@babel/how-to

NICOLÒ RIBAUDO

Babel team

@NicoloRibaudo

nicolo.ribaudo@gmail.com

@nicolo-ribaudo

https://opencollective.com/babel

@babeljs

https://babeljs.io

@babel

What is Babel?

What is Babel?

Babel is a JavaScript compiler

```
const square = n => n ** 2;
```

```
[square]
   StackCheck
   Ldar a0
   ExpSmi [2], [0]
   Return
```

```
const square = n => n ** 2;
```

It’s a JavaScript to JavaScript compiler

```
const square = n => n ** 2;
```

```
var square = function (n) {
  return Math.pow(n, 2);
};
```

Compilers' data structure:

Abstract Syntax Tree (AST)

A naive approach to compilation: *Regular Expressions*

```
n => n ** 2
```

A naive approach to compilation: *Regular Expressions*

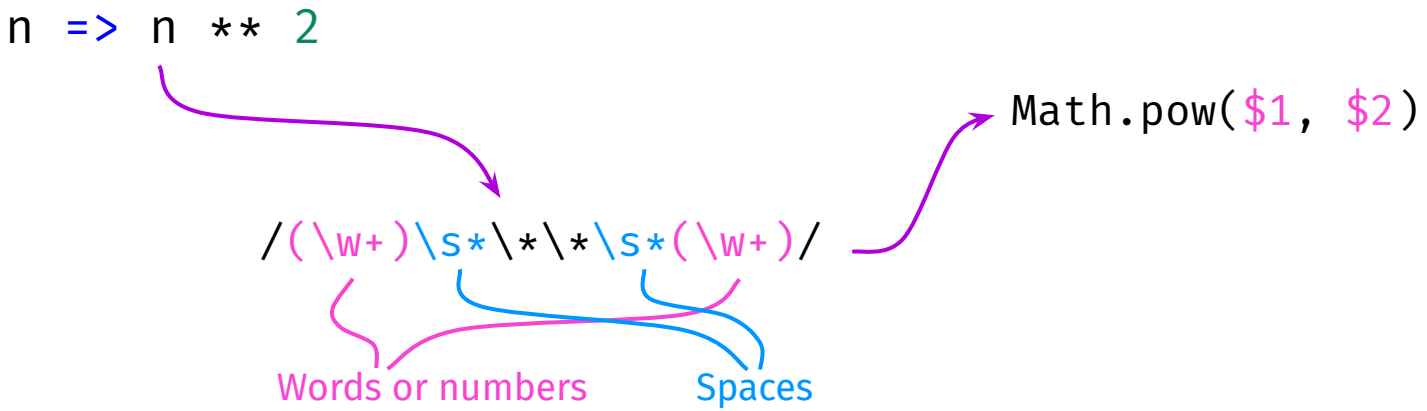
```
n => n ** 2
```

```
/(\w+)\s*\*\*\s*(\w+)/
```

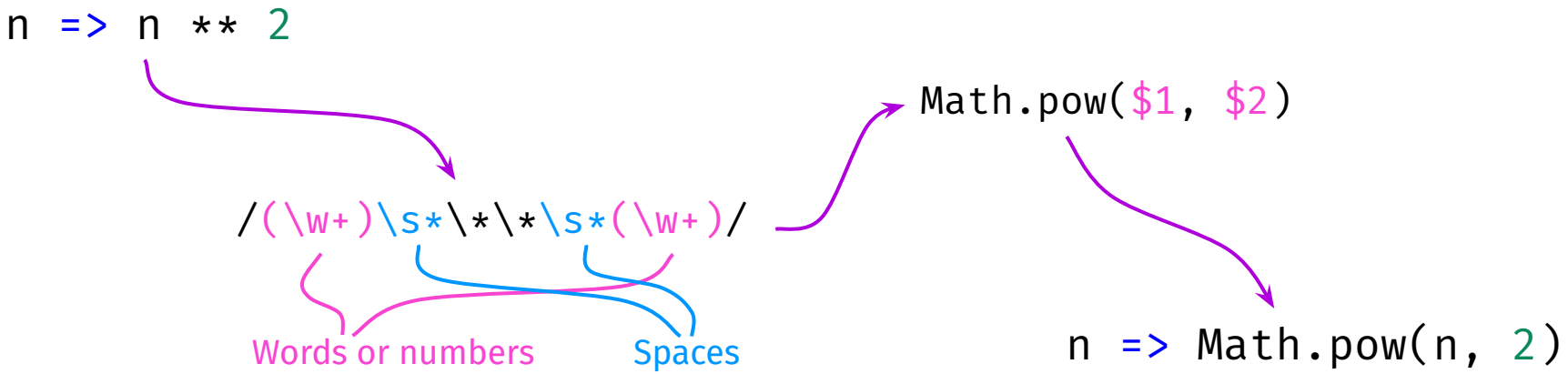
Words or numbers

Spaces

A naive approach to compilation: *Regular Expressions*



A naive approach to compilation: *Regular Expressions*



A naive approach to compilation: *Regular Expressions*

A naive approach to compilation: *Regular Expressions*

COMPLEXITY

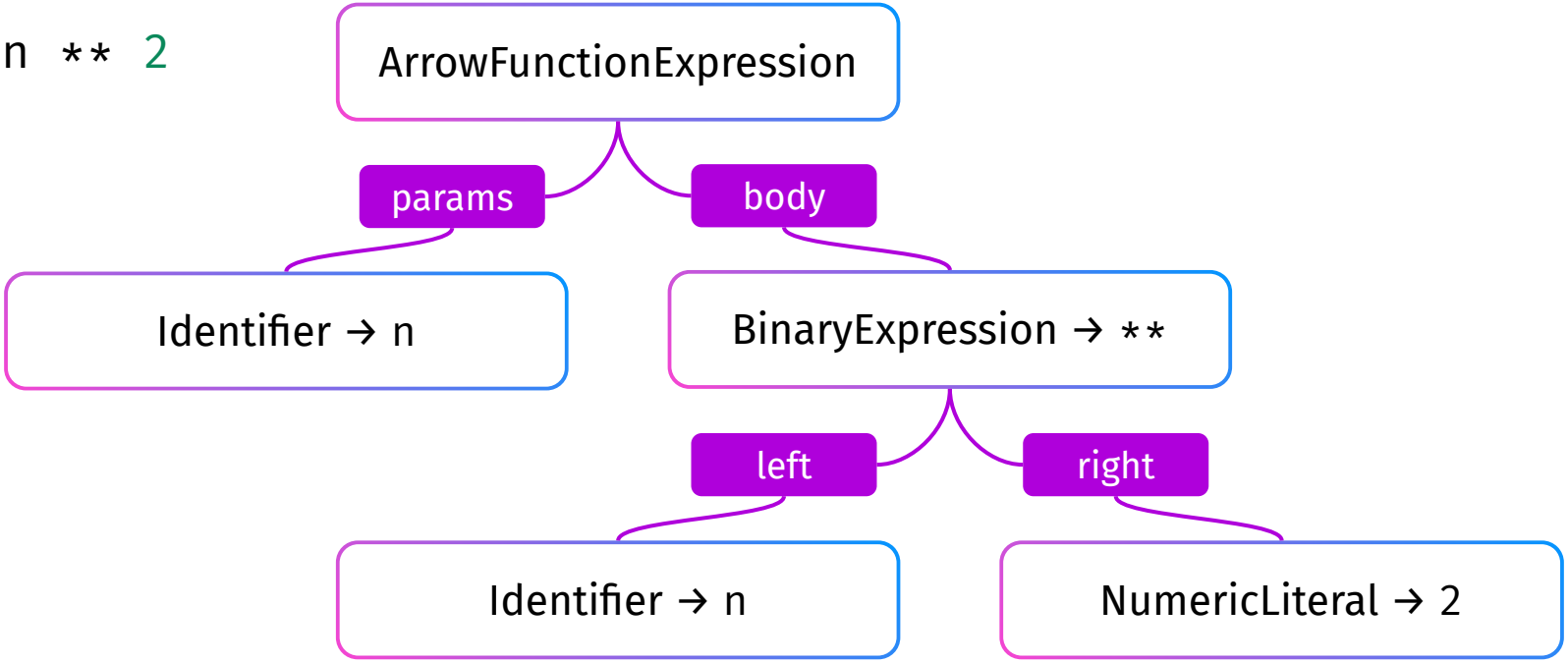
```
fn(a) ** (2 ** 3)
```

INACCURACY

```
"8" !== "2 ** 3"
```

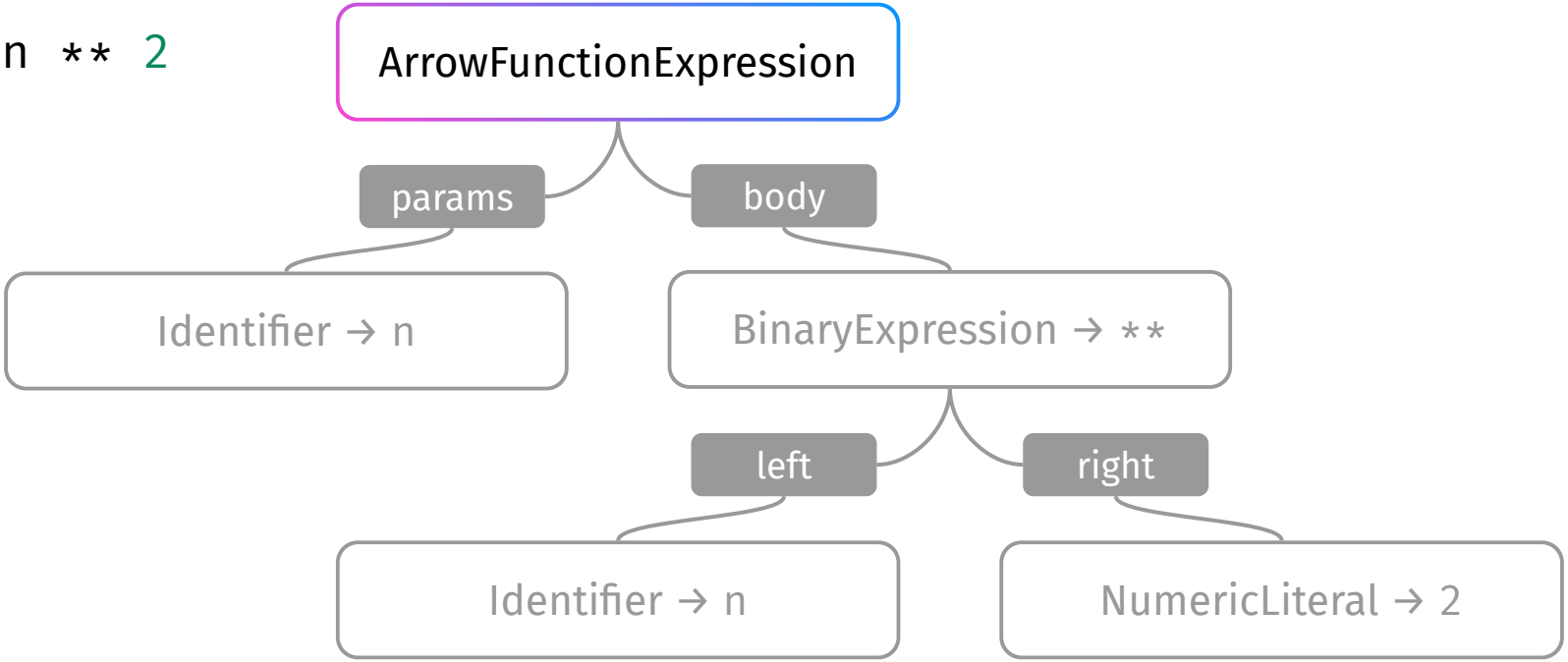
Abstract Syntax Tree

```
n => n ** 2
```



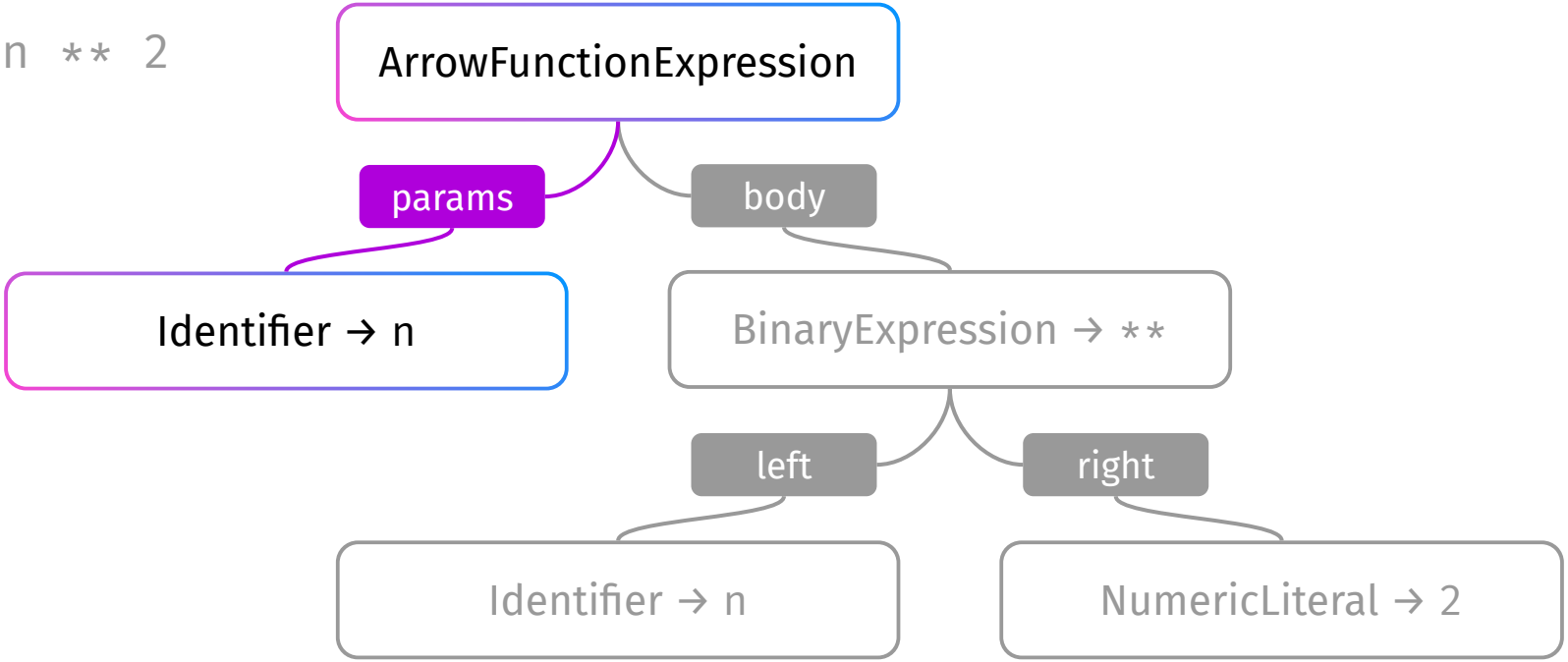
Abstract Syntax Tree

```
n => n ** 2
```



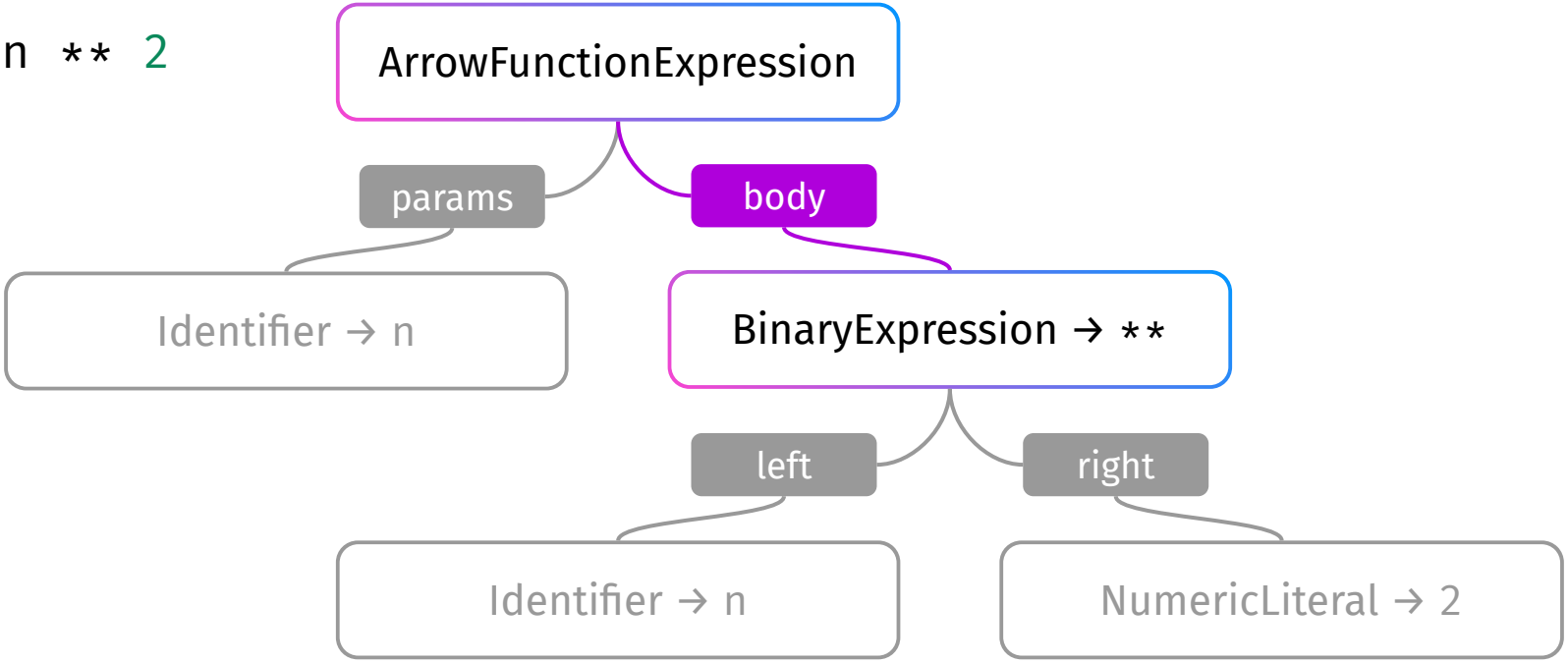
Abstract Syntax Tree

```
n => n ** 2
```



