# Combiner Java et réécriture, c'est possible et utile

11 mars 2010

Pierre-Etienne Moreau
LORIA - INRIA - Ecole des Mines de Nancy

# Tom

- Termes
- Filtrage
- Stratégies

dans des langages classiques

# Constructions de haut niveau

# Fondations théoriques solides

# Utilisé dans les milieux académiques et industriels

- décrire des transformations
- support à la recherche / prototypage
- compilateurs
- outils de preuve
- traduction de requêtes

*tom.loria.fr*

# Travail d’équipe

E. Balland

A. Reilles
Dassault Systemes

R. Kopetz
Yahoo

C. Kirchner

C. Ringeissen

M. Vittek

# Réécriture

expressif - exécutable - formel

# Concepts principaux

a $\rightarrow$ b : *règle* décrivant une transformation

S : *stratégie* contrôlant les applications

# Exemple

*système de réécriture*

$$\left.\begin{array}{l} x + zero \to x \\ x + s(y) \to s(x + y) \end{array}\right\} R$$

*terme*

s(zero) + s(s(zero))

*stratégie*

innermost(R)

innermost(R)[s(zero) + s(s(zero))] ➠ s(s(s(zero)))

# Permet de raisonner

- terminaison
- confluence
- complexité
- combinaison

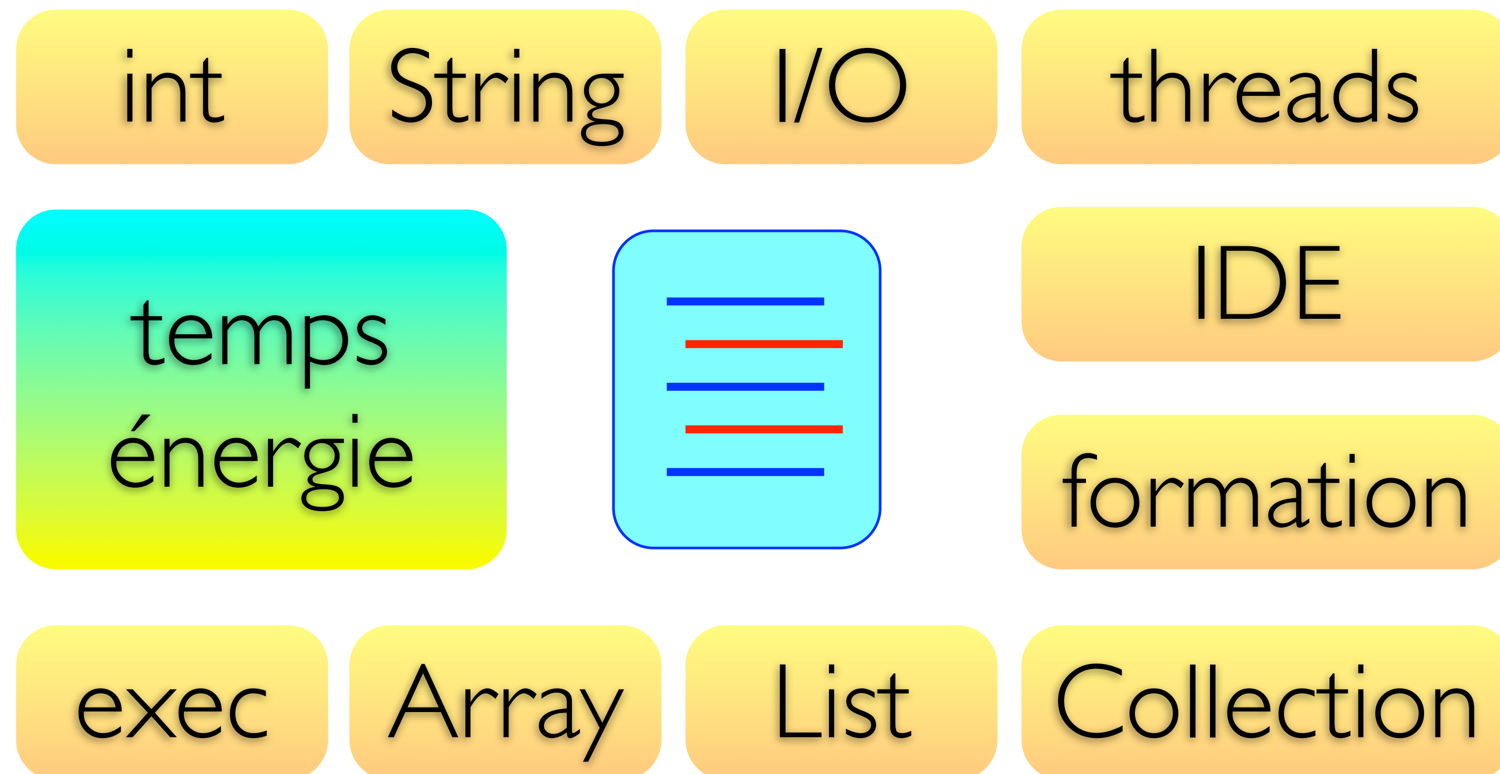
# Réécriture

à la base de nombreux langages

- OBJ[1976, 1985]
- *ELAN* [1993, 1999]
- Maude [1996]
- ASF+SDF[1985], Rascal[2009]
- Stratego[1998], DMS[1998], Hats[2003], ...

