Plan

Modèle centralisé ou réparti

♪♪♪

Modèle centralisé

Les processus se partagent des ressources critiques ou pas

Modèle réparti

Les processus échangent des données par messages

Intérêts

♪♪♪

Apports de la répartition

- Accès aux ressources distantes et partage :
  - ressources physiques : imprimantes, traceurs...
  - ressources logiques : fichiers
  - données : textuelles, audio, images, vidéo
- Répartition géographique
- Puissance de calcul
- Disponibilité
- Flexibilité

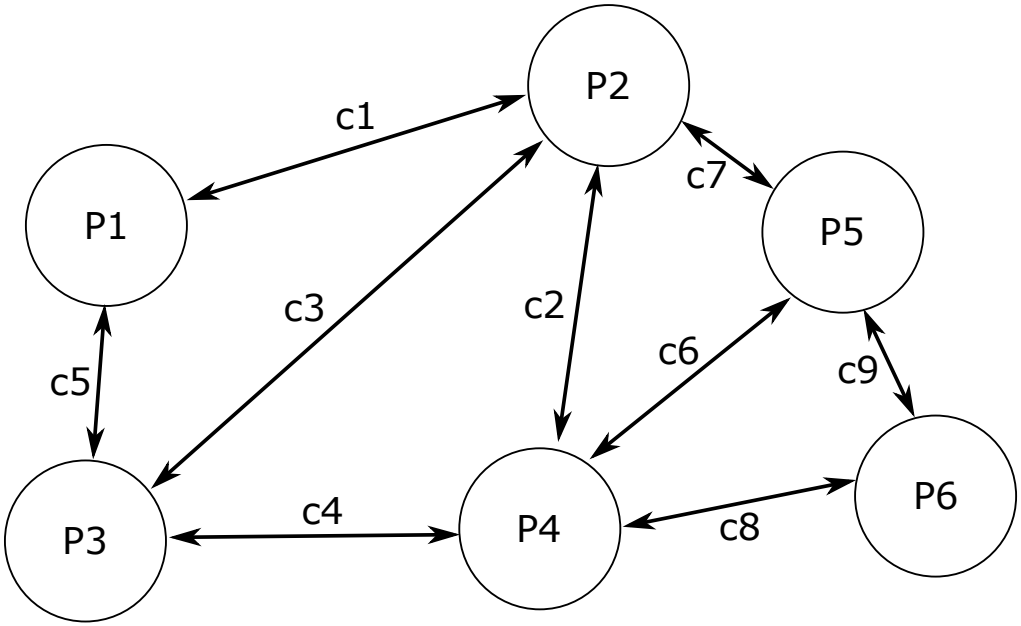
[Précis 1.3 pp.9–10]

Exemple : découvrir le graphe de communication ♪♪♪

Le problème

On considère un ensemble de sites connectés deux à deux par des canaux bidirectionnels.

Comment un site peut-il apprendre la structure du graphe ?



Hypothèses

♪♪♪

Connaissance initiale

Chaque site connaît son identité (id_i) et l'identité de ses voisins $voisins_i = \{id_x, id_y, \ldots\}$.
Le couple (id_i, id_j) représente le canal entre id_i et id_j (et symétriquement).
Aucun site ne connaît l'identité de tous les sites, ni leur nombre.

Communication

Un site peut envoyer/recevoir un message uniquement à ses voisins.

Découverte : principe de l'inondation ♪♪♪

Le problème redéfini

Comment chaque site id_i peut-il connaître l'ensemble des canaux (id_j, id_k) existants ?

Principe

- Un site i qui veut connaître le graphe envoie sa position $\langle id_i, voisins_i \rangle$ à ses voisins.
- La première fois qu'un site i reçoit un message, il s'active et envoie sa position $\langle id_i, voisins_i \rangle$ à ses voisins.
- La première fois qu'un site i reçoit un message $\langle id_k, voisins_k \rangle$, il le transmet à ses voisins et met à jour sa connaissance du graphe (pour k). Sinon, il l'ignore.
- Un site conclut quand il a reçu un message de tous les sites dont il a eu connaissance via les voisinages.

