M1 Informatique · Théorie et pratique de la concurrence

CM1 Introduction

Lundi 13·09·2021

Equipe enseignante

M· Ahmed Bouajjani , Email: abou@irif·fr

Dans cette UE

On va faire essentiellement de la mémoire partager, et à la fin, s'il reste du temp : systèmes distribués·

- D'abord on va faire des abstractions en regardent des modelés·
- Comment ça se présente quand on veut écrire un programme·

Notation

- Partiel a mis parcours (fin octobre - mi-novembre)
- Examen finale·

Programmation séquentielle

L'exécution se fait de manière déterministe en fonction des entrées

Jusqu'à maintenant vous avez vu des programmes séquentiels : les instructions s'exécutent une après l'autre, et on obtient un résultat, l'exécution est unique en fonction des données·

Programmes parallèles et concurrent

On va voir dans ce cours des programmes qui sont parallèles et concurrent (Concurrent : forcément parallèle ; Parallèle : pas forcément concurrent).

- **Parallèle** : plusieurs processus qui s'exécutent en même temp.
- **Concurrent** : il y a une sorte de compétition, il y a des conflits, de la concurrence sur un certain nombre de choses. Comme on peut partager des ressources, il y a forcément des conflits a un certain moment, donc il faut des mécanismes pour régler ses conflits.

Pourquoi on a besoin du parallélisme ?

Le parallélisme apparait du niveau du hardware jusqu'au logiciel du plus haut niveau pour augmenter **la performance**.

- **Matériel** : les fabricants des matériels on tjr mis l'accent sur la rapidité, mais au bout d'un moment on a arrivé a la plus grande rapidité possible, et c'est pourquoi on commence à avoir des machines avec plusieurs processeurs (multi - cœur)
- **Systèmes d'exploitation** : faut paralléliser beaucoup de choses : les terminaux, le gestionnaire de fichiers, etc. Ces différents composants von s'exécuter en parallèle, donc il y a un ordonnanceur qui va switcher entre les taches sans laisser une tache attendre trop de temp, et cela, afin, qu'on aura l'impression que tous va rapidement (illusion du parallélisme). C'est donc à la charge de l'ordonnanceur de décider qui s'exécute à quel moment. Mtn, avec les machines avec plusieurs processeurs, on a plus de ressources, mais il y a tjr bcp de taches.

- **Processus parallèles de manière naturel** : par exemple, le gestionnaire de fichiers : il y a quelqu'un qui efrit le fichier et il y a quelqu'un qui peut le lire au même moment.
- **Les drivers** (les pilots pour les périphériques)
- **Langage de programmation** : au niveau des langages de programmation on nous offre des structures de donnée de haut niveau (Pile, File, etc.) qu'on utilise à partir d'une certaine interface. Pour que ça soit efficace les personnes qui crée ces libraires, vont essayer de paralléliser, mais cela est cacher dans les librairies, nous on utilise que des primitives de haut niveau.

Atomicité (= indivisible)

Si je dois implémenter, par exemple, une pile (avec une liste chainée etc.) et on veut paralléliser -> plusieurs utilisateurs von vouloir faire des actions sur la pile en même temp. Donc. La fonction empiler (par exemple) doit être atomique.

Pour cela, on va utiliser des verrous, c.-à-d. : je prends le droit d'accéder a cette structure et je ne lâche pas tant que j'ai pas fini mon opération. Par conséquent, les autres utilisateurs ne peuvent pas y toucher.

L'inconvénient : tous cela est très lent, ça block la structure aux autres. C'est pour cela que les bibliothèques (qui fournisse des piles par exemple) ne sont pas implémentées de cette façon. On veut laisser a tlm d'accéder à la structure, tant que la consistance existe tjr (-> algo plus sophistiquer -> primitives de bas niveau -> plus compliqué).

Les systèmes distribués

- Parallélisme
- Concurrence
- Pas de base de donnes partager (les donnes sont distribuées partout dans le monde) -> la base de donnes est implémentée comme un réseau.
- On n'a pas de consistance forte (forte = tt le monde la vue).

Synchronisation

Quand il y a de la concurrence, il y a une interaction, et donc ce qui est important c'est d'avoir une synchronisation.

