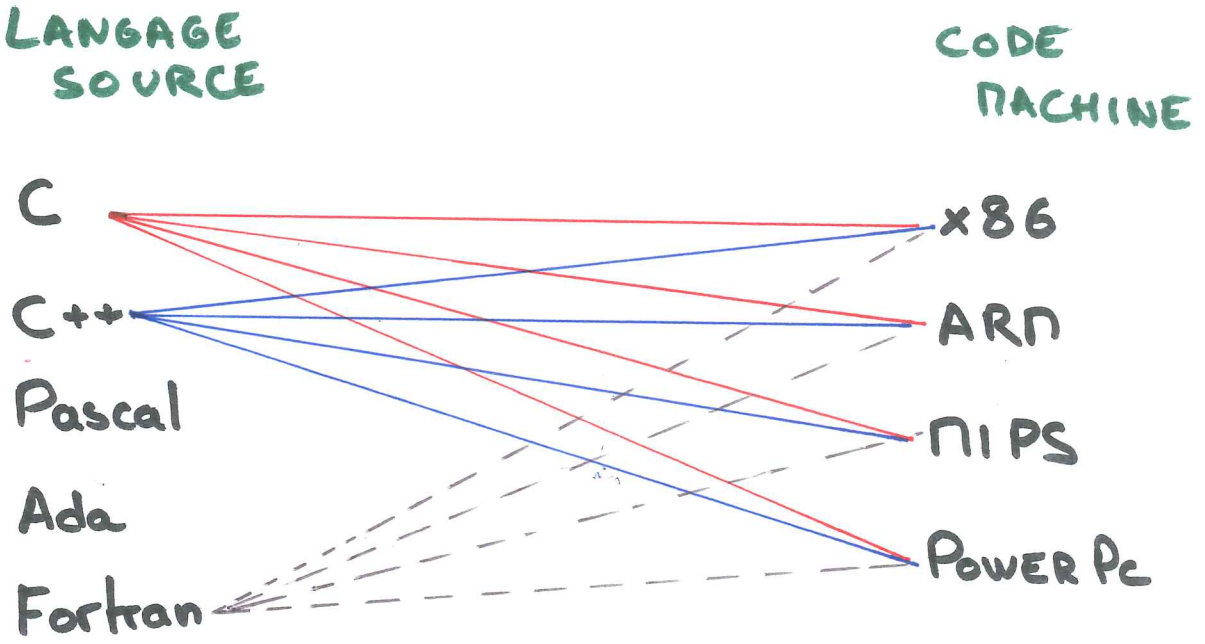
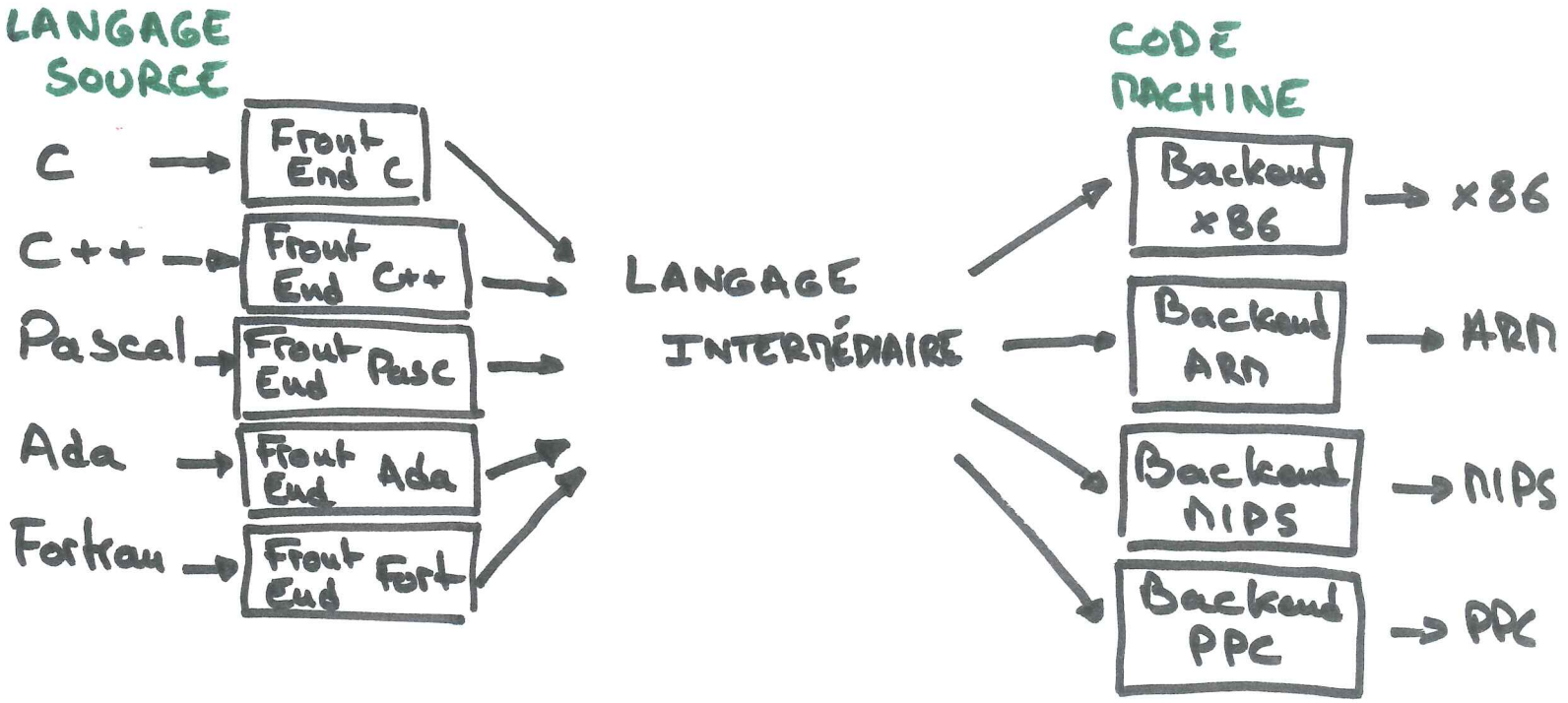
Compilateur



