Teaching MLIR Concepts to Undergraduate Students

Sasha Lopoukhine, Mathieu Fehr, Tobias Grosser - University of Cambridge, University of Edinburgh

Based on Real-World Tools

Core Compilation Concepts

Parsing

Type Checking

Lowering

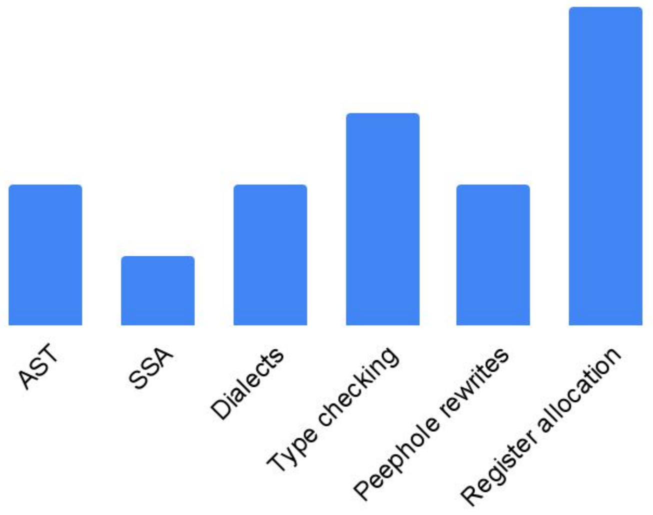
More Core Compilation Concepts

Parsing
&
Error Reporting

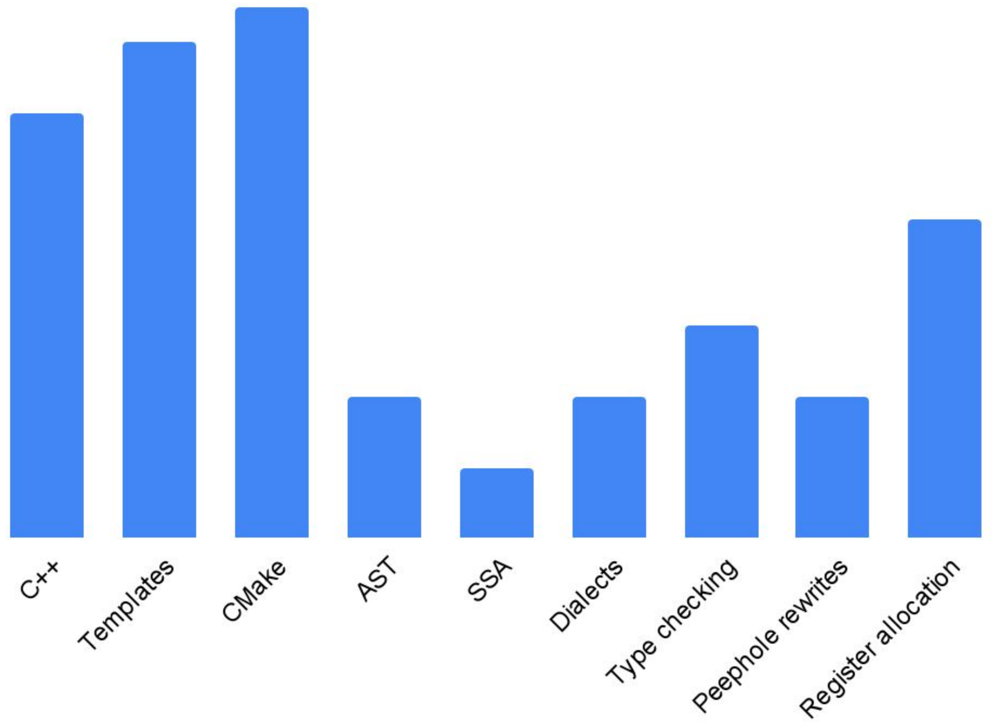
Type Checking
&
Analysis

Lowering
&
Optimisation

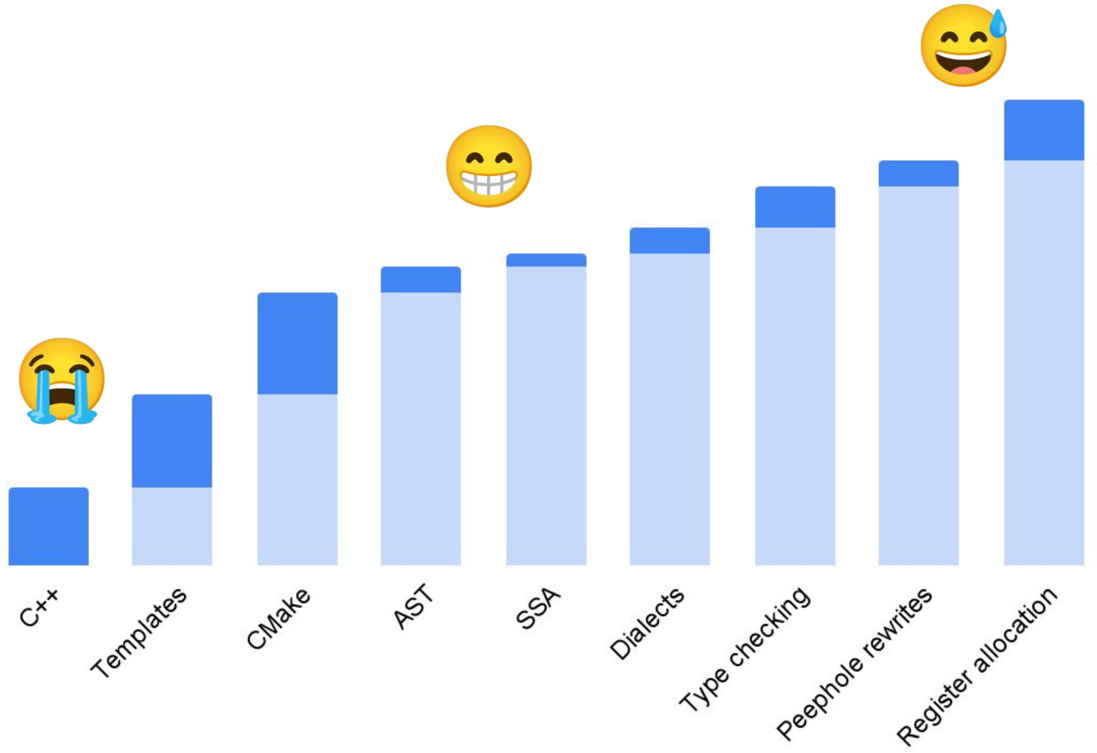
Difficulty of Concepts



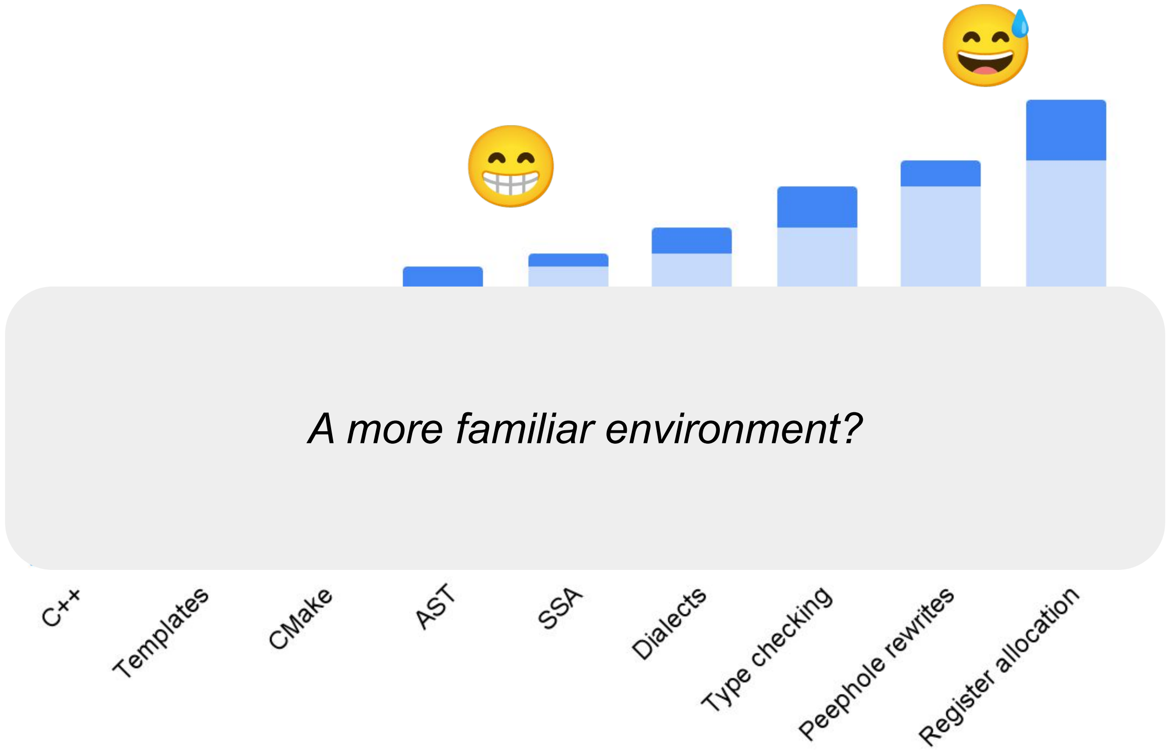
Difficulty of MLIR Infrastructure



MLIR Has a Steep Learning Curve



Skipping Directly to Compilation Concepts?



A more familiar environment?

https://xdsl.dev/

xDSL: MLIR-Style Compiler Development

MLIR Concepts

MLIR IR

Python + types

Designed for exploration

Small footprint

Based on Real-World Tools

(ChocoPy)

(xDSL)

https://chocopy.org/

ChocoPy: A Typed Subset of Python

```
def is_zero(items: [int],
            idx: int) -> bool:
  val: int = 0
  val = items[idx]
  return val == 0

mylist: [int] = None
mylist = [1, 0, 1]
print(is_zero(mylist, 1))
```

```
# Designed for teaching

# Detailed language reference

# Familiar Syntax

# Static typing
```

https://riscv.org/

RISC-V: Open-Source ISA

```
memcpy_general:
    add     a5,a1,a2
    beq     a1,a5,.L2
    add     a2,a0,a2
    mv      a5,a0
.L3:
    addi    a1,a1,1
    addi    a5,a5,1
    lbu     a4,-1(a1)
    sb      a4,-1(a5)
    bne     a5,a2,.L3
.L2:
    ret
```