- Systèmes asynchrones : souvent dans les applications mobiles, par exemple, il y a bcp de choses qui peuvent se faire de manière complètement asynchrone. Ces taches ne sont pas dépendantes entre elle.
- Cependant, il y a des taches qui nécessite de la synchronisation : lorsque on veut mettre à jour une structure de donnée par exemple.
- Synchronisation : si je rentre dans la section critique, faut que tlm soit au courant.
- Le mécanisme de synchronisation ralenti le system -> donc faut le rendre le plus minime possible.
- Il y a de différentes primitives de synchronisation selon les niveaux (matériels, system d'exploitation etc.)

CM1 • Théorie et pratique de la concurrence

Raisonner de manière abstraite

Dans les systèmes parallèles on aura des variables a affecter, mais on peut aussi avoir des primitives d'envoie et de réception de messages.

Pour l'instant, chaque action va être énumérer A, B, C, D, etc.

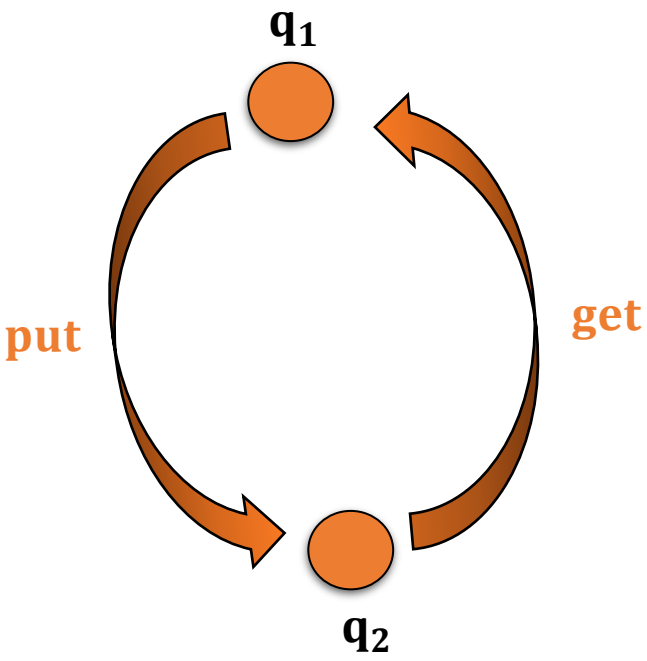
Donc,

On va avoir un system qui est donnée par un ensemble d'états :

$$\delta \leq Q \; x \; A \; x \; Q$$

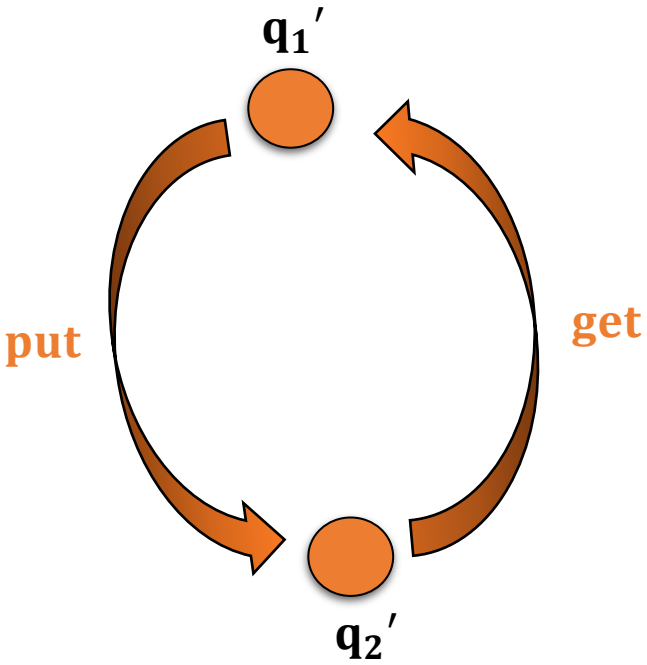
$$\text{Not} : q \xrightarrow{a}_{\delta} q' \quad pour \quad (q, a, q') \in \delta$$

On a un system avec 2 actions : put, get *et les états* q_1, q_2



Un registre / une case mémoire

Maintenant, j'ai un 2em système :