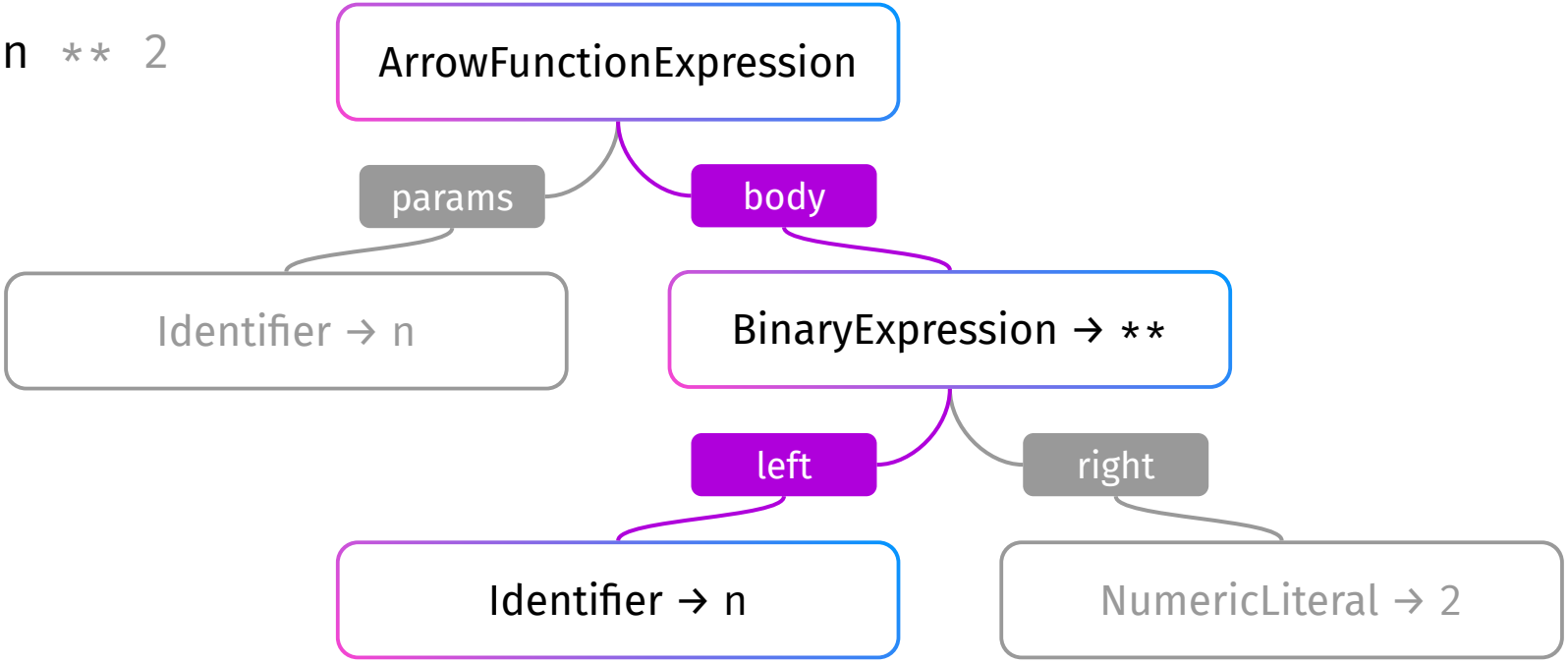
Abstract Syntax Tree

```
n => n ** 2
```



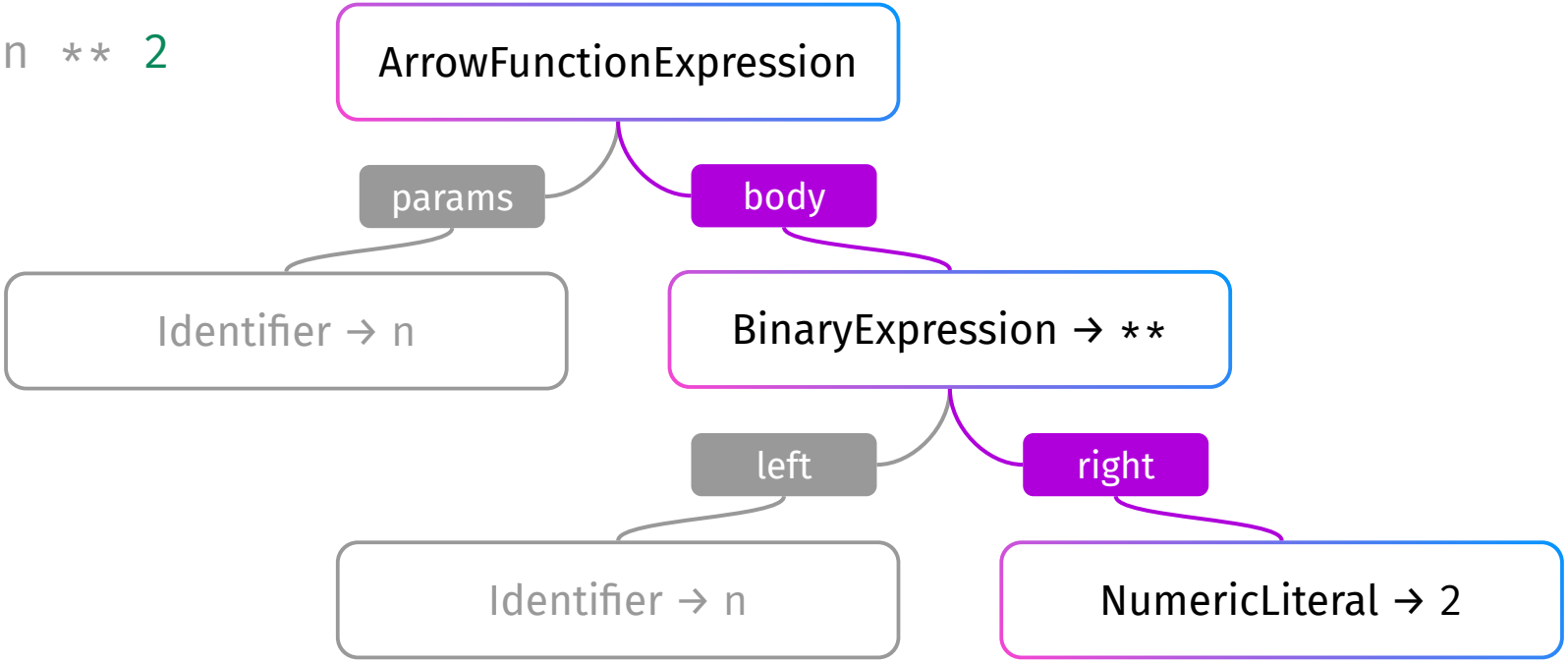
Abstract Syntax Tree

```
n => n ** 2
```

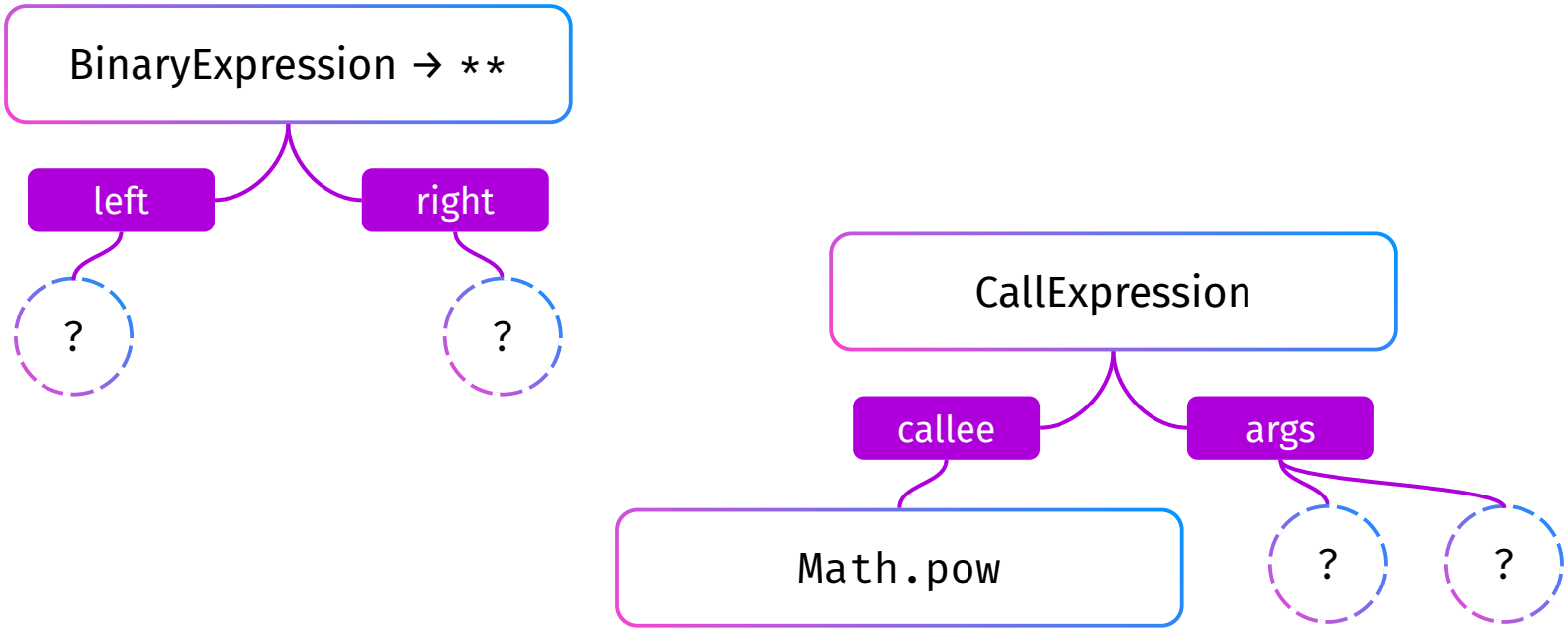


Abstract Syntax Tree

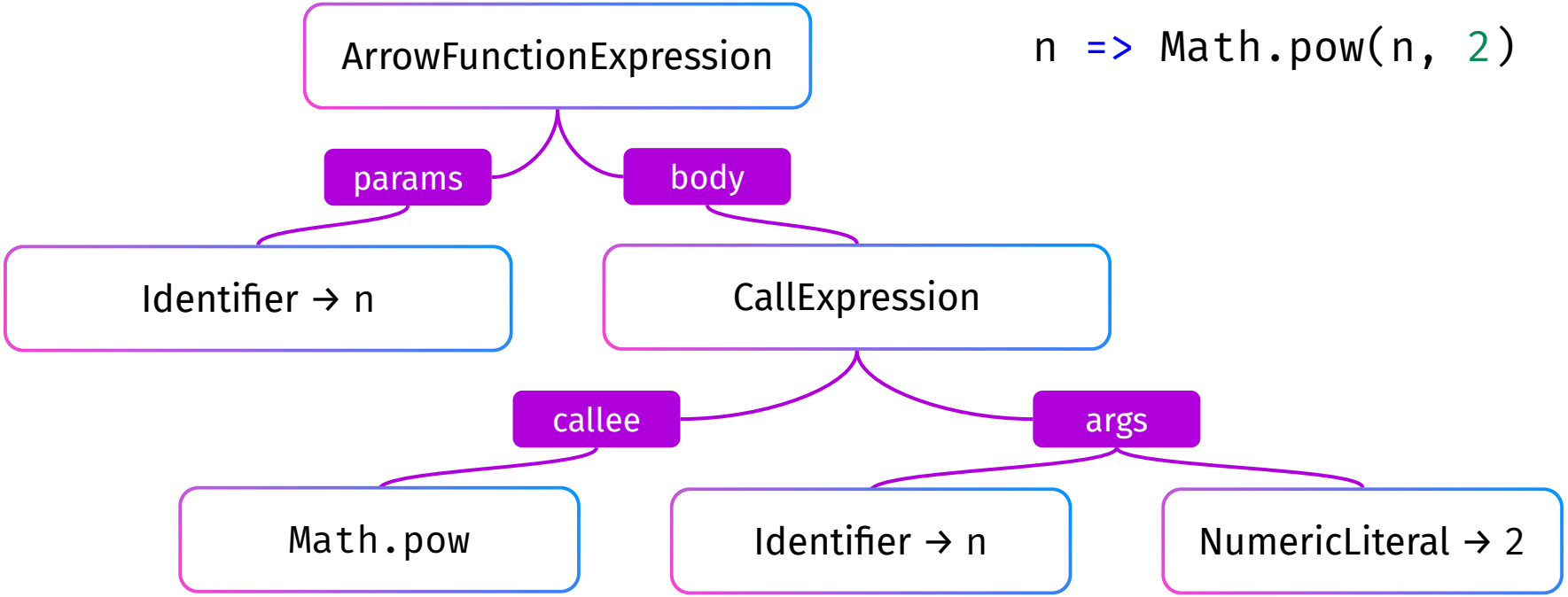
```
n => n ** 2
```



AST node replacement



AST node replacement




```
n => Math.pow(n, 2)
```

AST node replacement

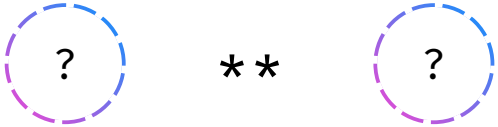
AST node replacement

~~COMPLEXITY~~

```
fn(a) ** (2 ** 3)
```

~~INACCURACY~~

```
"8" !== "2 ** 3"
```



StringLiteral → `"2 ** 3"`

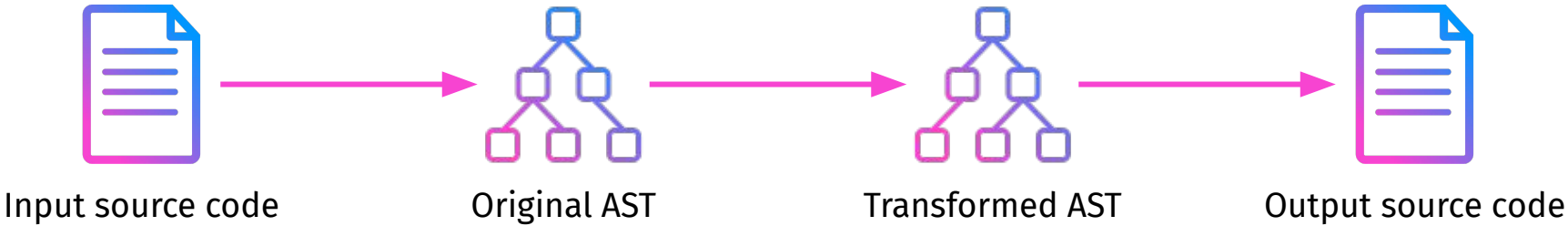
Babel's AST

```
{
  "type": "BinaryExpression",
  "operator": "**",
  "left": {
    "type": "Identifier",
    "name": "n"
  },
  "right": {
    "type": "NumericLiteral",
    "value": 2
  }
}
```

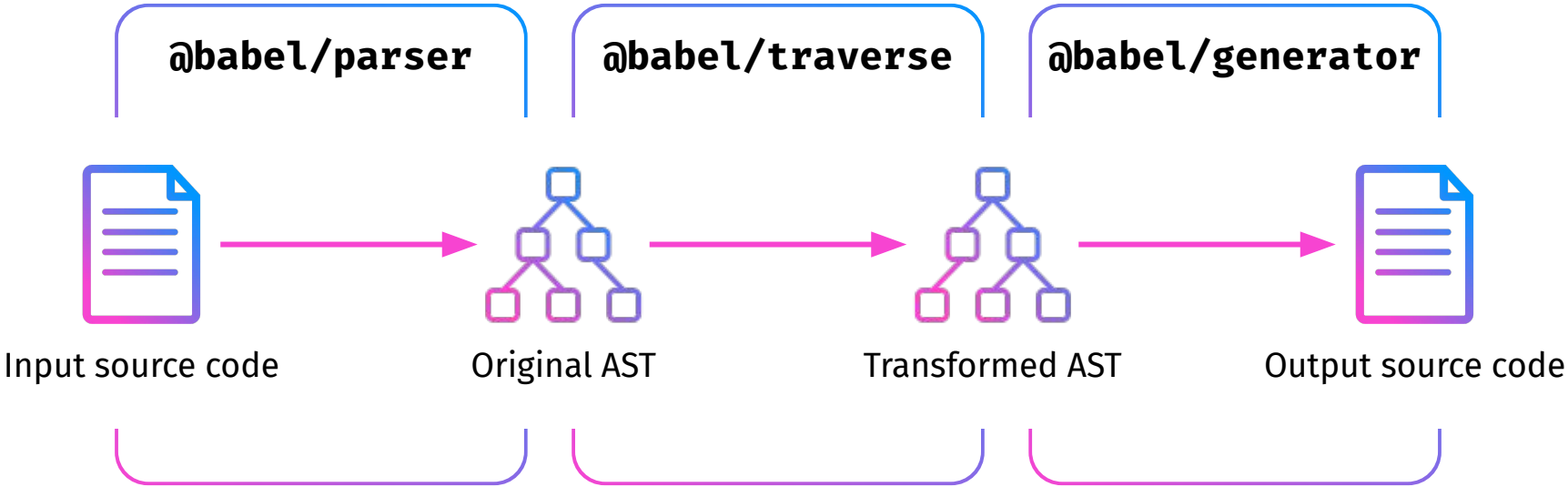
```
{
  "type": "CallExpression",
  "callee": { /* ... */ },
  "arguments": [{
    "type": "Identifier",
    "name": "n"
  }, {
    "type": "NumericLiteral",
    "value": 2
  }]
}
```

A look inside Babel

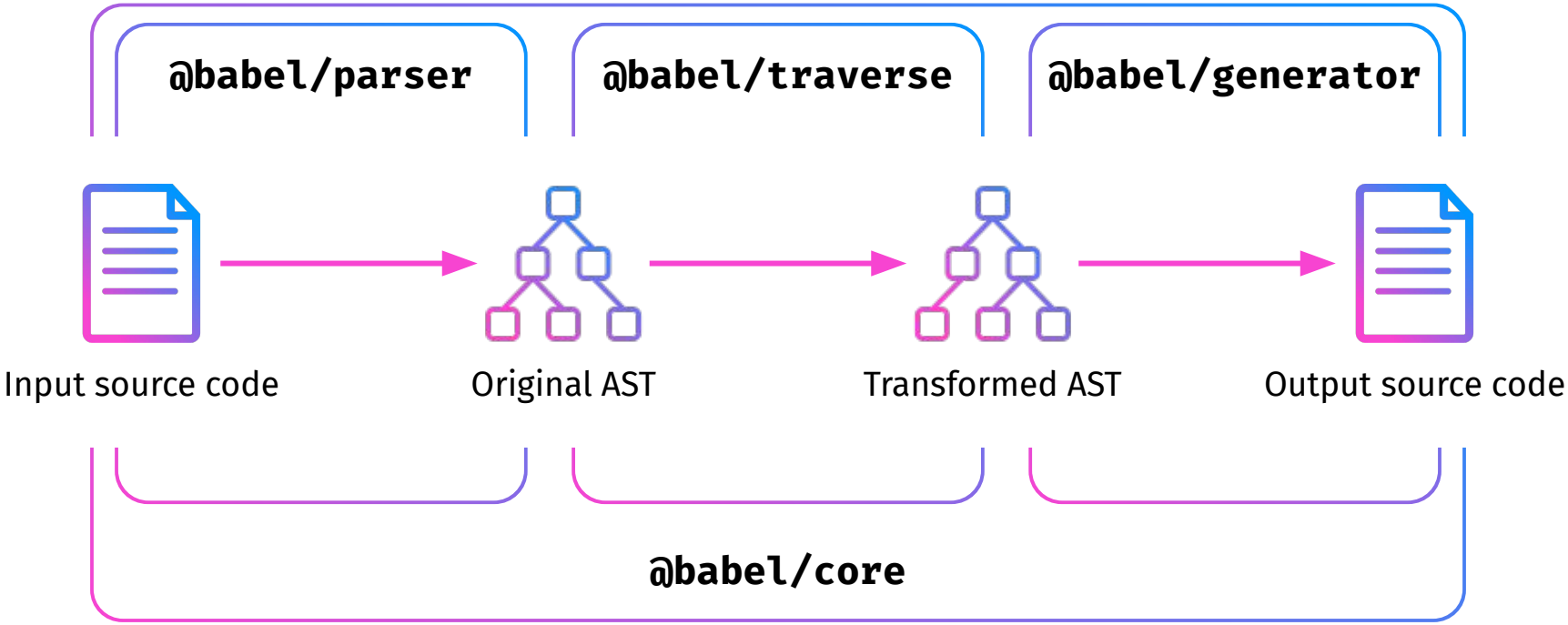
A look inside Babel



A look inside Babel

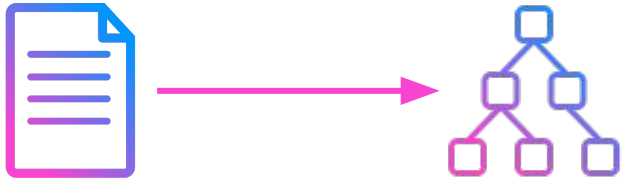


A look inside Babel



A look inside Babel:

