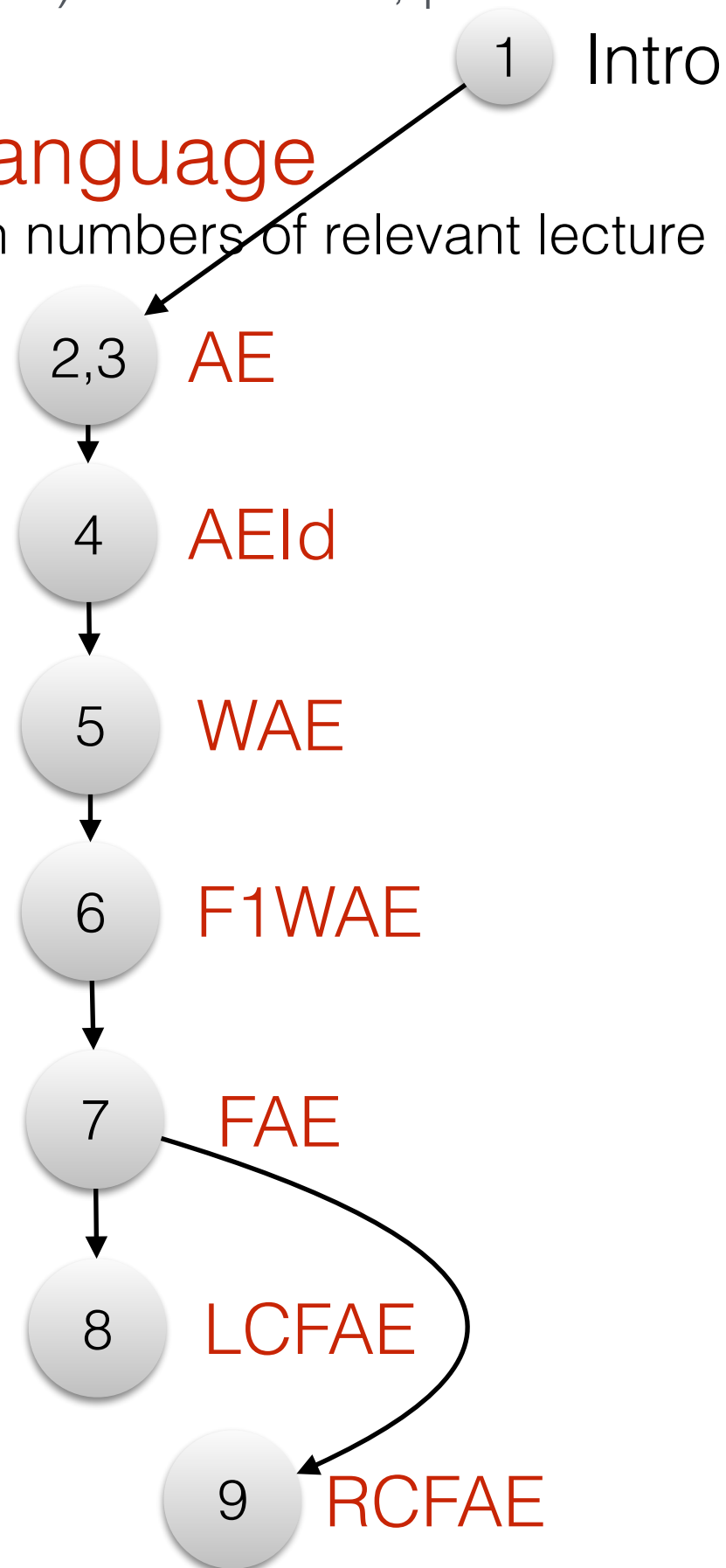


→ = possible reading order

Language
(with numbers of relevant lecture notes)



Concepts covered

Interpretation, Desugaring

Environments, Visitors (+ Currying)

Binding (+ Scope), Substitution

First-Order Functions,
Static & Dynamic Scoping

Higher-Order Functions,
Substitution- Vs. Environment-Based Interpreter,
Closures (!)

Call-By-Name, Call-By-Need,
Thunks

Recursion with **Letrec**

To be continued...

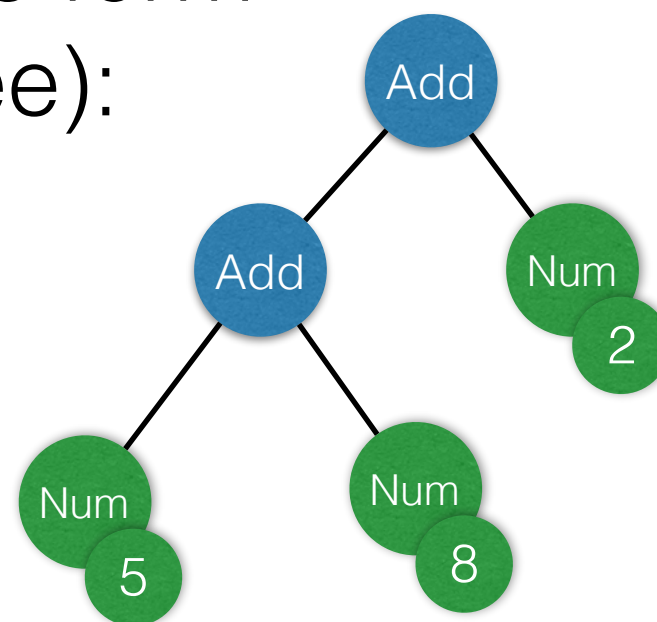


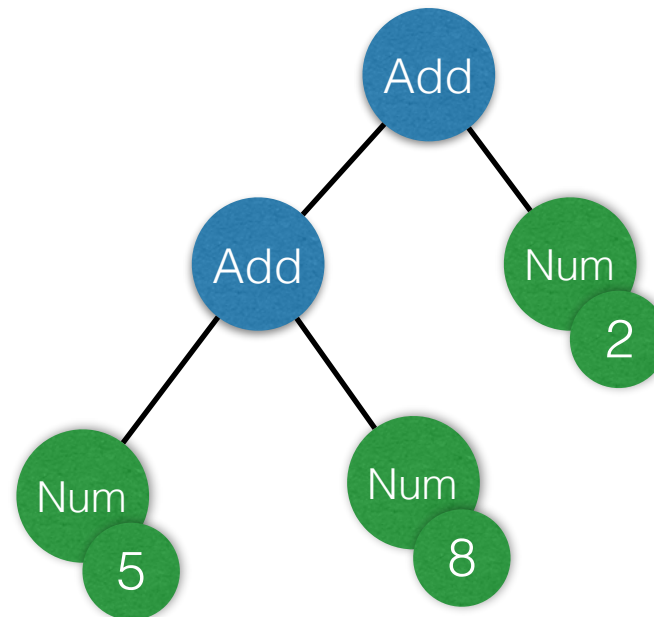
- A formal language consists of:
- types of nodes, each with
 - arity, and possibly
 - other information (data)

In AE, those are

- Num: arity 0, other data: a number
- Add: arity 2

An example term in tree form
(Abstract Syntax Tree):



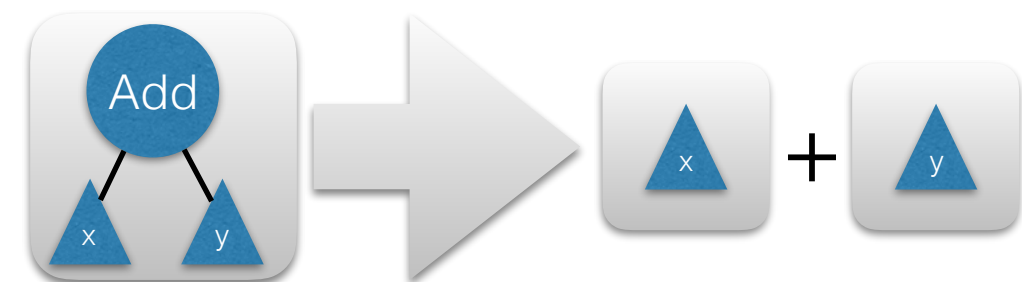


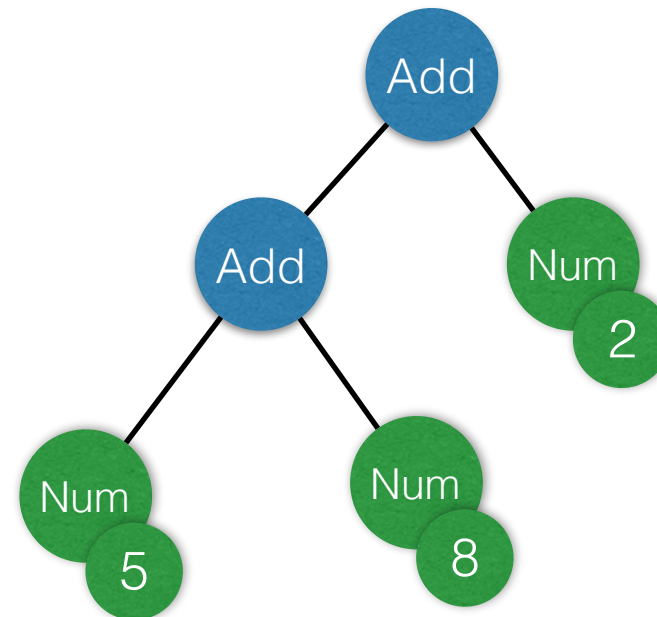
Interpretation: assigns a **value** to each term of the language

Example (Scala):

```
def eval(e: Exp): Int =  
  e match {  
    case Num(n) => n  
    case Add(lhs, rhs) =>  
      eval(lhs) + eval(rhs)  
  }
```

visually:





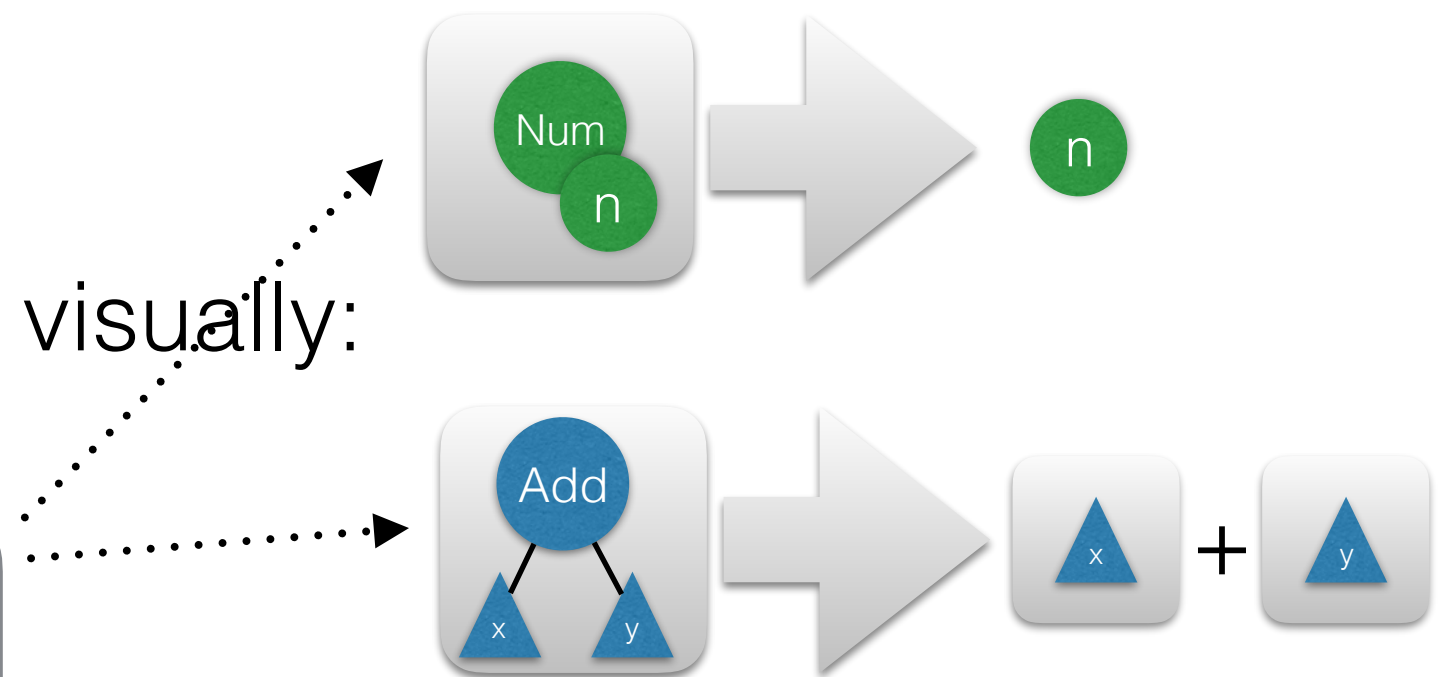
Interpretation: assigns a **value** to each term of the language

Example (Scala):

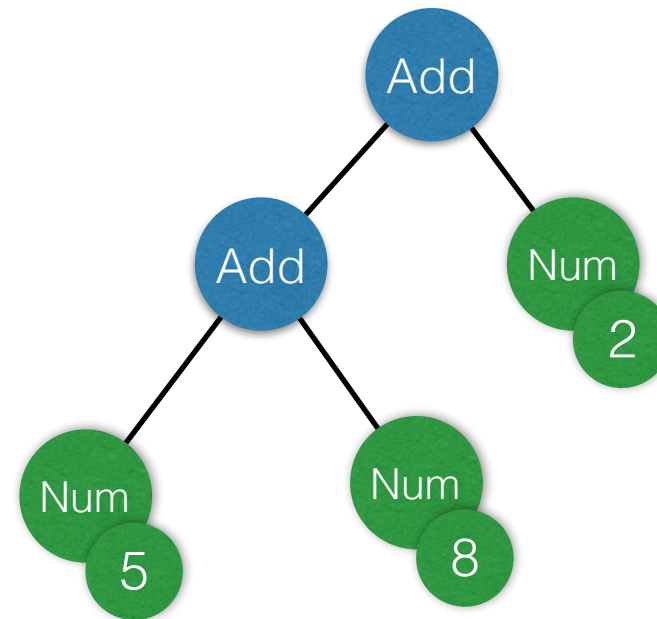
```
def eval(e: Exp): Int =  
  e match {  
    case Num(n) => n  
    case Add(lhs, rhs) =>  
      eval(lhs) + eval(rhs)  
  }
```

Matching on the
node type

visually:



2,3 AE



Interpretation: assigns a **value** to each term of the language

Example (Scala):

```
def eval(e: Expr): Int = {  
  e match {  
    case Add(lhs, rhs) =>  
      eval(lhs) + eval(rhs)  
  }  
}
```