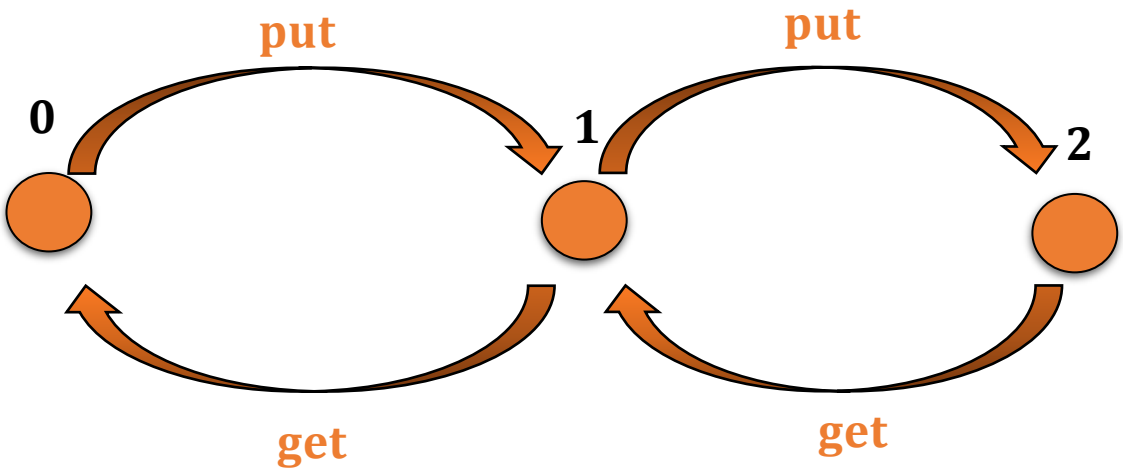


Un registre / une case mémoire

Si ces 2 systèmes sont en parallèle ?

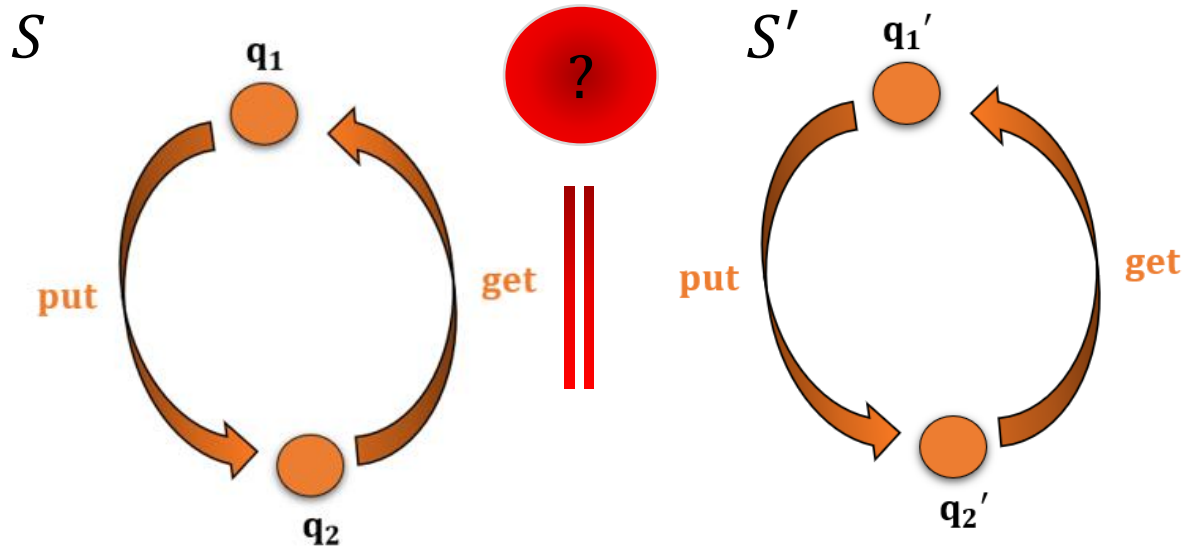
Comment imaginer un registre a 2 places ?

J'aurai une machine comme ça :



Dans cette machine je peux faire 2 **put**, *mais pas plus.*

Quelle est le sens de ces 2 machines en parallèles ?

