

# **Incremental, zero-config Code Navigation using stack graphs**

Douglas Creager  
@dcreager



Strange Loop  
October 1, 2021 – St. Louis

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 {  
    def z1 = 5  
}  
module A2 {  
    import A1 as SB  
    def x2 = 1 + z1  
}
```

SB  
SA  
A<sub>1</sub>  
A<sub>2</sub>  
x<sub>1</sub>  
x<sub>2</sub>

HUAWEI

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 definitions: 'bake()', 'broil()', 'saute()', and 'broil()' again at the bottom. The word 'broil' is highlighted in blue in both the definition and the call. A red arrow points from the call 'broil()' at the bottom up to the definition 'def broil():' in the middle. Another red arrow points from the definition 'def broil():' up to the word 'broil' in the same line, creating a loop-like effect.

A white marble statue of a man, possibly David, is shown from the waist up, set against a clear blue sky. The man 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 lighting creates strong shadows on the statue's face and body.

**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 three definitions of the `broil()` function. The first two definitions are enclosed in a dotted red rectangle, indicating they are part of the same scope. A red arrow points from the second definition back to the first, forming a circular reference. The third definition of `broil()` is outside this scope, and a dotted red arrow points from its `broil()` call back to the first definition, creating a wider loop.

# 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

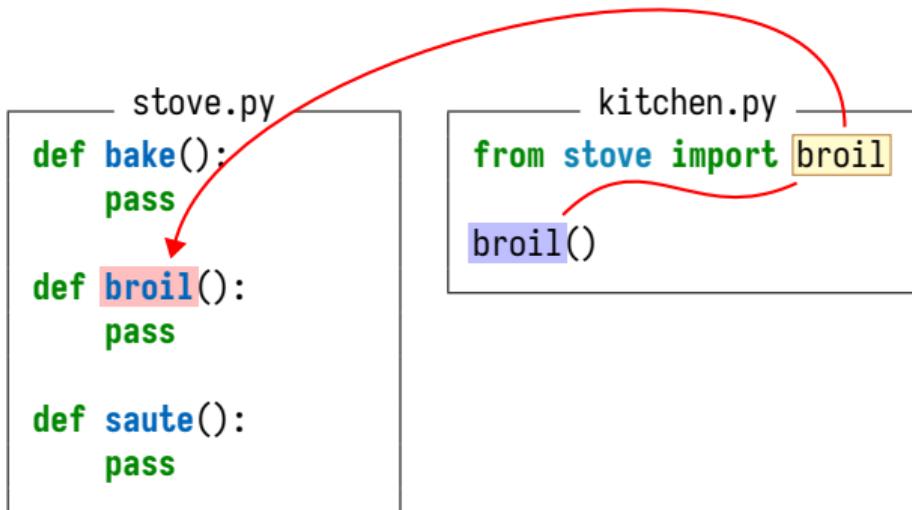
def saute():
    pass
```

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 *
```

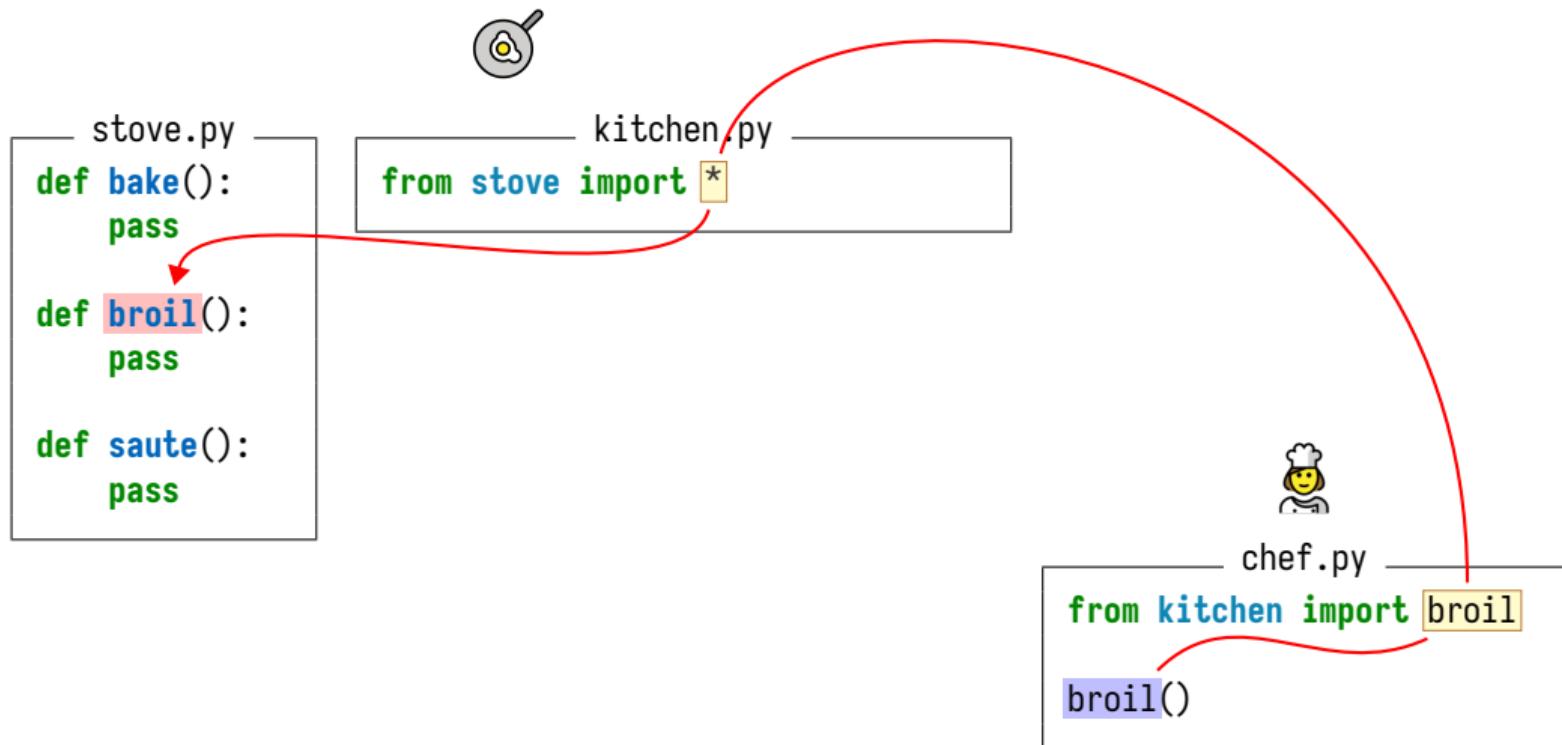


chef.py

```
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
```

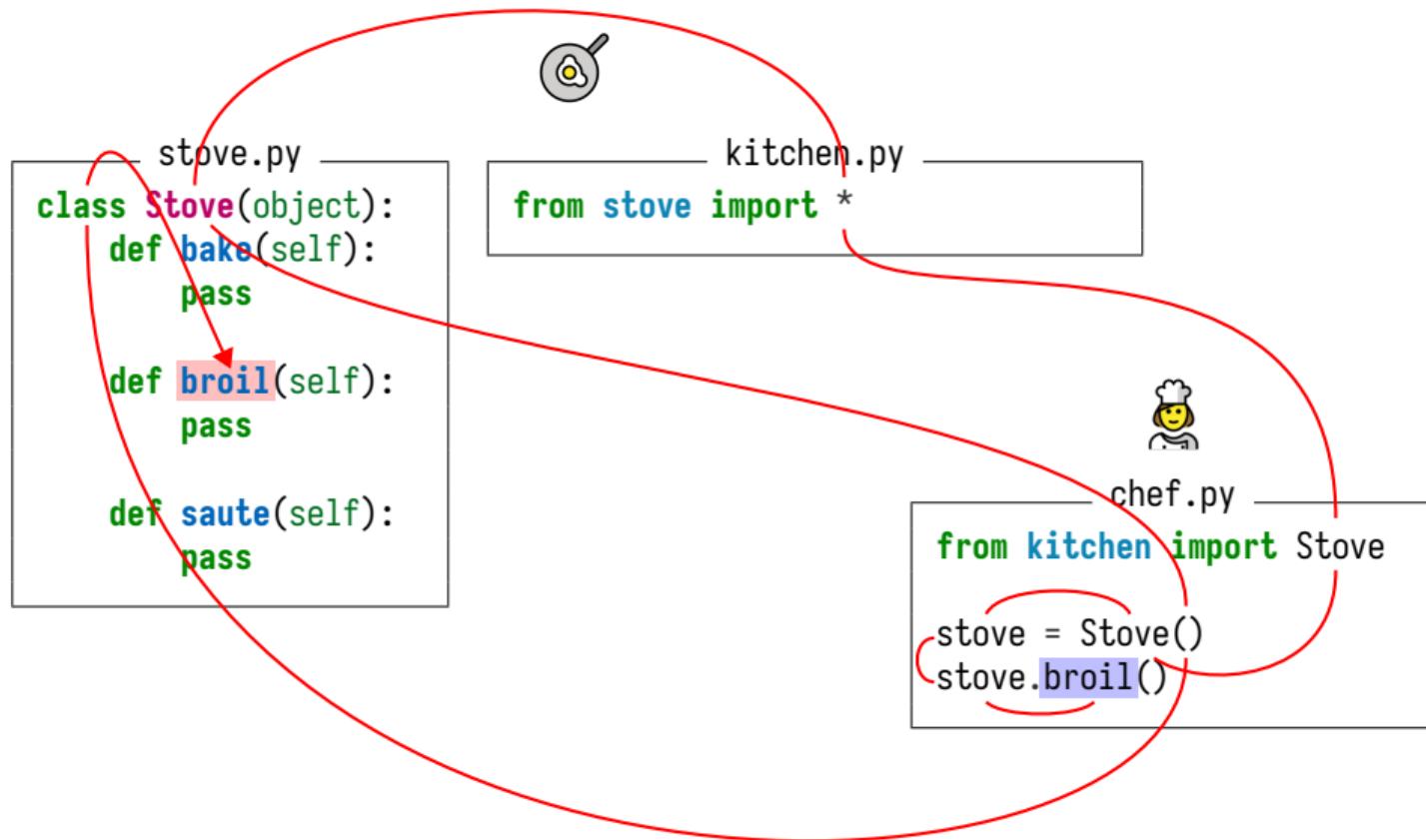
```
from stove import *
```



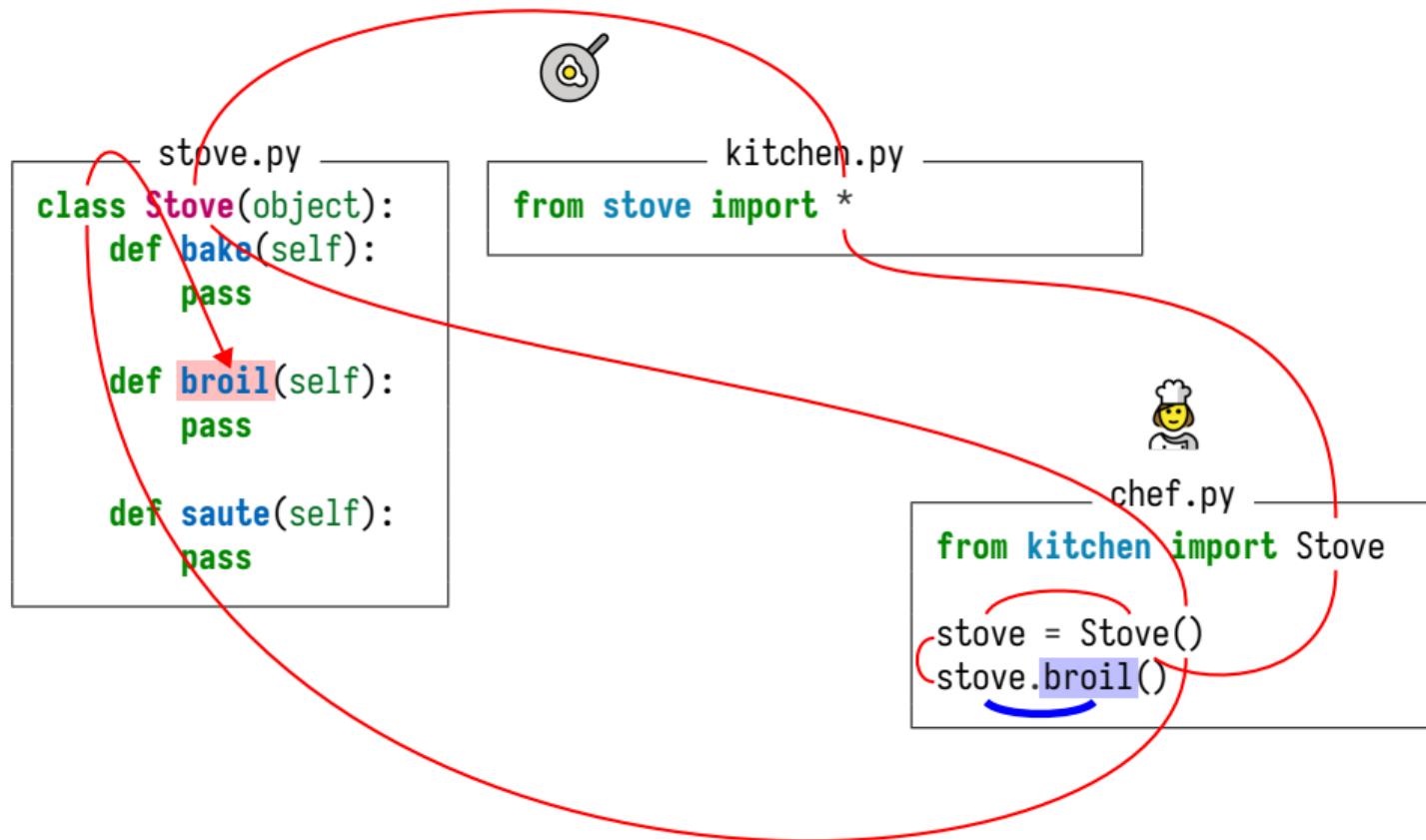
```
chef.py
```

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

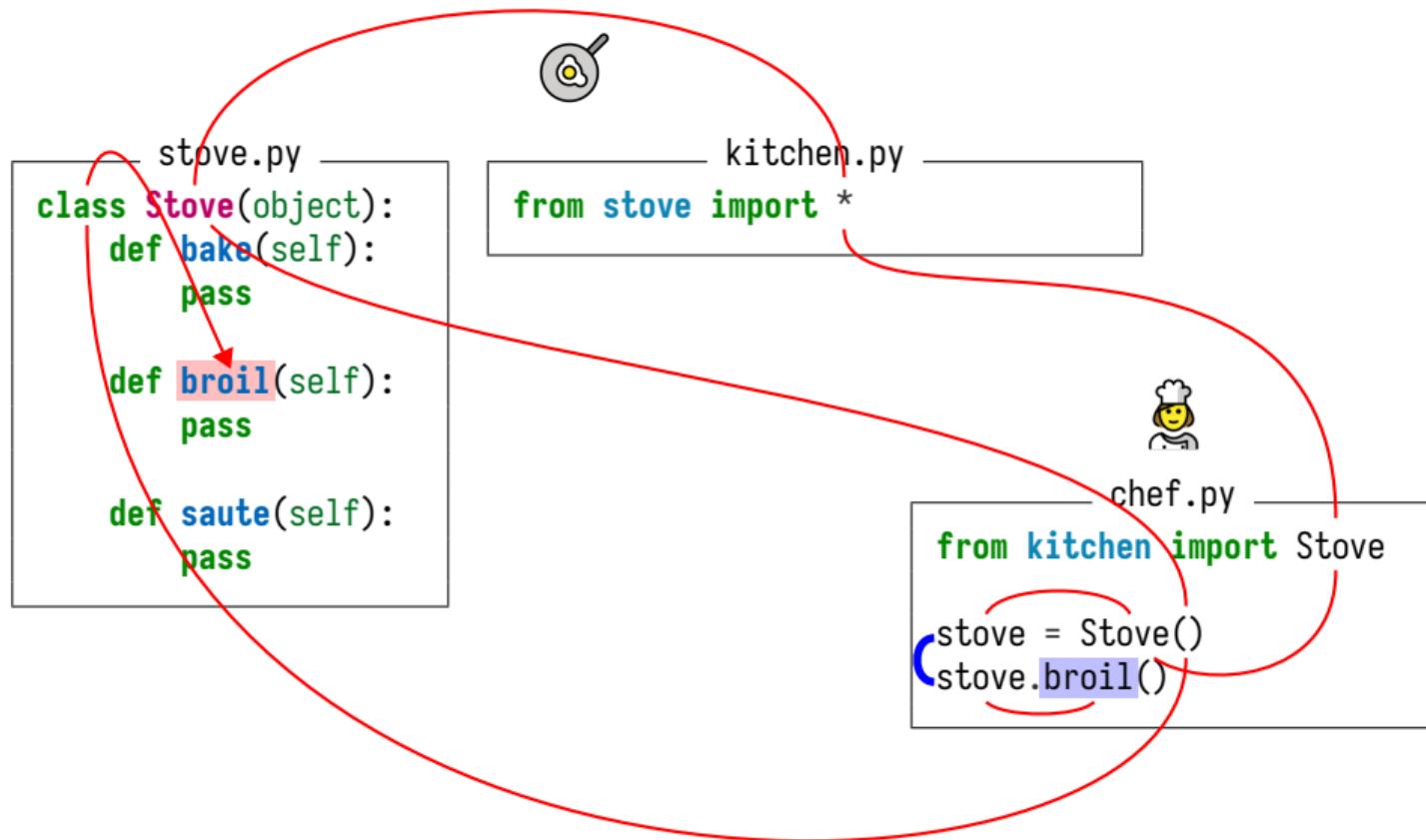
# Why is this hard?



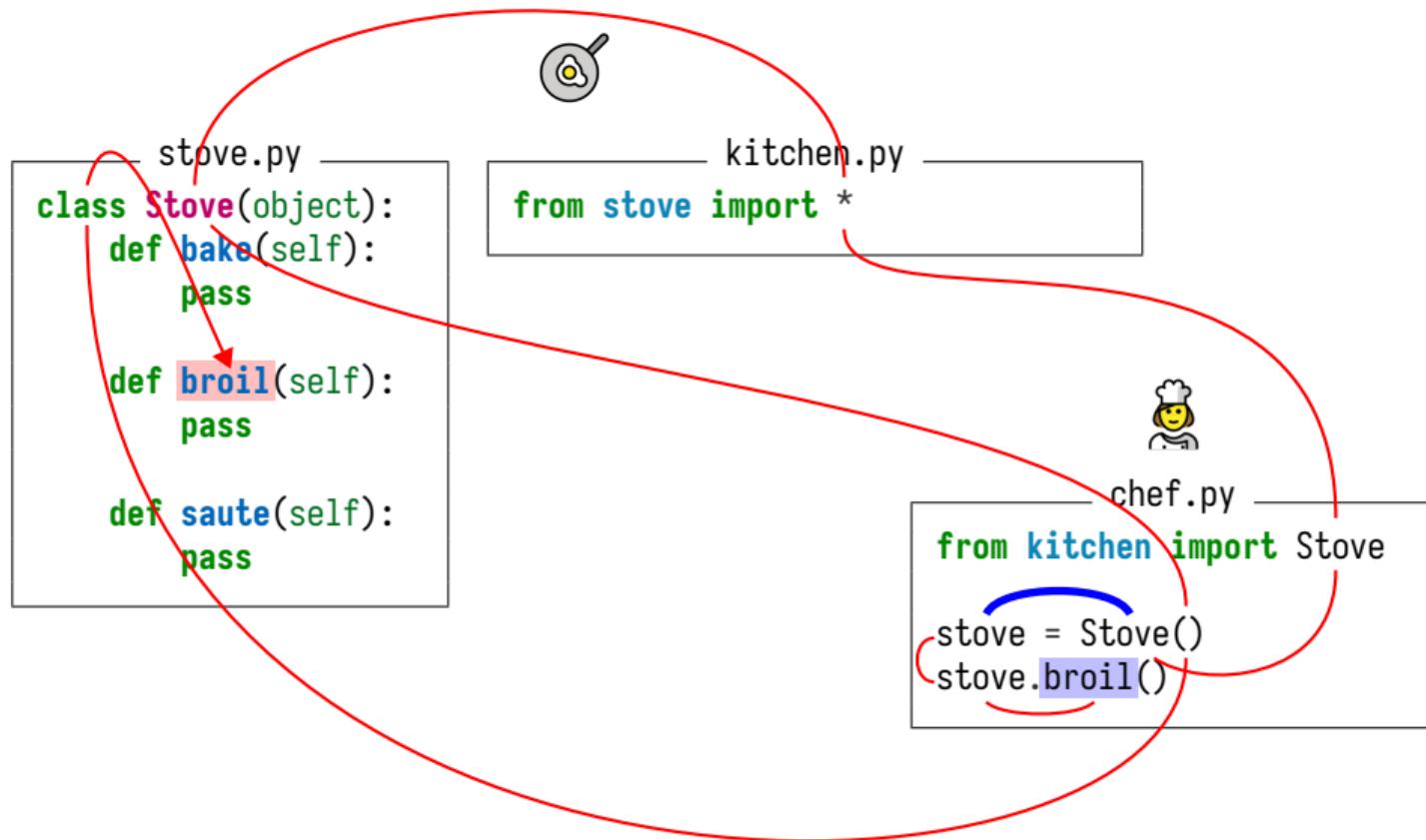
# Why is this hard?



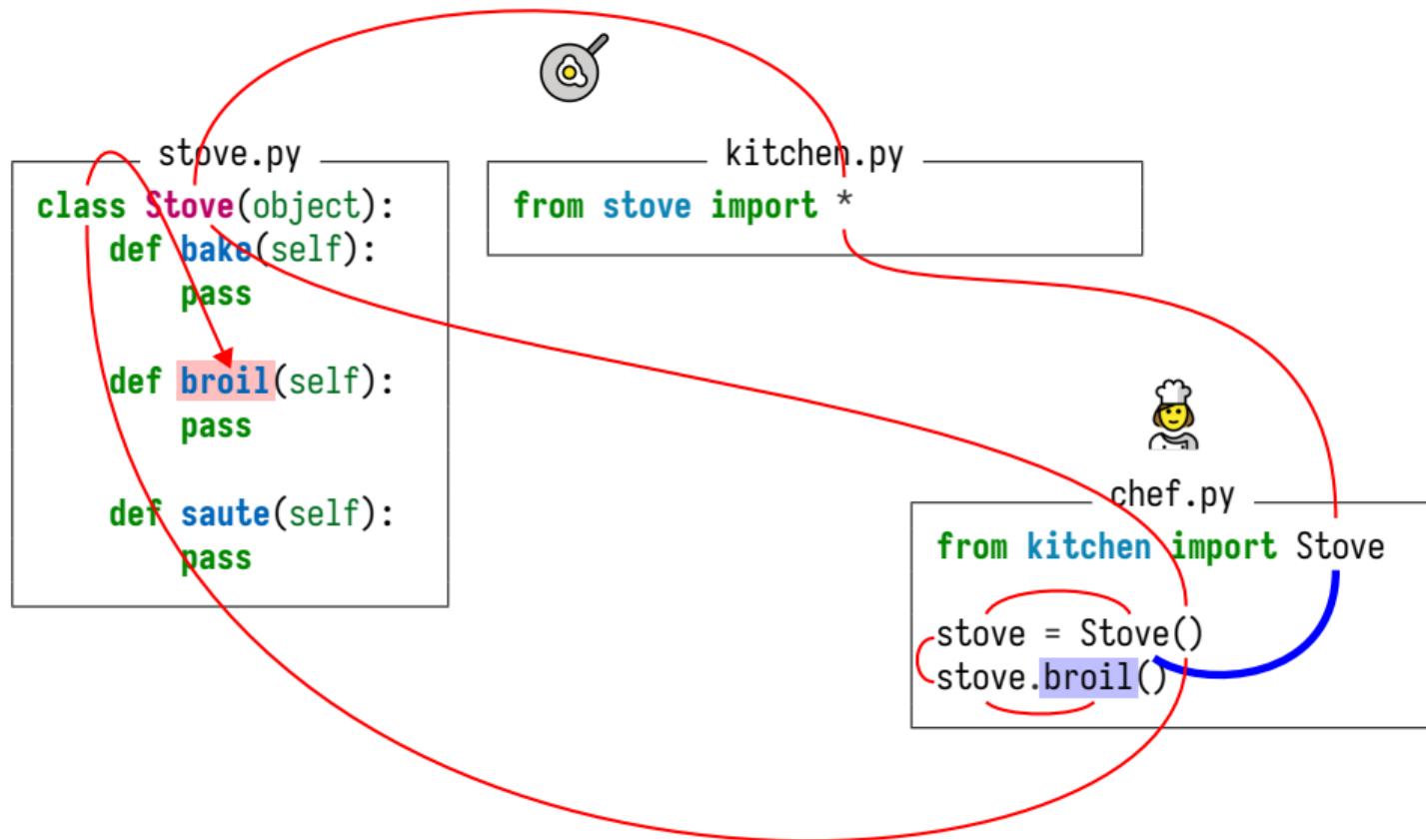
# Why is this hard?



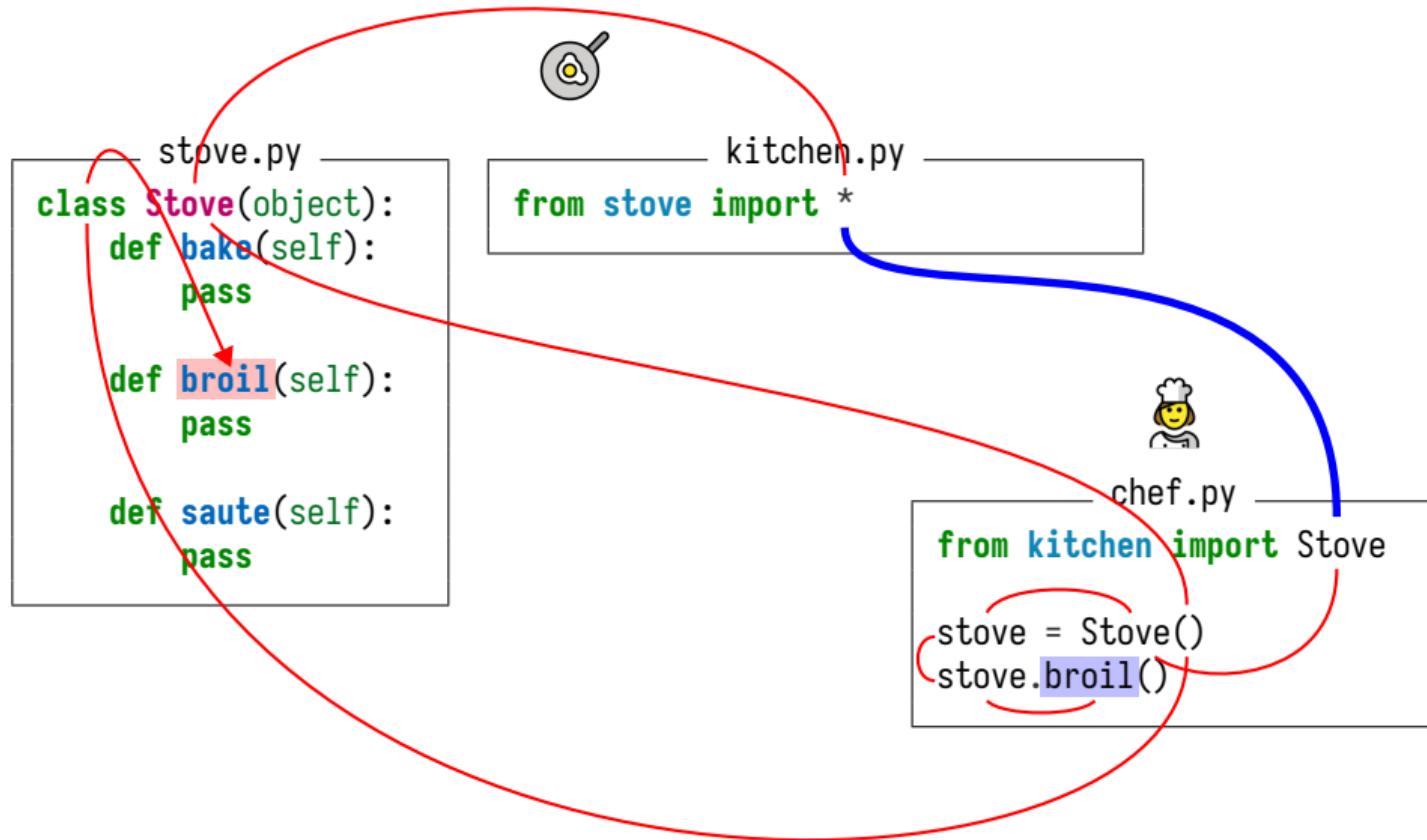
# Why is this hard?



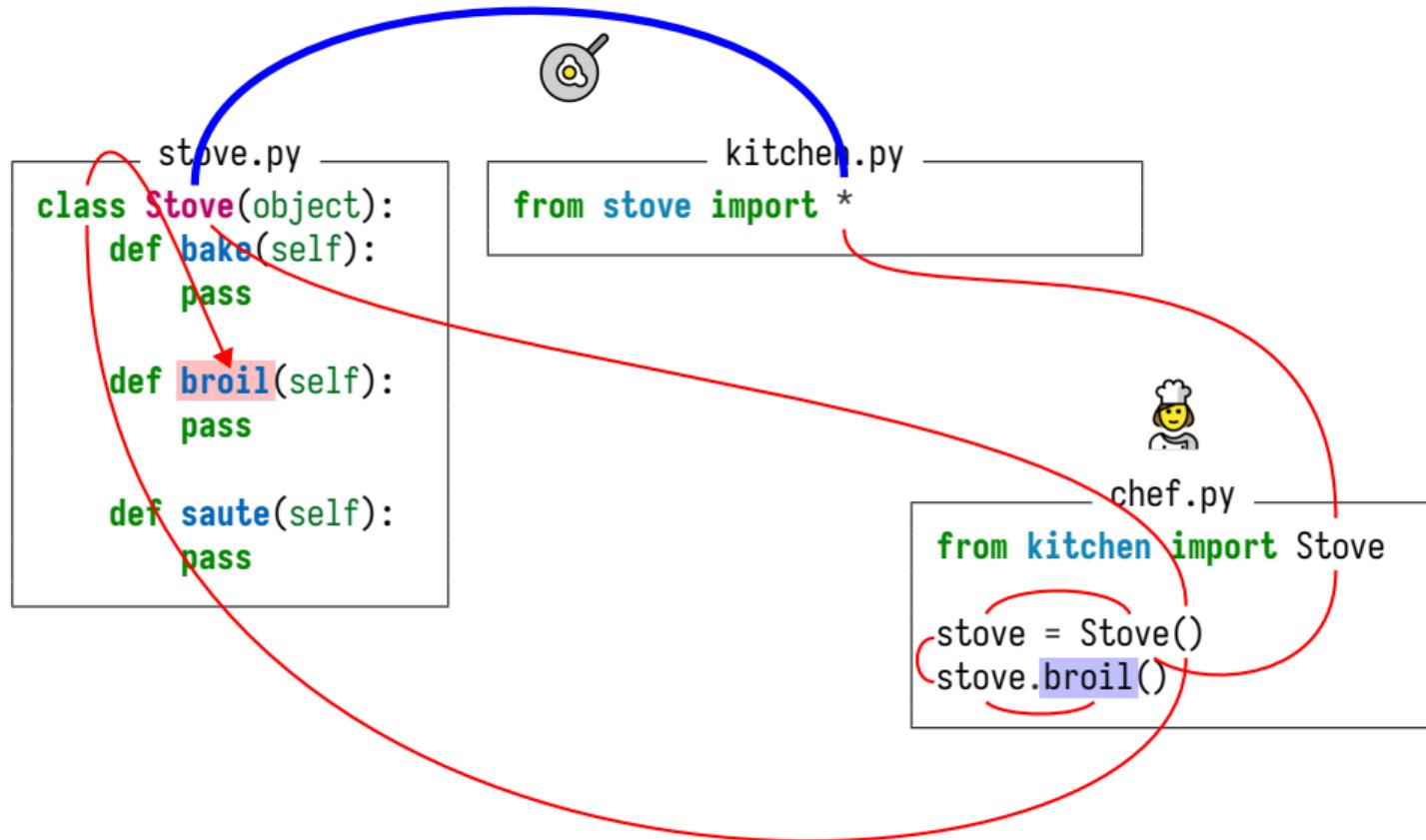
# Why is this hard?



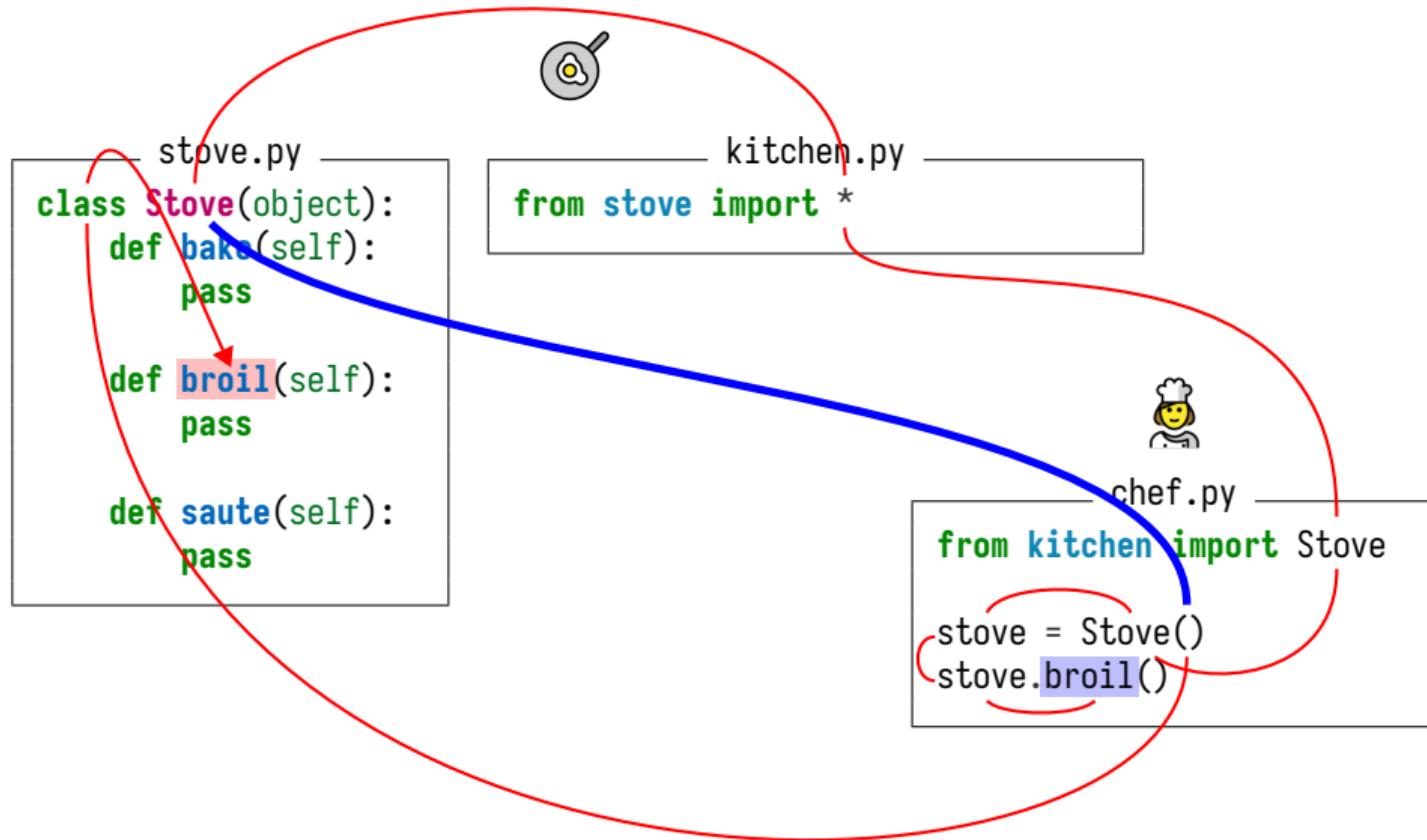
# Why is this hard?



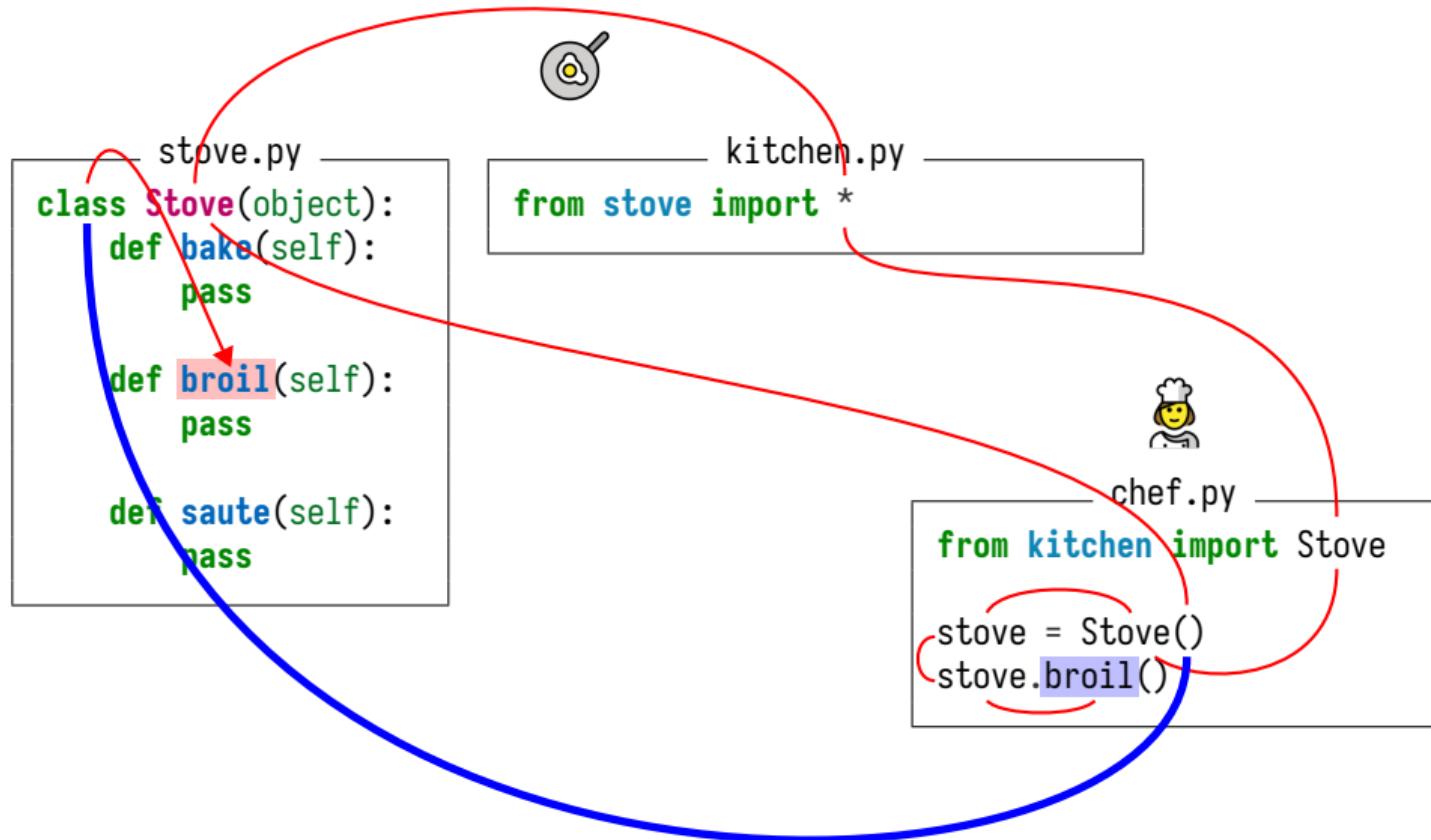
# Why is this hard?



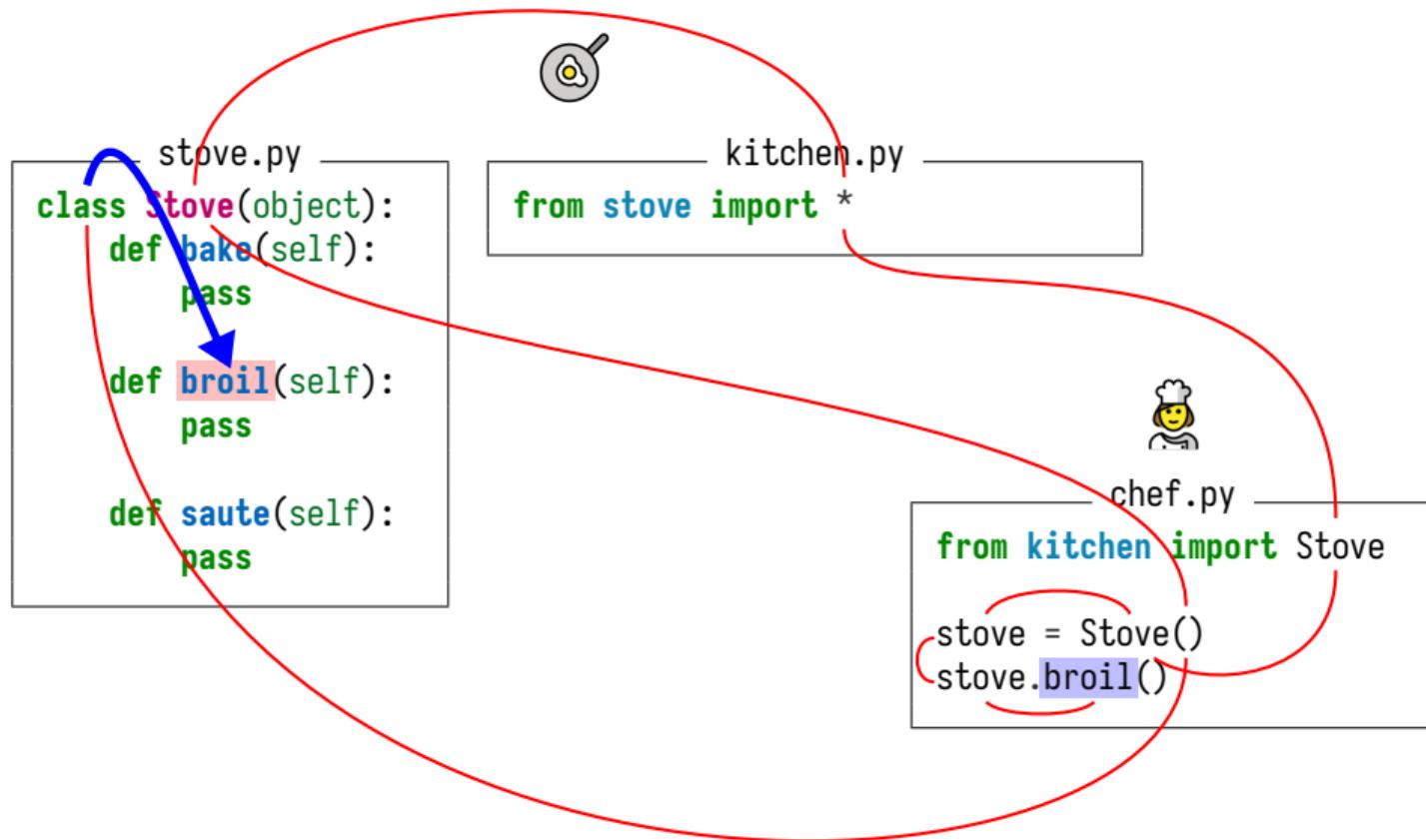
# Why is this hard?



# Why is this hard?



# 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

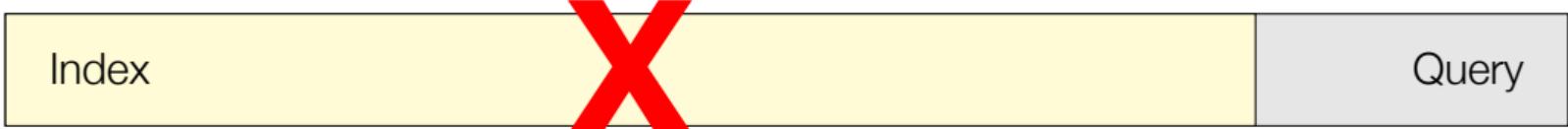
# When do we do the work?



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

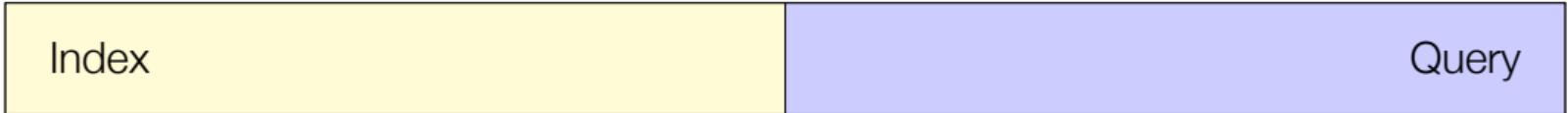
Goal: < 100ms

# 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.

# Incremental processing

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.

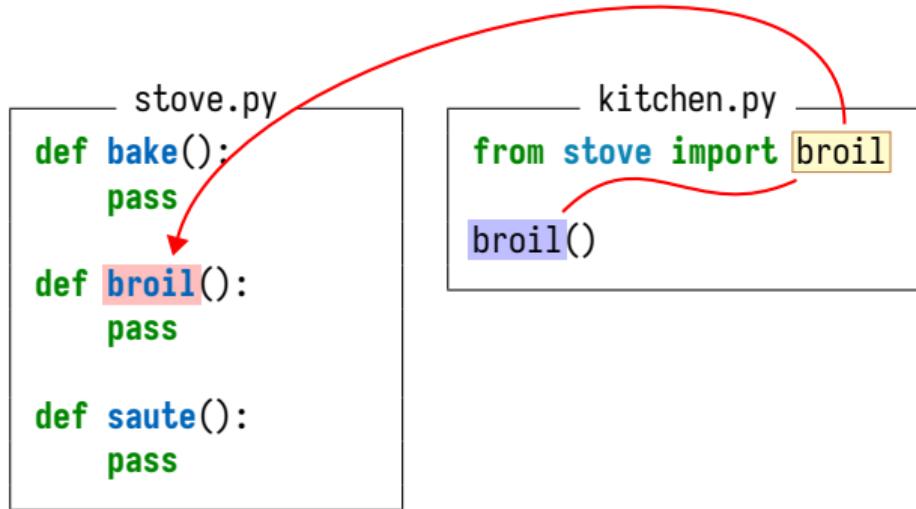
# Why is this hard?

- ▶ 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 incremental processing to handle our scale.

# Incremental results



# What would incremental results look like?



# What would incremental results look like?

stove.py

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

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

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

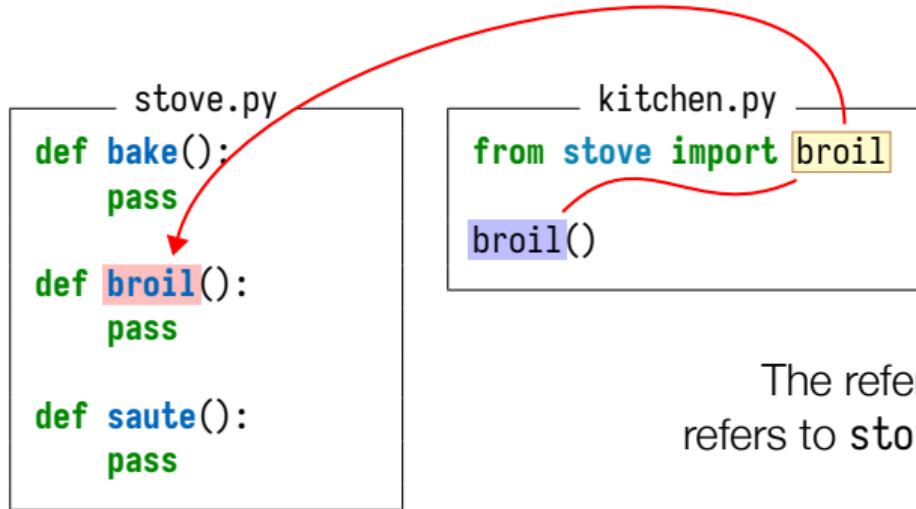
stove.broil is defined at stove.py:4:5

# 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

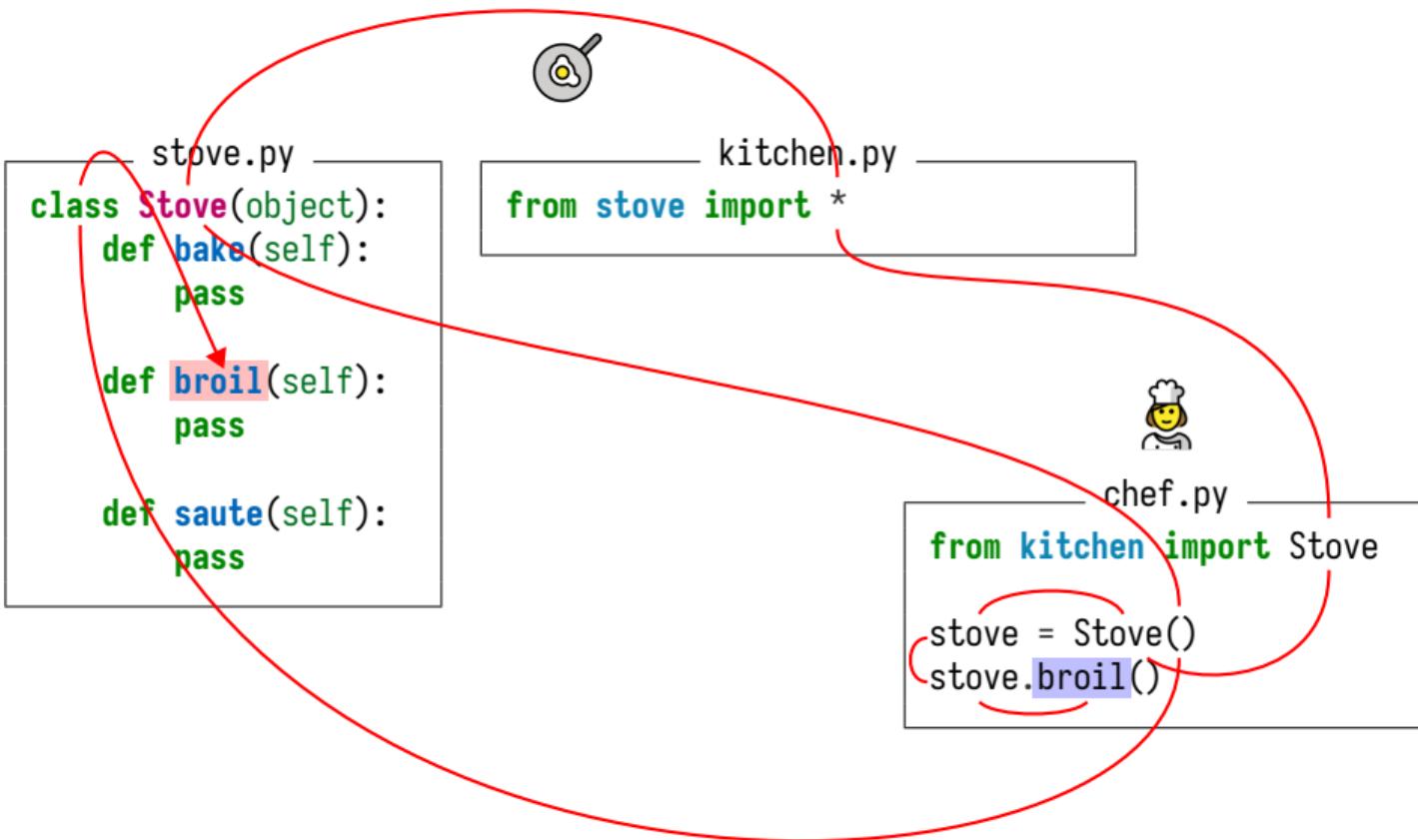
# What would incremental results look like?



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*

# The more complex example



# The more complex example

