Connect to the wifi and grab this repo:

Wi-Fiに接続し、このリポジトリをクローンしてください：

https://github.com/damncabbage/js-compiler-workshop

Building a Small Compiler in JavaScript

Rob Howard
@damncabbage
http://robhoward.id.au

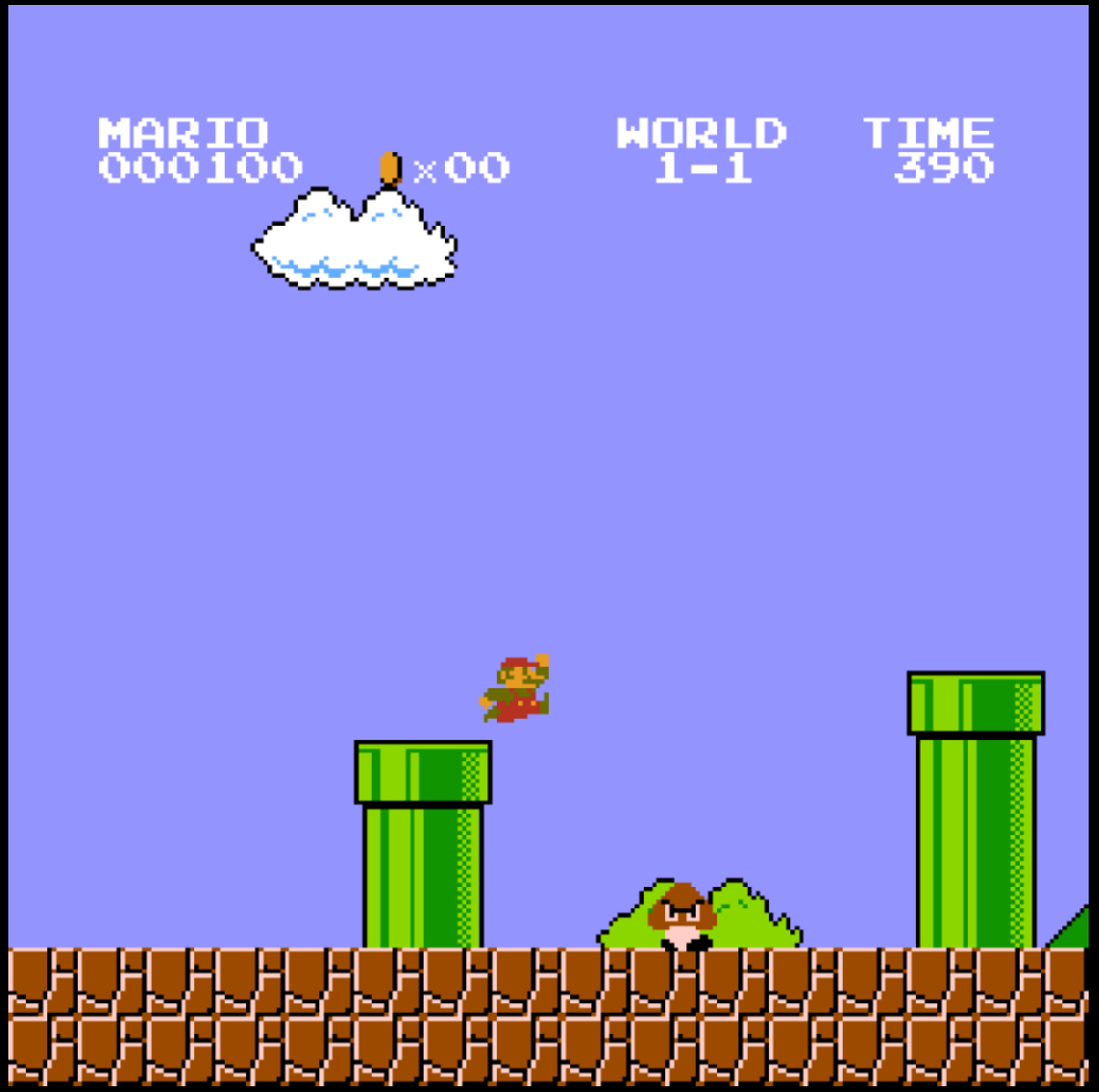

HOUSE

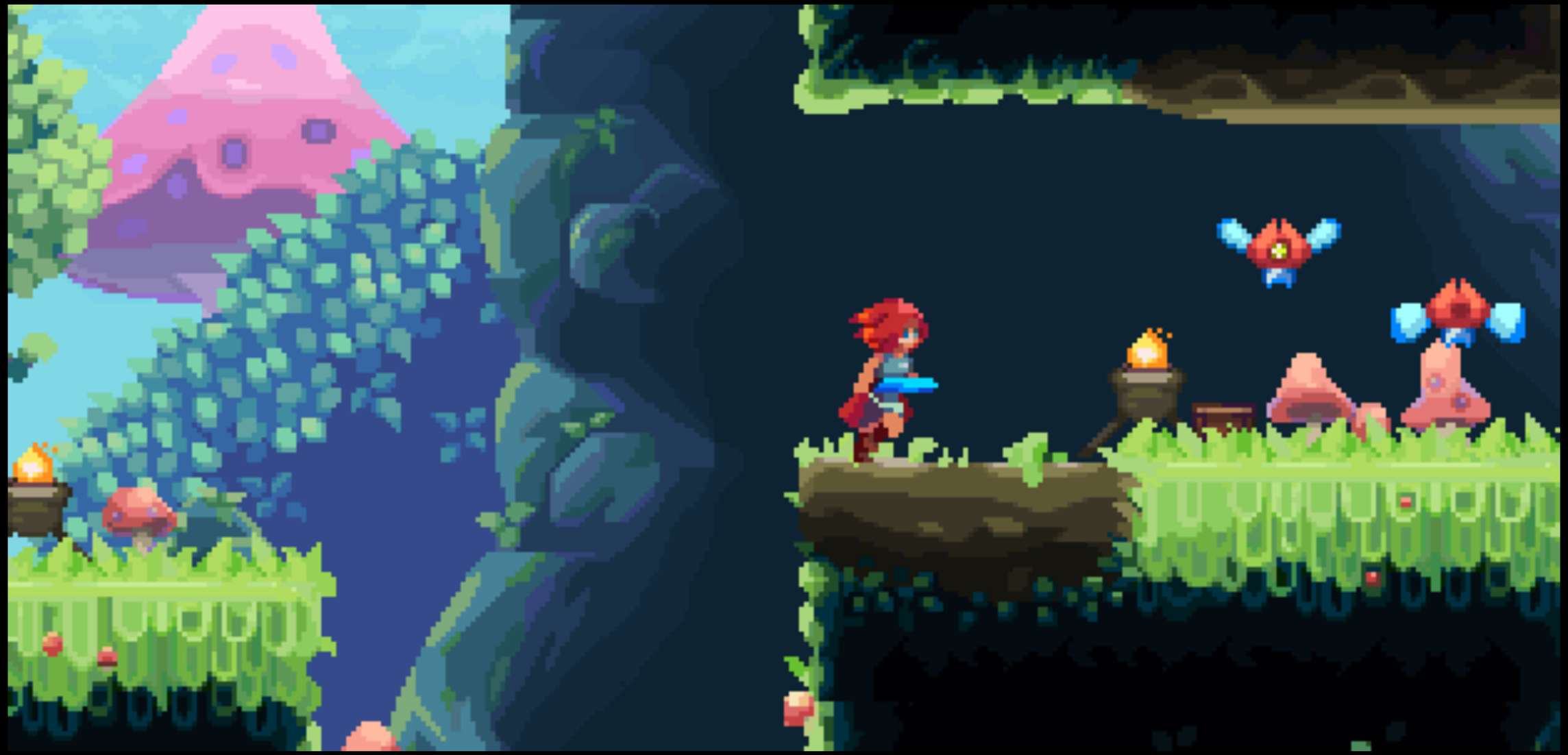
— https://twitter.com/Paige_Burton/status/1063725333487149056

— https://twitter.com/Paige_Burton/status/1063725333487149056

What are Compilers?


“Compilers” are a
fuzzy categorisation
of Programs


MARIO
000100
×00
WORLD
1-1
TIME
390




Input →
Compiler
→ Output


Text →
Compiler
→ Program


C →
clang + llvm
→ .exe


Elm
elm
JS

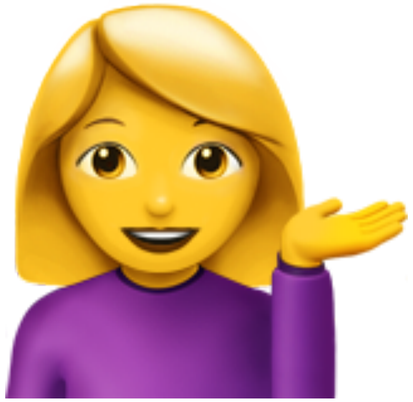

ES2018 →
Babel
→ ES5

But Also


PNG
ImageMagick
JPEG


PDF →
Pandoc
→ HTML

Programming Languages



```
x = add(1, 2);
```

`x = add(1, 2);` "Tokens" 構文素

- A variable called “x”, ”x”と呼ばれる変数
- An Equals symbol, 等号
- A variable called "add", ”add”と呼ばれる変数
- An Open Parenthesis symbol, 開いている括弧記号
- A literal number “1”, 1の定数式
- A Comma symbol, コンマ記号
- A literal number “2", 2の定数式
- A Close Parenthesis symbol, 閉じ括弧記号
- A statement terminator symbol. 文終了記号

