Systèmes et algorithmes répartis

Principes et concepts

Philippe Quéinnec, Gérard Padiou
queinnec@enseeiht.fr
http://queinnec.perso.enseeiht.fr/Ens/ens-so.html

Département Informatique et Mathématiques Appliquées
ENSEEIHT

4 octobre 2016

plan

Plan

Sources

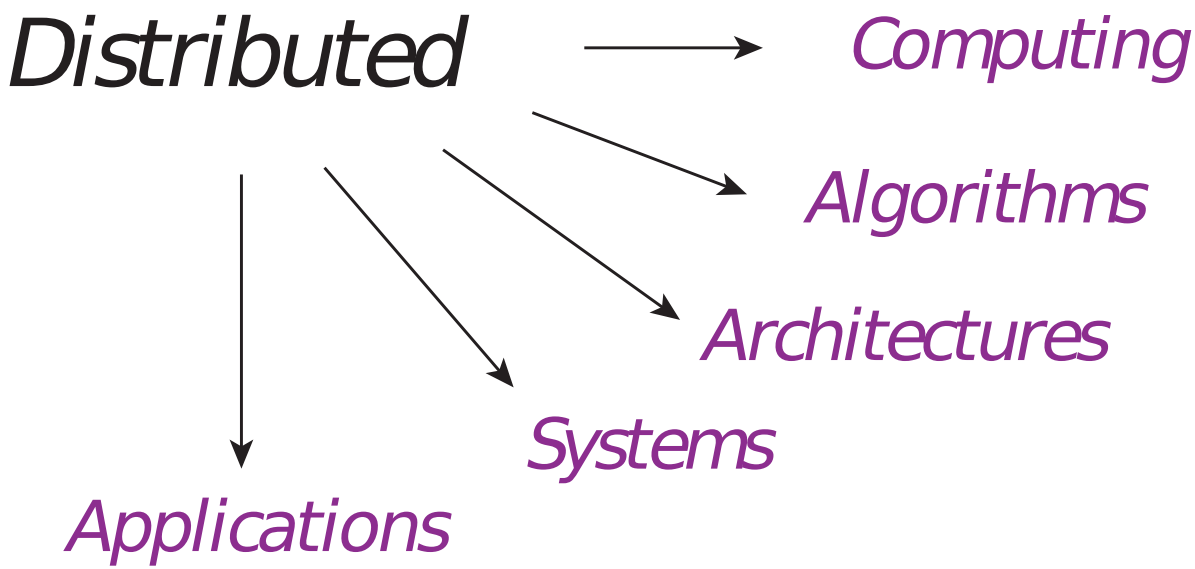
- M. Raynal, « *Distributed Algorithms for Message-Passing Systems* », « *Fault-Tolerant Agreement in Synchronous Message-passing Systems* » et « *Communication and Agreement Abstractions for Fault-Tolerant Asynchronous Distributed Systems* », 2010–2012
- S. Krakowiak, « *Algorithmique et techniques de base des systèmes répartis* », `lig-membres.imag.fr/krakowia/Files/Enseignement/M2R-SL/SR/`
- A.D. Kshemkalyani, M. Singhal, « *Distributed Computing : Principles, Algorithms, and Systems* », 2008
  `http://www.cs.uic.edu/~ajayk/DCS-Book`

