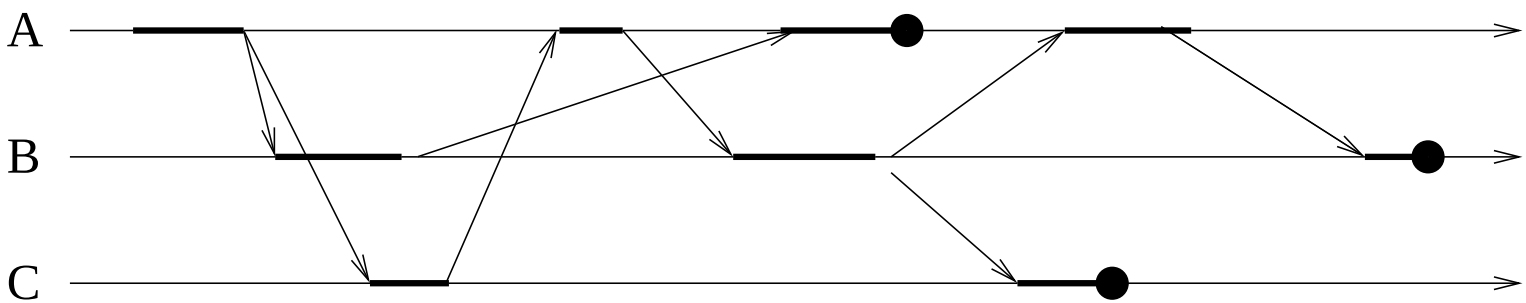
- Sûreté : Pas de fausse détection :
  $Term \Rightarrow (\forall\ i\ ::\ P_i.passif \wedge EnTransit = \emptyset)$
- Vivacité : La terminaison finit par être détectée :
  $(\forall\ i\ ::\ P_i.passif \wedge EnTransit = \emptyset) \rightsquigarrow Term$



Définition d'un calcul diffusant

- Un processus au moins émet un ou plusieurs messages ;
- Puis, tous les processus adoptent le même comportement :

```
loop {   /* un pas de calcul  */
         recevoir(m) ;
         traiter m ;
         envoyer 0 à N − 1 messages ;
}
```

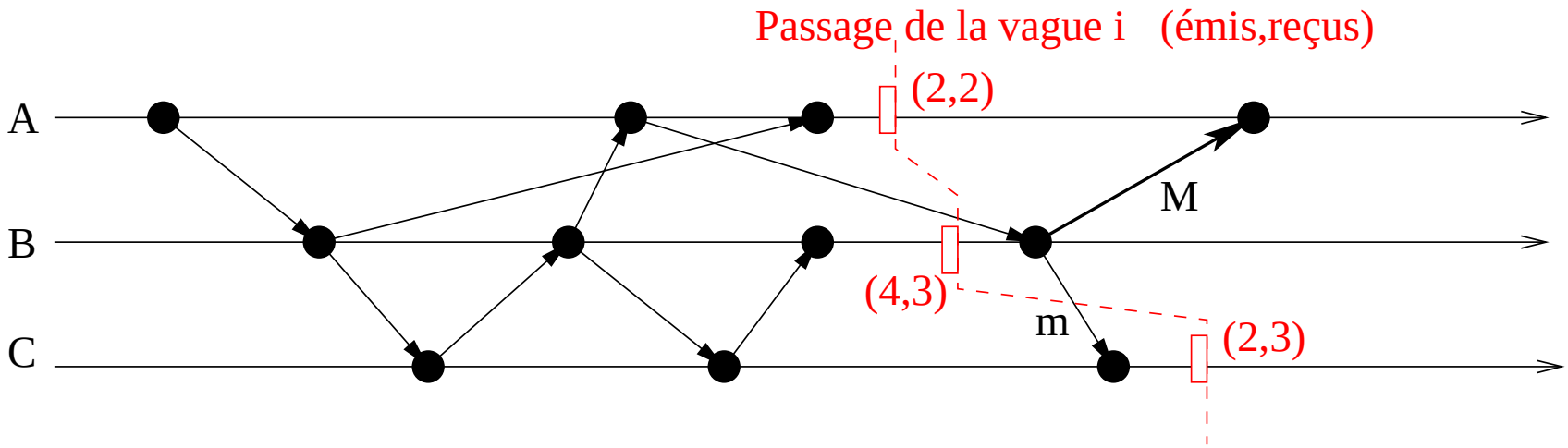


☞ Les phases actives peuvent être vues comme atomiques

Algorithme des compteurs (Mattern)

Principe

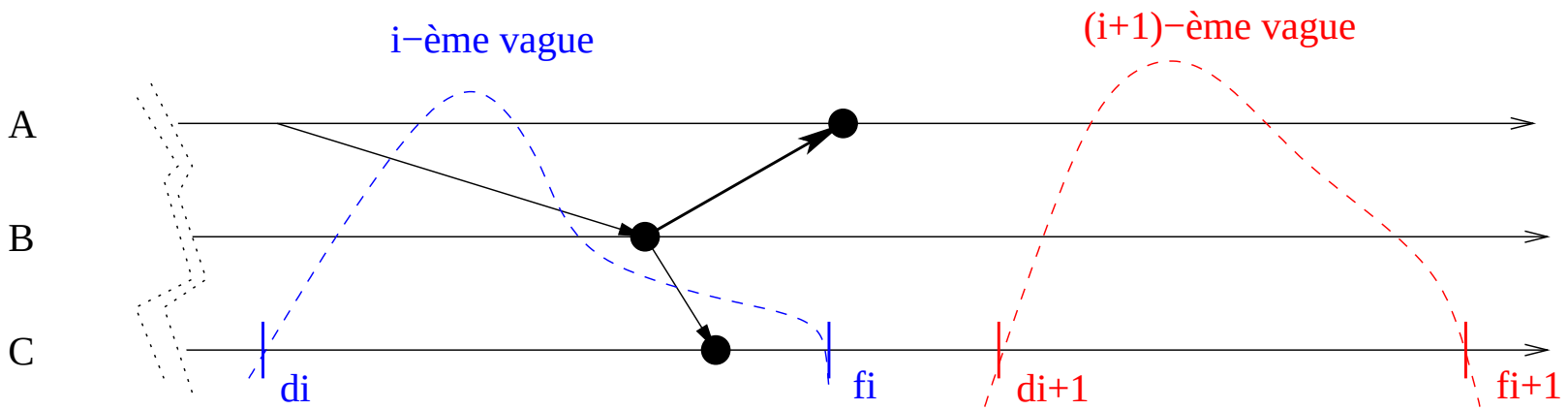
- Terminaison $\equiv E(t) == R(t)$ MAIS impossible à évaluer ;
- Approche : Compteurs locaux des messages émis et reçus ;
- Mécanisme de vague pour collecter les valeurs des compteurs ;
- La vague i collecte $R_i = \sum r_i$ et $E_i = \sum e_i$.



Algorithme des compteurs (Mattern)

Détection de la terminaison

- nécessite deux vagues successives ;
- Terminaison si : $R_i == E_{i+1}$ ($\Rightarrow \exists t < d_{i+1} : E(t) == R(t)$)
- Détection avec un retard d'au + la durée de la dernière vague ;



Algorithme des crédits (Mattern)

Un peu oublié mais pourtant simple et performant . . .

Principe

- Le processus initial possède un crédit de 1 ;
- Le crédit courant est partagé entre les messages émis ;
- Un processus rend son crédit s'il n'envoie pas de message ;
- Terminaison lorsque la somme collectée égale 1.