Esterel de A à Z

1. Principes, idées et styles pour la programmation réactive

Gérard Berry

Collège de France
Chaire Algorithmes, machines et langages

Cours du 7 février 2018

gerard.berry@college-de-france.fr
http://www-sop.inria.fr/members/Gerard.Berry

COLLÈGE DE FRANCE
1530

Pourquoi ce cours

Unifier, intégrer et approfondir des présentations partielles dans les cours précédents à Paris et Sophia-Antipolis

- 13/01/2010 : Parallélisme synchrone et vibratoire
- 16/04/2013 : Systèmes réactifs logiciels, le design du langage synchrone Esterel v5
- 23/04/2013 : La compilation logicielle d'Esterel v5
- 14/05/2013 : La conception de circuits synchrones et multi-horloges en Esterel v7
- 21/05/2013 : Synthèse matérielle et compilation logicielle d'Esterel v7
- 15/01/2015 : Esterel, de la recherche à l'industrie: la vision labo
- 15/01/2015 : Esterel, de la recherche à l'industrie: la vision industrielle

Principes, idées et styles pour la programmation réactive

Pièce en cinq actes,
avec un intermède, un entracte
et un épilogue

Acte 1

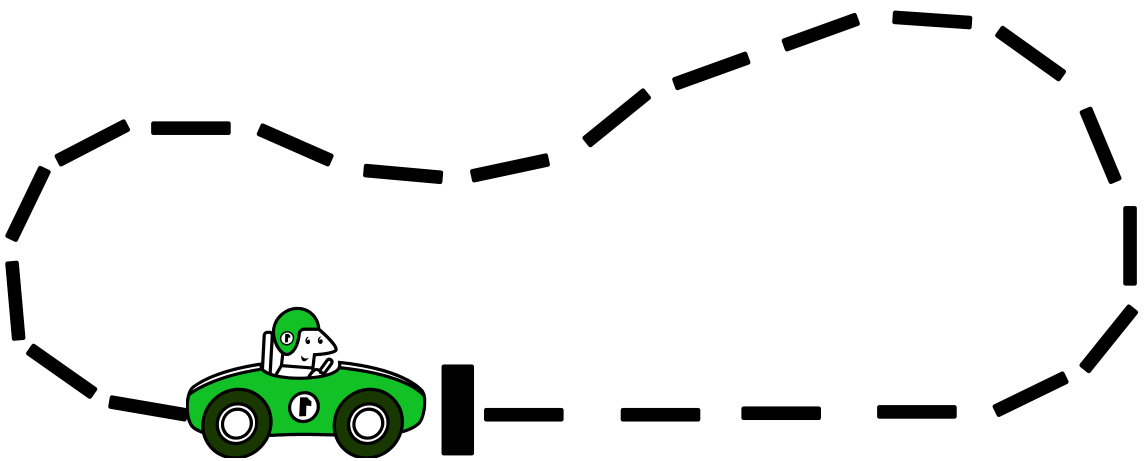
La jeunesse d'Esterel

Acte 1 : La jeunesse d'Esterel

1. De la voiture robot aux systèmes réactifs synchrones

Automatique → électronique → informatique

- 1982 : concours Micro Systèmes, naissance de l'idée
  - J-P. Marmorat, J-P. Rigault, J-M. Tanzi

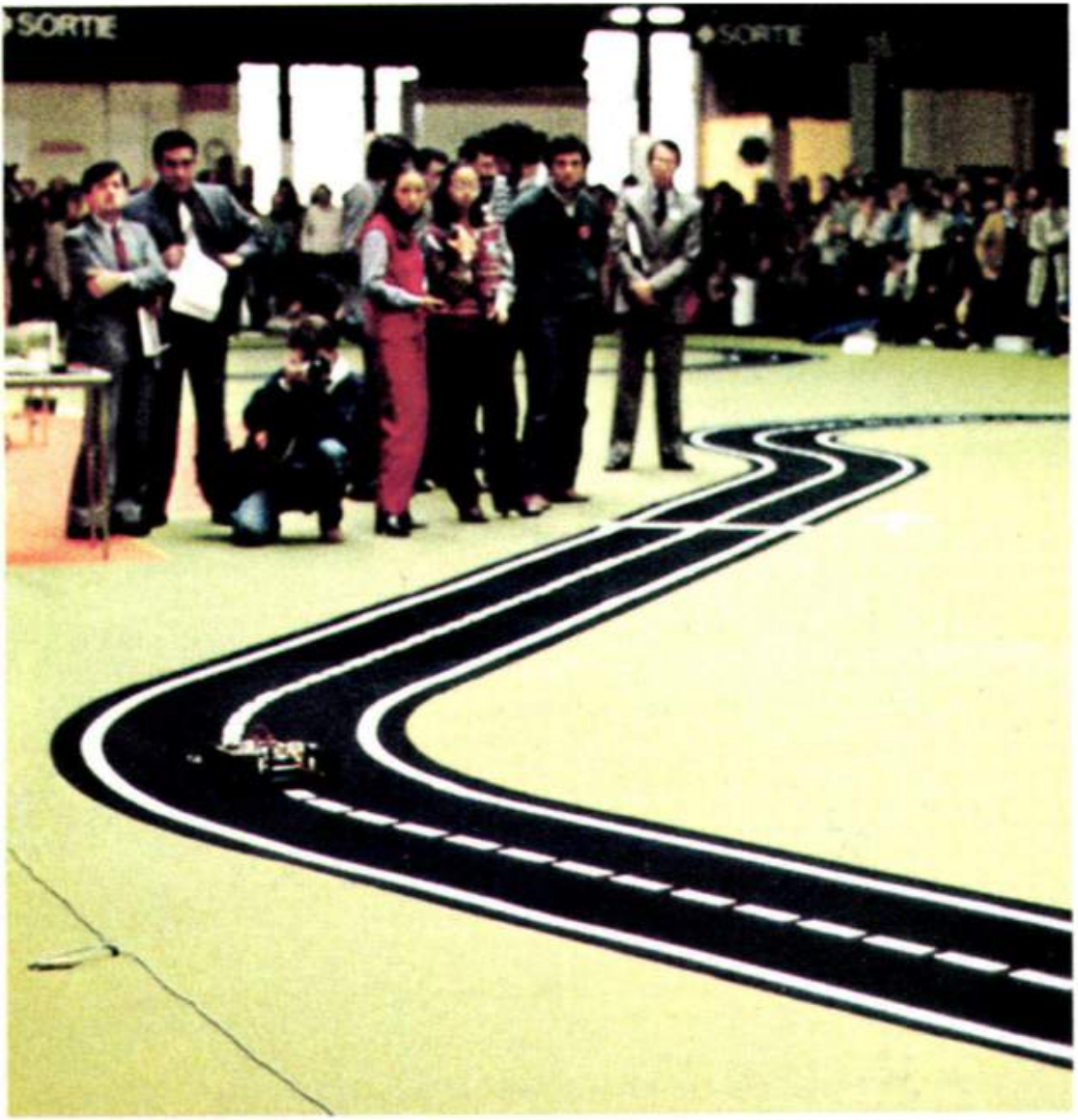


1 tour libre

2 chronométrés

- Voiture perfectionnée : gros moteur, freins à disques
- Automatique : accélération et freinage, coupure de virage
- Le grand jour : record du tour par hasard, mais défaite contre une voiture à vitesse constante, nez au sol…

Circuit trop facile, 2e place seulement



... Chronométrage sur le circuit II.

Le premier prix est attribué au Microtel Club de Bordeaux qui arrache la victoire en parcourant les 108 mètres représentant deux tours du grand circuit en 34 secondes et 15 centièmes.

La remise du deuxième prix à M. Wybo, 46 secondes pour les deux tours.

Micro Systèmes janvier-février 1981

La grande question

Comment spécifier et programmer une application de ce type ?

Elle allait m'occuper de 1982 à 2009+, avec beaucoup d'autres chercheurs et industriels !

Trois classes d'applications informatiques

- Transformationnelles : données → résultats
- Interactives : interaction au rythme de la machine

  time-sharing, OS, Web, etc.
- Réactives : au rythme du processus physique

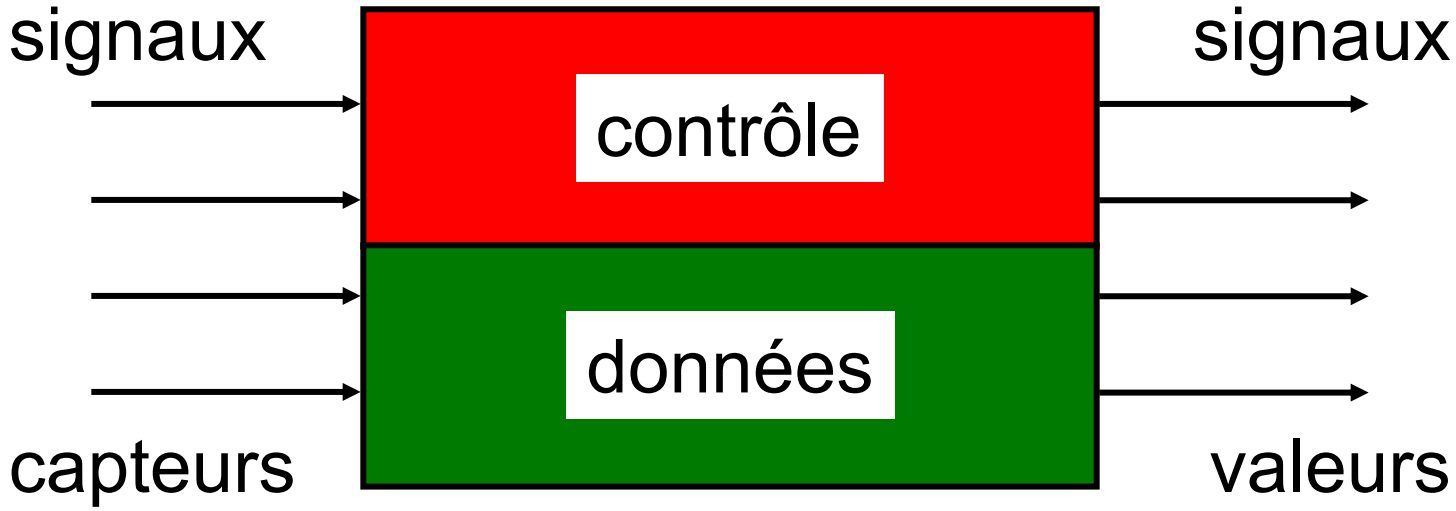
  temps-réel, robotique, automatismes, protocoles réseau, IHM, musique, etc.

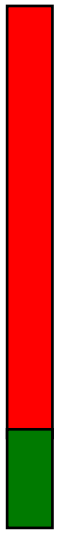
La grande majorité des méthodes et langages s'intéressaient au transformationnel / interactif

→ nouvelle communauté réactive

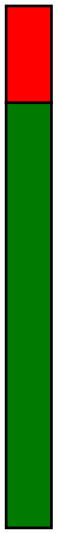
(initialement en France)

Systèmes réactifs





Esterel
Statecharts
Argos
SyncCharts
Scl, ScCharts
HipHop….



Lustre
Signal
SCADE 5
Jazz
Lucy-N
Zelus…

Esterel v7
SCADE 6
(Esterel Tech.)

Contraintes des systèmes réactifs

- Le parallélisme : faire plusieurs choses à la fois
- Le déterminisme : l'exécution doit être identique pour des suites d'entrées identiques
- Le temps-réel : les calculs doivent se faire à vitesse suffisante pour respecter les contraintes physiques (ex.: 1/100s pour un avion)

Attention : beaucoup d'applications réactives ne sont pas « temps-réel dur » !

La clef : synchronisme / vibration

Synchrone = temps 0

musiciens et spectateurs negligent la vitesse du son

Vibration = temps prévisible

les acousticiens s'occupent de la propagation du son

La clef : synchronisme / vibration

Si la pièce est assez petite

La vibration implémente bien le synchronisme

Mais si l'orchestre est trop grand

Les musiciens ont besoin de voir le chef d'orchestre

L'idée fondamentale du modèle synchrone

Réconcilier parallélisme et déterminisme.
en n'accordant aucun temps à l'administration :

Calculs et communications
conceptuellement instantanés !

Equivalent : acheter un ordinateur infiniment rapide !




Acte 1 : La jeunesse d'Esterel

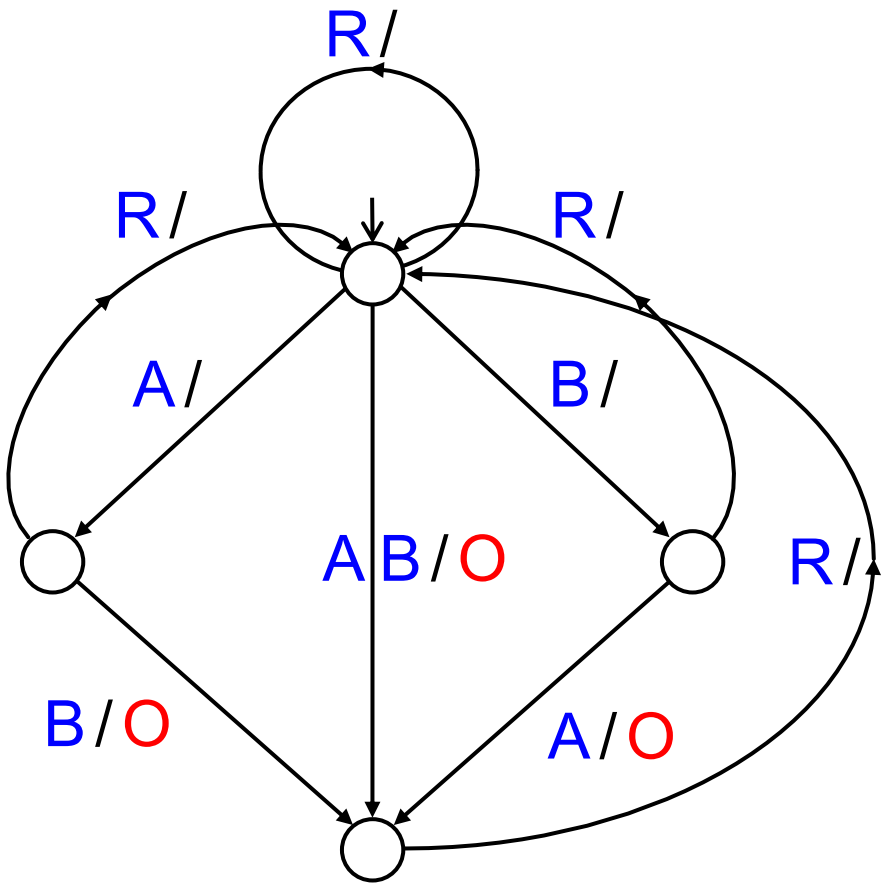
1. De la voiture robot aux systèmes réactifs synchrones
2. Les méthodes classiques, avantages et défauts

Automates : ABRO

Emettre O dès que
A et B sont arrivés.
Réinitialiser à chaque R

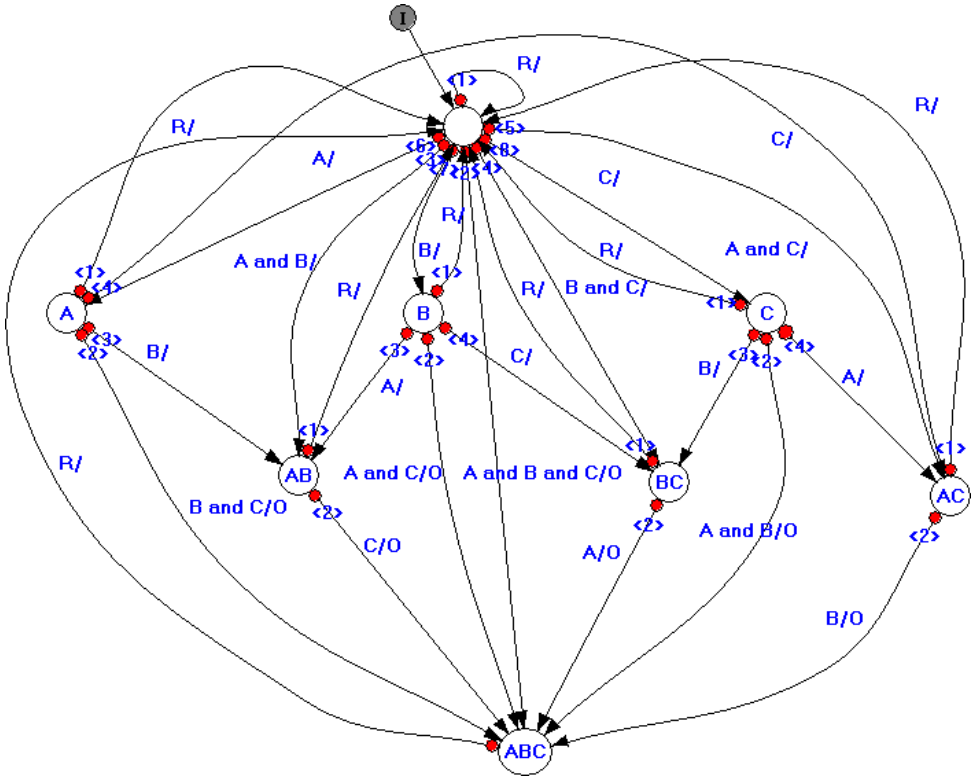
Beaucoup de copies :
3 A, 3 B, 4 R, 3 O

Ambigu :
quid si A R, B R, A B R ?



ABCRO : explosion exponentielle !