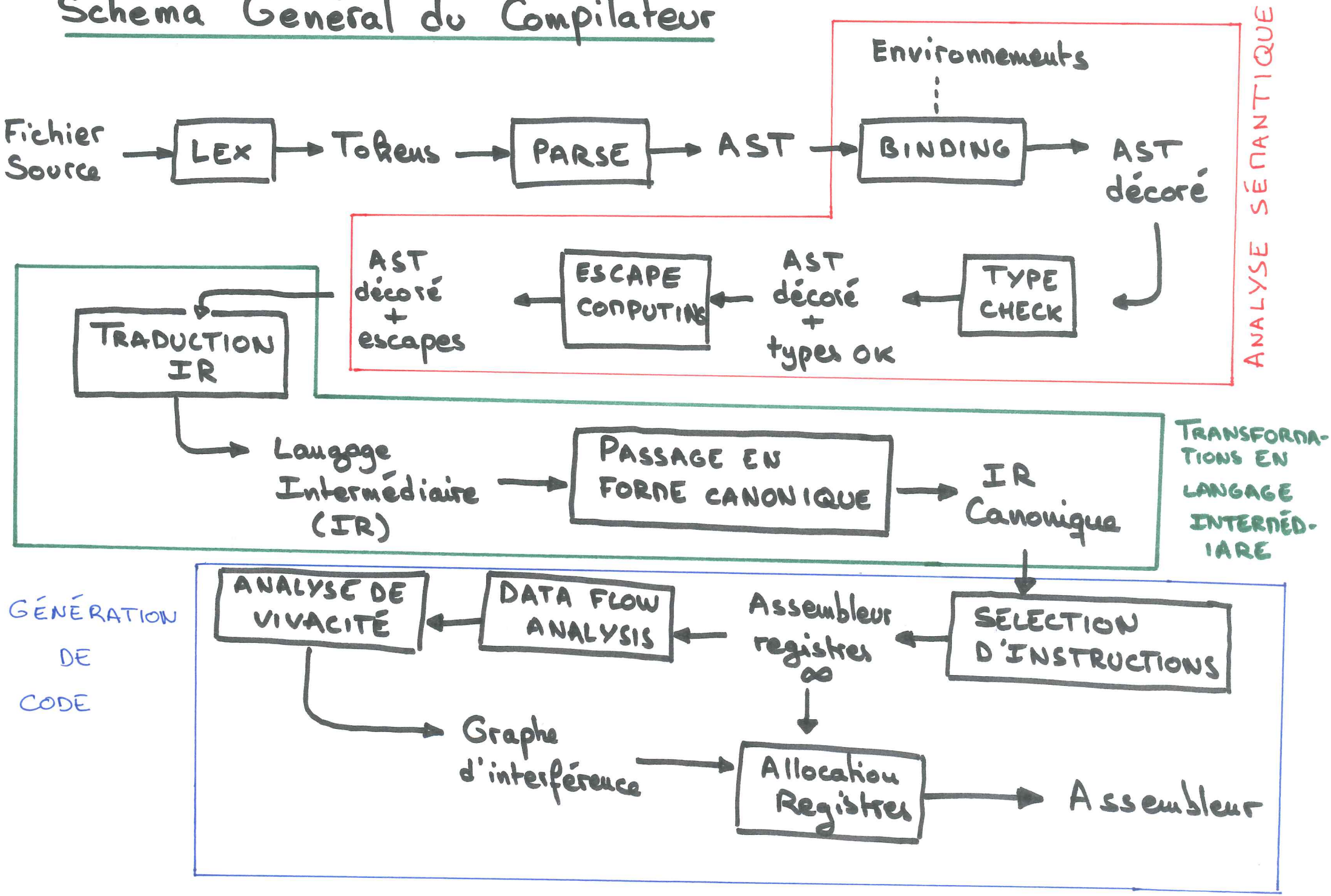
5 x 4 = 20 compilateurs ?

