Eco: A language composition editor


Edd Barrett


Carl Friedrich Bolz


Lukas Diekmann


Laurence Tratt


Naveneetha Krishnan Vasudevan

KING'S College LONDON

Software Development Team

2014-05-19

Our problem

Our problem

We want **better** programming languages

Our problem

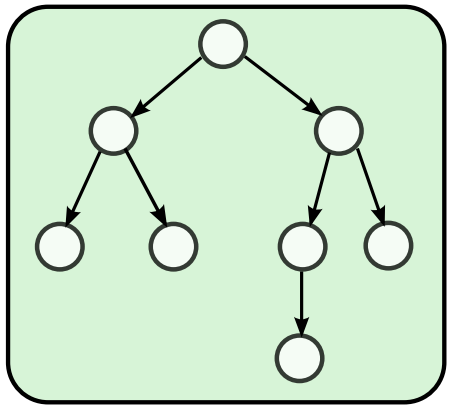
We want **better** programming languages

But better always seems to end up **bigger**

Language composition

Underlying language composition challenges

Parsing

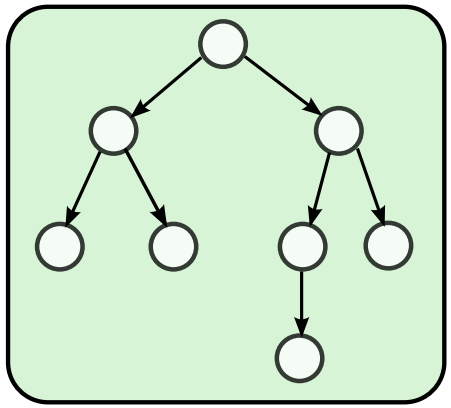


Running

```
SUB     AX,AX
MOV     ES,AX
SUB     BH,BH
MOV     BL,INT_NUMBER
SHL     BX,1
SHL     BX,1
MOV     DI,ES:[BX]
MOV     ES,ES:[BX+2]
ADD     DI,4
LEA     SI,TAG
MOV     CX,TAG_LEN
```

