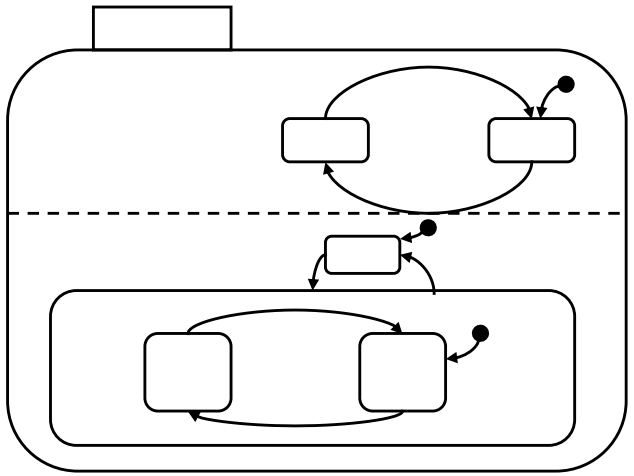
# SysML/UML Statecharts

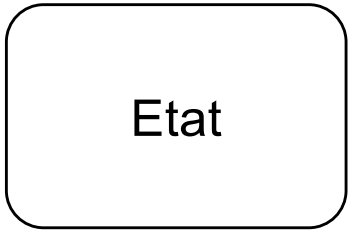
# Statecharts

- Formalisme de description du comportement de systèmes inventé par David Harel (Weizmann Institute of Science, Israël) en 1983, et intégré ultérieurement dans UML/SysML avec quelques adaptations mineures et pour certaines contestables.
- Décrit le comportement du système par diagramme états/transitions
- Hiérarchie d'états représentée par inclusion
- Parallélisme représenté par ligne pointillée

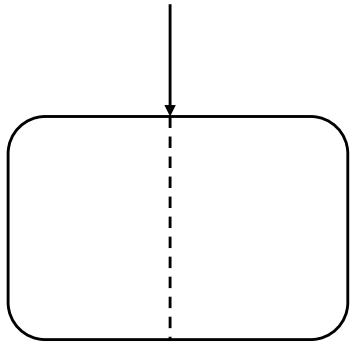


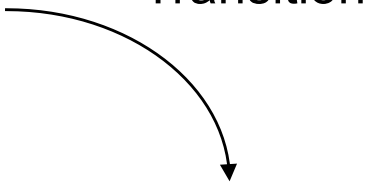
# Objets graphiques



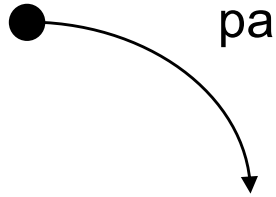



Ligne « ET »





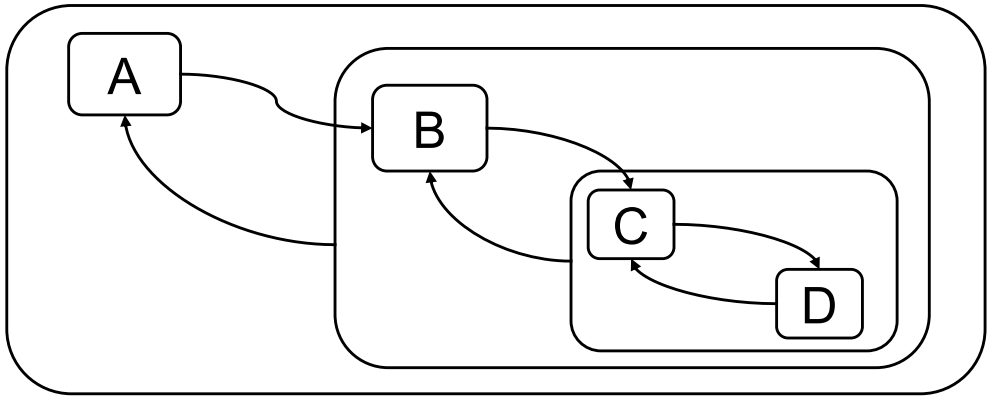

Transition
par défaut



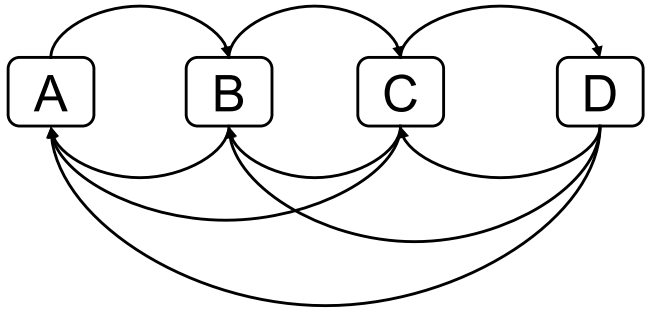
# Hiérarchie d'états



- Un statechart :



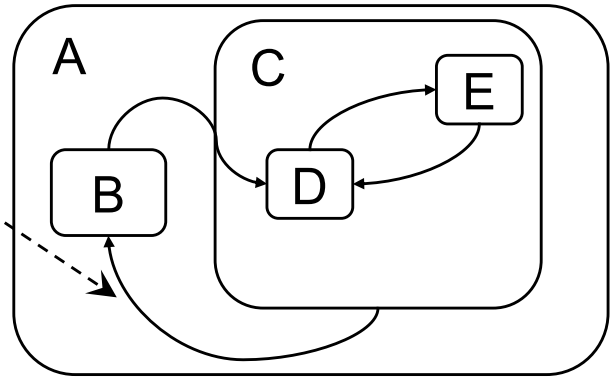

- Le diagramme « à plat » équivalent



# Hiérarchie d'états



- Permet d'exprimer plusieurs transitions possibles avec une seule flèche
- Etat avec sous états = état « OU » : on est dans l'un (OU exclusif) des sous-états => forcément dans un état « feuille » (B, D, E)



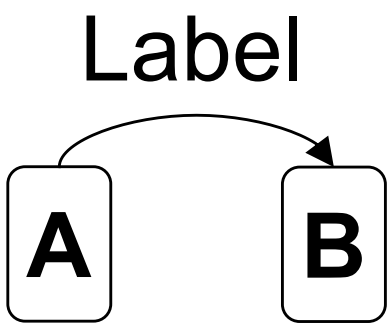
# Statecharts : Hiérarchie d'états



- Limite l'explosion combinatoire du nombre de transitions
- Description très naturelle des modes de fonctionnement d'un système (approche montante ou descendante)
- Très utile pour la spécification de systèmes à interruptions (une seule transition depuis un état de haut niveau)

# Statecharts : Transitions





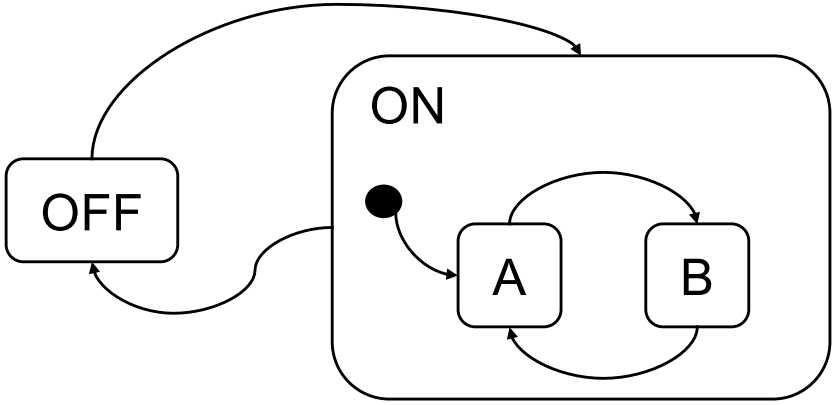

- Label = Trigger/Action
- Trigger = Evénement[Condition]

- La transition est prise si étant dans l'état A, l'événement survient [la condition étant vraie] et alors l'action est faite.
- Tous les éléments du label sont facultatifs, une transition sans label est prise immédiatement.

# Transitions par défaut



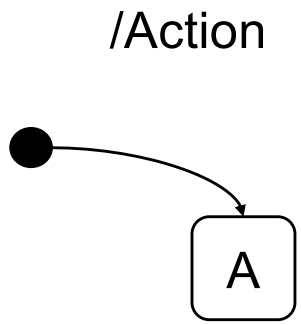
- La transition par défaut permet de faire arriver une transition au bord d'un état composé
- Elle est dans ce cas obligatoire et « prolonge » toutes les transitions arrivant au bord



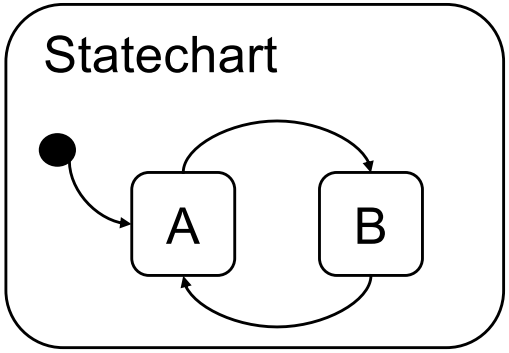
# Transitions par défaut



- La transition par défaut ne peut pas avoir de trigger. Elle peut par contre avoir une action.



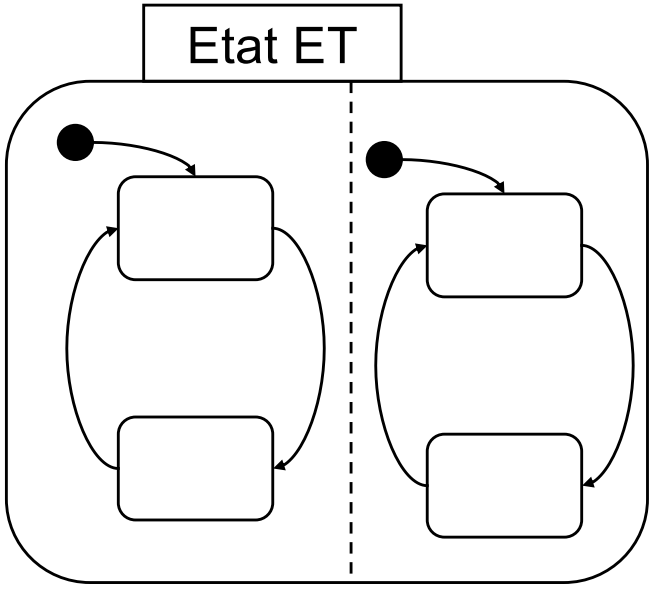

- Une transition par défaut est obligatoire au plus haut niveau d'un statechart



# Etats « ET »



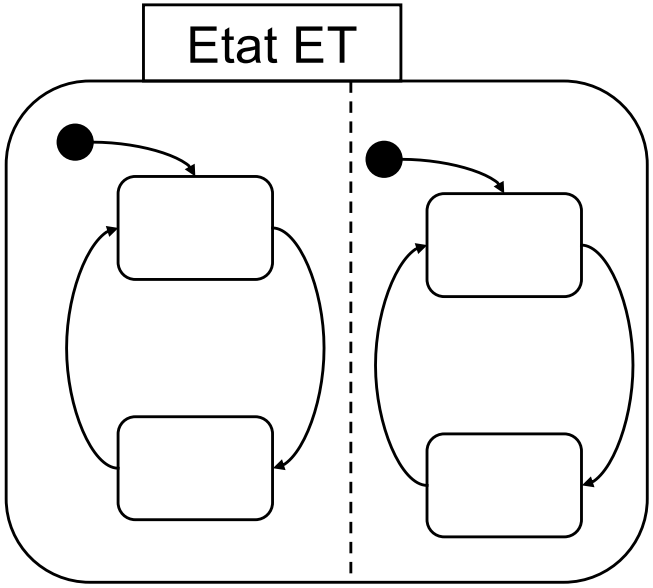
- Aussi appelés états concurrents ou états orthogonaux, ils permettent d'exprimer  l'évolution en parallèle (!) de plusieurs aspects d'un même système
- L'état est séparé par une ligne pointillée dite « ligne ET »



# Etats « ET »



- Etre dans un état « ET » signifie cette fois être dans un état ET dans un autre de part et d'autre de la ligne
- Autant de transitions par défaut que de composantes orthogonales si transitions sur le bord ou niveau statechart

