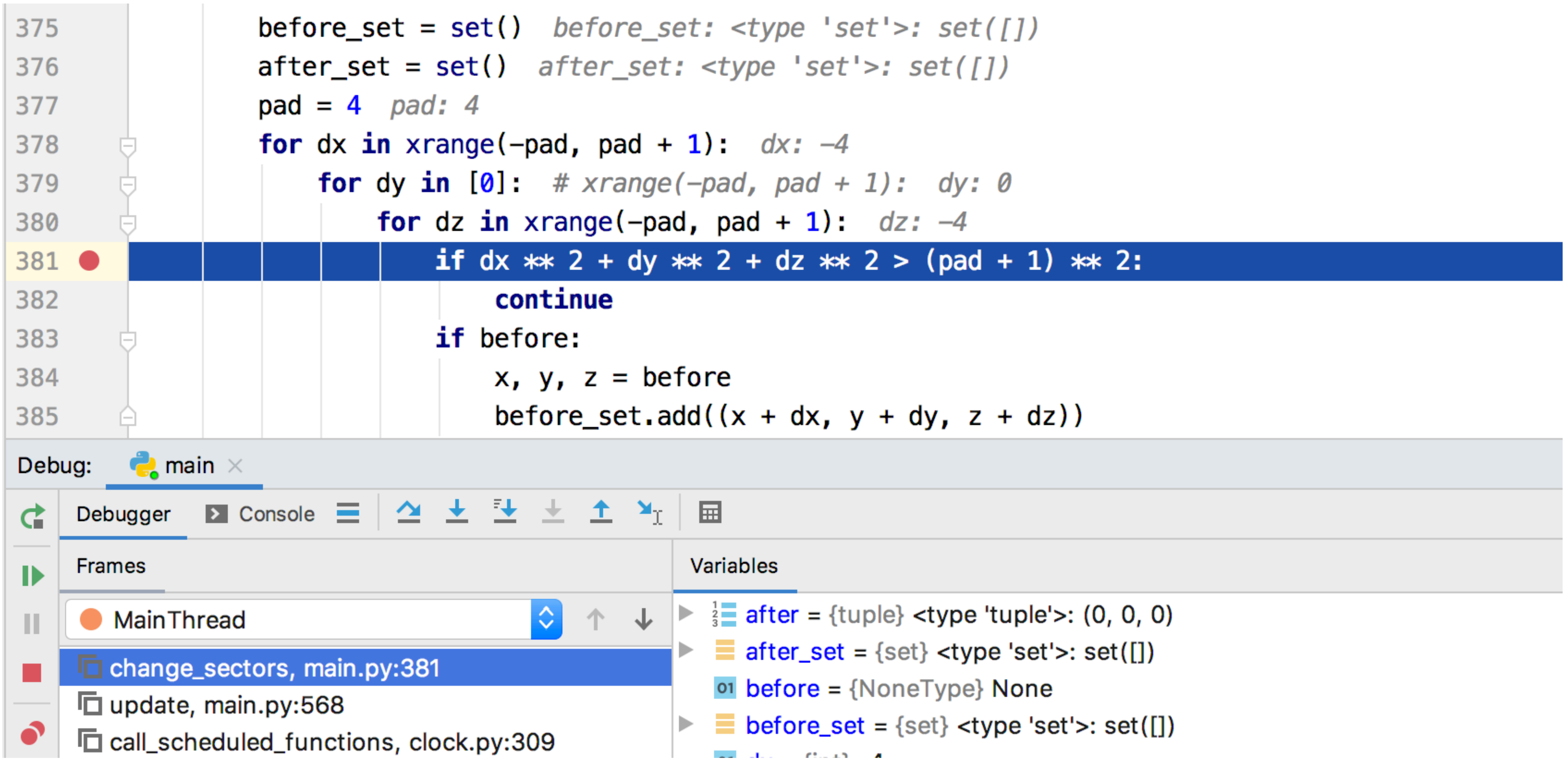
Visual Debugger for Jupyter Notebooks: Myth or Reality?

Elizaveta Shashkova
EuroPython 2019

About Me

- Software Developer at JetBrains, PyCharm IDE
- Debugger and Data Science tools
- @lisa_shashkova

Visual Debugger



```
        before_set = set()  before_set: <type 'set'>: set([])
        after_set = set()  after_set: <type 'set'>: set([])
        pad = 4  pad: 4
        for dx in xrange(-pad, pad + 1):  dx: -4
            for dy in [0]:  # xrange(-pad, pad + 1):  dy: 0
                for dz in xrange(-pad, pad + 1):  dz: -4
                    if dx ** 2 + dy ** 2 + dz ** 2 > (pad + 1) ** 2:
                        continue
                    if before:
                        x, y, z = before
                        before_set.add((x + dx, y + dy, z + dz))
```

```
Frames
MainThread
change_sectors, main.py:381
update, main.py:568
call_scheduled_functions, clock.py:309

Variables
after = {tuple} <type 'tuple'>: (0, 0, 0)
after_set = {set} <type 'set'>: set([])
before = {NoneType} None
before_set = {set} <type 'set'>: set([])
```

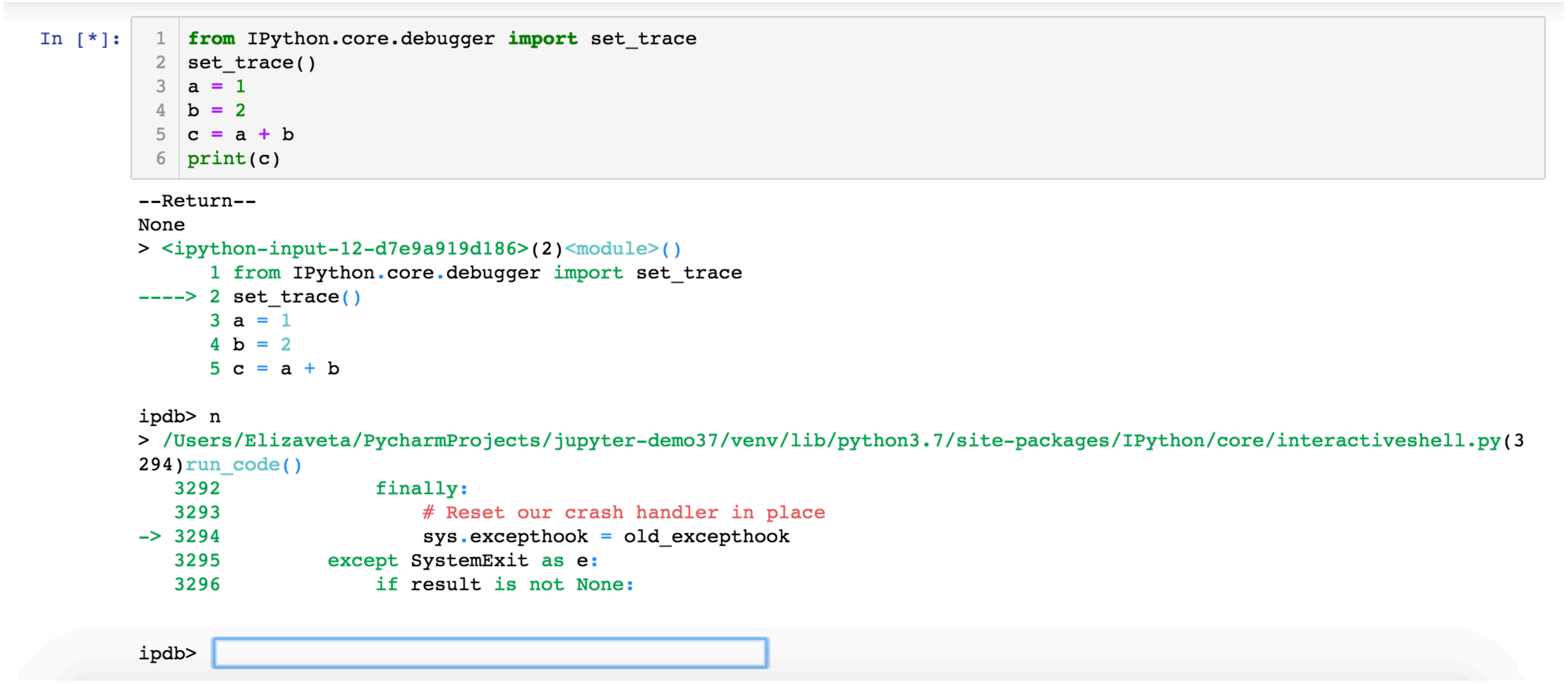
Jupyter Notebooks

- Popular scientific tool
- File is a sequence of cells

Jupyter Notebooks Debug

- Logging with print statements
- Command-line debugger ipdb

Jupyter Notebooks Debug



```
In [*]:
from IPython.core.debugger import set_trace
set_trace()
a = 1
b = 2
c = a + b
print(c)
```

```
--Return--
None
> <ipython-input-12-d7e9a919d186>(2)<module>()
      1 from IPython.core.debugger import set_trace
----> 2 set_trace()
      3 a = 1
      4 b = 2
      5 c = a + b

ipdb> n
> /Users/Elizaveta/PycharmProjects/jupyter-demo37/venv/lib/python3.7/site-packages/IPython/core/interactiveshell.py(3
294)run_code()
   3292                 finally:
   3293                     # Reset our crash handler in place
-> 3294                     sys.excepthook = old_excepthook
   3295             except SystemExit as e:
   3296                 if result is not None:

ipdb>
```

Myth or Reality?