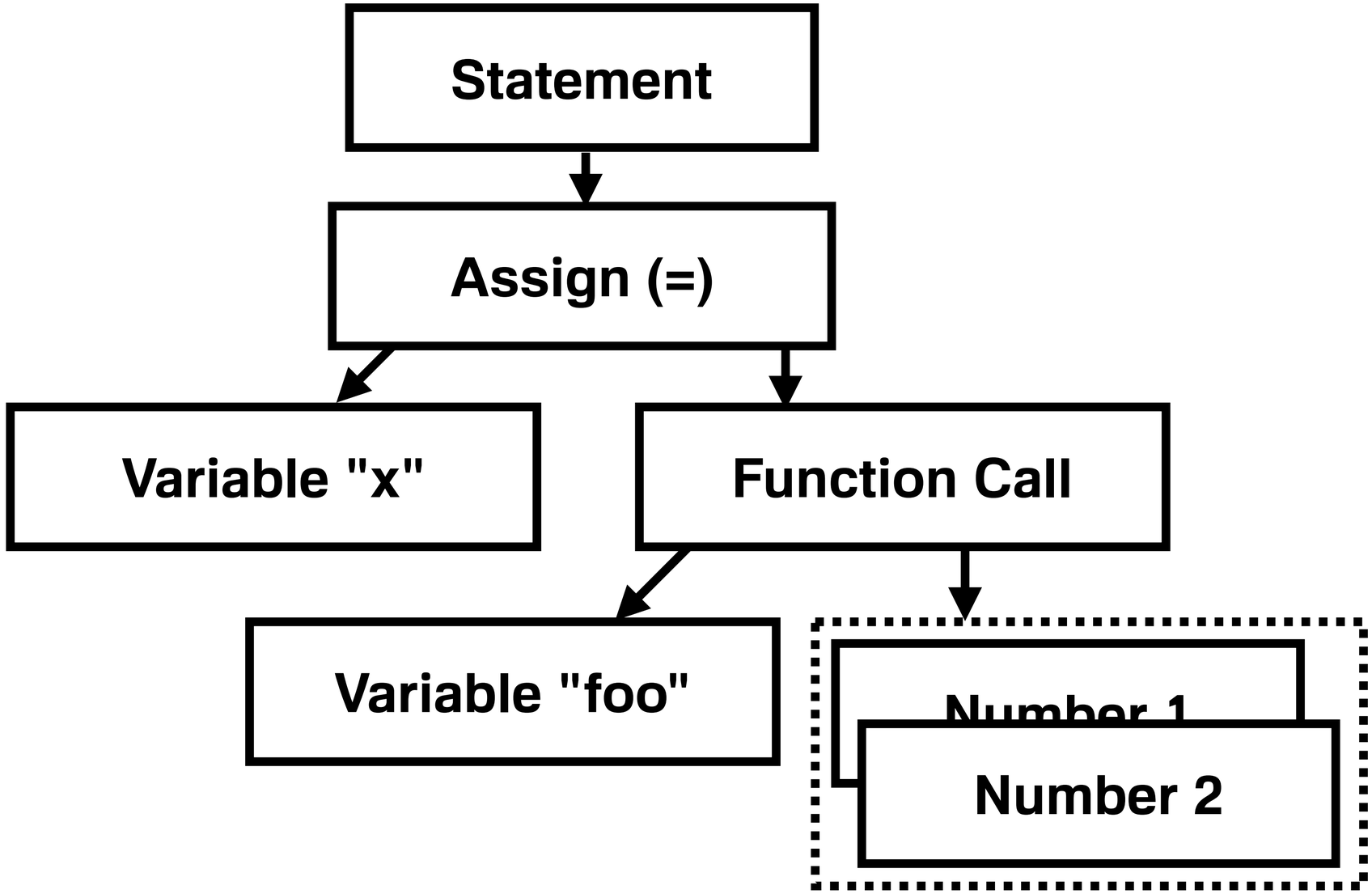
```
x = add(1, 2);
```

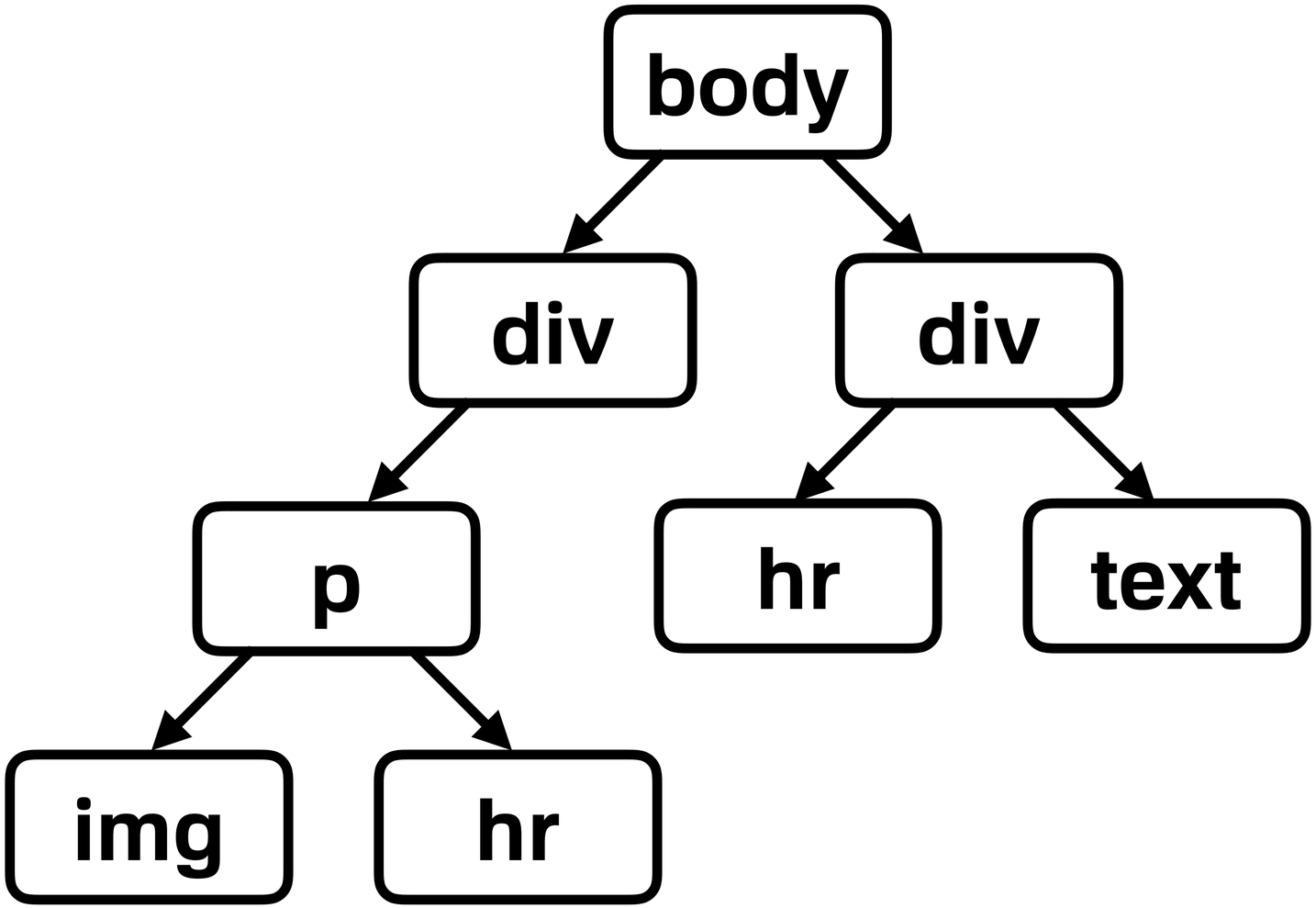
"Grammar" 文法

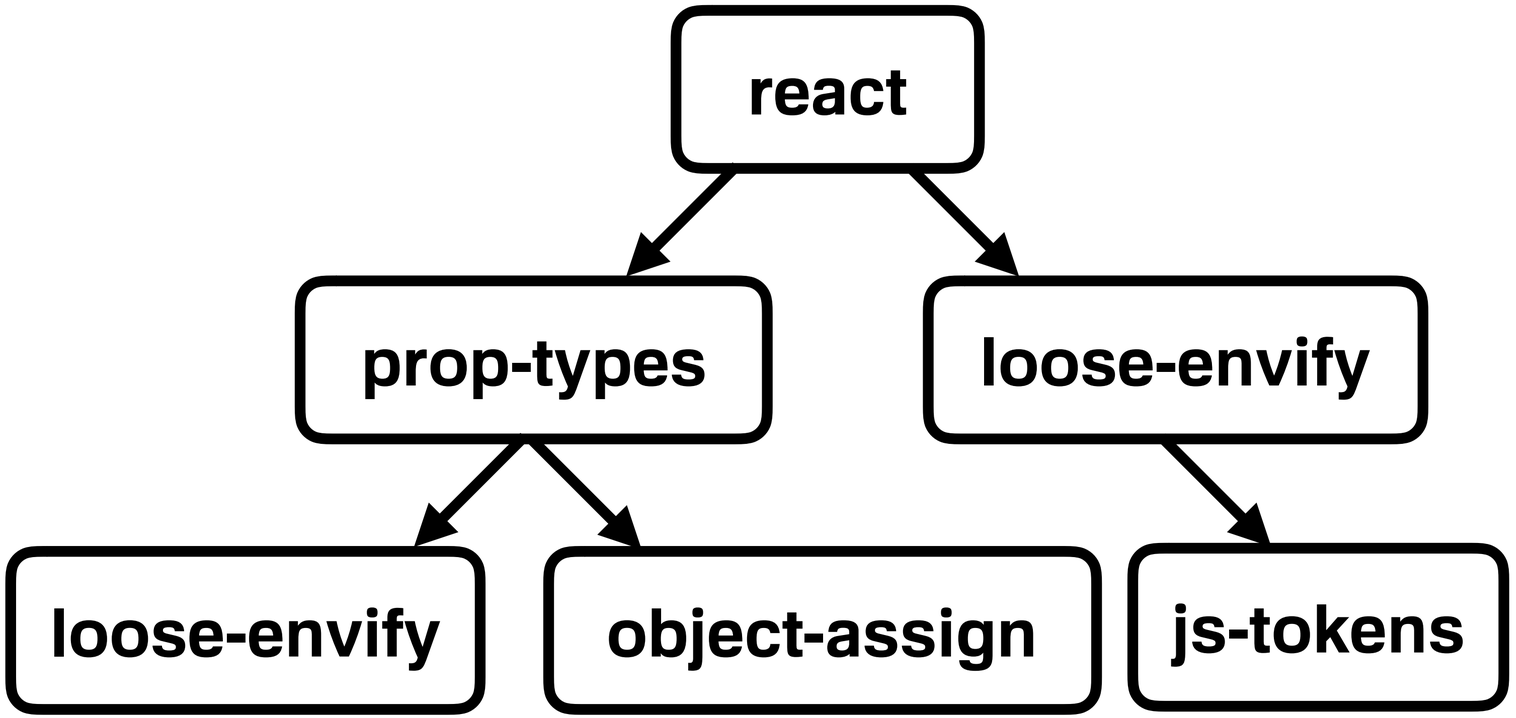
```
statement   = (assignment | expression) ';'
assignment  = identifier '=' expression
expression  = (number | identifier | funcCall)
number      = ('0' | '1' | '2' | '3' | …)
identifier  = ('a' | 'b' | …) identifier?
funcCall    = identifier '(' funcArgs? ')'
funcArgs    = expression funcArgTail?
funcArgTail = ',' funcArgs
```

```
x = add(1, 2);
```

