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

Le temps et la concurrence

A. Beugnard

ELU 512 – C1

1er semestre 2019

Objectifs

Un système peut être décrit selon plusieurs points de vue.
En POO, on se concentre sur sa structure ; ce qui le compose (les objets, les relations), ce qui caractérise son état.

Une autre dimension essentielle est son évolution ; ce qui fait évoluer son état dans *le temps*.

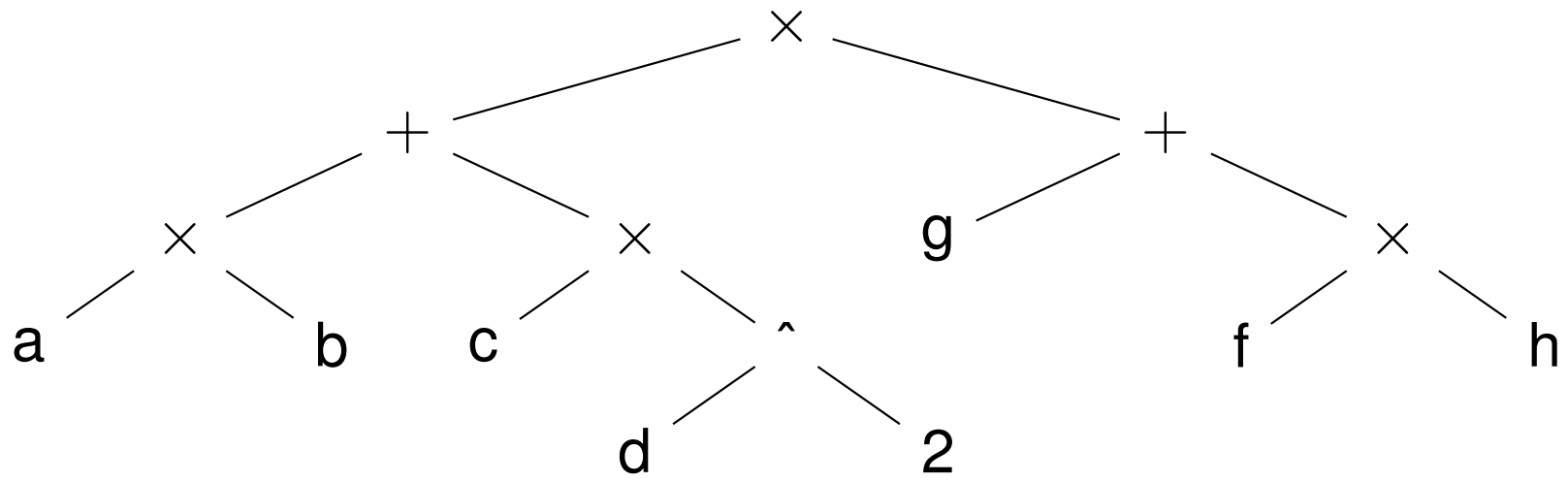
- Définir des concepts
- Donner quelques intuitions
- Proposer quelques notations

N'hésitez pas à prendre des notes. . .

Plusieurs choses se passent « en même temps ».

Des exemples ?

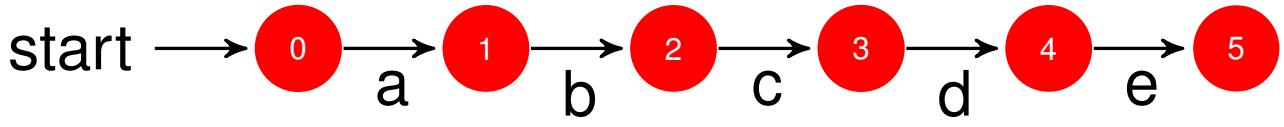
Combien de tic d'horloge pour calculer : $(ab + cd^2)(g + fh)$?



- L’environnement change. . .
  - processeurs multi-cœurs
  - calcul distribué (réseaux d’ordinateurs)
  - *cloud*
- On cherche la performance
- Il y a de la concurrence partout
  - les machines sont partagées avec d’autres

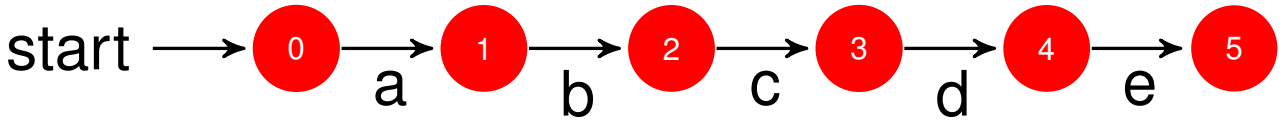
- Il faut coordonner
- Il faut partager
- Il faut communiquer
- Il peut y avoir des *crashes*
- …

Un programme = une séquence d'instructions



Peut-il se passer plusieurs choses simultanément ?

