A Theory of Name Resolution

Pierre Neron[1]

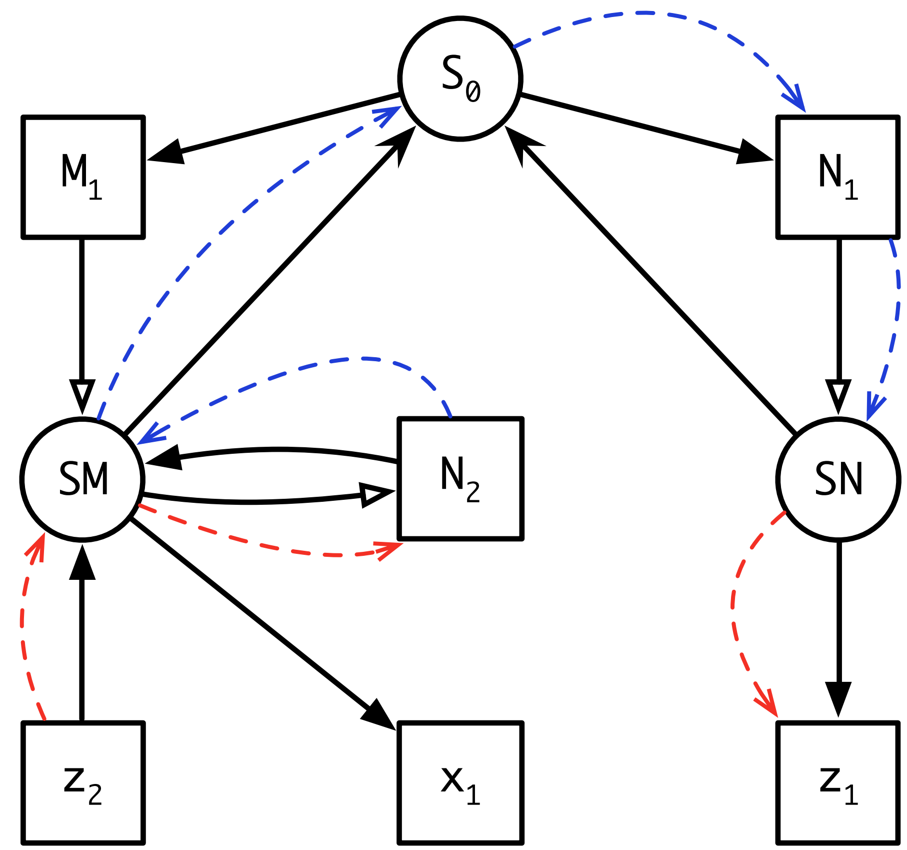
Andrew Tolmach[2]

Eelco Visser[1]

Guido Wachsmuth[1]

1 TU Delft Delft University of Technology

2 Portland State University

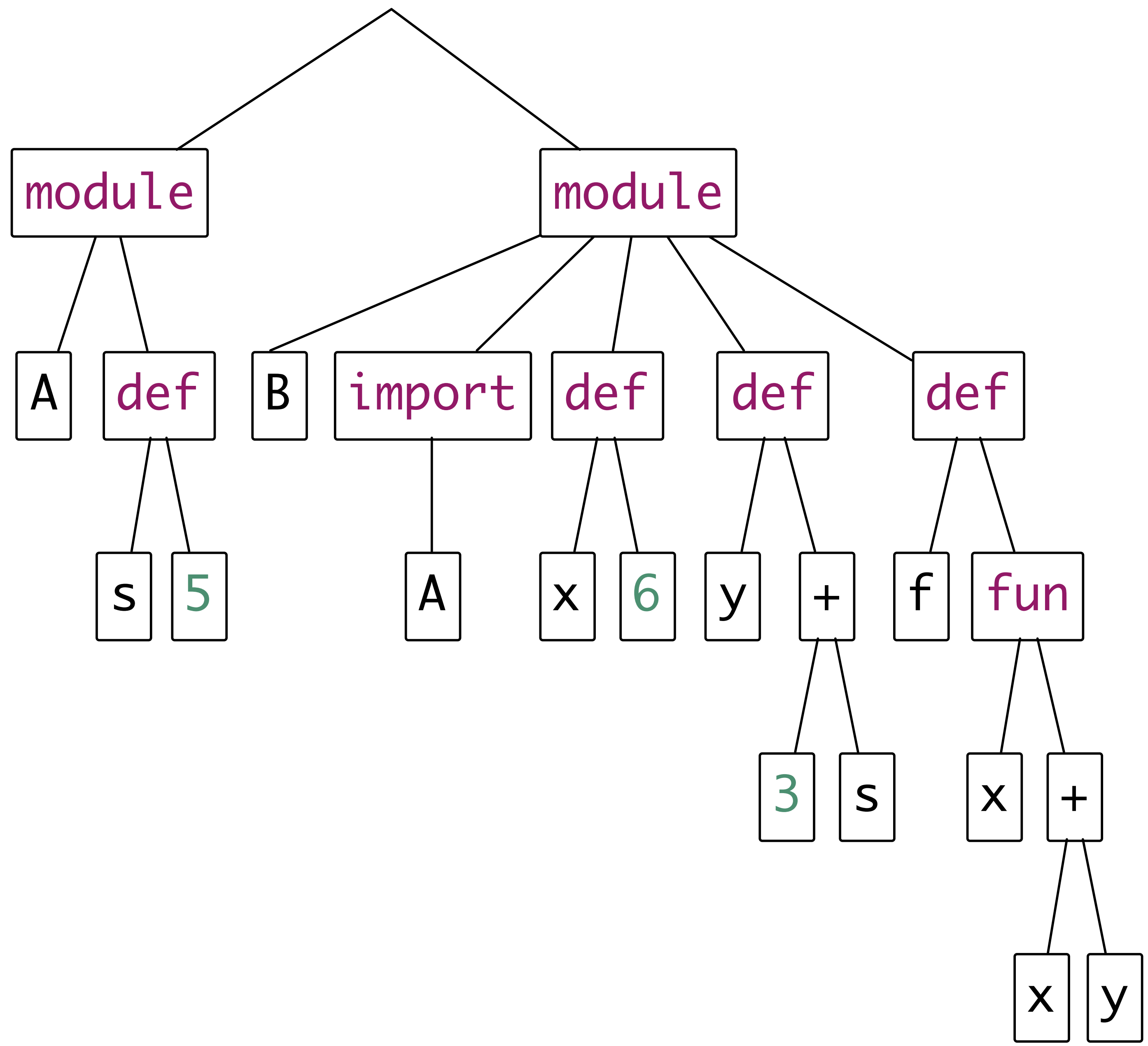


Name Resolution?

```
module A {
    def s = 5
}

module B {
    import A
    def x = 6
    def y = 3 + s
    def f =
      fun x { x + y }
}
```

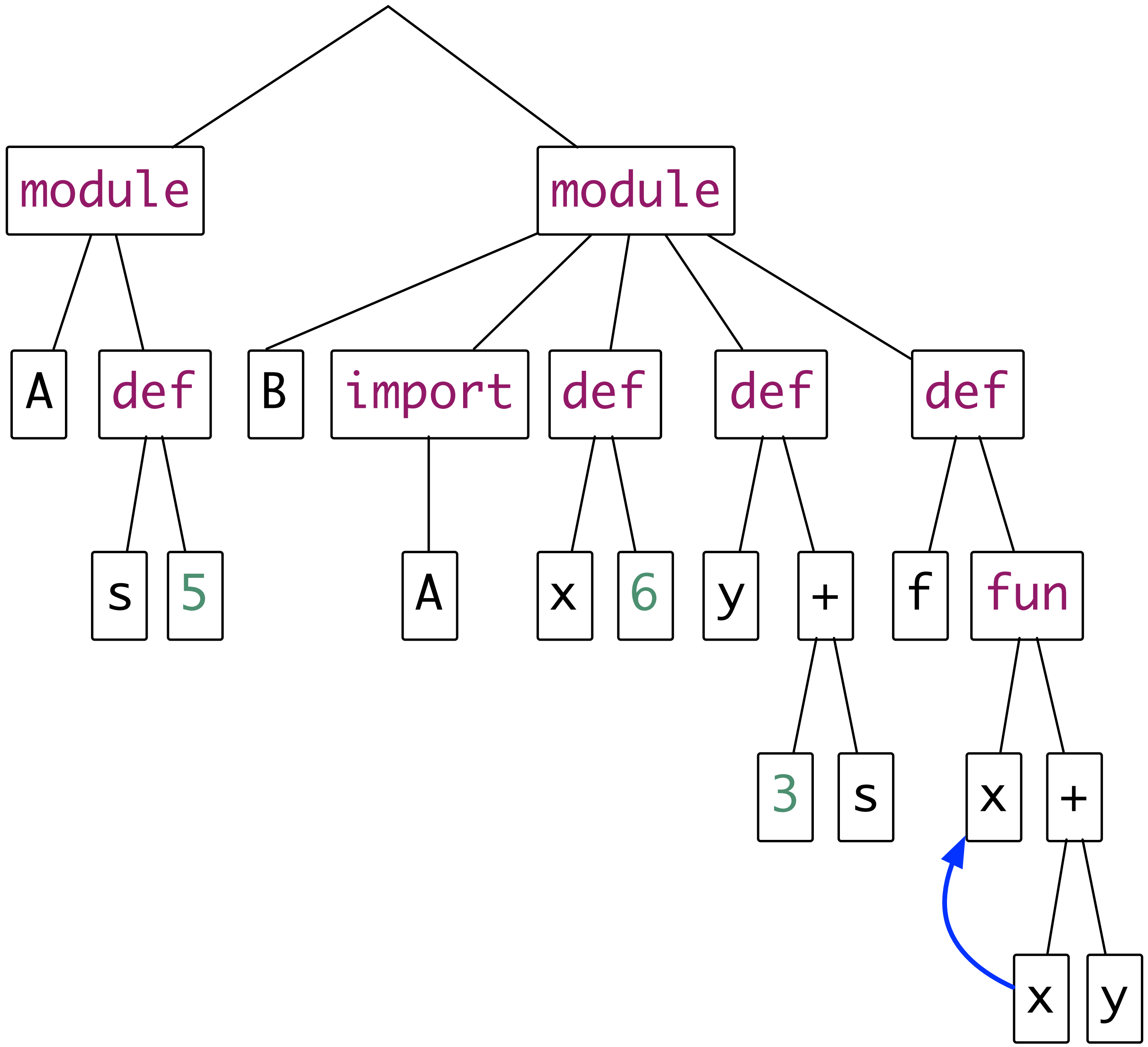
Name Resolution?

```
module A {
    def s = 5
}

module B {
    import A
    def x = 6
    def y = 3 + s
    def f =
      fun x { x + y }
}
```



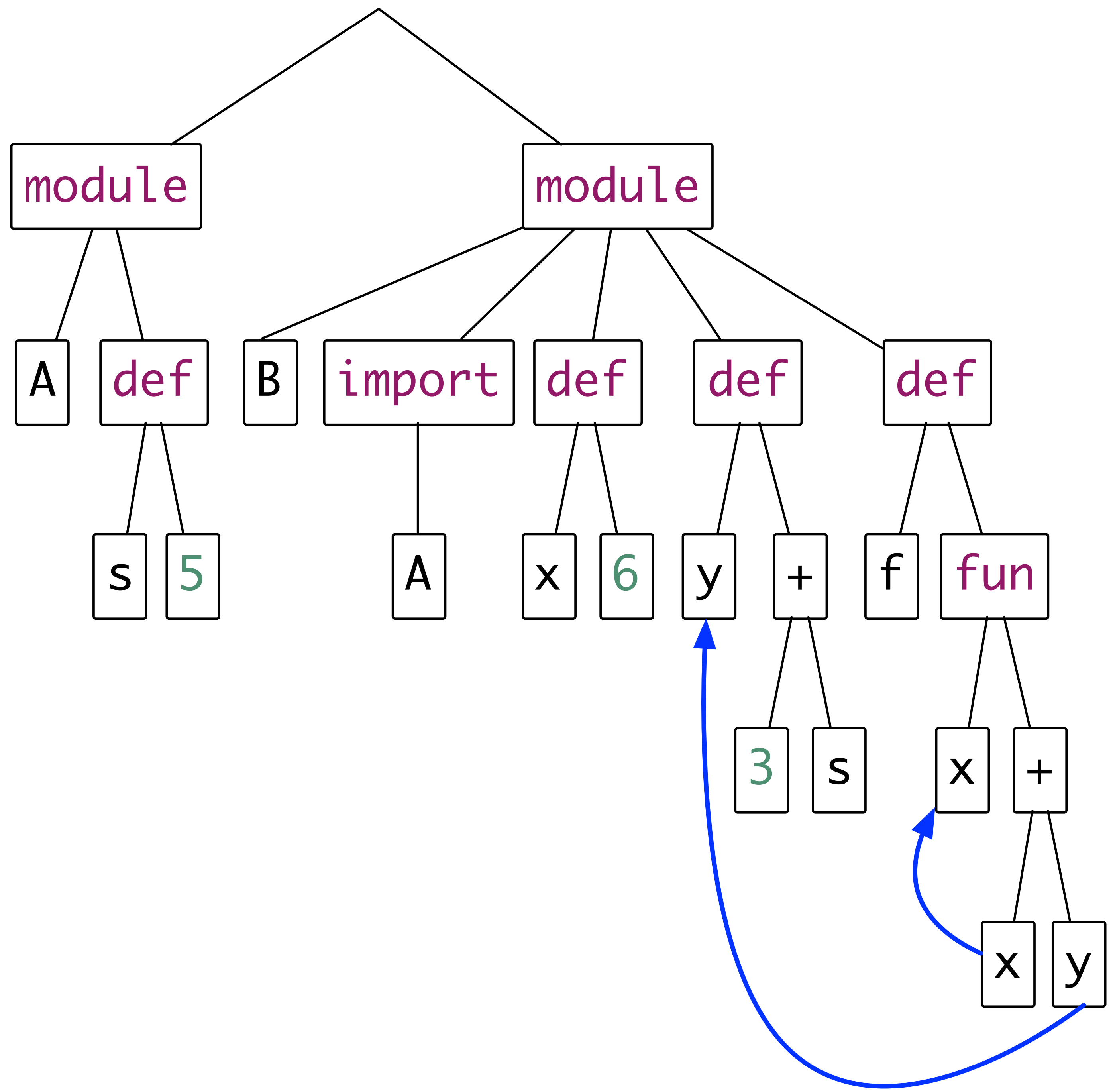
Name Resolution?

```
module A {
    def s = 5
}

module B {
    import A
    def x = 6
    def y = 3 + s
    def f =
      fun x { x + y }
}
```



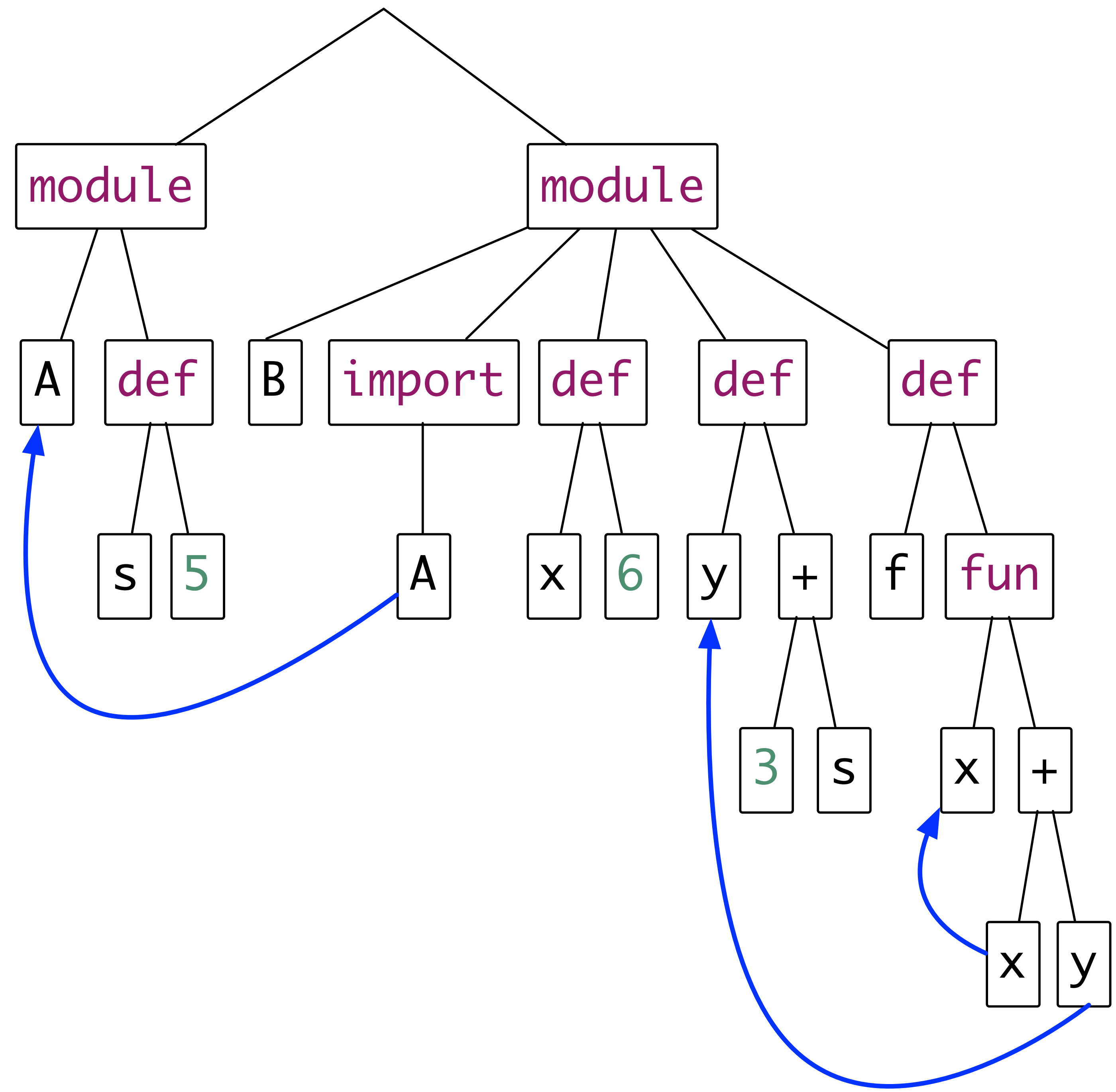
Name Resolution?

```
module A {
    def s = 5
}

module B {
    import A
    def x = 6
    def y = 3 + s
    def f =
      fun x { x + y }
}
```



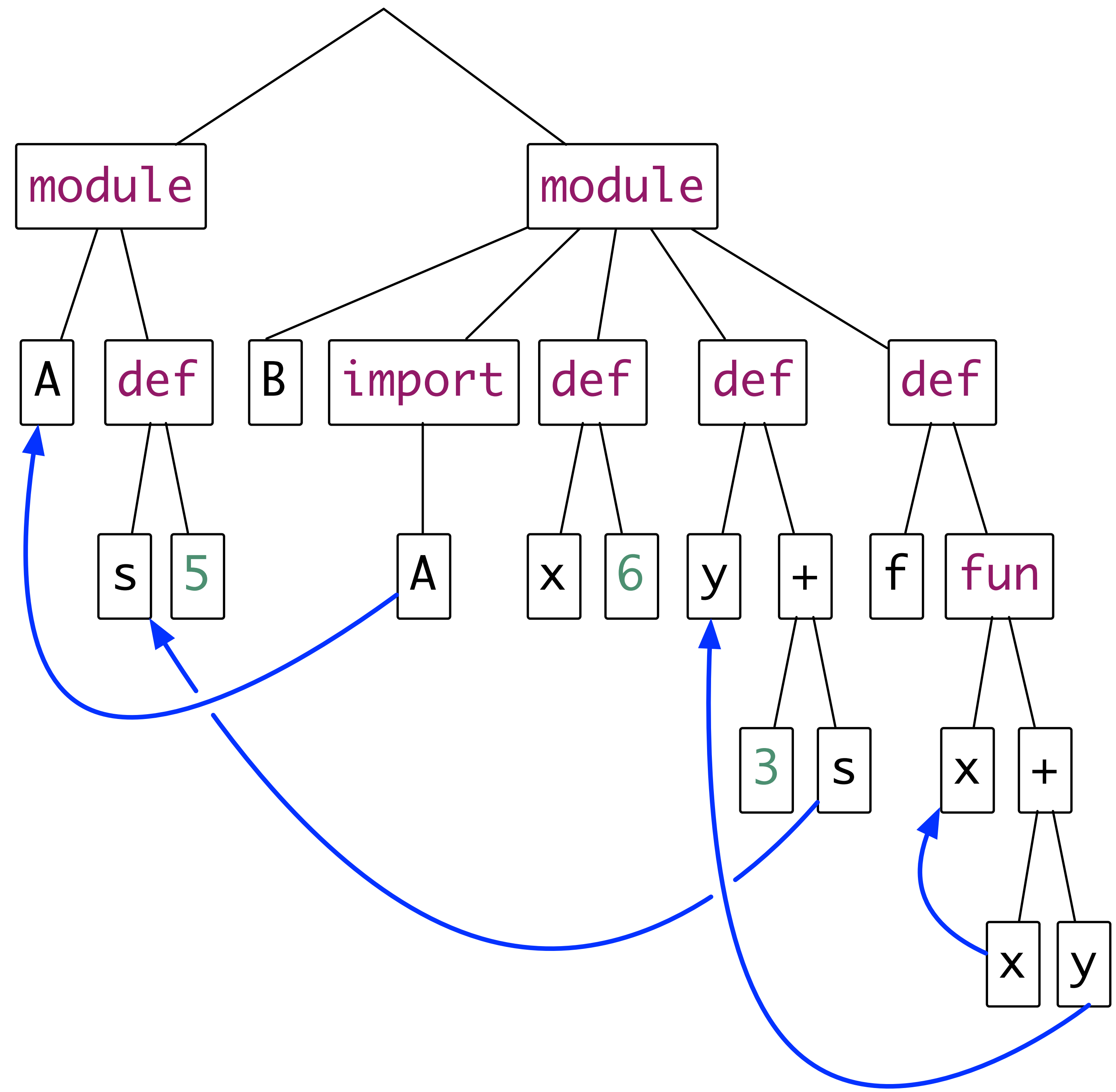
Name Resolution?

```
module A {
    def s = 5
}

module B {
    import A
    def x = 6
    def y = 3 + s
    def f =
      fun x { x + y }
}
```



Name Resolution?

```
module A {
    def s = 5
}

module B {
    import A
    def x = 6
    def y = 3 + s
    def f =
      fun x { x + y }
}
```



Name Resolution Concerns

Multiple uses for each programming language …

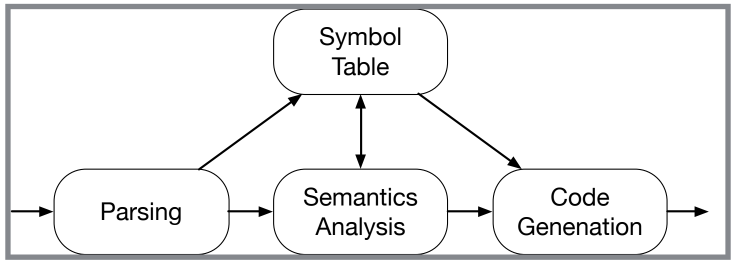
Name Resolution Concerns

Multiple uses for each programming language …

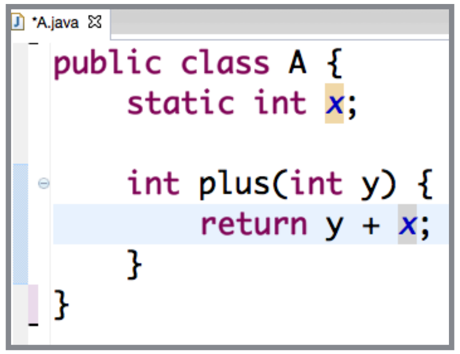
$$\frac{x : \tau_1, \Gamma \vdash e : \tau_2}{\Gamma \vdash \lambda x.e : \tau_1 \rightarrow \tau_2}$$

$$\frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau}$$

Type systems



Compiler



IDE

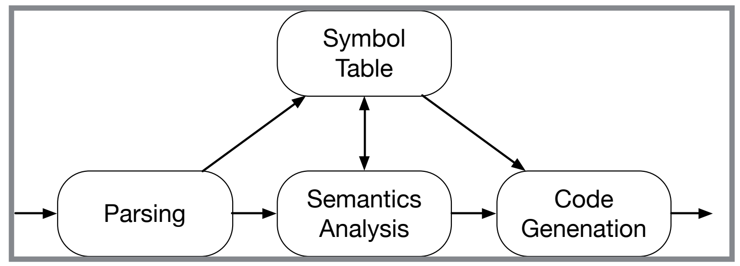
Name Resolution Concerns

Multiple uses for each programming language …

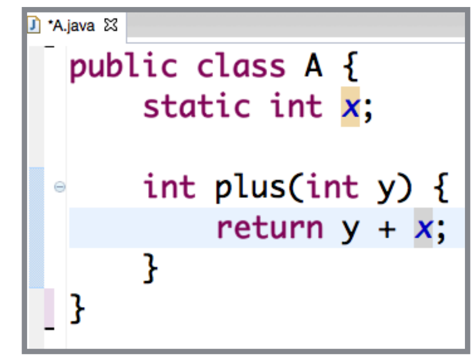
$$\frac{x : \tau_1, \Gamma \vdash e : \tau_2}{\Gamma \vdash \lambda x.e : \tau_1 \rightarrow \tau_2}$$

$$\frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau}$$

Type systems



Compiler

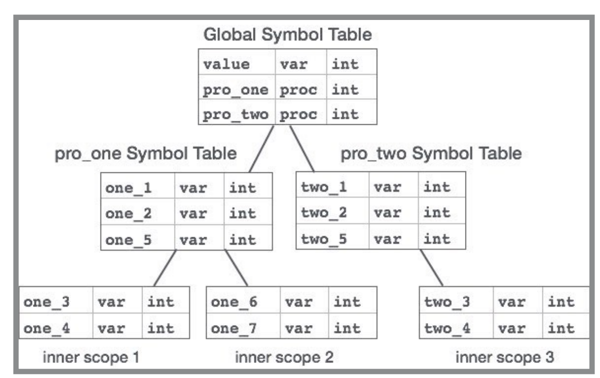


IDE

… resulting in different encodings of name resolution

`x:int, Γ`

`[3/x].σ`



$\texttt{lookup}(x_i)$

What about Syntax?

What about Syntax?

A standard formalism

Context-Free Grammars

What about Syntax?

A unique definition

```
program  =  decl*
   decl  =  module id { decl* }
         |  import qid
         |  def id = exp
    exp  =  qid
         |  fun id { exp }
         |  fix id { exp }
         |  let bind* in exp
         |  letrec bind* in exp
         |  letpar bind* in exp
         |  exp exp
         |  exp ⊕ exp
         |  int
    qid  =  id
         |  id . qid
   bind  =  id = exp
```

A standard formalism

Context-Free Grammars

What about Syntax?

A unique definition

A standard formalism

Provides

```
program  =  decl*
   decl  =  module id { decl* }
         |  import qid
         |  def id = exp
    exp  =  qid
         |  fun id { exp }
         |  fix id { exp }
         |  let bind* in exp
         |  letrec bind* in exp
         |  letpar bind* in exp
         |  exp exp
         |  exp ⊕ exp
         |  int
    qid  =  id
         |  id . qid
   bind  =  id = exp
```

Context-Free Grammars

- Parser
- AST
- Pretty-Printer
- Highlighting

A Theory of Name Resolution

A Theory of Name Resolution

*For **statically lexically scoped** languages*

A Theory of Name Resolution

*For **statically lexically scoped** languages*

A standard formalism

Scope

Graphs

A Theory of Name Resolution

*For **statically lexically scoped** languages*

A unique definition

A standard formalism

Program

↓

Scope Graph

Scope

Graphs

A Theory of Name Resolution

*For **statically lexically scoped** languages*

A unique definition

Program

↓

Scope Graph

A standard formalism

Scope Graphs

Provides

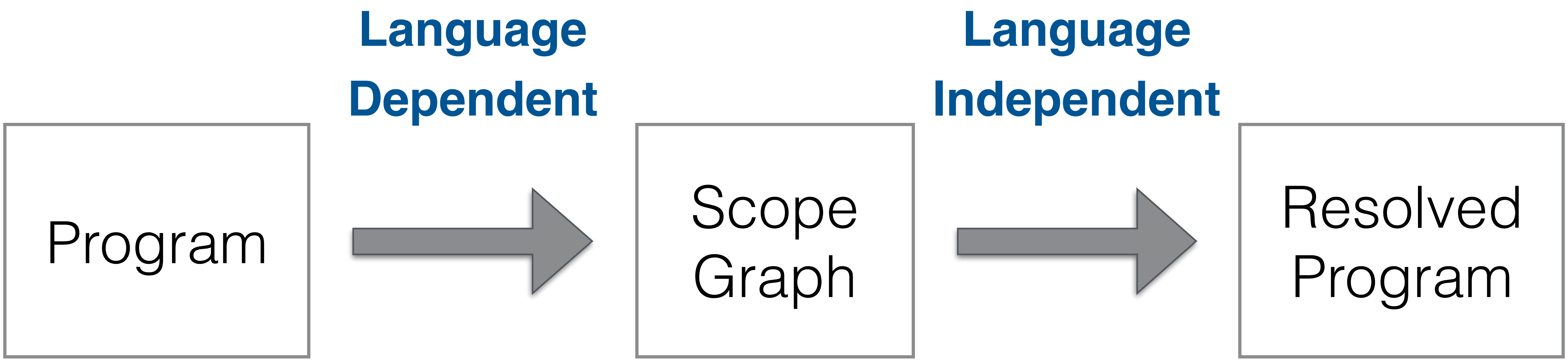
Resolution

$\boldsymbol{\alpha}$-equivalence

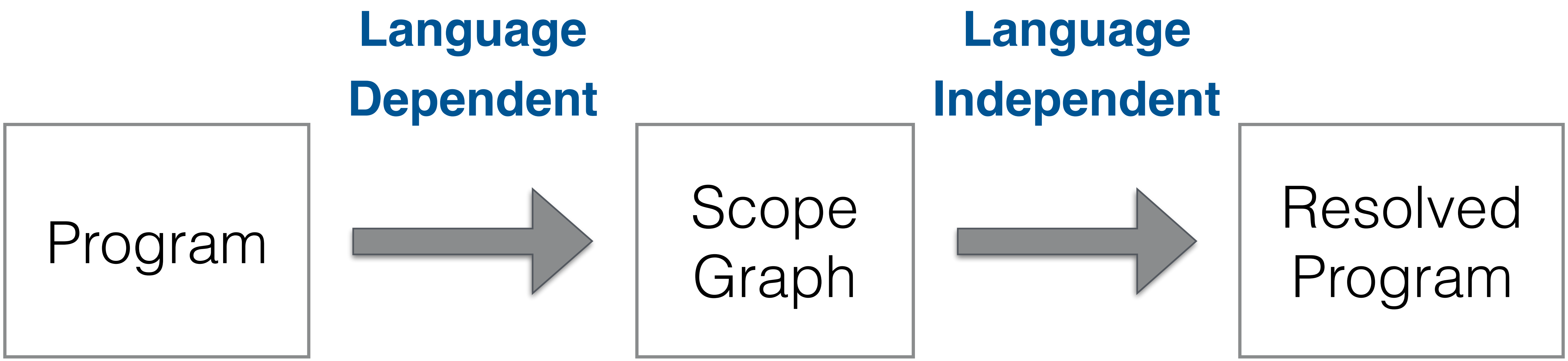
IDE Navigation

Refactoring tools

Resolution Scheme



Resolution Scheme



Resolution of a reference in a scope graph:

Building a **path**
from a **reference** node
to a **declaration** node
following path construction **rules**

