La nécessaire mais délicate coopération de modèles

Gérard Berry

Collège de France
Chaire Informatique et sciences numériques

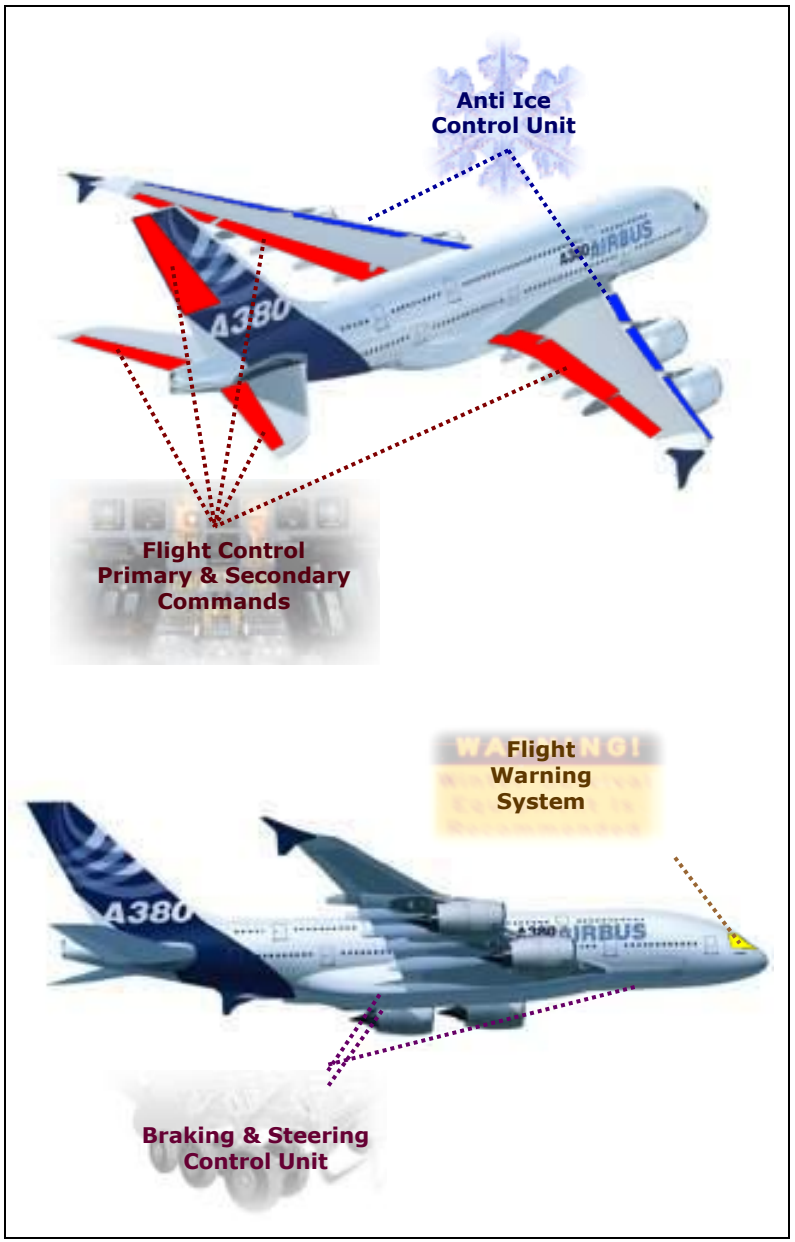
Cours 8 du 27 janvier 2010

La dernière séance

Forces et limites

- Séquentiel
  - naturel pour nous, bien compris, bien maîtrisé
  - mais pas de parallélisme
- Parallélisme synchrone / vibratoire
  - déterminisme, excellente maîtrise
  - surtout adapté aux circuits et à l'embarqué critique
- Parallélisme asynchrone / diffus
  - naturel pour un très grand nombre d'usages
  - mais le non-déterminisme le rend dur à maîtriser

Beaucoup d'applications doivent conjuguer les trois : transports, System on Chips (SoCs), etc.

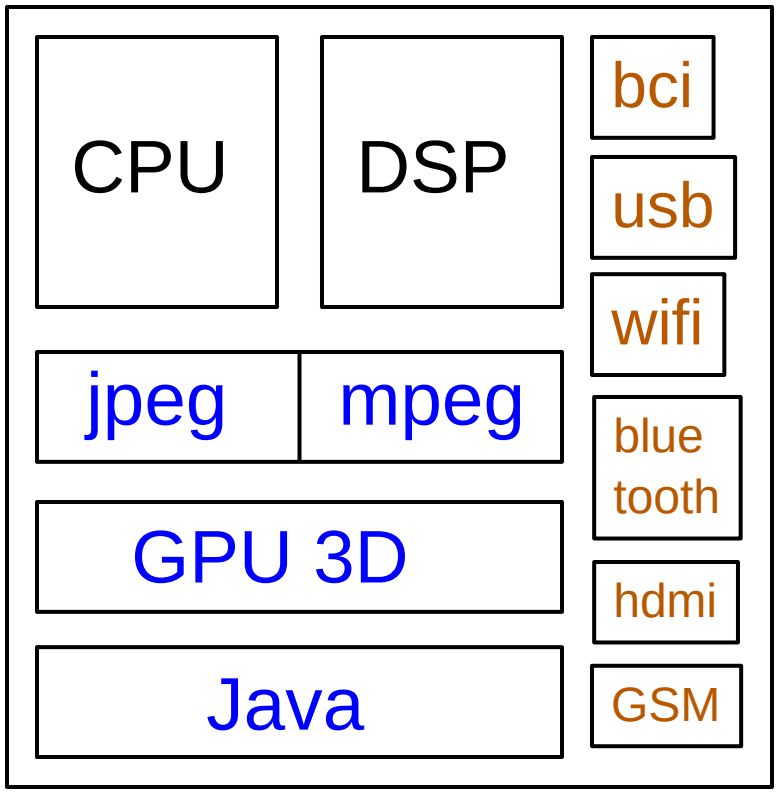


- Chaque système critique est synchrone
- Mais la surveillance globale de ces systèmes ne peut pas l'être
- Le contrôle aérien non plus
- Dans l'avenir, multiprocesseurs