Emettre O dès que A, B et C sont arrivés.
Réinitialiser à chaque R



Tâches et systèmes d'exploitation

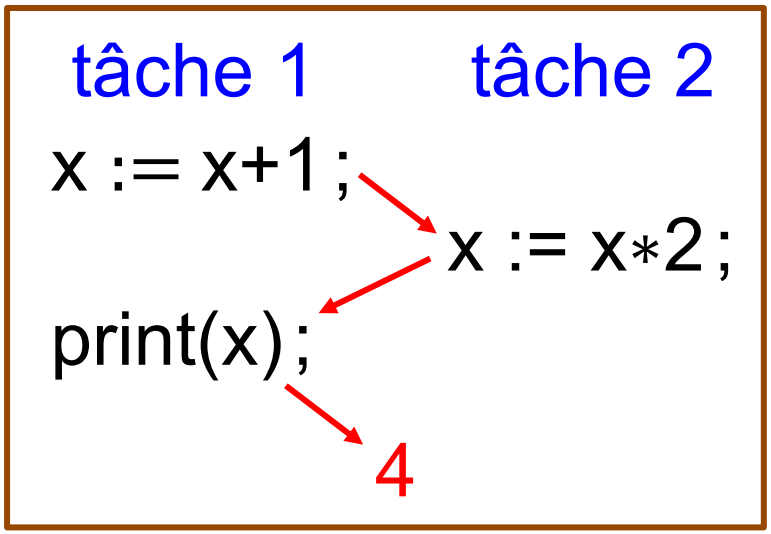
- Tâche unique périodique : automates programmables
  - déterminisme, mais expressivité faible
  - contrôle du temps rudimentaire (détection de dépassement)
- Tâches multiples avec ordonnancement fixe
  - déterminisme, puissance, automatisation (Sildex, Ptolemy)
  - critères variés : au plus tôt, *earliest deadline first*, etc.
  - difficulté de passage à l'échelle : réserver à l'implem. finale
- Tâches interruptibles, ordonnancement dynamique (OS classiques), réseaux asynchrones
  - indéterminisme de l'exécution
  - variables partagées difficiles à contrôler
  - impossibilité de vérifier les contraintes temporelles

Le problème des variables partagées

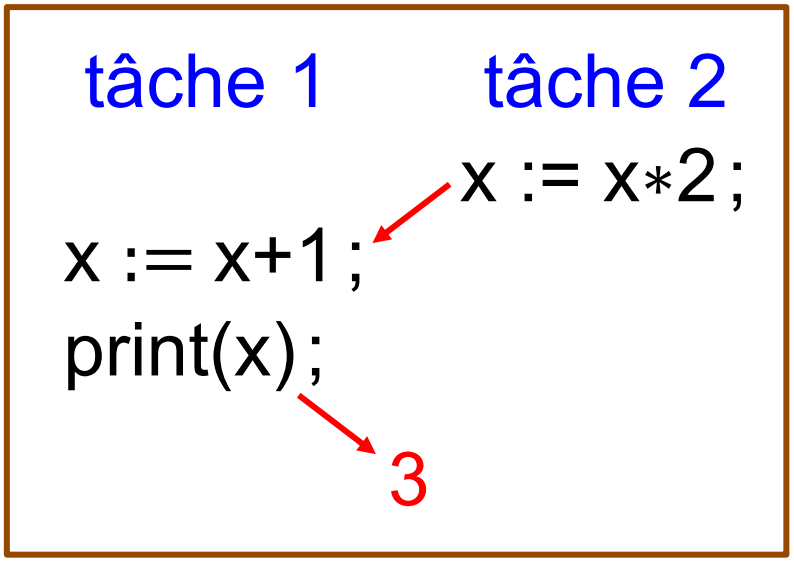
```
int x := 1 ;
```

tâche 1	tâche 2
x := x+1 ;	x := x∗2 ;
print(x) ;	

exécution 1



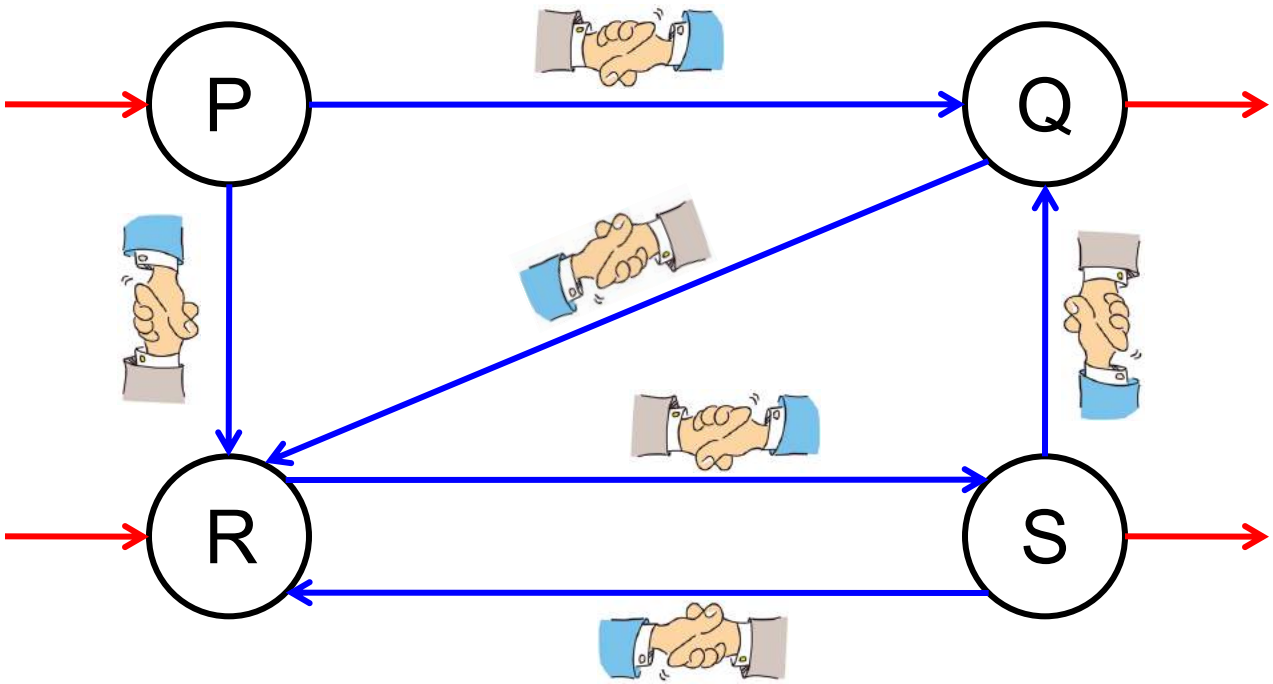
exécution 2



CSP, ADA : asynchronisme+ rendez-vous

- P et Q sont asynchrones, mais communiquent de façon localement synchrone
- A ce moment, chacun sait ce que sait l'autre
- Les rendez-vous sont ordonnancés de manière non-déterministe

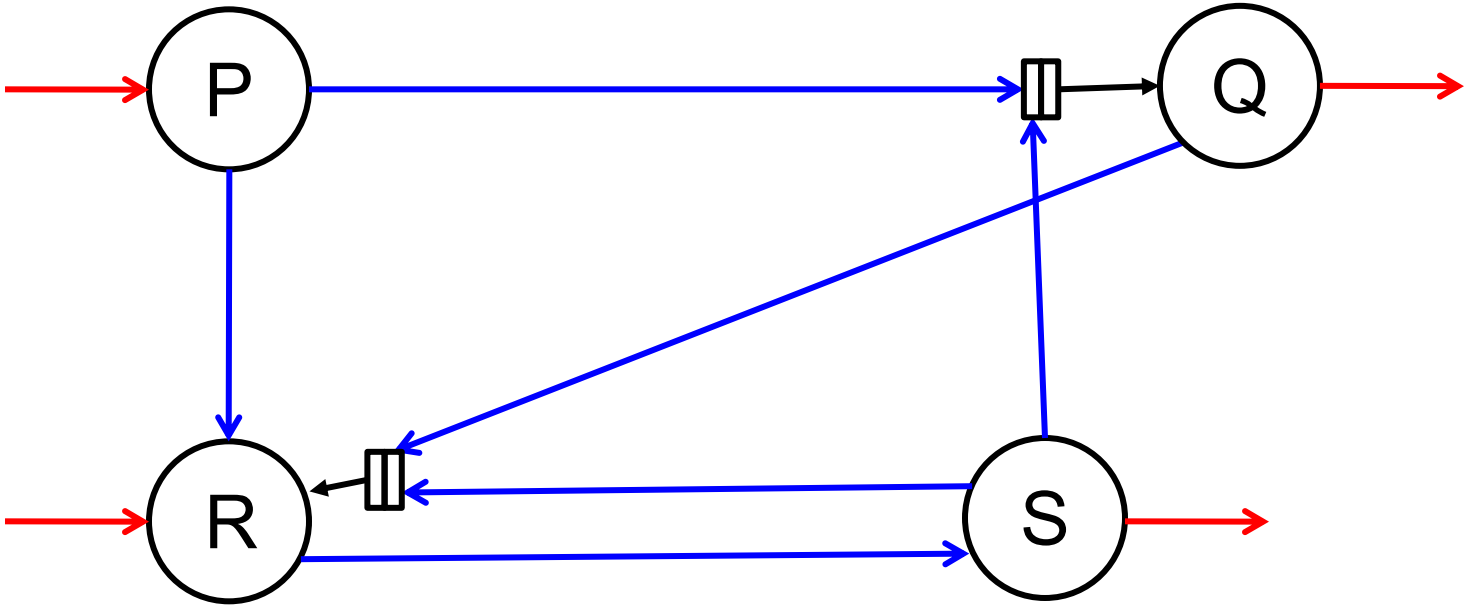
CSP : rendez-vous localement synchrone



Bien contrôlable, mais asychrone, lourd et lent

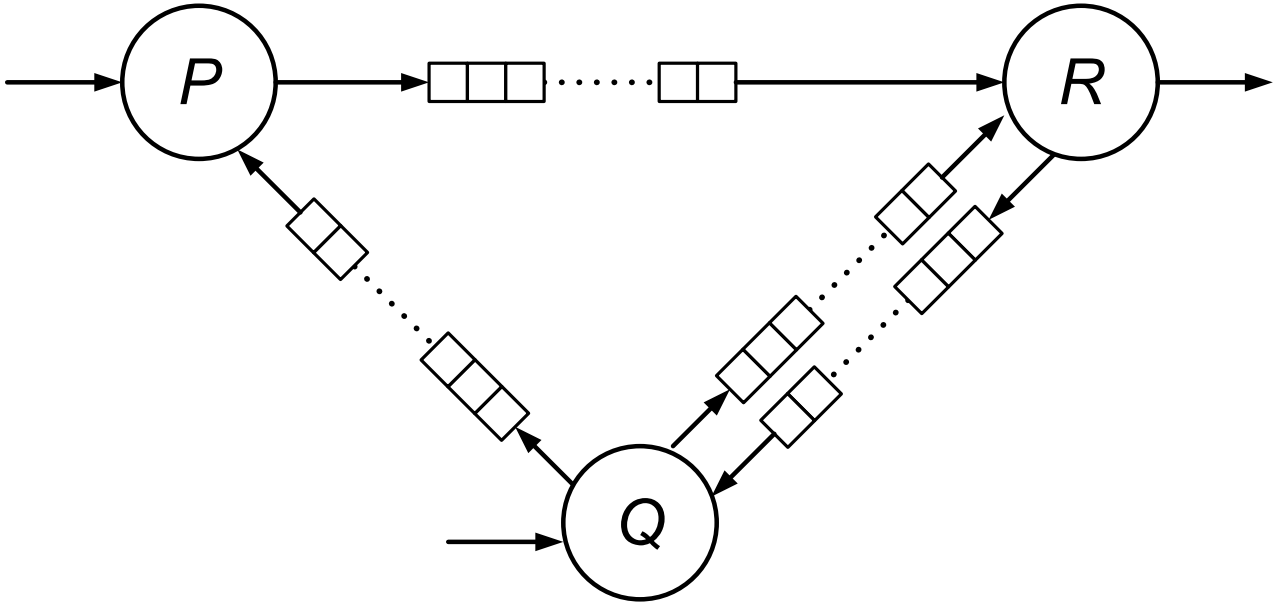
Voir cours du 2 avril 2013

ADA : ajout de files d'attente



Gestion d'exceptions compliquée
souvent restrictions d'utilisation

Réseaux de Kahn



- Nœuds séquentiels, ou récursivement réseaux de Kahn
- get bloquant, send non-bloquant

  $\Rightarrow$ files FIFO non bornées
- Asynchronisme des calculs, mais déterminisme des résultats

Élégants, superbe sémantique, mais restant asynchrones

Acte 1 : La jeunesse d'Esterel

1. De la voiture robot aux systèmes réactifs synchrones
2. Les méthodes classiques, avantages et défauts
3. Naissance et adolescence d'Esterel

Nouveaux principes de programmation
J-P. Marmorat, J-P. Rigault (1982)

- Mette les événements au centre de la programmation
  et les traiter de façon uniforme
  - entrées : seconde, clic_roue
  - événements dérivés : scotch, ligne_départ
- Définir le contrôle par des instructions temporelles
  - attendre événement
  - jusqu'à événement faire …
  - tous les événement faire …
- Parallélisme synchrone, le grand simplificateur
  - événements vus simultanément partout

Spécifier comme on pense et non pas comme fait la machine

```
Premier tour :
vitesse (10) ;
attendre ligne_départ ;
jusqu'à ligne_départ faire
   scotch_num := 0 ;
   tous les scotch faire
      scotch_num := scotch_num+1 ;
      longueur [scotch_num] := 0 ;
      tous les clic_roue faire
         longueur [scotch_num] := longueur [scotch_num] + 1 ;
```

préemption

+ suivi et mesures des courbes, etc.

1983 : Irruption de la science informatique

Esterel : towards a synchronous and semantically sound high-level language for real time applications
Gérard Berry, Sabine Moisan, Jean-Paul Rigault
Proc. IEEE 1883 Real-Time Systems Symposium

- Proposer un langage de spécification parlant réellement du temps et des événements
- Lui donner une vraie sémantique formelle
- Le faire évoluer vers un vrai langage de programmation compilable pour embarquer les programmes dans de vraies applications

Les balbutiements d'Esterel

affectation, appel de fonction

nothing

pause

emit S

if S then p fi

p ; q

loop p end

p // q

up-to c [S]

do p [; terminate]

[abnormal q] end

Jouer avec les temps

courir 100m
en moins de 15s

```
up-to 15 [Second]
do
   up-to 100 [Meter]
   do Run end;
   terminate
abnormal emit TooSlow
end
```

courir 15s
mais pas plus de 100m

```
up-to 100 [Meter]
do
   up-to 15 [Second]
   do Run end ;
   terminate
abnormal emit TooFast
end
```

Evolution de l'instruction de préemption

up-to *c* [S]
do p ;
~~[terminate]~~
[abnormal q] end

trop compliqué !

v2

do
  p
watching *c* S
[timeout q end]

v5 / v7

abort
  p
when *c* S
[do q end]

Jouer avec les temps

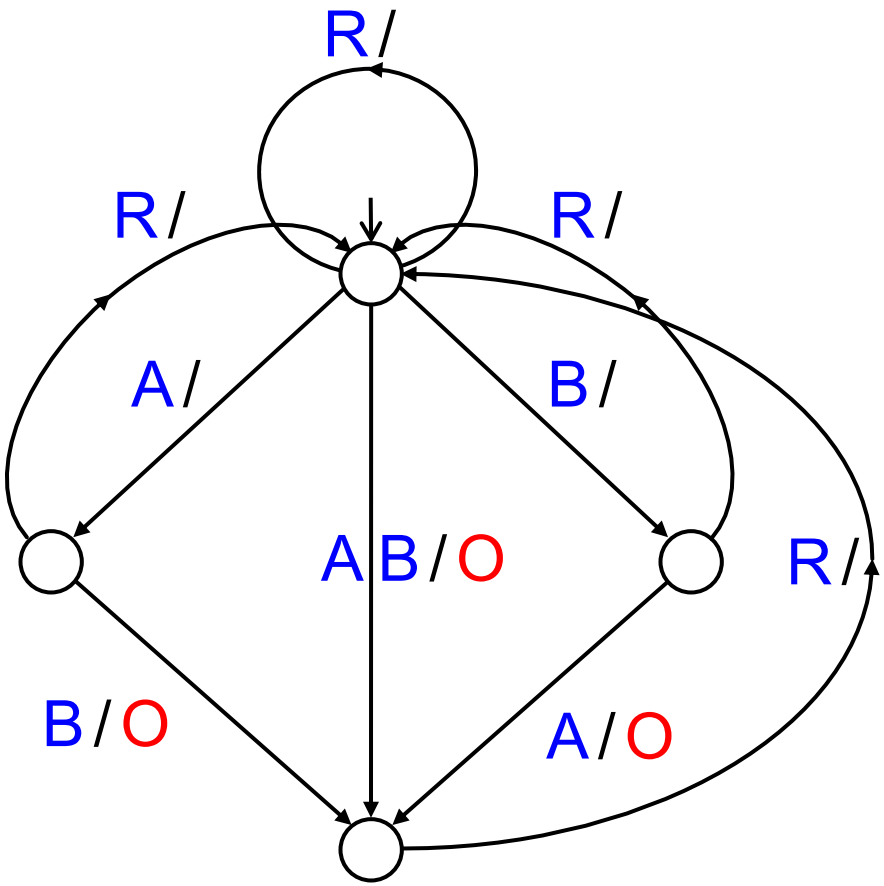
Courir 100m
en moins de 15s

```
abort
   abort
      run Run
   when 100 Meter
when 15 Second
   emit TooSlow
end abort
```

Courir 15s
mais pas plus de 100m

```
abort
   abort
      run Run
   when 15 Second
when 100 Meter
   emit TooFast
end abort
```

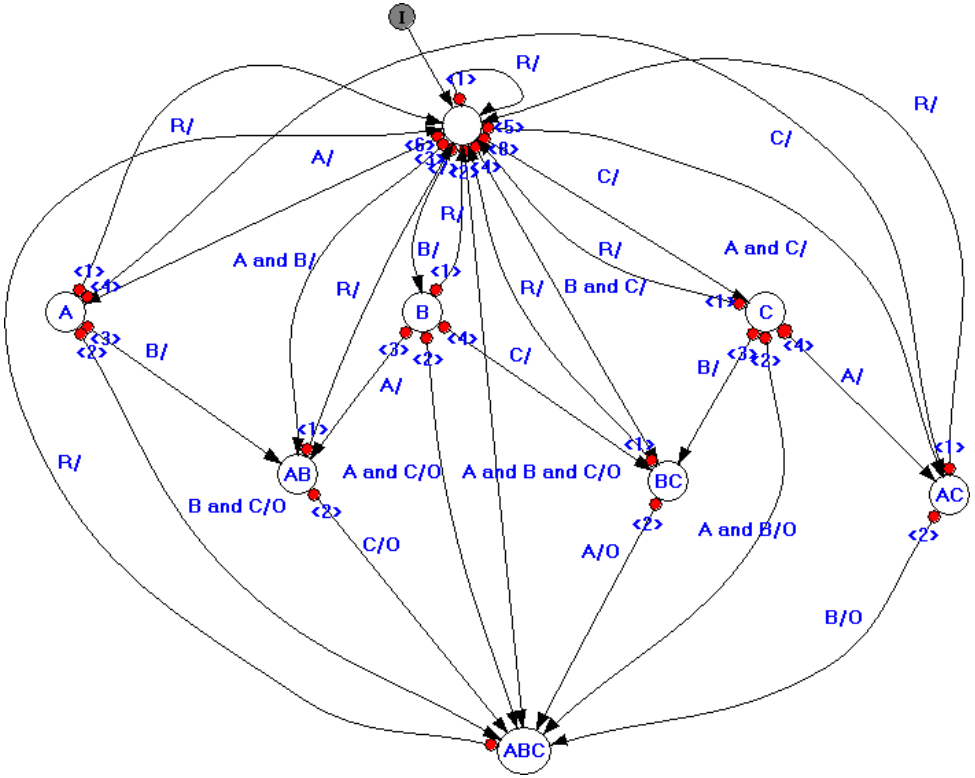
Abro en Esterel




```
loop
   abort
        {await A || await B};
        emit O;
        halt
   when R
end loop
```

Mais encore ambigu : quid de A R, A B R, etc?

