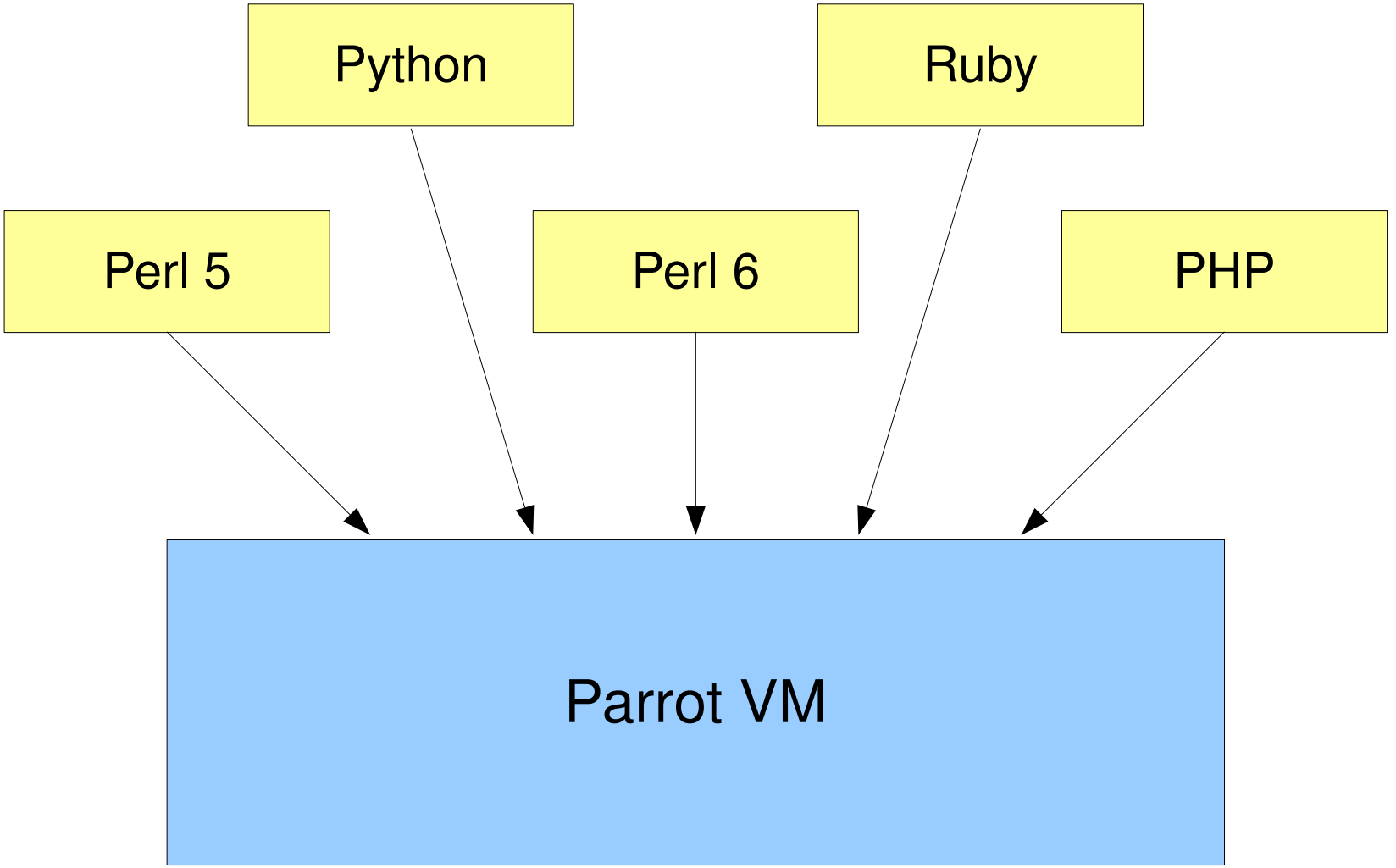
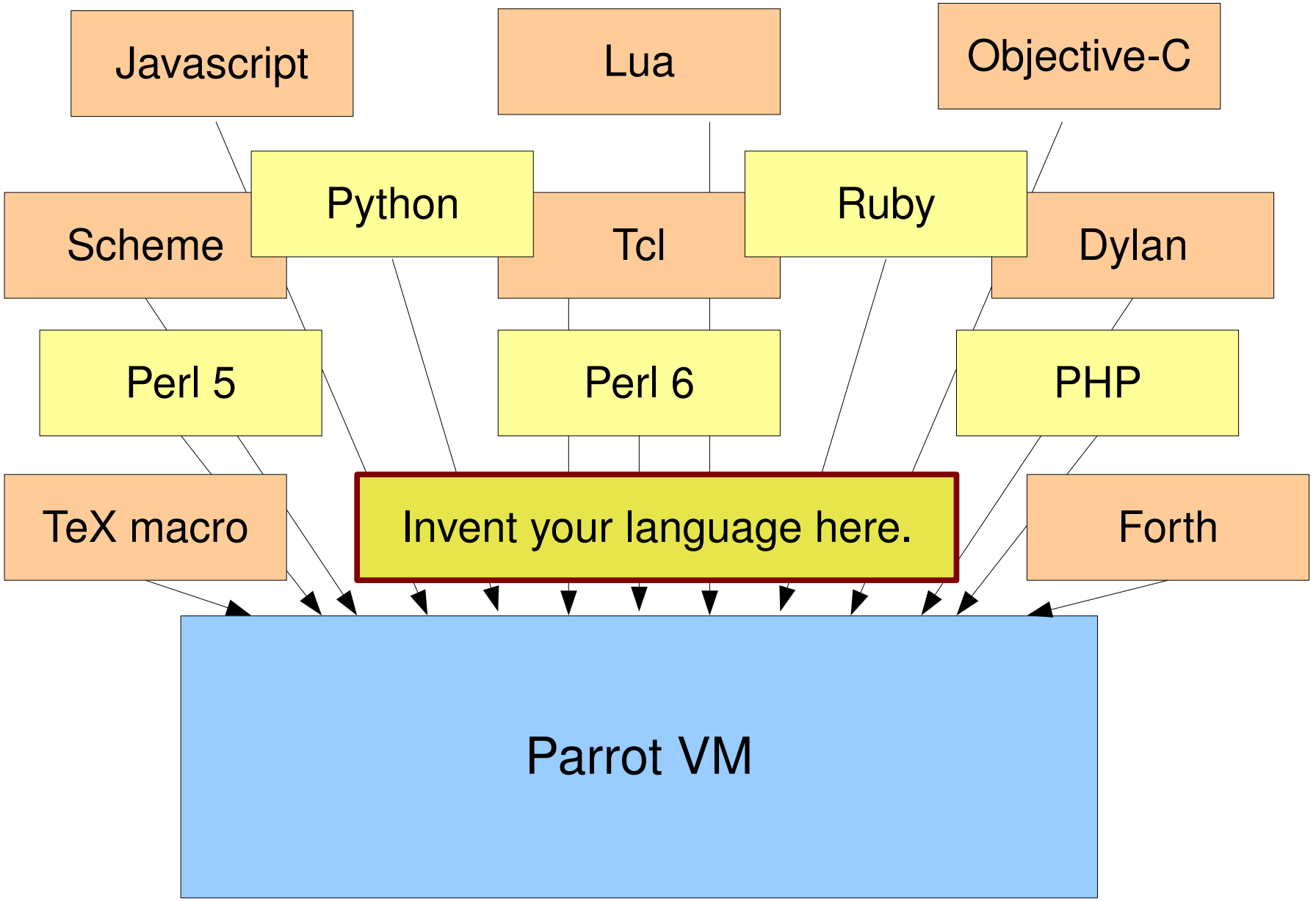
Parrot VM

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.