Underlying language composition challenges

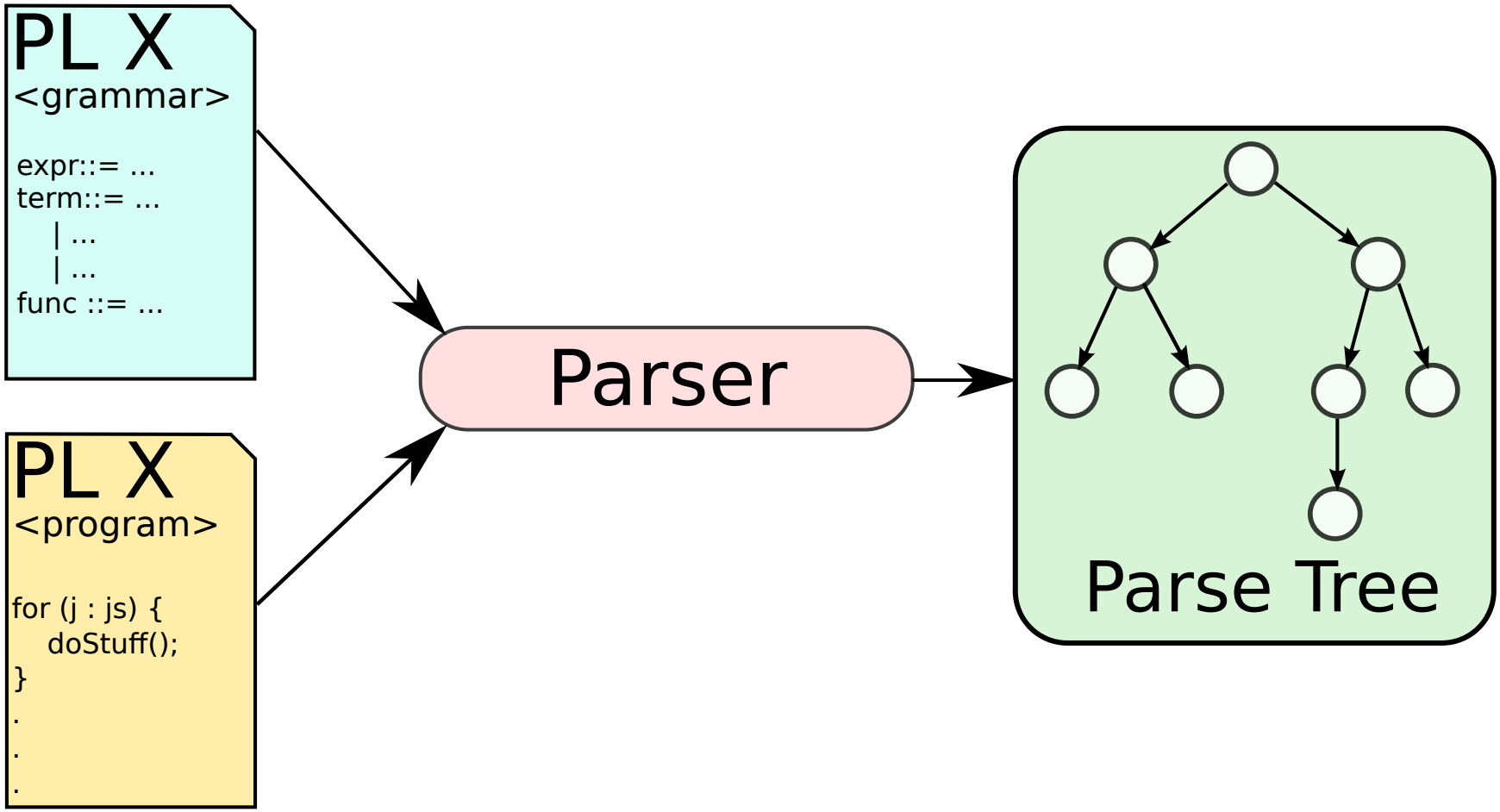
Parsing



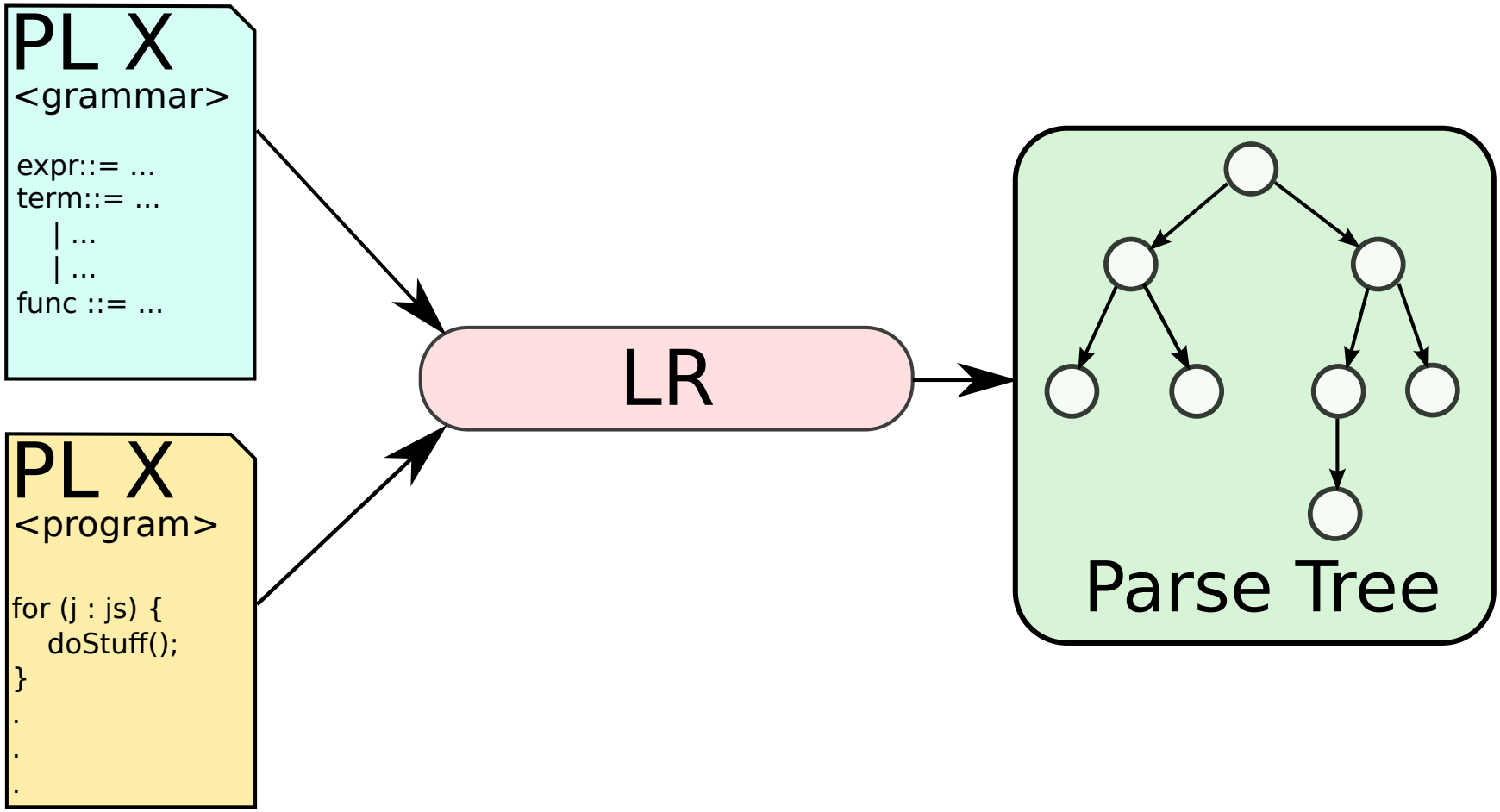
Running

```
SUB     AX,AX
MOV     ES,AX
SUB     BH,BH
MOV     BL,INT_NUMBER
SHL     BX,1
SHL     BX,1
MOV     DI,ES:[BX]
MOV     ES,ES:[BX+2]
ADD     DI,4
LEA     SI,TAG
MOV     CX,TAG_LEN
```