```
stove.py
class Stove(object):
    def bake(self):
        pass

    def broil(self):
        pass

    def saute(self):
        pass
```

# The more complex example

stove.py

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

    def broil(self):
        pass

    def saute(self):
        pass
```

Invoking stove.Stove

# The more complex example

```
stove.py
class Stove(object):
    def bake(self):
        pass

    def broil(self):
        pass

    def saute(self):
        pass
```

Invoking `stove.Stove`  
gives you an instance of the `Stove` class.

# The more complex example

```
stove.py
class Stove(object):
    def bake(self):
        pass

    def broil(self):
        pass

    def saute(self):
        pass
```

Invoking `stove.Stove`  
gives you an instance of the `Stove` class.

The `Stove` class

# The more complex example

```
stove.py
class Stove(object):
    def bake(self):
        pass

    def broil(self):
        pass

    def saute(self):
        pass
```

Invoking `stove.Stove`  
gives you an instance of the `Stove` class.

The `Stove` class  
has an instance member named `broil`

# The more complex example

```
stove.py
class Stove(object):
    def bake(self):
        pass

    def broil(self):
        pass

    def saute(self):
        pass
```

Invoking `stove.Stove`  
gives you an instance of the `Stove` class.

The `Stove` class  
has an instance member named `broil`  
defined at `stove.py:5:9`.

# The more complex example

```
kitchen.py  
from stove import *
```

## The more complex example

```
kitchen.py  
from stove import *
```

If you are looking for kitchen.[anything]

## The more complex example

```
kitchen.py  
from stove import *
```

If you are looking for `kitchen.[anything]`  
then you might find it at `stove.[anything]`.

# The more complex example

chef.py

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

A diagram illustrating a Python code snippet. The code is contained within a rectangular box labeled "chef.py" at the top. Inside the box, the code is as follows:  
`from kitchen import Stove  
stove = Stove()  
stove.broil()`

A red oval is drawn around the `stove.broil()` line, highlighting the method call. The word "broil" is also highlighted with a blue rectangle.

# The more complex example

chef.py

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

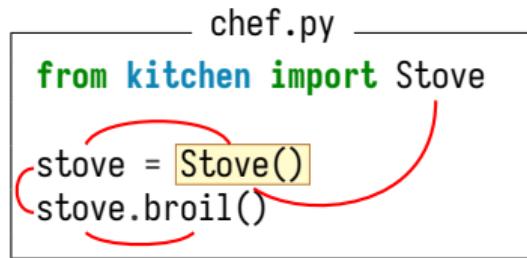
The diagram shows a code snippet in a box. At the top, it says "chef.py". Below that is the Python code: "from kitchen import Stove", "stove = Stove()", and "stove.broil()". Red arrows point from the word "kitchen" in the first line to the "kitchen" module, and from the word "Stove" in the same line to the "Stove" class definition in the second line.

If you can find what `kitchen.Stove` resolves to

# The more complex example

chef.py

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



If you can find what `kitchen.Stove` resolves to and can call it

# The more complex example

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

A diagram showing a code snippet in a box. A red arrow points from the word "kitchen" in the import statement to the "broil()" method call, indicating they both resolve to the same object.

If you can find what `kitchen.Stove` resolves to and can call it then the result should have a member named `broil`

# The more complex example

chef.py

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

The code in `chef.py` is:

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

Annotations:

- A red circle highlights the word `kitchen` in the `import` statement.
- A red arrow points from the `broil()` method call in the code below to the word `broil` in the same line.

If you can find what `kitchen.Stove` resolves to and can call it then the result should have a member named `broil` which the reference at `chef.py:4:7` resolves to.

## The more complex example

If you can find what `kitchen.Stove` resolves to and can call it then the result should have a member named `broil` which the reference at `chef.py:4:7` resolves to.

## The more complex example

If you can find what `kitchen.Stove` resolves to and can call it then the result should have a member named `broil` which the reference at `chef.py:4:7` resolves to.

+

If you are looking for `kitchen.[anything]` then you might find it at `stove.[anything]`.

## The more complex example

If you can find what `stove.Stove` resolves to and can call it then the result should have a member named `broil` which the reference at `chef.py:4:7` resolves to.

## The more complex example

If you can find what `stove.Stove` resolves to and can call it then the result should have a member named `broil` which the reference at `chef.py:4:7` resolves to.

+

Invoking `stove.Stove`  
gives you an instance of the `Stove` class.

## The more complex example

The `Stove` class  
should have a member named `broil`  
which the reference at `chef.py:4:7` resolves to.

## The more complex example

The **Stove** class  
should have a member named **broil**  
which the reference at *chef.py:4:7* resolves to.

+

The **Stove** class  
has an instance member named **broil**  
defined at *stove.py:5:9*.

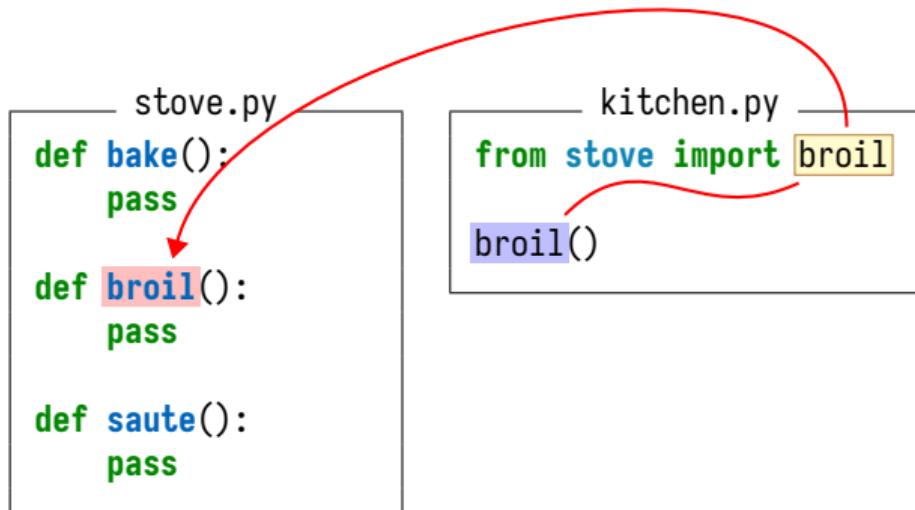
## The more complex example

The definition at *stove.py:5:9*  
is what the reference at *chef.py:4:7* resolves to.

# Stack graphs



# Stack graphs



# Stack graphs

stove.py

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

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

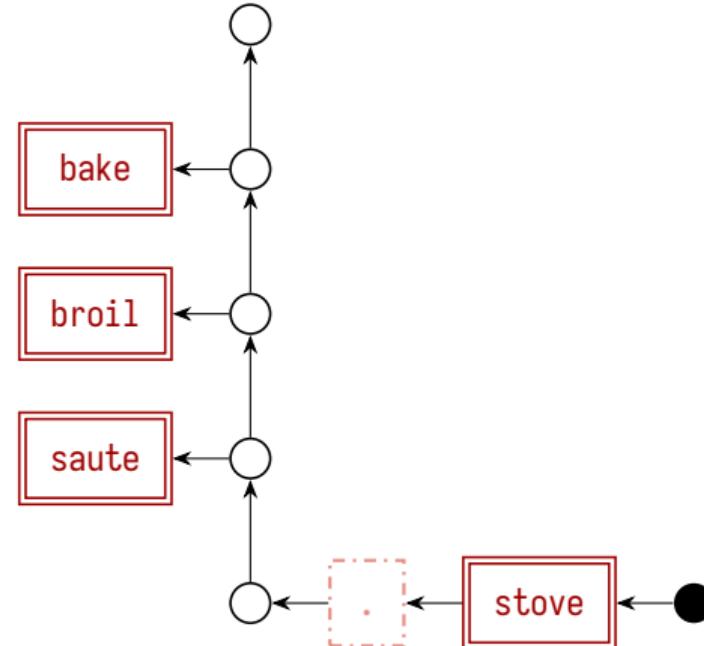
```
def saute():
    pass
```

# Stack graphs

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

def broil():
    pass

def saute():
    pass
```

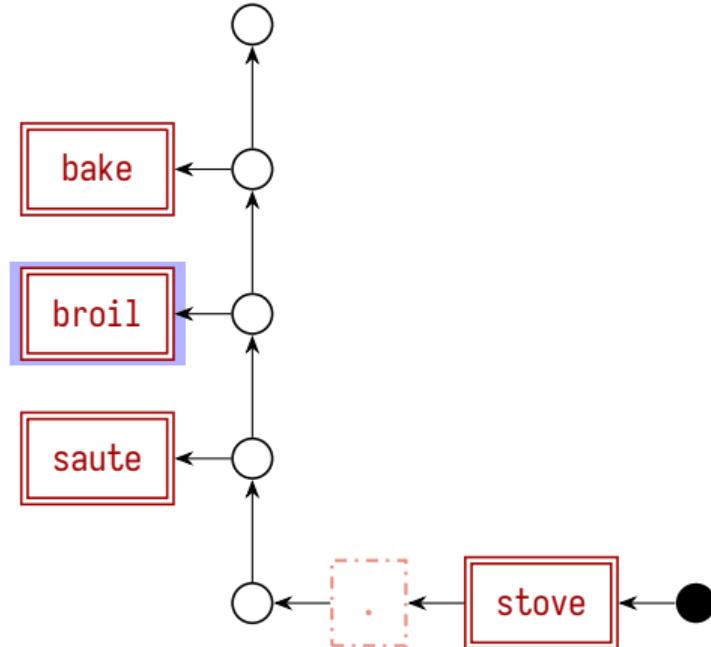


# Stack graphs

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

def broil():
    pass

def saute():
    pass
```

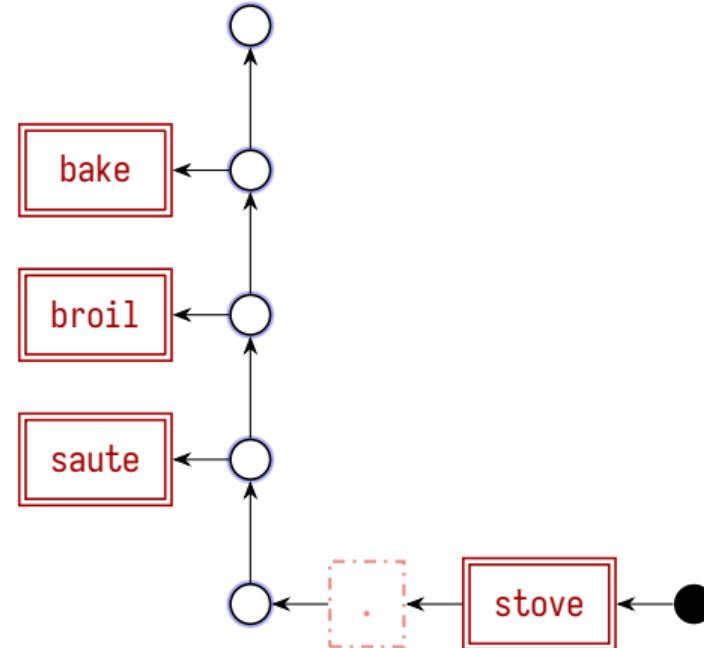


# Stack graphs

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

def broil():
    pass

def saute():
    pass
```

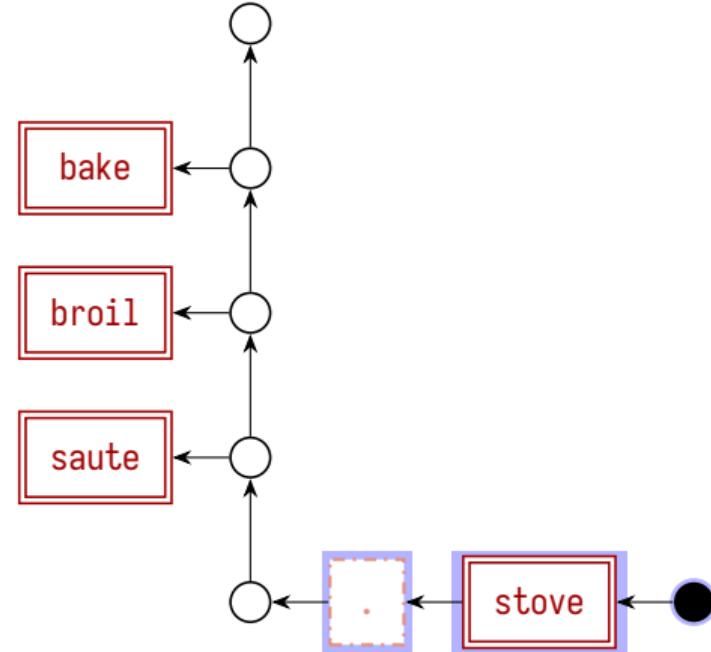


# Stack graphs

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

def broil():
    pass

def saute():
    pass
```



