AST

Analyse statique pour l'optimisation de programme

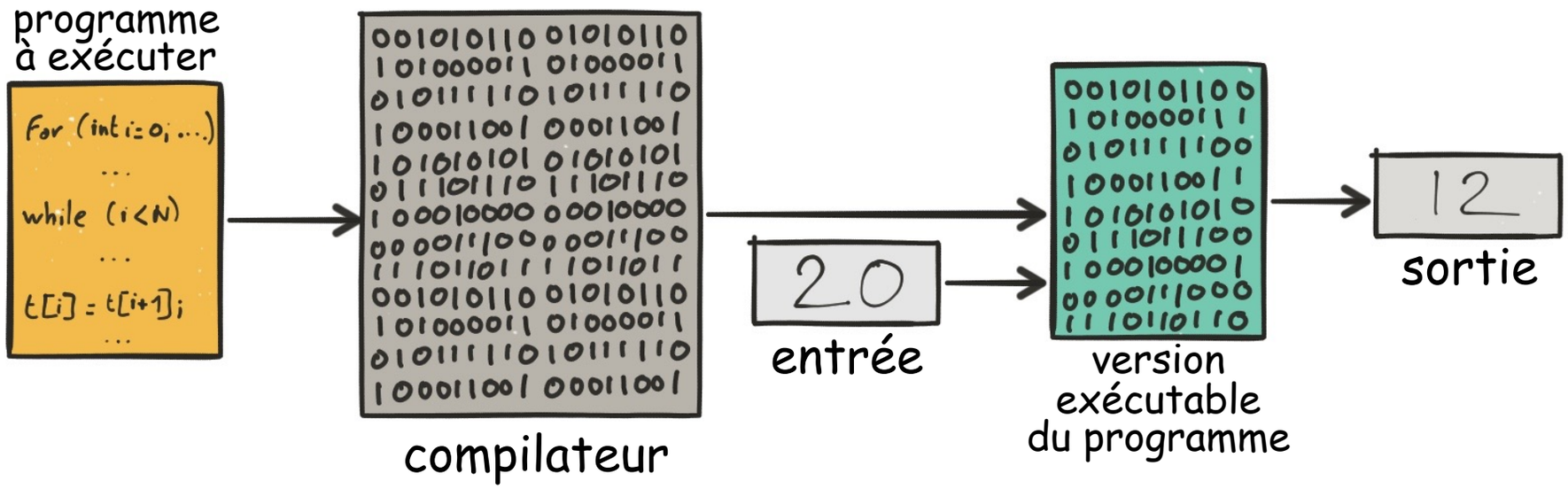
10 janvier 2020

David Pichardie

Les compilateurs

Qu'est-ce qu'un compilateur ?

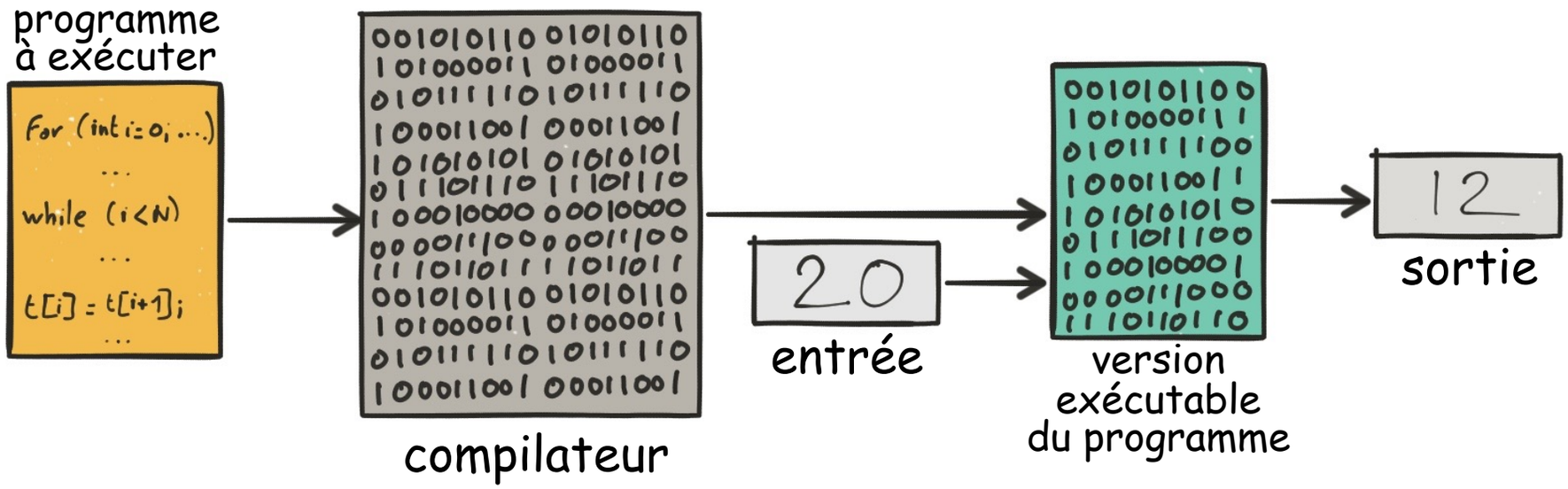
- un programme qui traduit un programme exprimé dans un langage source, en un programme exécutable dans un autre langage



Les compilateurs

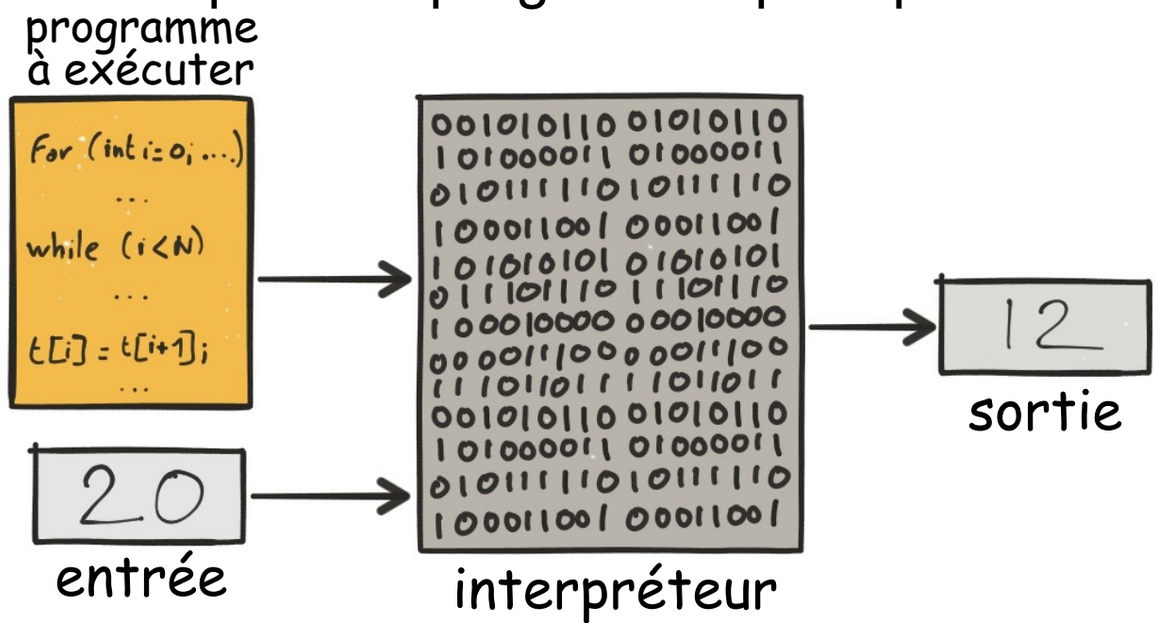
Qu'est-ce qu'un compilateur ?