⇒ couplage synchrone / asynchrone

Cf. cours 2008, leçon 4,
systèmes embarqués

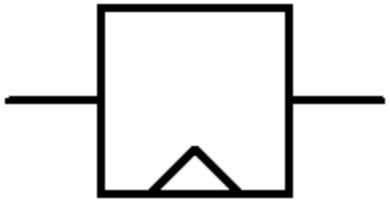
Système sur puce

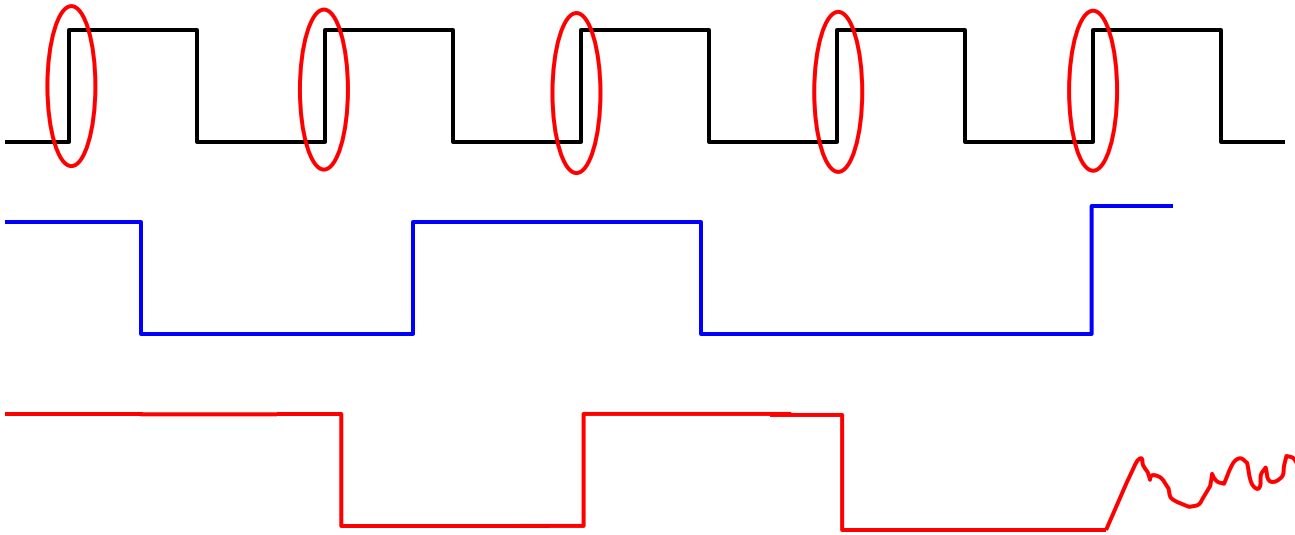


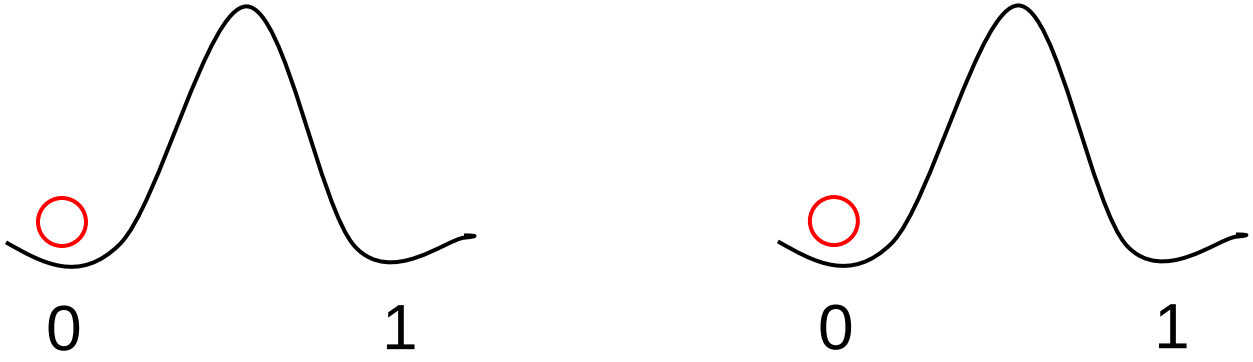
- Plusieurs horloges asynchrones
  - chaque composant a son rythme propre
- Des dizaines de sous-domaines
  - couper l'horloge ⇒ économiser l'énergie

Comment transférer de l'information entre zones d'horloges asynchrones ?