Préambule : tendance

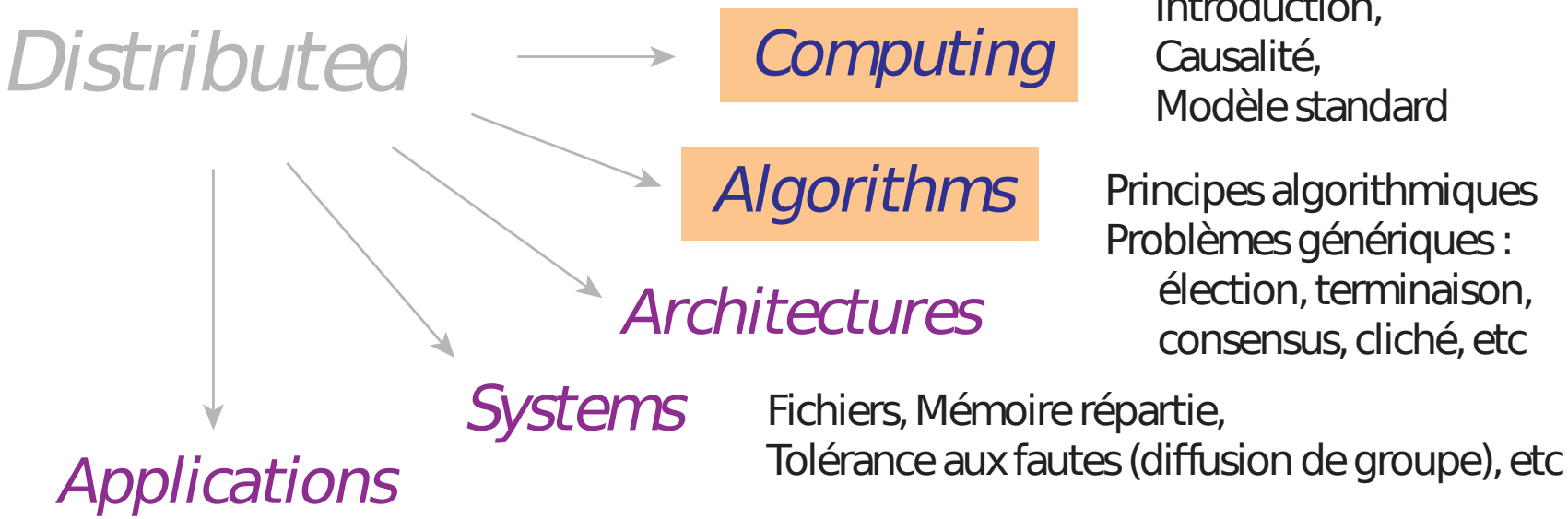
Répartition ≡ communication entre objets informatisés

- Actuellement : ordinateurs + téléphones + tablettes toujours connectés
- L'Internet des objets (*The Internet of things*)
  - 24 milliards d'appareils connectés entre eux en 2020
  - du porte-clefs au réfrigérateur en passant par les plantes
- L'informatique dans les nuages (*cloud computing*) : l'accès pour tous aux ressources/services informatiques

Préambule : aspects de la distribution (répartition)



Préambule : de quoi allons nous parler ?



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

Les parfums

Apports de la répartition

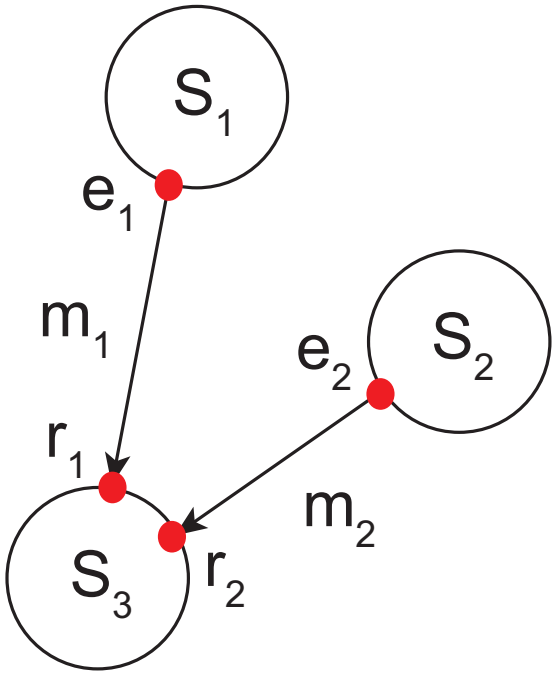
- Accès aux ressources distantes et partage :
  - ressources physiques : imprimantes, traceurs...
  - ressources logiques : fichiers
  - données : textuelles, audio, images, vidéo
- Répartition géographique
- Puissance de calcul
- Disponibilité
- Flexibilité

MAIS 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 ?

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



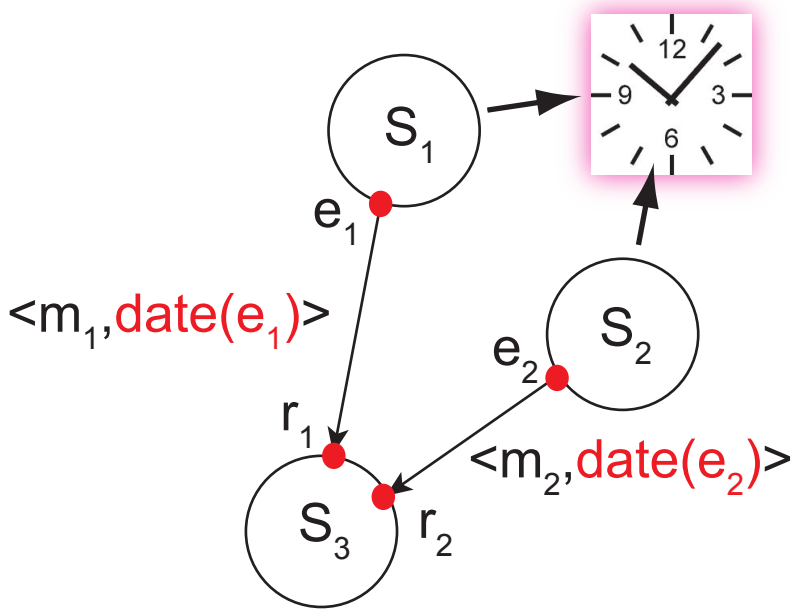
Fort non déterminisme : explosion des états possibles

