Semantic Analysis
Using
Semantic Actions,
Semantic Action Stack
&
Operator Stack

Semantic Analysis

Semantic Actions are added to the grammar to indicate where a Semantic Routine is to be executed.

Each grammar symbol terminal or non-terminal my have zero or more associated Semantic Actions.

Semantic Routines are placed in a parser to help perform Semantic Analysis.

Semantic Routines communicate via Semantic Action Records.

Semantic Action Records (SAR) contain information needed by Semantic Routines to perform Semantic Analysis.

The Semantic Action Stack (SAS) is used to store and pass Semantic Records between Semantic Routines.

Operator Precedence

Infix expressions must be converted to Postfix expressions to correctly evaluate.

Postfix expression

No Parenthesis

Enforce Operator Precedence

Shuntyard Algorithm used to convert Infix to Postfix

J = I + K * E;

J I K E * + =

Hint: Semantic Analysis (2 Pass) is only done after the first pass has already loaded the Symbol table for the entire program.

Hint: Focus on how nested or complex operations are always converted to simpler generally binary operations.

```
x+y+z+g ≡ t1=x+y
          t2=t1+z
          t3=t2+g


x.y.z.g ≡ t1=x.y
          t2=t1.z
          t3=t2.g
```

Hint: Be sure you understand how #rExists and #iExists work before adding lots of complexity to your Compiler. A common mistake is to write naïve versions of these critical semantic functions and then try to compensate by adding complexity all over the Compiler.

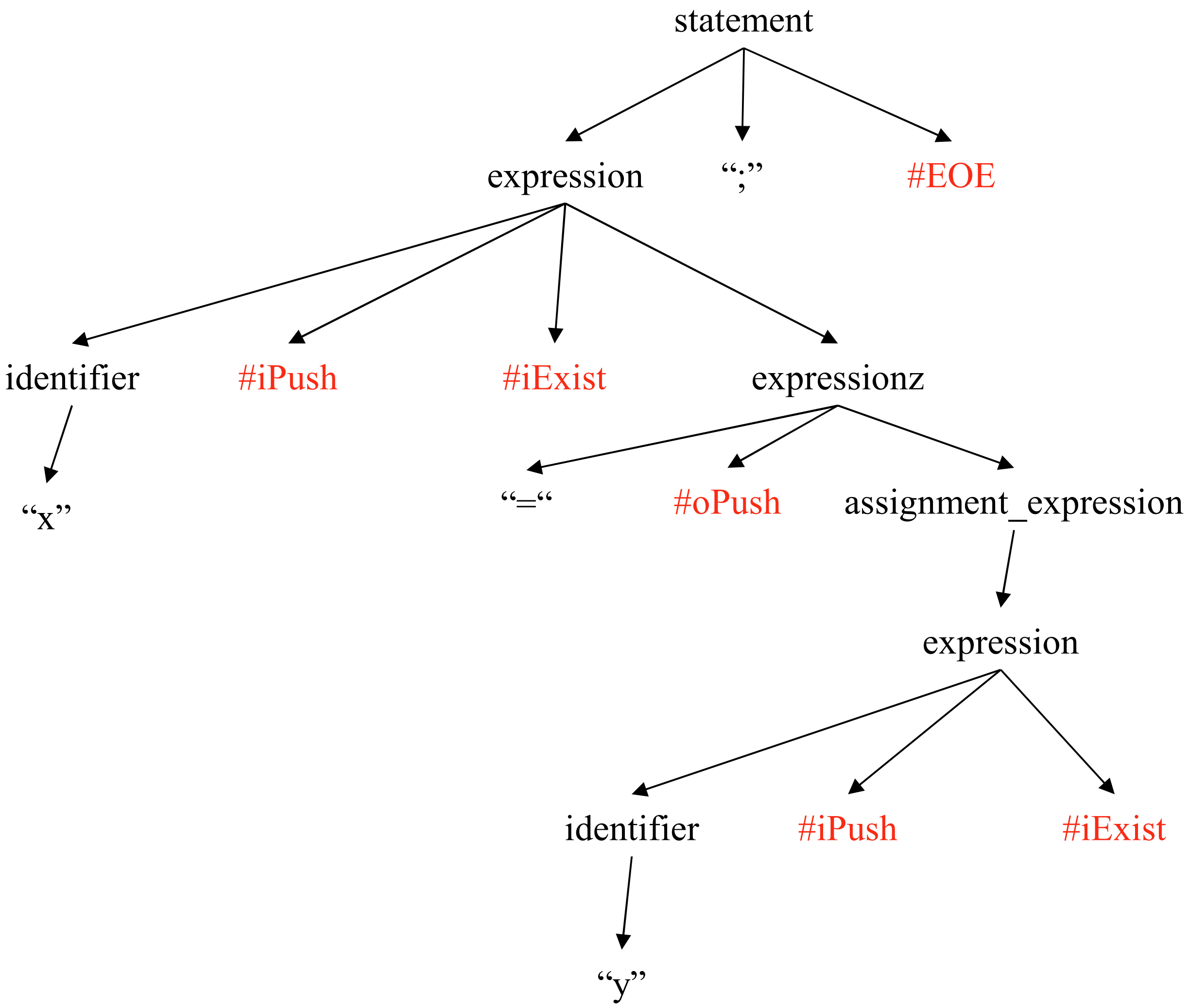
```
x = y ;
```

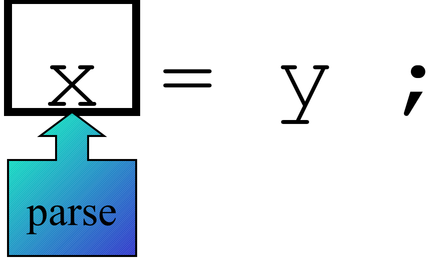
Simple Assignment Statement

Operator Stack

Semantic Action Stack

```
x = y ;
```


statement
expression
“;”
#EOE
identifier
#iPush
#iExist
expressionz
“x”
“=“
#oPush
assignment_expression
expression
identifier
#iPush
#iExist
“y”



```
x = y ;
```

parse

x - #iPush

Operator Stack

Semantic Action Stack

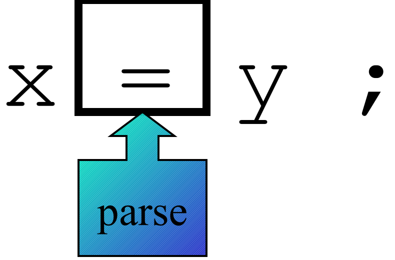
x $\boxed{=}$ y ;

Note: The Semantic Routine for #iExist creates an id_sar for x, however most example will simply show this as:
x - #iExist

x id_sar - #iExist

Operator Stack

Semantic Action Stack



```
x = y ;
```

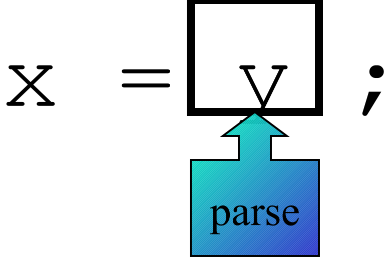
parse

= - #oPush

Operator Stack

x id_sar - #iExist

Semantic Action Stack



x = y ;

parse

= - #oPush

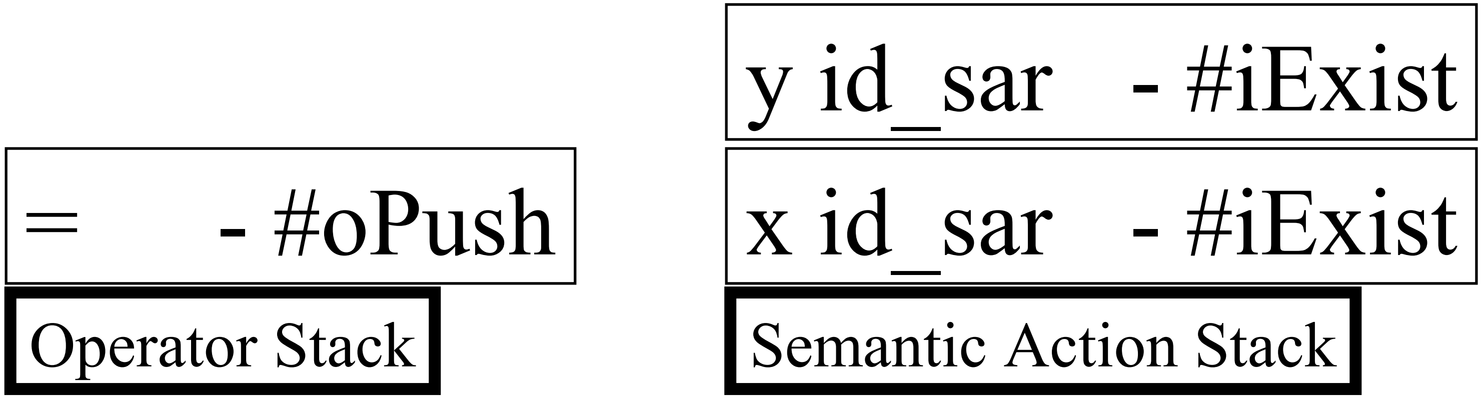
Operator Stack

y - #iPush

x id_sar - #iExist

Semantic Action Stack

x = y $\boxed{;}$



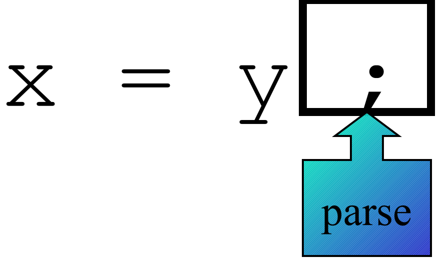
y id_sar - #iExist

= - #oPush

x id_sar - #iExist

Operator Stack

Semantic Action Stack



- #EOE

pop = #=

y id_sar = SAS.pop()

x id_sar = SAS.pop()

Test x = y is valid

Operator Stack

Semantic Action Stack

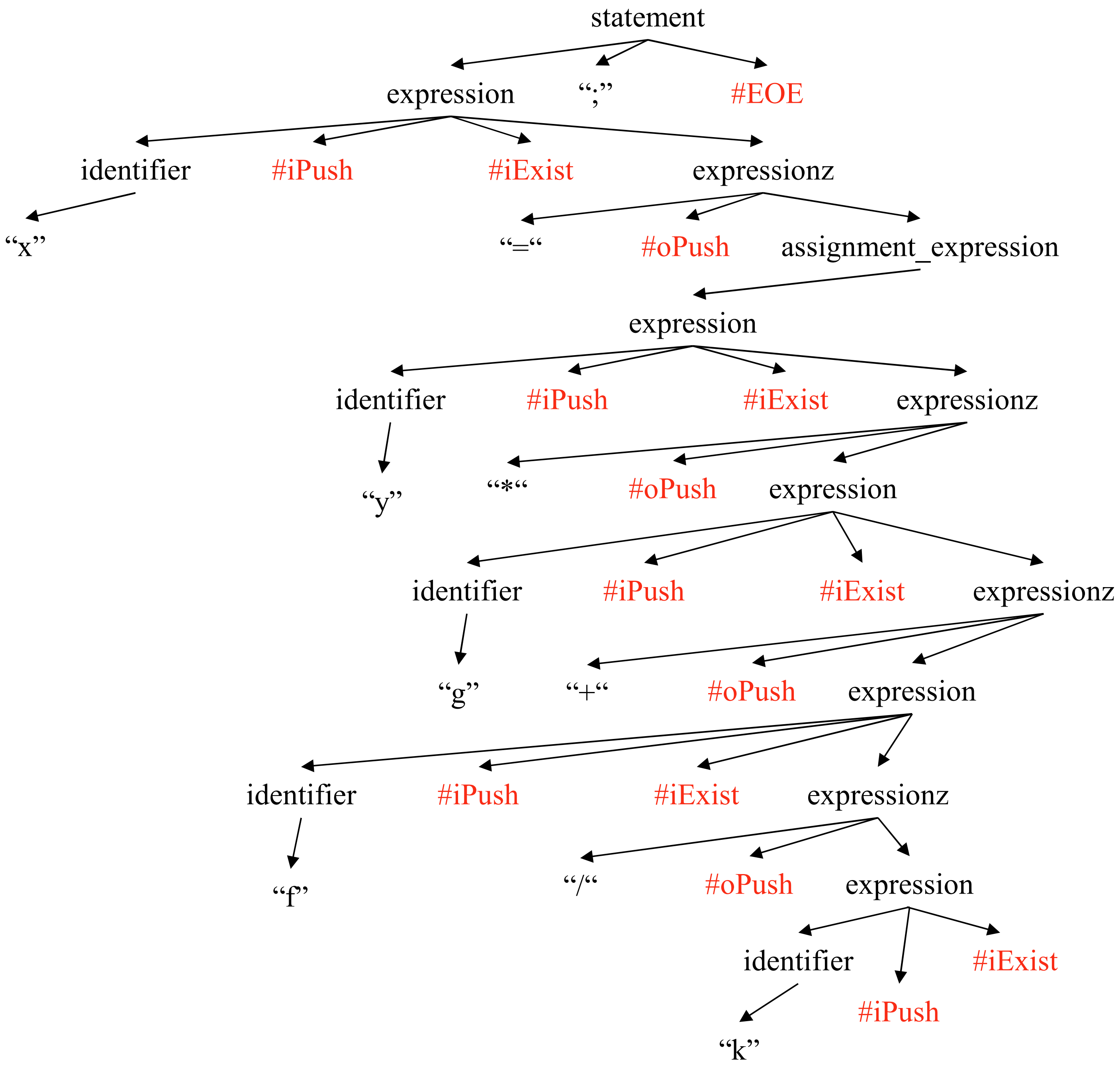
```
x = y * g + f / k ;
```

Assignment Statement with infix to Postfix Conversion

Operator Stack

Semantic Action Stack

`x = y * g + f / k ;`



statement

expression “;” #EOE

identifier #iPush #iExist expressionz

“x”

“=“ #oPush assignment_expression

expression

identifier #iPush #iExist expressionz

“y”

“*“ #oPush expression

identifier #iPush #iExist expressionz

“g”

“+“ #oPush expression

identifier #iPush #iExist expressionz

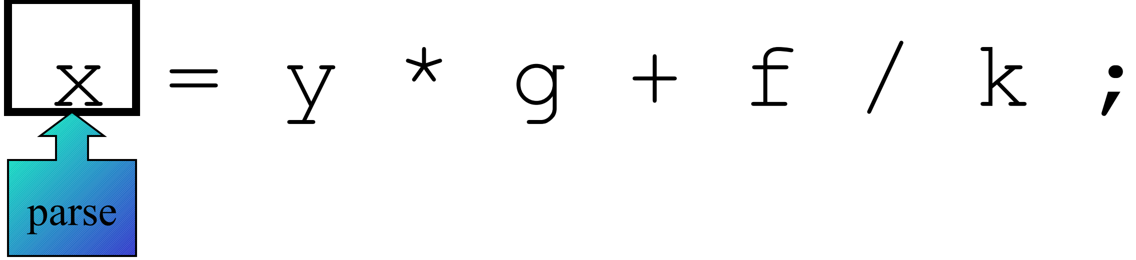
“f”

“/“ #oPush expression

identifier #iExist

#iPush

“k”



```
x = y * g + f / k ;
```

x - #iPush

Operator Stack

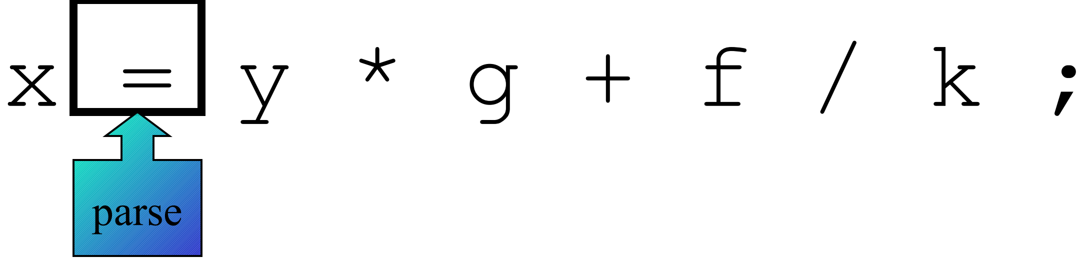
Semantic Action Stack

$x \boxed{=} y * g + f / k ;$

x - #iExist

Operator Stack

Semantic Action Stack



x = y * g + f / k ;

parse

= - #oPush

Operator Stack

x - #iExist

Semantic Action Stack

```
x = y * g + f / k ;
```

parse

=   - #oPush

Operator Stack

y   - #iPush

x   - #iExist

Semantic Action Stack

x = y $\boxed{*}$ g + f / k ;

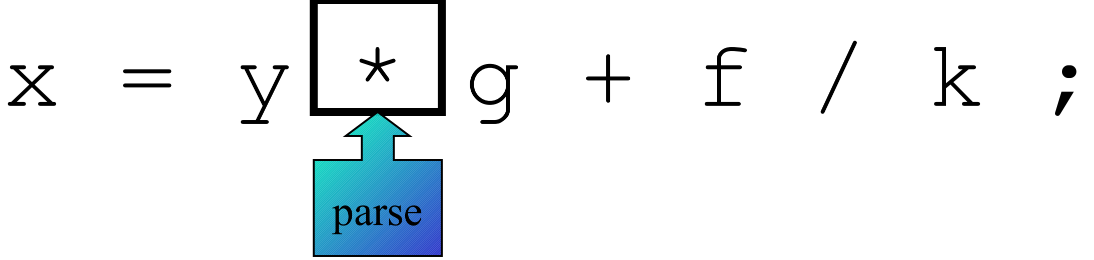
= - #oPush

Operator Stack

y - #iExist

x - #iExist

Semantic Action Stack



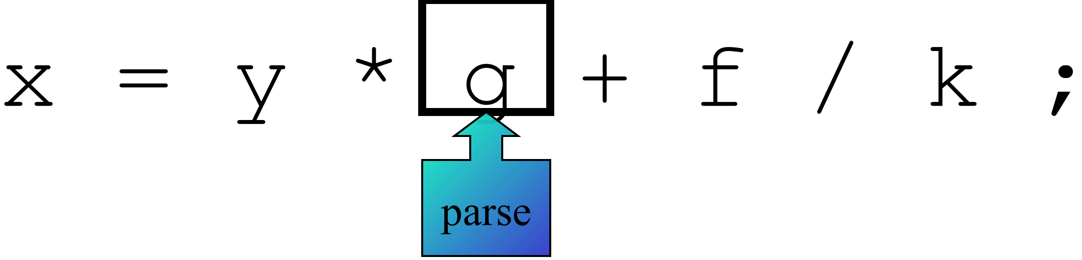
```
x = y * g + f / k ;
```

*	- #oPush
=	- #oPush

Operator Stack

y	- #iExist
x	- #iExist

Semantic Action Stack



```
x = y * g + f / k ;
```

* - #oPush

= - #oPush

Operator Stack

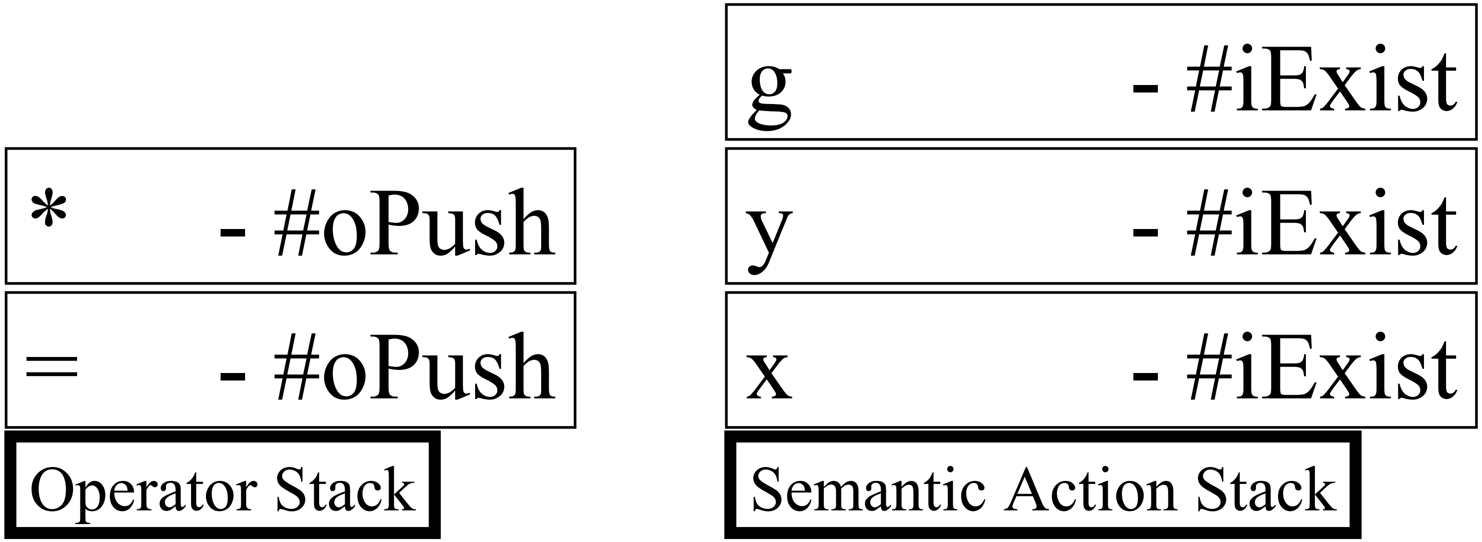
g - #iPush

y - #iExist

x - #iExist

Semantic Action Stack

$x = y * g \boxed{+} f / k ;$



* - #oPush

= - #oPush

Operator Stack

g - #iExist

y - #iExist

x - #iExist

Semantic Action Stack

```
x = y * g + f / k ;
```

parse

pop *   - #*

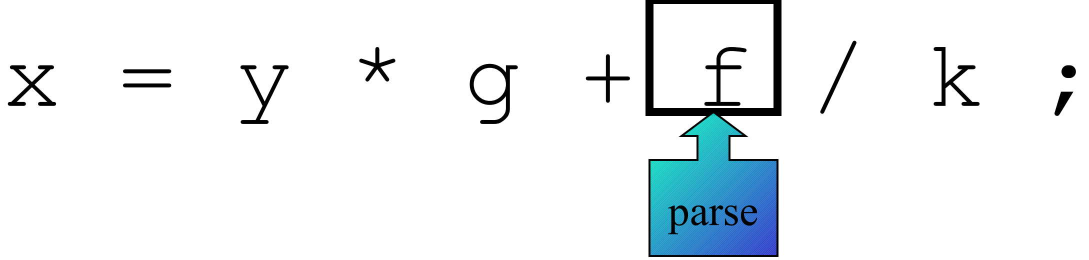
g = SAS.pop()
y = SAS.pop()
Test y * g is valid
t1 = y * g
SAS.push(t1)

+	- #oPush
=	- #oPush

Operator Stack

t1	- #*
x	- #iExist

Semantic Action Stack



x = y * g + f / k ;

parse

+ - #oPush

= - #oPush

Operator Stack

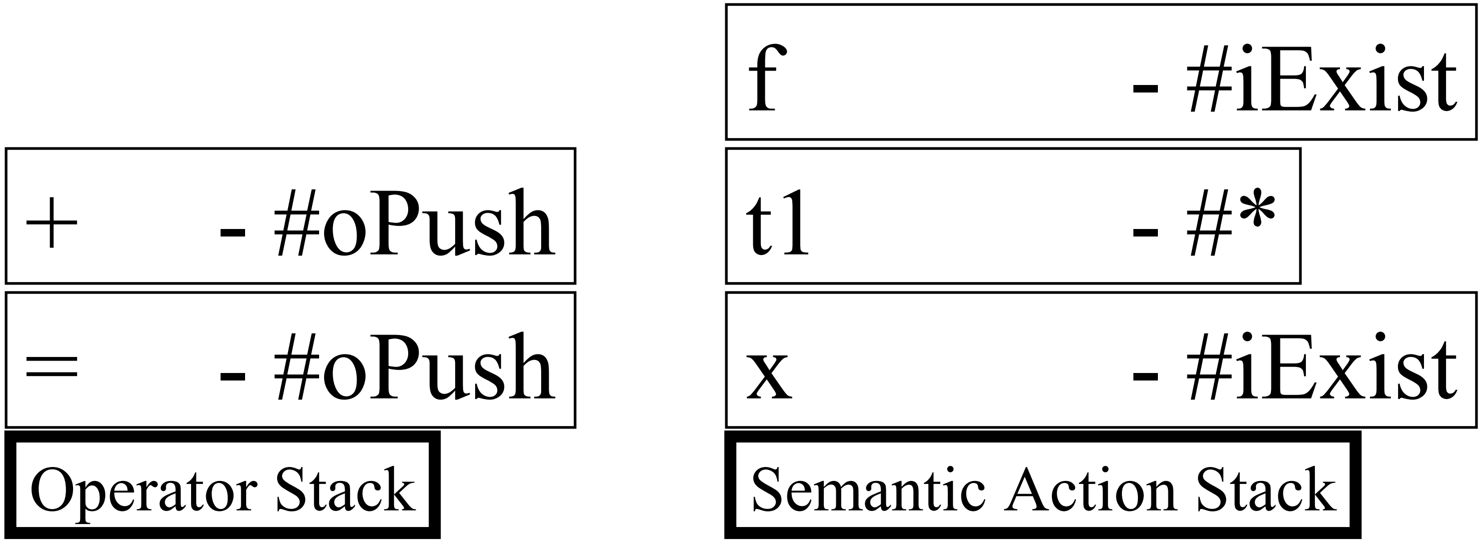
f - #iPush

t1 - #*

x - #iPush

Semantic Action Stack

x = y * g + f $\boxed{/}$ k ;



