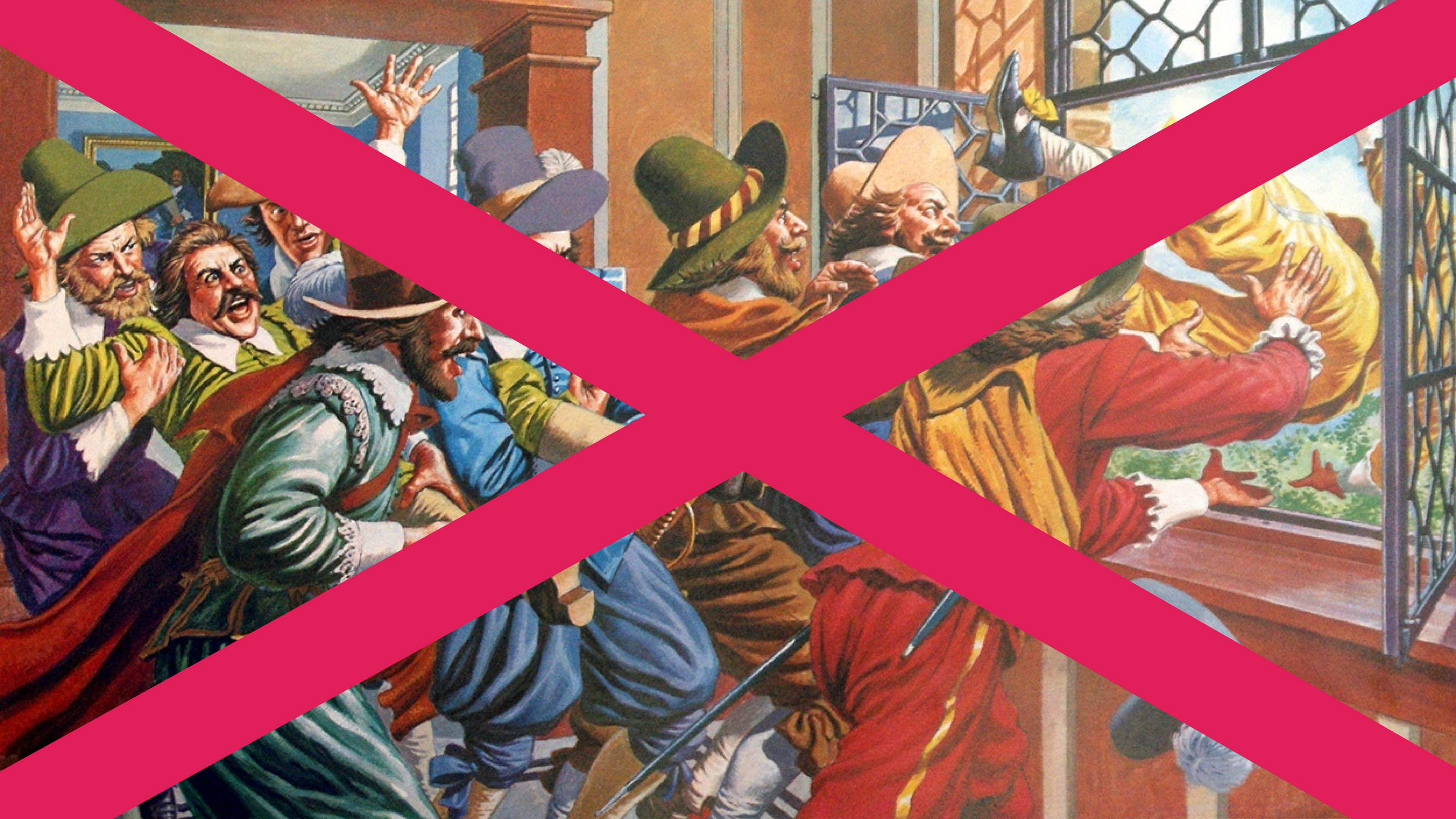
@janiczek
GlobalWebIndex

# Elm in Elm

@janiczek
GlobalWebIndex


YISSSSS


…What? 😮



Cool!!! 