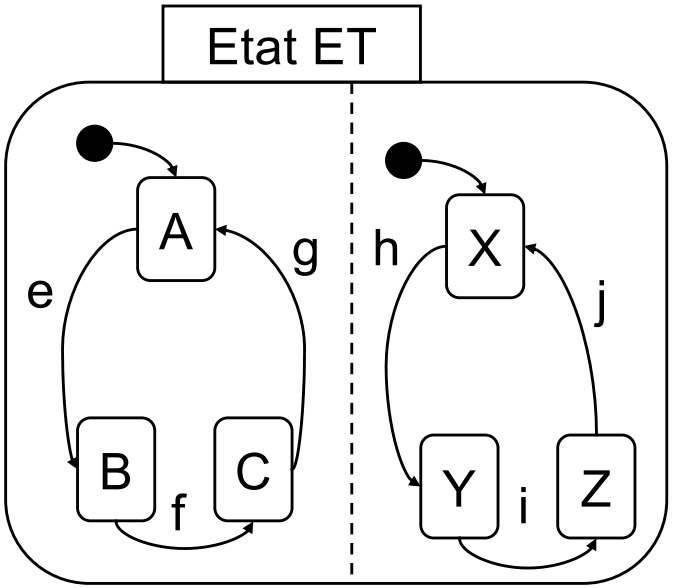


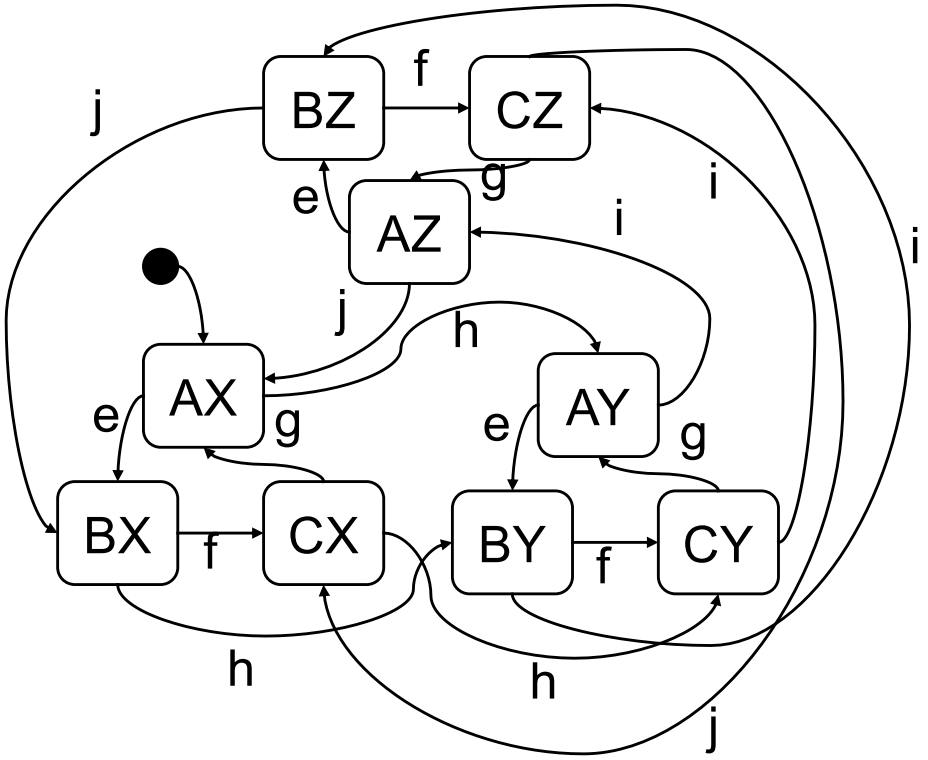
# Etats « ET »



- Statechart avec état ET

Diagramme «à plat» correspondant...

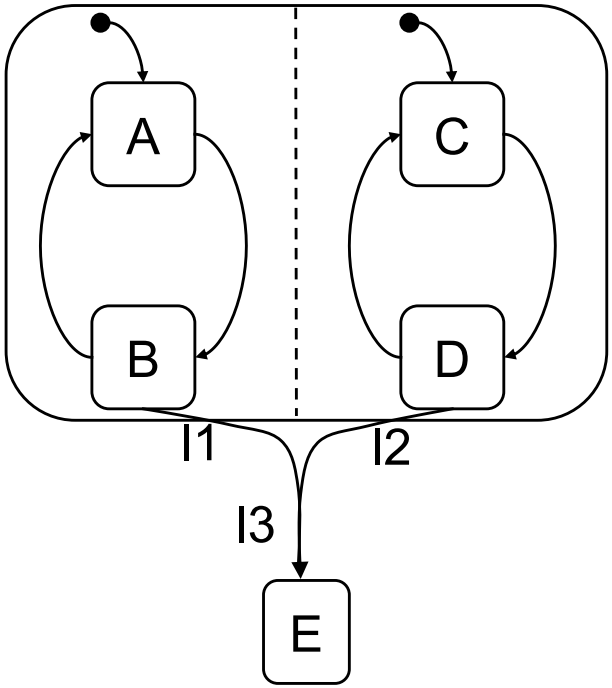


# Connecteurs Fourche convergente



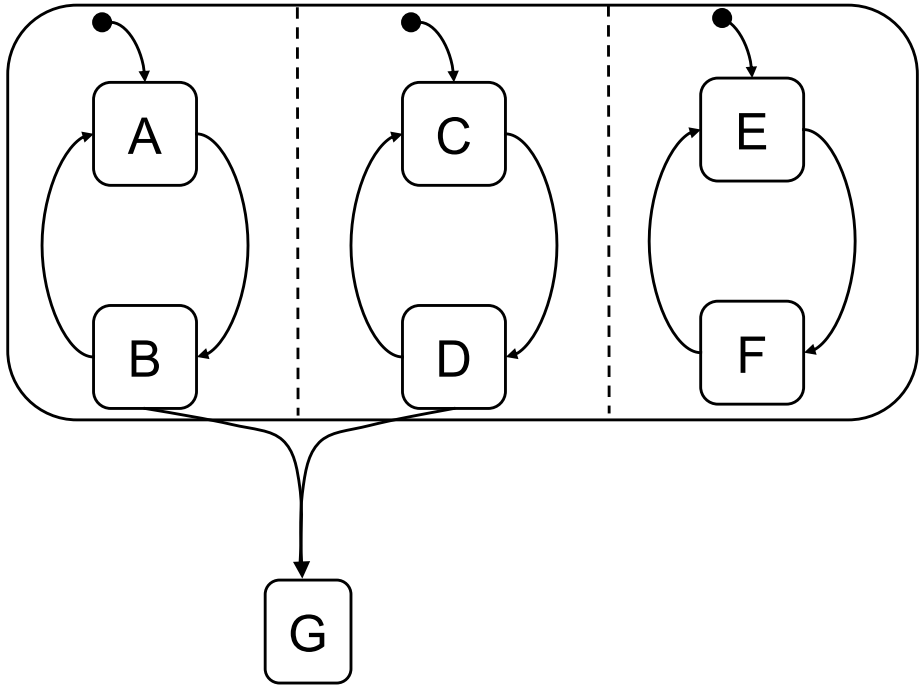
- Sortie d'un état ET
- Correspond à une transition unique
- Chaque tronçon peut avoir son label
- N'est prise que si l'on est dans tous les états de départ et que tous les triggers sont ouverts
- Toutes les actions sont alors prises



# Connecteurs Fourche convergente (joint)



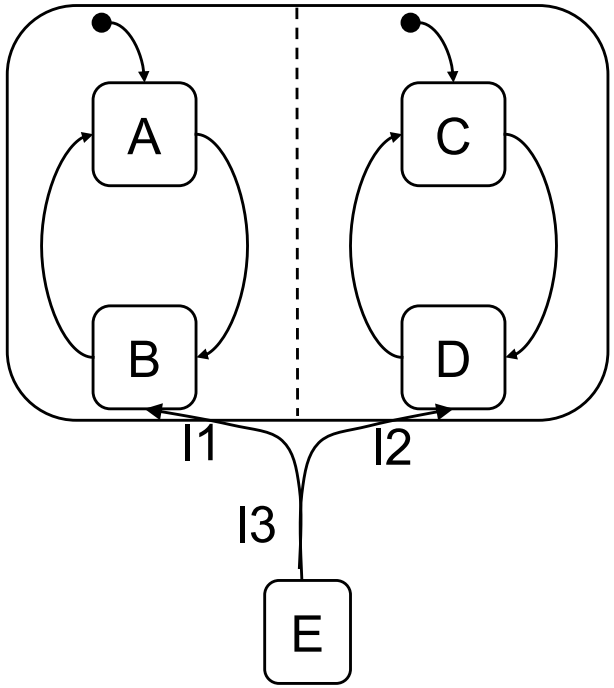
- Certaines composantes orthogonales peuvent ne pas être état de départ. Elles sont alors ignorées dans les critères de transition



# Connecteurs Fourche divergente (fork)



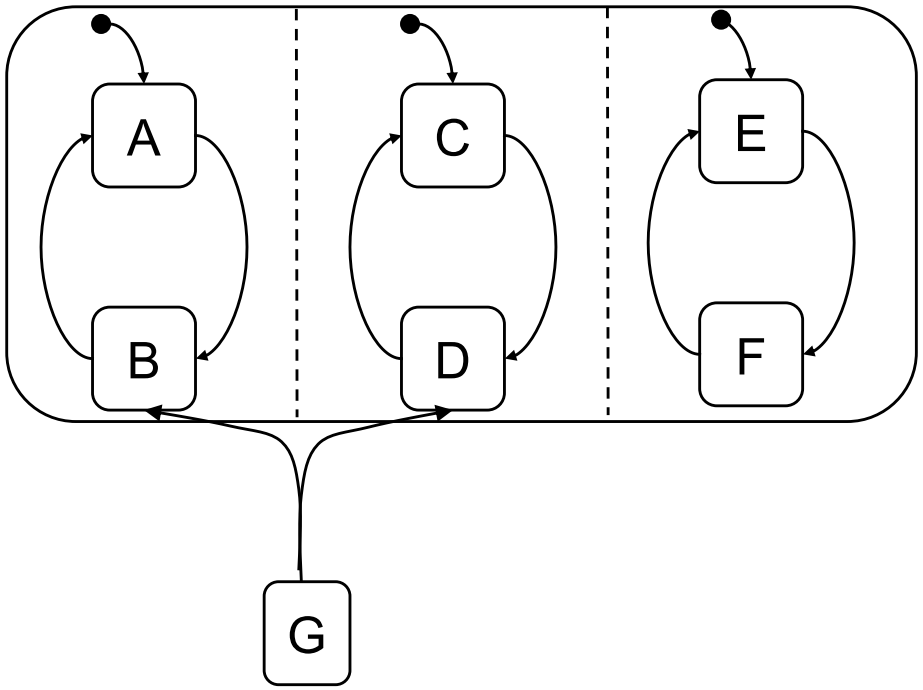
- Entrée dans un état ET
- Correspond à une transition unique
- Chaque tronçon peut avoir son label
- Est  prise depuis l'état de départ et si tous les triggers sont ouverts
- Toutes les actions sont alors prises



# Connecteurs Fourche divergente (fork)



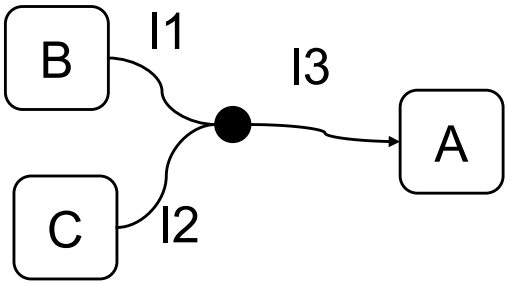
- Certaines composantes orthogonales peuvent ne pas être état d'arrivée. Elles sont alors ignorées dans les critères de transition et la transition par défaut prend le relais