Myth or Reality?

Contents

Tracing Function

```python
def tracefunc(frame, event, arg):
    print(frame.f_lineno, event)
    return tracefunc


sys.settrace(tracefunc)
```

Tracing Function

```python
def greet_neighbors():
    planets = ["Mars", "Venus"]
    for p in planets:
        print(f"Hi {p}!")
    return len(planets)


sys.settrace(tracefunc)
greet_neighbors()
```

Tracing Function

```
1   def greet_neighbors():
2       planets = ["Mars", "Venus"]
3       for p in planets:
4           print(f"Hi {p}!")
5       return len(planets)
6
7
8   sys.settrace(tracefunc)
9   greet_neighbors()
```

```
1 call
```

Tracing Function

```
1   def greet_neighbors():
2       planets = ["Mars", "Venus"]
3       for p in planets:
4           print(f"Hi {p}!")
5       return len(planets)
6
7
8   sys.settrace(tracefunc)
9   greet_neighbors()
```

```
1 call
2 line
```

Tracing Function

```
1   def greet_neighbors():
2       planets = ["Mars", "Venus"]
3       for p in planets:
4           print(f"Hi {p}!")
5       return len(planets)
6
7
8   sys.settrace(tracefunc)
9   greet_neighbors()
```

```
1 call
2 line
3 line
4 line
Hi Mars!
```

Tracing Function

```
1   def greet_neighbors():
2       planets = ["Mars", "Venus"]
3       for p in planets:
4           print(f"Hi {p}!")
5       return len(planets)
6
7
8   sys.settrace(tracefunc)
9   greet_neighbors()
```

```
1 call
2 line
3 line
4 line
Hi Mars!
3 line
4 line
Hi Venus!
```

Tracing Function

```
1   def greet_neighbors():
2       planets = ["Mars", "Venus"]
3       for p in planets:
4           print(f"Hi {p}!")
5       return len(planets)
6
7
8   sys.settrace(tracefunc)
9   greet_neighbors()
```

```
1 call
2 line
3 line
4 line
Hi Mars!
3 line
4 line
Hi Venus!
5 line
5 return
```

Breakpoint

- **`frame.f_lineno`** - current line number
- **`frame.f_code.co_filename`** - current file name

Breakpoint

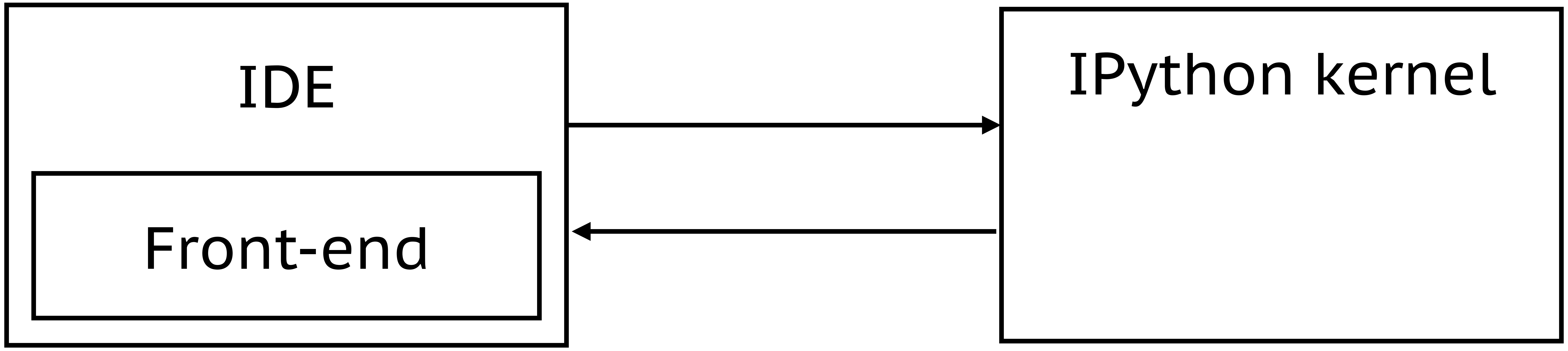
- **`frame.f_lineno`** - current line number
- **`frame.f_code.co_filename`** - current file name
- Equals to breakpoint’s file and line -> suspend program!

Contents

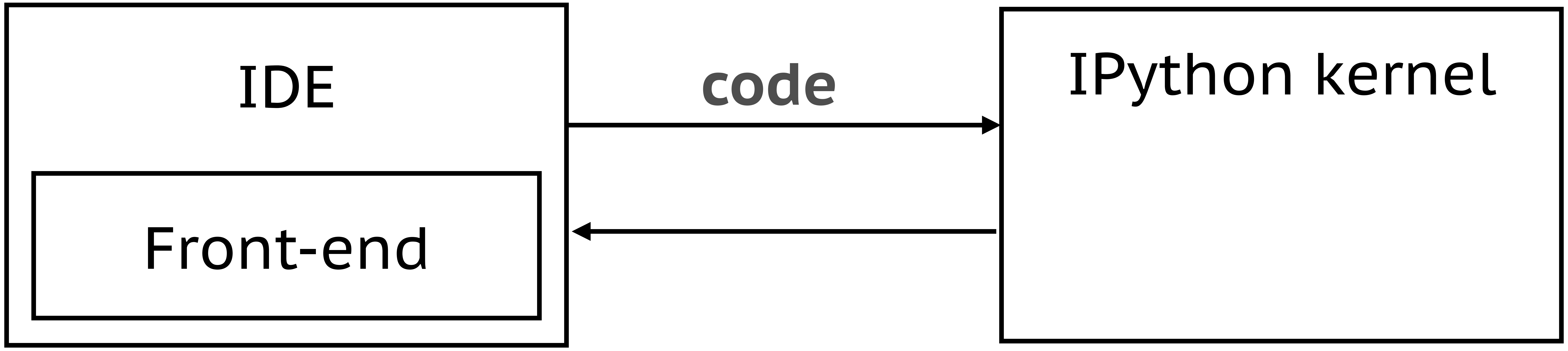
- Python files debugging
- Jupyter breakpoints
- Debugger communication
- Jupyter visual debugger

Contents

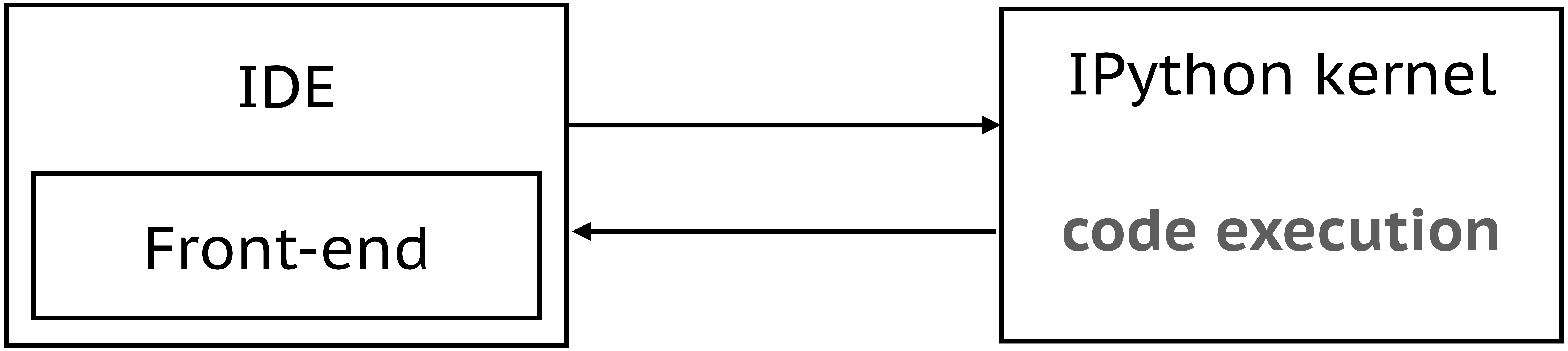
Cells Execution



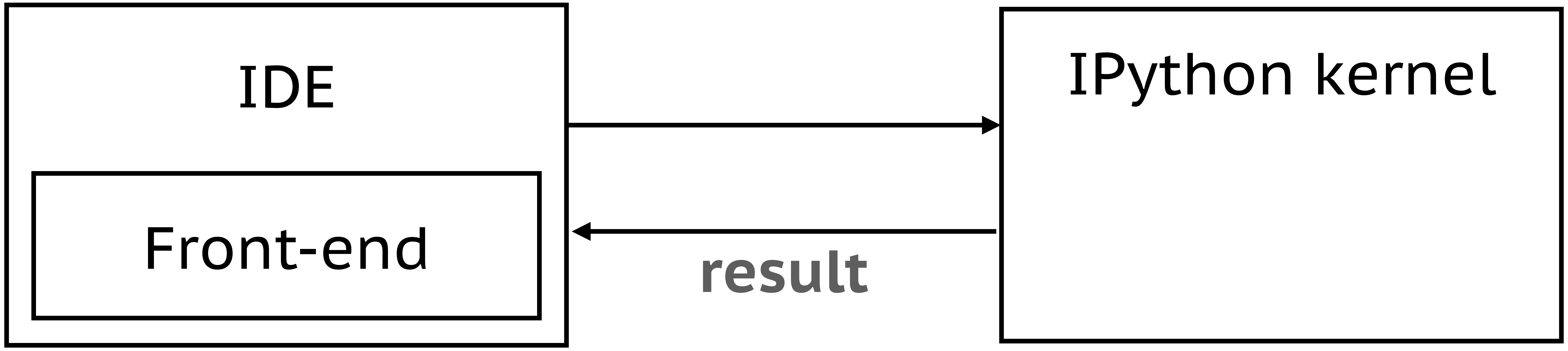
Cells Execution



Cells Execution



Cells Execution



Cells Execution

- Kernel generates a unique name for each cell
- <ipython-input-5-11faed10a894>
- File name of a generated code object

