# Incremental, zero-config Code Navigation using stack graphs

Douglas Creager
@dcreager

Languages, Systems, and Data Seminar

May 27, 2021 – UC Santa Cruz

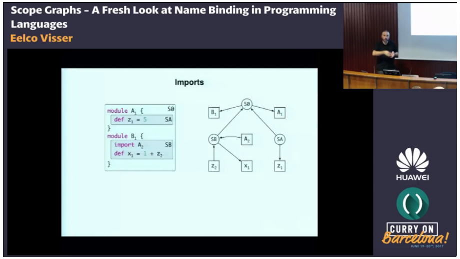
Builds on the Scope Graphs framework
from Eelco Visser's group at TU Delft.

`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



Scope Graphs – A Fresh Look at Name Binding in Programming Languages
Eelco Visser

Imports

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

## Code Navigation

## Code Navigation

stove.py

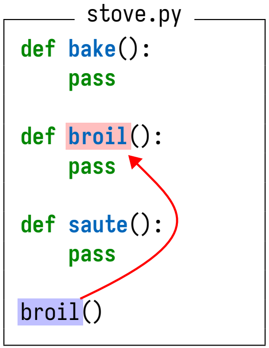
```python
def bake():
    pass

def broil():
    pass

def saute():
    pass

broil()
```

## Code Navigation

stove.py

```python
def bake():
    pass

def broil():
    pass

def saute():
    pass

broil()
```

# Code Navigation



stove.py

```
def bake():
    pass

def broil():
    pass

def saute():
    pass

broil()
```

## Why is this hard?

## Why is this hard?

stove.py

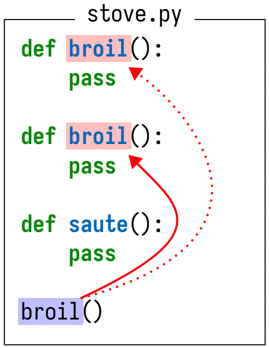
```python
def broil():
    pass

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

```python
def broil():
    pass

def broil():
    pass

def saute():
    pass

broil()
```

## Why is this hard?

stove.rs

```rust
fn broil() {}

fn broil() {}

fn saute() {}

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

## Why is this hard?

stove.rs

```rust
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.py

```python
def bake():
    pass

def broil():
    pass

def saute():
    pass
```

kitchen.py

```python
from stove import broil

broil()
```

## Why is this hard?

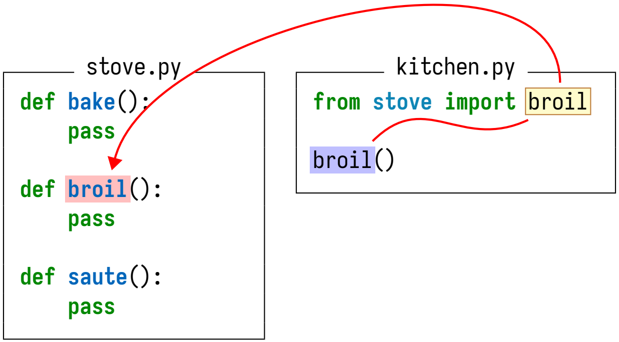
stove.py

```python
def bake():
    pass

def broil():
    pass

def saute():
    pass
```

kitchen.py

```python
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?

stove.py

```python
def bake():
    pass

def broil():
    pass

def saute():
    pass
```

kitchen.py

```python
from stove import *
```

chef.py

```python
from kitchen import broil

broil()
```

## Why is this hard?

stove.py

```python
def bake():
    pass

def broil():
    pass

def saute():
    pass
```

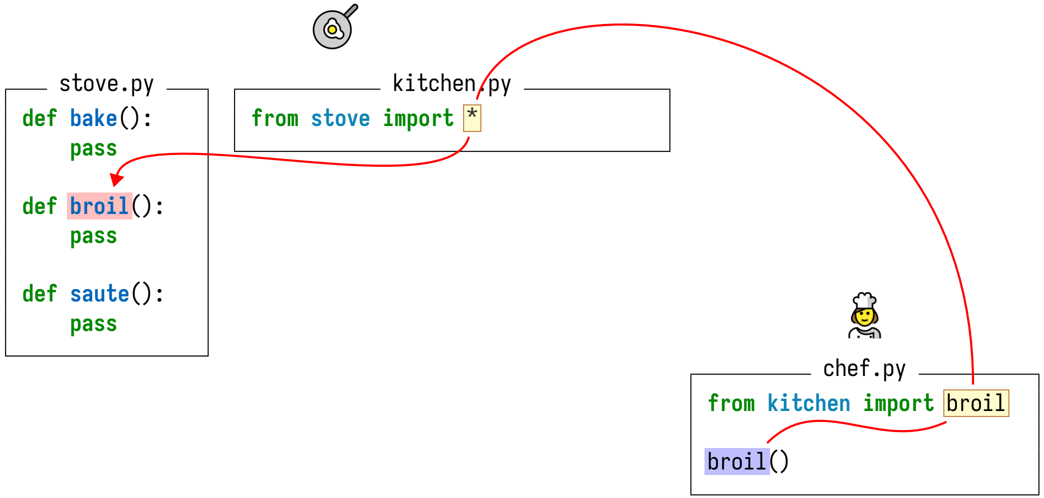
kitchen.py

```python
from stove import *
```

chef.py

```python
from kitchen import broil

broil()
```

## Why is this hard?


stove.py

```python
def bake():
    pass

def broil():
    pass

def saute():
    pass
```

kitchen.py

```python
from stove import *
```

chef.py

```python
from kitchen import broil

broil()
```

# Why is this hard?

stove.py

```python
def bake():
    pass

def broil():
    pass

def saute():
    pass
```

kitchen.py

```python
from stove import *

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

chef.py

```python
from kitchen import broil

broil()
```

## Why is this hard?

stove.py

```python
def bake():
    pass

def broil():
    pass

def saute():
    pass
```

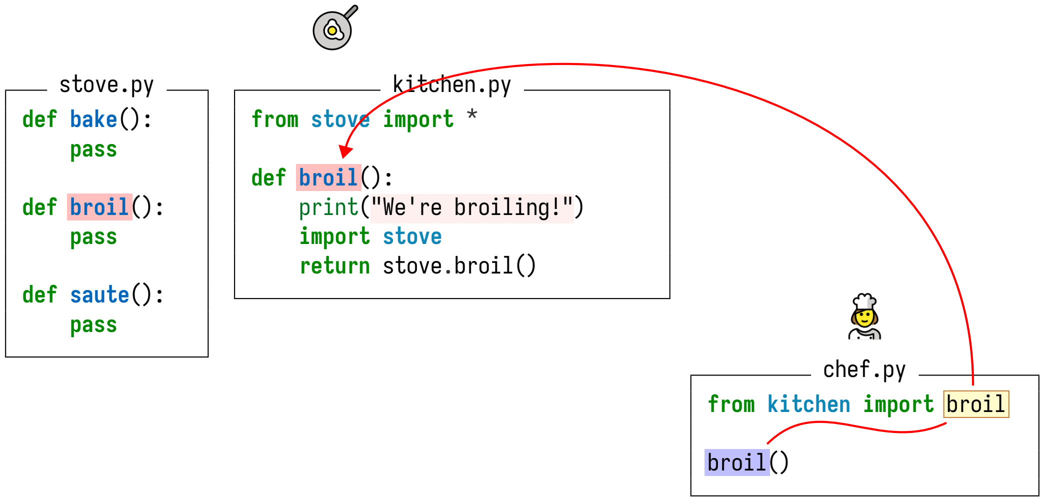
kitchen.py

```python
from stove import *

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

chef.py

```python
from kitchen import broil

broil()
```

## Why is this hard?



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

def broil():
    pass

def saute():
    pass
```

kitchen.py
```python
from stove import *

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

chef.py
```python
from kitchen import broil

broil()
```

## Why is this hard?

stove.py

```python
class Stove(object):
    def bake(self):
        pass

    def broil(self):
        pass

    def saute(self):
        pass
```

kitchen.py

```python
from stove import *
```

chef.py

```python
from kitchen import Stove

stove = Stove()
stove.broil()
```

## Why is this hard?

stove.py

```python
class Stove(object):
    def bake(self):
        pass

    def broil(self):
        pass

    def saute(self):
        pass
```

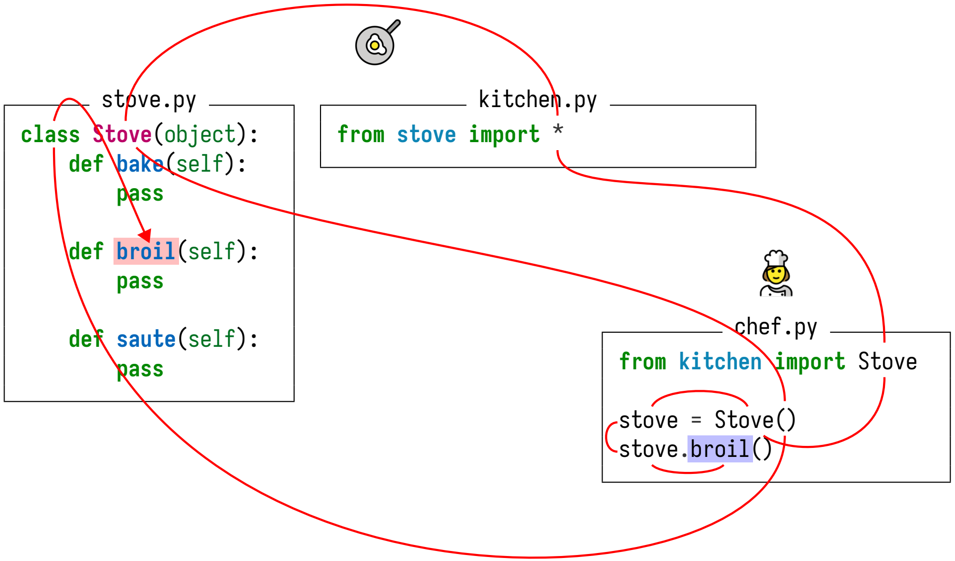
kitchen.py

```python
from stove import *
```

chef.py

```python
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?

dataflow.py

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

a.py

```python
from dataflow import passthrough

class A:
    one = 1

passthrough(A).one
```

## Why is this hard?

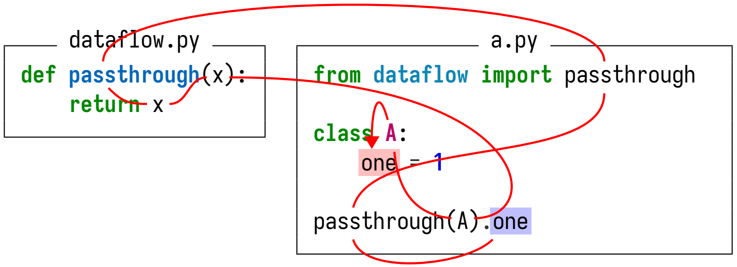
dataflow.py

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

a.py

```python
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
```

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**.

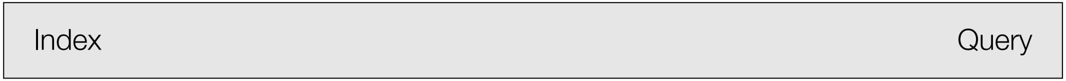
## SCALE

200 million repositories and counting

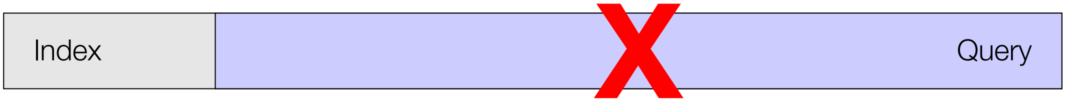
2 billion contributions
in the last 12 months

500 programming languages

## When do we do the work?



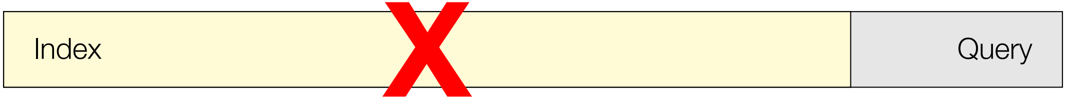
## When do we do the work?



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

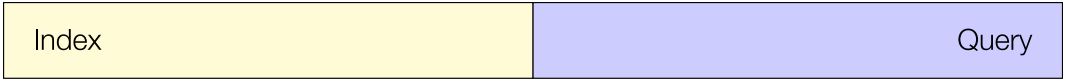
Goal: < 100ms

## When do we do the work?



Because of our scale, we can’t doo 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

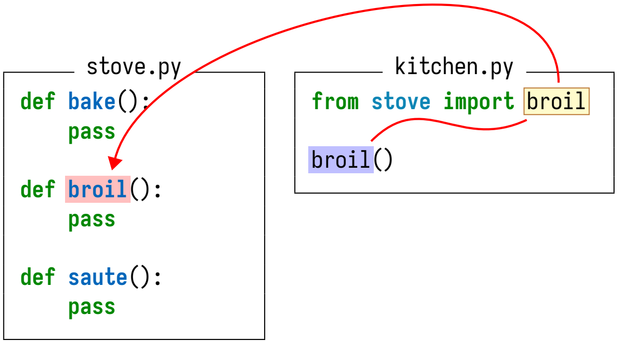
In a typical commit, a small fraction of files in the repo change.

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?



stove.py

```
def bake():
    pass

def broil():
    pass

def saute():
    pass
```

kitchen.py

```
from stove import broil

broil()
```

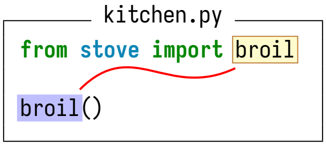
## What would incremental results look like?

stove.py

```
def bake():
    pass

def broil():
    pass

def saute():
    pass
```

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

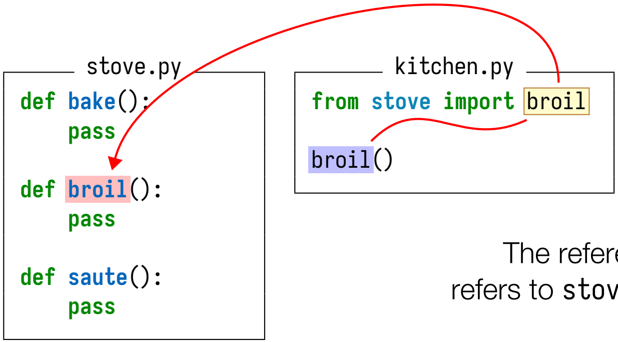
## What would incremental results look like?