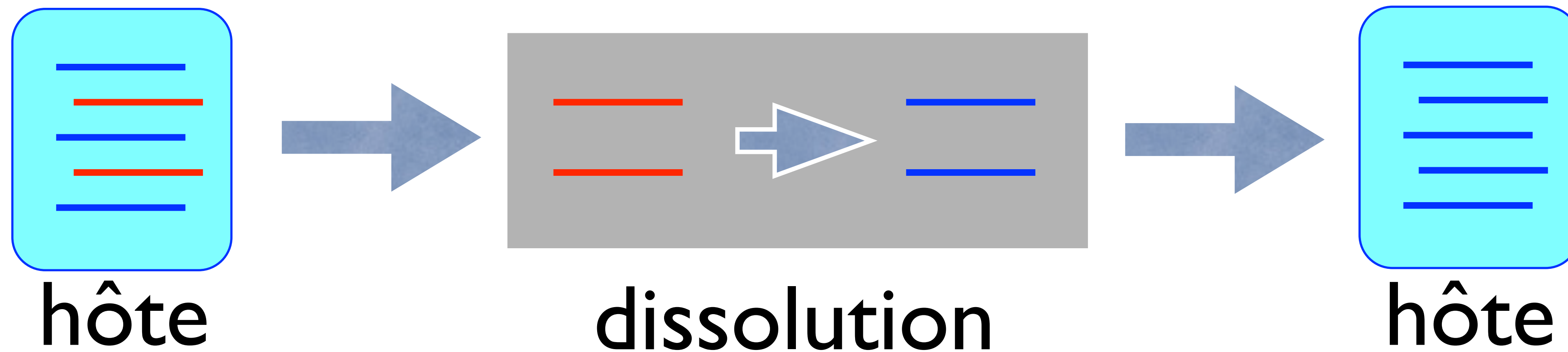
# Comment rendre utilisables les concepts liés à la réécriture ?

# Création d'un langage




- *ELAN*, Maude, ASF+SDF, Stratego, ...
- Caml, Clean, Haskell, Scala, ...