IPython kernel

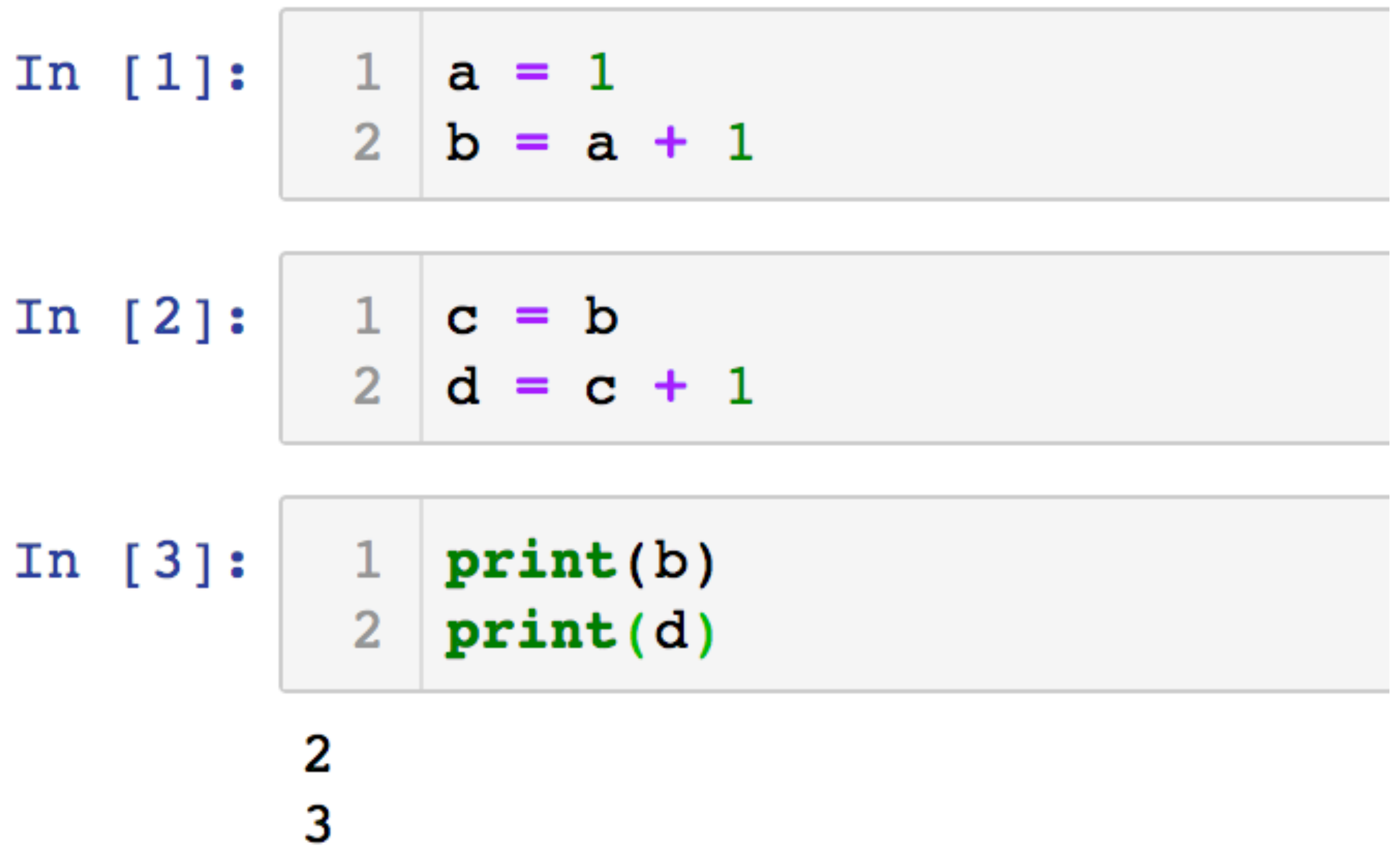
code execution

Jupyter Breakpoints

- Python files: (filename, line number) -> unique location

Jupyter Breakpoints

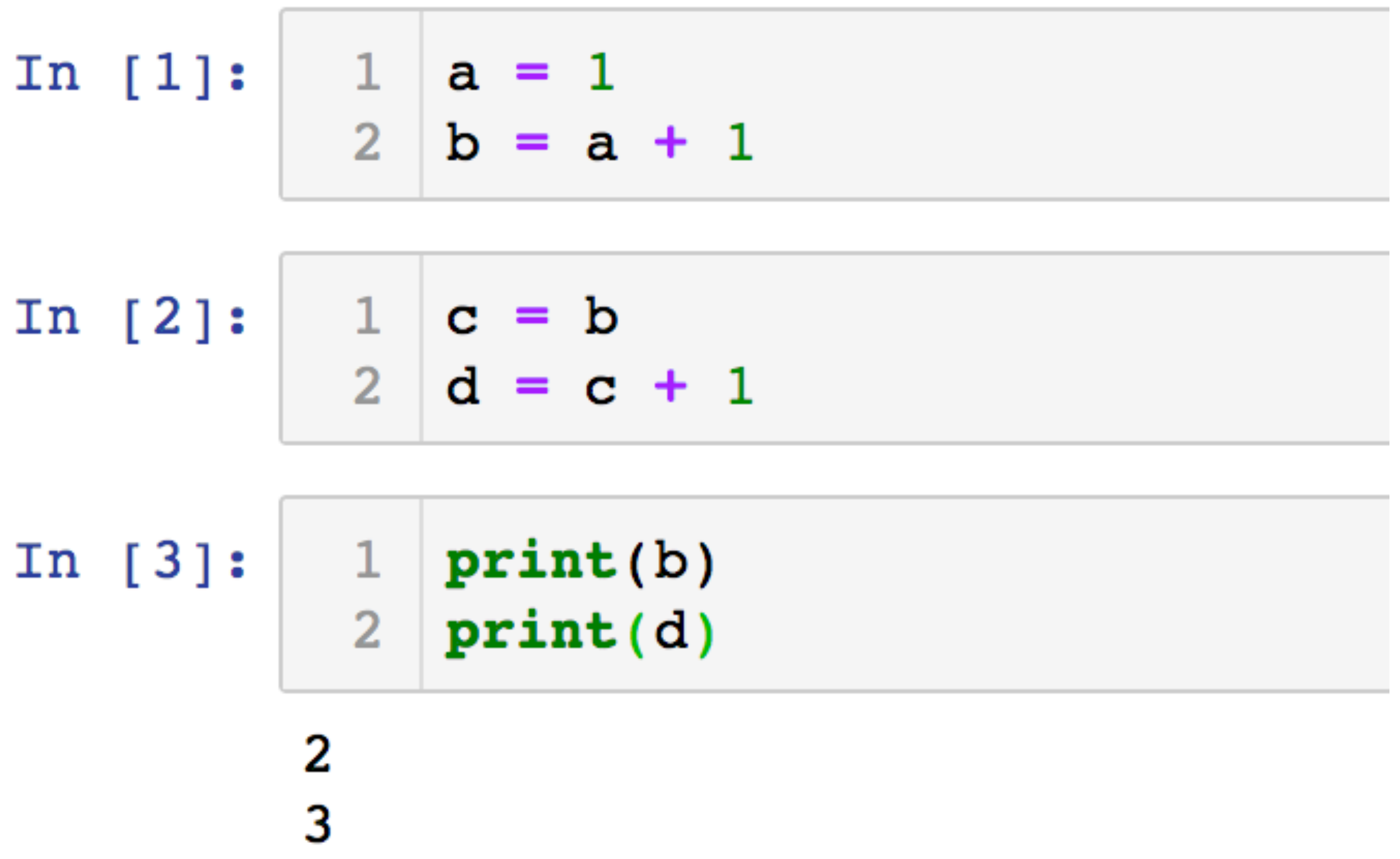
- Python files: (filename, line number) -> unique location
- Jupyter Notebooks?



```
In [1]:  a = 1
         b = a + 1

In [2]:  c = b
         d = c + 1

In [3]:  print(b)
         print(d)

2
3
```

