

61A Lecture 35

Monday, 28th November, 2011

Last time: sequential data and iterators

Last time: sequential data and iterators

Sequences

Last time: sequential data and iterators

Sequences

- The sequence abstraction so far

Last time: sequential data and iterators

Sequences

- The sequence abstraction so far
 - Length

Last time: sequential data and iterators

Sequences

- The sequence abstraction so far
 - Length
 - Element selection

Last time: sequential data and iterators

Sequences

- The sequence abstraction so far
 - Length
 - Element selection
- Lists and tuples

Last time: sequential data and iterators

Sequences

- The sequence abstraction so far
 - Length
 - Element selection
- Lists and tuples
 - Store all elements up-front

Last time: sequential data and iterators

Sequences

- The sequence abstraction so far
 - Length
 - Element selection
- Lists and tuples
 - Store all elements up-front
 - can't deal with huge data

Last time: sequential data and iterators

Sequences

- The sequence abstraction so far
 - Length
 - Element selection
- Lists and tuples
 - Store all elements up-front
 - can't deal with huge data
 - can't deal with infinite sequences

Last time: sequential data and iterators

Sequences

- The sequence abstraction so far
 - Length
 - Element selection
- Lists and tuples
 - Store all elements up-front
 - can't deal with huge data
 - can't deal with infinite sequences

Iterators

Last time: sequential data and iterators

Sequences

- The sequence abstraction so far
 - Length
 - Element selection
- Lists and tuples
 - Store all elements up-front
 - can't deal with huge data
 - can't deal with infinite sequences

Iterators

- Store how to compute elements

Last time: sequential data and iterators

Sequences

- The sequence abstraction so far
 - Length
 - Element selection
- Lists and tuples
 - Store all elements up-front
 - can't deal with huge data
 - can't deal with infinite sequences

Iterators

- Store how to compute elements
- Compute one element at a time

Last time: sequential data and iterators

Sequences

- The sequence abstraction so far
 - Length
 - Element selection
- Lists and tuples
 - Store all elements up-front
 - can't deal with huge data
 - can't deal with infinite sequences

Iterators

- Store how to compute elements
- Compute one element at a time
- Delay evaluation

Last time: sequential data and iterators

Last time: sequential data and iterators

Streams -- a unit of delayed evaluation.

Last time: sequential data and iterators

Streams -- a unit of delayed evaluation.

- 2 elements, first and rest.

Last time: sequential data and iterators

Streams -- a unit of delayed evaluation.

- 2 elements, first and rest.
 - “first” is stored

Last time: sequential data and iterators

Streams -- a unit of delayed evaluation.

- 2 elements, first and rest.
 - “first” is stored
 - “compute_rest” is stored

Last time: sequential data and iterators

Streams -- a unit of delayed evaluation.

- 2 elements, first and rest.
 - “first” is stored
 - “compute_rest” is stored
 - calculate “rest” on demand

Last time: sequential data and iterators

Streams -- a unit of delayed evaluation.

- 2 elements, first and rest.
 - “first” is stored
 - “compute_rest” is stored
 - calculate “rest” on demand

Native python iterator interface

Last time: sequential data and iterators

Streams -- a unit of delayed evaluation.

- 2 elements, first and rest.
 - “first” is stored
 - “compute_rest” is stored
 - calculate “rest” on demand

Native python iterator interface

- `__iter__()`

Last time: sequential data and iterators

Streams -- a unit of delayed evaluation.

- 2 elements, first and rest.
 - “first” is stored
 - “compute_rest” is stored
 - calculate “rest” on demand

Native python iterator interface

- `__iter__()`
- `__next__()`

Last time: sequential data and iterators

Streams -- a unit of delayed evaluation.

- 2 elements, first and rest.
 - “first” is stored
 - “compute_rest” is stored
 - calculate “rest” on demand

Native python iterator interface

- `__iter__()`
- `__next__()`
- for-loops rely on these methods

Last time: sequential data and iterators

Streams -- a unit of delayed evaluation.

- 2 elements, first and rest.
 - “first” is stored
 - “compute_rest” is stored
 - calculate “rest” on demand

Native python iterator interface

- `__iter__()`
- `__next__()`
- for-loops rely on these methods

Generator functions

Last time: sequential data and iterators

Streams -- a unit of delayed evaluation.

- 2 elements, first and rest.
 - “first” is stored
 - “compute_rest” is stored
 - calculate “rest” on demand

Native python iterator interface

- `__iter__()`
- `__next__()`
- for-loops rely on these methods

Generator functions

- Functions that use `yield` to output values

Last time: sequential data and iterators

Streams -- a unit of delayed evaluation.

- 2 elements, first and rest.
 - “first” is stored
 - “compute_rest” is stored
 - calculate “rest” on demand

Native python iterator interface

- `__iter__()`
- `__next__()`
- for-loops rely on these methods

Generator functions

- Functions that use `yield` to output values
- Creates a generator object

Last time: sequential data and iterators

Streams -- a unit of delayed evaluation.

- 2 elements, first and rest.
 - “first” is stored
 - “compute_rest” is stored
 - calculate “rest” on demand

Native python iterator interface

- `__iter__()`
- `__next__()`
- for-loops rely on these methods

Generator functions

- Functions that use `yield` to output values
- Creates a generator object
- `__iter__()` and `__next__()` automatically defined

Today: modularity, processing pipelines, and coroutines

Modularity in programs so far

- Helper functions a.k.a “subroutines”

Coroutines: what are they?

Coroutines in python

Types of coroutines

Multitasking

Modularity so far: helper functions

Modularity so far: helper functions

Modularity in programming?

Modularity so far: helper functions

Modularity in programming?

- Helper functions!

Modularity so far: helper functions

Modularity in programming?

- Helper functions!
 - a.k.a. “subroutines”

Modularity so far: helper functions

Modularity in programming?

- Helper functions!
 - a.k.a. “subroutines”
- A sub-program responsible for a small piece of computation

Modularity so far: helper functions

Modularity in programming?

- Helper functions!
 - a.k.a. “subroutines”
- A sub-program responsible for a small piece of computation

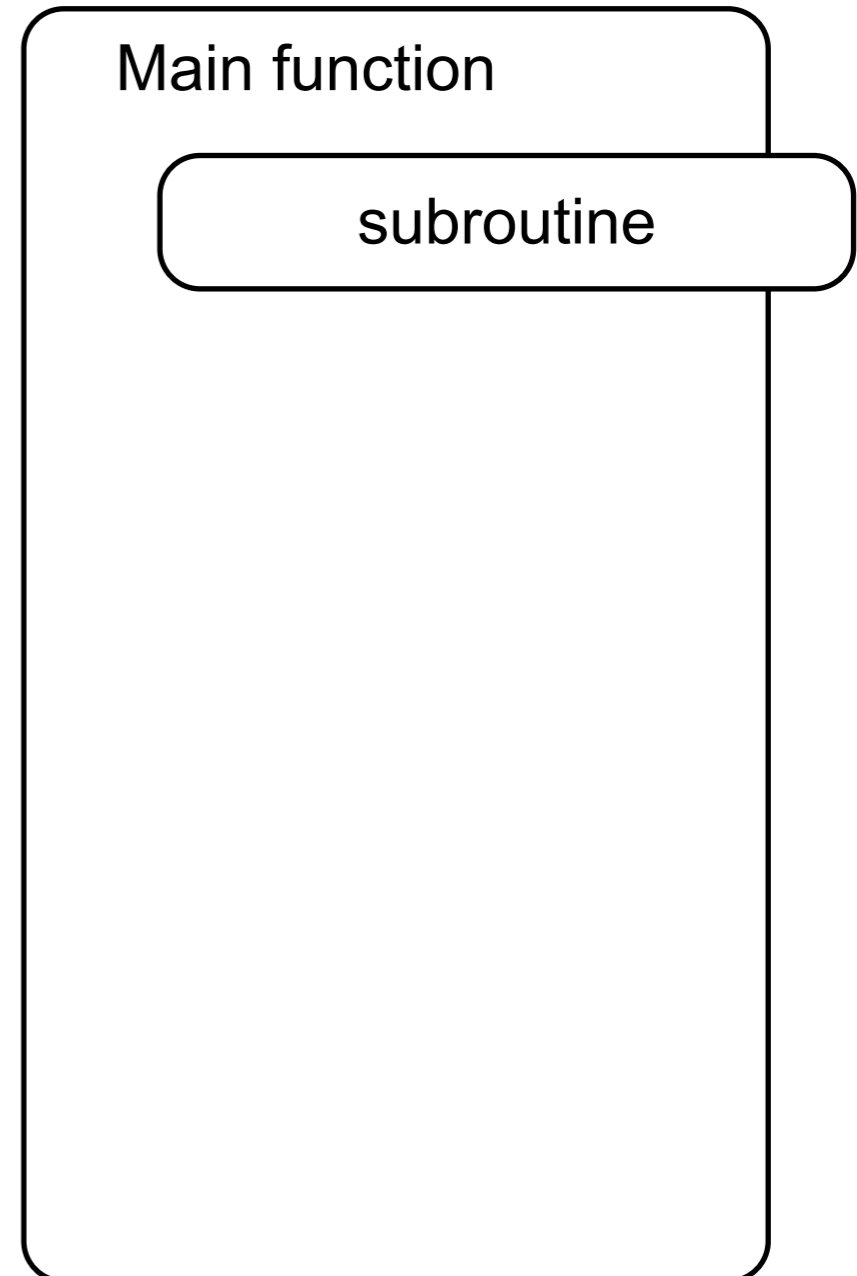


Main function

Modularity so far: helper functions

Modularity in programming?

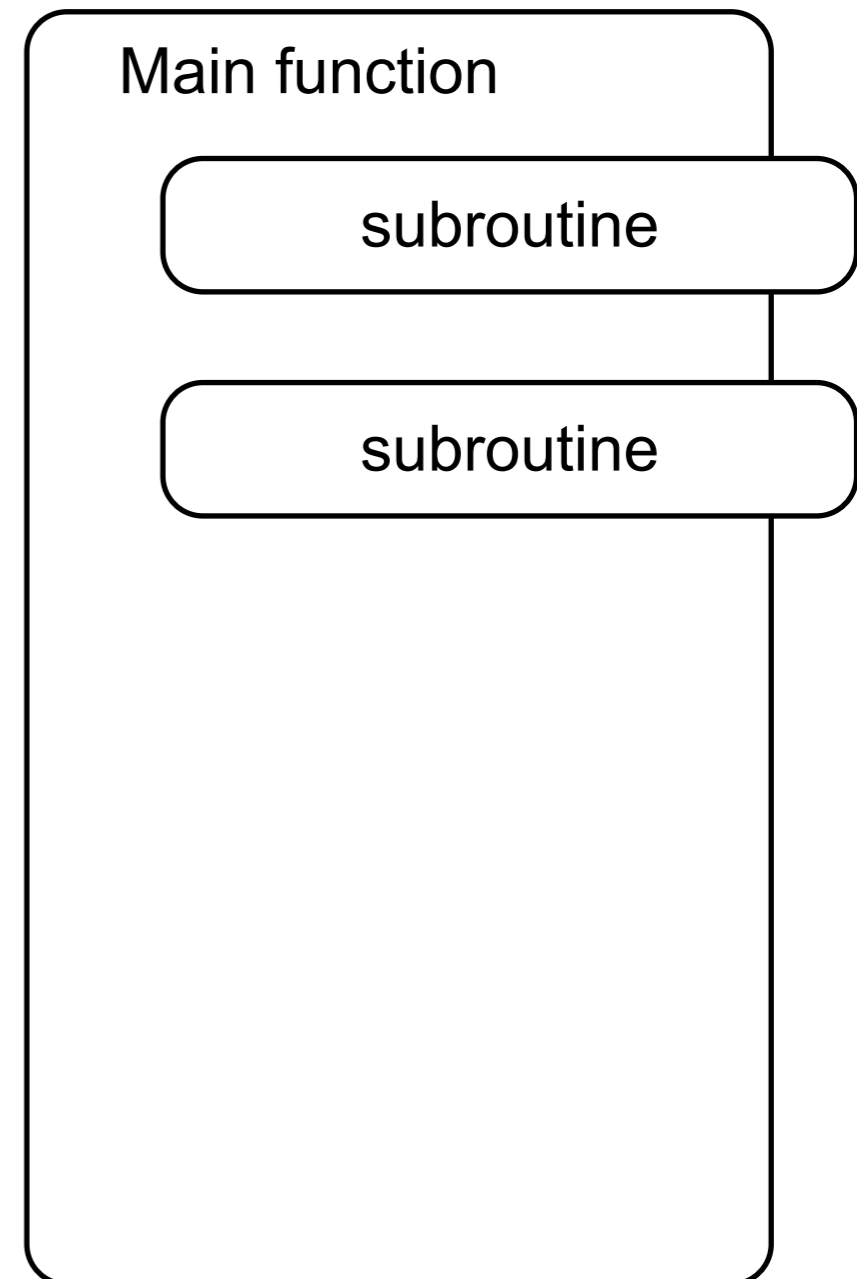
- Helper functions!
 - a.k.a. “subroutines”
- A sub-program responsible for a small piece of computation



Modularity so far: helper functions

Modularity in programming?

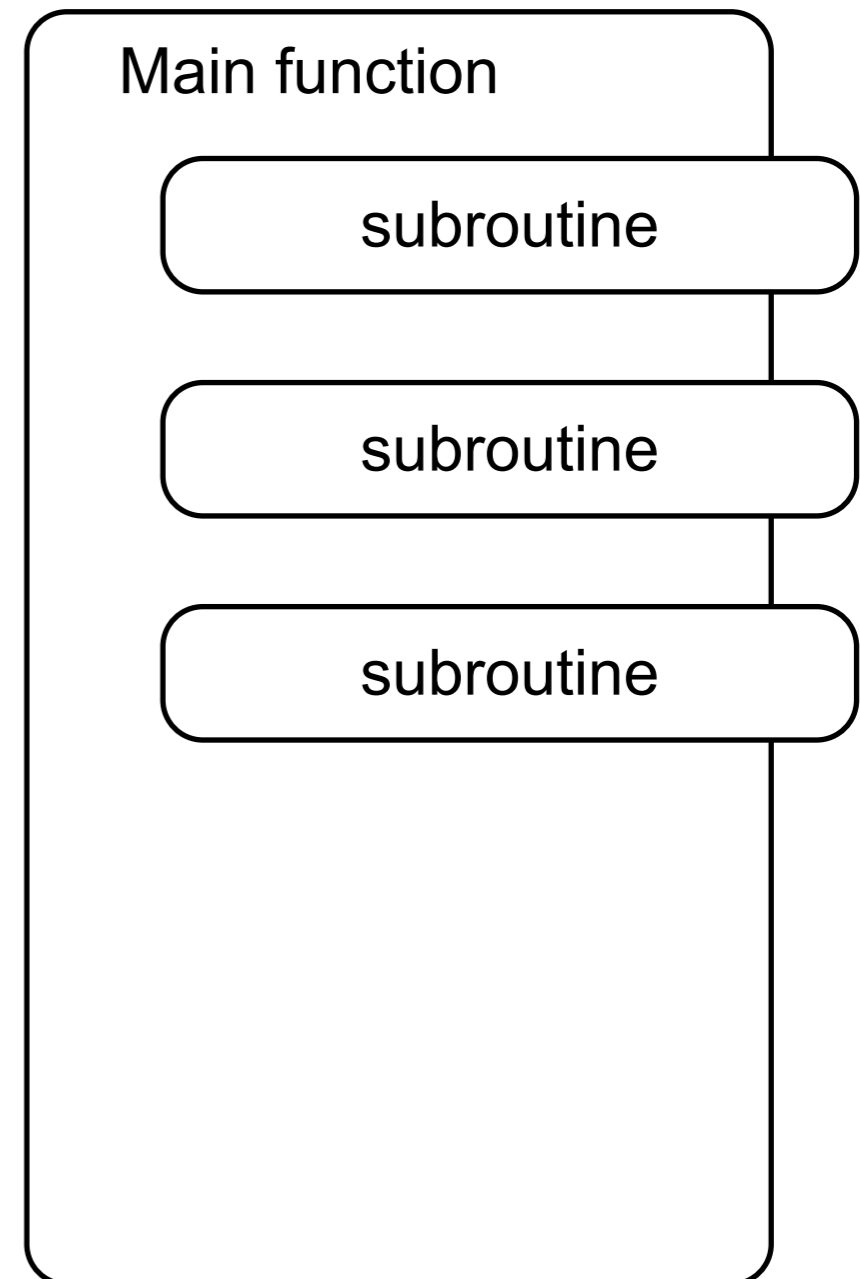
- Helper functions!
 - a.k.a. “subroutines”
- A sub-program responsible for a small piece of computation



Modularity so far: helper functions

Modularity in programming?

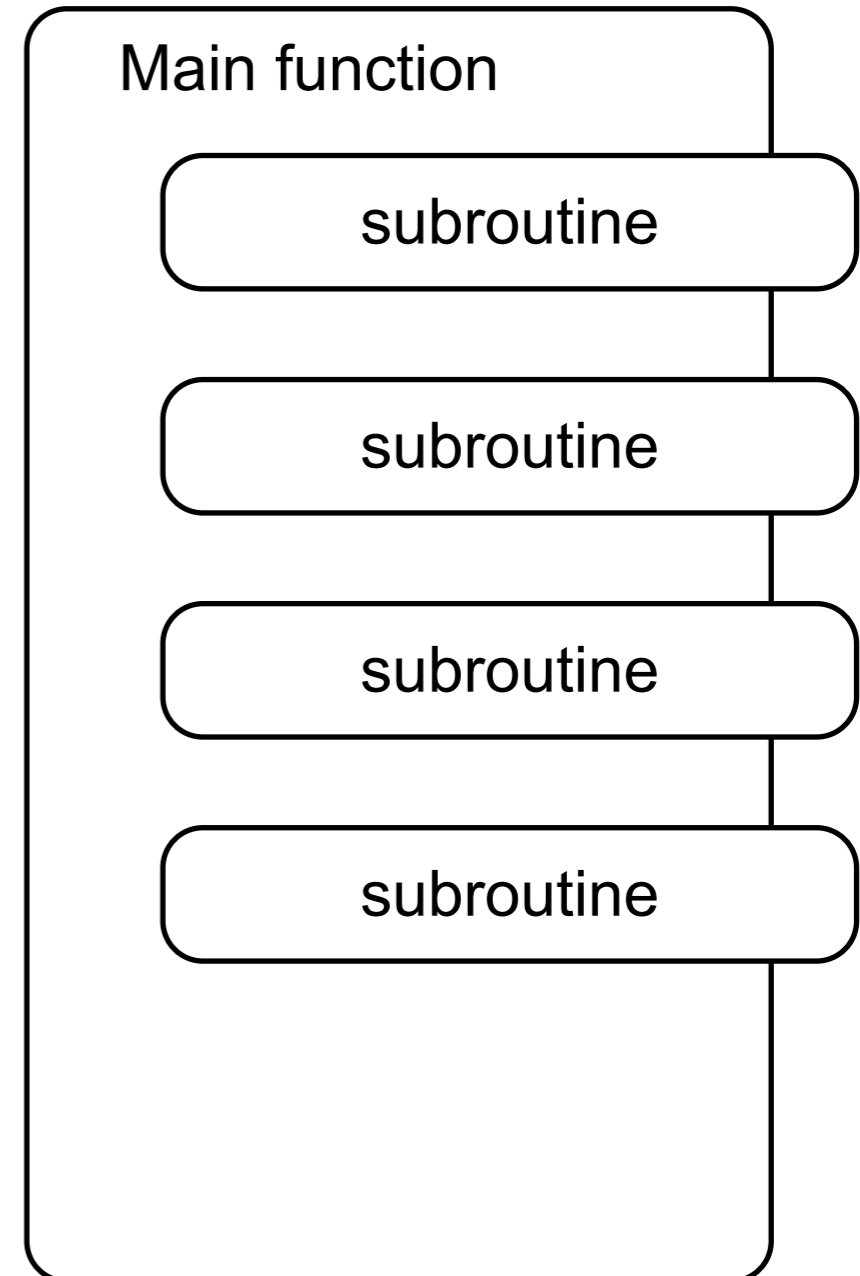
- Helper functions!
 - a.k.a. “subroutines”
- A sub-program responsible for a small piece of computation



Modularity so far: helper functions

Modularity in programming?

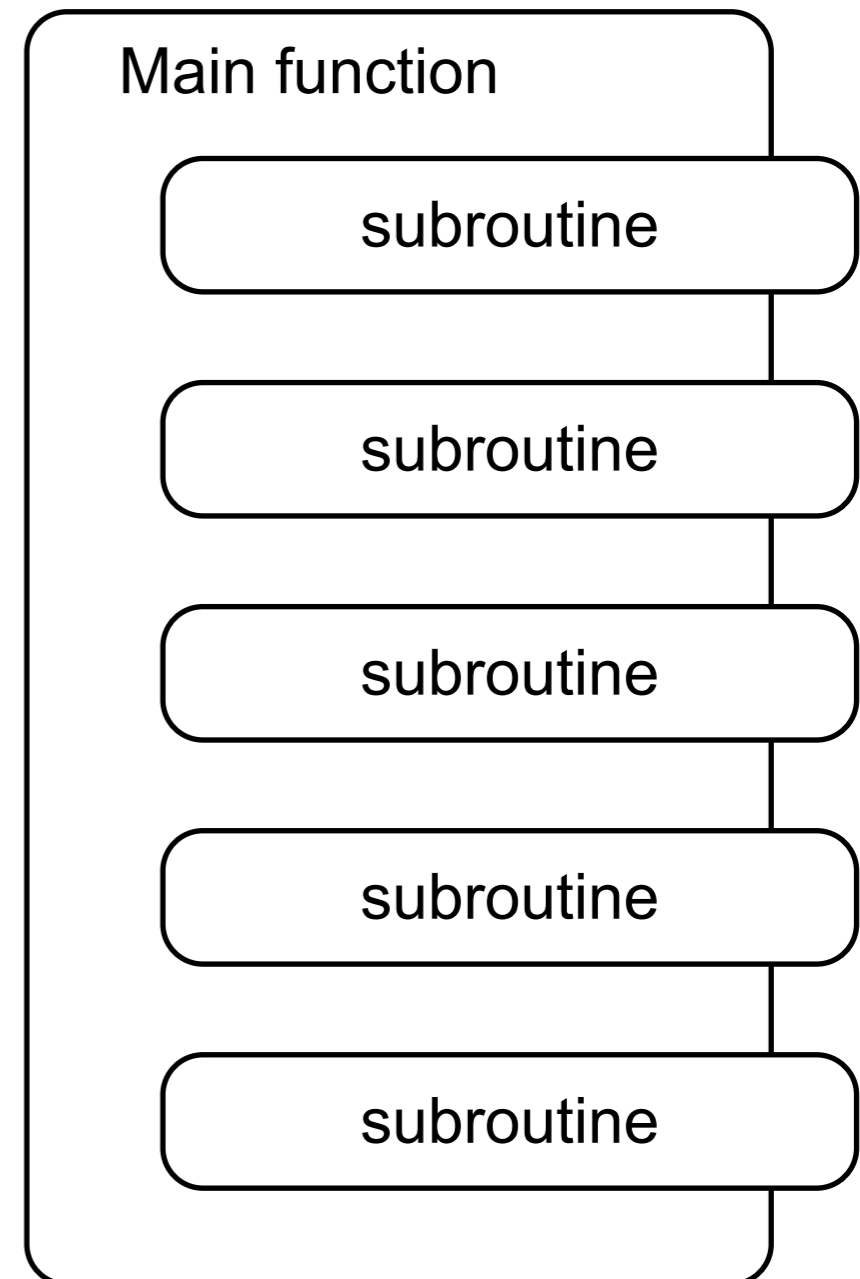
- Helper functions!
 - a.k.a. “subroutines”
- A sub-program responsible for a small piece of computation



Modularity so far: helper functions

Modularity in programming?

- Helper functions!
 - a.k.a. “subroutines”
- A sub-program responsible for a small piece of computation

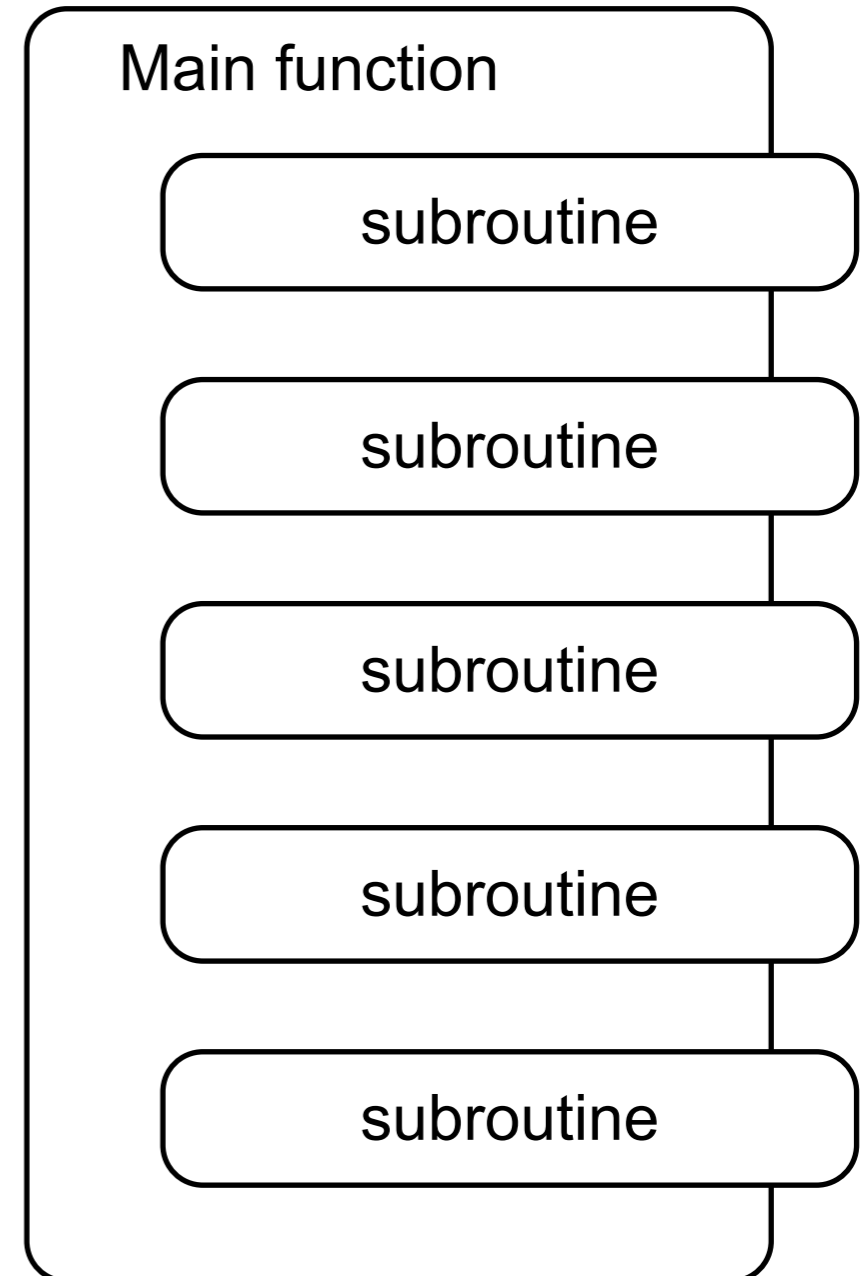


Modularity so far: helper functions

Modularity in programming?

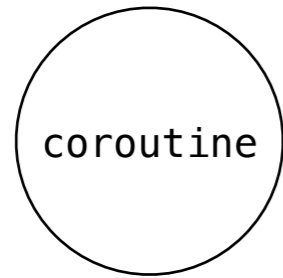
- Helper functions!
 - a.k.a. “subroutines”
- A sub-program responsible for a small piece of computation

