Debugging in Python 3.6: Better, Faster, Stronger

Elizaveta Shashkova
JetBrains

EuroPython 2017

Bio

- Software developer of PyCharm IDE at JetBrains
- Debugger
- Saint Petersburg, Russia

Debugging

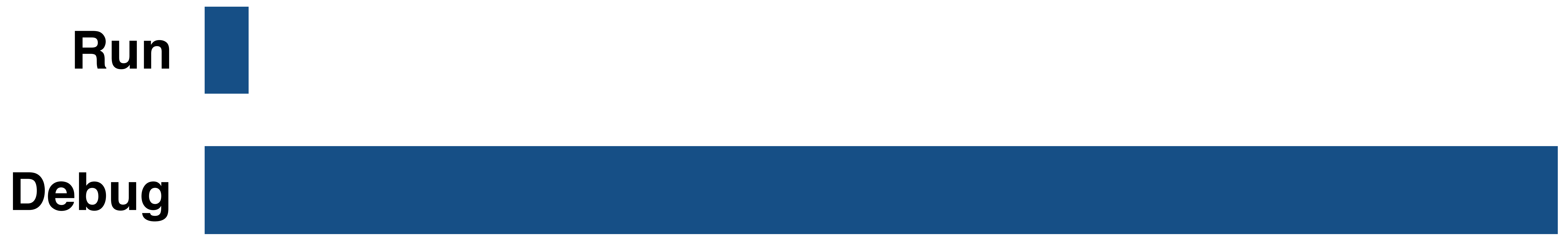
- Adding print statements
- Logging

Debugging

- Adding print statements
- Logging
- Special tools: debuggers

Debugger's Performance

Debuggers are 30 times slower



Contents

- Tracing debugger
- Python 3.6
- Frame evaluation debugger
- Results

Contents

- **Tracing debugger**
- Python 3.6
- Frame evaluation debugger
- Results

Tracing Function

sys.settrace(tracefunc) - set system tracing function

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


sys.settrace(tracefunc)
```

Tracing Function

```
def foo():
    friends = ["Bob", "Tom"]
    for f in friends:
        print("Hi %s!” % f)
    return len(friends)


sys.settrace(tracefunc)
foo()
```

Tracing Function

```
1   def foo():
2       friends = ["Bob", "Tom"]
3       for f in friends:
4           print("Hi %s!” % f)
5       return len(friends)
6
7
8   sys.settrace(tracefunc)
9   foo()
```

```
1 call
```

Tracing Function

```
1   def foo():
2       friends = ["Bob", “Tom"]
3       for f in friends:
4           print("Hi %s!” % f)
5       return len(friends)
6
7
8   sys.settrace(tracefunc)
9   foo()
```

```
1 call
2 line
```

Tracing Function

```
1   def foo():
2       friends = ["Bob", "Tom"]
3       for f in friends:
4           print("Hi %s!” % f)
5       return len(friends)
6
7
8   sys.settrace(tracefunc)
9   foo()
```

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

Tracing Function

```
1   def foo():
2       friends = ["Bob", "Tom"]
3       for f in friends:
4           print("Hi %s!” % f)
5       return len(friends)
6
7
8   sys.settrace(tracefunc)
9   foo()
```

```
1 call
2 line
3 line
4 line
Hi Bob!
3 line
4 line
Hi Tom!
```

Tracing Function

```
1   def foo():
2       friends = ["Bob", "Tom"]
3       for f in friends:
4           print("Hi %s!” % f)
5       return len(friends)
6
7
8   sys.settrace(tracefunc)
9   foo()
```

```
1 call
2 line
3 line
4 line
Hi Bob!
3 line
4 line
Hi Tom!
5 line
5 return
```

Build Python Debugger

- Breakpoints
- Stepping

Tracing Debugger

- Suspend program if breakpoint's line equals `frame.f_lineno`
- Handle events for stepping

Performance

```
1   def foo():
2       friends = ["Bob", "Tom"]
3       for f in friends:
4           print("Hi %s!” % f)
5       return len(friends)
6
7
8   sys.settrace(tracefunc)
9   foo()
```