ABCRO : de l'exponentiel au linéaire !




```
loop
   abort
      {   await A
        || await B
        || await C
      };
      emit O;
      halt
   when R
end loop
```

WTO = Write Things Once

Acte 2

Esterel Pur

Acte 2 : Esterel Pur

1. Un premier noyau

Le noyau synchrone de base

1. Exécution par une série d'instants sans épaisseur
2. Signaux S, S1, S2,…, chacun ayant un statut présent ou absent à chaque instant, vu identiquement par toutes les instructions
3. Un premier jeu d'instructions très simples

```
p ::= nothing
    | pause
    | emit S
    | if S then q else r end
    | p ; q
    | loop p end
    | p || q
```

Exemple

```
input I, J ;
output preI, X, Y ;
   loop
      if I then pause ; emit preI
      else pause end
   end loop
||
   loop
      if I then emit X end ;
      pause ;
      if J then emit Y end ;
      pause ;
      if J then emit X end
   end
```

Exemple, 1e instant

```
input I, J;
output preI, X, Y;
   loop
      if I then pause; emit preI
      else pause end
   end loop
||
   loop
      if I then emit X end;
      pause;
      if J then emit Y end;
      pause;
      if J then emit X end
   end
```

1 : I J → preI X Y

Exemple, début du 2e instant

```
input I, J;
output preI, X, Y;
   loop
      if I then pause; emit preI
      else pause end
   end loop
||
   loop
      if I then emit X end;
      pause;
      if J then emit Y end;
      pause;
      if J then emit X end
   end
```

1 : I J → preI X Y

2 : I J → preI X Y

Exemple, début du 2e instant

```
input I, J;
output preI, X, Y;
   loop
      if I then pause; emit preI
      else pause end
   end loop
||
   loop
      if I then emit X end;
      pause;
      if J then emit Y end;
      pause;
      if J then emit X end
   end
```

1 : I J → preI X Y

2 : I J → preI X Y

Exemple, fin du 2e instant

```
input I, J;
output preI, X, Y;
   loop
      if I then pause; emit preI
      else pause end
   end loop
||
   loop
      if I then emit X end;
      pause;
      if J then emit Y end;
      pause;
      if J then emit X end
   end
```

1 : I J → preI X Y

2 : I J → preI X Y

Exemple, fin du 2e instant

```
input I, J ;
output preI, X, Y ;
   loop
      if I then pause ; emit preI
      else pause end
   end loop
||
   loop
      if I then emit X end ;
      pause ;
      if J then emit Y end ;
      pause ;
      if J then emit X end
   end
```

1 : I J → preI X Y

2 : I J → preI X Y

Exemple, fin du 2e instant

```
input I, J;
output preI, X, Y;
   loop
      if I then pause; emit preI
      else pause end
   end loop
||
   loop
      if I then emit X end;
      pause;
      if J then emit Y end;
      pause;
      if J then emit X end
   end
```

1 : I J → preI X Y

2 : I J → preI X Y

Exemple, fin du 2e instant

```
input I, J;
output preI, X, Y;
   loop
      if I then pause; emit preI
      else pause end
   end loop
||
   loop
      if I then emit X end;
      pause;
      if J then emit Y end;
      pause;
      if J then emit X end
   end
```

1 : I J → preI X Y

2 : I J → preI X Y

Exemple, fin du 2e instant

```
input I, J;
output preI, X, Y;
   loop
      if I then pause; emit preI
      else pause end
   end loop
||
   loop
      if I then emit X end;
      pause;
      if J then emit Y end;
      pause;
      if J then emit X end
   end
```

1 : I J → preI X Y

2 : I J → preI X Y

Exemple, début du 3e instant

```
input I, J ;
output preI, X, Y ;
   loop
      if I then pause ; emit preI
      else pause end
   end loop
||
   loop
      if I then emit X end ;
      pause ;
      if J then emit Y end ;
      pause ;
      if J then emit X end
   end
```

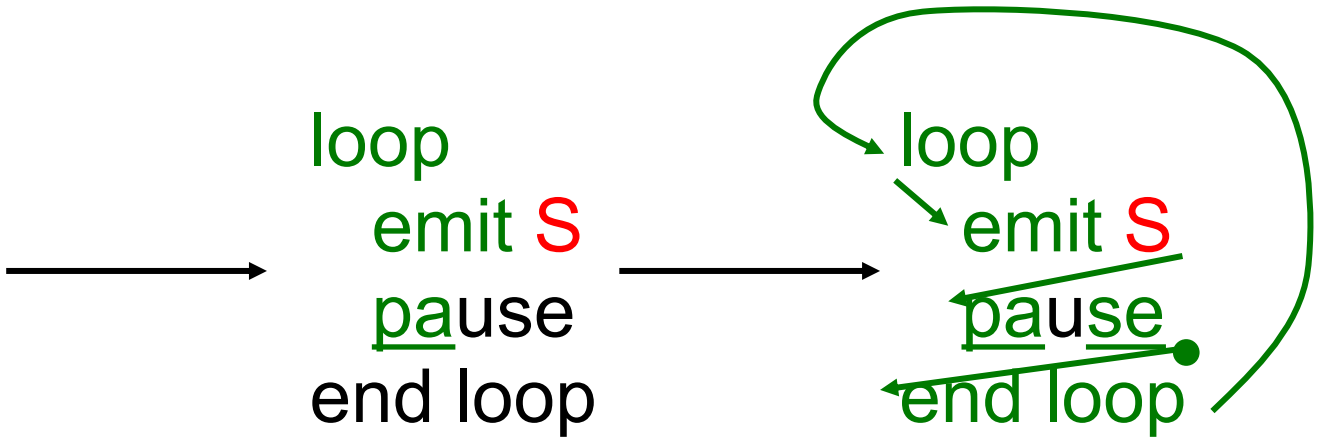
1 : I J → preI X Y
2 : I J → preI X Y
3 : I J → preI X Y

X émis 2 fois,
pas de problème

Deux instructions dérivées : halt et sustain

halt → loop pause end

sustain S →
```
loop
   emit S ;
   pause
end loop
```



Acte 2 : Esterel Pur

1. Un premier noyau
2. Les instructions de préemption

Première erreur de design

- Choix initial : « upto S do p end » se termine immédiatement si S est présent à l'instant de démarrage du upto

- Définition évidente et utile :

  await S = upto S do nothing end

- Mais alors, gros problème :

  await S ≈ « await S ; await S » ≠ await 2 S

⇒ upto ne doit pas terminer instantanément !

Pour la préemption, il est indispensable de bien traiter les incontournables problèmes de poteaux

Synchronisme ⟹ problèmes de poteaux

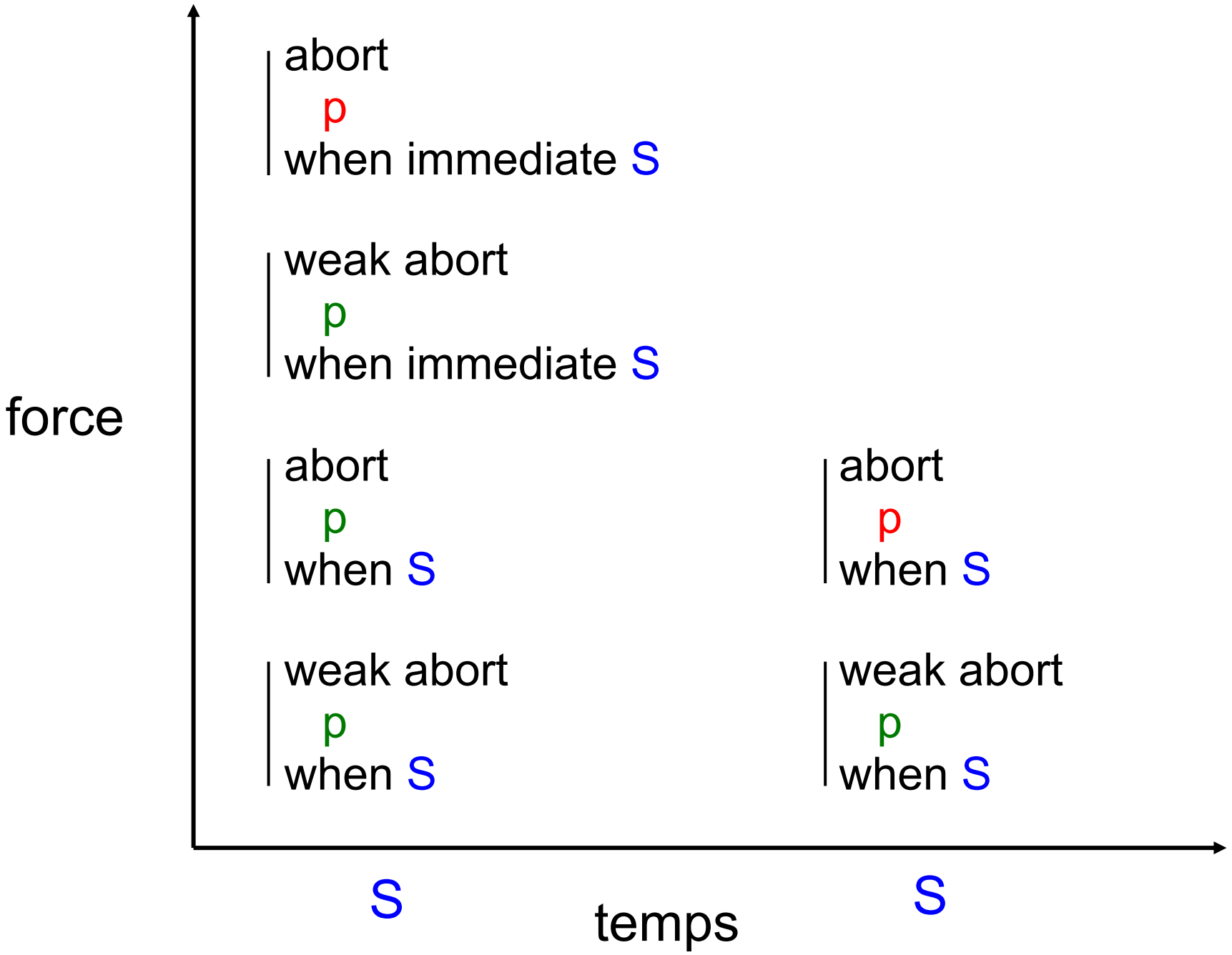
```
abort
   p
when S
```

- Si S est là au lancement de l'instruction abort, faut-il en tenir compte ou pas ?
  → préemption immédiate ou retardée
- A l'instant où S arrive, faut-il exécuter p ou pas ?
  → préemption faible ou forte
- Si p se termine, faut-il que l'abort se termine ou pas ?
  → oui, car on peut toujours écrire « p ; halt »
  (upto utilisait terminate, moche)

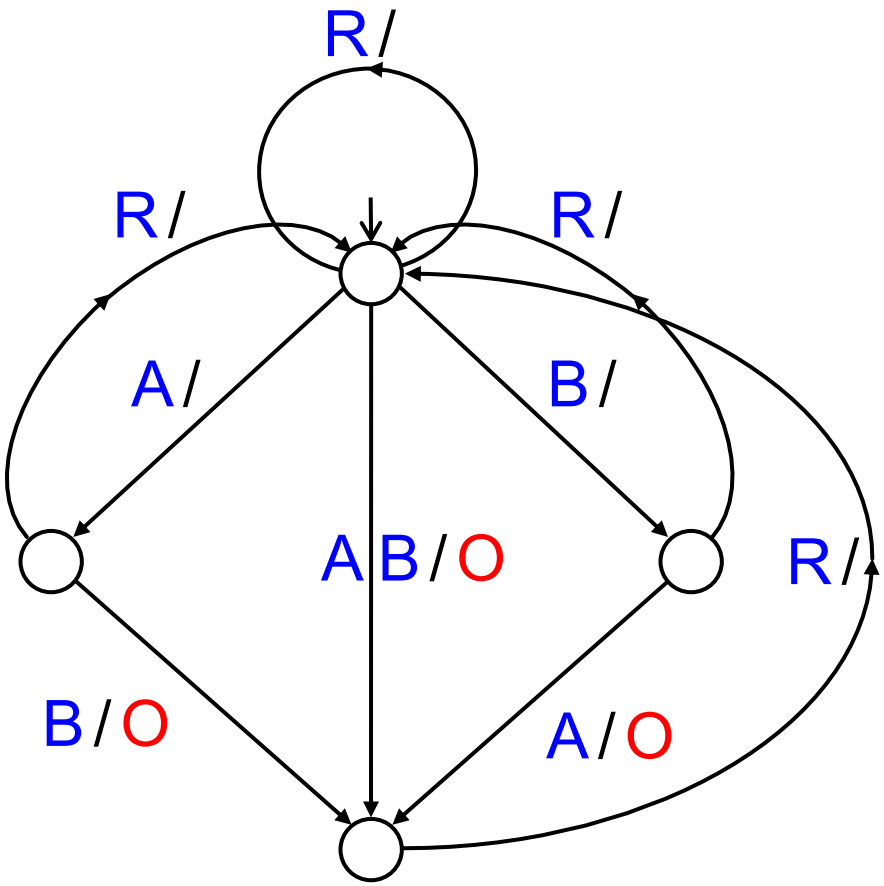
abort simple : retardé mais fort

```
abort
   p
when S
```

- La préemption est retardée :
  à l'instant initial, S n'est pas testé
  et p est exécuté dans l'instant
- La préemption est forte :
  au moment où S devient présent,
  l'abort termine avec p non exécuté dans l'instant
- Les 3 autres cas : ajout de « weak » / « immediate »


abort
p
when immediate S
weak abort
p
when immediate S
abort
p
when S
weak abort
p
when S
abort
p
when S
weak abort
p
when S
force
S
temps
S

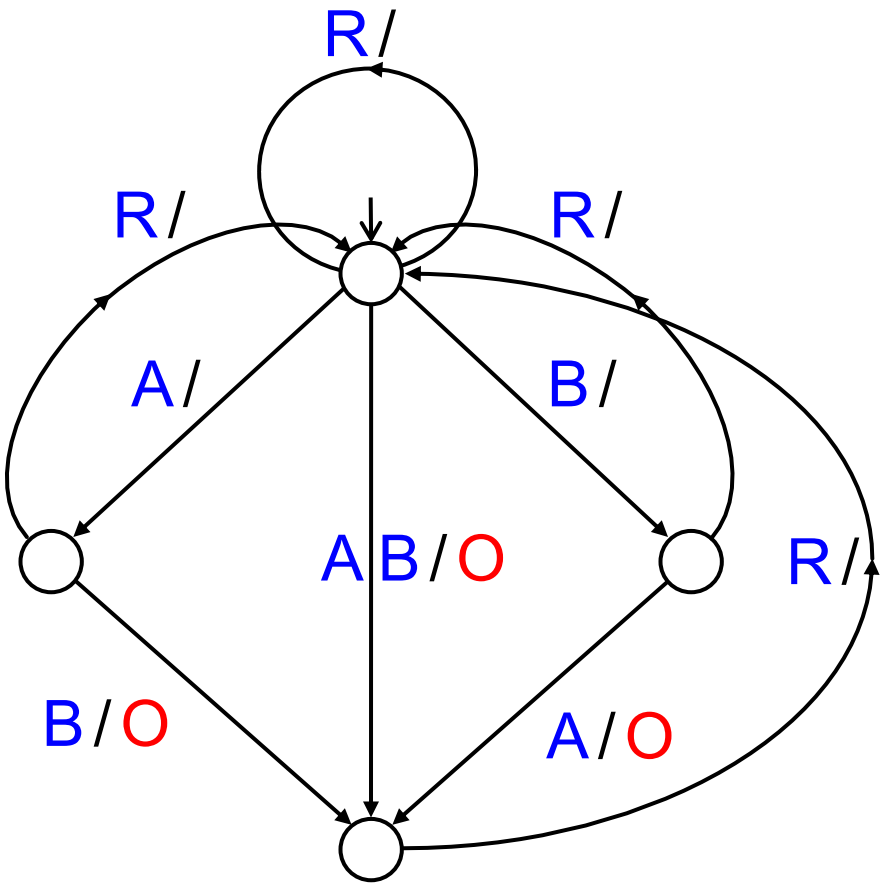
Amibguïté résolue !




```
loop
   abort
        {await A || await B};
        emit O;
        halt
   when R
end loop
```

abort : AR, BR, ABR → reset mais pas O

Amibguïté résolue !




```
loop
   weak abort
        {await A || await B};
        emit O;
        halt
   when R
end loop
```

weak abort : A R, B R, A B R → reset et O

Quelles primitives choisir ?

1. « abort » fort et retardé est un bon premier choix
2. « abort … when immediate », « await », « await immediate », et les boucles temporelles « loop-each » et « every » s'en déduisent facilement
3. « weak abort » va se déduire d'une instruction de contrôle « trap », plus générale
4. Avec « trap », on pourra déduire « abort » de « suspend », plus général (^C vs. ^Z en Unix)

A la fin, il ne restera plus que trap et suspend !

Immediate et await

```
abort
   p
when immediate S
```
→
```
if S then
   nothing
else
   abort p when S
end
```

await S → abort halt when S

await immediate S → abort halt when immediate S

Les boucles temporelles

- Démarrage immédiat :

  loop p each S

  → loop
  abort p when S
  end loop

- Démarrage au premier S :

  every [immediate] S do p end

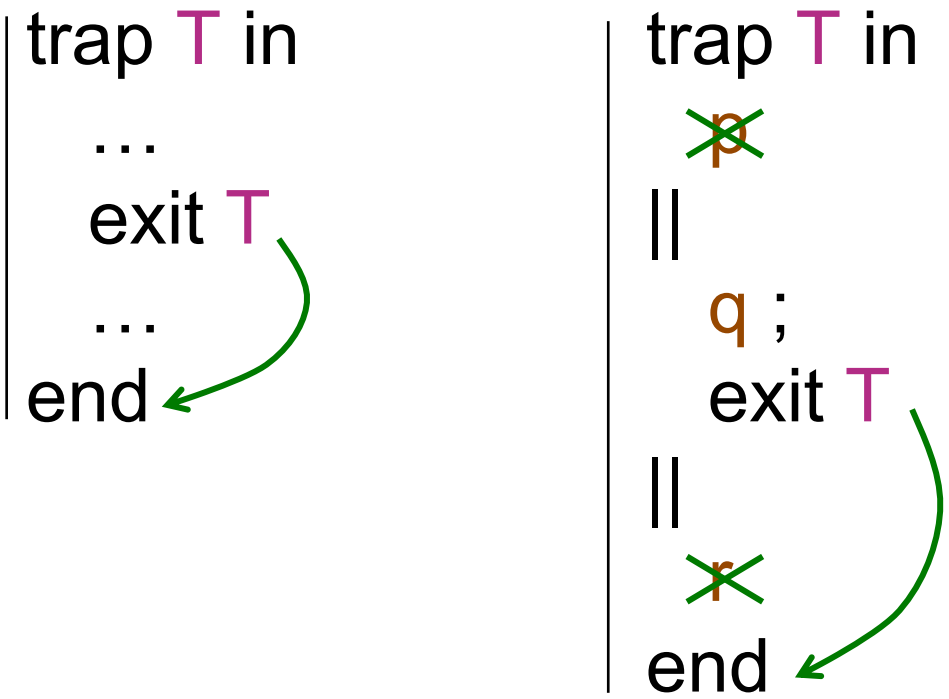
  → await [immediate] S ;
  loop p each S

⚠ Avec weak abort, p exécuté 2 fois quand S arrive ⚠
on dit que p se réincarne instantanément, voir cours 5

Acte 2 : Esterel Pur

1. Un premier noyau
2. Les instructions de préemption
3. Les trappes pour la préemption faible

Les trappes, inspirées de catch / throw en Lisp



Préemption faible :

à l'instant où « exit T » est exécuté,

p et r travaillent une dernière fois

(cigarette du condamné)

Traitement de la sortie de trappe

```
trap T in
   p
handle T do
   q
end
```

→

```
trap Terminate in
   trap T in
      p ; exit Terminate
   end ;
   q
end
```

Trappes imbriquées et exits parallèles

```
output X, Y ;
trap T in
   trap U in
      p ; exit T
   ||
      q ; exit U
   end trap ;
   emit X
end trap ;
emit Y
```

Trappes imbriquées et exits parallèles

```
output X, Y;
trap T in
   trap U in
      p; exit T
   ||
      q; exit U
   end trap;
   emit X
end trap;
emit Y
```

p se termine avant q → T → Y

Trappes imbriquées et exits parallèles

```
output X, Y;
trap T in
   trap U in
     p; exit T
   ||
      q; exit U
   end trap;
   emit X
end trap;
emit Y
```

q se termine avant p → U → X Y

Trappes imbriquées et exits parallèles

```
output X, Y;
trap T in
   trap U in
      p; exit T
   ||
      q; exit U
   end trap;
   emit X
end trap;
emit Y
```

p et q se terminent ensemble

→ T ~~U~~ → Y

Seule compte la trappe
la plus extérieure

Traduction de « weak abort »

- p actif une dernière fois lors de la préemption (cigarette du condamné)

```
weak abort
   p
when [immediate] S
```

→

```
trap Terminate in
   p ; exit Terminate
||
   await [immediate] S do
      exit Terminate
   end
end
```

Traitement des abort activés

```
[weak] abort
   p
when [immediate] S do →
   q
end
```

```
trap Terminate in
   [weak] abort
      p ; exit Terminate
   when [immediate] S ;
   q
end trap
```

Distraction juste pour garder la numérotation des transparents de la vidéo (le 67 était faux, voir le nouveau 66 qui couvre tous les cas)

Voici pourquoi beaucoup de pôvres auditeurs
ont raté ce double cours du 7 février 2018

Acte 2 : Esterel Pur

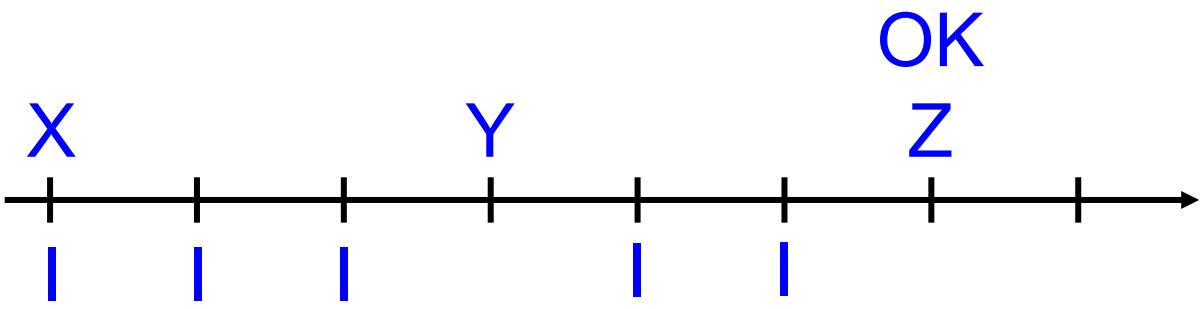
1. Un premier noyau
2. Les instructions de préemption
3. Les trappes
4. La suspension

suspend p when S (1993)

