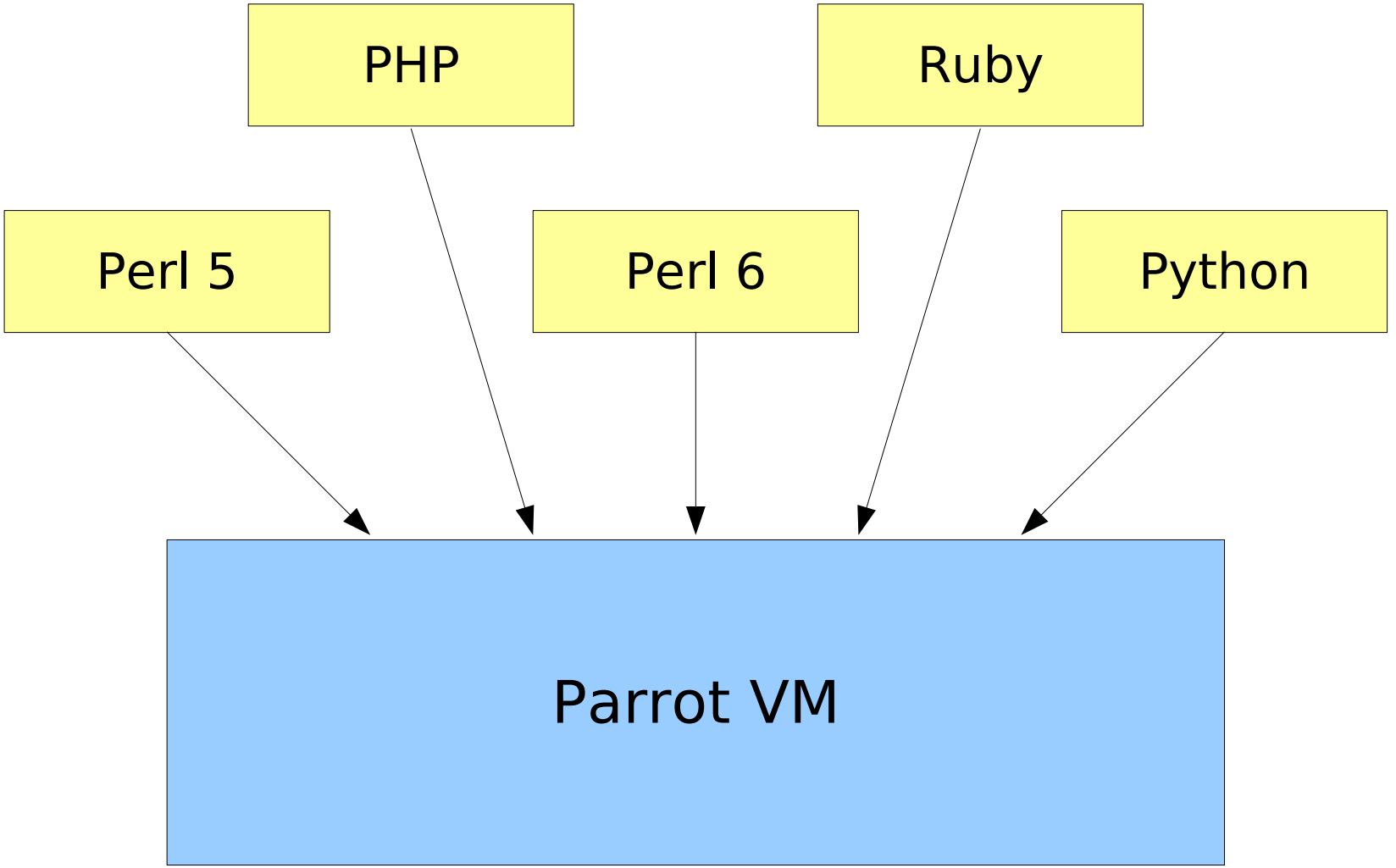
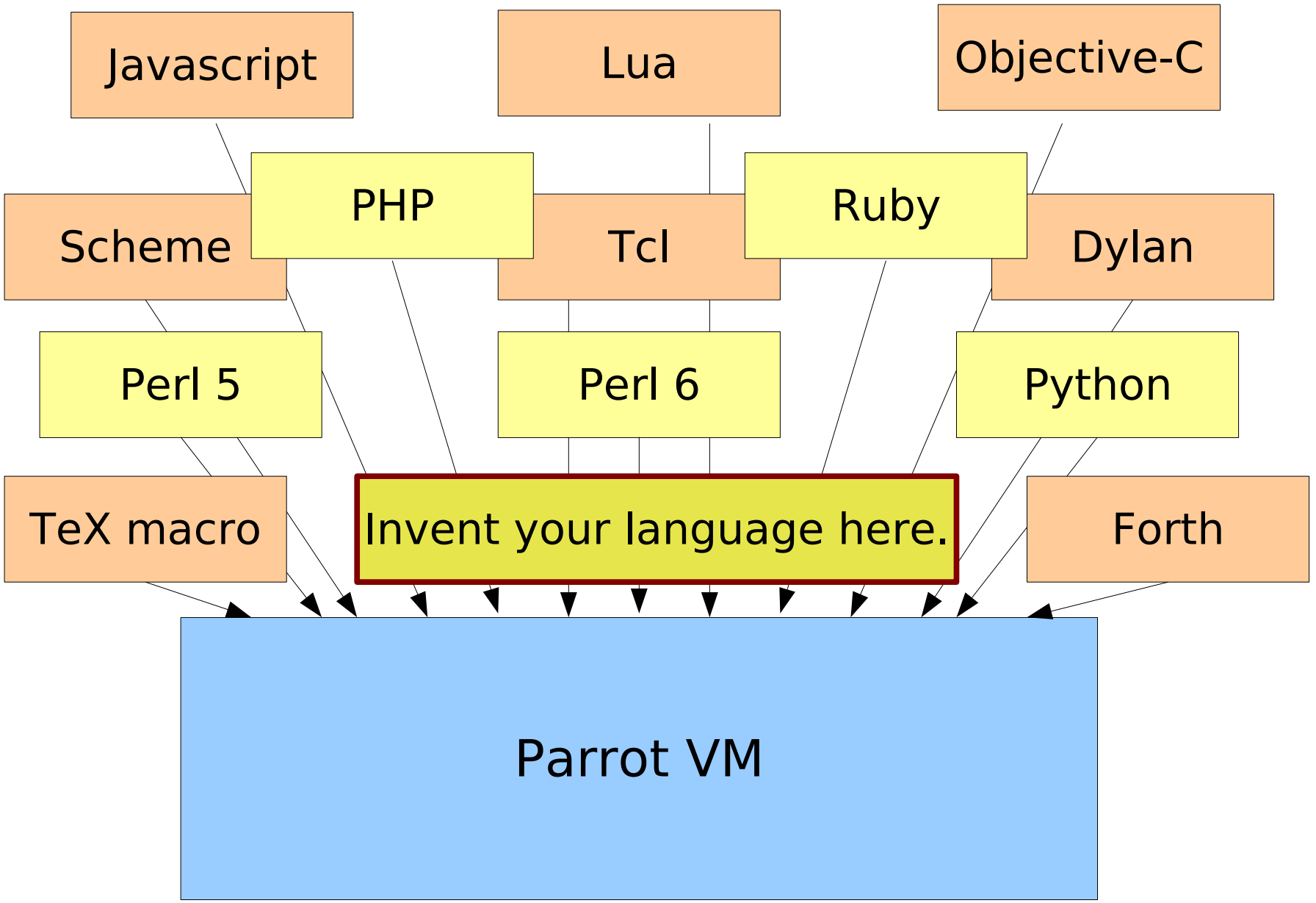
Parrot VM

Allison Randal
Parrot Foundation &
O'Reilly Media, Inc.

There's an odd misconception in the computing world that writing compilers is hard. This view is fueled by the fact that we don't write compilers very often. People used to think writing CGI code was hard. Well, it is hard, if you do it in C without any tools.


PHP
Ruby
Perl 5
Perl 6
Python
Parrot VM


Javascript
Lua
Objective-C
PHP
Ruby
Scheme
Tcl
Dylan
Perl 5
Perl 6
Python
TeX macro
Invent your language here.
Forth
Parrot VM

Dynamic Languages

Runtime vs. compile-time

Extend code (eval, load)

Define classes

Alter type system

Higher-order functions

Closures, Continuations, Coroutines

Why?