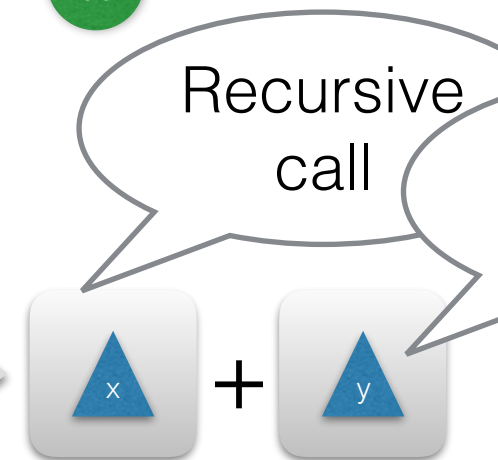
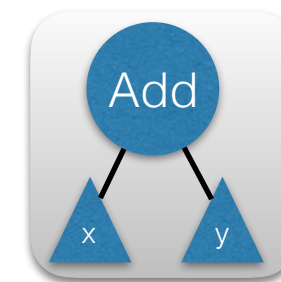
Recursive
call

Recursive
call

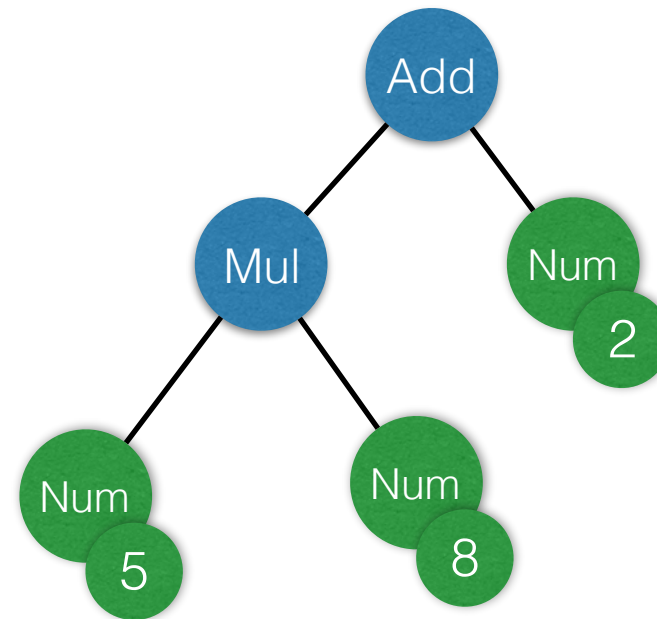
ually:



n



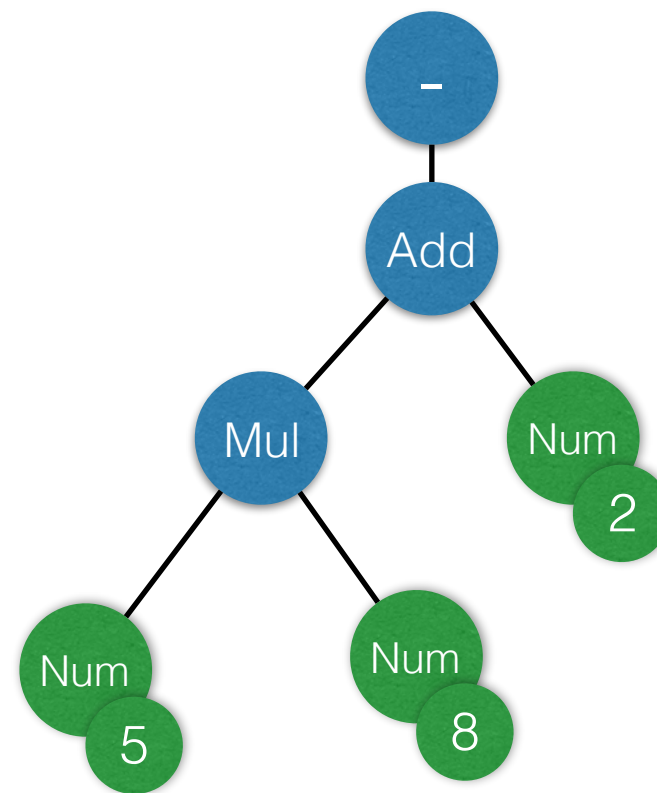
2,3 AE



Mul

We add a new node type for multiplication

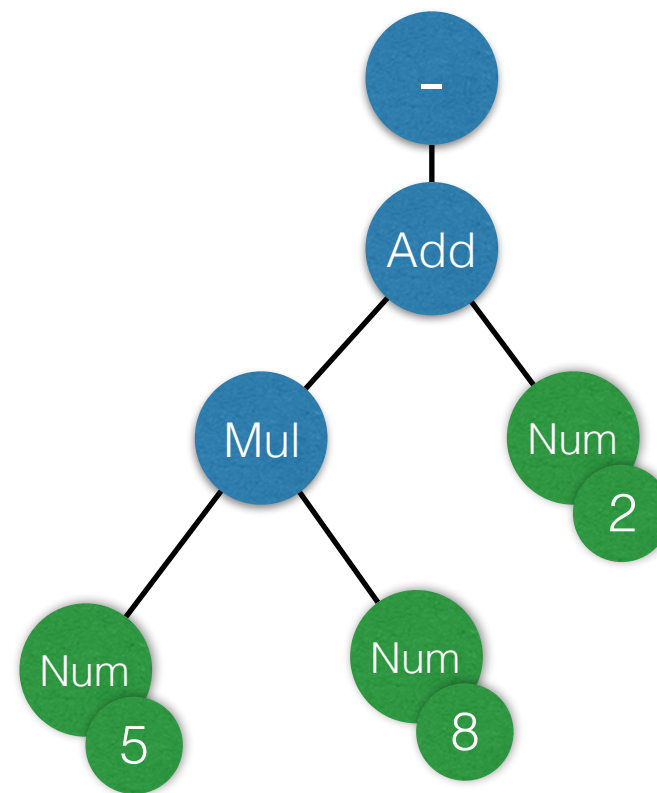
2,3 AE



−

Then we add another new node type for unary minus

2,3 AE



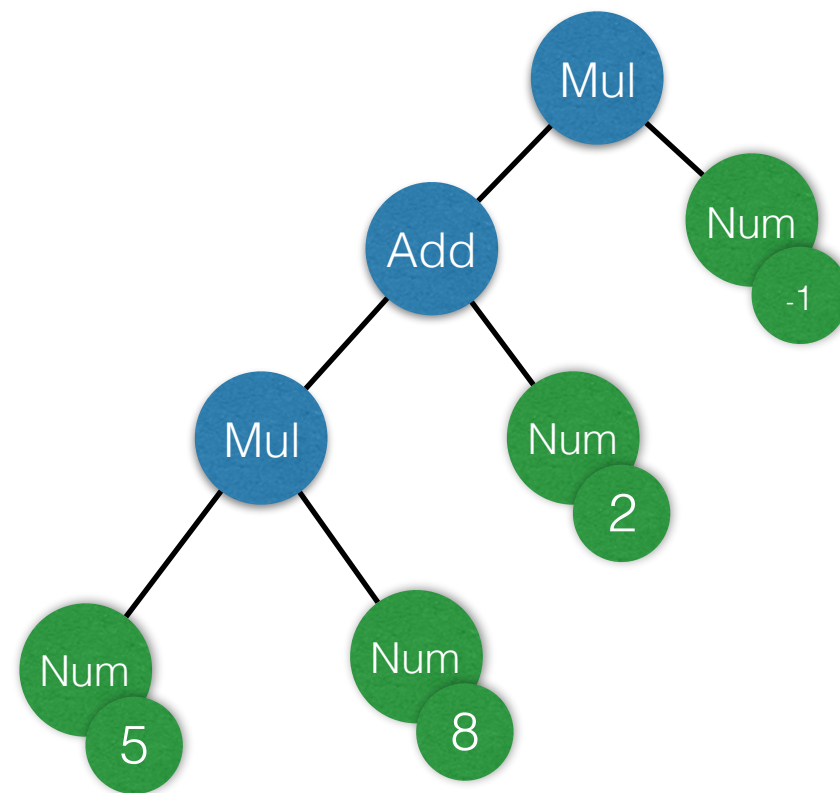
-

But we do not really need a new node type for this.

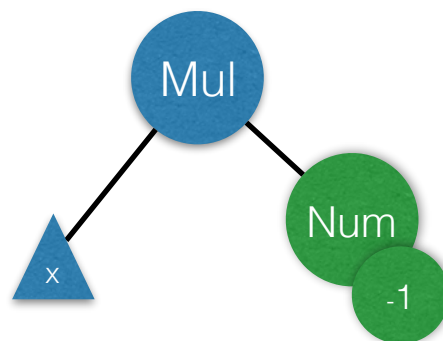
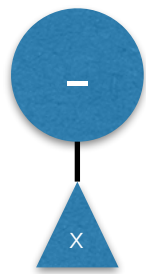
Desugaring: transforms the AST to eliminate a node type

1. Semantics must stay the same.
2. Ideally we do not want to look into the child nodes for this.

2,3 AE

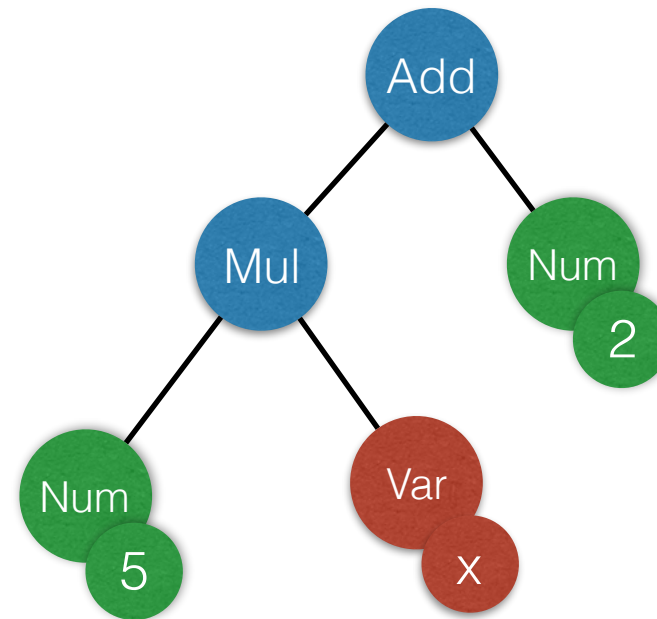


desugar

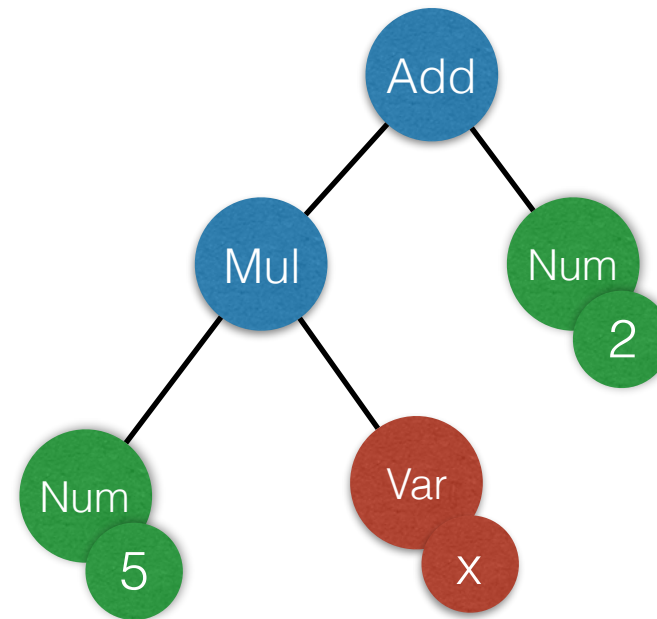


Desugaring: transforms the AST to eliminate a node type

1. Semantics must stay the same.
2. Ideally we do not want to look into the child nodes for this.

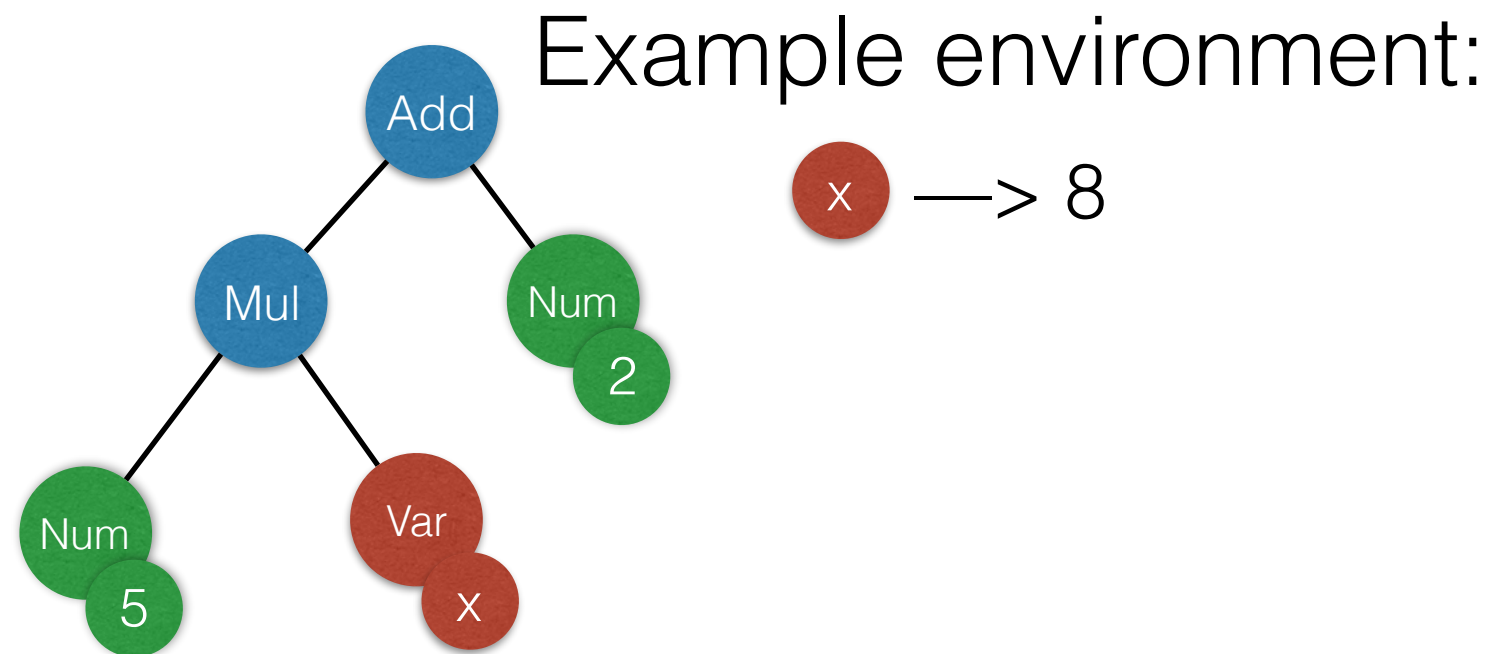


 We add a new node for a new feature: **variables**



We add a new node for a new feature: **variables**
On its own, a term does not have meaning anymore.