```
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?


stove.py

```
def bake():
    pass

def broil():
    pass

def saute():
    pass
```

kitchen.py

```
from stove import broil

broil()
```

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

+

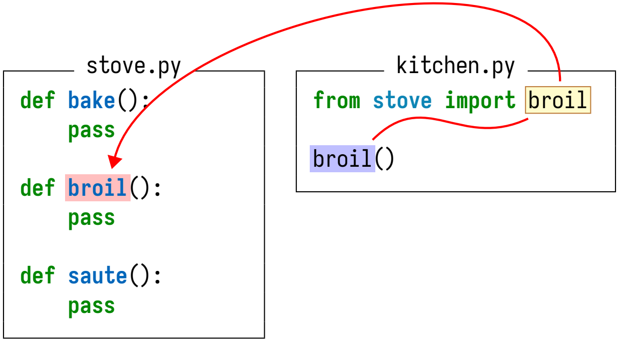
`stove.broil` is defined at *stove.py:4:5*

=

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

## Stack graphs

# Stack graphs



```
def bake():
    pass

def broil():
    pass

def saute():
    pass
```

```
from stove import broil

broil()
```

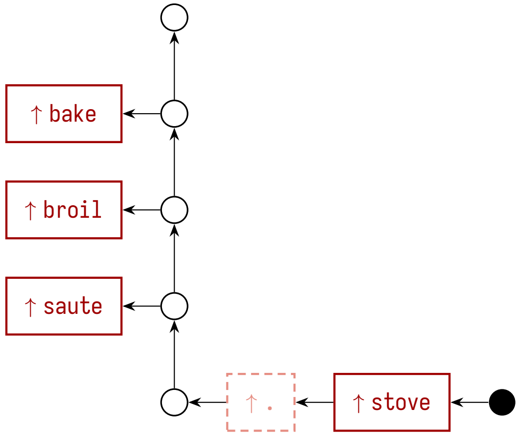
## Stack graphs

stove.py

```python
def bake():
    pass

def broil():
    pass

def saute():
    pass
```

# Stack graphs

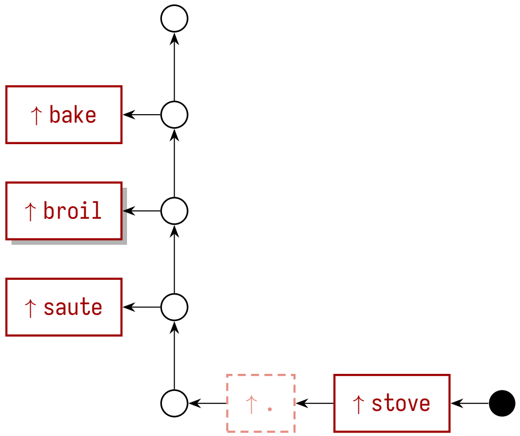
```
stove.py
def bake():
    pass

def broil():
    pass

def saute():
    pass
```



## Stack graphs

stove.py

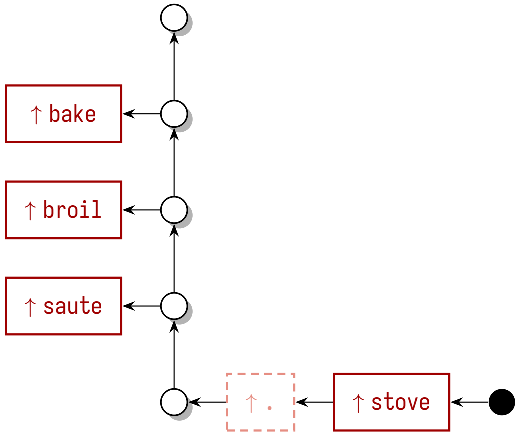
```
def bake():
    pass

def broil():
    pass

def saute():
    pass
```



## Stack graphs

stove.py

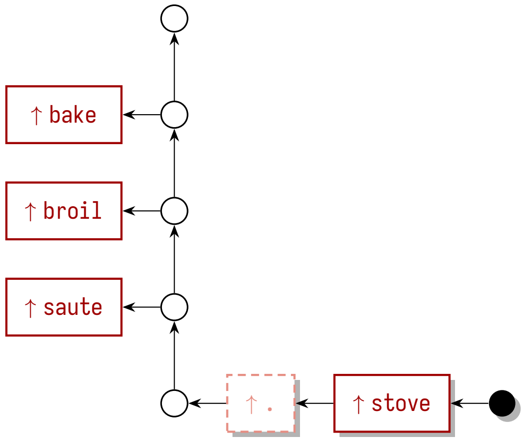
```
def bake():
    pass

def broil():
    pass

def saute():
    pass
```



# Stack graphs

stove.py

```
def bake():
    pass

def broil():
    pass

def saute():
    pass
```



## Stack graphs

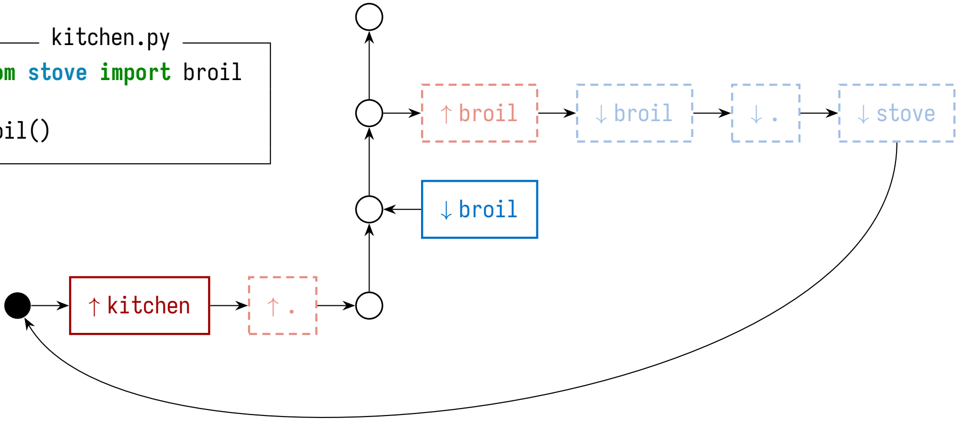
kitchen.py

```
from stove import broil

broil()
```

## Stack graphs

kitchen.py

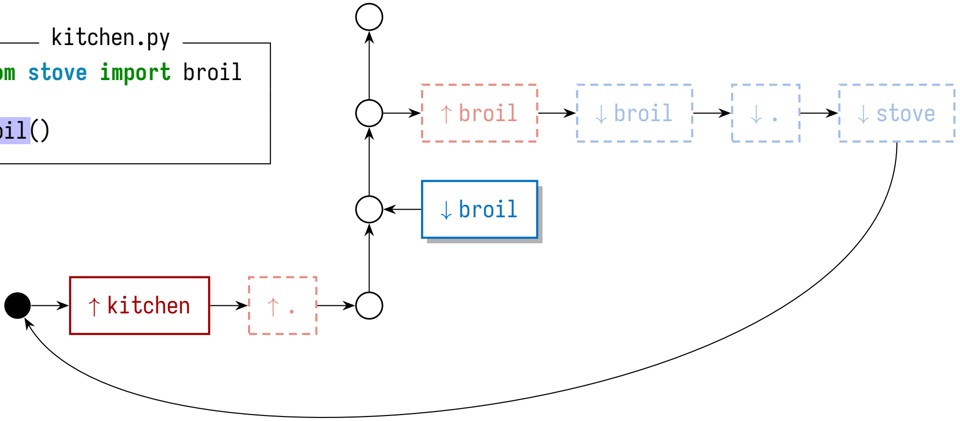
```
from stove import broil

broil()
```



# Stack graphs

kitchen.py

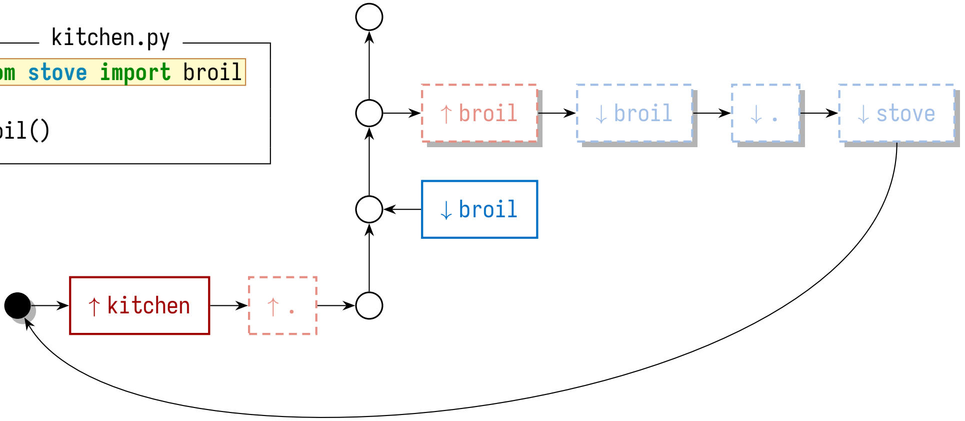
```
from stove import broil

broil()
```



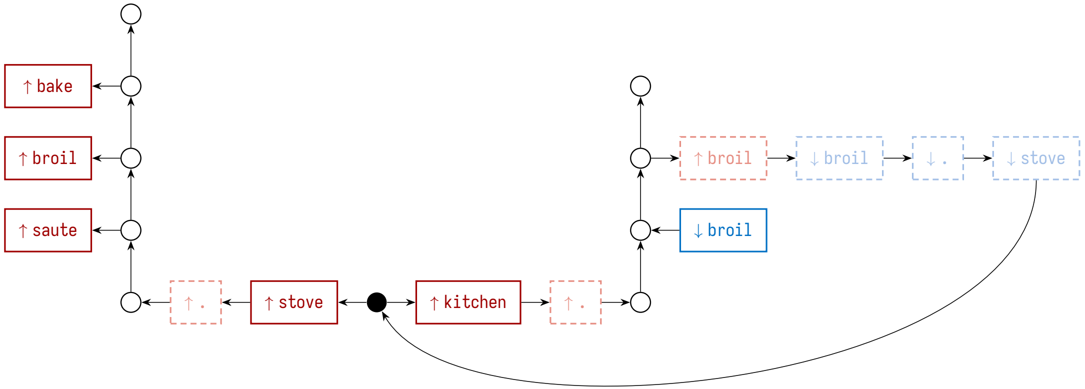
# Stack graphs

kitchen.py

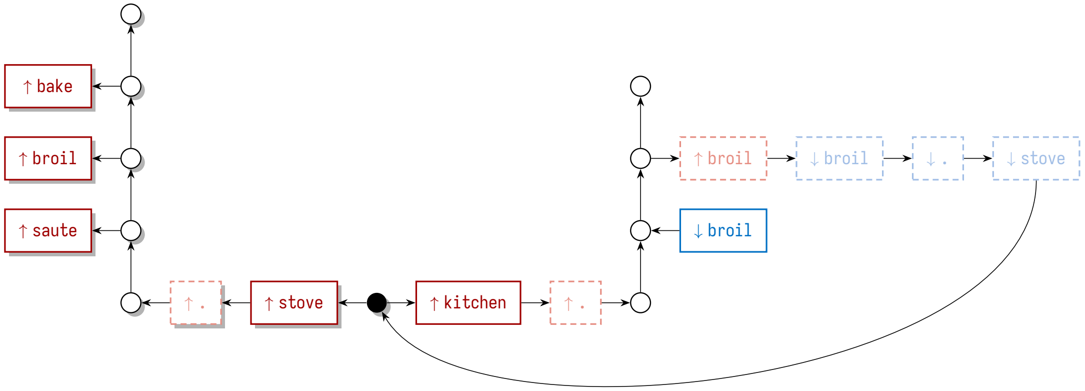
```
from stove import broil

broil()
```



