

Incremental, zero-config Code Navigation using stack graphs

Douglas Creager
@dcreager



Languages, Systems, and Data Seminar
May 27, 2021 – UC Santa Cruz

Builds on the Scope Graphs framework
from Eelco Visser's group at TU Delft.

<https://pl.ewi.tudelft.nl/research/projects/scope-graphs/>

Builds on the Scope Graphs framework
from Eelco Visser's group at TU Delft.

<https://pl.ewi.tudelft.nl/research/projects/scope-graphs/>

Curry On Barcelona 2017

The screenshot shows a presentation slide with the following content:

- Title:** Scope Graphs – A Fresh Look at Name Binding in Programming
- Author:** Languages
Eelco Visser
- Code Snippets:**

```
module A1 {  
    def z1 = 5  
}  
module A2 {  
    import A1  
    def x1 = 1 + z1  
}
```
- Diagram:** A scope graph labeled "imports". It shows nodes for variables z1, x1, and z2, and functions A1 and A2. Edges connect z1 to A1, A1 to x1, and A1 to z2. A2 has an incoming edge from z2.
- Logos:** HUAWEI and CURRY ON Barcelona!

Code Navigation



Code Navigation

stove.py

```
def bake():
    pass
```

```
def broil():
    pass
```

```
def saute():
    pass
```

```
broil()
```

Code Navigation

stove.py

```
def bake():
    pass
```

```
def broil():
    pass
```

```
def saute():
    pass
```

```
broil()
```

Code Navigation

```
stove.py
def bake():
    pass

def broil():
    pass
    ^

def saute():
    pass

broil()
```

The diagram illustrates code navigation within a Python file named 'stove.py'. It shows four function definitions: 'bake()', 'broil()', 'saute()', and 'bake()'. A red curved arrow originates from the word 'broil' in the fourth line and points back to the start of the 'broil()' function definition. The file name 'stove.py' is positioned above the first function definition.

A white marble statue of a man, likely David by Michelangelo, is shown from the waist up. He is in a state of deep despair, with his head buried in his hands and his body slumped forward. A dark-colored pigeon is perched on top of his head, facing away from the viewer. The background is a clear blue sky.

Why is this hard?

Why is this hard?

stove.py

```
def broil():
    pass
```

```
def broil():
    pass
```

```
def saute():
    pass
```

```
broil()
```

Why is this hard?

stove.py

```
def broil():
    pass
```

```
def broil():
    pass
```

```
def saute():
    pass
```

```
broil()
```

Why is this hard?

```
stove.py
def broil():
    pass

def broil():
    pass

def saute():
    pass

broil()
```

The diagram illustrates a self-referencing loop in the `stove.py` code. It shows four definitions of the `broil()` function. The first two definitions are highlighted with pink boxes around their names. A red dotted arrow points from the end of the second definition back up to its own name, forming a loop. The third definition is highlighted with a blue box around its name. The fourth definition is highlighted with a purple box around its name.

Why is this hard?

stove.rs

```
fn broil() {}
```

```
fn broil() {}
```

```
fn saute() {}
```

```
fn main() {  
    broil();  
}
```

Why is this hard?

stove.rs

```
fn broil() {}
```

```
fn broil() {}
```

```
fn saute() {}
```

```
fn main() {  
    broil();  
}
```

Why is this hard?

stove.rs

```
fn broil() {}
```

```
fn bril() {}  
X
```

```
fn saute() {}
```

```
fn main() {  
    broil();  
}
```

Why is this hard?

stove.py

```
def bake():
    pass

def broil():
    pass

def saute():
    pass
```

kitchen.py

```
from stove import broil

broil()
```

Why is this hard?

stove.py

```
def bake():
    pass

def broil():
    pass

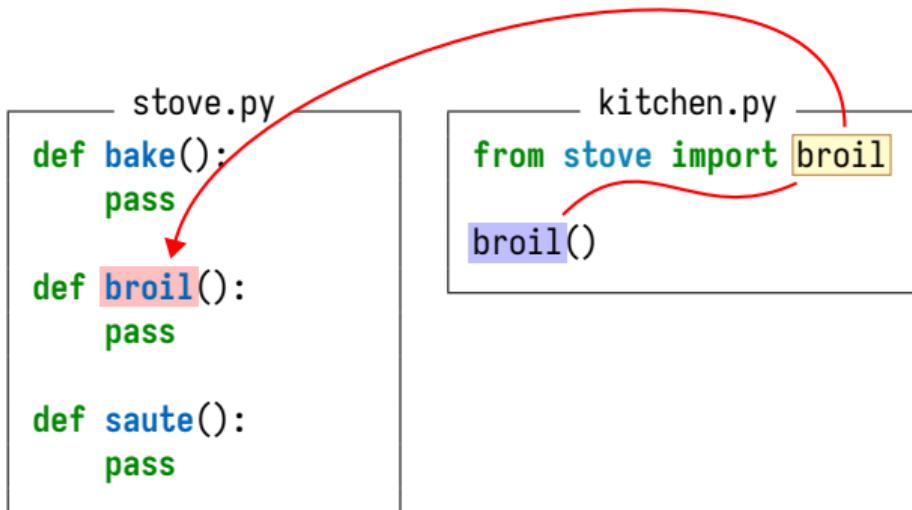
def saute():
    pass
```

kitchen.py

```
from stove import broil

broil()
```

Why is this hard?



Why is this hard?



stove.py

```
def bake():
    pass

def broil():
    pass

def saute():
    pass
```

kitchen.py

```
from stove import *
```



chef.py

```
from kitchen import broil

broil()
```

Why is this hard?



stove.py

```
def bake():
    pass

def broil():
    pass

def saute():
    pass
```

kitchen.py

```
from stove import *
```

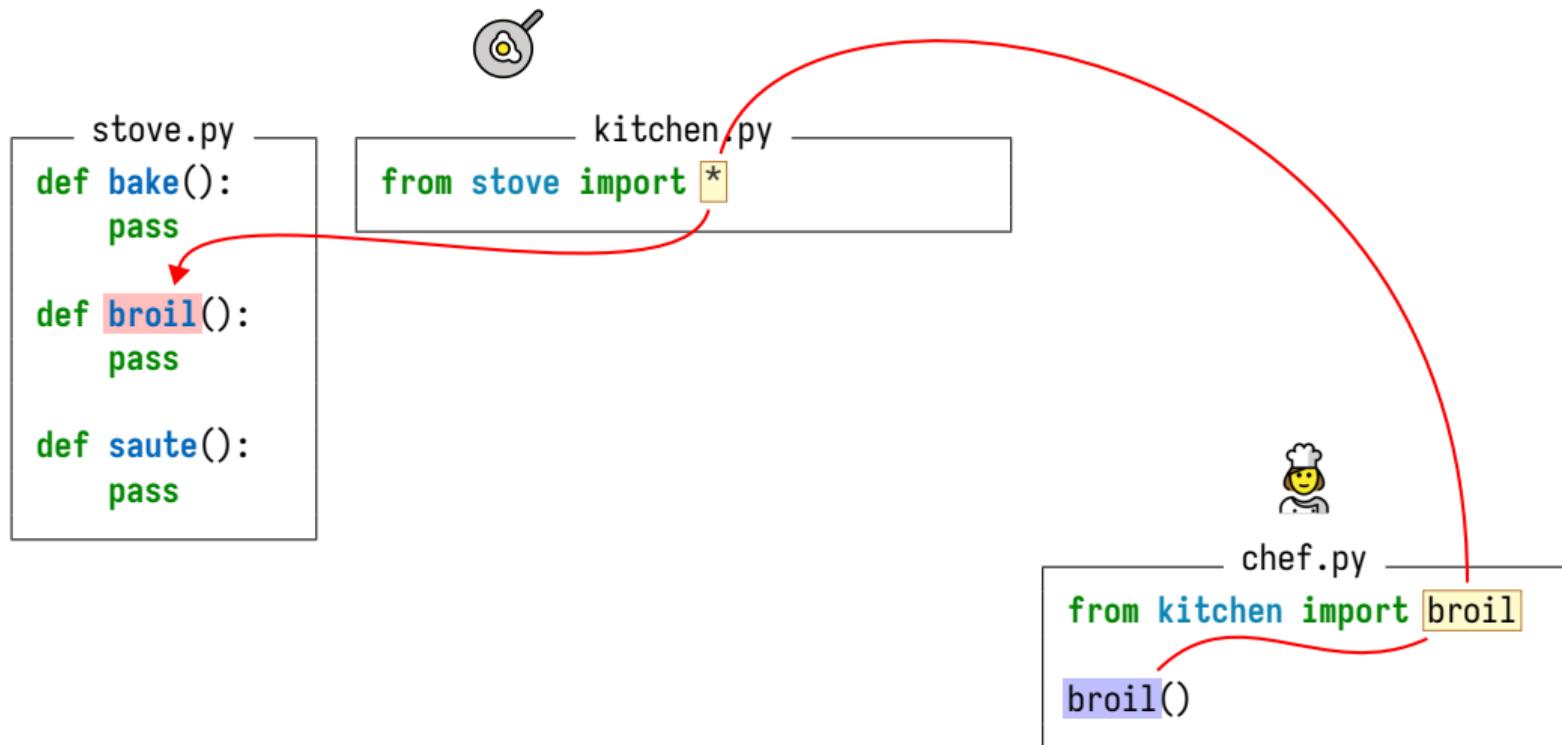


chef.py

```
from kitchen import broil

broil()
```

Why is this hard?



Why is this hard?



stove.py

```
def bake():
    pass

def broil():
    pass

def saute():
    pass
```

kitchen.py

```
from stove import *

def broil():
    print("We're broiling!")
    import stove
    return stove.broil()
```



chef.py

```
from kitchen import broil

broil()
```

Why is this hard?



stove.py

```
def bake():
    pass

def broil():
    pass

def saute():
    pass
```

kitchen.py

```
from stove import *

def broil():
    print("We're broiling!")
    import stove
    return stove.broil()
```



chef.py

```
from kitchen import broil

broil()
```

Why is this hard?



```
stove.py
def bake():
    pass

def broil():
    pass

def saute():
    pass
```

```
kitchen.py
from stove import *

def broil():
    print("We're broiling!")
    import stove
    return stove.broil()
```



```
chef.py
from kitchen import broil

broil()
```

Why is this hard?



```
stove.py
```

```
class Stove(object):  
    def bake(self):  
        pass  
  
    def broil(self):  
        pass  
  
    def saute(self):  
        pass
```

```
kitchen.py
```

```
from stove import *
```



```
chef.py
```

```
from kitchen import Stove  
  
stove = Stove()  
stove.broil()
```

Why is this hard?



```
stove.py
```

```
class Stove(object):  
    def bake(self):  
        pass  
  
    def broil(self):  
        pass  
  
    def saute(self):  
        pass
```

```
kitchen.py
```

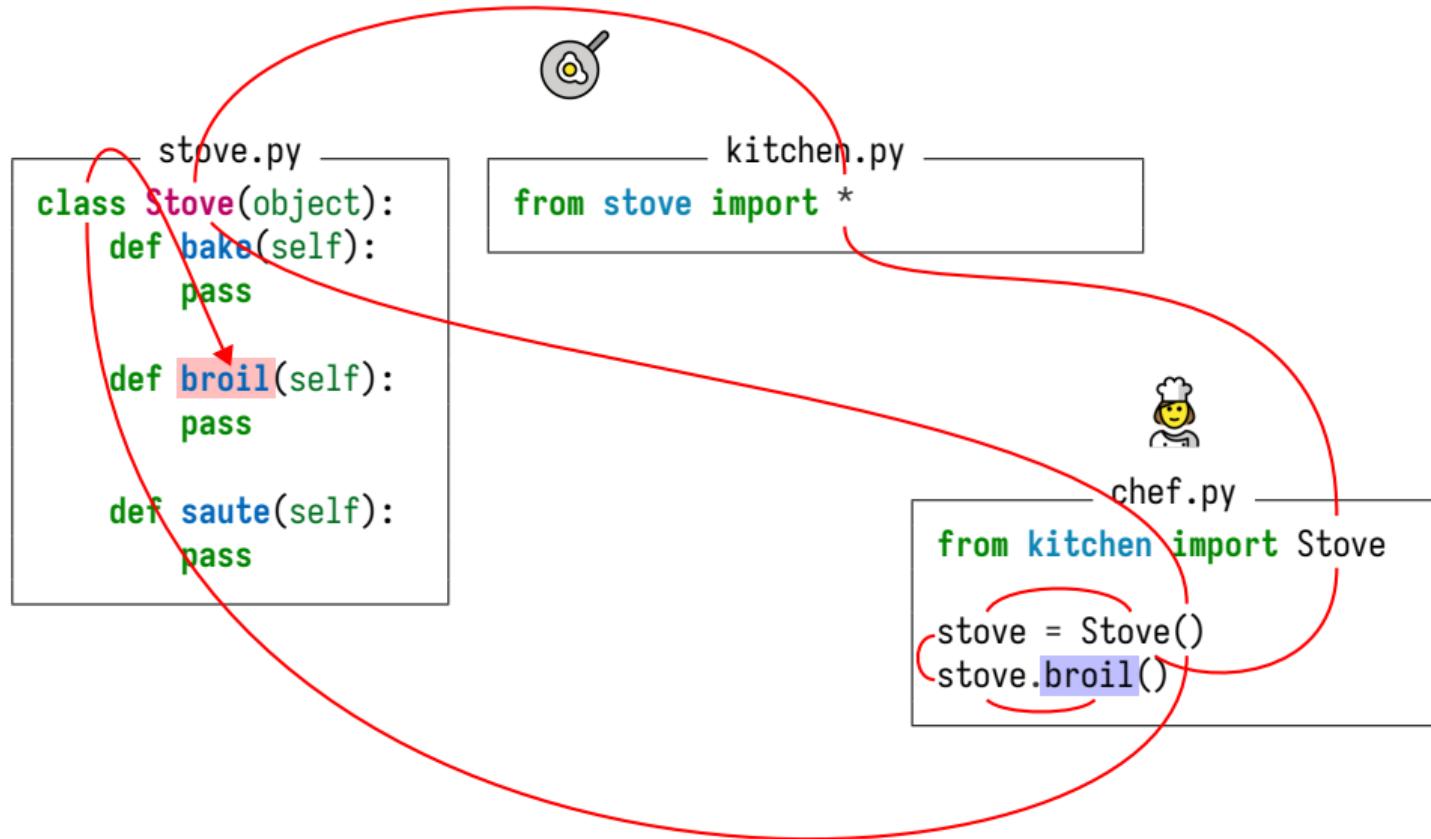
```
from stove import *
```



```
chef.py
```

```
from kitchen import Stove  
  
stove = Stove()  
stove.broil()
```

Why is this hard?



Why is this hard?

dataflow.py

```
def passthrough(x):
    return x
```

a.py

```
from dataflow import passthrough

class A:
    one = 1

passthrough(A).one
```

Why is this hard?

dataflow.py

```
def passthrough(x):
    return x
```

a.py

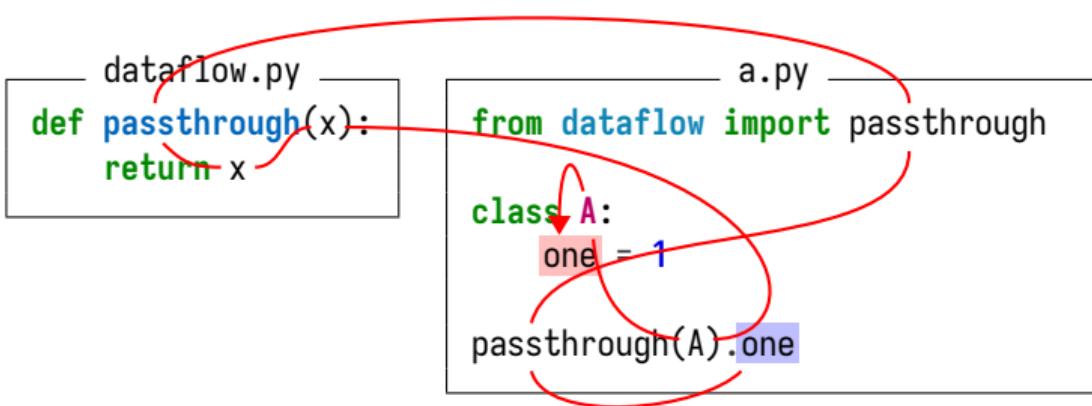
```
from dataflow import passthrough
```

```
class A:
```

```
    one = 1
```

```
passthrough(A).one
```

Why is this hard?





Oh is that all?

Zero configuration

We don't want to have to ask the package owner how to collect the data we need.

Or ask them to configure a job to produce that data.

It should **Just Work.**

SCALE

200 million repositories and counting

2 billion contributions
in the last 12 months

500 programming languages



When do we do the work?

Index

Query

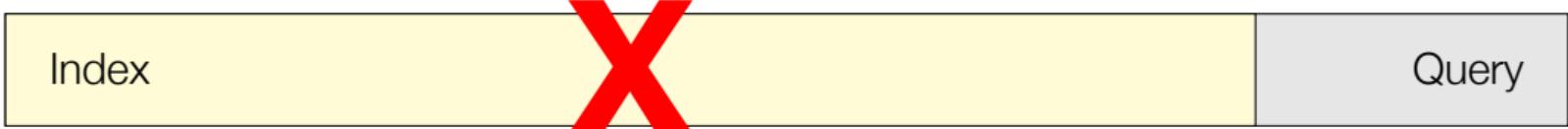
When do we do the work?



This is an interactive feature, so we can't do too much work at query time.

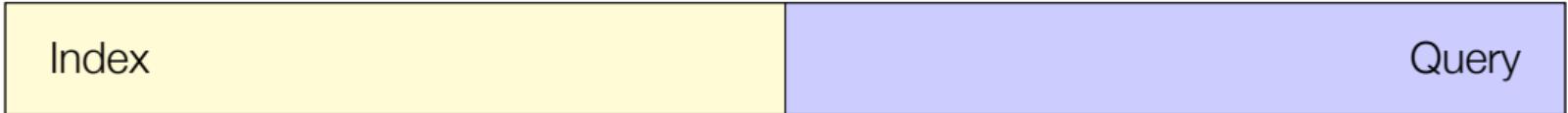
Goal: < 100ms

When do we do the work?



Because of our scale, we can't do too much work at index time, either!
(Compute and storage costs are too high, work is wasted, etc.)

When do we do the work?



We want to strike a balance.

Precalculate as much as we can.

Minimize the amount of **duplicated** work.

Defer **some** work until query time to make that happen.

Why is this hard?

- ▶ Different languages have different name binding rules.
- ▶ Some of those rules can be quite complex.
- ▶ The result might depend on intermediate files.
- ▶ We don't want to require manual per-repo configuration.
- ▶ We need to balance work between index time vs query time.

Incremental results



Incremental results

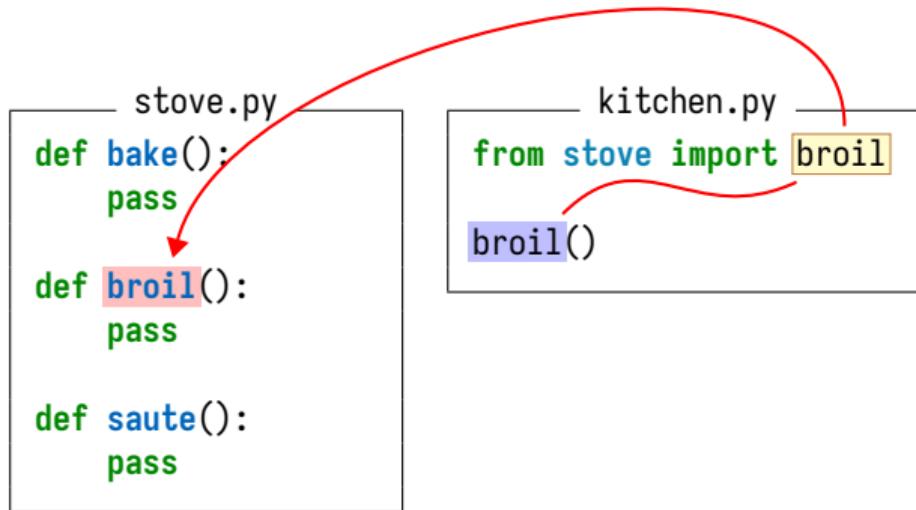
In a typical commit, a small fraction of files in the repo change.

We want to reuse results that we've already calculated for unchanged files.

Structural sharing (like git itself) helps save storage.

Incremental processing also helps save compute.

What would incremental results look like?



What would incremental results look like?

stove.py

```
def bake():
    pass
```

```
def broil():
    pass
```

```
def saute():
    pass
```

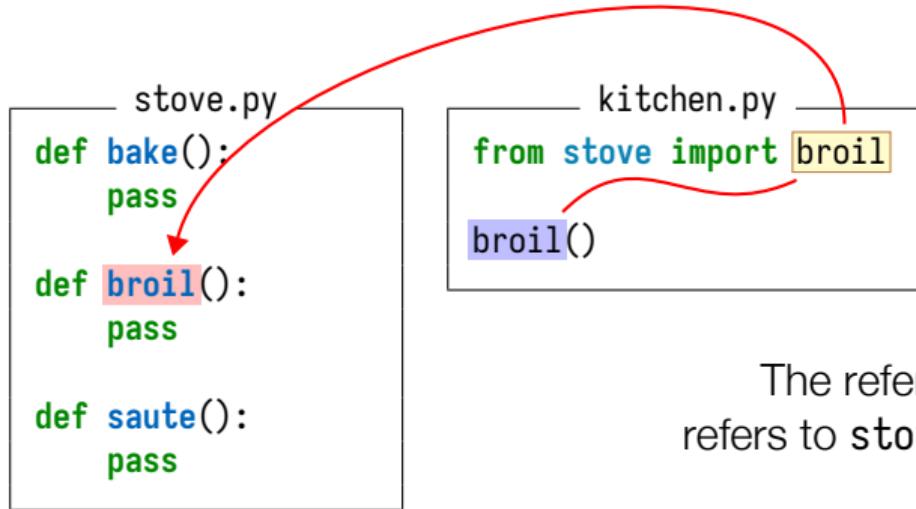
stove.broil is defined at stove.py:4:5

What would incremental results look like?

```
kitchen.py
from stove import broil
broil()
```

The reference at *kitchen.py:3:1*
refers to **stove.broil** in some other file

What would incremental results look like?



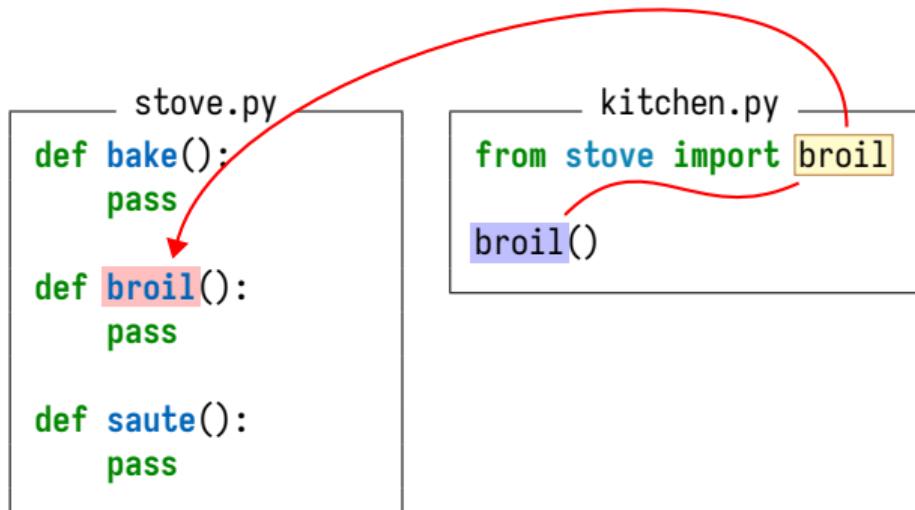
The reference at `kitchen.py:3:1`
refers to `stove.broil` in some other file
+
`stove.broil` is defined at `stove.py:4:5`

=
The reference at `kitchen.py:3:1`
is defined at `stove.py:4:5`

Stack graphs



Stack graphs



Stack graphs

stove.py

```
def bake():
    pass
```

```
def broil():
    pass
```

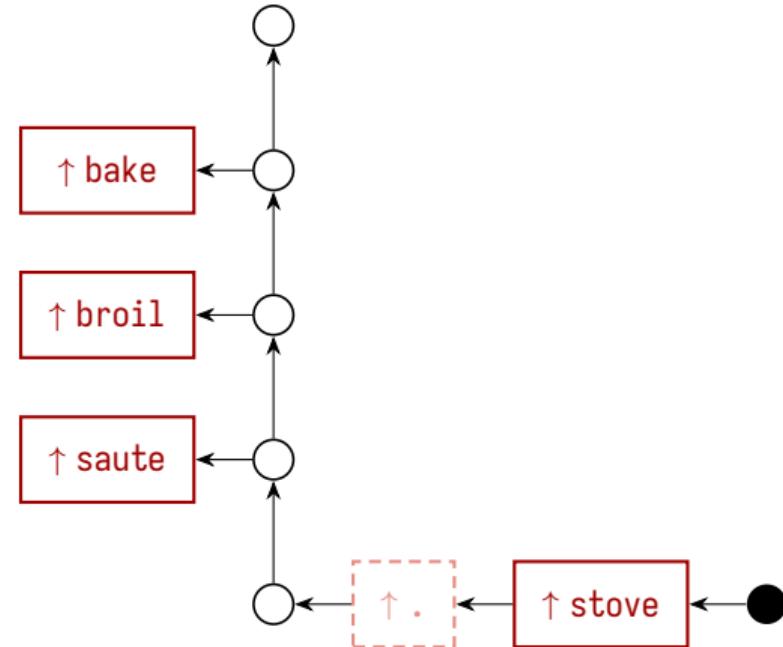
```
def saute():
    pass
```

Stack graphs

```
stove.py
def bake():
    pass

def broil():
    pass

def saute():
    pass
```

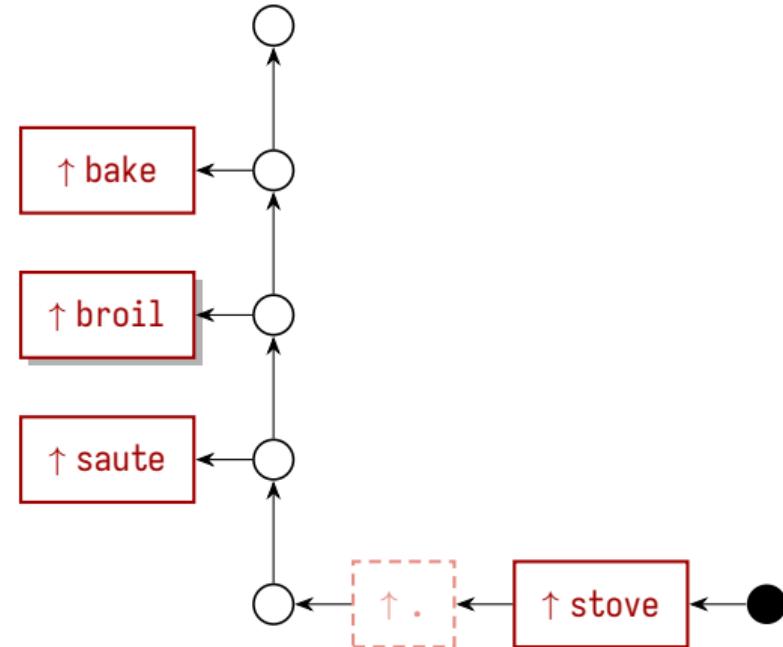


Stack graphs

```
stove.py
def bake():
    pass

def broil():
    pass

def saute():
    pass
```

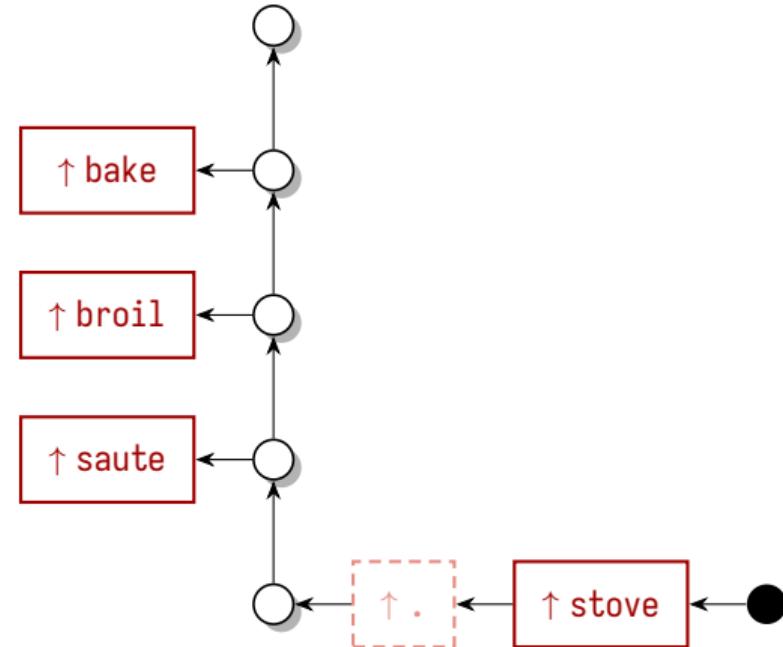


Stack graphs

```
stove.py
def bake():
    pass

def broil():
    pass

def saute():
    pass
```

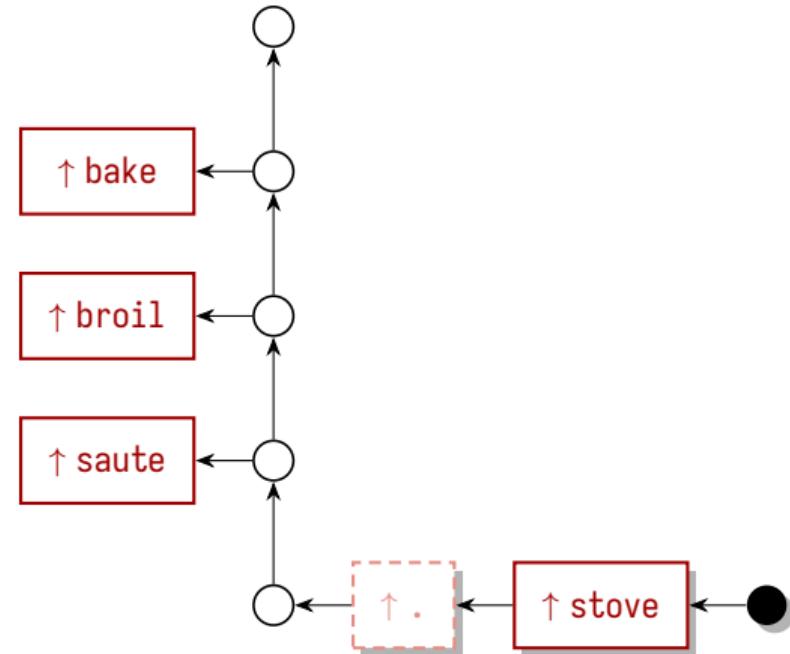


Stack graphs

```
stove.py
def bake():
    pass

def broil():
    pass

def saute():
    pass
```

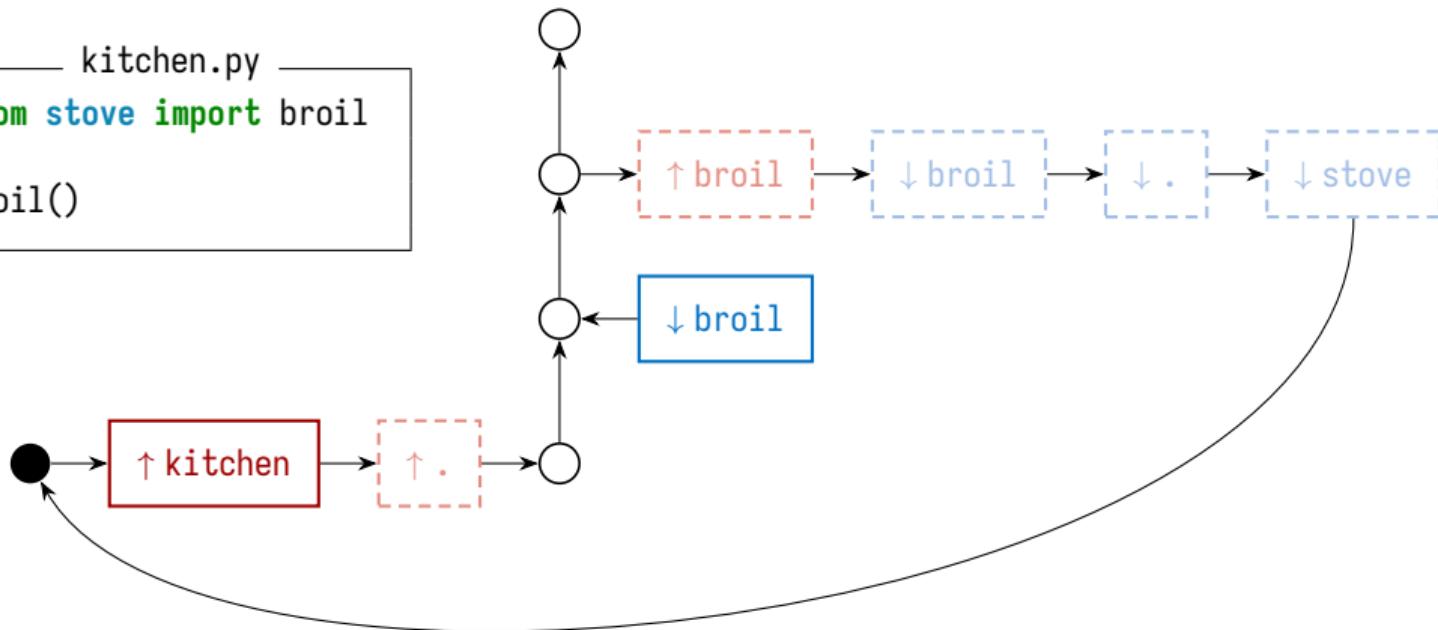


Stack graphs

```
----- kitchen.py -----  
from stove import broil  
  
broil()
```