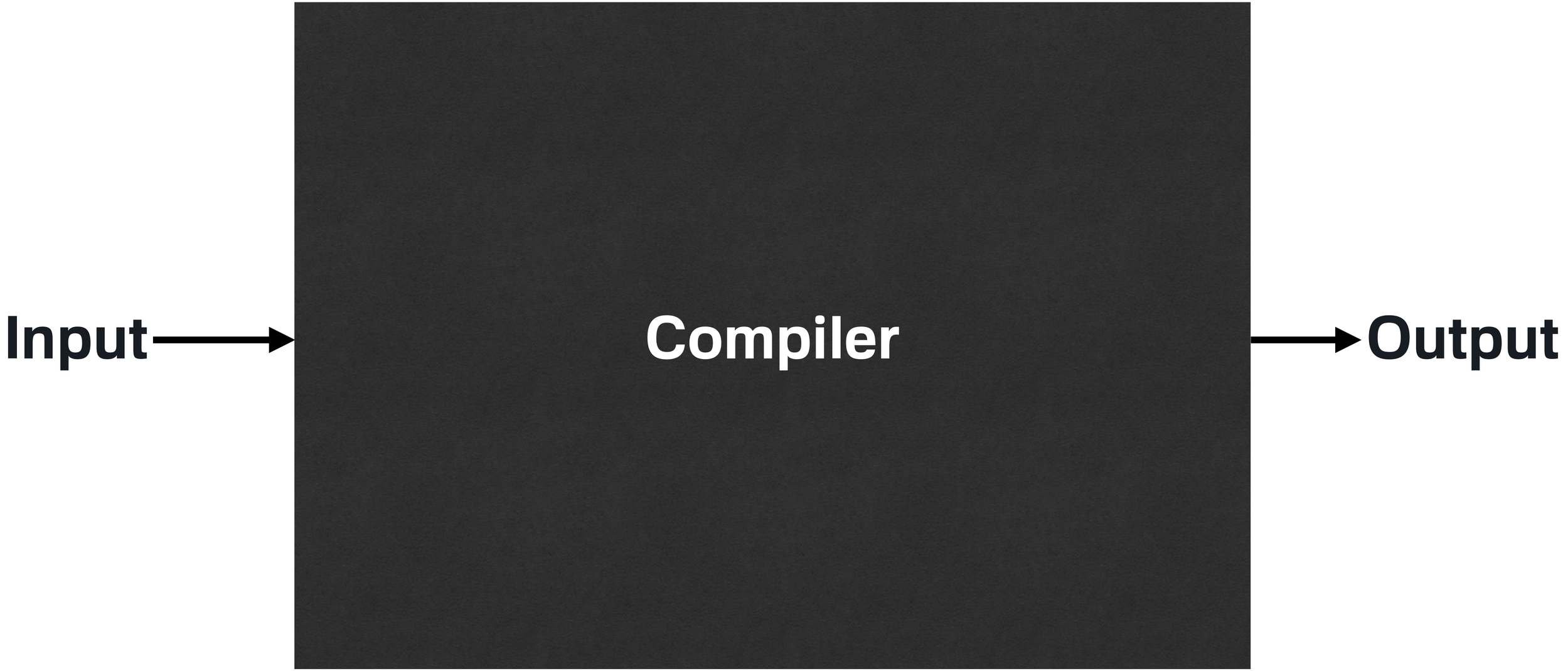
"Syntax Tree" (AST) 抽象構文木

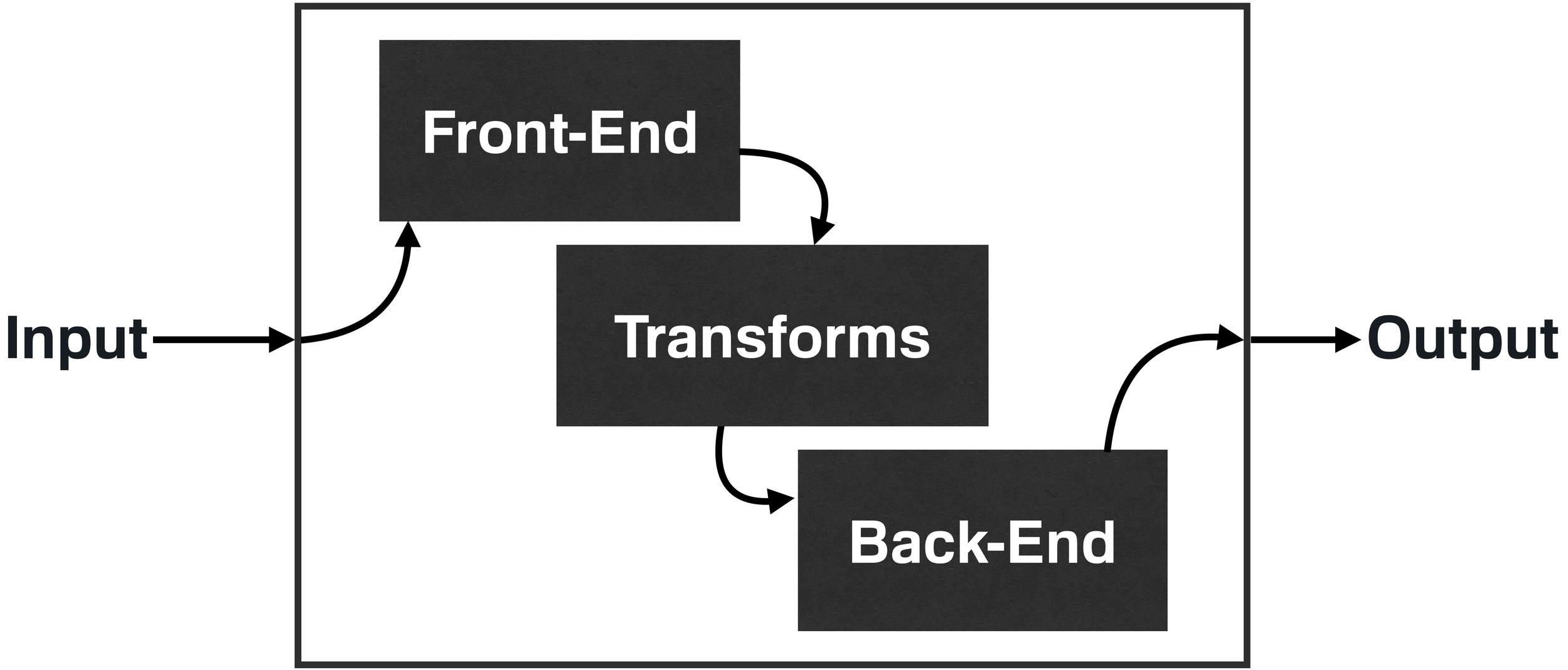


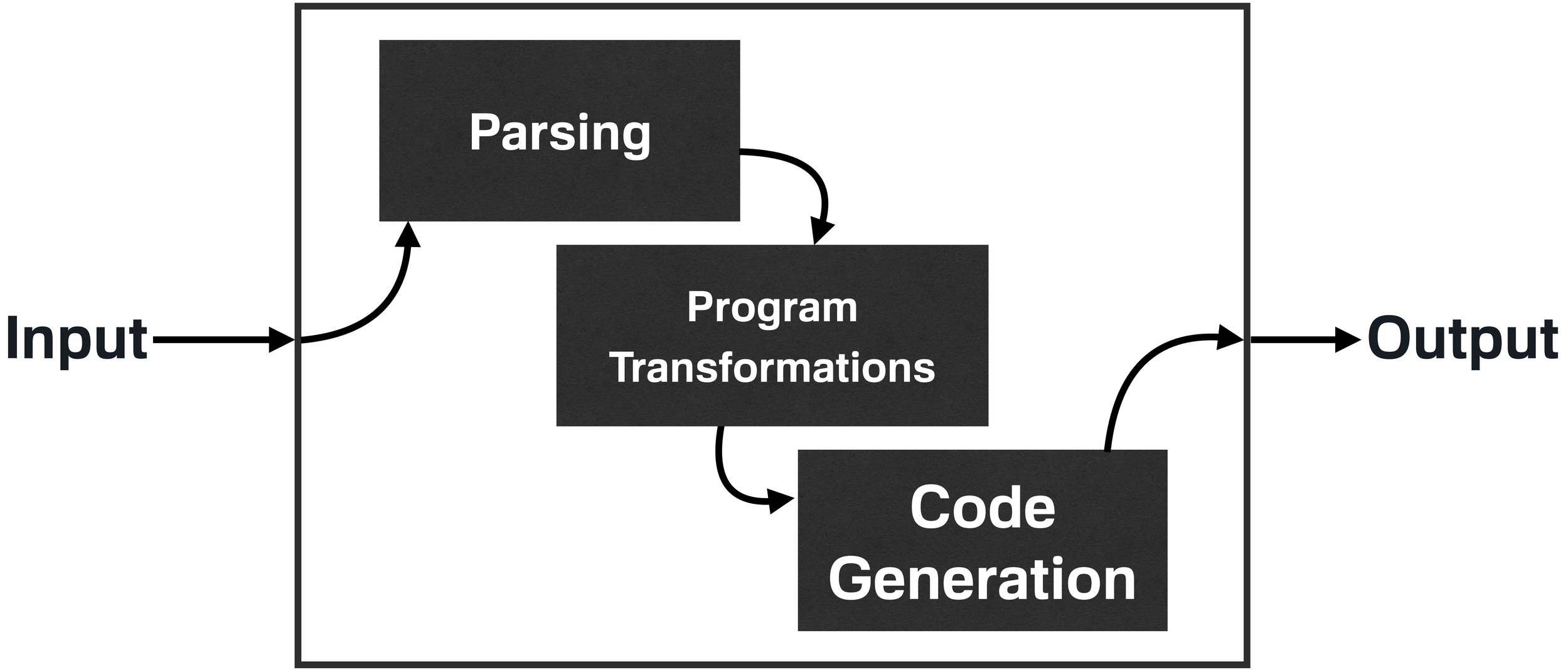

body
div
div
p
hr
text
img
hr

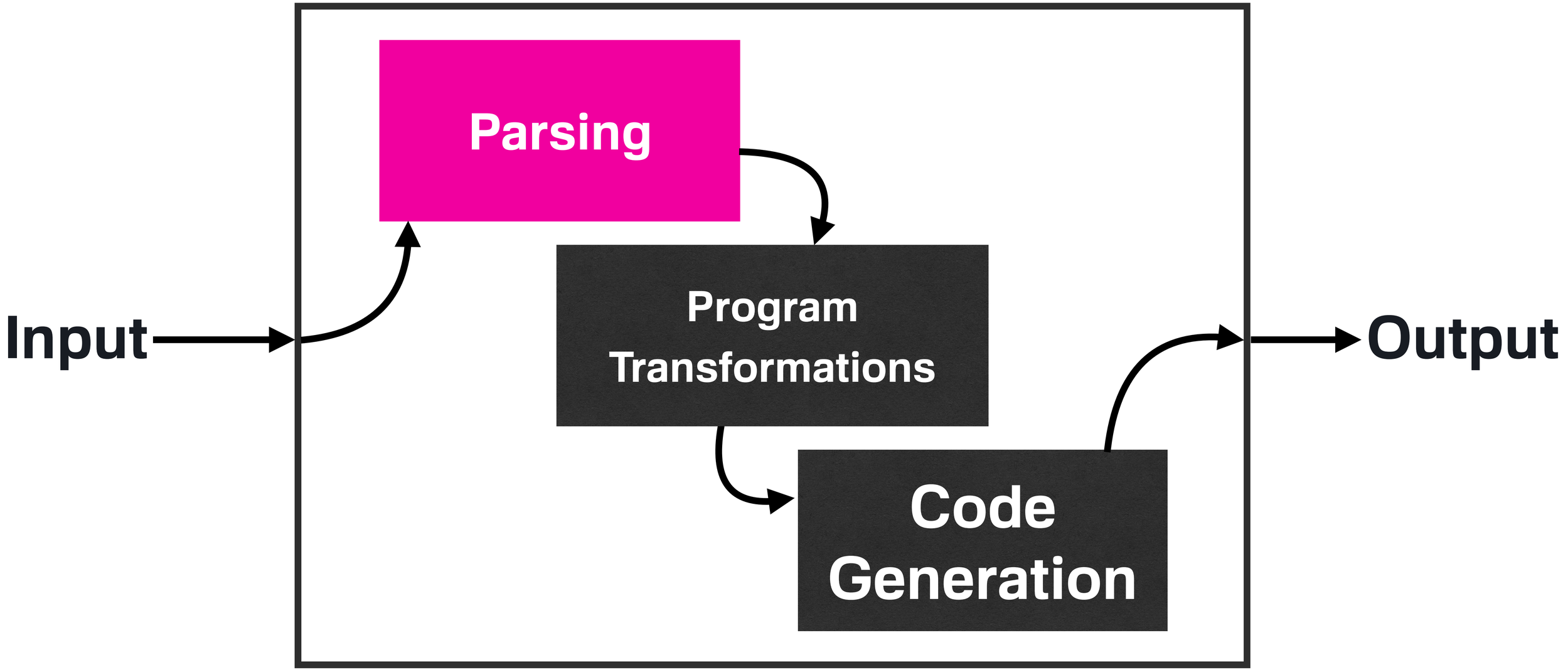

react
prop-types
loose-envify
loose-envify
object-assign
js-tokens


Parts of a
Compiler

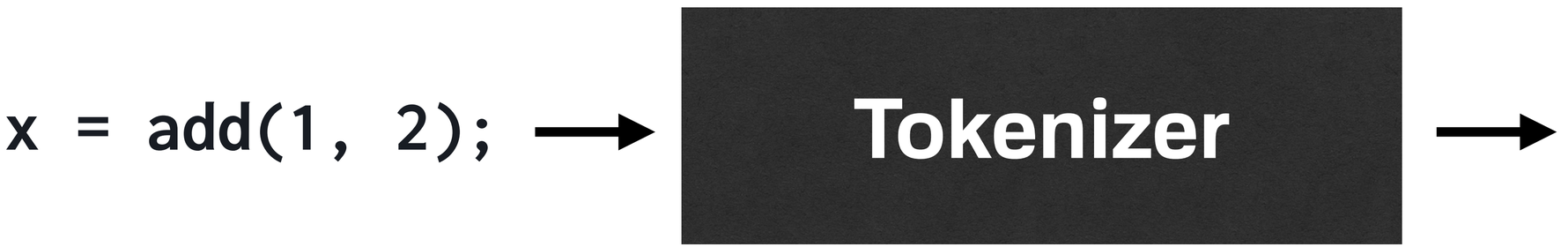

Input
Compiler
Output


Front-End
Transforms
Back-End
Input
Output

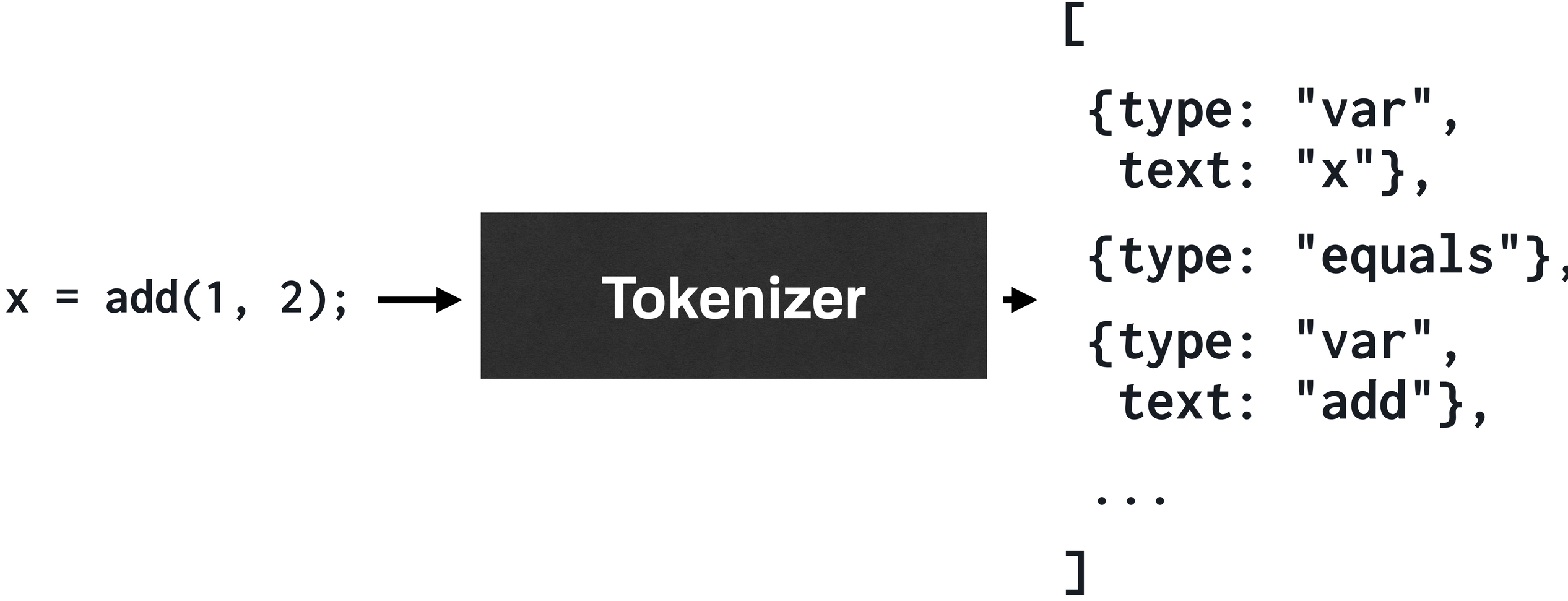