- Idée : ne pas tuer p, mais le suspendre quand S est présent
  - la suspension est retardée : p est exécuté au premier instant
  - ensuite :
    1. p est exécuté à tous les instants où S est absent ;
       si p se termine, alors le suspend se termine aussi
    2. p n'est pas exécuté à tous les instants où S est présent ;
       p garde son état, et le suspend ne se termine pas

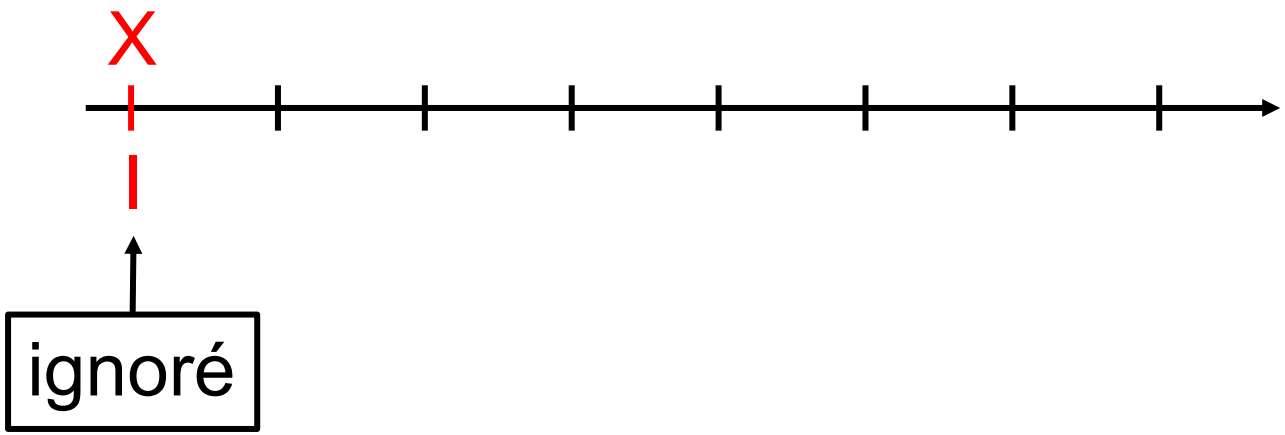
Exemple de suspension

```
suspend
    emit X;
    pause;
    emit Y;
    pause;
    emit Z
when I;
emit OK
```



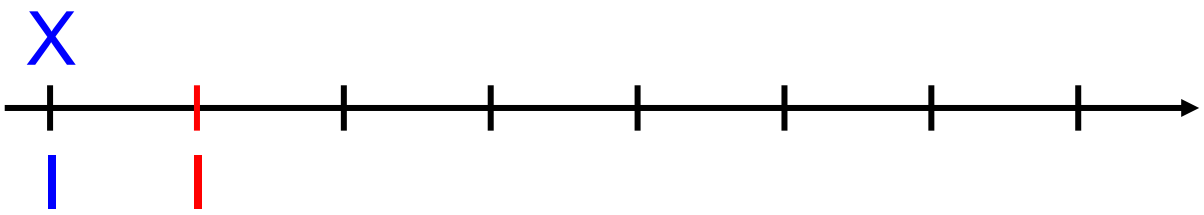
Exemple de suspension

```
suspend
      emit X;
      pause;
      emit Y;
      pause;
      emit Z
when I;
emit OK
```



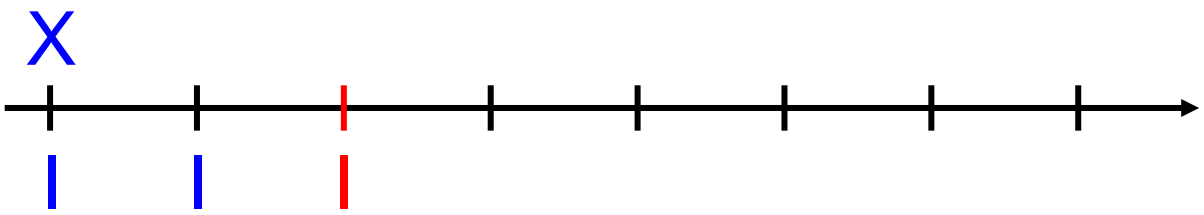
Exemple de suspension

```
suspend
      emit X;
      pause;
      emit Y;
      pause;
      emit Z
when I;
emit OK
```



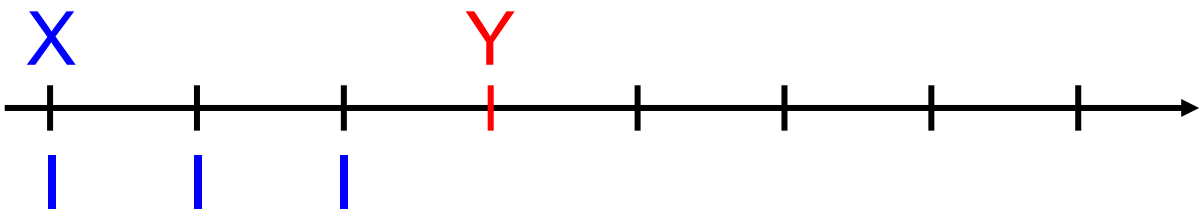
Exemple de suspension

```
suspend
      emit X;
      pause;
      emit Y;
      pause;
      emit Z
when I;
emit OK
```



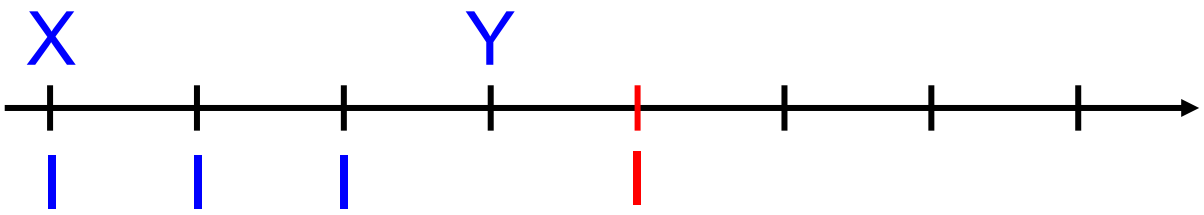
Exemple de suspension

```
suspend
      emit X;
      pause;
      emit Y;
      pause;
      emit Z
when I;
emit OK
```



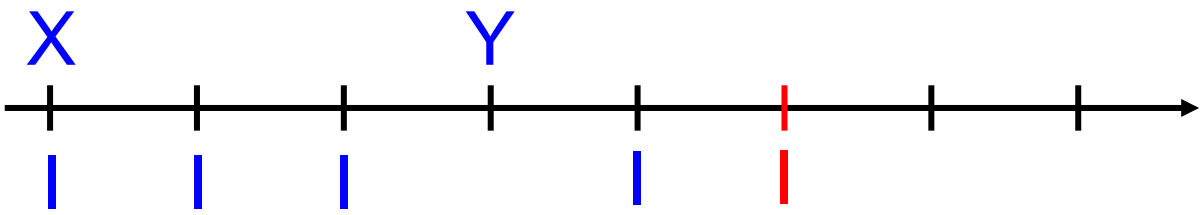
Exemple de suspension

```
suspend
      emit X;
      pause;
      emit Y;
      pause;
      emit Z
when I;
emit OK
```



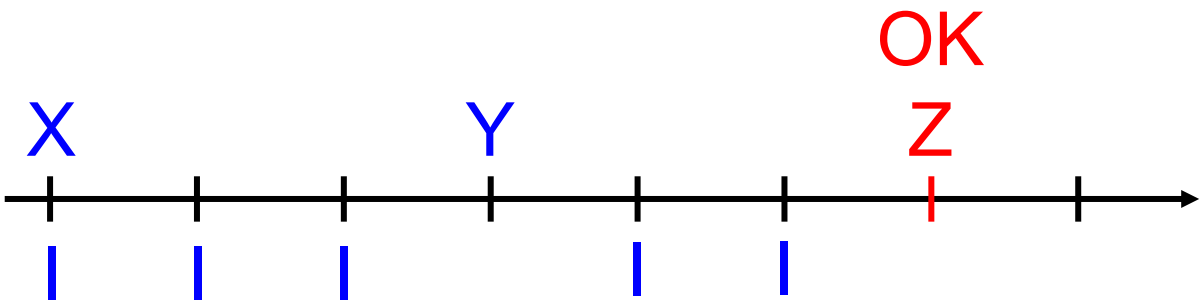
Exemple de suspension

```
suspend
      emit X;
      pause;
      emit Y;
      pause;
      emit Z
when I;
emit OK
```



Exemple de suspension

```
suspend
      emit X;
      pause;
      emit Y;
      pause;
      emit Z
when I;
emit OK
```



Abort se déduit de suspend

abort p when S ⟶

```
trap Terminate in
   suspend p when S ;
   exit Terminate
||
   await S ;
   exit Terminate
end trap
```

Attention: oubli du premier exit dans la vidéo, désolé le abort *doit se terminer quand* p *termine*

Suspension immédiate

```
suspend
   p
when immediate S
```



```
await immediate not S;
suspend
   p
when S
```

await immediate not S


```
trap Terminate in
   loop
      if S then nothing
      else exit Terminate
      end if;
      pause;
   end loop
end trap
```

Suspension immédiate

```
suspend
   p
when immediate S
```

```
await immediate not S;
suspend
   p
when S
```

await immediate not S

```
trap Terminate in
   loop
      if S else exit Terminate end;
      pause;
   end loop
end trap
```

Dans if, on peut omettre soit « then » soit « else », commode !

Acte 2 : Esterel Pur

1. Un premier noyau
2. Les instructions de préemption
3. Les trappes
4. La suspension
5. *Esterel Pur*, enfin !

Le noyau d'Esterel pur

`nothing`	*ne fait rien et termine instantanément*
`emit` S	*émet le signal* S
`pause`	*arrête l'exécution, puis redémarre au prochain instant*
p ; q	*exécute* p *puis* q *si et quand* p *termine*
`loop` p `end`	*répète* p *indéfiniment*
p || q	*exécute* p *et* q *en parallèle synchrone*
`trap` T `in` p `end`	*déclare et gère la trappe* T *dans* p
`exit` T^k	*lève la trappe* T *d'index* k
`if` S `then` p `else` q `end`	*exécute* p *si* S *est présent,* q *sinon*
`suspend` S `when` p	*lance* p *puis le suspend si* S *est présent termine si* p *termine et premier instant ou* S *absent*
`signal` S `in` p `end`	*déclare le signal* S *local dans* p

Intermède

Données, variables, et signaux valués

Les données

- Esterel v5 : Types de base (boolean, integer, string) avec opérations classiques
- Types abstraits définis dans le langage hôte
- Fonctions et procédures connues par leur nom et définies seulement dans le langage hôte
- Signaux valués à valeur unique dans chaque instant
- Variables non partagées

- Esterel v7 : types et opérations beaucoup plus riches, avec tableaux, boucles statiques, etc.

  (voir cours du 14 mai 2013)

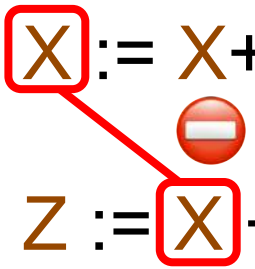
Variables non partagées

Si une variable est lue dans une branche d'un parallèle, elle ne peut pas être écrite dans une autre branche

```
var X : integer in
   X: = 0 ;
   {
      Y := X+1
    ||
      Z := X+2
    };
    X := X+1
end var
```

OK

```
var X : integer in
   X: = 0 ;
   {
      X := X+1
    ||
      Z := X+2
    };
    X := X+1
end var
```



NOK

Les signaux valués

- Signaux portant à tout instant une valeur unique
- La valeur persiste d'un instant à l'autre, mais, en cas d'émission, seule la nouvelle valeur compte
- La valeur courante de S est ?S, sa valeur précédente pre(?S)

```
output X : integer;
signal Count : integer :=0;
    emit Count(5);
    pause;
    emit X(?Count);
    pause;
    emit Count(pre(?Count)+1)
    emit X(?Count)
end signal
```

	?X = undef
?Count = 0	?X = undef
?Count = 5	?X = undef
?Count = 5	?X = 5
?Count = 6	?X = 5
	?X = 6

Signaux simples ou combinés

- Simple : au plus émetteur à chaque instant

  output S : integer ;
  emit S(1) || emit S(2) → single signal emitted twice

- Multiple : combinaison associative commutative

  output S : combine integer with + ;
  emit S(1) || emit S(2) → ?S = 3

Levez chacun le doigt et je saurais combien vous êtes !

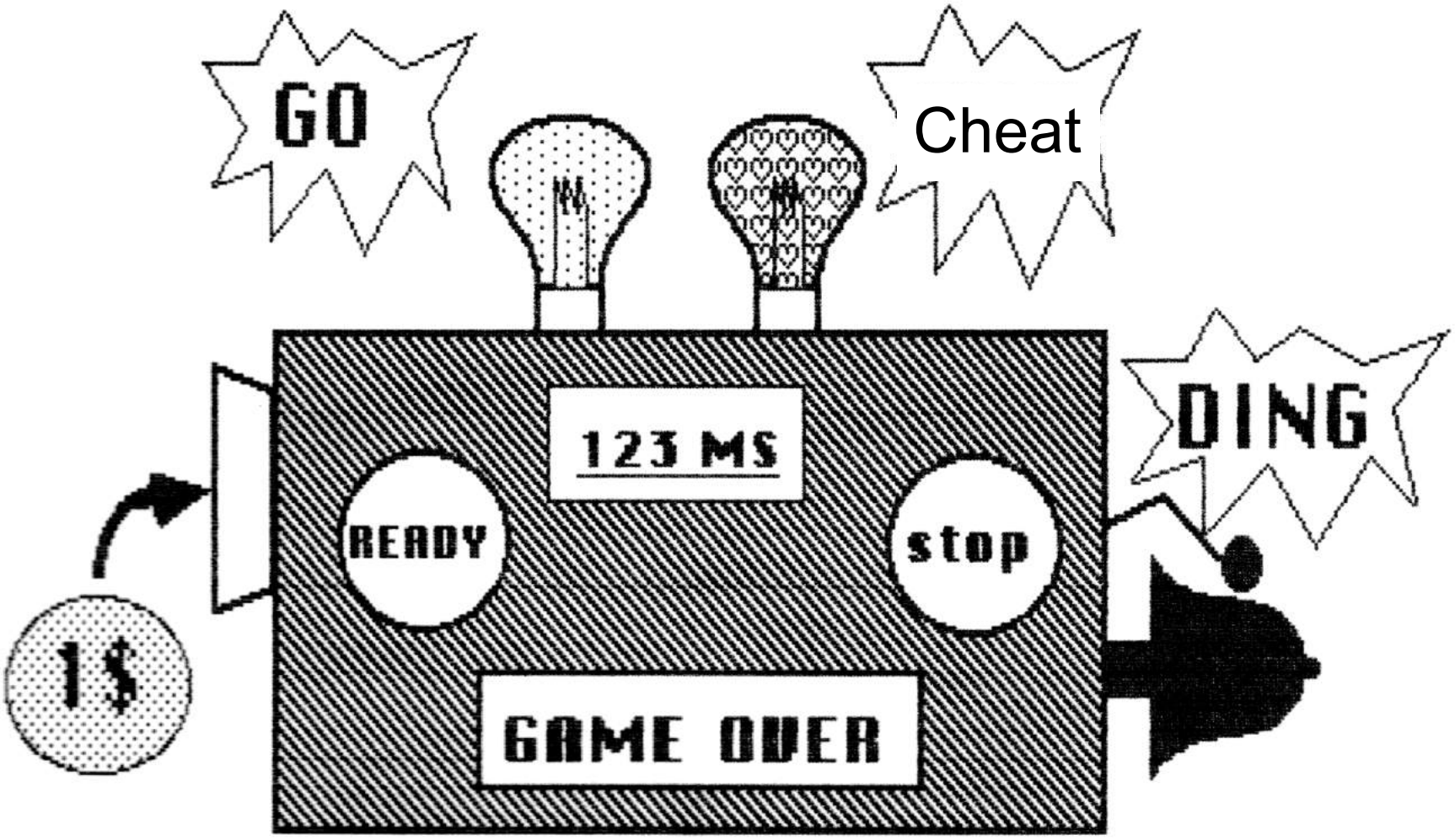
Acte 3

Où l'on mesure ses réflexes avant de lire l'heure

Acte 3 :
Où l'on mesure ses réflexes, avant de lire l'heure

1. De bons réflexes pour une bonne santé

Le jeu de réflexes



```
module ReflexGame :
% interface de données
constant LimitTime,
         MeasureNumber,
         PauseLength : integer ;
function Random () : integer ;
% interface de signaux
input MS, Coin, Ready, Stop ;
relation MS # Coin # Ready,
         Coin # Stop,
         Ready # Stop ;
output Display : integer,
       GoOn, GoOff,
       GameOverOn, GameOverOff,
       CheatOn, CheatOff,
       RingBell ;
```

Corps du module

```
% initialisations au démarrage
emit Display(0);
emit GoOff ; emit CheatOff ;
emit GameOverOn ;
% boucle sur un jeu simple
every Coin do
   % initialisation du jeu
   emit Display(0);
   emit GoOff ; emit CheatOff ;
   emit GameOverOff ;
   % le jeu proprement dit
    < game>
end every
end module
```

< game>

```
trap EndGame, Cheater in
   signal ReflexTime : integer, AverageReflexTime : integer in
      % module de librairie pour calculer le temps moyen
      run Average [ signal ReflexTime / I,
                                 AverageReflexTime / O ]
   ||
      repeat MeasureNumber times
         < phase 1 : attendre Ready > ;
         < phases 2-3 :attendre un délai aléatoire puis Stop >
      end repeat ;
      % phase 4 : afficher le temps après un délai final
      await PauseLength MS;
      emit Display (?AverageReflexTime)
   end signal
handle Cheater do
      emit CheatOn
end trap ; % EndGame et Cheater
emit GameOverOff
```

< *phase 1 : attendre Ready* >

```
% phase 1: attendre Ready
abort
   abort
      every Stop do emit RingBell end
   when Ready
when LimitTime MS do
   exit EndGame
end abort;
```

< *phases 2-3 : attendre délai aléatoire puis STOP* >

code mis en facteur (WTO)

```
trap EndMeasure in
   every Ready do emit RingBell end
||
   < phase 2 : attendre délai aléatoire > ;
    <phase 3 : attendre Stop >
end trap
```