Algorithme pour le site i

Variables locales au site i :
id_i : son identité (const)
$voisins_i$: ses voisins (const)
$sites\_known_i$: les sites dont il a reçu un message
$channels\_known_i$: les canaux qu'il connaît

```
on start :
  for each id_j ∈ voisins_i do
      send ⟨id_i, voisins_i⟩ to id_j
  end for
  sites_known_i ← {id_i}
  channels_known_i ←
     { (id_i, id_k) :  id_k ∈ voisins_i }

  on reception ⟨id, voisins⟩ :
    if sites_known_i = ∅ then start(); fi
    if id ∉ sites_known_i then        -- premier message de id
      sites_known_i ← sites_known_i ∪ {id}
      channels_known_i ← channels_known_i ∪ { (id, id_k) :  id_k ∈ voisins }
      for each id_j ∈ voisins_i        -- propage le message
          send ⟨id, voisins⟩ to id_j
      end for
      if ∀ (id_j, id_k) ∈ channels_known_i : {id_j, id_k} ⊆ sites_known_i then
          id_i connaît le graphe. FIN
      endif
    endif
```

Questions pas triviales

- Terminaison : tous les sites finissent-ils par atteindre FIN ?
- Terminaison : un site peut-il savoir que les autres ont terminé ?
- Correction : à la terminaison de i, $channels\_known_i$ = le graphe ?
- Correction : après terminaison de tous, $\forall i, j : channels\_known_i = channels\_known_j$?
- Coût en messages ?
- Complexité en temps ?
- Résistance à un arrêt de site ?

Questions pas triviales

♪♪♪

- Terminaison : tous les sites finissent-ils par atteindre FIN ? (oui)
- Terminaison : un site peut-il savoir que les autres ont terminé ? (non)
- Correction : à la terminaison de i, $channels\_known_i$ = le graphe ? (oui)
- Correction : après terminaison de tous, $\forall i, j : channels\_known_i = channels\_known_j$? (oui)
- Coût en messages ? (2 * nombre de sites * nombre de canaux)
- Complexité en temps ? (2 * diamètre)
  (diamètre = $\max_{(i,j)} \min distance(i, j)$)
- Résistance à un arrêt de site ? (∼ok si le graphe reste connexe)

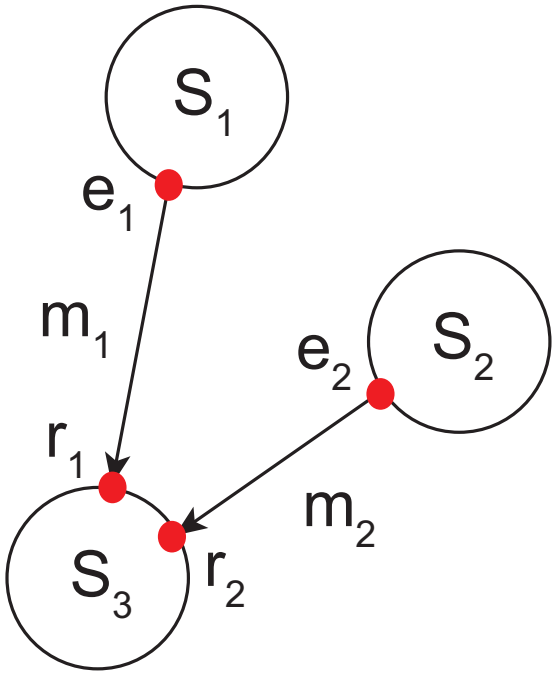
Modèle d'exécution plus complexe

♪♪♪

Problèmes...

