

# Compilation de Mini-Lustre vers LLVM

Lemaire & Patault

Université Paris-Saclay

January 16, 2022

## 1 Introduction

- Présentation LLVM
- Utilisation LLVM

## 2 Travail Réalisé

- Dans le dur
- Benchmarks
- Démonstration

# Présentation Générale

- LLVM := Low Level Virtual Machine
- LLVM IR := langage intermédiaire que l'on utilise
- Interface C++ et bindings OCaml

# Pourquoi compiler vers LLVM ?

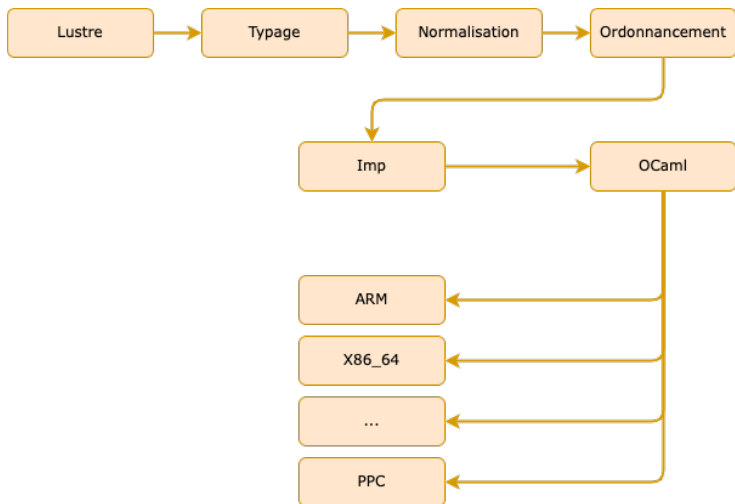
## ■ “Assembleur de haut niveau”

- Typage
- Pointeurs
- Vecteurs
- Tableaux
- Structures
- Fonctions
- ...

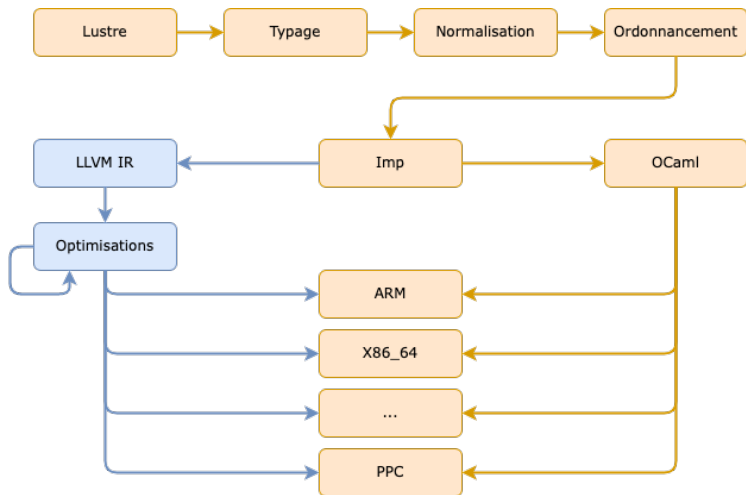
## ■ Bindings simples d'utilisation

## ■ Compilateur optimisant vers toutes les architectures

# Schéma de compilation simplifié



# Schéma de compilation simplifié



# En pratique

## ■ Interface C++ et bindings OCaml

```
let main = Llvm.declare_function "main" i32 in
let o = Llvm.build_alloca llvm_typ "o" llvm_builder in
let one = Llvm.const_int i32_ty 1 in
let store = Llvm.build_store o one llvm_builder in
...
```

## ■ Génération du code LLVM IR (fichiers .ll)

```
declare i32 @main() #0 {
  entry:
    %o = alloca i32, align 4
    store i32 1, i32* %o, align 4
    ...
}
```

## 1 Introduction

- Présentation LLVM
- Utilisation LLVM

## 2 Travail Réalisé

- Dans le dur
- Benchmarks
- Démonstration



# Exemple de transformation

```
node f(x:int)
returns (o:int);
let
  o = 1;
tel
```

 $\mapsto$ 

```
define i64 @f_1_step(i32 x) {
  %o = alloca i32, align 4
  store i32 1, i32* %o, align 4
  %0 = alloca %ret_f_1_step, align 8
  %1 = getelementptr inbounds %ret_f_1_step,
    %ret_f_1_step* %0, i32 0, i32 0
  %2 = load i32, i32* %o, align 4
  store i32 %2, i32* %1, align 4
  %3 = bitcast %ret_f_1_step* %0 to i64*
  %4 = load i64, i64* %3, align 4
  ret i64 %4
}
```

# Performances

	simple.mls ( $10^7$ )	simple.mls ( $10^8$ )	todo.mls
LLVM	13.84s	140.70s	todo
OCaml	16.29s	185.49s	todo

**Table:** Table des performances comparées en fonction du langage cible

# Démonstration

En direct