Revolution

Powerful tools

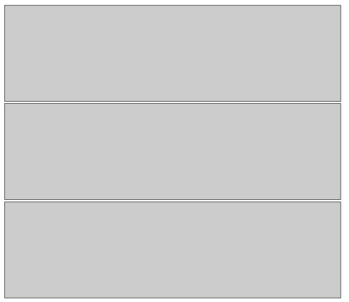
Portability

Interoperability

Innovation

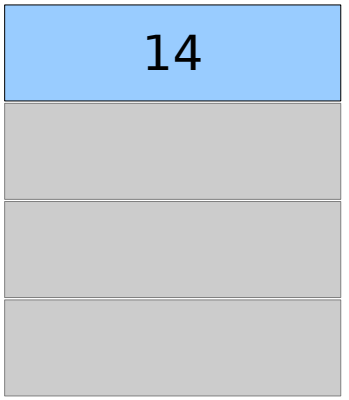
Register-based

Stack operations



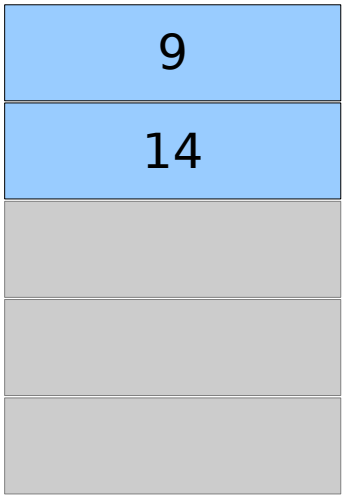
Register-based

Stack operations



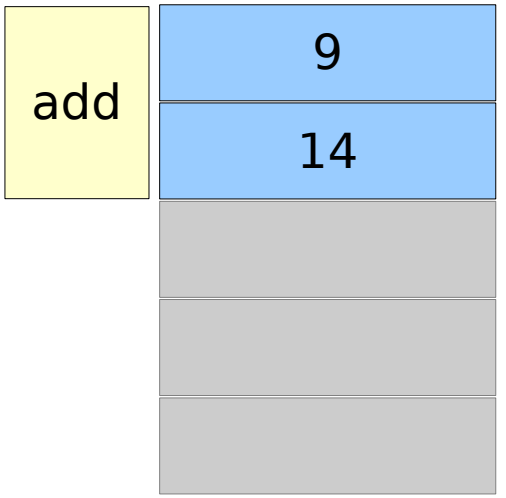
Register-based

Stack operations



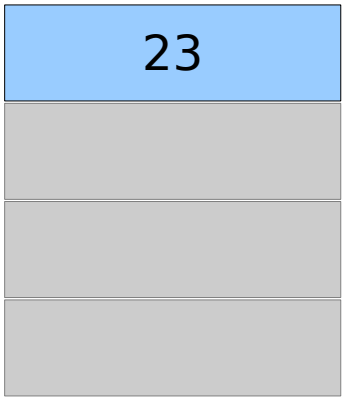
Register-based

Stack operations



Register-based

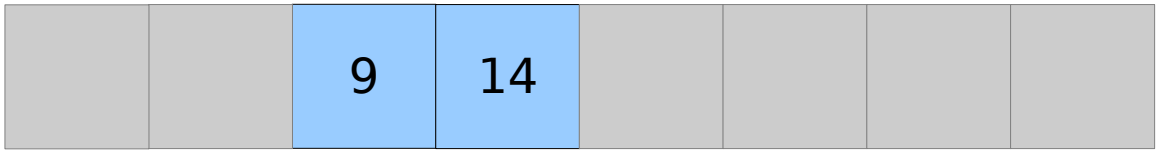
Stack operations



Register-based

Stack operations

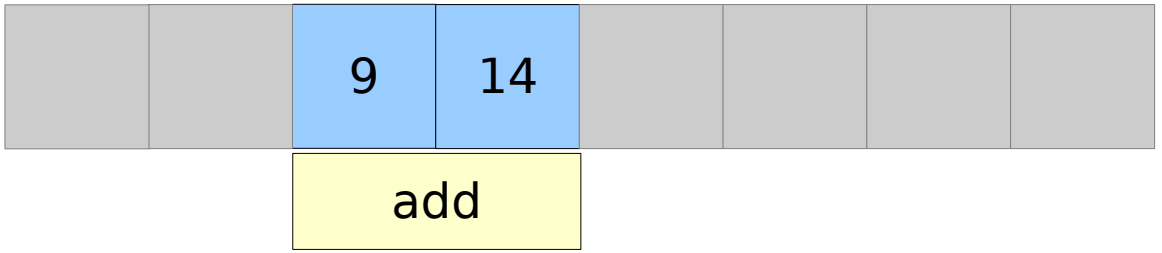
Register operations



Register-based

Stack operations

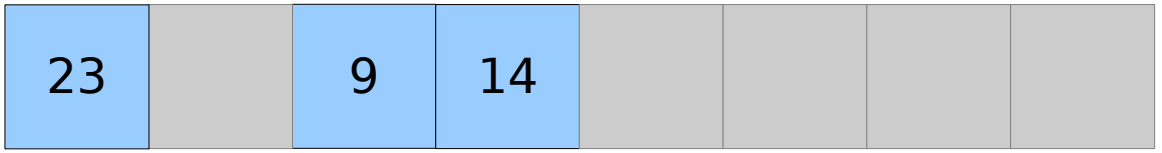
Register operations



Register-based

Stack operations

Register operations



Register-based

Stack operations

Register operations

Fewer instructions

Hardware registers

Register spilling

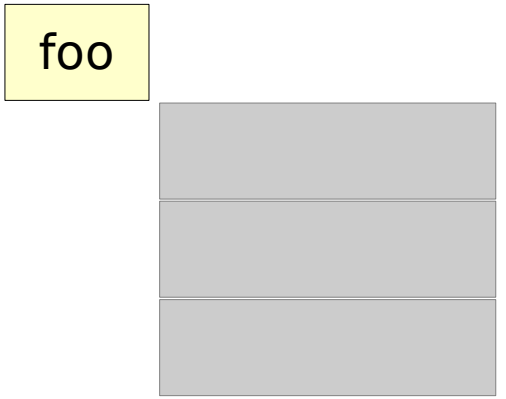
Flexible register sets

Continuation Passing Style

Stack-based control flow

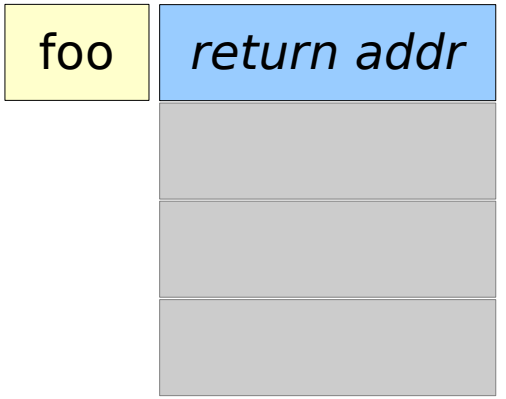
Continuation Passing Style

Stack-based control flow



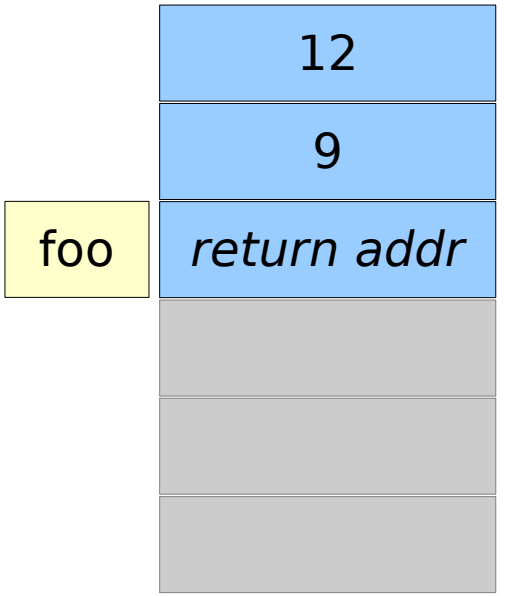
Continuation Passing Style

Stack-based control flow



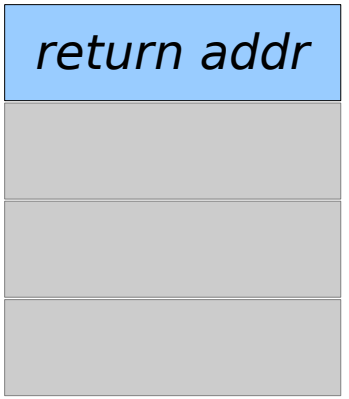
