i116: Basic of Programming

10. Assignment calculator: virtual machine & compiler

Kazuhiro Ogata, Canh Minh Do

Roadmap

- Virtual machine for assignment calculator
- Compiler for assignment calculator

Virtual machine for assignment calculator

- The virtual machine needs to use an environment to handle variables occurring in expressions and assignments.
- A command load(*var*) is introduced, which moves the value associated with *var* in an environment to a stack.
- A command store(*var*) is introduced, which moves the value *val* located at the top of a stack to an environment, updating the value associating *var* in an environment with *val*.

Virtual machine for assignment calculator

```
from enum import *
class CName(Enum):
    PUSH = auto()
    LOAD = auto()
    STORE = auto()
    MONE = auto()
    …
```

```
def __str__(self):
    if self == CName.PUSH:
        return 'push‘
    elif self == CName.LOAD:
        return 'load'
    elif self == CName.STORE:
        return 'store'
    elif self == CName.MONE:
        return 'mone'
    …
```

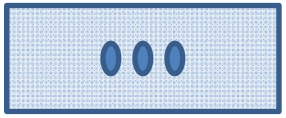
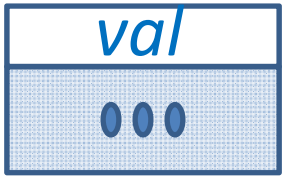
Adding two command names: LOAD and STORE

Virtual machine for an arithmetic calculator

```
class Command(object):
    ...
    name = ''
    def __init__(self, cn, x):
        ...
        elif cn == CName.LOAD:
            self.name = x
        elif cn == CName.STORE:
            self.name = x
    def __str__(self):
        ...
        elif self.cname == CName.LOAD:
            return str(self.cname) + '(' + str(self.name) + ')'
        elif self.cname == CName.STORE:
            return str(self.cname) + '(' + str(self.name) + ')'
        ...
```

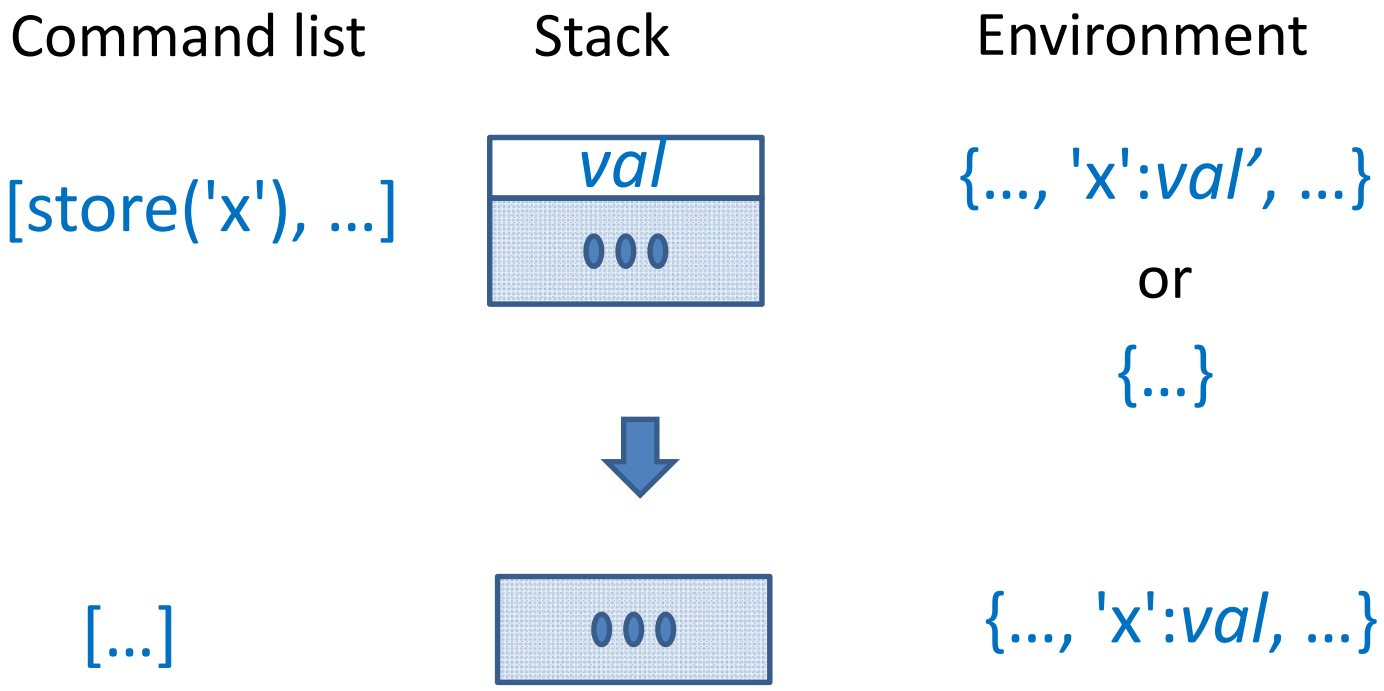
Adding two commands:
load('x') and store('x')

Virtual machine for an arithmetic calculator

Command list	Stack	Environment
[load('x'), …]		{…, 'x':*val*, …}
[…]		{…, 'x':*val*, …}

If 'x' is not registered in Environment, an exception called VMError is raised.