+ - #oPush

= - #oPush

Operator Stack

f - #iExist

t1 - #*

x - #iExist

Semantic Action Stack

```
x = y * g + f / k ;
```

parse

/	- #oPush
+	- #oPush
=	- #oPush

Operator Stack

f	- #iExist
t1	- #*
x	- #iExist

Semantic Action Stack

```
x = y * g + f / k ;
```

parse

/ - #oPush

+ - #oPush

= - #oPush

Operator Stack

k - #iPush

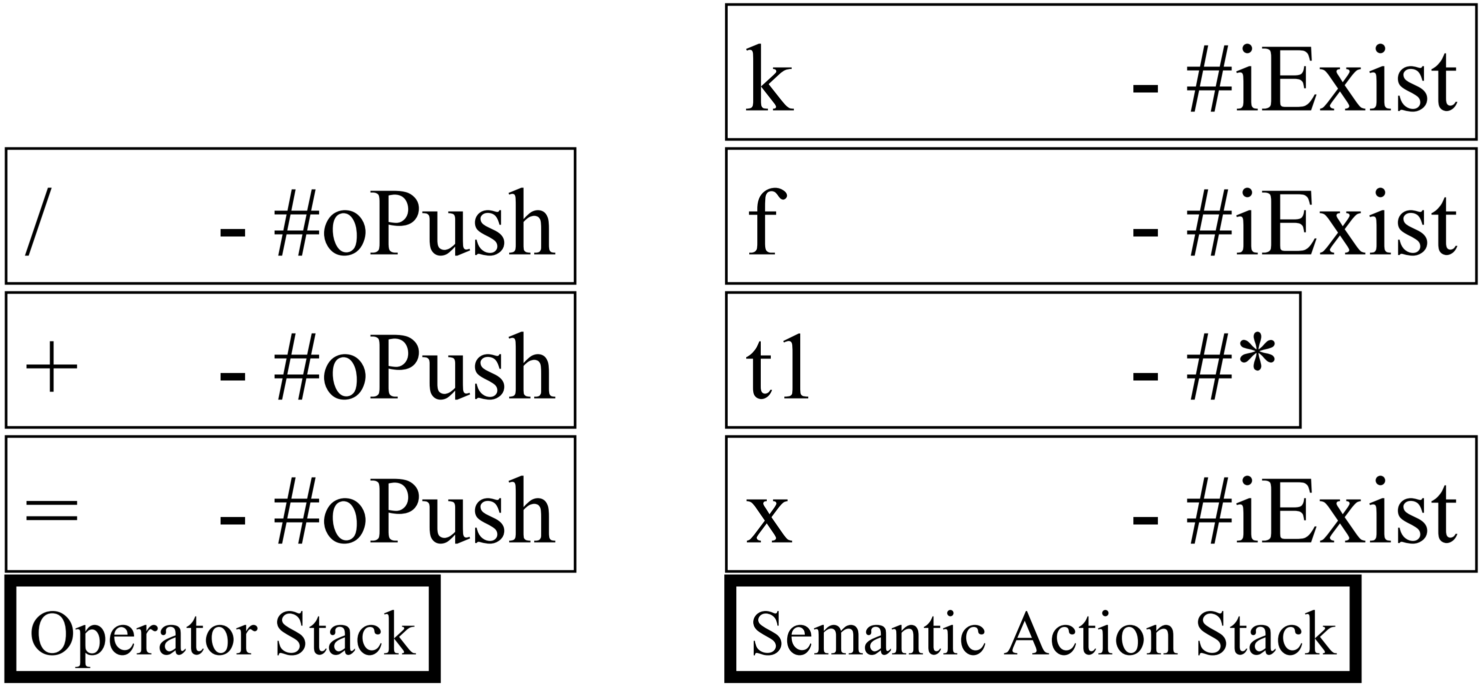
f - #iExist

t1 - #*

x - #iExist

Semantic Action Stack

```
x = y * g + f / k ;
```



/ - #oPush

+ - #oPush

= - #oPush

Operator Stack

k - #iExist

f - #iExist

t1 - #*

x - #iExist

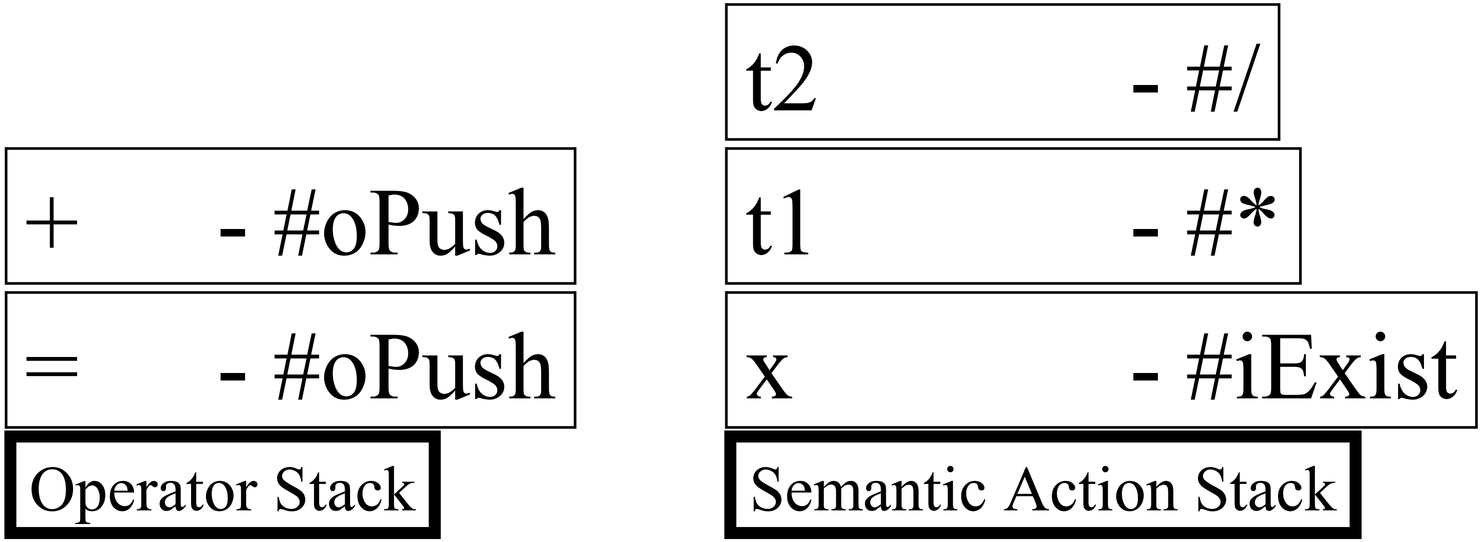
Semantic Action Stack

x = y * g + f / k ;

parse

- #EOE

```
pop /    #/
         k = SAS.pop()
         f = SAS.pop()
         Test f / k is valid
         t2 = f / k
         SAS.push(t2)
```



+	- #oPush
=	- #oPush

Operator Stack

t2	- #/
t1	- #*
x	- #iExist

Semantic Action Stack

```
x = y * g + f / k ;
```

parse

- #EOE

pop + #+

t2 = SAS.pop()

t1 = SAS.pop()

Test t1 + t2 is valid

t3 = t1 + t2

SAS.push(t3)

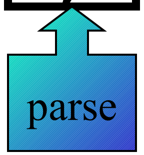
= - #oPush

Operator Stack

t3 - #+
x - #iExist

Semantic Action Stack

x = y * g + f / k $\boxed{;}$



- #EOE

pop = #=

t3 = SAS.pop()

x = SAS.pop()

Test x = t3 is valid

Operator Stack

Semantic Action Stack

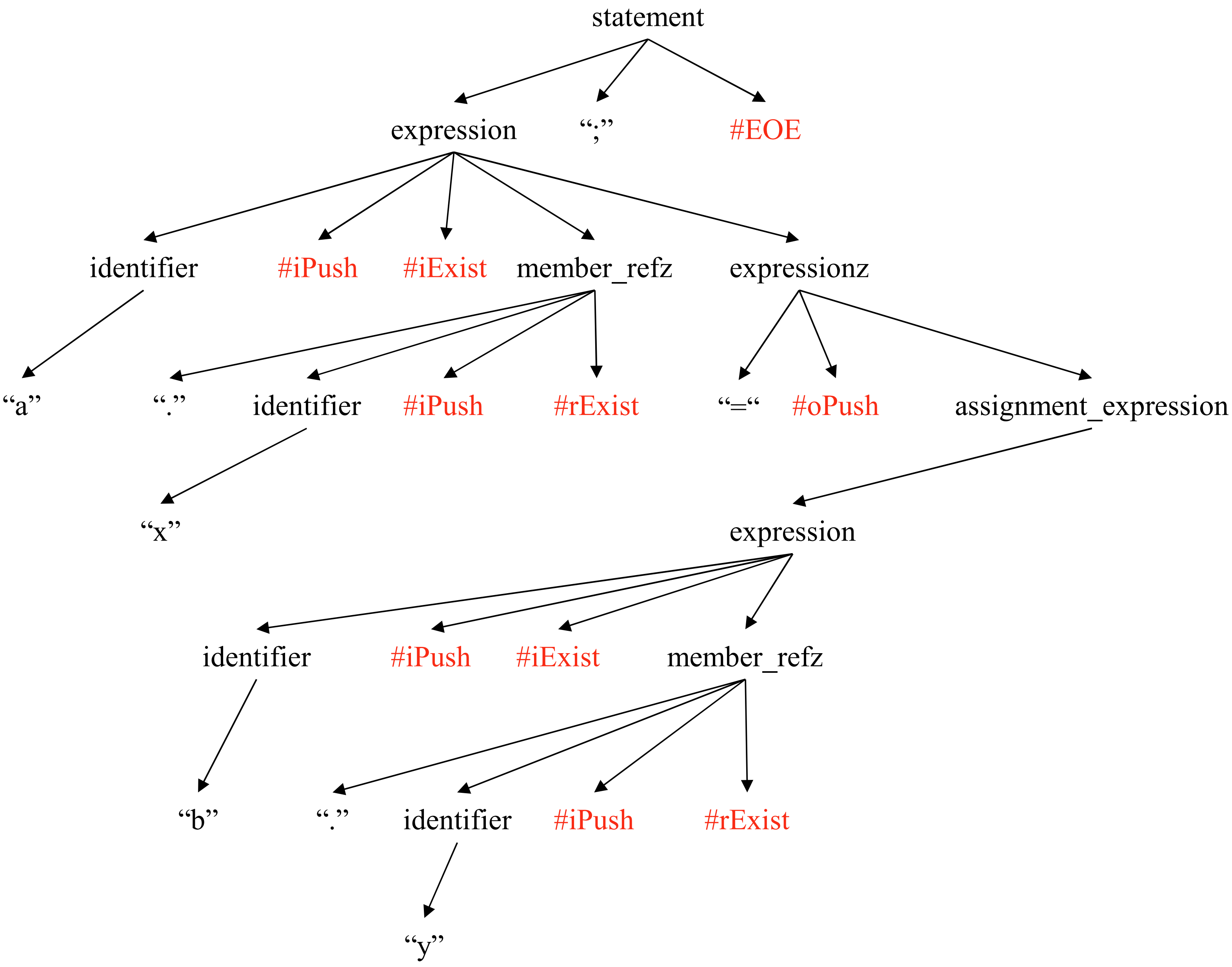
```
a . x = b . y ;
```

Simple Assignment Statement with references to a Class Member

Operator Stack

Semantic Action Stack

```
a . x = b . y ;
```



statement

expression “;” #EOE

identifier #iPush #iExist member_refz expressionz

“a” “.” identifier #iPush #rExist “=“ #oPush assignment_expression

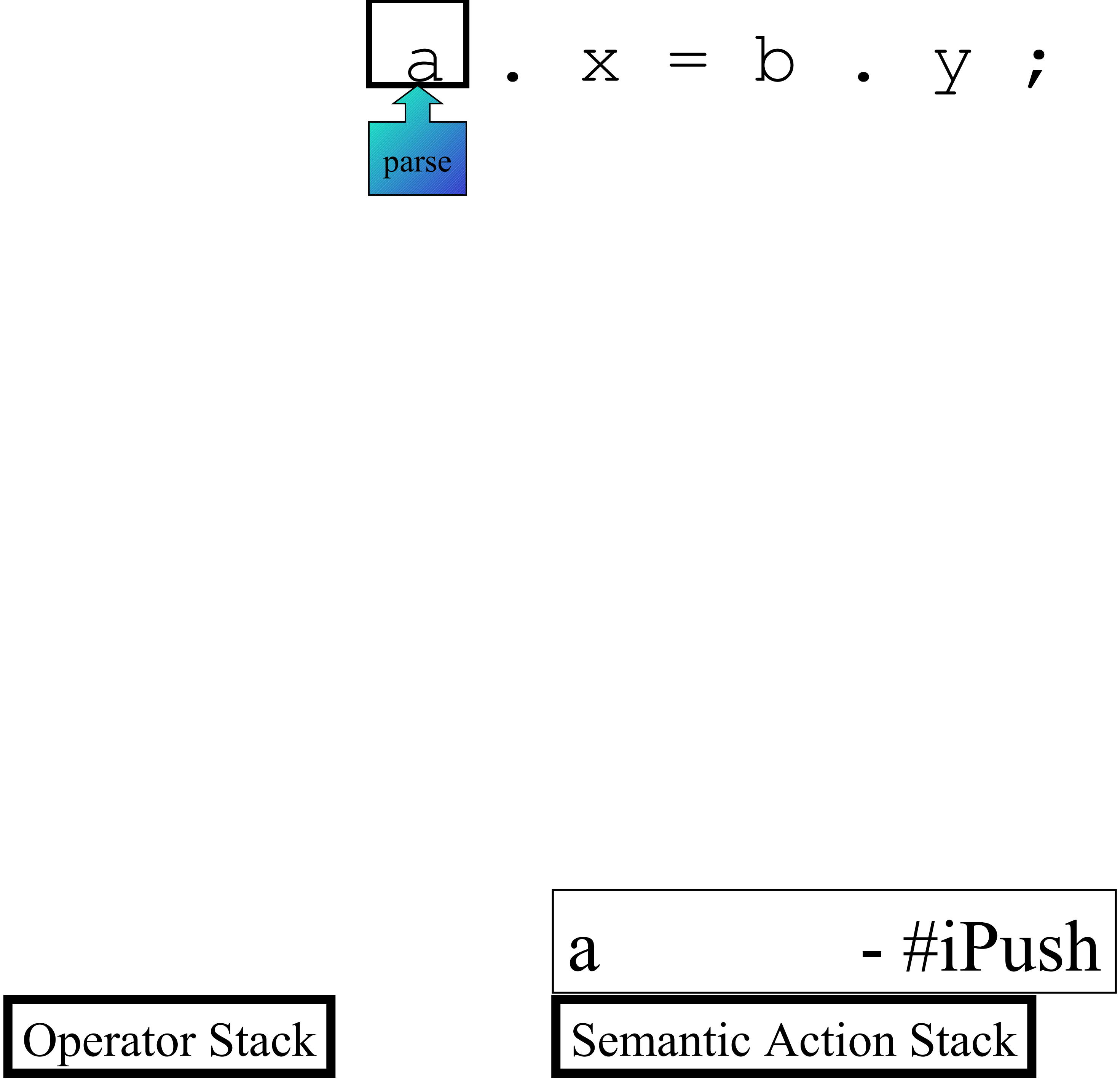
“x”

expression

identifier #iPush #iExist member_refz

“b” “.” identifier #iPush #rExist

“y”



`a . x = b . y ;`

parse

a - #iPush

Operator Stack

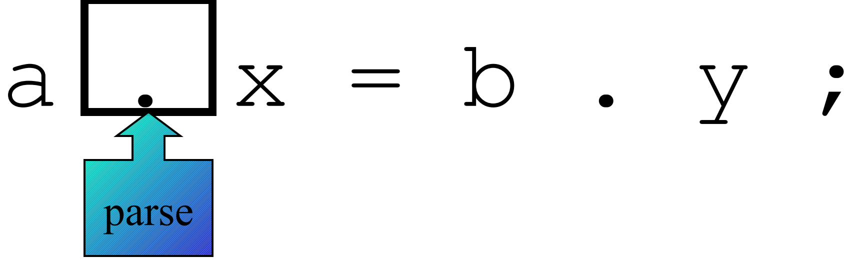
Semantic Action Stack

```
a . x = b . y ;
```

a - #iExist

Operator Stack

Semantic Action Stack



```
a . x = b . y ;
```

parse

a - #iExist

Operator Stack

Semantic Action Stack

a . x = b . y ;

parse

x - #iPush

a - #iExist

Operator Stack

Semantic Action Stack

a . x $\boxed{=}$ b . y ;

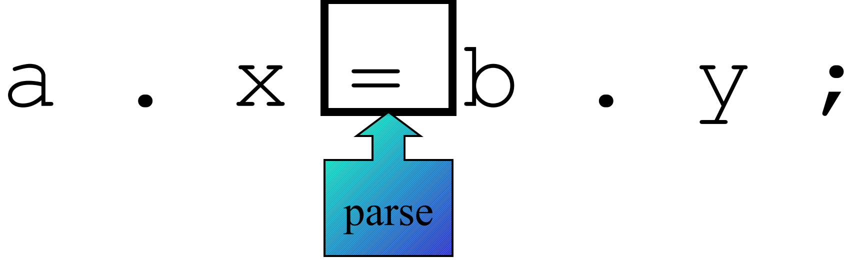
Note: #rExist creates a ref_sar for a.x by checking that the class (type) of a has an element named x. Also note there is only one SAR on the Semantic Action Stack after the #rExist routine has executed.

lvalue (location) vs. rvalue (contents)

a.x ref_sar - #rExist

Operator Stack

Semantic Action Stack



a . x = b . y ;

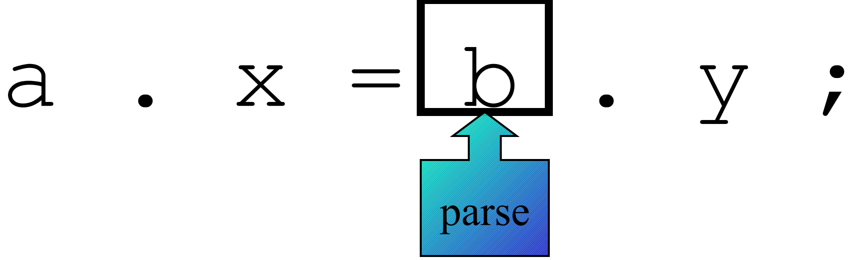
parse

= - #oPush

Operator Stack

a.x ref_sar - #rExist

Semantic Action Stack



```
a . x = b . y ;
```

b - #iPush

= - #oPush

a.x ref_sar - #rExist

Operator Stack

Semantic Action Stack

a . x = b $\boxed{.}$ y ;

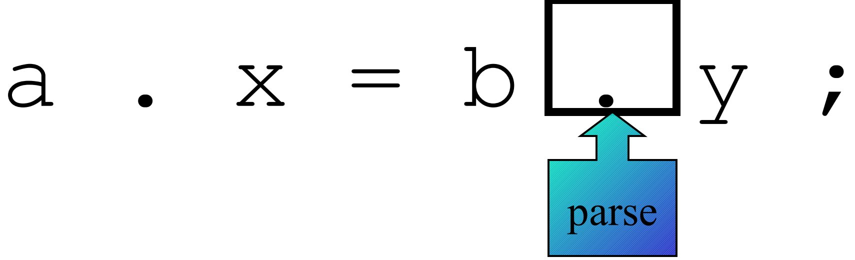
= - #oPush

Operator Stack

b - #iExit

a.x ref_sar - #rExist

Semantic Action Stack



```
a . x = b . y ;
```

b - #iExit

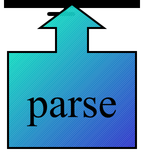
= - #oPush

a.x ref_sar - #rExist

Operator Stack

Semantic Action Stack

```
a . x = b . y ;
```



y - #iPush

b - #iExit

a.x ref_sar - #rExist

= - #oPush

Operator Stack

Semantic Action Stack

`a . x = b . y ;`

Note: #rExist creates a ref_sar for b.y by checking that the class (type) of b has an element named y. Also note there are only two SAR on the Semantic Action Stack after the #rExist routine has executed.

= - #oPush

Operator Stack

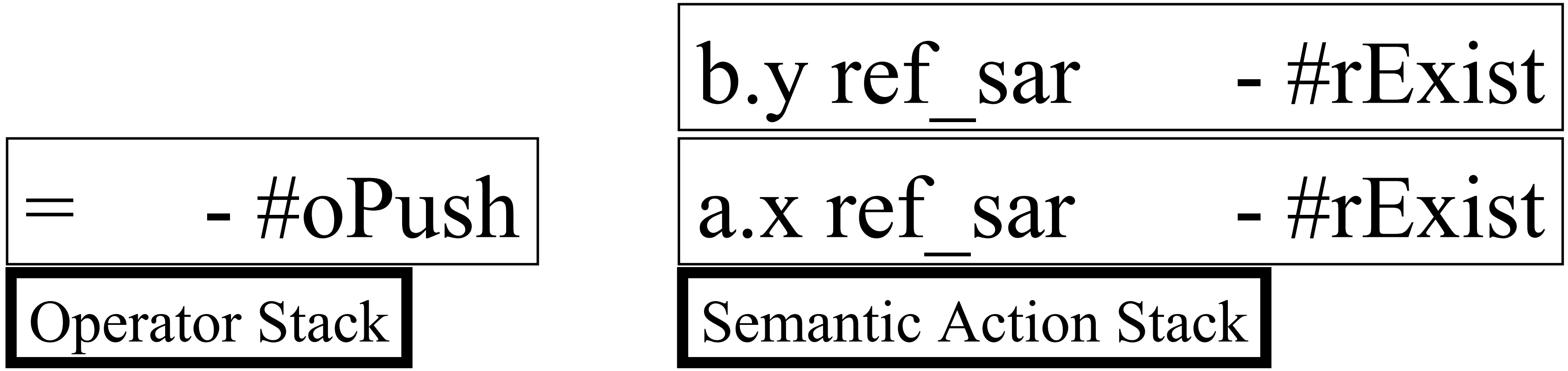
b.y ref_sar - #rExist

a.x ref_sar - #rExist

Semantic Action Stack

$\texttt{a . x = b . y} \boxed{\texttt{;}}$

Note: The ref_sar for a.x and b.y, ultimately will relate to a collection of code that will compute a location in memory (e.g., Base Address + Offset). The computed address should be stored in the Symbol Table like any other identifier. Thus there will need to be a symbol id (e.g., t9 and t10) associated with each of the ref_sar's which is not shown in the example. The symbol id, can be created and added to the Symbol Table when the ref_sar is created or as I prefer to think of it when the ref_sar is popped from the Semantic Action Stack.



