IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Interaction et Vérification

A. Beugnard
SIIA – IV – C2
2020

Pourquoi on communique ?

- Se coordonner, coopérer, négocier, . . .

Que communique-t-on ?

- Des informations (état, résultat, intention, . . .)

Comment communique-t-on ?

- Observation (1 actif/1 passif)
- Partage (canal, mémoire, conventions) de message (plusieurs actifs ; ex. expéditeur, destinataire)

- Protocole de transport (partagé)
- Langage de communication (partagé)
- Protocole d'interaction (partagé)

Ce cours se concentre sur le protocole d'interaction.

- Avec qui communiquer et comment le ou les trouver
- Comment initialiser et maintenir un échange
- (Hors scope) Effet de la communication

Quels critères de classification ?

- Acteurs actifs/passifs
- Nombre d'acteurs : 2, plus de 2
- Rôle des acteurs symétrique ou non
- Initiateur de la communication
- État partagé
- Asynchrone ou synchrone
- Attente blocante ou non
- Ordre des messages préservé (FIFO) [asynchrone]
- Perte de message possible [gestion des erreurs]

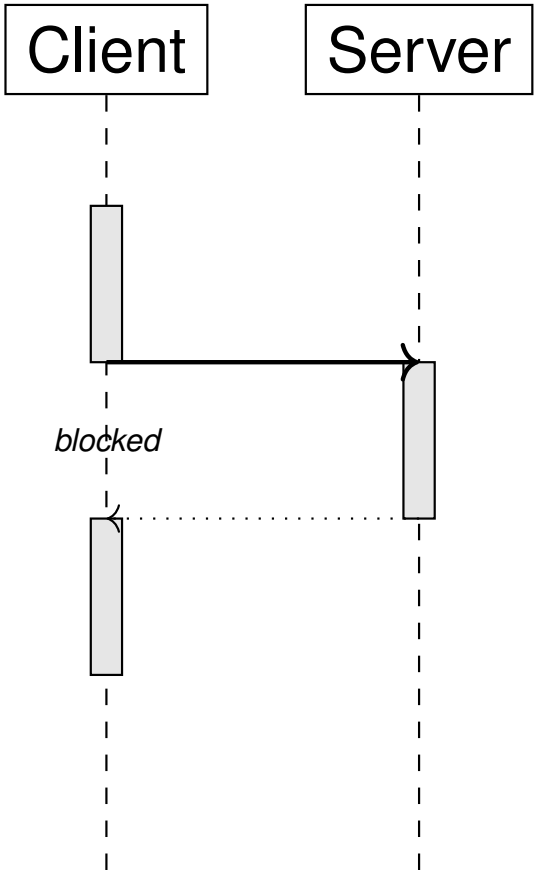
Quels critères de classification ?

- Informer
- Demande d'information
- Demander de faire
- Réponses
- Promesse
- Proposition
- Négociation
- Choisir, élire, décider
- …

2-interaction

4 modèles :

- Synchrone
- Asynchrone
- Future
- Par-nécessité [Car93]



Coté client

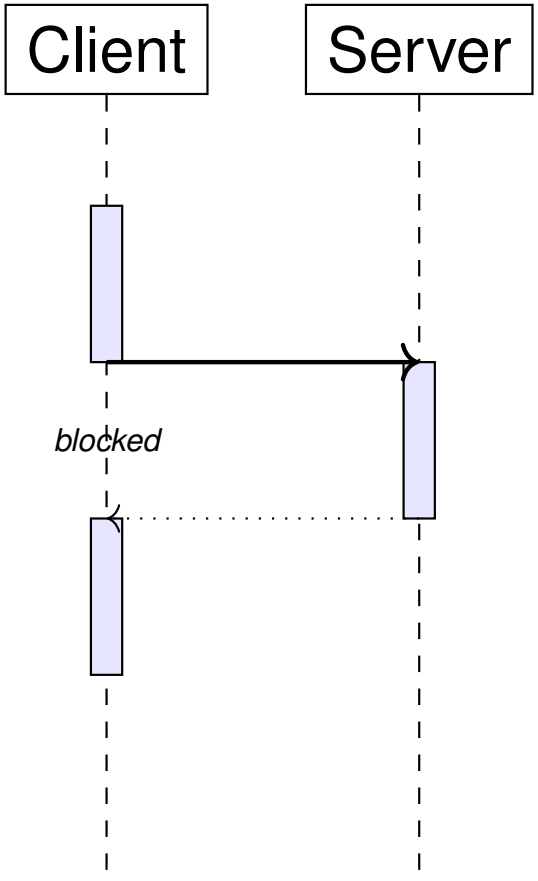
- call
- wait result
- get result
- continue

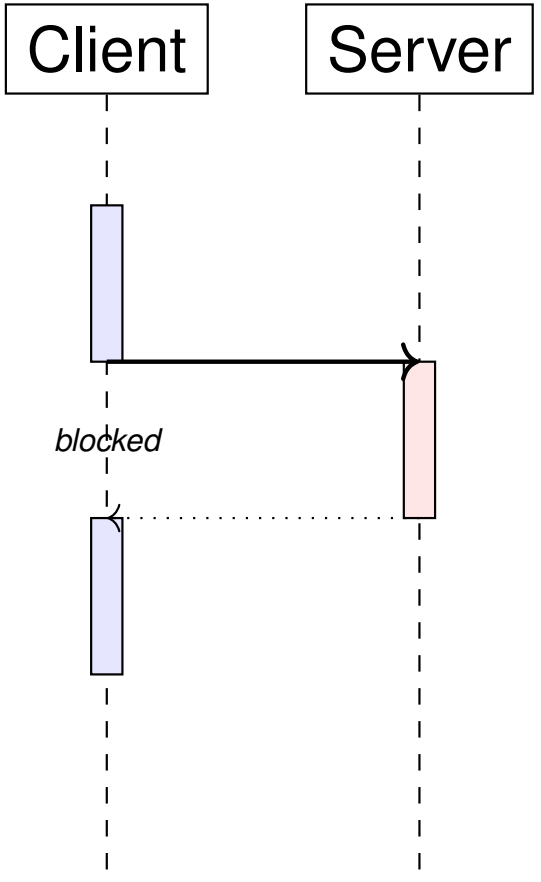
Pas visible (interne)

Coté serveur

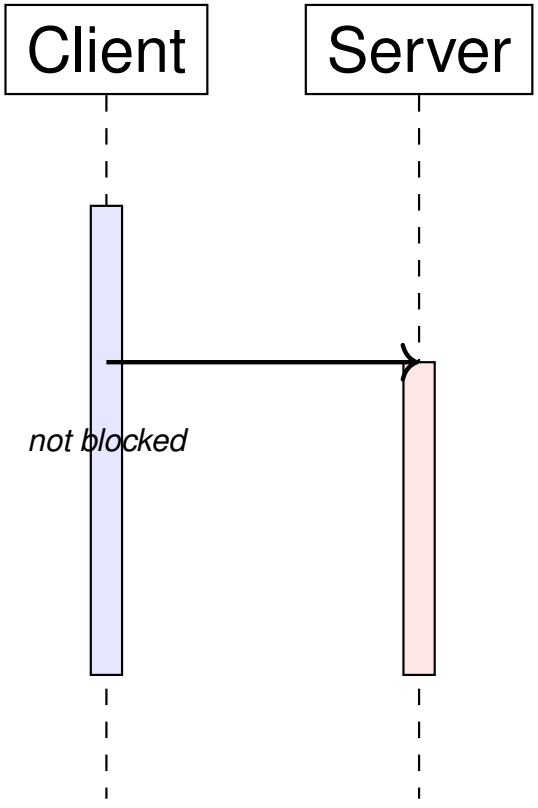
- wait call
- compute
- return result

Visible dans l'interface





Ou Rendezvous (ADA)



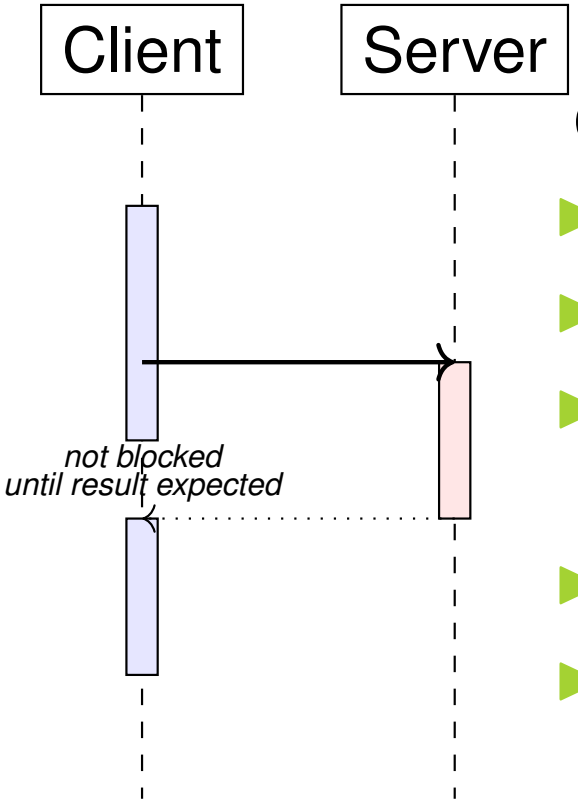
Coté client

- send message
- continue

Coté serveur

- wait message
- compute

On a nécessairement 2 fils d'exécutions . . .(soit avec des processus sur la même machine, soit sur des machines distantes.)



Coté client

- call
- continue
- wait result when needed
- get result
- continue

Coté serveur

- wait call
- compute
- return result

On a nécessairement 2 fils d'exécutions . . .(soit avec des processus sur la même machine, soit sur des machines distantes.)

Abstraction et mécanisme implicite qui se comporte comme

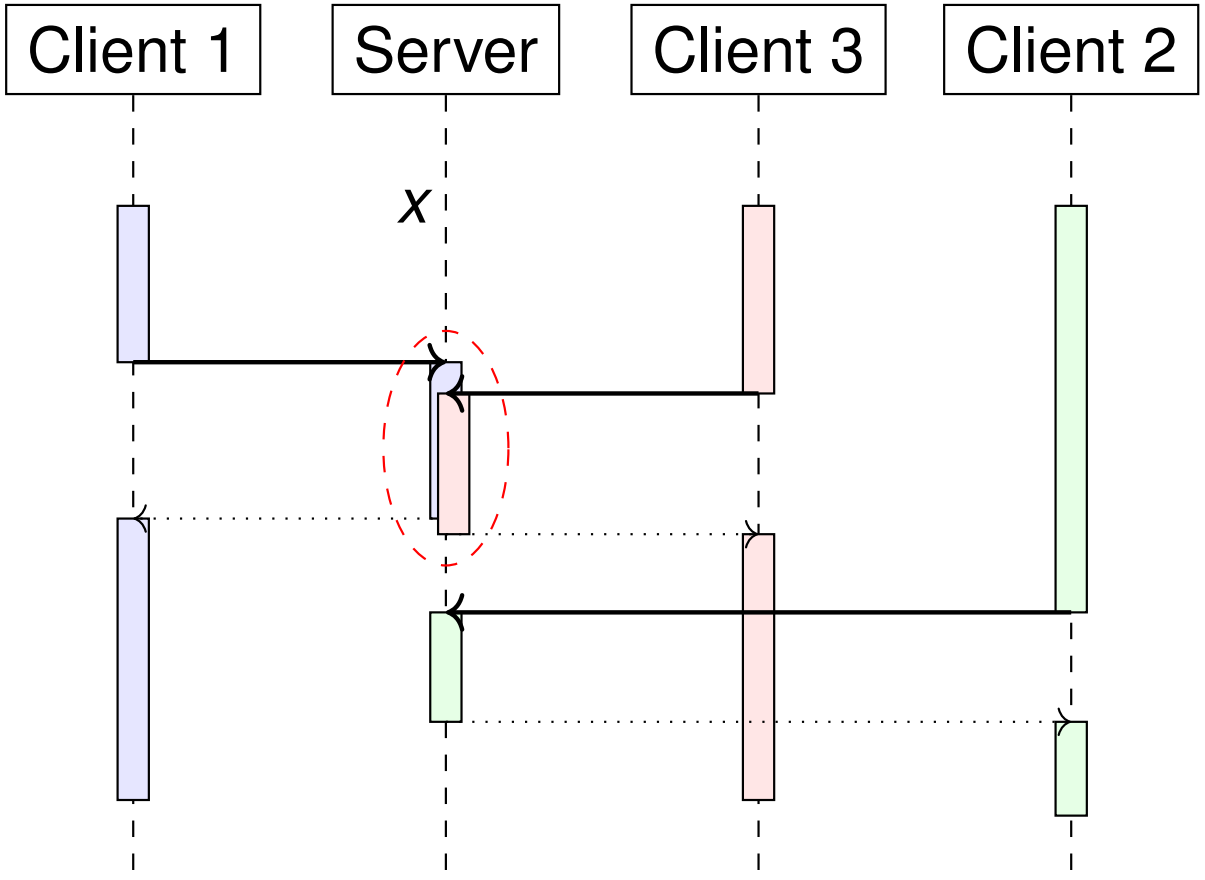
- Asynchrone ; pas besoin du résultat
- Synchrone ; besoin immédiat du résultat
- Future ; besoin du résultat plus tard

Les interactions ne sont intéressantes qu'entre entités actives [1].