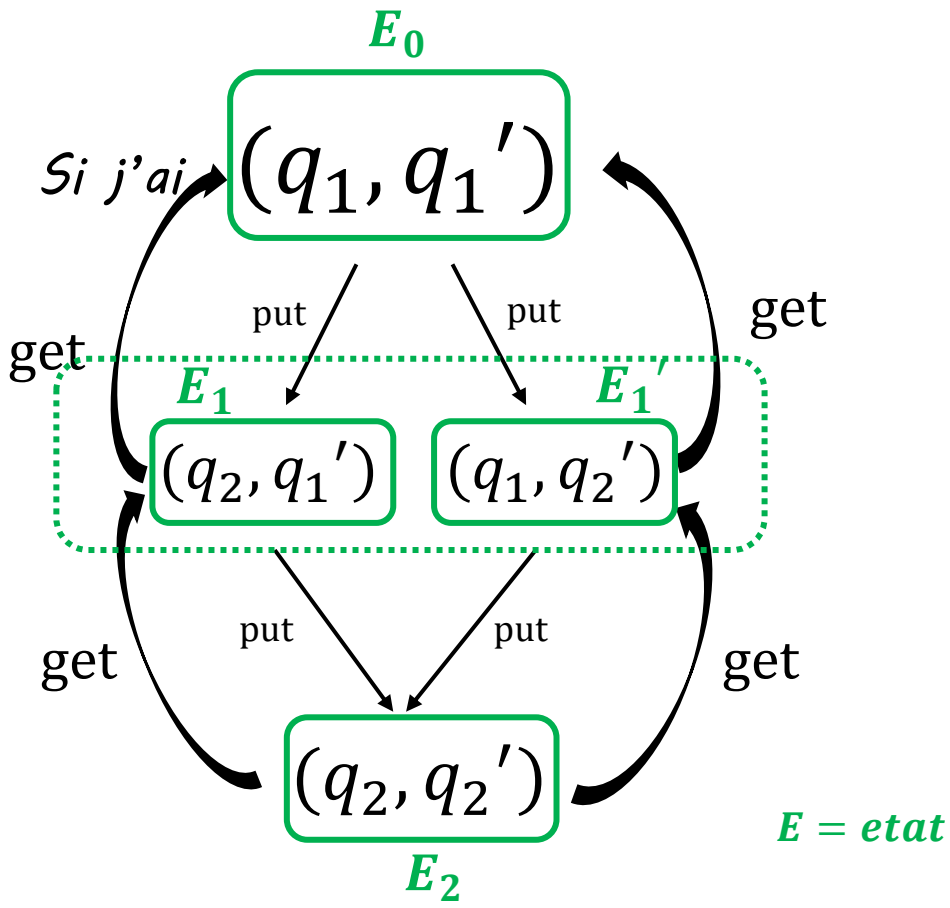


$S = (Q, A, \delta)$ $S' = (Q', A', \delta')$

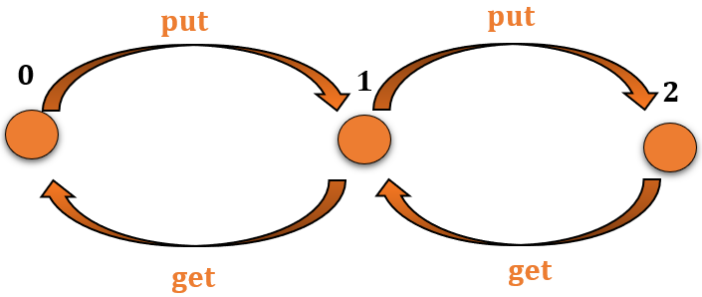
$$S \,||\, S' = (QxQ', A \ldots)$$

Parallélisme

Les 2 machines sont équivalentes, A chaque fois qu'on parle d'abstraction , on doit choisir le niveau d'abstraction · En effet, on a les mêmes actions sur les 2 diagrammes·



