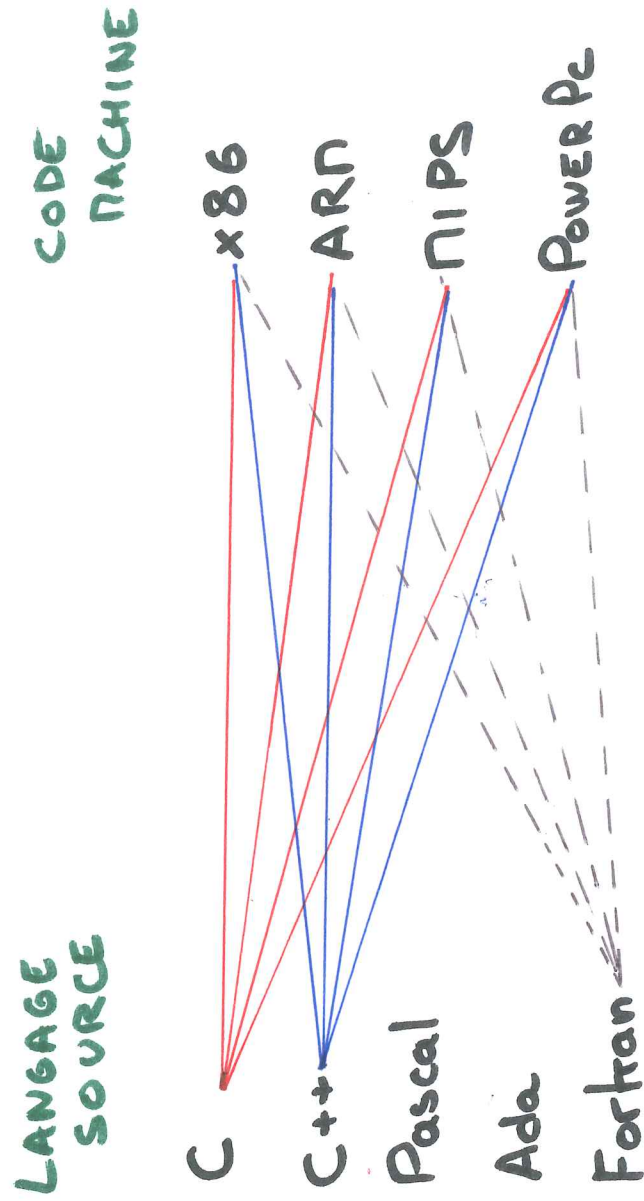
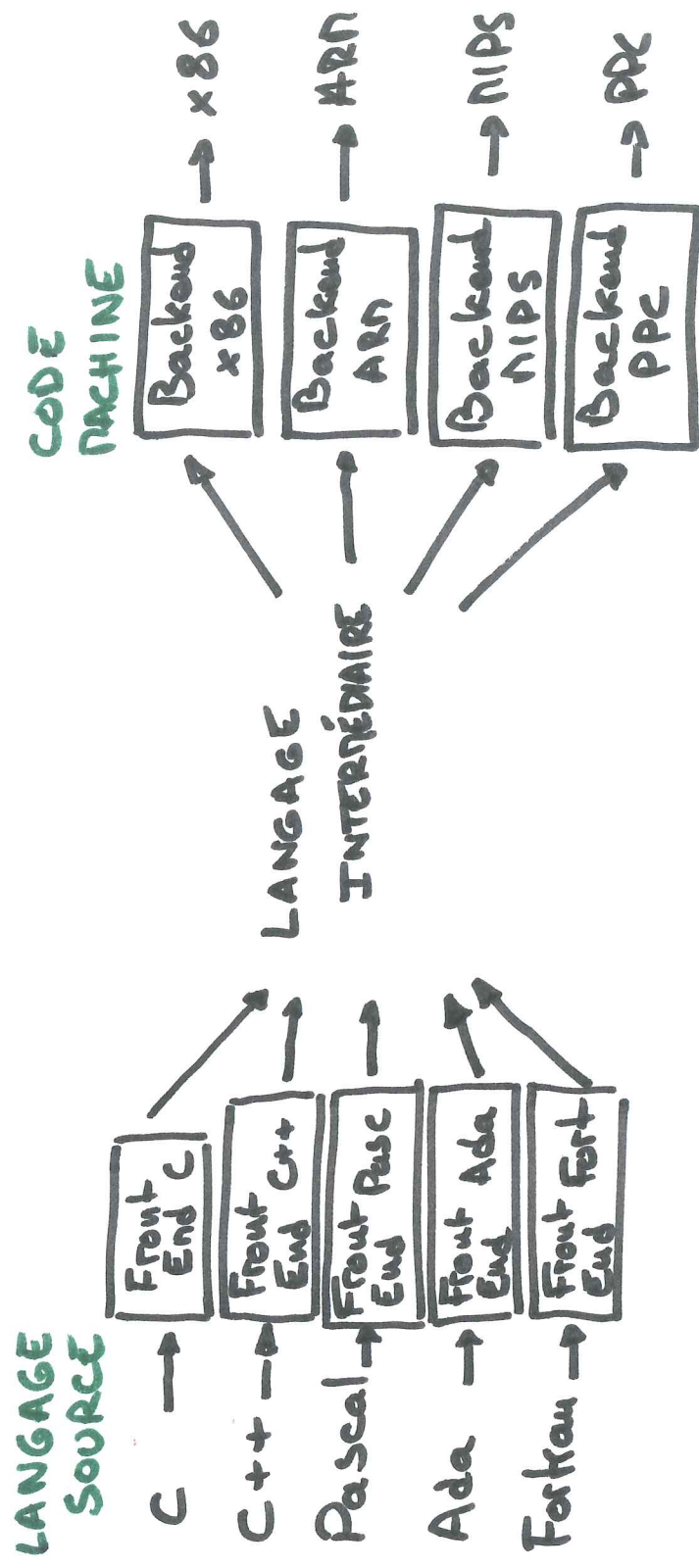