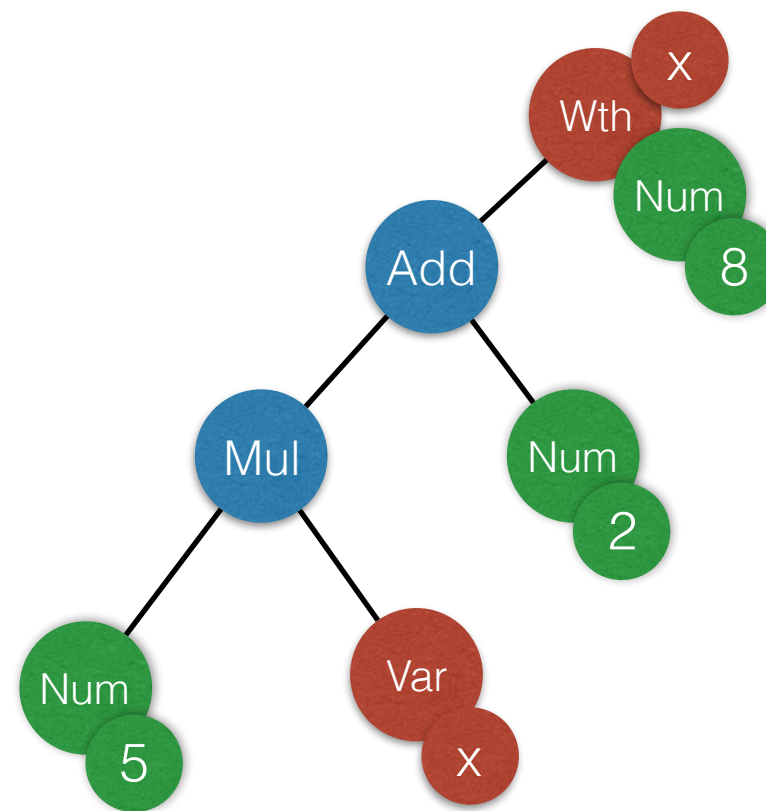
We need to give the interpreter a map
from variable names to values.
This map is called an environment.



On its own, a term does not have meaning anymore.

We need to give the interpreter a map
from variable names to values.
This map is called an environment.

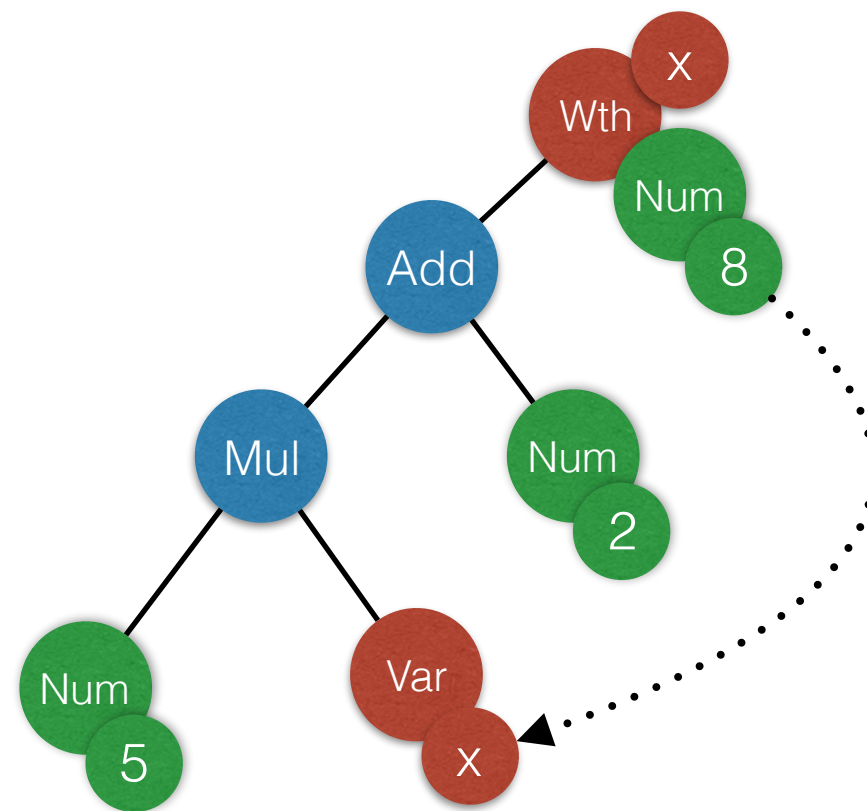
5 WAE



With

We add a new node for binding variables

5 WAE



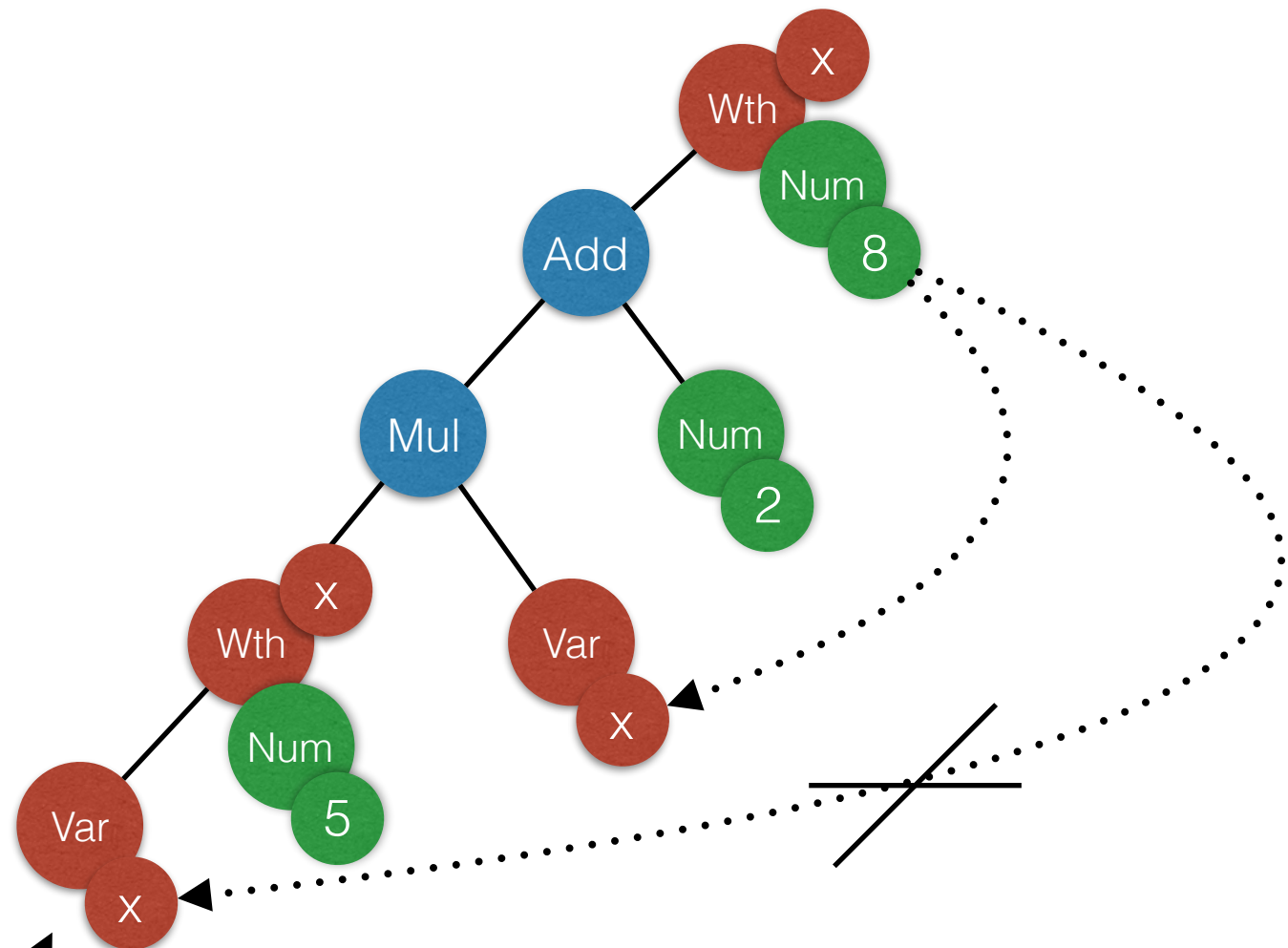
With

We add a new node for binding variables

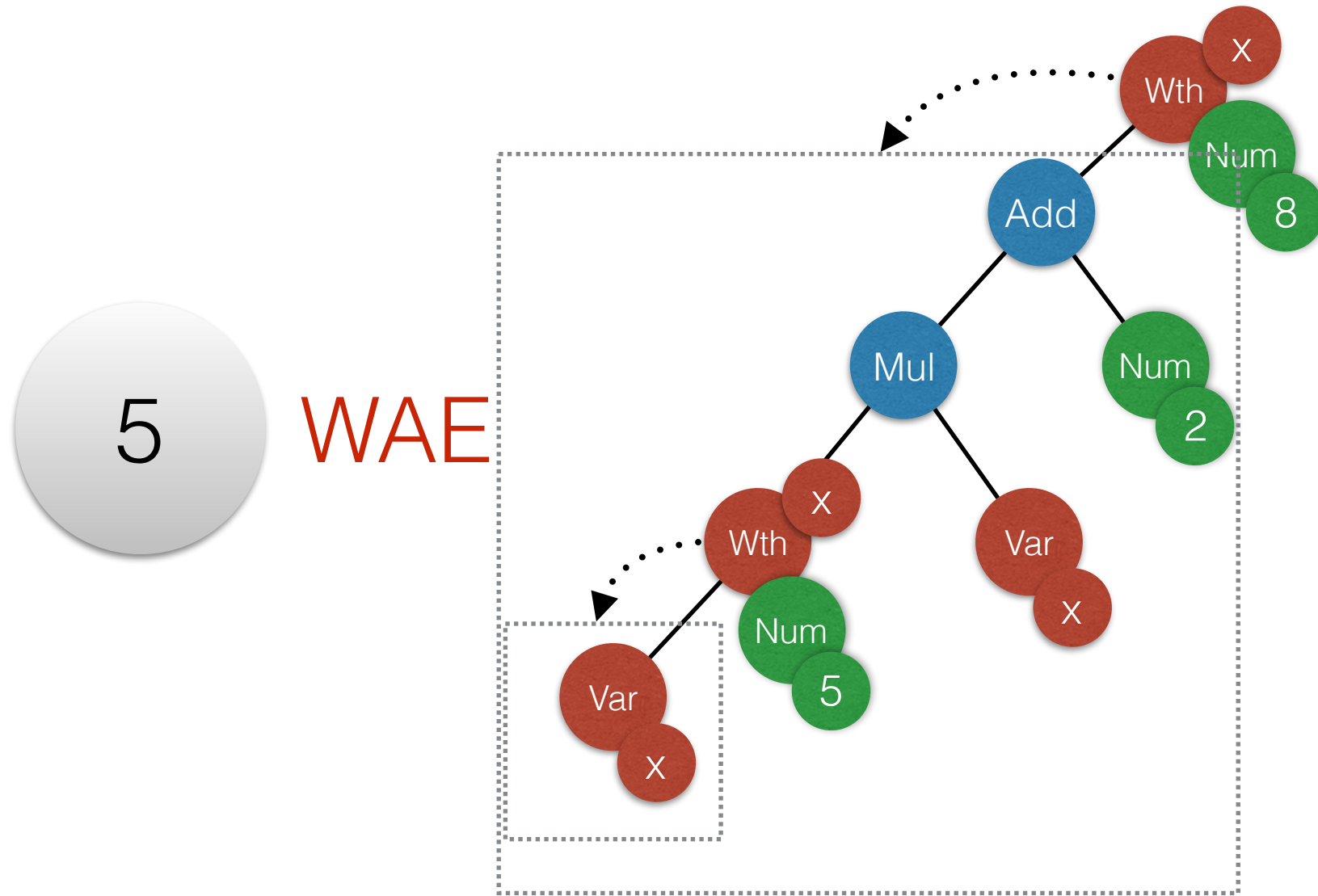
An interpreter can produce a value from such a term by:
substituting the value from evaluating the bound term
for all **free** variables with the name