- un programme qui traduit un programme exprimé dans un langage source, en un programme exécutable dans un autre langage



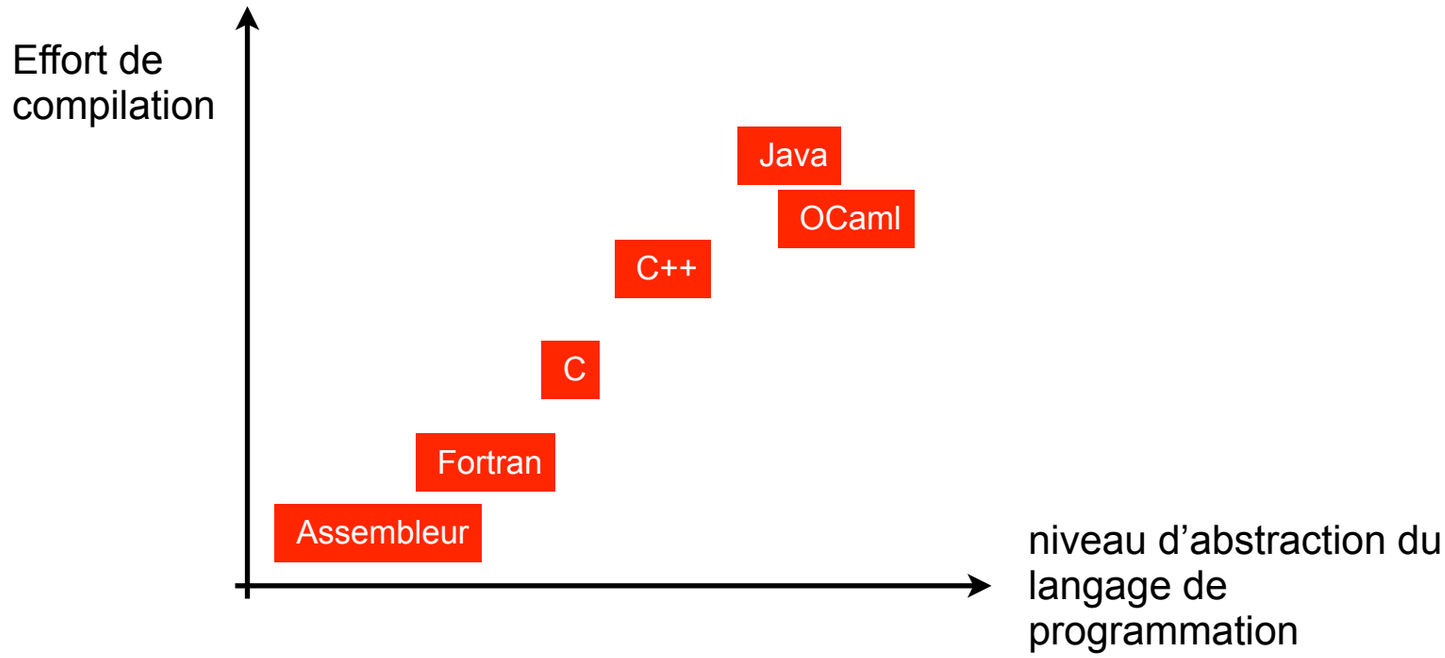
Qu'est-ce qu'un interpréteur ?

- un programme qui lit un programme pour produire le résultat de son exécution



Compiler : est-ce difficile ?

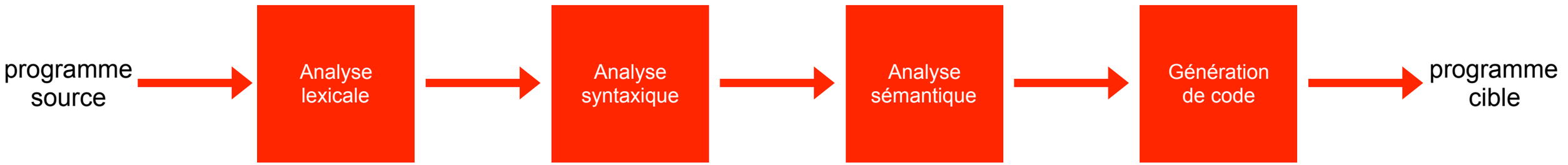
Les langages de haut-niveau d'abstraction diminuent le travail du programmeur mais augmentent celui du programmeur.



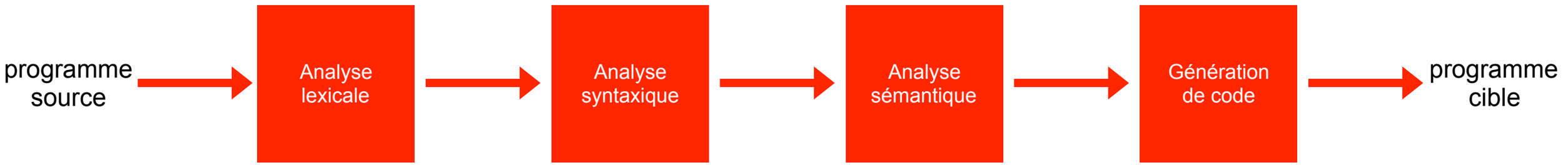
Autres difficultés

- il faut passer passer à l'échelle des programmes gigantesques
- les langages dynamiques (Java, Javascript) doivent être compilés pendant l'exécution : *Just-in-time compiler*
- il faut tenir compte des spécificités de chaque architecture cible, et savoir en tirer partie

Les grandes étapes de compilation



Les grandes étapes de compilation



Théorie des automates finis et des langages rationnels

Théorie des automates à pile et des langages algébriques

Sémantique des langages de programmation

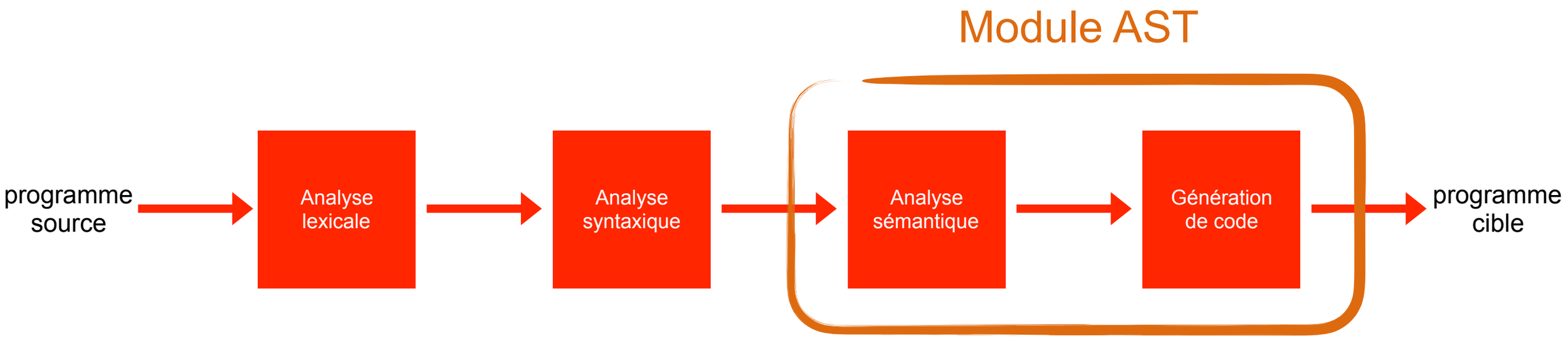
Interprétation abstraite

Algorithmique des problèmes NP-complets

Algorithmique parallèle

Architecture des ordinateurs

Les grandes étapes de compilation



Théorie des automates finis et des langages rationnels

Théorie des automates à pile et des langages algébriques

Sémantique des langages de programmation

Interprétation abstraite

Algorithmique des problèmes NP-complets

Algorithmique parallèle

Architecture des ordinateurs

Quelles sont les qualités attendues de la part d’un compilateur ?

