Les langages synchrones: des dessins qui commandent!

Marc Pouzet

UPMC/DI ENS/Inria Paris

Marc.Pouzet@ens.fr

June 15, 2017

Programmer et contrôler un système réactif?

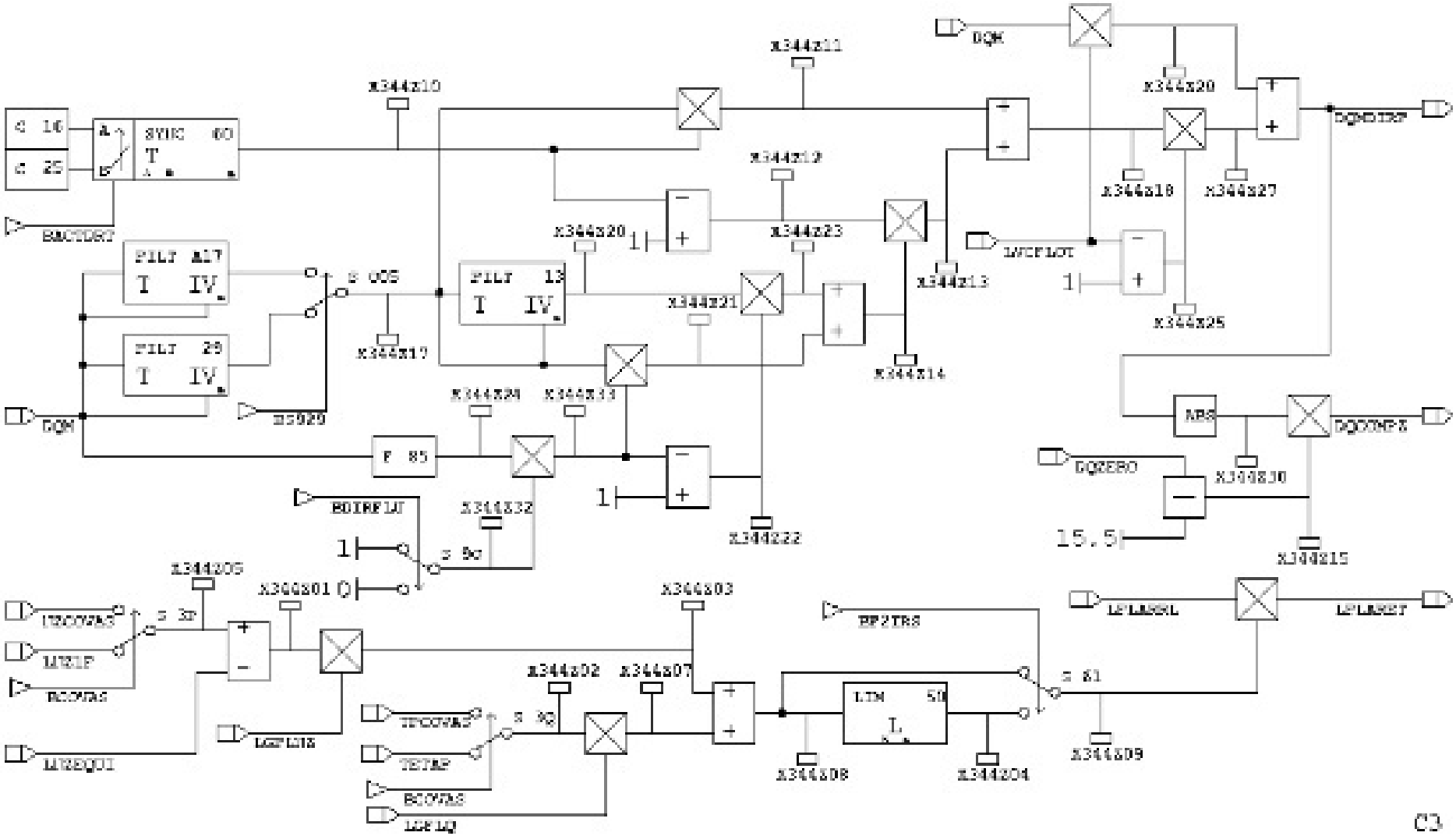
- commande de vol: $\approx$ 1,5MLOC
- logiciel critique
- “dynamique” = “trop tard”
- exécution cyclique
- architecture redondée

Écrire du code Assembleur/C/C++/JavaScript/Python/OCaml/Coq à la main?