La métastabilité physique

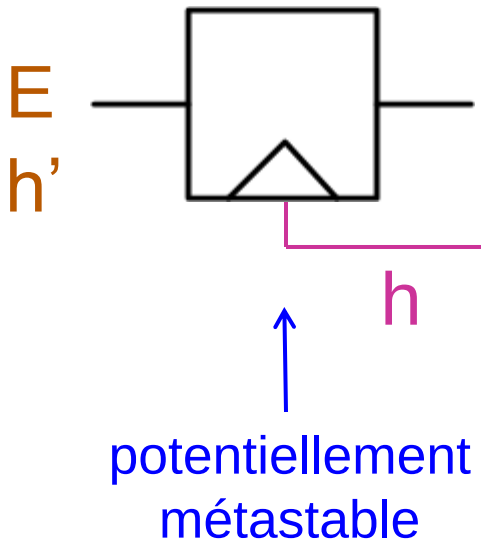




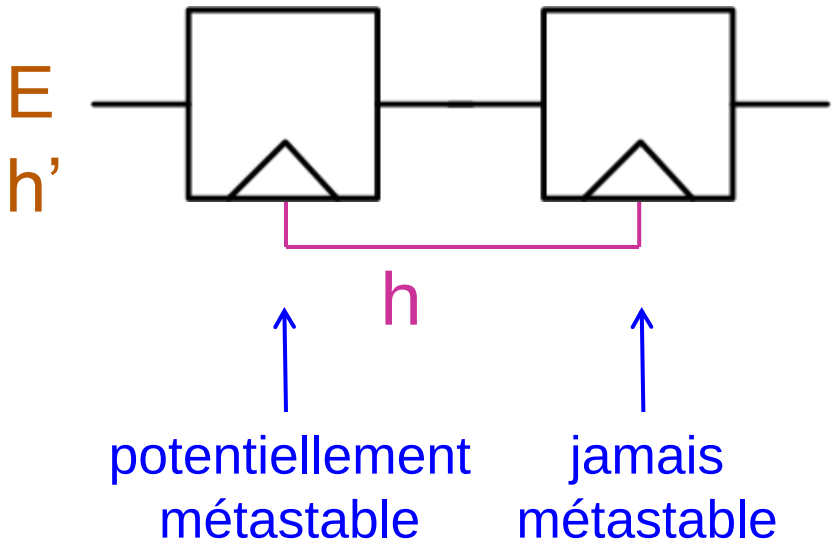


- Le vent fera tomber la balle d'un côté ou de l'autre de façon non déterminisme
- Dans les circuits, le bruit thermique fera le vent
- Théoriquement, en un temps arbitraire
- Pratiquement, en moins d'un cycle