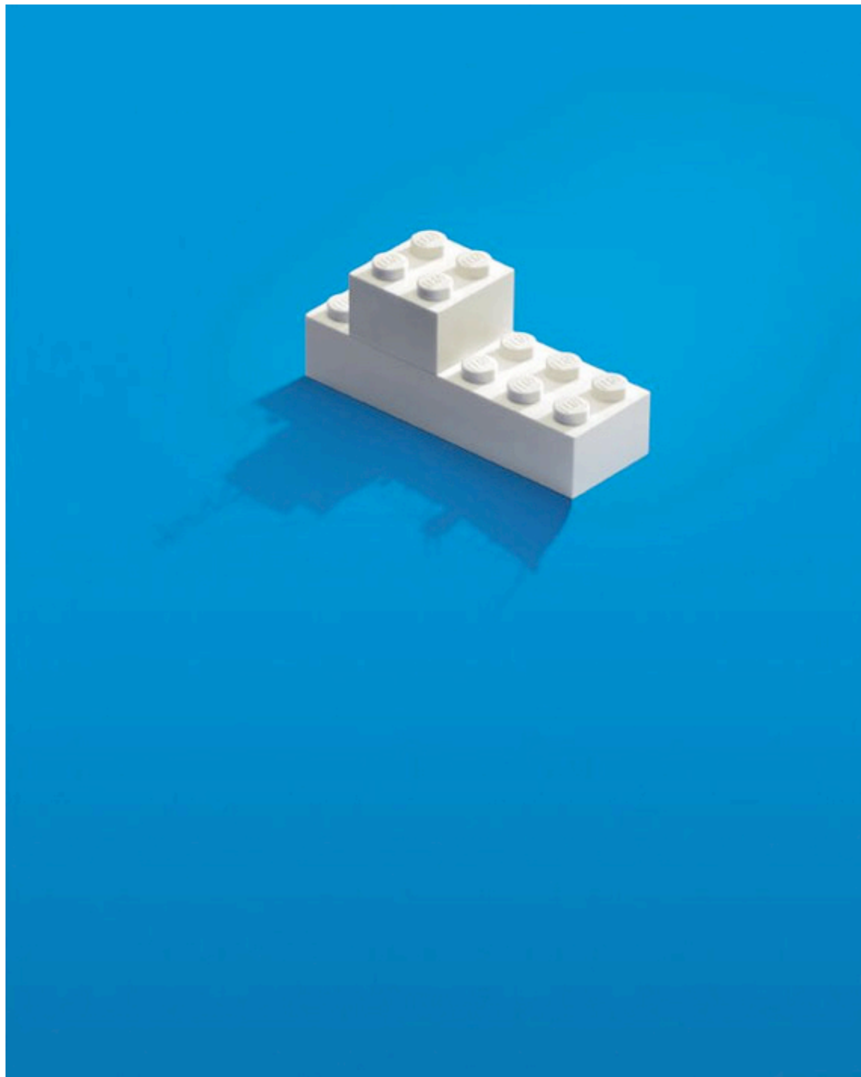
# A piggyback ride



# Java

langage objet - portable - largement utilisé

# Demo



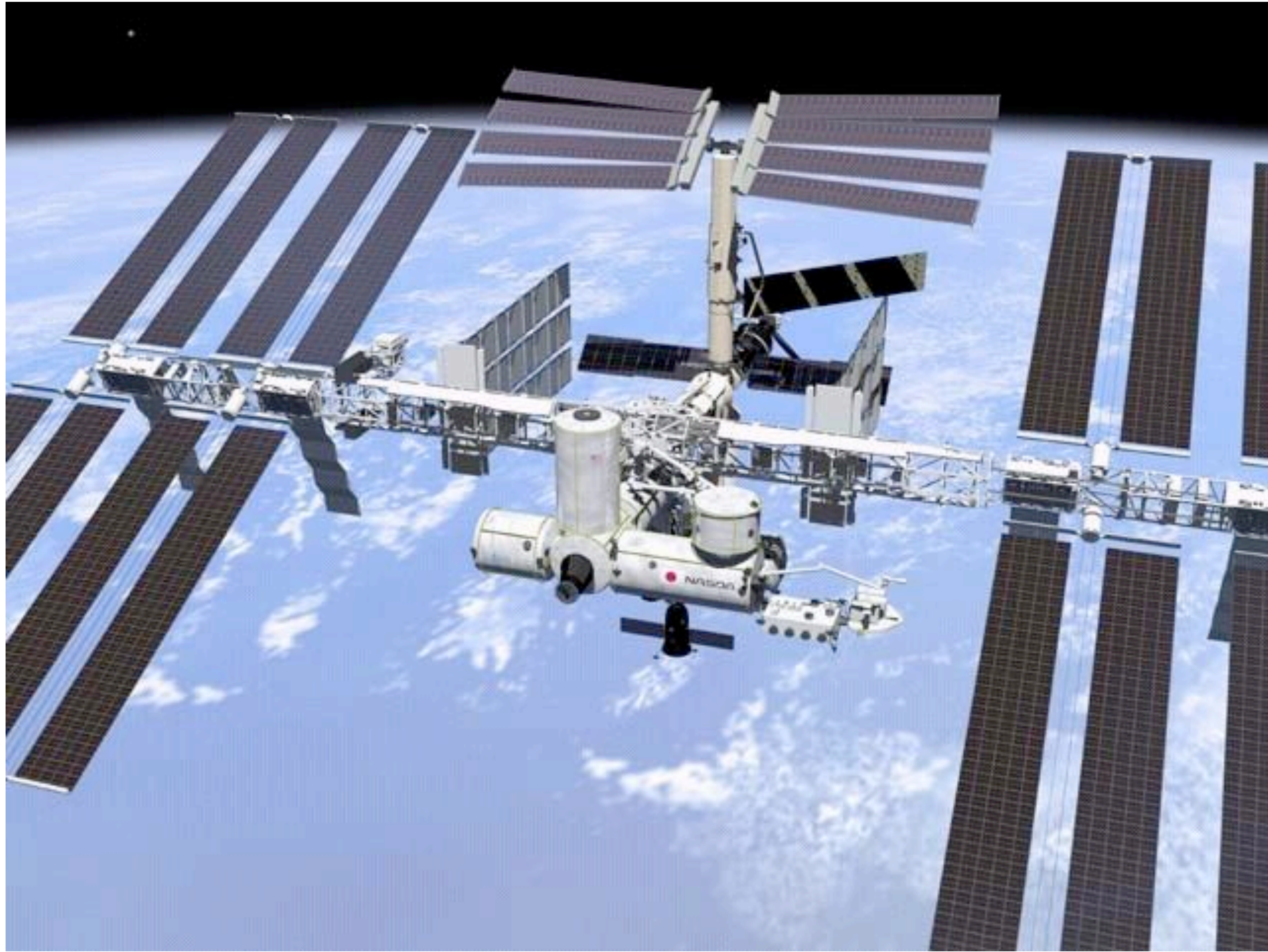
- signature
- filtrage
- réécriture
- filtrage associatif

# Résumé

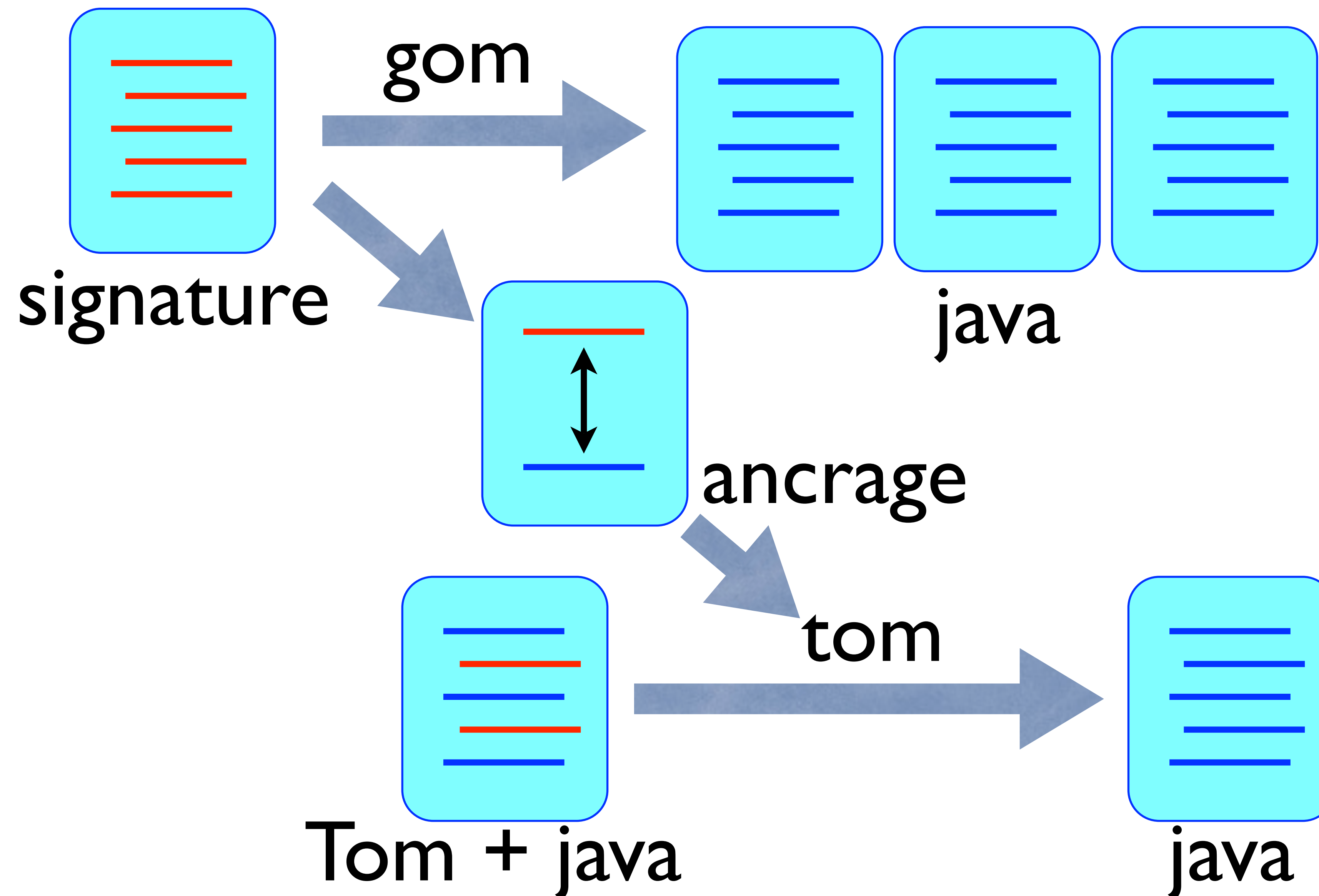
- construction de termes
- filtrage
- manipulation de listes
- représentation efficace en mémoire

