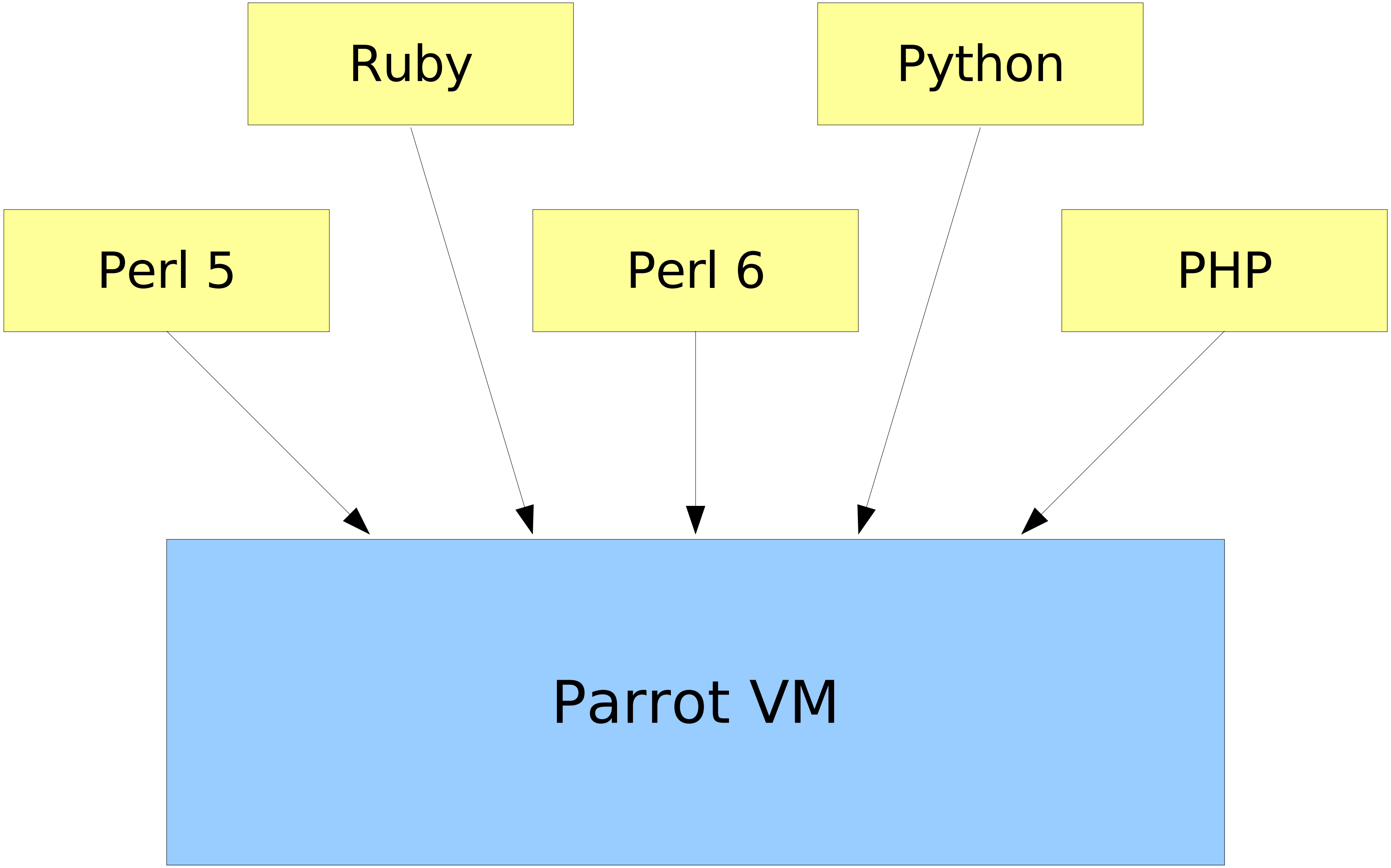
Cardinal:
Ruby on Parrot

Allison Randal
The Parrot Project &
O'Reilly Media, Inc.



Ruby

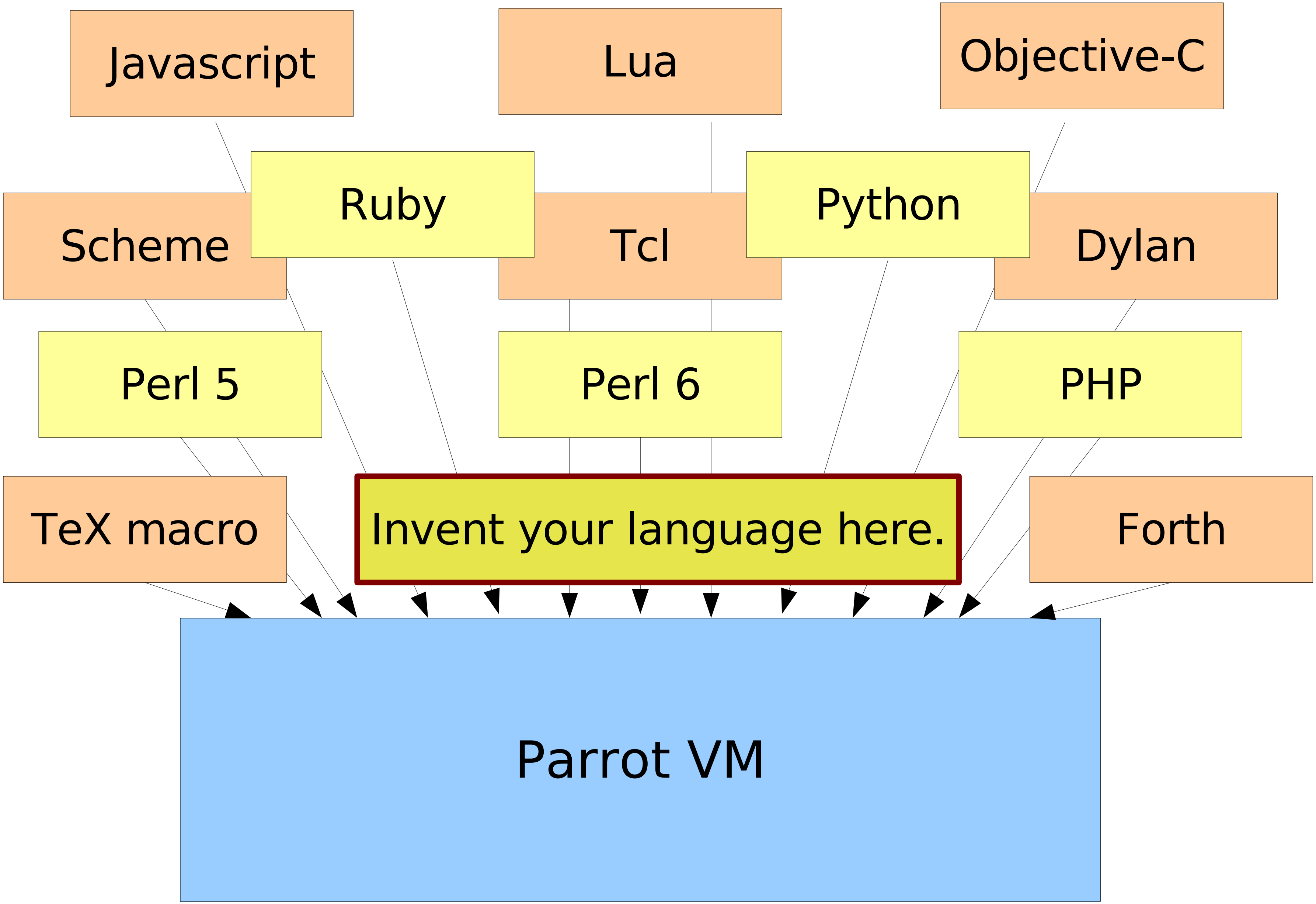
Python

Perl 5

Perl 6

PHP

Parrot VM



Javascript

Lua

Objective-C

Scheme

Ruby

Tcl

Python

Dylan

Perl 5

Perl 6

PHP

TeX macro

Invent your language here.