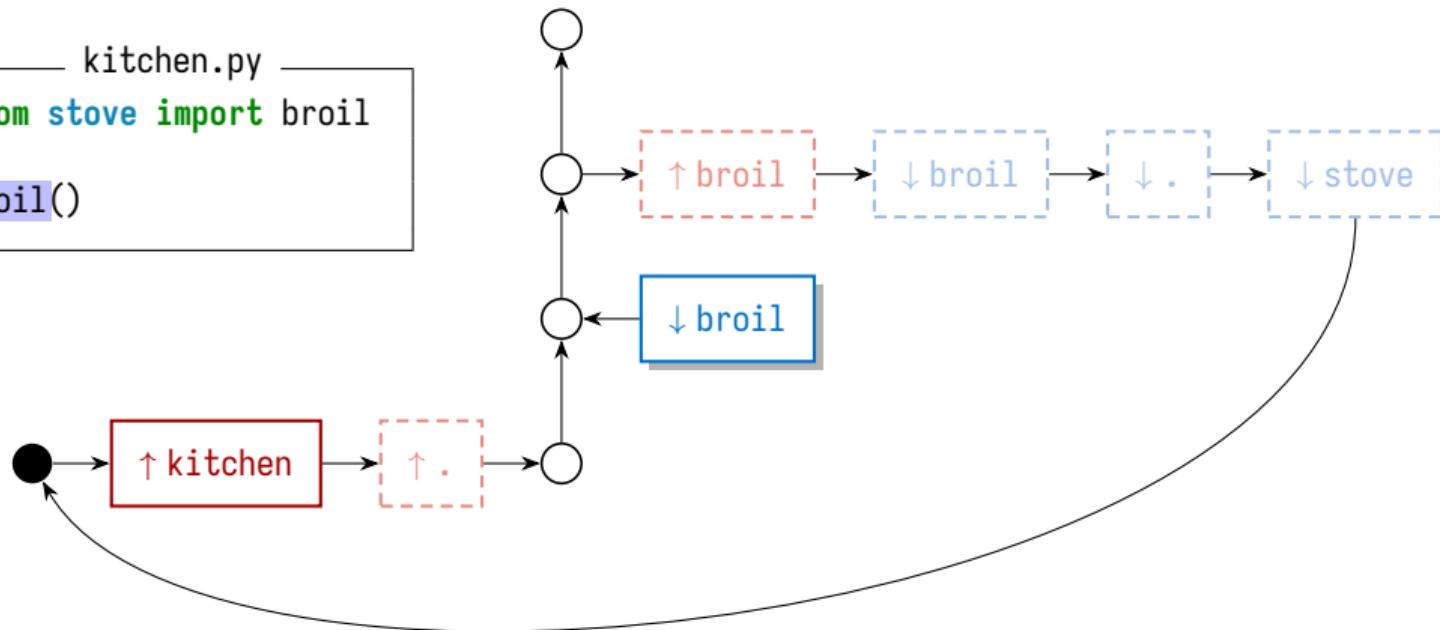
Stack graphs

```
kitchen.py  
from stove import broil  
  
broil()
```

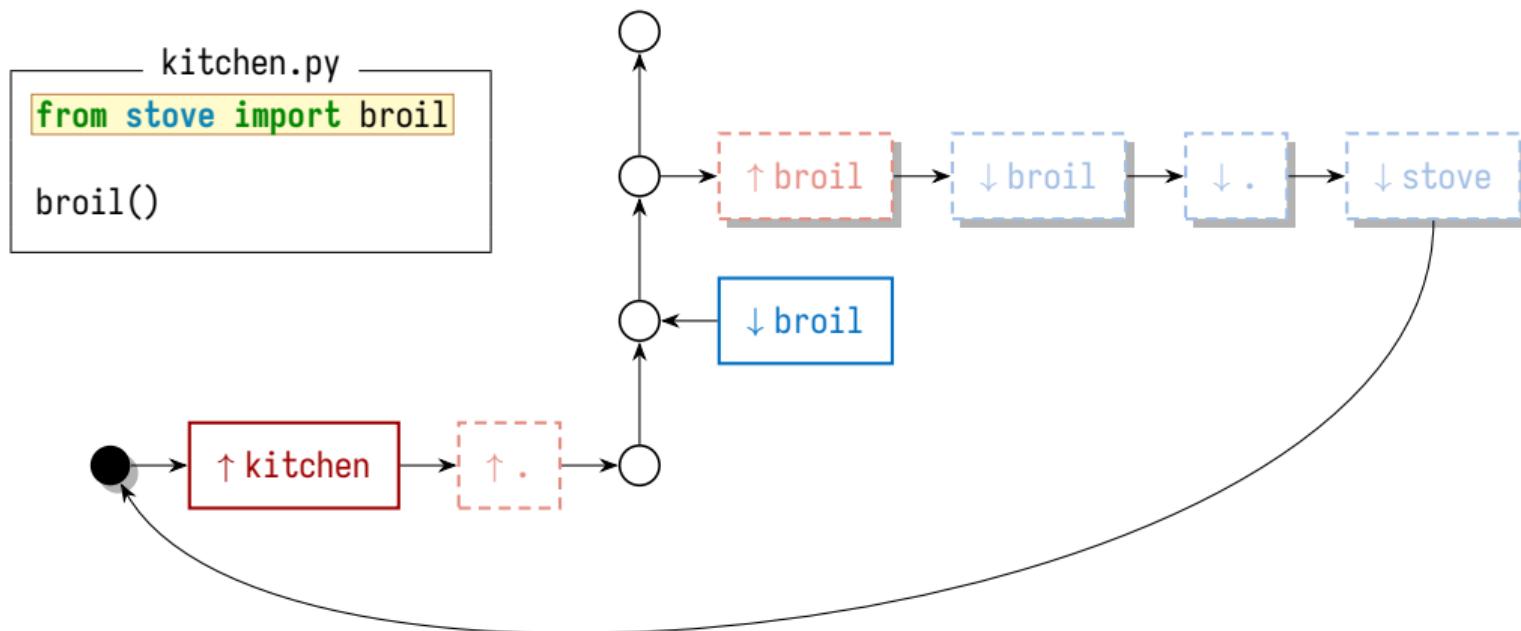


Stack graphs

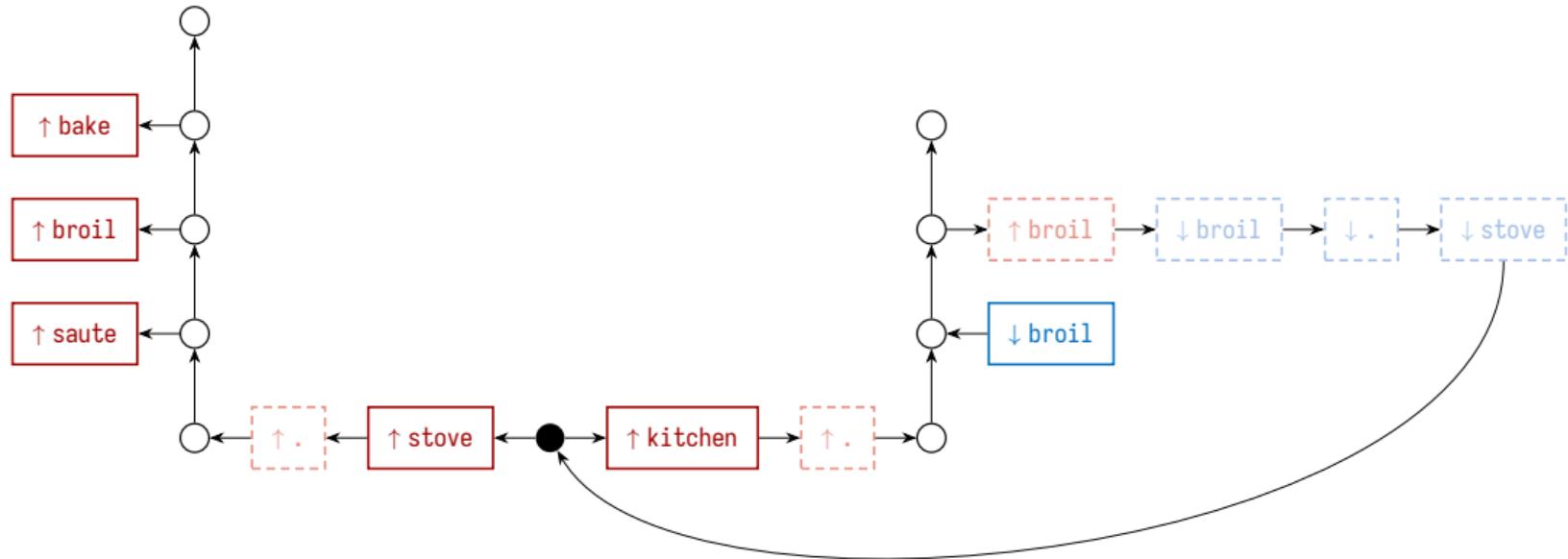
```
kitchen.py  
from stove import broil  
  
broil()
```



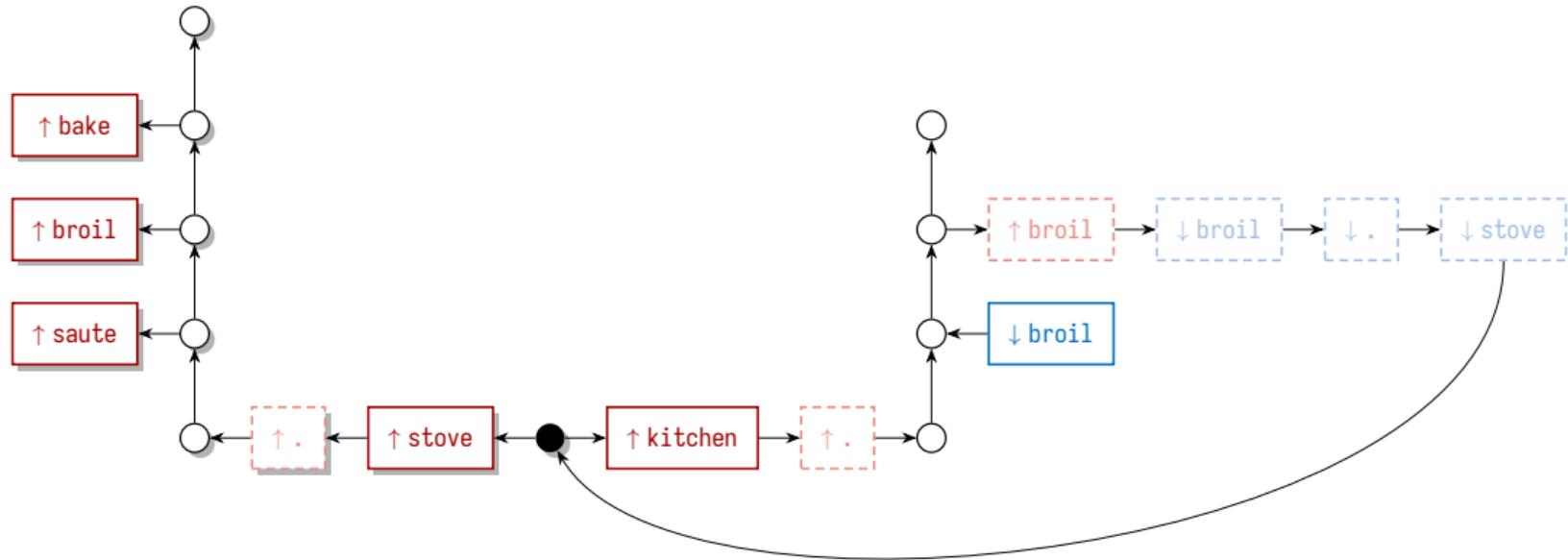
Stack graphs



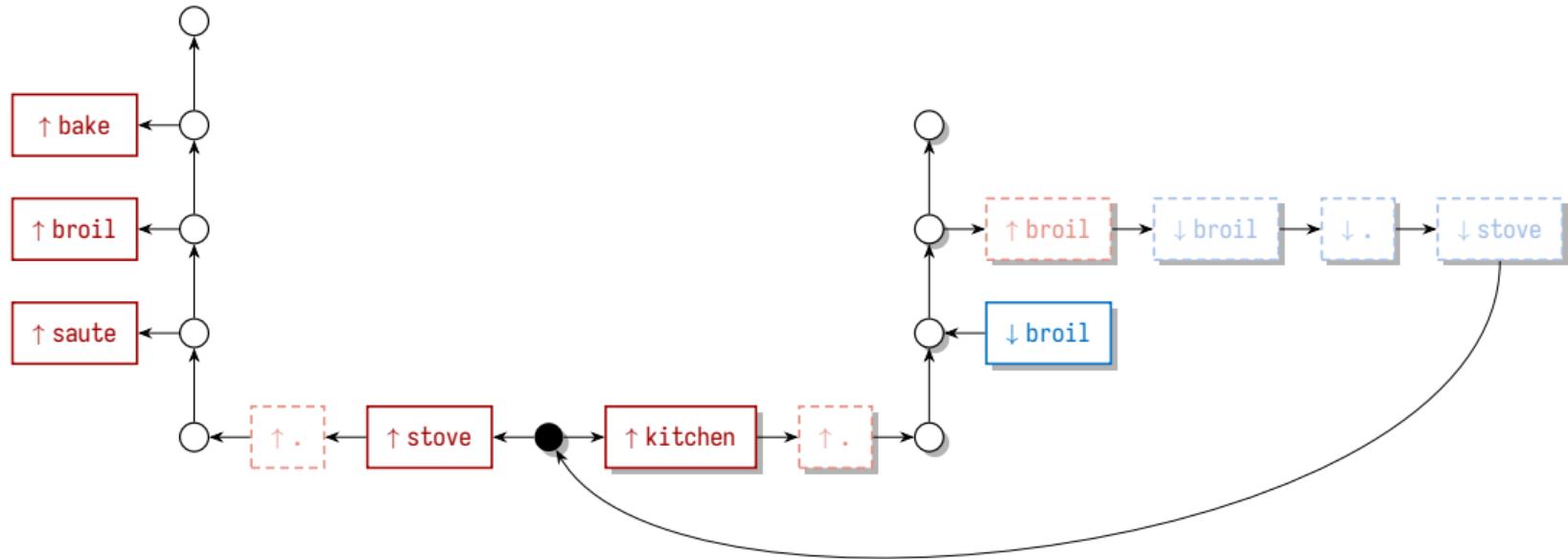
Stack graphs



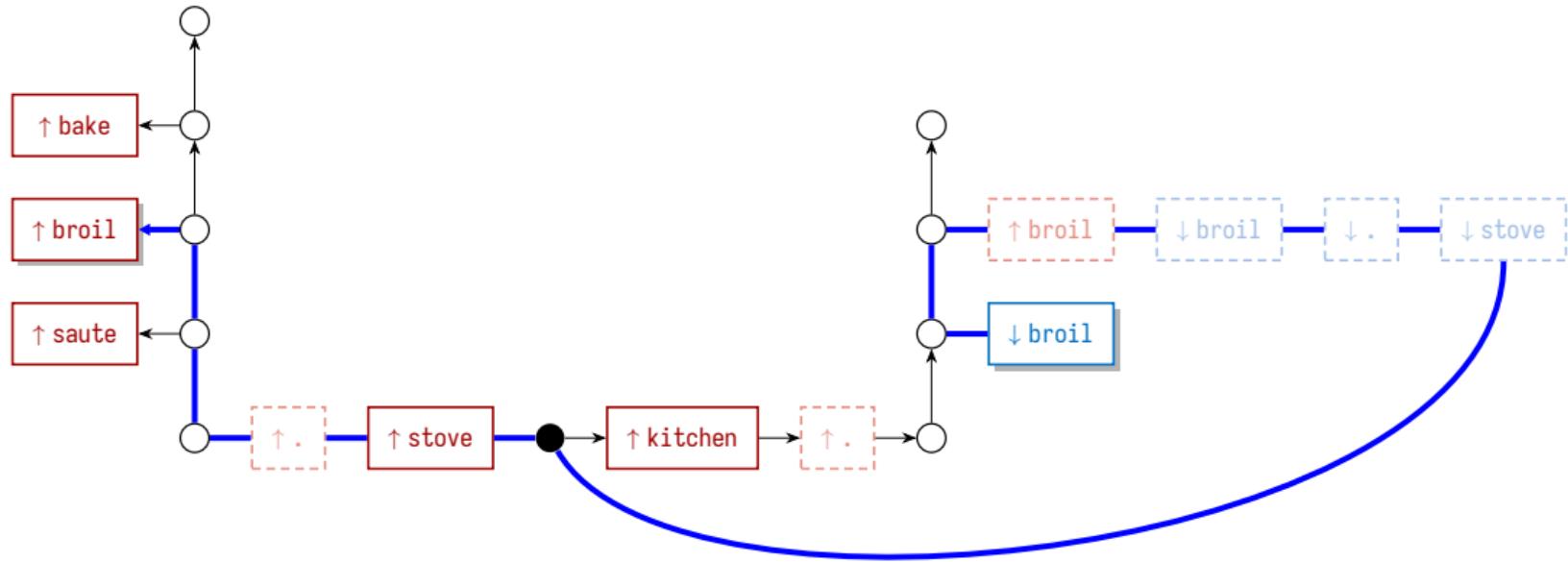
Stack graphs



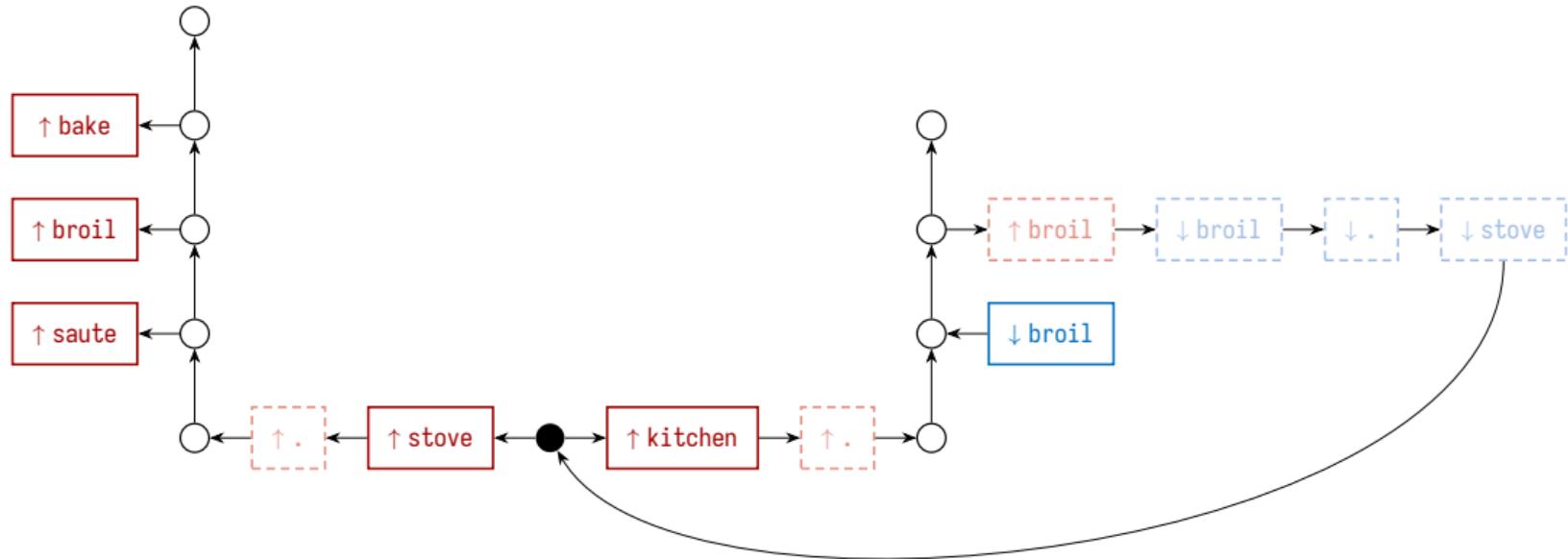
Stack graphs



Stack graphs

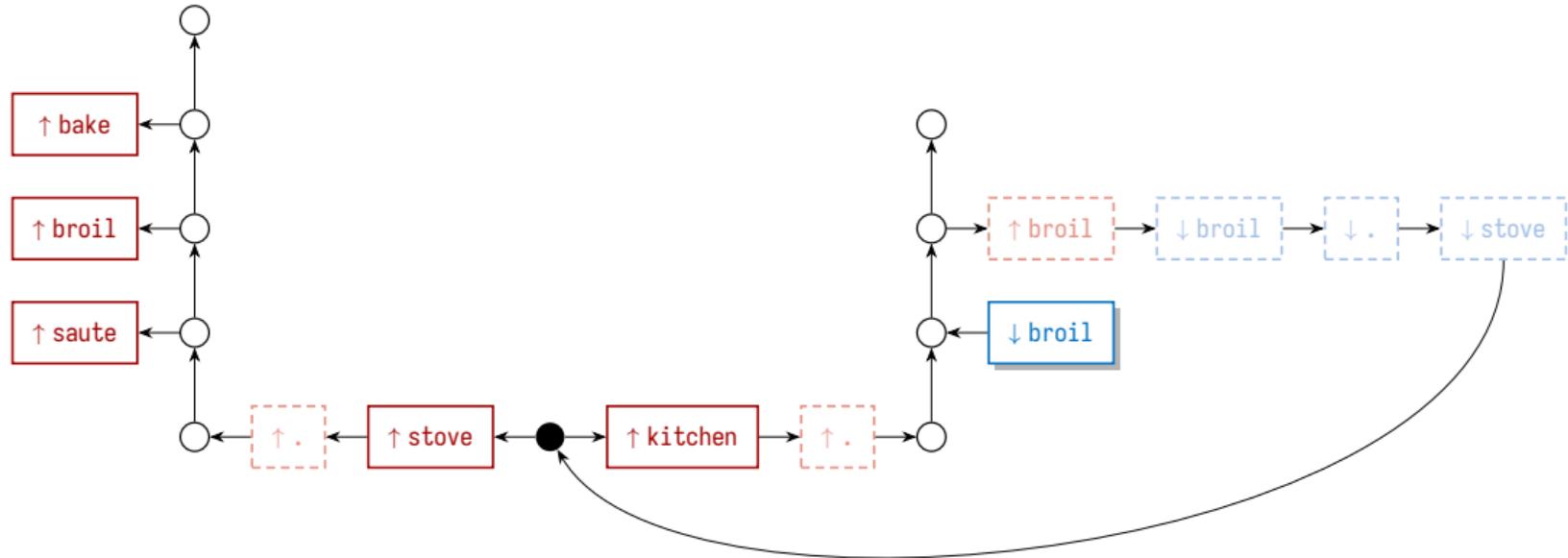


Stack graphs



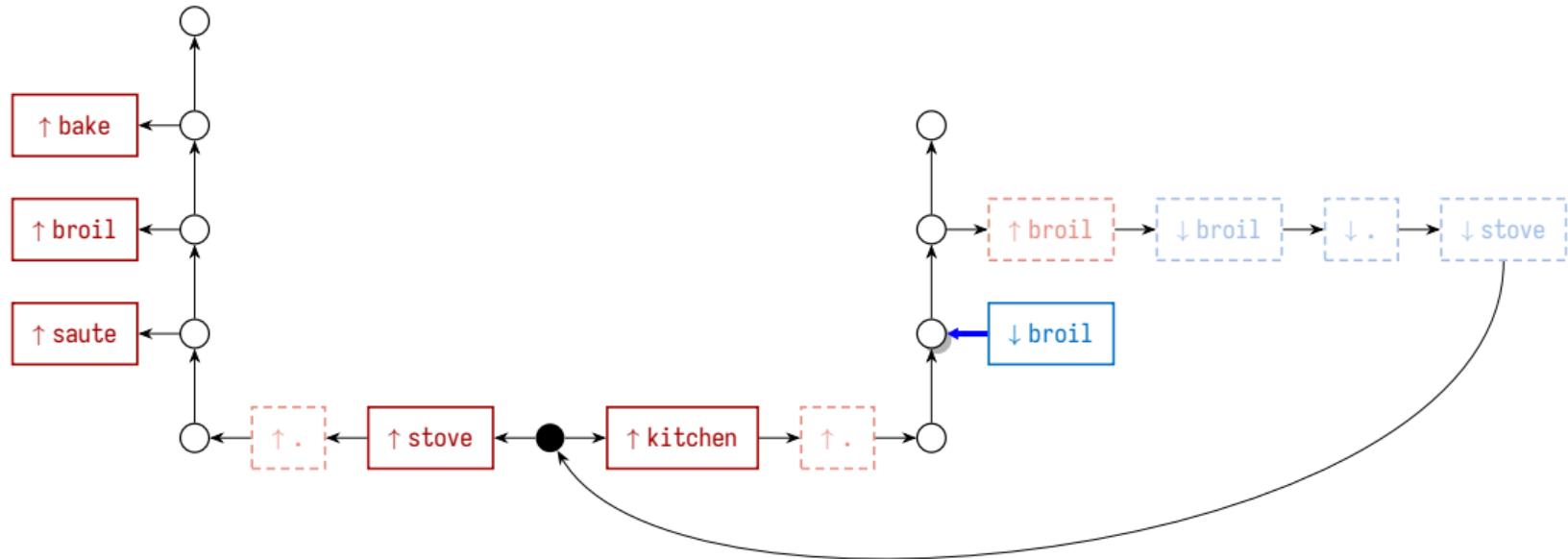
Symbol stack: ◇

Stack graphs



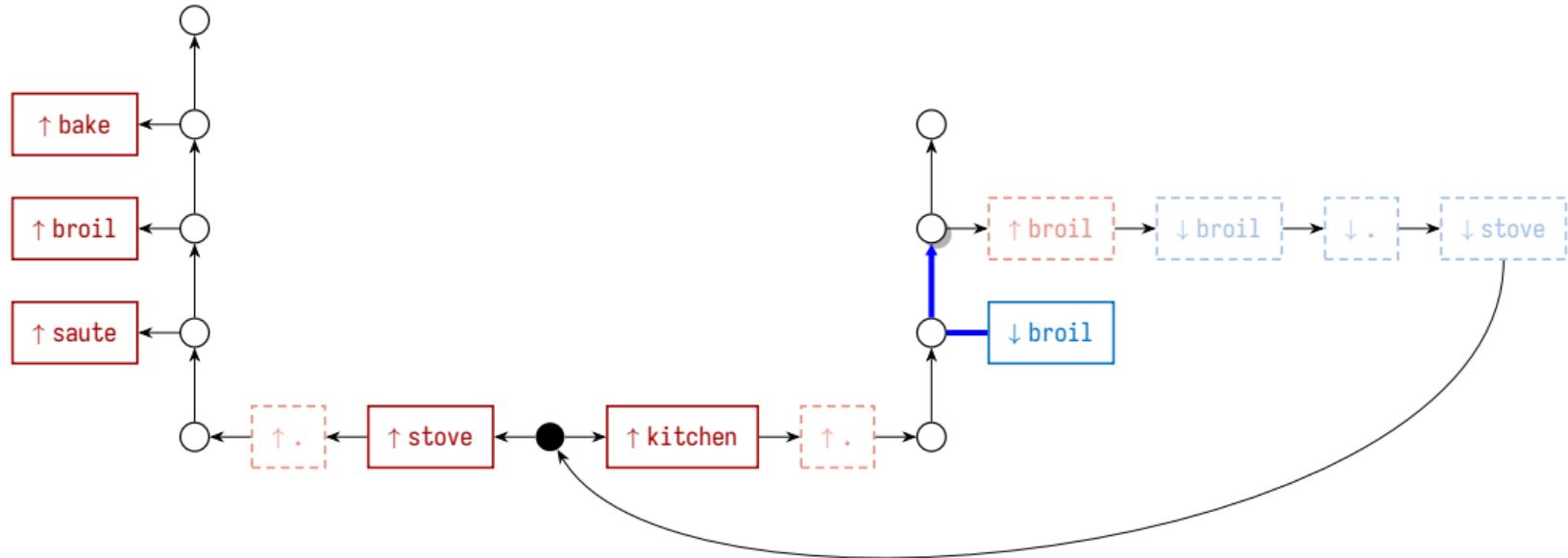
Symbol stack:
⟨broil⟩

Stack graphs



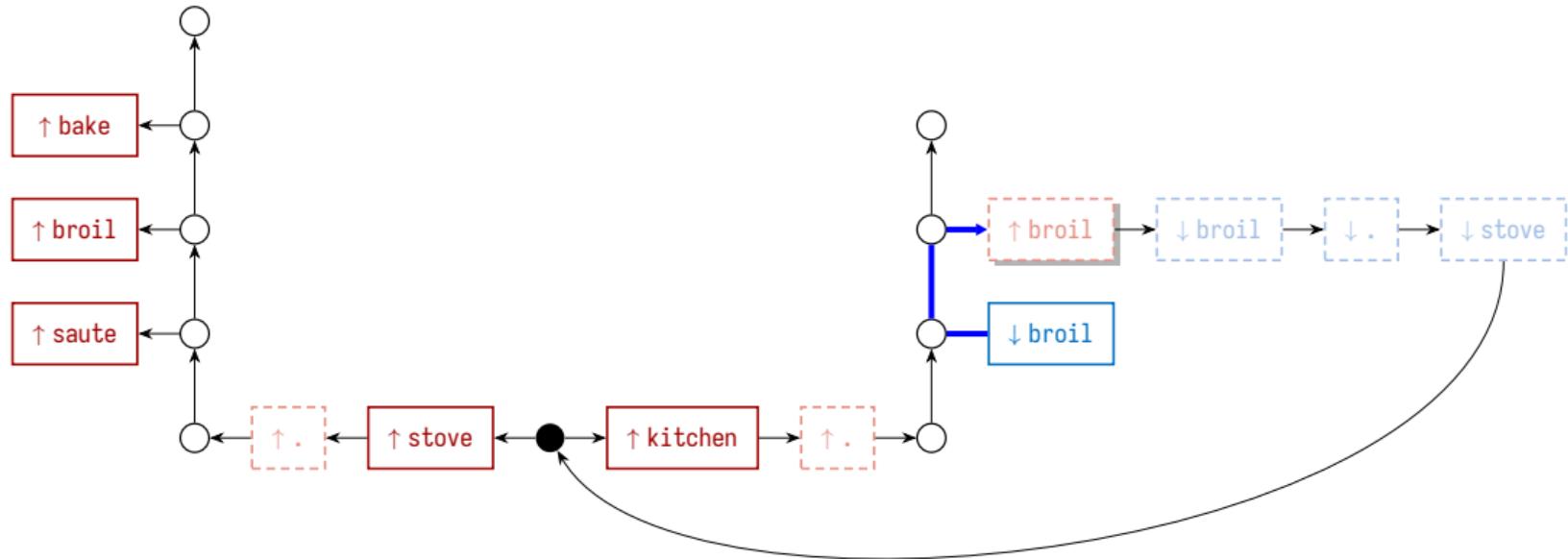
Symbol stack:
⟨broil⟩

Stack graphs



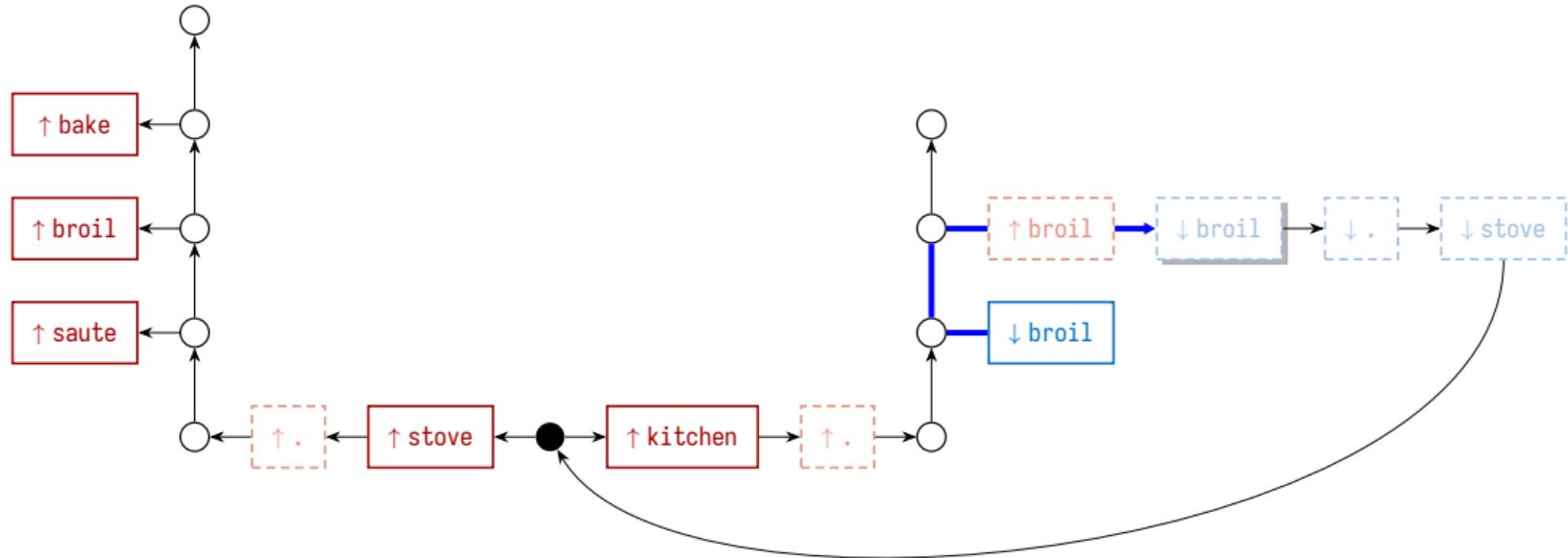
Symbol stack:
 $\langle \text{broil} \rangle$

