



# Modélisation comportementale en UML

Elie Najm



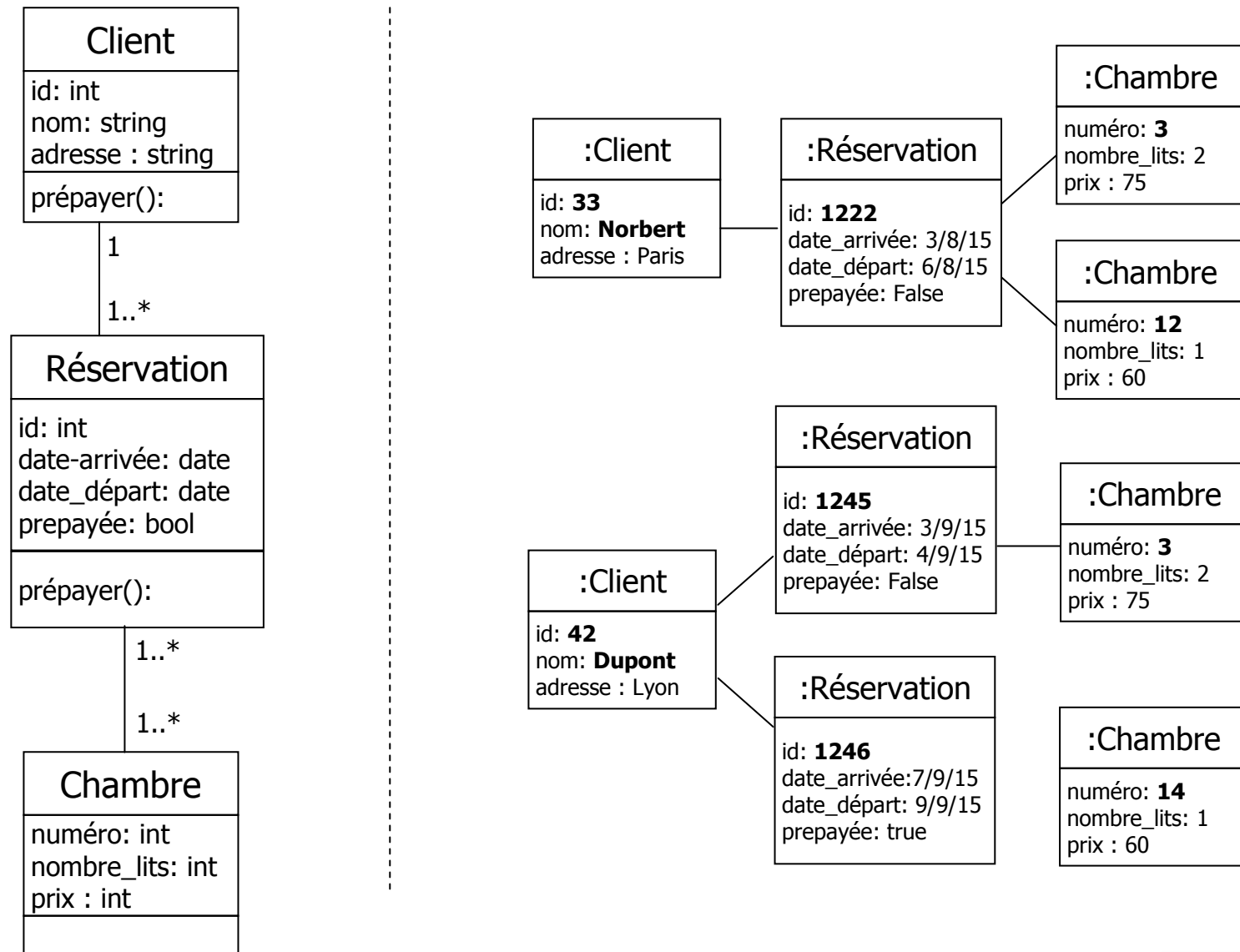
(certains transparents sont inspirés de tutoriels donnés à l'OMG)

# Les principaux diagrammes UML

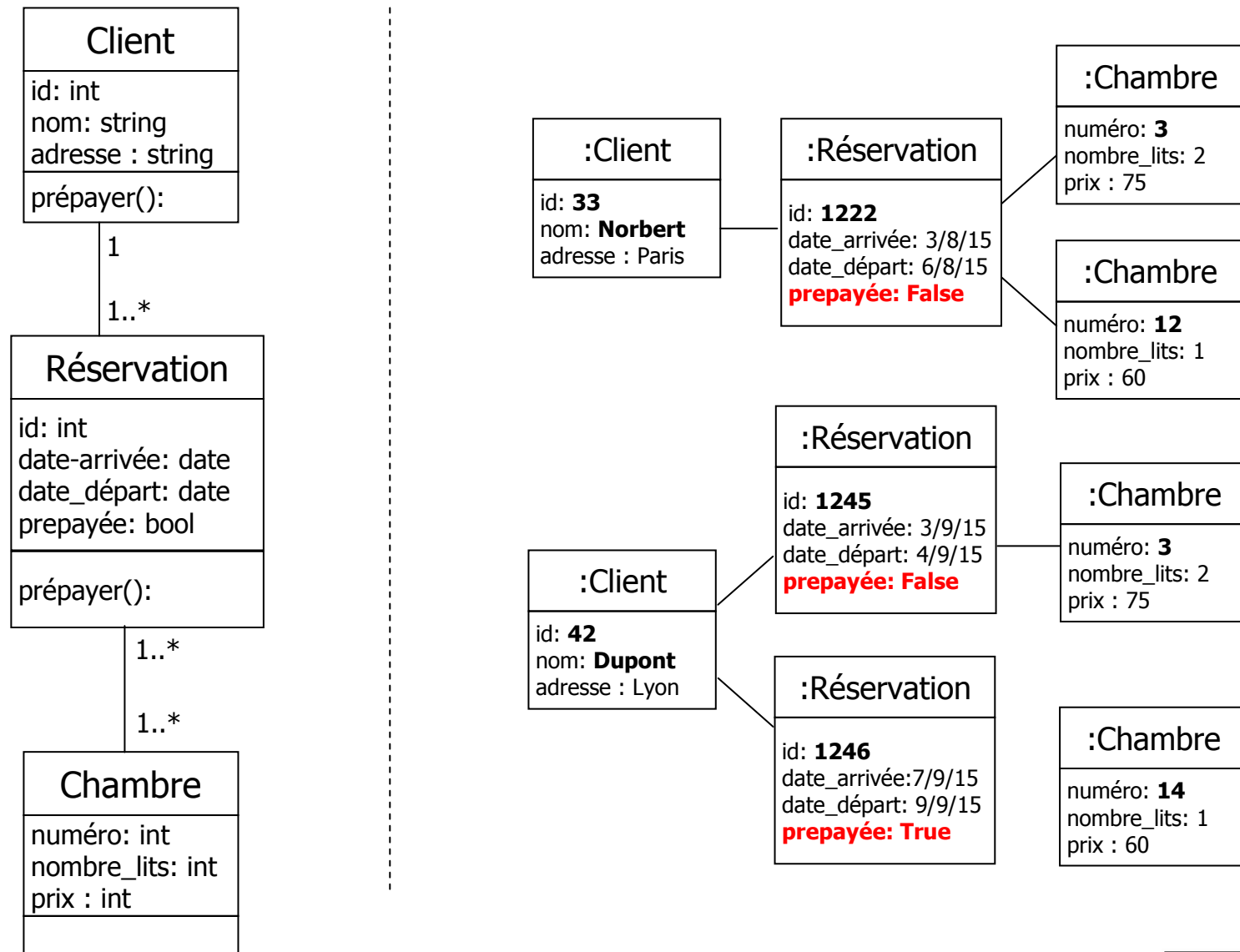
---

- Description des besoins
  - Diagramme de cas d'utilisation
- Vue structurelle (aspect statiques d'un modèle)
  - Diagramme de classe
  - Diagramme d'objet
- Vue fonctionnelle (interaction entre objets)
  - Diagramme de séquence
  - Diagramme de communication
- Vue comportementale (dynamique des objets)
  - Diagramme d'activité
  - Machine à états
- Vue réalisation et déploiement
  - Diagrammes de composants
  - Diagrammes de déploiement

# Vue structurelle et Système réel

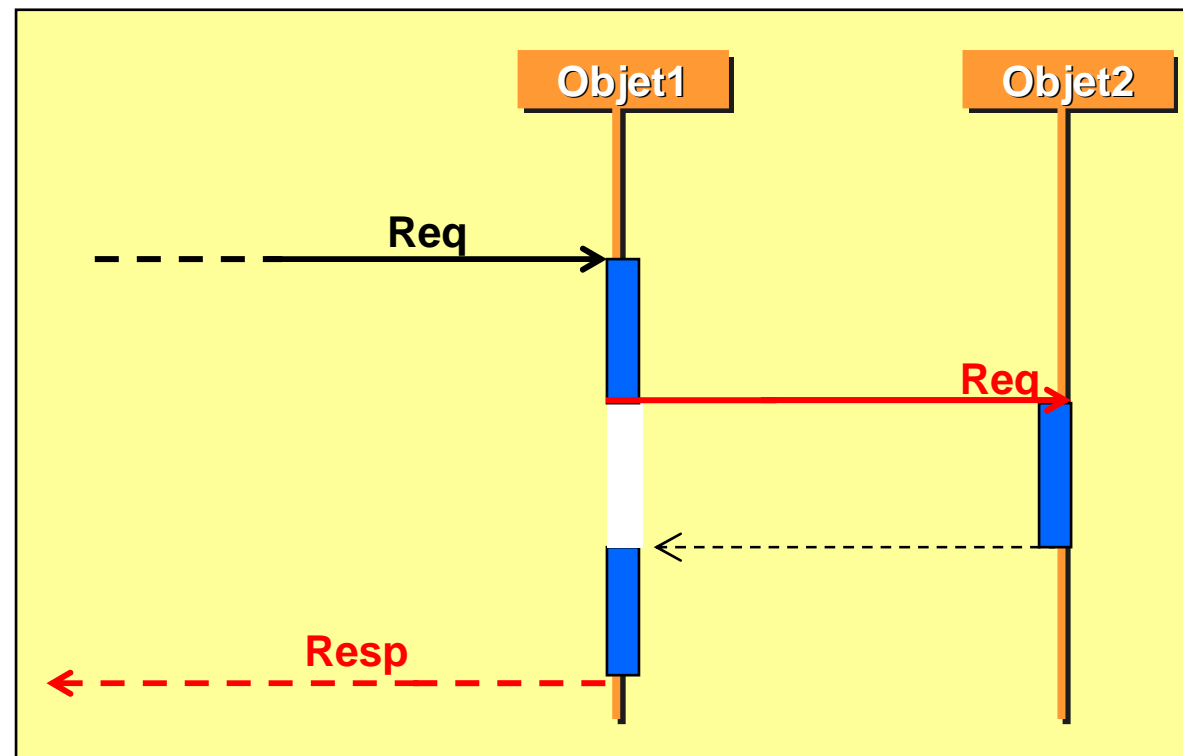


# Vue structurelle et Système réel



# Comportement des Classes “classiques”

- Recevoir des invocations d'opérations et les traiter
- Dans un traitement, émission d'invocations et attente de leur retour
- Comportement uniforme (ne change pas dans le temps)



# Besoin d'un comportement basé état

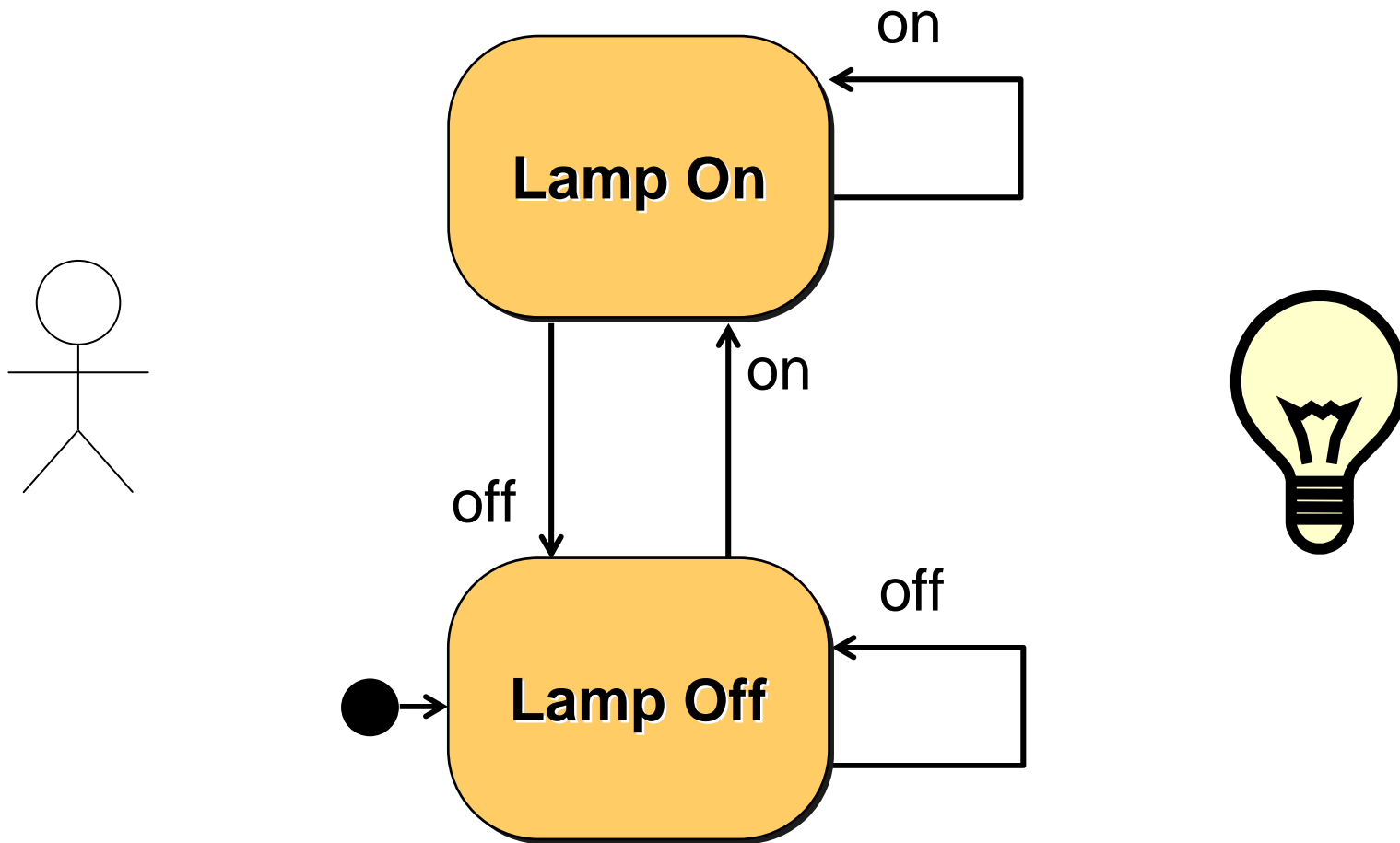
## AUTOMATE

- Une "machine" dont le comportement en sortie ne dépend pas seulement de *l'entrée courante*, mais aussi de *l'historique des entrées*
- L'*Historique des entrées* peut être caractérisée par *un état* qui constitue *l'abstraction de l'expérience passée*.

=> Diagramme de Machine à Etats (UML 2.0)  
StateCharts (UML 1.4)

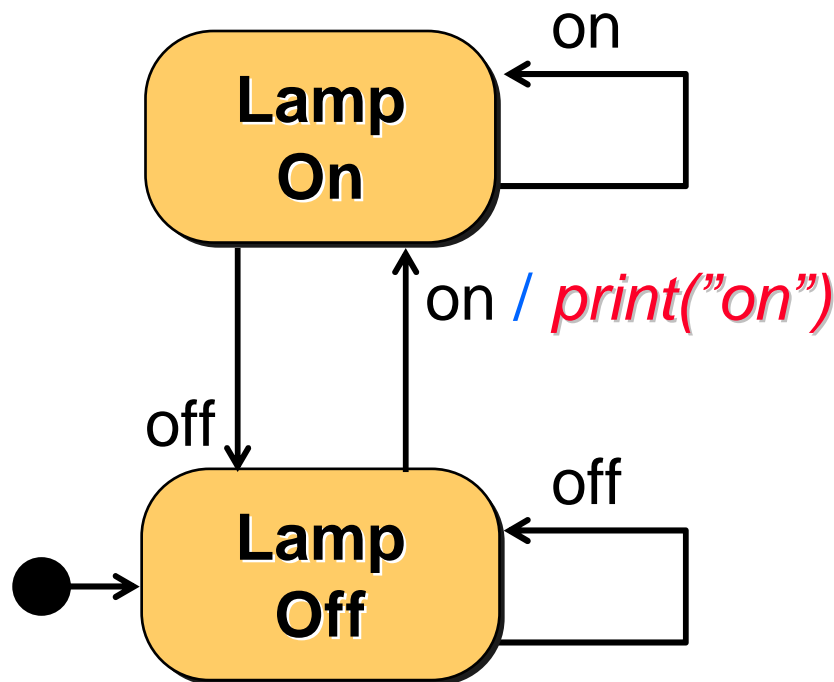
# Diagramme de Machine à Etat (exemple)

Un contrôleur de lampe

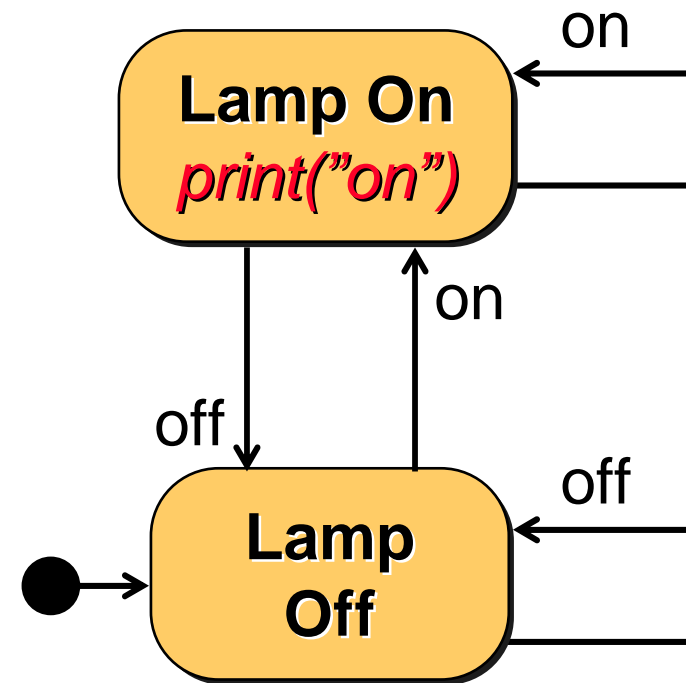


# Actions et Outputs (I)

- Quand l'automate change d'état il peut faire des Actions:



**Mealy** automaton

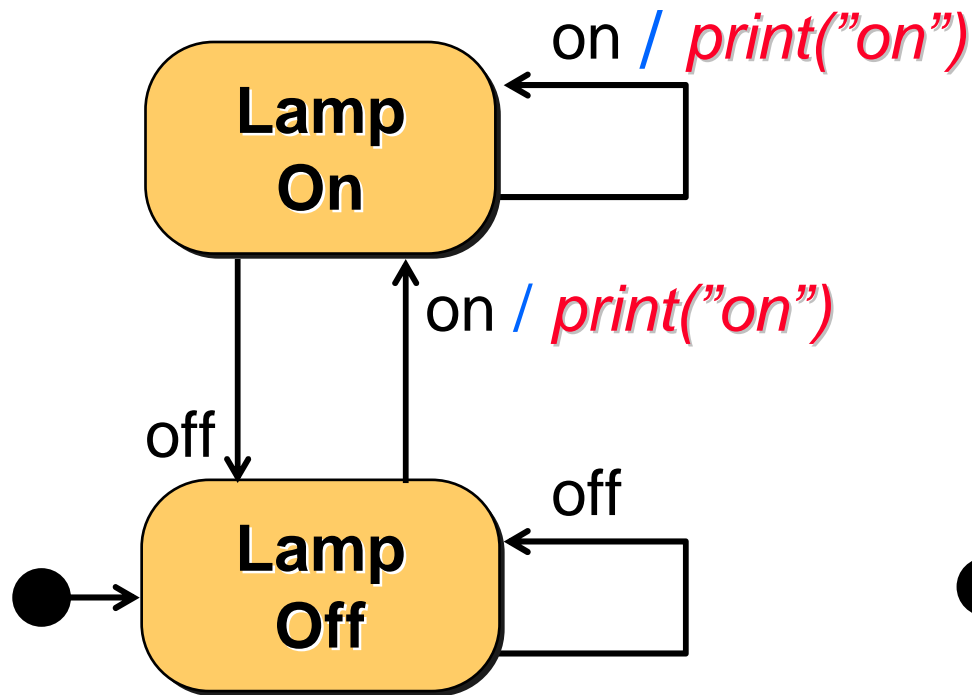


**Moore** automaton

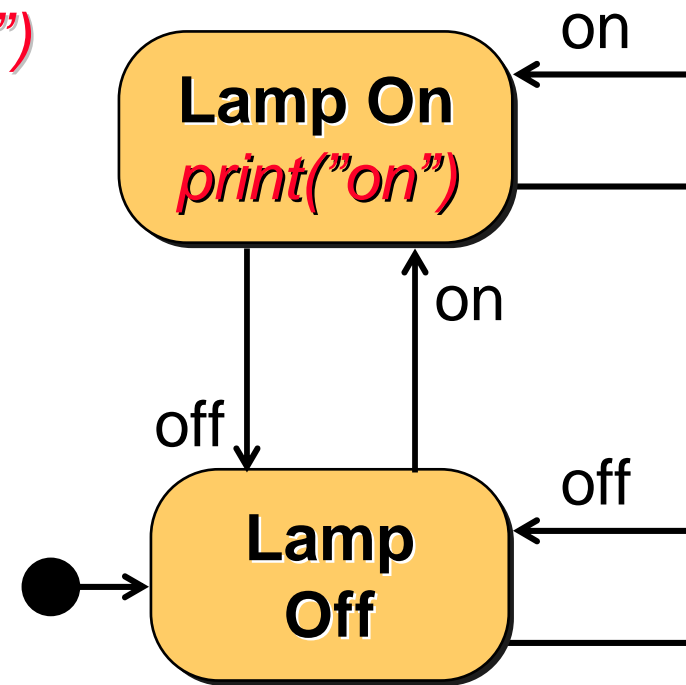


# Actions et Outputs (I)

- Equivalences entre diagrammes ?



