

BSON-Transpilers

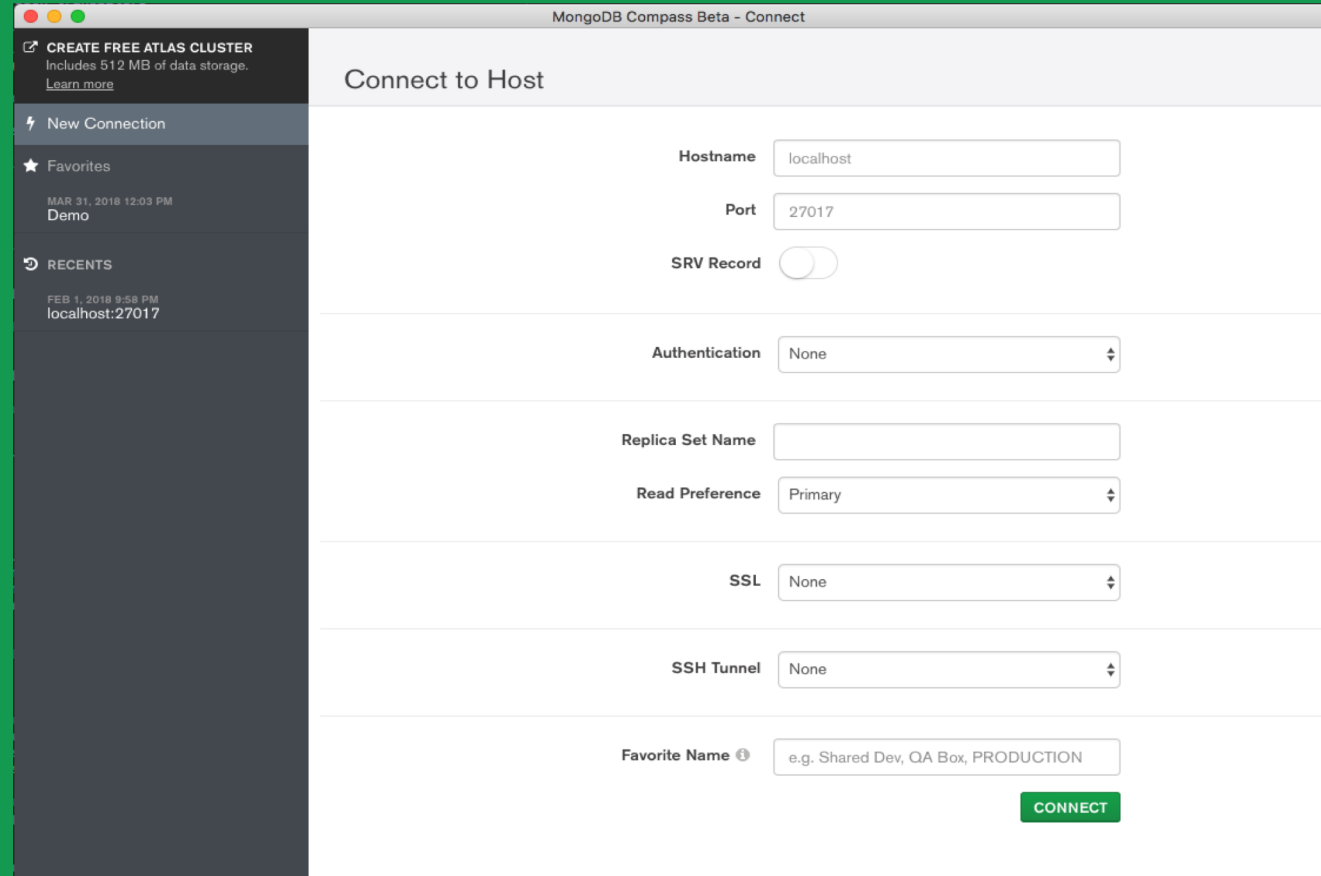
Anna Herlihy
Senior Software Engineer
Stockholm

 @annaisworking

1. Feature Requirements
2. Technical Requirements
3. How to contribute!



Compass = The UI for MongoDB



The screenshot shows the 'MongoDB Compass Beta - Connect' window. On the left is a sidebar with a dark grey background containing the following sections:

- CREATE FREE ATLAS CLUSTER**: Includes 512 MB of data storage. [Learn more](#)
- New Connection**: Represented by a lightning bolt icon.
- Favorites**: Represented by a star icon. It lists a connection: 'MAR 31, 2018 12:03 PM Demo'.
- RECENTS**: Represented by a circular arrow icon. It lists a connection: 'FEB 1, 2018 9:58 PM localhost:27017'.

The main area of the window is titled 'Connect to Host' and contains the following fields:

- Hostname**: A text input field containing 'localhost'.
- Port**: A text input field containing '27017'.
- SRV Record**: A toggle switch, currently turned off.
- Authentication**: A dropdown menu with 'None' selected.
- Replica Set Name**: An empty text input field.
- Read Preference**: A dropdown menu with 'Primary' selected.
- SSL**: A dropdown menu with 'None' selected.
- SSH Tunnel**: A dropdown menu with 'None' selected.
- Favorite Name**: A text input field with a help icon (i) and placeholder text 'e.g. Shared Dev, QA Box, PRODUCTION'.

A green **CONNECT** button is located at the bottom right of the form.

Export To Language (query)

mbw.answers_etl

DOCUMENTS	338	TOTAL SIZE	41.0KB	AVG. SIZE	124B
INDEXES	1	TOTAL SIZE	36.0KB	AVG. SIZE	36.0KB

Documents Aggregations Schema Explain Plan Indexes Validation

FILTER `{answer: {$type: 'array'}}` **OPTIONS** **FIND** **RESET** **...**

INSERT DOCUMENT **VIEW** **LIST** **TABLE** Displaying documents

Export To Language
Toggle Query History

```
_id: ObjectId("5b159983f8e55ca3b2cc9304")
  answer: Array
    0: 1
    1: 0
    2: 0
    3: 0
    4: 1
    5: 0
  problem_id: "5963b30cc1da5a32116dc5ae"
```

Export To Language (aggregation)



mdbw.answers_etl

DOCUMENTS 338 TOTAL SIZE 41.0KB AVG. SIZE 124B INDEXES 1 TOTAL SIZE 36.0KB AVG. SIZE 36.0KB

Documents Aggregations Schema Explain Plan Indexes Validation

> Enter a pipeline name... SAVE PIPELINE ...

COMMENT MODE SAMPLE MODE AUTO PREVIEW Unsaved changes

Export To Language
Clone Pipeline
New Pipeline
New Pipeline From Text

\$match

```
1 /**
2  * query - The query in MQL.
3  */
4 {
5   answer: [{ $type: 'array' } ]
6 }
```

Output after \$match stage (Sample of 20 documents)

```
{
  "_id": ObjectId("5b159983f8e55ca3b2cc9304"),
  "answer": Array
    0: 1
    1: 0
    2: 0
    3: 0
    4: 1
    5: 0
  "problem_id": "5963b30cc1da5a32116dc5ae"
}
```

Also used on MongoDB Atlas

The screenshot displays the MongoDB Atlas web interface. The browser address bar shows the URL: `https://cloud.mongodb.com/v2/57f303493b34b93fbdeb7649#metrics/replicaSet/5a9d56f14e6581425e109a86/explorer/test/users/aggregation`. The interface includes a sidebar with navigation options like Clusters, Security, Project, Services, and Help. The main content area shows the 'testusers' collection with 794 documents and 44KB of indexes. A tooltip 'Export pipeline code to language' is visible over the 'COLLATION' button. The 'Preview of Documents in the Collection' section shows two sample documents:

```
{ "_id": ObjectId("5bd3098bc4b7b2bcce104f2f"),  
  "phone": "966.226.2487 x159",  
  "website": "coleman.info",  
  "company": Object,  
  "name": "Hobart Greenholt",  
  "username": "Keaton.Koepp",  
  "age": 50,  
  "email": "Howell174@gmail.com",  
  "address": Object }  
  
{ "_id": ObjectId("5bd3098bc4b7b2bcce104f40"),  
  "company": Object,  
  "country": "Gibraltar",  
  "name": "Amely Lowe Jr.",  
  "username": "Vivienne_Crooks",  
  "email": "Marlee68@gmail.com",  
  "address": Object,  
  "phone": "990-245-5297 x351",  
  "website": "icrael.name" }
```

The bottom section shows an aggregation pipeline stage with a 'Select...' dropdown and an 'ADD STAGE' button. The right side of the aggregation stage shows 'No Preview Documents'.

But wait...

**Wouldn't it be great if you could just write
whatever language you want, directly into
Compass?**

Language-Modes

MongoDB Compass - compass-data-sets-e06dc.mongodb.net:27017/citibike.trips

compass-data-sets-shard-0 REPLICA SET **3 NODES** MongoDB 3.6.8 Enterprise

citibike.trips

DOCUMENTS	2.1m	TOTAL SIZE	873.3MB	AVG. SIZE	439B
INDEXES	7	TOTAL SIZE	114.3MB	AVG. SIZE	16.3MB

Documents Aggregations Schema Explain Plan Indexes Validation

> Enter a pipeline name... SAVE PIPELINE ...

2084654 L

- Switch Language
- Export To Language
- Clone Pipeline
- New Pipeline
- New Pipeline From Text

Select an operator to construct expressions used in the aggregation pipeline stages. [Learn more](#)

☒ COMMENT MODE ☒ SAMPLE MODE ☒ AUTO PREVIEW Unsaved changes

Preview of Documents in the Collection

```
{
  "start station longitude": -74.00070227,
  "end station id": 259,
  "end station name": "South St & Whitehall St",
  "end station latitude": 40.70122128,
  "end station longitude": -74.01234218,
  "bikeid": 18534,
  "usertype": "Subscriber",
  "birth year": 1997,
  "gender": 1,
  "_id": ObjectId("5a9e9620c628c0a06c09"),
  "tripduration": 207,
  "starttime": "2018-01-01 00:02:44",
  "stoptime": "2018-01-01 00:06:11",
  "start station id": 3224,
  "start station name": "W 13 St & Huds",
  "start station latitude": 40.73997354,
  "start station longitude": -74.005138,
  "end station id": 170
}
```

BSON is a large enough subset to treat the problem as if we're parsing the entire language syntax


Requirements

Accept query or aggregation in any language

+

Export query or aggregation to any language

=

A decorative border made of pink and purple flowers and swirling vines surrounds the central text. The border is composed of various floral motifs, including large daisies, smaller blossoms, and elegant, curving stems with leaves.

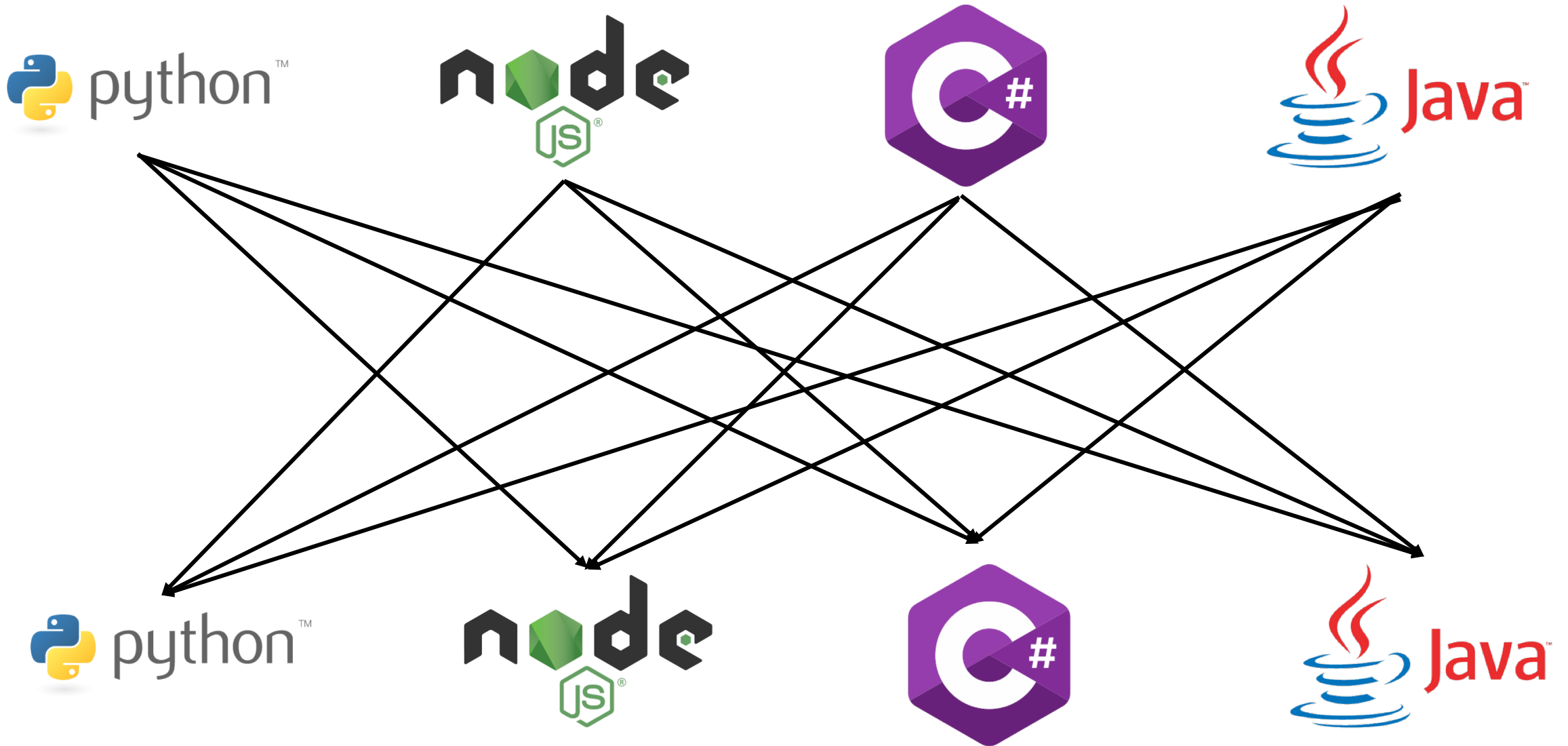
**Any language
to
any language
translation!**