# Goodies

- filtrage AC
- term-graphes
- syntaxe XML
- connexion aisée avec un parseur
- taches ant

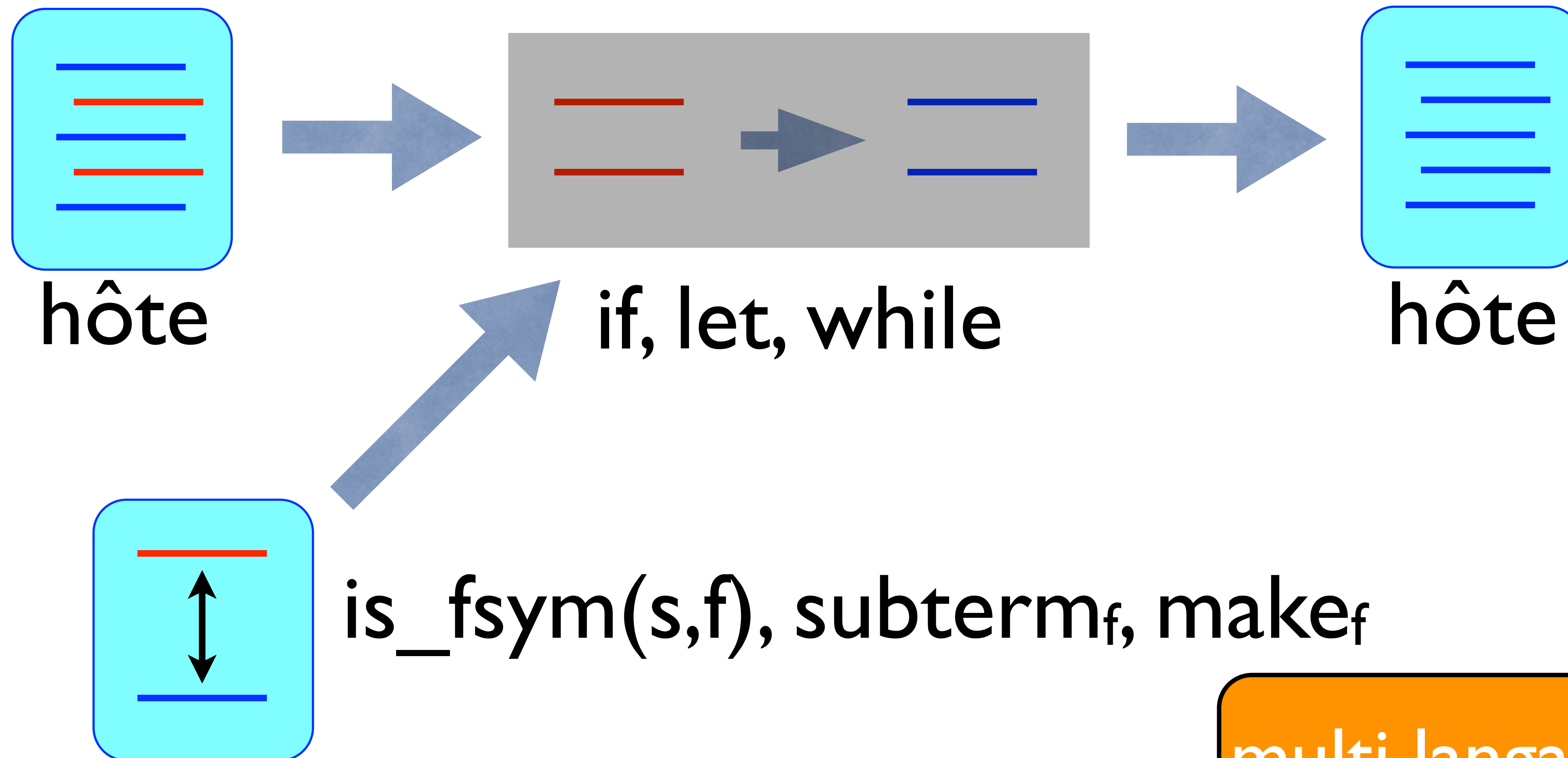


# Station MIR




Tom connait-il le code généré par Gom ?

# A piggyback ride




multi langages :
Java, C, C#, Caml, Python, ...