A main function is responsible for calling all the subroutines

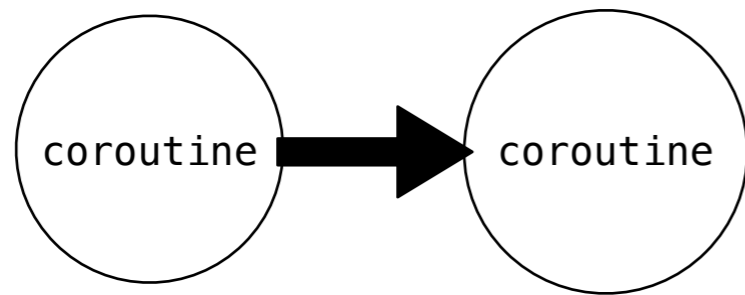


Modularity with Coroutines

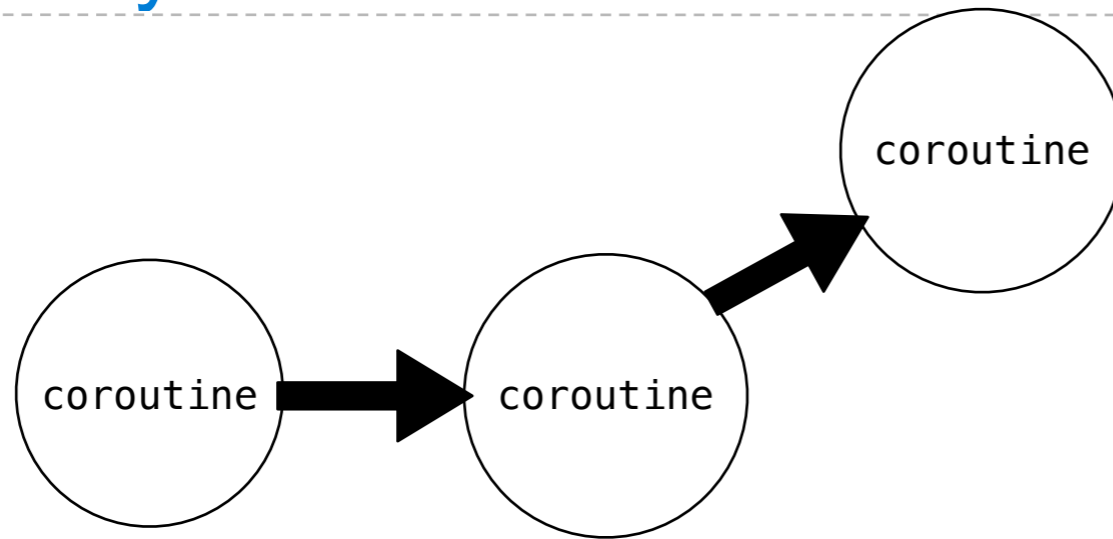
Modularity with Coroutines



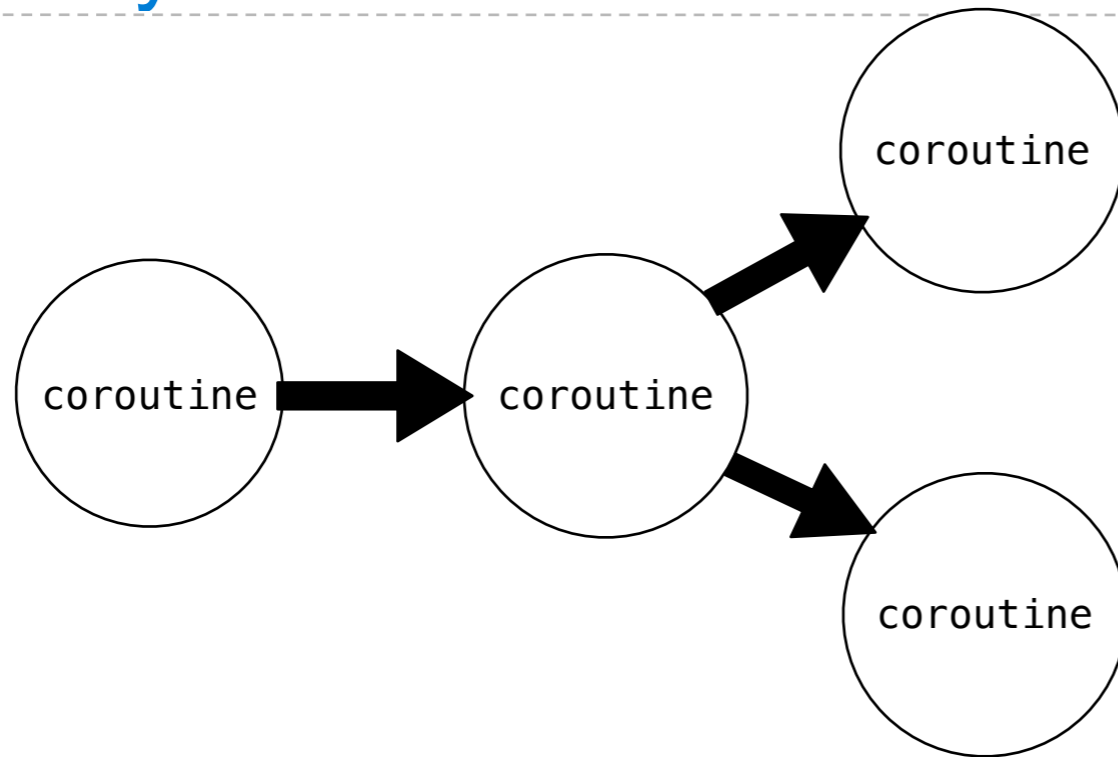
Modularity with Coroutines



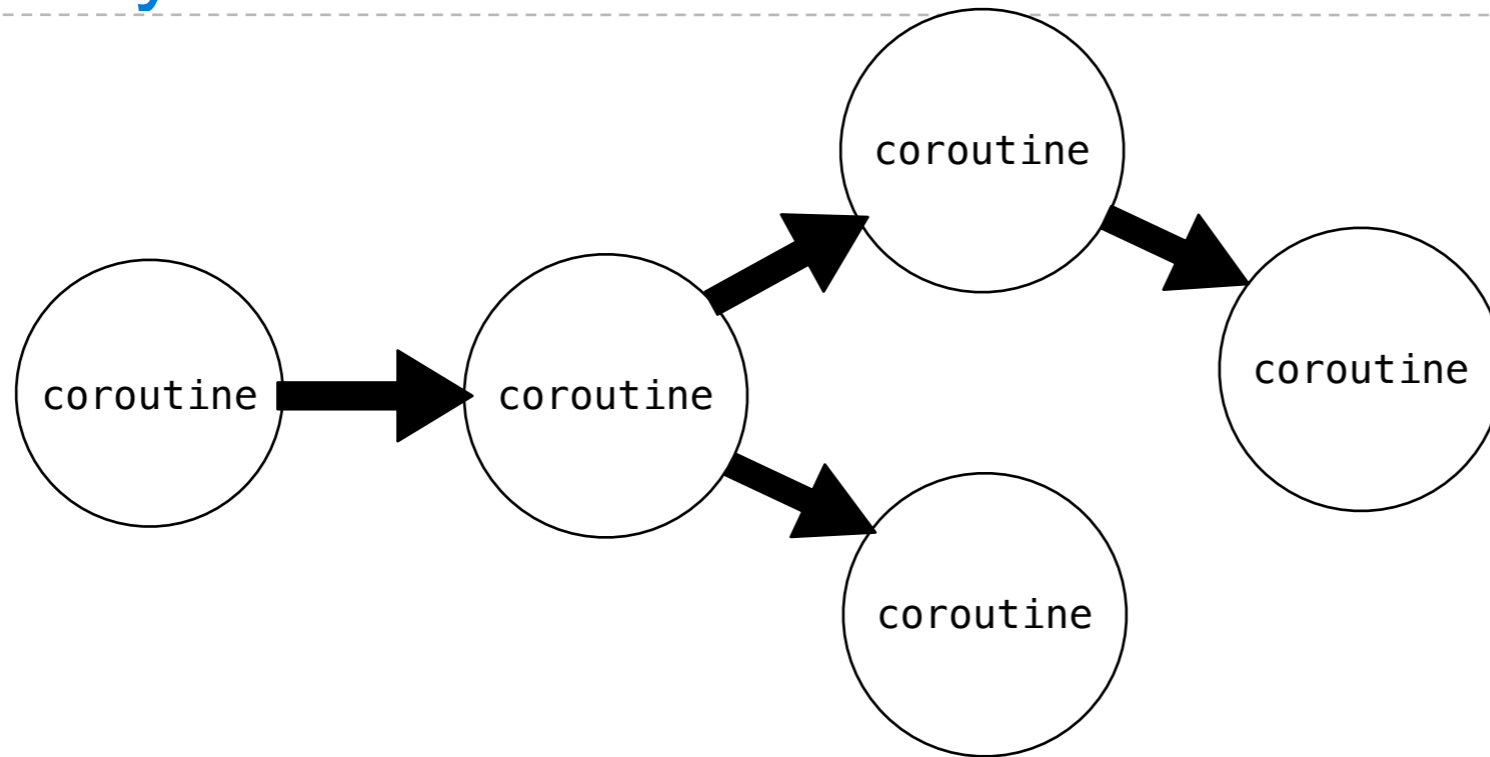
Modularity with Coroutines



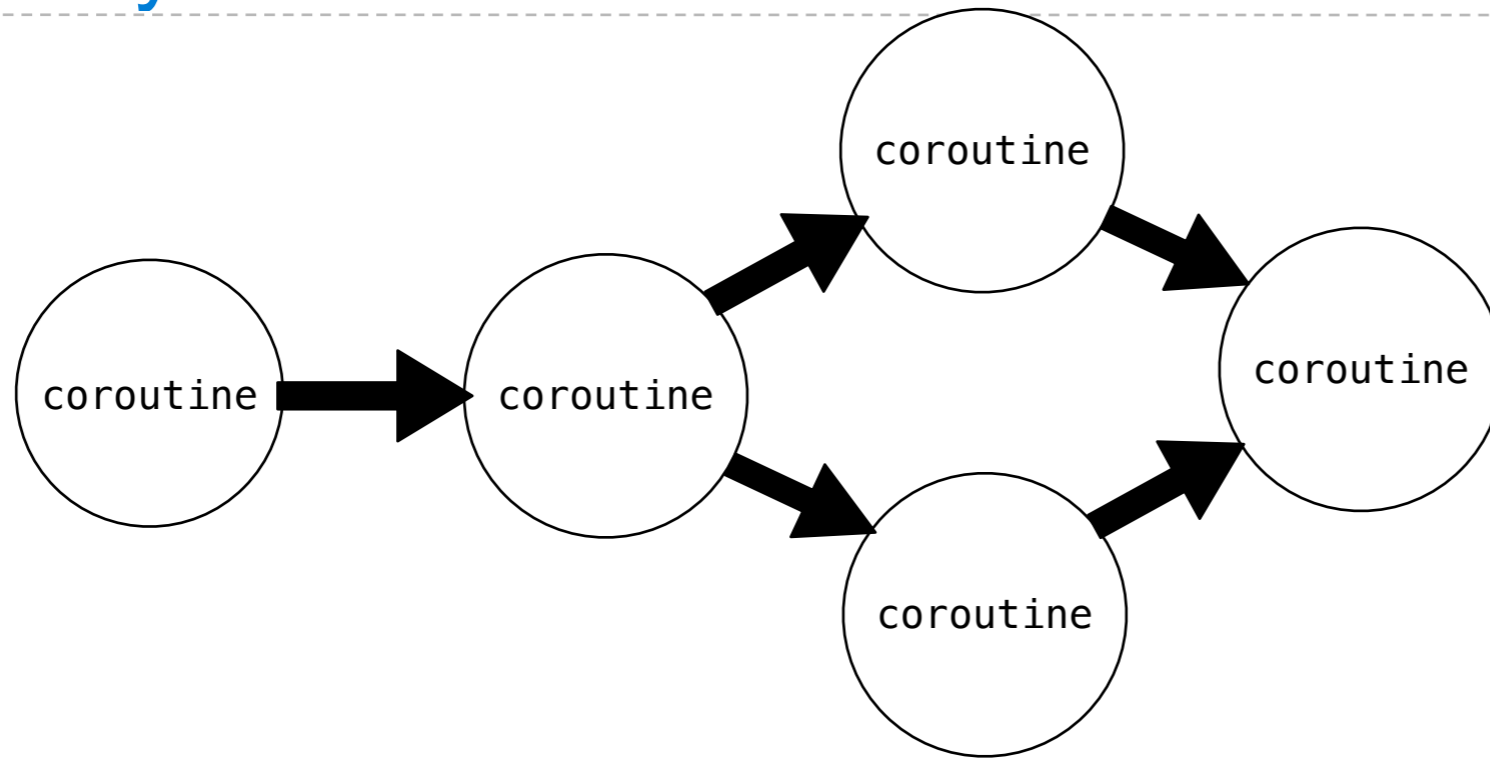
Modularity with Coroutines



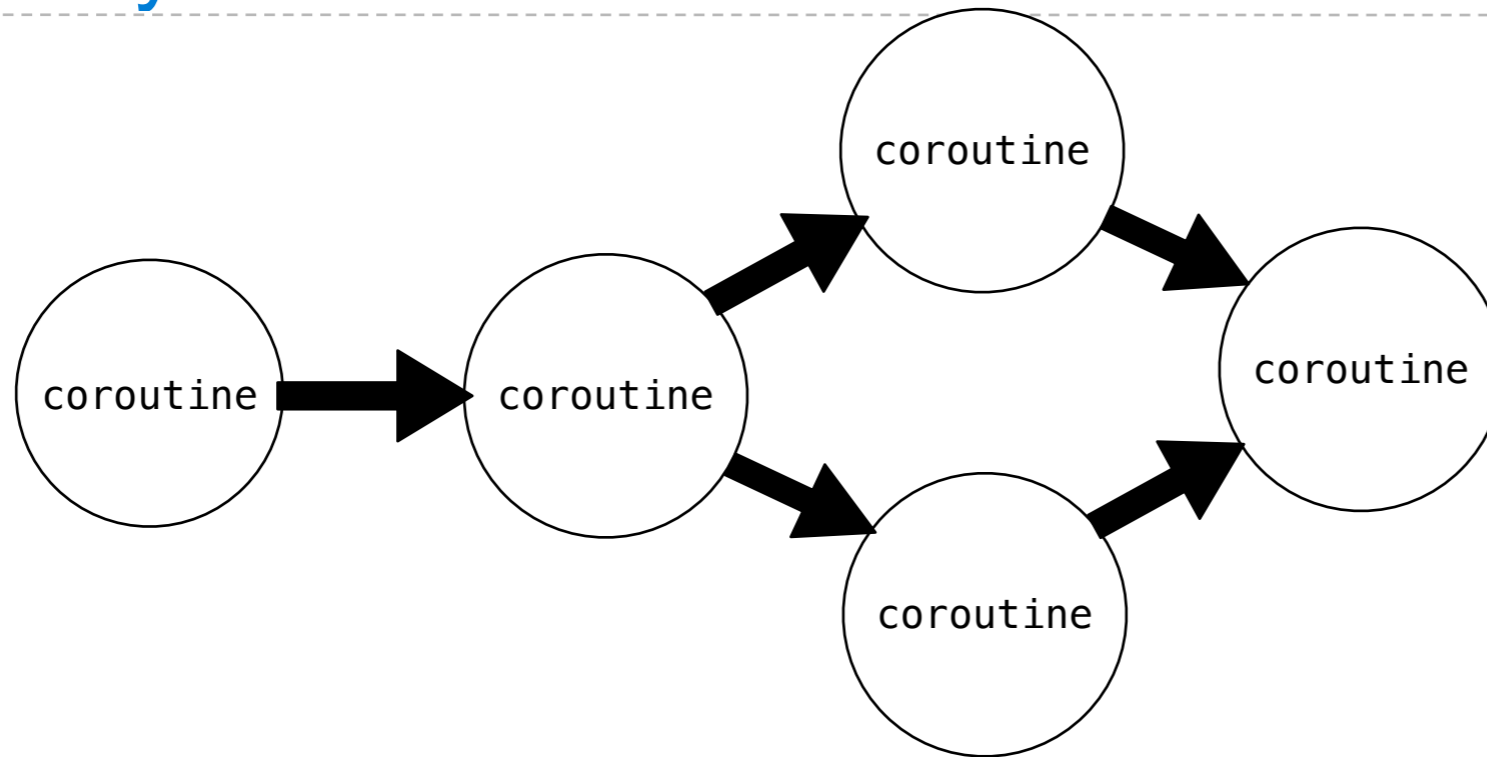
Modularity with Coroutines



Modularity with Coroutines



Modularity with Coroutines

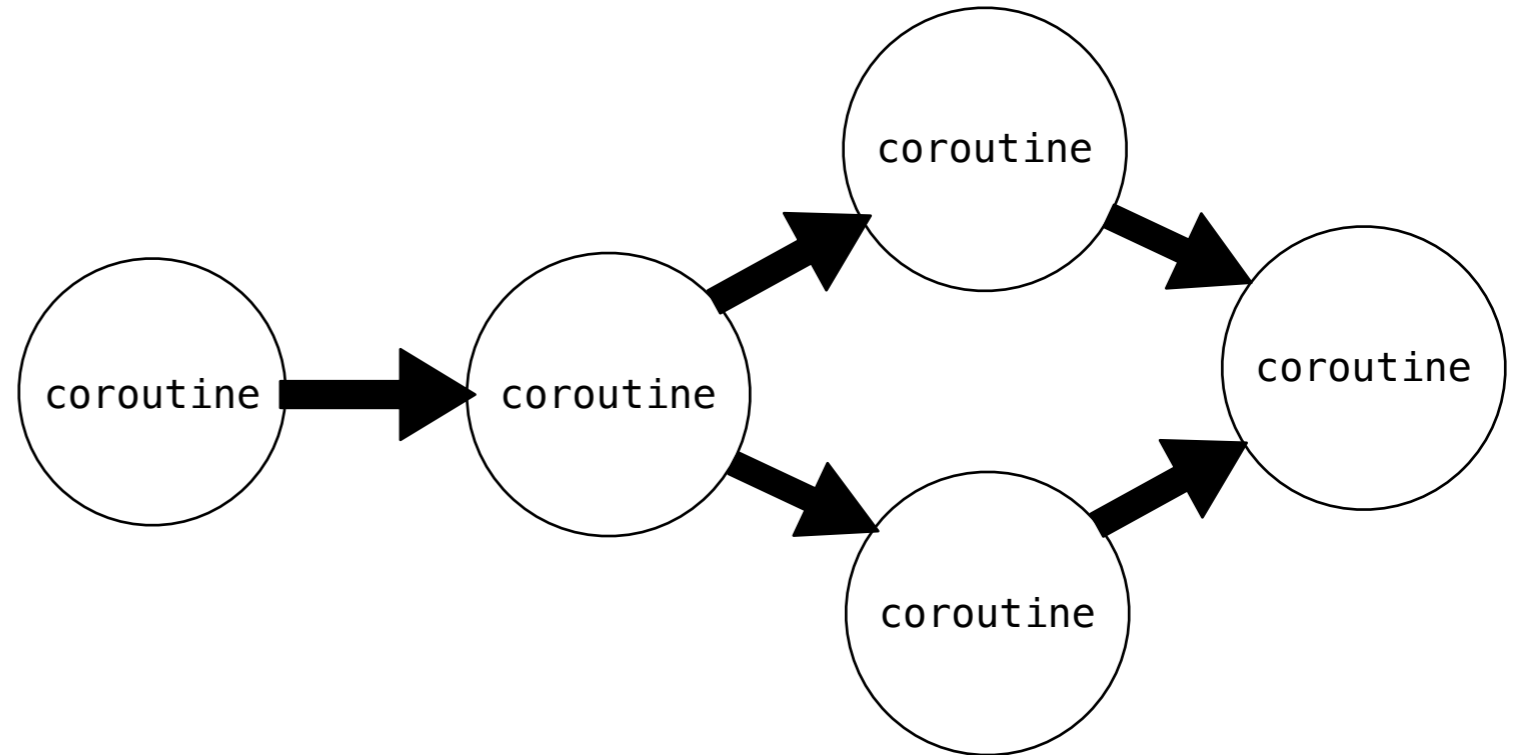
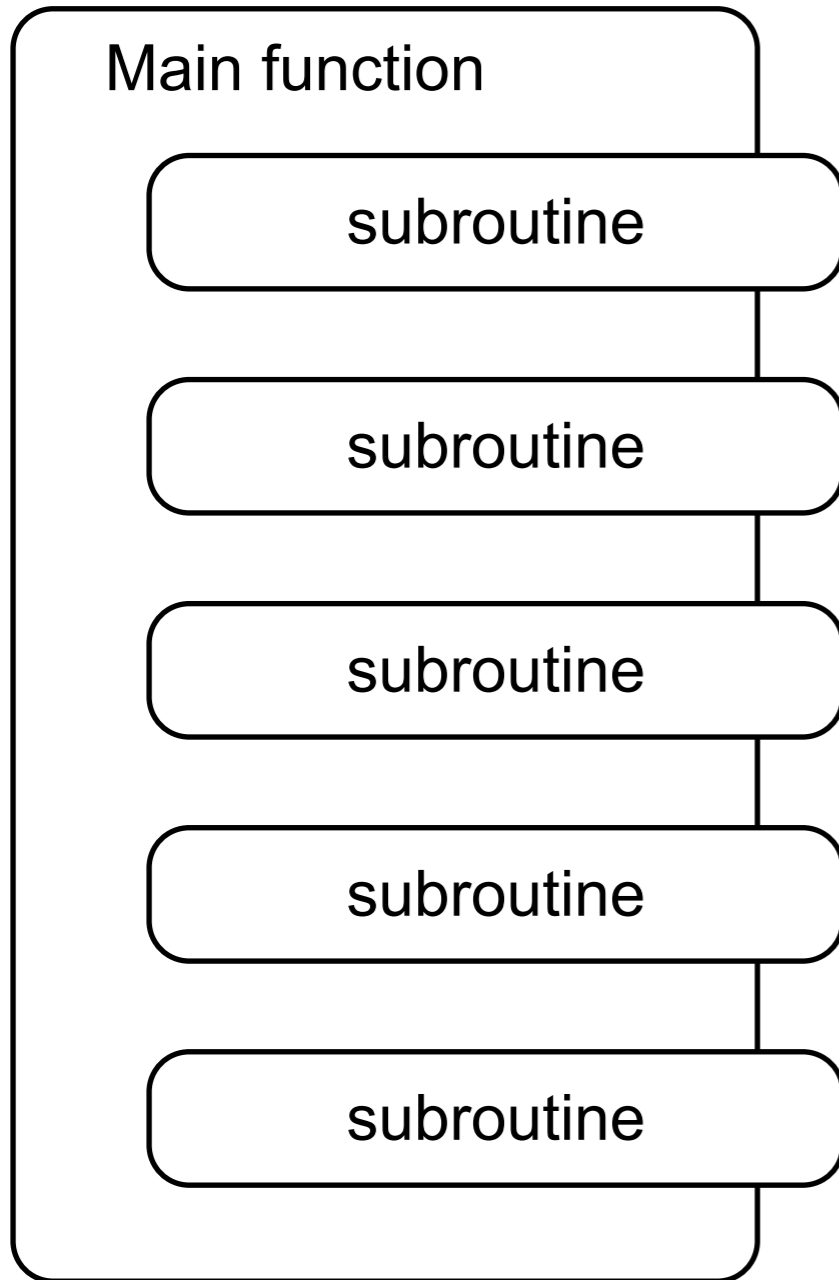


Coroutines are *also* sub-computations

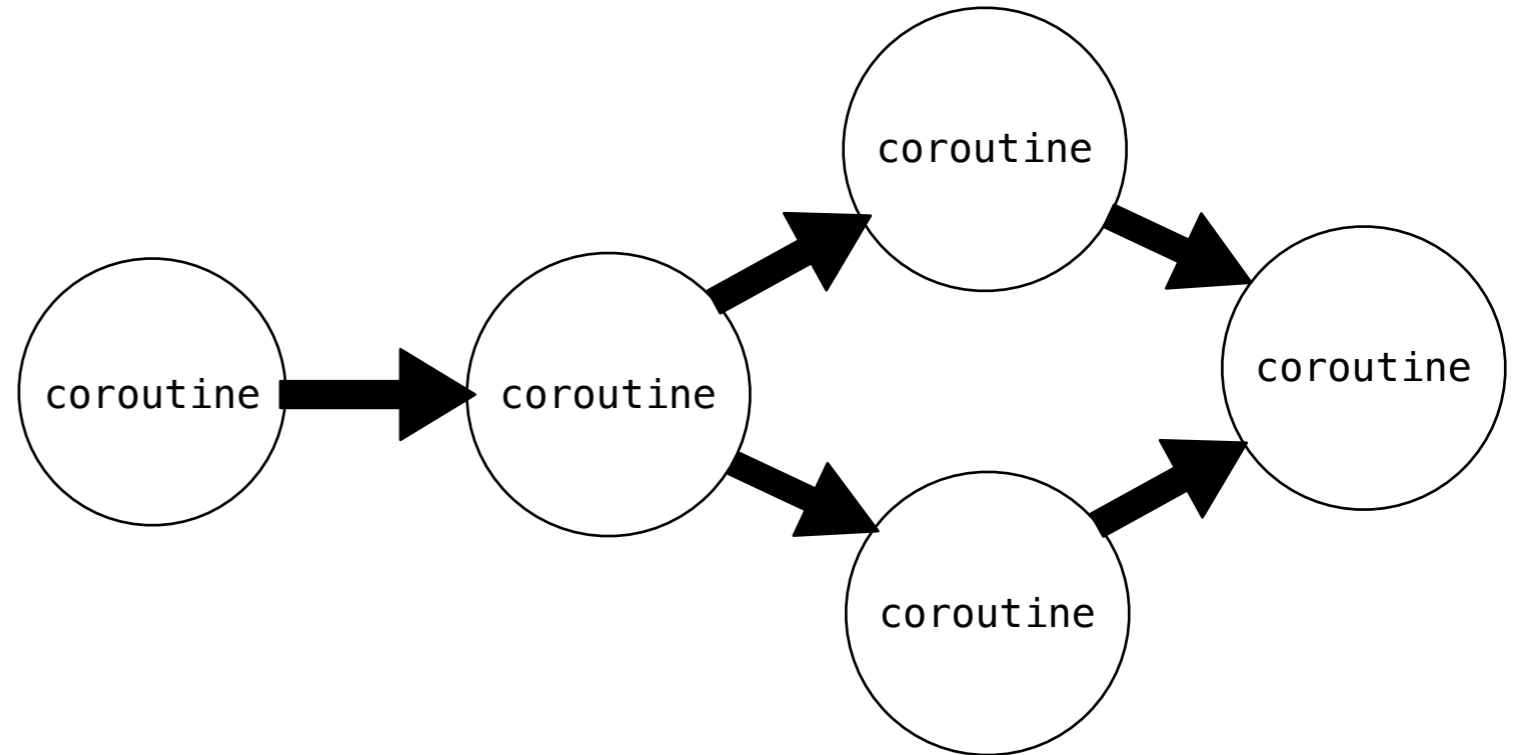
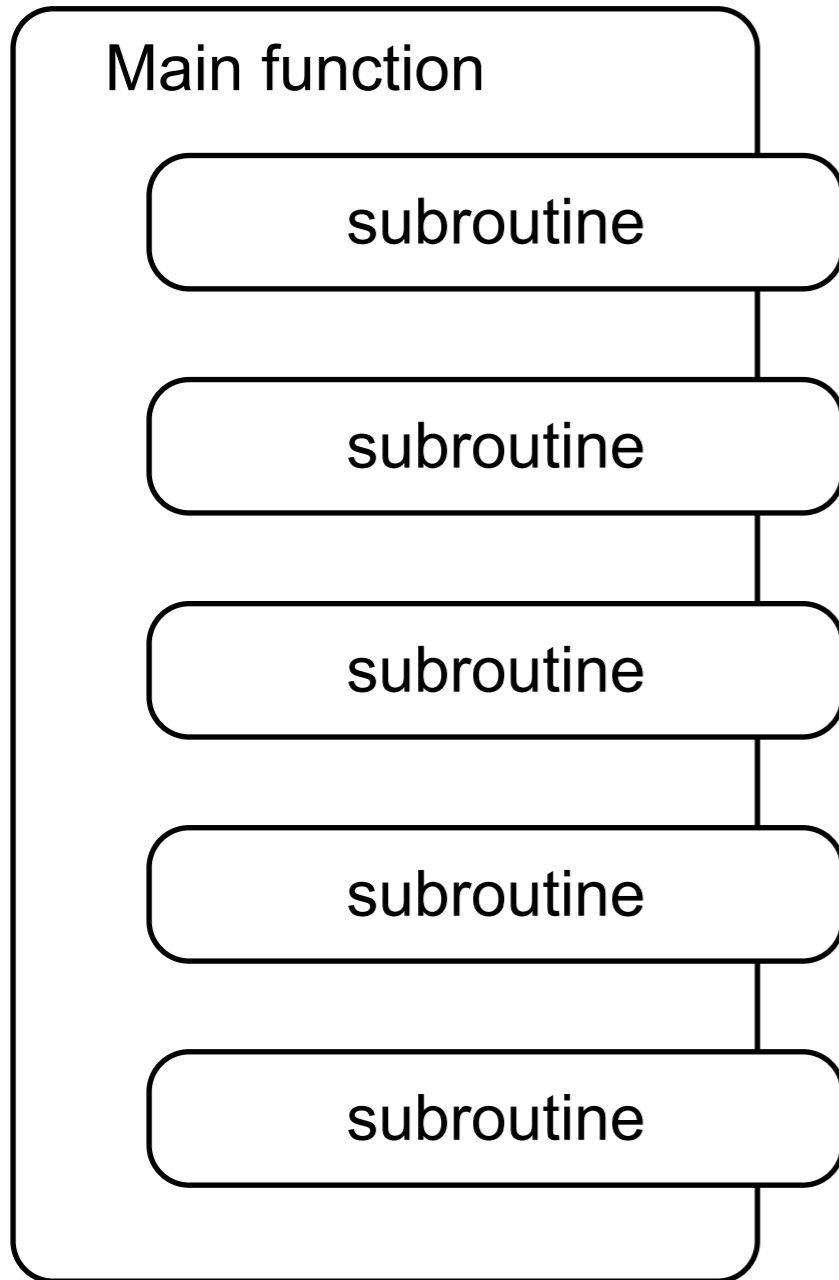
The difference: no main function

Separate coroutines link together to form a complete pipeline

Coroutines vs. subroutines: a conceptual difference

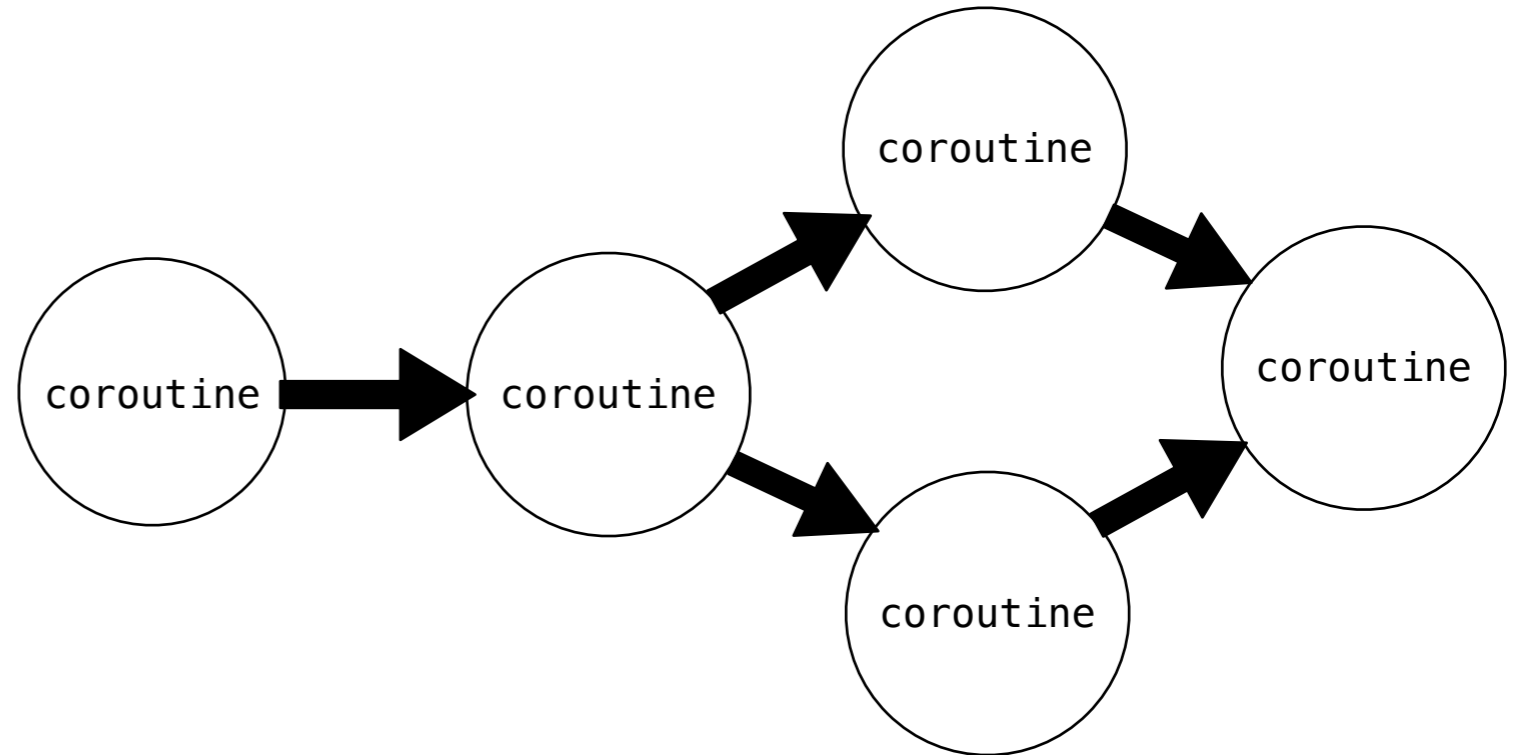
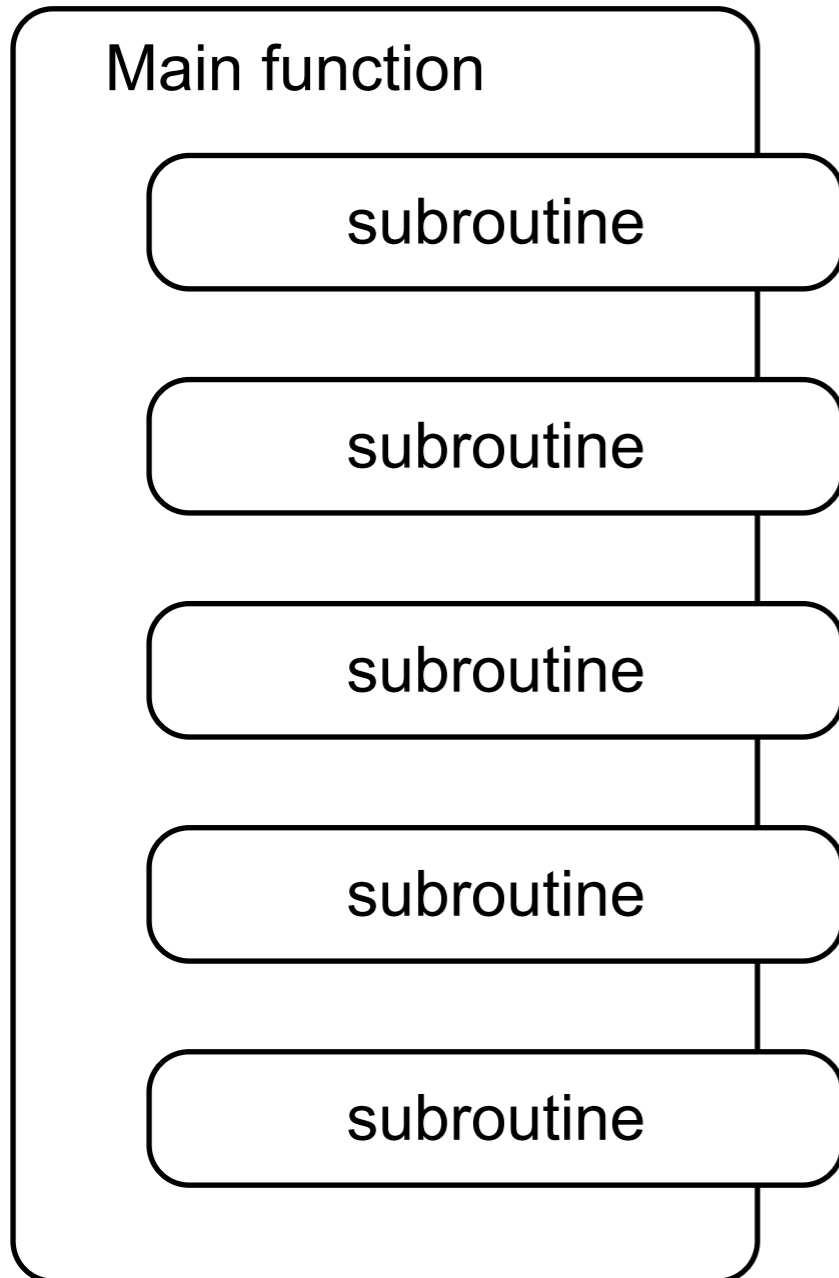


Coroutines vs. subroutines: a conceptual difference



subordinate to a main function

Coroutines vs. subroutines: a conceptual difference



colleagues that cooperate

subordinate to a main function

Coroutines in python, or, the many faces of “yield”

Coroutines in python, or, the many faces of “yield”

Previously: generator functions

Coroutines in python, or, the many faces of “yield”

Previously: generator functions

- ***Produce*** data with yield

Coroutines in python, or, the many faces of “yield”

Previously: generator functions

- **Produce** data with yield

```
def letters_generator():
```


Coroutines in python, or, the many faces of “yield”

Previously: generator functions

- **Produce** data with yield

```
def letters_generator():  
    current = 'a'
```

Coroutines in python, or, the many faces of “yield”

Previously: generator functions

- **Produce** data with yield

```
def letters_generator():  
    current = 'a'  
    while current <= 'd':
```

Coroutines in python, or, the many faces of “yield”

Previously: generator functions

- **Produce** data with yield

```
def letters_generator():  
    current = 'a'  
    while current <= 'd':  
        yield current
```

Coroutines in python, or, the many faces of “yield”

Previously: generator functions

- **Produce** data with yield


```
def letters_generator():  
    current = 'a'  
    while current <= 'd':  
        yield current  
        current = chr(ord(current)+1)
```

Coroutines in python, or, the many faces of “yield”

Previously: generator functions

- **Produce** data with yield

```
def letters_generator():  
    current = 'a'  
    while current <= 'd':  
        yield current  
        current = chr(ord(current)+1)
```




Coroutines in python, or, the many faces of “yield”

Previously: generator functions

- **Produce** data with yield pauses execution

```
def letters_generator():  
    current = 'a'  
    while current <= 'd':  
        yield current  
        current = chr(ord(current)+1)
```




Coroutines in python, or, the many faces of “yield”

Previously: generator functions

- **Produce** data with yield

pauses execution
local variables preserved

```
def letters_generator():  
    current = 'a'  
    while current <= 'd':  
        yield current  
        current = chr(ord(current)+1)
```




Coroutines in python, or, the many faces of “yield”

Previously: generator functions

- **Produce** data with yield

pauses execution
local variables preserved
resumes when **.__next__** is called

```
def letters_generator():  
    current = 'a'  
    while current <= 'd':  
        yield current  
        current = chr(ord(current)+1)
```



Coroutines in python, or, the many faces of “yield”

Previously: generator functions

- **Produce** data with yield

```
def letters_generator():  
    current = 'a'  
    while current <= 'd':  
        yield current  
        current = chr(ord(current)+1)
```

pauses execution
local variables preserved
resumes when **.__next__** is called
returns the yielded value



Coroutines in python, or, the many faces of “yield”

Previously: generator functions

- **Produce** data with yield

```
def letters_generator():  
    current = 'a'  
    while current <= 'd':  
        yield current  
        current = chr(ord(current)+1)
```

pauses execution
local variables preserved
resumes when `.__next__` is called
returns the yielded value



Now: coroutines

Coroutines in python, or, the many faces of “yield”

Previously: generator functions

- **Produce** data with yield

```
def letters_generator():  
    current = 'a'  
    while current <= 'd':  
        yield current  
        current = chr(ord(current)+1)
```

pauses execution
local variables preserved
resumes when `.__next__` is called
returns the yielded value



Now: coroutines

- **Consume** data with yield

Coroutines in python, or, the many faces of “yield”

Previously: generator functions

- **Produce** data with yield

```
def letters_generator():  
    current = 'a'  
    while current <= 'd':  
        yield current  
        current = chr(ord(current)+1)
```

pauses execution
local variables preserved
resumes when `.__next__` is called
returns the yielded value



Now: coroutines

- **Consume** data with yield

```
value = (yield)
```

Coroutines in python, or, the many faces of “yield”

Previously: generator functions

- **Produce** data with yield

```
def letters_generator():  
    current = 'a'  
    while current <= 'd':  
        yield current  
        current = chr(ord(current)+1)
```

pauses execution
local variables preserved
resumes when `.__next__` is called
returns the yielded value



Now: coroutines

- **Consume** data with yield

```
value = (yield)
```



Coroutines in python, or, the many faces of “yield”

Previously: generator functions

- **Produce** data with yield

```
def letters_generator():  
    current = 'a'  
    while current <= 'd':  
        yield current  
        current = chr(ord(current)+1)
```

pauses execution
local variables preserved
resumes when `.__next__` is called
returns the yielded value



Now: coroutines

pauses execution

- **Consume** data with yield

```
value = (yield)
```



Coroutines in python, or, the many faces of “yield”

Previously: generator functions

- **Produce** data with yield

```
def letters_generator():  
    current = 'a'  
    while current <= 'd':  
        yield current  
        current = chr(ord(current)+1)
```

pauses execution
local variables preserved
resumes when `.__next__` is called
returns the yielded value



Now: coroutines

- **Consume** data with yield

```
value = (yield)
```

pauses execution
local variables preserved



Coroutines in python, or, the many faces of “yield”

Previously: generator functions

- **Produce** data with yield

```
def letters_generator():  
    current = 'a'  
    while current <= 'd':  
        yield current  
        current = chr(ord(current)+1)
```

pauses execution
local variables preserved
resumes when **.__next__** is called
returns the yielded value



Now: coroutines

- **Consume** data with yield

```
value = (yield)
```

pauses execution
local variables preserved
resumes when **.send(data)** is called



Coroutines in python, or, the many faces of “yield”

Previously: generator functions

- **Produce** data with yield

```
def letters_generator():  
    current = 'a'  
    while current <= 'd':  
        yield current  
        current = chr(ord(current)+1)
```

pauses execution
local variables preserved
resumes when **.__next__** is called
returns the yielded value



Now: coroutines

- **Consume** data with yield

```
value = (yield)
```

pauses execution
local variables preserved
resumes when **.send(data)** is called
assigns value to yielded data



Coroutines in python, or, the many faces of “yield”

Previously: generator functions

- **Produce** data with yield

```
def letters_generator():  
    current = 'a'  
    while current <= 'd':  
        yield current  
        current = chr(ord(current)+1)
```

pauses execution
local variables preserved
resumes when `.__next__` is called
returns the yielded value



Now: coroutines

- **Consume** data with yield

```
value = (yield)
```

pauses execution
local variables preserved
resumes when `.send(data)` is called
assigns value to yielded data



send(data)

Coroutines in python, or, the many faces of “yield”

Previously: generator functions

- **Produce** data with yield

```
def letters_generator():  
    current = 'a'  
    while current <= 'd':  
        yield current  
        current = chr(ord(current)+1)
```

pauses execution
local variables preserved
resumes when `.__next__` is called
returns the yielded value



Now: coroutines

- **Consume** data with yield

```
value = (yield)
```

pauses execution
local variables preserved
resumes when `.send(data)` is called
assigns value to yielded data



```
value = (yield)
```

```
send(data)
```

Coroutines in python, or, the many faces of “yield”

Previously: generator functions

- **Produce** data with yield

```
def letters_generator():  
    current = 'a'  
    while current <= 'd':  
        yield current  
        current = chr(ord(current)+1)
```

pauses execution
local variables preserved
resumes when `.__next__` is called
returns the yielded value

Now: coroutines

- **Consume** data with yield

```
value = (yield)
```

pauses execution
local variables preserved
resumes when `.send(data)` is called
assigns value to yielded data

```
value = (yield)  
send(data)
```

Coroutines in python, or, the many faces of “yield”

Previously: generator functions

- **Produce** data with yield

```
def letters_generator():  
    current = 'a'  
    while current <= 'd':  
        yield current  
        current = chr(ord(current)+1)
```

pauses execution
local variables preserved
resumes when `.__next__` is called
returns the yielded value


Now: coroutines

- **Consume** data with yield

```
value = (yield)
```

pauses execution
local variables preserved
resumes when `.send(data)` is called
assigns value to yielded data

```
value = (yield)
```

`send(data)`  `(yield)` returns the sent data.

Coroutines in python, or, the many faces of “yield”

Previously: generator functions

- **Produce** data with yield

```
def letters_generator():  
    current = 'a'  
    while current <= 'd':  
        yield current  
        current = chr(ord(current)+1)
```

pauses execution
local variables preserved
resumes when **.__next__** is called
returns the yielded value



Now: coroutines

- **Consume** data with yield

```
value = (yield)
```

pauses execution
local variables preserved
resumes when **.send(data)** is called
assigns value to yielded data



```
value = (yield)  
send(data)
```

(yield) returns the sent data.
Execution resumes

Coroutines in Python

Coroutines in Python

Consuming data with `yield`:

Coroutines in Python

Consuming data with `yield`:

- `value = (yield)`

Coroutines in Python

Consuming data with `yield`:

- `value = (yield)`
- Execution pauses waiting for data to be sent

Coroutines in Python

Consuming data with `yield`:

- `value = (yield)`
- Execution pauses waiting for data to be sent

Send a coroutine data using `send(...)`

Coroutines in Python

Consuming data with `yield`:

- `value = (yield)`
- Execution pauses waiting for data to be sent

Send a coroutine data using `send(...)`

Start a coroutine using `__next__()`

Coroutines in Python

Consuming data with `yield`:

- `value = (yield)`
- Execution pauses waiting for data to be sent

Send a coroutine data using `send(...)`

Start a coroutine using `__next__()`

Signal the end of a computation using `close()`

Coroutines in Python

Consuming data with `yield`:

- `value = (yield)`
- Execution pauses waiting for data to be sent

Send a coroutine data using `send(...)`

Start a coroutine using `__next__()`

Signal the end of a computation using `close()`

- Raises `GeneratorExit` exception inside coroutine

Example: print out strings that match a pattern

Example: print out strings that match a pattern

```
def match(pattern):
```


Example: print out strings that match a pattern

```
def match(pattern):  
    print('Looking for ' + pattern)
```

Example: print out strings that match a pattern

```
def match(pattern):  
    print('Looking for ' + pattern)  
    try:
```

Example: print out strings that match a pattern

```
def match(pattern):  
    print('Looking for ' + pattern)  
    try:  
        while True:
```

Example: print out strings that match a pattern

```
def match(pattern):  
    print('Looking for ' + pattern)  
    try:  
        while True:  
            s = (yield)
```

Example: print out strings that match a pattern

```
def match(pattern):  
    print('Looking for ' + pattern)  
    try:  
        while True:  
            s = (yield)  
            if pattern in s:
```

Example: print out strings that match a pattern

```
def match(pattern):  
    print('Looking for ' + pattern)  
    try:  
        while True:  
            s = (yield)  
            if pattern in s:  
                print(s)
```

Example: print out strings that match a pattern

```
def match(pattern):  
    print('Looking for ' + pattern)  
    try:  
        while True:  
            s = (yield)  
            if pattern in s:  
                print(s)  
    except GeneratorExit:
```

Example: print out strings that match a pattern

```
def match(pattern):
    print('Looking for ' + pattern)
    try:
        while True:
            s = (yield)
            if pattern in s:
                print(s)
    except GeneratorExit:
        print("=== Done ===")
```


Example: print out strings that match a pattern

```
def match(pattern):  
    print('Looking for ' + pattern)  
    try:  
        while True:  
            s = (yield)  
            if pattern in s:  
                print(s)  
    except GeneratorExit:  
        print("=== Done ===")
```

Step 1: Initialize

Example: print out strings that match a pattern

```
def match(pattern):
    print('Looking for ' + pattern)
    try:
        while True:
            s = (yield)
            if pattern in s:
                print(s)
    except GeneratorExit:
        print("=== Done ===")
```

Step 1: Initialize

```
>>> m = match("Jabberwock")
```

Example: print out strings that match a pattern

```
def match(pattern):  
    print('Looking for ' + pattern)  
    try:  
        while True:  
            s = (yield)  
            if pattern in s:  
                print(s)  
    except GeneratorExit:  
        print("=== Done ===")
```

Step 1: Initialize

```
>>> m = match("Jabberwock")
```



does nothing
creates a new object

Example: print out strings that match a pattern

```
def match(pattern):  
    print('Looking for ' + pattern)  
    try:  
        while True:  
            s = (yield)  
            if pattern in s:  
                print(s)  
    except GeneratorExit:  
        print("=== Done ===")
```

Step 1: Initialize

```
>>> m = match("Jabberwock")
```



does nothing
creates a new object

Step 2: Start with `__next__()`

Example: print out strings that match a pattern

```
def match(pattern):
    print('Looking for ' + pattern)
    try:
        while True:
            s = (yield)
            if pattern in s:
                print(s)
    except GeneratorExit:
        print("=== Done ===")
```

Step 1: Initialize

```
>>> m = match("Jabberwock")
```



does nothing
creates a new object

Step 2: Start with `__next__()`

```
>>> m.__next__()
```

Example: print out strings that match a pattern

```
def match(pattern):  
    print('Looking for ' + pattern) ← execution starts  
    try:  
        while True:  
            s = (yield)  
            if pattern in s:  
                print(s)  
    except GeneratorExit:  
        print("=== Done ===")
```

Step 1: Initialize

```
>>> m = match("Jabberwock")
```

← does nothing
creates a new object

Step 2: Start with `__next__()`

```
>>> m.__next__()
```

Example: print out strings that match a pattern

```
def match(pattern):  
    print('Looking for ' + pattern) ← execution starts  
    try:  
        while True:  
            s = (yield)  
            if pattern in s:  
                print(s)  
    except GeneratorExit:  
        print("=== Done ===")
```

Step 1: Initialize

```
>>> m = match("Jabberwock")
```



does nothing
creates a new object

Step 2: Start with `__next__()`

```
>>> m.__next__()
```

```
'Looking for Jabberwock'
```

Example: print out strings that match a pattern

```
def match(pattern):  
    print('Looking for ' + pattern) ← execution starts  
    try:  
        while True:  
            s = (yield) ← stops here, waiting  
                for data  
            if pattern in s:  
                print(s)  
    except GeneratorExit:  
        print("=== Done ===")
```

Step 1: Initialize

```
>>> m = match("Jabberwock")
```

← does nothing
creates a new object

Step 2: Start with `__next__()`

```
>>> m.__next__()
```

```
'Looking for Jabberwock'
```


Example: print out strings that match a pattern

```
def match(pattern):  
    print('Looking for ' + pattern) ← execution starts  
    try:  
        while True:  
            s = (yield) ← stops here, waiting  
                for data  
            if pattern in s:  
                print(s)  
    except GeneratorExit:  
        print("=== Done ===")
```

Step 1: Initialize

```
>>> m = match("Jabberwock") ← does nothing  
                                creates a new object
```

Step 2: Start with `__next__()`

```
>>> m.__next__()
```

```
'Looking for Jabberwock'
```

Step 3: Send data

Example: print out strings that match a pattern

```
def match(pattern):  
    print('Looking for ' + pattern) ← execution starts  
    try:  
        while True:  
            s = (yield) ← stops here, waiting  
                for data  
            if pattern in s:  
                print(s)  
    except GeneratorExit:  
        print("=== Done ===")
```

Step 1: Initialize

```
>>> m = match("Jabberwock")
```

← does nothing
creates a new object

Step 2: Start with `__next__()`

```
>>> m.__next__()
```

```
'Looking for Jabberwock'
```

Step 3: Send data

```
>>> m.send("the Jabberwock with eyes of flame")
```

Example: print out strings that match a pattern

```
def match(pattern):  
    print('Looking for ' + pattern) ← execution starts  
    try:  
        while True:  
            s = (yield)  
            if pattern in s:  
                print(s)  
    except GeneratorExit:  
        print("=== Done ===")
```

Step 1: Initialize

```
>>> m = match("Jabberwock")
```



does nothing
creates a new object

Step 2: Start with `__next__()`

```
>>> m.__next__()
```

```
'Looking for Jabberwock'
```

Step 3: Send data

```
>>> m.send("the Jabberwock with eyes of flame")
```

Example: print out strings that match a pattern

```
def match(pattern):  
    print('Looking for ' + pattern) ← execution starts  
    try:  
        while True:  
            s = (yield) ← resumes here  
            if pattern in s:  
                print(s)  
    except GeneratorExit:  
        print("=== Done ===")
```

Step 1: Initialize

```
>>> m = match("Jabberwock")
```



does nothing
creates a new object

Step 2: Start with `__next__()`

```
>>> m.__next__()
```

```
'Looking for Jabberwock'
```

Step 3: Send data

```
>>> m.send("the Jabberwock with eyes of flame")
```

Example: print out strings that match a pattern

```
def match(pattern):  
    print('Looking for ' + pattern) ← execution starts  
    try:  
        while True:  
            s = (yield) ← resumes here  
            if pattern in s:  
                print(s) ← match found  
    except GeneratorExit:  
        print("=== Done ===")
```

Step 1: Initialize

```
>>> m = match("Jabberwock") ← does nothing  
                                creates a new object
```

Step 2: Start with `__next__()`

```
>>> m.__next__()
```

```
'Looking for Jabberwock'
```

Step 3: Send data

```
>>> m.send("the Jabberwock with eyes of flame")
```

Example: print out strings that match a pattern

```
def match(pattern):  
    print('Looking for ' + pattern) ← execution starts  
    try:  
        while True:  
            s = (yield) ← resumes here  
            if pattern in s:  
                print(s) ← match found  
    except GeneratorExit:  
        print("=== Done ===")
```

Step 1: Initialize

```
>>> m = match("Jabberwock") ← does nothing  
                                creates a new object
```

Step 2: Start with `__next__()`

```
>>> m.__next__()
```

```
'Looking for Jabberwock'
```

Step 3: Send data

```
>>> m.send("the Jabberwock with eyes of flame")
```

```
'the Jabberwock with eyes of flame'
```

Example: print out strings that match a pattern

```
def match(pattern):  
    print('Looking for ' + pattern) ← execution starts  
    try:  
        while True:  
            s = (yield) ← resumes here  
            if pattern in s: ← s = "the Jabberwock ..."  
                print(s) ← match found  
    except GeneratorExit:  
        print("=== Done ===")
```

Step 1: Initialize

```
>>> m = match("Jabberwock")
```

← does nothing
creates a new object

Step 2: Start with `__next__()`

```
>>> m.__next__()
```

```
'Looking for Jabberwock'
```

Step 3: Send data

```
>>> m.send("the Jabberwock with eyes of flame")
```

```
'the Jabberwock with eyes of flame'
```

Step 4: close the coroutine

Example: print out strings that match a pattern

```
def match(pattern):  
    print('Looking for ' + pattern) ← execution starts  
    try:  
        while True:  
            s = (yield) ← resumes here  
            if pattern in s: ← s = "the Jabberwock ..."  
                print(s) ← match found  
    except GeneratorExit:  
        print("=== Done ===")
```

Step 1: Initialize

```
>>> m = match("Jabberwock") ← does nothing  
                                creates a new object
```

Step 2: Start with `__next__()`

```
>>> m.__next__()
```

```
'Looking for Jabberwock'
```

Step 3: Send data

```
>>> m.send("the Jabberwock with eyes of flame")
```

```
'the Jabberwock with eyes of flame'
```

Step 4: close the coroutine

```
>>> m.close()
```


Example: print out strings that match a pattern

```
def match(pattern):  
    print('Looking for ' + pattern) ← execution starts  
    try:  
        while True:  
            s = (yield) ← resumes here  
            if pattern in s: ← s = "the Jabberwock ..."  
                print(s) ← match found  
    except GeneratorExit: ← catch exception  
        print("=== Done ===")
```

Step 1: Initialize

```
>>> m = match("Jabberwock") ← does nothing  
                                creates a new object
```

Step 2: Start with `__next__()`

```
>>> m.__next__()
```

```
'Looking for Jabberwock'
```

Step 3: Send data

```
>>> m.send("the Jabberwock with eyes of flame")
```

```
'the Jabberwock with eyes of flame'
```

Step 4: close the coroutine

```
>>> m.close()
```

Example: print out strings that match a pattern

```
def match(pattern):  
    print('Looking for ' + pattern) ← execution starts  
    try:  
        while True:  
            s = (yield) ← resumes here  
            if pattern in s: ← s = "the Jabberwock ..."  
                print(s) ← match found  
    except GeneratorExit: ← catch exception  
        print("=== Done ===")
```

Step 1: Initialize

```
>>> m = match("Jabberwock") ← does nothing  
                                creates a new object
```

Step 2: Start with `__next__()`

```
>>> m.__next__()
```

```
'Looking for Jabberwock'
```

Step 3: Send data

```
>>> m.send("the Jabberwock with eyes of flame")
```

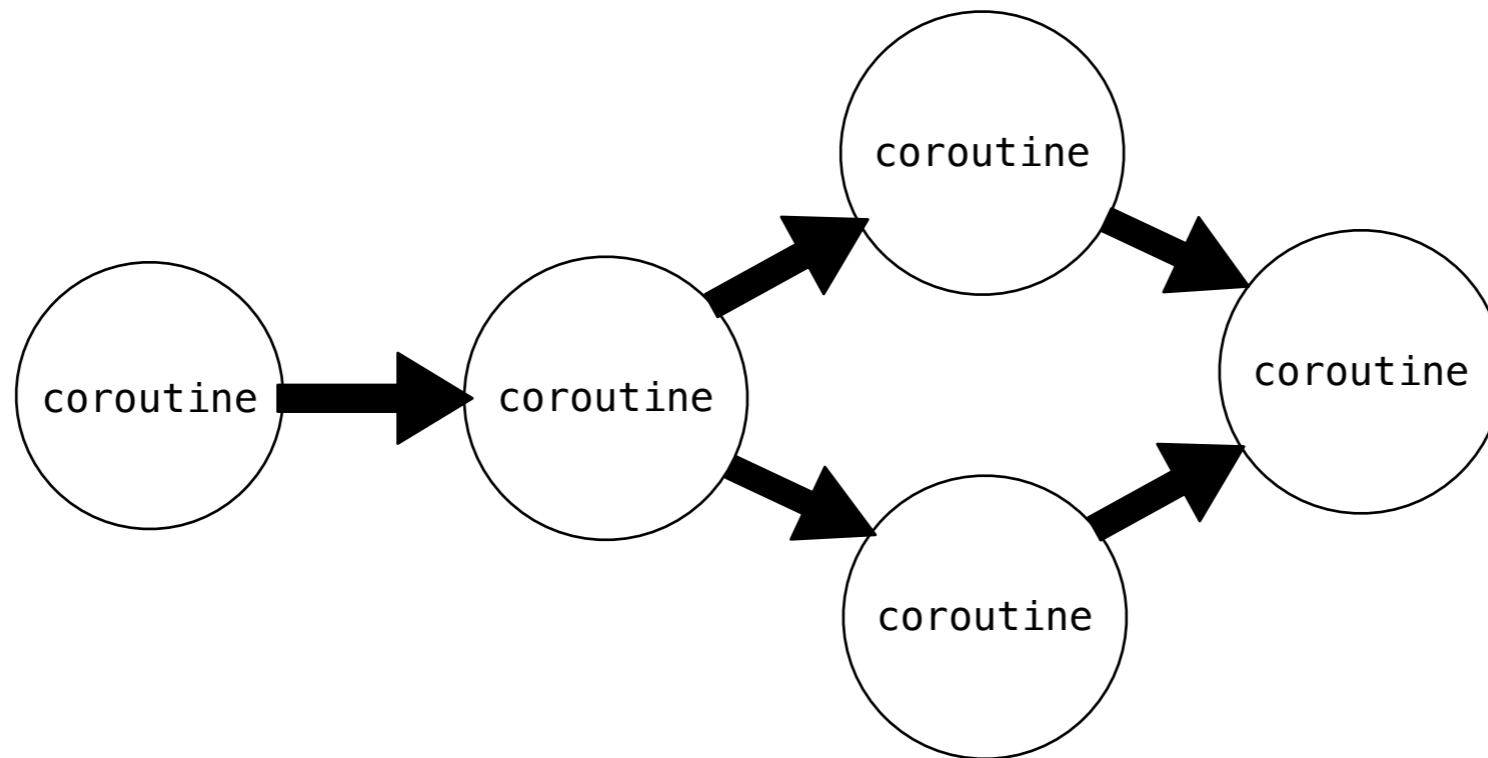
```
'the Jabberwock with eyes of flame'
```