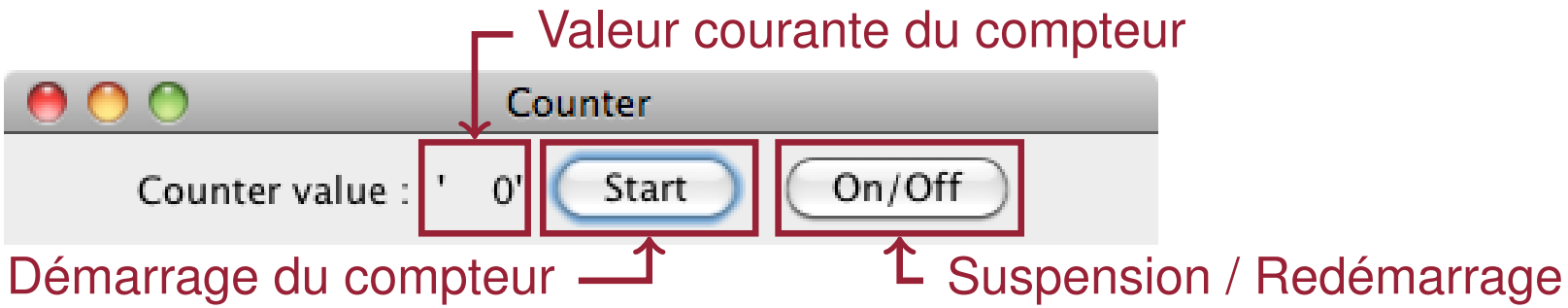
Un programme = une séquence d'instructions



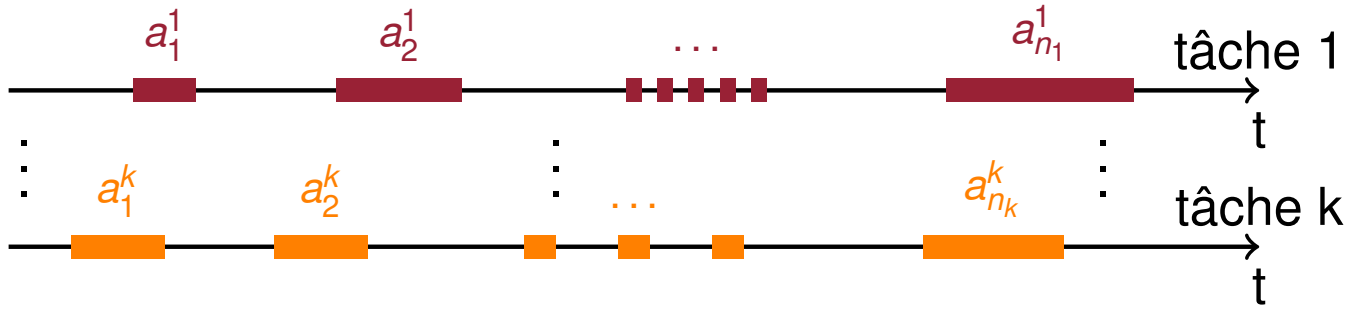
Peut-il se passer plusieurs choses simultanément ? Non !

- L'instruction *b* ne peut commencer que si l'instruction *a* est terminée.
- Le programme se termine quand (et si) l'instruction *e* se termine.

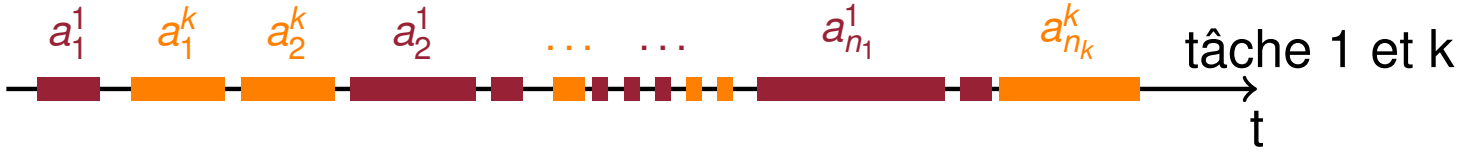
Une interface utilisateur réactive (*Thinking in Java*, `www.BruceEckel.com` 2ed p825)



- Le programme est découpé en tâches
  - les instructions d'une tâche s'exécutent séquentiellement
- Ces tâches sont exécutées indépendamment
  1. si assez de processeurs / cœurs, exécution simultanée



  2. sinon, il y a entrelacement



- Si plusieurs tâches s'exécutent : parallélisme

- Dans quel ordre s'exécutent les tâches ?
  - si les tâches sont indépendantes (c'est rare...)
    - il suffit de respecter l'ordre interne à chaque tâche
  - si des tâches partagent des ressources ou échangent des données
    - il faut qu'elles s'entendent, c'est la synchronisation
    - certains ordres d'exécution ne sont plus acceptables
- On parle de concurrence des tâches
- Comment est assurée la synchronisation ?
  - soit par construction (temps partagé, algorithme, ...)
  - soit le système d'exécution fournit des verrous que les actions doivent prendre et rendre

Un programme concurrent

- exécute simultanément plusieurs tâches (séquences d'instructions) grâce au support de plusieurs fils d'activités (*threads*)

Une spécification d'un programme concurrent

- un ensemble de processus (ou d'activités) qui décrivent des actions séquentielles.
- un ensemble de contraintes (synchronisation, partage de ressources, échange de données)

Une trace d'exécution

- la séquence des instructions *effectivement* exécutée

On considère un système avec une unique imprimante et des dizaines d'ordinateurs
On imagine que pour imprimer un document, un ordinateur envoie page par page

```
1 print(document, printer) {
2   while(document.nextPage()) {
3     printer.printPage();
4   }
5 }
```

```
1 Computer c1, c2, ..., c23
2 Document d1, ..., d23
3 ...
4 Run
5   c1.print(d1,p)
6   || c2.print(d2,p)
7   || ...
8   || c23.print(d23,p)
9 }
```

Une trace (vue par l'imprimante) :
d23(p1),d2(p1),d2(p2),d2(p3),d23(p2),d1(p1), ...

Définition : On dit que deux instructions (actions, calculs) sont parallèles si elles ont lieu au même instant

Qu'est-ce qu'un instant ?