Les entités passives ne sont utiles que pour partager du code
. . .et des états.

Le partage d'état est compliqué ; il introduit des problématiques d'exclusion mutuelle (sûreté) et d'interblocage (vivacité).

1. S'il n'y a qu'un seul fil, la communication est obligatoirement synchrone !



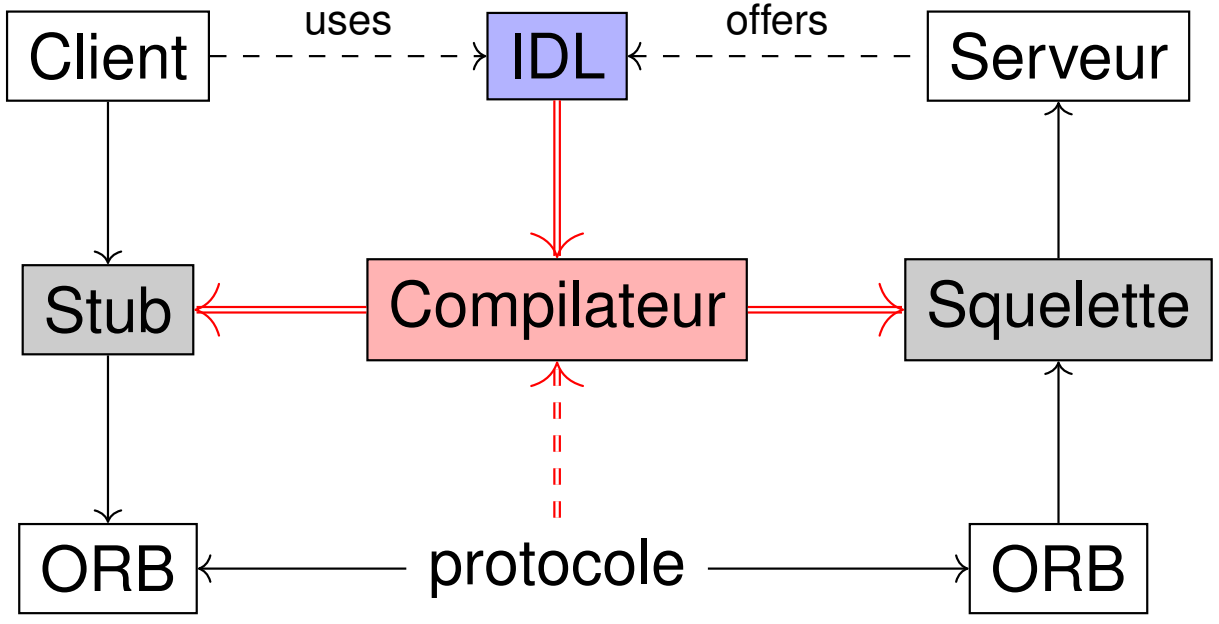
Partage, compétition pour l'accès à l'état x.
Dans certaines situation, on préconise une approche REST (Representational state transfer).

Sûreté propriété garantissant que rien de mal ne se produit. Par ex. respect d'un invariant, exclusion mutuelle...

Vivacité propriété garantissant qu'il se passe des choses. Par exemple absence d'interblocage.

Synchrone	**Asynchrone**	**Future**	**Par nécessité**
ADA, C, Caml, fonctionnel, objet	dart, erlang, elixir, via bibliothèques	via bibliothèques	ProActive [OW217], via bibliothèques

Les propriétés précédentes s'appliquent sur des interactions à distance (ou hétérogènes) :



Principe des connecteurs (voir) utilisé par RPC, CORBA, Java RMI, .NET, etc.

2+-interaction

Des interactions avec plus de 2 utilisateurs :

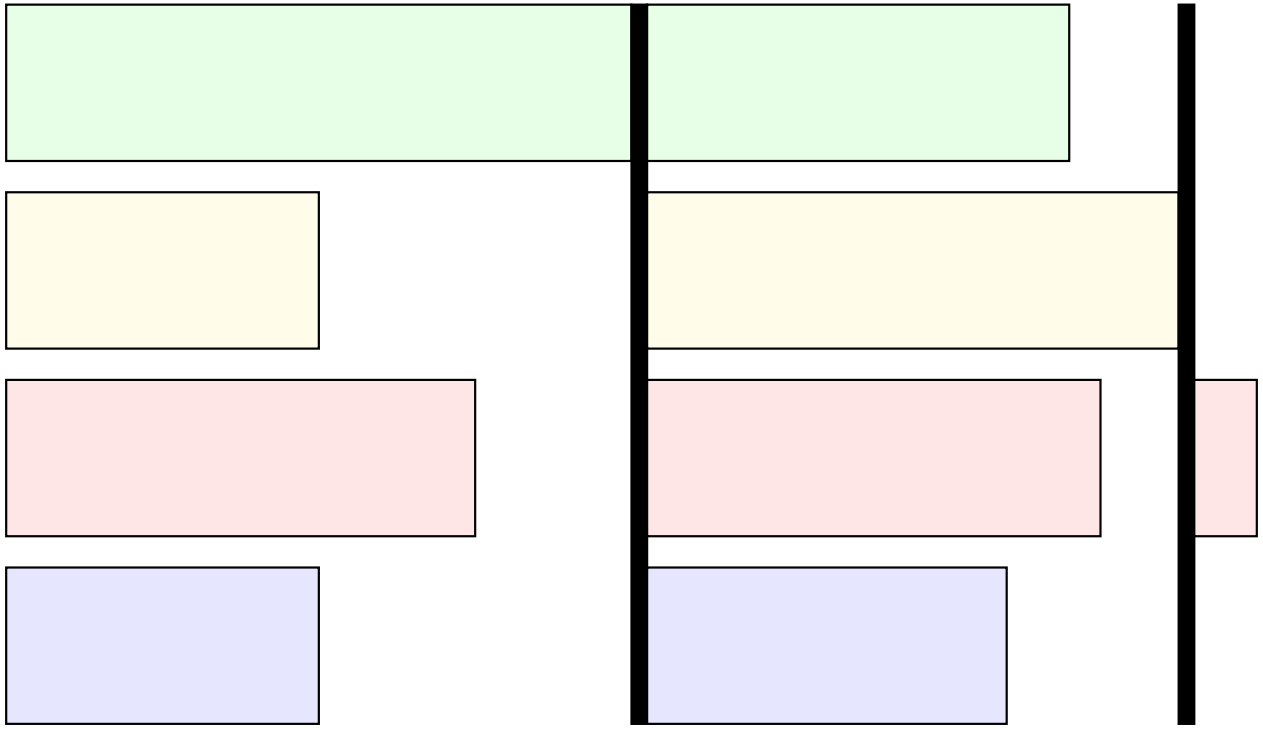
- Barrière de synchronisation
- Diffusion
  - Asynchrone (ex ; UDP)
  - Avec garanties. . .
- Consensus
- Group membership
- Espaces de tuples
- Langage de conversation

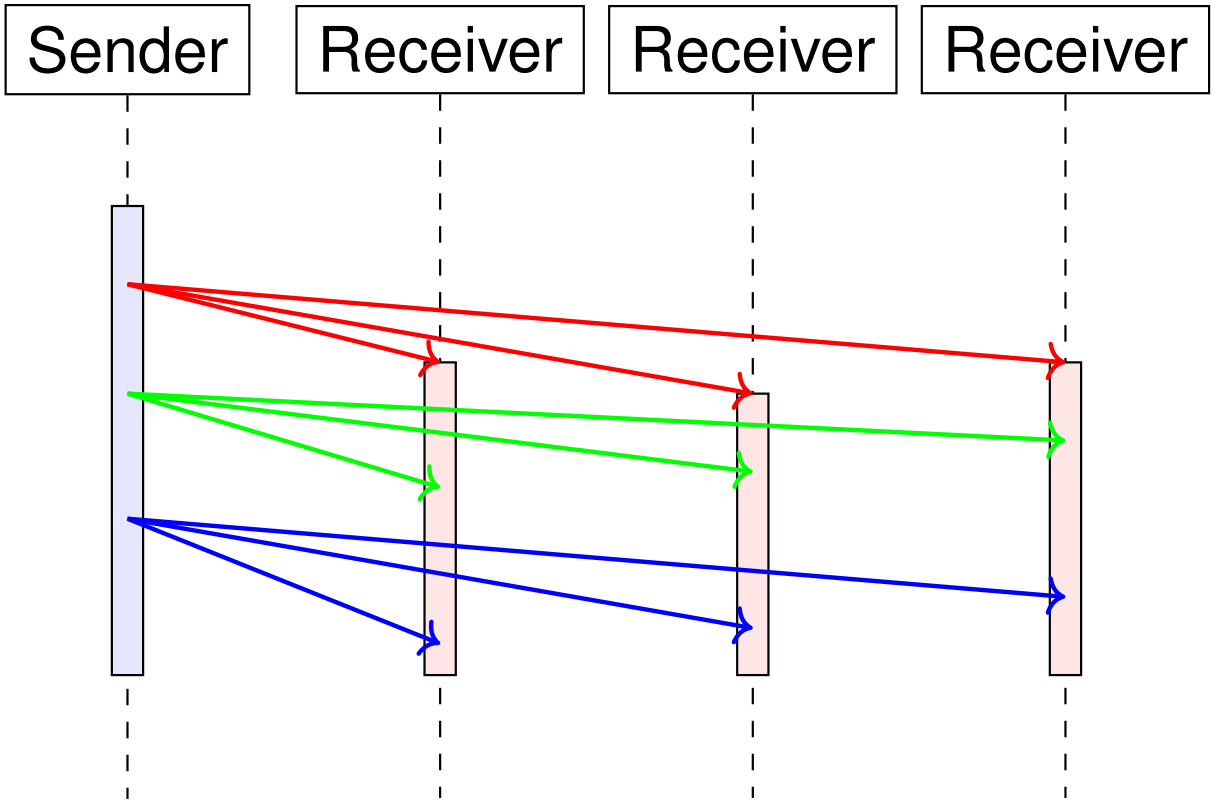
Des interactions plus abstraites :

- Publish/subscribe [EFGK03]
- Négociation
- Vote
- Enchères
- …
- Abstractions de communication

Au prochain cours

Mécanisme de coordination élémentaire qui garantit que des entités aient atteint un point spécifique (la barrière) pour continuer leur activité.