Step 4: close the coroutine

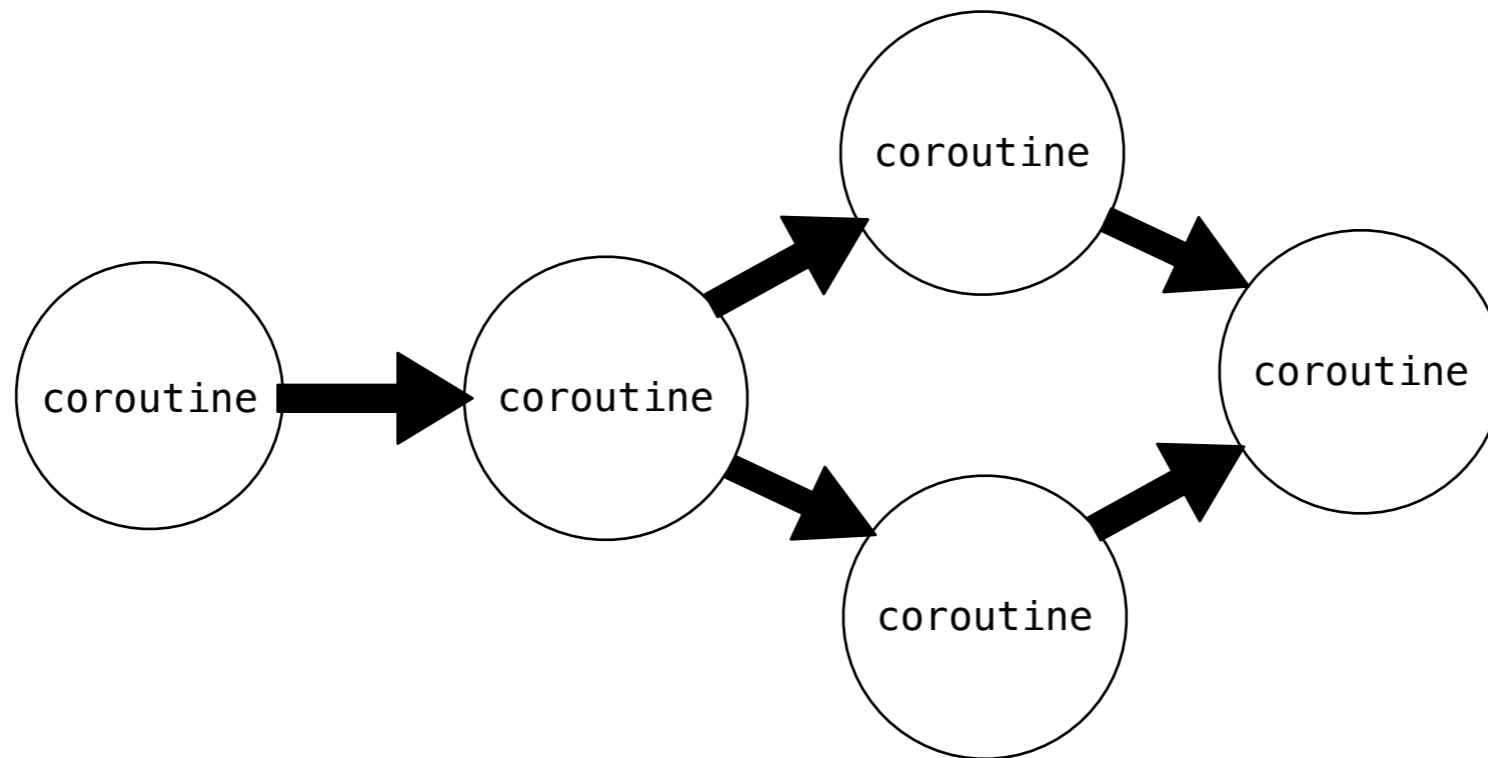
```
>>> m.close()
```

```
'=== Done ==='
```

Pipelines: the power of coroutines

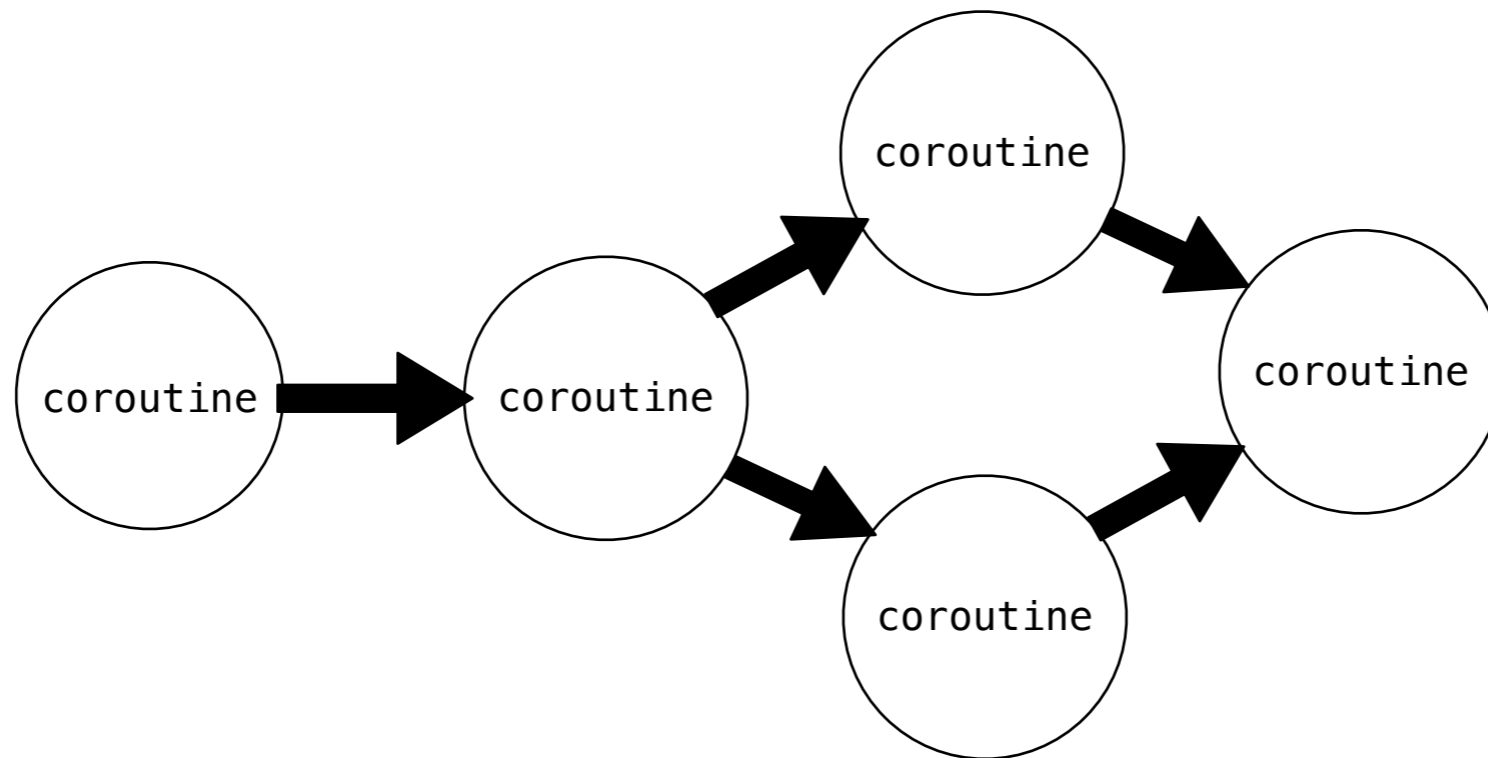


Pipelines: the power of coroutines



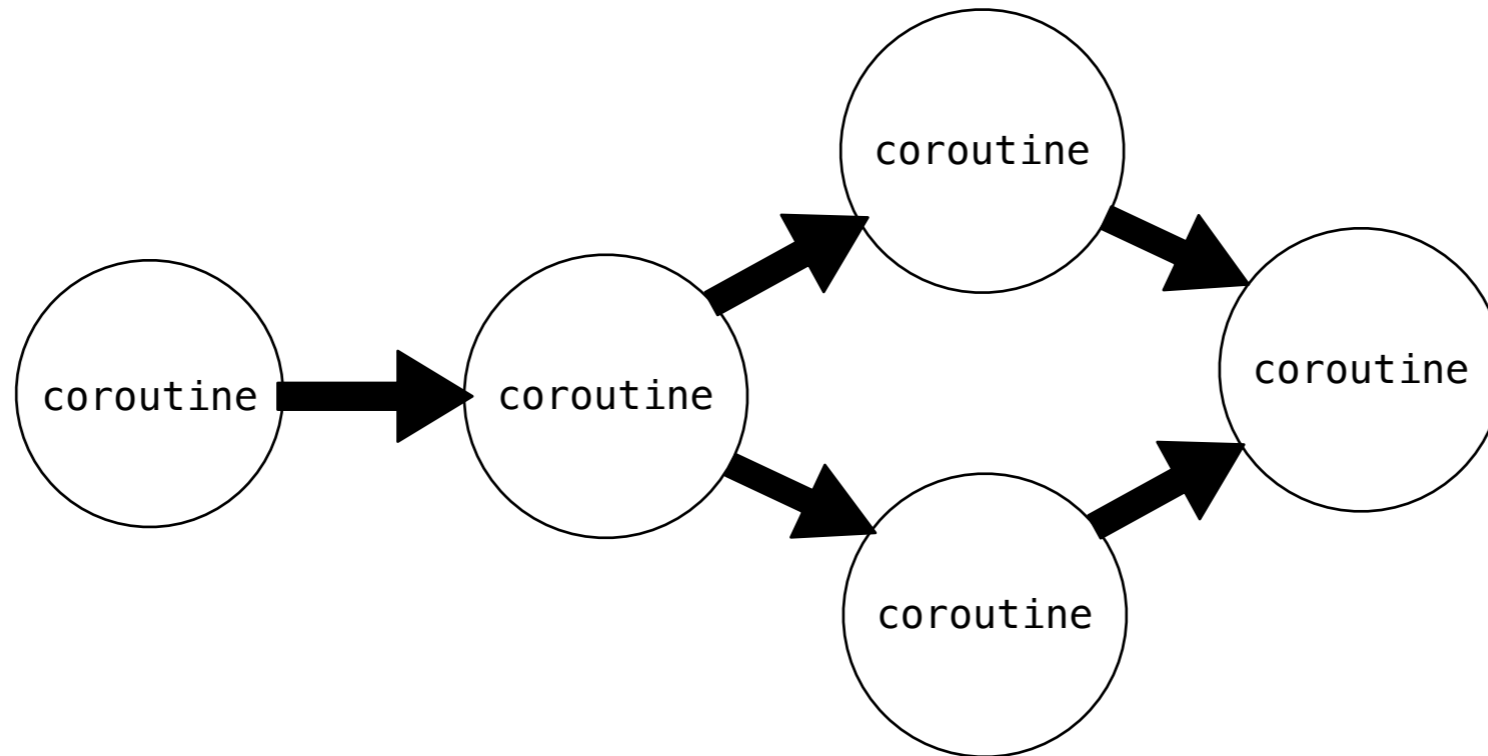
We can chain coroutines together to achieve complex behaviors

Pipelines: the power of coroutines



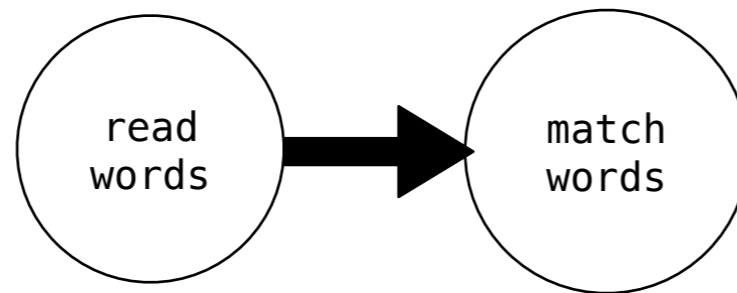
We can chain coroutines together to achieve complex behaviors
Create a **pipeline**

Pipelines: the power of coroutines

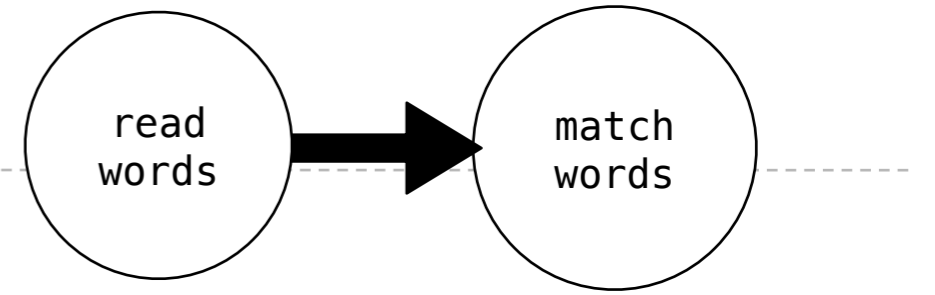


We can chain coroutines together to achieve complex behaviors
Create a **pipeline**
Coroutines send data to others downstream

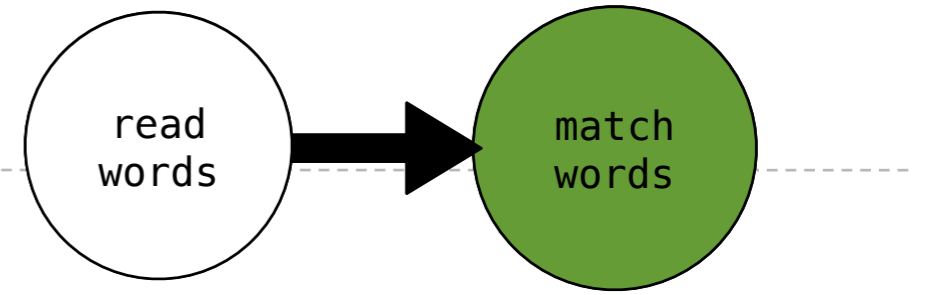
A simple pipeline



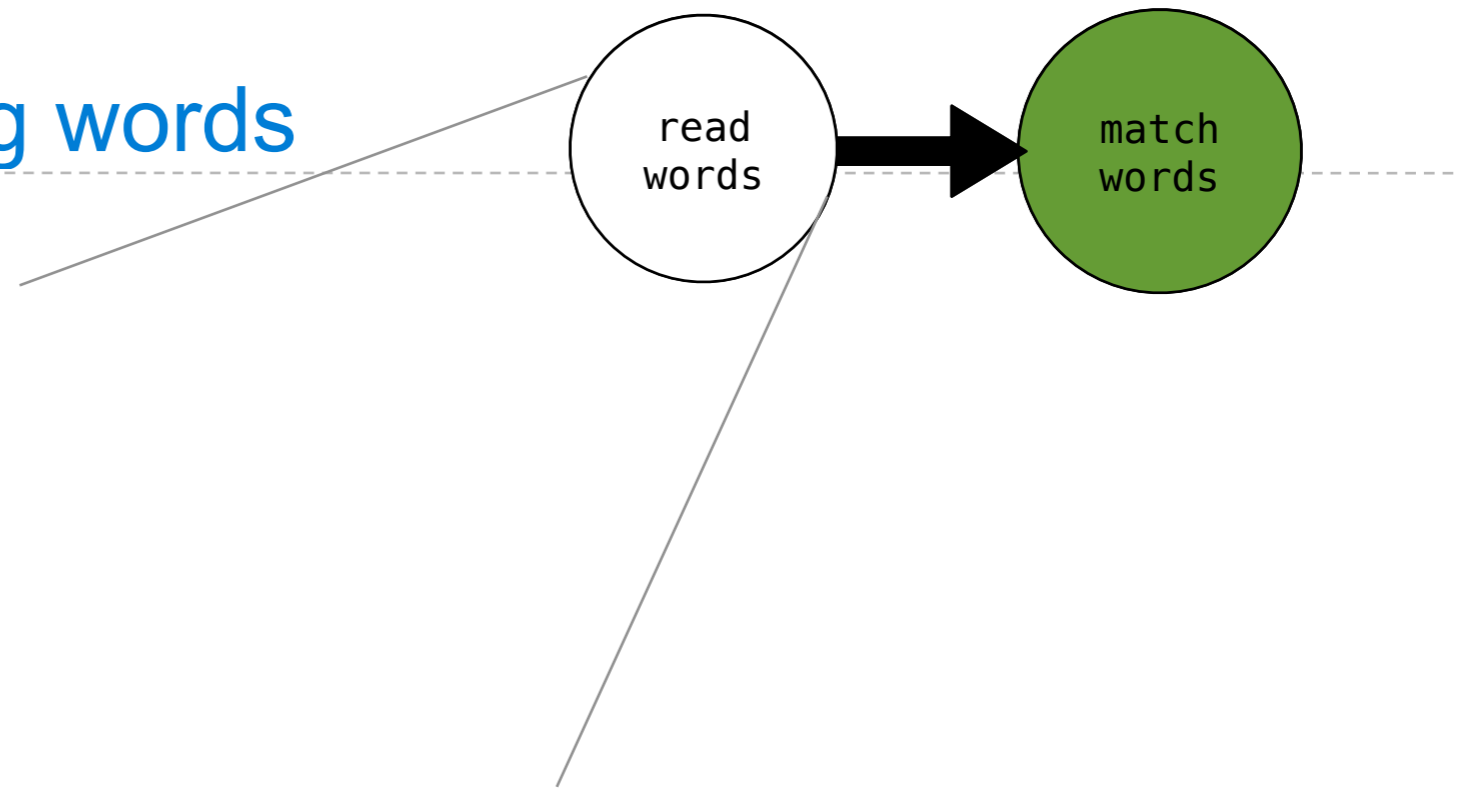
A simple pipeline: reading words



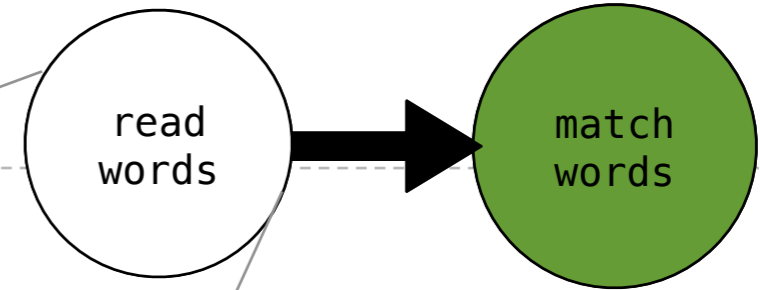
A simple pipeline: reading words



A simple pipeline: reading words

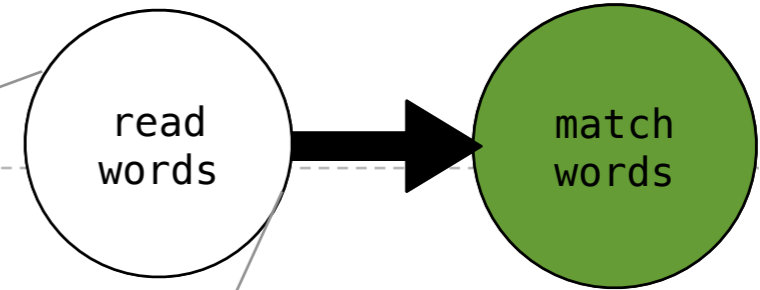


A simple pipeline: reading words



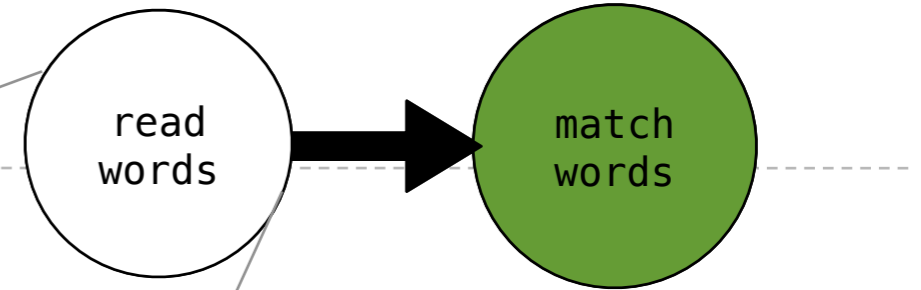
```
def read(text, next_coroutine):
```

A simple pipeline: reading words



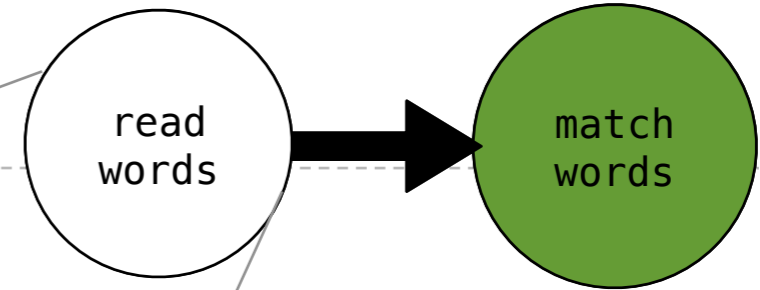
```
def read(text, next_coroutine):  
    for word in text.split():
```

A simple pipeline: reading words



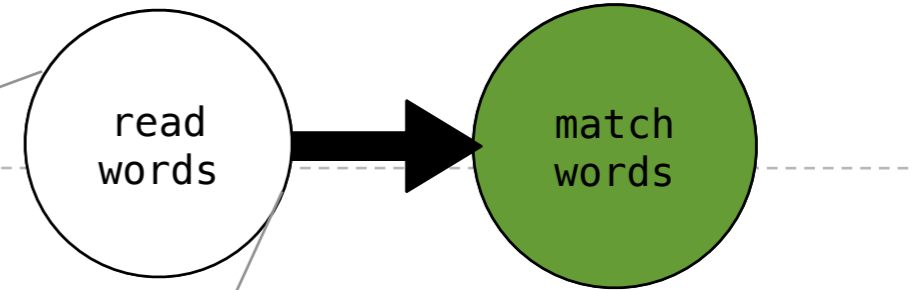
```
def read(text, next_coroutine):  
    for word in text.split():  
        next_coroutine.send(word)
```

A simple pipeline: reading words



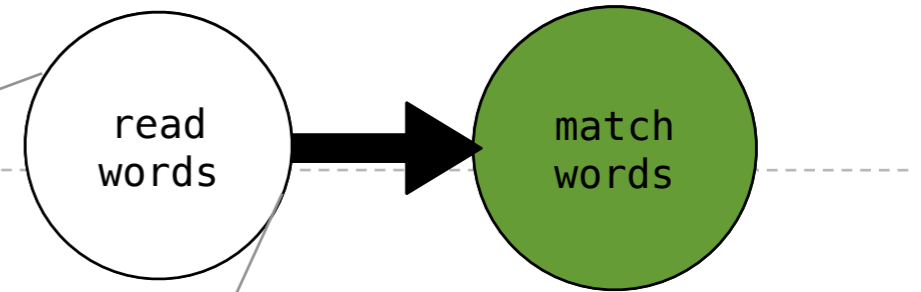
```
def read(text, next_coroutine):  
    for word in text.split():  
        next_coroutine.send(word)  
    next_coroutine.close()
```

A simple pipeline: reading words



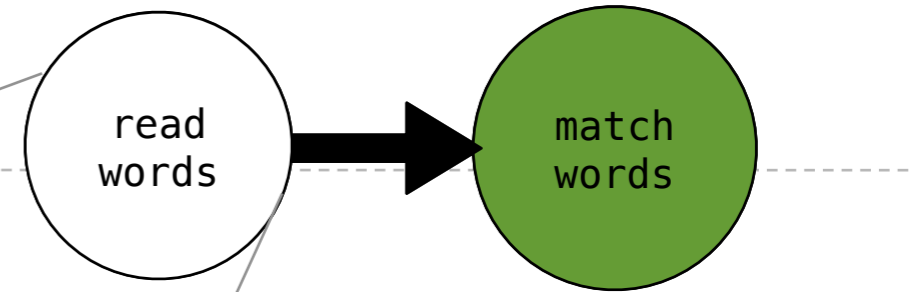
```
def read(text, next_coroutine):  
    for word in text.split():  
        next_coroutine.send(word)  
    next_coroutine.close()
```

A simple pipeline: reading words



```
def read(text, next_coroutine): needs to know where to send()
    for word in text.split():
        next_coroutine.send(word)
    next_coroutine.close()
```


A simple pipeline: reading words

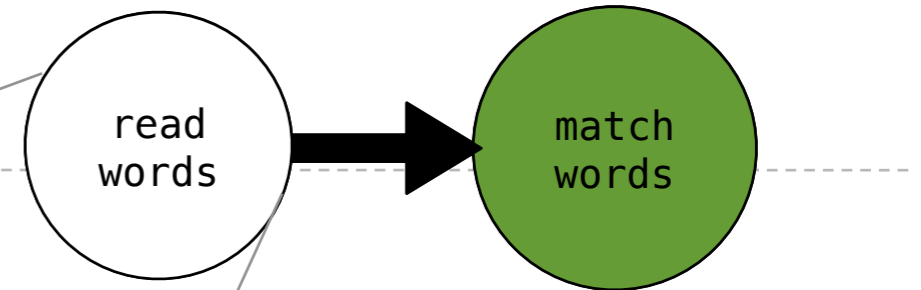


```
def read(text, next_coroutine): needs to know where to send()
    for word in text.split():
        next_coroutine.send(word)
    next_coroutine.close()
```

read

```
for word in text.split():
    next_coroutine.send(word)
```

A simple pipeline: reading words



```
def read(text, next_coroutine): needs to know where to send()
    for word in text.split():
        next_coroutine.send(word)
    next_coroutine.close()
```

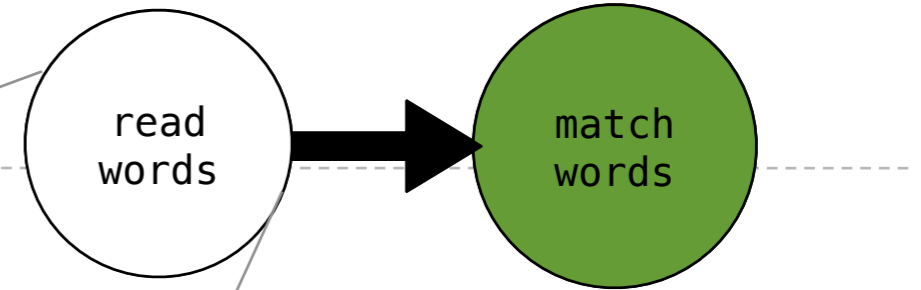
read

```
for word in text.split():
    next_coroutine.send(word)
```

send -- activate (yield)



A simple pipeline: reading words



```
def read(text, next_coroutine): needs to know where to send()
    for word in text.split():
        next_coroutine.send(word)
    next_coroutine.close()
```

read

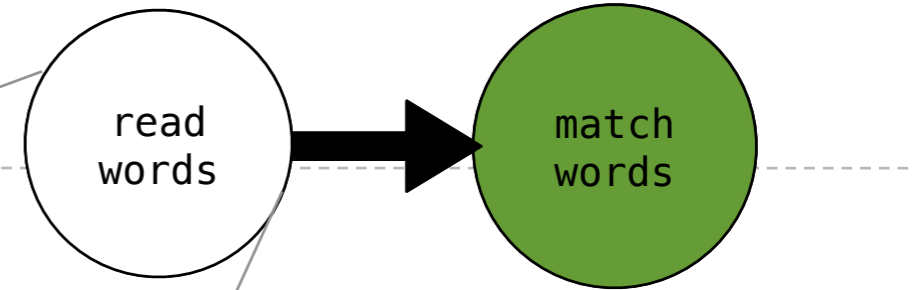
```
for word in text.split():
    next_coroutine.send(word)
```

send -- activate (yield)

value = (yield)

next_coroutine

A simple pipeline: reading words



```
def read(text, next_coroutine): needs to know where to send()  
    for word in text.split():  
        next_coroutine.send(word)  
    next_coroutine.close()
```

read

```
for word in text.split():  
    next_coroutine.send(word)
```

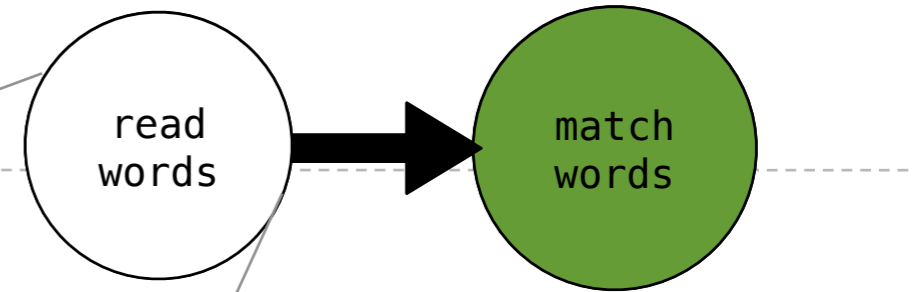
send -- activate (yield)

value = (yield)

loop

next_coroutine

A simple pipeline: reading words



```
def read(text, next_coroutine): needs to know where to send()
    for word in text.split():
        next_coroutine.send(word)
    next_coroutine.close()
```

read

```
for word in text.split():
    next_coroutine.send(word)
```

send -- activate (yield)

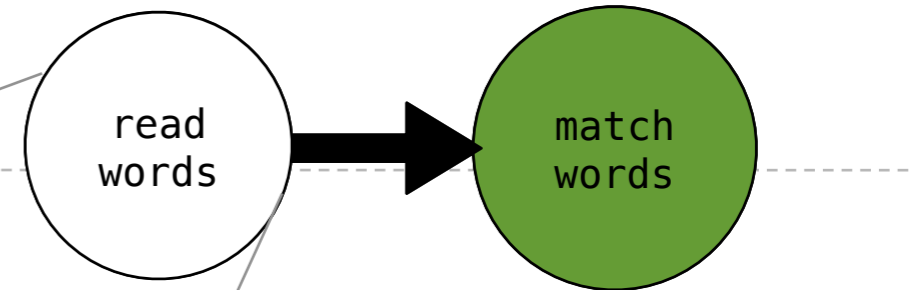
value = (yield)

(yield) -- wait for next send

loop

next_coroutine

A simple pipeline: reading words



```
def read(text, next_coroutine): needs to know where to send()  
    for word in text.split():  
        next_coroutine.send(word)  
    next_coroutine.close()
```

read

```
for loop ↻ for word in text.split():  
    next_coroutine.send(word)
```

(yield) -- wait for next send

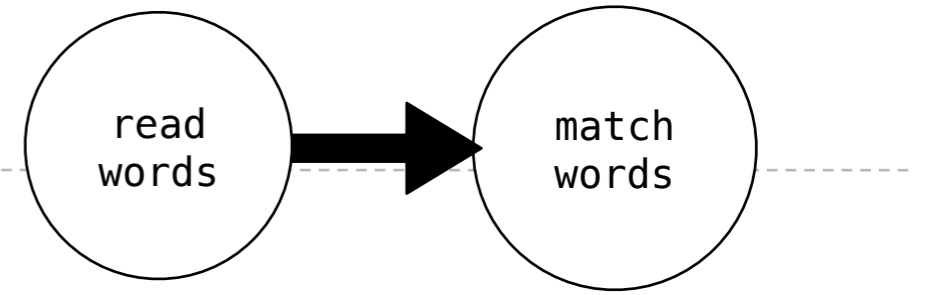
send -- activate (yield)

value = (yield)

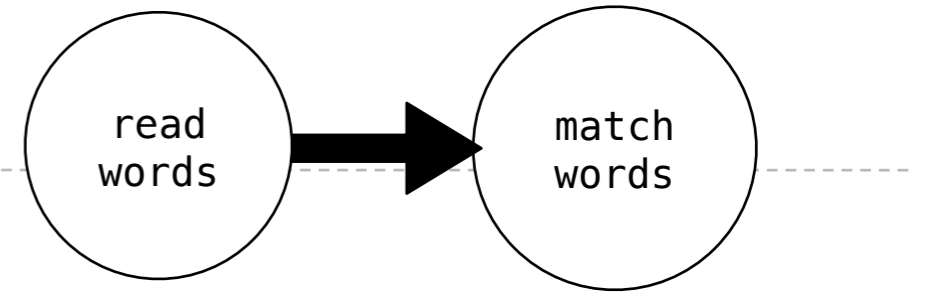
loop ↻

next_coroutine

A simple pipeline



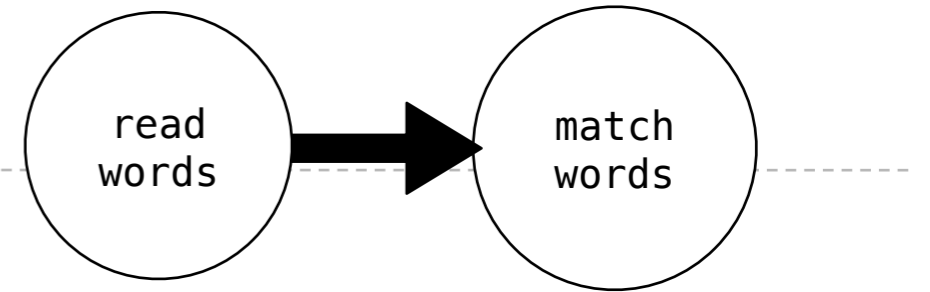
A simple pipeline



read

```
for word in text.split():  
    next_coroutine.send(word)
```


A simple pipeline



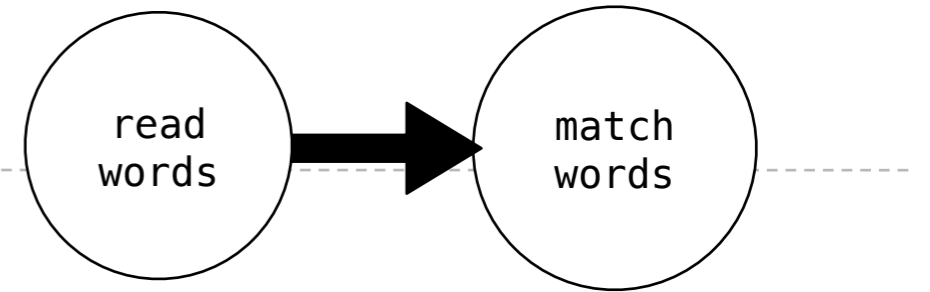
read

```
for word in text.split():  
    next_coroutine.send(word)
```

send -- activate (yield)



A simple pipeline



read

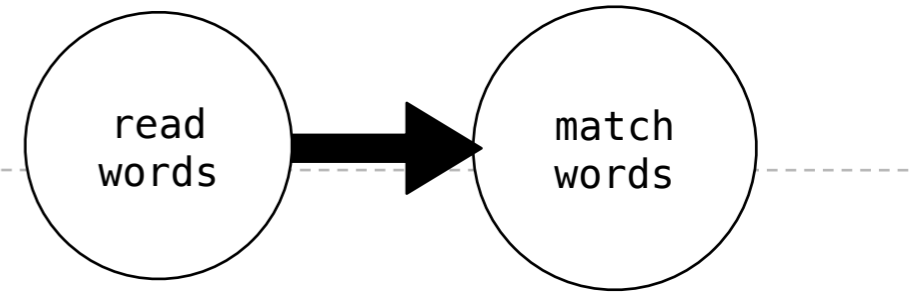
```
for word in text.split():  
    next_coroutine.send(word)
```

send -- activate (yield)

```
while True:  
    line = (yield)  
    if pattern in line:  
        print(line)
```

match

A simple pipeline



read

```
for word in text.split():  
    next_coroutine.send(word)
```

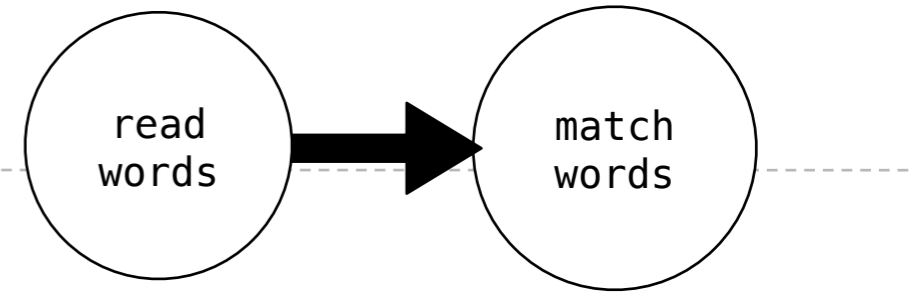
send -- activate (yield)

```
while True:  
    line = (yield)  
    if pattern in line:  
        print(line)
```

while loop

match

A simple pipeline



read

```
for word in text.split():  
    next_coroutine.send(word)
```

send -- activate (yield)

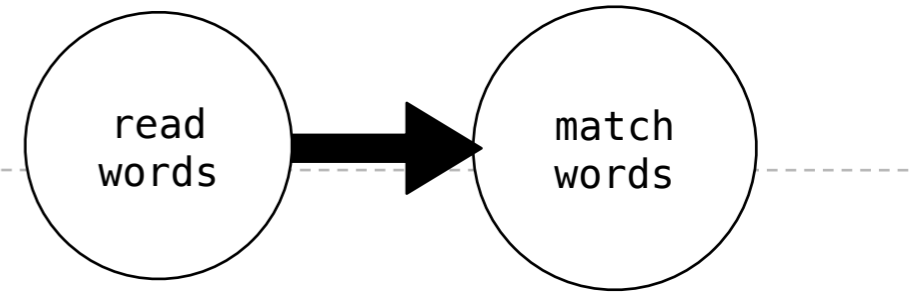
```
while True:  
    line = (yield)  
    if pattern in line:  
        print(line)
```

match


(yield) -- wait for next send

while loop

A simple pipeline



read

```
for loop  for word in text.split():  
    next_coroutine.send(word)
```

(yield) -- wait for next send

send -- activate (yield)

```
while True:  
    line = (yield)  
    if pattern in line:  
        print(line)
```

while loop

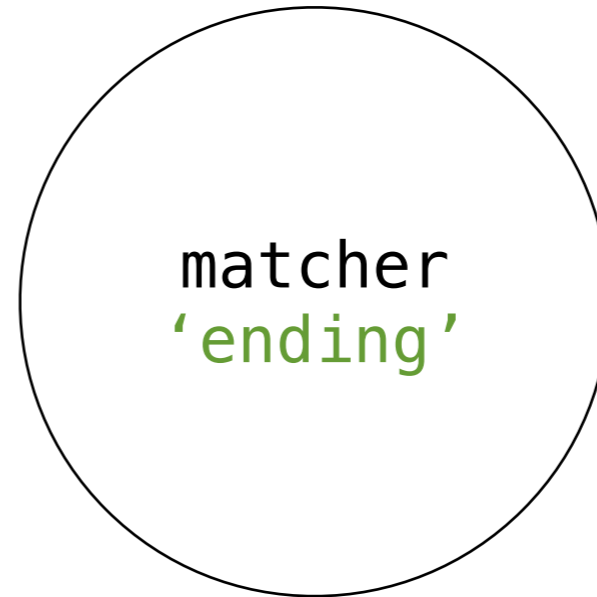
match

A simple pipeline

A simple pipeline

```
>>> matcher = match('ending')
```

A simple pipeline



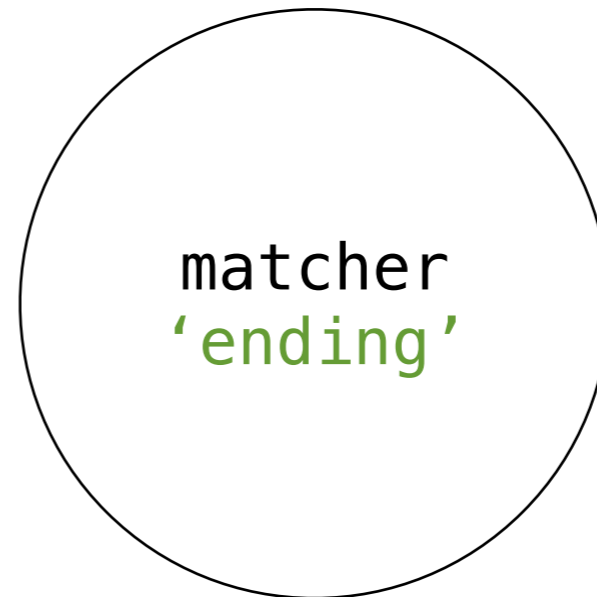
```
>>> matcher = match('ending')
```


A simple pipeline



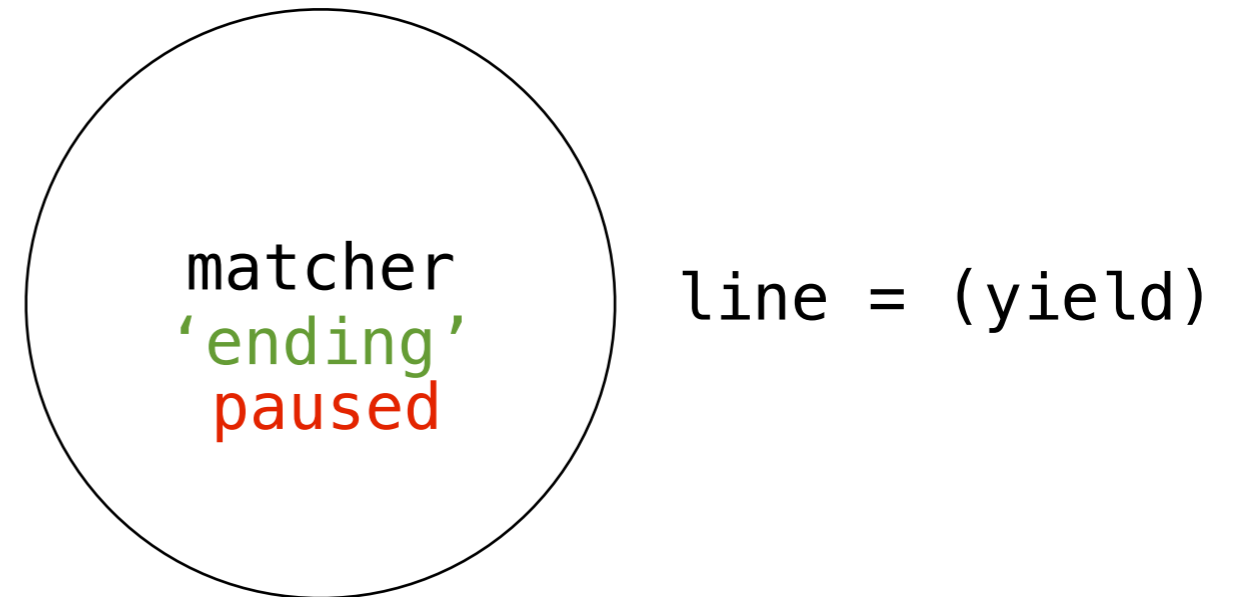
```
>>> matcher = match('ending')  
>>> matcher.__next__()
```