Possible Approaches

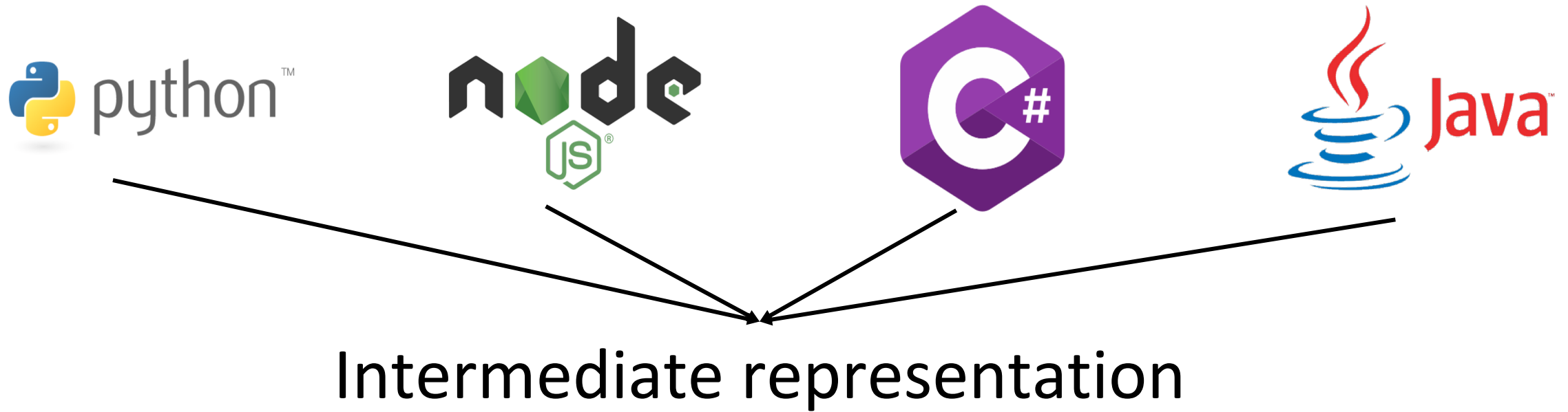




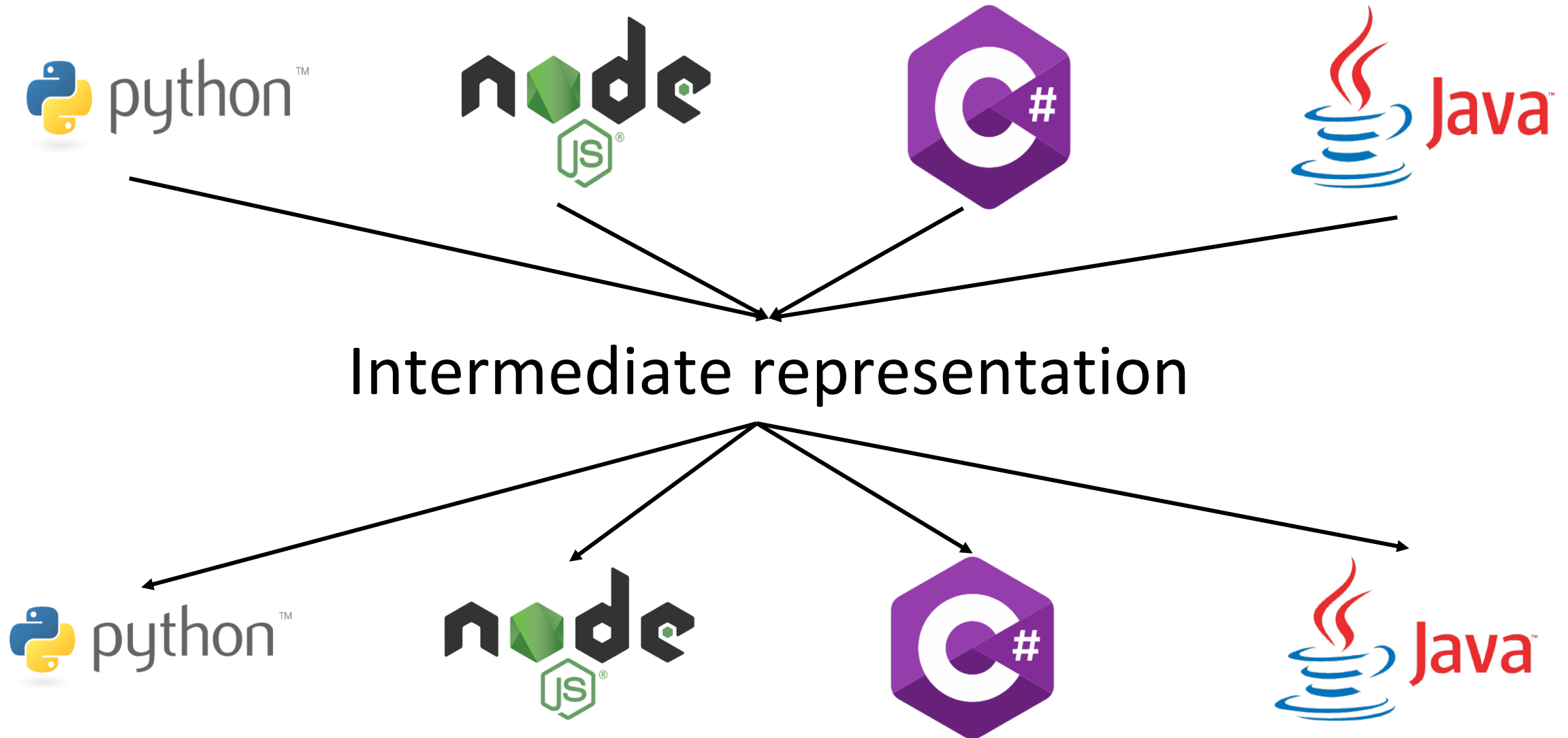
Naïve Approach



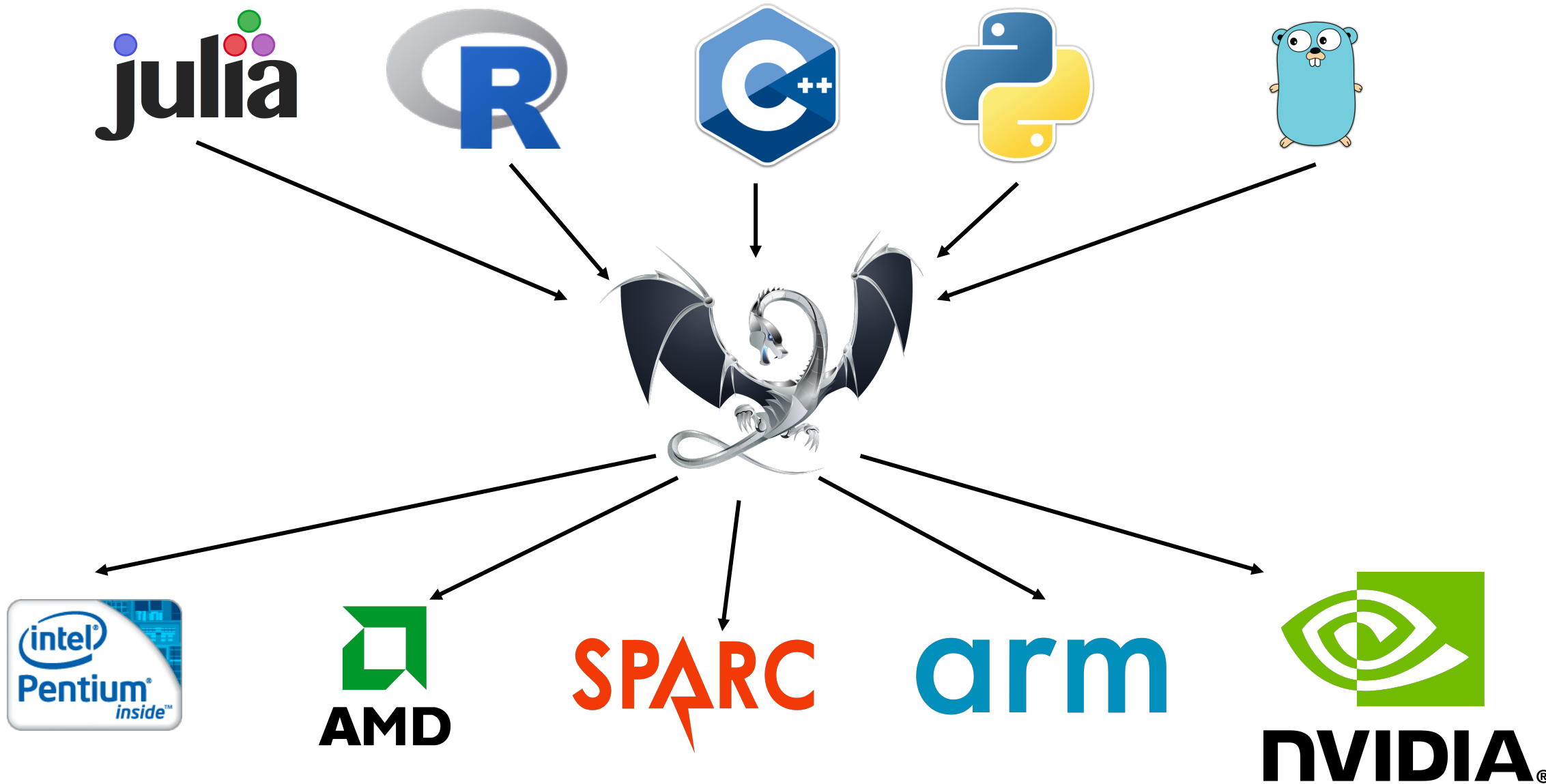
Intermediate Representation (IR)



Intermediate Representation (IR)



LLVM-IR



Linear Regression in Python

```
def linreg(data=listf100, w=listf100, g=listf100, dims=int):  
    dot = 1.0  
    c = 0  
    while c<dims:  
        dot+=data[c]*w[c]  
        c+=1  
    label = data[dims]  
    dot = dot* -label  
    c2=0  
    while(c2<dims):  
        g[c2] += dot*data[c2]  
        c2+=1
```


Linear Regression in LLVM-IR

```
define void @linreg([100 x float]* %data, [100 x float]* %w, [100 x float]* %g, i32 %dims) {
```

```
entry:
```

```
    %tmp48 = alloca [100 x float]*
    store [100 x float]* %data, [100 x float]** %tmp48
    %tmp49 = alloca [100 x float]*
    store [100 x float]* %w, [100 x float]** %tmp49
    %tmp50 = alloca [100 x float]*
    store [100 x float]* %g, [100 x float]** %tmp50
    %tmp51 = alloca i32
    store i32 %dims, i32* %tmp51
    %tmp52 = alloca float
    store float 1.000000e+00, float* %tmp52
    %tmp53 = load float* %tmp52
    %dot = alloca float
    store float %tmp53, float* %dot
    %tmp54 = alloca i32
    store i32 8, i32* %tmp54
    %tmp55 = load i32* %tmp54
    %c = alloca i32
    store i32 %tmp55, i32* %c
    br label %start_while
```

```
start_while:
```

```
    %tmp56 = load i32* %c
    %tmp57 = load i32* %tmp51
    %cmptmp = icmp slt i32 %tmp56, %tmp57
    %booltmp = uitofp i1 %cmptmp to float
    %whilecond = fcmp one float %booltmp, 0.000000e+00
    br i1 %whilecond, label %do_while, label %end_while
```

```
do_while:
```

```
    %tmp58 = load [100 x float]** %tmp48
    %tmp59 = load i32* %c
    %0 = getelementptr [100 x float]:
    %tmp60 = load float* %0
    %tmp61 = load [100 x float]** %tmp49
    %tmp62 = load i32* %c
    %1 = getelementptr [100 x float]:
    %tmp63 = load float* %1
    %tmp64 = fmul float %tmp60, %tmp63
    %tmp65 = load float* %dot
    %2 = fadd float %tmp64, %tmp65
    %tmp66 = load float* %dot
    store float %2, float* %dot
    %tmp67 = alloca i32
    store i32 1, i32* %tmp67
    %tmp68 = load i32* %tmp67
    %tmp69 = load i32* %c
    %3 = add i32 %tmp69, %tmp68
    %tmp70 = load i32* %c
    store i32 %3, i32* %c
    br label %start_while
```

```
end_while:
```

```
    %tmp71 = load [100 x float]** %tmp48
    %tmp72 = load i32* %tmp51
    %4 = getelementptr [100 x float]* %tmp71, i32 0, i32 %tmp72
    %tmp73 = load float* %4
    %label = alloca float
    store float %tmp73, float* %label
    %tmp74 = load float* %dot
    %tmp75 = load float* %label
    %tmp76 = fsub float 0.000000e+00, %tmp75
    %tmp77 = fmul float %tmp74, %tmp76
    store float %tmp77, float* %dot
    %tmp78 = alloca i32
    store i32 0, i32* %tmp78
    %tmp79 = load i32* %tmp78
    %c2 = alloca i32
    store i32 %tmp79, i32* %c2
    br label %start_while1
```

```
define void @main() {
```

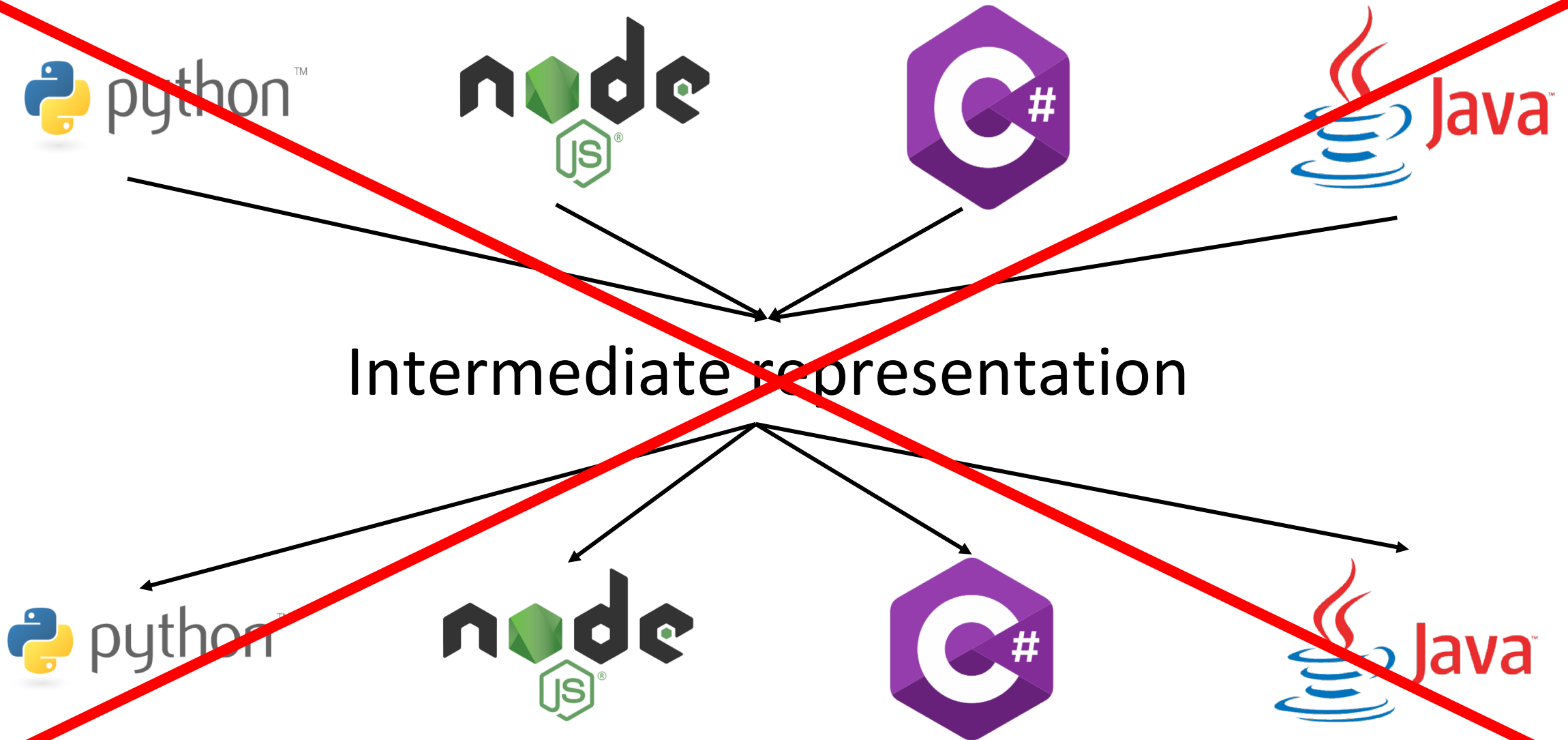
```
entry:
```

```
    %tmp112 = alloca float
    store float 1.000000e+02, float* %tmp112
    %tmp113 = load float* %tmp112
    %0 = alloca [100 x float], i32 100
    %1 = getelementptr [100 x float]* %0, i32 0, i32 0
    store float 0.000000e+00, float* %1
```

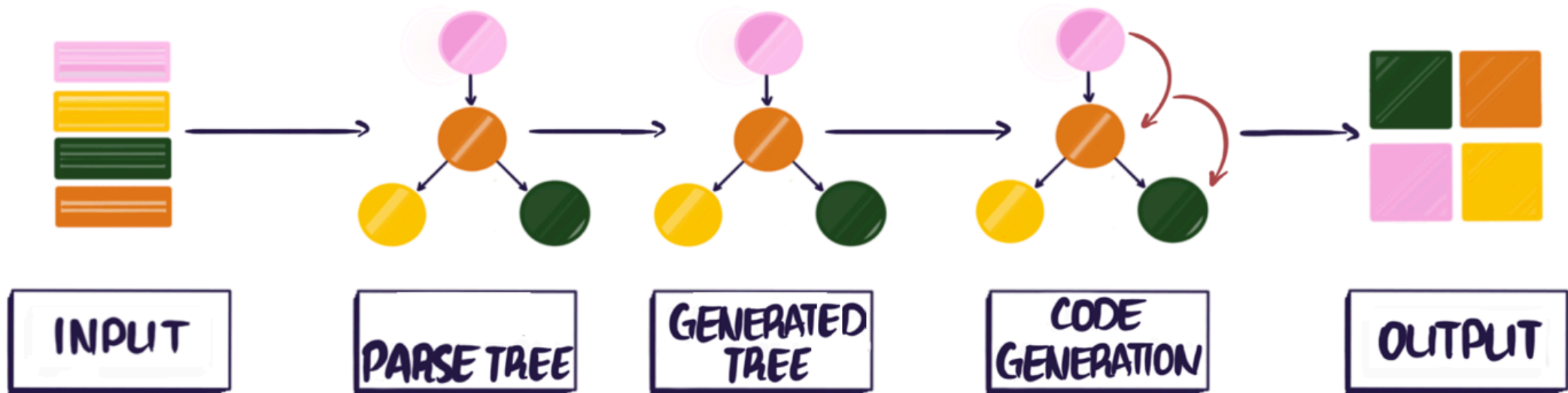
```
; preds = %start_while
```

```
%g = alloca [100 x float]*
store [100 x float]* %205,
[tmp124 = load [100 x float]
[tmp125 = load [100 x float]
[tmp126 = load [100 x float]
[tmp127 = alloca i32
store i32 99, i32* %tmp127
[tmp128 = load i32* %tmp127
call void @linreg([100 x fl
ret void
```

Intermediate Representation (IR)



How does Babel work?