Stack graphs



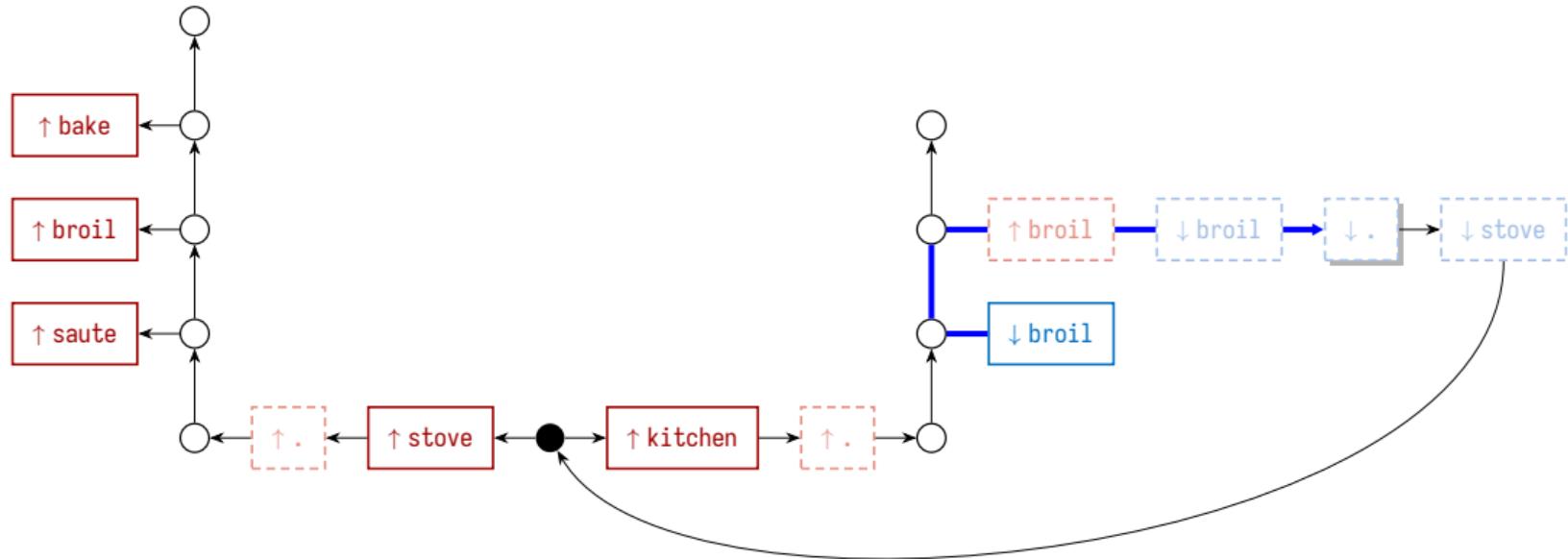
Symbol stack: ◇

Stack graphs



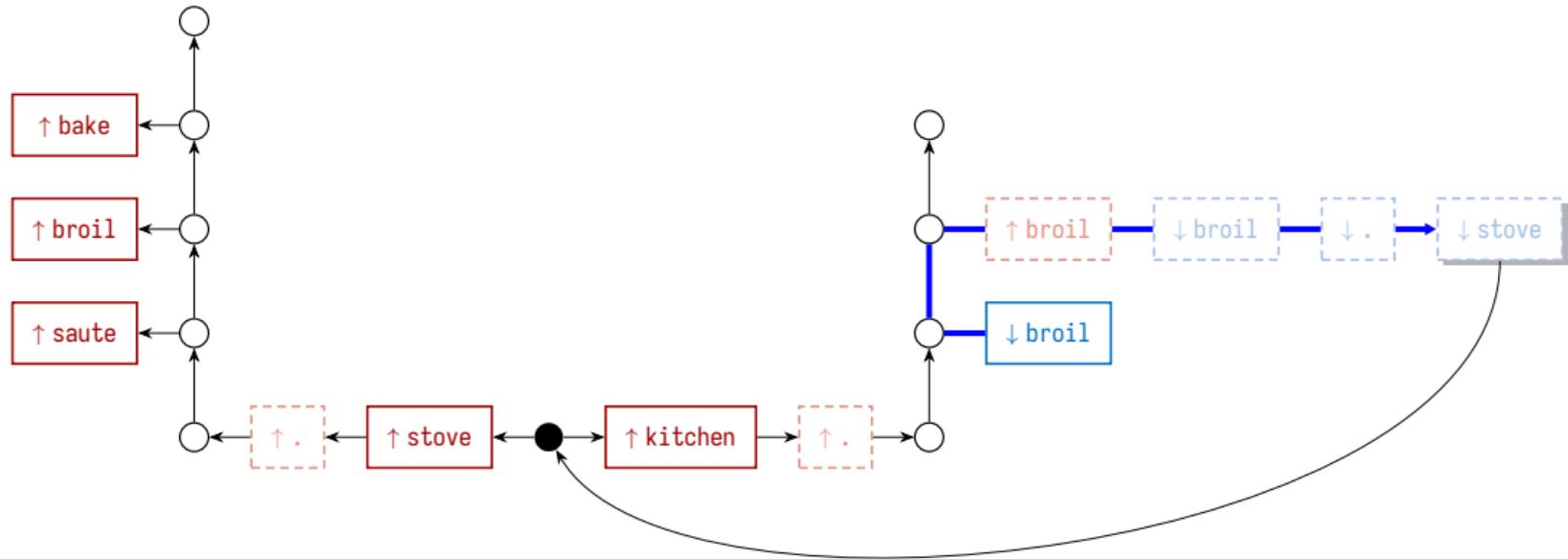
Symbol stack:
 $\langle \text{broil} \rangle$

Stack graphs

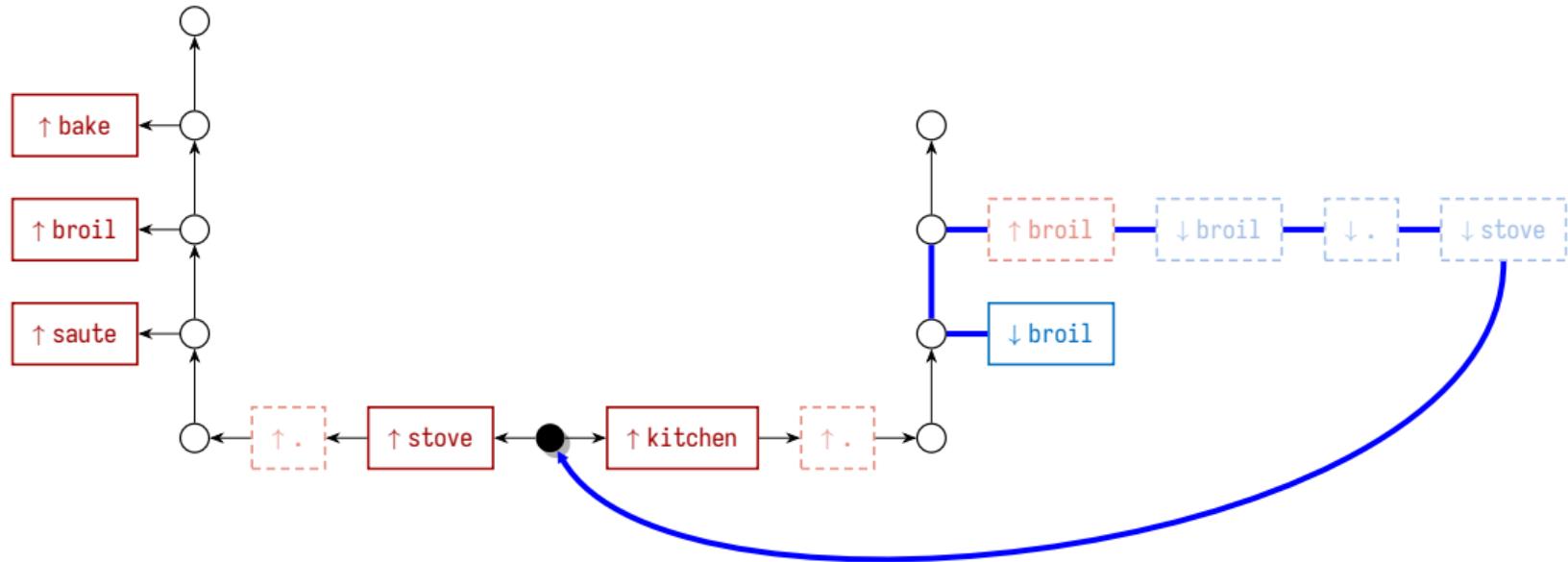


Symbol stack: $\langle \text{.broil} \rangle$

Stack graphs

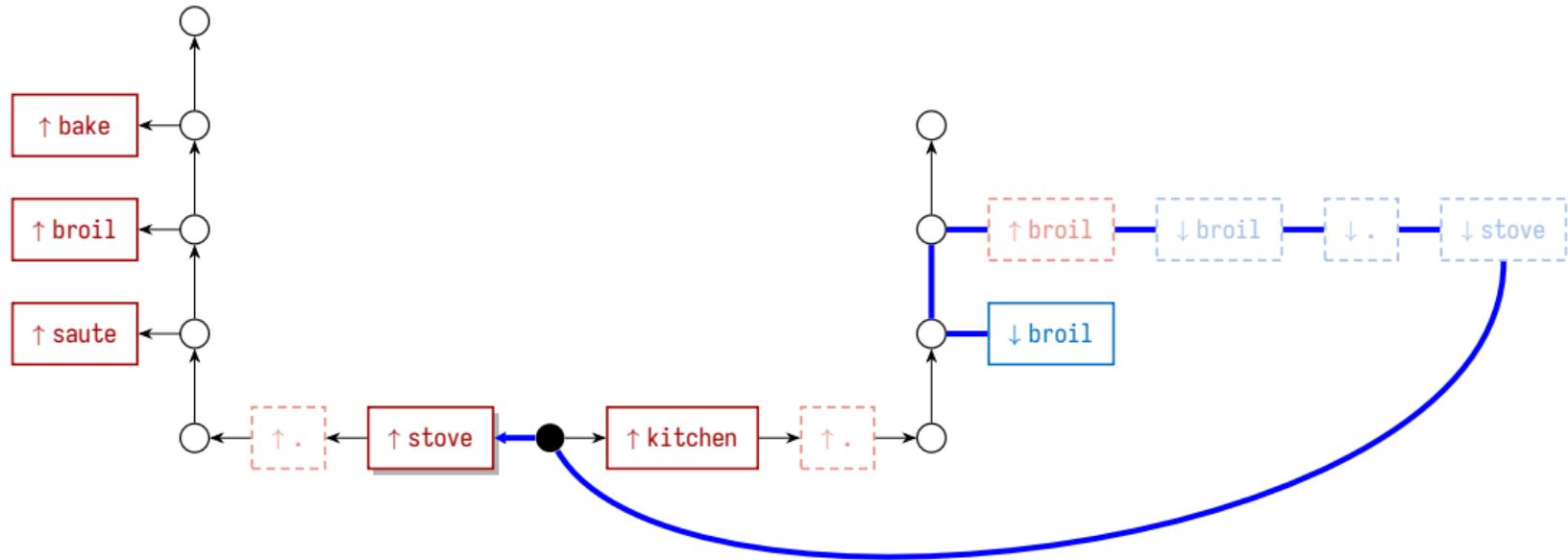


Stack graphs



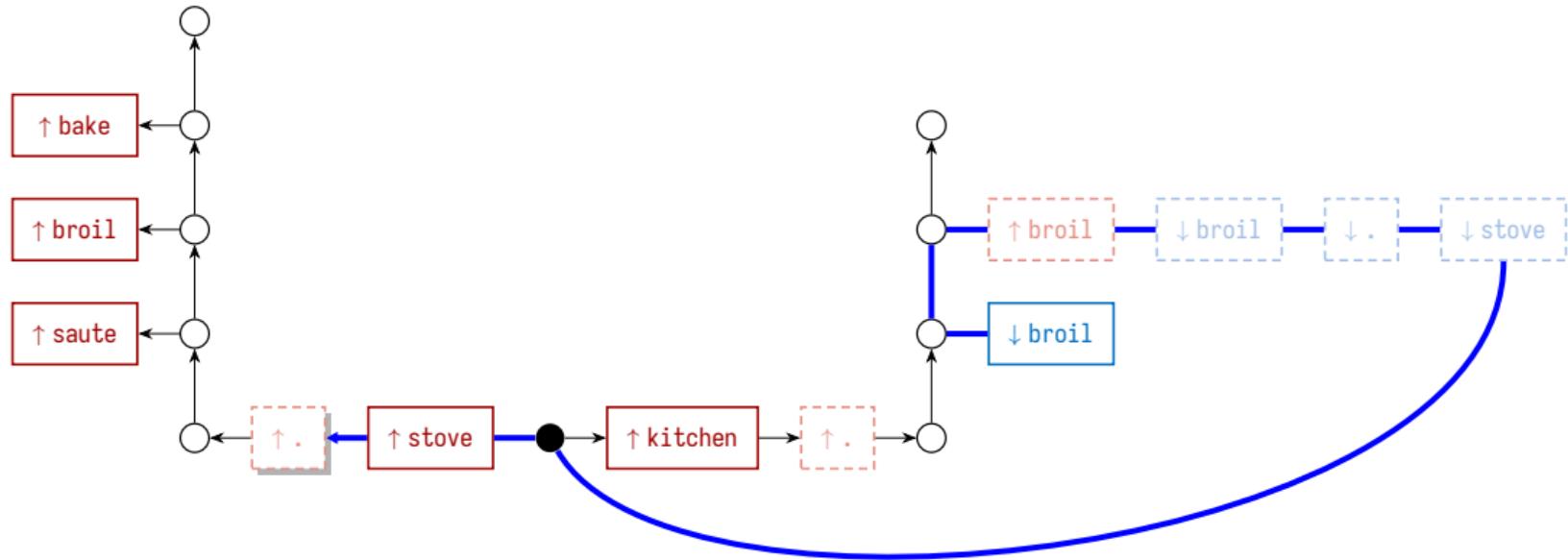
Symbol stack: $\langle \text{stove.broil} \rangle$

Stack graphs



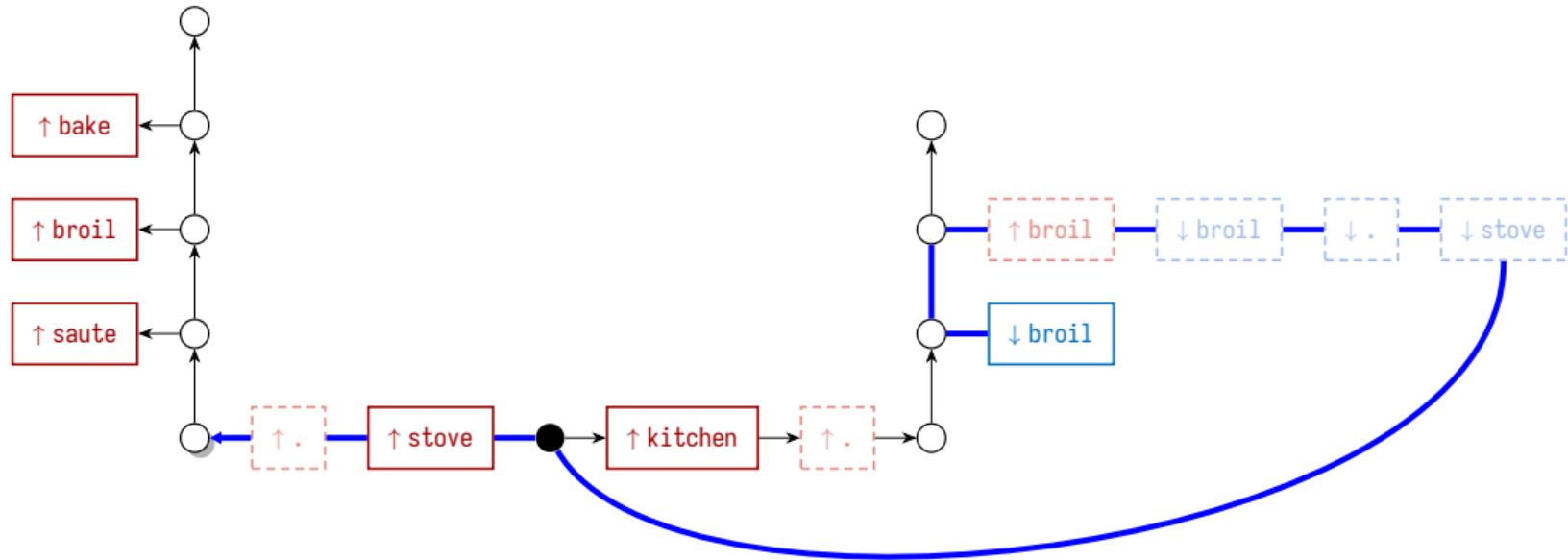
Symbol stack: $\langle .\text{broil} \rangle$

Stack graphs



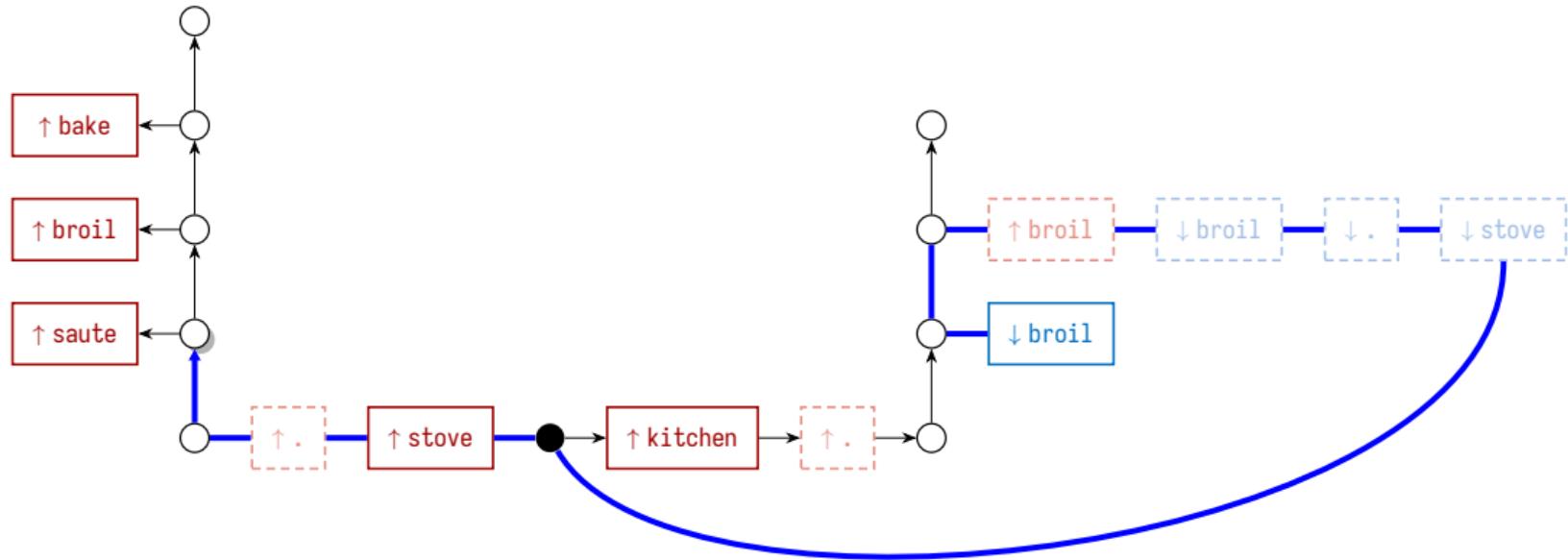
Symbol stack:
 $\langle \text{broil} \rangle$

Stack graphs



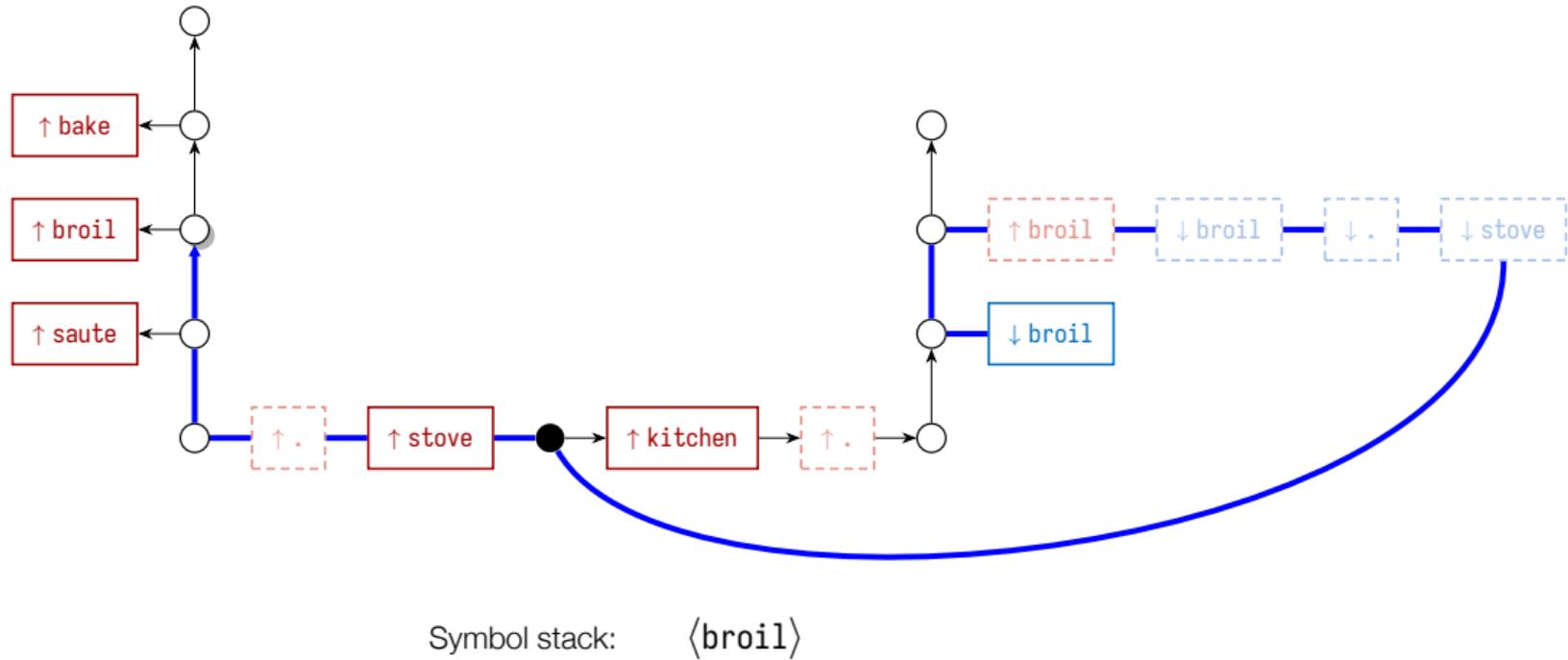
Symbol stack: $\langle \text{broil} \rangle$