Virtual machine for an arithmetic calculator



If Stack is empty, an exception called VMError is raised.

Virtual machine for an arithmetic calculator

```
class VM(object):
    ...
    env = {}
    ...
    def str(self):
        return ...+ ', env: ' + str(self.env)
    def run(self):
        for com in self.clist:
            ...
            elif com.cname == CName.LOAD:
                try:
                    x = self.env[com.name]
                except KeyError:
                    raise UndefinedVar('undefined variable')
                self.stk = self.stk.push(x)
```

Virtual machine for an arithmetic calculator

```
        elif com.cname == CName.STORE:
            if self.stk.isEmpty():
                raise VMError('stk is empty for store')
            self.env[com.name] = self.stk.top()
            self.stk = self.stk.pop()
        ...
    return self.env
```

Virtual machine for an arithmetic calculator

[push(2), store(x), load(x), load(x), mul, store(x)]

{}

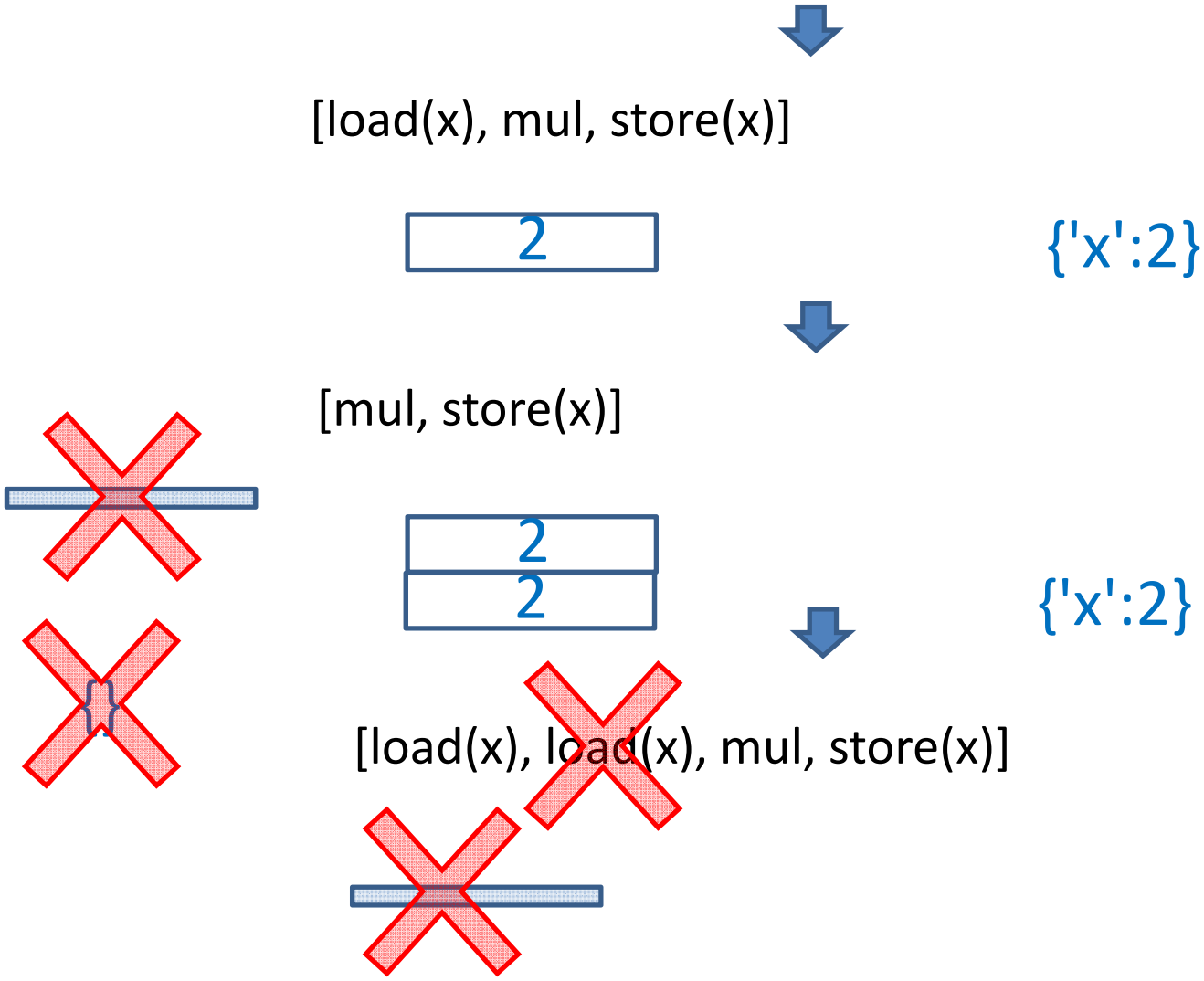
[store(x), load(x), load(x), mul, store(x)]