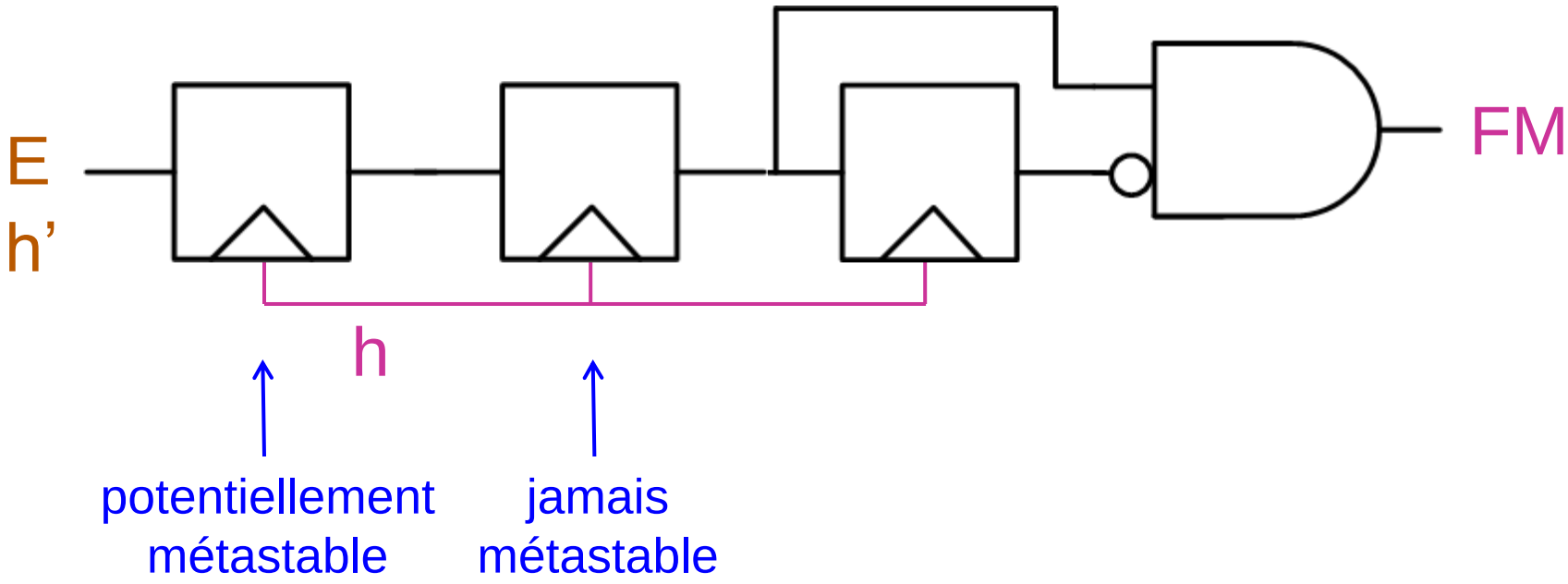
Détecteur sûr de fronts montants



Détecteur sûr de fronts montants

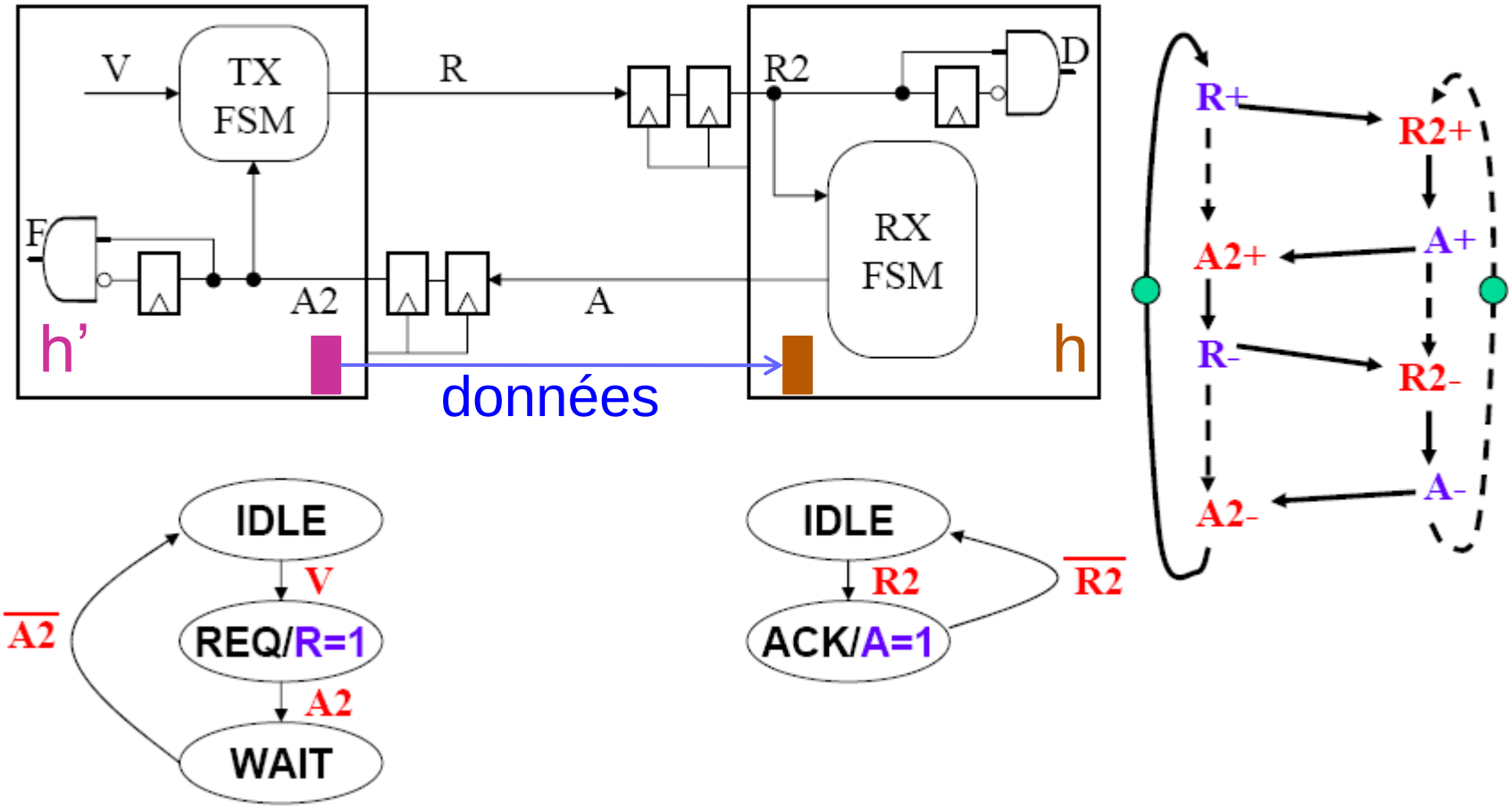


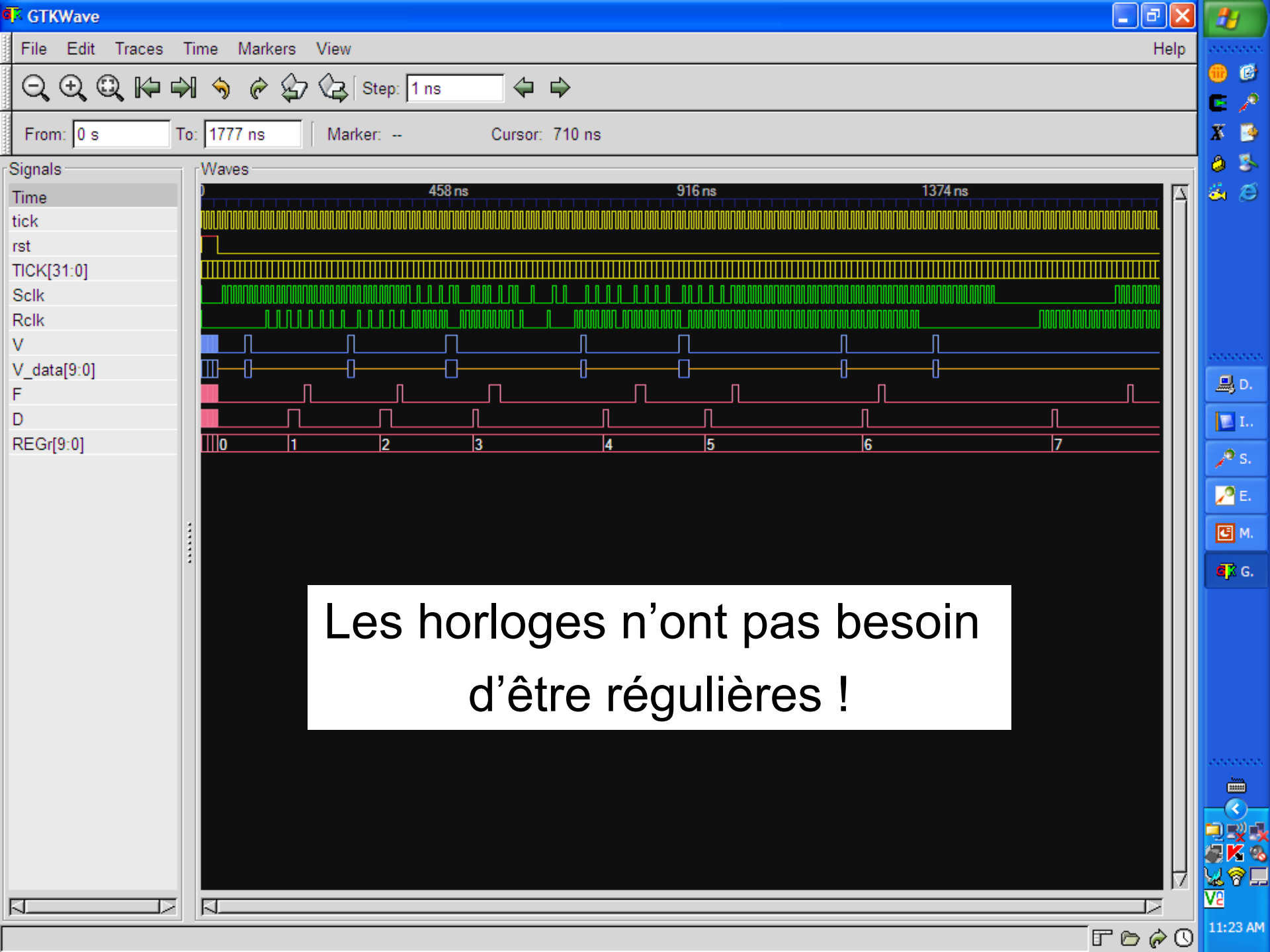
Détecteur sûr de fronts montants



Si E sur h' passe de 0 à 1 et y reste,
alors FM passe de 0 à 1 au bout de 2 ou 3 h

Le synchroniseur 4 phases (Ran Ginosar [1])