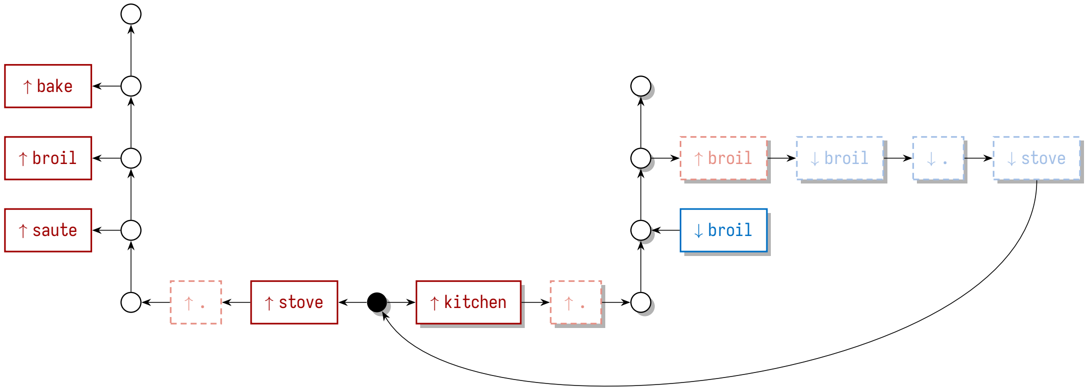
# Stack graphs



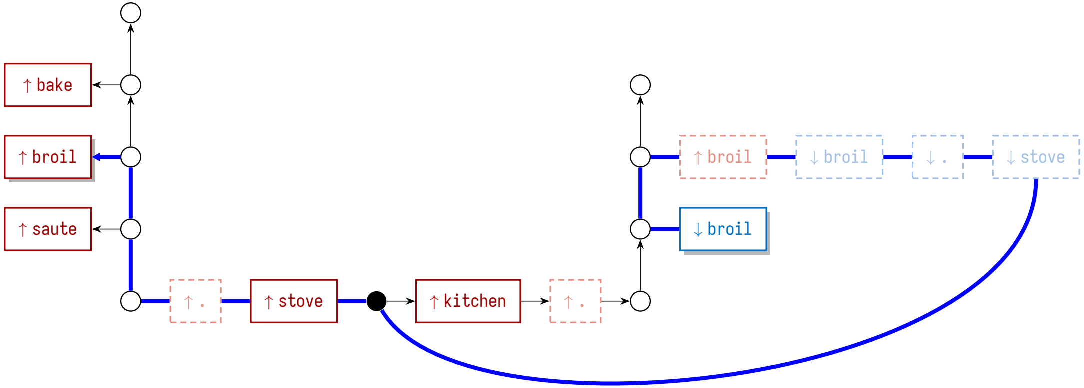
# Stack graphs



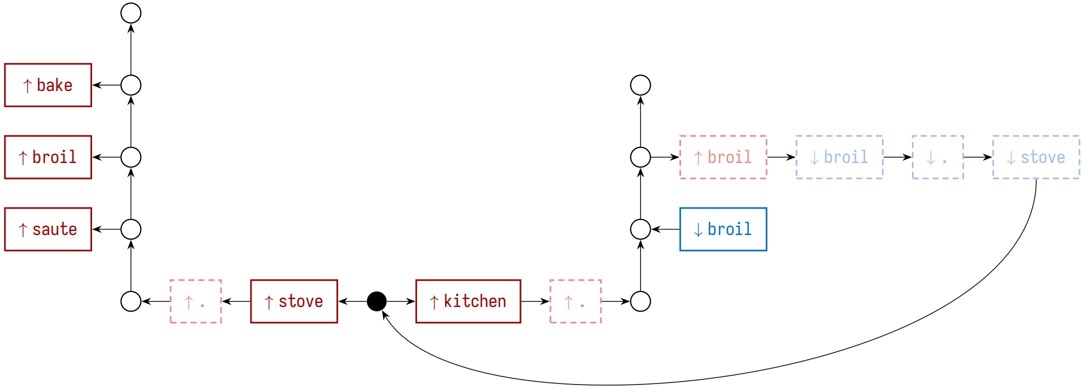
## Stack graphs



# Stack graphs

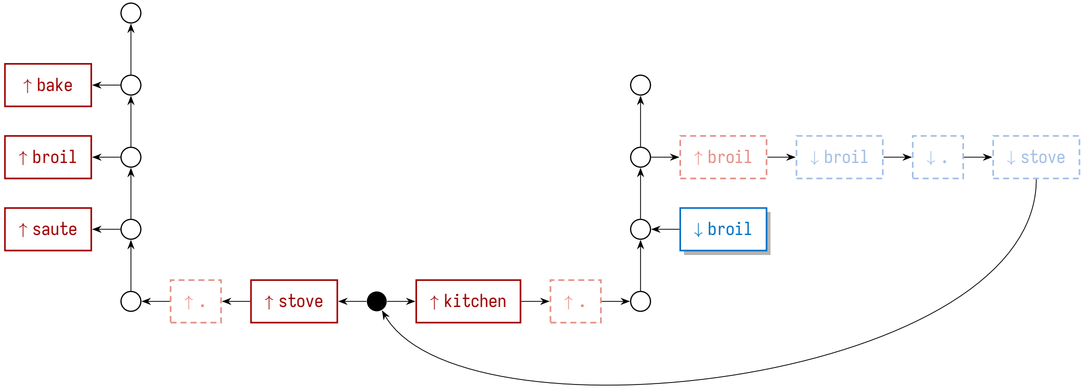


# Stack graphs



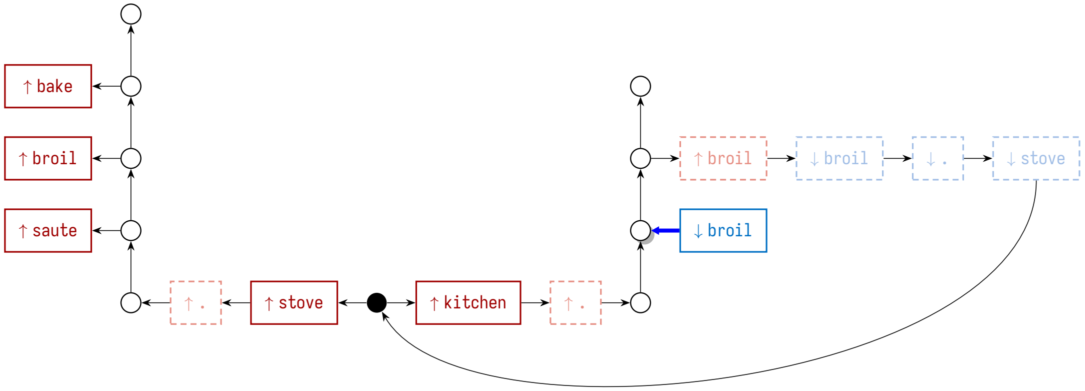
Symbol stack: ◇

# Stack graphs



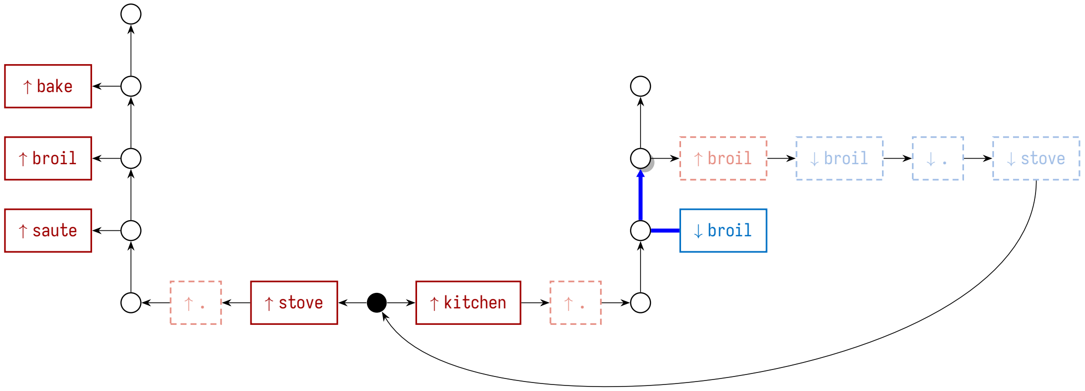
Symbol stack: ⟨broil⟩

# Stack graphs



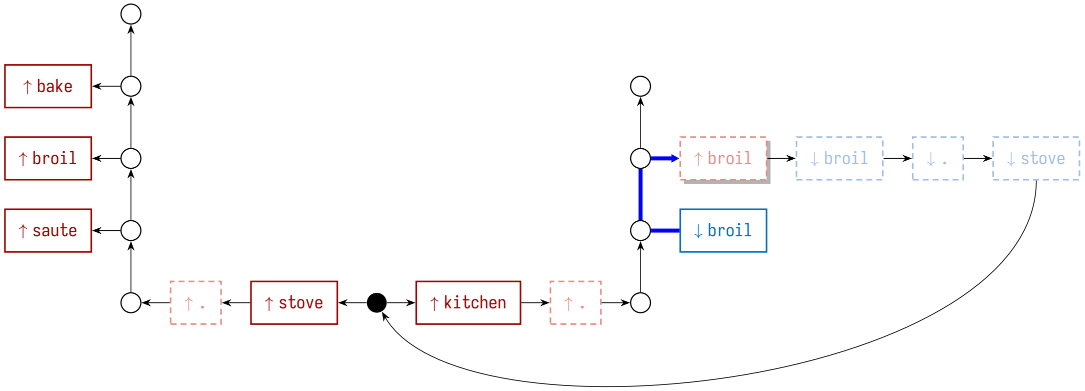
Symbol stack: ⟨broil⟩

# Stack graphs



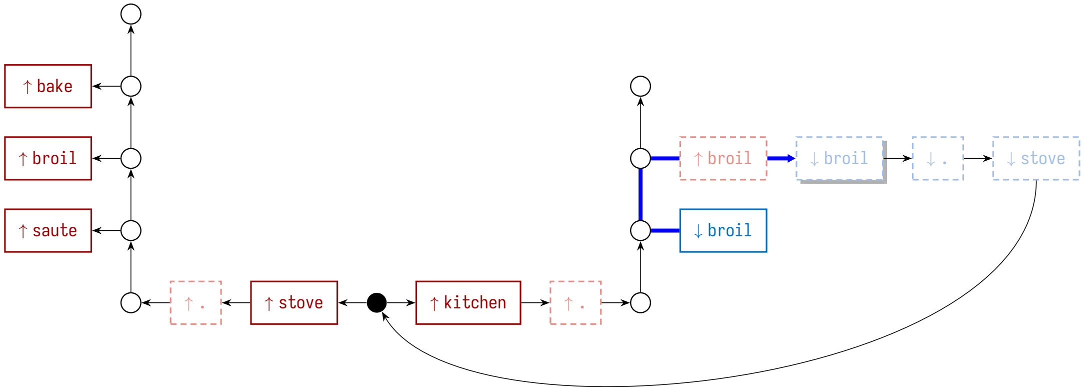
Symbol stack: ⟨broil⟩

# Stack graphs



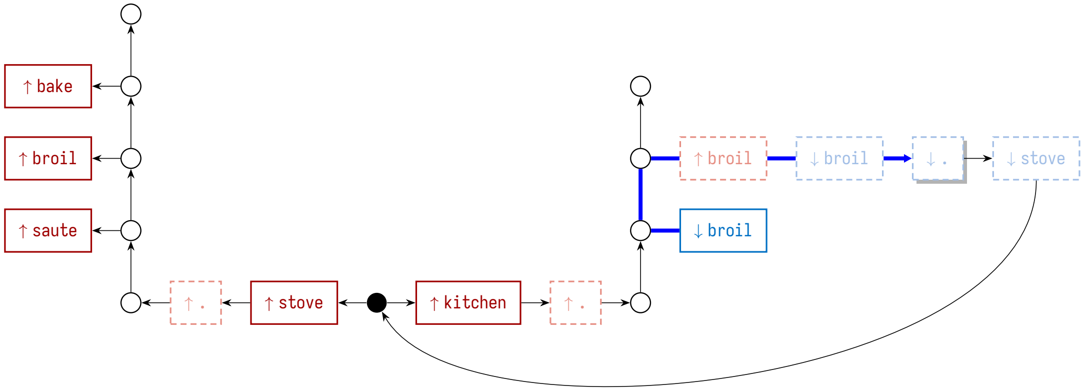
Symbol stack: ◇

# Stack graphs



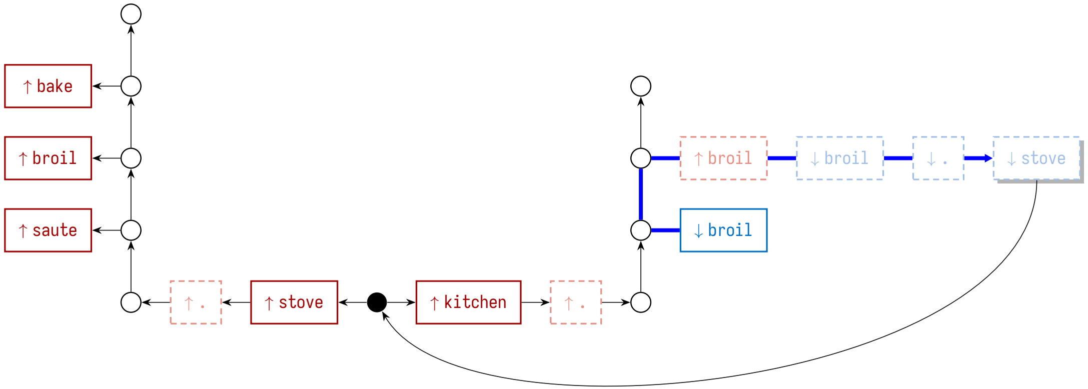
Symbol stack: ⟨`broil`⟩

# Stack graphs



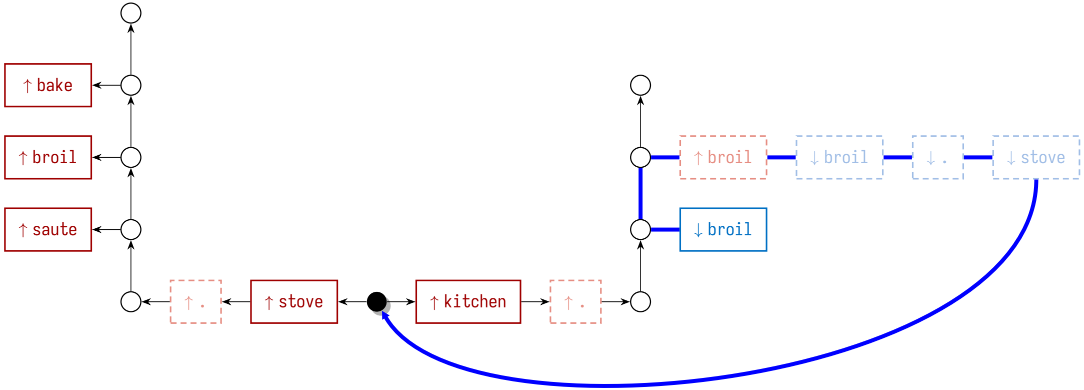
Symbol stack: ⟨.broil⟩

# Stack graphs



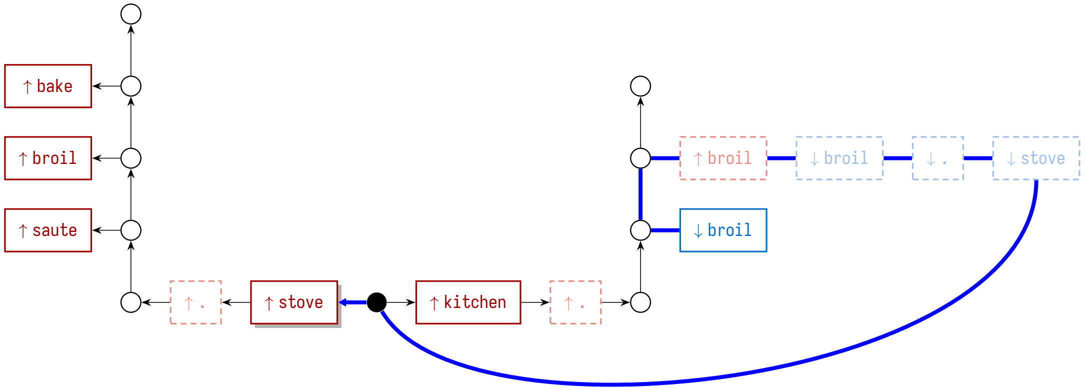
Symbol stack: ⟨`stove.broil`⟩

# Stack graphs



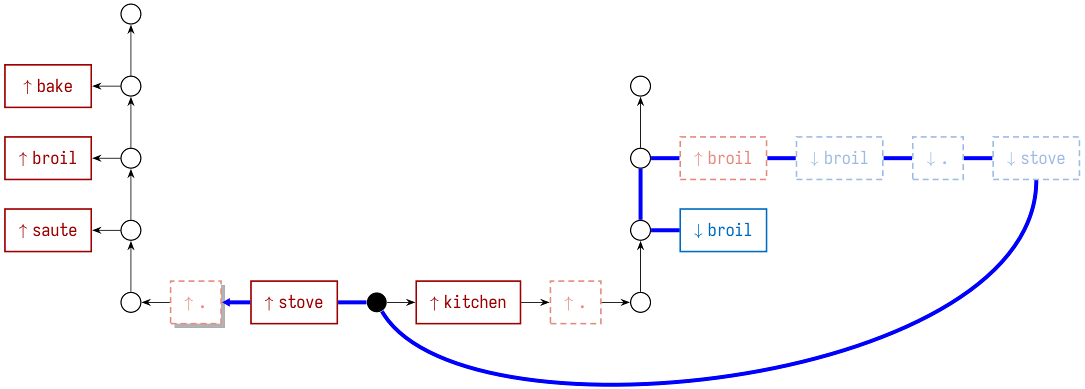
Symbol stack: ⟨stove.broil⟩

# Stack graphs



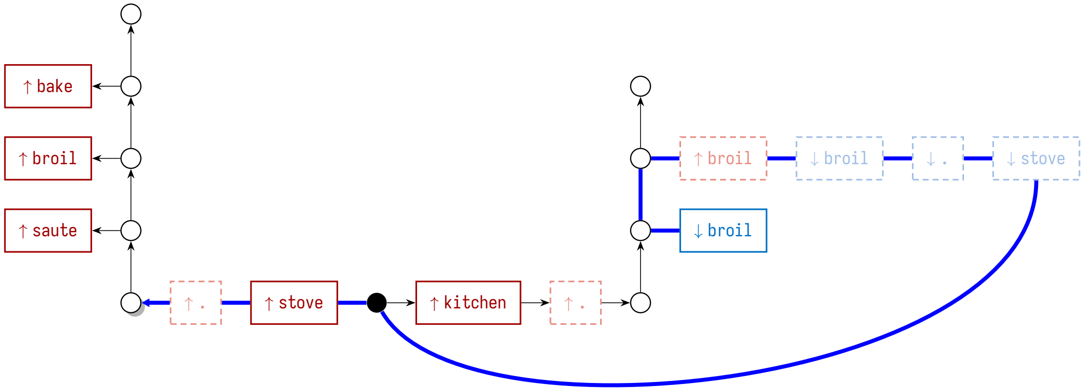
Symbol stack: ⟨.broil⟩

# Stack graphs



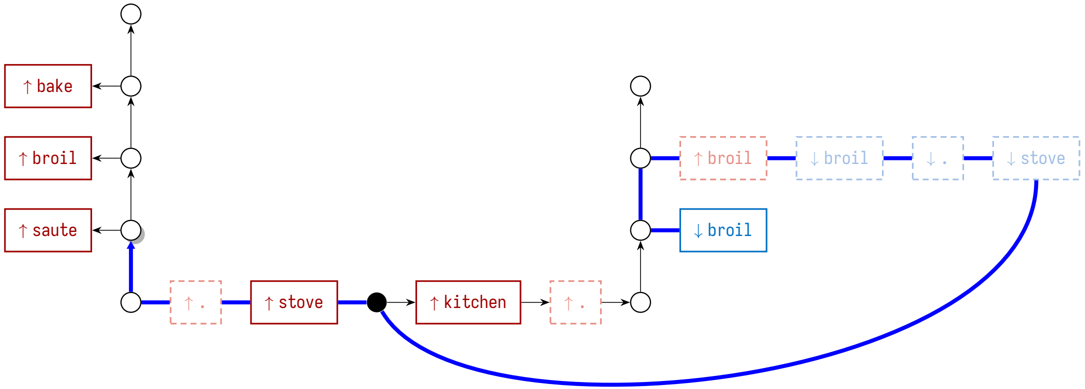
Symbol stack: ⟨broil⟩

# Stack graphs



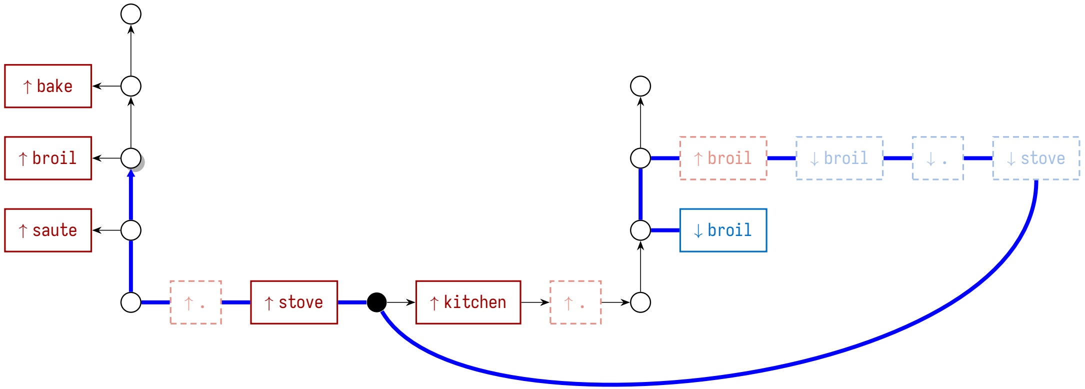
Symbol stack: ⟨broil⟩

# Stack graphs



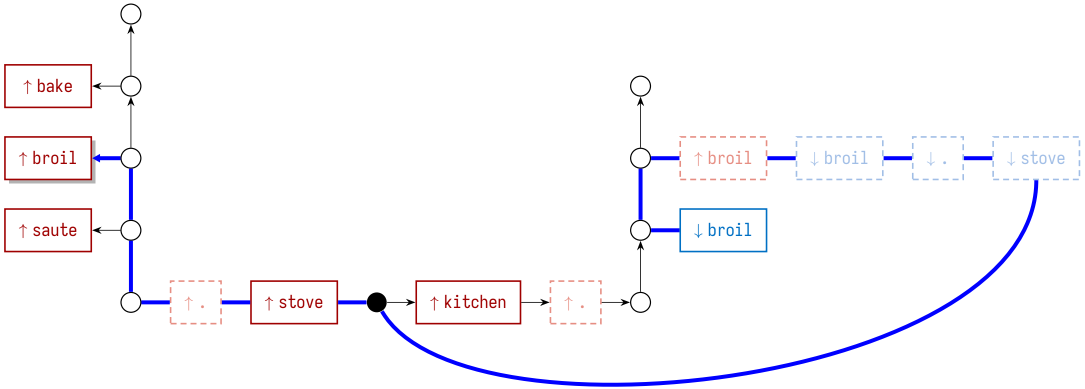
Symbol stack: ⟨broil⟩

# Stack graphs



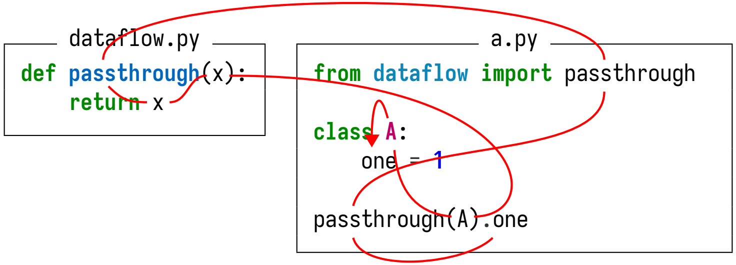
Symbol stack: ⟨`broil`⟩

# Stack graphs



Symbol stack: ◇

## The dataflow example



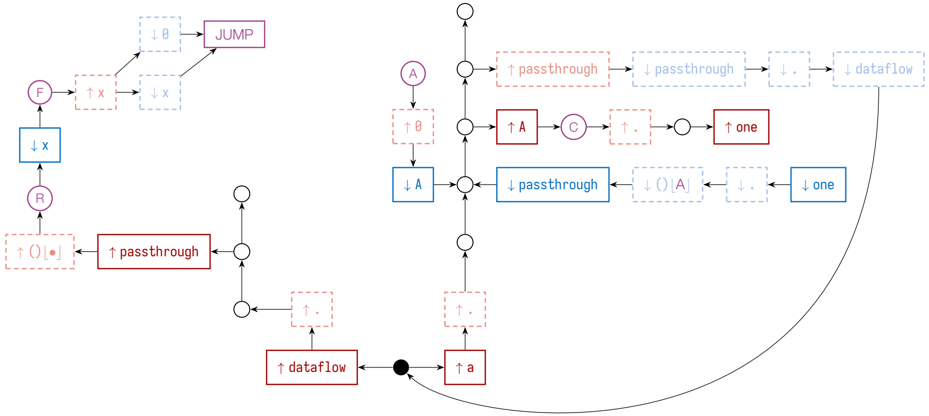
dataflow.py

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

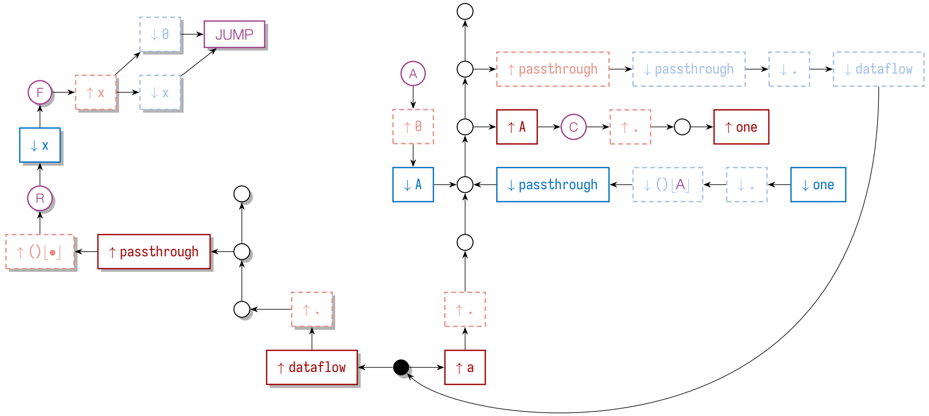
a.py

```
from dataflow import passthrough

class A:
    one = 1

passthrough(A).one
```