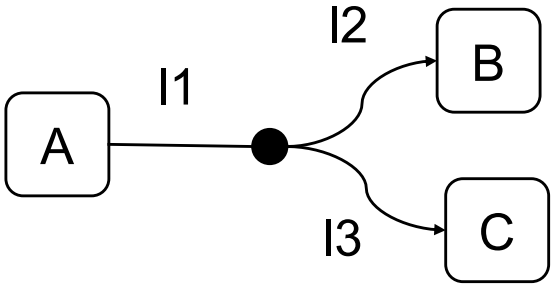


# Connecteurs Jonction (Junction)



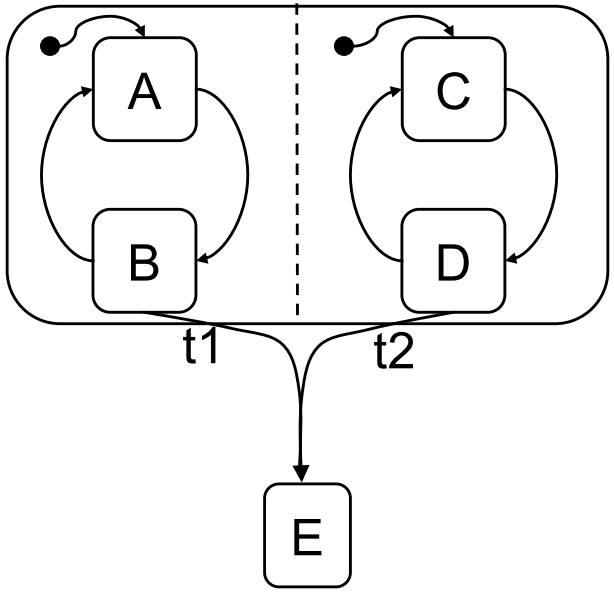
- Correspond à n (nombre de paires départ/arrivée) transitions
- Chacune d'entre elle est tirée si on est dans l'état de départ et que tous les triggers rencontrés sur le chemin correspondant sont ouverts
- Toutes les actions du chemin sont alors prises



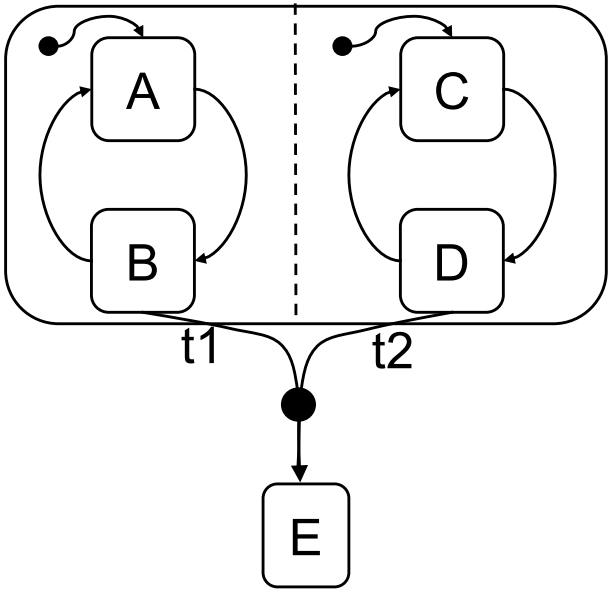



# Connecteurs





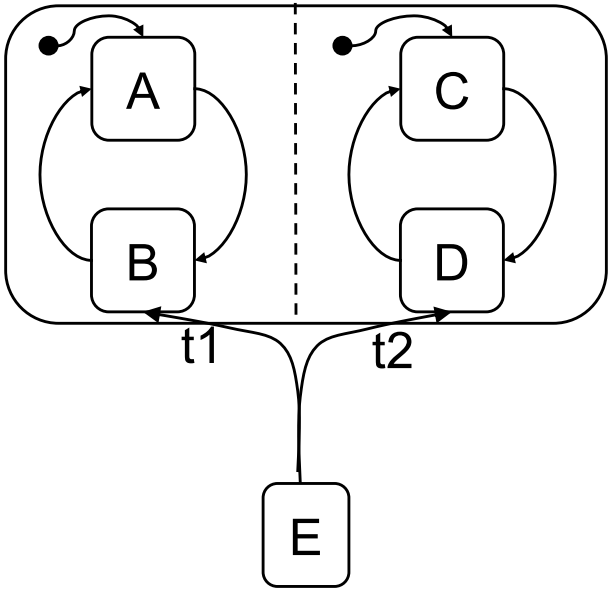

Transition si dans
(B,D) t1 et t2 ouverts




Transition si dans
(B,*) avec t1 ouvert ou
(*,D) avec t2 ouvert

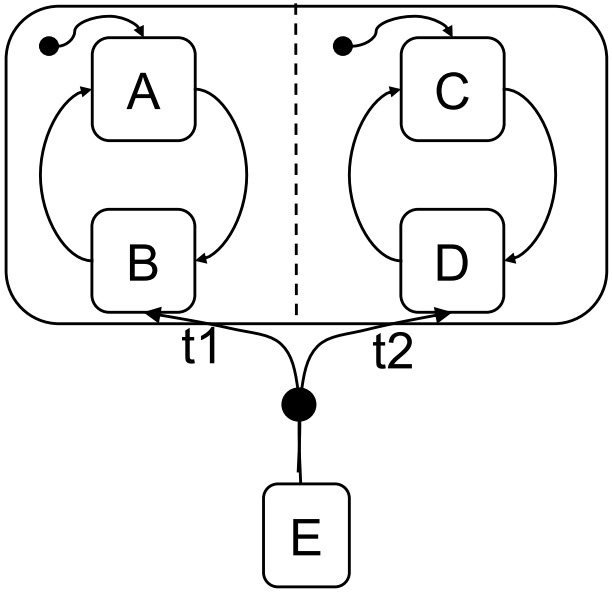
# Connecteurs






Transition vers
(B,D) si t1 et t2 ouverts

**Depuis E :**

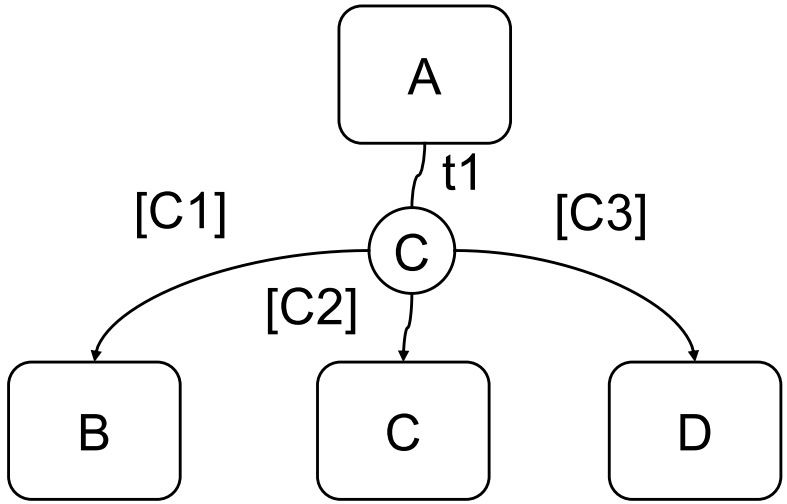



Transition vers :
(B,C) si t1 ouvert ou
(A,D) si t2 ouvert

# Connecteurs Condition



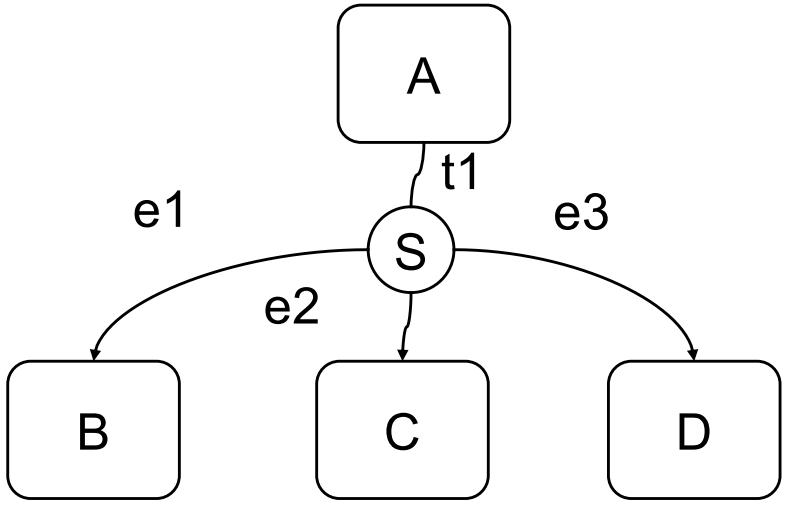
- Fonctionne comme le connecteur jonction
- Sert à exprimer un choix sur conditions
- Les conditions doivent être mutuellement exclusives



# Connecteurs Aiguillage (switch)

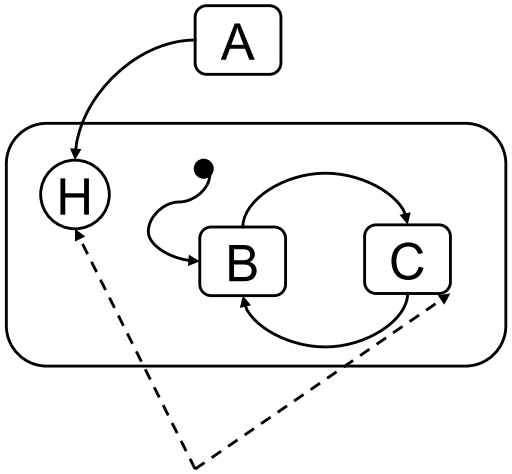


- Fonctionne comme le connecteur jonction
- Sert à exprimer un choix sur événements
- Les événements doivent être mutuellement exclusifs



# Connecteurs Historique



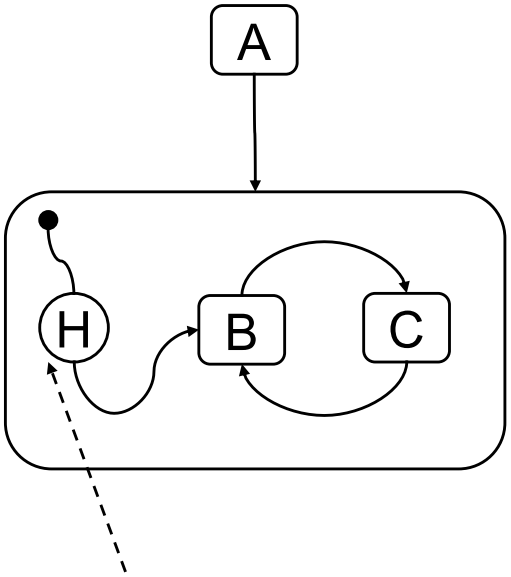



Transition vers C si C est le dernier état visité

- Sert à mémoriser le dernier sous-état visité d'un état composé
- Est prolongé implicitement vers ce dernier si l'historique existe
- Est remplacé par la transition par défaut sinon

# Connecteurs Historique






L'état par défaut est le dernier état visité s'il existe, B sinon

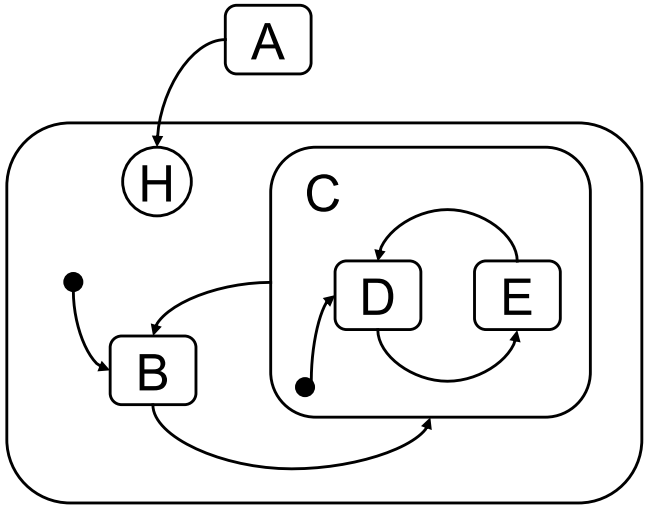
- Le connecteur historique peut être placé le long d'une transition par défaut
- La mémorisation de l'historique peut être effacée par l'action