A simple pipeline



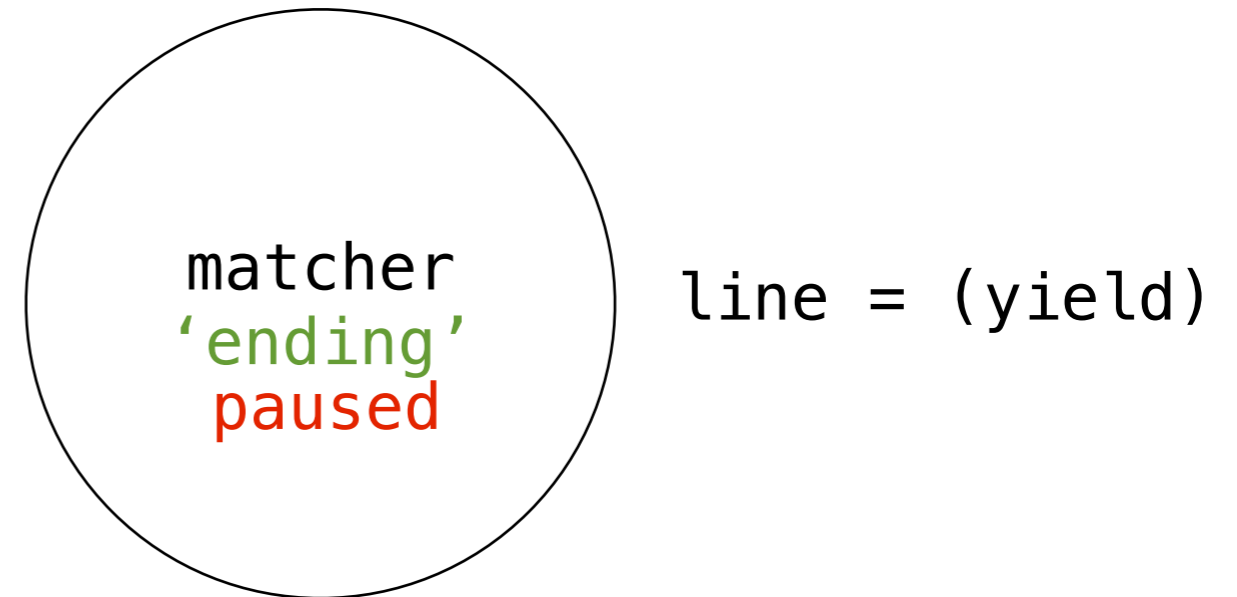
```
>>> matcher = match('ending')
>>> matcher.__next__()
'Looking for ending'
```

A simple pipeline



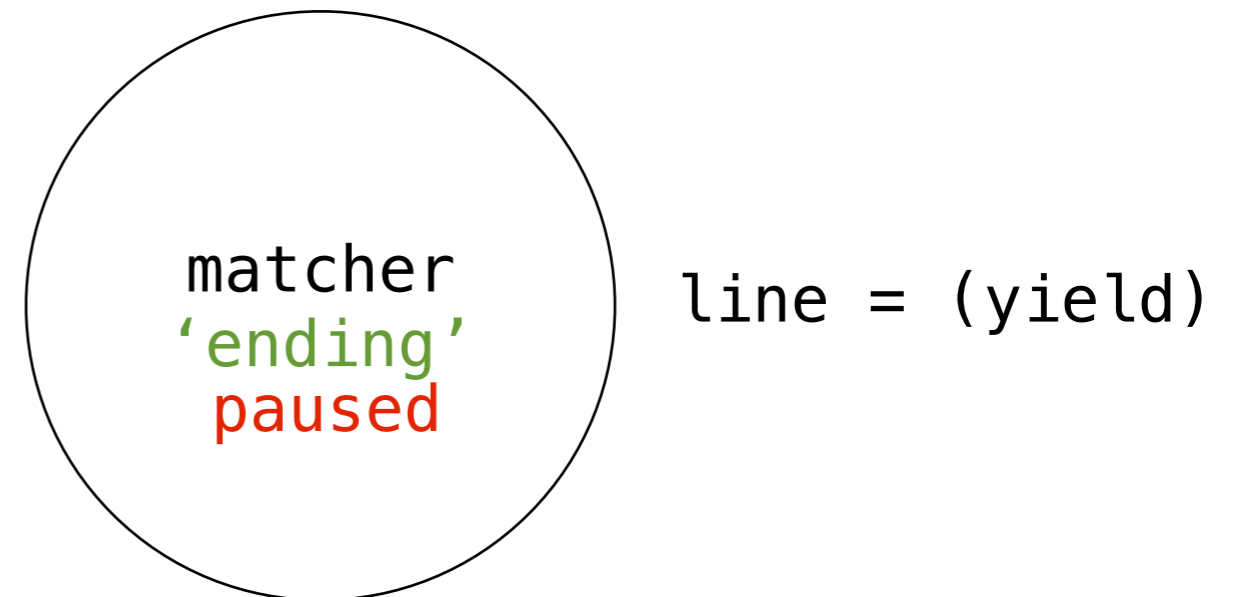
```
>>> matcher = match('ending')
>>> matcher.__next__()
'Looking for ending'
```

A simple pipeline



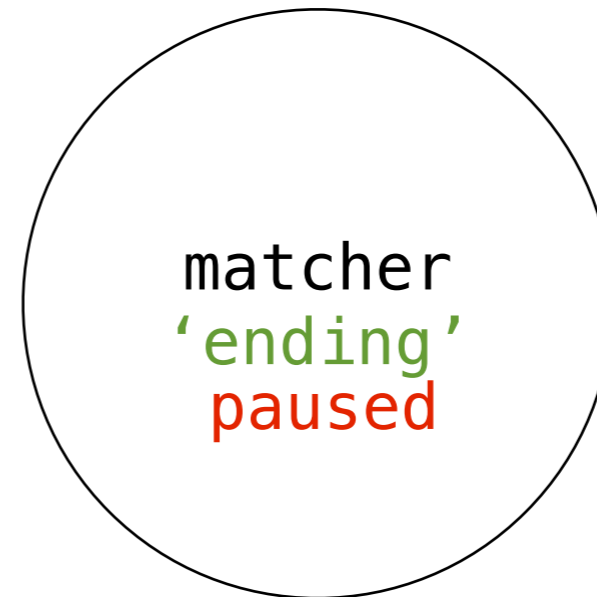
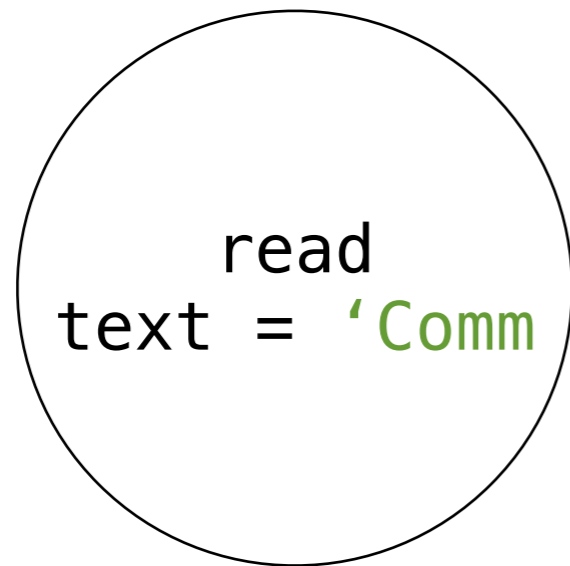
```
>>> matcher = match('ending')
>>> matcher.__next__()
'Looking for ending'
>>> text = 'Commending spending is offending to people pending lending!'
```

A simple pipeline



```
>>> matcher = match('ending')
>>> matcher.__next__()
'Looking for ending'
>>> text = 'Commending spending is offending to people pending lending!'
>>> read(text, matcher)
```

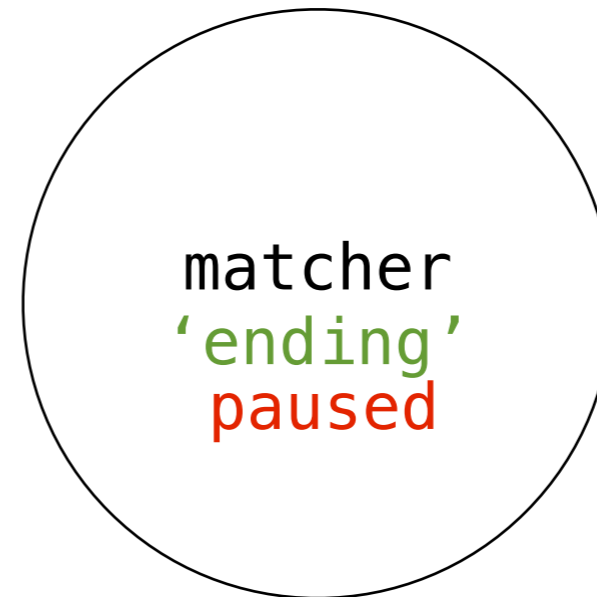
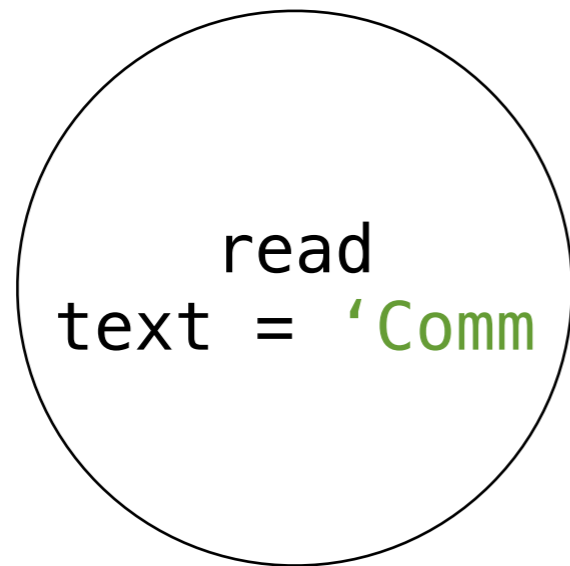
A simple pipeline



line = (yield)

```
>>> matcher = match('ending')
>>> matcher.__next__()
'Looking for ending'
>>> text = 'Commending spending is offending to people pending lending!'
>>> read(text, matcher)
```

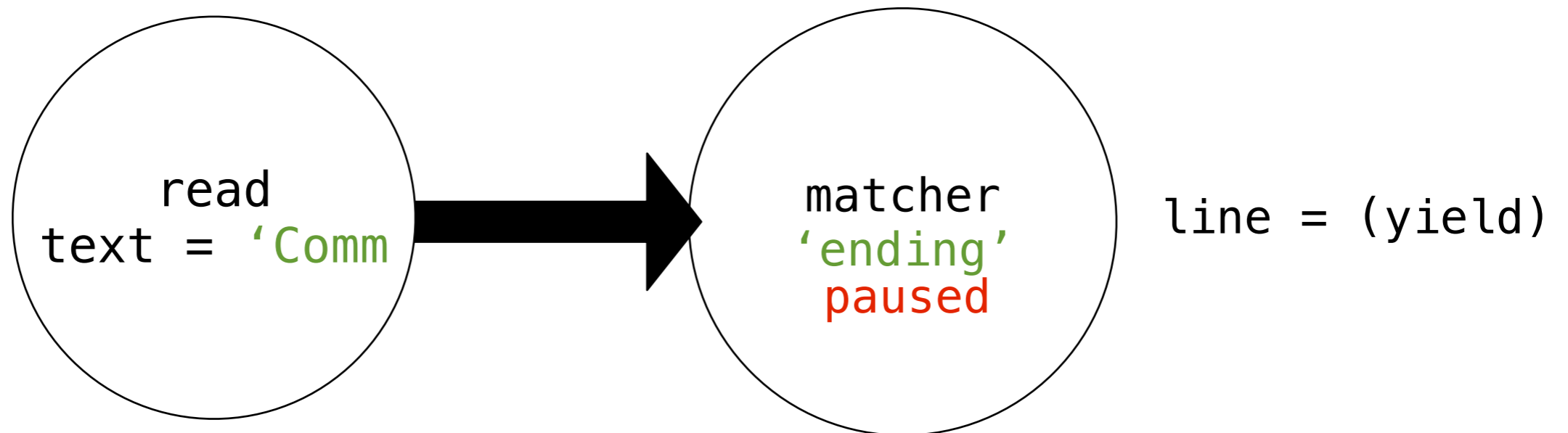
A simple pipeline



line = (yield)

```
>>> matcher = match('ending')
>>> matcher.__next__()
'Looking for ending'
>>> text = 'Commending spending is offending to people pending lending!'
>>> read(text, matcher)
```

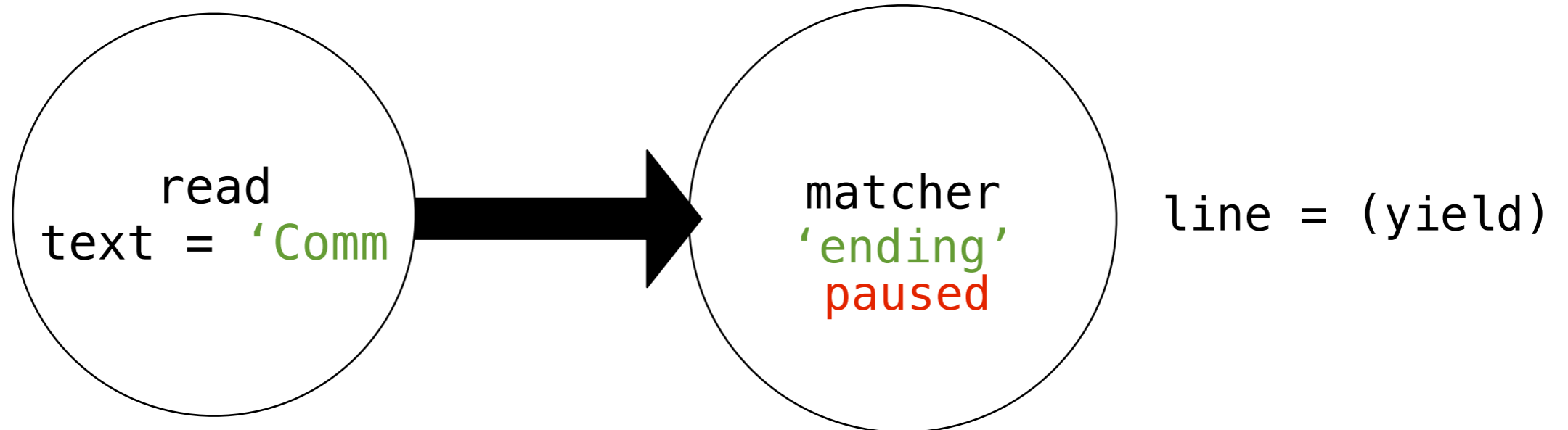
A simple pipeline



```
>>> matcher = match('ending')
>>> matcher.__next__()
'Looking for ending'
>>> text = 'Commending spending is offending to people pending lending!'
>>> read(text, matcher)
```


A simple pipeline

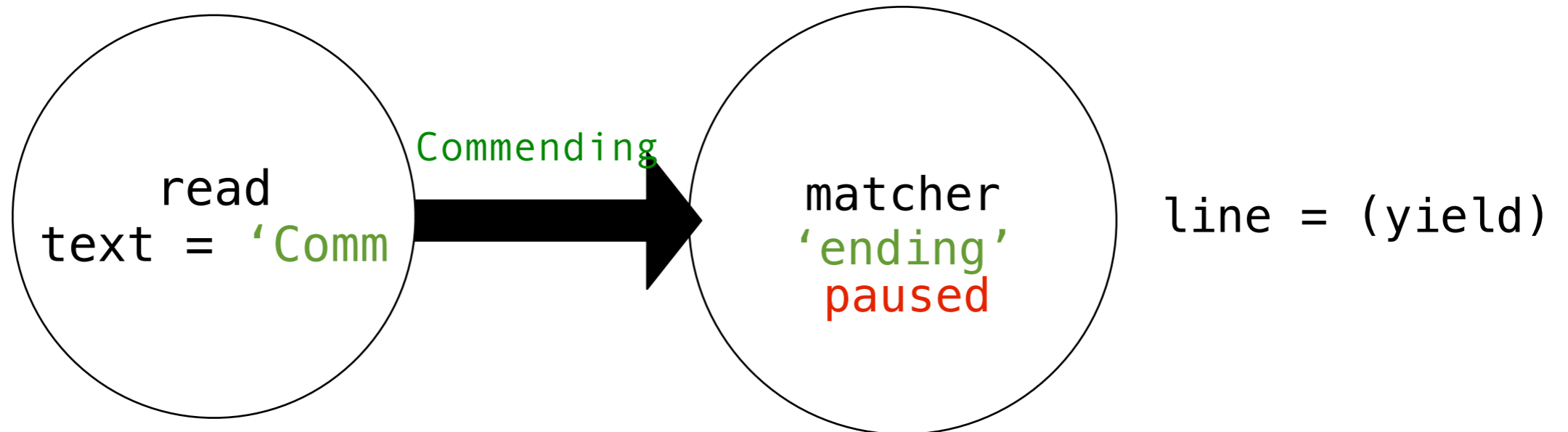
```
for word in text.split():  
    next_coroutine.send(word)
```



```
>>> matcher = match('ending')  
>>> matcher.__next__()  
'Looking for ending'  
>>> text = 'Commending spending is offending to people pending lending!'  
>>> read(text, matcher)
```

A simple pipeline

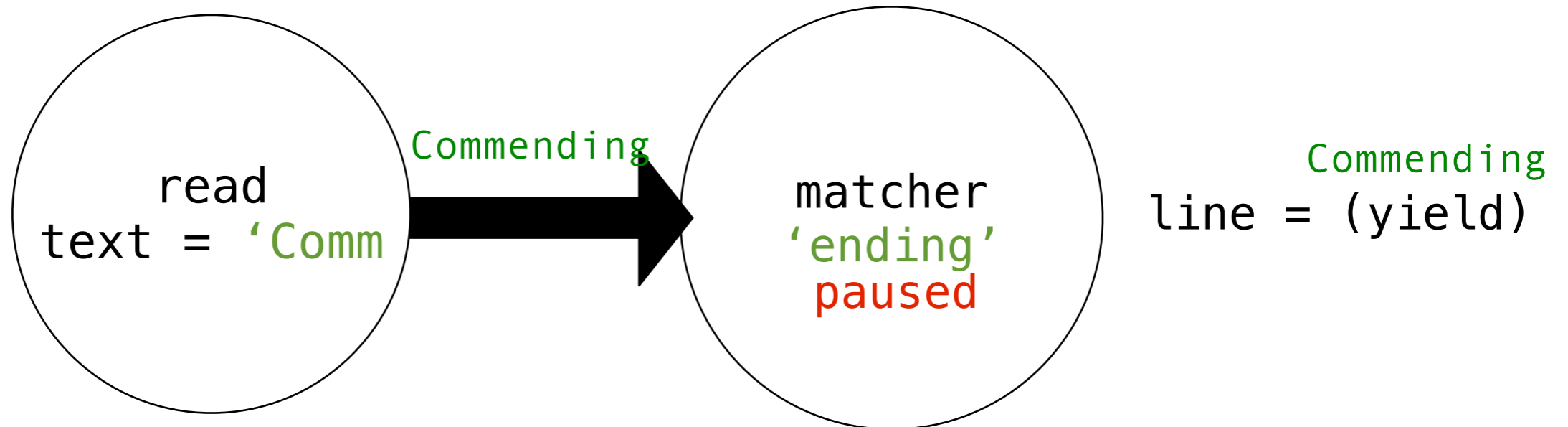
```
for word in text.split():  
    next_coroutine.send(word)
```



```
>>> matcher = match('ending')  
>>> matcher.__next__()  
'Looking for ending'  
>>> text = 'Commending spending is offending to people pending lending!'  
>>> read(text, matcher)
```

A simple pipeline

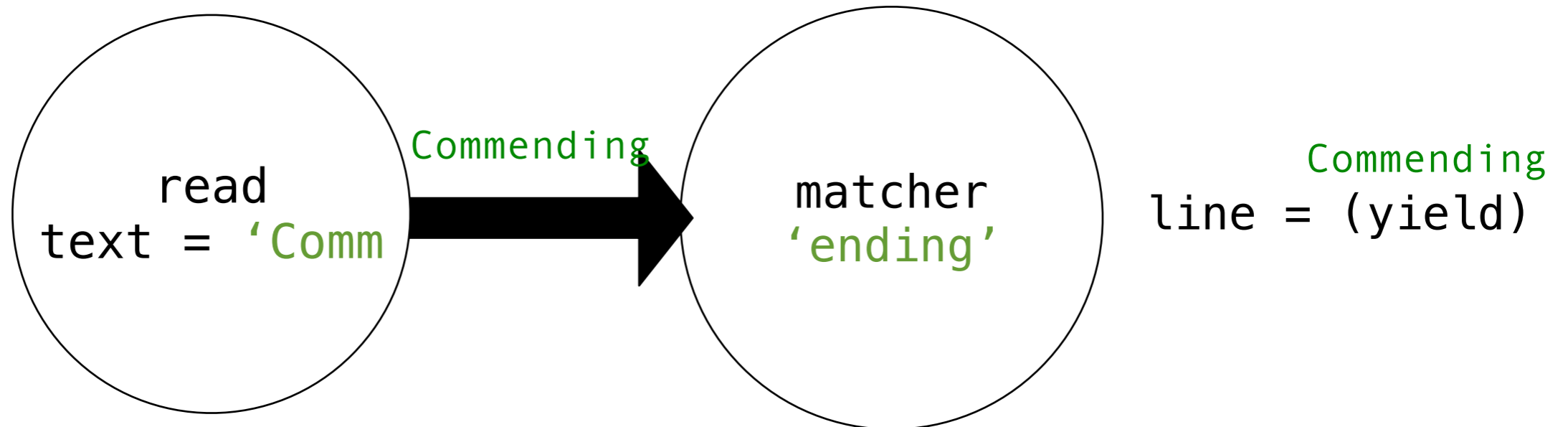
```
for word in text.split():  
    next_coroutine.send(word)
```



```
>>> matcher = match('ending')  
>>> matcher.__next__()  
'Looking for ending'  
>>> text = 'Commending spending is offending to people pending lending!'  
>>> read(text, matcher)
```

A simple pipeline

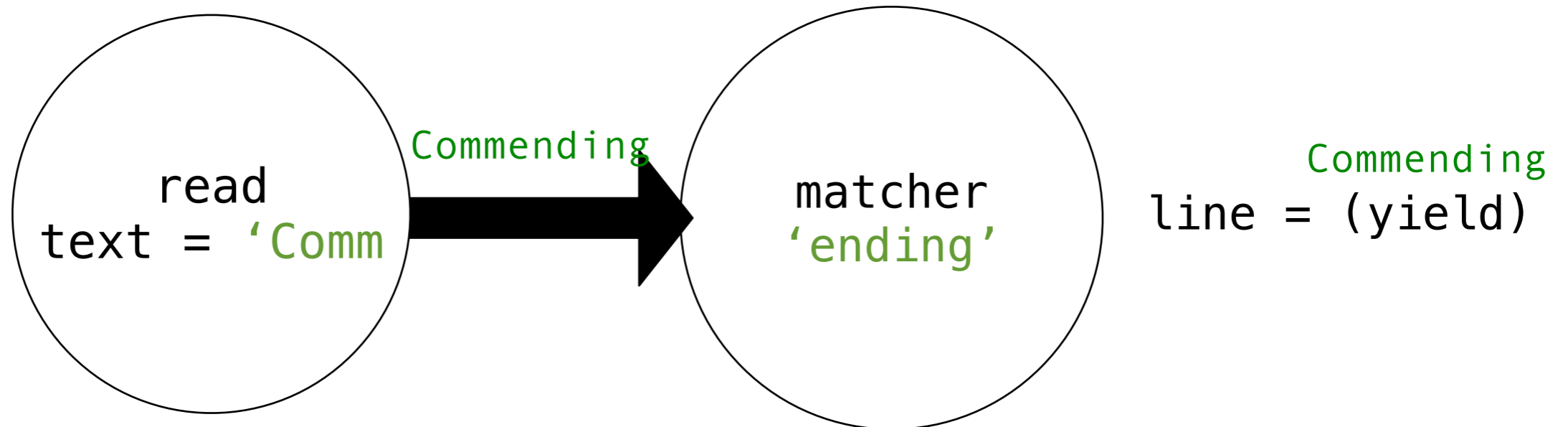
```
for word in text.split():  
    next_coroutine.send(word)
```



```
>>> matcher = match('ending')  
>>> matcher.__next__()  
'Looking for ending'  
>>> text = 'Commending spending is offending to people pending lending!'  
>>> read(text, matcher)
```

A simple pipeline

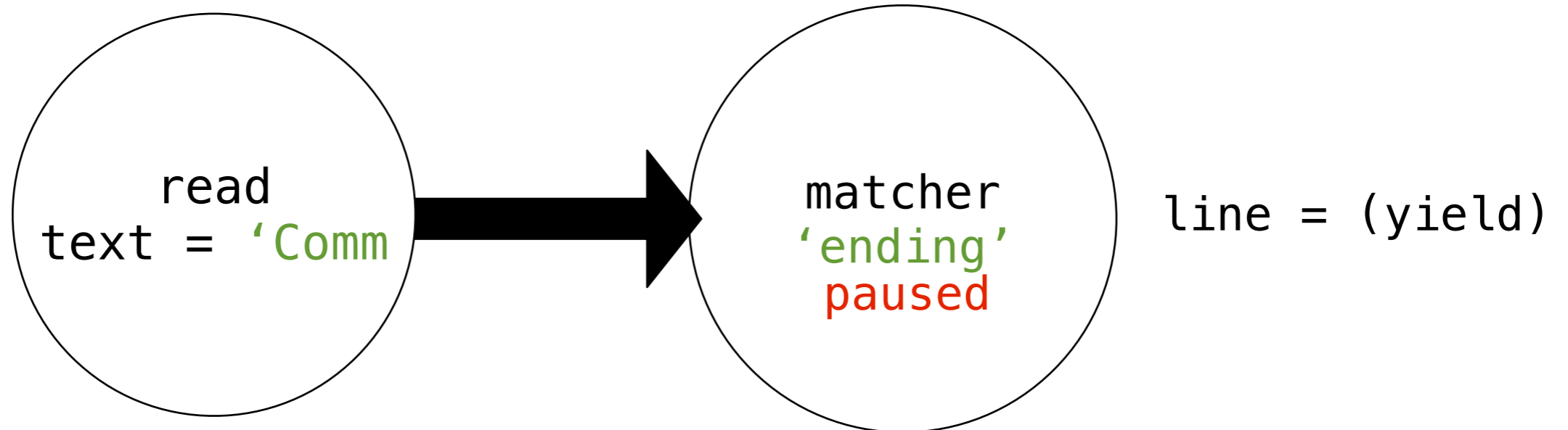
```
for word in text.split():  
    next_coroutine.send(word)
```



```
>>> matcher = match('ending')  
>>> matcher.__next__()  
'Looking for ending'  
>>> text = 'Commending spending is offending to people pending lending!'  
>>> read(text, matcher)  
'Commending'
```

A simple pipeline

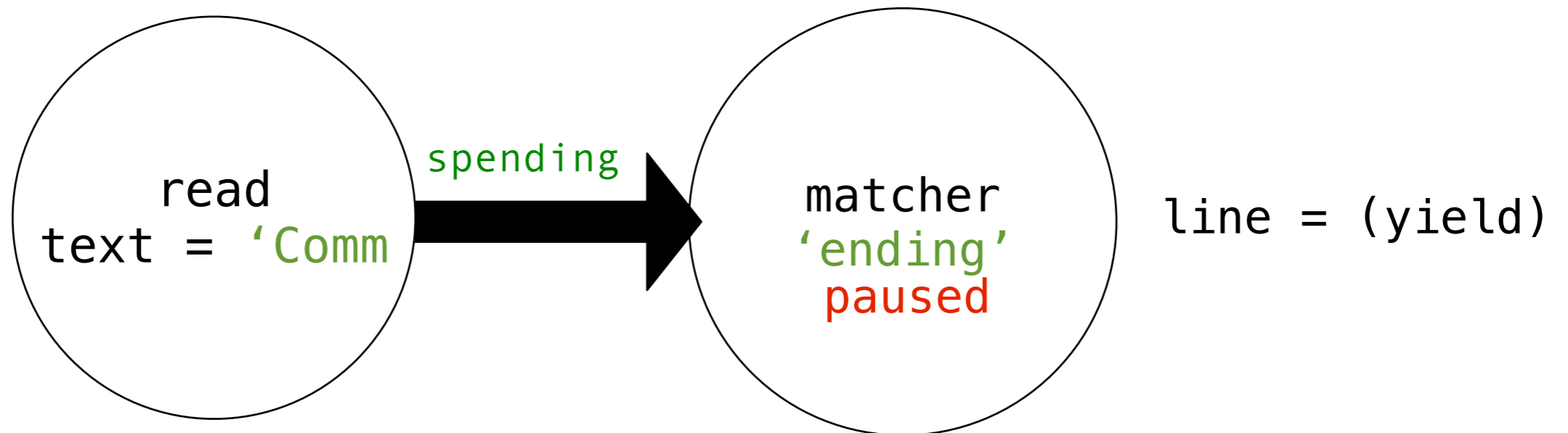
```
for word in text.split():  
    next_coroutine.send(word)
```



```
>>> matcher = match('ending')  
>>> matcher.__next__()  
'Looking for ending'  
>>> text = 'Commending spending is offending to people pending lending!'  
>>> read(text, matcher)  
'Commending'
```

A simple pipeline

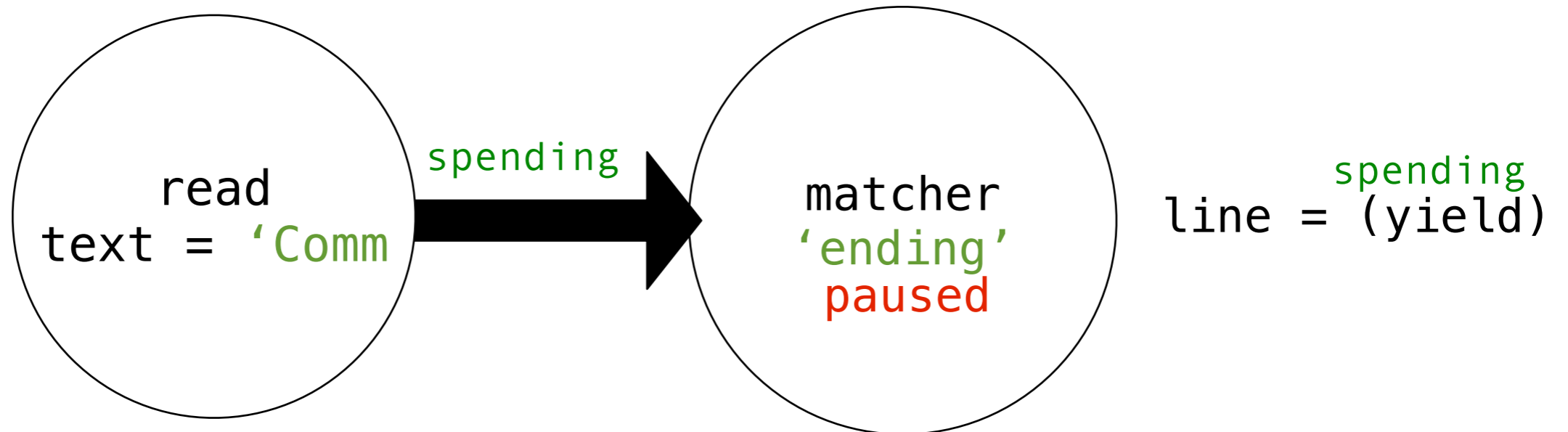
```
for word in text.split():  
    next_coroutine.send(word)
```



```
>>> matcher = match('ending')  
>>> matcher.__next__()  
'Looking for ending'  
>>> text = 'Commending spending is offending to people pending lending!'  
>>> read(text, matcher)  
'Commending'
```

A simple pipeline

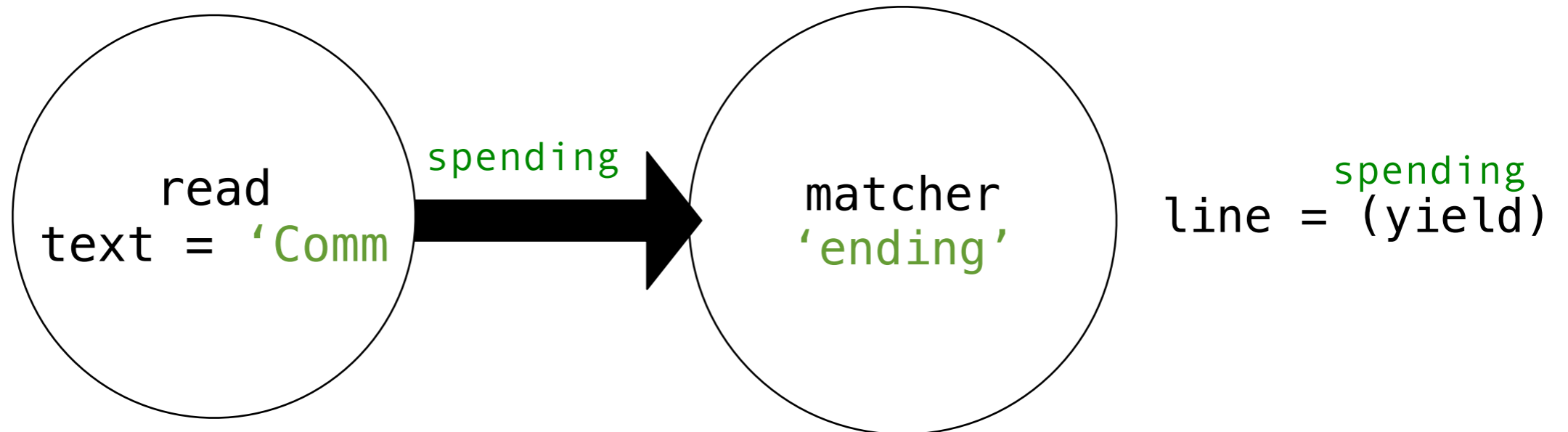
```
for word in text.split():  
    next_coroutine.send(word)
```



```
>>> matcher = match('ending')  
>>> matcher.__next__()  
'Looking for ending'  
>>> text = 'Commending spending is offending to people pending lending!'  
>>> read(text, matcher)  
'Commending'
```


A simple pipeline

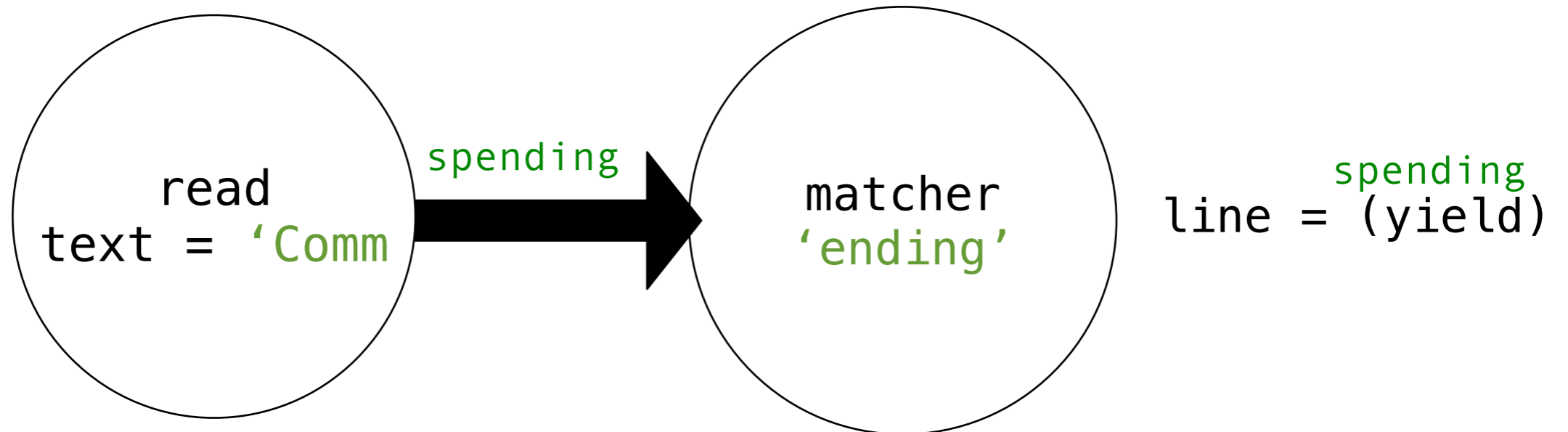
```
for word in text.split():  
    next_coroutine.send(word)
```



```
>>> matcher = match('ending')  
>>> matcher.__next__()  
'Looking for ending'  
>>> text = 'Commending spending is offending to people pending lending!'  
>>> read(text, matcher)  
'Commending'
```

A simple pipeline

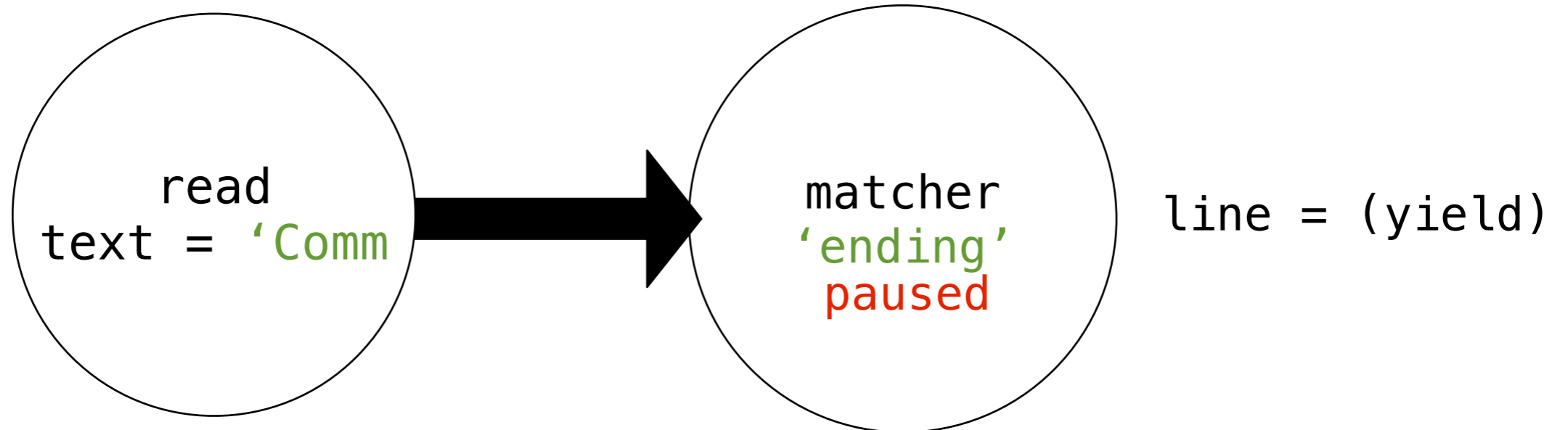
```
for word in text.split():  
    next_coroutine.send(word)
```



```
>>> matcher = match('ending')  
>>> matcher.__next__()  
'Looking for ending'  
>>> text = 'Commending spending is offending to people pending lending!'  
>>> read(text, matcher)  
'Commending'  
'spending'
```

A simple pipeline

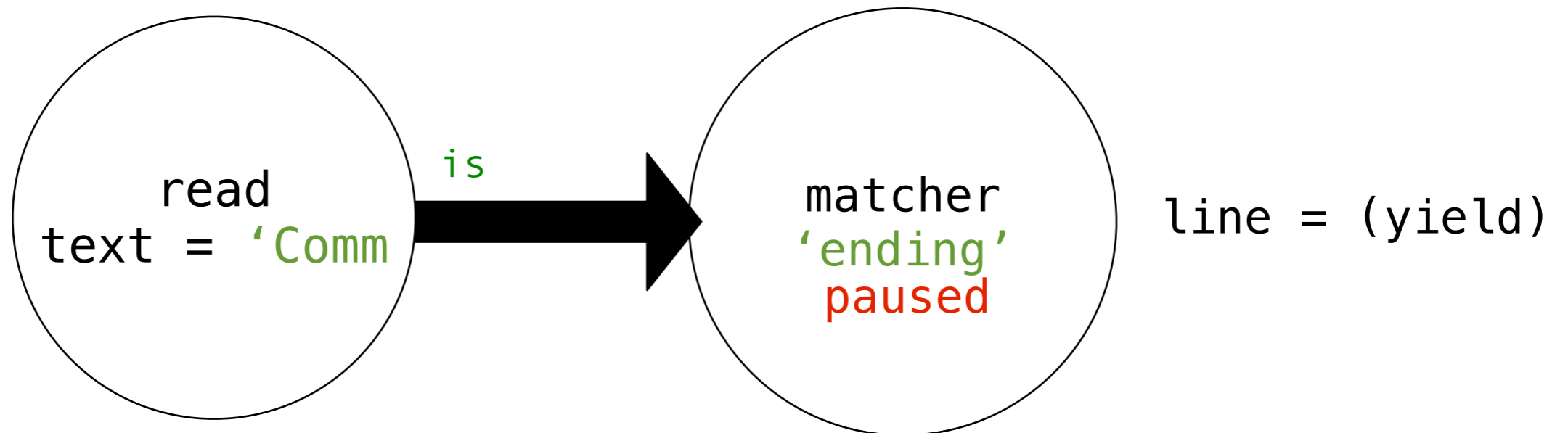
```
for word in text.split():  
    next_coroutine.send(word)
```



```
>>> matcher = match('ending')  
>>> matcher.__next__()  
'Looking for ending'  
>>> text = 'Commending spending is offending to people pending lending!'  
>>> read(text, matcher)  
'Commending'  
'spending'
```

A simple pipeline

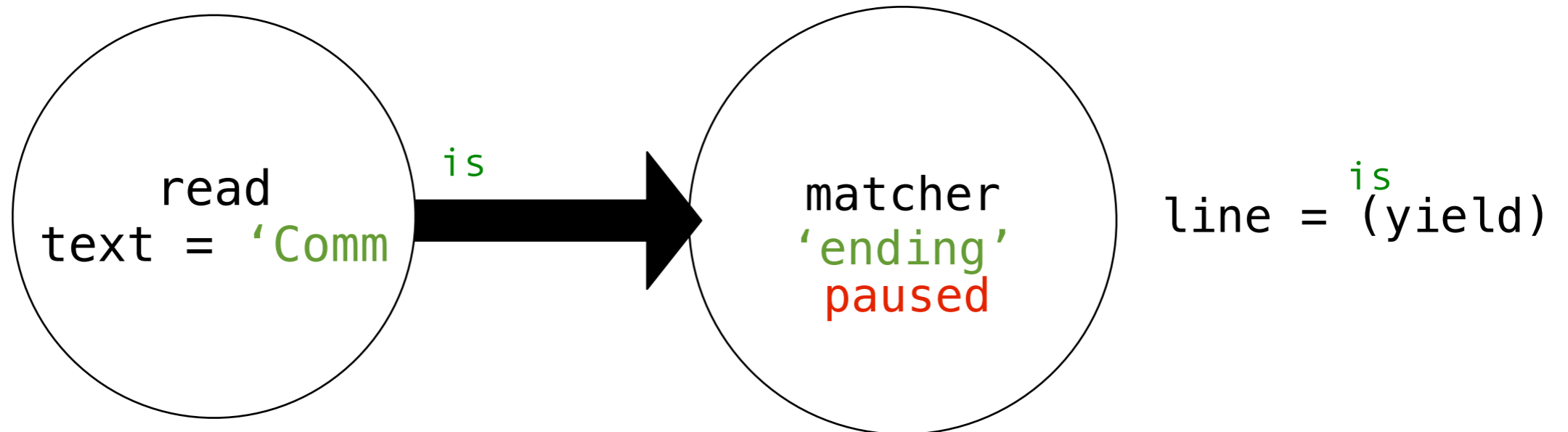
```
for word in text.split():  
    next_coroutine.send(word)
```



```
>>> matcher = match('ending')  
>>> matcher.__next__()  
'Looking for ending'  
>>> text = 'Commending spending is offending to people pending lending!'  
>>> read(text, matcher)  
'Commending'  
'spending'
```