Python
Ruby
Perl 5
Perl 6
PHP
Parrot VM


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

Dynamic Features

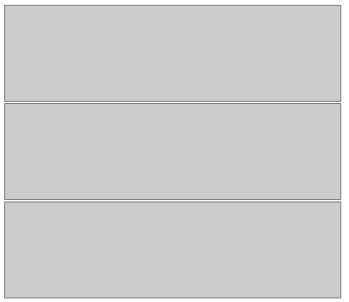
- Runtime vs. compile-time
- Extend code (eval, load)
- Define classes
- Alter type system
- Higher-order functions
- Closures, Continuations, Coroutines

Goals

- Revolution...
- ...becoming Establishment
- Powerful tools
- Portability
- Interoperability
- Language evolution

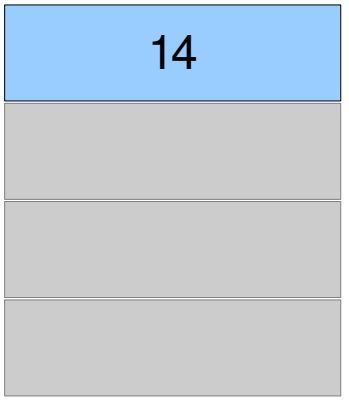
Register-based

- Stack operations



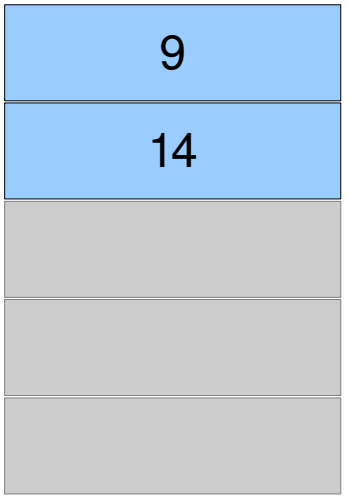
Register-based

- Stack operations



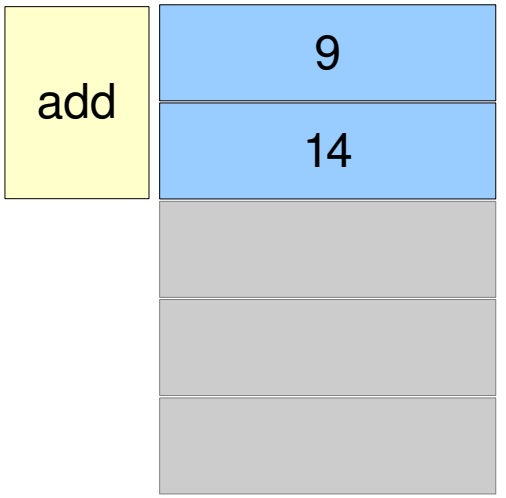
Register-based

- Stack operations



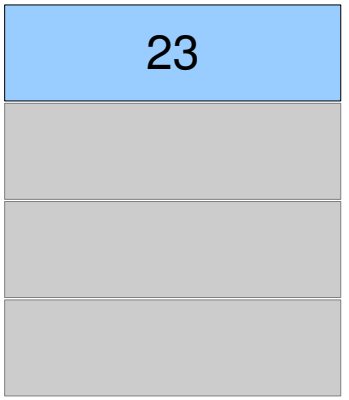
Register-based

- Stack operations



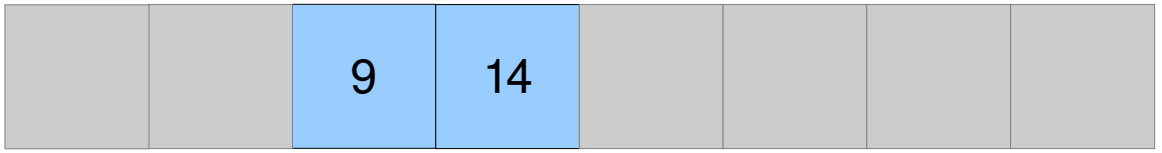
Register-based

- Stack operations



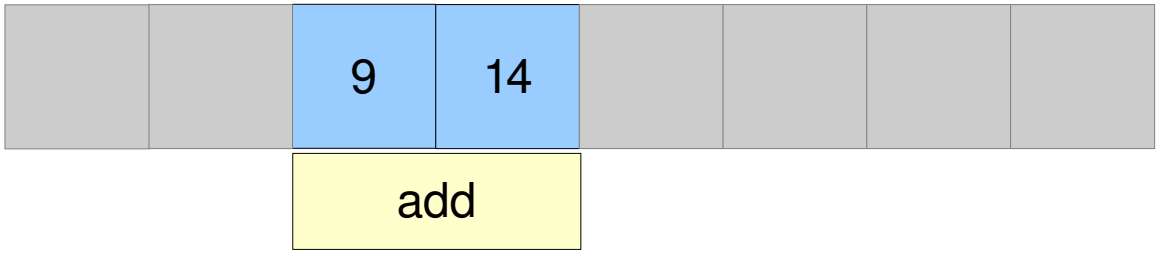
Register-based

- Stack operations
- Register operations



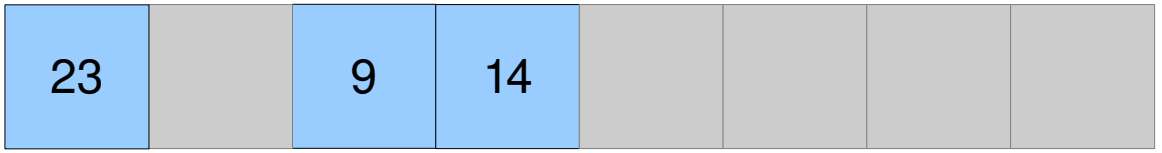
Register-based

- Stack operations
- Register operations



Register-based

- Stack operations
- Register operations



Register-based

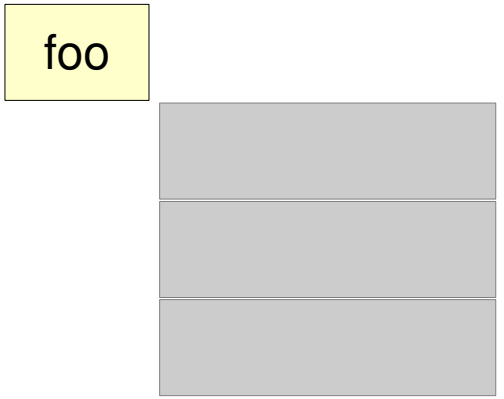
- Stack operations
- Register operations
- Fewer instructions
- Hardware registers
- Register spilling
- Flexible register sets