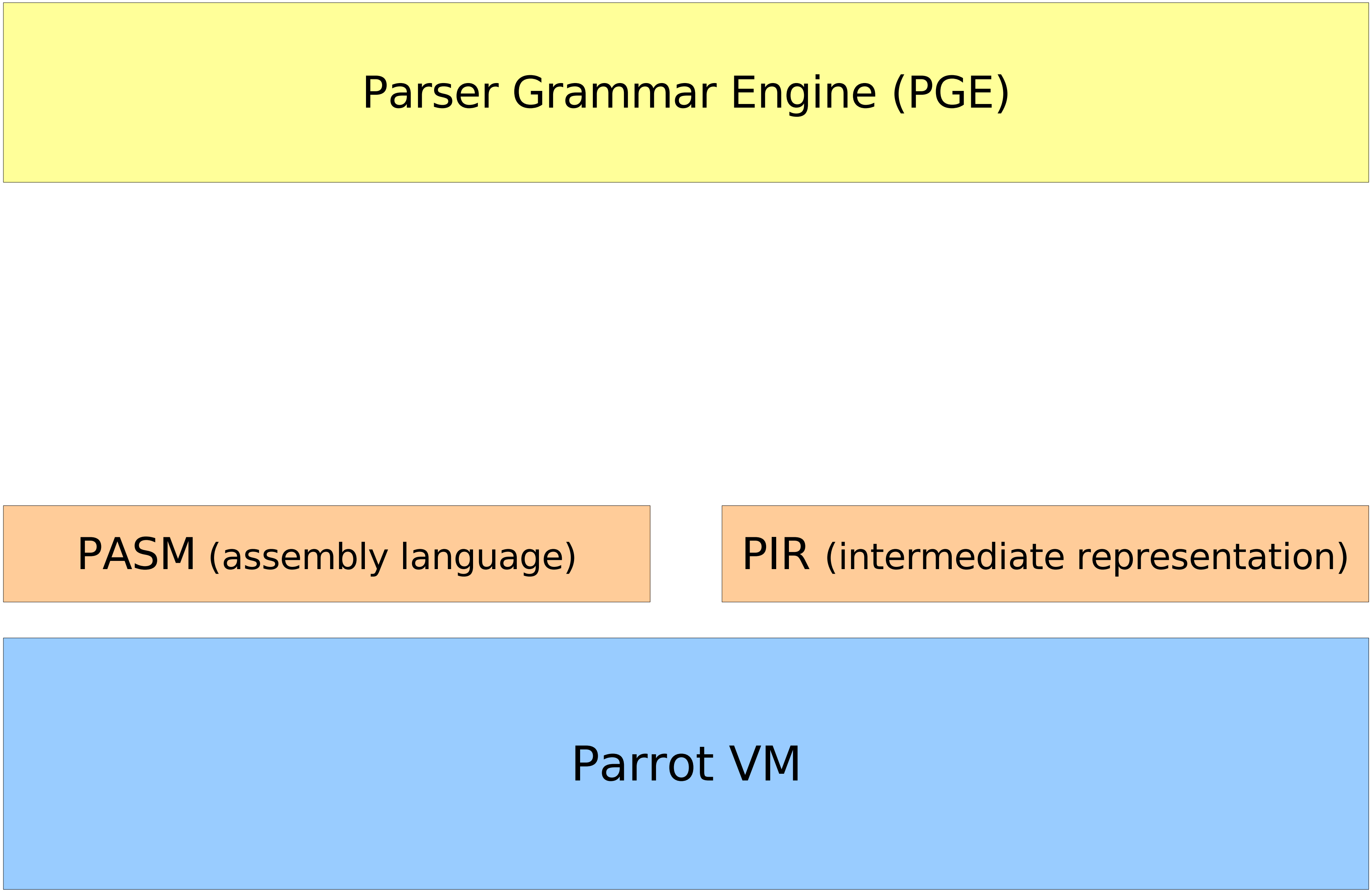
Forth

Parrot VM

Cardinal

On Wed, 2 Jan 2002, Phil Tomson wrote:

> We've had a couple of different threads flowing here the last few days
> about getting Ruby to work with Parrot. Dan Sugalski has suggested that
> we (the Ruby community) need to create a Ruby parser in Ruby that
> initially needs to emit Parrot bytecode and later we could pass on an AST.
>
> So..... I want to propose a new project called Cardinal (the name is open
> to discussion, I chose Cardinal because Parrots are birds, Rubys are red
> and Cardinals are red birds - is the name already taken? I don't see
> any Cardinal project on RAA). The goal of Cardinal is to create a Ruby
> frontend for Parrot.

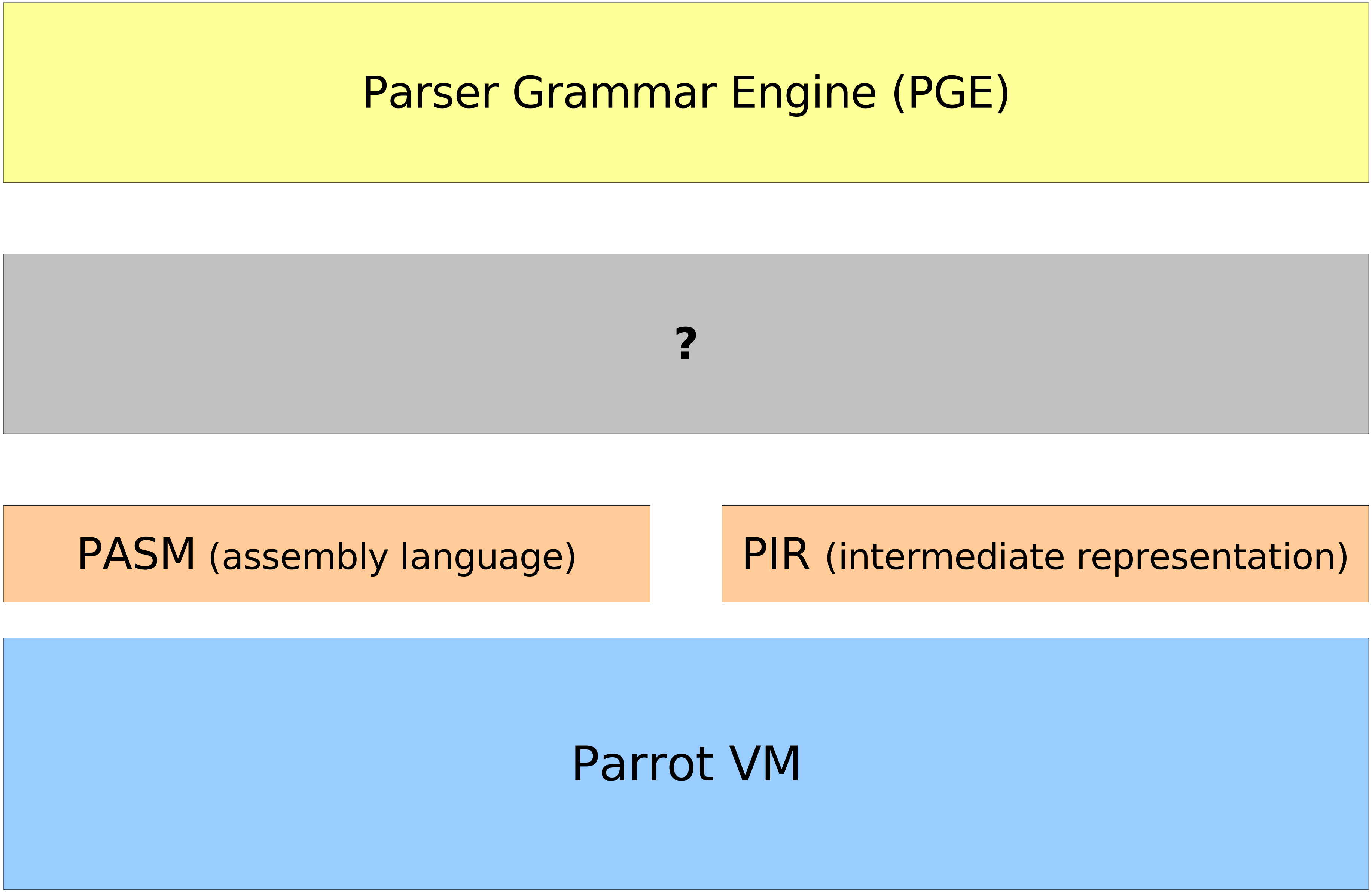


Parser Grammar Engine (PGE)

PASM (assembly language)

PIR (intermediate representation)