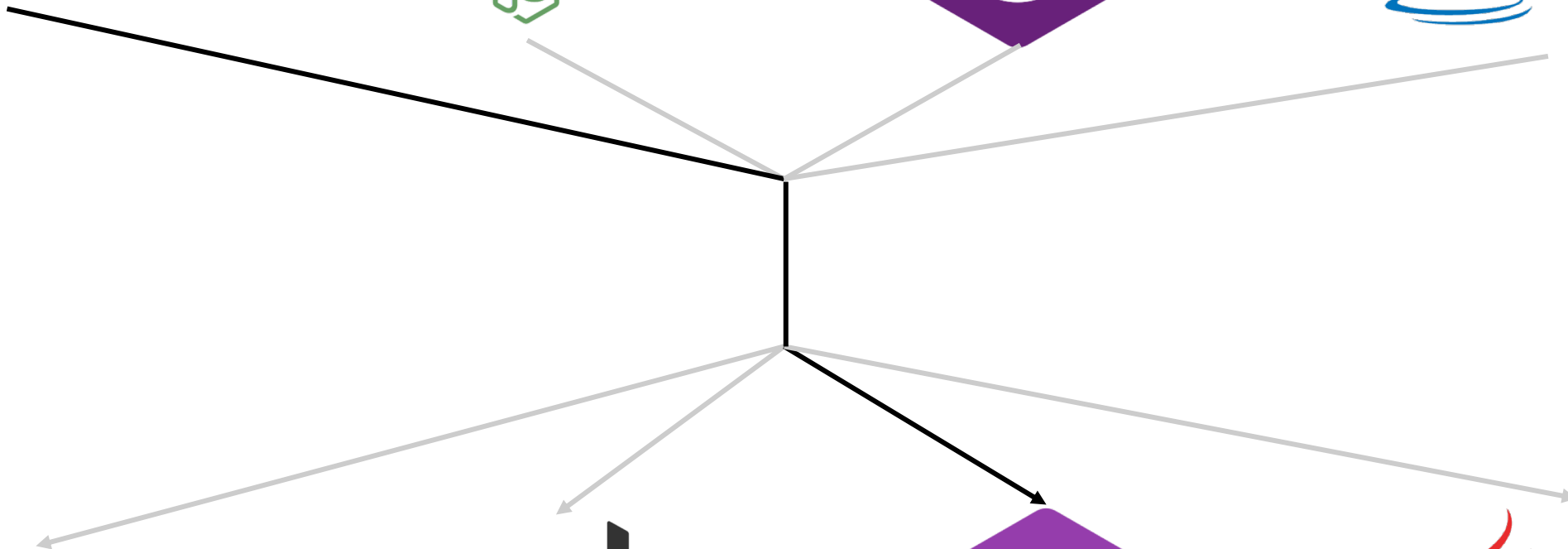


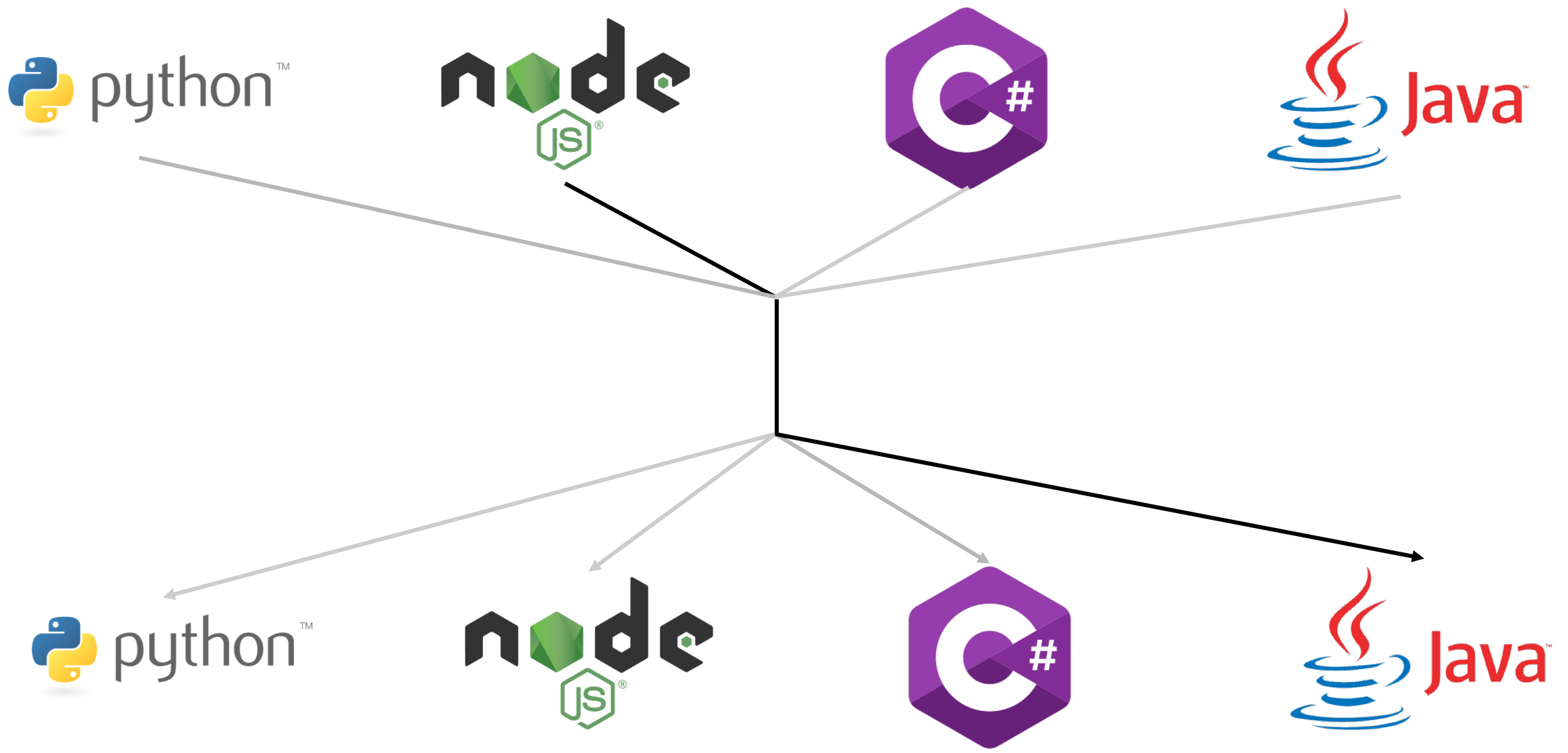
Babel Parser only parses JavaScript 😞

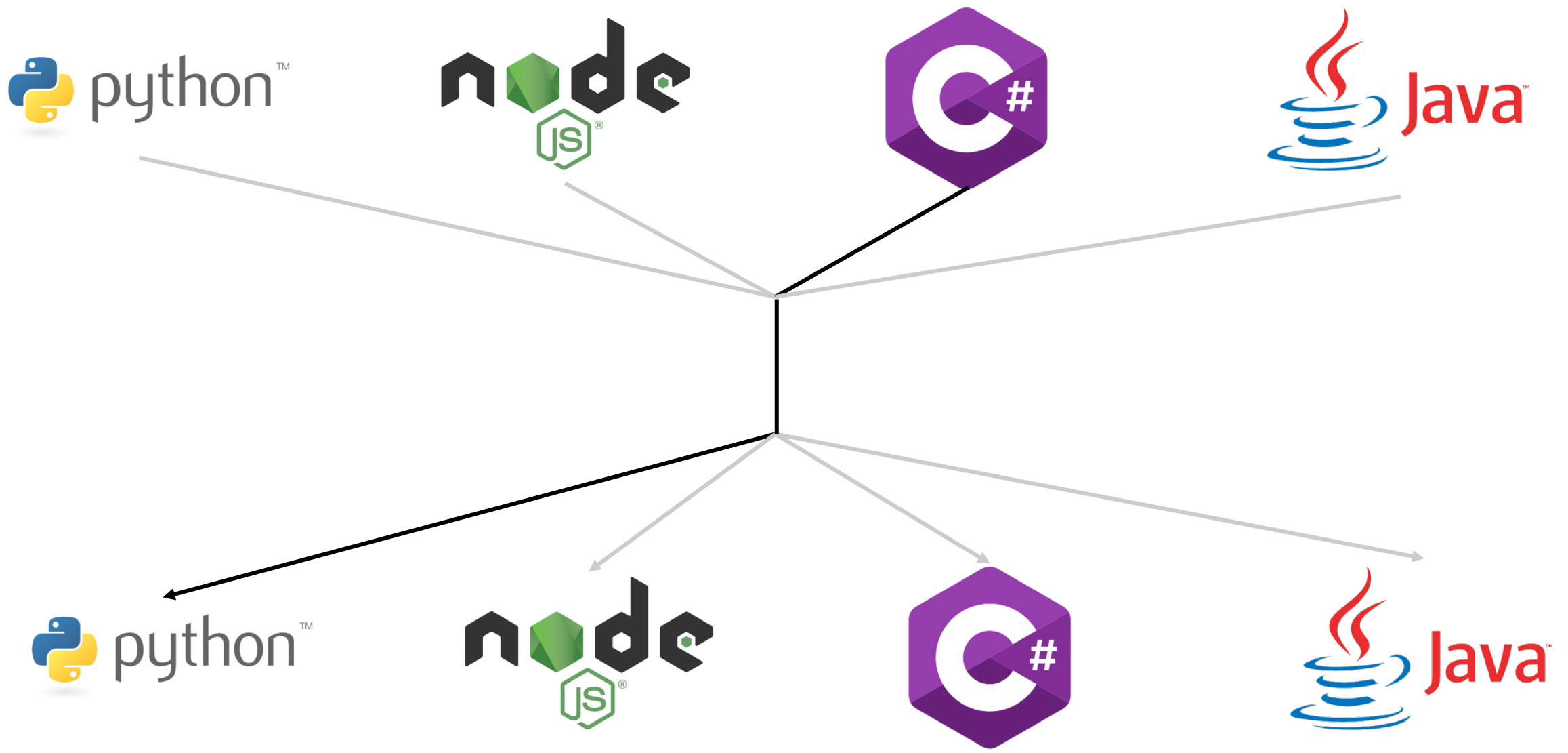


???









Complexity



For every input language we support:

1. **We want to only have to do the work once**
2. We want to define the input language without knowing or caring how many output languages exist

We do not want to have to
rewrite the translation for
every possible combination,
that would be $O(n^2)$ and we
want $O(n)$

For every input language we support:

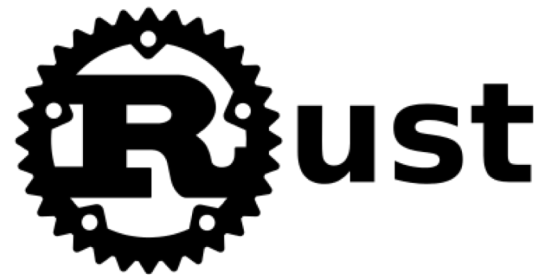
1. We want to only have to do the work once
2. We want to define the input language without knowing or caring how many output languages exist



For every input language we support:

1. We want to only have to do the work once
2. We want to define the input language without knowing or caring how many output languages exist

Same principle for target languages



Distributed Development

1. Many communities are small + passionate
2. Open source ethos :)
3. How many people are compiler experts?

Summary of Technical Requirements

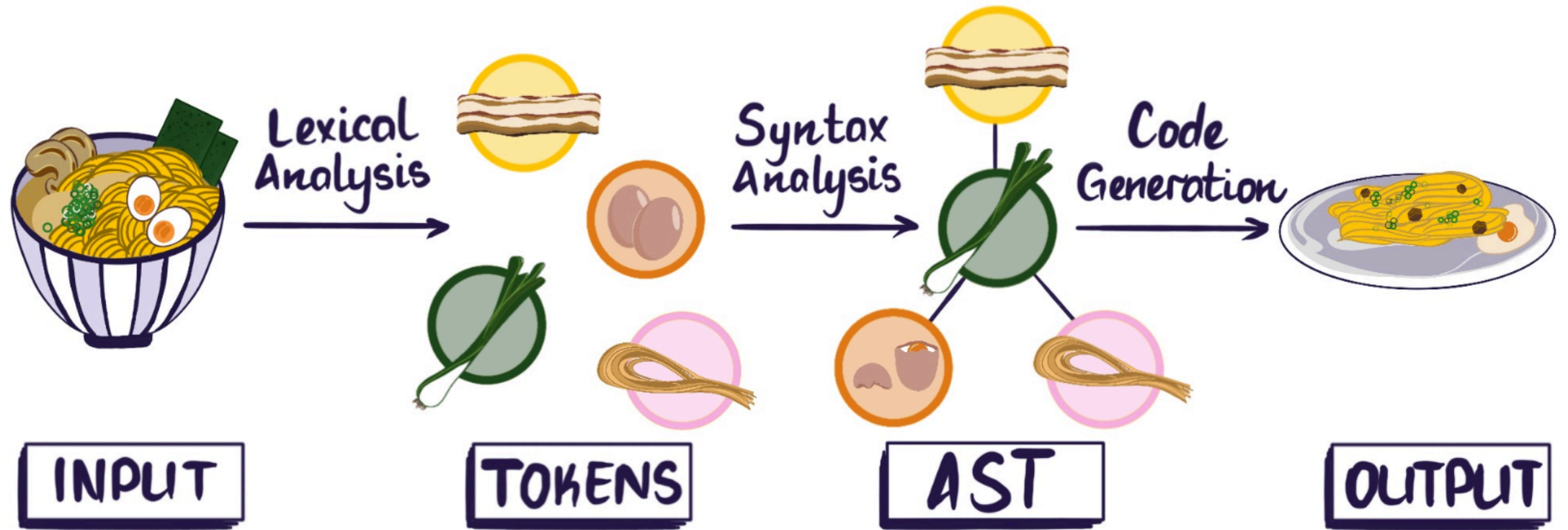
1. Accept arbitrary # of input languages
2. Generate arbitrary # of output languages
3. Support distributed development
4. Linear-time development cost
5. Web-friendly JavaScript library

How to add your own output language



Compiler 101

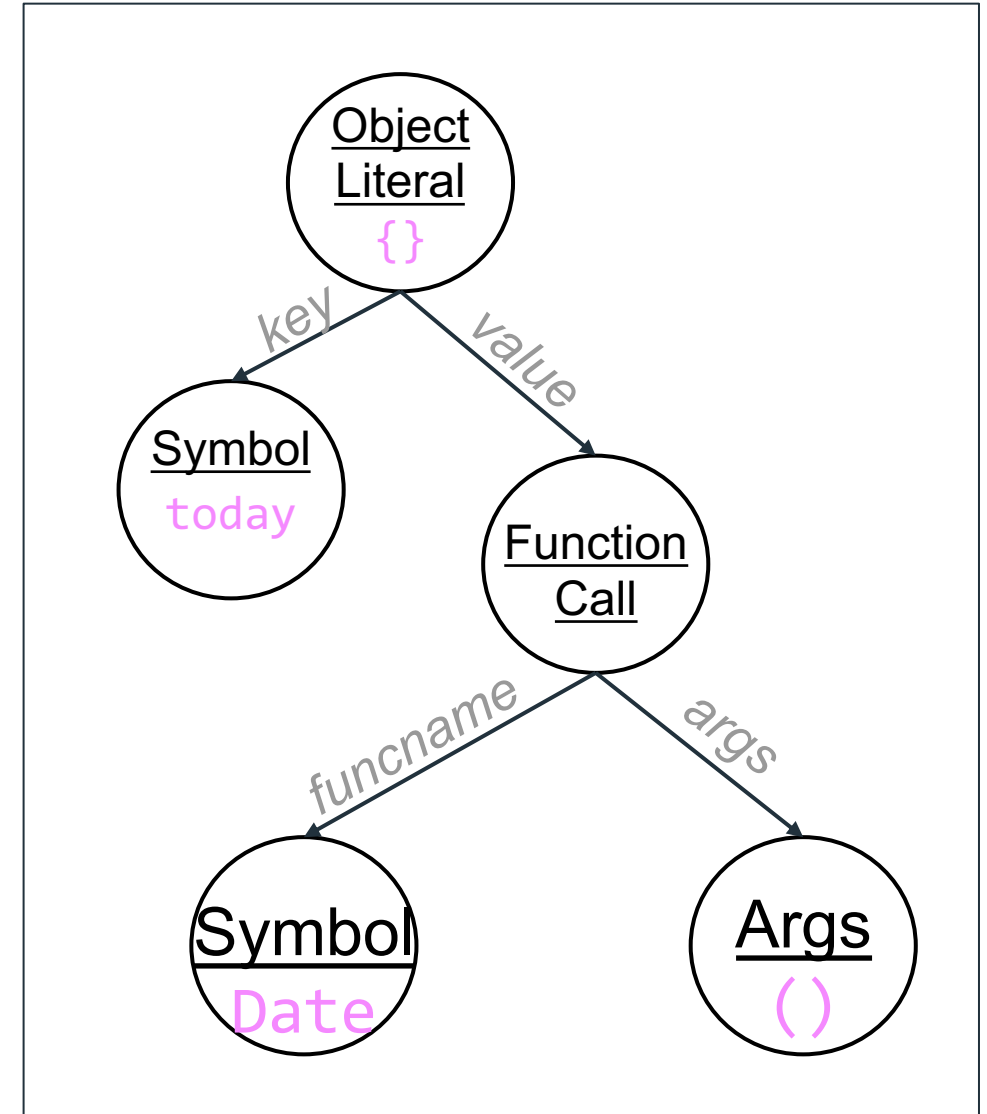
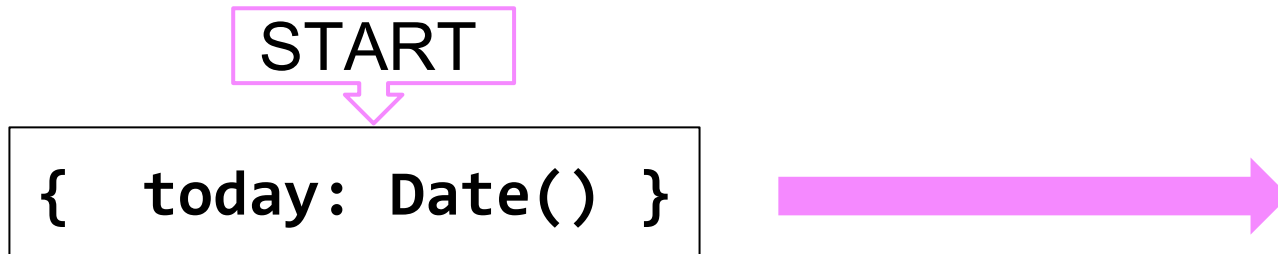


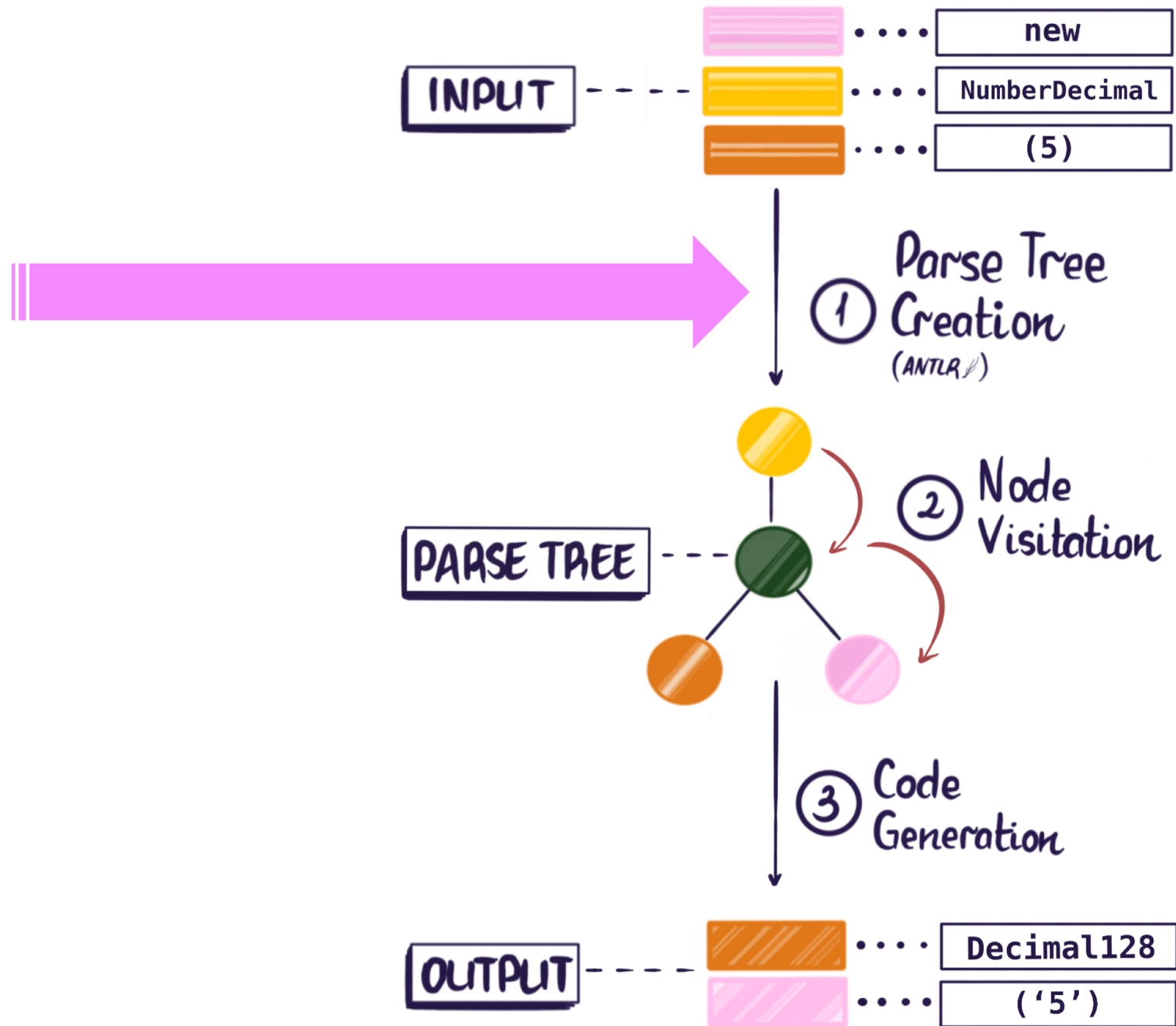


Illustrations by @Irina!