{}

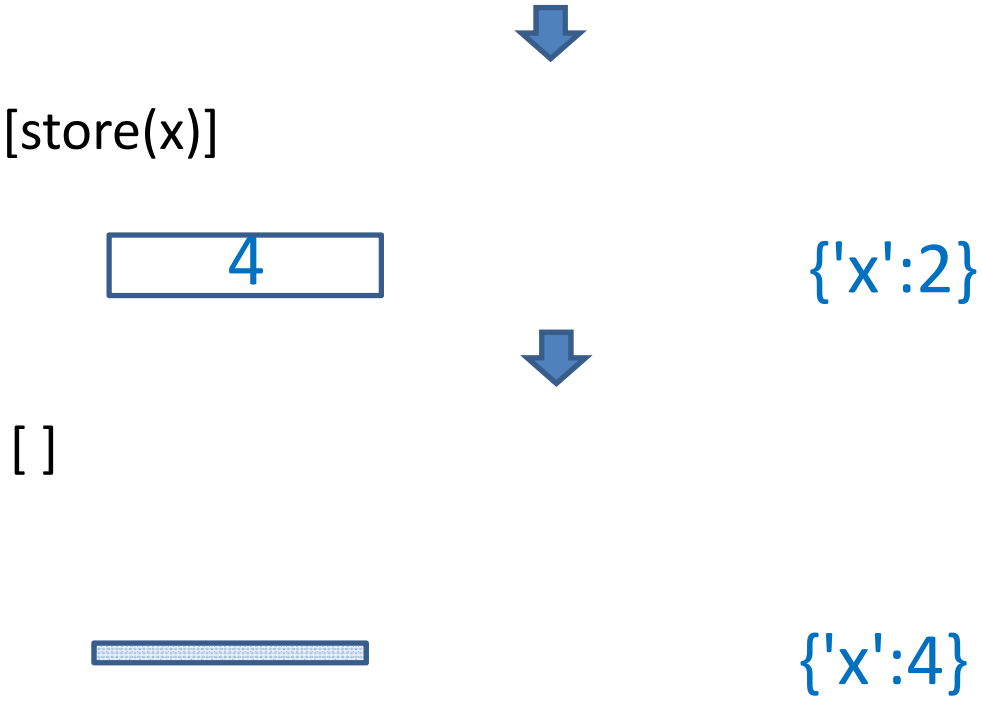
[load(x), load(x), mul, store(x)]

{'x':2}

Virtual machine for an arithmetic calculator



Virtual machine for an arithmetic calculator



{'x':4} is returned as the result of run() of VM.

Compiler for assignment calculator

- We should come up with how to compile
  - a variable appearance in the right-hand side of an assignment
  - an assignment
  - a sequential composition
- We can reuse how to compile numbers, addition, multiplication, etc.

Compiler for assignment calculator

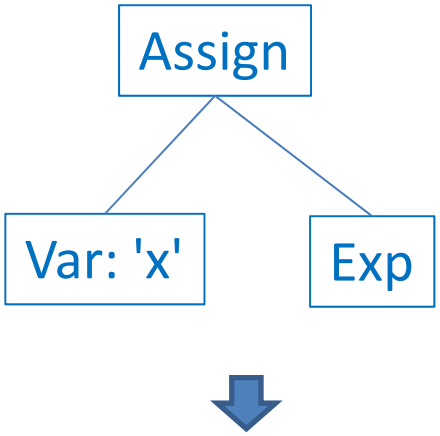
Suppose that `Var: 'x'` appears in the ~~left~~right-hand side of an assignment.

`Var: 'x'` ⇒ `[load('x')]`

Note that each compiler developed in this course just generates a list of commands but never evaluates what happens if the list is evaluated, which is done by the VM.

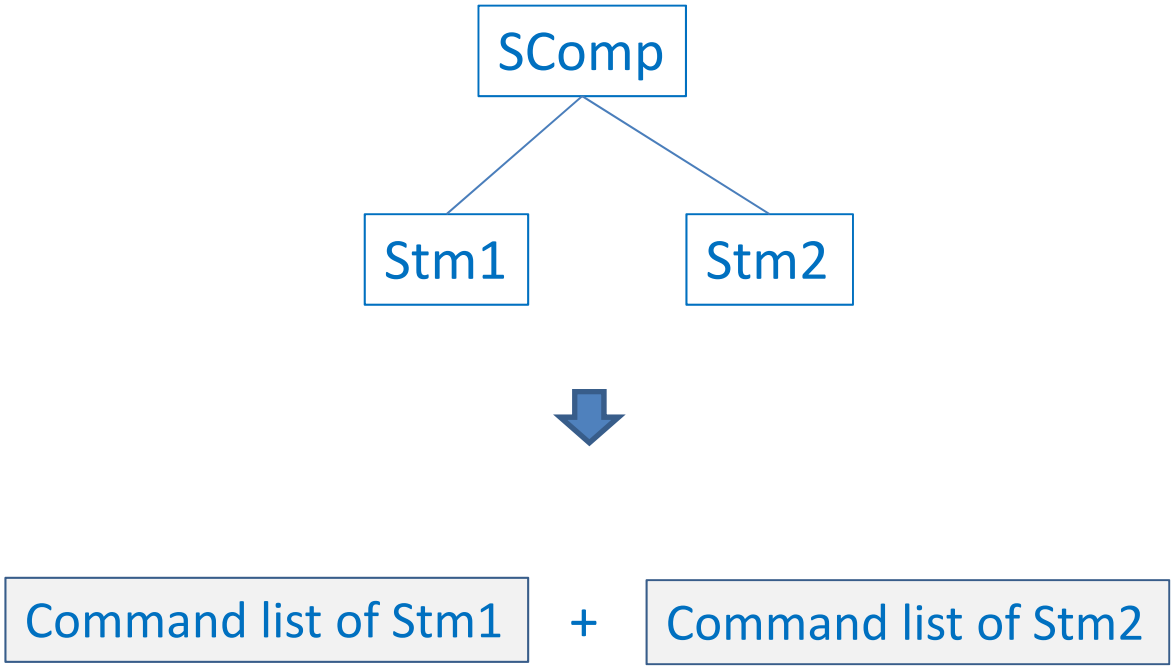
When load('x') is evaluated by the VM, an exception may be raised.