Propriétés

- pas de perte de message
- équité : garantie que tous reçoivent
- atomicité : tous ou personne
- les messages ne se doublent pas

UDP : perte et aucune garantie
Algorithme de diffusion fiable (*reliable broadcast*)
Garantir l'ordre des messages (rouge, vert, bleu)

Garantit qu'un message envoyé à un groupe est reçu par tous, sinon par aucun.

Les systèmes distribués étudient les interactions entre processus (machine, acteurs, agents) distants en prenant en compte :

- Les délais de transmission
- Les erreurs potentielles des acteurs
- Les erreurs potentielles des canaux de communication

Des algorithmes distribués proposent des solutions pour maîtriser les propriétés quand des solutions théoriques existent.

Les messages ne sont pas transmis instantanément ; ils peuvent être perdu.
Il est impossible de distinguer une perte de message d'un délais très long de transmission.
La notion de temps global n'a pas de sens - chaque acteur possède son propre temps - ; il est impossible de dater de manière absolue un événement.
La notion de causalité doit être reconstruite ; horloge de lamport, horloge vectorielle, etc.

On dit que m_1 précède causalement m_2 ($m_1 \rightsquigarrow m_2$) ssi :

- p a diffusé m_1 avant de diffuser m_2
- p a reçu m_1 puis a diffuser m_2
- Il existe m_3 tel que $m_1 \rightsquigarrow m_3 \wedge m_3 \rightsquigarrow m_2$

Il faut prendre en compte le fait qu'un acteur (processus, programme, machine, canal, réseau, humain, etc.) peut faire des erreurs.

- Par omission ; oublie d'envoyer un message, de répondre, ...
- Arbitraires ; envoie d'un mauvais message (volontairement [2] ou non)

Pour les omissions, le modèle le plus simple consiste à considérer qu'un acteur tombe en panne (crash-stop model) ; quand il omet d'envoyer un message, il omet d'envoyer tous les suivants...

2. on dit que l'acteur est malicieux ou Byzantin.

Perfect (or Reliable) links (PL)

- (Validity) Si p_i et p_j sont corrects, alors tout message émis par p_i vers p_j est finalement délivré à p_j
- (No duplication) Aucun message n'est délivré plus d'une fois
- (No creation) Aucun message n'est délivré sauf à avoir été émis

Reliable FIFO links (FIFO)

- Perfect links
- (FIFO) Les messages sont délivrés dans le même ordre que leur émission

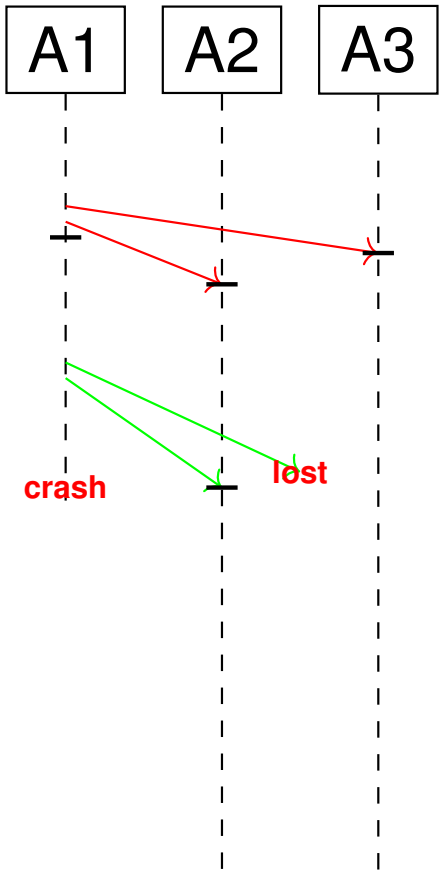
Dans le cours nous ferons l'hypothèse de canaux fiable (PL)

Best-effort Validity, No duplication, No creation (comme PL)

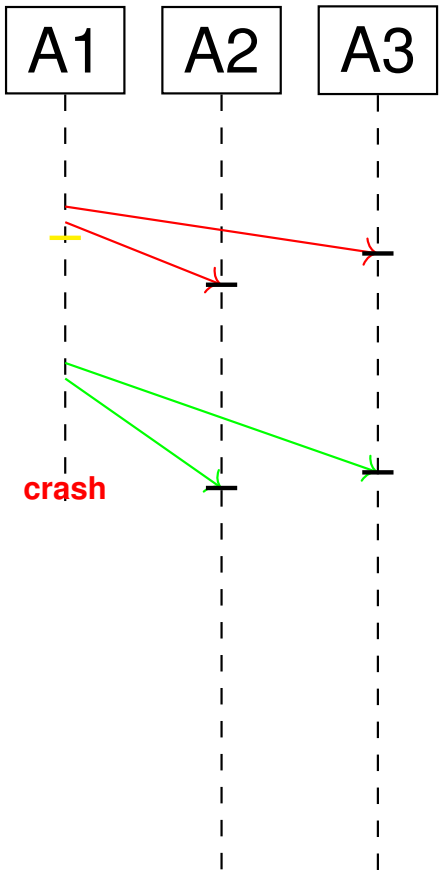
Reliable BE + Agreement : Si un message m est délivré à un destinataire correct, alors tous les destinataires corrects recevront le message.

Uniform BE + Uniform agreement : Pour tout message m, si un destinataire reçoit m alors tous les destinataires corrects reçoivent m

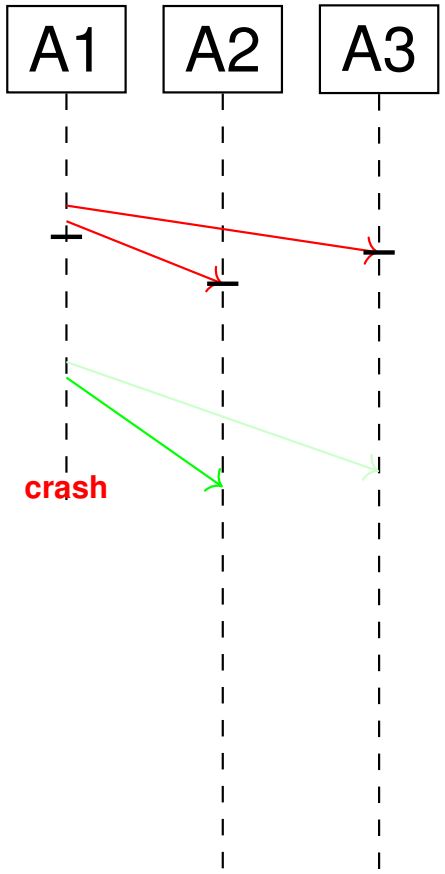
Pour les algorithmes, voir le cours de R. Guerraoui par exemple.



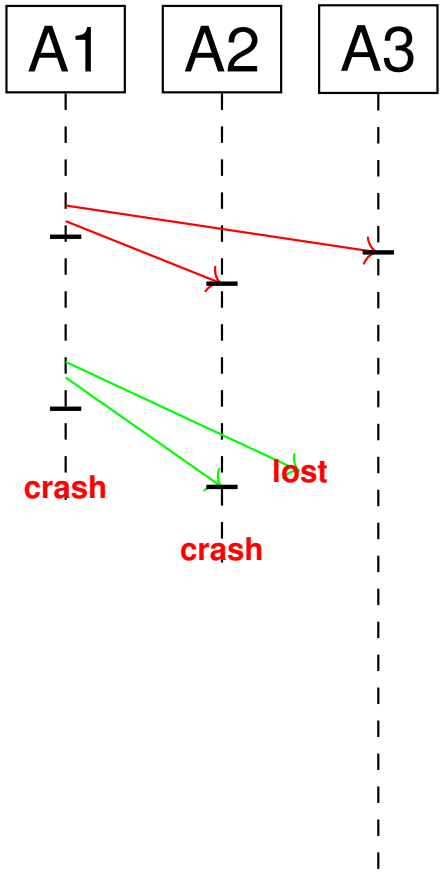
A1 fait de son mieux, mais bien que correct A3 ne reçoit pas un message que A2 a reçu.



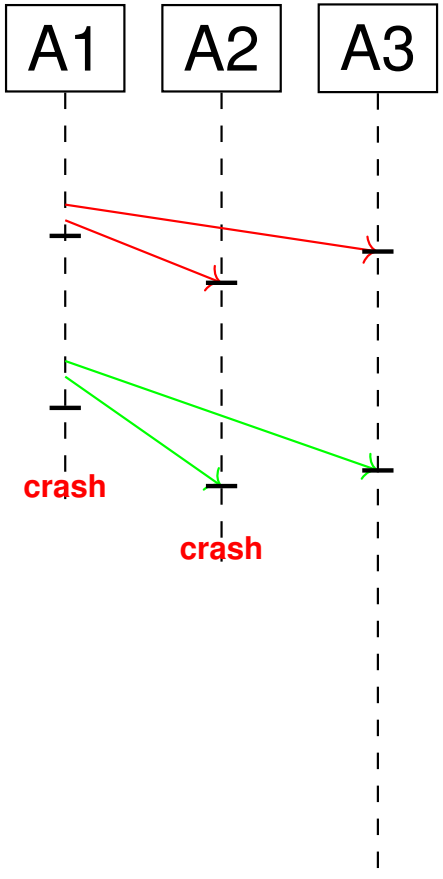
Tous les corrects reçoivent le message.



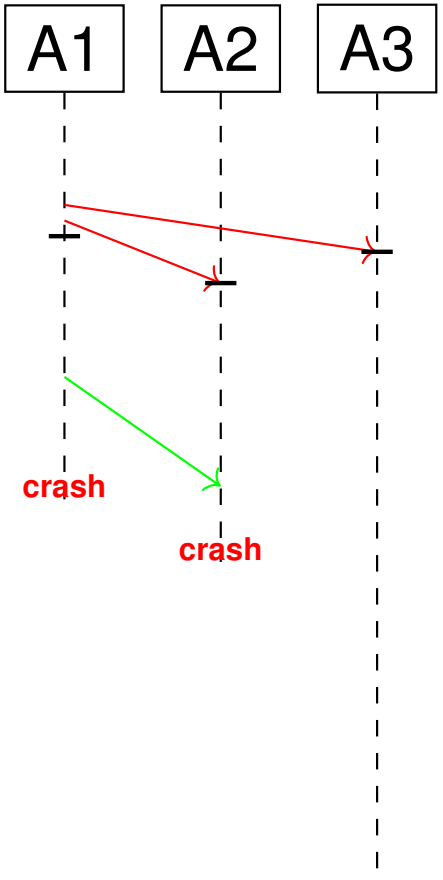
Ou aucun.



Diffusion fiable, car A2 non correct, *m*2 peut être perdu.



Même si un récepteur crash, c'est tout le monde. . .

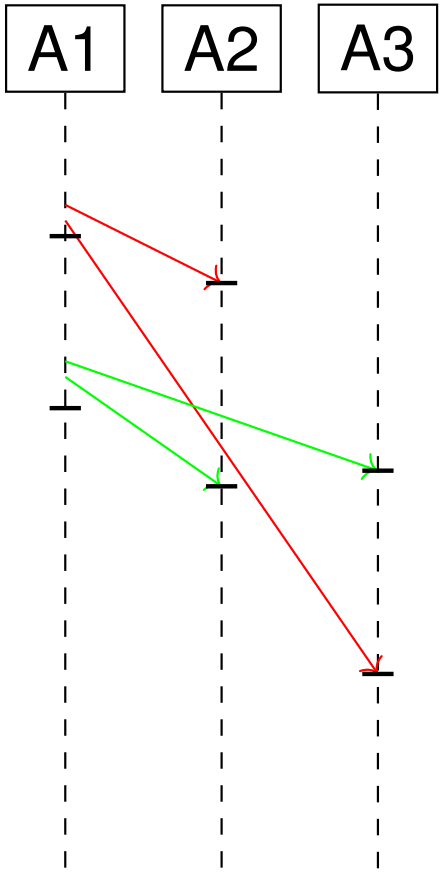


. . .ou personne.