But… why?? 😶

# compiler as a library

compiler as a library

**learning resource**

compiler as a library

learning resource

**experimentation ready**

compiler as a library

learning resource

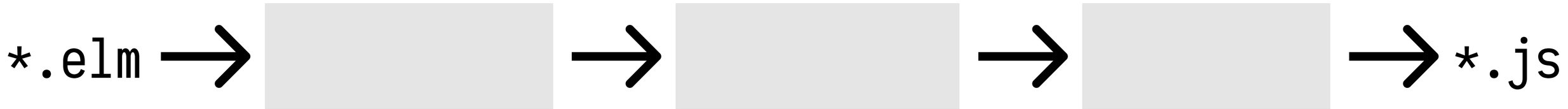
experimentation ready

**extensible**

Awesome! 

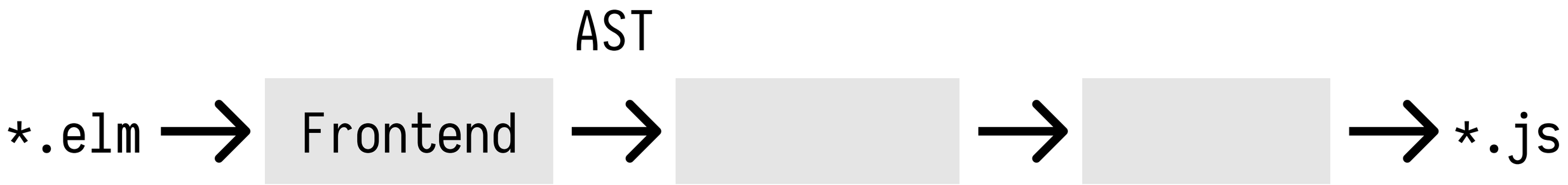
But… how? 