Continuation Passing Style

- Stack-based control flow

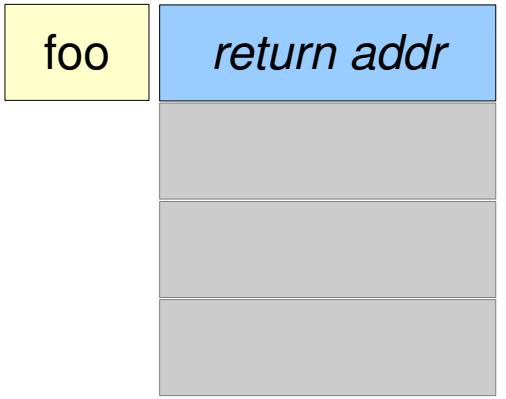
Continuation Passing Style

- Stack-based control flow



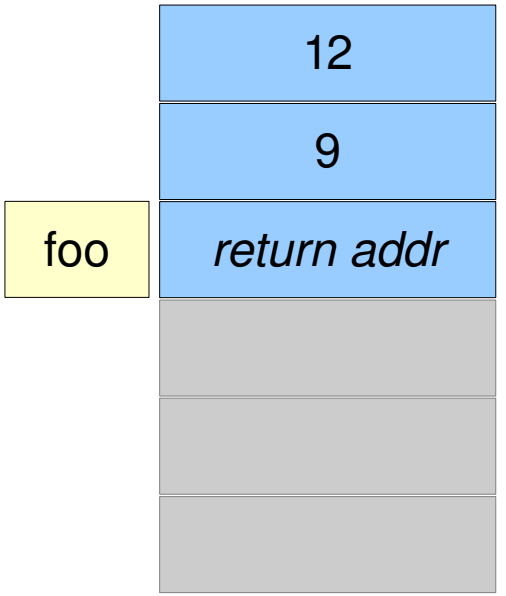
Continuation Passing Style

- Stack-based control flow



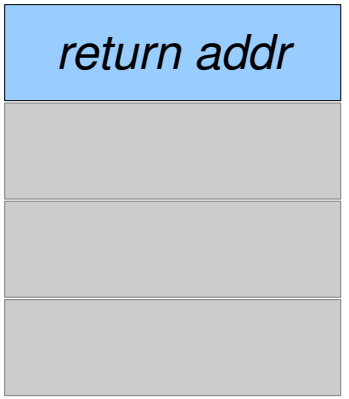
Continuation Passing Style

- Stack-based control flow



Continuation Passing Style

- Stack-based control flow

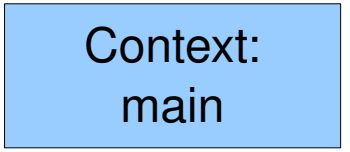


Continuation Passing Style

- Stack-based control flow

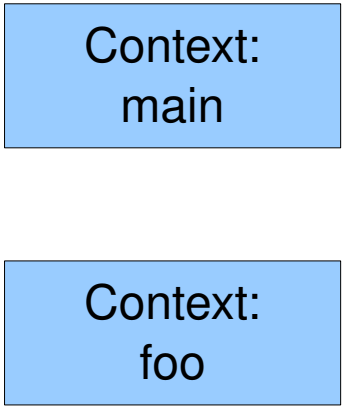
Continuation Passing Style

- Stack-based control flow
- Continuation-based control flow



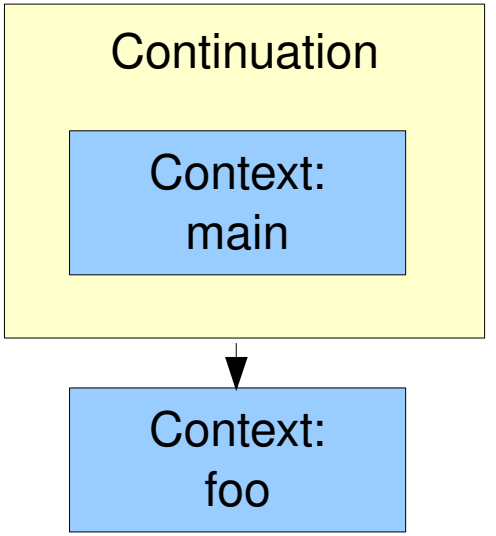
Continuation Passing Style

- Stack-based control flow
- Continuation-based control flow



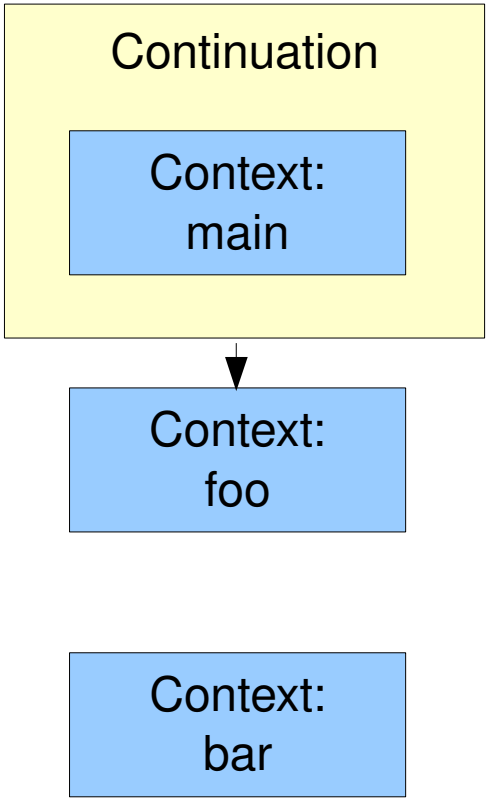
Continuation Passing Style

- Stack-based control flow
- Continuation-based control flow



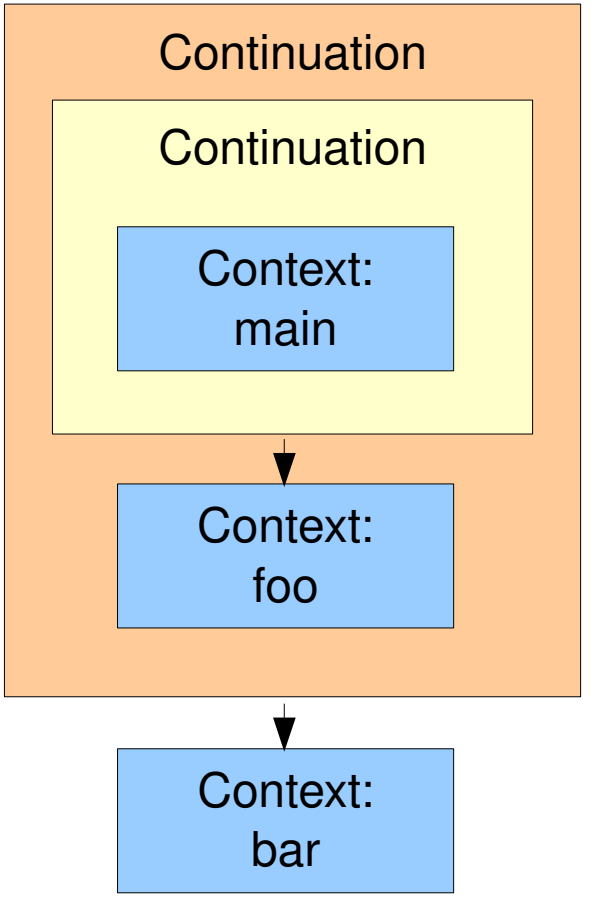
Continuation Passing Style

- Stack-based control flow
- Continuation-based control flow



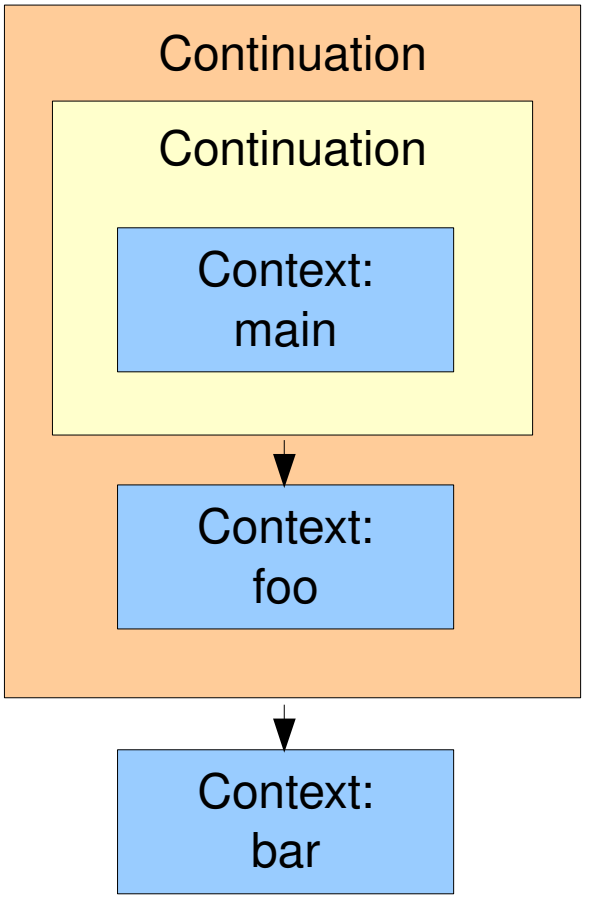
Continuation Passing Style

- Stack-based control flow
- Continuation-based control flow



Continuation Passing Style

- Stack-based control flow
- Continuation-based control flow
- Linked list
- Cheap continuations
- Deeply nested contexts
- Tail recursion



Dynamic Dispatch

- Runtime variation
- Optional parameters
- Named parameters
- Aggregating parameters
- Multiple dispatch
- Dispatch by value

Dynamic Dispatch

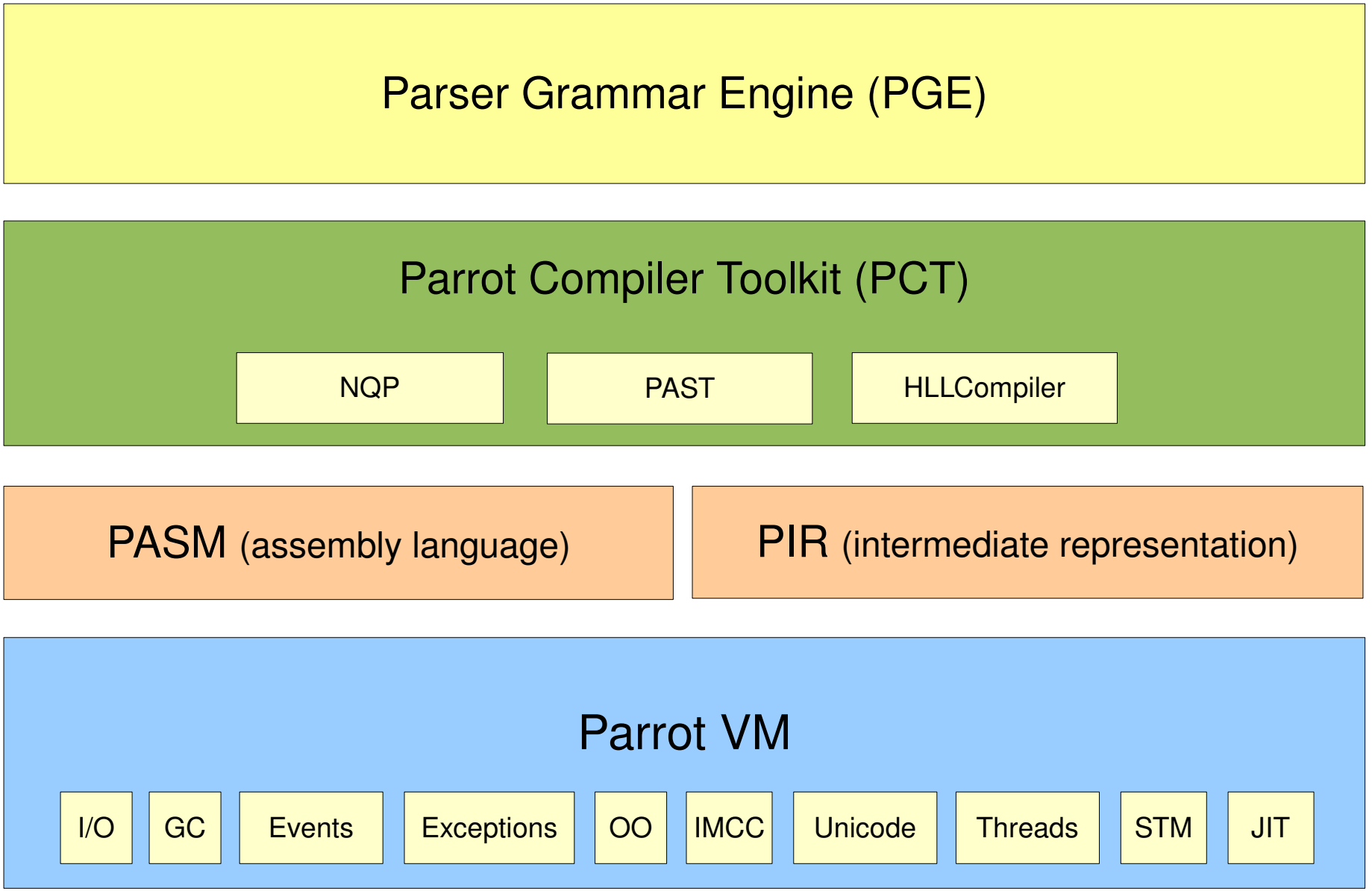
- Difficult?
- Static call munging
- Function as object
- Integrate CPS
- Encapsulated dispatch
- Concurrency