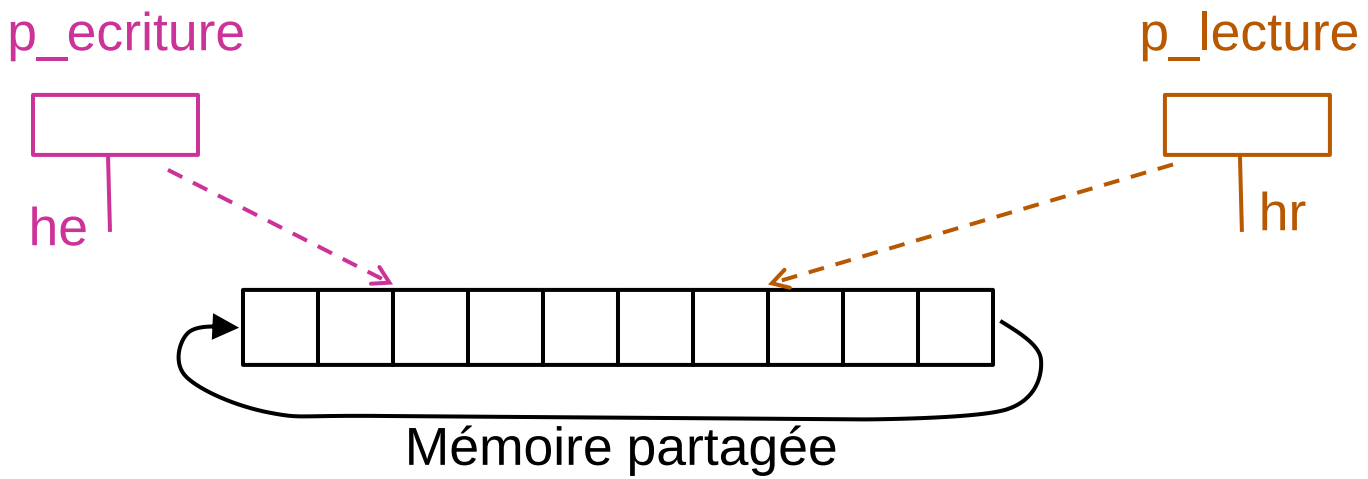



GTKWave
File Edit Traces Time Markers View Help
Step: 1 ns
From: 0 s
To: 1777 ns
Marker: --
Cursor: 710 ns
Signals
Time
tick
rst
TICK[31:0]
Sclk
Rclk
V
V_data[9:0]
F
D
REGr[9:0]
Waves
458 ns
916 ns
1374 ns
0
1
2
3
4
5
6
7
Les horloges n’ont pas besoin d’être régulières !
11:23 AM

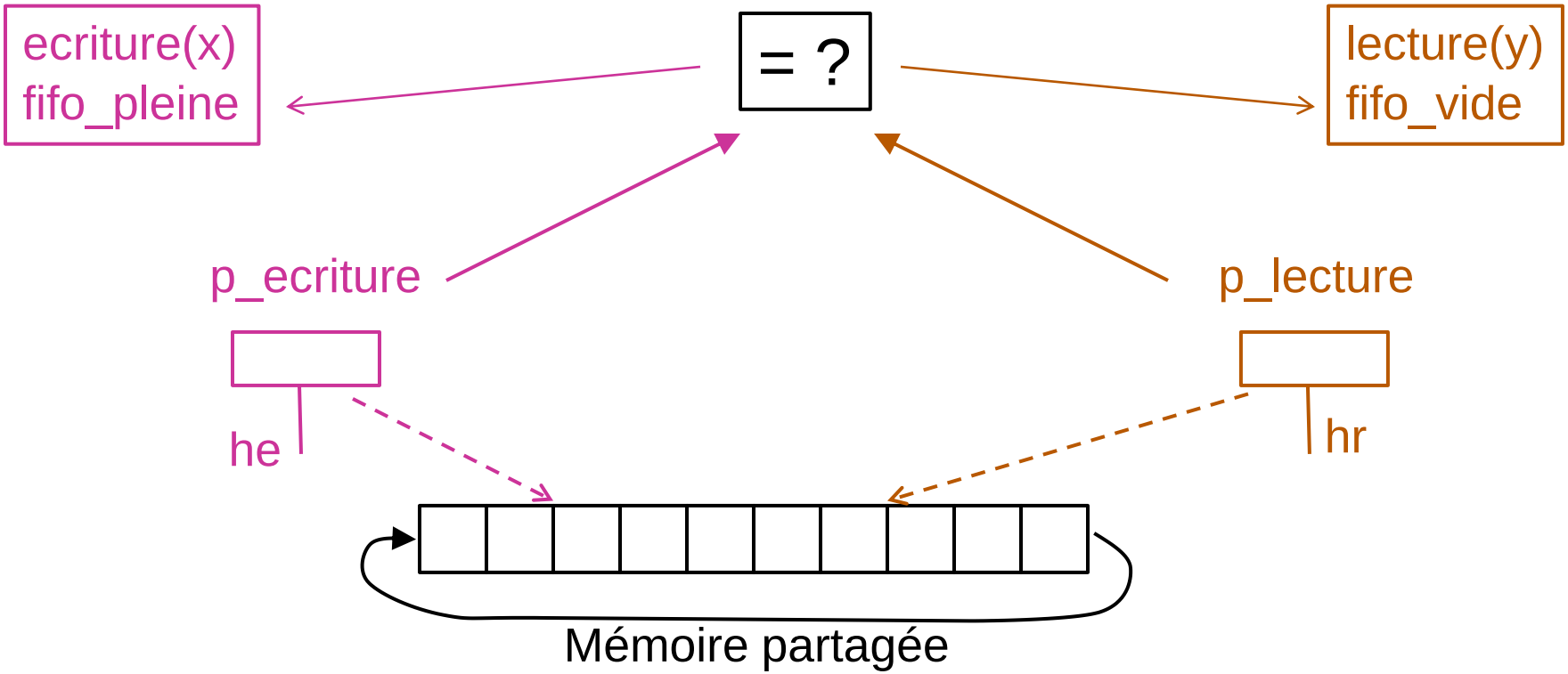
Fifos multi-horloges [2]