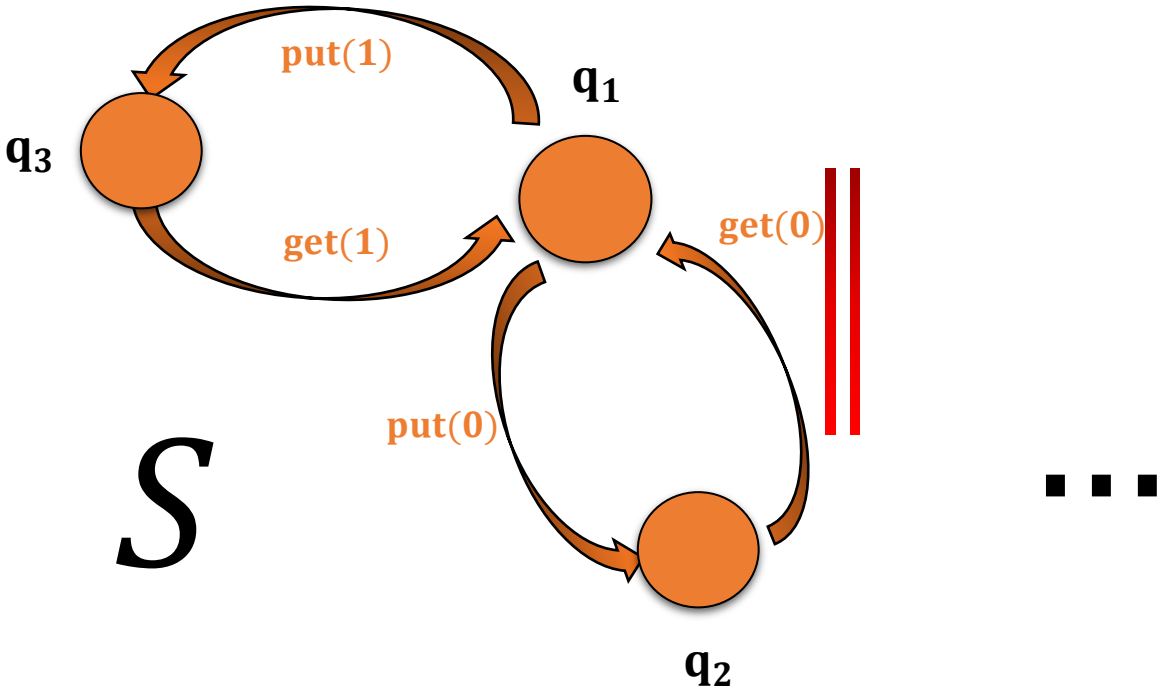
C'est pareil que :



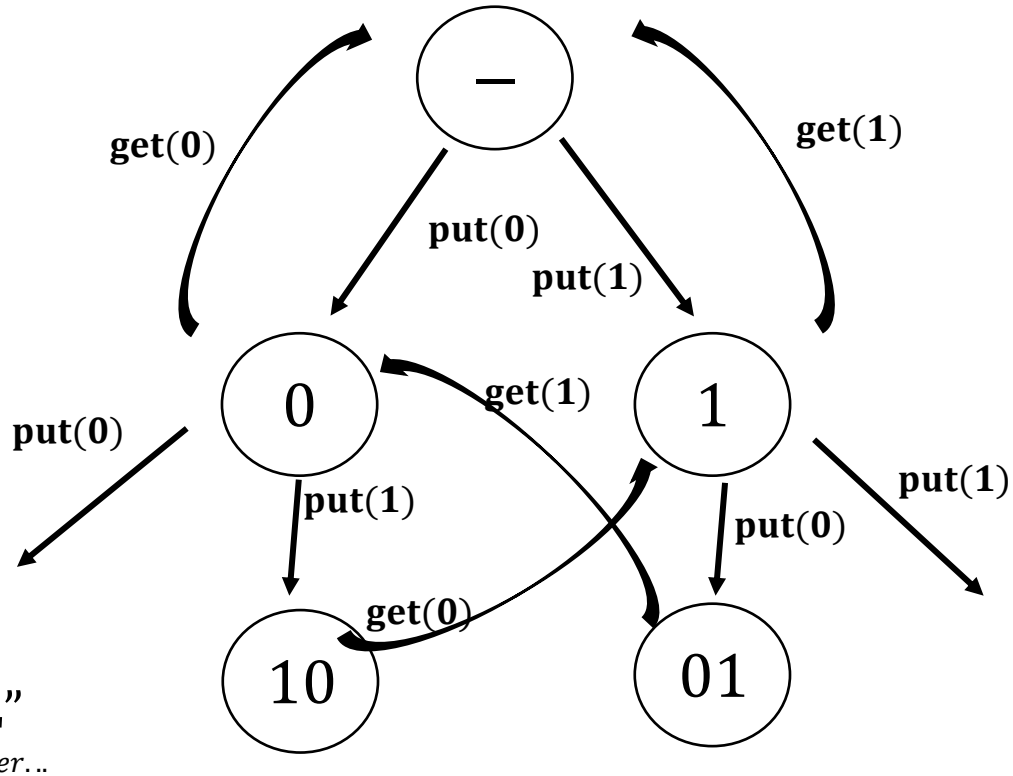
- Dans cette façon de définir la sémantique, on suppose qu'il y a une action d'une machine a la fois· Dans ce modèle on a supposé que les actions à ce niveau soient atomiques·
- Il y a toujours un niveau d'atomicité sur lequel il faut se mettre d'accord·
- L'atomicité de ces actions : une hypothèse importante·
- Donc, on a obtenu un mélange des 2 systèmes de manière complétement asynchrone·

Comment je fais pour implémenter un FIFO ?