- $m1$ est-il toujours envoyé avant $m2$ dans toute exécution ?
- $m1$ est-il toujours reçu avant $m2$ dans toute exécution ?
- Peut-on déduire ?

$$date(r_1) < date(r_2)$$
$$\Downarrow ?$$
$$date(e_1) < date(e_2)$$



Fort non-déterminisme : explosion des états possibles

[Précis 1.2 pp.7–9]

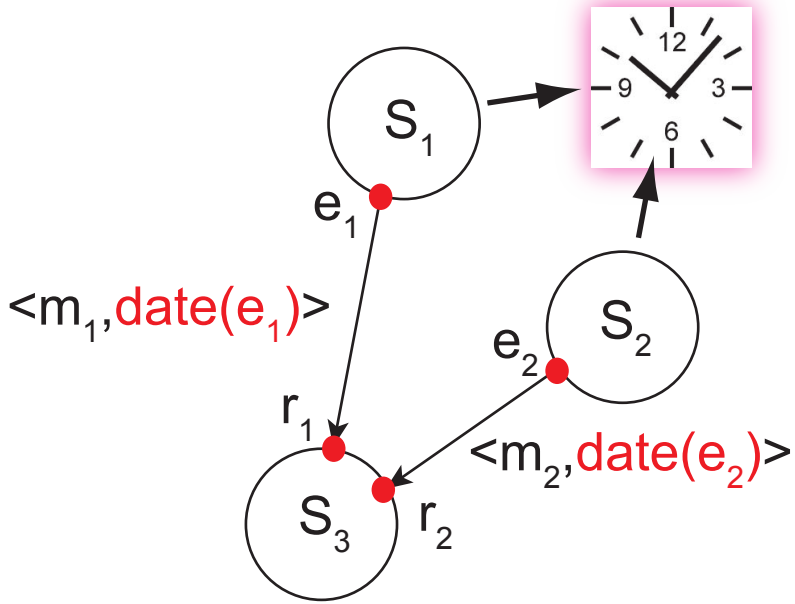
Épine : le temps

♪♪♪

Dates dans messages

$$date(e_1) < date(e_2)$$
$$\Downarrow ?$$
$$e_1 \text{ avant ? } e_2$$

Pas sûr, car
l'horloge n'existe pas ! ! !



Épine : pas de temps global

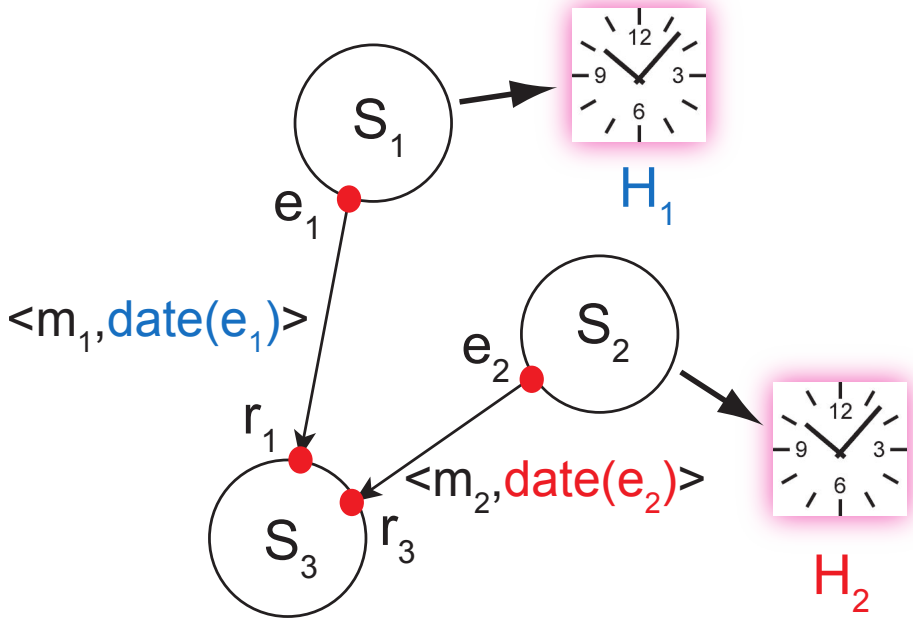
il existe 2 horloges

$$date(e_1) < date(e_2)$$

$$\Downarrow$$

e_1 avant e_2

Si les horloges sont synchronisées !