# Three-stage compiler



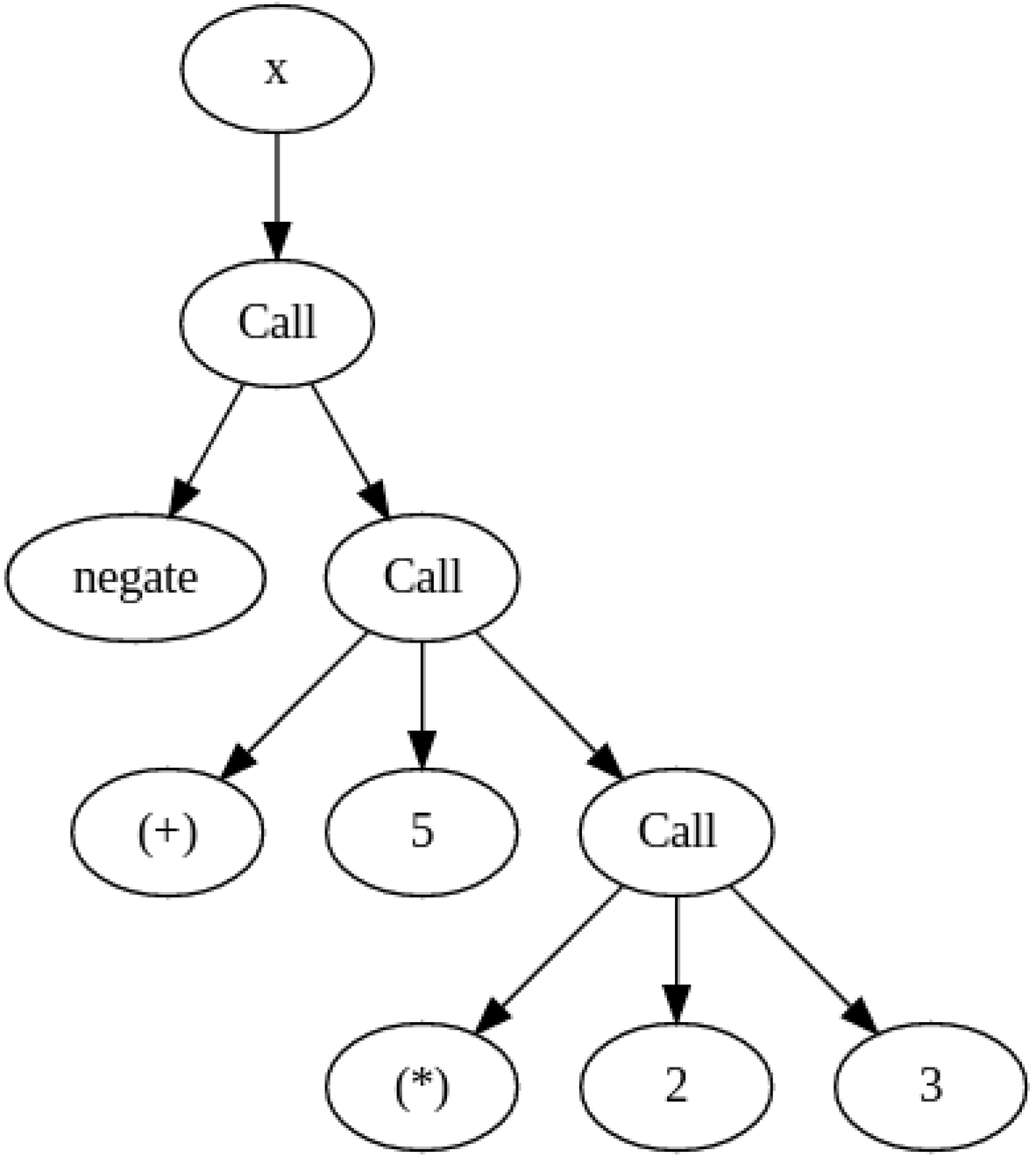
```
x = negate (5 + 2 * 3)
```

# Three-stage compiler



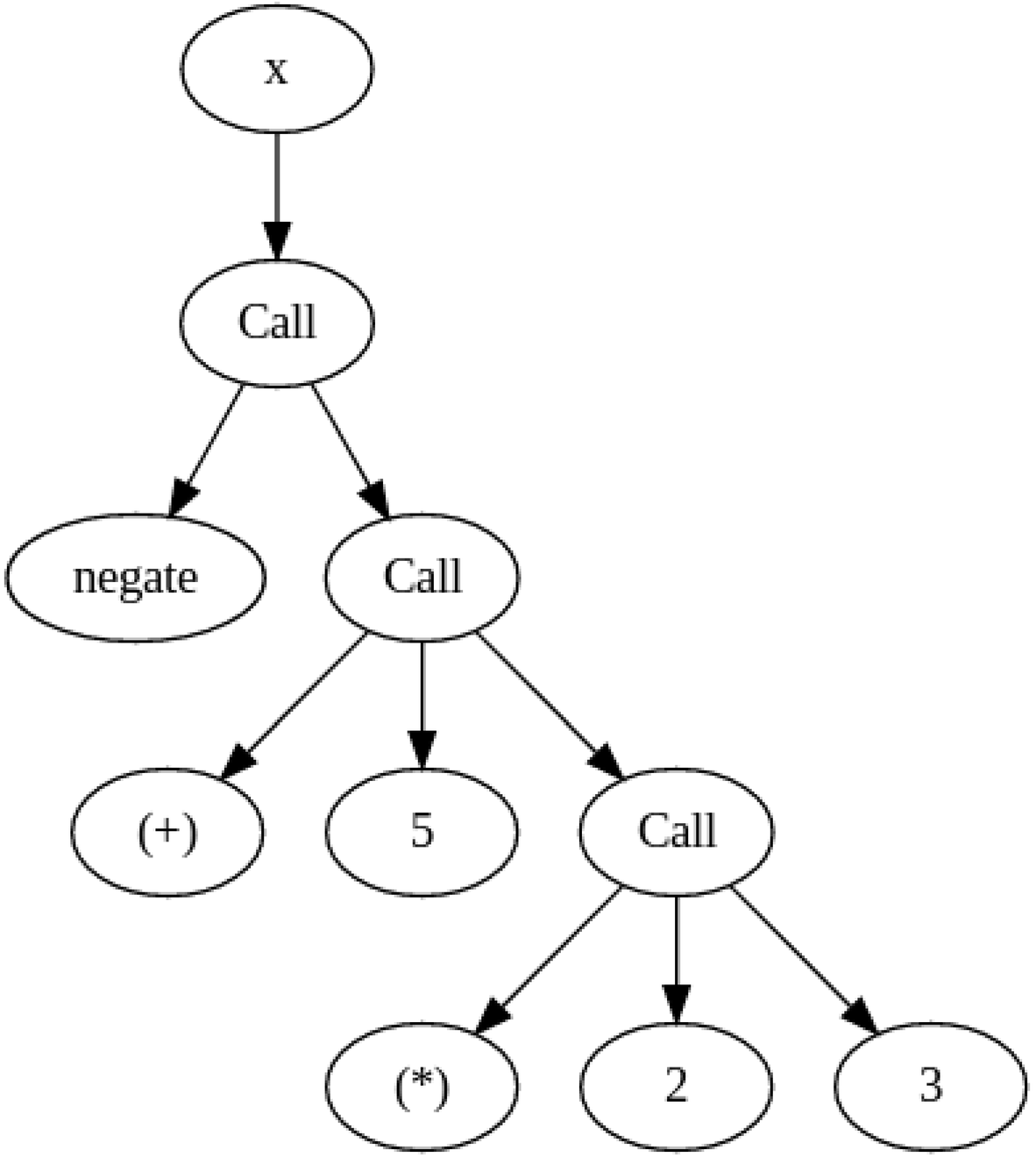

```
parse : String → Result ParseError AST
```