Continuation Passing Style

Stack-based control flow



Continuation Passing Style

Stack-based control flow



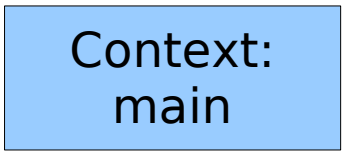
Continuation Passing Style

Stack-based control flow

Continuation Passing Style

Stack-based control flow

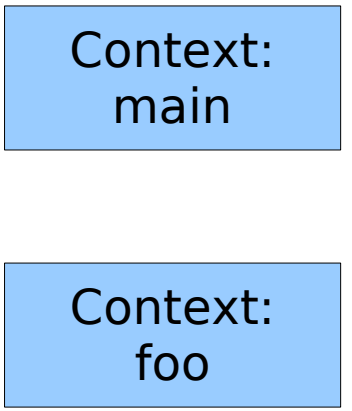
Continuation-based
control flow



Continuation Passing Style

Stack-based control flow

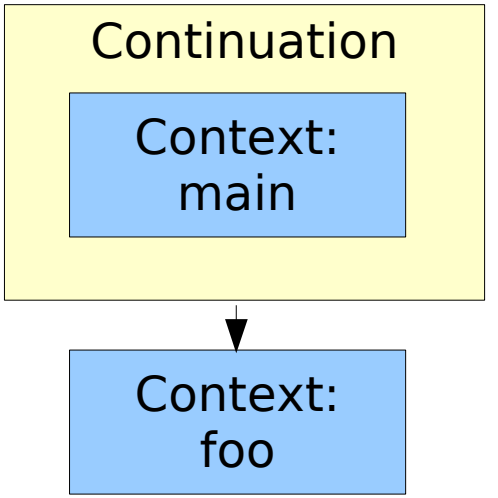
Continuation-based
control flow



Continuation Passing Style

Stack-based control flow

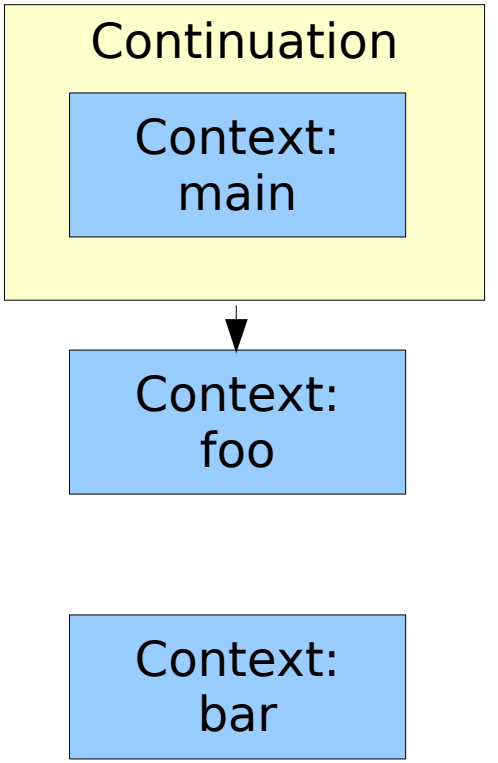
Continuation-based control flow



Continuation Passing Style

Stack-based control flow

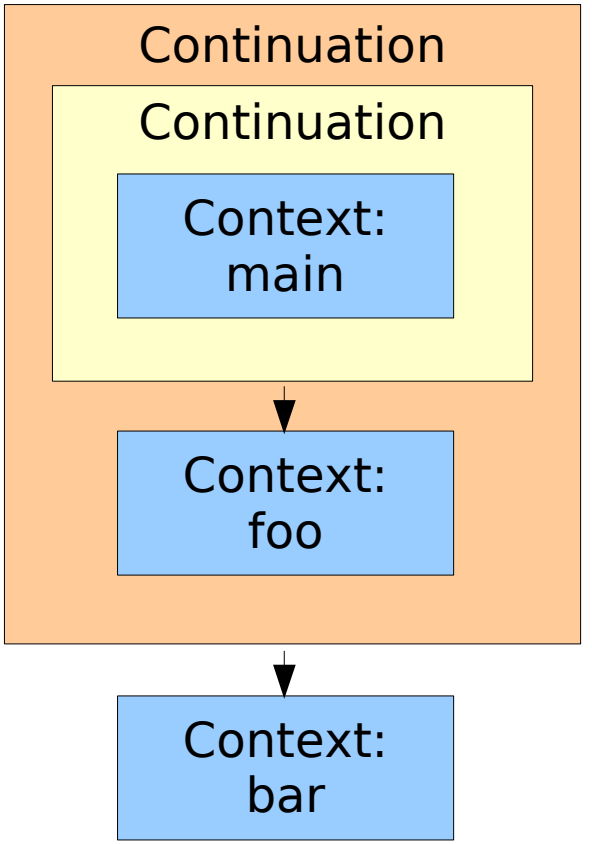
Continuation-based control flow



Continuation Passing Style

Stack-based control flow

Continuation-based control flow



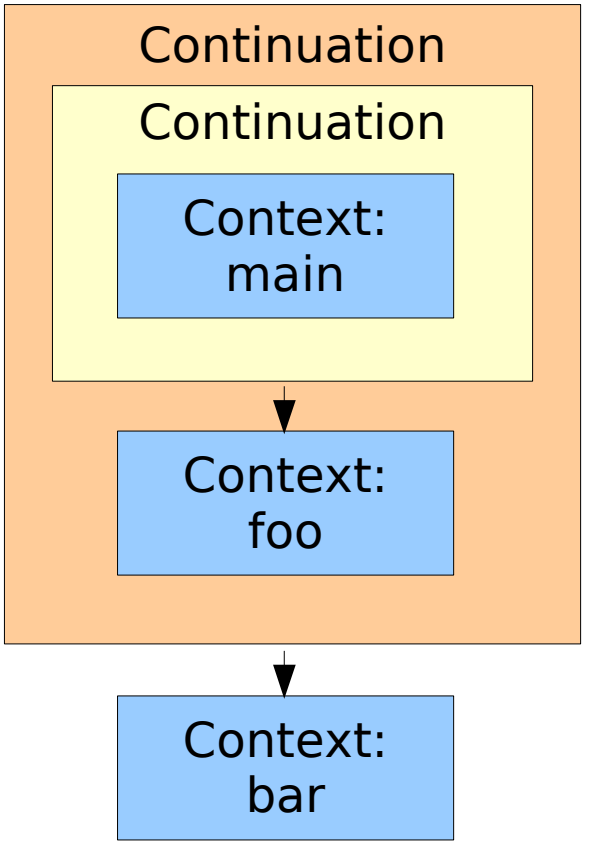
Continuation Passing Style

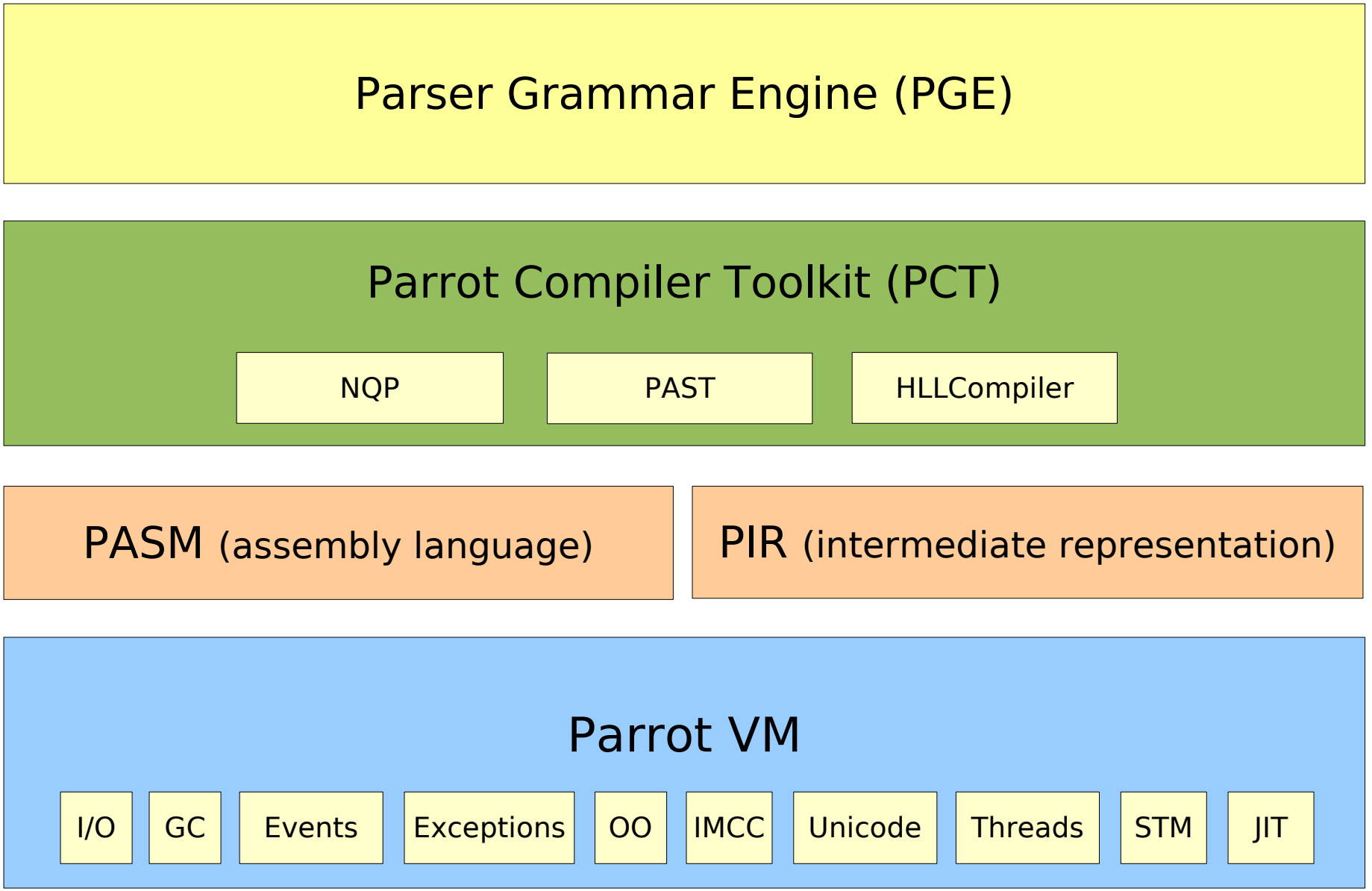
Stack-based control flow

Continuation-based control flow

Deeply nested contexts

Tail recursion




Parser Grammar Engine (PGE)
Parrot Compiler Toolkit (PCT)
NQP
PAST
HLLCompiler
PASM (assembly language)
PIR (intermediate representation)
Parrot VM
I/O
GC
Events
Exceptions
OO
IMCC
Unicode
Threads
STM
JIT