Scope Graph

```
def y = x + 1
def x = 5
```

Scope Graph

```
def y₁ = x₂ + 1
def x₁ = 5
```

Scope Graph

```
def y₁ = x₂ + 1
def x₁ = 5
```

Scope Graph

S0

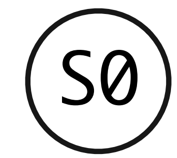
$\texttt{def } y_1 = x_2 + 1$

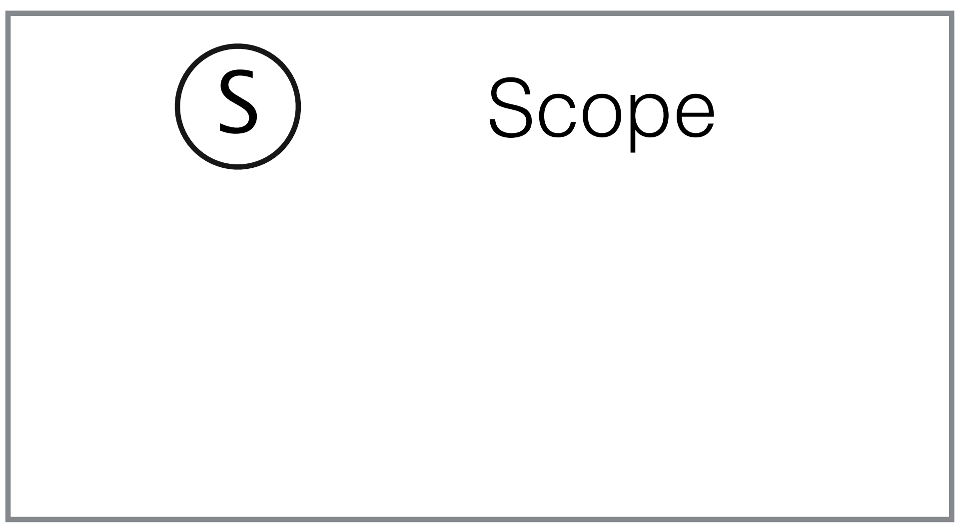
$\texttt{def } x_1 = 5$

Scope Graph

S0

```
def y1 = x2 + 1
def x1 = 5
```

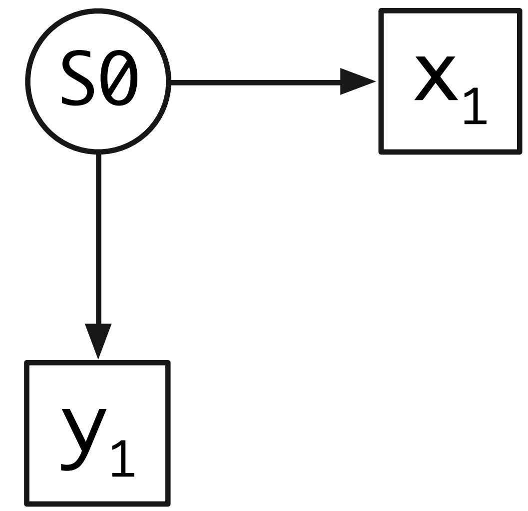


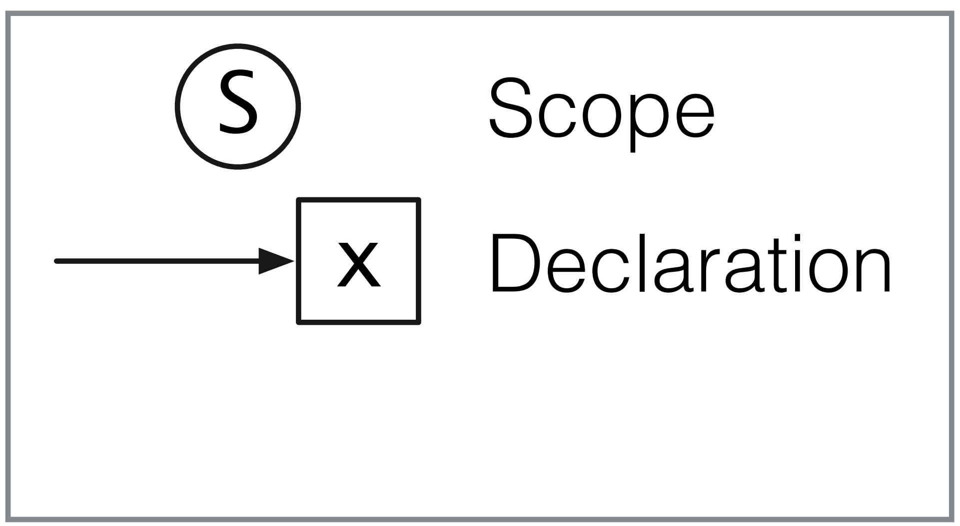


Scope Graph

S0

def y_1 = x_2 + 1
def x_1 = 5



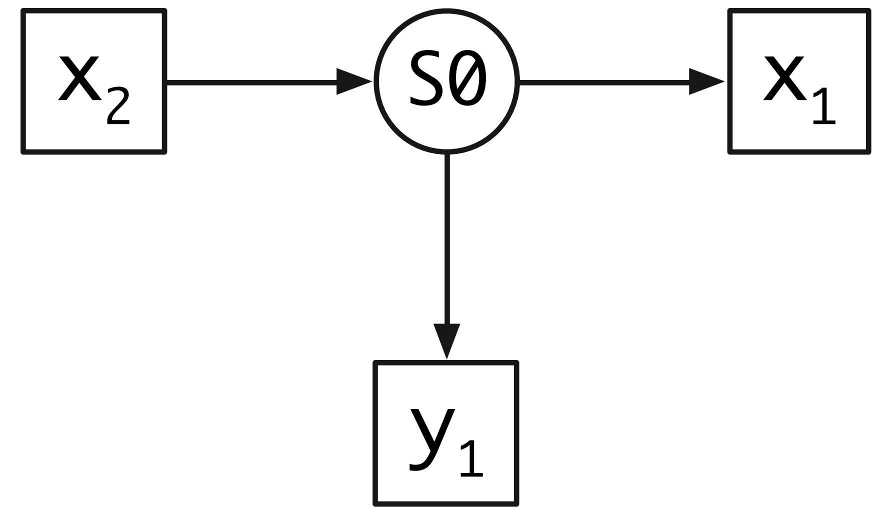


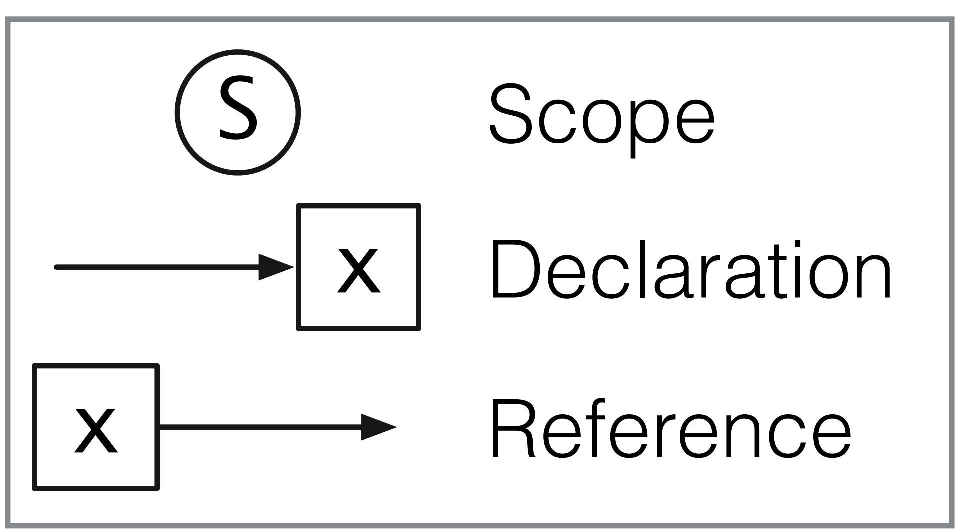
Scope Graph

S0

$\texttt{def}\ y_1 = x_2 + 1$

$\texttt{def}\ x_1 = 5$

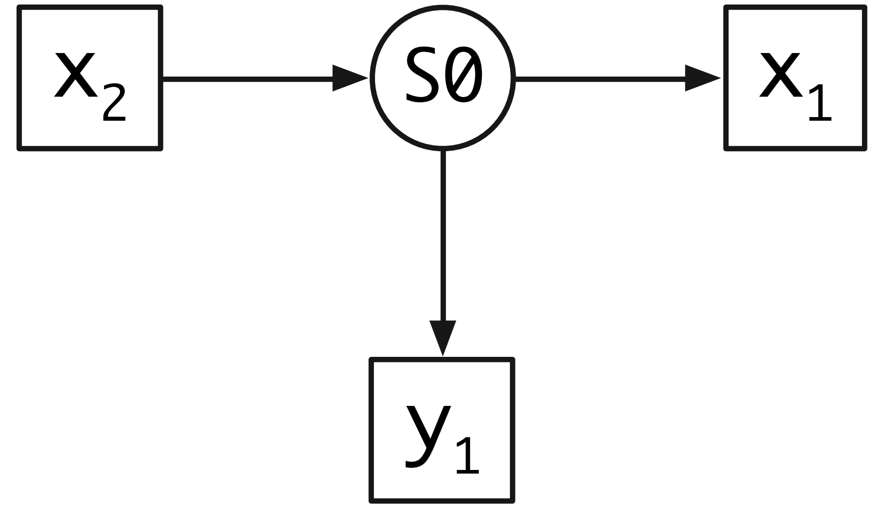


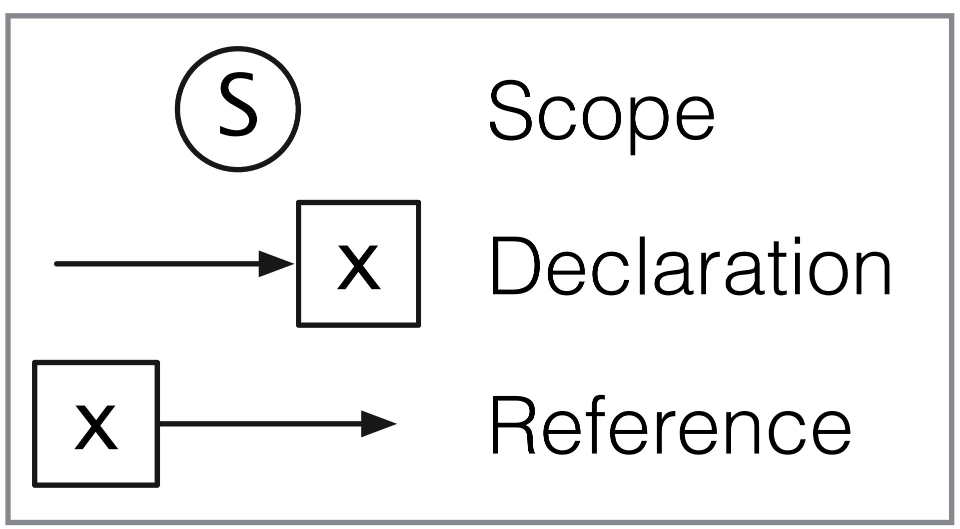


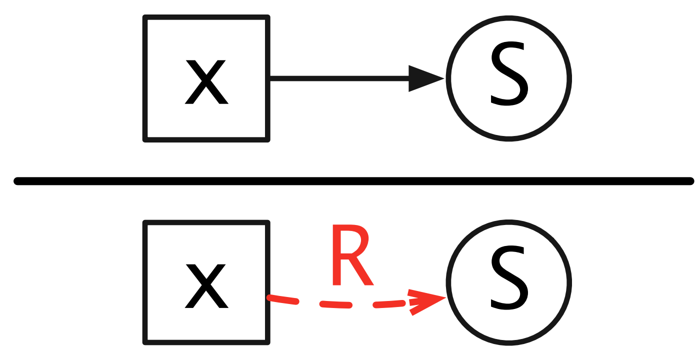
Scope Graph

S0

```
def y1 = x2 + 1
def x1 = 5
```







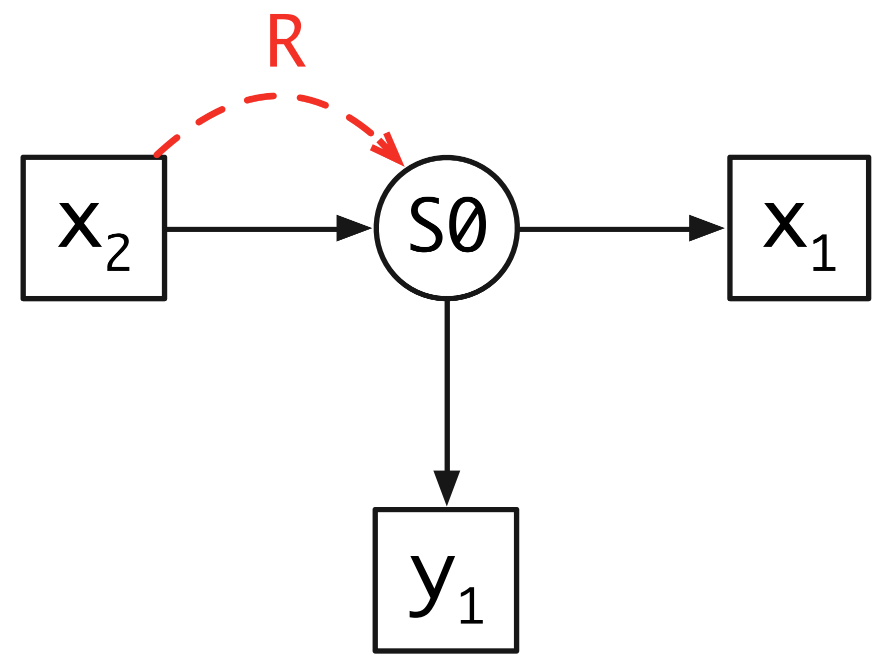
Reference Step

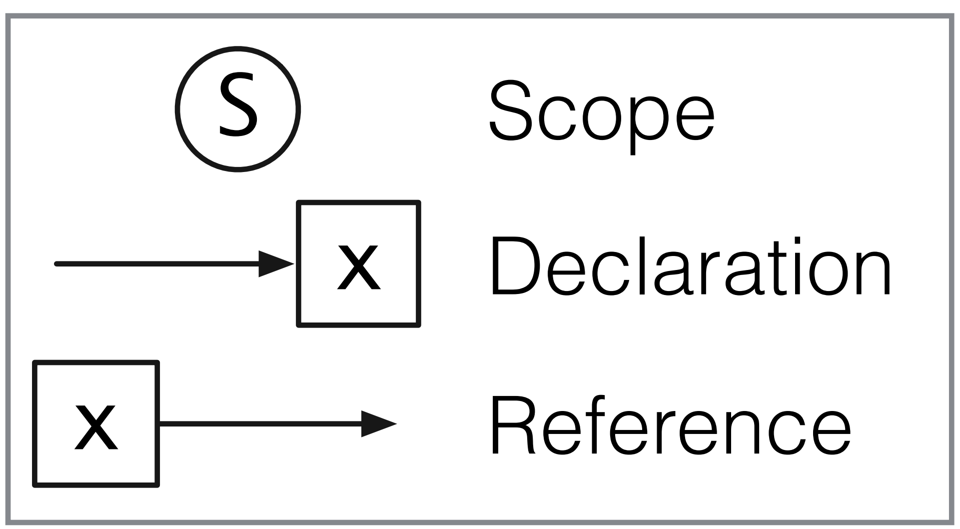
Scope Graph

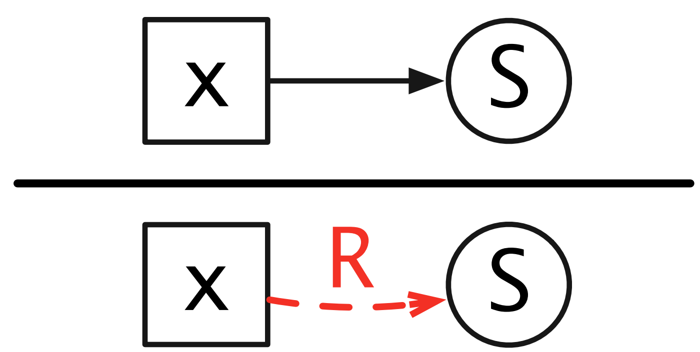
S0

def y_1 = x_2 + 1

def x_1 = 5







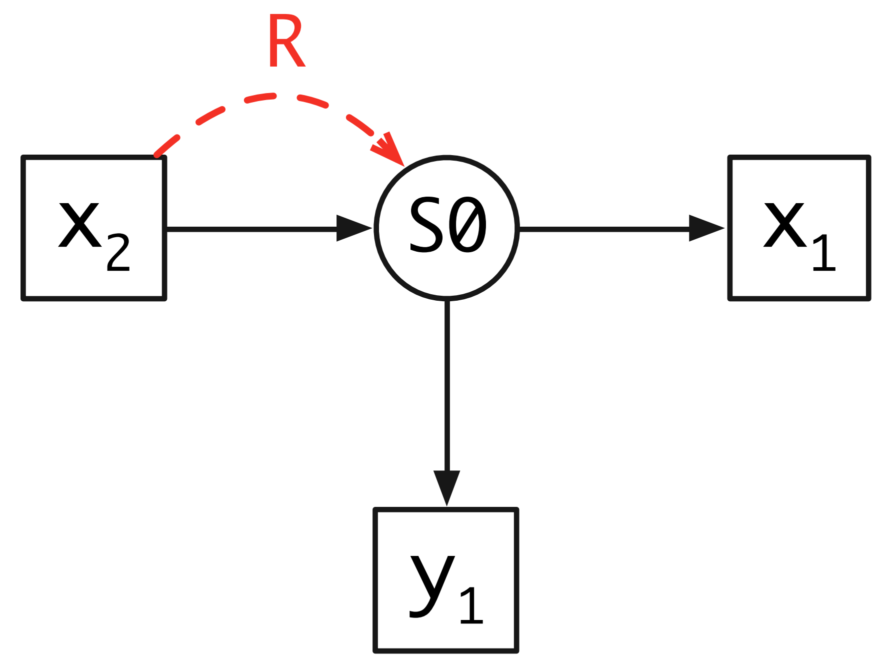
Reference Step

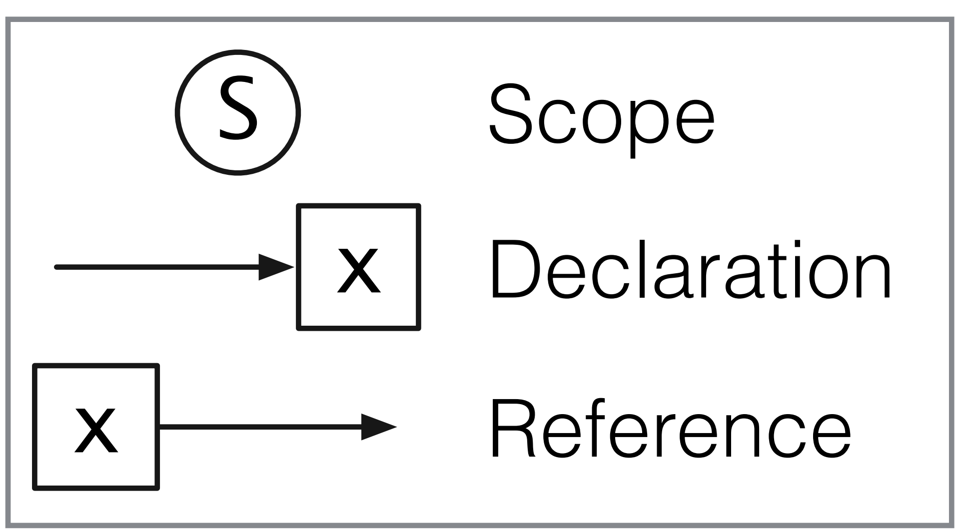
Scope Graph

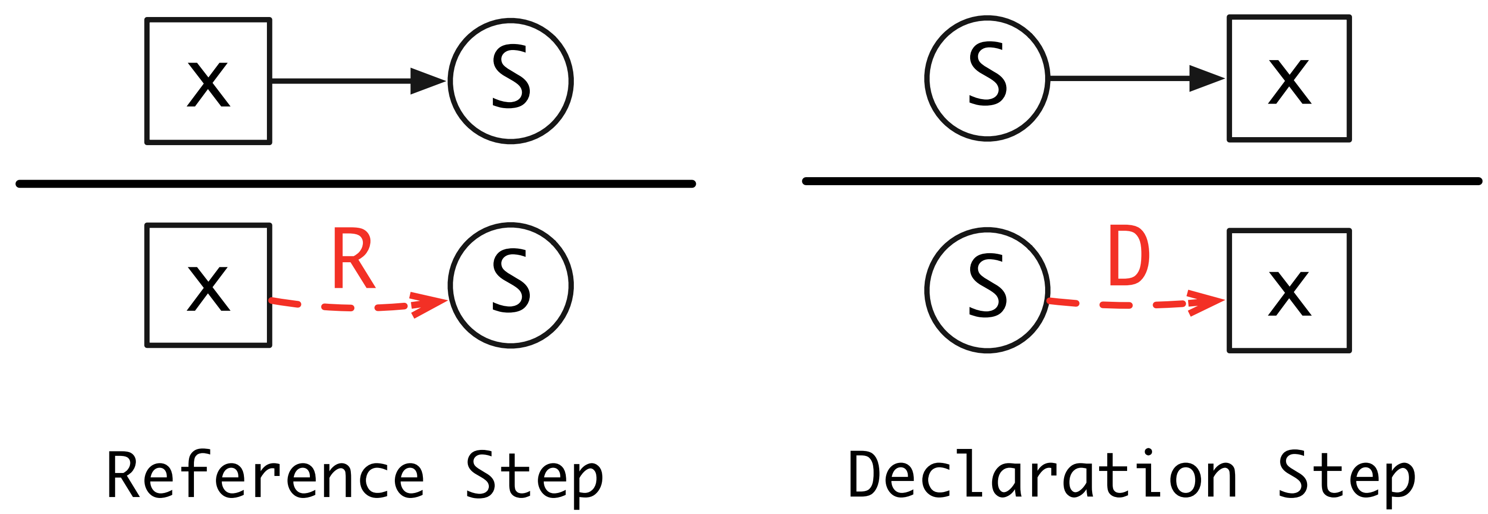
S0

$\mathtt{def}\ y_1 = x_2 + 1$

$\mathtt{def}\ x_1 = 5$





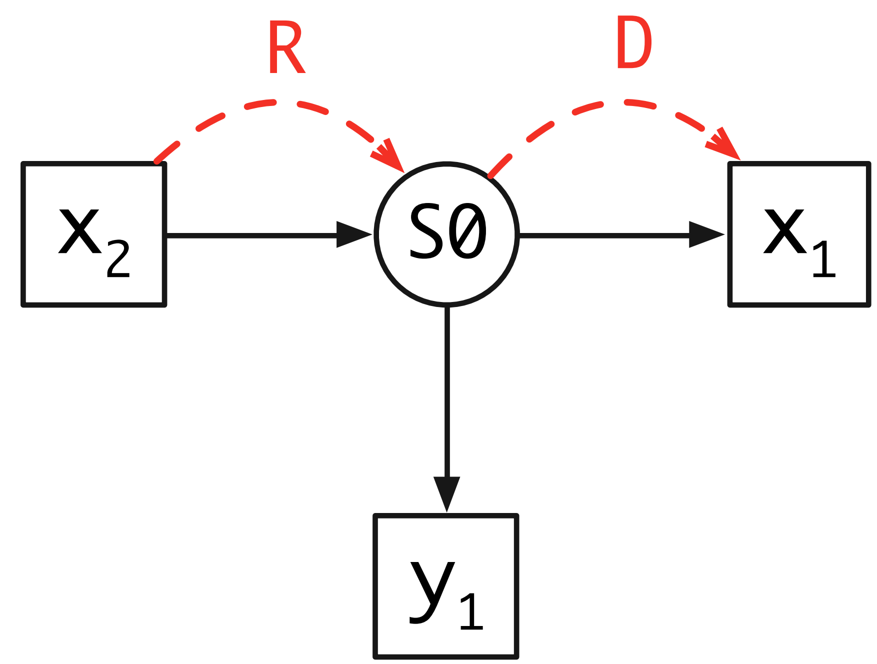


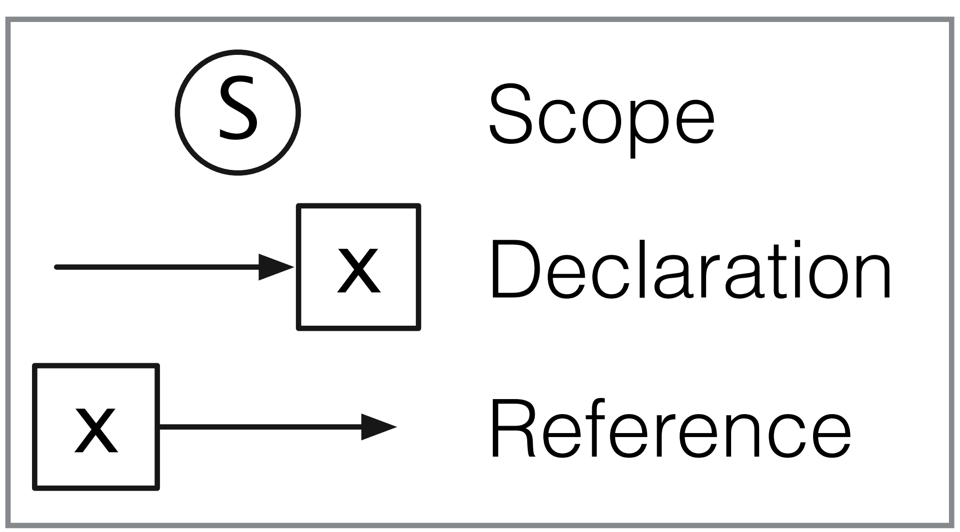
Scope Graph

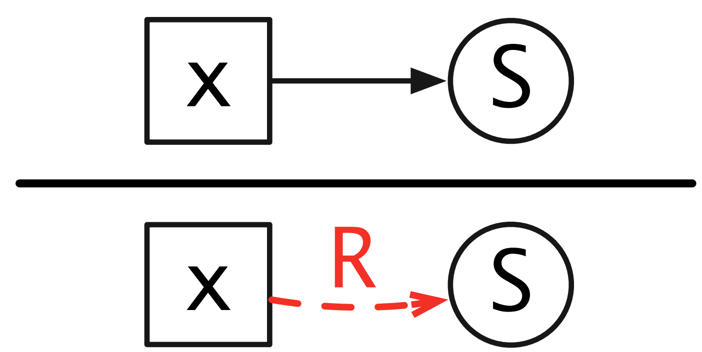
S0

def y_1 = x_2 + 1

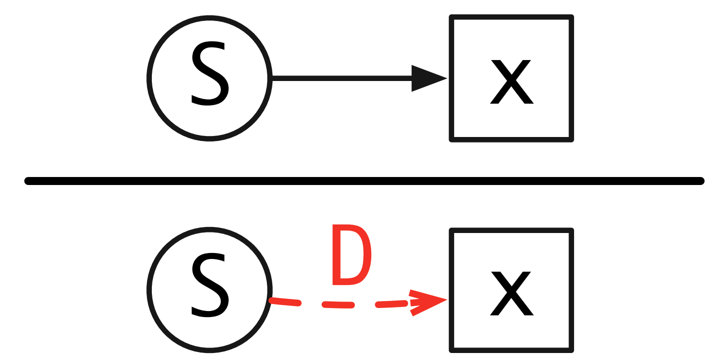
def x_1 = 5







Reference Step

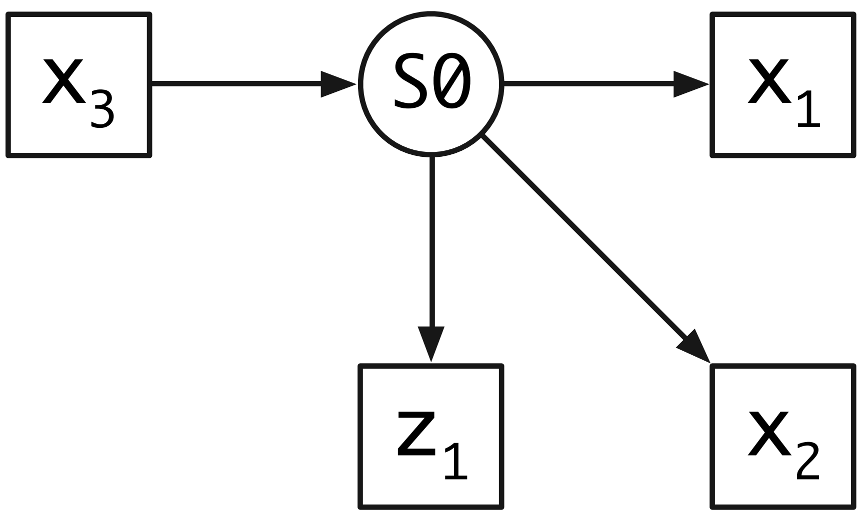


Declaration Step

Ambiguous Resolutions

```
def x₁ = 5

def x₂  = 3

def z₁ = x₃ + 1
```

Ambiguous Resolutions

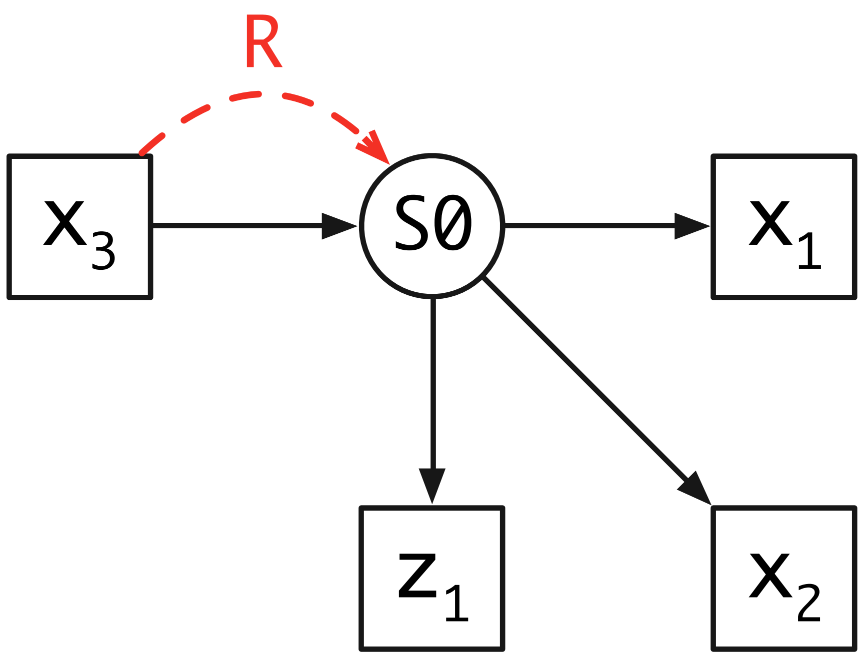


def $\boxed{x_1}$ = 5

def $\boxed{x_2}$ = 3

def $z_1 = x_3 + 1$

Ambiguous Resolutions

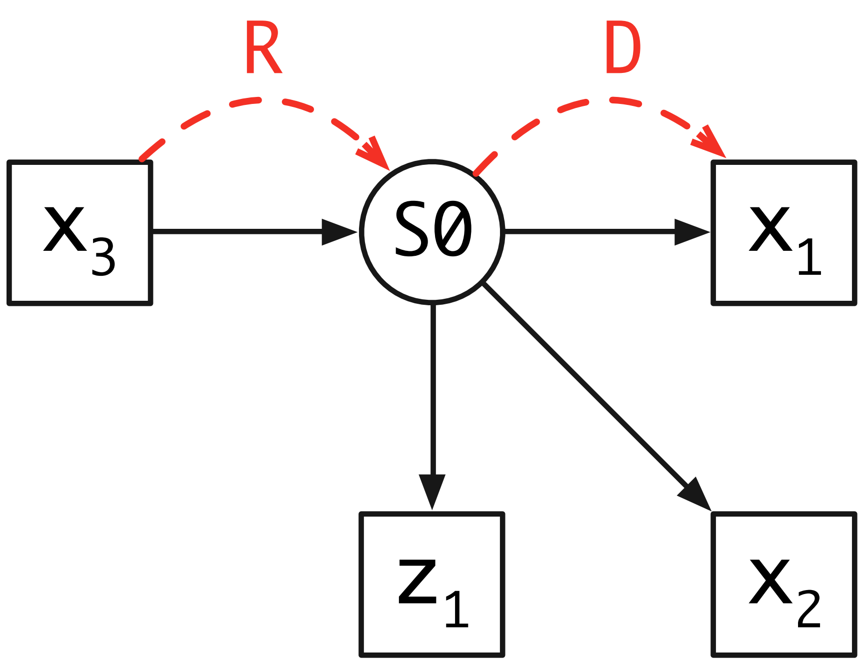


$$\texttt{def}\ \boxed{x_1} = 5$$

$$\texttt{def}\ \boxed{x_2} = 3$$

$$\texttt{def}\ z_1 = x_3 + 1$$

Ambiguous Resolutions

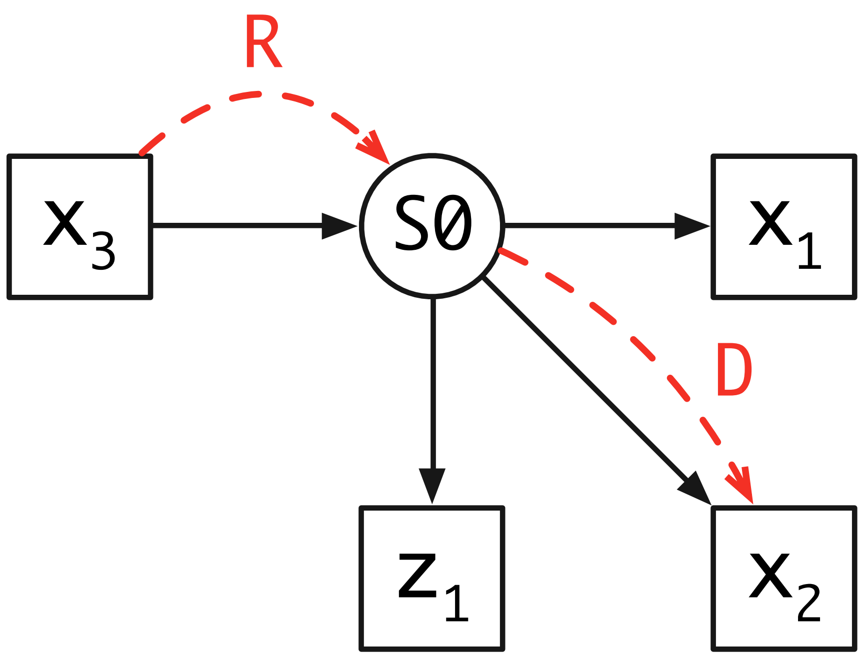


$$\texttt{def}\ \boxed{x_1} = 5$$

$$\texttt{def}\ \boxed{x_2} = 3$$

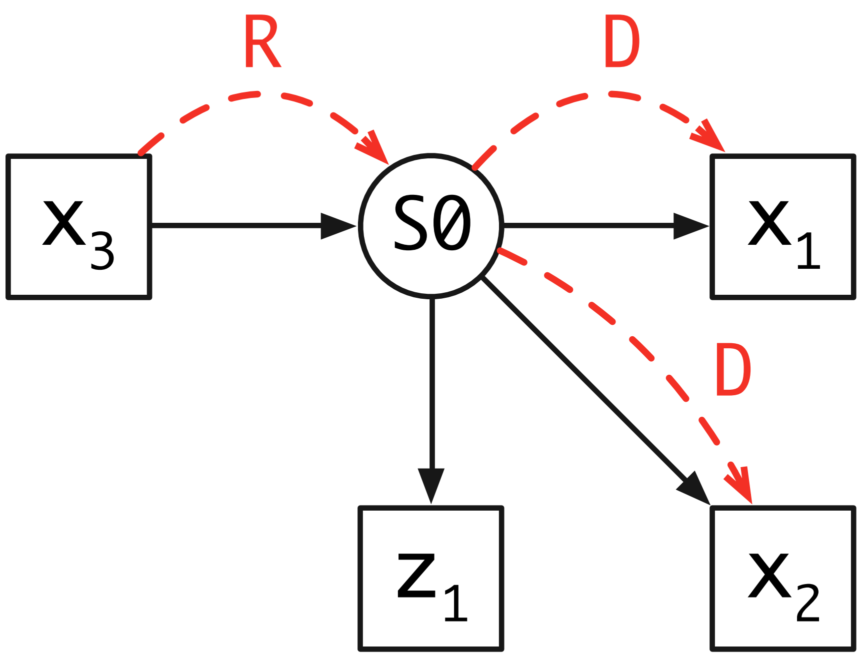
$$\texttt{def}\ z_1 = x_3 + 1$$

Ambiguous Resolutions




```
def x₁ = 5

def x₂ = 3

def z₁ = x₃ + 1
```

Ambiguous Resolutions

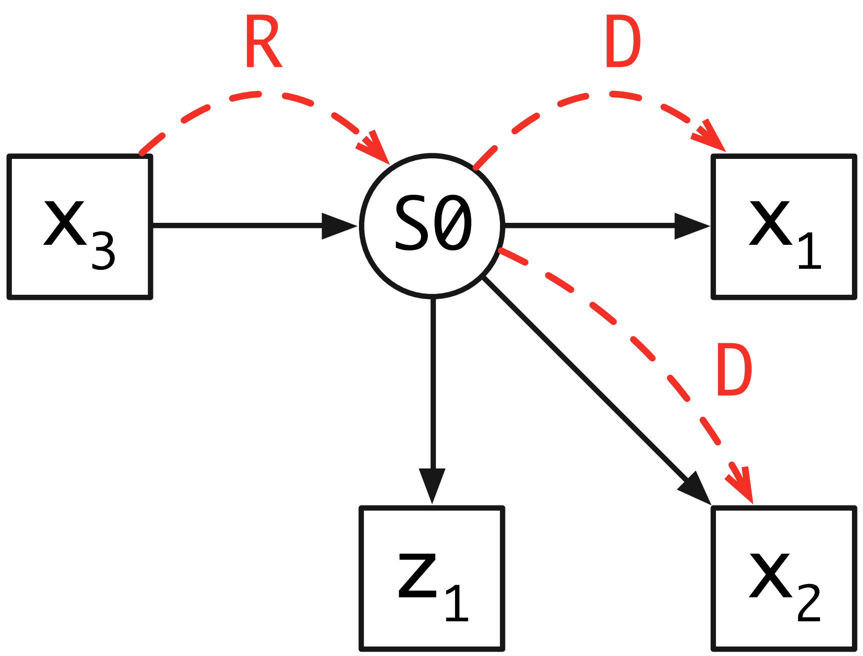


$\texttt{def}\ \boxed{x_1} = 5$

$\texttt{def}\ \boxed{x_2} = 3$

$\texttt{def}\ z_1 = x_3 + 1$

Ambiguous Resolutions



def $\boxed{x_1}$ = 5

def $\boxed{x_2}$ = 3

def z_1 = x_3 + 1

match t with
| A $\boxed{x}$ | B $\boxed{x}$ => …

Lexical Scoping

```
def x₁ = z₂ 5

def z₁ =
  fun y₁ {
    x₂ + y₂
  }
```

Lexical Scoping

S0

```
def x₁ = z₂ 5

def z₁ =
  fun y₁ {
    x₂ + y₂
  }
```

Lexical Scoping

S0

def x_1 = z_2 5

def z_1 =

fun y_1 { S1

x_2 + y_2

}

Lexical Scoping

S0

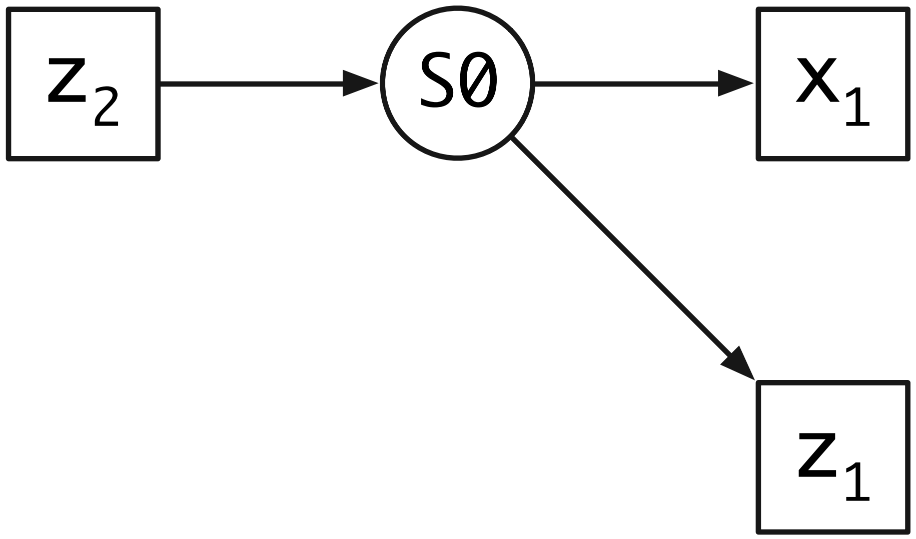
def x_1 = z_2 5

def z_1 =

fun y_1 {   S1

$x_2 + y_2$

}



Lexical Scoping

S0

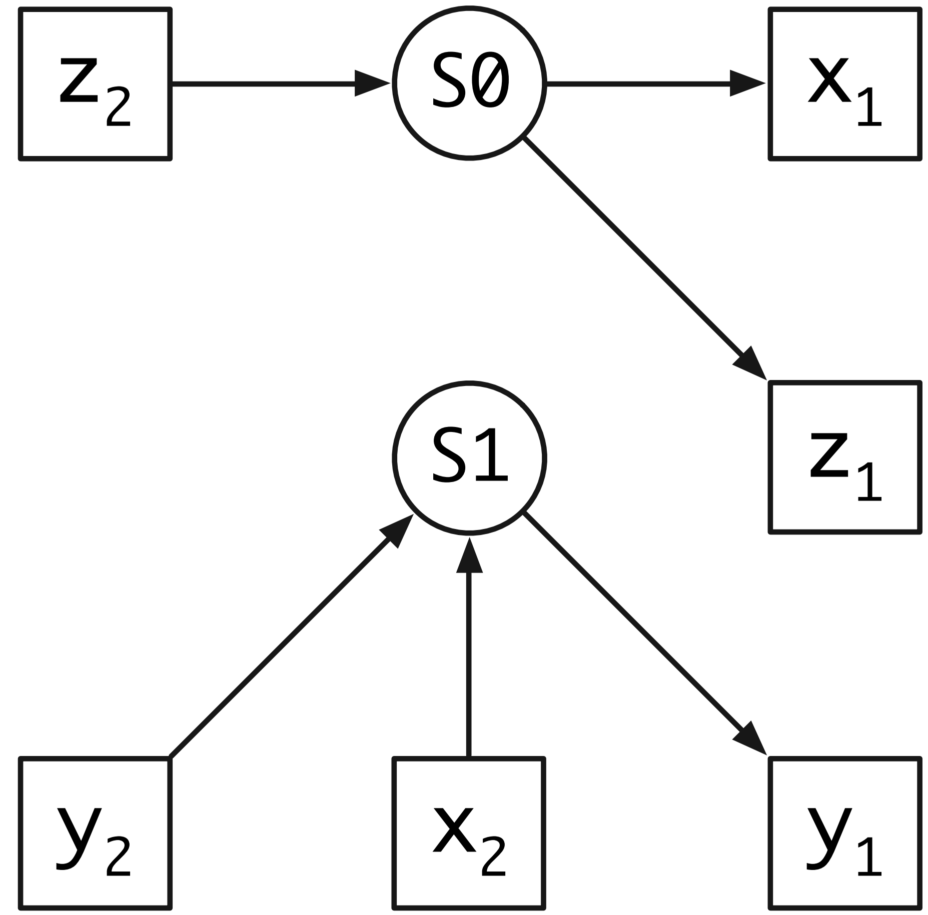
def x_1 = z_2 5

def z_1 =

fun y_1 { S1

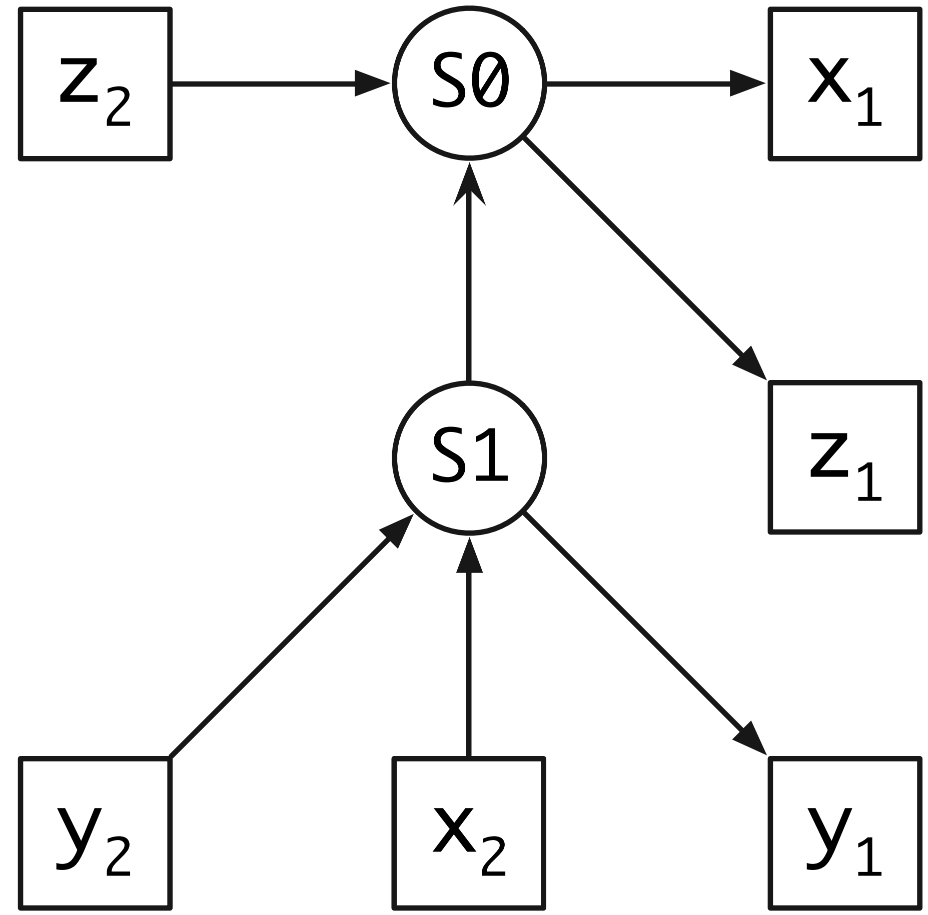
x_2 + y_2

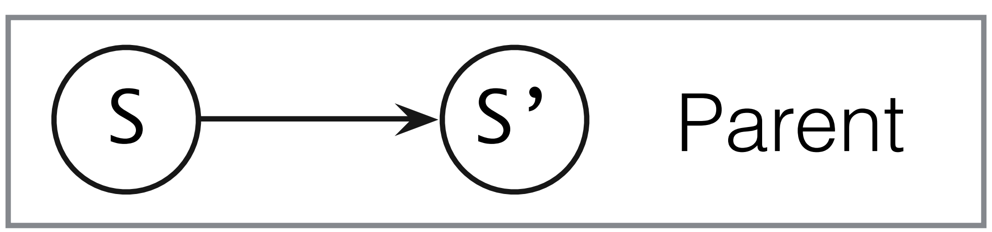
}



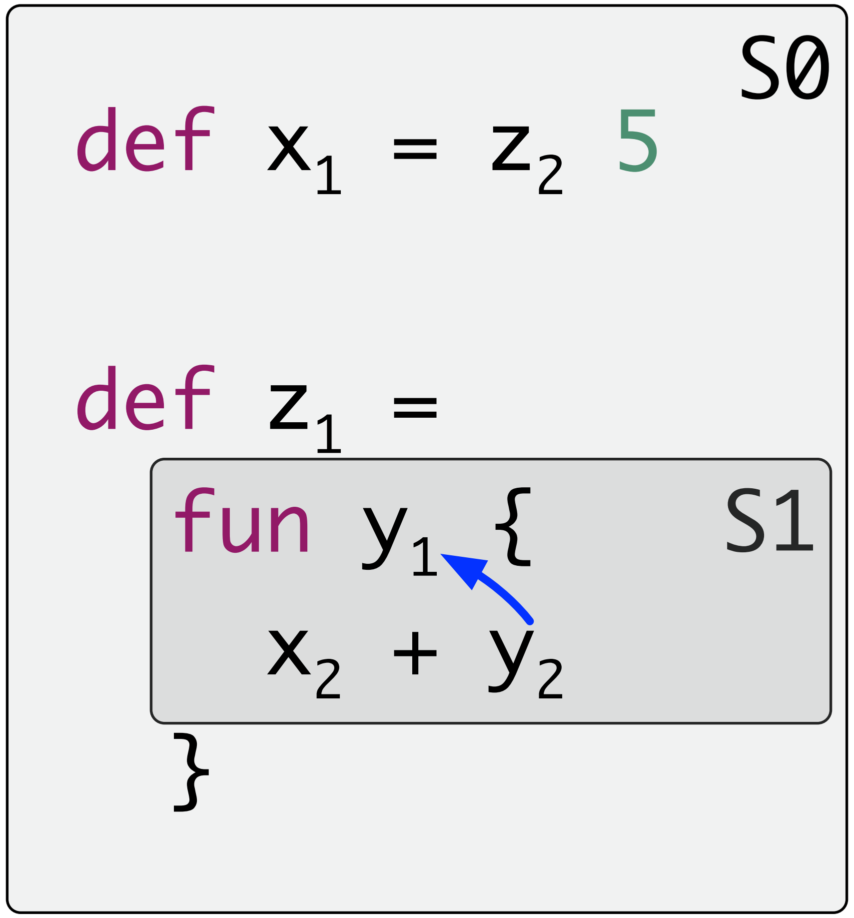
Lexical Scoping

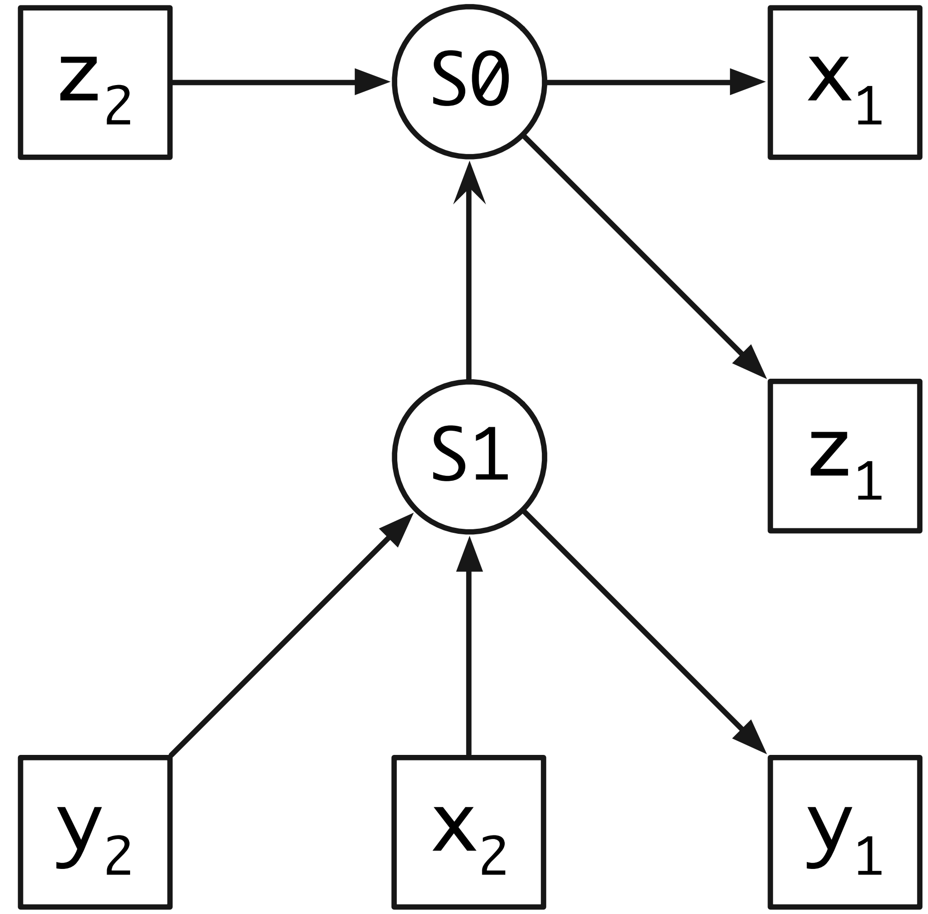
```
                       S0
def x1 = z2 5


def z1 =
   fun y1 {         S1
     x2 + y2
   }
```