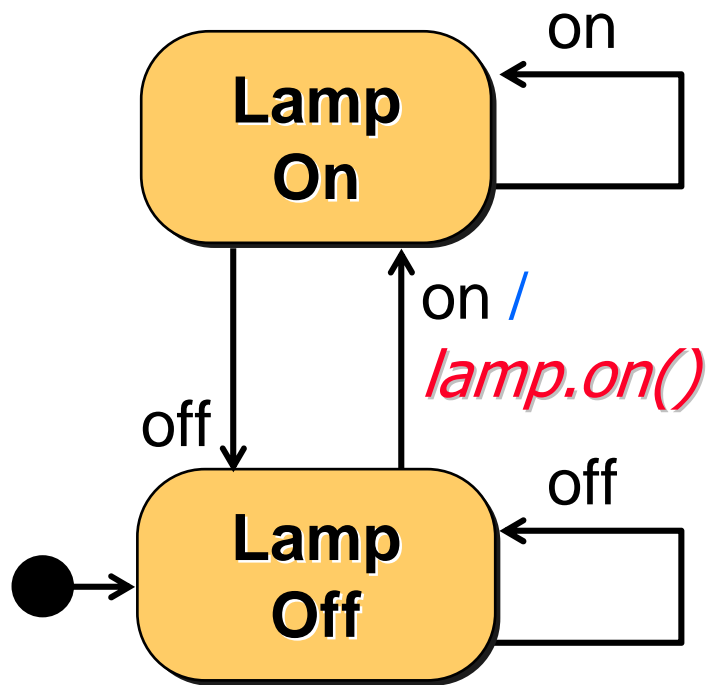
**Mealy** automaton



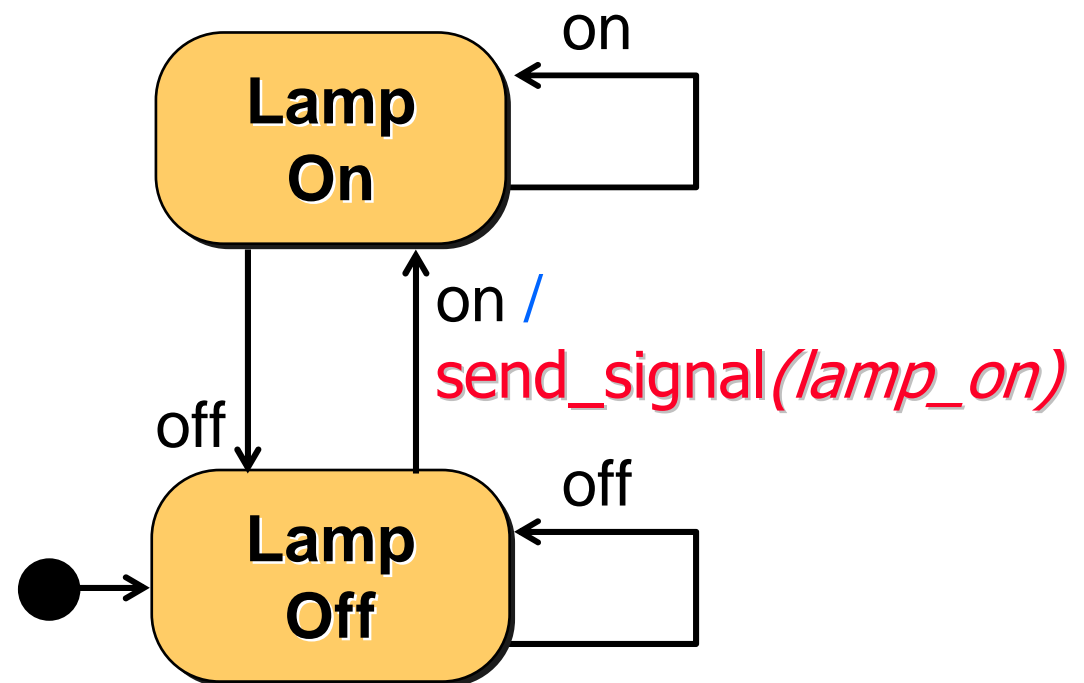
**Moore** automaton

# Actions et Outputs (II)

- Quand l'automate change d'état il peut exécuter une action d'output: appel synchrone ou signal asynchrone



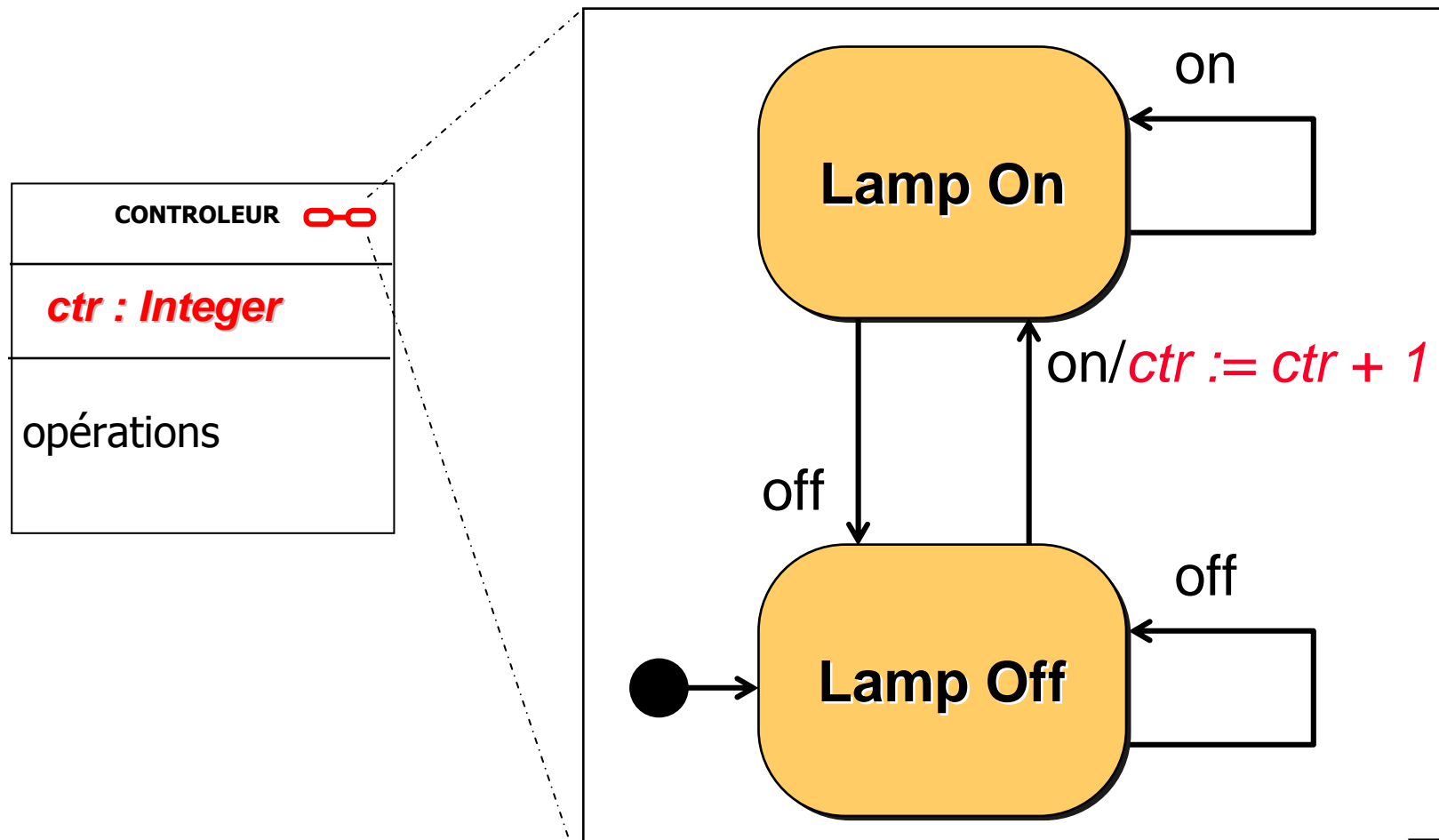
**appel synchrone**



**envoi d'un signal**

# Machine à Etats attachée à une classe

- Opérations et Attributs de la classe utilisés dans les actions ("état étendu")

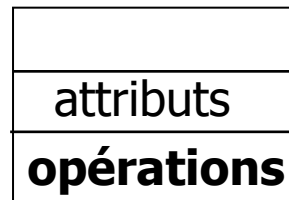


# Machine à Etats réagit aux évènements

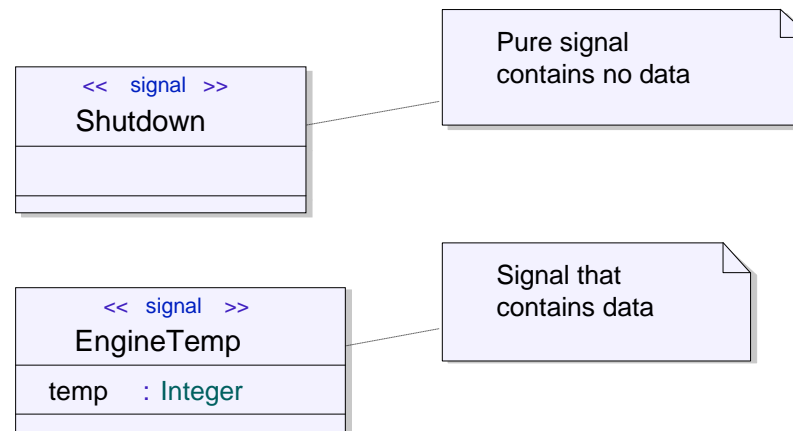
- Types d'évènements
  - interaction:
    - Appel synchrone d'une **operation** sur un objet (call event)
    - Réception d'un **signal** asynchrone (signal event)
  - occurrence temporelle (time event)
    - Expiration d'un délai
    - Arrivée à un instant prédéfini
  - Changement d'une valeur d'une variable (change event)

# Signaux et Opérations : déclarations

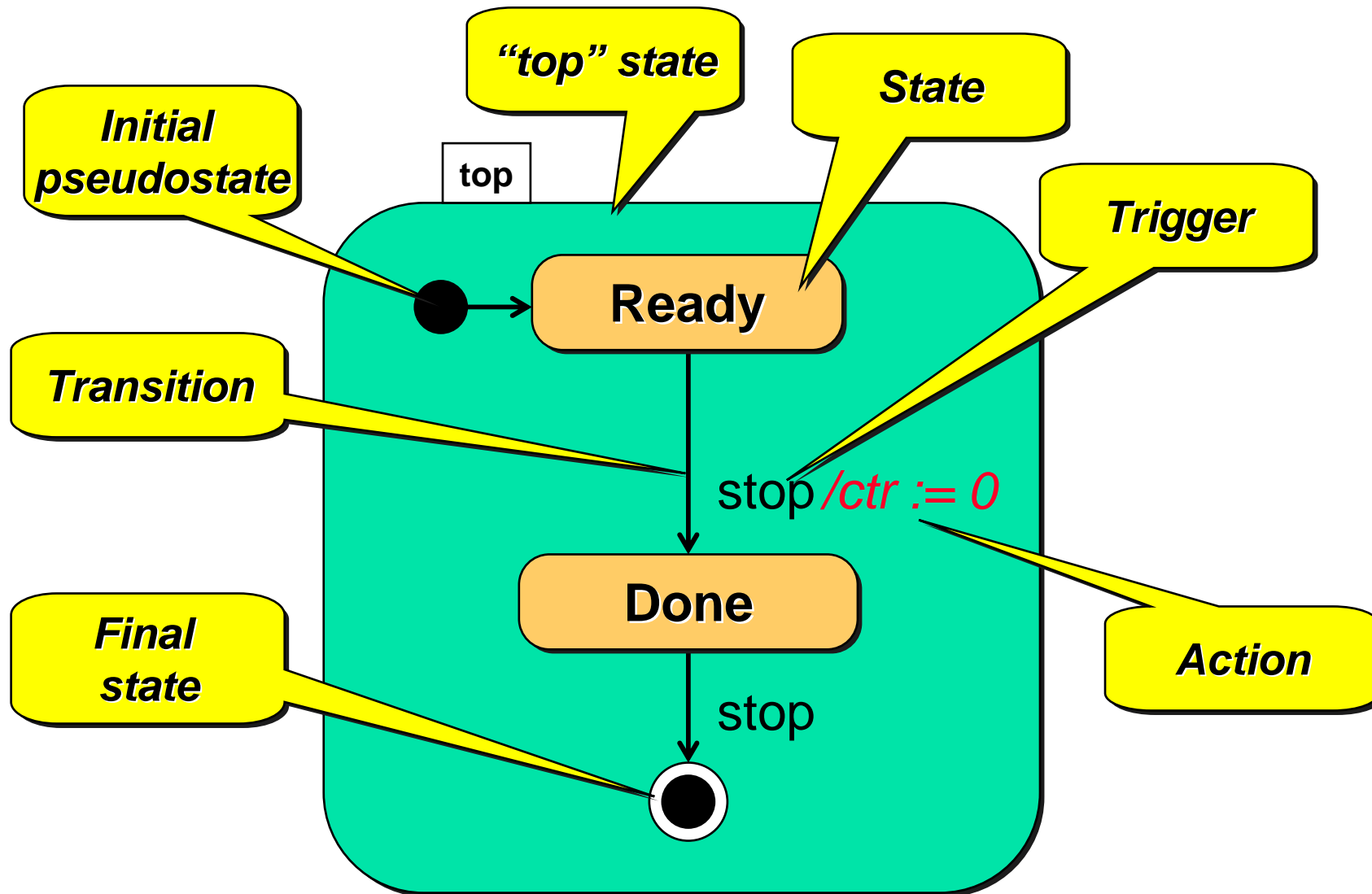
Opérations : déclarées dans (et attachées aux) classes



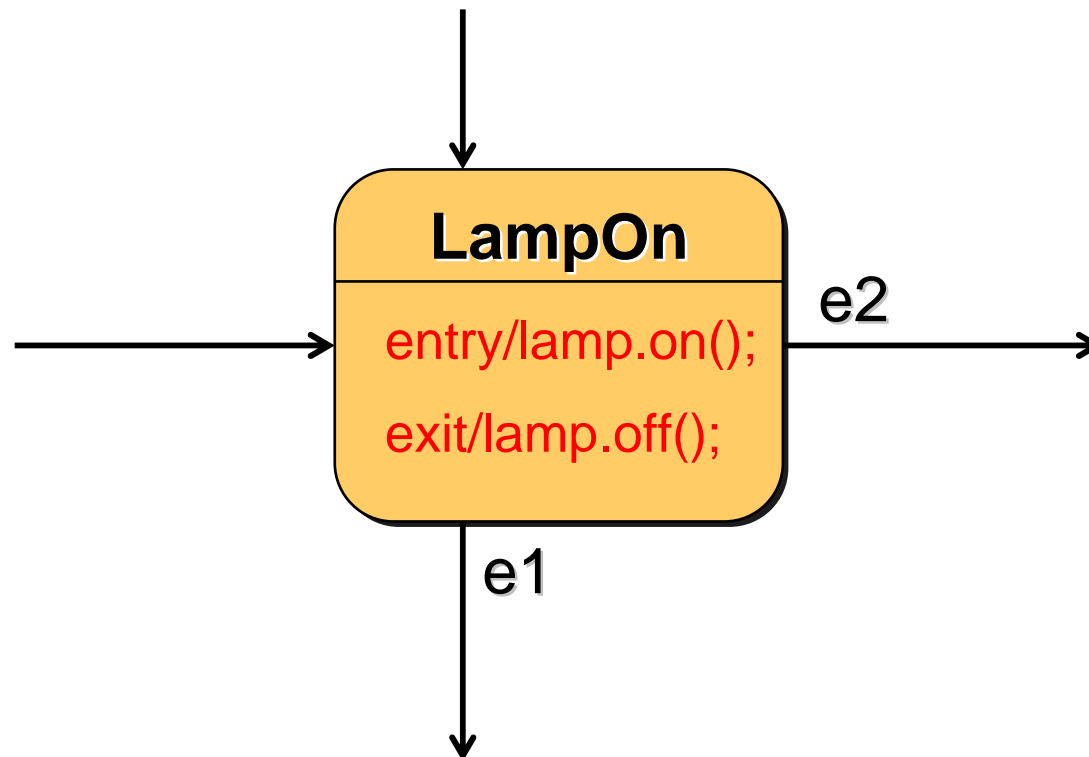
Signaux : déclarés de façon indépendante (classes)



# Diagramme de Machine à Etat simple

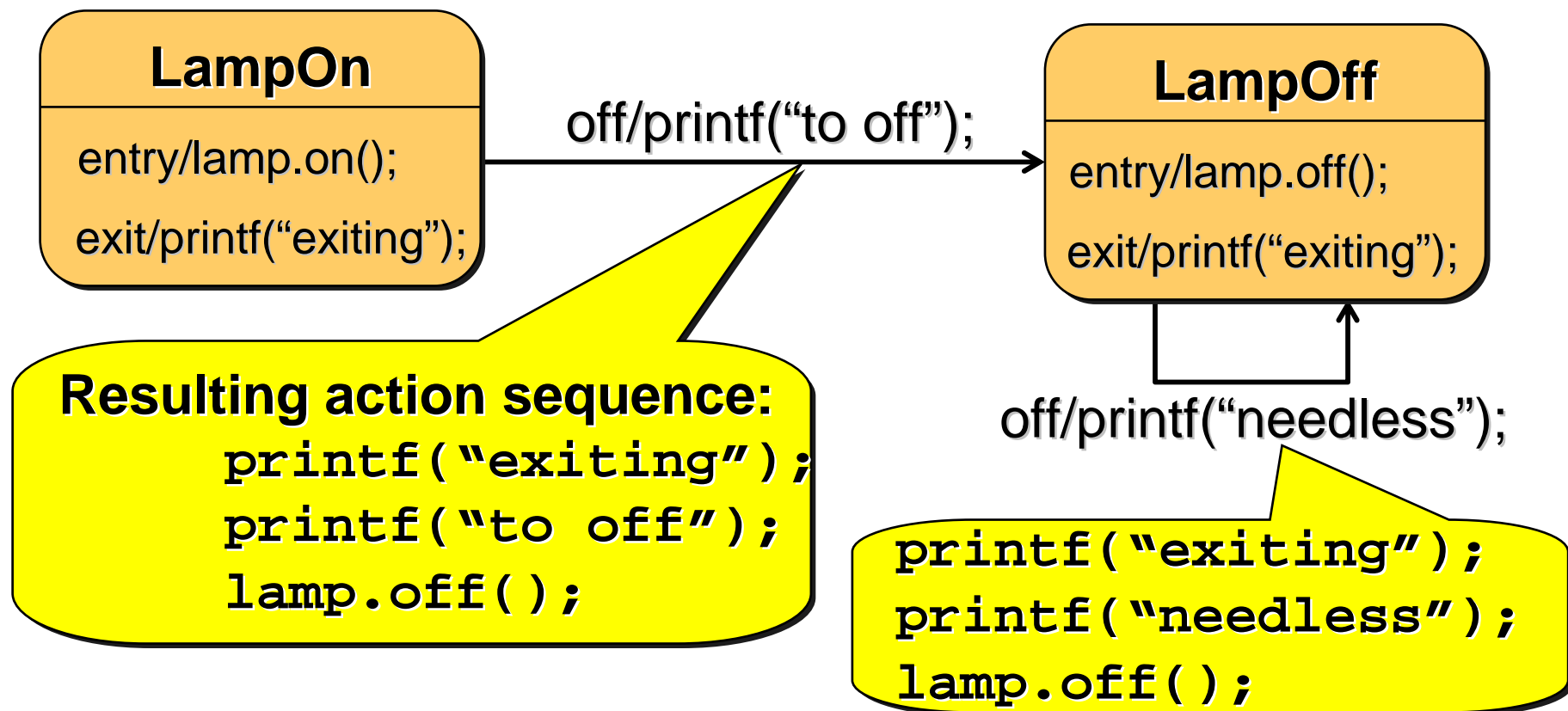


# Actions Entry and Exit



# Ordre des Actions: Cas Simple

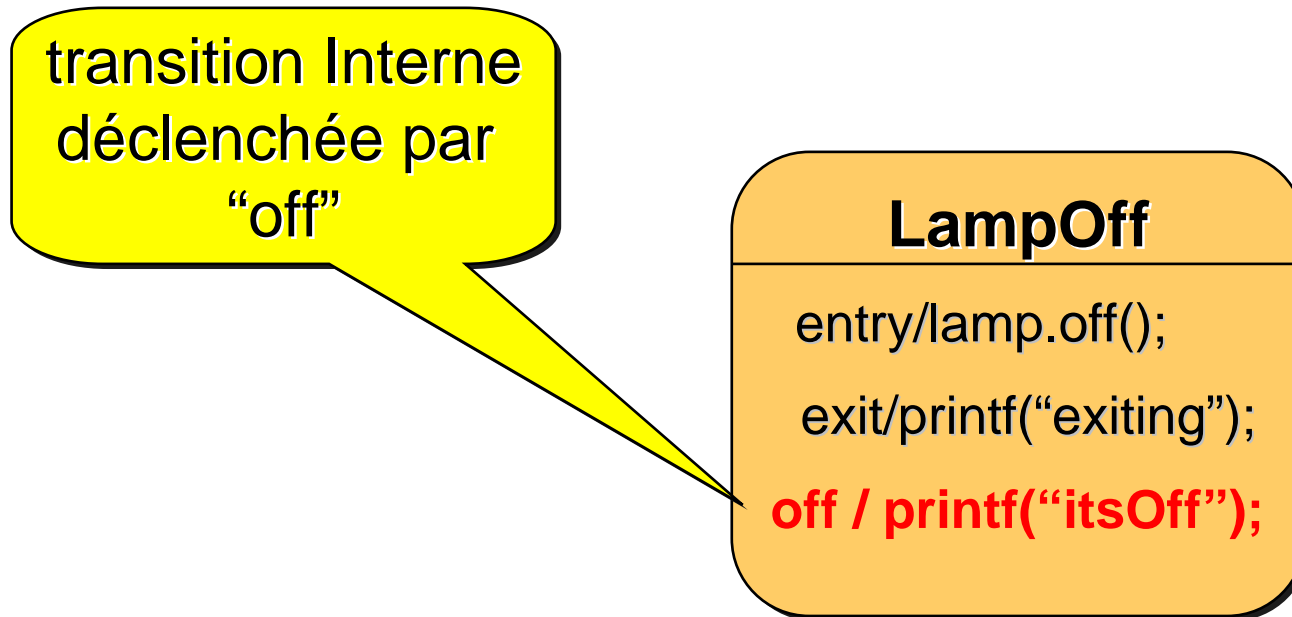
- Les actions Exit précèdent les actions des transitions
- Les actions Entry s'exécutent après les actions des transition