ecriture(x)
fifo_pleine

lecture(y)
fifo_vide

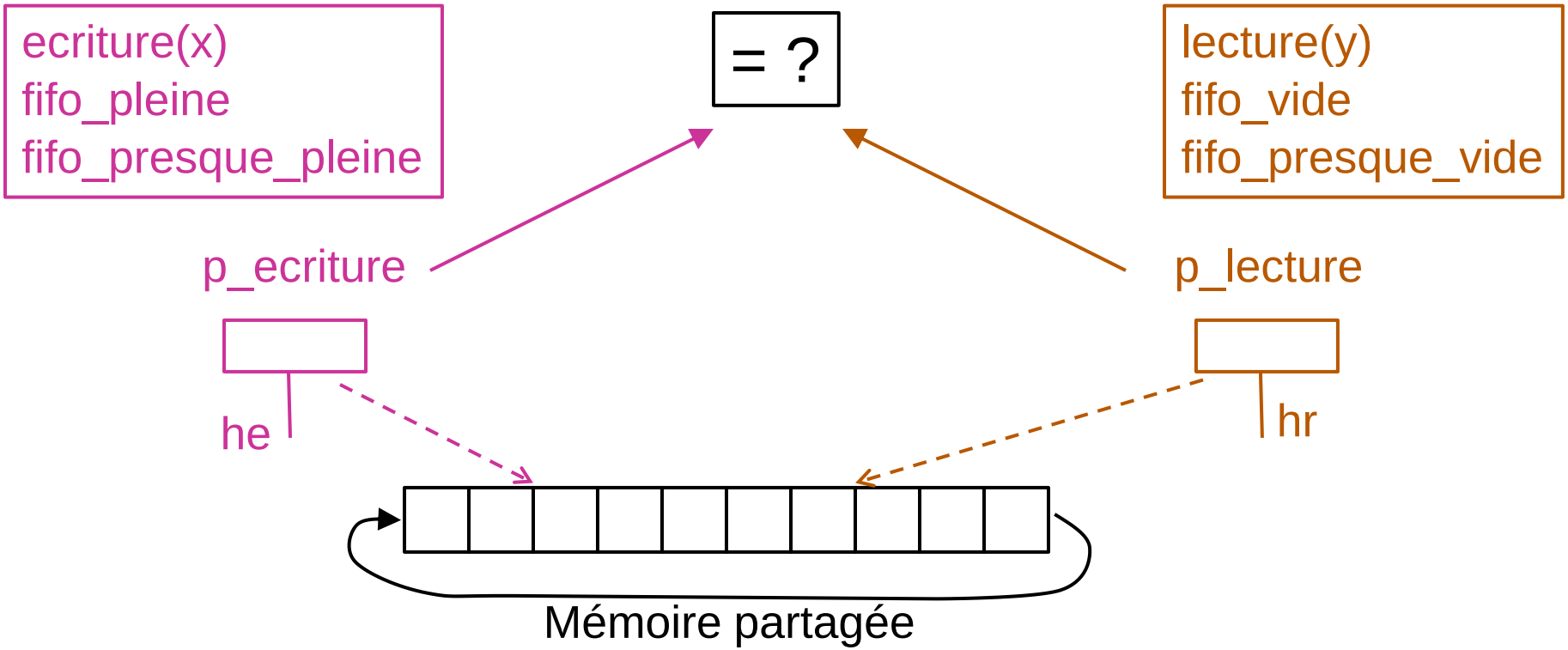


Fifos multi-horloges [2]



Comparer les pointeurs : jamais juste, toujours sûr !
Les transférer en code Gray (1 seul bit change à chaque fois)

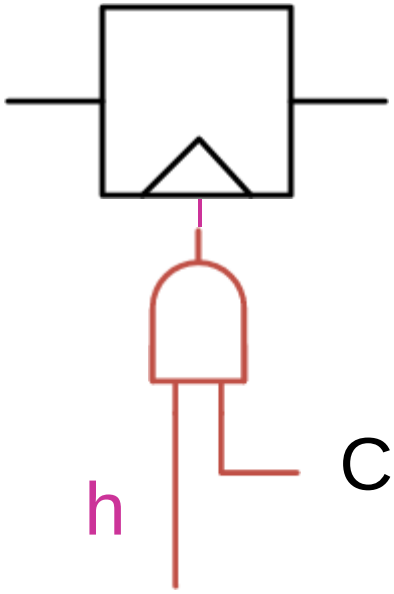
Fifos multi-horloges [2]



Pré-détecter les événements dangereux !
Tout ça est très délicat...

Esterel multi-horloge [3]

1. Sous-échantillonner les registres en coupant l'horloge
   "clock gating" en anglais
   fondamental pour l'économie d'énergie :
   faire commuter un registre consomme beaucoup



Pas faisable en

Esterel original (v5)

Sémantique de suspend

$$\frac{s \notin E \qquad p \xrightarrow[E]{E' \quad k} p'}{s \supset p \xrightarrow[E]{E' \quad k} s \supset p'}$$

$$\frac{s \in E}{s \supset p \xrightarrow[E]{\emptyset \quad 1} s \supset p}$$

weak suspend (Klaus Schneider)

$$\frac{s \notin E \quad p \xrightarrow[E]{E' \;\; k} p'}{s \supseteq p \xrightarrow[E]{E' \;\; k} s \supseteq p'}$$

$$\frac{s \in E \quad p \xrightarrow[E]{E' \;\; k} p'}{s \supseteq p \xrightarrow[E]{E' \;\; \max(k,1)} s \supseteq p}$$

état inchangé

Weak suspend $\Rightarrow$ clock gating + multi-horloge

Circuits asynchrones

- asynchrone, insensible aux délais
- mais CAO complexe et non-industrialisée
- reste marginal en pratique

Circuits élastiques [4]

- asynchrone, insensible aux délais
- mais logique fonctionnelle standard, CAO classique
- plus logique spéciale de clock-gating
  et latchs transparents au lieu de registres

La programmation réactive (F. Boussinot [4])

- Threads synchrones, pouvant être créés dynamiquement, avec ordonnancement déterministe
- Mémoire de contrôle dans les fonctions
- Communiquer par diffusion de signaux
- Causalité simplifiée pour éviter les cycles
  réaction immédiate à la présence d'un signal,
  réaction retardée à son absence

Parallélisme coopératif non-préemptif
≠ Parallélisme compétitif préemptif (Java)

Voir la vidéo pour la démo du crible de Darwin !