Et le comparer alors a quoi?

Quelle est la spec?

SAO (Spécification Assistée par Ordinateur) — Airbus 80's



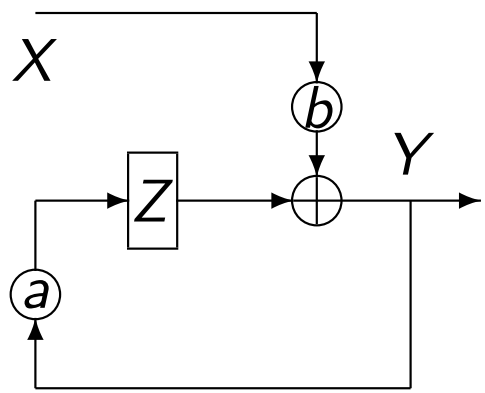
Des dessins très précis...

Les automaticiens/traiteurs de signaux décrivaient les systèmes de contrôle/commande avec des mathématiques très précises avant même l'arrivée de calculateurs.

Equations de suites, transformée en Z, etc.

Exemple: un filtre linéaire

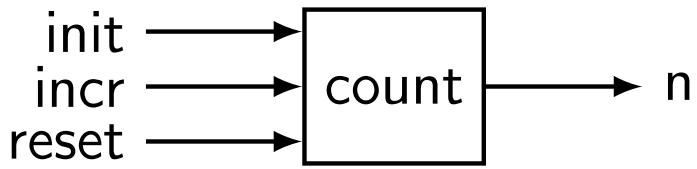
$$Y_0 = bX_0 \ , \ \forall n \ Y_{n+1} = aY_n + bX_{n+1}$$



...mais non exécutables! Écrire du code et se convaincre qu'il est juste?

Comment rendre ces mathématiques exécutables?

Quelque part à Grenoble... le langage Lustre (1987)




```
node COUNT (init, incr: int; reset: bool)
   returns (n: int);
let
   n = init ->
       if reset then init else pre(n) + incr;
tel;
```

Programmer en écrivant des équations de suites

Un système discret: une fonction de suites; les suites sont synchrones.

X	1	2	1	4	5	6	...
Y	2	4	2	1	1	2	...
$X + Y$	3	6	3	5	6	8	...
pre X	*nil*	1	2	1	4	5	...
Y -> X	2	2	1	4	5	6	...

L'équation $Z = X + Y$ signifie $\forall n. Z_n = X_n + Y_n$.

Le temps est logique: les entrées X et Y arrivent "en même temps"; la sortie Z est produite "en même temps"

Euh... c'est temps réel?

Raisonner en pire cas: vérifier que le code généré produit la sortie avant l'arrivée de l'entrée suivante.