Parrot VM



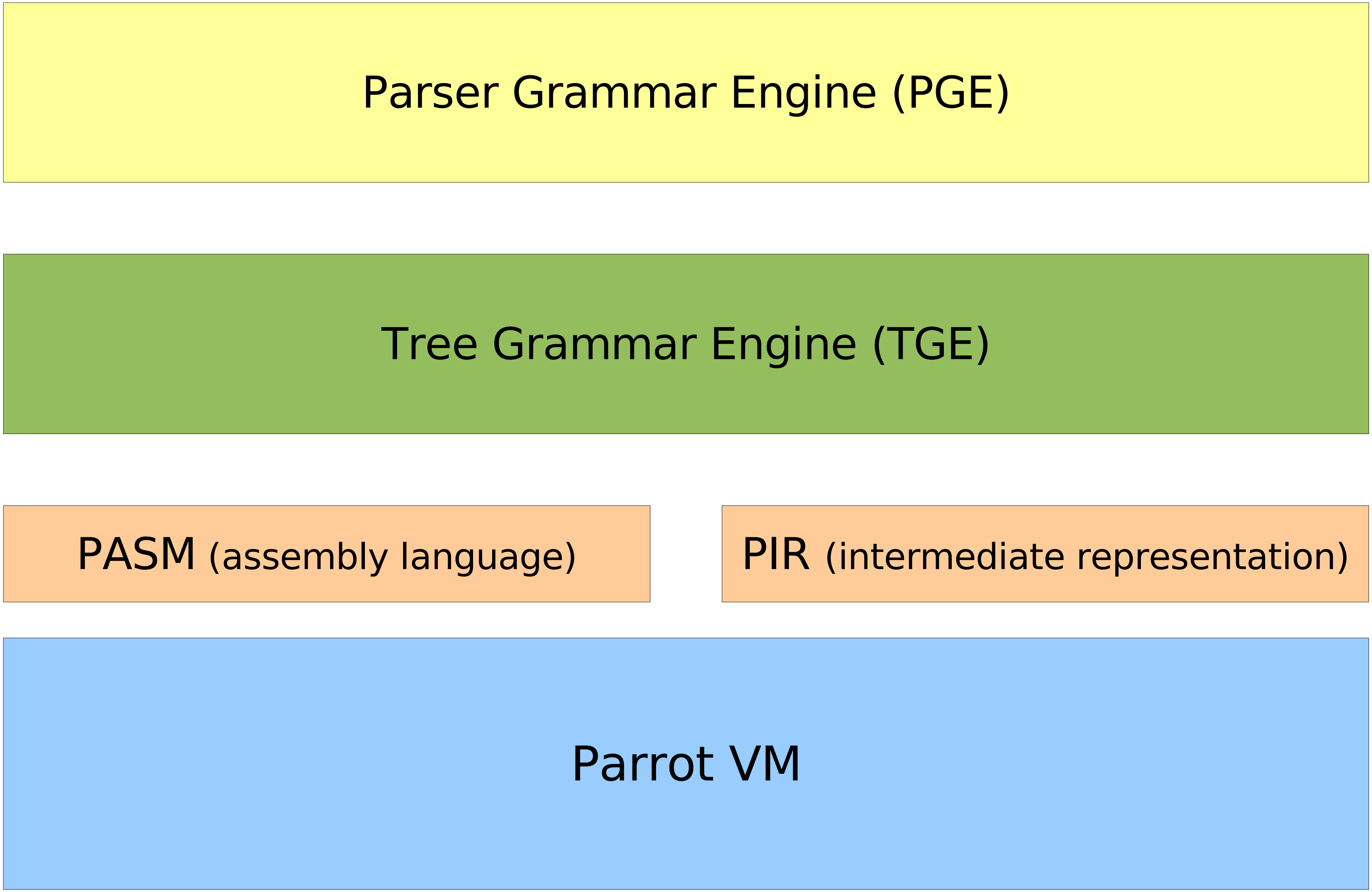
Parser Grammar Engine (PGE)

?

PASM (assembly language)

PIR (intermediate representation)

Parrot VM



Parser Grammar Engine (PGE)

Tree Grammar Engine (TGE)

PASM (assembly language)

PIR (intermediate representation)

Parrot VM

Parser Grammar Engine

- Recursive Descent
- Regular Expressions
- Operator Precedence Parser

Tree Grammar Engine

- Attribute Grammars

(Early February, 1967)

Peter [Wegner] asked me what I thought about formal semantics, and I said I liked [Ned] Iron's idea of synthesizing an overall meaning from submeanings. I also said that I liked the way other people had combined Irons's approach with a top-down or “recursive-descent” parser...

So Peter asked, “Why can't attributes be defined from the top down as well as from the bottom up?”

A shocking idea! Of course I instinctively replied that it was impossible to go both bottom-up and top-down. But after some discussion I realized that his suggestion wasn't so preposterous after all...

- D. E. Knuth, “The genesis of attribute grammars”

Tree Grammar Engine

- Attribute Grammars
- Minimalist Program

Tree Grammar Engine

- Attribute Grammars
- Minimalist Program
- Attract multiple languages

Tree Grammar Engine

- Attribute Grammars
- Minimalist Program
- Attract multiple languages
- Easy to use

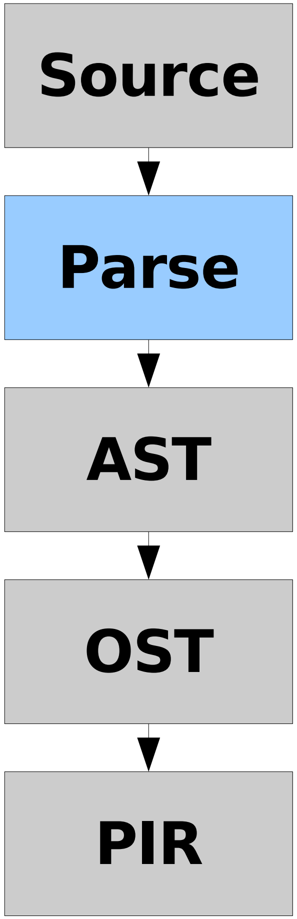
There's an odd misconception in the computing world that writing compilers is hard. This view is fueled by the fact that we don't write compilers very often. People used to think writing CGI code was hard. Well, it is hard, if you do it in C without any tools.

Compiler Tools

- 4 stages

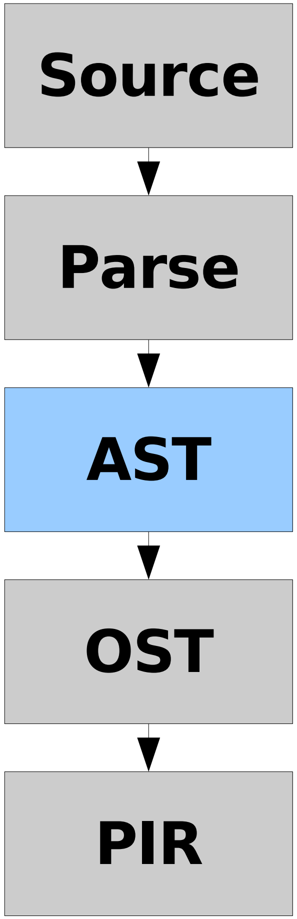
Compiler Tools

- 4 stages
- Parse Tree



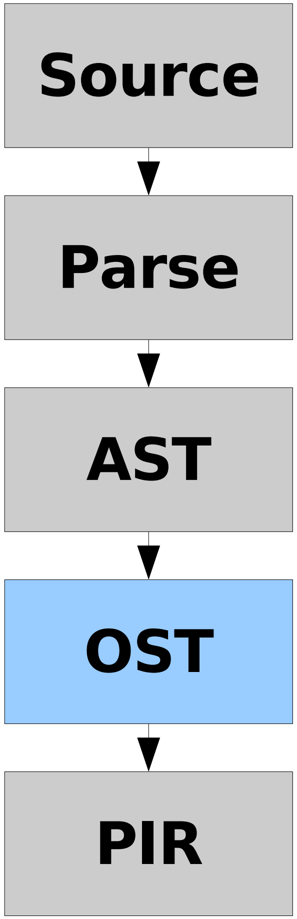
Compiler Tools

- 4 stages
- Parse Tree
- Abstract Syntax Tree



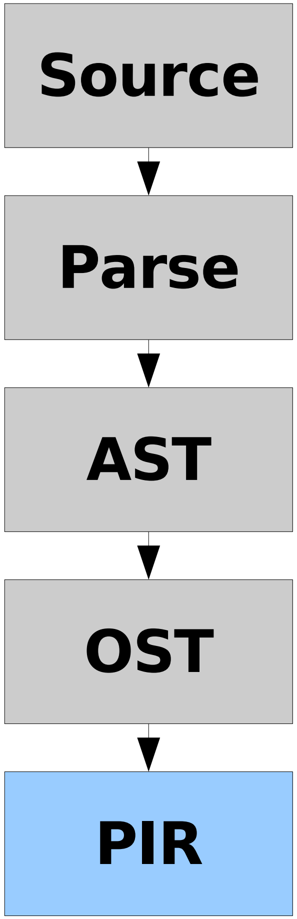
Compiler Tools

- 4 stages
- Parse Tree
- Abstract Syntax Tree
- Opcode Syntax Tree

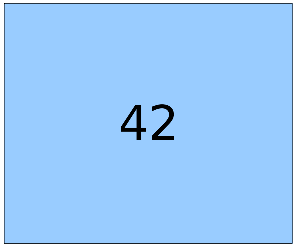


Compiler Tools

- 4 stages
- Parse Tree
- Abstract Syntax Tree
- Opcode Syntax Tree
- PIR (or bytecode)