Définition : On dit que deux instructions (actions, calculs) sont parallèles si elles ont lieu au même instant

Qu'est-ce qu'un instant ?
Un point dans le temps. . . qui peut être considéré comme de durée nulle

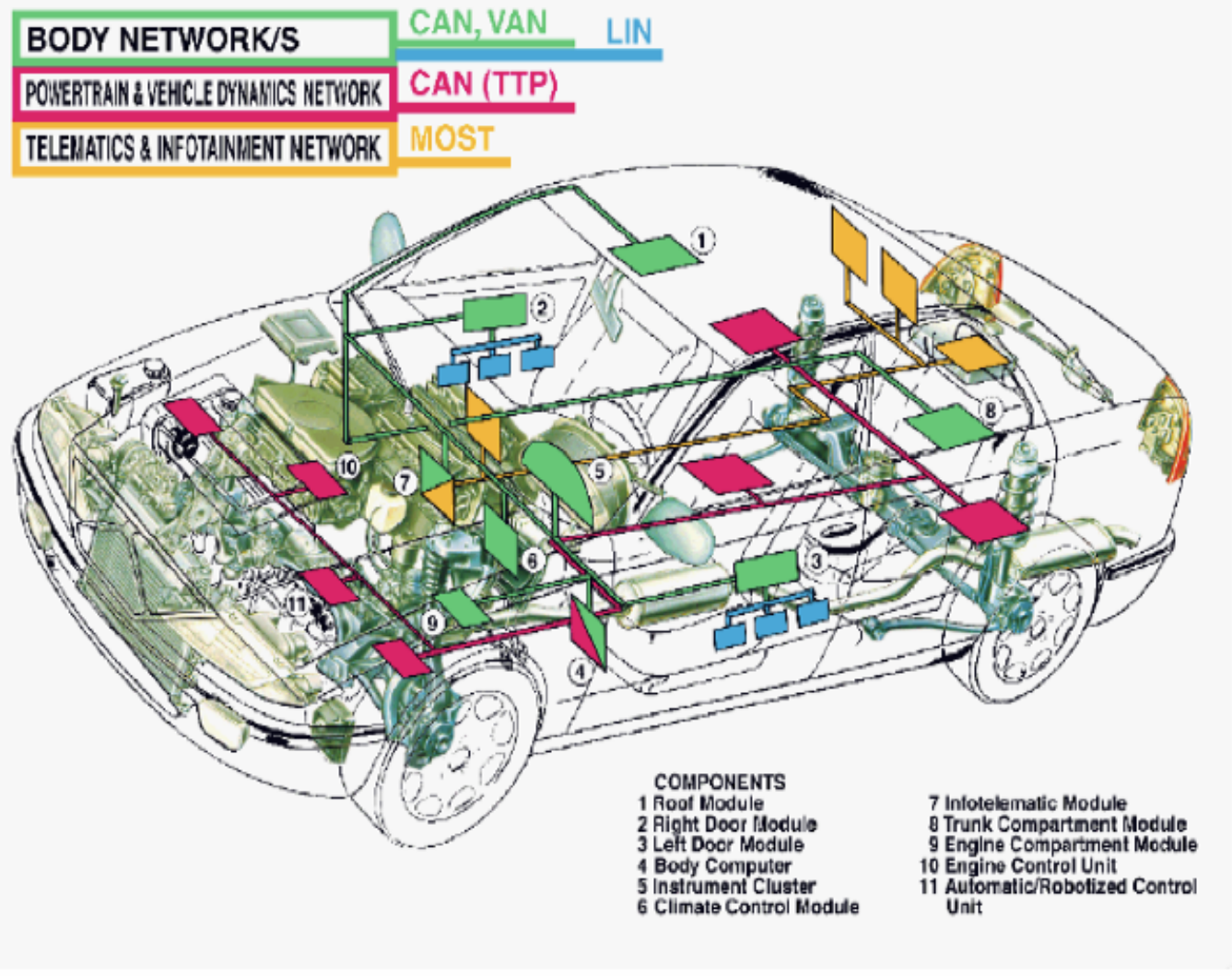
Définition : On dit que deux instructions (actions, calculs) sont parallèles si elles ont lieu au même instant

Qu'est-ce qu'un instant ?
Un point dans le temps. . . qui peut être considéré comme de durée nulle
Abstraitement : pas de relation causale entre les deux instructions (indépendance). Aucun des événements n'est lié à l'autre par une relation *avant* ou *après*

Exemples

- Sur une machine : 2 sessions d'utilisateurs différents
- Avec plusieurs machines : des clients et un serveur

Exemple de système concurrent



- C’est naturel !
- Meilleure utilisation des ressources (partage)
- Meilleures performances
- Plus de souplesse dans les interactions (plus de blocage sur une entrée)
- Meilleur contrôle, en jouant sur les priorités

Donc, pourquoi la concurrence ?

- C’est naturel !
- Meilleure utilisation des ressources (partage)
- Meilleures performances
- Plus de souplesse dans les interactions (plus de blocage sur une entrée)
- Meilleur contrôle, en jouant sur les priorités

Mais c’est aussi beaucoup plus complexe !

- Incident Therac 25
- Mars Rover
- TGV

Les raisons des difficultés :

- Interblocage
- Famine
- Mauvais contrôles
  - partages, atomicité, transactions
  - gestion de priorités
  - synchronisation
- Explosition combinatoire du nombre d'états et des chemins
- …

Partager correctement

Radar

Radar

position de
la fusée
(x,y)
partagée

Console

```
Radar   := while(true) {write(x); write(y);}
Console := while(true) {read(x); read(y); decision();}
```

Une trajectoire est prévue. Si on observe une erreur de trajectoire, on détruit la fusée

Et alors. . . ?