Quelles sont les qualités attendues de la part d'un compilateur ?

1. préserve, voir améliore, le comportement du programme qu'il compile
2. génère du code efficace
3. compile rapidement
4. temps de compilation environ proportionnel à la taille du programme
5. compilation séparée
6. messages d'erreurs explicites
7. interagit bien avec un debugger

Correction d’un compilateur

Si le compilateur traduit un programme p en un programme p’, alors les comportements observables de p et p’ coincident

Correction d’un compilateur

- une première version

> Si le compilateur traduit un programme p en un programme p’, alors les comportements observables de p et p’ coincident

Correction d’un compilateur

- une première version

> Si le compilateur traduit un programme p en un programme p’, alors les comportements observables de p et p’ coincident

- en C, un programme source peut avoir des comportements indéfinis (division par zéro, débordements d’entiers). Le programme cible doit alors conserver le comportement jusqu’au moment où se comportement apparaît, et est libre de faire n’importe quoi ensuite (*amélioration*)

Correction d'un compilateur

- une première version

> Si le compilateur traduit un programme p en un programme p', alors les comportements observables de p et p' coincident

- en C, un programme source peut avoir des comportements indéfinis (division par zéro, débordements d'entiers). Le programme cible doit alors conserver le comportement jusqu'au moment où se comportement apparaît, et est libre de faire n'importe quoi ensuite (*amélioration*)

- dans les langages fortement typés, le compilateur utilise un système de type pour *filtrer* les programmes dont il est certains qu'ils n'auront que des comportements définis

Correction d'un compilateur

- une première version

> Si le compilateur traduit un programme p en un programme p', alors les comportements observables de p et p' coincident

- en C, un programme source peut avoir des comportements indéfinis (division par zéro, débordements d'entiers). Le programme cible doit alors conserver le comportement jusqu'au moment où se comportement apparaît, et est libre de faire n'importe quoi ensuite (*amélioration*)

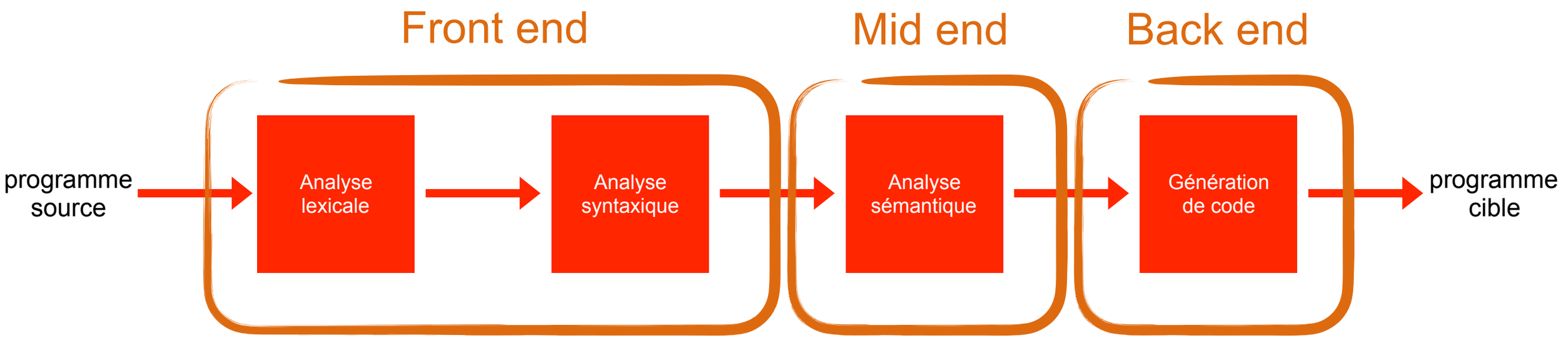
- dans les langages fortement typés, le compilateur utilise un système de type pour *filtrer* les programmes dont il est certains qu'ils n'auront que des comportements définis

- en programmation concurrente, le programme cible peut restreindre les entrelacements des threads (moins de comportements qu'au niveau source)

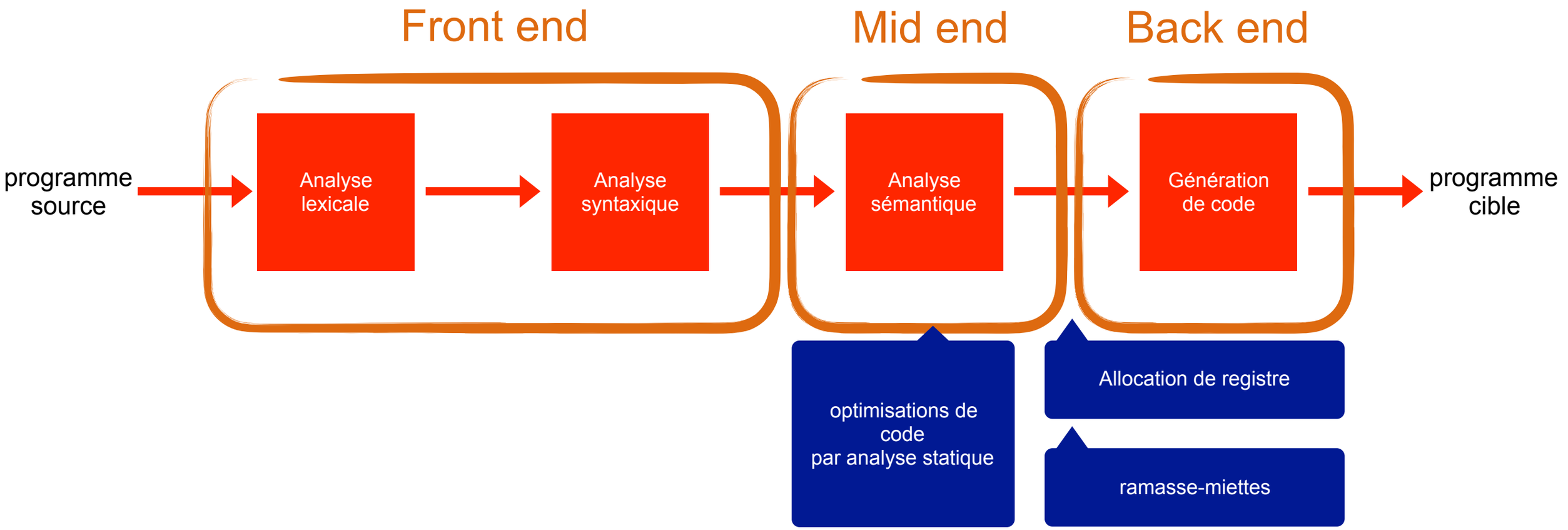
Questions

- Java est fortement typé : mais alors comment sont gérées les divisions par zéros ?
- Un système de type est toujours incomplet : il rejette des programmes sans comportement indéfinis. Donnez un exemple !