Compilateur : importance du langage intermédiaire



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

Schéma Général du Compilateur



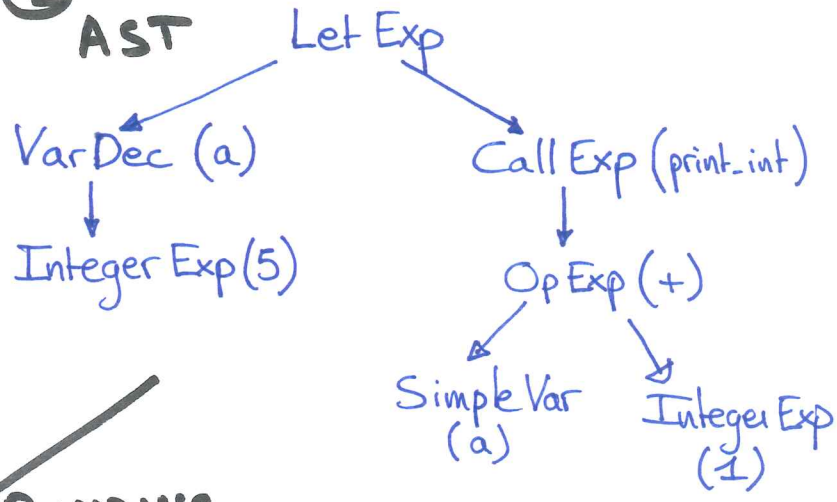
EXEMPLE TIGER (1/2)