Imaginez l'entrelacement :

```
write(x); read(x); read(y); write(y); decision();
```

La console observe l'ancienne valeur de y !!! La décision peut être erronée.

Imaginez l'entrelacement :

```
write(x); read(x); read(y); write(y); decision();
```

La console observe l'ancienne valeur de y !!! La décision peut être erronée.

Il faut garantir la mise à jour de x et y « ensemble » : on parle d'opération atomique

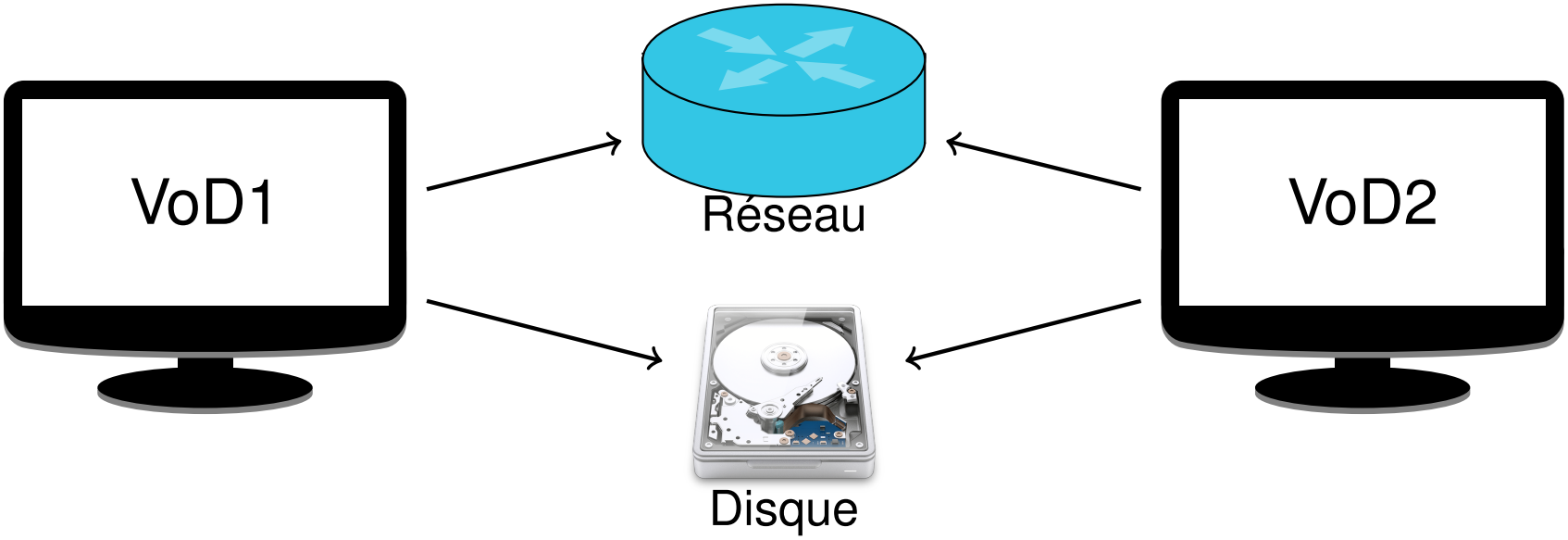
Rendre atomique permet d'assurer une exclusion mutuelle, en interdisant à tout autre processus d'intervenir pendant l'exécution de la zone critique.

Avec la concurrence il faut imaginer que tous les entrelacements sont possibles. Le choix du chemin réellement exécuté est inconnu.

On parle de non déterminisme.

10 processus avec 10 instructions chacun c'est déjà 10^{10} entrelacements possibles !

En conséquence, il est difficile de tester exhaustivement un programme concurrent. De nouveaux risques d'erreurs arrivent.




```
VoD1 := while(true) { acquire(Rx); acquire(Dd); compute();
                      release(Rx); release(Dd);}
VoD2 := while(true) { acquire(Dd); acquire(Rx); compute();
                      release(Rx); release(Dd);}
```

Et alors. . . ?

Imaginez l'entrelacement :

```
VoD1.acquire(Rx); VoD2.acquire(Dd); VoD1.acquire(Dd);
VoD2.acquire(Rx);
```

- VoD1 attend que VoD2 libère le disque
- VoD2 que VoD1 libère le réseau

Plus rien ne se passe, c'est un interblocage

Imaginez l'entrelacement :

```
VoD1.acquire(Rx); VoD2.acquire(Dd); VoD1.acquire(Dd);
VoD2.acquire(Rx);
```

- VoD1 attend que VoD2 libère le disque
- VoD2 que VoD1 libère le réseau

Plus rien ne se passe, c'est un interblocage

Deux solutions à ce type de problème :

- prévention : éviter que cela arrive.
- guérison : tuer un des processus, libérer ses "possessions" (verrous) et le redémarrer

Cette expression peut être calculée de plusieurs façons :

- en séquence (7 pas) : $a \times b$ puis $d \times d$ puis $d \times d \times c$ etc.
- en parallèle (4 pas) : $a \times b$ et d^2 et $f \times h$ puis $g + \times h$ et $c \times d^2$ puis etc.

Qui choisit ?