Compiler for assignment calculator



Command list of Exp + [store('x')]

Compiler for assignment calculator



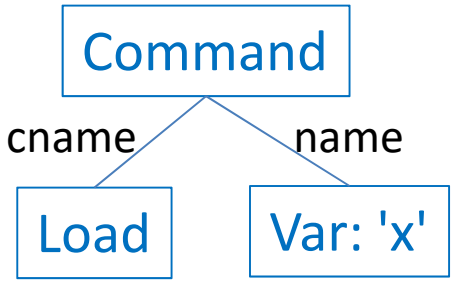
Compiler for assignment calculator

```
class VarParseTree(ExpParseTree):
    name = ''
    ...
    def compile(self):
        return [Command(CName.LOAD,self.name)]
```

Var: 'x' .compile()

[load('x')] is returned.

Precisely, [*aCommandObject*] is returned, where *aCommandObject* is



Compiler for assignment calculator

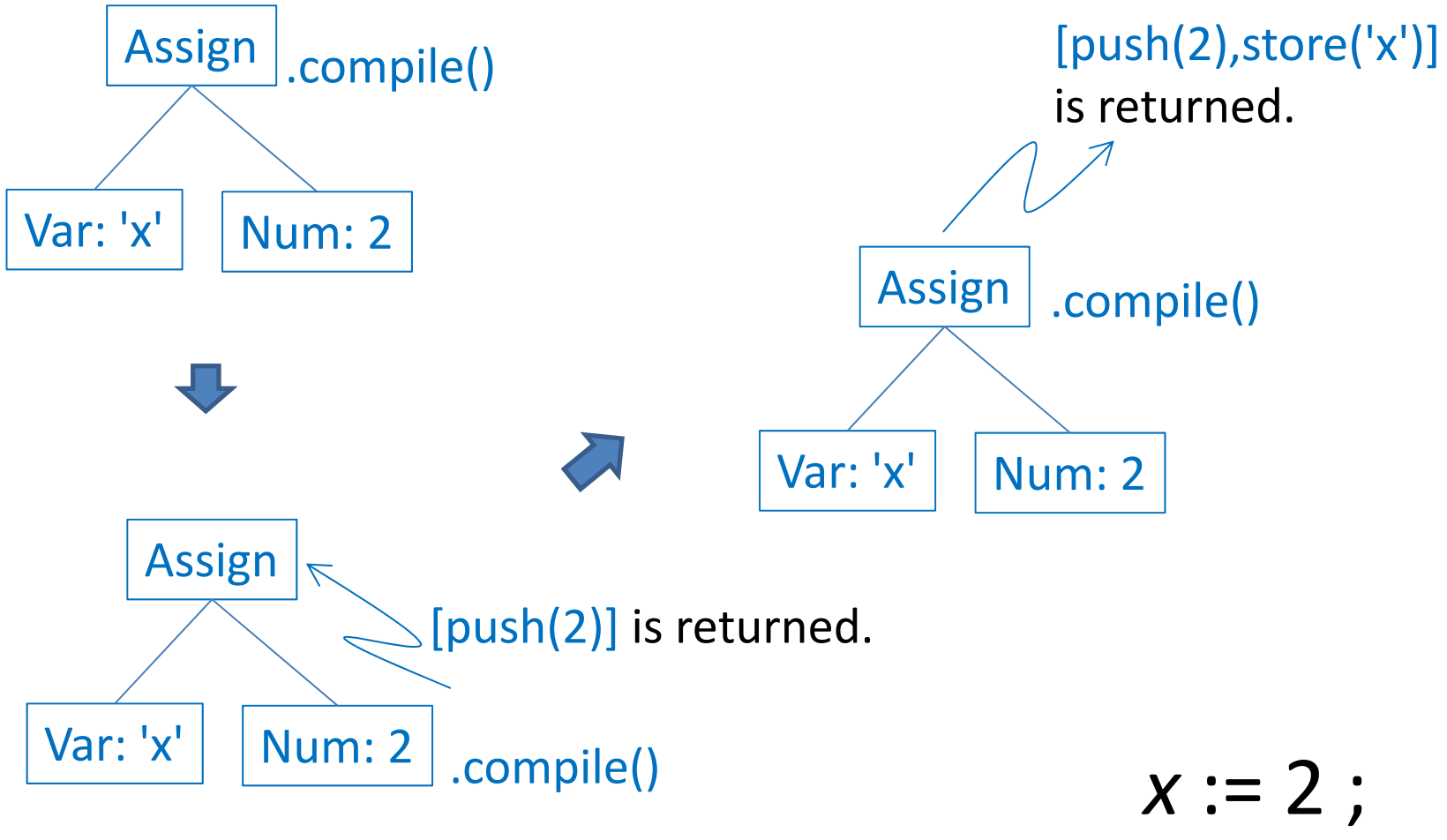
We should define method compile() for each class of statements.

```
class StmParseTree(object):
    ...
    def compile(self):
        pass
```