= - #oPush

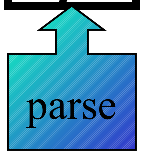
Operator Stack

b.y ref_sar - #rExist

a.x ref_sar - #rExist

Semantic Action Stack

a . x = b . y ;



- #EOE

pop = #=
b.y ref_sar (t9) = SAS.pop()
a.x ref_sar (t10) = SAS.pop()
Test a.x = b.y is valid

Note: Just as when x = y, the semantic action #= only has to deals with two SAR's (each ref_sar should have an entry in the Symbol Table), one for the LHS and one for the RHS. The assignment statement is not more complex, it simply must deals with the references before processing the assignment statement.

Operator Stack

Semantic Action Stack

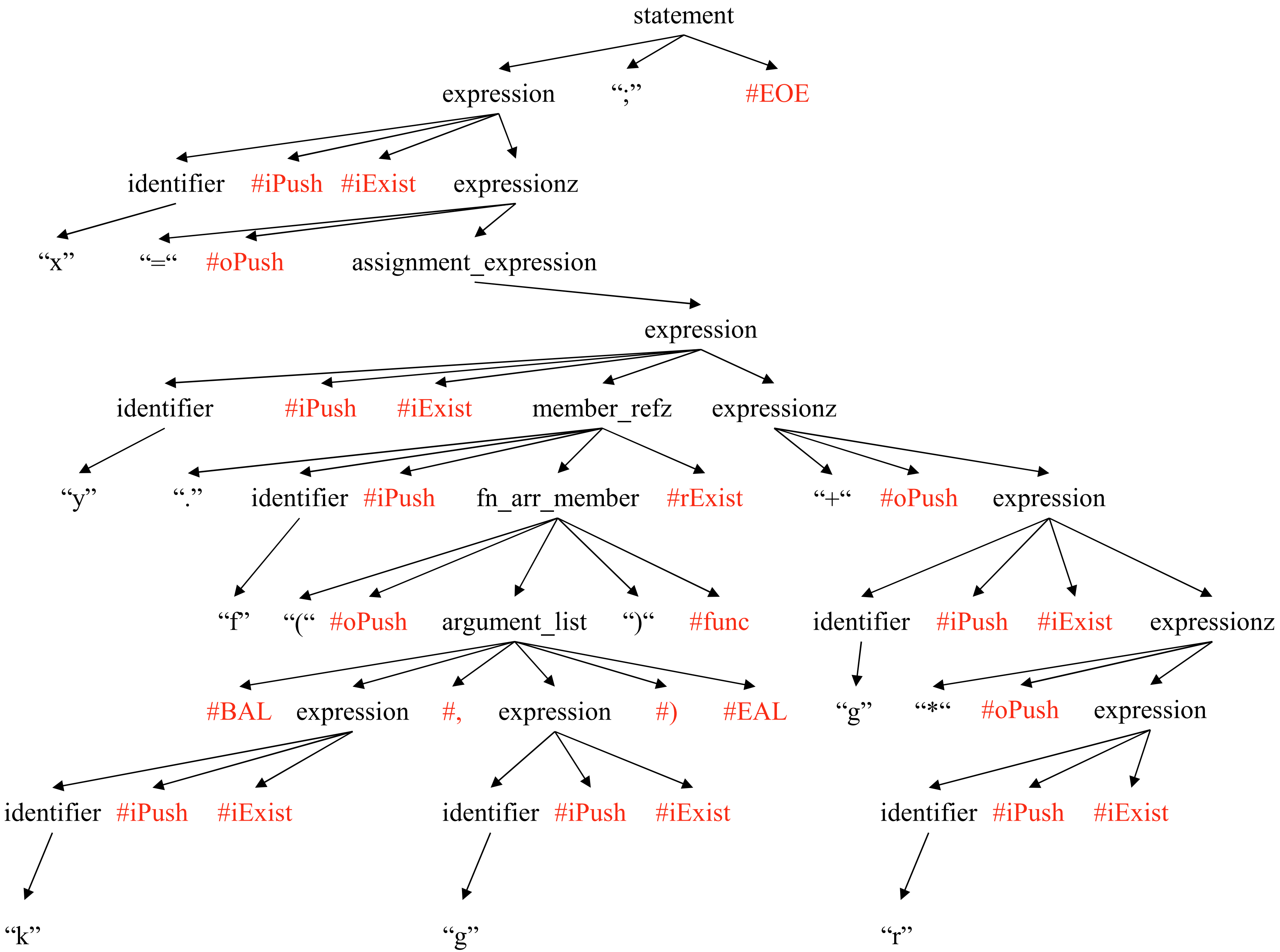
```
x = y . f( k , g ) + g * r ;
```

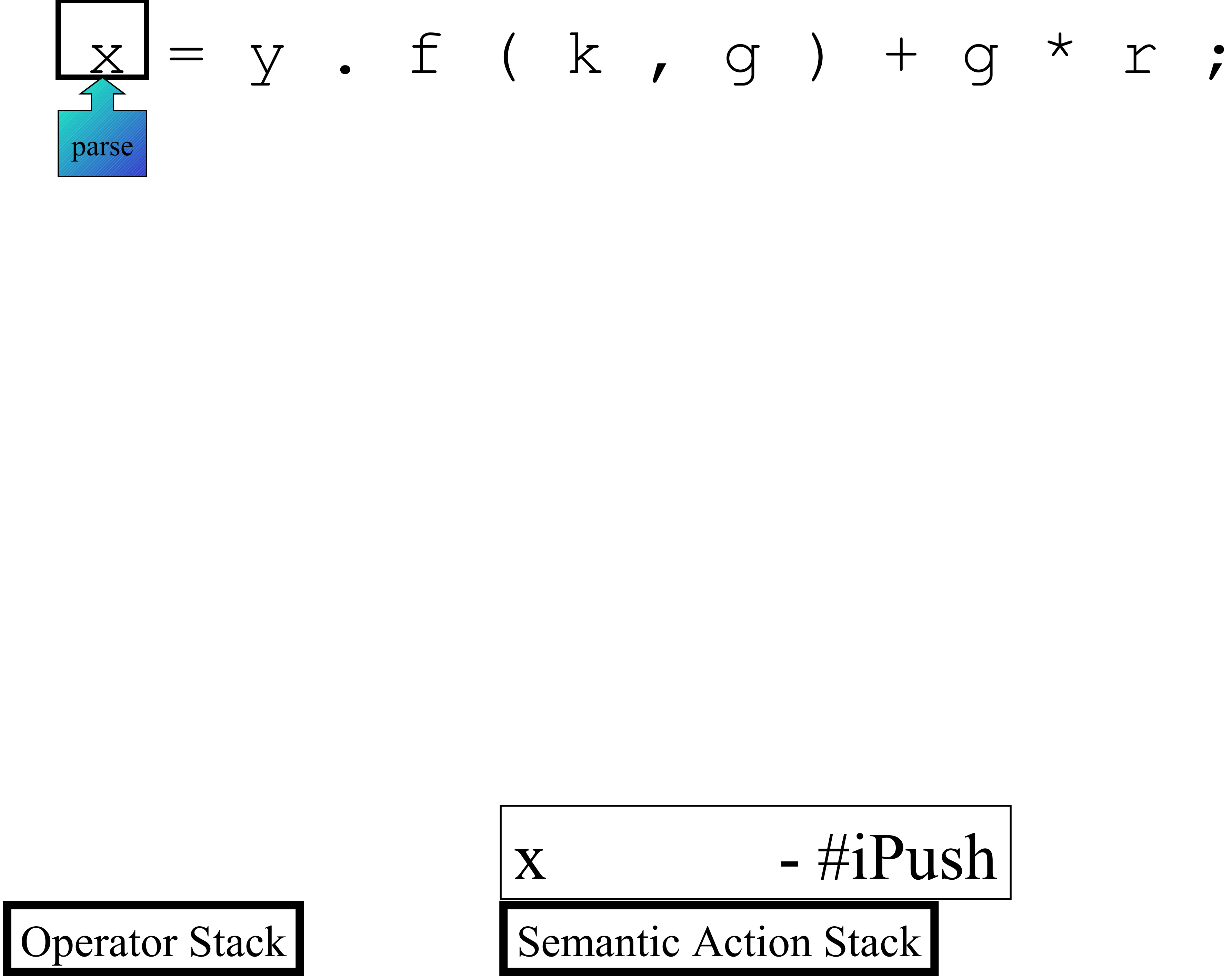
Assignment Statement of member function with infix to postfix conversion

Operator Stack

Semantic Action Stack

```
x = y . f( k , g ) + g * r ;
```


statement
expression
“;”
#EOE
identifier
#iPush
#iExist
expressionz
“x”
“=“
#oPush
assignment_expression
expression
identifier
#iPush
#iExist
member_refz
expressionz
“y”
“.”
identifier
#iPush
fn_arr_member
#rExist
“+“
#oPush
expression
“f”
“(“
#oPush
argument_list
“)“
#func
identifier
#iPush
#iExist
expressionz
#BAL
expression
#,
expression
#)
#EAL
“g”
“*“
#oPush
expression
identifier
#iPush
#iExist
identifier
#iPush
#iExist
identifier
#iPush
#iExist
“k”
“g”
“r”



```
x = y . f ( k , g ) + g * r ;
```

parse

x - #iPush

Operator Stack

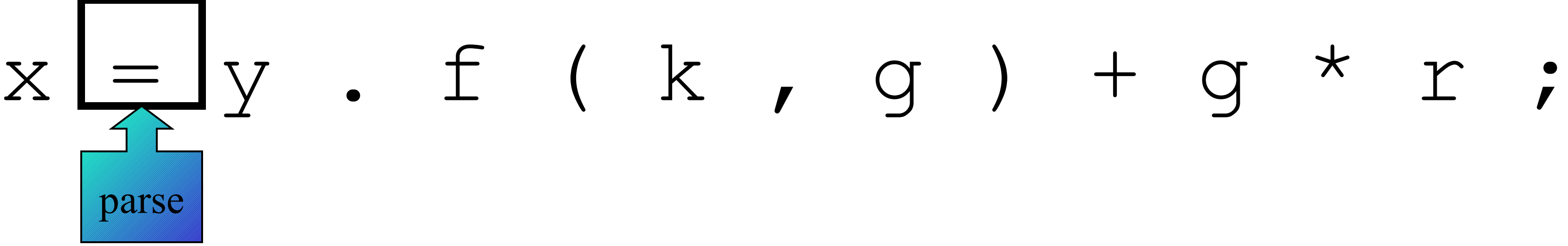
Semantic Action Stack

x $\boxed{=}$ y . f (k , g) + g * r ;

x - #iExist

Operator Stack

Semantic Action Stack



x = y . f (k , g) + g * r ;

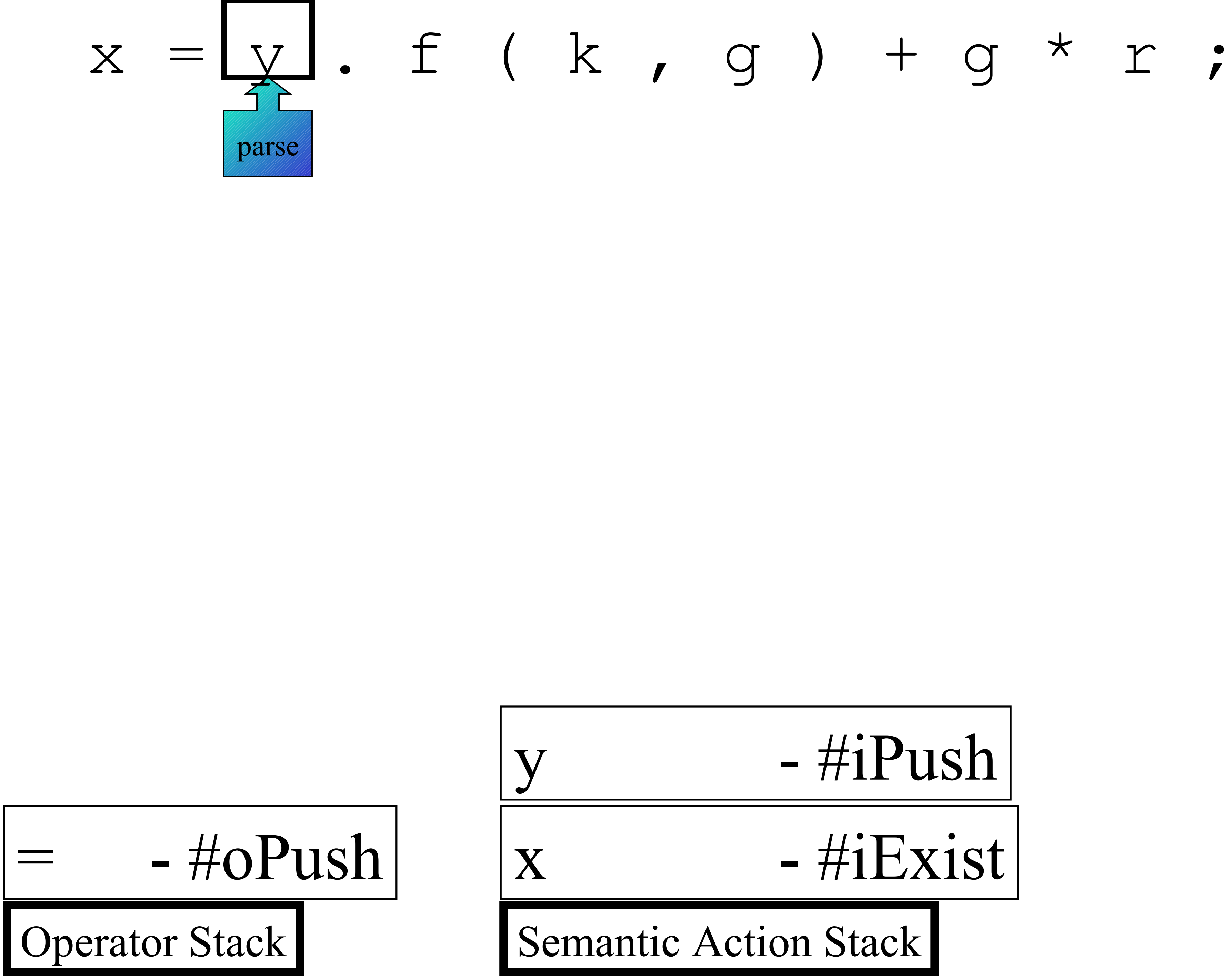
parse

= - #oPush

Operator Stack

x - #iExist

Semantic Action Stack



```
x = y . f ( k , g ) + g * r ;
```

y	- #iPush
x	- #iExist

Semantic Action Stack

=	- #oPush

Operator Stack

x = y $\boxed{.}$ f (k , g) + g * r ;

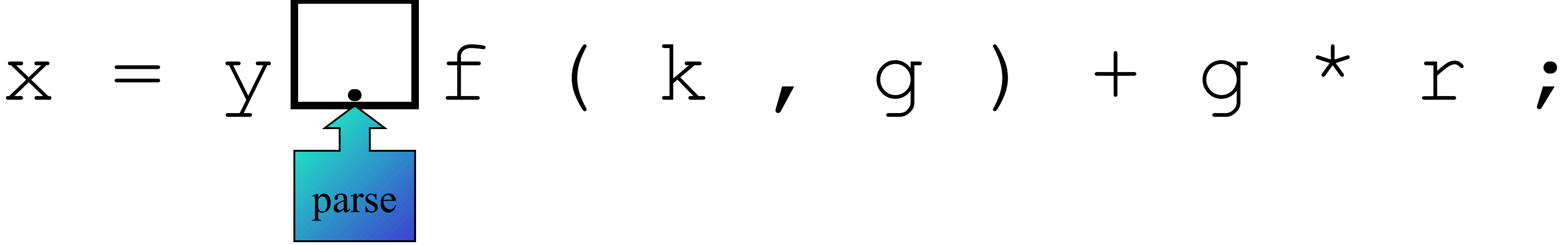
= - #oPush

Operator Stack

y - #iExist

x - #iExist

Semantic Action Stack



```
x = y . f ( k , g ) + g * r ;
```

parse

y - #iExist

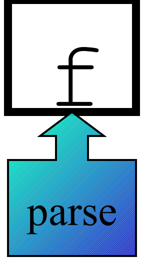
= - #oPush

x - #iExist

Operator Stack

Semantic Action Stack

```
x = y . f ( k , g ) + g * r ;
```



f - #iPush

y - #iExist

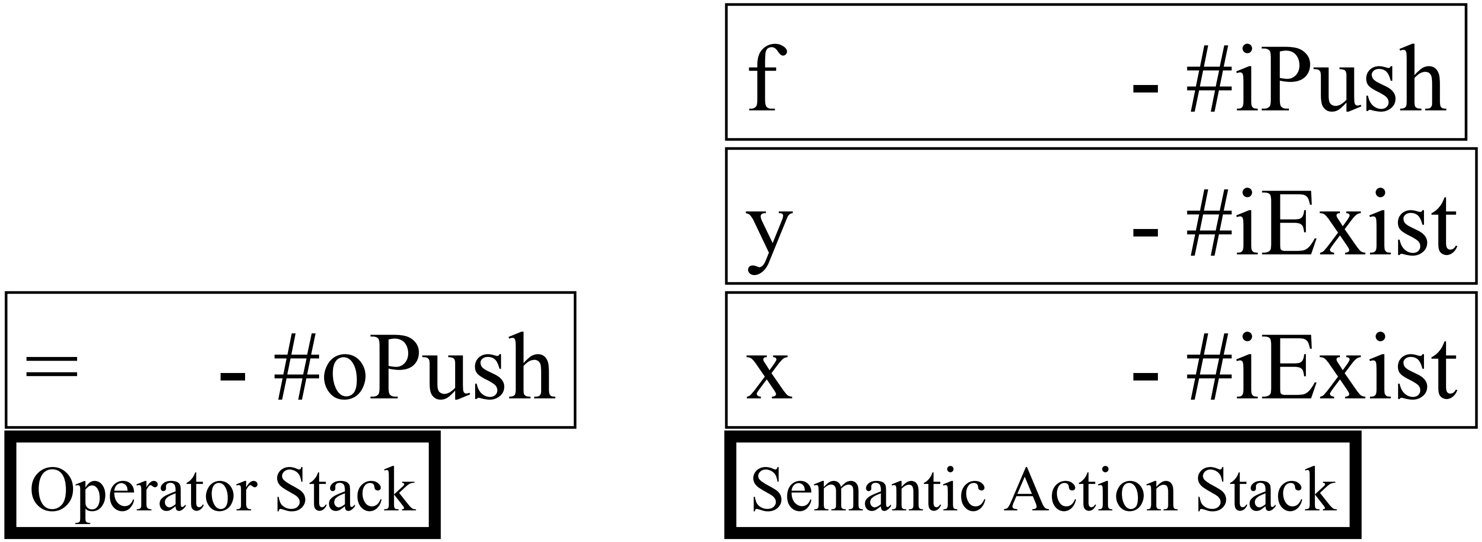
x - #iExist

Semantic Action Stack

= - #oPush

Operator Stack

```
x = y . f ( k , g ) + g * r ;
```



f - #iPush

y - #iExist

x - #iExist

Semantic Action Stack

= - #oPush

Operator Stack

```
x = y . f ( k , g ) + g * r ;
```

parse

bal_sar - #BAL

f - #iPush

(- #oPush

y - #iExist

= - #oPush

x - #iExist

Operator Stack

Semantic Action Stack

```
x = y . f ( k , g ) + g * r ;
```

parse

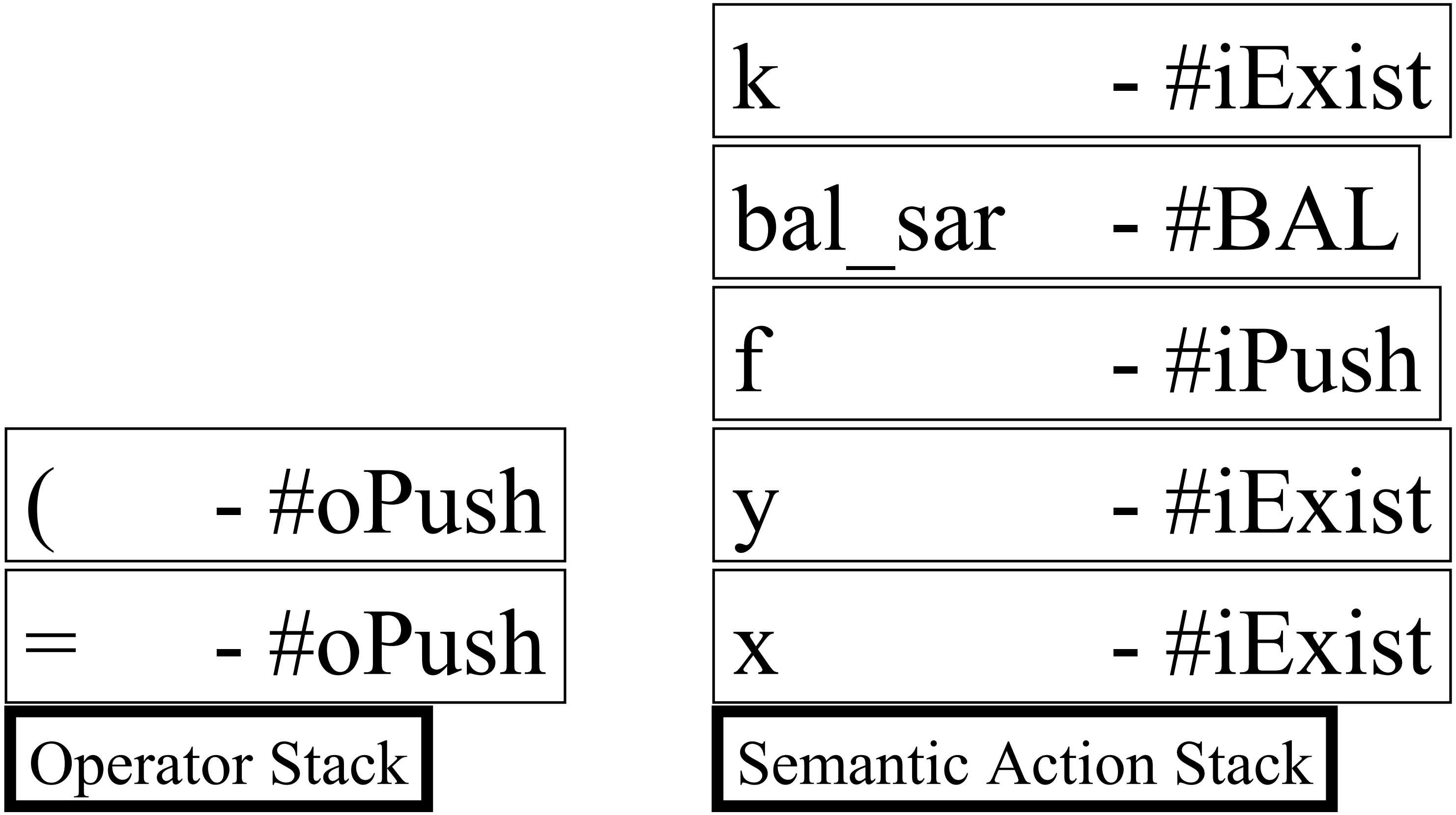
(	- #oPush
=	- #oPush

Operator Stack

k	- #iPush
bal_sar	- #BAL
f	- #iPush
y	- #iExist
x	- #iExist

Semantic Action Stack

```
x = y . f ( k , g ) + g * r ;
```



(- #oPush

= - #oPush

Operator Stack

k - #iExist

bal_sar - #BAL

f - #iPush

y - #iExist

x - #iExist

Semantic Action Stack

x = y . f (k $\boxed{,}$ g) + g * r ;