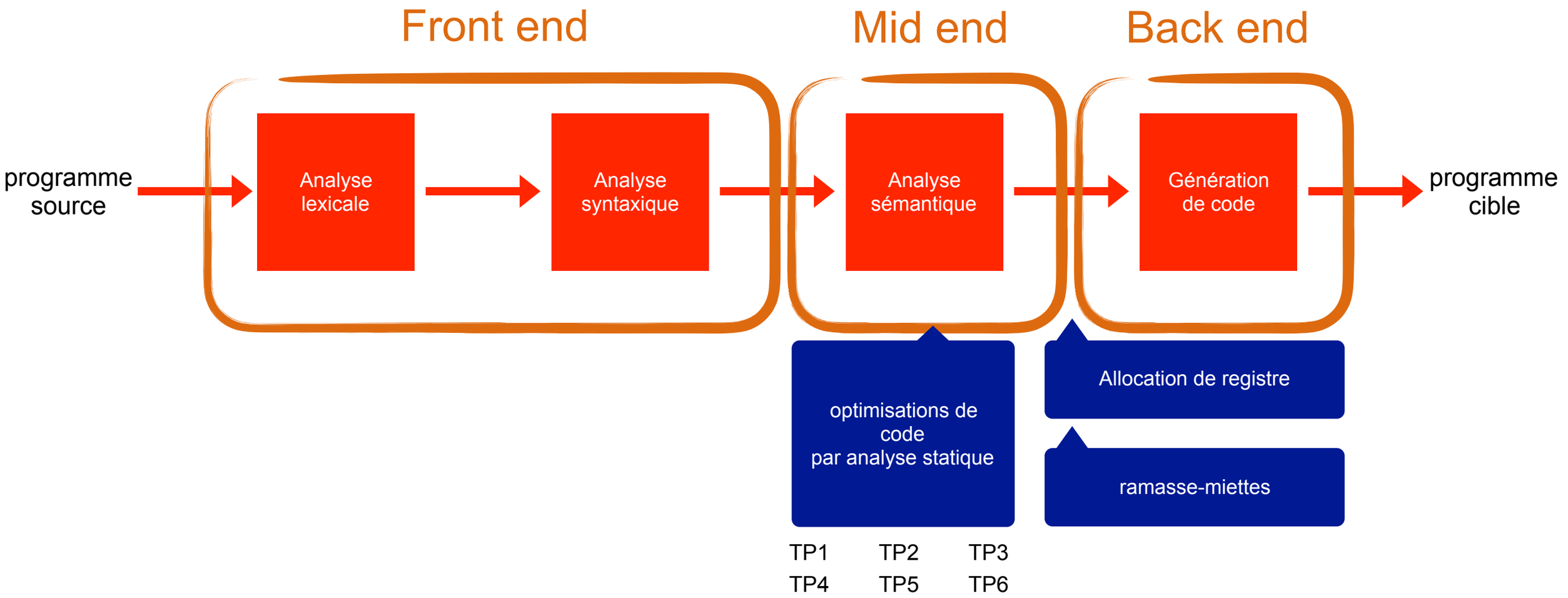
Architecture globale



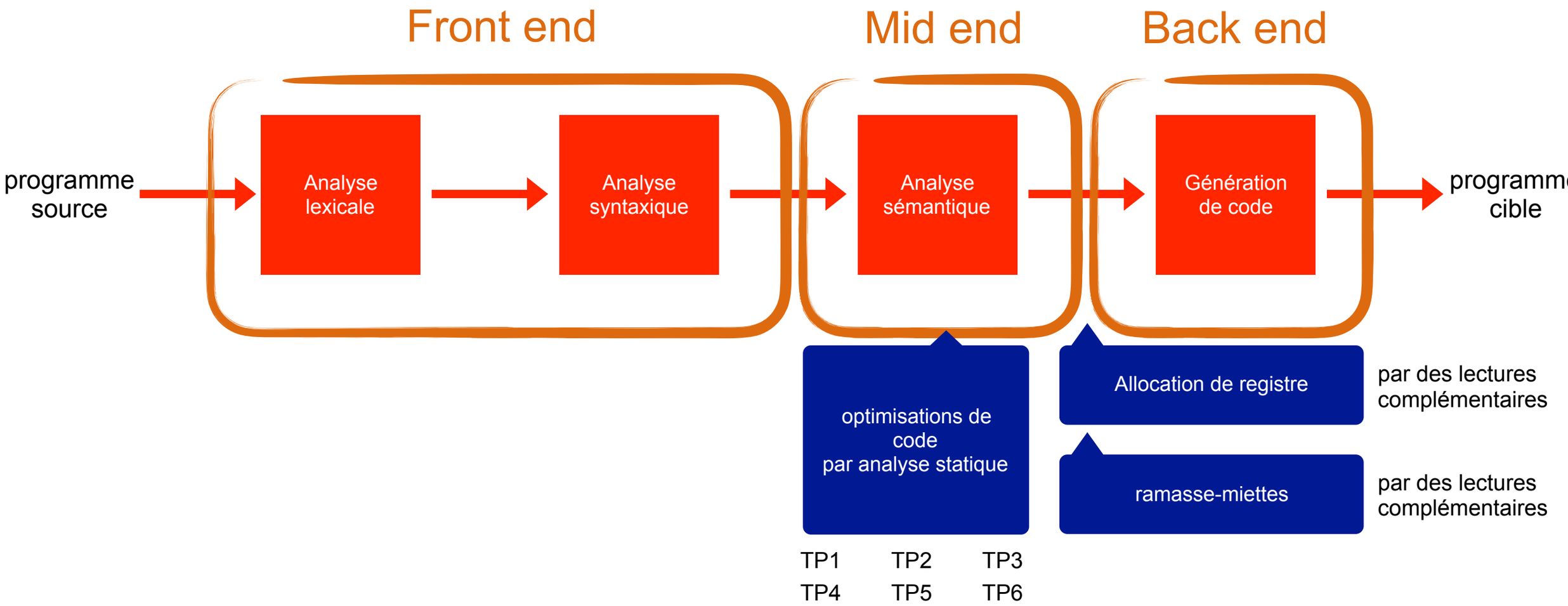
Ce que nous étudierons



Ce que nous étudierons

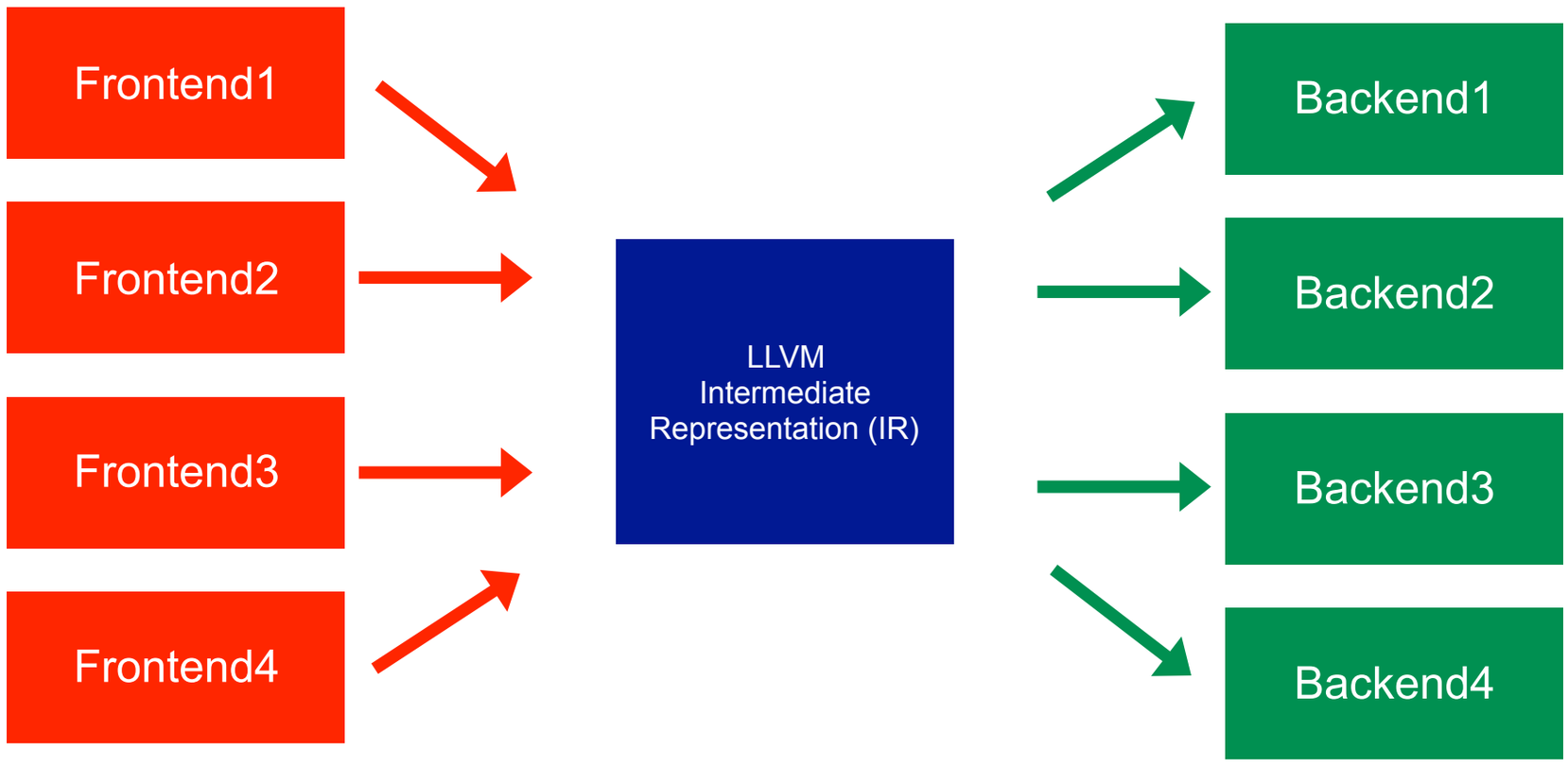


Ce que nous étudierons



Les bienfaits d'une bonne représentation intermédiaire : l'exemple LLVM

https://llvm.org



Compilation optimisante

Un exemple : le produit scalaire

```
double p = 0.0;
for (int i = 0; i < N, i++)
   p = p + A[i] * B[i];
return p;
```

Compilation optimisante

Passage en code intermédiaire *3-adresses*

```
double p = 0.0;
for (int i = 0; i < N, i++)
   p = p + A[i] * B[i];
return p;
```

```
p = 0.0;
for (int i = 0; i < N, i++)
   a = load(A + i * 8);
   b = load(B + i * 8);
   c = a * b;
   p = p + c;
}
...
```

Compilation optimisante

Les multiplications sont coûteuses...

```
p = 0.0;
for (int i = 0; i < N, i++)
   a = load(A + i * 8);
   b = load(B + i * 8);
   c = a * b;
   p = p + c;
}
...
```

```
p = 0.0;
for (int i = 0; i < N, i++)
   a = load(A);
   b = load(B);
   c = a * b;
   p = p + c;
   A = A + 8;
   B = B + 8;
}
...
```

Compilation optimisante

On regroupe l'itération i et i+1 dans un même corps de boucle

```
p = 0.0;
for (int i = 0; i < N, i++)
   a = load(A);
   b = load(B);
   c = a * b;
   p = p + c;
   A = A + 8;
   B = B + 8;
}
...
```

```
p = 0.0;
for (int i = 0; i < N-1, i += 2)
   a1 = load(A);
   b1 = load(B);
   c1 = a1 * b1;
   p = p + c1;
   a2 = load(A + 8);
   b2 = load(B + 8);
   c2 = a2 * b2;
   p = p + c2;
   A = A + 16;
   B = B + 16;
}
if (i < N) { ... }
```

Compilation optimisante

On réordonne certaines instructions pour profiter du pipeline d'instructions

```
p = 0.0;
for (int i = 0; i < N-1, i += 2)
   a1 = load(A);
   b1 = load(B);
   c1 = a1 * b1;
   p = p + c1;
   a2 = load(A + 8);
   b2 = load(B + 8);
   c2 = a2 * b2;
   p = p + c2;
   A = A + 16;
   B = B + 16;
}
if (i < N) { ... }
```

```
p = 0.0;
for (int i = 0; i < N-1, i += 2)
   a1 = load(A);
   b1 = load(B);
   a2 = load(A + 8);