# Simple instructions

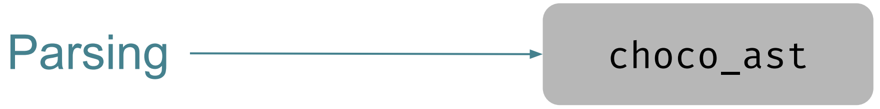
# Growing in popularity

# Emulate code with `riscemu`

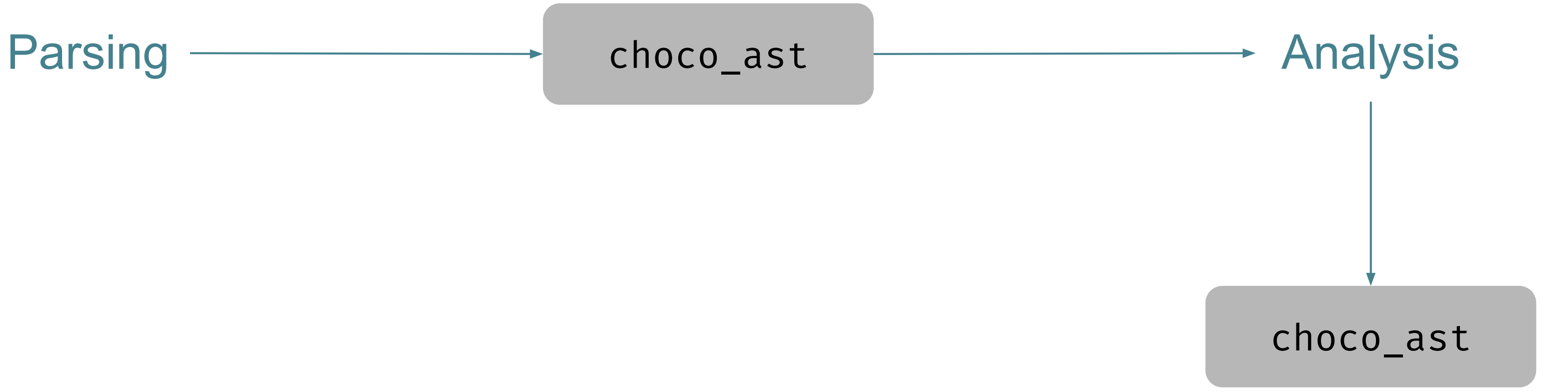
The Student’s Journey

Parsing

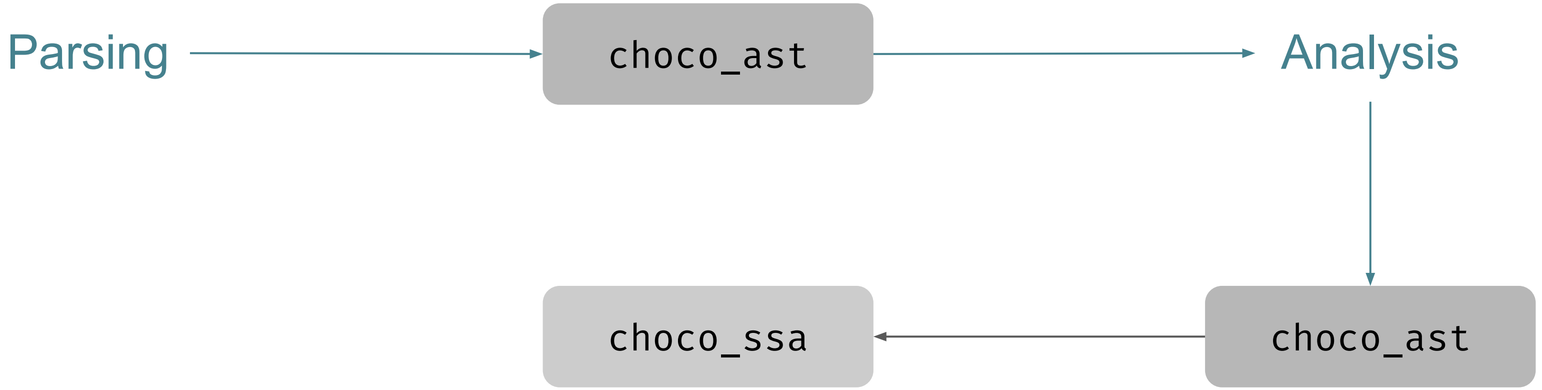
The Student’s Journey



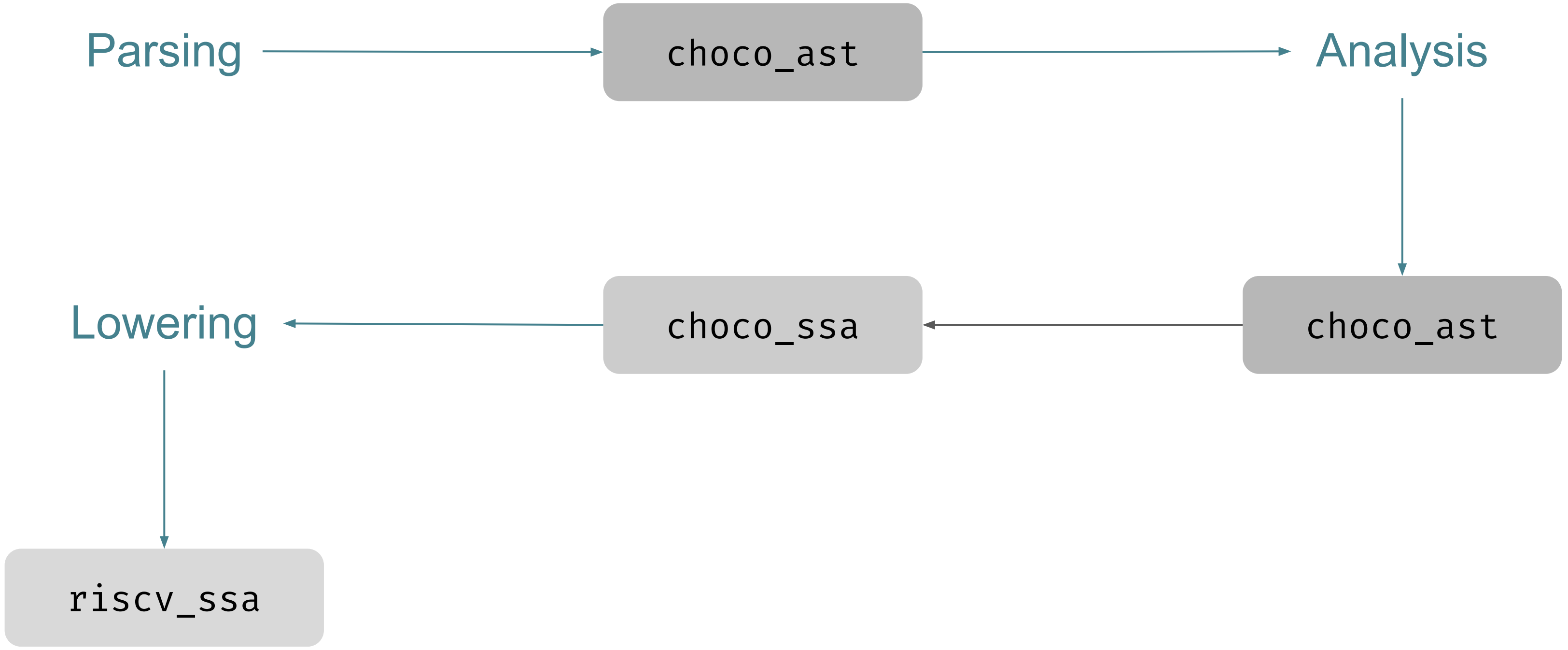
The Student’s Journey