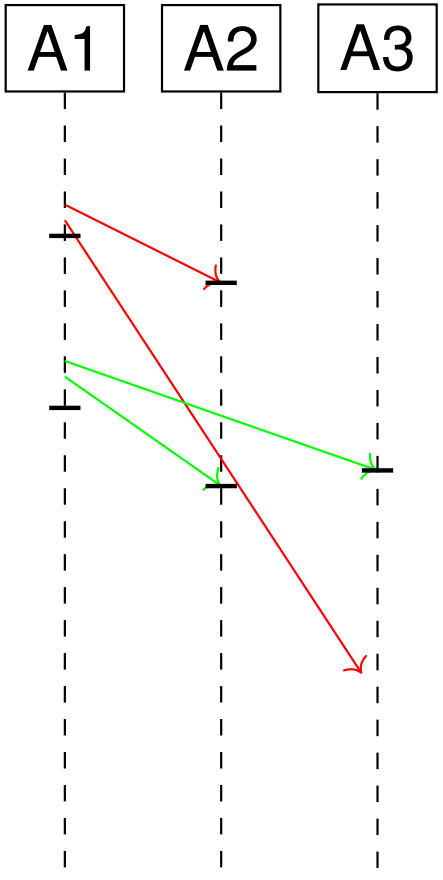
Garantit l'ordre des messages

Si p reçoit m_2 alors p a reçu tous les m tels que $m \rightsquigarrow m_2$

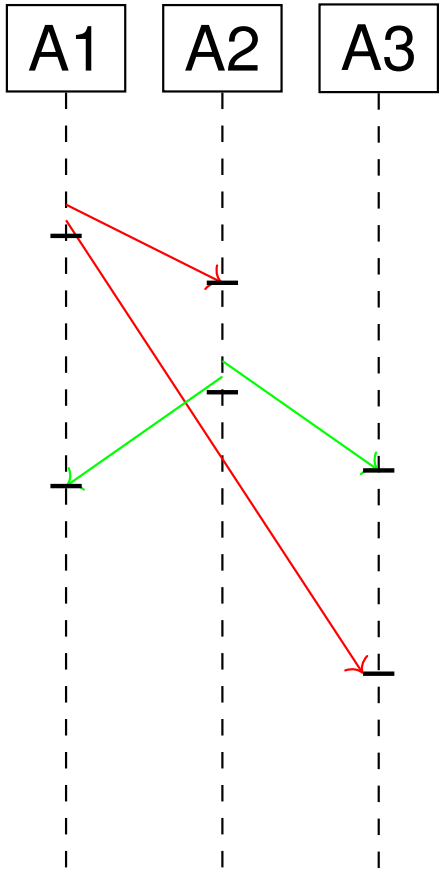
Causalité respectée ?



Non.



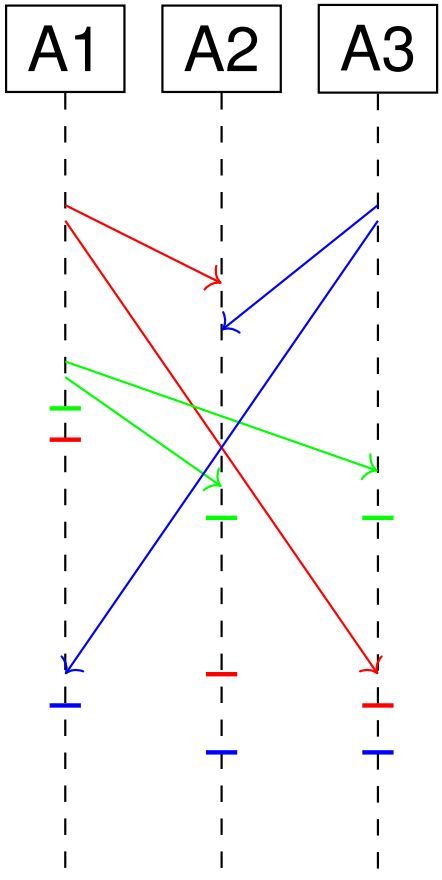
Non.



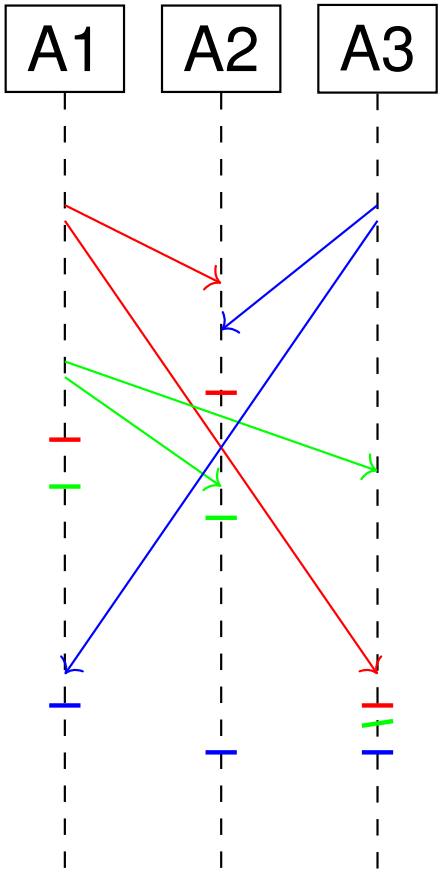
Non.

Garantit que tous les participants voient le même ordre de messages.
Si l'ordre choisi respecte la causalité on parle de diffusion causale totale.

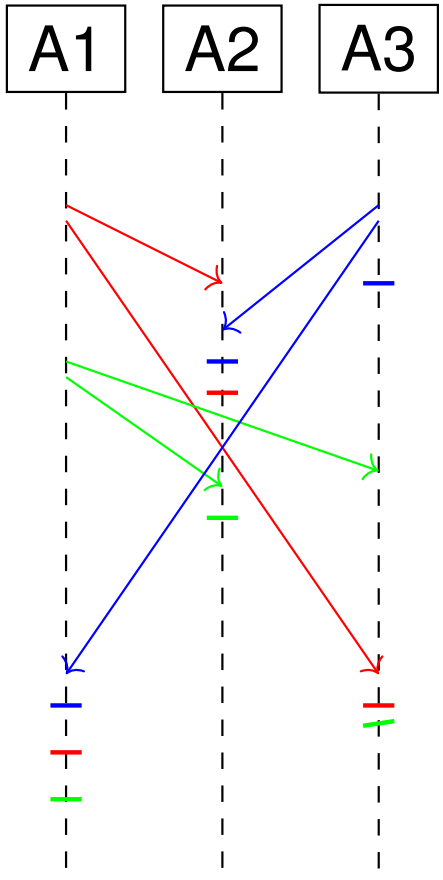
Ordre total ?



Total, pas causal



Total et causal



Total et causal

Le bleu est indépendant causalement des rouges et verts

Best-effort broadcast

- Garanti la fiabilité *si l'émetteur est correct*

Reliable broadcast

- Garanti la fiabilité *même si l'émetteur est incorrect*

Uniform reliable broadcast

- Prend en compte *les récepteurs incorrects*

Total reliable broadcast

- Reliable broadcast *avec même ordre de réception*

Causal reliable broadcast

- Reliable broadcast *avec respect de la causalité*

Une façon de réaliser un ordre total est de s'appuyer sur un algorithme de consensus.

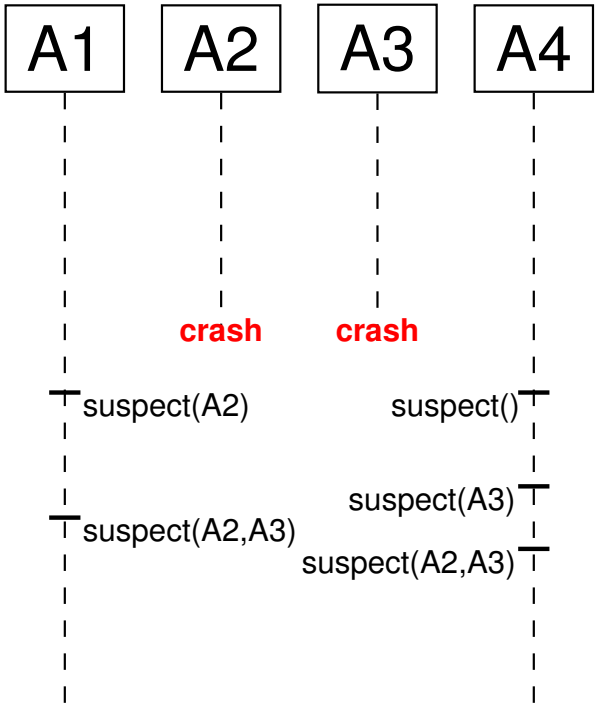
Consensus

- (Validity) La valeur choisie a été proposée
- (Uniform agreement) : Deux acteurs différents et corrects ne peuvent choisir différemment
- (Termination) Tout acteur correct finit par choisir
- (Integrity) Chaque acteur choisit une fois au plus

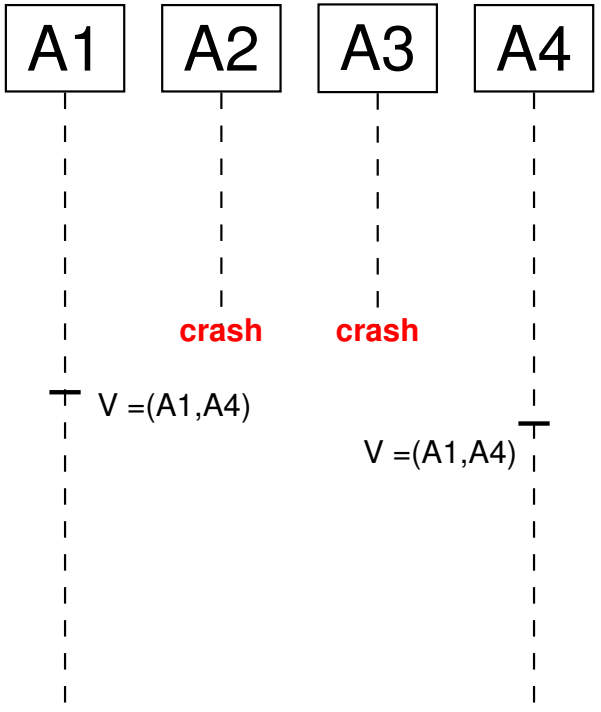
Comment connaitre les acteurs qui participent à une interaction (ou un calcul) ?

- En cas de pannes
- En cas d'arrivée ou de départ d'acteurs

Comment s'assurer que tous les participants ont la même vue des participants ? La vue est coordonnée.