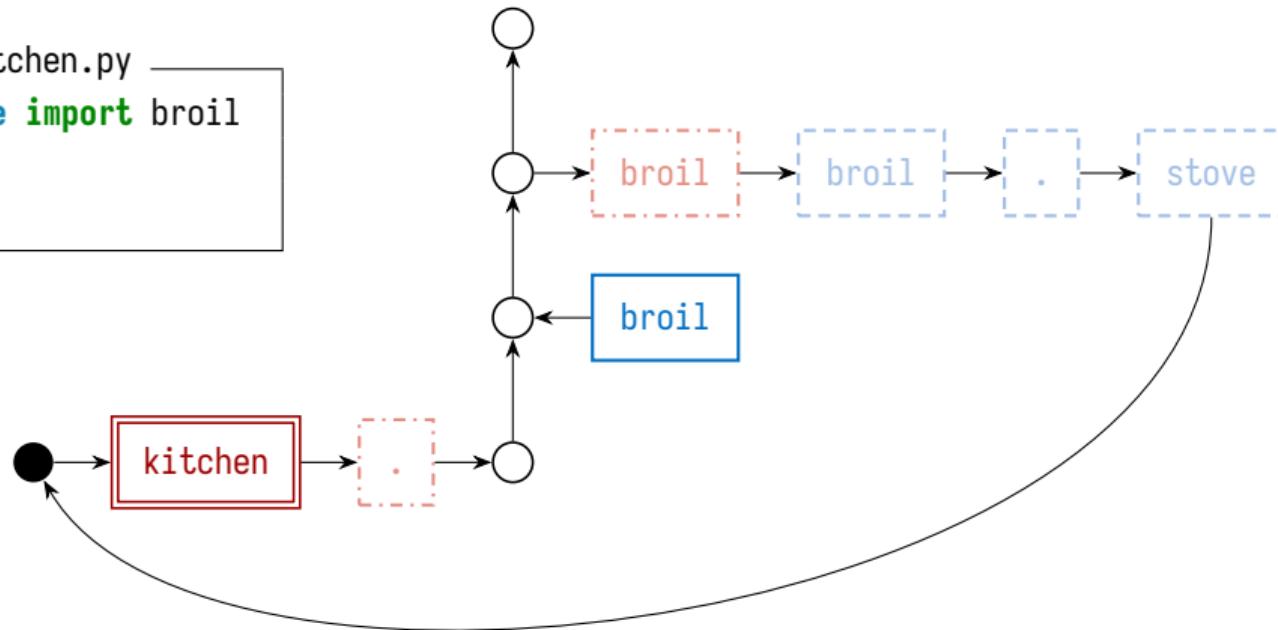
# Stack graphs

kitchen.py

```
from stove import broil  
broil()
```

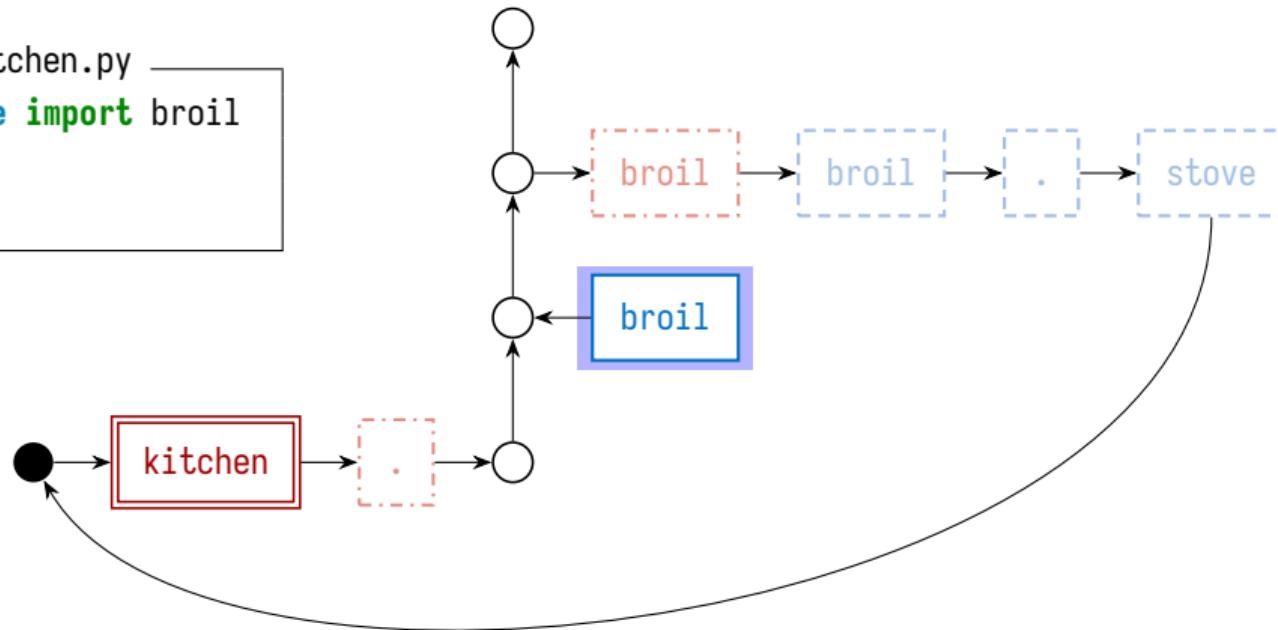
# Stack graphs

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



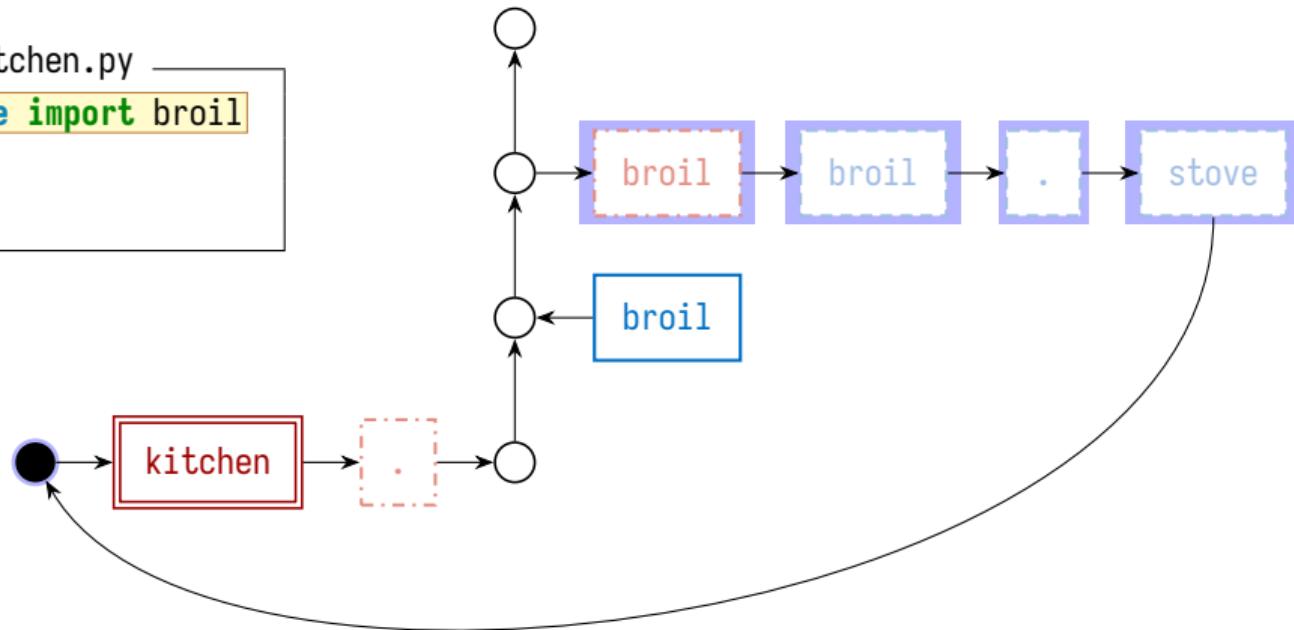
# Stack graphs

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

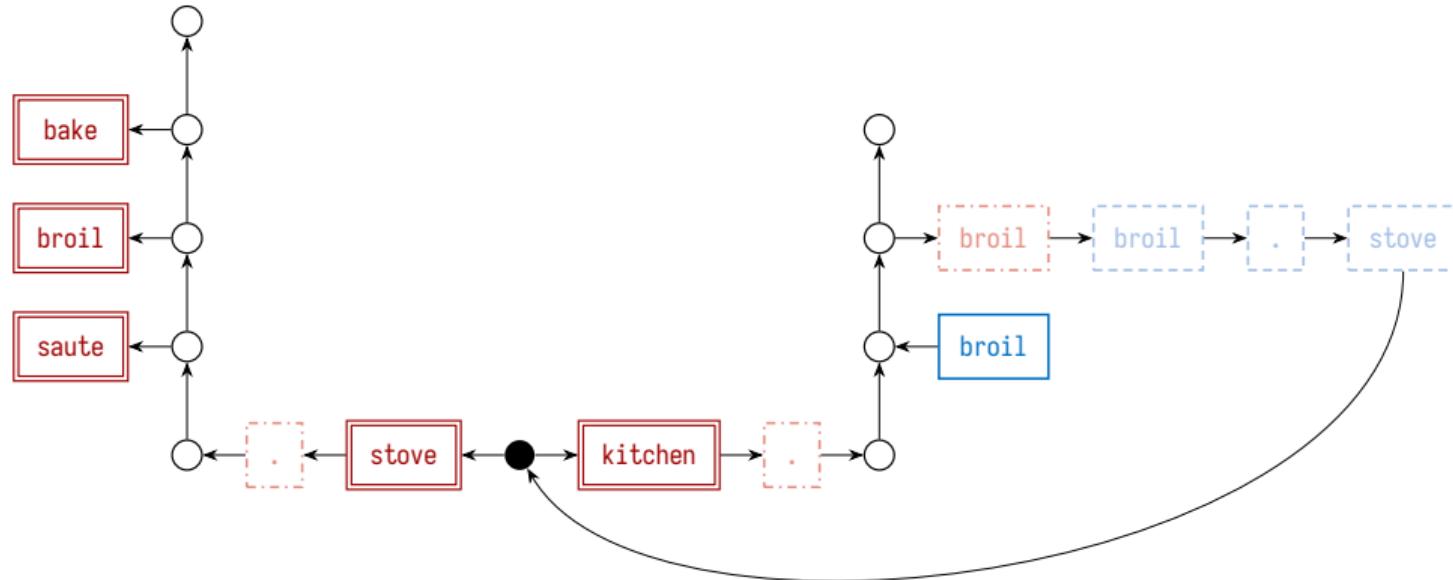


# Stack graphs

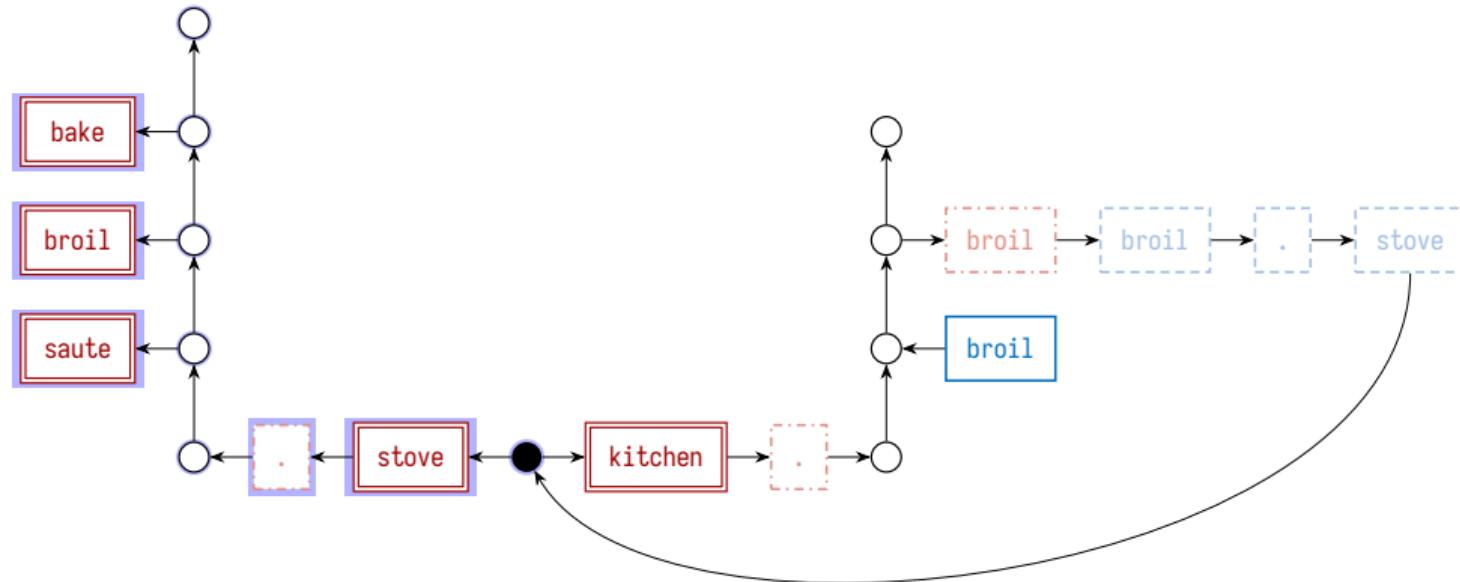
```
kitchen.py
from stove import broil
broil()
```