```
1 call
2 line
3 line
4 line
Hi Bob!
3 line
4 line
Hi Tom!
5 line
5 return
```

Example 1

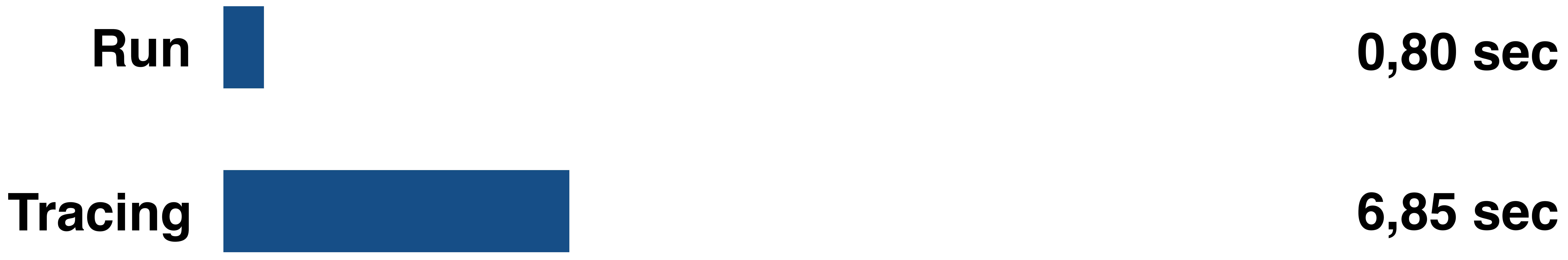
```python
def calculate():
    sum = 0
    for i in range(10 ** 7):
        sum += i
    return sum
```

Example 1

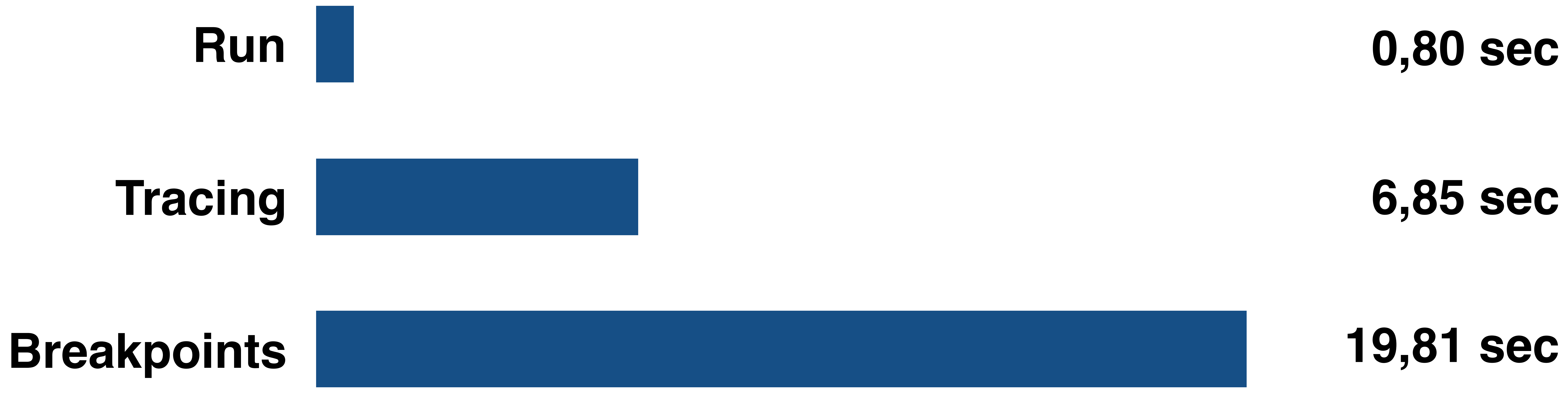
```python
def calculate():
    sum = 0
    for i in range(10 ** 7):
        sum += i
    return sum


def tracefunc(frame, event, arg):
    return tracefunc
```

Performance

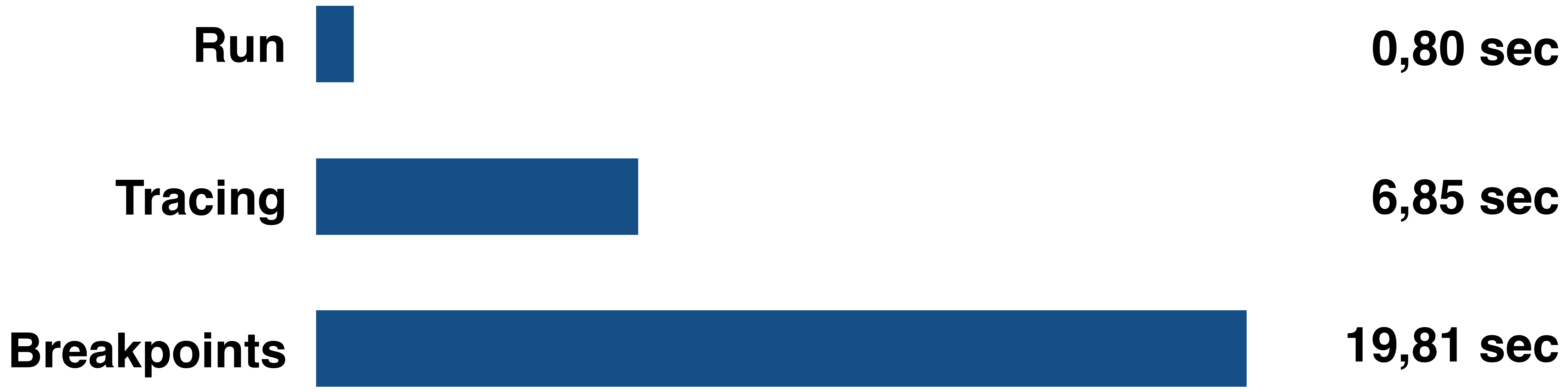
Performance



Performance



Performance



~ 25 times slower!

Problem

- **Tracing call on every line**

Contents

- **Tracing debugger**
- Python 3.6
- Frame evaluation debugger
- Results

Contents

- Tracing debugger
- **Python 3.6**
- Frame evaluation debugger
- Results

Python 3.6

Python 3.6

- New frame evaluation API
- PEP 523

PEP 523

- Handle evaluation of frames
- Add a new field to code objects

Frame Evaluation