`@babel/parser`



1. Lexical analysis

Transform the input source code into a list of ***tokens***

```
var a = 7;
```

1. Lexical analysis

Transform the input source code into a list of ***tokens***

```
var a = 7;
```

1. Keyword `var`

1. Lexical analysis

Transform the input source code into a list of ***tokens***

```
var a = 7;
```

1. Keyword `var`
2. Identifier `a`

1. Lexical analysis

Transform the input source code into a list of ***tokens***

```
var a = 7;
```

1.	Keyword	`var`
2.	Identifier	`a`
3.	Punctuator	`=`

1. Lexical analysis

Transform the input source code into a list of ***tokens***

```
var a = 7;
```

1.	Keyword	`var`
2.	Identifier	`a`
3.	Punctuator	`=`
4.	Literal	`7`

1. Lexical analysis

Transform the input source code into a list of ***tokens***

```
var a = 7;
```

1.	Keyword	`var`
2.	Identifier	`a`
3.	Punctuator	`=`
4.	Literal	`7`
5.	Punctuator	`;`

1. Lexical analysis

Report errors about invalid literals or characters

1. Lexical analysis

Report errors about invalid literals or characters

Unterminated comment

```
/* var a = 7;
```

1. Lexical analysis

Report errors about invalid literals or characters

Unterminated comment

```
/* var a = 7;
```

Unexpected character '°'

```
var a = 7°;
```

1. Lexical analysis

Report errors about invalid literals or characters

Unterminated comment	`/* var a = 7;`
Unexpected character '°'	`var a = 7°;`
Expected number in radix 2	`var a = 0b20;`

2. Syntax analysis

Transform the list of tokens into an ***AST***

```
var a = 7;
```

2. Syntax analysis

Transform the list of tokens into an ***AST***

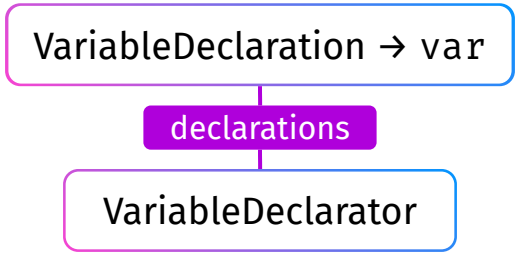
```
var a = 7;
```

VariableDeclaration → `var`

2. Syntax analysis

Transform the list of tokens into an ***AST***

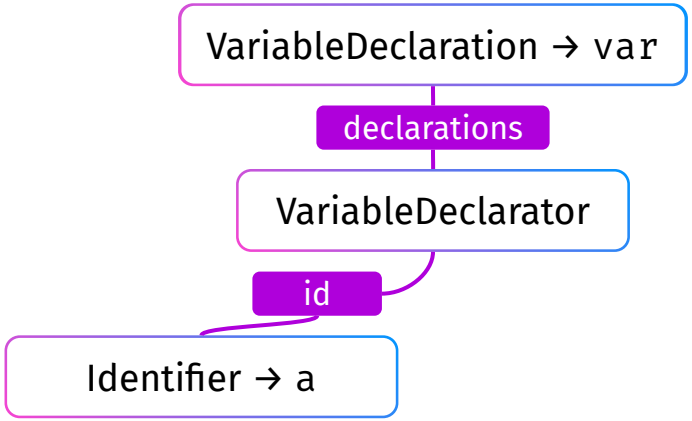
```
var a = 7;
```



2. Syntax analysis

Transform the list of tokens into an ***AST***

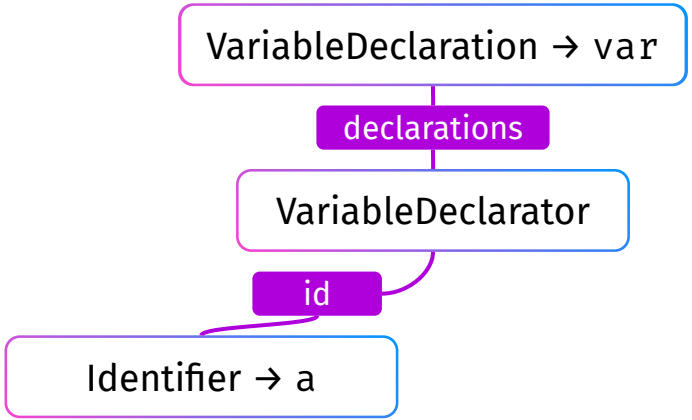
```
var a = 7;
```



2. Syntax analysis

Transform the list of tokens into an ***AST***

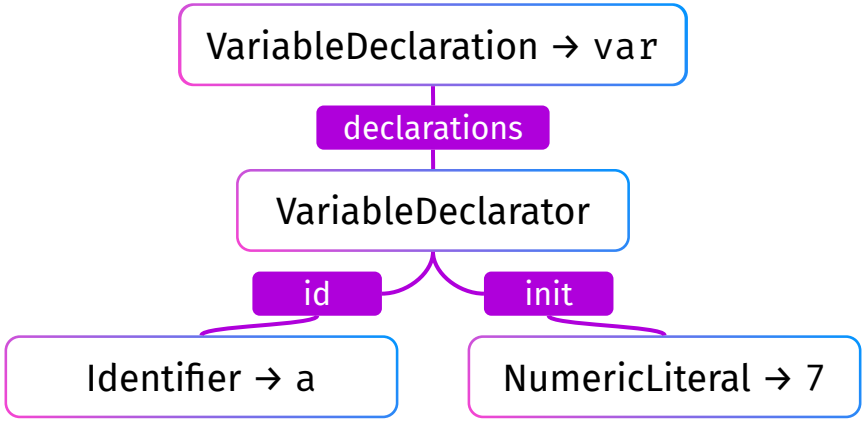
```
var a = 7;
```



2. Syntax analysis

Transform the list of tokens into an ***AST***

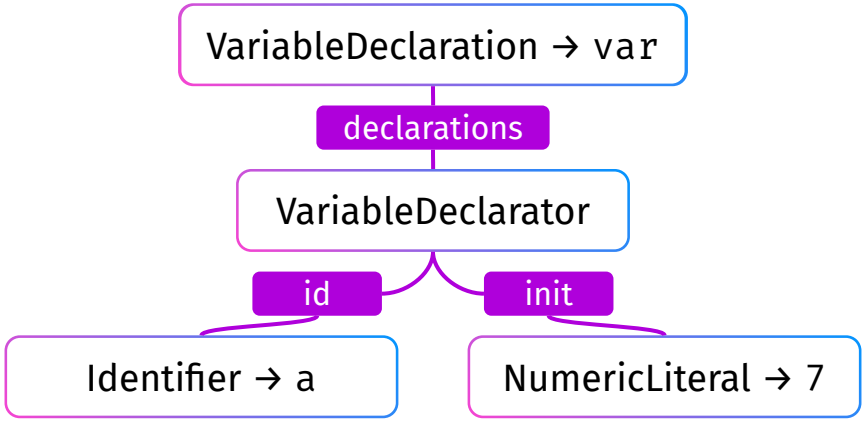
```
var a = 7;
```



2. Syntax analysis

Transform the list of tokens into an ***AST***

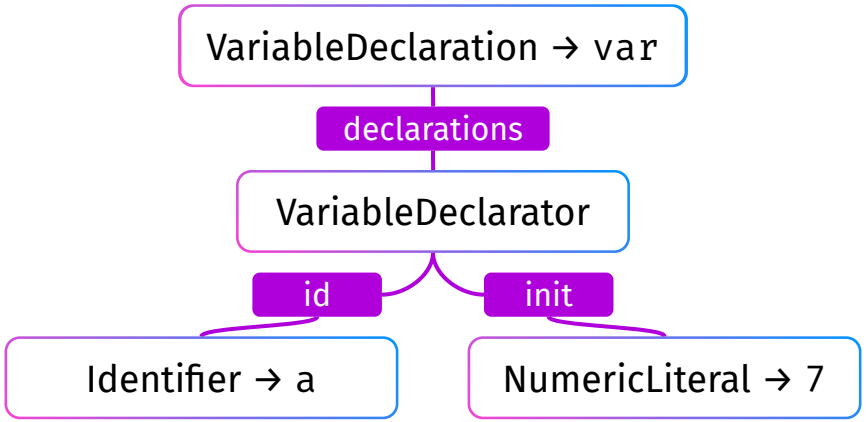
```
var a = 7;
```



2. Syntax analysis

Transform the list of tokens into an ***AST***

```
var a = 7;
```



2. Syntax analysis

Handle automatic semicolon insertion (*ASI*)

2. Syntax analysis

Handle automatic semicolon insertion (*ASI*)

```
var a = foo
foo.forEach(fn)
```

2. Syntax analysis

Handle automatic semicolon insertion (*ASI*)

```
var a = foo
foo.forEach(fn)
```

```
var a = foo;
foo.forEach(fn);
```

2. Syntax analysis

Handle automatic semicolon insertion (*ASI*)

```
var a = foo
foo.forEach(fn)
```

```
var a = foo;
foo.forEach(fn);
```

```
var a = foo
[7].forEach(fn)
```

2. Syntax analysis

Handle automatic semicolon insertion (*ASI*)

```
var a = foo
foo.forEach(fn)
```

```
var a = foo;
foo.forEach(fn);
```

```
var a = foo
[7].forEach(fn)
```

```
var a = foo[7].forEach(n);
```

2. Syntax analysis

Report errors about misplaced tokens

2. Syntax analysis

Report errors about misplaced tokens

Unexpected token, expected ")"

```
var a = double(7;
```

2. Syntax analysis

Report errors about misplaced tokens

Unexpected token, expected ")"

```
var a = double(7;
```

Unexpected keyword 'if'

```
1 + if;
```

3. Semantic analysis

Check the AST respects all the static ECMAScript rules: ***early errors***

3. Semantic analysis

Check the AST respects all the static ECMAScript rules: ***early errors***

Redefinition of __proto__ property

```
({ __proto__: x,
   __proto__: y,
})
```

3. Semantic analysis

Check that the AST respects all the static ECMAScript rules: ***early errors***

Redefinition of __proto__ property

```
({ __proto__: x,
   __proto__: y,
})
```

'with' in strict mode

```
"use strict";
with (obj) {}
```

3. Semantic analysis

Report errors about invalid variables, using a ***scope tracker***

3. Semantic analysis

Report errors about invalid variables, using a ***scope tracker***

Identifier 'foo' has already been declared

```
let foo = 2;
let foo = 3;
```

3. Semantic analysis

Report errors about invalid variables, using a ***scope tracker***

Identifier 'foo' has already been declared

```
let foo = 2;
let foo = 3;
```

Export 'bar' is not defined

```
{ let bar = 2 }
export { bar };
```

A look inside Babel:

`@babel/traverse`

Declarative traversal

Algorithm: Depth-first search, in-order (`enter`) and out-order (`exit`)

```
traverse(ast, {
  CallExpression: {
    enter() {
      console.log("Function call!")
    }
  }
})
```

Declarative traversal

Algorithm: Depth-first search, in-order (`enter`) and out-order (`exit`)

```
traverse(ast, {
  CallExpression: {
    enter() {
      console.log("Function call!")
    }
  }
})
```

Declarative traversal

Algorithm: Depth-first search, in-order (`enter`) and out-order (`exit`)

```
traverse(ast, {
  CallExpression: {
    enter() {
      console.log("Function call!")
    }
  }
})
```

Declarative traversal

Algorithm: Depth-first search, in-order (`enter`) and out-order (`exit`)

```
traverse(ast, {
  CallExpression: {
    enter() {
      console.log("Function call!")
    }
  }
})
```

`enter` is the default traversal order

Declarative traversal

Algorithm: Depth-first search, in-order (`enter`) and out-order (`exit`)

```
traverse(ast, {
  CallExpression() {

      console.log("Function call!")

  }
})
```

`enter` is the default traversal order

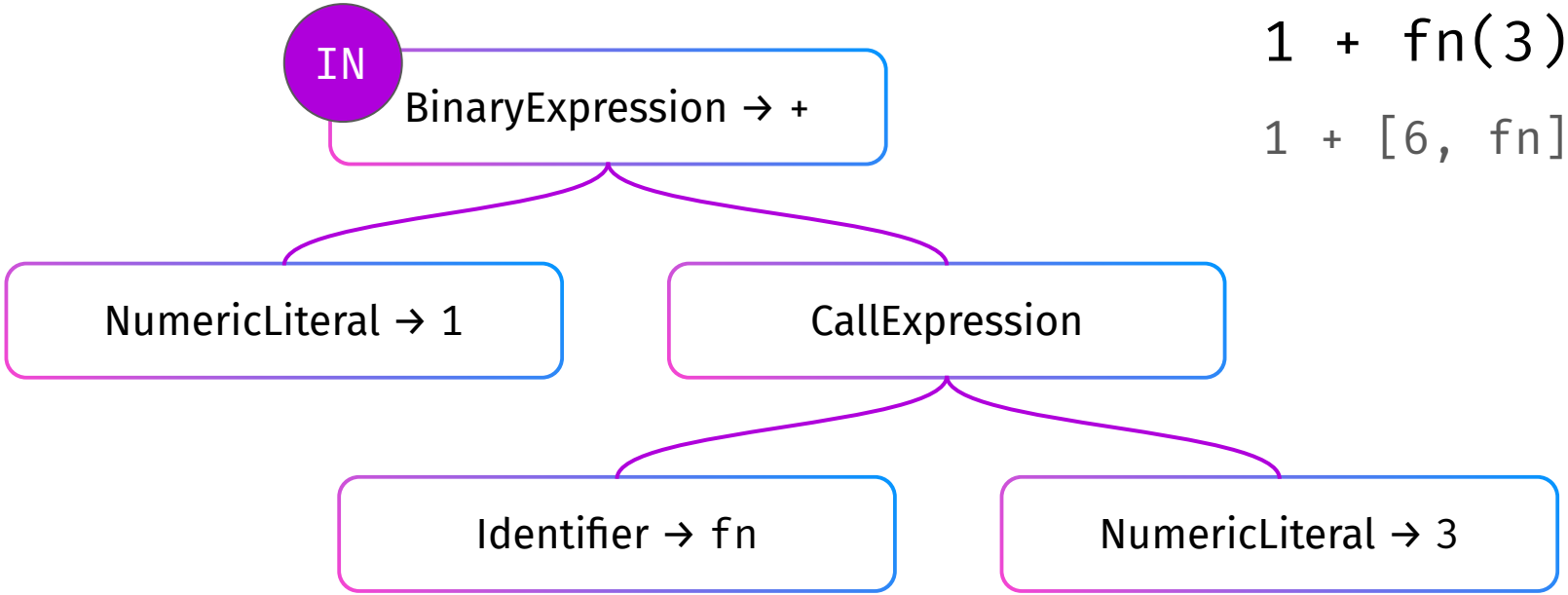
Dynamic Abstract Syntax Tree

Example:

- ➔ Traverse `1 + fn(3)`
- ➔ When we reach `fn(3)`, during the `"exit"` phase, replace it with `[6, fn]`

```
1 + fn(3)
1 + [6, fn]
```