Compiler for assignment calculator

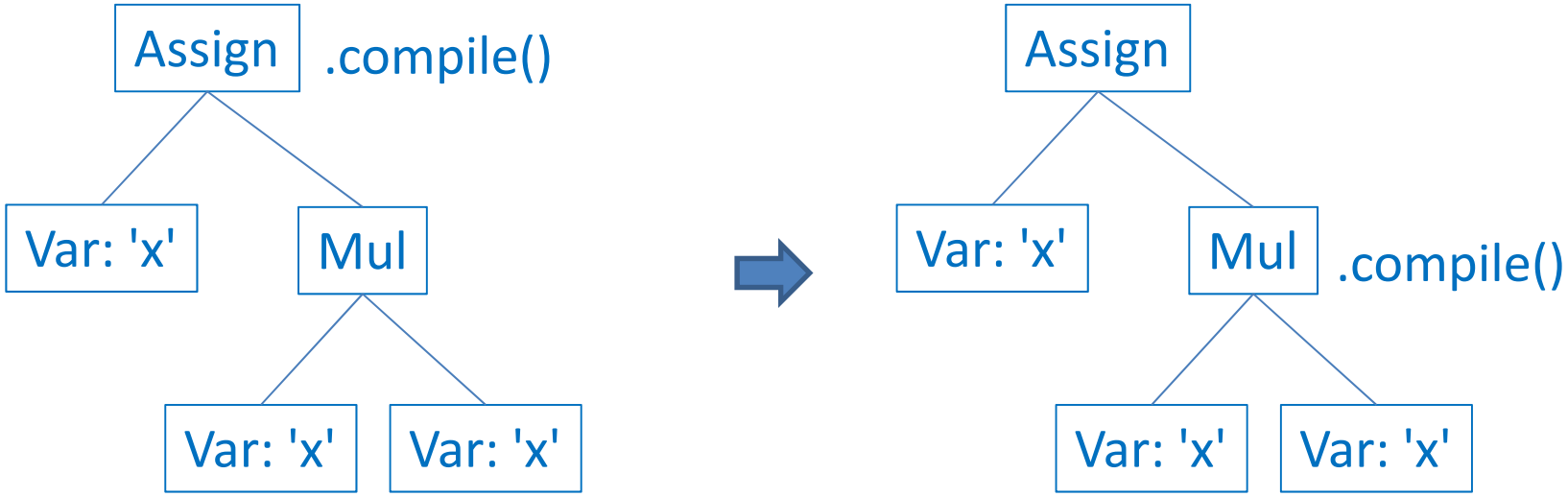
```
class AssignParseTree(StmParseTree):
    var = ExpParseTree()
    exp = ExpParseTree()
    ...
    def compile(self):
        cl1 = self.exp.compile()
        return cl1 + [Command(CName.STORE,self.var.name)]
```