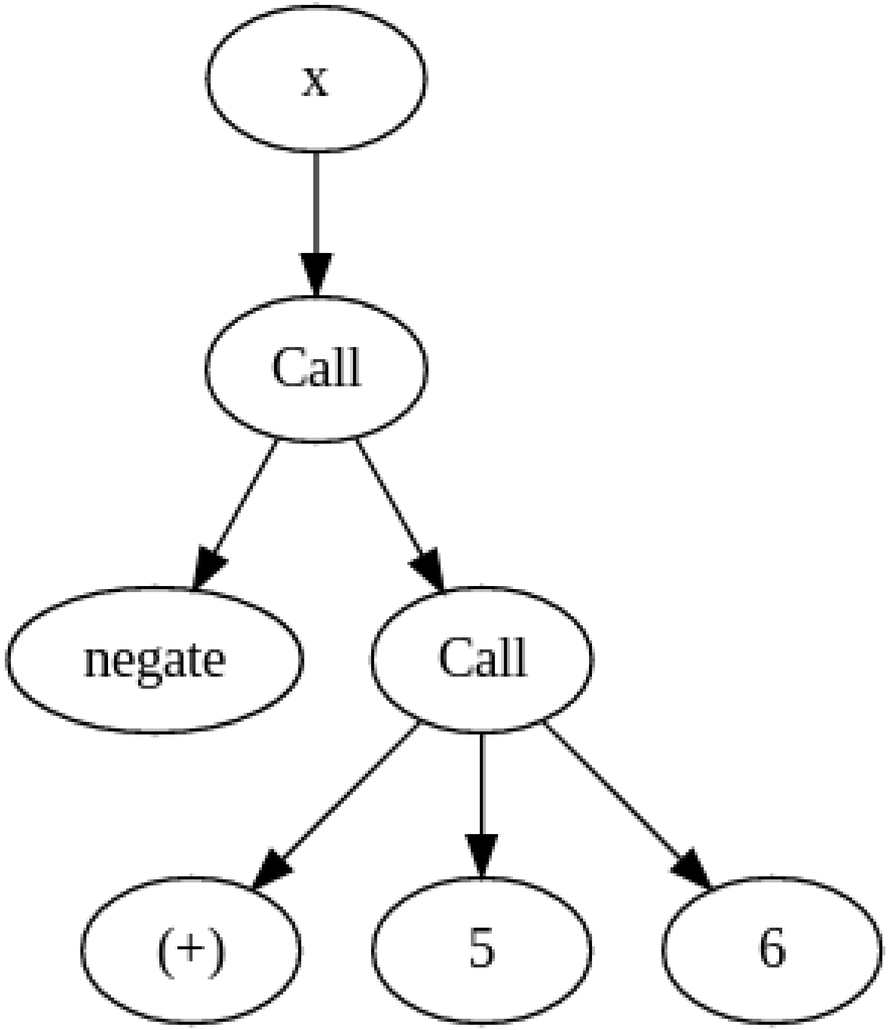
```
x = negate (5 + 2 * 3)
```

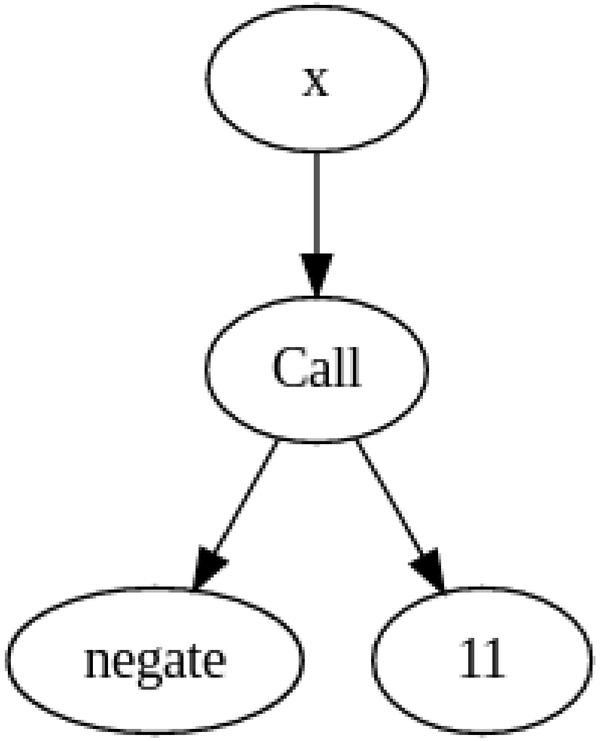

x
Call
negate
Call
(+)
5
Call
(*)
2
3

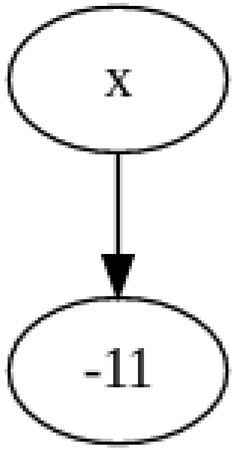
# Three-stage compiler


```
parse    : String → Result ParseError AST
optimize : AST    → AST
```


x
Call
negate
Call
(+)
5
Call
(*)
2
3

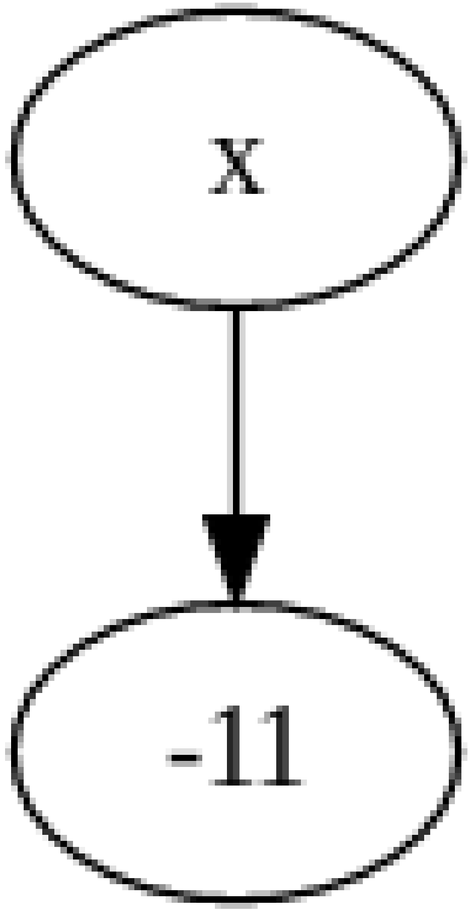

x
Call
negate
Call
(+)
5
6


x
Call
negate
11


x
-11

# Three-stage compiler


```
parse    : String → Result ParseError AST
optimize : AST    → AST
emitJS   : AST    → String
```


x
-11

```
var x = -11;
```

# The reality*

```
Ok input
    |> Result.andThen parse
    |> Result.andThen desugar
    |> Result.andThen inferTypes
    |> Result.andThen optimize
    |> Result.andThen prepareForBackend
    |> Result.andThen emit
    |> writeToFSAndExit
```

*: not the reality

# Parsing


F***ING PARSERS
HOW DO THEY WORK?
memegenerator.net

# elm/parser

# BINARY OPERATORS

# **elm/parser**
# on precedence, binops, etc.:

“This code is kind of tricky, but it is a baseline for what you would need if you wanted to add `/`, `-`, `==`, `&&`, etc. which bring in more complex associativity and precedence rules.”

https://is.gd/precedence

# PRATT PARSERS

elm/parser

**dmy/elm-pratt-parser**

https://is.gd/elmpratt

elm/parser

dmy/elm-pratt-parser

**Thank you** @dmy, @ilias, @turbo_mack

💙💙💙💙💙💙💙💙💙💙💙💙💙💙💙💙💙💙💙

# Type inference

# Hindley-Milner

# Hindley-Milner

**deduce types from shape of expressions**

42

foo 42

```
foo 42 == "abc"
```

1. generate constraints for subexpressions

1. generate constraints for subexpressions
2. solve them as generally as possible