Dynamic Abstract Syntax Tree



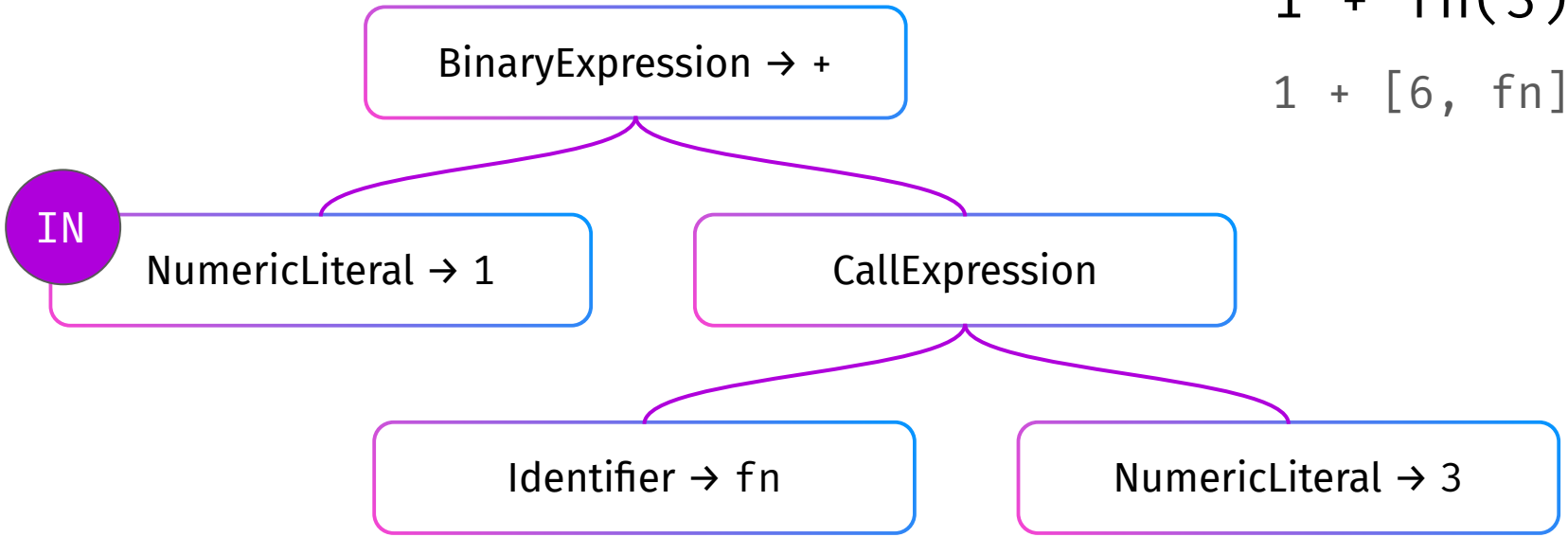

```
1 + fn(3)
```

```
1 + [6, fn]
```

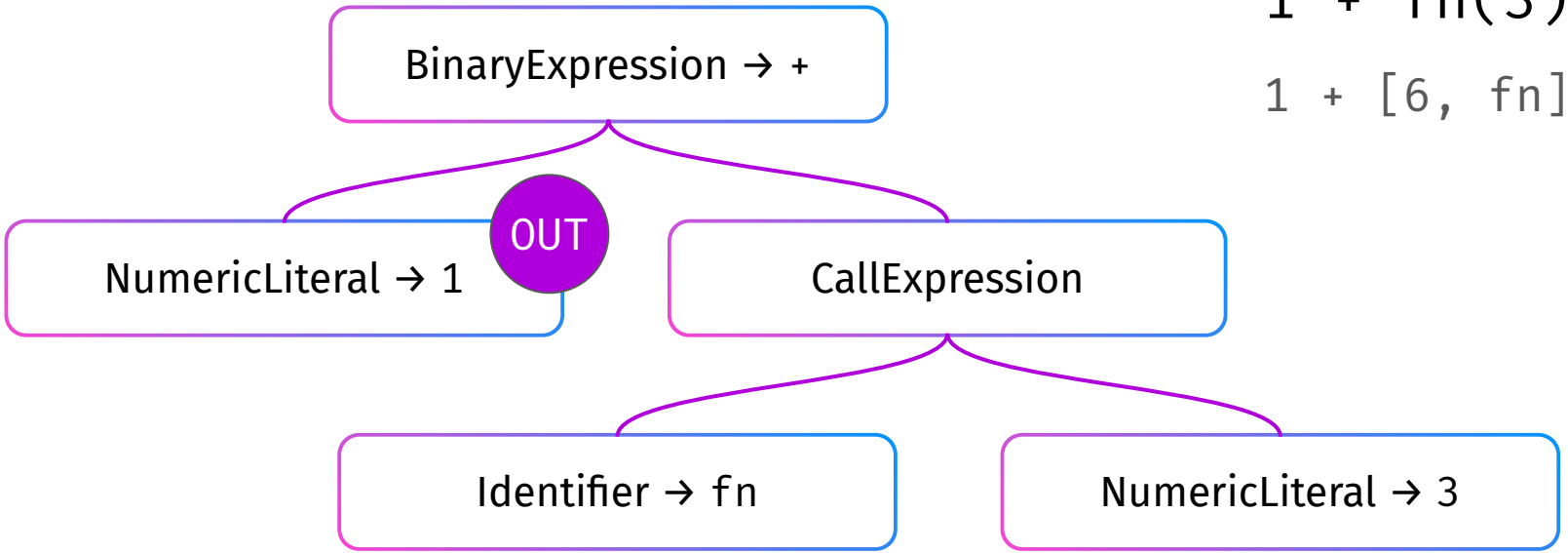
Dynamic Abstract Syntax Tree

```
1 + fn(3)
```

```
1 + [6, fn]
```



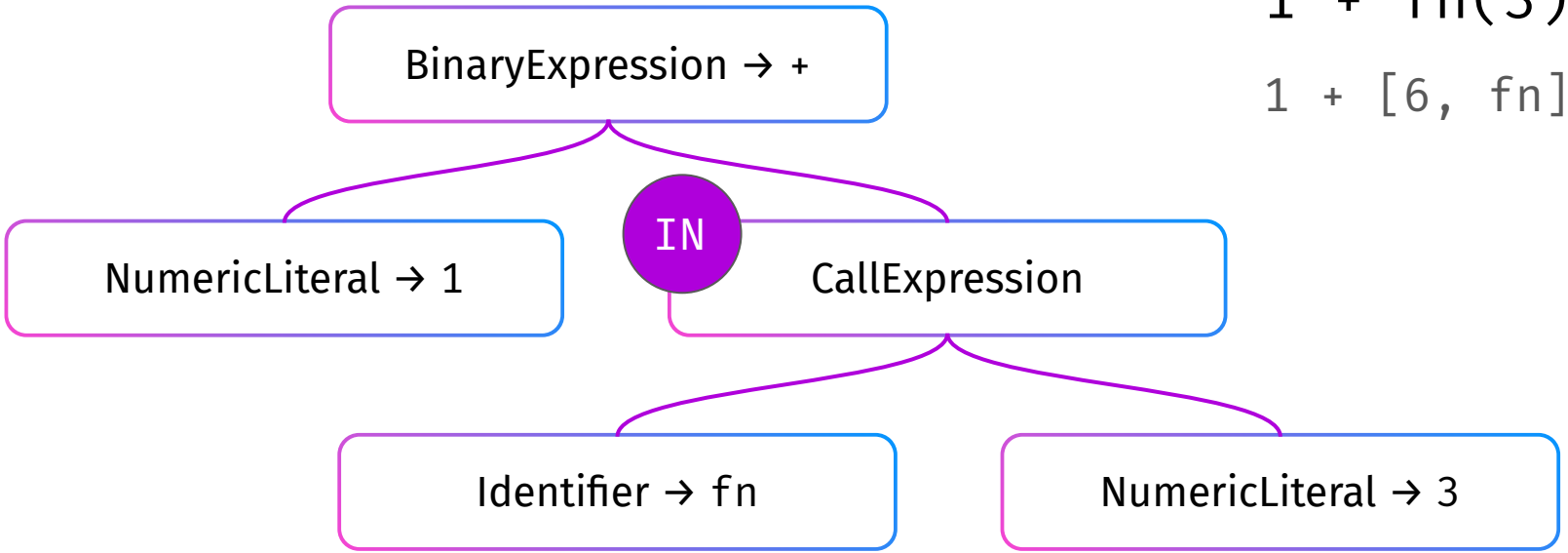
Dynamic Abstract Syntax Tree




```
1 + fn(3)
```

```
1 + [6, fn]
```

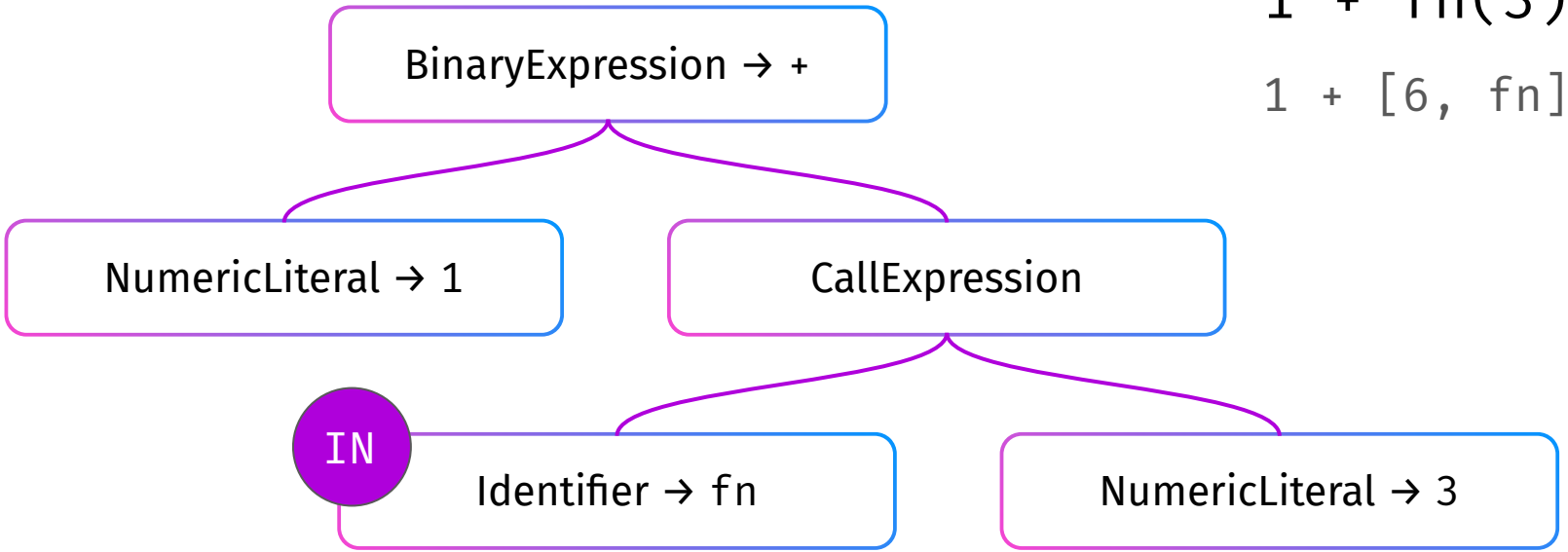
Dynamic Abstract Syntax Tree




```
1 + fn(3)
```

```
1 + [6, fn]
```

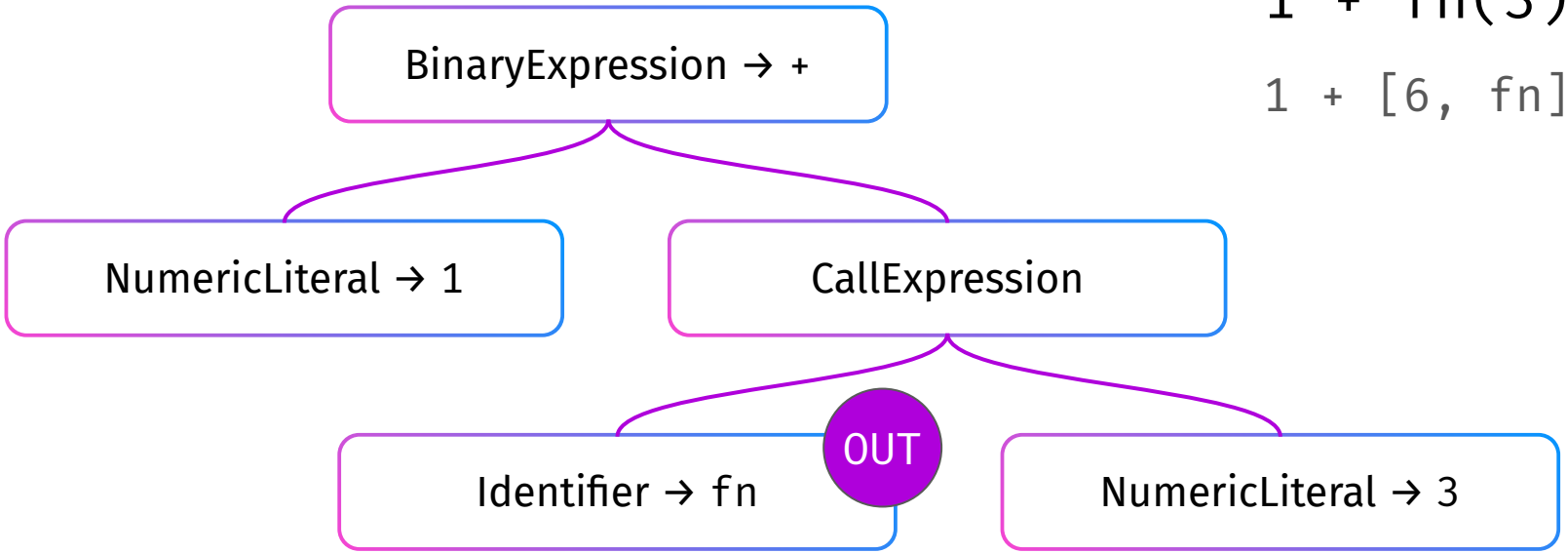
Dynamic Abstract Syntax Tree




```
1 + fn(3)
```

```
1 + [6, fn]
```