parse

- #,

pop nothing

No infix to postfix conversion necessary on expression k

k - #iExist

bal_sar - #BAL

f - #iPush

y - #iExist

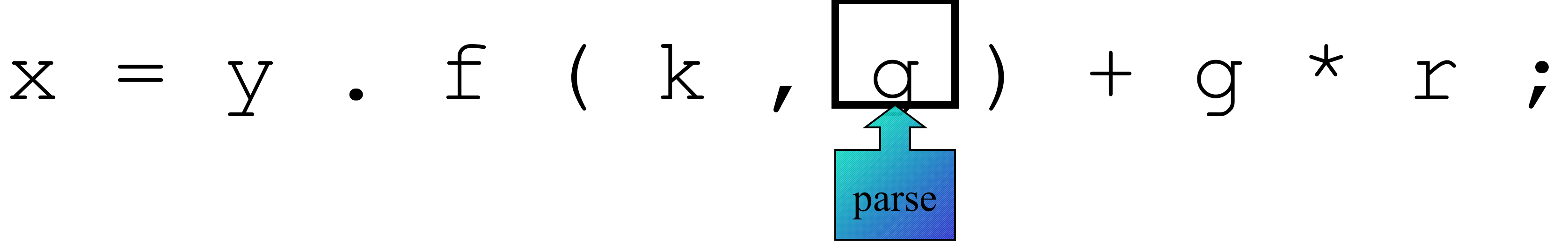
x - #iExist

Semantic Action Stack

(- #oPush

= - #oPush

Operator Stack

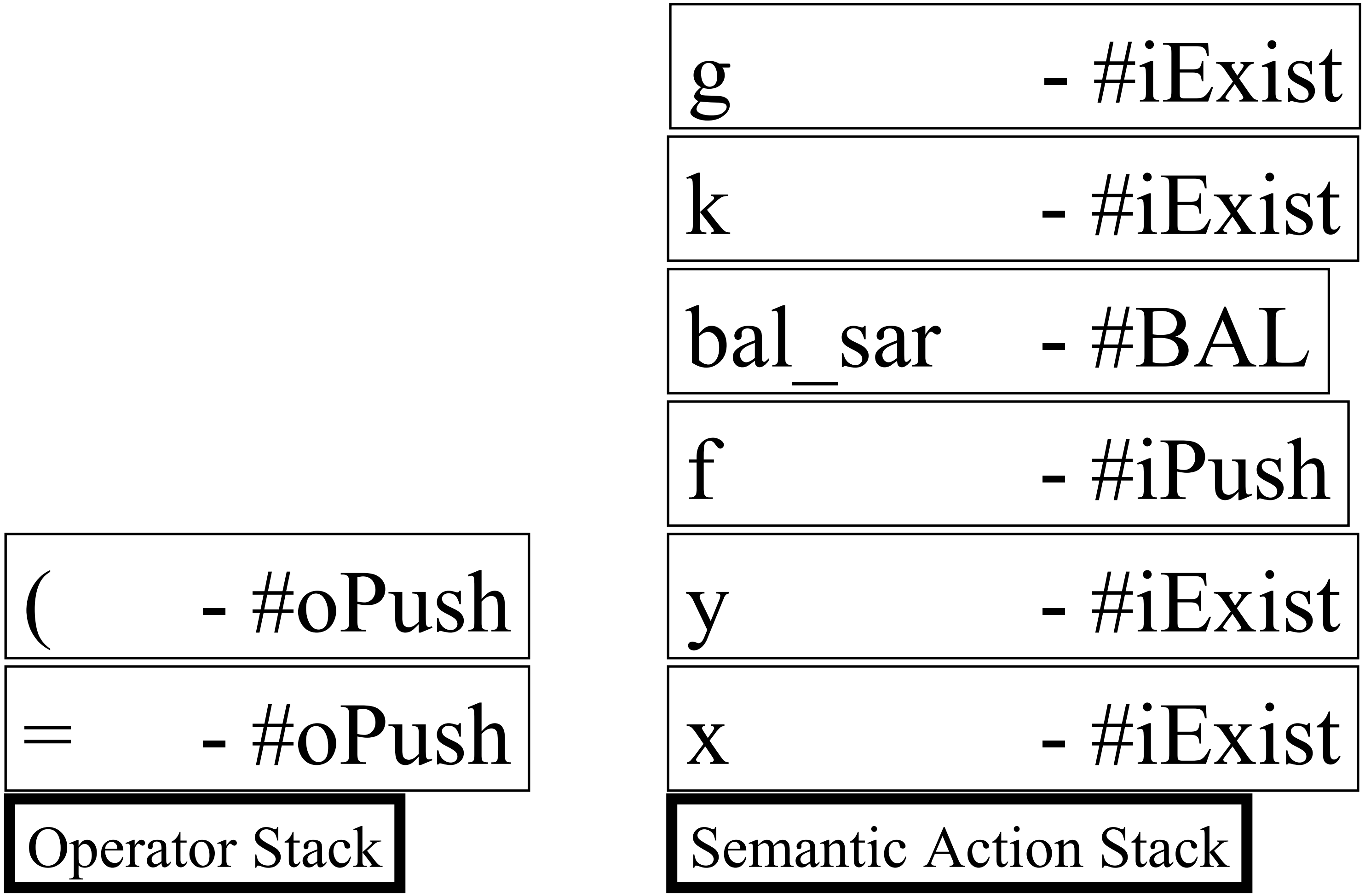


```
x = y . f ( k , g ) + g * r ;
```

g - #iPush
k - #iExist
bal_sar - #BAL
f - #iPush
y - #iExist
x - #iExist
Semantic Action Stack

(- #oPush
= - #oPush
Operator Stack

```
x = y . f ( k , g ) + g * r ;
```



Operator Stack

(- #oPush

= - #oPush

Semantic Action Stack

g - #iExist

k - #iExist

bal_sar - #BAL

f - #iPush

y - #iExist

x - #iExist

x = y . f (k , g) + g * r ;

- #)

pop (

No infix to postfix conversion necessary on expression g

= - #oPush

Operator Stack

al_sar - #EAL

f - #iPush

y - #iExist

x - #iExist

Semantic Action Stack

x = y . f (k , g $\boxed{)}$ + g * r ;

- #EAL
g = SAS.pop()
k = SAS.pop()
bal_sar == SAS.pop()
Add g & k to al_sar
SAS.push(al_sar)

al_sar - #EAL

f - #iPush

y - #iExist

x - #iExist

Semantic Action Stack

= - #oPush

Operator Stack

`x = y . f ( k , g ) + g * r ;`

parse

- #func

al_sar = SAS.pop()
f = SAS.pop()
Add f & al_sar to func_sar
SAS.push(func_sar)

Note: #func ***can't determine if the function exist*** it simply combines the argument list with the function name to form the function_sar.

func_sar - #func
y - #iExist
x - #iExist

Semantic Action Stack

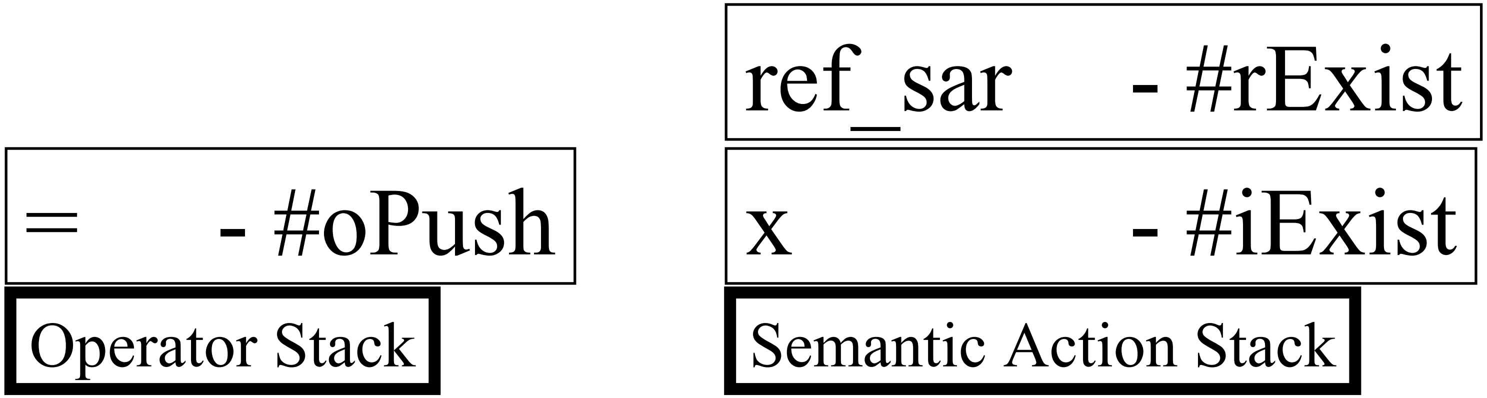
= - #oPush

Operator Stack

$x = y . f (k , g) \boxed{+} g * r ;$

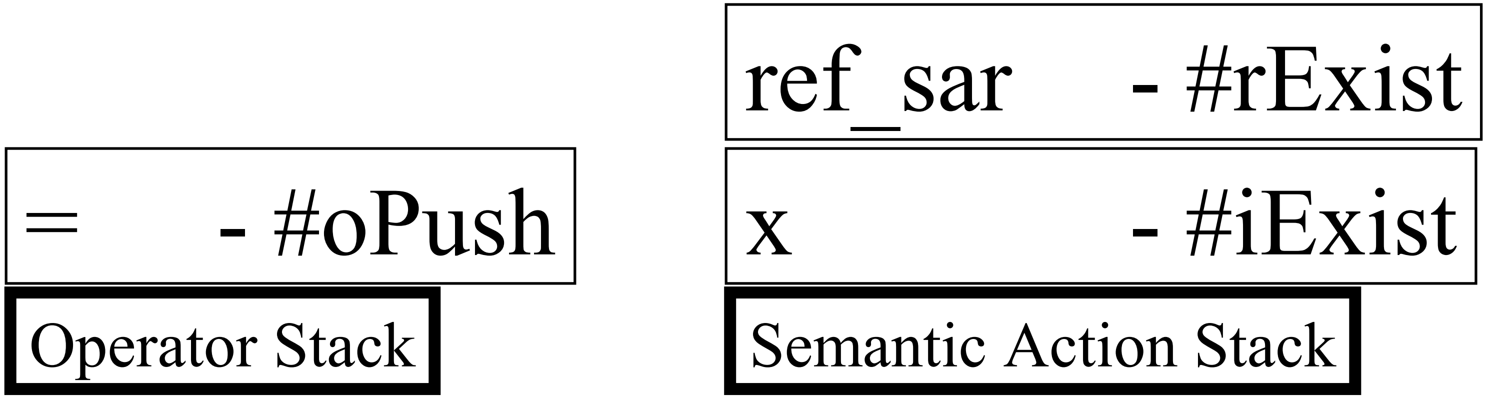
```
- #rExist
top = SAS.pop()
next = SAS.pop()
Test that top is a valid member of next (next.top)
Create ref_sar for top and next
SAS.push(ref_sar)
```

Note: #rExist *will determine if the function exist* because it is part of a reference [e.g, y.f(k, g)]. *#iExist would be used* if the function is not part of reference (see later slides).



x = y . f (k , g) $\boxed{+}$ g * r ;

Note: When evaluating function calls from semantic analysis *a ref_sar created via a #rExists* or *id_sar created via an #iExist* will be on the Semantic Action Stack after the function has been determined to exist. While the SAR on the SA Stack is sufficient at this time, it may help to understand that a temporary variable must be associated with the SAR if the function returns a value. The temporary variable will not be used until Intermediate and Target Code Generation. Hint: Just let the ref_sar and id_sar have a temporary variable attribute, if you want.



= - #oPush

Operator Stack

ref_sar - #rExist

x - #iExist

Semantic Action Stack

```
x = y . f ( k , g ) + g * r ;
```

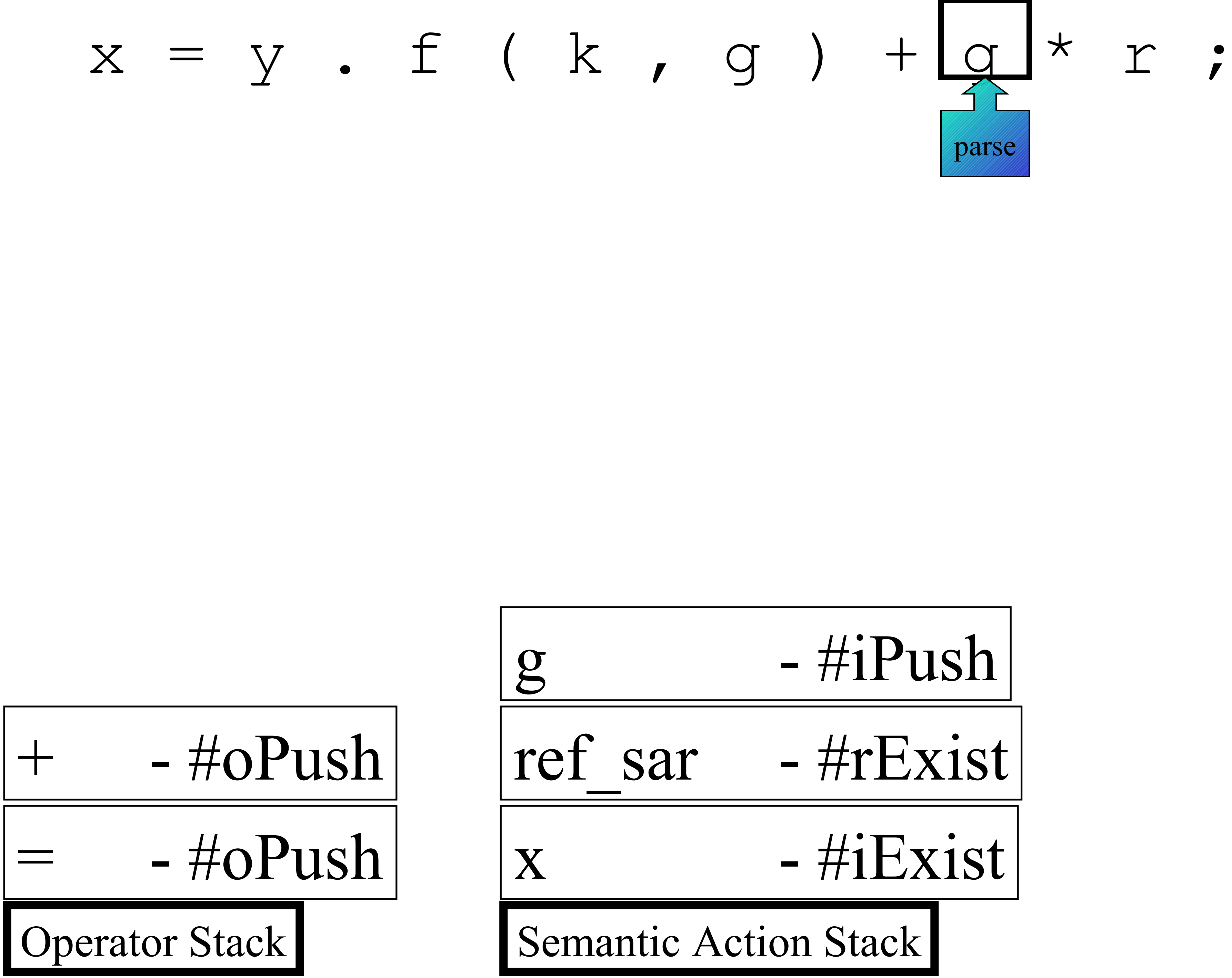
parse

+	- #oPush
=	- #oPush

Operator Stack

ref_sar	- #rExist
x	- #iExist

Semantic Action Stack



```
x = y . f ( k , g ) + g * r ;
```

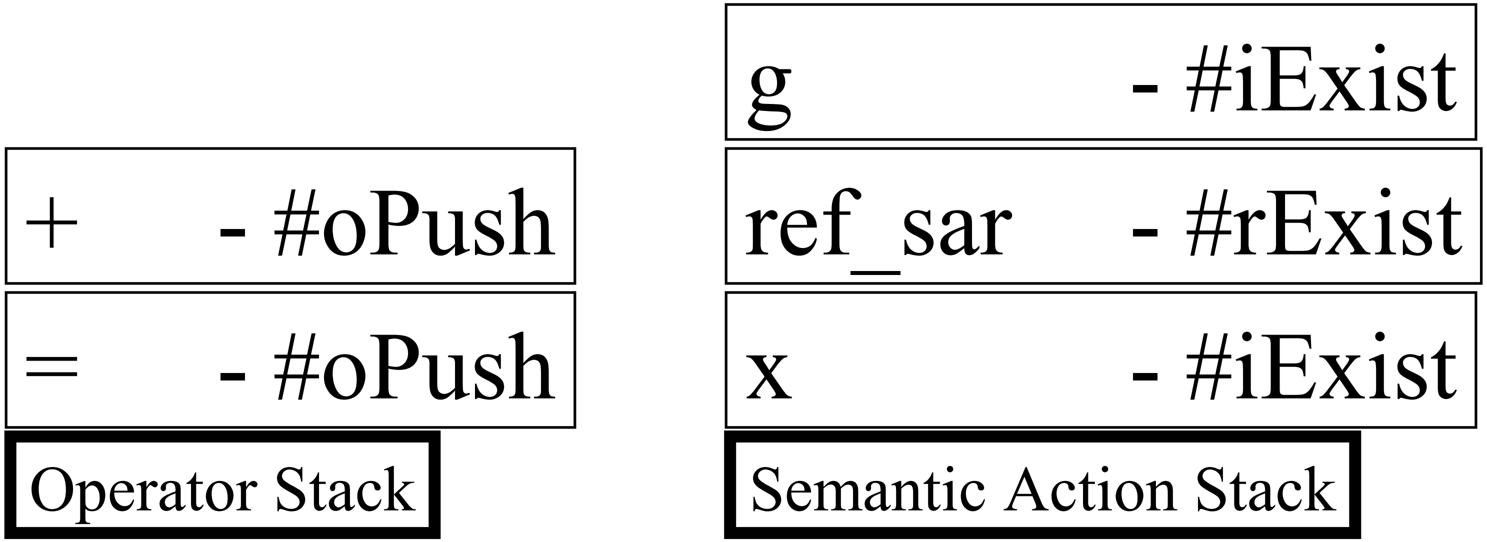
+	- #oPush
=	- #oPush

Operator Stack

g	- #iPush
ref_sar	- #rExist
x	- #iExist

Semantic Action Stack

```
x = y . f ( k , g ) + g * r ;
```



+ - #oPush

= - #oPush

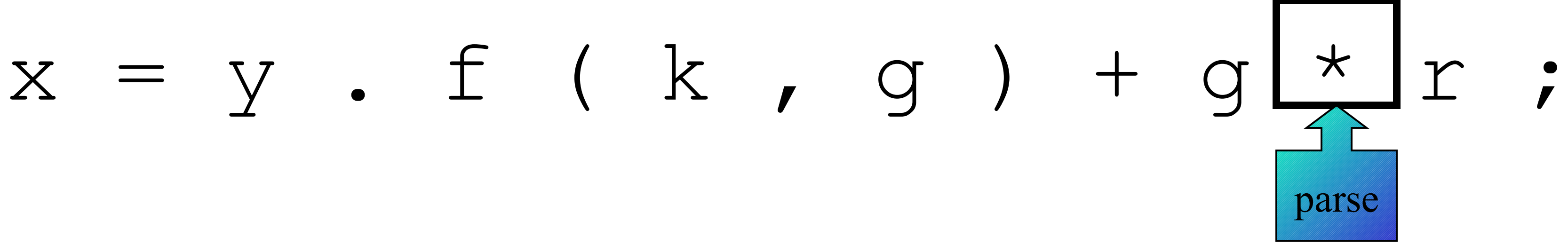
Operator Stack

g - #iExist

ref_sar - #rExist

x - #iExist

Semantic Action Stack



```
x = y . f ( k , g ) + g * r ;
```

Operator Stack	
*	- #oPush
+	- #oPush
=	- #oPush

Semantic Action Stack	
g	- #iExist
ref_sar	- #rExist
x	- #iExist

```
x = y . f ( k , g ) + g * r ;
```

parse

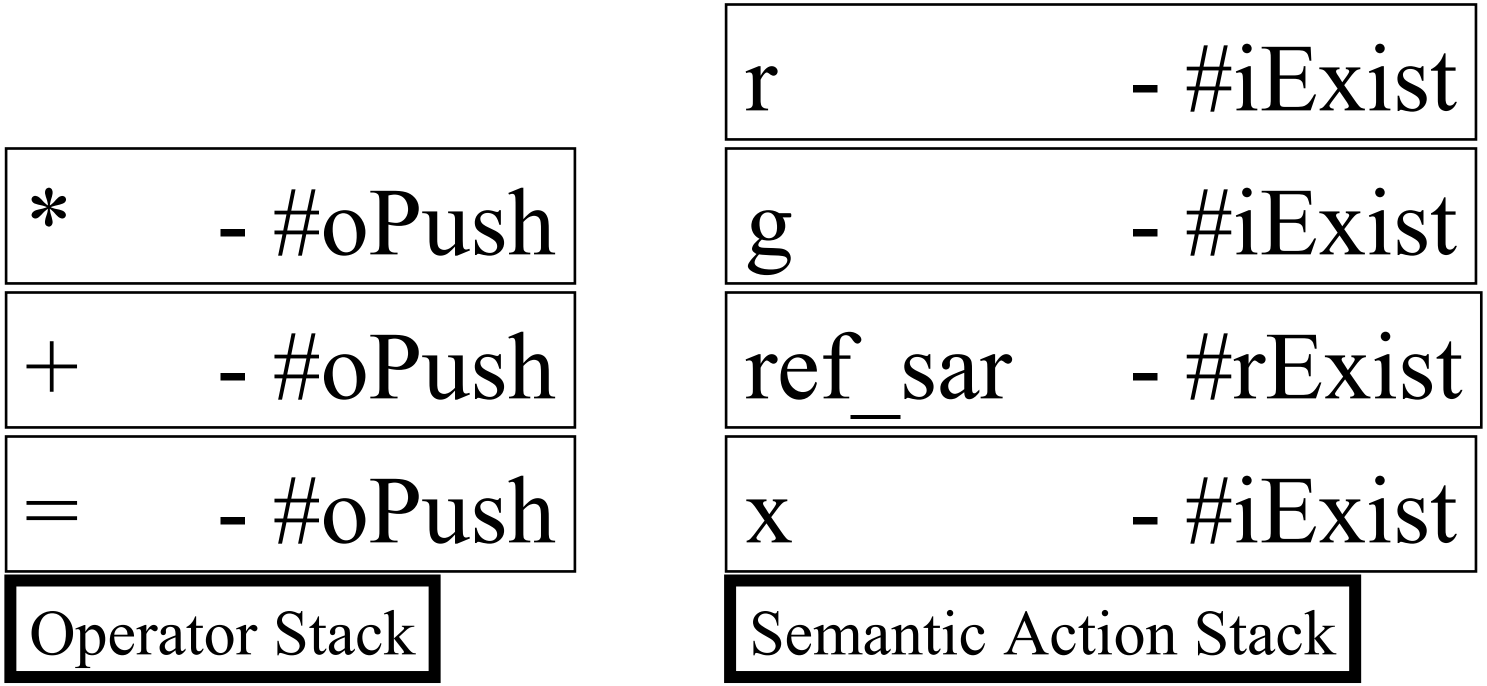
*	- #oPush
+	- #oPush
=	- #oPush

Operator Stack

r	- #iPush
g	- #iExist
ref_sar	- #rExist
x	- #iExist

Semantic Action Stack

```
x = y . f ( k , g ) + g * r ;
```



* - #oPush

+ - #oPush

= - #oPush

Operator Stack

r - #iExist

g - #iExist

ref_sar - #rExist

x - #iExist

Semantic Action Stack

```
x = y . f ( k , g ) + g * r ;
```

parse

```
        - #EOE
pop *   #*
        r = SAS.pop()
        g = SAS.pop()
        Test g * r is valid
        t4 = g * r
        SAS.push(t4)
```

+	- #oPush
=	- #oPush

Operator Stack

t4	- #EOE
ref_sar	- #rExist
x	- #iExist

Semantic Action Stack

```
x = y . f ( k , g ) + g * r ;
```

parse

- #EOE

pop + #+

```
t4 = SAS.pop()
ref_sar = SAS.pop()
Test ref_sar + t4 is valid
t6 = ref_sar + t4
SAS.push(t6)
```

Note: An entry should be created in the Symbol Table for the y.f(k,g) ref_sar.