Références

[1] *Fourteen ways to fool your synchronizer*

Ran Ginosar

Proc. 9th Int. Symposium on Asynchronous Circuits and Systems, 2003

[2] *Simulation and Synthesis Techniques for Asynchronous FIFO design*

Cliff Cummings

Synopsys User Group Conference, San Jose, 2002
www.sunburst-design.com

[4] *Clocking Schemes in Esterel*

L. Arditi, G. Berry, M. Kishinevsky et M. Perreaut

Proc. Designing Correct Circuits DCC'06, Vienna, Austria.

Références

[4] *Synthesis of Synchronous Elastic Architectures*

Sava Krstić, Jordi Cortadella, Michael Kishinevsky

Proc. FMCAD'2006, San José, USA

[5] *Objets réactifs en Java*

Frédéric Boussinot

Presses Polytechniques et Universitaires Romandes (PPUR), 2000

voir aussi http://www-sop.inria.fr/mimosa/rp/

Question : non-déterminisme de C

```
int x = 1;
int A() { x = x+1; return x; }
int B() { x = 2*x; return x; }
int F(x,y) { return x+y; }
main () { printf("%d", F(A(),B()); }
```

ordre d'évaluation des arguments indéterminé

- A() exécuté avant B() => 2*(1+1) = 4
- B() exécuté avant A() => (2*1)+1 = 3

C n'est ni compositionnel ni déterministe

Q : le style est ici non-déterministe, mais la programmation déterministe est possible en C avec quelques règles

R : oui, mais savoir si un programme est déterministe est indécidable
le risque vaut-il vraiment la chandelle?

La notion de « sous-langage » (SystemC, Stateflow, etc.) est rarement bonne

Un bon langage est un langage dans lequel tous les programmes sont clairs

Question : non-déterminisme de C

```
int x = 1;
int A() { x = x+1; return x; }
int B() { x = 2*x; return x; }
int F(x,y) { return x+y; }
main () { printf("%d", F(A(),B()); }
```

ordre d'évaluation des arguments indéterminé

- A() exécuté avant B() => 2*(1+1) = 4
- B() exécuté avant A() => (2*1)+1 = 3

C n'est ni compositionnel ni déterministe

Q : quelles sont les méthodes pour éviter ça?

R : 1. fixer l'ordre d'évaluation

2. travailler avec des langages purement fonctionnels (Lustre)

Attention : la partie non-fonctionnelle de CAML a le même problème (car traduite en C?)

Question : contraintes sémantiques

- Q : pour Esterel, que veut dire définir la sémantique par des contraintes? N'y a-t-il pas contradiction avec l'affirmation qu'elle doit être exacte ?

- R : la sémantique comportementale d'Esterel vue en cours n'est effectivement qu'un premier pas, qui définit des contraintes que toute sémantique précise doit satisfaire
- Aller plus loin demande l'étude complète de la causalité dans Esterel. Ici, pas de réponse unique, mais plusieurs choix possibles satisfaisant les contraintes. Pour moi, le plus naturel est celui de la sémantique constructive, fondée sur celle des circuits cycliques ; en pratique, la sémantique acyclique est moins générale mais plus efficace

- L'essentiel est que toutes ces sémantiques restent parfaitement compatibles pour les programmes qu'elles acceptent, et que l'acceptation est décidée à la compilation, pas à l'exécution

Question : verrous et bases de données

- Q : les verrous posés par les clients sont-ils protégés contre les blocages provoqués par les crash de clients ?
- R : bien sûr, mais ce n'est pas simple : il faut d'abord savoir qu'un client est crashé - t*ime-out* ? Il faut surtout assurer que la transaction n'est pas à moitié faite (caches, journalisation, etc)

- Q : les techniques des bases de données ne suffisent-elles pas à résoudre tous les problèmes de verrous ?
- R : Les solutions BD efficaces sont lourdes et chères (banques). Les verrous dont je parlais sont entre processus légers (threads). Ils doivent être très rapides, pas chers, et traiter des structures de données complexes : listes, arbres, graphes, etc.

Question : !p en π-calcul

- Q : !p → p | !p dangereux pour la calculabilité

  !p → p | !p → p | p | !p →...

- R1 : idem pour Y en λ-calcul :

  (YM)→ M(YM) → M(M(YM)) → ...

- R2 : mais règle plus claire en Lπ-calcul

  !a(x).p | a<b> → p[b/x] | !a(x).p

  ! ne s'use que si l'on s'en sert !

  (mais secondaire car π-calcul ≈ π-calcul)

Question: réseau CAN et déterminisme

- Q : le réseau CAN peut être utilisé en applications critiques comme le freinage automobile car il est déterministe pour les communications de priorité maximale, contrairement à Ethernet
- R : exact, mais c'est loin de suffire dans le cas où le réseau est partagé par *plusieurs* applications critiques.
- Les réseaux comme TTP ou Ethernet déterministe pré-allouent des tranches de temps aux application critiques, en gardant une communication classique pour les autres.
- TTP fournit aussi la synchronisation d'horloges, la redondance physique des transmissions, etc.
- Alternative asynchrone : l'approche contrôle distribué de Caspi & Benveniste, cf cours 2008 « systèmes embarqués »

Les frontières du calcul - 1

- Q : dans quel mesure peut-on assurer la fiabilité d'une application diffuse ?
- R : c'est un des axes de recherche importants du projet HOP de l'INRIA.
- Pour les pannes, comment en tenir compte dépend de l'application, pas de théorie générale pour l'instant
- Pour la sécurité : garantir automatiquement l'anti-intrusion par l'insertion de mécanismes cryptographiques lors de la compilation et la certification des téléchargements, voir par exemple « proof-carrying code »

Sujet de recherche majeur dans le monde !

Les frontières du calcul - 2

- Q : La programmation diffuse peut-elle avoir des applications dans les applications industrielles critiques, p.ex. composants mobiles reliés à la physique des objets ?