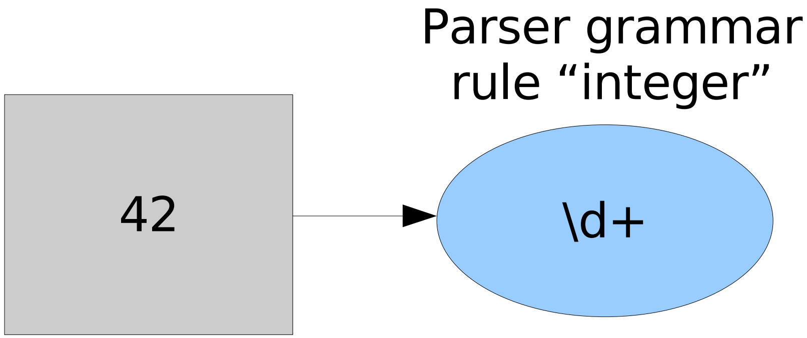


Value Transformation



42

Value Transformation

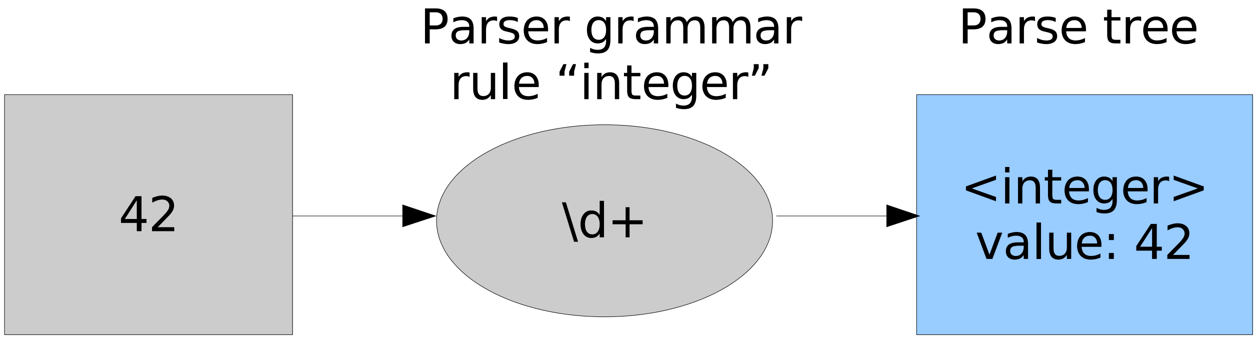


Parser grammar rule “integer”

42

\d+

Value Transformation



42

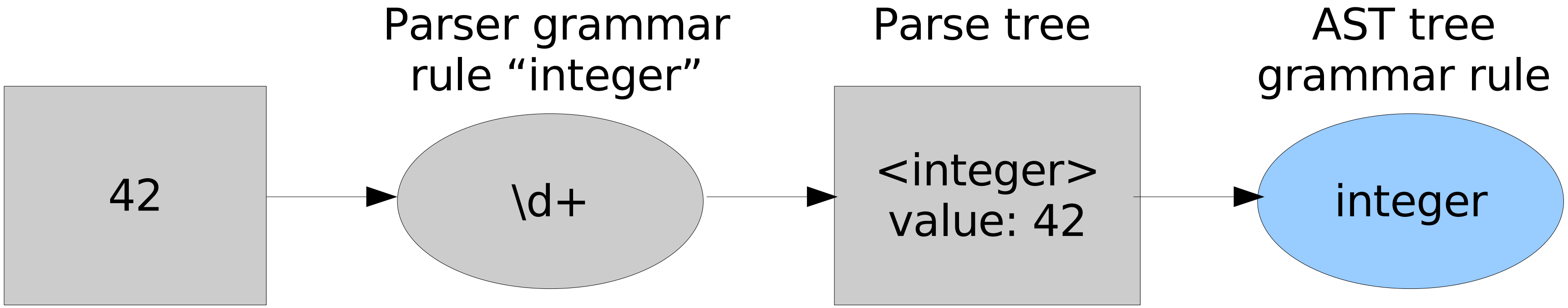
Parser grammar rule “integer”

\d+

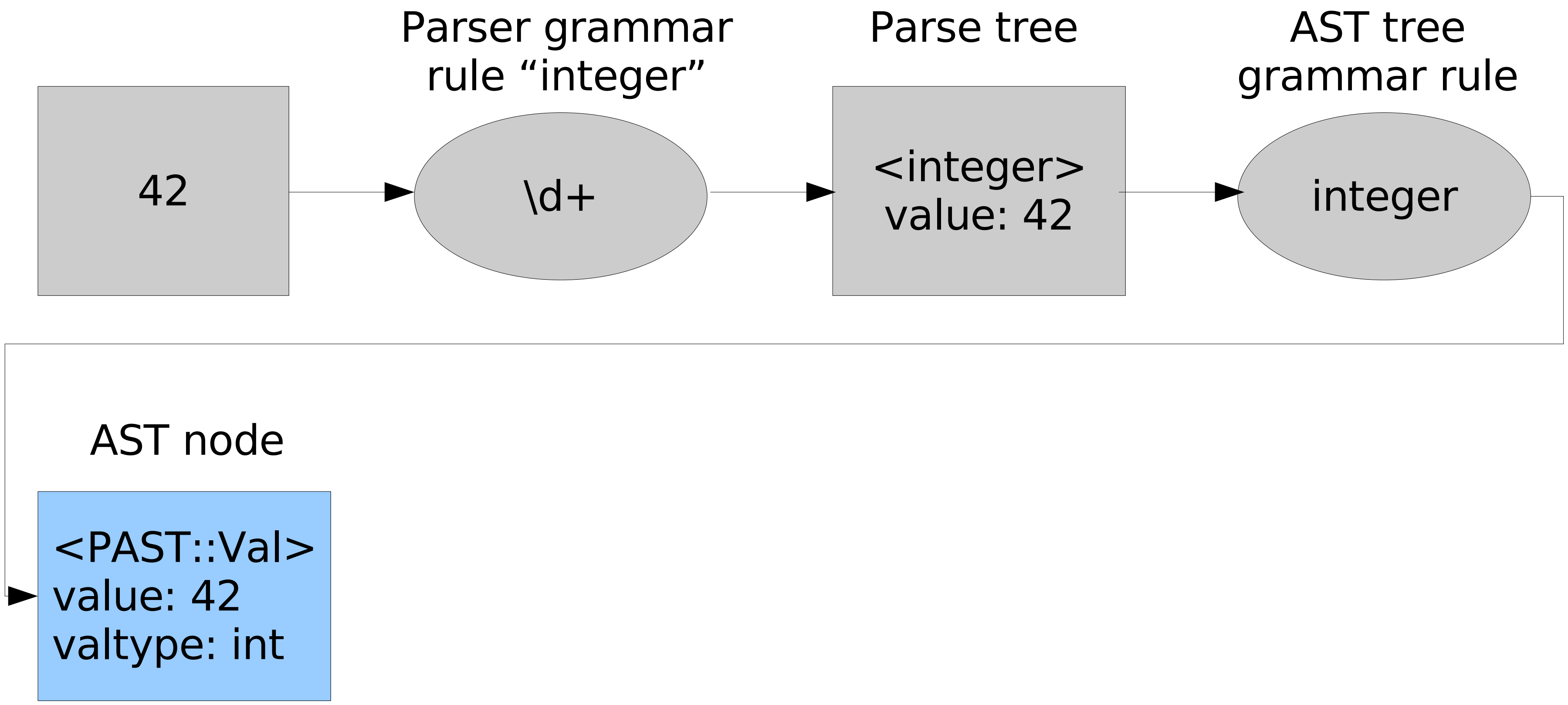
Parse tree

<integer> value: 42

Value Transformation



Value Transformation



Parser grammar rule “integer”