  History_clear! : hc!(état)

# Connecteurs Historique



- Le connecteur historique ne mémorise que l'histoire concernant les états se trouvant au même niveau hiérarchique

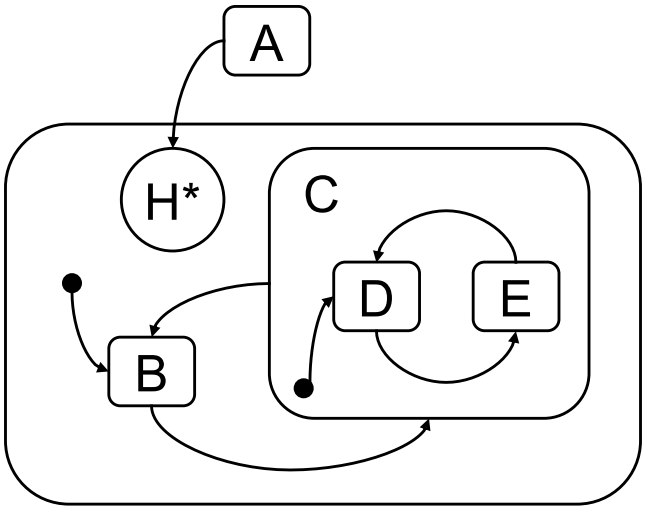



- E est le dernier état visité
- Le connecteur H ne mémorise que l'état C
- La transition sur H aboutit à D et non à E

# Connecteurs Historique profond



- Le connecteur historique profond H* mémorise l'histoire à tout niveau de hiérarchie
- Peut être effacé par l'action  Deep_clear! : dc!(état)
- hc! n'efface que son niveau immédiat

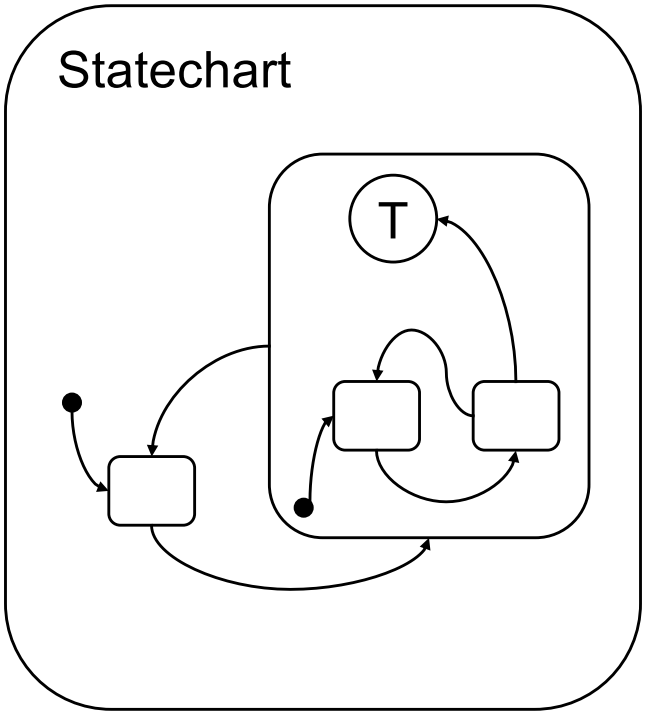



- E est le dernier état visité
- La transition sur H* aboutit à E

# Connecteurs fin (termination)



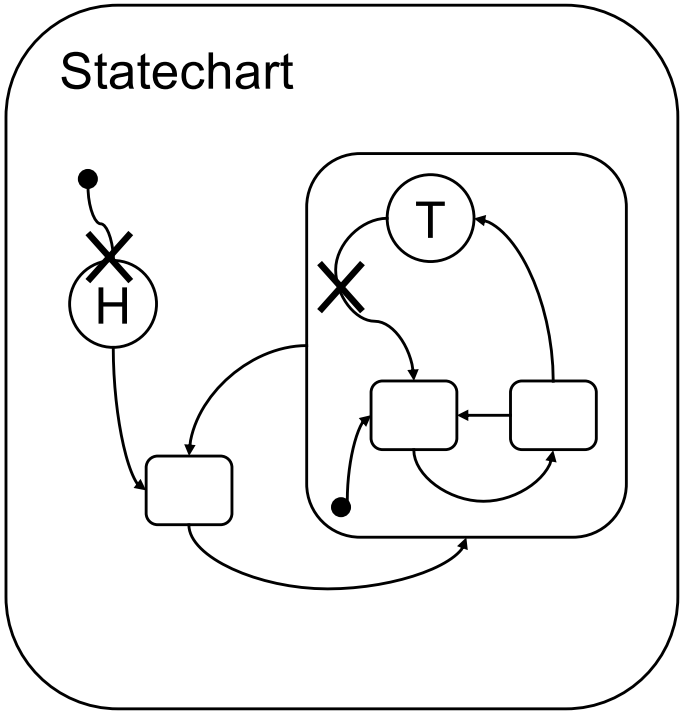
- Le connecteur de fin (termination) met fin au statechart
- Il peut être placé à tout niveau de la hiérarchie
- L'activité contrôlée par le statechart est alors arrêtée



# Connecteurs fin (termination)



- Il est évidemment interdit de placer une transition depuis un connecteur T
- Il est en outre déconseillé de placer un connecteur H ou H* sur la transition de défaut d'un statechart



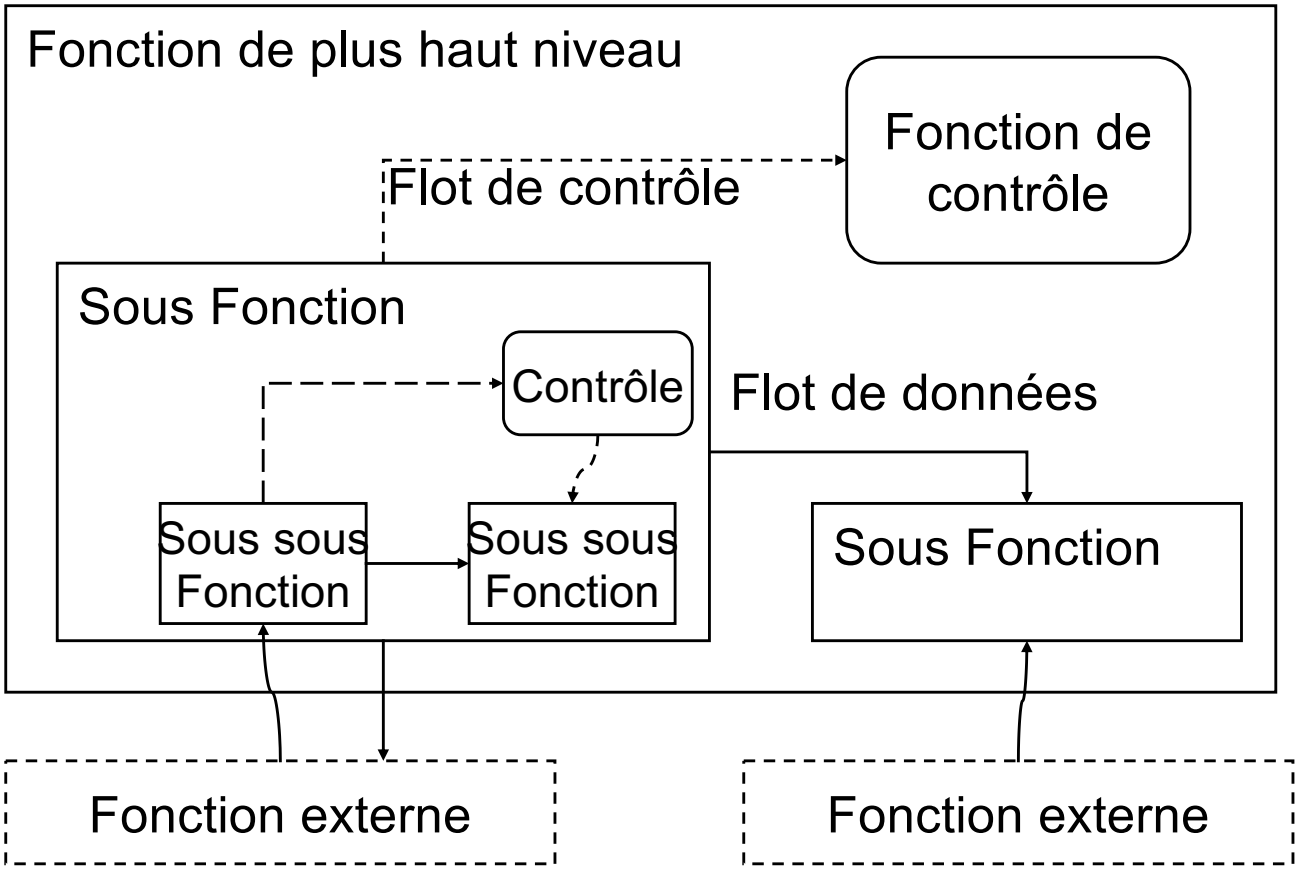
# Contrôle d'activités par des Statecharts



- Statemate : Outil de spécification et de simulation commercialisé par la société iLogix (Andover Massachusetts) fondée en 1987.
- Permet la modélisation d'un système sous trois aspects :
  - Fonctionnel (arborescence de fonctions appelées « activités »).
  - Comportemental ( dynamique des fonctions exprimée par des Statecharts).
  - Structurel (arborescence des composants physiques ou modules).