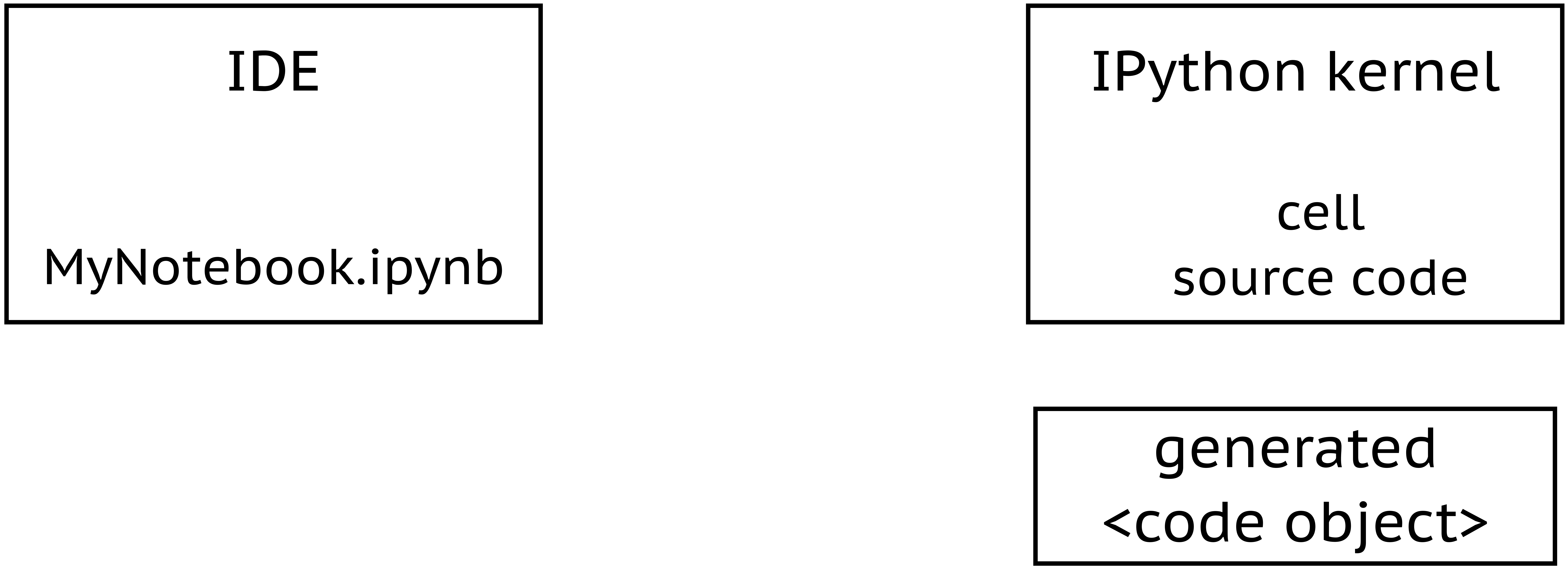
Jupyter Breakpoints

- Python files: (filename, line number) -> unique location
- Jupyter Notebooks:
  - generated cell name
  - line inside code object

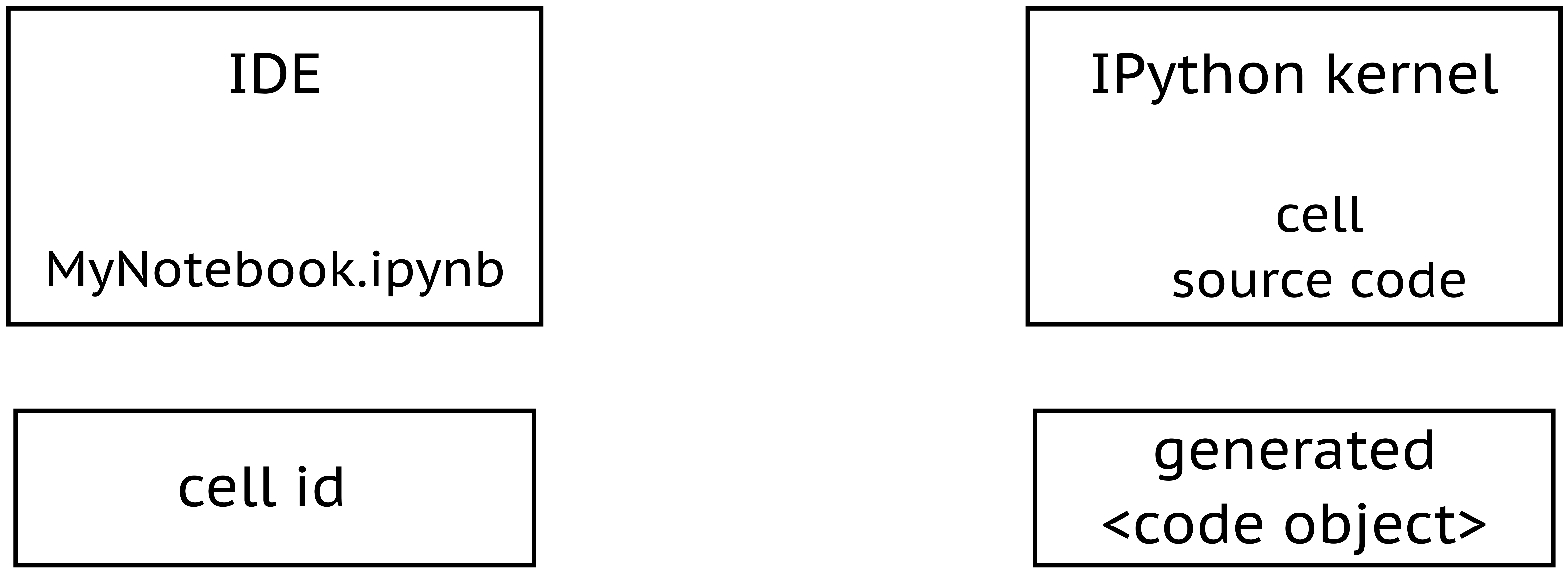


```
In [1]:  a = 1
         b = a + 1

In [2]:  c = b
         d = c + 1

In [3]:  print(b)
         print(d)

2
3
```