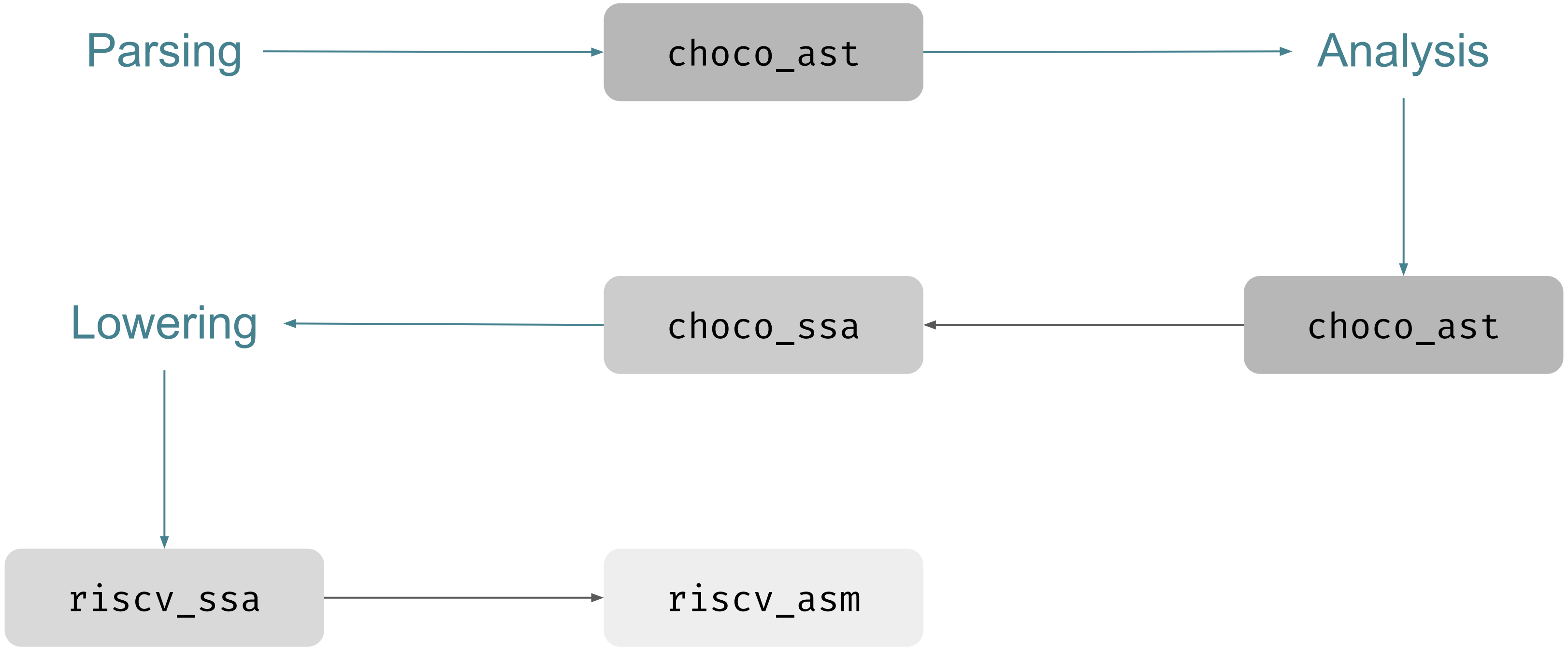
The Student's Journey



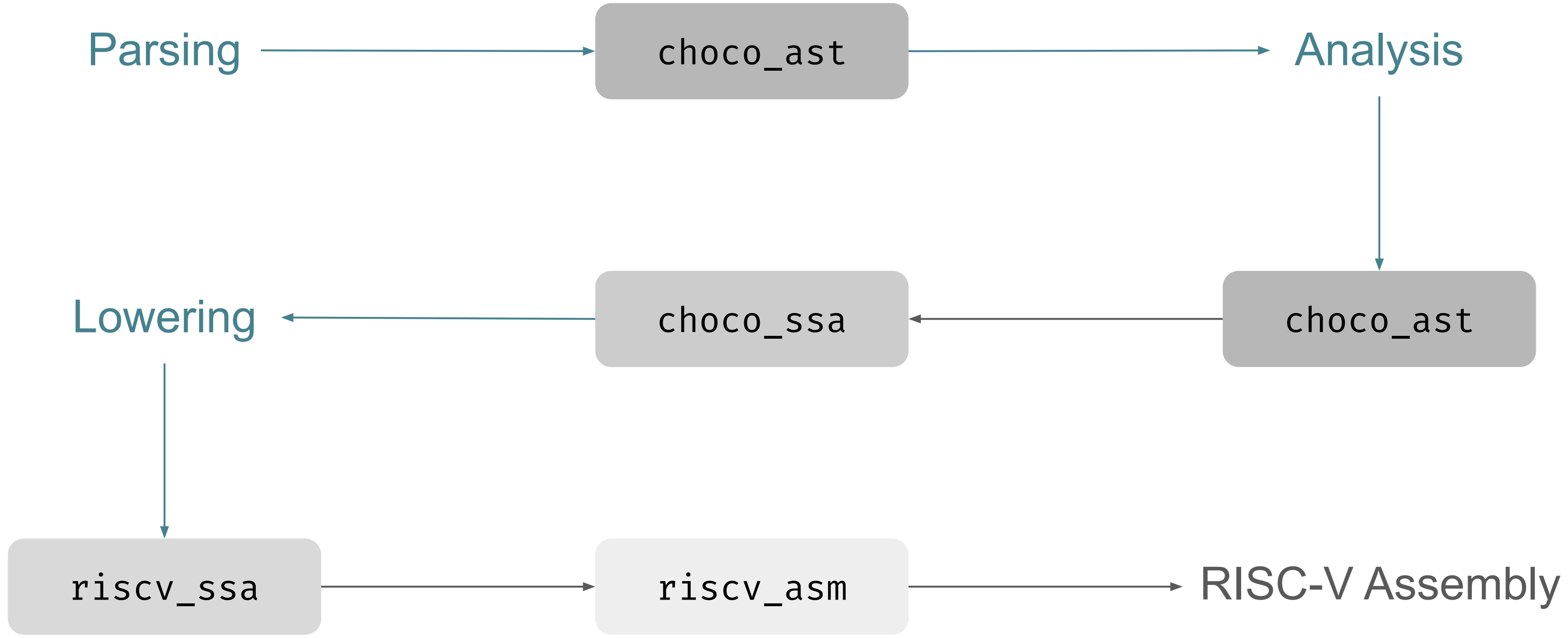
The Student’s Journey



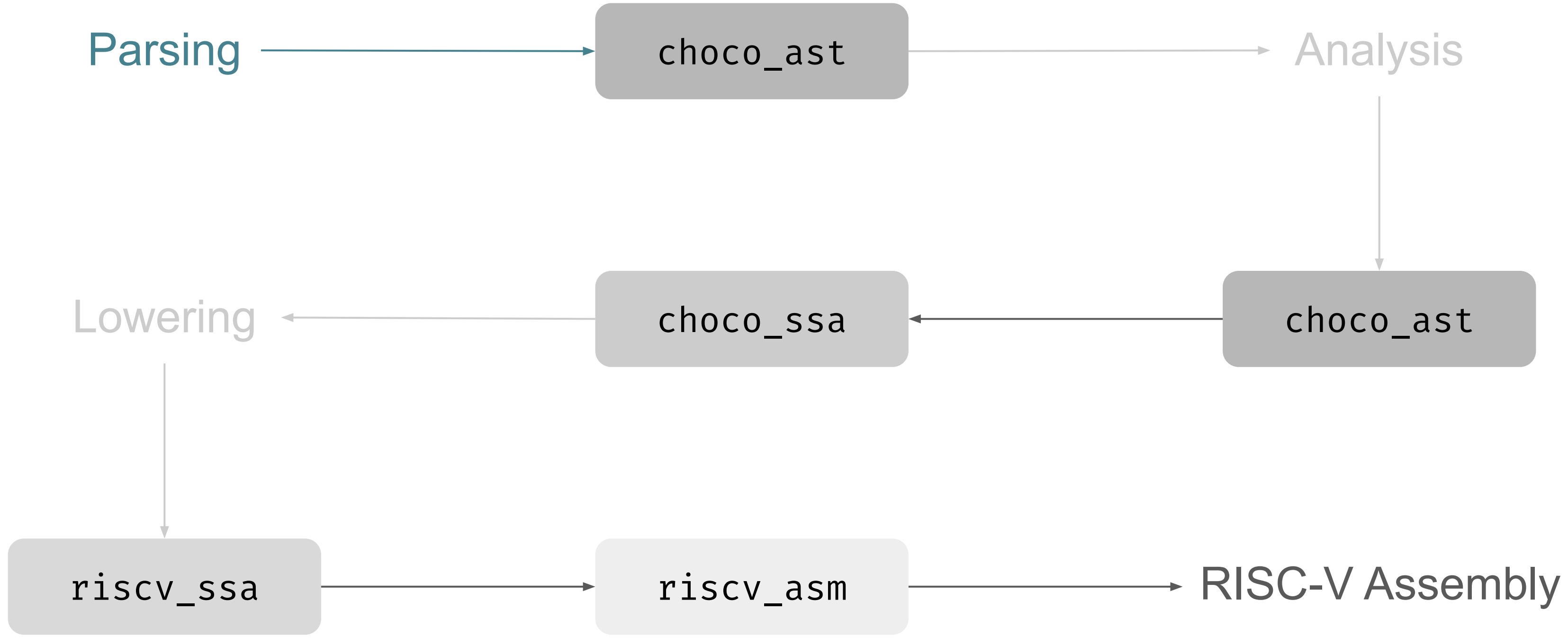
The Student’s Journey



The Student’s Journey



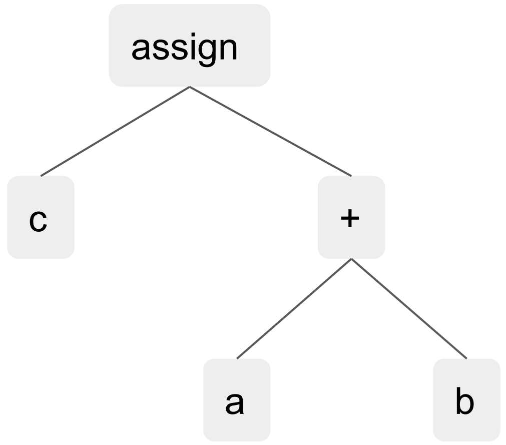
The Student’s Journey



✓ AST

Task 1.1: Parsing ChocoPy

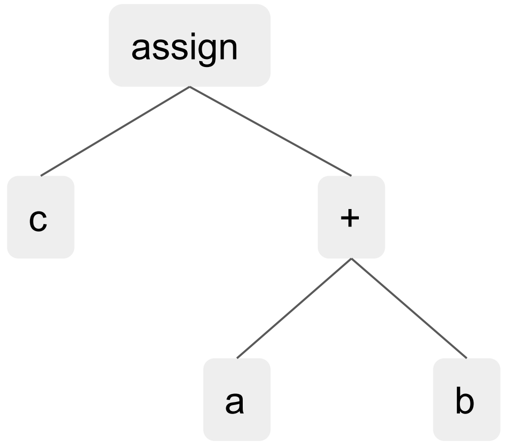
c = a + b



Task 1.1: Parsing ChocoPy into MLIR IR

✓ AST
✓ Operations + Regions

```
c = a + b
```



```
choco_ast.assign {
  choco_ast.id_expr "c"
} {
  choco_ast.binary_expr "+" {
    choco_ast.id_expr "a"
  } {
    choco_ast.id_expr "b"
  }
}
```

No SSA values !