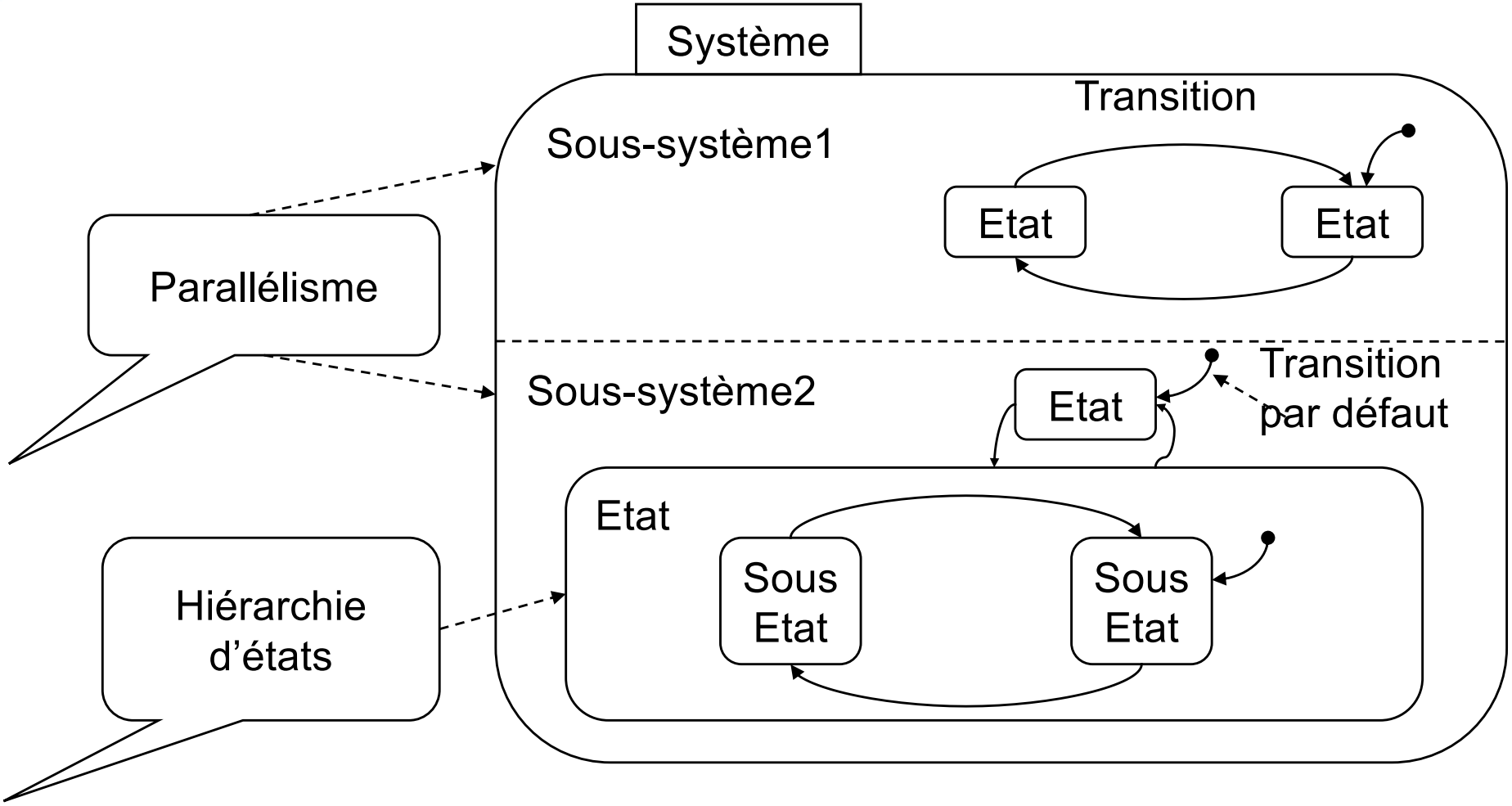
# Statemate : vue fonctionnelle





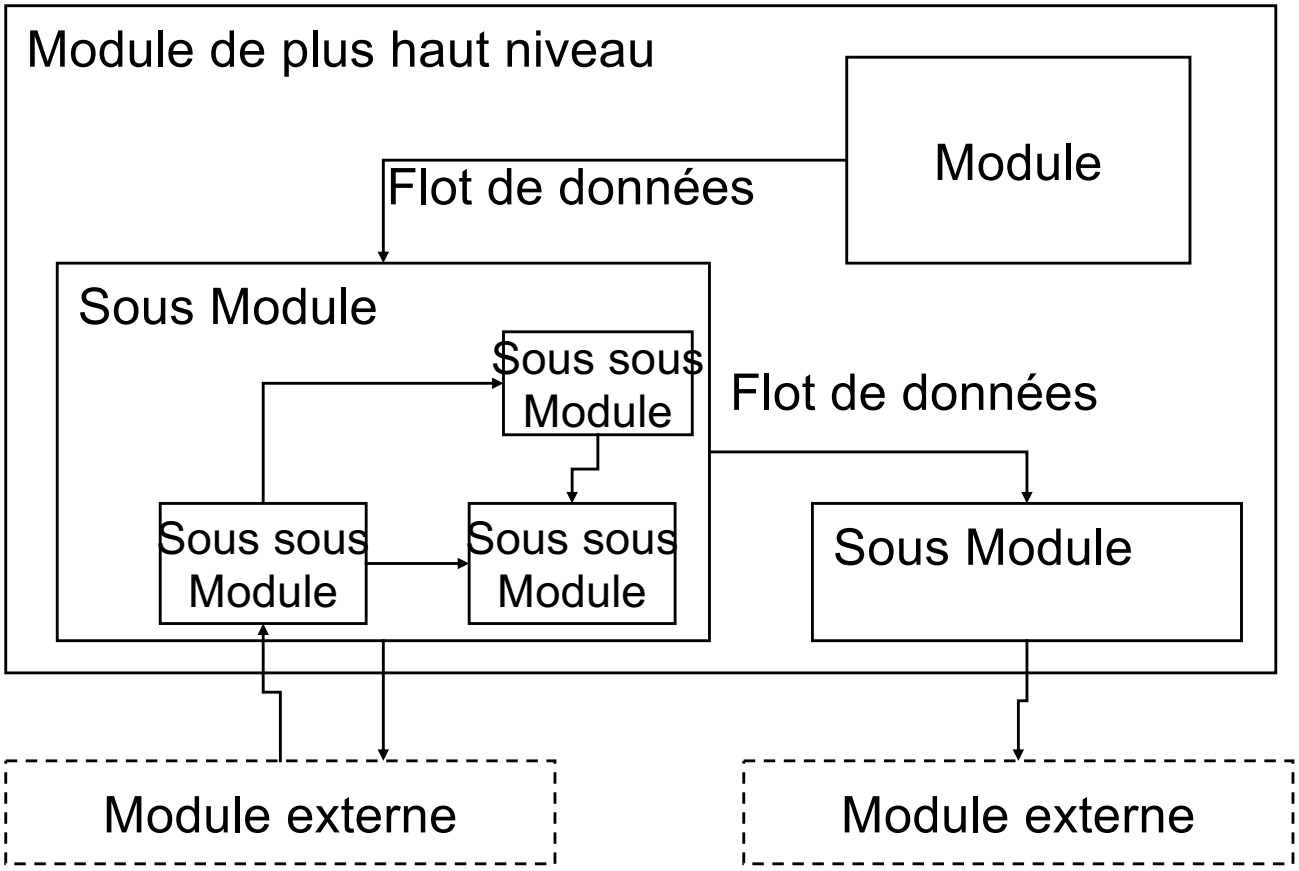
# Statemate : vue comportementale





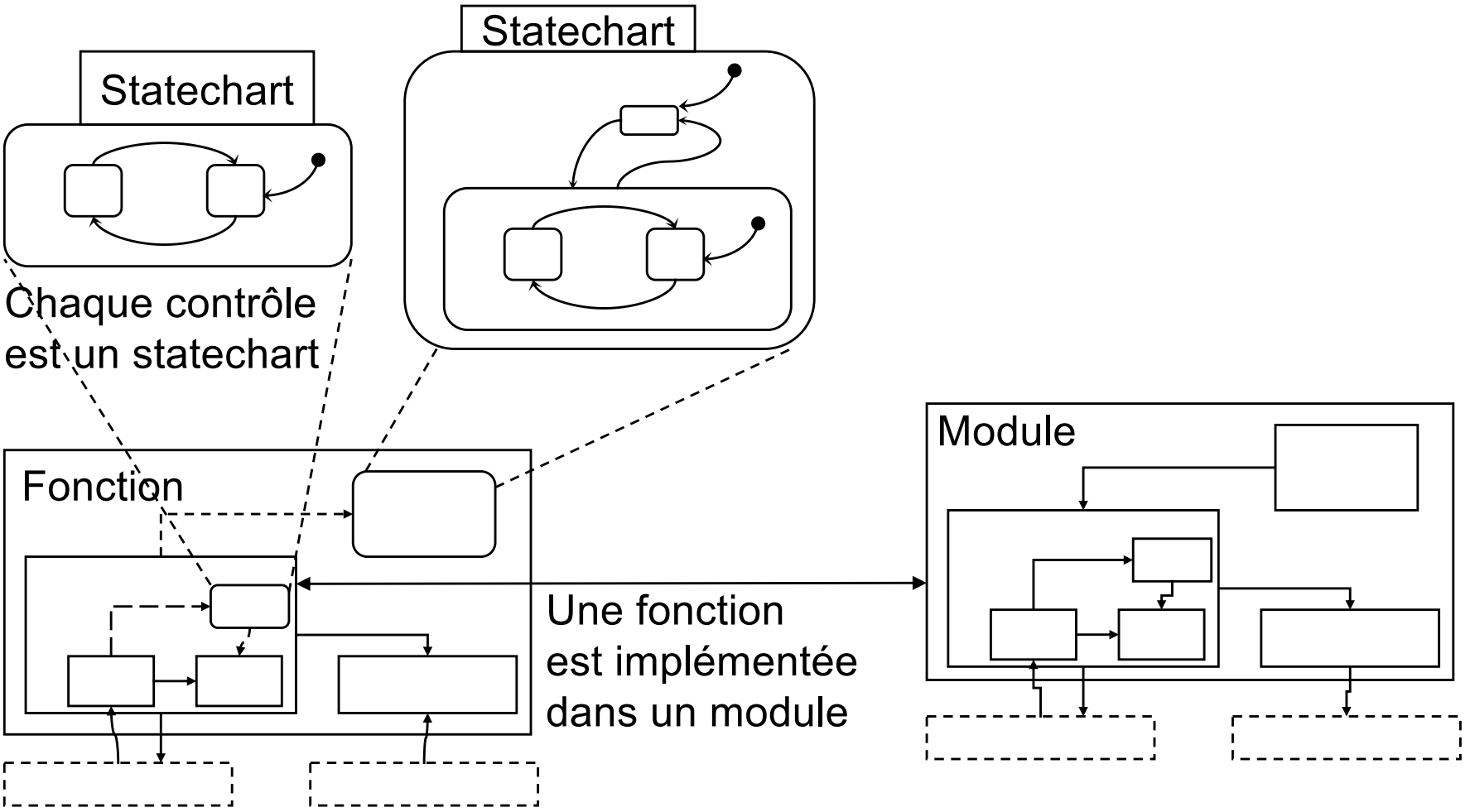
# Statemate : vue matérielle





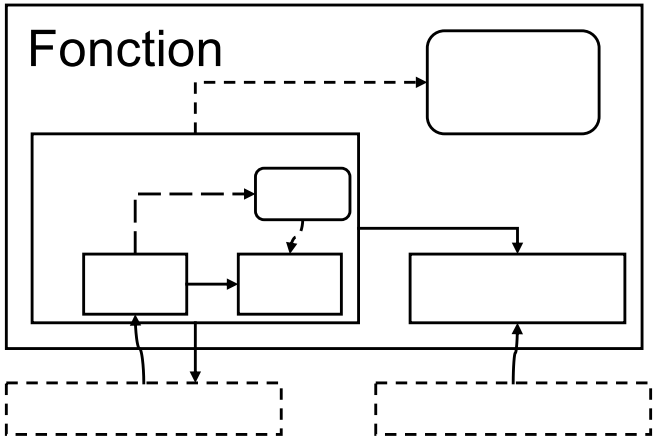
# Statemate : liens entre vues





# Diagrammes fonctionnels

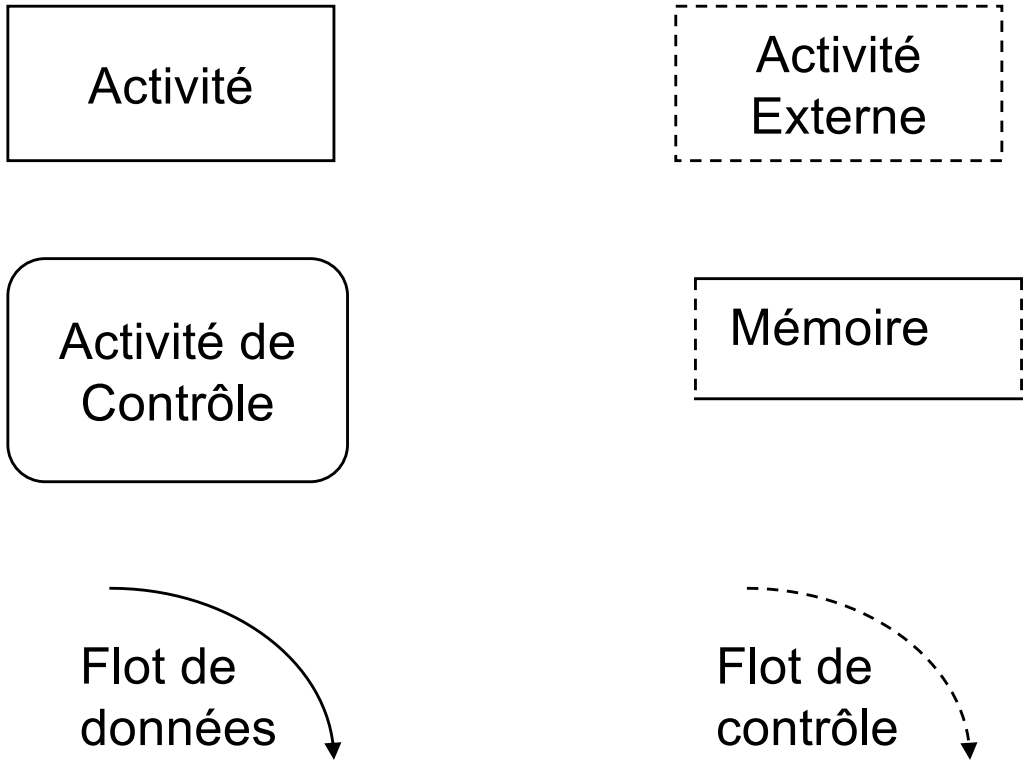





- Description hiérarchique des fonctions du système.
- Hiérarchie représentée par l'inclusion des boites.
- Echanges entre fonctions représentés par des flèches.