# Transitions Internes

- Évitent l'exécution des actions entry et exit

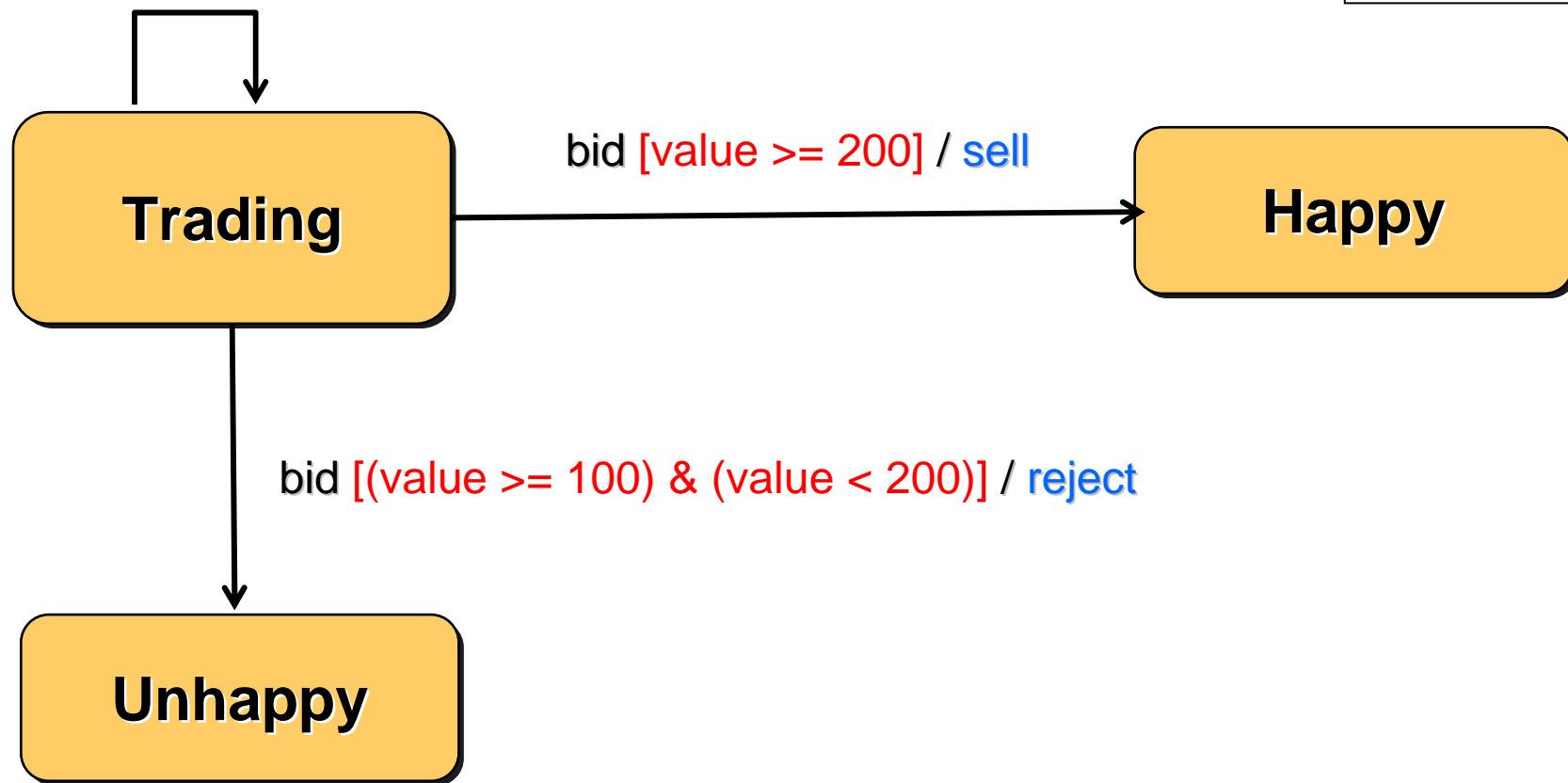


# Gardes

- exécution conditionnelle des transitions
- prédicats booléens sans effet de bord

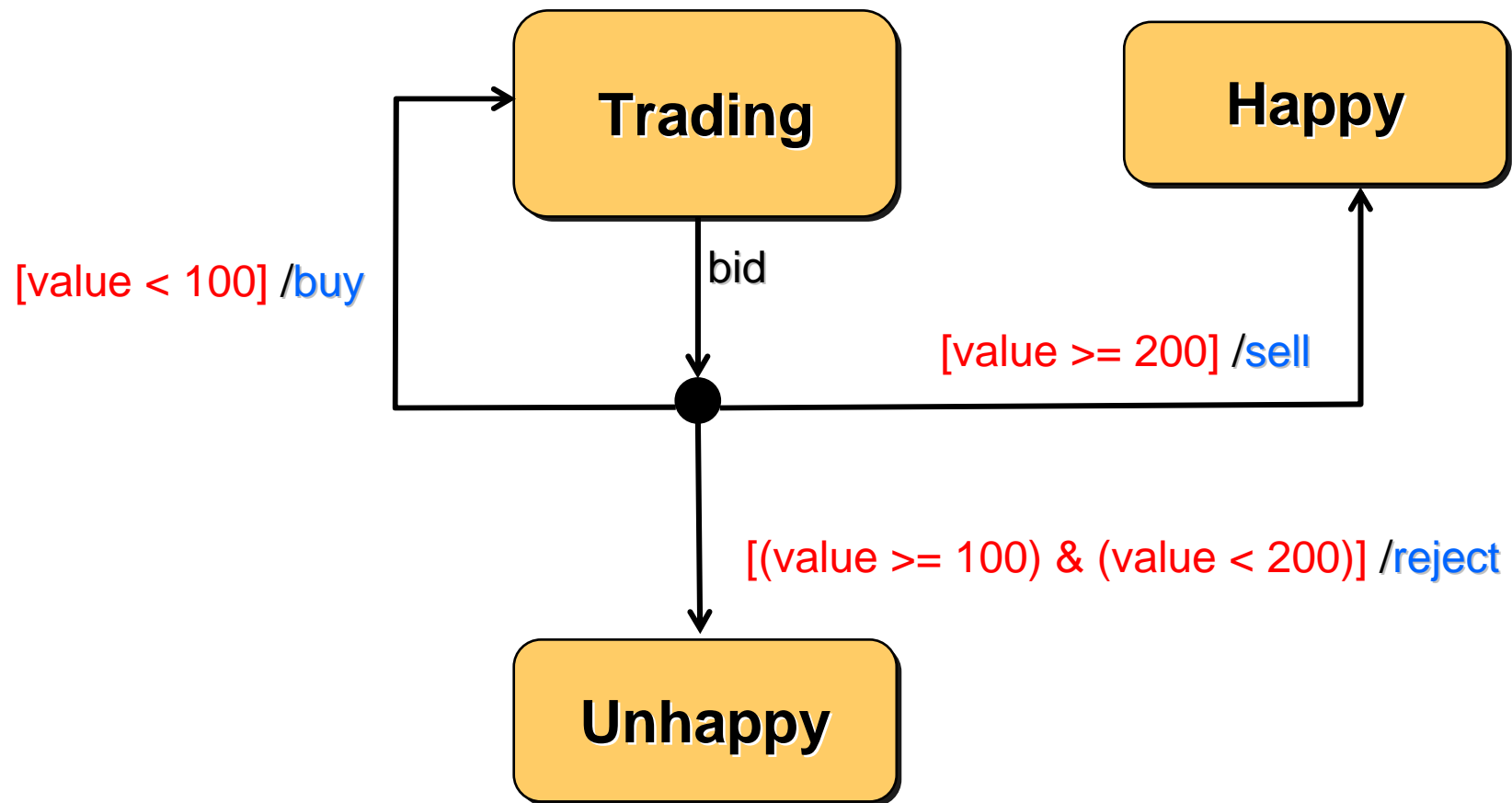
bid [value < 100] / buy

<< signal >> bid
value: int



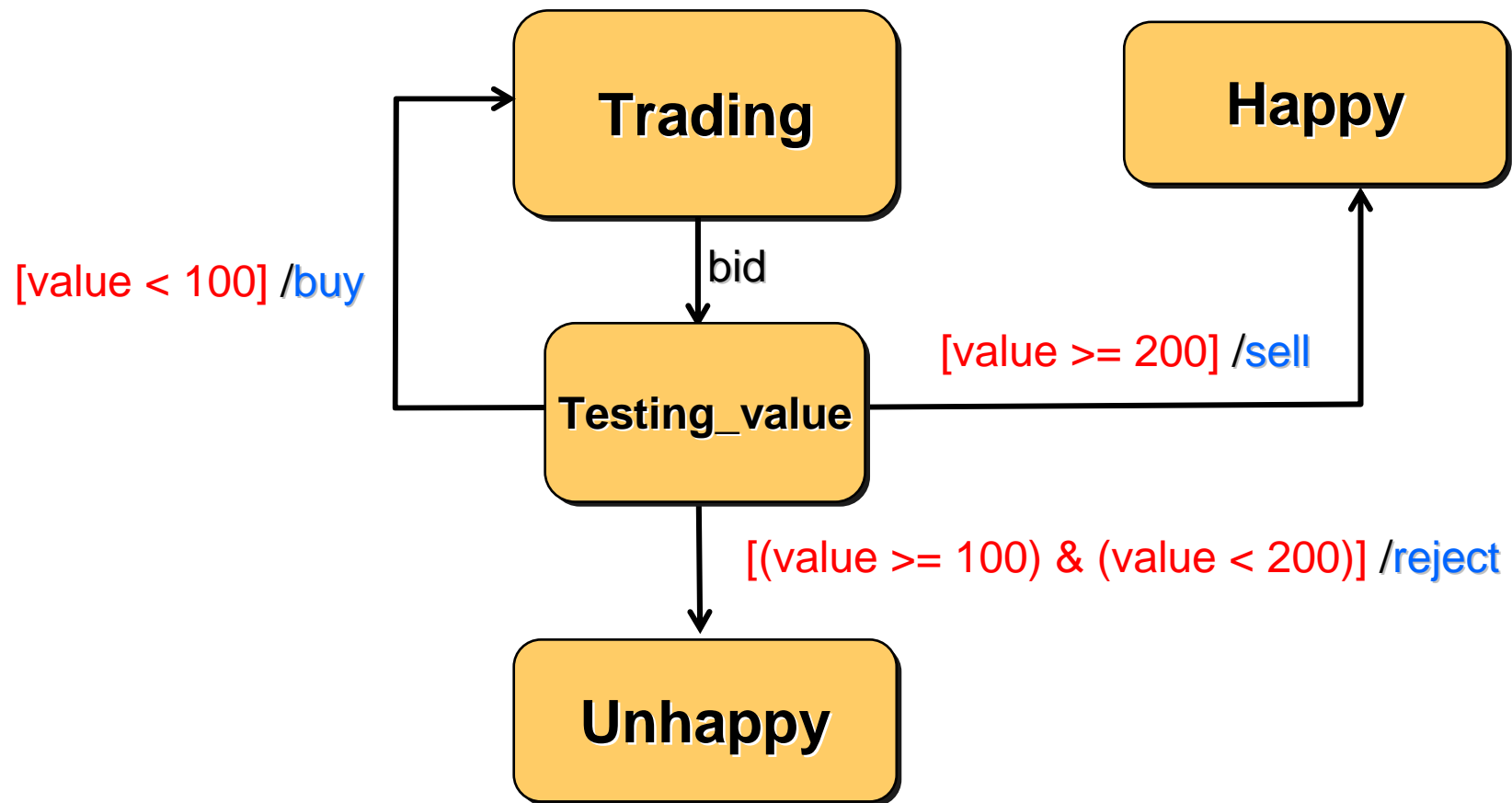
# Branchement Conditionnel

- Raccourci graphique pour un meilleur rendu des arbres de décision

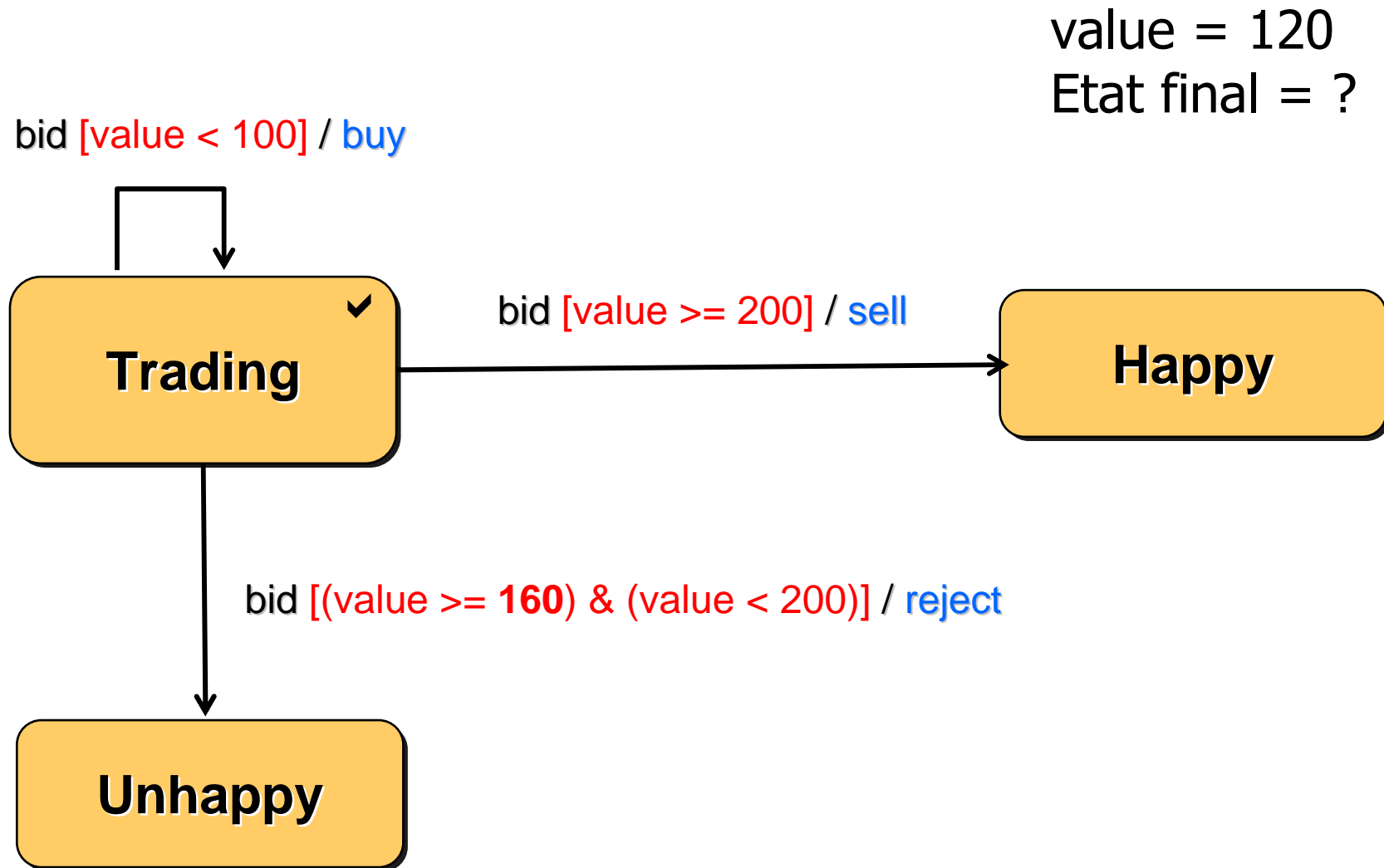


# Branchement Conditionnel

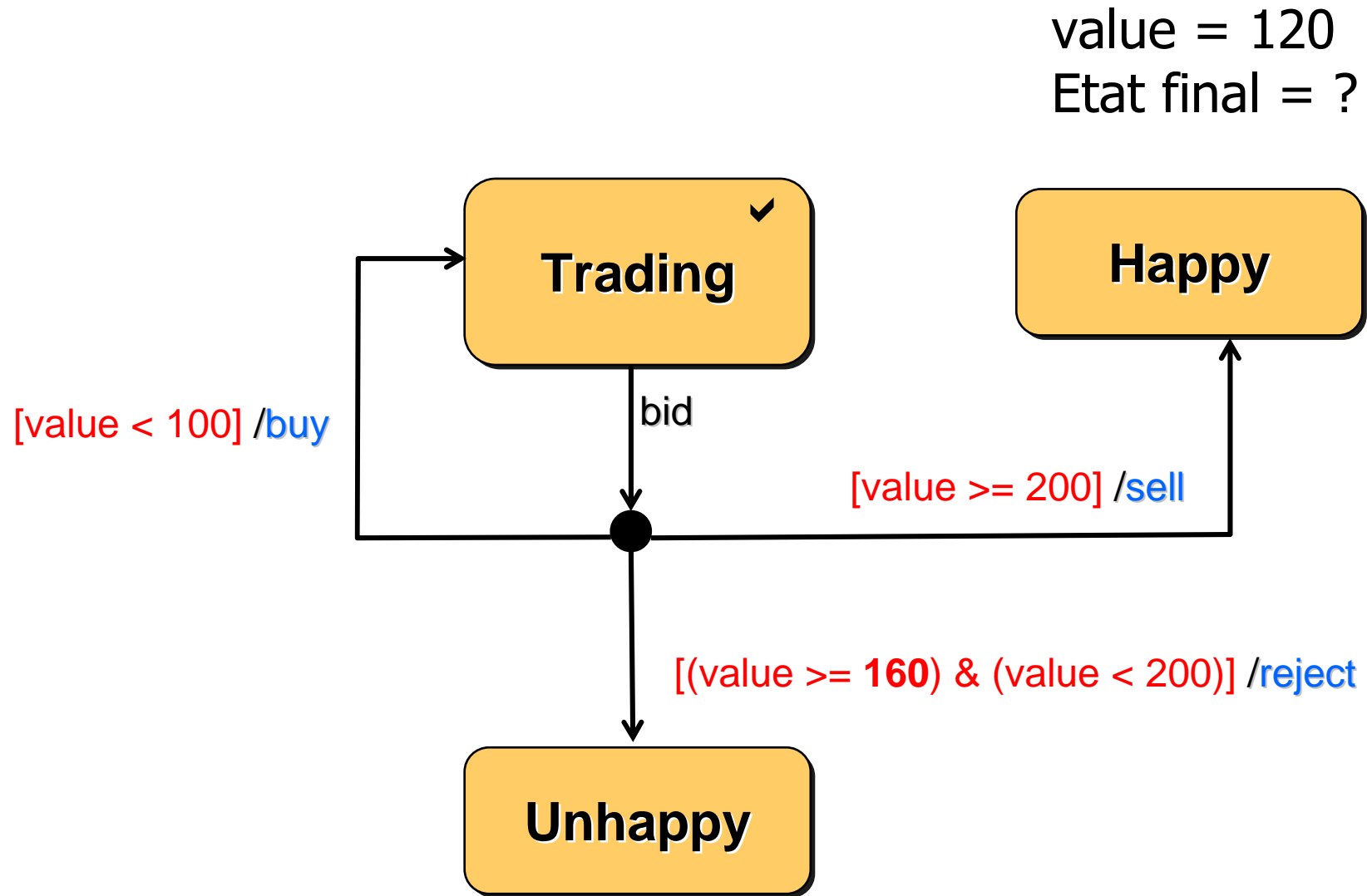
- Utilisation d'un état à la place du pseudo-état



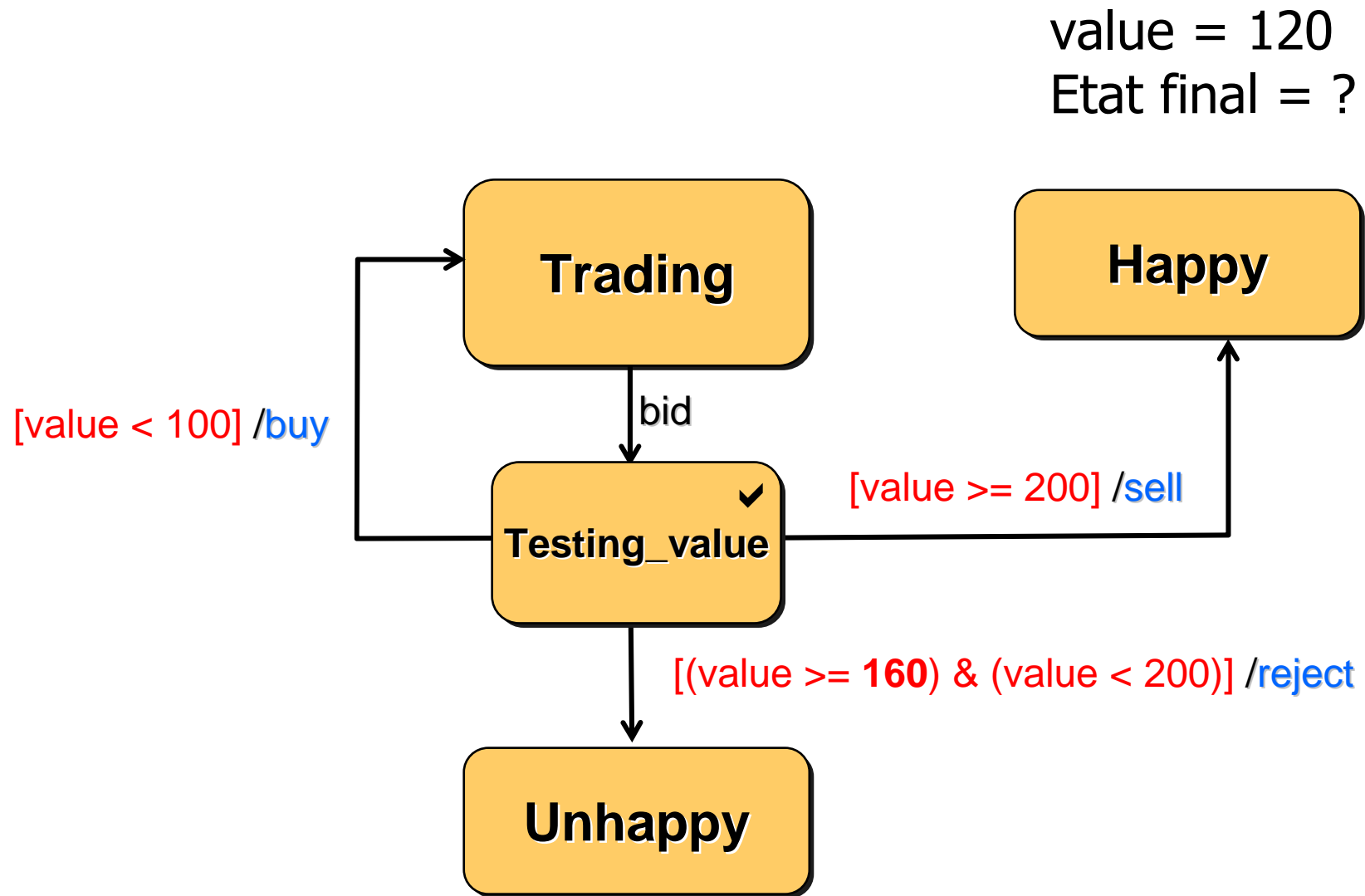
# Branchement conditionnel et transitions