Stack graphs

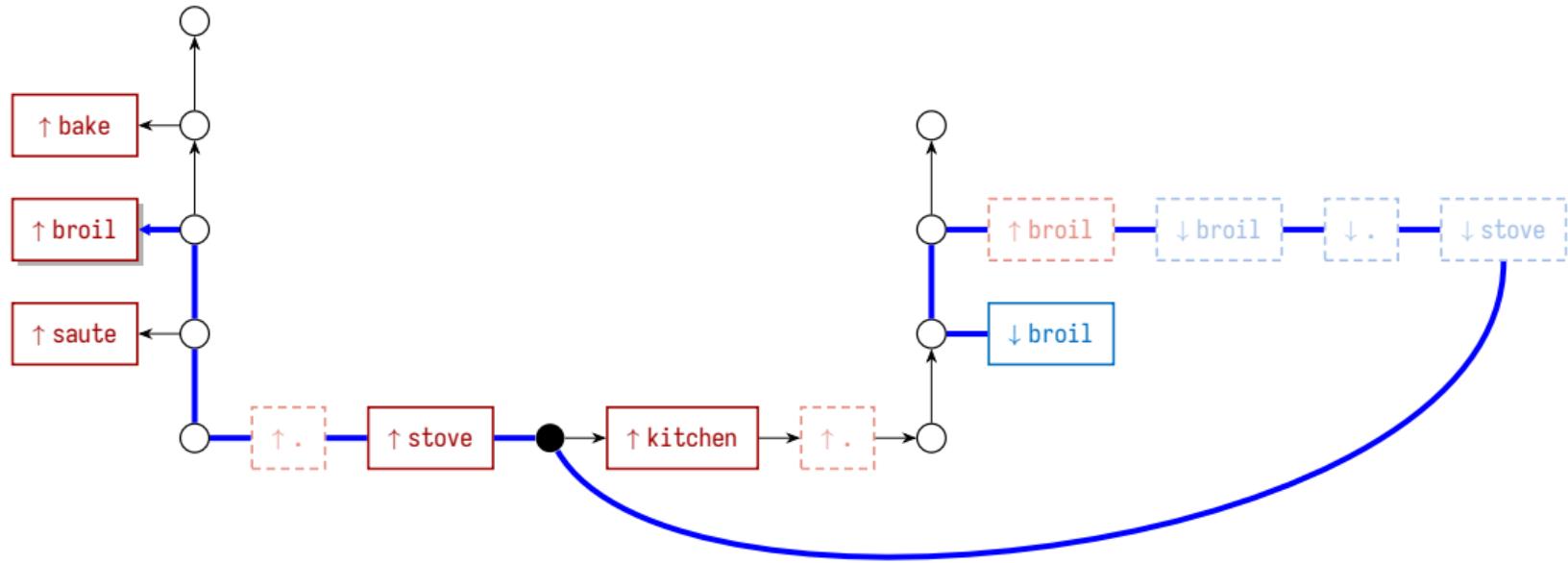


Symbol stack: $\langle \text{broil} \rangle$

Stack graphs

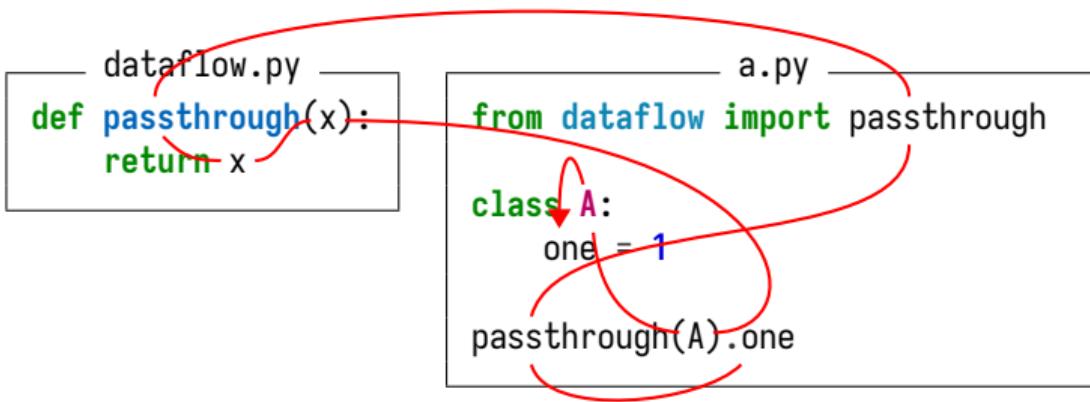


Stack graphs

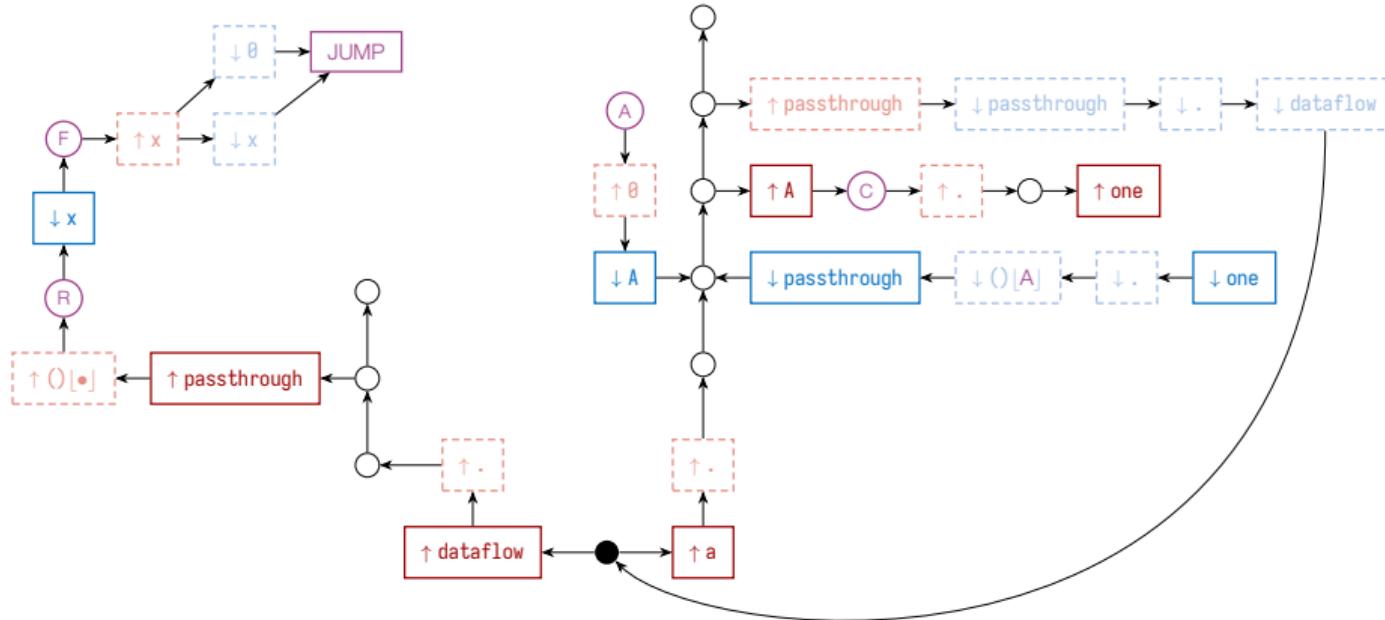


Symbol stack: ◇

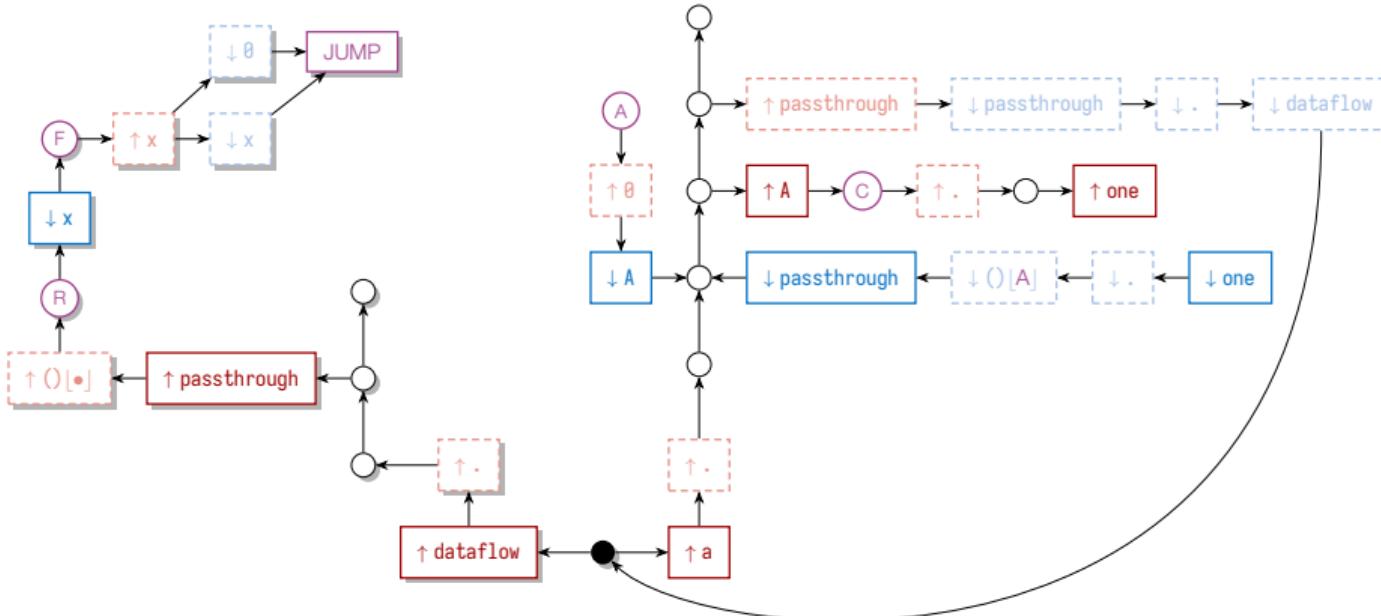
The dataflow example



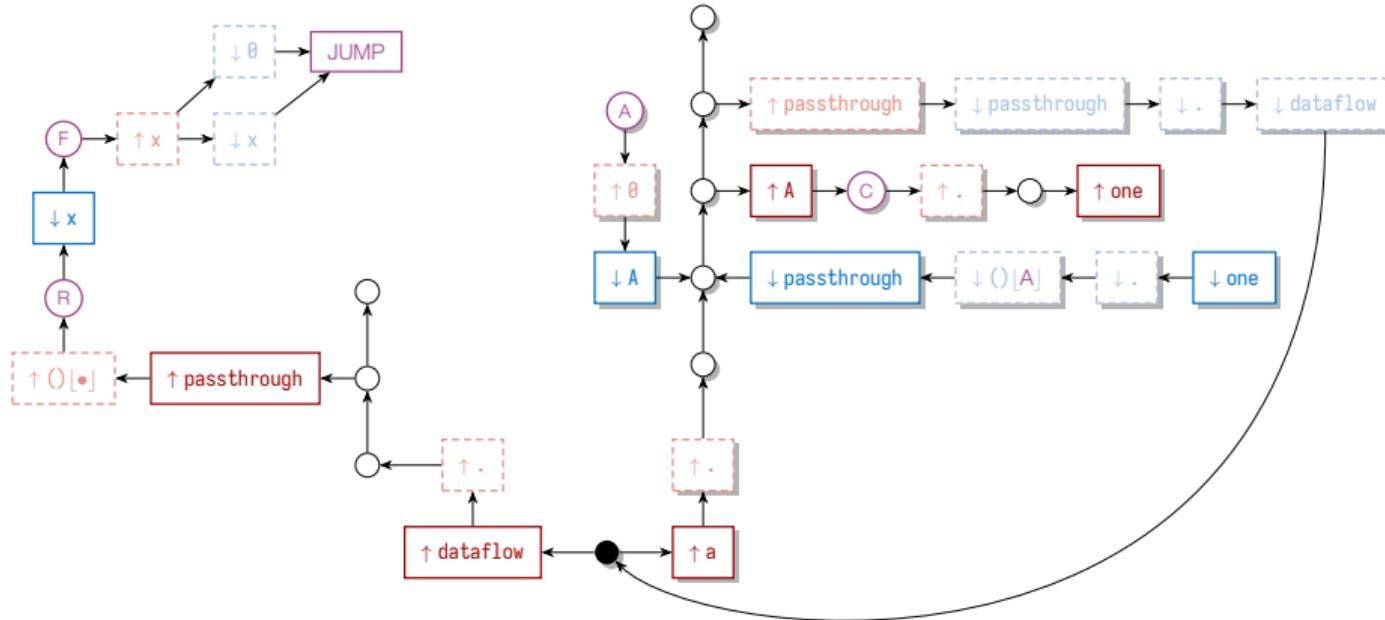
The dataflow example



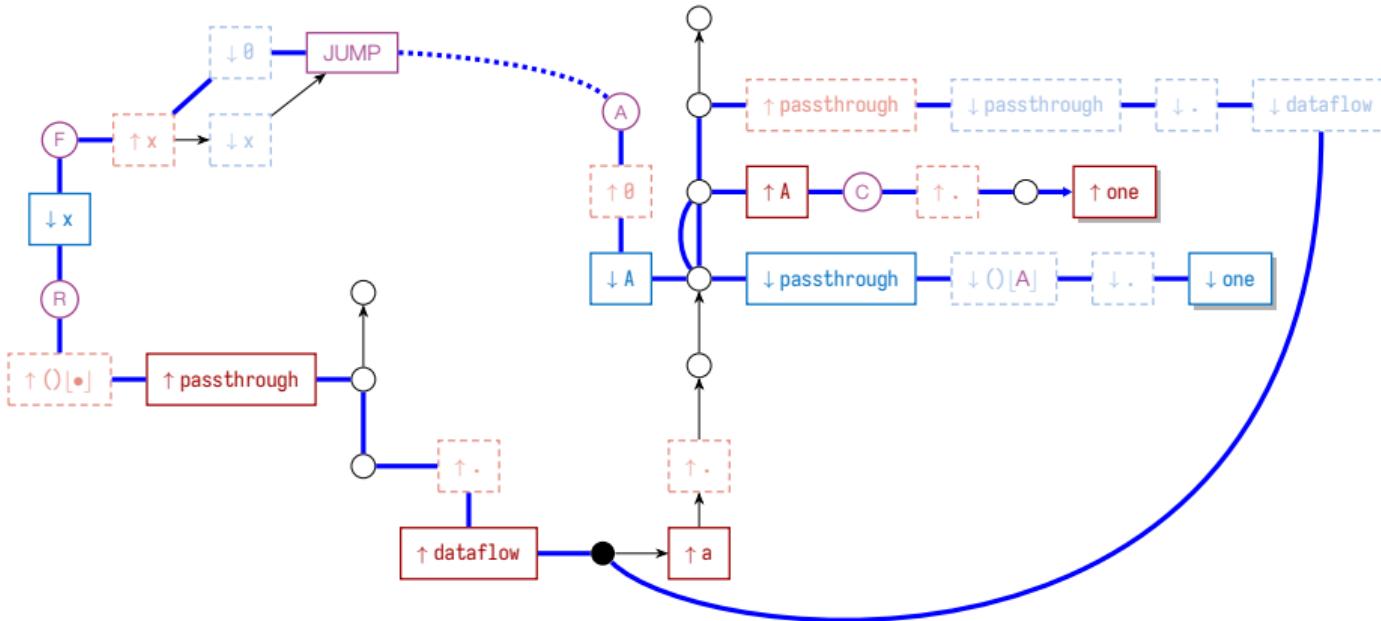
The dataflow example



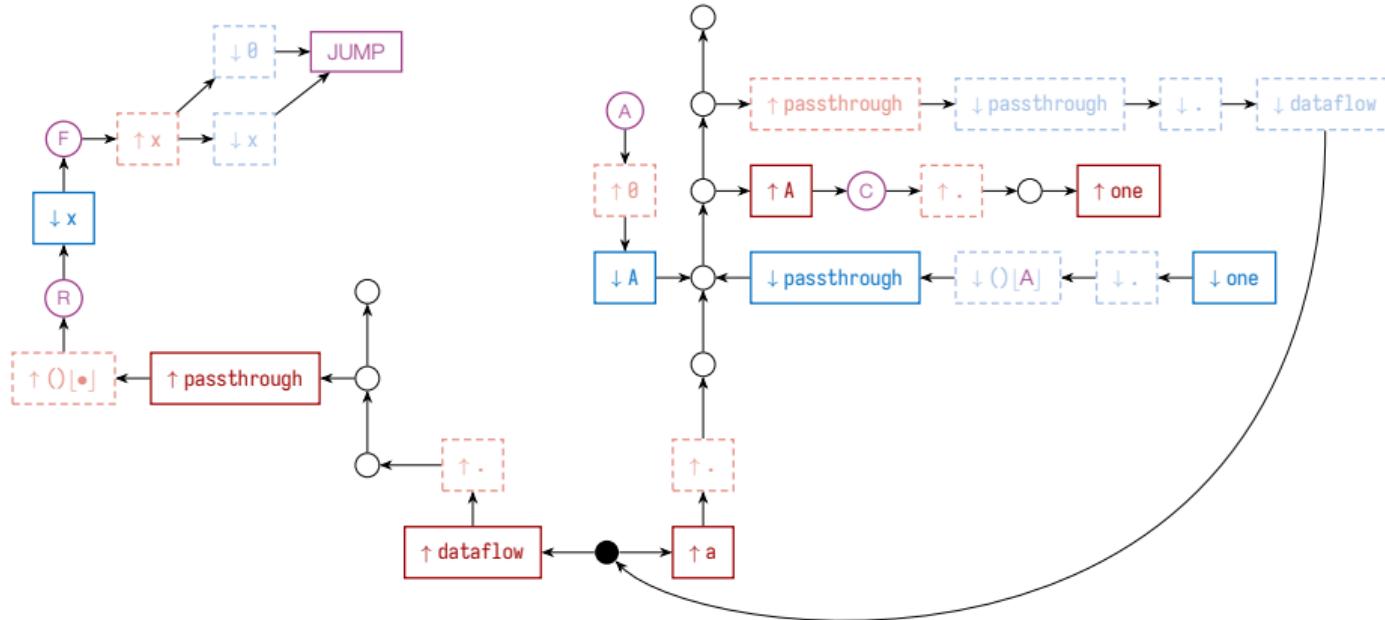
The dataflow example



The dataflow example



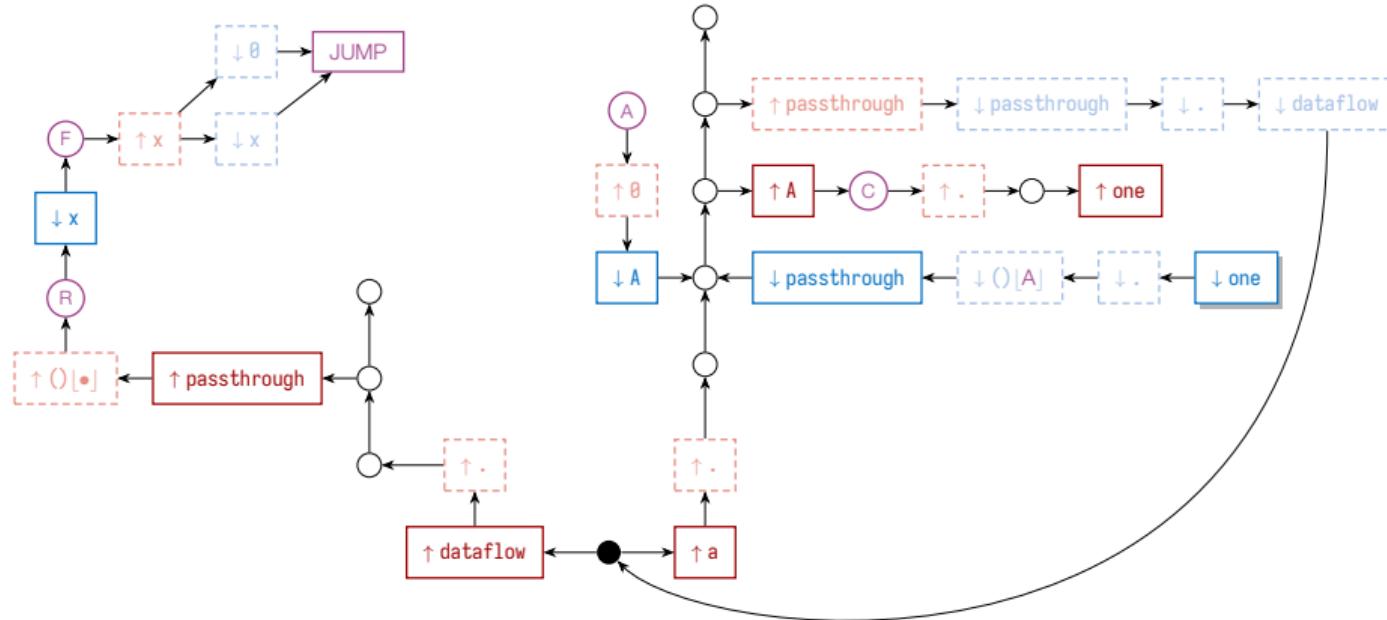
The dataflow example



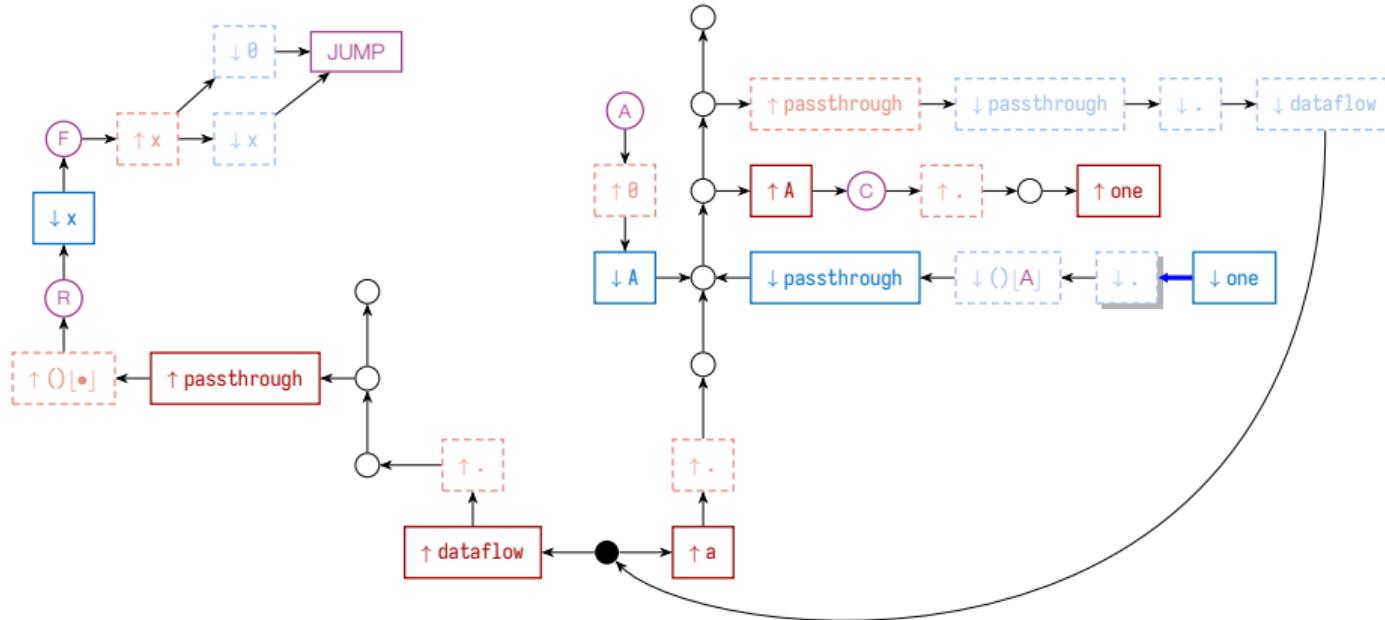
Symbol stack: ◇

Scope stack: ○

The dataflow example



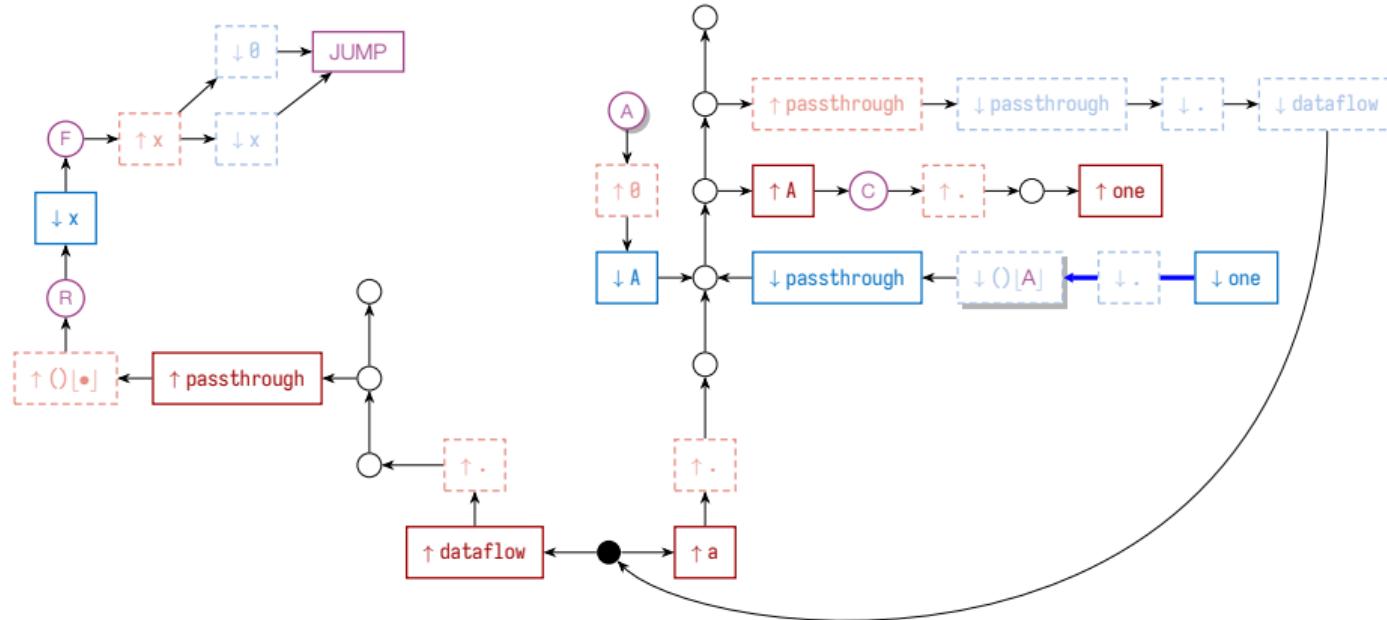
The dataflow example



Symbol stack: $\langle .\text{one} \rangle$

Scope stack: \circ

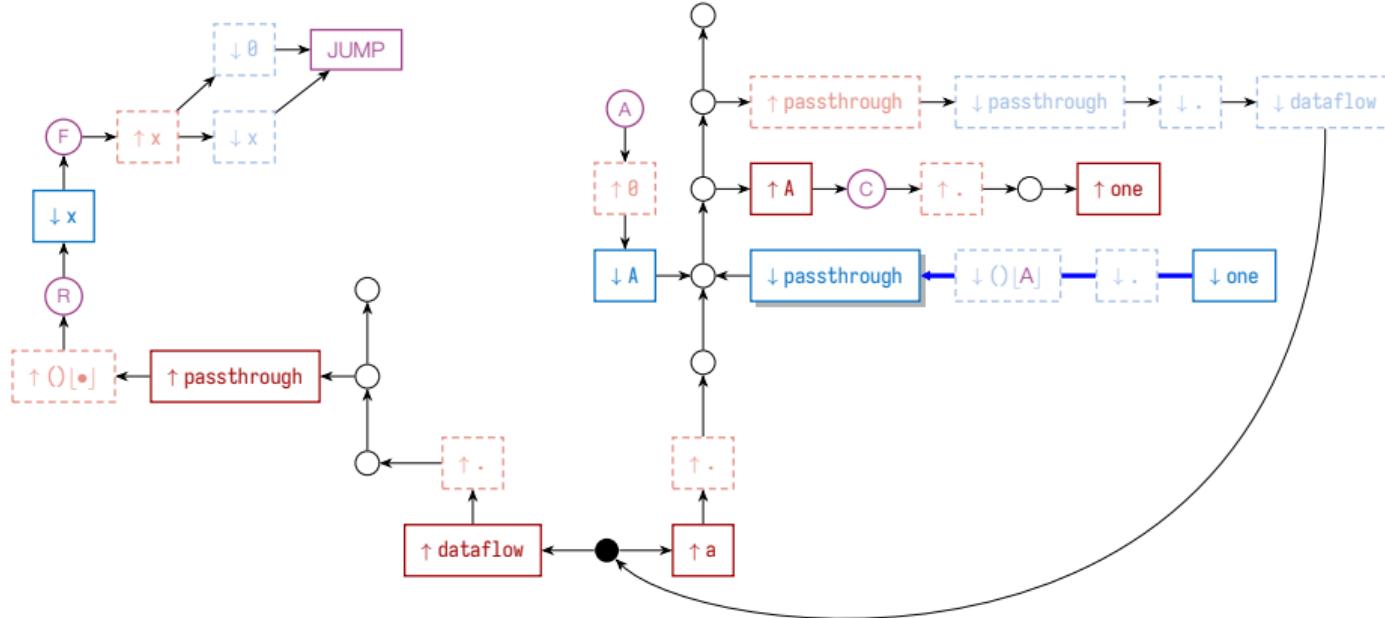
The dataflow example



Symbol stack: $\langle ()[A].\text{one} \rangle$

Scope stack: \circ

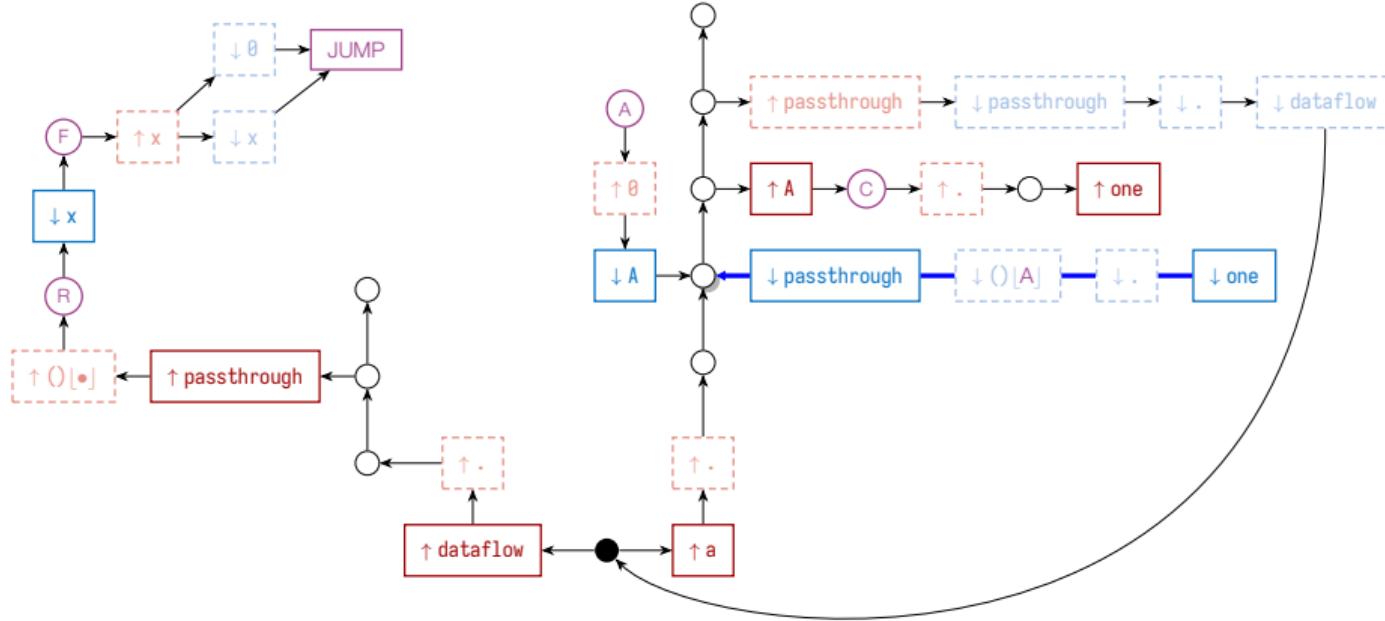
The dataflow example



Symbol stack: $\langle \text{passthrough}() | A]. \text{one} \rangle$

Scope stack: \circ

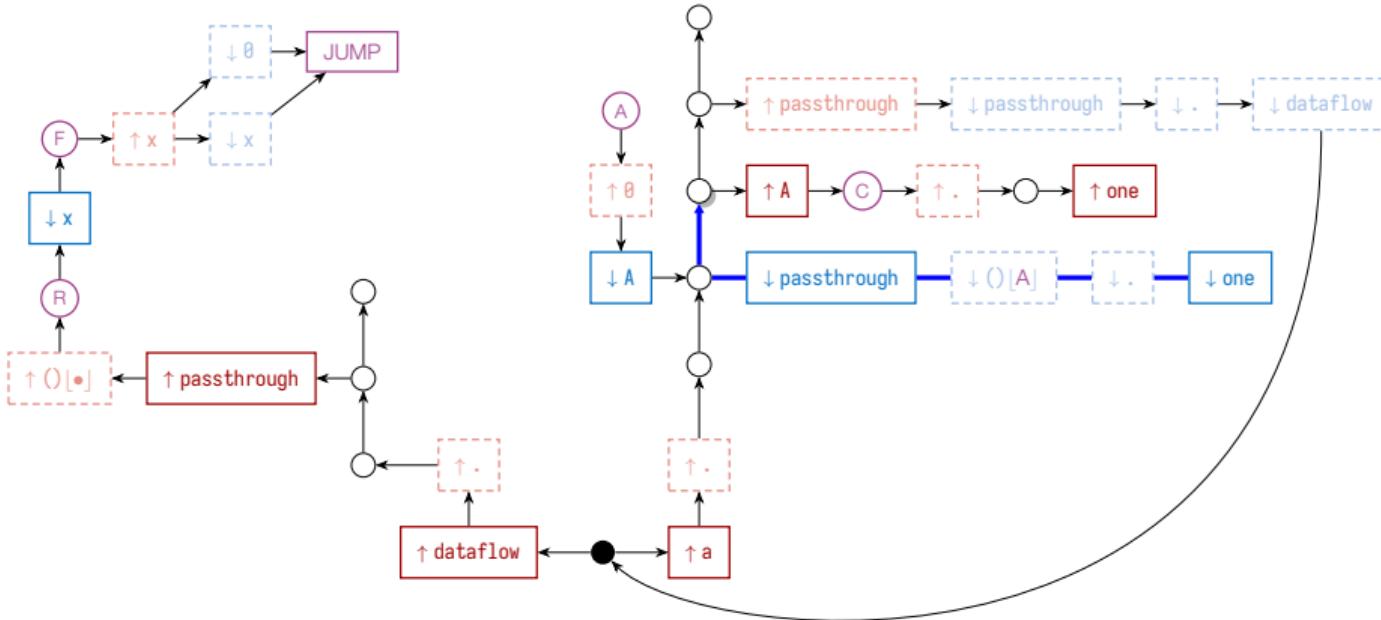
The dataflow example



Symbol stack: $\langle \text{passthrough}() | A]. \text{one} \rangle$

Scope stack: \circ

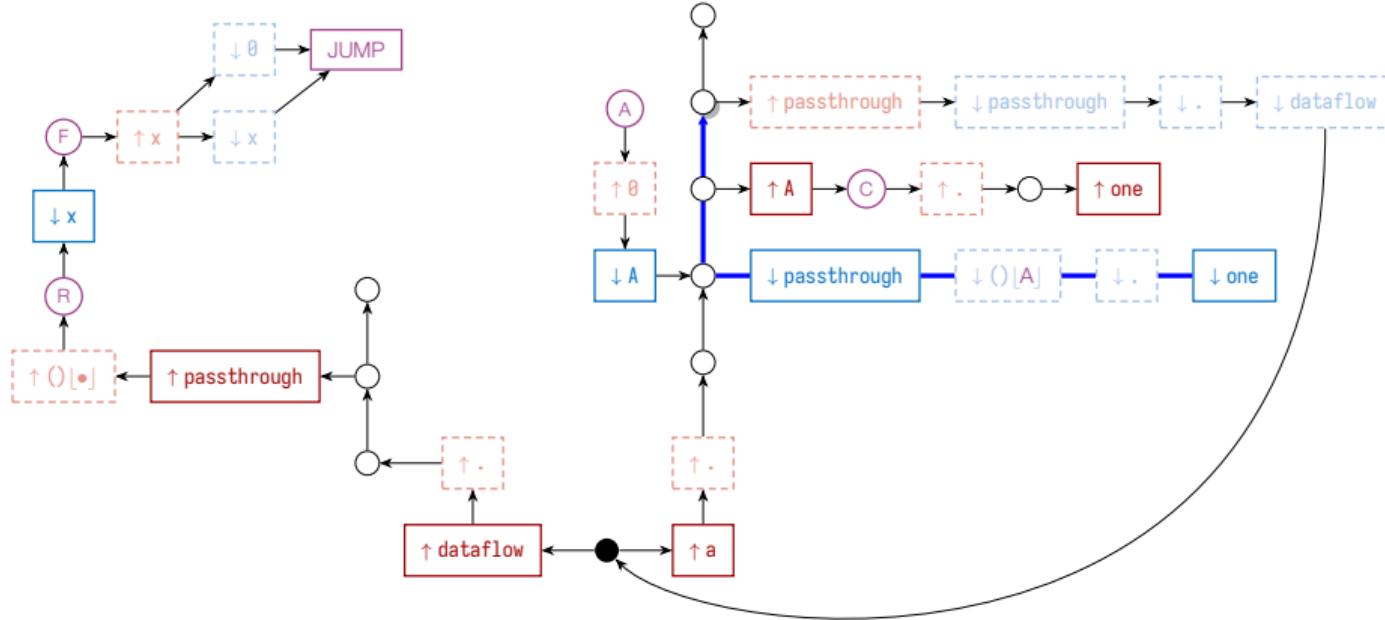
The dataflow example



Symbol stack: `⟨passthrough() [A].one⟩`

Scope stack: `c`

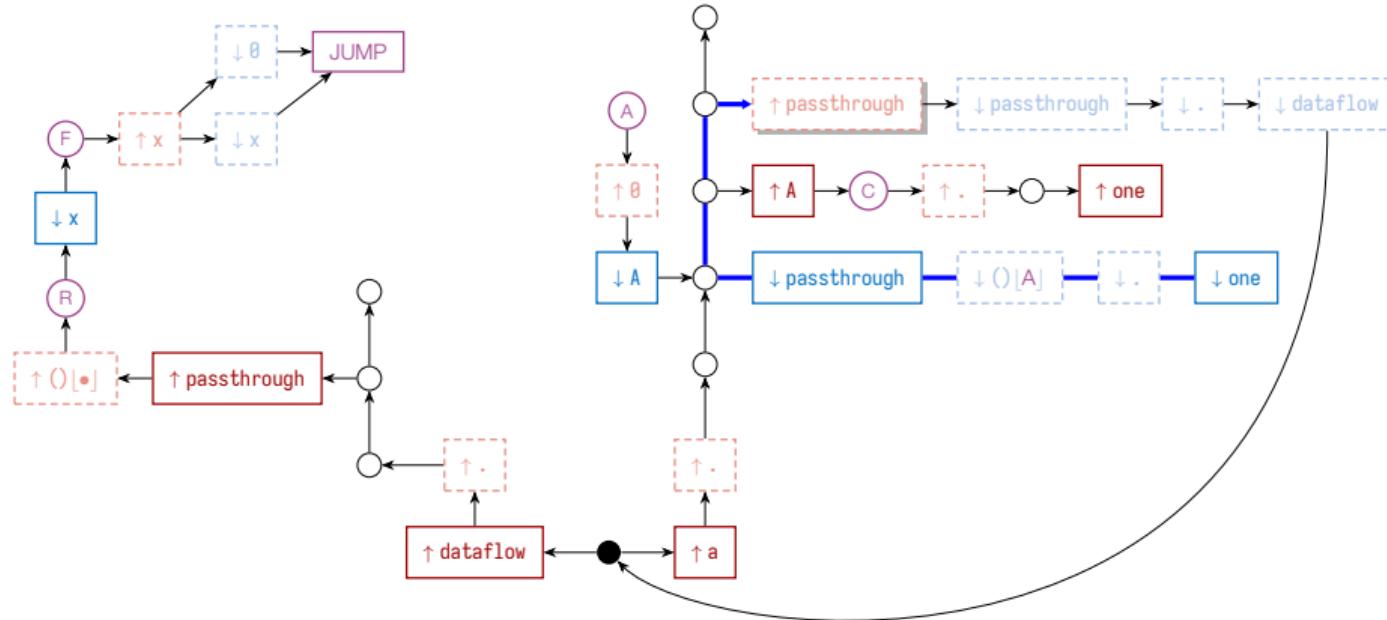
The dataflow example



Symbol stack: $\langle \text{passthrough}() | A \rangle . \text{one} \rangle$

Scope stack: \circ

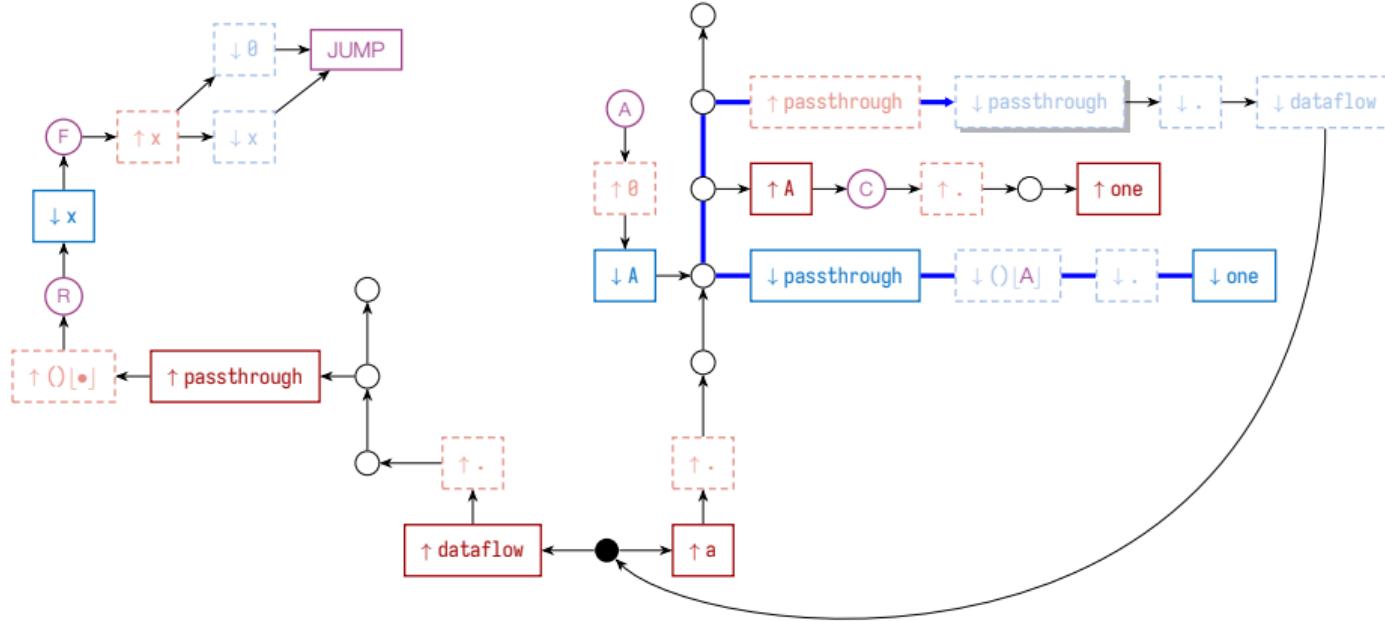
The dataflow example