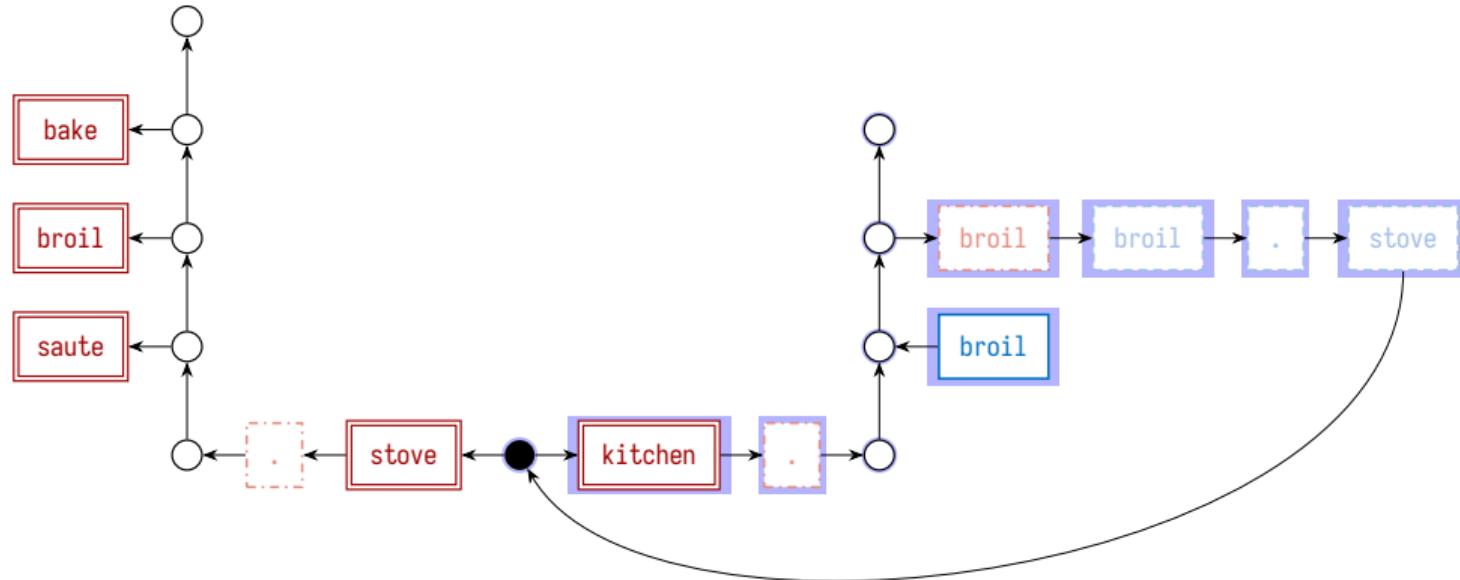
# Stack graphs



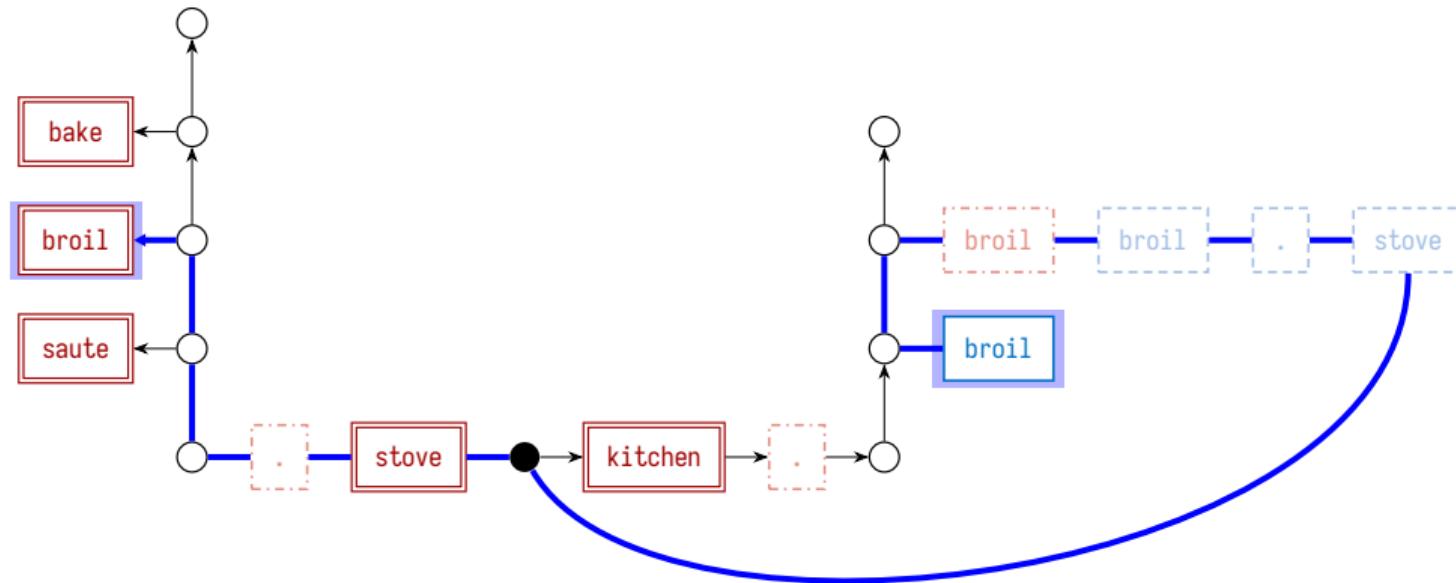
# Stack graphs



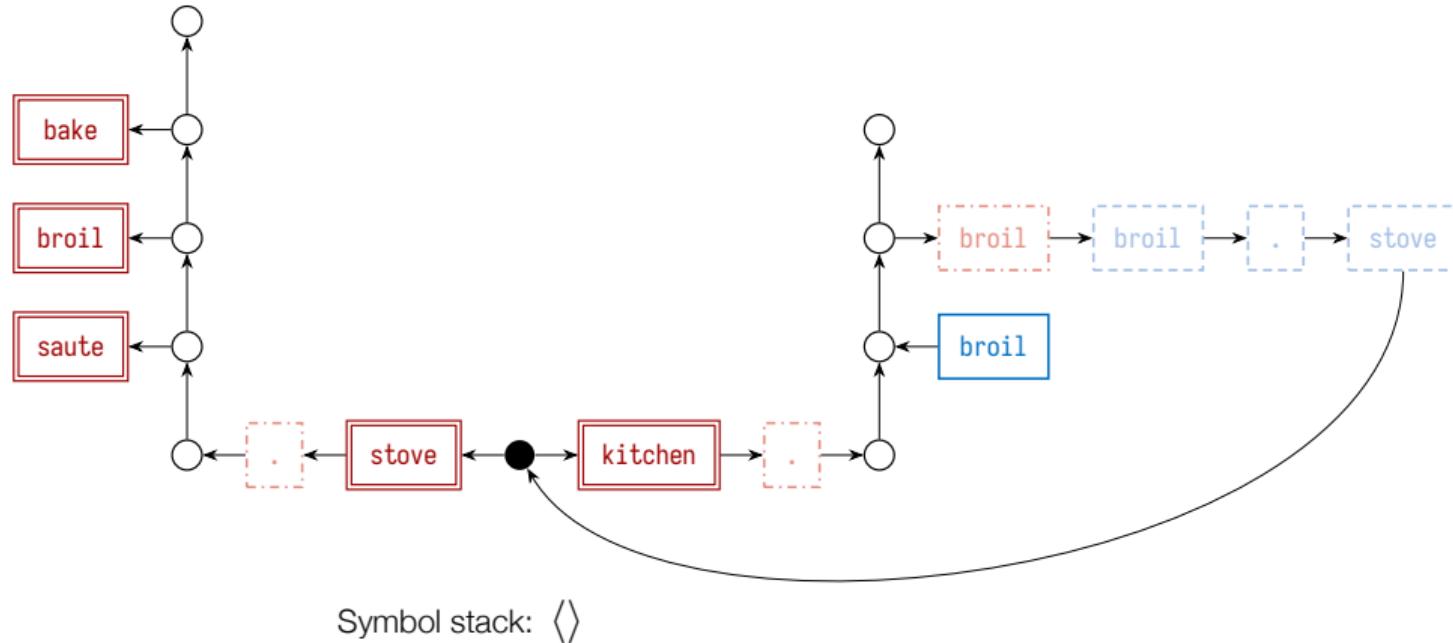
# Stack graphs



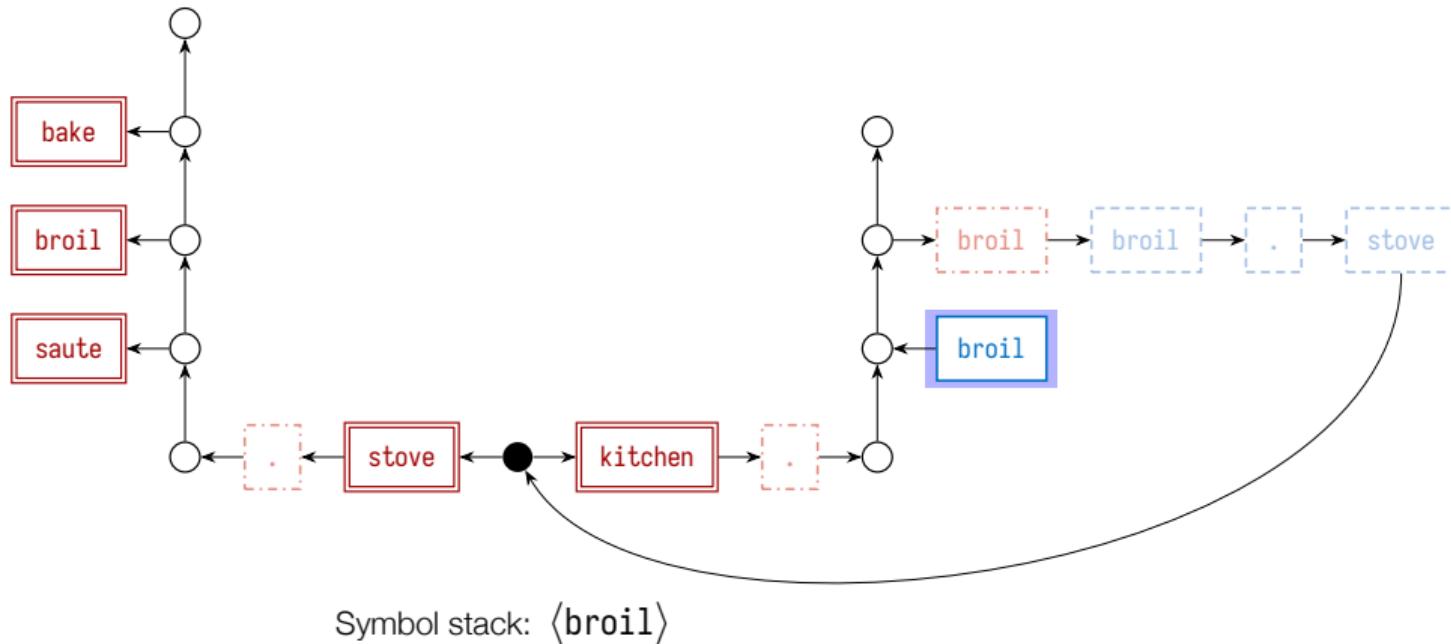
# Stack graphs



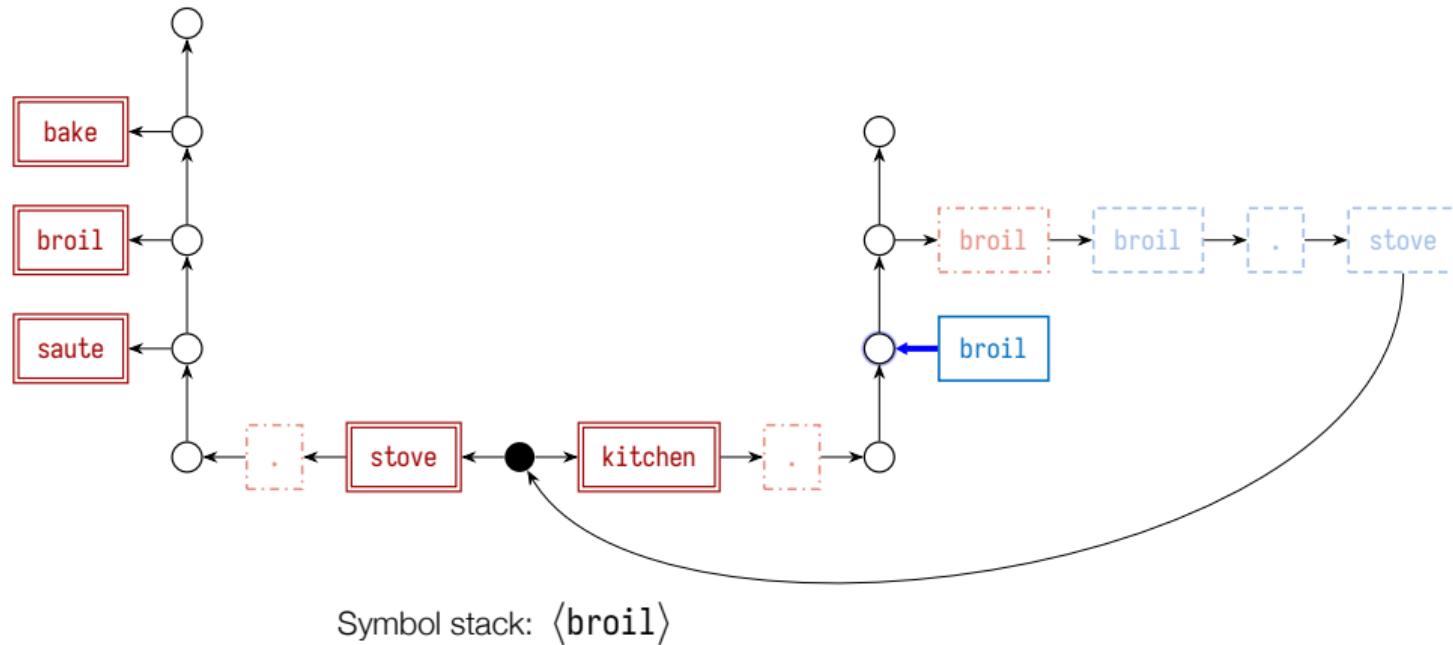
# Stack graphs



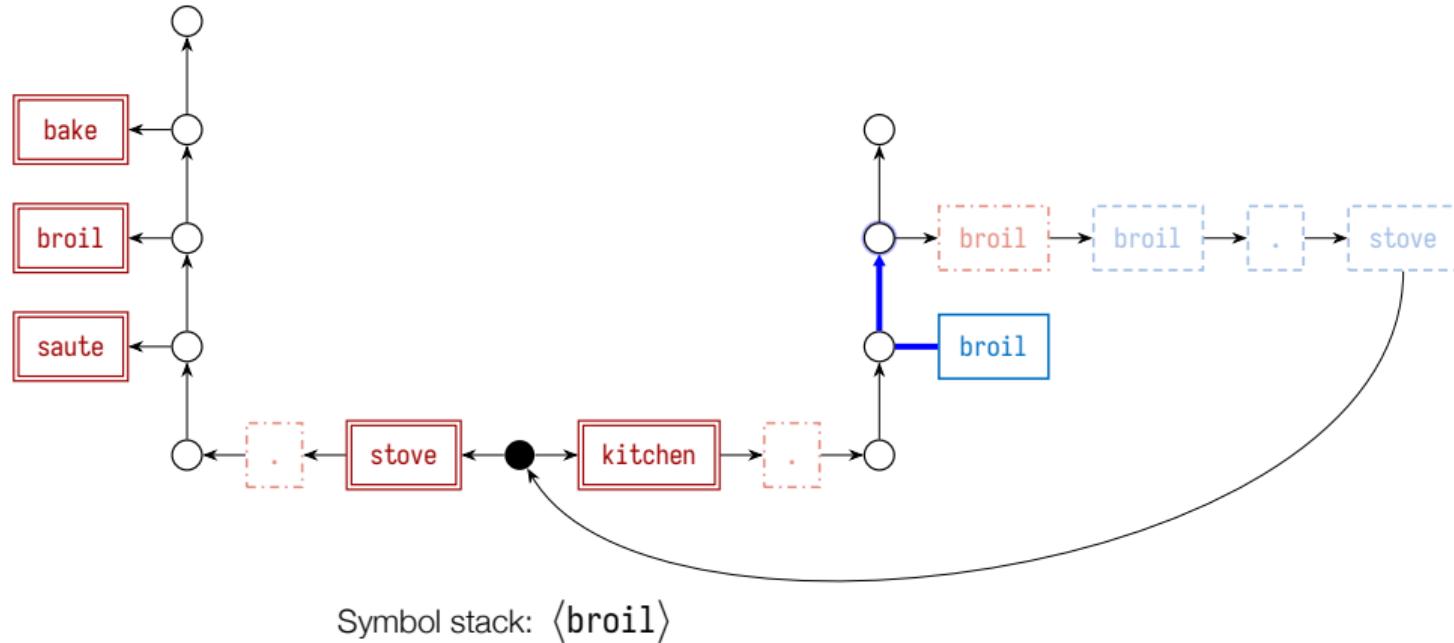
# Stack graphs



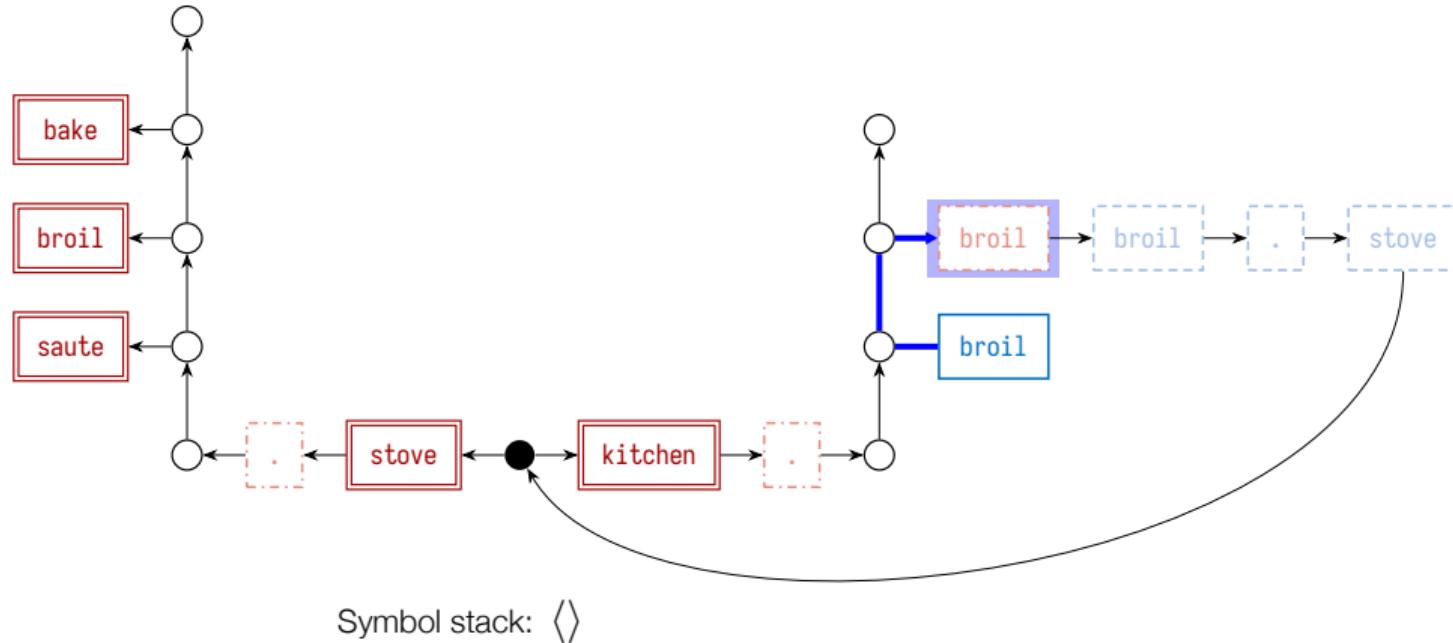
# Stack graphs



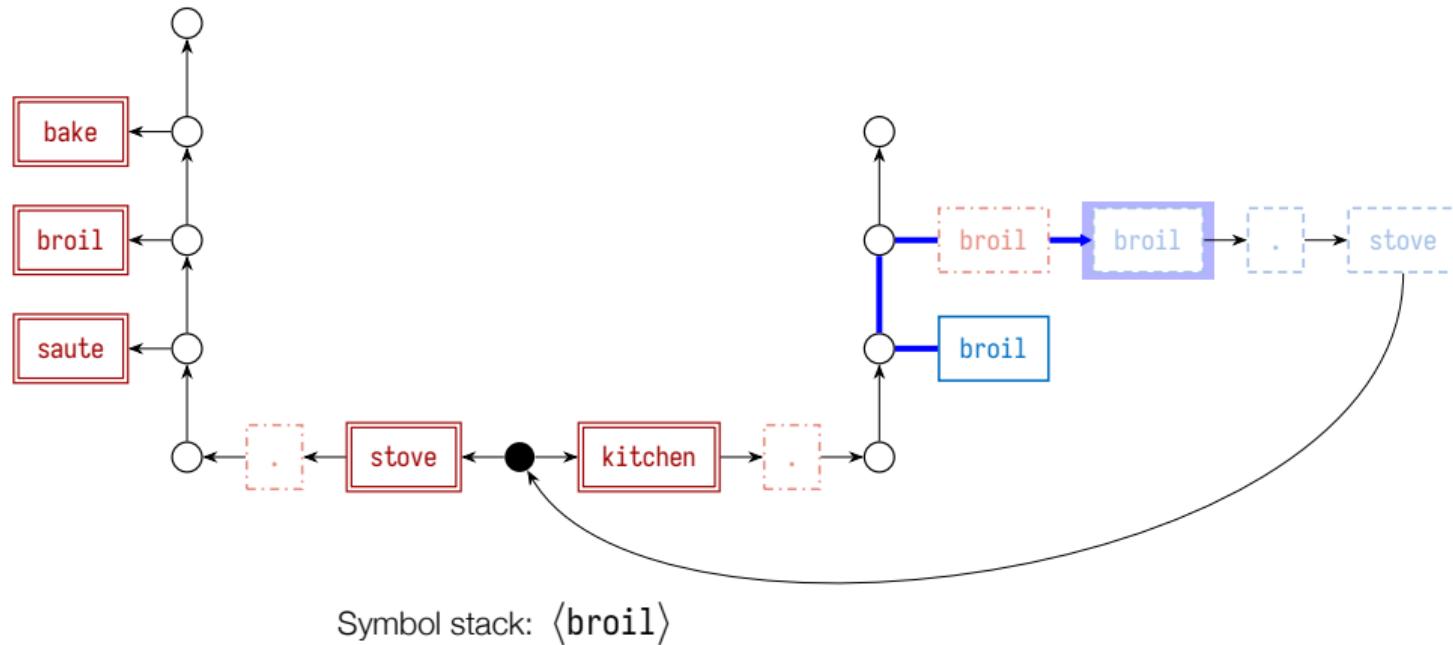
# Stack graphs



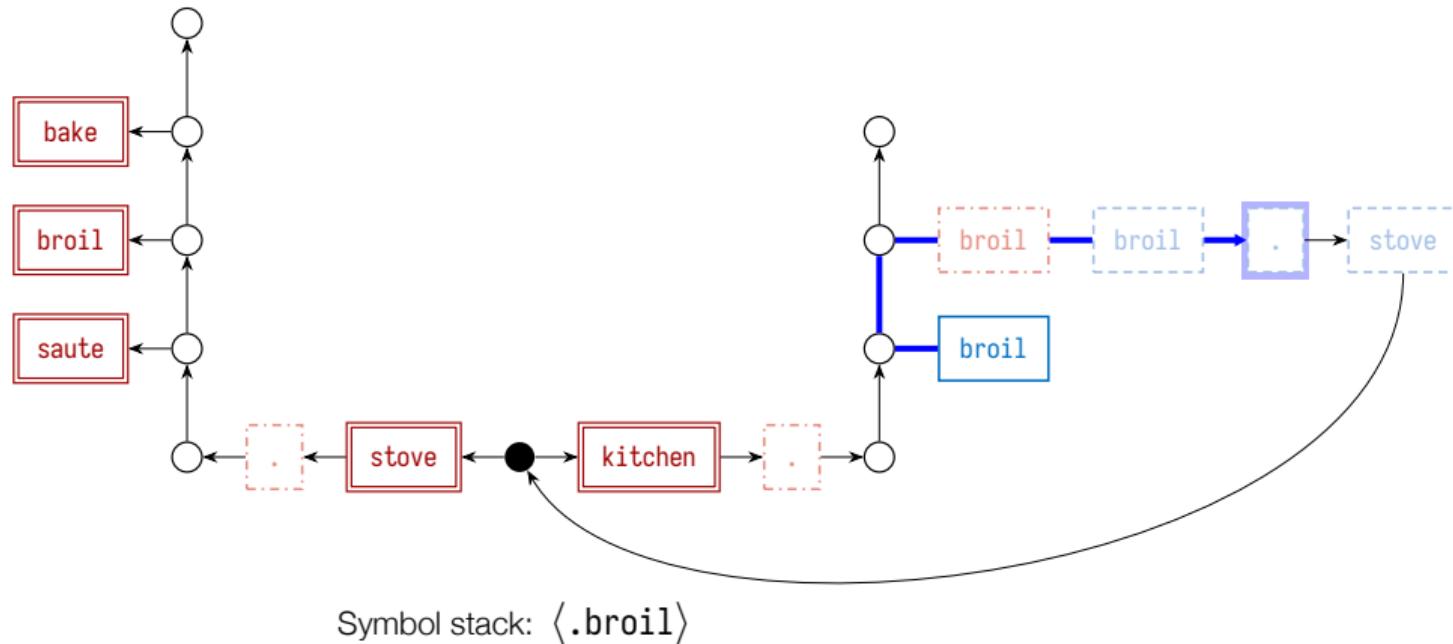
# Stack graphs



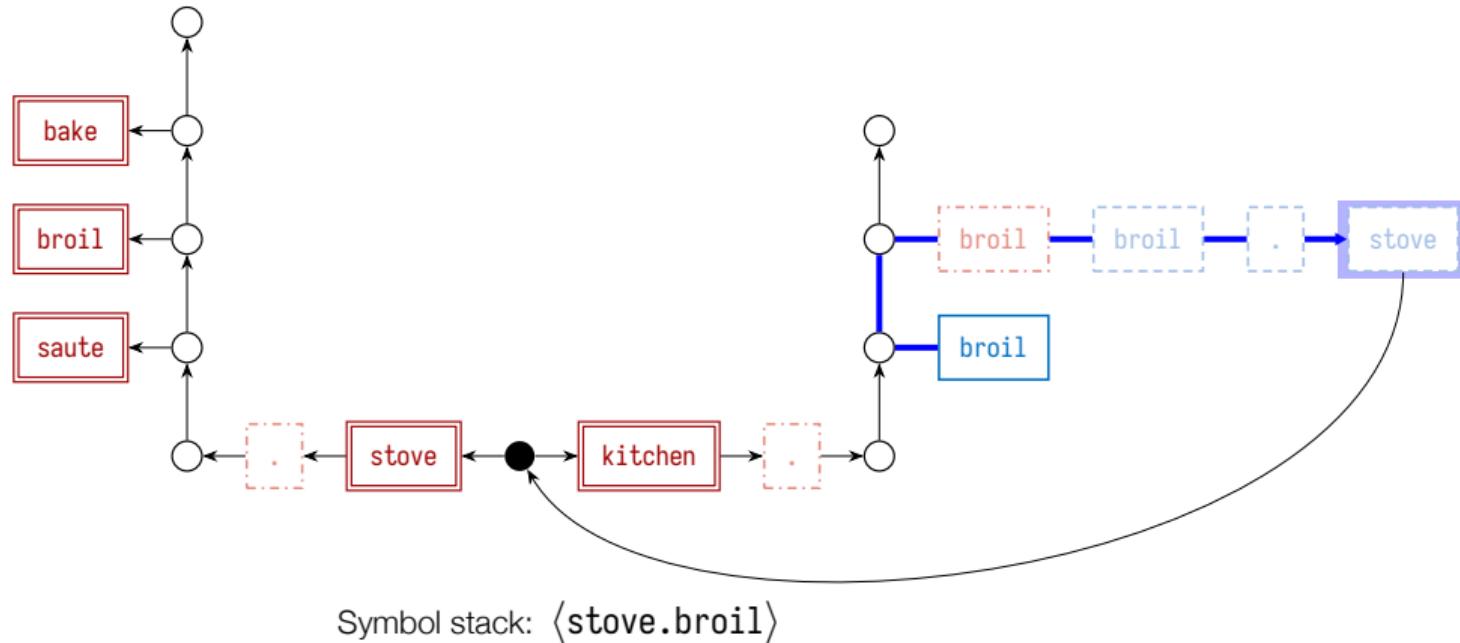
# Stack graphs



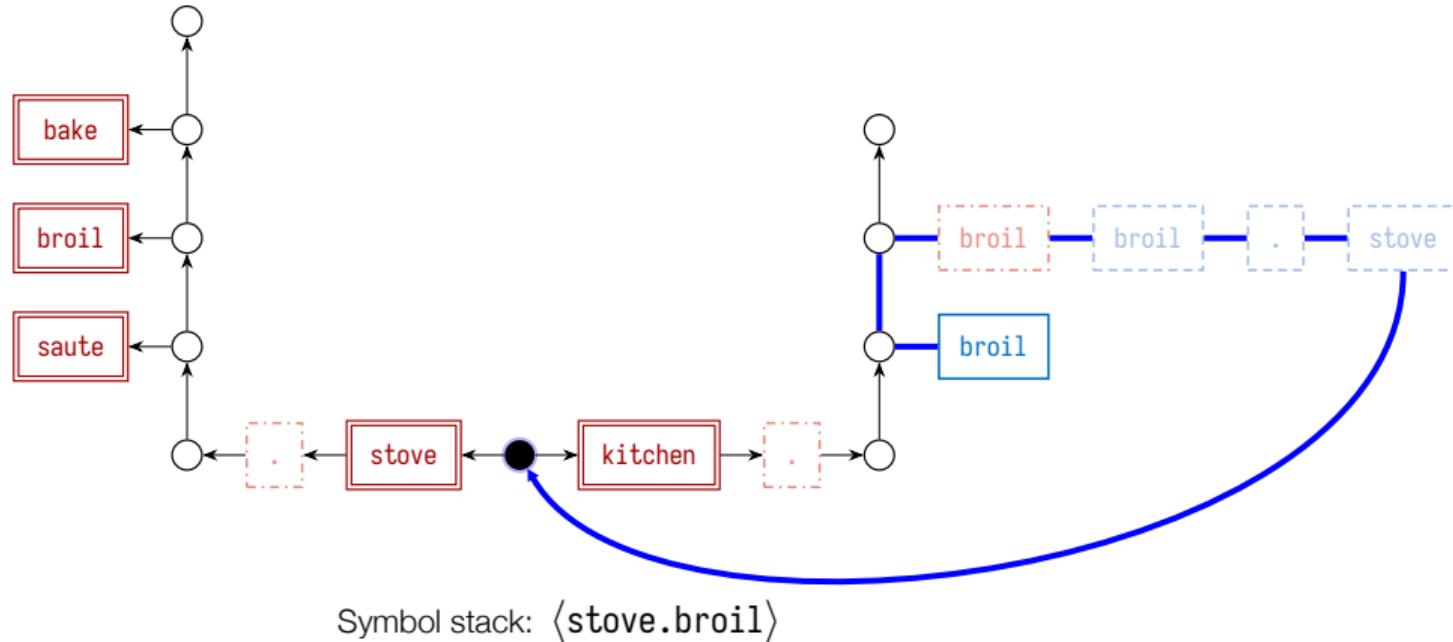
# Stack graphs



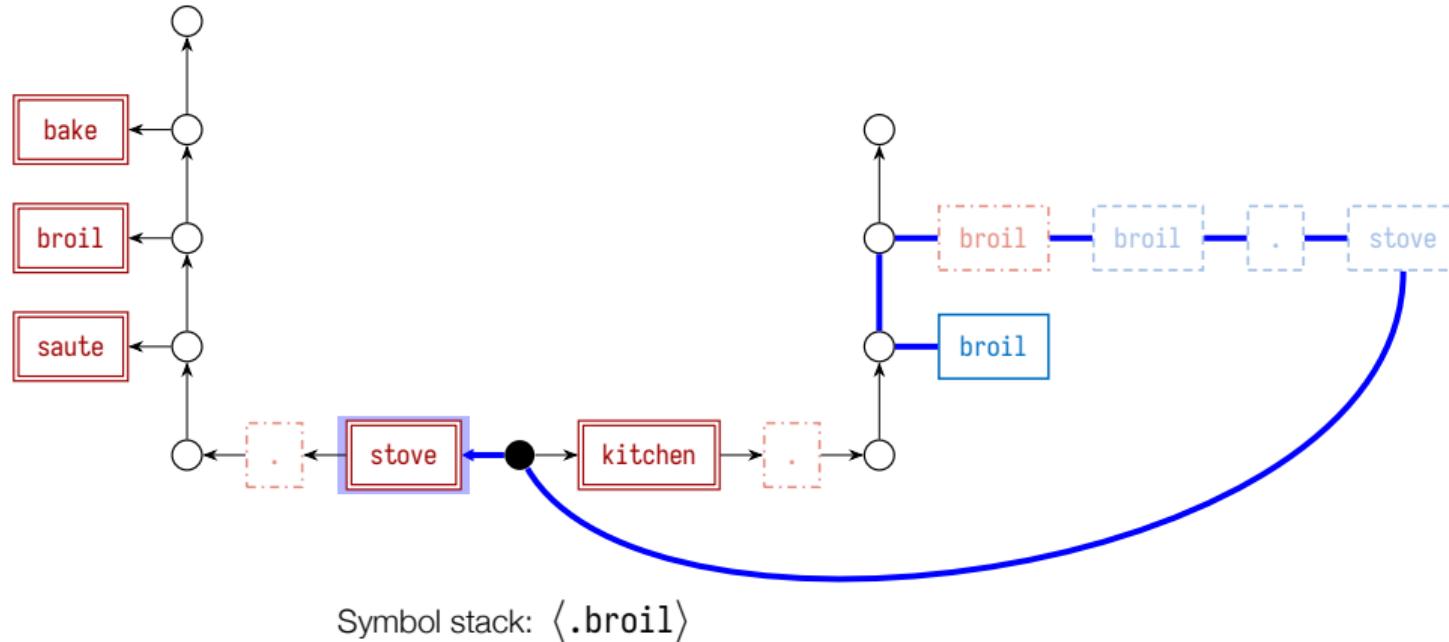
# Stack graphs



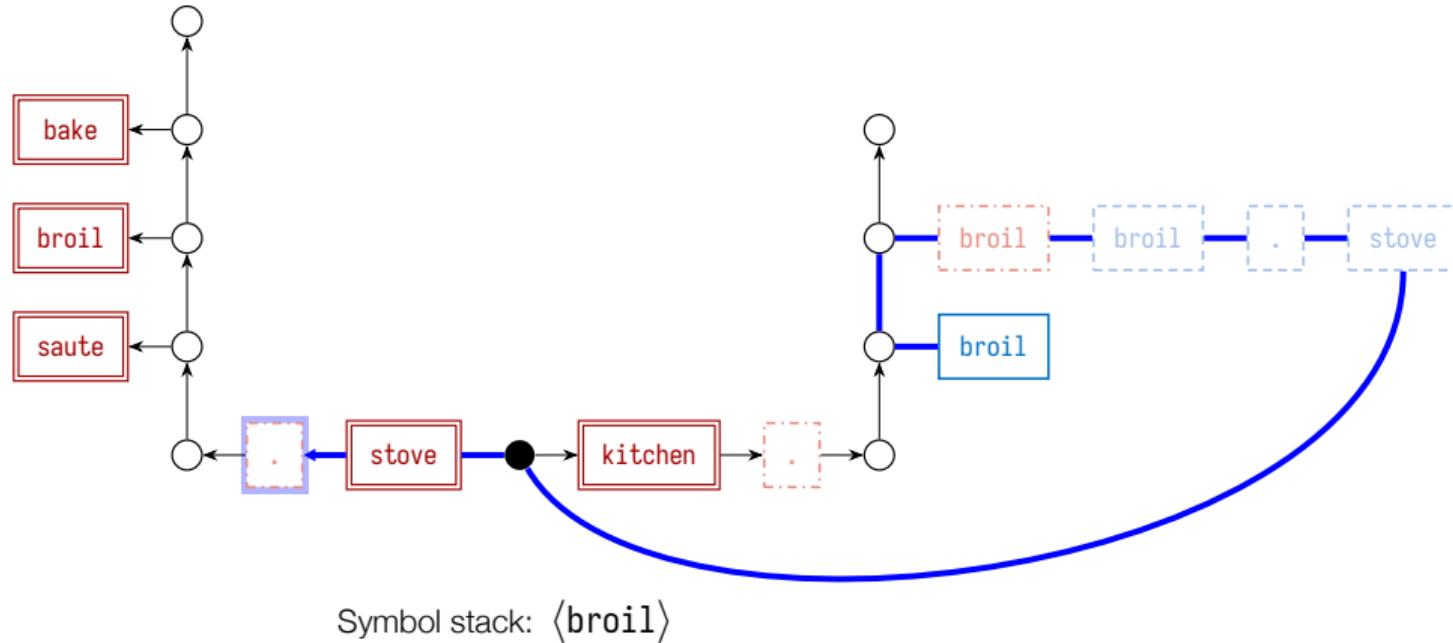
# Stack graphs



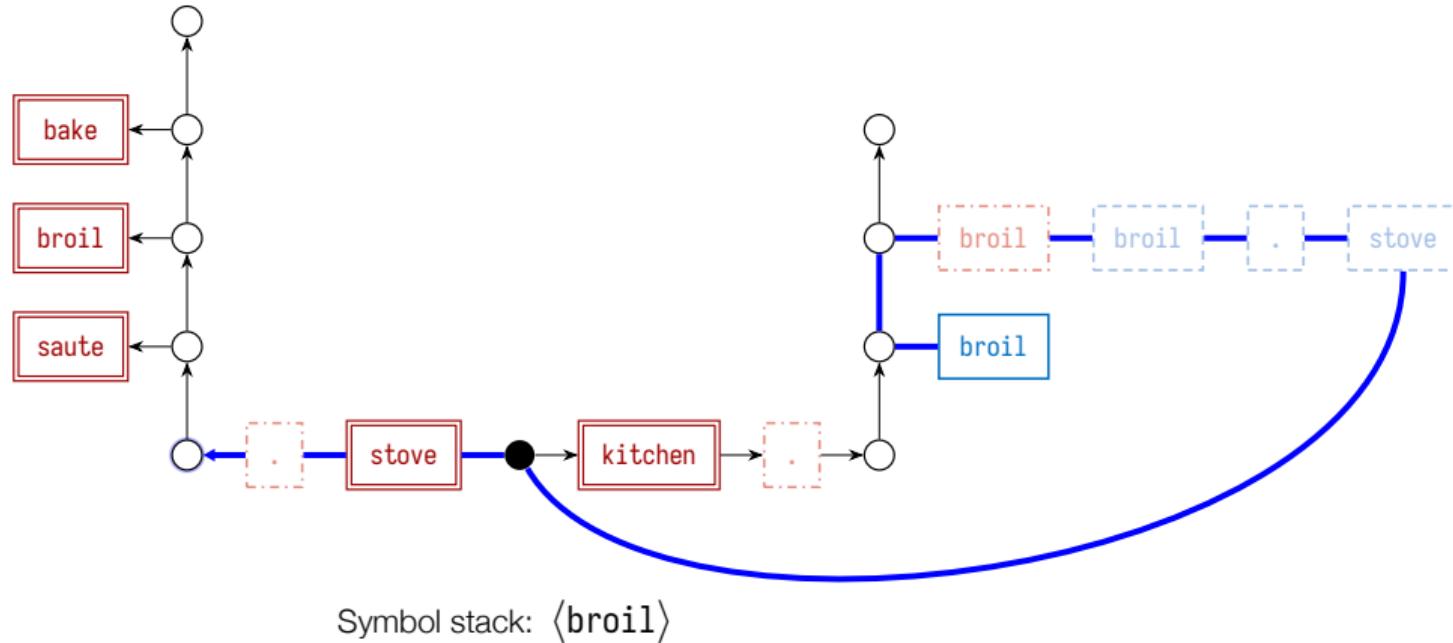
# Stack graphs



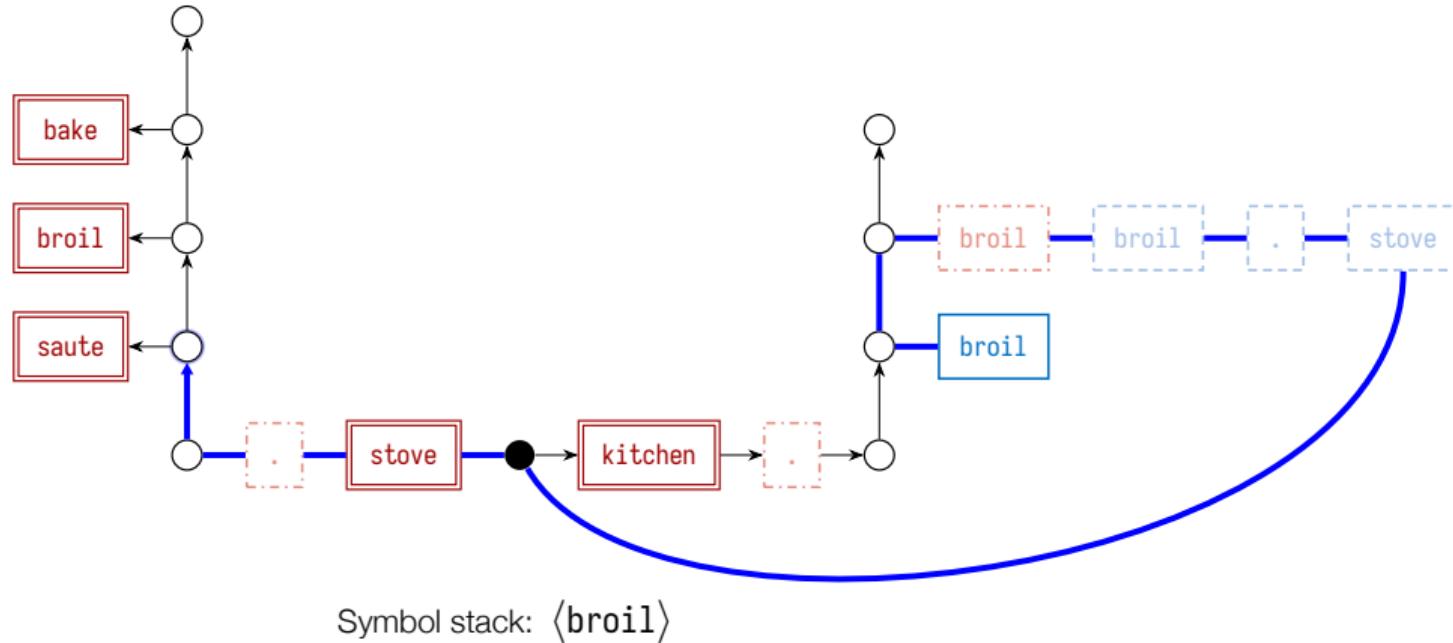
# Stack graphs



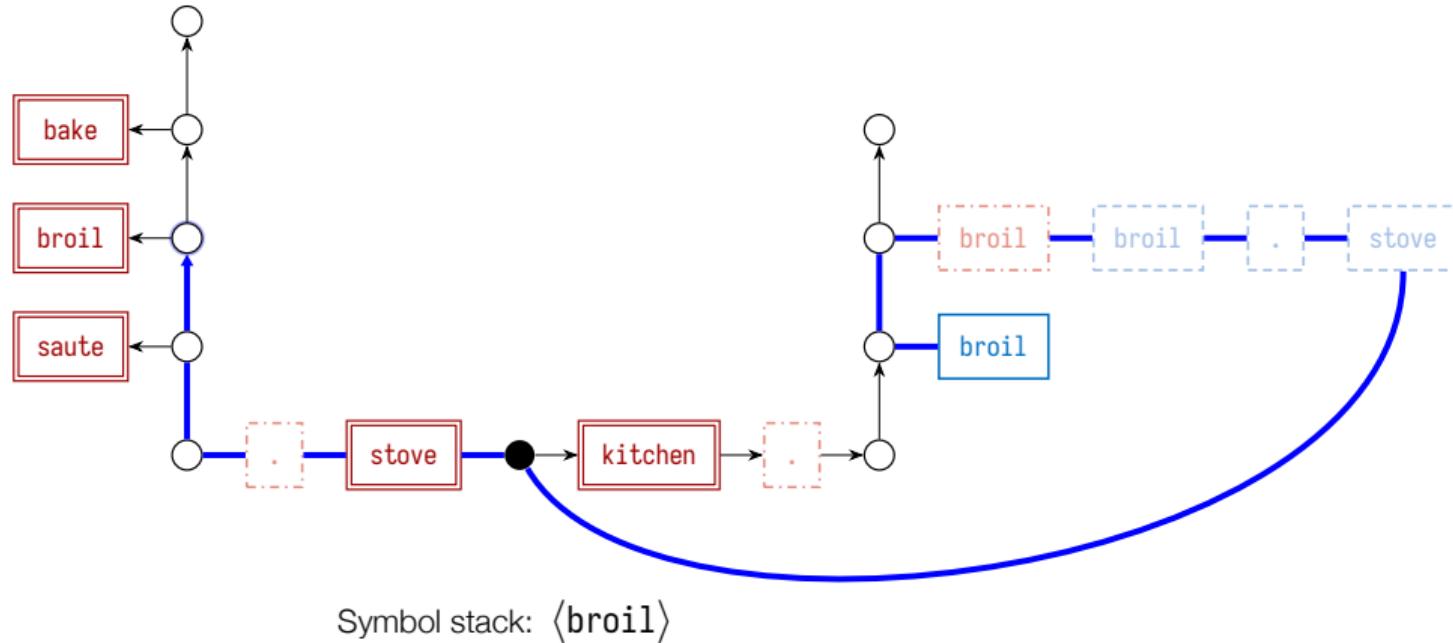
# Stack graphs



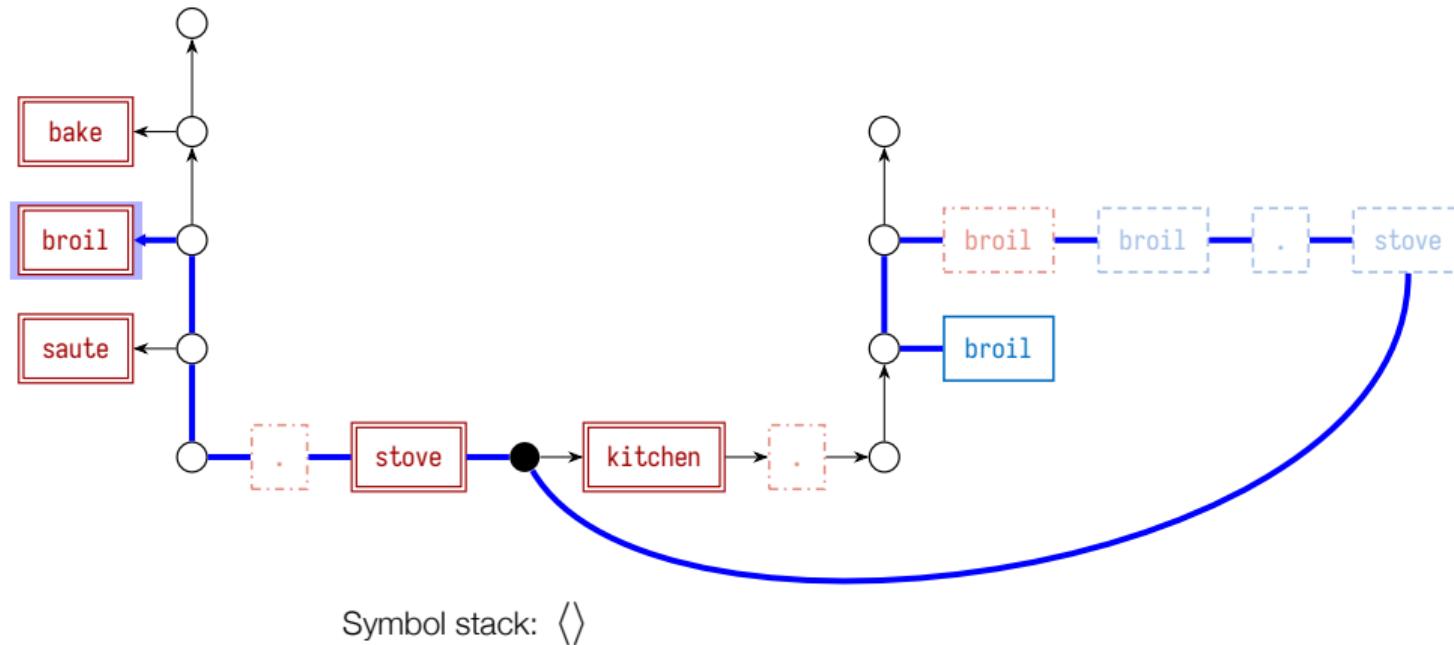
# Stack graphs



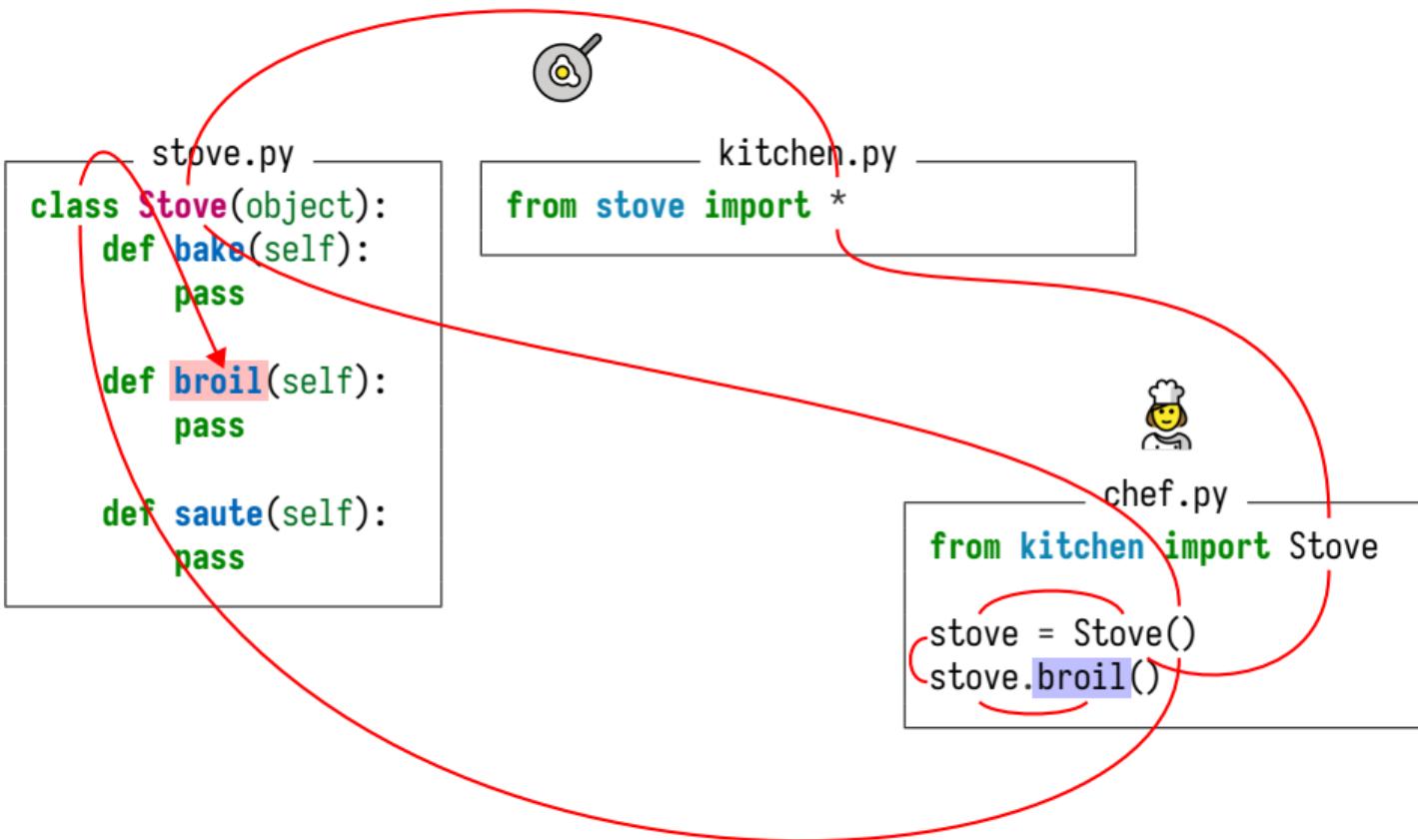
# Stack graphs



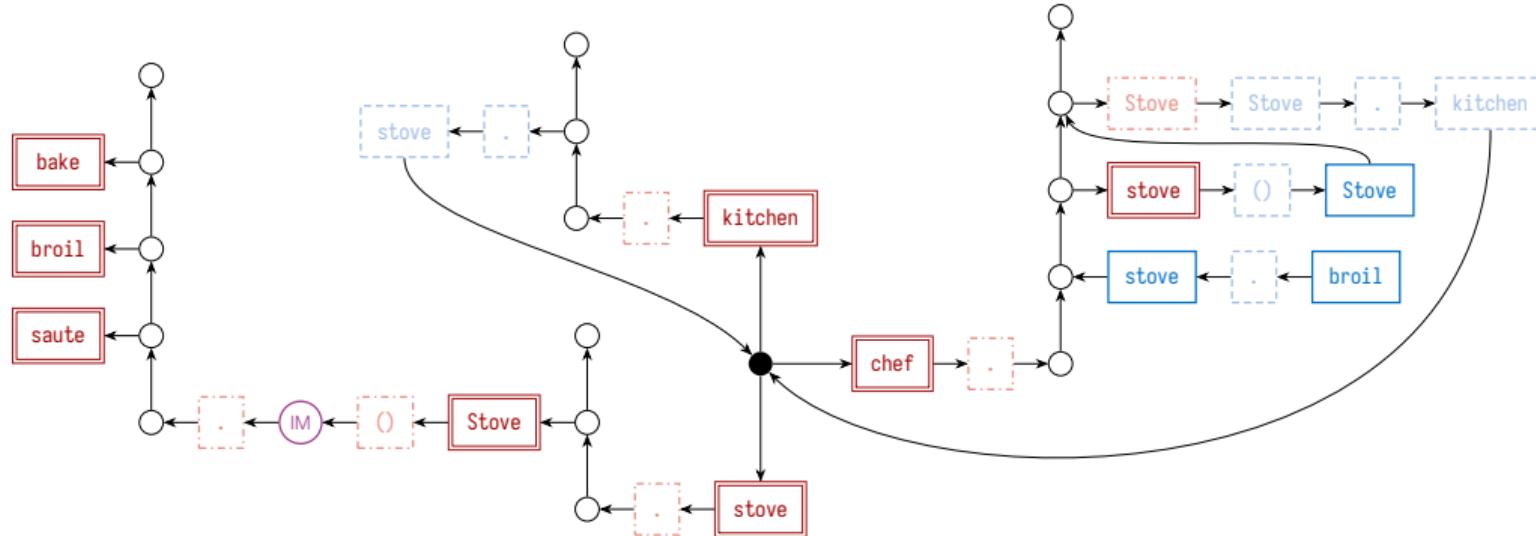