✓ AST
✓ Operations + Regions
✓ Filecheck

Testing with FileCheck

```
// RUN: choco-opt %s | filecheck %s

c = a + b

// CHECK:      choco_ast.assign {
// CHECK-NEXT:   choco_ast.id_expr "c"
// CHECK-NEXT: } {
// CHECK-NEXT:   choco_ast.binary_expr "+" {
// CHECK-NEXT:     choco_ast.id_expr "a"
// CHECK-NEXT:   } {
// CHECK-NEXT:     choco_ast.id_expr "b"
// CHECK-NEXT:   }
// CHECK-NEXT: }
```

Specification

- ✓ AST
- ✓ Operations + Regions
- ✓ Filecheck
- ✓ Parsing Theory
- ✓ Dialect Definitions

LL(2) grammar

```
bin_expr := cexpr bin_op cexpr


bin_op := `+` | `-` | `*` | `//` | …
```

choco_ast dialect

```
class BinaryExpr(IRDLOperation):
    name = "choco_ast.binary_expr"

    op: StringAttr = prop_def(StringAttr)
    lhs: Region = region_def("single_block")
    rhs: Region = region_def("single_block")
```

Writing a Recursive Descent Parser

- ✓ AST
- ✓ Operations + Regions
- ✓ Filecheck
- ✓ Parsing Theory
- ✓ Dialect Definitions

```
def parse_bin_expr(self) -> Operation:
    expr1 = self.parse_simple_expr()
    while self.check(PLUS):
        self.match(PLUS)
        expr2 = self.parse_simple_expr()
        expr1 = choco_ast.BinaryExpr("+", expr1, expr2)
    return expr1
```

Writing a Recursive Descent Parser

- ✓ AST
- ✓ Operations + Regions
- ✓ Filecheck
- ✓ Parsing Theory
- ✓ Dialect Definitions
- ✓ Building MLIR IR

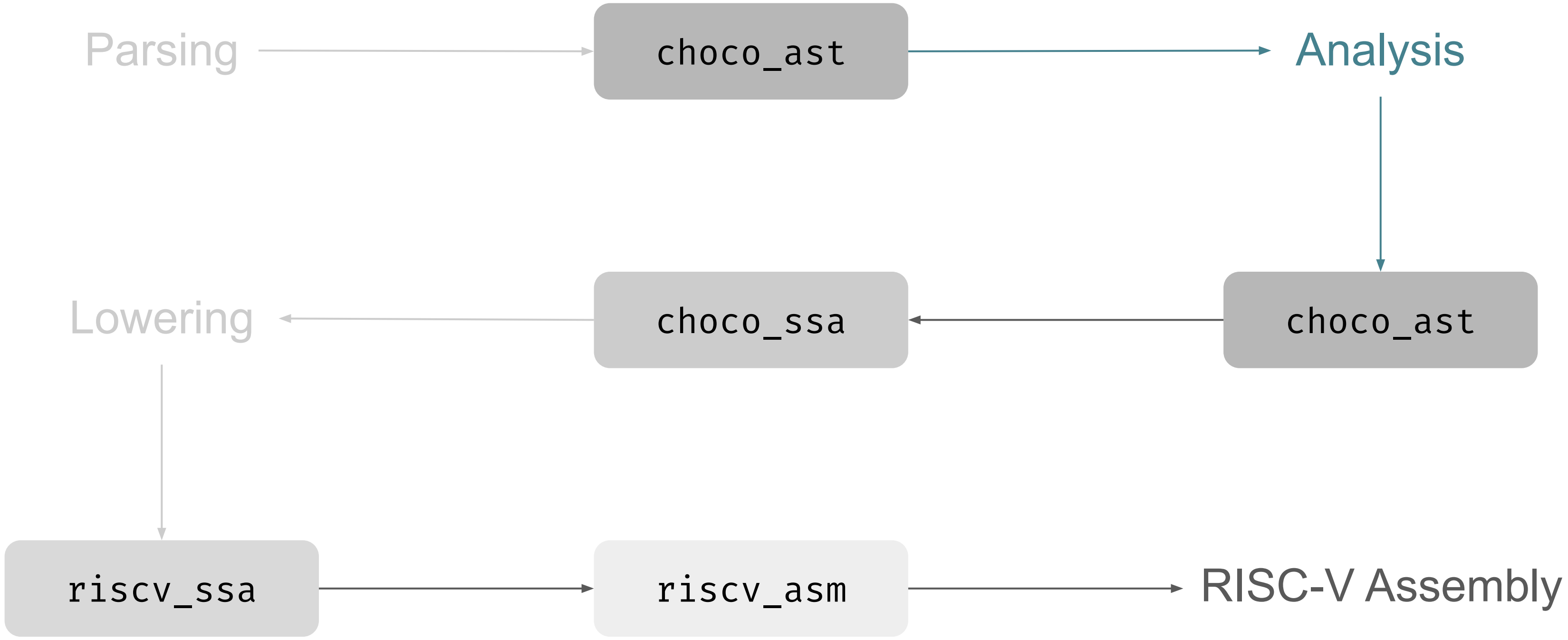
```
def parse_bin_expr(self) -> Operation:
    expr1 = self.parse_simple_expr()
    while self.check(PLUS):
        self.match(PLUS)
        expr2 = self.parse_simple_expr()
        expr1 = choco_ast.BinaryExpr("+", expr1, expr2)
    return expr1
```

Task 1.2: Error Reporting

- ✓ AST
- ✓ Operations + Regions
- ✓ Filecheck
- ✓ Parsing Theory
- ✓ Dialect Definitions
- ✓ Building MLIR IR
- ✓ Compiler frontend

```
for i [1,2]
   pass

# CHECK-NEXT: SyntaxError (line 1, column 9): token of kind TokenKind.IN not found.
# CHECK-NEXT: >>>  for i [1,2]
# CHECK-NEXT: >>>--------^
```

The Student’s Journey



Task 2.1: Type Checking

```
O, R ⊢  e : int
------------------- [NEGATE]
O, R ⊢ -e : int
```

✓ Type theory
✓ MLIR Attributes

Task 2.1: Type Checking

```
O, R ⊢  e : int
------------------- [NEGATE]
O, R ⊢ -e : int
```

```
x: int = 0
-x

// CHECK:      choco_ast.unary_expr {type = !choco_ast.int} "-" {
// CHECK-NEXT:   choco_ast.id_expr {type = !choco_ast.int} "x"
// CHECK-NEXT: }
```