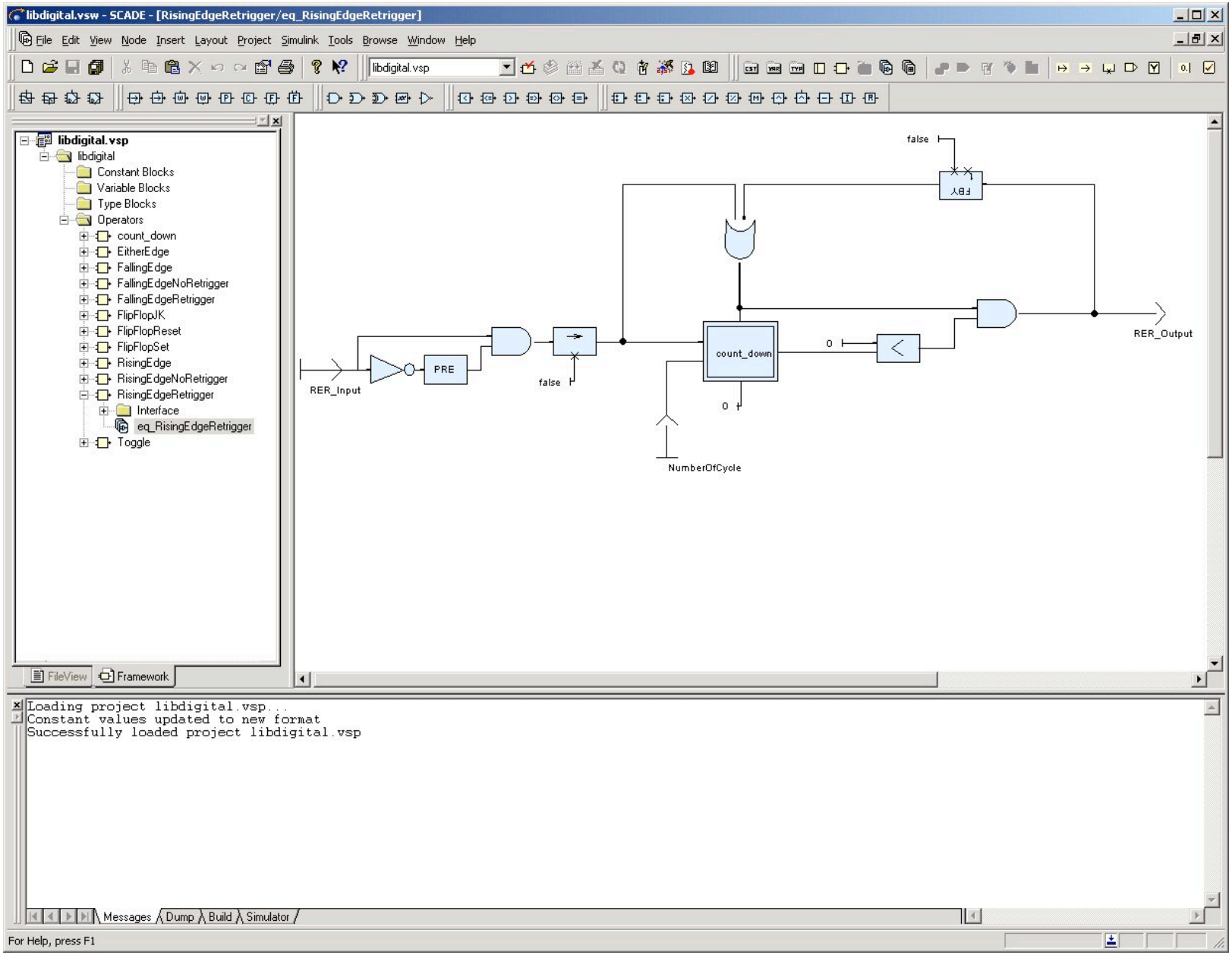
L'idée géniale de Lustre

Programmer en écrivant directement des équations mathématiques

Les analyser/transformer/simuler/tester/vérifier

Les traduire automatiquement vers du code exécutable

SCADE: Safety Critical Application Dev. Env. (Verilog, 95)



Au même moment...

À Rennes: le langage Signal (1987)

- Même approche que Lustre mais un peu plus expressif.
- Un système = un ensemble de contraintes sur des suites.

À Nice: le langage Esterel (1985)

- Style impératif plus proche de l'informatique.
- Interruption, suspension d'une tâche, mise en parallèle.
- Comment rendre cela déterministe?

La même approche synchrone:
(1) raisonner idéalement; (2) calculer le WCET du code généré.

C'est bien plus simple.

Mais certain programmes conduisent à des monstres...

comment les rejeter?

Différente échelles de temps

Synchroniser des processus lents et rapide?

X	1	2	3	4	5	...
half	*true*	*false*	*true*	*false*	*true*	...
X *when half*	1		3		5	...
$X + (X$ *when half*$)$	2		5		8	...

```
let half = true -> not (pre half);
    o = x + (x when half);
tel
```

Définit la suite: $\forall n \in \mathbb{N}.o_n = x_n + x_{2n}$

- ne peut être implémentée à mémoire (buffer) bornée;
- le rejeter statiquement: on peut le faire par typage.

Analyser les causalités entre signaux

Des programmes ont zéro solutions (*deadlock*) ou trop (*non déterminisme*)

En Lustre/Signal

- `x = y + 1 and y = x + 2`
- `y = x and x = y`

En Esterel

- `present S else emit S`
- `present S1 then emit S2 || present S2 else emit S1`

Découverte

La "bonne" notion de causalité est celle de l'électricité.

Si on "cable" le programme synchrone, les sorties sont-elles stables?

Coincide avec ce qui est démontrable en logique constructive.

Un peu après, quelque part entre Grenoble et Jussieu...

Lucid Synchrone et ReactiveML

Une idée de Paul Caspi, en 1994, à Grenoble.

