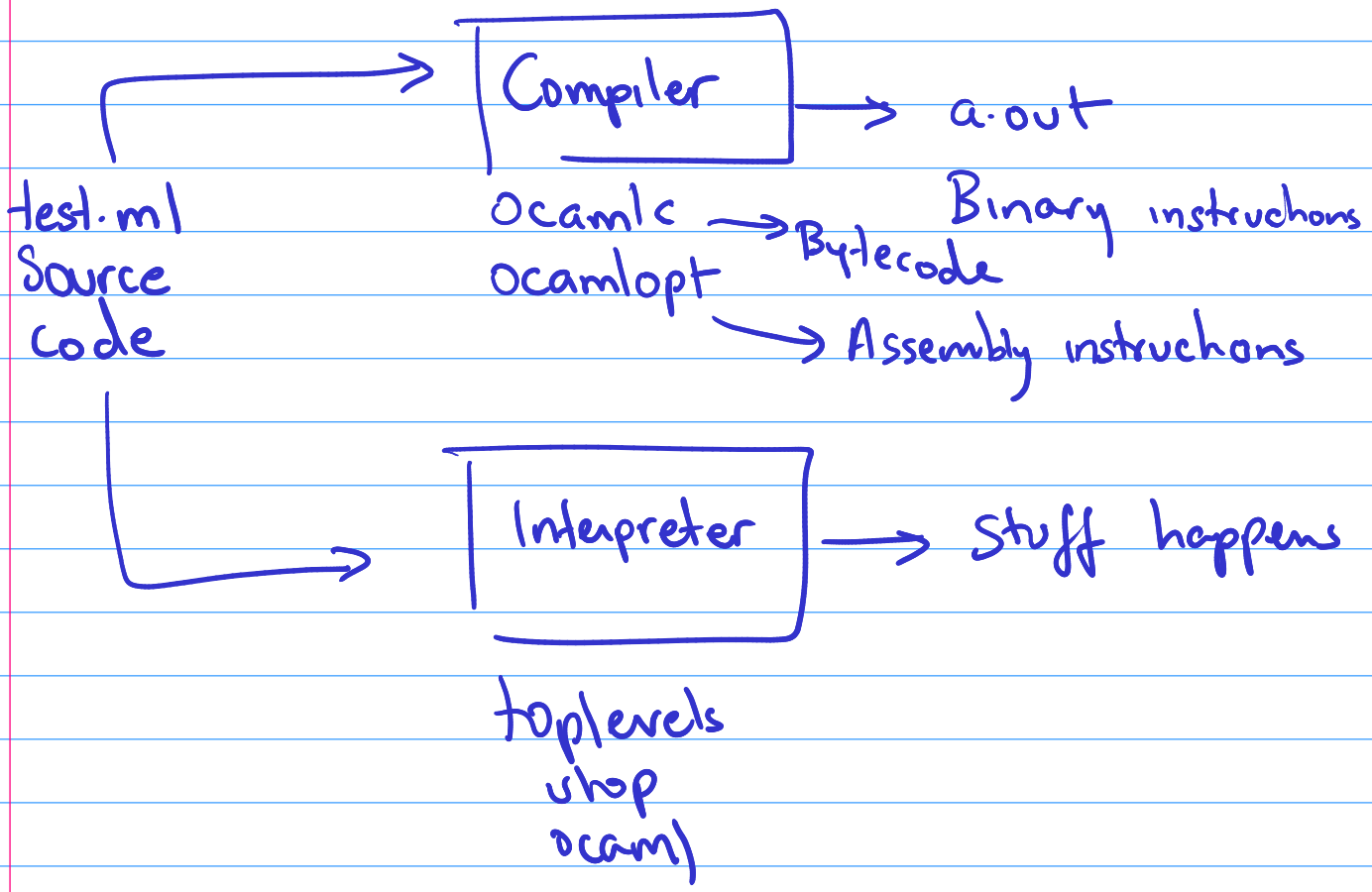
Unit 2 : Implementing a Language Interpreter