A simple pipeline

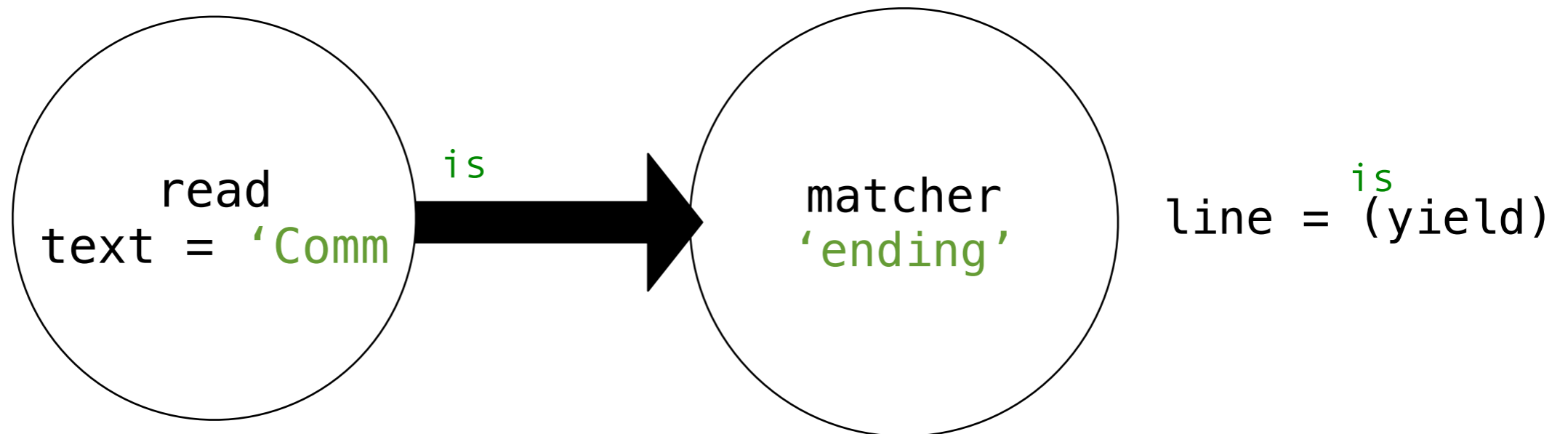
```
for word in text.split():  
    next_coroutine.send(word)
```



```
>>> matcher = match('ending')  
>>> matcher.__next__()  
'Looking for ending'  
>>> text = 'Commending spending is offending to people pending lending!'  
>>> read(text, matcher)  
'Commending'  
'spending'
```

A simple pipeline

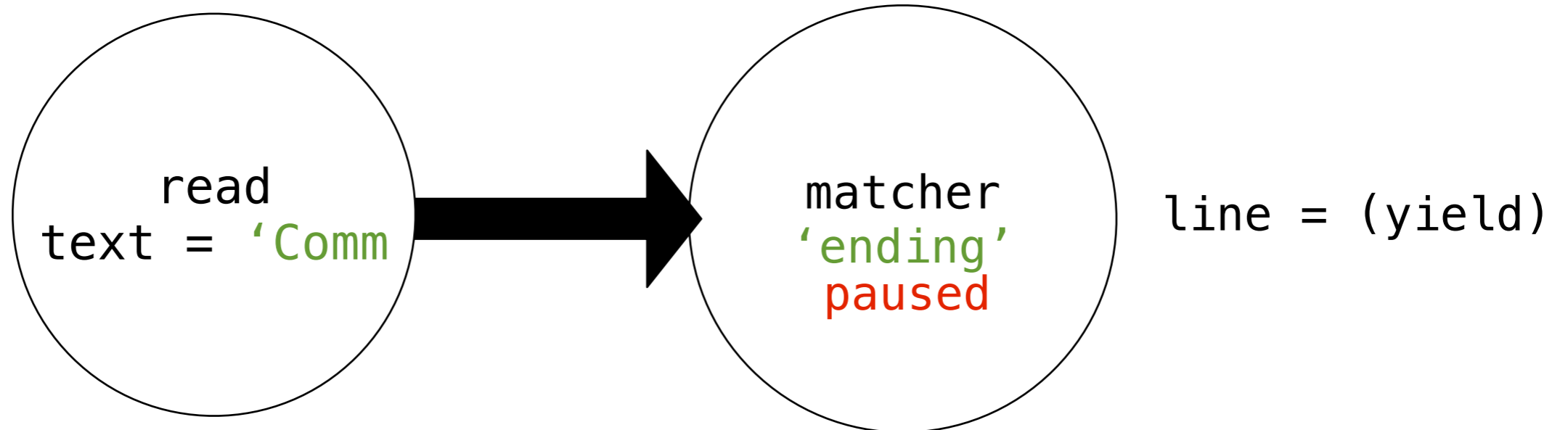
```
for word in text.split():  
    next_coroutine.send(word)
```



```
>>> matcher = match('ending')  
>>> matcher.__next__()  
'Looking for ending'  
>>> text = 'Commending spending is offending to people pending lending!'  
>>> read(text, matcher)  
'Commending'  
'spending'
```

A simple pipeline

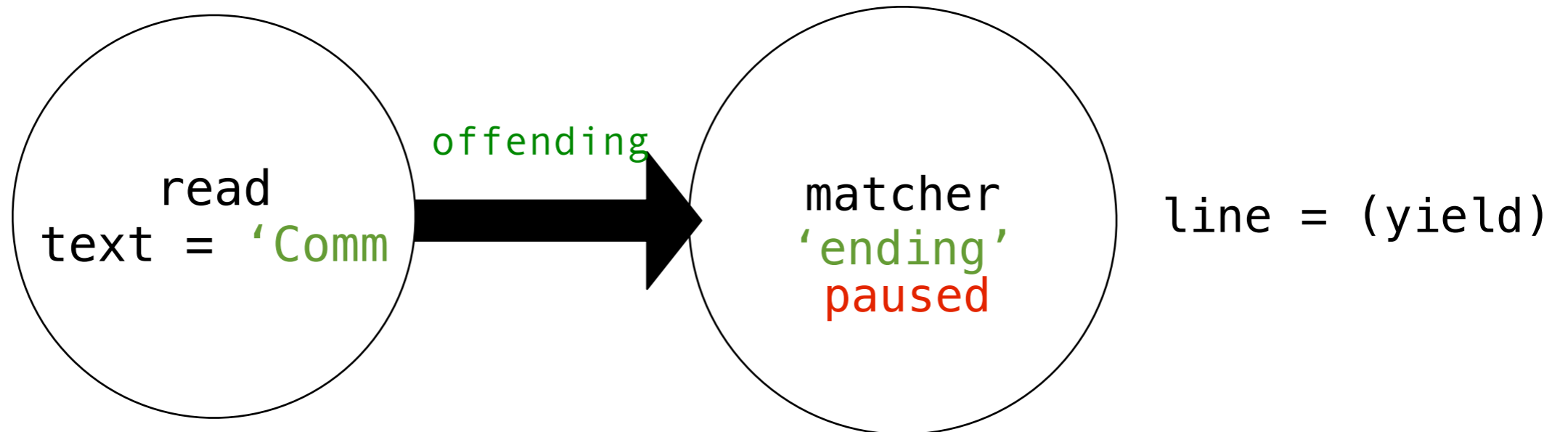
```
for word in text.split():  
    next_coroutine.send(word)
```



```
>>> matcher = match('ending')  
>>> matcher.__next__()  
'Looking for ending'  
>>> text = 'Commending spending is offending to people pending lending!'  
>>> read(text, matcher)  
'Commending'  
'spending'
```

A simple pipeline

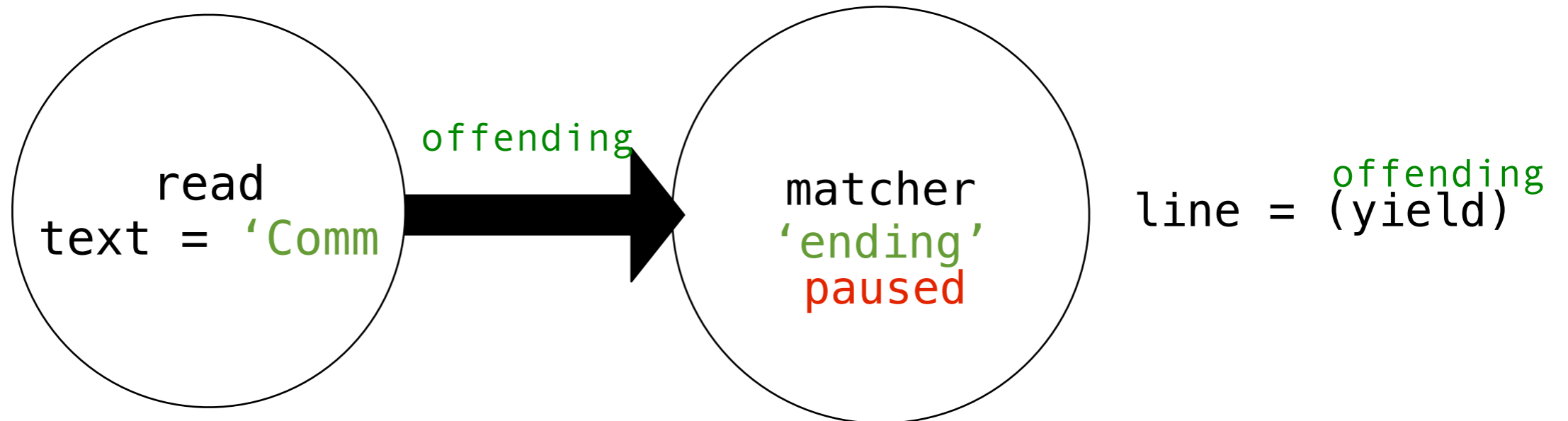
```
for word in text.split():  
    next_coroutine.send(word)
```



```
>>> matcher = match('ending')  
>>> matcher.__next__()  
'Looking for ending'  
>>> text = 'Commending spending is offending to people pending lending!'  
>>> read(text, matcher)  
'Commending'  
'spending'
```


A simple pipeline

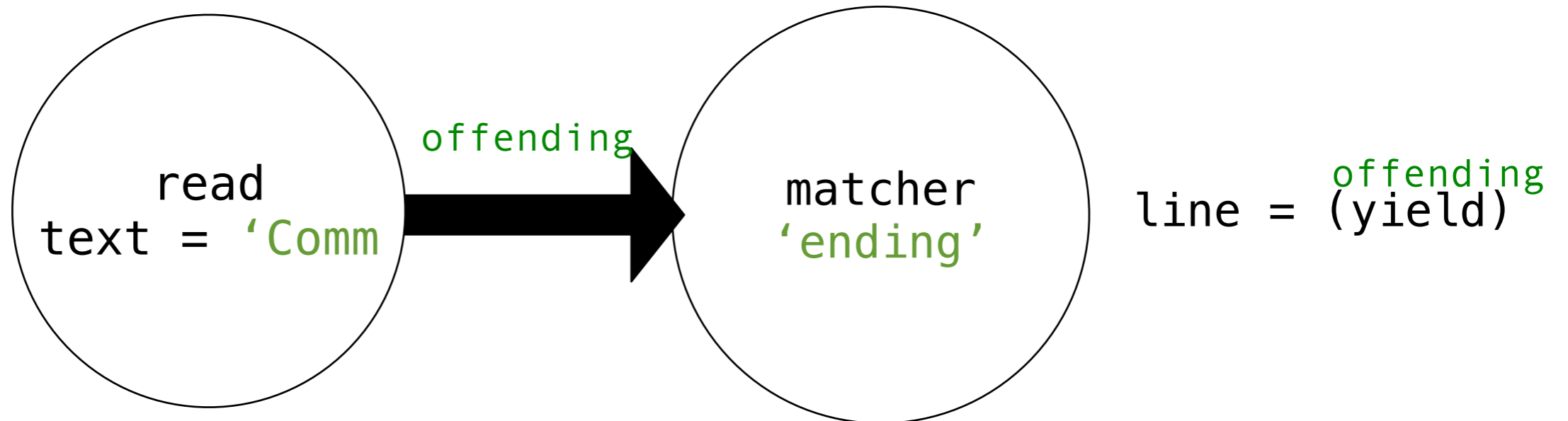
```
for word in text.split():  
    next_coroutine.send(word)
```



```
>>> matcher = match('ending')  
>>> matcher.__next__()  
'Looking for ending'  
>>> text = 'Commending spending is offending to people pending lending!'  
>>> read(text, matcher)  
'Commending'  
'spending'
```

A simple pipeline

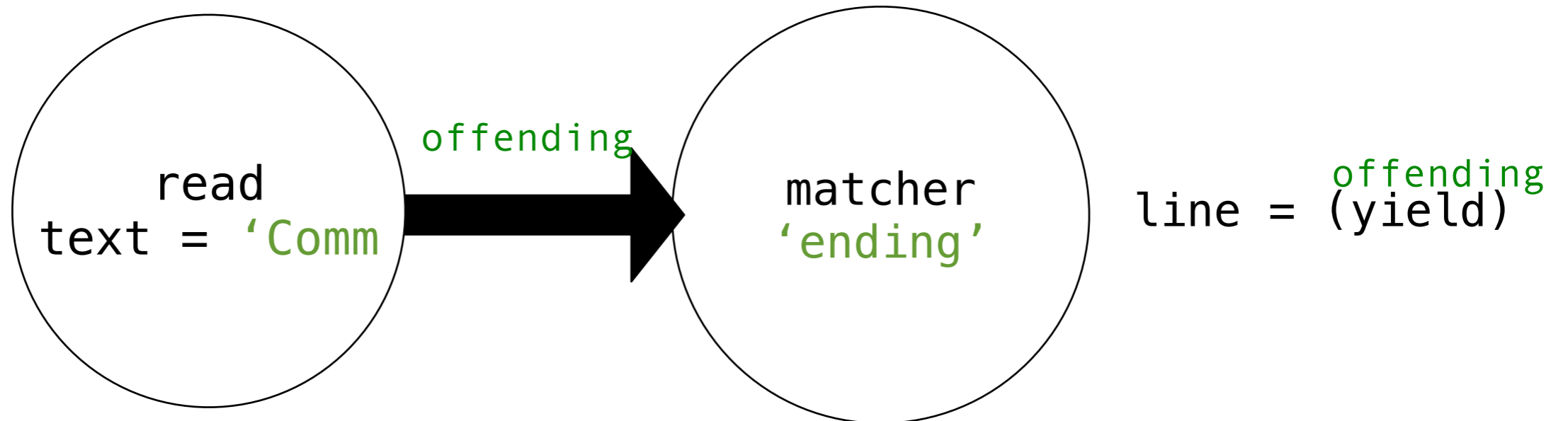
```
for word in text.split():  
    next_coroutine.send(word)
```



```
>>> matcher = match('ending')  
>>> matcher.__next__()  
'Looking for ending'  
>>> text = 'Commending spending is offending to people pending lending!'  
>>> read(text, matcher)  
'Commending'  
'spending'
```

A simple pipeline

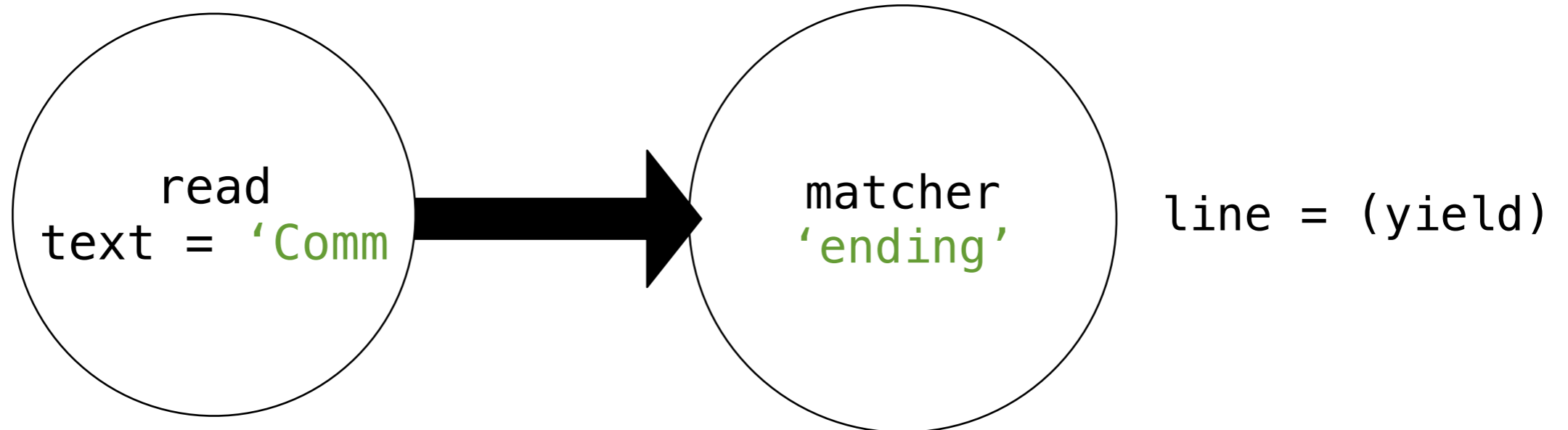
```
for word in text.split():  
    next_coroutine.send(word)
```



```
>>> matcher = match('ending')  
>>> matcher.__next__()  
'Looking for ending'  
>>> text = 'Commending spending is offending to people pending lending!'  
>>> read(text, matcher)  
'Commending'  
'spending'  
'offending'
```

A simple pipeline

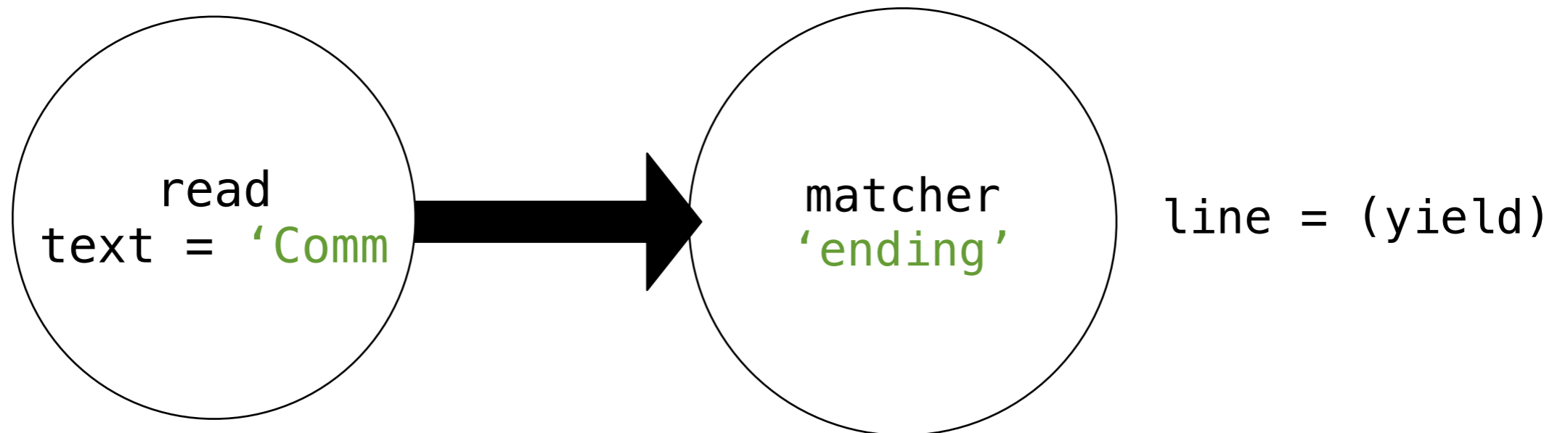
```
for word in text.split():  
    next_coroutine.send(word)
```



```
>>> matcher = match('ending')  
>>> matcher.__next__()  
'Looking for ending'  
>>> text = 'Commending spending is offending to people pending lending!'  
>>> read(text, matcher)  
'Commending'  
'spending'  
'offending'  
'pending'
```

A simple pipeline

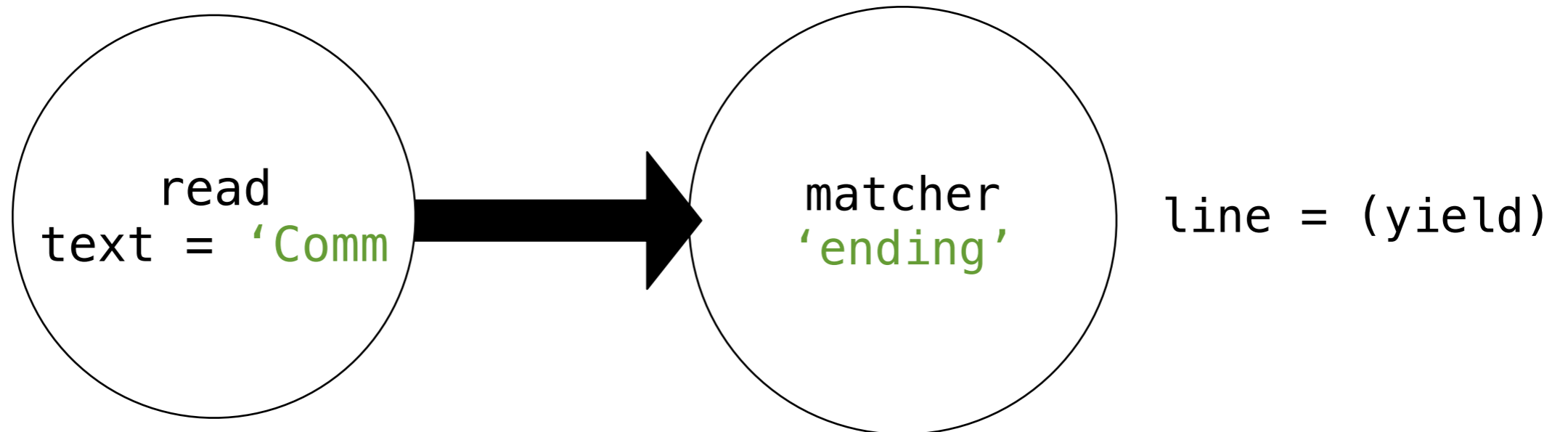
```
for word in text.split():  
    next_coroutine.send(word)
```



```
>>> matcher = match('ending')  
>>> matcher.__next__()  
'Looking for ending'  
>>> text = 'Commending spending is offending to people pending lending!'  
>>> read(text, matcher)  
'Commending'  
'spending'  
'offending'  
'pending'  
'lending!'
```

A simple pipeline

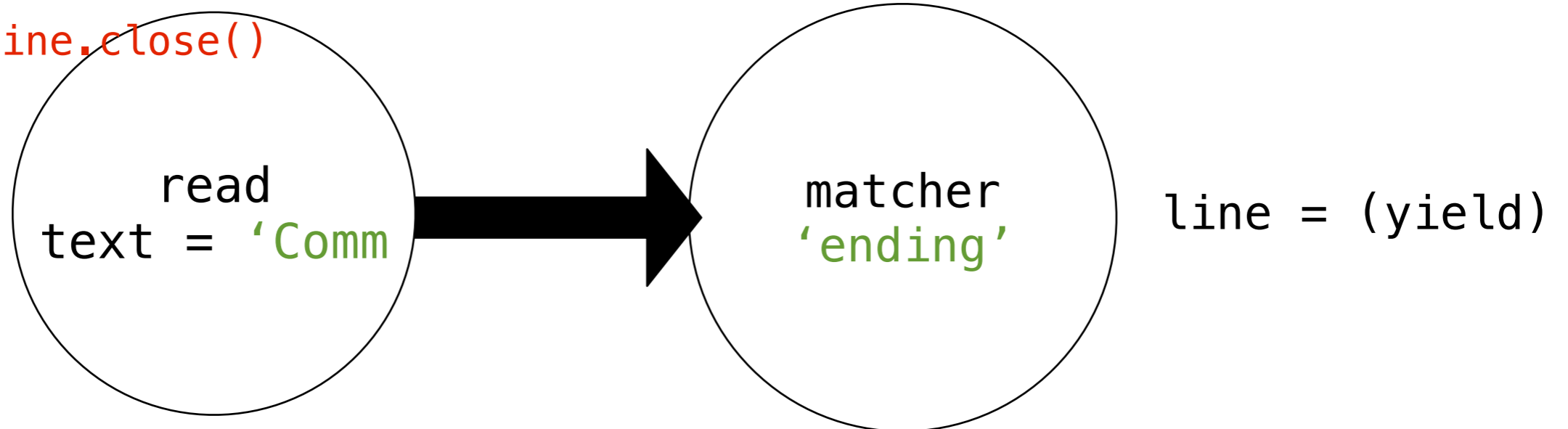
```
for word in text.split():  
    next_coroutine.send(word)
```



```
>>> matcher = match('ending')  
>>> matcher.__next__()  
'Looking for ending'  
>>> text = 'Commending spending is offending to people pending lending!'  
>>> read(text, matcher)  
'Commending'  
'spending'  
'offending'  
'pending'  
'lending!' ← last word!
```

A simple pipeline

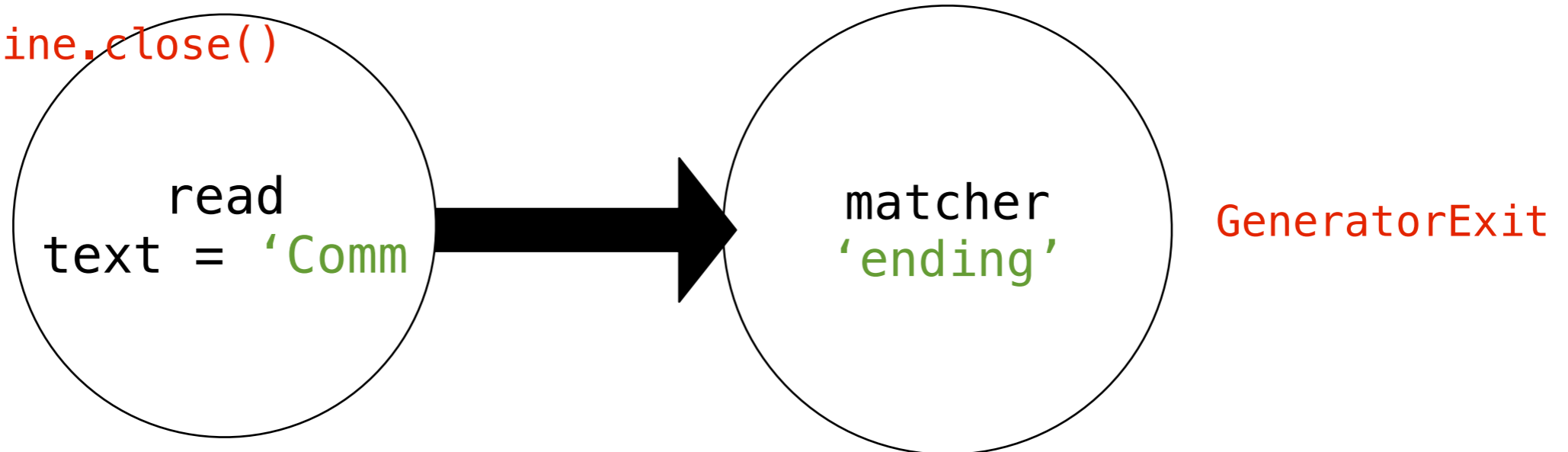
```
for word in text.split():
    next_coroutine.send(word)
next_coroutine.close()
```



```
>>> matcher = match('ending')
>>> matcher.__next__()
'Looking for ending'
>>> text = 'Commending spending is offending to people pending lending!'
>>> read(text, matcher)
'Commending'
'spending'
'offending'
'pending'
'lending!' ← last word!
```

A simple pipeline

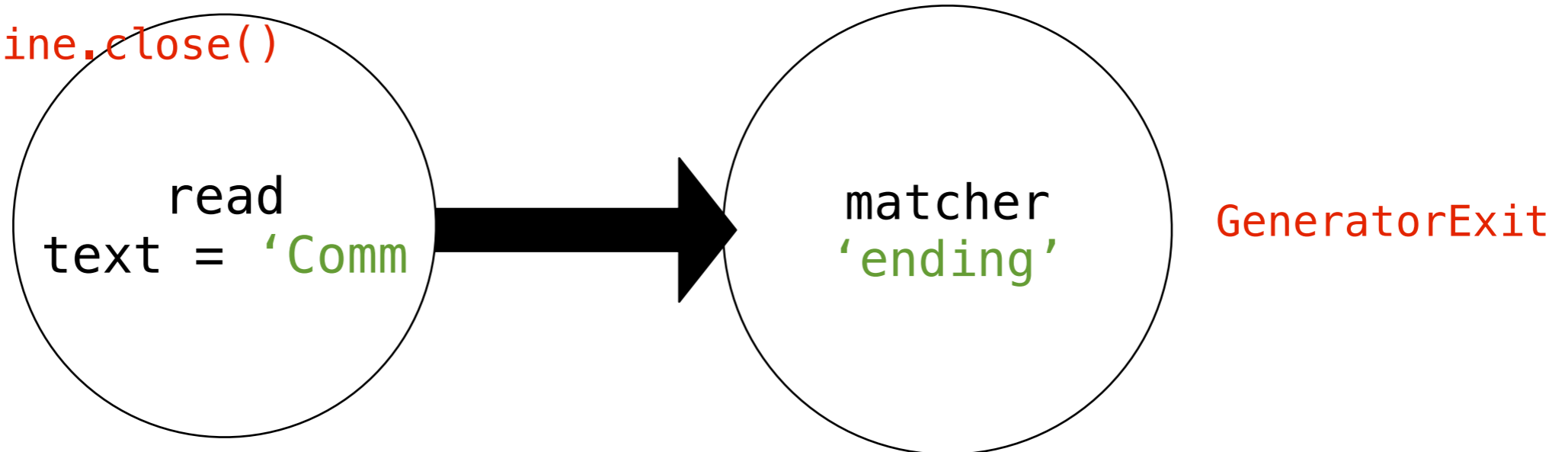
```
for word in text.split():
    next_coroutine.send(word)
next_coroutine.close()
```



```
>>> matcher = match('ending')
>>> matcher.__next__()
'Looking for ending'
>>> text = 'Commending spending is offending to people pending lending!'
>>> read(text, matcher)
'Commending'
'spending'
'offending'
'pending'
'lending!' ← last word!
```


A simple pipeline

```
for word in text.split():
    next_coroutine.send(word)
next_coroutine.close()
```

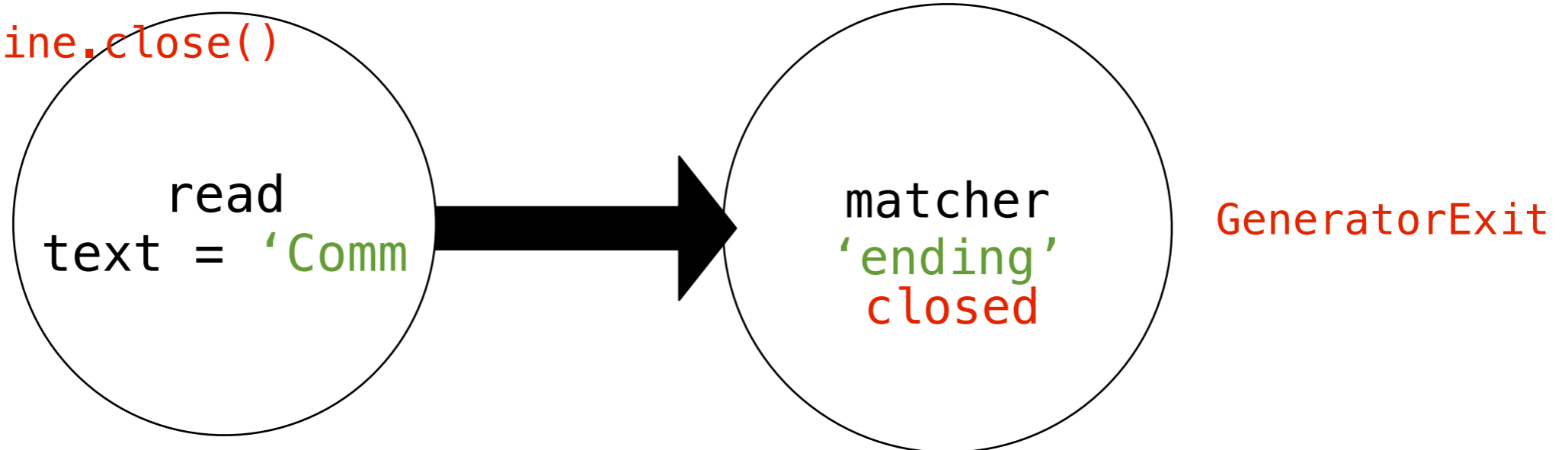


```
>>> matcher = match('ending')
>>> matcher.__next__()
'Looking for ending'
>>> text = 'Commending spending is offending to people pending lending!'
>>> read(text, matcher)
'Commending'
'spending'
'offending'
'pending'
'lending!'
'=== Done ==='
```

← last word!

A simple pipeline

```
for word in text.split():
    next_coroutine.send(word)
next_coroutine.close()
```



```
>>> matcher = match('ending')
>>> matcher.__next__()
'Looking for ending'
>>> text = 'Commending spending is offending to people pending lending!'
>>> read(text, matcher)
'Commending'
'spending'
'offending'
'pending'
'lending!'
'=== Done ==='
```

last word!

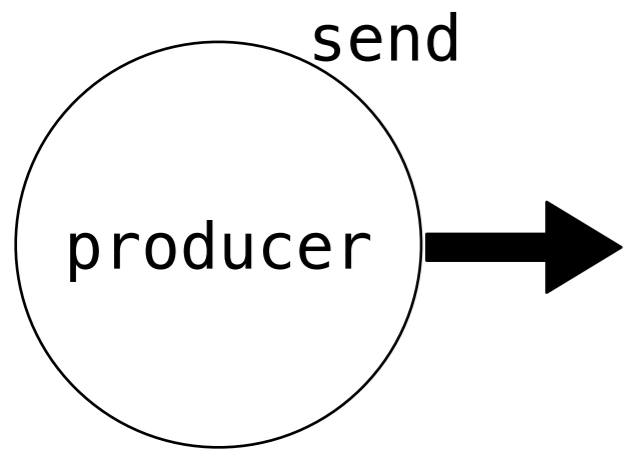
Produce, Filter, Consume

Produce, Filter, Consume

Coroutines can have different roles in a pipeline
Based on how they use `send()` and `yield`

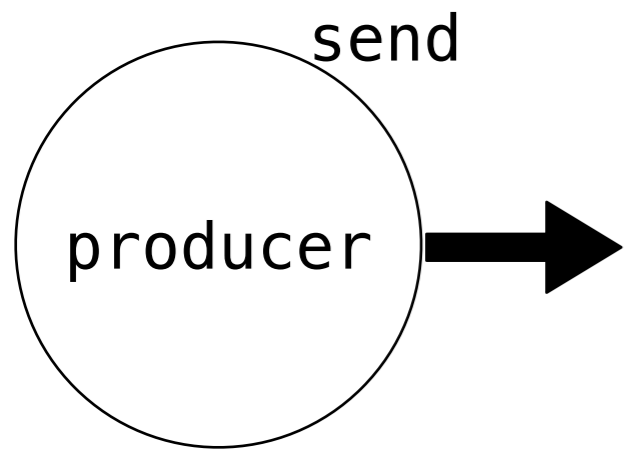
Produce, Filter, Consume

Coroutines can have different roles in a pipeline
Based on how they use `send()` and `yield`



Produce, Filter, Consume

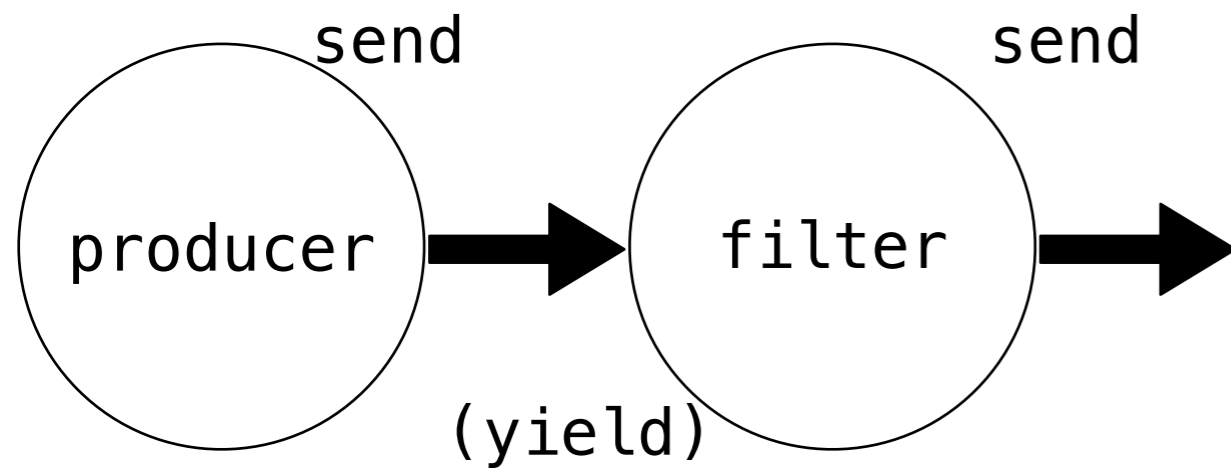
Coroutines can have different roles in a pipeline
Based on how they use `send()` and `yield`



The **producer**
only sends data

Produce, Filter, Consume

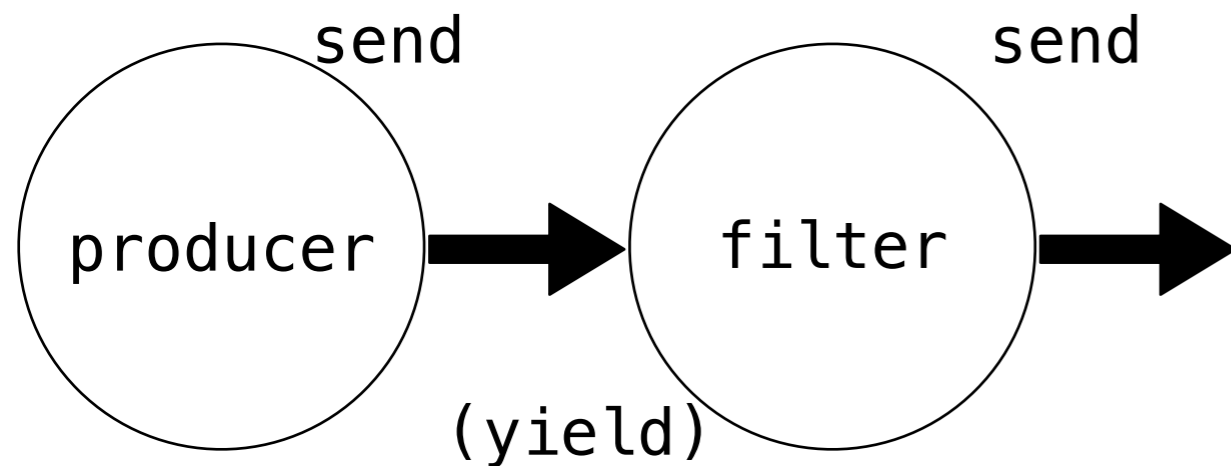
Coroutines can have different roles in a pipeline
Based on how they use `send()` and `yield`



The **producer**
only sends data

Produce, Filter, Consume

Coroutines can have different roles in a pipeline
Based on how they use `send()` and `yield`

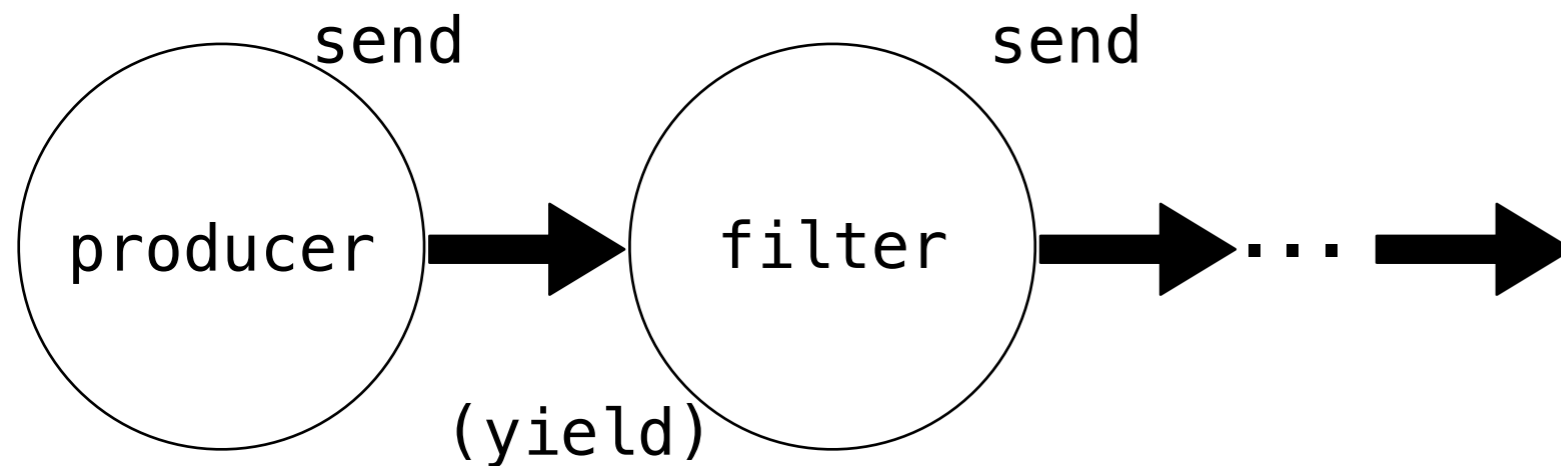


The **producer**
only sends data

The **filter**
consumes with `(yield)`
and sends results
downstream

Produce, Filter, Consume

Coroutines can have different roles in a pipeline
Based on how they use `send()` and `yield`

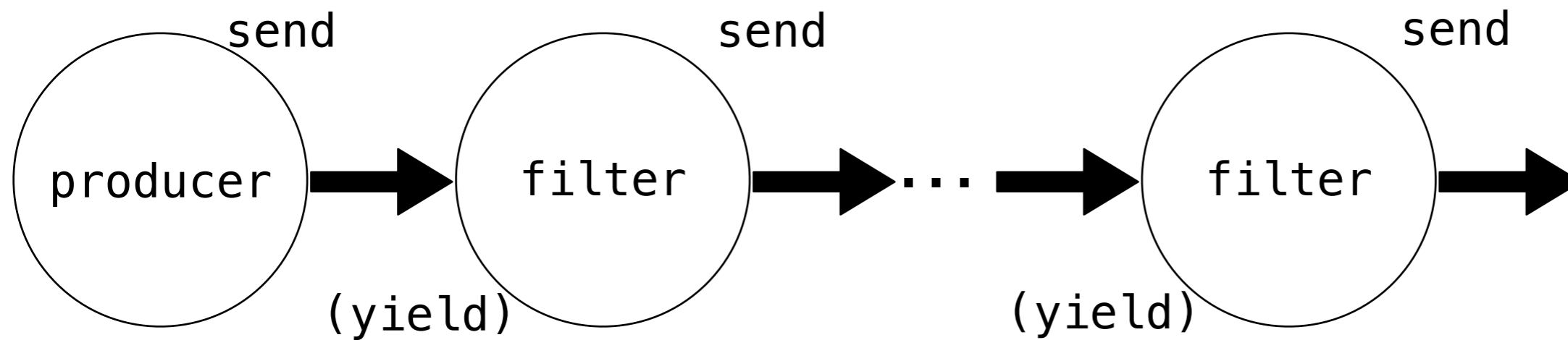


The **producer**
only sends data

The **filter**
consumes with `(yield)`
and sends results
downstream

Produce, Filter, Consume

Coroutines can have different roles in a pipeline
Based on how they use `send()` and `yield`