# Optimizing

```
optimizePlus : Expr → Maybe Expr
```

```
optimizePlus : Expr → Maybe Expr
optimizePlus expr =
    case expr of
        Plus (Int left, Int right) →
            Just (Int (left + right))

        _ →
            Nothing
```

```
optimizePlus   : Expr → Maybe Expr
optimizeNegate : Expr → Maybe Expr
optimizeIfBool : Expr → Maybe Expr
```

```
optimize : Expr → Expr
```

# Janiczek/**transform**

Janiczek/transform

# **Control.Lens.Plated**

https://is.gd/lensplated

# Library

```
-- PARSER
Elm.Parser.parseExpr "negate (5 + 2 * 3)"
--> Ok (Call “negate”
            (Plus
                (Int 5)
                (Times
                    (Int 2)
                    (Int 3))))


Elm.Parser.parseModule moduleSource
Elm.Parser.parseProject elmJson elmFiles
```

```
-- TYPE INFERENCE
Elm.TypeInference.inferExpr (Plus (Int 5) (Int 2))
--> Ok ( T.Plus ( T.Int 5, T.TypeInt )
                ( T.Int 2, T.TypeInt )
       , T.TypeInt
       )

Elm.TypeInference.inferModule module
Elm.TypeInference.inferProject project
```

```
-- OPTIMIZE
Elm.Optimize.optimize thatNegateExample
--> ( T.Int (-11), T.TypeInt )

Elm.Optimize.optimizeWith myOptimizations typedExpr
```

```
-- EMIT
Elm.Emit.JavaScript.emitExpr thatNegateExample
⟶ "-(5 + 2 * 3)"
```

```
Elm.Emit.emitProject
    ToOneFile
    myEmitNative
    project
```

```
Elm.Emit.emitProject
    ToSeparateFiles
    myEmitElixir
    project
```

```
-- BONUS ???
Elm.Eval.evalString elmString
--> Ok (Elm.Value.Int -11)

Elm.Eval.evalExpr elmExpr
Elm.Eval.evalExprWithModule module elmExpr
Elm.Eval.evalExprWithProject project elmExpr
```

# What’s next?


DO TODO TODO TODO TODO TODO TO

- [ ] feature parity of language itself

- [ ] feature parity of language itself
- [ ] publish as a library!

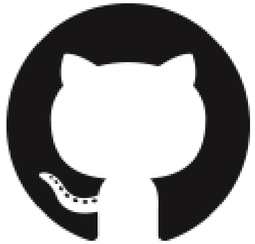
- [ ] feature parity of language itself
- [ ] publish as a library!
- [ ] publish as a CLI tool!

- [ ] feature parity of language itself
- [ ] publish as a library!
- [ ] publish as a CLI tool!
- [ ] example usage *(Slack bot, Klipse, ...)*

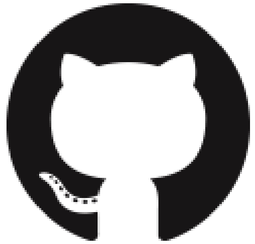
- [ ] feature parity of language itself
- [ ] publish as a library!
- [ ] publish as a CLI tool!
- [ ] example usage *(Slack bot, Klipse, ...)*
- [ ] experiment *(optimizations, native?, ...)*

- [ ] feature parity of language itself
- [ ] publish as a library!
- [ ] publish as a CLI tool!
- [ ] example usage *(Slack bot, Klipse, ...)*
- [ ] experiment *(optimizations, native?, ...)*

...

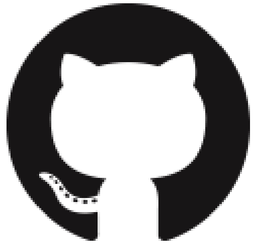


# elm-in-elm/compiler



elm-in-elm/compiler

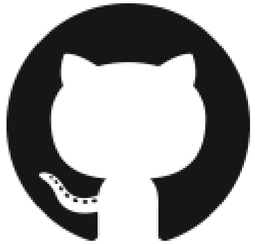
is.gd/elmdiscord



elm-in-elm/compiler

is.gd/elmdiscord

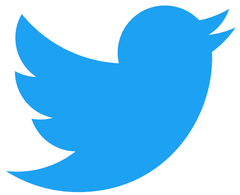
... #elm-in-elm ???



elm-in-elm/compiler

is.gd/elmdiscord

... #elm-in-elm ???



@janiczek

elm-in-elm/compiler

is.gd/elmdiscord

... #elm-in-elm ???

@janiczek

# Thank you!