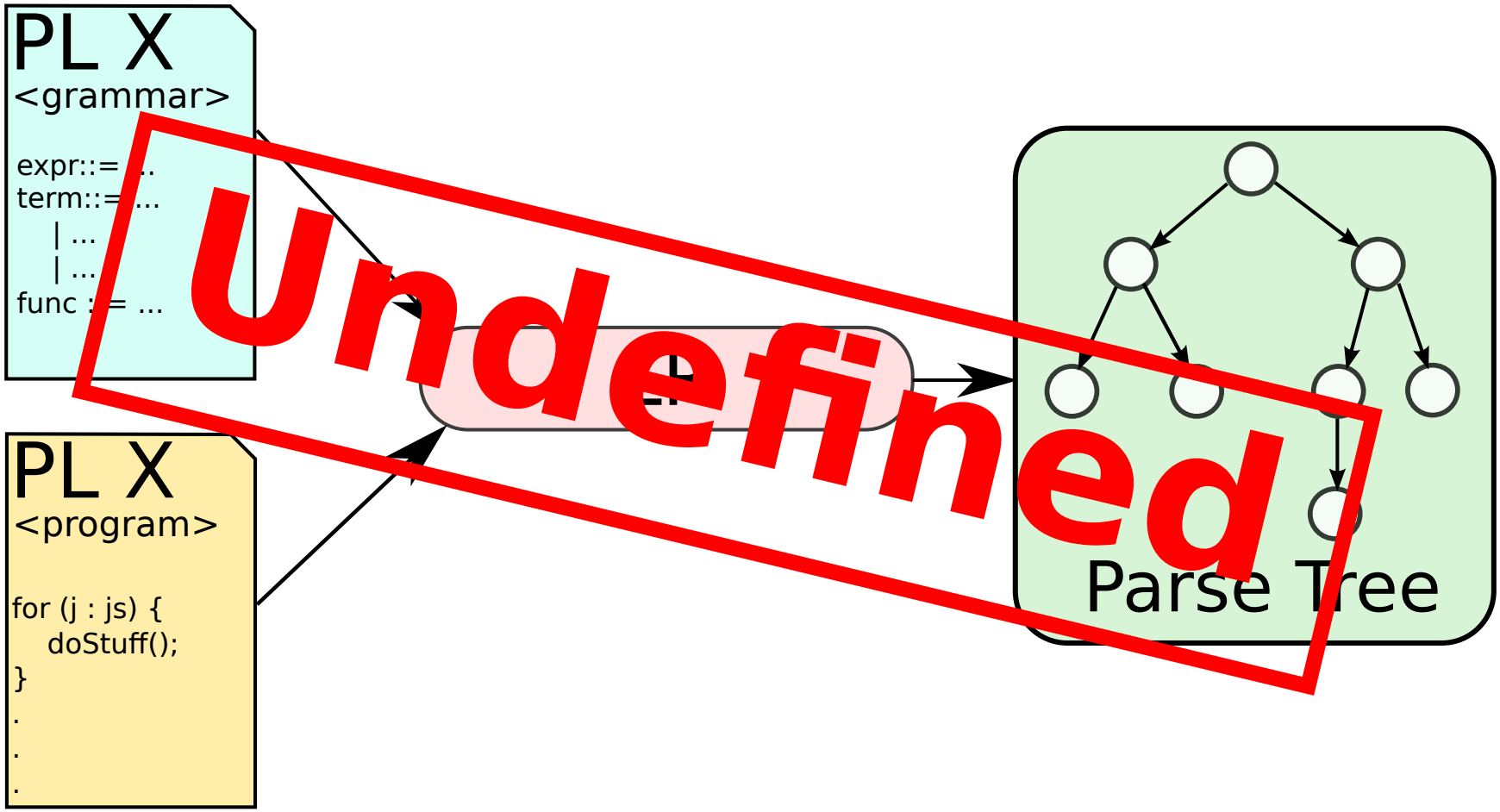
Parsing composition



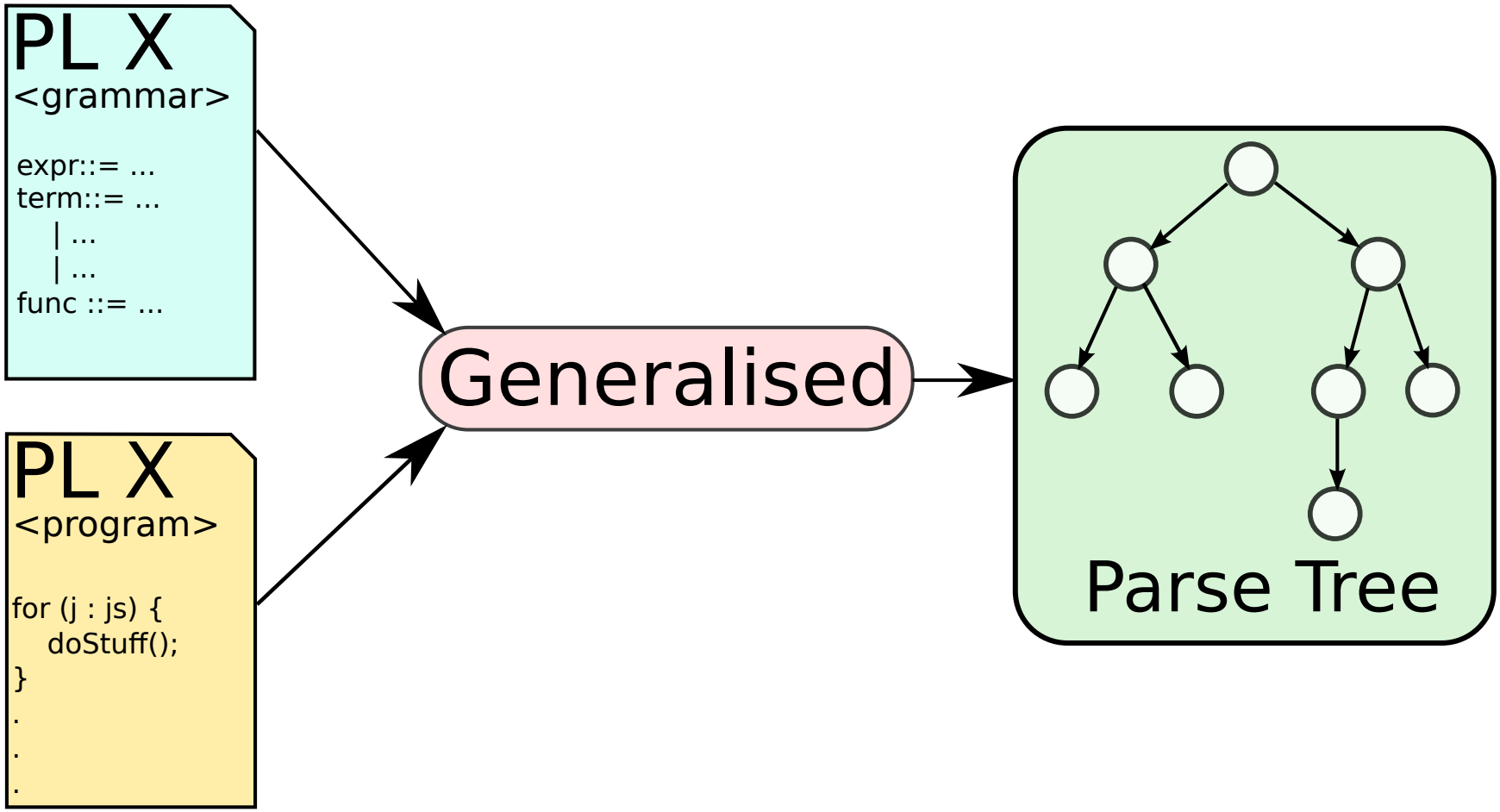
Parsing composition



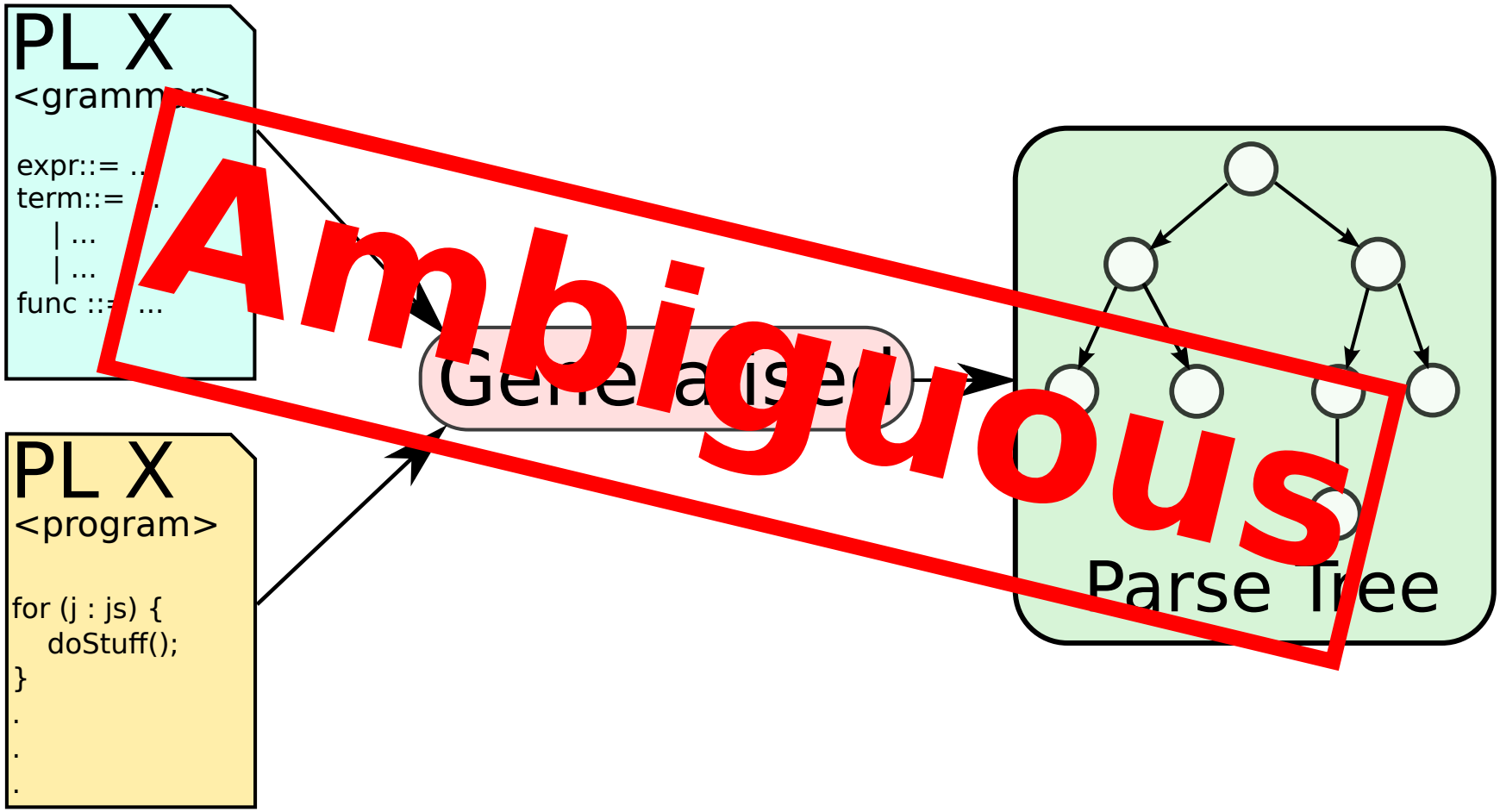
Parsing composition



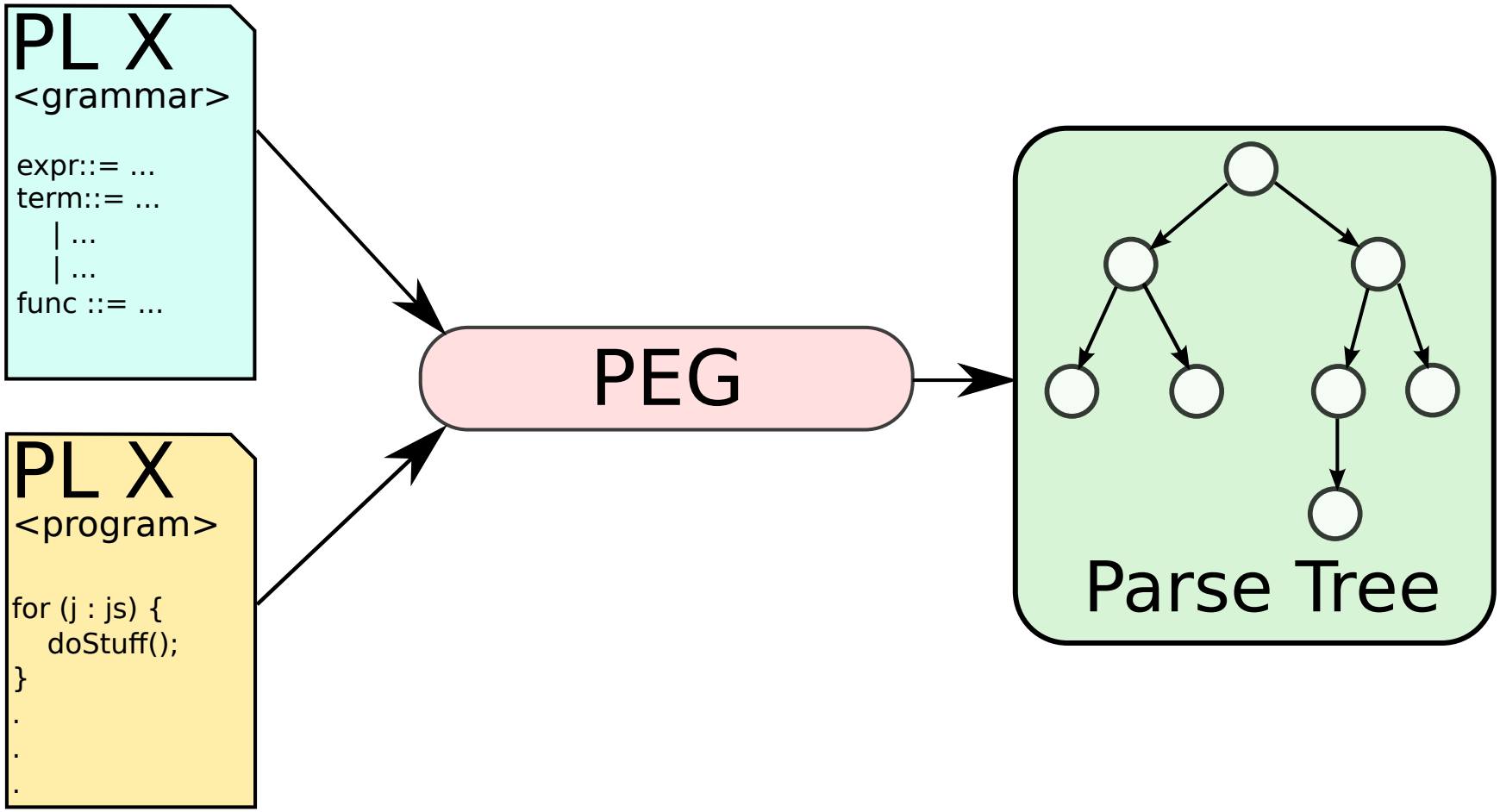
Parsing composition



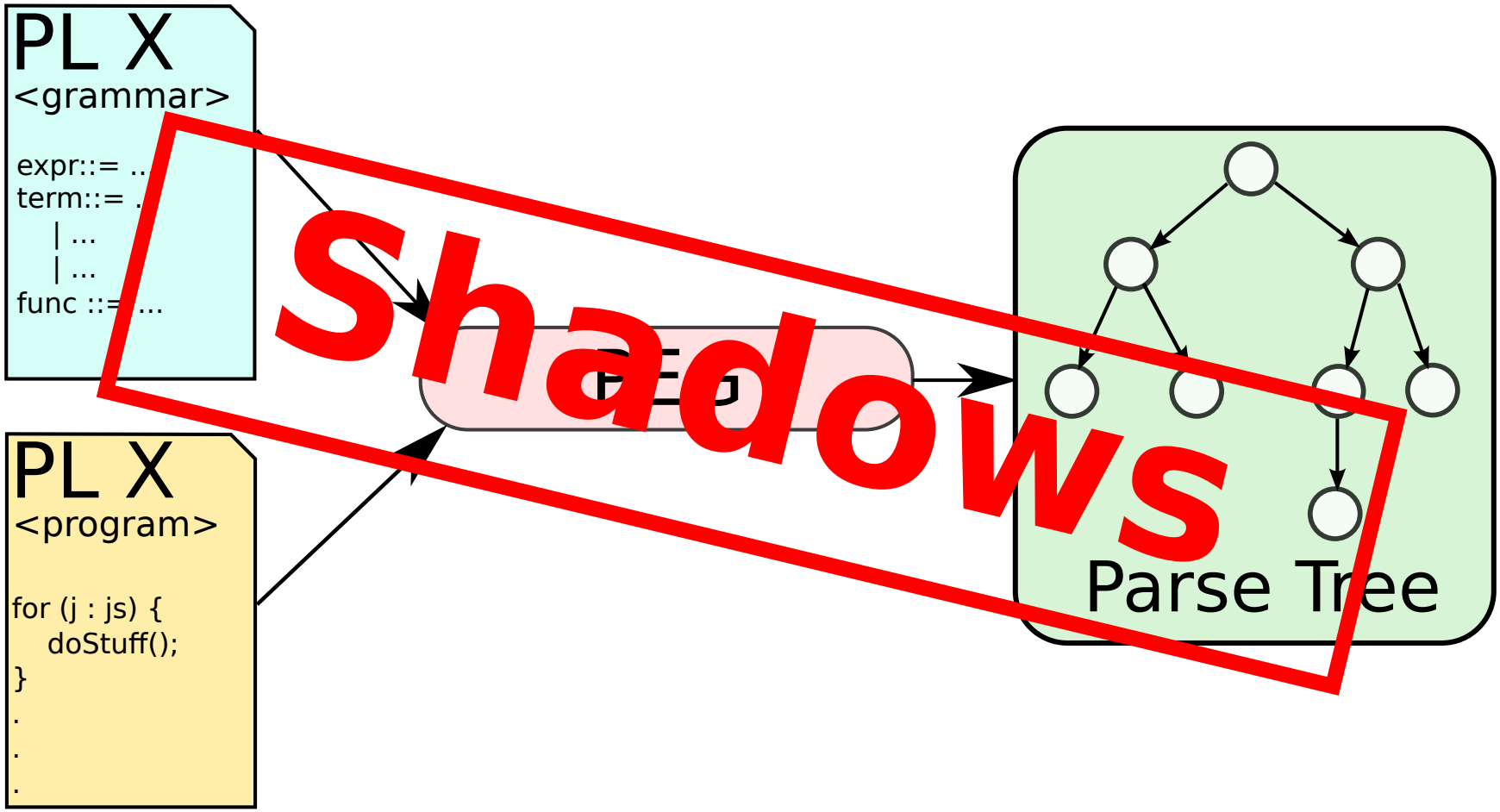
Parsing composition



Parsing composition



Parsing composition



The only choice?

Syntax-directed editing (SDE)