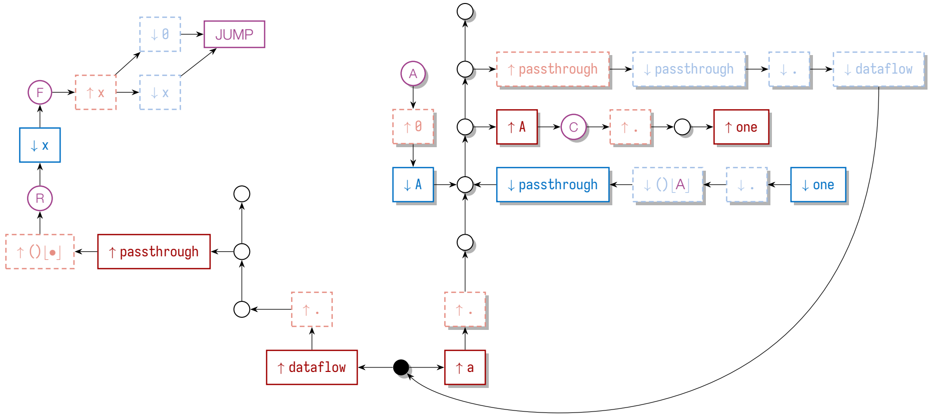
# The dataflow example



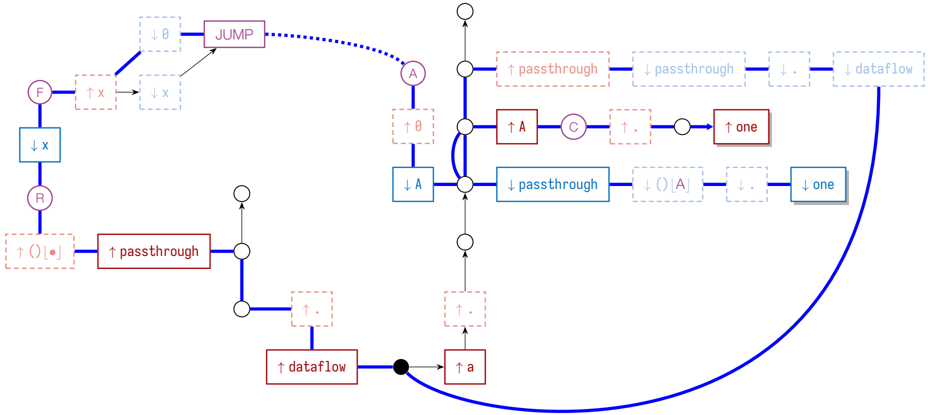
# The dataflow example



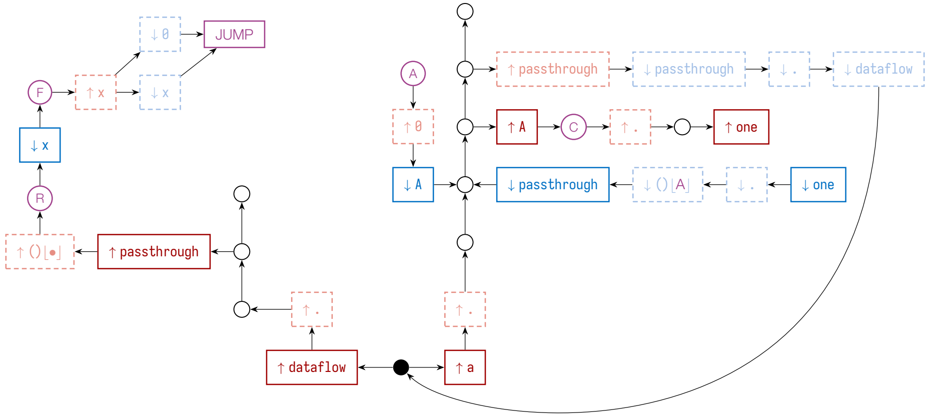
# The dataflow example



## The dataflow example



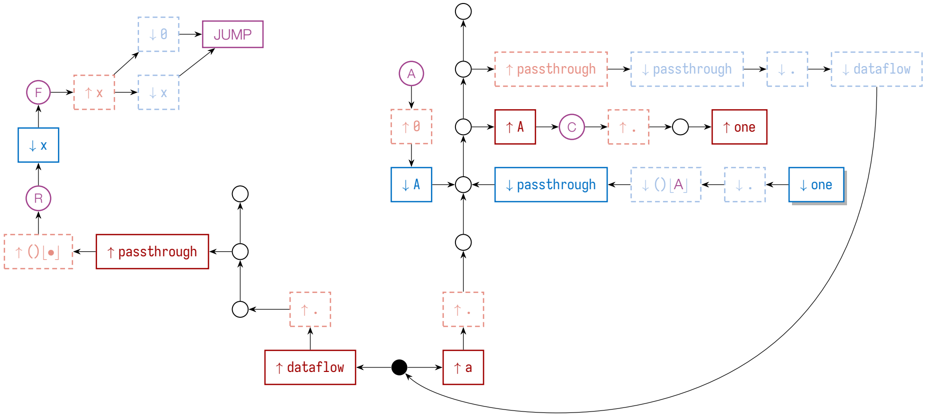
# The dataflow example



Symbol stack: ◇

Scope stack: ○

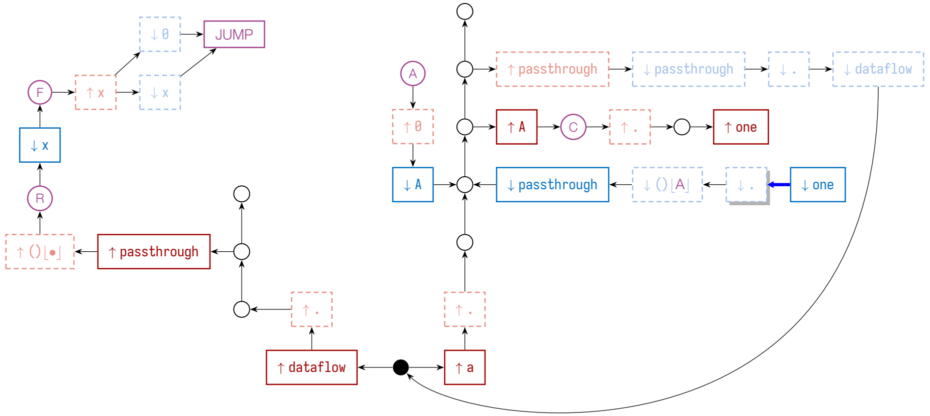
# The dataflow example



Symbol stack: ⟨one⟩

Scope stack: ○

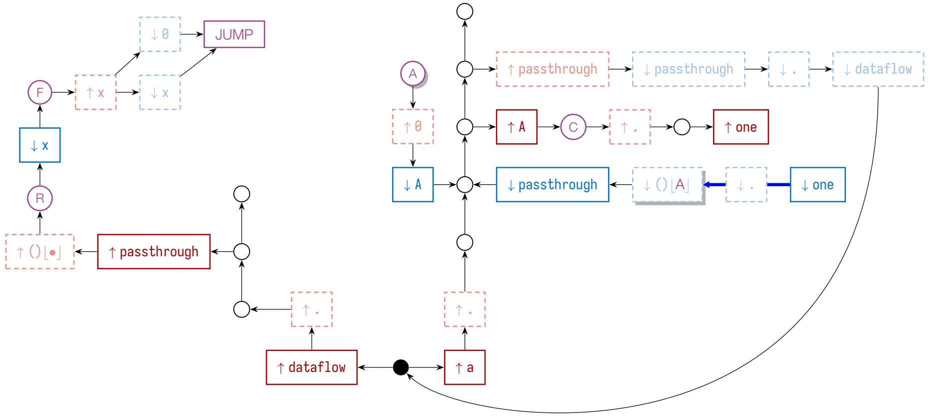
# The dataflow example



Symbol stack: ⟨`.one`⟩

Scope stack: ○

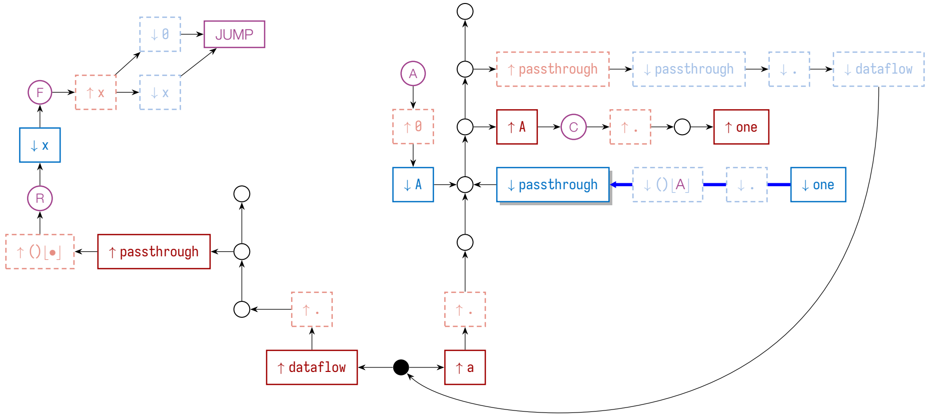
# The dataflow example



Symbol stack: ⟨()⌊A⌋.one⟩

Scope stack: ○

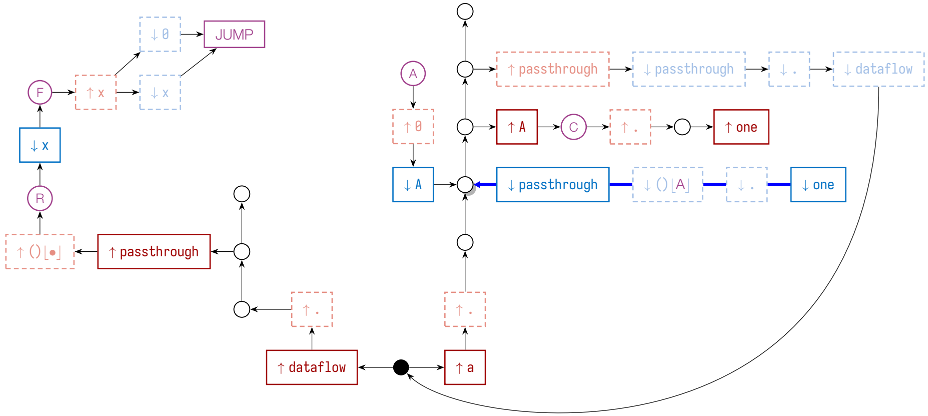
# The dataflow example



Symbol stack: ⟨`passthrough()`⌊A⌋`.one`⟩

Scope stack: ○

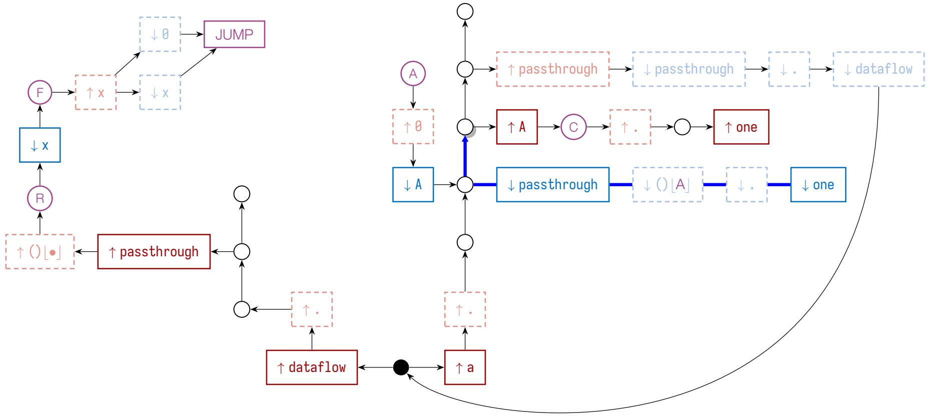
# The dataflow example



Symbol stack: ⟨passthrough()⌊A⌋.one⟩

Scope stack: ○

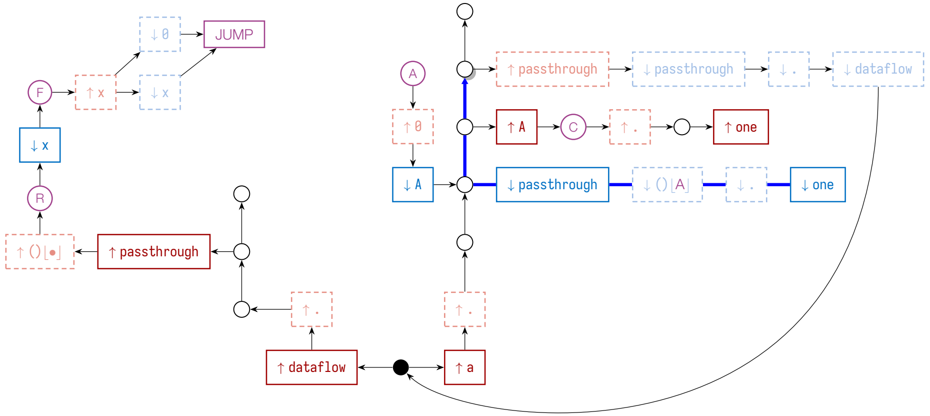
# The dataflow example



Symbol stack: ⟨passthrough()⌊A⌋.one⟩

Scope stack: ○

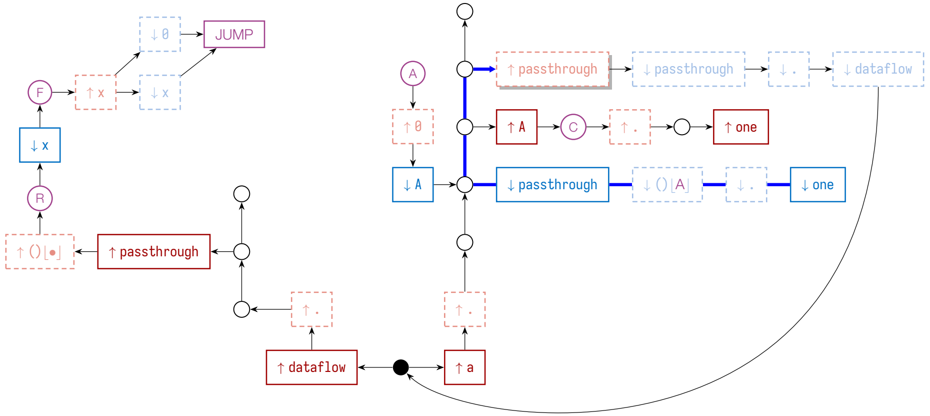
# The dataflow example



Symbol stack: ⟨passthrough()⌊A⌋.one⟩

Scope stack: ○

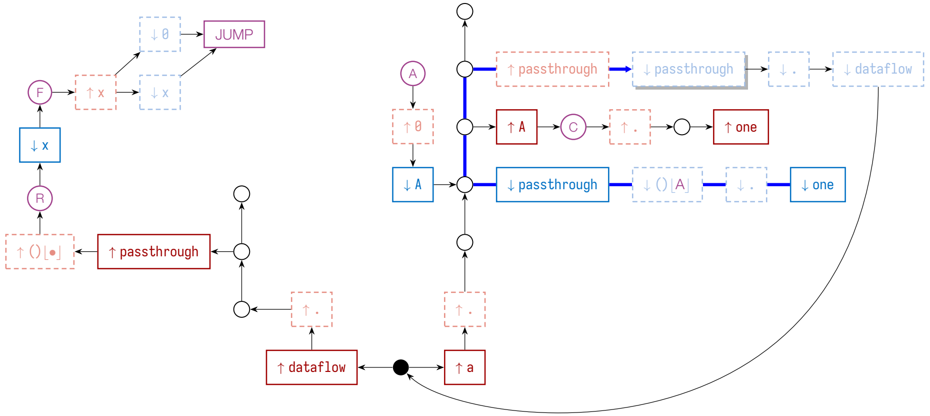
# The dataflow example



Symbol stack: ⟨()⌊A⌋.one⟩

