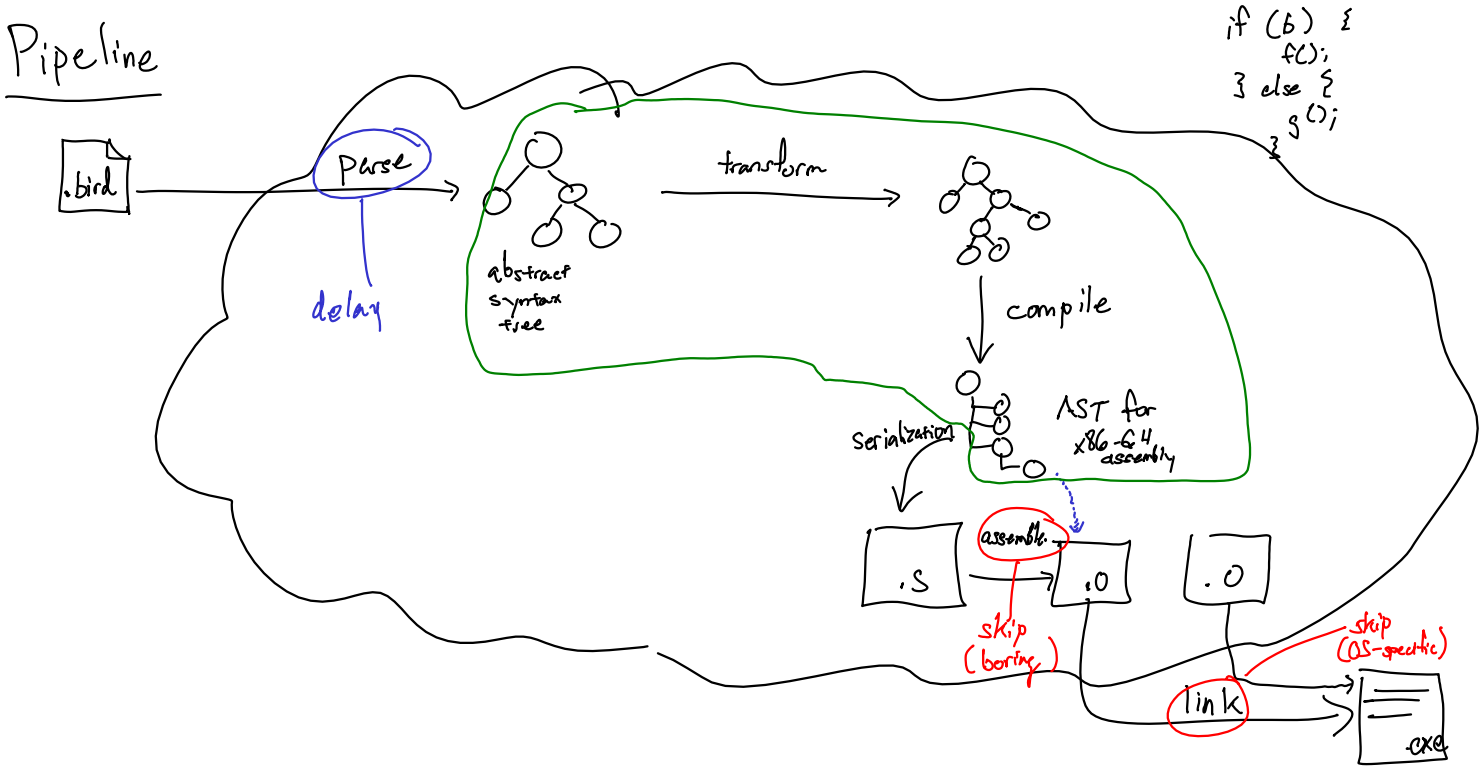
Compiler

program	input (human-readable)	output (machine-usable)
gcc	C	executables
clang	C	executables
rustc	Rust	executables
javac	Java	bytecode
pdflatex	LaTeX	document (PDF)

Pipeline



Syntax and Semantics

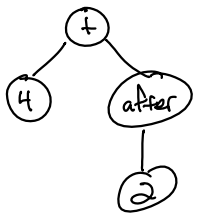
Similar to BNF

Concrete Syntax

```
<expr> ::= 0 | 1 | -1 | 2 | -2 | ...
         | before ( <expr> )
         | after ( <expr> )
         | <expr> + <expr>
         | <expr> - <expr>
         | <expr> * <expr>
         | <variable>
         | let <variable> = <expr> in <expr>
```

} Grammar for Adder

4 + after(2)



AST

```
type expr = EInt of int
          | EBefore of expr
          | EAfter of expr
          | EPlus of expr * expr
          | EMinus of expr * expr
          | ETimes of expr * expr
          | EVar of string
          | ELet of string * expr * expr
```

Abstract Syntax

Semantics "before" gives back number one less than input

x86-64 assembly

AT&T

mov $5, %eax

Intel

mov rax, 5

rax	r8
rbx	r9
rcx	r10
rdx	⋮
rsp	⋮
rbp	⋮
rdi	
rsi	r15

Example

```
mov rax, 4
add rax, 2
```

```
[ AsmMov (ArgReg RAX, ArgConst "4");
  AsmAdd (ArgReg RAX, ArgConst "2");
]
```

```
type register =
  | RAX | ...

type argument =
  | ArgReg of register
  | ArgConst of string

type instruction =
  | AsmMov ...
  | AsmAdd ...
```

Compilation

program AST ⟶ assembly AST

```
let rec compile_expression (e : expr) : instruction list =
  match e with
  | EInt n -> [AsmMov(ArgReg RAX, ArgConst (string_of_int n))]
  | EBefore e' ->
```