# Branchement conditionnel et transitions

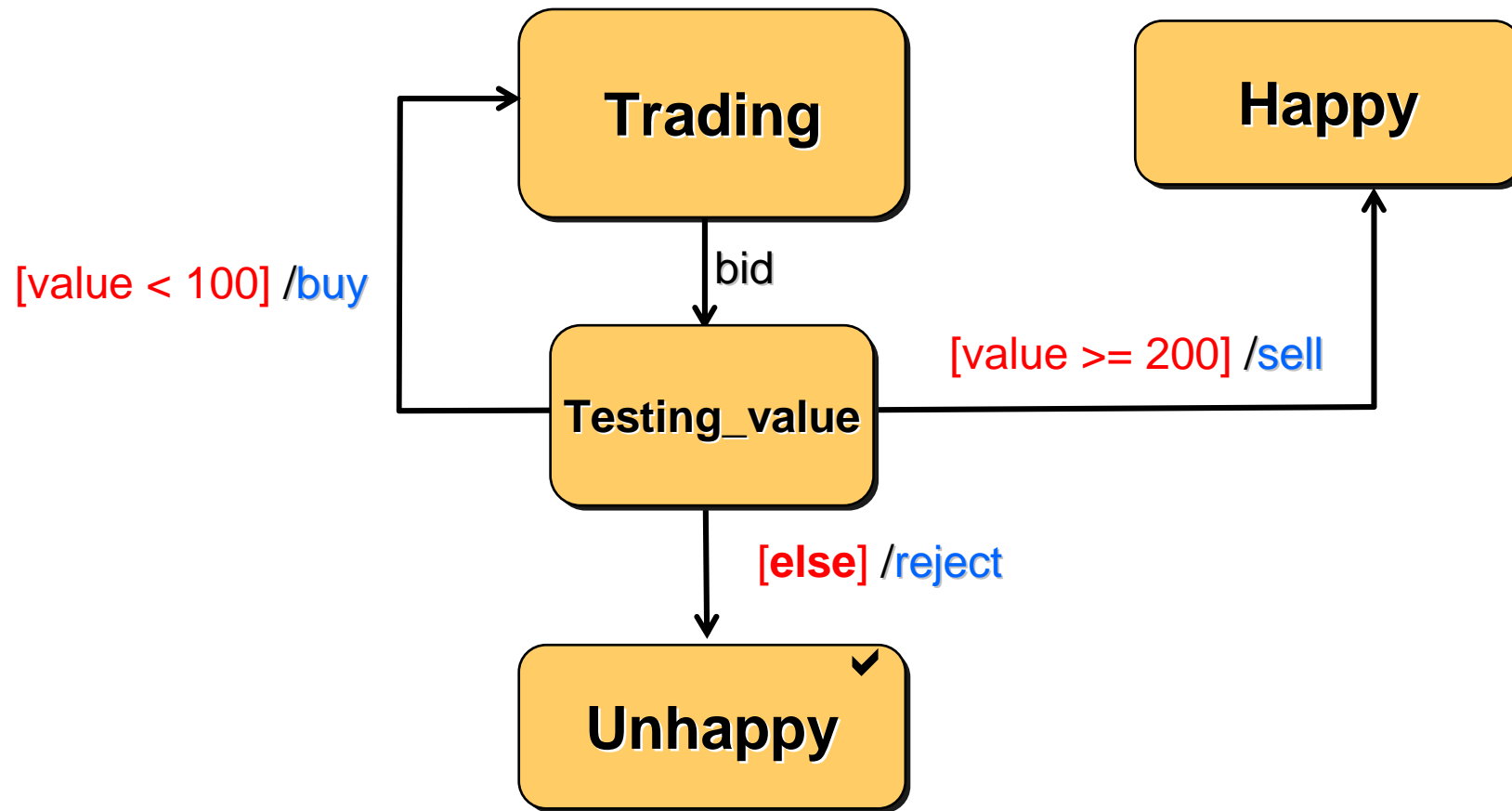


# Branchement conditionnel et transitions



# Branchement conditionnel et transitions

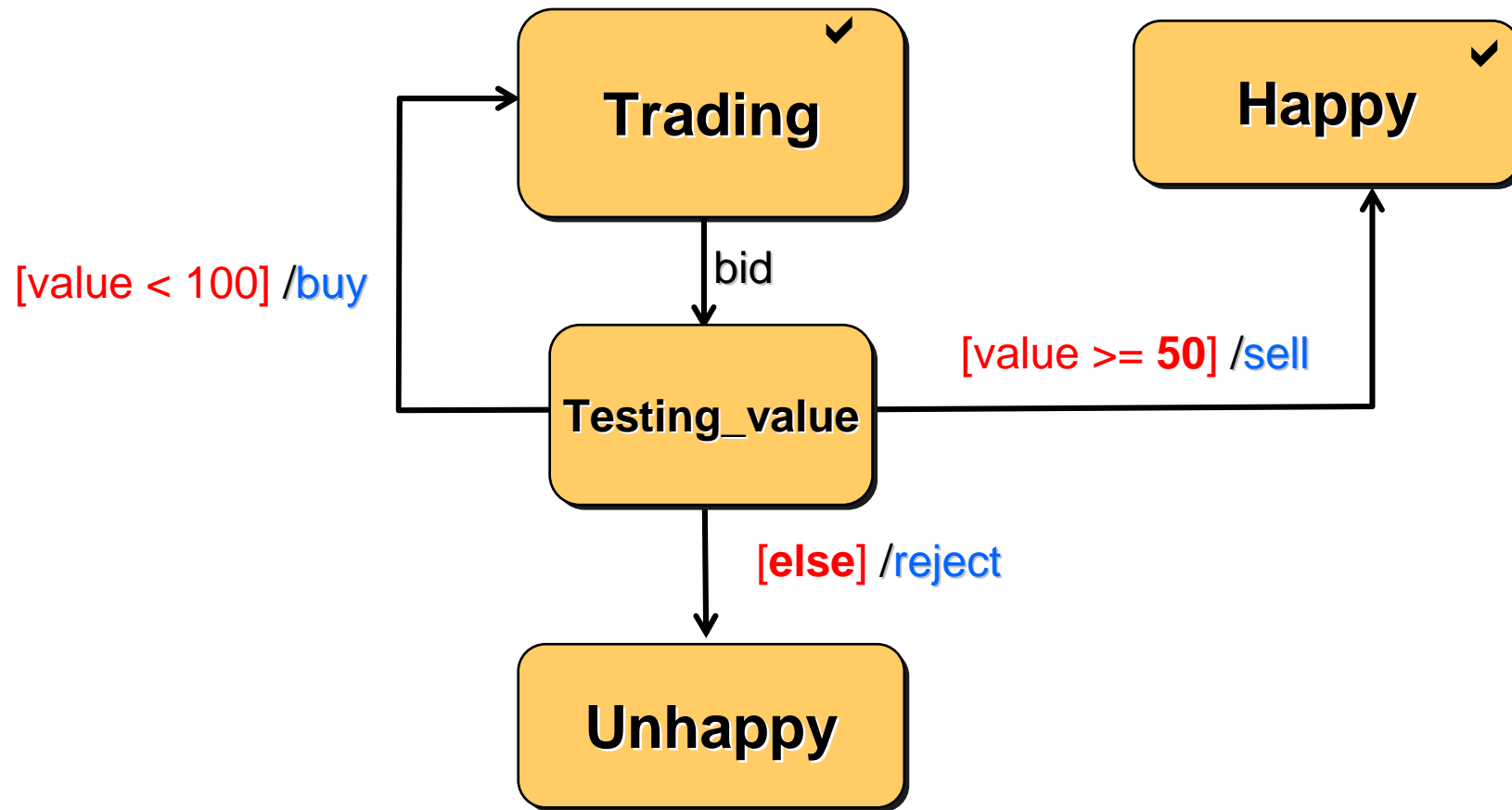
value = 120  
Etat final = ?



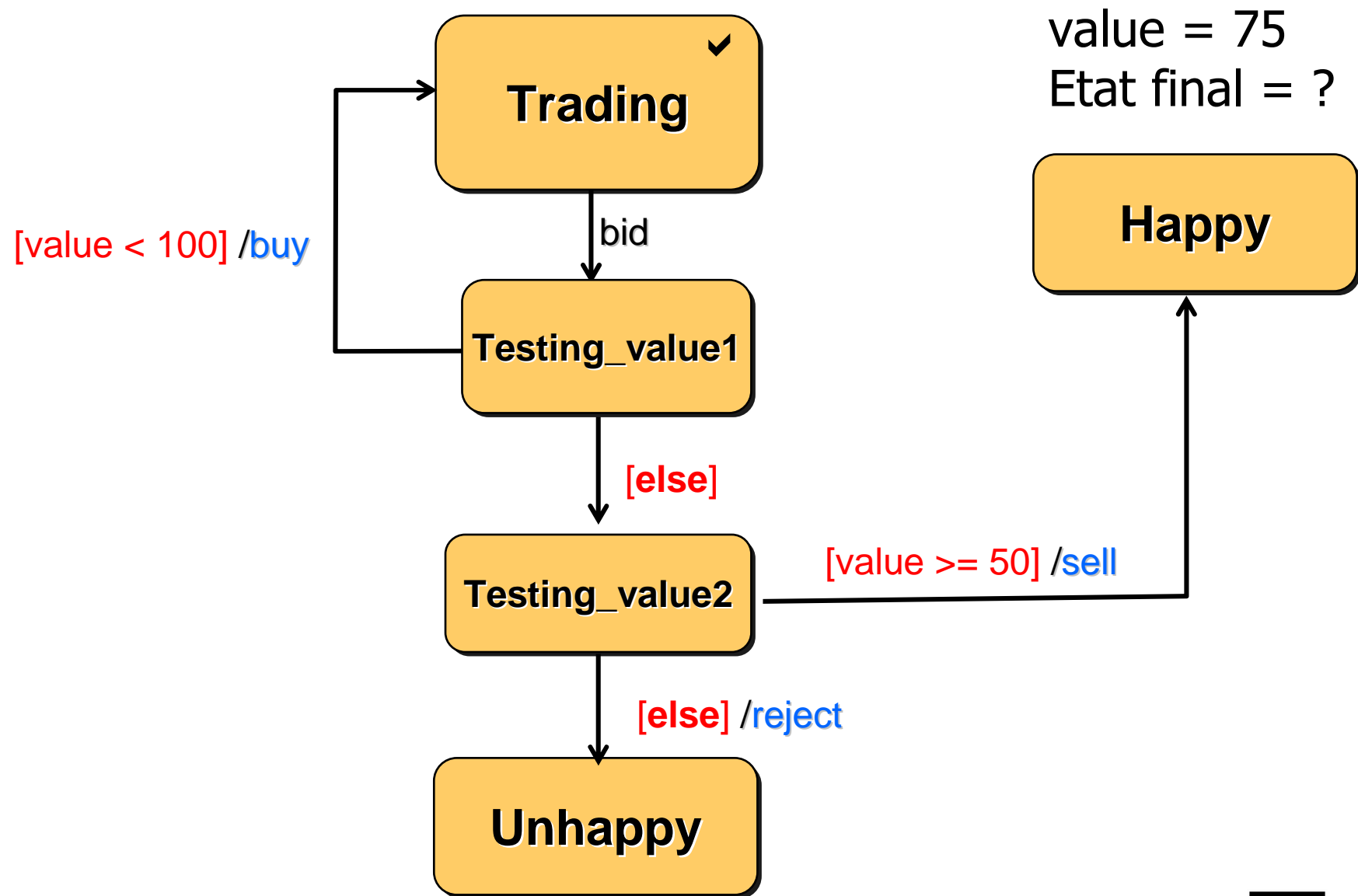


# Transition et non déterminisme

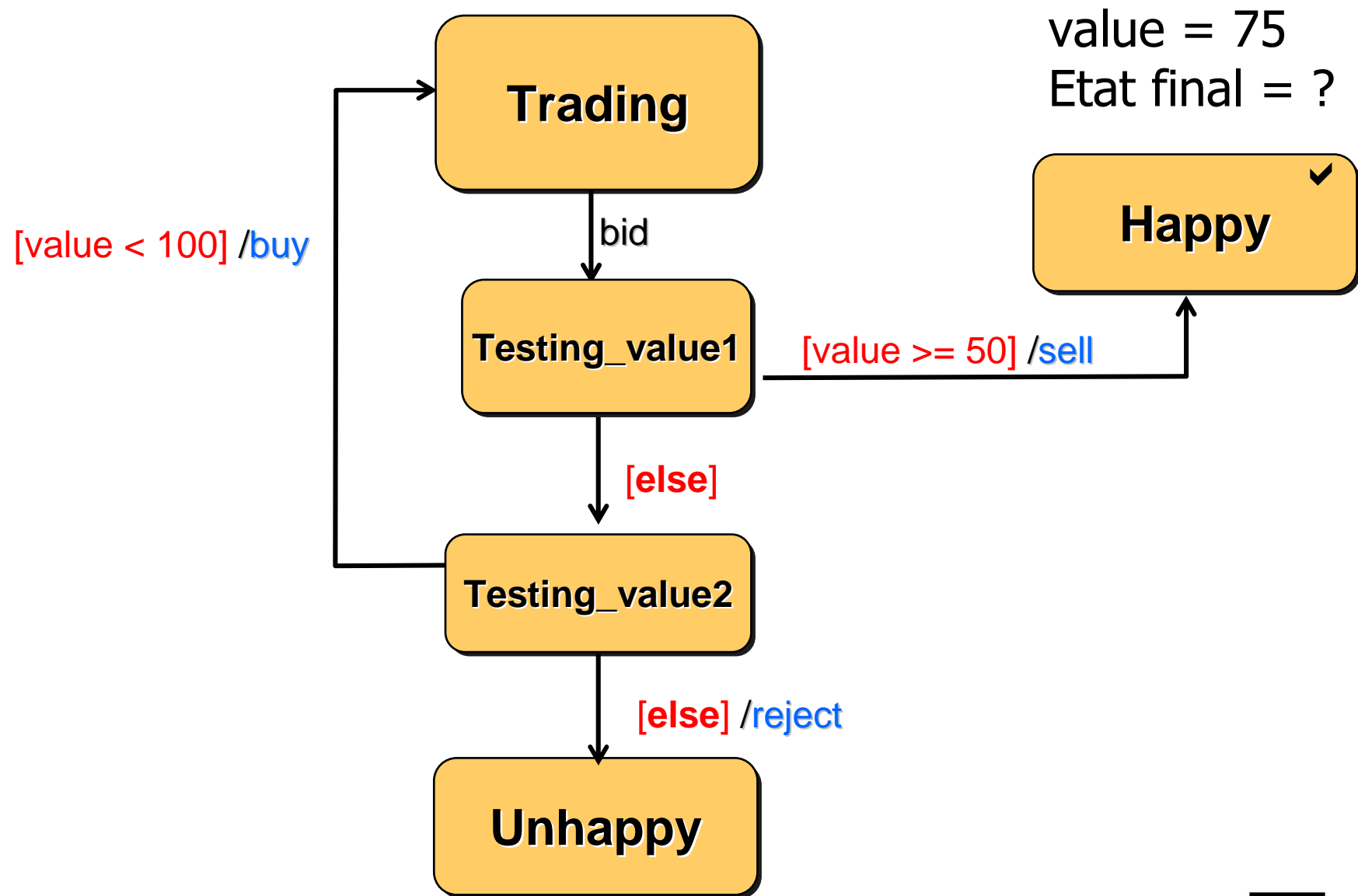
value = 75  
Etat final = ?



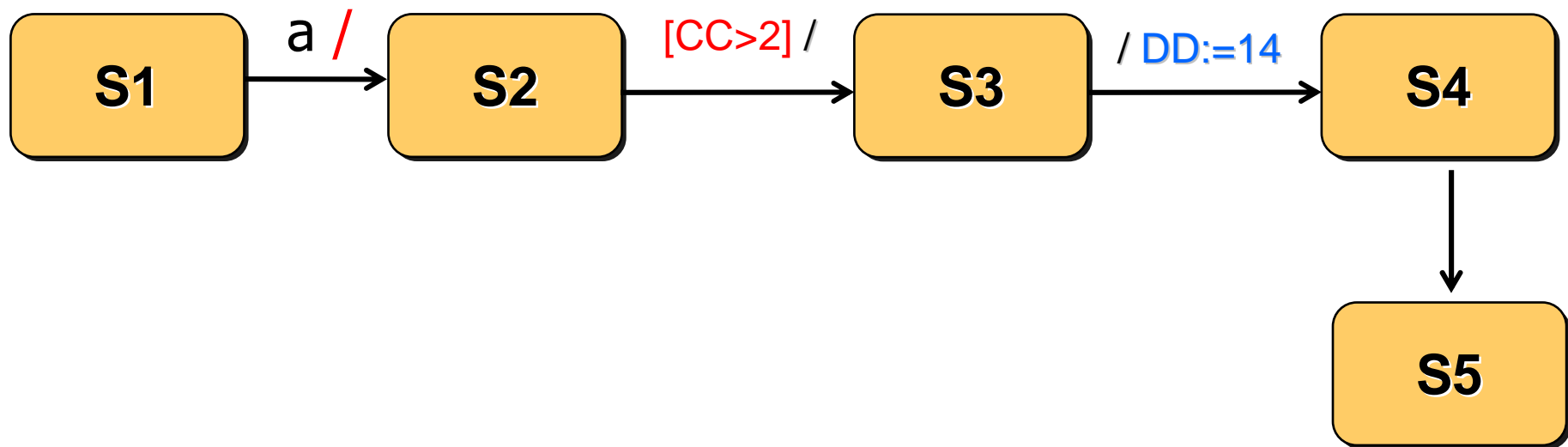
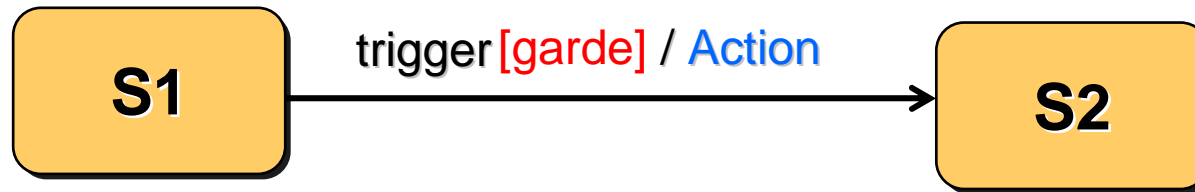
# Transition et tests des gardes en cascade



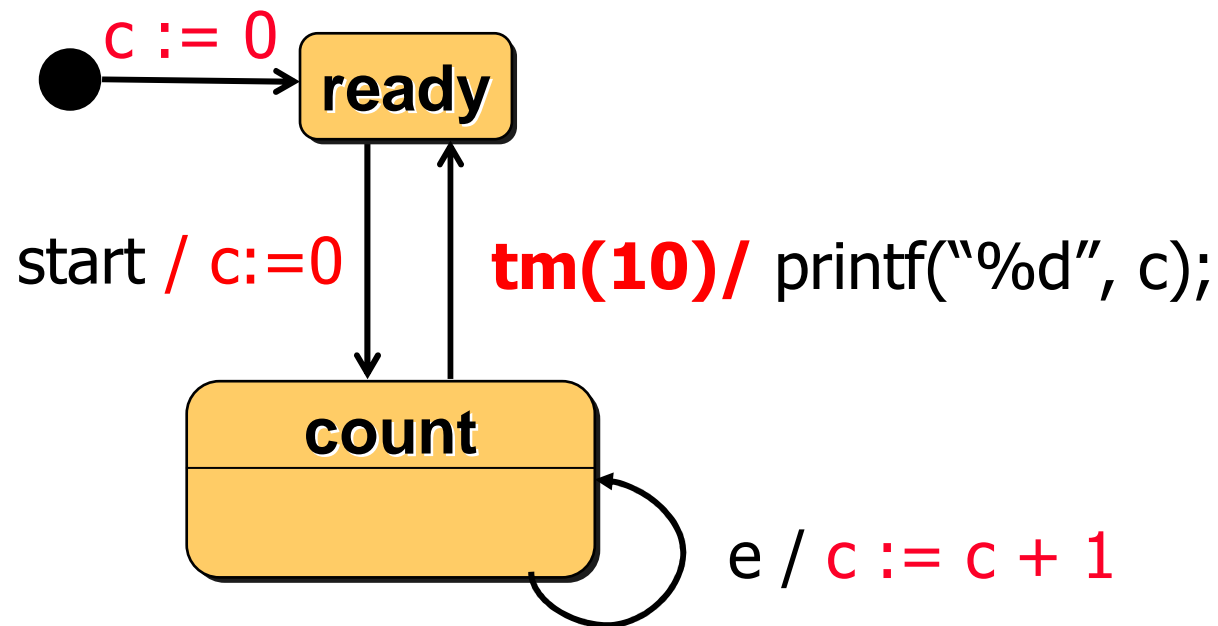
# Transition et tests des gardes en cascade