Dynamic Optimization

- Procrastination principle
- Late binding
- Lazy evaluation
- Just-in-time compilation
- Polymorphic inline cache
- Scripting
- Startup time

Dynamic Parsing Tools

- Regular expressions
- Recursive descent
- Operator precedence parser
- Cheap backtracking
- Tree transformation


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(erl|ython|HP|uby)
- Lightweight language

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

- Compiler tools

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

PGE

- Parser Grammar Engine
- Regular expressions

```
rule integer { \d+ }
```

- Subrules
- Smart whitespace with comments

```
rule number {
  <integer>      # an integer, followed by
  '.' <integer> # a dot, and an integer
}
```

PCT

- Parrot Compiler Toolkit
- Abstract syntax tree (PAST)
- Opcode syntax tree (POST)
- HLLCompiler base library
- Quick start
- Common features

Squaak

- Download

```
http://www.parrot.org
```

- Build

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

- Language

```
$ cd examples/languages/squaak
$ make squaak
$ make test
```

Squaak

- hello.sq

```
print("Hello, World!")
```

- Run

```
$ ./squaak hello.sq
```

squaak.pir

- 94 lines

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

grammar.pg

- Parser

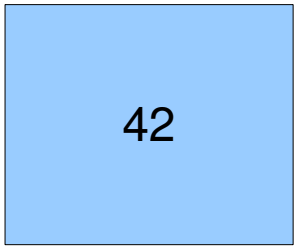
```
rule do_block {
    'do' <block> 'end'
    {*}
}
```

actions.nqp

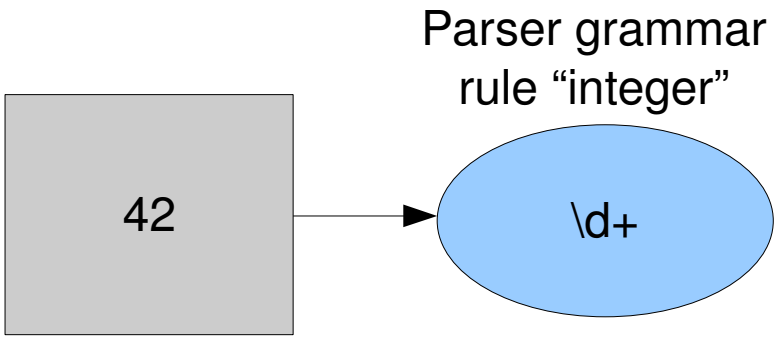
- Transform to AST

```
method do_block($/) {
    make $( $<block> );
}
```