# Intérêt des ancrages

- structures C
- termes Caml
- DOM XML
- code généré par Eclipse EMF

permet de s'adapter à des structures existantes

# Demo

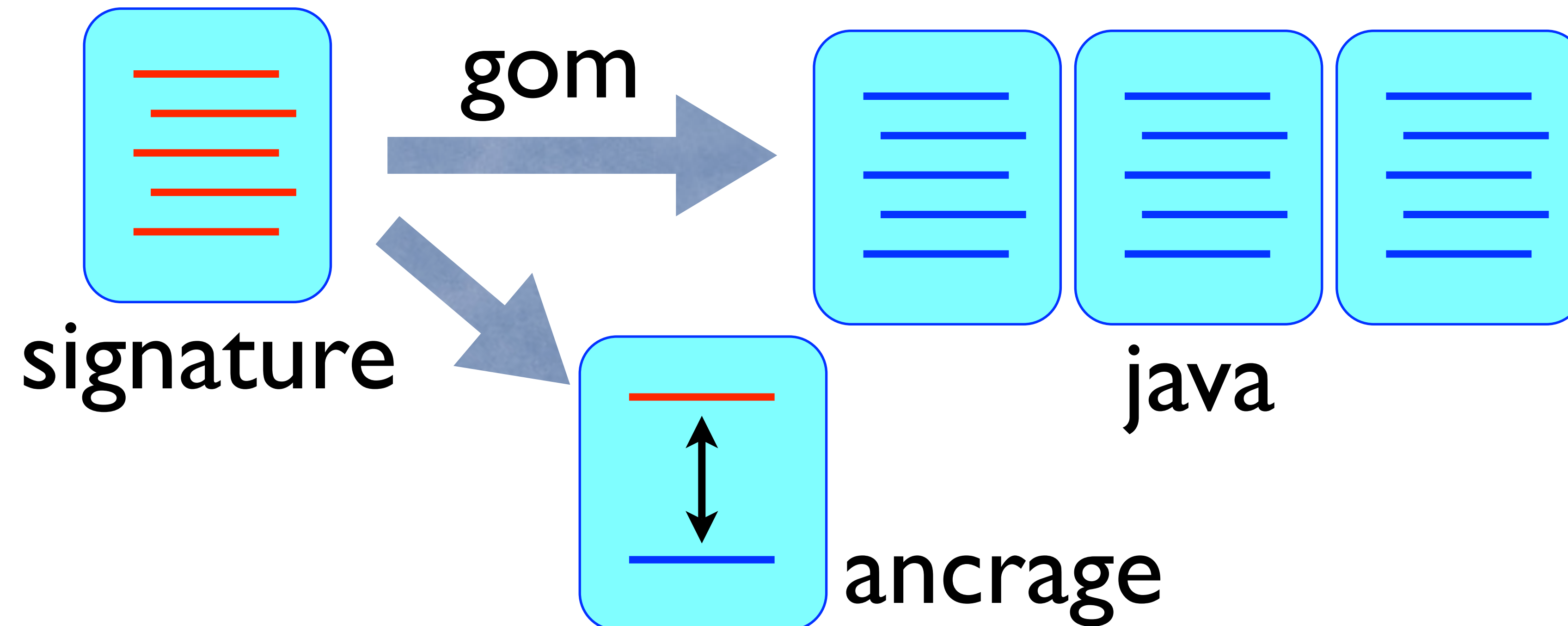
ancrage String

# Approches possibles

- partir de structures existantes
  - écrire l’ancrage à la main
  - inférer automatiquement un ancrage
- générer des structures de données
  - générer l’ancrage

# Génération

# Générateur de structures




partage maximal
typage fort
invariants
API (getters, compareTo, fromString, getCollection, etc.)

# Règles de réécriture

```
module Bool
abstract syntax
Bool = True()
       | False()
       | Not(b:Bool)
       | And(l:Bool,r:Bool)
       | Or(l:Bool,r:Bool)
```

Signature Gom

```
module Bool:rules() {
  Or(True(),_) → True()
  Or(_,True()) → True()
}
```

Règle de normalisation

```
public static void main(String[] args) {
  System.out.println(`Or(False(),True()));
}
>True()
```

# Actions exécutées lors de la construction

```
module Bool
abstract syntax
Bool = True()
     | False()
     | Not(b:Bool)
     | And(l:Bool,r:Bool)
     | Or(l:Bool,r:Bool)
```

Signature Gom

```
Not:make(b) {
  %match(Bool b) {
    True()    → { return `False(); }
    Not(x)    → { return `x; }
    And(l,r)  → { return `Or(Not(l),Not(r)); }
    Or(l,r)   → { return `And(Not(l),Not(r)); }
  }
}
```

Hook de normalisation

```
System.out.println(`Not(Or(False(),False())))
>And(True(),True())
System.out.println(`Not(And(False(),True())));
>True()
```

# Exemples d’utilisation

# Enseignement

```
module Minijava
imports int String
abstract syntax
Stm = Seq(s1:Stm, s2:Stm)
    | Assign(id:String, e:Exp)
    | Print(el:ExpList)

Exp  = Id(id:String)
    | Num(n:int)
    | Op(e1:Exp,op:BinOp,e2:Exp)
```

```
public static Env eval(Stm s, Env env) {
  %match(s) {
    Seq(s1,s2) → {
      return eval(`s2,eval(`s1,env));
    }
    Assign(id,e) → {
      return insertEnv(`id,eval(`e,env).getVal(),env);
    }
    ...
  }
  throw new RuntimeException(
        "Unknown statment: " + s);
}
```