Scope stack: ○

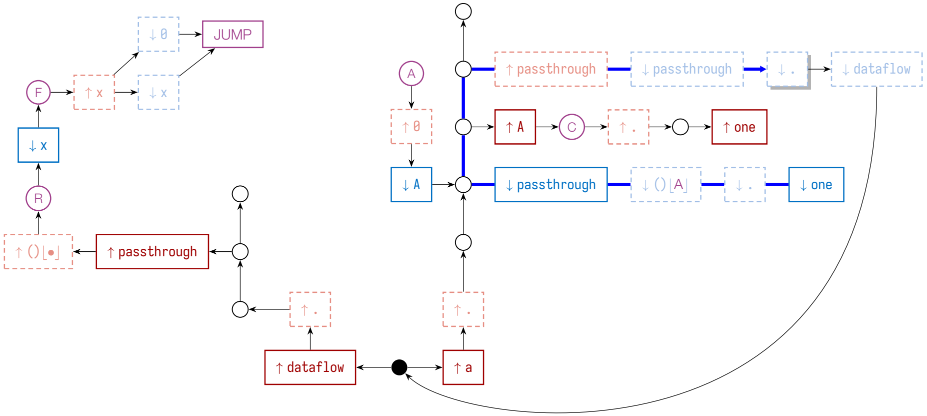
## The dataflow example



Symbol stack: ⟨passthrough()⌊A⌋.one⟩

Scope stack: ○

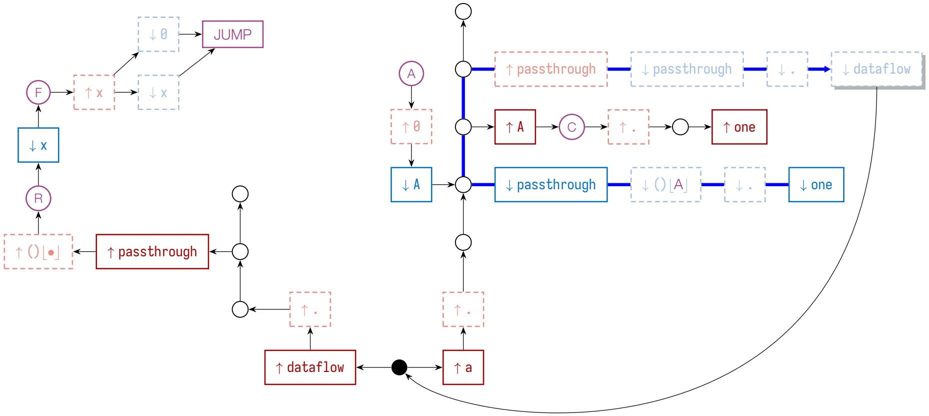
# The dataflow example



Symbol stack: ⟨.passthrough()⌊A⌋.one⟩

Scope stack: ○

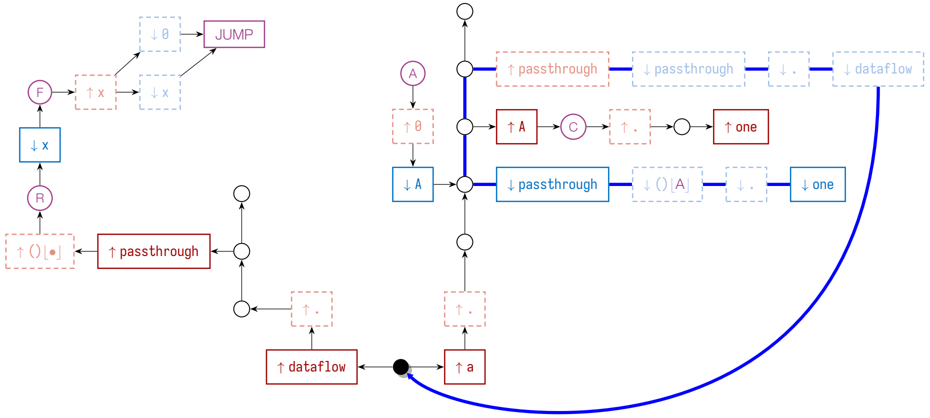
# The dataflow example



Symbol stack: ⟨dataflow.passthrough()⌊A⌋.one⟩

Scope stack: ○

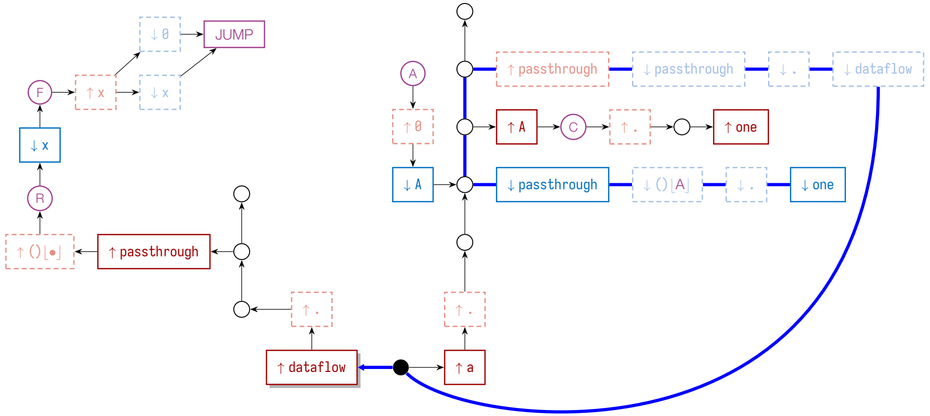
# The dataflow example



Symbol stack: ⟨`dataflow.passthrough()`⌊A⌋`.one`⟩

Scope stack: ○

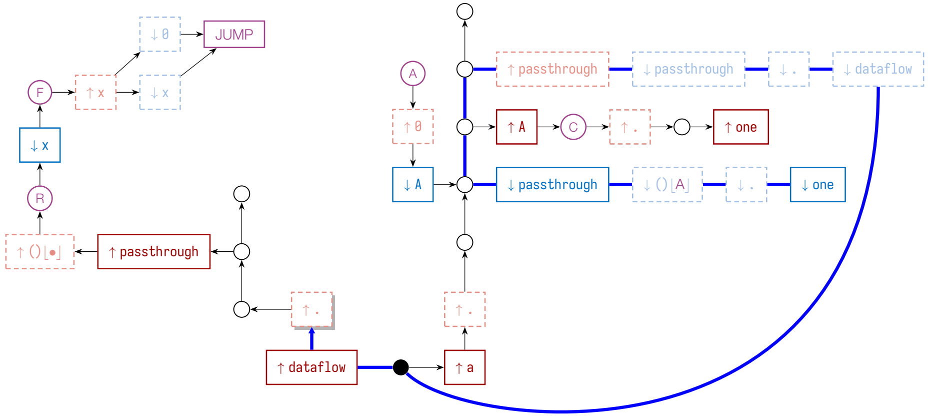
# The dataflow example



Symbol stack: ⟨.passthrough()⌊A⌋.one⟩

Scope stack: ○

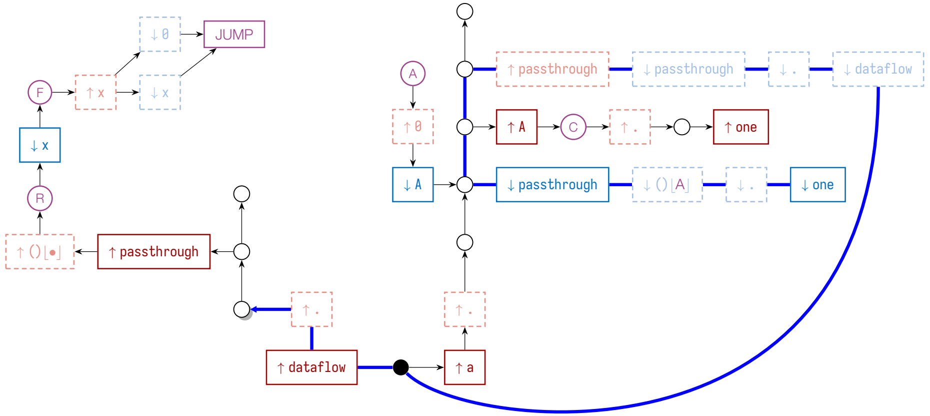
# The dataflow example



Symbol stack: ⟨passthrough()⌊A⌋.one⟩

Scope stack: ○

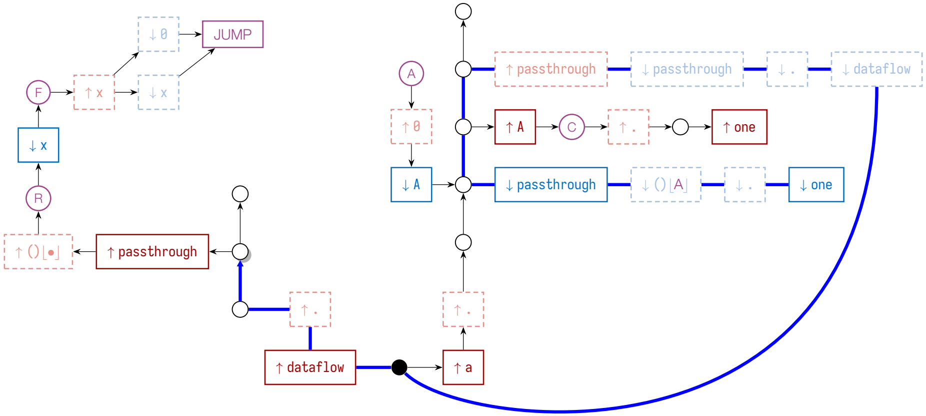
# The dataflow example



Symbol stack: ⟨passthrough()⌊A⌋.one⟩

Scope stack: ○

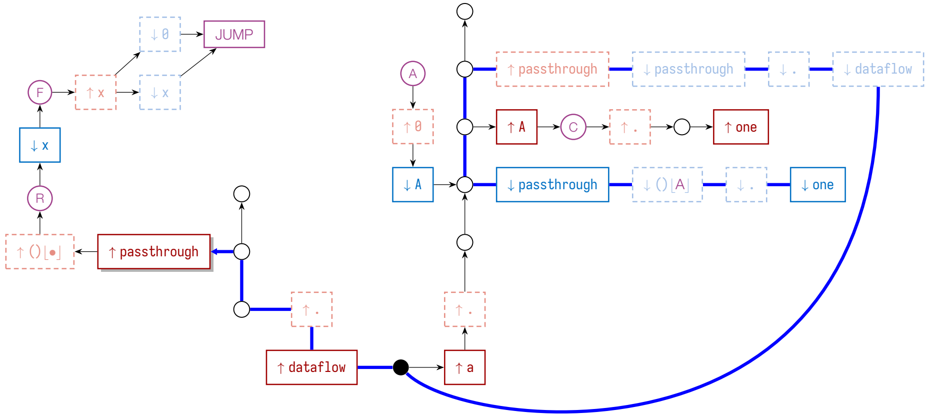
# The dataflow example



Symbol stack: ⟨passthrough()⌊A⌋.one⟩

Scope stack: ○

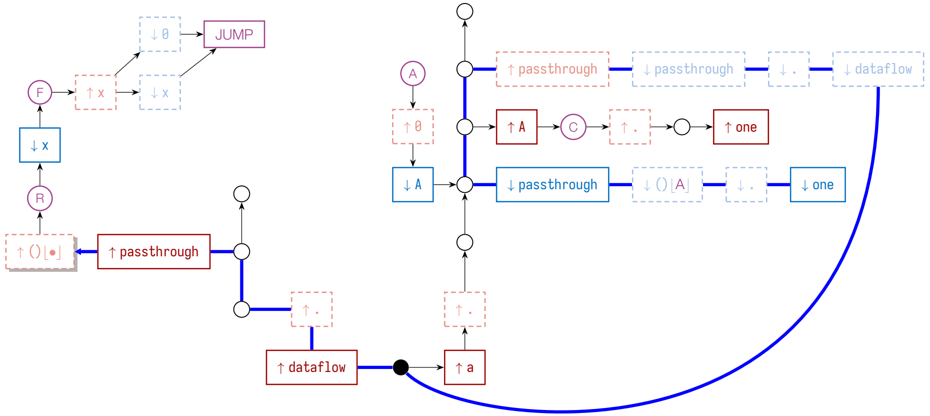
# The dataflow example



Symbol stack: ⟨()⌊A⌋.one⟩

Scope stack: ○

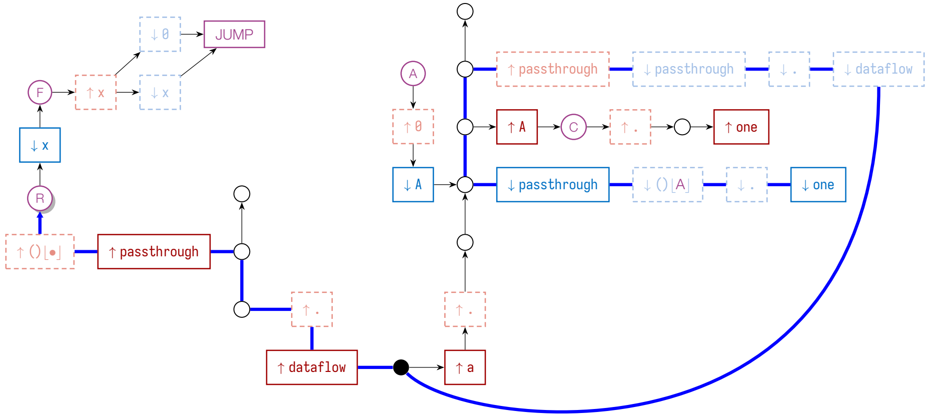
# The dataflow example



Symbol stack: ⟨.one⟩

Scope stack: (A)