Tree Building

The tree-building stage includes lexing, parsing, syntax analysis, and more!





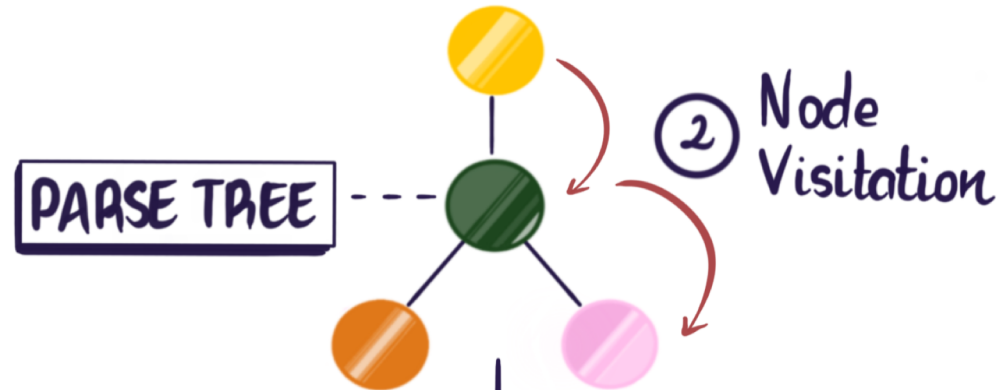
- Parsers are hard!
- Support for most languages
- We apply ANTLR to our input and we get a tree. ***The tree building stage is completely handled!***
- Bonus: ANTLR generates a visitor class!

OUR SAVIOR





① Parse Tree Creation
(ANTLR)



③ Code Generation



The Visitor Pattern

Visitors traverse trees by “*visiting*” each node

For each type of node, the visitor calls the corresponding function.

- When the visitor sees a node that is a “string” type, it will call the **visitString** method and expect the generated code to be returned.

Extremely Simple Visitor Example

START



`'testing...testing'`

JavaScript Code

Extremely Simple Visitor Example

START

`'testing...testing'`

JavaScript Code

END

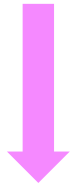
Java Code

`"testing...testing"`

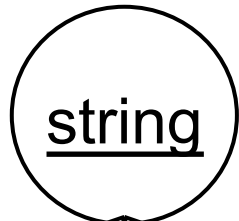
Extremely Simple Visitor Example

START

`'testing...testing'`



ANTLR



value

`'`

`'`

`testing...testing`

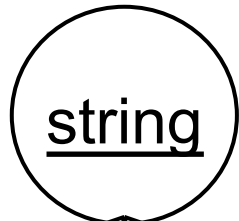
END

`"testing...testing"`

Extremely Simple Visitor Example

START

'testing...testing'



value

'

'

testing...testing

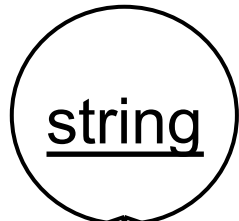
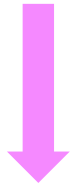
END

"testing...testing"

Extremely Simple Visitor Example

START

`'testing...testing'`



value

`'`

`'`

`testing...testing`

VISITOR



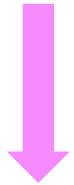
END

`"testing...testing"`

Extremely Simple Visitor Example

START

'testing...testing'



string

value

'

'

testing...testing

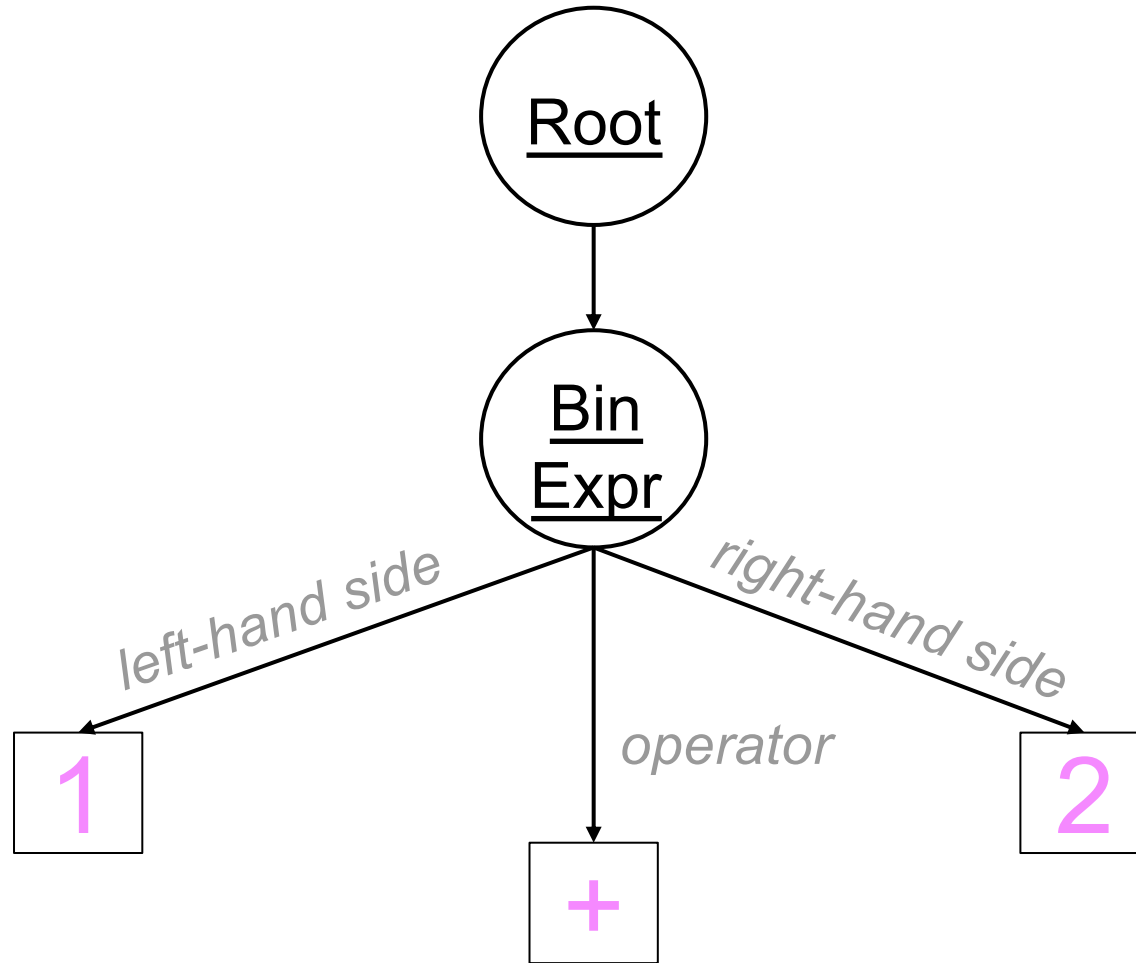
```
class Visitor() extends ANTLRVisitor {  
    visitString(node) {  
        const val = node.value;  
        return "" + val + "";  
    }  
}
```

END

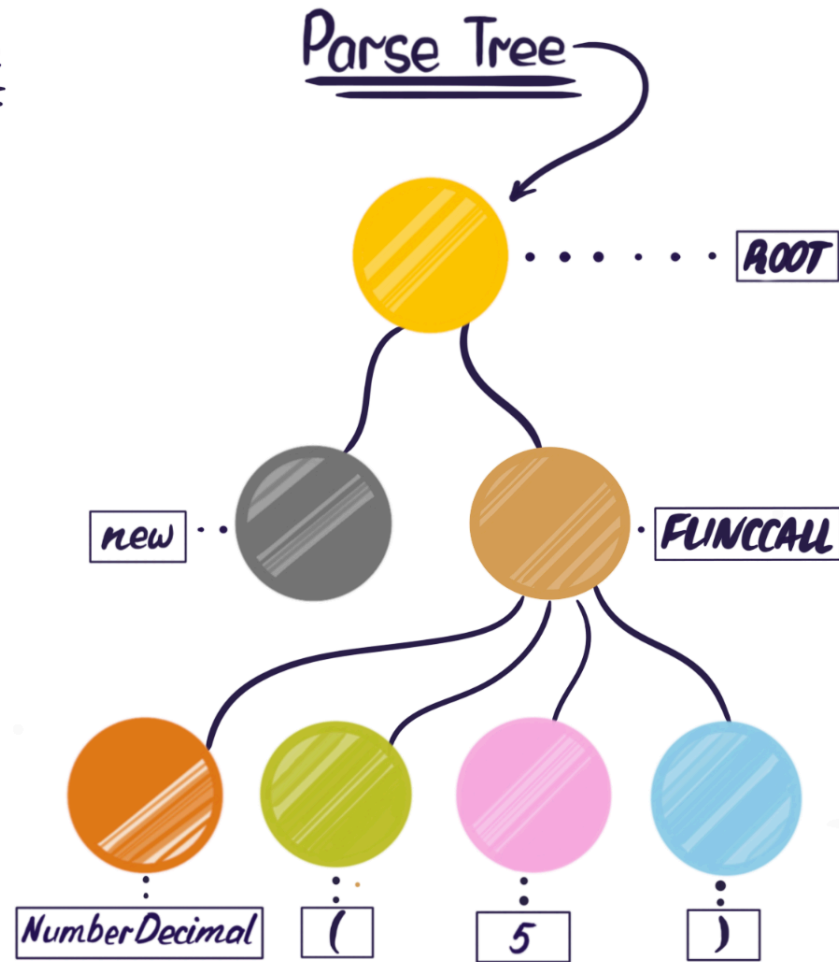
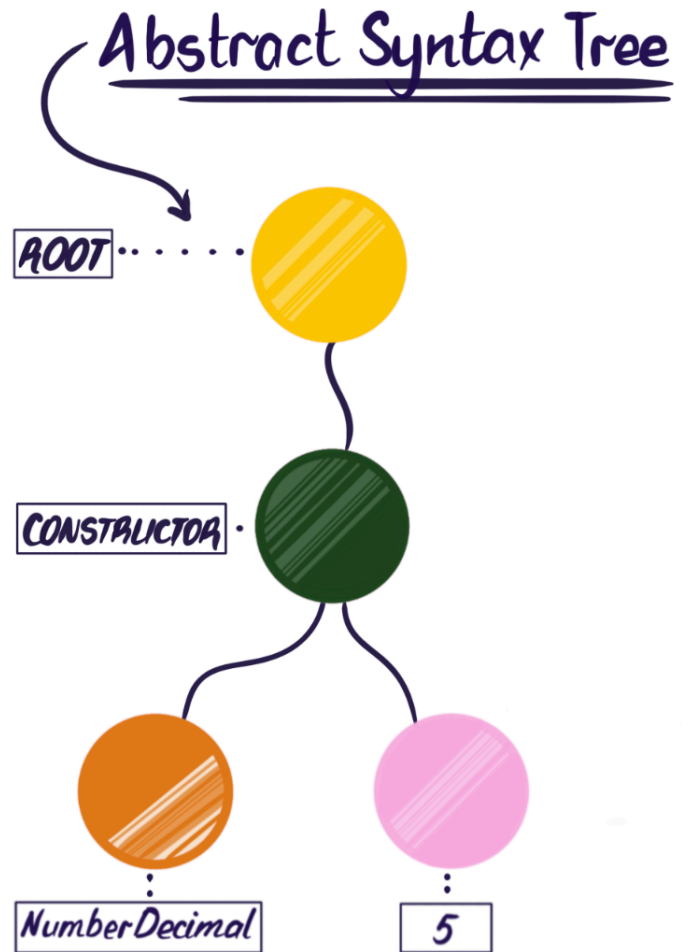
"testing...testing"



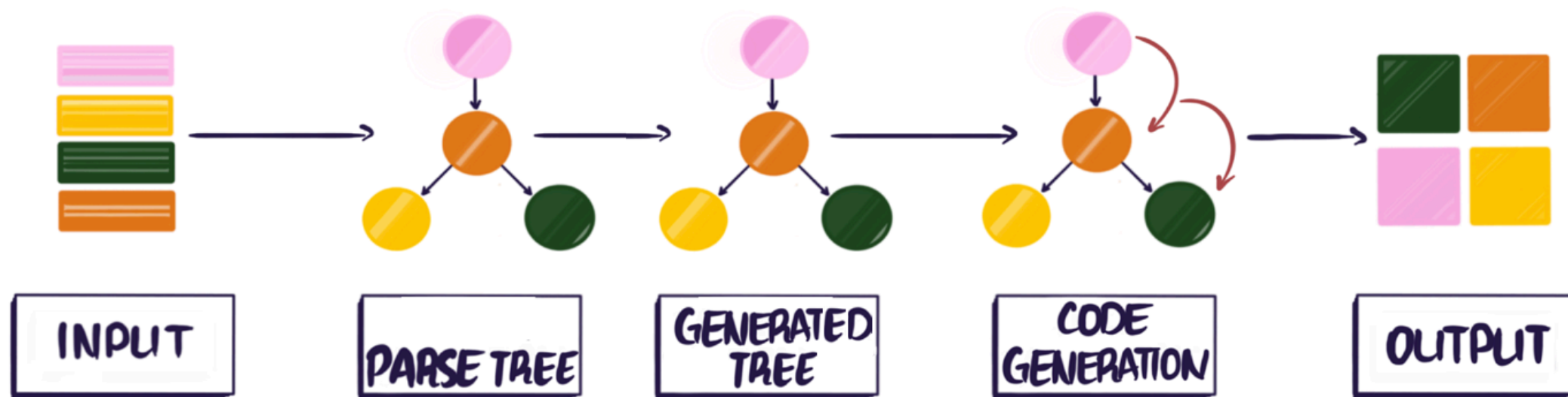
What does ANTLR look like?



Parse Tree vs AST

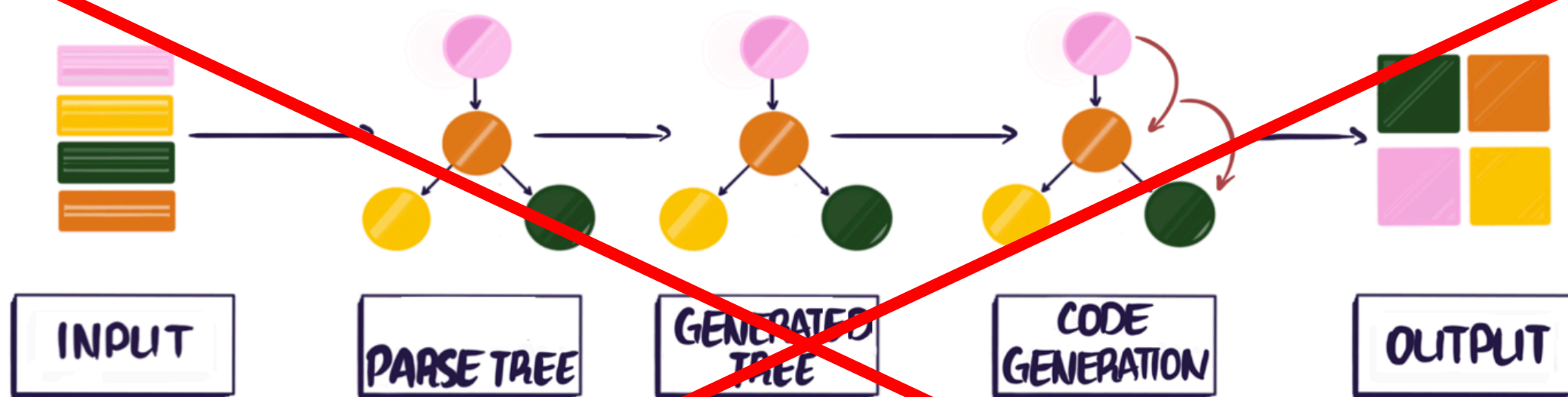


How does Babel work?