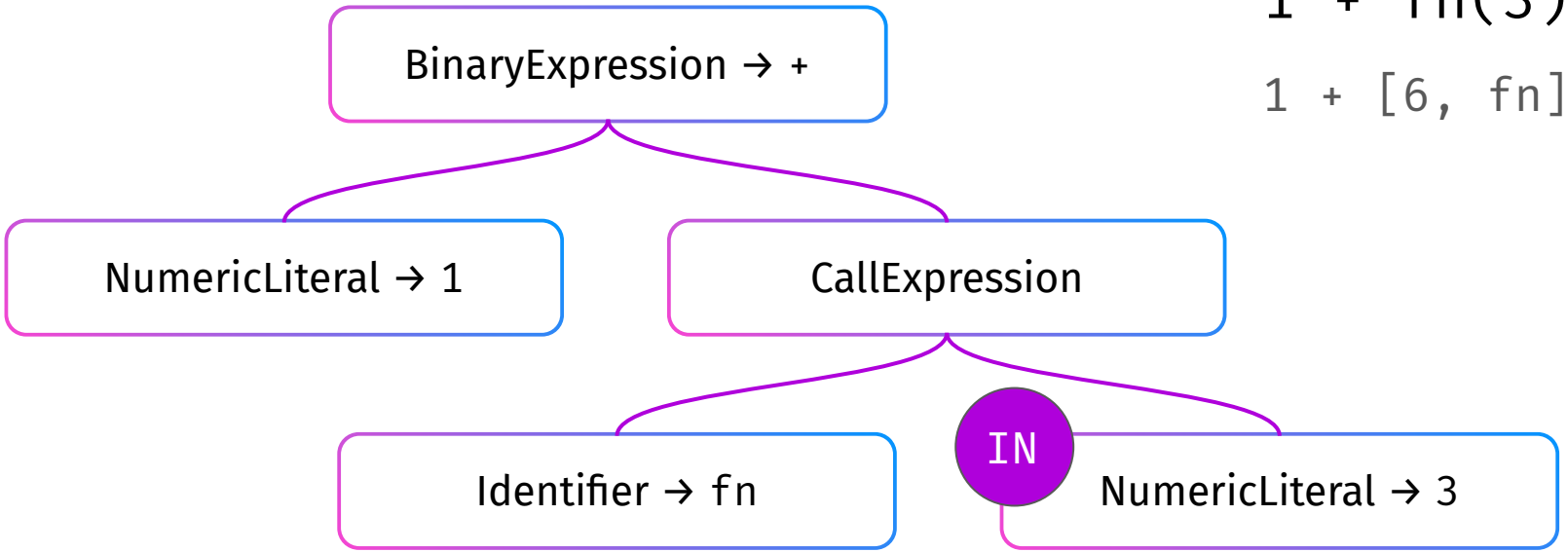
Dynamic Abstract Syntax Tree



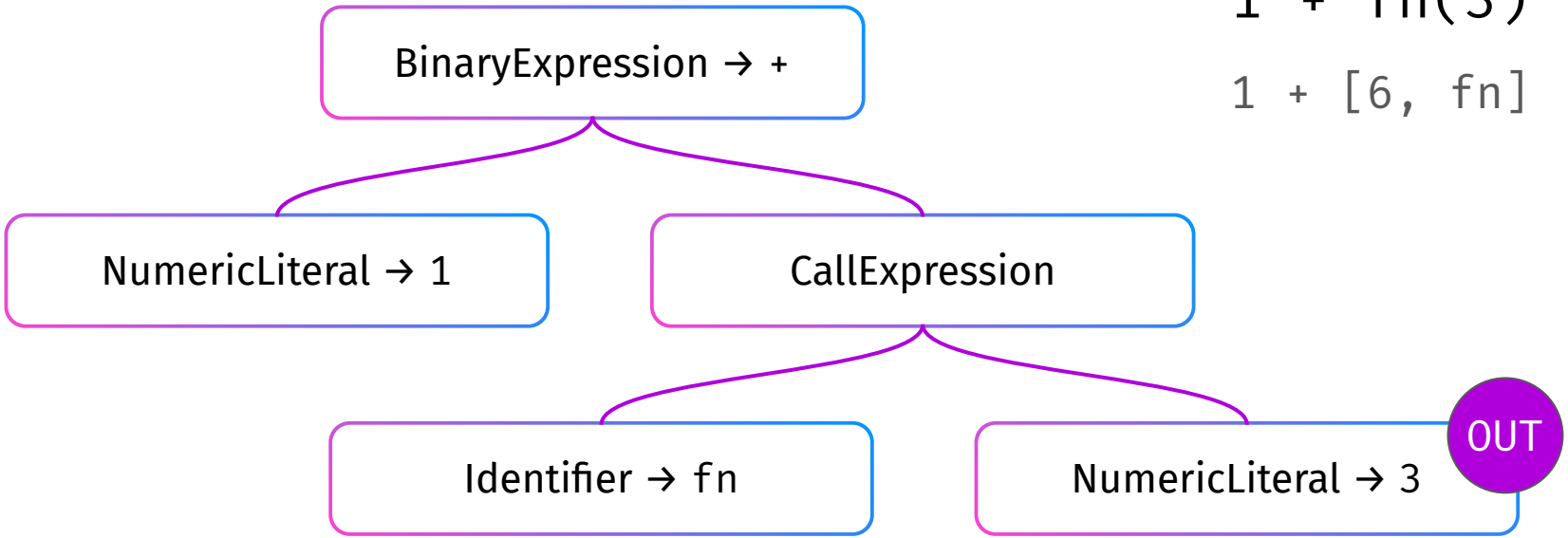

```
1 + fn(3)
```

```
1 + [6, fn]
```

Dynamic Abstract Syntax Tree



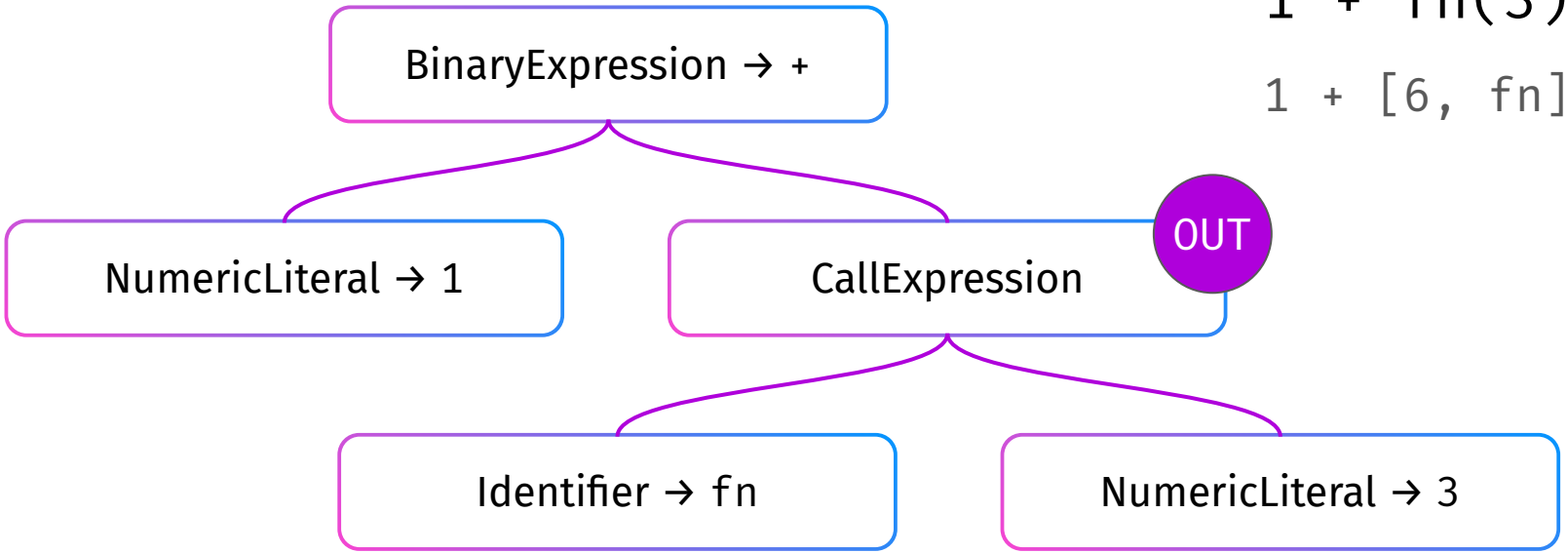
Dynamic Abstract Syntax Tree



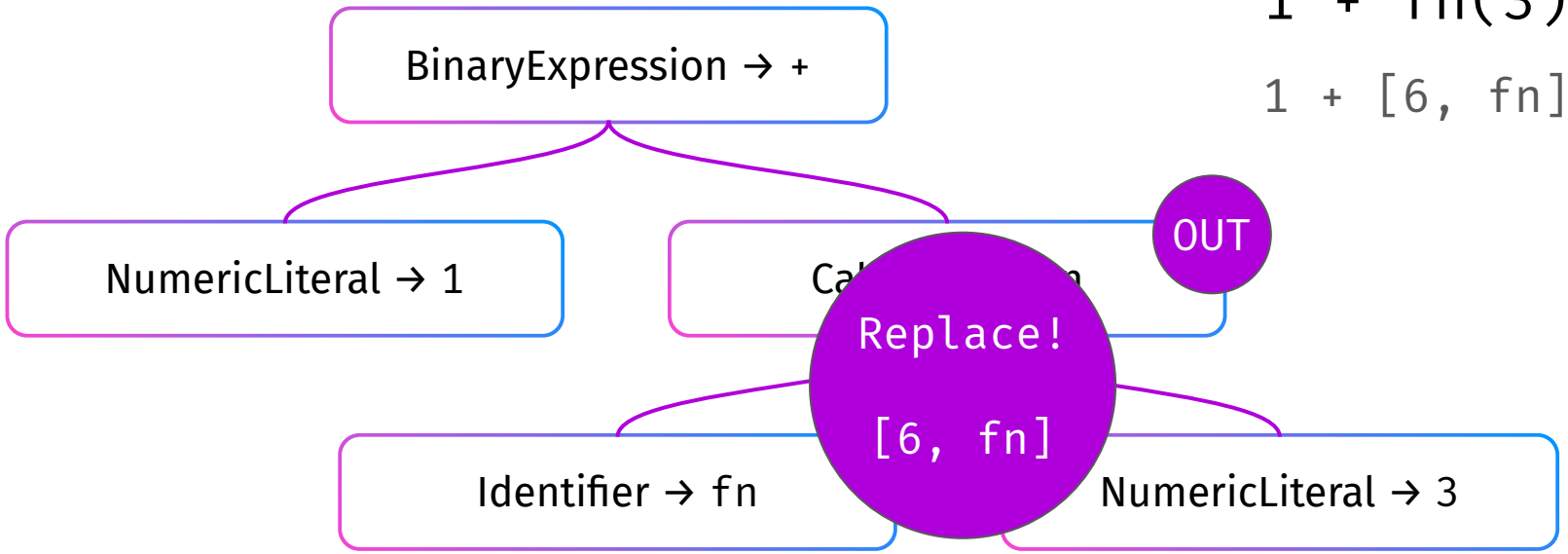
Dynamic Abstract Syntax Tree

```
1 + fn(3)
```

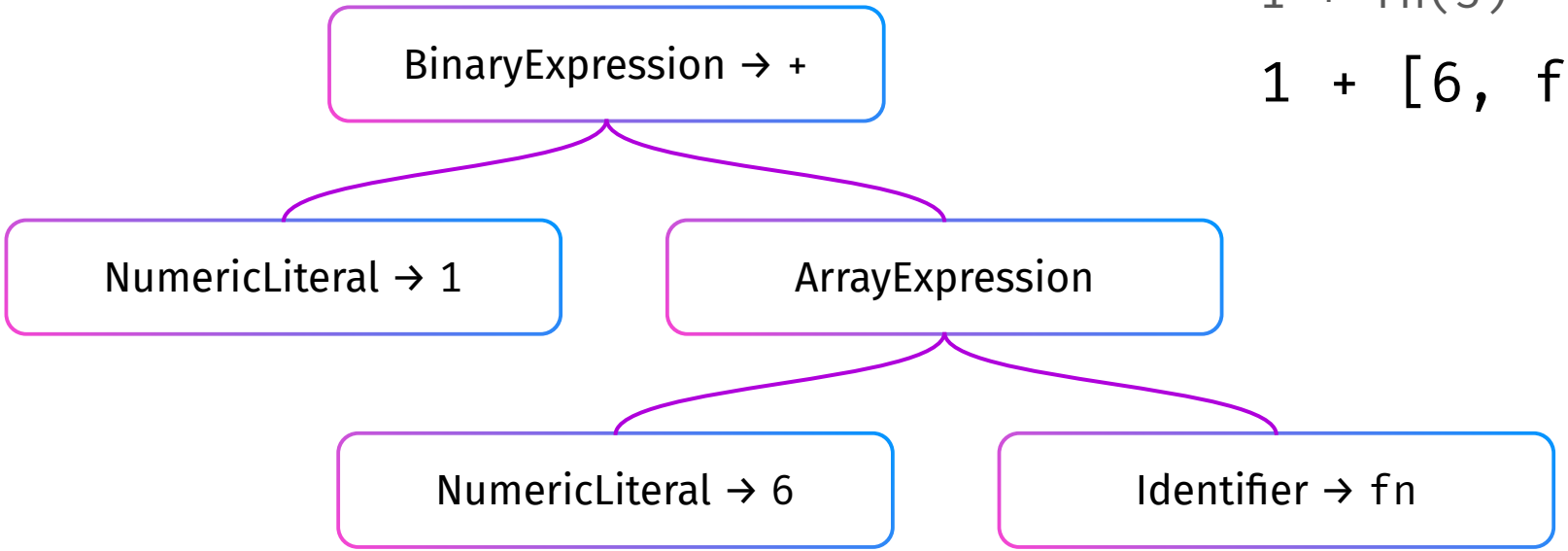
```
1 + [6, fn]
```



Dynamic Abstract Syntax Tree



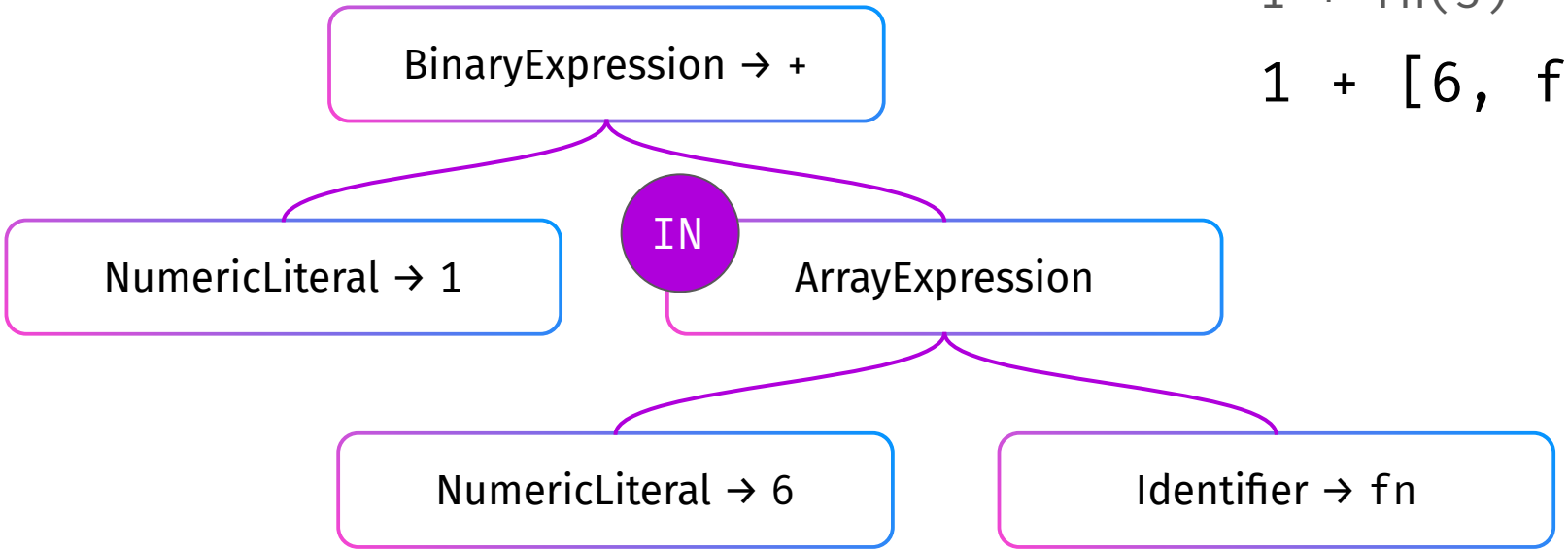
Dynamic Abstract Syntax Tree




```
1 + fn(3)
```

```
1 + [6, fn]
```

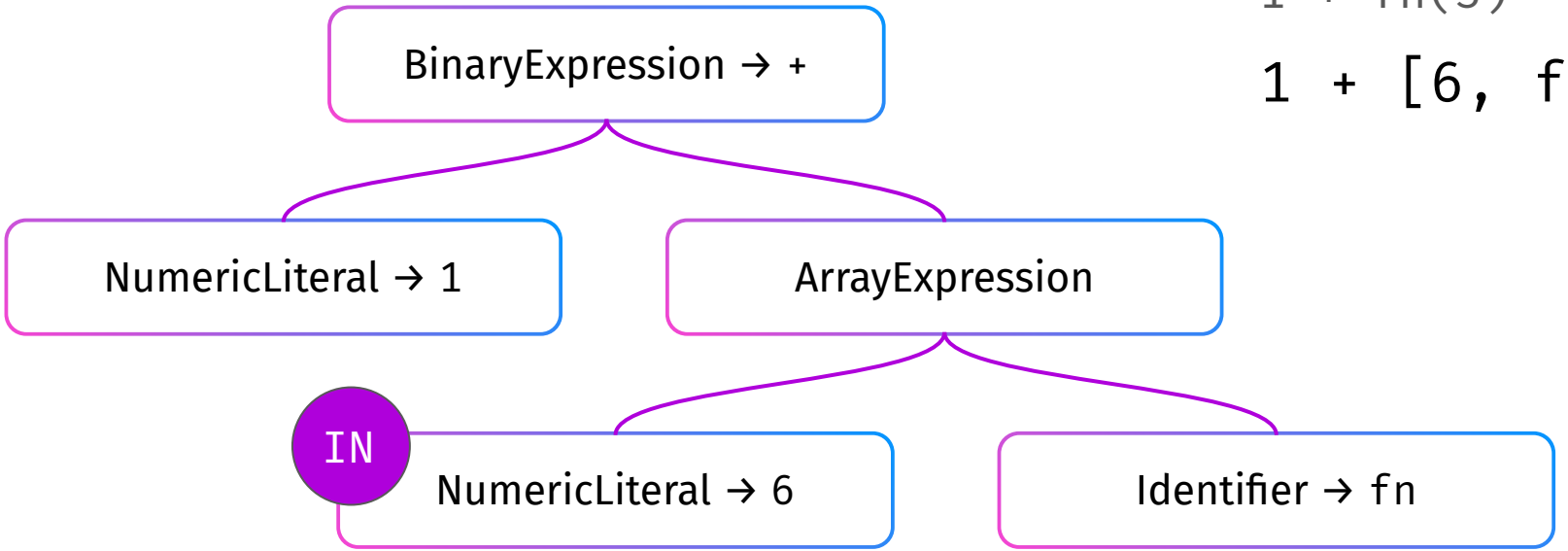
Dynamic Abstract Syntax Tree




```
1 + fn(3)
```

```
1 + [6, fn]
```

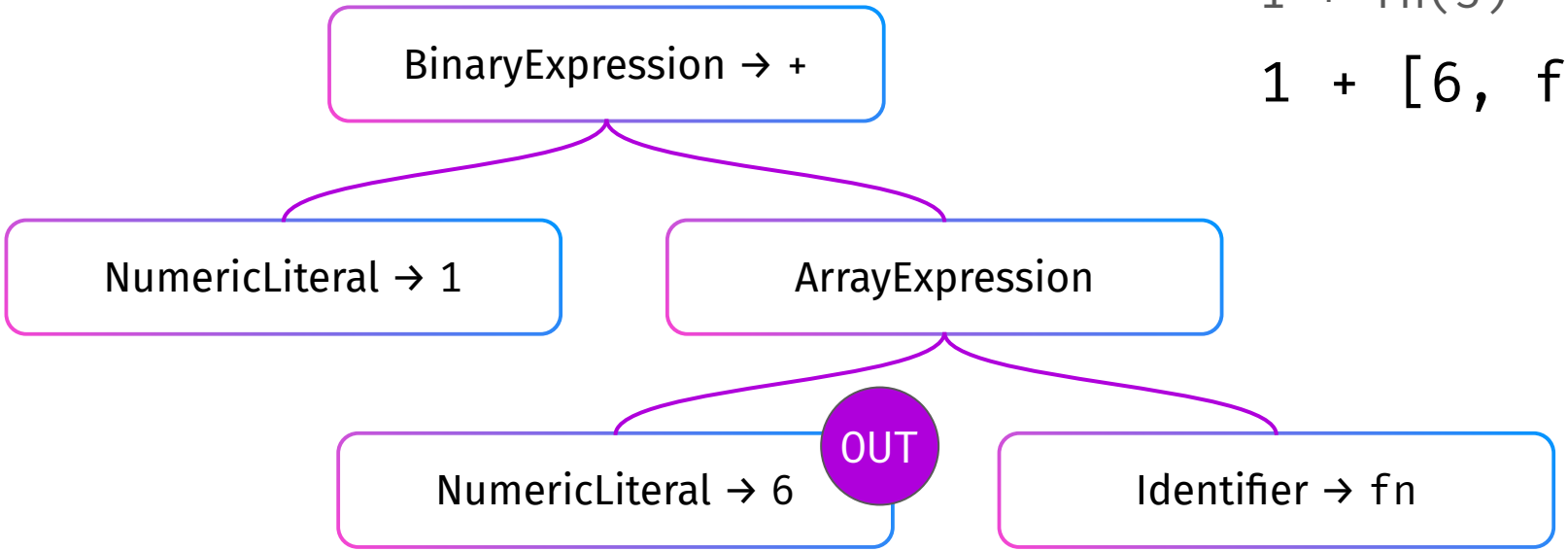
Dynamic Abstract Syntax Tree




```
1 + fn(3)
```

```
1 + [6, fn]
```

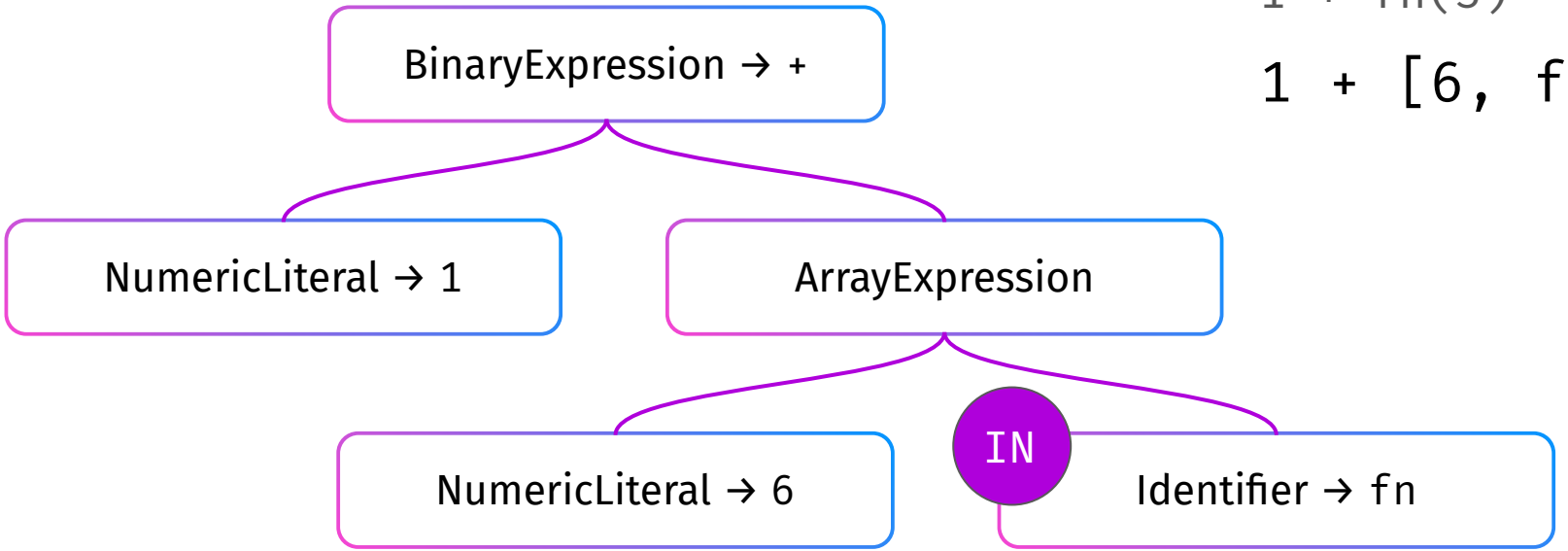
Dynamic Abstract Syntax Tree




```
1 + fn(3)
```

```
1 + [6, fn]
```