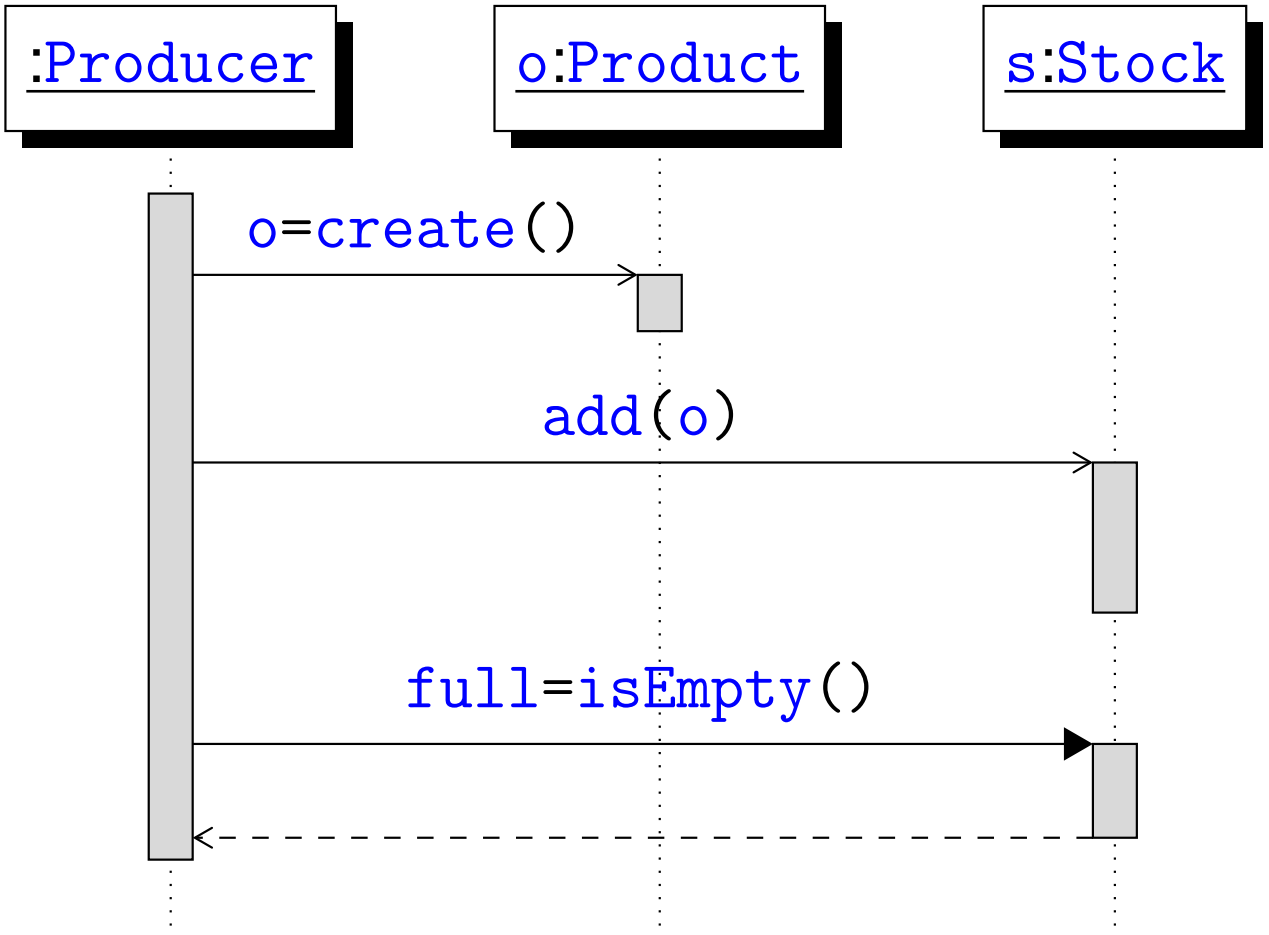
- Le compilateur du langage que vous utilisez. Qui peut *détecter* du parallélisme dans certaines expressions et exploiter les ressources matérielles ou système. C'est le plus abstrait, le plus transparent.
- Le compilateur du langage que vous utilisez. Qui peut utiliser des mots clés comme Task, PAR, synchronize, etc. Vous devez expliciter et traduire le parallélisme vers les concepts du langage. Encore assez abstrait, moins transparent.
- Vous explicitez tout en utilisant les mécanismes de « bas » niveau fournis par votre système.