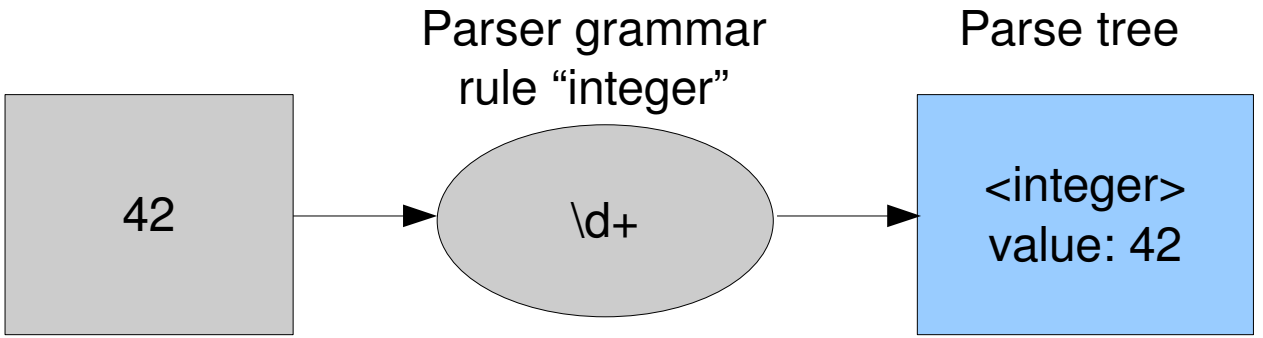
Value Transformation



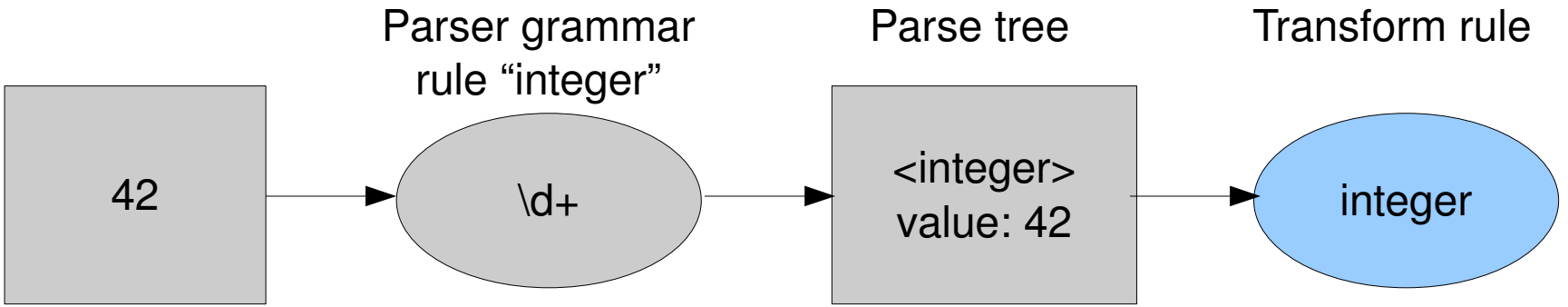
Value Transformation



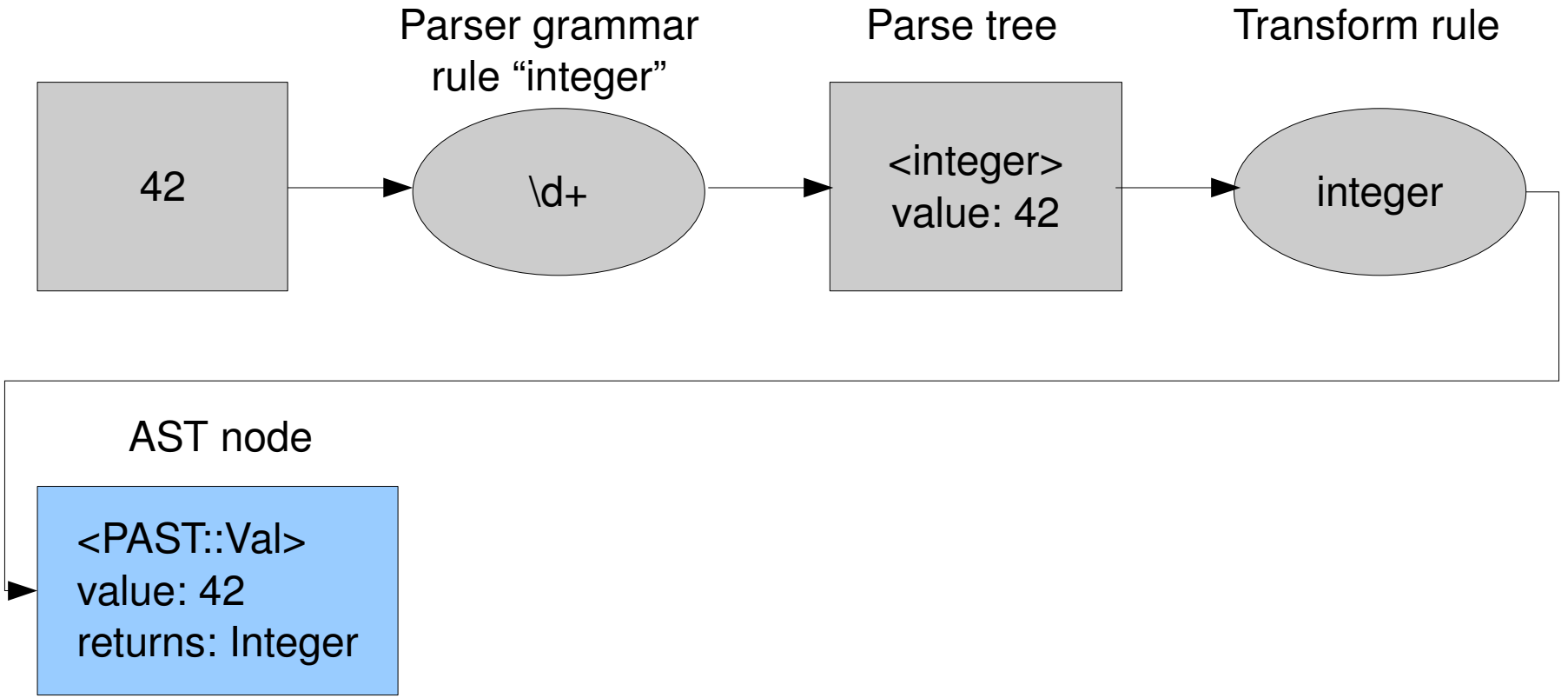
Value Transformation



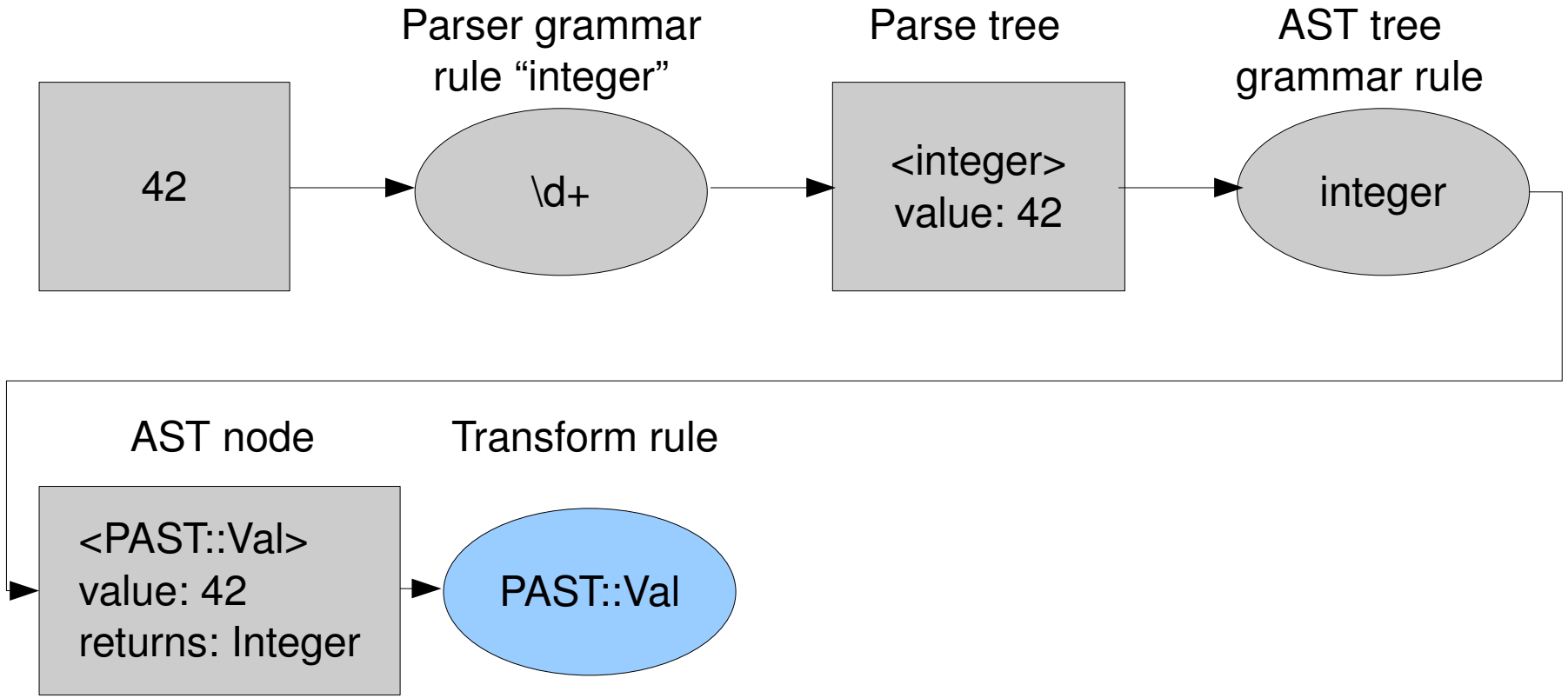
Value Transformation



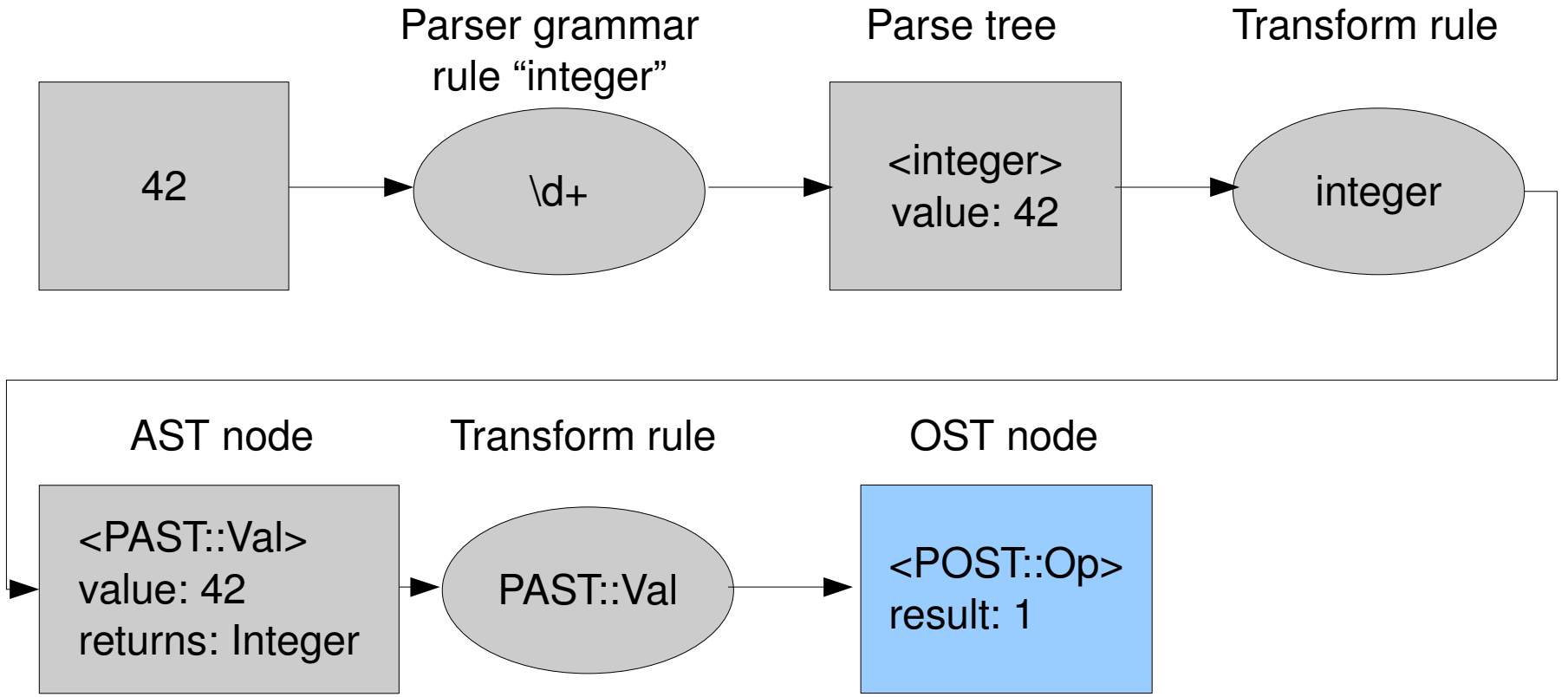