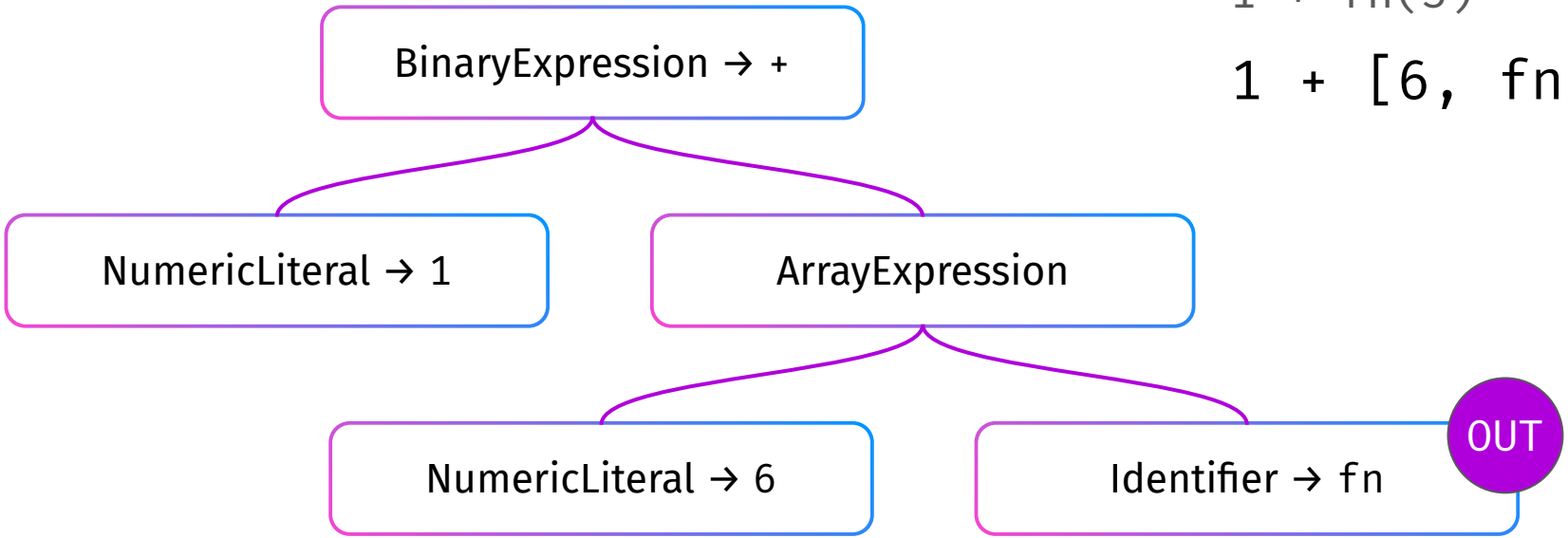
Dynamic Abstract Syntax Tree




```
1 + fn(3)
```

```
1 + [6, fn]
```

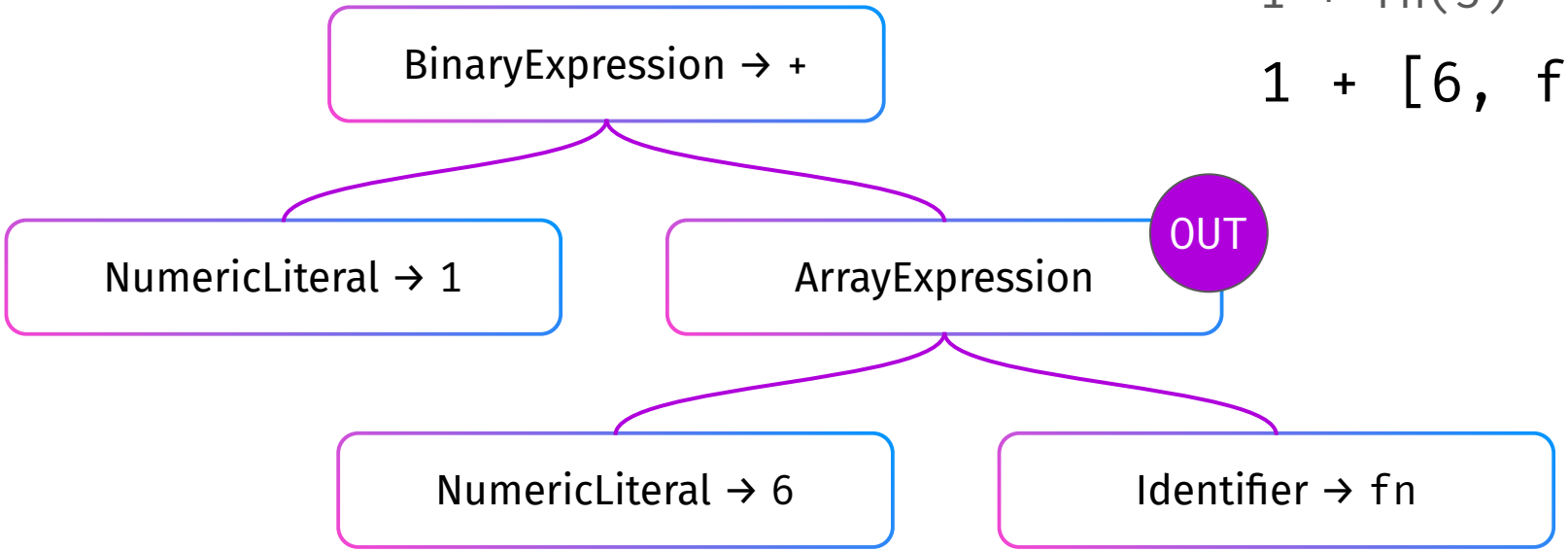
Dynamic Abstract Syntax Tree




```
1 + fn(3)
```

```
1 + [6, fn]
```

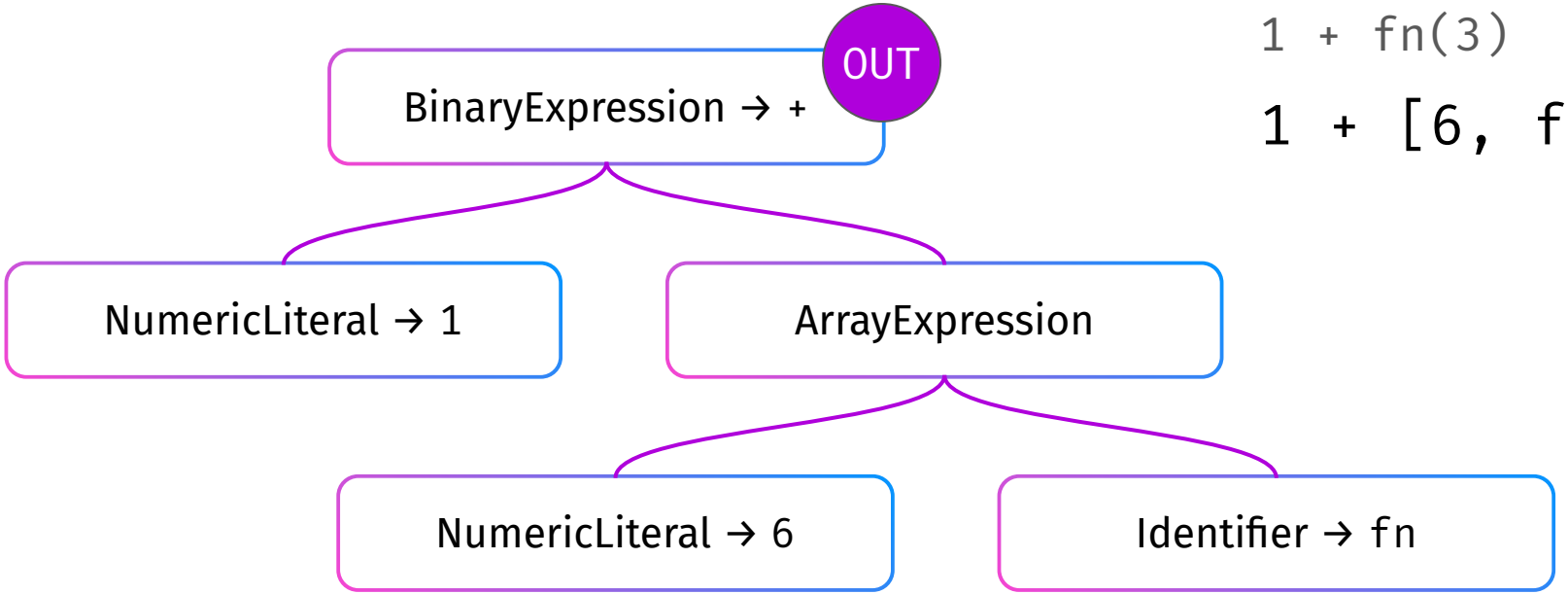
Dynamic Abstract Syntax Tree




```
1 + fn(3)
```

```
1 + [6, fn]
```

Dynamic Abstract Syntax Tree




```
1 + fn(3)
```

```
1 + [6, fn]
```

Scope analysis

```
let MAX = 5;

export function isValid(val, big) {
  if (big) {
    const MAX = 10;
    return val < MAX;
  }
  MAX += 1;
  return val < MAX;
}
```

Scope analysis

1. Collect different scopes

```
let MAX = 5;

export function isValid(val, big) {
  if (big) {
    const MAX = 10;
    return val < MAX;
  }
  MAX += 1;
  return val < MAX;
}
```

Scope analysis

```
let MAX = 5;

export function isValid(val, big) {
  if (big) {
    const MAX = 10;
    return val < MAX;
  }
  MAX += 1;
  return val < MAX;
}
```

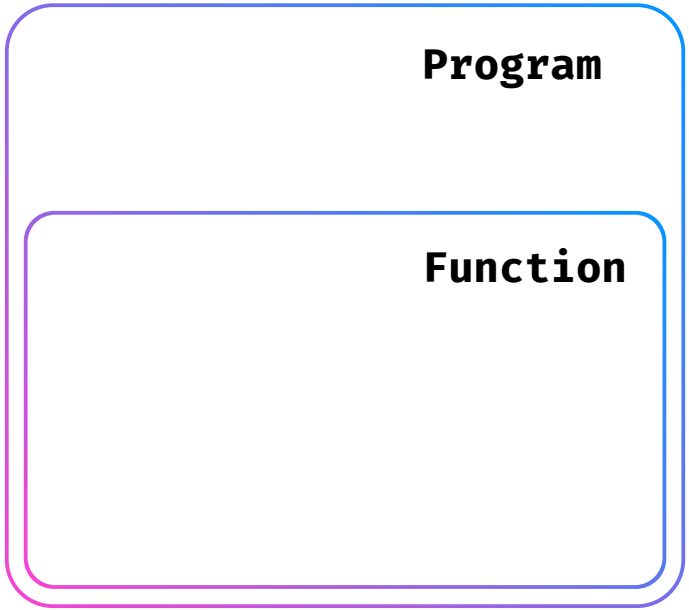
1. Collect different scopes

Scope analysis

```
let MAX = 5;

export function isValid(val, big) {
  if (big) {
    const MAX = 10;
    return val < MAX;
  }
  MAX += 1;
  return val < MAX;
}
```

1. Collect different scopes

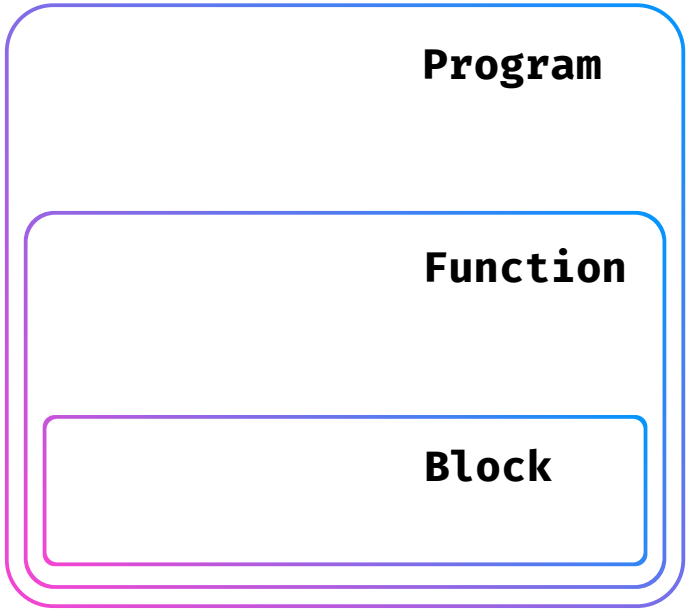


Scope analysis

```
let MAX = 5;

export function isValid(val, big) {
  if (big) {
    const MAX = 10;
    return val < MAX;
  }
  MAX += 1;
  return val < MAX;
}
```

1. Collect different scopes

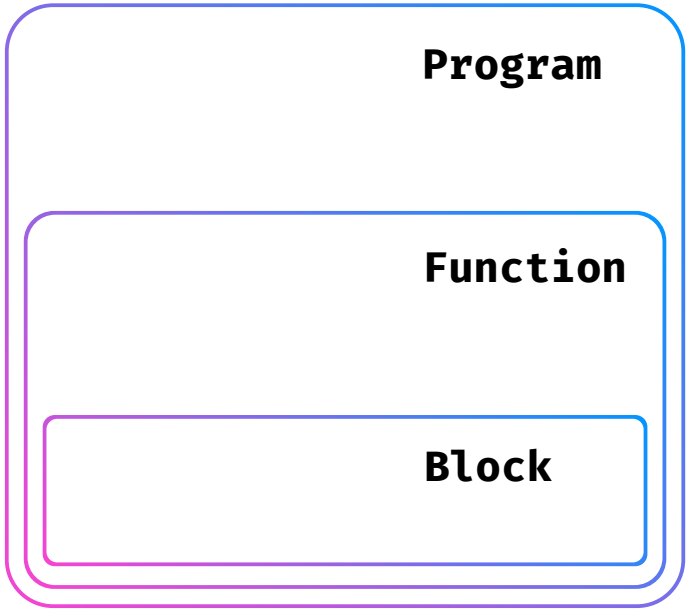


Scope analysis

```
let MAX = 5;

export function isValid(val, big) {
  if (big) {
    const MAX = 10;
    return val < MAX;
  }
  MAX += 1;
  return val < MAX;
}
```

2. Collect declarations

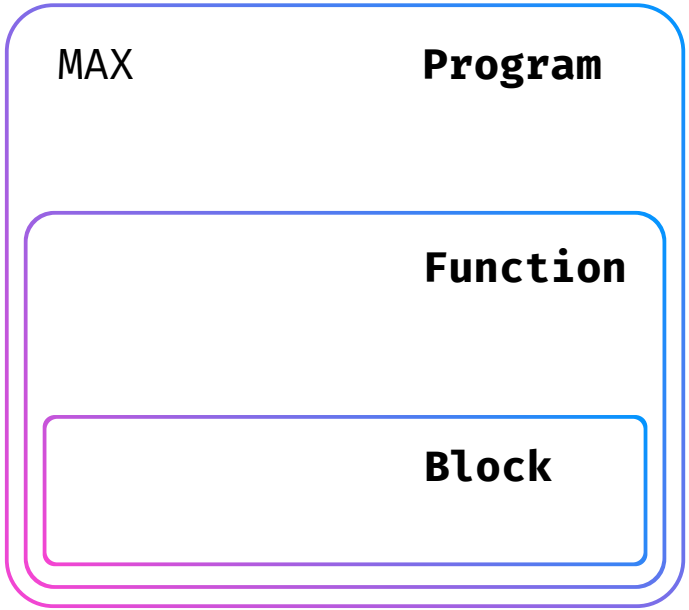


Scope analysis

```
let MAX = 5;

export function isValid(val, big) {
  if (big) {
    const MAX = 10;
    return val < MAX;
  }
  MAX += 1;
  return val < MAX;
}
```

2. Collect declarations

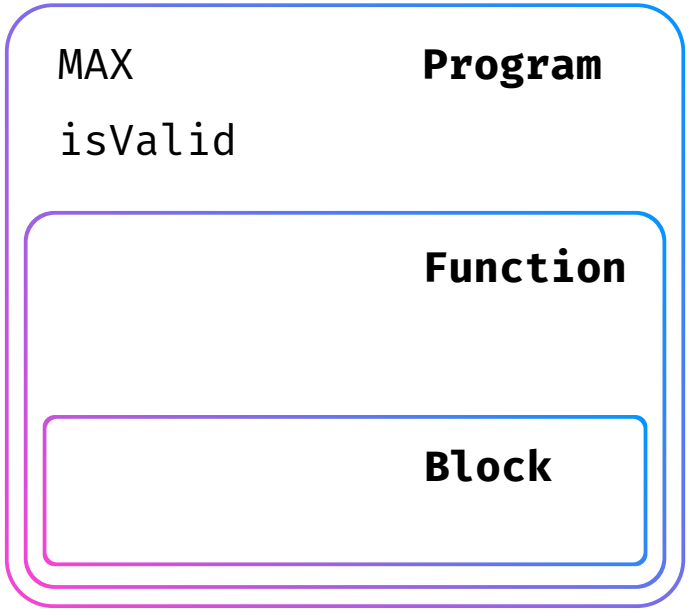


Scope analysis

```
let MAX = 5;

export function isValid(val, big) {
  if (big) {
    const MAX = 10;
    return val < MAX;
  }
  MAX += 1;
  return val < MAX;
}
```

2. Collect declarations

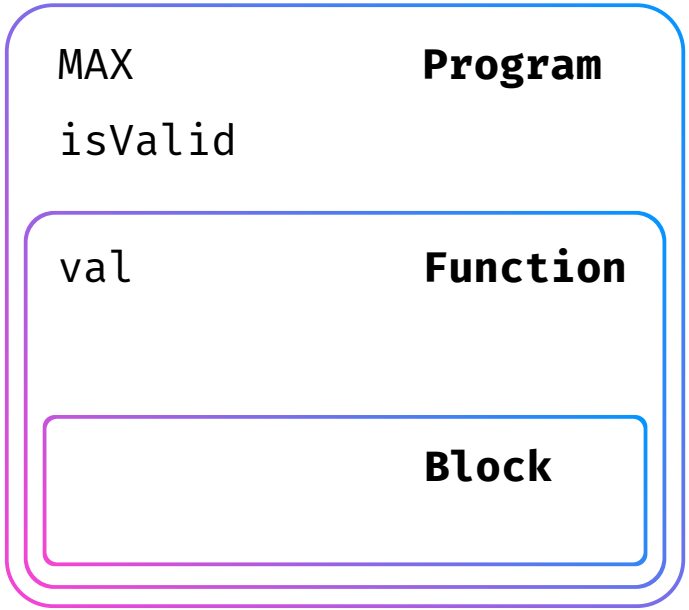


Scope analysis

```
let MAX = 5;

export function isValid(val, big) {
  if (big) {
    const MAX = 10;
    return val < MAX;
  }
  MAX += 1;
  return val < MAX;
}
```