Symbol stack: $\langle ()[A].\text{one} \rangle$

Scope stack: ○

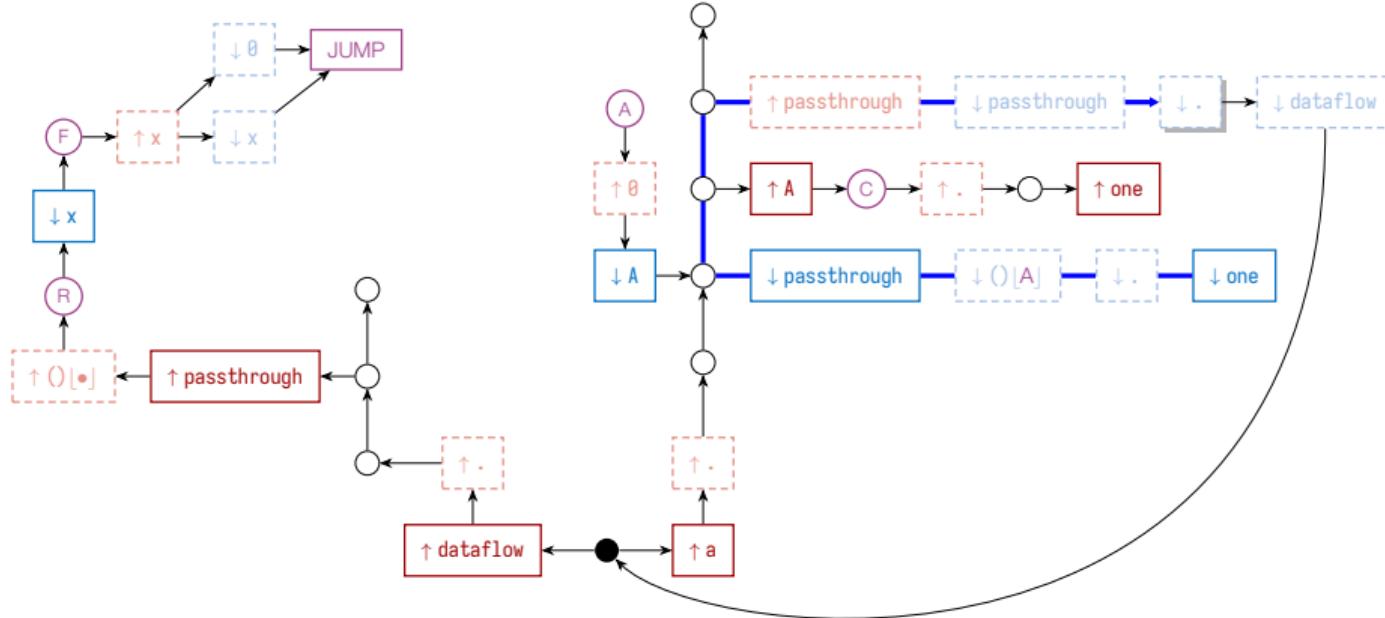
The dataflow example



Symbol stack: $\langle \text{passthrough}()[\text{A}] . \text{one} \rangle$

Scope stack: \circ

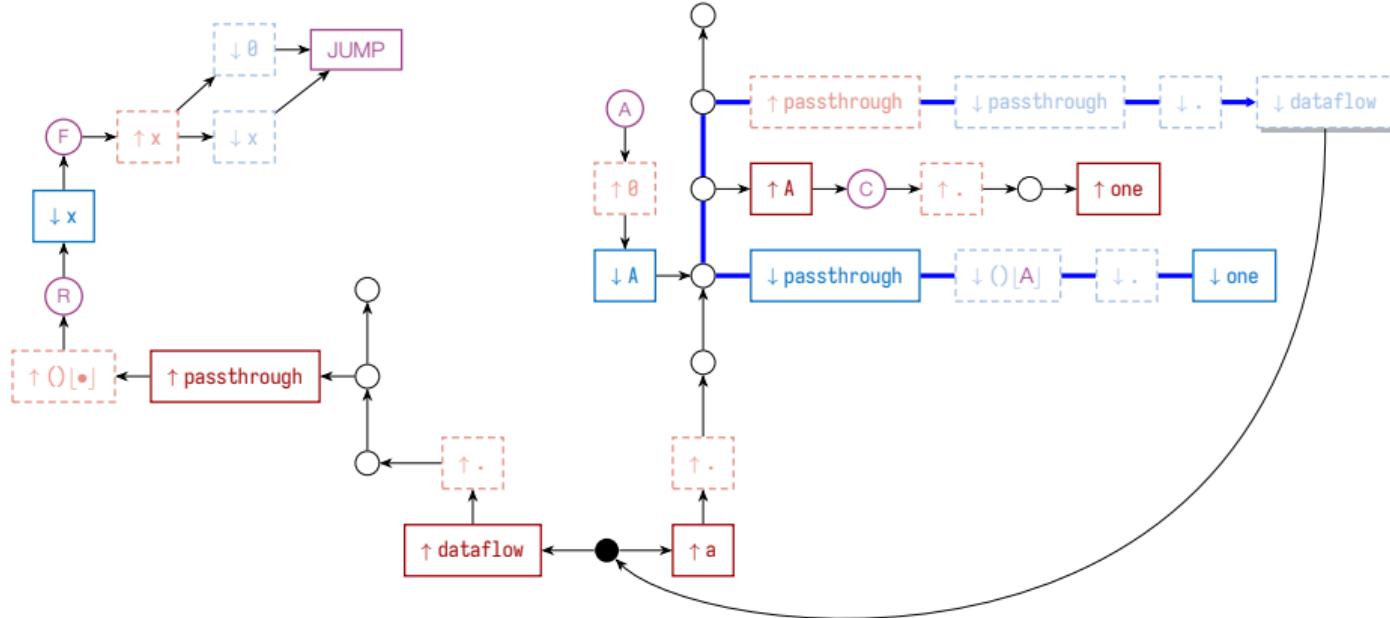
The dataflow example



Symbol stack: $\langle .passthrough()|A|.one \rangle$

Scope stack: \circ

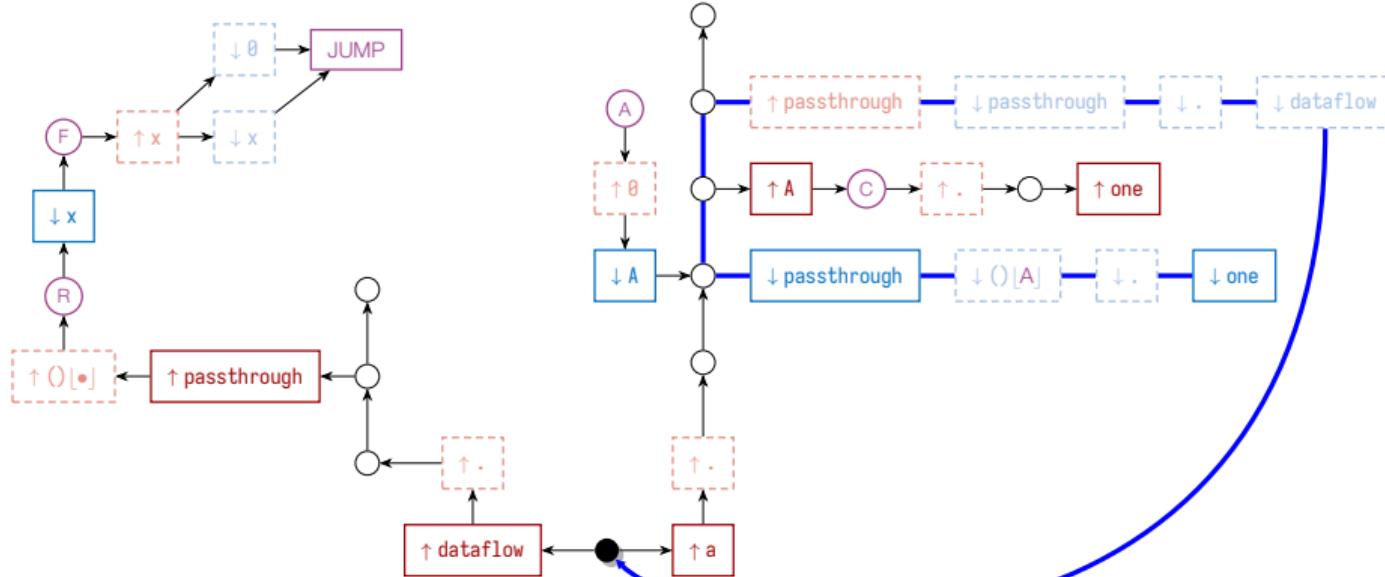
The dataflow example



Symbol stack: $\langle \text{dataflow}. \text{passthrough}().[A].\text{one} \rangle$

Scope stack: \circ

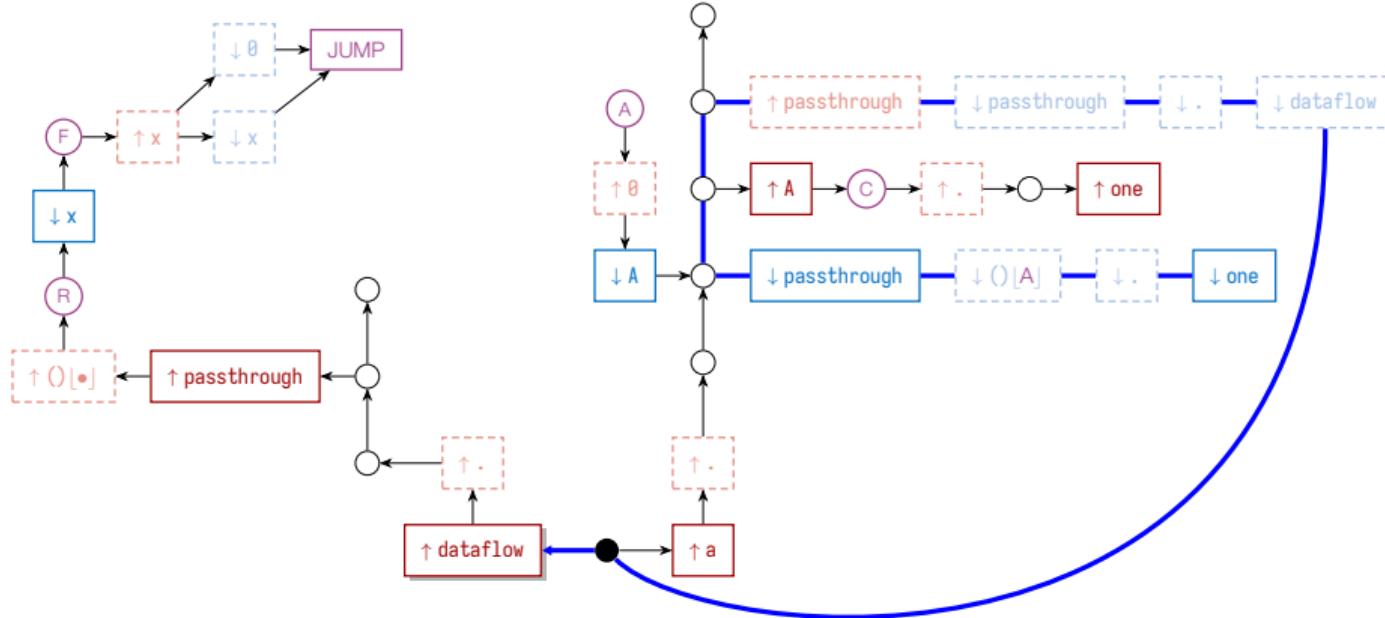
The dataflow example



Symbol stack: $\langle \text{dataflow}. \text{passthrough}()[\text{A}] . \text{one} \rangle$

Scope stack: \circ

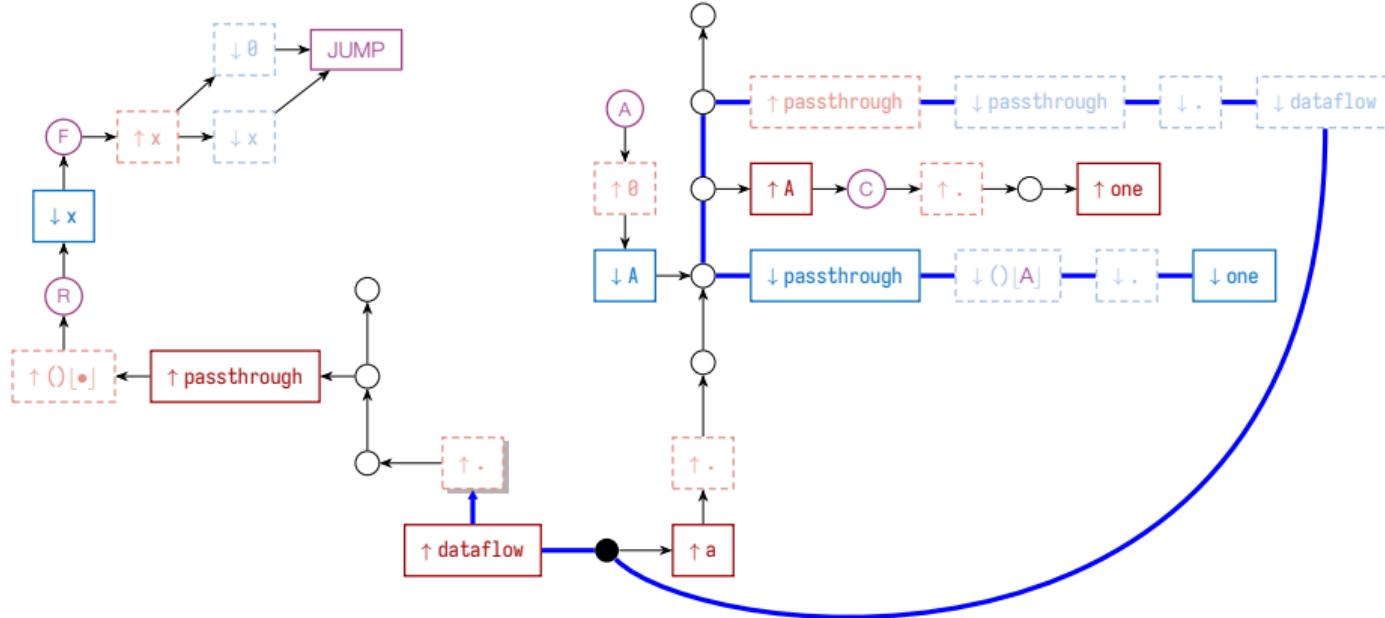
The dataflow example



Symbol stack: $\langle \cdot.\text{passthrough}()[\text{A}].\text{one} \rangle$

Scope stack: \circ

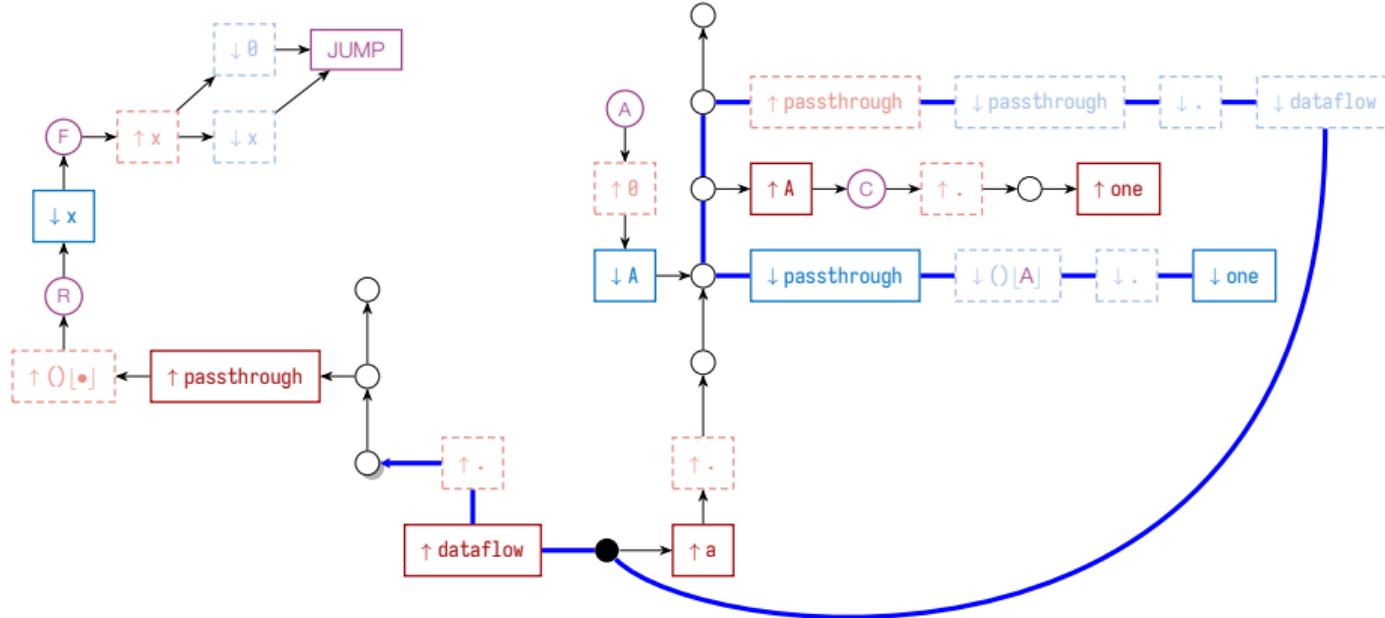
The dataflow example



Symbol stack: $\langle \text{passthrough}()[\text{A}] . \text{one} \rangle$

Scope stack: ○

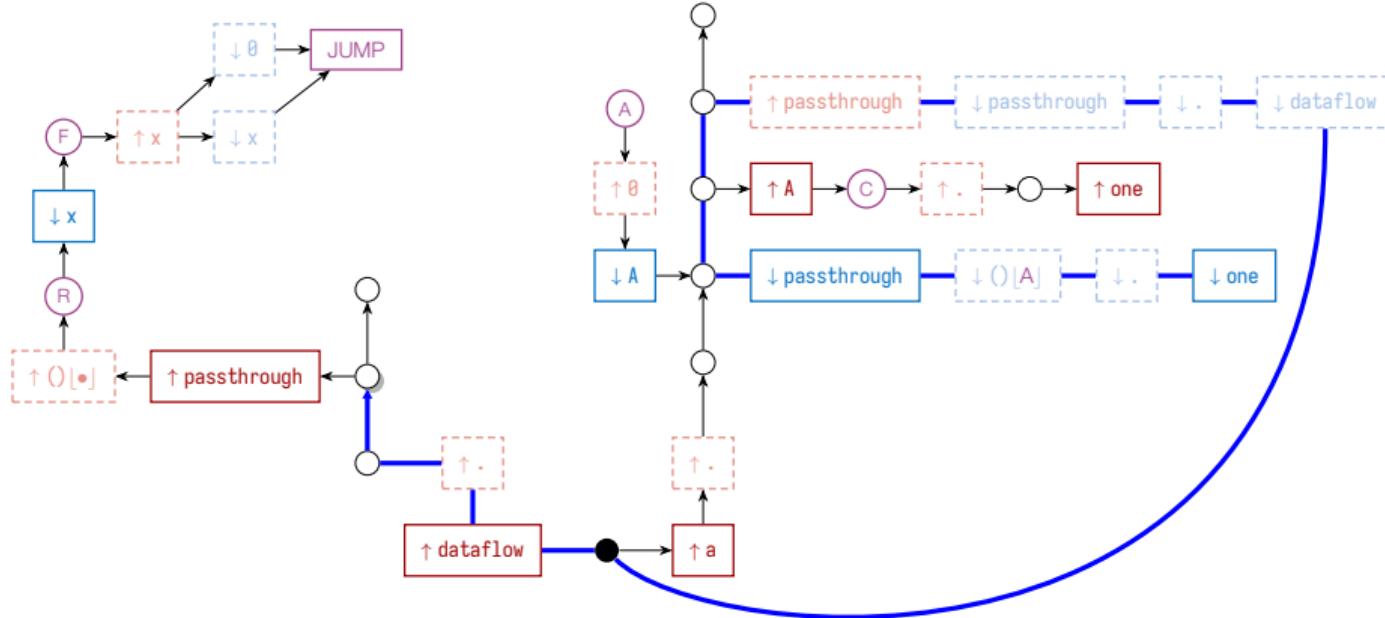
The dataflow example



Symbol stack: $\langle \text{passthrough}()[\text{A}].\text{one} \rangle$

Scope stack: \circ

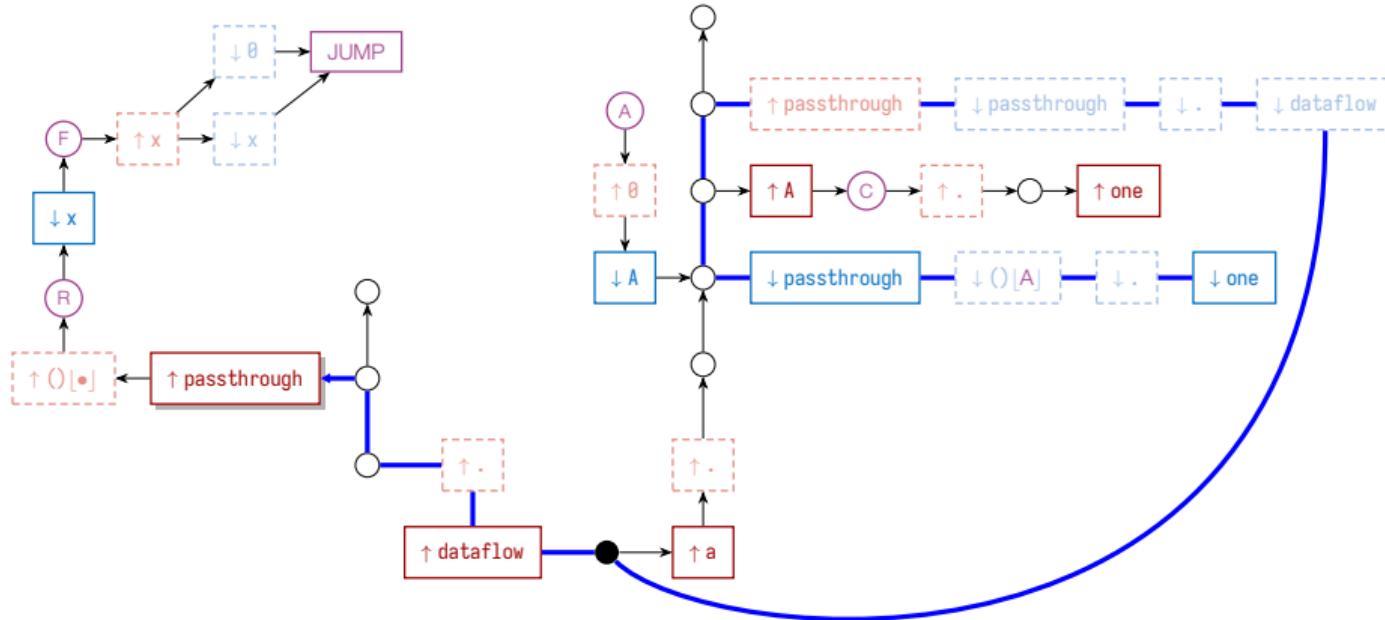
The dataflow example



Symbol stack: $\langle \text{passthrough}()|A|.one \rangle$

Scope stack: \circ

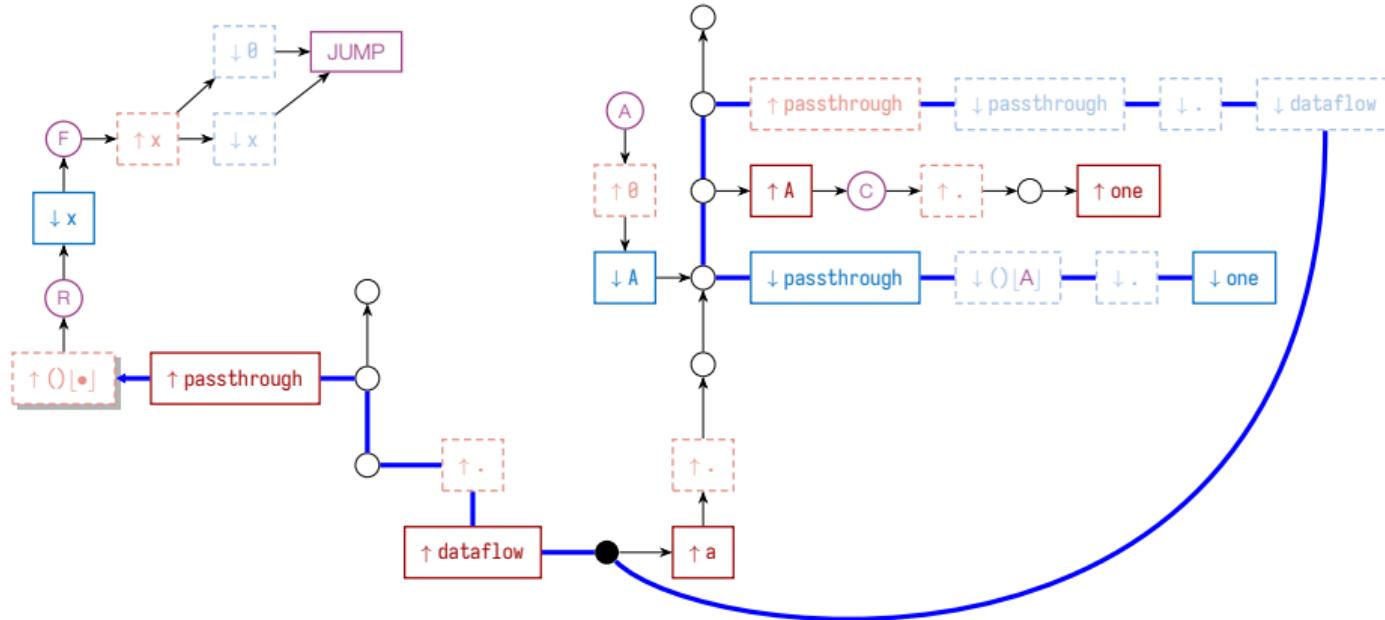
The dataflow example



Symbol stack: $\langle ()[A].\text{one} \rangle$

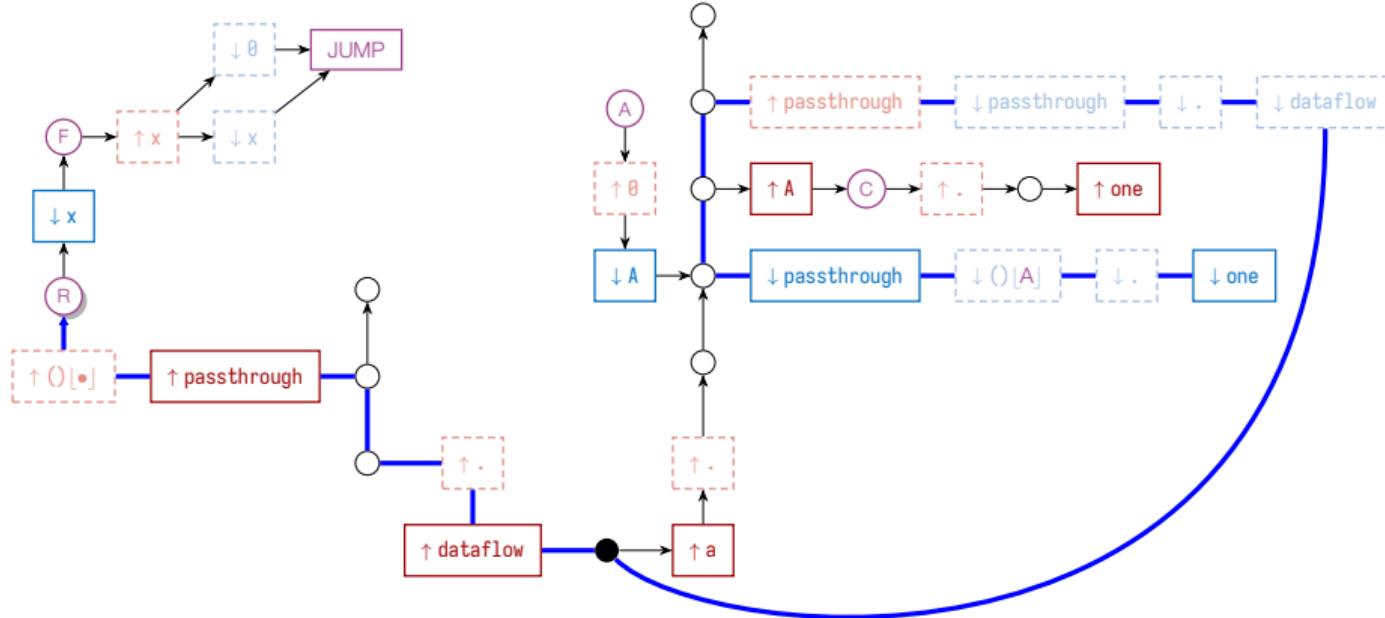
Scope stack: ○

The dataflow example



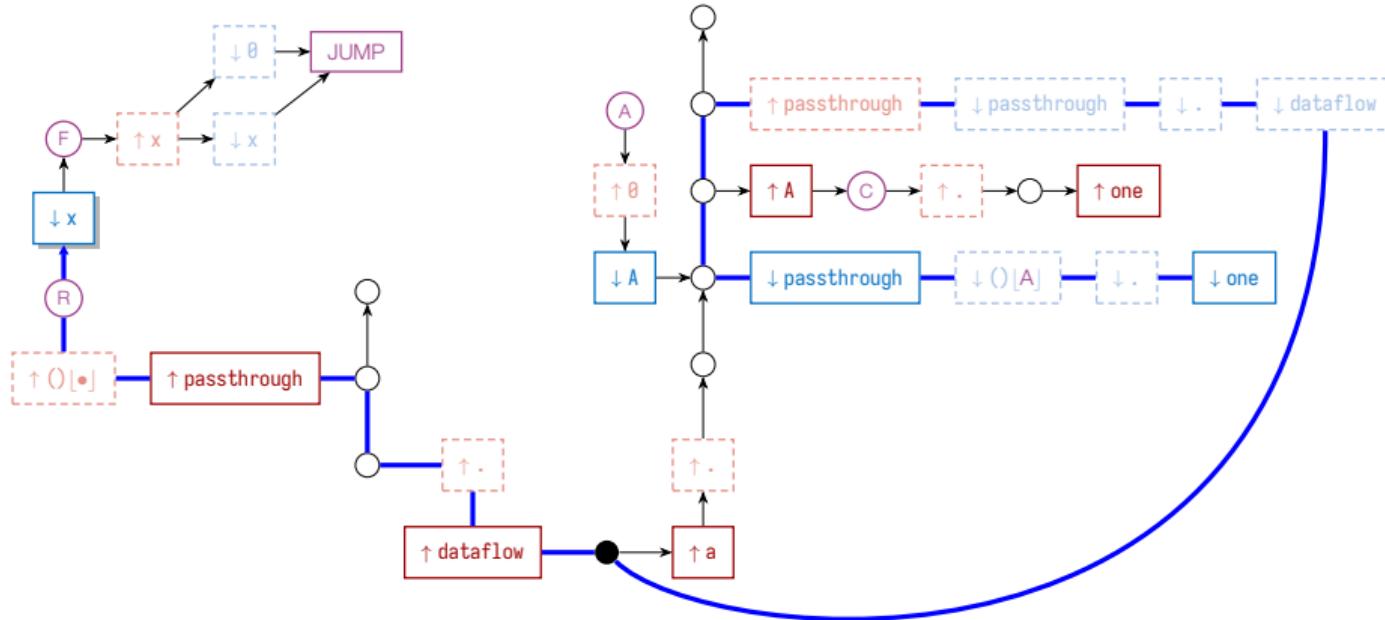
Symbol stack:
 $\langle .\text{one} \rangle$
Scope stack:
 (A)