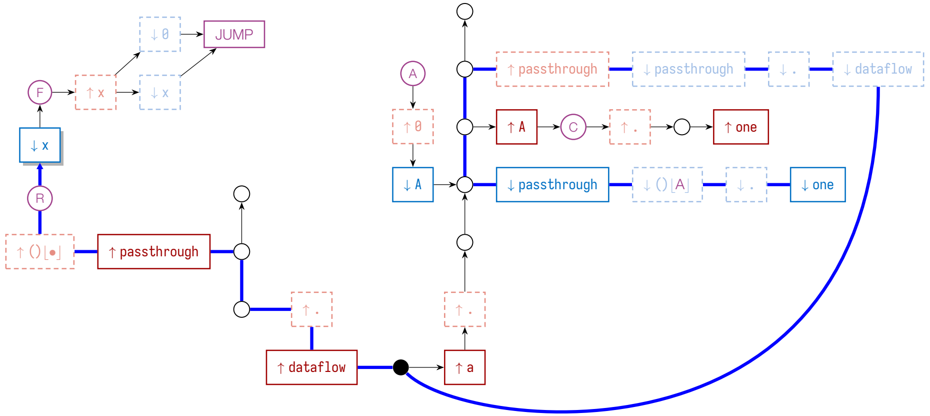
# The dataflow example



Symbol stack: ⟨.one⟩

Scope stack: (A)

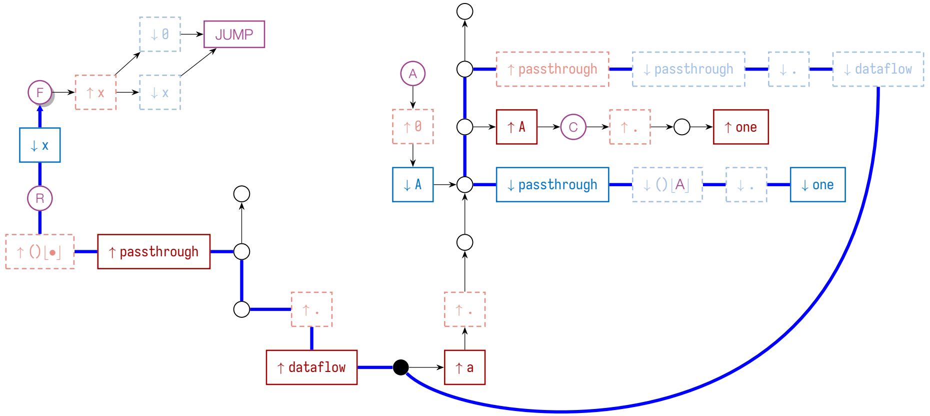
# The dataflow example



Symbol stack: ⟨x.one⟩

Scope stack: (A)

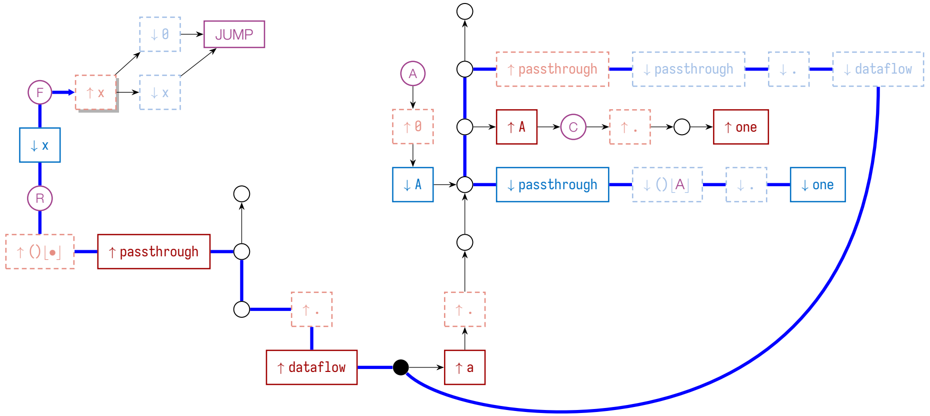
# The dataflow example



Symbol stack: $\langle$`x.one`$\rangle$

Scope stack: (A)

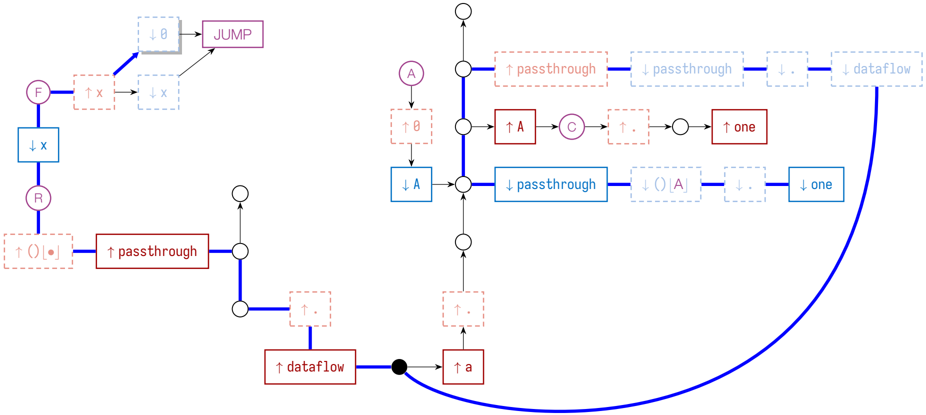
## The dataflow example



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

Scope stack: (A)

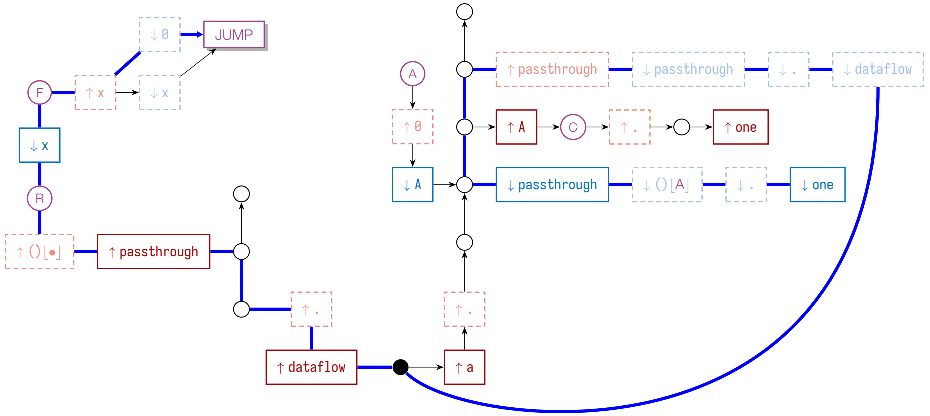
# The dataflow example



Symbol stack: ⟨0.one⟩

Scope stack: (A)

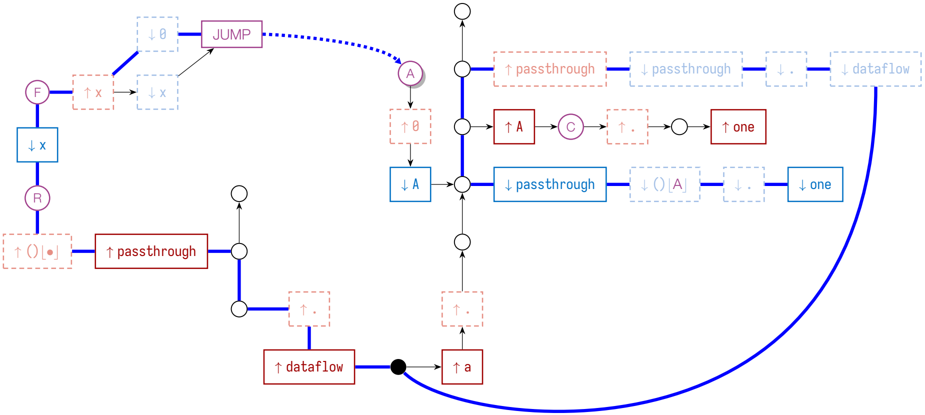
# The dataflow example



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

Scope stack: (A)