Parse tree

AST tree grammar rule

42

\d+

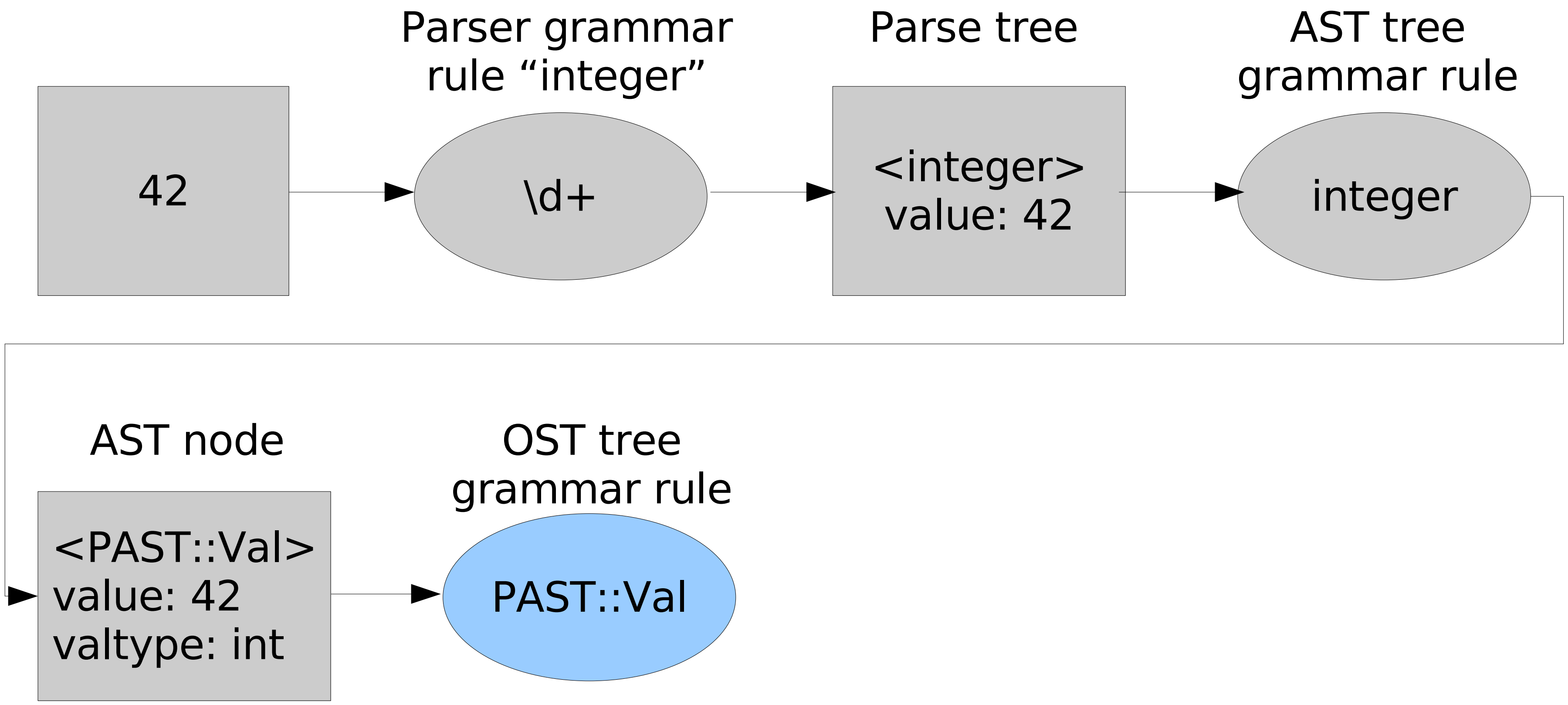
<integer>
value: 42

integer

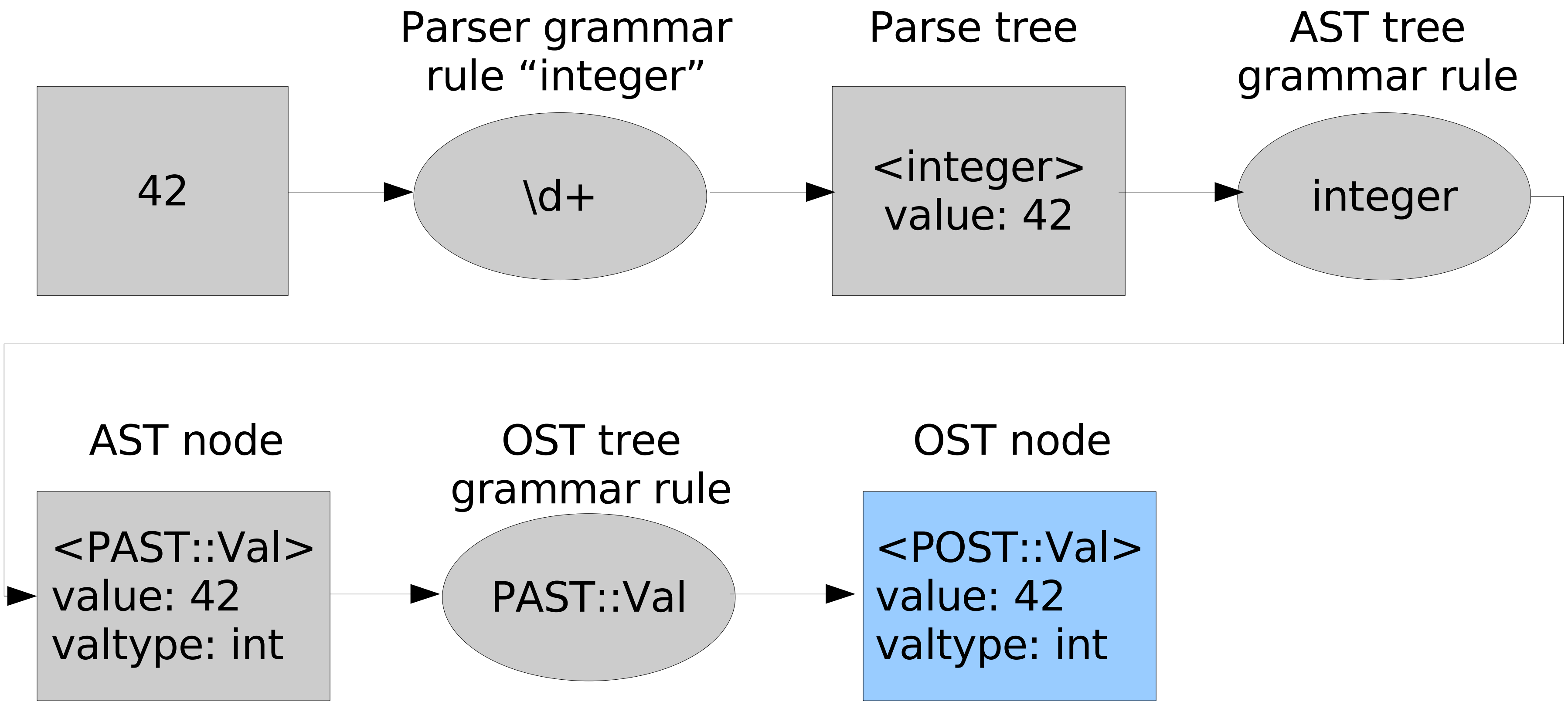
AST node

<PAST::Val>
value: 42
valtype: int