= - #oPush

Operator Stack

t6 - #EOE

x - #iExist

Semantic Action Stack

```
x = y . f ( k , g ) + g * r ;
```

parse

- #EOE

pop = #=

t6 = SAS.pop()

x = SAS.pop()

Test x = t6 is valid

Operator Stack

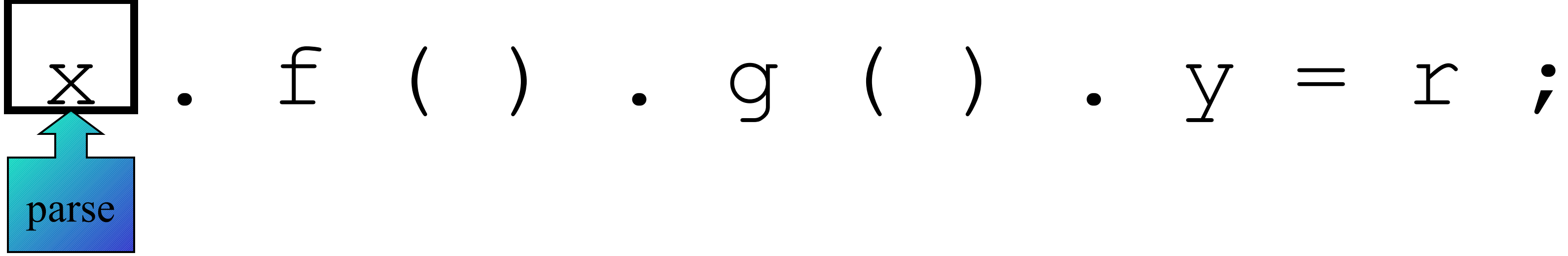
Semantic Action Stack

$\boxed{\texttt{x}}$`.f().g().y=r;`

Assignment Statement to a member element returned by a function

Operator Stack

Semantic Action Stack



```
x . f ( ) . g ( ) . y = r ;
```

parse

x - #iPush

Operator Stack

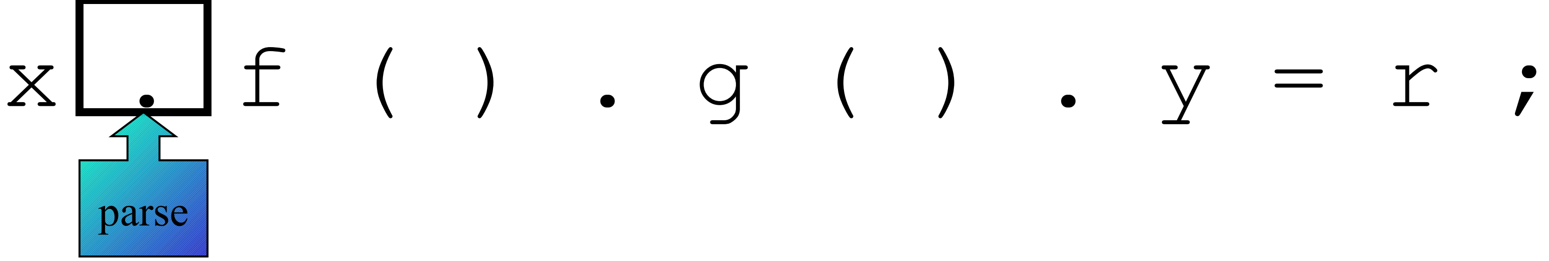
Semantic Action Stack

$\texttt{x}\boxed{\texttt{.}}\texttt{ f ( ) . g ( ) . y = r ;}$

x - #iExist

Operator Stack

Semantic Action Stack



```
x . f ( ) . g ( ) . y = r ;
```

x - #iExist

Semantic Action Stack

Operator Stack

```
x . f ( ) . g ( ) . y = r ;
```

parse

f - #iPush

x - #iExist

Operator Stack

Semantic Action Stack

$$\texttt{x . f }\boxed{\texttt{(}}\texttt{) . g ( ) . y = r ;}$$

f - #iPush

x - #iExist

Operator Stack

Semantic Action Stack

```
x . f ( ) . g ( ) . y = r ;
```

parse

(- #oPush

Operator Stack

bal_sar - #BAL

f - #iPush

x - #iExist

Semantic Action Stack

```
x . f ( ) . g ( ) . y = r ;
```

- #)

pop (

al_sar - #EAL

f - #iPush

x - #iExist

Operator Stack

Semantic Action Stack

```
x . f ( ) . g ( ) . y = r ;
```

- #EAL

bal_sar == SAS.pop()
Create empty al_sar
SAS.push(al_sar)

al_sar - #EAL
f - #iPush
x - #iExist

Operator Stack

Semantic Action Stack

```
x . f ( ) . g ( ) . y = r ;
```

parse

- #func

```
al_sar = SAS.pop()
f = SAS.pop()
Add f & al_sar to func_sar
SAS.push(func_sar)
```

func_sar	- #func
x	- #iExist

Operator Stack

Semantic Action Stack

```
x . f ( ) . g ( ) . y = r ;
```

- #rExist

func_sar = SAS.pop()

x = SAS.pop()

Test that func_sar (function f) is a valid member of the class of x

Create ref_sar for the result

SAS.push(ref_sar)

ref_sar - #rExist

Operator Stack

Semantic Action Stack

```
x.f().g().y=r;
```

parse

ref_sar - #rExist

Operator Stack

Semantic Action Stack

```
x . f ( ) . g ( ) . y = r ;
```

parse

g - #iPush

ref_sar - #rExist

Operator Stack

Semantic Action Stack

```
x . f ( ) . g ( ) . y = r ;
```

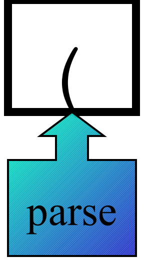
g - #iPush

ref_sar - #rExist

Operator Stack

Semantic Action Stack

```
x . f ( ) . g ( ) . y = r ;
```



(- #oPush

Operator Stack

bal_sar - #BAL

g - #iPush

ref_sar - #rExist

Semantic Action Stack

```
x . f ( ) . g ( ) . y = r ;
```

- #)

pop (

al_sar - #EAL

g - #iPush

ref_sar - #rExist

Operator Stack

Semantic Action Stack

```
x . f ( ) . g ( ) . y = r ;
```

- #EAL

bal_sar == SAS.pop()

Create empty al_sar

SAS.push(al_sar)

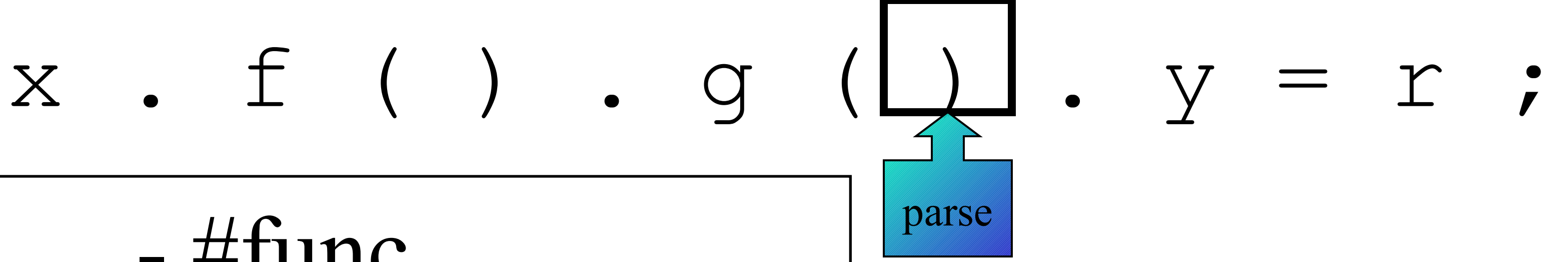
al_sar - #EAL

g - #iPush

ref_sar - #rExist

Operator Stack

Semantic Action Stack



```
x . f ( ) . g ( ) . y = r ;
```

parse

- #func

al_sar = SAS.pop()

g = SAS.pop()

Add g & al_sar to func_sar

SAS.push(func_sar)

func_sar - #func

ref_sar - #rExist

Operator Stack

Semantic Action Stack

x . f ( ) . g ( ) $\boxed{.}$ y = r ;

- #rExist

func_sar = SAS.pop()

ref_sar = SAS.pop(), the x.f() ref_sar

Test that the func_sar (function g) is a valid member of the ref_sar (the return type of function f)

Create ref_sar for the result

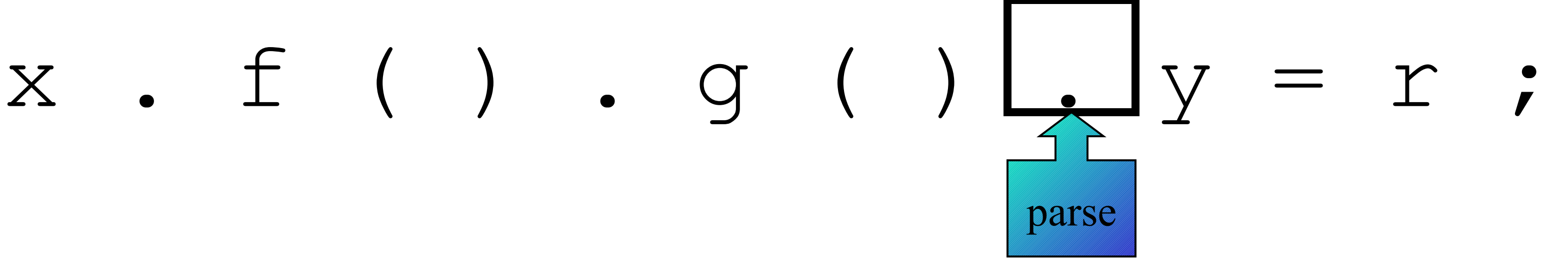
SAS.push(ref_sar)

Note: A a Symbol Table entry is needed for the x.f() ref_sar result (e.g., t1).

ref_sar - #rExist

Operator Stack

Semantic Action Stack



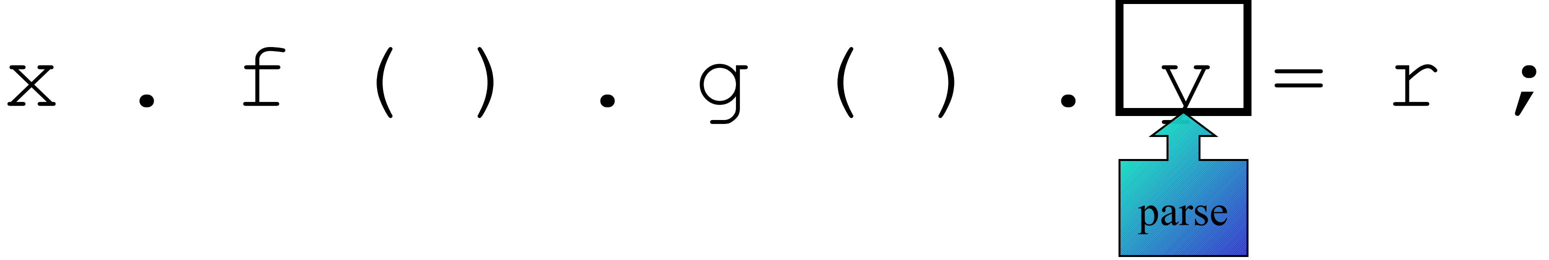
```
x . f ( ) . g ( ) . y = r ;
```

parse

ref_sar - #rExist

Operator Stack

Semantic Action Stack



```
x . f ( ) . g ( ) . y = r ;
```

y - #iPush

ref_sar - #rExist

Operator Stack

Semantic Action Stack

```
x.f().g().y = r;
```

- #rExist

y = SAS.pop()

ref_sar = SAS.pop(), t1.g() ref_sar

Test that variable y is a valid member of the ref_sar (the return type of function g)

Create ref_sar for top and next

SAS.push(ref_sar)

Note: The result of t1.g() will be placed in the Symbol Table at t2.

ref_sar - #rExist

Operator Stack

Semantic Action Stack

```
x . f ( ) . g ( ) . y = r ;
```

parse

= - #oPush

Operator Stack

ref_sar - #rExist

Semantic Action Stack

```
x.f().g().y=r;
```

parse

r - #iPush

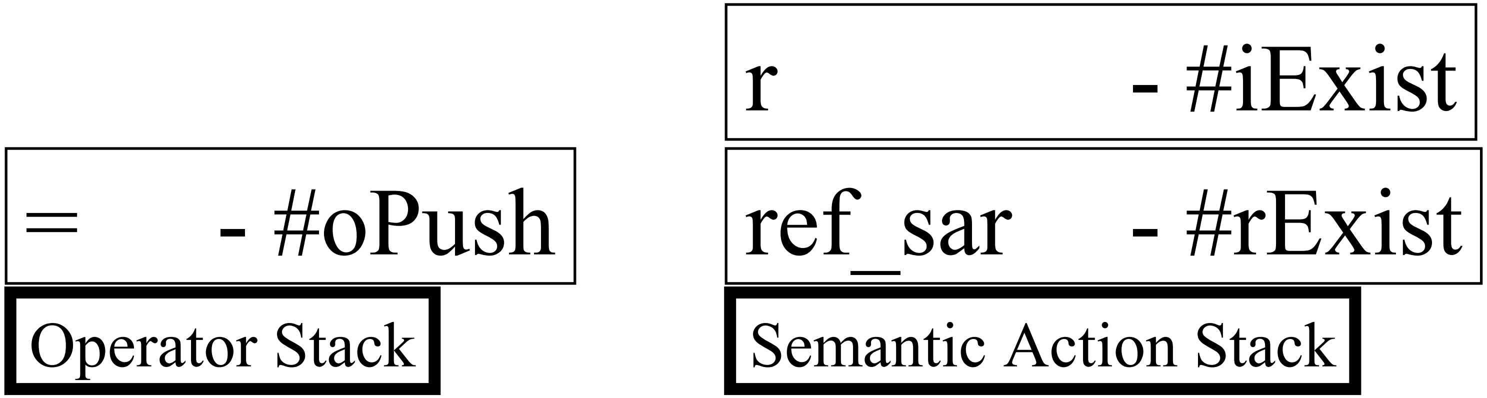
= - #oPush

ref_sar - #rExist

Operator Stack

Semantic Action Stack

```
x . f ( ) . g ( ) . y = r ;
```



= - #oPush

Operator Stack

r - #iExist

ref_sar - #rExist

Semantic Action Stack

x . f ( ) . g ( ) . y = r ;

parse

- #EOE

```
pop =   #=
        r = SAS.pop()
        ref_sar = SAS.pop()
        Test ref_sar (variable y which is a public member
                of the return type of function g ) = r is valid
```

Note: The ref_sar result is placed in the Symbol Table at t3.

Operator Stack

Semantic Action Stack

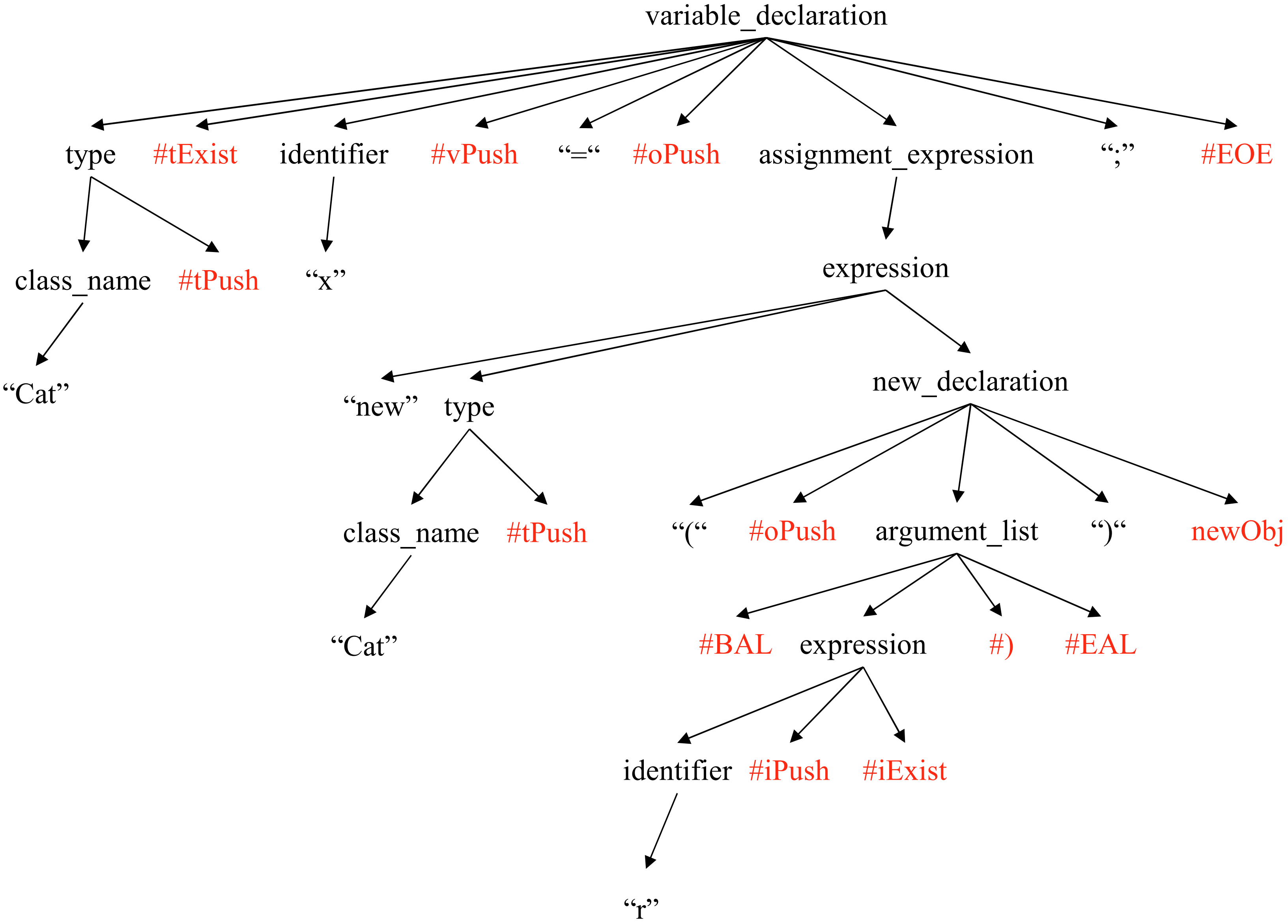
```
Cat x = new Cat( r ) ;
```

Creating a instance of a Class

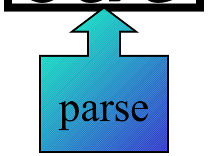
Operator Stack

Semantic Action Stack

```
Cat x = new Cat( r ) ;
```



```
Cat x = new Cat( r ) ;
```



Cat - #tPush

Operator Stack

Semantic Action Stack

```
Cat x = new Cat( r ) ;
```

- #tExists

Cat = SAS.pop()

Test that Cat is a valid Class

Operator Stack

Semantic Action Stack

```
Cat x = new Cat( r ) ;
```

parse

Operator Stack

Semantic Action Stack

Cat x $\boxed{=}$ new Cat(r) ;