   b2 = load(B + 8);

   c1 = a1 * b1;
   c2 = a2 * b2;
   A = A + 16;

   p = p + c1;
   B = B + 16;

   p = p + c2;
}
if (i < N) { ... }
```

Compilation optimisante

On effectue les lectures mémoires une itération en avance

```
p = 0.0;
for (int i = 0; i < N-1, i += 2)
   a1 = load(A);
   b1 = load(B);
   a2 = load(A + 8);
   b2 = load(B + 8);

   c1 = a1 * b1;
   c2 = a2 * b2;
   A = A + 16;

   p = p + c1;
   B = B + 16;

   p = p + c2;
}
if (i < N) { ... }
```

```
...
a3 = load(A); a4 = load(A + 8);
b3 = load(B); b4 = load(B + 8);
for (int i = 0; i < N-3, i += 2)
   a1 = a3; a3 = load(A + 16);
   b1 = b3; b3 = load(B + 16);
   c1 = a1 * b1;
   a2 = a4; a4 = load(A + 24);
   b2 = b4; b4 = load(B + 24);
   c2 = a2 * b2;
   A = A + 16;
   p = p + c1;
   B = B + 16;
   p = p + c2;
}
c1 = a3 * b3;
c2 = a4 * b4;
p = p + c1;
p = p + c2;
```

Compilation optimisante

Version complète

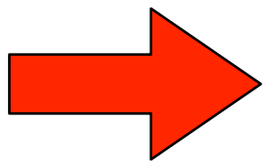
```
...
a3 = load(A); a4 = load(A + 8);
b3 = load(B); b4 = load(B + 8);
for (int i = 0; i < N-3, i += 2)
   a1 = a3; a3 = load(A + 16);
   b1 = b3; b3 = load(B + 16);
   c1 = a1 * b1;
   a2 = a4; a4 = load(A + 24);
   b2 = b4; b4 = load(B + 24);
   c2 = a2 * b2;
   A = A + 16;
   p = p + c1;
   B = B + 16;
   p = p + c2;
}
c1 = a3 * b3;
c2 = a4 * b4;
p = p + c1;
p = p + c2;
```

```
p = 0.0;
i = 0;
if (N >=4) {
  a3 = load(A); a4 = load(A + 8);
  b3 = load(B); b4 = load(B + 8);
  for (; i < N-3, i += 2)
    a1 = a3; a3 = load(A + 16);
    b1 = b3; b3 = load(B + 16);
    c1 = a1 * b1;
    a2 = a4; a4 = load(A + 24);
    b2 = b4; b4 = load(B + 24);
    c2 = a2 * b2;
    A = A + 16; p = p + c1;
    B = B + 16; p = p + c2;
  }
  c1 = a3 * b3;
  c2 = a4 * b4;
  p = p + c1; p = p + c2;
  A = A + 16; B = B + 16;
}
for (; i < N; i++) {
  a = load(A); A = A + 8;
  b = load(B); B = B + 8;
  c = a * b;
  p = p + c;
}
```

Compilation optimisante

Version complète

```
double p = 0.0;
for (int i = 0; i < N, i++)
   p = p + A[i] * B[i];
return p;
```