Value Transformation



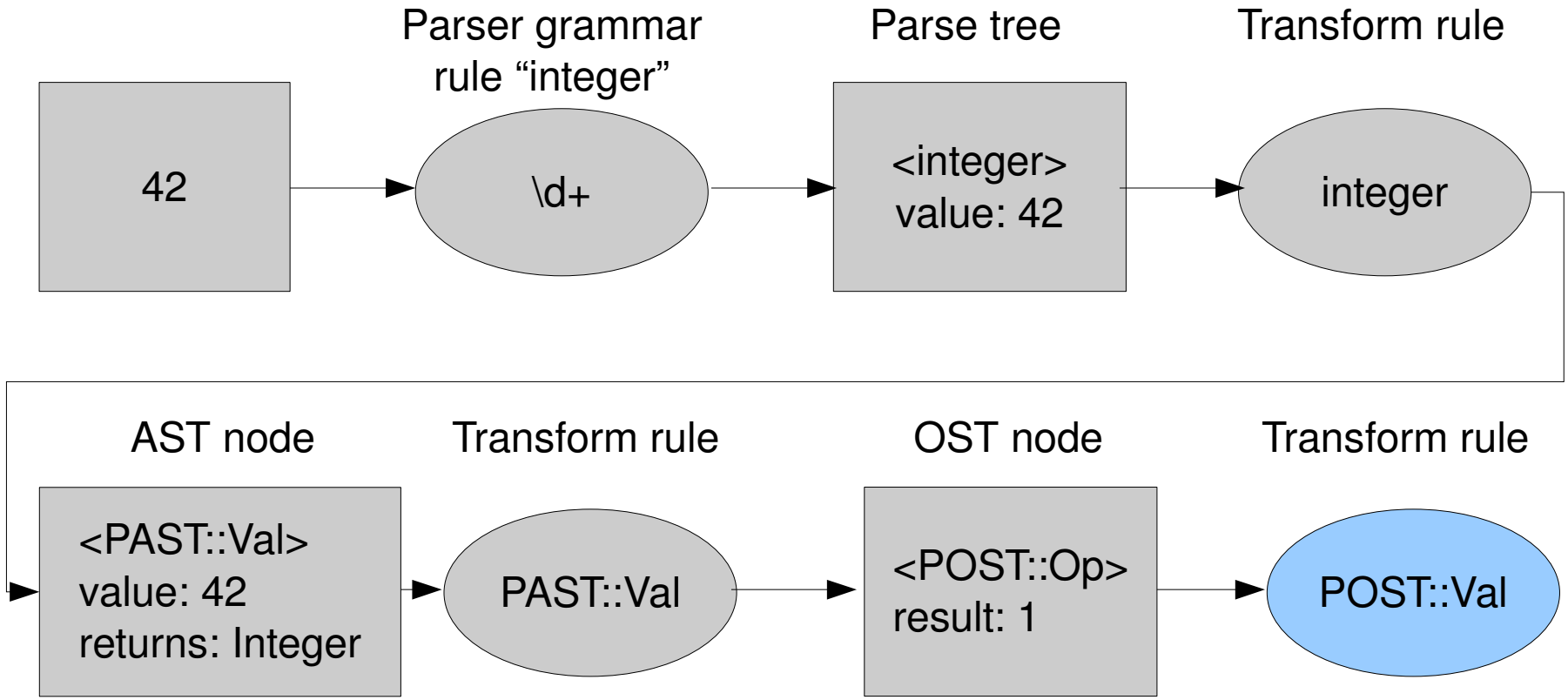
Value Transformation



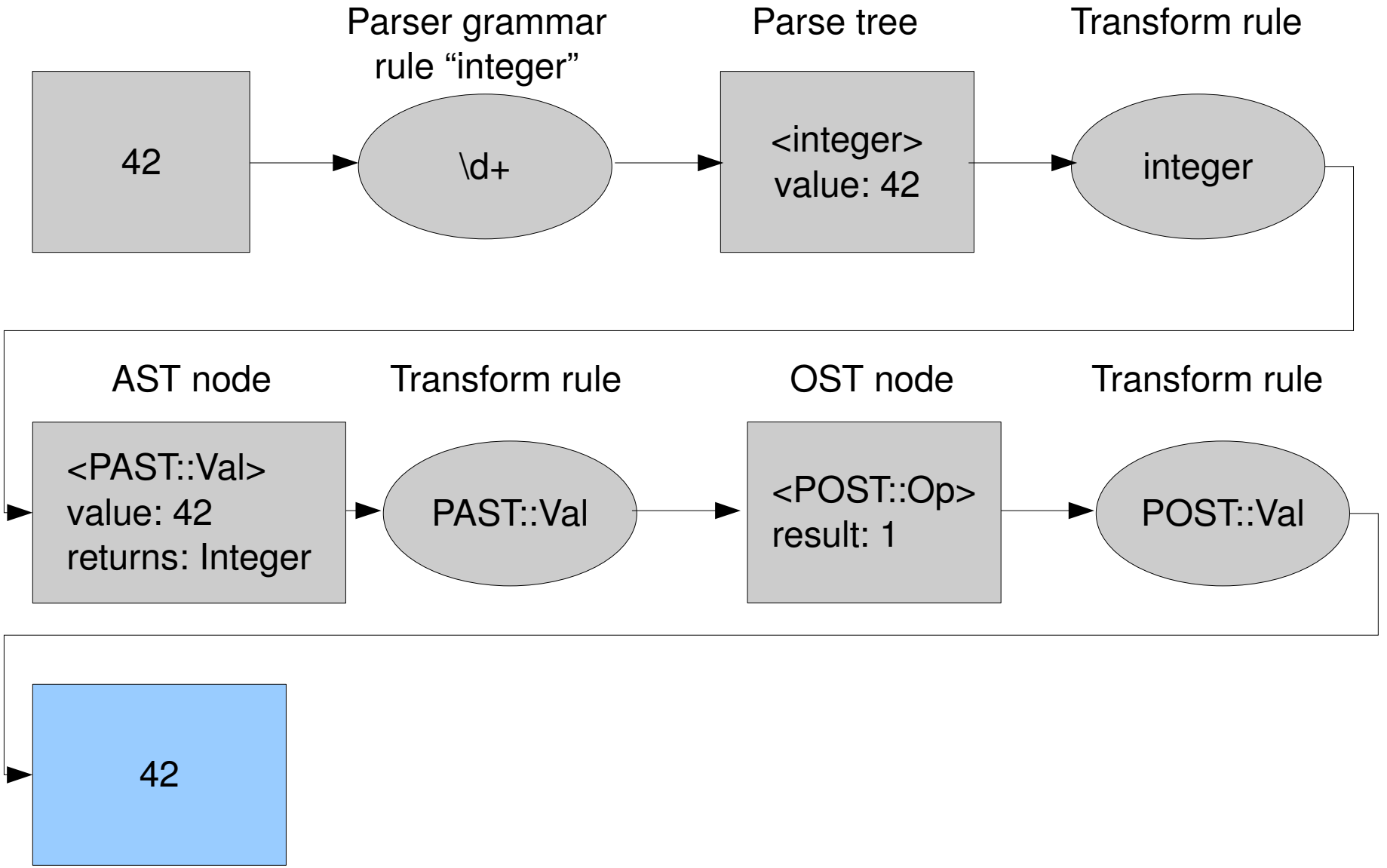
Value Transformation



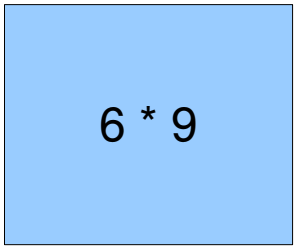
Value Transformation



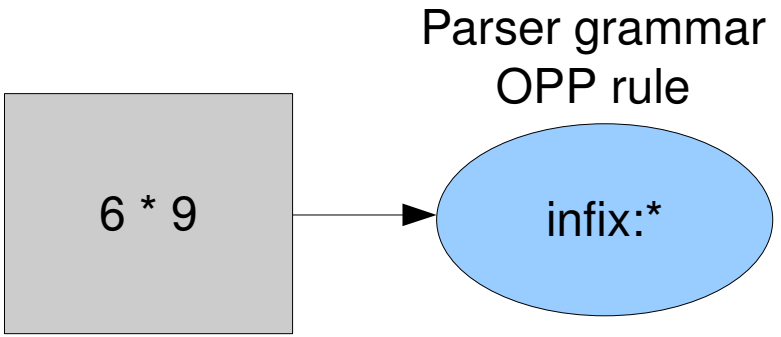
Value Transformation