"Hosted" implementations

Python — CPython — C

HotSpot — JVM — C++

JavaScript — SpiderMonkey — Rust

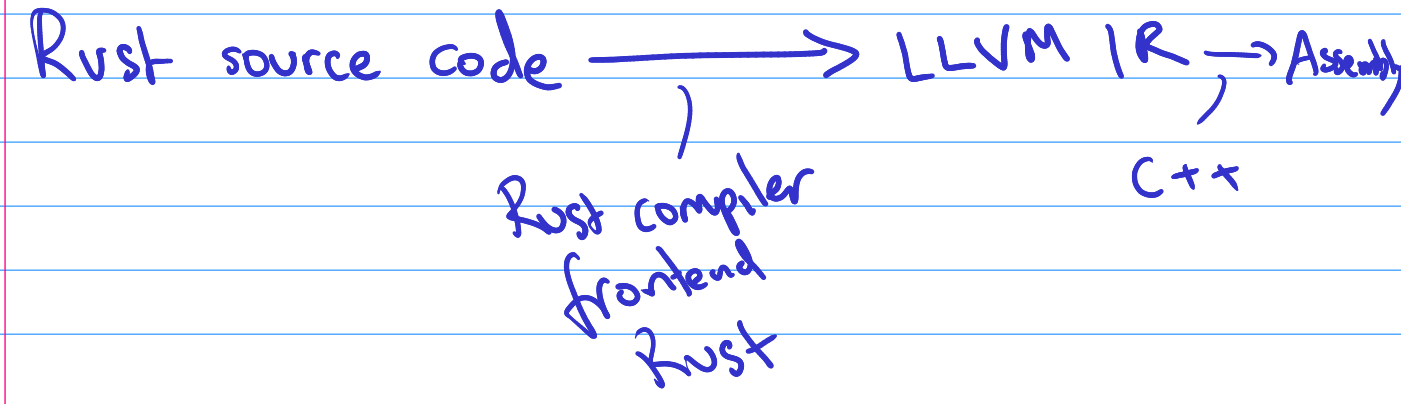
V8 — C++

"Bootstrapped" Implementations

Compiler is written in its own language

gcc — C++

Clang/LLVM — C++

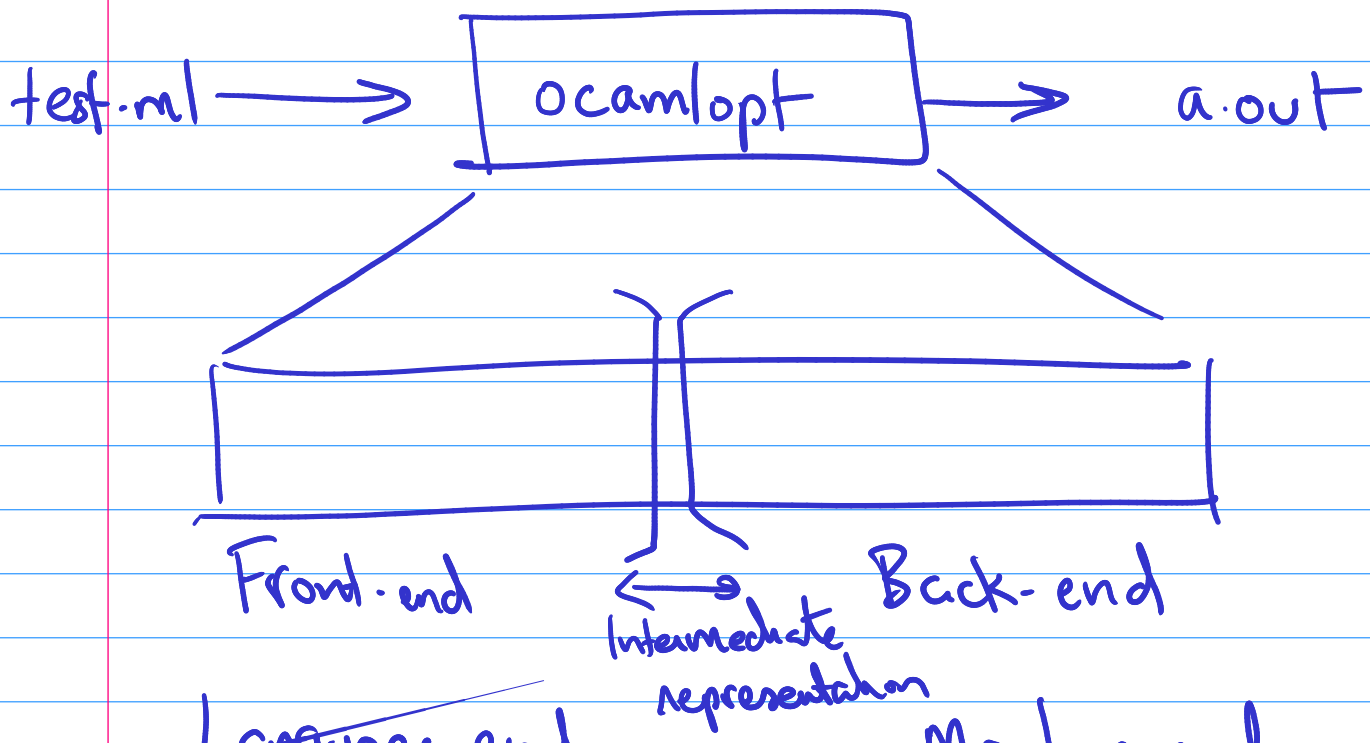


Ocaml opt — OCaml

Boot strapped interpreter

Pypy

Rikes JVM



Language end