> *"Marc, observe bien, on peut écrire des programmes Lustre récursifs en quelques lignes de LazyML!"*

Très expressif: ordre supérieur, inférence des types, récursivité, etc.

mais les "monstres" sont toujours la.

du typage, du typage, du typage... et adapter la compilation.

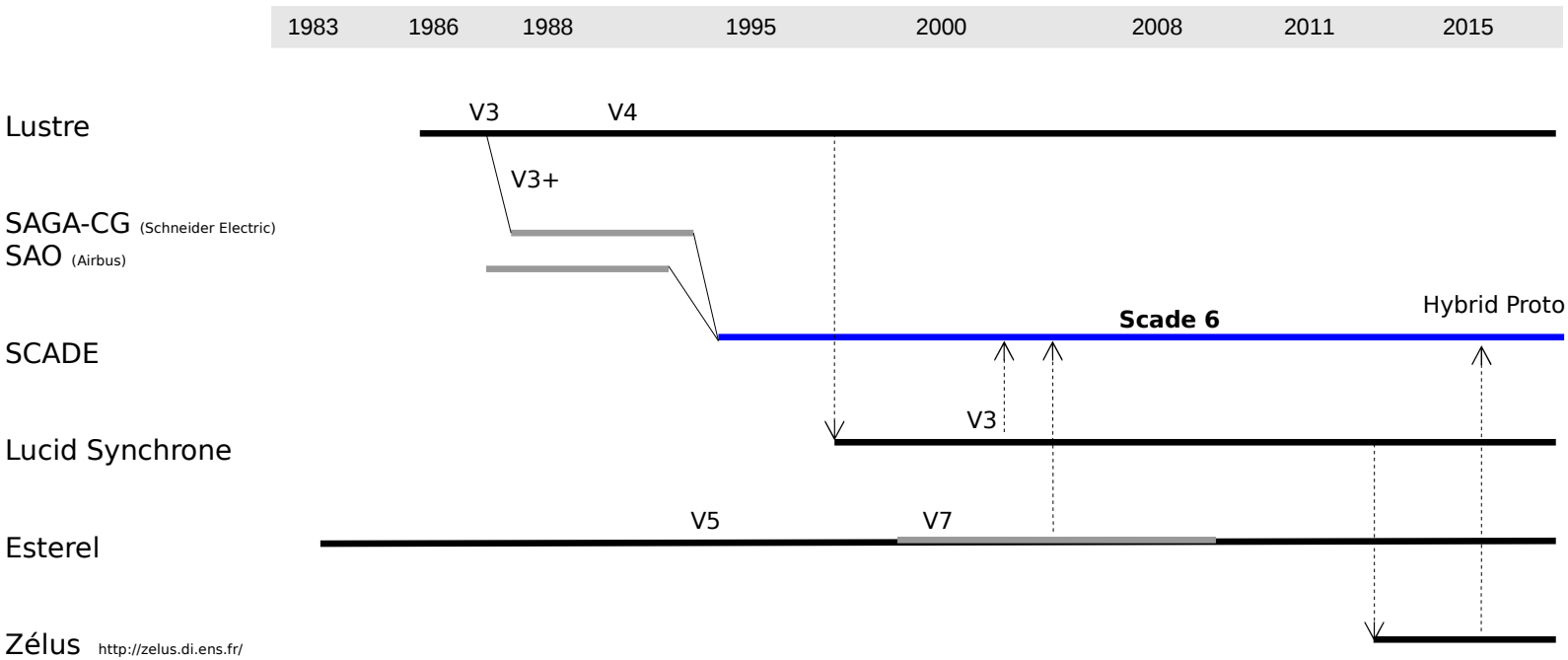
Lucid Synchrone (95-05)

Construire un langage synchrone fonctionnel avec des traits de ML

ReactiveML (05-15)