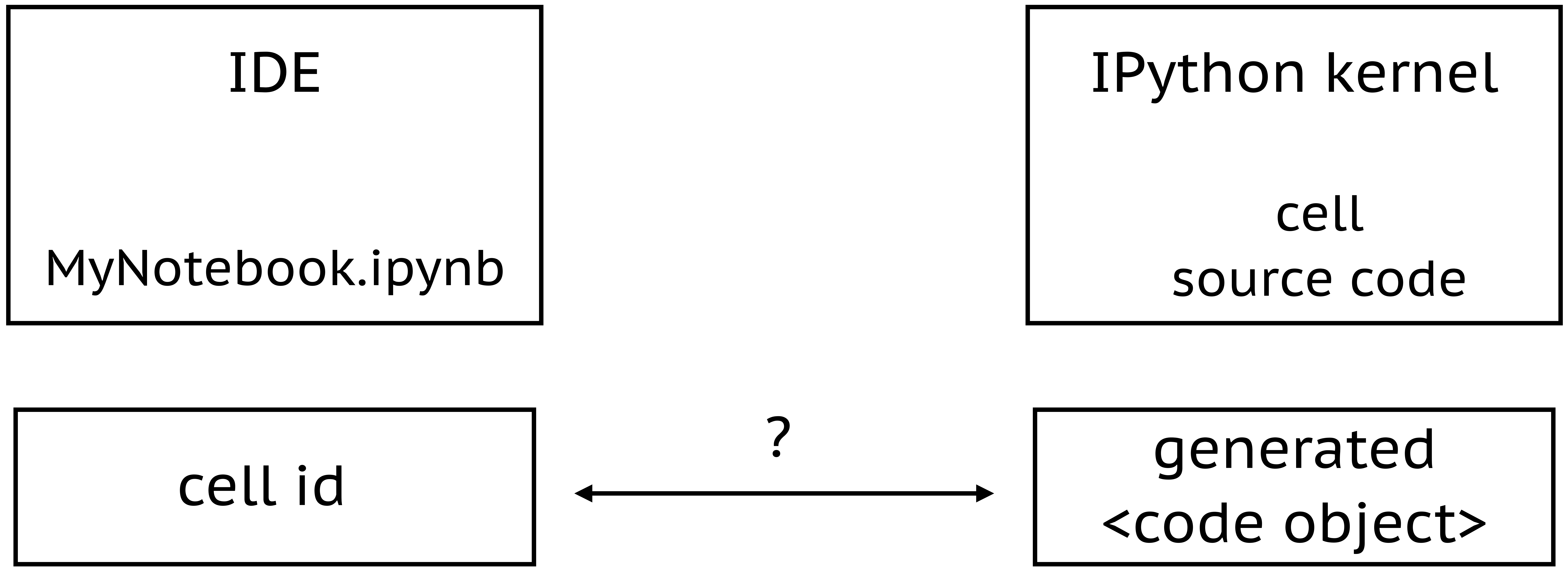
Source Mapping



Source Mapping



Source Mapping



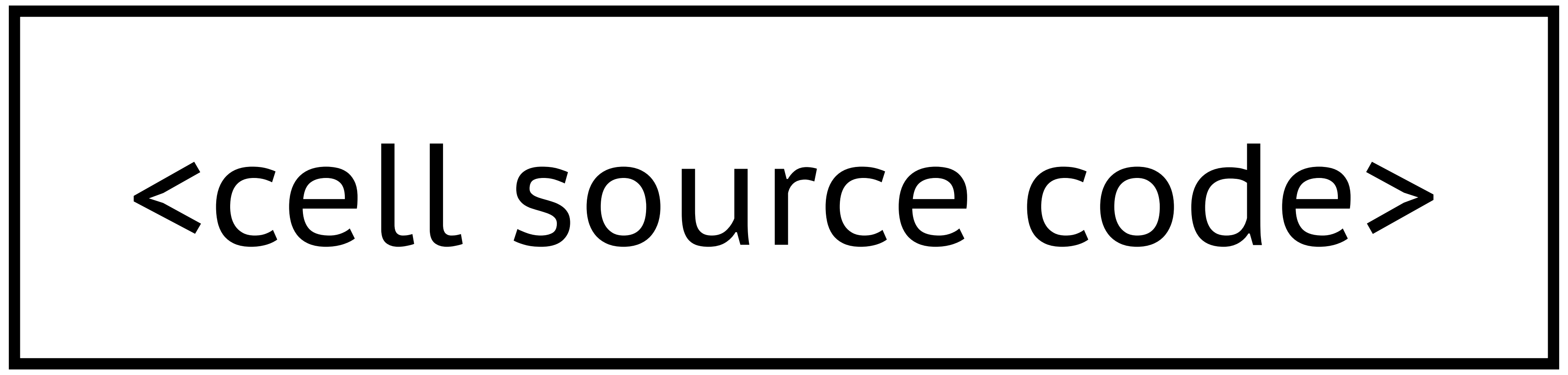
Source Mapping

- Tracking cells execution in the IDE

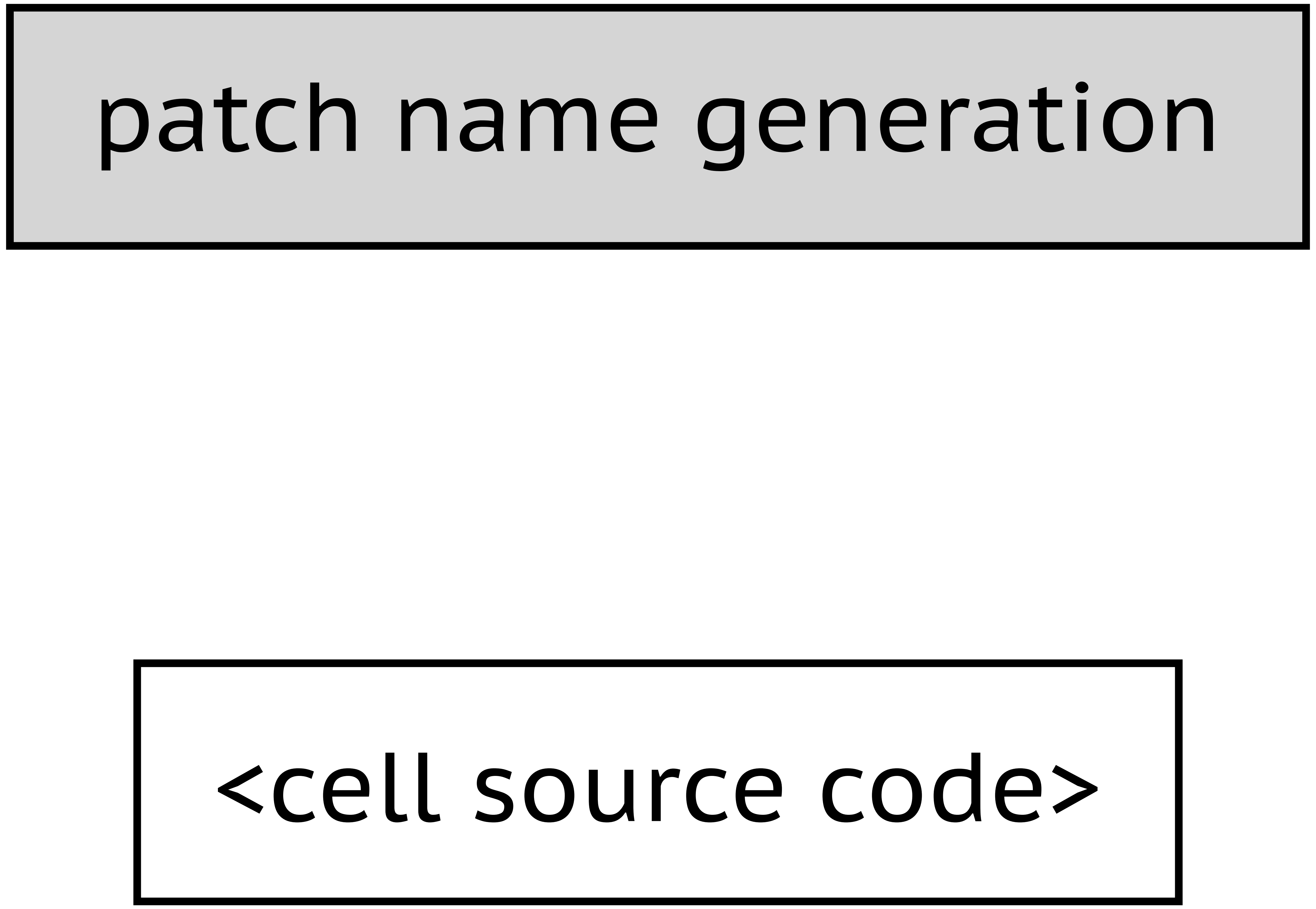
Source Mapping

- Tracking cells execution in the IDE
- Silent cell execution in IPython kernel

Debug Cell Execution

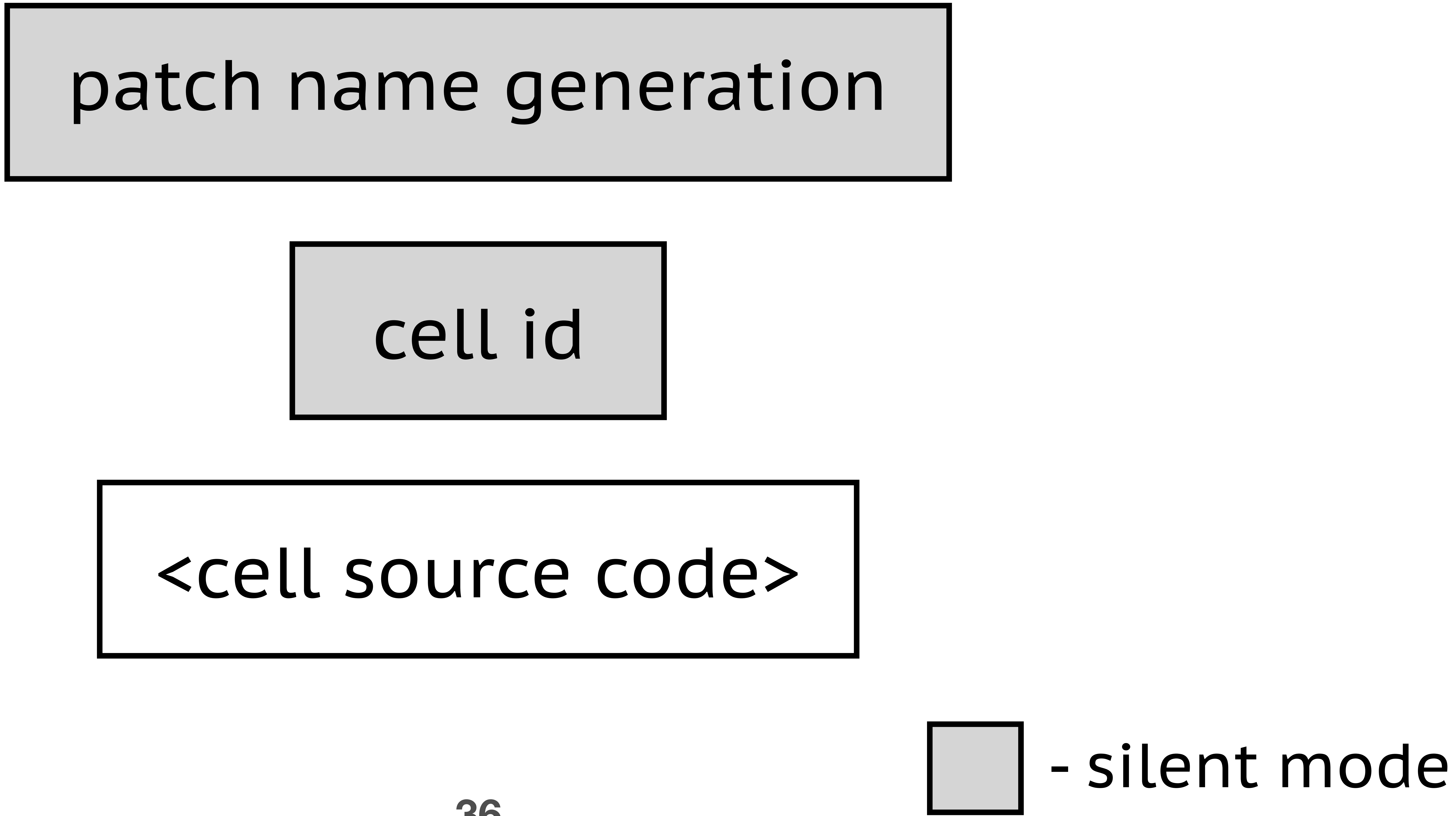


Debug Cell Execution



▨ - silent mode

Debug Cell Execution



Jupyter Tracing Function

- **`frame.f_code.co_filename`** - generated name

Jupyter Tracing Function

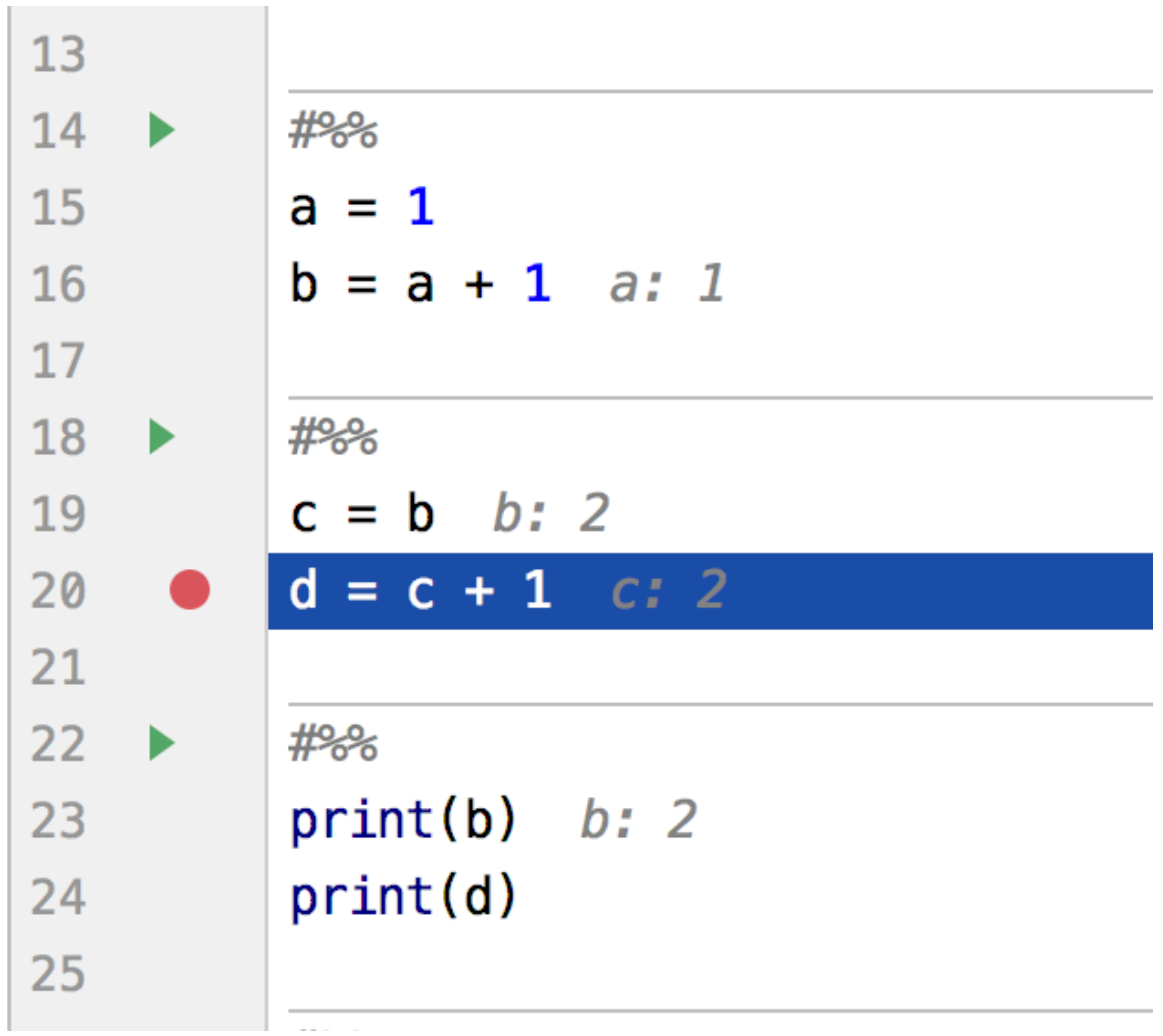
- **`frame.f_code.co_filename`** - generated name
- Map: generated name -> cell id

Jupyter Tracing Function