x - #vPush

Operator Stack

Semantic Action Stack

```
Cat x = new Cat( r ) ;
```

parse

= - #oPush

Operator Stack

x - #vPush

Semantic Action Stack

```
Cat x = new Cat( r ) ;
```

parse

= - #oPush

Operator Stack

x - #vPush

Semantic Action Stack

```
Cat x = new Cat( r ) ;
```

parse

= - #oPush

Operator Stack

Cat - #tPush

x - #vPush

Semantic Action Stack

```
Cat x = new Cat( r ) ;
```

parse

(- #oPush

= - #oPush

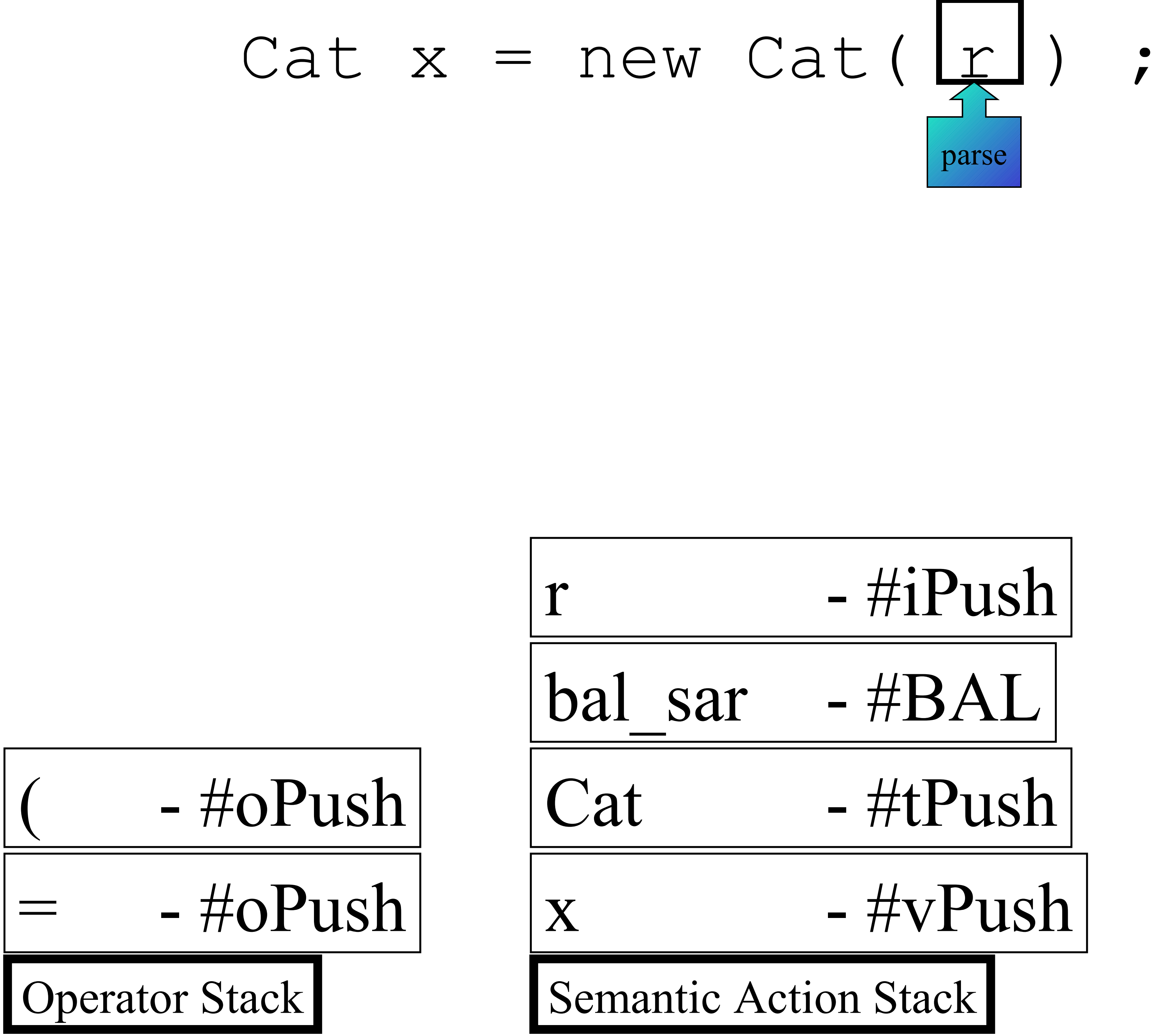
Operator Stack

bal_sar - #BAL

Cat - #tPush

x - #vPush

Semantic Action Stack



```
Cat x = new Cat( r ) ;
```

r - #iPush

bal_sar - #BAL

(- #oPush

Cat - #tPush

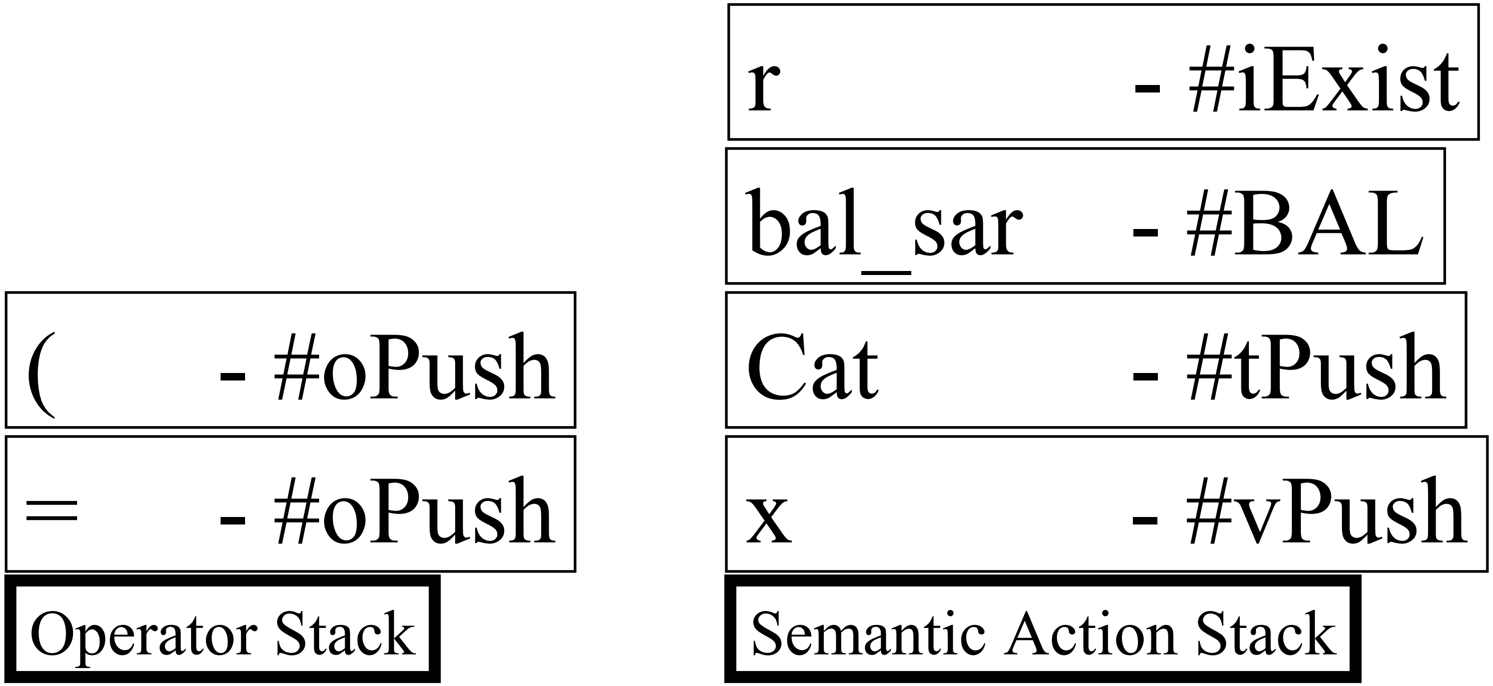
= - #oPush

x - #vPush

Operator Stack

Semantic Action Stack

```
Cat x = new Cat( r ) ;
```



(- #oPush

= - #oPush

Operator Stack

r - #iExist

bal_sar - #BAL

Cat - #tPush

x - #vPush

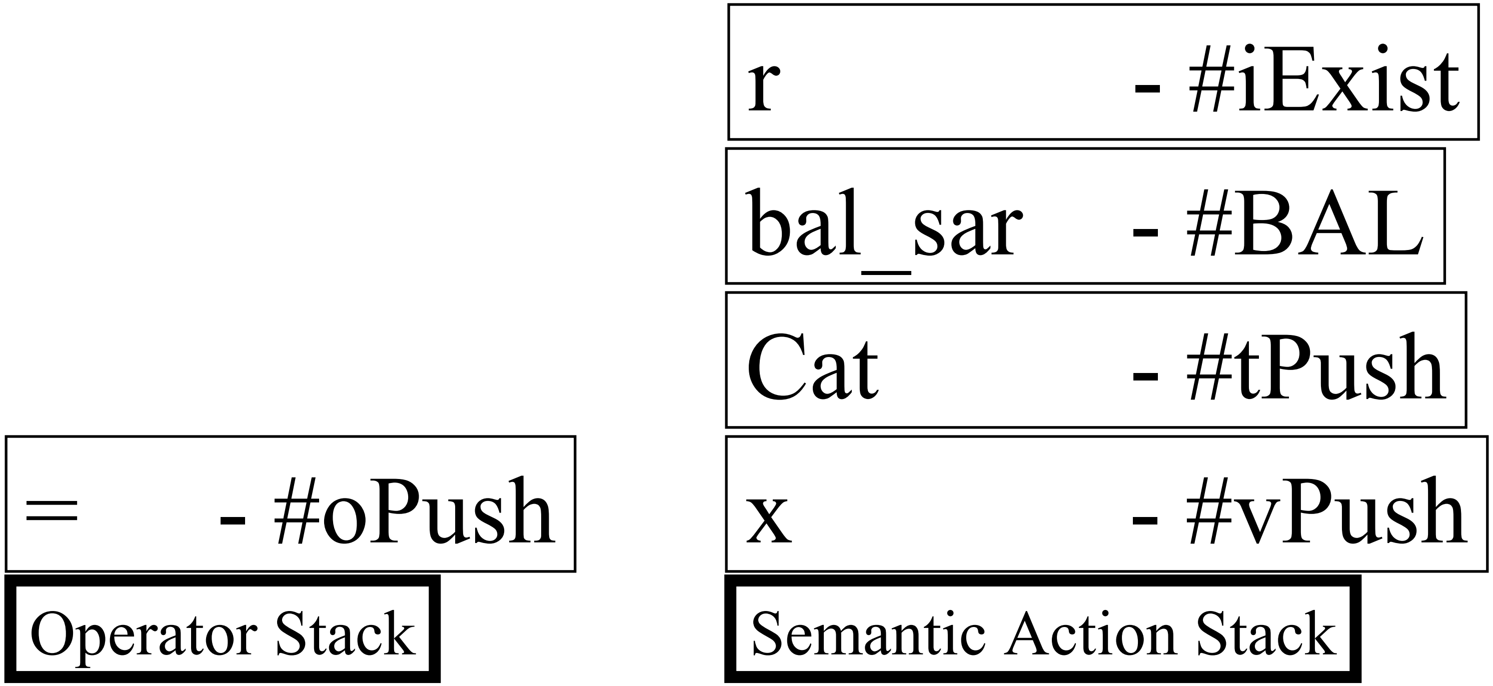
Semantic Action Stack

```
Cat x = new Cat( r ) ;
```

- #)

pop (

No infix to postfix conversion necessary on expression r



r - #iExist

bal_sar - #BAL

Cat - #tPush

x - #vPush

= - #oPush

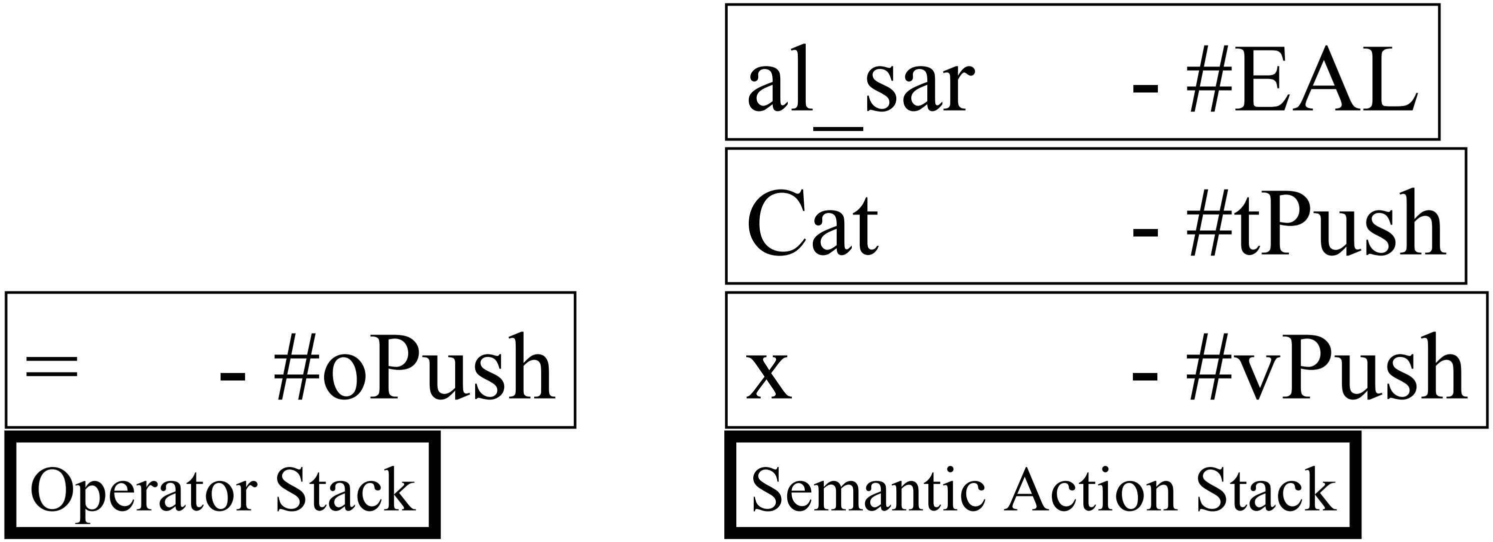
Operator Stack

Semantic Action Stack

```
Cat x = new Cat( r ) ;
```

- #EAL

r = SAS.pop()
bal_sar == SAS.pop()
Create al_sar for r
SAS.push(al_sar)



al_sar - #EAL

Cat - #tPush

x - #vPush

Semantic Action Stack

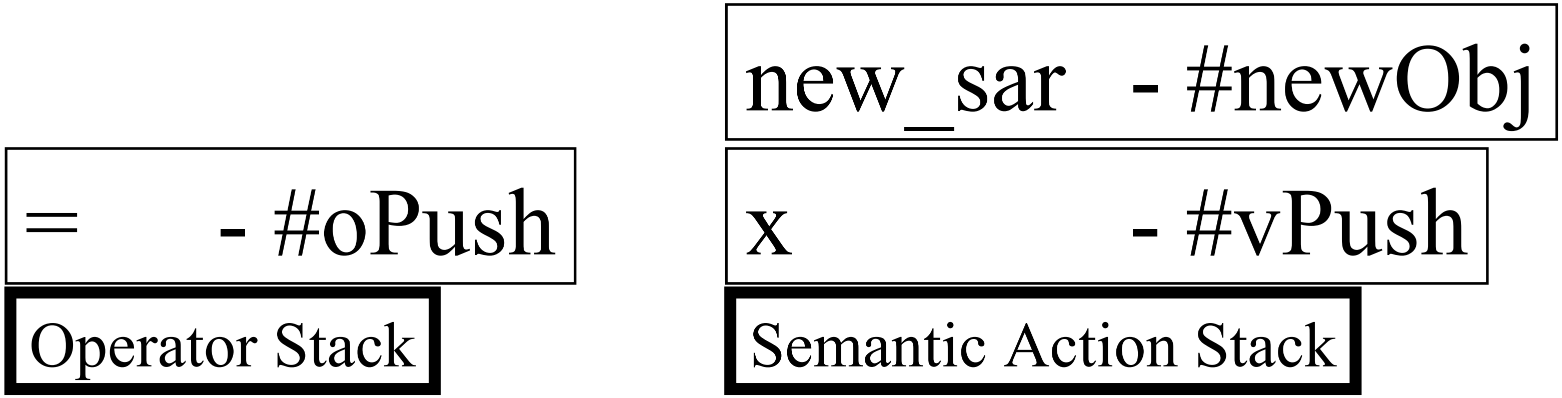
= - #oPush

Operator Stack

```
Cat x = new Cat( r ) ;
```

parse

```
- #newObj
al_sar = SAS.pop()
Cat type_sar == SAS.pop()
Test that class Cat has a constructor that
        takes the arguments of al_sar
Create a new_sar for the constructor of
        Cat with arguments al_sar
SAS.push(new_sar)
```



Operator Stack
= - #oPush

Semantic Action Stack
new_sar - #newObj
x - #vPush

```
Cat x = new Cat( r ) ;
```

parse

```
        - #EOE
pop =   #=
        new_sar = SAS.pop()
        x = SAS.pop()
        Test that x = new_sar is valid
Note: As new_sar
```

Operator Stack

Semantic Action Stack

```
Cat x[ ] = new Cat[ r ] ;
```

Creating an instance of an Array

Operator Stack

Semantic Action Stack

```
Cat x[ ] = new Cat[ r ] ;
```

parse

Operator Stack

Cat - #tPush

Semantic Action Stack

```
Cat x[ ] = new Cat[ r ] ;
```

- #tExists

Cat = SAS.pop()

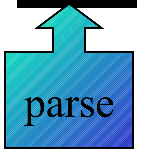
Test that Cat is a valid Class

Cat - #tPush

Operator Stack

Semantic Action Stack

```
Cat x[ ] = new Cat[ r ] ;
```



Operator Stack

Semantic Action Stack

```
Cat x[ ] = new Cat[ r ] ;
```

parse

Operator Stack

Semantic Action Stack

```
Cat x[ ] = new Cat[ r ] ;
```

parse

Operator Stack

Semantic Action Stack

```
Cat x[ ] = new Cat[ r ] ;
```

Note: While we show the vPush as x[], your stored information might be more like x, @:Cat. Indicating variable x is an array of type Cat.

x[] - #vPush

Operator Stack

Semantic Action Stack

```
Cat x[ ] = new Cat[ r ] ;
```

parse

= - #oPush

Operator Stack

x[] - #vPush

Semantic Action Stack

```
Cat x[ ] = new Cat[ r ] ;
```

parse

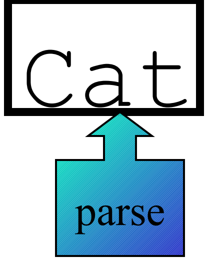
= - #oPush

Operator Stack

x[] - #vPush

Semantic Action Stack

```
Cat x[ ] = new Cat[ r ] ;
```



parse

= - #oPush

Operator Stack

Cat - #tPush

x[] - #vPush

Semantic Action Stack

```
Cat x[ ] = new Cat[ r ] ;
```

parse

[	- #oPush
=	- #oPush

Operator Stack

r	- #iPush
Cat	- #tPush
x[]	- #vPush

Semantic Action Stack

```
Cat x[ ] = new Cat[ r ] ;
```

[- #oPush

= - #oPush

Operator Stack

r - #iExist

Cat - #tPush

x[] - #vPush

Semantic Action Stack

```
Cat x[ ] = new Cat[ r ] ;
```

parse

- #]

pop [

No infix to postfix conversion necessary on expression r

= - #oPush

Operator Stack

r - #iExist

Cat - #tPush

x[] - #vPush

Semantic Action Stack

```
Cat x[ ] = new Cat[ r ] ;
```

parse

- #new[]

r = SAS.pop() this is an expression

Test that r is an integer

=	- #oPush

Operator Stack

Cat	- #tPush
x[]	- #vPush

Semantic Action Stack

```
Cat x[ ] = new Cat[ r ] ;
```

parse

- #new[]

  r = SAS.pop();

  Test that r is an integer

  Cat type_sar = SAS.pop()

  Test that an array of type Cat can be can be created

  Create a new_sar for an array of type Cat with r elements

  SAS.push(new_sar)

Note: A new_sar is used when constructing a new object or new array. This is similar to using a ref_sar when referencing a member attribute or member function.

= - #oPush

Operator Stack

new_sar - #new[]

x[] - #vPush

Semantic Action Stack

```
Cat x[ ] = new Cat[ r ] ;
```

parse

- #EOE

pop = #=

new_sar = SAS.pop()

x[] = SAS.pop()

Test that x[] = new_sar is valid

Operator Stack

Semantic Action Stack

```
f ( r * 3 , g < k ) ;
```

Function call with infix to postfix conversion of arguments

Operator Stack

Semantic Action Stack

```
f ( r * 3 , g < k ) ;
```

parse

f - #iPush

Operator Stack

Semantic Action Stack

```
f ( r * 3 , g < k ) ;
```

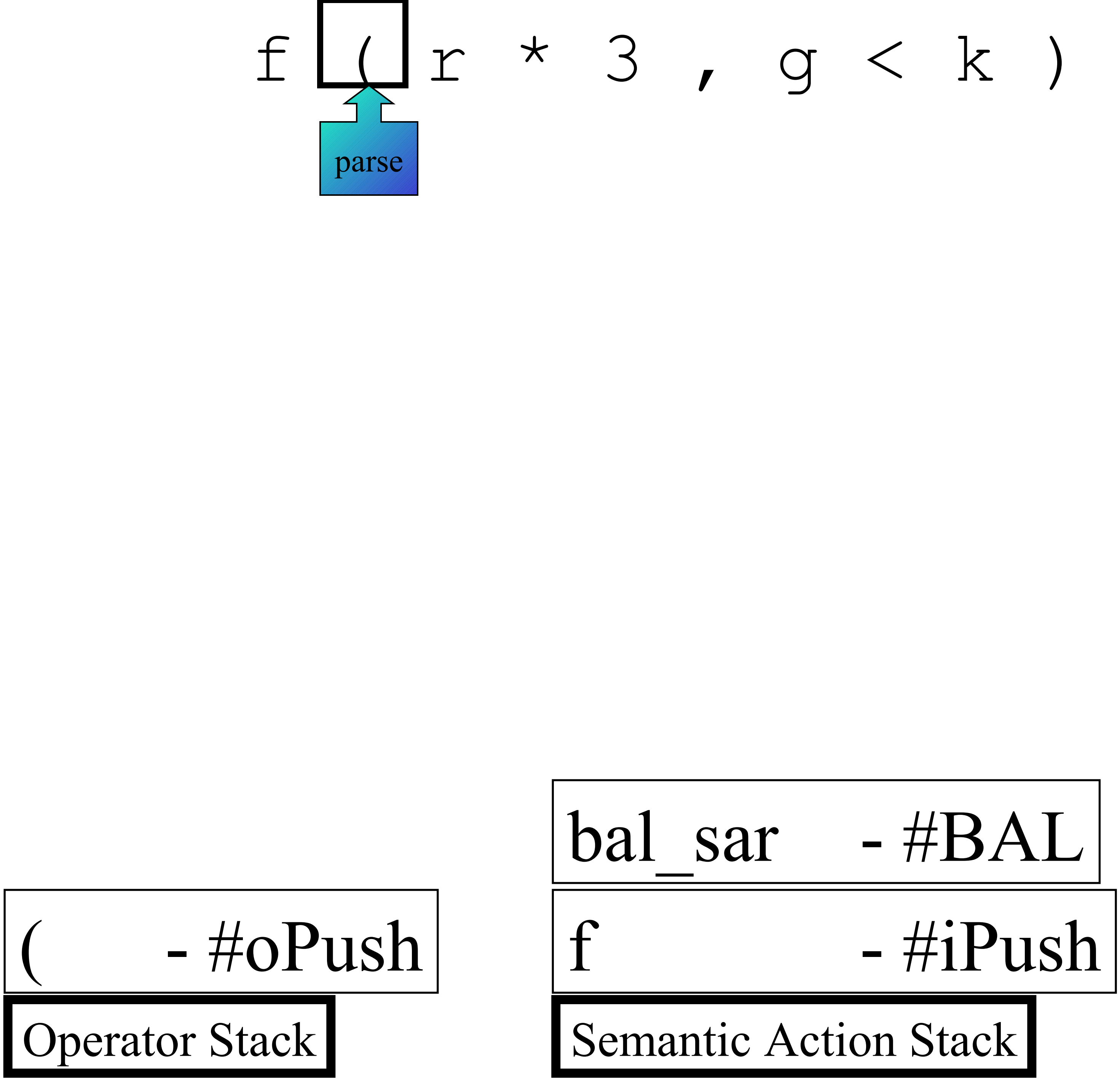
parse

(- #oPush

Operator Stack

f - #iPush

Semantic Action Stack



```
f ( r * 3 , g < k ) ;
```

parse

bal_sar - #BAL

f - #iPush

(- #oPush

Operator Stack

Semantic Action Stack

```
f ( r * 3 , g < k ) ;
```

parse

(- #oPush

Operator Stack

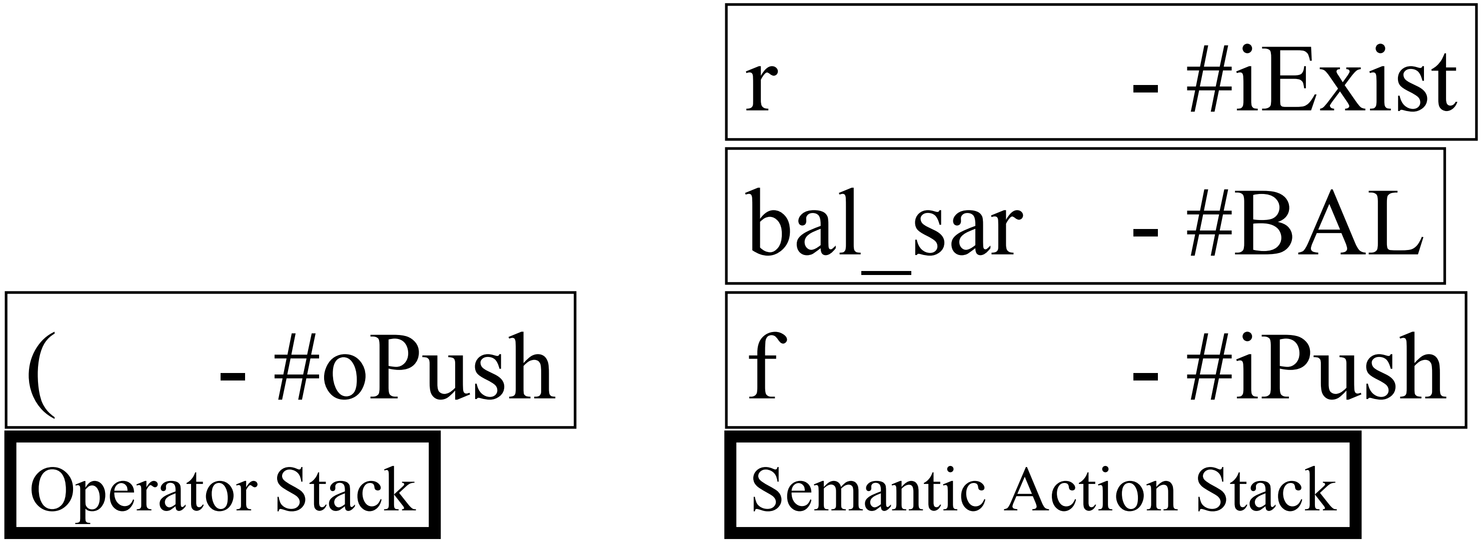
r - #iPush

bal_sar - #BAL

f - #iPush

Semantic Action Stack

```
f ( r * 3 , g < k ) ;
```



r - #iExist

bal_sar - #BAL

(- #oPush

f - #iPush

Operator Stack

Semantic Action Stack

```
f ( r * 3 , g < k ) ;
```

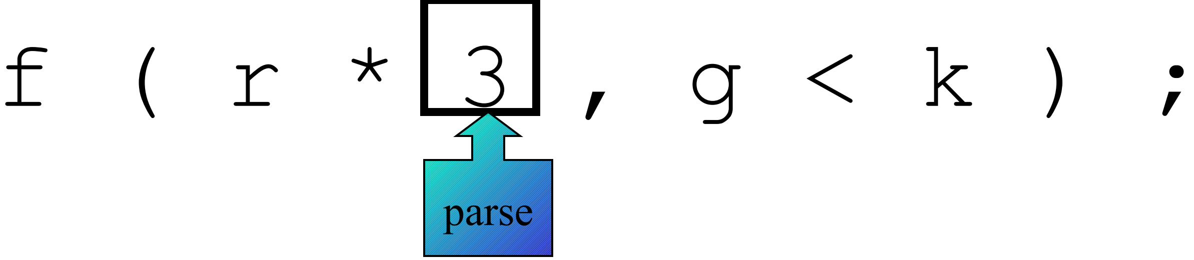
parse

*	- #oPush
(	- #oPush

Operator Stack

r	- #iExist
bal_sar	- #BAL
f	- #iPush

Semantic Action Stack



f (r * 3 , g < k) ;

parse

Operator Stack	
*	- #oPush
(	- #oPush

Semantic Action Stack	
3	- #lPush
r	- #iExist
bal_sar	- #BAL
f	- #iPush

```
f ( r * 3 , g < k ) ;
```

parse

- #,
pop * #*
3 = SAS.pop()
r = SAS.pop()
Test that r * 3 is valid
t7 = r * 3
SAS.push(t7)