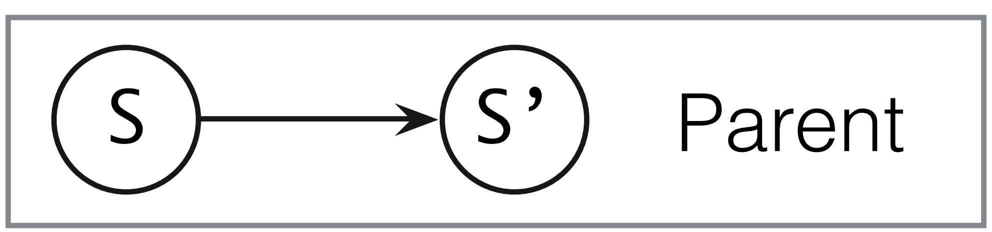




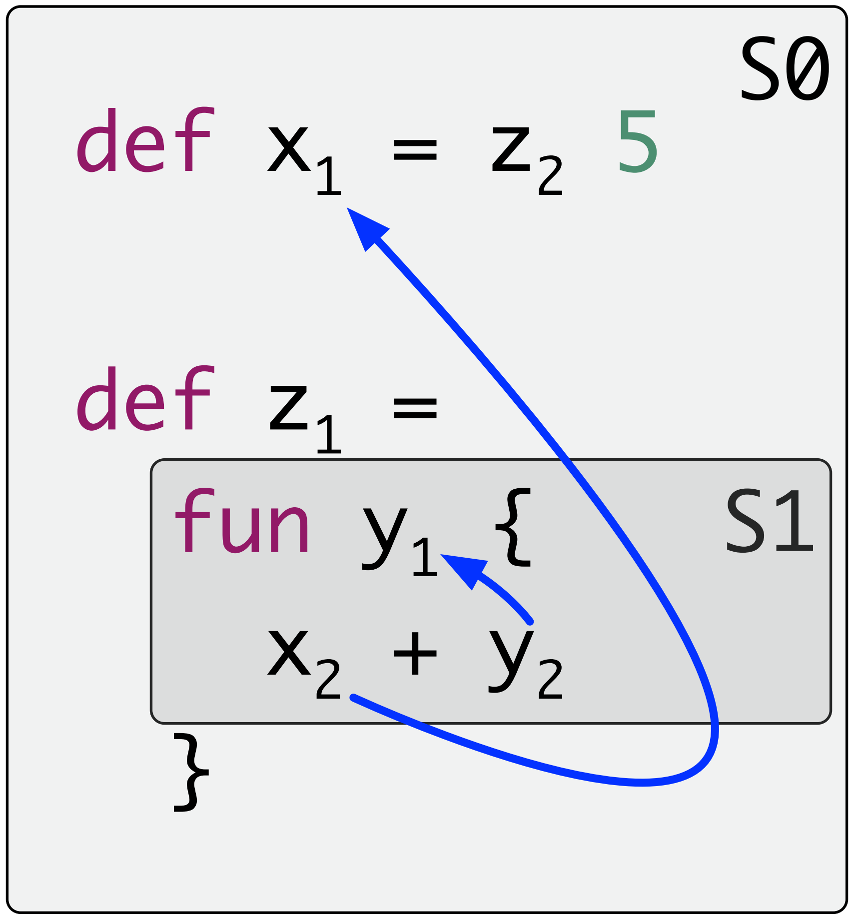
Lexical Scoping

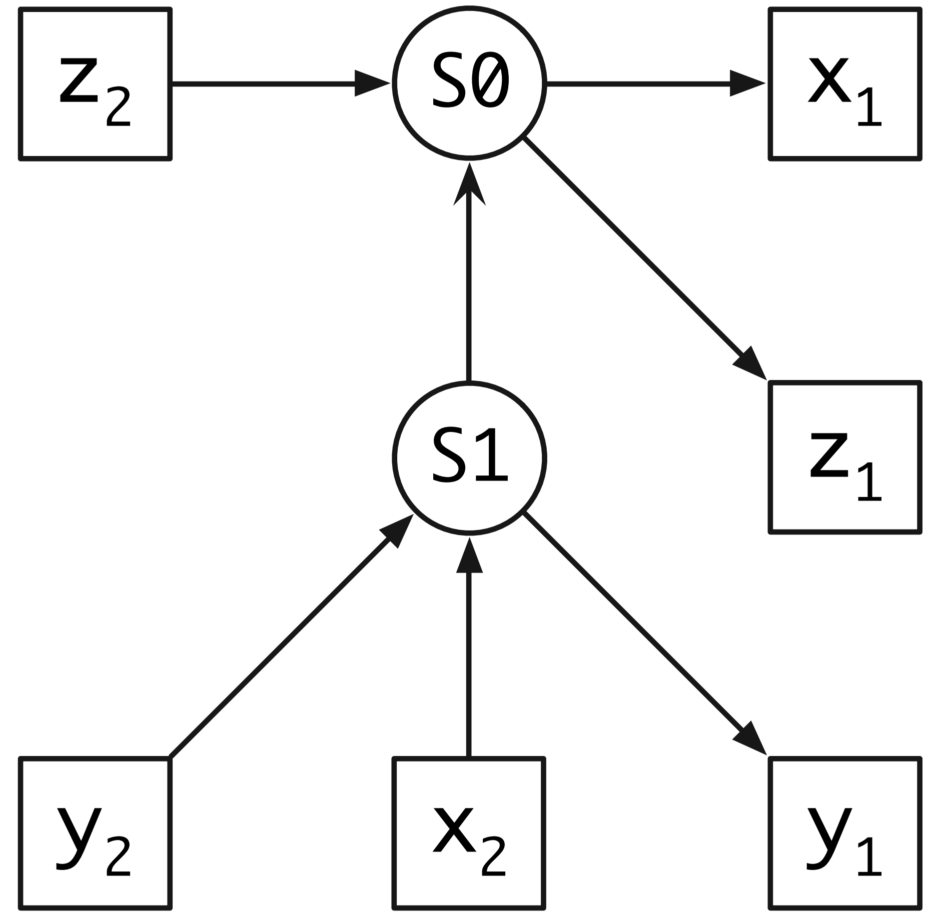


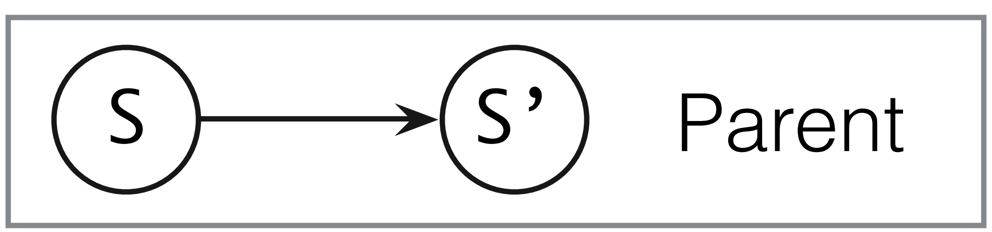




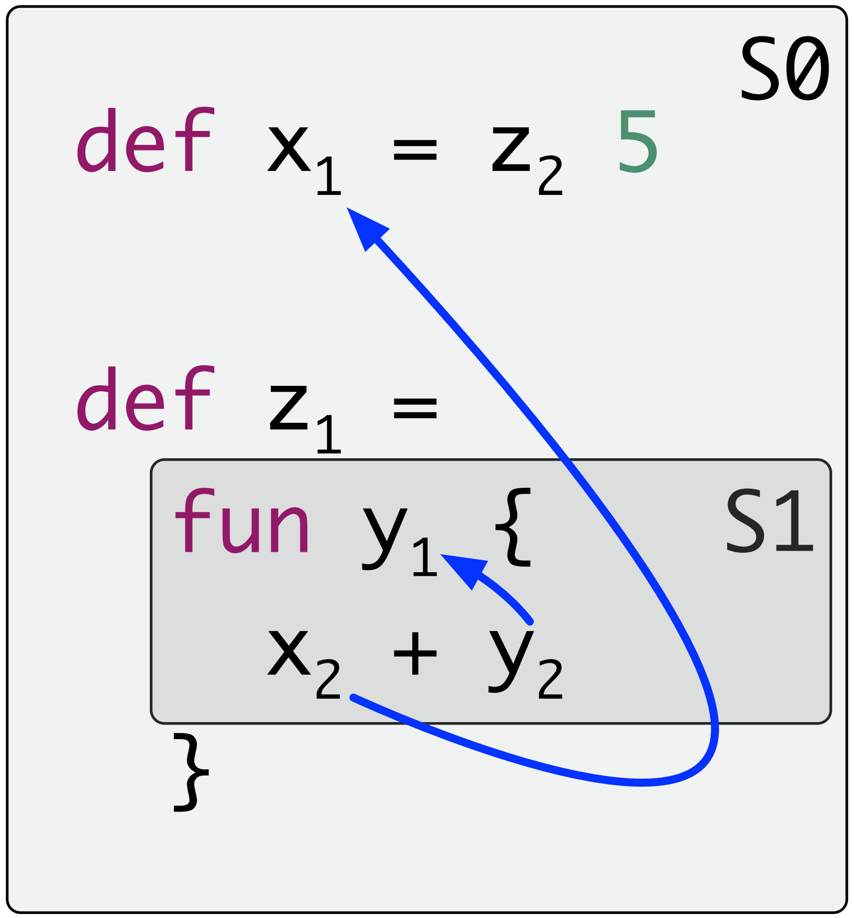
Lexical Scoping

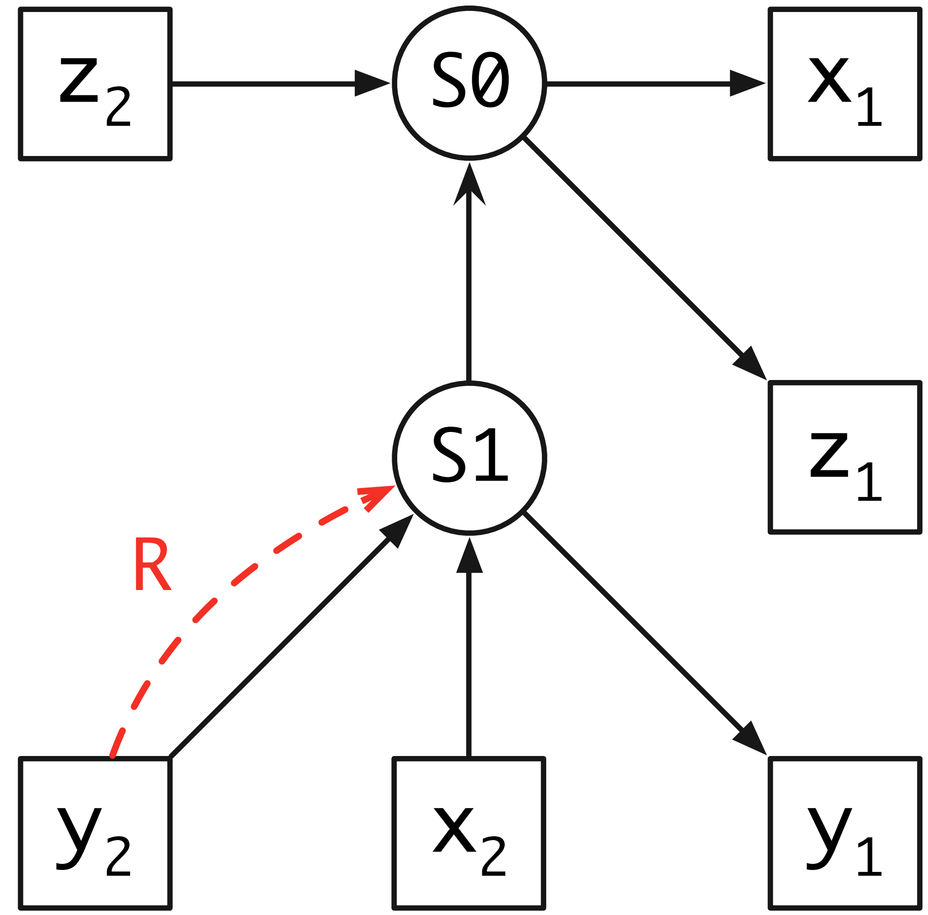


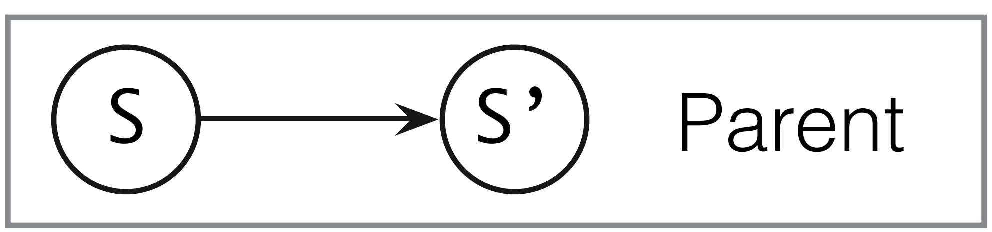




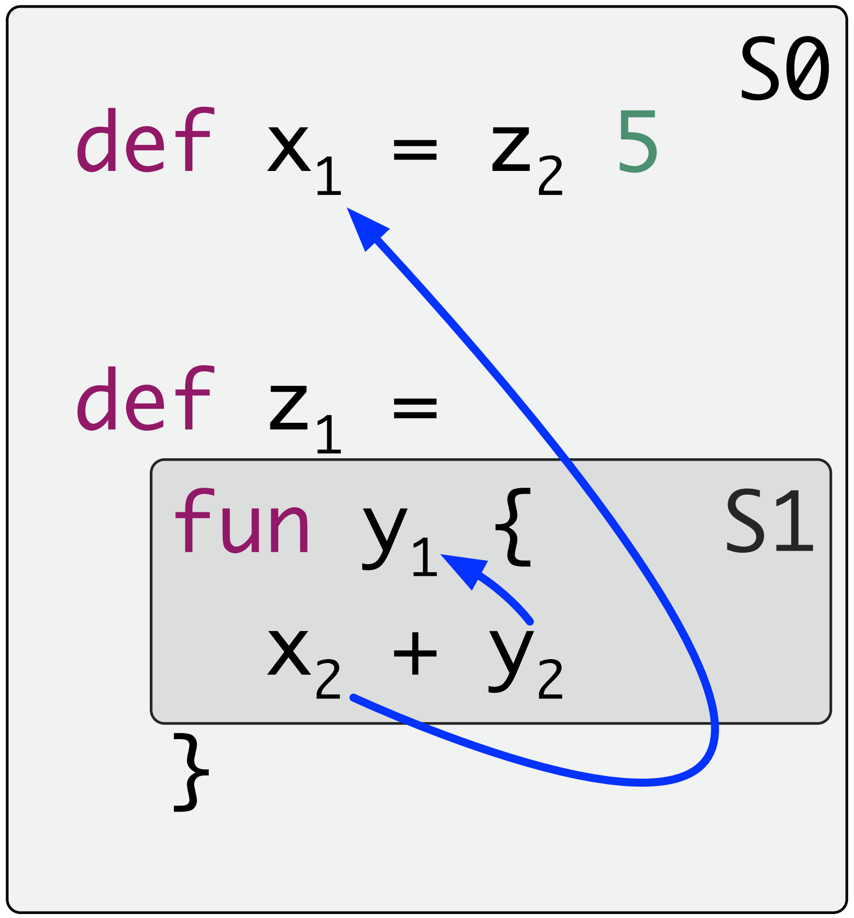
Lexical Scoping

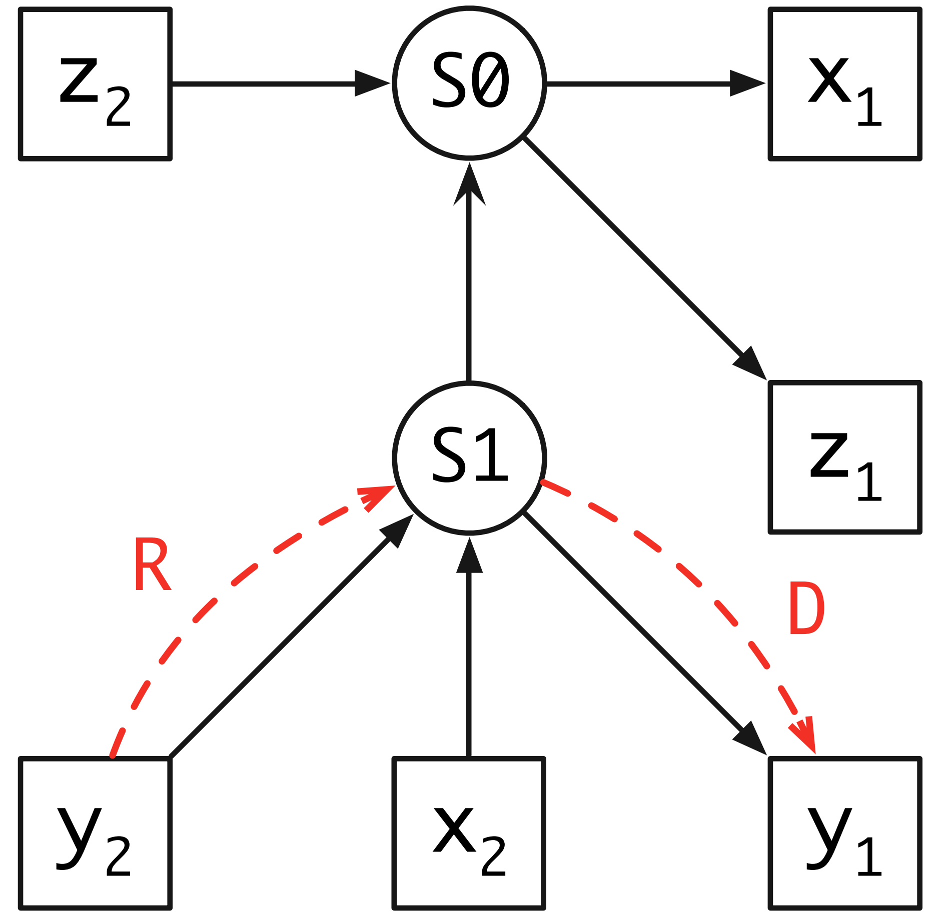


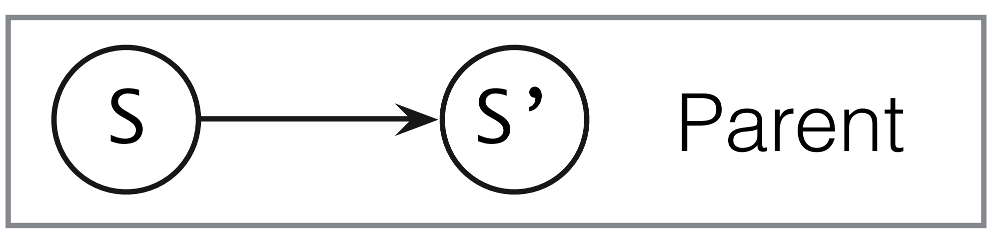




Lexical Scoping



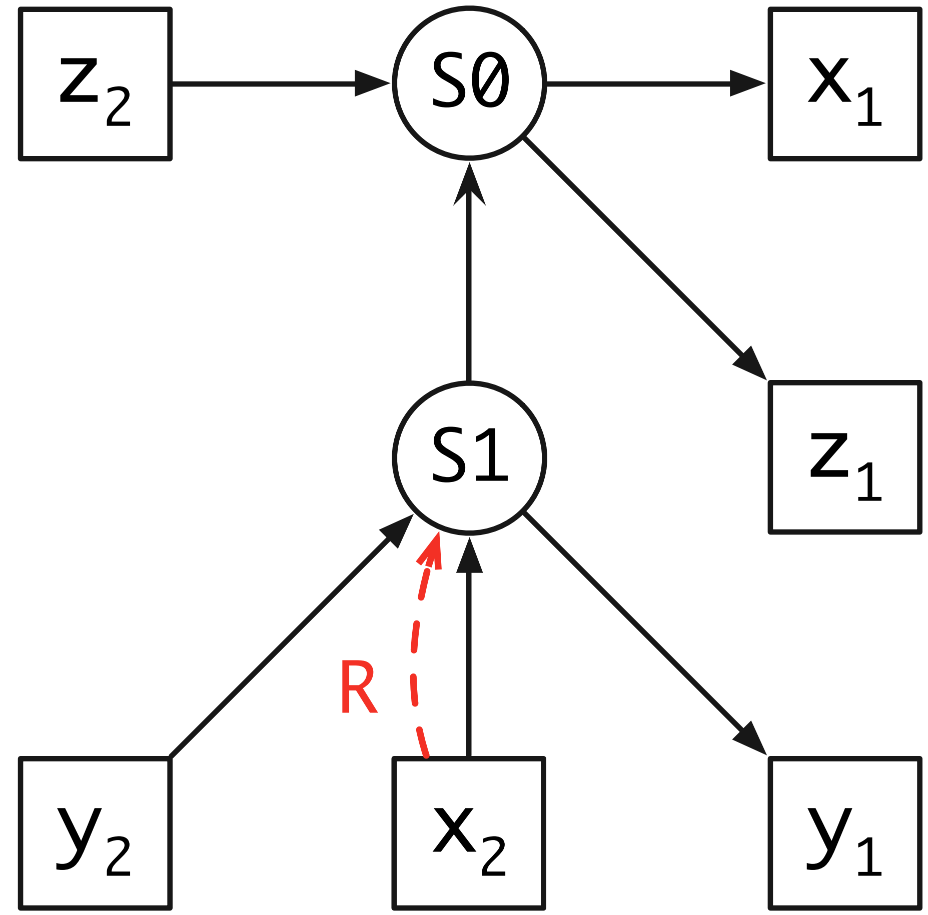


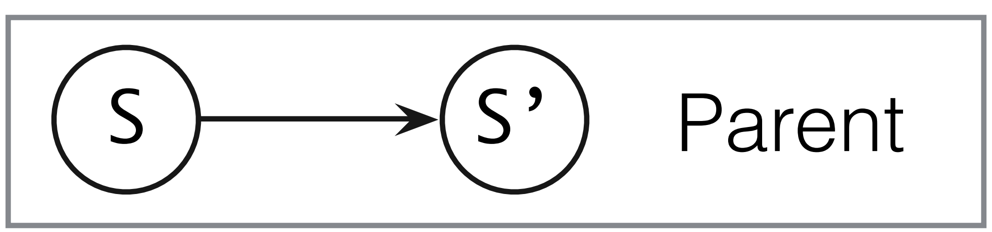


Lexical Scoping

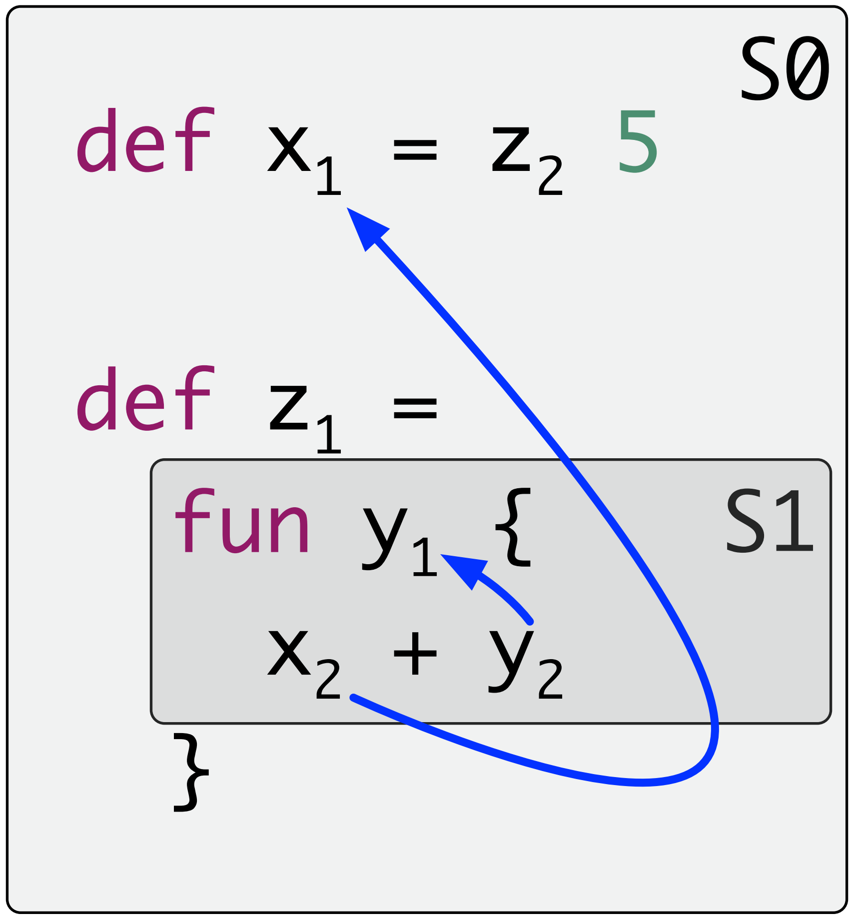
S0

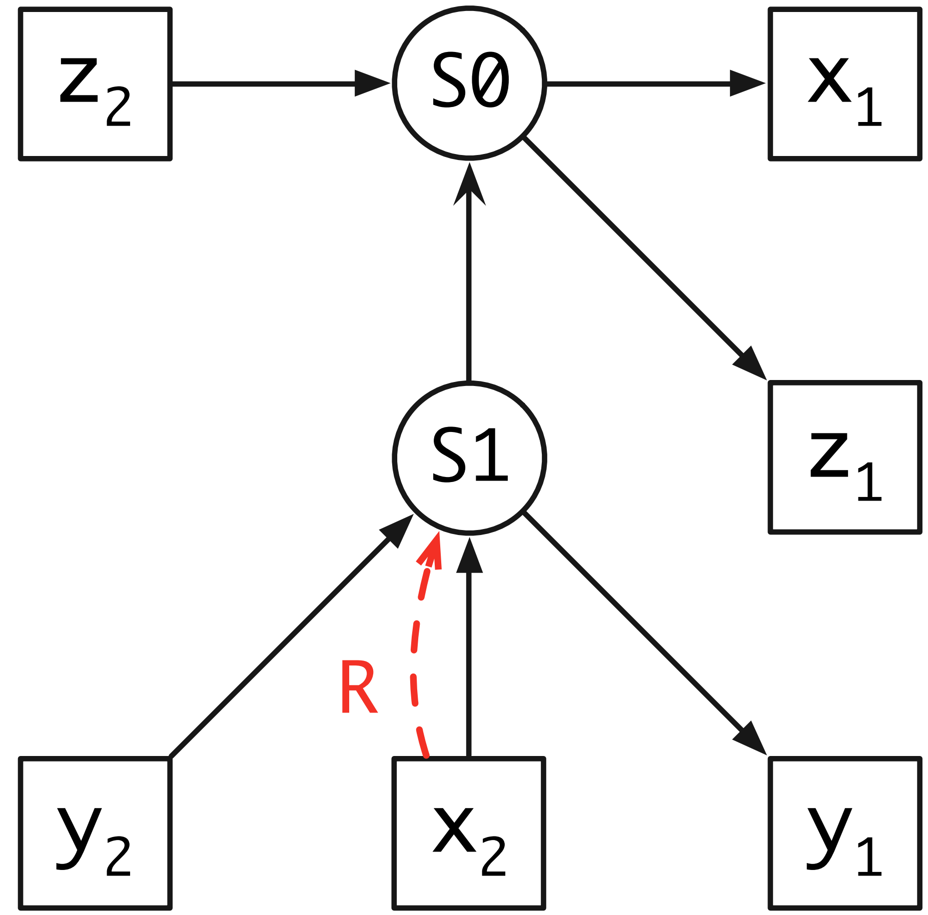
```
def x1 = z2 5

def z1 =
  fun y1 {          S1
    x2 + y2
  }
```

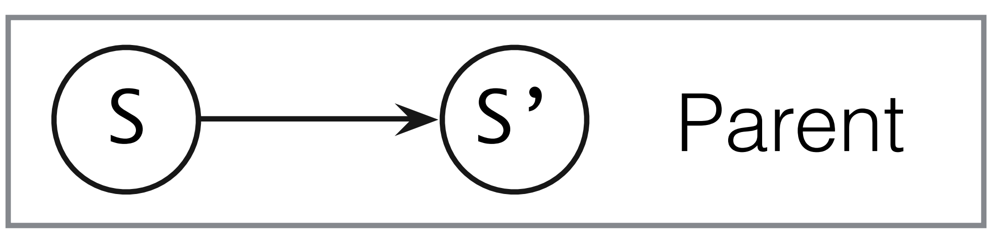


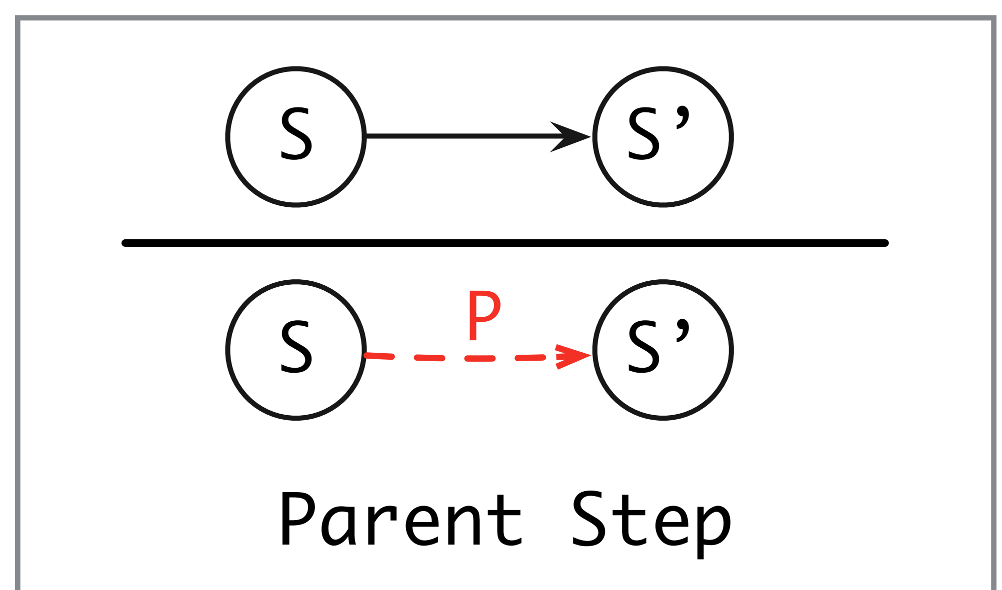


Lexical Scoping

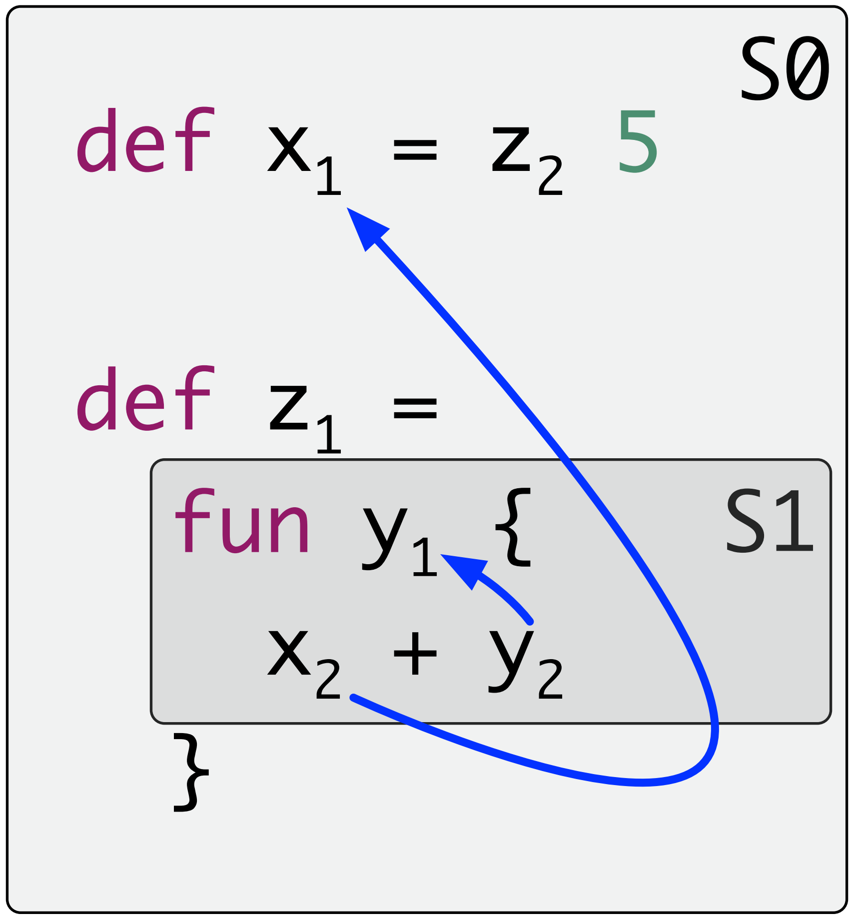


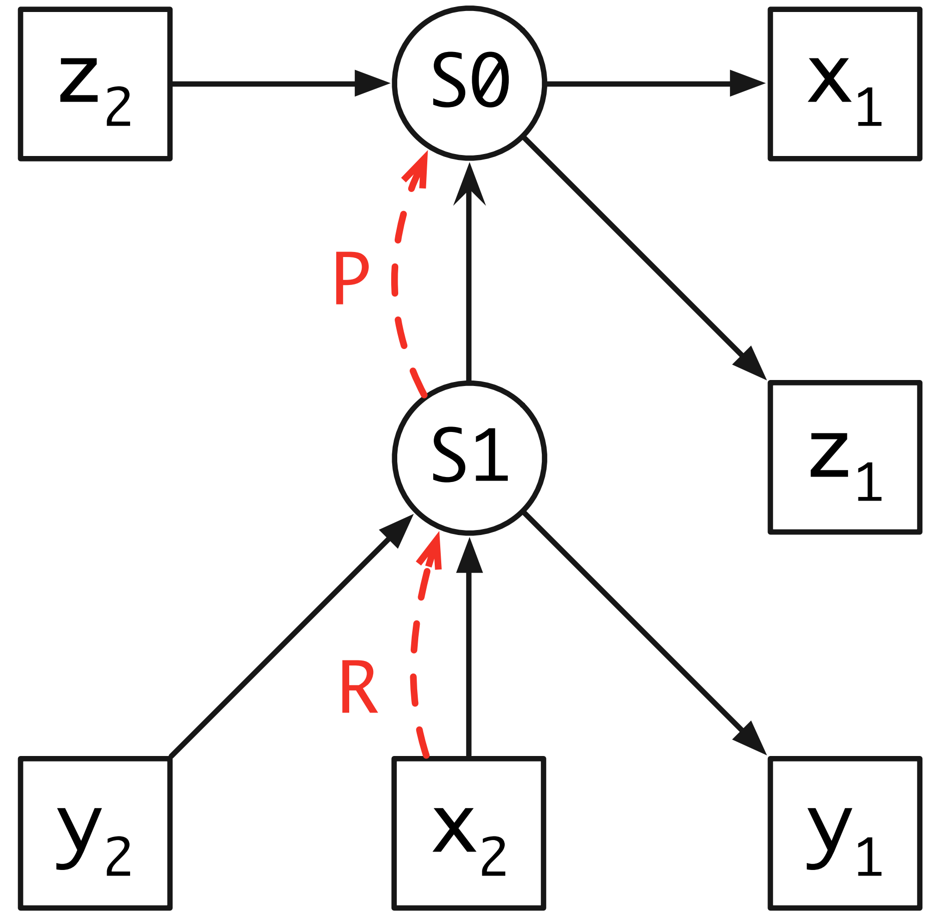


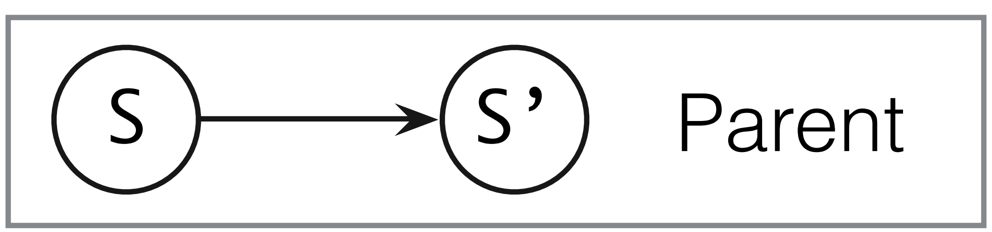


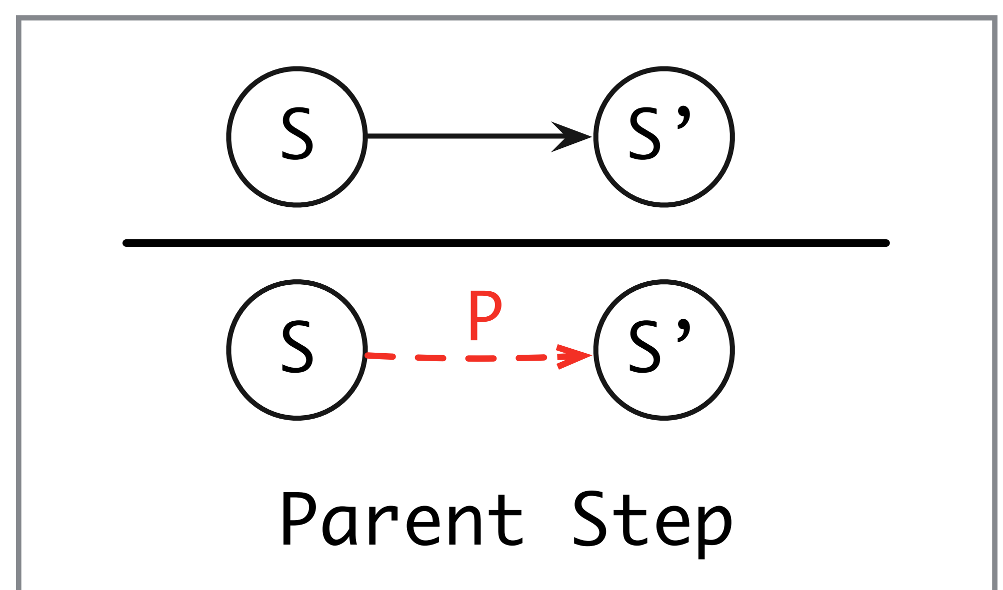


Lexical Scoping

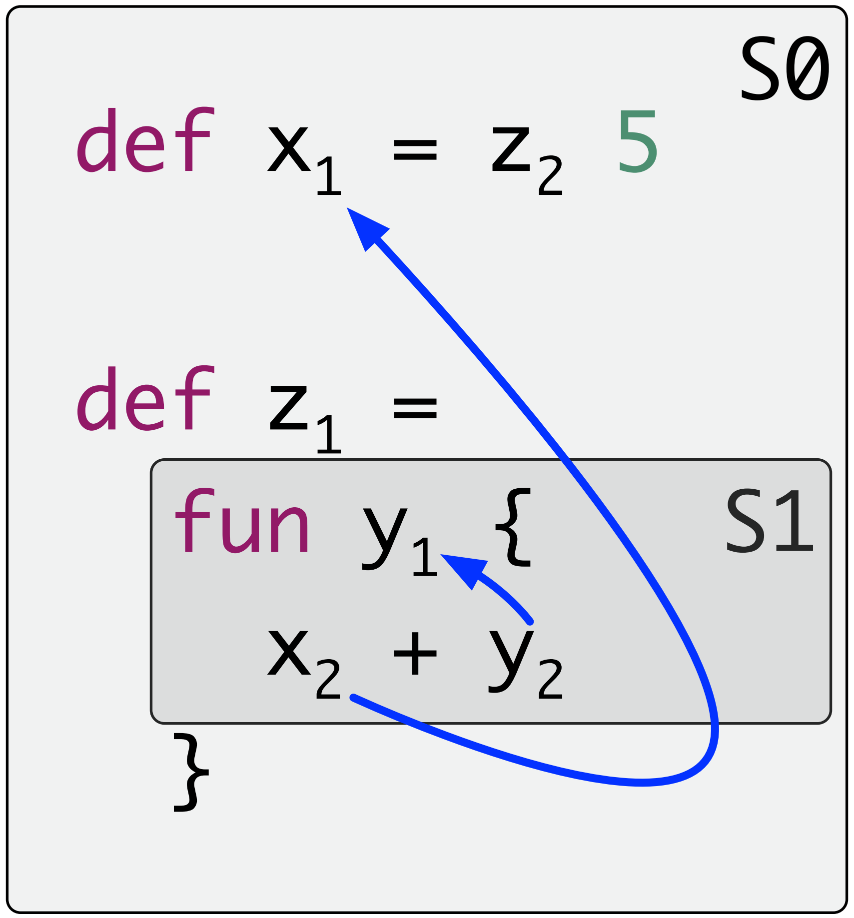


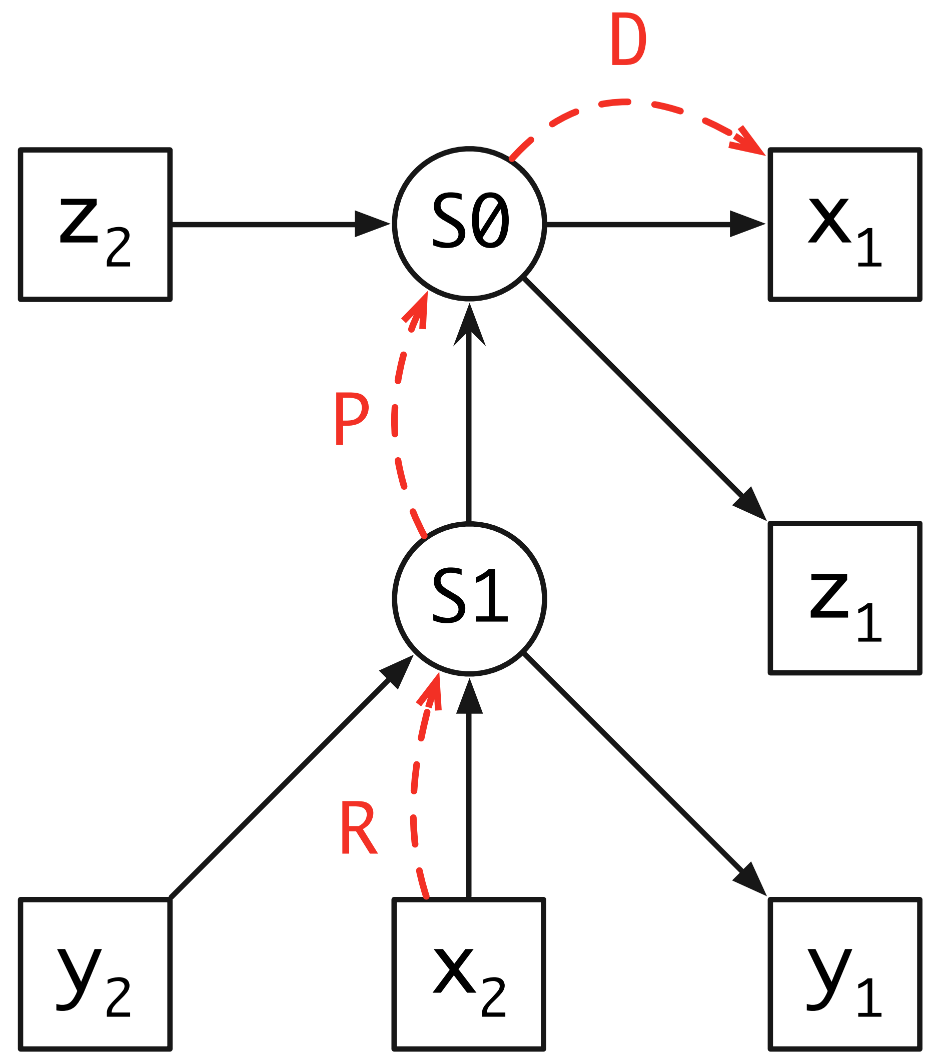


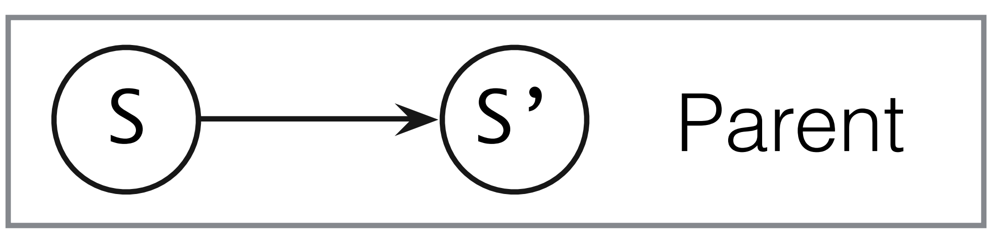


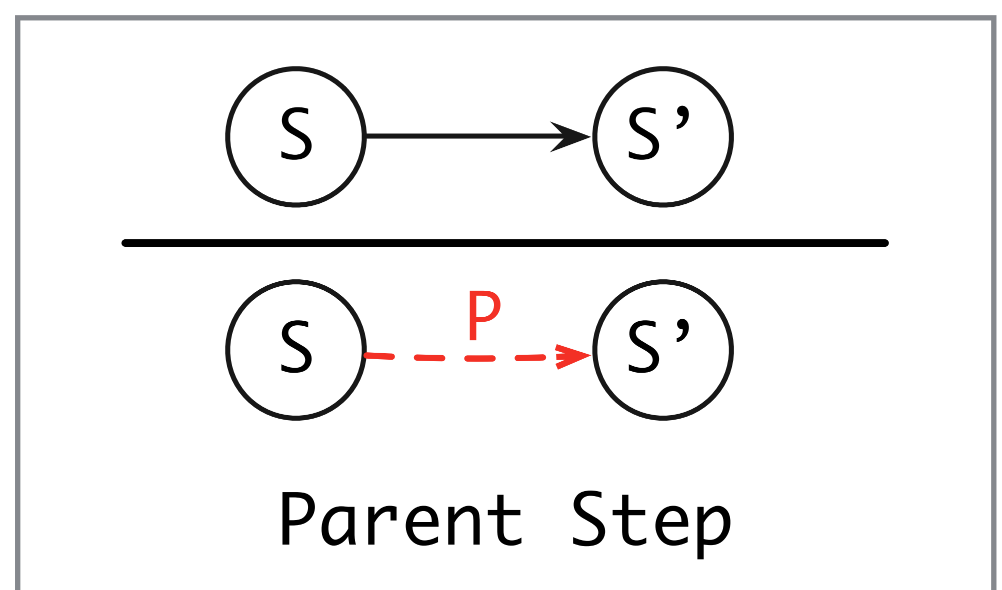


Lexical Scoping

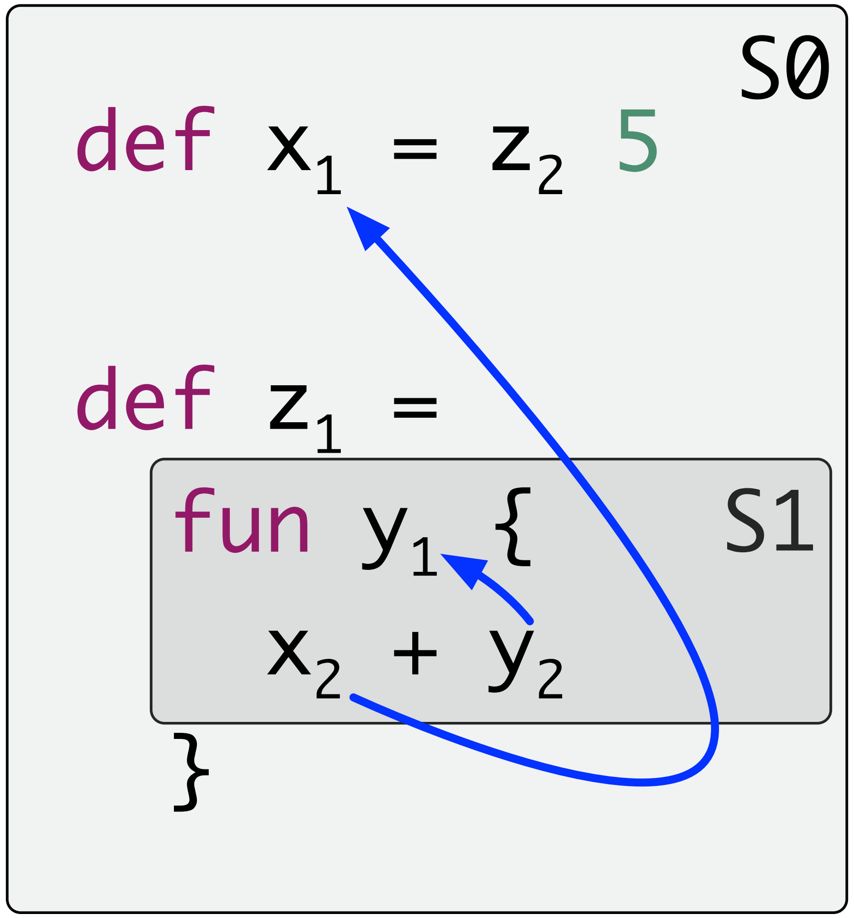


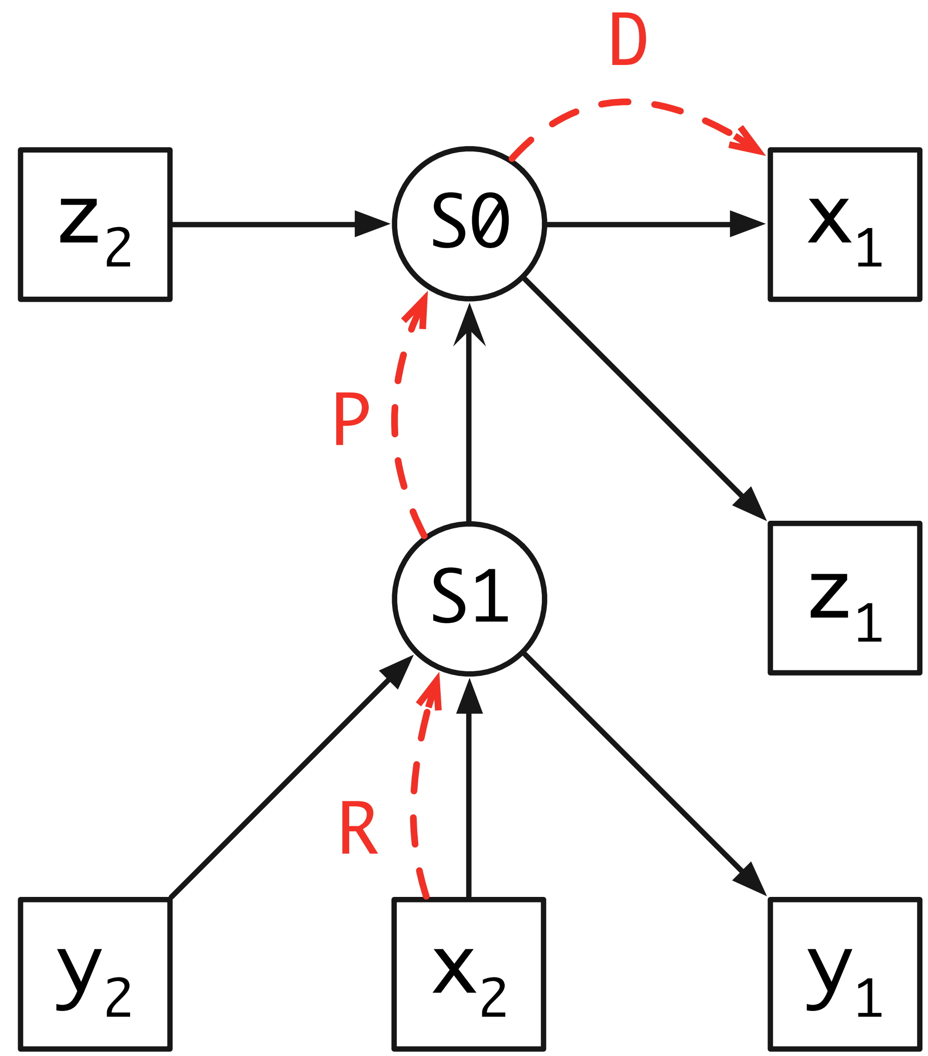


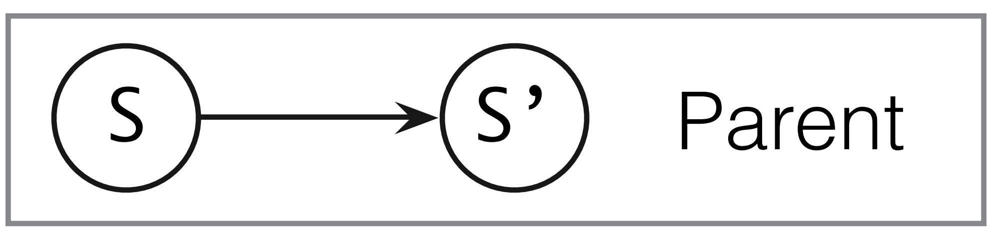




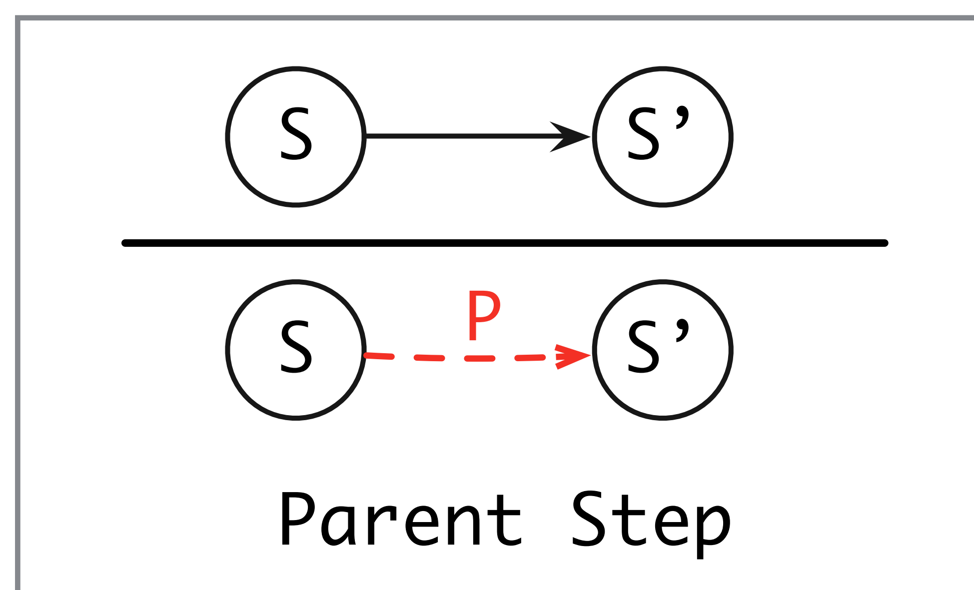
Lexical Scoping







Well formed path: R.P*.D



Shadowing

$$
\begin{array}{l}
\texttt{def } x_3 = z_2\ 5\ 7 \\
\\
\texttt{def } z_1 = \\
\quad \texttt{fun } x_1\ \{ \\
\quad\quad \texttt{fun } y_1\ \{ \\
\quad\quad\quad x_2 + y_2 \\
\quad\quad \} \\
\quad \}
\end{array}
$$