# Stack graphs



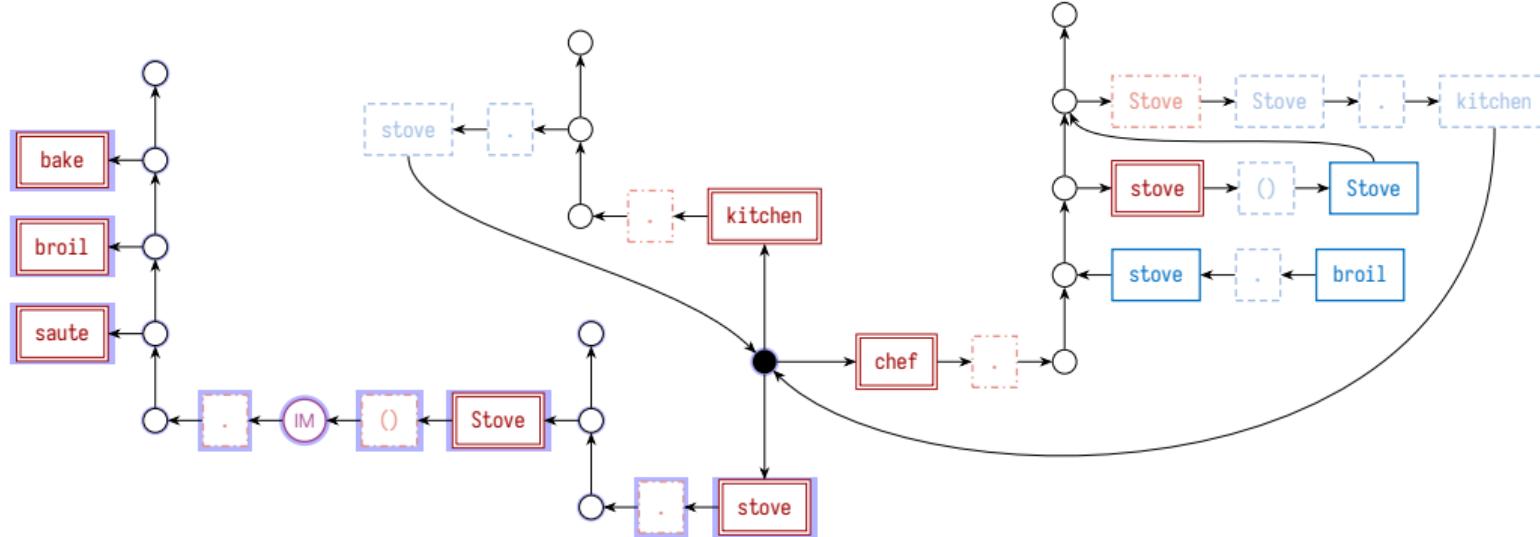
# The more complex example



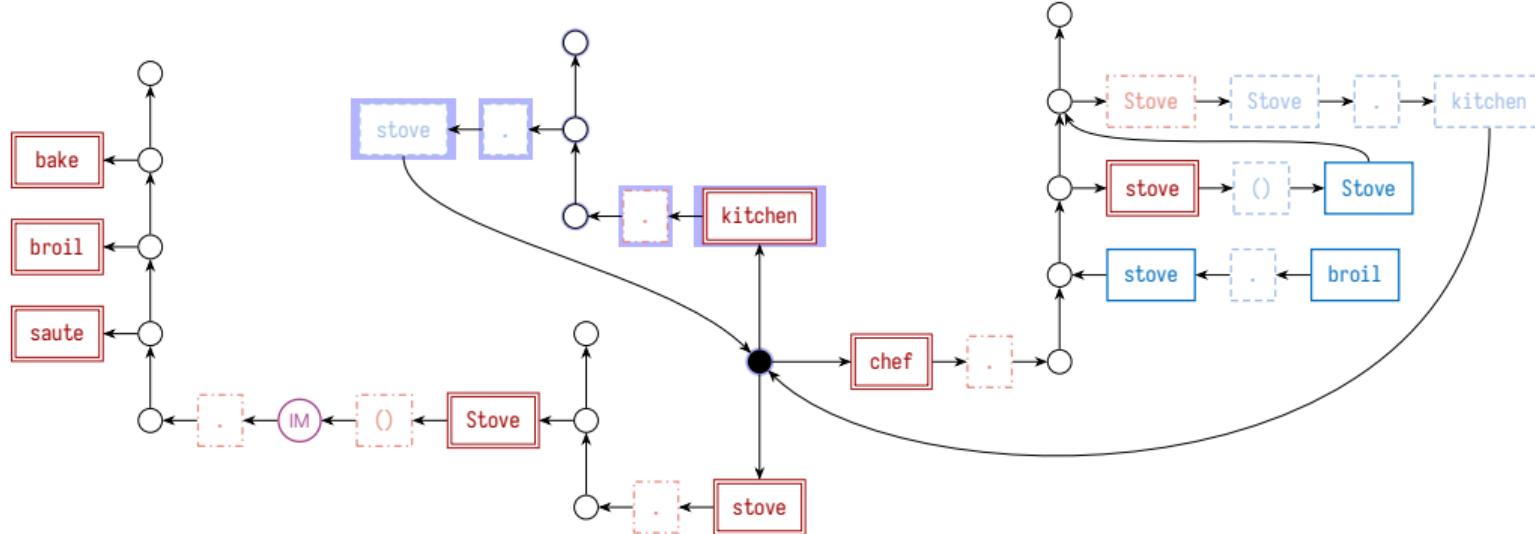
# The more complex example



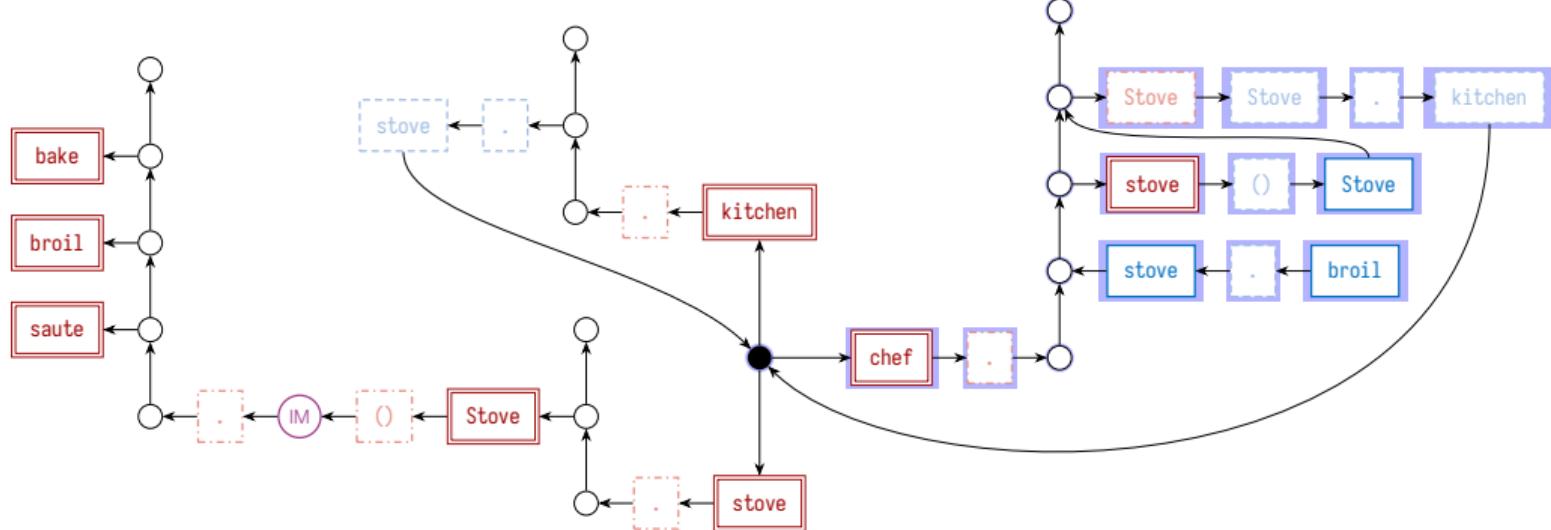
# The more complex example



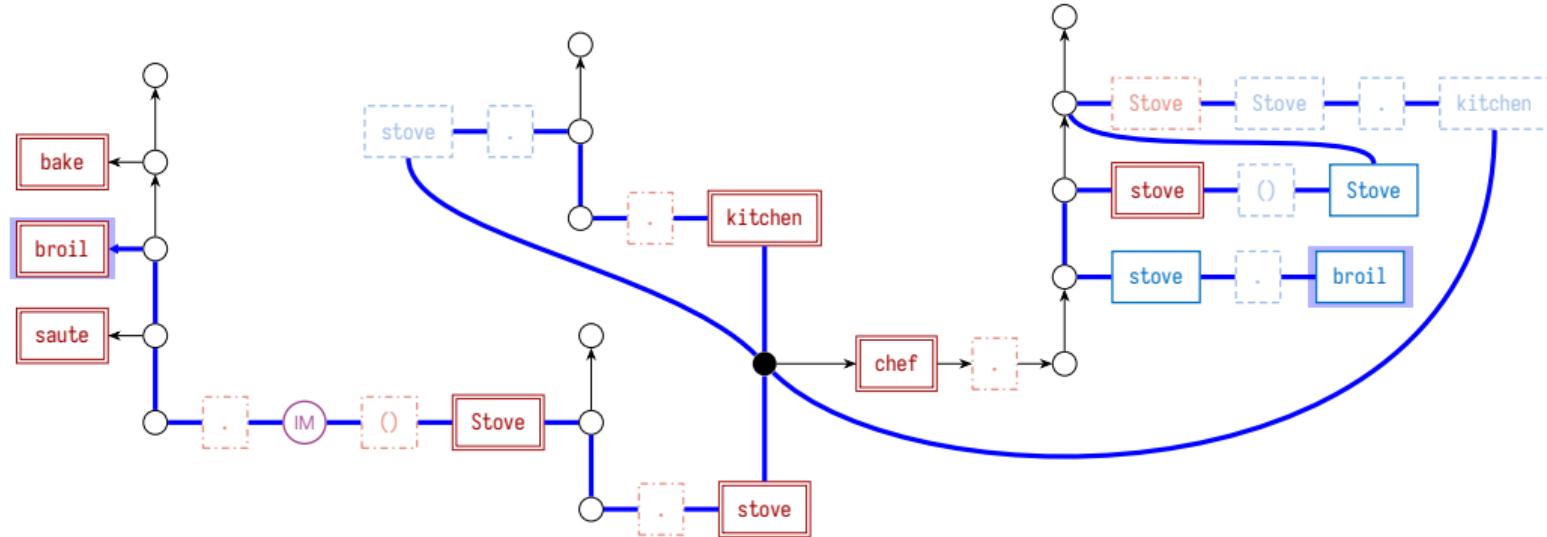
# The more complex example



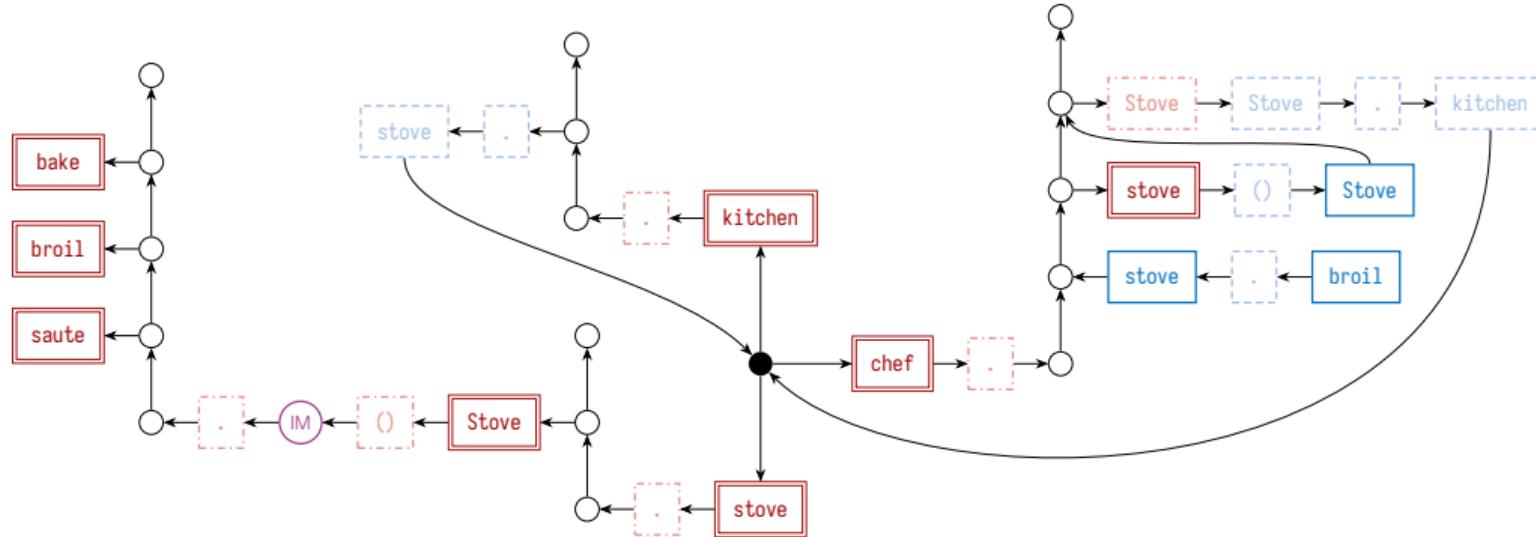
# The more complex example



## The more complex example

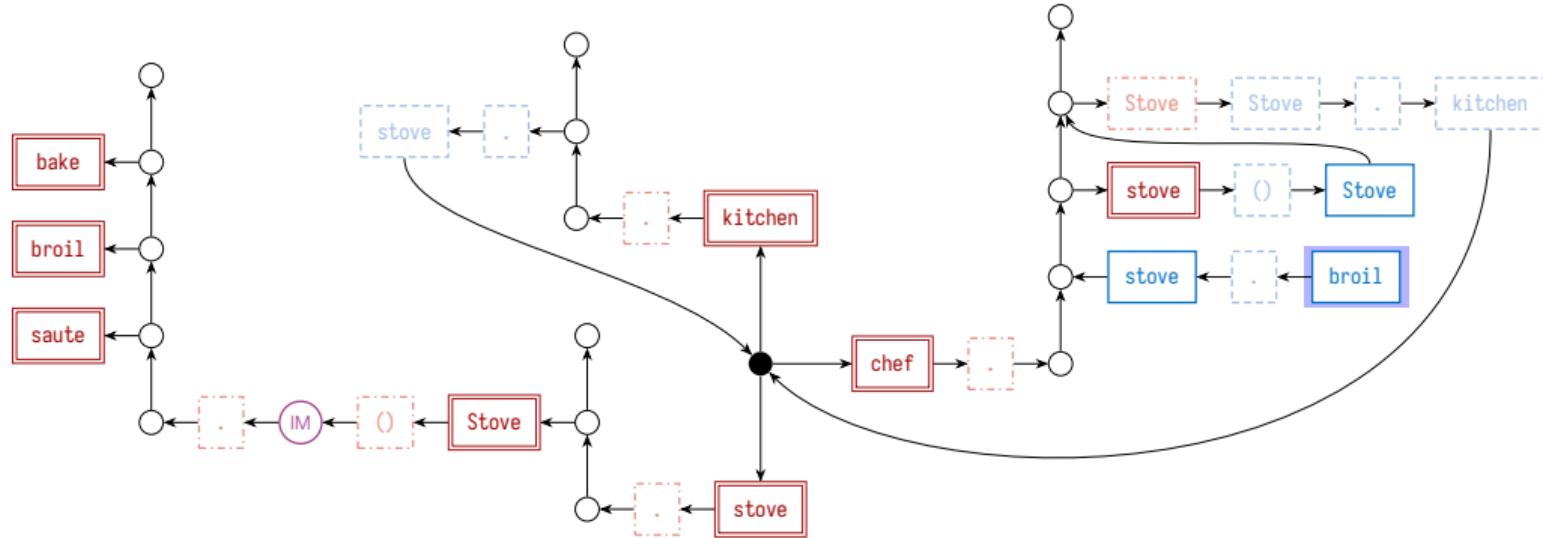


# The more complex example



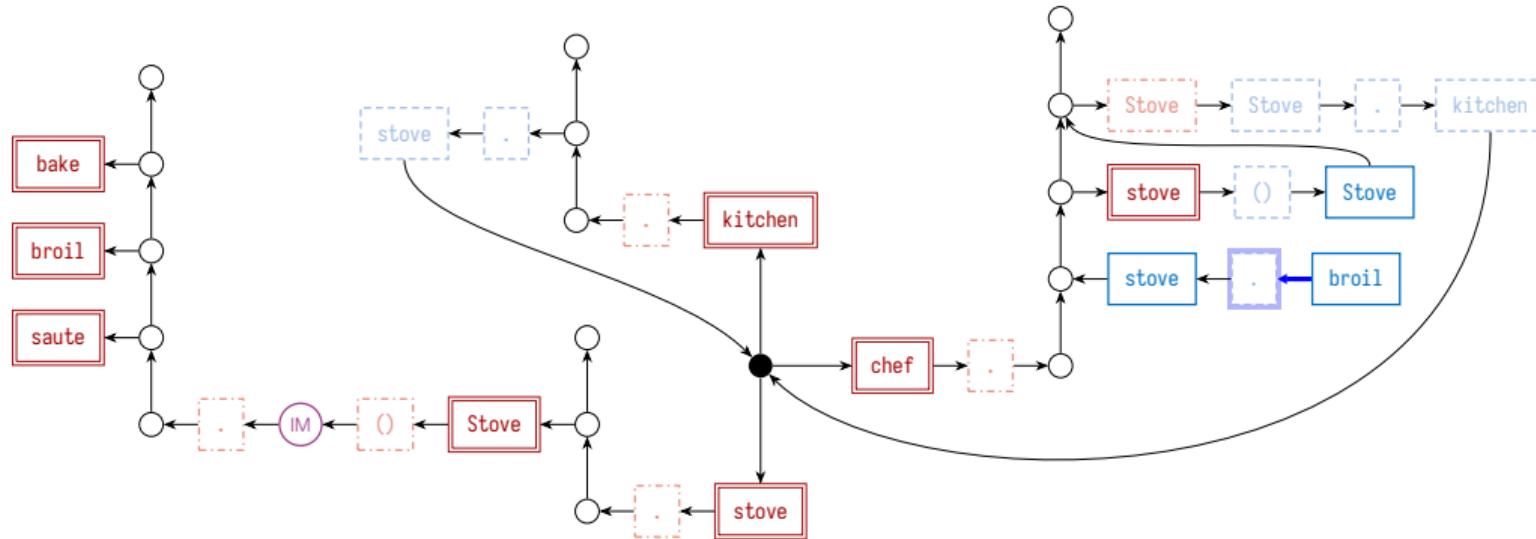
Symbol stack:  $\langle \rangle$

# The more complex example



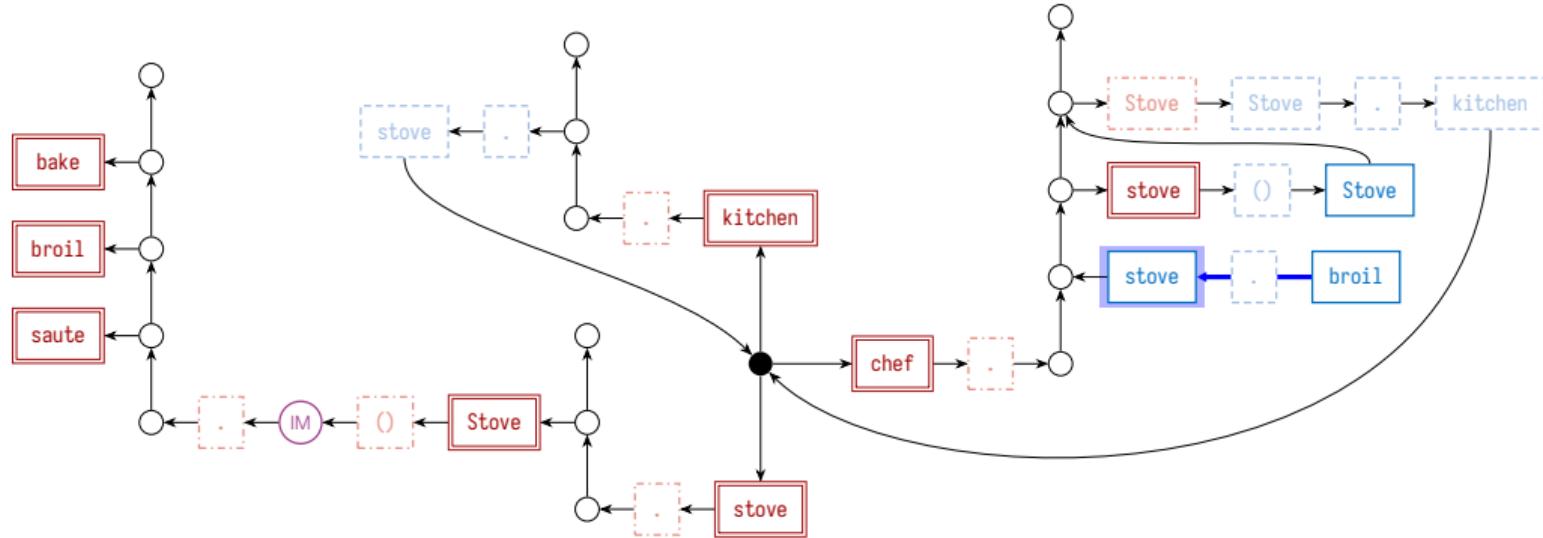
Symbol stack:  $\langle \text{broil} \rangle$

# The more complex example



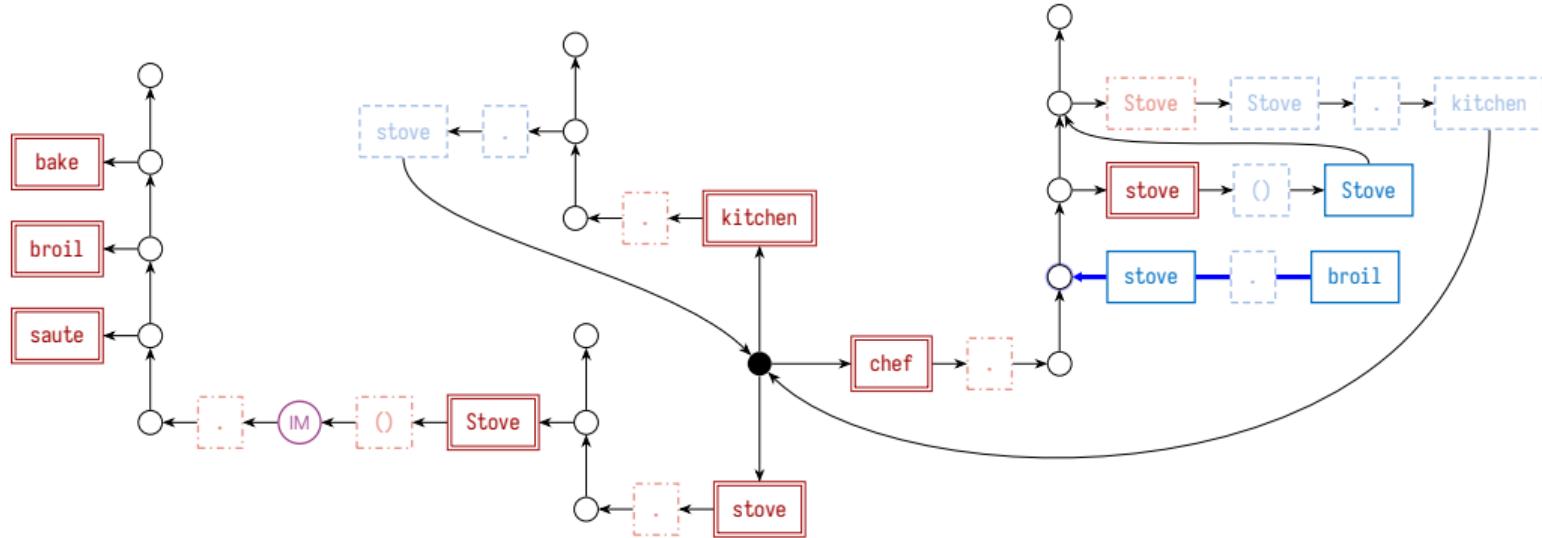
Symbol stack: `<.broil>`

# The more complex example



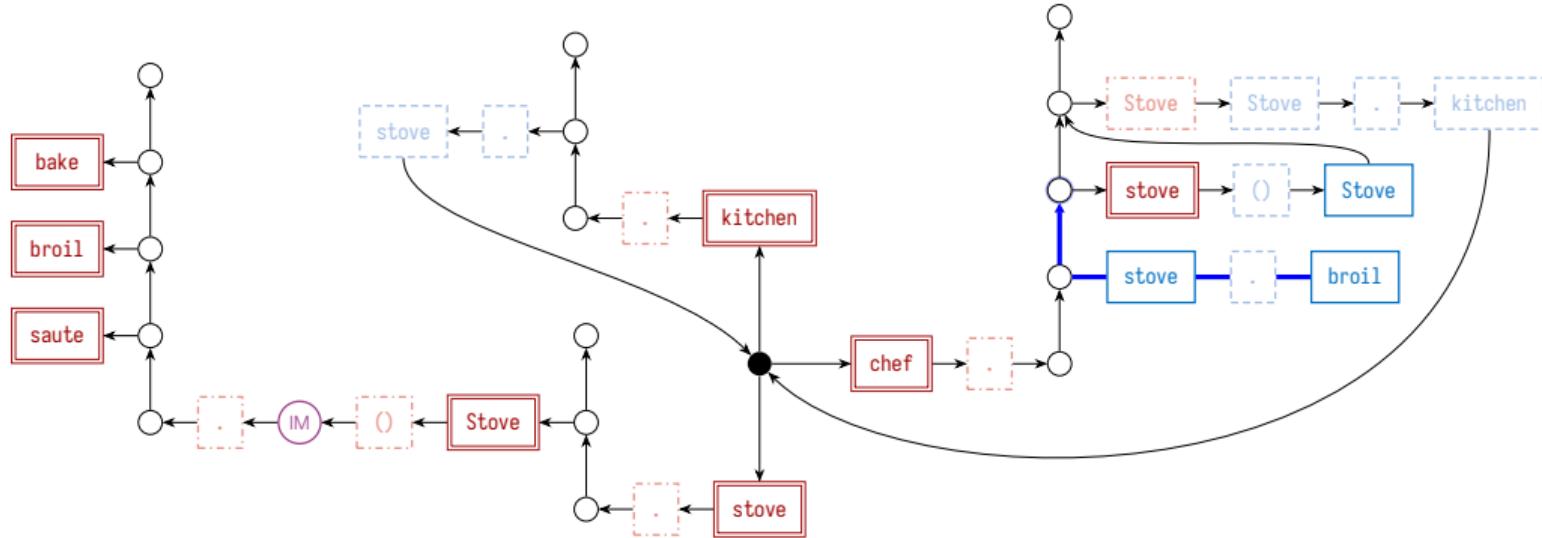
Symbol stack:  $\langle \text{stove}.\text{broil} \rangle$

# The more complex example



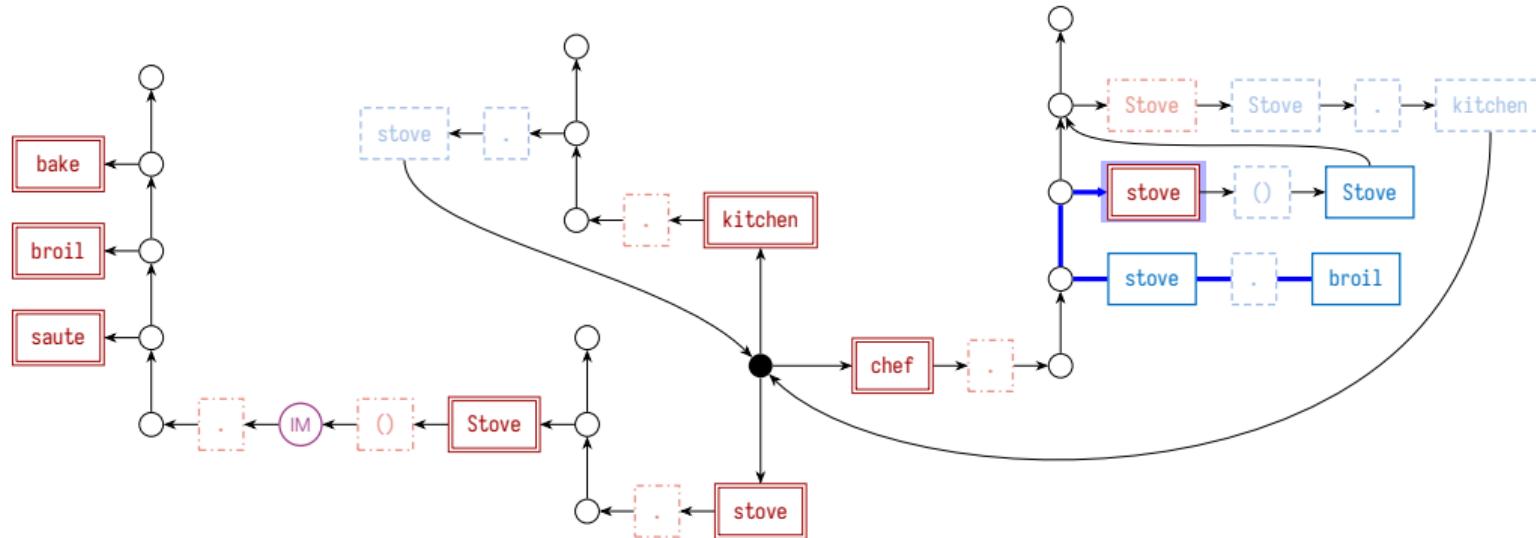
Symbol stack:  $\langle \text{stove}.\text{broil} \rangle$

# The more complex example



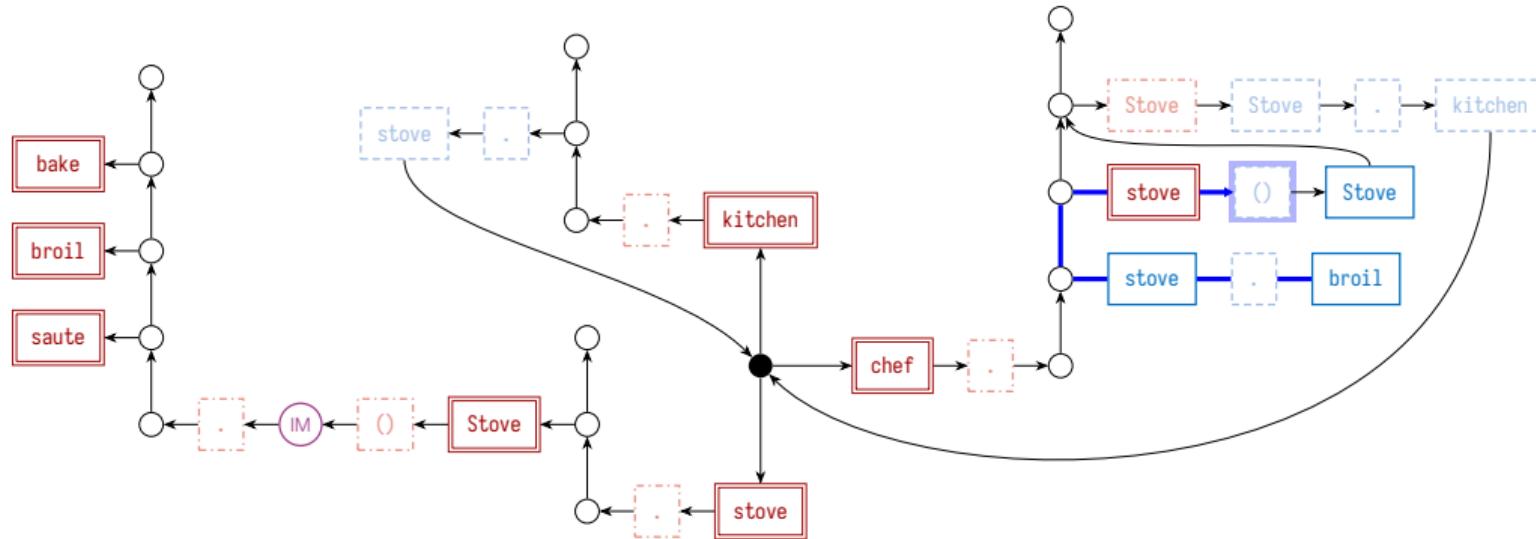
Symbol stack:  $\langle \text{stove.broil} \rangle$