The dataflow example



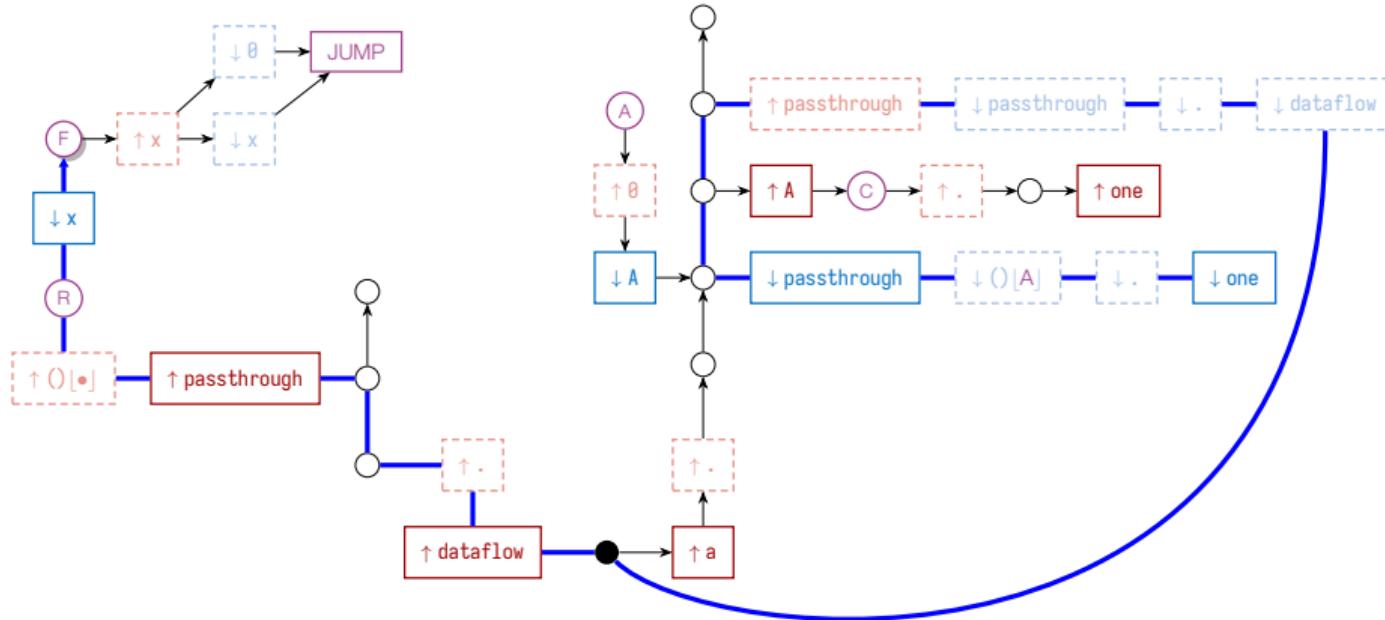
Symbol stack: $\langle .\text{one} \rangle$
Scope stack: (A)

The dataflow example



Symbol stack: $\langle x.\text{one} \rangle$
Scope stack: (A)

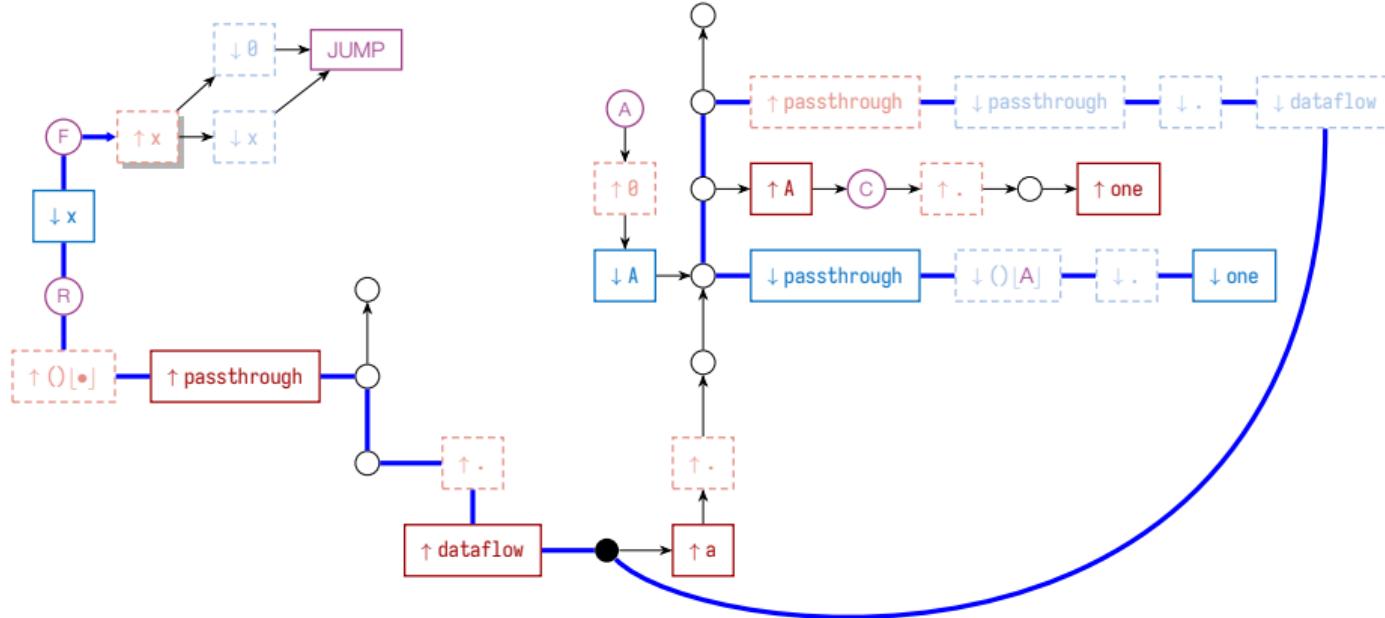
The dataflow example



Symbol stack: $\langle x.\text{one} \rangle$

Scope stack: (A)

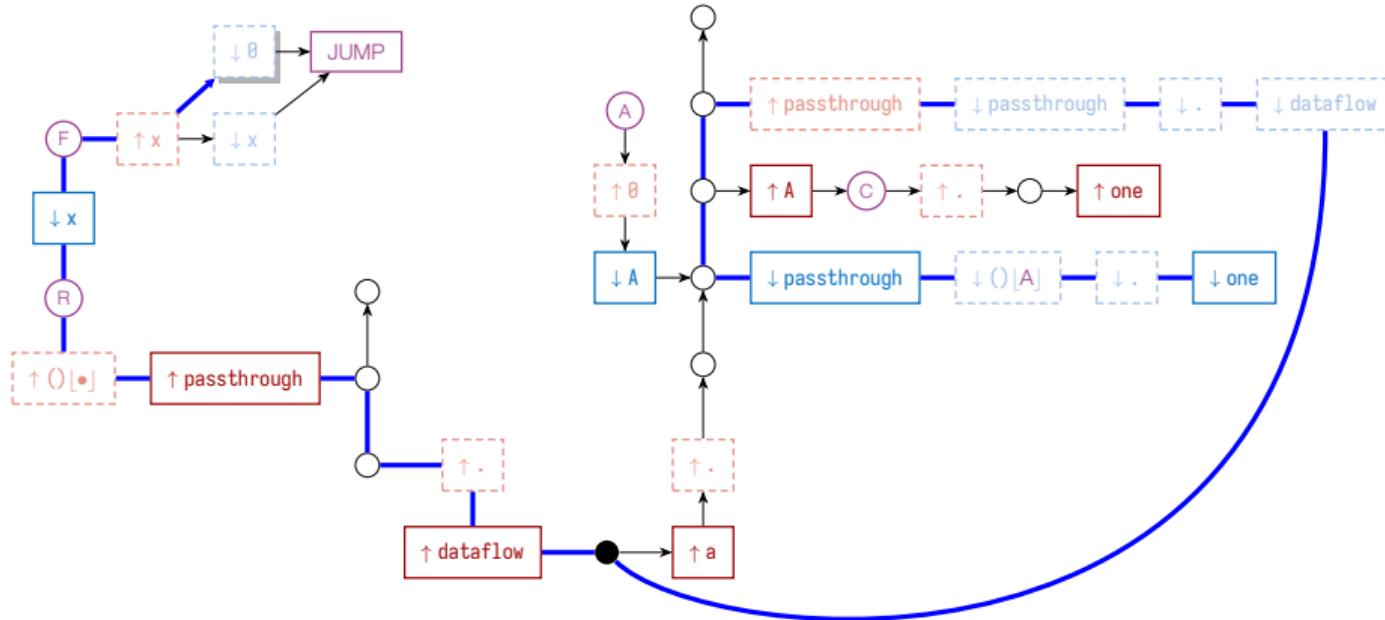
The dataflow example



Symbol stack: $\langle .one \rangle$

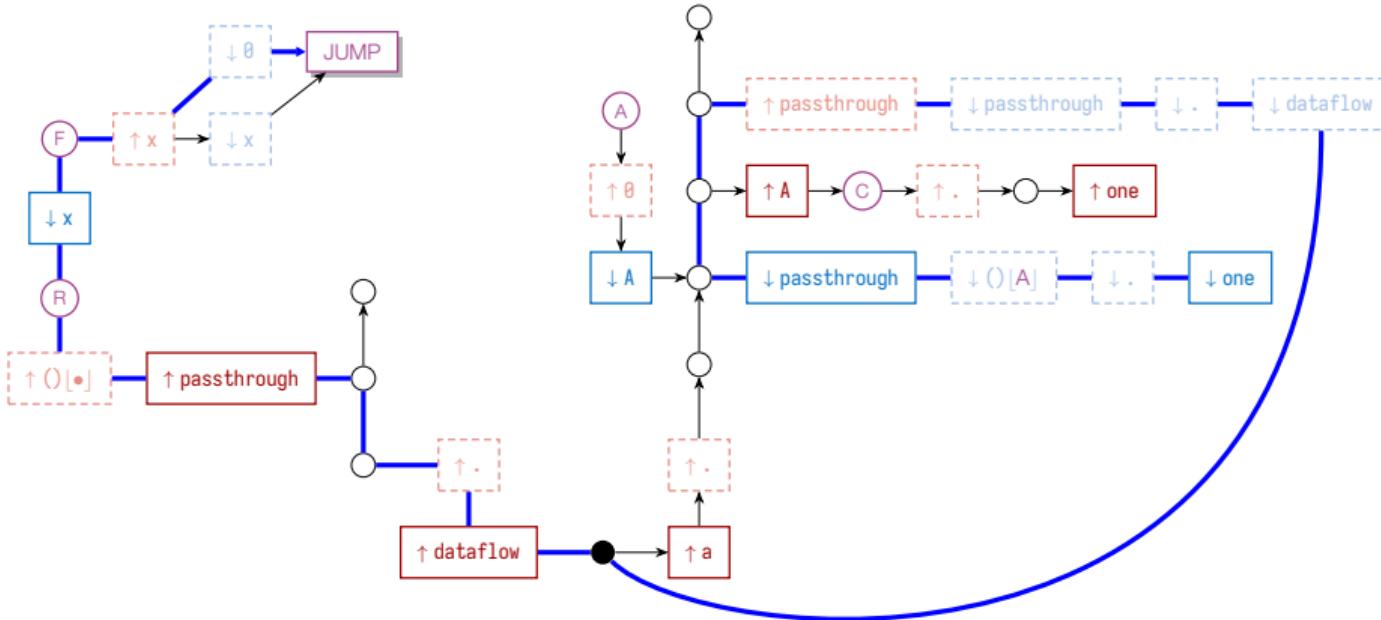
Scope stack: (A)

The dataflow example



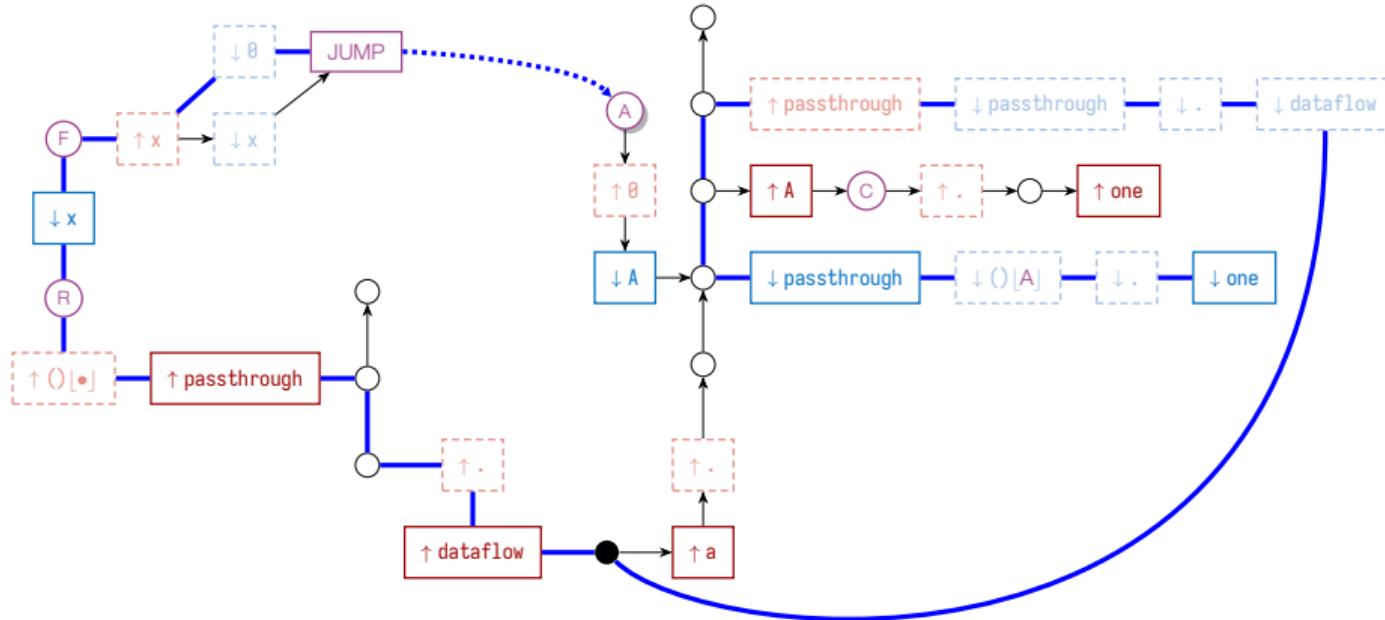
Symbol stack: $\langle 0.one \rangle$
Scope stack: (A)

The dataflow example



Symbol stack: $\langle \theta.\text{one} \rangle$
Scope stack: (A)

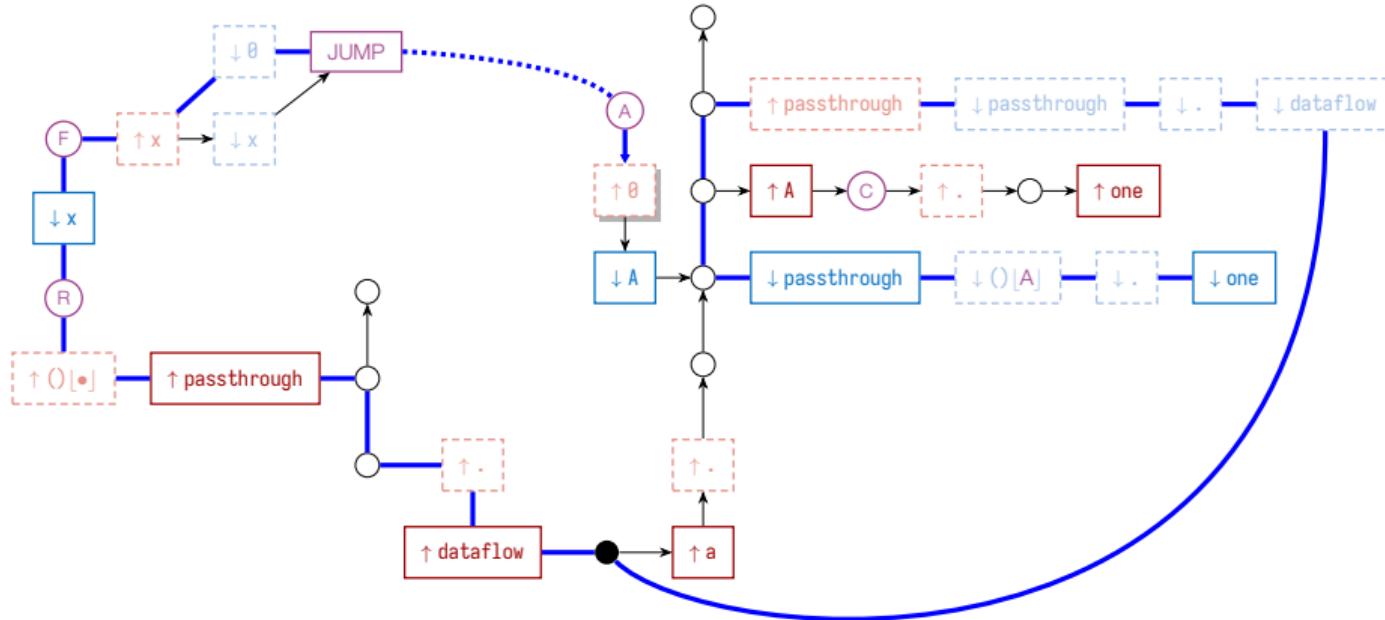
The dataflow example



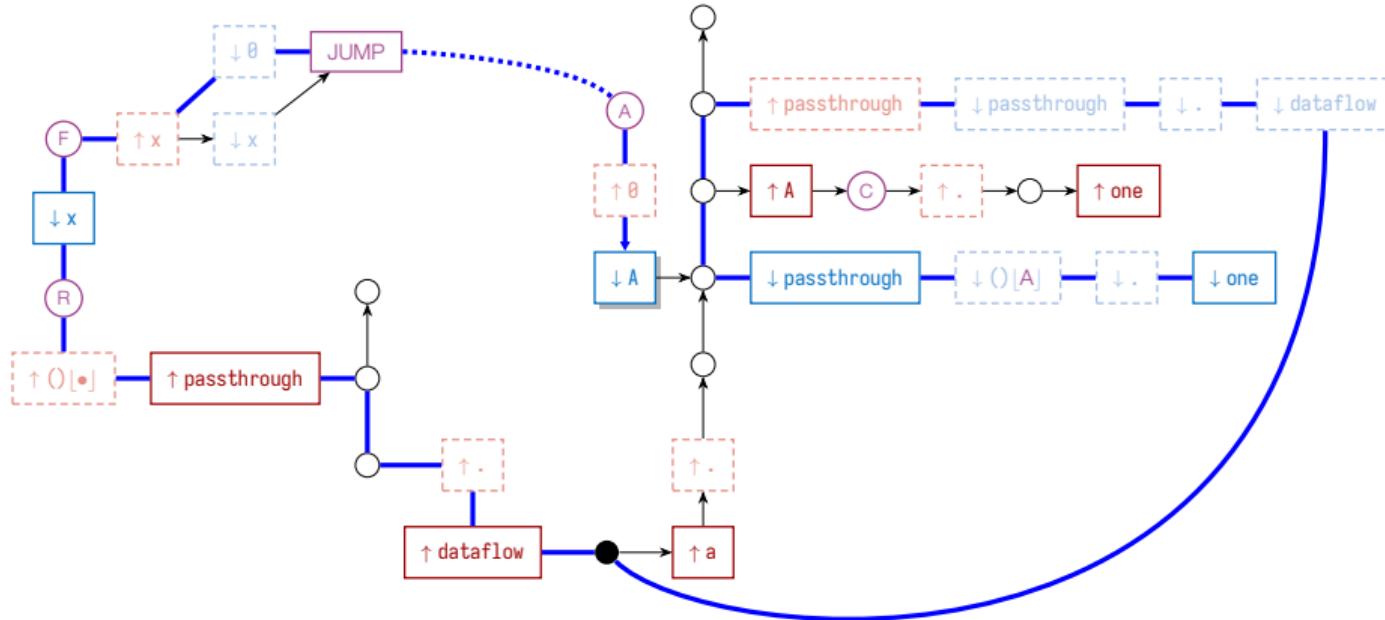
Symbol stack: $\langle 0.one \rangle$

Scope stack: $\langle \rangle$

The dataflow example



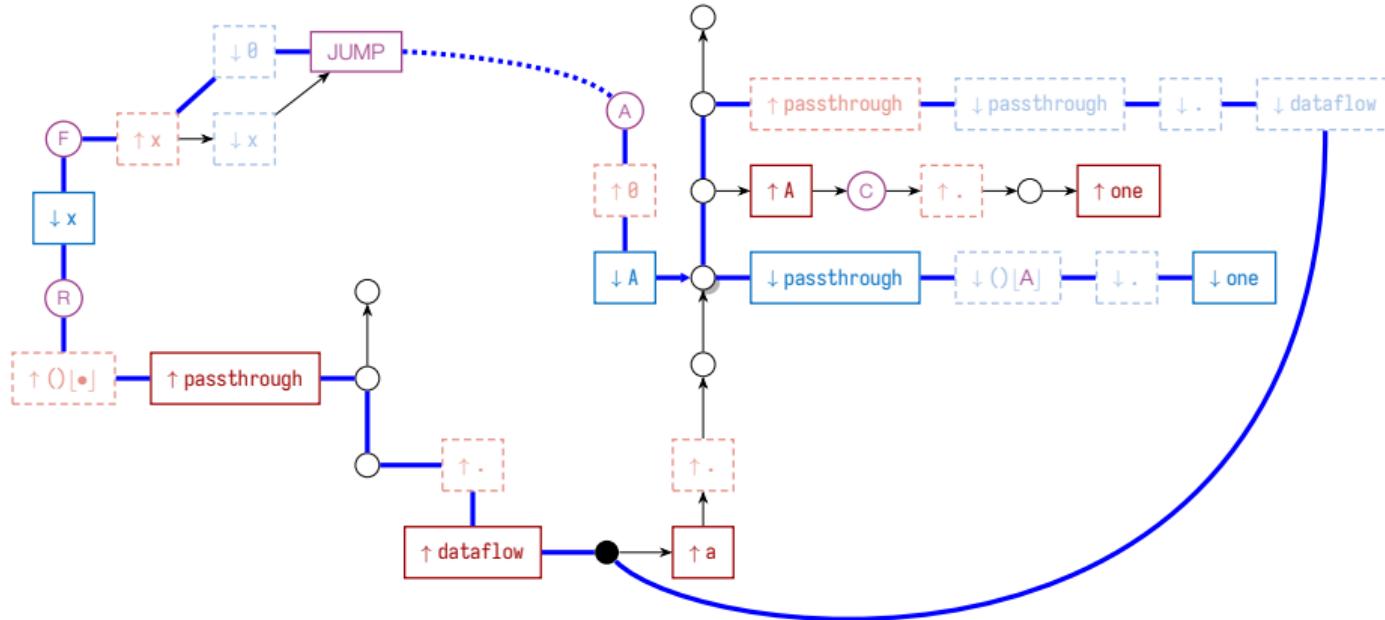
The dataflow example



Symbol stack: $\langle A.\text{one} \rangle$

Scope stack: ○

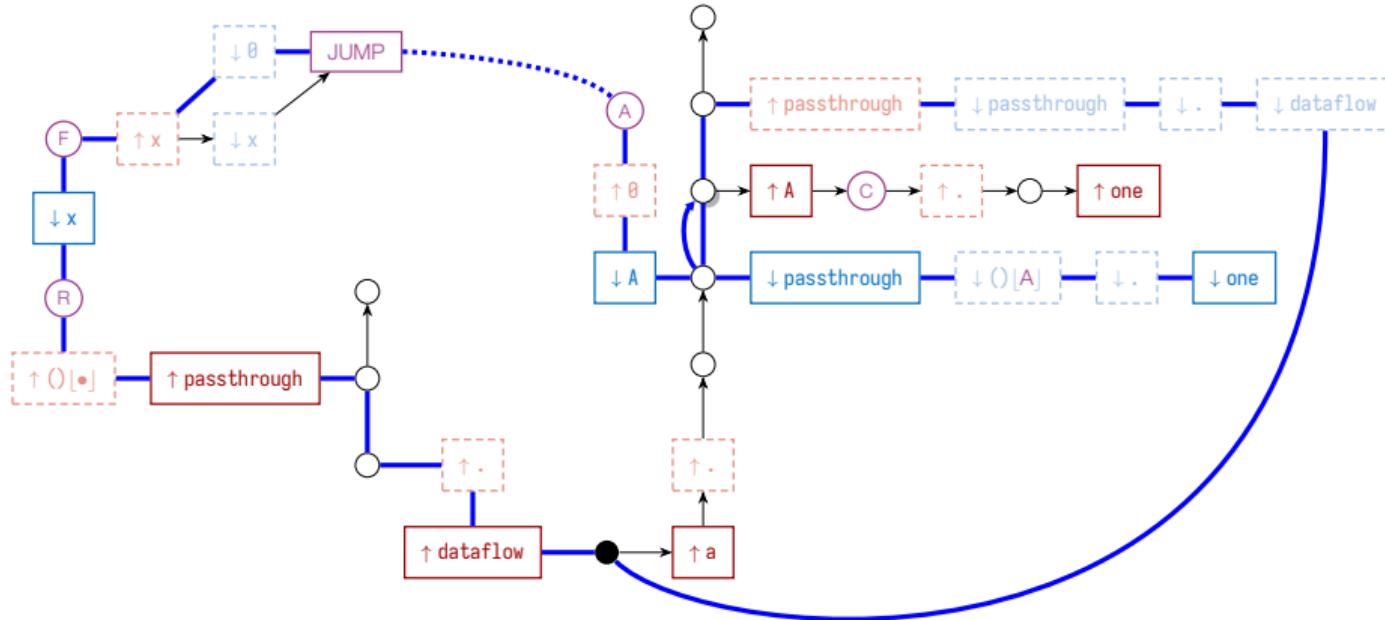
The dataflow example



Symbol stack: $\langle A.\text{one} \rangle$

Scope stack: ○

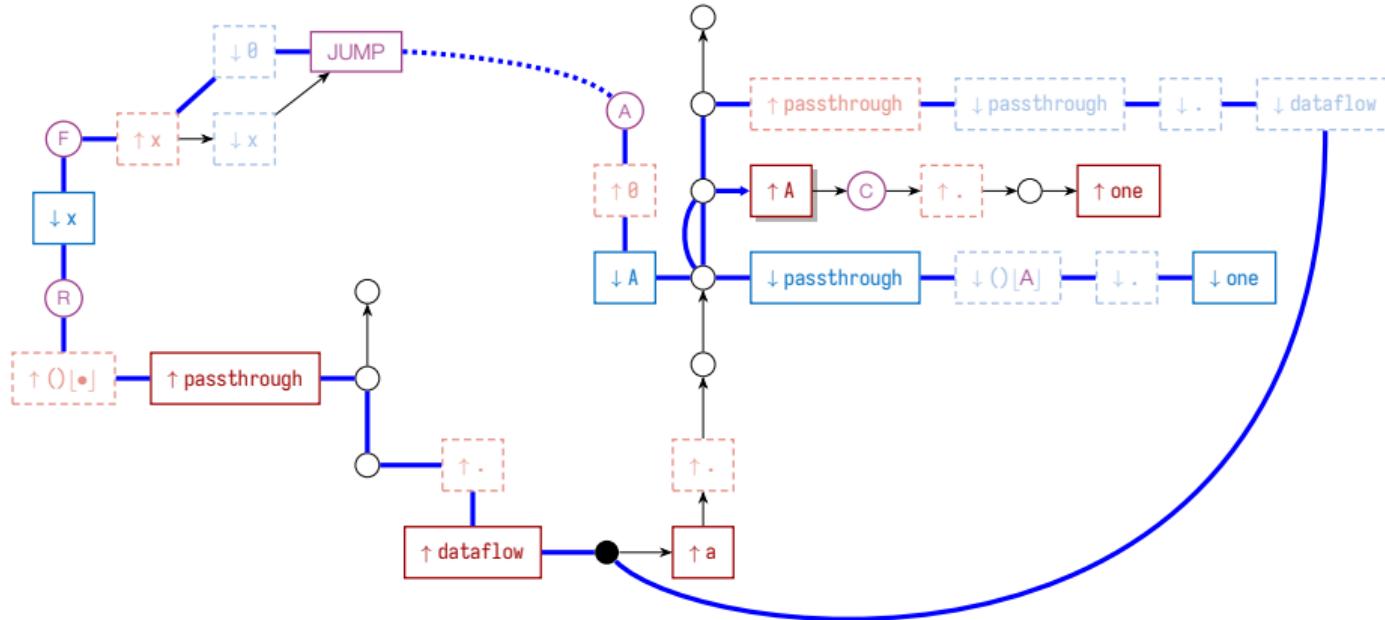
The dataflow example



Symbol stack: $\langle A.\text{one} \rangle$

Scope stack: ○

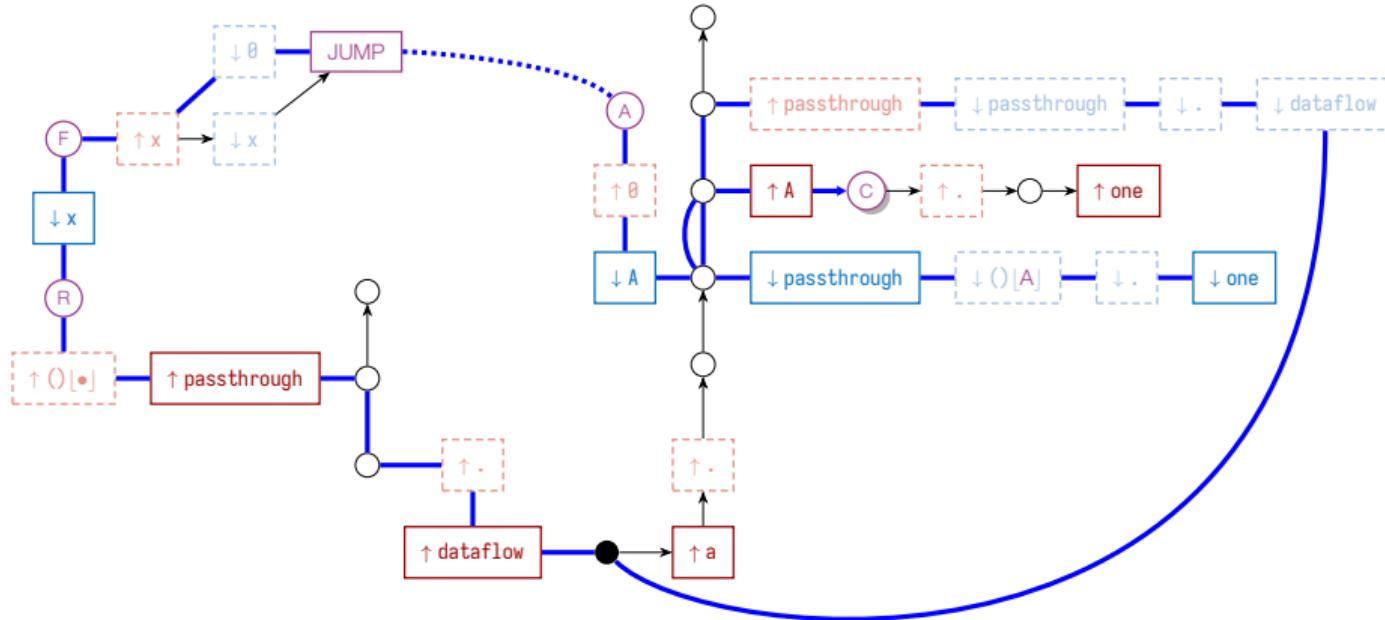
The dataflow example



Symbol stack: $\langle .\text{one} \rangle$

Scope stack: ○

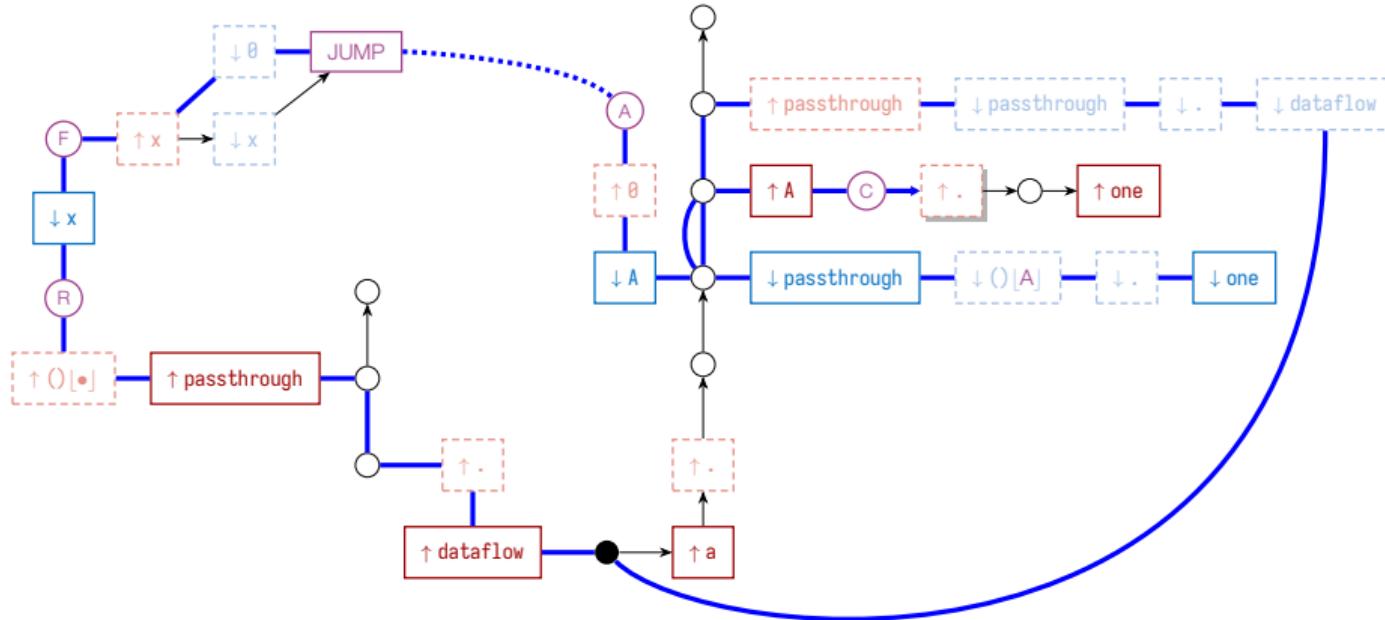
The dataflow example



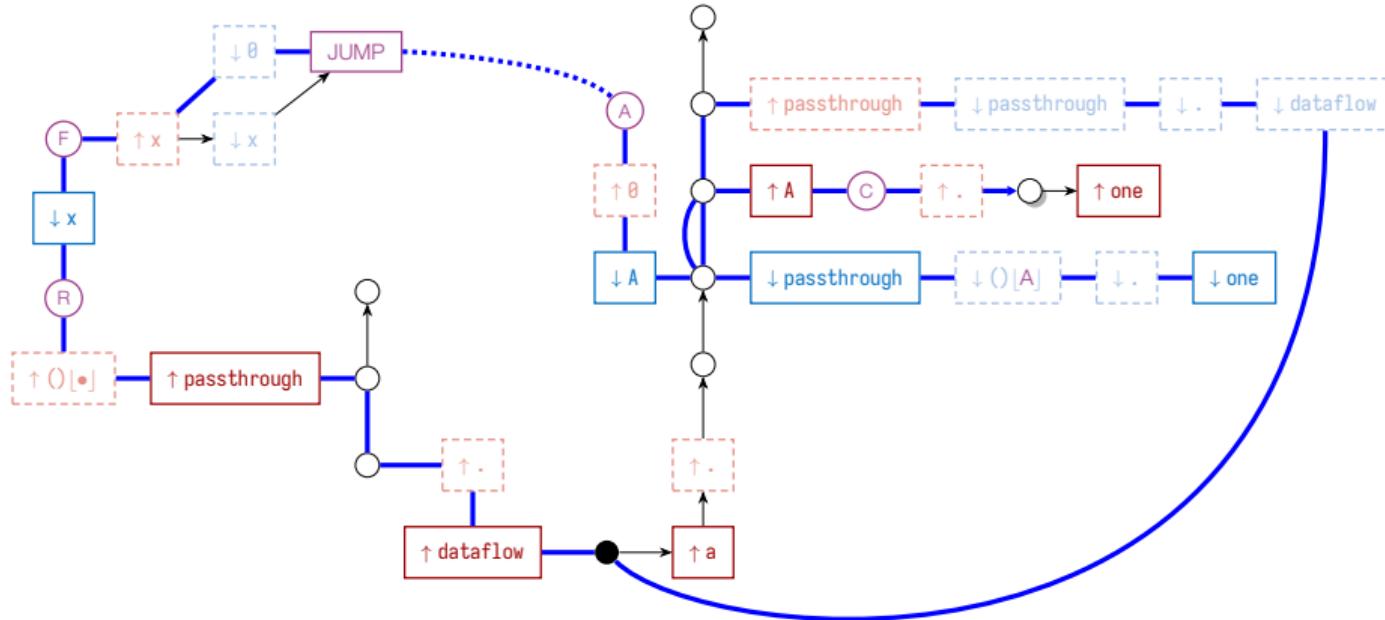
Symbol stack: $\langle \cdot.\text{one} \rangle$