Idée. . . pour en savoir plus

Dates dans messages

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

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



En réalité

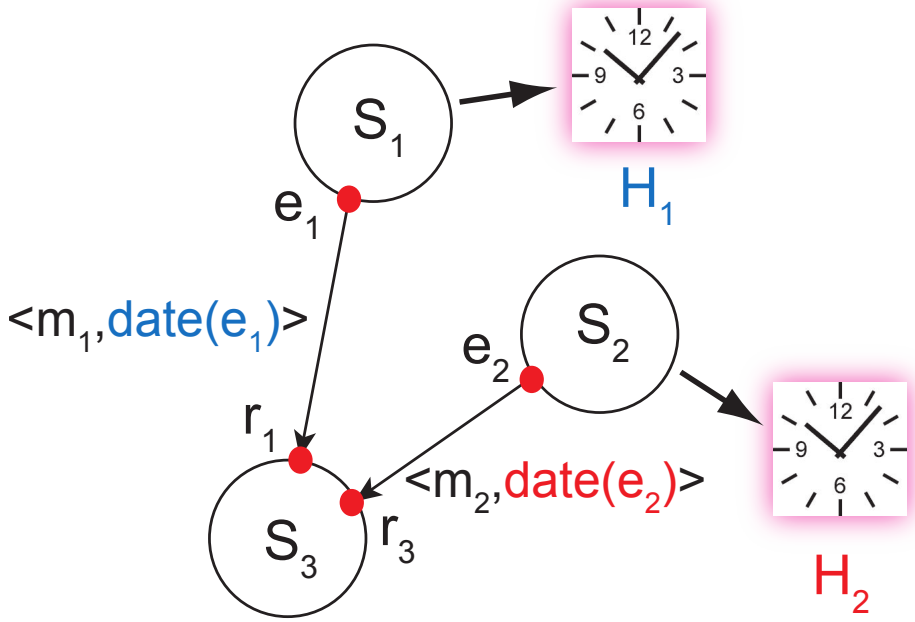
il existe 2 horloges

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

$$\Downarrow$$

e_1 avant e_2

Si les horloges sont synchronisées !

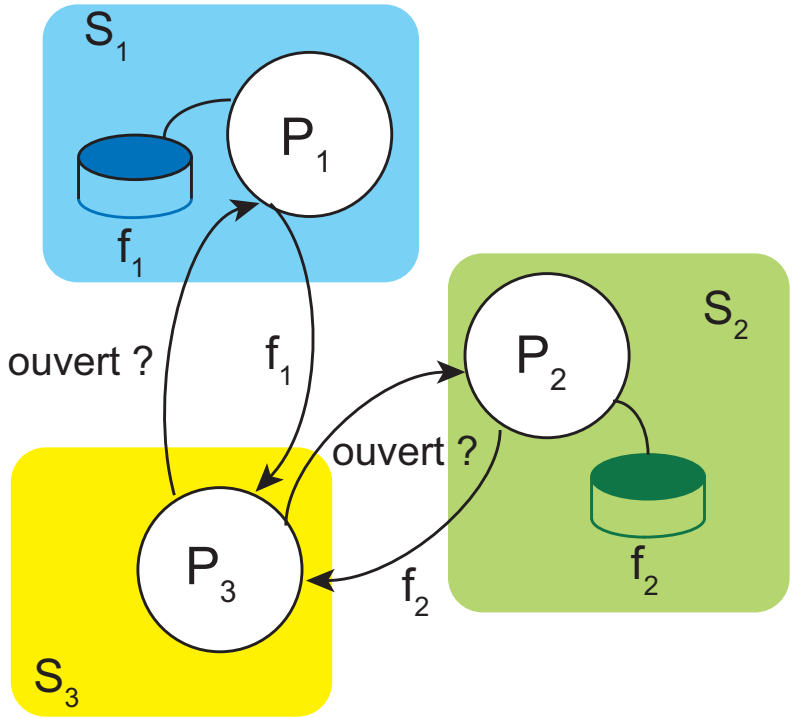


Pas de référentiel temporel unique

Modèle d'exécution plus complexe (suite)

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.