5

WAE

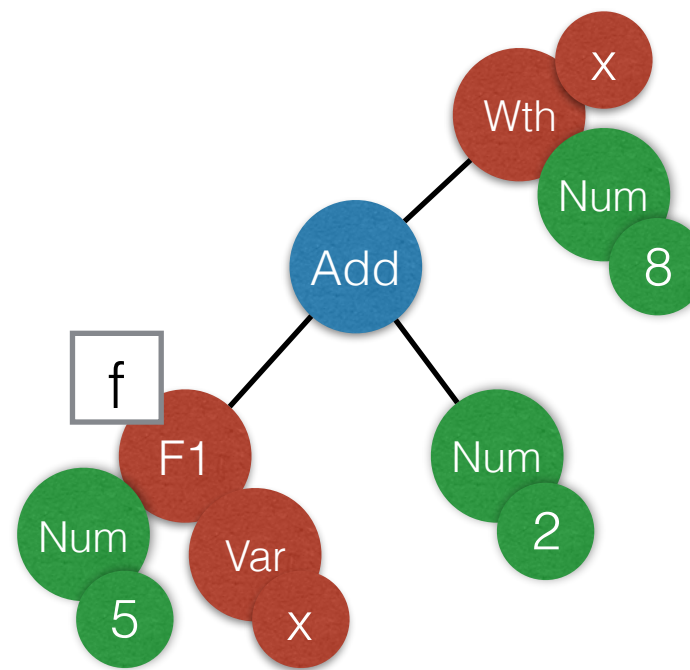


not free in the overall term



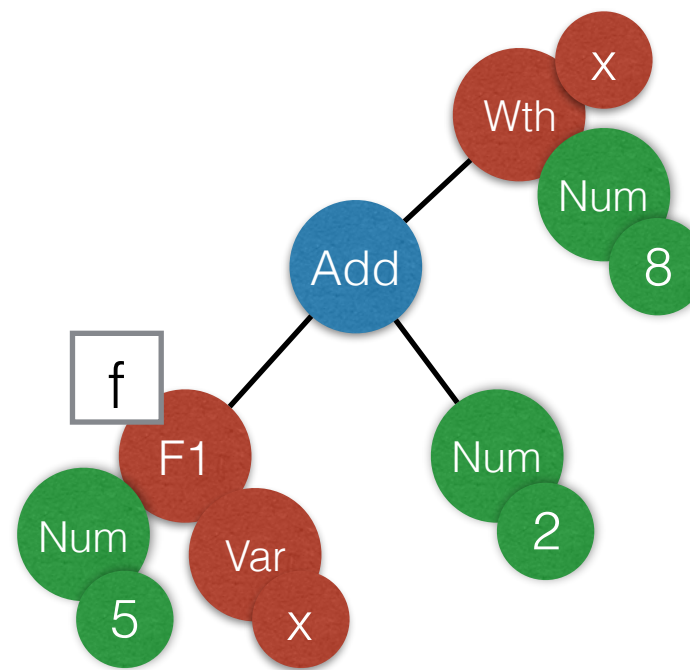
Static (lexical) scope

6 F1WAE



F1

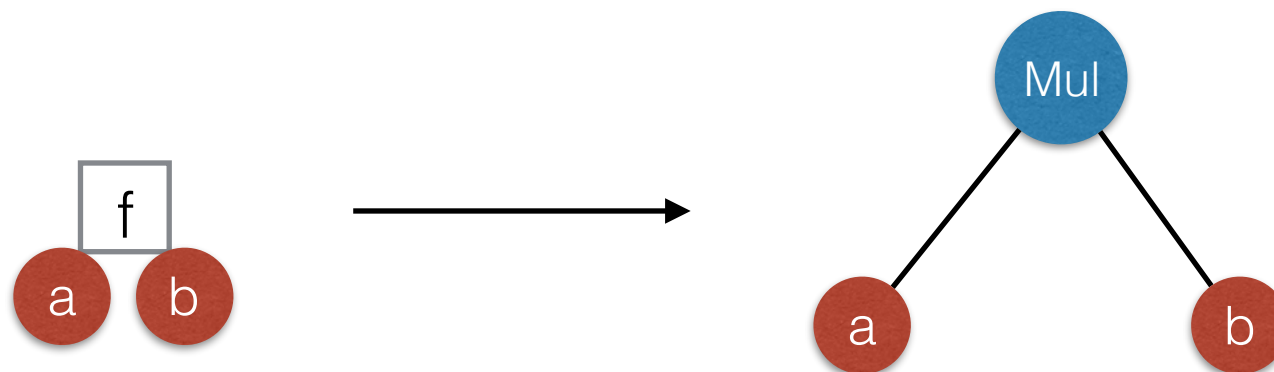
We add a new node for calling first-order functions



F1

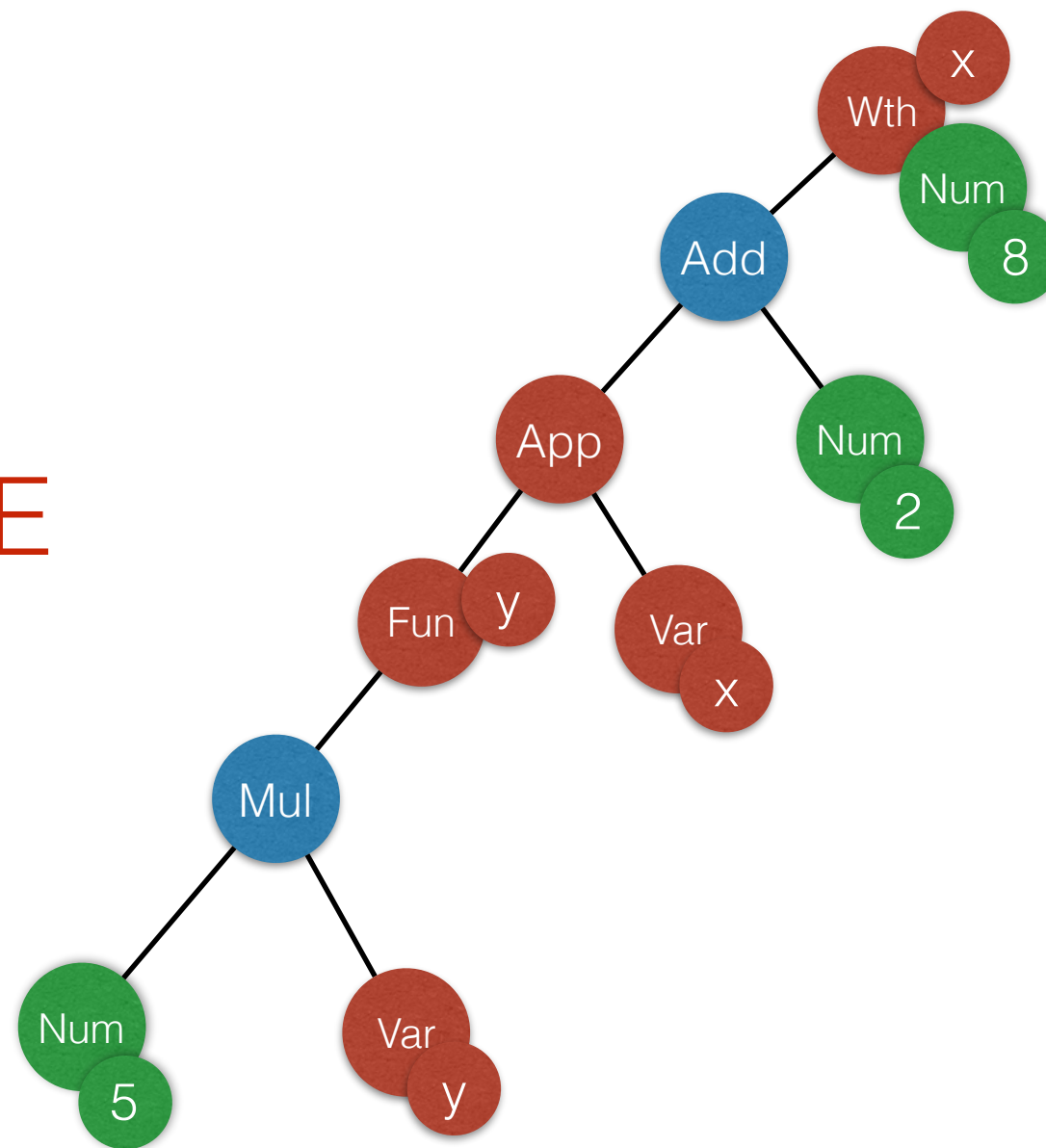
We add a new node for calling first-order functions

We have a dictionary with an entry per function:



7

FAE



Fun

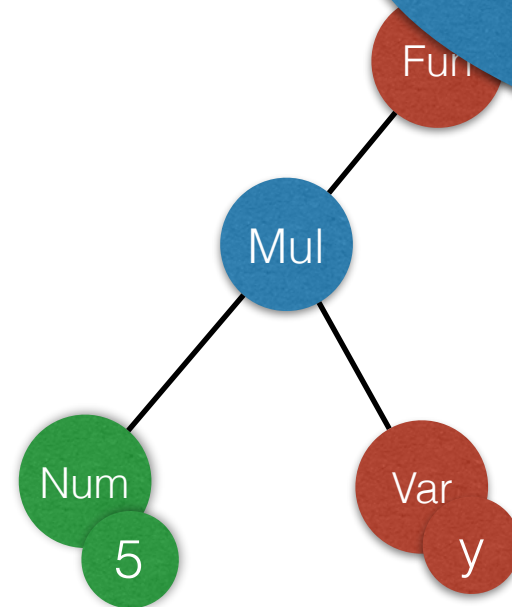
Now we turn functions into **first-class** objects, that is, their definition happens locally in the AST

App

Thus we also need a new node to call, or **apply**, them

7

FAE



also **higher-order**, because functions can now be input/output of other functions

Fun

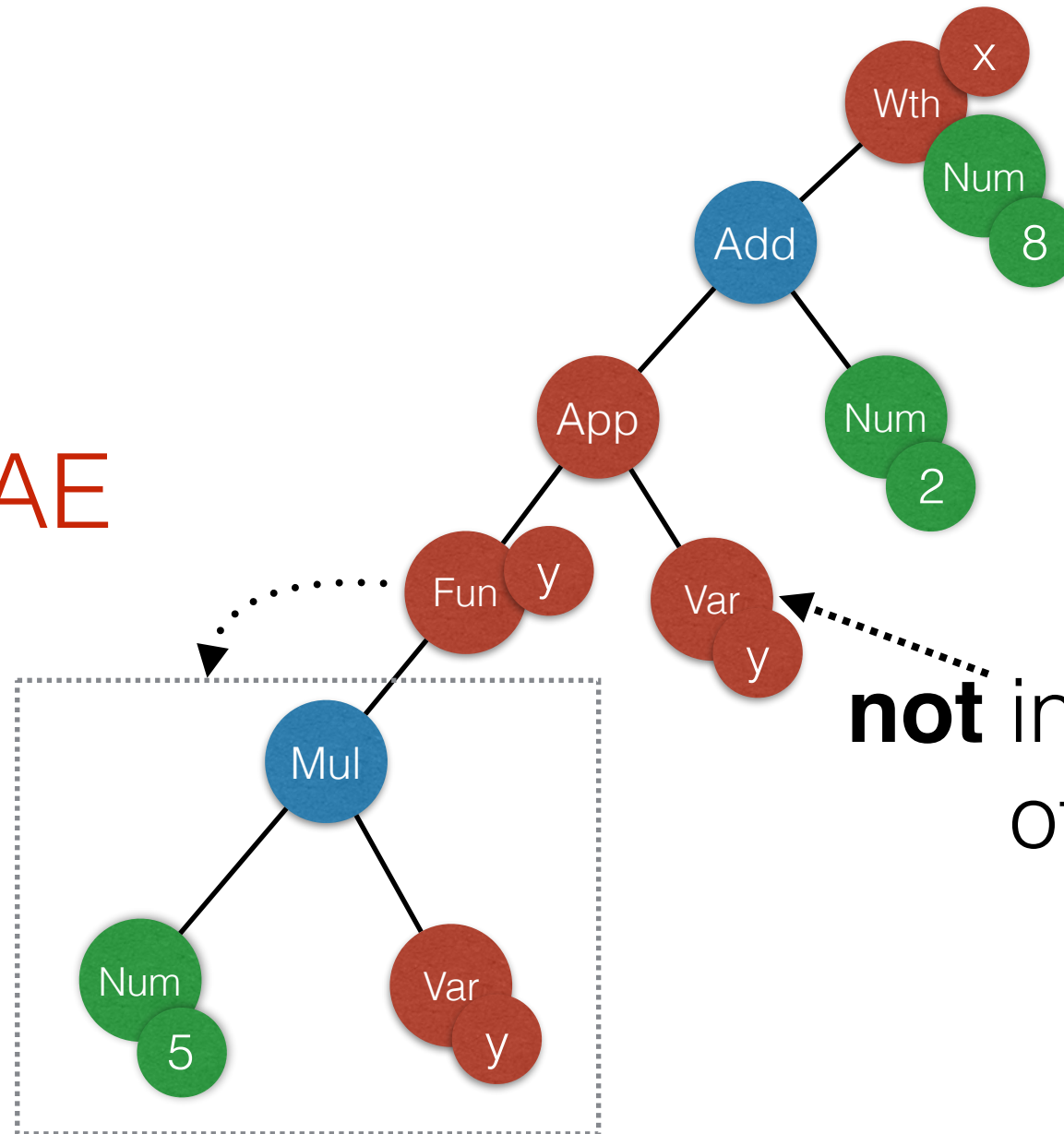
Now we turn functions into **first-class** objects, that is, their definition happens locally in the AST