2. Collect declarations

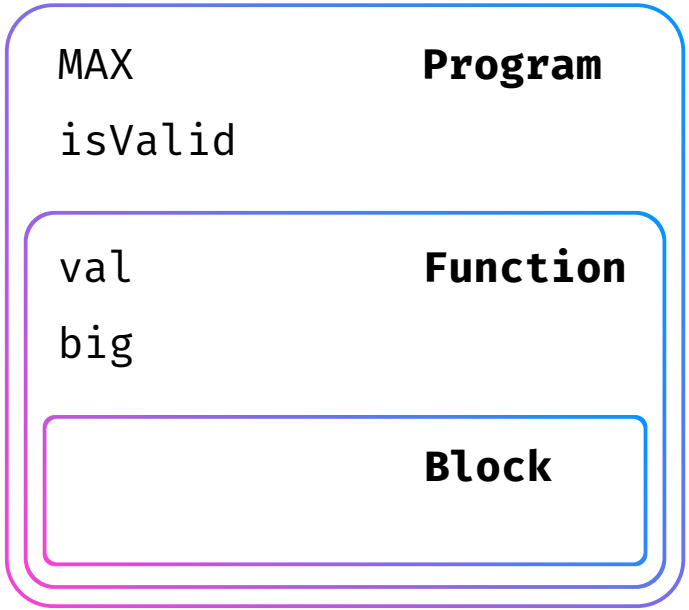


Scope analysis

```
let MAX = 5;

export function isValid(val, big) {
  if (big) {
    const MAX = 10;
    return val < MAX;
  }
  MAX += 1;
  return val < MAX;
}
```

2. Collect declarations

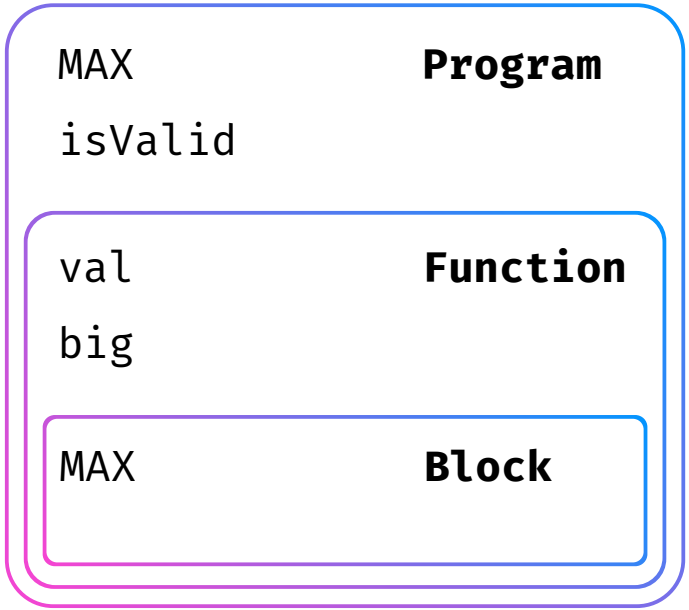


Scope analysis

```
let MAX = 5;

export function isValid(val, big) {
  if (big) {
    const MAX = 10;
    return val < MAX;
  }
  MAX += 1;
  return val < MAX;
}
```

2. Collect declarations

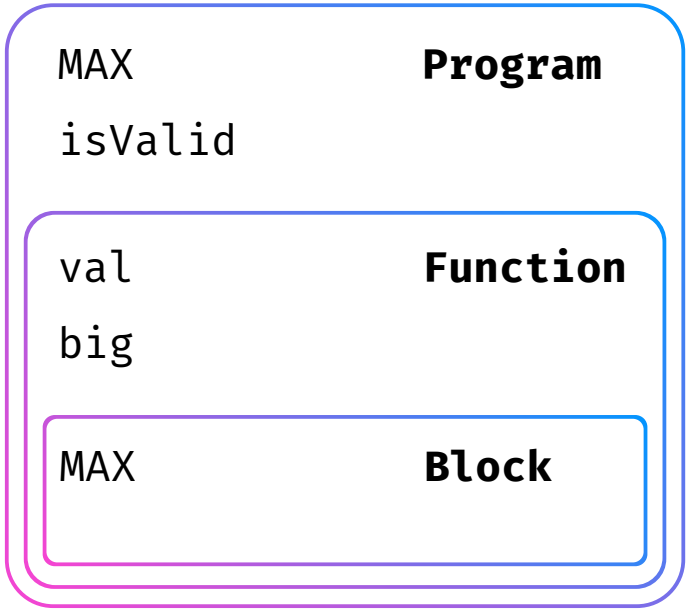


Scope analysis

```
let MAX = 5;

export function isValid(val, big) {
  if (big) {
    const MAX = 10;
    return val < MAX;
  }
  MAX += 1;
  return val < MAX;
}
```

3. Collect references

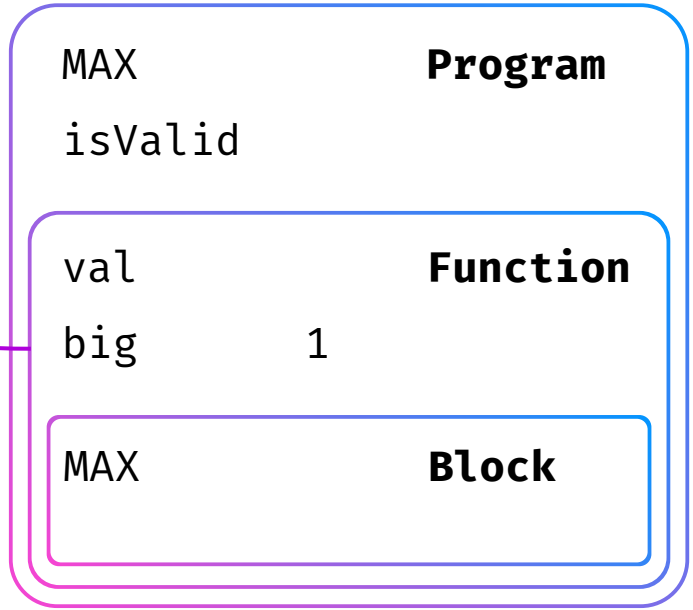


Scope analysis

```
let MAX = 5;

export function isValid(val, big) {
  if (big) {
    const MAX = 10;
    return val < MAX;
  }
  MAX += 1;
  return val < MAX;
}
```

3. Collect references

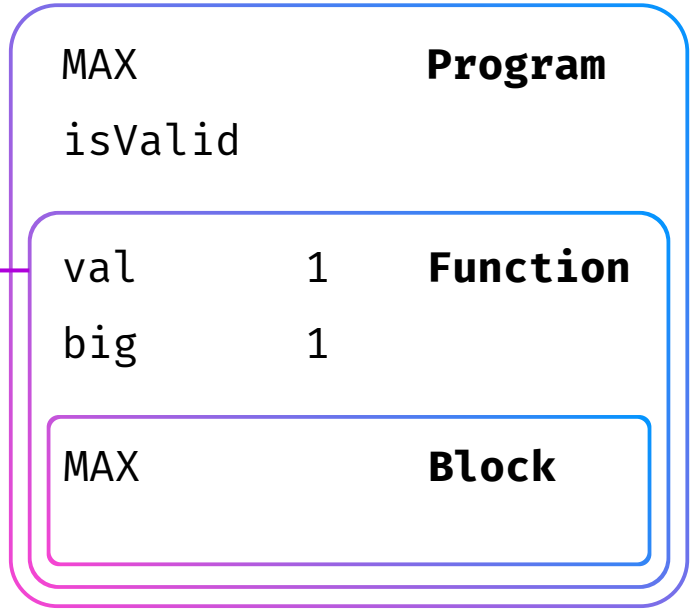


Scope analysis

```
let MAX = 5;

export function isValid(val, big) {
  if (big) {
    const MAX = 10;
    return val < MAX;
  }
  MAX += 1;
  return val < MAX;
}
```

3. Collect references

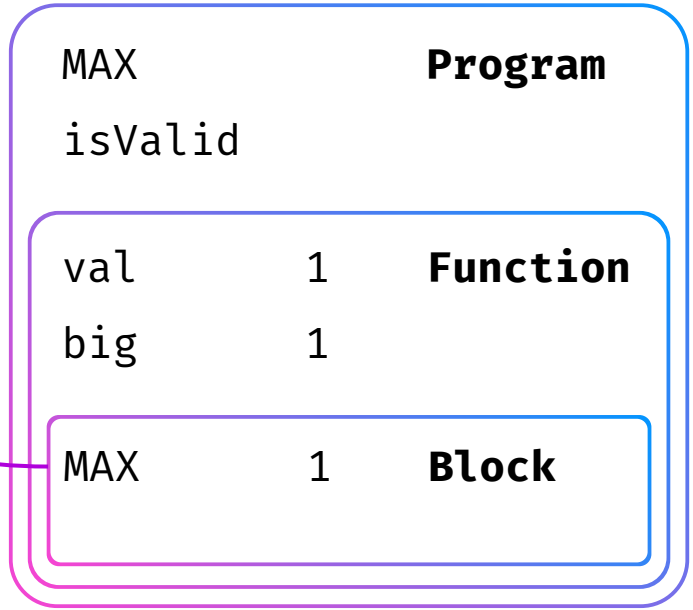


Scope analysis

```
let MAX = 5;

export function isValid(val, big) {
  if (big) {
    const MAX = 10;
    return val < MAX;
  }
  MAX += 1;
  return val < MAX;
}
```

3. Collect references

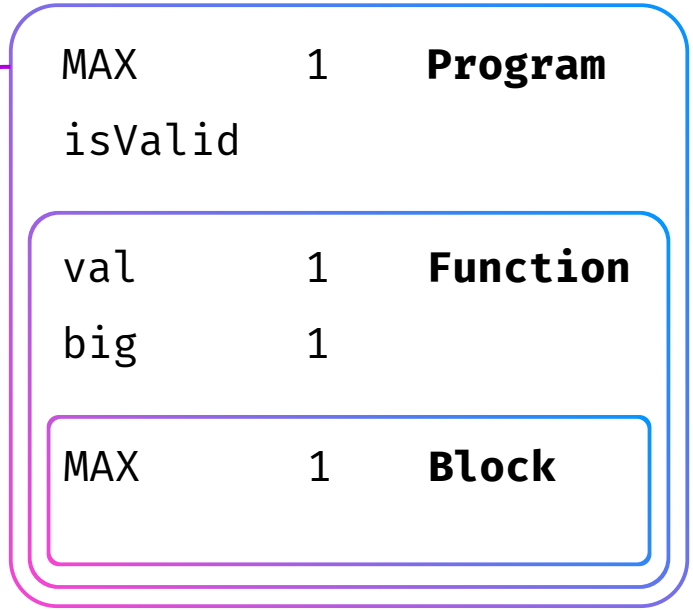


Scope analysis

```
let MAX = 5;

export function isValid(val, big) {
  if (big) {
    const MAX = 10;
    return val < MAX;
  }
  MAX += 1;
  return val < MAX;
}
```

3. Collect references

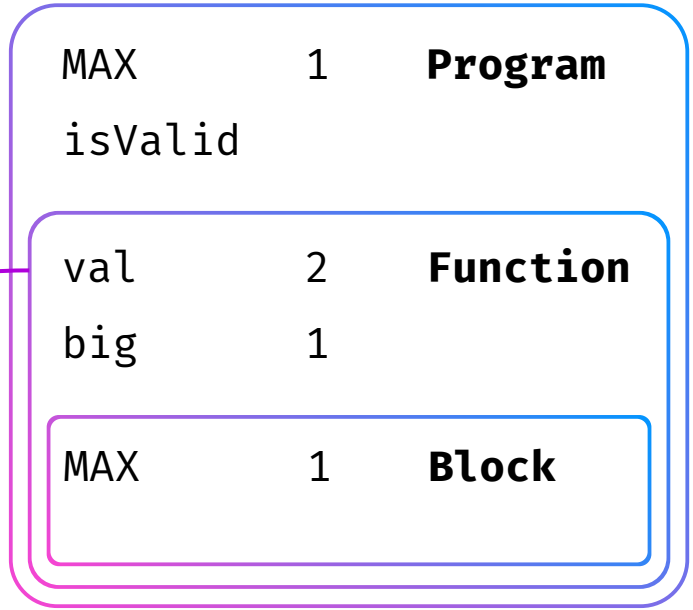


Scope analysis

```
let MAX = 5;

export function isValid(val, big) {
  if (big) {
    const MAX = 10;
    return val < MAX;
  }
  MAX += 1;
  return val < MAX;
}
```

3. Collect references

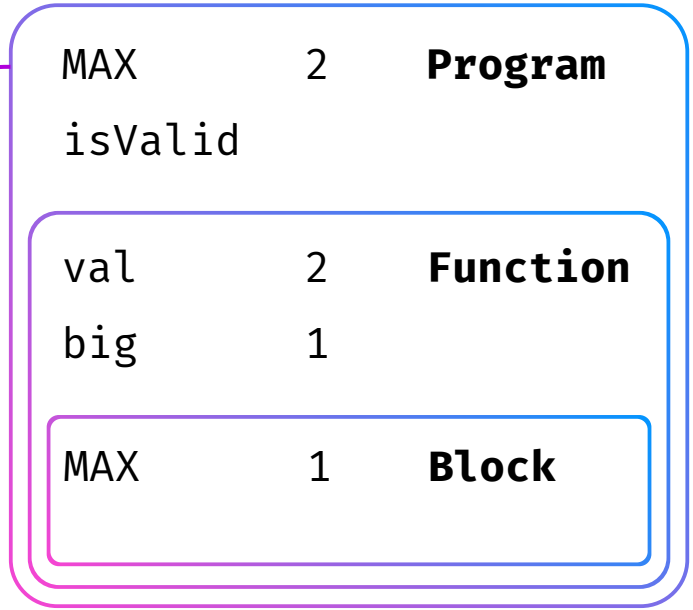


Scope analysis

```
let MAX = 5;

export function isValid(val, big) {
  if (big) {
    const MAX = 10;
    return val < MAX;
  }
  MAX += 1;
  return val < MAX;
}
```

3. Collect references

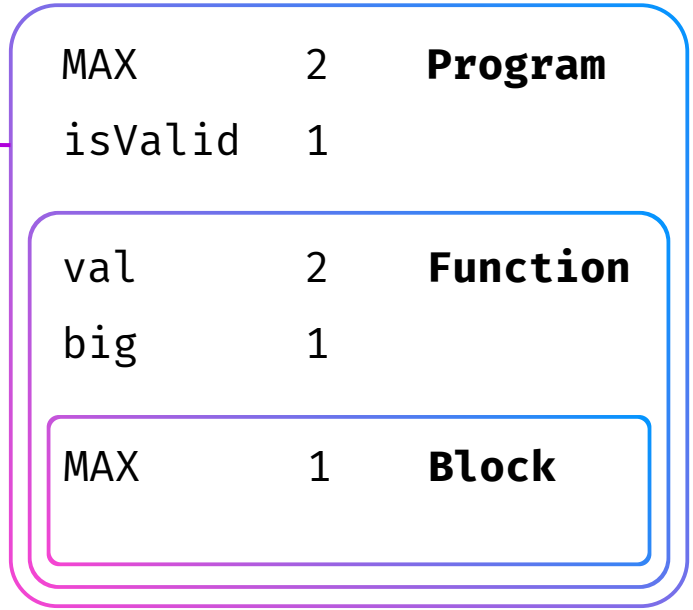


Scope analysis

```
let MAX = 5;

export function isValid(val, big) {
  if (big) {
    const MAX = 10;
    return val < MAX;
  }
  MAX += 1;
  return val < MAX;
}
```

3. Collect references

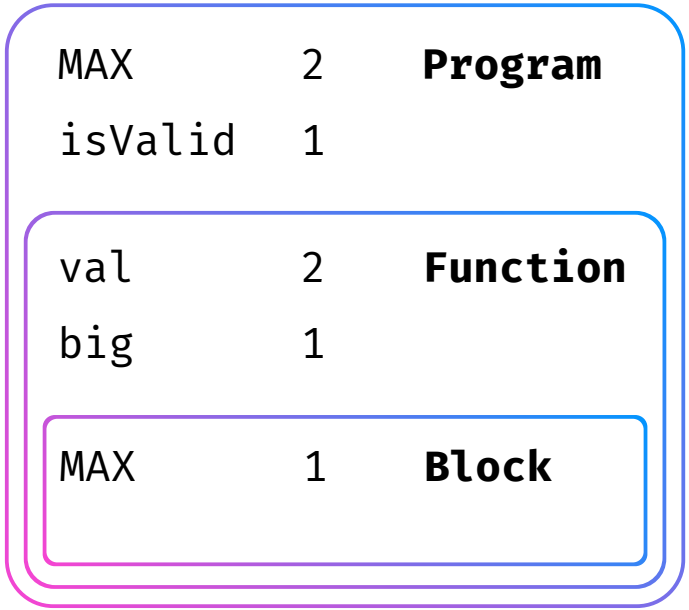


Scope analysis

```
let MAX = 5;

export function isValid(val, big) {
  if (big) {
    const MAX = 10;
    return val < MAX;
  }
  MAX += 1;
  return val < MAX;
}
```

3. Collect references



Utilities

NODES (AST)

Search

Introspection

Evaluation

Insertion

Removal

Replacement

Utilities

NODES (AST)

Search

Introspection

Evaluation

Insertion

Removal

Replacement

BINDINGS (SCOPE)

Validation

Tracking

Creation

Renaming

A look inside Babel:

`@babel/generator`

Transform AST to source code

Insert parentheses, comments and indentation where needed

Fast and opinionated

Transform AST to source code

Insert parentheses, comments and indentation where needed

Fast and opinionated

⚠️ **IT'S NOT A PRETTY PRINTER** ⚠️

A look inside Babel:

`@babel/core`

Babel's entrypoint

Used by Babel integrations

`@babel/cli`

`@babel/register`

`babel-loader`

`gulp-plugin-babel`

`babelify`

Parcel

Babel's entrypoint

Used by Babel integrations

`@babel/cli`

`@babel/register`

`babel-loader`

`gulp-plugin-babel`

`babelify`

Parcel

Merges configuration sources

`babel.config.js`

`.babelrc`

`package.json`

programmatic options

Babel's entrypoint

Used by Babel integrations

`@babel/cli`

`@babel/register`

`babel-loader`

`gulp-plugin-babel`

`babelify`

Parcel

Merges configuration sources

`babel.config.js`

`.babelrc`

`package.json`

programmatic options

Runs plugins and presets

Bonus package:

`@babel/types`

Nodes validation

Is this node an expression?

```
t.isExpression(node)
```

Nodes validation

Is this node an expression?

```
t.isExpression(node)
```

Is this node an identifier whose `name` is `"test"`?

```
t.isIdentifier(node, { name: "test" })
```

Nodes validation

Is this node an expression?

```
t.isExpression(node)
```

Is this node an identifier whose `name` is `"test"`?

```
t.isIdentifier(node, { name: "test" })
```

Is this node a sum whose left operand is the `child` node?

```
t.isBinaryExpression(node, { operator: "+", left: child })
```

Nodes building

Given a `varId` node, how to increment the value it represents by 2?

```
t.assignmentExpression(
  "+=",
  varId,
  t.numericLiteral(2)
);
```

```
{
  type: "AssignmentExpression",
  operator: "+=",
  right: varId,
  left: {
    type: "NumericLiteral",
    value: 2,
  },
}
```

Nodes building

Given a `varId` node, how to increment the value it represents by 2?

```
t.assignmentExpression(
  "+=",
  varId,
  t.numericLiteral(2)
);
```

```
{
  type: "AssignmentExpression",
  operator: "+=",
  right: varId,
  left: {
    type: "NumericLiteral",
    value: 2,
  },
}
```