Compiler for assignment calculator



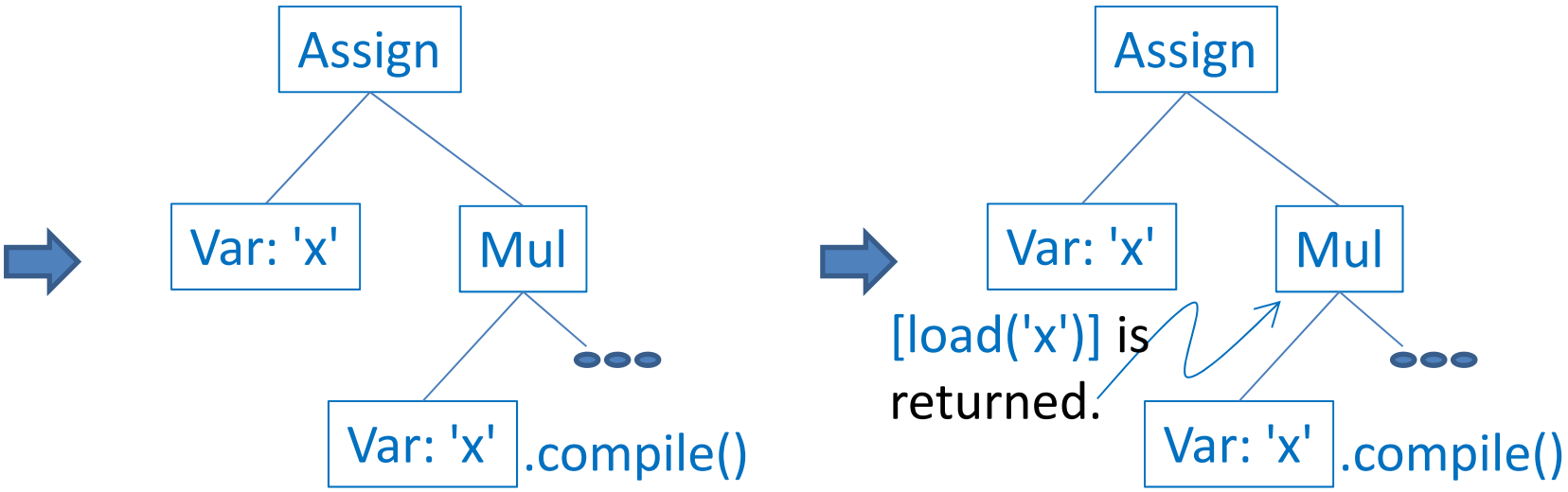
$x := 2 ;$

Compiler for assignment calculator

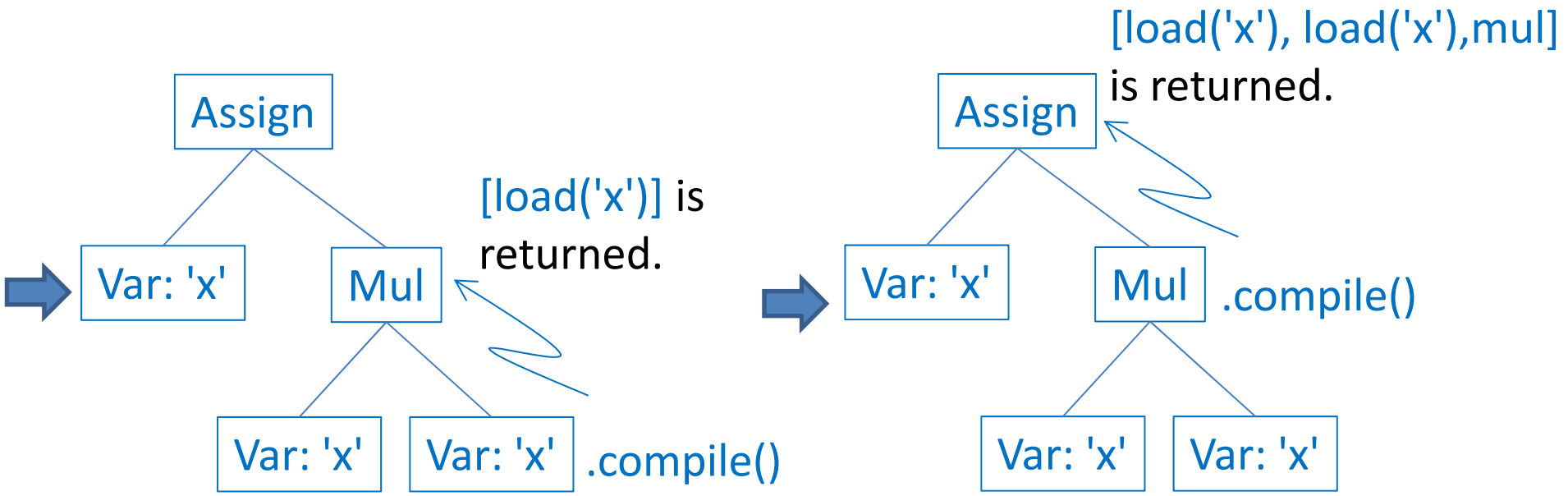


x := *x* * *x* ;

Compiler for assignment calculator

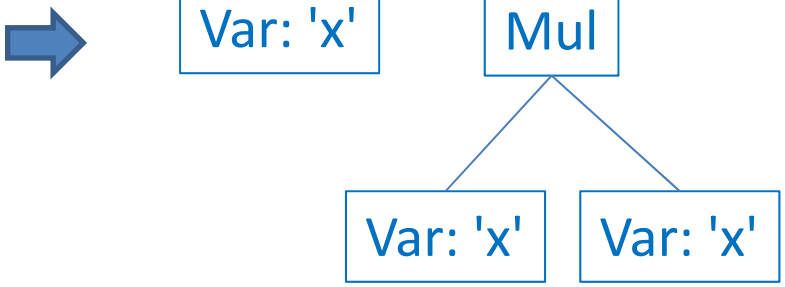


Compiler for assignment calculator



Compiler for assignment calculator

[load('x'), load('x'), mul, store('x')]
is returned.