Pas de coordination



Coordination

- Les acteurs sont informés des crashes, des entrées et sorties ; on dit que les acteurs installent des vues
- On fait l'hypothèse que les informations sont précises (pas de perte)
- Les acteurs installent tous la même séquence de vues

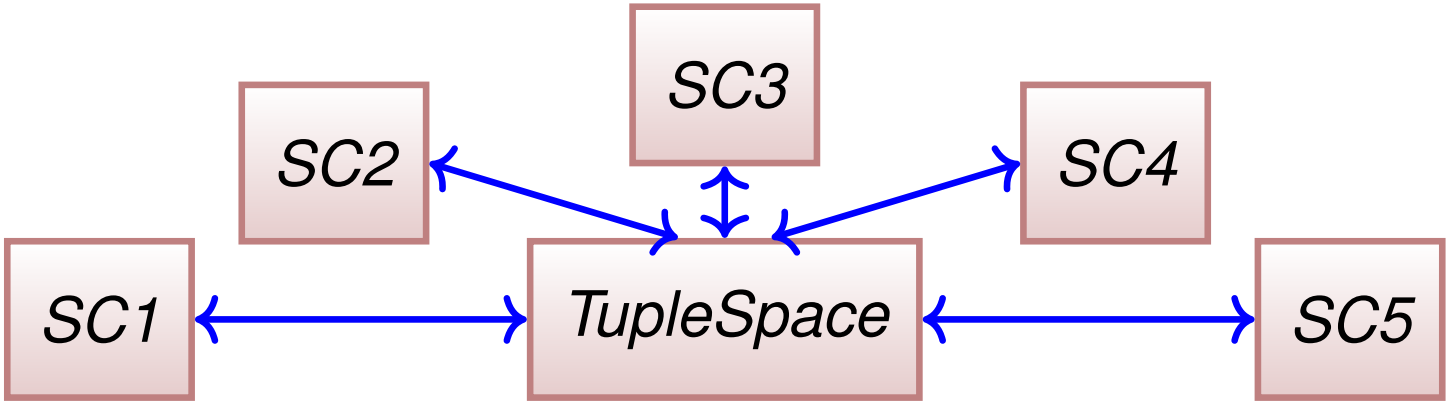
Si on ne prend en compte que les crashes (ni entrées, ni sorties) :

- (Monotone localement) Si un acteur installe une vue (j, M) après avoir installé (k, N), alors $j > k$ et $M \subsetneq N$
- (Agrément) Aucune paire d'acteurs n'installent des vues (j, M) et (j, M') tels que $M \neq M'$
- (Complétude) Si un acteur a crashe, alors il existe un entier j tel que tout acteur correct finit par installer une vue (j, M) tel que $a \notin M$
- (Précision) Si un acteur a installé une vue (i, M) et que $a \notin M$ alors a a crashé

Equité face à l'information : par ex. à la bourse
Gestion des erreurs : locale, globale, dépend des propriétés connues.

- Langage de coordination (à la Linda)
- Langage de conversation (ex. RCA)

Modèle original : Linda [ea94]



Le Linda original définissait 4 opérations :

- **in** lit et supprime (consomme) de manière atomique un tuple
- **rd** lit, sans le détruire, un tuple
- **out** ajoute un tuple (duplication possible)
- **eval** crée un nouveau processus

Un tuple est une description de voyage :

- (destination, date, durée, propriétés).

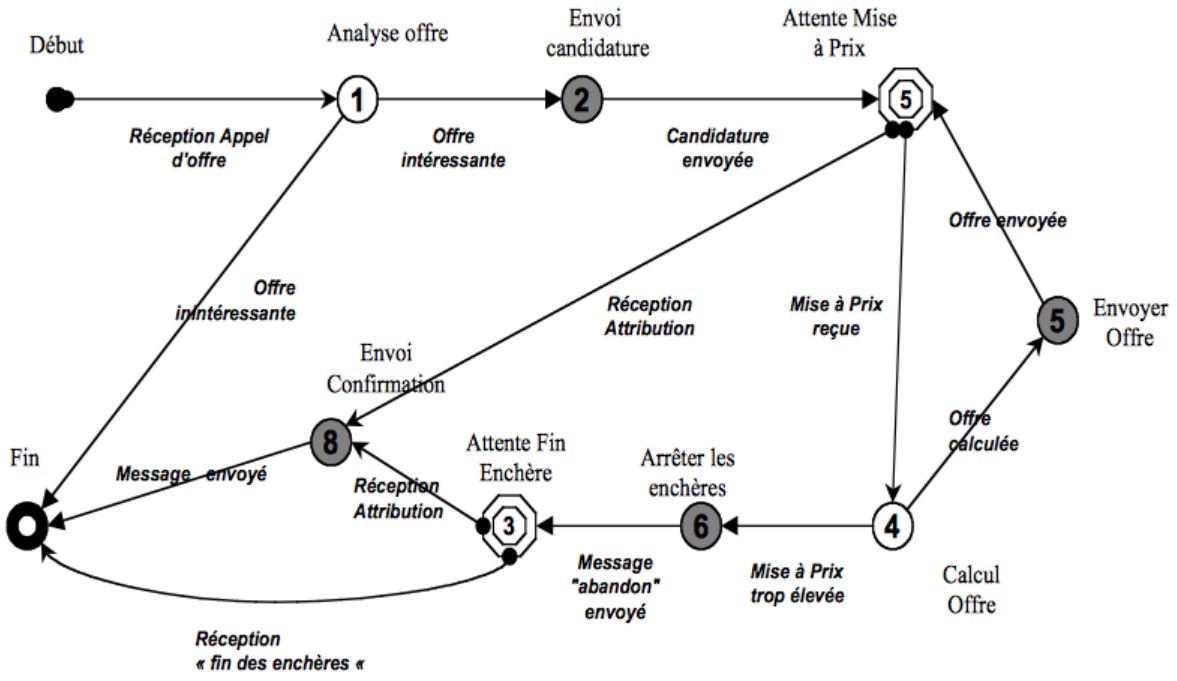
Des processus (agence de voyage) produisent des offres (out).

Des processus (client, agence de voyage) les consultent (rd) ou les réservent (in).

- Mécanisme simple et abstrait.
- Découplage entre les processus ; ils n'ont pas besoin de se connaitre.
- Le protocole est codé dans la forme du tuple.

Implémentations : CppLinda, Erlinda, JavaSpace, PyLinda, etc.

État							Transition	
initial	final	action élémen-taire	action compo-site	attente illimitée	attente limitée	communi-cation	interne	externe
●	◉	○	□	⯃	▽	◉	→	●→



Points positifs

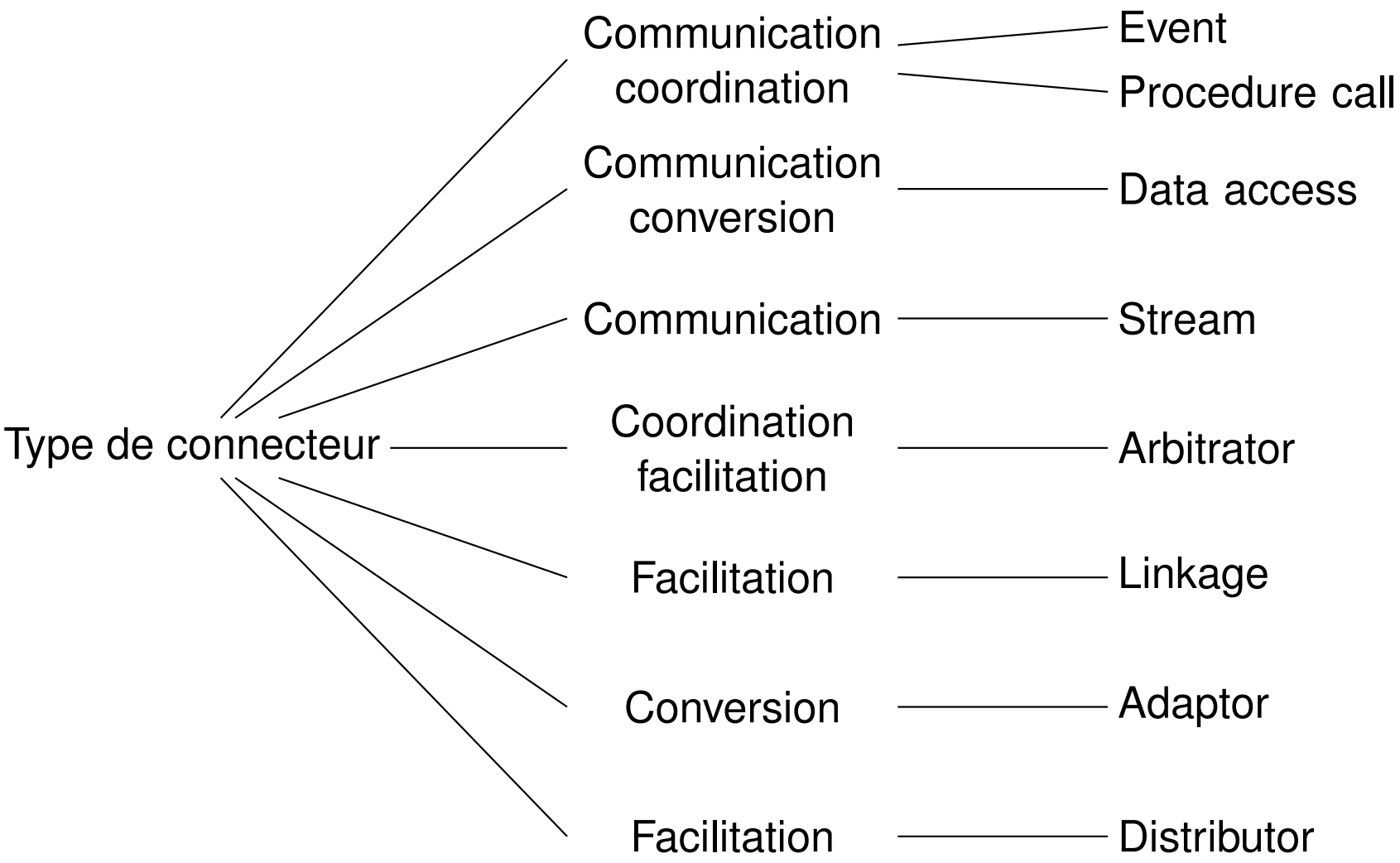
- Vision globale
- Vision temporelle (automate)

Points négatifs

- Pas de rôles
- Pas de dynamique (nombres d'acteurs)

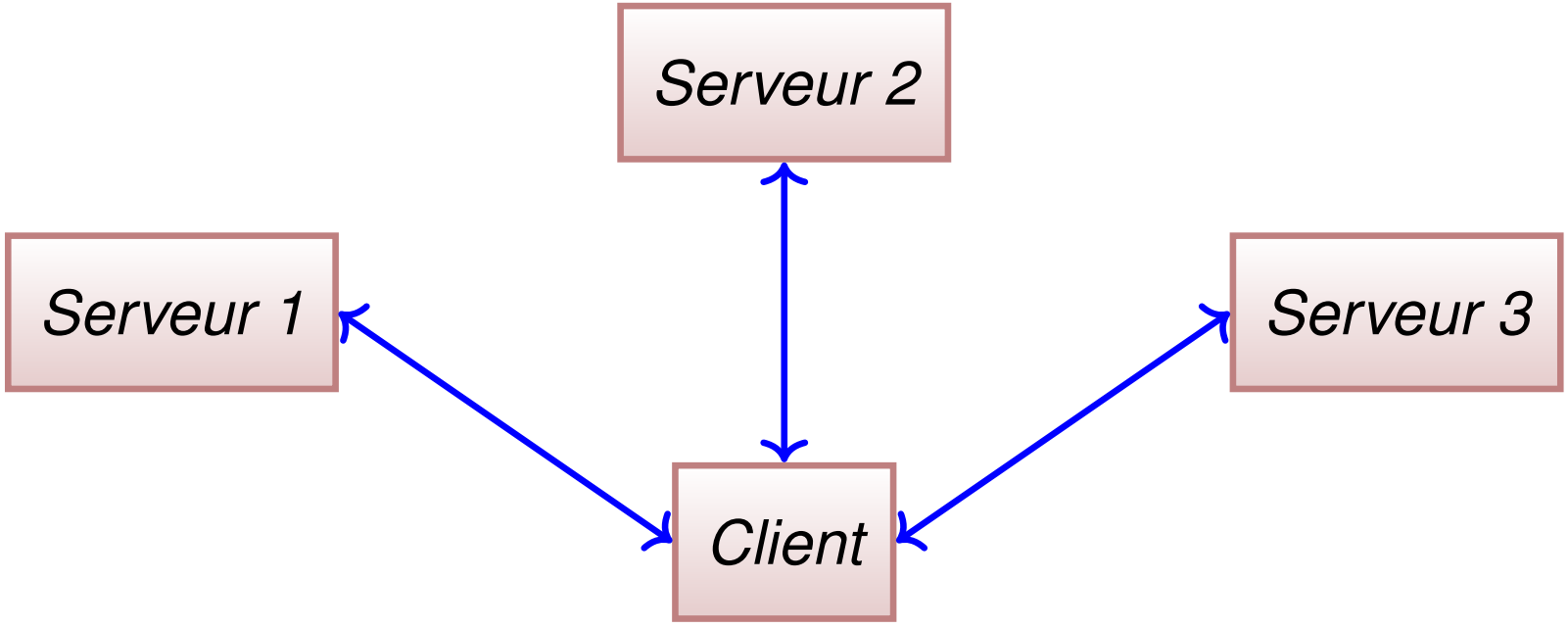
Connecteurs

- Nombre d'acteurs, de rôles (*unicast, multicast, broadcast*)
- Sens
- Initiateur (*push/pull*)
- Synchrone/asynchrone (bloquant)
- Flux/unique
- Politique (exact, *best effort*, ACID, etc.)
- Sûr, crypté (rejoint la politique)
- Taille, rythme, gigue (*jitter*), bande-passante