App

Thus we also need a new node to call, or **apply**, them

7

FAE

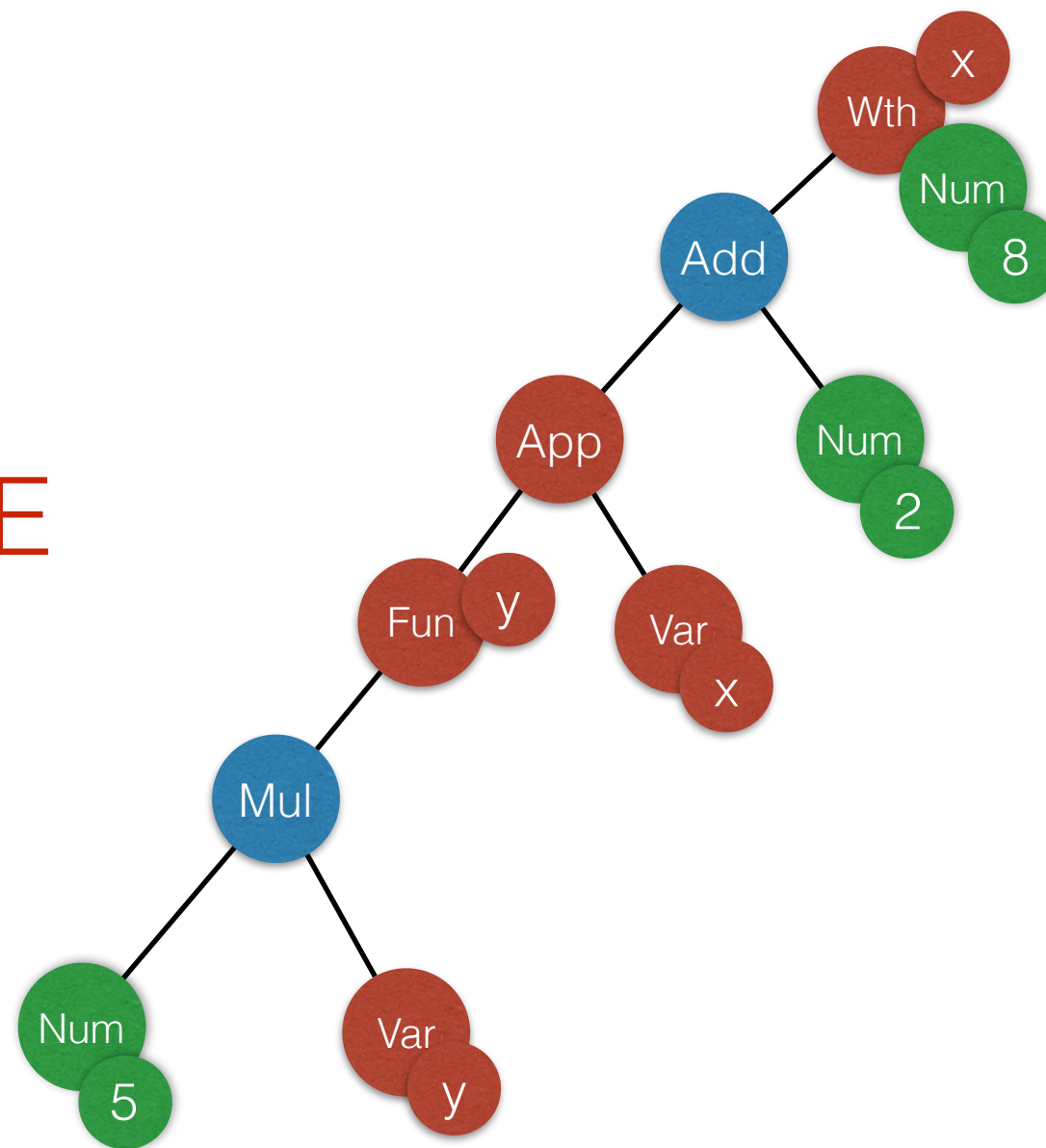


not in the static scope
of the function

static scope of the function

7

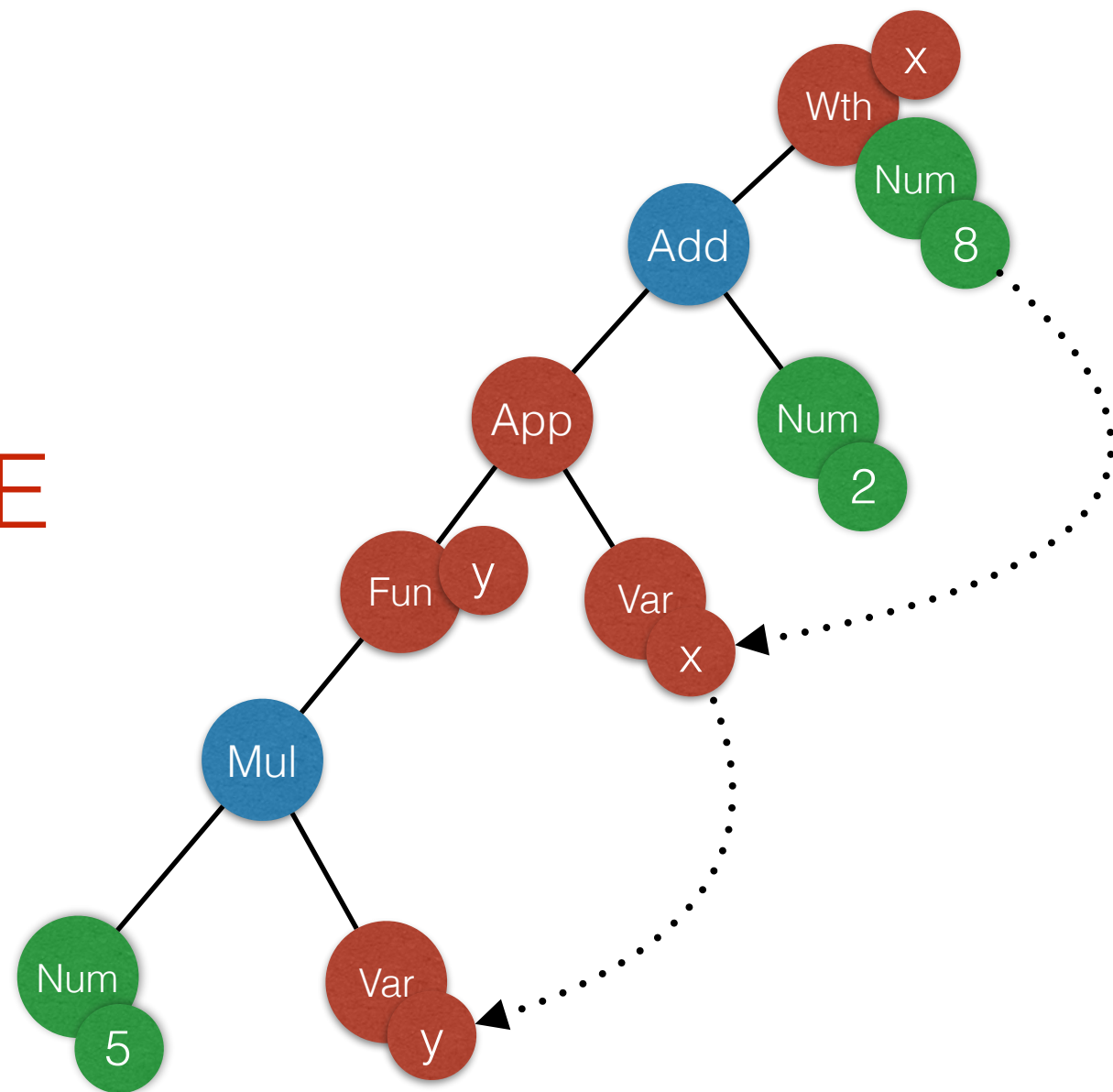
FAE



Two possible interpreters:
substitution-based and environment-based

7

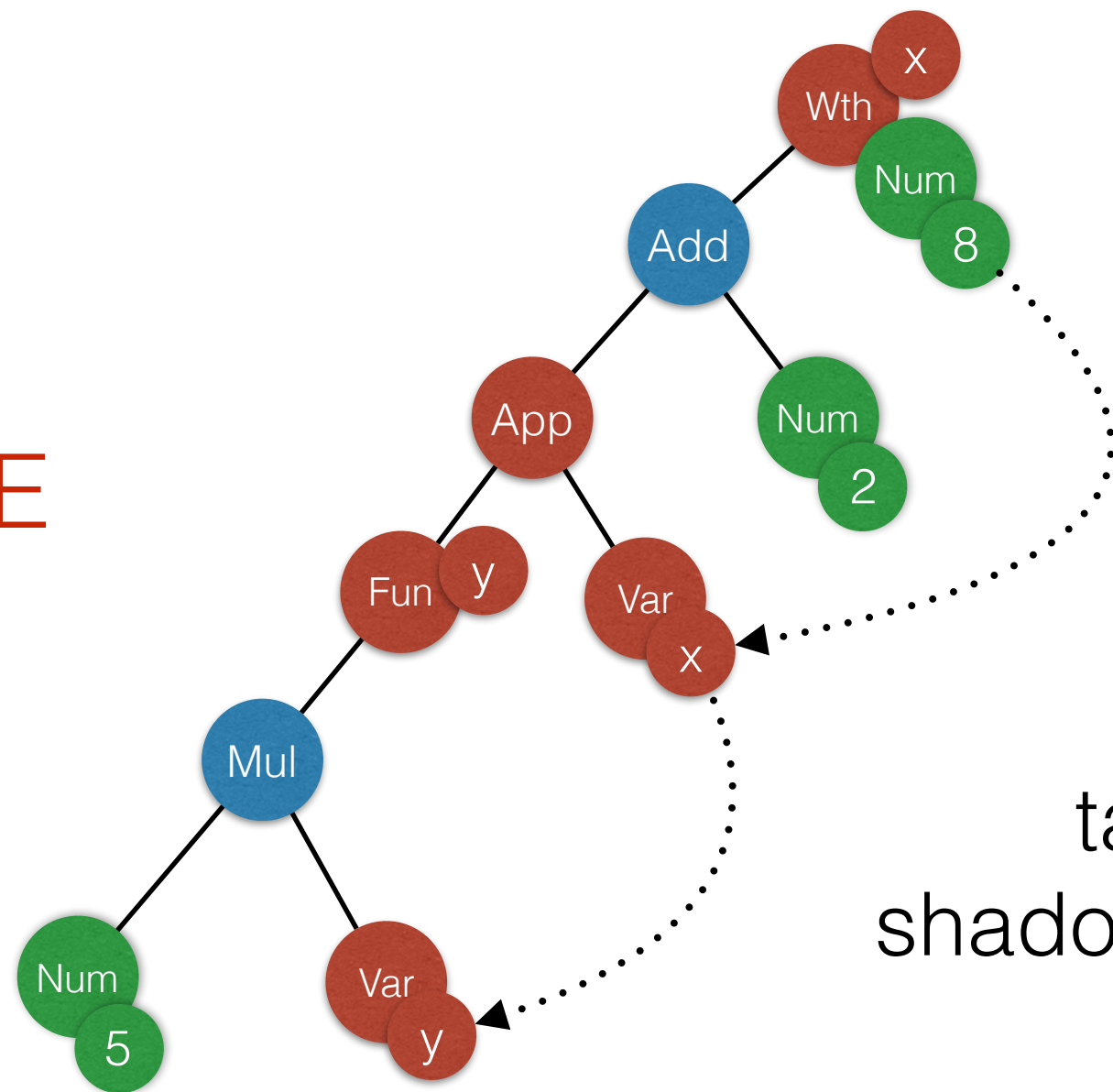
FAE



Substitution-based: just substitutes step-by-step

7

FAE



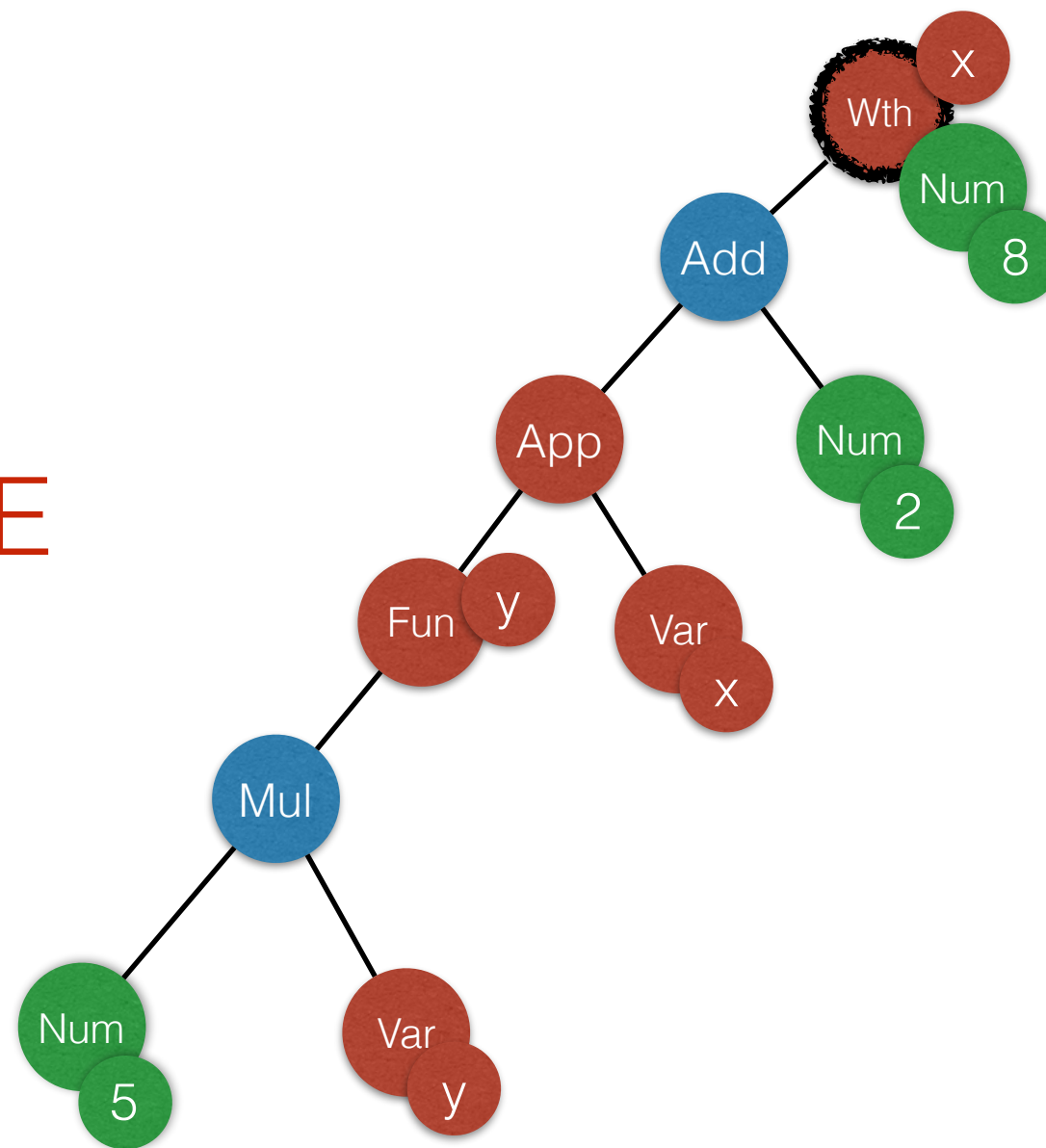
Substitution-based: just substitutes step-by-step