# The more complex example



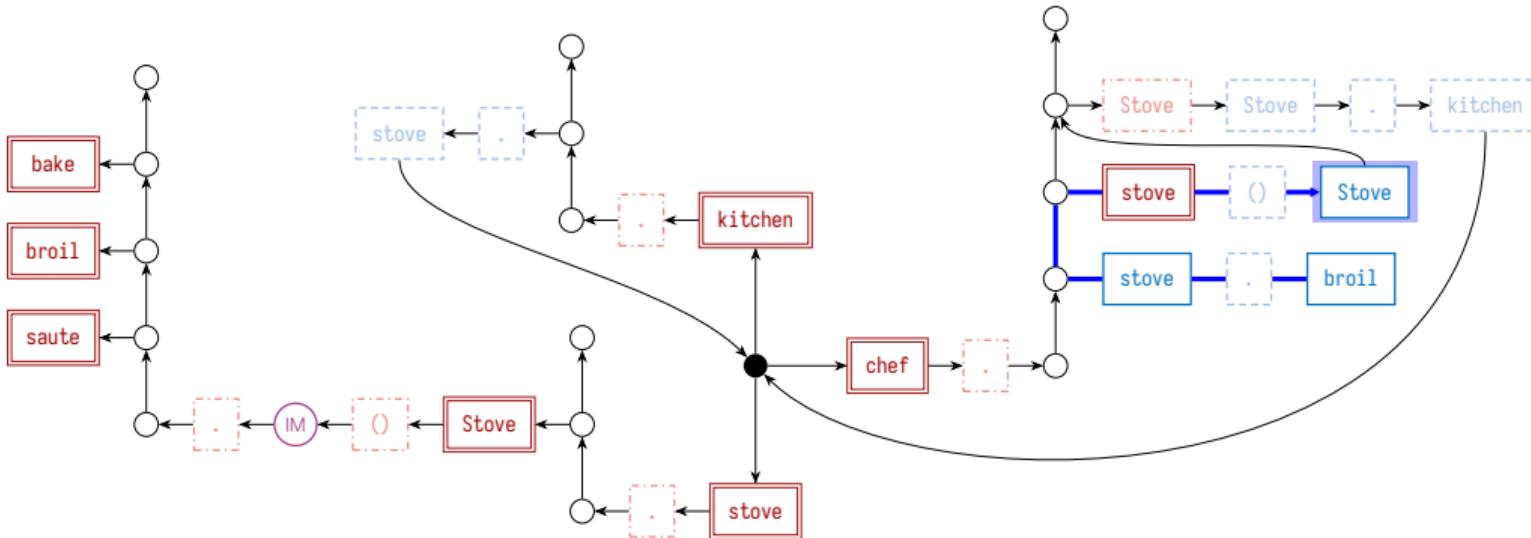
Symbol stack: `<.broil>`

# The more complex example



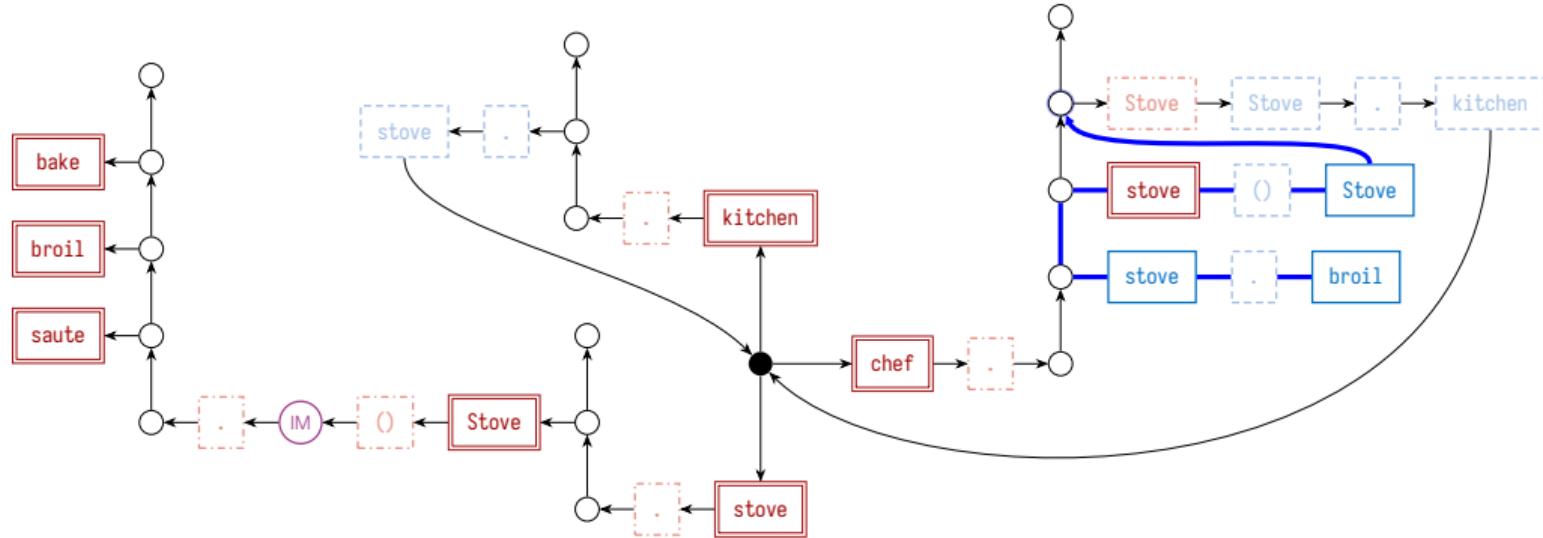
Symbol stack:  $\langle () . \text{broil} \rangle$

## The more complex example



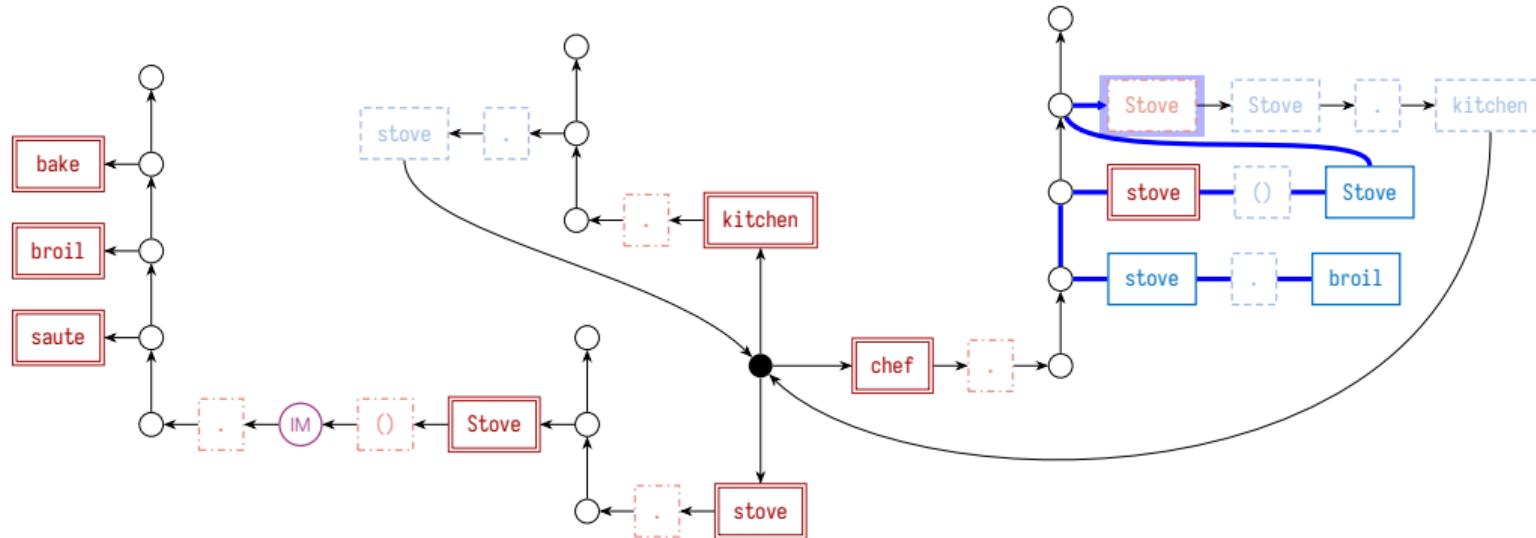
Symbol stack: <Stove().broil>

# The more complex example



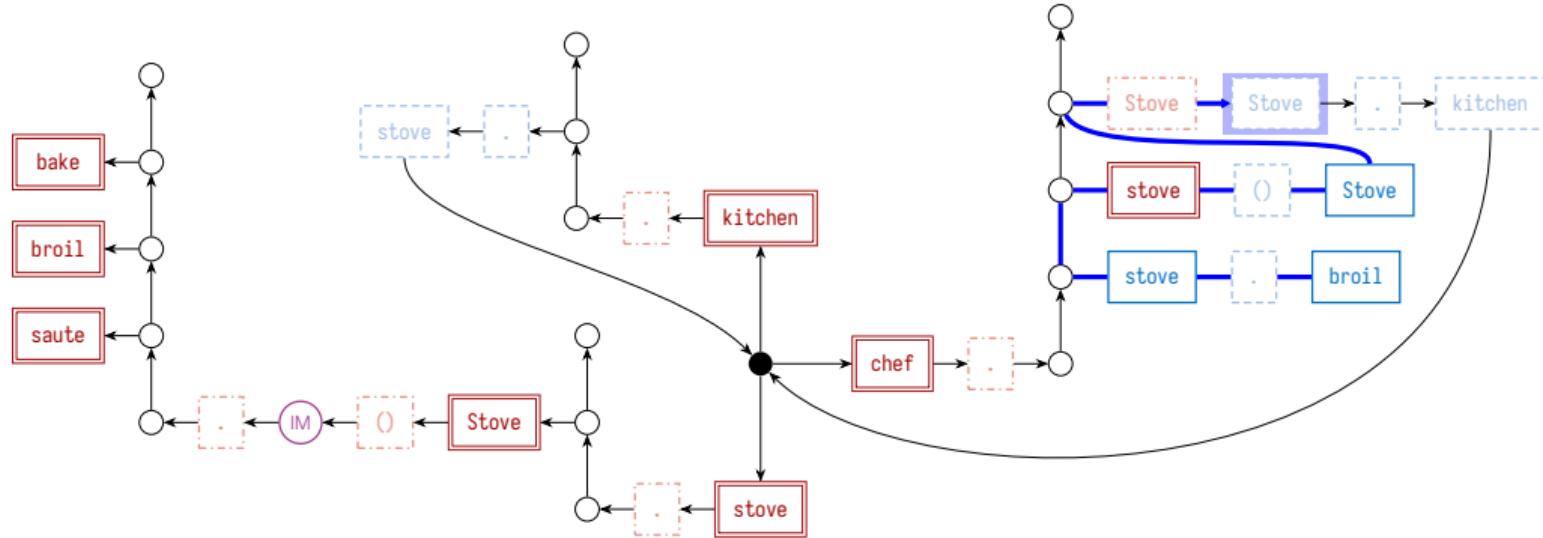
Symbol stack:  $\langle \text{Stove}().\text{broil} \rangle$

# The more complex example



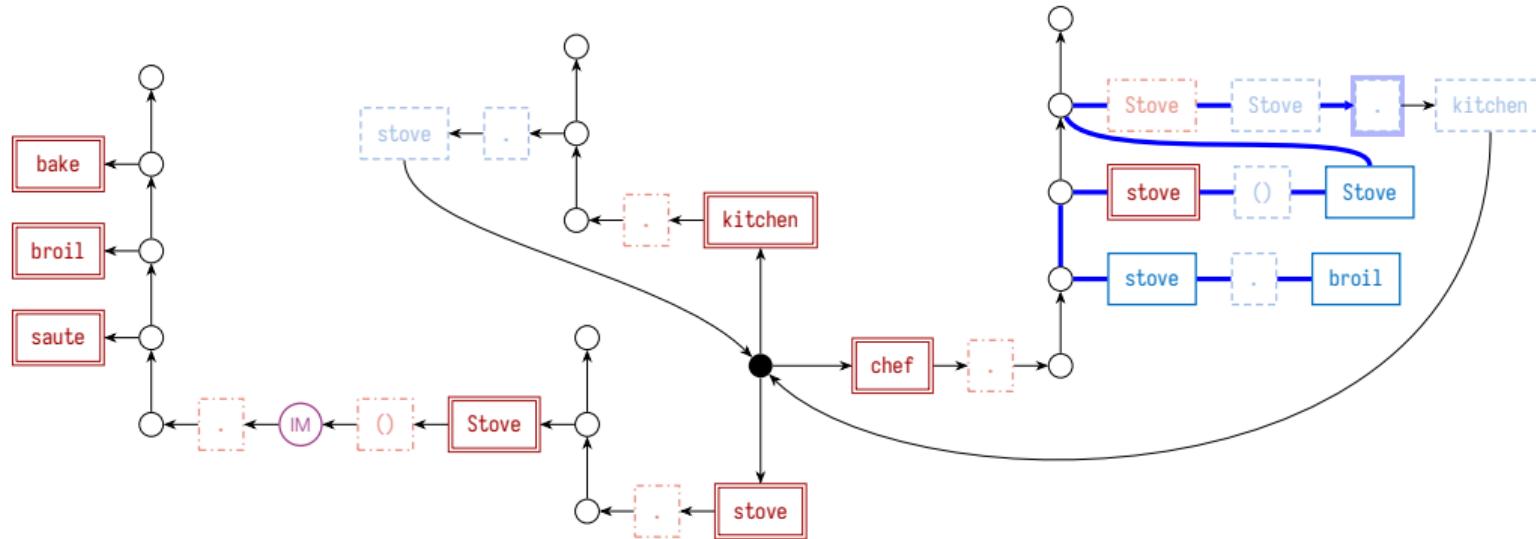
Symbol stack:  $\langle () . \text{broil} \rangle$

# The more complex example



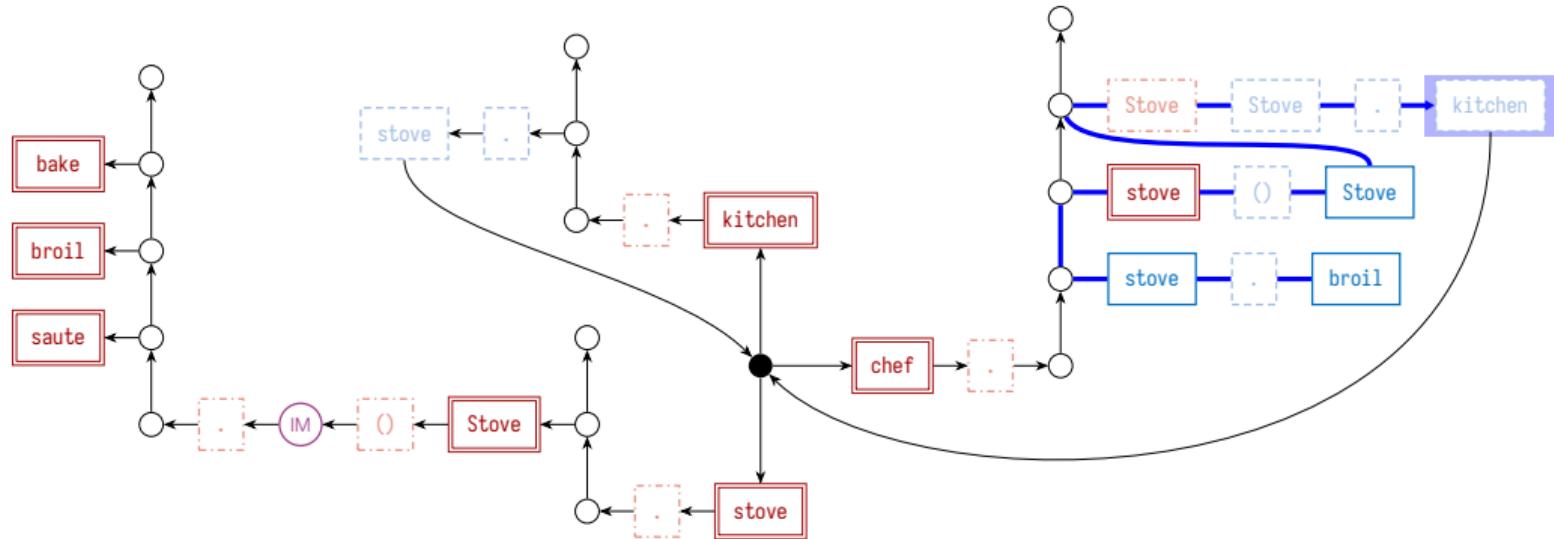
Symbol stack:  $\langle \text{Stove}().\text{broil} \rangle$

# The more complex example



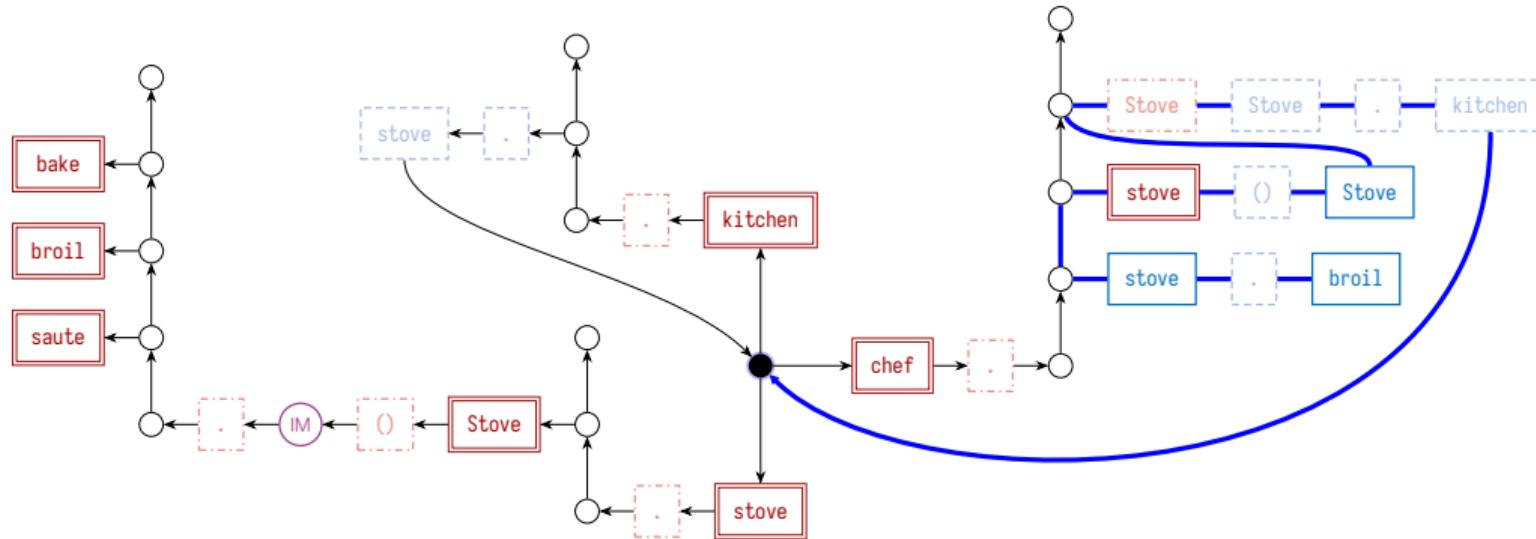
Symbol stack: `<.Stove().broil>`

# The more complex example



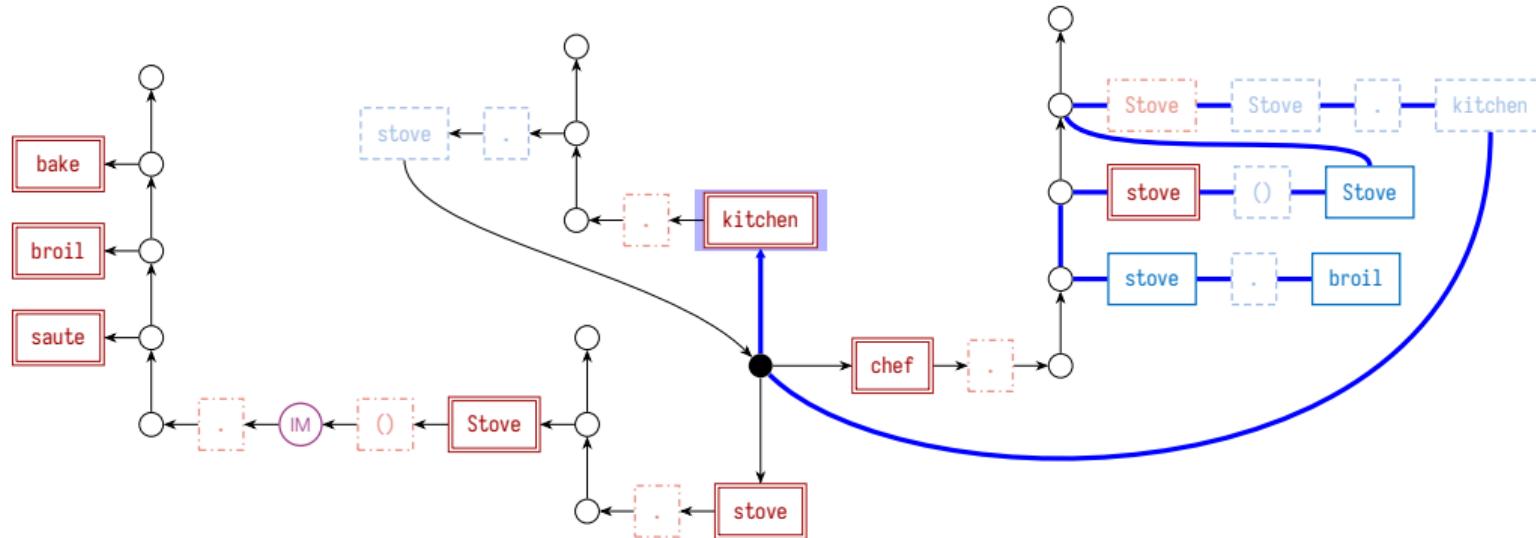
Symbol stack:  $\langle \text{kitchen}.\text{Stove}().\text{broil} \rangle$

# The more complex example



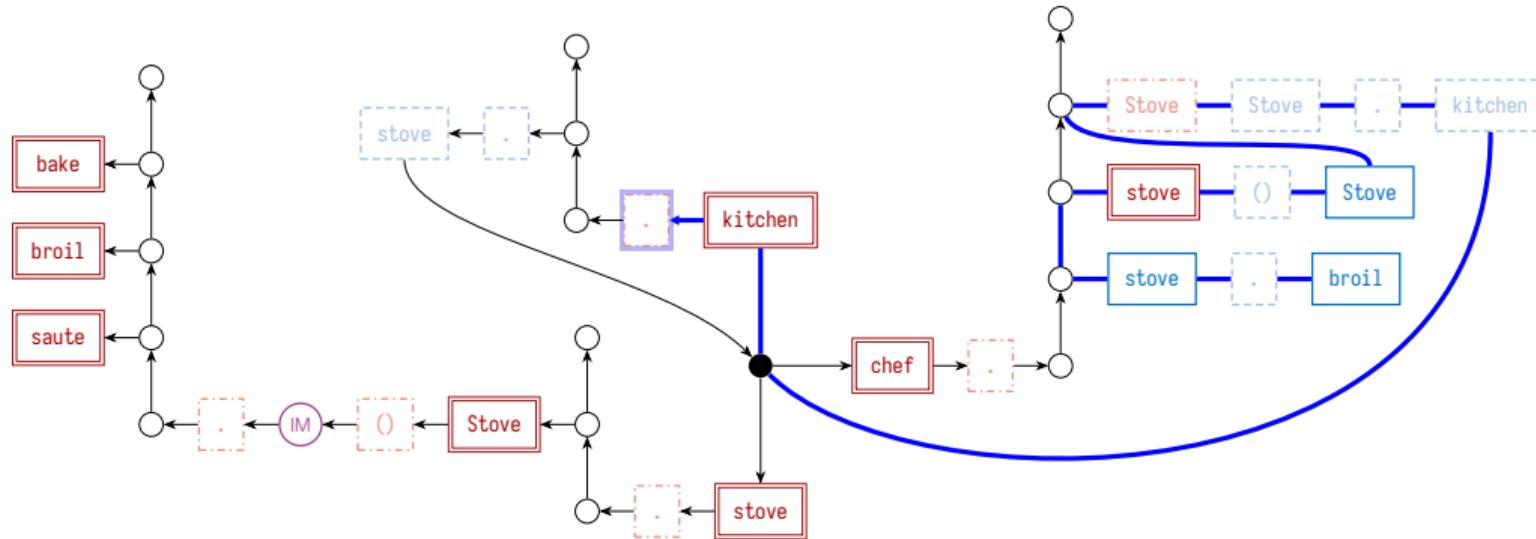
Symbol stack:  $\langle \text{kitchen}.\text{Stove}().\text{broil} \rangle$

# The more complex example



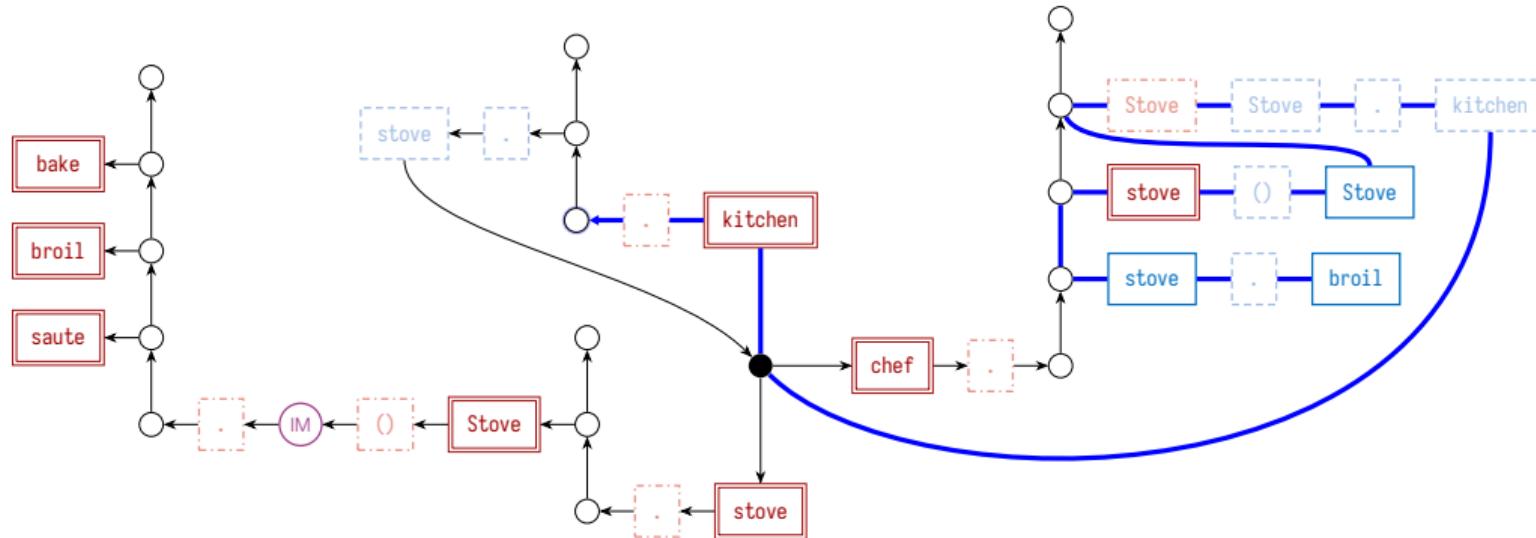
Symbol stack: `<.Stove().broil>`

# The more complex example



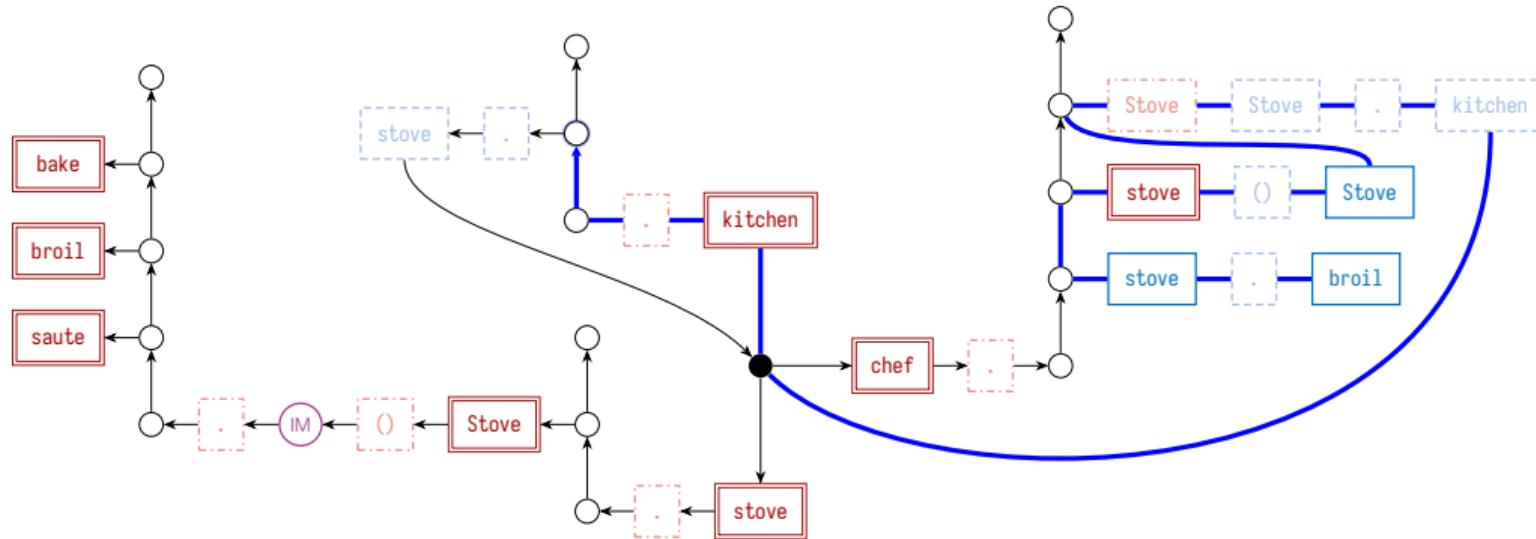
Symbol stack: `⟨Stove().broil⟩`

# The more complex example



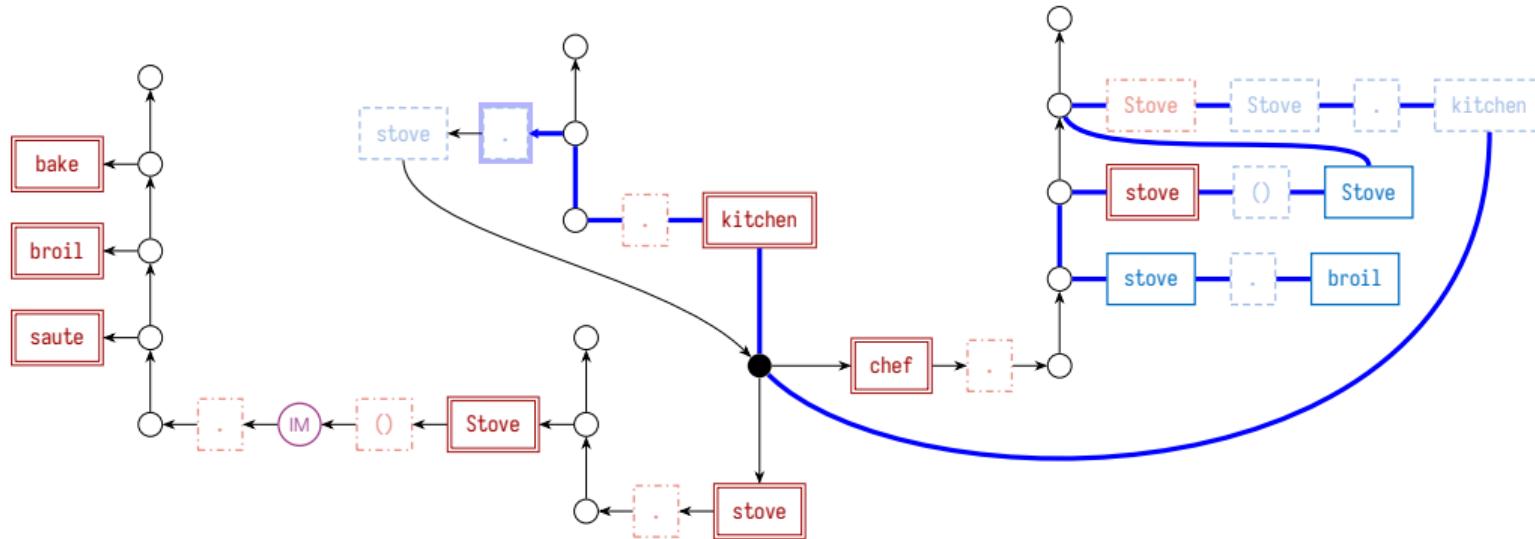
Symbol stack:  $\langle \text{Stove}().\text{broil} \rangle$

# The more complex example



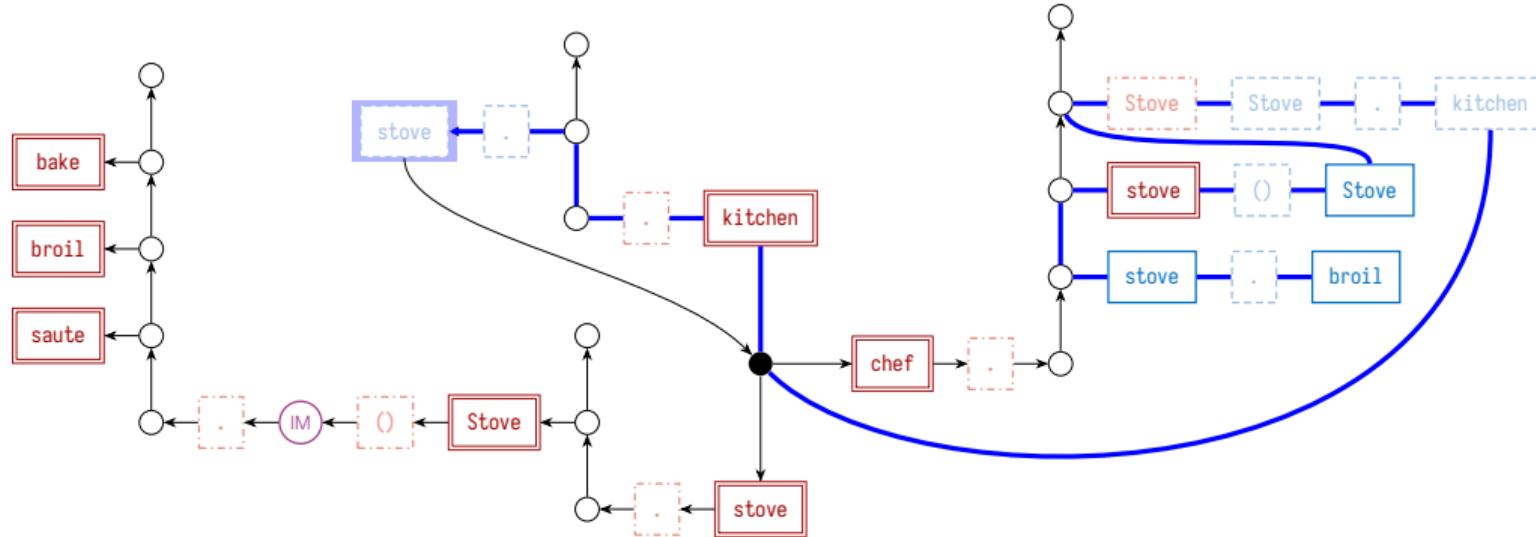
Symbol stack:  $\langle \text{Stove}().\text{broil} \rangle$