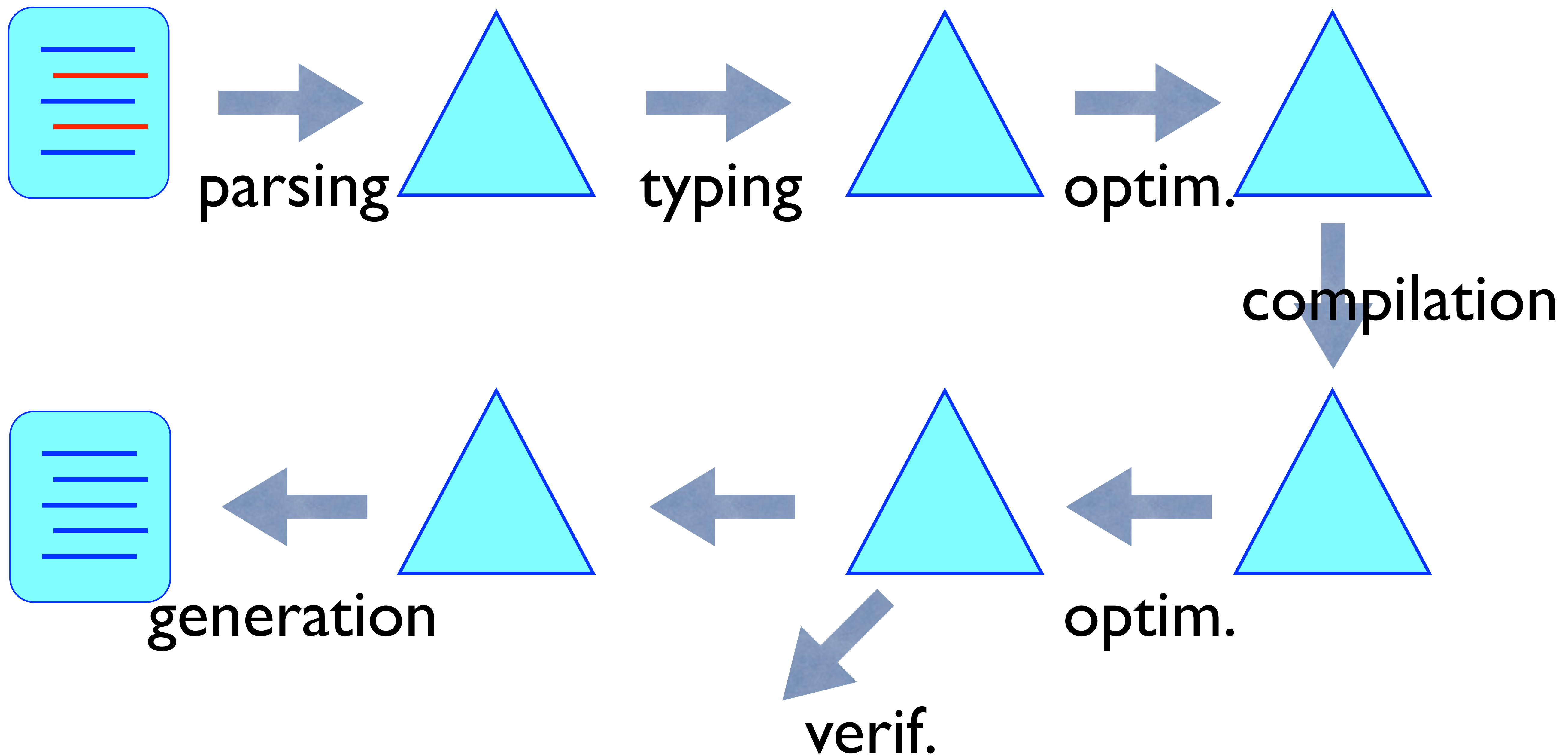
# Recherche

- analyser, transformer des AST
  - compilation, optimisation, analyse, etc.
  - environnement pour DSL
- manipuler des preuves
- modéliser des systèmes et des fonctions de transition

# Industrie

- Manipulation d’AST
  - analyse de programmes Java
  - transformation de requêtes (MDX → SQL)
  - optimisation de requêtes

