cwft

A simple yet powerful translator that converts Swift into C++.

Our Goal

```
...
print("Hello, World!")
class A {
    var a: Int = 1
}
var b = "Aha!"
for i in 1...10 {
    b += "yo"
}
...
```

Convert a Swift file ...

Our Goal

... into an equivalent cpp file that can be compiled and run instantly

```
...
class A {
    int a = 1;
};
int main() {
    print("Hello, World!");
    string b = "Aha!";
    for (int i = 0; i <= 10; ++i) {
        b += "yo";
    }
...
```

Overview

```
inout
class
struct
enum
func
switch
while
repeat while
if else
typealias
for in
```

Overview

no auto

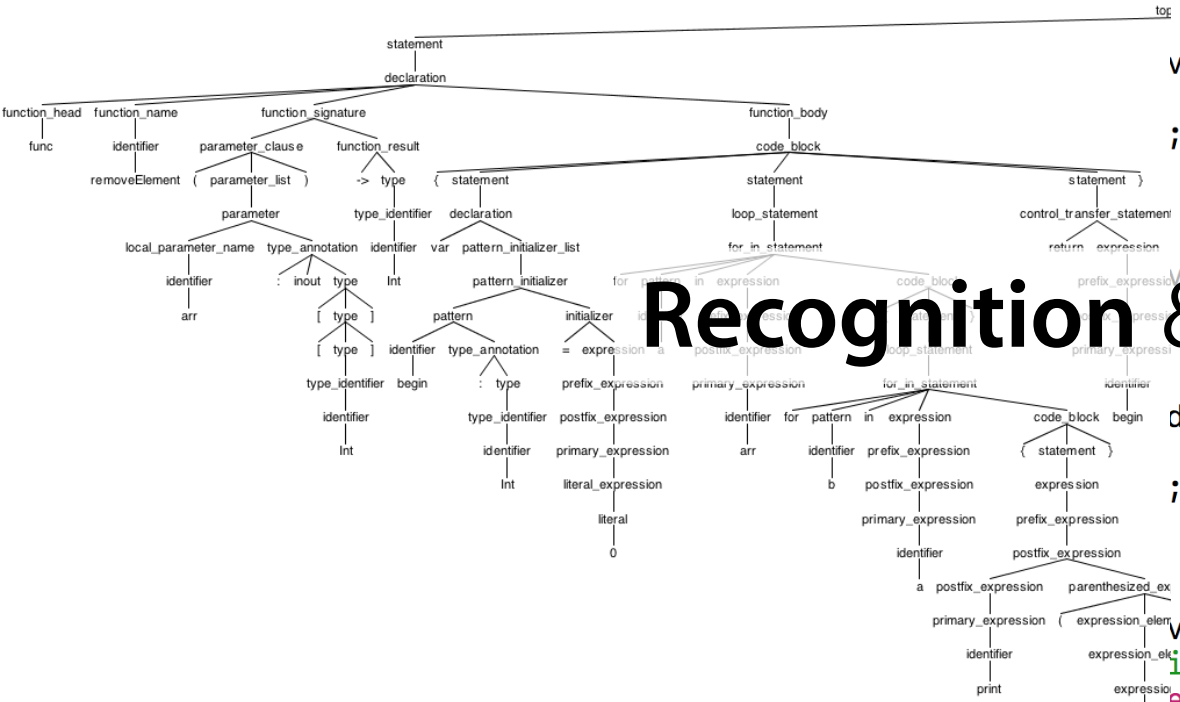
type inference

no c++11 for style

simple error checking