Scope stack: ○

The dataflow example



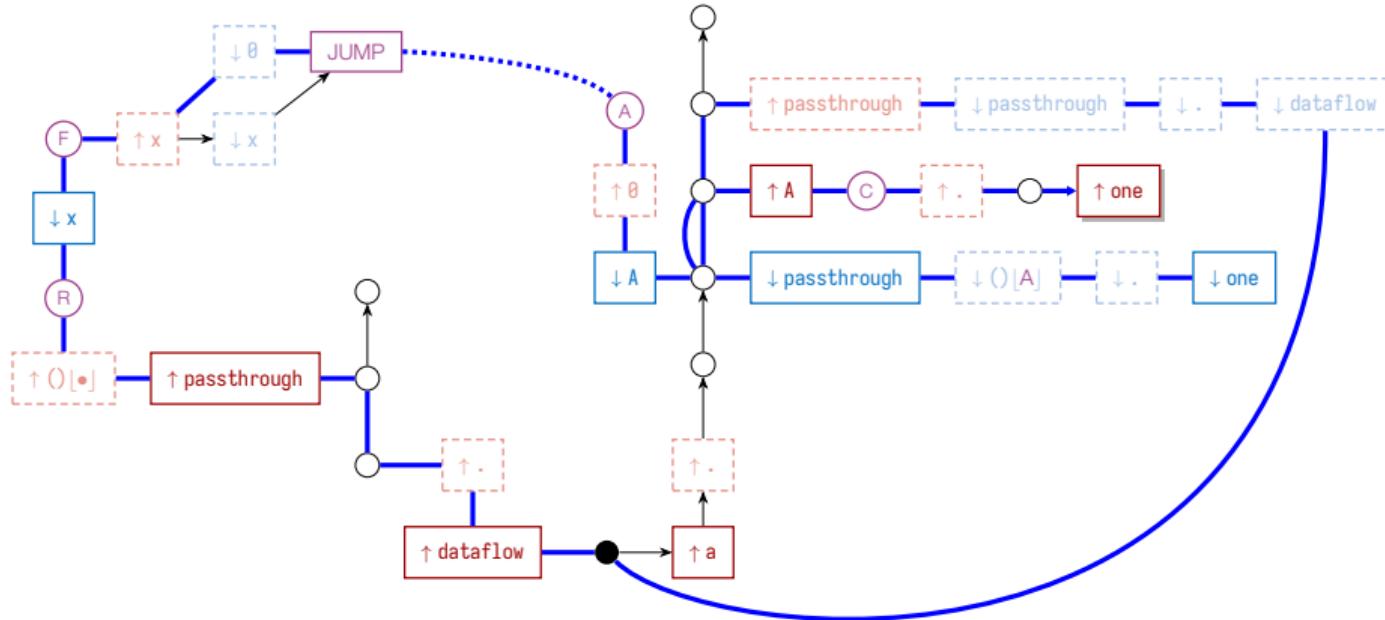
The dataflow example



Symbol stack: $\langle one \rangle$

Scope stack: \circ

The dataflow example



Symbol stack: ◇

Scope stack: ○

Are we done?



We're still doing too much work at query time!

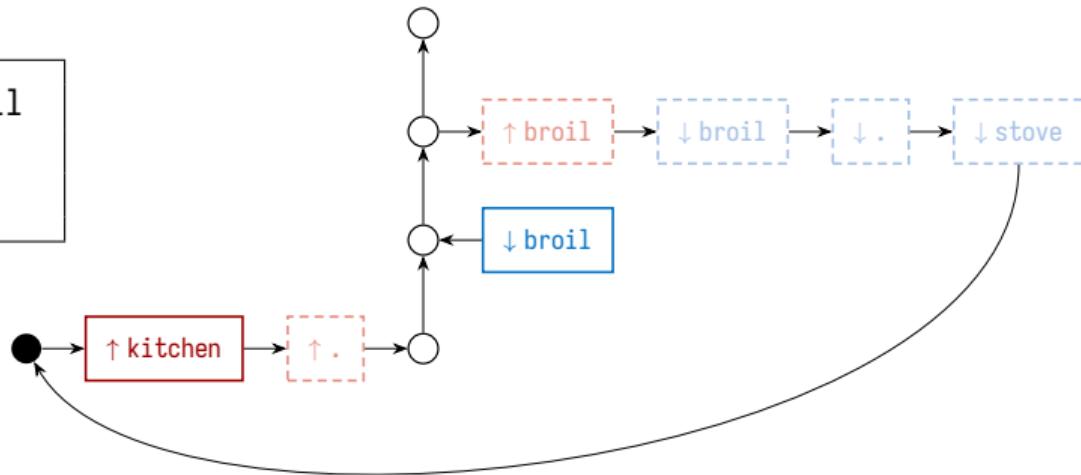
Can we shift more of the work to index time,
while still remaining incremental?

A photograph showing a large-scale bridge construction project. In the foreground, a white barge with the name "BENJAMIN FOSS" is carrying a massive, dark grey concrete bridge girder. The girder is suspended from a red lattice boom crane standing on the barge. To the left, a green truss bridge spans a river, its structure partially collapsed or under construction. In the background, a long, elevated concrete bridge extends across the water, supported by numerous tall, cylindrical piers. The sky is clear and blue.

Partial paths

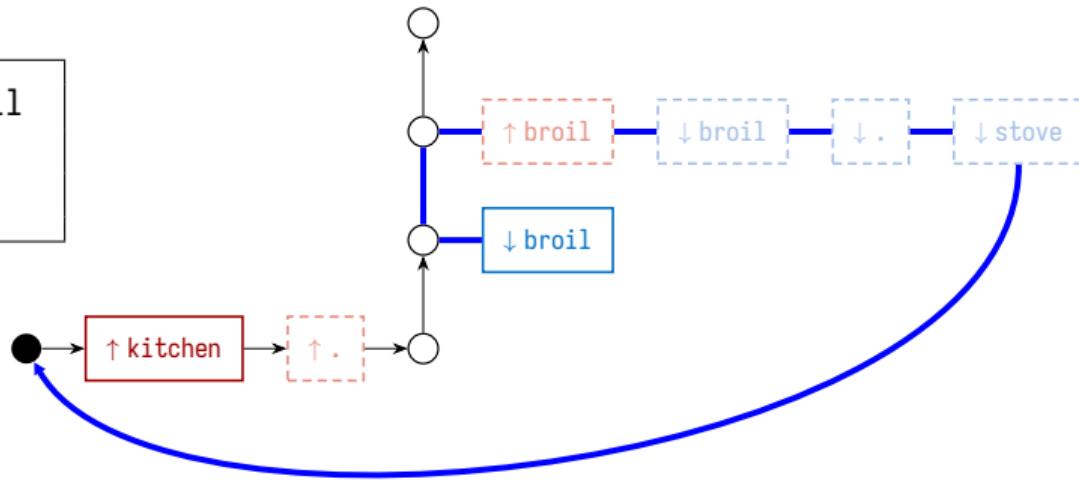
Partial paths

```
kitchen.py
from stove import broil
broil()
```



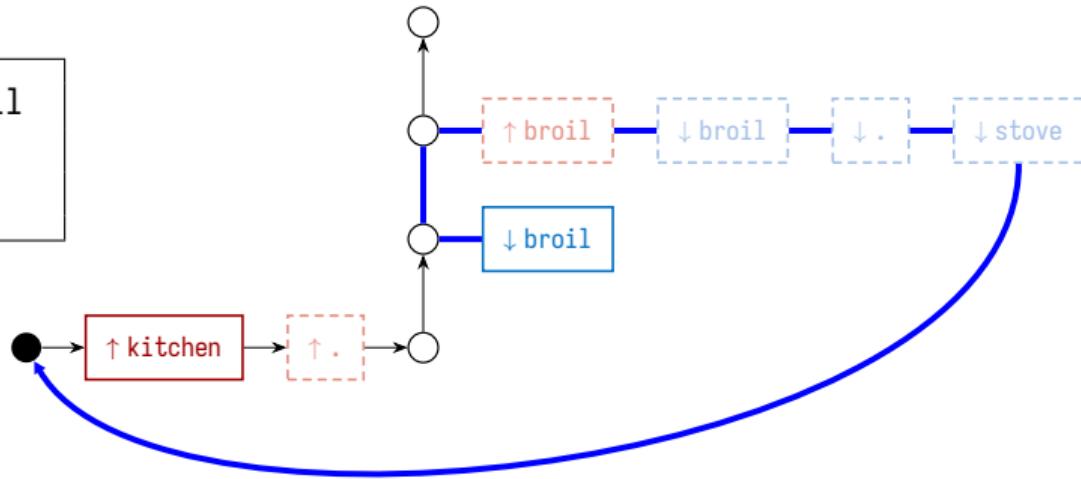
Partial paths

```
kitchen.py
from stove import broil
broil()
```



Partial paths

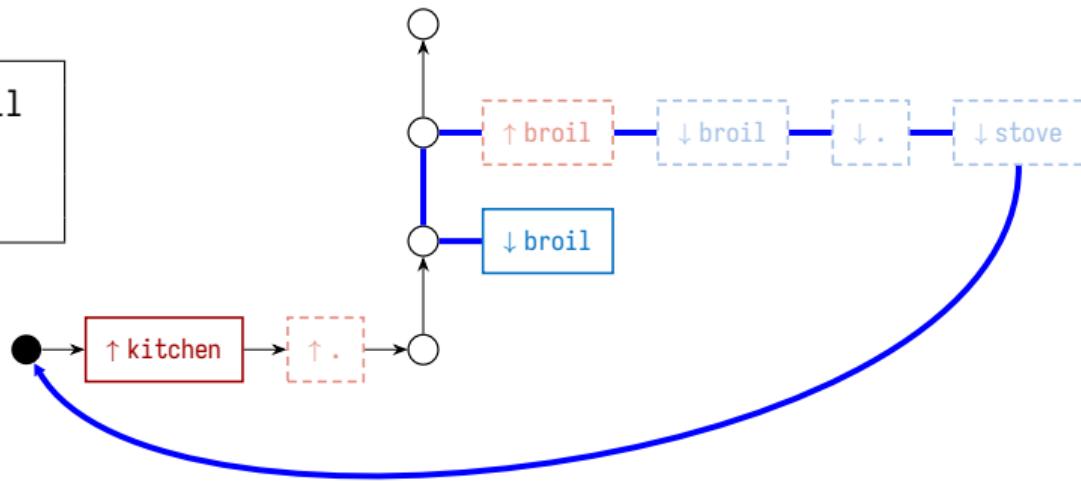
```
kitchen.py
from stove import broil
broil()
```



$$\{\diamond, \circ\} \quad \boxed{\downarrow \text{broil}} \rightsquigarrow \bullet \quad \{\langle \text{stove.broil} \rangle, \circ\}$$

Partial paths

```
kitchen.py
from stove import broil
broil()
```



$$\{\diamond, \circ\} \quad \boxed{\downarrow \text{broil}} \rightsquigarrow \bullet \quad \{\langle \text{stove.broil} \rangle, \circ\}$$

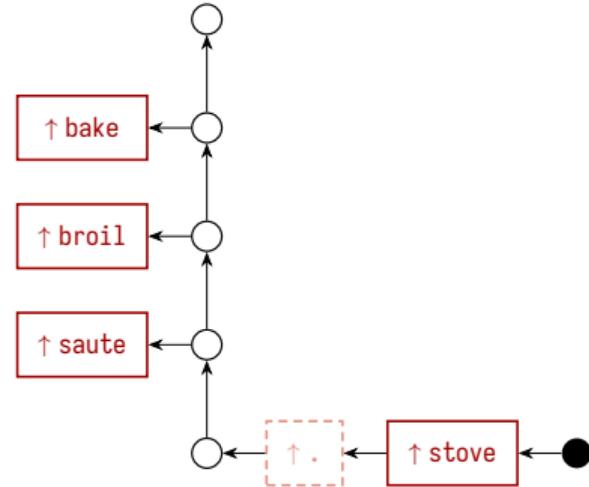
The reference at `kitchen.py:3:1` refers to `stove.broil` in some other file

Partial paths

```
stove.py
def bake():
    pass

def broil():
    pass

def saute():
    pass
```

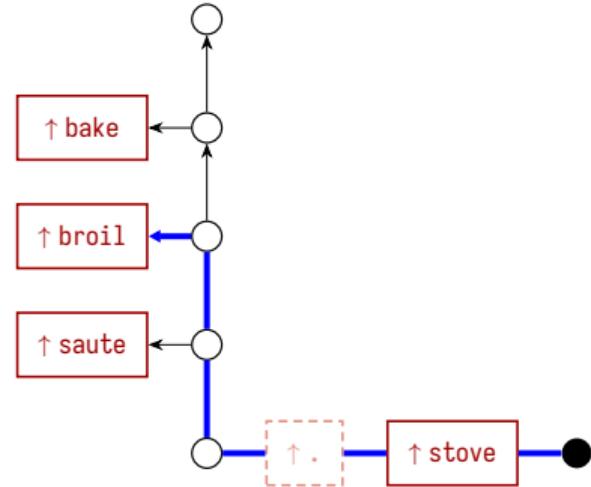


Partial paths

```
stove.py
def bake():
    pass

def broil():
    pass

def saute():
    pass
```

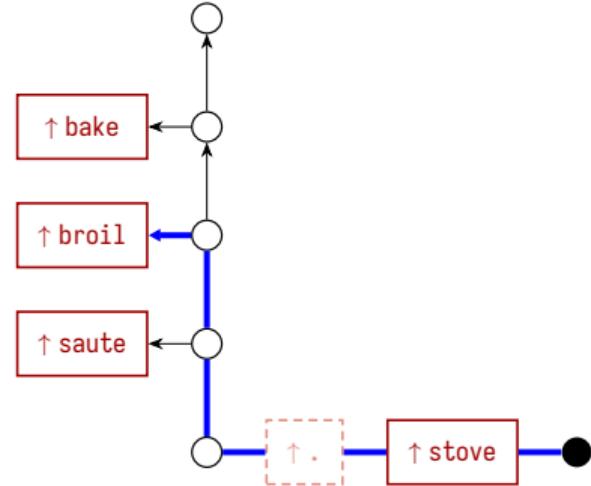


Partial paths

```
stove.py
def bake():
    pass

def broil():
    pass

def saute():
    pass
```



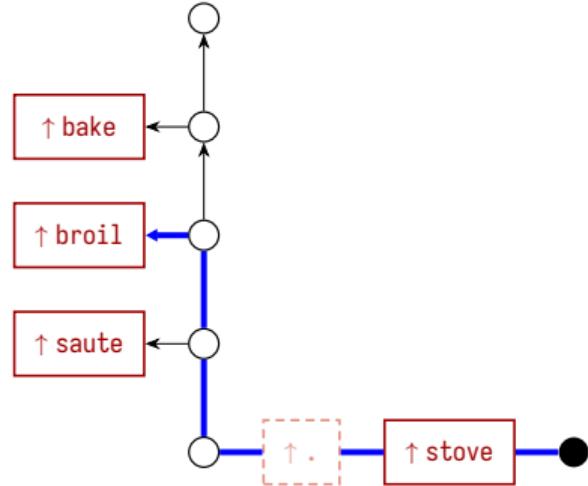
$$\{\langle \text{stove.broil} \rangle \cdot \psi, \phi\} \quad \bullet \rightsquigarrow \boxed{\uparrow \text{broil}} \quad \{\psi, \phi\}$$

Partial paths

```
stove.py
def bake():
    pass

def broil():
    pass

def saute():
    pass
```



$$\{\langle \text{stove.broil} \rangle \cdot \psi, \phi\} \quad \bullet \rightsquigarrow \boxed{\uparrow \text{broil}} \quad \{\psi, \phi\}$$

`stove.broil` is defined at `stove.py:4:5`.

Concatenating partial paths

$\{\diamond, \circ\} \xrightarrow{\downarrow \text{broil}} \bullet \{\langle \text{stove.broil} \rangle, \circ\}$

+

$\{\langle \text{stove.broil} \rangle \cdot \psi, \phi\} \bullet \xrightarrow{\uparrow \text{broil}} \{\psi, \phi\}$

The reference at *kitchen.py:3:1*
refers to `stove.broil` in some other file

+

`stove.broil` is defined at *stove.py:4:5*

Concatenating partial paths

$$\{\diamond, \circ\} \xrightarrow{\downarrow \text{broil}} \bullet \{\langle \text{stove.broil} \rangle, \circ\} + \{\langle \text{stove.broil} \rangle \cdot \psi, \phi\} \bullet \xrightarrow{\uparrow \text{broil}} \{\psi, \phi\}$$
$$\psi = \diamond, \phi = \circ$$

The reference at *kitchen.py:3:1*
refers to `stove.broil` in some other file + `stove.broil` is defined at *stove.py:4:5*

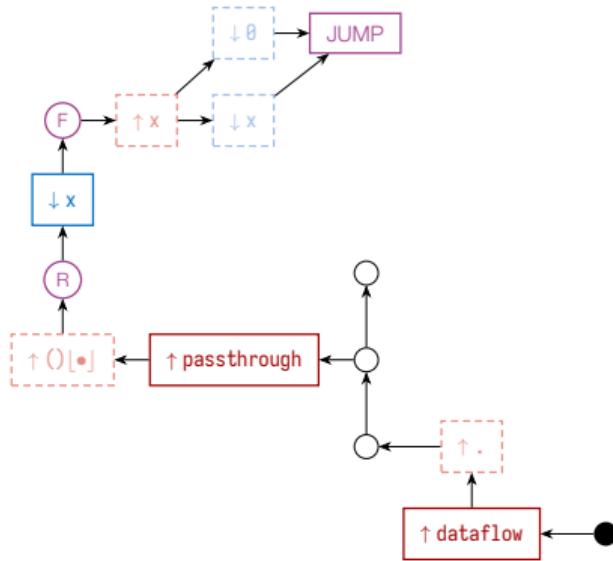
Concatenating partial paths



The reference at *kitchen.py:3:1*
is defined at *stove.py:4:5*.

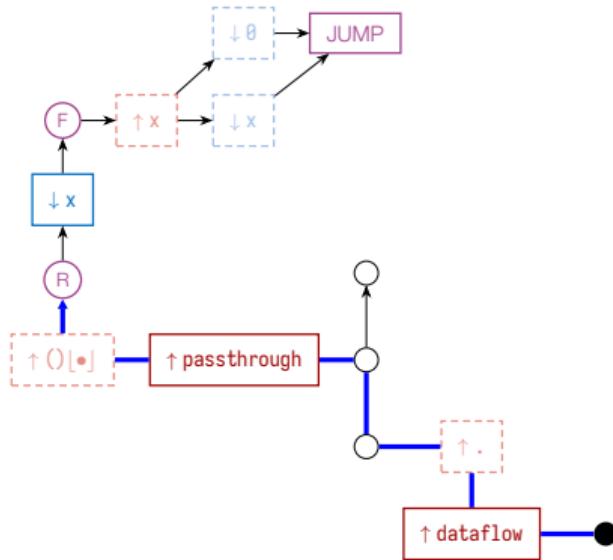
The dataflow example

```
dataflow.py
def passthrough(x):
    return x
```



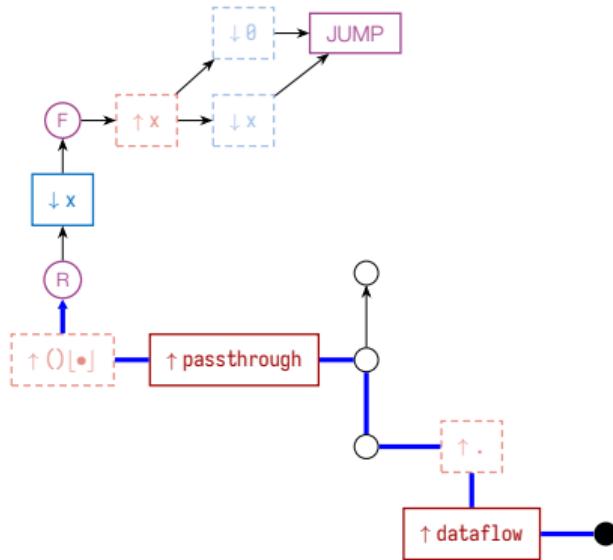
The dataflow example

```
dataflow.py
def passthrough(x):
    return x
```



The dataflow example

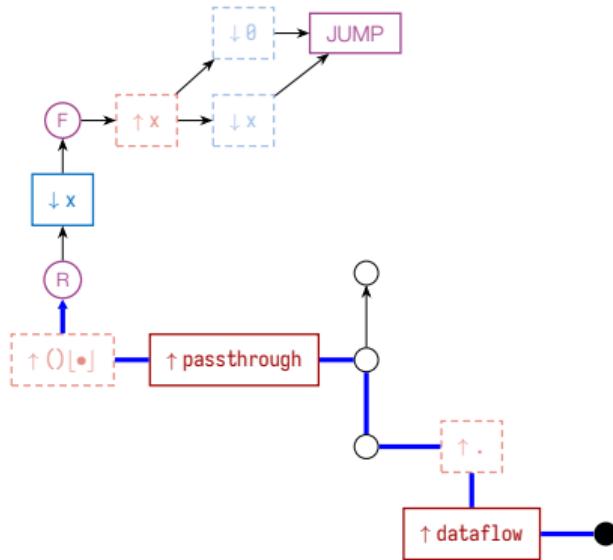
```
dataflow.py
def passthrough(x):
    return x
```



$$\{\langle \text{dataflow.passthrough}() \rfloor_{\phi_A} \rangle \cdot \psi, \phi\} \quad \bullet \rightsquigarrow \textcircled{R} \quad \{\psi, \phi_A\}$$

The dataflow example

```
dataflow.py
def passthrough(x):
    return x
```

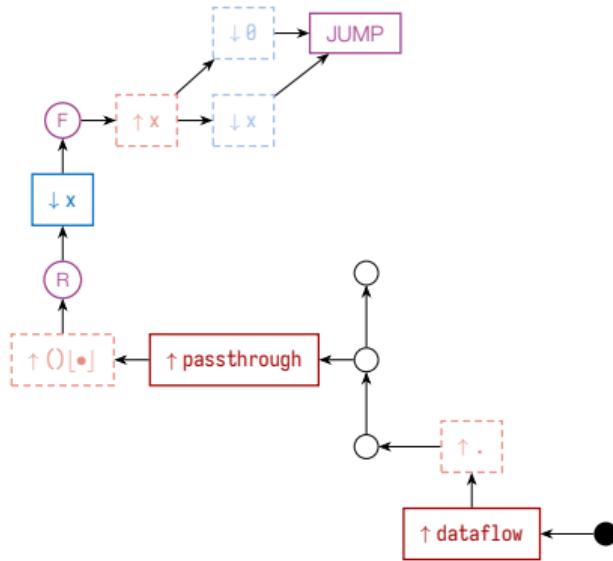


$$\{\langle \text{dataflow.passthrough}()[\phi_A] \rangle \cdot \psi, \phi\} \quad \bullet \rightsquigarrow \textcolor{purple}{(R)} \quad \{\psi, \phi_A\}$$

`dataflow.passthrough` is a function
that can be invoked.

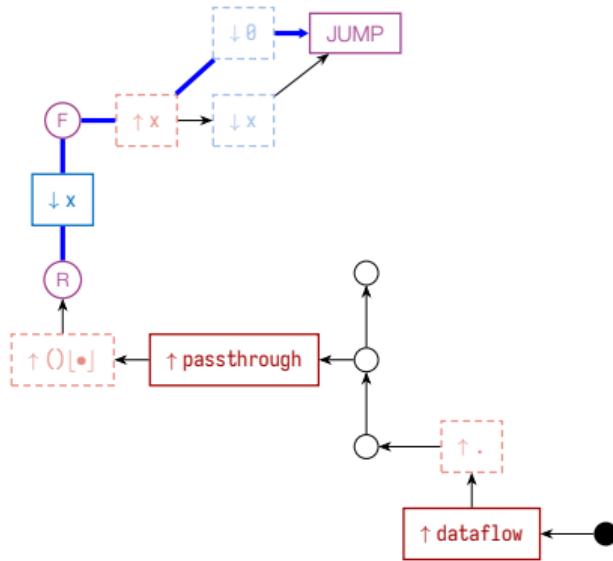
The dataflow example

```
dataflow.py
def passthrough(x):
    return x
```



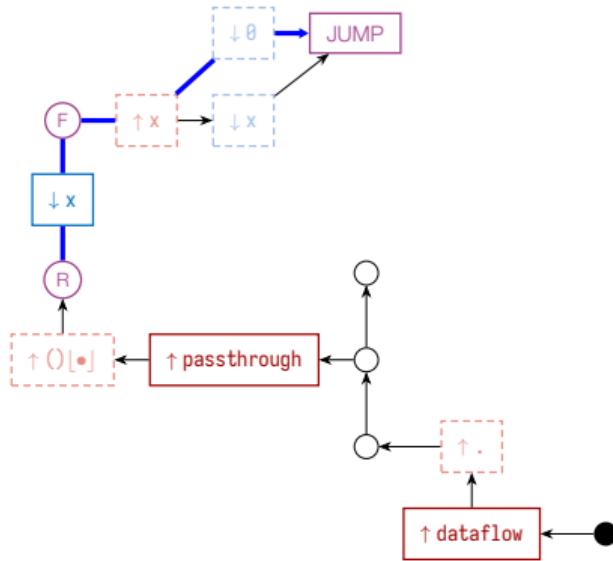
The dataflow example

```
dataflow.py
def passthrough(x):
    return x
```



The dataflow example

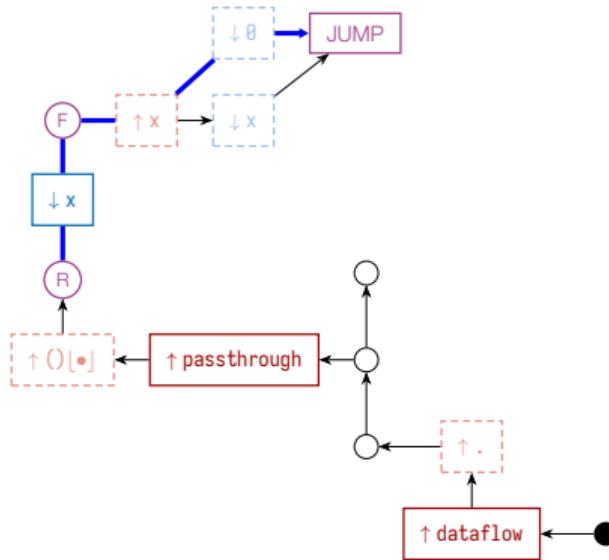
```
dataflow.py
def passthrough(x):
    return x
```



$$\{\psi, \phi\} \xrightarrow{R} \text{JUMP} \quad \{\langle \theta \rangle \cdot \psi, \phi\}$$

The dataflow example

```
dataflow.py
def passthrough(x):
    return x
```

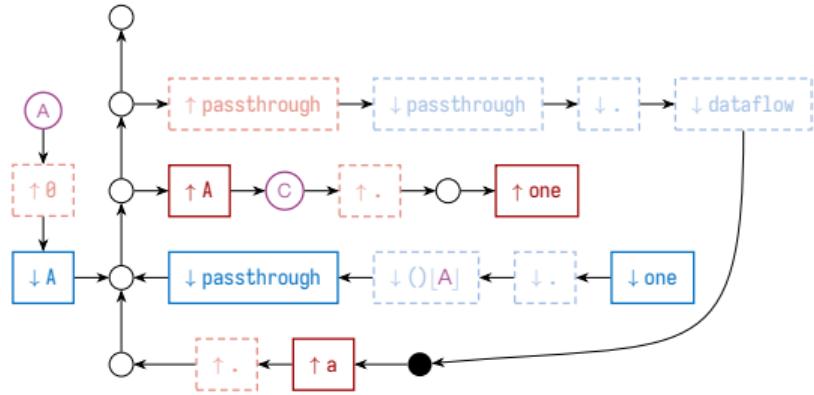


$$\{\psi, \phi\} \xrightarrow{R} \text{JUMP} \quad \{\langle 0 \rangle \cdot \psi, \phi\}$$

The return value of `dataflow.passthrough`
has the same type as positional parameter 0.