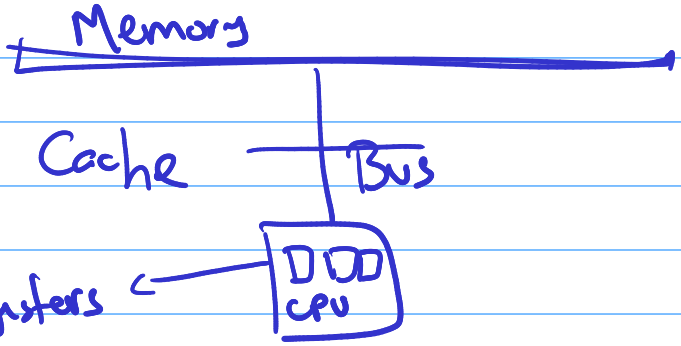
Syntactic analysis

Type system

Optimization

Machine end

Instruction selection

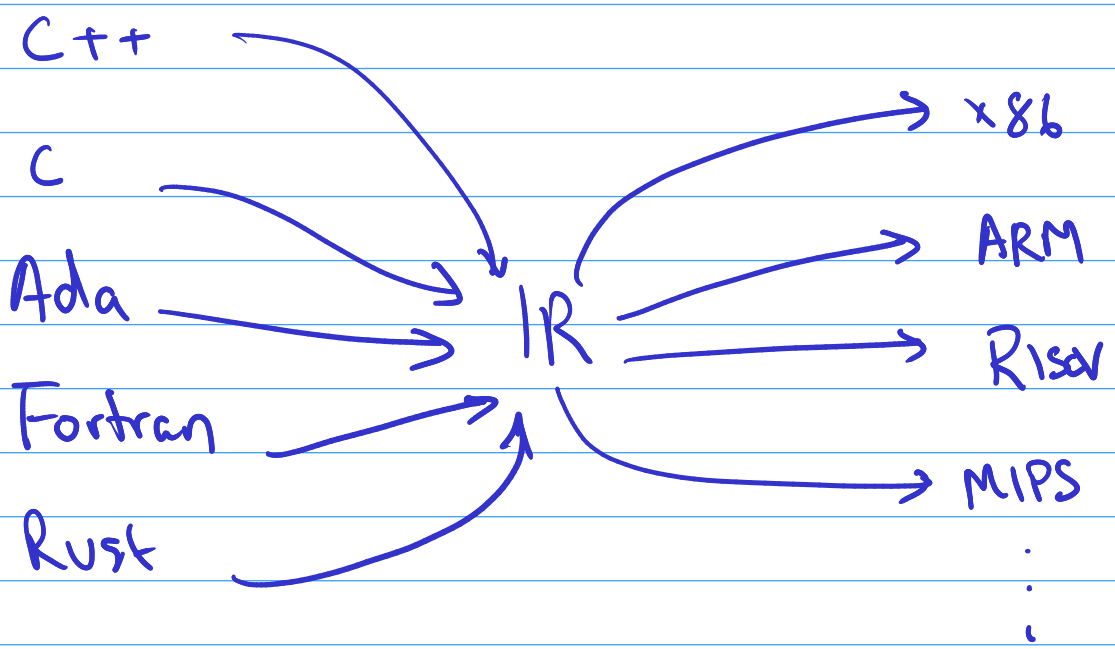


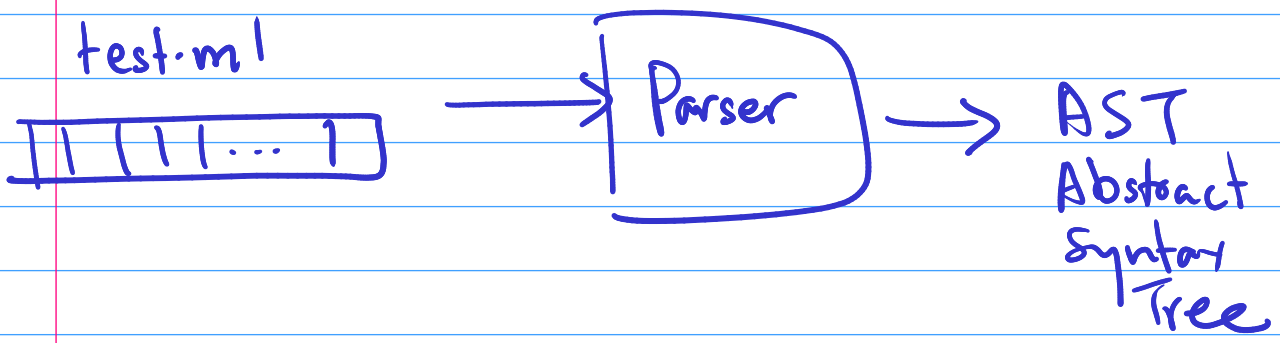
add 32, iax
mov iax, ibx
load 0x938C, ibx

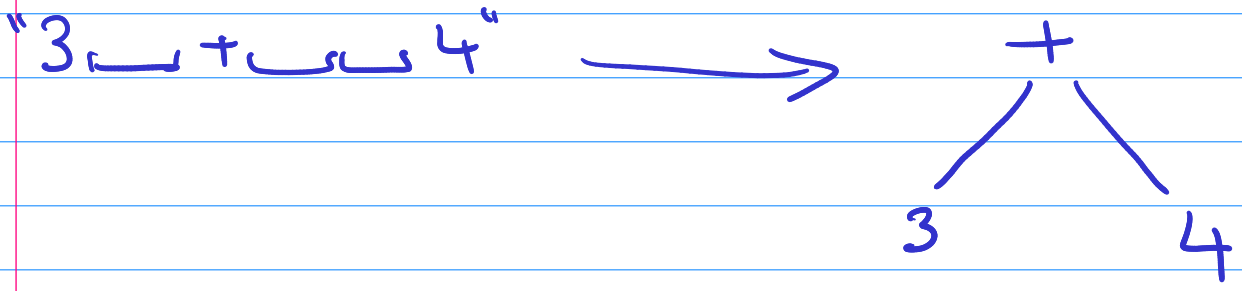
Register allocation

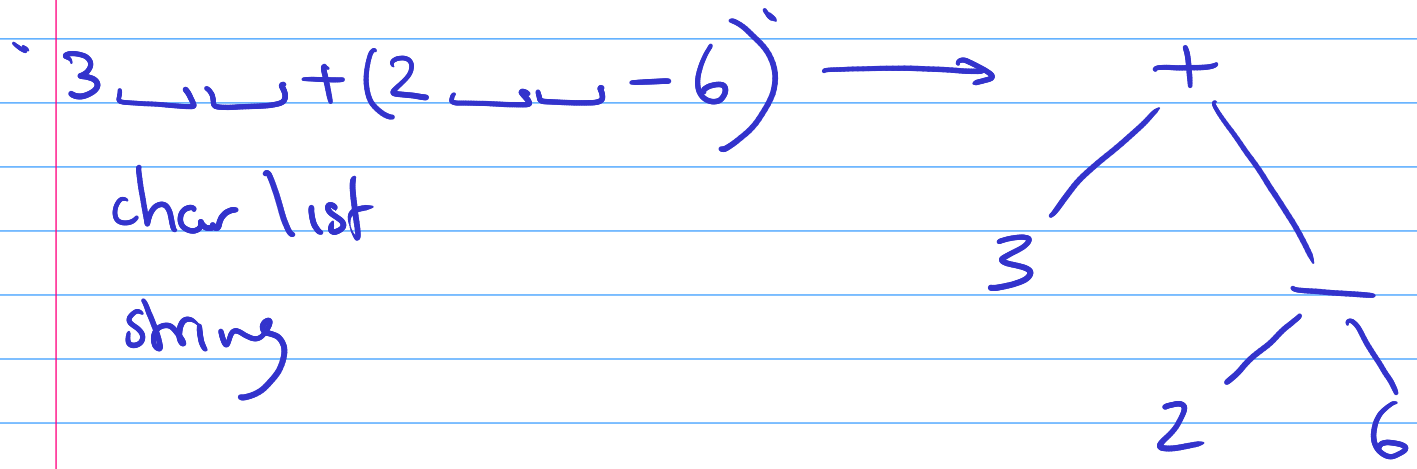
Optimization

Runtime
Garbage
collection


C++
C
Ada
Fortran
Rust
IR
x86
ARM
Risc
MIPS


test.ml
Parser
AST
Abstract
Syntax
Tree


"3 + 4"
+
3
4


`3 + (2 - 6)`
char list
string
+
3
2
6

Parsing

Language specification

Grammar ——— EBNF

Extended Backus Naur Form

Well-formed programs

```
Void foo() {
  int x = 3;
}
```

Ill-formed programs

```
void foo() {
  int int = 5;
}

void foo() {
  int * = 3
}
```

Expr ::= IntegerLiteral (38 | 42 | 36 | ...)