(	- #oPush

Operator Stack

t7	- #,
bal_sar	- #BAL
f	- #iPush

Semantic Action Stack

```
f ( r * 3 , g < k ) ;
```

parse

g - #iPush

t7 - #,

bal_sar - #BAL

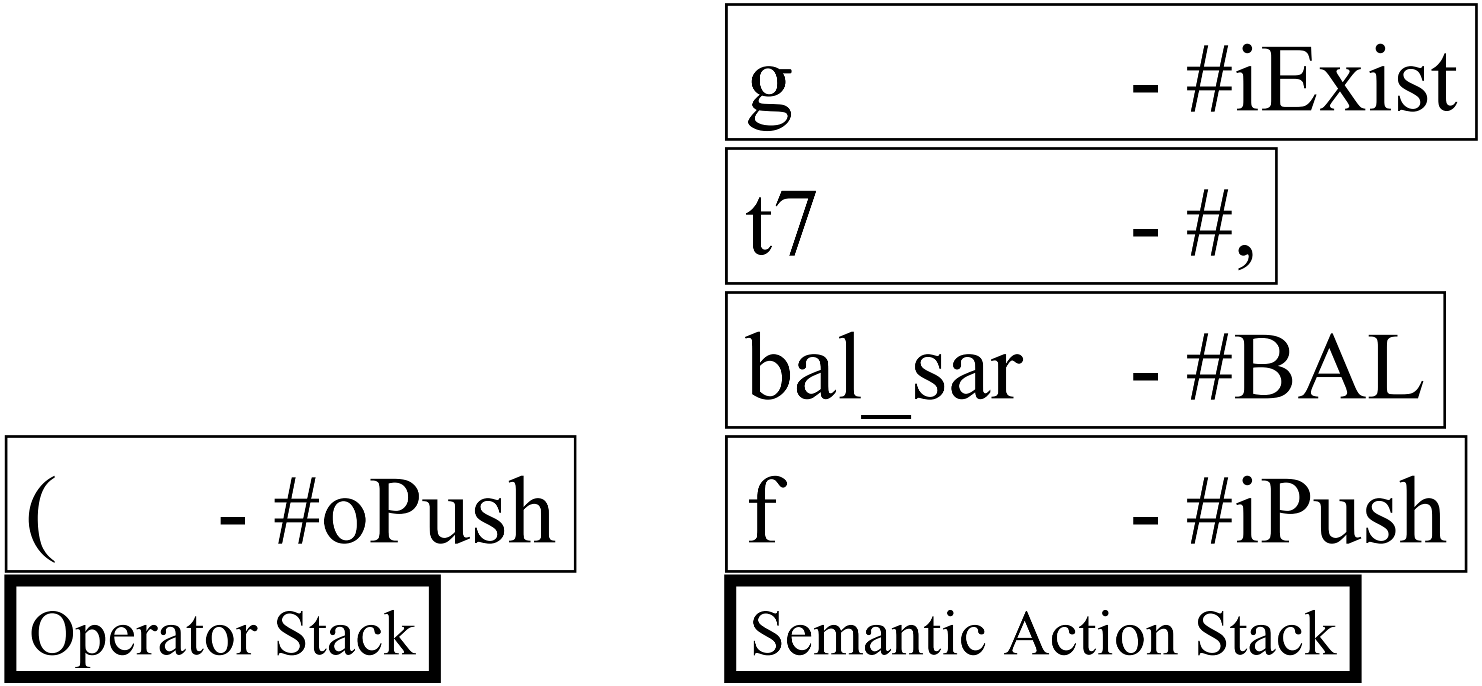
f - #iPush

(- #oPush

Operator Stack

Semantic Action Stack

```
f ( r * 3 , g < k ) ;
```



g - #iExist

t7 - #,

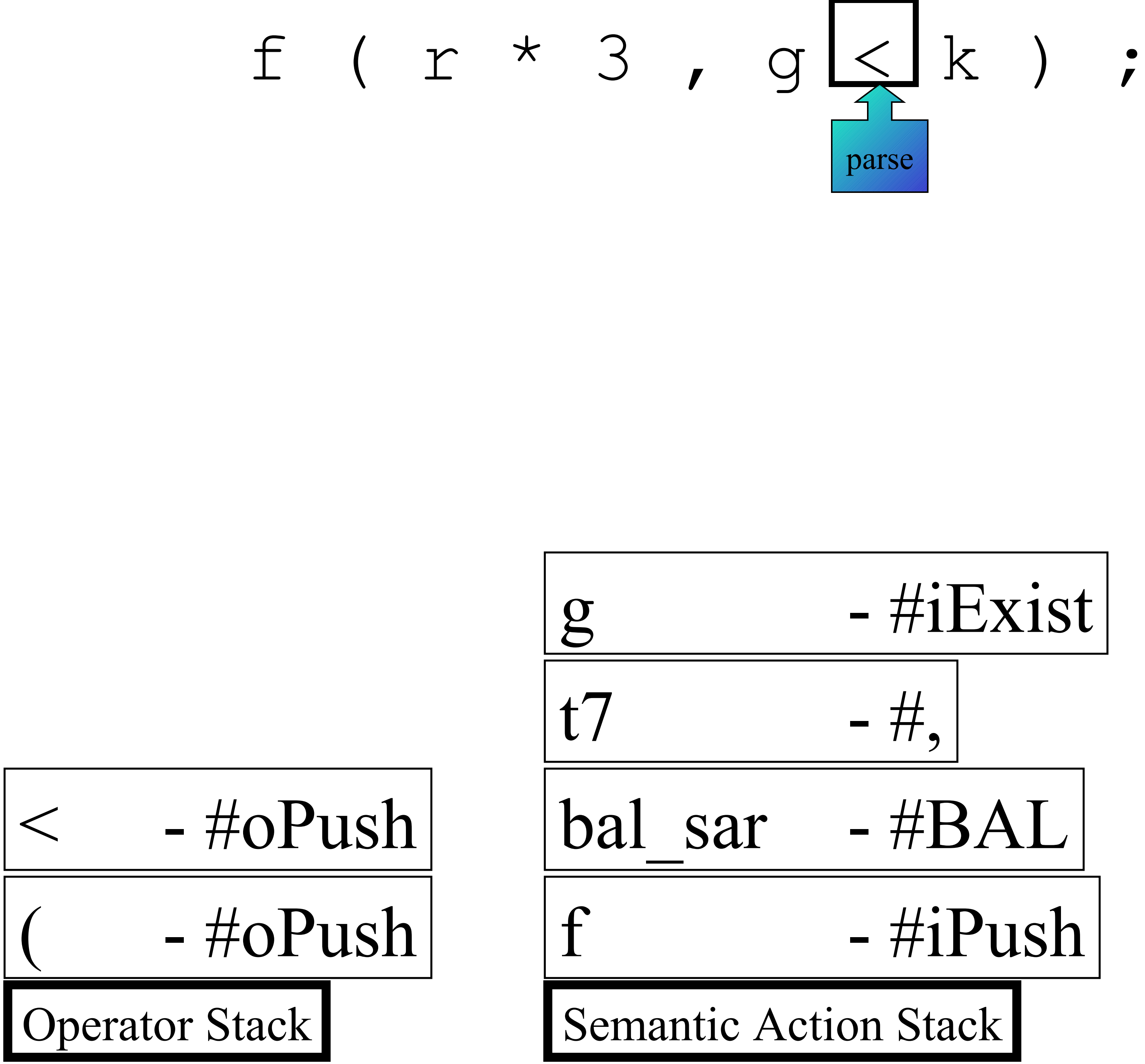
bal_sar - #BAL

f - #iPush

Semantic Action Stack

(- #oPush

Operator Stack



```
f ( r * 3 , g < k ) ;
```

parse

< - #oPush

(- #oPush

Operator Stack

g - #iExist

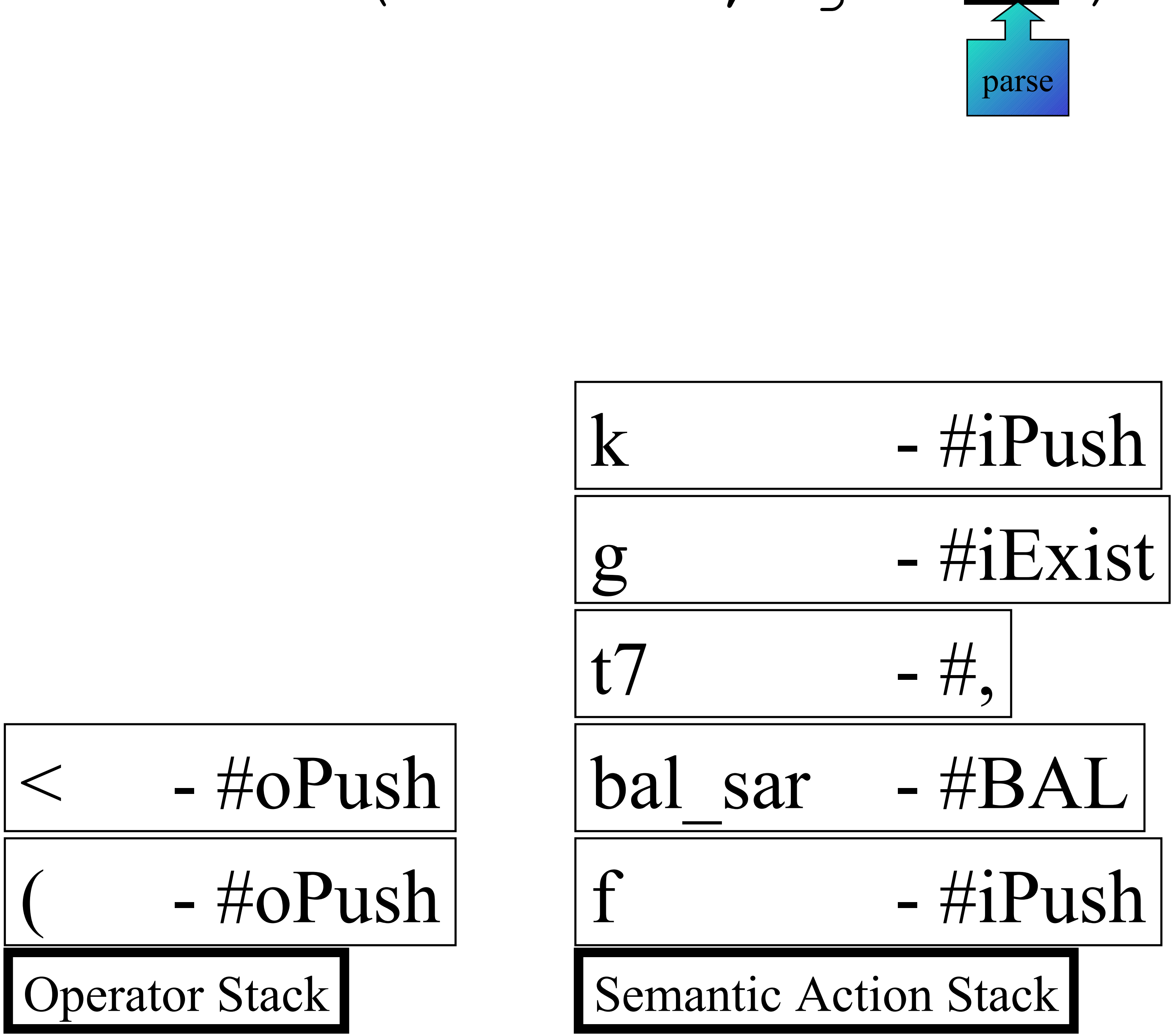
t7 - #,

bal_sar - #BAL

f - #iPush

Semantic Action Stack

```
f ( r * 3 , g < k ) ;
```



parse

k - #iPush

g - #iExist

t7 - #,

< - #oPush

bal_sar - #BAL

(- #oPush

f - #iPush

Operator Stack

Semantic Action Stack

```
f ( r * 3 , g < k ) ;
```

<	- #oPush
(	- #oPush

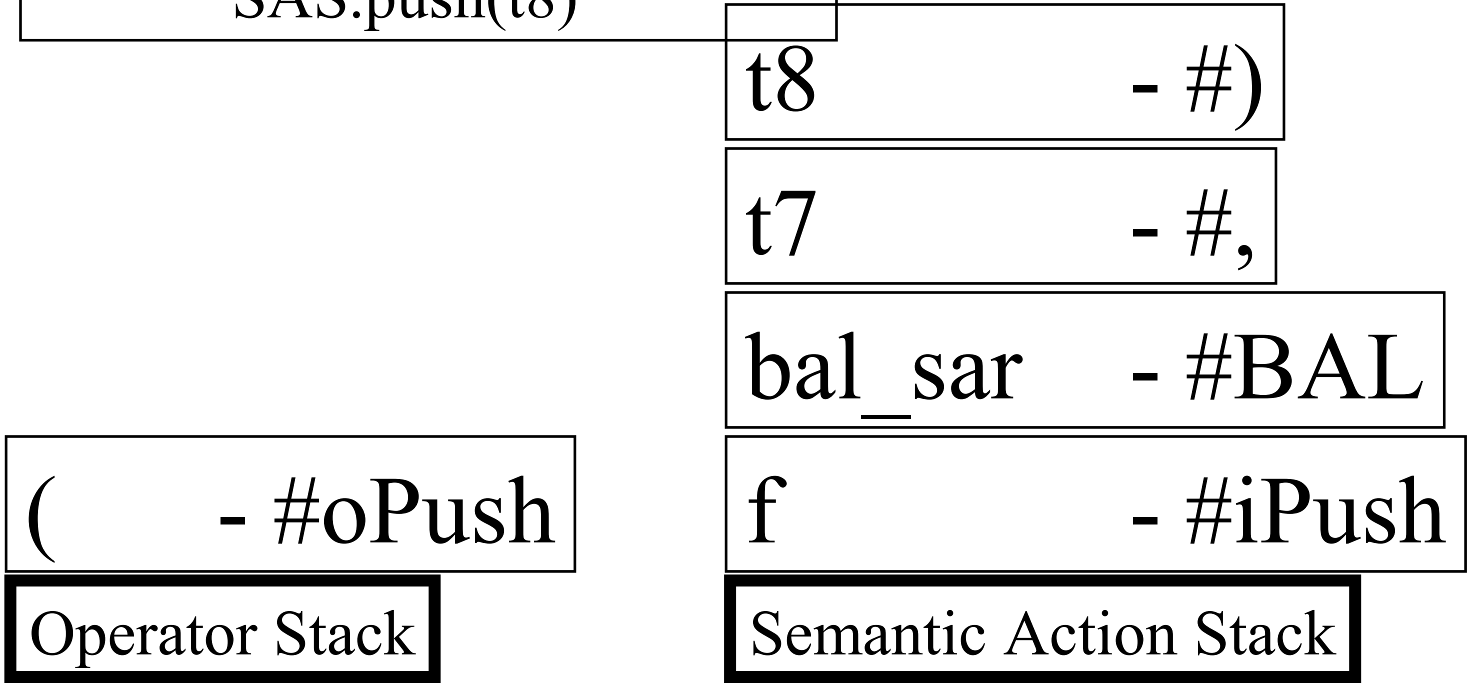
Operator Stack

k	- #iExist
g	- #iExist
t7	- #,
bal_sar	- #BAL
f	- #iPush

Semantic Action Stack

```
f ( r * 3 , g < k ) ;
```

```
        - #)
pop <   #<
        k = SAS.pop()
        g = SAS.pop()
        Test that g < k is valid
        t8 = g < k
        SAS.push(t8)
```



t8 - #)

t7 - #,

bal_sar - #BAL

f - #iPush

(- #oPush

Operator Stack

Semantic Action Stack

```
f ( r * 3 , g < k ) ;
```

- #)

pop (

t8 - #)

t7 - #,

bal_sar - #BAL

f - #iPush

Operator Stack

Semantic Action Stack

```
f ( r * 3 , g < k ) ;
```

- #EAL

t8 = SAS.pop()

t7 = SAS.pop()

bal_sar == SAS.pop()

Create al_sar for t7 & t8

SAS.push(al_sar)

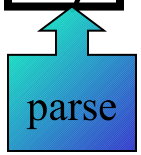
al_sar - #EAL

f - #iPush

Operator Stack

Semantic Action Stack

```
f ( r * 3 , g < k ) ;
```



```
- #func
al_sar = SAS.pop()
f = SAS.pop()
Add f & al_sar to func_sar
SAS.push(func_sar)
```

Operator Stack

func_sar - #func

Semantic Action Stack

f (r * 3 , g < k) ;

f(t7,t8) id_sar - #iExist

Operator Stack

Semantic Action Stack

```
f ( r * 3 , g < k ) ;
```

parse

- #EOE

Operator Stack

Semantic Action Stack

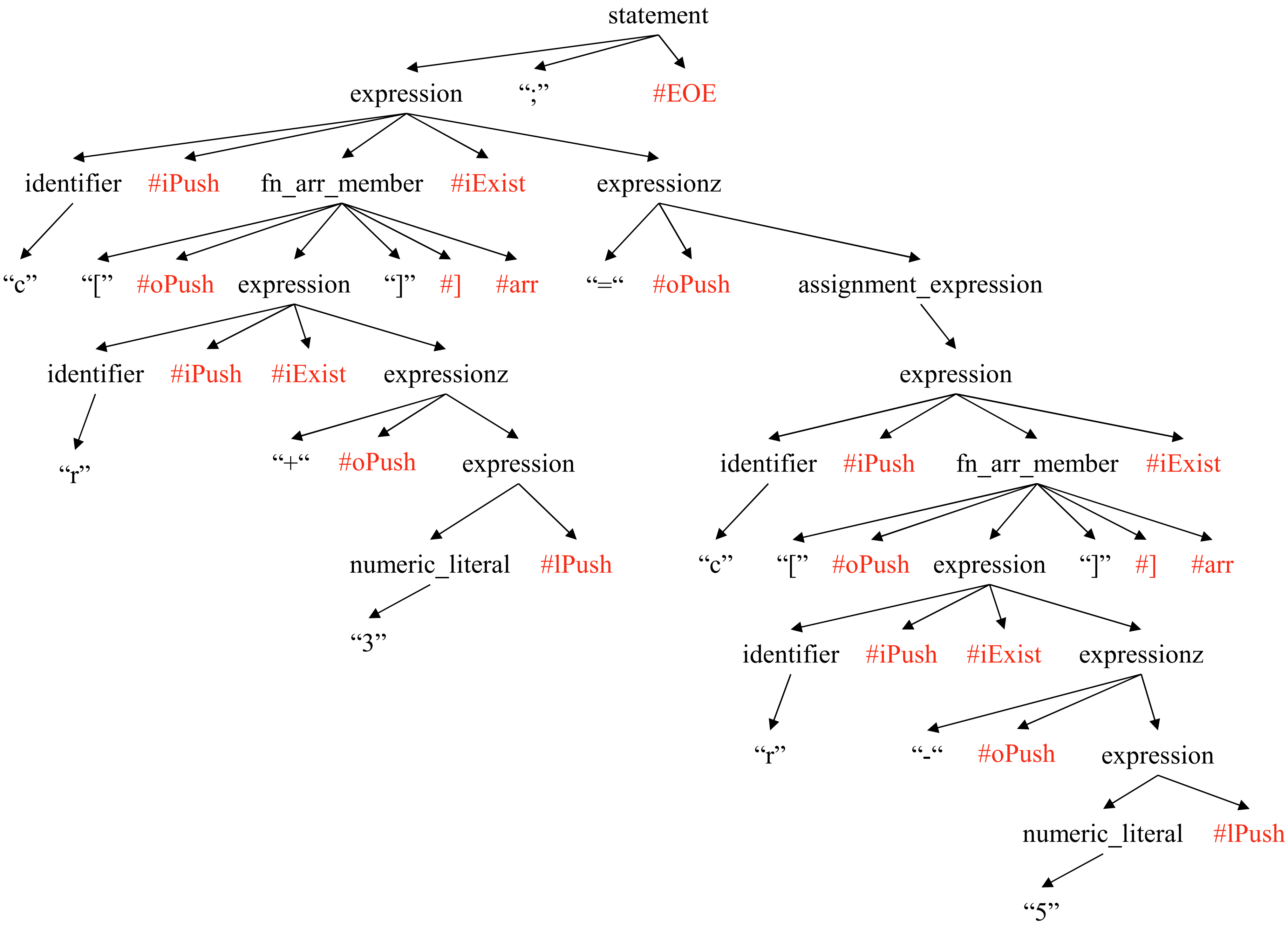
```
c [ r + 3 ] = c [ r - 5 ] ;
```

Array to Array assignment with infix to postfix conversion of index expressions

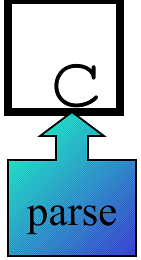
Operator Stack

Semantic Action Stack

```
c [ r + 3 ] = c [ r - 5 ] ;
```


statement
expression
“;”
#EOE
identifier
#iPush
fn_arr_member
#iExist
expressionz
“c”
“[”
#oPush
expression
“]”
#]
#arr
“=“
#oPush
assignment_expression
identifier
#iPush
#iExist
expressionz
expression
“r”
“+“
#oPush
expression
identifier
#iPush
fn_arr_member
#iExist
numeric_literal
#lPush
“c”
“[”
#oPush
expression
“]”
#]
#arr
“3”
identifier
#iPush
#iExist
expressionz
“r”
“-“
#oPush
expression
numeric_literal
#lPush
“5”

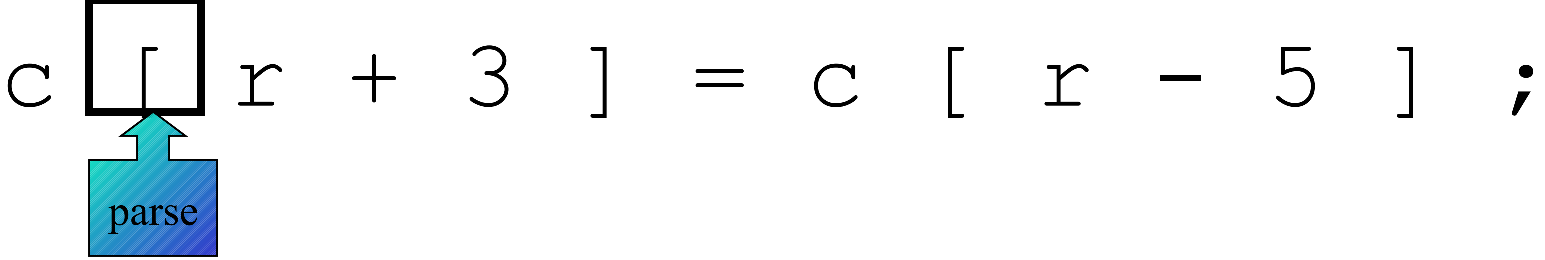
```
c [ r + 3 ] = c [ r - 5 ] ;
```



c - #iPush

Operator Stack

Semantic Action Stack



```
c [ r + 3 ] = c [ r - 5 ] ;
```

parse

[- #oPush

Operator Stack

c - #iPush

Semantic Action Stack

```
c [ r + 3 ] = c [ r - 5 ] ;
```

parse

[- #oPush

Operator Stack

r - #iPush

c - #iPush

Semantic Action Stack

```
c [ r + 3 ] = c [ r - 5 ] ;
```

[- #oPush

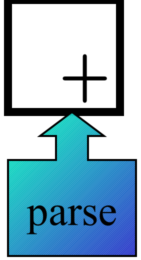
Operator Stack

r - #iExist

c - #iPush

Semantic Action Stack

```
c [ r + 3 ] = c [ r - 5 ] ;
```



+	- #oPush
[	- #oPush

Operator Stack

r	- #iExist
c	- #iPush

Semantic Action Stack

```
c [ r + 3 ] = c [ r - 5 ] ;
```

parse

+	- #oPush
[	- #oPush

Operator Stack

3	- #lPush
r	- #iExist
c	- #iPush

Semantic Action Stack

```
c [ r + 3 ] = c [ r - 5 ] ;
```

parse

- #]

pop + #+

3 = SAS.pop()

r = SAS.pop()

Test that r + 3 is valid

t23 = r + 3

SAS.push(t23)

[- #oPush

Operator Stack

t23 - #]

c - #iPush

Semantic Action Stack

```
c [ r + 3 ] = c [ r - 5 ] ;
```

parse

- #]

pop [

t23 - #]

c - #iPush

Operator Stack

Semantic Action Stack

```
c [ r + 3 ] = c [ r - 5 ] ;
```

parse

- #arr

t23 = SAS.pop()

c = SAS.pop()

Test that t23 is an integer

Create an arr_sar for the array c with index t23

SAS.push(arr_sar)

arr_sar - #arr

Operator Stack

Semantic Action Stack

```
c[r+3] = c[r-5];
```

- #iExist

arr_sar = SAS.pop()
Test that an arry c exist in the current scope
Create an id_sar for the array c with index t23
SAS.push(c[t23])

Note: How the id_sar must be able to work with a simple variable (e.g., r) , a function call (e.g., f(t7,t8)) or an array reference (e.g., c[t23]).