Shadowing

S0

def x_3 = z_2 5 7

def z_1 =

S1

fun x_1 {

S2

fun y_1 {

$x_2 + y_2$

}

}

Shadowing

$\mathtt{def}\ x_3 = z_2\ 5\ 7$ S0

$\mathtt{def}\ z_1 =$

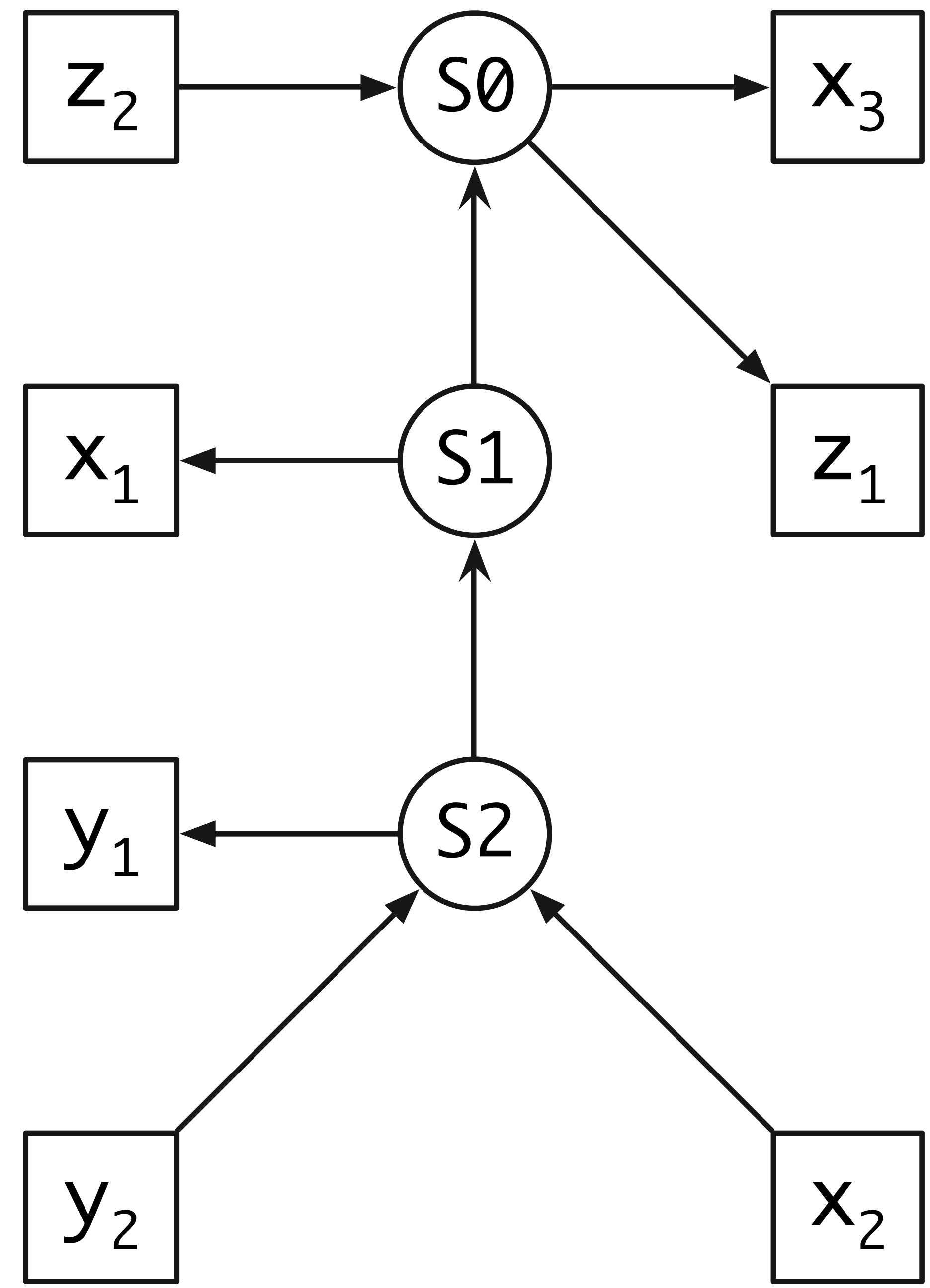
$\quad \mathtt{fun}\ x_1\ \{$ S1

$\quad\quad \mathtt{fun}\ y_1\ \{$ S2

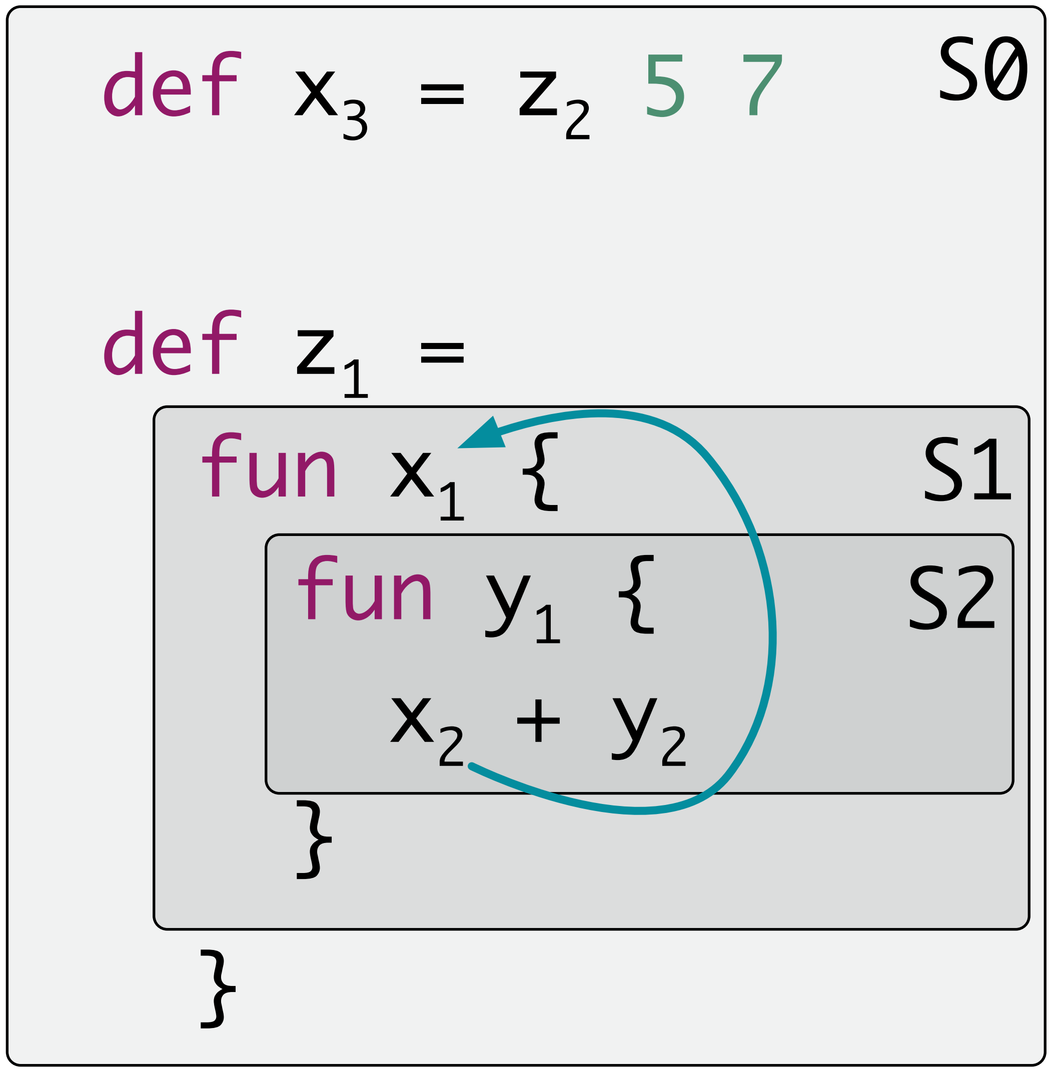
$\quad\quad\quad x_2 + y_2$

$\quad\quad \}$

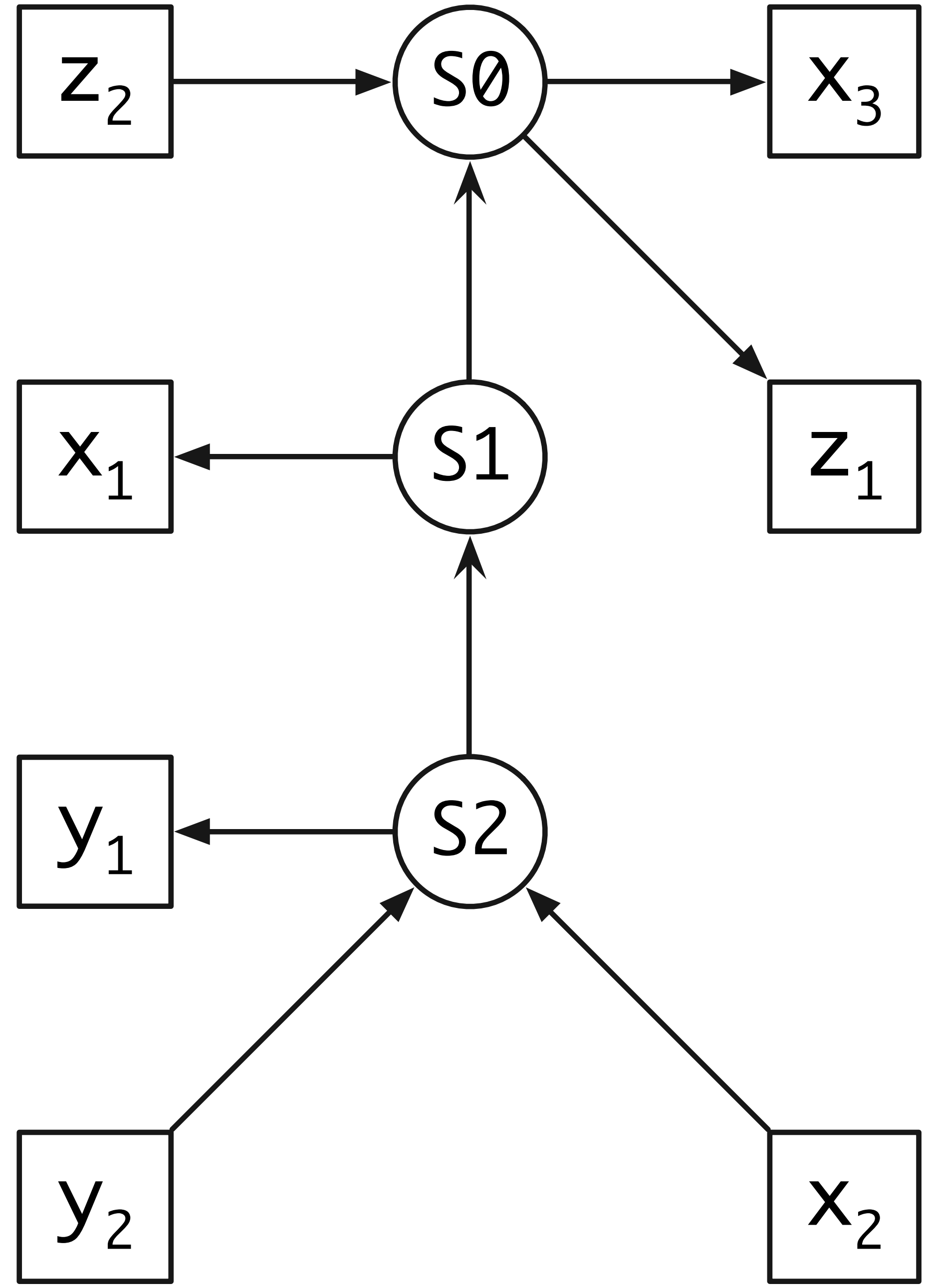
$\quad \}$



Shadowing

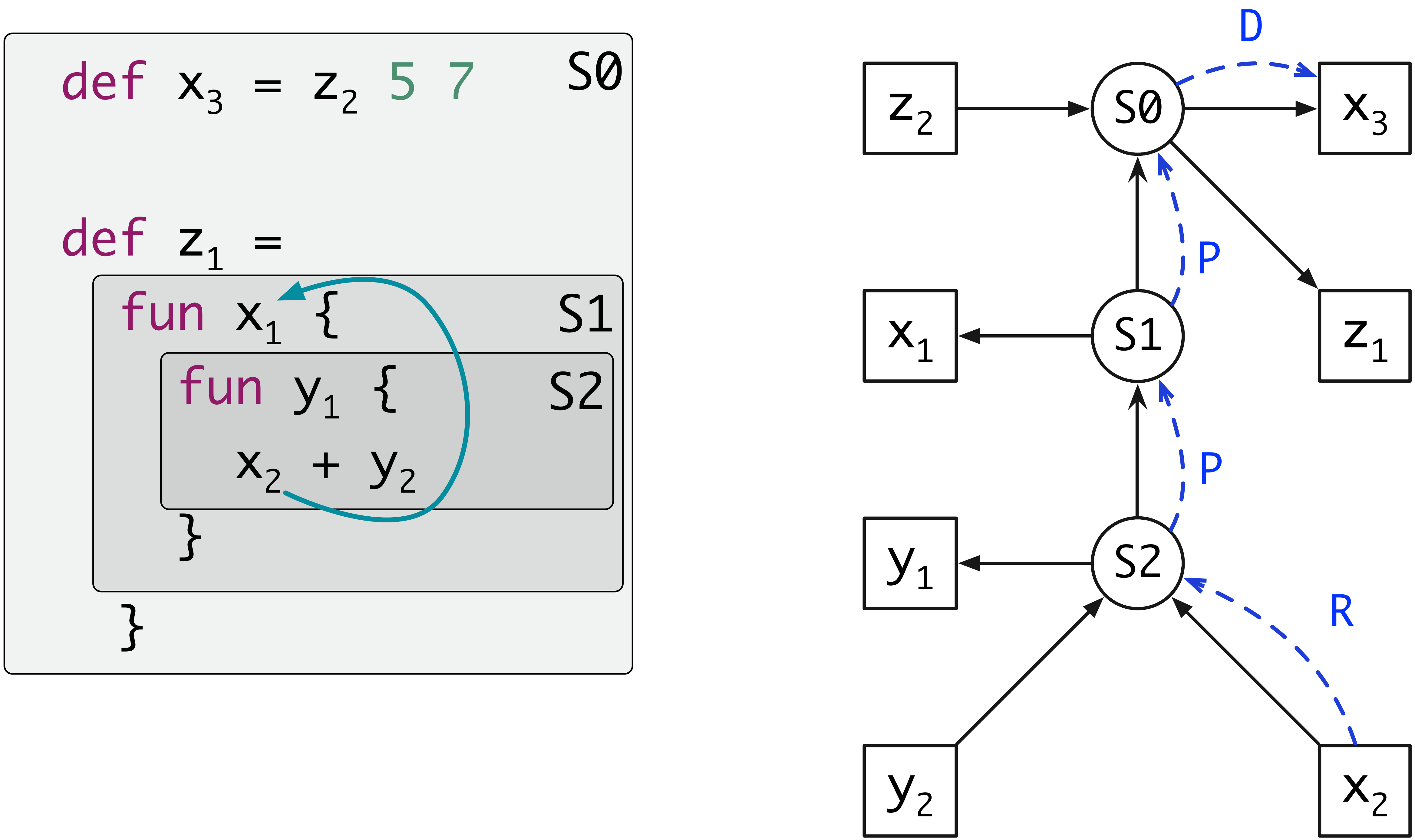


def x3 = z2 5 7
S0
def z1 =
fun x1 {
S1
fun y1 {
S2
x2 + y2
}
}



z2
S0
x3
x1
S1
z1
y1
S2
y2
x2

Shadowing



def x3 = z2 5 7
S0
def z1 =
fun x1 {
S1
fun y1 {
S2
x2 + y2
}
}
D
P
P
R
z2
S0
x3
x1
S1
z1
y1
S2
y2
x2

Shadowing

S0

$\mathtt{def}\ x_3 = z_2\ 5\ 7$

$\mathtt{def}\ z_1 =$

S1

$\mathtt{fun}\ x_1\ \{$

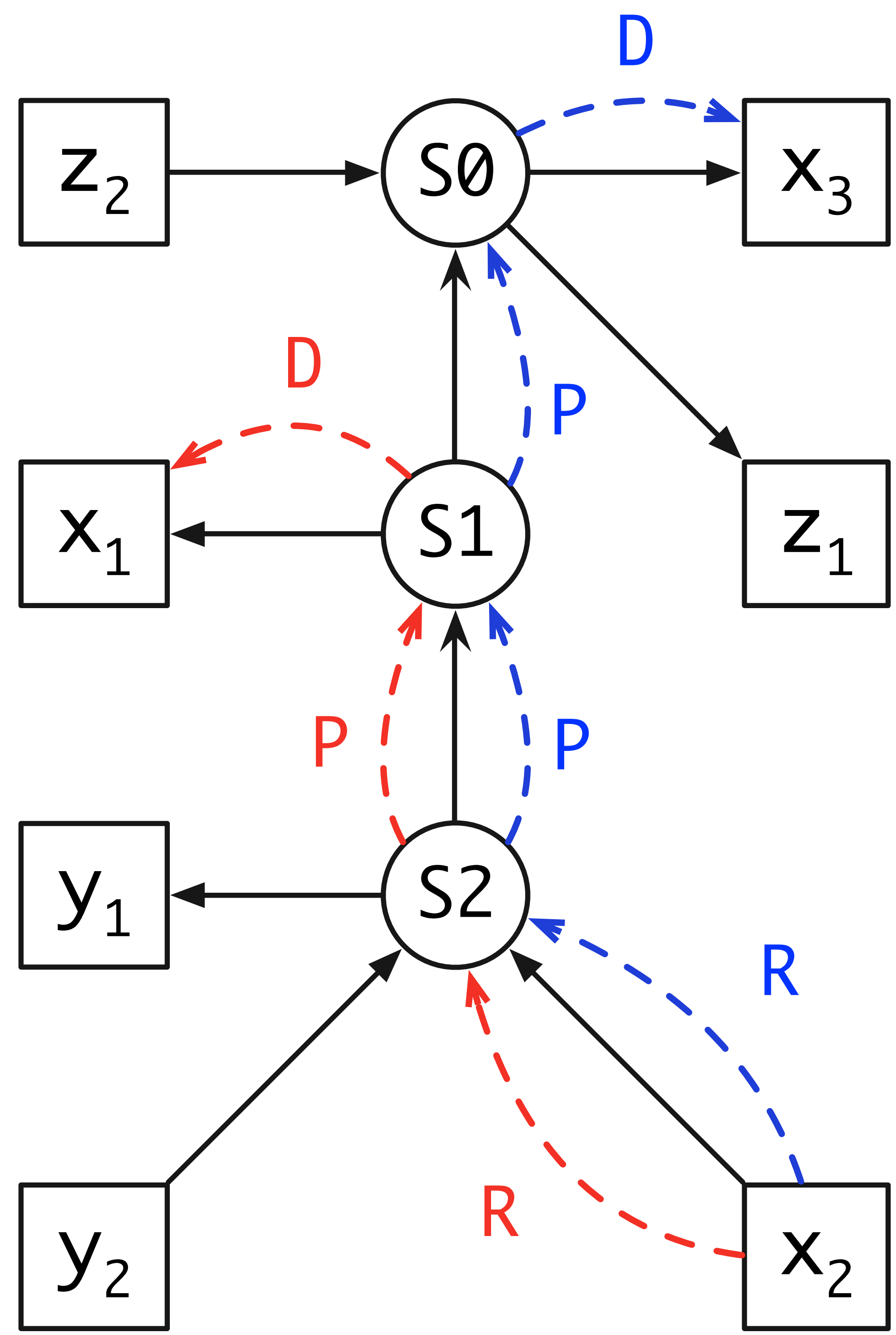
S2

$\mathtt{fun}\ y_1\ \{$

$x_2 + y_2$

$\}$

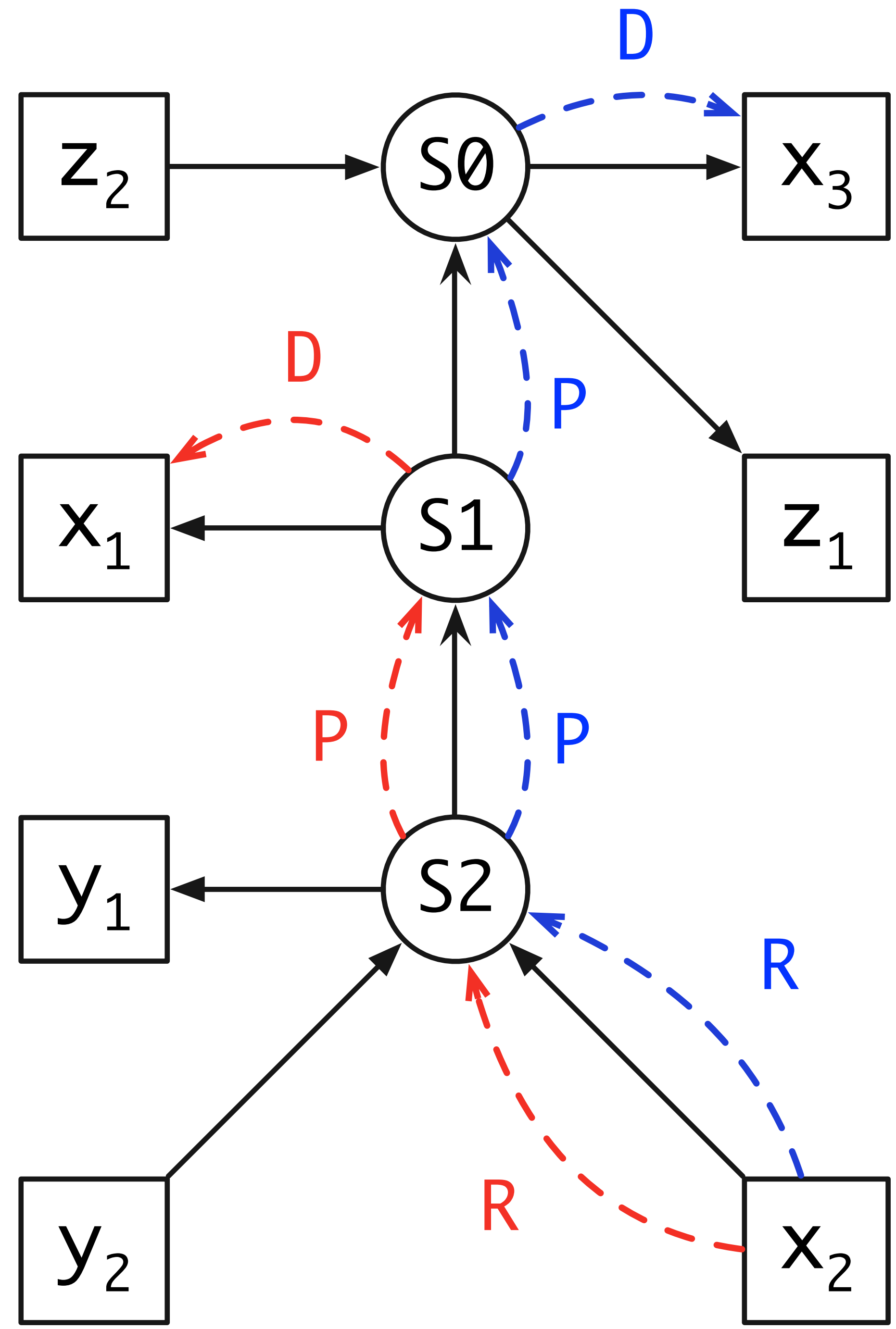
$\}$



Shadowing

```
def x3 = z2 5 7          S0

def z1 =
  fun x1 {               S1
    fun y1 {             S2
      x2 + y2
    }
  }
```

$D < P.p$

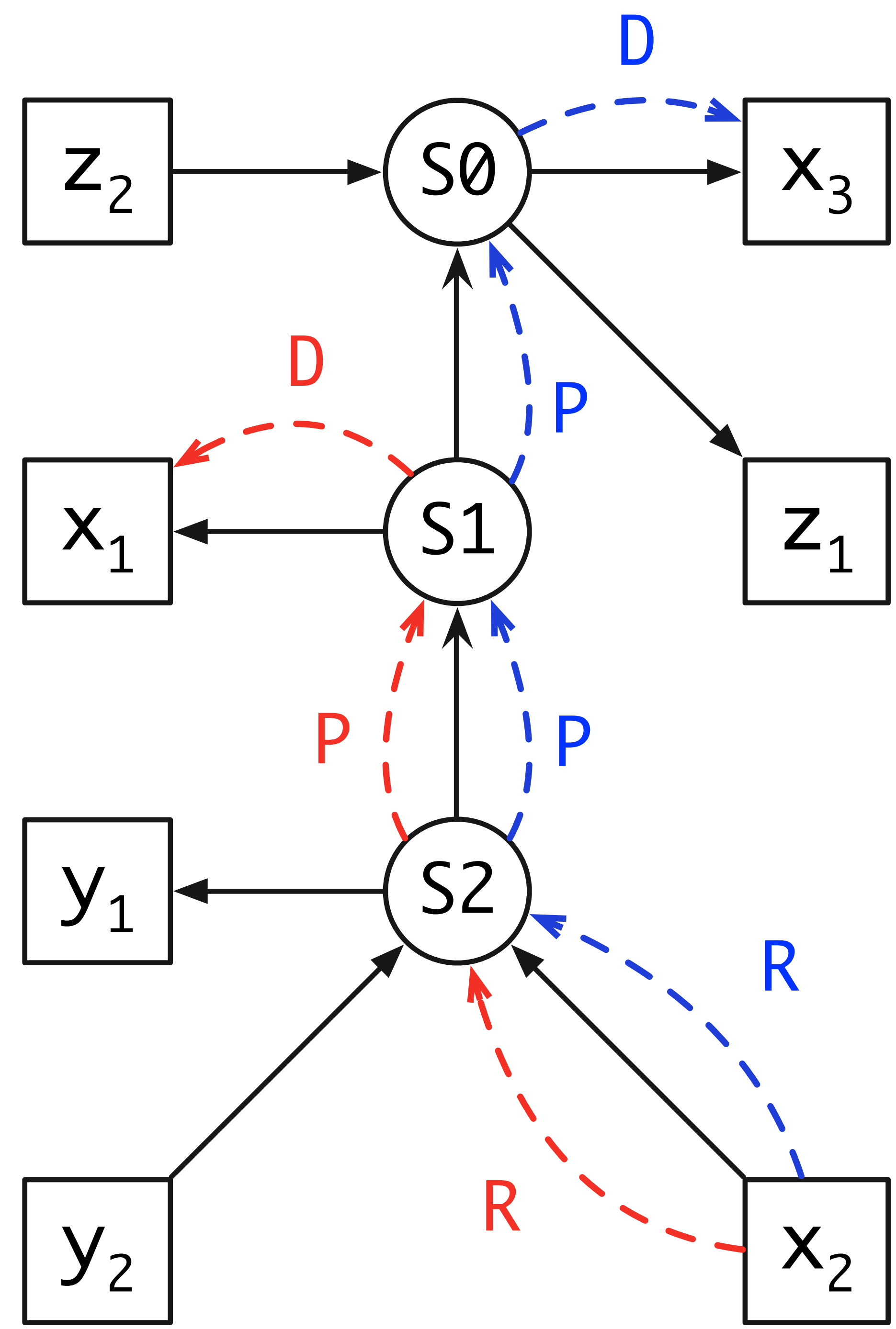


Shadowing

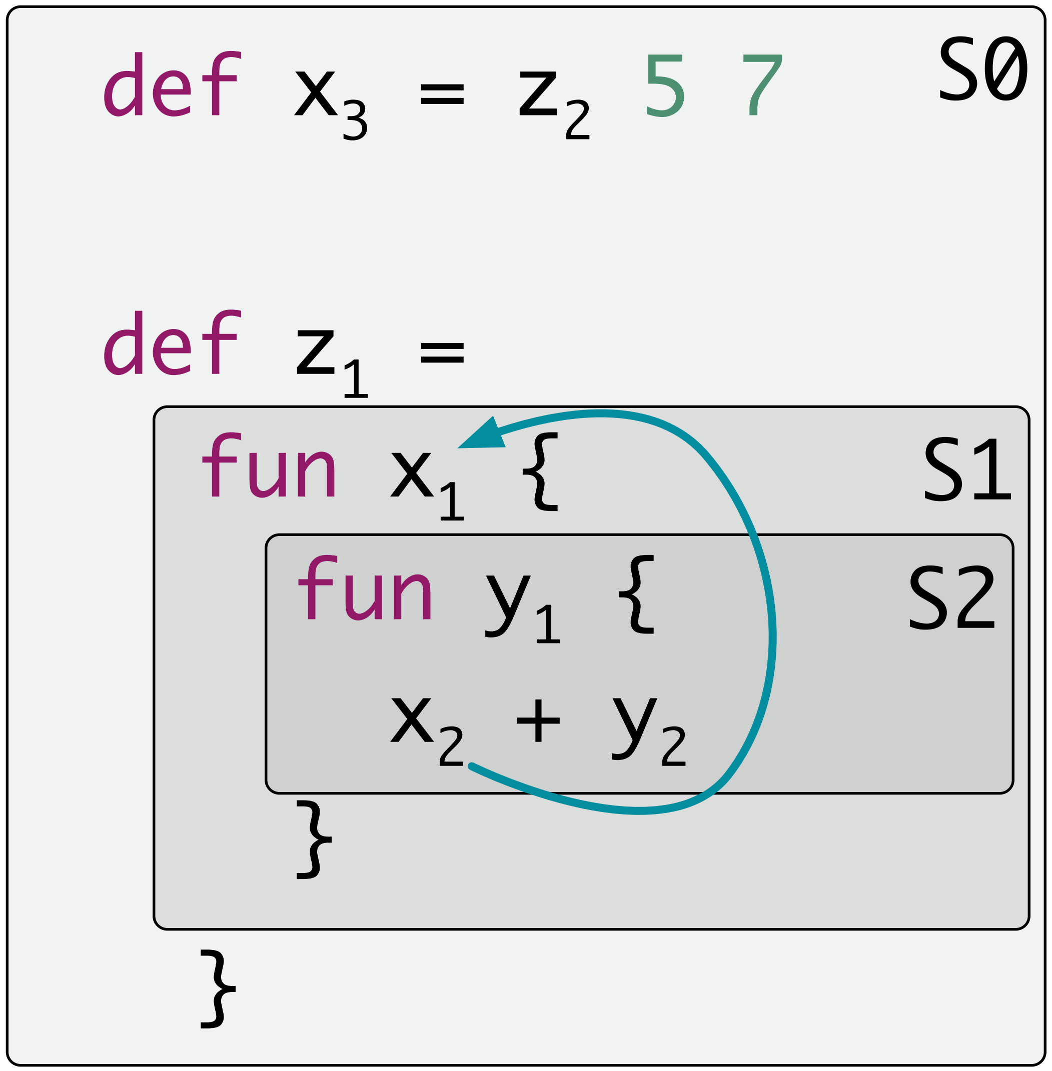
```
def x3 = z2 5 7                 S0

def z1 =
  fun x1 {                      S1
    fun y1 {                    S2
      x2 + y2
    }
  }
```

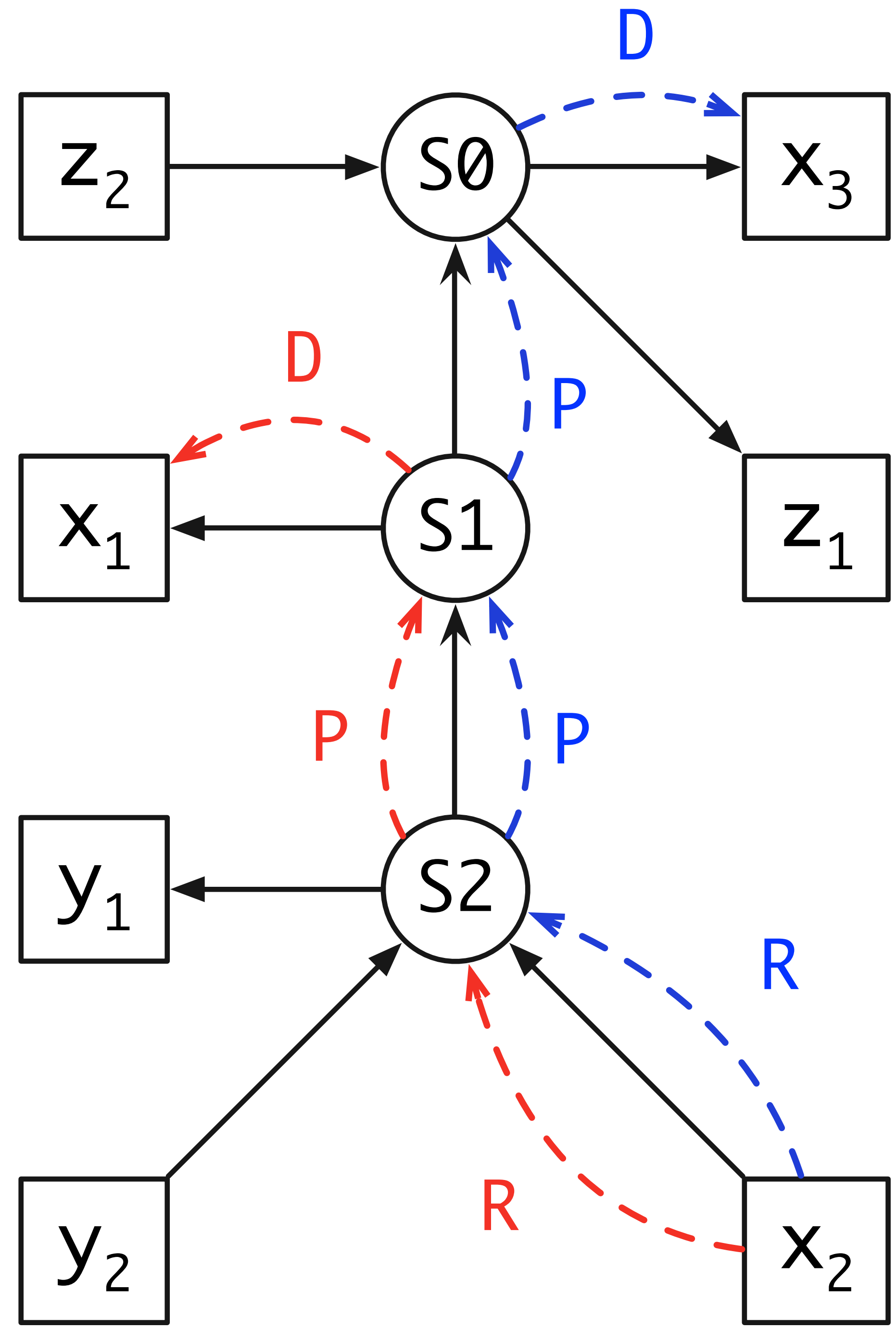
$$\frac{}{D < P.p}$$

$$\frac{p < p'}{s.p < s.p'}$$



Shadowing





$$\frac{}{D < P.p}$$

$$\frac{p < p'}{s.p < s.p'}$$

$$R.P.D < R.P.P.D$$

Imports

S0

$\texttt{module}\ A_1\ \{$

SA

$\quad \texttt{def}\ z_1 = 5$

$\}$

$\texttt{module}\ B_1\ \{$

SB

$\quad \texttt{import}\ A_2$

$\quad \texttt{def}\ x_1 = 1 + z_2$

$\}$

Imports

S0

module A_1 {

  def z_1 = 5   SA

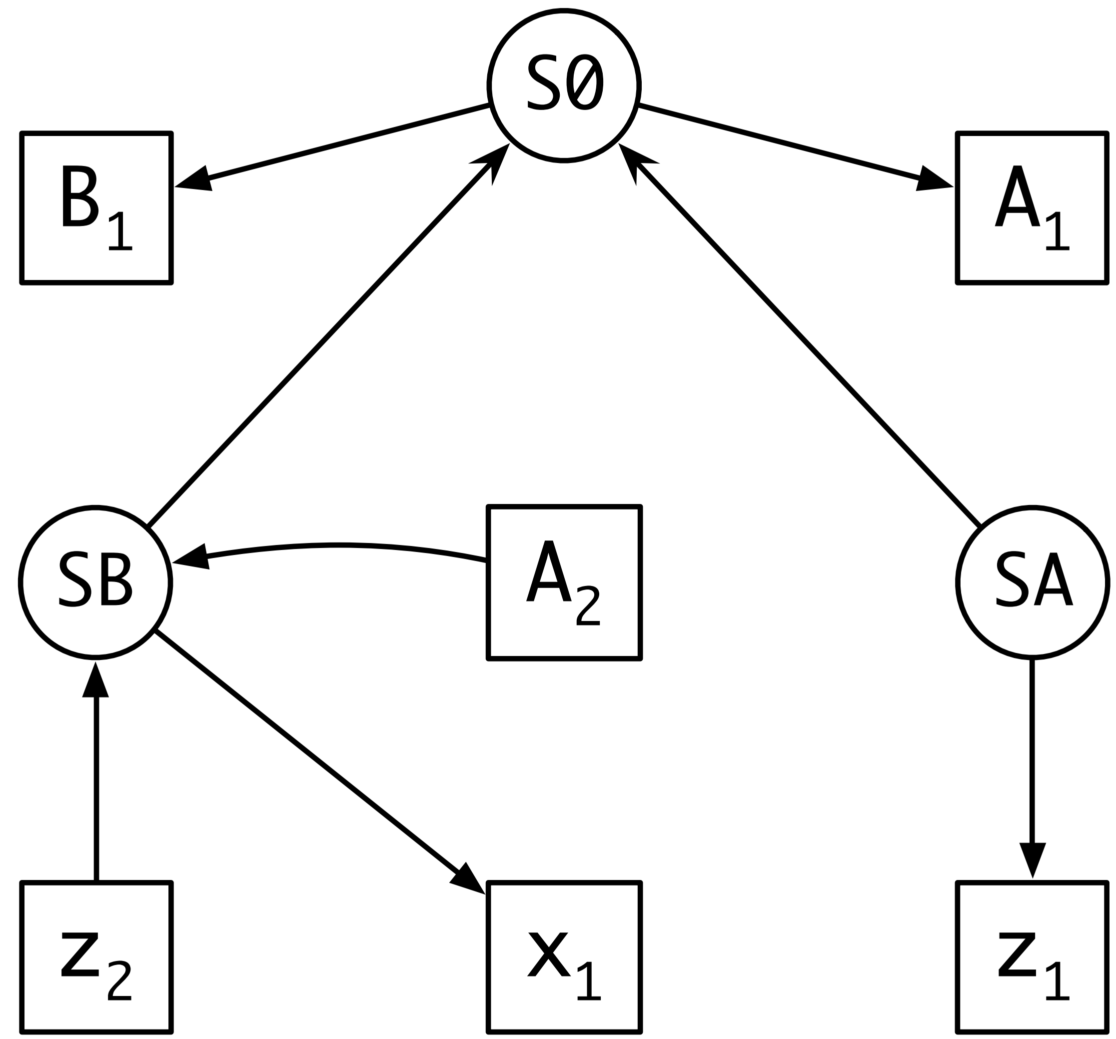
}

module B_1 {

  import A_2 SB

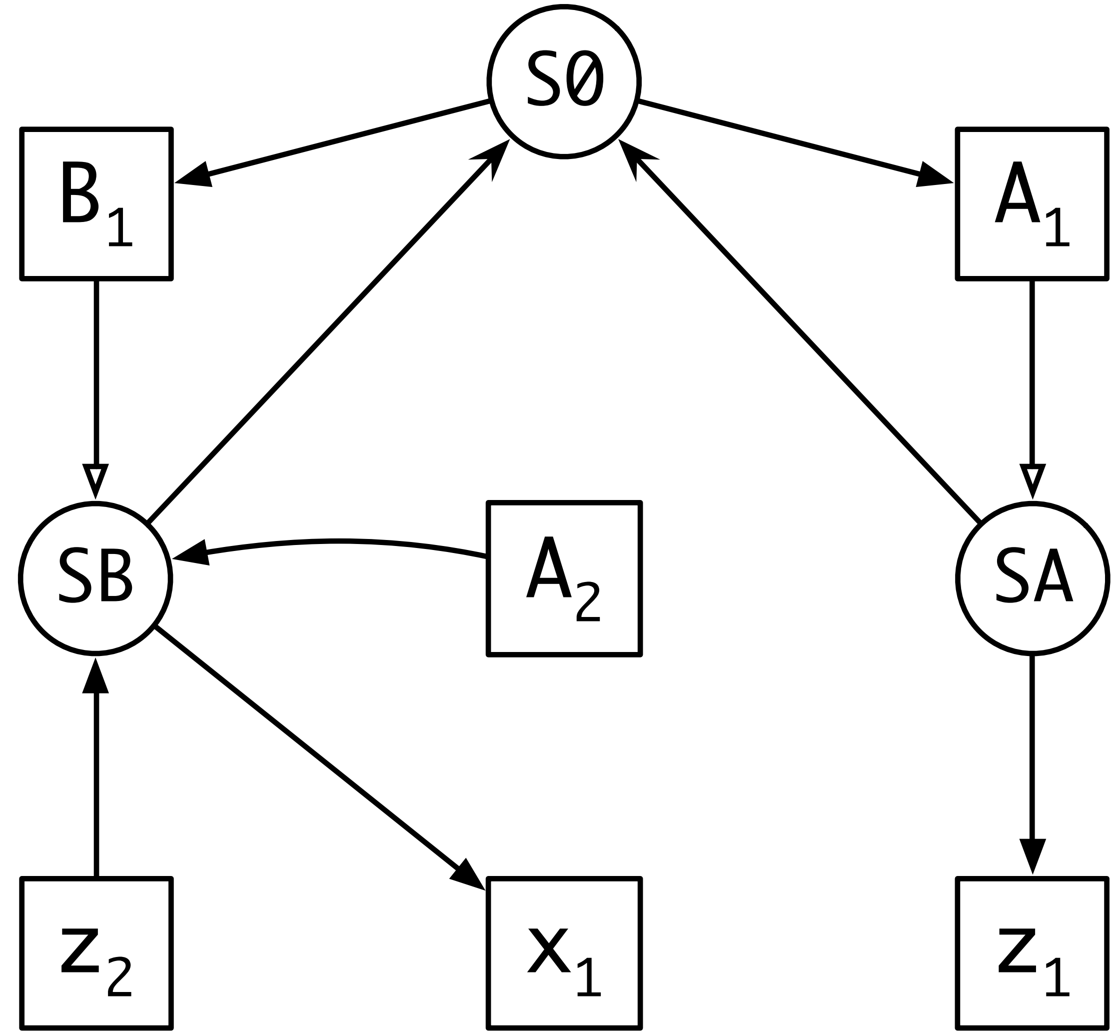
  def x_1 = 1 + z_2

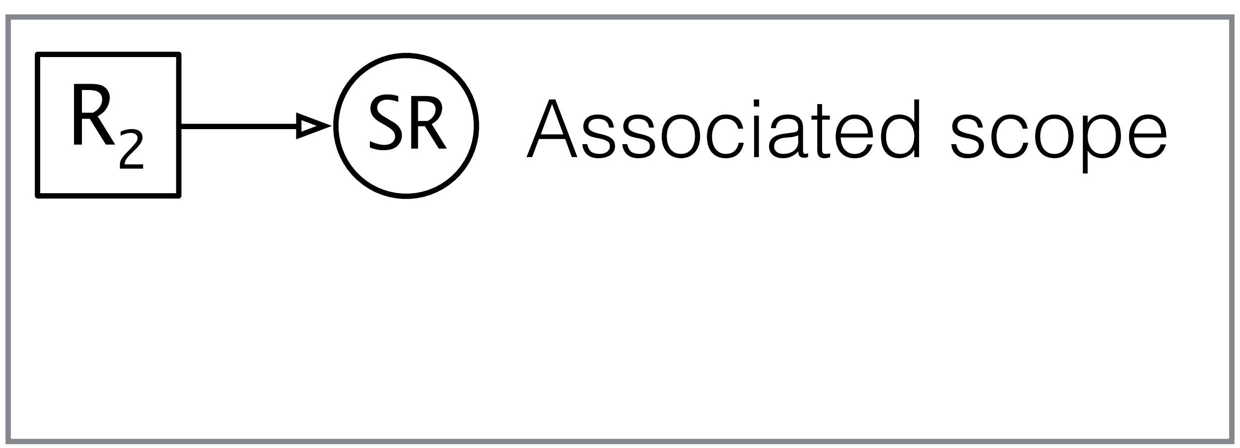
}



Imports

```
module A1 {                S0
  def z1 = 5           SA
}
module B1 {
  import A2            SB
  def x1 = 1 + z2
}
```