Non-empty sequence of digits

['0' - '9']+

"346" ✓

"" ✓

"-86" ✗

"38+46" ✓

"38++46" ✗

| Expr '+' Expr

↑ Two expressions with a plus sign between

"38-46" ✓

"-46" ✗

| Expr '-' Expr

"38+(46-23)" ✓

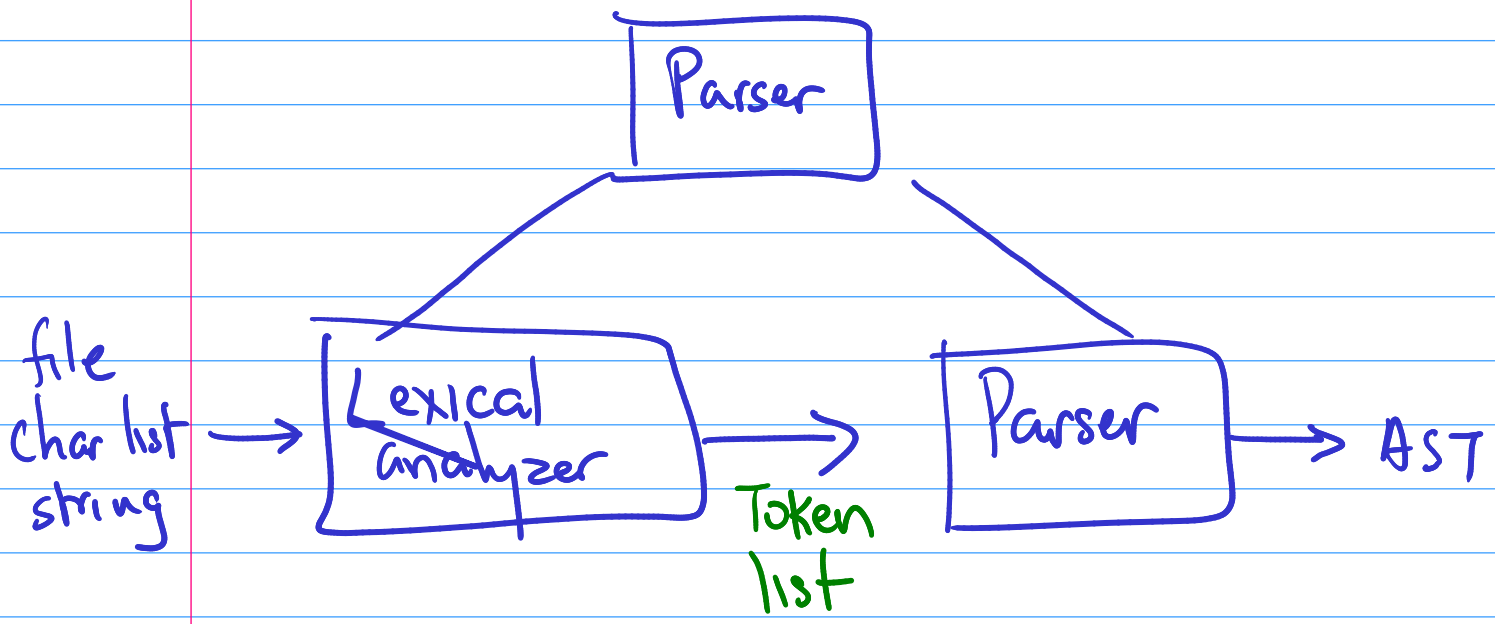
| '(' Expr ')'

"38-42-21" ✓

↑ Might have 2 parse trees

Ambiguity is bad.

Question: How does go from a sequence of characters to a parse tree, if it exists?



"38 + (42 - 21)"

⇓

Int(38); PLUS; LPAREN; Int(42); MINUS; Int(21); RPAREN

Tokens

⇓

Abstract syntax tree ⇒ HW1 eval1 eval2