Tools

A two-step procedure



Recognition & Translation

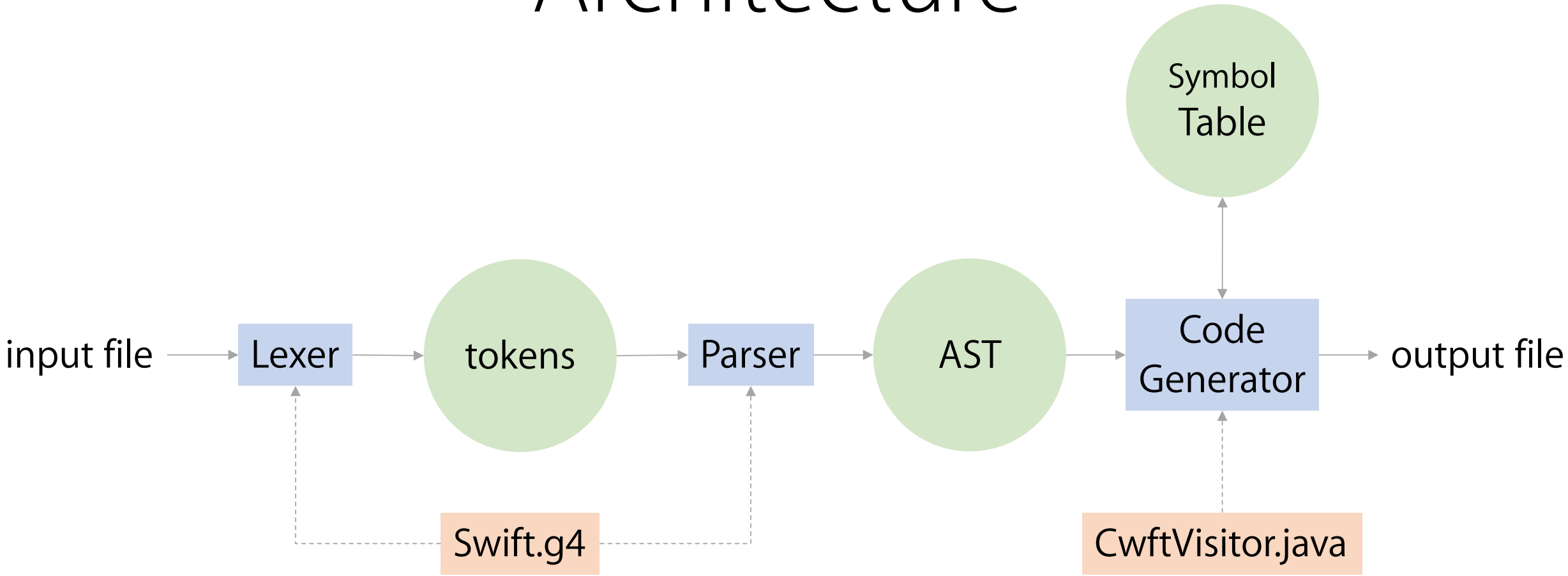
```
visitPrimary_expression(SwiftParser.Primary_expressionContext ctx) {
 = super.visitPrimary_expression(ctx);
;
```

```
visitLiteral_express(SwiftParser.Literal_expressContext ctx) {
 = visit(ctx.literal());
 = new Record(rec.type);
dd(rec.code.get(0));
 = rec.isType;
;
```

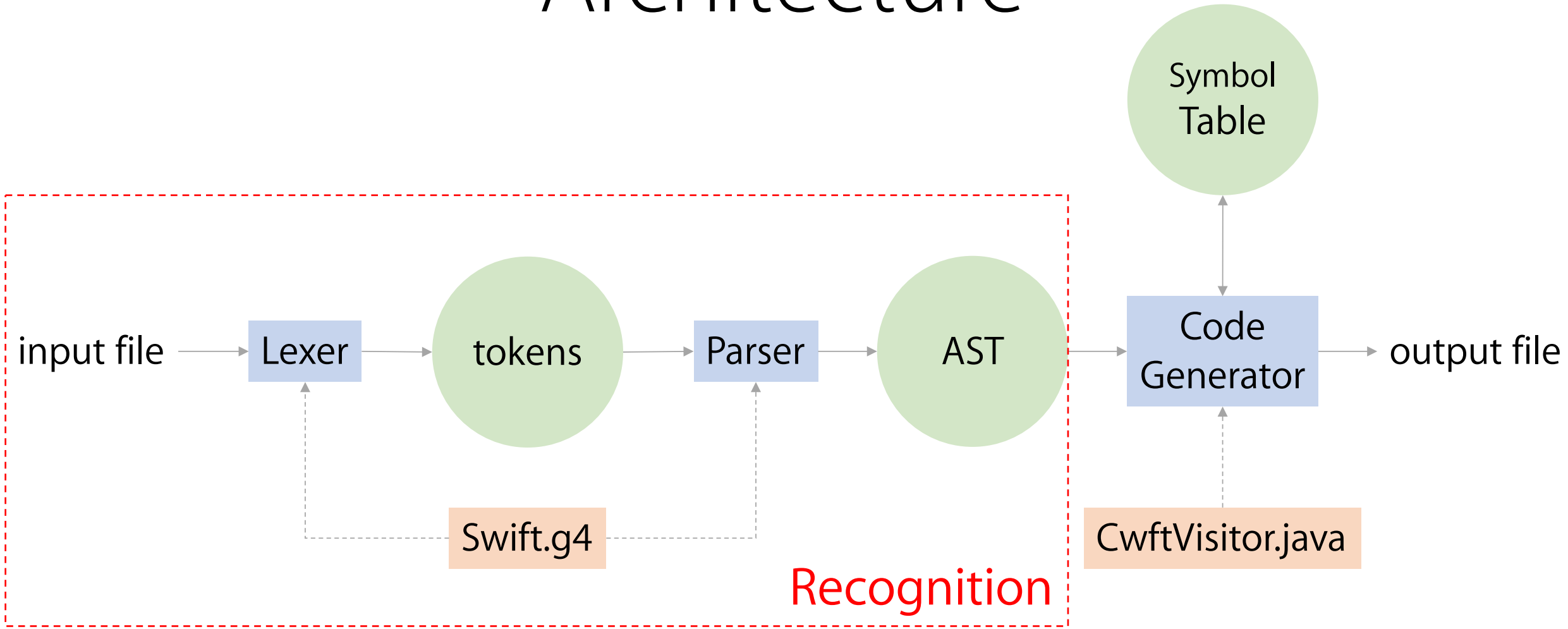
```
visitArray_literal_express(SwiftParser.Array_literal_expressContext ctx) {
iteral
er.visitArray_literal_express(ctx);
```

```
visitDict_literal_express(SwiftParser.Dict_literal_expressContext ctx) {
ary_literal
er.visitDict_literal_express(ctx);
```