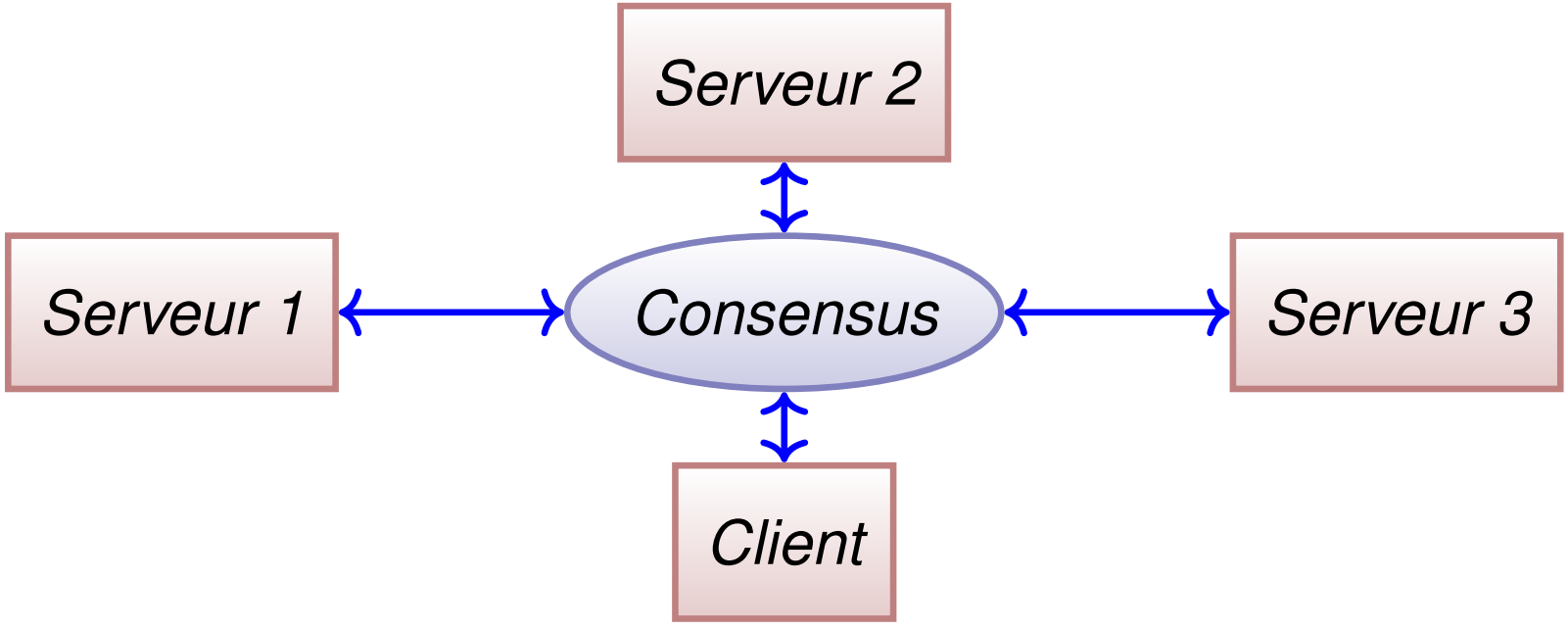

Type de connecteur
Communication coordination
Event
Procedure call
Communication conversion
Data access
Communication
Stream
Coordination facilitation
Arbitrator
Facilitation
Linkage
Conversion
Adaptor
Facilitation
Distributor

- Mémoire (registre, table, pile, etc.)
- Protocole/langage
- Transaction

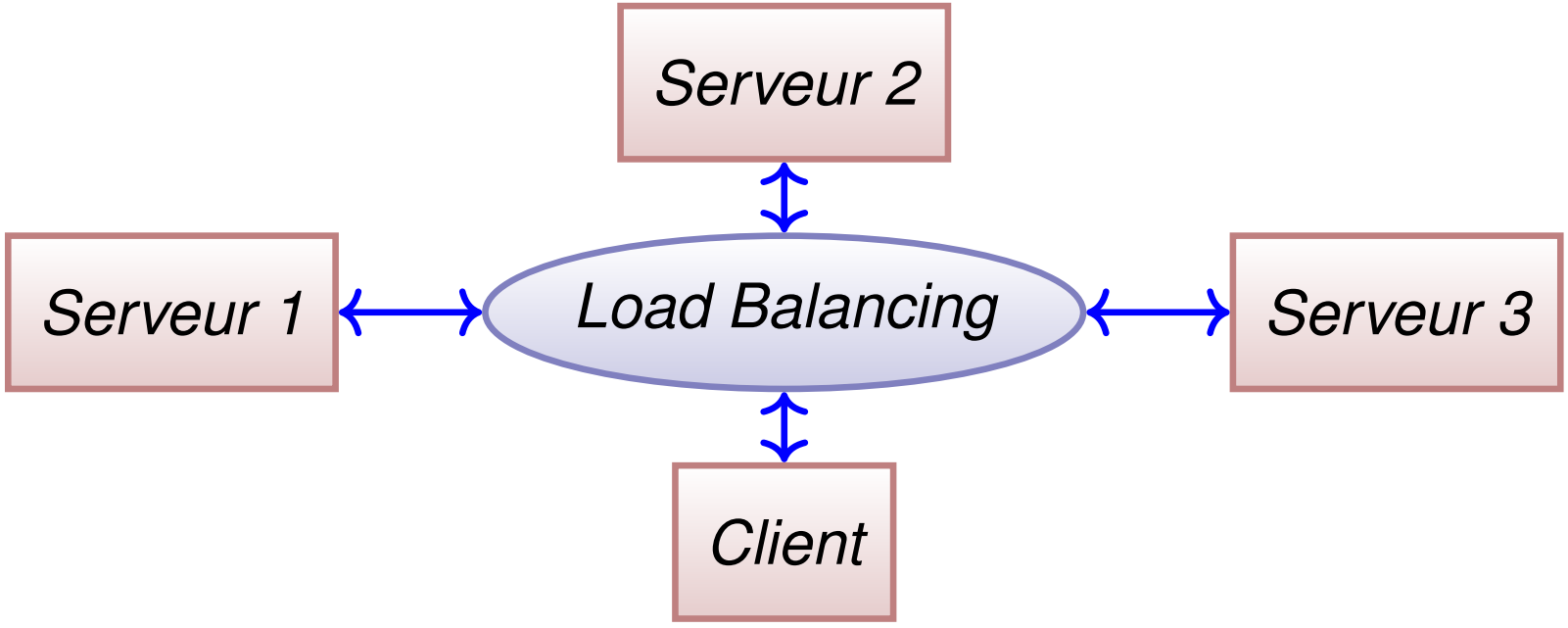
Ces moyens sont interdépendants (protocoles et transactions utilisent de la mémoire) ; se sont les règles et politiques d'usage qui les différencient.



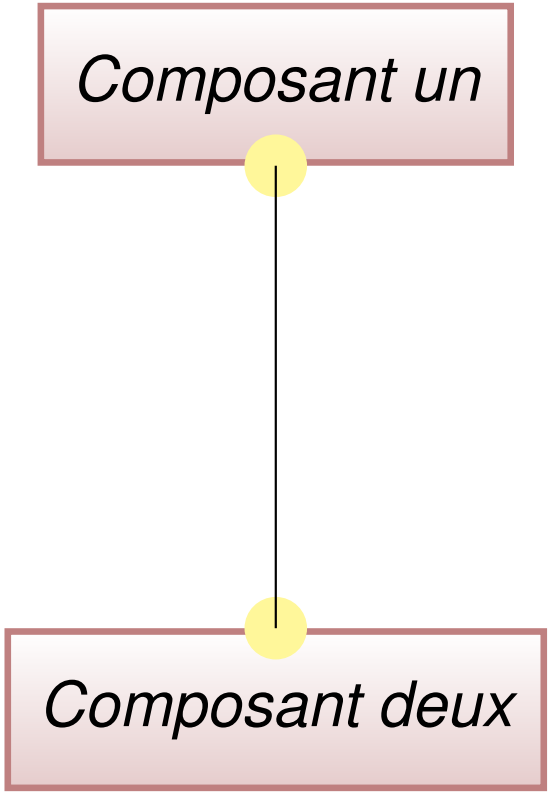
S’agit-il d’un client qui peut choisir entre 3 serveurs, un système d’équilibrage de charge ou un système redondant avec consensus ?

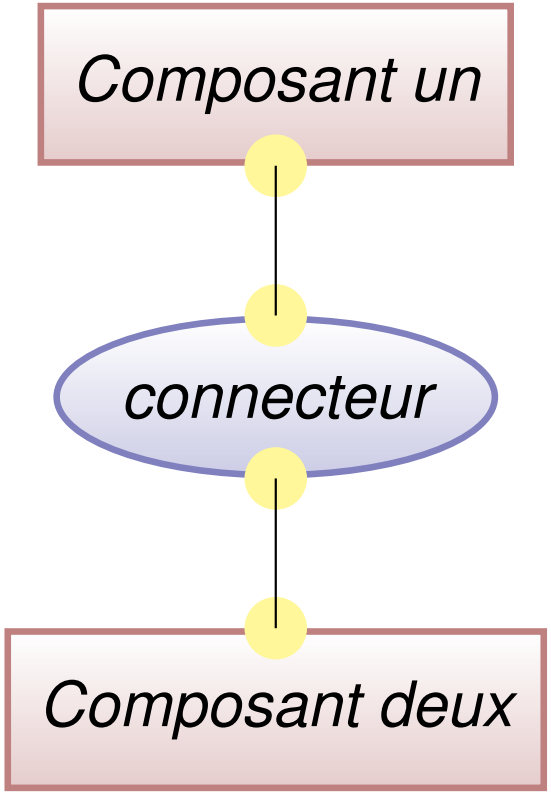


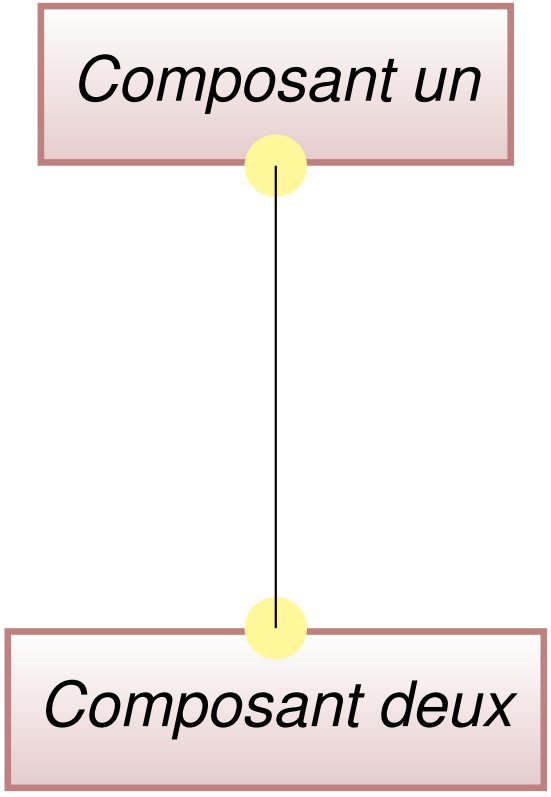
S’agit-il d’un client qui peut choisir entre 3 serveurs, un système d’équilibrage de charge ou un système redondant avec consensus ? Plus d’ambiguïté.