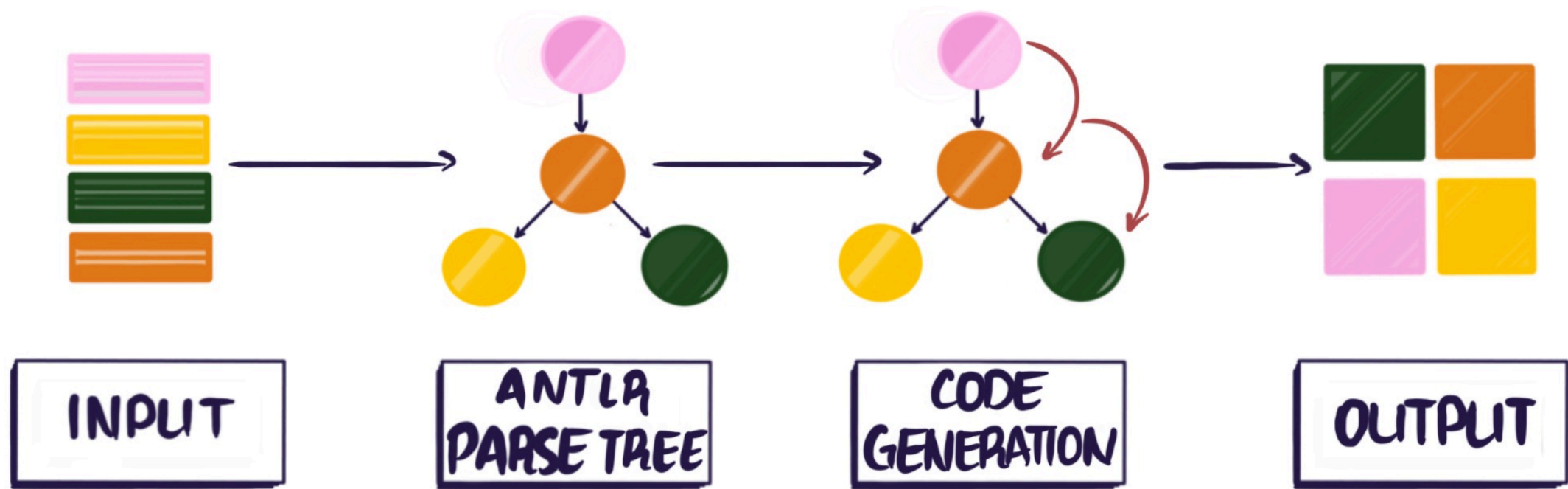


ANTLR trees are read-only ☹️

How does Babel work?



ANTLR trees are read-only ☹️



ANTLR-generated Visitor Classes

```
class ECMAScriptVisitor {  
    // Visit a parse tree produced by ECMAScriptParser#LogicalOrExpression.  
    visitLogicalOrExpression = function(ctx) {  
        return this.visitChildren(ctx);  
    };  
    // Visit a parse tree produced by ECMAScriptParser#LogicalAndExpression.  
    visitLogicalAndExpression = function(ctx) {  
        return this.visitChildren(ctx);  
    };  
    // Visit a parse tree produced by ECMAScriptParser#NotExpression.  
    visitNotExpression = function(ctx) {  
        return this.visitChildren(ctx);  
    };  
}
```

...

Abstract away our problems (part I)

Visitor classes can only visit trees generated from a single grammar
→ need one Visitor per input language

- To avoid having the same code in every visitor, abstract the shared code into a super class!
- Each visitor will act as abstraction layer between superclass visitor and grammar-generated nodes.

codegeneration/python/visitor.js

```
const Python3Visitor = require(
  '../lib/antlr/Python3Visitor'
);

class Visitor extends Python3Visitor {
  visitAnd_expr(ctx) {
    return this.generateAndExpression(ctx);
  }
}
```

codegeneration/python/visitor.js

```
const Python3Visitor = require(
  '../lib/antlr/Python3Visitor'
);

class Visitor extends Python3Visitor {
  visitAnd_expr(ctx) {
    return this.generateAndExpression(ctx);
  }
}
```

codegeneration/javascript/visitor.js

```
const ECMAScriptVisitor = require(
  '../lib/antlr/ECMAScriptVisitor'
);

class Visitor extends ECMAScriptVisitor {
  visitLogicalAndExpression(ctx) {
    return this.generateAndExpression(ctx);
  }
}
```

codegeneration/python/visitor.js

```
const Python3Visitor = require(
  '../lib/antlr/Python3Visitor'
);

class Visitor extends Python3Visitor {
  visitAnd_expr(ctx) {
    return this.generateAndExpression(ctx);
  }
}
```

codegeneration/javascript/visitor.js

```
const ECMAScriptVisitor = require(
  '../lib/antlr/ECMAScriptVisitor'
);

class Visitor extends ECMAScriptVisitor {
  visitLogicalAndExpression(ctx) {
    return this.generateAndExpression(ctx);
  }
}
```

codegeneration/code-generation-visitor.js

```
export default (ANTLRVisitor) =>
  class CodeGenerationVisitor extends ANTLRVisitor {
    generateAndExpression(ctx) {
      const lhs = this.visit(ctx.getChildAt(0));
      const rhs = this.visit(ctx.getChildAt(2));

      return `${lhs} && ${rhs}`;
    }
  };
```


codegeneration/python/visitor.js

```
const Python3Visitor = require(
  '../lib/antlr/Python3Visitor'
);

class Visitor extends Python3Visitor {
  visitAnd_expr(ctx) {
    return this.generateAndExpression(ctx);
  }
}
```

codegeneration/javascript/visitor.js

```
const ECMAScriptVisitor = require(
  '../lib/antlr/ECMAScriptVisitor'
);

class Visitor extends ECMAScriptVisitor {
  visitLogicalAndExpression(ctx) {
    return this.generateAndExpression(ctx);
  }
}
```

codegeneration/code-generation-visitor.js

```
export default (ANTLRVisitor) =>
  class CodeGenerationVisitor extends ANTLRVisitor {
    generateAndExpression(ctx) {
      const lhs = this.visit(ctx.getChildAt(0));
      const rhs = this.visit(ctx.getChildAt(2));

      return `${lhs} && ${rhs}`;
    }
  };
};
```

codegeneration/python/visitor.js

```
const Python3Visitor = require(
  '../lib/antlr/Python3Visitor'
);

class Visitor extends Python3Visitor {
  visitAnd_expr(ctx) {
    return this.generateAndExpression(ctx);
  }
}
```

codegeneration/javascript/visitor.js

```
const ECMAScriptVisitor = require(
  '../lib/antlr/ECMAScriptVisitor'
);

class Visitor extends ECMAScriptVisitor {
  visitLogicalAndExpression(ctx) {
    return this.generateAndExpression(ctx);
  }
}
```

codegeneration/code-generation-visitor.js

```
export default (ANTLRVisitor) =>
  class CodeGenerationVisitor extends ANTLRVisitor {
    generateAndExpression(ctx) {
      const lhs = this.visit(ctx.getChildAt(0));
      const rhs = this.visit(ctx.getChildAt(2));

      return `${lhs} && ${rhs}`;
    }
  };
};
```

Abstract away our problems (part II)

We now have one visitor per input language

How do we avoid having to specialize each visitor for every combination of languages?

- Define "Generator" classes that generate code in methods called "emit"
- Treat Visitors as **abstract interfaces**.

codegeneration/python/generator.js

```
export default (Visitor) =>
  class Generator extends Visitor {
    emitAndExpression(lhs, rhs) {
      return `${lhs} and ${rhs}`;
    }
  };

```

codegeneration/javascript/generator.js

```
export default (Visitor) =>
  class Generator extends Visitor {
    emitAndExpression(lhs, rhs) {
      return `${lhs} && ${rhs}`;
    }
  };

```

codegeneration/python/generator.js

```
export default (Visitor) =>
  class Generator extends Visitor {
    emitAndExpression(lhs, rhs) {
      return `${lhs} and ${rhs}`;
    }
  };

```

codegeneration/javascript/generator.js

```
export default (Visitor) =>
  class Generator extends Visitor {
    emitAndExpression(lhs, rhs) {
      return `${lhs} && ${rhs}`;
    }
  };

```

codegeneration/code-generation-visitor.js

```
export default (ANTLRVisitor) =>
  class CodeGenerationVisitor extends ANTLRVisitor {
    generateAndExpression(ctx) {
      const lhs = this.visit(ctx.getChildAt(0));
      const rhs = this.visit(ctx.getChildAt(2));

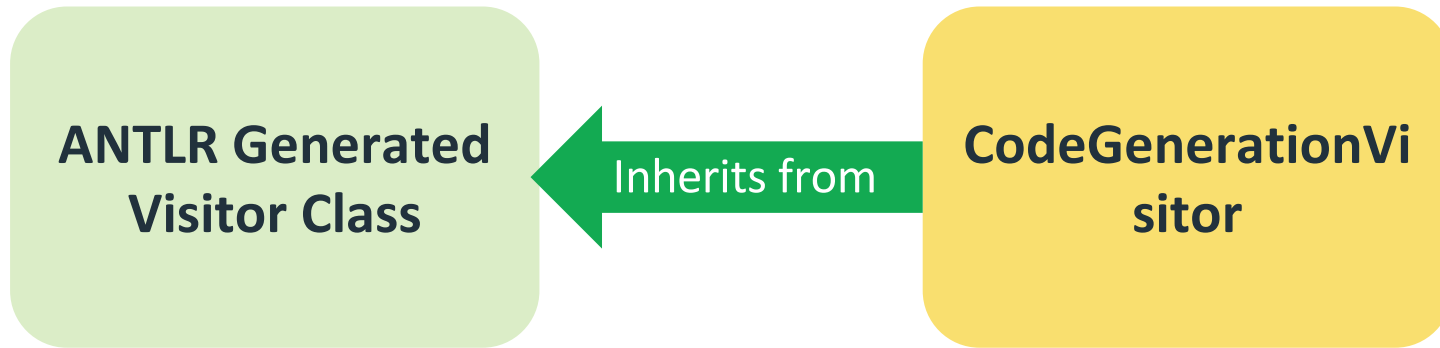
      return this.emitAndExpression(lhs, rhs);
    }
  };