Environment-based: carries along an environment

7

FAE

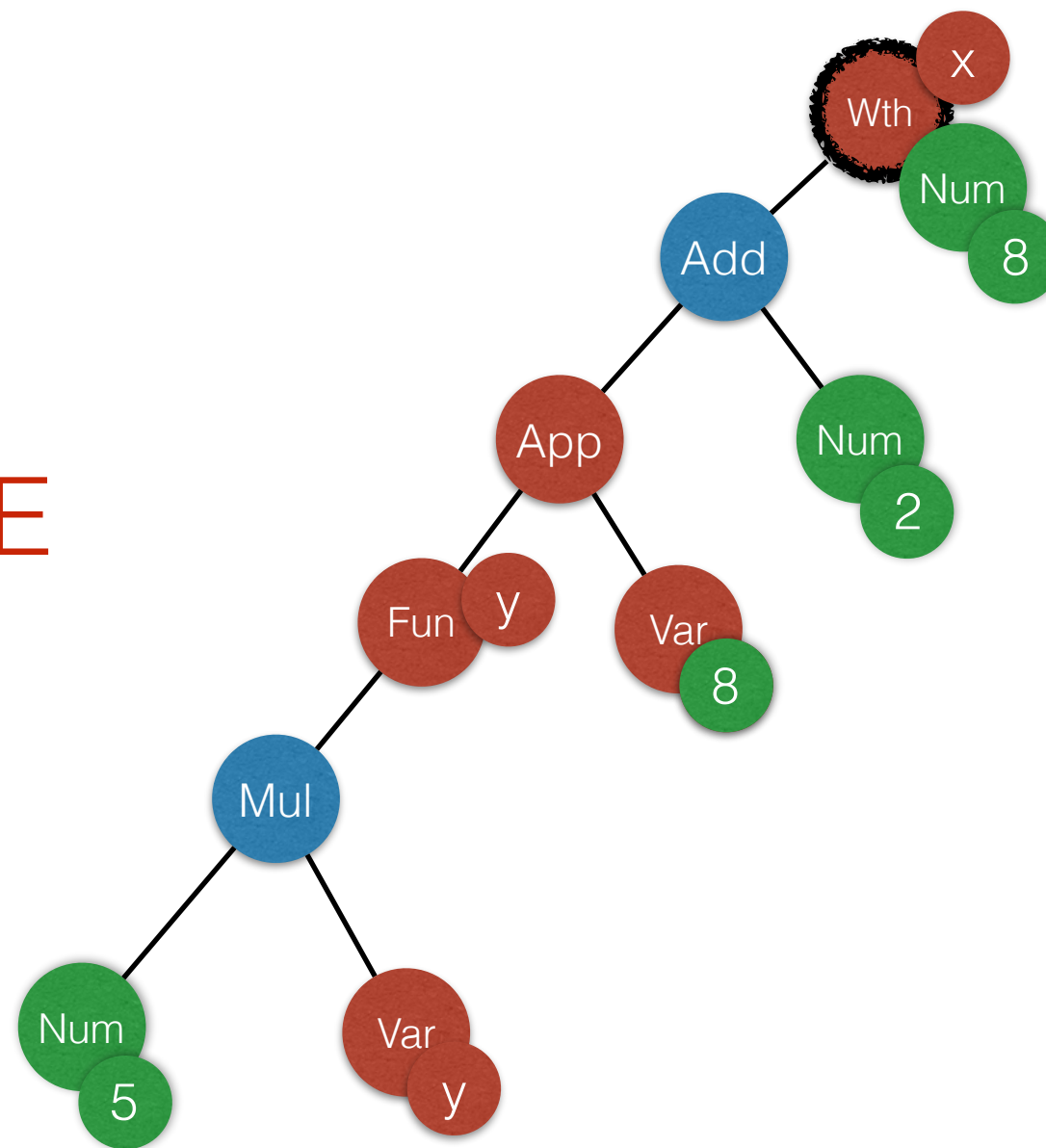


$x \rightarrow 8$

Environment-based: carries along an environment

7

FAE



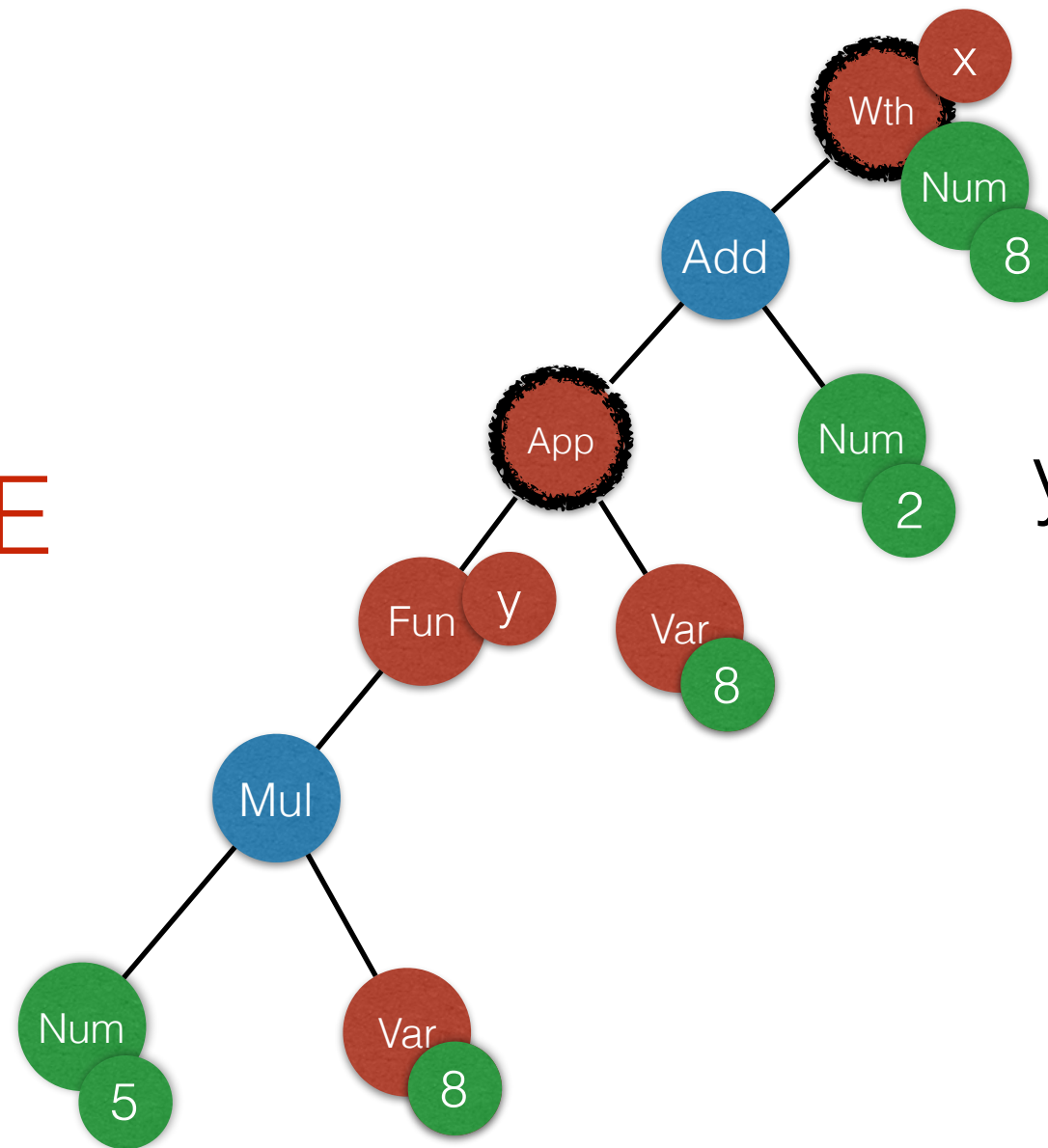
$x \longrightarrow 8$

Environment-based: carries along an environment



7

FAE



$x \longrightarrow 8$

$y \longrightarrow 8, x \longrightarrow 8$

Environment-based: carries along an environment

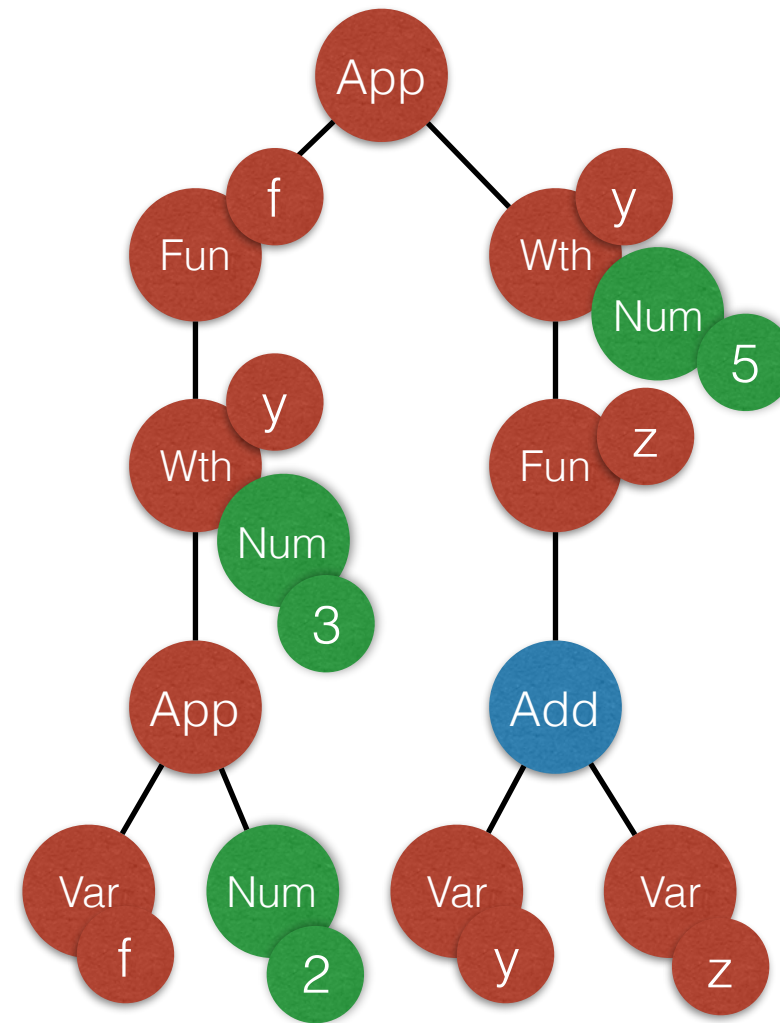


Note: Values can now also be functions, not just numbers!

But how exactly should they be represented?

7

FAE

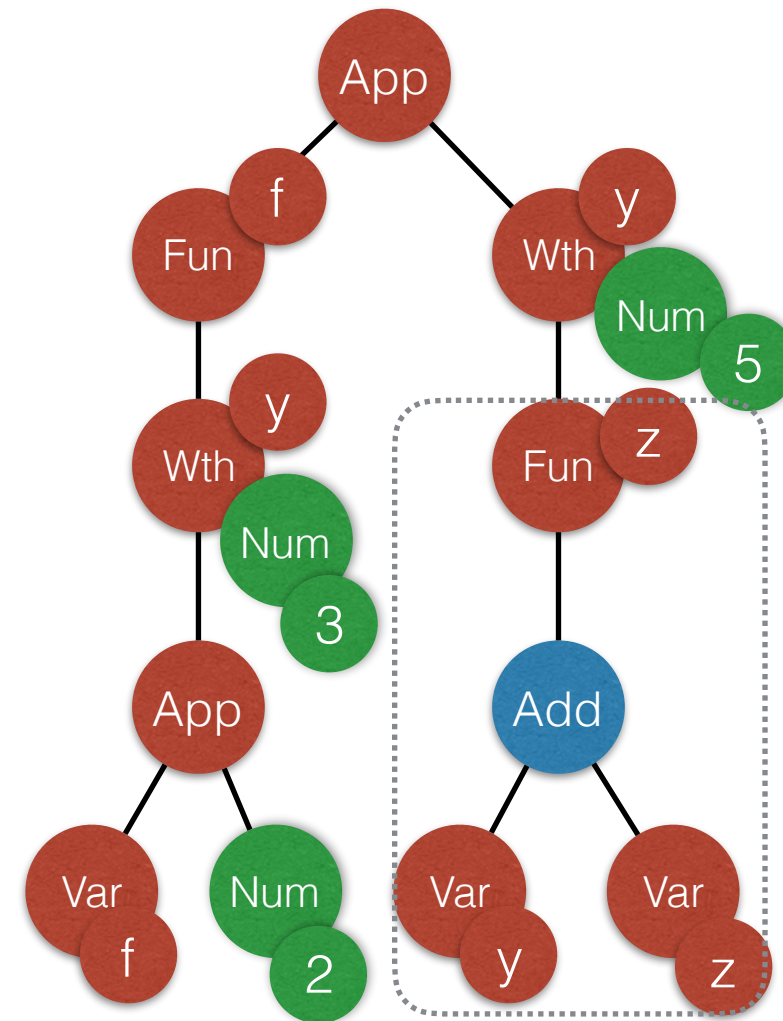


Note: Values can now also be functions, not just numbers!

But how exactly should they be represented as values?

7

FAE



First idea:
value = term

Note: Values can now also be functions, not just numbers!

But how exactly should they be represented as values?

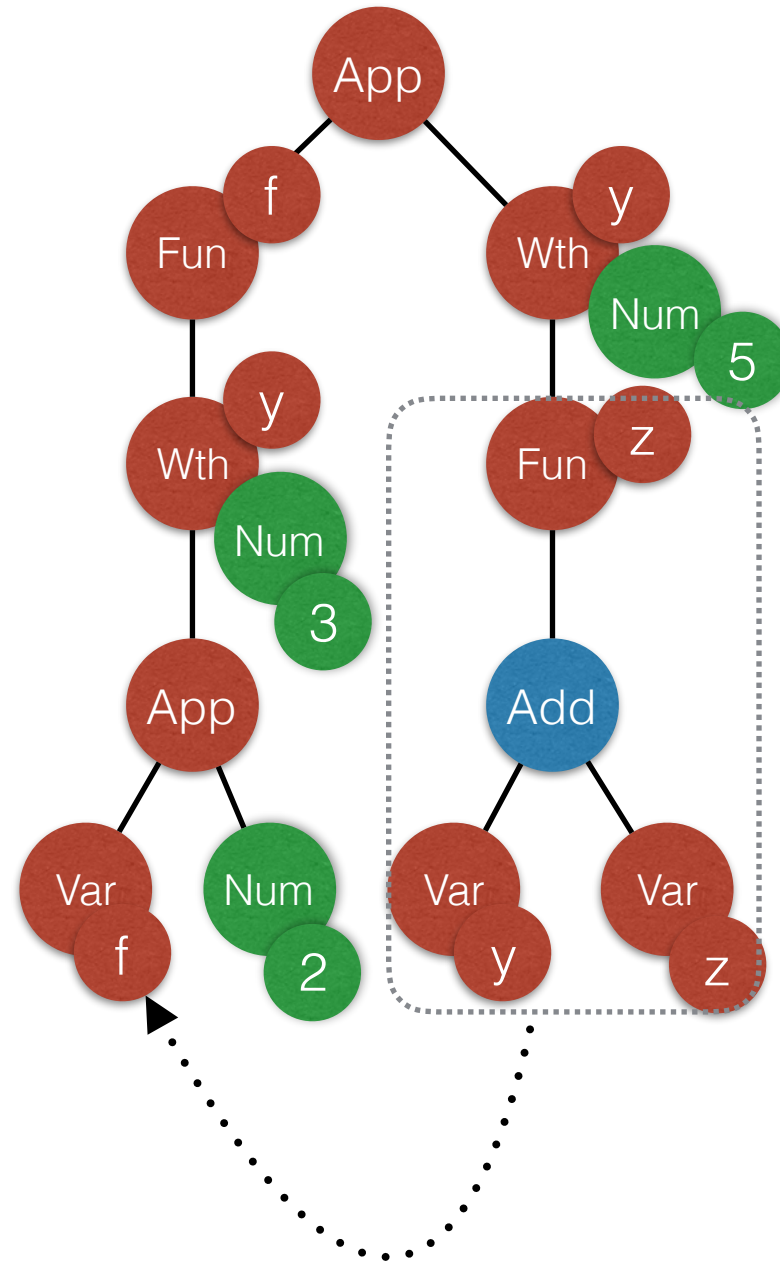
7

FAE

Problem: we then have an environment assignment:

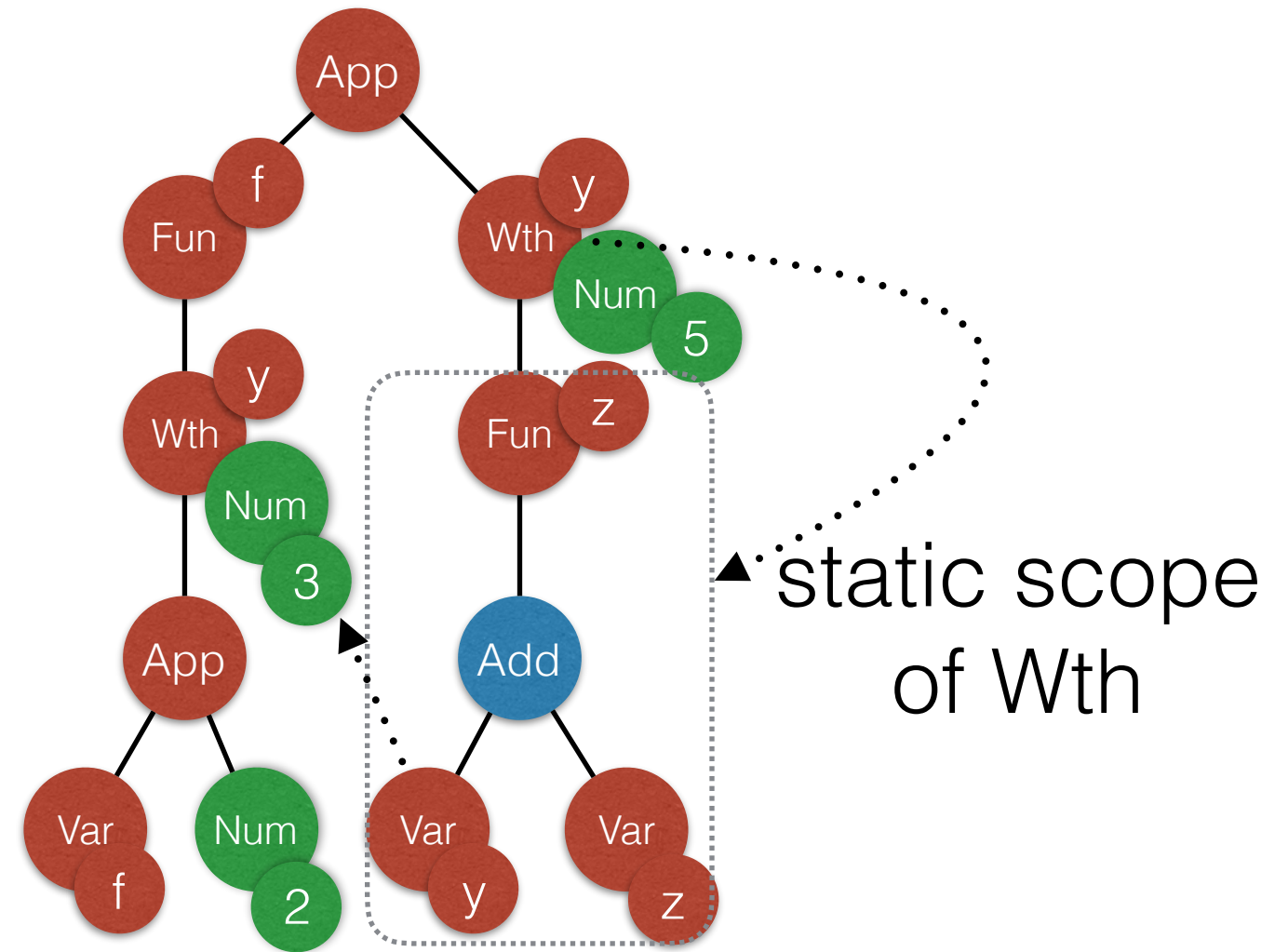
$f \longrightarrow \text{Fun}(z, \text{Add}(y, z))$

So this term gets inserted in place of f .



7

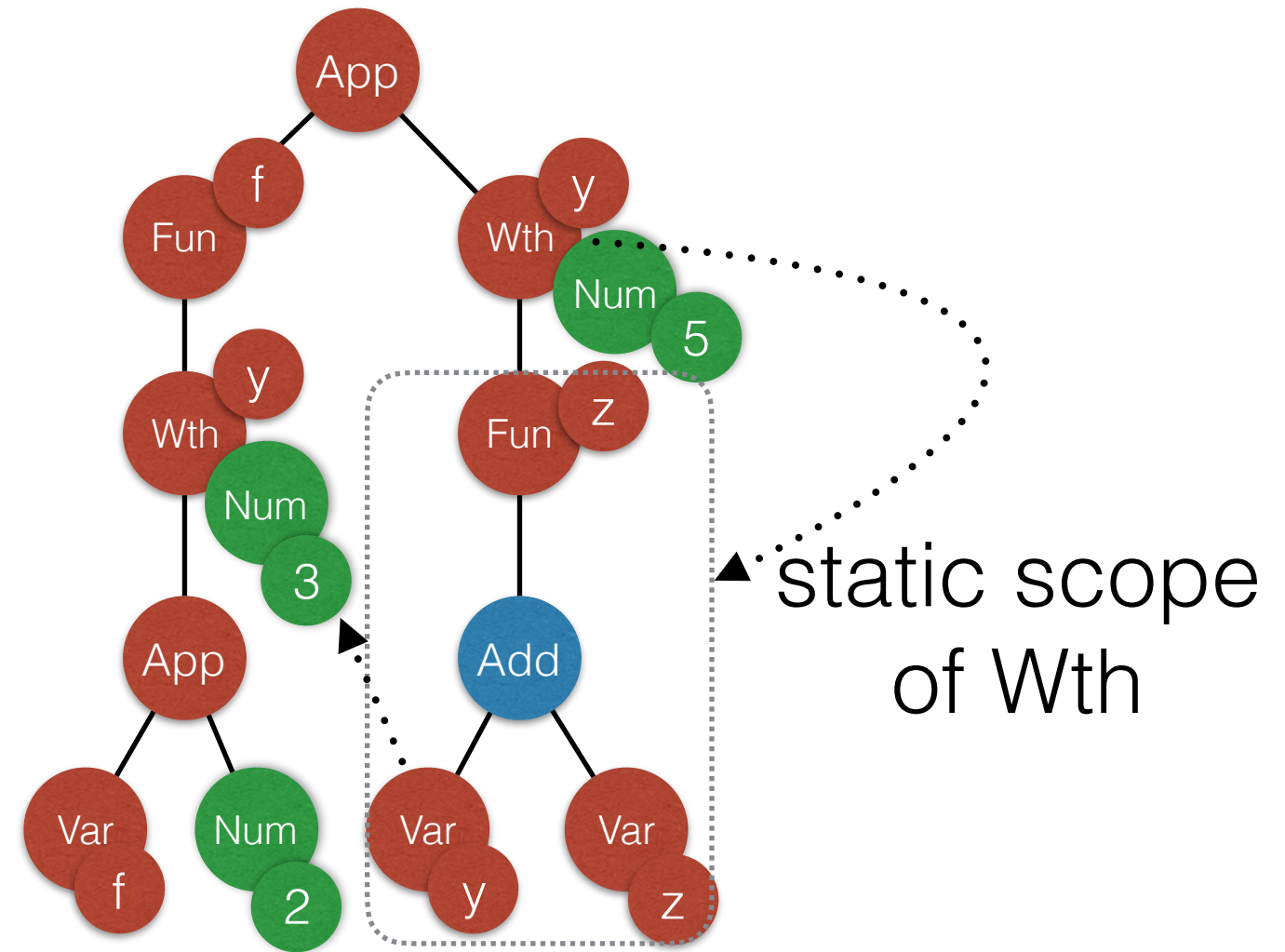
FAE



In effect, *y* is bound outside of the static scope.
=> We have a violation of static scoping!

7

FAE



Underlying problem:

The function term alone does not reflect **all** the knowledge of the evaluation.
Thus it does not really qualify as a value.

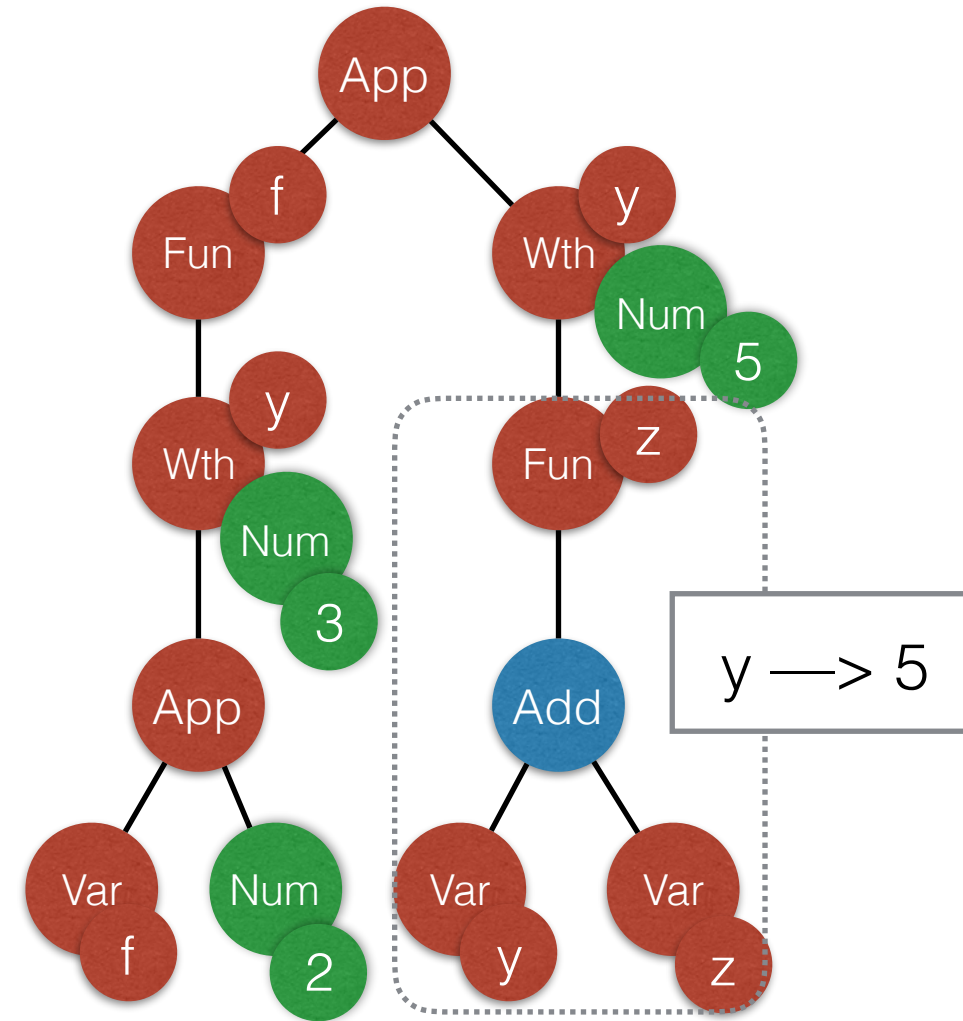
The remaining knowledge is in the environment!
With this in mind, the solution is rather straightforward:

7

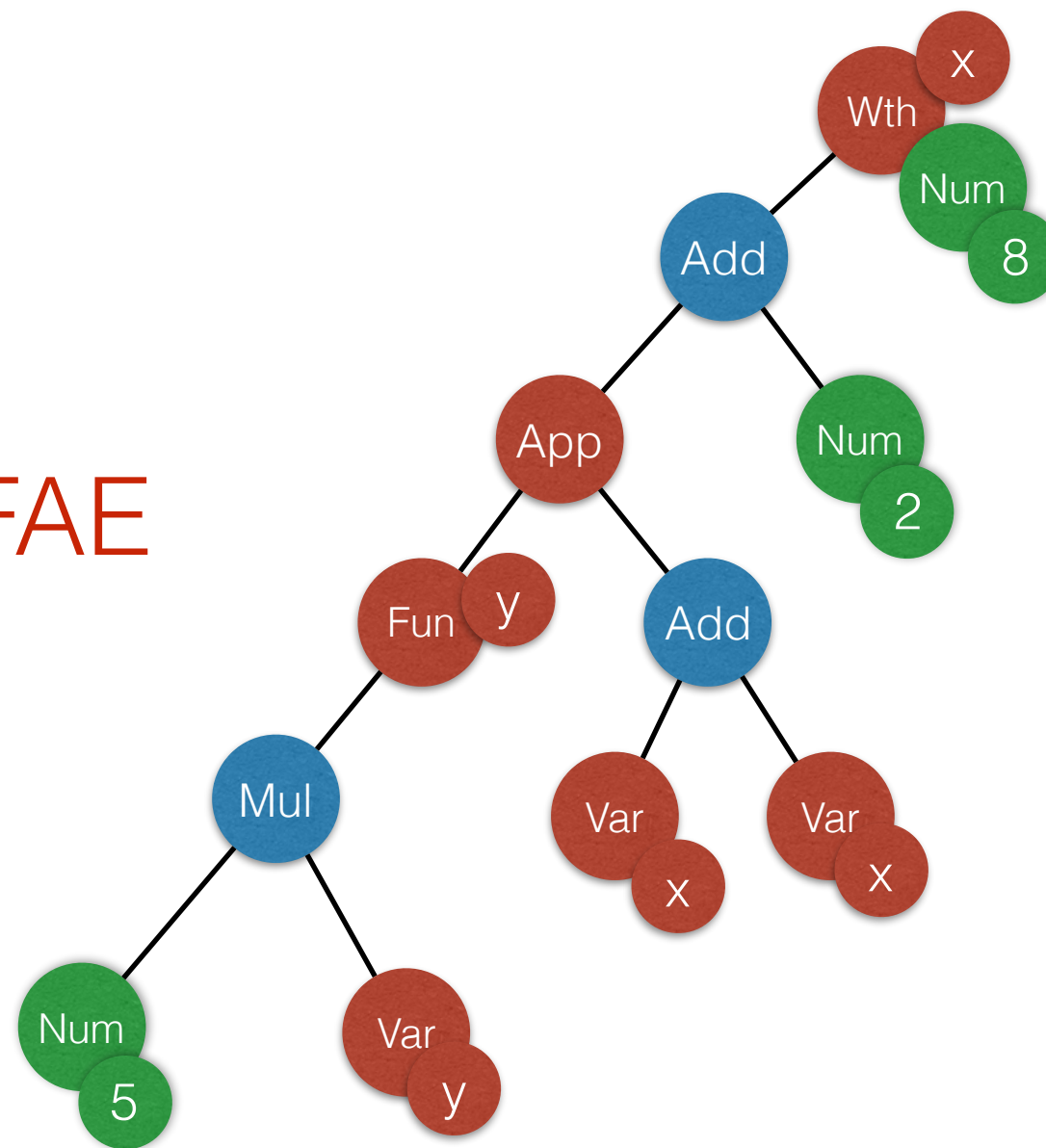
FAE

Solution: value = term
together with environment
at the time the term is
evaluated

$f \longrightarrow$
Closure(
Fun(z, Add(y, z)),
($y \longrightarrow 5$))