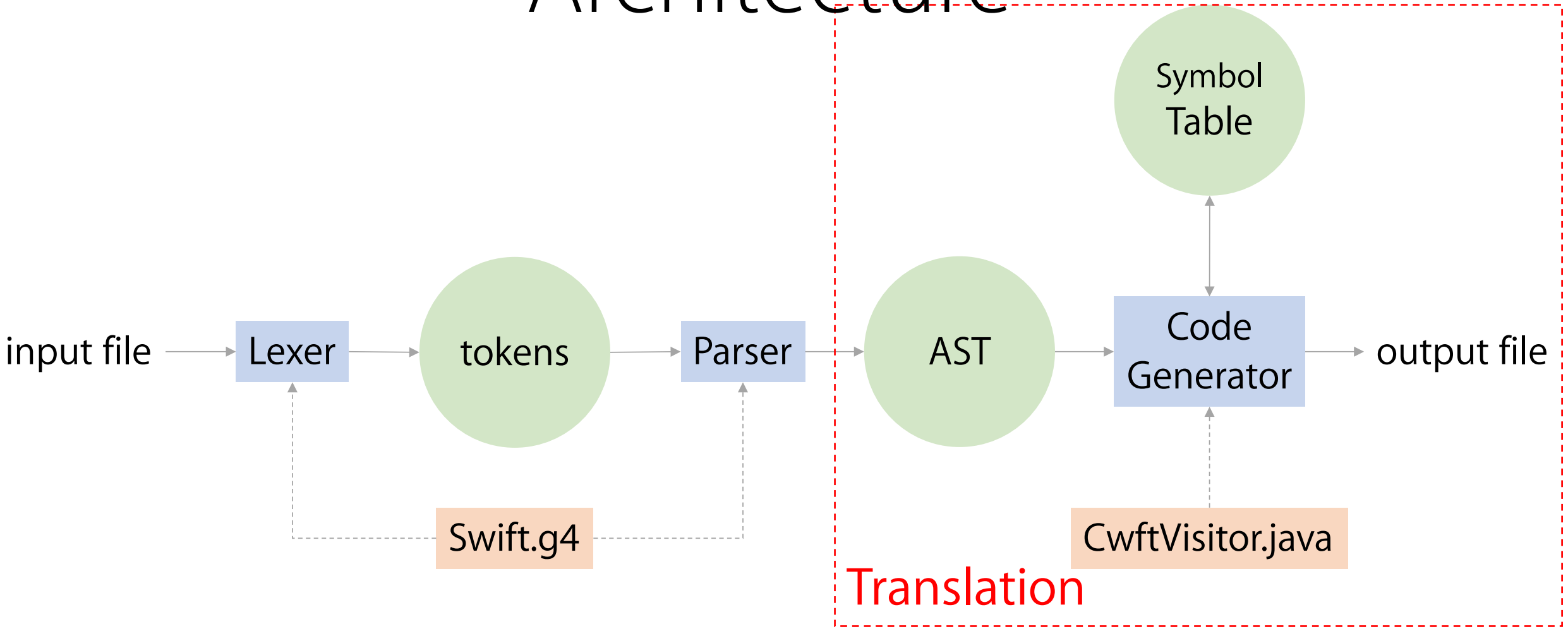
Architecture



Architecture



Architecture



Recognition

lexer parser

lexer

comment

key word

operator

separator

literal

identifier

white space

lexer

comment `// a comment`

key word

operator

separator

literal

identifier

white space

lexer

comment

key word `inout class enum`

operator

separator

literal

identifier

white space

lexer

comment

key word

operator `->` `+` `-` `...` `+=`

separator

literal

identifier

white space

lexer

comment

key word

operator

separator `, . : [] () {}`

literal

identifier

white space

lexer

comment

key word

operator

separator

literal `123 12.34 true "hello"`

identifier

white space

lexer

comment

key word

operator

separator

literal

identifier `abc` `a_b_c_`

white space

lexer

comment

key word

operator

separator

literal

identifier

white space `\r\n\t`

parser

statement

parser

statement

expression
declaration
loop statement
branch statement
control transfer statement

parser

statement

expression

parser

statement

expression

prefix-expression binary-expression$_{opt}$

parser

expression

binary-expression

binary-operator prefix-expression

= prefix-expression

? expression **:** prefix-expression

parser

statement

declaration

constant-declaration
variable-declaration
typealias-declaration
function-declaration
enum-declaration
struct-declaration
class-declaration