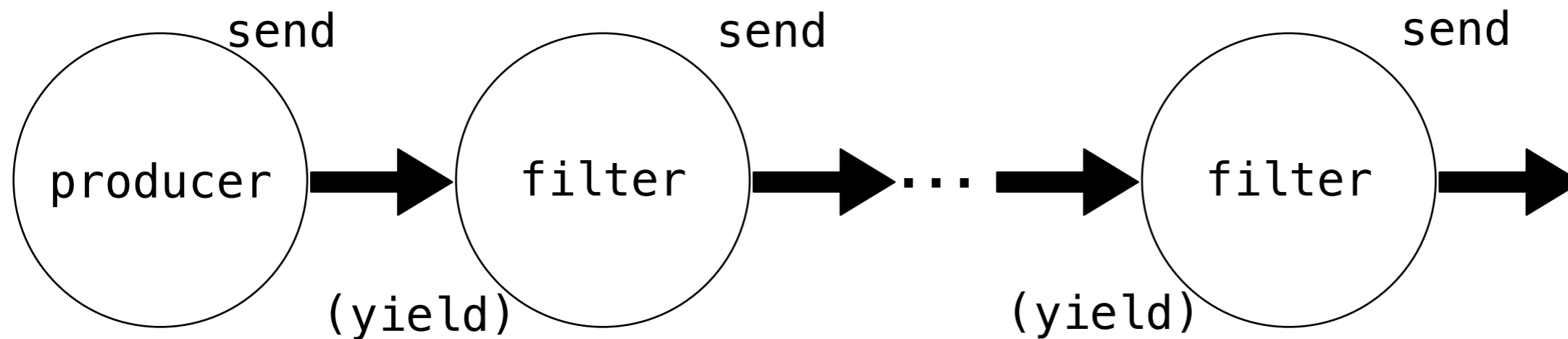


The **producer**
only sends data

The **filter**
consumes with `(yield)`
and sends results
downstream

Produce, Filter, Consume

Coroutines can have different roles in a pipeline
Based on how they use `send()` and `yield`



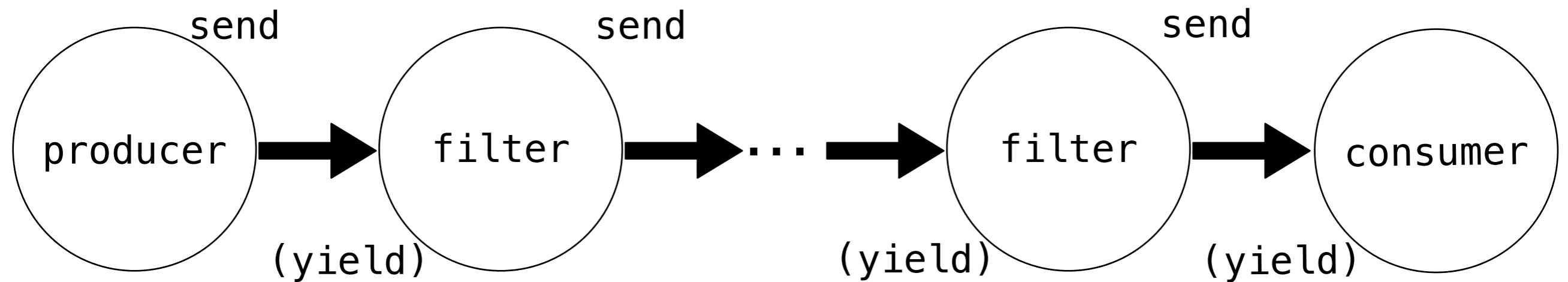
The **producer**
only sends data

The **filter**
consumes with `(yield)`
and sends results
downstream

There can be many
layers of filters

Produce, Filter, Consume

Coroutines can have different roles in a pipeline
Based on how they use `send()` and `yield`



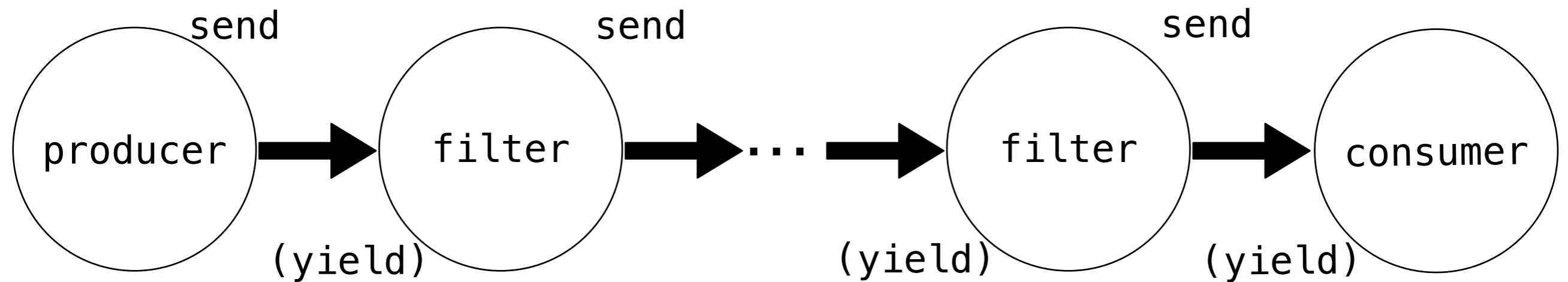
The **producer**
only sends data

The **filter**
consumes with `(yield)`
and sends results
downstream

There can be many
layers of filters

Produce, Filter, Consume

Coroutines can have different roles in a pipeline
Based on how they use `send()` and `yield`



The **producer**
only sends data

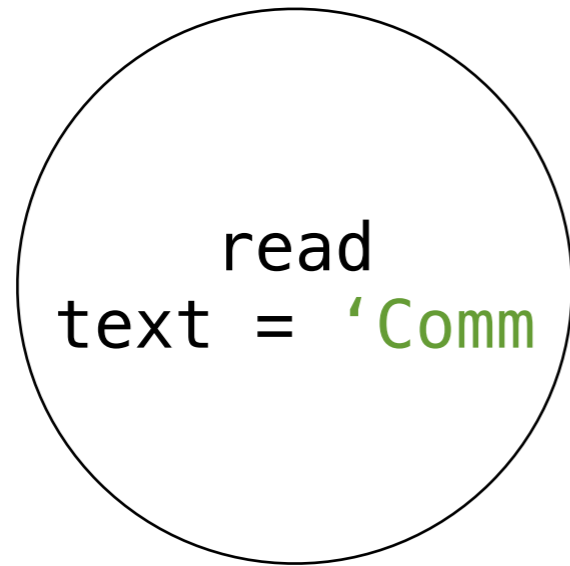
The **filter**
consumes with `(yield)`
and sends results
downstream

There can be many
layers of filters

The **consumer**
only consumes data

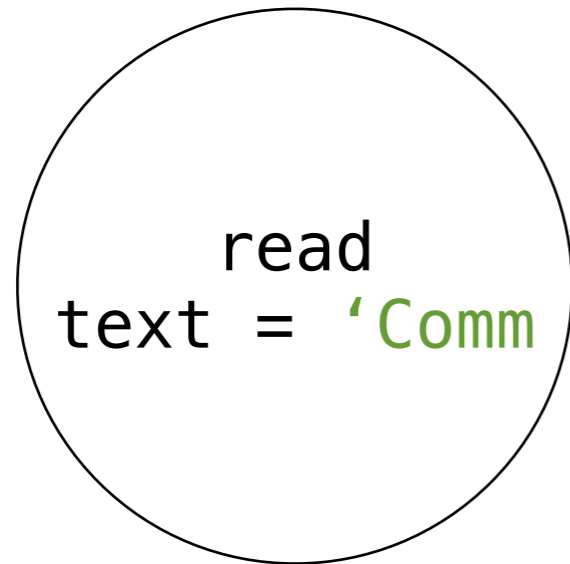
Example: simple pipeline

Example: simple pipeline



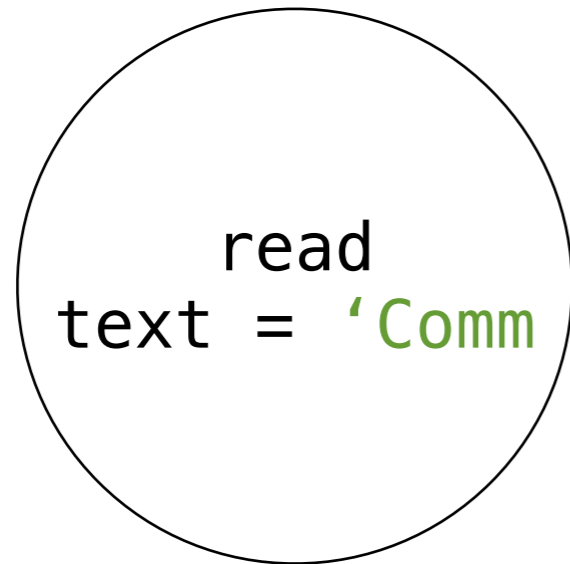
Example: simple pipeline

Producer



Example: simple pipeline

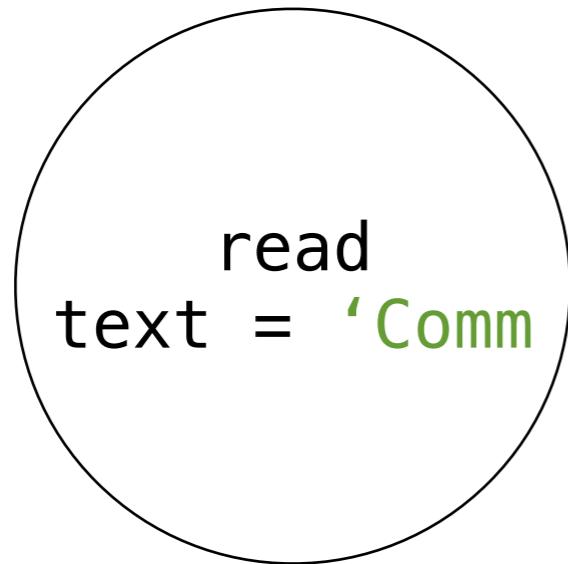
Producer



```
def read(text, next_coroutine):  
    for word in text.split():  
        next_coroutine.send(word)  
    next_coroutine.close()
```

Example: simple pipeline

Producer



```
def read(text, next_coroutine):  
    for word in text.split():  
        next_coroutine.send(word)  
    next_coroutine.close()
```

Example: simple pipeline

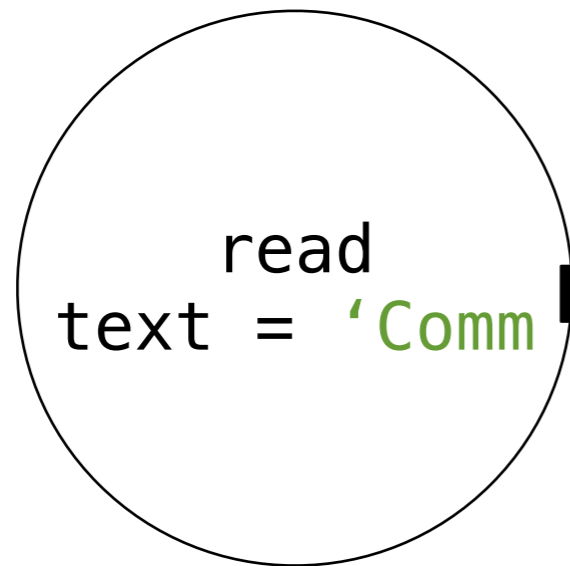
Producer



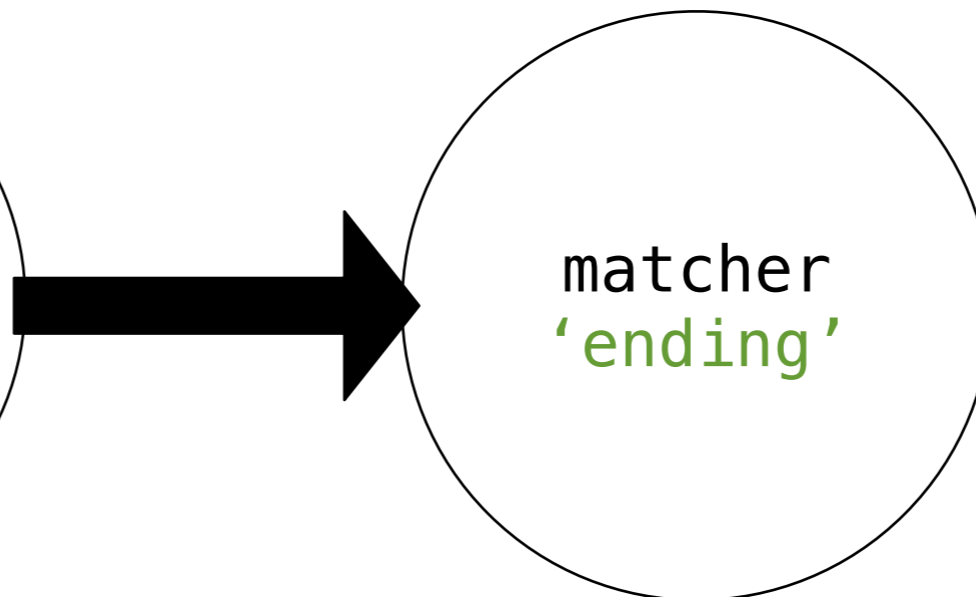
```
def match(pattern):  
    print('Looking for ' + pattern)  
    try:  
        while True:  
            s = (yield)  
            if pattern in s:  
                print(s)  
    except GeneratorExit:  
        print("=== Done ===")
```

Example: simple pipeline

Producer



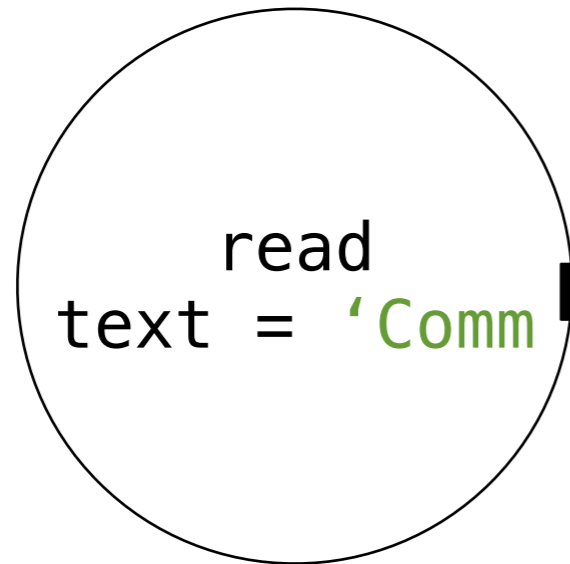
Consumer



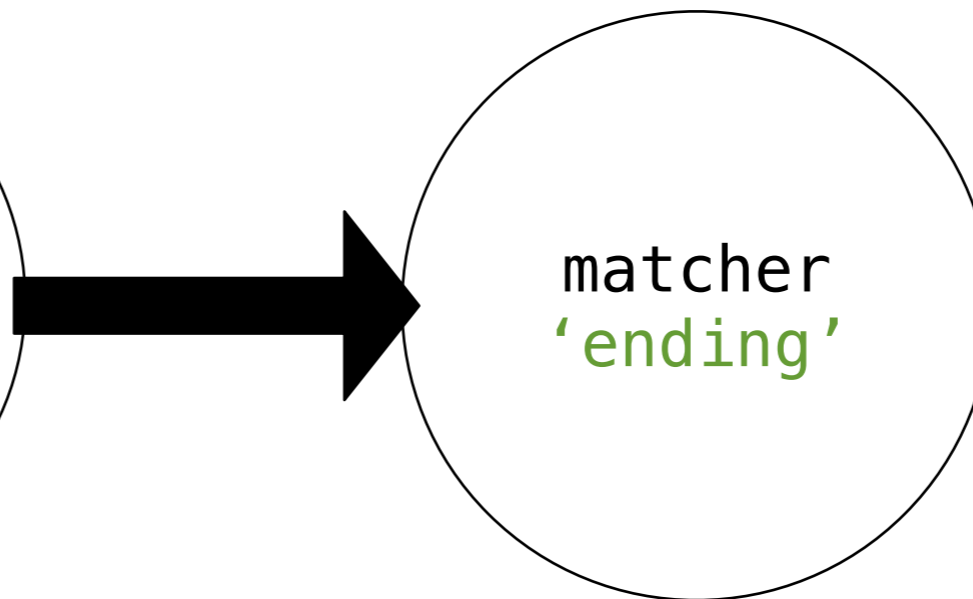
```
def match(pattern):  
    print('Looking for ' + pattern)  
    try:  
        while True:  
            s = (yield)  
            if pattern in s:  
                print(s)  
    except GeneratorExit:  
        print("=== Done ===")
```

Example: simple pipeline

Producer

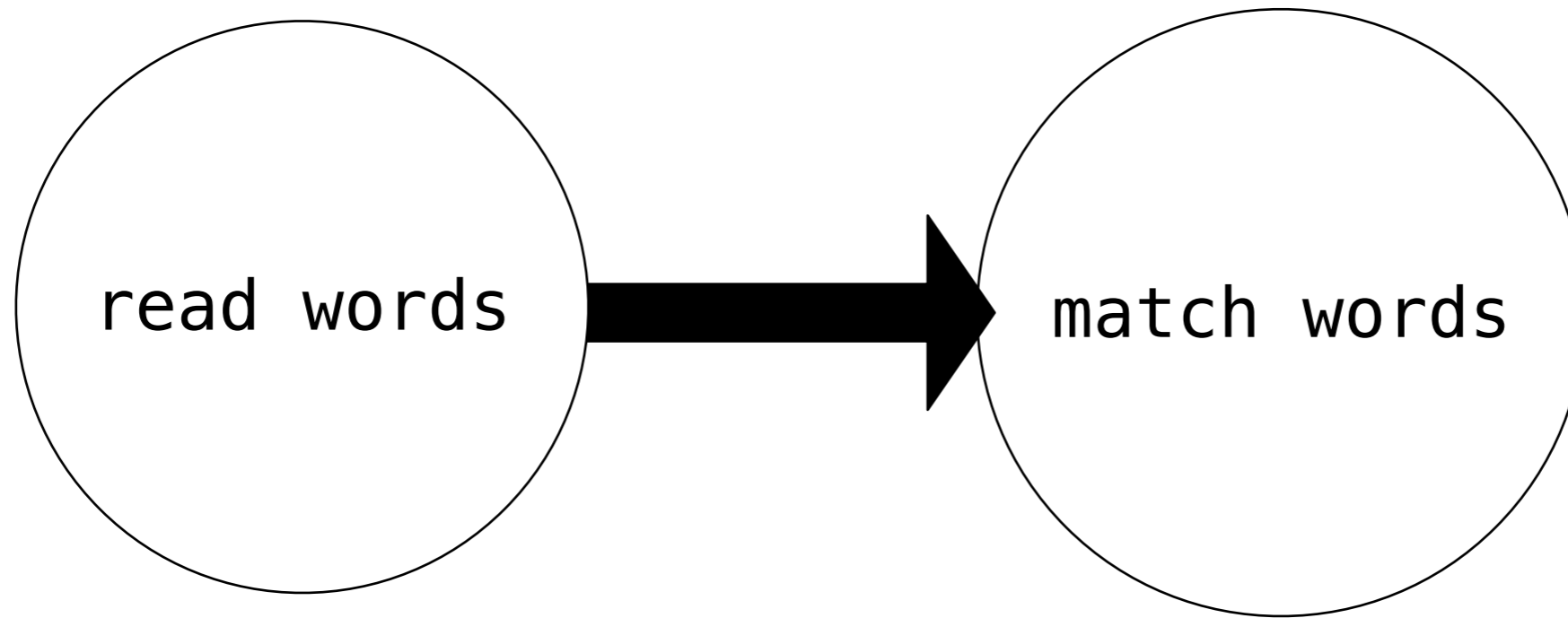


Consumer



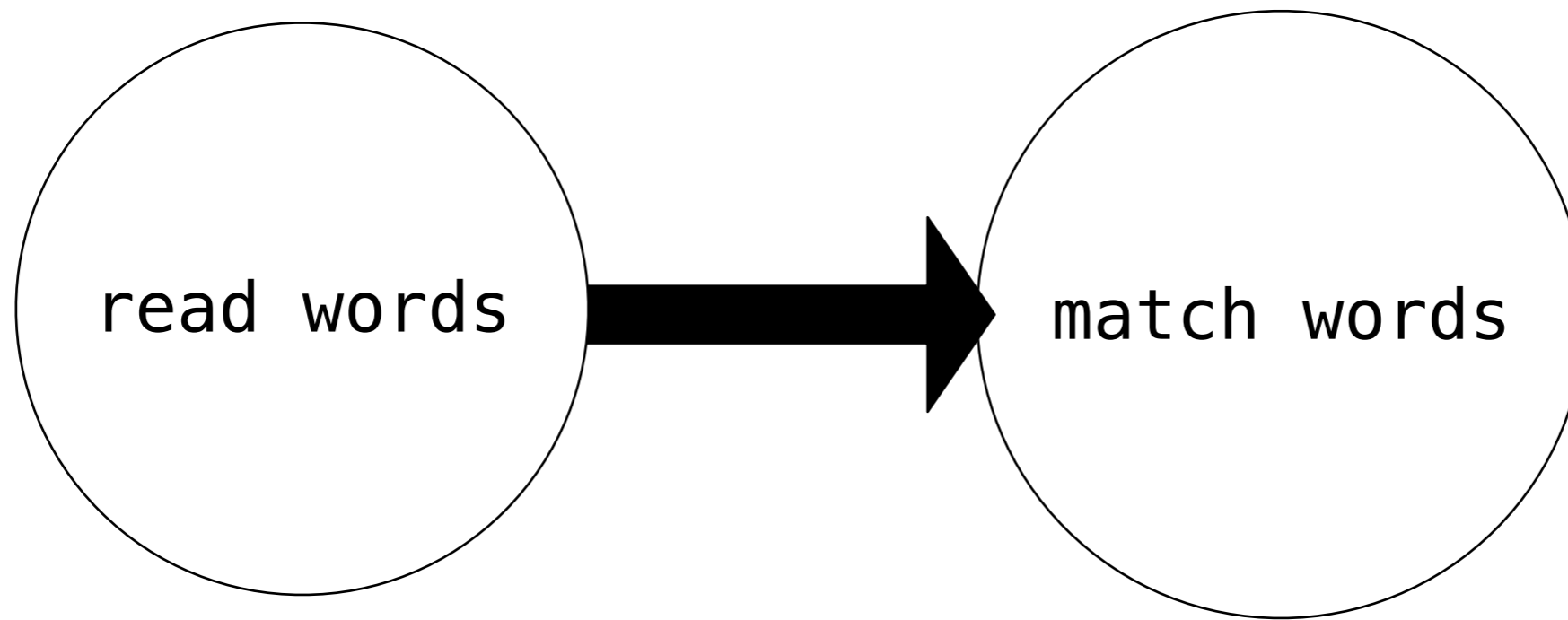
```
def match(pattern):  
    print('Looking for ' + pattern)  
    try:  
        while True:  
            s = (yield)  
            if pattern in s:  
                print(s)  
    except GeneratorExit:  
        print("=== Done ===")
```

Breaking down match



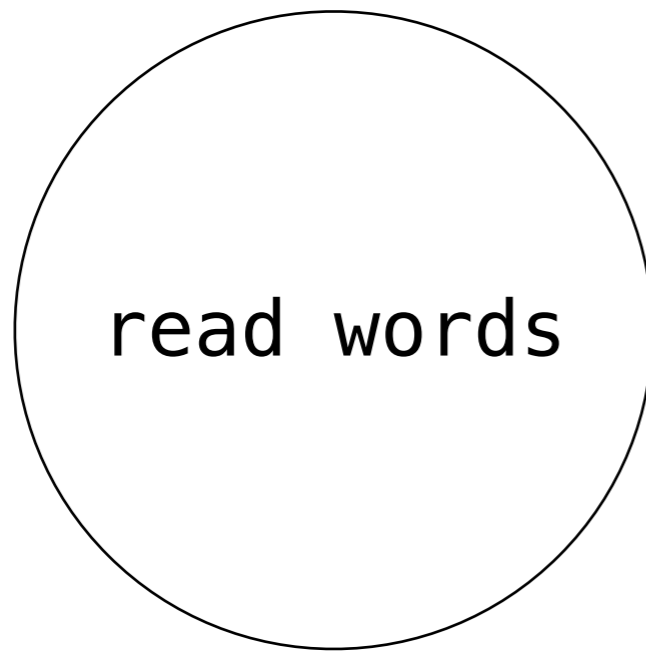
Breaking down match

Producer

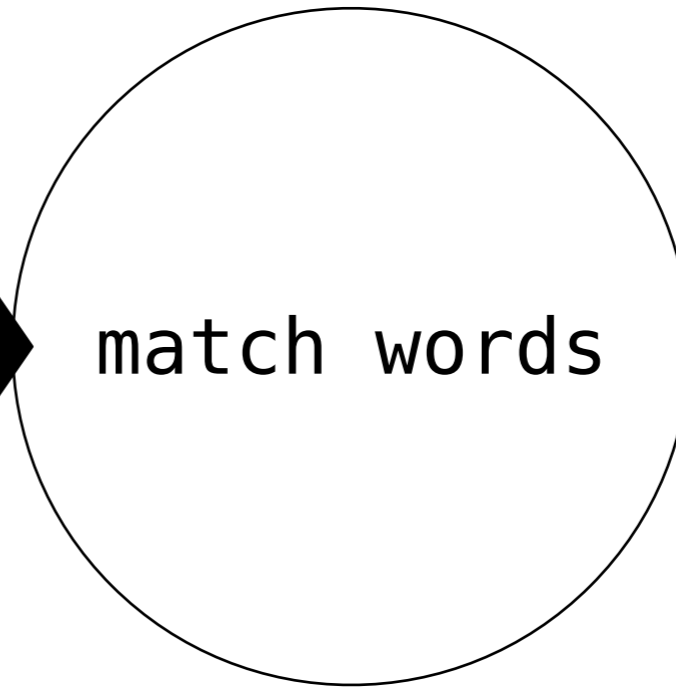


Breaking down match

Producer

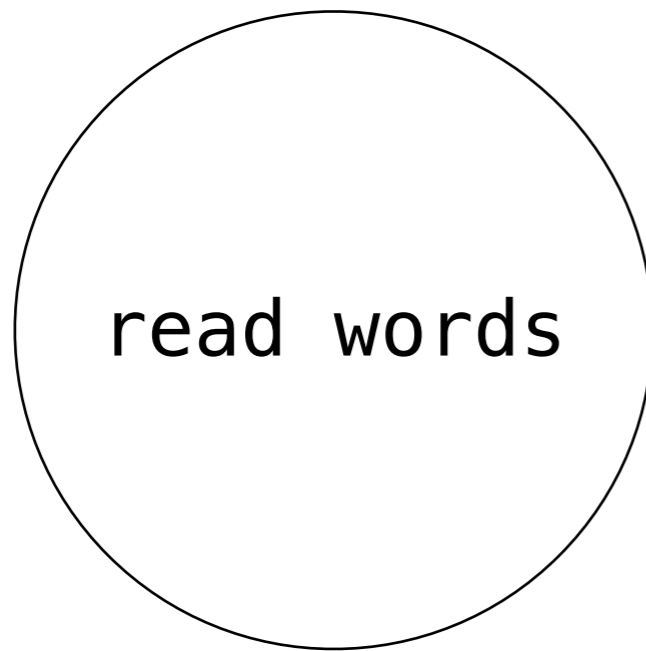


Consumer

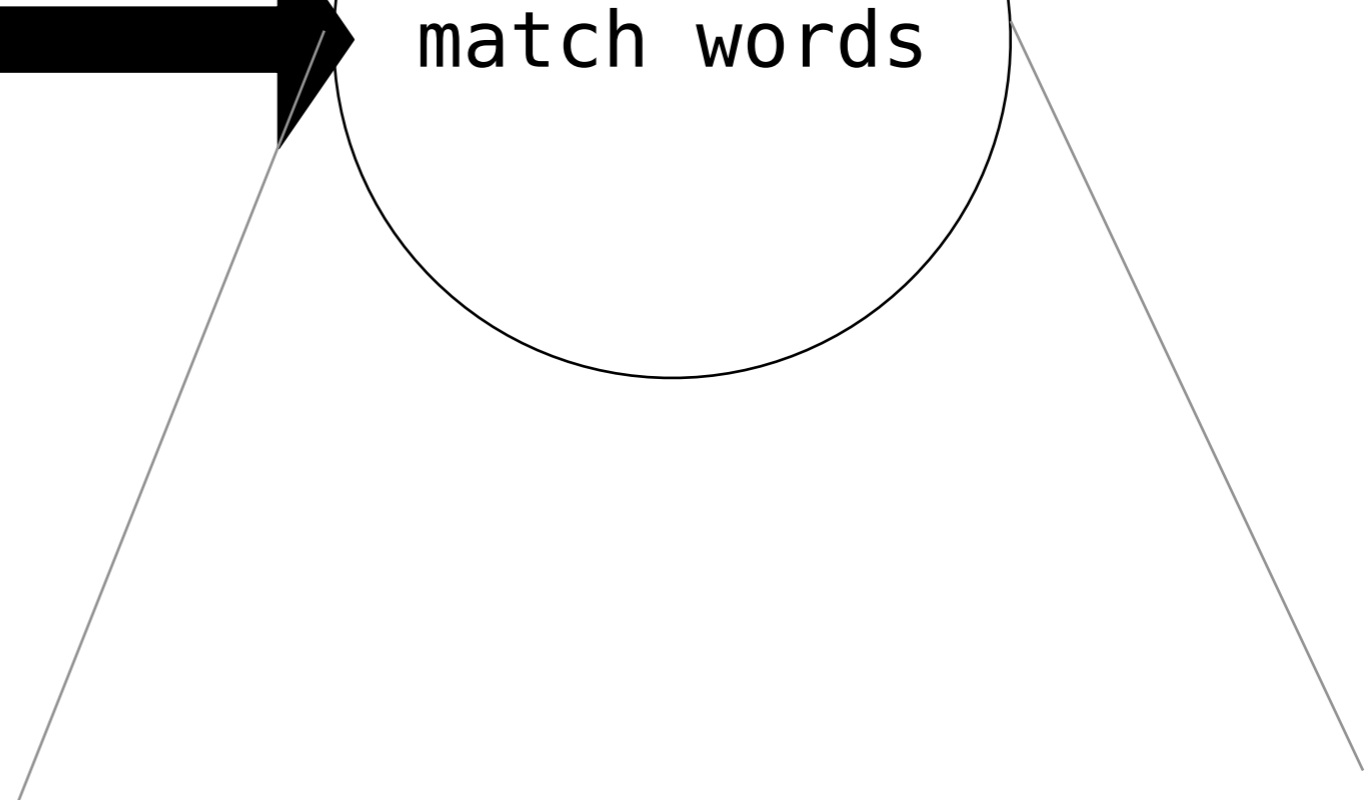
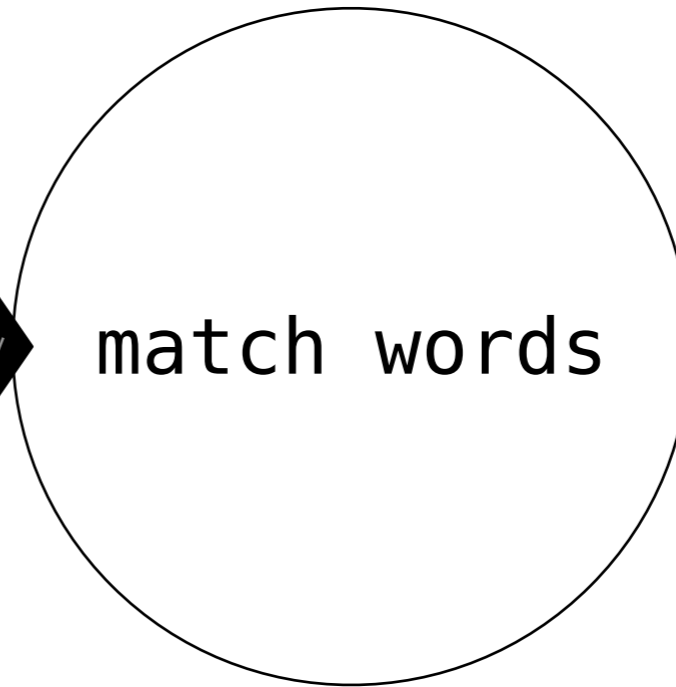


Breaking down match

Producer

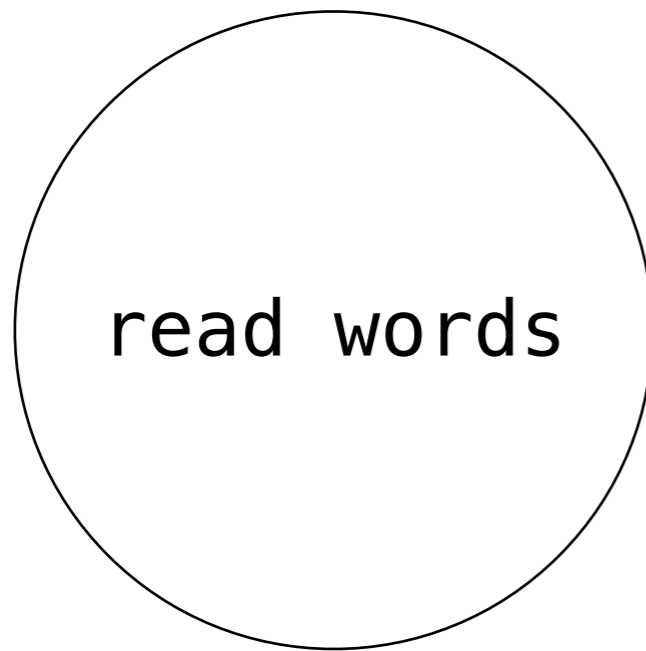


Consumer

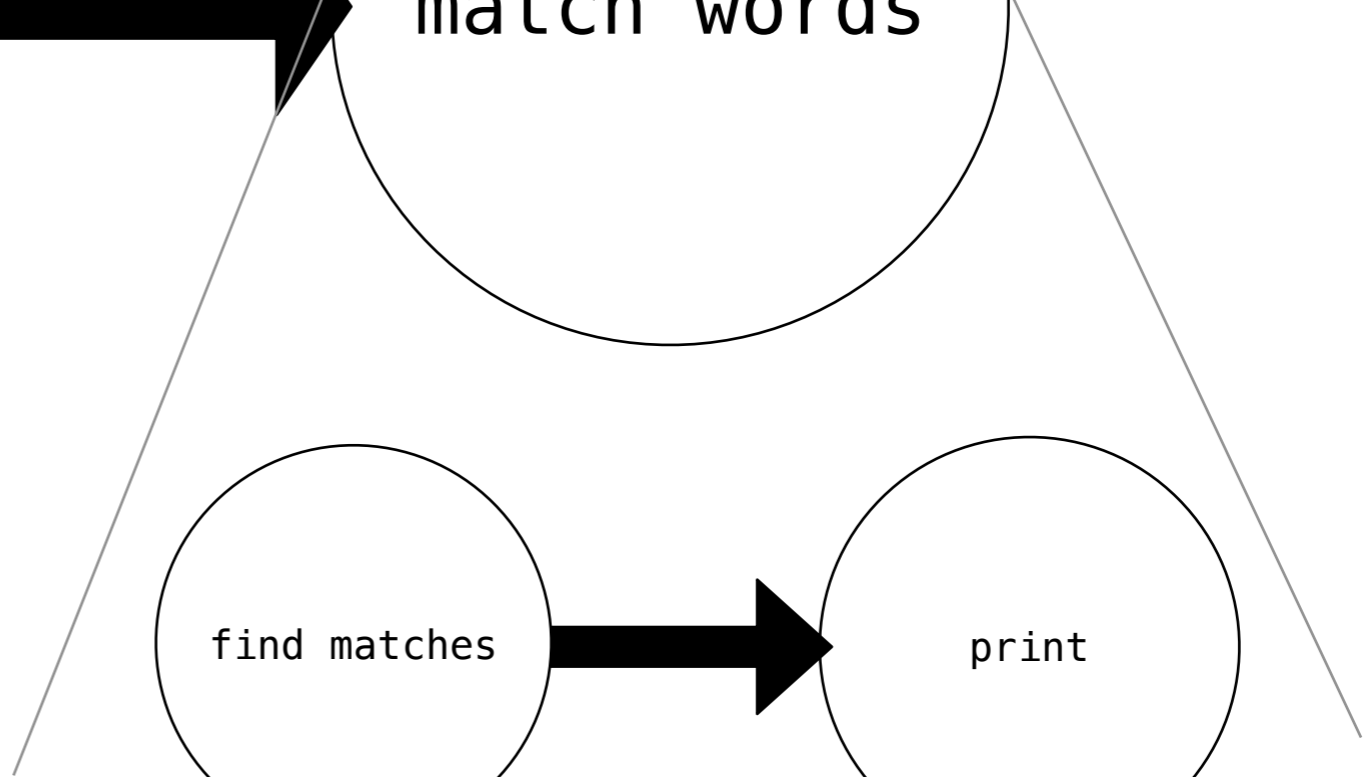
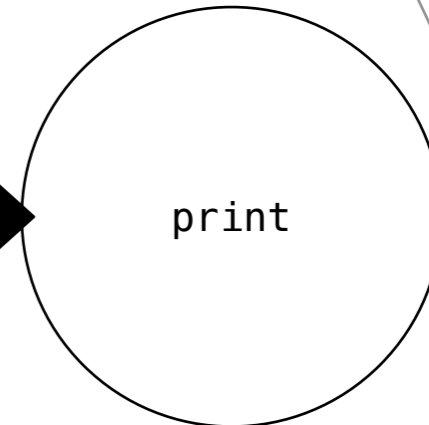
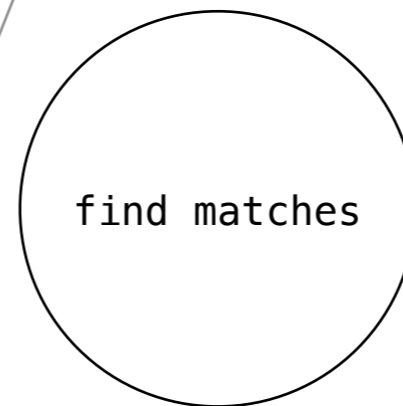
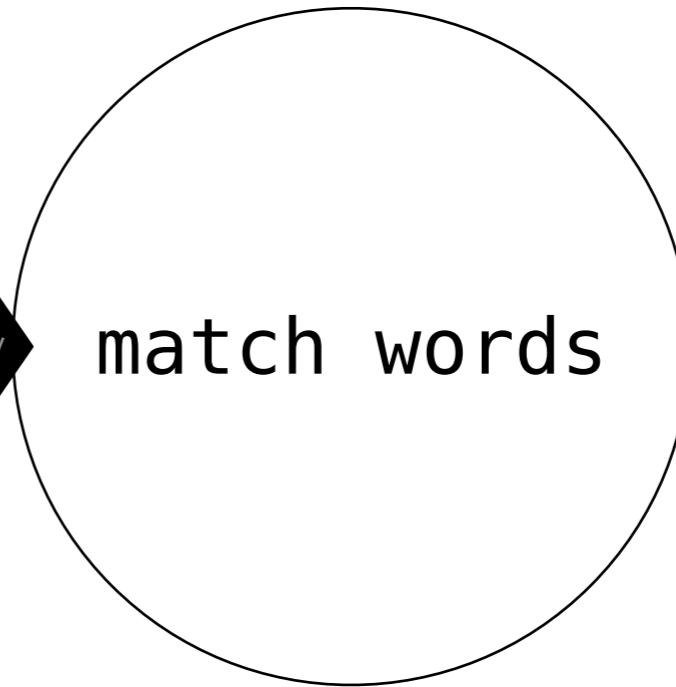