① TIGER

```
let var a := 5 in
    print_int (a+1) end
```

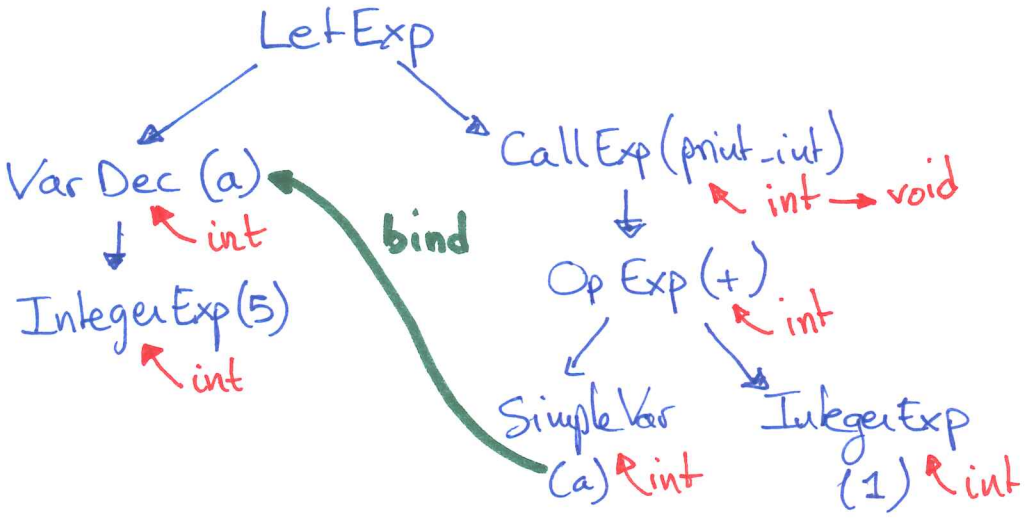
LEX + PARSE →

② AST



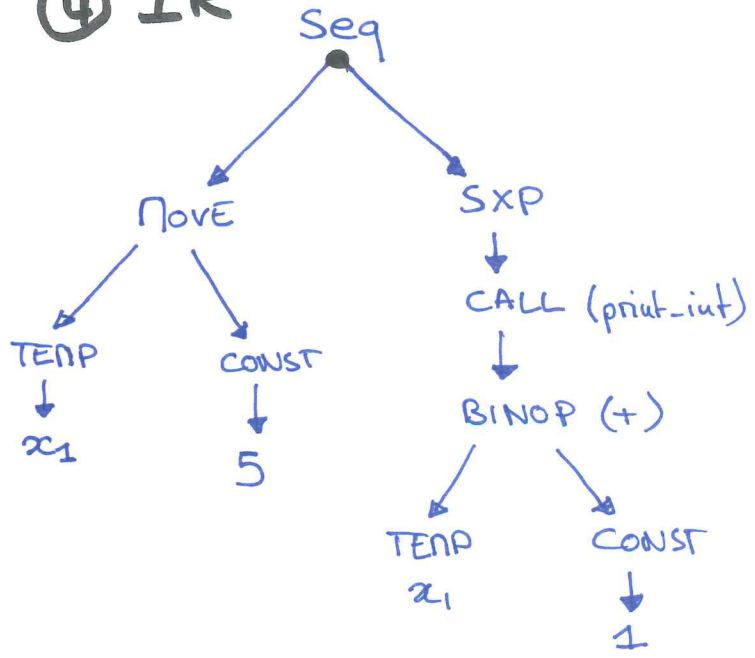
BINDING + TYPE CHECKING ↓

③ Decorated AST



TRADUCTION IR →

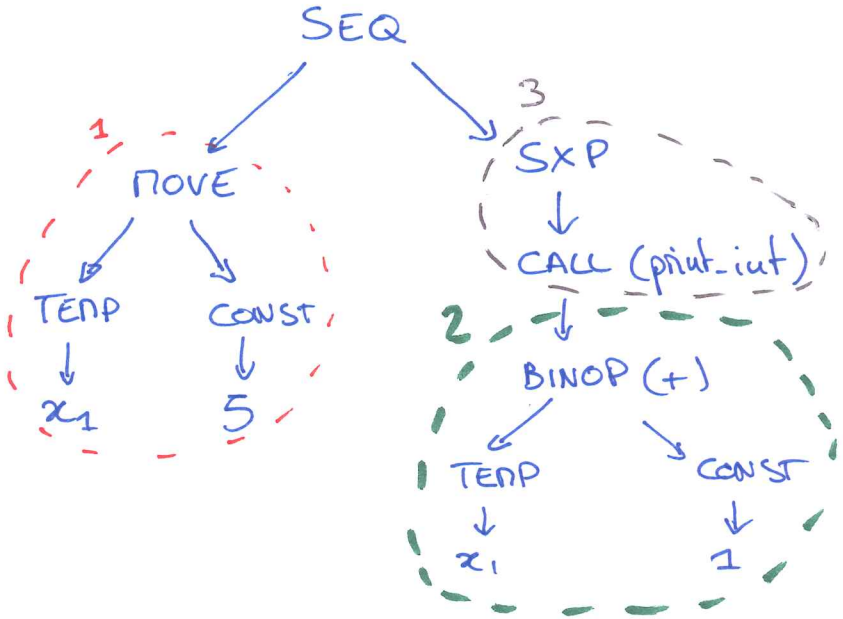
④ IR



Exemple Tiger (2/2)

Génération de Code →

④ IR



⑤ Assembleur, registres ∞

1. li $ x_1 , 5
2. add $ x_2, $ x_1, 1
3. move $ a_0 , $ x_2
   jal print_int

↓ Allocation Registres

⑥ Code final

```
li $t4, 5
add $a0 , $t4, 1
jal print_int
```

(registres x_2 et a_0 fusionnés)

Visiteur d'Arbre

- De nombreuses opérations sur des arbres (AST, IR)
  - → Binding, Type Check, Escapes, Traduction, etc...

- On souhaite écrire <u>une seule fois</u> 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 AST Traverse

⋮

Noeuds

VarDec
FuncDec
IntegerExp
OpExp
⋮

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

Java, dispatch dynamique

```
class A extends N {
    void do() {...}
}

class B extends N {
    void do() {...}
}
```

```
N n = new A();
n.do();
N m = new B();
m.do();
```

⚠ Dispatch dynamique nous permet de choisir la méthode selon un seul objet...