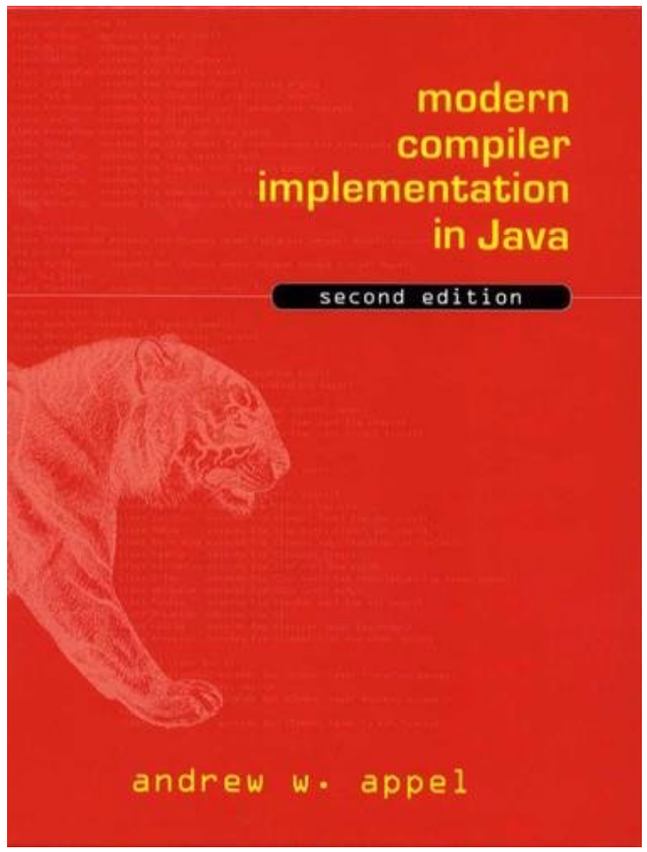


programme
2 à 3 fois plus
efficace

```
p = 0.0;
i = 0;
if (N >=4) {
  a3 = load(A); a4 = load(A + 8);
  b3 = load(B); b4 = load(B + 8);
  for (; i < N-3, i += 2)
    a1 = a3; a3 = load(A + 16);
    b1 = b3; b3 = load(B + 16);
    c1 = a1 * b1;
    a2 = a4; a4 = load(A + 24);
    b2 = b4; b4 = load(B + 24);
    c2 = a2 * b2;
    A = A + 16; p = p + c1;
    B = B + 16; p = p + c2;
  }
  c1 = a3 * b3;
  c2 = a4 * b4;
  p = p + c1; p = p + c2;
  A = A + 16; B = B + 16;
}
for (; i < N; i++) {
  a = load(A); A = A + 8;
  b = load(B); B = B + 8;
  c = a * b;
  p = p + c;
}
```

Méthode de travail

- Un ouvrage compagnon
  Andrew Appel and Jens Palsberg,
  Modern Compiler Implementation in Java, 2002



- Une série de 6 projets en Java, à rendre tout les 2 semaines (environs)
- 6 CM (incluants quelques exercices)
- 1 ou 2 TD en fin de semestre
- 12 TP (2 groupes de TP)
- Les TP peuvent être réalisés en binôme
- Nous vous fournirons certains chapitres du livre à lire pour approfondir

Évaluation

- Projets notés à rendre régulièrement (en binôme si possible)
  1. nous fournirons un jeu de tests
  2. nous évaluerons le projet avec un jeu étendu
  3. le projet compte pour 40% de la note finale
  4. des entretiens individuels seront organisés pour *affiner* certaines notes
- Examen terminal sur table
  1. une petite moitié de l'examen visera à évaluer votre implication dans les projets
  2. une deuxième moitié visera à évaluer votre compréhension du cours
  3. documents autorisés : nos slides imprimés (sans annotations)

Notre langage source : MiniJava

- Un sous-ensemble de Java
- Tiré de l'ouvrage de Appel & Palsberg
- Nous travaillerons ensuite sur une représentation intermédiaire appelée RTL, en vous fournissant un compilateur depuis MiniJava

Mini-Java : syntaxe

$$
\begin{array}{rrcl}
(Goal) & g & ::= & mc\ d_1\ \ldots\ d_n \\
(MainClass) & mc & ::= & \texttt{class}\ id\ \{\ \texttt{public static void main (String [] }\ id^S\texttt{)\{} \\
 & & & \quad t_1\ id_1;\ \ldots;\ t_r\ id_r;\ s_1 \ldots\ s_q\}\} \\
(TypeDeclaration) & d & ::= & \texttt{class}\ id\ \{\ t_1\ id_1;\ \ldots;\ t_f\ id_f;\ m_1\ \ldots\ m_k\ \} \\
 & & | & \texttt{class}\ id\ \texttt{extends}\ id^P\ \{\ t_1\ id_1;\ \ldots;\ t_f\ id_f;\ m_1\ \ldots\ m_k\ \} \\
(MethodDeclaration) & m & ::= & \texttt{public}\ t\ id^M\ \texttt{(}t_1^F\ id_1^F\texttt{,}\ \ldots\texttt{,}\ t_n^F\ id_n^F\texttt{)}\ \{ \\
 & & & \quad t_1\ id_1;\ \ldots;\ t_r\ id_r;\ s_1\ \ldots\ s_q\ \texttt{return}\ e;\ \} \\
(Type) & t & ::= & \texttt{int[]} \mid \texttt{boolean} \mid \texttt{int} \mid id \\
(Statement) & s & ::= & \{\ s_1\ \ldots\ s_q\ \} \mid id\ \texttt{=}\ e; \mid id\ \texttt{[}\ e_1\ \texttt{]}\ \texttt{=}\ e_2; \\
 & & | & \texttt{if (}\ e\ \texttt{)}\ s_1\ \texttt{else}\ s_2 \mid \texttt{while (}\ e\ \texttt{)}\ s \mid \texttt{System.out.println(}\ e\ \texttt{);} \\
(Expression) & e & ::= & p_1\ \texttt{\&\&}\ p_2 \mid p_1\ \texttt{<}\ p_2 \mid p_1\ \texttt{+}\ p_2 \mid p_1\ \texttt{-}\ p_2 \mid p_1\ \texttt{*}\ p_2 \mid p_1\ \texttt{[}\ p_2\ \texttt{]} \\
 & & | & p\ \texttt{.length} \mid p\ \texttt{.}id\ \texttt{(}e_1\texttt{,}\ \ldots\texttt{,}\ e_n\texttt{)} \mid p \\
(PrimaryExpression) & p & ::= & c \mid \texttt{true} \mid \texttt{false} \mid id \mid \texttt{this} \mid \texttt{new int[}e\texttt{]} \mid \texttt{new}\ id\texttt{()} \mid \texttt{!}e \mid \texttt{(}e\texttt{)} \\
(IntegerLiteral) & c & ::= & \langle \text{INTEGER\_LITERAL} \rangle \\
(Identifier) & id & ::= & \langle \text{IDENTIFIER} \rangle
\end{array}
$$