< *phase 2 : attendre un délai aléatoire* >

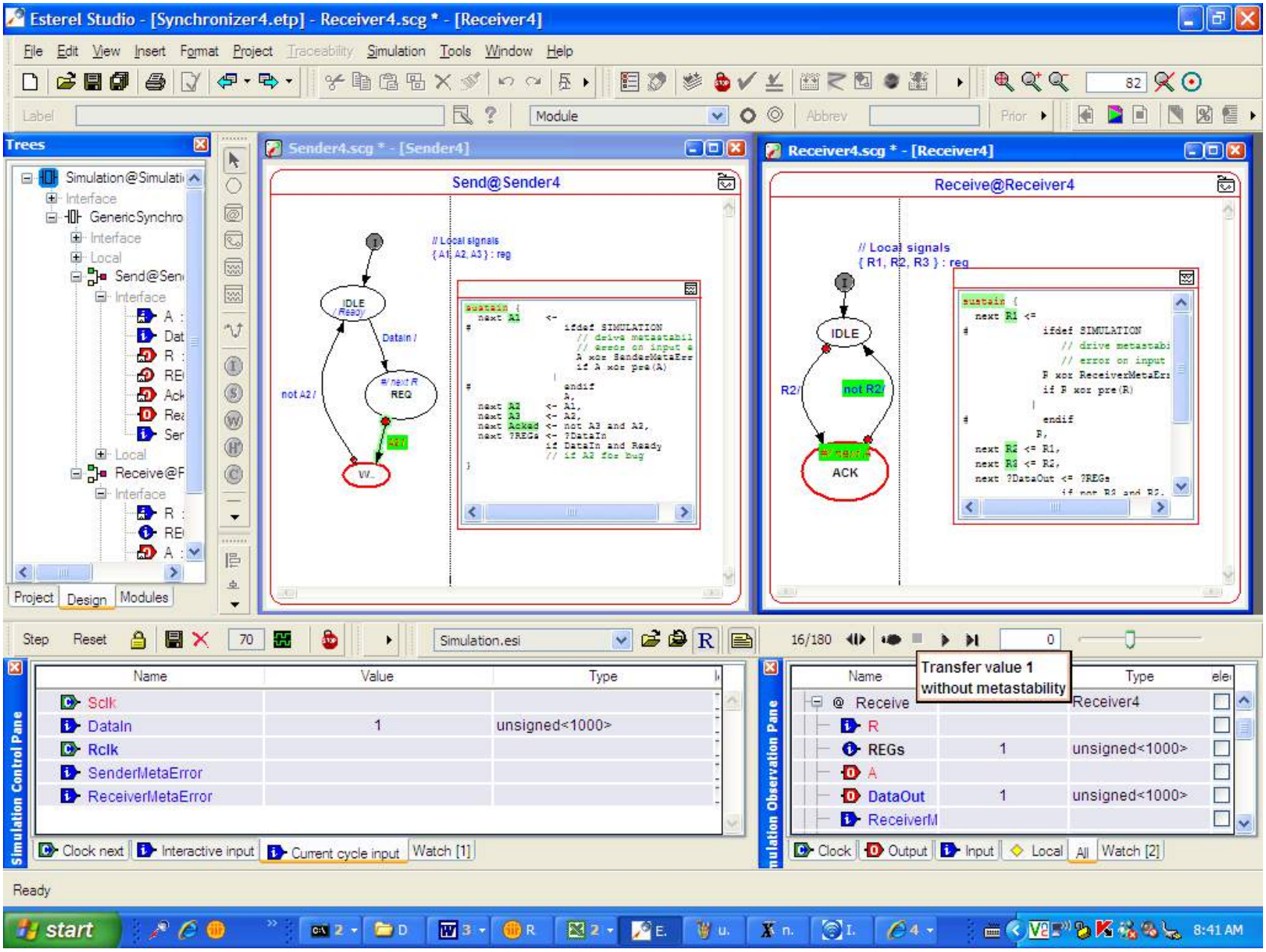
```
% phase 2 : attendre délai aléatoire
abort
   await Random () MS
when Stop do
   exit Cheated
end abort;
emit GoOn
```

```
% phase 3 : attendre Stop
abort % temps limite, fin du délai exclu
   abort
      every MS do
         emit ReflexTime(pre(?ReflexTime)+1)
      end every
   when Stop ;
   emit Display (?ReflexTime)
when LimitTime MS do
   exit EndGame
end abort ;
emit GoOff ;
exit EndMeasure
```

```
module ReflexGame :

% interface de données
constant LimitTime,
         MeasureNumber,
         PauseLength : integer ;
function Random () : integer ;

% interface de signaux

input MS, Coin, Ready, Stop ;

relation MS # Coin # Ready,
         Coin # Stop,
         Ready # Stop ;

output Display : integer,
       GoOn, GoOff,
       GameOverOn, GameOverOff,
       CheatOn, CheatOff,
       RingBell ;

% initialisations à l'allumage
emit Display(0) ;
emit GoOff ; emit CheatOff ;
emit GameOverOn ;
every Coin do

  % initialisation du jeu
  emit Display(0) ;
  emit GoOff ; emit CheatOff ;
  emit GameOverOff ;
  % le jeu proprement dit

  trap EndGame, Cheater in
    signal ReflexTime : integer,
           AverageReflexTime : integer in
      % on utiliser un module de librairie
      % pour calculer le temps de réponse moyen
      run Average [ signal ReflexTime / I,
                           AverageReflexTime / O ]

    ||
      repeat MeasureNumber times
        % phase 1: attendre Ready
        abort
          abort
            every Stop do emit RingBell end
```

```
module ReflexGame :

% interface de données
constant LimitTime,
         MeasureNumber,
         PauseLength : integer ;
function Random () : integer ;

% interface de signaux

input MS, Coin, Ready, Stop ;

relation MS # Coin # Ready,
         Coin # Stop,
         Ready # Stop ;

output Display : integer,
       GoOn, GoOff,
       GameOverOn, GameOverOff,
       CheatOn, CheatOff,
       RingBell ;

% initialisations à l'allumage
emit Display(0) ;
emit GoOff ; emit CheatOff ;
emit GameOverOn ;
every Coin do

  % initialisation du jeu
  emit Display(0) ;
  emit GoOff ; emit CheatOff ;
  emit GameOverOff ;
  % le jeu proprement dit

  trap EndGame, Cheated in
    signal ReflexTime : integer,
           AverageReflexTime : integer in

      % on utiliser un module de librairie
      % pour calculer le temps de réponse moyen
      run Average [ signal ReflexTime / I,
                           AverageReflexTime / O ]
||
  repeat MeasureNumber times
    % phase 1: attendre Ready
    abort
      abort
        every Stop do emit RingBell end
      when Ready
    when LimitTime MS do
       exit EndGame % trop lent !
    end abort ;

    % phases 2-3 : attendre délai aléatoire puis Stop
    trap EndMeasure in
      every Ready do emit RingBell end
    ||
      % phase 2 : attendre délai aléatoire
      abort
        await Random () MS
      when Stop do
        exit Cheated % Stop avant GoOn, ou en même temps !
      end abort ;
      emit GoOn ;

      % phase 3 : attendre Stop
      abort % temps limite, fin du délai exclu
        var Time :=0 : integer in
          abort
            every MS do Time := Time+1 end
          when Stop ;
          emit ReflexTime (Time)
        end var ;
        emit Display (?ReflexTime)
      when LimitTime MS do
        exit EndGame % trop lent !
      end abort ;
      emit GoOff ;
      exit EndMeasure
    end trap
  end repeat ;
      % phase 4 : afficher le temps après un délai
        await PauseLength MS ;
        emit Display (?AverageReflexTime)
      end signal
    handle Cheated do
      emit CheatOn
    end trap ; % EndGame et Cheated
    emit GameOverOn
  end every

end module
```

Animation + debugging symbolique (Esterel Studio)



Simulation au terminal

```
esterel -simul ReflexGame.strl
cc -c ReflexGame.c
cc -o ReflexGame ReflexGame.o
         $ESTEREL/lib/libcoresim.a
./ReflexGame < ReflexGame.esi
ReflexGame> % normal game;
ReflexGame> ;
--- Output: Display(0) GoOff
            GameOverOn CheatOff
ReflexGame> Coin;
--- Output: Display(0) GoOff
            GameOverOff CheatOff
ReflexGame> MS;
--- Output:
ReflexGame> Ready;
--- Output:
ReflexGame> MS;
--- Output:
ReflexGame> MS;
--- Output: GoOn
ReflexGame> MS;
--- Output:
ReflexGame> MS;
--- Output:
```

```
ReflexGame> MS;
--- Output :
ReflexGame> Stop;
--- Output: Display(3) GoOff
ReflexGame> MS;
--- Output:
ReflexGame> MS;
--- Output:
ReflexGame> Ready;
--- Output:
ReflexGame> MS;
--- Output:
ReflexGame> MS;
--- Output: GoOn
ReflexGame> MS;
--- Output:
ReflexGame> Stop;
--- Output: Display(1) GoOff
ReflexGame> MS;
--- Output:
ReflexGame> MS;
--- Output: Display(2)
```

Mauvais bouton ou trop lent

```
ReflexGame> % Ringing the bell
ReflexGame> Coin;
--- Output: Display(0) GoOff
GameOverOff CheatOff
ReflexGame> Stop;
--- Output: RingBell
ReflexGame> Ready;
--- Output:
ReflexGame> Ready;
--- Output: RingBell
ReflexGame>
```

```
ReflexGame> % Being too slow
ReflexGame> Coin;
--- Output: Display(0) GoOff
          GameOverOff CheatOff
ReflexGame> MS;
--- Output:
ReflexGame> MS;
--- Output:
ReflexGame> MS;
--- Output:
ReflexGame> MS;
--- Output: GameOverOn
```

Espèce de tricheur !

```
ReflexGame> % Trying to cheat
ReflexGame> Coin;
--- Output: Display(0) GoOff GameOverOff CheatOff
ReflexGame> Ready;
--- Output:
ReflexGame> MS;
--- Output:
ReflexGame> Stop;
--- Output: GameOverOn CheatOn
```

Tracer les variables internes

```
--- Source Variables:
V10 = 1  (source variable ReflexGame.Time)
V13 = 0  (source variable ReflexGame.Average.Sum)
V14 = 0  (source variable ReflexGame.Average.Count)
--- Signal Variables:
V4 = 0  (value of signal Display)
V5 = -*-  (value of signal ReflexTime)
V6 = -*-  (value of signal AverageReflexTime)
--- Counters:
V7 = 2 [line: 35, column: 17 of file: "ReflexGame.strl" (ReflexGame#0)]
V8 = 4 [line: 41, column: 18 of file: "ReflexGame.strl" (ReflexGame#0)]
V9 = 0 [line: 50, column: 25 of file: "ReflexGame.strl" (ReflexGame#0)]
V11 = 3 [line: 64, column: 21 of file: "ReflexGame.strl" (ReflexGame#0)]
V12 = 0 [line: 72, column: 16 of file: "ReflexGame.strl" (ReflexGame#0)]
```

Acte 3 :
Où l'on mesure ses réflexes, avant de lire l'heure

1. De bons réflexes pour une bonne santé
2. Comment faire plus joli en Esterel v7

Esterel v7 : structuration des interfaces

```
data ReflexGameData :
   constant LimitTime,
                MeasureNumber,
                PauseLength : integer ;
   function Random () : integer ;
end data

interface ReflexGameInput :
   input MS, Coin, Ready, Stop ;
   relation MS # Coin # Ready,
            Coin # Stop,
            Ready # Stop ;
end interface
```

Esterel v7 : structuration des interfaces

```
interface ReflexGameOutputOn :
   output GoOn,
          GameOverOn,
          TiltOn ;
end interface

interface ReflexGameOutputOff :
   output GoOff,
          GameOverOff,
          TiltOff ;
end interface
```

Esterel v7 : structuration des interfaces

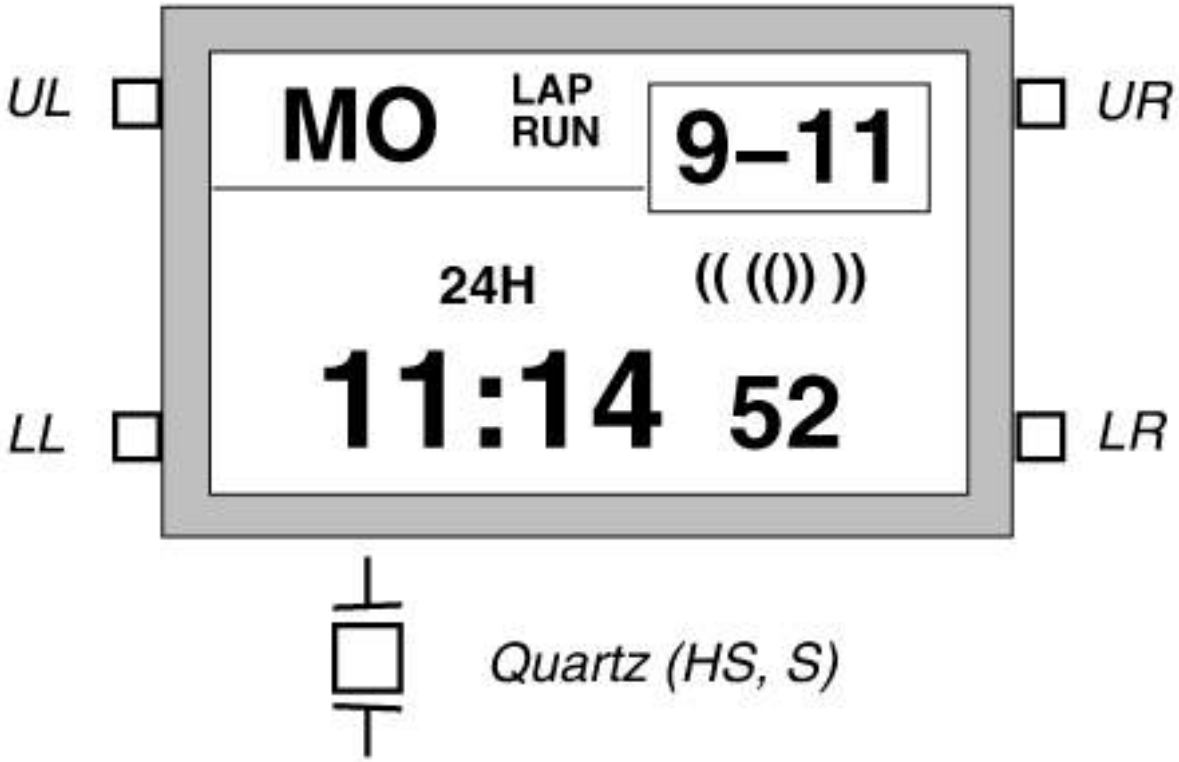
```
module ReflexGame :
   extends  ReflexGameData ;
   extends  ReflexGameInput ;
   extends  ReflexGameOutputOn ;
   extends  ReflexGameOutputOff ;
   output Display : integer ;
   …
end module

module Initialization :
  extends ReflexGameOutputOff ;
  …
end module
```

Acte 3 :
Où l'on mesure ses réflexes, avant de lire l'heure

1. De bons réflexes pour une bonne santé
2. Comment on fait mieux en Esterel v7
3. Avant l'heure, c'est pas l'heure, après l'heure, c'est plus l'heure

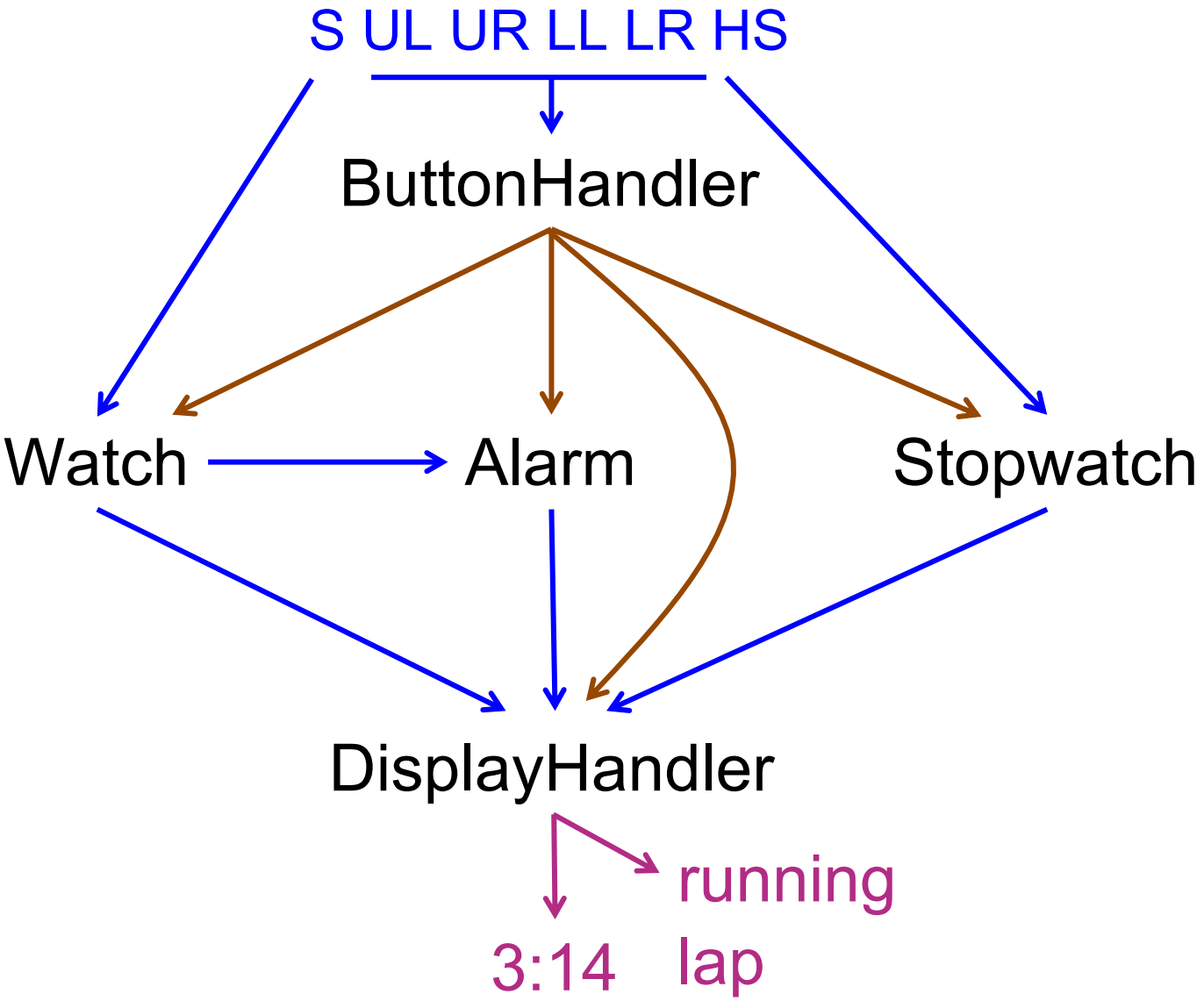
La montre digitale CASIO (1986)

Exemple bien plus riche, avec 8 modules communicants entre eux de façon synchrone



Voir le cours du 16 avril 2013

Structure des communications synchrones



Acte 4

Où la concurrence se dévoile

Acte 3 : Où la concurrence se dévoile

1. Lustre et Signal, experts en flots de données

Lustre et Signal

Nicolas Halbwachs
et Albert Benveniste

Paul Caspi

Lustre : Caspi / Halbwachs, 1983
→ SCADE, 1993. …

Signal : Benveniste, Le Guernic, Gautier 198*
→ Sildex, Polychrony (séminaire du 14/3)

Lustre : data-flow synchrone

- Inspiration : réseaux de Kahn, équations et *Block Diagrams* des automaticiens, langage LUCID
  - équations parallèles → langage déclaratif Lustre
  - *Block Diagrams* → langage graphique SCADE (*Safety-Critical Application Development Environment*)

- Principe : garder la beauté fonctionnelle des réseaux de Kahn, mais limiter leur expressivité :
  - garantir que le programme peut être exécuté en mémoire finie et temps borné
  - permette une exécution par cycles cohérents successifs
  - assurer et mesurer l'efficacité du code exécutable

Voir le séminaire de Nicolas Halbwachs, 13/01/2010

Opérateurs sur flots de valeurs (Data Flow)

$x = x_0, x_1, \ldots, x_i, \ldots$: suite infinie de valeurs

vue asynchrone (Kahn) : indice i = position dans la suite

vue synchrone : indice = i instant logique

2	2, 2, ..., 2, ...	flot constant
$x+y$	$x_0+y_0, x_1+y_1, \ldots, x_i+y_i, \ldots$	somme synchrone idem pour −, =, ≤
if x then y else z	if x_0 then y_0 else z_0 , ..., if x_i then y_i else z_i, ...,	test synchrone répété
pre(x)	nil, $x_0, x_1, \ldots, x_n, \ldots$	retard unitaire (registre)
$x \rightarrow y$	$x_0, y_1, \ldots, y_n, \ldots$	aiguillage
x fby $y = x \rightarrow$pre(y)	$x_0, y_0, \ldots, y_1, \ldots$	souvent pratique