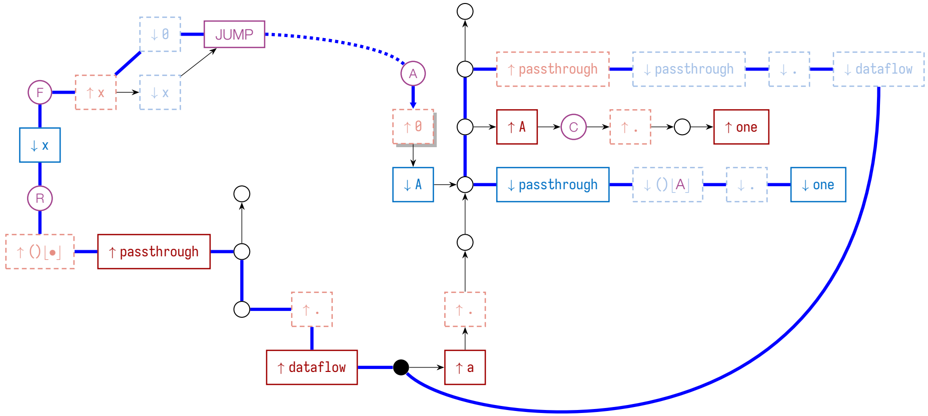
## The dataflow example



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

Scope stack: ○

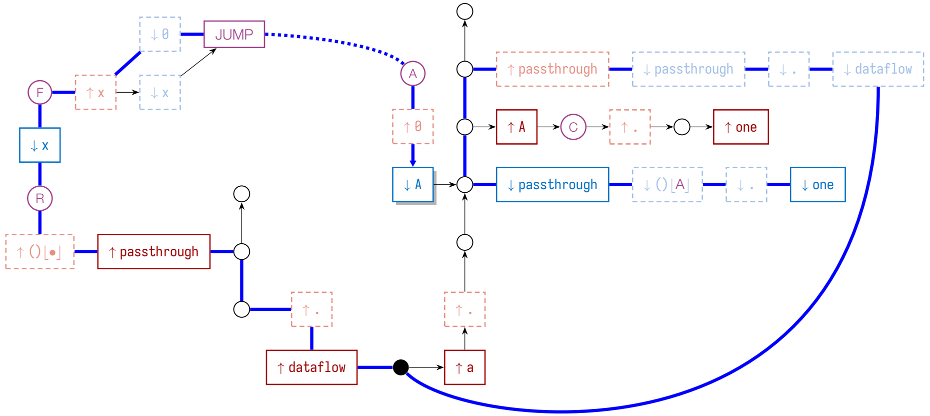
# The dataflow example



Symbol stack: ⟨.one⟩

Scope stack: ○

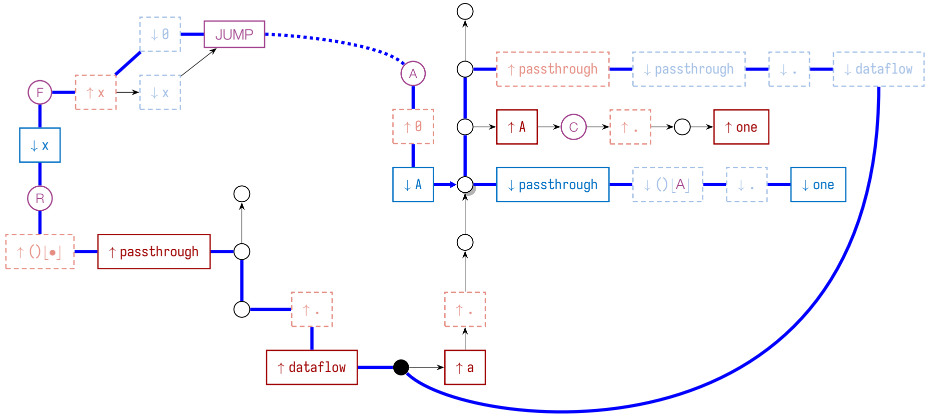
# The dataflow example



Symbol stack: ⟨`A.one`⟩

Scope stack: ○

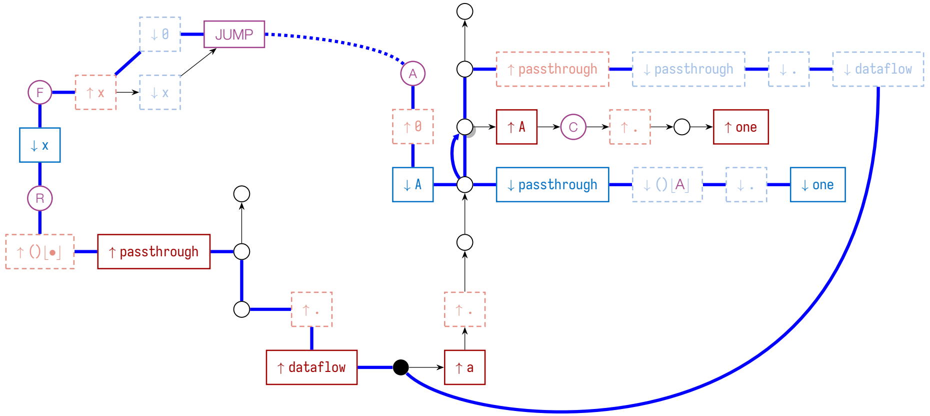
# The dataflow example



Symbol stack: ⟨A.one⟩

Scope stack: ○

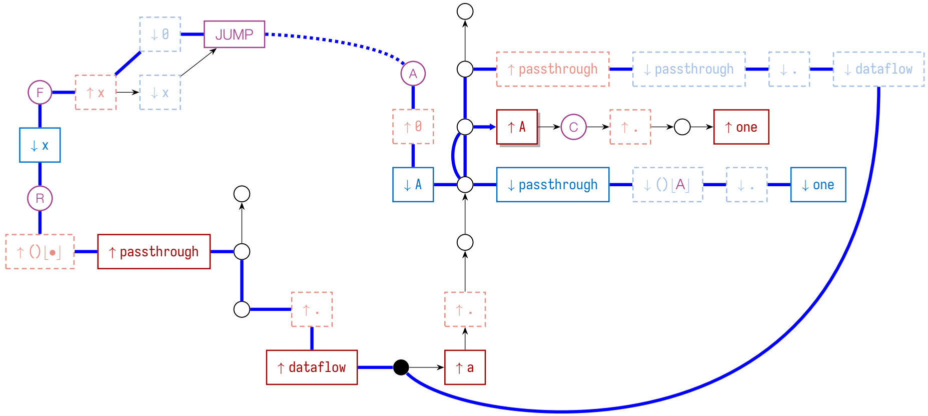
# The dataflow example



Symbol stack: ⟨`A.one`⟩

Scope stack: ○

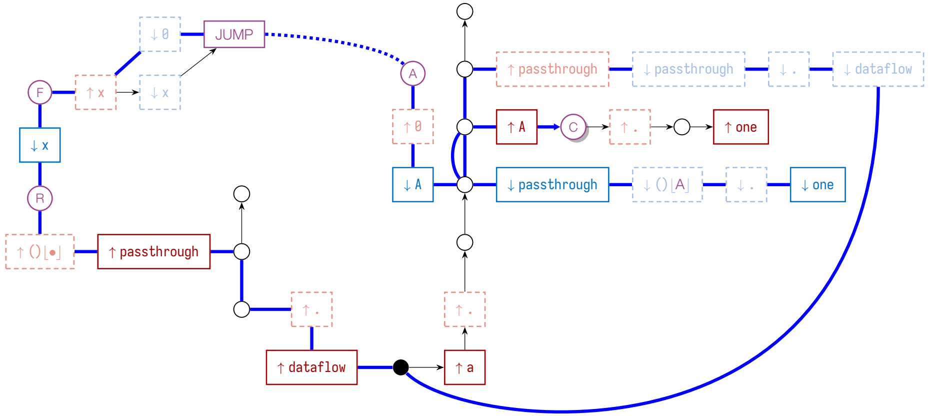
# The dataflow example



Symbol stack: ⟨.one⟩

Scope stack: ○

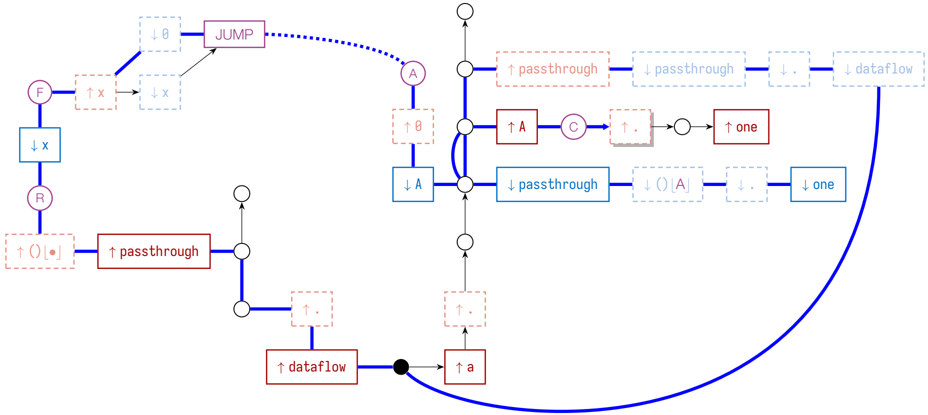
# The dataflow example



Symbol stack: ⟨`.one`⟩

Scope stack: ○

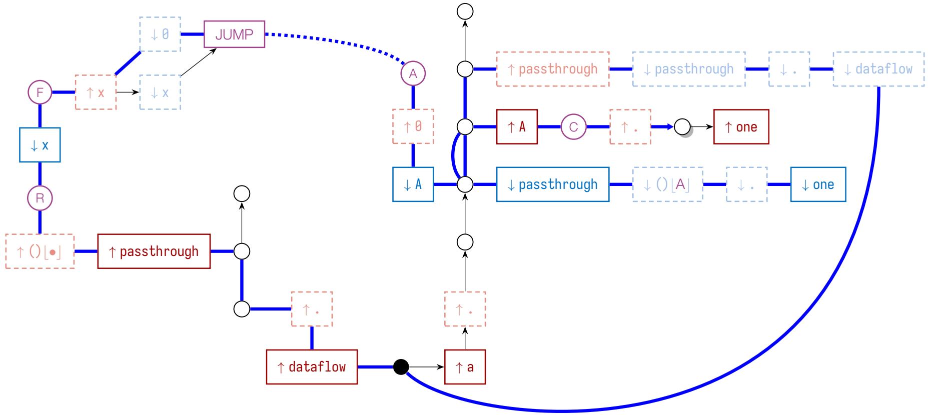
# The dataflow example



Symbol stack: ⟨one⟩

Scope stack: ○

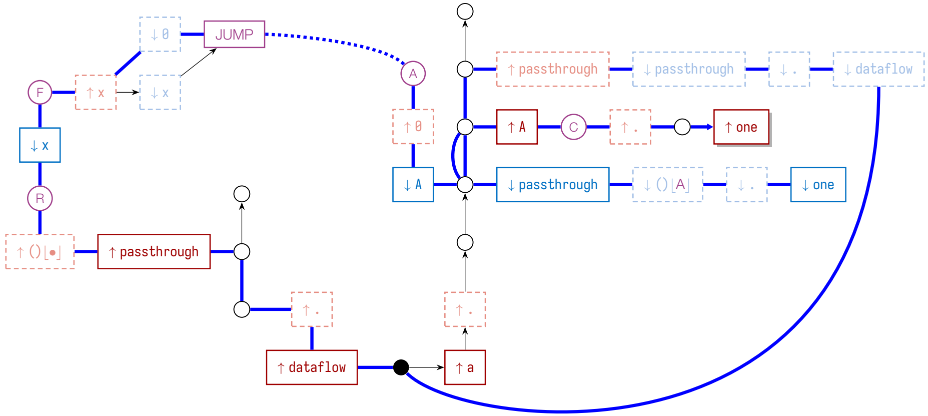
# The dataflow example



Symbol stack: ⟨one⟩

Scope stack: ○

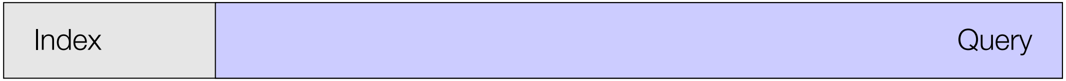
# The dataflow example



Symbol stack: ◇

Scope stack: ○

## Are we done?



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

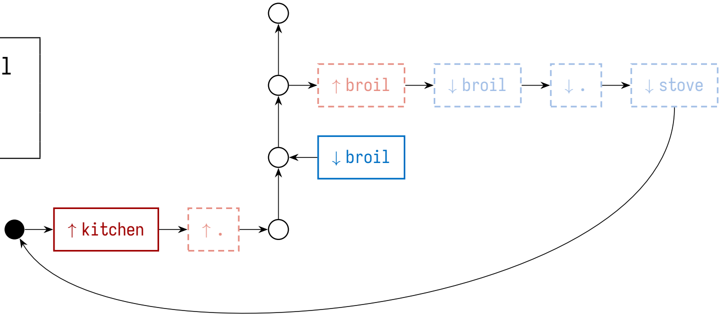
Can we shift more of the work to index time,
while still remaining incremental?

## Partial paths

## Partial paths

kitchen.py

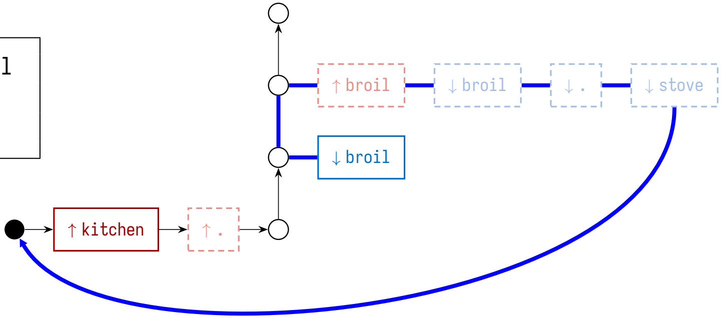
```
from stove import broil

broil()
```



## Partial paths

kitchen.py

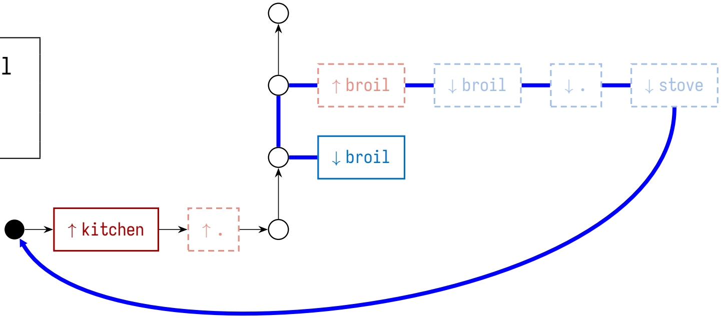
```
from stove import broil

broil()
```



## Partial paths

kitchen.py

```
from stove import broil

broil()
```

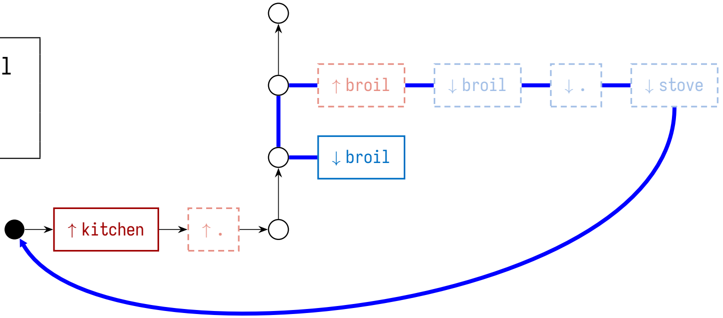


$\{\diamond, \circ\}$ $\boxed{\downarrow \texttt{broil}}$ $\rightsquigarrow$ ● $\{\langle \texttt{stove.broil} \rangle, \circ\}$

## Partial paths

kitchen.py

```python
from stove import broil

broil()
```



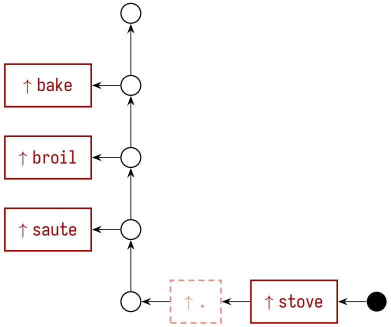
$$\{\diamond, \circ\} \; \boxed{\downarrow \texttt{broil}} \; \rightsquigarrow \; \bullet \; \{\langle \texttt{stove.broil} \rangle, \circ\}$$

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

## Partial paths

stove.py

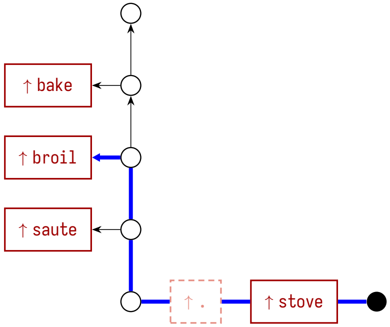
```
def bake():
    pass

def broil():
    pass

def saute():
    pass
```



## Partial paths

stove.py

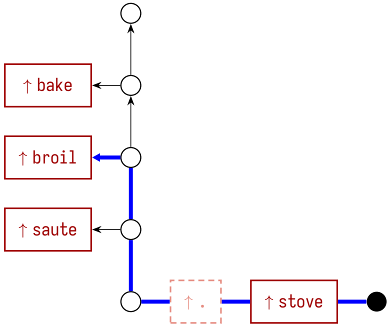
```
def bake():
    pass

def broil():
    pass

def saute():
    pass
```



## Partial paths

stove.py

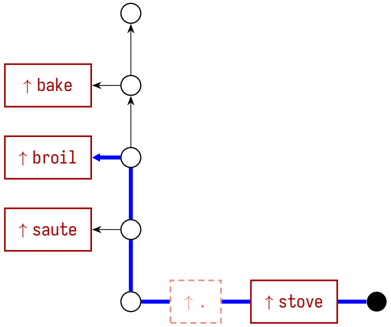
```python
def bake():
    pass

def broil():
    pass

def saute():
    pass
```



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

# Partial paths

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

def broil():
    pass

def saute():
    pass
```



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

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

## Concatenating partial paths

$\{\diamond, \circ\}$ $\boxed{\downarrow \texttt{broil}} \rightsquigarrow \bullet$ $\{\langle \texttt{stove.broil} \rangle, \circ\}$ $+$ $\{\langle \texttt{stove.broil} \rangle \cdot \psi, \phi\}$ $\bullet \rightsquigarrow \boxed{\uparrow \texttt{broil}}$ $\{\psi, \phi\}$

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

$\{\diamond, \circ\}$ $\boxed{\downarrow \texttt{broil}}$ $\rightsquigarrow \bullet$ $\{\langle \texttt{stove.broil} \rangle, \circ\}$ $+$ $\{\langle \texttt{stove.broil} \rangle \cdot \psi, \phi\}$ $\bullet \rightsquigarrow$ $\boxed{\uparrow \texttt{broil}}$ $\{\psi, \phi\}$

$$\psi = \diamond, \phi = \circ$$

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

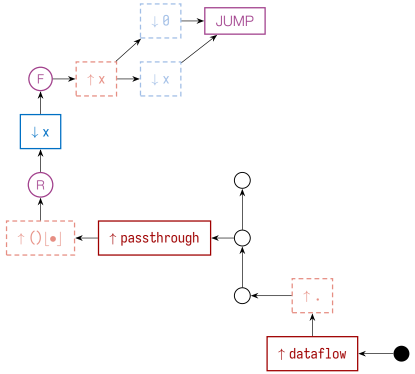
$\{\diamond, \circ\}$ $\boxed{\downarrow \texttt{broil}}$ $\rightsquigarrow$ $\boxed{\uparrow \texttt{broil}}$ $\{\diamond, \circ\}$

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

## The dataflow example

dataflow.py

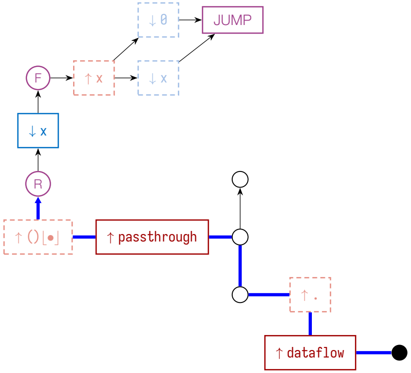
```python
def passthrough(x):
    return x
```



# The dataflow example

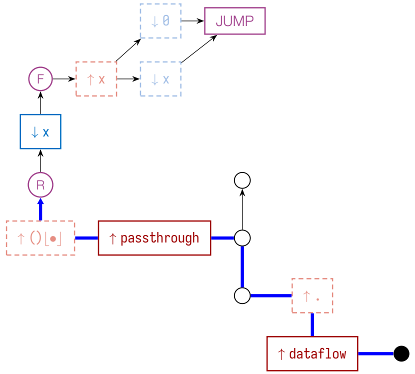
dataflow.py

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



## The dataflow example

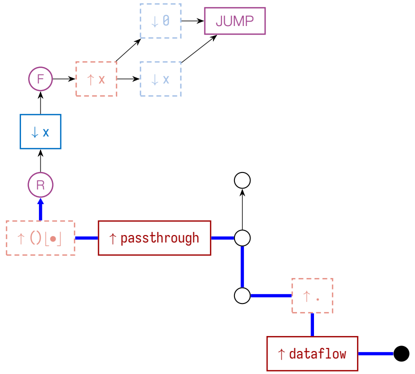
```
dataflow.py
def passthrough(x):
    return x
```



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

# The dataflow example

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



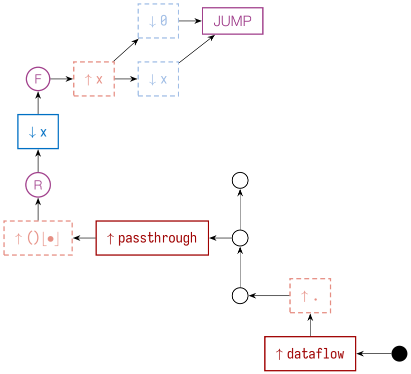
$$\{\langle \texttt{dataflow.passthrough()}\lfloor\phi_A\rfloor\rangle \cdot \psi, \phi\} \quad \bullet \rightsquigarrow \text{Ⓡ} \quad \{\psi, \phi_A\}$$

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

## The dataflow example

dataflow.py

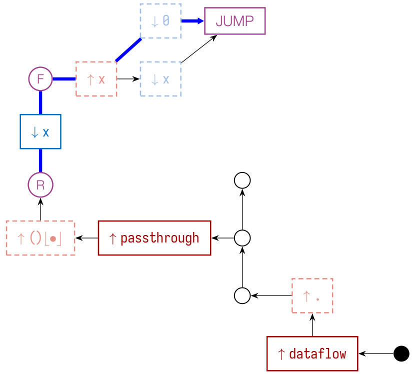
```python
def passthrough(x):
    return x
```



# The dataflow example

dataflow.py

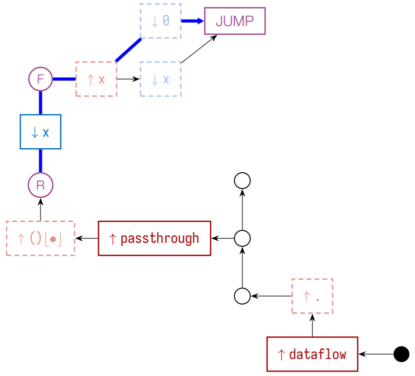
```python
def passthrough(x):
    return x
```



# The dataflow example

dataflow.py

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

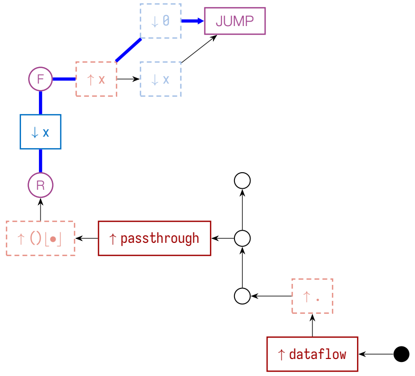


$\{\psi, \phi\}$ Ⓡ $\rightsquigarrow$ $\boxed{\text{JUMP}}$ $\{\langle 0 \rangle \cdot \psi, \phi\}$

## The dataflow example

dataflow.py

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



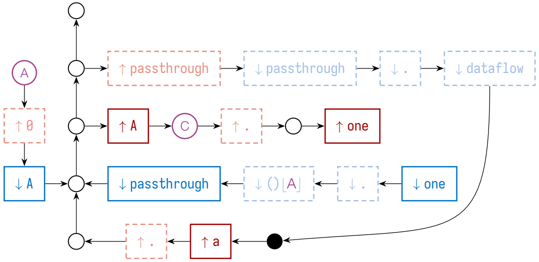
$$\{\psi, \phi\} \ \text{Ⓡ} \rightsquigarrow \boxed{\text{JUMP}} \ \{\langle \mathtt{0} \rangle \cdot \psi, \phi\}$$

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

# The dataflow example

a.py

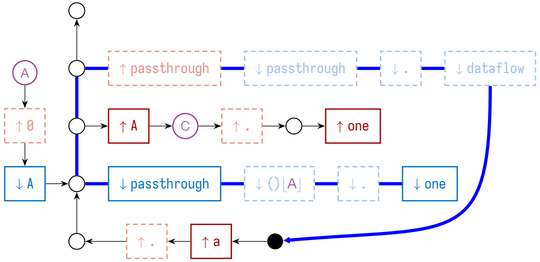
```
from dataflow import passthrough