Imports

S0

module A_1 {

  def z_1 = 5   SA

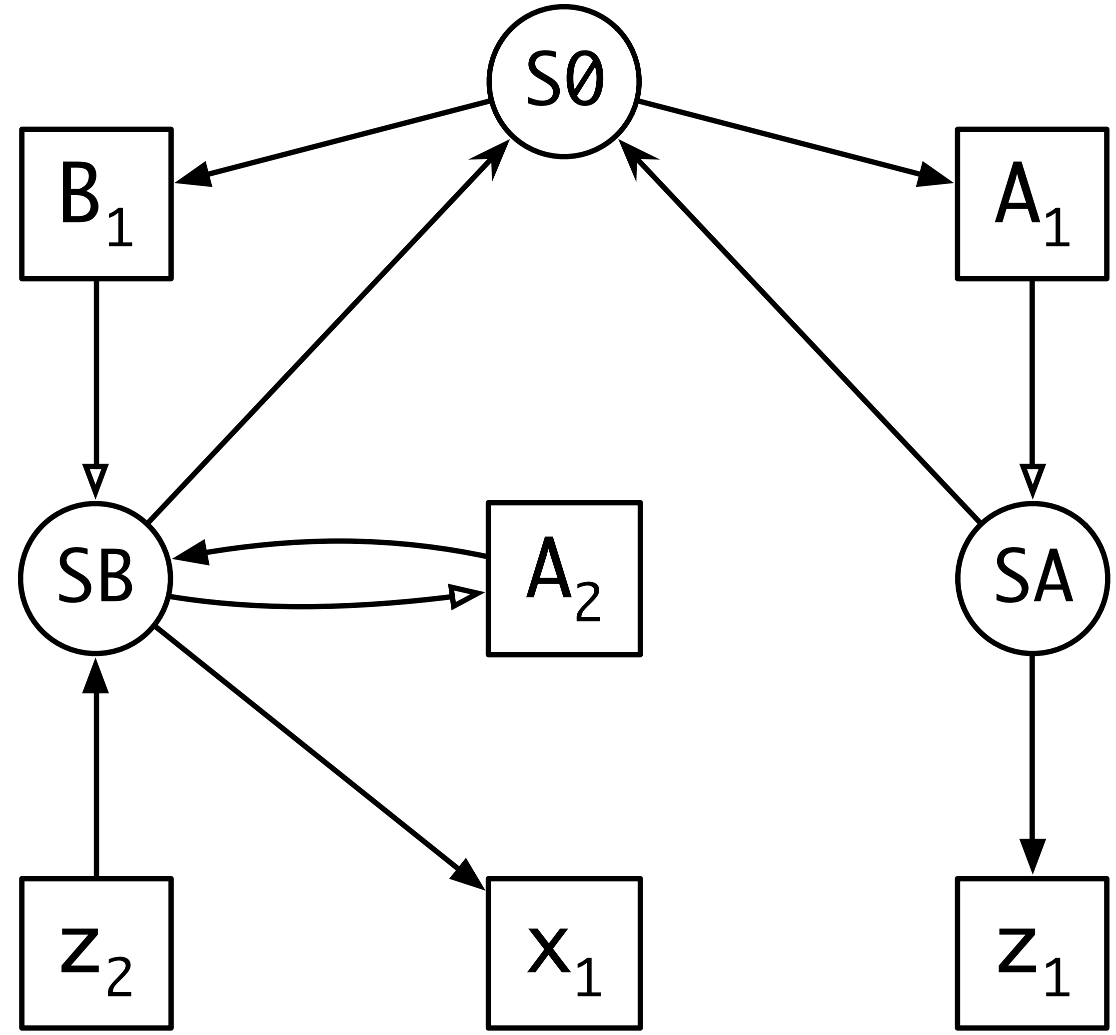
}

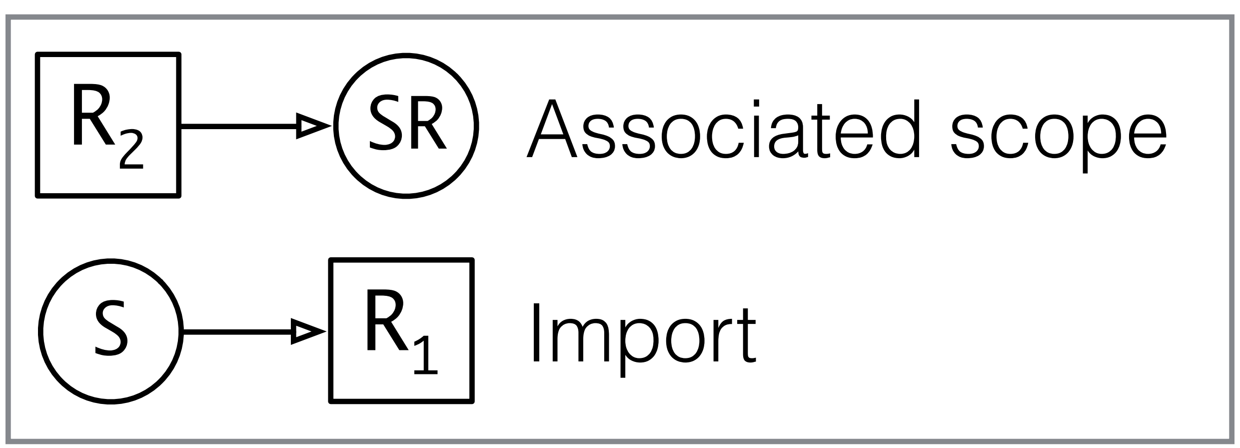
module B_1 {

  import A_2 SB

  def x_1 = 1 + z_2

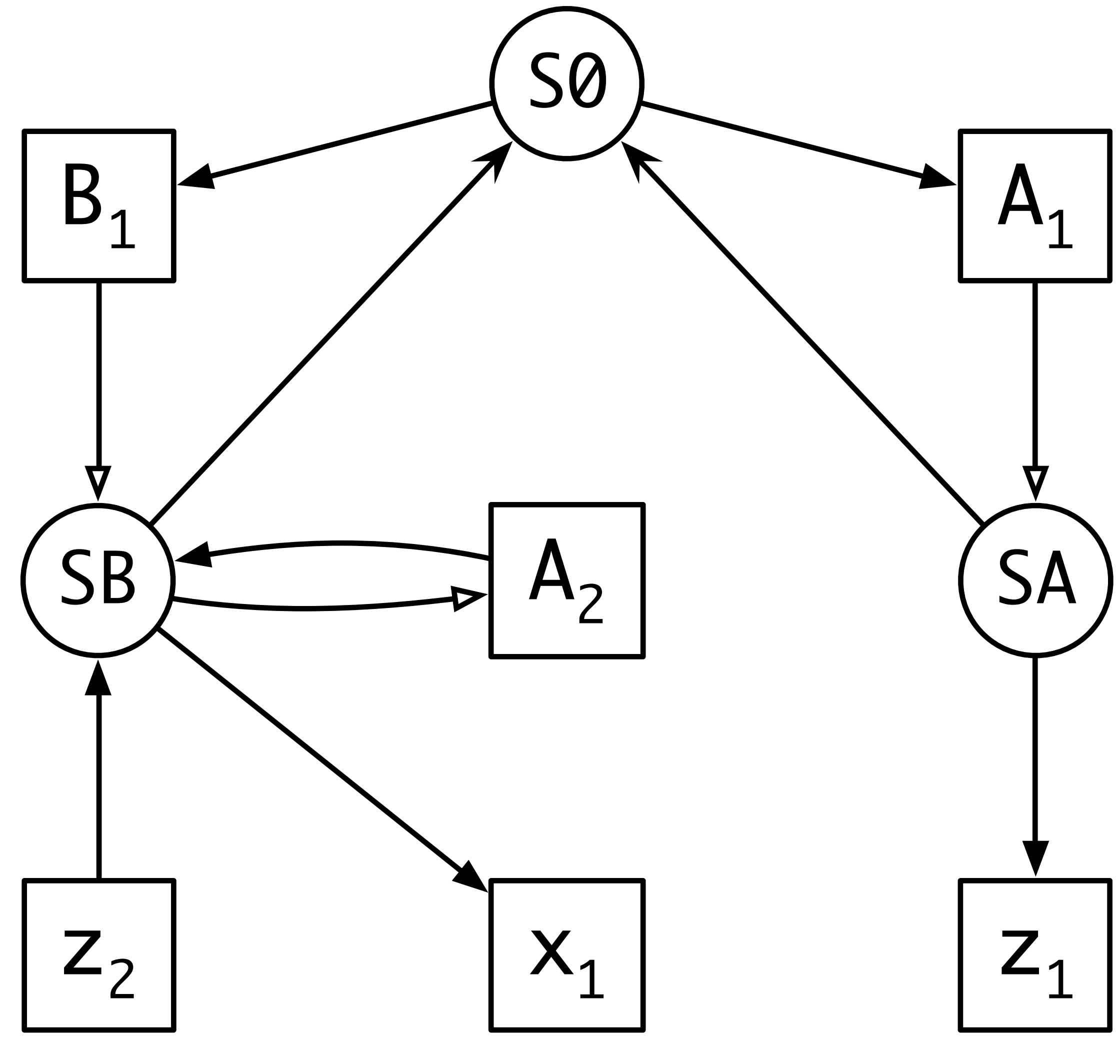
}

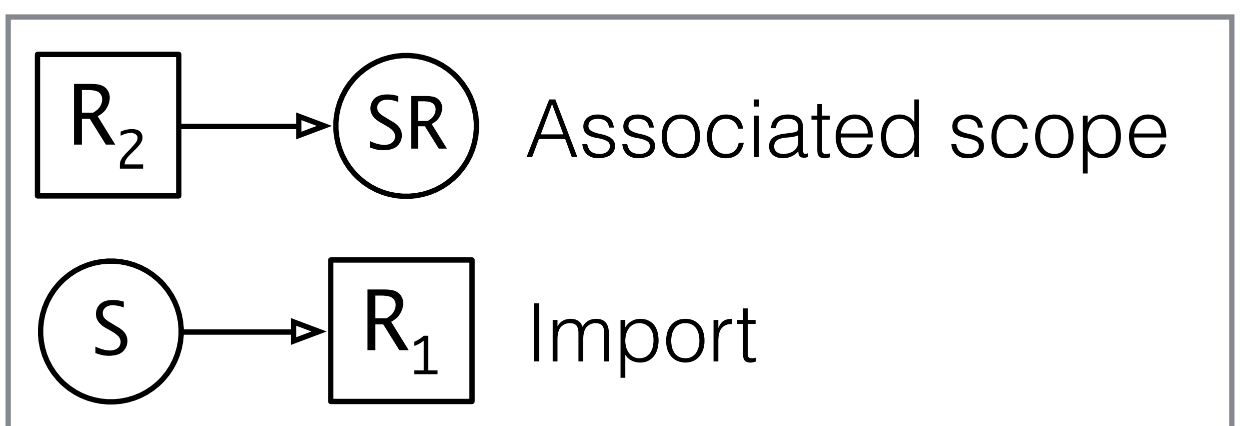




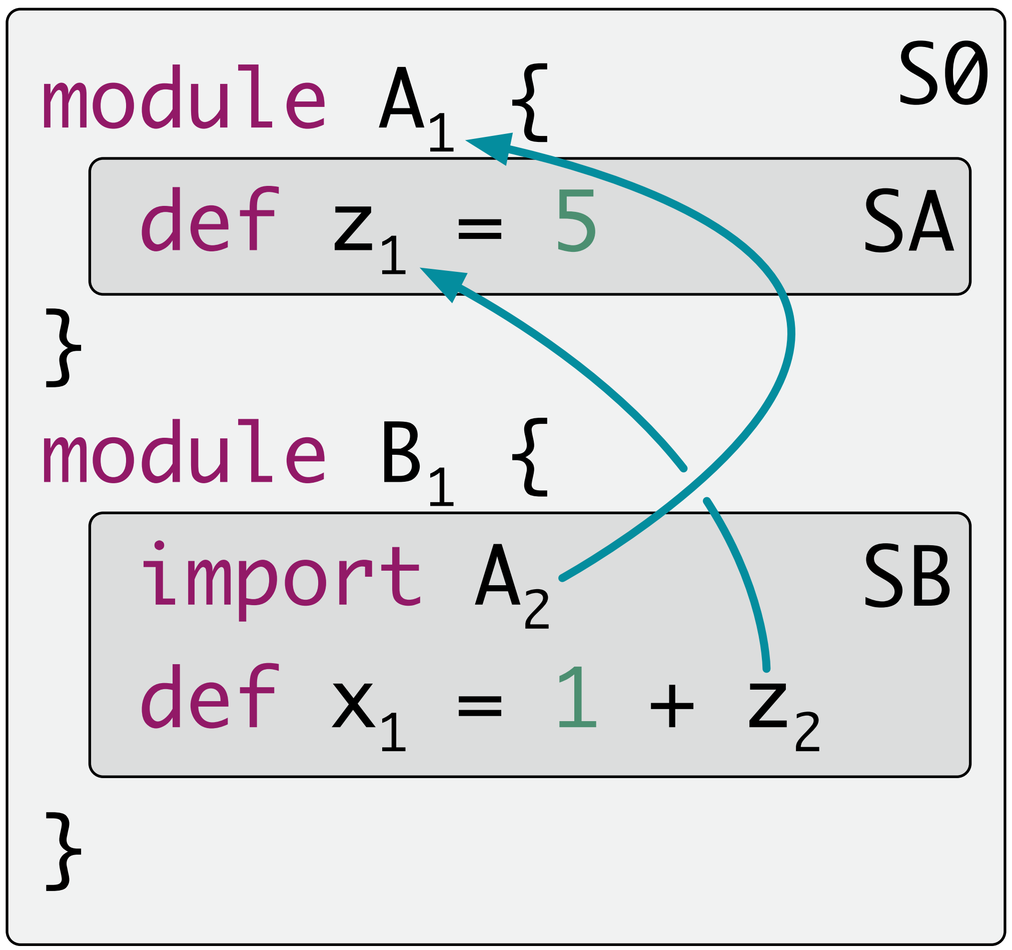
Imports

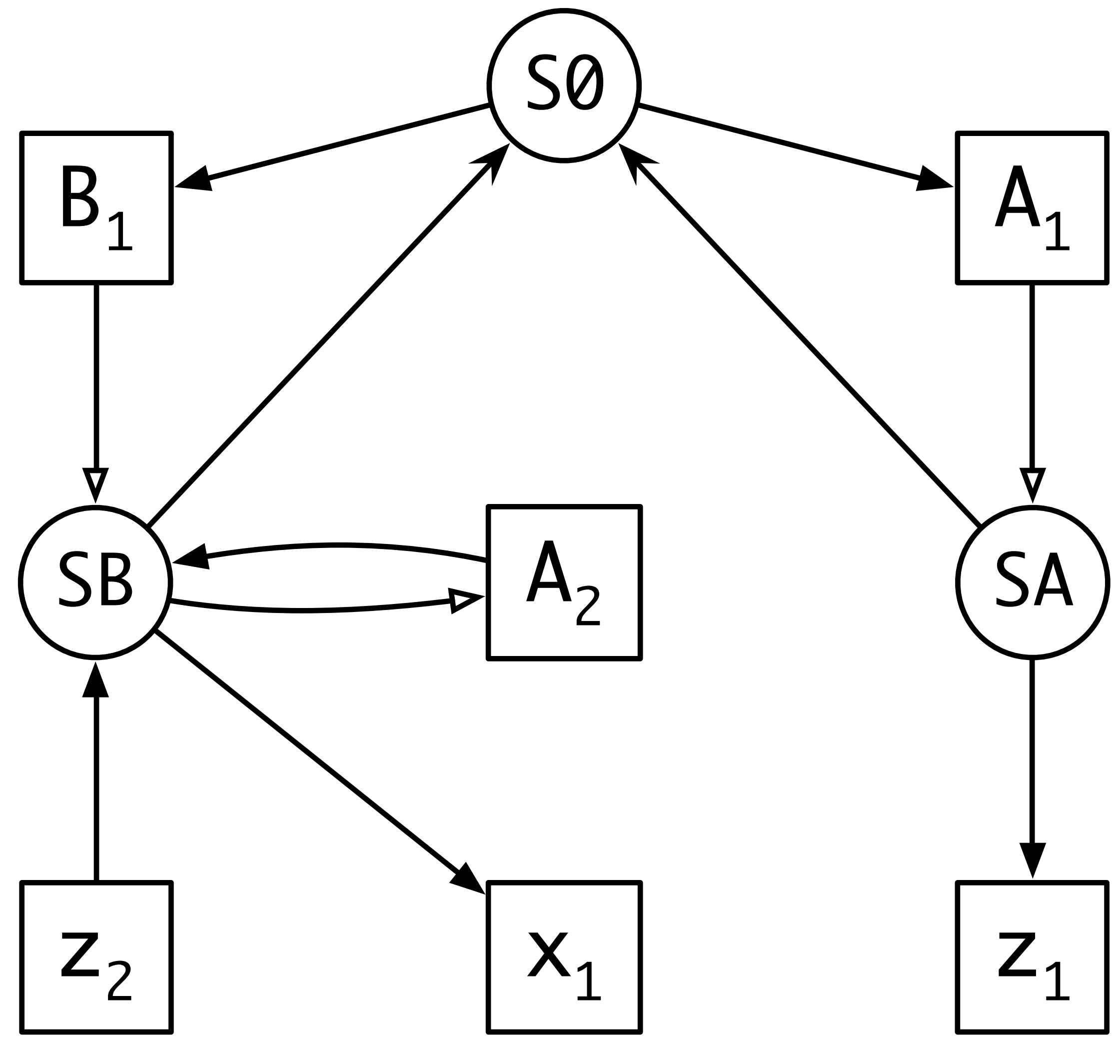
```
module A₁ {                S0
  def z₁ = 5               SA
}
module B₁ {
  import A₂                SB
  def x₁ = 1 + z₂
}
```

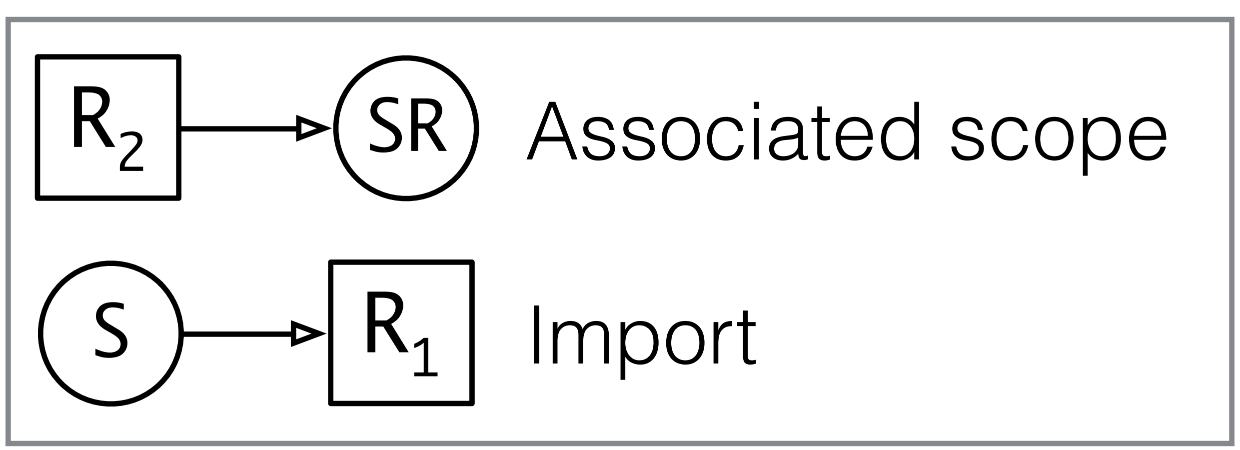




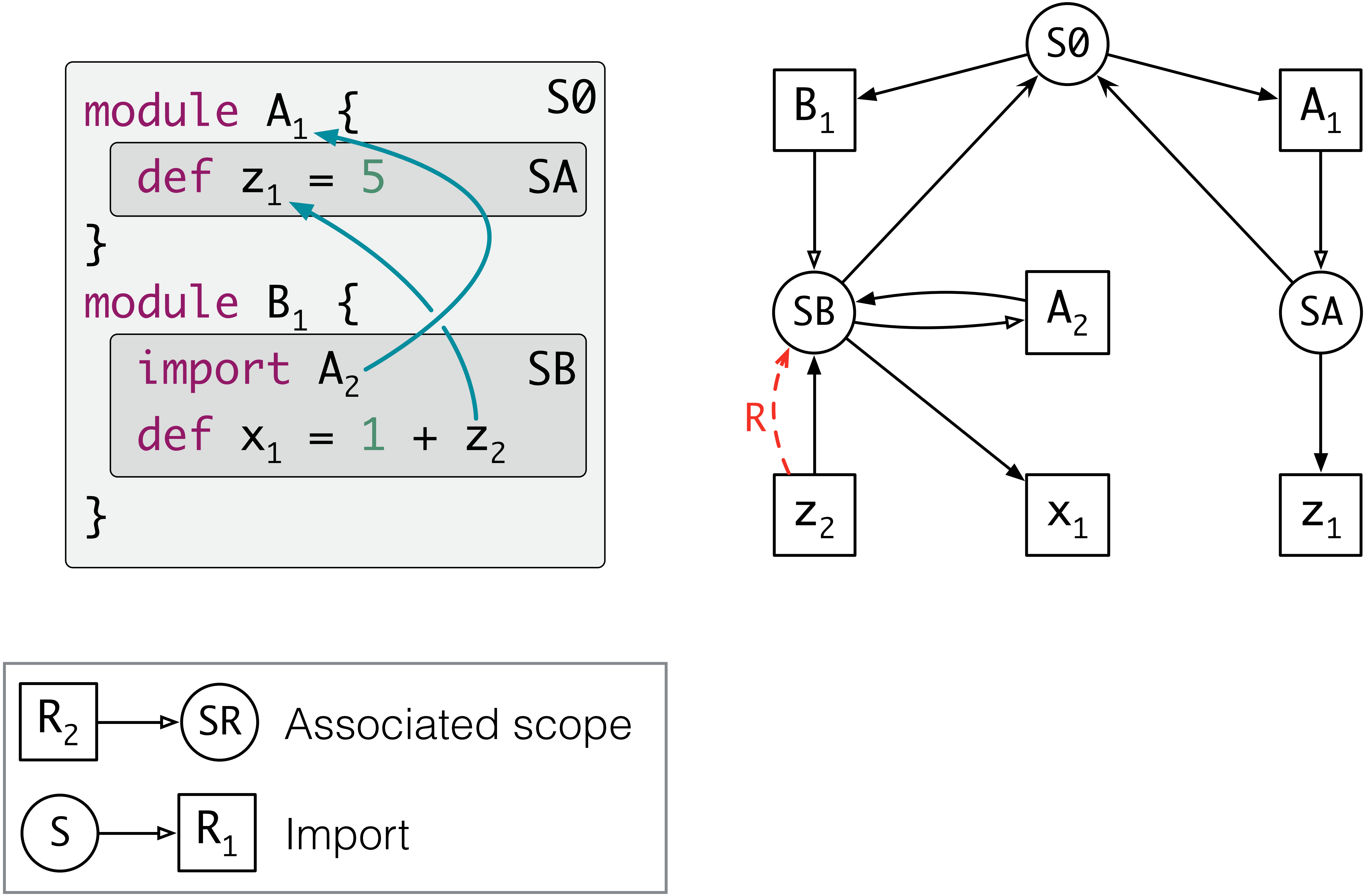
Imports


S0
module A1 {
def z1 = 5
SA
}
module B1 {
import A2
SB
def x1 = 1 + z2
}


S0
B1
A1
SB
A2
SA
z2
x1
z1

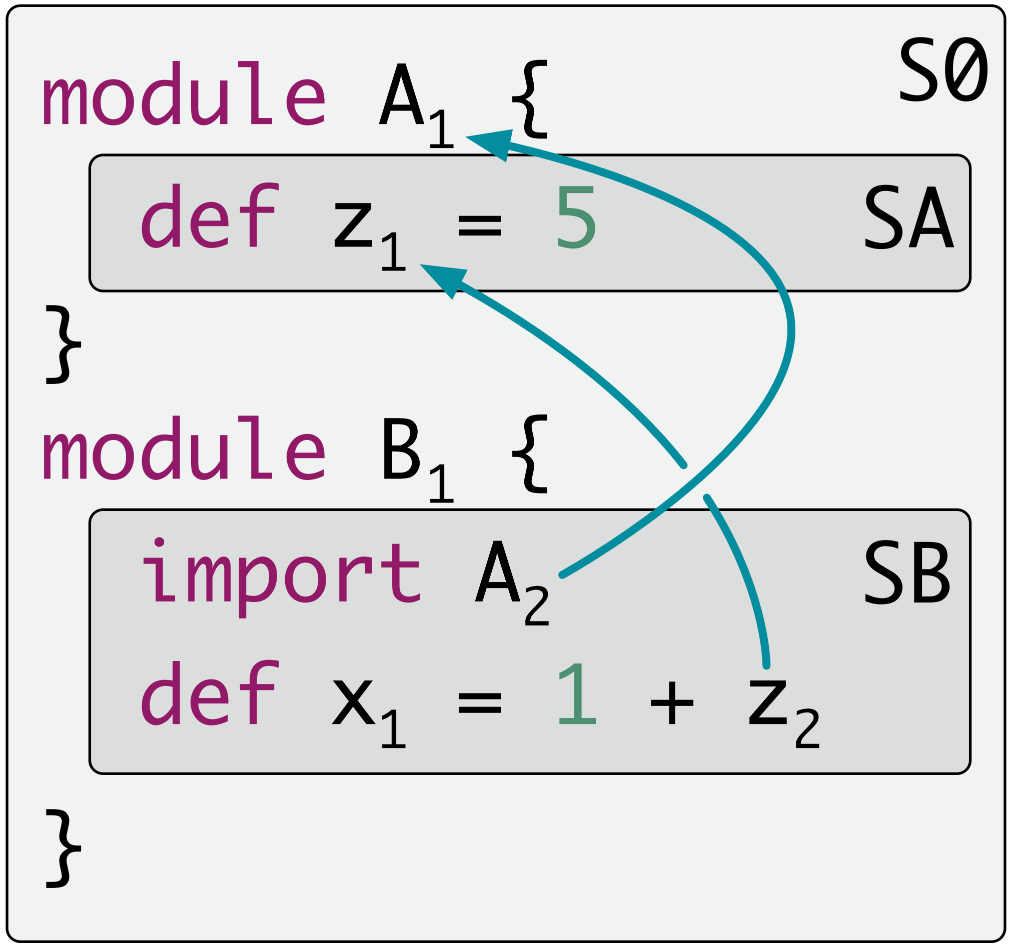

R2
SR
Associated scope
S
R1
Import

Imports

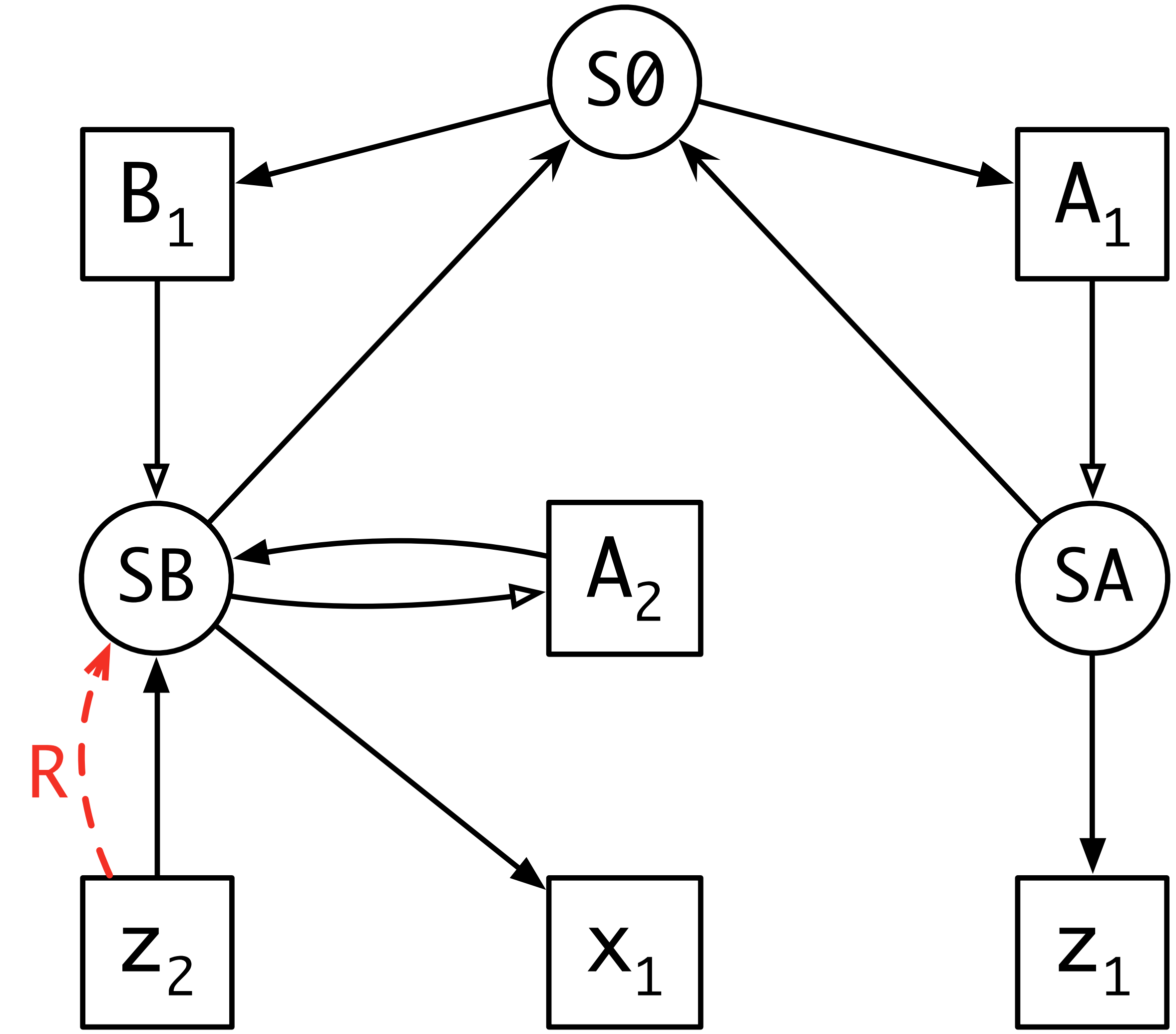


module A1 {
def z1 = 5
}
module B1 {
import A2
def x1 = 1 + z2
}
S0
SA
SB
S0
B1
A1
SB
A2
SA
R
z2
x1
z1
R2
SR
Associated scope
S
R1
Import

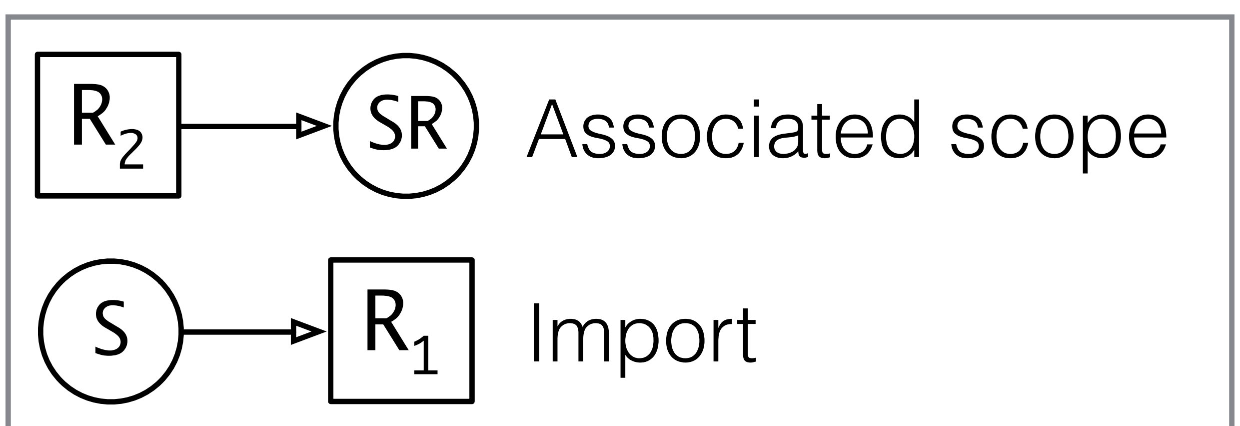
Imports



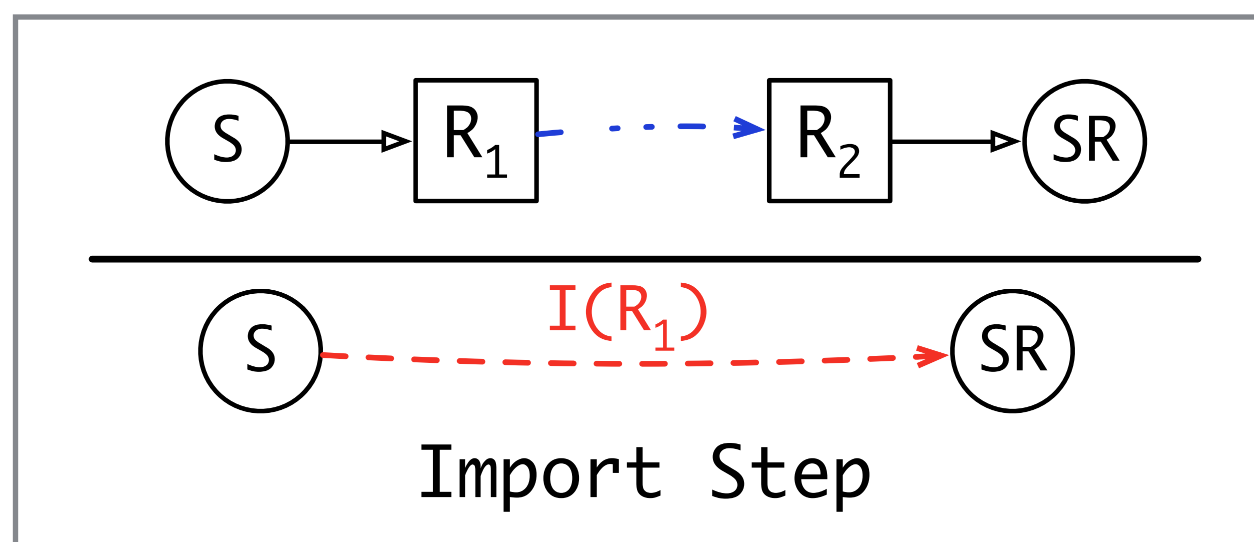
module A1 {
def z1 = 5
SA
}
module B1 {
import A2
def x1 = 1 + z2
SB
}
S0



S0
B1
A1
SB
A2
SA
R
z2
x1
z1

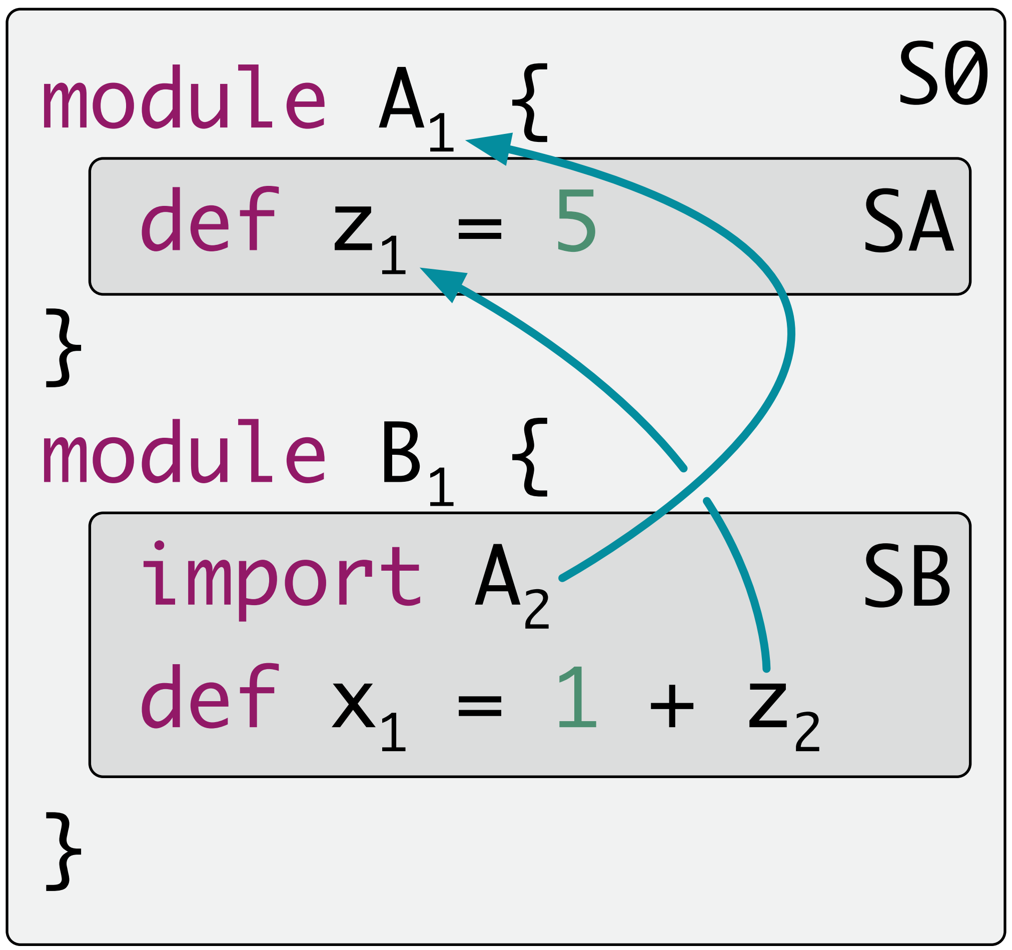


R2
SR
Associated scope
S
R1
Import

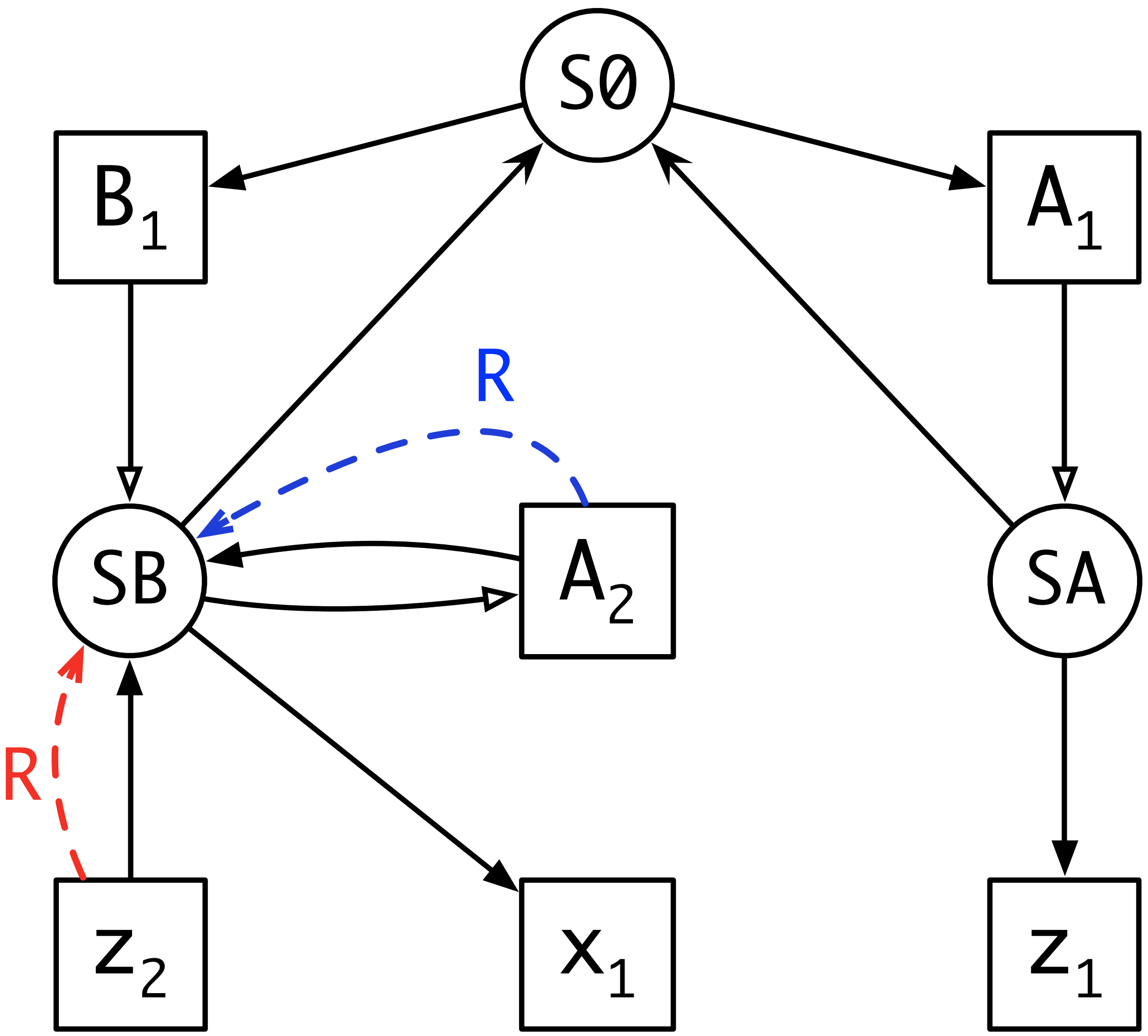


S
R1
R2
SR
I(R1)
S
SR
Import Step

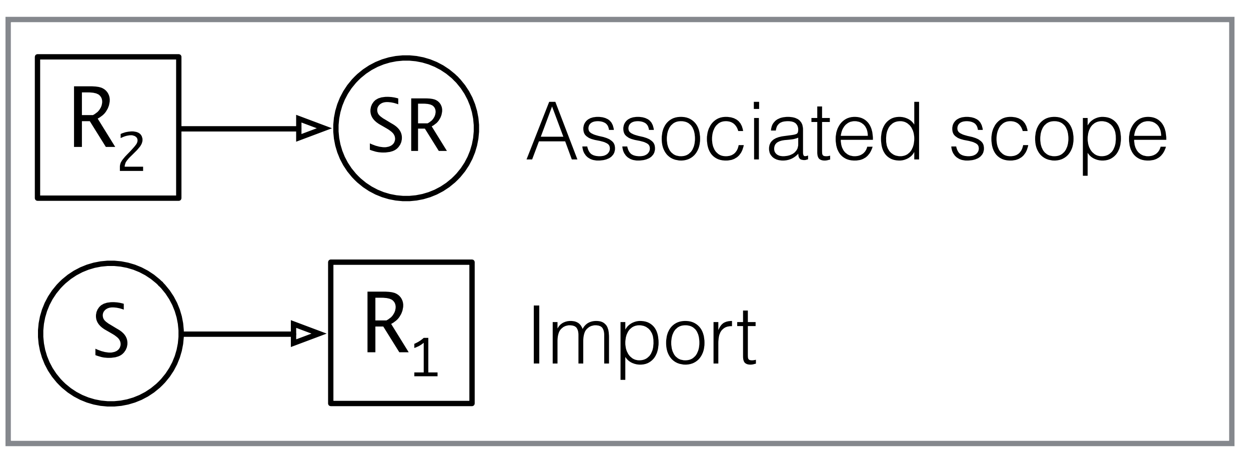
Imports



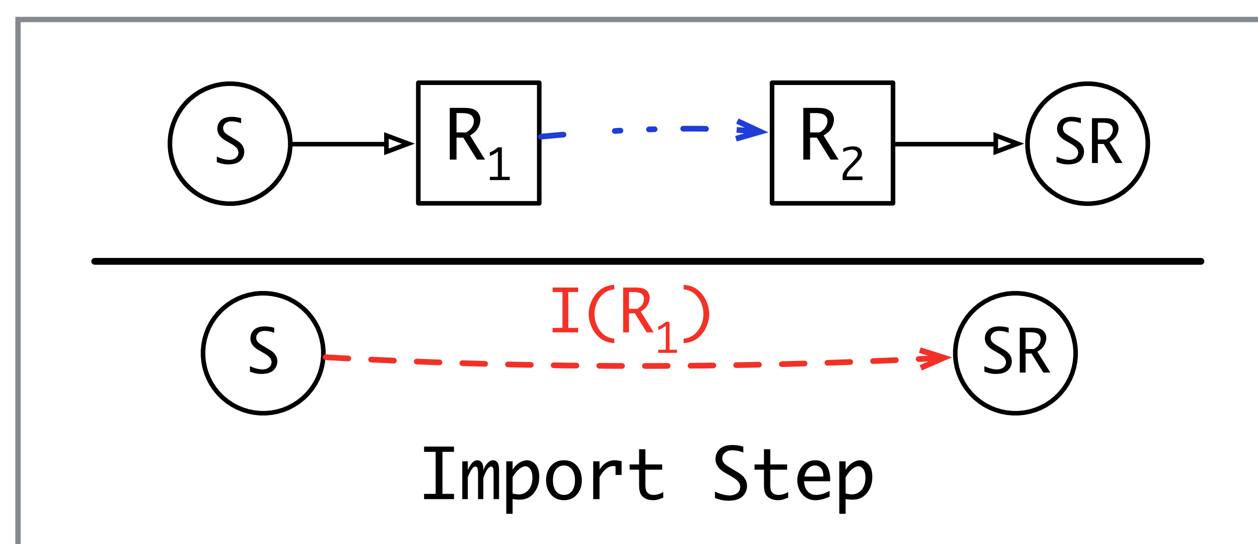
module A1 {
def z1 = 5
}
module B1 {
import A2
def x1 = 1 + z2
}
S0
SA
SB



S0
B1
A1
SB
A2
SA
z2
x1
z1
R
R

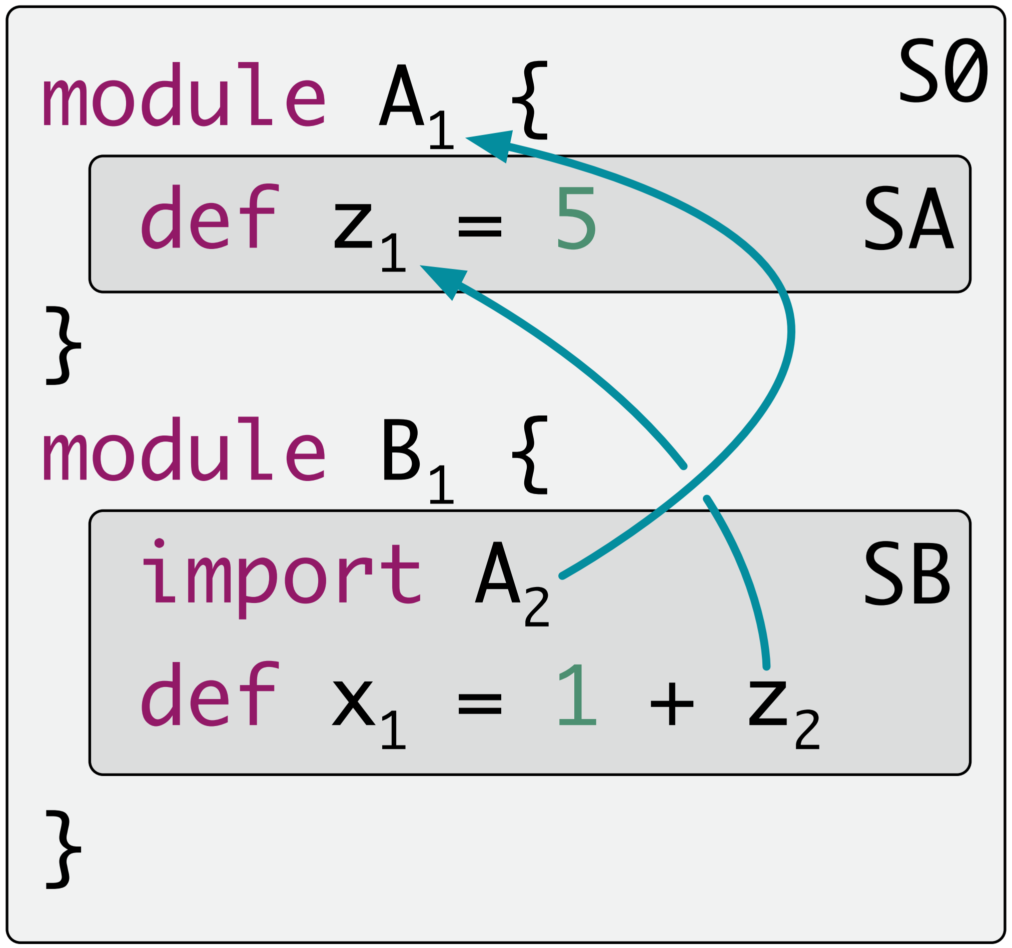


R2
SR
Associated scope
S
R1
Import

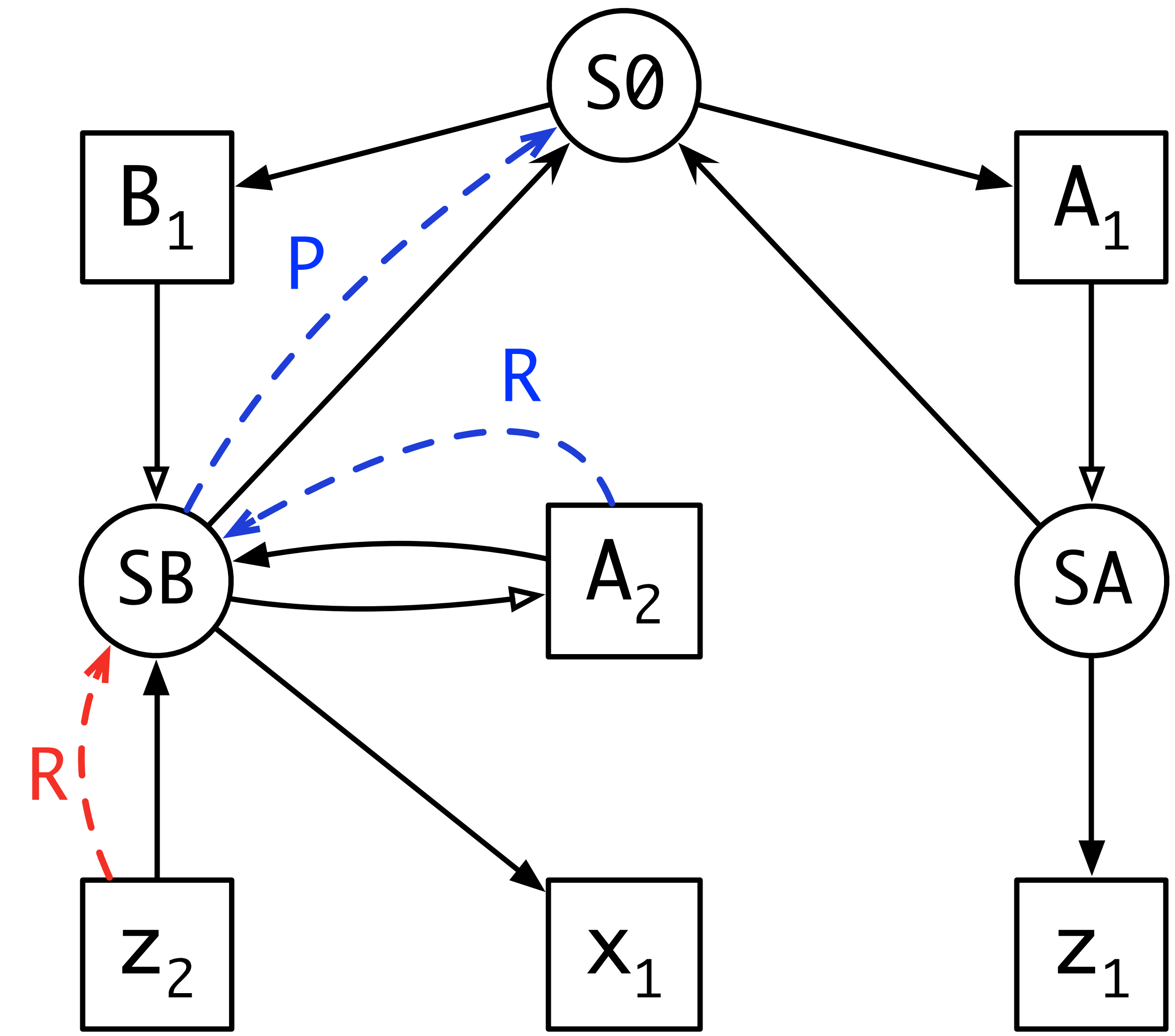


S
R1
R2
SR
I(R1)
S
SR
Import Step

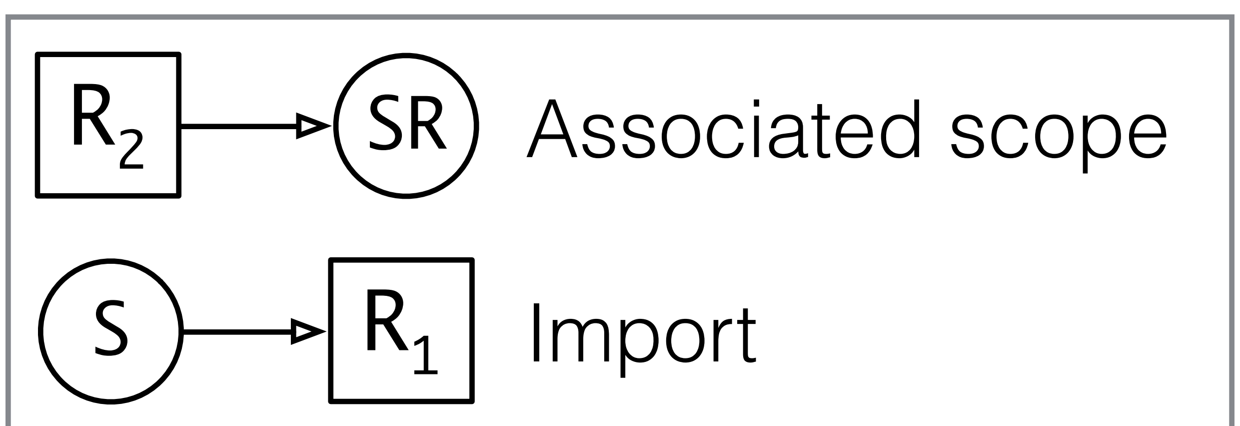
Imports



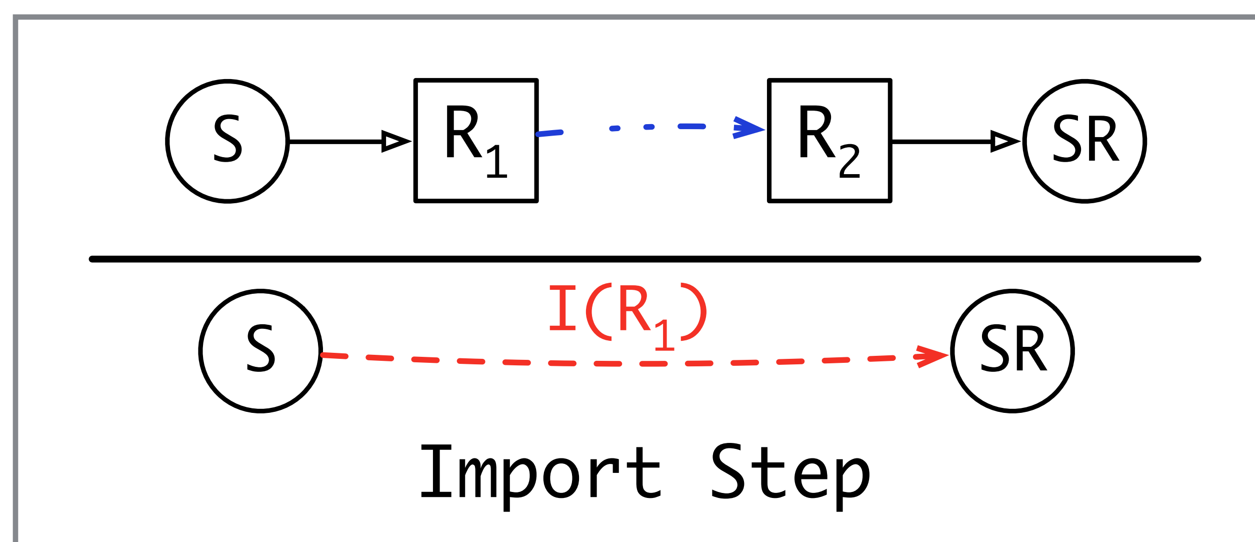
module A1 {
def z1 = 5
}
module B1 {
import A2
def x1 = 1 + z2
}
S0
SA
SB