The challenge

Challenge:
SDE's power +
a text editor feel?

Demo

Challenges

- Whitespace-sensitive languages
- Incremental AST

Indentation

The traditional way to parse indentation based languages is too slow.

Indentation

```
def calc_indentl(l):
  if prev(l) == None:
      l.indentl = 0
  elif prev(l).wsl == l.wsl:
      l.indentl = prev(l).indentl
  elif prev(l).wsl < l.wsl:
      l.indentl = prev(l).indentl + 1
  else:
      assert prev(l).wsl > l.wsl
      prevl = prev(prev(l))
      while prevl != None:
          if prevl.wsl == l.wsl:
              l.indentl = prevl.indentl
              return
          elif prevl.wsl < l.wsl:
              break
          prevl = prev(prevl)
      mark_unbalanced(l)
```

Indentation

```
def calc_indentl(l):
  if prev(l) == None:
      l.indentl = 0
  elif prev(l).wsl == l.wsl:
      l.indentl = prev(l).indentl
  elif prev(l).wsl < l.wsl:
      l.indentl = prev(l).indentl + 1
  else:
      assert prev(l).wsl > l.wsl
      prevl = prev(prev(l))
      while prevl != None:
          if prevl.wsl == l.wsl:
              l.indentl = prevl.indentl
              return
          elif prevl.wsl < l.wsl:
              break
          prevl = prev(prevl)
      mark_unbalanced(l)
```