Parsing
Program
Transformations
Code
Generation
Input
Output


Parsing
Program Transformations
Code Generation
Input
Output


Program
Text
Tokenizer
Tokens

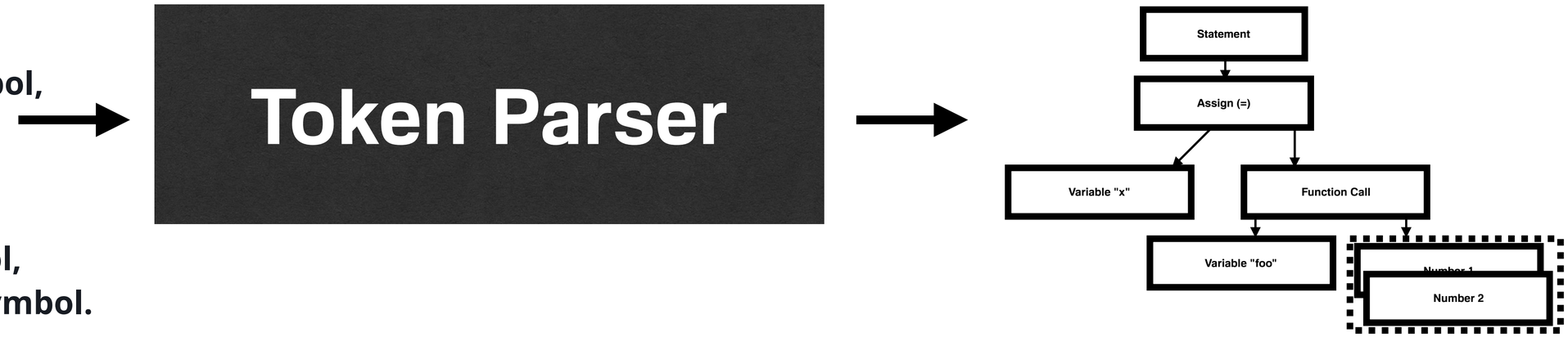


- A variable called "x",
- An Equals symbol,
- A variable called "add",
- An Open Parenthesis symbol,
- A literal number "1",
- A Comma symbol,
- A literal number "2",
- A Close Parenthesis symbol,
- A statement terminator symbol.