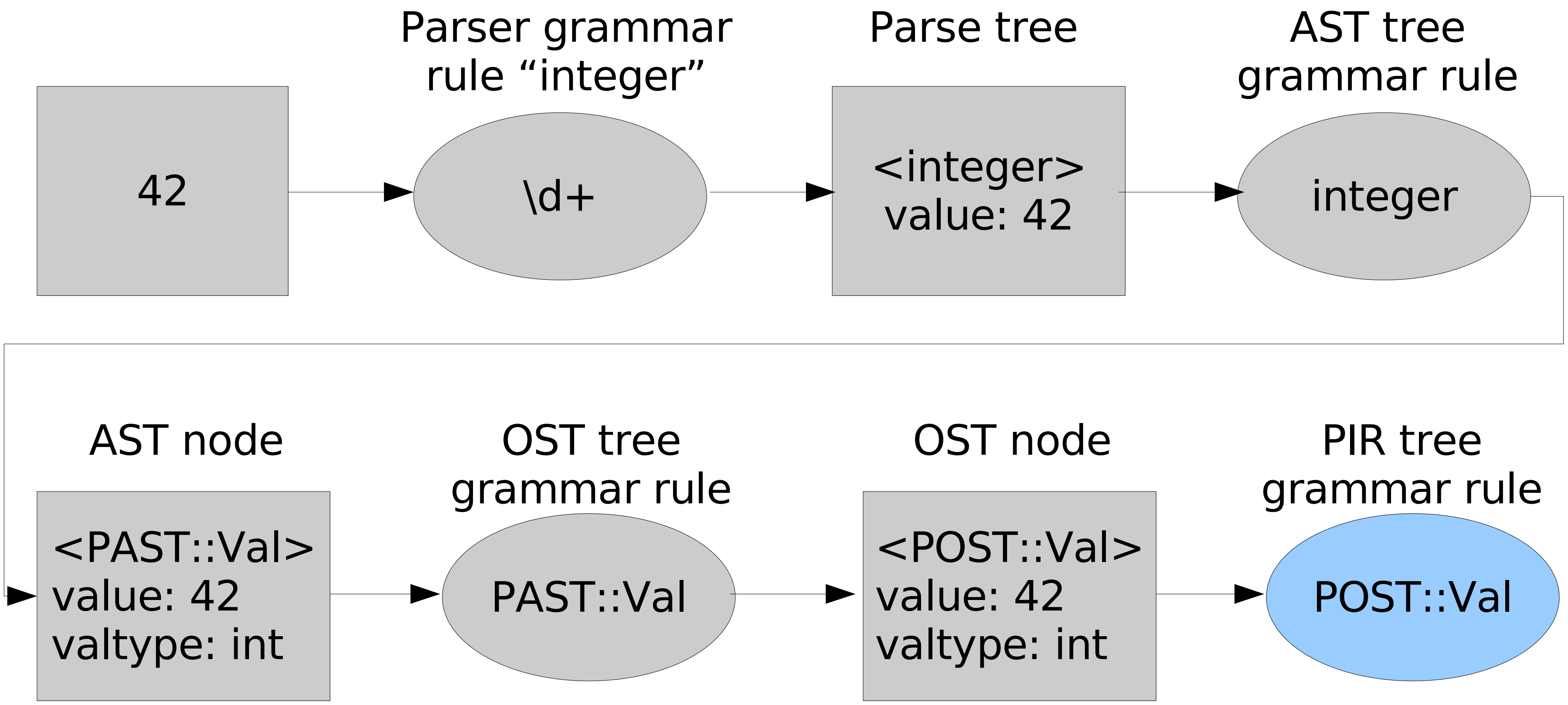
Value Transformation



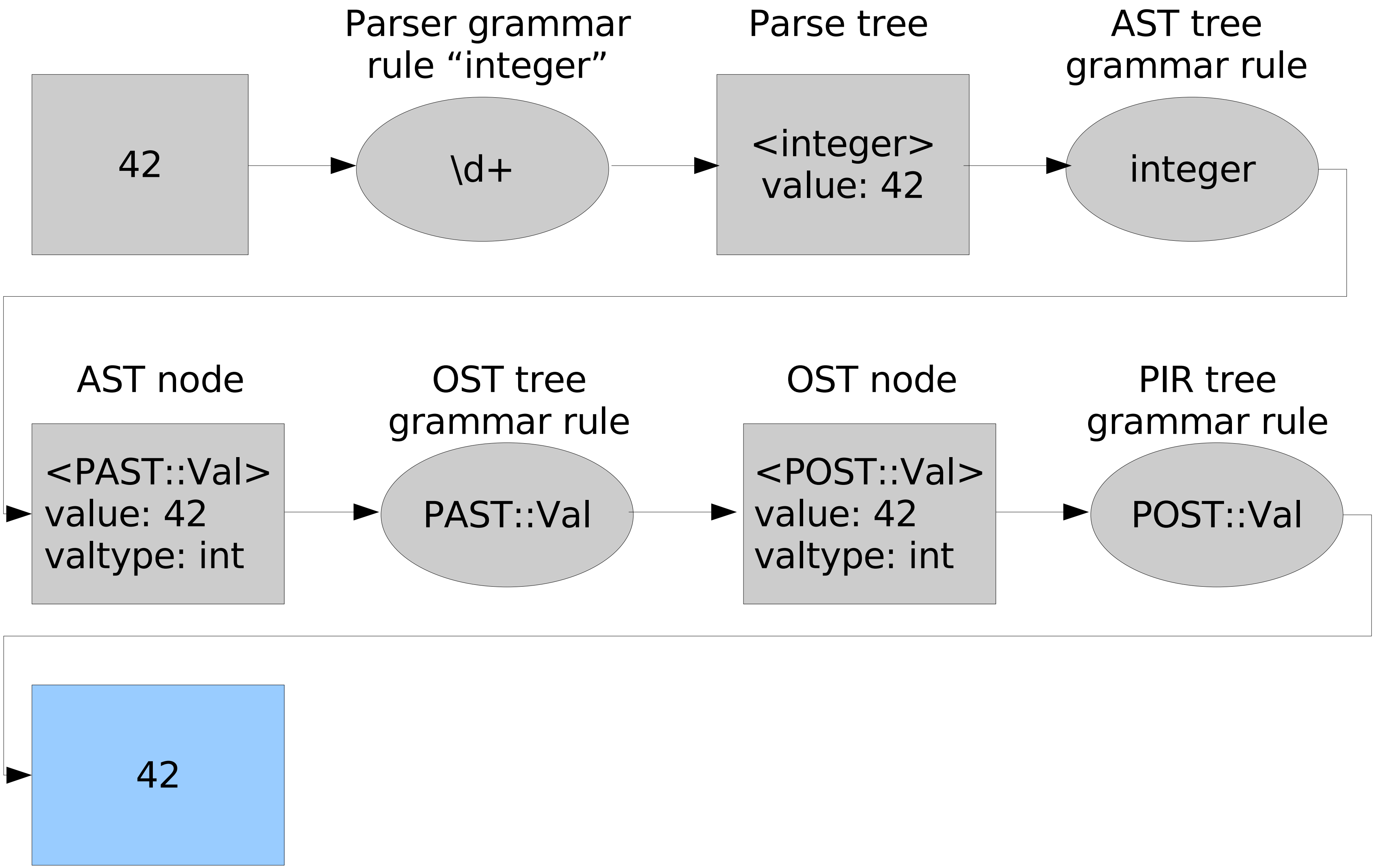
Value Transformation



Value Transformation



Value Transformation



42

Parser grammar rule “integer”