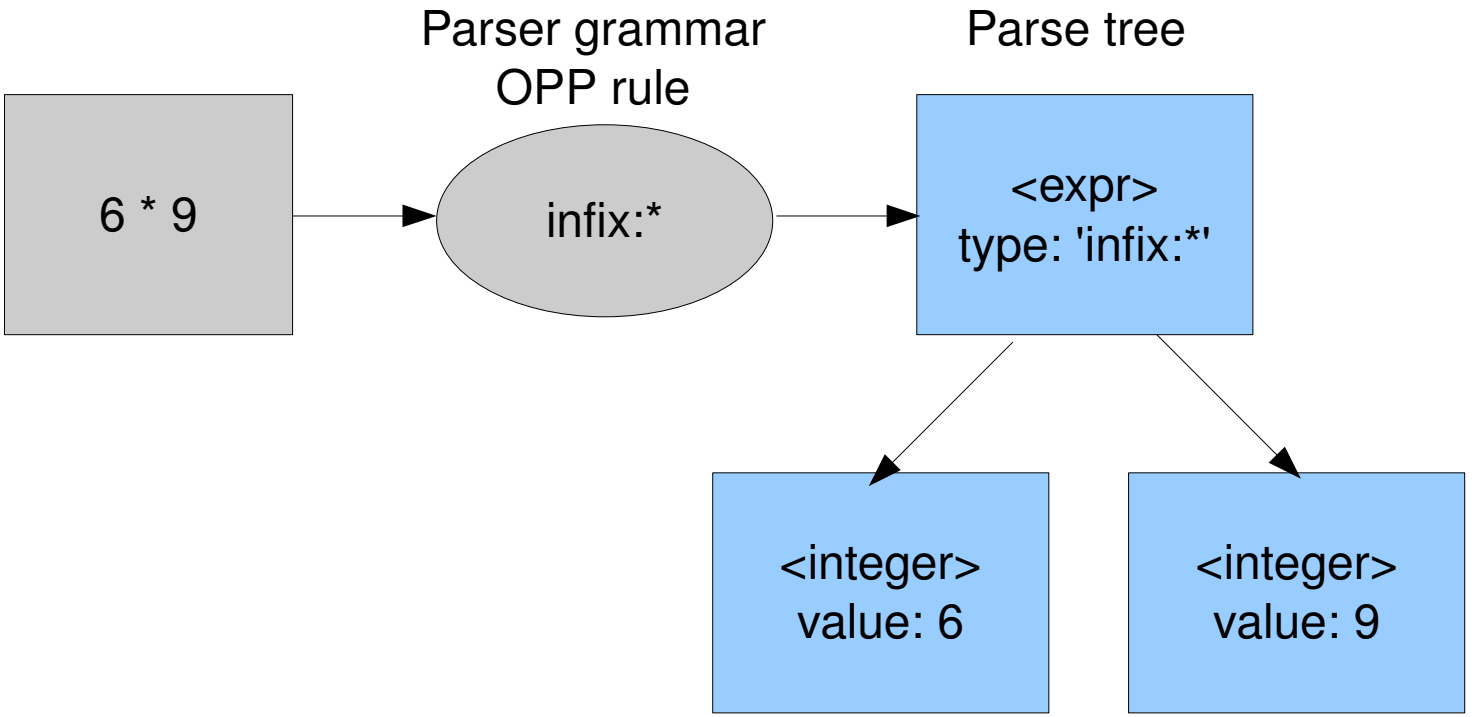
Operator Transformation



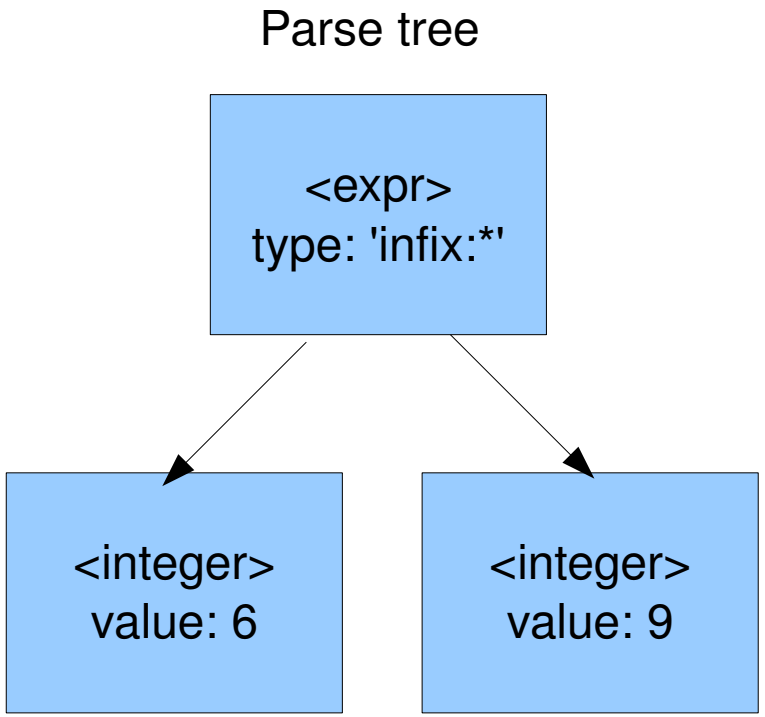
Operator Transformation



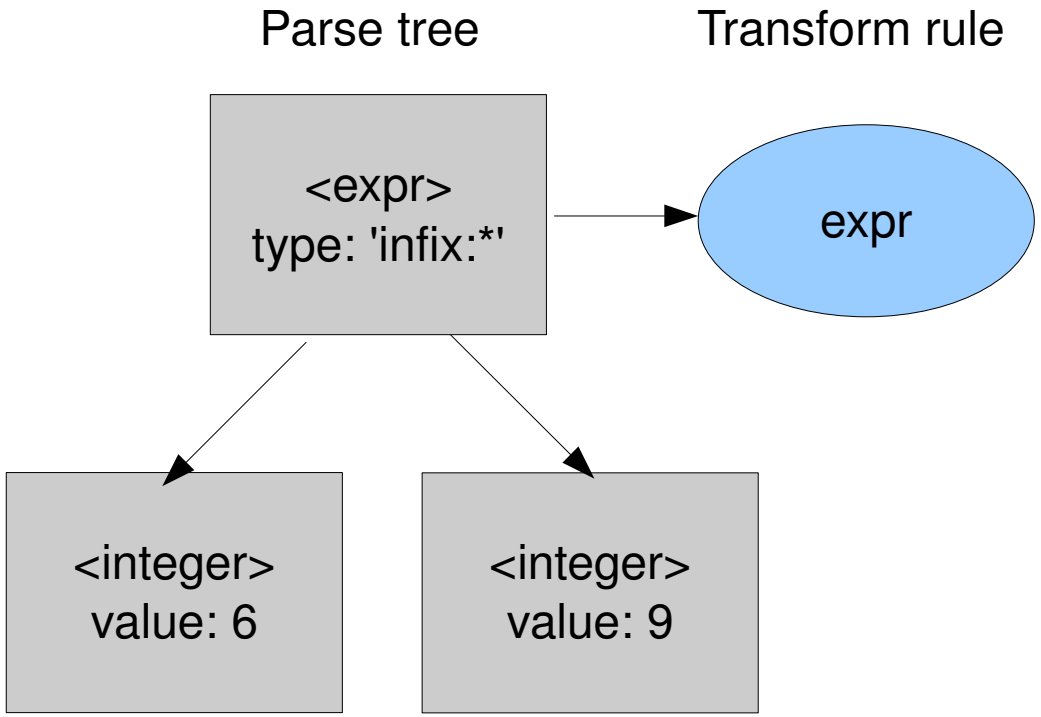
Operator Transformation



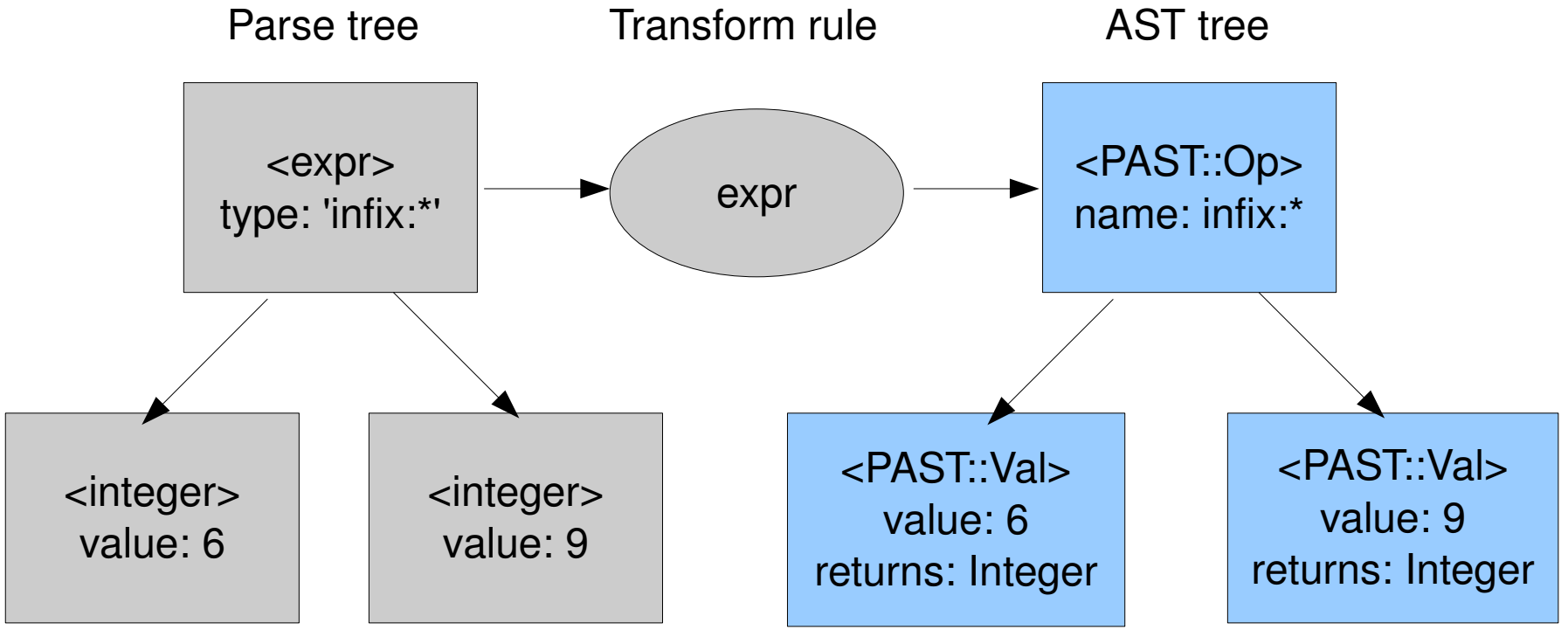
Operator Transformation



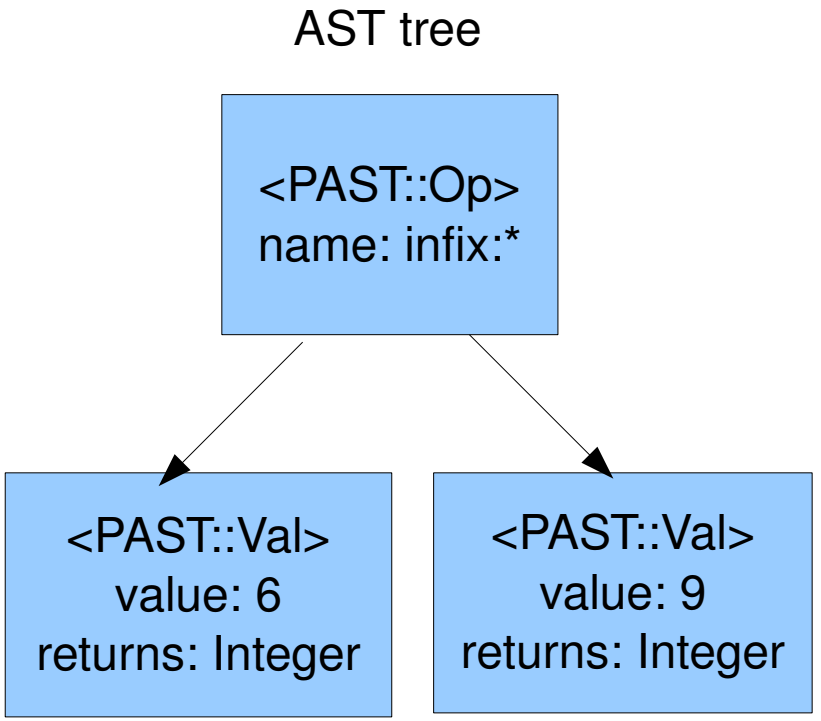