class A:
    one = 1

passthrough(A).one
```



## The dataflow example

a.py

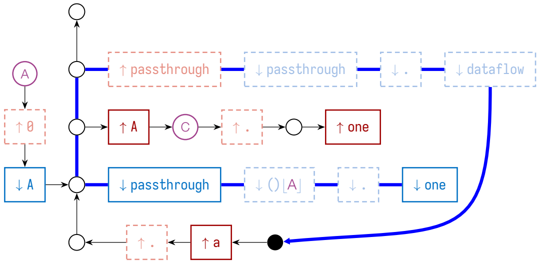
```
from dataflow import passthrough

class A:
    one = 1

passthrough(A).one
```



# The dataflow example

a.py

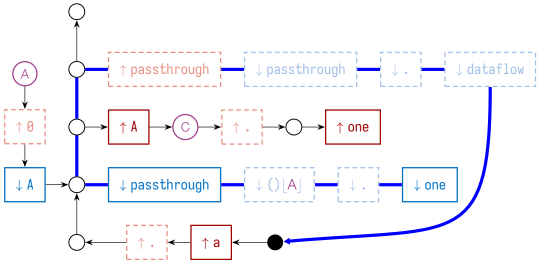
```python
from dataflow import passthrough

class A:
    one = 1

passthrough(A).one
```



$$\{\diamond, \circ\}\ \boxed{\texttt{one}}\ \rightsquigarrow\ \bullet\ \{\langle \texttt{dataflow.passthrough()}\lfloor A \rfloor\texttt{.one}\rangle, \circ\}$$

## The dataflow example

a.py
```
from dataflow import passthrough

class A:
    one = 1

passthrough(A).one
```



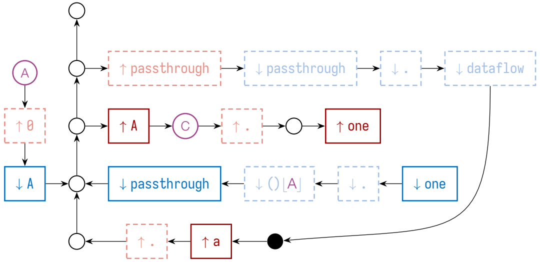
$\{\diamond, \circ\}$ $\boxed{\texttt{one}}$ $\rightsquigarrow$ $\bullet$ $\{\langle \texttt{dataflow.passthrough()}\lfloor A \rfloor\texttt{.one}\rangle, \circ\}$

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

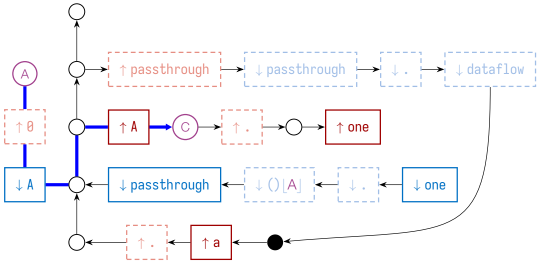
```
from dataflow import passthrough

class A:
    one = 1

passthrough(A).one
```



## The dataflow example

a.py

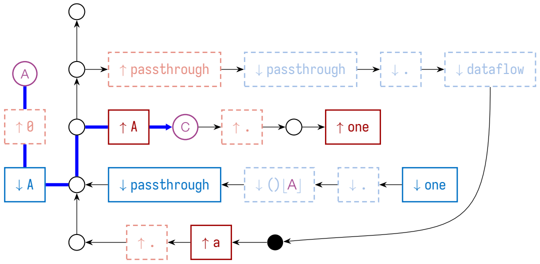
```python
from dataflow import passthrough

class A:
    one = 1

passthrough(A).one
```



# The dataflow example

a.py

```python
from dataflow import passthrough

class A:
    one = 1

passthrough(A).one
```

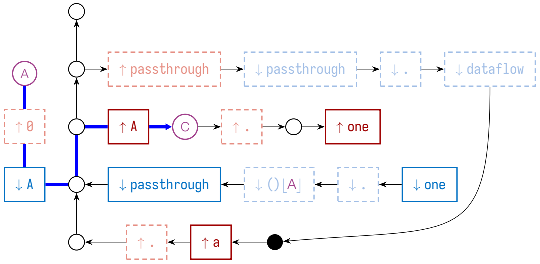


$$\{\langle 0 \rangle \cdot \psi, \phi\} \quad \textcircled{A} \rightsquigarrow \textcircled{C} \quad \{\psi, \phi\}$$

# The dataflow example

a.py

```
from dataflow import passthrough

class A:
    one = 1

passthrough(A).one
```



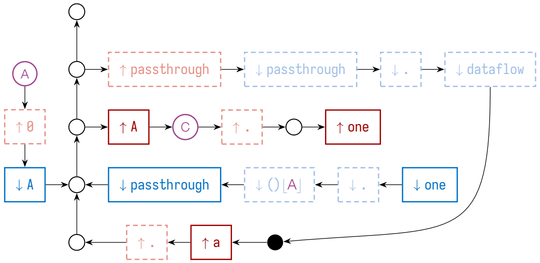
$$\{\langle 0\rangle \cdot \psi, \phi\} \; \text{Ⓐ} \rightsquigarrow \text{Ⓒ} \; \{\psi, \phi\}$$

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

# The dataflow example

a.py

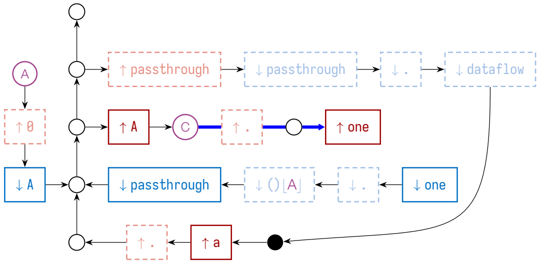
```python
from dataflow import passthrough

class A:
    one = 1

passthrough(A).one
```



# The dataflow example

a.py

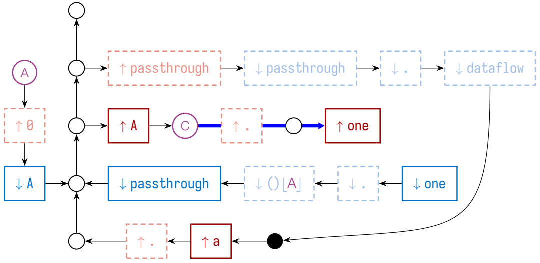
```
from dataflow import passthrough

class A:
    one = 1

passthrough(A).one
```



# The dataflow example

a.py

```
from dataflow import passthrough

class A:
    one = 1

passthrough(A).one
```

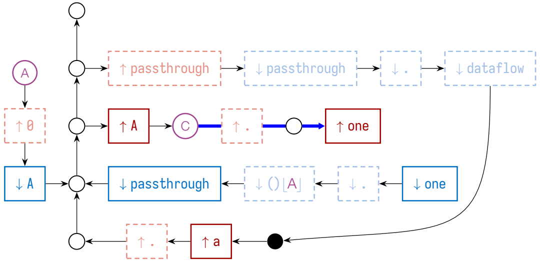


$$\{\langle .\mathsf{one}\rangle \cdot \psi, \phi\} \quad ⓒ \rightsquigarrow \boxed{\mathsf{one}} \quad \{\psi, \phi\}$$

# The dataflow example

a.py

```python
from dataflow import passthrough

class A:
    one = 1

passthrough(A).one
```



$$\{\langle .\mathtt{one}\rangle \cdot \psi, \phi\} \quad \textcircled{C} \rightsquigarrow \boxed{\mathtt{one}} \quad \{\psi, \phi\}$$

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

# The dataflow example

$$\{\diamond, \circ\} \; \boxed{\texttt{one}} \rightsquigarrow \bullet \; \{\langle \texttt{dataflow.passthrough()}\lfloor A \rfloor\texttt{.one}\rangle, \circ\}$$

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

$$\{\diamond, \circ\}\ \boxed{\texttt{one}} \rightsquigarrow \bullet\ \{\langle \texttt{dataflow.passthrough()}\lfloor A \rfloor\texttt{.one}\rangle, \circ\} \quad + \quad \{\langle \texttt{dataflow.passthrough()}\lfloor \phi_A \rfloor\rangle \cdot \psi, \phi\}\ \bullet \rightsquigarrow \textcircled{R}\ \{\psi, \phi_A\}$$

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

$$\{\diamond, \circ\}\ \boxed{\texttt{one}} \rightsquigarrow \bullet\ \{\langle \texttt{dataflow.passthrough()}\lfloor \mathsf{A} \rfloor\texttt{.one}\rangle, \circ\} \quad + \quad \{\langle \texttt{dataflow.passthrough()}\lfloor \phi_A \rfloor\rangle \cdot \psi, \phi\}\ \bullet \rightsquigarrow \textcircled{R}\ \{\psi, \phi_A\}$$

$$\psi = \langle\texttt{.one}\rangle, \phi = \circ, \phi_A = (\mathsf{A})$$

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

$$\{\diamond, \circ\}\ \boxed{\texttt{one}} \rightsquigarrow \textcircled{R}\ \{\langle\texttt{.one}\rangle, (\mathrm{A})\}$$

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

$$\{\diamond, \circ\}\ \boxed{\texttt{one}} \rightsquigarrow \textcircled{\text{R}}\ \{\langle\texttt{.one}\rangle, (\text{A})\} \quad + \quad \{\psi, \phi\}\ \textcircled{\text{R}} \rightsquigarrow \boxed{\text{JUMP}}\ \{\langle\texttt{0}\rangle \cdot \psi, \phi\}$$

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

$$\{\diamond, \circ\}\ \boxed{\texttt{one}} \rightsquigarrow \text{Ⓡ}\ \{\langle\texttt{.one}\rangle, (\text{A})\} \qquad + \qquad \{\psi, \phi\}\ \text{Ⓡ} \rightsquigarrow \boxed{\text{JUMP}}\ \{\langle\texttt{0}\rangle \cdot \psi, \phi\}$$

$$\psi = \langle\texttt{.one}\rangle, \phi = \circ$$

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

$$\{\diamond, \circ\}\ \boxed{\text{one}} \rightsquigarrow \boxed{\text{JUMP}}\ \{\langle \texttt{0.one} \rangle, (\text{A})\}$$

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

# The dataflow example

$$\{\diamond, \circ\}\ \boxed{\text{one}} \rightsquigarrow \boxed{\text{JUMP}}\ \{\langle \texttt{0.one} \rangle, (\text{A})\}$$

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

$$\{\diamond, \circ\} \; \boxed{\texttt{one}} \rightsquigarrow \text{Ⓐ} \; \{\langle \texttt{0.one} \rangle, \circ\}$$

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

## The dataflow example

$$\{\diamond, \circ\} \; \boxed{\texttt{one}} \rightsquigarrow \textcircled{A} \; \{\langle \texttt{0.one} \rangle, \circ\} \qquad + \qquad \{\langle \texttt{0} \rangle \cdot \psi, \phi\} \; \textcircled{A} \rightsquigarrow \textcircled{C} \; \{\psi, \phi\}$$

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

$$\{\diamond, \circ\} \; \boxed{\texttt{one}} \rightsquigarrow \textcircled{A} \; \{\langle \texttt{0.one} \rangle, \circ\} \qquad + \qquad \{\langle \texttt{0} \rangle \cdot \psi, \phi\} \; \textcircled{A} \rightsquigarrow \textcircled{C} \; \{\psi, \phi\}$$

$$\psi = \langle \texttt{.one} \rangle, \phi = \circ$$

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

$$\{\diamond, \circ\} \ \boxed{\texttt{one}} \rightsquigarrow \textcircled{c} \ \{\langle\texttt{.one}\rangle, \circ\}$$

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

# The dataflow example

$$\{\diamond, \circ\}\ \boxed{\texttt{one}} \rightsquigarrow \text{\textcircled{c}}\ \{\langle\texttt{.one}\rangle, \circ\} \qquad + \qquad \{\langle\texttt{.one}\rangle \cdot \psi, \phi\}\ \text{\textcircled{c}} \rightsquigarrow \boxed{\texttt{one}}\ \{\psi, \phi\}$$

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

$$\{\diamond, \circ\}\ \boxed{\texttt{one}} \rightsquigarrow \textcircled{C}\ \{\langle\texttt{.one}\rangle, \circ\} \qquad + \qquad \{\langle\texttt{.one}\rangle \cdot \psi, \phi\}\ \textcircled{C} \rightsquigarrow \boxed{\texttt{one}}\ \{\psi, \phi\}$$

$$\psi = \diamond, \phi = \circ$$

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

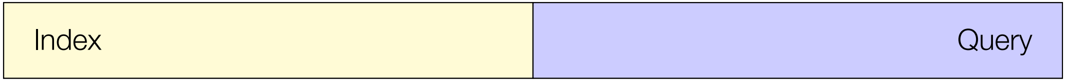
$+$

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

## The dataflow example

$$\{\diamond, \circ\}\ \boxed{\texttt{one}} \rightsquigarrow \boxed{\texttt{one}}\ \{\diamond, \circ\}$$

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



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();
    }
}
```

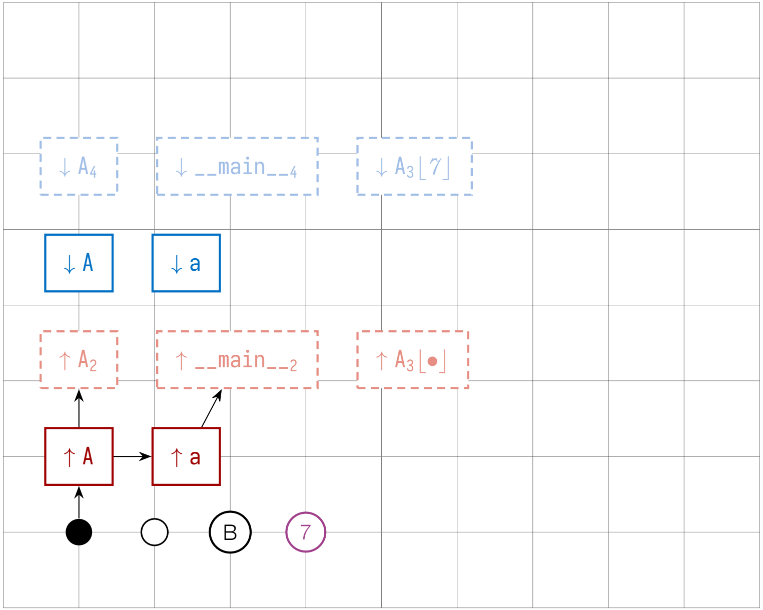
# Picture credits

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

## Picture credits

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

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


↓ A₄
↓ __main__₄
↓ A₃⌊7⌋
↓ A
↓ a
↑ A₂
↑ __main__₂
↑ A₃⌊●⌋
↑ A
↑ a
B
7

## 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

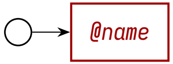
```
(module [0, 0] - [10, 0]
  (function_definition [0, 0] - [1, 8]
    name: (identifier [0, 4] - [0, 8])
    parameters: (parameters [0, 8] - [0, 10])
    body: (block [1, 4] - [1, 8]
      (pass_statement [1, 4] - [1, 8])))
  (function_definition [3, 0] - [4, 8]
    name: (identifier [3, 4] - [3, 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
```

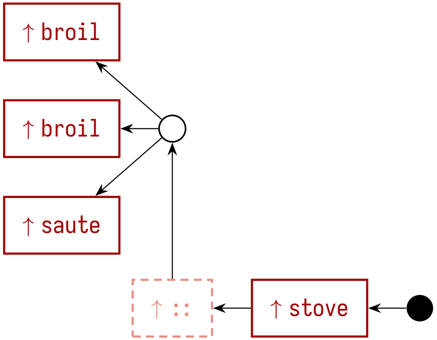
github/stack-graphs
tree-sitter/tree-sitter
tree-sitter/tree-sitter-graph

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() {}
```


↑ broil
↑ broil
↑ saute
↑ ::
↑ stove