Je vais considérer un alphabet avec 0 et 1.



Je peux mettre en parallèle et toujours obtenir le diagramme suivant ?

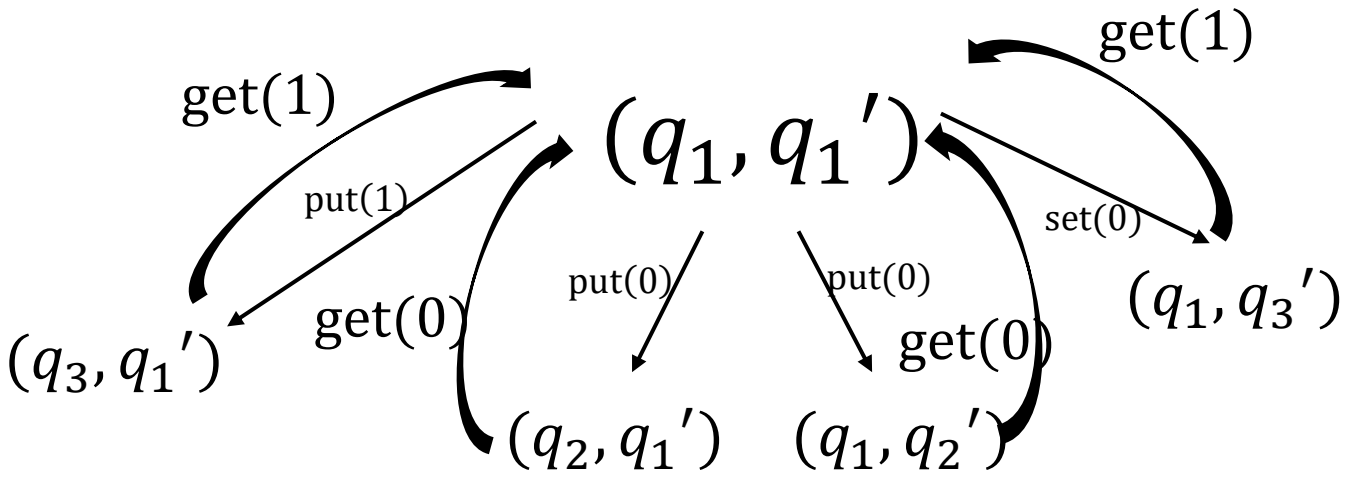


"XYZL"

A compléter…

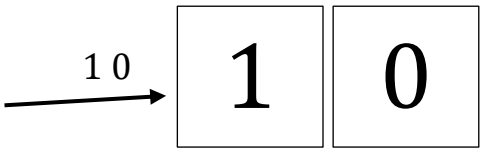
Implémentation

Si on continue cet arbre ça va marcher ? (On a déjà vu le l'arbre dans le cas asynchrone).



Exemple

Dans une machine FIFO, pour une entrée 1,0
la sortie doit être 1,0



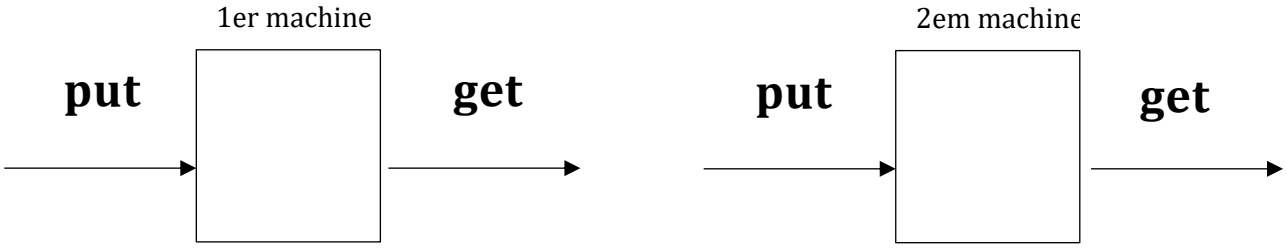
Mais, je peux rentrer et sortir de n'importe quelle machine dans n'importe quel ordre. Par exemple, maintenant je peux sortir d'abord le 1 avec l'action get(1)

Donc, on a 2 machines que si elles sont en parallèles faut faire un mécanisme qui assure le FIFO et n'autorise pas de comportement asynchrone.