Think of a sar as a base class, with an id_sar as a derived class and the variable, function and array sars as classes derived from id_sar.

c[t23] id_sar - #iExist

Operator Stack

Semantic Action Stack

```
c [ r + 3 ] = c [ r - 5 ] ;
```

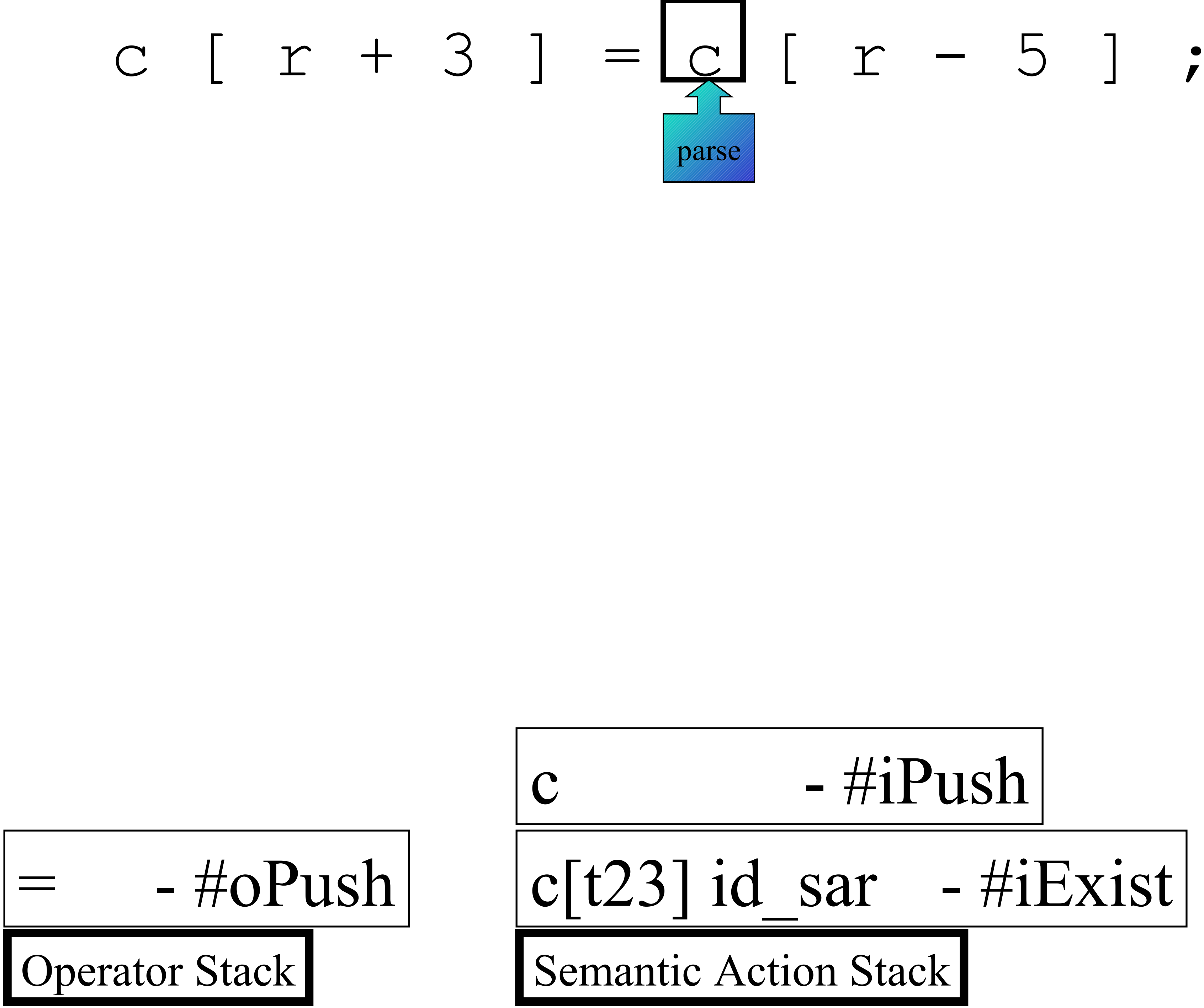
parse

= - #oPush

Operator Stack

c[t23] id_sar - #iExist

Semantic Action Stack



```
c [ r + 3 ] = c [ r - 5 ] ;
```

= - #oPush

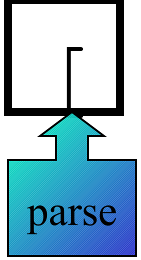
Operator Stack

c - #iPush

c[t23] id_sar - #iExist

Semantic Action Stack

```
c [ r + 3 ] = c [ r - 5 ] ;
```



[- #oPush

= - #oPush

Operator Stack

c - #iPush

c[t23] id_sar - #iExist

Semantic Action Stack

```
c [ r + 3 ] = c [ r - 5 ] ;
```

parse

[- #oPush

= - #oPush

Operator Stack

r - #iPush

c - #iPush

c[t23] id_sar - #iExist

Semantic Action Stack

c [r + 3] = c [r $\boxed{-}$ 5] ;

[- #oPush

= - #oPush

Operator Stack

r - #iExist

c - #iPush

c[t23] id_sar - #iExist

Semantic Action Stack

```
c [ r + 3 ] = c [ r - 5 ] ;
```

parse

- - #oPush

[- #oPush

= - #oPush

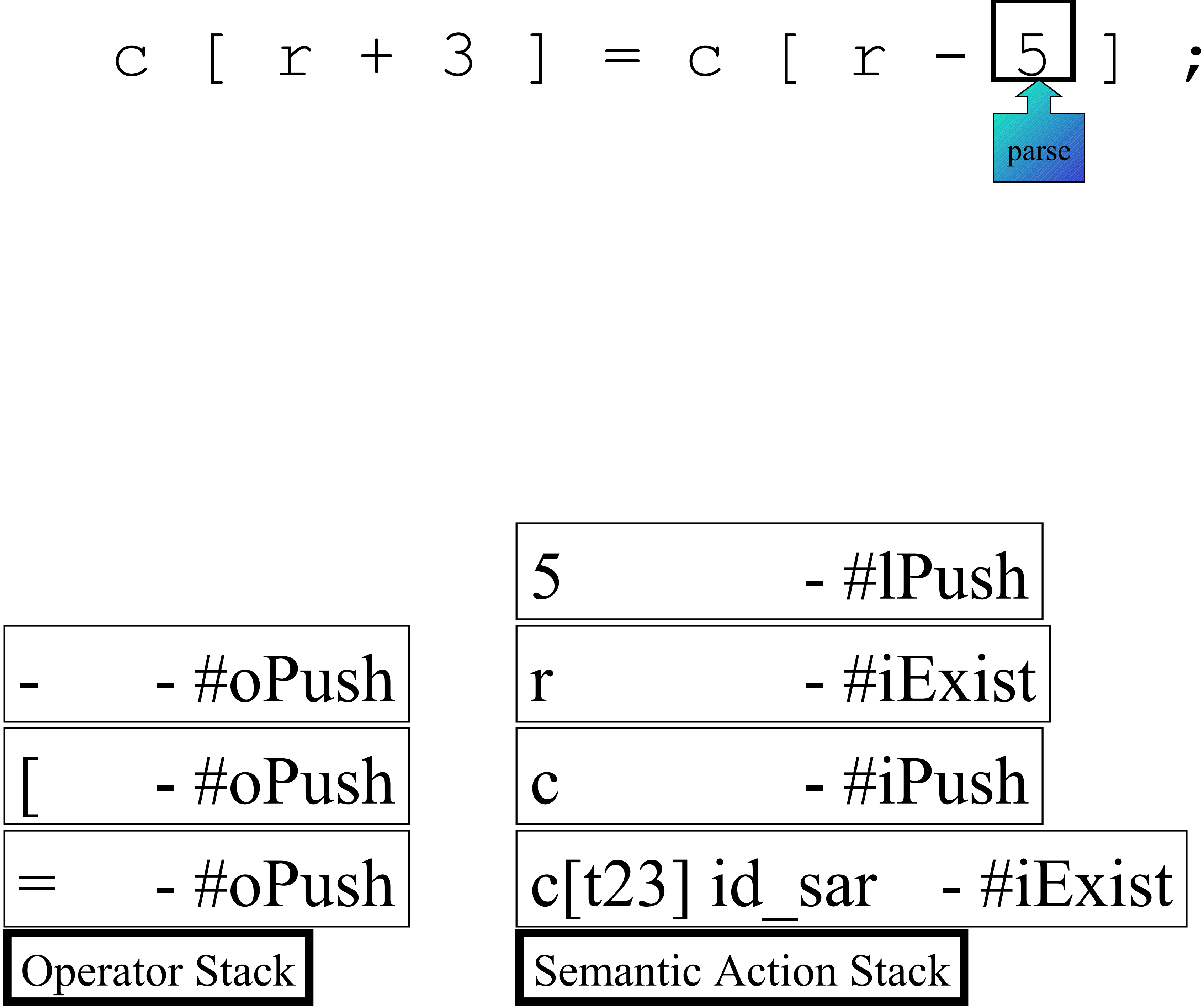
Operator Stack

r - #iExist

c - #iPush

c[t23] id_sar - #iExist

Semantic Action Stack



```
c [ r + 3 ] = c [ r - 5 ] ;
```

parse

-	- #oPush
[	- #oPush
=	- #oPush

Operator Stack

5	- #lPush
r	- #iExist
c	- #iPush
c[t23] id_sar	- #iExist

Semantic Action Stack

```
c [ r + 3 ] = c [ r - 5 ] ;
```

parse

- #]

pop - #-

5 = SAS.pop()

r = SAS.pop()

Test that r - 1 is valid

t24 = r - 1

SAS.push(t24)

[- #oPush

= - #oPush

Operator Stack

t24 - #]

c - #iPush

c[t23] id_sar - #iExist

Semantic Action Stack

```
c [ r + 3 ] = c [ r - 5 ] ;
```

parse

- #]

pop [

t24 - #]

c - #iPush

c[t23] id_sar - #iExist

Semantic Action Stack

= - #oPush

Operator Stack

```
c [ r + 3 ] = c [ r - 5 ] ;
```

parse

- #arr

t24 = SAS.pop()

c = SAS.pop()

Test that t24 is an integer

Create an arr_sar for the array c with index t24

SAS.push(arr_sar)

= - #oPush

Operator Stack

arr_sar - #arr

c[t23] id_sar - #iExist

Semantic Action Stack

```
c [ r + 3 ] = c [ r - 5 ] ;
```

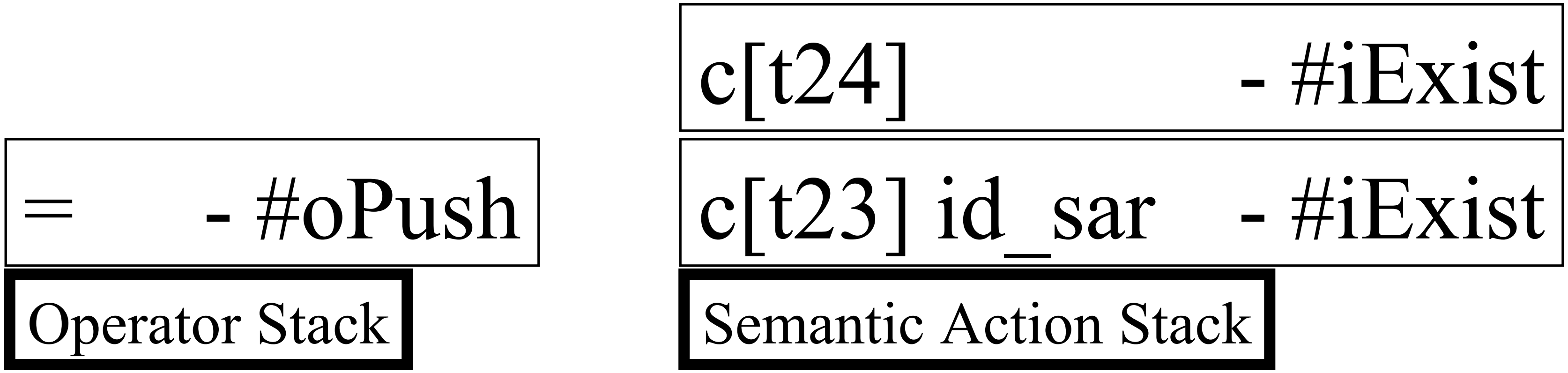
- #iExist

arr_sar = SAS.pop()

Test that an arry c exist in the current scope

Create an id_sar for the array c with index t24

SAS.push(c[t24])



c[t24] - #iExist

= - #oPush

c[t23] id_sar - #iExist

Operator Stack

Semantic Action Stack

```
c [ r + 3 ] = c [ r - 5 ] ;
```

parse

- #EOE

pop = #=

c[t24] = SAS.pop()

c[t23] = SAS.pop()

Test that c[t23] = c[t24] is valid

Operator Stack

Semantic Action Stack

Multi-Threaded Semantic Actions

- Implementing Multi-Threaded features is not part of CS4490 only parsing the statements is require.
- DO NOT Implement these features until you have completed your Compiler!

```
spawn d . go ( i ) set i;
```

Spawn a Thread

Operator Stack

Semantic Action Stack

```
spawn d . go ( i ) set i;
```

d - #iPush

Operator Stack

Semantic Action Stack

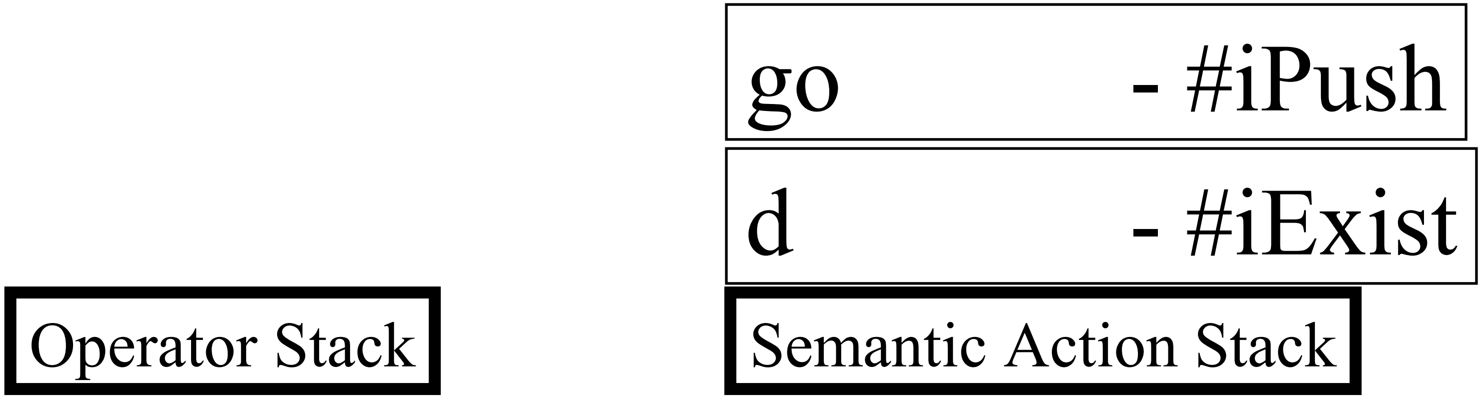
```
spawn d . go ( i ) set i;
```

d - #iExists

Operator Stack

Semantic Action Stack

```
spawn d . go ( i ) set i;
```



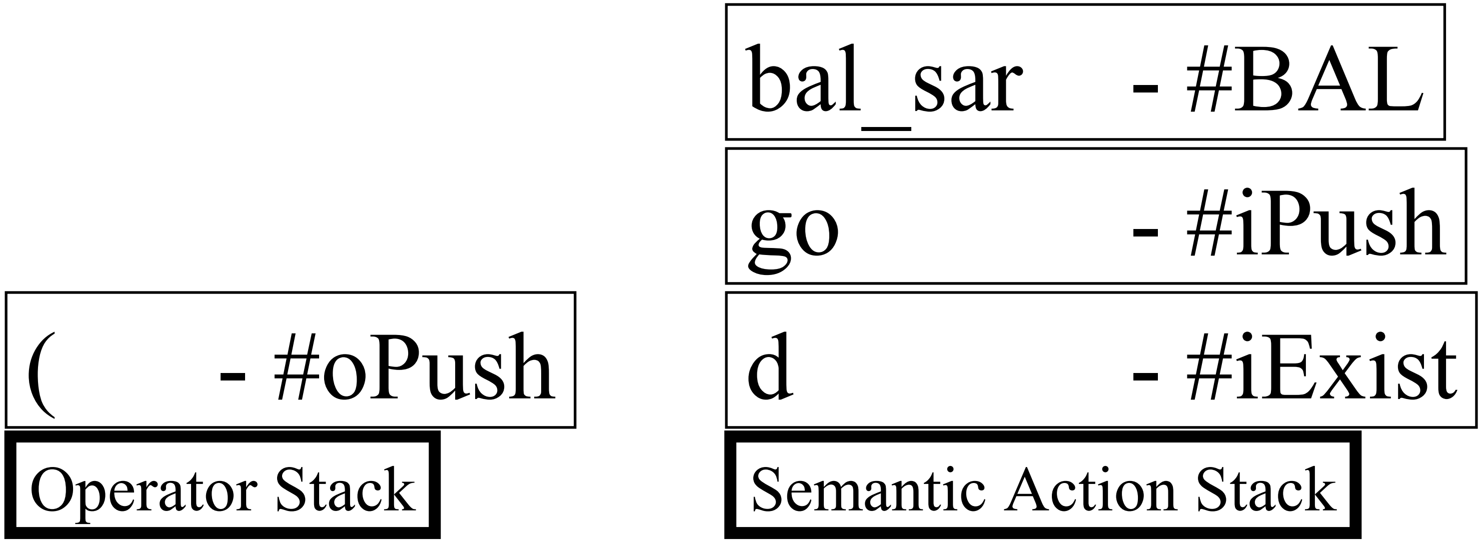
go - #iPush

d - #iExist

Operator Stack

Semantic Action Stack

```
spawn d . go ( i ) set i;
```



bal_sar - #BAL

go - #iPush

d - #iExist

Semantic Action Stack

(- #oPush

Operator Stack

```
spawn d . go ( i ) set i;
```

(- #oPush

Operator Stack

i - #iPush

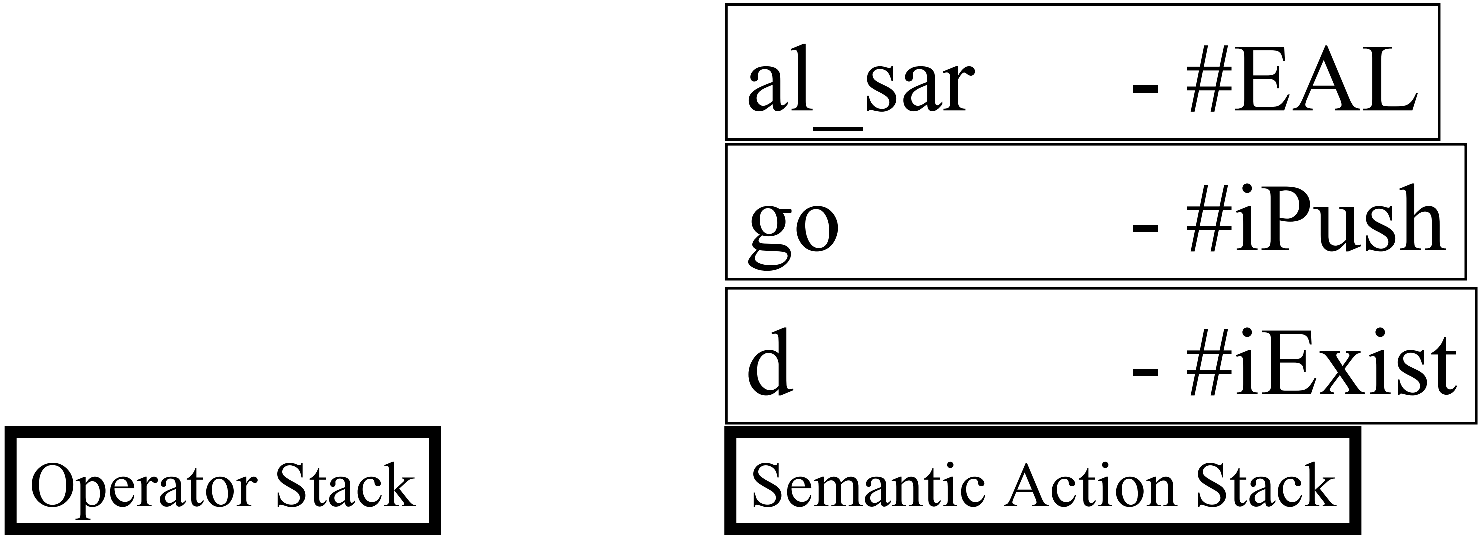
bal_sar - #BAL

go - #iPush

d - #iExist

Semantic Action Stack

```
spawn d . go ( i ) set i;
```



al_sar - #EAL

go - #iPush

d - #iExist

Operator Stack

Semantic Action Stack

```
spawn d . go ( i ) set i;
```

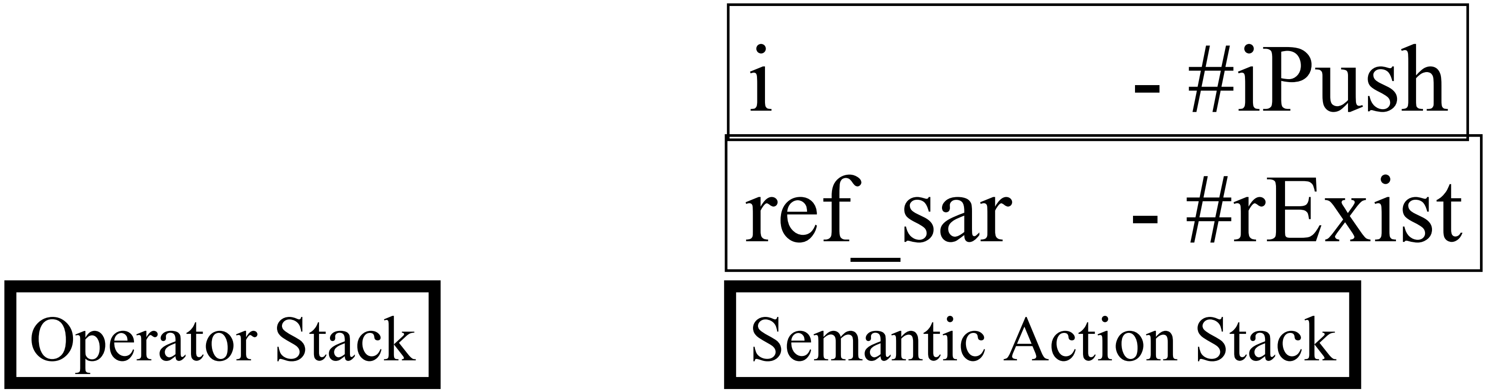
func_sar - #func

d - #iExist

Operator Stack

Semantic Action Stack

```
spawn d . go ( i ) set i;
```



i - #iPush

ref_sar - #rExist

Operator Stack

Semantic Action Stack

```
spawn d . go ( i ) set i;
```

- #spawn

sar = SAS.pop()

sar must be an integer variable defined in the local scope (just like #iExists)

ref_sar == SAS.pop()

Test that the ref_sar is a valid function call. Note: The variable will be used to set the status of the spawned thread (-1 failed, > 0 for the thread spawned successfully). It is possible for the variable to be passed as a parameter to the function allowing the function to know it's thread id (cool).

Operator Stack

Semantic Action Stack