pas d'héritage

Mini-Java : syntaxe

$$
\begin{array}{rrcl}
(Goal) & g & ::= & mc\ d_1\ \ldots\ d_n \\
(MainClass) & mc & ::= & \texttt{class}\ id\ \{\ \texttt{public static void main (String [] }\ id^S\texttt{)\{} \\
 & & & \quad t_1\ id_1;\ \ldots;\ t_r\ id_r;\ s_1 \ldots\ s_q\}\} \\
(TypeDeclaration) & d & ::= & \texttt{class}\ id\ \{\ t_1\ id_1;\ \ldots;\ t_f\ id_f;\ m_1\ \ldots\ m_k\ \} \\
 & & | & \texttt{class}\ id\ \texttt{extends}\ id^P\ \{\ t_1\ id_1;\ \ldots;\ t_f\ id_f;\ m_1\ \ldots\ m_k\ \} \\
(MethodDeclaration) & m & ::= & \texttt{public}\ t\ id^M\ (t_1^F\ id_1^F,\ \ldots,\ t_n^F\ id_n^F)\ \{ \\
 & & & \quad t_1\ id_1;\ \ldots;\ t_r\ id_r;\ s_1\ \ldots\ s_q\ \texttt{return}\ e;\ \} \\
(Type) & t & ::= & \texttt{int[]} \mid \texttt{boolean} \mid \texttt{int} \mid id \\
(Statement) & s & ::= & \{\ s_1\ \ldots\ s_q\ \} \mid id\ \texttt{=}\ e; \mid id\ \texttt{[}\ e_1\ \texttt{]}\ \texttt{=}\ e_2; \\
 & & | & \texttt{if (}\ e\ \texttt{)}\ s_1\ \texttt{else}\ s_2 \mid \texttt{while (}\ e\ \texttt{)}\ s \mid \texttt{System.out.println(}\ e\ \texttt{);} \\
(Expression) & e & ::= & p_1\ \texttt{\&\&}\ p_2 \mid p_1\ \texttt{<}\ p_2 \mid p_1\ \texttt{+}\ p_2 \mid p_1\ \texttt{-}\ p_2 \mid p_1\ \texttt{*}\ p_2 \mid p_1\ \texttt{[}\ p_2\ \texttt{]} \\
 & & | & p\ \texttt{.length} \mid p\ .id\ (e_1,\ \ldots,\ e_n) \mid p \\
(PrimaryExpression) & p & ::= & c \mid \texttt{true} \mid \texttt{false} \mid id \mid \texttt{this} \mid \texttt{new int[}e\texttt{]} \mid \texttt{new}\ id\texttt{()} \mid \texttt{!}e \mid \texttt{(}e\texttt{)} \\
(IntegerLiteral) & c & ::= & \langle \text{INTEGER\_LITERAL} \rangle \\
(Identifier) & id & ::= & \langle \text{IDENTIFIER} \rangle
\end{array}
$$

pas d'héritage

Exercice

- Lister les restrictions du langage MiniJava par rapport à Java

Exemple

```
class Factorial{
    public static void main(String[] a){
        System.out.println(new Fac().ComputeFac(10));
    }
}

class Fac {
    public int ComputeFac(int num){
        int num_aux ;
        if (num < 1)
            num_aux = 1 ;
        else
            num_aux = num * (this.ComputeFac(num-1)) ;
        return num_aux ;
    }
}
```

Exercice

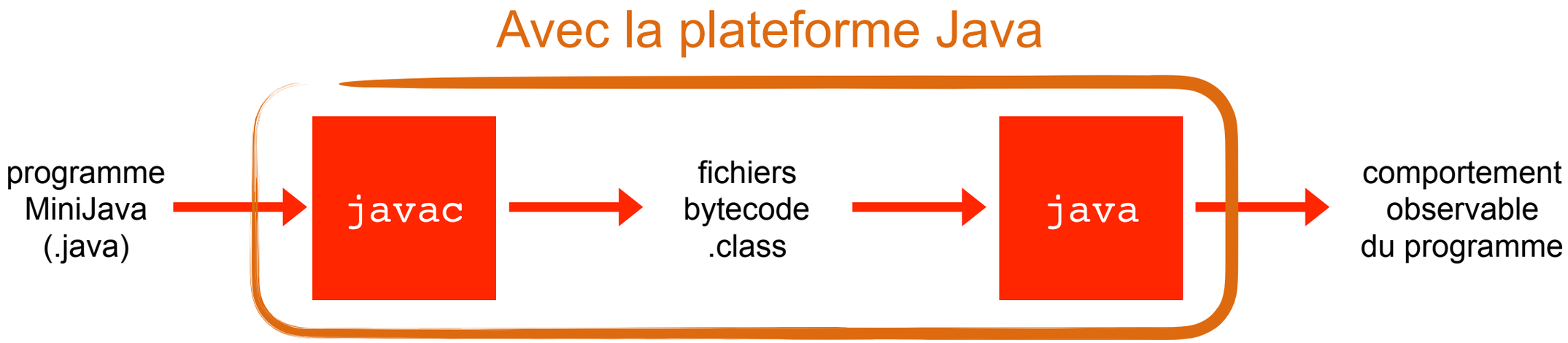
- Écrire un programme MiniJava contenant une classe `Point`.
- Chaque instance de la classe `Point` aura une abscisse et une ordonnée entière.
- Une méthode d'instance `equals()` permettra de tester l'égalité entre points.
- La procédure principale testera tout cela sur un petit exemple.