- **`frame.f_code.co_filename`** - generated name
- Map: generated name -> cell id
- Send message to the IDE

Jupyter Tracing Function

- **`frame.f_code.co_filename`** - generated name
- Map: generated name -> cell id
- Send message to the IDE



```
#%%
a = 1
b = a + 1  a: 1

#%%
c = b  b: 2
d = c + 1  c: 2

#%%
print(b)  b: 2
print(d)
```

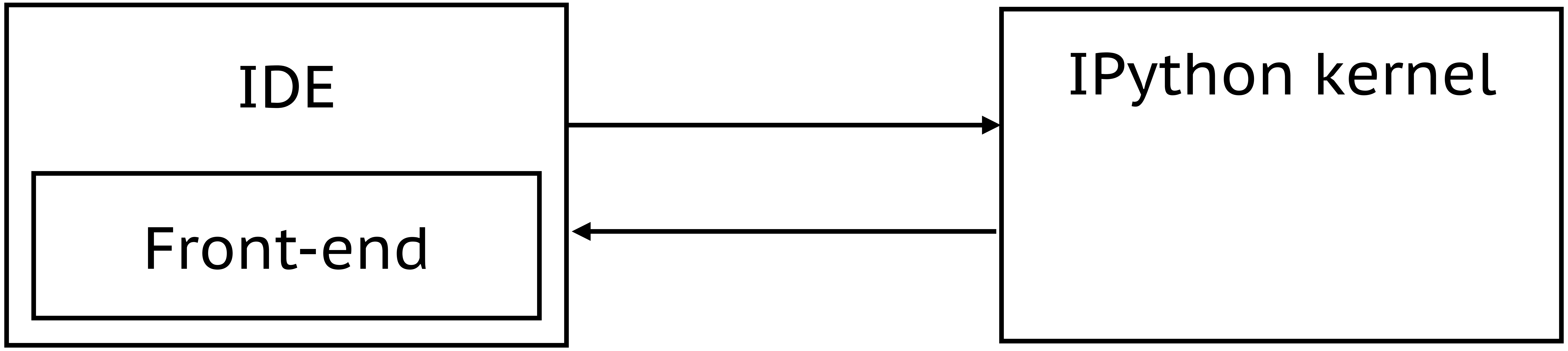
Contents

- Python files debugging
- Jupyter breakpoints
- Debugger communication
- Jupyter visual debugger

Contents

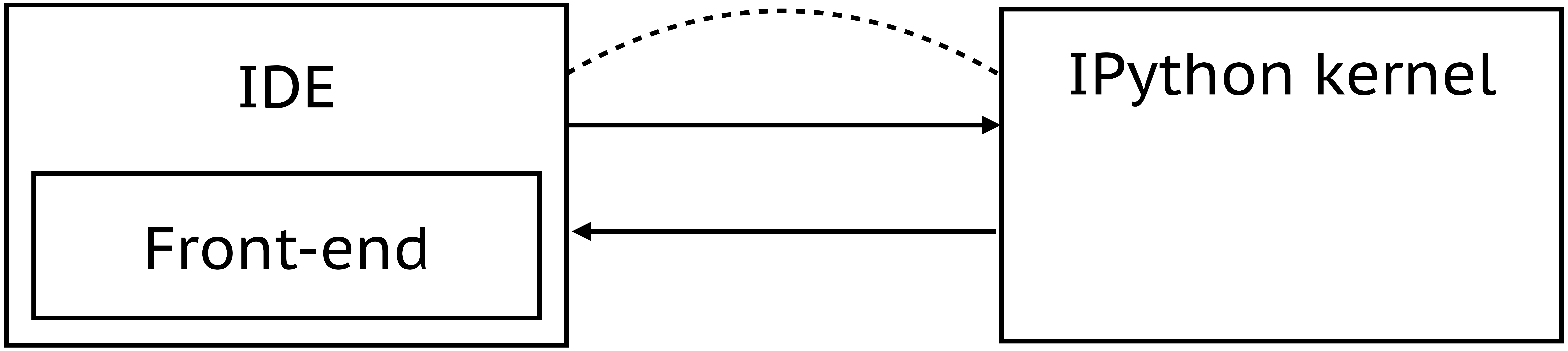
- Python files debugging
- Jupyter breakpoints
- Debugger communication
- Jupyter visual debugger

Debug Communication



“Add breakpoint in a cell 3, line 2”