PASM

Assembly language

Simple syntax

```
add I0, I1, I2
```

Human-readable bytecode

PIR

Syntactic sugar

```
$I0 = $I1 + $I2
```

Named variables

```
.local int myvar
$I0 = myvar + 5
```

Sub and method calls

```
result = object.'method'($I0)
```

NQP

Not Quite P(HP|ython|erl|uby)

Lightweight language

```
$a := 1;
print($a, "\n");
```

Compiler tools

```
$past := PAST::Op.new( :name('printnl') );
```

Parser Grammar Engine

Regular expressions

Recursive descent

Operator precedence parser

HLLCompiler

Base library

Quick start

Common features

Pipp

Download

http://www.parrot.org

Build

```
$ perl Configure.PL
$ make test
```

Language

```
$ cd languages/pipp
$ make test
```

Pipp

hello.php

```
<?php
echo "Hello, World!\n";
?>
```

Run

```
$ parrot pipp.pir hello.php
$ pipp hello.php
```

pipp.pir

367 lines

```
$P1 = new ['PCT';'HLLCompiler']
$P1.'language'('Pipp')
$P1.'parsegrammar'(['Pipp';'Grammar'])
$P1.'parseactions'(['Pipp';'Grammar';'Actions'])
```

grammar.pg

Parser

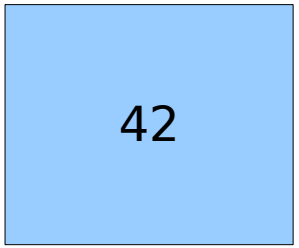
```
rule argument_list {
    [ <expression> [',' <expression>]* ]?
    {*}
}
```

actions.nqp

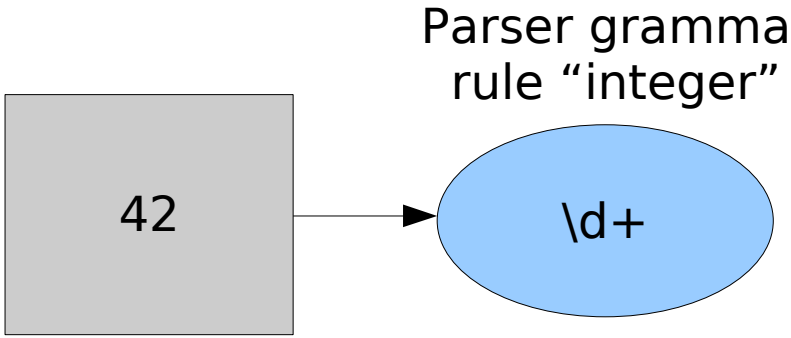
Transform to AST

```
method echo_statement($/) {
    my $past := $( $<argument_list> );
    $past.name( 'echo' );
    make $past;
}
```