8 LCFAE

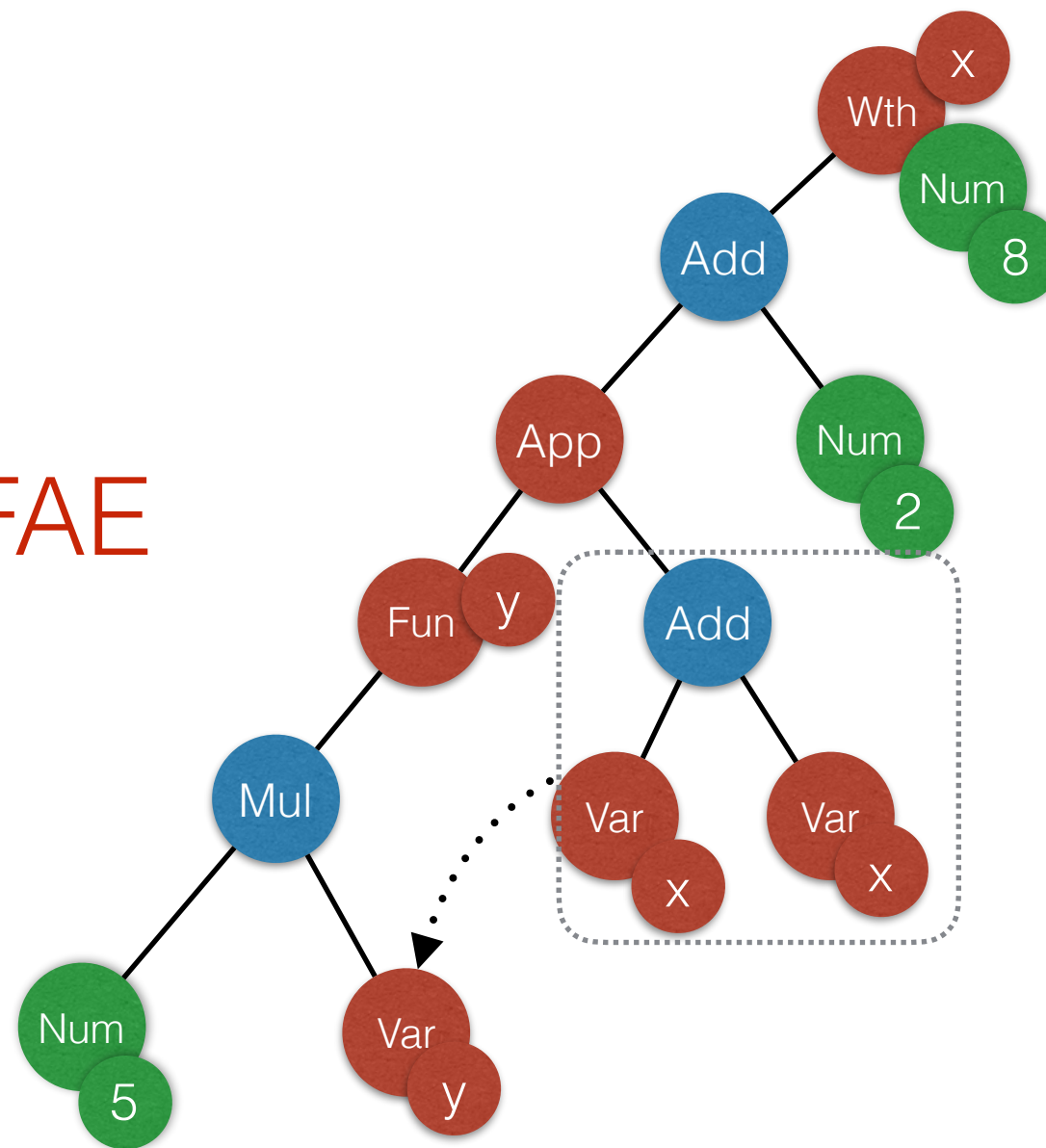


Fun

Same node types, but the Fun node is now evaluated differently, namely **call-by-name**

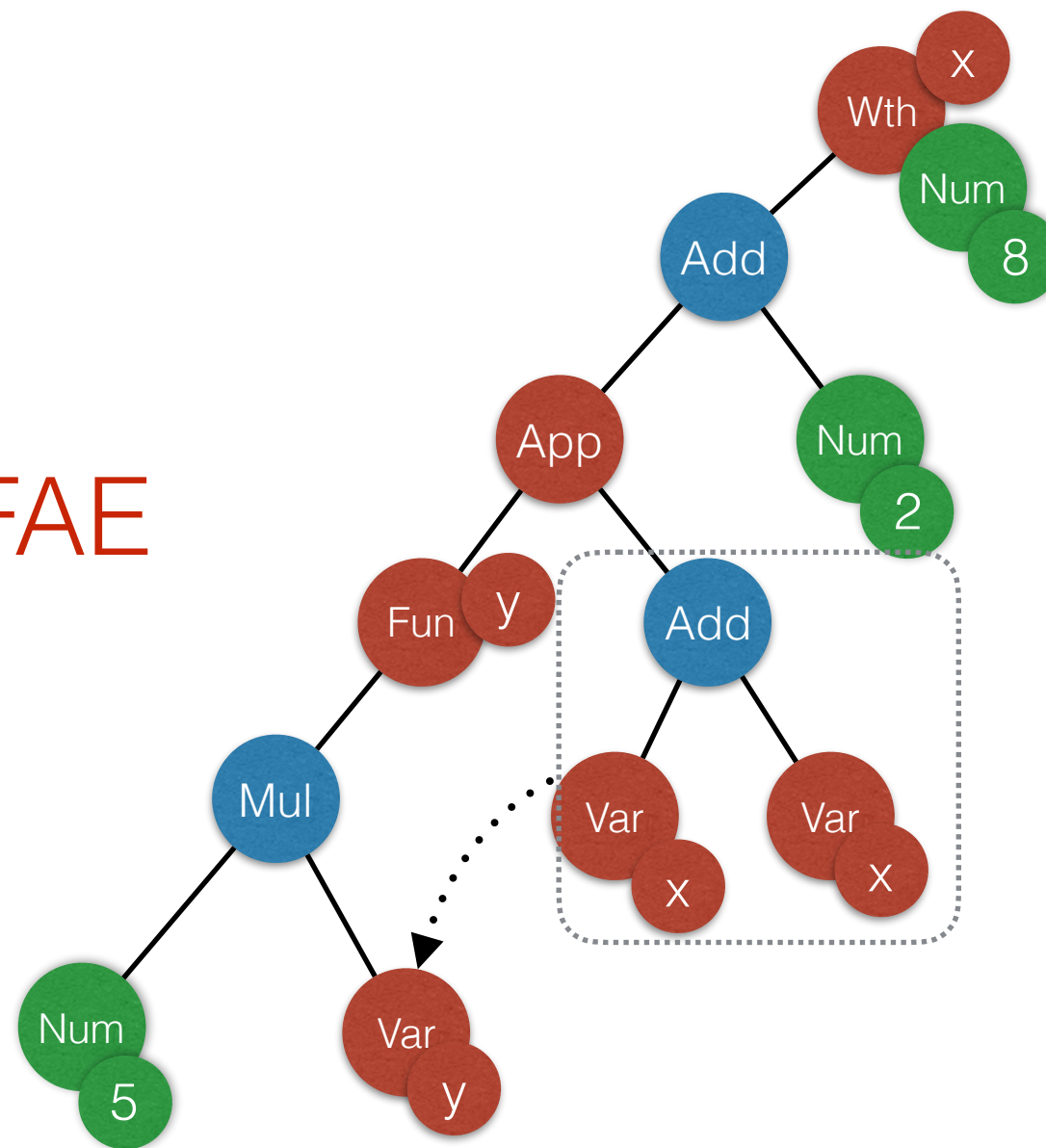
(or alternatively **call-by-need**, which is the same with caching)

8 LCFAE



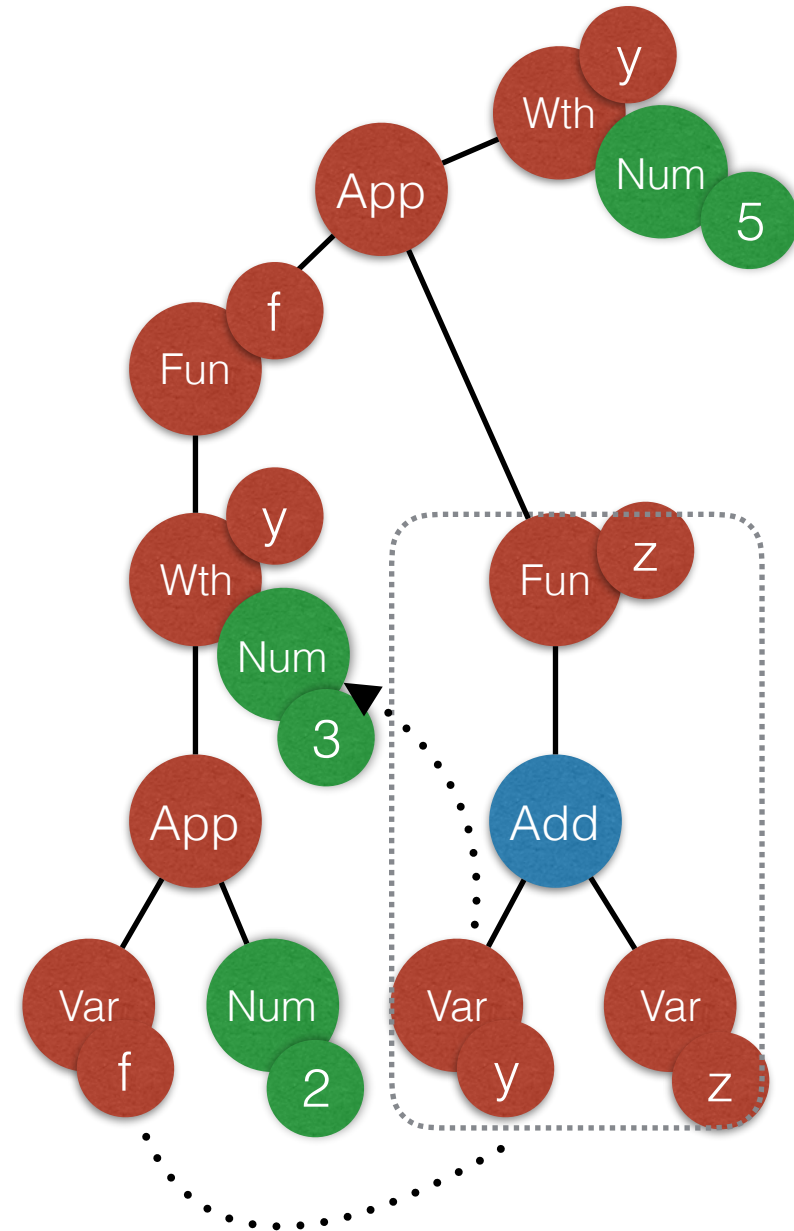
substitution-based interpreter:
the argument is not evaluated, but copied entirely

8 LCFAE



What does this mean for environment-based interpretation?

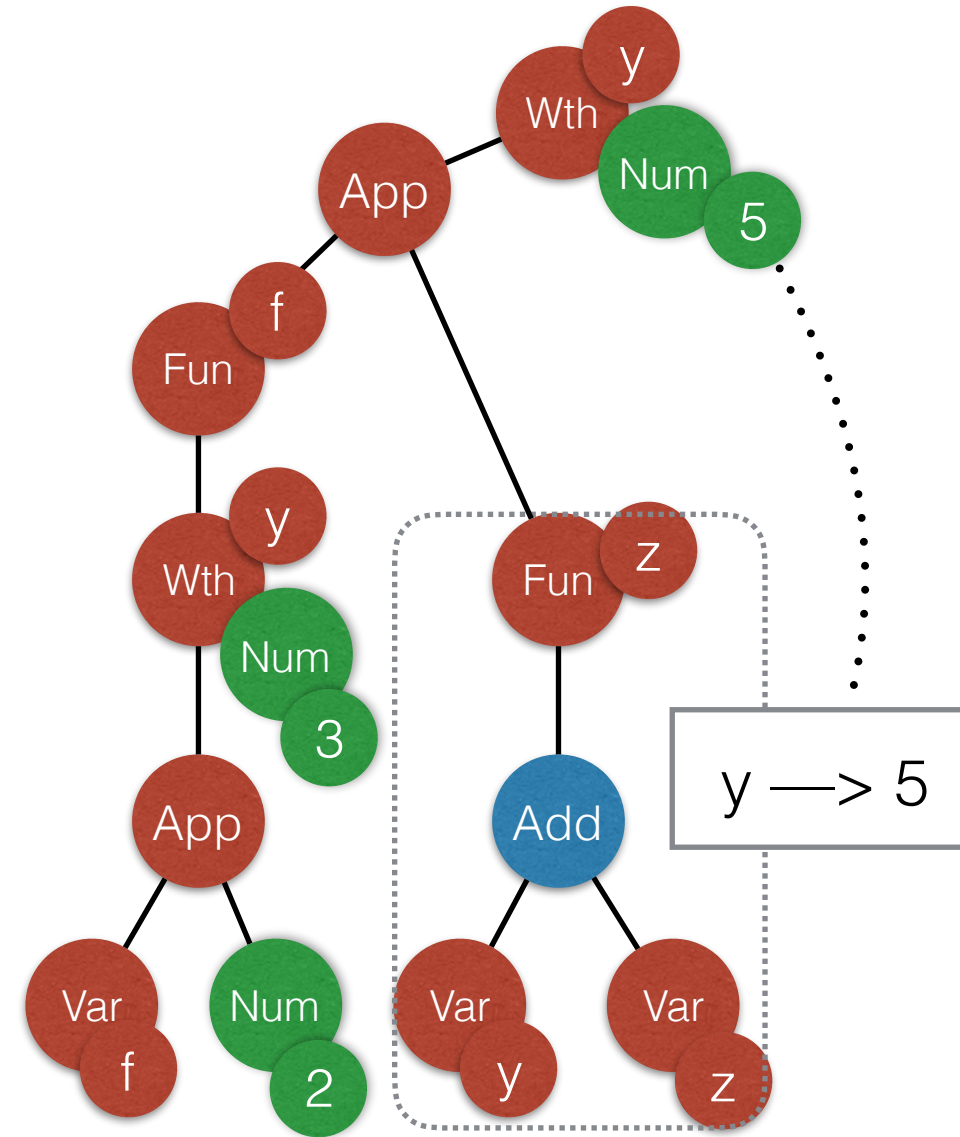
8 LCFAE



We can't just use *values* for the right-hand sides, because we don't evaluate unless necessary.

But when we just use terms, we run into a similar problem as with first-class functions under call-by-value, violating static scoping!

8 LCFAE



Again, the underlying problem is that we "threw away" the environment!

8 LCFAE

Solution: rhs = term together with environment at the time the term is copied

$f \longrightarrow$
Thunk
 Fun(z, Add(y, z)),
 (y \longrightarrow Num(5)))