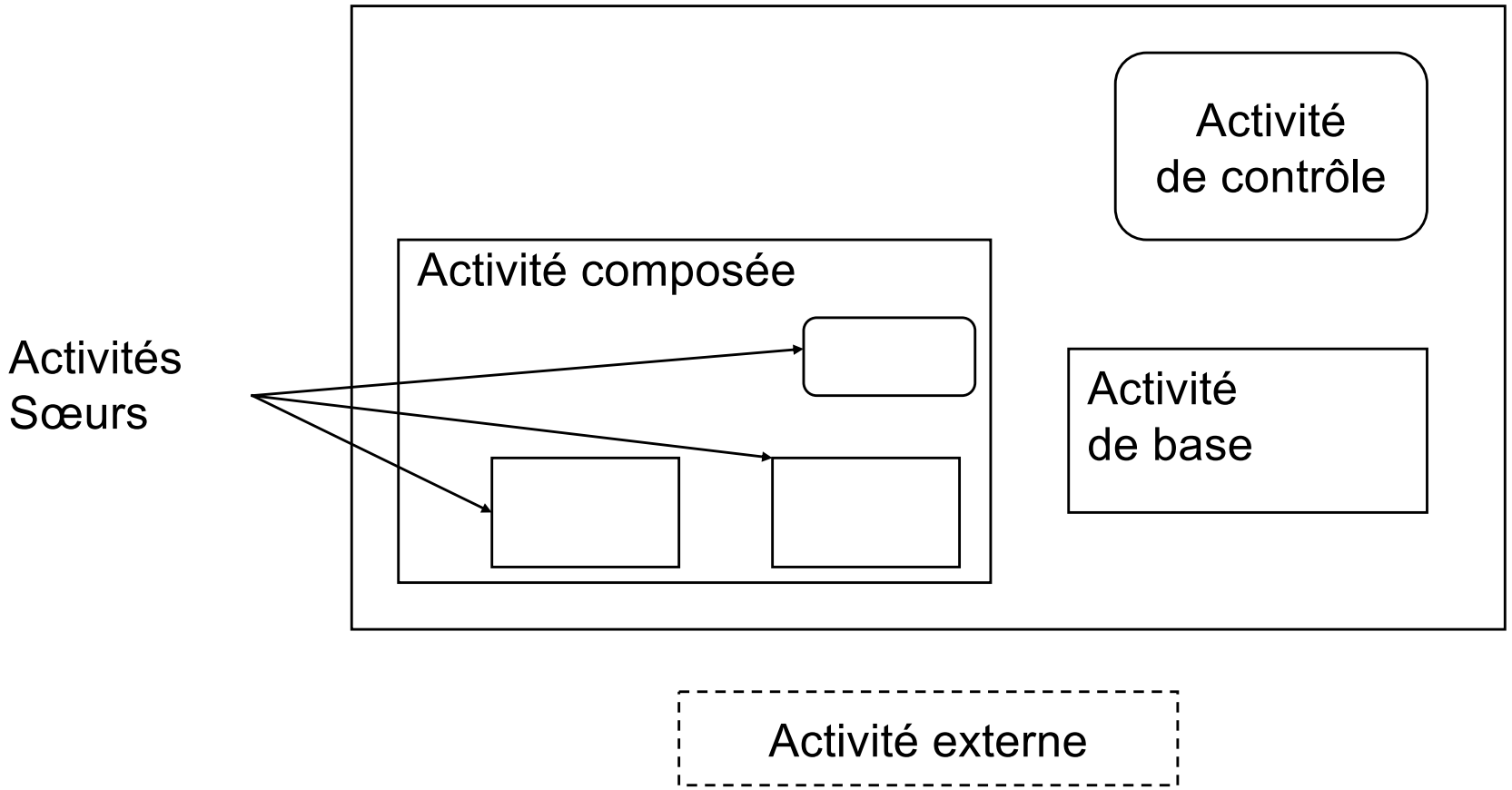
# Diagrammes fonctionnels
# Objets graphiques





# Diagramme fonctionnels
# Hiérarchie des activités





# Diagrammes fonctionnels
# Hiérarchie des activités



- Les activités externes et de contrôle ainsi que les mémoires ne peuvent être décomposées.
- Les activités sœurs doivent avoir un nom unique (instances multiples interdites).
- Au maximum une activité de contrôle par niveau hiérarchique.

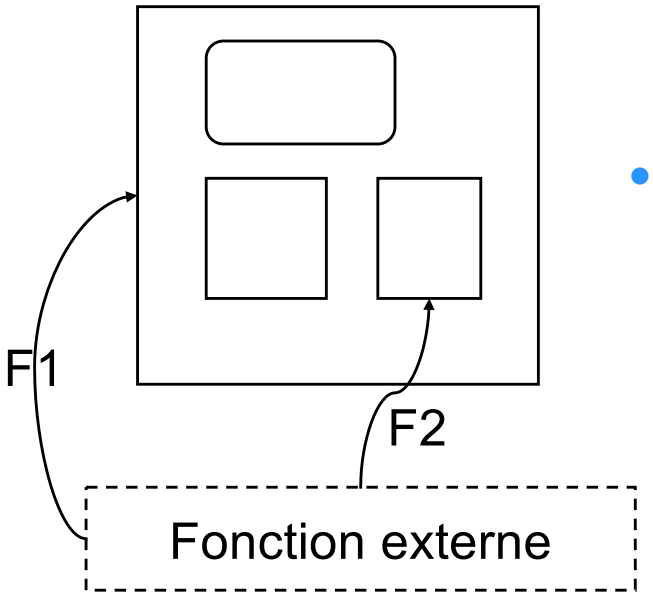
# Diagrammes fonctionnels
# Flots de contrôle et de données



- Flots de contrôle : événements (ne vivent que sur un pas de simulation) et conditions (conservent leur valeur jusqu'à changement explicite).
- Flots de données : entier, réel, chaîne, bit, trame, donnée structurée, événement, condition...

# Diagrammes fonctionnels
# Flots de contrôle et de données

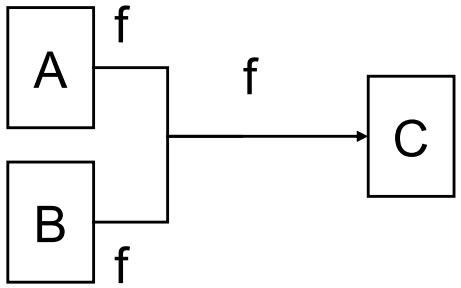




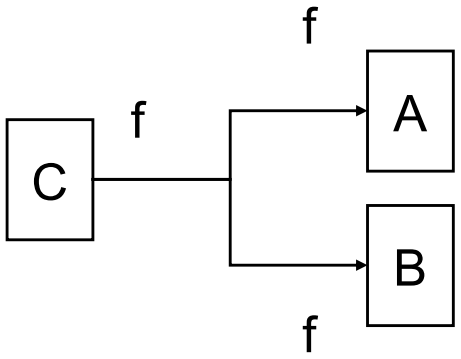

- Les flots arrivant sur le bord d'une activité composée sont accessibles à tous les descendants.
- Les flots arrivant explicitement sur une sous-activité ne sont accessibles qu'à celle-ci.

# Diagrammes fonctionnels
# Connecteurs





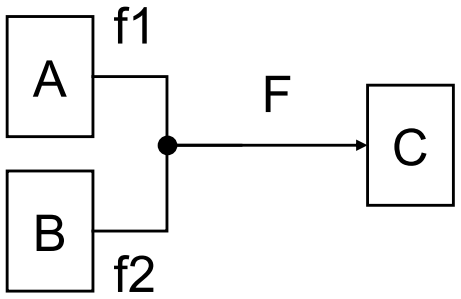

- Fourche convergente (joint): une même information venant de plusieurs sources.

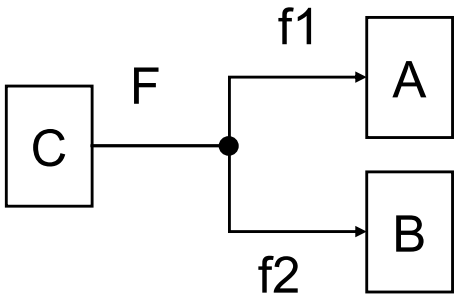



- Fourche divergente (fork): une même information allant vers plusieurs destinations.

# Diagrammes fonctionnels
# Connecteurs






F=(f1;f2)

Junction : ● ou record : ◑

- Fusion : combinaison de plusieurs informations.
- Dissociation : séparation des éléments d'une information composée.

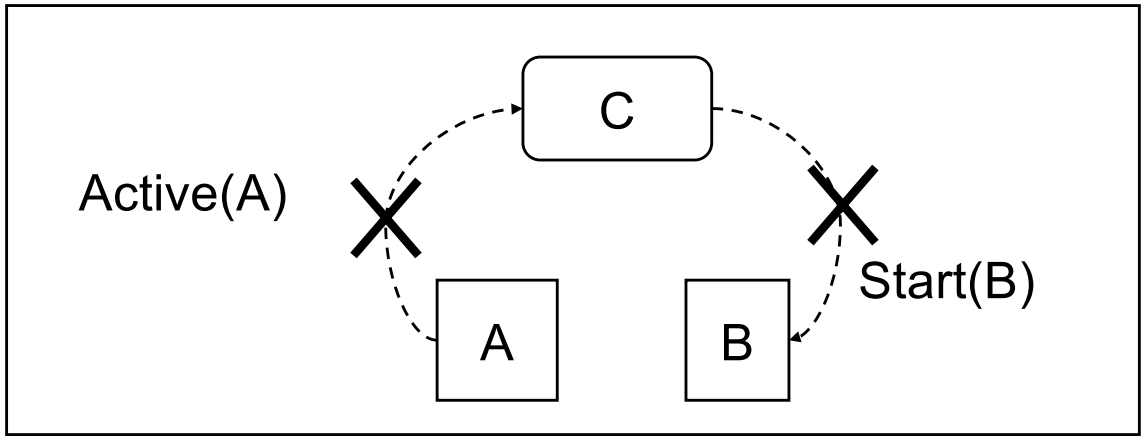
# Relations diagrammes fonctionnels / Statecharts



- Chaque activité de contrôle est décrite par un statechart décrivant le comportement de son activité parente.
- Elle connaît l'état (actif, arrêté, suspendu) de ses activités sœurs.
- Elle commande l'état de ses activités sœurs (démarrer, arrêter, suspendre, reprendre).

# Relations diagrammes fonctionnels / Statecharts






Les flots de contrôle depuis et vers les activités de contrôle transportant les commandes et contrôles d'état (actif, stoppé…) sont implicites.

# Relations diagrammes fonctionnels / Statecharts



- Une activité suspendue est gelée dans l'état où elle était lors de la commande de suspension. Elle reprendra dans cet état lors de la commande de reprise.
- Au contraire une activité arrêtée oublie définitivement l'état où elle était.
  Elle reprendra dans un état par défaut lors de la commande de démarrage.