```

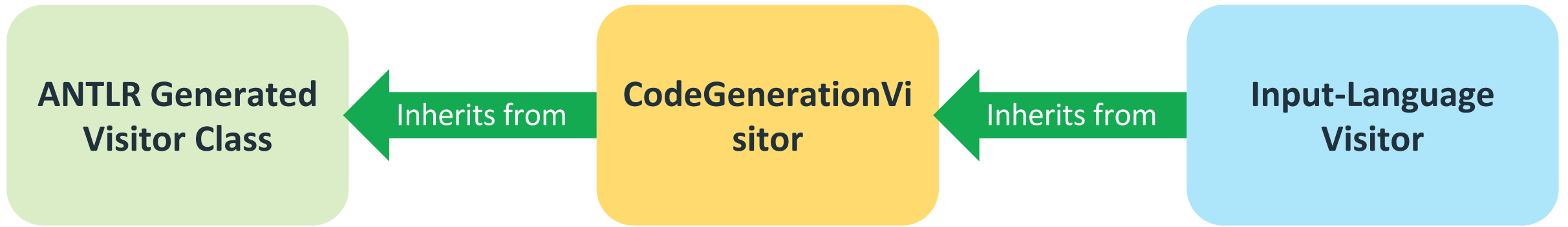
Composable Transpiler

**ANTLR Generated
Visitor Class**

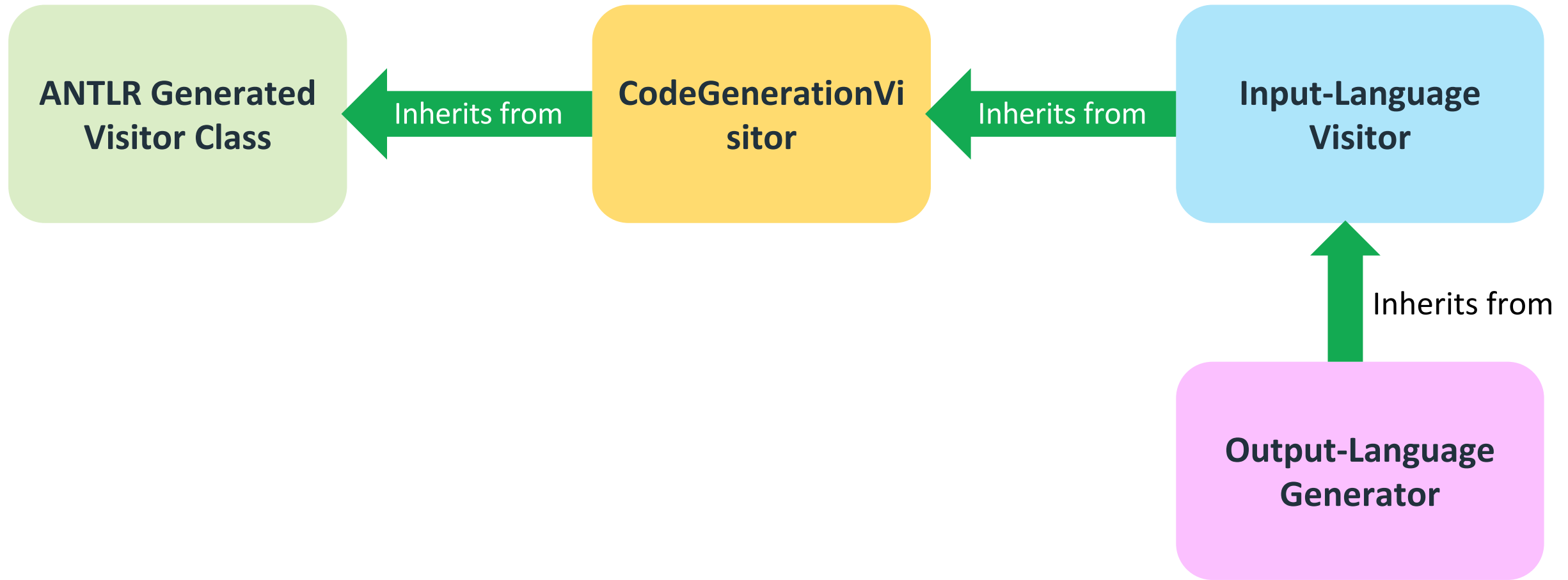
Composable Transpiler



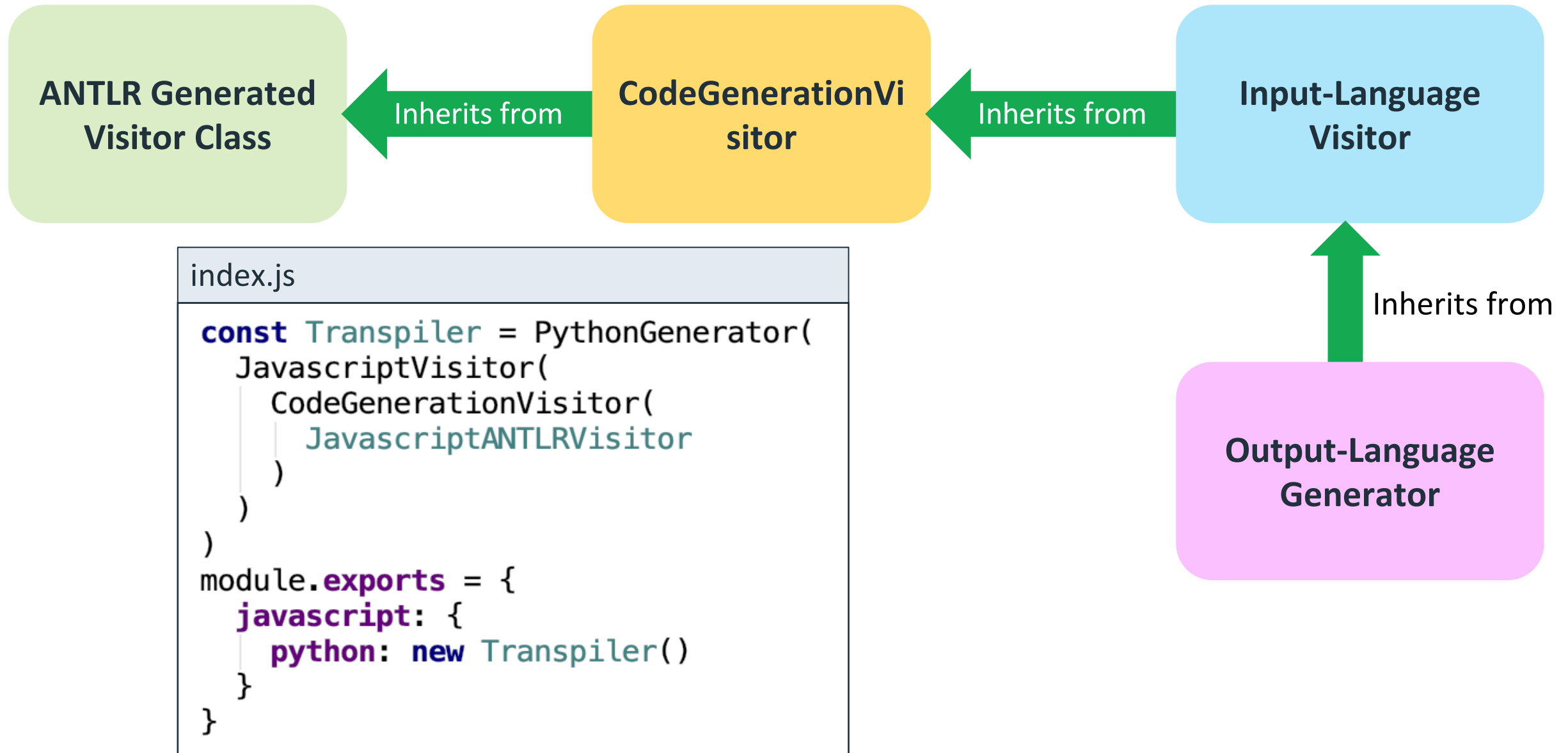
Composable Transpiler



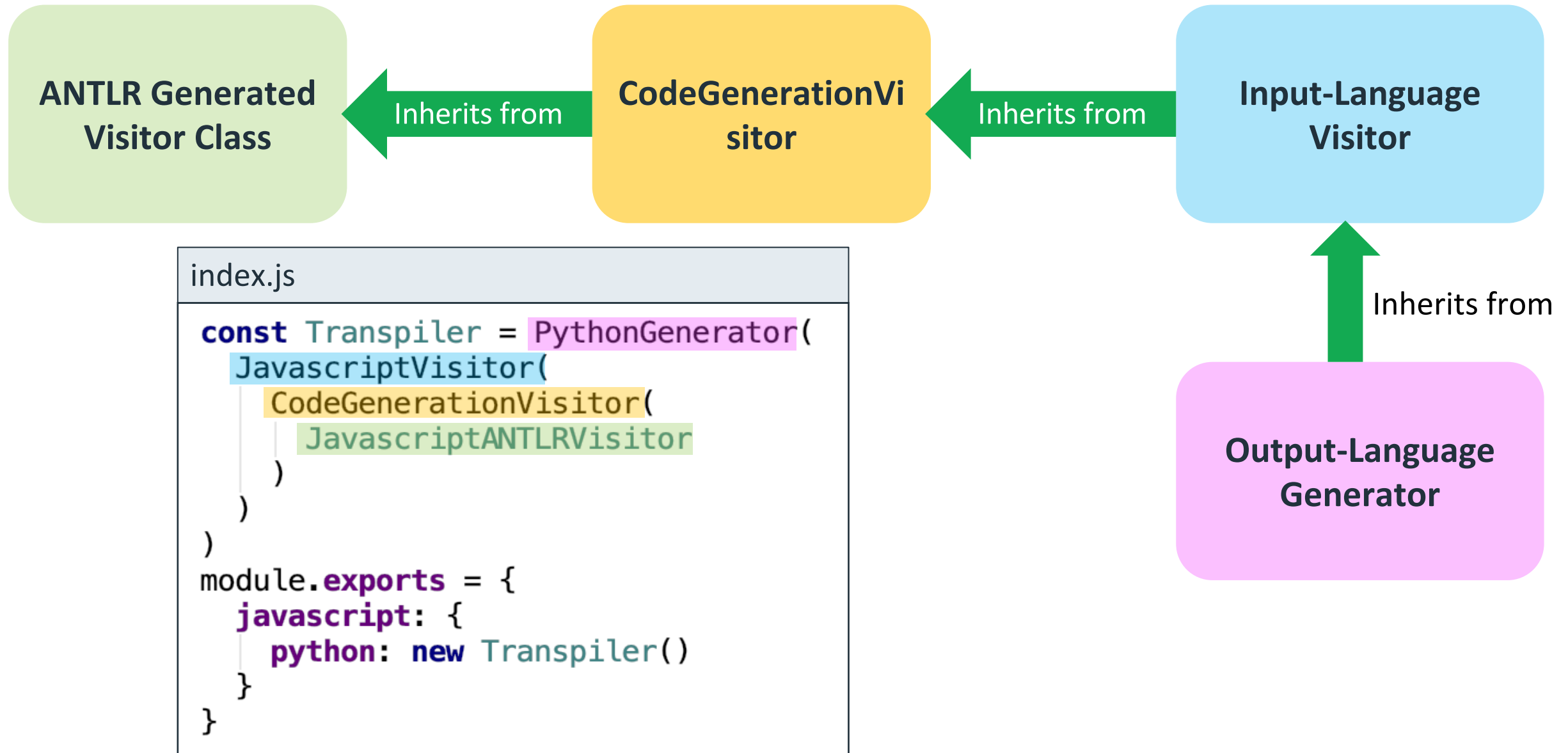
Composable Transpiler



Composable Transpiler



Composable Transpiler

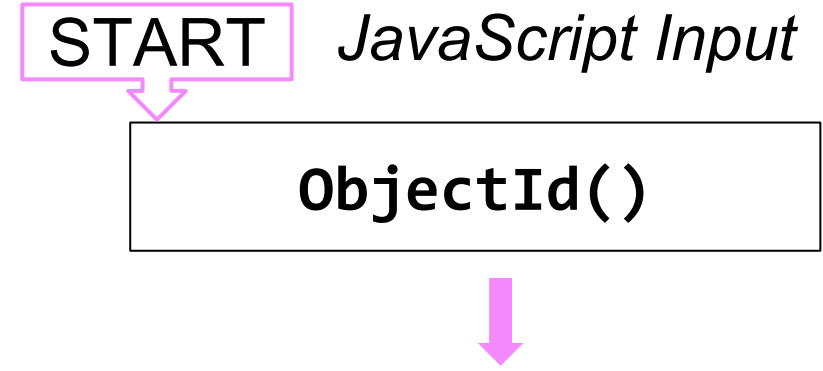
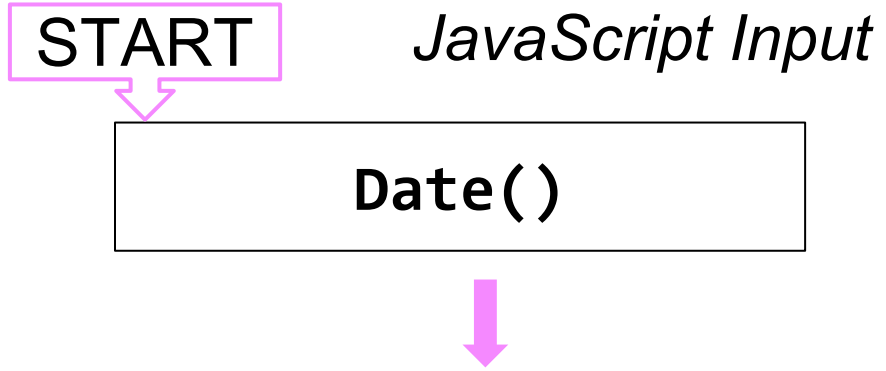


What about functions or variables?

We need to support native language features as well as BSON-specific types.

ObjectId, Date, Decimal128, Timestamp, etc...

How can we tell the difference between variables?



START

Date()



new java.util.Date()

END

JavaScript Input

Java Output

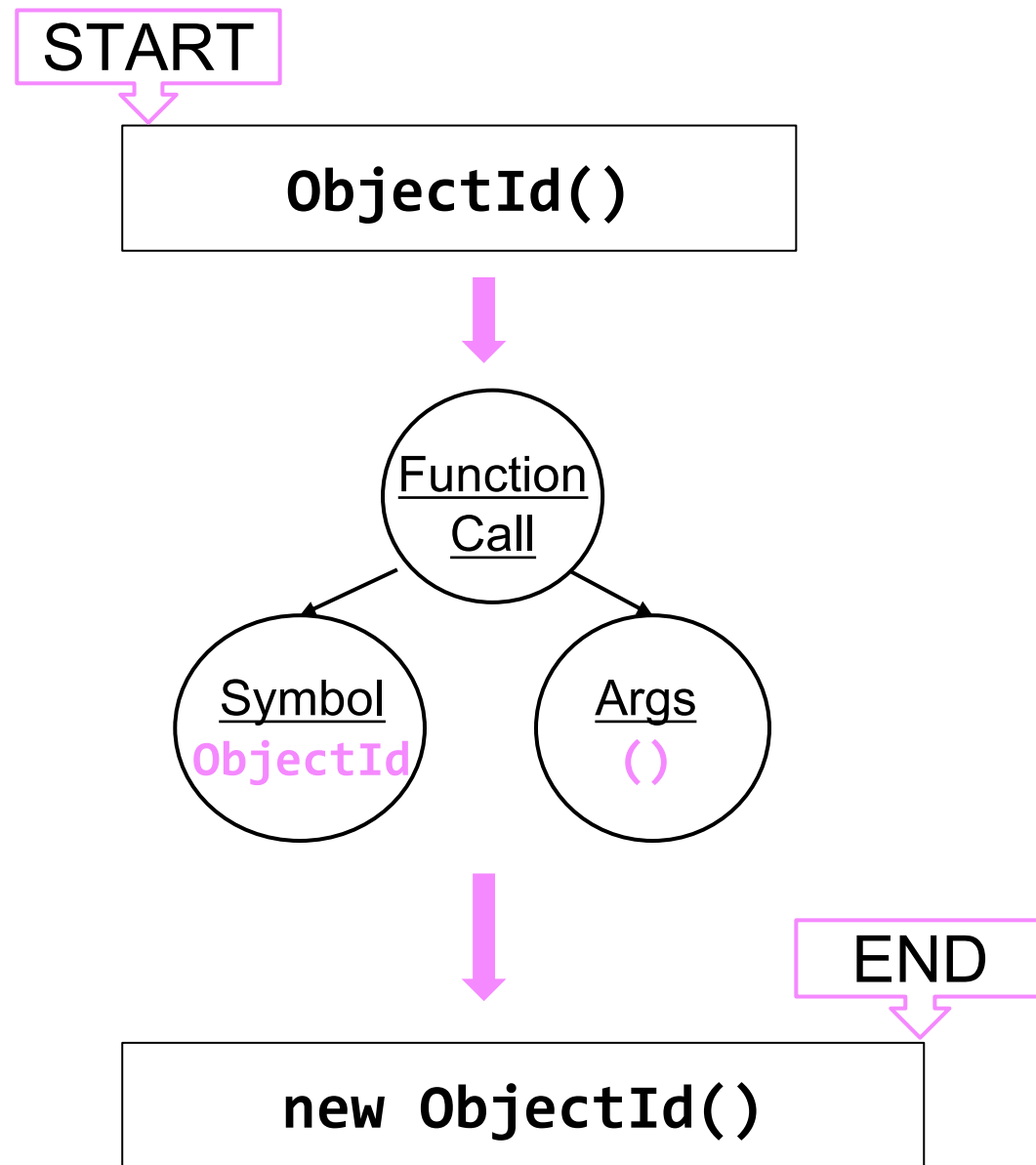
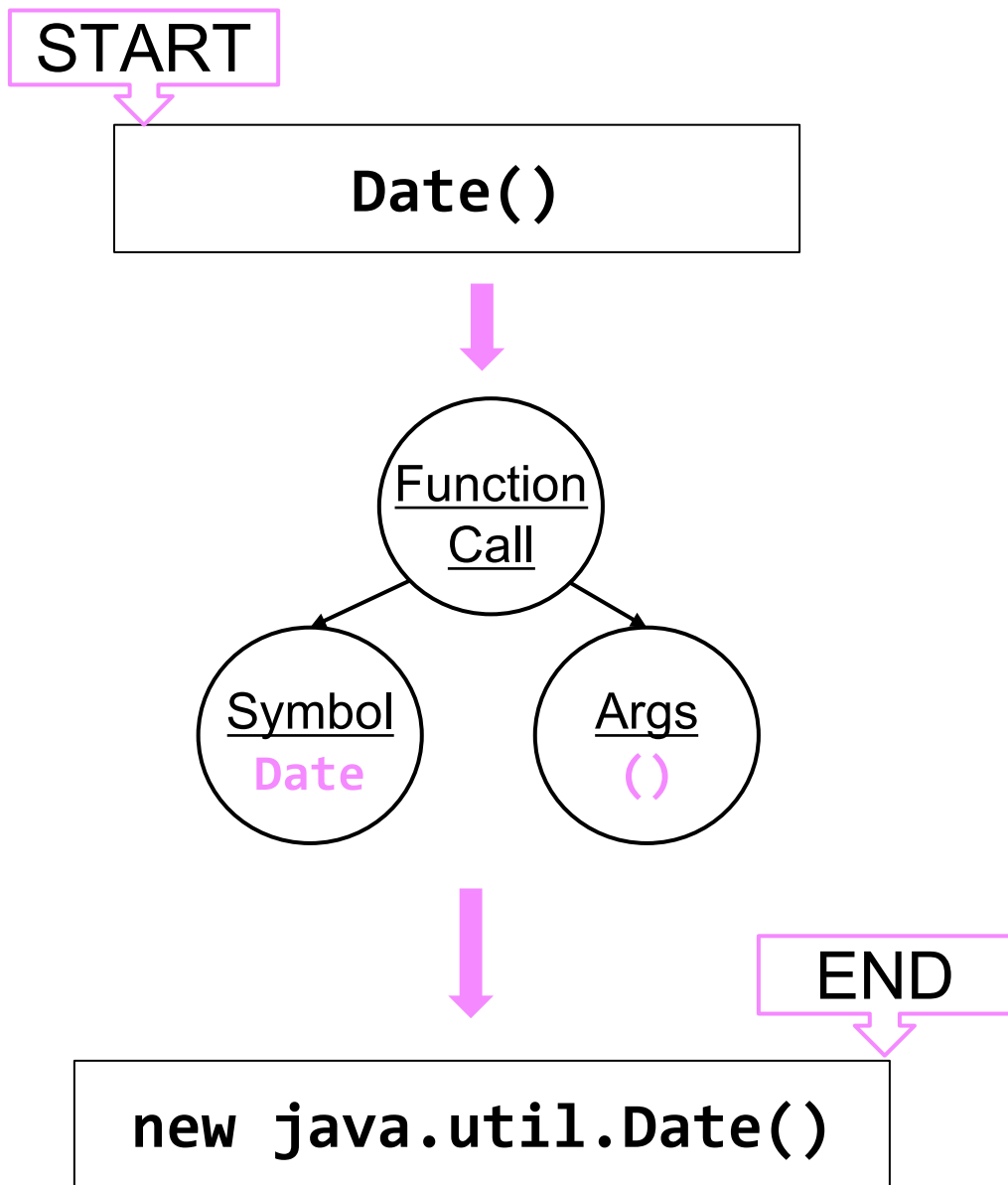
START

ObjectId()

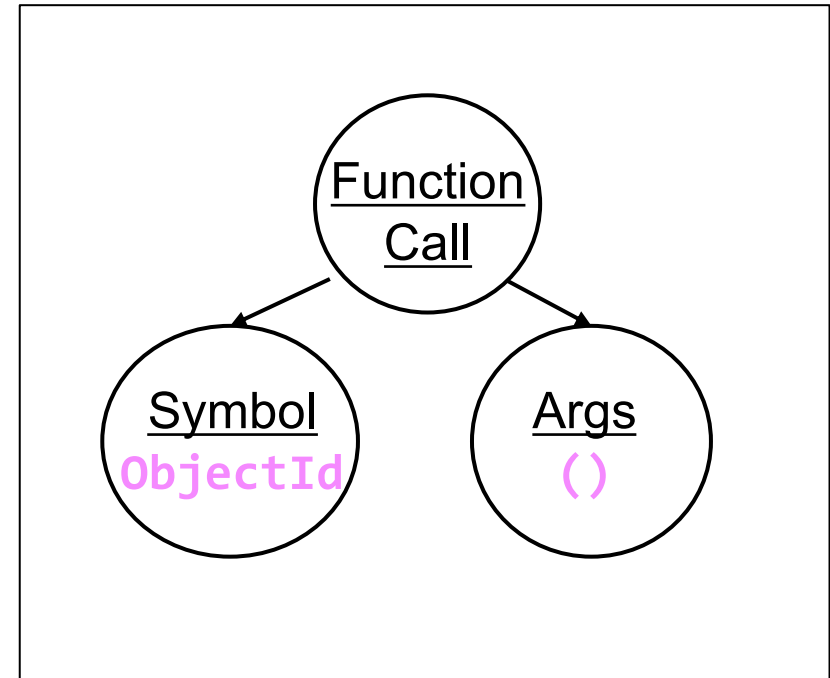
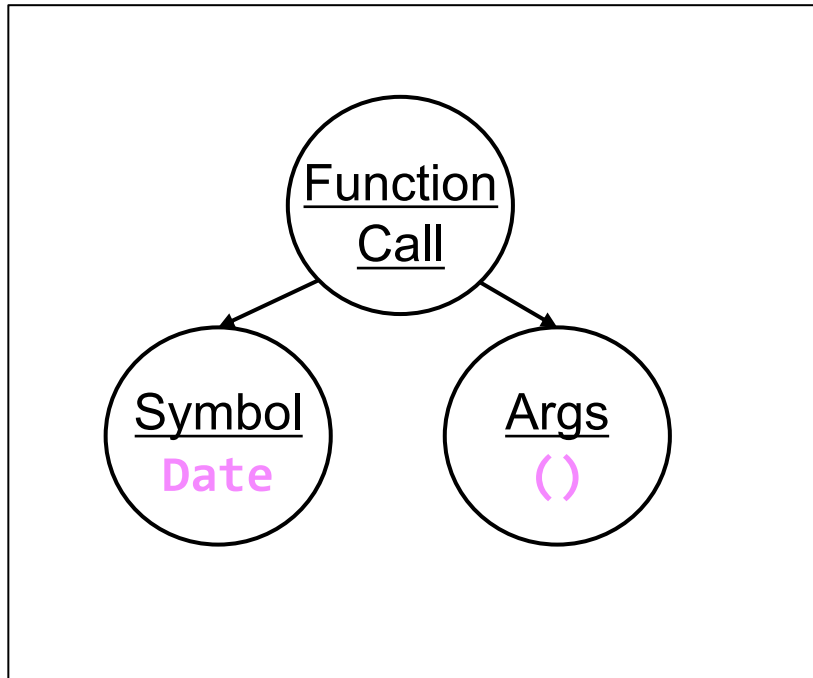


new ObjectId()

END



To the visitor, these two trees look the same...



The same `visit` methods are going to be called for both...

A decorative border composed of stylized purple and blue flowers and swirling vines, framing a central white area with a dashed blue border.

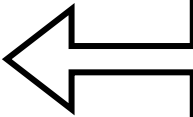
***We need a
Symbol Table!***

Symbol Table

- Need a place to keep track of all the symbols, i.e. variable or function names.
- When the visitor reaches a Symbol node, it looks it up in the **Symbol Table**
- This is also a convenient place to differentiate between output languages...