- R : certainement à l'avenir, par exemple pour relier les voitures entre elles et à la ville. Mais les applications resteront probablement de criticité modérée comparées à la conduite directe du véhicule. L'échelle de criticité aura de plus en plus de barreaux, à bien relier entre eux !

Le sujet est ouvert, mais les acteurs industriels restent très prudents (à juste titre)

Les frontières du calcul - 3

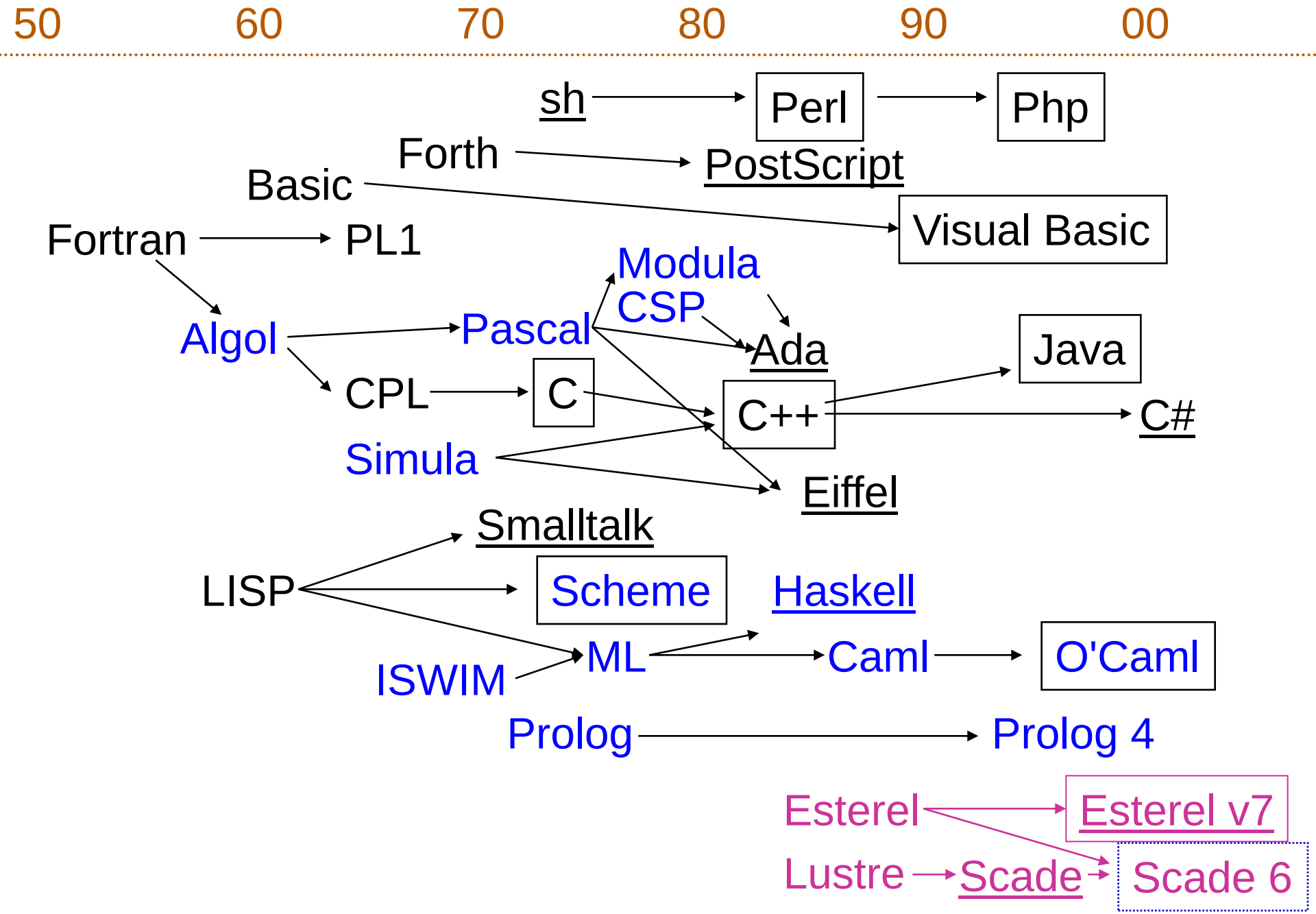
- Q : En programmation diffuse, peut-on se passer d'un coordinateur, par exemple pour les nombres premiers ?
- R : pas vraiment dans ce cas ; mais, dans le cas général, le coordinateur peut n'être pas connu d'avance (cf élection d'un leader, séminaire Raynal), et la coordination peut aussi être répartie (p.ex. par dichotomie)
- Pour les grands calculs répartis, comme Seti, les choses seraient-elles plus simples si les ordinateurs ne tombaient jamais en panne ?
- R : sans doute, mais cette hypothèse est devenue complètement irréaliste et doit être abandonnée

La Tour de Babel des langages
Comment s'y repérer ? Ouvrage ?

- Beaucoup de styles, des guerres de religions...

- *Programming Languages: Concepts and Construct*
  Ravi Sethi
  Addison-Wesley

- http://oreilly.com/news/graphics/prog_lang_poster.pdf
- http://en.wikipedia.org , programming languages


50
60
70
80
90
00
sh
Perl
Php
Forth
PostScript
Basic
Visual Basic
Fortran
PL1
Modula
CSP
Algol
Pascal
Ada
Java
CPL
C
C++
C#
Simula
Eiffel
Smalltalk
LISP
Scheme
Haskell
ML
Caml
O'Caml
ISWIM
Prolog
Prolog 4
Esterel
Esterel v7
Lustre
Scade
Scade 6

Conclusion globale

- Un grand voyage dans les modèles
  - abstraits pour la calculabilité
  - abstraits + concrets pour la séquentialité : λ-calculs
  - nombreux et moins clairs pour le parallélisme asynchrones
  - clairs mais limités pour le parallélisme synchrone
  - en naissance pour le parallélisme diffus
- Objectif : mettre en perspective ces modèles techniquement reliés mais socialement disjoints

Merci à tous, séminaristes, présents et internautes !