Operator Transformation



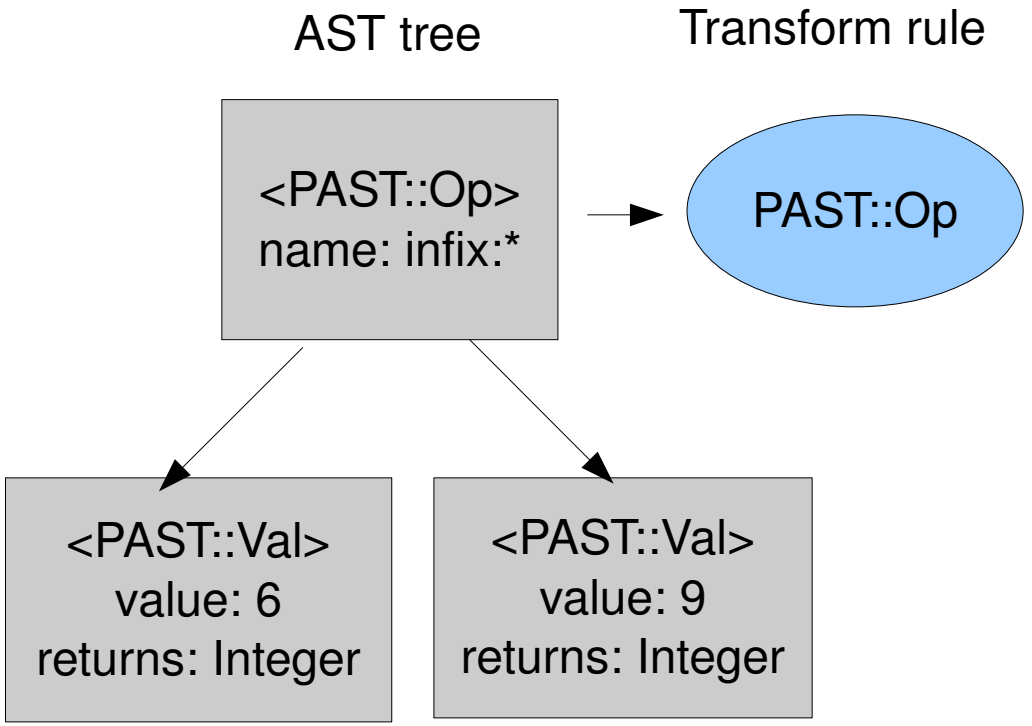
Operator Transformation



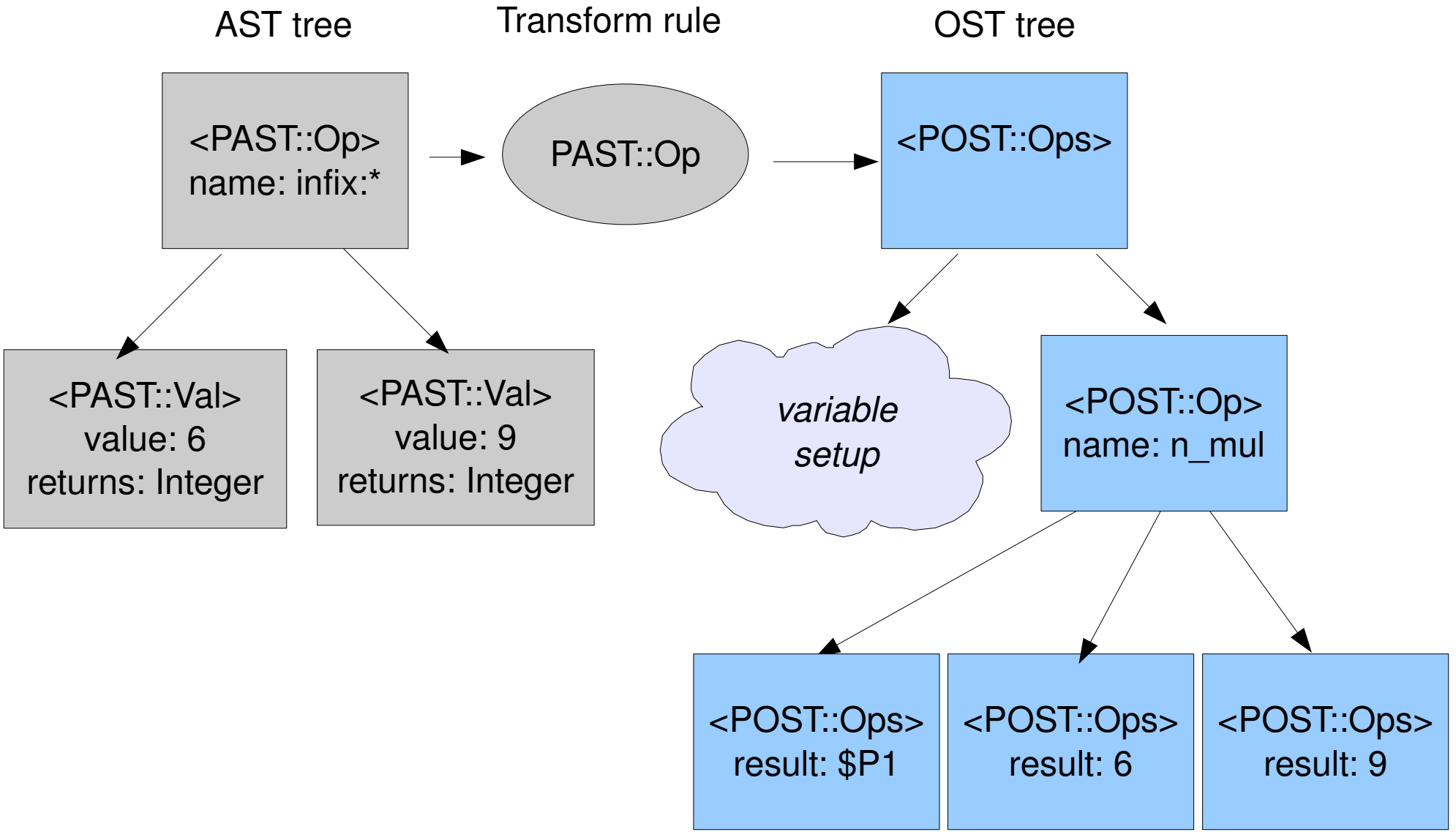
Operator Transformation



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?

- OSCON - 20% discount for user groups. Register with the discount code 'os09usrg'.