Nodes building

Given a `varId` node, how to increment the value it represents by 2?

```
t.assignmentExpression(
  "+=",
  varId,
  t.numericLiteral(2)
);
```

```
{
  type: "AssignmentExpression",
  operator: "+=",
  right: varId,
  left: {
    type: "NumericLiteral",
    value: 2,
  },
}
```

Nodes building

Given a `varId` node, how to increment the value it represents by 2?

```
t.assignmentExpression(
  "+=",
  varId,
  t.numericLiteral(2)
);
```

```
{
  type: "AssignmentExpression",
  operator: "+=",
  right: varId,
  left: {
    type: "NumericLiteral",
    value: 2,
  },
}
```

Nodes building

Given a `varId` node, how to increment the value it represents by 2?

```
t.assignmentExpression(
  "+=",
  varId,
  t.numericLiteral(2)
);
```

```
{
  type: "AssignmentExpression",
  operator: "+=",
  right: varId,
  left: {
    type: "NumericLiteral",
    value: 2,
  },
}
```

Nodes building

Given a `varId` node, how to increment the value it represents by 2?

```
t.assignmentExpression(
  "+=",
  varId,
  t.numericLiteral(2)
);
```

```
{
  type: "AssignmentExpression",
  operator: "+=",
  right: varId,
  left: {
    type: "NumericLiteral",
    value: 2,
  },
}
```

Nodes building

Given a `varId` node, how to increment the value it represents by 2?

```
t.assignmentExpression(
  "+=",
  varId,
  t.numericLiteral(2)
);
```

```
{
  type: "AssignmentExpression",
  operator: "+=",
  right: varId,
  left: {
    type: "NumericLiteral",
    value: 2,
  },
}
```

Bonus package:

`@babel/template`

High level AST building

Given a `varId` node referencing an array, how to increment each of its elements by 2 and then take only the values greater than 10?

```
t.assignmentExpression("=", varId, t.callExpression(
    t.memberExpression(t.callExpression(
      t.memberExpression(varId, t.identifier("map")),
      [t.arrowFunctionExpression([t.identifier("val")],
                                 t.binaryExpression("+", t.identifier("val"), t.numericLiteral(2))
      )]
    ), t.identifier("filter")),
    [t.arrowFunctionExpression([t.identifier("val")],
                               t.binaryExpression(">", t.identifier("val"), t.numericLiteral(10))
    )]
  )
)
```

High level AST building

Given a `varId` node referencing an array, how to increment each of its elements by 2 and then take only the values greater than 10?

```
template.expression.ast`
  ${varId} = ${varId}
                  .map(val => val + 2)
                  .filter(val => val > 10)
`
```

High level AST building

Different parsing goals

```
template.expression

template.statement

template.statements

template.program
```

High level AST building

Different parsing goals

```
template.expression

template.statement

template.statements

template.program
```

Immediate usage...

```
ast = template.*.ast`${val} * 2`
```

... or deferred usage

```
build = template.*(`%%val%% * 2`)
// ...
ast = build({ val })
```

Plugins

Everything is a plugin

ECMAScript features	`@babel/plugin-transform-classes`
ECMAScript proposals	`@babel/plugin-proposal-optional-chaining`
ECMAScript extensions	`@babel/plugin-transform-typescript` `@babel/plugin-transform-react-jsx`
Optimization	`@babel/plugin-transform-runtime`

Everything is a plugin

babel-plugin-module-resolver

babel-plugin-macros

babel-plugin-transform-define

babel-plugin-emotion

babel-plugin-inferno

babel-plugin-add-module-exports

babel-plugin-istanbul

babel-plugin-react-css-modules

babel-plugin-react-intl-auto

babel-plugin-transform-async-to-promises

How to create a plugin

1. Create a function

```
function myPlugin(babel, options) {
  return {
    name: "my-plugin",
    visitor: {
      CallExpression(path) { /* ... */ }
    },
    manipulateOptions(babelOptions) {},
    inherits: require("another-plugin"),
  };
}
```

1. Create a function

```
function myPlugin(babel, options) {
  return {
    name: "my-plugin",
    visitor: {
      CallExpression(path) { /* ... */ }
    },
    manipulateOptions(babelOptions) {},
    inherits: require("another-plugin"),
  };
}
```

The first parameter exposes all the public API and utilities

```
// @babel/types
const t = babel.types;
```

1. Create a function

```
function myPlugin(babel, options) {
  return {
    name: "my-plugin",
    visitor: {
      CallExpression(path) { /* ... */ }
    },
    manipulateOptions(babelOptions) {},
    inherits: require("another-plugin"),
  };
}
```

The second parameter contains the options for this plugin defined in the user's config

2. Choose a name

Required

```
function myPlugin(babel, options) {
  return {
    name: "my-plugin",
    visitor: {
      CallExpression(path) { /* ... */ }
    },
    manipulateOptions(babelOptions) {},
    inherits: require("another-plugin"),
  };
}
```

Should match the plugin package name

```
babel-plugin-my-plugin
```

3. Define traversal visitor

Optional

```
function myPlugin(babel, options) {
  return {
    name: "my-plugin",
    visitor: {
      CallExpression(path) { /* ... */ }
    },
    manipulateOptions(babelOptions) {},
    inherits: require("another-plugin"),
  };
}
```

4. Modify Babel options

Optional

```
function myPlugin(babel, options) {
  return {
    name: "my-plugin",
    visitor: {
      CallExpression(path) { /* ... */ }
    },
    manipulateOptions(babelOptions) {},
    inherits: require("another-plugin"),
  };
}
```

It also handles options for `@babel/parser` and `@babel/generator`

```
opts.parserOpts
opts.generatorOprs
```

5. Extend another plugin

Optional

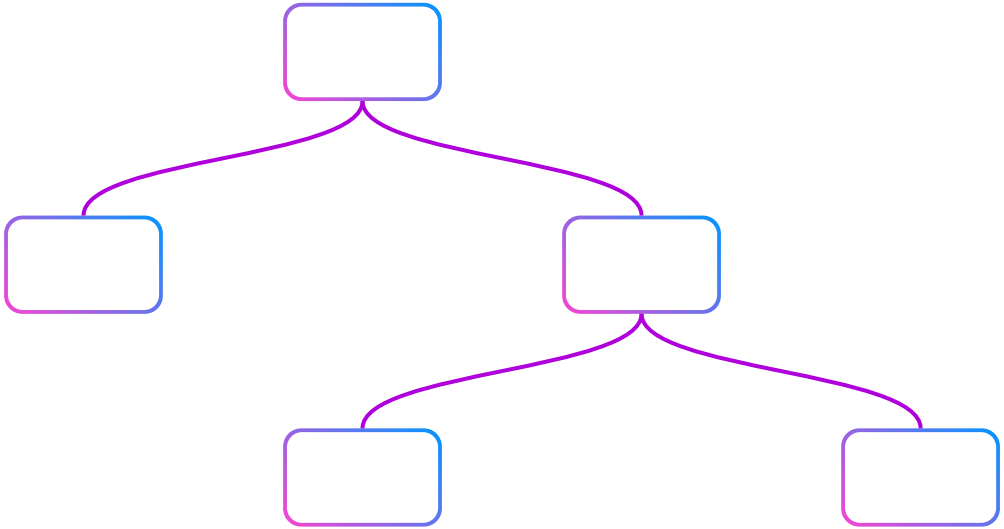
```
function myPlugin(babel, options) {
  return {
    name: "my-plugin",
    visitor: {
      CallExpression(path) { /* ... */ }
    },
    manipulateOptions(babelOptions) {},
    inherits: require("another-plugin"),
  };
}
```

A node with superpowers:

NodePath

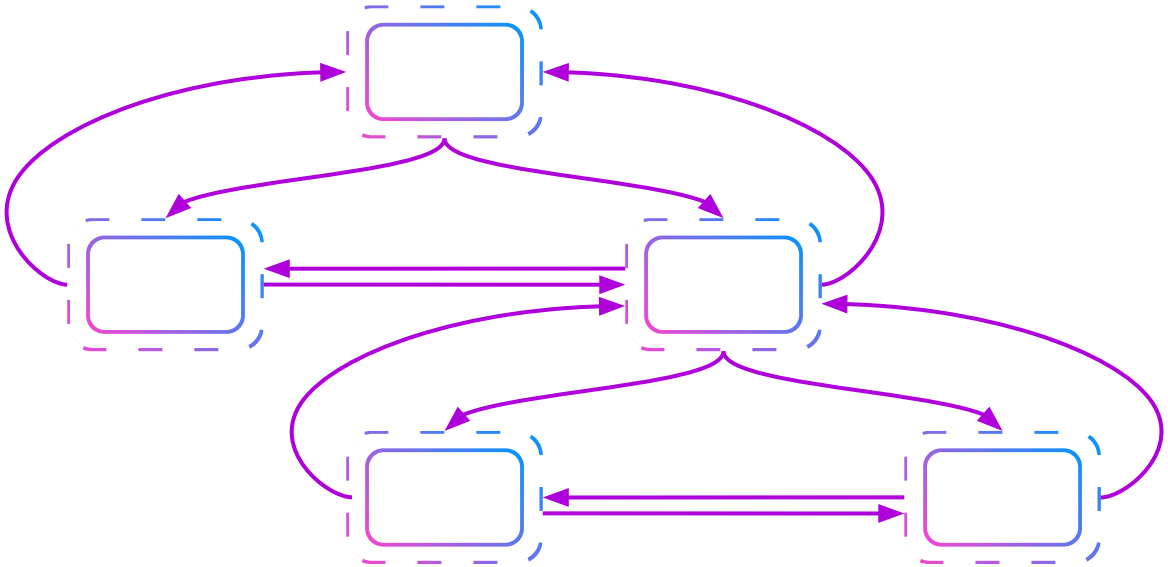
NodePath

Transformations need *context* and *ergonomics* for AST manipulation



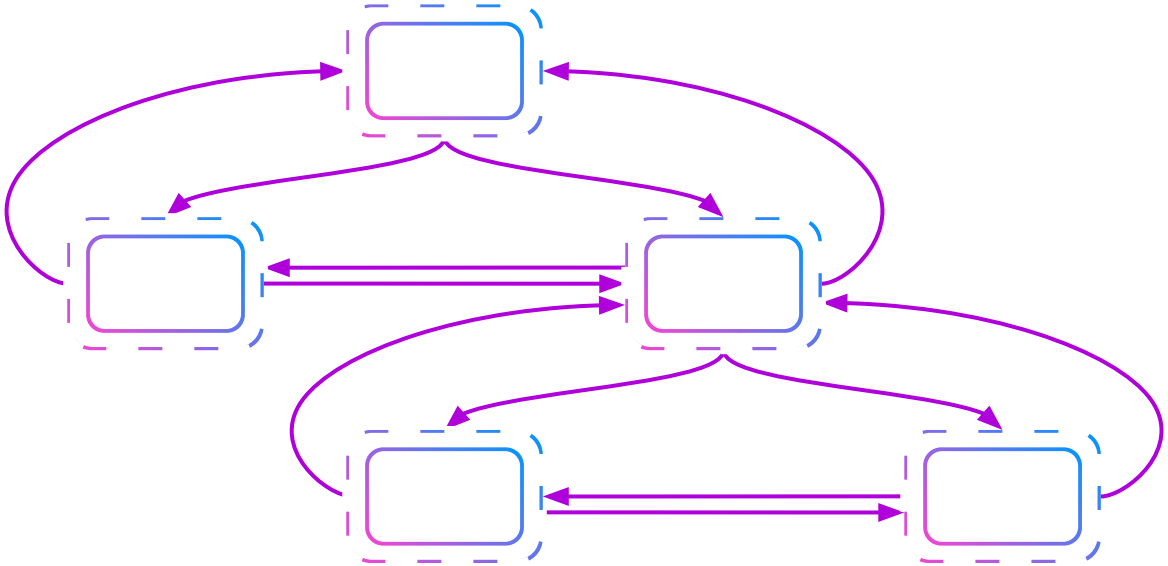
NodePath

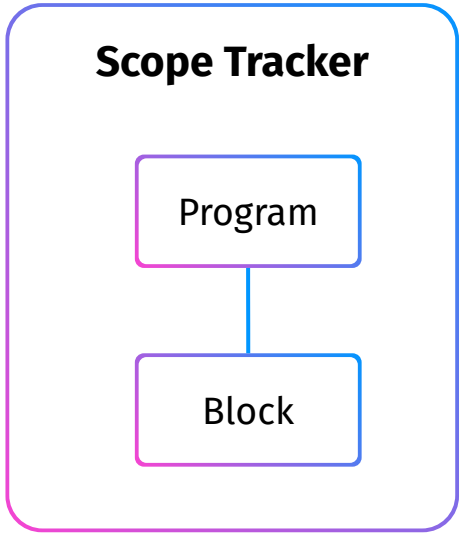
Transformations need *context* and *ergonomics* for AST manipulation



NodePath

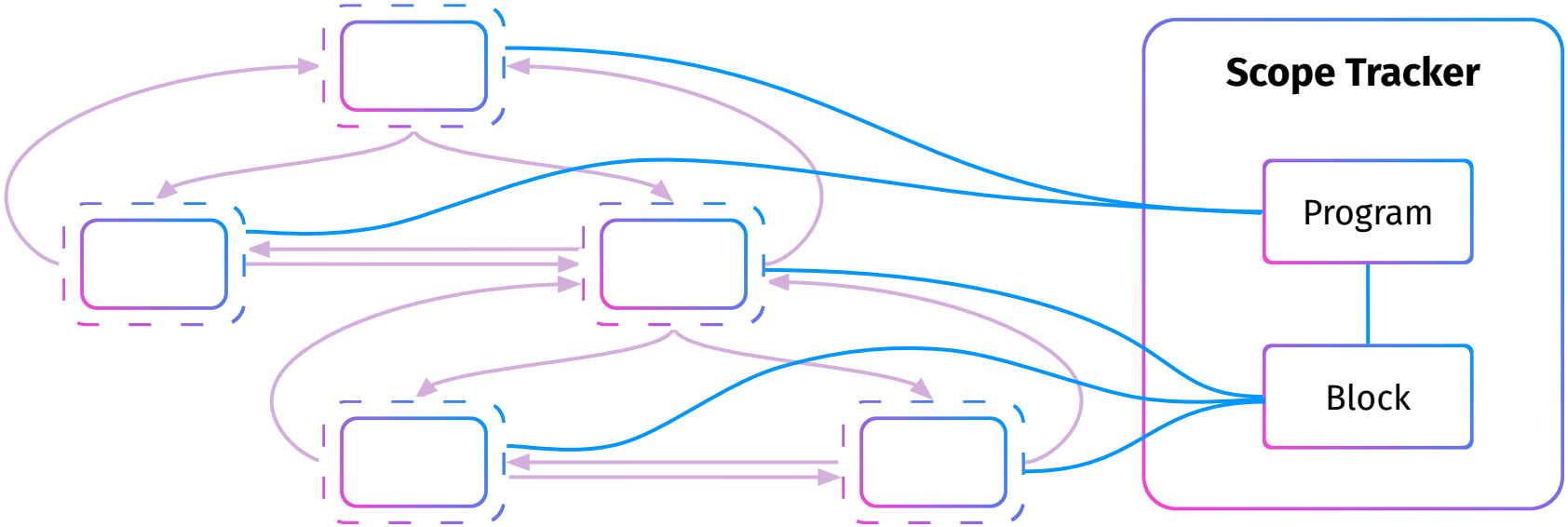
Transformations need *context* and *ergonomics* for AST manipulation





NodePath

Transformations need *context* and *ergonomics* for AST manipulation



NodePath

NodePath

`path.node` Get the original, unwrapped, node

NodePath

`path.node`	Get the original, unwrapped, node
`path.parentPath`	Get the path of parent node …
`path.get("body.0.id")`	.. or of a child node

NodePath

`path.node`	Get the original, unwrapped, node
`path.parentPath`	Get the path of parent node …
`path.get("body.0.id")`	.. or of a child node
`path.scope`	Get the scope of the current path

NodePath

`path.node`	Get the original, unwrapped, node
`path.parentPath`	Get the path of parent node …
`path.get("body.0.id")`	.. or of a child node
`path.scope`	Get the scope of the current path
`path.replaceWith(node)`	Replace the current node with another one …
`path.insertBefore(...nodes)`	… or just insert some new nodes before …
`path.insertAfter(...nodes)`	… or after

NodePath

`path.node`	Get the original, unwrapped, node
`path.parentPath`	Get the path of parent node …
`path.get("body.0.id")`	.. or of a child node
`path.scope`	Get the scope of the current path
`path.replaceWith(node)`	Replace the current node with another one …
`path.insertBefore(...nodes)`	… or just insert some new nodes before …
`path.insertAfter(...nodes)`	… or after
`path.toString()`	Call `@babel/generator`, useful when debugging

Scope

Scope

`scope.buildUndefinedNode()` Create a node which evaluates to `undefined`

`t.idenfier("undefined")` is not safe,
because users can have `var undefined = 2;`

Scope

`scope.buildUndefinedNode()`	Create a node which evaluates to `undefined` `t.idenfier("undefined")` is not safe, because users can have `var undefined = 2;`
`scope.generateUidIdentifier()`	Generate an identifier which is guaranteed not to conflict with existing variables
`scope.push({ id })`	Declare a variable in the current scope

Scope

`scope.buildUndefinedNode()`	Create a node which evaluates to `undefined` `t.idenfier("undefined")` is not safe, because users can have `var undefined = 2;`
`scope.generateUidIdentifier()`	Generate an identifier which is guaranteed not to conflict with existing variables
`scope.push({ id })`	Declare a variable in the current scope
`scope.getBinding(name)` `scope.hasBinding(name)`	Get information about the currently defined bindings

Case study

Optional chaining proposal

`obj?.prop`

Optional chaining

Optionally get properties from possibly `null` or `undefined` objects:

```
var street = user.address && user.address.street;
```

```
var street = user.address?.street;
```

Optional chaining

Optionally get properties from possibly `null` or `undefined` objects:

```
var street = user.address && user.address.street;

var street = user.address?.street;
```

Also works with nested properties:

```
var _tmp = a.b && a.b[3].c(x);
var result = _tmp && _tmp.d;

var result = a.b?.[3].c(x)?.d;
```

NICOLÒ RIBAUDO

Babel team

@NicoloRibaudo

nicolo.ribaudo@gmail.com

@nicolo-ribaudo

Links

- ➔ Babel AST specification
  https://github.com/babel/babel/blob/master/packages/babel-parser/ast/spec.md
- ➔ @babel/types builders API
  https://babeljs.io/docs/en/babel-types
- ➔ Optional chaining proposal
  https://github.com/tc39/proposal-optional-chaining
- ➔ babel-plugin-tester
  https://github.com/babel-utils/babel-plugin-tester
- ➔ GitHub repository
  https://github.com/nicolo-ribaudo/conf-holyjs-moscow-2019