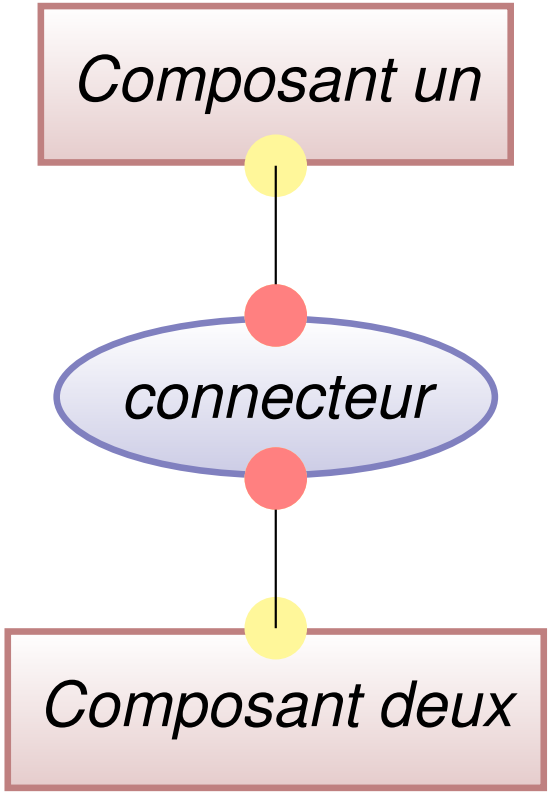


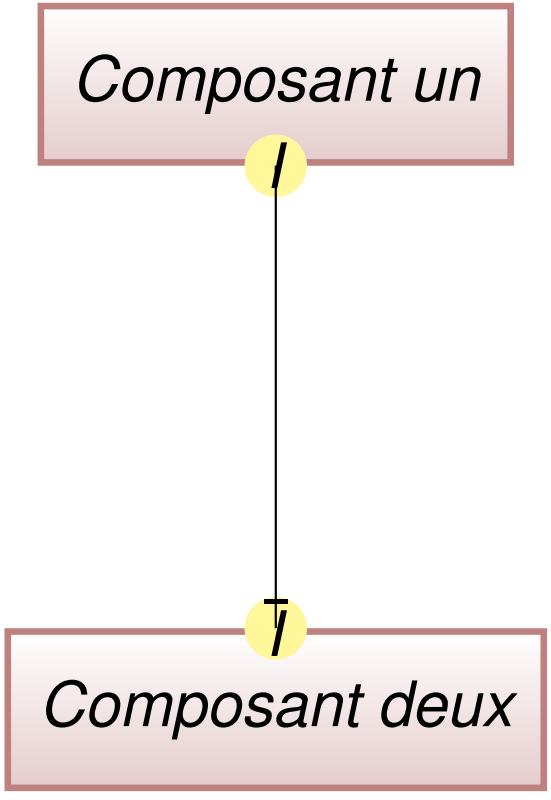
S’agit-il d’un client qui peut choisir entre 3 serveurs, un système d’équilibrage de charge ou un système redondant avec consensus ? Plus d’ambiguïté.