- Each line knows its own indentation level

Indentation

```
def calc_indentl(l):
  if prev(l) == None:
      l.indentl = 0
  elif prev(l).wsl == l.wsl:
      l.indentl = prev(l).indentl
  elif prev(l).wsl < l.wsl:
      l.indentl = prev(l).indentl + 1
  else:
      assert prev(l).wsl > l.wsl
      prevl = prev(prev(l))
      while prevl != None:
          if prevl.wsl == l.wsl:
              l.indentl = prevl.indentl
              return
          elif prevl.wsl < l.wsl:
              break
          prevl = prev(prevl)
      mark_unbalanced(l)
```

- Each line knows its own indentation level
- Editing whitespace of a line recalculates its indentation level by looking at previous lines

Indentation

```
def calc_indentl(l):
  if prev(l) == None:
      l.indentl = 0
  elif prev(l).wsl == l.wsl:
      l.indentl = prev(l).indentl
  elif prev(l).wsl < l.wsl:
      l.indentl = prev(l).indentl + 1
  else:
      assert prev(l).wsl > l.wsl
      prevl = prev(prev(l))
      while prevl != None:
          if prevl.wsl == l.wsl:
              l.indentl = prevl.indentl
              return
          elif prevl.wsl < l.wsl:
              break
          prevl = prev(prevl)
      mark_unbalanced(l)
```

- Each line knows its own indentation level
- Editing whitespace of a line recalculates its indentation level by looking at previous lines
- Afterwards, affected succeeding lines need to be updated as well

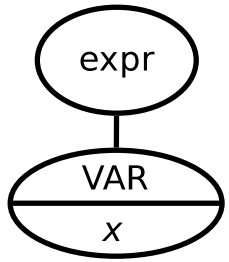
Indentation

```
def calc_indentl(l):
  if prev(l) == None:
      l.indentl = 0
  elif prev(l).wsl == l.wsl:
      l.indentl = prev(l).indentl
  elif prev(l).wsl < l.wsl:
      l.indentl = prev(l).indentl + 1
  else:
      assert prev(l).wsl > l.wsl
      prevl = prev(prev(l))
      while prevl != None:
          if prevl.wsl == l.wsl:
              l.indentl = prevl.indentl
              return
          elif prevl.wsl < l.wsl:
              break
          prevl = prev(prevl)
      mark_unbalanced(l)
```

- Each line knows its own indentation level
- Editing whitespace of a line recalculates its indentation level by looking at previous lines
- Afterwards, affected succeeding lines need to be updated as well
- Worst-case: O(n)