S0
B1
A1
P
R
SB
A2
SA
R
z2
x1
z1

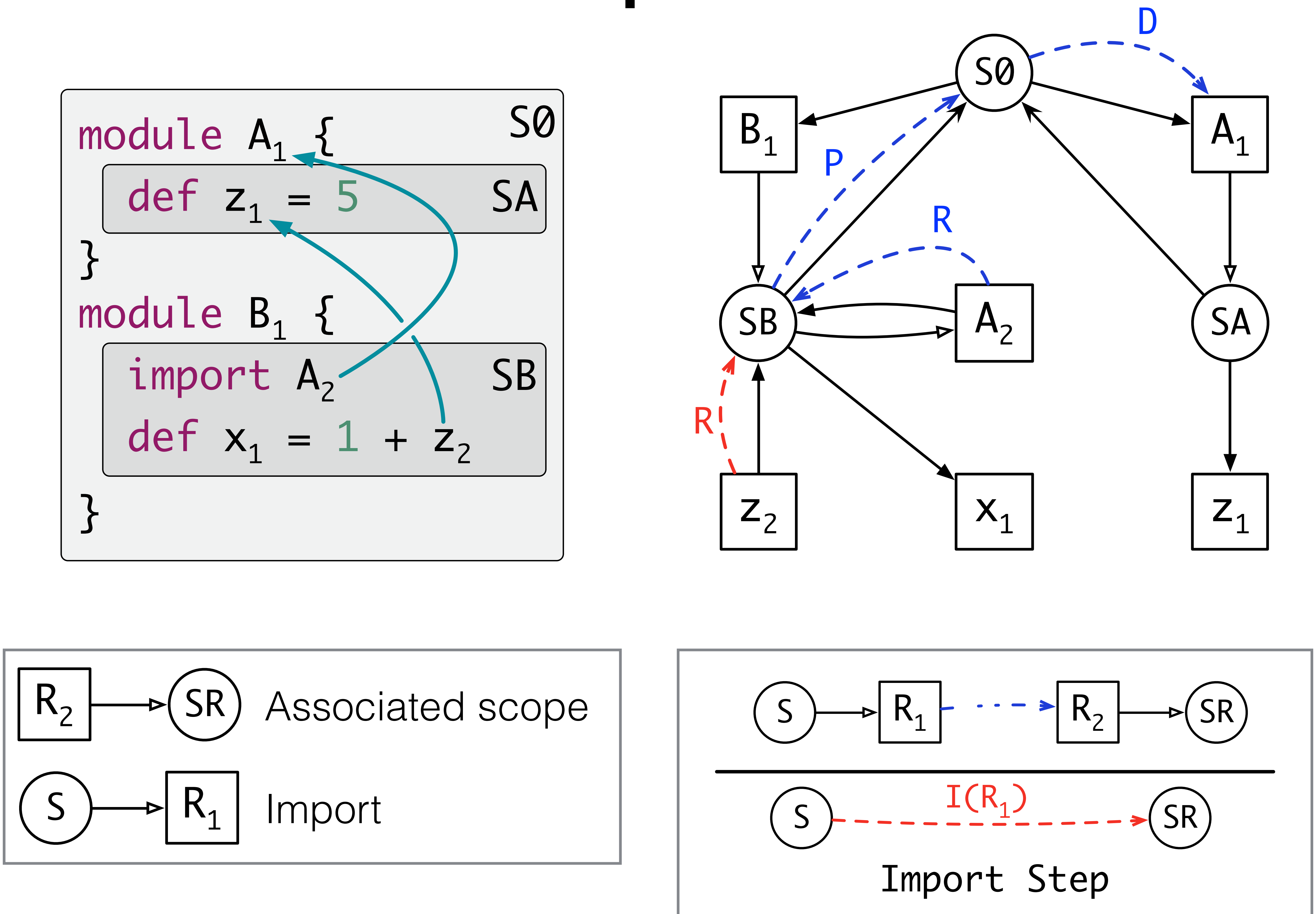


R2
SR
Associated scope
S
R1
Import



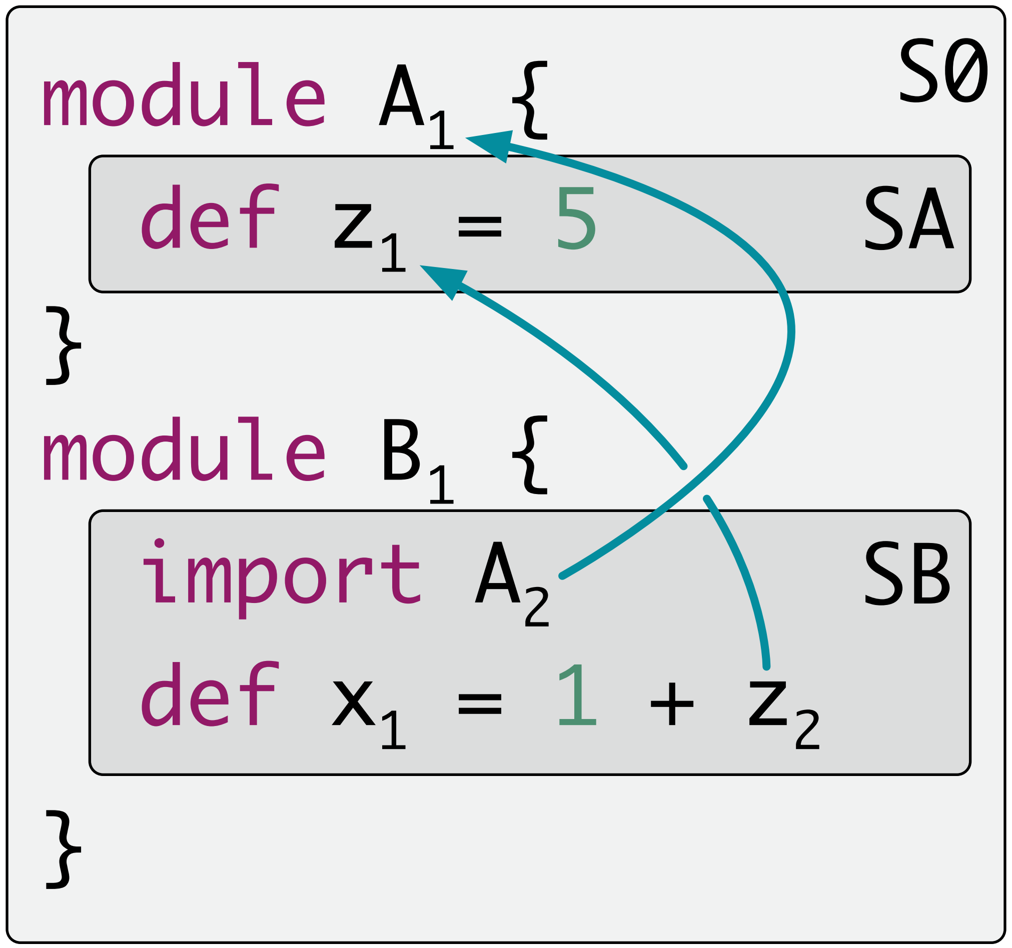
S
R1
R2
SR
I(R1)
S
SR
Import Step

Imports

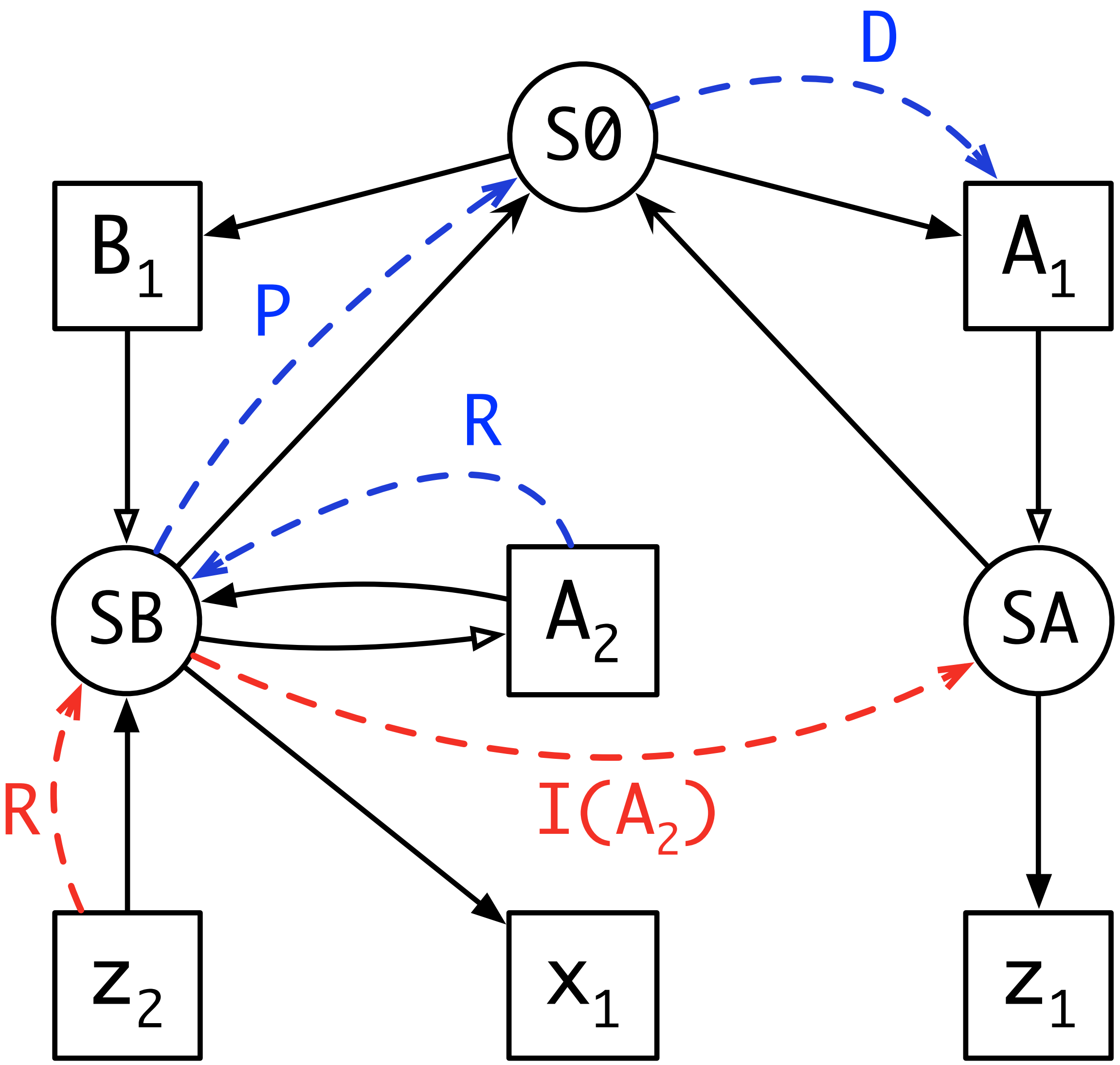


module A1 {
def z1 = 5
}
module B1 {
import A2
def x1 = 1 + z2
}
S0
SA
SB
D
P
R
R
B1
A1
A2
z2
x1
z1
Associated scope
Import
I(R1)
Import Step

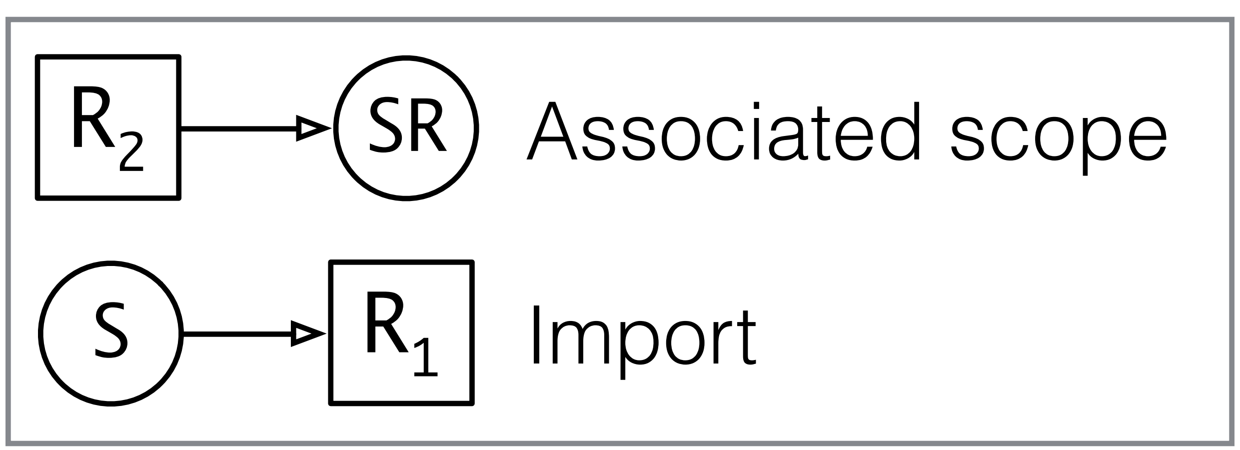
Imports



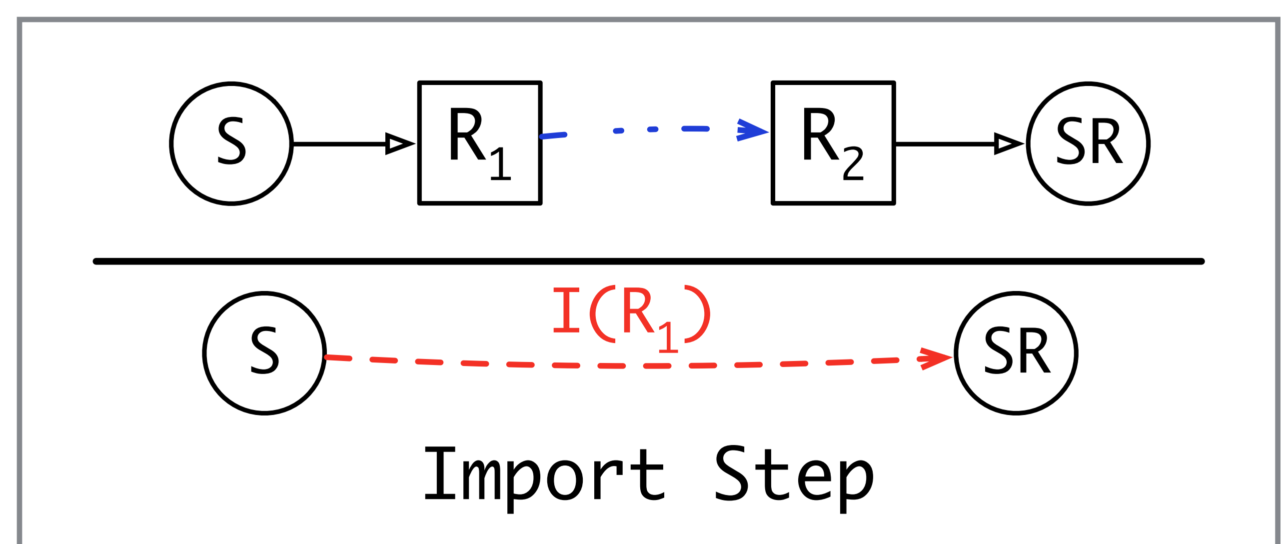
module A1 {
def z1 = 5
SA
}
S0
module B1 {
import A2
SB
def x1 = 1 + z2
}



S0
D
B1
A1
P
R
SB
A2
SA
R
I(A2)
z2
x1
z1

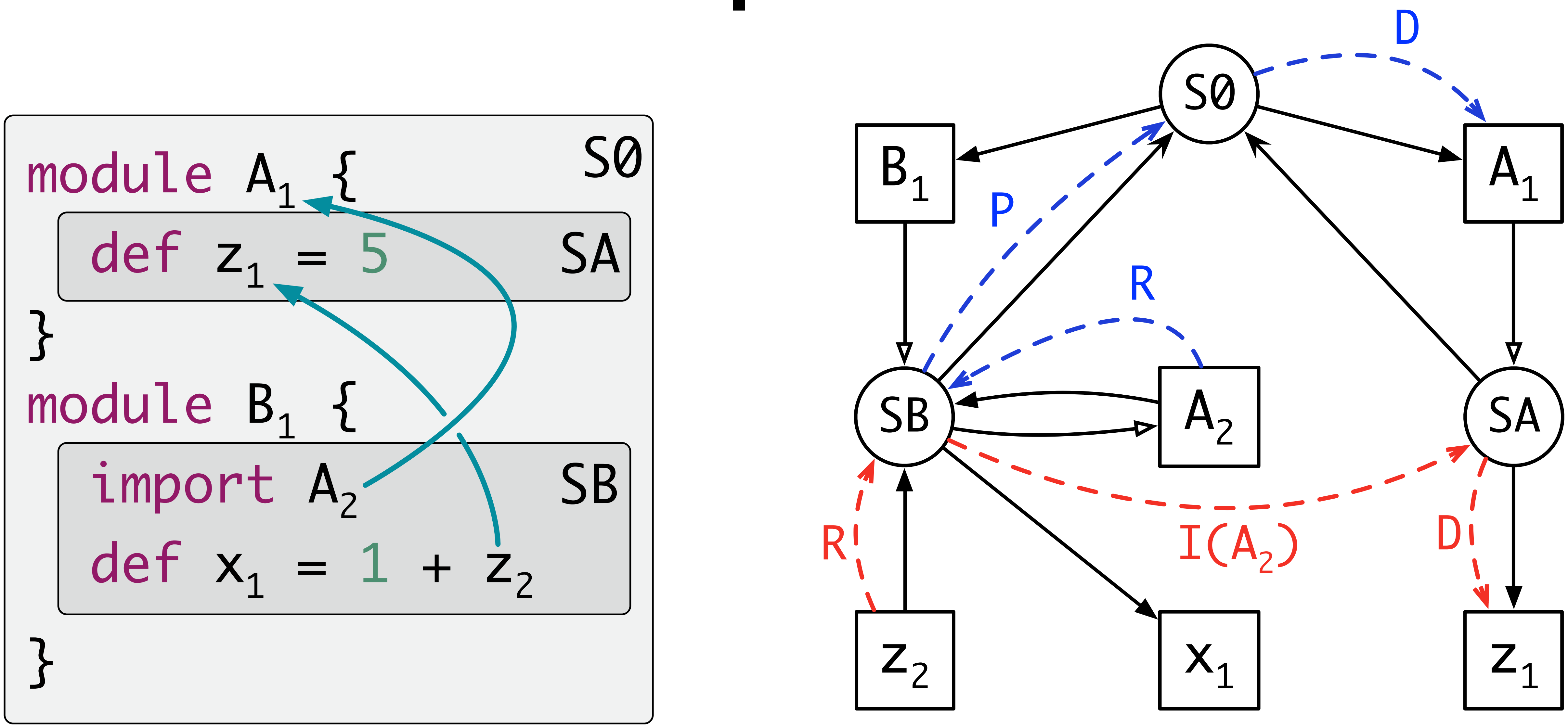


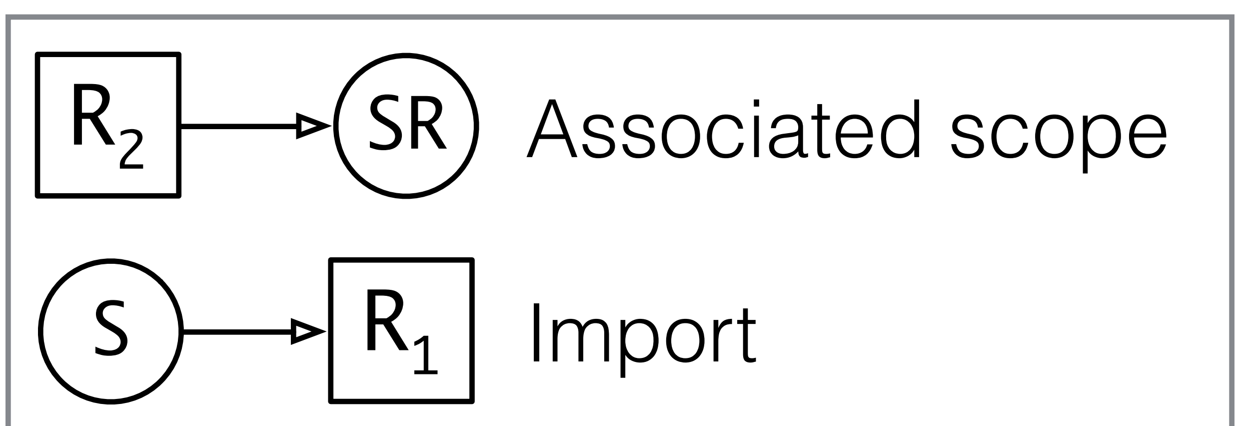
R2
SR
Associated scope
S
R1
Import

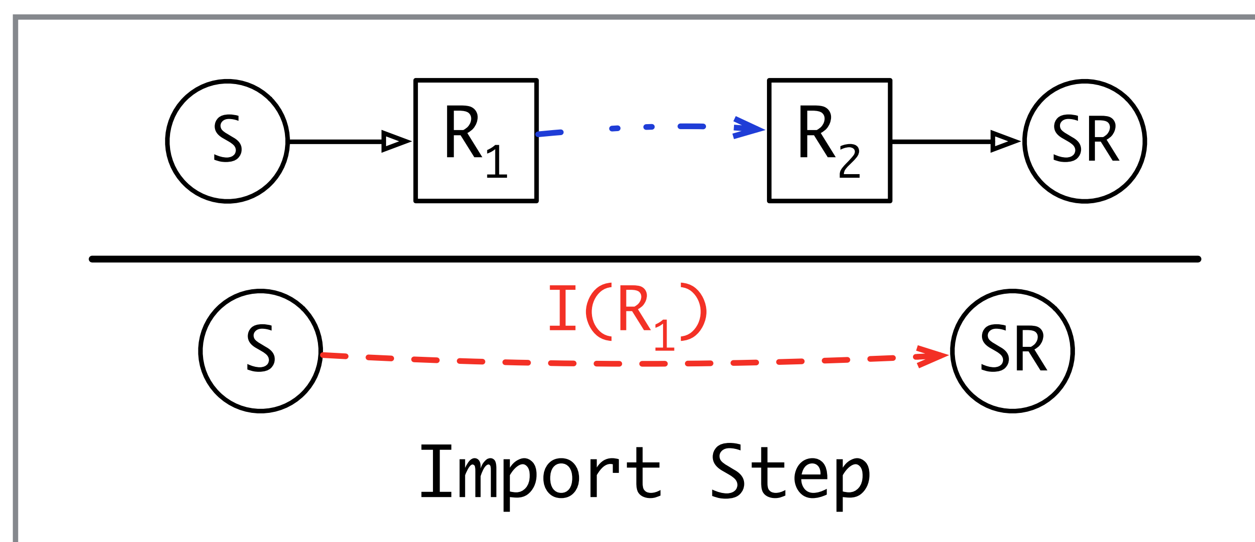


S
R1
R2
SR
S
I(R1)
SR
Import Step

Imports


S0
module A1 {
def z1 = 5
SA
}
module B1 {
import A2
SB
def x1 = 1 + z2
}
D
S0
B1
A1
P
R
SB
A2
SA
R
I(A2)
D
z2
x1
z1


R2
SR
Associated scope
S
R1
Import


S
R1
R2
SR
S
I(R1)
SR
Import Step

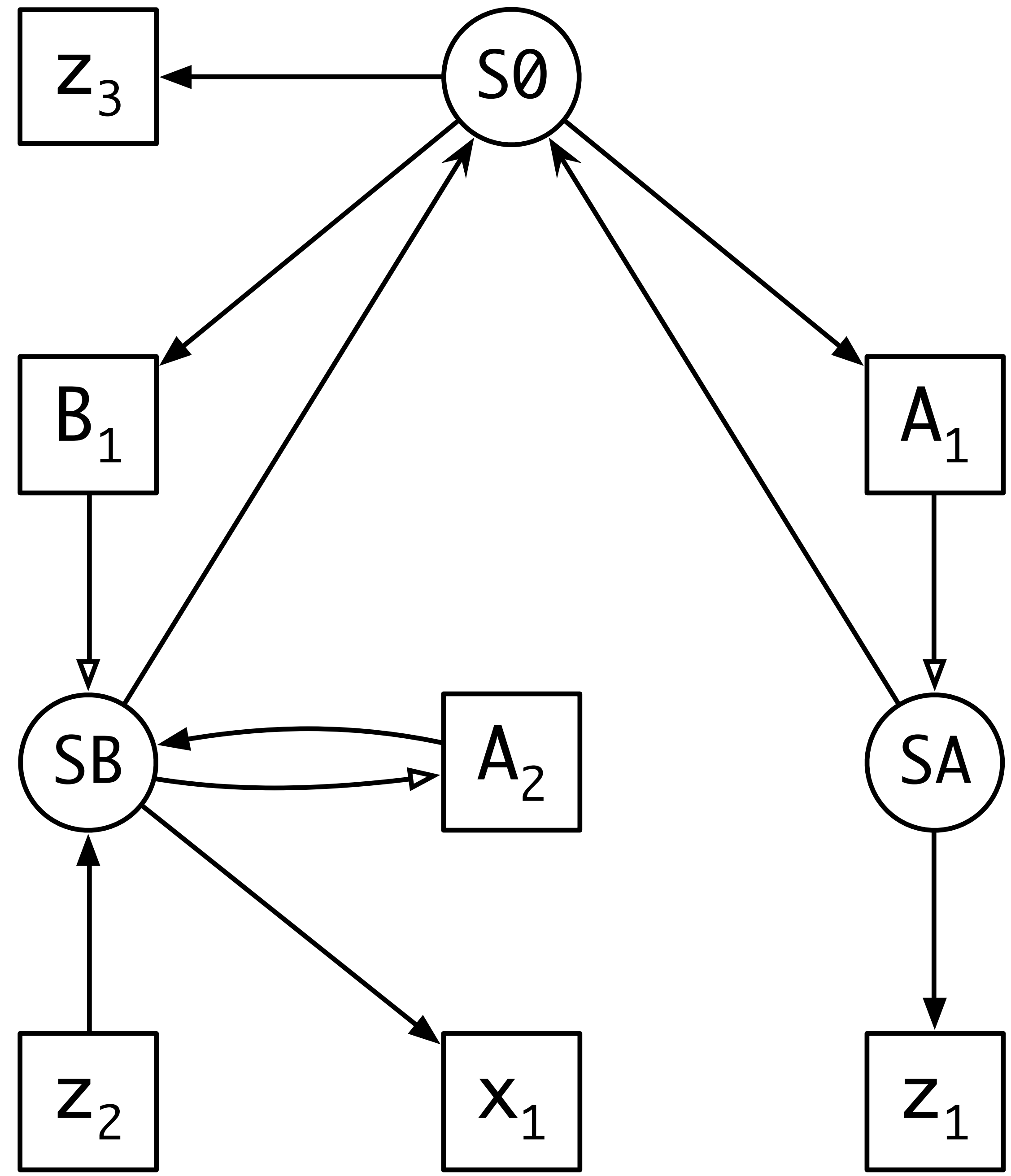
Imports over Parents

```
def z₃ = 2

module A₁ {
  def z₁ = 5
}
module B₁ {
  import A₂
  def x₁ = 1 + z₂
}
```

Imports over Parents

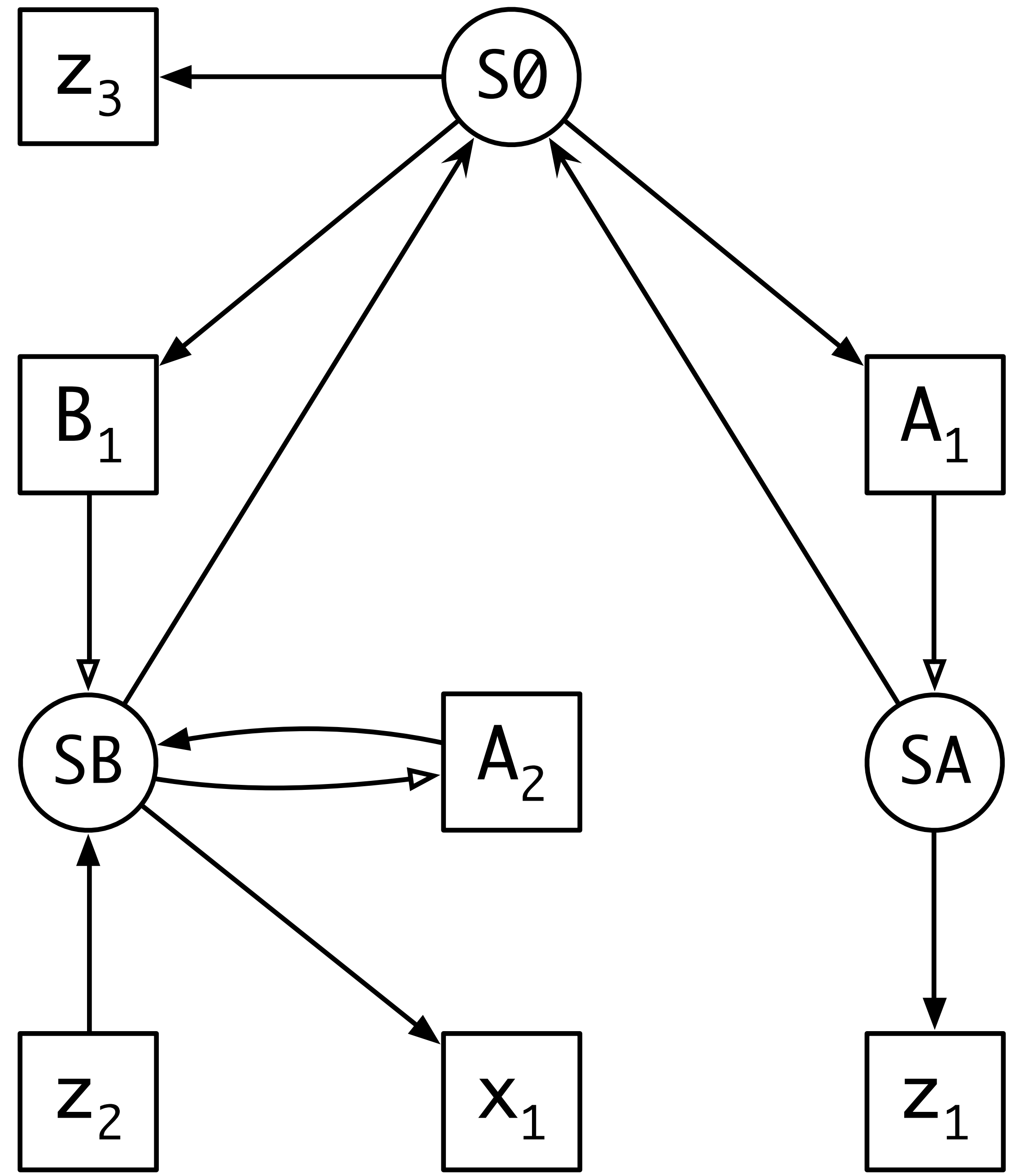
def z_3 = 2

module A_1 {
  def z_1 = 5
}
module B_1 {
  import A_2
  def x_1 = 1 + z_2
}



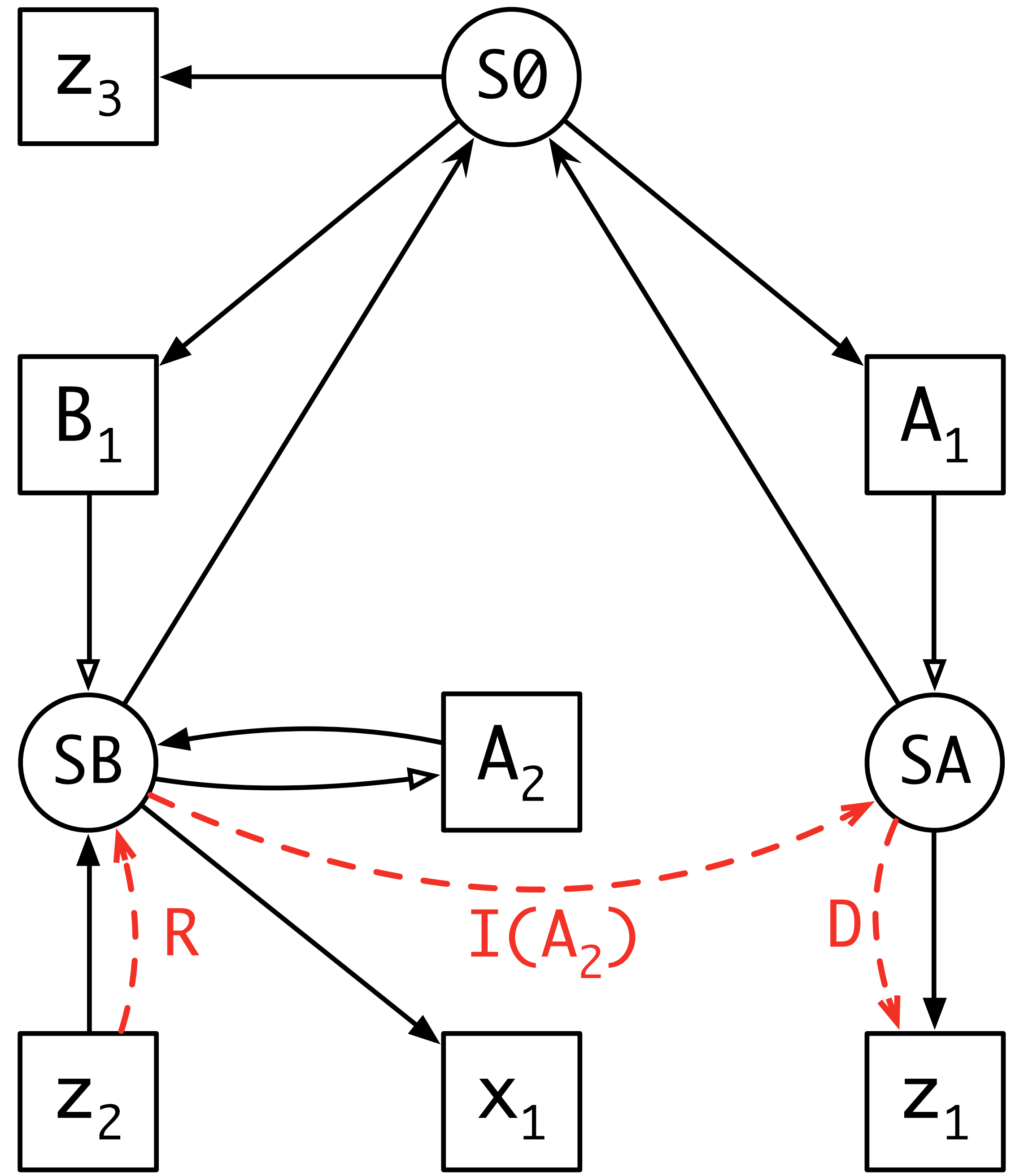
Imports over Parents

```
def z_3 = 2

module A_1 {
  def z_1 = 5
}
module B_1 {
  import A_2
  def x_1 = 1 + z_2
}
```



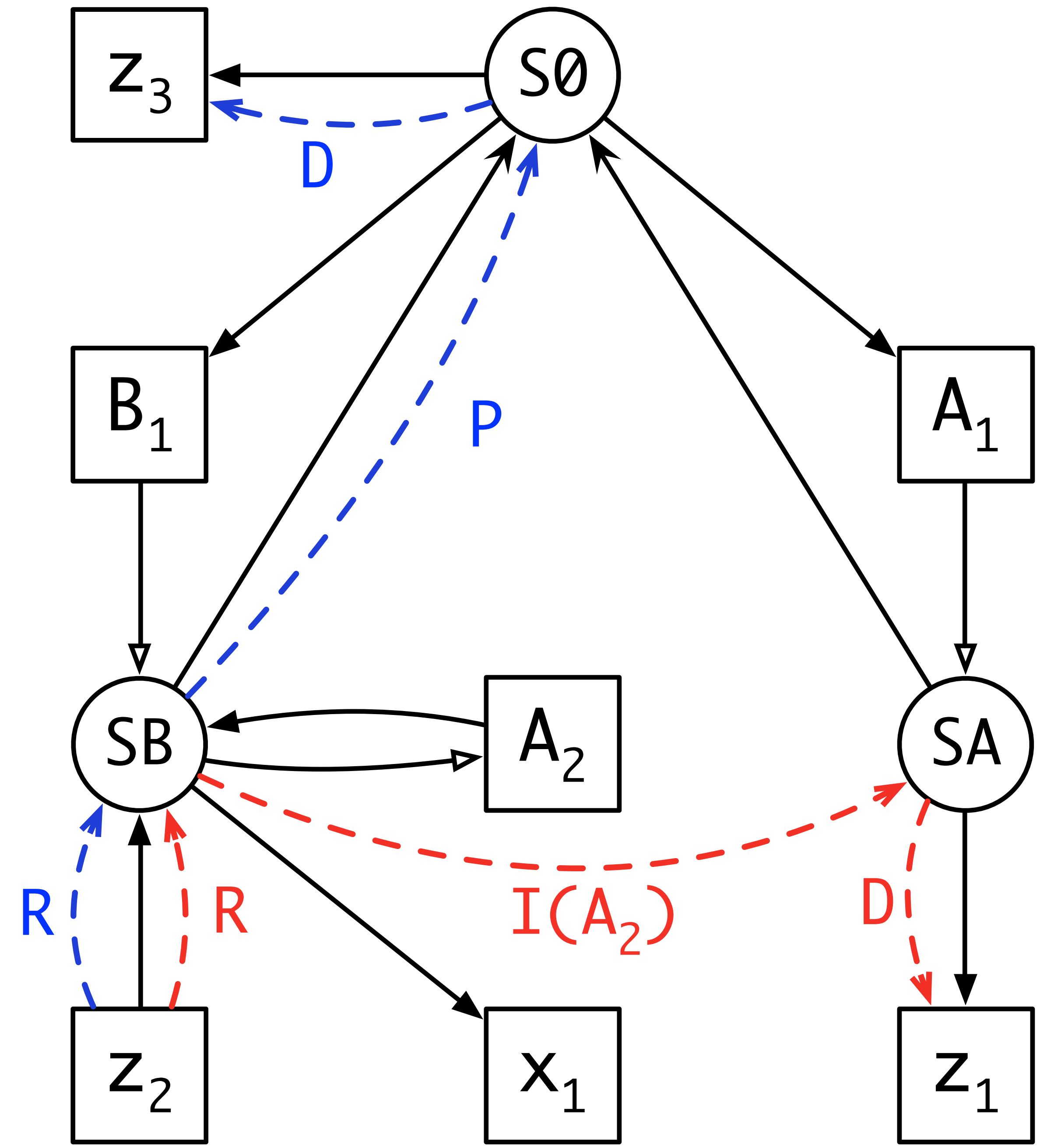
Imports over Parents

```
def z_3 = 2

module A_1 {
  def z_1 = 5
}
module B_1 {
  import A_2
  def x_1 = 1 + z_2
}
```



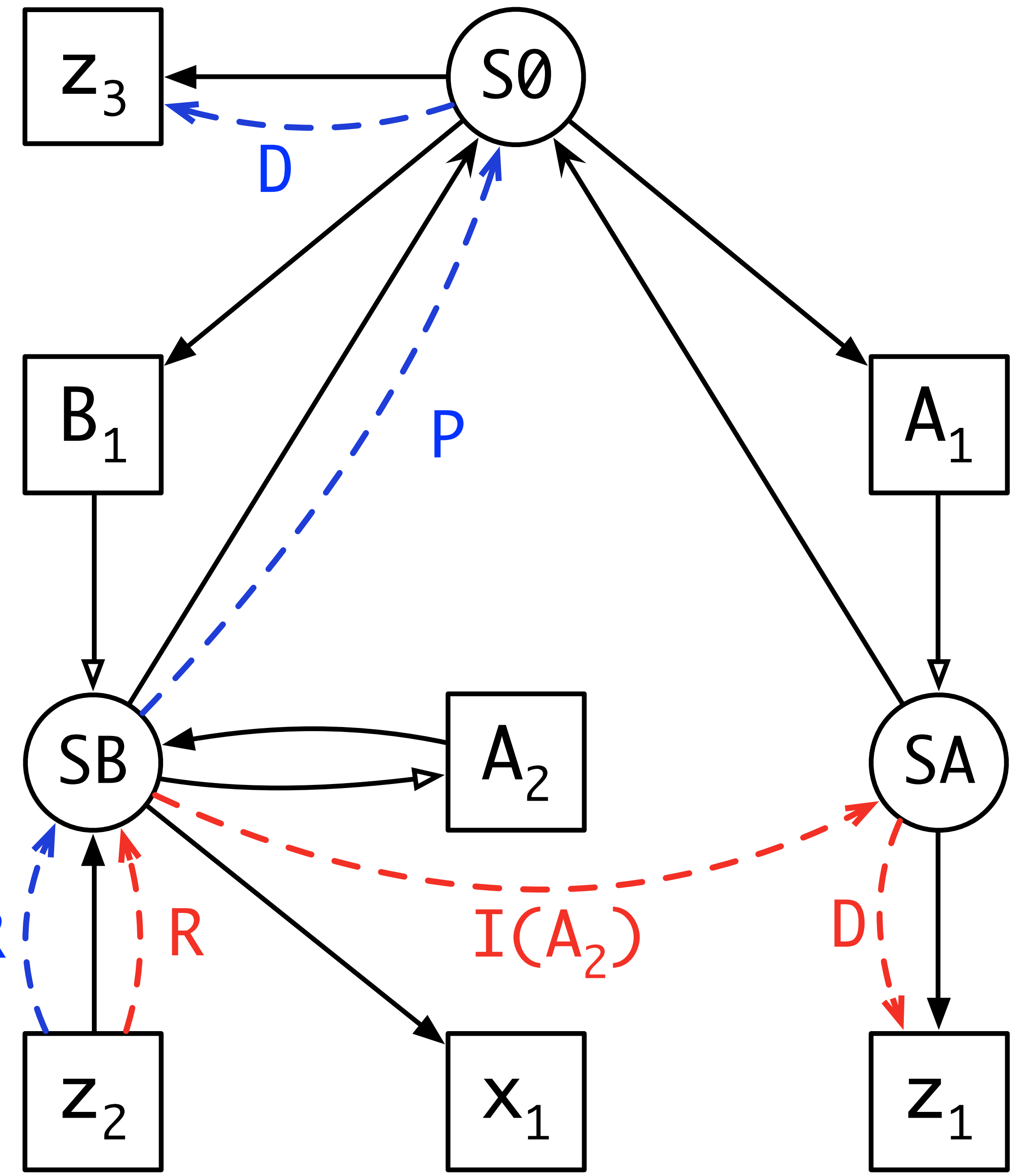
Imports over Parents

```
def z₃ = 2

module A₁ {
  def z₁ = 5
}
module B₁ {
  import A₂
  def x₁ = 1 + z₂
}
```