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 globale relative : probabilité non négligeable d'un 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

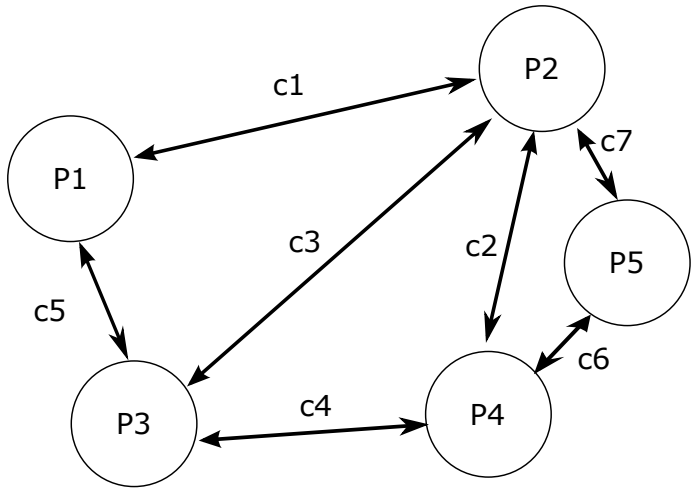
- Modélisation théorique
- Algorithmique
- Langages
- Systèmes d'exploitation
- Intergiciels (middleware)

Exemple : découvrir le graphe de communication

Le problème

On considère un ensemble de sites connectés 2 à 2 par des canaux bidirectionnels.

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



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.

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 diffuse à tous ses voisins sa position $(id_i, voisins_i)$
- Quand un site i reçoit un message $(id_k, voisins_k)$ pour la première fois, il met à jour sa connaissance du graphe et transmet le message à ses voisins. Sinon, il l'ignore.
- Il conclut quand il a reçu un message de tous les sites dont il a eu connaissance via les voisinages

Algorithme pour le site i

```
on start :
  for each id_j ∈ voisins_i do
      send (id_i, voisins_i) to id_j
  end for
  sites_known_i ← {id_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 a appris

```
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 ? (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 = $\max_{(i,j)} \min distance(i, j)$)

Principe de conception

Une idée clé : la transparence

Principe de conception 1

Un bon système réparti
est un système
qui semble centralisé
(qui s'utilise comme)

Principe de conception 2

Un bon système réparti
n'est pas un système centralisé

Idée : masquer la répartition

Niveaux de transparence

- Accès
- Localisation
- Partage
- Réplication
- Fautes
- Migration
- Charge
- Échelle

Mécanismes

- Interface
- Nommage
- Synchronisation
- Groupe
- Atomicité
- Mobilité
- Réflexivité
- Reconfiguration

Transparence d'accès

Propriété

Accès à une ressource distante $\equiv$ accès à une ressource locale

Exemple

- Niveau langage de commande : sh $\neq$ ssh (non transparence)
- Niveau service système : read,write identiques que le fichier opérande soit local ou distant (transparence)
- Niveau langage à objet : appel de méthode local ou à distance identique pour l'appelant (transparence)

Solution : Notion d'interface

Cas des intergiciels à objets : langage IDL et bus logiciel

Transparence de localisation

Propriété

La localisation d'une ressource reste cachée.

Exemple

- Non transparence : commande scp bach.enseeiht.fr:/foo .
- Transparence :
  - Niveau service système : open("nom-fichier",...) : nom du fichier indépendant de la localisation du fichier
  - Niveau langage à objet : références aux objets distants sans nécessité de connaître leur localisation

Solution : Services de nommage gérant des noms globaux

Cas des intergiciels à objets : serveurs de noms

Transparence du partage

Propriété

L'usage partagé (et en parallèle) d'une ressource doit rester cohérent ($\equiv$ sémantique équivalente au cas centralisé).