# Compiler



$$5 \times 4 = 20 \text{ compilers ?}$$

# Compilateur: importance du langage intermédiaire



- ARCHITECTURE DÉCOUPLÉE + MODULAIRE
- Exemple: GCC, des transformations en plusieurs IR (GIMPLE, RTL, ...)

# Schéma Général du Compilateur

ANALYSE SÉMANTIQUE

Fichier Source

LEX

Tokens

PARSE

AST

BINDING

AST décoré

AST décoré

Environnements

AST décoré + escapes

ESCAPE COMPUTING

AST décoré + types ok

TYPE CHECK

TRADUCTION IR

Langage Intermédiaire (IR)

PASSAGE EN FORME CANONIQUE

IR

IR Canonique

TRANSFORMATIONS EN LANGAGE INTERMÉDIAIRE

ANALYSE DE VIVACITÉ

DATA FLOW ANALYSIS

Assembleur registers  $\infty$

SÉLECTION D'INSTRUCTIONS

Graphes d'interférence

Allocation Register

Assembleur

GÉNÉRATION DE CODE

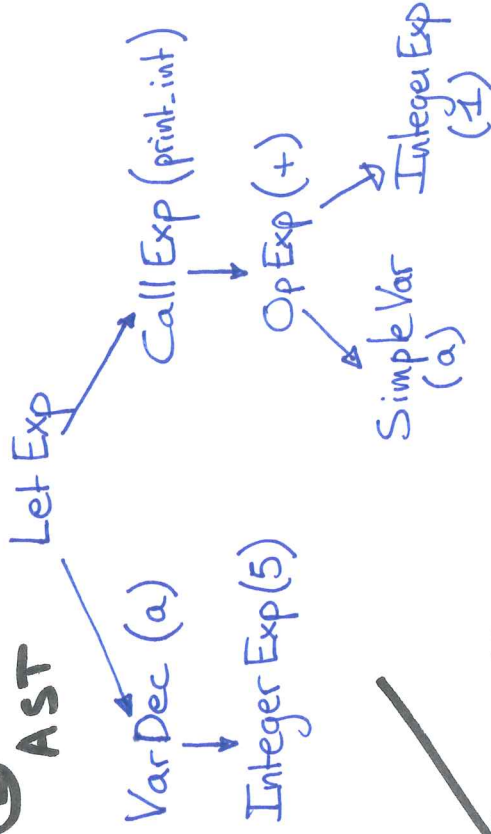
# EXAMPLE TIGER (1/2)