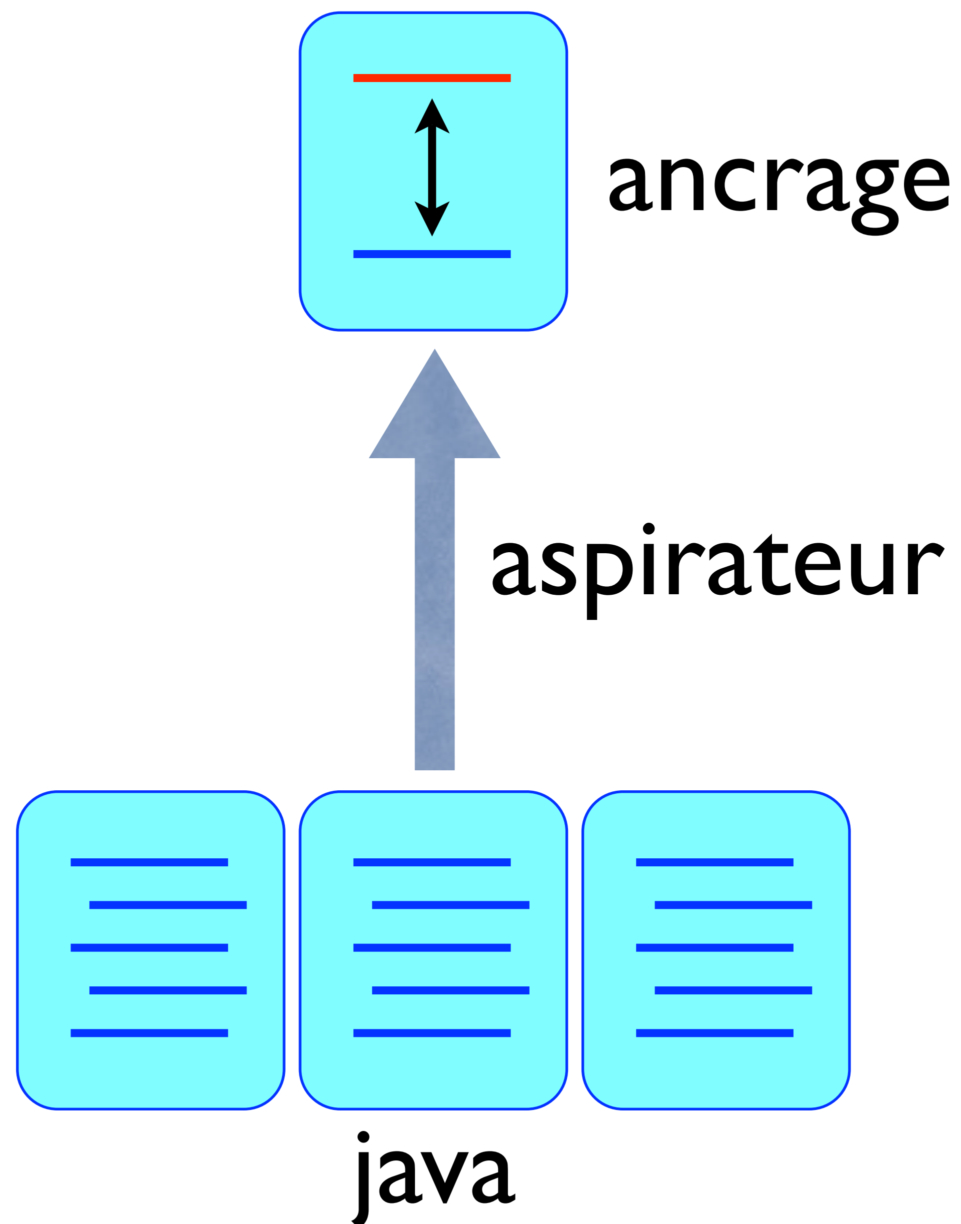
# Tom écrit en Tom



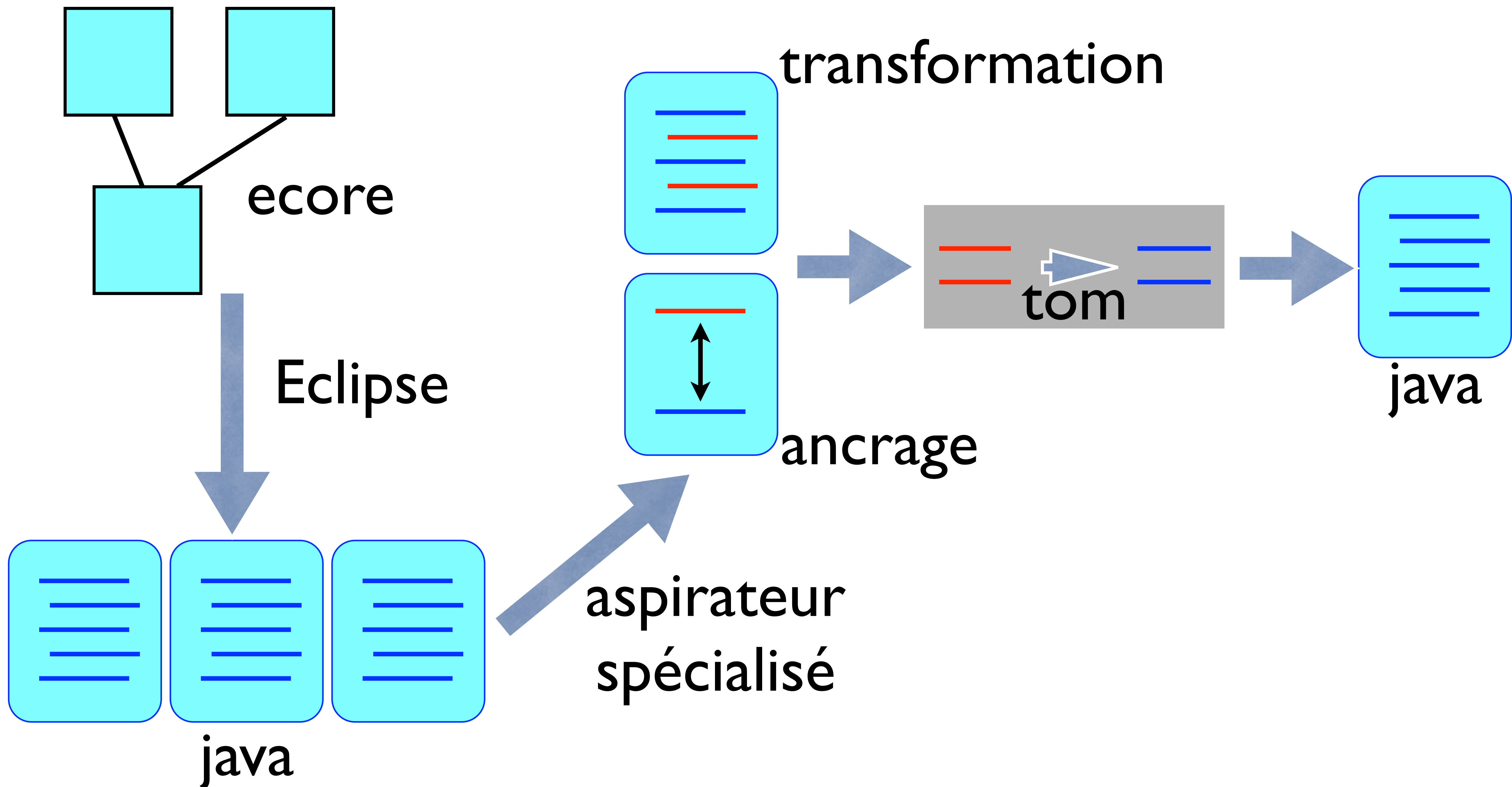
# Aspirateur

Générateur d’ancrage

# Aspirateur de structures



# Application à Eclipse Modeling Framework



# Générateur vs. aspirateur

- génération : belle intégration, permet d’innover et de diffuser les idées
- aspiration : nouveaux marchés, nous incite à généraliser nos outils

# Positionnement

- similitudes avec Caml
- multi-langages mais Java est au coeur
- similitudes avec Pizza et Scala
- nos différences :
  - multi-résultats (list-matching)
  - pas de programmation fonctionnelle

il manque un concept propre à la réécriture

# Séparer les règles du contrôle

$(x+y)*z \rightarrow (x*z)+(y*z)$

$z*(x+y) \rightarrow (z*x)+(z*y)$

$(x*z)+(y*z) \rightarrow (x+y)*z$

$(z*x)+(z*y) \rightarrow z*(x+y)$

distrib$((x+y)*z) \rightarrow (x*z)+(y*z)$

distrib$(z*(x+y)) \rightarrow (z*x)+(z*y)$

distrib$(x+y) \rightarrow$ distrib$(x)$ + distrib$(y)$

distrib$(x*y) \rightarrow$ distrib$(x)$ * distrib$(y)$

$a + ((a+b)*c)$

réécriture innermost

$a + a*c + b*c$

termine ?

confluent ?