\d+

Parse tree

<integer>
value: 42

AST tree grammar rule

integer

AST node

<PAST::Val>
value: 42
valtype: int

OST tree grammar rule

PAST::Val

OST node

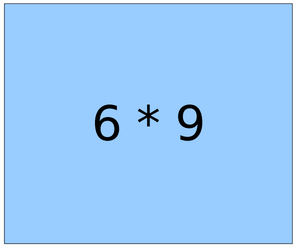
<POST::Val>
value: 42
valtype: int

PIR tree grammar rule

POST::Val

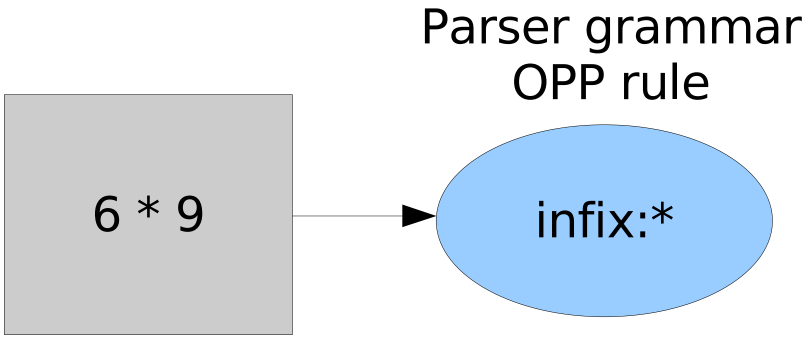
42

Operator Transformation



6 * 9

Operator Transformation

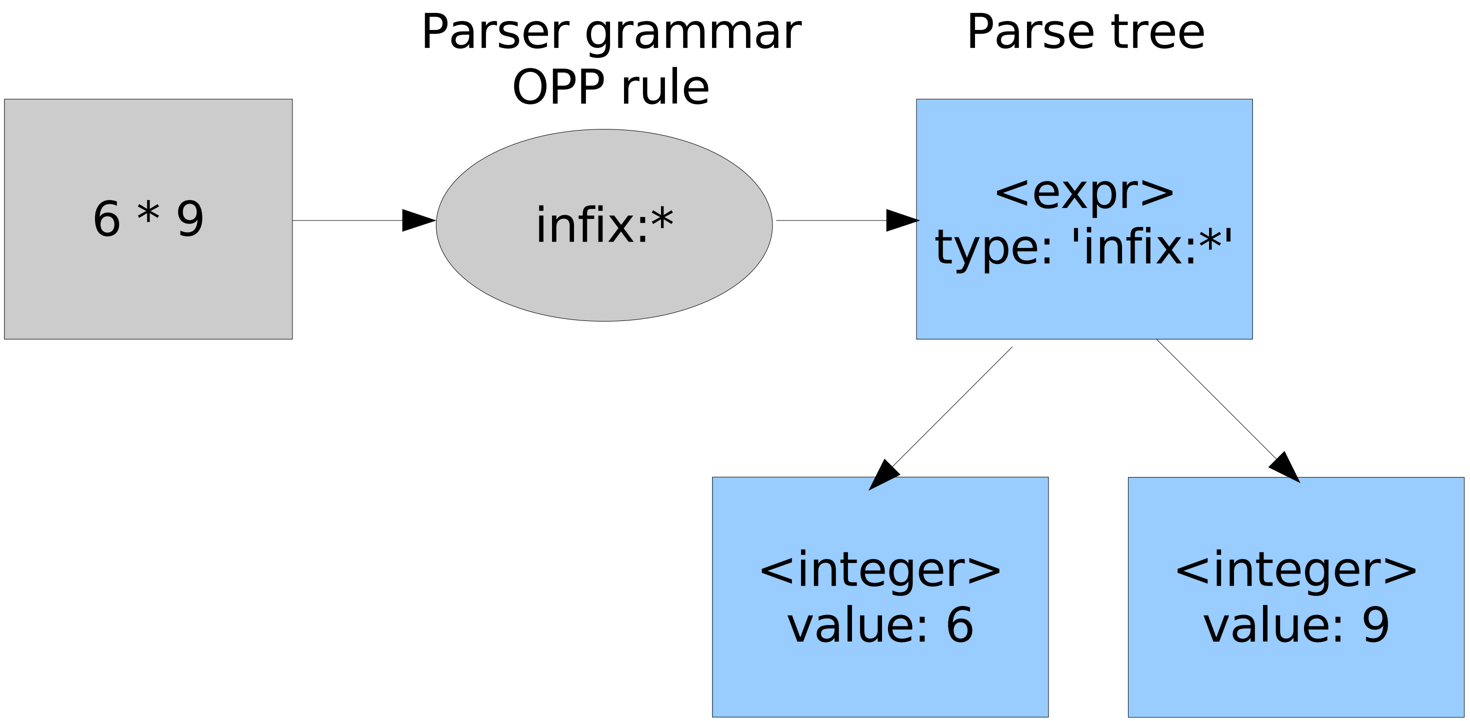


Parser grammar
OPP rule

6 * 9

infix:*

Operator Transformation



Parser grammar
OPP rule

Parse tree

6 * 9

infix:*

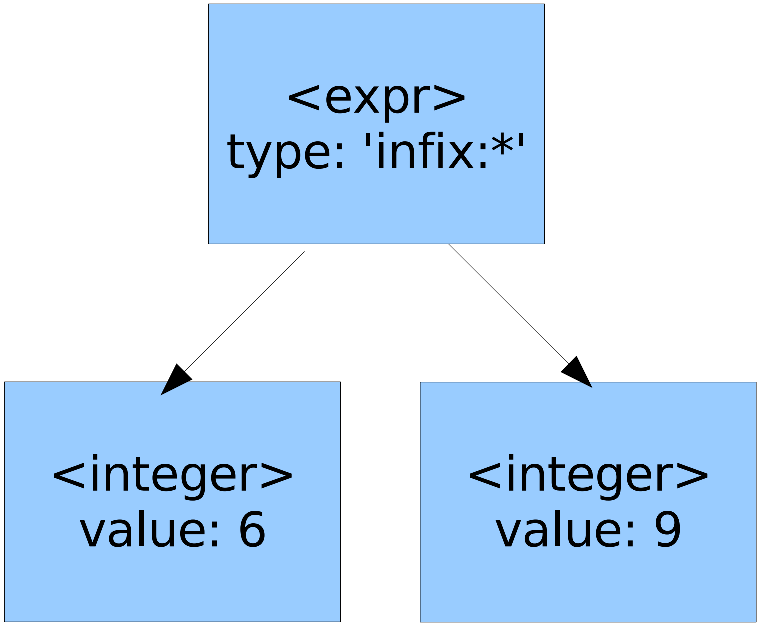
<expr>
type: 'infix:*'

<integer>
value: 6

<integer>
value: 9

Operator Transformation

Parse tree

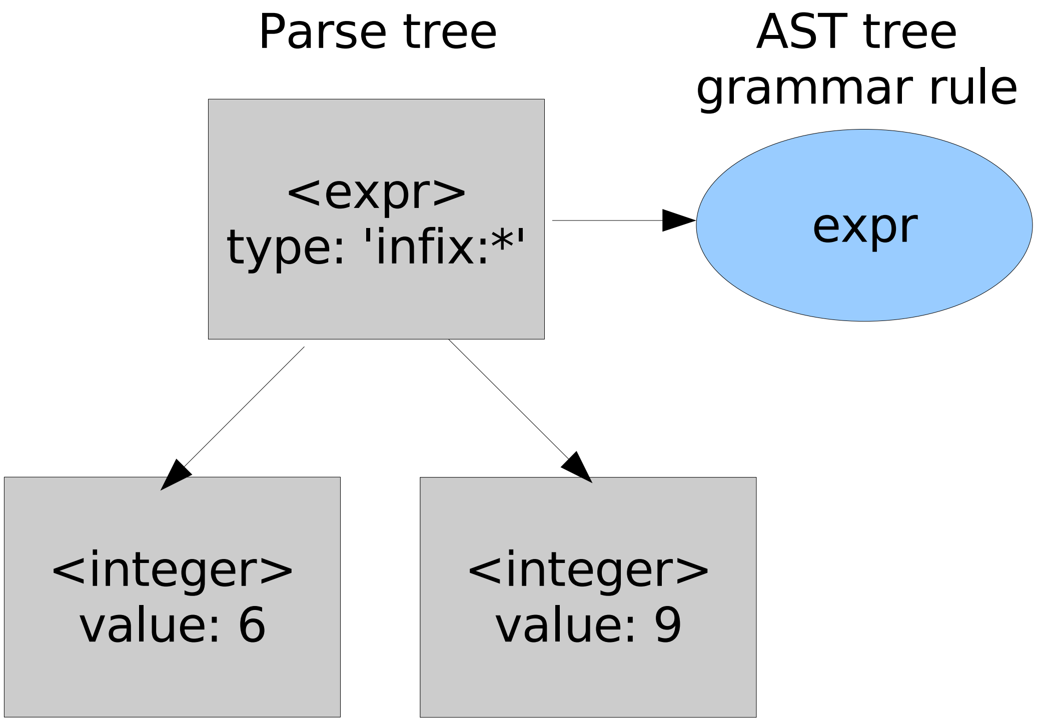


<expr>
type: 'infix:*'

<integer>
value: 6

<integer>
value: 9

Operator Transformation



Parse tree

AST tree
grammar rule

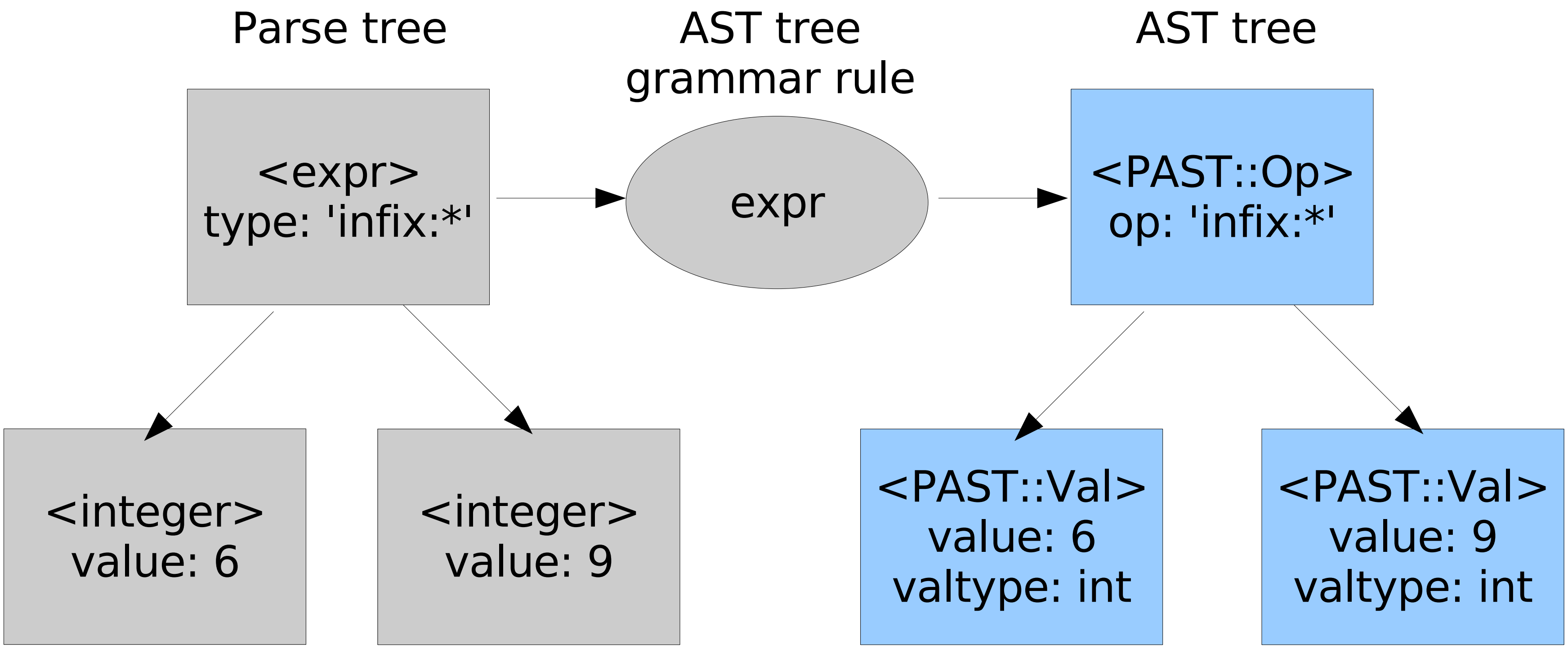
<expr>
type: 'infix:*'

expr

<integer>
value: 6

<integer>
value: 9

Operator Transformation



Parse tree

<expr>
type: 'infix:*'