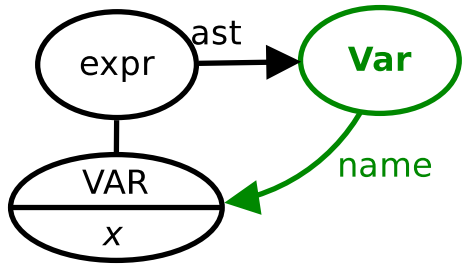
Incremental ASTs

```
print x
```



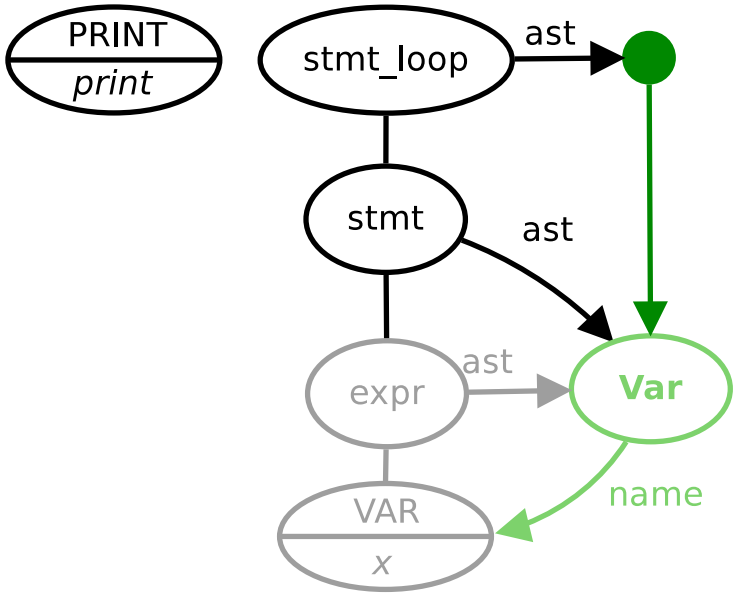
Incremental ASTs

```
print x
```



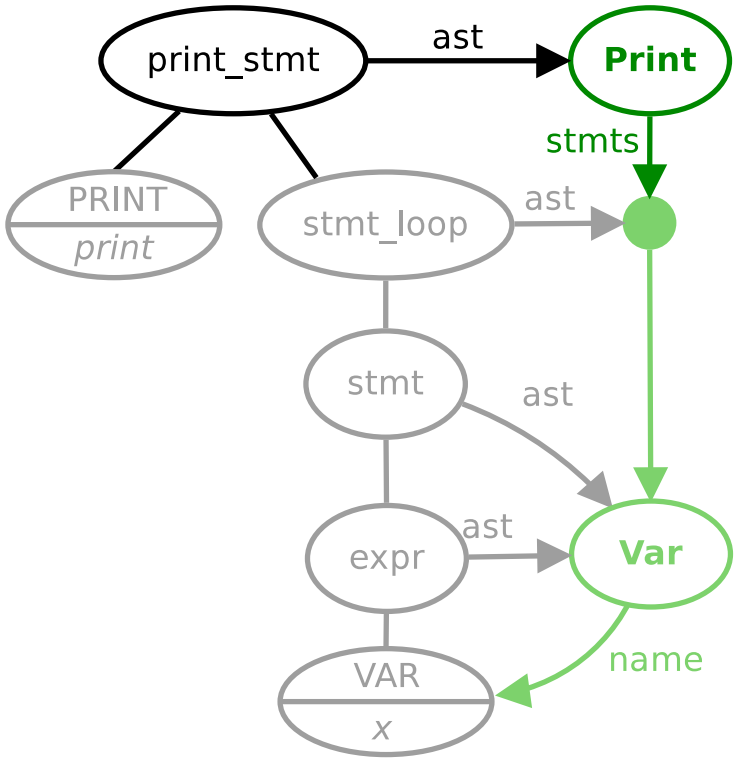
Incremental ASTs

```
print x
```



Incremental ASTs

```
print x
```

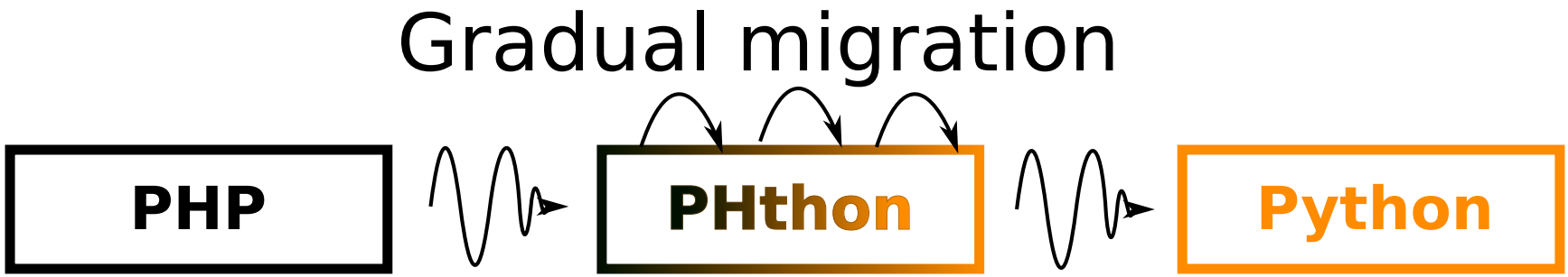


What is it good for?

First experiment

Unipycation

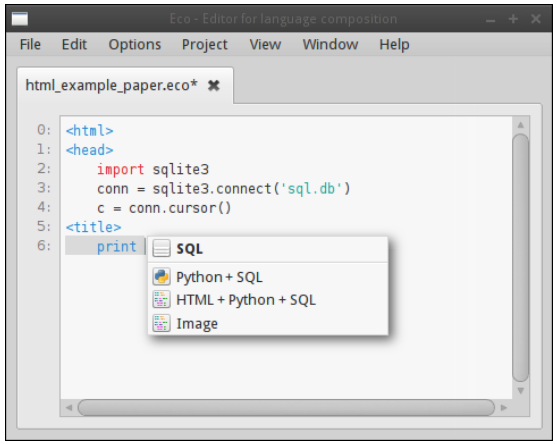
Recent experiment

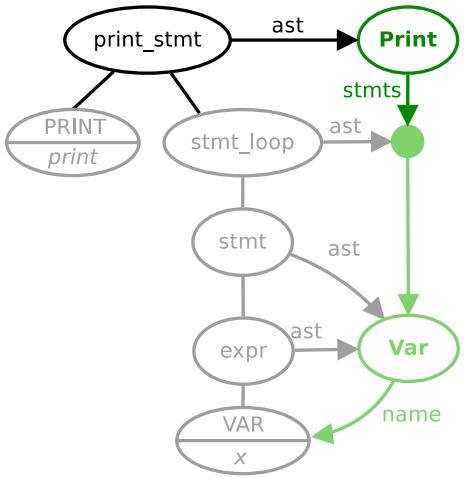


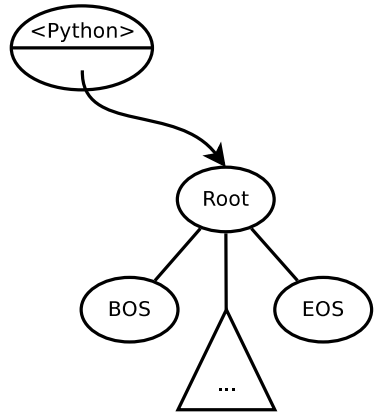
What's next?

Incremental semantic analysis on ASTs and code generation

Thanks for listening







http://soft-dev.org/