parser

statement

loop-statement

for-in-statement
while-statement
repeat-while-statement

parser

statement

branch-statement

if-statement

switch-statement

parser

statement

control-transfer-statement

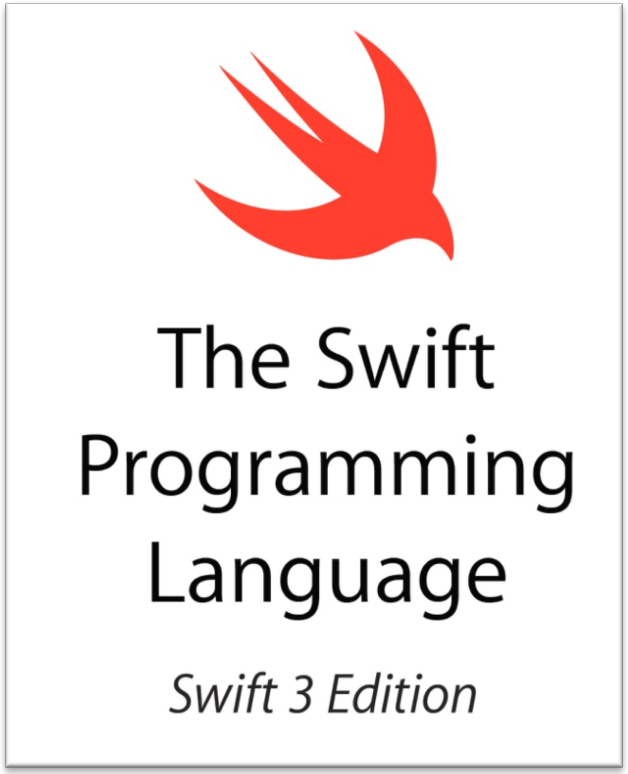
break

continue

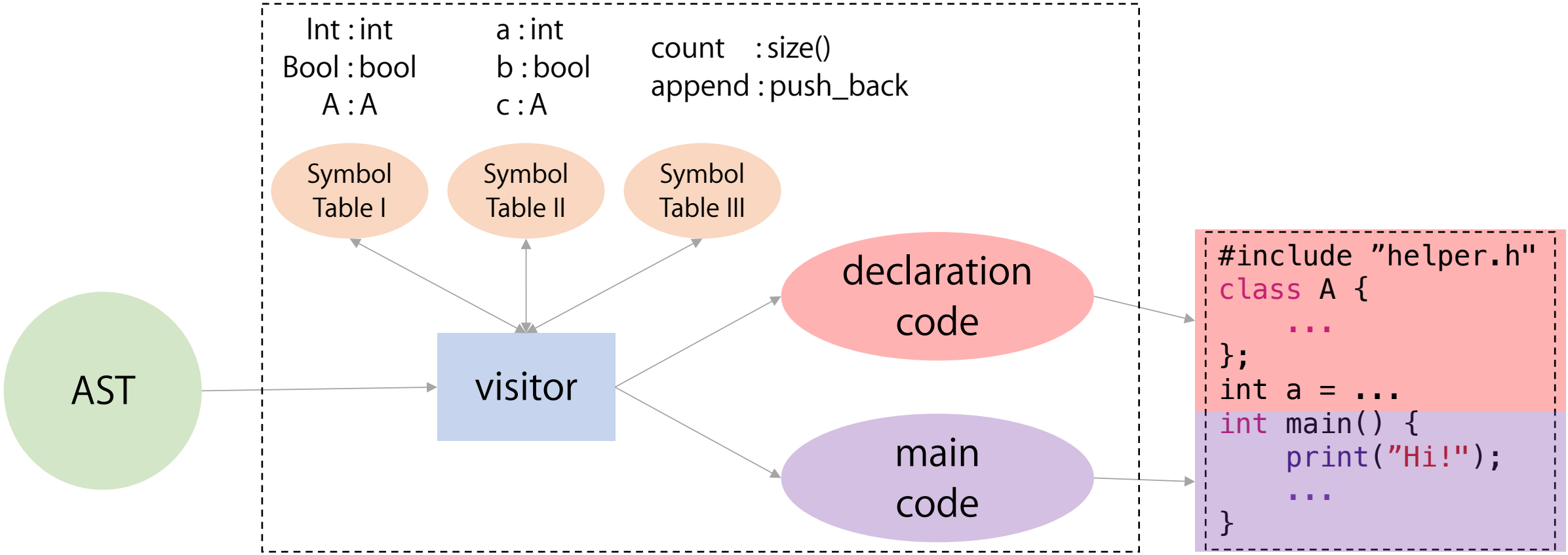
fallthrough

return expression

Reference



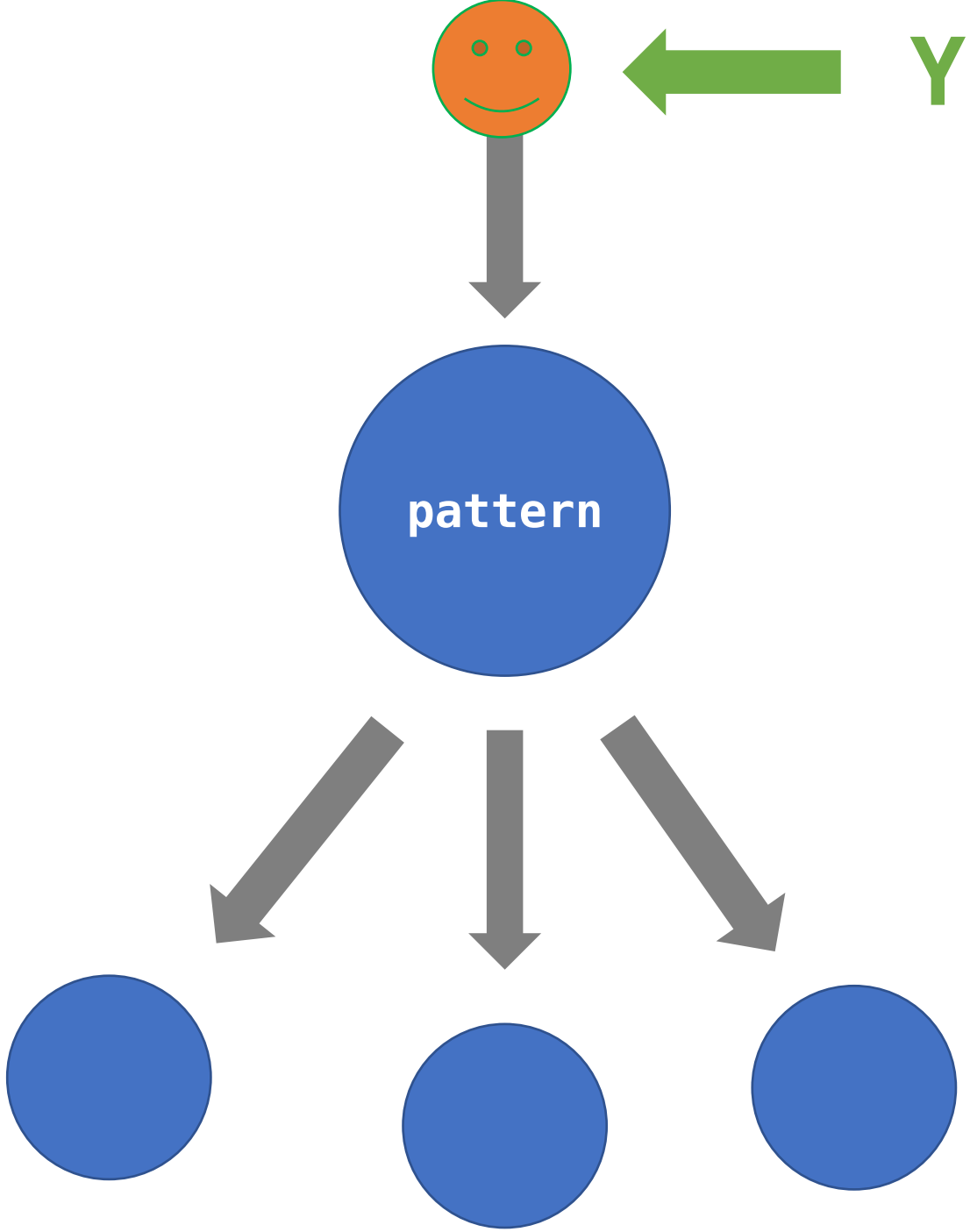
Translation

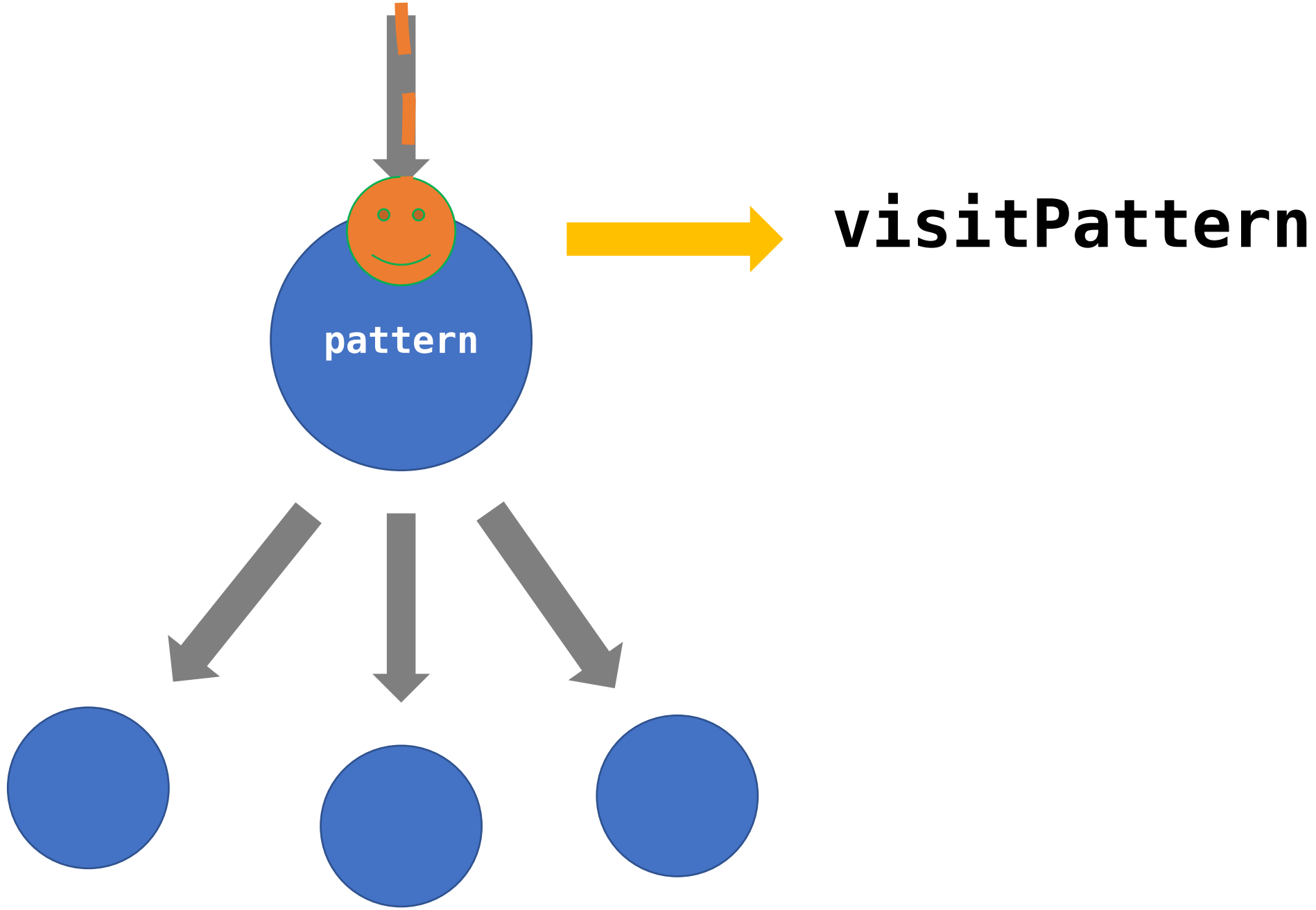


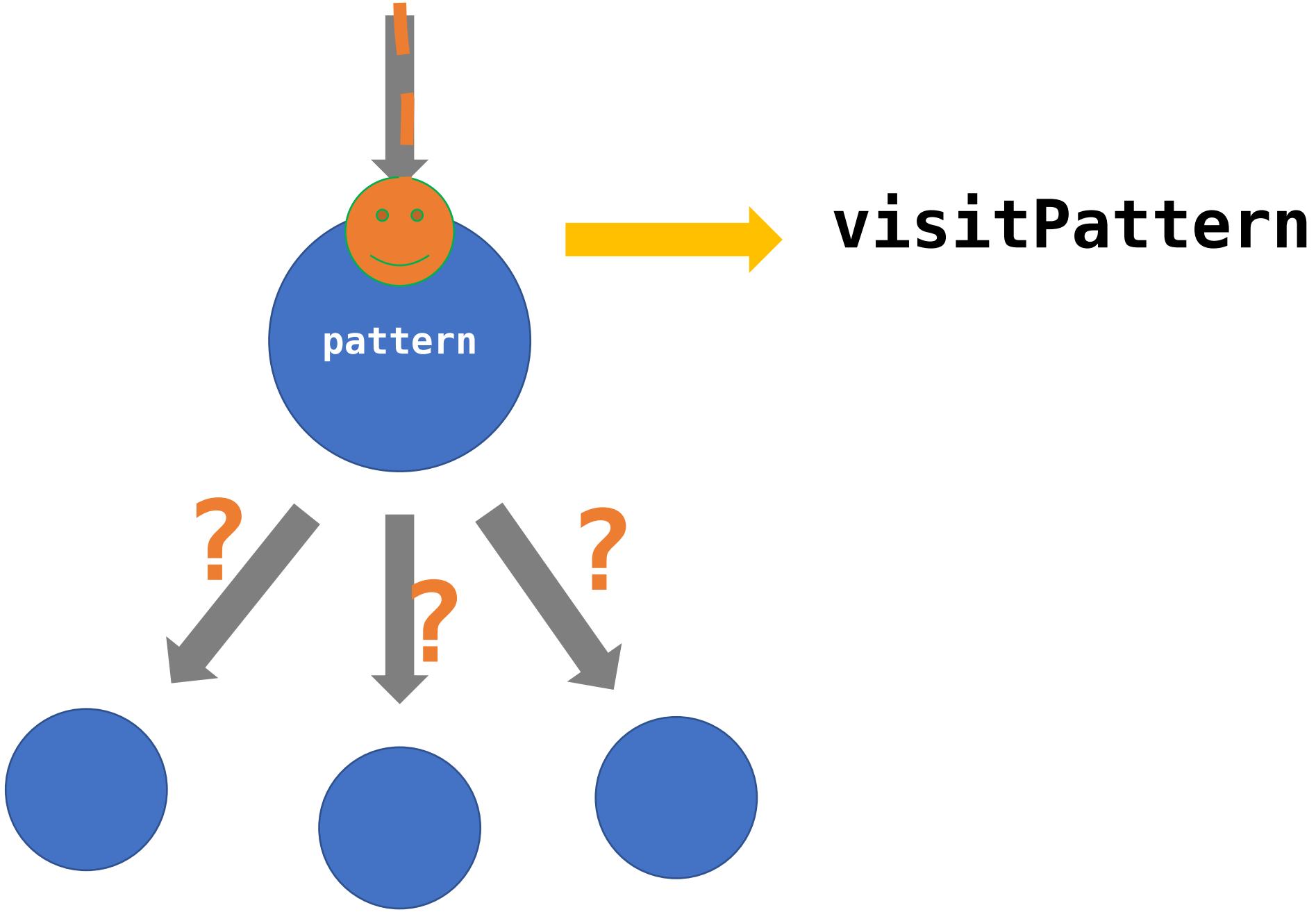
Visitor Review

visitor

you travel the tree **on your own, decide the order,** or even **stop**


YOU
pattern


pattern
visitPattern


pattern
visitPattern
?
?
?

```
visitPattern {

  // do sth
  // sth more
  // still not done

  return val;
}
```

Details

literal