Exemple

- Niveau service système : cohérence d'accès à un fichier partagé : assurer les contraintes d'exclusion mutuelle des lecteurs/rédacteurs, mais coûteux
- Niveau langage à objets : limiter l'exécution en parallèle des méthodes sur un objet

Solution : Mécanismes de synchronisation

Problème : mécanismes connus mais souvent coûteux en réparti

Transparence de la réplication

Propriété

La répartition permet la redondance pour plus de fiabilité

Exemple

- Niveau service système : assurer le maintien de plusieurs copies cohérentes d'un même fichier
- Niveau langage à objets : assurer la réplication transparente d'un objet
- Niveau intergiciel : assurer que plusieurs serveurs répliqués évoluent en cohérence

Solution : Synchronisme virtuel

Notion de groupe et de protocoles de diffusion atomique

Transparence des fautes

Propriété

La répartition induit un contexte moins fiable que celui du centralisé : panne partielle

Exemple

- Niveau service système : un service n'est plus accessible (serveur de noms !)
- Niveau langage à objets : un appel à distance de méthode peut échouer...

Solution : Traitement d'exception et atomicité

Atomicité : un traitement s'exécute en entier ou pas du tout

Transparence de la migration

Propriété

Permettre la migration de code, de processus, d'agents, d'objets.

Exemple

- Niveau service système : déplacer un serveur d'une machine chargée à une machine sous-utilisée
- Niveau langage à objets :
  - code mobile : exemple des applets Java, exemple des fichiers postscript
  - objets mobiles (ou agents mobiles)

Solution : la mobilité des traitements et/ou des données

Agents mobiles (contexte d'exécution mobile), code mobile

Transparence de charge

Propriété

Masquer (et empêcher) les phénomènes de surcharge, écroulement

Exemple

La répartition permet naturellement la mise en œuvre de techniques d'équilibrage de charge

- Niveau système : reconfigurer dynamiquement les services sur les machines disponibles selon la charge des serveurs
- Niveau grappe (cluster) : répartir les traitements parallèles de façon équilibrée sur les différents processeurs

Solutions : réflexivité, machine virtuelle

Réflexivité : possibilité d'auto observation des composants
Machine virtuelle : dissocier environnement d'exécution et support matériel

Transparence d'échelle

Propriété

Permettre l'extension d'un système sans remettre en cause son fonctionnement global

Exemple

- Niveau système : introduire de nouveaux serveurs sur de nouvelles machines pour s'adapter à une augmentation de l'activité applicative

Solution : Adaptabilité et autonomie

Adaptabilité et autonomie : mise en œuvre de mécanismes automatique d'adaptation dynamique

En résumé

Répartition

$\equiv$

Accès et partage de ressources
via un réseau de communication
à tout usager qui en a le droit
et où qu'il soit

Plan du cours

I. Principes et concepts
II. Modèle standard et principes algorithmiques
III. Causalité et datation
IV. Problèmes génériques
V. Tolérance aux fautes
VI. Données réparties, systèmes de fichier répartis
VII. Grande échelle, pair-à-pair
VIII. Consensus, détecteur de fautes
IX. Simulation répartie

Plan

Modéliser un calcul réparti

Objectifs

- Description statique et comportementale
- Abstraction pour faciliter l'analyse
- Validation de propriétés (sûreté et vivacité)

Les éléments de modélisation

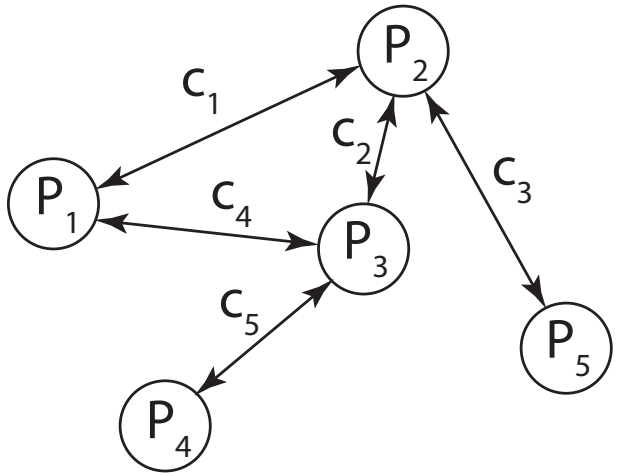
- Les activités, processus, sites, etc $\Rightarrow$ site logique
- La communication : liens, liaisons, canaux, protocoles (point à point, diffusion)...
- Les connaissances globales de chaque site logique

Modèle de calcul réparti

Aspect statique

Graphe structurel (statique)

- Sommets ≡ sites logiques
- Arcs ≡ liaisons
- Nombreux attributs :
  - Parallélisme
  - Fiabilité
  - Performances



Inconvénient : modèle statique, mais permet de poser les bases des problèmes de la répartition.

Modèle de calcul réparti

Aspect dynamique

- Description globale (dans un repère temporel global)
- Description événementielle
- Trois types d'événements : émission, réception, interne
- Modélisation de la communication : diffusion, perte, délais, etc