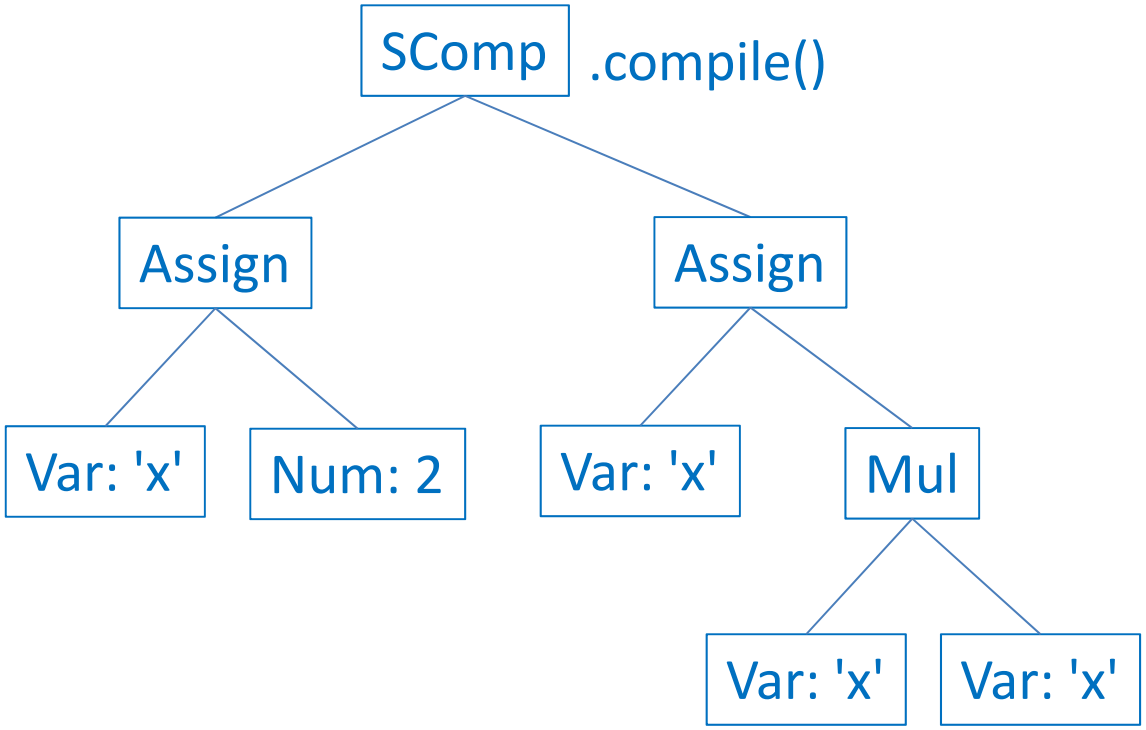
Compiler for assignment calculator

```
class AssignParseTree(StmParseTree):
    var = ExpParseTree()
    exp = ExpParseTree()
    ...
    def compile(self):
        cl1 = self.exp.compile()
        return cl1 + [Command(CName.STORE,self.var.name)]
```

Compiler for assignment calculator

```
class SCompParseTree(StmParseTree):
    stm1 = StmParseTree()
    stm2 = StmParseTree()
    ...
    def compile(self):
        cl1 = self.stm1.compile()
        cl2 = self.stm2.compile()
        return cl1 + cl2
```