Exercice

- Écrire un programme MiniJava gérant une liste simplement chaînée circulaire

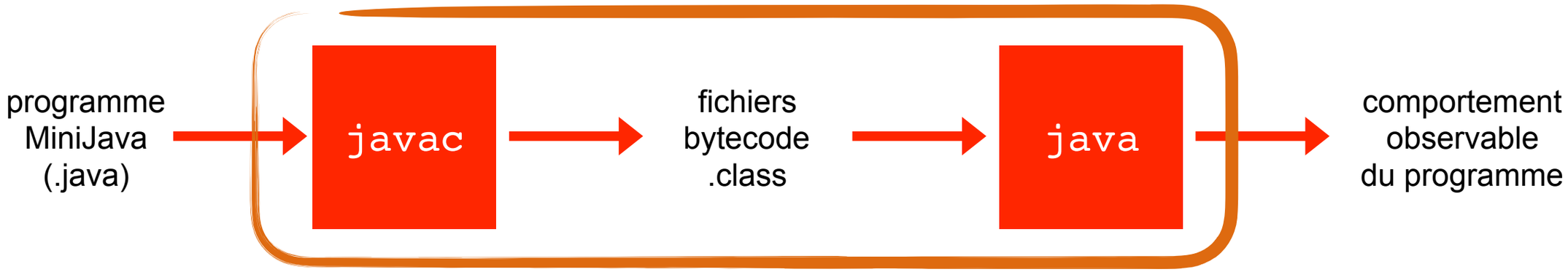
Architecture globale du projet AST

Architecture globale du projet AST



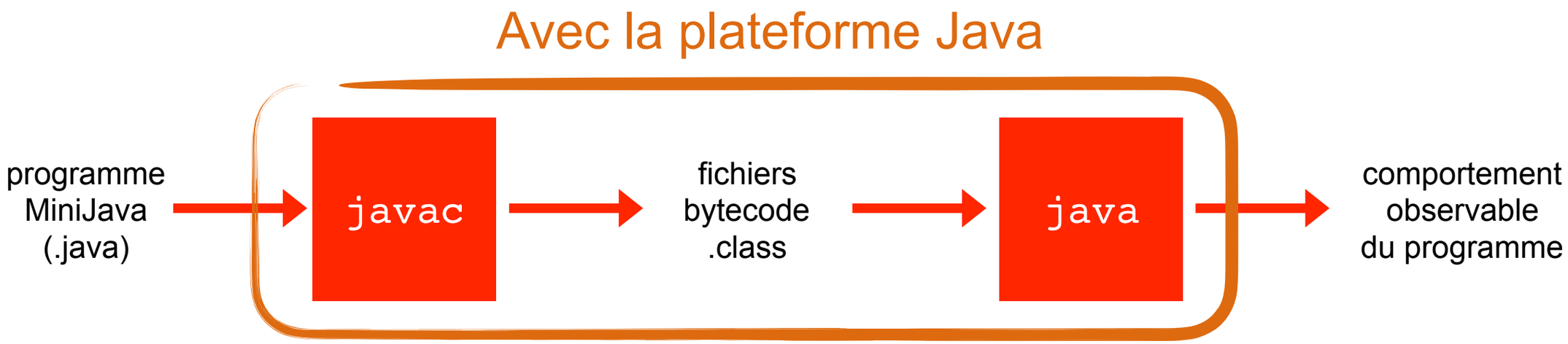
Architecture globale du projet AST

Avec la plateforme Java



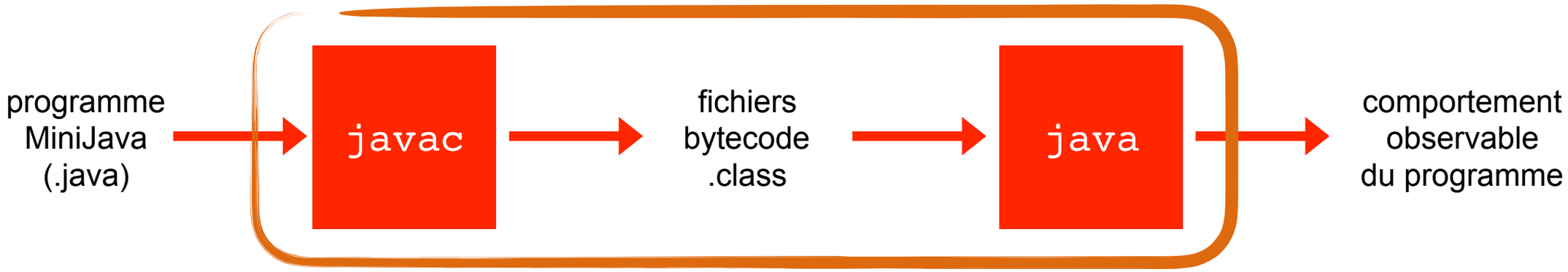
Interprétation directe

Architecture globale du projet AST

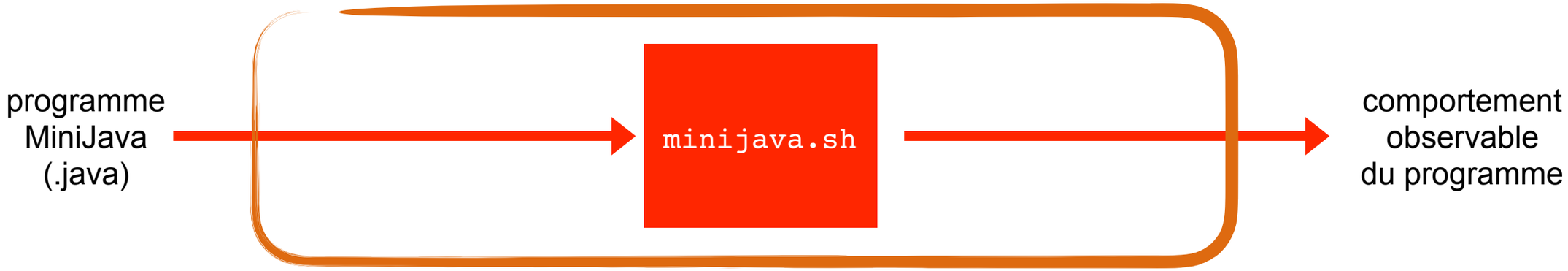


Architecture globale du projet AST

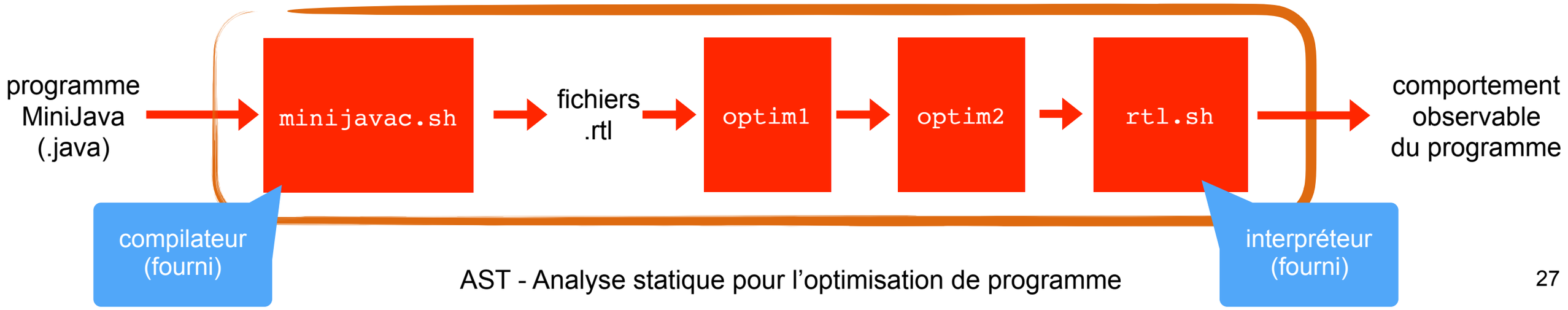
Avec la plateforme Java



Interprétation directe



Compilation, optimisation et interprétation



Le langage RTL

- Inspiré des représentations intermédiaires des compilateurs (LLVM, GCC)
- … mais simplifié pour des raisons pédagogiques

RTL

- Un programme est une liste de fonctions

Avec toujours une fonction *Main*

- Chaque fonction contient
  - un nom
  - une liste de paramètres
  - une liste de blocs

Avec toujours une block *entry*

- Chaque bloc contient
  - un label
  - une liste d’instructions
  - une instruction de sortie

```
func Main(a)
  entry:
    t.0 = call f(10)
    PrintInt(t.0)
    ret
```

```
func f(num)
  entry:
    t.0 = Add(num 1)
    t.1 = Add(t.0 1)
    t.2 = Add(t.1 1)
    ret t.2
```

Instructions

(Instr) *i* ::=	*id* **=** *op*	(Assign)
|	**PrintInt**(*op*)	(BuiltIn)
|	*id* **= Alloc**(*op*$_1$)	(BuiltIn)
|	*id* **= Add**(*op*$_1$ *op*$_2$)	(BuiltIn)
|	*id* **= Sub**(*op*$_1$ *op*$_2$)	(BuiltIn)
|	*id* **= Mul**(*op*$_1$ *op*$_2$)	(BuiltIn)
|	*id* **= Lt**(*op*$_1$ *op*$_2$)	(BuiltIn)
|	*id* **= And**(*op*$_1$ *op*$_2$)	(BuiltIn)
|	*id* **= call** *F*(*op*$_1$ … *op*$_n$)	(Call)
|	*id*$_2$ **= [** *id*$_1$ **+/-** *i* **]**	(MemRead)
|	**[** *id* **+/-** *i* **]** = *op*	(MemWrite)
(EndInstr) *ei* ::=	**return** *op*	(Return)
|	**return**	(Return)
|	**goto** *label*	(Goto)
|	**if** *op* **goto** *label*$_1$ **else** *label*$_2$	(Branch)
(Operand) *op* ::=	*id*	(Ident)
|	*i*	(LitInt)

Alloue un bloc mémoire de taille *op*$_1$, initialisé avec des zéros, et renvoie son adresse

lit/écrit à l'adresse contenue dans *id*$_1$, plus ou moins l'entier i

Exemple

```
class Factorial{
  public static void main(String[] a){
    System.out.println(new Fac().ComputeFac(10));
  }
}

class Fac {
  public int ComputeFac(int num){
    int num_aux ;
      if (num < 1)
        num_aux = 1 ;
      else
        num_aux = num * (this.ComputeFac(num-1));
      return num_aux ;
  }
}
```

minijavac.sh

```
func Main(a)
  entry:
    t.1 = Alloc(1)
    t.0 = call Fac.ComputeFac(t.1 10)
    PrintInt(t.0)
    ret

func Fac.ComputeFac(this num)
  entry:
    t.0 = Lt(num 1)
    if t.0 goto if0_then else if0_else
if0_then:
  num_aux = 1
  goto if0_end
if0_else:
  t.2 = Sub(num 1)
  t.1 = call Fac.ComputeFac(this t.2)
  num_aux = Mul(num t.1)
  goto if0_end
if0_end:
  ret num_aux
```

Exercice

- Écrire un programme RTL représentant la version compilée du programme MiniJava suivant

```
class Simple {
  public static void main(String[] a) {
  }
}
```

```
class T {

  int s;
  int[] t;

  public int init(int size) {
    s = size;
    t = new int[size];
    return 0;
  }

  public int size() {
    return t.length;
  }

}
```