Donc, il faut une synchronisation !

Comment réaliser cette synchronisation ?

Si on a :

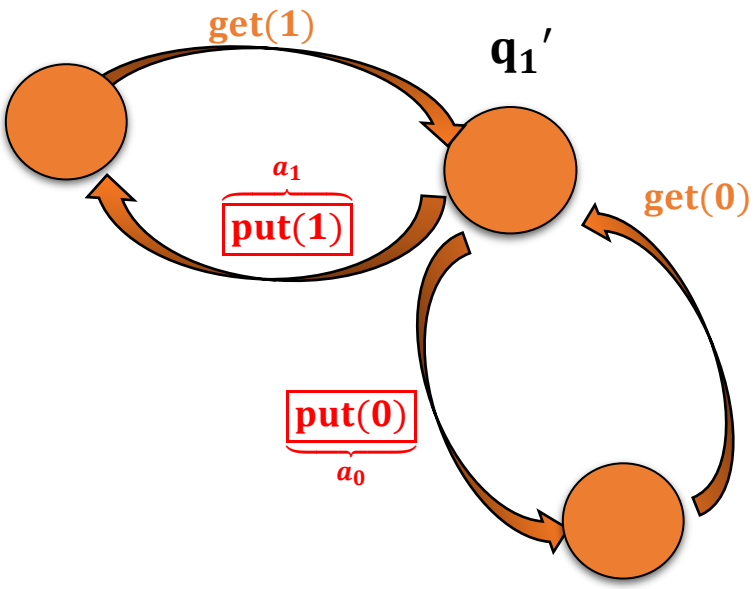


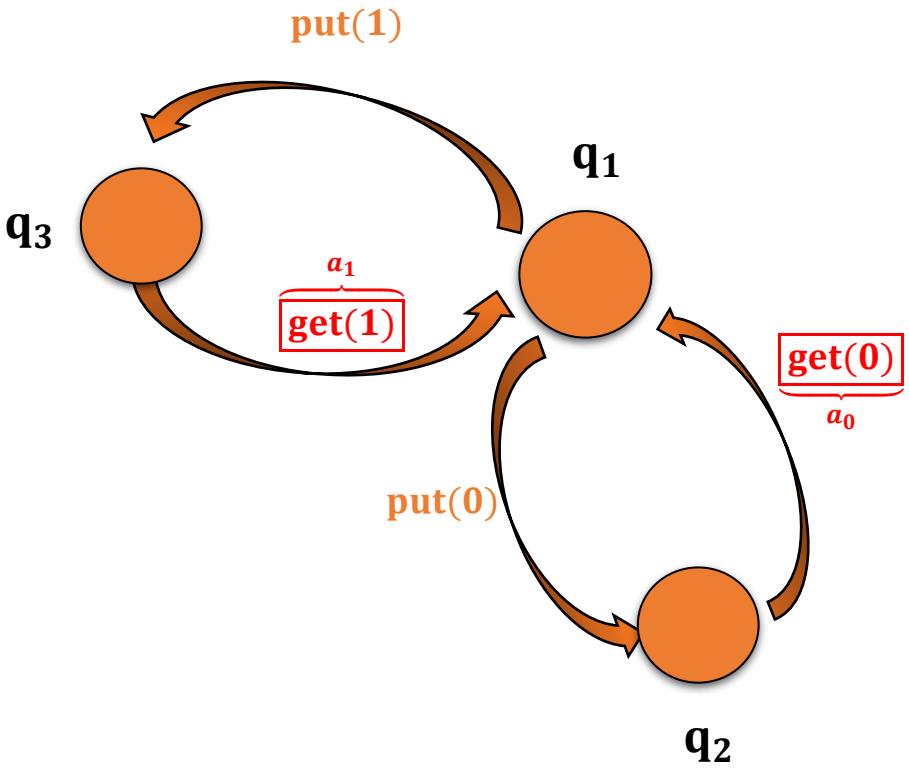
Comment synchroniser ?