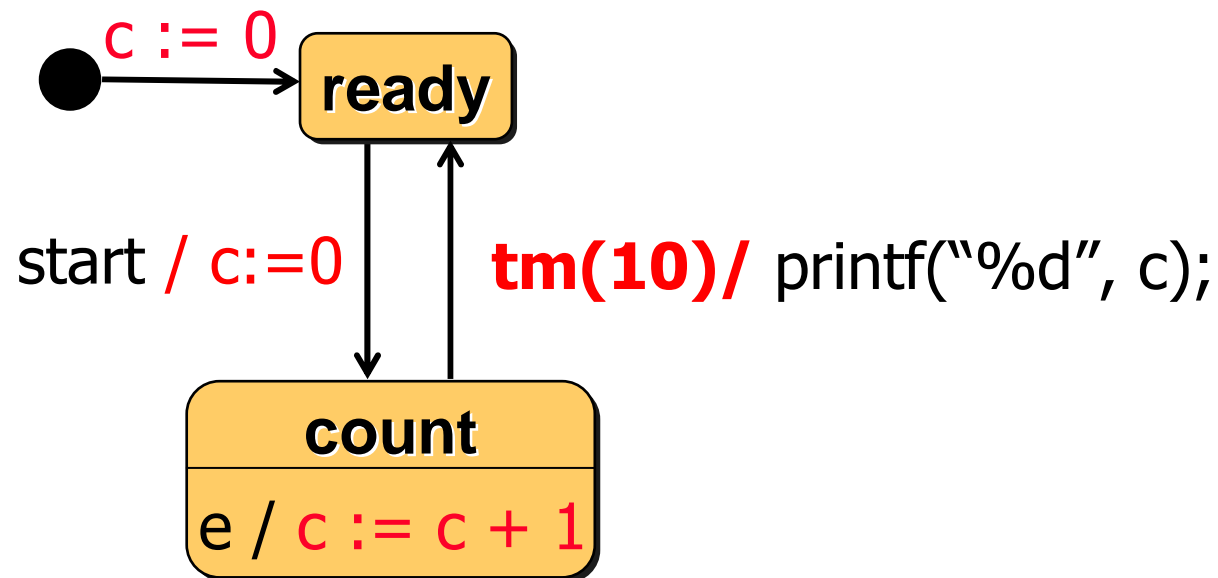
# Syntaxe des transitions



# Modélisation du temps

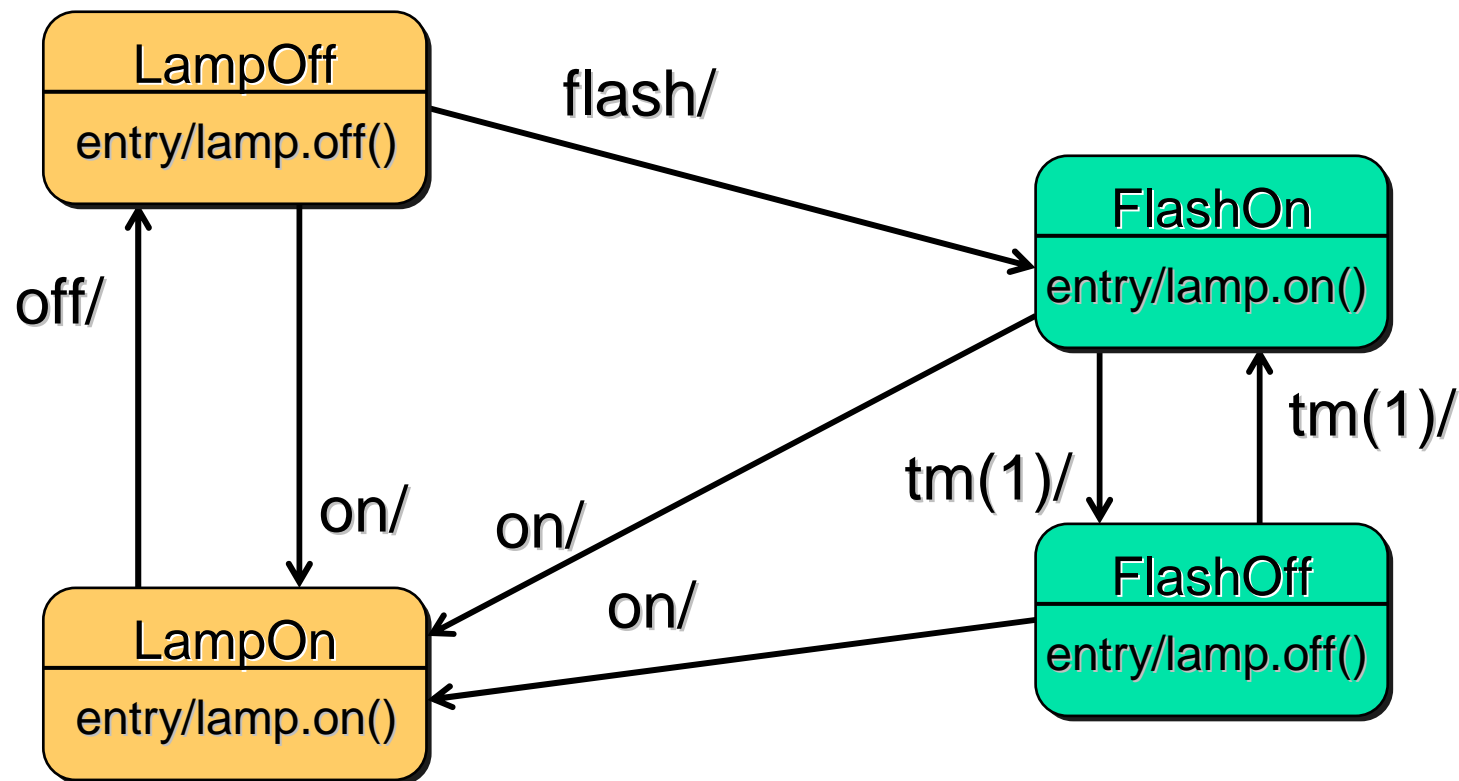


# Modélisation du temps

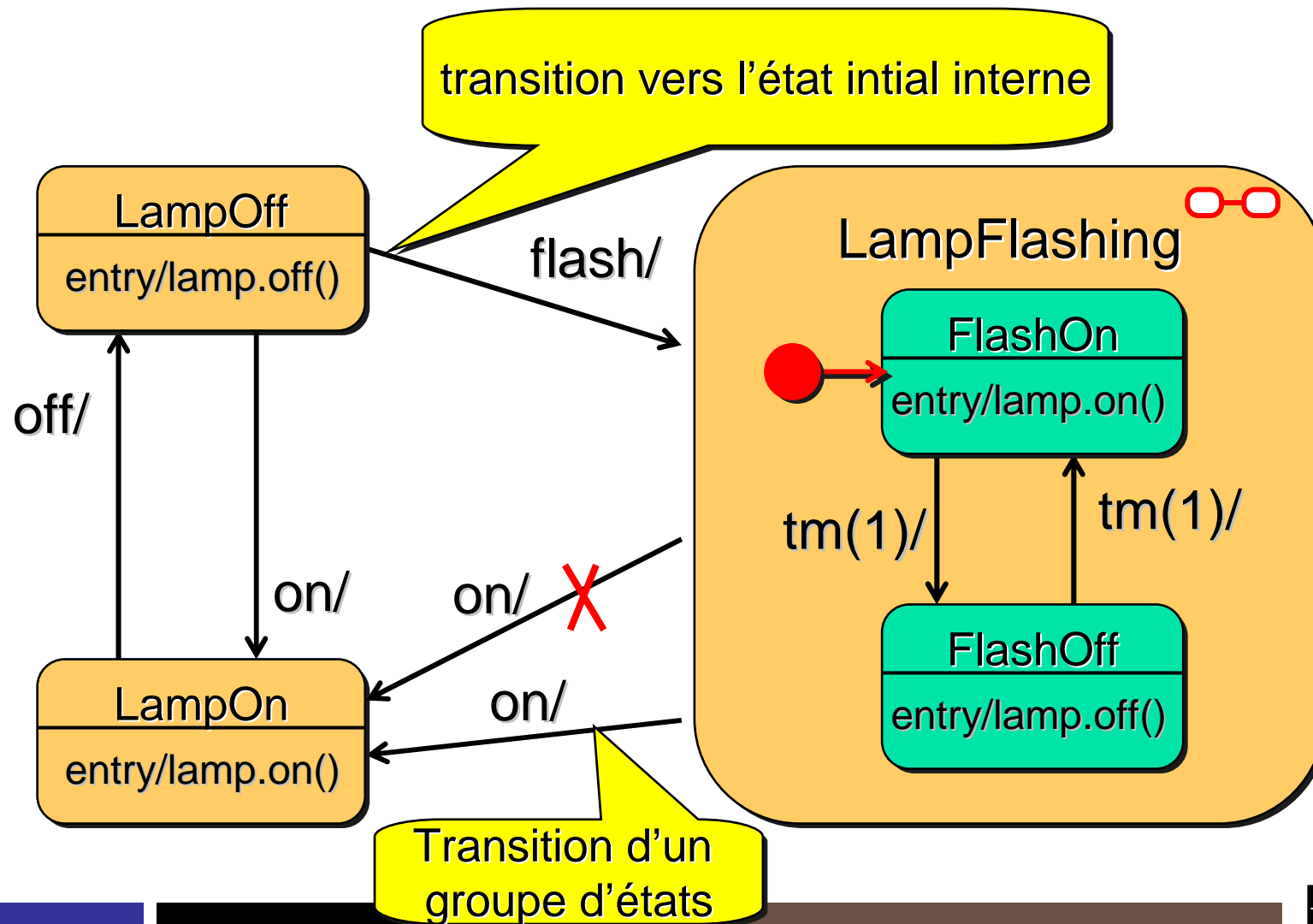


# Machines à Etats Hierarchiques

- Réduction de la complexité
- états décomposés en machines à états
- n'augmente pas le pouvoir d'expression

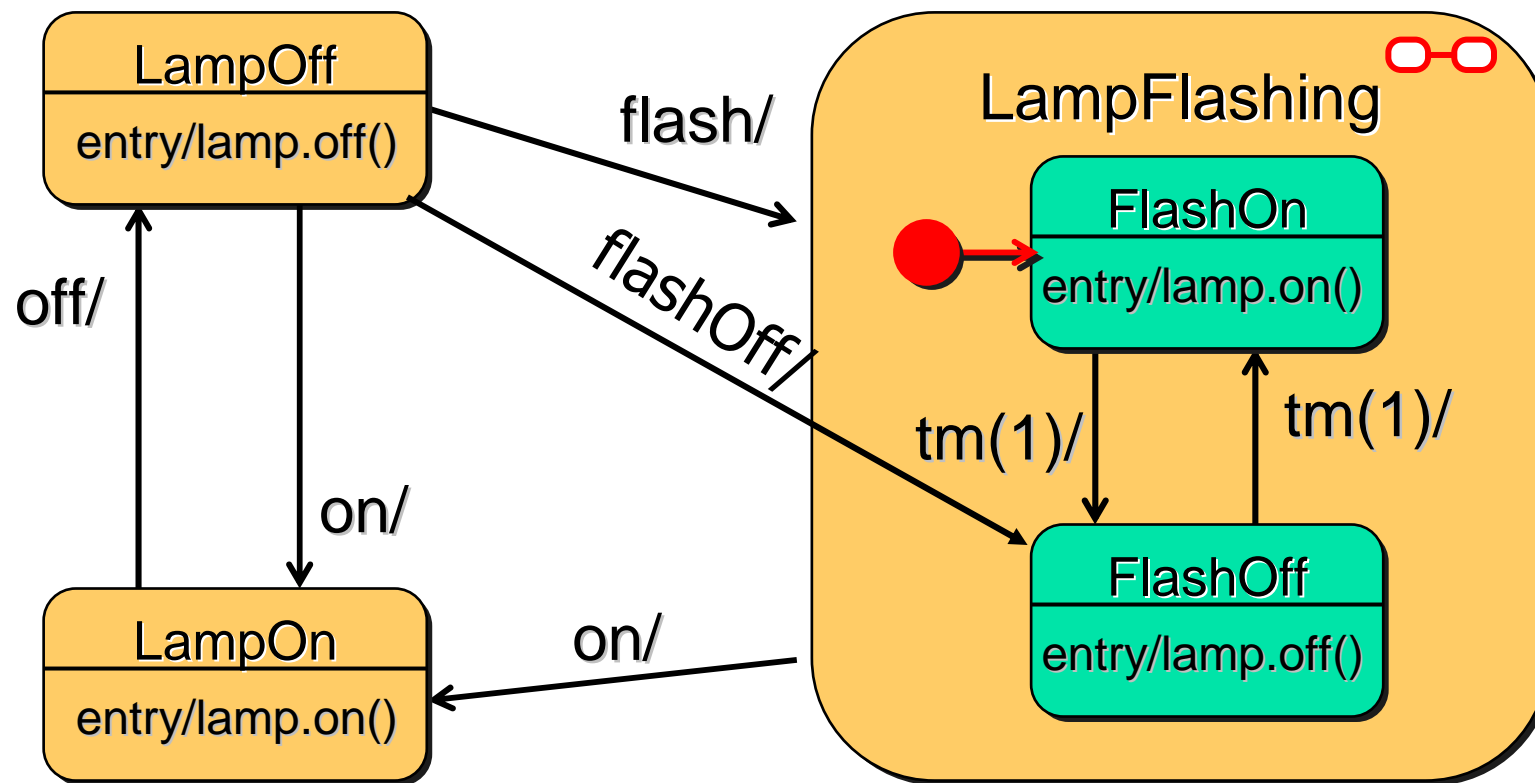


# Machines à Etats Hierarchiques



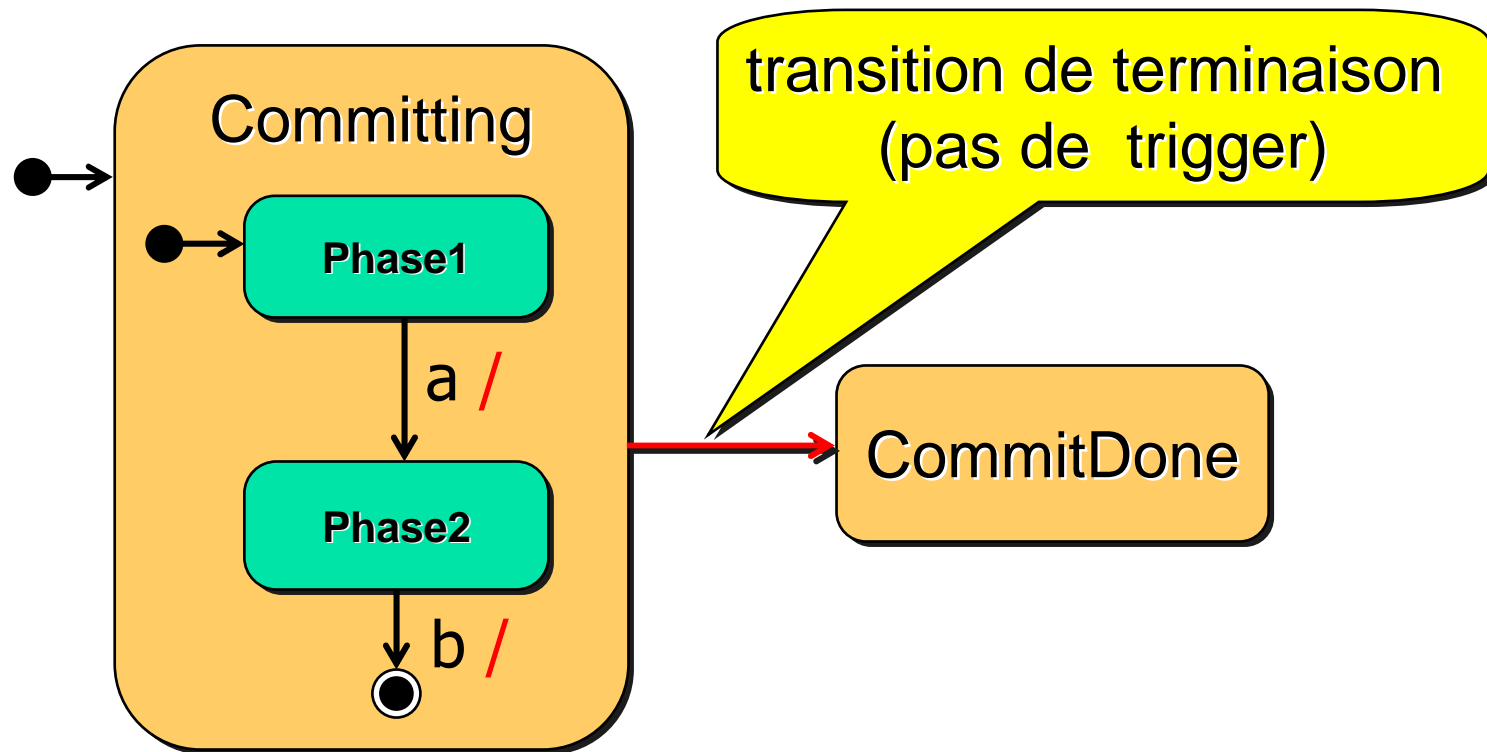


# Machines à Etats Hierarchiques



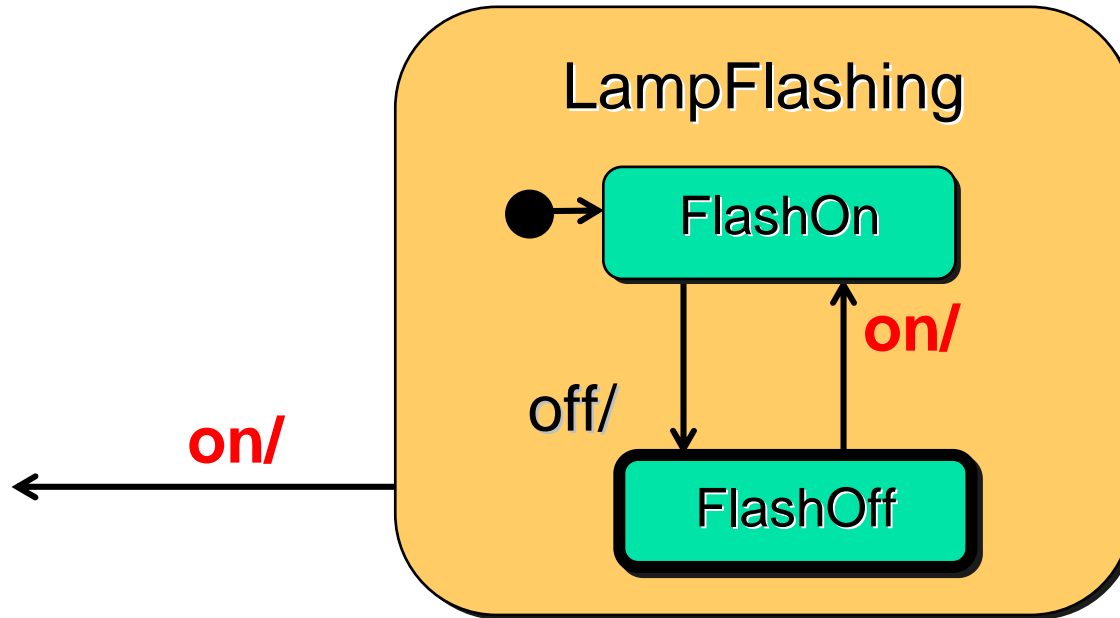
# Transitions de terminaison

- Déclenchée automatiquement dès l'arrivée dans un état terminal



# Règles de déclenchement

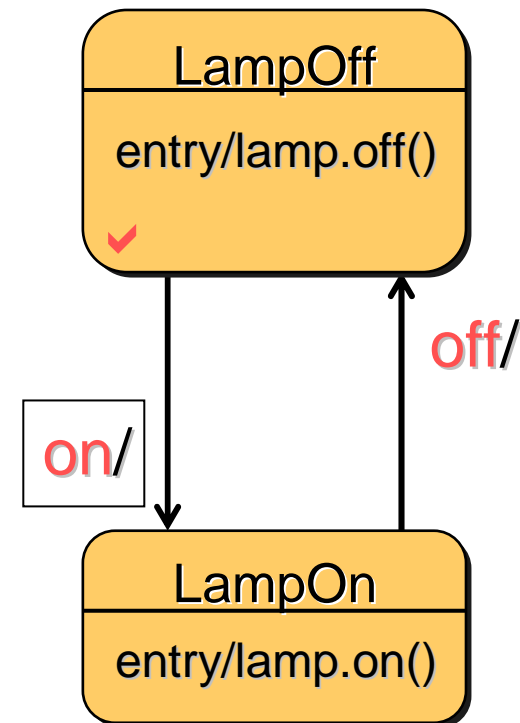
- Deux transitions (ou plus) peuvent avoir le même évènement :
  - Précédence à la transition la plus interne