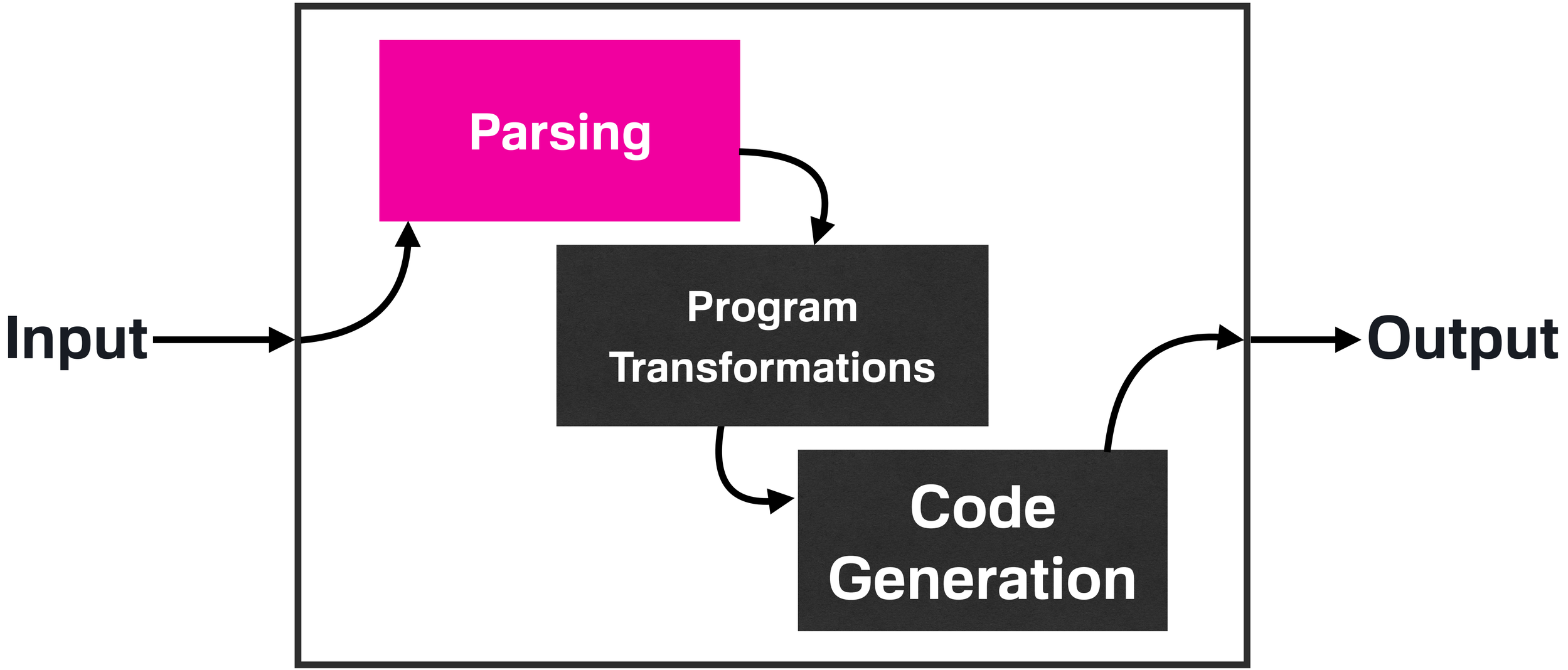

x = add(1, 2);
Tokenizer
[
{type: "var",
text: "x"},
{type: "equals"},
{type: "var",
text: "add"},
...
]

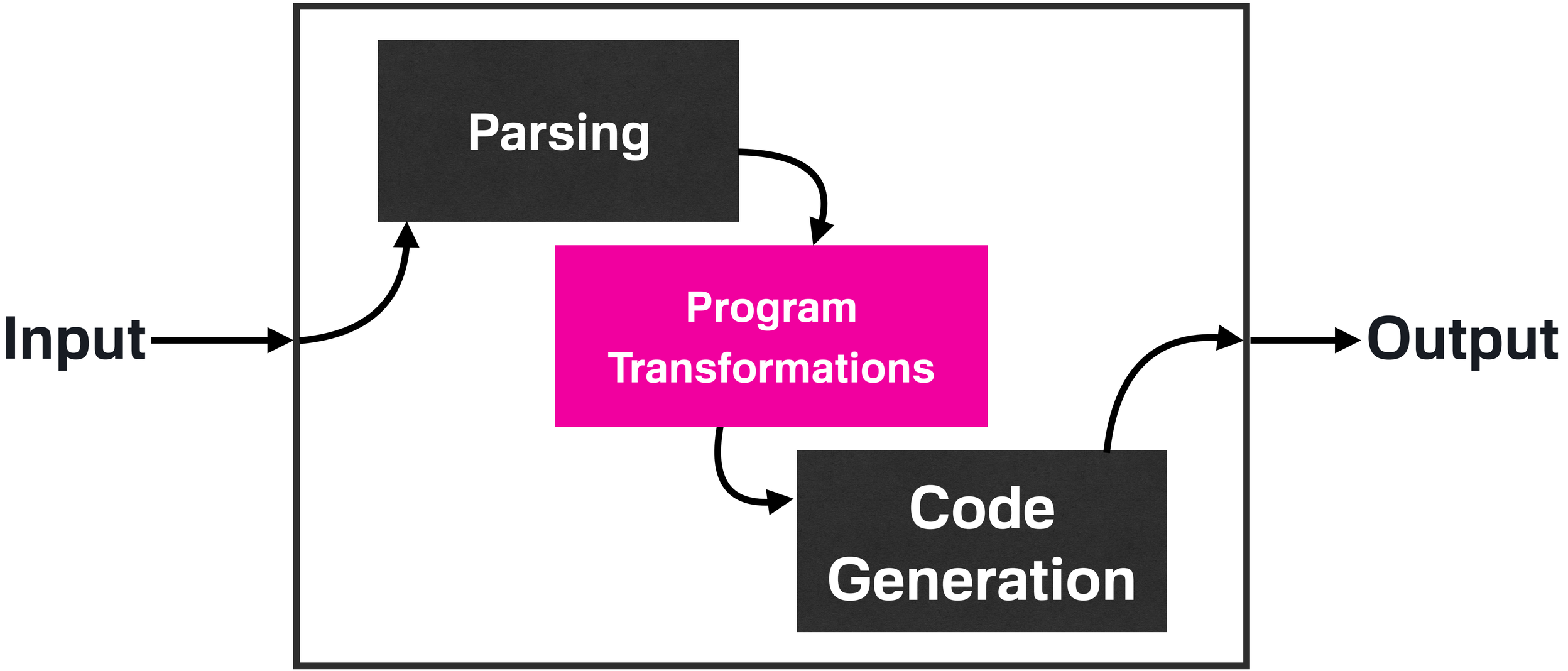

Tokens
Token Parser
Abstract
Syntax Tree

- A variable called "x",
- An Equals symbol,
- A variable called "add",
- An Open Parenthesis symbol,
- A literal number "1",
- A Comma symbol,
- A literal number "2",
- A Close Parenthesis symbol,
- A statement terminator symbol.




[
{type: "var",
text: "x"},
{type: "equals"},
{type: "var",
text: "add"},
...
]
Token Parser
{
type: "statement",
body: {
type: "assign",
identifier: "foo",
expression: {
...
}
}
}