Value Transformation

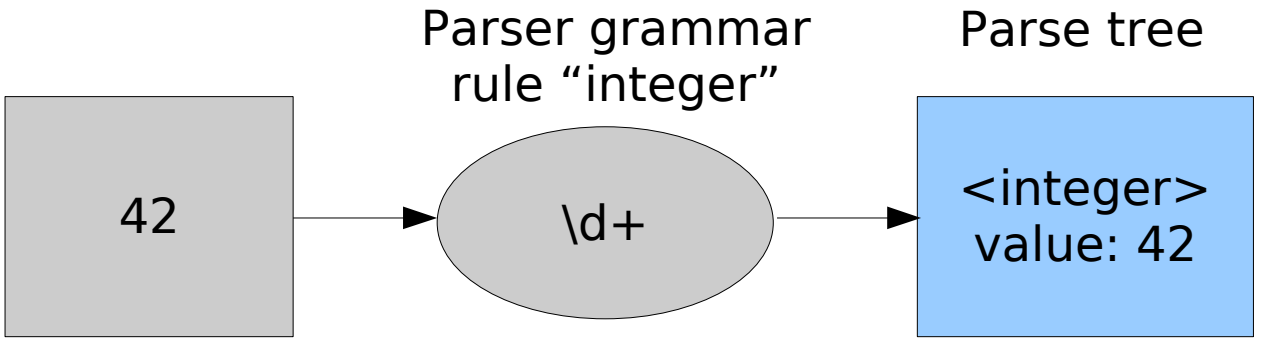


Value Transformation

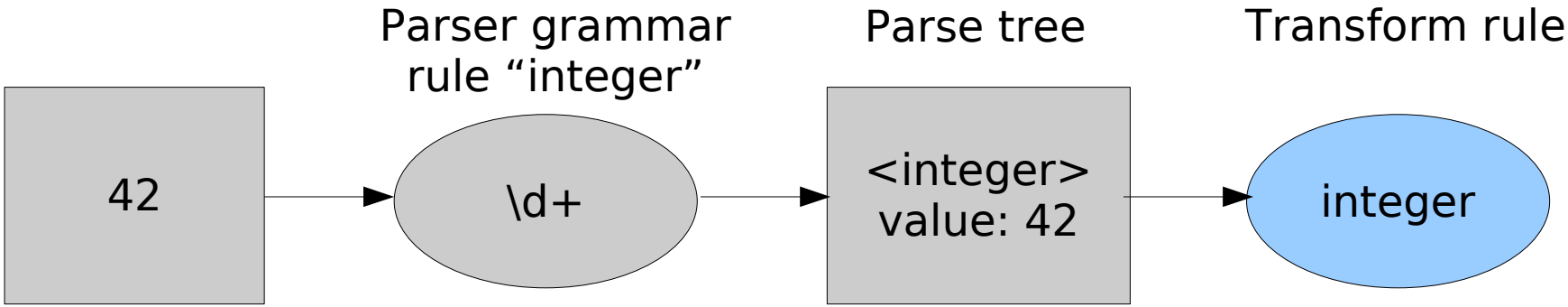


token integer { \d+ }

Value Transformation

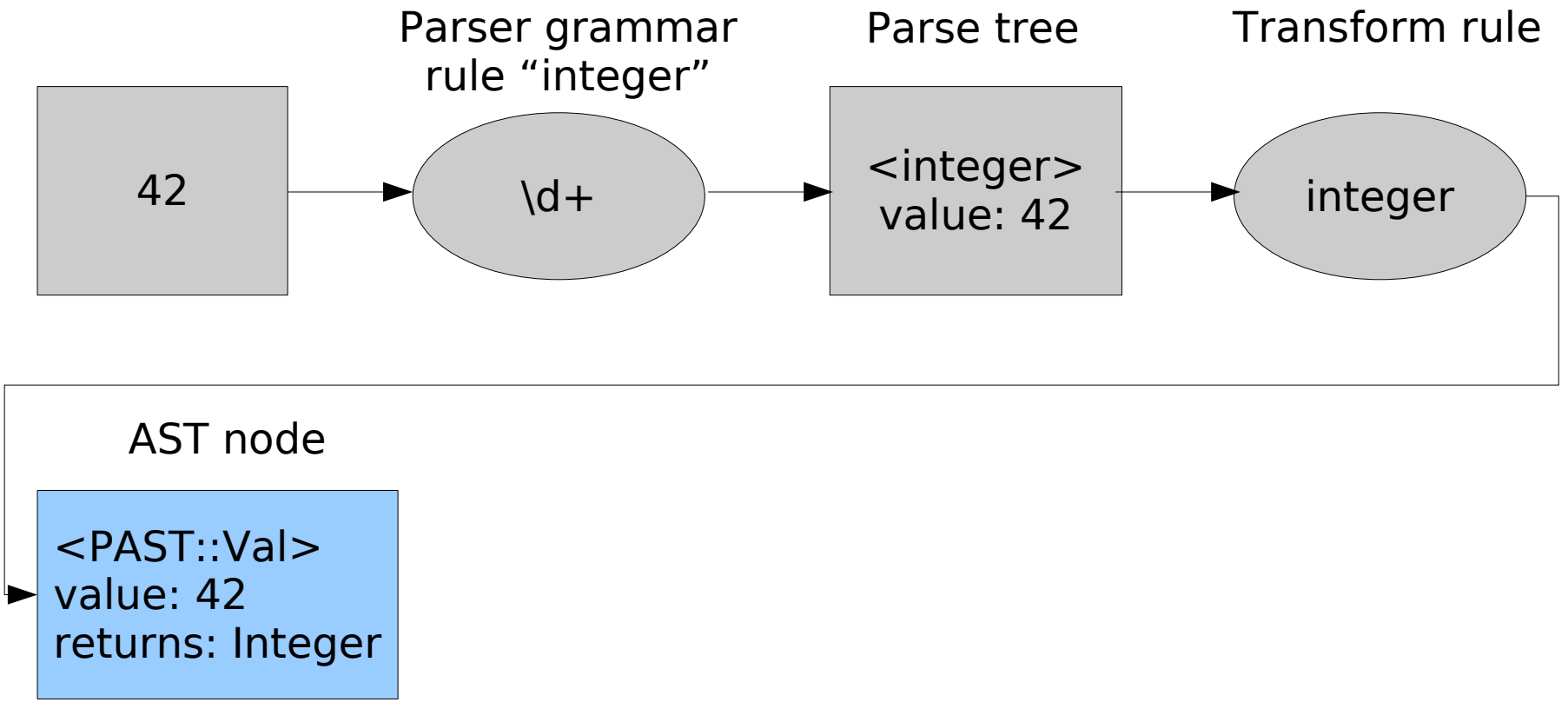


Value Transformation

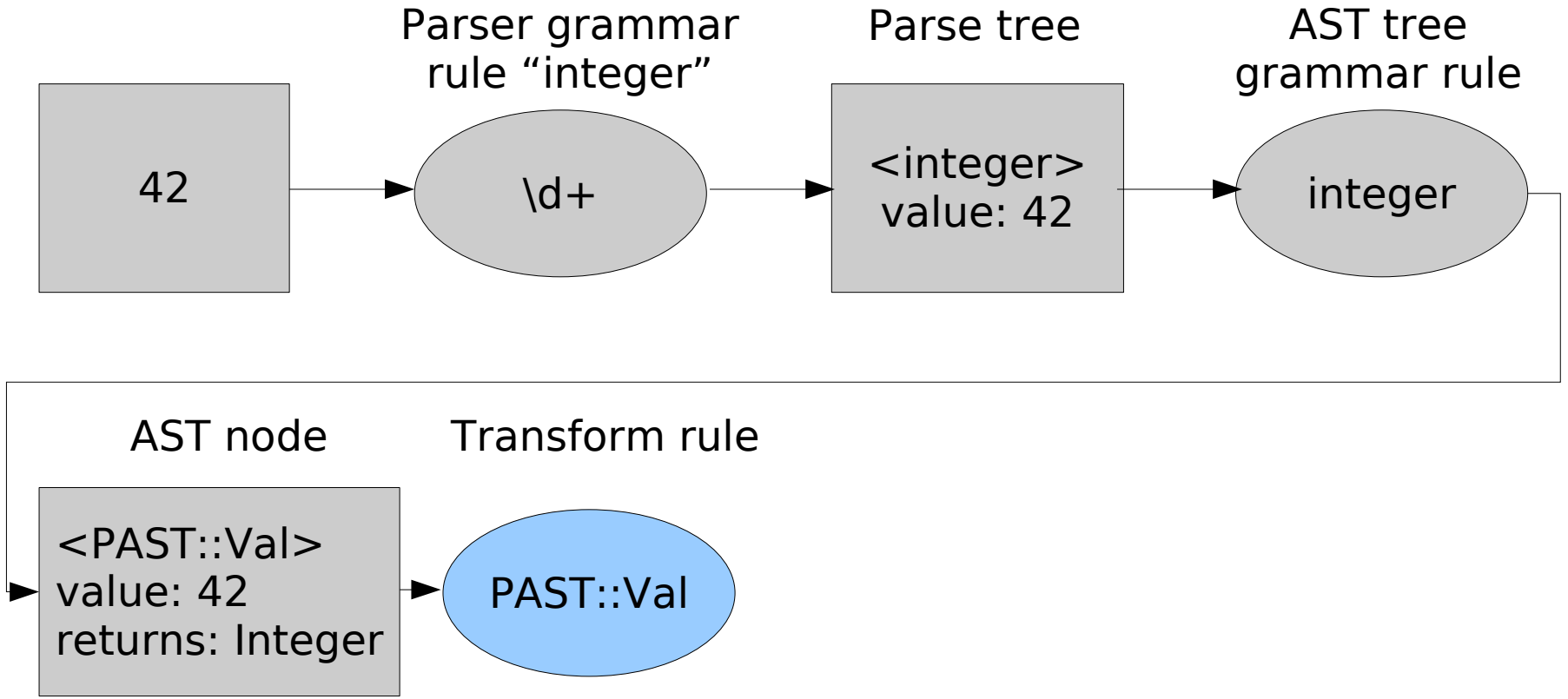


method integer($/) {...}