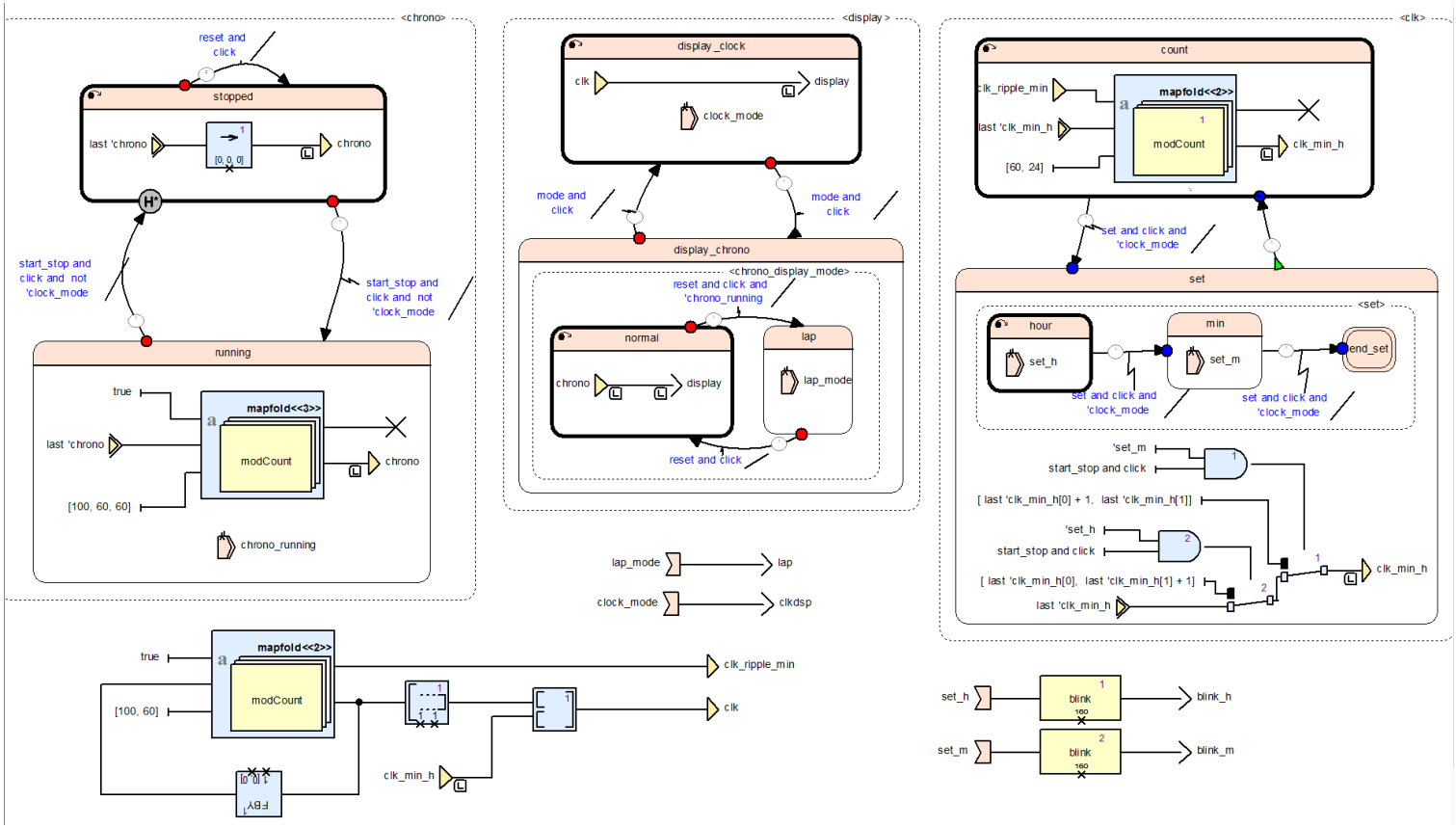
Du parallélisme synchrone dans un langage à la ML (OCaml)

Timeline


1983
1986
1988
1995
2000
2008
2011
2015
Lustre
V3
V4
V3+
SAGA-CG (Schneider Electric)
SAO (Airbus)
SCADE
Scade 6
Hybrid Proto
Lucid Synchrone
V3
Esterel
V5
V7
Zélus http://zelus.di.ens.fr/

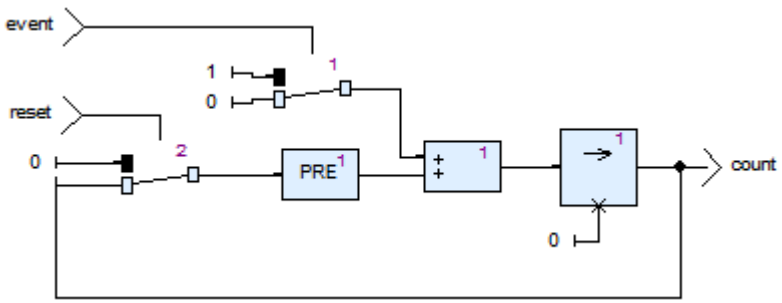
SCADE 6 (2008-): la grande unification

Un nouveau langage et un nouveau compilateur, en OCaml
(et plein d'idées de Lucid Synchrone dedans)



Tes dessins sont justes? Prouve le!