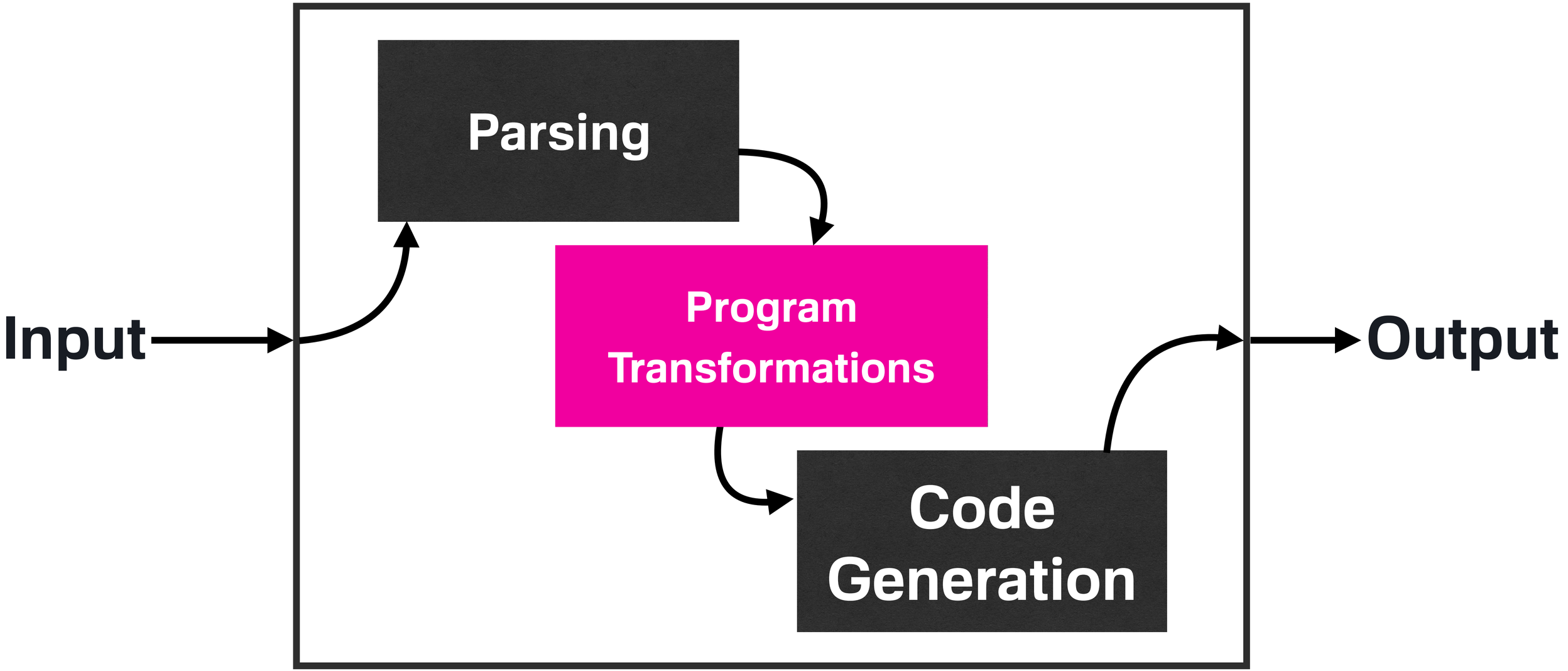

Parsing
Program Transformations
Code Generation
Input
Output

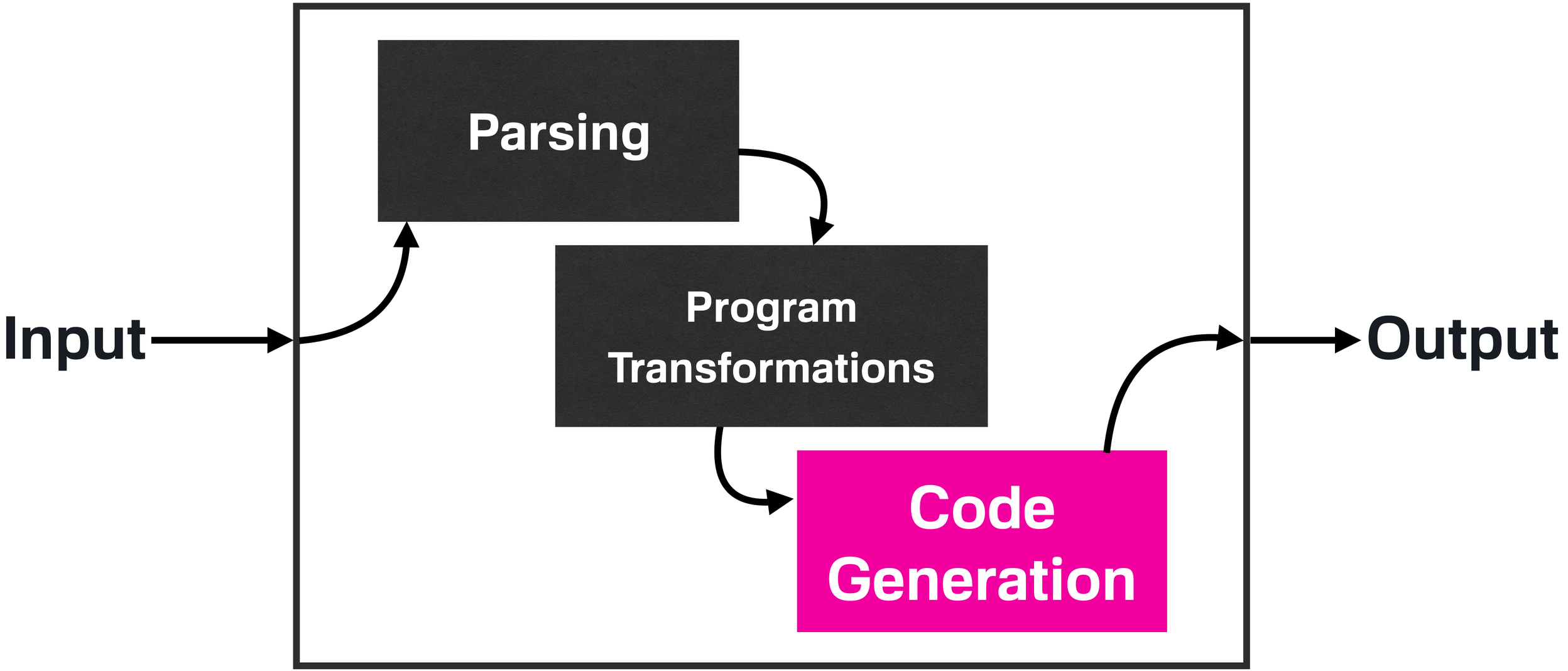

Parsing
Program
Transformations
Code
Generation
Input
Output

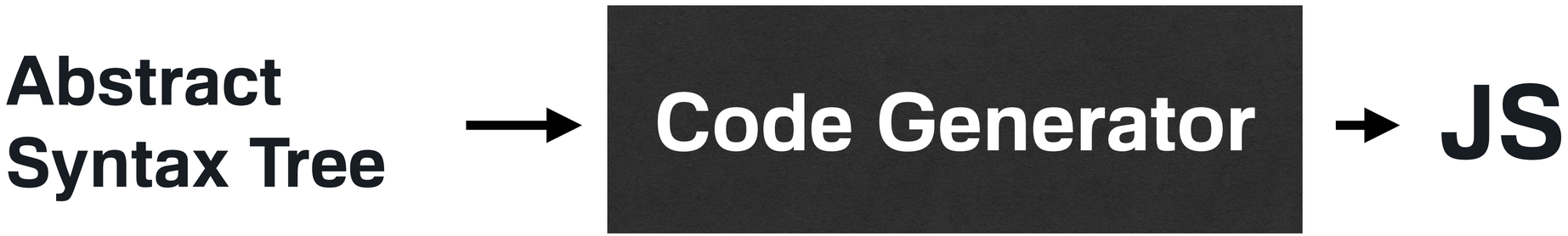

a + 5 + 5
Optimiser
a + 10

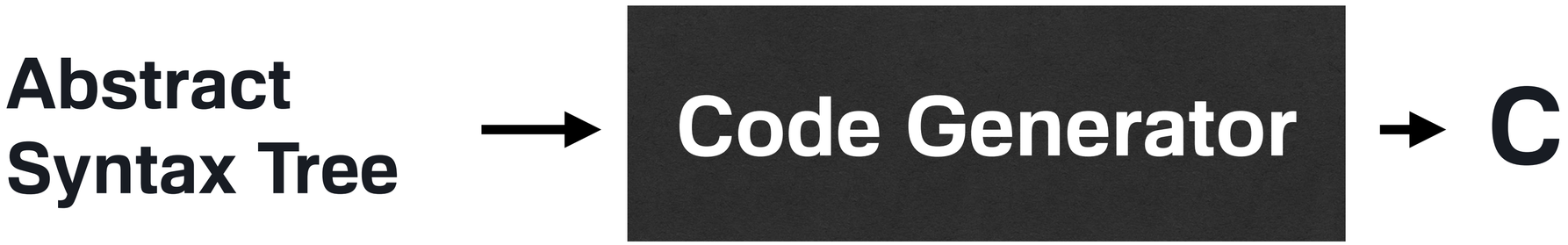

+
a
+
5
5
Optimiser
+
a
10

(or a big error message)

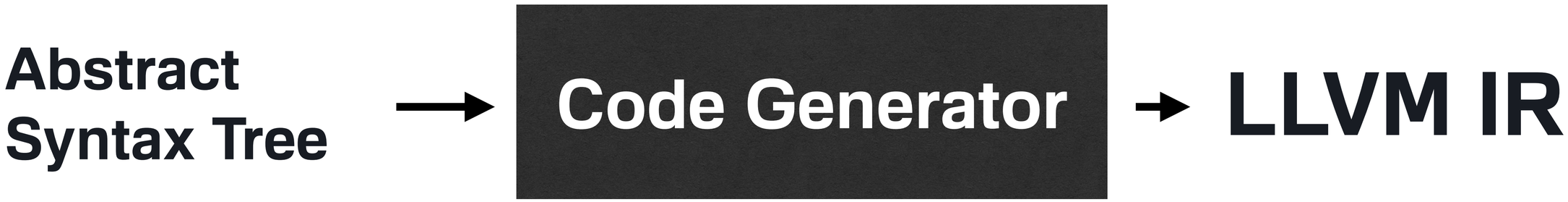

Parsing
Program
Transformations
Code
Generation
Input
Output


Parsing
Program Transformations
Code Generation
Input
Output


Abstract
Syntax Tree
Code Generator
JS


Abstract
Syntax Tree
Code Generator
C


Abstract
Syntax Tree
Code Generator
ELF Binary


Abstract
Syntax Tree
Code Generator
LLVM IR

"Runtime"

- In-built core or "primitive" library functions, eg. `+`, `process.env`, etc.
- A run-loop or other thing that makes the code "go".