Imports over Parents

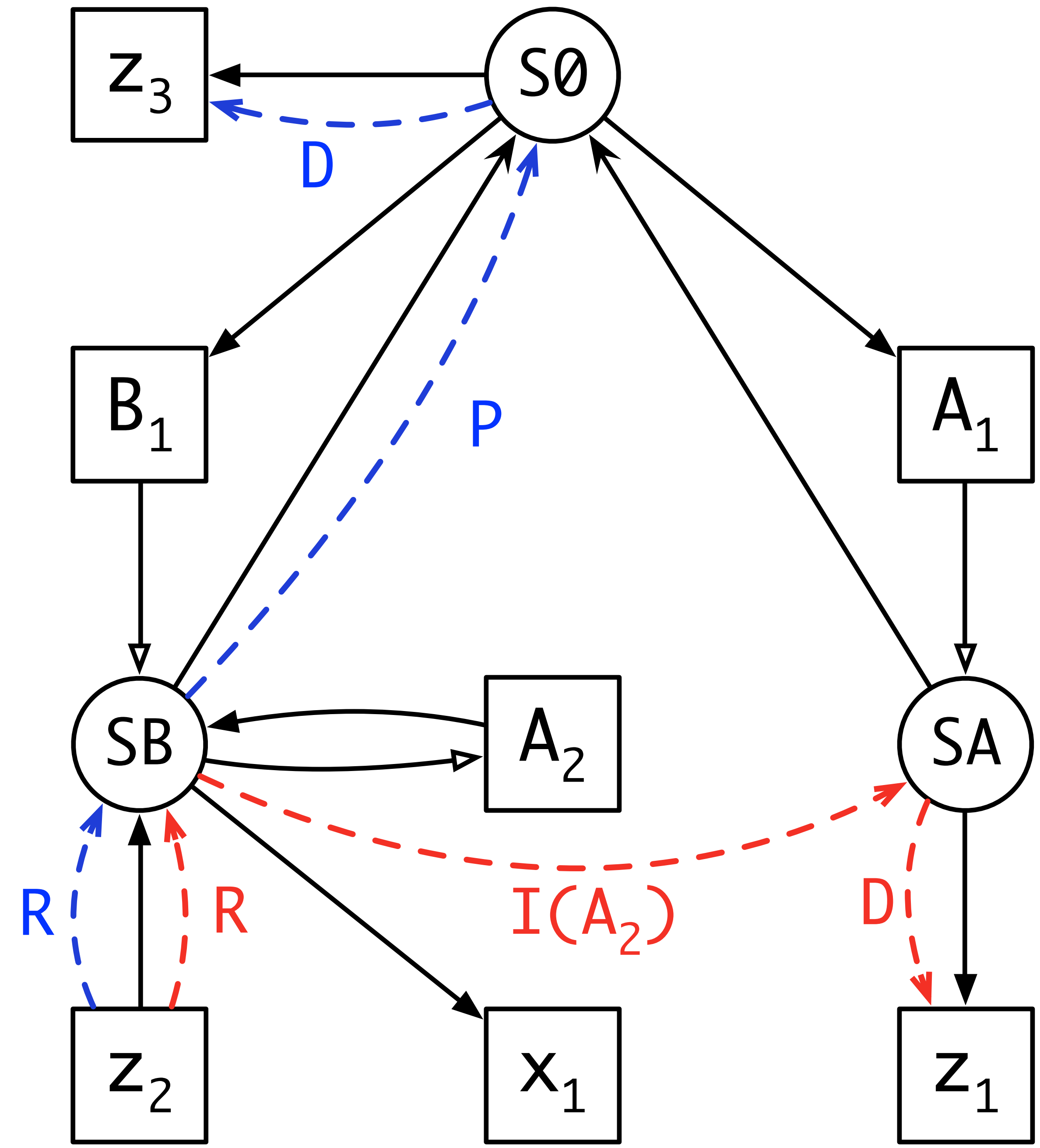
```
def z₃ = 2

module A₁ {
  def z₁ = 5
}
module B₁ {
  import A₂
  def x₁ = 1 + z₂
}
```



I(_).p' < P.p

Imports over Parents

```
def z3 = 2

module A1 {
  def z1 = 5
}
module B1 {
  import A2
  def x1 = 1 + z2
}
```

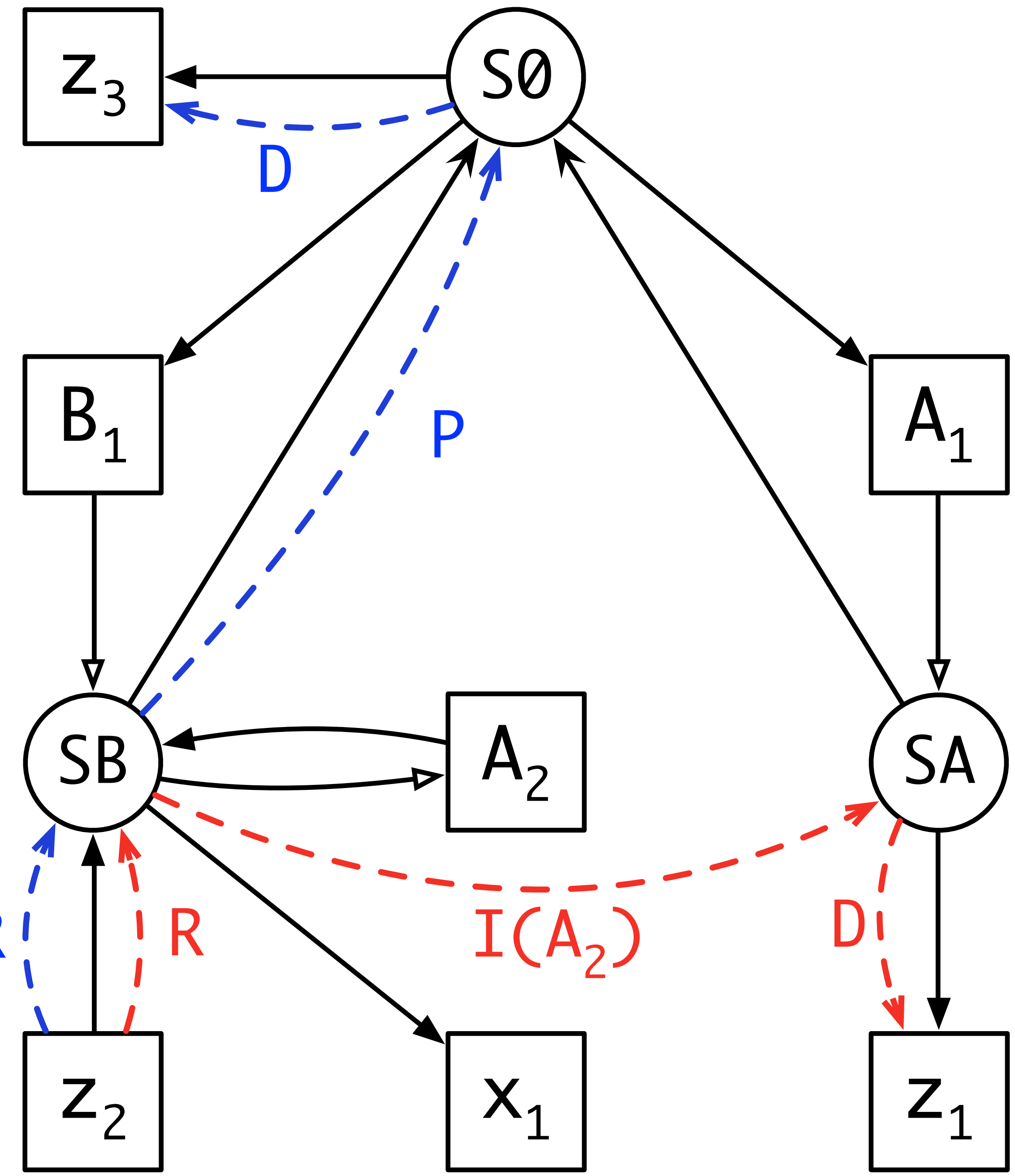


$$\frac{}{I(\_).p' < P.p}$$

$$\frac{}{D < I(\_).p'}$$

Imports over Parents

```
def z₃ = 2

module A₁ {
  def z₁ = 5
}
module B₁ {
  import A₂
  def x₁ = 1 + z₂
}
```



$$\frac{\dfrac{}{\textcolor{red}{I(\_).p'} < \textcolor{blue}{P.p}}}{\textcolor{red}{D} < \textcolor{blue}{I(\_).p'}}$$

$\Rightarrow$ $\textcolor{red}{R.I(A_2).D} < \textcolor{blue}{R.P.D}$

Transitive / Non-Transitive

```
module A₁ {
  def z₁ = 5
}
module B₁ {
  import A₂
}
module C₁ {
  import B₂
  def x₁ = 1 + z₂
}
```

Transitive / Non-Transitive

```
module A₁ {
  def z₁ = 5
}
module B₁ {
  import A₂
}
module C₁ {
  import B₂
  def x₁ = 1 + z₂
}
```

Transitive / Non-Transitive

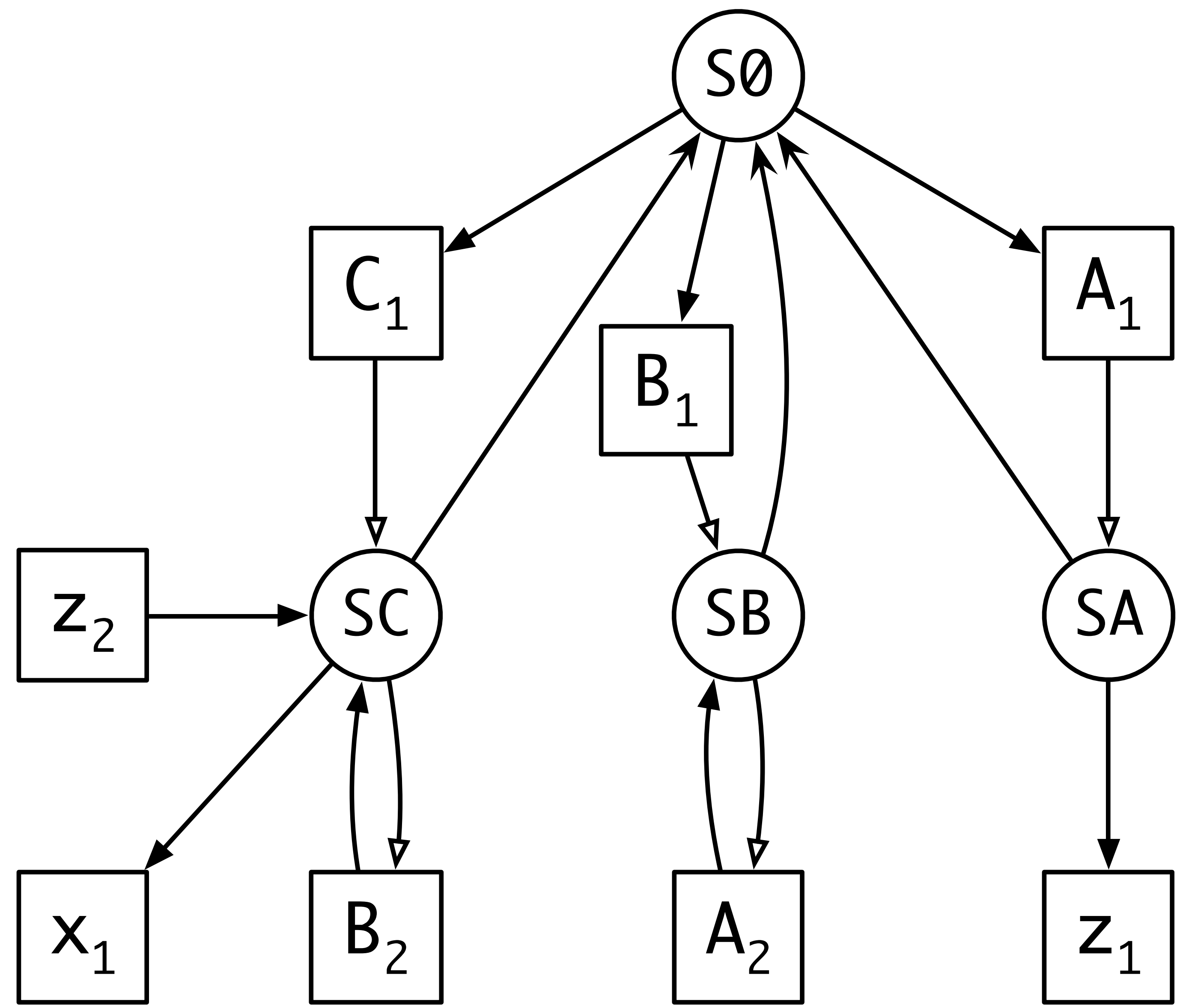
```
module A_1 {
  def z_1 = 5
}
module B_1 {
  import A_2
}
module C_1 {
  import B_2
  def x_1 = 1 + z_2
}
```

??

Transitive / Non-Transitive

module A_1 {
  def $z_1 = 5$
}
module B_1 {
  import A_2
}
module C_1 {
  import B_2
  def $x_1 = 1 + z_2$
}

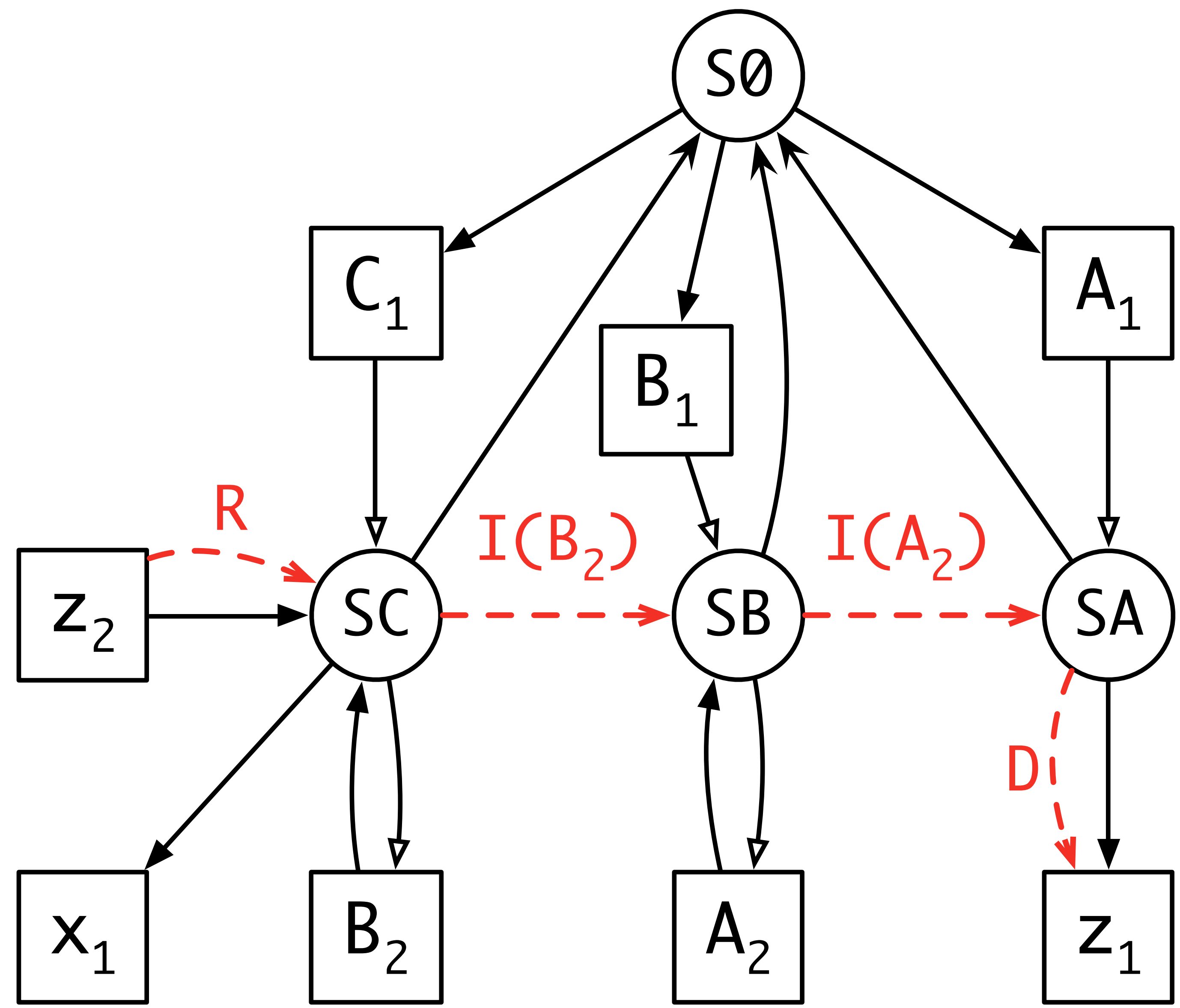
??



Transitive / Non-Transitive

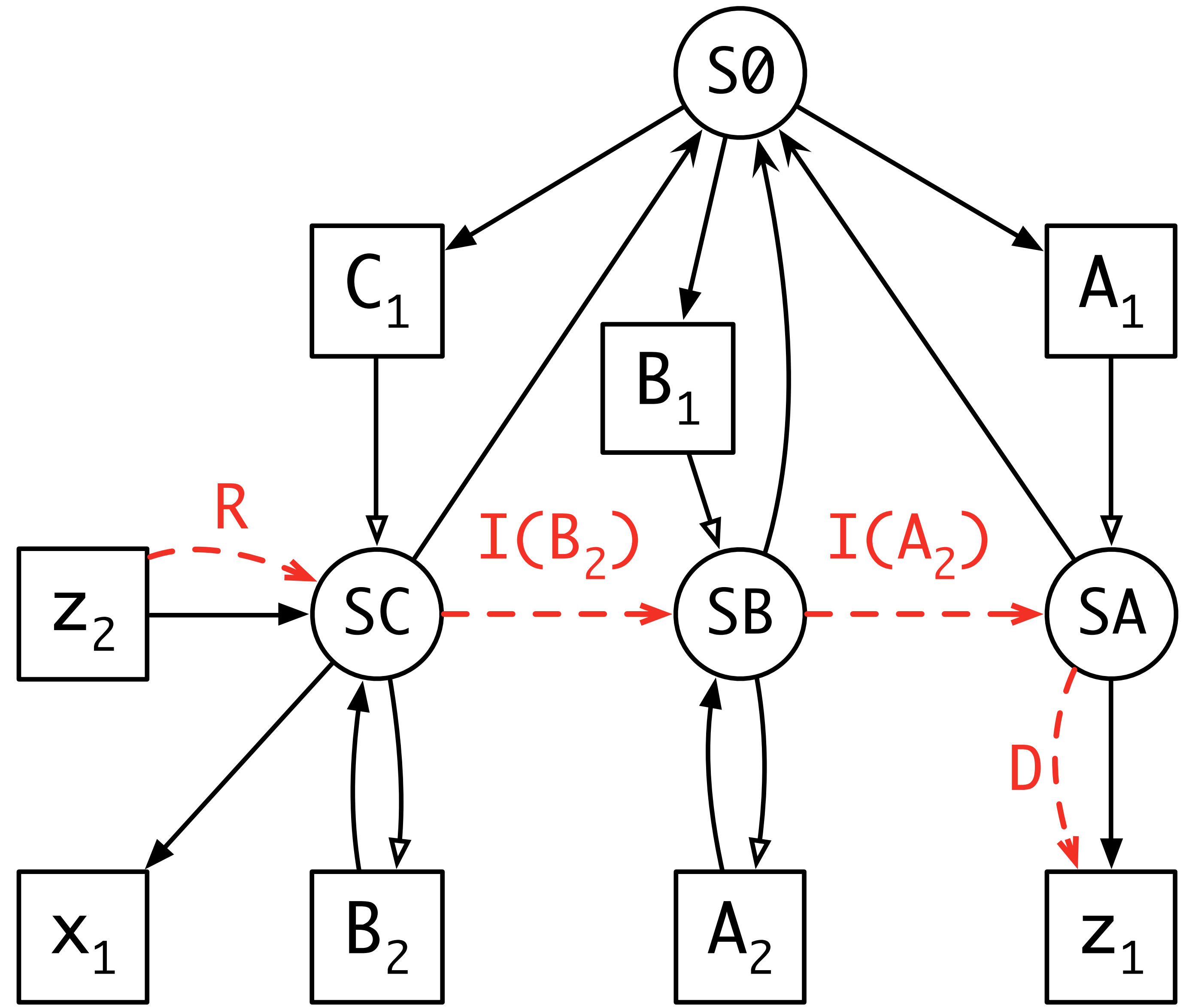
```
module A_1 {
  def z_1 = 5
}
module B_1 {
  import A_2
}
module C_1 {
  import B_2
  def x_1 = 1 + z_2
}
```

??



Transitive / Non-Transitive

```
module A_1 {
  def z_1 = 5
}                ??
module B_1 {
  import A_2
}
module C_1 {
  import B_2
  def x_1 = 1 + z_2
}
```

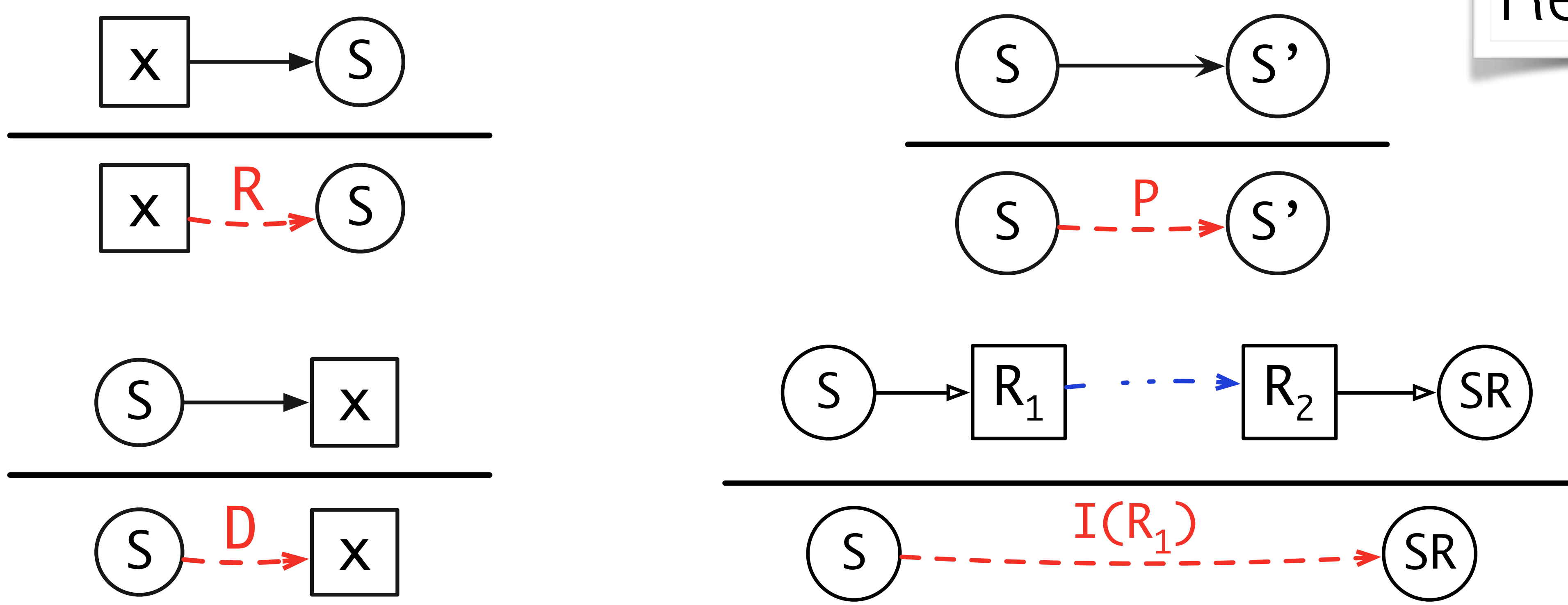


With transitive imports, a well formed path is $\texttt{R.P*.I(\_)*.D}$

With non-transitive imports, a well formed path is $\texttt{R.P*.I(\_)}^{?}\texttt{.D}$

A Calculus for Name Resolution

Reachability



Disambiguation

$$\frac{}{D < P.p} \qquad \frac{}{I(\_).p' < P.p}$$

$$\frac{p < p'}{s.p < s.p'} \qquad \frac{}{D < I(\_).p'}$$

Well formed path: `R.P*.I(_)*.D`

In the Paper

- Formalization of the resolution calculus

Edges in scope graph

$$\frac{\mathcal{P}(S_1) = S_2}{\mathbb{I} \vdash \mathbf{P} : S_1 \longrightarrow S_2} \quad (P)$$

$$\frac{y_i^{\mathsf{R}} \in \mathcal{I}(S_1) \setminus \mathbb{I} \quad \mathbb{I} \vdash p : y_i^{\mathsf{R}} \longmapsto y_j^{\mathsf{D}}{:}S_2}{\mathbb{I} \vdash \mathbf{I}(y_i^{\mathsf{R}}, y_j^{\mathsf{D}}{:}S_2) : S_1 \longrightarrow S_2} \quad (I)$$

Transitive closure

$$\frac{}{\mathbb{I} \vdash [] : A \twoheadrightarrow A} \quad (N)$$

$$\frac{\mathbb{I} \vdash s : A \longrightarrow B \quad \mathbb{I} \vdash p : B \twoheadrightarrow C}{\mathbb{I} \vdash s \cdot p : A \twoheadrightarrow C} \quad (T)$$

Reachable declarations

$$\frac{x_i^{\mathsf{D}} \in \mathcal{D}(S') \quad \mathbb{I} \vdash p : S \twoheadrightarrow S' \quad WF(p)}{\mathbb{I} \vdash p \cdot \mathbf{D}(x_i^{\mathsf{D}}) : S \rightarrowtail x_i^{\mathsf{D}}} \quad (R)$$

Visible declarations

$$\frac{\mathbb{I} \vdash p : S \rightarrowtail x_i^{\mathsf{D}} \qquad \forall j, p'(\mathbb{I} \vdash p' : S \rightarrowtail x_j^{\mathsf{D}} \Rightarrow \neg(p' < p))}{\mathbb{I} \vdash p : S \longmapsto x_i^{\mathsf{D}}} \quad (V)$$

Reference resolution

$$\frac{x_i^{\mathsf{R}} \in \mathcal{R}(S) \quad \{x_i^{\mathsf{R}}\} \cup \mathbb{I} \vdash p : S \longmapsto x_j^{\mathsf{D}}}{\mathbb{I} \vdash p : x_i^{\mathsf{R}} \longmapsto x_j^{\mathsf{D}}} \quad (X)$$

Well-formed paths

$$WF(p) \Leftrightarrow p \in \mathbf{P}^* \cdot \mathbf{I}(\_,\_)^*$$

Specificity ordering on paths

$$\frac{}{\mathbf{D}(\_) < \mathbf{I}(\_,\_)} \quad (DI) \qquad \frac{}{\mathbf{I}(\_,\_) < \mathbf{P}} \quad (IP) \qquad \frac{}{\mathbf{D}(\_) < \mathbf{P}} \quad (DP)$$

$$\frac{s_1 < s_2}{s_1 \cdot p_1 < s_2 \cdot p_2} \quad (Lex1) \qquad \frac{p_1 < p_2}{s \cdot p_1 < s \cdot p_2} \quad (Lex2)$$

In the Paper

- Formalization of the resolution calculus
- Validation by an extensive set of examples

In the Paper

- Formalization of the resolution calculus
- Validation by an extensive set of examples
    - definition before use

In the Paper

- Formalization of the resolution calculus

- Validation by an extensive set of examples

    - definition before use
    - different let binding flavors

In the Paper

- Formalization of the resolution calculus
- Validation by an extensive set of examples
  - definition before use
  - different let binding flavors
  - modules

In the Paper

- Formalization of the resolution calculus
- Validation by an extensive set of examples
    - definition before use
    - different let binding flavors
    - modules
    - imports

In the Paper

- Formalization of the resolution calculus

- Validation by an extensive set of examples

    - definition before use
    - different let binding flavors
    - modules
    - imports
    - qualified names

In the Paper

- Formalization of the resolution calculus
- Validation by an extensive set of examples
    - definition before use
    - different let binding flavors
    - modules
    - imports
    - qualified names
    - inheritance

In the Paper

- Formalization of the resolution calculus
- Validation by an extensive set of examples
  - definition before use
  - different let binding flavors
  - modules
  - imports
  - qualified names
  - inheritance
  - partial classes

In the Paper

- Formalization of the resolution calculus
- Validation by an extensive set of examples
  - definition before use
  - different let binding flavors
  - modules
  - imports
  - qualified names
  - inheritance
  - partial classes
  - …

In the Paper

- Formalization of the resolution calculus
- Validation by an extensive set of examples
- Sound and complete resolution algorithm

$$
\begin{array}{ll}
Res[\mathbb{I}](x_i^{\mathsf{R}}) & := \{x_j^{\mathsf{D}} \mid \exists S\ s.t.\ x_i^{\mathsf{R}} \in \mathcal{R}(S) \wedge x_j^{\mathsf{D}} \in Env_V[\{x_i^{\mathsf{R}}\} \cup \mathbb{I}, \emptyset](S)\} \\
Env_V[\mathbb{I}, \mathbb{S}](S) & := Env_L[\mathbb{I}, \mathbb{S}](S) \triangleleft Env_P[\mathbb{I}, \mathbb{S}](S) \\
Env_L[\mathbb{I}, \mathbb{S}](S) & := Env_D[\mathbb{I}, \mathbb{S}](S) \triangleleft Env_I[\mathbb{I}, \mathbb{S}](S) \\
Env_D[\mathbb{I}, \mathbb{S}](S) & := \begin{cases} \emptyset \text{ if } S \in \mathbb{S} \\ \mathcal{D}(S) \end{cases} \\
Env_I[\mathbb{I}, \mathbb{S}](S) & := \begin{cases} \emptyset \text{ if } S \in \mathbb{S} \\ \bigcup \{ Env_L[\mathbb{I}, \{S\} \cup \mathbb{S}](S_y) \mid y_i^{\mathsf{R}} \in \mathcal{I}(S) \setminus \mathbb{I} \wedge y_j^{\mathsf{D}}{:}S_y \in Res[\mathbb{I}](y_i^{\mathsf{R}}) \} \end{cases} \\
Env_P[\mathbb{I}, \mathbb{S}](S) & := \begin{cases} \emptyset \text{ if } S \in \mathbb{S} \\ Env_V[\mathbb{I}, \{S\} \cup \mathbb{S}](\mathcal{P}(S)) \end{cases}
\end{array}
$$

$$
\forall\ \mathbb{I}, x_i^{\mathsf{R}}, j, (x_j^{\mathsf{D}} \in Res[\mathbb{I}](x_i^{\mathsf{R}})) \iff (\exists p\ s.t.\ \mathbb{I} \vdash p : x_i^{\mathsf{R}} \longmapsto x_j^{\mathsf{D}}).
$$

In the Paper

- Formalization of the resolution calculus
- Validation by an extensive set of examples
- Sound and complete resolution algorithm
- Language-independent definition of alpha-equivalence

$$\frac{\vdash p : x_i^{\mathsf{R}} \longmapsto x_{i'}^{\mathsf{D}}}{i \overset{\mathtt{P}}{\sim} i'} \qquad \frac{i' \overset{\mathtt{P}}{\sim} i}{i \overset{\mathtt{P}}{\sim} i'} \qquad \frac{i \overset{\mathtt{P}}{\sim} i' \quad i' \overset{\mathtt{P}}{\sim} i''}{i \overset{\mathtt{P}}{\sim} i''} \qquad \frac{}{i \overset{\mathtt{P}}{\sim} i}$$

$$\mathtt{P1} \overset{\alpha}{\approx} \mathtt{P2} \triangleq \mathtt{P1} \simeq \mathtt{P2} \land \forall\, i\; i',\; i \overset{\mathtt{P1}}{\sim} i' \Leftrightarrow i \overset{\mathtt{P2}}{\sim} i'$$

In the Paper

- Formalization of the resolution calculus
- Validation by an extensive set of examples
- Sound and complete resolution algorithm
- Language-independent definition of alpha-equivalence

Future Work

- Interaction with types

In the Paper

- Formalization of the resolution calculus
- Validation by an extensive set of examples
- Sound and complete resolution algorithm
- Language-independent definition of alpha-equivalence

Future Work

- Interaction with types
- Resolution-sensitive program transformations

In the Paper

- Formalization of the resolution calculus
- Validation by an extensive set of examples
- Sound and complete resolution algorithm
- Language-independent definition of alpha-equivalence

Future Work

- Interaction with types
- Resolution-sensitive program transformations
- And more …

The Future of Language Design

A universal formalism to describe the name binding patterns in programming languages

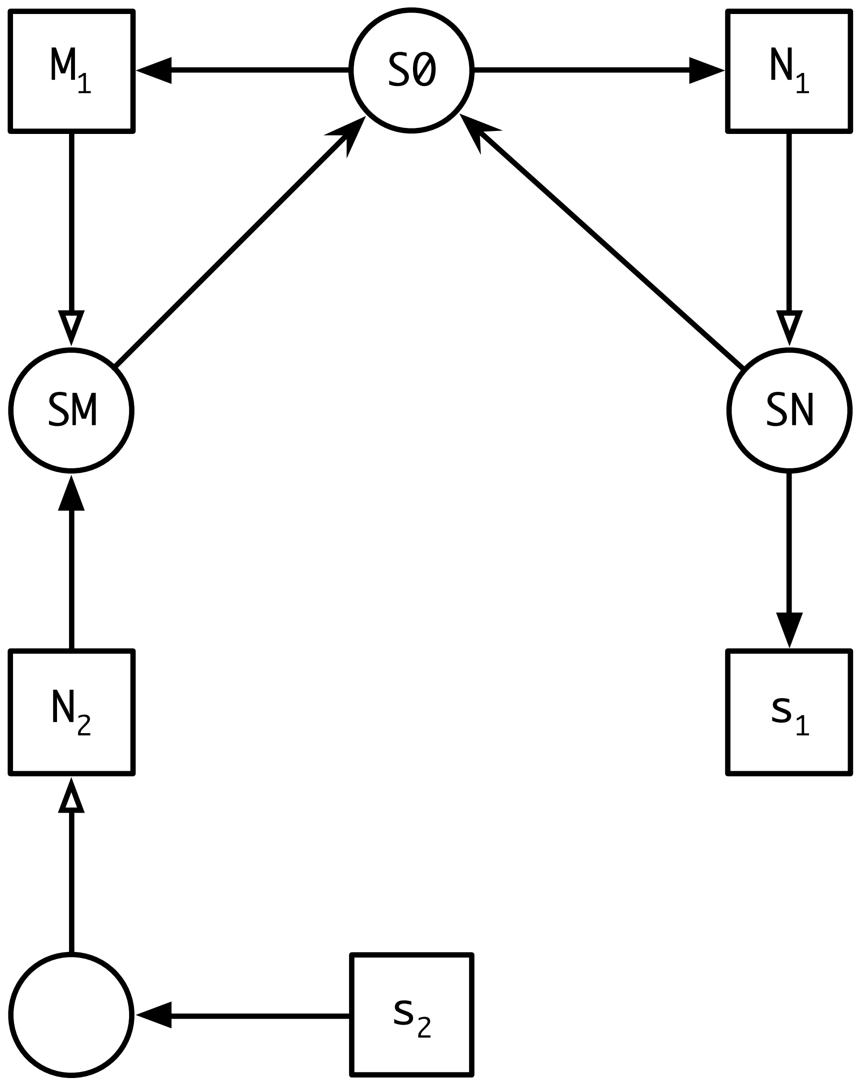
A single representation of the name binding structure of programs to reuse across name-sensitive language artifacts

Qualified Names

```
module N₁ {
    def s₁ = 5
}

module M₁ {

    def x₁ = 1 + N₂.s₂
}
```

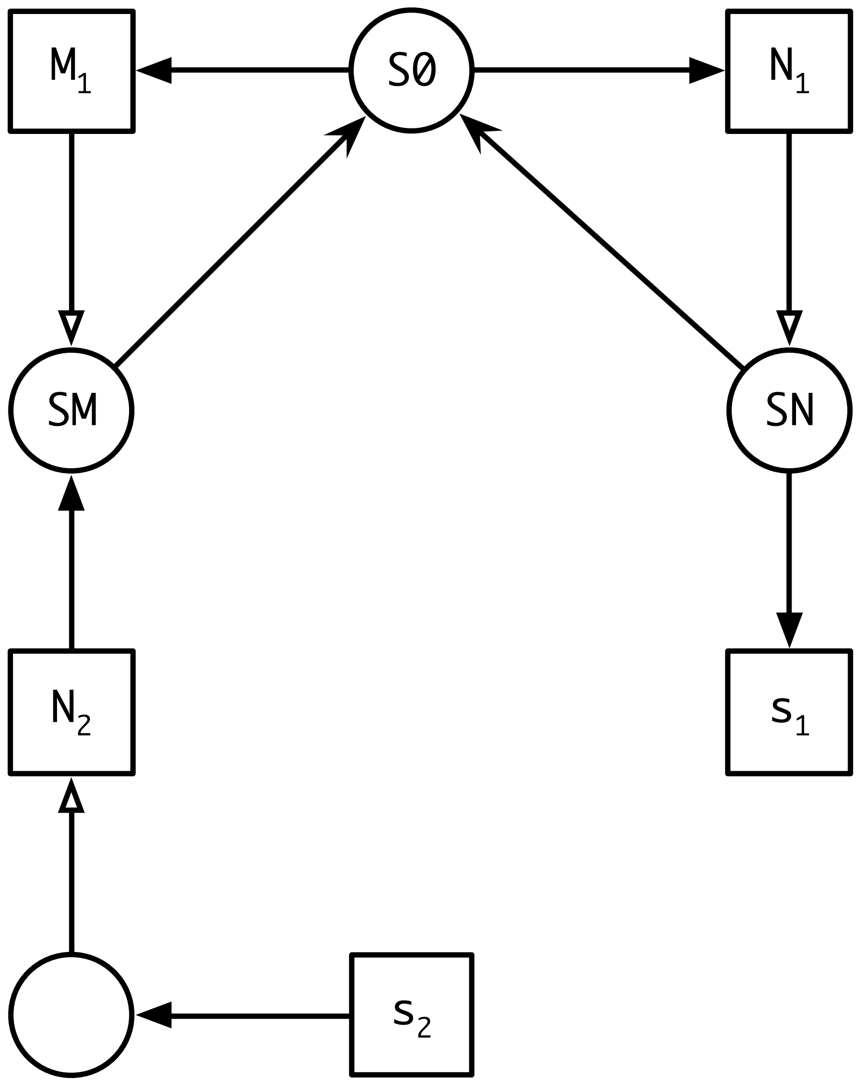
Qualified Names

```
module N₁ {
    def s₁ = 5
}

module M₁ {

    def x₁ = 1 + N₂.s₂
}
```



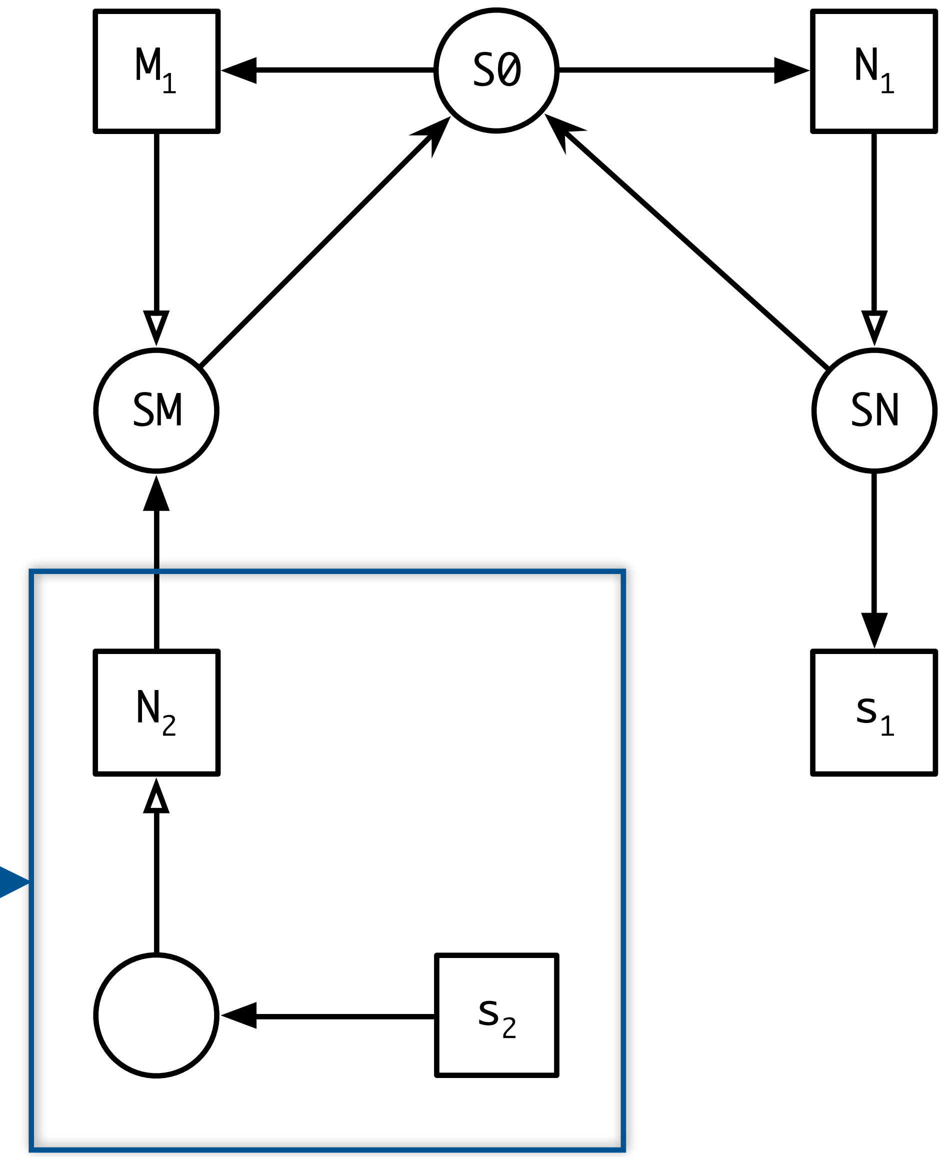
Qualified Names

```
module N₁ {
    def s₁ = 5
}

module M₁ {

    def x₁ = 1 + N₂.s₂
}
```



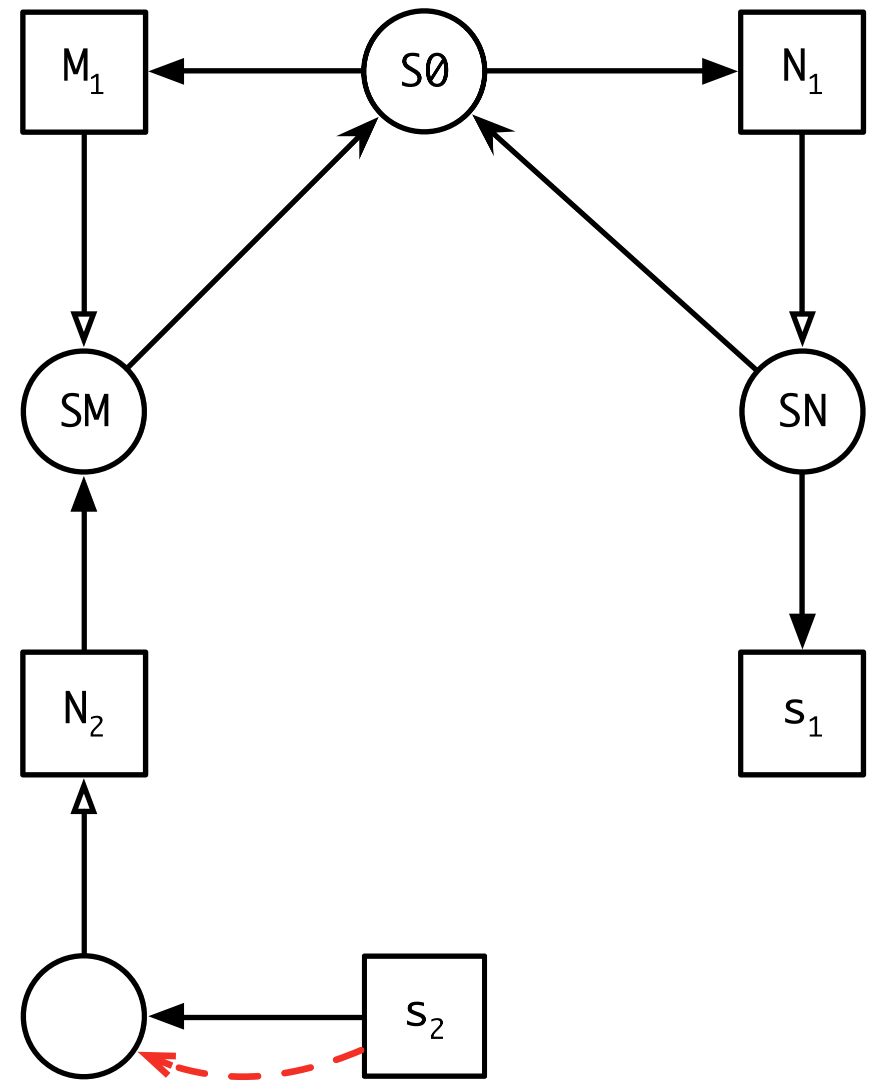
Qualified Names

```
module N_1 {
    def s_1 = 5
}

module M_1 {

    def x_1 = 1 + N_2.s_2
}
```



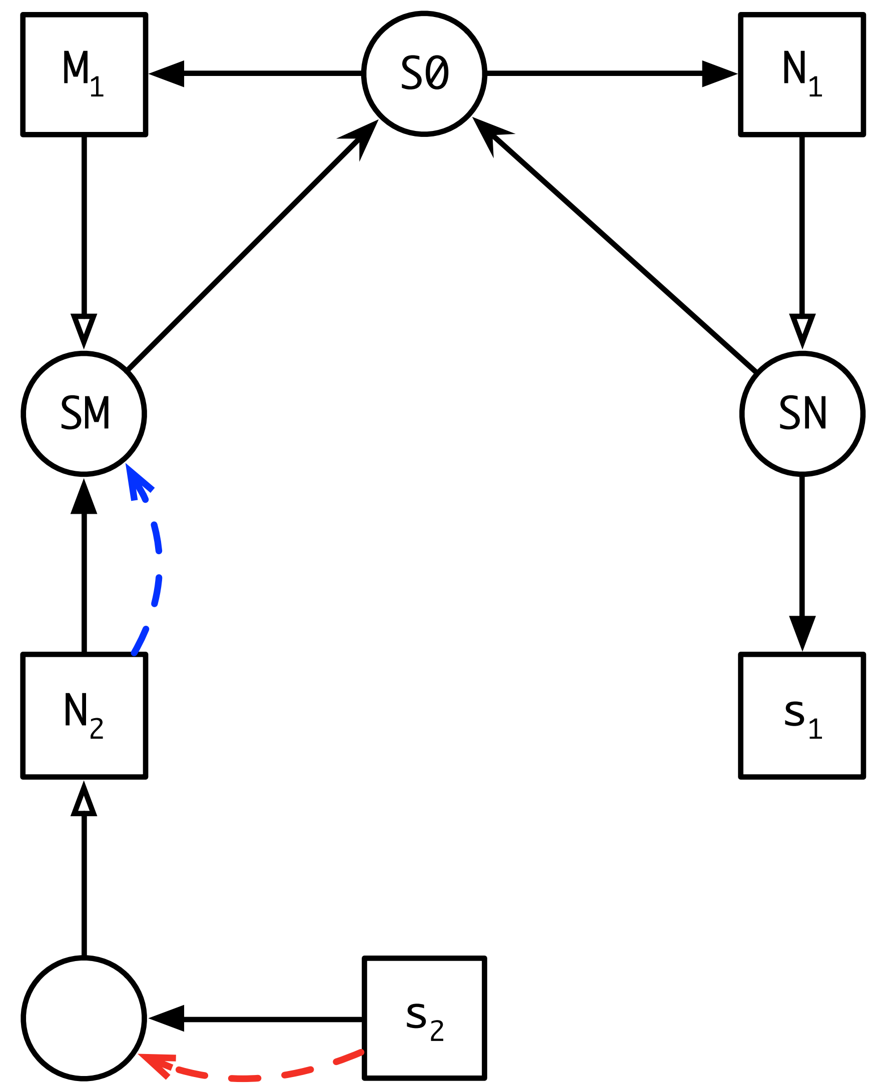
Qualified Names

```
module N₁ {
    def s₁ = 5
}

module M₁ {

    def x₁ = 1 + N₂.s₂
}
```