Visiting a Symbol

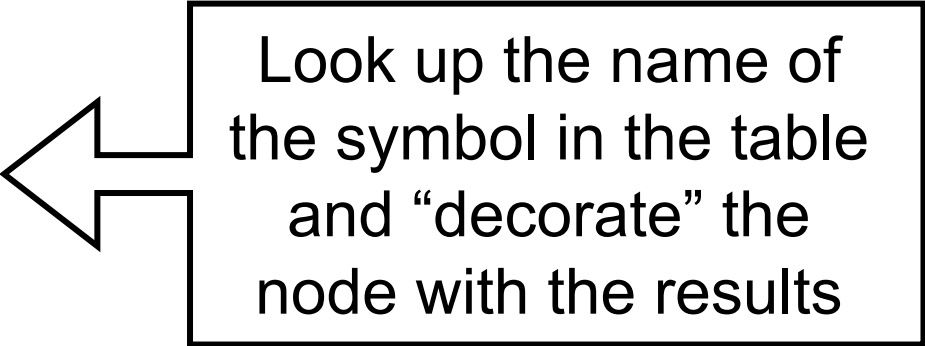
```
visitSymbol(node) {  
    const name = node.symbol;  
    node.type = this.Symbols[name];  
    if (node.type === undefined) {  
        throw new ReferenceError(`Symbol ${name} is undefined`);  
    }  
    return name;  
}
```



First get the symbol
itself from the node

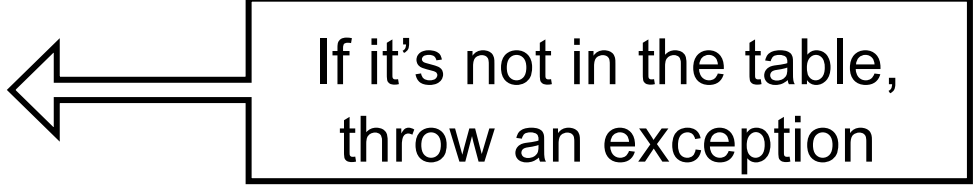
Visiting a Symbol

```
visitSymbol(node) {  
    const name = node.symbol;  
    node.type = this.Symbols[name];  
    if (node.type === undefined) {  
        throw new ReferenceError(`Symbol ${name} is undefined`);  
    }  
    return name;  
}
```



Visiting a Symbol

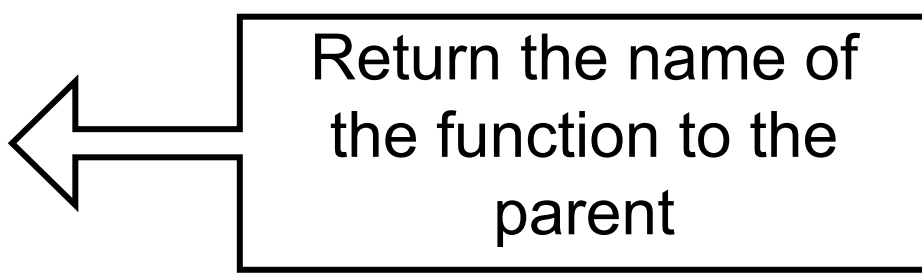
```
visitSymbol(node) {  
    const name = node.symbol;  
    node.type = this.Symbols[name];  
    if (node.type === undefined) {  
        throw new ReferenceError(`Symbol ${name} is undefined`);  
    }  
    return name;  
}
```



If it's not in the table,
throw an exception

Visiting a Symbol

```
visitSymbol(node) {  
    const name = node.symbol;  
    node.type = this.Symbols[name];  
    if (node.type === undefined) {  
        throw new ReferenceError(`Symbol ${name} is undefined`);  
    }  
    return name;  
}
```



Return the name of
the function to the
parent

What's in the Symbol Table?

ObjectId:

id: "ObjectId"

callable: *constructor

args:

- [*StringType, *NumericType, null]

type: *ObjectIdType

attr: {}

template: *ObjectIdSymbolTemplate

argsTemplate: *ObjectIdSymbolArgsTemplate

What's in the Symbol Table?

ObjectId:

id: "ObjectId"



The name of the attribute. Mostly used for error reporting.

What's in the Symbol Table?

ObjectId:

id: "ObjectId"

callable: *constructor

There are 3 types of symbol:

- *func: a function name

- *constructor: also a function name, but may require a "new"

- *var: a variable. Indicates that the symbol cannot be called.

What's in a Symbol?

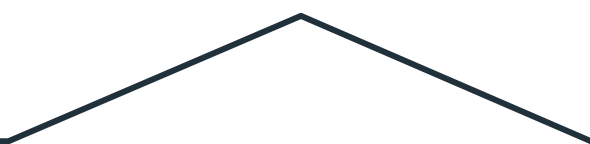
ObjectId:

id: "ObjectId"

callable: *constructor

args:

- [*StringType, *NumericType, null]



If the symbol is callable, this is where the arguments are defined. Each element in the array is a positional argument and contains the list of acceptable types. So ObjectId accepts one string or number argument, or no arguments at all.

What's in a Symbol?

ObjectId:

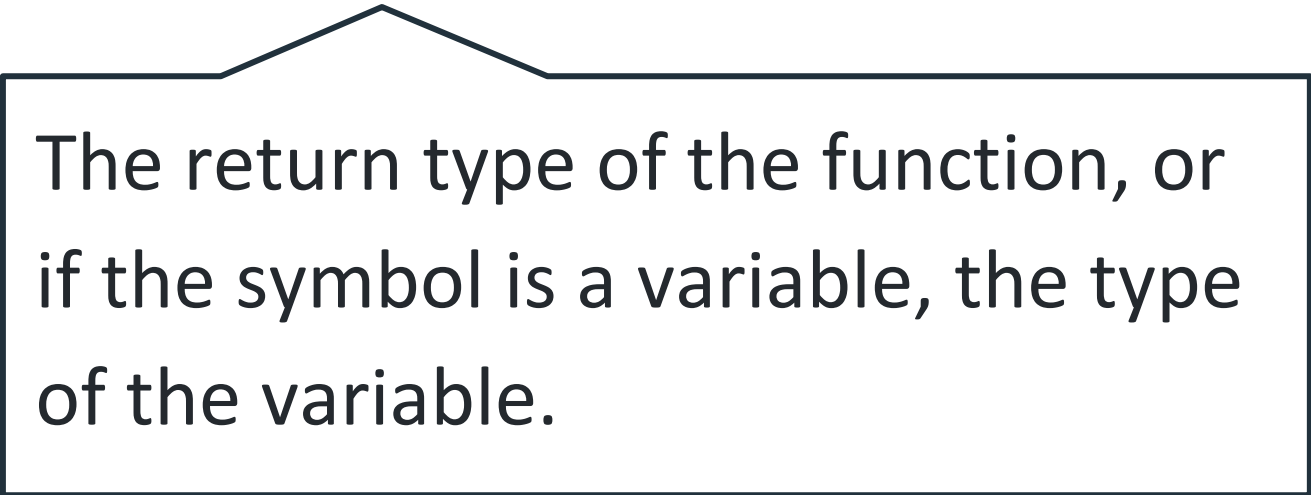
id: "ObjectId"

callable: *constructor

args:

- [*StringType, *NumericType, null]

type: *ObjectIdType



The return type of the function, or if the symbol is a variable, the type of the variable.

What's in a Symbol?

ObjectId:

id: "ObjectId"

callable: *constructor

args:

- [*StringType, *NumericType, null]

type: *ObjectIdType

attr: {...}

Any attributes of the symbol. This is a sub-symbol table, i.e. a mapping of names to symbols. Ex: `ObjectId.fromDate()`

What's in a Symbol?

ObjectId:

id: "ObjectId"

callable: *constructor

args:

- [*StringType, *NumericType, null]

type: *ObjectIdType

attr: {}

template: *ObjectIdSymbolTemplate



These are functions that accept strings and return strings.

Templates

Simple functions that accept strings and return strings

Responsible for doing the string transformations from one language syntax to another language's syntax

These are specific to the output language and **defined in a separate file that is loaded when the compiler is initialized.**

Each output language has a file
(in YAML)
where the templates are defined.

Symbol File (input language)

```
Double:  
  id: "Double"  
  code: 104  
  callable: *constructor  
  args:  
    - [ *NumericType, *StringType ]  
  type: *DoubleType  
  attr: {}  
  template: *DoubleSymbolTemplate
```

Template File (output language)

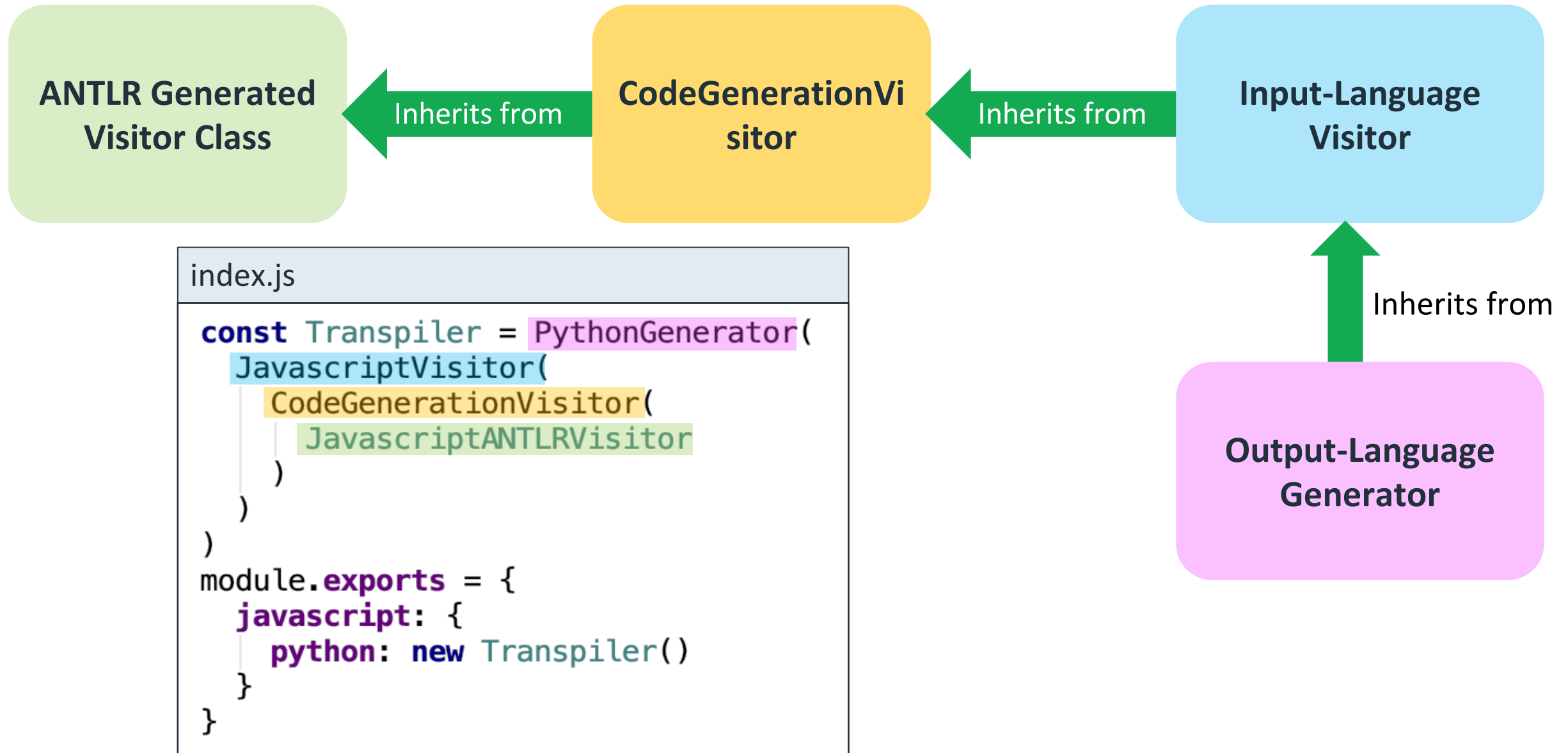
```
DoubleSymbolTemplate: &DoubleSymbolTemplate  
  () => {  
    return 'float';  
  }
```


Symbol File + Template File

=

Symbol Table

Composable Transpiler



Composable Transpiler

index.js

```
const Symbols = require('javascript/symbols.yaml');
const Templates = require('python/templates.yaml');

const SymbolTable = yaml.load(Symbols + Templates);

const Transpiler = PythonGenerator(
  JavascriptVisitor(
    CodeGenerationVisitor(
      JavascriptANTLRVisitor
    )
  )
)
module.exports = {
  javascript: {
    python: new Transpiler(SymbolTable)
  }
}
```

To Recap

ANTLR creates a tree from the user input

We visit the ANTLR-generated tree using visitors

When the visitor reaches a symbol, it looks up metadata in the Symbol Table.

- The metadata includes **template** functions that specify what code should be generated

So how do I add my own output language to Compass?

Add your own template file!!

All you need to do to add an output language is fill out the templates!

`symbols/sample_template.yaml`

There is a skeleton template file available

To add a new output language:

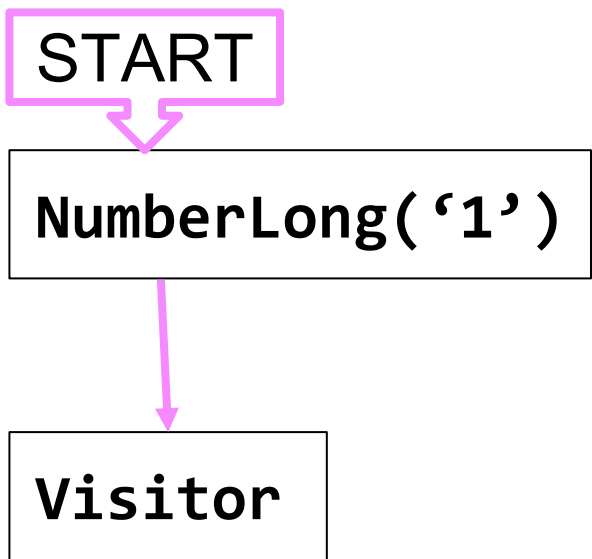
- fill out each template with the correct translation to your language.

Templates mostly apply to symbols, but there are also templates for literals and other syntax.

START

NumberLong('1')

Visitor



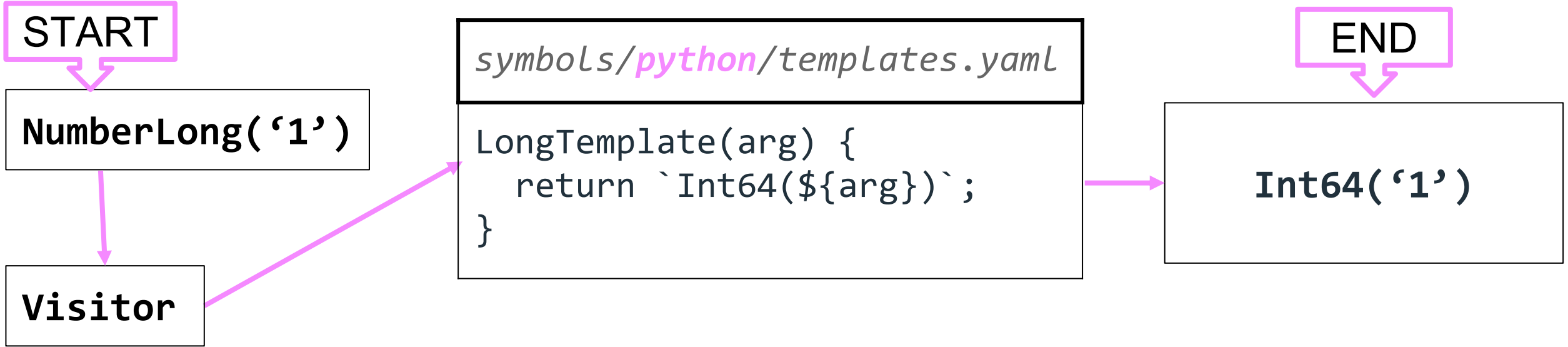
START

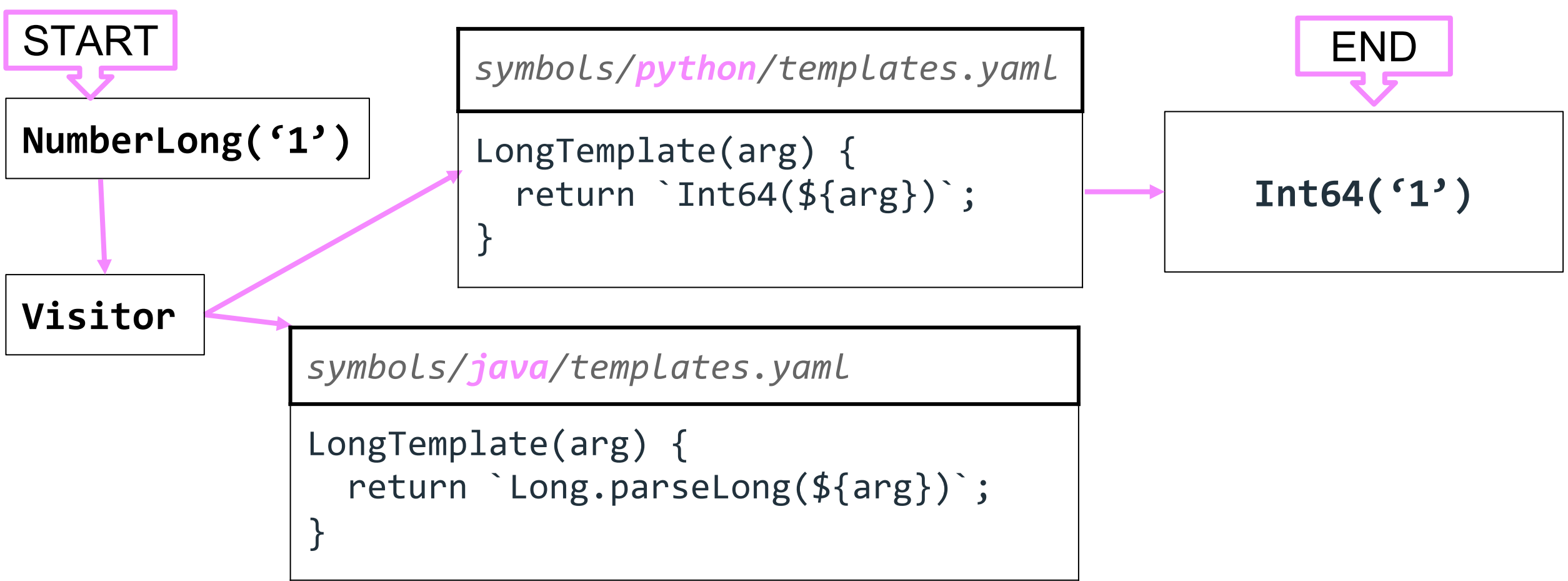
NumberLong('1')