## ① TIGER

LEX + PARSE

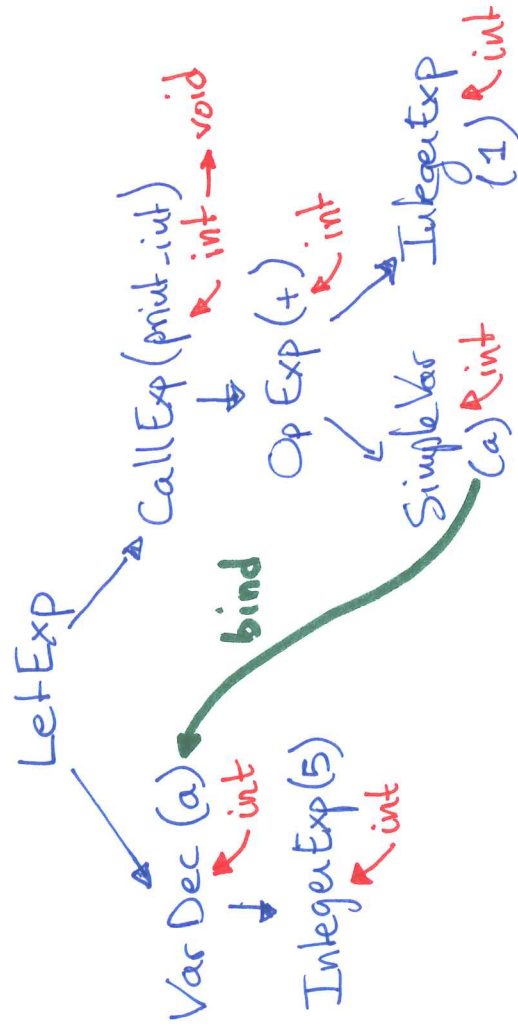
let var a := 5 in  
print-int (a+1) end

## ② AST

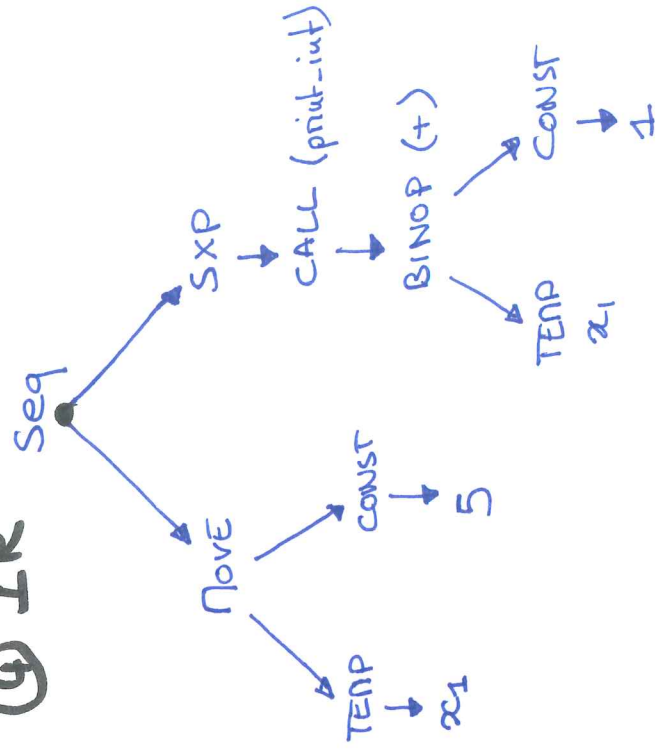


BINDING  
+  
TYPE CHECKING

## ③ Decorated AST



## ④ IR



TRANSLATION  
IR

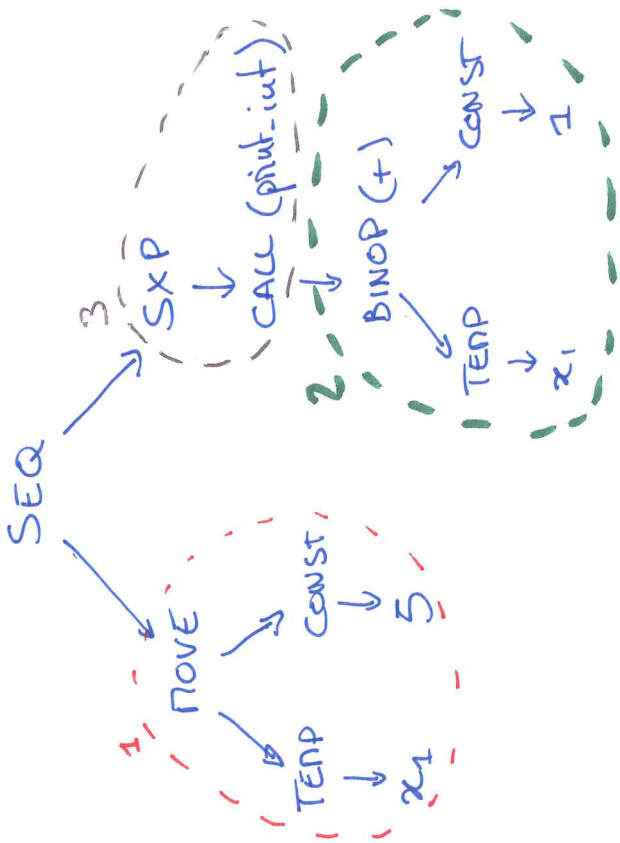


EXEMPLE TIGER (2/2)

Génération de Code



④ IR



⑤ Assembleur, registers ∞

- 1. li \$x1, 5
- 2. add \$x2, \$x1, 1
- 3. move \$a0, \$x2  
jal print-int

↓ Allocation Registers

⑥ Code final

- li \$t4, 5
- add \$a0, \$t4, 1
- jal print-int

(registers x2 et a0 fusionnés)

# Visiteur d'Arbre

- De nombreuses opérations sur des arbres (AST, IR)
  - Binding, Type Check, Escapes, Traduction, etc...
- On souhaite écrire une seule fois le code pour
  - parcourir l'arbre
  - décider du traitement en fonction du type de noeud
- DESIGN PATTERN, proposés par C. Alexander dans les années 70.

## Visiteur d'Arbre en Java

```
class VarDec  
implements Visitable <Visitor> {  
  
    void acceptVisitor (Visitor v) {  
        v.enter (this);  
    }  
}
```

Dans chaque classe  
Noeud

```
class ASTPrinter  
implements Visitor {  
  
    void enter (VarDec v) {  
        System.out.println ("VarDec (" + v.name + ")");  
    }  
  
    void enter (IntegerExp i) {  
        System.out.println ("Int (" + i.value + ")");  
    }  
}
```

Pour chaque opération sur les Noeuds, un Visiteur

Exemple

```
Node n = new VarDec (...);  
n.acceptVisitor (new ASTPrinter ()); ] DOUBLE DISPATCH  
    → "VarDec (a)"
```

## Visiteur en java

### Opérations

Afficher noeud de l'AST

→ Visiteur ASTPrinter

Parcourir un arbre à partir du noeud

→ Visiteur ASTTraverse

:

### Noeuds

VarDec

FuncDec

IntegerExp

OpExp

:

Chaque combineur (Opération, Noeud) appelle une méthode spécifique.

Java, dispatch dynamique

```
class A extends N {  
    void do() { ... {  
        N n = new A();
```

```
    }  
    N.do();
```

```
class B extends N {  
    void do() { ... {  
        N m = new B();
```

```
    }  
    m.do();
```

⚠ Dispatch dynamique

nous permet de  
choisir la méthode  
selon un seul  
objet...