```
123
12.3
true
"123"
```

```
123
12.3
true
"123"
```

typealias

```
typealias A = Int
```

```
typedef int A;
```

class member

```
array.count
array.append(1)
```

```
array.size();
array.push_back(1);
```

while

```
while true {

}
```

```
while (true) {

}
```

if

```
if a < b {

} else {

}
```

```
if (a < b) {

} else {

}
```

struct / class

```
struct A {
    var a: Int = 8
}
class B {
    var b: Int = 8
}
```

```
class A {
    int a = 8;
};
class B {
    int b = 8;
};
```

array / dict

```
[1,2,3,4,5]
["a":1,"b":2,"c":3]
```

```
{1,2,3,4,5}
{{"a",1},{"b",2},{"c",3}}
```

declaration

```swift
let a: Int = 4, b = 3.14,
c = [1,2,3]
```

```cpp
const int a = 4;
const double b = 3.14;
const vector<int> c = {1,2,3};
```

* need c++11

enum

```
enum C {
      case a, b, c
      case d, e, f
}
```

```
enum C {
      a,b,c,d,e,f
};
```

function

```
func f(a:Int) -> [Int] {
    ...
}
```

```
vector<int> f(int a) {
    ...
}
```

switch

```
switch a {
case 1:
    fallthrough
case 2:
    print("ha")
default:
    break
}
```

```
switch (a) {
case 1:
case 2:
    print("ha");
    break;
default:
    break;
}
```

for-in

```
for item in array {

}

for i in 1..<5 {

}
```

```
for (vector<int>::iterator it
= array.begin(); it !=
array.end(); ++it) {
    int item = *it;
}

for (int i = 0; i < 5; ++i) {

}
```

```
visitPattern {

  // do sth
  // sth more
  // still not done

  return val;
}
```

Return Type

```
class Record {
    List<String> code;
    Type type;
    boolean isType;
}
```

isType

```swift
var a = [6]
var b = [Int]()
```

```cpp
vector<int> a = {6};
vector<int> b = vector<int>();
```

Type Class

```
class Type {
    String basic;
    Type type1;
    Type type2;
    int wrap;
    public String toString();
}
```

Type Inference

Add

```
init
typealias
enum(case)
struct
class
declaration
function(param)
```

Check

```
identifier
declaration
param
enum(case)
```

Type Inference

```
var array = [1,2,3,4,5,6]
var item = array[1]
for i in array {

}
```

Process

```
// generate parser and lexer
$ antlr4 -no-listener -visitor Swift.g4
// hook them up
$ javac Master.java Swift*.java
// generate target code
$ java Master test.swift > res.cpp
// compile…
$ g++ -std=c++11 res.cpp
// …and run
$ ./a.out
```