S = repeat(distrib) ; repeat(facto)

# Stratégies

- un DSL pour contrôler l'application des règles
- sépare les règles "métier" de leur contexte d'utilisation
- augmente leur ré-utilisation

# Stratégies

- objet qui s'applique sur un terme
- pour produire un nouveau terme
- peut échouer

# Stratégies élémentaires

- identité
- échec
- règle de réécriture
  - $(a \rightarrow b)[a] = b$
  - $(a \rightarrow b)[c] =$ échec
  - $(a \rightarrow b)[f(a)] =$ échec

# Combinateurs élémentaires

- $s_1 ; s_2$ (séquence)
- $s_1 <+ s_2$ (choix de gauche à droite)

# Exercice

- définir la stratégie *try(s)* telle que :
  - $(try(a \rightarrow b))[a] = b$
  - $(try(a \rightarrow b))[c] = c$

# Stratégie composée

- try(s) = s <+ id

# Exercice

- définir la stratégie *repeat(s)* qui applique *s*, tant que *s* n’échoue pas
- exemple : (repeat(0+x → x))[0+0+a] = a

# Stratégie récursive

- repeat(s) = (s ; repeat(s)) <+ id

# Peut-on décrire innermost ?

outermost ?

# Congruence générique

- all(s) : applique s à tous les descendants
- one(s) : applique s à un descendant

# all et one

- (all(s))[cst] = cst
- (all(a → b))[g(a,a)] = g(b,b)
- (all(a → b))[g(a,b)] = échec

- (one(s))[cst] = échec
- (one(a → b))[g(a,a)] = g(b,a)
- (one(a → b))[f(f(a))] = échec

# innermost

- oncebu(s) = one(oncebu(s)) <+ s
- innermost(s) = repeat(oncebu(s))

# Quelques stratégies

- oncebu(s) = one(oncebu(s)) <+ s
- innermost(s) = repeat(oncebu(s))
- oncetd(s) = s <+ one(oncetd(s))
- bottomup(s) = all(bottomup(s)) ; s
- topdown(s) = s ; all(topdown(s))
- innermost(s) = all(innermost(s)) ; try(s ; innermost(s))

# En Tom

- une stratégie est un terme
- un objet Java
- qui s'applique sur un autre terme (s.visit(t))

```
Strategy innermost(Strategy s) {
  return `repeat(oncebu(s));
}
```

$t_2$ = innermost(regle_ab).visit($t_1$);

# Si vous avez tout suivi

- all, one, identity, fail, ; et <+ sont des builtins
- écrits en Java
- on peut définir un ancrage pour les voir comme des termes
- on peut donc filtrer, et appliquer des stratégies sur des stratégies

transformation dynamique de programmes

# Demo

règle, stratégie, collect

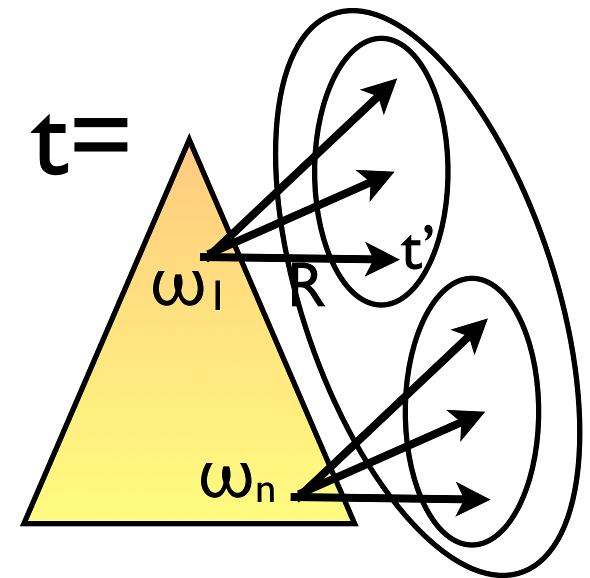
# Résumé

- séparer règles et contrôle
- permet de traverser un terme
  - collecter de l'information
  - effectuer des transformations

# Un formalisme puissant

- stratégies paramétrées par des objets Java
- s'applique sur une structure de donnée quelconque (ancrage)
- s'applique sur une stratégie
- maintient une méta-information : la position où sont appliquées les règles

# Un problème à résoudre




- soit R = lhs→rhs et t
- trouver les t' tels que t $\rightarrow_{R,\omega}$ t'
- { $t[\sigma_1\ rhs]_{\omega 1},\ldots,t[\sigma_m\ rhs]_{\omega n}$ }
- *exercice* : calculer cet ensemble en Java, C, ELAN, Maude, Stratego, ASF+SDF, ... *(not easy)*
- *difficulté* : se souvenir du contexte $t[\ldots]_\omega$

# Positions explicites

*solution* : $\{ t[\sigma_1 \text{ rhs}]_{\omega 1}, \ldots, t[\sigma_m \text{ rhs}]_{\omega n} \}$

R(Collection c) {

lhs → c.add($t[\sigma \text{ rhs}]_{\omega}$)

}

- positions explicites
- accès aux ancêtres

# Demo

explore

# Je vous ai parlé

- de Tom : *http://tom.loria.fr*
- de programmation par filtrage
- de stratégies
- *mais pas* : de graphe, d'anti-pattern, de certification, *etc.*

# Travaux en cours

- Stratégies
  - de plus haut niveau
  - montrer des propriétés
  - caractériser les formes normales

# Travaux en cours

- Chercher de nouveaux concepts pour
  - analyser des systèmes
  - représenter et transformer des modèles
  - modéliser des politiques de sécurité

# Géographie

Royaume-Uni
Manche
Belgique
Allemagne
Lux.
Calais
Lille
Amiens
Rouen
Cherbourg
Caen
Perros-Guirec
Brest
St-Malo
Rennes
PARIS
Chartres
1h30
1h15
Metz
Strasbourg
Colmar
Auxerre
Dijon
Besançon
Suisse
Nantes
Tours
Bourges
Poitiers
Mâcon
La Rochelle
Limoges
Clermont-Ferrand
Lyon
Annecy
Chambéry
Grenoble
océan Atlantique
Royan
Bordeaux
Italie
Biarritz
Pau
Toulouse
Nîmes
Avignon
Nice
Montpellier
Marseille
Toulon
Carcassonne
Perpignan
Espagne
mer Méditerranée
100 km
© InterCarto

# Pareo - Tom

- aide pour utiliser Tom
  - enseignement
  - recherche
- projets communs (ANR, etc.)

*tom.loria.fr*