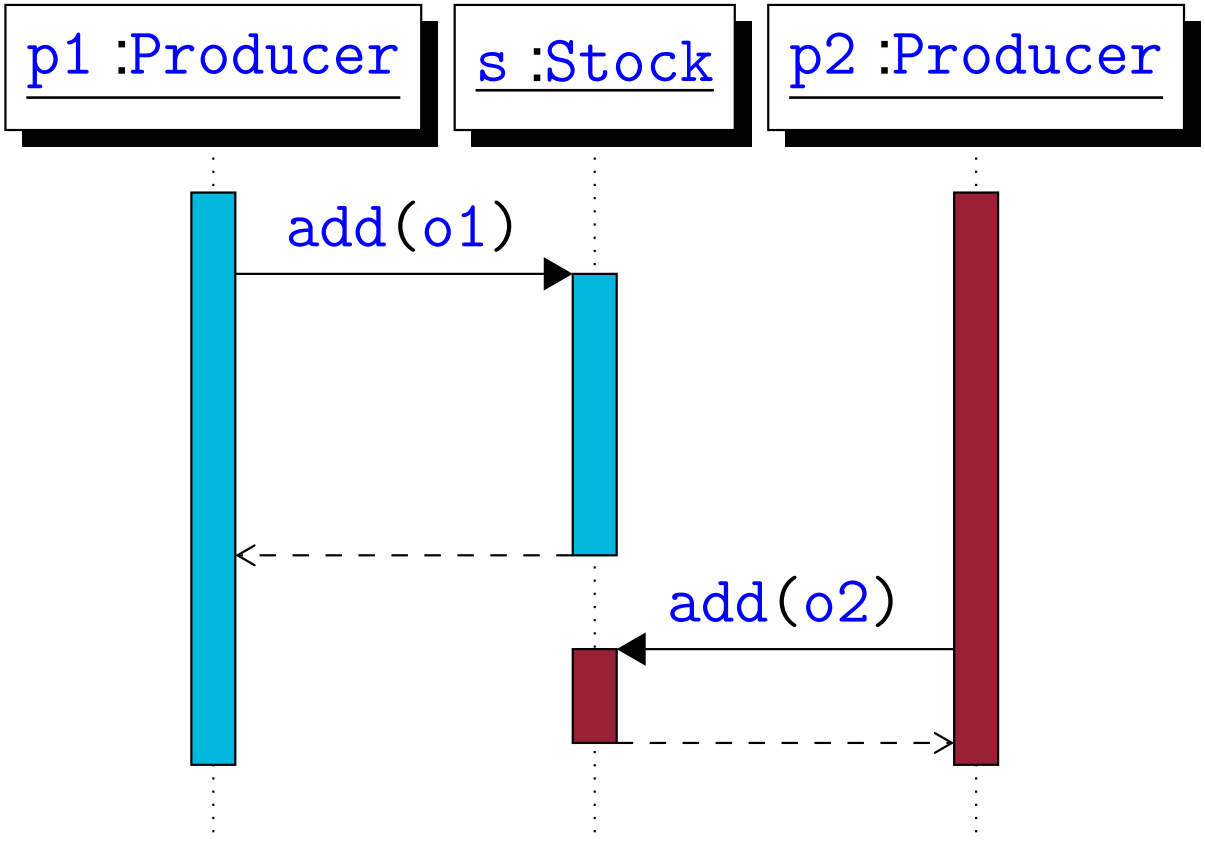
On a quelques intuitions de la concurrence et de ses difficultés.

Comment décrire le temps ?

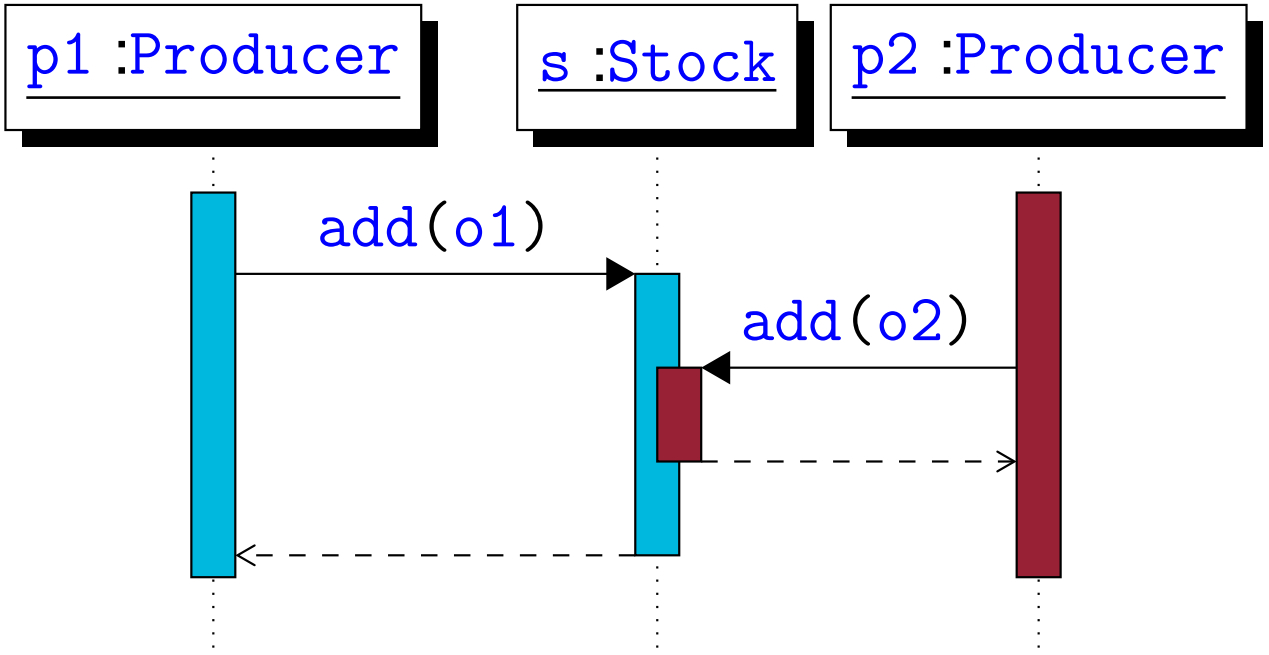
Comment décrire la concurrence ?

```
1  public class Producer {
2    ...
3    private Stock s ; //initialisé par le constructeur
4    ...
5    public void produce() {
6      Product o;
7      boolean full;
8      o = new Product("rubber");
9      s.add(o);
10     full = s.isEmpty();
11   }
12   ...
13 }
```


:Producer
o:Product
s:Stock
o=create()
add(o)
full=isEmpty()