# Evènements différés (1)

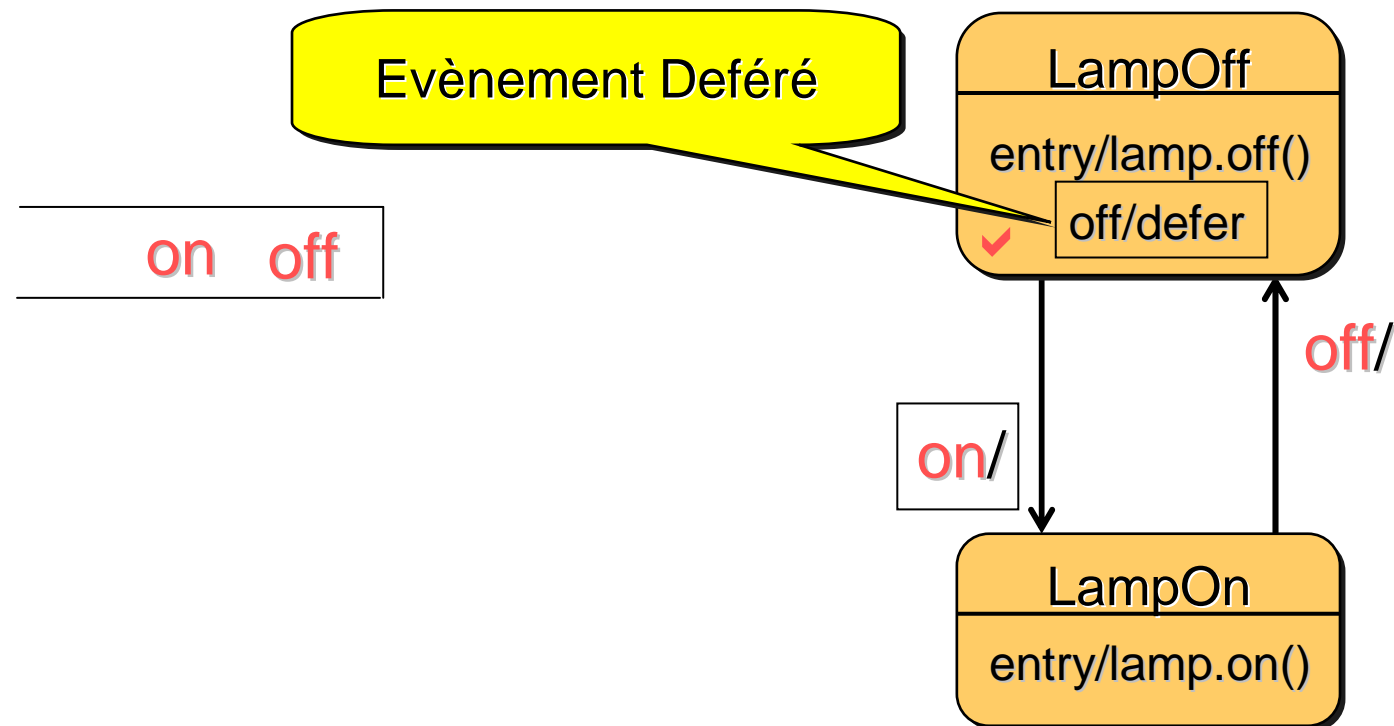
- Les évènements en attente sont supprimés s'il n'y a pas de transition correspondante

on off

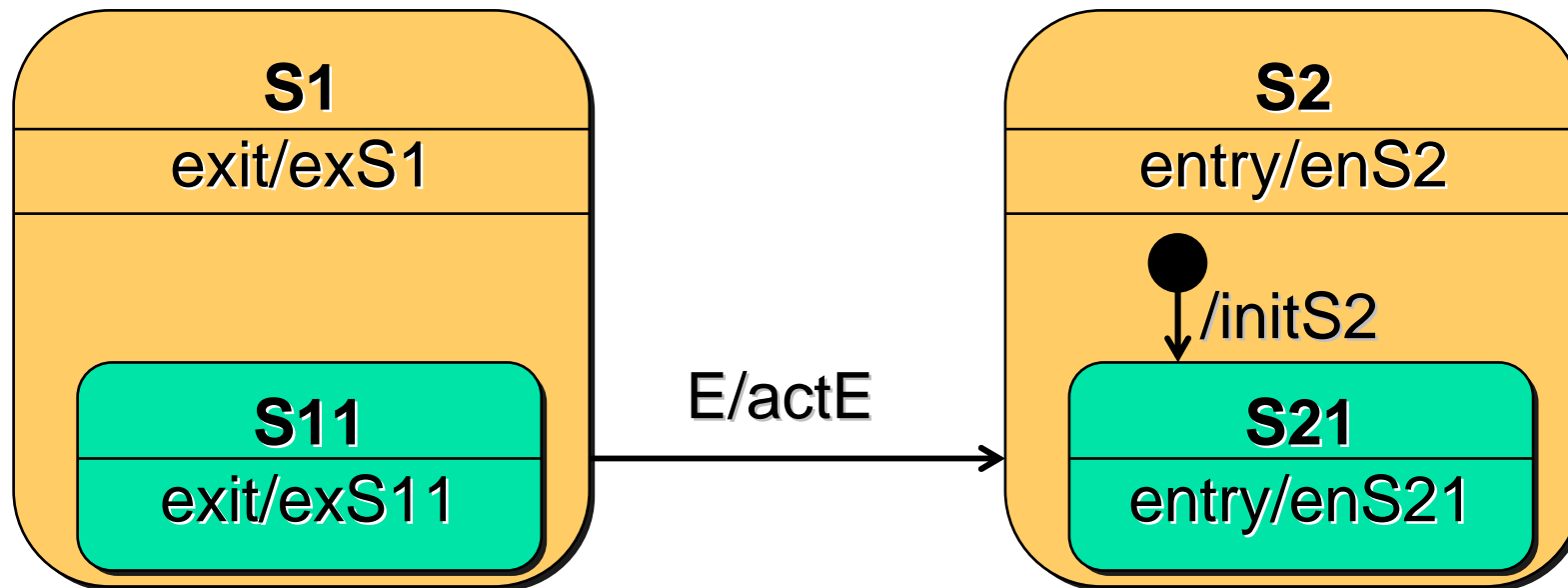


# Evènements déferés (2)

- A moins qu'ils ne soient déclarés comme "deferred" dans l'état



# Ordre des Actions: Cas Complexe



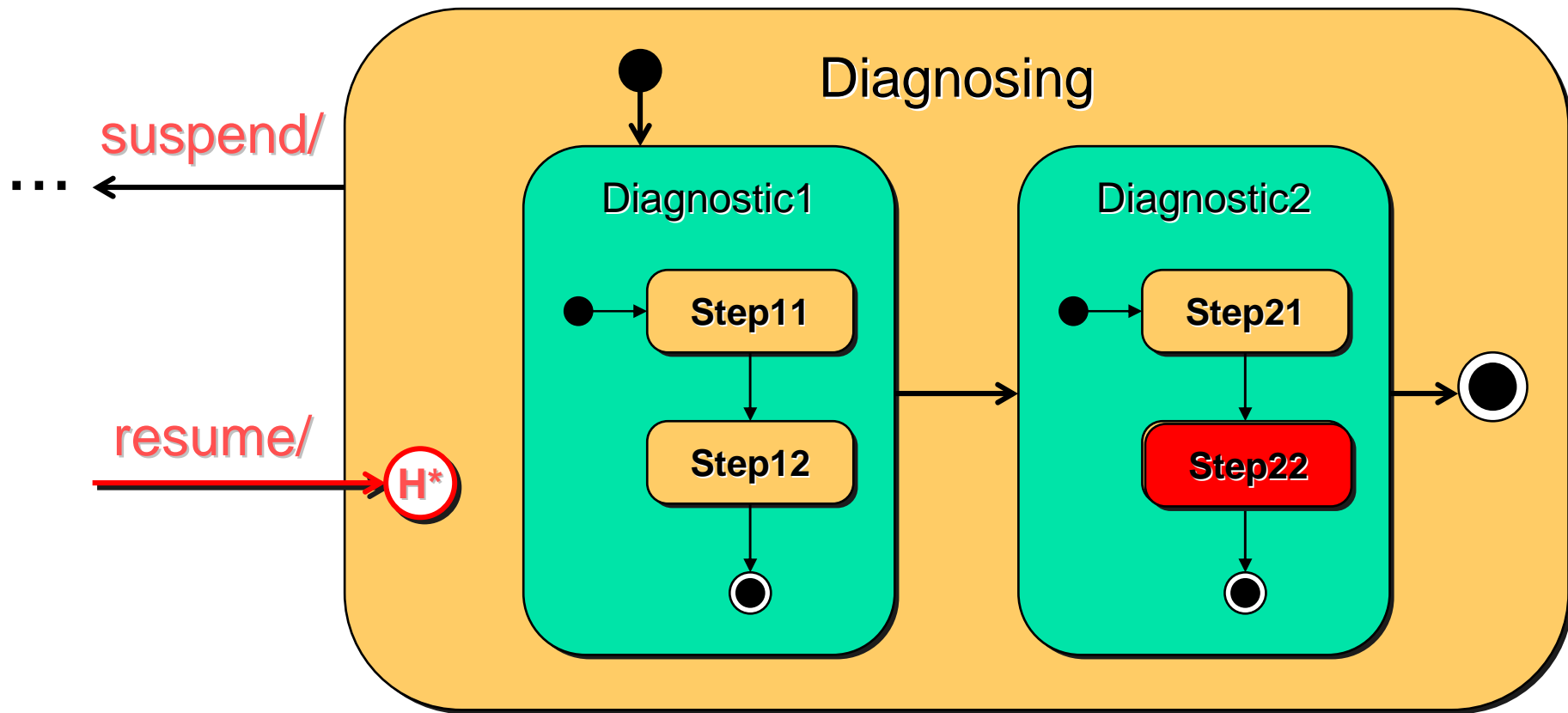
**Séquence d'exécution des Actions :**

`exS11`  $\Rightarrow$  `exS1`  $\Rightarrow$  `actE`  $\Rightarrow$  `enS2`  $\Rightarrow$  `initS2`  $\Rightarrow$  `enS21`

# Histoire

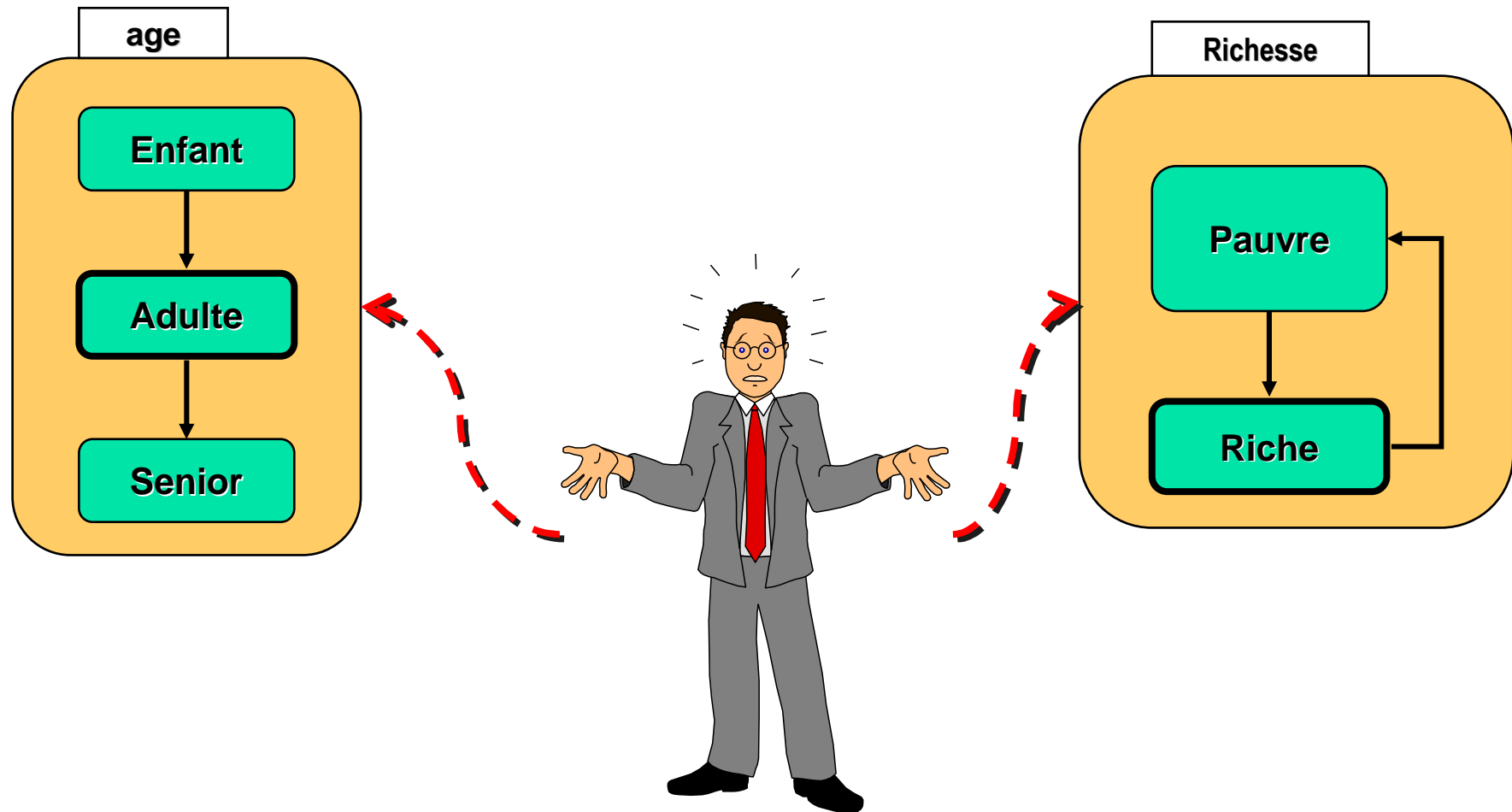
- Permet de revenir au dernier état visité

suspend    resume



# Orthogonalité

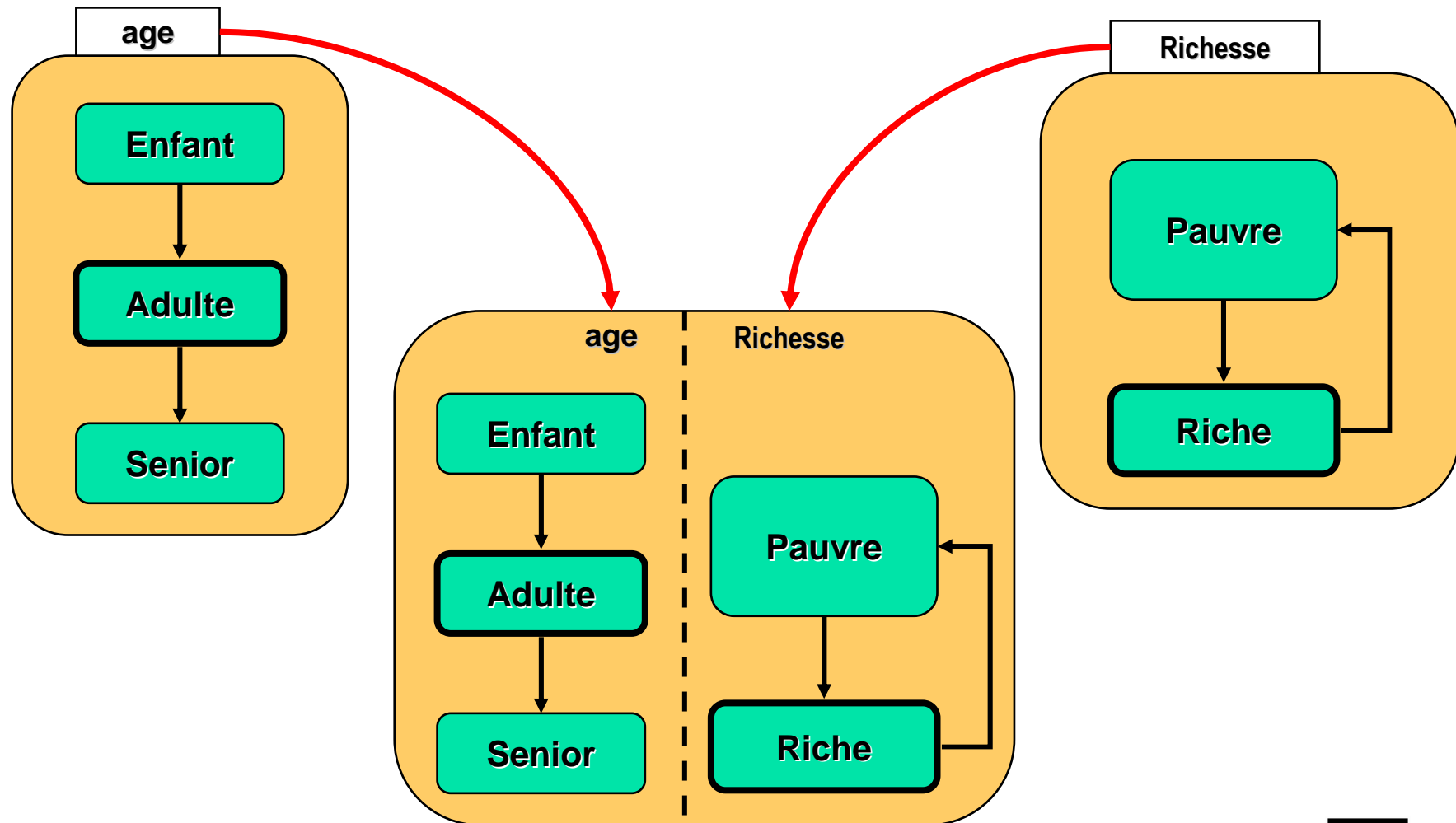
- Permet de modéliser des vecteurs d'états





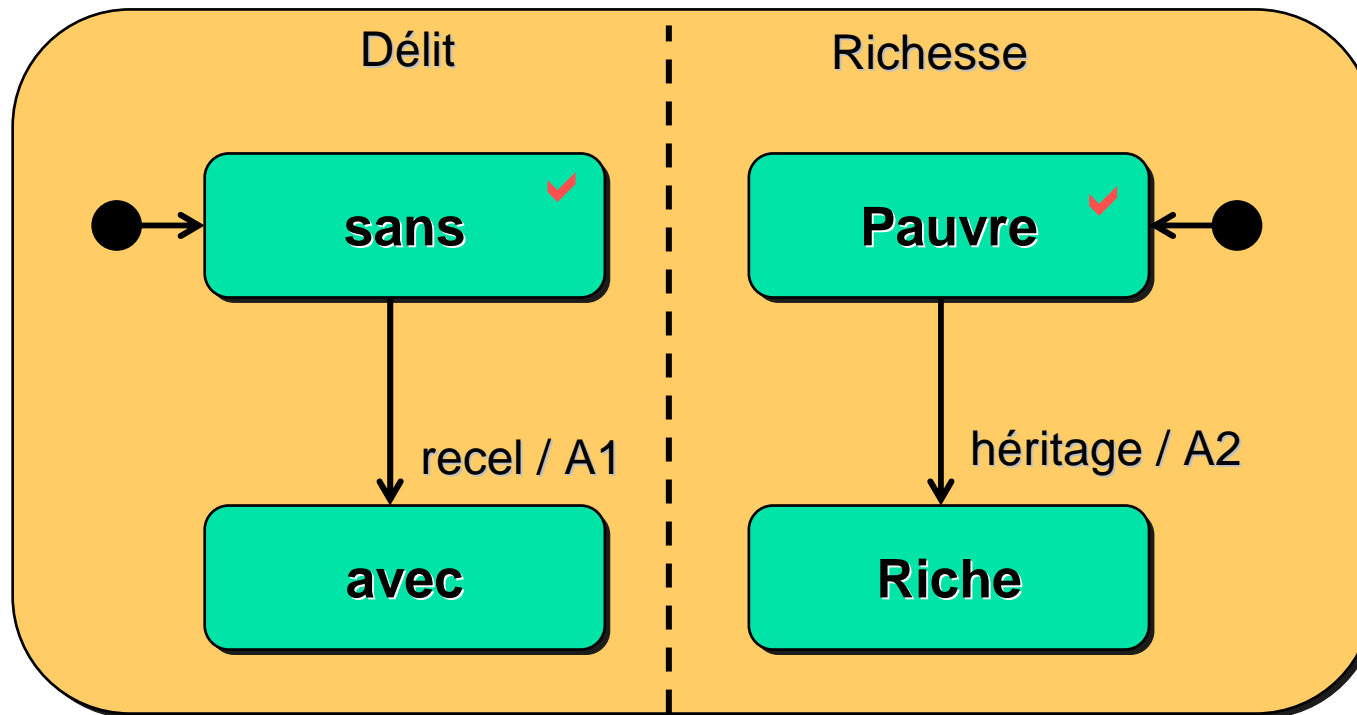
# Régions Orthogonales

- Etat → Couple d'états jointifs (produit cartésien)



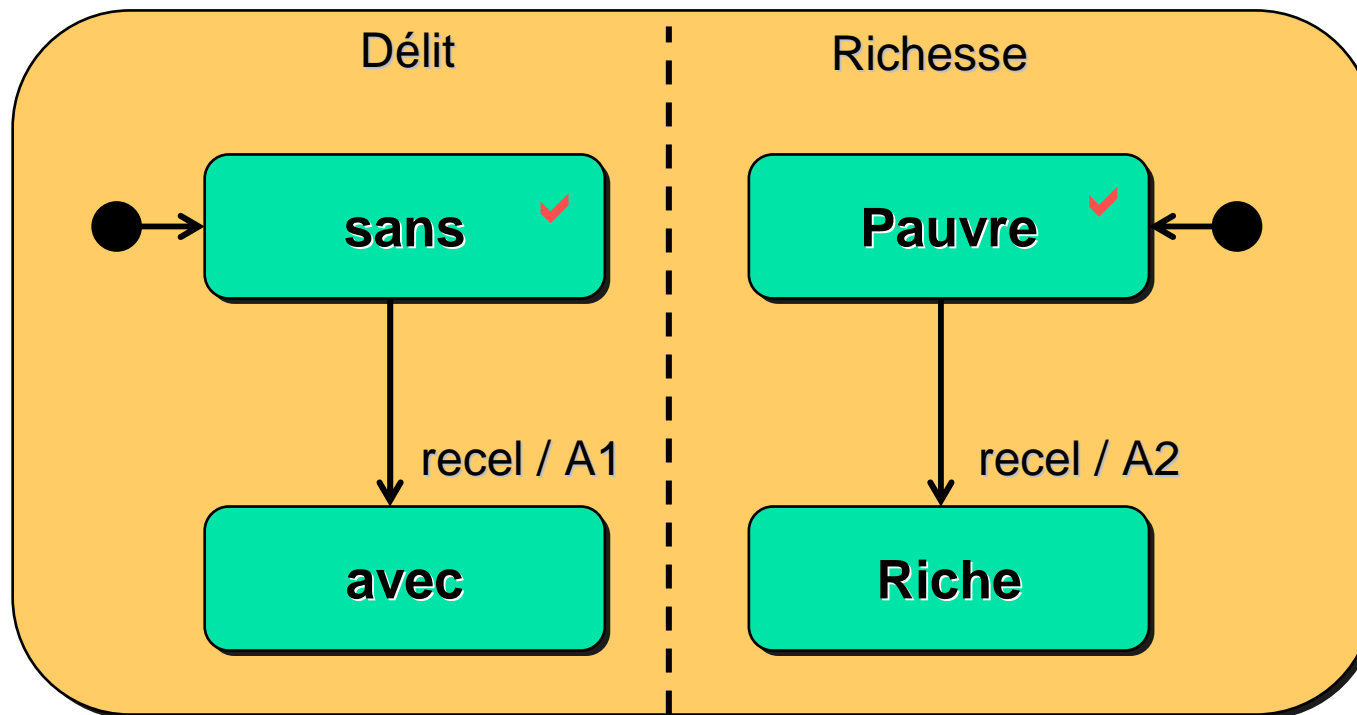
# Semantique des Régions Orthogonales (1)

- Chaque région peut détecter et exécuter un évènement indépendamment des autres régions