## The more complex example



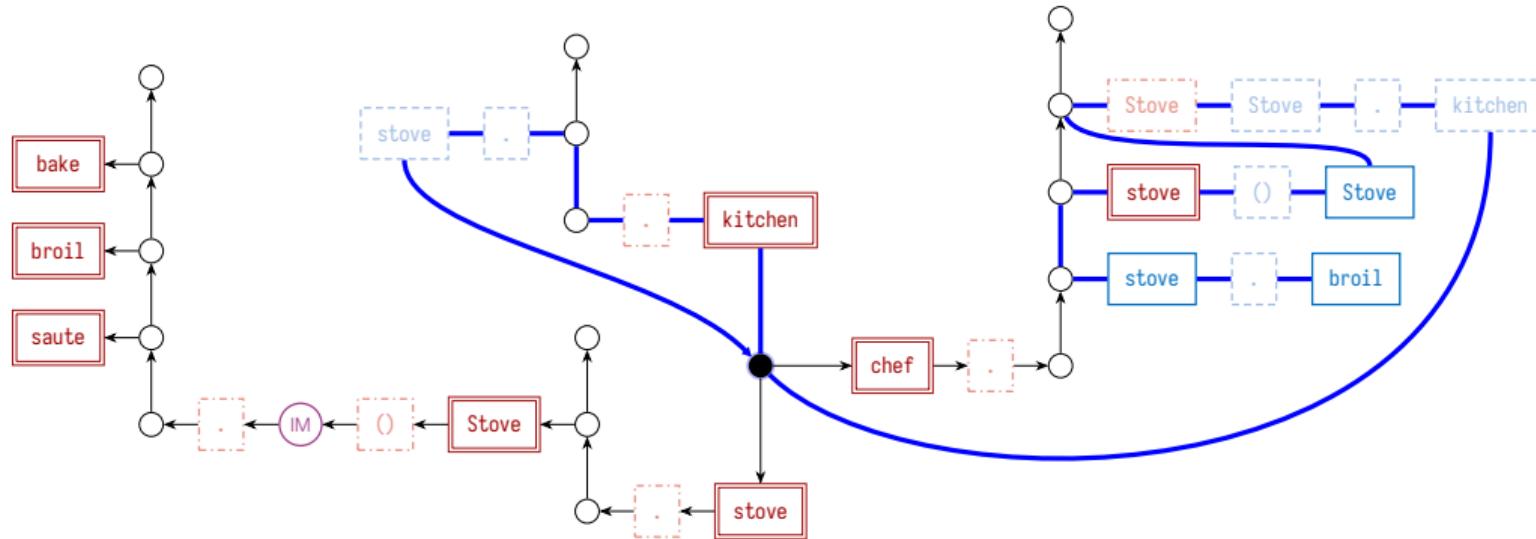
Symbol stack: <.Stove().broil>

# The more complex example



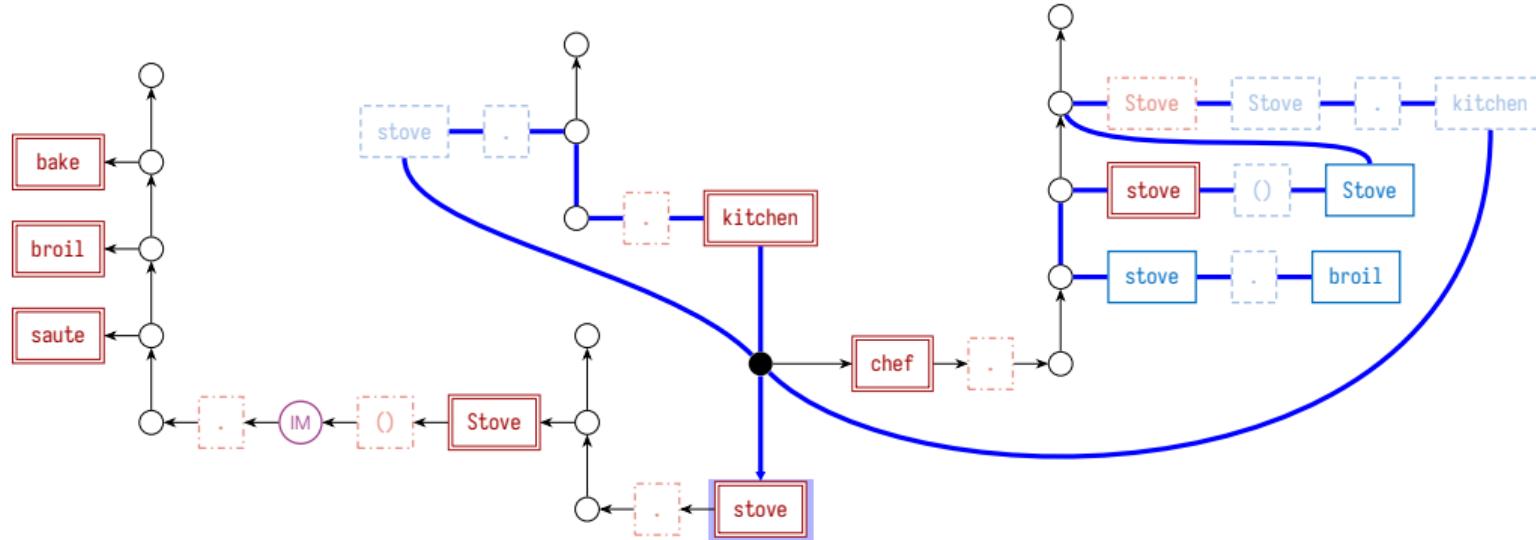
Symbol stack: `<stove.Stove().broil>`

# The more complex example



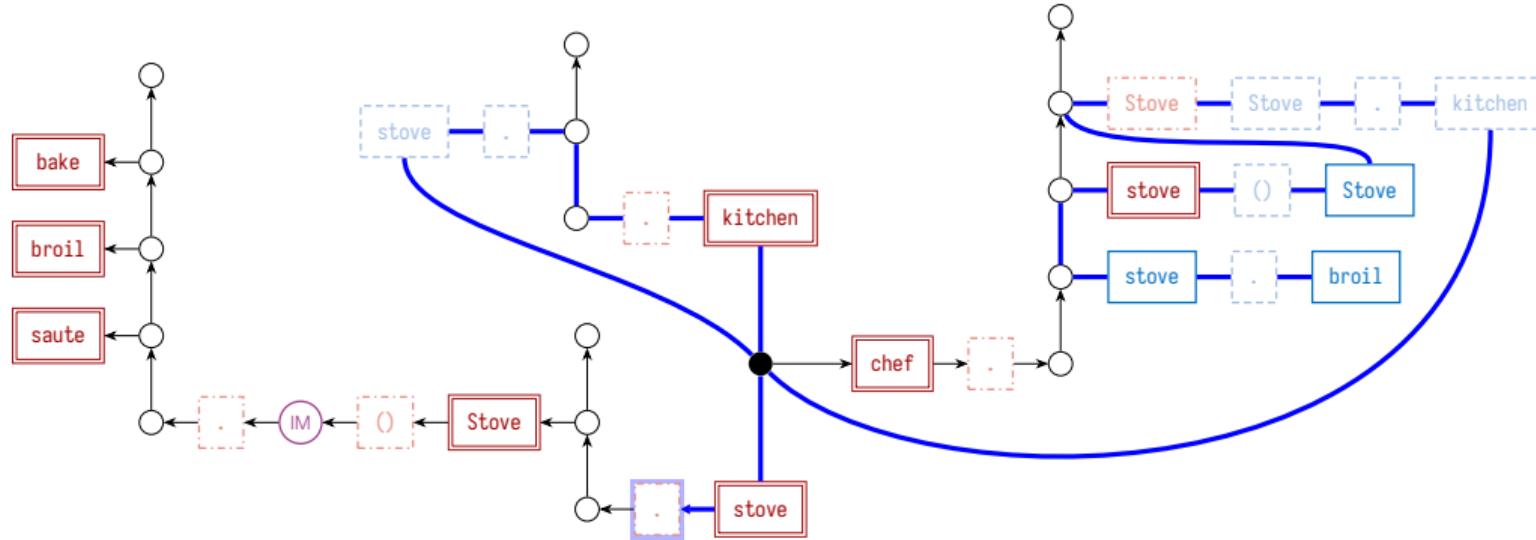
Symbol stack: `<stove.Stove().broil>`

# The more complex example



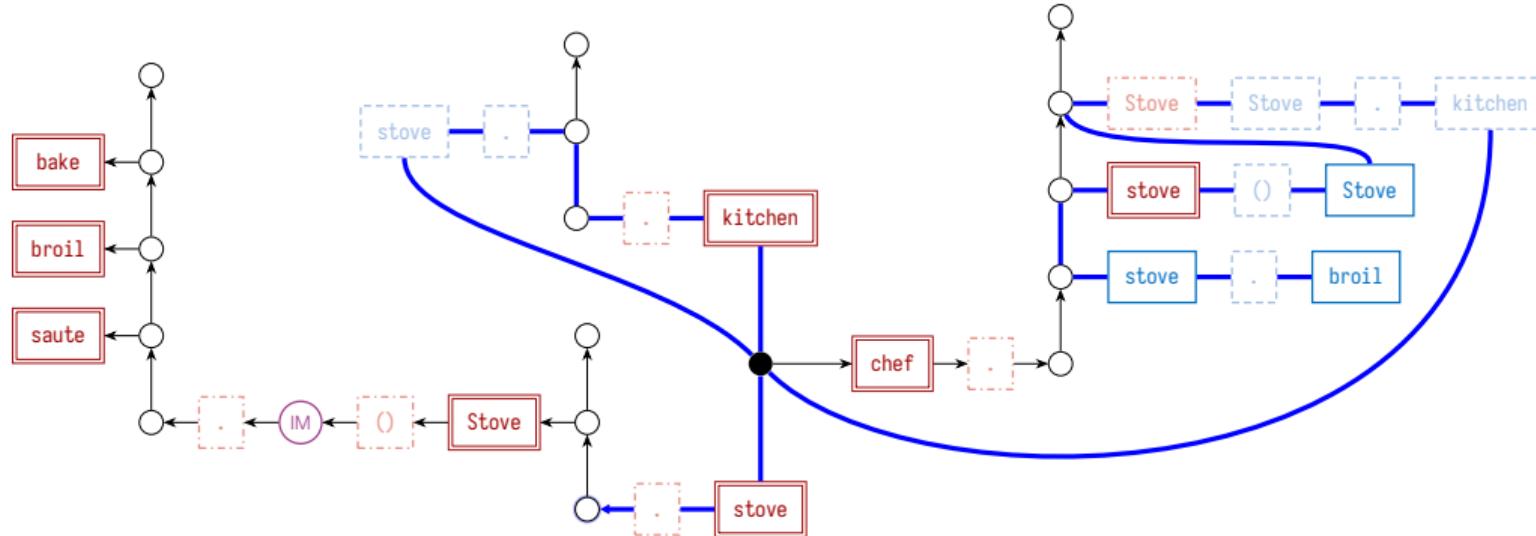
Symbol stack: `<.Stove().broil>`

# The more complex example



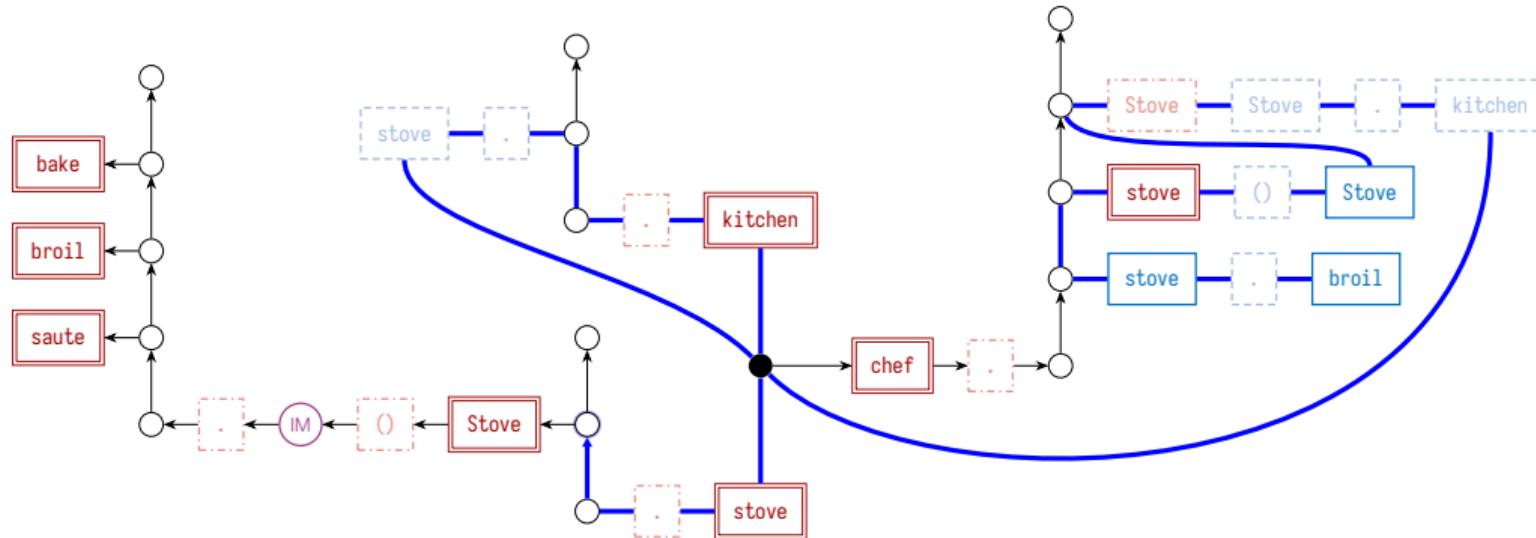
Symbol stack:  $\langle \text{Stove}().\text{broil} \rangle$

# The more complex example



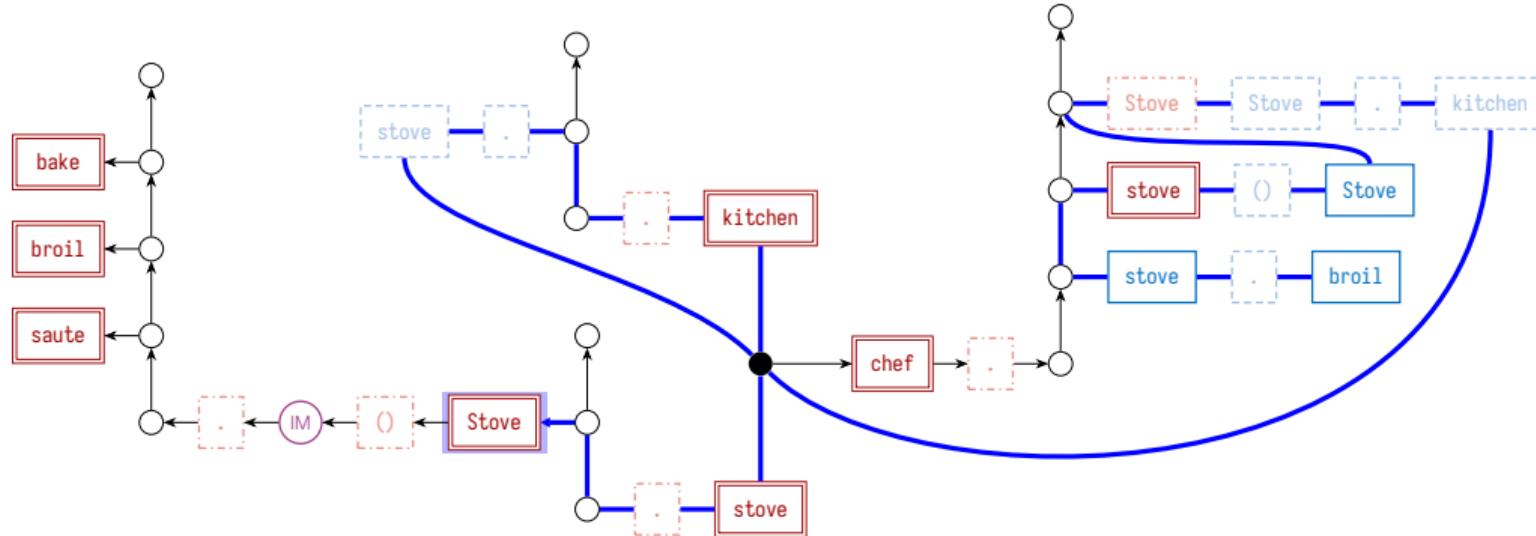
Symbol stack:  $\langle \text{Stove}().\text{broil} \rangle$

# The more complex example



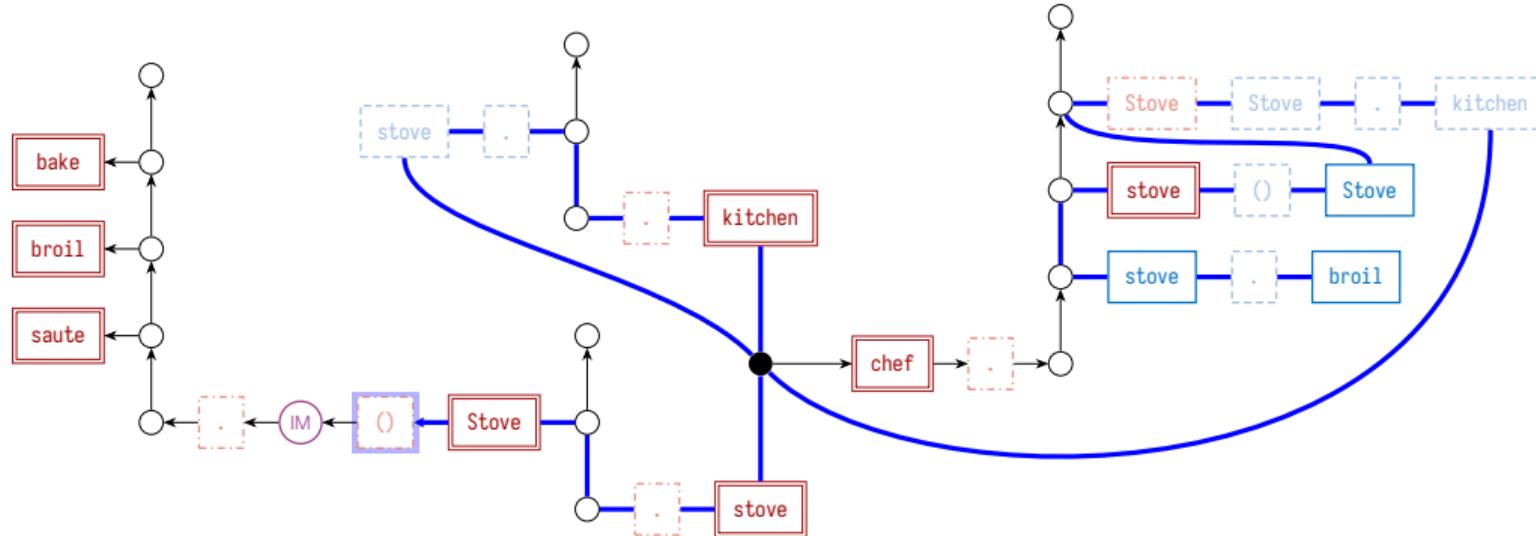
Symbol stack:  $\langle \text{Stove}().\text{broil} \rangle$

# The more complex example



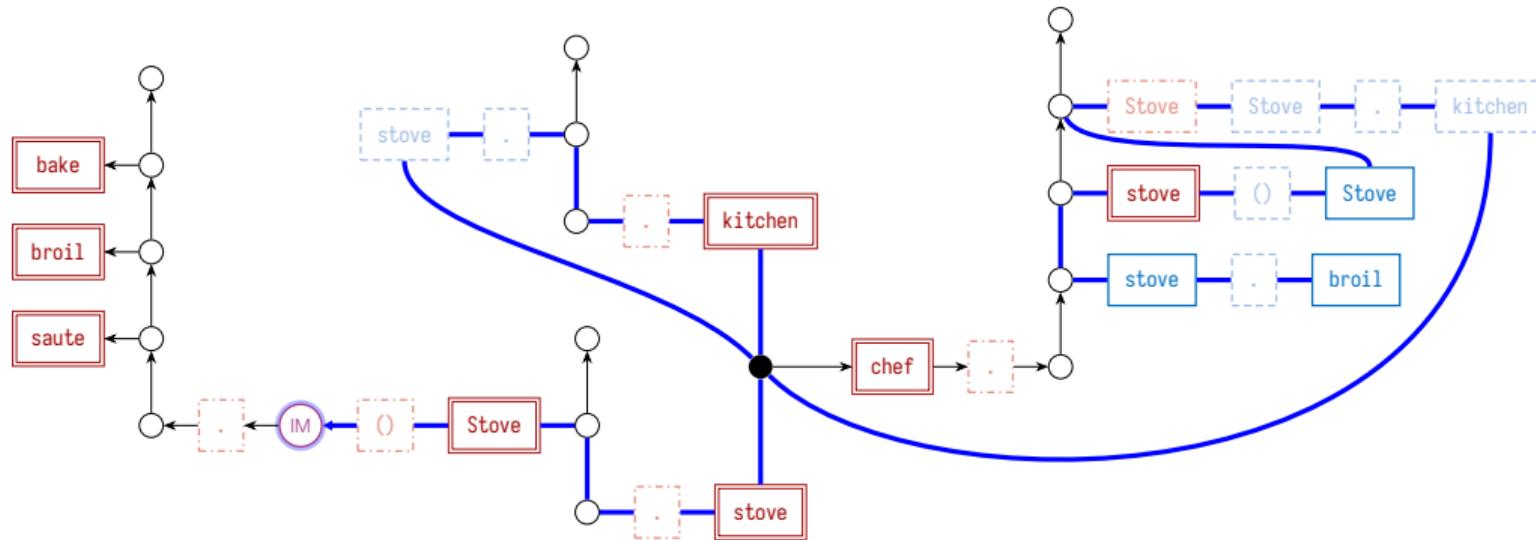
Symbol stack:  $\langle () . \text{broil} \rangle$

# The more complex example



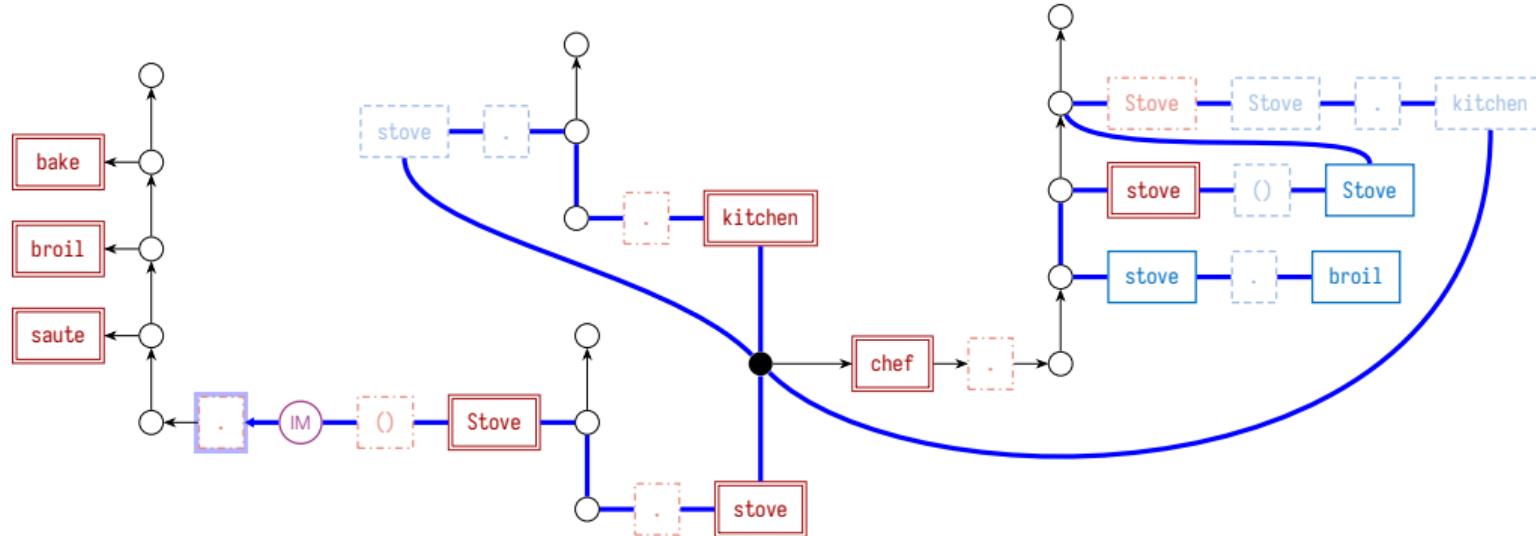
Symbol stack: `<.broil>`

## The more complex example



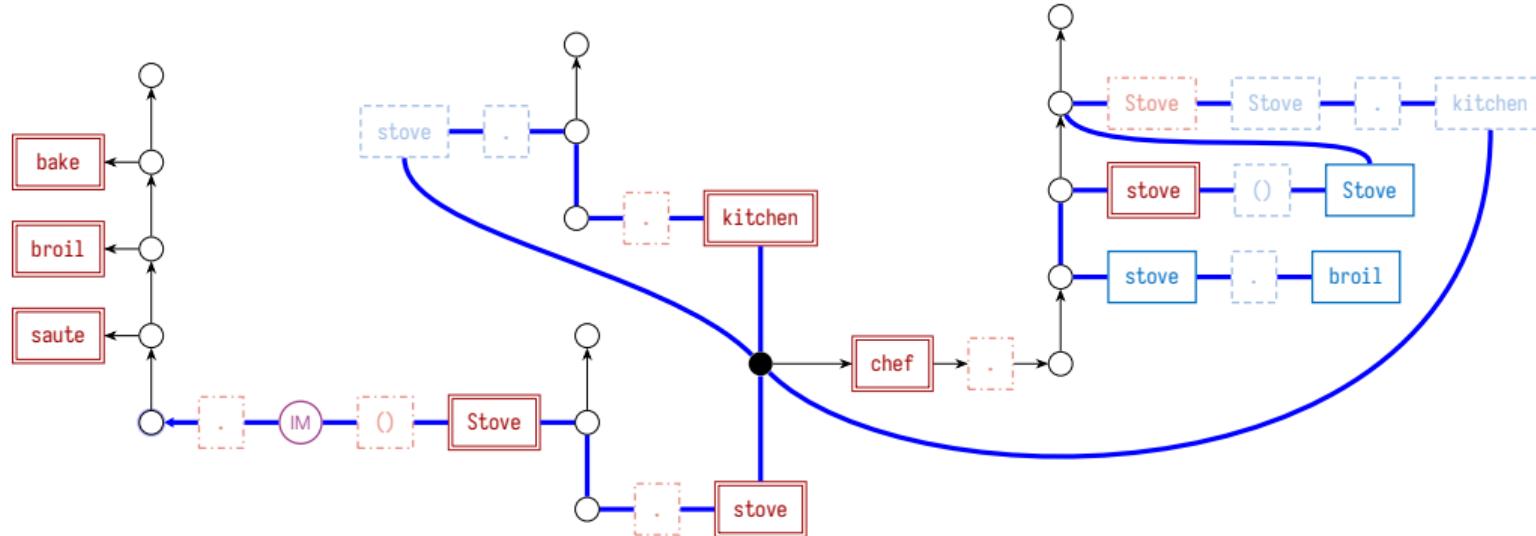
Symbol stack: <.broil>

# The more complex example



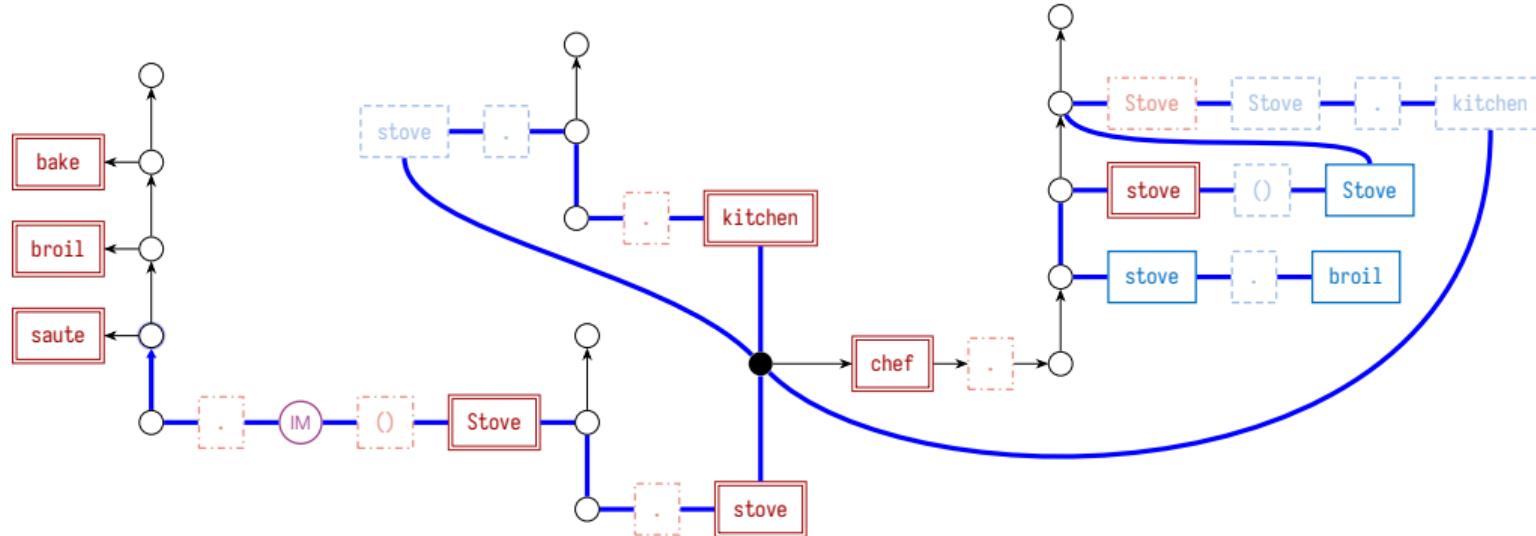
Symbol stack:  $\langle \text{broil} \rangle$

# The more complex example

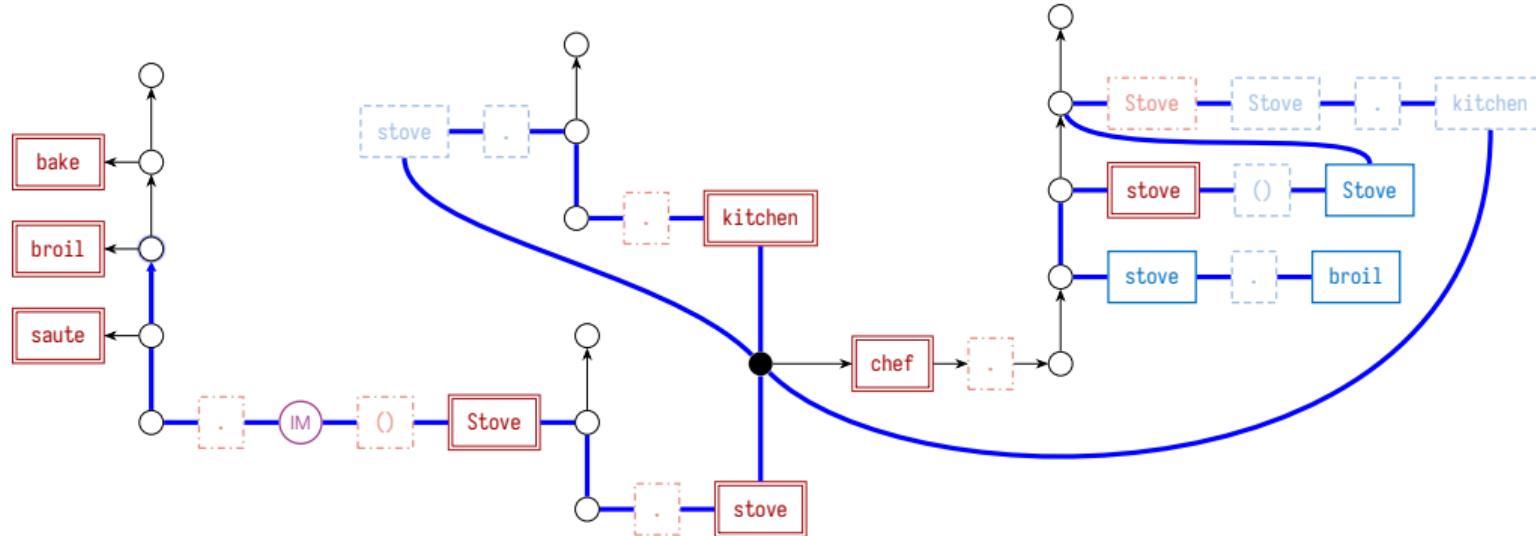


Symbol stack:  $\langle \text{broil} \rangle$

# The more complex example

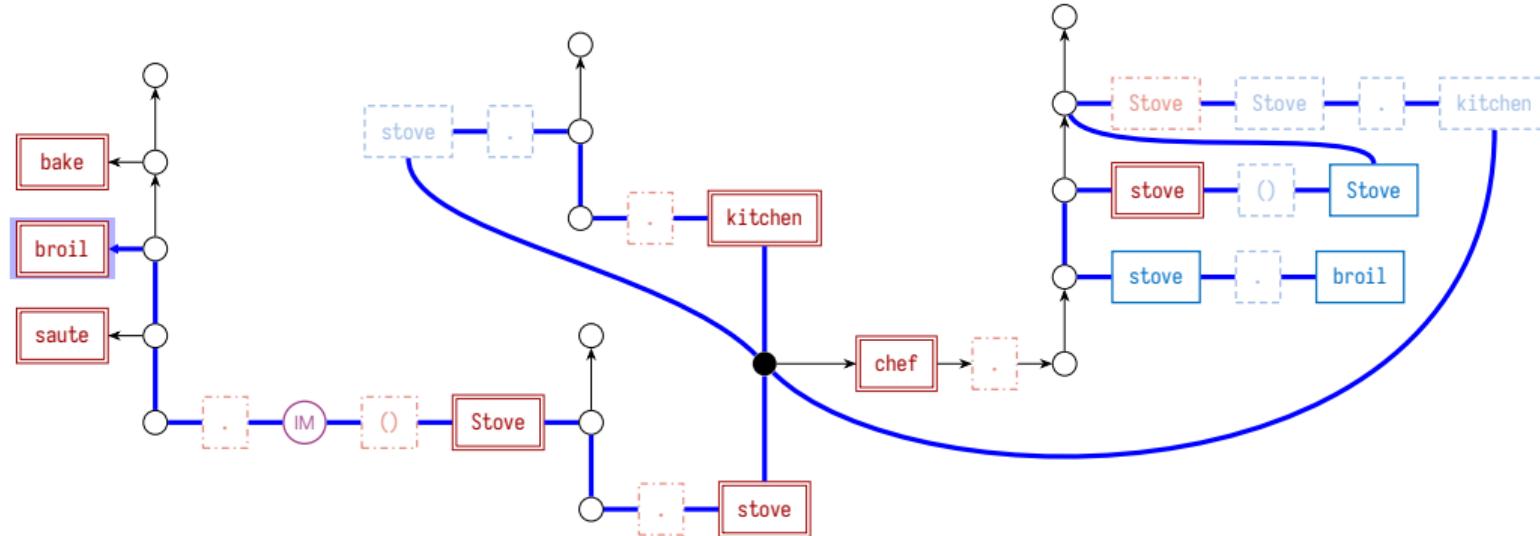


# The more complex example



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

# The more complex example



Symbol stack:  $\langle \rangle$

# Are we done?

Index

Query

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

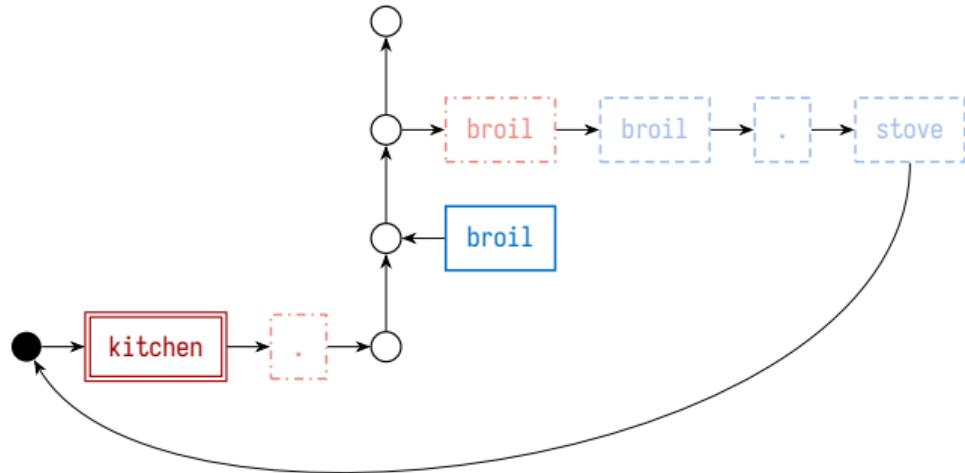
Can we shift more of the work to index time,  
while still remaining incremental?