Examples

```swift
var randomNumbers = [42, 12, 88, 62, 63, 56, 1, 77, 88,
97, 97, 20, 45, 91, 62, 2, 15, 31, 59, 5]
func quickSort(array: [Int]) -> [Int] {
    var less = [Int]()
    var equal = [Int]()
    var greater = [Int]()

    if array.count > 1 {
        let pivot = array[0]

        for x in array {
            if x < pivot {
                less.append(x)
            } else if x == pivot {
                equal.append(x)
            } else {
                greater.append(x)
            }
        }

        return (quickSort(array: less) + equal +
quickSort(array: greater))
    } else {
        return array
    }
}
print(quickSort(array: randomNumbers))
```

```cpp
#include "helper.cpp"
vector<int> randomNumbers = {42, 12, 88, 62, 63, 56, 1, 77, 88, 97, 97, 20, 45, 91, 62,
2, 15, 31, 59, 5};
vector<int> quickSort(vector<int> array)
{
    vector<int> less = vector<int>();
    vector<int> equal = vector<int>();
    vector<int> greater = vector<int>();
    if (array.size() > 1)
    {
        const int pivot = array[0];
        for (vector<int>::iterator it = array.begin(); it != array.end(); ++it)
        {
            int x = *it;
            if (x < pivot)
            {
                less.push_back(x);
            }
            else
                if (x == pivot)
                {
                    equal.push_back(x);
                }
                else
                {
                    greater.push_back(x);
                }
        }
        return (quickSort(less) + equal + quickSort(greater));
    }
    else
    {
        return array;
    }
}
int main() {
    print(quickSort(randomNumbers));
    return 0;
}
```

```
#include "helper.cpp"
vector<int> randomNumbers = {42, 12, 88, 62, 63, 56, 1, 77, 88, 97, 97, 20, 45, 91, 62,
2, 15, 31, 59, 5};
vector<int> quickSort(vector<int> array)
{
    vector<int> less = vector<int>();
    vector<int> equal = vector<int>();
    vector<int> greater = vector<int>();
    if (array.size() > 1)
    {
        const int pivot = array[0];
        for (vector<int>::iterator it = array.begin(); it != array.end(); ++it)
        {
            int x = *it;
            if (x < pivot)
            {
                less.push_back(x);
            }
            else if (x == pivot)
            {
                equal.push_back(x);
            }
            else
            {
                greater.push_back(x);
            }
        }
        return (quickSort(less) + equal + quickSort(greater));
    }
    else
    {
        return array;
    }
}
int main() {
    print(quickSort(randomNumbers));
    return 0;
}
```


```
const int pivot = array[0];
        for (vector<int>::iterator it = array.begin();
it != array.end(); ++it)
        {
            int x = *it;
```

```
class A {
    var a: Int = 5
    var b: Double = 6
    func f(d:Int)->Int {
        return a+b
    }
}
var c = A()
print(c.f(d:2))
```

```
#include "helper.cpp"
class A
{
public:
    int a = 5;
    double b = 6;
    int f(int d)
    {
        return a + b;
    }
};
A c = A();
int main() {
    print(c.f(2));
    return 0;
}
```

```swift
enum A {
    case a, b, c
}
var cc = A.a
switch cc {
case .a:
    print("haha")
case .b:
    print("hehe")
default:
    print("hoho")
}
```

```cpp
#include "helper.cpp"
enum A
{
    a,b,c,
};
A cc = a;
int main() {
    switch (cc)
    {
        case a:
            print("haha");
            break;
        case b:
            print("hehe");
            break;
        default:
            print("hoho");
    }
    return 0;
}
```

```swift
func f(a: inout Int) -> Int {
    a = 10
    return 8
}
var b = 5
print(f(a:&b)+b)
```

```cpp
#include
"helper.cpp"
int f(int & a)
{
    a = 10;
    return 8;
}
int b = 5;
int main() {
    print(f(b) + b);
    return 0;
}
```

```swift
func printElement(arr: inout [[Int]]) {
    for a in arr {
        for b in a {
            print(b)
        }
    }
}
var c = [[1,2,3,4],[5,6,7,8]]
printElement(arr: &c)
```

```cpp
#include "helper.cpp"
void printElement (vector<vector<int>> &
arr)
{
    int begin = 0;
    for (vector<vector<int>>::iterator it
= arr.begin(); it != arr.end(); ++it)
    {
        vector<int> a = *it;
        for (vector<int>::iterator it =
a.begin(); it != a.end(); ++it)
        {
            int b = *it;
            print(b);
        }
    }
}
vector<vector<int>> c = {{1, 2, 3, 4}, {5,
6, 7, 8}};
int main() {
    printElement(c);
    return 0;
}
```

```
var a = 10
// please don't ignore me!!
print(a)
```

You WISH!!

```
#include "helper.cpp"
int a = 10;
int main() {
    print(a);
    return 0;
}
```

Error Checking

```
var a = 1
var a = "a string"
```

Error: a already defined!

Future Work

~~works that are not done yet and will never be done~~

generics
optional
tuple
where-clause
…

Lessons Learned

start early!

start with simple cases

first make it run, then make it run fast (and beautiful)

languages are alike

don’t try to make it right at the 1st time

cwft

A simple yet powerful translator that converts Swift into C++.