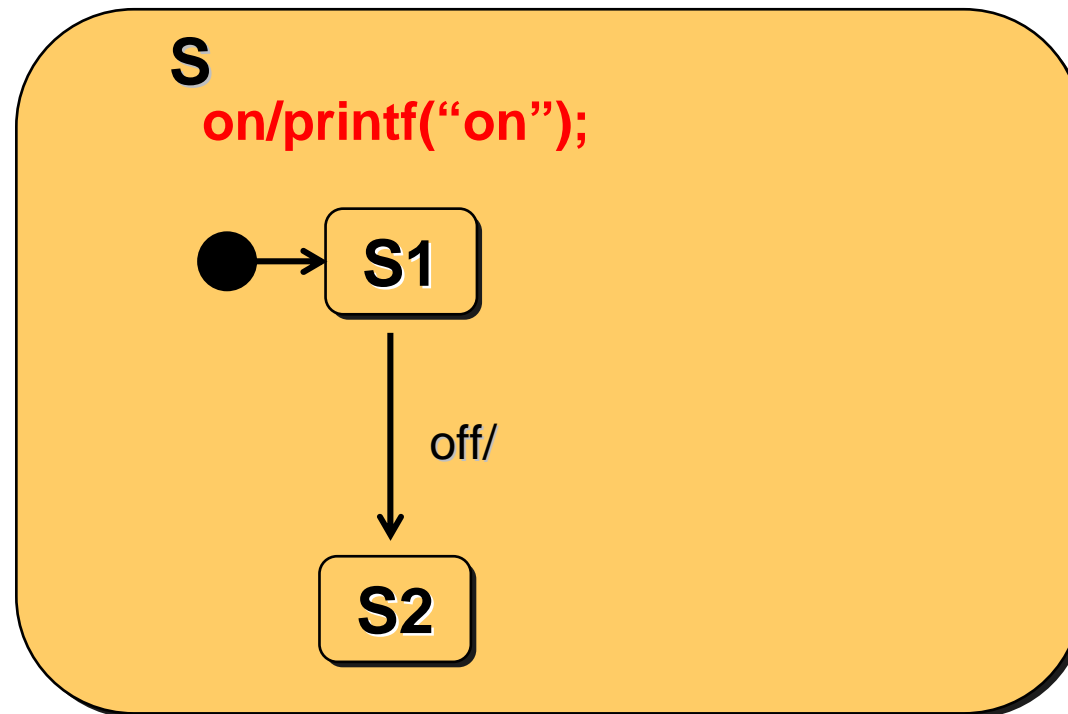


# Semantique des Régions Orthogonales (2)

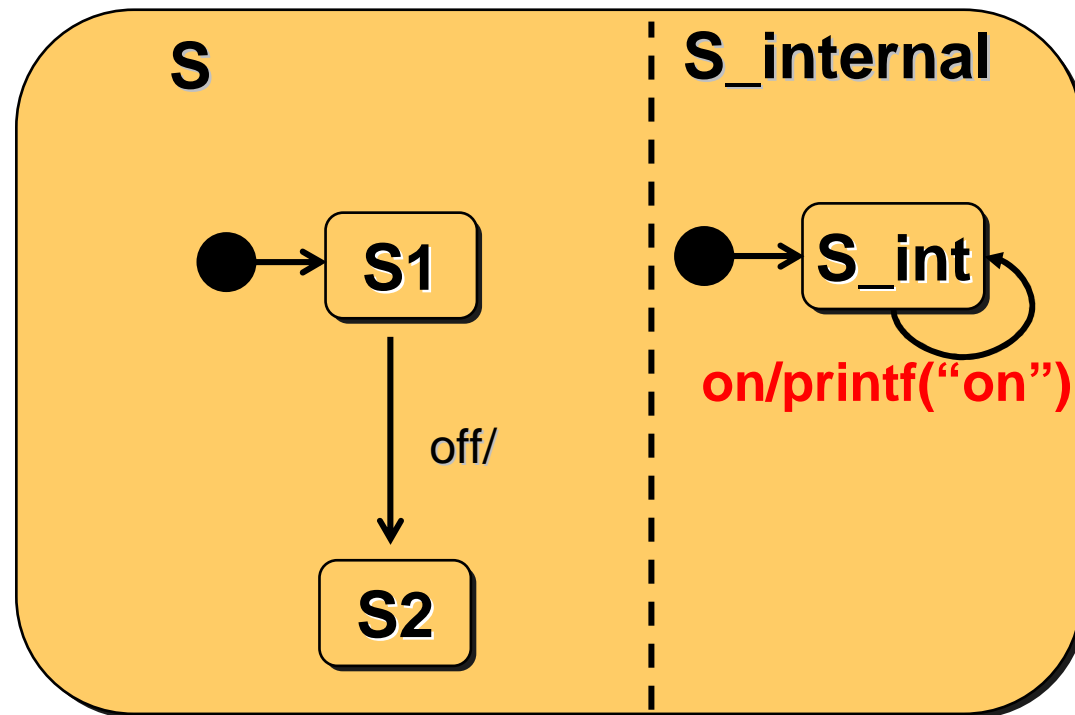
- Toutes les régions peuvent détecter et exécuter simultanément le même évènement
- L'exécution est entrelacée (non forcément déterministe)



# Transitions Internes et Orthogonalité

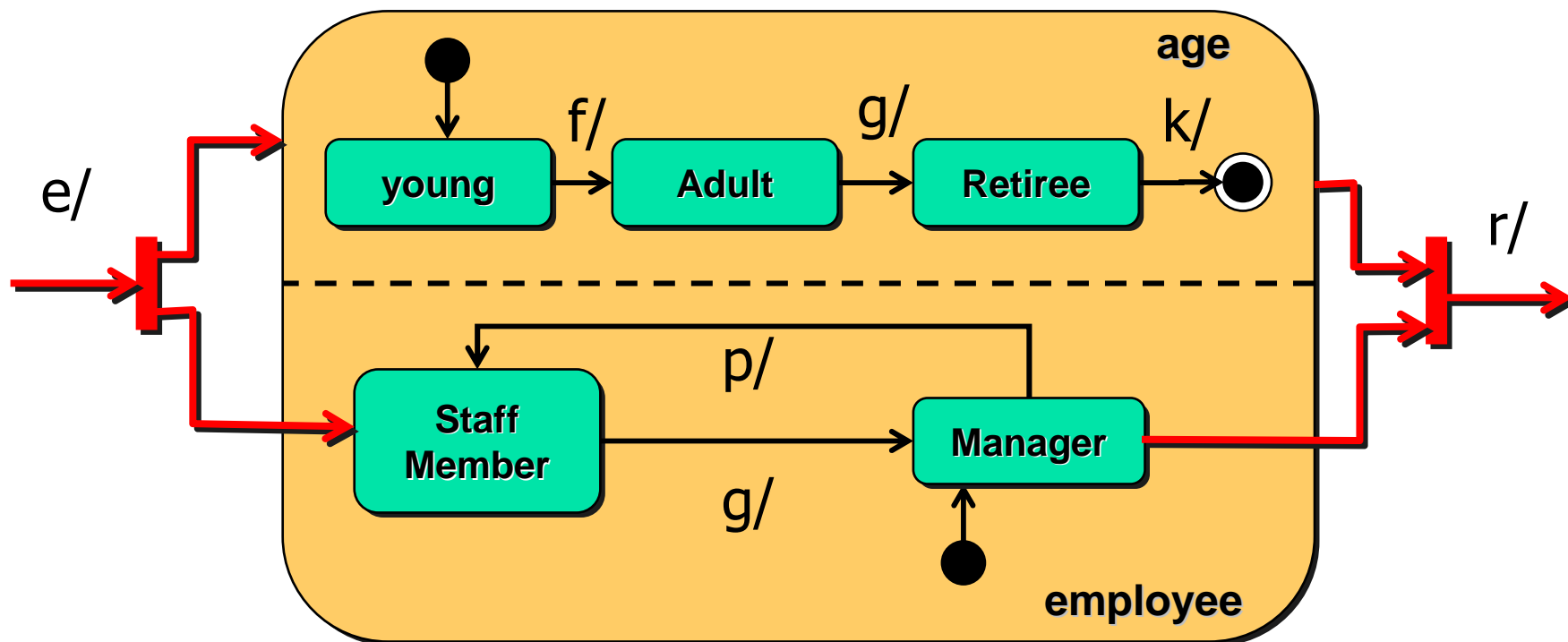


# Transitions Internes et Orthogonalité



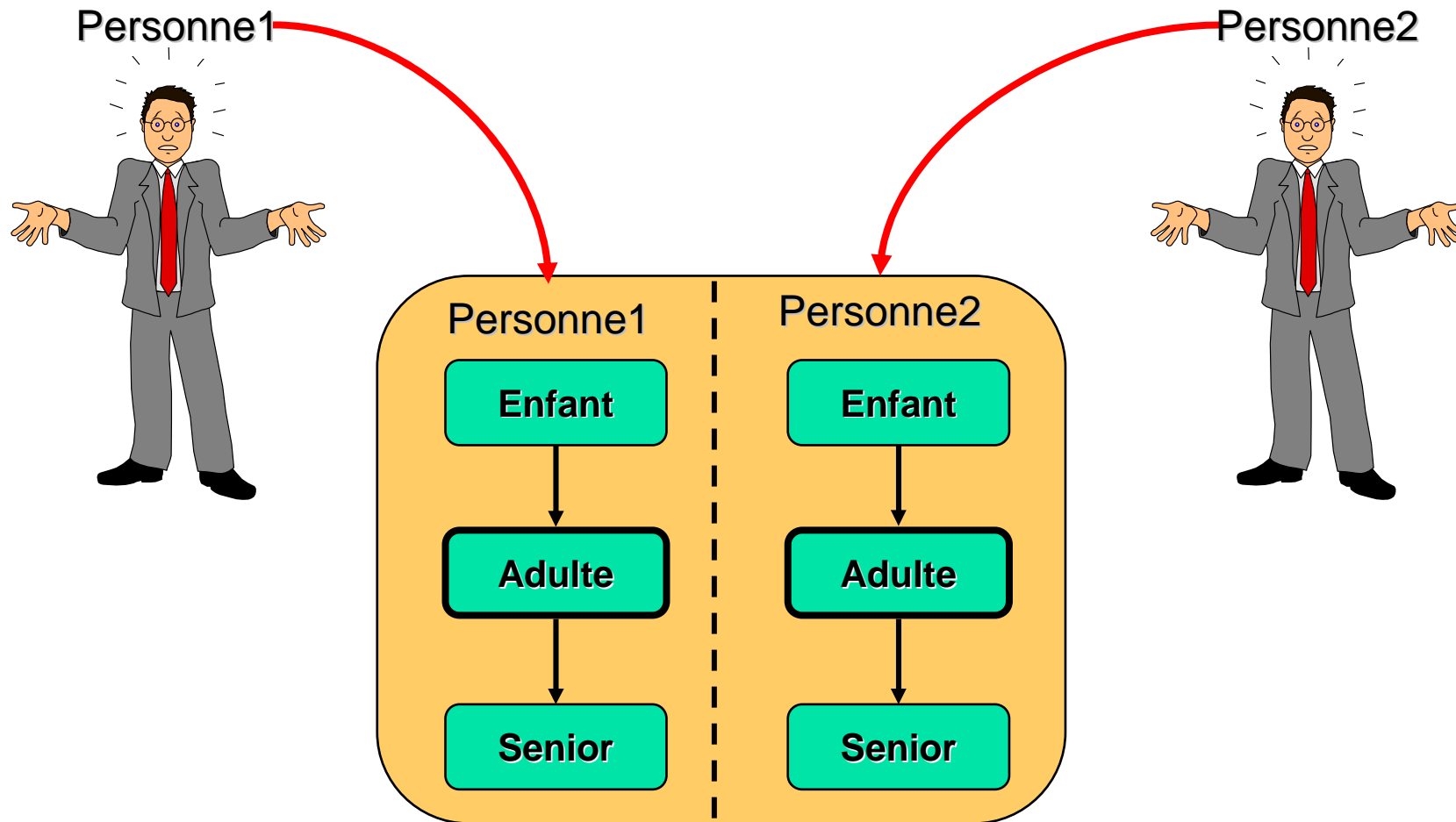
# Transition Fork et Join

- Pour entrer et sortir des régions orthogonales :





# Du mauvais usage de l'Orthogonalité

- régions pour modéliser des instances différentes



# Comportement, MàE et Classes

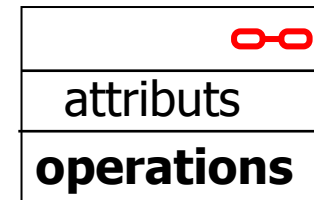
Classes: {  
 ■ Avec ou sans Machine à Etat  
 ■ Avec ou sans Thread de Contrôle

MàE Thread Control	sans	avec
sans  (Passives)	<div><div></div><div>attributs</div><div>operations</div></div>	<div><div></div><div>attributs</div><div>operations</div></div> Réactive
avec  (Actives)	<div><div></div><div>attributs</div><div>operations</div></div>	<div><div></div><div>attributs</div><div>operations</div></div> Active

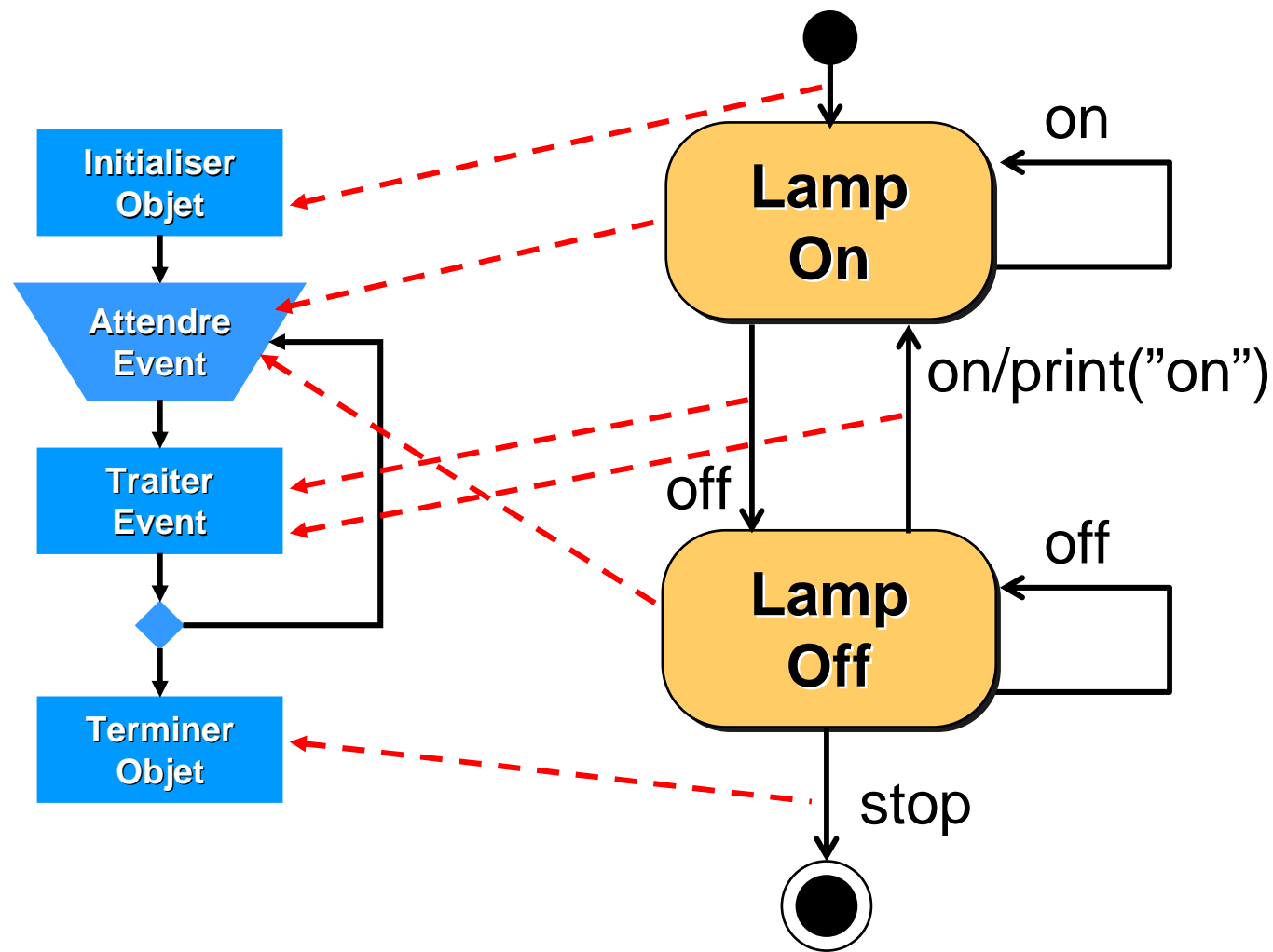


# Propriétés des Classes Réactives

- Un objet instance d'une classe réactive :
  - Démarre et devient disponible à l'instant de sa création
  - Reste réactif jusqu'à ce que :
    - Le comportement atteint un état final
    - Il est terminé par un autre objet
  - Ne possède pas de boîte à lettre : il réagit à la réception d'opération dans le contexte du thread de l'appelant
  - 2 types d'opérations :
    - Primitives : l'opération (et son corps) est déclarée dans la classe
    - Triggered : l'opération est un trigger d'une transition de la machine à état

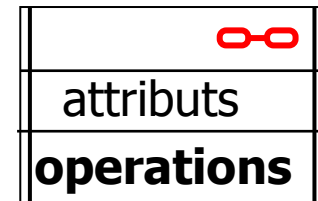


# Comportement d'un objet réactif

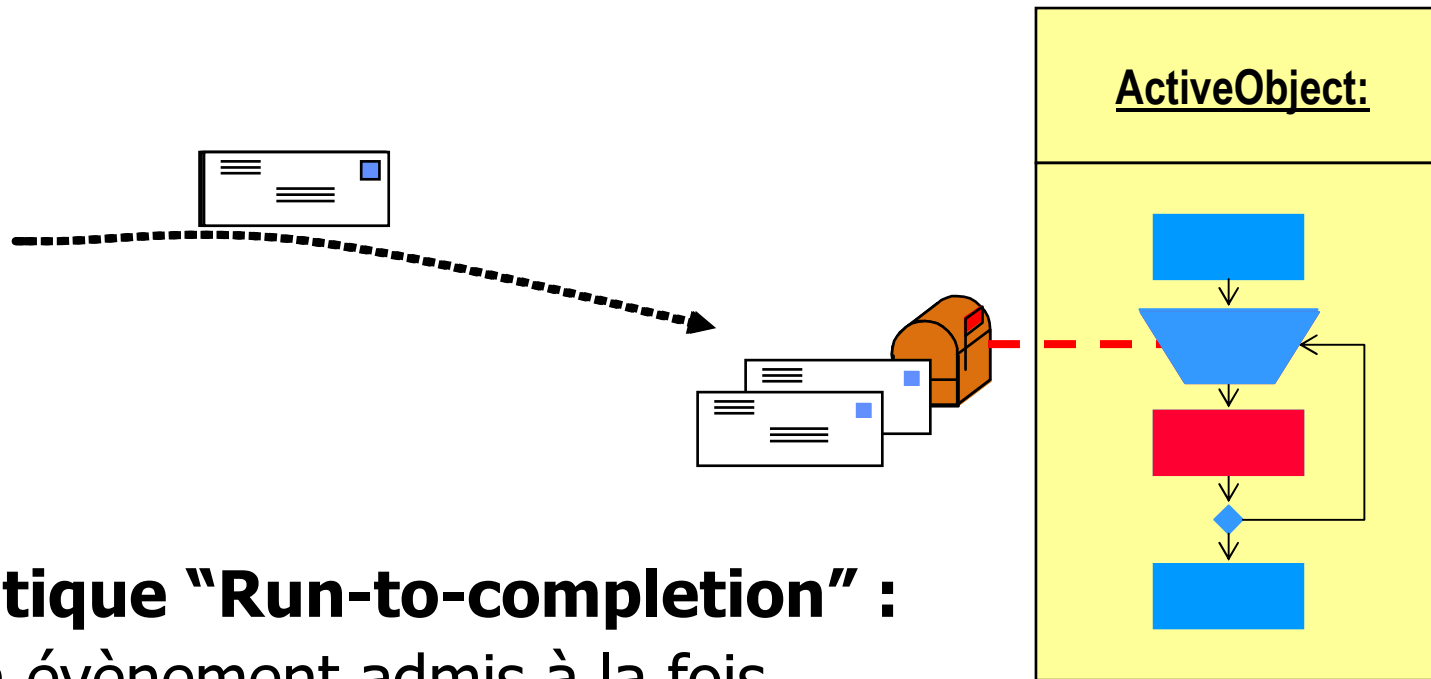


# Propriétés des Classes Actives

- Un objet instance d'une classe active :
  - Démarre et devient disponible à l'instant de sa création
  - Reste actif jusqu'à ce que :
    - Le comportement atteint un état final
    - Il est terminé par un autre objet
  - **Dispose d'un thread de contrôle :**
    - possède une boîte à lettre pour recevoir les signaux
    - les signaux sont déposés dans la boîte à lettre
    - chaque évènement est traité jusqu'à complétion avant de considérer l'évènement suivant
  - 2 types d'opérations :
    - Primitives : l'opération (et son corps) est déclarée dans la classe
    - Triggered : l'opération est un trigger d'une transition de la machine à état



# Objets Actifs : Semantique Dynamique



## Sémantique “Run-to-completion” :

- Un évènement admis à la fois
- Exécution d’une (séquence de) transition(s) qui se termine sur un état ne possédant pas de transition exécutable

## Avantages :

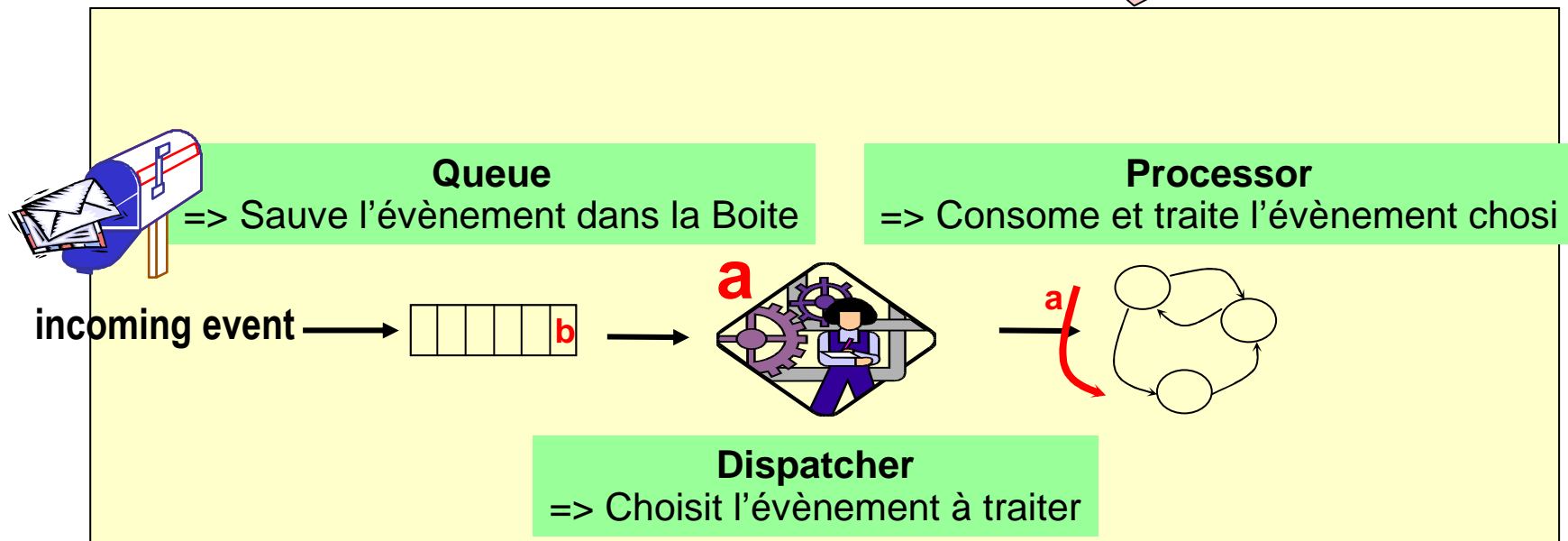
- Evite la complexité due à la concurrence interne
- Réduction de la surcharge due à la gestion des contextes