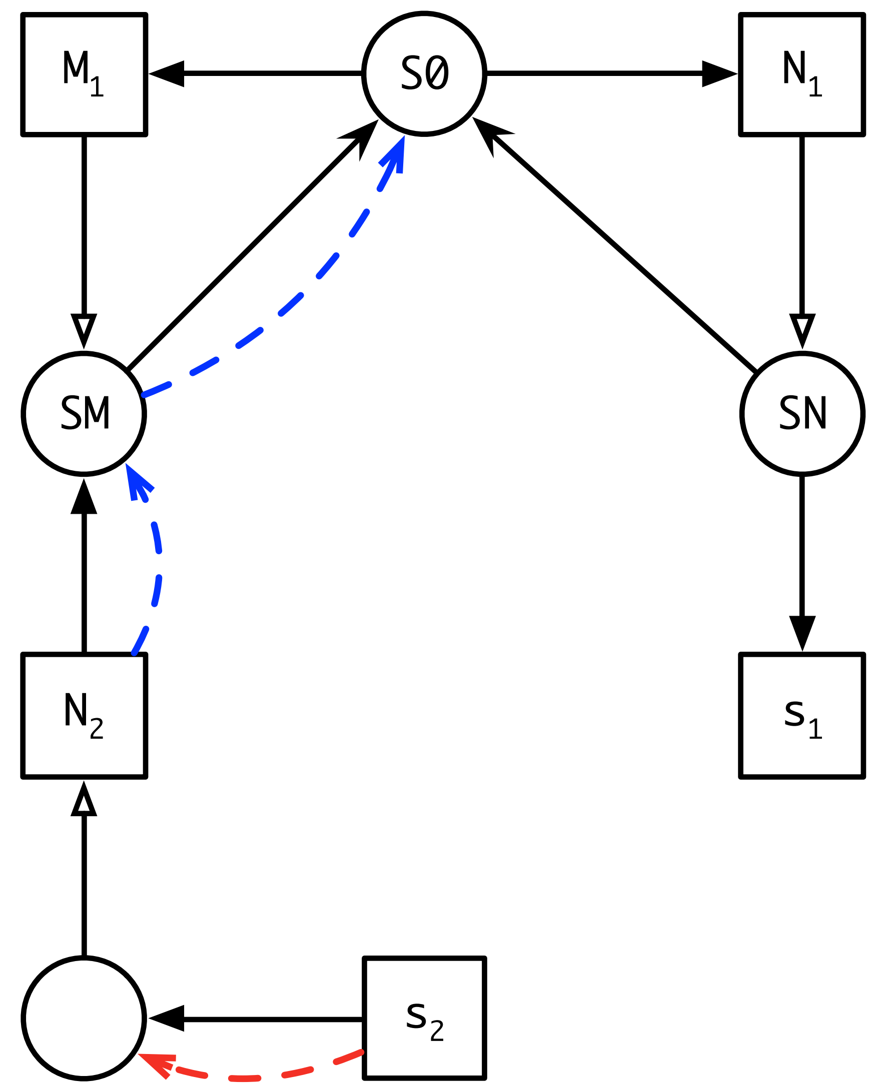
Qualified Names

```
module N₁ {
    def s₁ = 5
}

module M₁ {

    def x₁ = 1 + N₂.s₂
}
```



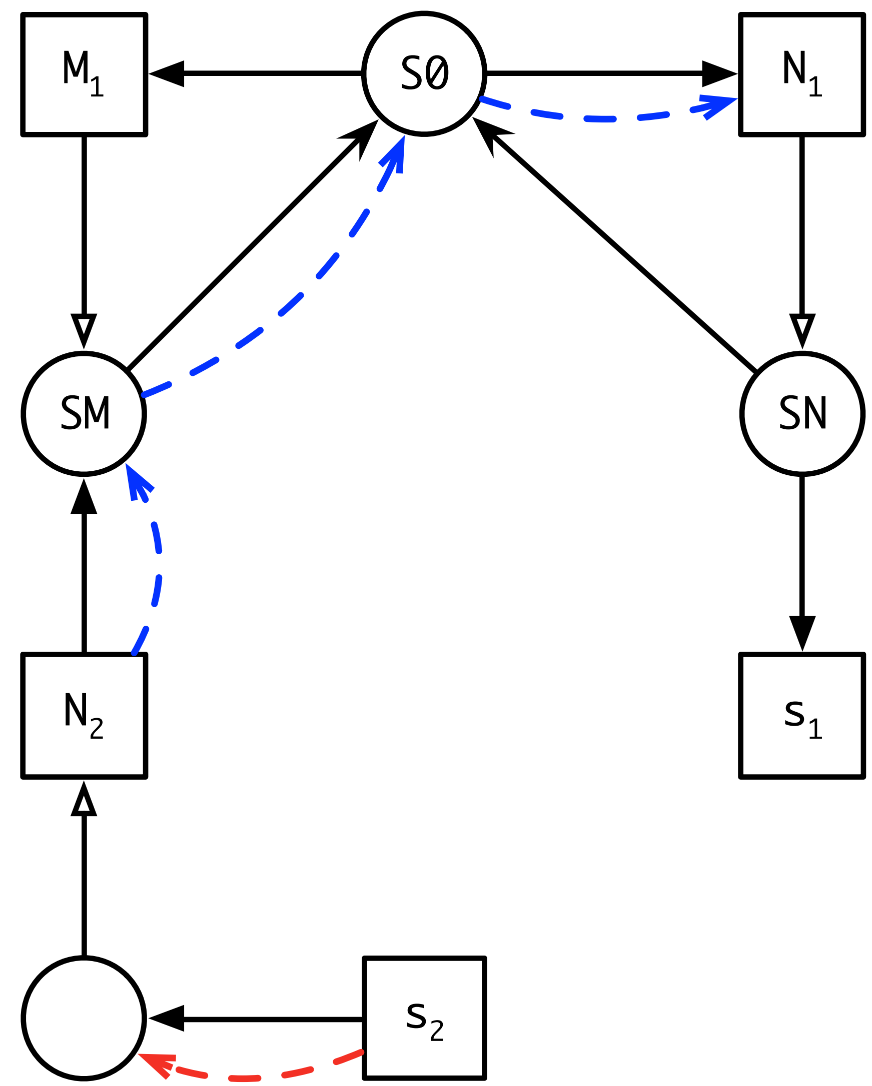
Qualified Names

```
module N₁ {
    def s₁ = 5
}

module M₁ {

    def x₁ = 1 + N₂.s₂
}
```



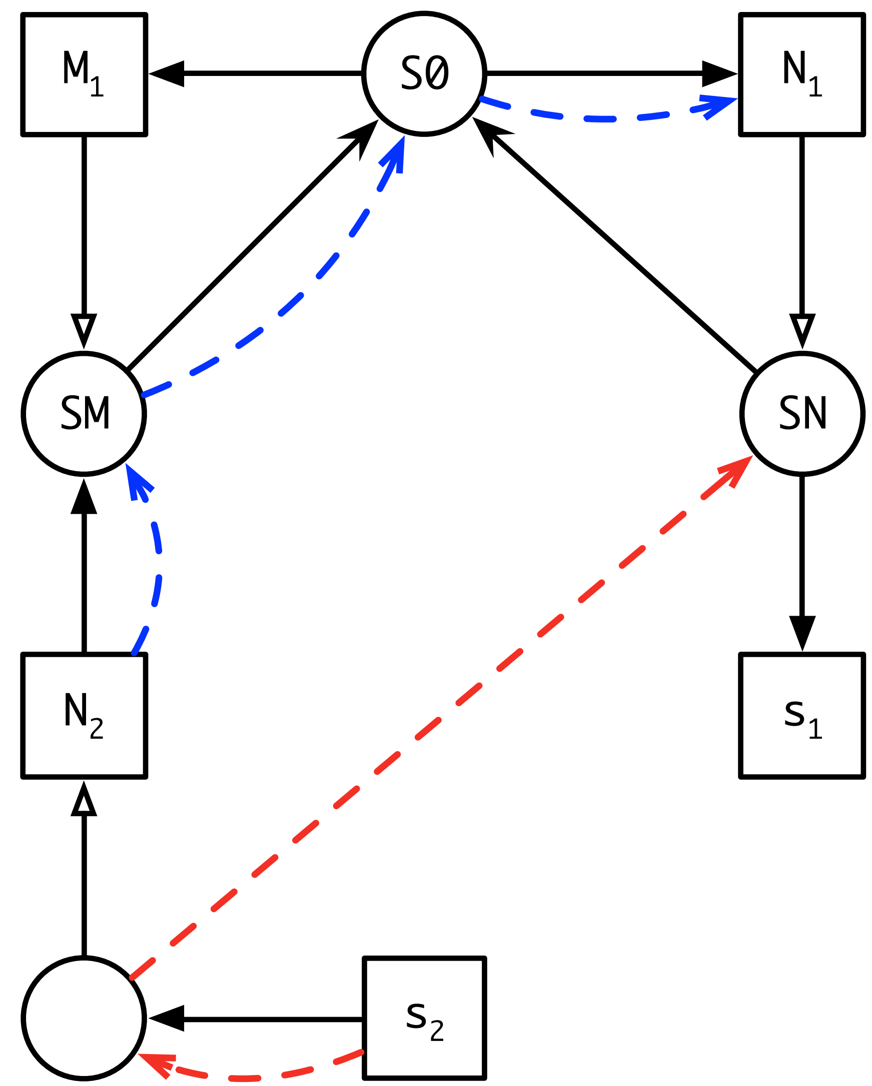
Qualified Names

```
module N₁ {
    def s₁ = 5
}

module M₁ {

    def x₁ = 1 + N₂.s₂
}
```



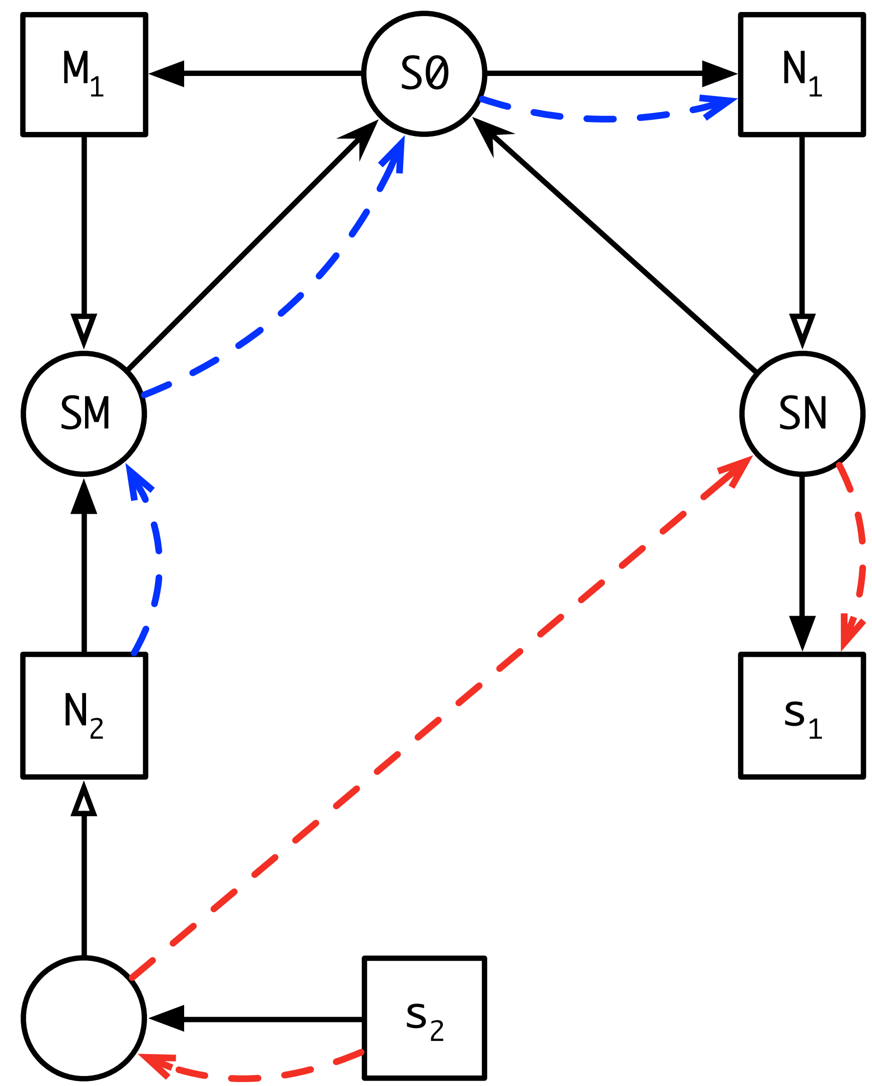
Qualified Names

```
module N₁ {
    def s₁ = 5
}

module M₁ {

    def x₁ = 1 + N₂.s₂
}
```



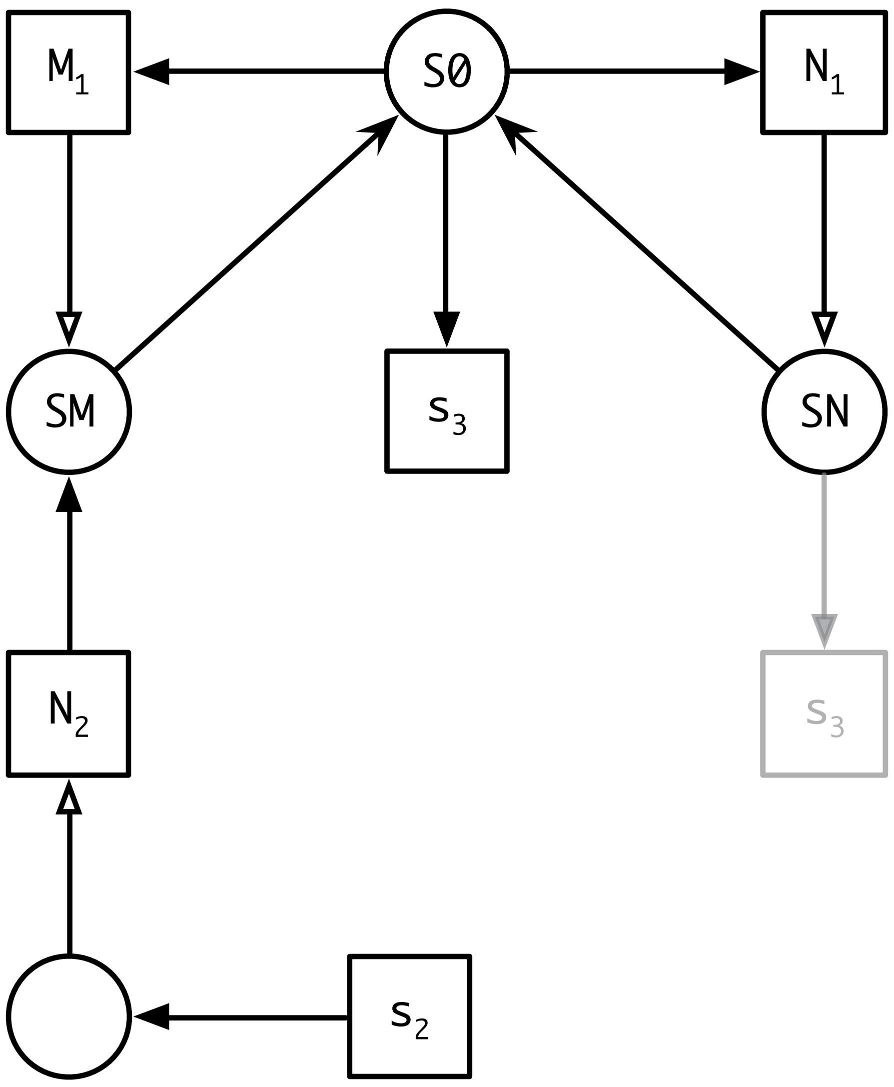
Qualified Names

```
module N₁ {
    def s₁ = 5
}

module M₁ {

    def x₁ = 1 + N₂.s₂
}
```



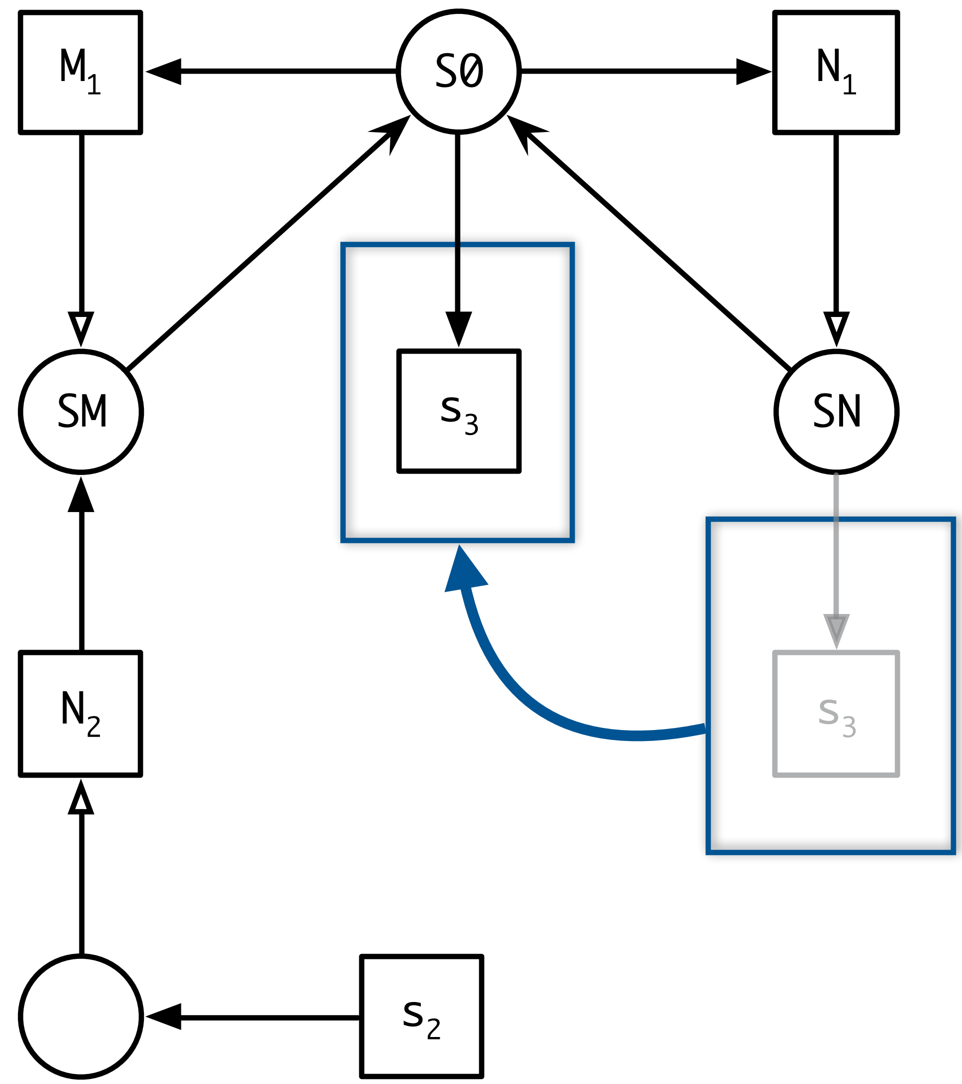
Well-Formedness

```
def s₃ = 6

module N₁ {

}

module M₁ {

    def x₁ = 1 + N₂.s₂
}
```



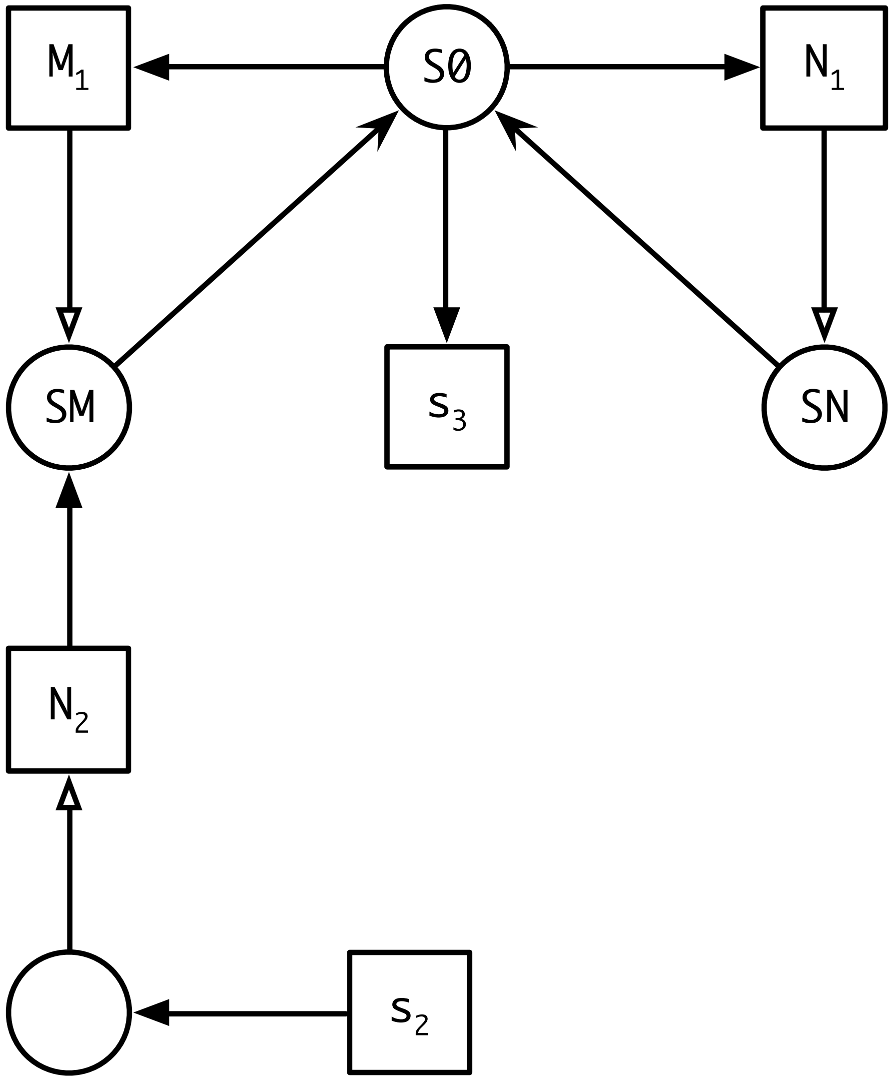
Well-Formedness

```
def s₃ = 6

module N₁ {

}

module M₁ {

    def x₁ = 1 + N₂.s₂
}
```



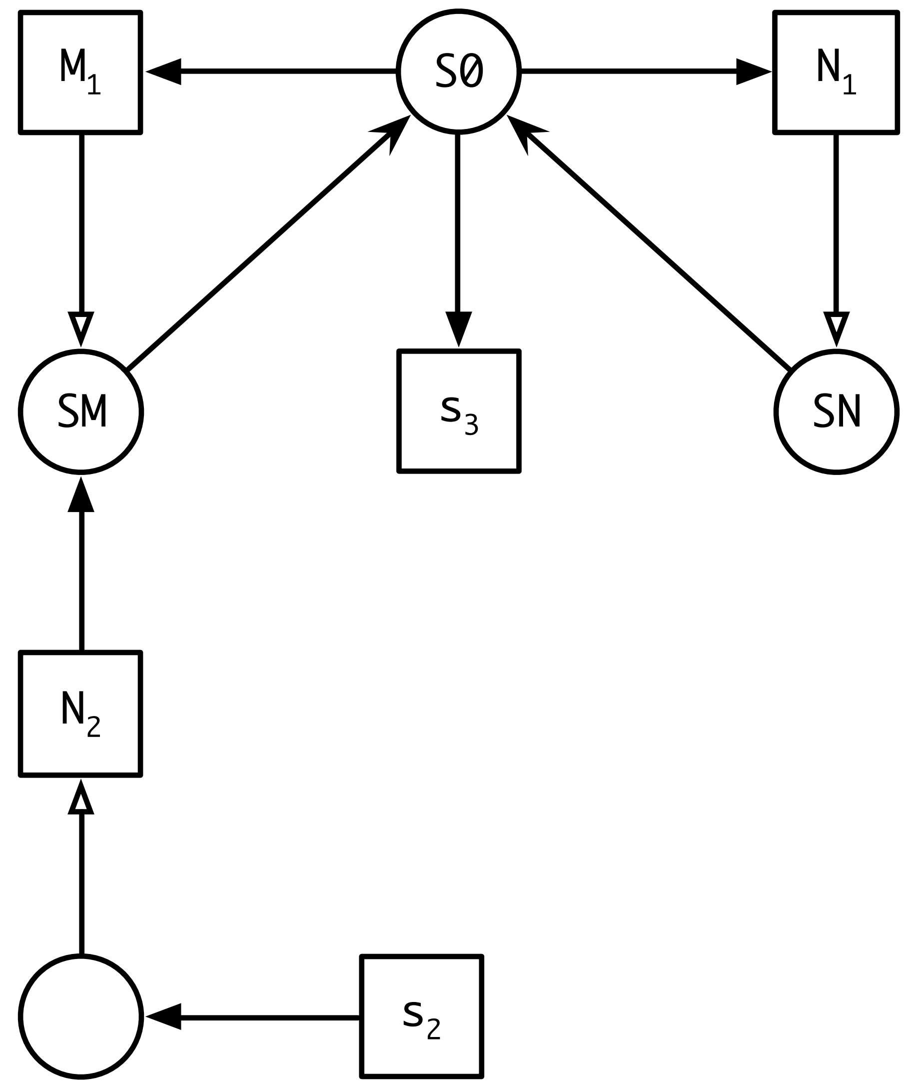
Well-Formedness

```
def s3 = 6

module N1 {

}

module M1 {

    def x1 = 1 + N2.s2
}
```



Well-Formedness

```
def s₃ = 6

module N₁ {

}

module M₁ {

    def x₁ = 1 + N₂.s₂
}
```



Well-Formedness

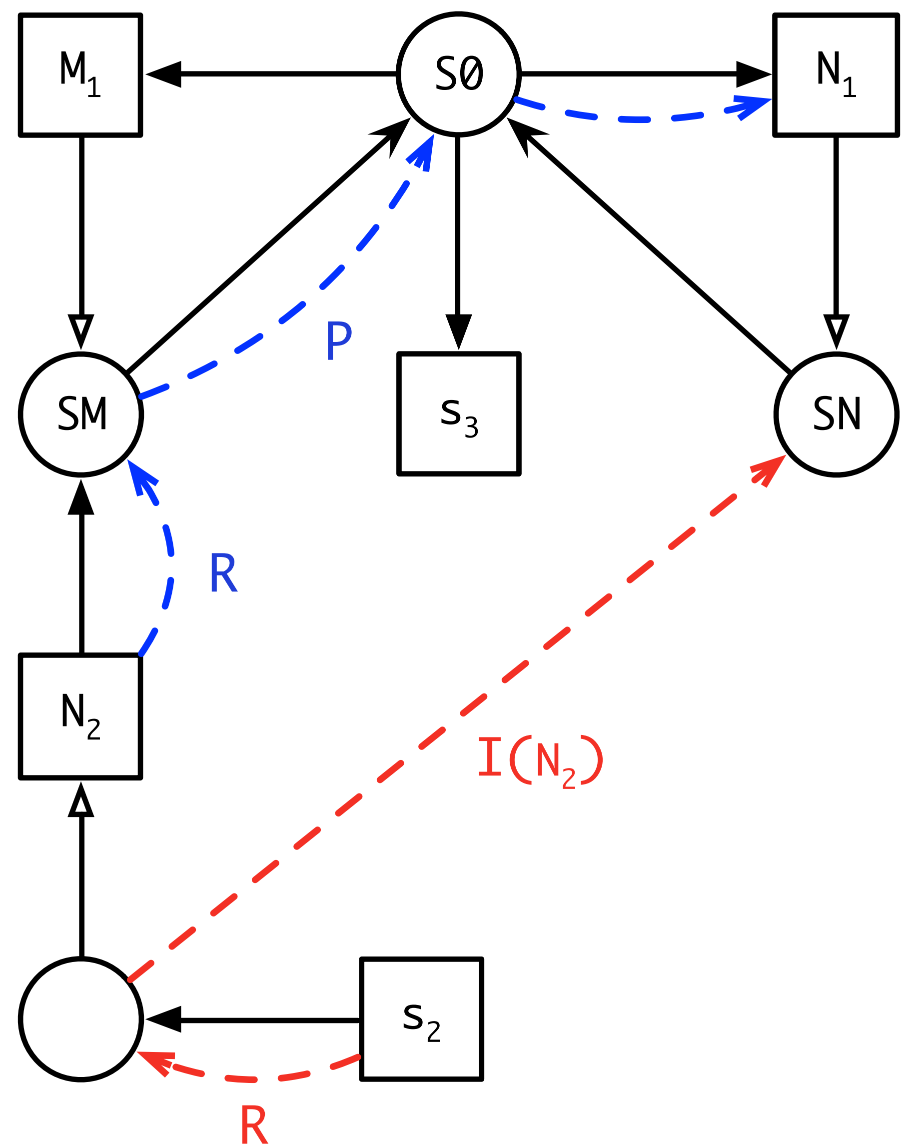
def s_3 = 6

module N_1 {

}

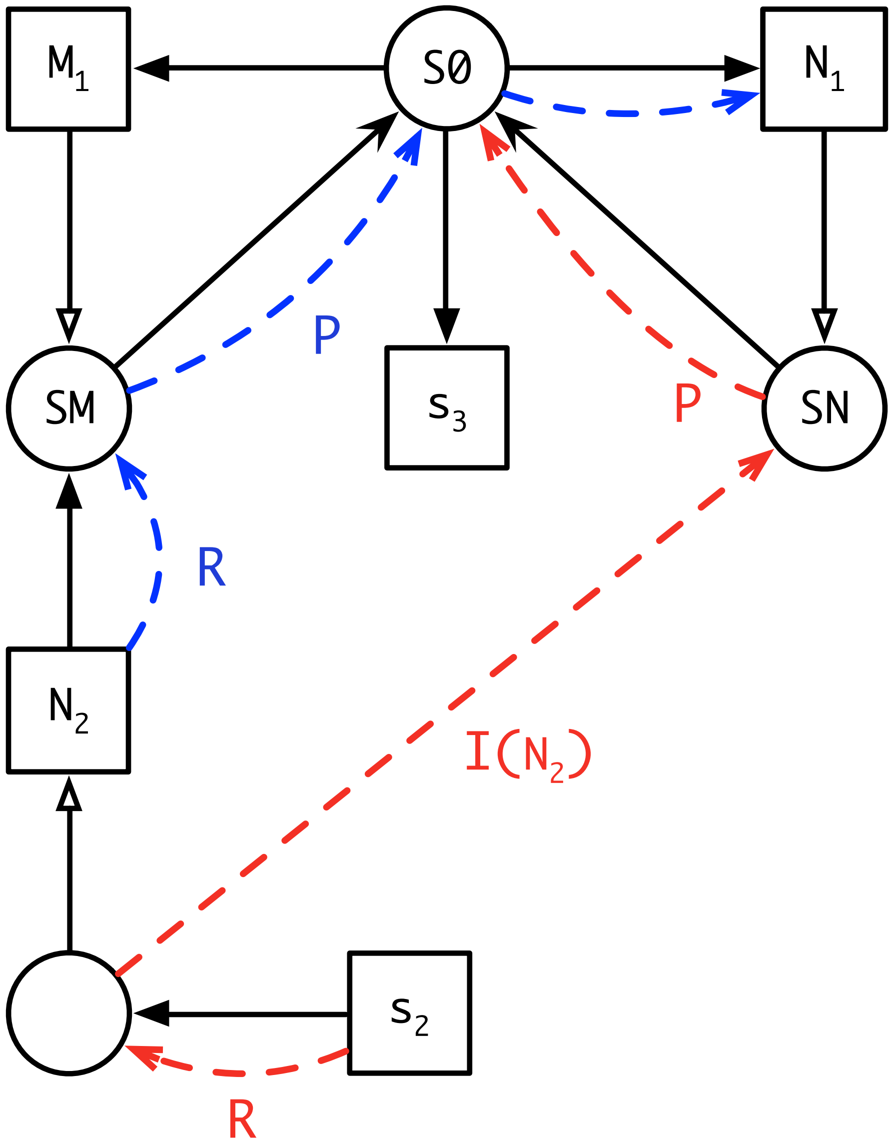
module M_1 {

    def x_1 = 1 + $N_2.s_2$
}



Well-Formedness

```
def s₃ = 6

module N₁ {

}

module M₁ {
    def x₁ = 1 + N₂.s₂
}
```



Well-Formedness

def s_3 = 6

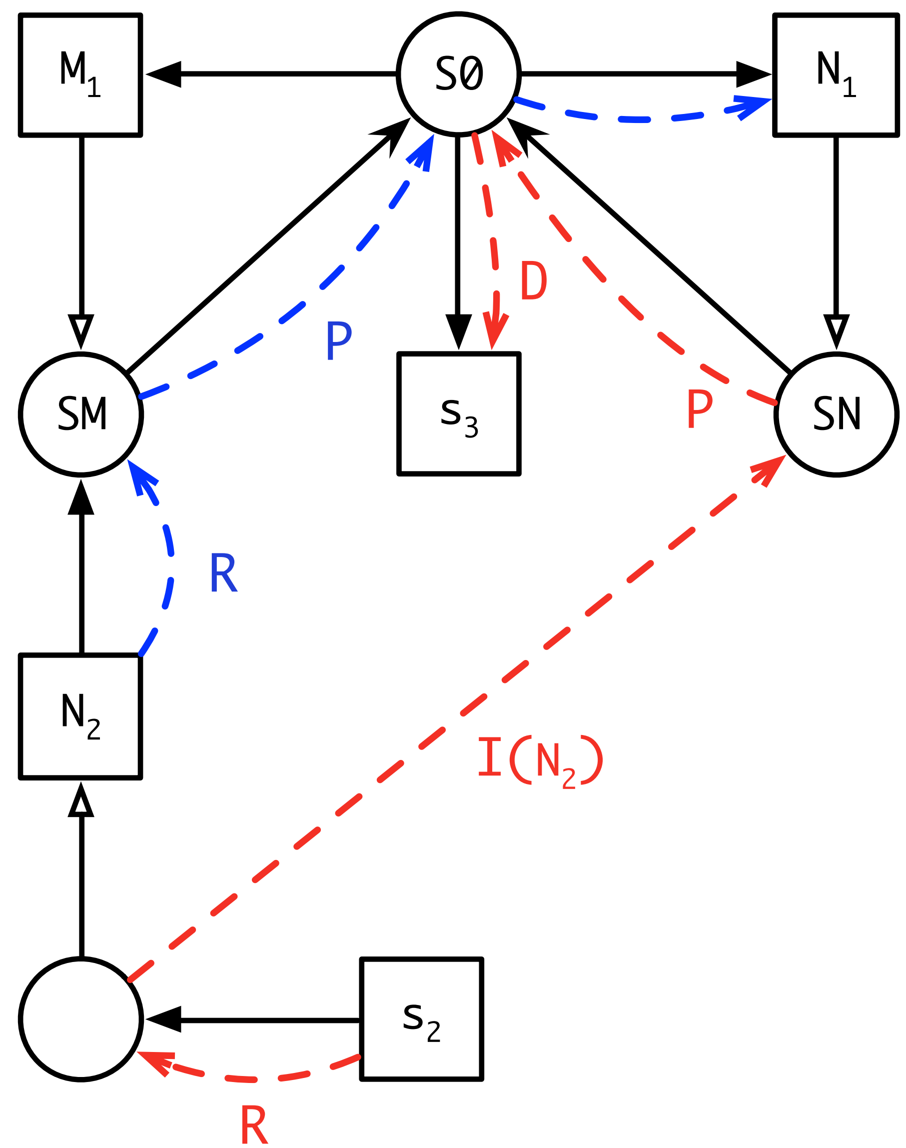
module N_1 {

}

module M_1 {

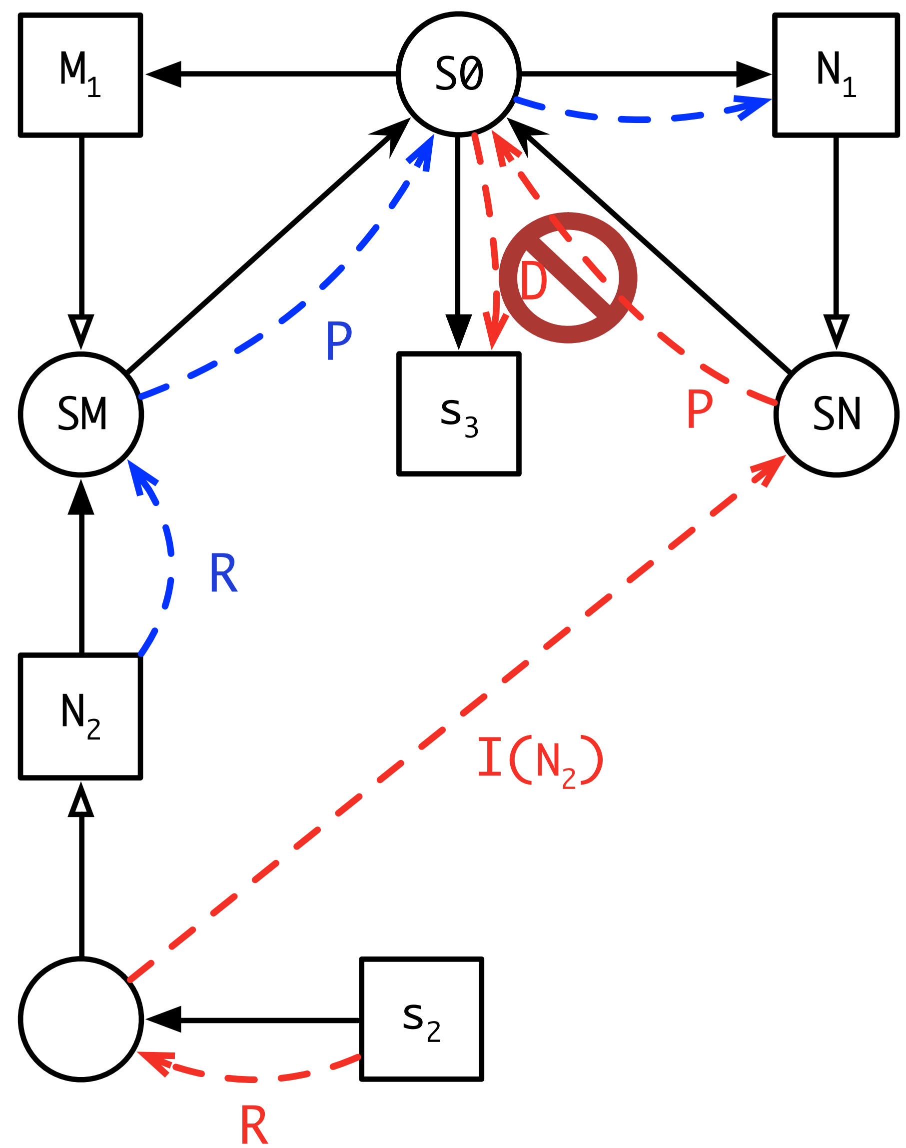
    def x_1 = 1 + $N_2.s_2$

}



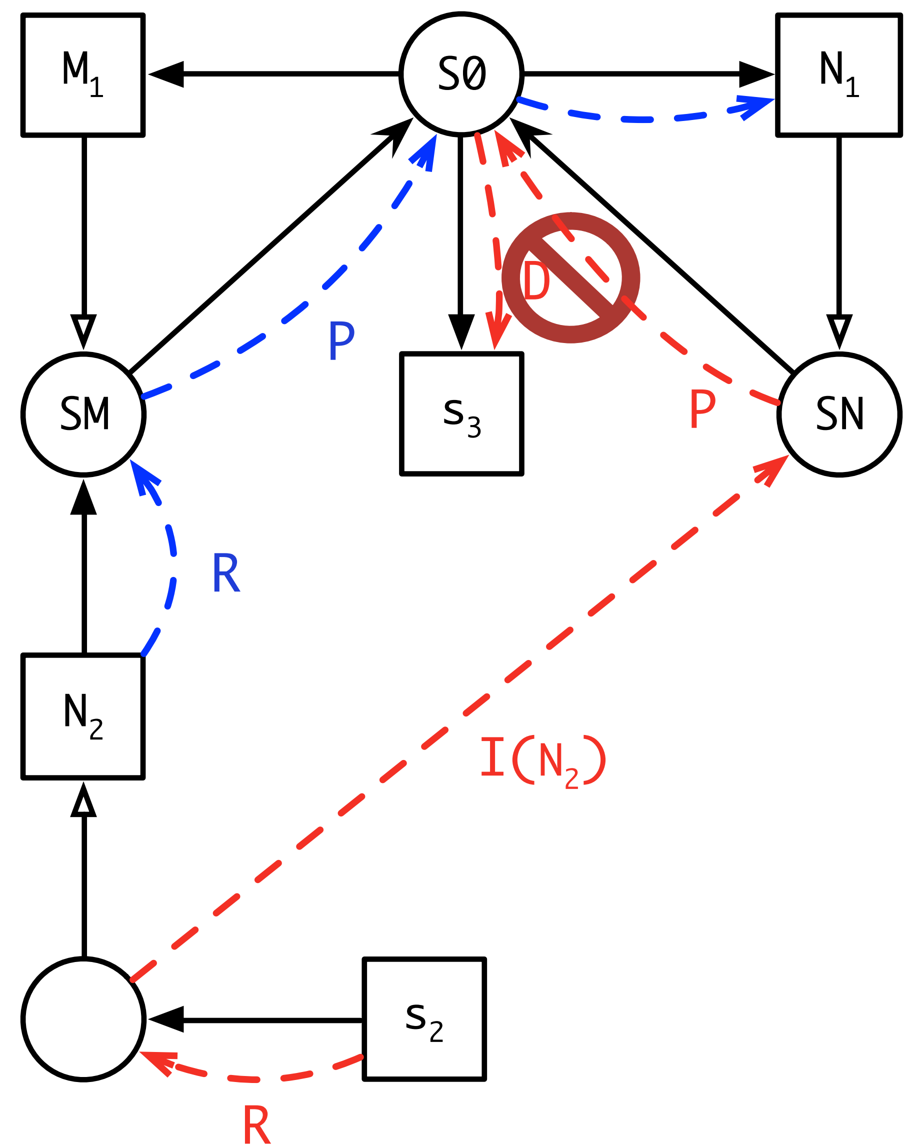
Well-Formedness

```
def s₃ = 6

module N₁ {

}

module M₁ {

    def x₁ = 1 + N₂.s₂
}
```



Well-Formedness

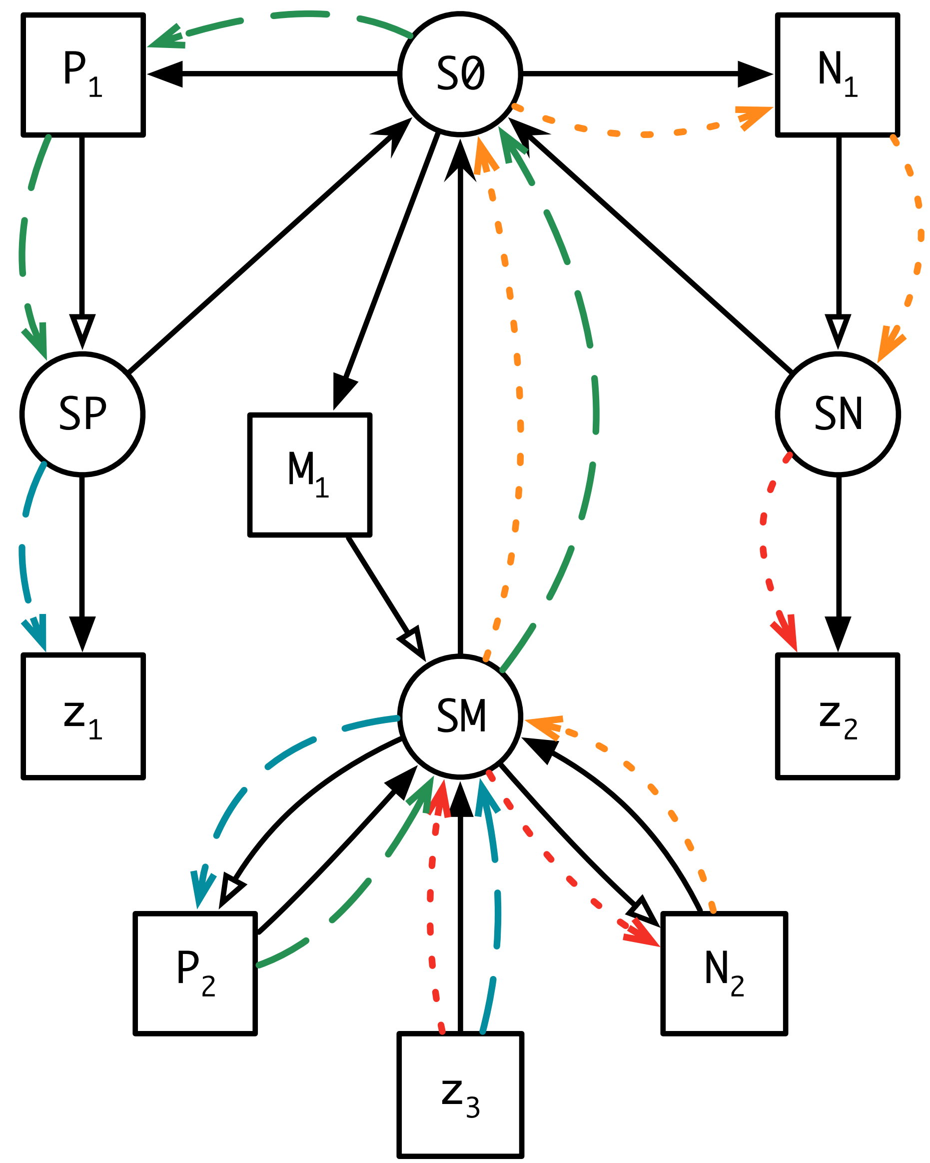
```
def s3 = 6

module N1 {

}

module M1 {

    def x1 = 1 + N2.s2
}
```



Well formed path: R.P*.I(_)*.D

Duplicate Resolution (imports)

```
module P₁ {
    def z₁ = 5
}

module N₁ {
    def z₂ = 5
}

module M₁ {

    import N₂
    import P₂

    def x₁ = 1 + z₃
}
```



Nested Imports

```
module P₁ {
    module N₁ {
        def z₂ = 5
}

module M₁ {

    import N₂
    import P₂

    def x₁ = 1 + z₃
}
```