The dataflow example

```
a.py  
from dataflow import passthrough  
  
class A:  
    one = 1  
  
    passthrough(A).one
```

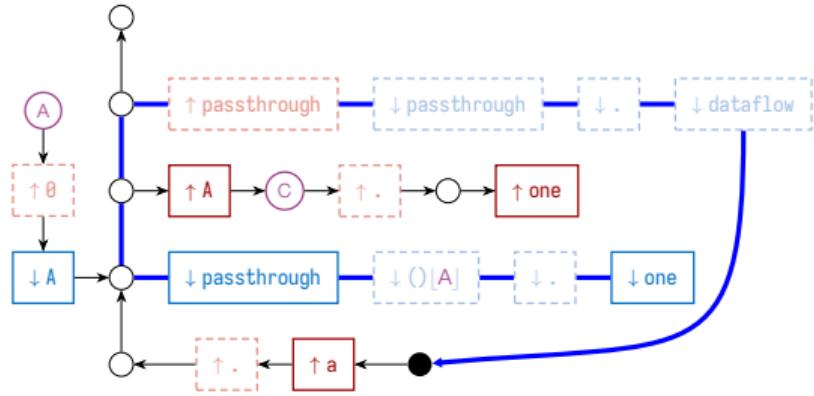


The dataflow example

```
a.py
from dataflow import passthrough

class A:
    one = 1

    passthrough(A).one
```

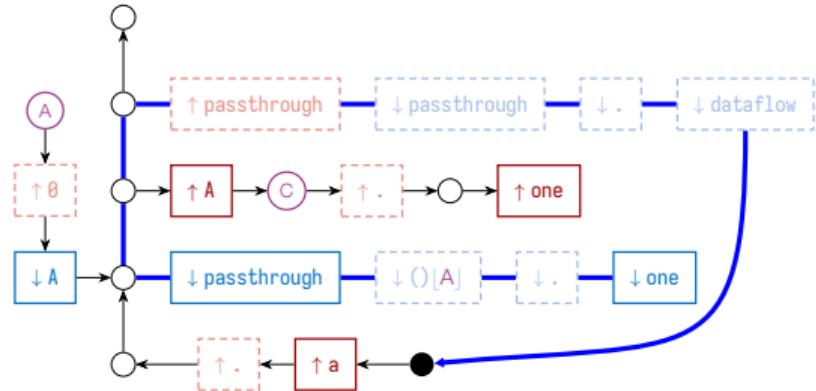


The dataflow example

```
a.py
from dataflow import passthrough

class A:
    one = 1

    passthrough(A).one
```



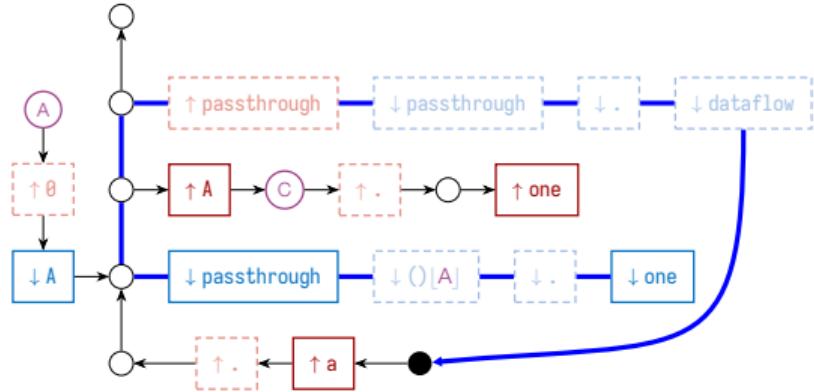
$$\{\diamond, \circ\} \quad \boxed{\text{one}} \rightsquigarrow \bullet \quad \{(\text{dataflow.passthrough()}[\text{A}].\text{one}), \circ\}$$

The dataflow example

```
a.py
from dataflow import passthrough

class A:
    one = 1

    passthrough(A).one
```

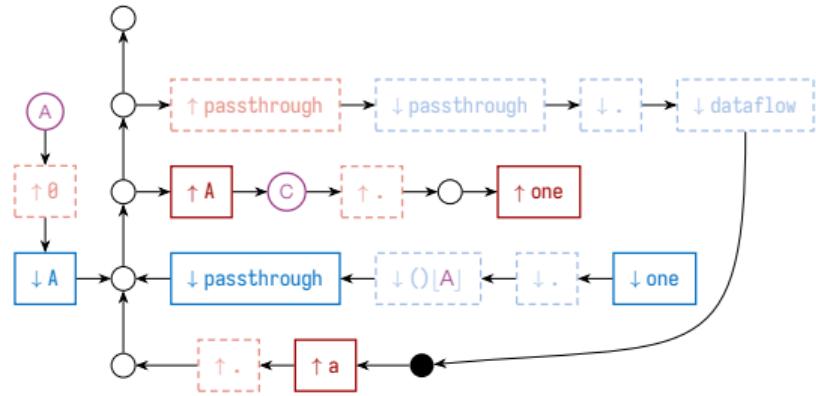


$$\{\diamond, \circ\} \quad \boxed{\text{one}} \rightsquigarrow \bullet \quad \{\langle \text{dataflow.passthrough()}[\text{A}].\text{one} \rangle, \circ\}$$

If you can find what `dataflow.passthrough` resolves to and can call it
then the result should have a member named `one`
which the reference at `a.py:6:16` resolves to.

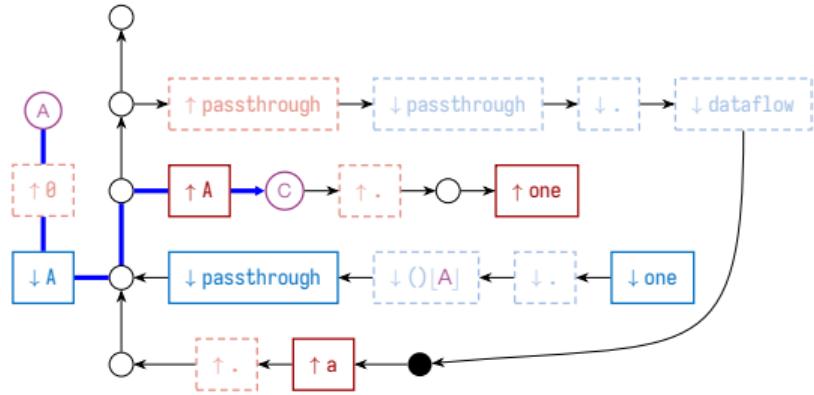
The dataflow example

```
a.py  
from dataflow import passthrough  
  
class A:  
    one = 1  
  
passthrough(A).one
```



The dataflow example

```
a.py  
from dataflow import passthrough  
  
class A:  
    one = 1  
  
    passthrough(A).one
```

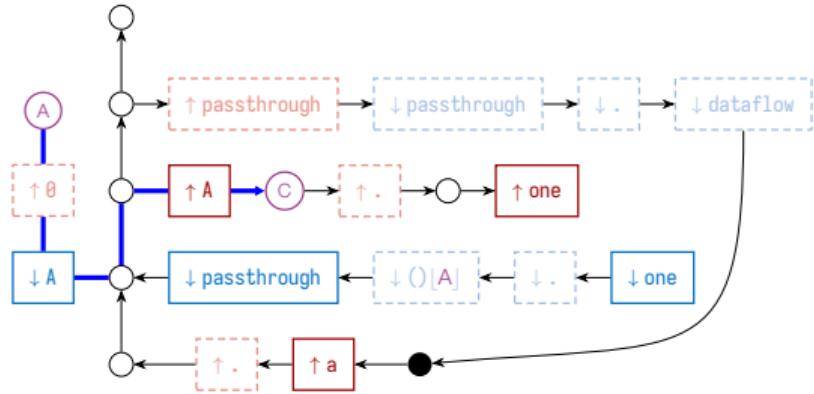


The dataflow example

```
a.py
from dataflow import passthrough

class A:
    one = 1

    passthrough(A).one
```



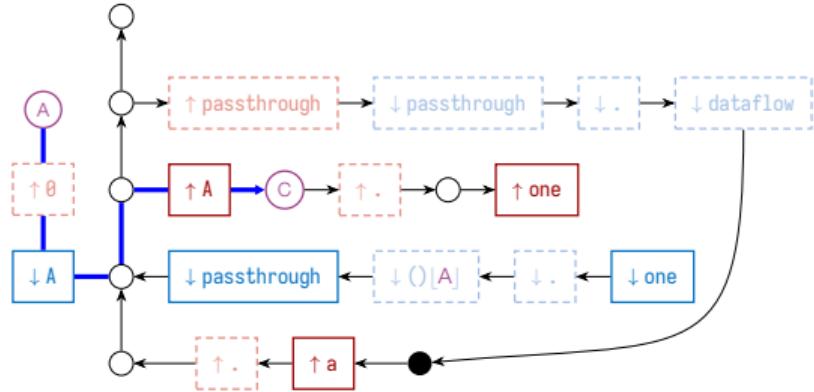
$$\{\langle \theta \rangle \cdot \psi, \phi\} \quad \textcolor{purple}{A} \rightsquigarrow \textcolor{purple}{C} \quad \{\psi, \phi\}$$

The dataflow example

```
a.py
from dataflow import passthrough

class A:
    one = 1

    passthrough(A).one
```

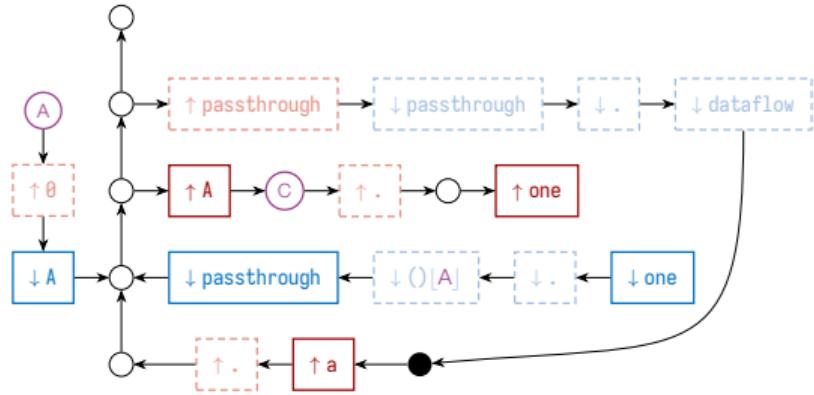


$$\{\langle \theta \rangle \cdot \psi, \phi\} \quad \textcircled{A} \rightsquigarrow \textcircled{C} \quad \{\psi, \phi\}$$

The class A is positional parameter 0
in the call to `dataflow.passthrough`.

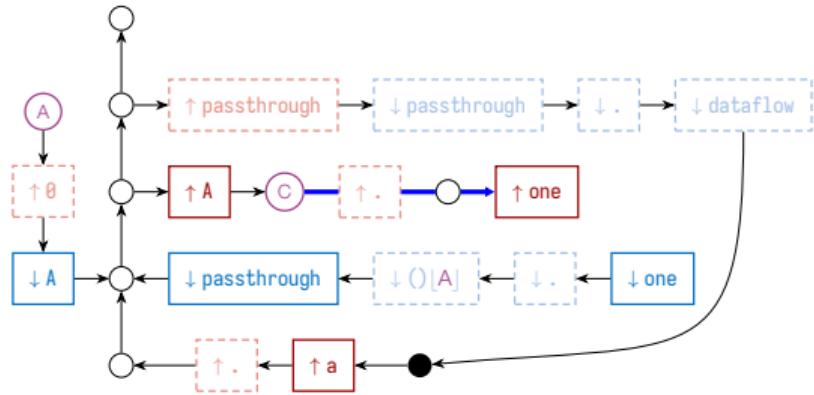
The dataflow example

```
a.py  
from dataflow import passthrough  
  
class A:  
    one = 1  
  
    passthrough(A).one
```



The dataflow example

```
a.py  
from dataflow import passthrough  
  
class A:  
    one = 1  
  
    passthrough(A).one
```

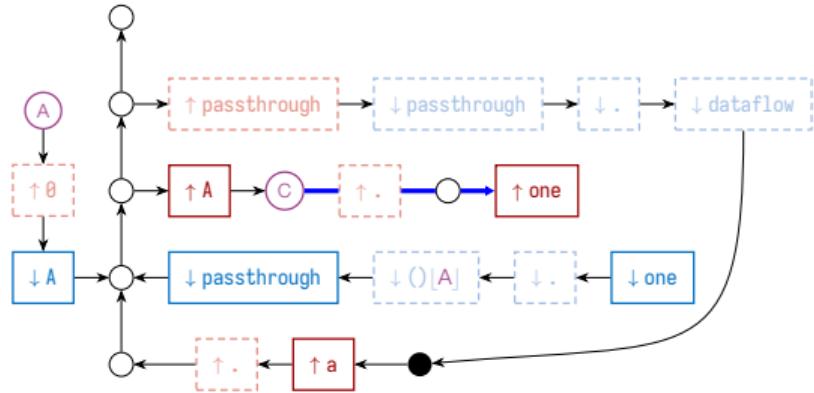


The dataflow example

```
a.py
from dataflow import passthrough

class A:
    one = 1

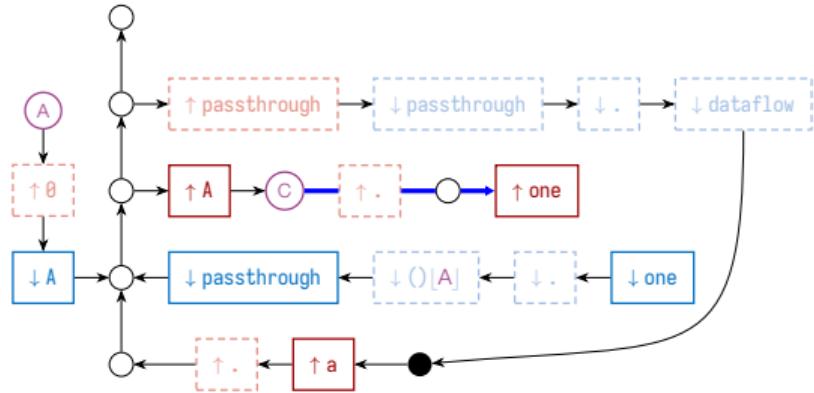
    passthrough(A).one
```



$$\{\langle .\text{one} \rangle \cdot \psi, \phi\} \xrightarrow{C} \boxed{\text{one}} \quad \{\psi, \phi\}$$

The dataflow example

```
a.py  
from dataflow import passthrough  
  
class A:  
    one = 1  
  
    passthrough(A).one
```



$$\{\langle \cdot.\text{one} \rangle \cdot \psi, \phi\} \circledC \rightsquigarrow \boxed{\text{one}} \quad \{\psi, \phi\}$$

The class A has a class member named one
which is defined at a.py:4:5.

The dataflow example

```
{◊, ◊} one ~~~> ● {⟨dataflow.passthrough() [A].one⟩, ◊}
```

If you can find what `dataflow.passthrough` resolves to and can call it, then the result should have a member named `one` which the reference at `a.py:6:16` resolves to.

The dataflow example

$$\{\diamond, \circ\} \xrightarrow{\text{one}} \bullet \{(\text{dataflow.passthrough}()[\text{A}].\text{one}), \circ\} + \{(\text{dataflow.passthrough}([\phi_A]) \cdot \psi, \phi\} \bullet \xrightarrow{\text{R}} \textcolor{pink}{\circ} \{\psi, \phi_A\}$$

If you can find what `dataflow.passthrough` resolves to and can call it, then the result should have a member named `one` which the reference at `a.py:6:16` resolves to.

+

`dataflow.passthrough` is a function that can be invoked.

The dataflow example

$$\{\diamond, \circ\} \xrightarrow{\text{one}} \bullet \{(\text{dataflow.passthrough}([\text{A}].\text{one}), \circ)\} + \{(\text{dataflow.passthrough}([\phi_A]) \cdot \psi, \phi)\} \bullet \xrightarrow{\text{R}} \textcolor{red}{\circ} \{\psi, \phi_A\}$$
$$\psi = \langle .\text{one} \rangle, \phi = \circ, \phi_A = (\text{A})$$

If you can find what `dataflow.passthrough` resolves to and can call it, then the result
should have a member named `one`
which the reference at `a.py:6:16` resolves to.

+

`dataflow.passthrough` is a function
that can be invoked.

The dataflow example

```
{◊, ◊}  ~~~>  {⟨.one⟩, (A)}
```

The result of calling `dataflow.passthrough`
should have a member named `one`
which the reference at `a.py:6:16` resolves to.

The dataflow example

$\{\diamond, \circ\}$  \rightsquigarrow  $\{(\cdot.\text{one}), (\text{A})\}$

+

$\{\psi, \phi\}$  \rightsquigarrow  $\{\langle \theta \rangle \cdot \psi, \phi\}$

The result of calling `dataflow.passthrough`
should have a member named `one`
which the reference at `a.py:6:16` resolves to.

+

The return value of `dataflow.passthrough`
has the same type as positional parameter 0.

The dataflow example

$$\{\diamond, \circ\} \text{ [one]} \rightsquigarrow \text{R} \quad \{\langle .\text{one} \rangle, (\text{A})\} \quad + \quad \{\psi, \phi\} \text{ R} \rightsquigarrow \text{JUMP} \quad \{\langle \theta \rangle \cdot \psi, \phi\}$$
$$\psi = \langle .\text{one} \rangle, \phi = \circ$$

The result of calling `dataflow.passthrough`
should have a member named `one`
which the reference at `a.py:6:16` resolves to.

+

The return value of `dataflow.passthrough`
has the same type as positional parameter 0.

The dataflow example

$\{\diamond, \circ\}$  \rightsquigarrow  $\{\langle \theta.\text{one} \rangle, (\text{A})\}$

Positional parameter 0
should have a member named `one`
which the reference at `a.py:6:16` resolves to.

The dataflow example

$\{\diamond, \circ\}$  \rightsquigarrow  $\{\langle \theta.\text{one} \rangle, (\text{A})\}$



Positional parameter 0
should have a member named `one`
which the reference at `a.py:6:16` resolves to.

Resolve the JUMP node.

The dataflow example

$\{\diamond, \circ\}$  \rightsquigarrow  $\{\langle \theta.\text{one} \rangle, \circ\}$

Positional parameter 0
should have a member named `one`
which the reference at `a.py:6:16` resolves to.

The dataflow example

$\{\diamond, \circ\}$  \rightsquigarrow  $\{\langle \theta.\text{one} \rangle, \circ\}$

+

$\{\langle \theta \rangle \cdot \psi, \phi\}$  \rightsquigarrow  $\{\psi, \phi\}$

Positional parameter 0
should have a member named `one`
which the reference at `a.py:6:16` resolves to.

+

The class `A` is positional parameter 0
in the call to `dataflow.passthrough`.

The dataflow example

$$\{\diamond, \circ\} \xrightarrow{\text{one}} \textcircled{A} \{\langle \theta.\text{one} \rangle, \circ\} + \{\langle \theta \rangle \cdot \psi, \phi\} \xrightarrow{\text{A}} \textcircled{C} \{\psi, \phi\}$$
$$\psi = \langle .\text{one} \rangle, \phi = \circ$$

Positional parameter 0
should have a member named `one`
which the reference at `a.py:6:16` resolves to.

+

The class `A` is positional parameter 0
in the call to `dataflow.passthrough`.

The dataflow example

$\{\diamond, \circ\}$  \rightsquigarrow  $\{(\cdot.\text{one}), \circ\}$

The class A
should have a member named one
which the reference at a.py:6:16 resolves to.

The dataflow example

$\{\diamond, \circ\}$  \rightsquigarrow  $\{(\cdot.\text{one}), \circ\}$

+

$\{(\cdot.\text{one}) \cdot \psi, \phi\}$  \rightsquigarrow  $\{\psi, \phi\}$

The class A
should have a member named `one`
which the reference at `a.py:6:16` resolves to.

+

The class A has a class member named `one`
which is defined at `a.py:4:5`.

The dataflow example

$$\{\diamond, \circ\} \text{ } \boxed{\text{one}} \rightsquigarrow \textcolor{pink}{\odot} \text{ } \{(\cdot.\text{one}), \circ\} + \{(\cdot.\text{one}) \cdot \psi, \phi\} \text{ } \textcolor{pink}{\odot} \rightsquigarrow \boxed{\text{one}} \text{ } \{\psi, \phi\}$$
$$\psi = \diamond, \phi = \circ$$

The class A
should have a member named `one`
which the reference at `a.py:6:16` resolves to.

+

The class A has a class member named `one`
which is defined at `a.py:4:5`.

The dataflow example

```
{◊, ◊} [one] ~~~> [one] {◊, ◊}
```

The definition at *a.py:4:5*
is what the reference at
a.py:6:16 resolves to.

Index

Query

Clone changed files
Parse using tree-sitter
Construct stack graph
Find partial paths

Load partial paths lazily
Stitch them together

Index

- Clone changed files
- Parse using tree-sitter
- Construct stack graph
- Find partial paths

p50: 5 sec
p99: 1-2 min

Query

- Load partial paths lazily
- Stitch them together

p50: 50ms
p99: 100ms

One more for the road

MyMap.java

```
import java.util.HashMap;

class MyMap extends HashMap<String, String> {
    int firstLength() {
        return this.entrySet().iterator()
            .next().getKey().length();
    }
}
```

Picture credits

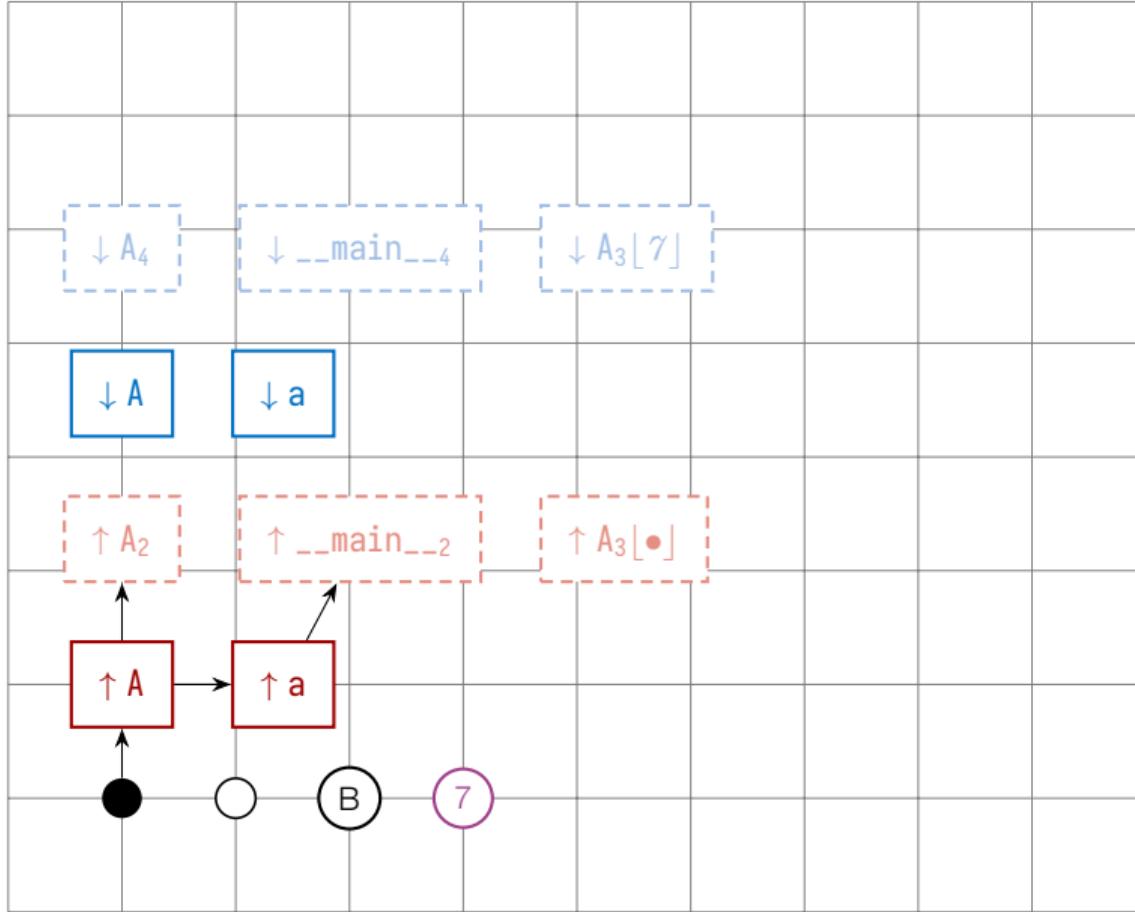
- Slide 3 Ivan Radic, "Close-up of a compass graffiti on the ground"
CC-BY-2.0, <https://flic.kr/p/2kGKmM>
- Slide 5 Mustang Joe, "I swear..."
Public domain, <https://flic.kr/p/VSLwD6>
- Slide 13 Marco Verch, "Close-up, a piece of yellow cake with red currant berries"
CC-BY-2.0, <https://flic.kr/p/2jikJsQ>
- Slide 15 Joseph Gage, "Massive goose gaggle"
CC-BY-SA-2.0, <https://flic.kr/p/2kJfaCt>
- Slide 21 Katja Schulz, "Inchworm"
CC-BY-2.0, <https://flic.kr/p/PJMP4w>
- Slide 24 Marco Verch, "Stack of pancakes with berries on a plate"
CC-BY-2.0, <https://flic.kr/p/2jYUh8M>
- Slide 32 Seattle Municipal Archives, "West Seattle Bridge under construction, circa 1983"
CC-BY-2.0, <https://flic.kr/p/7jKWYi>

Picture credits

- Slide 3 Ivan Radic, "Close-up of a compass graffiti on the ground"
CC-BY-2.0, <https://flic.kr/p/2kGKmT>
- Slide 5 Mustang Joe, "I swear..."
Public domain, <https://flic.kr/p/VSLwD6>
- Slide 13 Marco Verch, "Close-up, a piece of yellow cake with red currant berries"
CC-BY-2.0, <https://flic.kr/p/2jikJsQ>
- Slide 15 Joseph Gage, "Massive goose gaggle"
CC-BY-SA-2.0, <https://flic.kr/p/2kJfaCt>
- Slide 21 Katja Schulz, "Inchworm"
CC-BY-2.0, <https://flic.kr/p/PJMP4w>
- Slide 24 Marco Verch, "Stack of pancakes with berries on a plate"
CC-BY-2.0, <https://flic.kr/p/2jYUh8M>
- Slide 32 Seattle Municipal Archives, "West Seattle Bridge under construction, circa 1983"
CC-BY-2.0, <https://flic.kr/p/7jKWYi>



[github/stack-graphs
tree-sitter/tree-sitter-graph](https://github.com/tree-sitter/tree-sitter-graph)



Are we done?

- ▶ Different languages have different name binding rules.
- ▶ Some of those rules can be quite complex.
- ▶ The result might depend on intermediate files.
- ▶ We don't want to require manual per-repo configuration.
- ▶ We need incremental processing to handle our scale.

Are we done?

- ▶ Different languages have different name binding rules.
- ▶ Some of those rules can be quite complex.
- ▶ The result might depend on intermediate files.
- ▶ We don't want to require manual per-repo configuration.
- ▶ We need incremental processing to handle our scale.

Are we done?

- ▶ Different languages have different name binding rules.
- ▶ Some of those rules can be quite complex.
- ▶ The result might depend on intermediate files.
- ▶ We don't want to require manual per-repo configuration.
- ▶ We need incremental processing to handle our scale.

Are we done?

- ▶ Different languages have different name binding rules.
- ▶ Some of those rules can be quite complex.
- ▶ The result might depend on intermediate files.
- ▶ We don't want to require manual per-repo configuration.
- ▶ We need incremental processing to handle our scale.



Making stack graphs



tree-sitter

tree-sitter

```
stove.py
```

```
def bake():
    pass

def broil():
    pass

def saute():
    pass

broil()
```

tree-sitter

```
(module [0, 0] - [10, 0]
  (function_definition [0, 0] - [1, 8]
    name: (identifier [0, 4] - [0, 8])
    parameters: (parameters [0, 8] - [0, 10])
    body: (block [1, 4] - [1, 8]
      (pass_statement [1, 4] - [1, 8])))
  (function_definition [3, 0] - [4, 8]
    name: (identifier [3, 4] - [3, 9])
    parameters: (parameters [3, 9] - [3, 11])
    body: (block [4, 4] - [4, 8]
      (pass_statement [4, 4] - [4, 8])))
  (function_definition [6, 0] - [7, 8]
    name: (identifier [6, 4] - [6, 9])
    parameters: (parameters [6, 9] - [6, 11])
    body: (block [7, 4] - [7, 8]
      (pass_statement [7, 4] - [7, 8])))
  (expression_statement [9, 0] - [9, 7]
    (call [9, 0] - [9, 7]
      function: (identifier [9, 0] - [9, 5])
      arguments: (argument_list [9, 5] - [9, 7]))))
```

tree-sitter

```
(module [0, 0] - [10, 0]
  (function_definition [0, 0] - [1, 8]
    name: (identifier [0, 4] - [0, 8])
    parameters: (parameters [0, 8] - [0, 10])
    body: (block [1, 4] - [1, 8]
      (pass_statement [1, 4] - [1, 8])))
  (function_definition [3, 0] - [4, 8]
    name: (identifier [3, 4] - [3, 9])
    parameters: (parameters [3, 9] - [3, 11])
    body: (block [4, 4] - [4, 8]
      (pass_statement [4, 4] - [4, 8])))
  (function_definition [6, 0] - [7, 8]
    name: (identifier [6, 4] - [6, 9])
    parameters: (parameters [6, 9] - [6, 11])
    body: (block [7, 4] - [7, 8]
      (pass_statement [7, 4] - [7, 8])))
  (expression_statement [9, 0] - [9, 7]
    (call [9, 0] - [9, 7]
      function: (identifier [9, 0] - [9, 5])
      arguments: (argument_list [9, 5] - [9, 7]))))
```

```
(function_definition
  name: (identifier @name) @function
```

tree-sitter

```
(module [0, 0] - [10, 0]
  (function_definition [0, 0] - [1, 8]
    name: (identifier [0, 4] - [0, 8])
    parameters: (parameters [0, 8] - [0, 10])
    body: (block [1, 4] - [1, 8]
      (pass_statement [1, 4] - [1, 8])))
  (function_definition [3, 0] - [4, 8]
    name: (identifier [3, 4] - [3, 9])
    parameters: (parameters [3, 9] - [3, 11])
    body: (block [4, 4] - [4, 8]
      (pass_statement [4, 4] - [4, 8])))
  (function_definition [6, 0] - [7, 8]
    name: (identifier [6, 4] - [6, 9])
    parameters: (parameters [6, 9] - [6, 11])
    body: (block [7, 4] - [7, 8]
      (pass_statement [7, 4] - [7, 8])))
  (expression_statement [9, 0] - [9, 7]
    (call [9, 0] - [9, 7]
      function: (identifier [9, 0] - [9, 5])
      arguments: (argument_list [9, 5] - [9, 7]))))
```

(function_definition
 name: (identifier @name) @function

tree-sitter

```
(module [0, 0] - [10, 0]
  (function_definition [0, 0] - [1, 8]
    name: (identifier [0, 4] - [0, 8])
    parameters: (parameters [0, 8] - [0, 10])
    body: (block [1, 4] - [1, 8]
      (pass_statement [1, 4] - [1, 8])))
  (function_definition [3, 0] - [4, 8]
    name: (identifier [3, 4] - [3, 9])
    parameters: (parameters [3, 9] - [3, 11])
    body: (block [4, 4] - [4, 8]
      (pass_statement [4, 4] - [4, 8])))
  (function_definition [6, 0] - [7, 8]
    name: (identifier [6, 4] - [6, 9])
    parameters: (parameters [6, 9] - [6, 11])
    body: (block [7, 4] - [7, 8]
      (pass_statement [7, 4] - [7, 8])))
  (expression_statement [9, 0] - [9, 7]
    (call [9, 0] - [9, 7]
      function: (identifier [9, 0] - [9, 5])
      arguments: (argument_list [9, 5] - [9, 7]))))
```

```
(function_definition
  name: (identifier) @name) @function
```

tree-sitter

```
(module [0, 0] - [10, 0]
  (function_definition [0, 0] - [1, 8]
    name: (identifier [0, 4] - [0, 8])
    parameters: (parameters [0, 8] - [0, 10])
    body: (block [1, 4] - [1, 8]
      (pass_statement [1, 4] - [1, 8])))
  (function_definition [3, 0] - [4, 8]
    name: (identifier [3, 4] - [3, 9])
    parameters: (parameters [3, 9] - [3, 11])
    body: (block [4, 4] - [4, 8]
      (pass_statement [4, 4] - [4, 8])))
  (function_definition [6, 0] - [7, 8]
    name: (identifier [6, 4] - [6, 9])
    parameters: (parameters [6, 9] - [6, 11])
    body: (block [7, 4] - [7, 8]
      (pass_statement [7, 4] - [7, 8])))
  (expression_statement [9, 0] - [9, 7]
    (call [9, 0] - [9, 7]
      function: (identifier [9, 0] - [9, 5])
      arguments: (argument_list [9, 5] - [9, 7]))))
```

```
(function_definition
  name: (identifier @name) @function
{
  node @function.def
  attr (@function.def) kind = "definition"
  attr (@function.def) symbol = @name
}
edge @function.containing_scope → @function.def
```

tree-sitter

```
(module [0, 0] - [10, 0]
  (function_definition [0, 0] - [1, 8]
    name: (identifier [0, 4] - [0, 8])
    parameters: (parameters [0, 8] - [0, 10])
    body: (block [1, 4] - [1, 8]
      (pass_statement [1, 4] - [1, 8])))
  (function_definition [3, 0] - [4, 8]
    name: (identifier [3, 4] - [3, 9])
    parameters: (parameters [3, 9] - [3, 11])
    body: (block [4, 4] - [4, 8]
      (pass_statement [4, 4] - [4, 8])))
  (function_definition [6, 0] - [7, 8]
    name: (identifier [6, 4] - [6, 9])
    parameters: (parameters [6, 9] - [6, 11])
    body: (block [7, 4] - [7, 8]
      (pass_statement [7, 4] - [7, 8])))
  (expression_statement [9, 0] - [9, 7]
    (call [9, 0] - [9, 7]
      function: (identifier [9, 0] - [9, 5])
      arguments: (argument_list [9, 5] - [9, 7]))))
```

```
(function_definition
  name: (identifier @name) @function
{
  node @function.def
  attr (@function.def) kind = "definition"
  attr (@function.def) symbol = @name
}
edge @function.containing_scope → @function.def
```





github/stack-graphs
tree-sitter/tree-sitter
tree-sitter/tree-sitter-graph



github/stack-graphs
tree-sitter/tree-sitter
tree-sitter/tree-sitter-graph

-  tree-sitter/tree-sitter-python
-  tree-sitter/tree-sitter-javascript
-  tree-sitter/tree-sitter-rust
-  tree-sitter/tree-sitter-ruby
-  elixir-lang/tree-sitter-elixir
-  r-lib/tree-sitter-r

⋮

Extras

```
stove.rs
fn broil() {}
fn broil() {}
fn saute() {}
```