Why

Some Things are Actual Compilers

Sass

TypeScript

Elm's Compiler

(elm, née elm-make)

Some Things Look Like Compilers

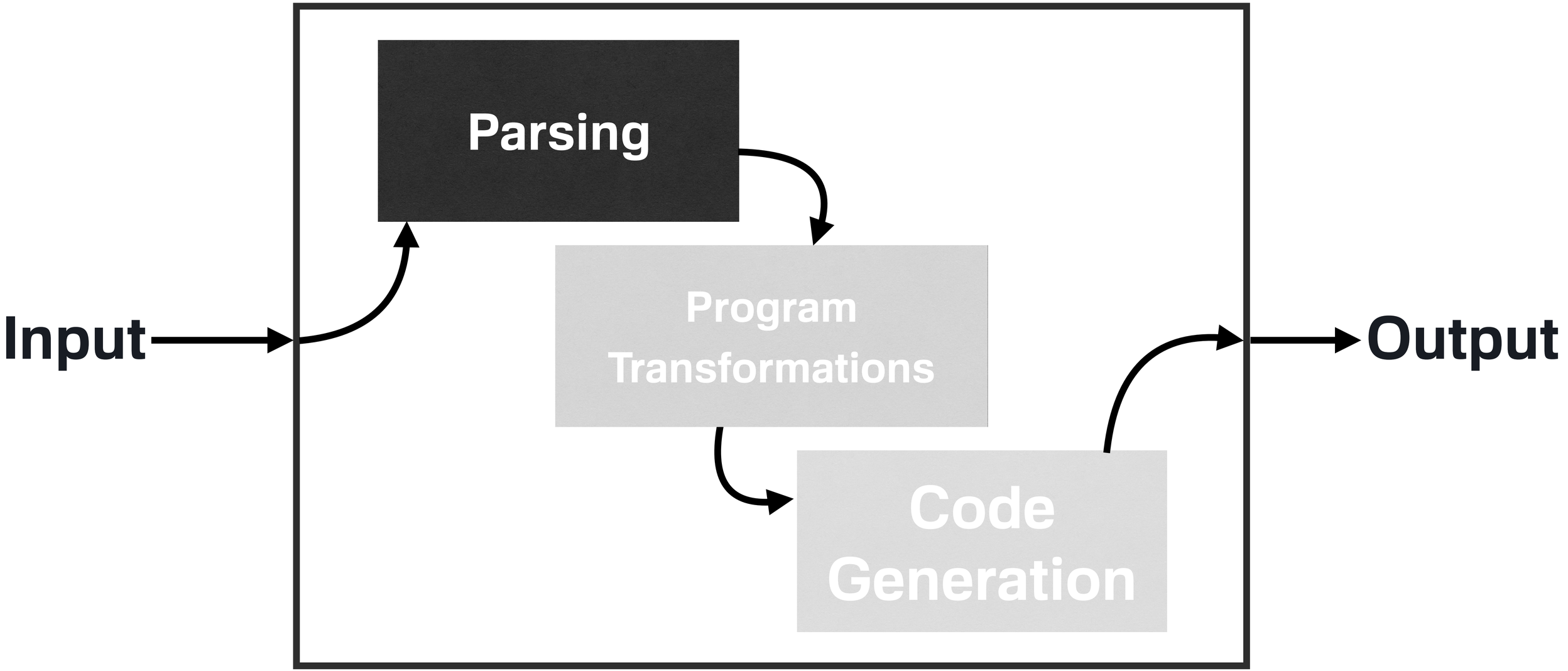
File Converters

styled-components

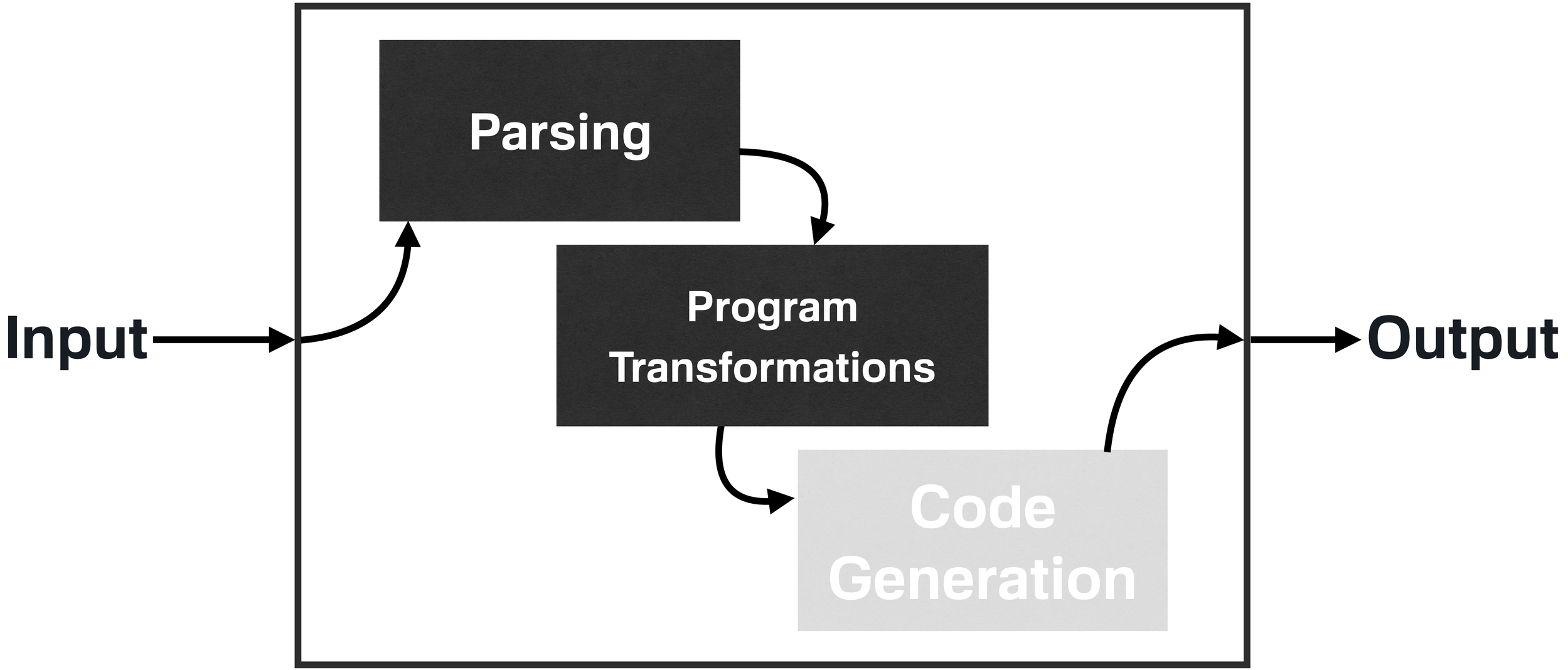
Facebook's Prepack

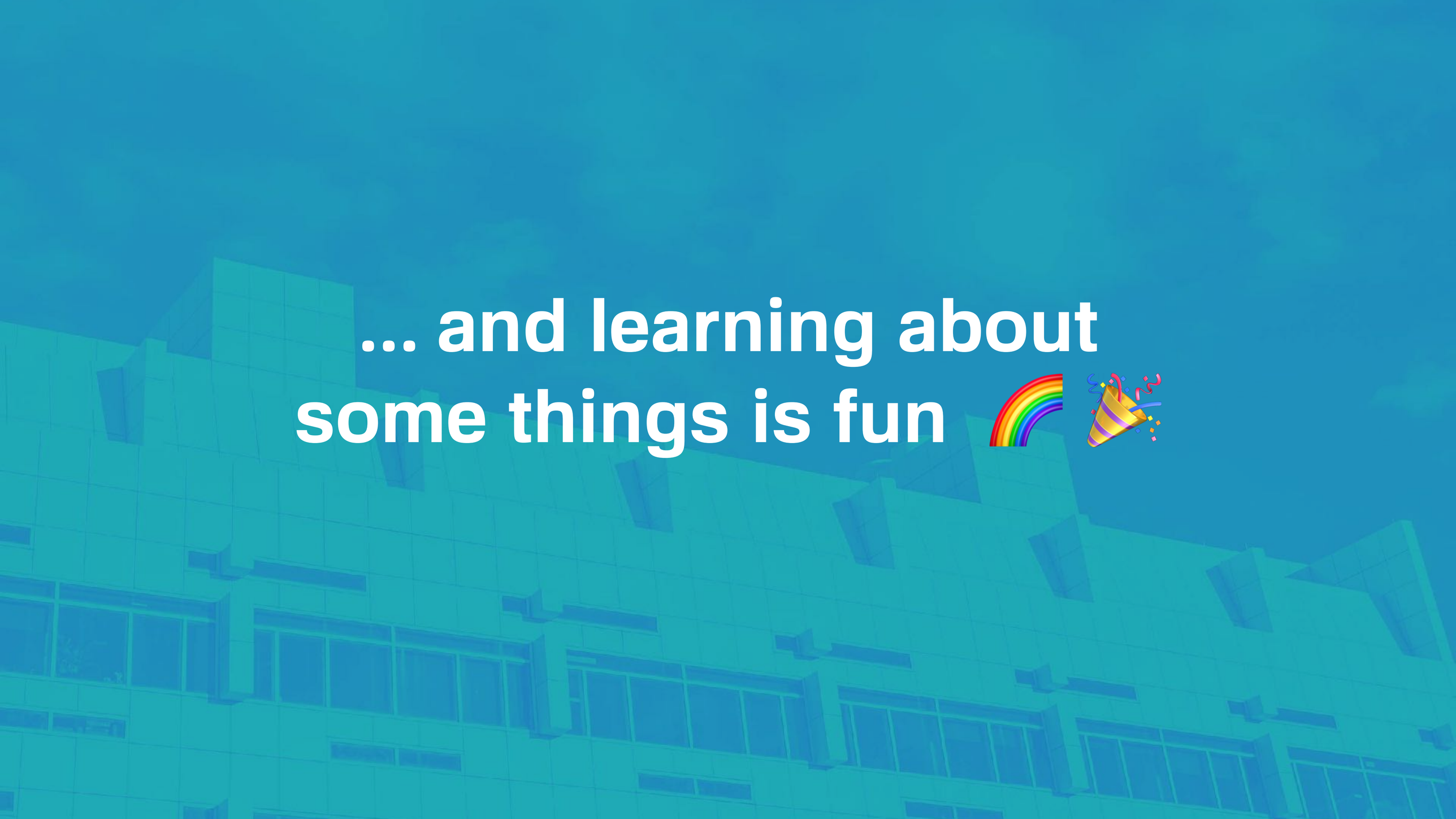
Some Things are Pieces of Compilers

Input Validation (eg. from JSON)