- ✓ Type theory
- ✓ MLIR Attributes
- ✓ Dead code analysis

Task 2.2: "Dead Code" Analysis

Unreachable statement

```
def foo():
    return
    print("DEAD")
    return
```

Task 2.2: "Dead Code" Analysis

✓ Type theory
✓ MLIR Attributes
✓ Dead code analysis

Unreachable statement

```
def foo():
    return
    print("DEAD")
    return
```

Unused variable

```
x: int = 0
```

✓ Type theory
✓ MLIR Attributes
✓ Dead code analysis

Task 2.2: "Dead Code" Analysis

Unreachable statement

```
def foo():
    return
    print("DEAD")
    return
```

Unreachable expression

```
def foo():
    print("foo")

None if True else foo()
```

Unused variable

```
x: int = 0
```

Task 2.2: "Dead Code" Analysis

- ✓ Type theory
- ✓ MLIR Attributes
- ✓ Dead code analysis

Unreachable statement

```
def foo():
    return
    print("DEAD")
    return
```

Unreachable expression

```
def foo():
    print("foo")

None if True else foo()
```

Unused variable

```
x: int = 0
```

Unused store

```
x: int = 0
x = 5
```

Task 2.2: "Dead Code" Analysis

✓ Type theory
✓ MLIR Attributes
✓ Dead code analysis
✓ Constant analysis

Unreachable statement

```
def foo():
    return
    print("DEAD")
    return
```

Unreachable expression*

```
def foo():
    print("foo")

x = True
None if x else foo()
```

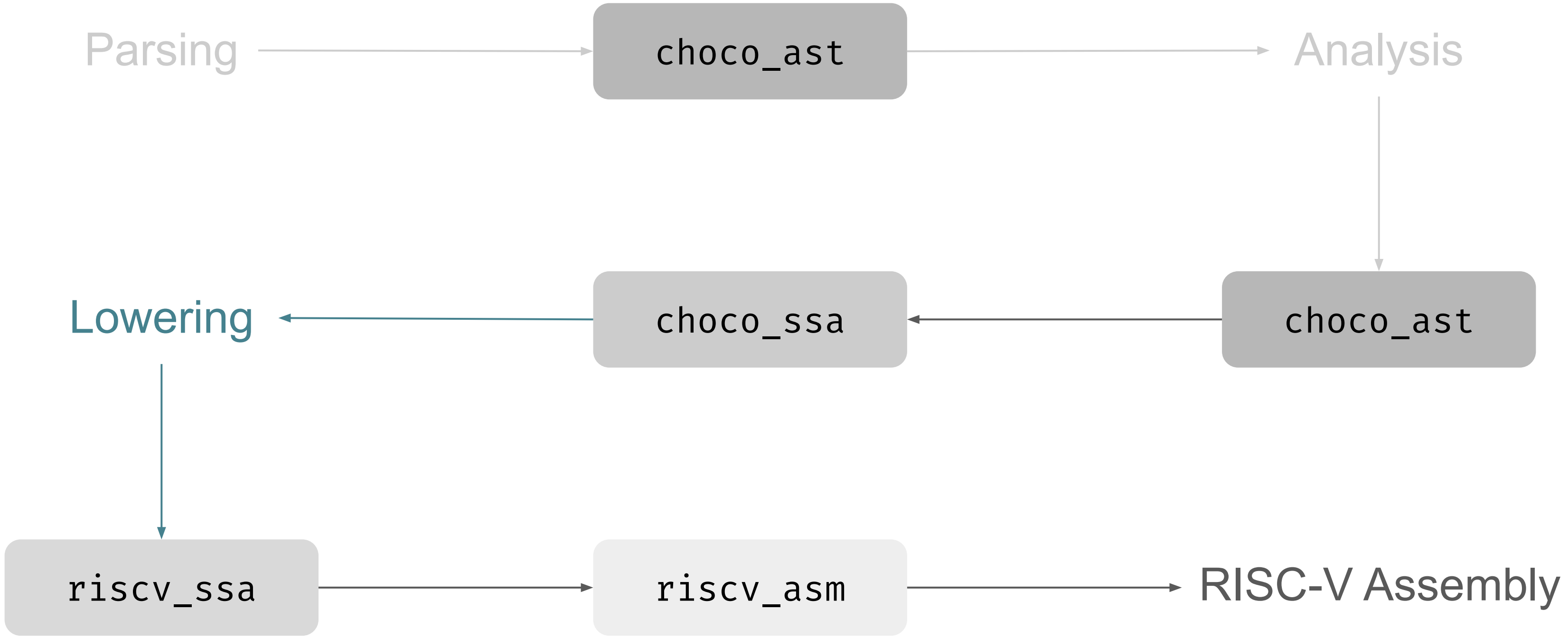
Unused variable

```
x: int = 0
```

Unused store

```
x: int = 0
x = 5
```

The Student’s Journey



Task 3.1: Lowering with SSA

```
x: bool = True
if x:
  print(0)
else:
  print(1)
```

Task 3.1: Lowering with SSA

```
choco_ssa.func_def @_main() {
  %0 = choco_ssa.literal True
  %1 = choco_ssa.alloc bool
  choco_ssa.store(%1, %0) : bool

  choco_ssa.if (%2) {

    %3 = choco_ssa.literal 0 : int
    choco_ssa.call_expr @print(%3)
  } else {

    %4 = choco_ssa.literal 1 : int
    choco_ssa.call_expr @print(%4)
  }
  return
}
```

```
riscv_ssa.func @_main() {
  %0 = riscv_ssa.li 1
  %1 = riscv_ssa.alloc
  riscv_ssa.sw(%0, %1)
  %3 = riscv_ssa.li 0
  riscv_ssa.beq(%2, %3) "if_else_1"
  riscv_ssa.label "if_then_1"
  %4 = riscv_ssa.li 0
  riscv_ssa.call @_print_int(%4)
  riscv_ssa.j "if_after_1"
  riscv_ssa.label "if_else_1"
  %5 = riscv_ssa.li 1
  riscv_ssa.call @_print_int(%5)
  riscv_ssa.label "if_after_1"
  riscv_ssa.ret
}
```

✓ SSA
✓ Peephole Rewrites
✓ Alloca, Store, Load

Task 3.1: Variable Initialisation

```
choco_ssa.func_def @_main() {
  %0 = choco_ssa.literal True
  %1 = choco_ssa.alloc bool
  choco_ssa.store(%1, %0) : bool

  choco_ssa.if (%2) {

    %3 = choco_ssa.literal 0 : int
    choco_ssa.call_expr @print(%3)
  } else {

    %4 = choco_ssa.literal 1 : int
    choco_ssa.call_expr @print(%4)
  }
  return
}
```

```
riscv_ssa.func @_main() {
  %0 = riscv_ssa.li 1
  %1 = riscv_ssa.alloc
  riscv_ssa.sw(%0, %1)
  %3 = riscv_ssa.li 0
  riscv_ssa.beq(%2, %3) "if_else_1"
  riscv_ssa.label "if_then_1"
  %4 = riscv_ssa.li 0
  riscv_ssa.call @_print_int(%4)
  riscv_ssa.j "if_after_1"
  riscv_ssa.label "if_else_1"
  %5 = riscv_ssa.li 1
  riscv_ssa.call @_print_int(%5)
  riscv_ssa.label "if_after_1"
  riscv_ssa.ret
}
```