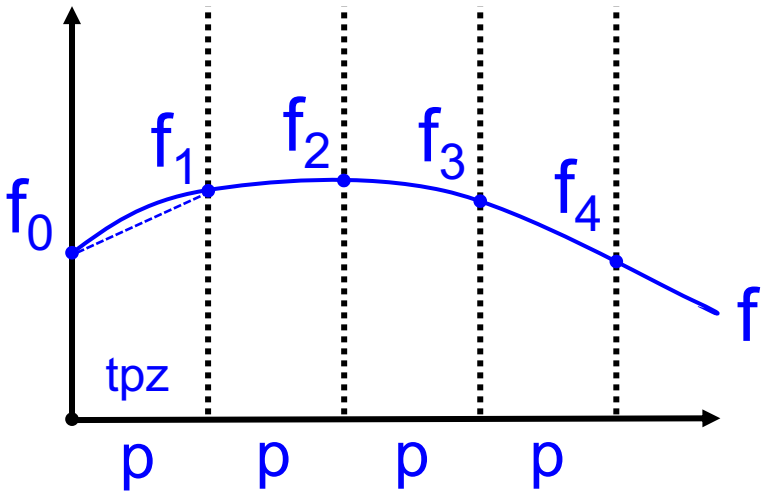
Nœuds Lustre ~ modules Esterel

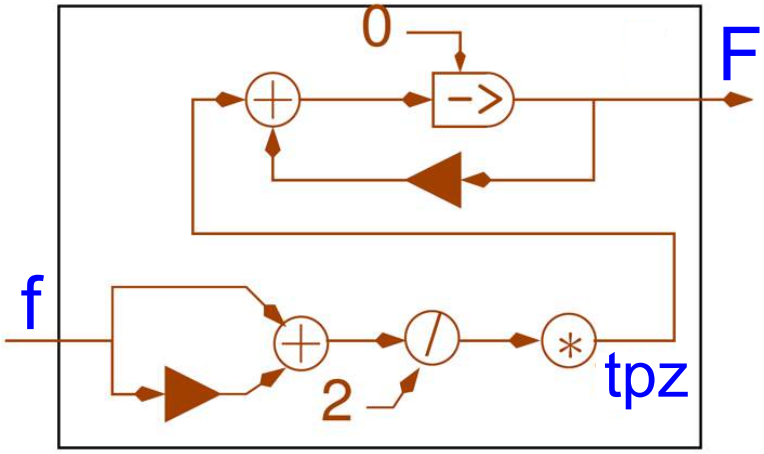
- Equations : égalité terme à terme des suites
- Noeud : entrées, sorties, équations

```
node Max (m, n : int) returns (max : int);
let
   max = if m>n then m else n;
tel;
```

```
node Naturals returns (n : int);
let
   n = 0 → pre(n)+1;
tel;

n = 0, 1, 2, 3, …
```

Intégration par la méthode des trapèzes





Block diagram

```
node Integrate (f, p : float) returns (F : float);
var tpz : float;
let
   F= 0 → pre(F) + tpz;
   tpz = p*(pre(f)+f)/2;
tel;
```

Programmation fonctionnelle

```
node SineFlow (p : float) returns (sinf : float);
let
   step = Naturals();
   sinf = sin(step*p*Pi/180);
tel ;
```

SinFlow(10.) = 0.000, 0.174, 0.342, 0.500 , 0.643, 0,767, ...
Integrate(SinFlow(10.)) = 0.000, 0.087, 0.345, .,679, 0.992,
Integrate(Integrate(SinFlow(10.))) = 0.000, 0.043, 0.215, ...

Passe à l'échelle de milliers de variables et d'équations !

Horloges

Traiter des flots évoluant à des rythmes variables

Horloge : flot Booléen, horloge de base true = T, T, T, ...

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13
c1 (on true)	T	F	F	T	T	T	F	T	F	T	T	F	F	T
y = x when c1 (on c1)	0			3	4	5		7		9	10			13
z = current (y) (on true)	0	0	0	3	4	5	5	7	7	9	10	10	10	13
c2 (on c1)	F	F	F	T	T	F	F	T	F	F	T	F	F	T
z = y when c2 (on c2)				3	4			7			10			13
current (z) (on c1)	nil			3	4	4		7		7	10			13

Horloges

Les opérateurs s'éxécutent au rythme de leurs entrées

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13
c1 (on true)	T	F	F	T	T	T	F	T	F	T	T	F	F	T
y = x when c1 (on c1)	0			3	4	5		7		9	10			13
c2 (on c1)	F	F	F	T	T	F	F	T	F	F	T	F	F	T
z = y when c2 (on c2)				3	4			7			10			13
current (z) (on c1)	nil			3	4	4		7		7	10			13
~~y+z~~ ~~y+current(z)~~	horloges différentes valeur indéfinie													

Horloges

Les opérateurs s'éxécutent au rythme de leurs entrées

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13
c1 (on B)	T	F	F	T	T	T	F	T	F	T	T	F	F	T
y = x when c1 (on c1)	0			3	4	5		7		9	10			13
c2 (on c1)	F	F	F	T	T	F	F	T	F	F	T	F	F	T
z = y when c2 (on c2)		1	2	3	4	5	6	7	8	9	10	11	12	13
4 → current (z) (on c1)	4			3	4	4	6	7	8	7	10	11	12	13
y+(4 → current(z)) (on c1)	4			6	8	9		14		16	20			26

Opération correcte ssi les flots ont la même horloge

Acte 3 : Où la concurrence se dévoile

1. Lustre, charmant langage flots de données
2. Esterel et Lustre : de la confrontation au mariage

Rafale = Esterel, Airbus = SCADE

Solution 1 : combat aérien

Solution 2 : unification des langages

→ Esterel v7, SCADE 6

(Esterel Technologies)

voir le séminaire de Bruno Pagano, le 23/04/2013

Points communs Lustre / Esterel

- La même notion du temps comme suite d'instants sans épaisseur
- La même vision du parallélisme et de la communication synchrone
- La même conception de mécanismes d'exécution souples
- Des puissances d'expression équivalentes
  - Lustre et Esterel sont inter-traductibles
  - Esterel vers circuits (voir cours 4) : traduction en Lustre+ du schéma de contrôle d'Esterel (+ effets de bord pour les données)
- Deux unifications chez Esterel Technologies
  - 2001–2009 : Esterel v7 pour les circuits
  - 2006– : SCADE 6 pour le logiciel embarqué critique

Différences Esterel / Lustre
1. Signaux vs flots

- Signal Esterel
  - le nom désigne le status et la valeur instantanée
  - asymétrie : absent par défaut et présent si émis
  - émission réception explicites et soumises au contrôle
  - la combinaison d'émissions multiple est possible

- Flot Lustre
  - le nom désigne la suite infinie des valeurs instantanées
  - symétrie vrai / faux en booléen
  - émission et réception implicites et soumises aux horloges
  - la combinaison d'émissions multiple est interdite

pre(S) et pre(?S) en Esterel

- pre(S) / pre(?S) = statut / valeur de S à l'instant précédent
- Importé de Lustre, hélas un peu tard (2000 !)
  (pas avant car déjà définissable à partir du noyau)

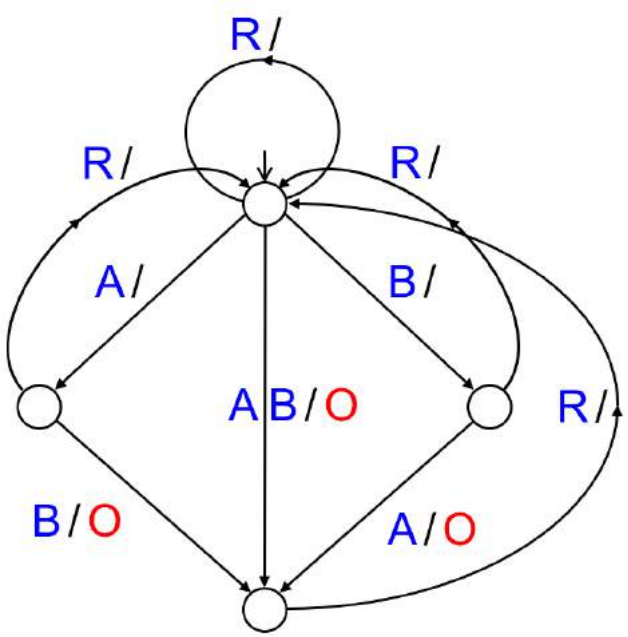
pre(S) ↔ preS défini par

```
loop
   if S then
      pause ; emit preS
   else
      pause
   end if
end loop
```

trop lourd
pour les utilisateurs !

Mais essentiel pour éviter les cycles de causalité,
(voir prochain cours)

Différences Esterel / Lustre
2. Traitement du contrôle




```
module ABRO (A, B, R) returns O ;
local Aok, Bok, Odone ;
let
   Aok = (A or pre(Aok)) and not R
   Bok = (B or pre(Bok)) and not R
   O = Aok and Bok
         and not Odone and not R
   Odone = (O or pre(Odone))
                and not R
tel ;
```

Plus de *Write Things Once* facile….

Incorporation de Lustre dans Esterel v7

ABRO

```
loop
   signal Aok, Bok, Odone in
      sustain {
         Aok <= (A or pre(Aok)),
         Bok <= (B or pre(Bok)),
         O <= Aok and Nok and not Odone,
         Odone <= (O or pre(Odone))
   end signal
each R
```

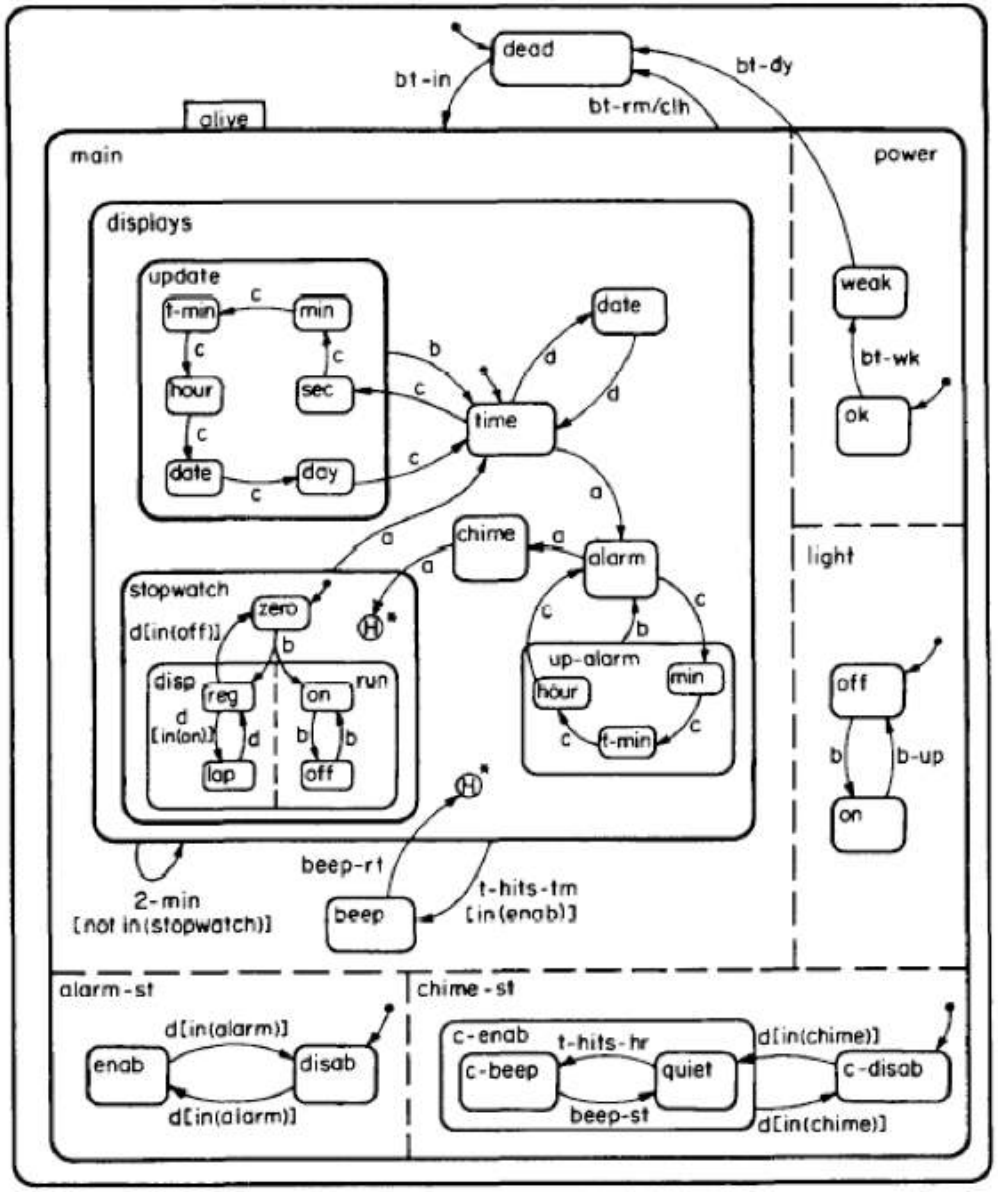
Combinaison
(ici implicitement par or)

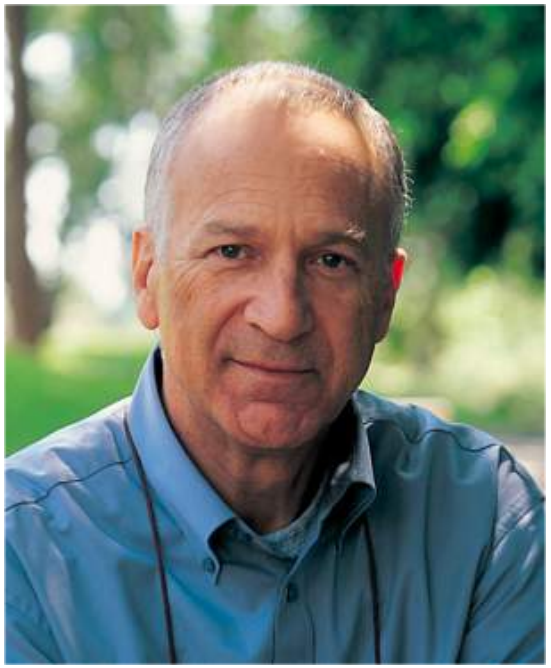
```
output X;
sustain {
    X <= I or J,
    X <= not K and L
}
```

Acte 3 : Où la concurrence se dévoile

1. Lustre, charmant langage flots de données
2. Esterel et Lustre : de la confrontation au mariage
3. Pour ceux qui préfèrent les dessins

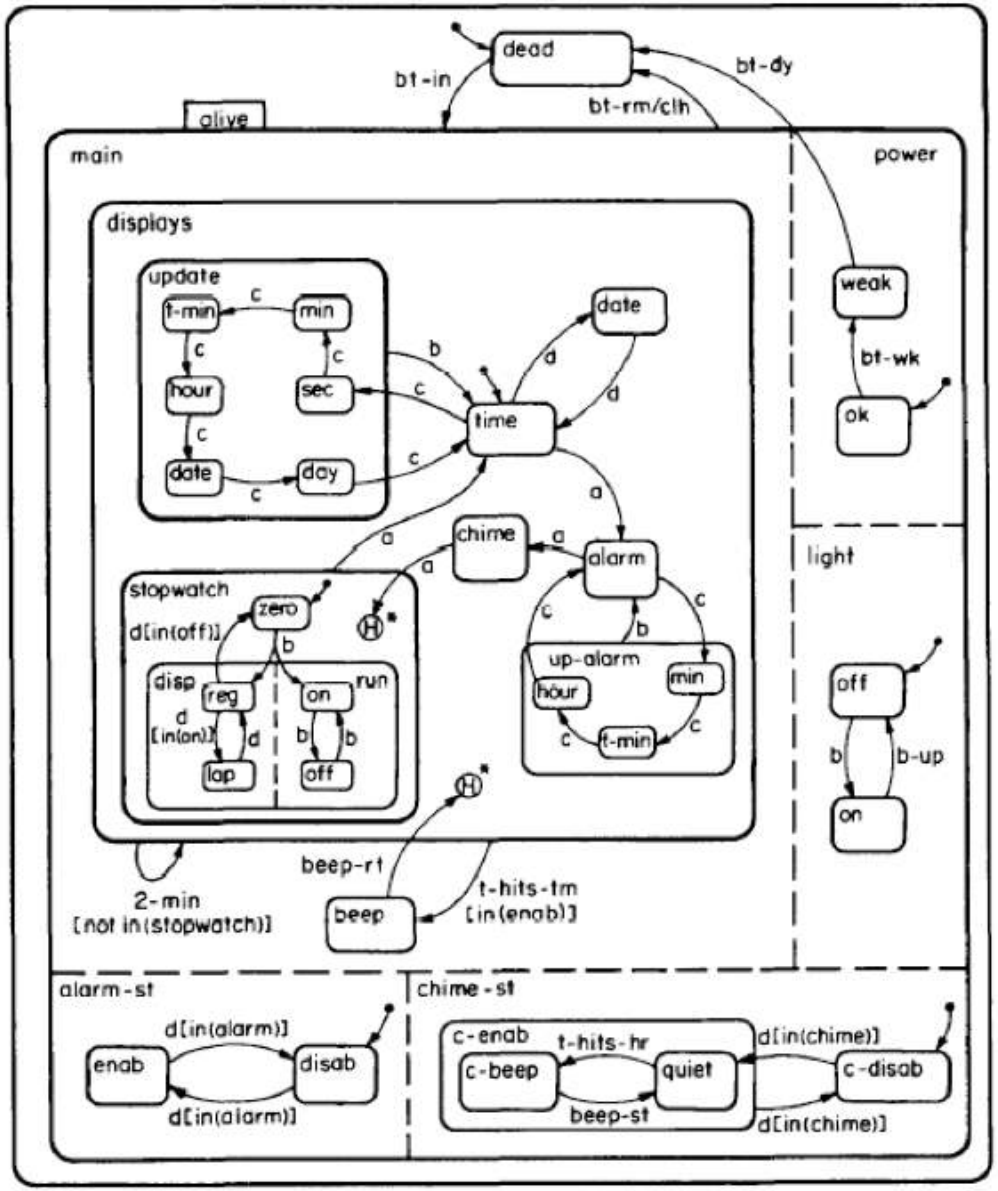
Statecharts, David Harel (1987)





Un formalisme graphique fondateur

Statecharts, David Harel (1987)



Objectif :

expressivité maximale

idée initiale

spécifications avioniques

évolution

code exécutable

version industrielle

(Ilogix → IBM)

Problèmes des Statecharts

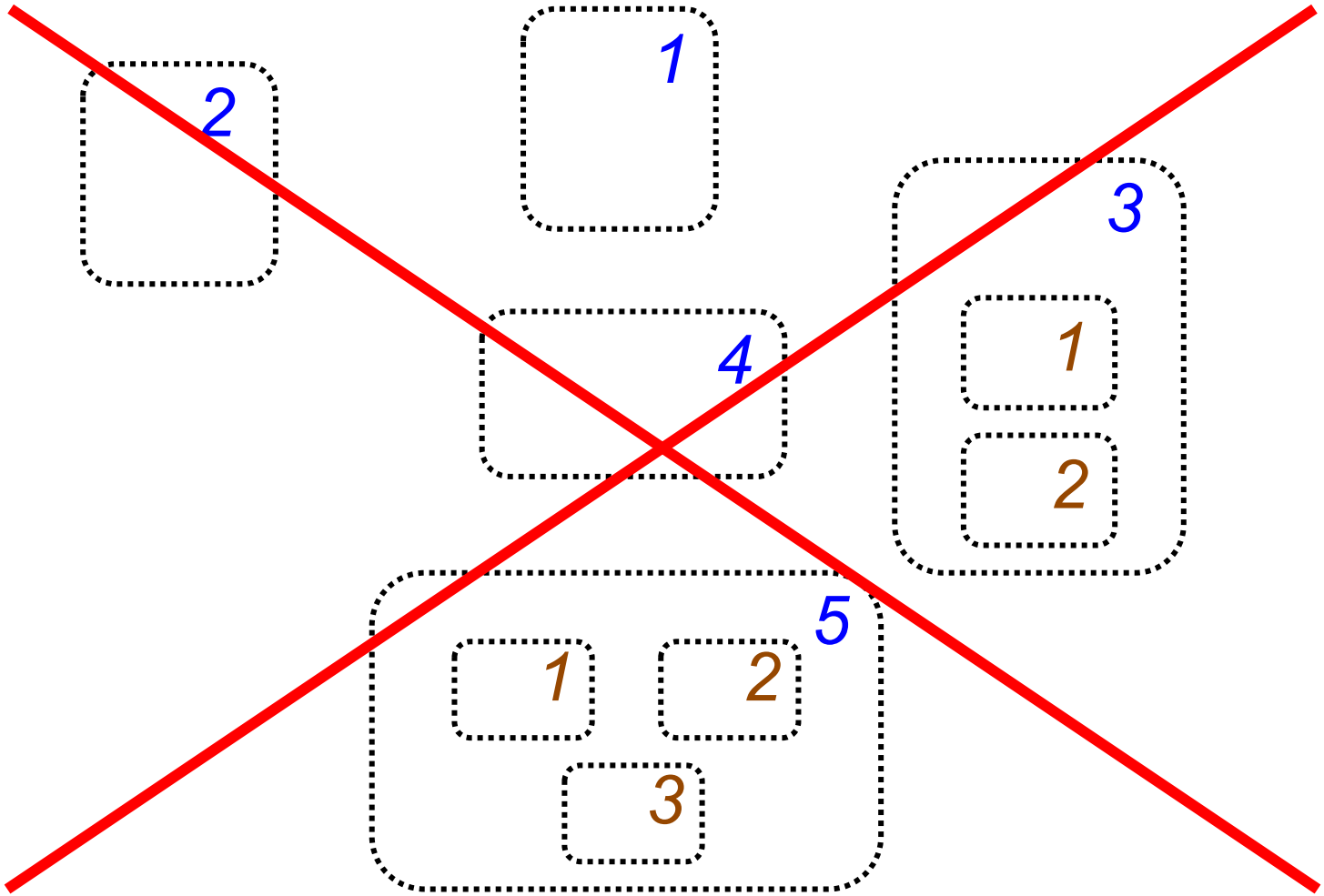
- Sémantique opérationnelle « officielle » , fondée sur les entrelacements de *microsteps*, avec la décision que tout programme doit avoir un sens
- La grande liberté pose des problèmes d'interprétations des constructions (ex. transitions traversant librement les niveaux)
- Mais beaucoup d'autres sémantiques proposées, chacune cohérente, toute différentes
- Laquelle a raison ?