Input
Parsing
Program
Transformations
Code
Generation
Output

Facebook's Flow


Parsing
Program Transformations
Code Generation
Input
Output


... and learning about
some things is fun 🌈🎉

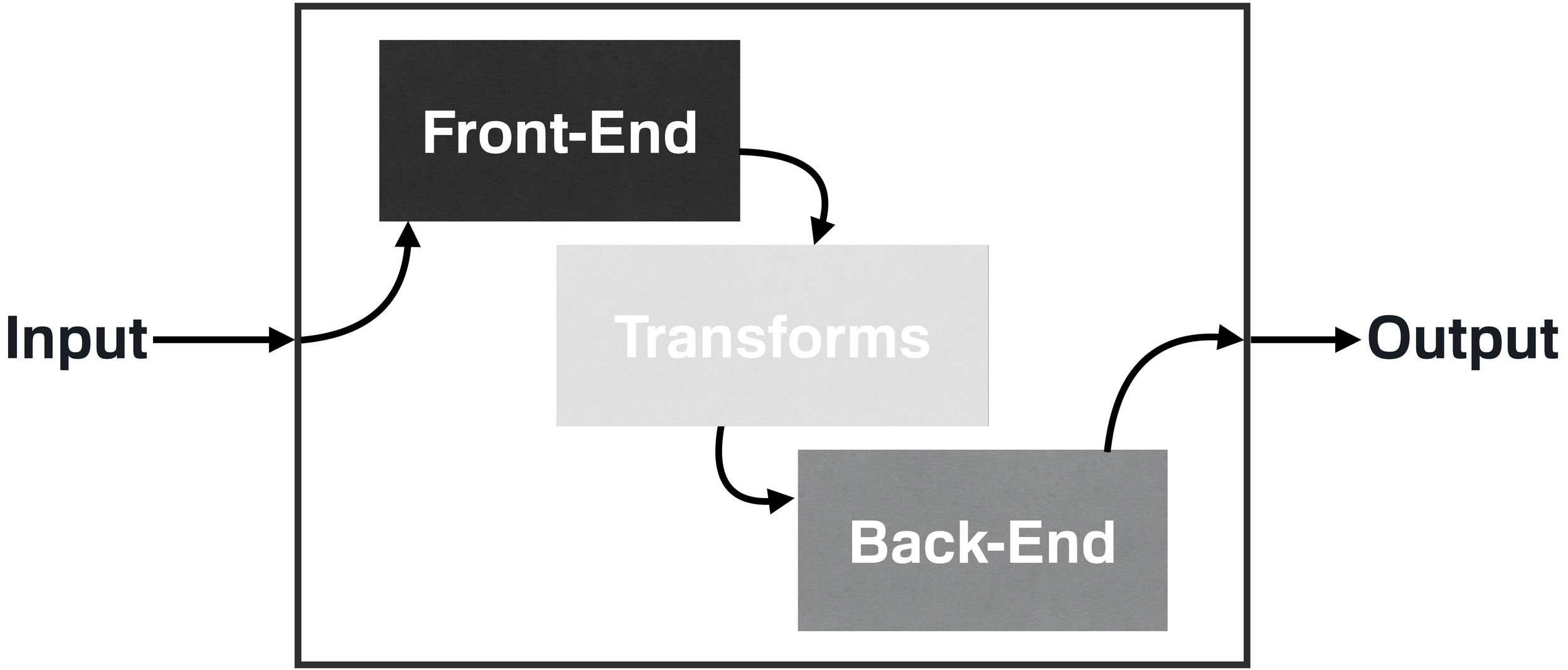
Workshop:

A Tiny Language

```
let x = 1;
add(x, 1);
```

```
let x = 1;
add(x, add(1, 2));
```

```
let foo = 7;
let x = 1;
add(foo, add(1, 2));
```


Front-End
Transforms
Back-End
Input
Output

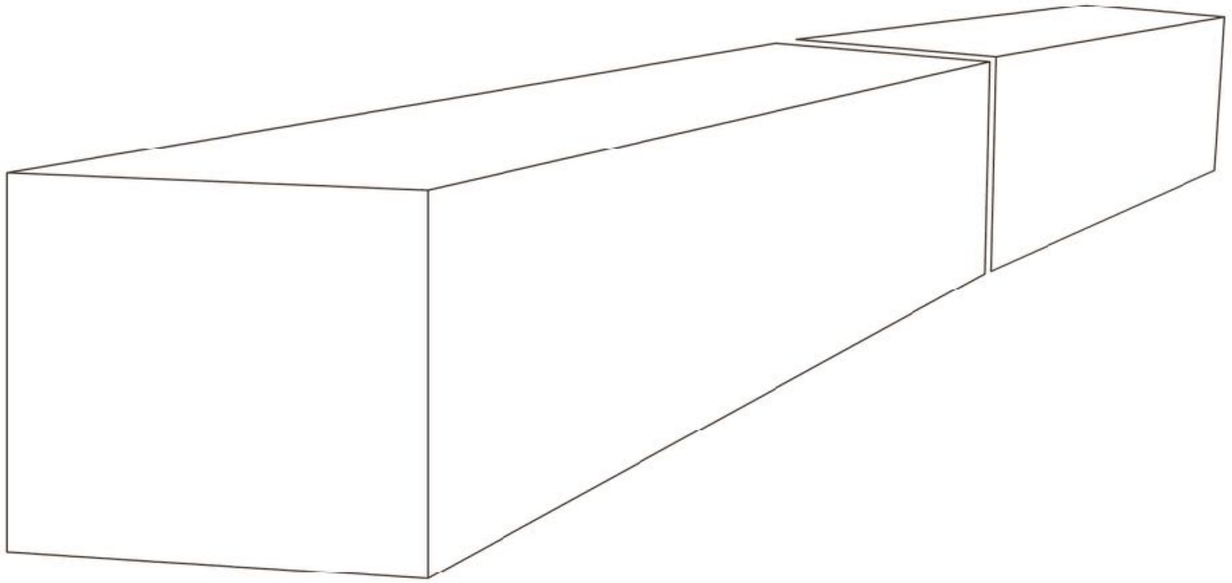
1　2　??　4　5

What's the missing number???

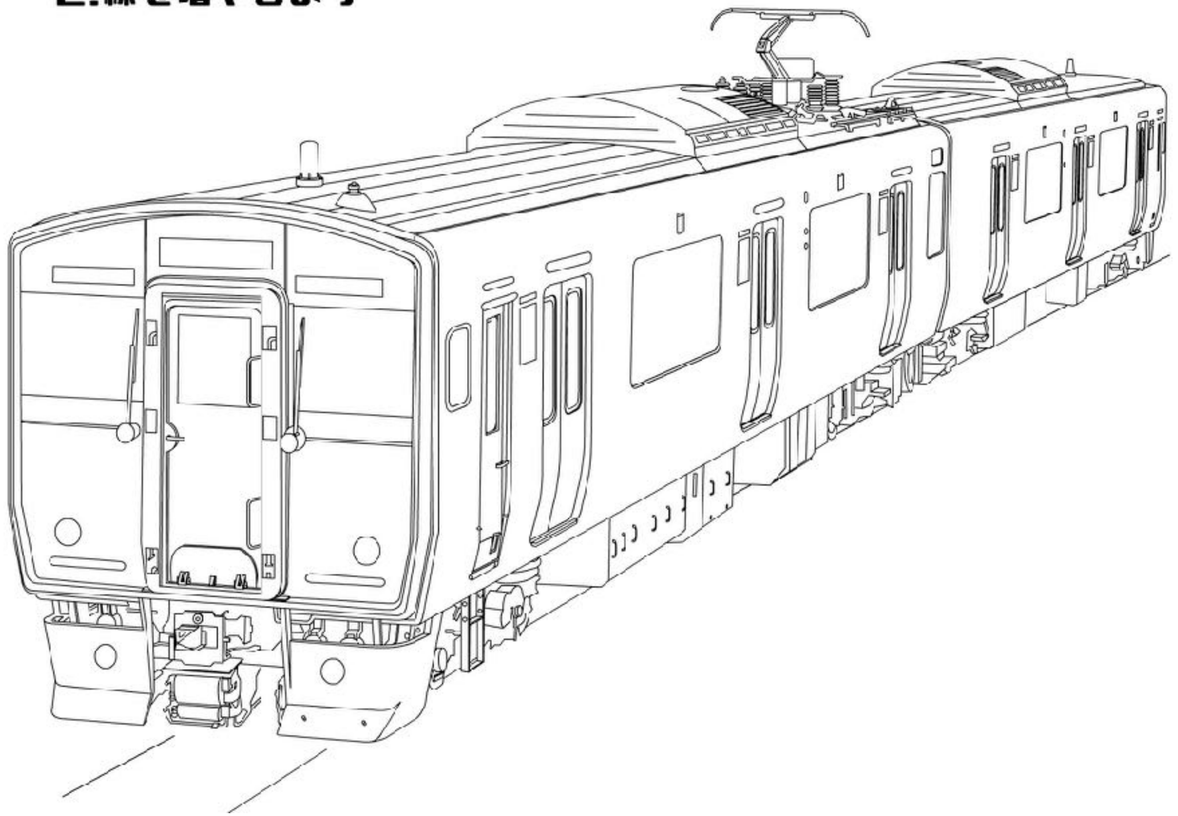
数を推測して下さい！！

電車の描きかた

1.箱を2つかきます



2.線を増やします



— https://twitter.com/calmicco/status/1006419384665980928

Building a Small Compiler in JavaScript

Rob Howard
@damncabbage
http://robhoward.id.au

Things To Read

- *Crafting Interpreters*, Bob Nystrom
  http://www.craftinginterpreters.com/contents.html
  (Free online)
- *Engineering a Compiler*, Cooper & Torczon
  https://www.amazon.com/dp/012088478X
- *Modern Compiler Design*, Grune et al
  https://dickgrune.com/Books/MCD_2nd_Edition/