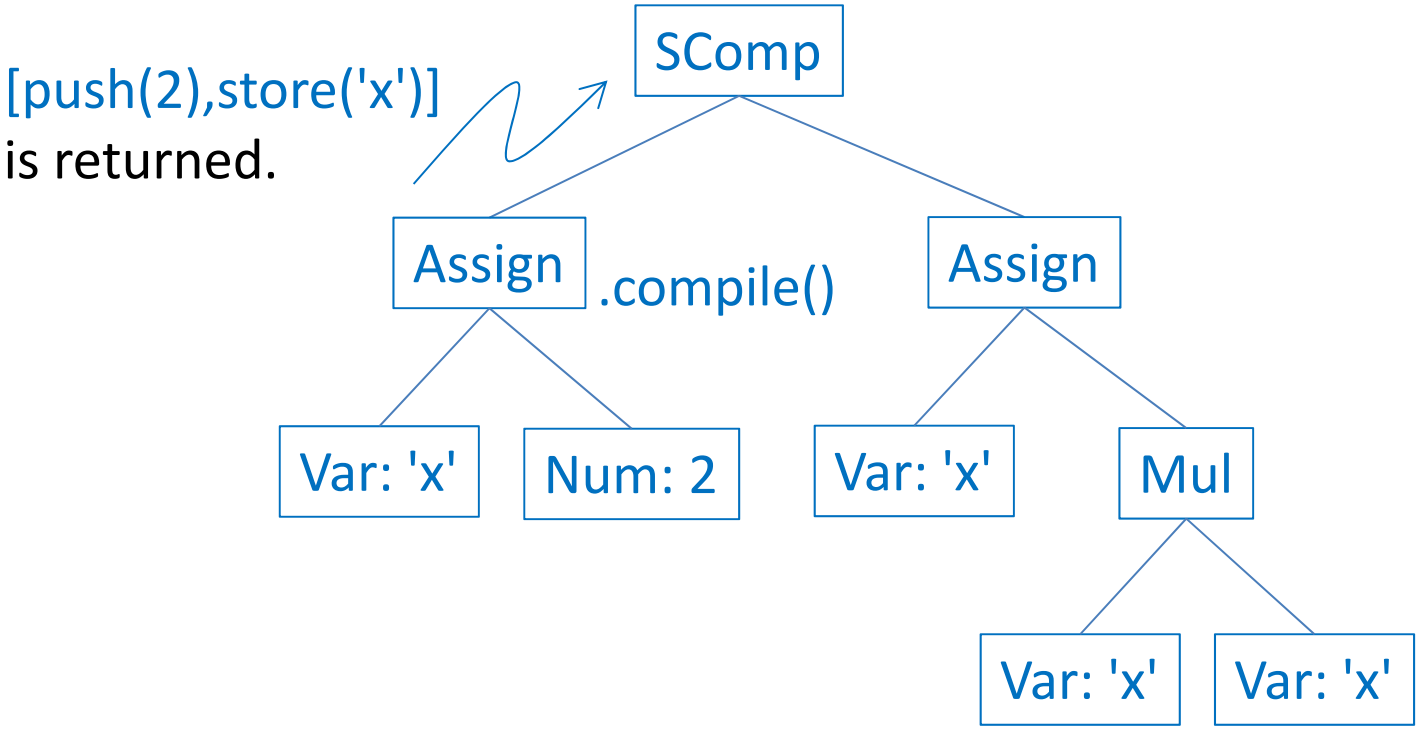
Compiler for assignment calculator



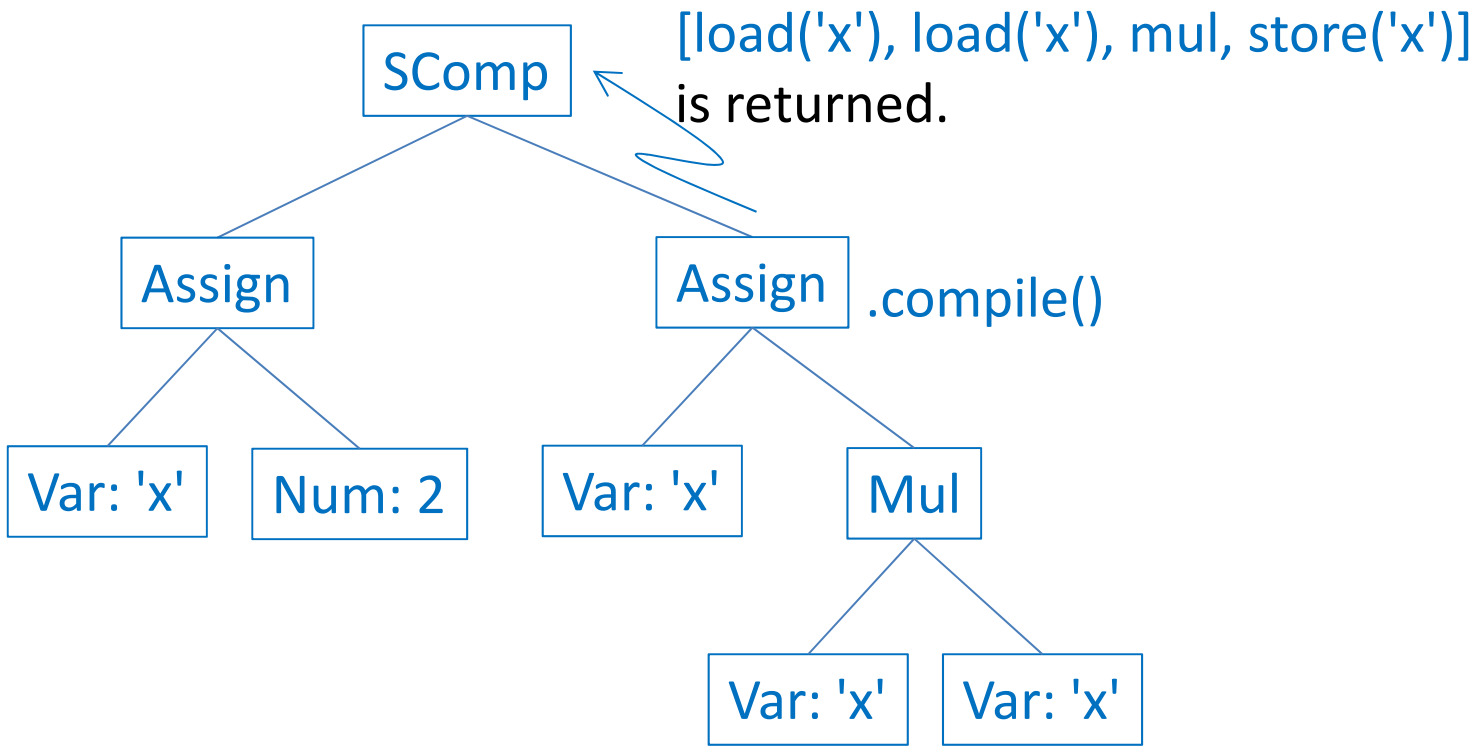
x := 2 ;

x := *x* * *x* ;

Compiler for assignment calculator



Compiler for assignment calculator



Compiler for assignment calculator

[push(2),store('x'), load('x'), load('x'), mul, store('x')]
is returned.