Nous : aucune, sauf une sémantique synchrone qui demanderait des restrictions syntaxiques

Stateflow (The Mathworks), UML, etc.

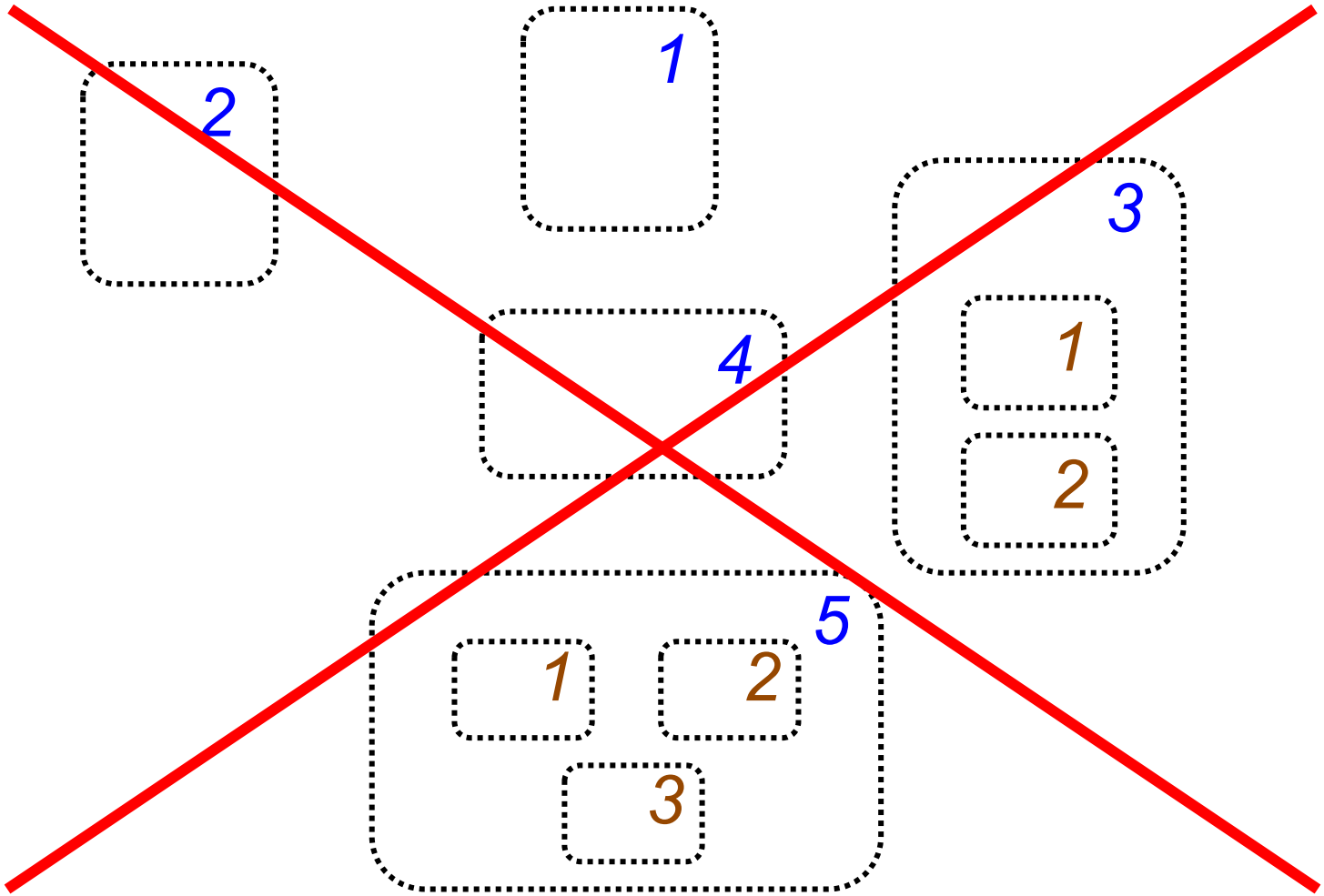
- Outil industriel inspiré de Statecharts et couplé à Simulink
- Très utilisé pour la simulation en automobile, avionique, etc.
- Avec génération de code C embarquable
- Mais refus du vrai parallélisme, traitement par blocs et règles complexes
- Soyons francs : sémantique mal fichue (dommage, la solution était connue !)

Ordre gauche-droite haut-bas (lecture)

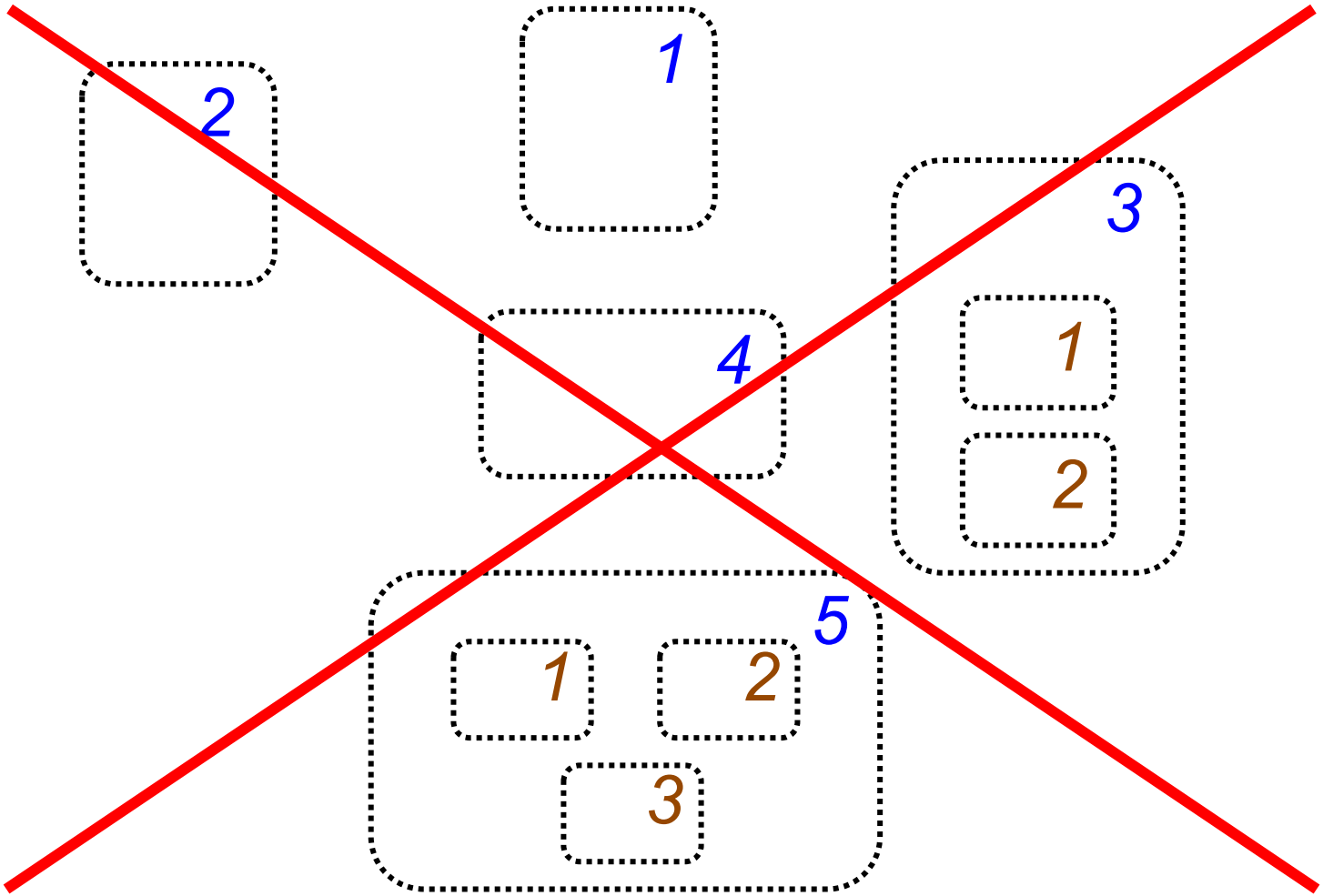


Bouger *1* ou *2* de 1 pixel ⇅ change le comportement !

Ordre d'exécution = ordre de création



Réordonnancement manuel



Plus règles de renumérotation générale,
changement local de priorités, *cut & paste* d'états., etc.

ARGOS (F. Maraninchi, 1991)

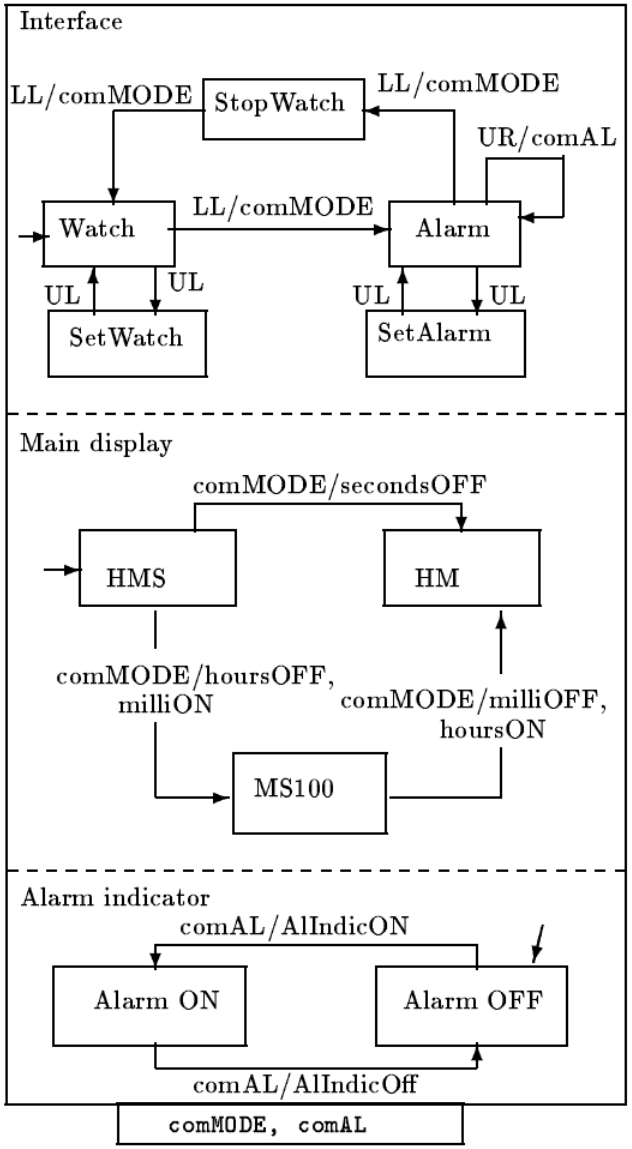


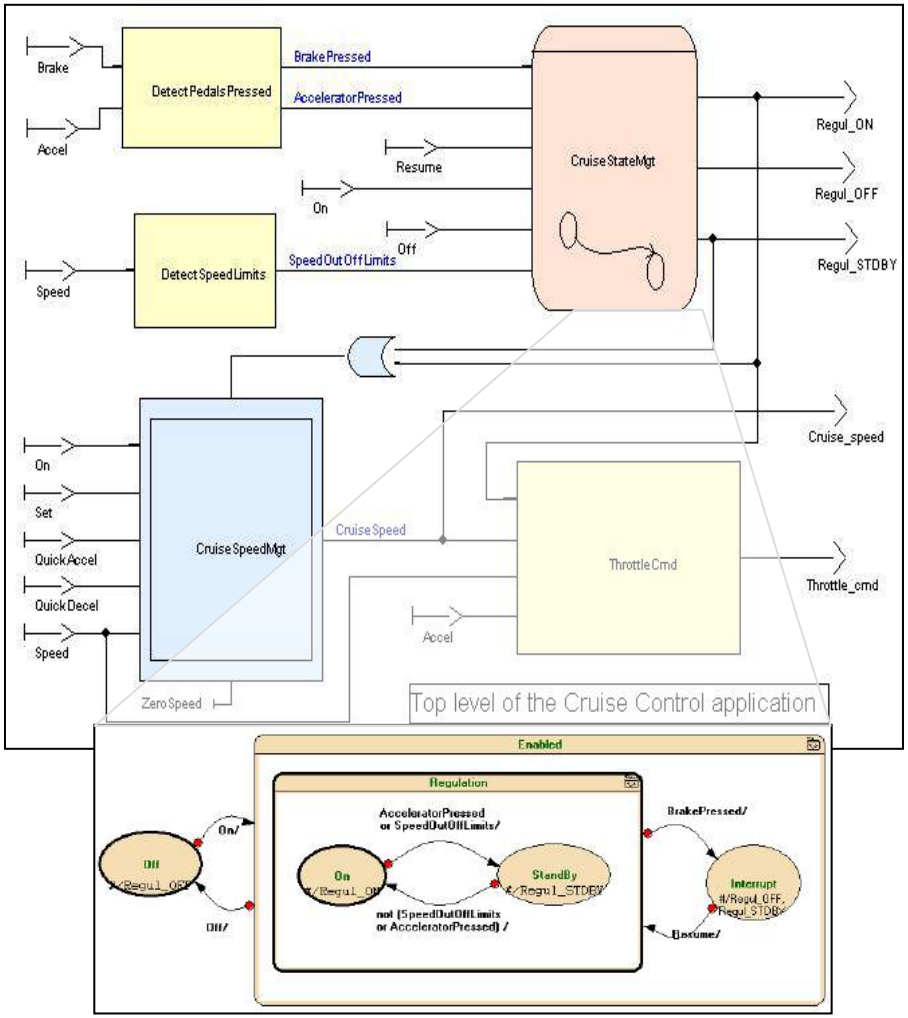
Figure 2: interface and ressources of the watch, an example of parallel components

Aussi inspiré de Statecharts

Automates parallèles et hiérarchiques bien structurés

rigueur et modularité

SCADE 5 = Lustre en dessins + Automates



Verilog 1993 → nucléaire, Airbus A380, etc.

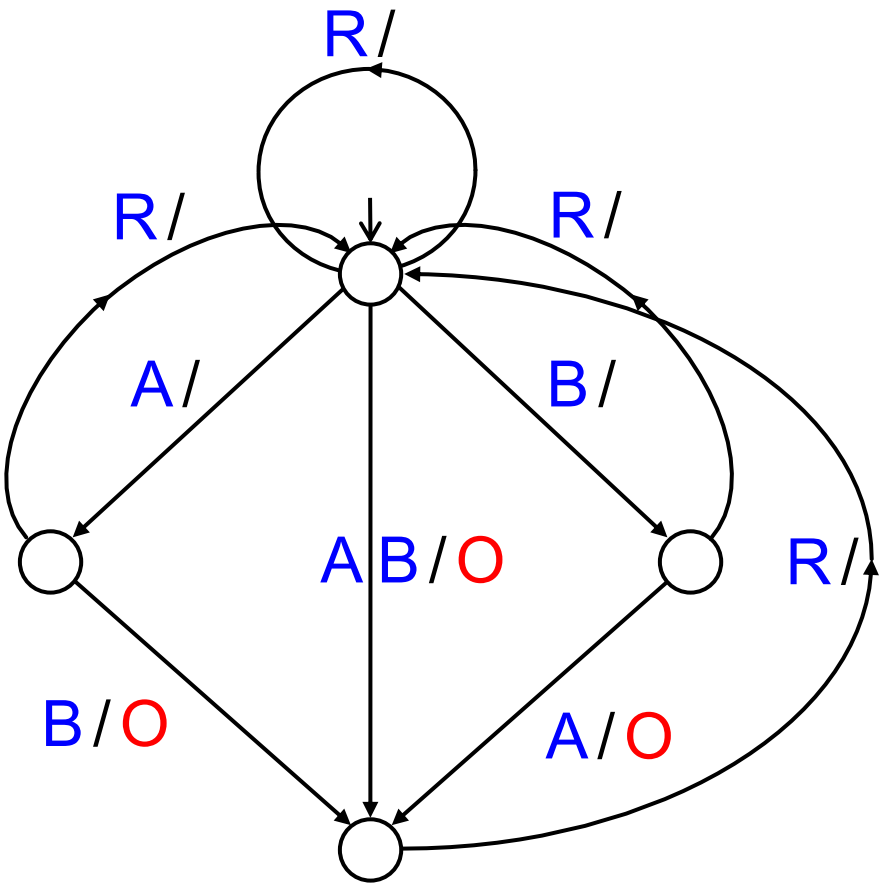
SyncCharts (1995), AGEL, Esterel Studio

Charles André
I3S Sophia-Antipolice*

Garder la puissance
et la rigueur d'Esterel
dans des dessins
à la Statecharts

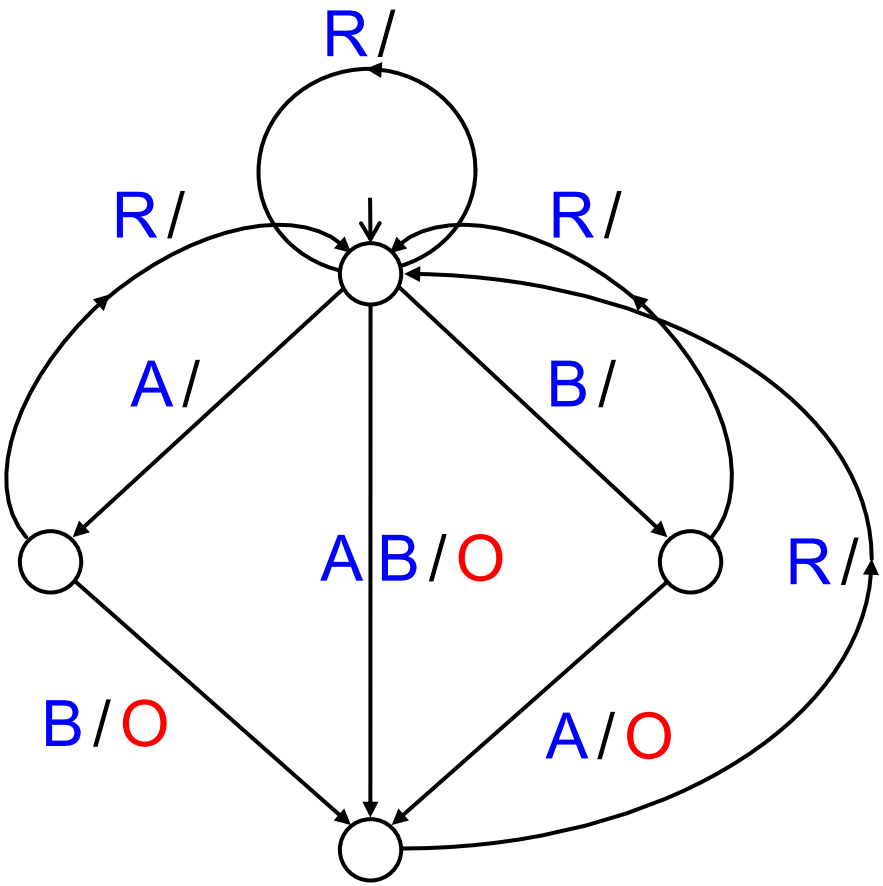
** Merveilleux nom écrit dans l'adresse d'une lettre envoyée à mon labo par le Ministère de l'intérieur !*