‘What you prove is what you execute’ (Berry ’89)




```
L1 = pre L7;
L2 = (L11) -> (L5);
L3 = event;
L4 = reset;
count = L2;
L5 = L6 + L1;
L6 = if L3 then (L8) else (L9);
L7 = if L4 then (L10) else (L2);
L8 = 1;
L9 = 0;
L10 = 0;
L11 = 0;
```

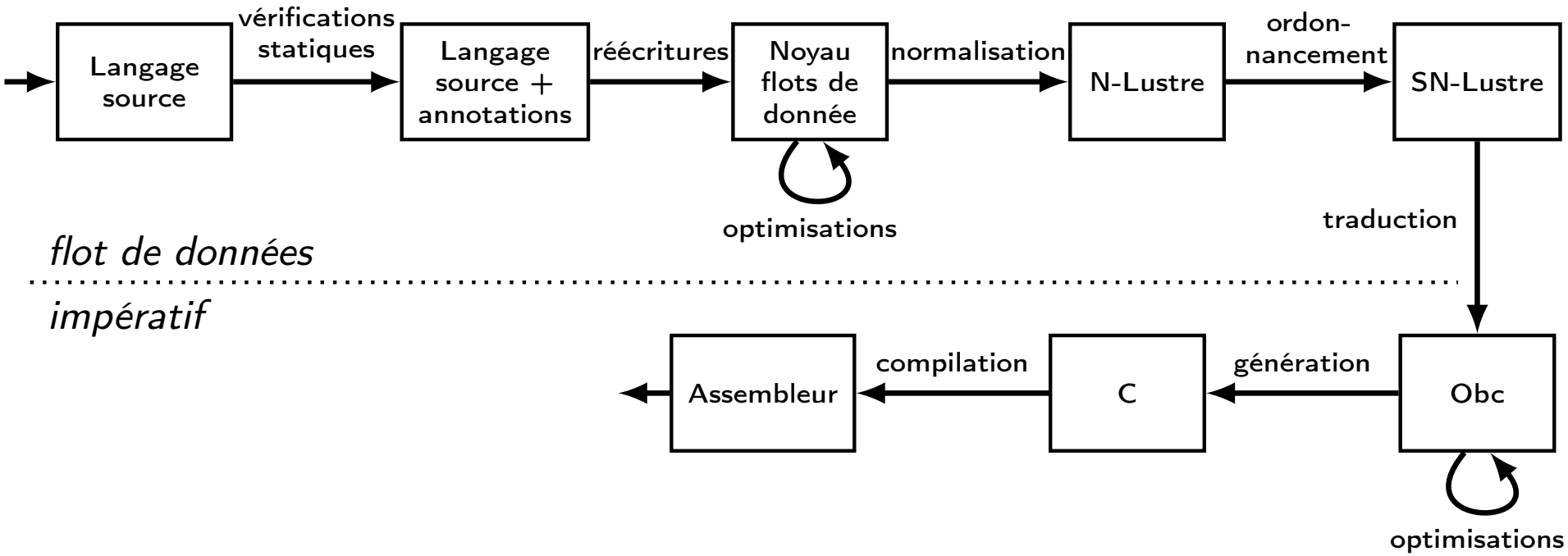
code gen.

```
void counter_reset(outC_counter *outC)
{
  outC->init = kcg_true;
}

void counter(inC_counter *inC, outC_counter *o
{
  kcg_int tmp;

  if (outC->init) {
    outC->count = 0;
  }
  else {
    if (inC->event) {
      tmp = 1;
    }
    else {
      tmp = 0;
    }
    outC->count = tmp + outC->_L9;
  }
  if (inC->reset) {
    outC->_L9 = 0;
  }
  else {
    outC->_L9 = outC->count;
  }
  outC->init = kcg_false;
}
```