Debug Communication

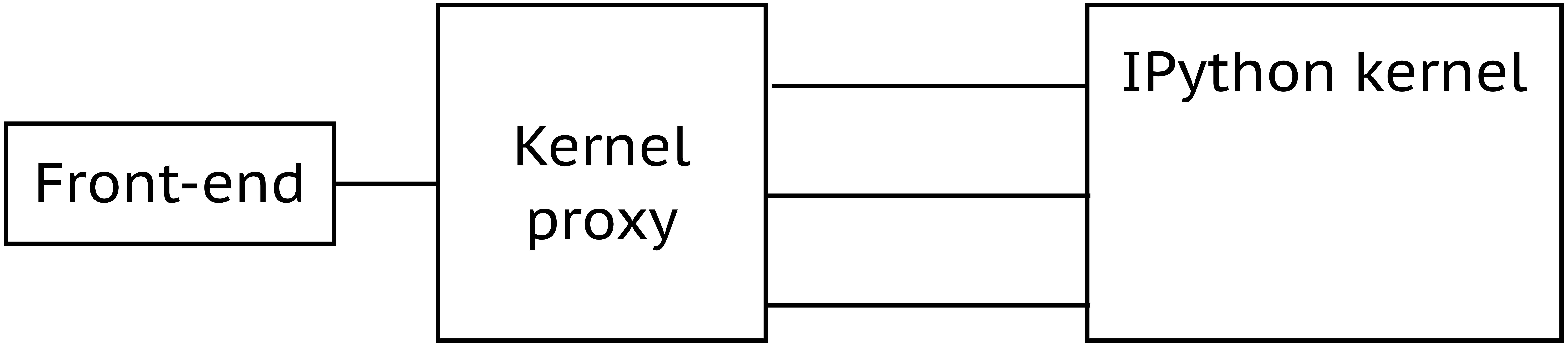


“Add breakpoint in a cell 3, line 2”

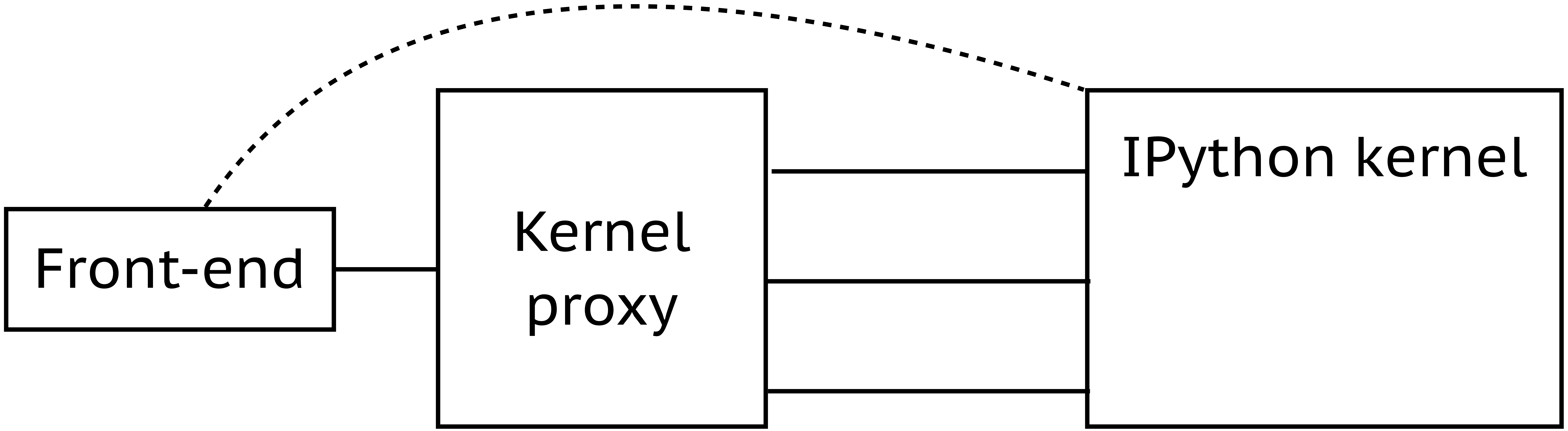
Debug Communication

- Additional connection
- Reuse Jupyter channels

Jupyter Messaging



Jupyter Messaging



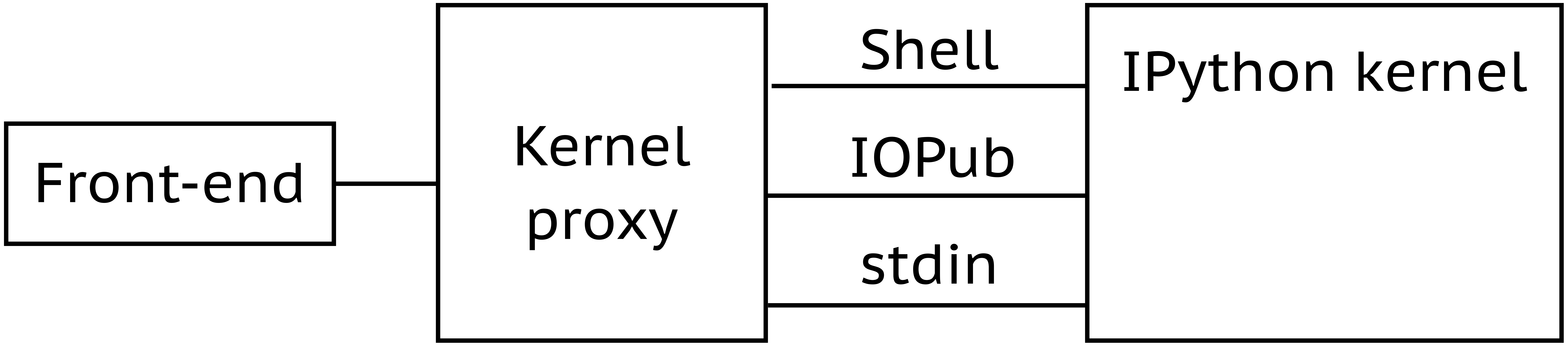
Debug Communication

- Additional connection
- Reuse Jupyter channels

Debug Communication

- Additional connection
- Reuse Jupyter channels

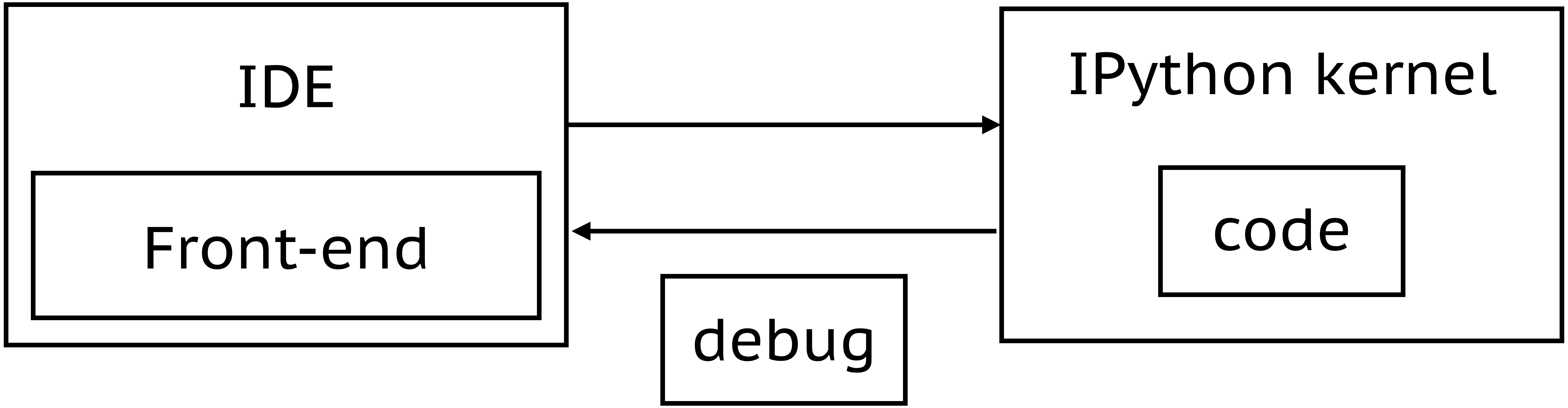
Jupyter Messaging



Jupyter Architecture

- Event loop in a main thread for execution events
- Event loop for output events