<integer>
value: 6

<integer>
value: 9

AST tree grammar rule

expr

AST tree

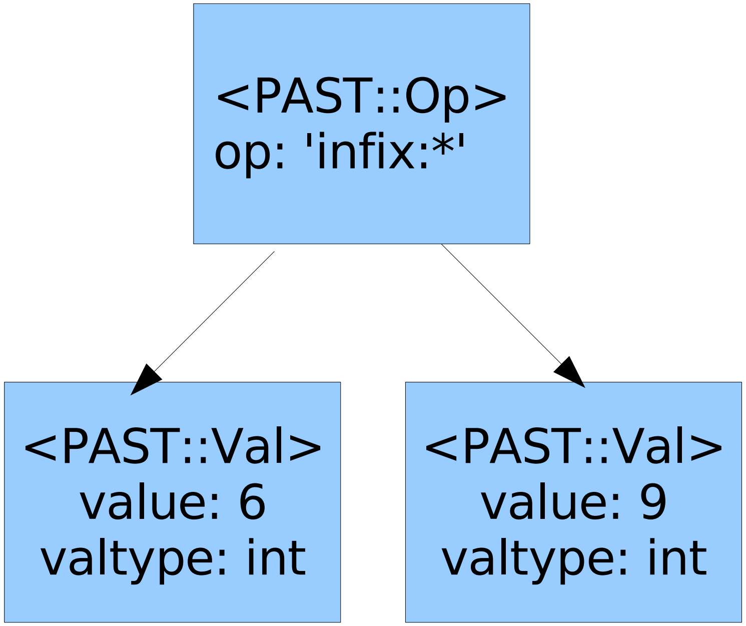
<PAST::Op>
op: 'infix:*'

<PAST::Val>
value: 6
valtype: int

<PAST::Val>
value: 9
valtype: int

Operator Transformation

AST tree

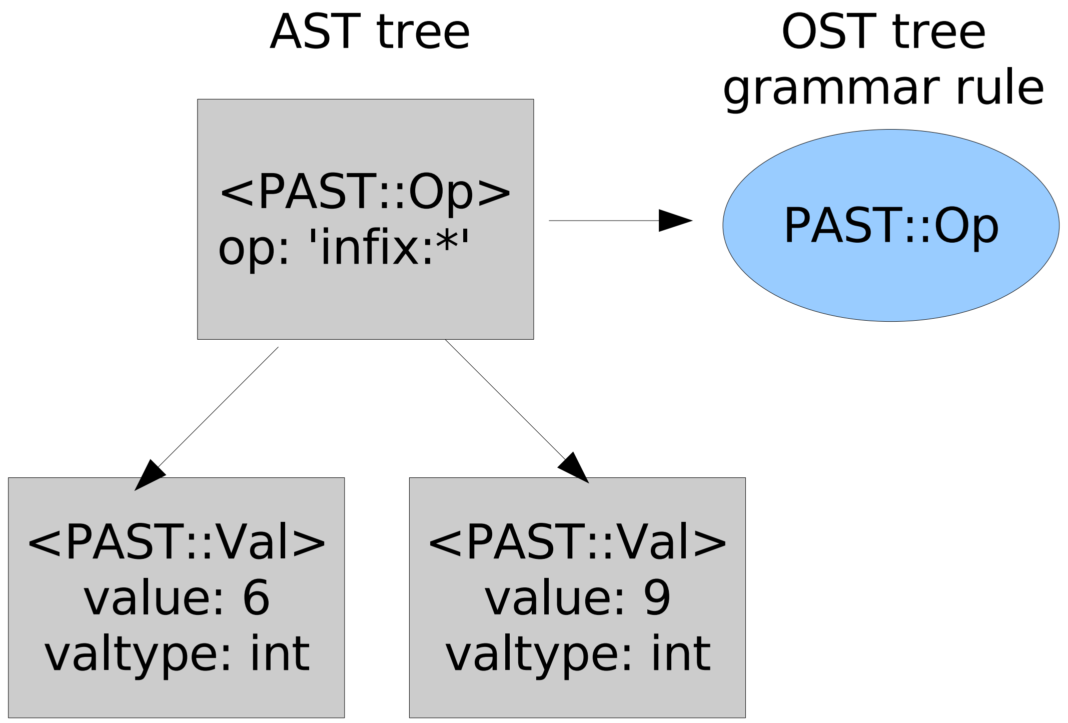


<PAST::Op>
op: 'infix:*'

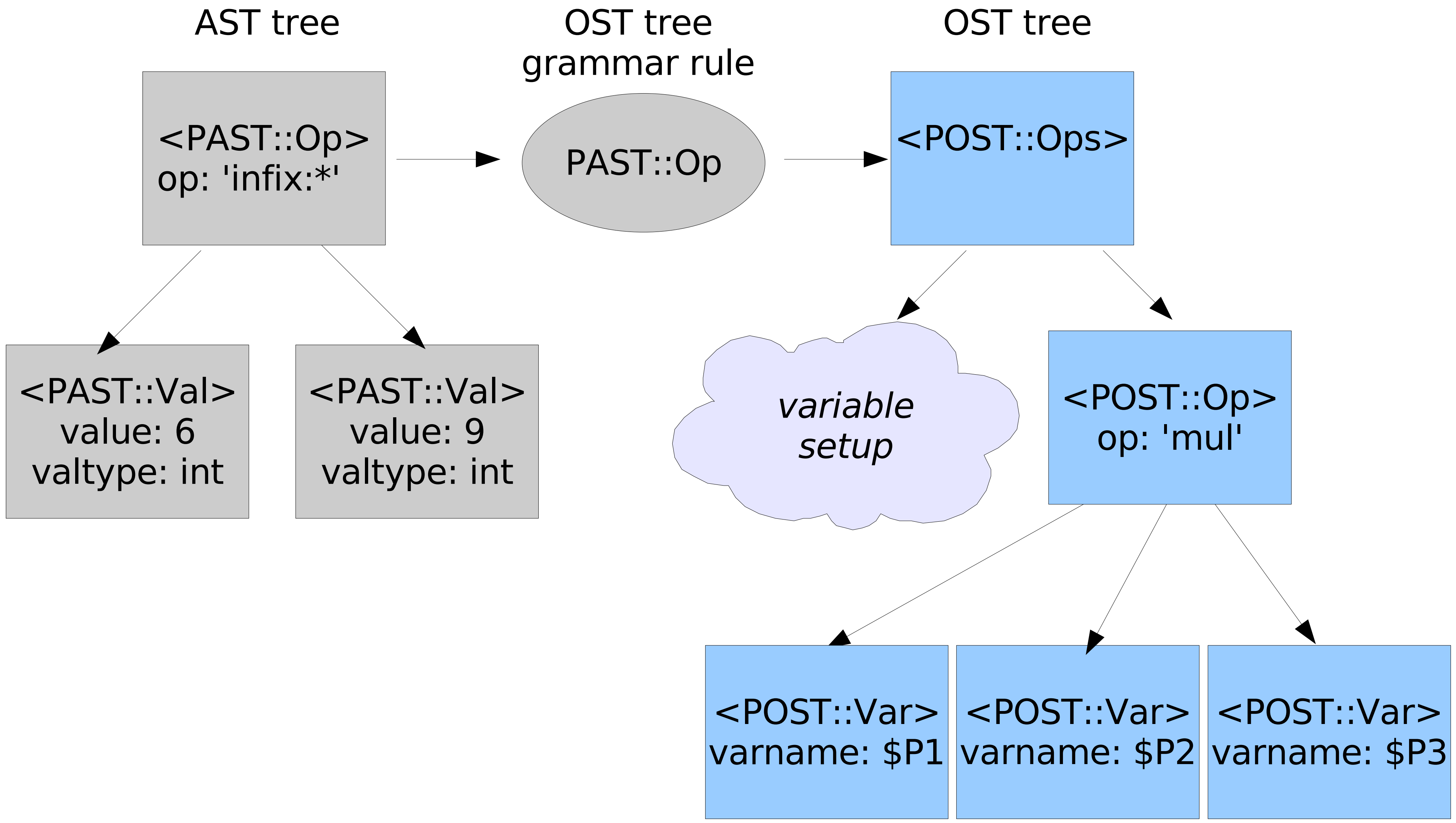
<PAST::Val>
value: 6
valtype: int

<PAST::Val>
value: 9
valtype: int

Operator Transformation



Operator Transformation



AST tree

<PAST::Op>
op: 'infix:*'

<PAST::Val>
value: 6
valtype: int

<PAST::Val>
value: 9
valtype: int

OST tree
grammar rule

PAST::Op

OST tree

<POST::Ops>

variable setup

<POST::Op>
op: 'mul'

<POST::Var>
varname: $P1

<POST::Var>
varname: $P2

<POST::Var>
varname: $P3

Operator Transformation

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

Tree Grammar Engine

- Simple steps
- Elegant
- Hide Complexity
- Impossible

Questions?

- Further Reading
  - *http://parrotcode.org/docs/compiler_tools.html*
  - Knuth, D. E. (1990) “The genesis of attribute grammars.” *Proceedings of the international conference on Attribute grammars and their applications*, 1–12.
  - Chomsky, Noam (1995). The Minimalist Program. MIT Press.