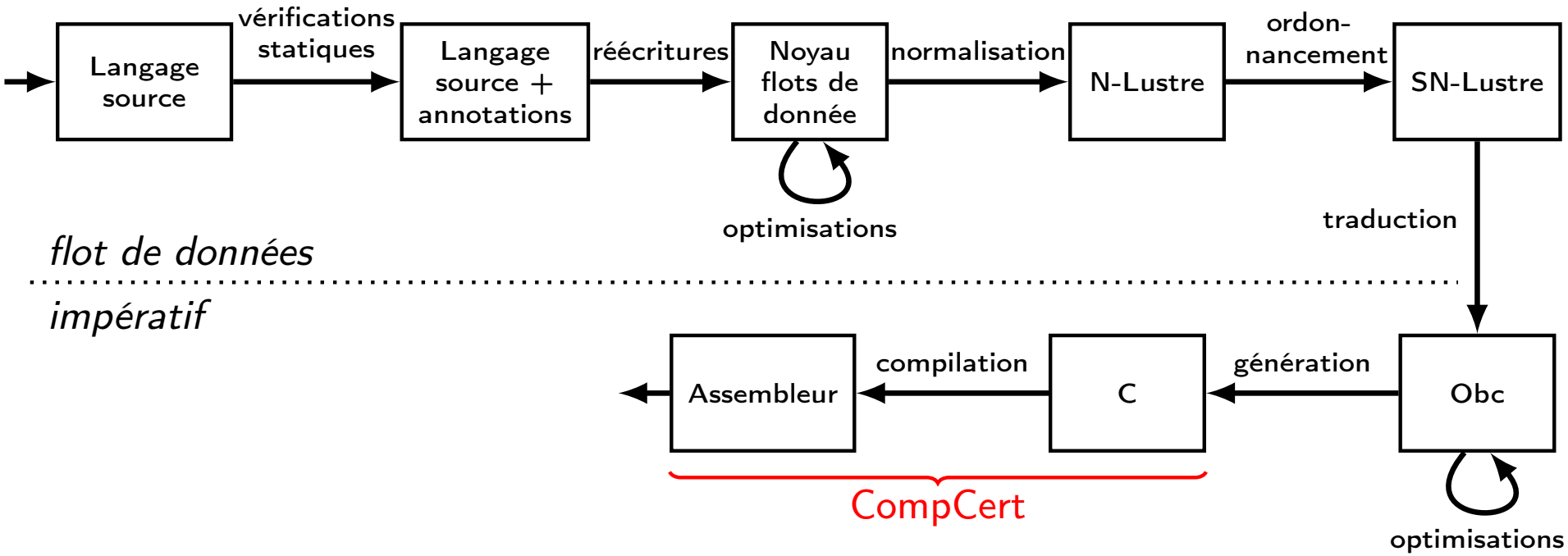
Architecture d'un compilateur synchrone



SCADE KCG 6 (Esterel Tech.)

- Qualifié avec la norme la plus stricte de l'avionique (DO178C/A)
- Évite la re-vérification que le code est conforme aux modèle SCADE.
- Basé sur la traçabilité du processus.

Architecture d'un compilateur synchrone



Génération de code formelle vérifiée

- combiné avec le compilateur CompCert pour un preuve complète jusqu'au binaire.
- implémenter des optimisations plus agressives

Vélus: the lemme ultime

- travail en cours (Timothy Bourke et al.): PLDI'17
- 20kLOC de Coq.

```
Lemma behavior_asm:
  ∀ G P main ins outs,
    wc_global G →
    wt_global G →
    wt_ins G main ins →
    wt_outs G main outs →
    sem_node G main (vstr ins) (vstr outs) →
    compile G main = OK P →
    ∃ T, program_behaves (Asm.semantics P) (Reacts T)
         ∧ bisim_io G main ins outs T.
```

correctness of clocks

correctness of types

input/output well typed

data-flow semantic m

the compilation succee

then, the assembly code
produces an infinite trace...

...which corresponds to the
data-flow model

Conclusion

Des langages dédiés, parallèles et déterministes

adaptés à la culture mathématique et la pratique des ingénieurs

propres et adoptés dès le début

des idées utilisées dans d'autres applications: web, trading haute fréquence, musique mixte, ChatBot, etc.

comment traiter les systèmes hybrides (continu/discret)?

comment générer du code parallèle prévisible?