Value Transformation

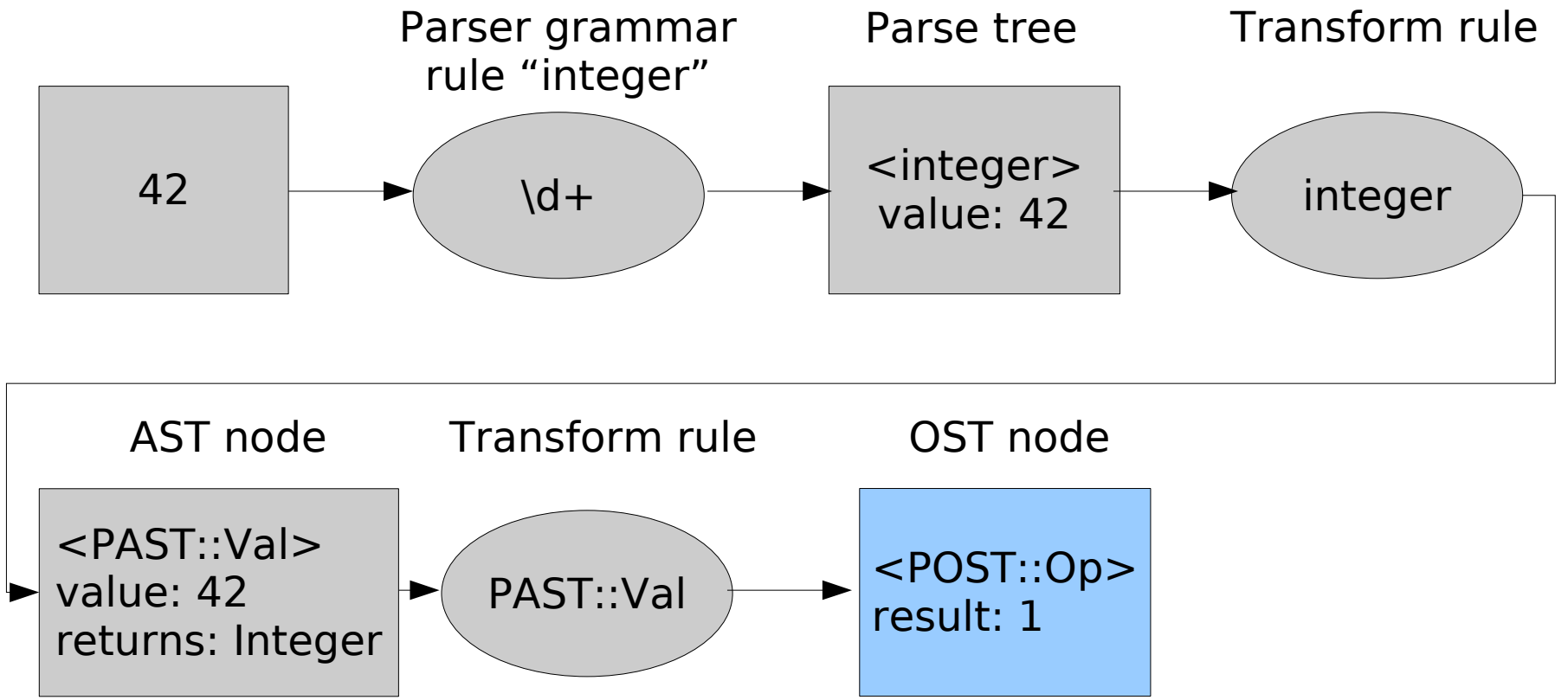


Value Transformation

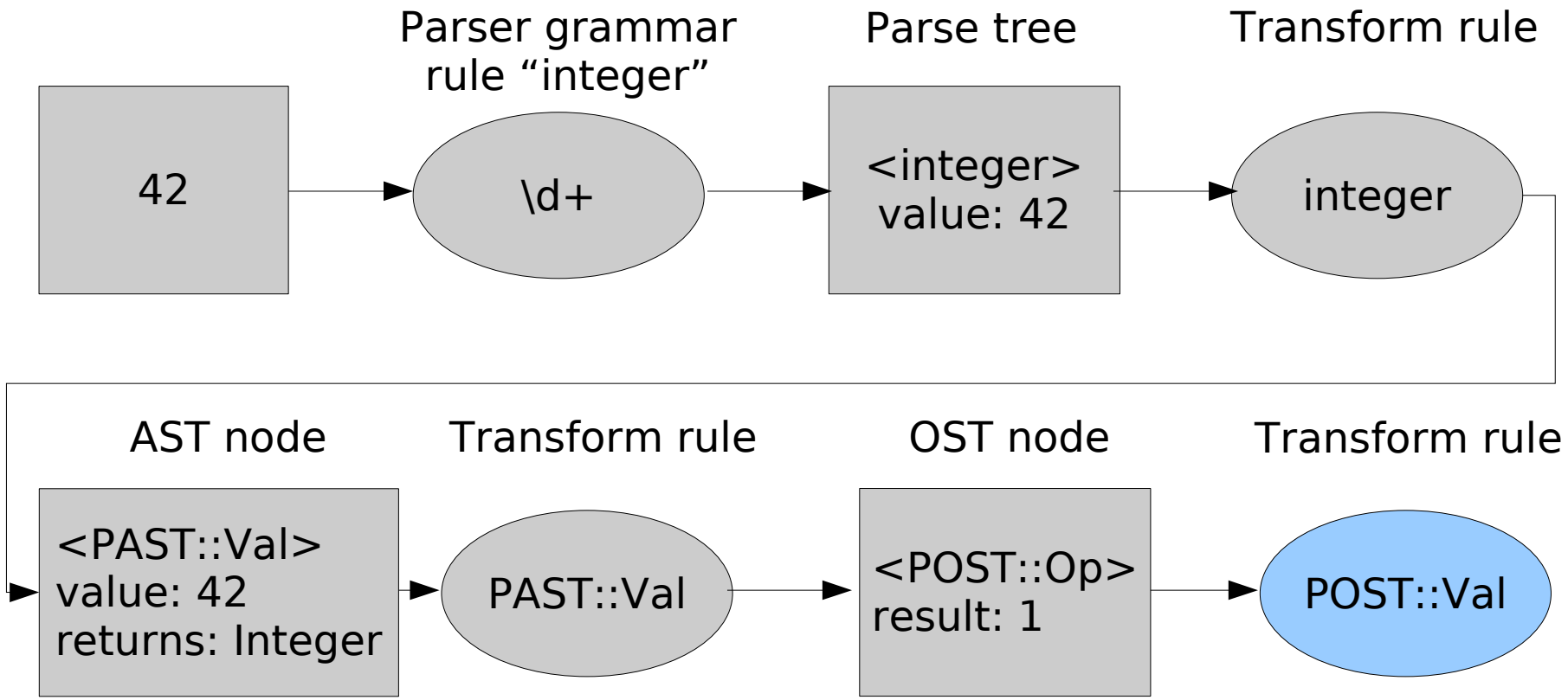


ost = self.as_post(ast)

Value Transformation

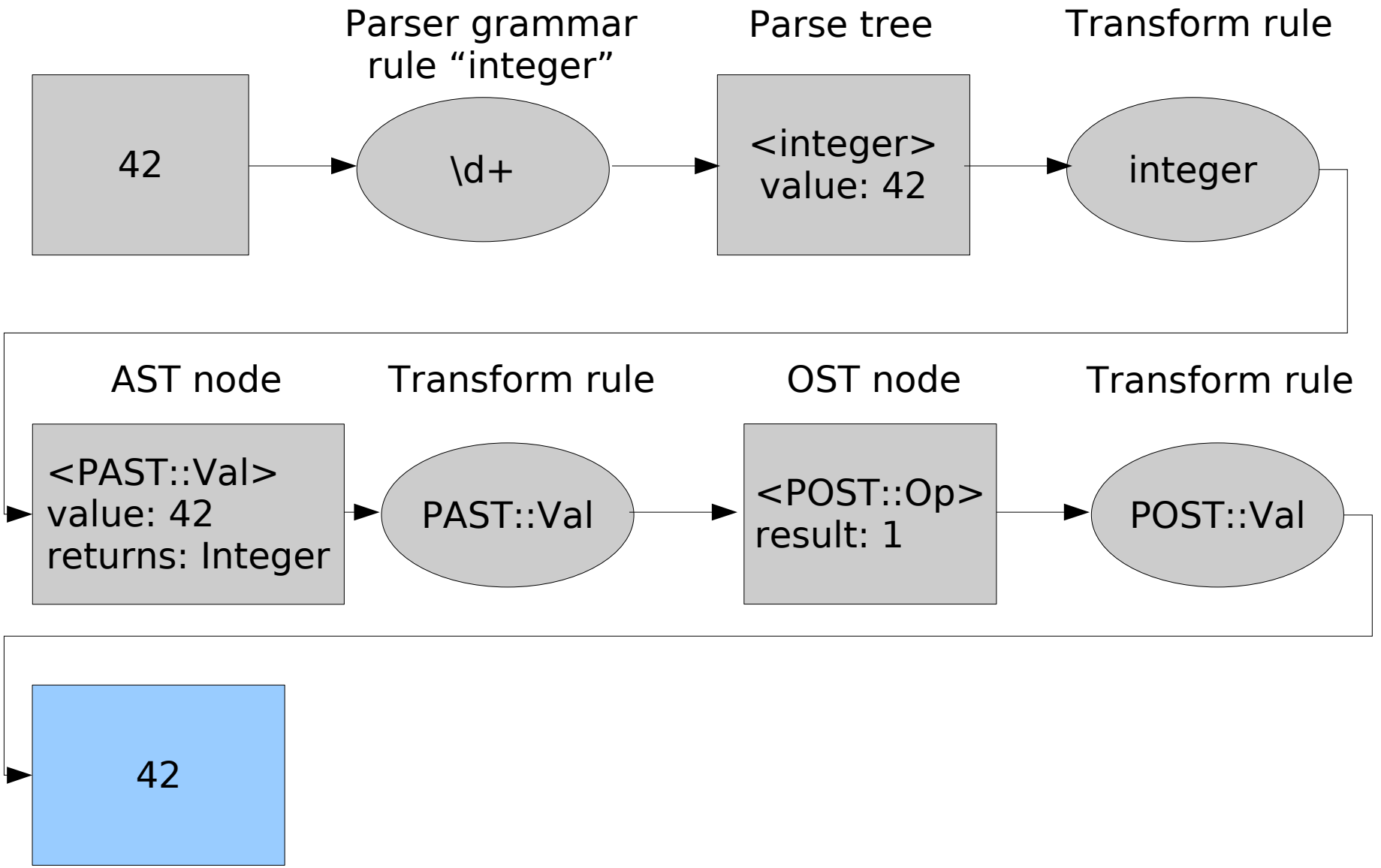


Value Transformation

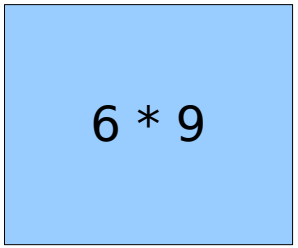


self.pir(ost)