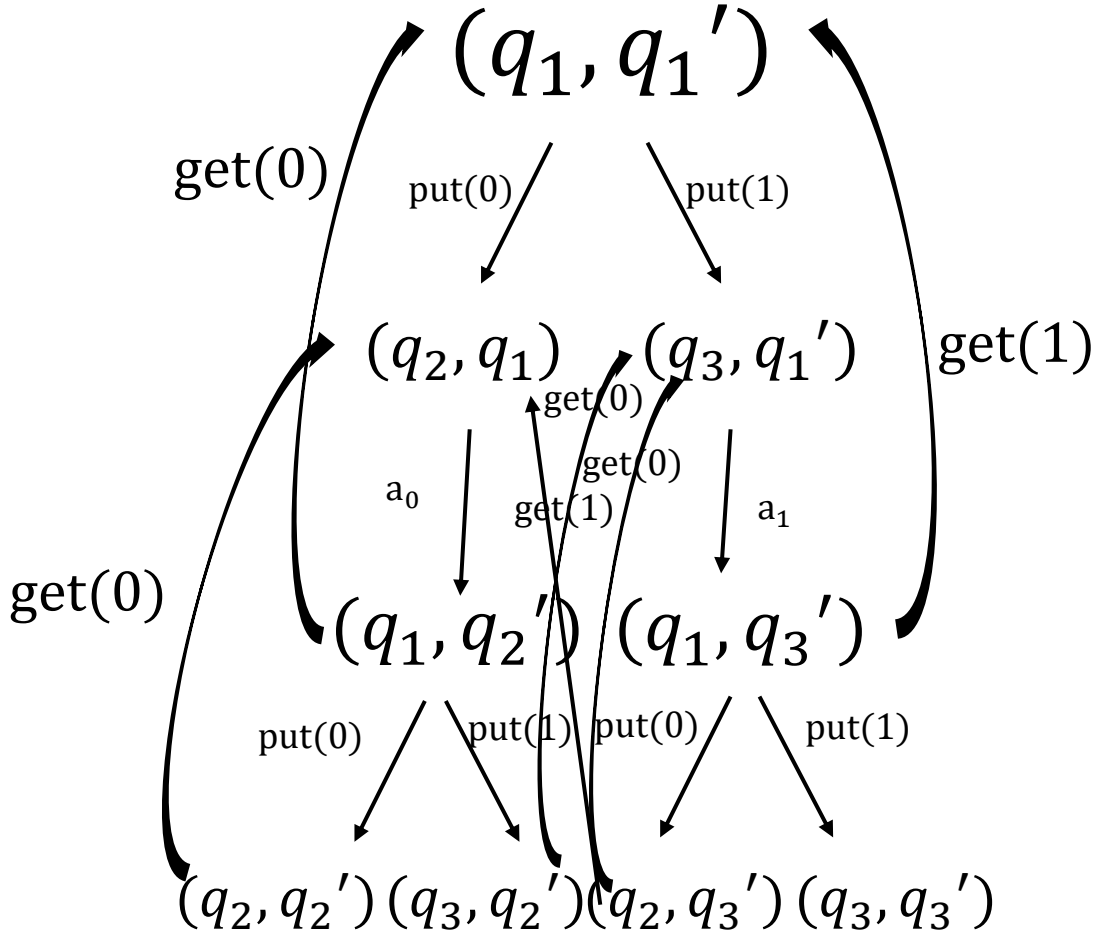
Il faut un tuyau entre les 2 machines : les choses entre par un cotée et sorte par l'autre cotée.

Donc, quand on a 0 dans la 1ère machine, et on fait un 2em put, le 0 va aller à la 2eme machine.

Dans cet exemple le « 1 » ne peux pas avancer tant que la 2eme machine n'as pas été vidé. C.-à-d. le « 1 » ne peux pas avancer a son rythme, **il y a donc de la synchronisation entre les 2 machines, on a plus d'opérations atomiques.**


get(1)
q_1'
a_1
put(1)
get(0)
put(0)
a_0


put(1)
q_1
q_3
a_1
get(1)
get(0)
a_0
put(0)
q_2


(q_1, q_1')
get(0)
put(0)
put(1)
(q_2, q_1)
(q_3, q_1')
get(1)
get(0)
get(0)
a_0
get(1)
a_1
get(0)
(q_1, q_2')
(q_1, q_3')
put(0)
put(1)
put(0)
put(1)
(q_2, q_2')
(q_3, q_2')
(q_2, q_3')
(q_3, q_3')

.