Il y a des objets actifs (p1 et p2) et d'autres passifs (s)



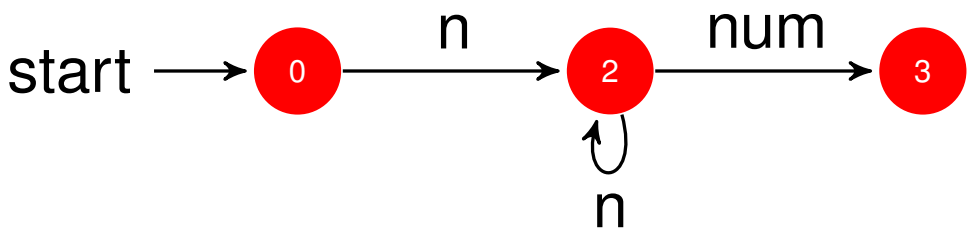
p1 et p2 veulent accéder en même temps à s. Il y a des risques si l'un des 2 modifie l'*état* de l'objet *partagé*.
Comment retarder ? Avec ou sans condition ? Par exemple si s est plein. . .

- Comment créer des activités ? la classe `Thread`
- Comment rendre atomique ?
- Comment assurer l'exclusion mutuelle ?
- Comment synchroniser ? faire attendre ?
- Comment utiliser ces outils ? Une méthode

Comment représenter des automates et des fils d'activité en UML ?

On considère un clavier de téléphone sur lequel on compose des chiffres pour former un numéro.

On considère un clavier de téléphone sur lequel on compose des chiffres pour former un numéro.

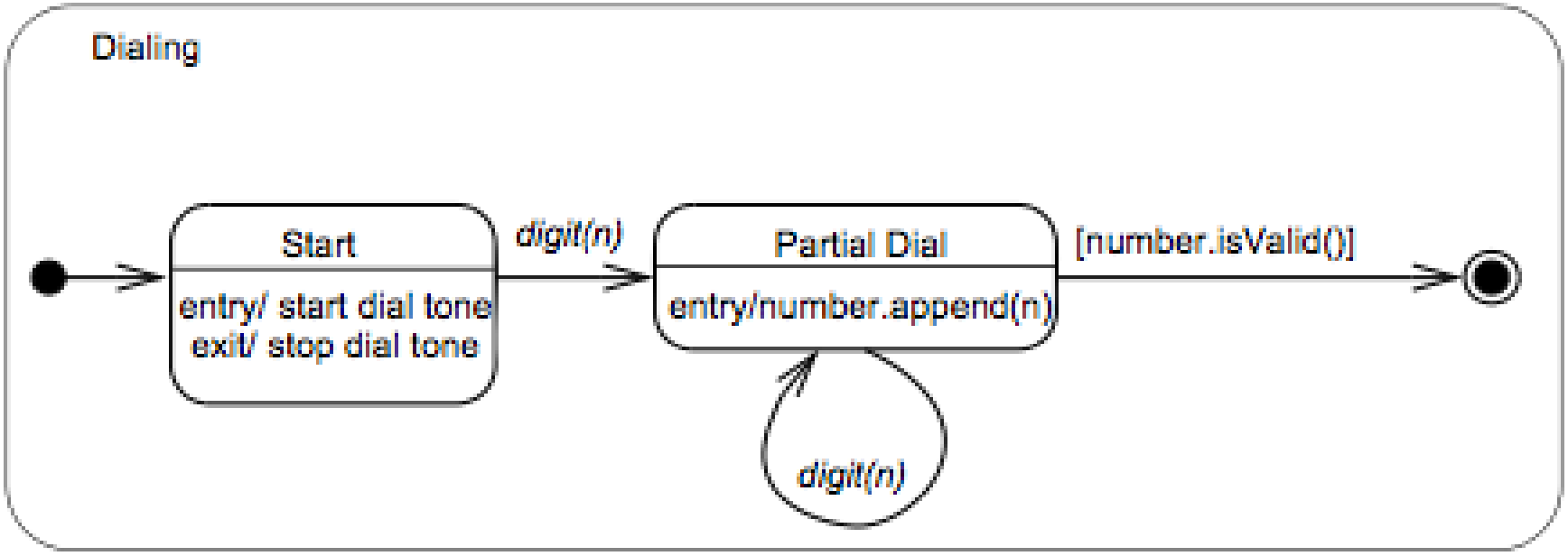


État Représente un moment dans le temps (qui peut durer). On distingue les états initiaux (sans origine) et finaux (sans issue).

Transition Représente un événement qui fait passer d'un état à un autre de manière (abstraite) instantanée. On distingue la transition ϵ (vide) qui permet une transition immédiate.

Garde Représente une condition pour qu'une transition puisse être *tirée*.

Action Opération attachée à une transition qui est exécutée lors du tirage de la transition.