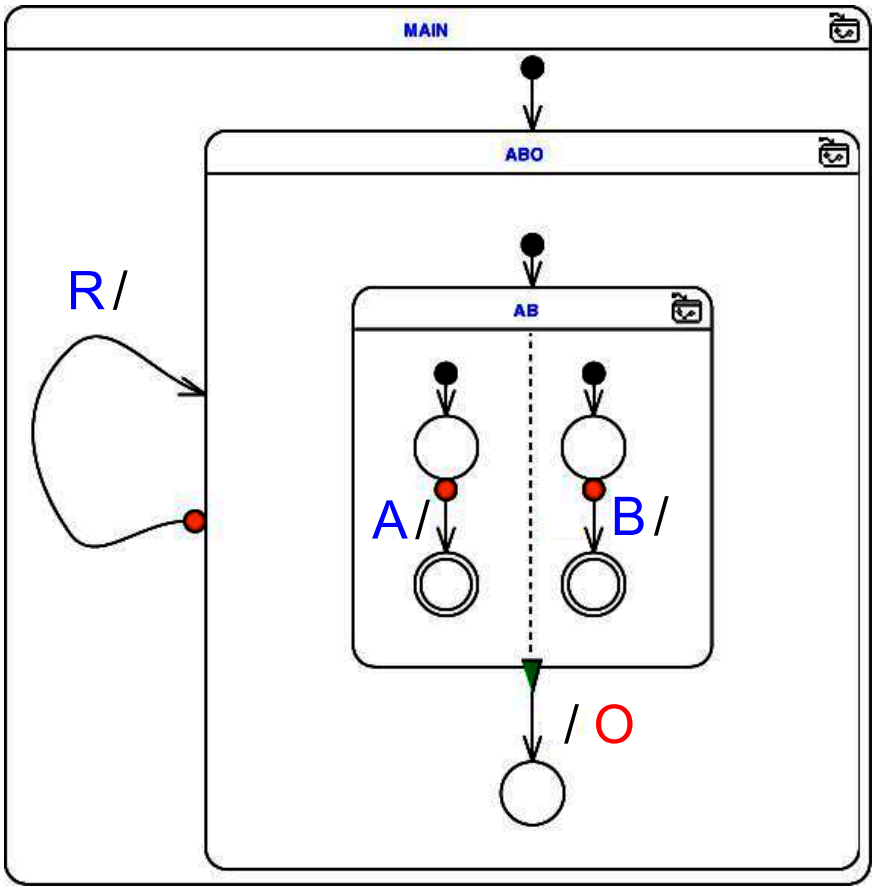
ABRO en Esterel




```
loop
   abort
        {await A || await B};
        emit O;
        halt
   when R
end loop
```

ABRO en SyncCharts

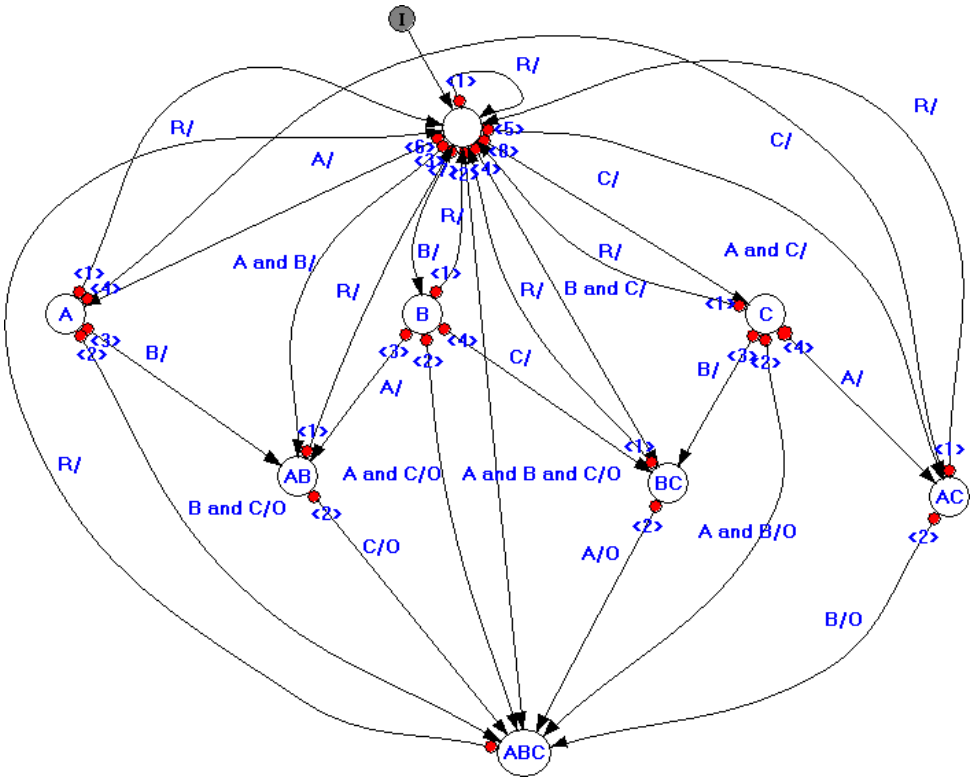




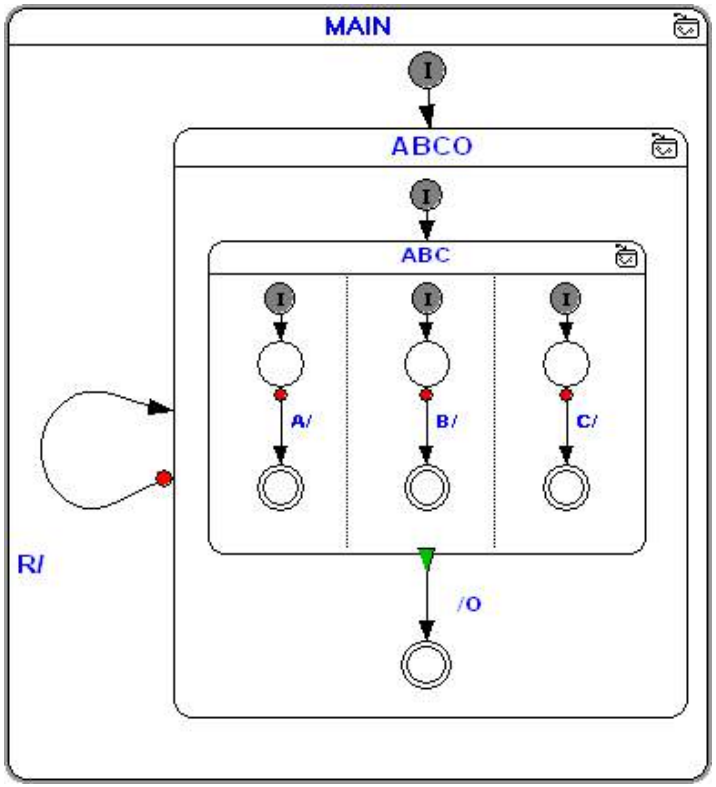
Hiérarchie et parallélisme

⊶ = abort, → = weak abort, ⤚ = terminaison

ABCRO : Draw Things Once (DTO)

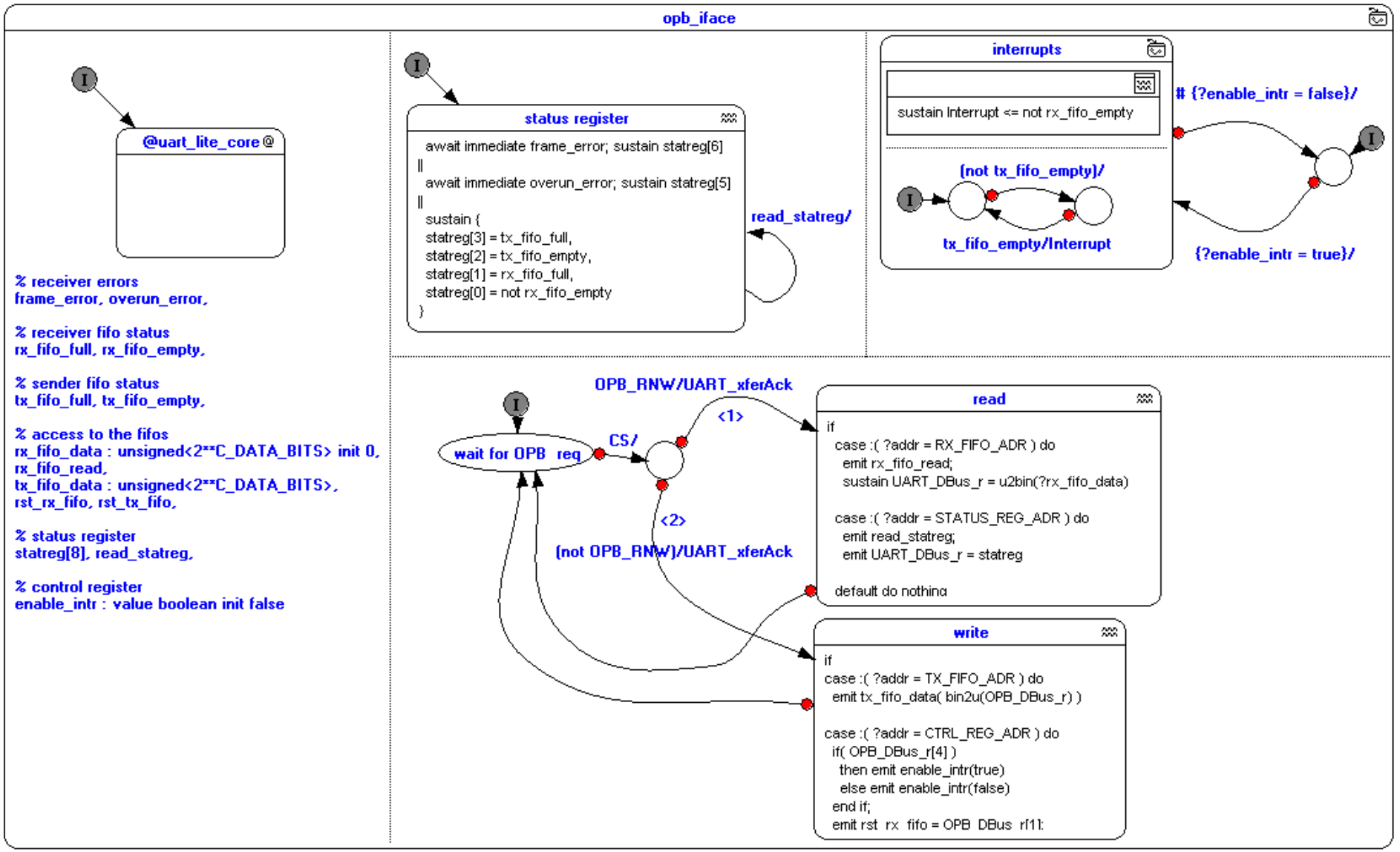


automate plat



SyncChart, linéaire

SyncCharts → AGEL → Esterel Studio

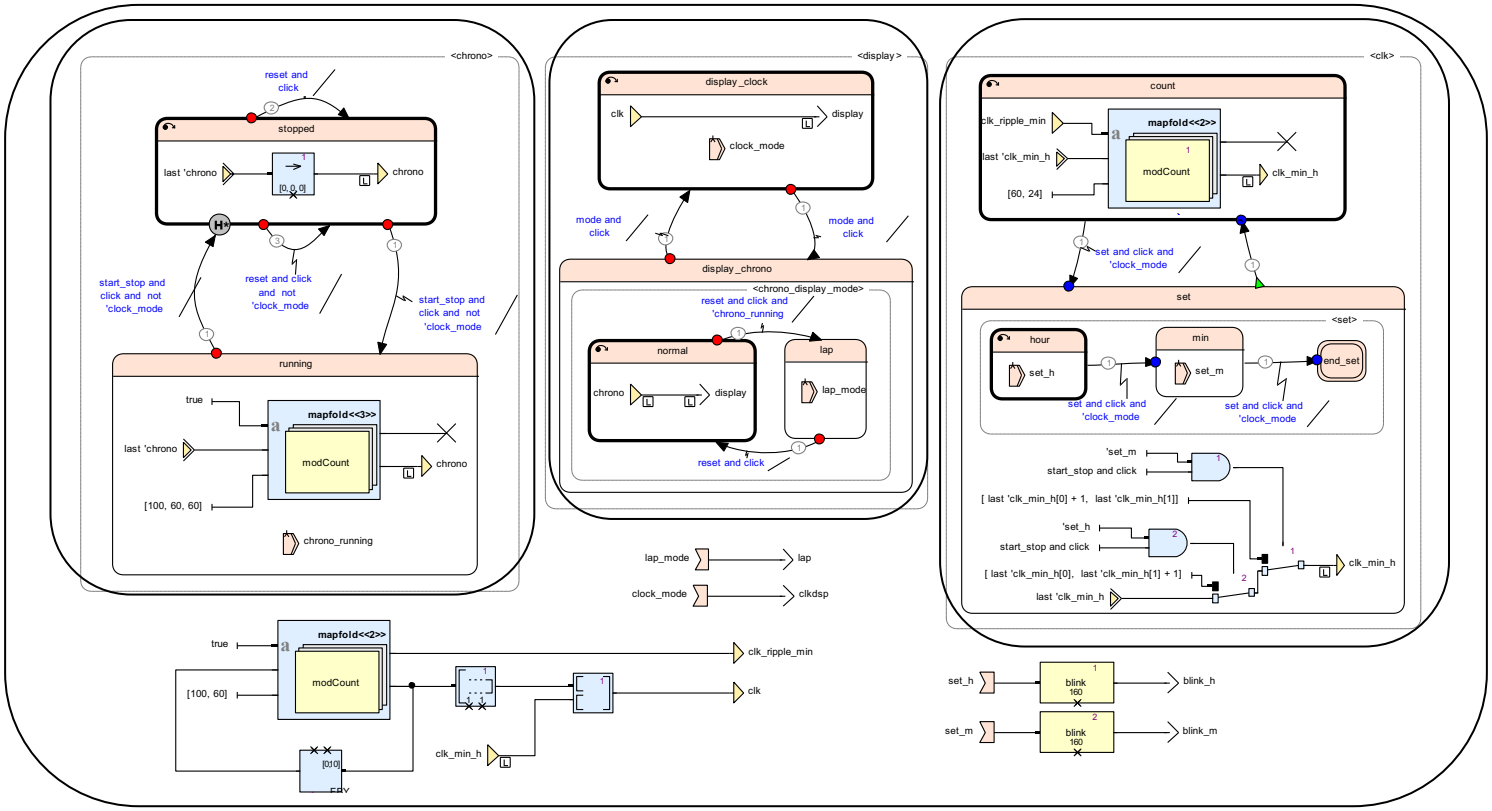


Protocole de lien rapide Xilinx, par Satnam Singh

Acte 3 : Où la concurrence se dévoile

1. Lustre, charmant langage flots de données
2. Esterel et Lustre : de la confrontation au mariage
3. Pour ceux qui préfèrent les dessins
4. La modernité

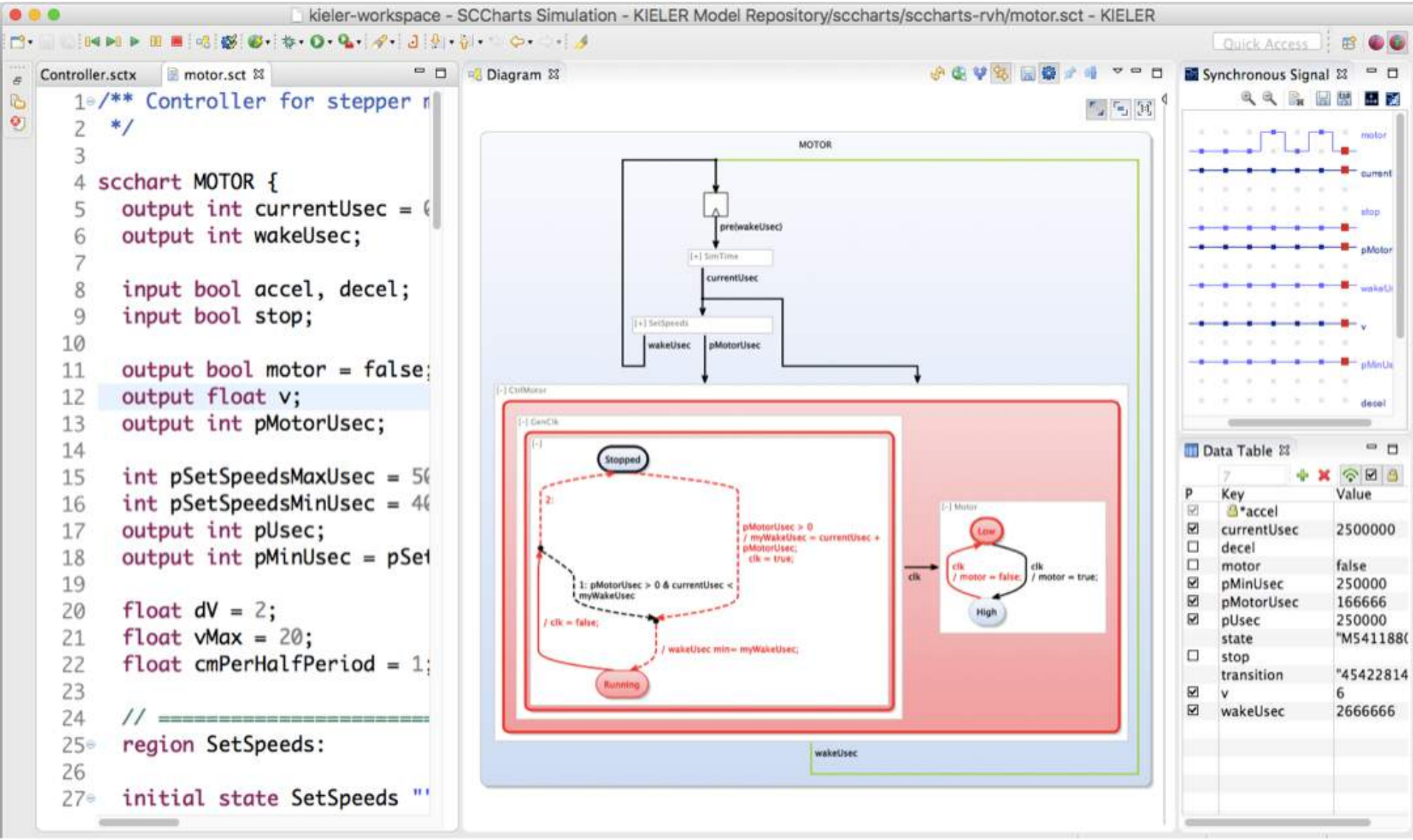
SCADE 6, Esterel Technologies
Esterel / SyncCharts + Lustre / SCADE 5



Cruise Control

Licence universitaire disponible

Le système KIELER pour ScCharts



Voir le séminaire du 14/03/2017 par R. von Hanxleden

Epilogue

Arbre généalogique du Synchrone (?)