✓ SSA
✓ Peephole Rewrites
✓ Alloca, Store, Load
✓ Compilation of SCF

Task 3.1: Control Flow

```
choco_ssa.func_def @_main() {
  %0 = choco_ssa.literal True
  %1 = choco_ssa.alloc bool
  choco_ssa.store(%1, %0) : bool

  choco_ssa.if (%2) {

    %3 = choco_ssa.literal 0 : int
    choco_ssa.call_expr @print(%3)
  } else {

    %4 = choco_ssa.literal 1 : int
    choco_ssa.call_expr @print(%4)
  }
  return
}
```

```
riscv_ssa.func @_main() {
  %0 = riscv_ssa.li 1
  %1 = riscv_ssa.alloc
  riscv_ssa.sw(%0, %1)
  %3 = riscv_ssa.li 0
  riscv_ssa.beq(%2, %3) "if_else_1"
  riscv_ssa.label "if_then_1"
  %4 = riscv_ssa.li 0
  riscv_ssa.call @_print_int(%4)
  riscv_ssa.j "if_after_1"
  riscv_ssa.label "if_else_1"
  %5 = riscv_ssa.li 1
  riscv_ssa.call @_print_int(%5)
  riscv_ssa.label "if_after_1"
  riscv_ssa.ret
}
```

✓ SSA
✓ Peephole Rewrites
✓ Alloca, Store, Load
✓ Compilation of SCF

Task 3.1: Simplified Function Calling

```
choco_ssa.func_def @_main() {
  %0 = choco_ssa.literal True
  %1 = choco_ssa.alloc bool
  choco_ssa.store(%1, %0) : bool

  choco_ssa.if (%2) {

    %3 = choco_ssa.literal 0 : int
    choco_ssa.call_expr @print(%3)
  } else {

    %4 = choco_ssa.literal 1 : int
    choco_ssa.call_expr @print(%4)
  }
  return
}
```

```
riscv_ssa.func @_main() {
  %0 = riscv_ssa.li 1
  %1 = riscv_ssa.alloc
  riscv_ssa.sw(%0, %1)
  %3 = riscv_ssa.li 0
  riscv_ssa.beq(%2, %3) "if_else_1"
  riscv_ssa.label "if_then_1"
  %4 = riscv_ssa.li 0
  riscv_ssa.call @_print_int(%4)
  riscv_ssa.j "if_after_1"
  riscv_ssa.label "if_else_1"
  %5 = riscv_ssa.li 1
  riscv_ssa.call @_print_int(%5)
  riscv_ssa.label "if_after_1"
  riscv_ssa.ret
}
```

- ✓ SSA
- ✓ Peephole Rewrites
- ✓ Alloca, Store, Load
- ✓ Compilation of SCF

Task 3.2: Optimization

```
x: bool = True
if x:
  print(0)
else:
  print(1)
```

✓ SSA
✓ Peephole Rewrites
✓ Alloca, Store, Load
✓ Compilation of SCF

Task 3.2: Optimization

```
x: bool = True
if x:
  print(0)
else:
  print(1)
```

```
addi sp, sp, -4
sw ra, 0(sp)
addi sp, sp, -24
addi sp, sp, -4
mv tp, sp

li t0, 1
sw t0, 0(sp)
addi t0, sp, 24
sw t0, 4(sp)

lw t1, 0(sp)
lw t2, 4(sp)
sw t1, 0(t2)

lw t1, 4(sp)
lw t0, 0(t1)
sw t0, 8(sp)

li t0, 0
sw t0, 12(sp)

lw t1, 8(sp)
lw t2, 12(sp)
beq t1, t2, if_else_1
```

```
if_then_1:

        li t0, 0
        sw t0, 16(sp)

         # riscv_ssa.call
        lw a0, 16(sp)
        jal ra, _print_int

        j if_after_1

if_else_1:

        li t0, 1
        sw t0, 20(sp)

         # riscv_ssa.call
        lw a0, 20(sp)
        jal ra, _print_int
```

```
if_after_1:

         # Footer Ops
__main_return:
        addi sp, sp, 24
        addi sp, sp, 4
        lw ra, 0(sp)
        addi sp, sp, 4

         # Exit program
        li a0, 0
        li a7, 93
        ecall
```

Task 3.2: Optimization

- ✓ SSA
- ✓ Peephole Rewrites
- ✓ Alloca, Store, Load
- ✓ Compilation of SCF
- ✓ Constant Folding

```
x: bool = True
if x:
  print(0)
else:
  print(1)
```

```
addi sp, sp, -4
sw ra, 0(sp)
addi sp, sp, -24
addi sp, sp, -4
mv tp, sp

li t0, 1
sw t0, 0(sp)
addi t0, sp, 24
sw t0, 4(sp)

lw t1, 0(sp)
lw t2, 4(sp)
sw t1, 0(t2)

lw t1, 4(sp)
lw t0, 0(t1)
sw t0, 8(sp)

li t0, 0
sw t0, 12(sp)

lw t1, 8(sp)
lw t2, 12(sp)
beq t1, t2, if_else_1
```

```
if_then_1:

            li t0, 0
            sw t0, 16(sp)

             # riscv_ssa.call
            lw a0, 16(sp)
            jal ra, _print_int

            j if_after_1

if_else_1:

            li t0, 1
            sw t0, 20(sp)

             # riscv_ssa.call
            lw a0, 20(sp)
            jal ra, _print_int
```

```
if_after_1:

             # Footer Ops
__main_return:
            addi sp, sp, 24
            addi sp, sp, 4
            lw ra, 0(sp)
            addi sp, sp, 4

             # Exit program
            li a0, 0
            li a7, 93
            ecall
```

✓ SSA
✓ Peephole Rewrites
✓ Alloca, Store, Load
✓ Compilation of SCF
✓ Constant Folding
✓ Mem2Reg

Task 3.2: Optimization

```
x: bool = True
if x:
  print(0)
else:
  print(1)
```

```
addi sp, sp, -4
sw ra, 0(sp)
addi sp, sp, -24
addi sp, sp, -4
mv tp, sp

li t0, 1
sw t0, 0(sp)
addi t0, sp, 24
sw t0, 4(sp)

lw t1, 0(sp)
lw t2, 4(sp)
sw t1, 0(t2)

lw t1, 4(sp)
lw t0, 0(t1)
sw t0, 8(sp)

li t0, 0
sw t0, 12(sp)

lw t1, 8(sp)
lw t2, 12(sp)
beq t1, t2, if_else_1
```

```
if_then_1:

          li t0, 0
          sw t0, 16(sp)

           # riscv_ssa.call
          lw a0, 16(sp)
          jal ra, _print_int

          j if_after_1

if_else_1:

          li t0, 1
          sw t0, 20(sp)

           # riscv_ssa.call
          lw a0, 20(sp)
          jal ra, _print_int
```

```
if_after_1:

           # Footer Ops
__main_return:
          addi sp, sp, 24
          addi sp, sp, 4
          lw ra, 0(sp)
          addi sp, sp, 4

           # Exit program
          li a0, 0
          li a7, 93
          ecall
```

Task 3.2: Optimization

- ✓ SSA
- ✓ Peephole Rewrites
- ✓ Alloca, Store, Load
- ✓ Compilation of SCF
- ✓ Constant Folding
- ✓ Mem2Reg
- ✓ Register allocation
- ✓ Instcombine opts