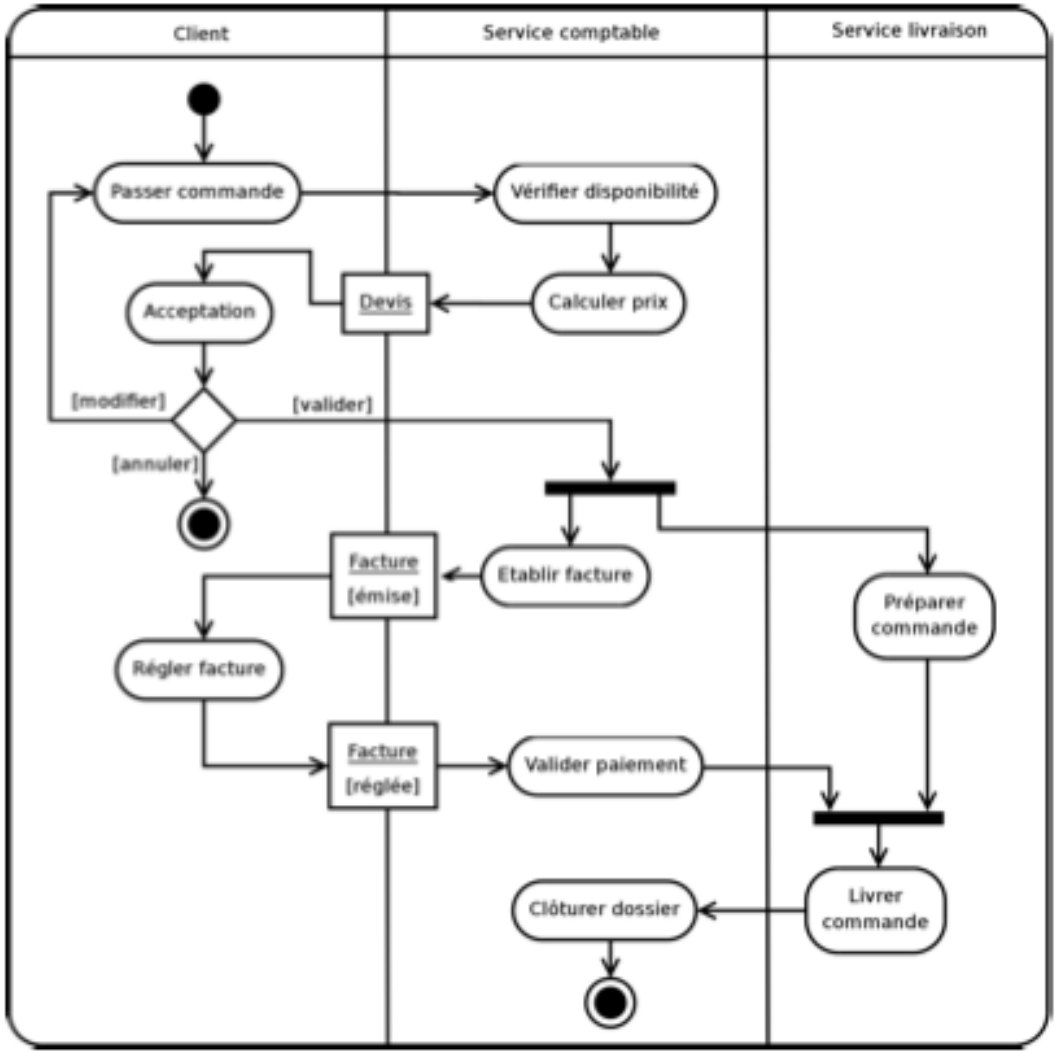
Un tel diagramme permet de décrire les évolutions des états d'un objet (ou d'un système).

- Ne permettent de décrire qu'un seul fil d'exécution
- Mais on peut composer des automates [voir cours suivants]
- On adopte une interprétation d'entrelacement :
  - On ne fait qu'une chose à la fois, mais en tenant compte de tous les entrelacements possibles.
  - On respecte les dépendances causales décrites par les automates.

Ex : (A ;B) || (x ;y) devient ABxy + AxBy + AxyB + xABy + xAyB + xyAB.

- Simples à décrire
- De nombreux algorithmes de calculs disponibles
- Permet de vérifier des propriétés. Par exemple :
  - vivacité (puits)
  - accessibilité (existence de chemin)

Diagramme d'activités (Activity)



- ActivityPartition
- InitialNode
- Action
- ControlFlow
- ObjectFlow
- DecisionNode
- ForkNode
- JoinNode
- ActivityFinal

Quand utiliser les différents diagrammes ?

- Diagramme de séquence : interactions entre objets
- Automate : un fil, évolution des états d'un objets
- Diagramme d'activité : plusieurs fils, plusieurs objets, peu de détails

Les diagrammes d'activité sont plus adaptés aux phases amont (analyse) du développement.

Un système (logiciel) nécessite de nombreux points de vue pour être décrit.
UML offre des outils pour décrire les points de vue :

- structurel : diagramme de classe, objet, et d'autres
- temporel : diagramme d'état, d'activité, de séquence, et d'autres
- fonctionnel : diagramme de cas d'utilisation ou d'activité

Ce cours donne quelques outils et intuitions sur la façon de décrire la concurrence.

La suite de l'UE approfondira la compréhension de la concurrence et proposera des outils (conceptuels et pratiques) pour

- Décrire des systèmes concurrents avec précision (mathématiquement)
- Vérifier des propriétés (preuves)

Le cours Java complètera avec une vision plus opérationnelle (exécutable) de la concurrence.

- **Spécification**
  - Processus
  - Activité
    - seq. d'actions
  - Contraintes
  - Dépendances
- **Programmation**
  - Tâche
    - seq. d'instructions
  - *Thread* (java)
  - *Runnable* (java)
- **Exécution**
  - Trace
  - Calcul
  - Support
    - Processeur
    - Cœur
    - Système d'exploitation
    - thread/process

Vocabulaire

- Flux, (*Thread*)
- Concurrence
- Non déterminisme
- Entrelacement
- Object actif, passif
- État, transition
- Partage

Diagramme UML

- de séquence
- d'état
- d'activité

Automate

- interblocage
- atomicité
- exclusion mutuelle