Breaking down match

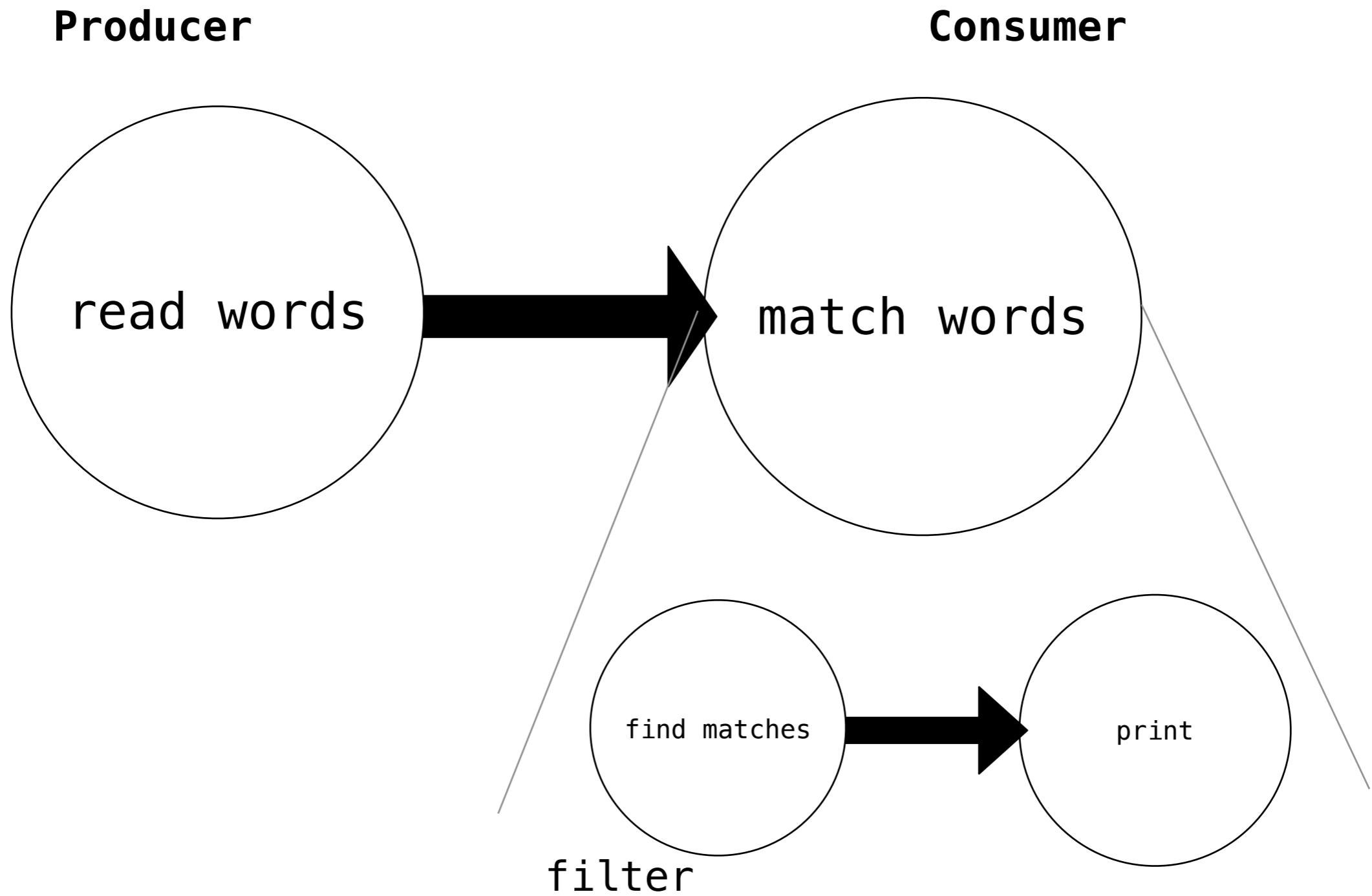
Producer



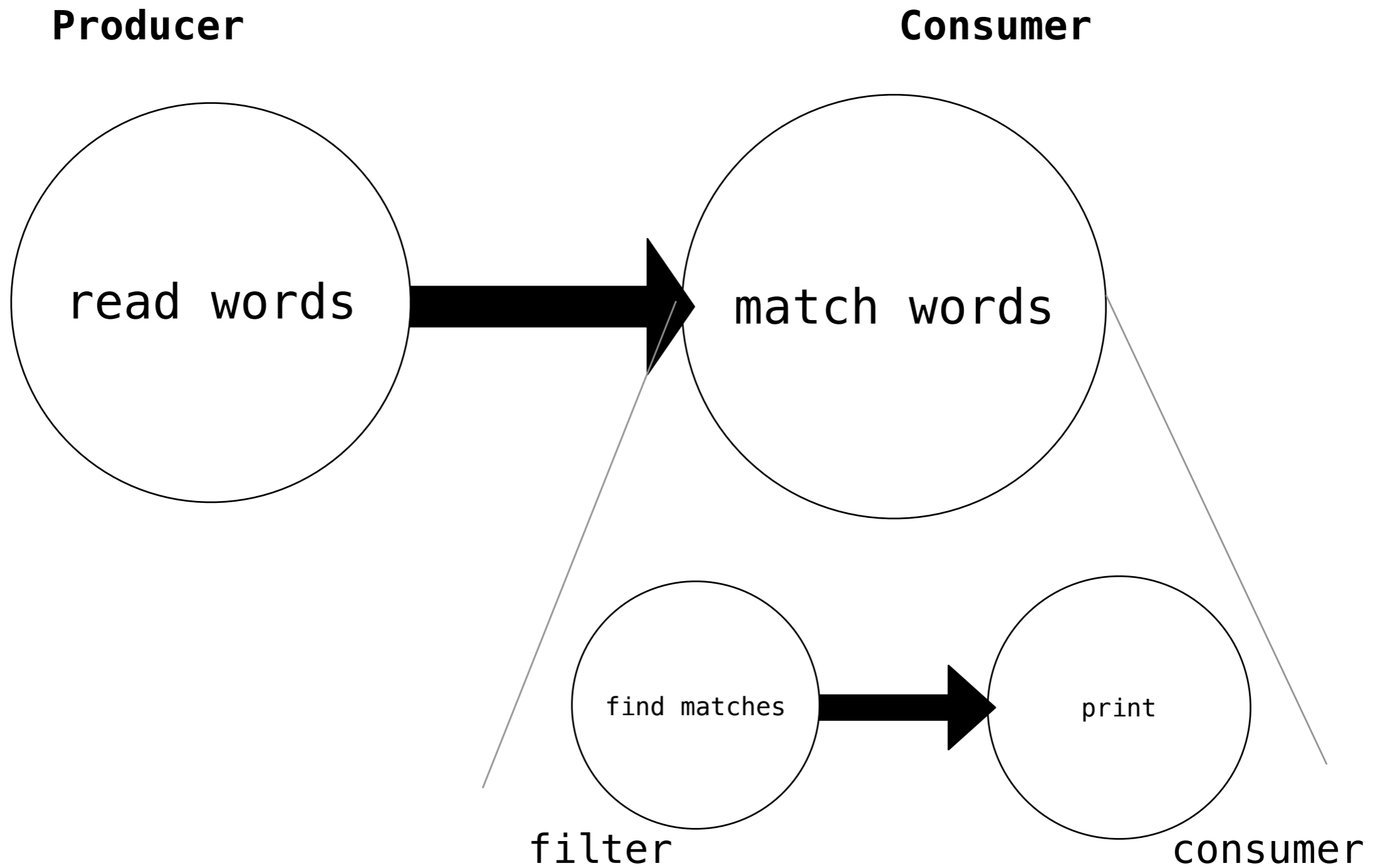
Consumer



Breaking down match

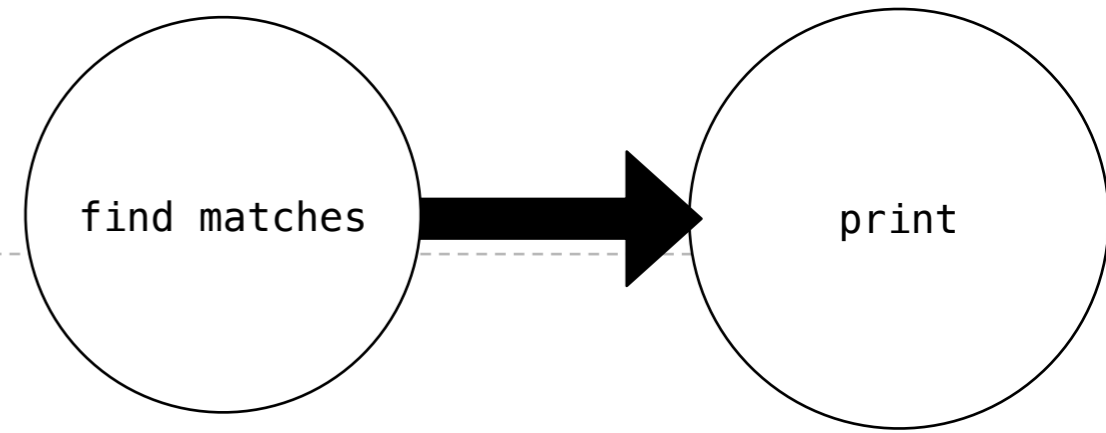


Breaking down match

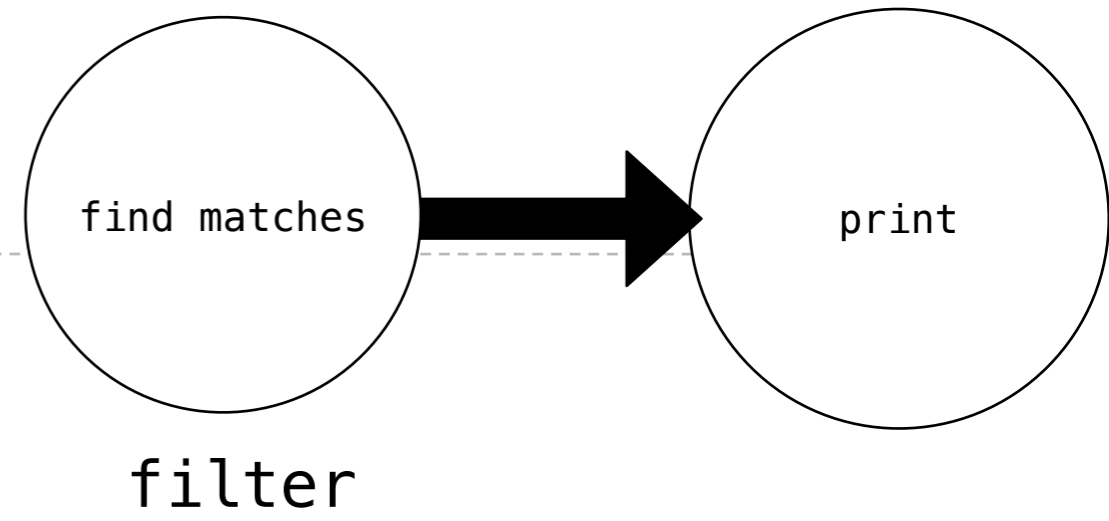


Breaking down match

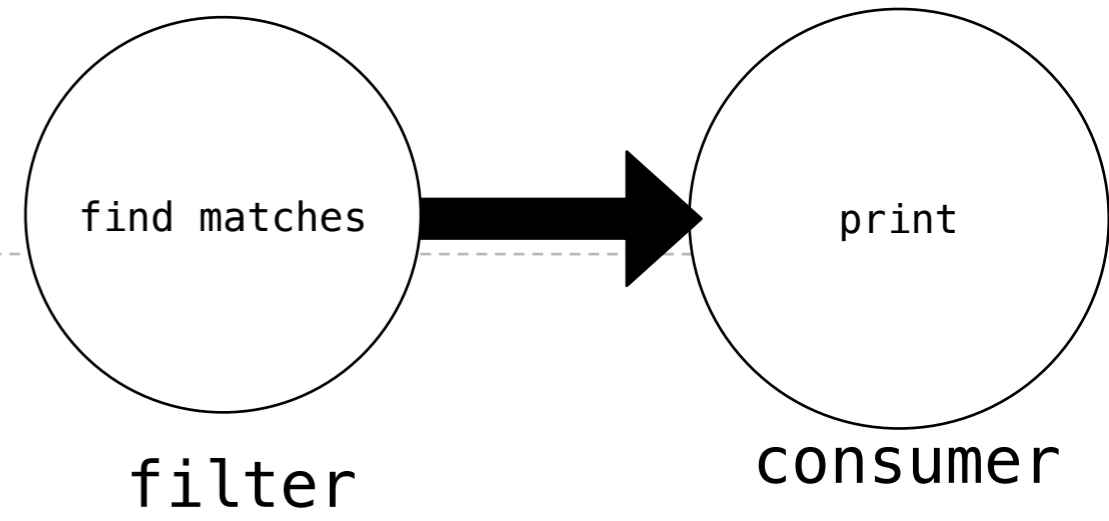
Breaking down match



Breaking down match

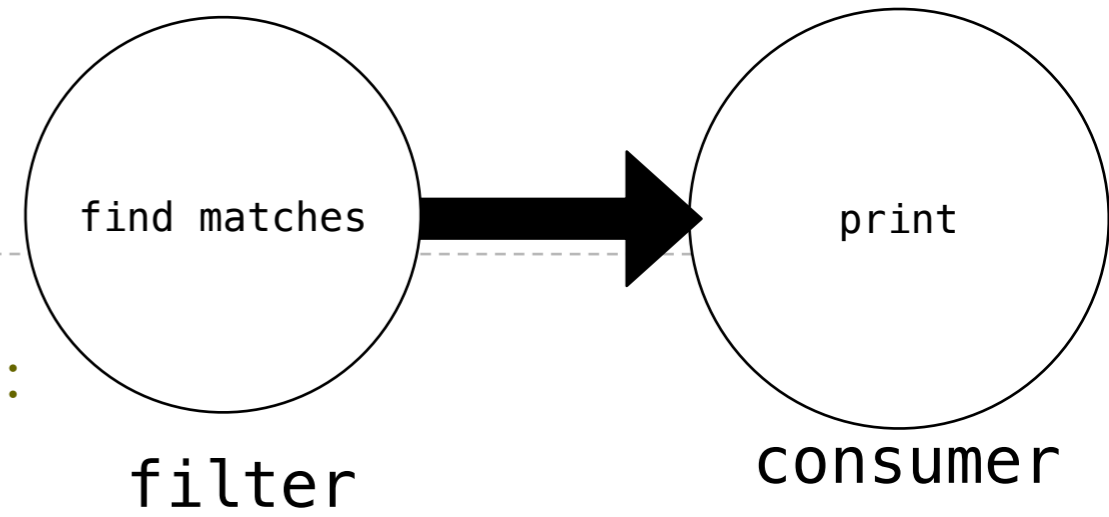


Breaking down match



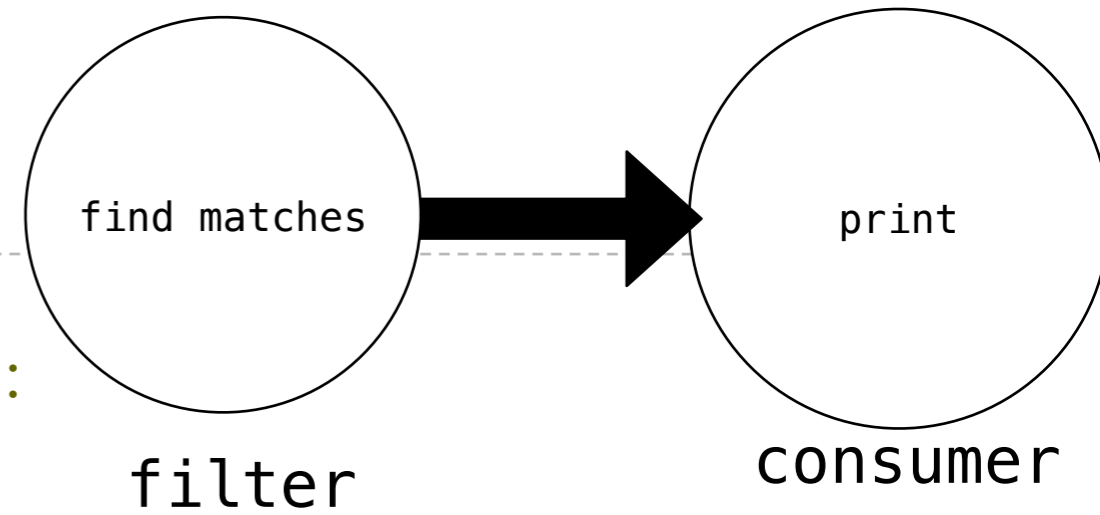
Breaking down match

```
def match_filter(pattern, next_coroutine):
```



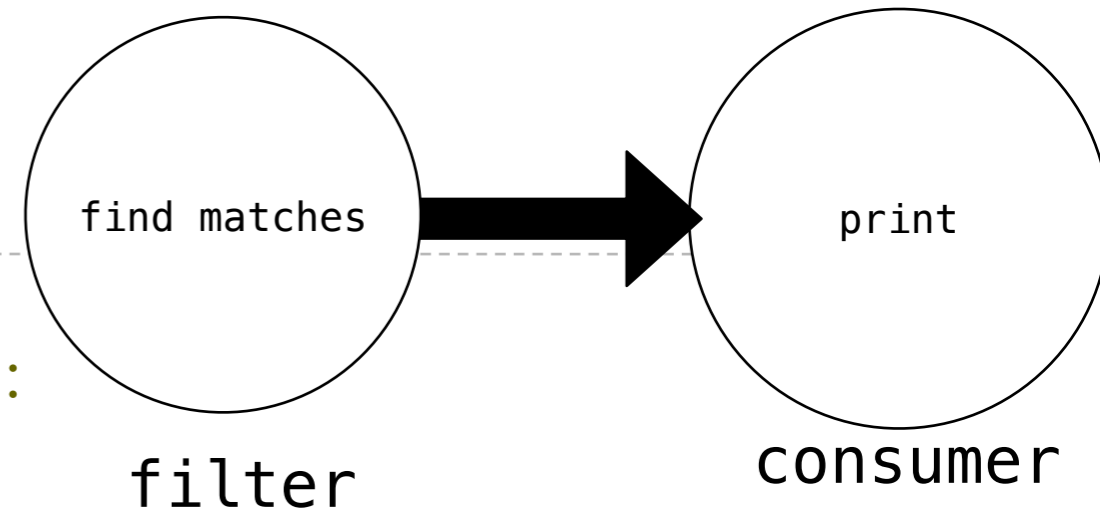
Breaking down match

```
def match_filter(pattern, next_coroutine):  
    print('Looking for ' + pattern)
```



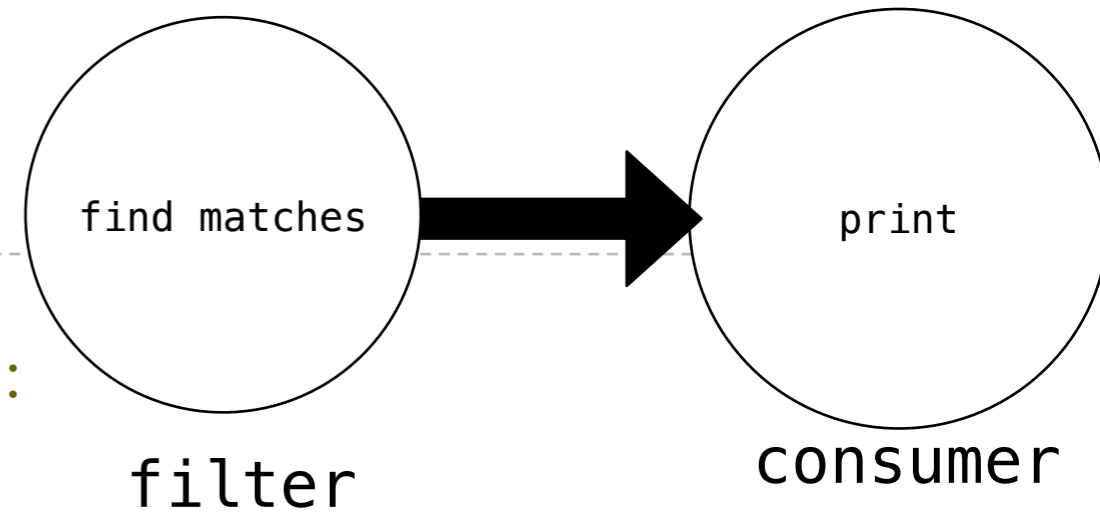
Breaking down match

```
def match_filter(pattern, next_coroutine):  
    print('Looking for ' + pattern)  
    try:
```



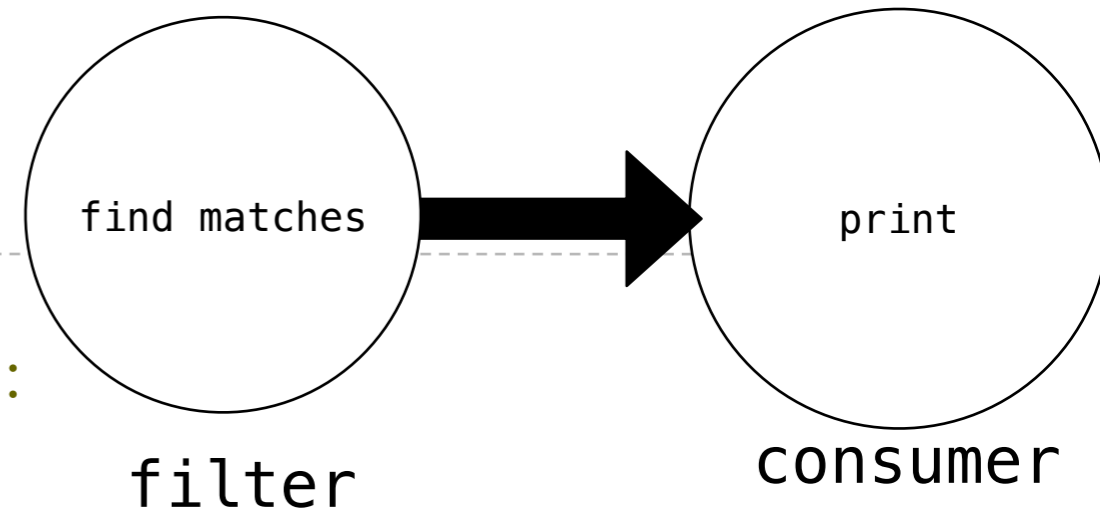
Breaking down match

```
def match_filter(pattern, next_coroutine):  
    print('Looking for ' + pattern)  
    try:  
        while True:
```



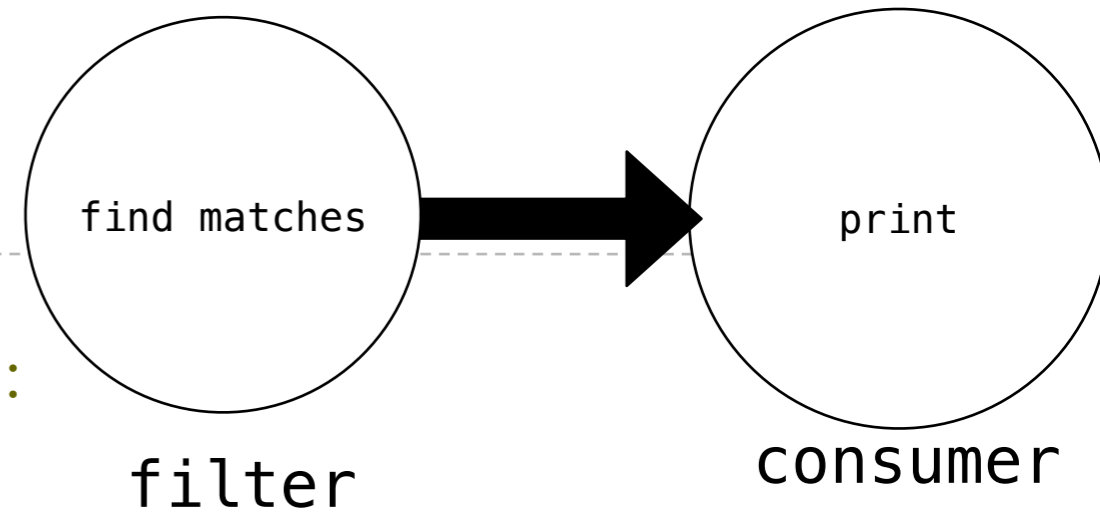
Breaking down match

```
def match_filter(pattern, next_coroutine):  
    print('Looking for ' + pattern)  
    try:  
        while True:  
            s = (yield)
```



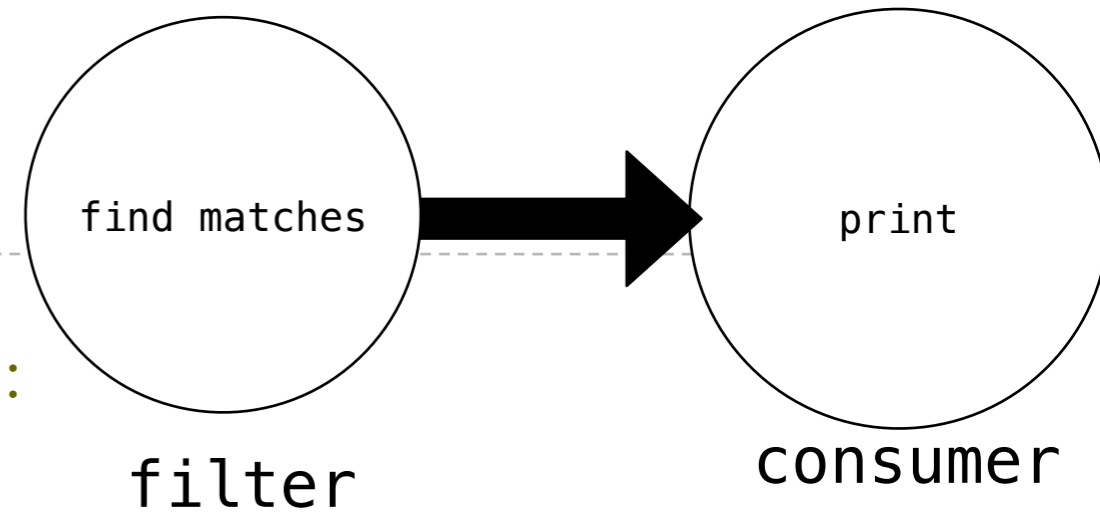
Breaking down match

```
def match_filter(pattern, next_coroutine):  
    print('Looking for ' + pattern)  
    try:  
        while True:  
            s = (yield)  
            if pattern in s:
```



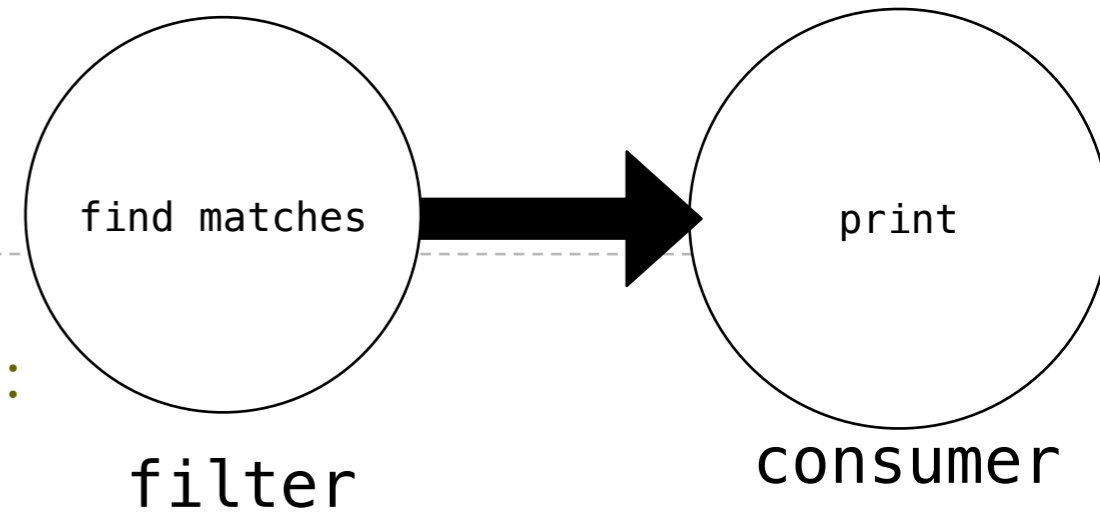
Breaking down match

```
def match_filter(pattern, next_coroutine):  
    print('Looking for ' + pattern)  
    try:  
        while True:  
            s = (yield)  
            if pattern in s:  
                next_coroutine.send(s)
```



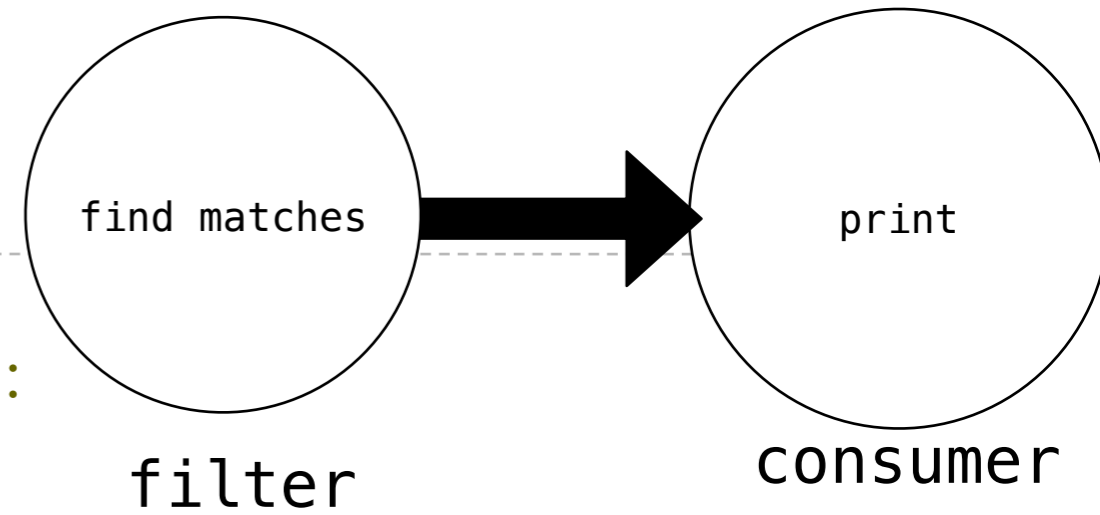
Breaking down match

```
def match_filter(pattern, next_coroutine):  
    print('Looking for ' + pattern)  
    try:  
        while True:  
            s = (yield)  
            if pattern in s:  
                next_coroutine.send(s)  
    except GeneratorExit:
```



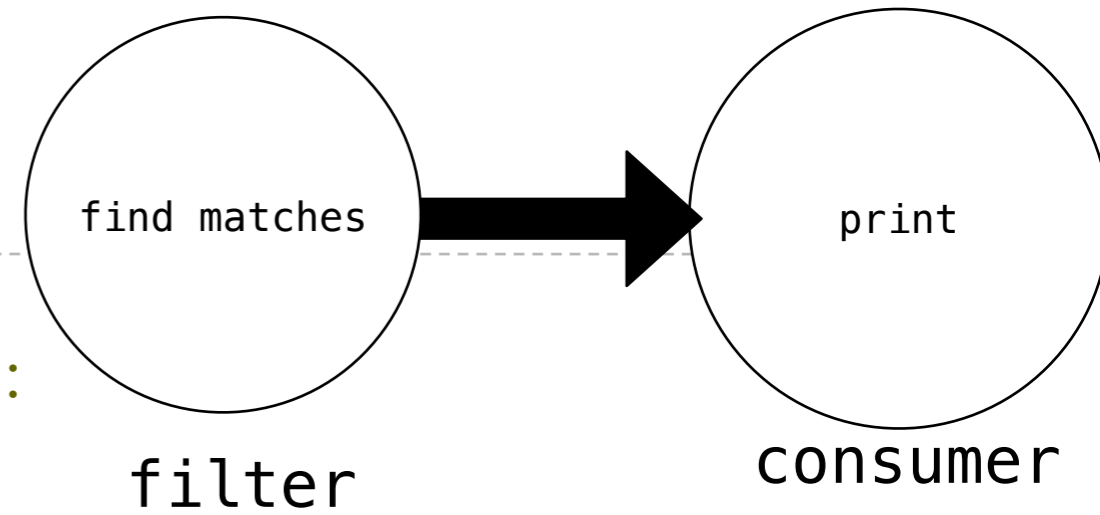
Breaking down match

```
def match_filter(pattern, next_coroutine):
    print('Looking for ' + pattern)
    try:
        while True:
            s = (yield)
            if pattern in s:
                next_coroutine.send(s)
    except GeneratorExit:
        next_coroutine.close()
```



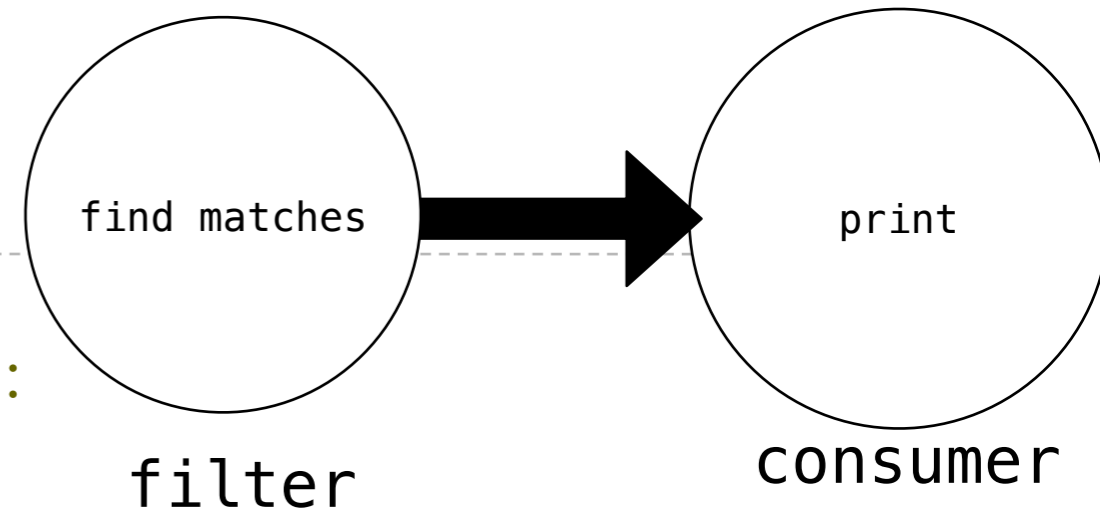
Breaking down match

```
def match_filter(pattern, next_coroutine):  
    print('Looking for ' + pattern)  
    try:  
        while True:  
            s = (yield)  
            if pattern in s:  
                next_coroutine.send(s)  
    except GeneratorExit:  
        next_coroutine.close()
```



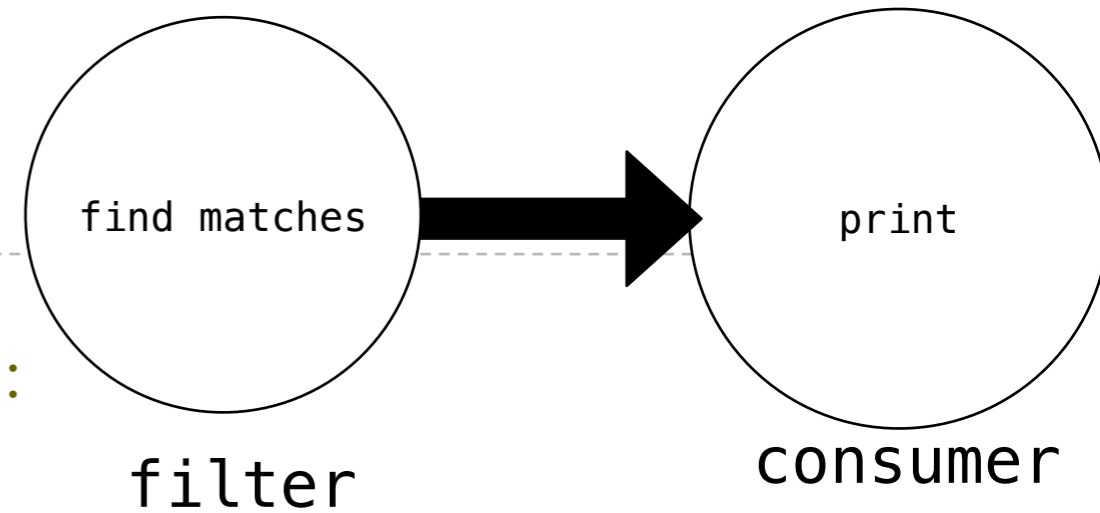
Breaking down match

```
def match_filter(pattern, next_coroutine):  
    print('Looking for ' + pattern)  
    try:  
        while True:  
            s = (yield)  
            if pattern in s:  
                next_coroutine.send(s)  
    except GeneratorExit:  
        next_coroutine.close()
```



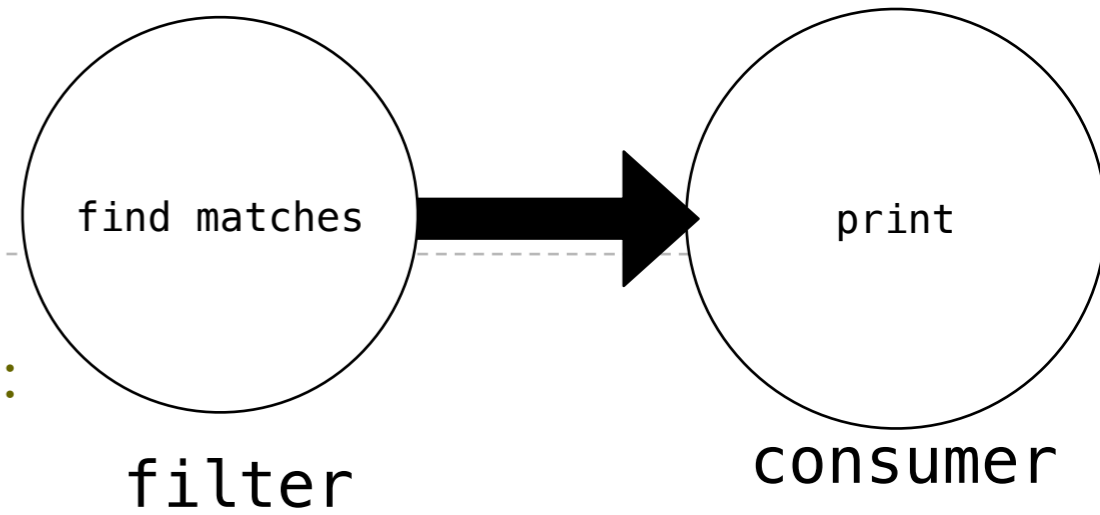
Breaking down match

```
def match_filter(pattern, next_coroutine):
    print('Looking for ' + pattern)
    try:
        while True:
            s = (yield)
            if pattern in s:
                next_coroutine.send(s)
    except GeneratorExit:
        next_coroutine.close()
```



```
def print_consumer():
    print('Preparing to print')
    try:
        while True:
            line = (yield)
            print(line)
    except GeneratorExit:
        print("=== Done ===")
```

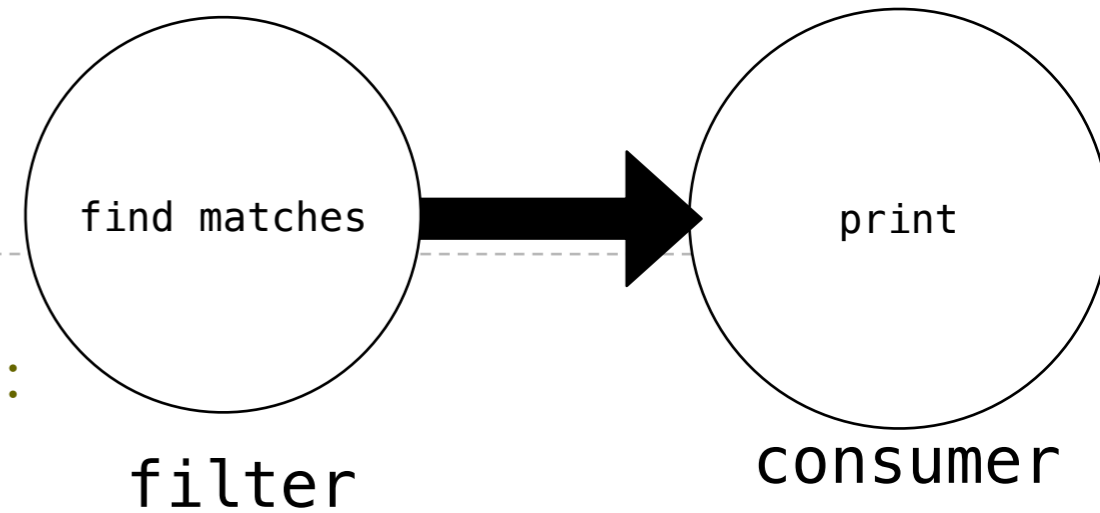
Breaking down match



```
def match_filter(pattern, next_coroutine):  
    print('Looking for ' + pattern)  
    try:  
        while True:  
            s = (yield)  
            if pattern in s:  
                next_coroutine.send(s)  
    except GeneratorExit:  
        next_coroutine.close()
```

```
def print_consumer():  
    print('Preparing to print')  
    try:  
        while True:  
            line = (yield)  
            print(line)  
    except GeneratorExit:  
        print("=== Done ===")
```

Breaking down match

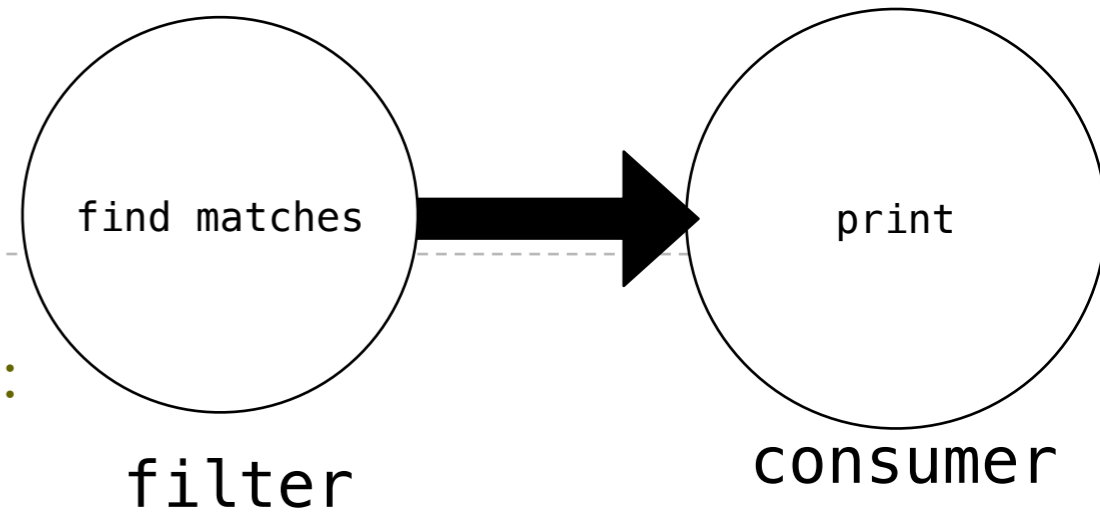


```
def match_filter(pattern, next_coroutine):  
    print('Looking for ' + pattern)  
    try:  
        while True:  
            s = (yield)  
            if pattern in s:  
                next_coroutine.send(s)  
    except GeneratorExit:  
        next_coroutine.close()
```

```
>>> printer = print_consumer()
```

```
def print_consumer():  
    print('Preparing to print')  
    try:  
        while True:  
            line = (yield)  
            print(line)  
    except GeneratorExit:  
        print("=== Done ===")
```

Breaking down match

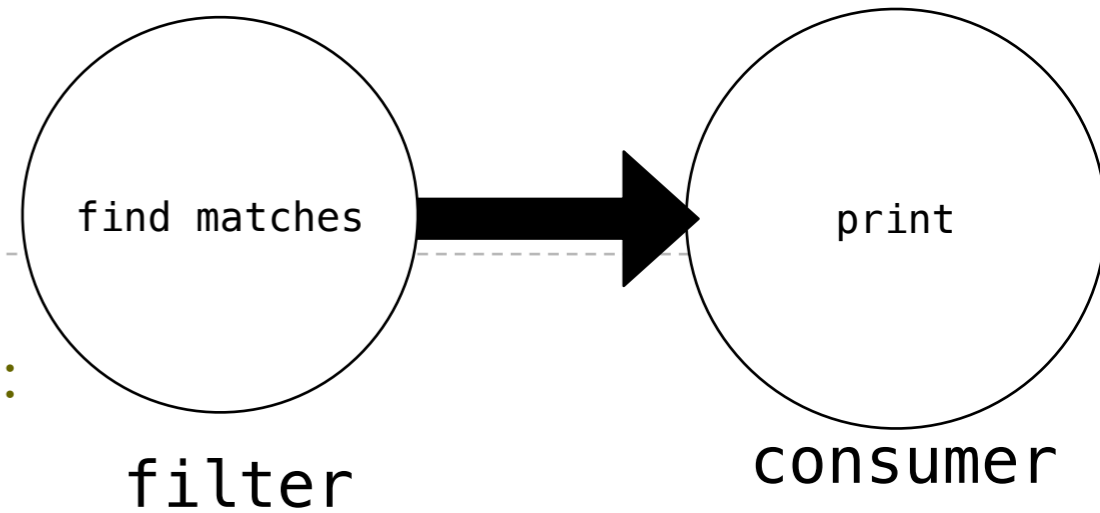


```
def match_filter(pattern, next_coroutine):  
    print('Looking for ' + pattern)  
    try:  
        while True:  
            s = (yield)  
            if pattern in s:  
                next_coroutine.send(s)  
    except GeneratorExit:  
        next_coroutine.close()
```

```
>>> printer = print_consumer()  
>>> printer.__next__()
```

```
def print_consumer():  
    print('Preparing to print')  
    try:  
        while True:  
            line = (yield)  
            print(line)  
    except GeneratorExit:  
        print("=== Done ===")
```

Breaking down match



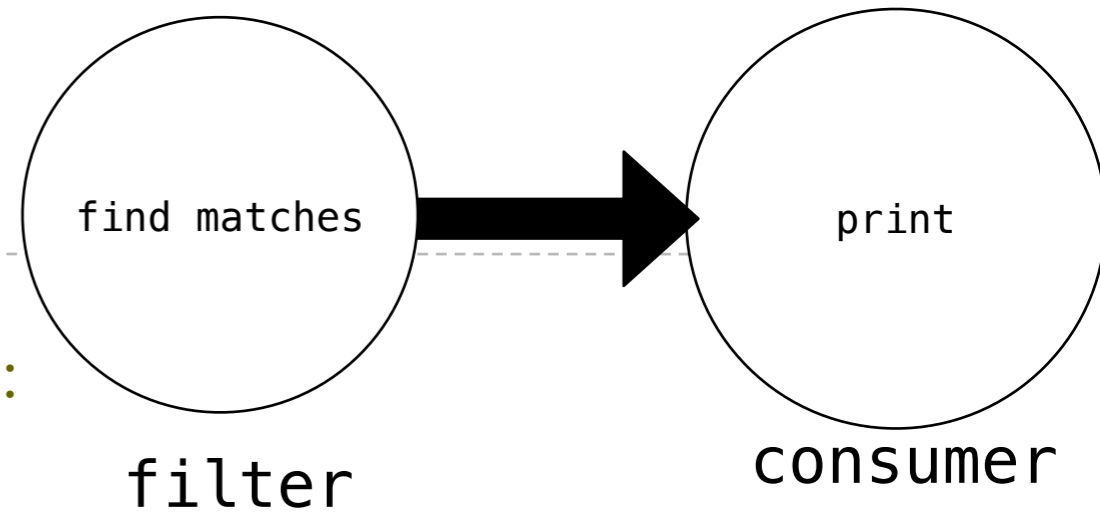
```
def match_filter(pattern, next_coroutine):  
    print('Looking for ' + pattern)  
    try:  
        while True:  
            s = (yield)  
            if pattern in s:  
                next_coroutine.send(s)  
    except GeneratorExit:  
        next_coroutine.close()
```

next_coroutine.send(s)

```
def print_consumer():  
    print('Preparing to print')  
    try:  
        while True:  
            line = (yield)  
            print(line)  
    except GeneratorExit:  
        print("=== Done ===")
```

```
>>> printer = print_consumer()  
>>> printer.__next__()  
'Preparing to print'
```