# Sémantique des machines à états de UML

Un objet actif accomplit les fonctions suivantes :

- ✓ Sauve les évènements dans sa file
- ✓ Dispatche (choisit) l'évènement
- ✓ Traite l'évènement

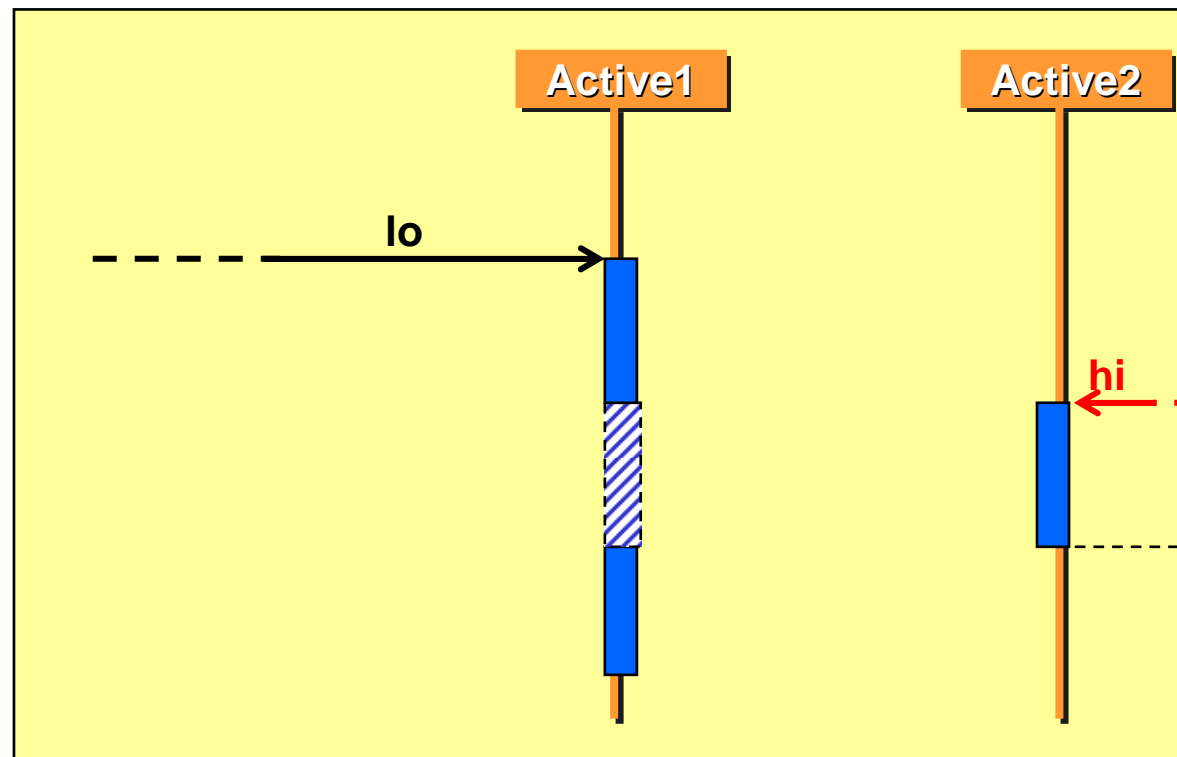
**Traitement jusqu'à  
terminaison :  
Un évènement à la fois**



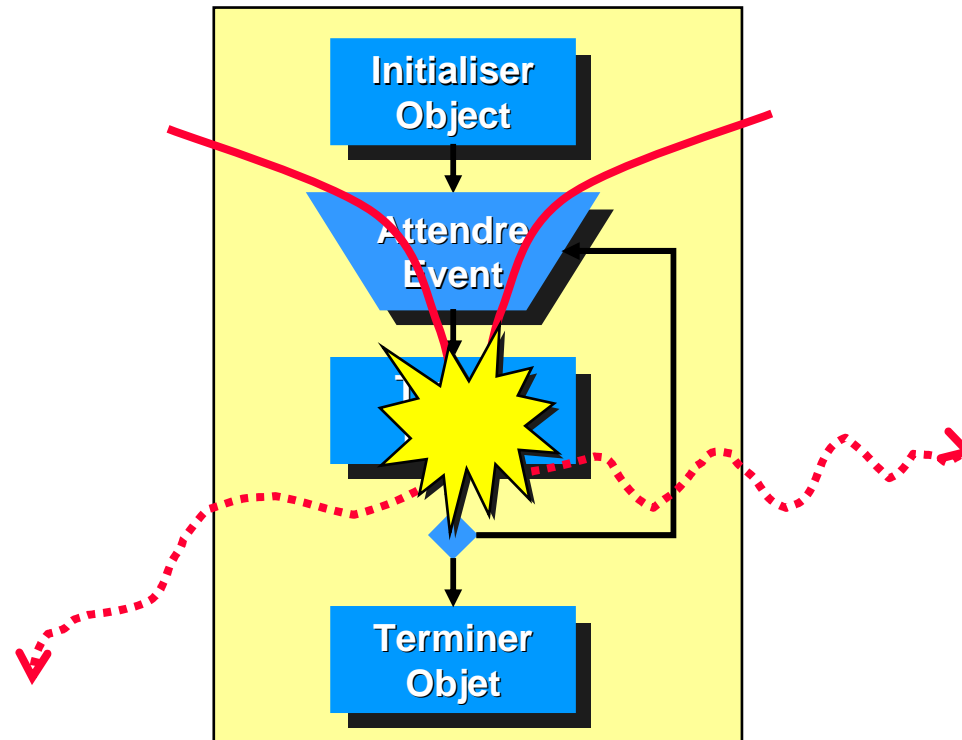
– FIFO la plus couramment utilisée dans les outils

# Les objets actifs peuvent être suspendus

Un objet actif peut être suspendu par un événement déclenchant un autre objet actif de plus haute priorité

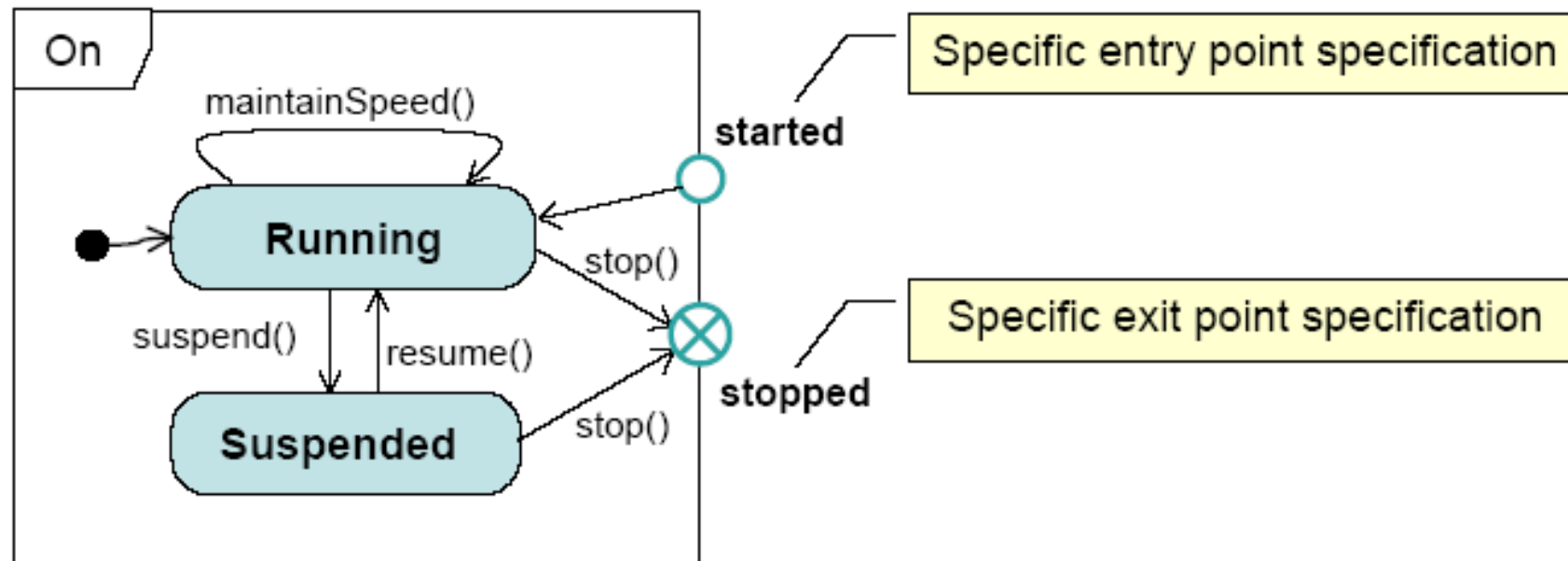


# Objets passifs : accès concurrents



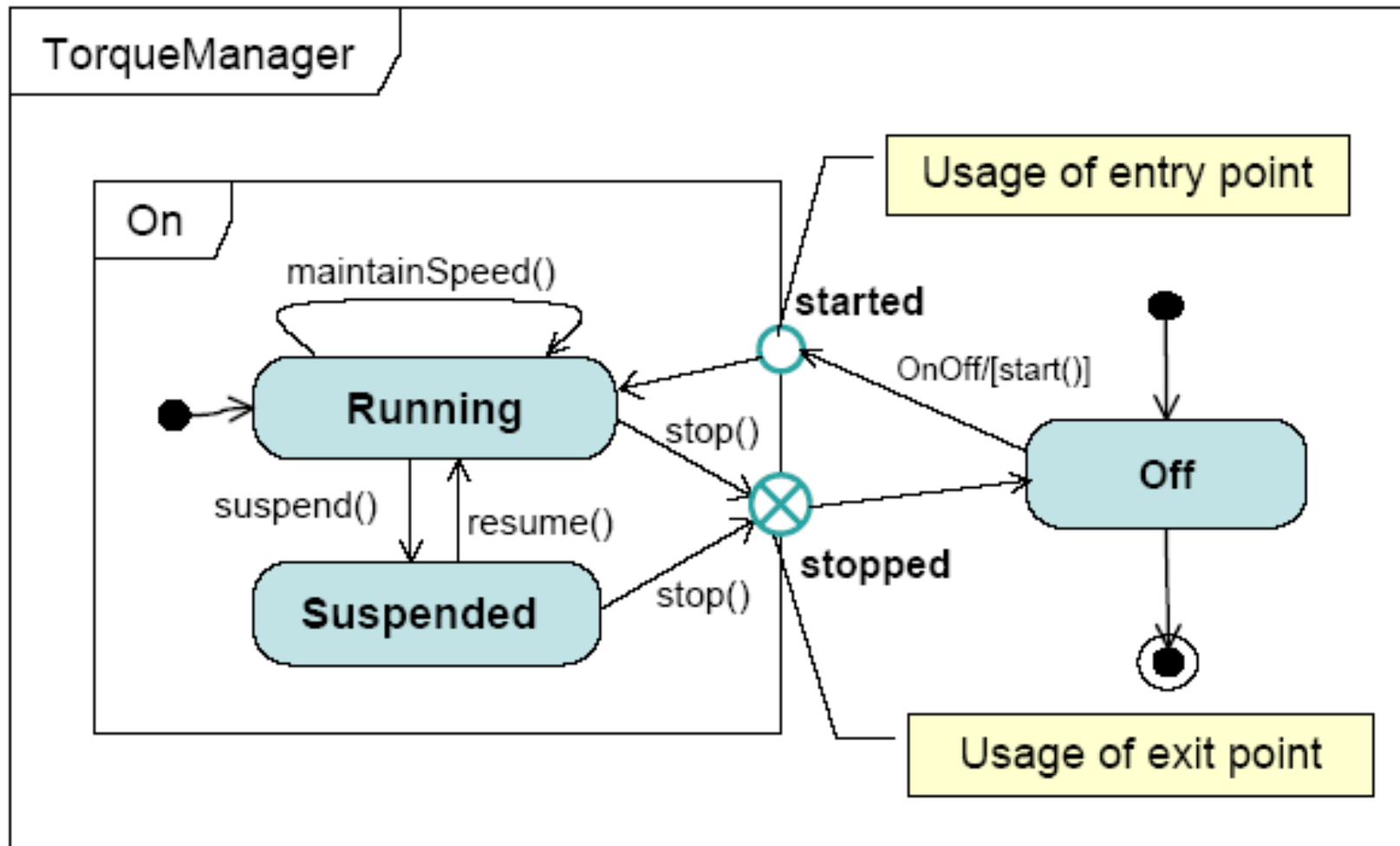
- L'encapsulation ne protège pas des accès concurrents
- Une synchronization explicite est nécessaire

# Extensions UML.2





# UML.2 extension

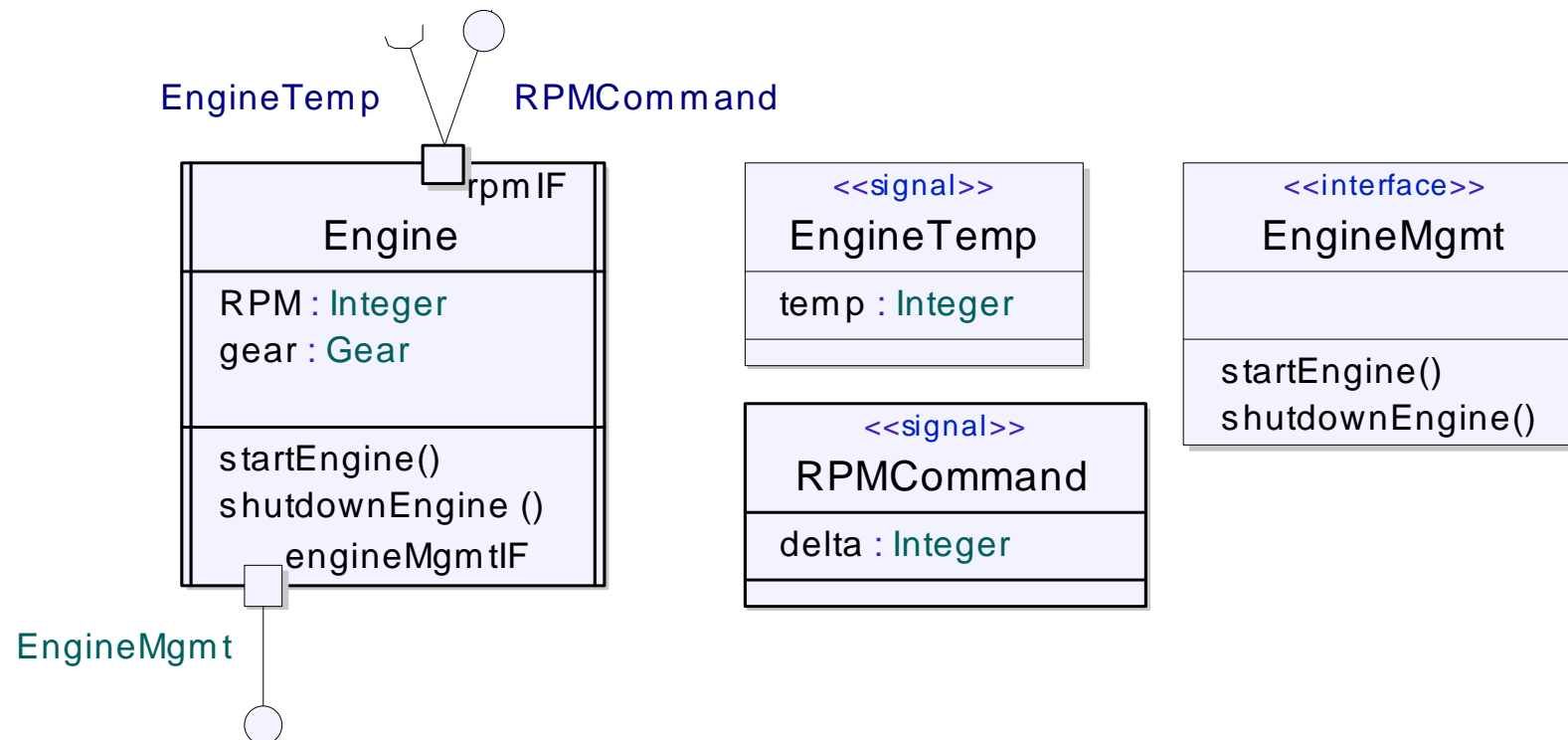


# Les Ports

---

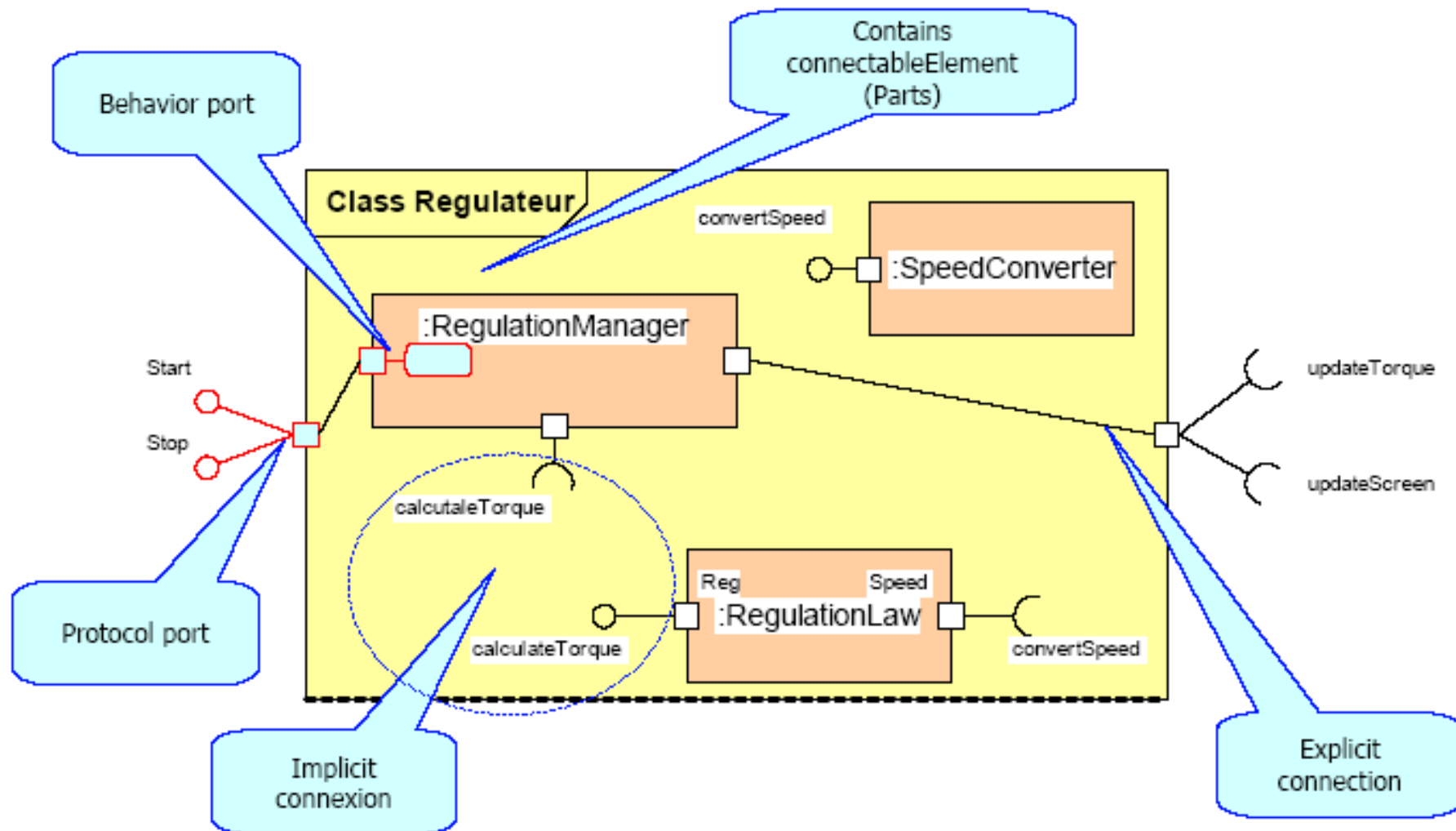
- Un port est un point d'interaction qui peut être ajouté à une classe (ou un objet)
- Sur un port, sont spécifiés :
  - Les services (opérations et signaux) fournis
  - Les services (opérations et signaux) requis
- Les ports des classes différentes peuvent être interconnectés. Les interconnexions matérialisent les possibilités d'interactions entre objets
- Propriété des ports
  - La navigation à travers les ports est bloquée
  - Uniquement les opérations et signaux déclarés dans le port peuvent le traverser

# Exemples de Ports



# Diagrammes de Classes Composites

- Agrégation forte des "parts"
- Création et Destruction des "parts"



# Classes Actives Composites

---

- Les “parts” (qui ne sont pas déclarés comme actifs) s'exécutent dans le thread de la classe composite
- La sémantique “run-to-completion” s'applique à l'exécution interne de la classe composite active :
  - Le déclenchement d'une transition dans une “part” (et sa propagation éventuelle aux autres “parts”) doit se terminer avant qu'un autre évènement externe ne puisse être considéré
  - Effet : limitation de la concurrence interne dans le diagramme composite

# MàE : quelques problèmes ou restrictions

- Restriction des mécanismes d'extraction d'évènement
  - On ne peut pas extraire plusieurs évènements ensembles
- Problème de boucle « infinie » de transitions
  - Déclenchement d'une transition par l'absence d'un événement
- Les états concurrents / action « do » introduisent du parallélisme interne aux objets
  - Signification pour les objets passifs-réactifs ?
  - Les objets actifs sont supposés n'avoir qu'une ressource d'exécution
- Le mécanisme de choix des événements n'est pas précisé
  - Le plus courant (simple) = FIFO (First In First Out)
  - Utilisation du « deferred » possible (mise en œuvre dans certains outils)

# Conclusion sur les Machines à Etats (I)

- Des constructions syntaxiques qui permettent différents styles de descriptions comportementales :
  - actions entry/exit
  - branchement conditionnel dynamique et statique
- Réduction de la complexité visuelle grâce à :
  - La hiérarchie dans les Etats et Transitions
  - L'orthogonalité
- La sémantique "run-to-completion" simplifie la gestion de la concurrence de façon significative

# Conclusion sur les Machines à Etats (II)

---

- Les Machines à Etats :
  - s'intègrent assez bien dans le paradigme de l'orienté objet d'UML
  - Introduisent la possibilité d'exprimer le comportement orienté évènement par échange de messages asynchrones
  - Tout en maintenant la possibilité d'exprimer les appels synchrones
- C'est un langage visuel à utiliser avec précaution
  - À cause de la richesse notationnelle
  - À cause des points de variation sémantiques et des choix spécifiques implémentés dans les outils