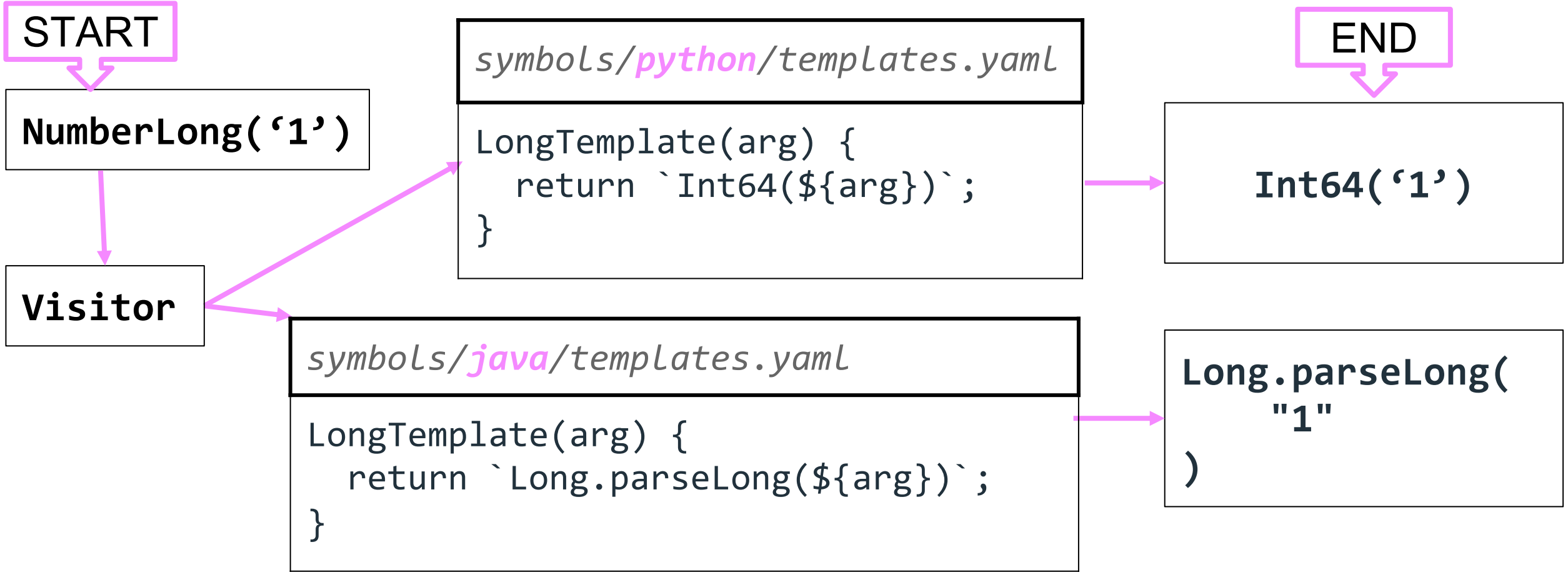
Visitor

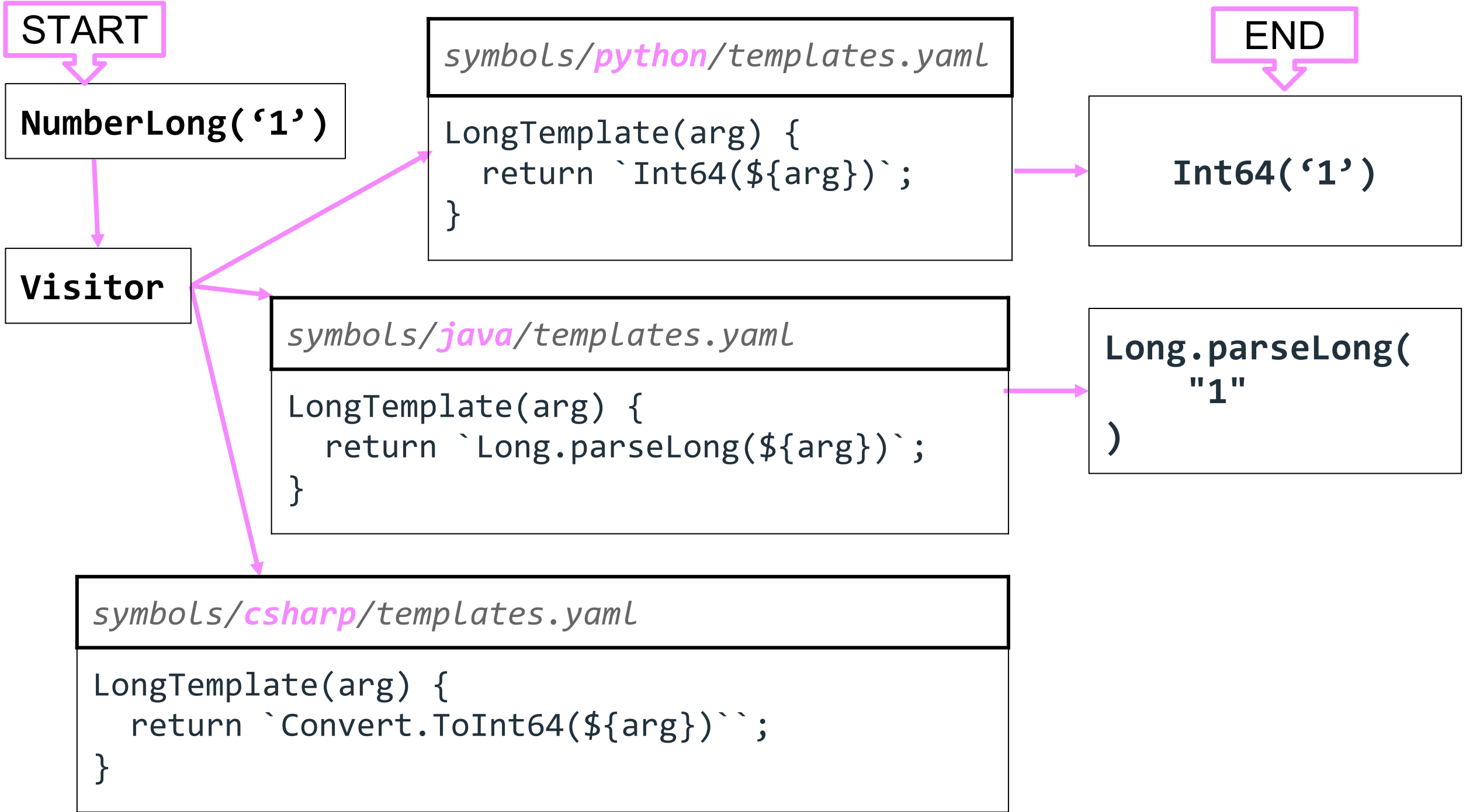
symbols/python/templates.yaml

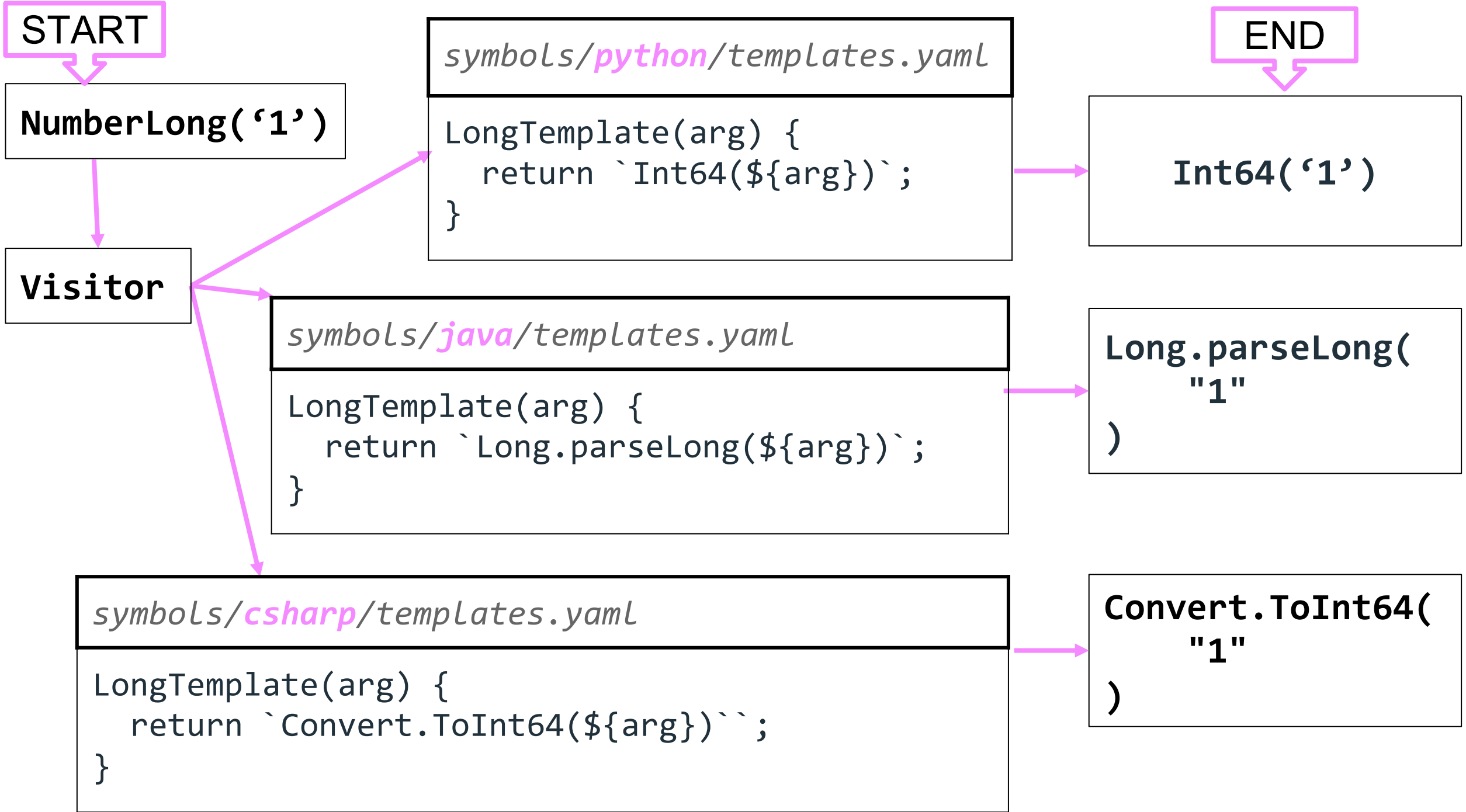
```
LongTemplate(arg) {  
  return `Int64(${arg})`;  
}
```









Expand Templates to Literals

Can apply the same method to literals

Example: Object Literals

- Python: `{ 'k' : 1 }`
- JS: `{k: 1}`
- C#: `new BsonDocument()`
- Java: `new Document()`

The slide features decorative floral elements in the corners. The top-left corner has a large green swirl and a small yellow flower. The top-right corner has a yellow flower and a green leaf. The bottom-left corner has a large yellow flower and a green leaf. The bottom-right corner has a green leaf and a small yellow flower.

Go forth and write templates!

Ruby, PHP, Go, R, Rust, C & more!

- ❖ Want to add an **output language**?
 - Just fill out a symbol table file!
- ❖ Want to write an **input language**?
 - Write a visitor

Expanded the syntax to include driver usage

Export Query To Language

My Pipeline:

```
1 {  
2   answer: {  
3     $type: 'Array'  
4   }  
5 }
```



Export Pipeline To: PYTHON 3

```
1 from pymongo import MongoClient  
2 client = MongoClient('mongodb://localhost:27017')  
3 query = {  
4   "$answer": {  
5     "$type": "Array"  
6   }  
7 }  
8  
9 result = client['myDB']['myColl'].find_one(query)  
10
```



☒ Include Import Statements

☒ Include Driver Code

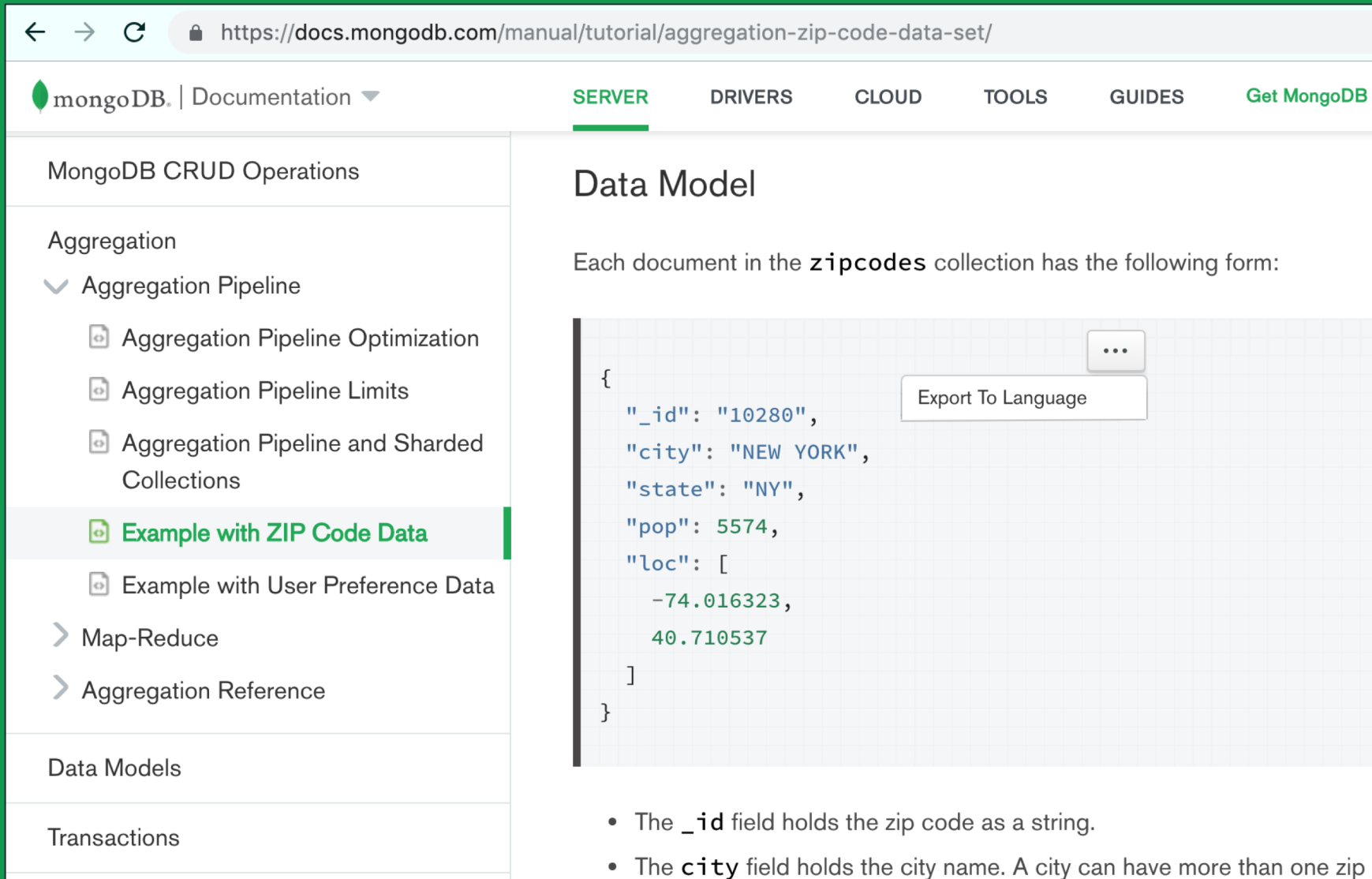
CLOSE

Future Features!

We now have a pluggable
transpiler from any language
BSON to any language
BSON....what can we do with it?



Generate examples for MongoDB University



The screenshot shows the MongoDB University documentation page for the 'zipcodes' collection. The left sidebar contains a navigation menu with categories like 'MongoDB CRUD Operations', 'Aggregation', 'Data Models', and 'Transactions'. The 'Aggregation' category is expanded, showing 'Aggregation Pipeline' and 'Example with ZIP Code Data' (which is highlighted). The main content area is titled 'Data Model' and explains that each document in the 'zipcodes' collection has a specific form. A code block displays a JSON document with fields for '_id', 'city', 'state', 'pop', and 'loc'. An 'Export To Language' button is visible next to the code. Below the code block, a list of bullet points explains the fields: '_id' holds the zip code as a string, and 'city' holds the city name.

← → ↻ 🔒 https://docs.mongodb.com/manual/tutorial/aggregation-zip-code-data-set/

mongoDB | Documentation ▾

SERVER DRIVERS CLOUD TOOLS GUIDES Get MongoDB

MongoDB CRUD Operations

Aggregation

✓ Aggregation Pipeline

- 📄 Aggregation Pipeline Optimization
- 📄 Aggregation Pipeline Limits
- 📄 Aggregation Pipeline and Sharded Collections
- 📄 **Example with ZIP Code Data**
- 📄 Example with User Preference Data

➤ Map-Reduce

➤ Aggregation Reference

Data Models

Transactions

Data Model

Each document in the **zipcodes** collection has the following form:

```
{
  "_id": "10280",
  "city": "NEW YORK",
  "state": "NY",
  "pop": 5574,
  "loc": [
    -74.016323,
    40.710537
  ]
}
```

Export To Language

- The **_id** field holds the zip code as a string.
- The **city** field holds the city name. A city can have more than one zip

```
annaherlihy@auckland:~ $ mongo
MongoDB shell version v3.6.3
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.6.3
Server has startup warnings:
2019-04-09T14:22:23.429+0200 I CONTROL [initandlisten]
2019-04-09T14:22:23.429+0200 I CONTROL [initandlisten]
2019-04-09T14:22:23.429+0200 I CONTROL [initandlisten]
2019-04-09T14:22:23.429+0200 I CONTROL [initandlisten]
2019-04-09T14:22:23.429+0200 I CONTROL [initandlisten]
2019-04-09T14:22:23.429+0200 I CONTROL [initandlisten]
2019-04-09T14:22:23.429+0200 I CONTROL [initandlisten]
2019-04-09T14:22:23.429+0200 I CONTROL [initandlisten]
2019-04-09T14:22:23.429+0200 I CONTROL [initandlisten]
2019-04-09T14:22:23.429+0200 I CONTROL [initandlisten]
2019-04-09T14:22:23.429+0200 I CONTROL [initandlisten]
MongoDB Enterprise > use language python
using language python
MongoDB Enterprise > db['myDB']['myColl'].find_one({
...     'answer': {
...         '$type': 'Array'
...     }
... })
```

Put it in front of the
shell!

Expand it to support 100% language
syntax!

Thanks to the Compass Team!

★ Alena Khineika

★ Irina Shestak

★ Durran Jordan





Thank you!

Everything I said, in much more detail:

github.com/mongodb-js/bson-transpilers

> CONTRIBUTING.md

Questions?

compass@mongodb.com or anna@mongodb.com

MONGODB