Jupyter Architecture



x Blocked

Debug Communication

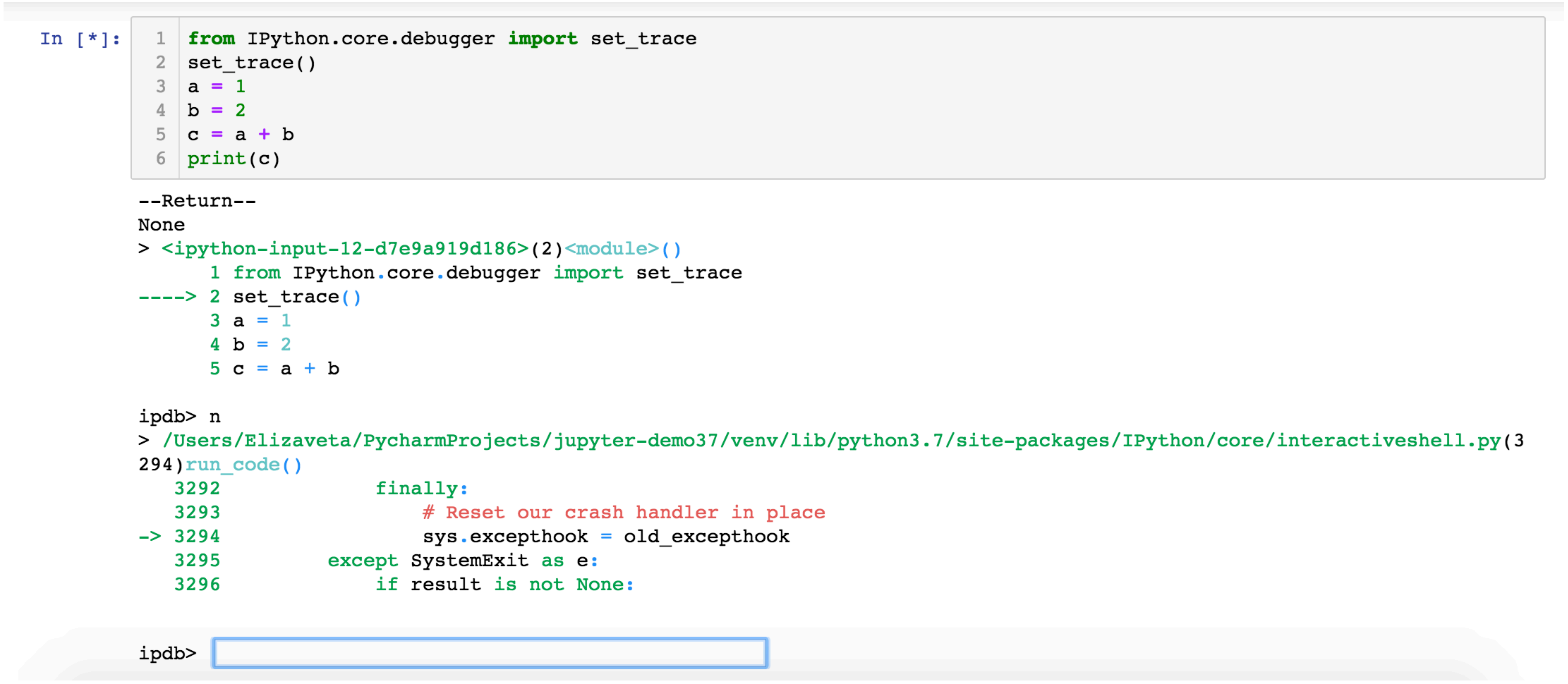
- Additional connection
- Reuse Jupyter channels

Debug Communication

- Additional connection
- Reuse Jupyter channels

But **`ipdb`** works!

But ipdb Works!



```
In [*]:
from IPython.core.debugger import set_trace
set_trace()
a = 1
b = 2
c = a + b
print(c)
```

```
--Return--
None
> <ipython-input-12-d7e9a919d186>(2)<module>()
      1 from IPython.core.debugger import set_trace
----> 2 set_trace()
      3 a = 1
      4 b = 2
      5 c = a + b

ipdb> n
> /Users/Elizaveta/PycharmProjects/jupyter-demo37/venv/lib/python3.7/site-packages/IPython/core/interactiveshell.py(3
294)run_code()
   3292             finally:
   3293                 # Reset our crash handler in place
-> 3294                 sys.excepthook = old_excepthook
   3295         except SystemExit as e:
   3296             if result is not None:

ipdb>
```

But ipdb Works!

- Based on **`input()`**
- Reuses user input channel

Debug Communication

- Additional connection
- Reuse Jupyter channels

Contents

Contents

- Python files debugging
- Jupyter breakpoints
- Debugger communication
- Jupyter visual debugger

Jupyter Visual Debugger

- Jupyter tracing function

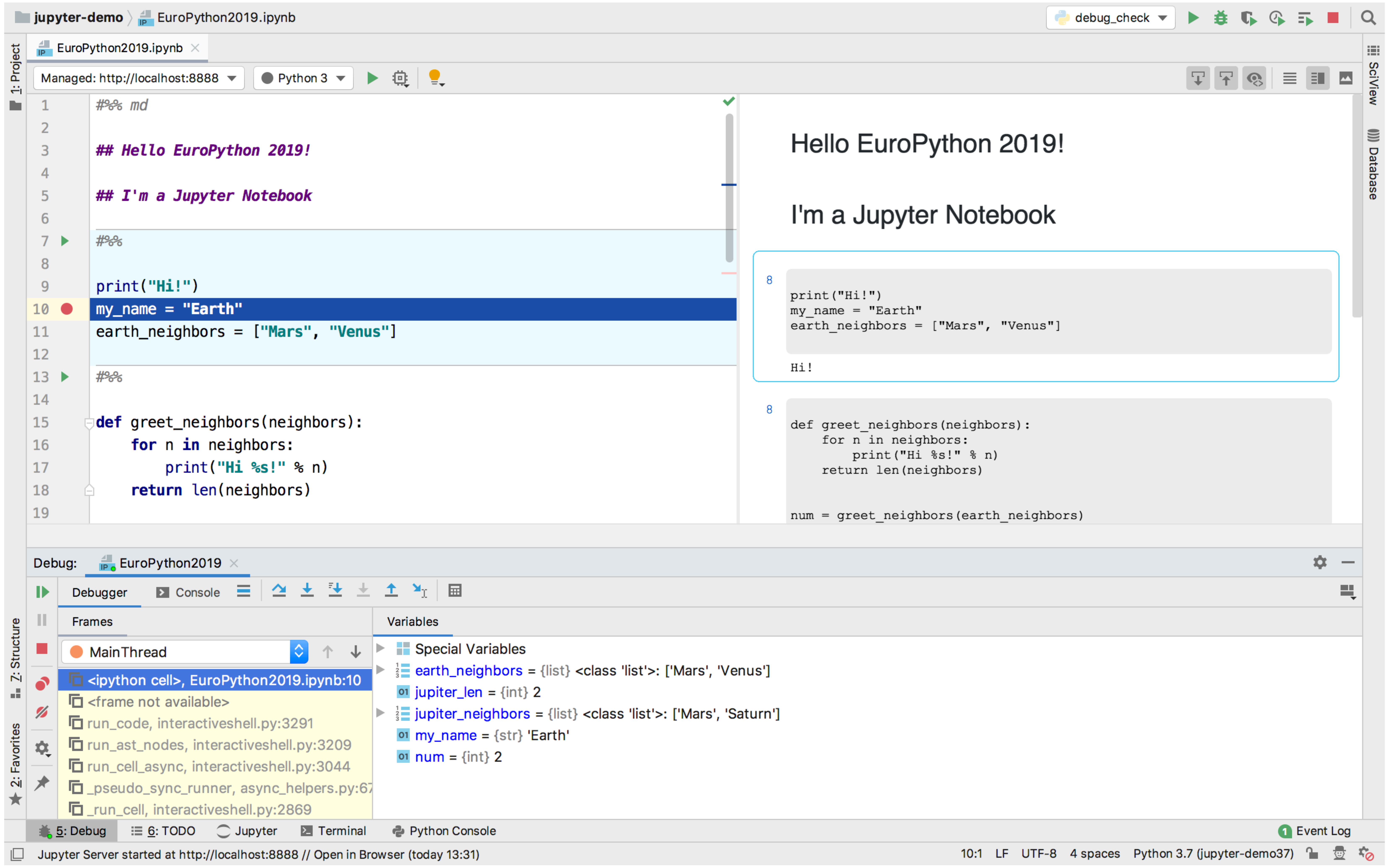
Jupyter Visual Debugger

- Jupyter tracing function
- Mapping between editor and generated code

Jupyter Visual Debugger

- Jupyter tracing function
- Mapping between editor and generated code
- Debugger connection

Live Demo



```
#%% md

## Hello EuroPython 2019!

## I'm a Jupyter Notebook

#%%

print("Hi!")
my_name = "Earth"
earth_neighbors = ["Mars", "Venus"]

#%%

def greet_neighbors(neighbors):
    for n in neighbors:
        print("Hi %s!" % n)
    return len(neighbors)

```

Hello EuroPython 2019!

I'm a Jupyter Notebook

```
8
print("Hi!")
my_name = "Earth"
earth_neighbors = ["Mars", "Venus"]

Hi!
```

```
8
def greet_neighbors(neighbors):
    for n in neighbors:
        print("Hi %s!" % n)
    return len(neighbors)


num = greet_neighbors(earth_neighbors)
```

```
<ipython cell>, EuroPython2019.ipynb:10
<frame not available>
run_code, interactiveshell.py:3291
run_ast_nodes, interactiveshell.py:3209
run_cell_async, interactiveshell.py:3044
_pseudo_sync_runner, async_helpers.py:67
_run_cell, interactiveshell.py:2869
```

```
Special Variables
earth_neighbors = {list} <class 'list'>: ['Mars', 'Venus']
jupiter_len = {int} 2
jupiter_neighbors = {list} <class 'list'>: ['Mars', 'Saturn']
my_name = {str} 'Earth'
num = {int} 2
```

Jupyter Server started at http://localhost:8888 // Open in Browser (today 13:31)   10:1 LF UTF-8 4 spaces Python 3.7 (jupyter-demo37)

Live Demo

- PyCharm doesn’t convert Jupyter Notebooks to Python files!
- On disk it’s still the same JSON file with **`.ipynb`** extension

Jupyter Visual Debugger

- Jupyter tracing function
- Mapping between editor and generated code
- Debugger connection

Jupyter Visual Debugger

- Implement in your favourite IDE

Jupyter Visual Debugger

- Implement in your favourite IDE
- Try it in PyCharm Pro!

Jupyter Visual Debugger

- Implement in your favourite IDE
- Try it in PyCharm Pro!
- Questions?

@lisa_shashkova