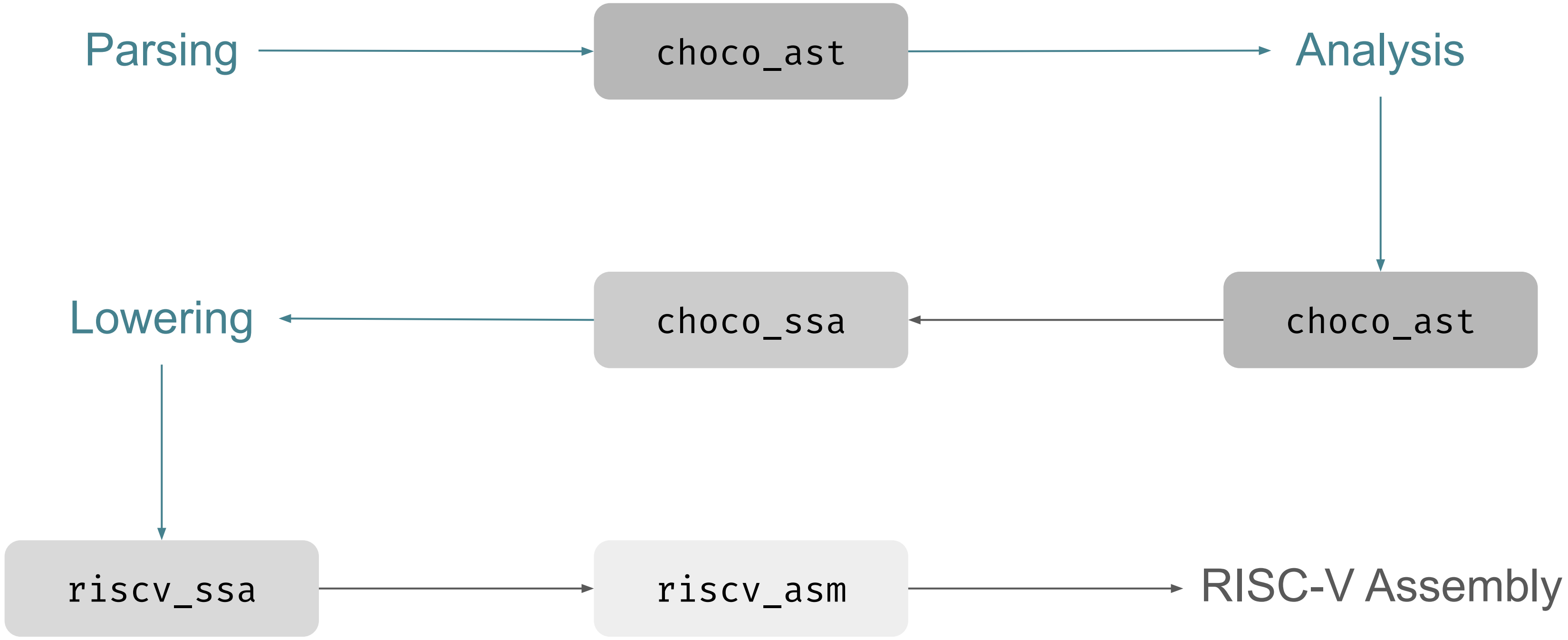
```
x: bool = True
if x:
  print(0)
else:
  print(1)
```

```
addi sp, sp, -4
sw ra, 0(sp)
addi sp, sp, -24
addi sp, sp, -4
mv tp, sp

li t0, 1
sw t0, 0(sp)
addi t0, sp, 24
sw t0, 4(sp)

lw t1, 0(sp)
lw t2, 4(sp)
sw t1, 0(t2)

lw t1, 4(sp)
lw t0, 0(t1)
sw t0, 8(sp)

li t0, 0
sw t0, 12(sp)

lw t1, 8(sp)
lw t2, 12(sp)
beq t1, t2, if_else_1
```

```
if_then_1:

        li t0, 0
        sw t0, 16(sp)

         # riscv_ssa.call
        lw a0, 16(sp)
        jal ra, _print_int

        j if_after_1

if_else_1:

        li t0, 1
        sw t0, 20(sp)

         # riscv_ssa.call
        lw a0, 20(sp)
        jal ra, _print_int
```

```
if_after_1:

         # Footer Ops
__main_return:
        addi sp, sp, 24
        addi sp, sp, 4
        lw ra, 0(sp)
        addi sp, sp, 4

         # Exit program
        li a0, 0
        li a7, 93
        ecall
```

The Student’s Journey



Onboarding to Open Source Compiler Development

`arith, scf, func`

↓

x86

ONNX

↓

`linalg`

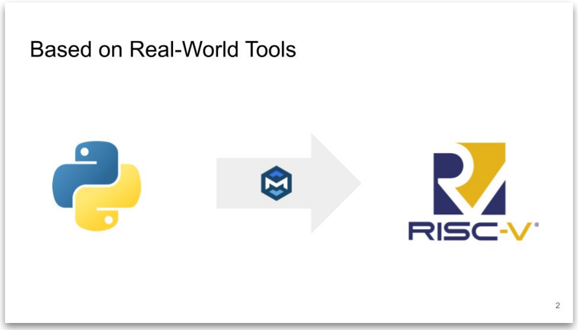
Use in Research

`linalg` microkernels

RISC-V

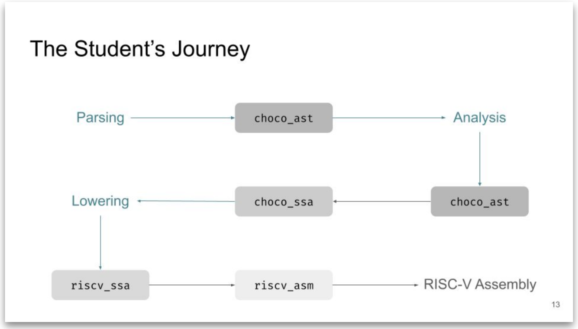
OSS contributions

PSyclone



Based on Real-World Tools

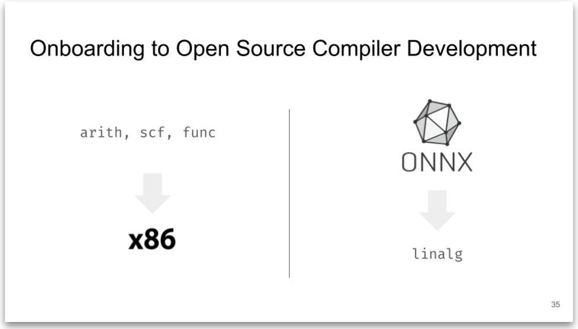
2



The Student's Journey

$$\begin{tikzcd}
\text{Parsing} \arrow[r] & \text{choco\_ast} \arrow[r] & \text{Analysis} \arrow[d] \\
\text{Lowering} \arrow[d] & \text{choco\_ssa} \arrow[l] & \text{choco\_ast} \arrow[l] \\
\text{riscv\_ssa} \arrow[r] & \text{riscv\_asm} \arrow[r] & \text{RISC-V Assembly}
\end{tikzcd}$$

13



Onboarding to Open Source Compiler Development

$$\begin{tikzcd}
\texttt{arith, scf, func} \arrow[d] & \text{ONNX} \arrow[d] \\
\textbf{x86} & \texttt{linalg}
\end{tikzcd}$$

35

https://xdsl.dev/