# Relations diagrammes fonctionnels / Statecharts

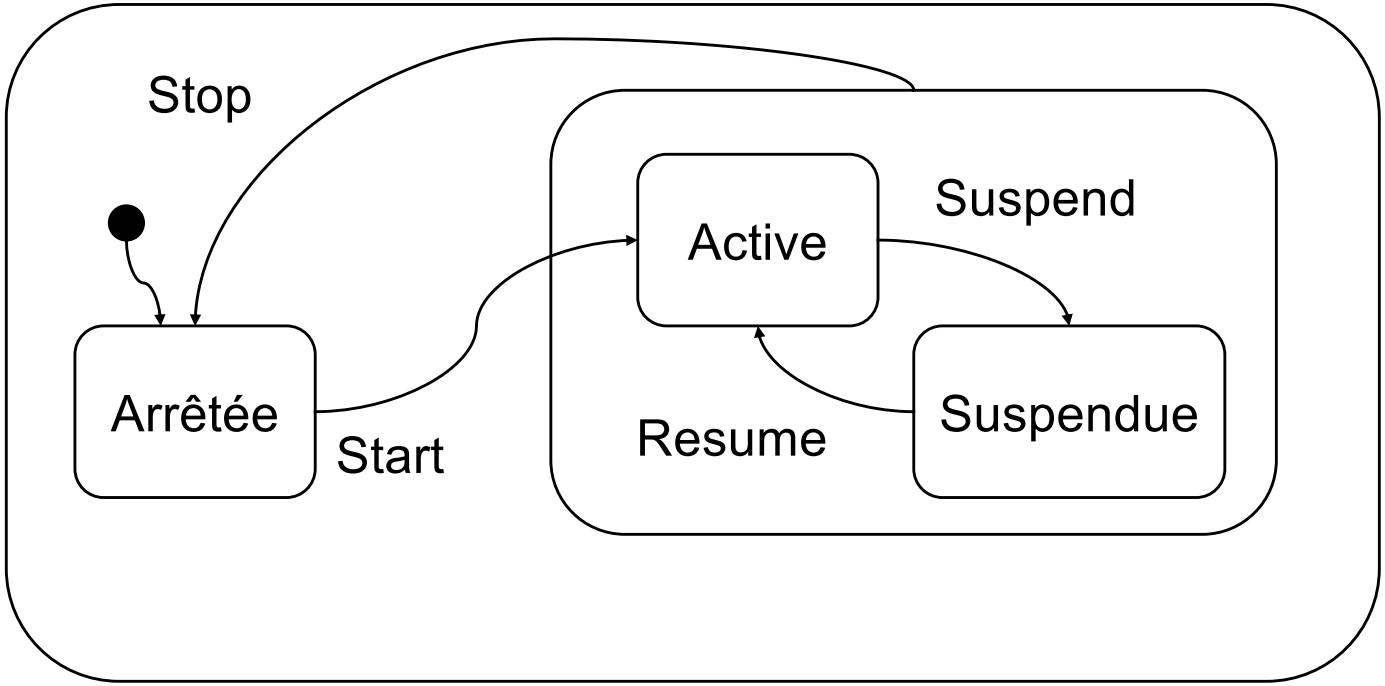


- Si une activité composée n'a pas d'activité de contrôle toutes les descendantes immédiates sont actives quand le parent est actif.
- Si l'activité de contrôle existe les activités réagissent aux commandes démarrer (start), arrêter (stop), suspendre (suspend), reprendre (resume).

# Relations diagrammes fonctionnels / Statecharts



- Leurs changements d'état possibles sont illustrés par le statechart suivant :



# Evénements



| Evénement | Abr | Se produit quand |
|---|---|---|
| entered(S) | en(S) | on entre dans l'état S |
| exited(S) | ex(S) | on sort de l'état S |
| started(A) | st(A) | l'activité A démarre |
| stopped(A) | sp(A) | l'activité A s'arrête |
| changed(V) | ch(V) | la valeur de la variable V (condition ou donnée alphanumérique) change |

# Evénements



| Evénement | Abr | Se produit quand |
|---|---|---|
| true(C) | tr(C) | la condition C devient vraie |
| false(C) | fs(C) | la condition C devient fausse |
| read(V) | rd(V) | la variable V est lue |
| written(V) | wr(V) | la variable V est écrite |
| timeout(E,N) | tm(E,N) | généré N unités de temps après l'événement E |

# Conditions



| Condition | Abr | Est vraie si |
|---|---|---|
| in(S) | | on est dans l'état S |
| active(A) | ac(A) | l'activité A est active |
| hanging(A) | hg(A) | l'activité A est suspendue |
| not C | | la condition C est fausse |
| C1 and C2 | | les conditions C1 et C2 sont vraies |

# Conditions



| Condition | Abr | Est vraie si |
|---|---|---|
| C1 or C2 | | L'une des conditions C1 ou C2 est vraie |
| all(C) | | Toutes les composantes du tableau de conditions C sont vraies |
| any(C) | | Au moins une composante du tableau de conditions C est vraie |

# Conditions



| Condition | Abr | Est vraie si |
|---|---|---|
| V1 = V2 | | Les variables V1 et V2 satisfont à la relation correspondante. (>,<,>=,<= ne s'appliquent qu'à des variables numériques) |
| V1 /= V2 | | |
| V1 > V2 | | |
| V1 < V2 | | |
| V1 >= V2 | | |
| V1 <= V2 | | |

# Actions



| Action | Abr | Résultat |
|---|---|---|
| E | | L'événement E se produit |
| make_true(C) | tr!(C) | La condition C devient vraie |
| make_false(C) | fs!(C) | La condition C devient fausse |
| start(A) | st!(A) | L'activité A démarre |
| stop(A) | sp!(A) | L'activité A s'arrête |

# Actions



| Action | Abr | Résultat |
|---|---|---|
| suspend(A) | sd!(A) | L'activité A devient pendante |
| resume(A) | rs!(A) | L'activité A redevient active |
| history_clear(S) | hc!(S) | Efface l'historique de l'état S |
| deep_clear(S) | dc!(S) | Efface l'historique profond de l'état S |

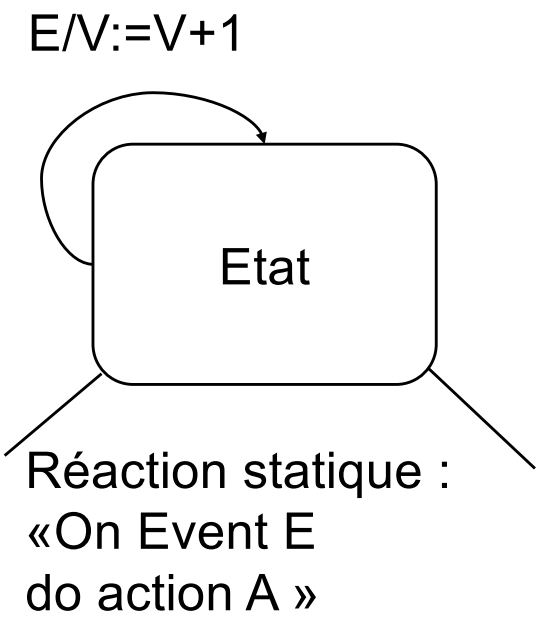
# Actions



| Action | Abr | Résultat |
|---|---|---|
| write_data(V) | wr!(V) | La variable V est écrite |
| read_data(V) | rd!(V) | La variable V est lue |
| V := expression | | La variable V prend pour valeur l'expression |
| schedule(A,N) | sc!(A,N) | L'action A sera effectuée après N unités de temps |

# Réactions statiques






- Permettent de décrire des changements de comportement au sein d'un même état.
- Evitent des transitions bouclées sur un état.
- Ont la même syntaxe E[C]/A que les labels de transitions
- Sont spécifiées dans un éditeur textuel.

# Réactions statiques

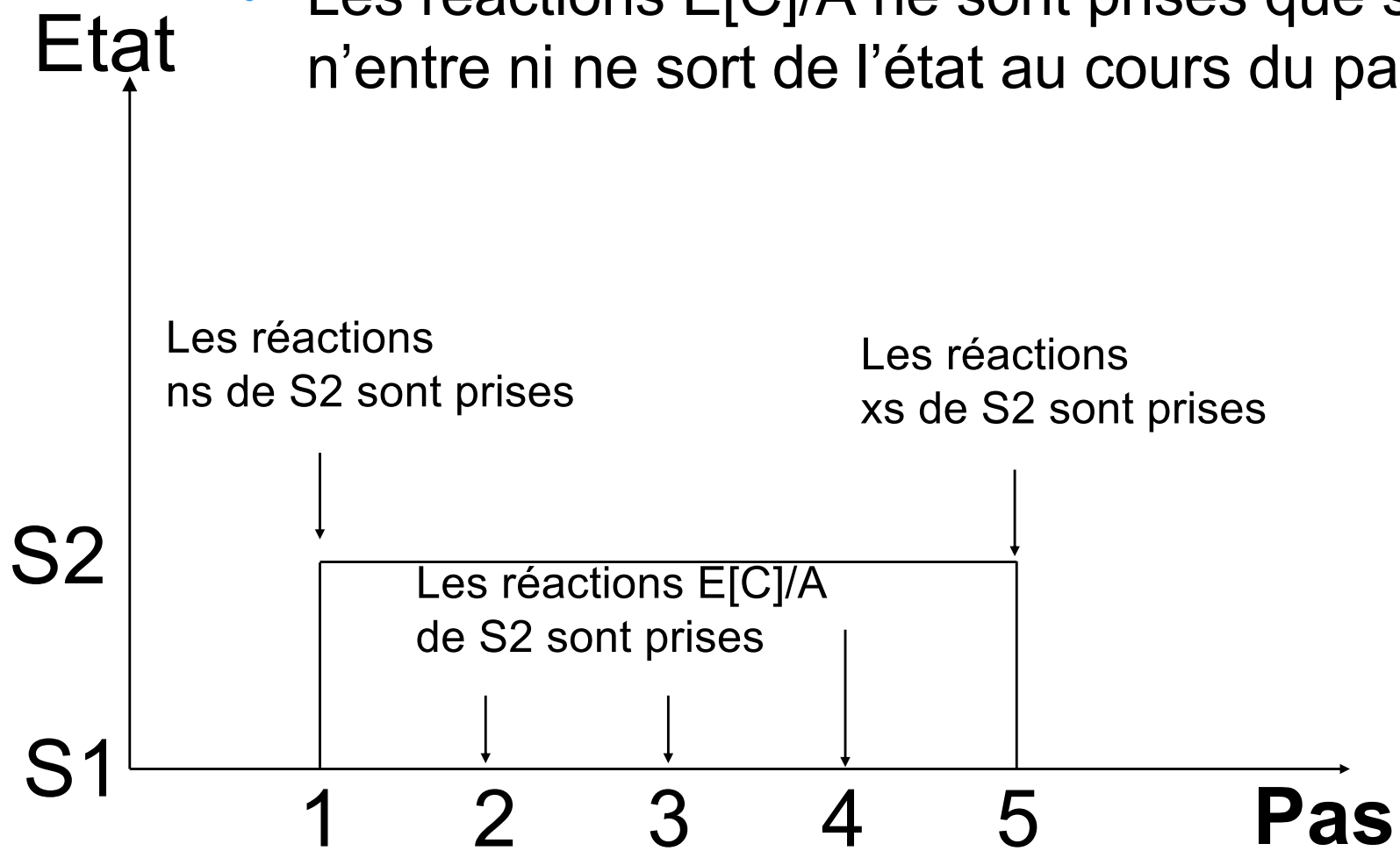


- Il est possible de spécifier des réactions statiques dont l'événement trigger est le fait d'entrer ou de sortir de l'état :
  - entering/A ou en abrégé ns/A
  - exiting/A ou en abrégé xs/A
- Est équivalent à rajouter /A sur le label de toutes les transitions entrantes (respectivement sortantes)

# Réactions statiques



- Les réactions E[C]/A ne sont prises que si l'on n'entre ni ne sort de l'état au cours du pas



# Relations statechart/activity chart

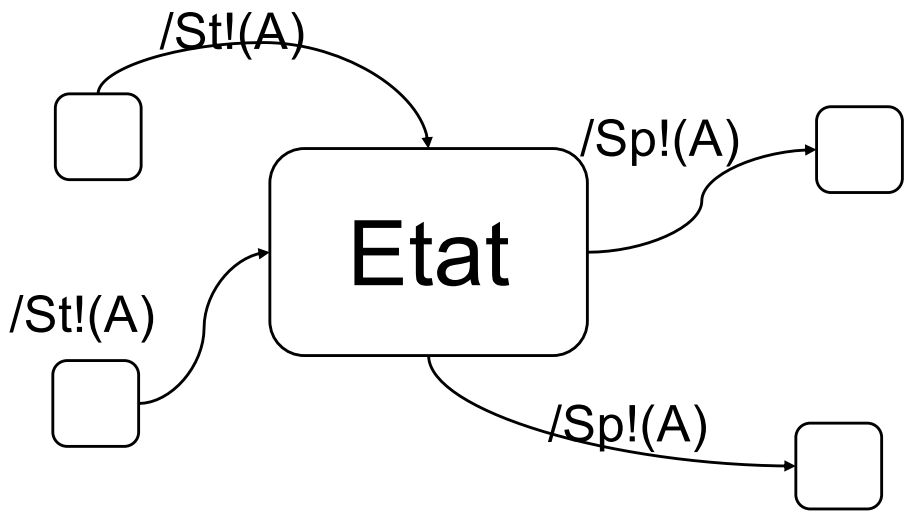


- Une activité peut être spécifiée comme « throughout » ou « within » un état
  - Throughout : l'activité est démarrée quand on rentre dans l'état et stoppée quand on en sort
  - Within : l'activité est seulement stoppée quand on sort de l'état

# Relations statechart/activity chart



- Les relations throughout et within évitent des st! et sp! explicites sur toutes les transitions entrantes (respectivement sortantes).