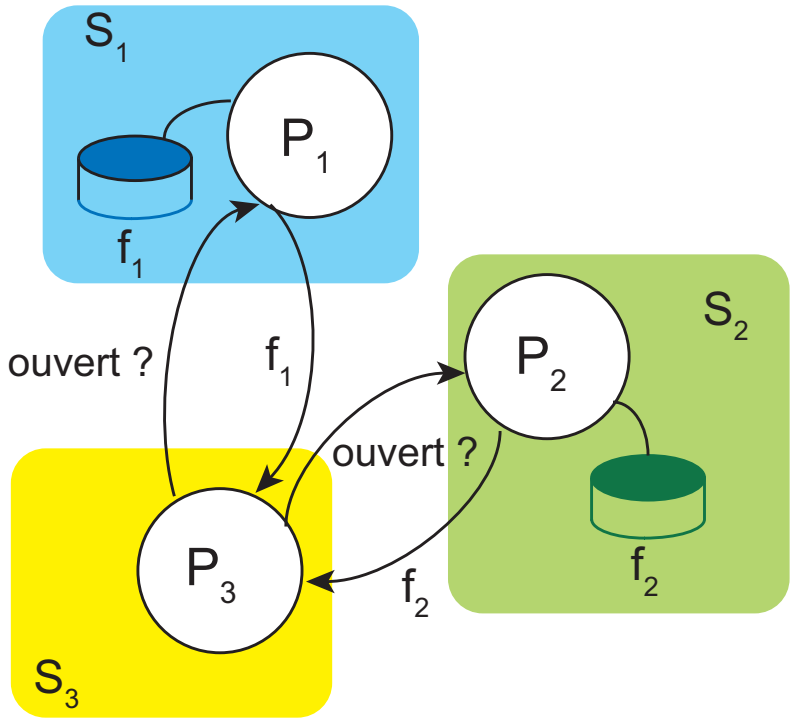


Pas de référentiel temporel unique

Épine : pas d'état global immédiat ♪♪♪

Problème

- P3 veut savoir si P1 ou P2 ont ouvert des fichiers ?
- Connaissance instantanée impossible



Un processus ne peut pas connaître instantanément l'état courant de ses partenaires : pas d'état global immédiat.

Épine : les défaillances

♪♪♪

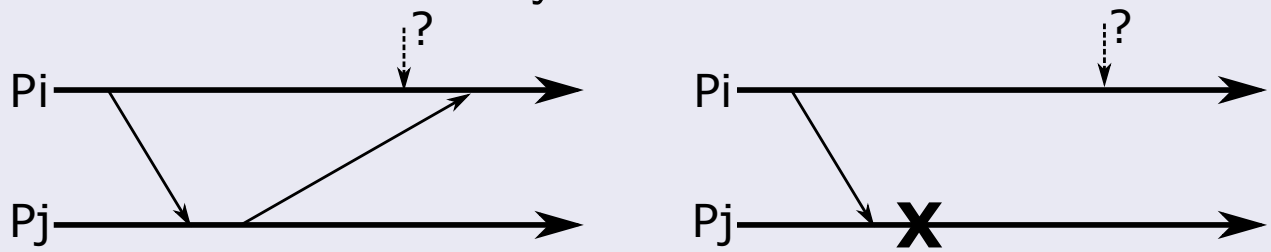
Défaillance de la communication

Perte de message, modification du contenu, ordre de délivrance

⇒ solutions réseau ou algorithmiques

Défaillance de site

- arrêt du site, réponse erronée, transition erronée
- défaillance partielle du système
- non détectable en asynchrone : lent ou cassé ?



≪ *A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable.* ≫ Leslie Lamport, 1987.

Les épines, en résumé

♪♪♪

Impact de la répartition

- Pas d'horloge globale : chaque site a son horloge
- Pas d'état global immédiat accessible à un site
- Fiabilité partielle : possibilité d'arrêt d'une machine, d'un processus quel que part
- Sécurité relative : usagers potentiels nombreux
- Non déterminisme (parallélisme) : systèmes asynchrones

Conséquence

Modèle de calcul différent du cas centralisé

- Ordre partiel entre les événements d'un calcul
- Calcul d'état global passé

Thèmes de recherche sur la répartition

Concevoir, modéliser, expérimenter

- Modélisation théorique
- Algorithmique
- Langages
- Systèmes d'exploitation
- Intergiciels (middleware)

[Précis 1.4 pp.11–12]