**Partial paths**



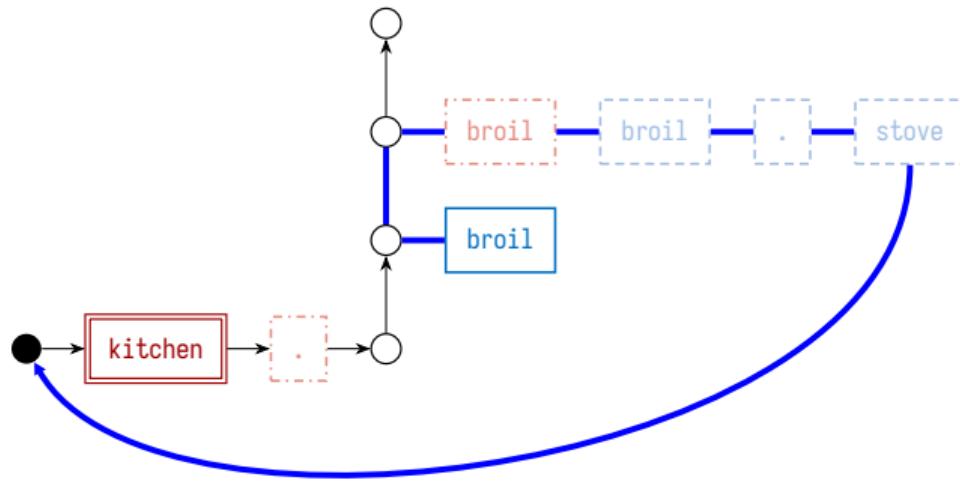
# Partial paths

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



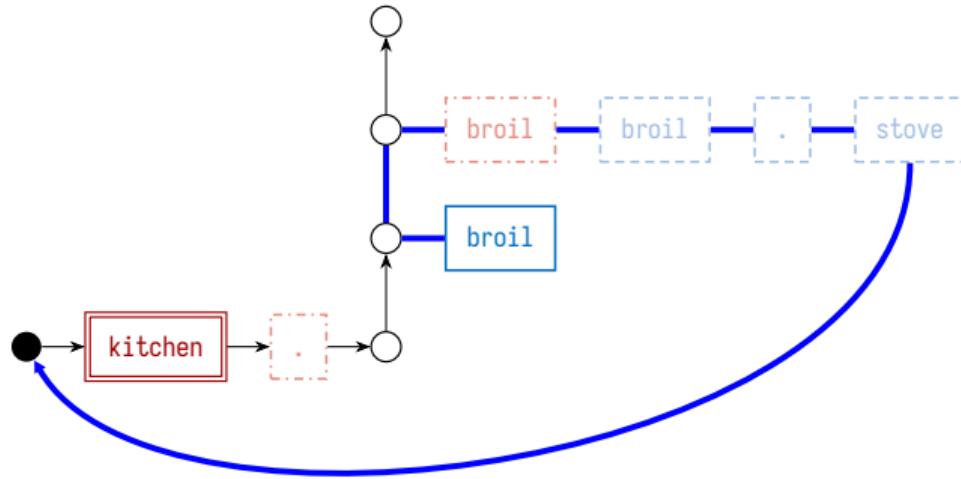
# Partial paths

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



# Partial paths

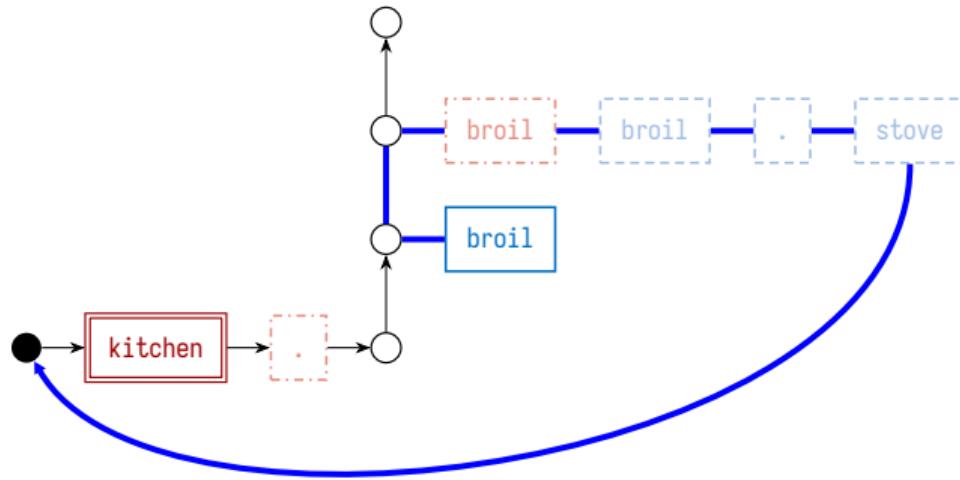
```
kitchen.py  
from stove import broil  
  
broil()
```



$\langle \rangle$  broil  $\rightsquigarrow$  ●  $\langle \text{stove.broil} \rangle$

# Partial paths

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



$\langle \rangle$  broil  $\rightsquigarrow$  ●  $\langle \text{stove.broil} \rangle$

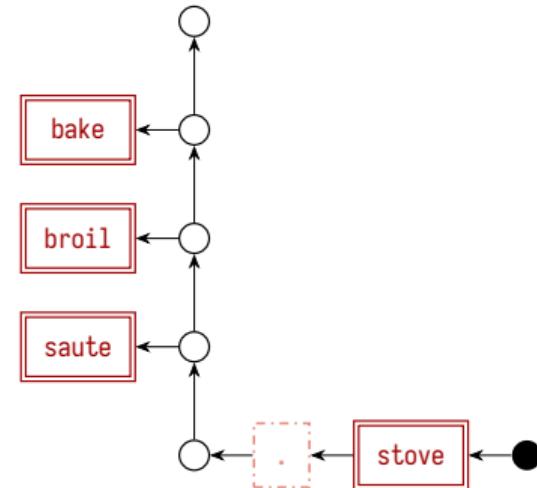
The reference at `kitchen.py:3:1` refers to `stove.broil` in some other file

# Partial paths

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

def broil():
    pass

def saute():
    pass
```

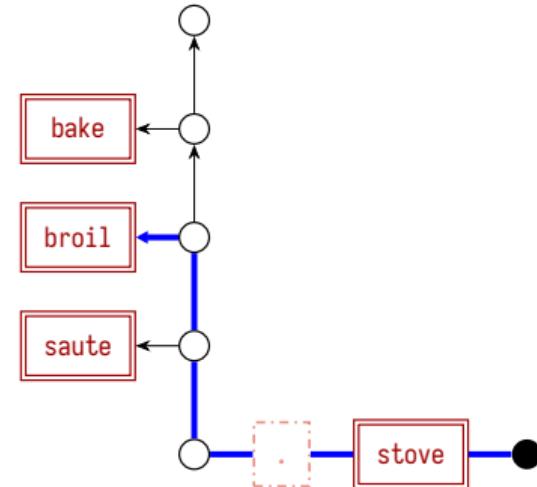


# Partial paths

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

def broil():
    pass

def saute():
    pass
```

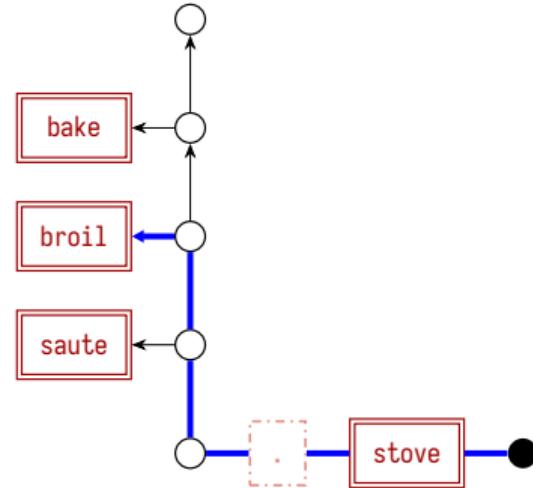


# Partial paths

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

def broil():
    pass

def saute():
    pass
```



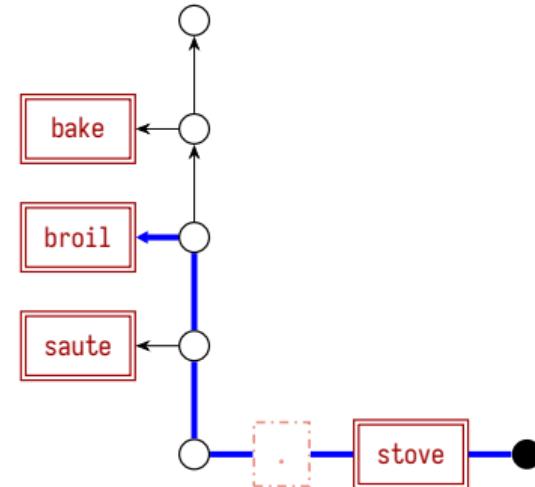
$\langle \text{stove.broil} \rangle$  ●  $\rightsquigarrow$   $\langle \rangle$

# Partial paths

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

def broil():
    pass

def saute():
    pass
```



`<stove.broil>` ●  $\rightsquigarrow$  `broil` ⟨ ⟩

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

# Concatenating partial paths

```
<> broil ~> ● <stove.broil> + <stove.broil> ● ~> broil <>
```

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

# Concatenating partial paths



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

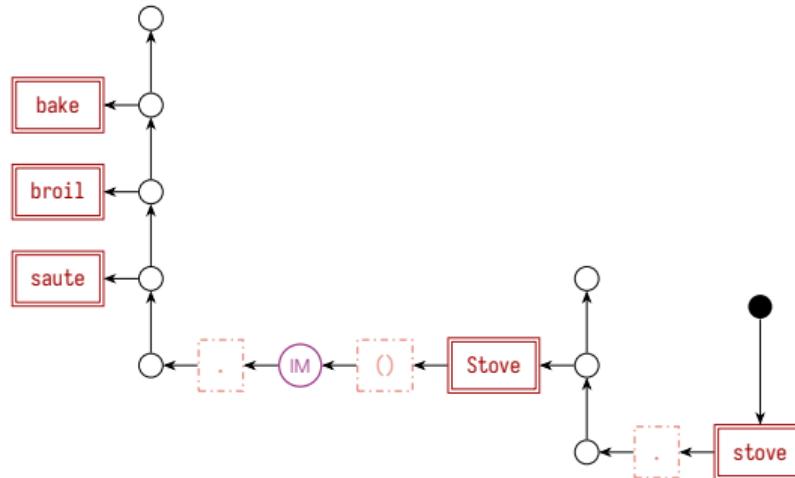
# The more complex example

stove.py

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

    def broil(self):
        pass

    def saute(self):
        pass
```



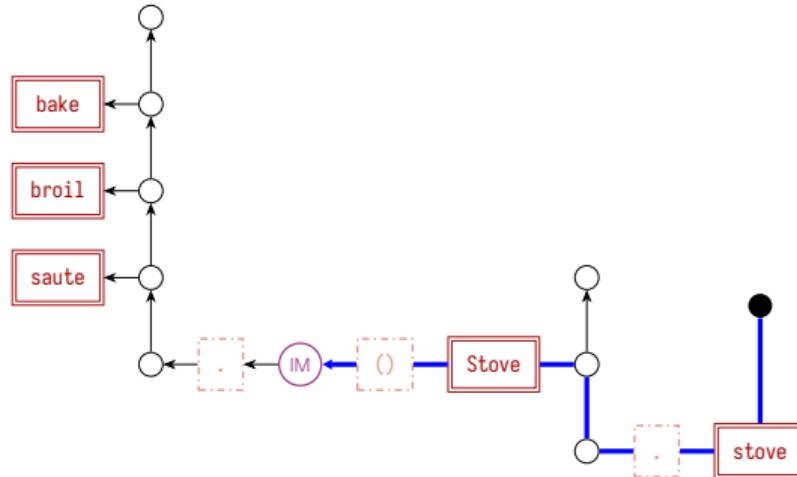
# The more complex example

stove.py

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

    def broil(self):
        pass

    def saute(self):
        pass
```



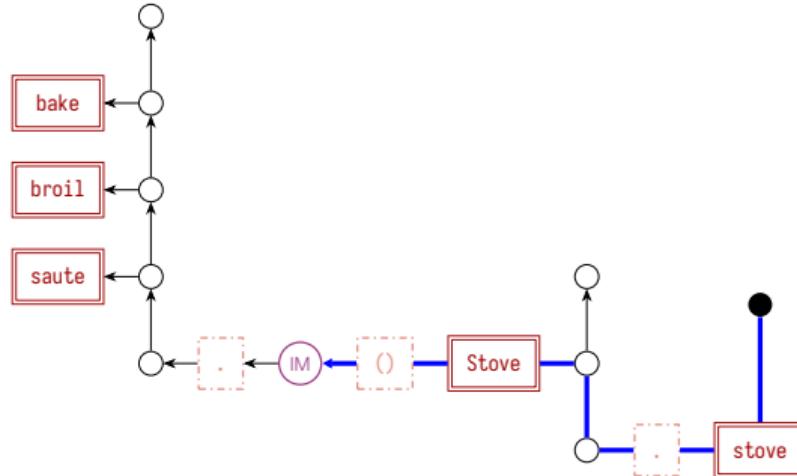
# The more complex example

stove.py

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

    def broil(self):
        pass

    def saute(self):
        pass
```



`<stove.Stove()>` ●  $\leadsto$  `IM` ⟨ ⟩

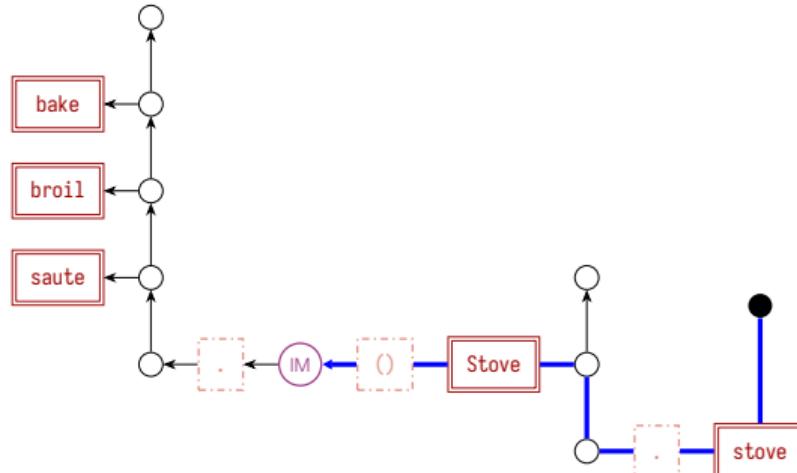
# The more complex example

stove.py

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

    def broil(self):
        pass

    def saute(self):
        pass
```



`<stove.Stove()>` ●  $\leadsto$  `IM` `<>`

Invoking `stove.Stove`  
gives you an instance of the `Stove` class.

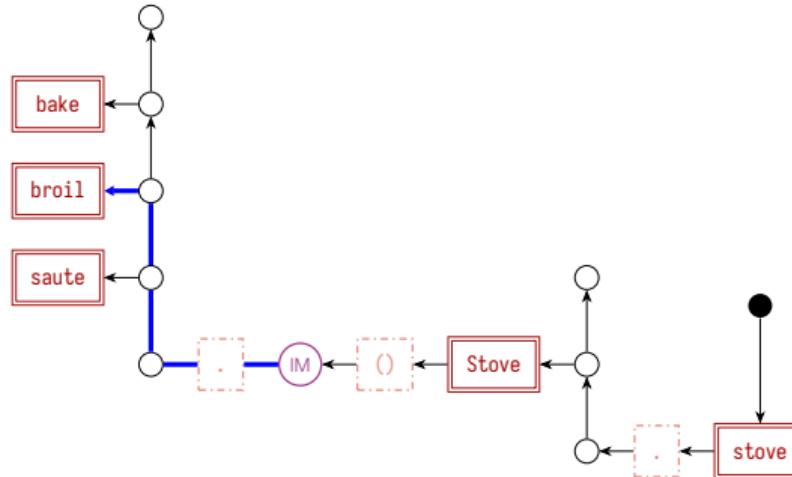
# The more complex example

stove.py

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

    def broil(self):
        pass

    def saute(self):
        pass
```



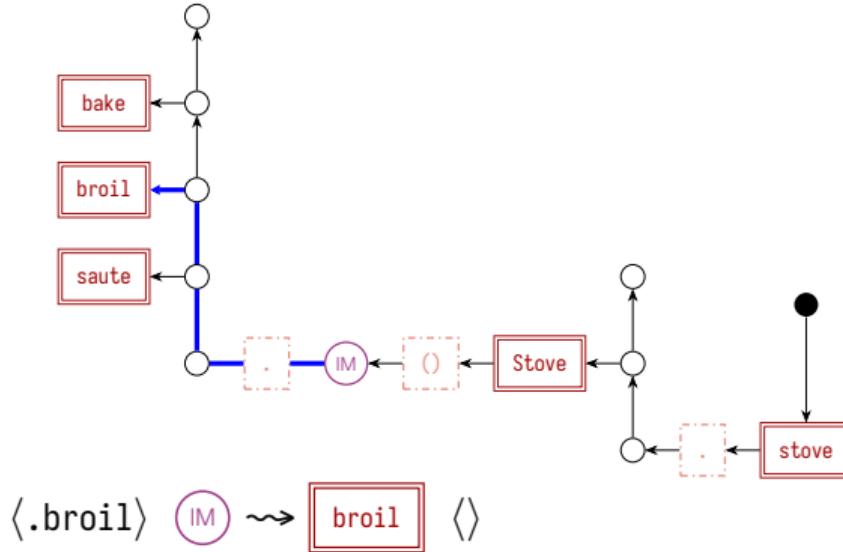
# The more complex example

stove.py

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

    def broil(self):
        pass

    def saute(self):
        pass
```



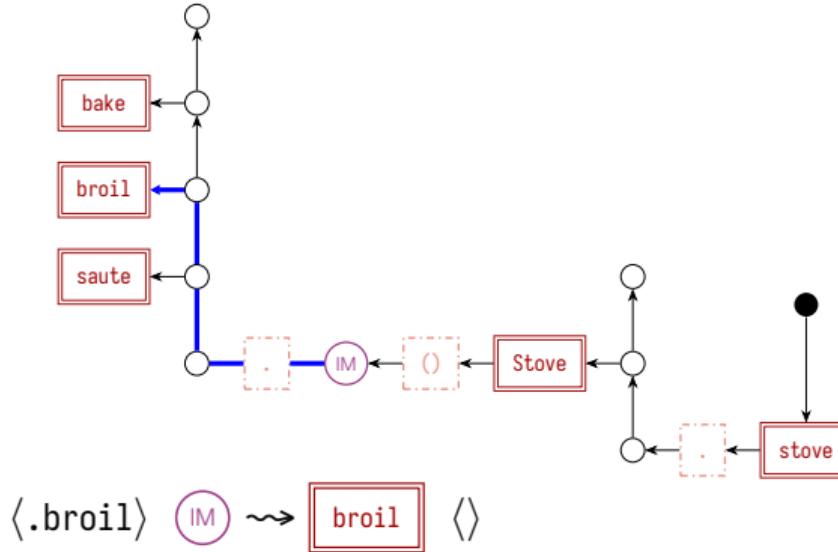
# The more complex example

stove.py

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

    def broil(self):
        pass

    def saute(self):
        pass
```

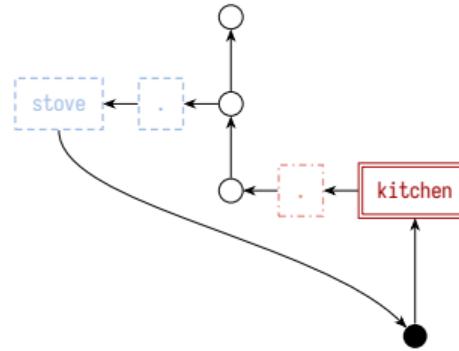


The `Stove` class

has an instance member named `broil`  
defined at `stove.py:5:9`.

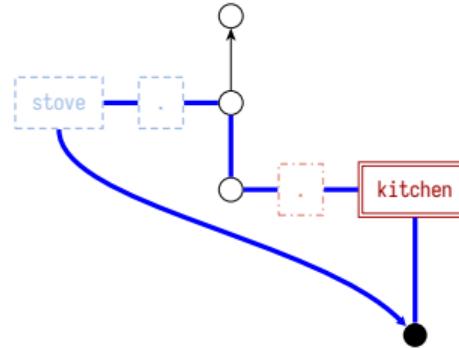
# The more complex example

```
kitchen.py  
from stove import *
```



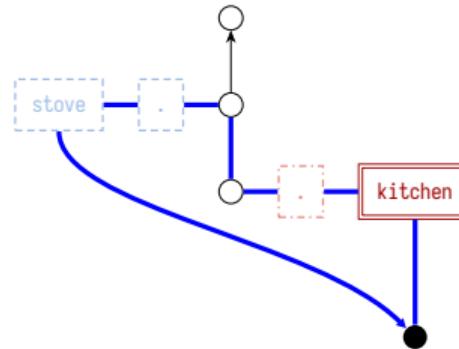
# The more complex example

```
kitchen.py  
from stove import *
```



# The more complex example

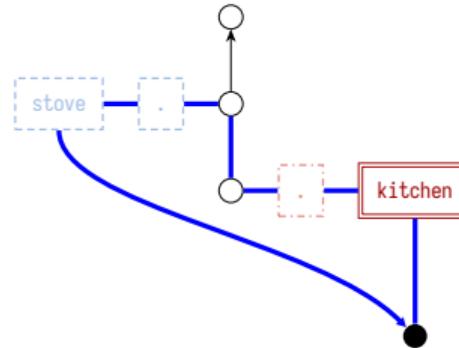
```
kitchen.py  
from stove import *
```



$\langle \text{kitchen.} \rangle \quad \bullet \rightsquigarrow \bullet \quad \langle \text{stove.} \rangle$

# The more complex example

```
kitchen.py  
from stove import *
```

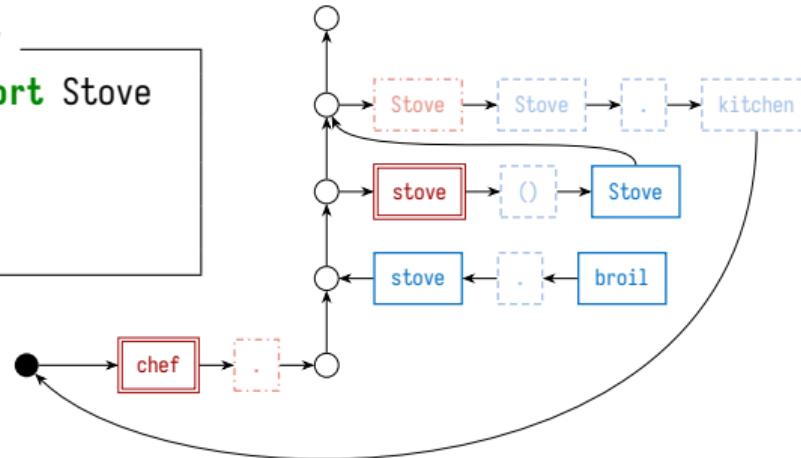


$\langle \text{kitchen.} \rangle \bullet \rightsquigarrow \bullet \langle \text{stove.} \rangle$

If you are looking for `kitchen.[anything]`  
then you might find it at `stove.[anything]`.

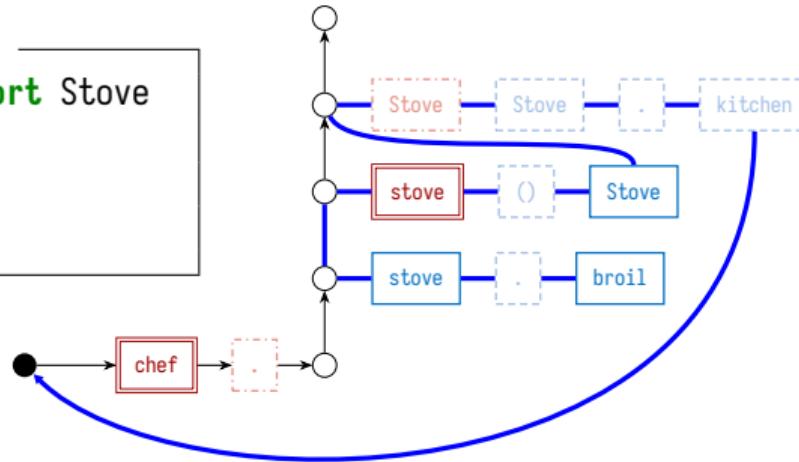
# The more complex example

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



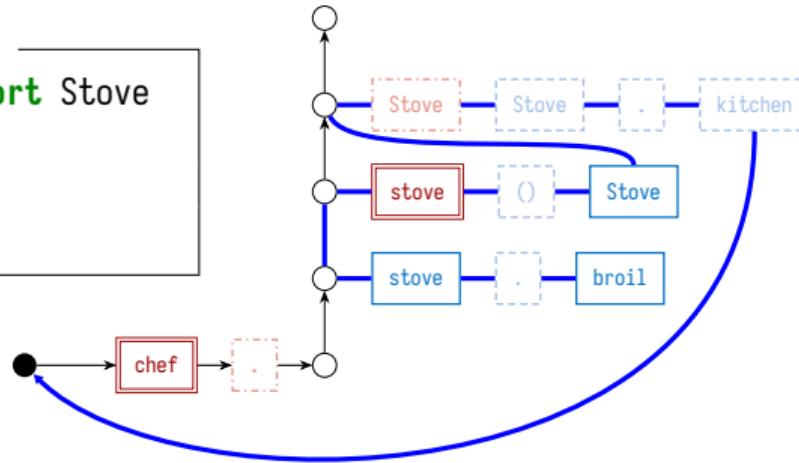
# The more complex example

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



# The more complex example

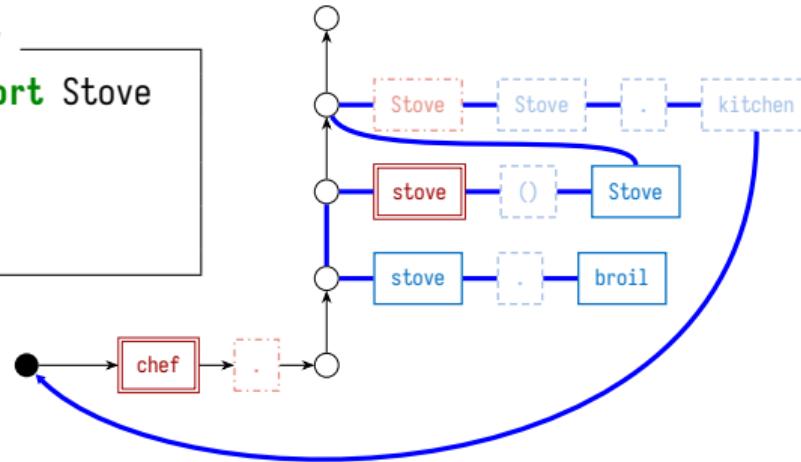
```
chef.py  
from kitchen import Stove  
  
stove = Stove()  
stove.broil()
```



$\langle \rangle$  broil  $\rightsquigarrow$  ● kitchen.Stove().broil

# The more complex example

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



$\langle \rangle \text{ broil} \rightsquigarrow \bullet \langle \text{kitchen.Stove().broil} \rangle$

If you can find what `kitchen.Stove` resolves to and can call it then the result should have a member named `broil` which the reference at `chef.py:4:7` resolves to.

## The more complex example

```
<> broil ~> ● <kitchen.Stove().broil>
```

If you can find what `kitchen.Stove` resolves to and can call it, then the result should have a member named `broil` which the reference at *chef.py:4:7* resolves to.

# The more complex example

`<> broil ~> ● <kitchen.Stove().broil>`

+

`<kitchen.> ● ~> ● <stove.>`

If you can find what `kitchen.Stove` resolves to and can call it, then the result should have a member named `broil` which the reference at `chef.py:4:7` resolves to.

+

If you are looking for `kitchen.[anything]` then you might find it at `stove.[anything]`.

## The more complex example

```
<> broil ~> ● <stove.Stove().broil>
```

If you can find what `stove.Stove` resolves to and can call it, then the result should have a member named `broil` which the reference at *chef.py:4:7* resolves to.

# The more complex example

`<() broil => ● <stove.Stove().broil>` + `<stove.Stove()> ● => ○ IM <>`

If you can find what `stove.Stove` resolves to and can call it, then the result should have a member named `broil` which the reference at `chef.py:4:7` resolves to.

+

Invoking `stove.Stove` gives you an instance of the `Stove` class.

# The more complex example

```
<> broil ~> IM <.broil>
```

The `Stove` class  
should have a member  
named `broil` which the reference at  
*chef.py:4:7* resolves to.

# The more complex example

$\langle \rangle$   $\rightsquigarrow$   $\langle .broil \rangle$

+

$\langle .broil \rangle$   $\rightsquigarrow$   $\langle \rangle$

The **Stove** class  
should have a member  
named **broil** which the reference at  
*chef.py:4:7* resolves to.

+

The **Stove** class has an  
instance member named **broil**  
defined at *stove.py:5:9*.

## The more complex example



The definition at *stove.py:5:9*  
is what the reference at  
*chef.py:4:7* resolves to.

# 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

⋮

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

# The *really* hard ones...

```
dataflow.py  
def passthrough(x):  
    return x  
  
class A:  
    one = 1  
  
passthrough(A).one
```

```
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/2kGKmT>
- Slide 5 Mustang Joe, "I swear..."  
Public domain, <https://flic.kr/p/VSLwD6>
- Slide 12 Marco Verch, "Close-up, a piece of yellow cake with red currant berries"  
CC-BY-2.0, <https://flic.kr/p/2jikJsQ>
- Slide 14 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 31 Marco Verch, "Stack of pancakes with berries on a plate"  
CC-BY-2.0, <https://flic.kr/p/2jYUh8M>
- Slide 39 Seattle Municipal Archives, "West Seattle Bridge under construction, circa 1983"  
CC-BY-2.0, <https://flic.kr/p/7jKWyi>
- Slide 53 Dave Lawler, "Good Morning"  
CC-BY-2.0, <https://flic.kr/p/WYowT7>
- Slide 54 Tony Guyton, "Treehouse Point"  
CC-BY-2.0, <https://flic.kr/p/rRrT5F>

# Picture credits

- Slide 3 Ivan Radic, "Close-up of a compass graffiti on the ground"  
CC-BY-2.0, <https://flic.kr/p/2kGKMtM>
- Slide 5 Mustang Joe, "I swear..."  
Public domain, <https://flic.kr/p/VSLwD6>
- Slide 12 Marco Verch, "Close-up, a piece of yellow cake with red currant berries"  
CC-BY-2.0, <https://flic.kr/p/2jikJsQ>
- Slide 14 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 31 Marco Verch, "Stack of pancakes with berries on a plate"  
CC-BY-2.0, <https://flic.kr/p/2jYUh8M>
- Slide 39 Seattle Municipal Archives, "West Seattle Bridge under construction, circa 1983"  
CC-BY-2.0, <https://flic.kr/p/7jKWYi>
- Slide 53 Dave Lawler, "Good Morning"  
CC-BY-2.0, <https://flic.kr/p/WYowT7>
- Slide 54 Tony Guyton, "Treehouse Point"  
CC-BY-2.0, <https://flic.kr/p/rRrT5F>



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