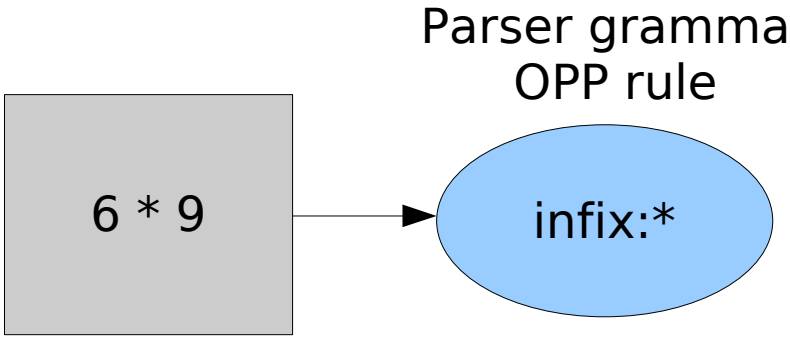
Value Transformation



Operator Transformation

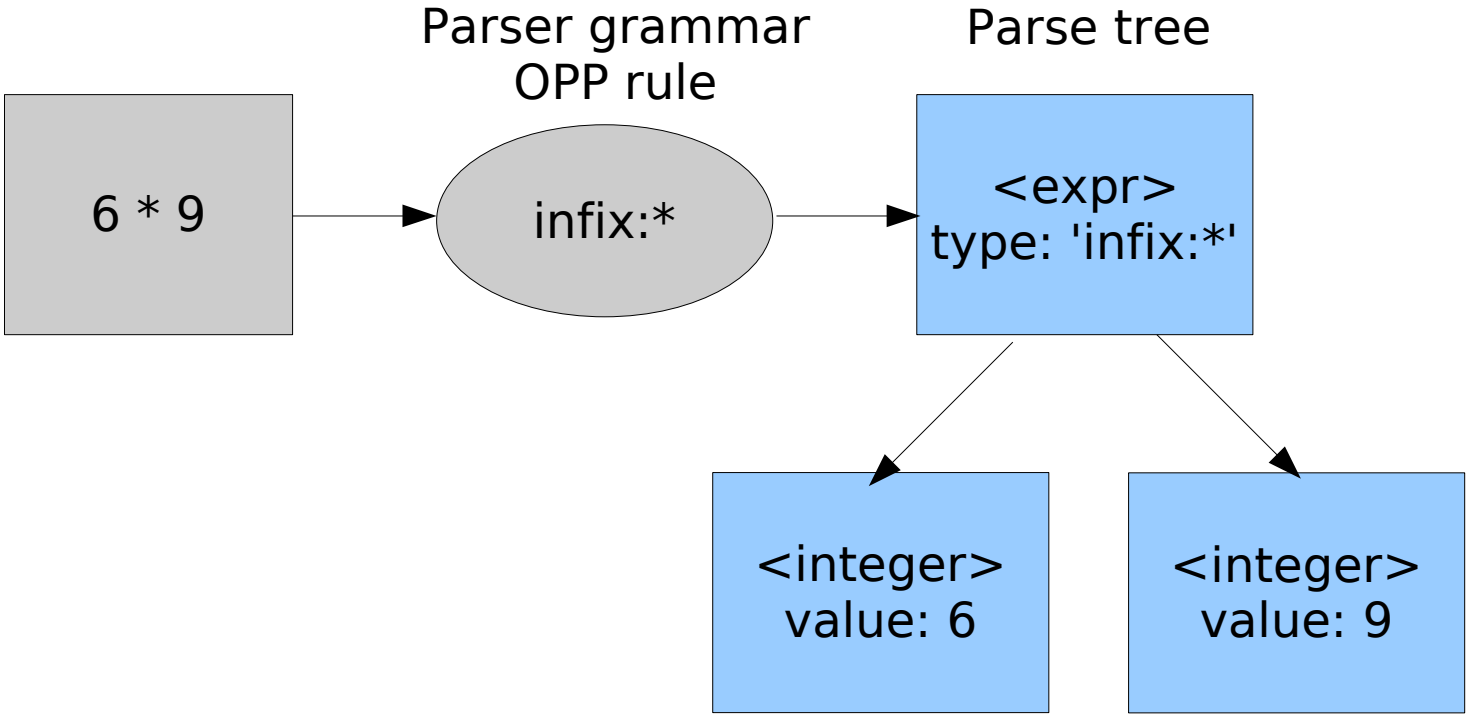


Operator Transformation

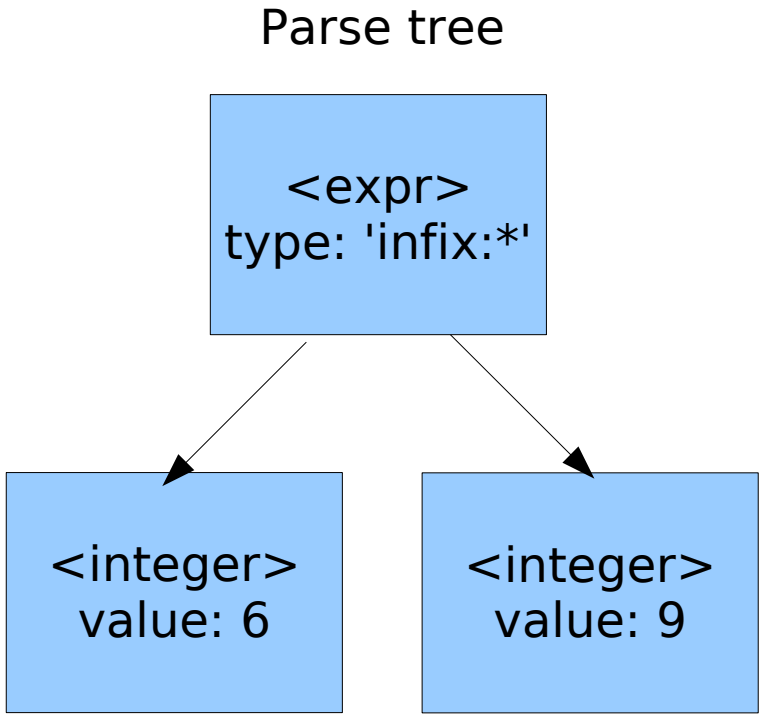


proto infix:* is looser(prefix:+) {...}

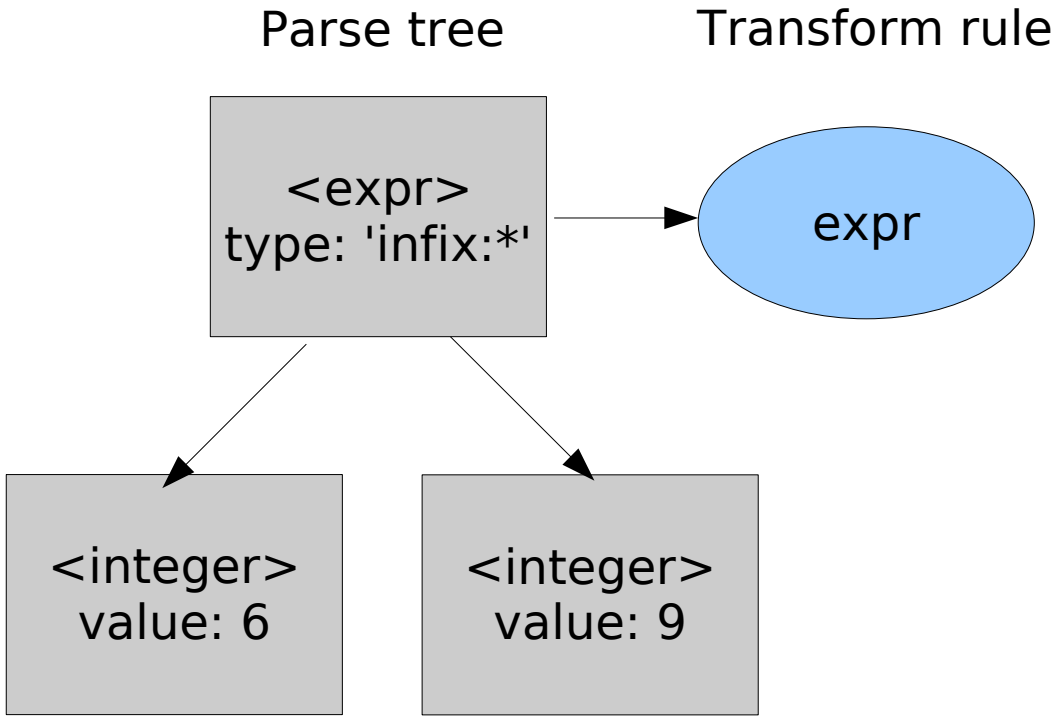
Operator Transformation



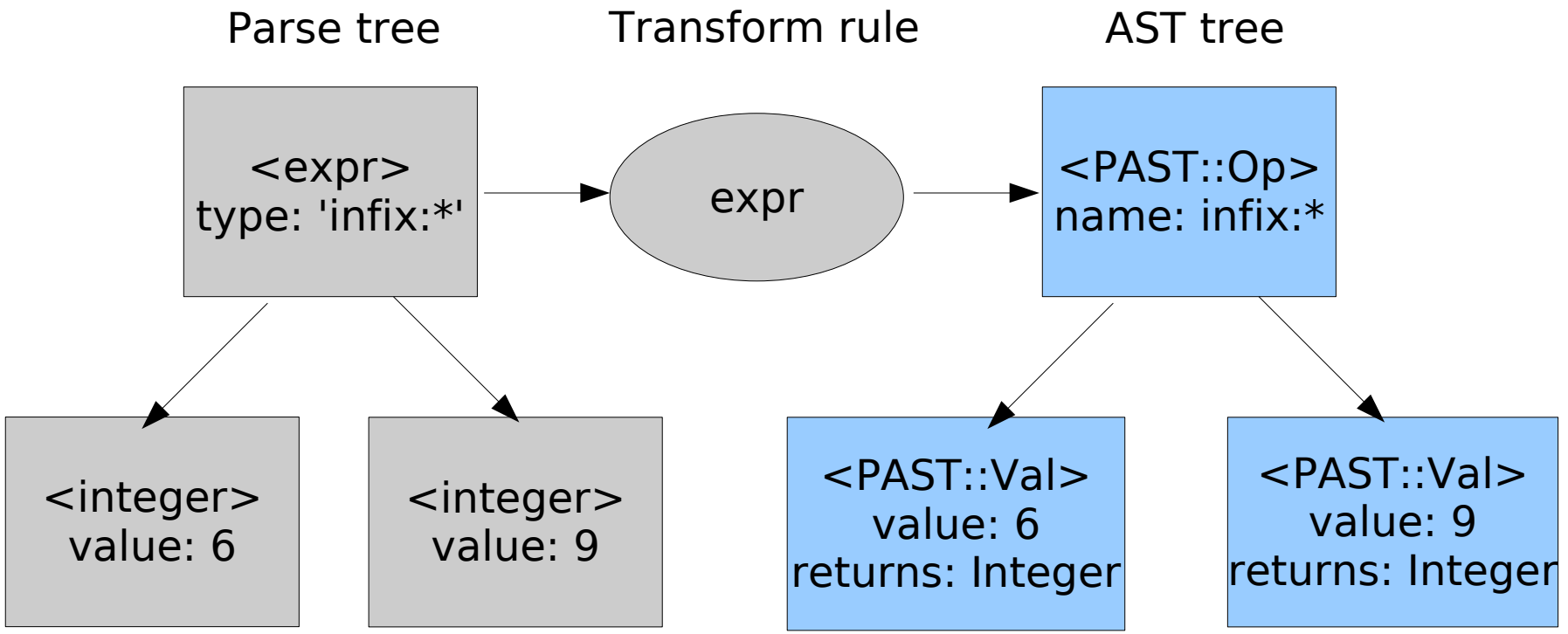
Operator Transformation



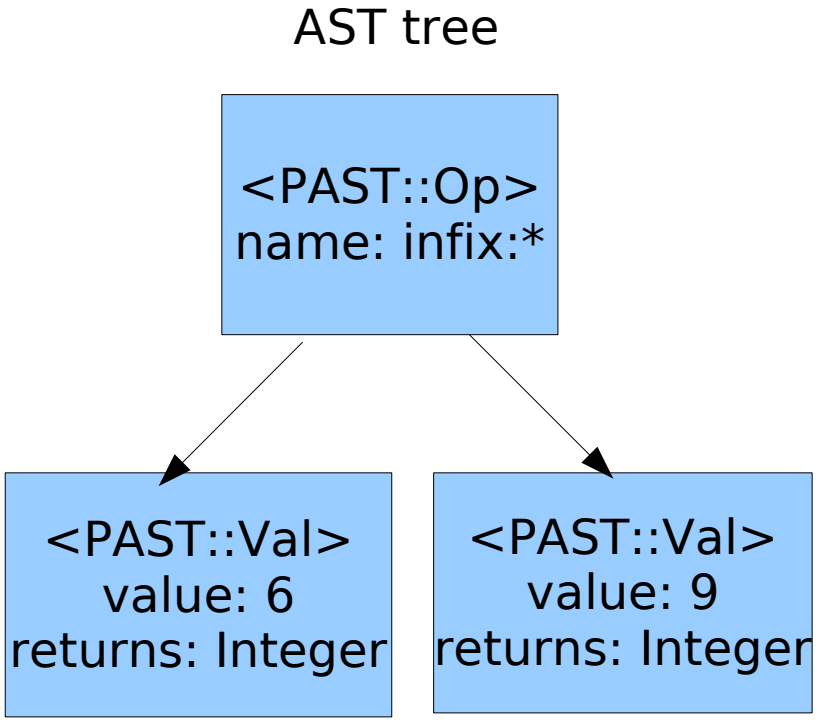
Operator Transformation



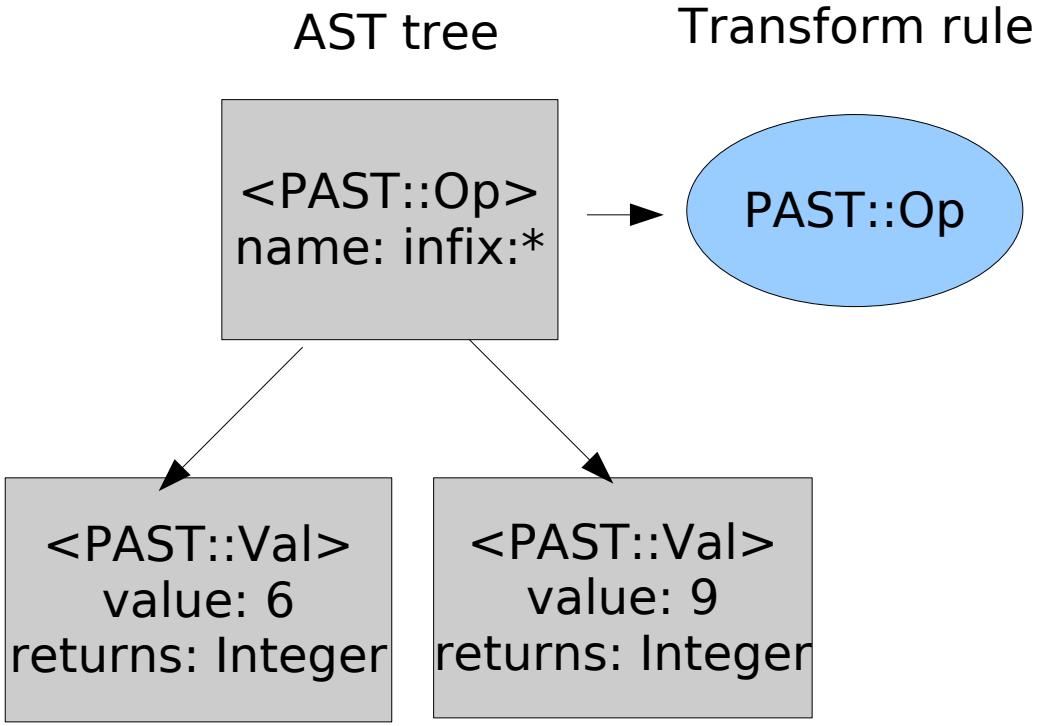
Operator Transformation



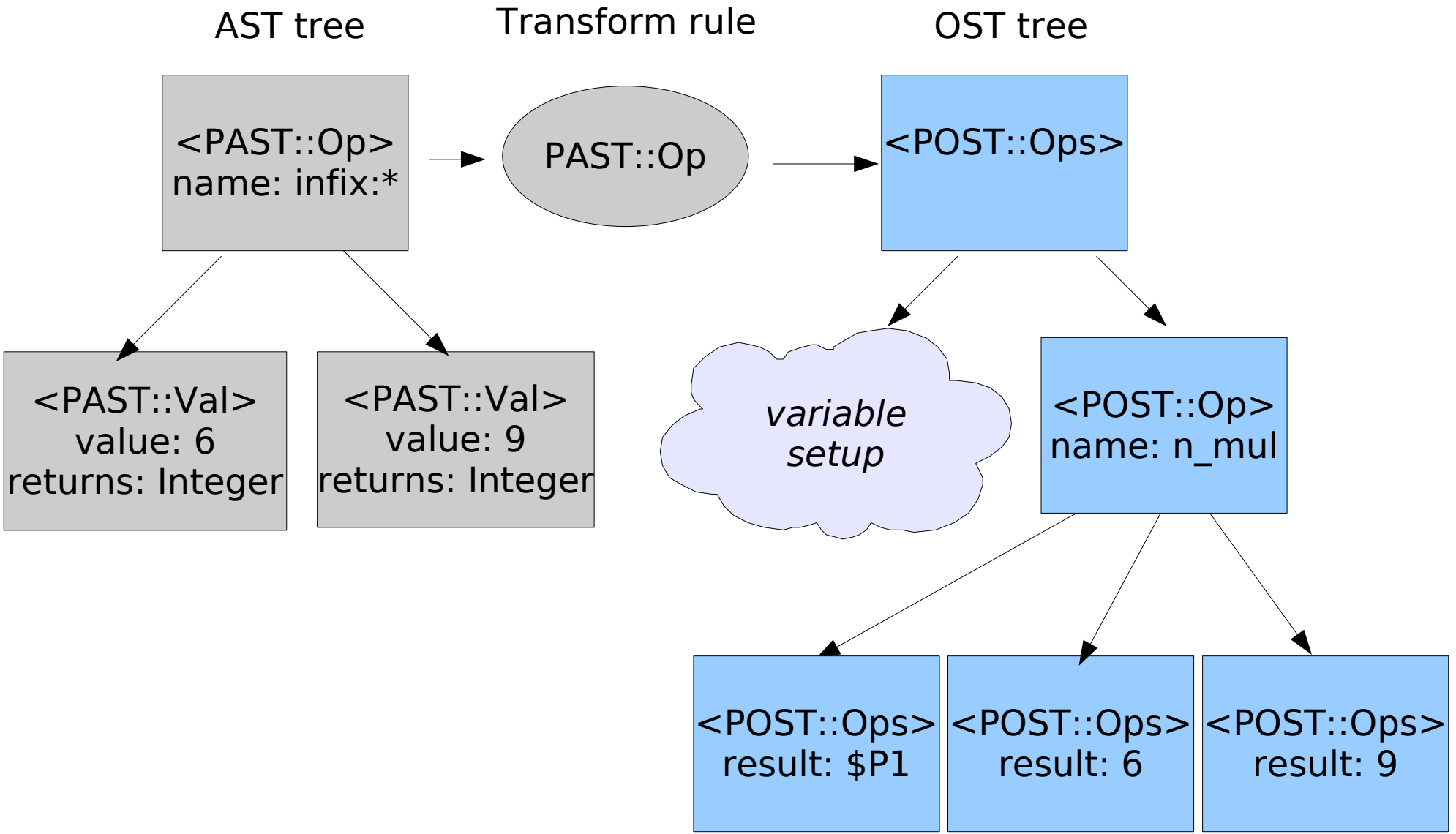
Operator Transformation



Operator Transformation



Operator Transformation



Operator Transformation

```
.sub _main :main
    new $P1, 'Integer'
    new $P2, 'Integer'
    set $P2, 6
    new $P3, 'Integer'
    set $P3, 9
    mul $P1, $P2, $P3
.end
```

Examples

In the Parrot distribution:

`examples/tutorial/*.pir`

Questions?

Further Reading

“Continuations and advanced flow control” by Jonathan Bartlett
<http://www.ibm.com/developerworks/linux/library/l-advflow.html>

“The case for virtual register machines” by Brian Davis, et al.

<http://portal.acm.org/citation.cfm?id=858575>

Pipp project site

http://www.pipp.org