Breaking down match

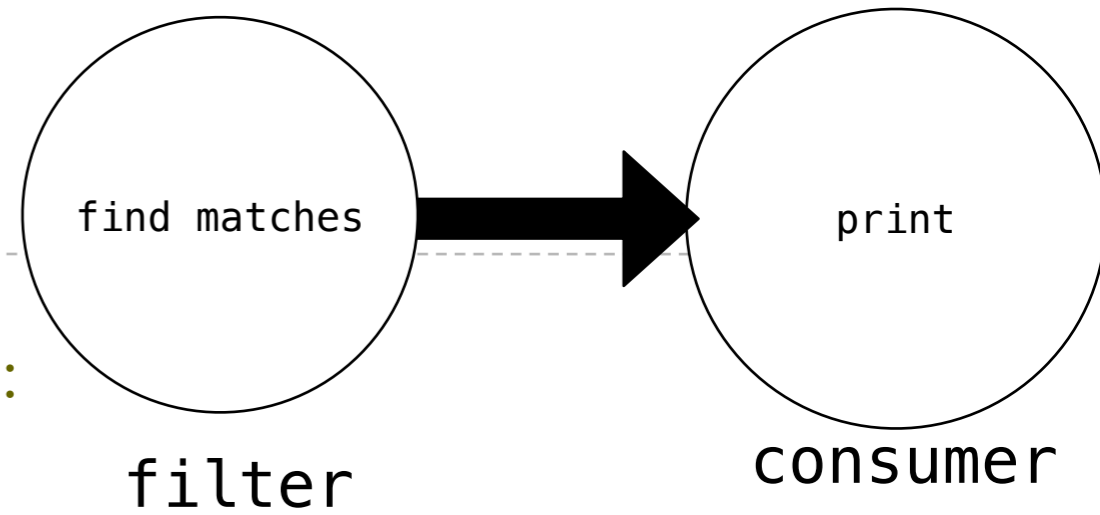


```
def match_filter(pattern, next_coroutine):  
    print('Looking for ' + pattern)  
    try:  
        while True:  
            s = (yield)  
            if pattern in s:  
                next_coroutine.send(s)  
    except GeneratorExit:  
        next_coroutine.close()
```

```
>>> printer = print_consumer()  
>>> printer.__next__()  
'Preparing to print'  
>>> matcher = match_filter('pend', printer)
```

```
def print_consumer():  
    print('Preparing to print')  
    try:  
        while True:  
            line = (yield)  
            print(line)  
    except GeneratorExit:  
        print("=== Done ===")
```

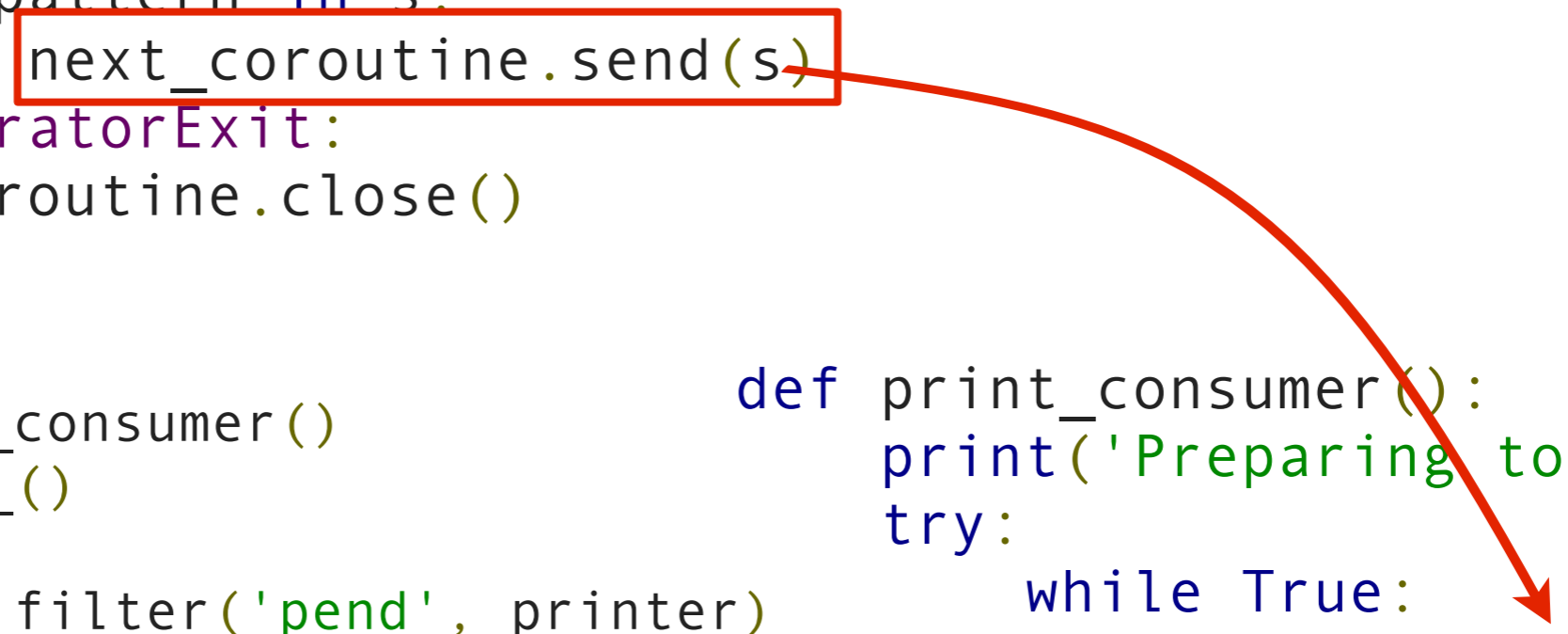
Breaking down match



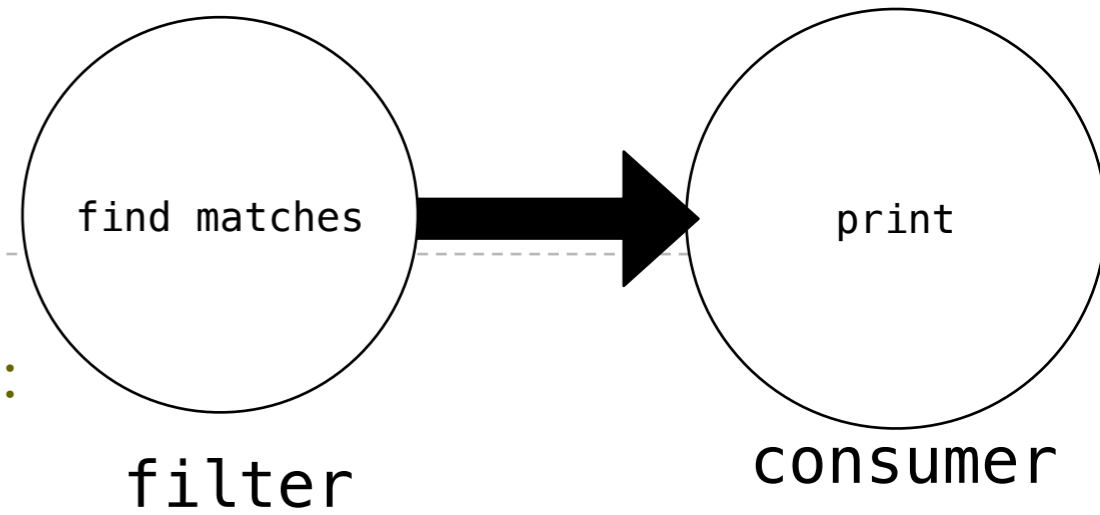
```
def match_filter(pattern, next_coroutine):  
    print('Looking for ' + pattern)  
    try:  
        while True:  
            s = (yield)  
            if pattern in s:  
                next_coroutine.send(s)  
    except GeneratorExit:  
        next_coroutine.close()
```

```
>>> printer = print_consumer()  
>>> printer.__next__()  
'Preparing to print'  
>>> matcher = match_filter('pend', printer)  
>>> matcher.__next__()
```

```
def print_consumer():  
    print('Preparing to print')  
    try:  
        while True:  
            line = (yield)  
            print(line)  
    except GeneratorExit:  
        print("=== Done ===")
```



Breaking down match

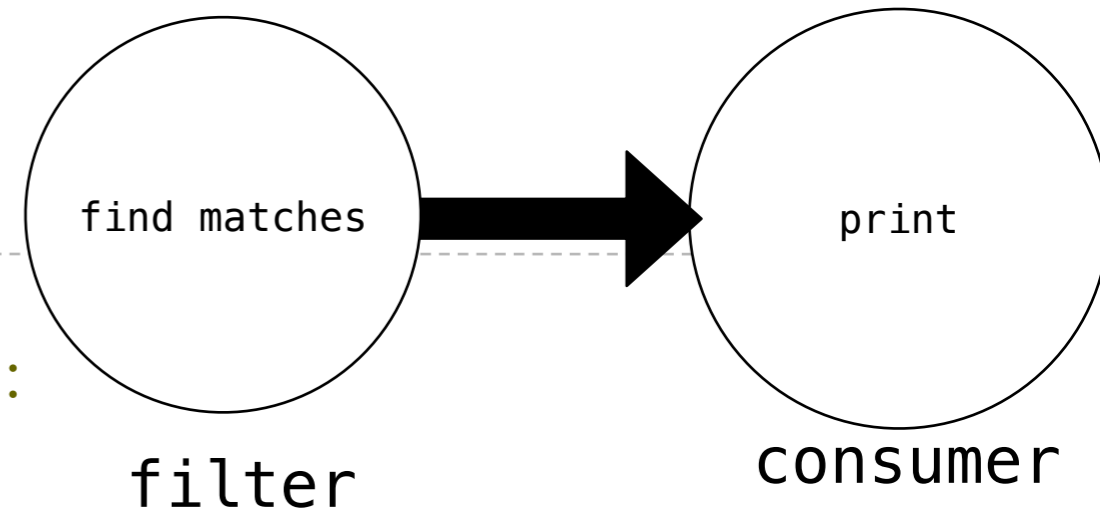


```
def match_filter(pattern, next_coroutine):  
    print('Looking for ' + pattern)  
    try:  
        while True:  
            s = (yield)  
            if pattern in s:  
                next_coroutine.send(s)  
    except GeneratorExit:  
        next_coroutine.close()
```

```
>>> printer = print_consumer()  
>>> printer.__next__()  
'Preparing to print'  
>>> matcher = match_filter('pend', printer)  
>>> matcher.__next__()  
'Looking for pend'
```

```
def print_consumer():  
    print('Preparing to print')  
    try:  
        while True:  
            line = (yield)  
            print(line)  
    except GeneratorExit:  
        print("=== Done ===")
```

Breaking down match

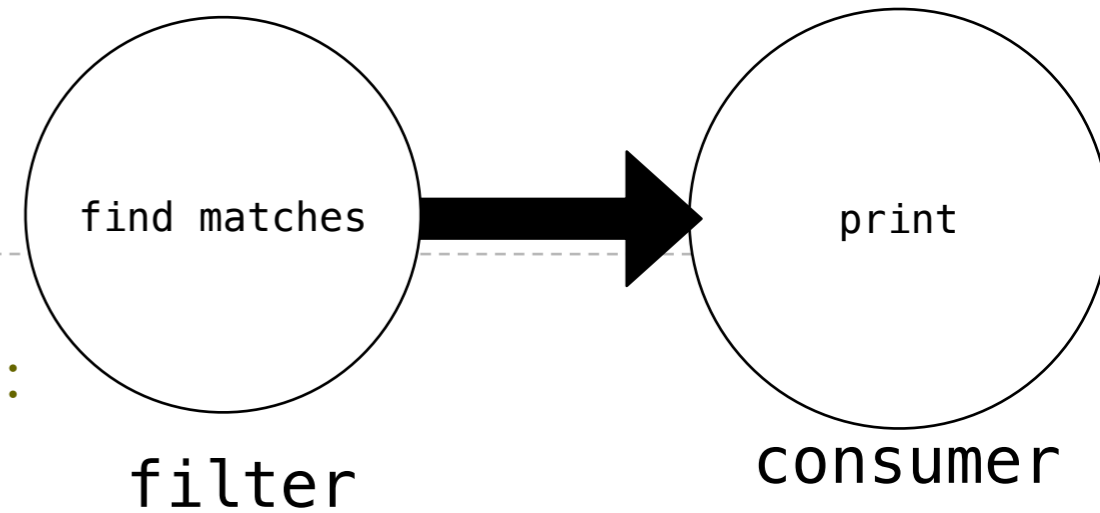


```
def match_filter(pattern, next_coroutine):  
    print('Looking for ' + pattern)  
    try:  
        while True:  
            s = (yield)  
            if pattern in s:  
                next_coroutine.send(s)  
    except GeneratorExit:  
        next_coroutine.close()
```

```
>>> printer = print_consumer()  
>>> printer.__next__()  
'Preparing to print'  
>>> matcher = match_filter('pend', printer)  
>>> matcher.__next__()  
'Looking for pend'  
>>> text = 'Commending spending is offending'
```

```
def print_consumer():  
    print('Preparing to print')  
    try:  
        while True:  
            line = (yield)  
            print(line)  
    except GeneratorExit:  
        print("=== Done ===")
```

Breaking down match

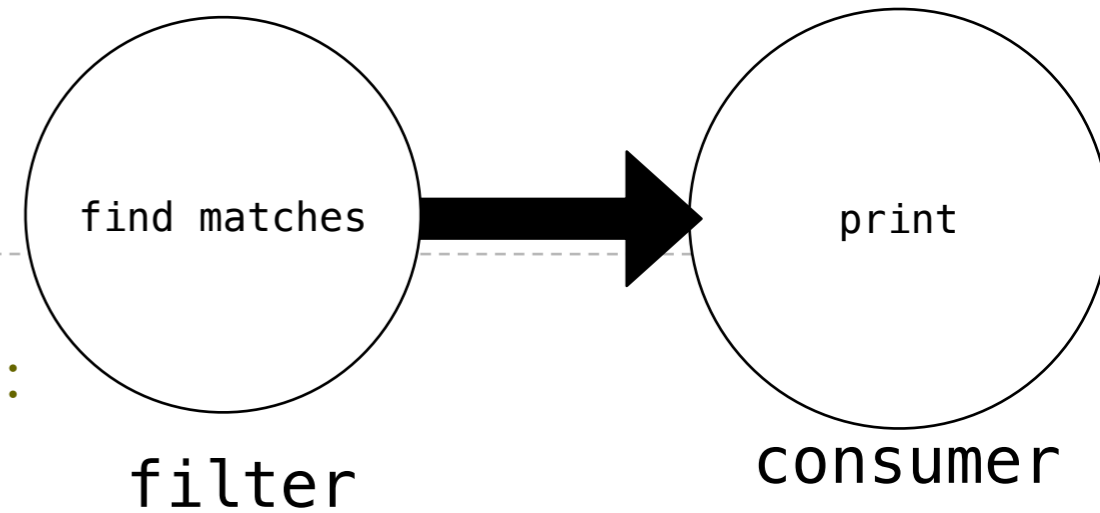


```
def match_filter(pattern, next_coroutine):  
    print('Looking for ' + pattern)  
    try:  
        while True:  
            s = (yield)  
            if pattern in s:  
                next_coroutine.send(s)  
    except GeneratorExit:  
        next_coroutine.close()
```

```
>>> printer = print_consumer()  
>>> printer.__next__()  
'Preparing to print'  
>>> matcher = match_filter('pend', printer)  
>>> matcher.__next__()  
'Looking for pend'  
>>> text = 'Commending spending is offending'  
>>> read(text, matcher)
```

```
def print_consumer():  
    print('Preparing to print')  
    try:  
        while True:  
            line = (yield)  
            print(line)  
    except GeneratorExit:  
        print("=== Done ===")
```

Breaking down match

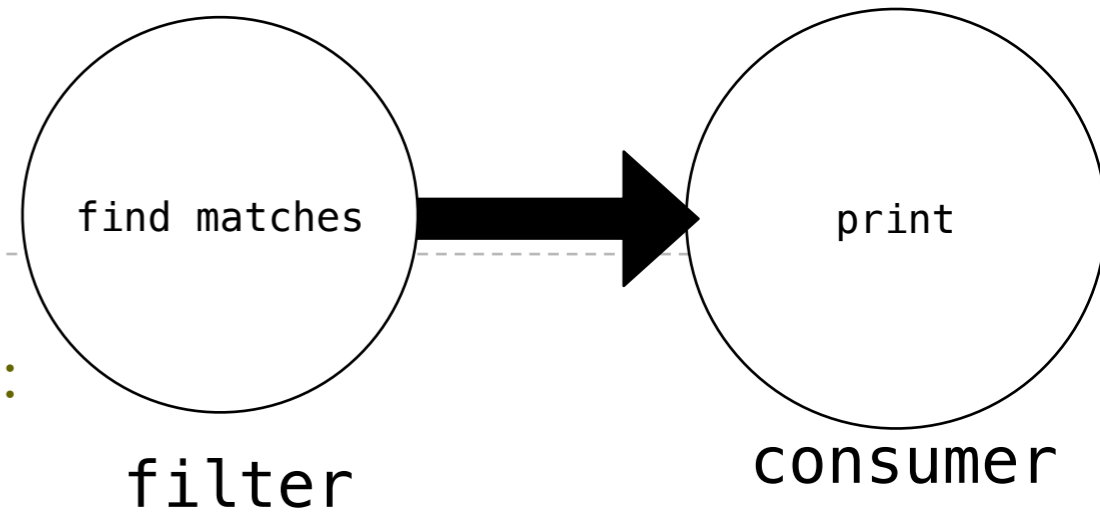


```
def match_filter(pattern, next_coroutine):  
    print('Looking for ' + pattern)  
    try:  
        while True:  
            s = (yield)  
            if pattern in s:  
                next_coroutine.send(s)  
    except GeneratorExit:  
        next_coroutine.close()
```

```
>>> printer = print_consumer()  
>>> printer.__next__()  
'Preparing to print'  
>>> matcher = match_filter('pend', printer)  
>>> matcher.__next__()  
'Looking for pend'  
>>> text = 'Commending spending is offending'  
>>> read(text, matcher)  
'spending'
```

```
def print_consumer():  
    print('Preparing to print')  
    try:  
        while True:  
            line = (yield)  
            print(line)  
    except GeneratorExit:  
        print("=== Done ===")
```

Breaking down match

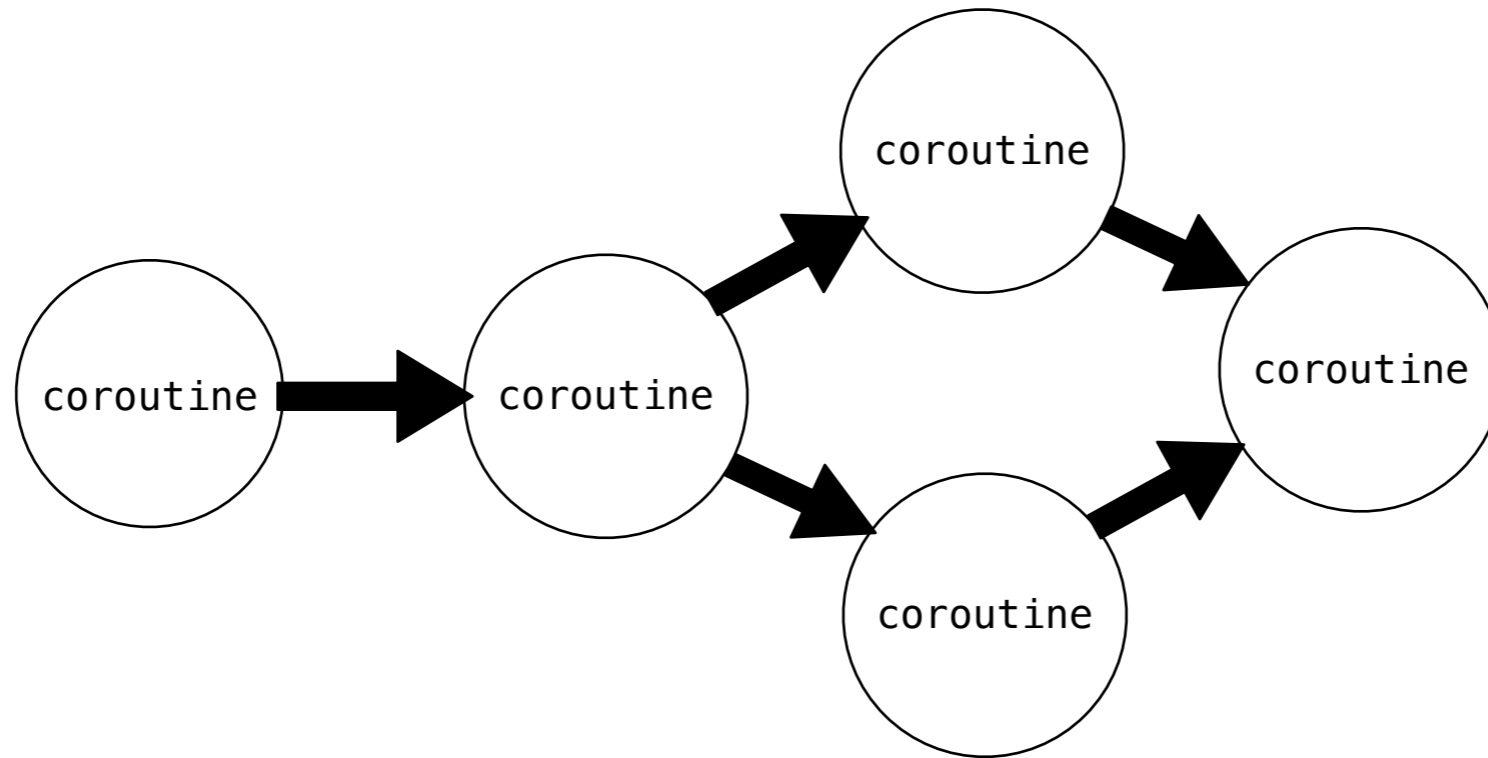


```
def match_filter(pattern, next_coroutine):  
    print('Looking for ' + pattern)  
    try:  
        while True:  
            s = (yield)  
            if pattern in s:  
                next_coroutine.send(s)  
    except GeneratorExit:  
        next_coroutine.close()
```

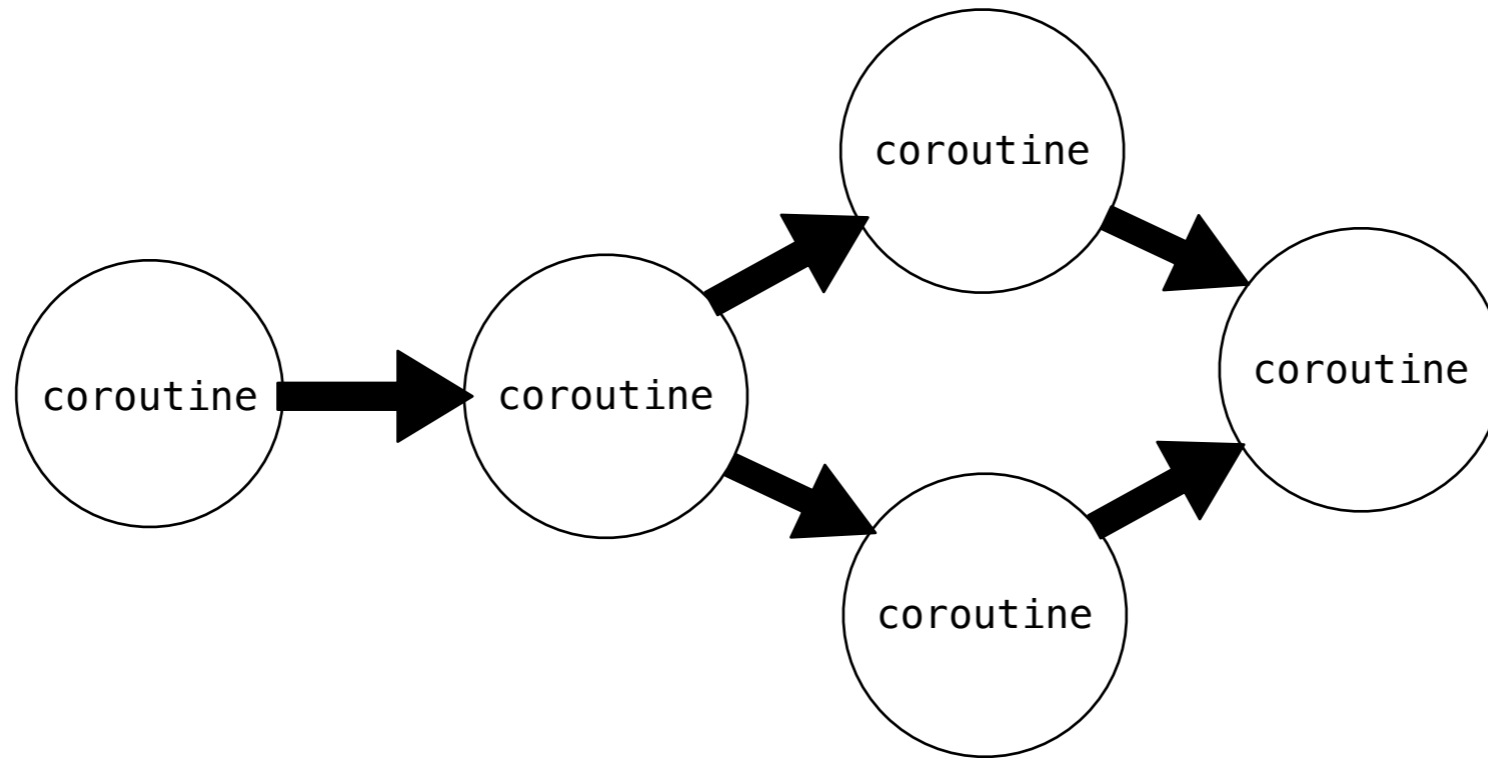
```
>>> printer = print_consumer()  
>>> printer.__next__()  
'Preparing to print'  
>>> matcher = match_filter('pend', printer)  
>>> matcher.__next__()  
'Looking for pend'  
>>> text = 'Commending spending is offending'  
>>> read(text, matcher)  
'spending'  
'=== Done ==='
```

```
def print_consumer():  
    print('Preparing to print')  
    try:  
        while True:  
            line = (yield)  
            print(line)  
    except GeneratorExit:  
        print("=== Done ===")
```

Multitasking



Multitasking



We do not need to be restricted to just one next step

Read-to-many

Read-to-many

```
def read(text, next_coroutine):
```

Read-to-many

```
def read(text, next_coroutine):  
    for word in text.split():
```

Read-to-many

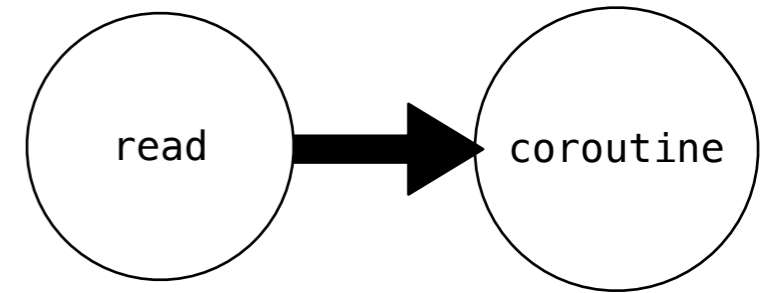
```
def read(text, next_coroutine):  
    for word in text.split():  
        next_coroutine.send(word)
```

Read-to-many

```
def read(text, next_coroutine):  
    for word in text.split():  
        next_coroutine.send(word)  
    next_coroutine.close()
```

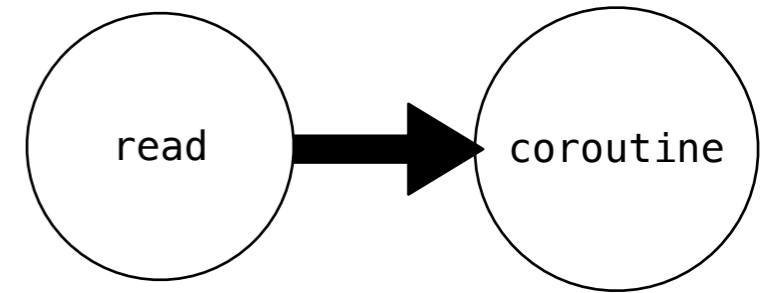
Read-to-many

```
def read(text, next_coroutine):  
    for word in text.split():  
        next_coroutine.send(word)  
    next_coroutine.close()
```



Read-to-many

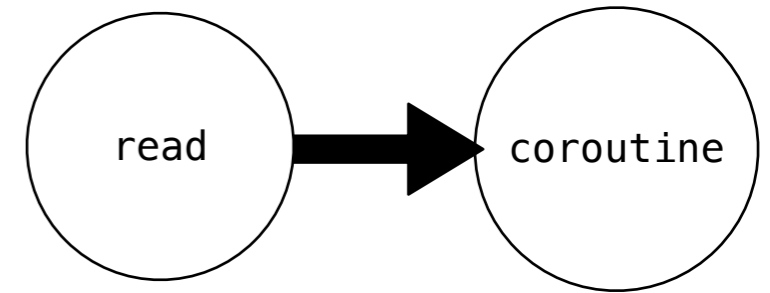
```
def read(text, next_coroutine):  
    for word in text.split():  
        next_coroutine.send(word)  
    next_coroutine.close()
```



```
def read_to_many(text, coroutines):  
    for word in text.split():  
        for coroutine in coroutines:  
            coroutine.send(word)  
    for coroutine in coroutines:  
        coroutine.close()
```


Read-to-many

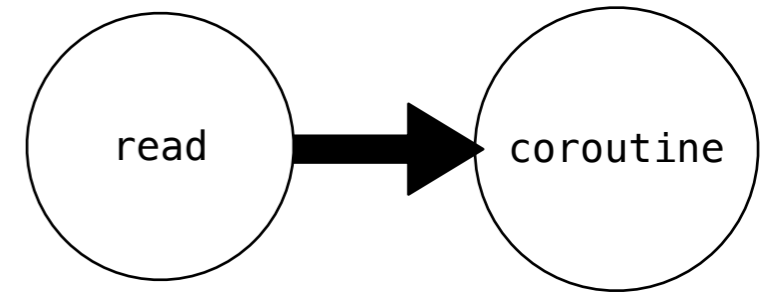
```
def read(text, next_coroutine):  
    for word in text.split():  
        next_coroutine.send(word)  
    next_coroutine.close()
```



```
def read_to_many(text, coroutines):  
    for word in text.split():  
        for coroutine in coroutines:  
            coroutine.send(word)  
    for coroutine in coroutines:  
        coroutine.close()
```

Read-to-many

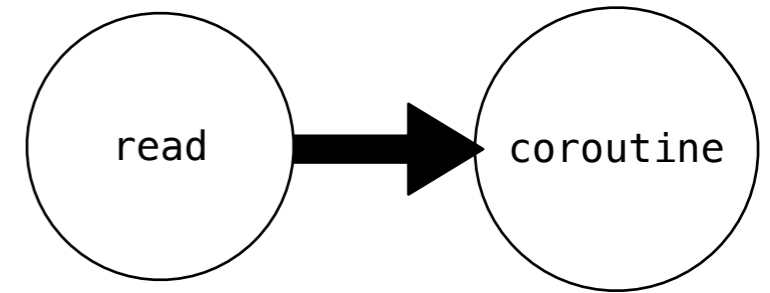
```
def read(text, next_coroutine):  
    for word in text.split():  
        next_coroutine.send(word)  
    next_coroutine.close()
```



```
def read_to_many(text, coroutines):  
    for word in text.split():  
        for coroutine in coroutines:  
            coroutine.send(word)  
    for coroutine in coroutines:  
        coroutine.close()
```

Read-to-many

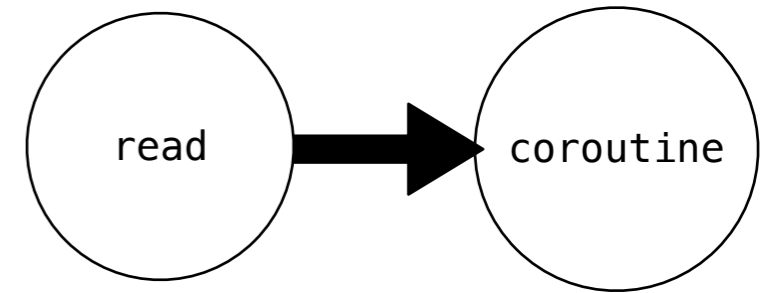
```
def read(text, next_coroutine):  
    for word in text.split():  
        next_coroutine.send(word)  
    next_coroutine.close()
```



```
def read_to_many(text, coroutines):  
    for word in text.split():  
        for coroutine in coroutines:  
            coroutine.send(word)  
    for coroutine in coroutines:  
        coroutine.close()
```

Read-to-many

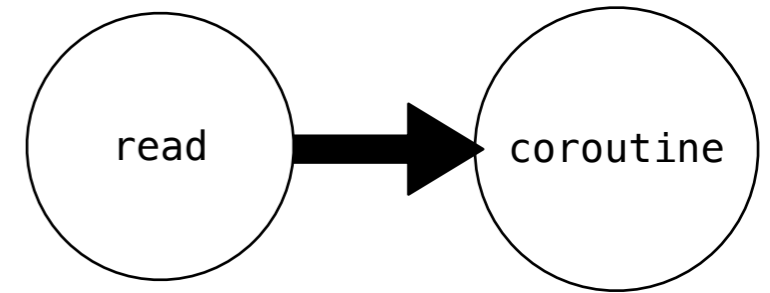
```
def read(text, next_coroutine):  
    for word in text.split():  
        next_coroutine.send(word)  
    next_coroutine.close()
```



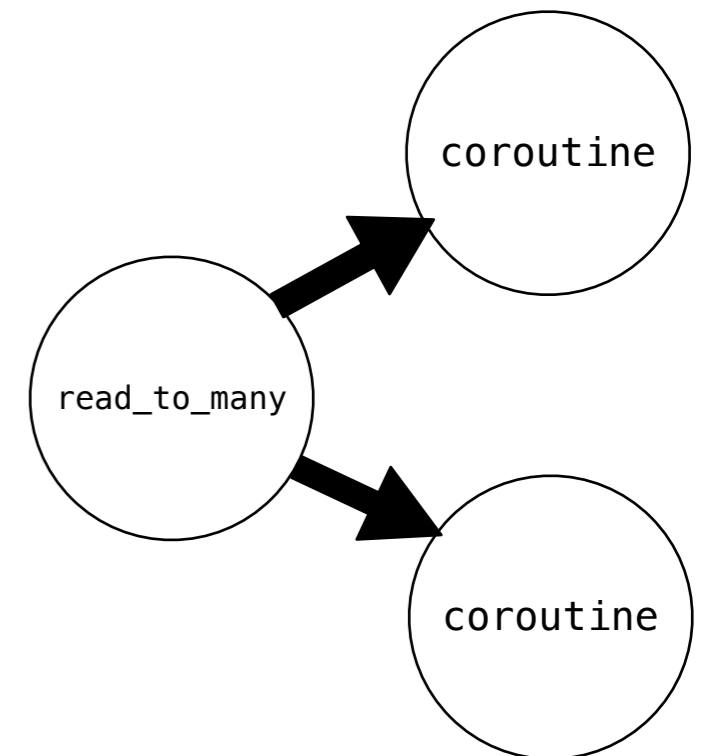
```
def read_to_many(text, coroutines):  
    for word in text.split():  
        for coroutine in coroutines:  
            coroutine.send(word)  
    for coroutine in coroutines:  
        coroutine.close()
```

Read-to-many

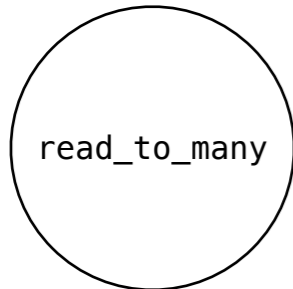
```
def read(text, next_coroutine):  
    for word in text.split():  
        next_coroutine.send(word)  
    next_coroutine.close()
```



```
def read_to_many(text, coroutines):  
    for word in text.split():  
        for coroutine in coroutines:  
            coroutine.send(word)  
    for coroutine in coroutines:  
        coroutine.close()
```



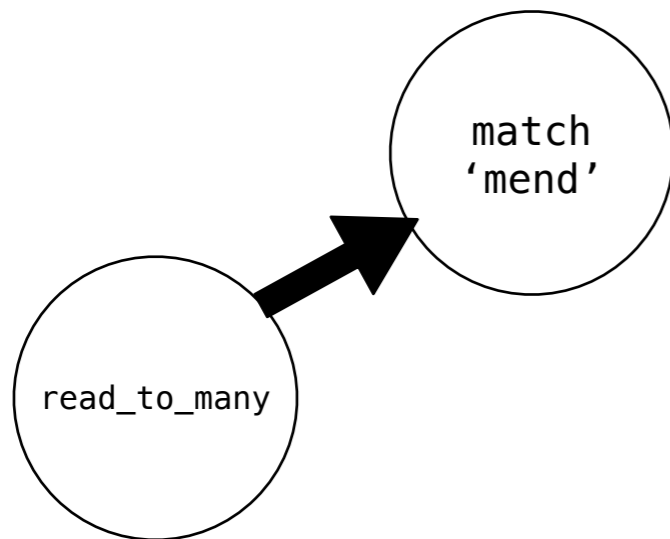
Matching multiple patterns



Any questions?

```
>>> printer = print_consumer()
>>> printer.__next__()
'Preparing to print'
>>> m = match_filter('mend', printer)
>>> m.__next__()
'Looking for mend'
>>> p = match_filter("pe", printer)
>>> p.__next__()
'Looking for pe'
>>> read_to_many(text, [m, p])
'Commending'
'spending'
'people'
'pending'
'=== Done ==='
```

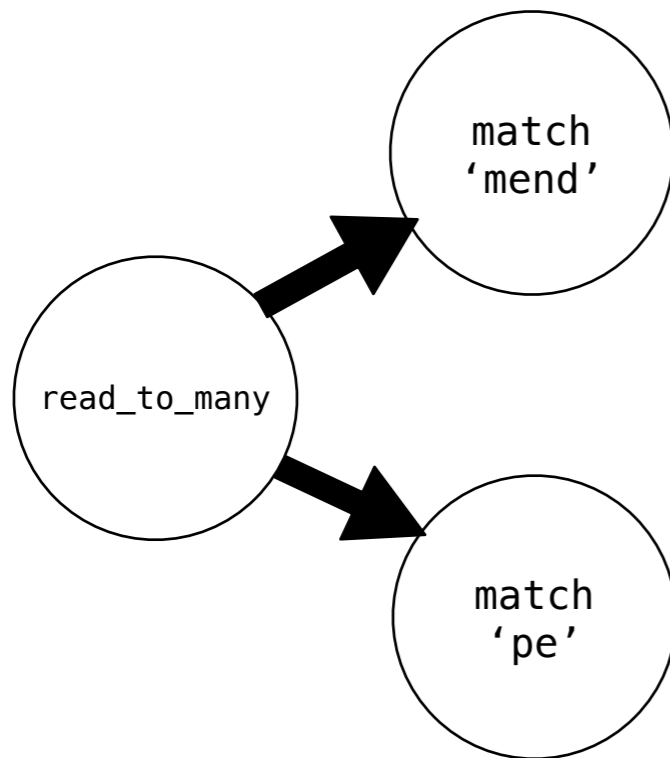
Matching multiple patterns



Any questions?

```
>>> printer = print_consumer()
>>> printer.__next__()
'Preparing to print'
>>> m = match_filter('mend', printer)
>>> m.__next__()
'Looking for mend'
>>> p = match_filter("pe", printer)
>>> p.__next__()
'Looking for pe'
>>> read_to_many(text, [m, p])
'Commending'
'spending'
'people'
'pending'
'=== Done ==='
```

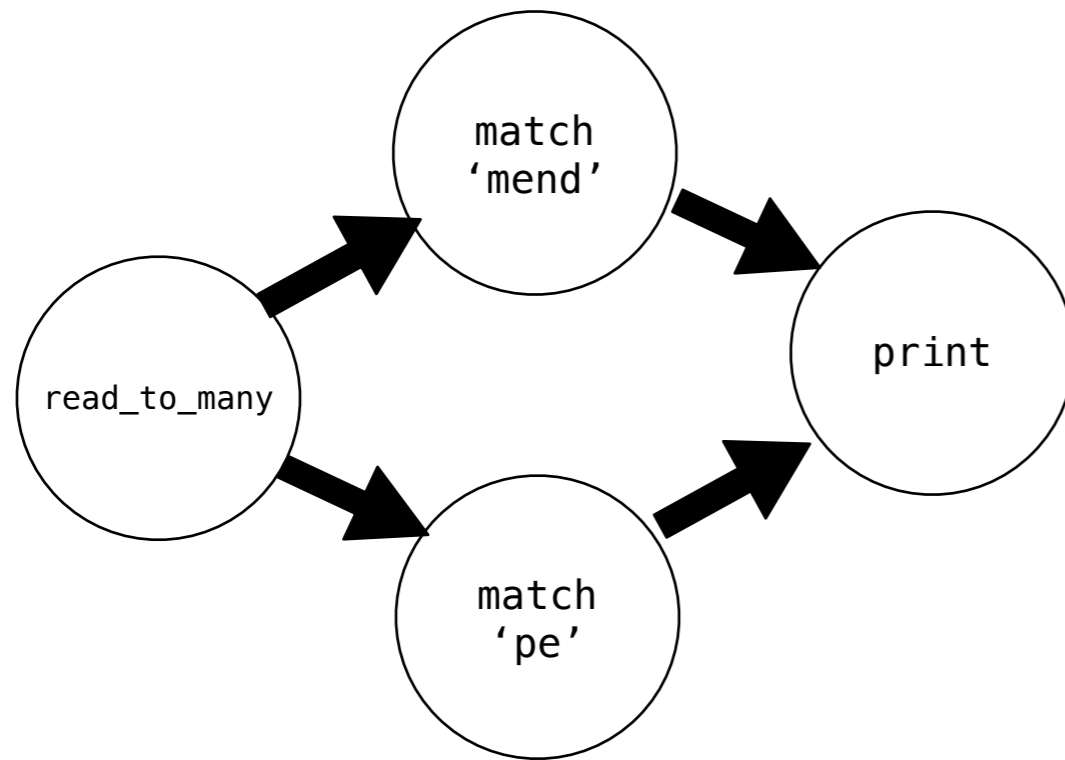
Matching multiple patterns



Any questions?

```
>>> printer = print_consumer()
>>> printer.__next__()
'Preparing to print'
>>> m = match_filter('mend', printer)
>>> m.__next__()
'Looking for mend'
>>> p = match_filter("pe", printer)
>>> p.__next__()
'Looking for pe'
>>> read_to_many(text, [m, p])
'Commending'
'spending'
'people'
'pending'
'=== Done ==='
```


Matching multiple patterns



Any questions?

```
>>> printer = print_consumer()
>>> printer.__next__()
'Preparing to print'
>>> m = match_filter('mend', printer)
>>> m.__next__()
'Looking for mend'
>>> p = match_filter("pe", printer)
>>> p.__next__()
'Looking for pe'
>>> read_to_many(text, [m, p])
'Commending'
'spending'
'people'
'pending'
'=== Done ==='
```

NEXT TIME

MAP REDUCE

http://www.infobarrel.com/Top_10_Tips_For_Snowboard_Beginners