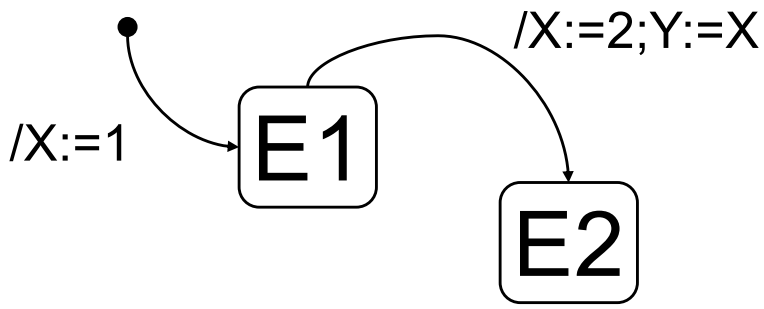



L'équivalent d'une relation throughout

# Actions multiples



- Plusieurs actions au cours d'un même pas. Syntaxe : /A1;A2;A3…
- Les valeurs de variables avant l'action sont utilisées pour calculer les nouvelles valeurs:

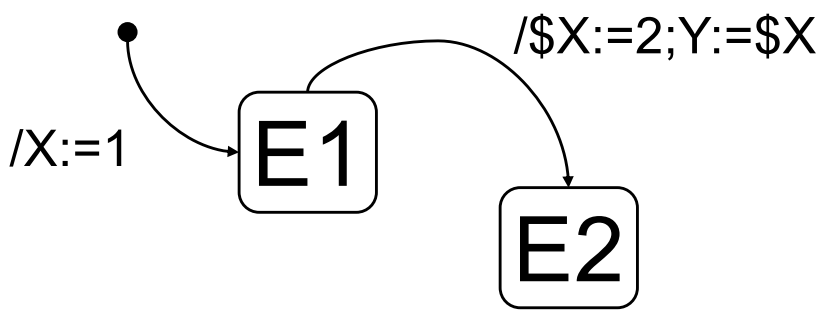



Après le pas E1->E2, Y=1
(valeur de X avant le pas)

# Variables de contexte



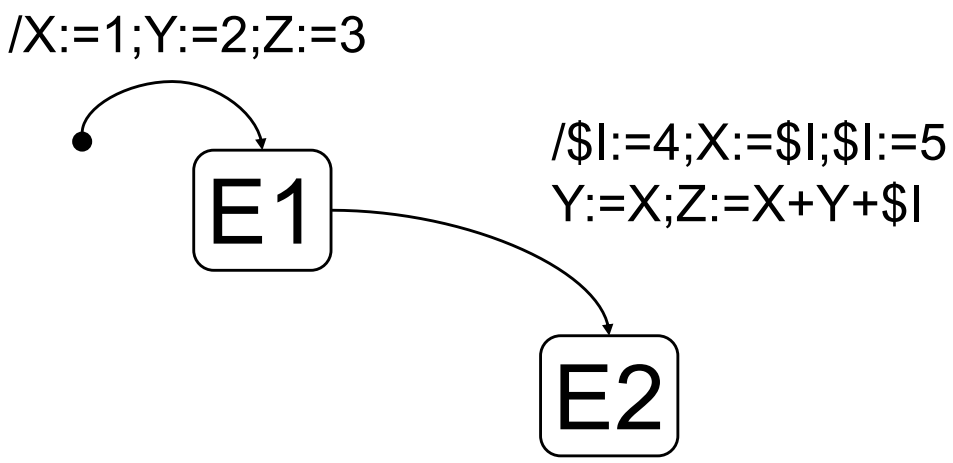
- Locales à une action multiple elles peuvent changer au cours d'un pas. Identifiées par leur nom commençant par un $
- Valeurs non mémorisées d'un appel à l'autre d'une action multiple (réinitialisées à 0)




Après le pas E1->E2, Y=2

# Variables de contexte : Exercice






- Après simulation de ce statechart
- X = ?  Y = ?  Z = ?
- X = 4  Y = 1  Z = 8

# Actions multiples : boucles



- Les variables de contexte permettent de réaliser des boucles dans des actions multiples
- De telles boucles sont exécutées en un seul pas => les indices de boucles sont forcément des variables de contexte

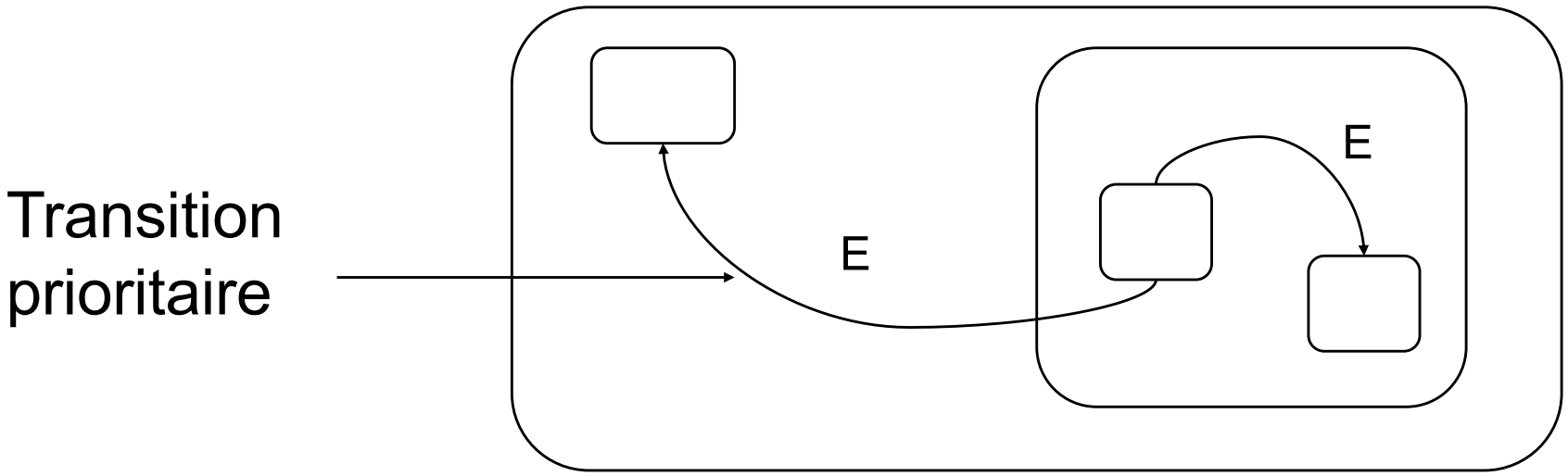
```
for $I in P to N loop
        action
end loop
```

```
while condition loop
        action
end loop
```

# Règle de priorité des transitions



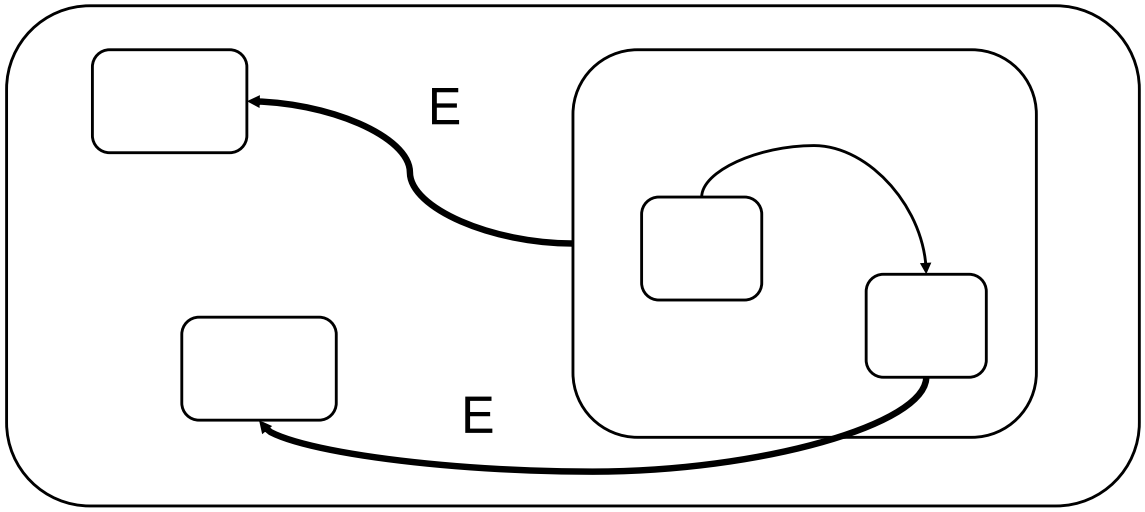
- Si plusieurs transitions sont en conflit, priorité est donnée à celle dont les états de départ et d'arrivée ont l'ancêtre commun de plus haut niveau :



# Non déterminisme



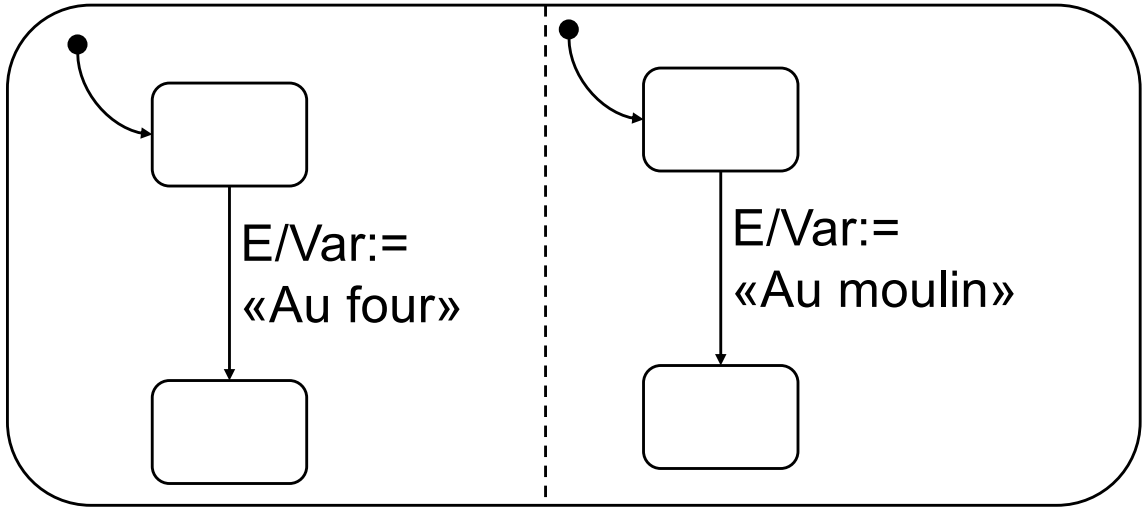
- Si la règle précédente ne permet pas de départager, on est dans un cas de non déterminisme détecté par l'outil de simulation



# Compétition (racing)



- Un autre problème détecté en simulation est la tentative de modifications contradictoires d'une  même variable au cours d'un même pas



# Spécificités SysML



- Hors sujet pour ce cours qui traite des Statecharts stricto sensu (diagrammes de Harel) avant leur intégration dans UML
- Pour les adaptations des diagrammes d'états UML (eux-mêmes adaptés des statecharts) dans SysML : voir cours suivant.