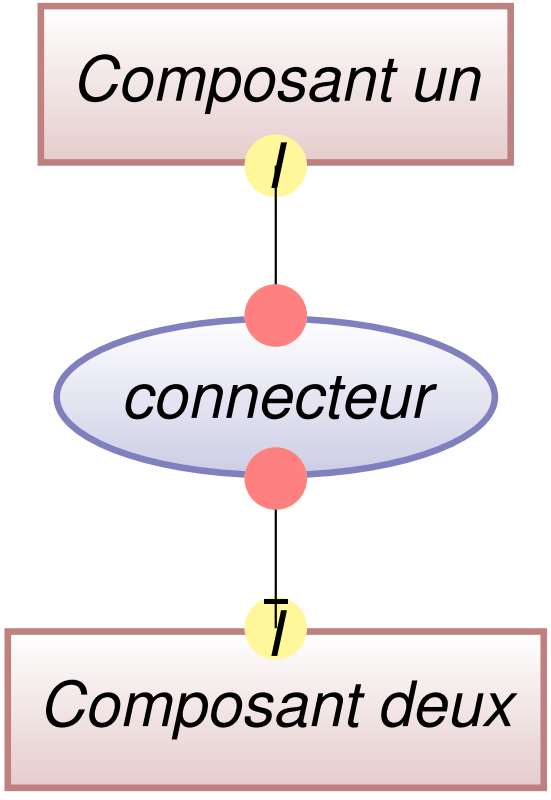

Composant un
Composant deux

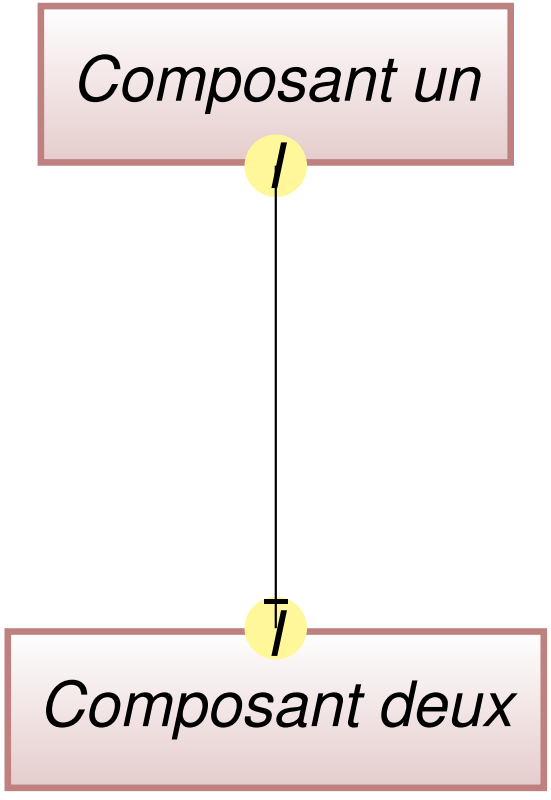

Composant un
connecteur
Composant deux

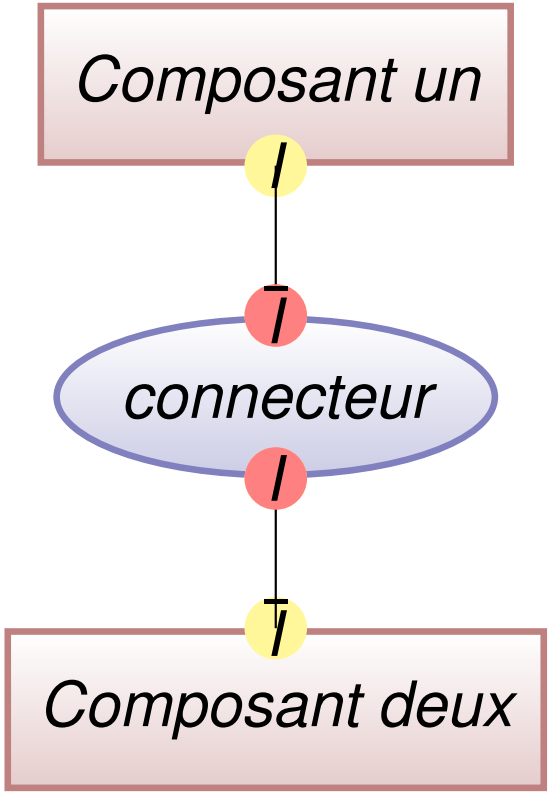

Composant un
Composant deux

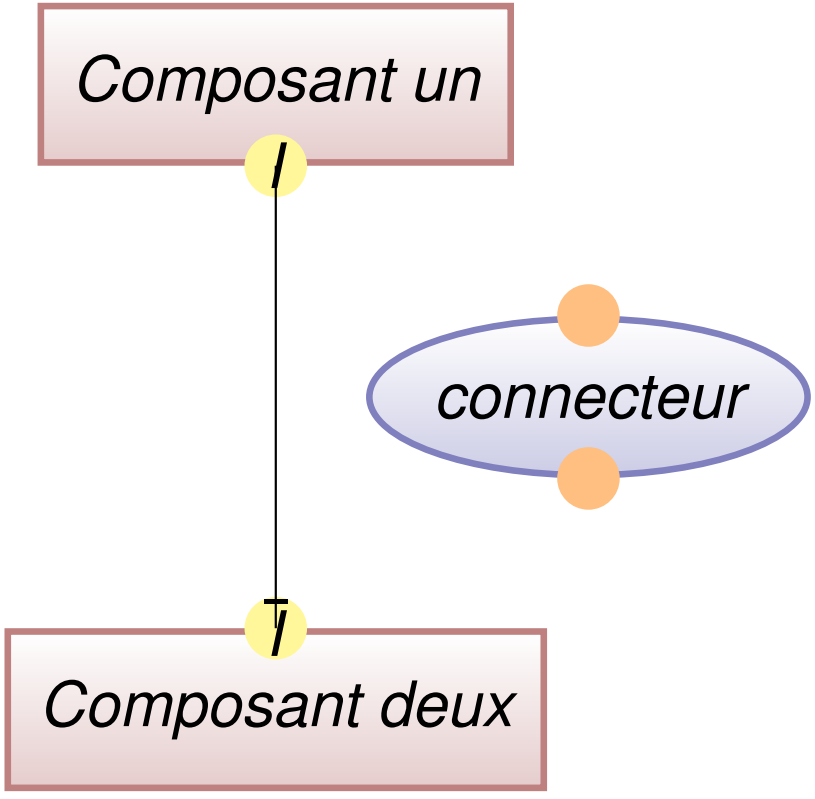

Composant un
connecteur
Composant deux

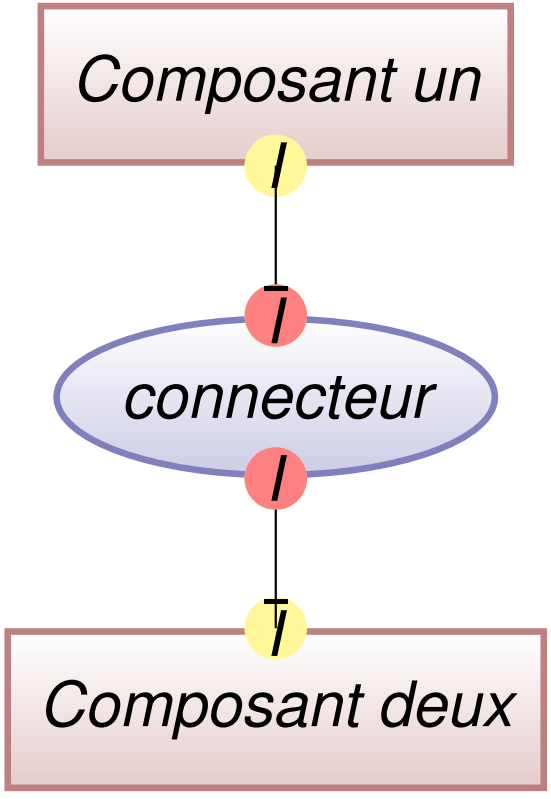

Composant un
I
I
Composant deux


Composant un
I
connecteur
I
Composant deux

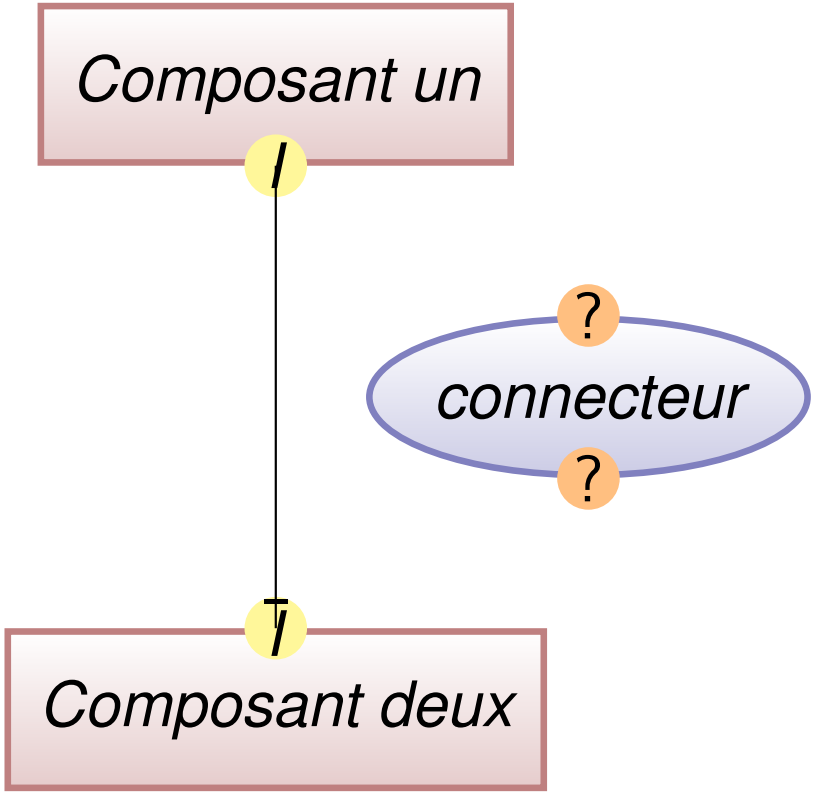

Composant un
Composant deux

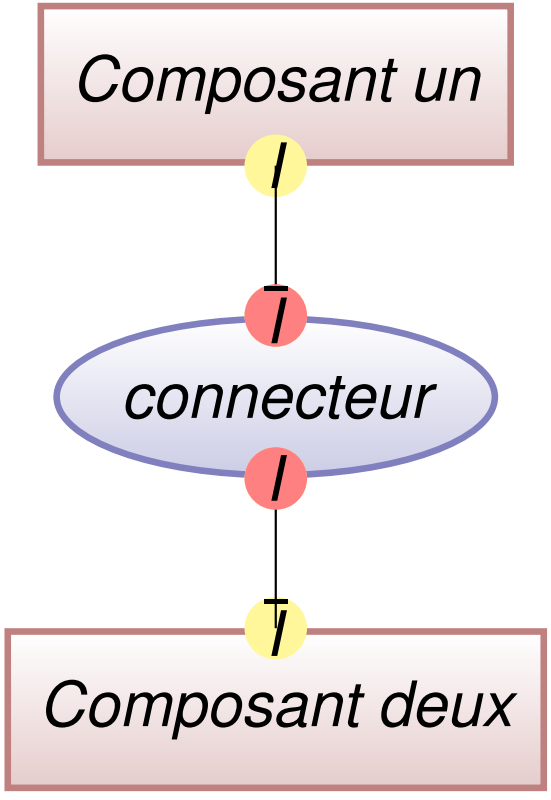

Composant un
connecteur
Composant deux


Composant un
connecteur
Composant deux


Composant un
connecteur
Composant deux

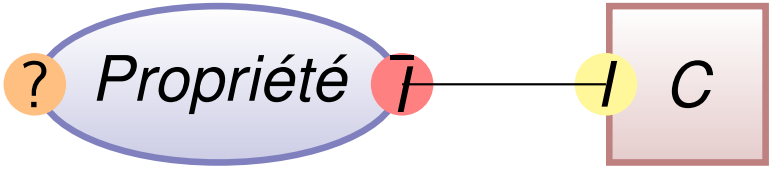
Connecteur ou composant ?





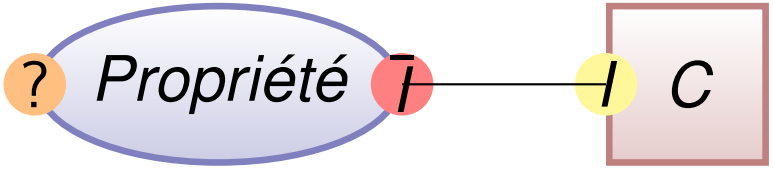
Un connecteur

Un connecteur

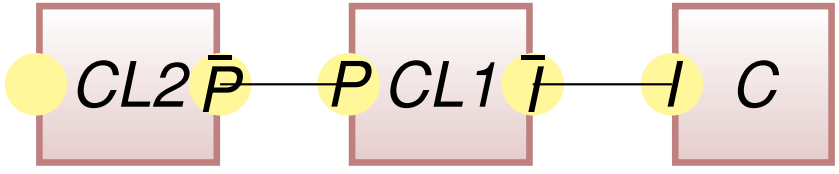


Une connexion

Un connecteur



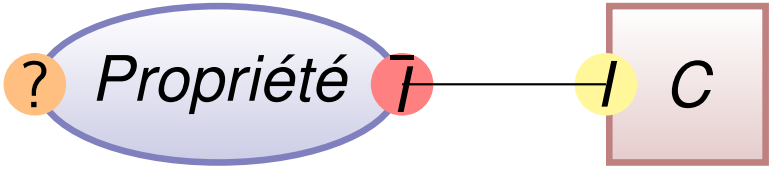
Une connexion



Des composants de liaison

Un connecteur

Et là ?

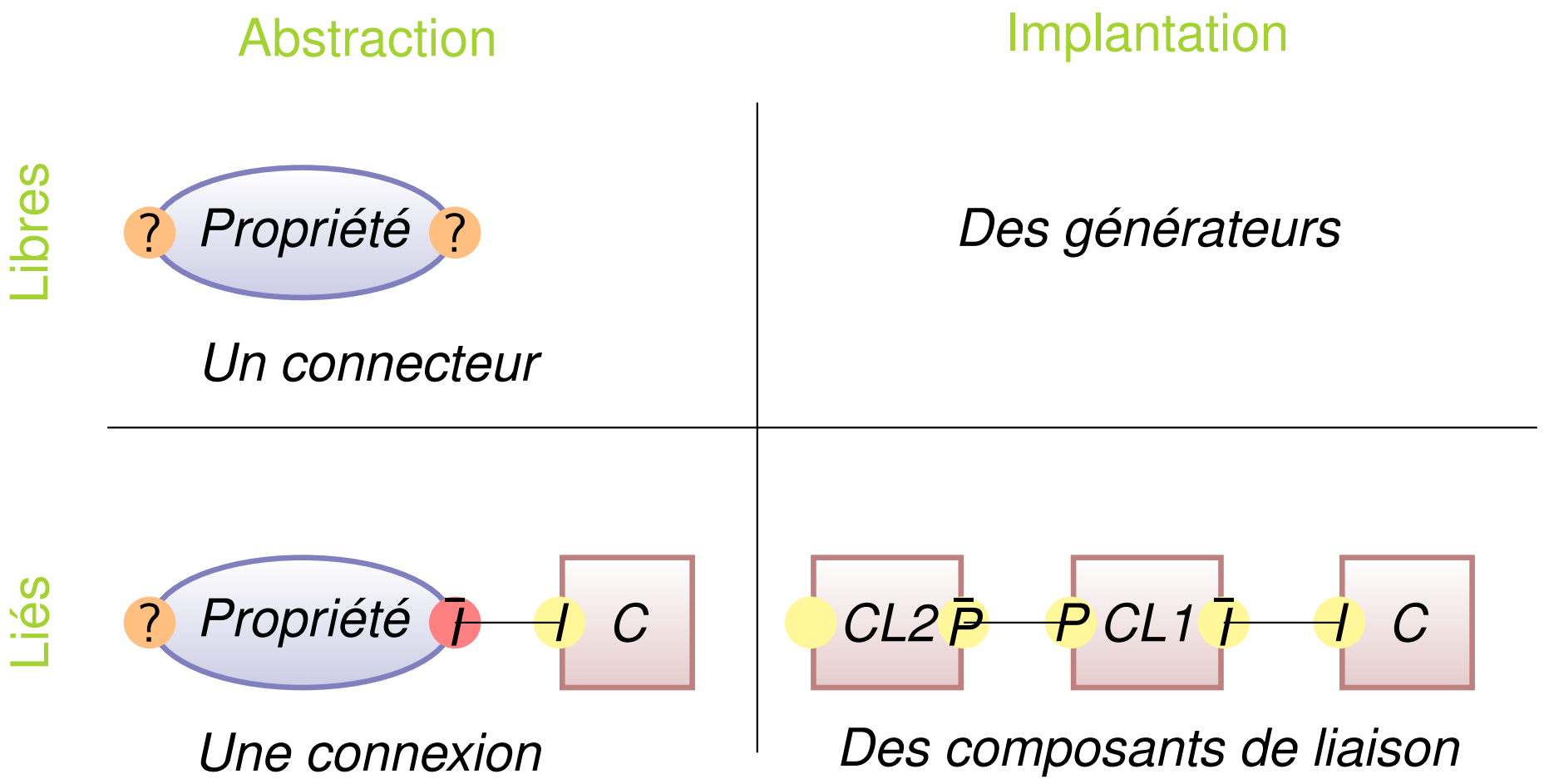


Une connexion

Des composants de liaison

Quelque chose qui *transforme* un rôle, relié à un port – donc une prise – en un composant qui offre l'interface complémentaire du port *et qui assure la propriété* du connecteur.

ex : en Corba, .NET, RPC, Java RMI. . . le générateur de stub
Les stubs, squelettes, talons, sont des composants de liaison.


Abstraction
Implantation
Libres
Liés
? Propriété ?
Un connecteur
Des générateurs
? Propriété Ī I C
Une connexion
CL2 P̄ P CL1 Ī I C
Des composants de liaison

- Appel de procédure (PC)
- Appel de procédure à distance (RPC)
- CORBA, RMI, ...
- Client/serveur avec équilibrage de charge
- Client/serveur avec consensus
- etc

De nombreux connecteurs existent ; parfois indépendant du modèle de composants (ex. les protocoles), parfois liés à un modèle (CORBA RPC).

Les protocoles sont des “connecteurs” livrés comme des composants, i.e. avec une interface explicite (le port = l’API).

Utiliser un composant comme un connecteur oblige à adopter l’interface du protocole pour communiquer.

Un connecteur est livré sous la forme d’un générateur.

References I

D Caromel.

Toward a method of object-oriented concurrent programming.
Communications of the ACM, 1993.

Carriero et al.

The linda alternative to message-passing systems.
Parallel Computing, 2(4) :633—-655, April 1994.

P Eugster, P Felber, R Guerraoui, and A Kermarrec.

The Many Faces of Publish/Subscribe.
ACM Computing Surveys (CSUR), 2003.

N Mehta, Nenad Medvidovic, and S Phadke.

Towards a taxonomy of software connectors.
Proceedings of the 22nd international conference on Software Engineering (ICSE), pages 178–187, 2000.

OW2.

Proactive web site.
`http://proactive.activeeon.com//`, 2017.

Erwan Tranvouez and Bernard Espinasse.

Protocoles de coopération pour le réordonnancement d'atelier.
In *JFIADSMA*, 1999.