```python
def frame_eval(frame, exc):
    func_name = frame.f_code.co_name
    line_number = frame.f_lineno
    print(line_number, func_name)
    return _PyEval_EvalFrameDefault(frame, exc)




```

Frame Evaluation

```python
def frame_eval(frame, exc):
    func_name = frame.f_code.co_name
    line_number = frame.f_lineno
    print(line_number, func_name)
    return _PyEval_EvalFrameDefault(frame, exc)
```

Frame Evaluation

```python
def frame_eval(frame, exc):
    func_name = frame.f_code.co_name
    line_number = frame.f_lineno
    print(line_number, func_name)
    return _PyEval_EvalFrameDefault(frame, exc)
```

Frame Evaluation

```python
def frame_eval(frame, exc):
    func_name = frame.f_code.co_name
    line_number = frame.f_lineno
    print(line_number, func_name)
    return _PyEval_EvalFrameDefault(frame, exc)
```

Frame Evaluation

```python
def frame_eval(frame, exc):
    func_name = frame.f_code.co_name
    line_number = frame.f_lineno
    print(line_number, func_name)
    return _PyEval_EvalFrameDefault(frame, exc)
```

Frame Evaluation

```
def frame_eval(frame, exc):
    func_name = frame.f_code.co_name
    line_number = frame.f_lineno
    print(line_number, func_name)
    return _PyEval_EvalFrameDefault(frame, exc)

def set_frame_eval():
    state = PyThreadState_Get()
    state.interp.eval_frame = frame_eval
```

Example

```python
def first():
    second()

def second():
    third()

def third():
    pass

set_frame_eval()
first()
```

Example

```python
1   def first():
2       second()
3
4   def second():
5       third()
6
7   def third():
8       pass
9
10  set_frame_eval()
11  first()
```

```
1 first
4 second
7 third
```

Custom Frame Evaluation

- It works!
- Executed while entering a frame
- Access to frame and code object

Contents

- Tracing debugger
- Python 3.6
- Frame evaluation debugger
- Results

Problem

- Tracing call on every line

Problem

- Tracing call on every line
- Remove the tracing function!

Replace tracing function with frame evaluation function

Contents

- Tracing debugger
- Python 3.6
- **Frame evaluation debugger**
- Results

Build Python Debugger

- Breakpoints
- Stepping

Breakpoints

- **Access to the whole code object**

Breakpoints

- Access to the whole code object
- Insert breakpoint’s code into frame’s code

Breakpoints

```python
def maximum(a, b):
    if a > b:
        return a
    else:
        return b
```

Breakpoints

```python
def maximum(a, b):
    if a > b:
        return a  # breakpoint
    else:
        return b
```

Breakpoints

```python
def maximum(a, b):
    if a > b:
        return a  # breakpoint
    else:
        return b
```

```
breakpoint()
```

Breakpoints

```python
def maximum(a, b):
    if a > b:
        breakpoint()
        return a  # breakpoint
    else:
        return b
```

Python Bytecode

```python
def maximum(a, b):
    if a > b:
        return a
    else:
        return b

import dis
dis.dis(maximum)
```

Python Bytecode

```python
def maximum(a, b):
    if a > b:
        return a
    else:
        return b

import dis
dis.dis(maximum)
```

```
2        0 LOAD_FAST           0 (a)
         2 LOAD_FAST           1 (b)
         4 COMPARE_OP          4 (>)
         6 POP_JUMP_IF_FALSE 12

3        8 LOAD_FAST           0 (a)
        10 RETURN_VALUE

5 >>    12 LOAD_FAST           1 (b)
        14 RETURN_VALUE
        16 LOAD_CONST          0 (None)
        18 RETURN_VALUE
```

Python Bytecode

```python
def maximum(a, b):
    if a > b:
        return a
    else:
        return b

import dis
dis.dis(maximum)
```

```
2         0 LOAD_FAST           0 (a)
          2 LOAD_FAST           1 (b)
          4 COMPARE_OP          4 (>)
          6 POP_JUMP_IF_FALSE  12

3         8 LOAD_FAST           0 (a)
         10 RETURN_VALUE

5    >>  12 LOAD_FAST           1 (b)
         14 RETURN_VALUE
         16 LOAD_CONST          0 (None)
         18 RETURN_VALUE
```

Python Bytecode

```python
def maximum(a, b):
    if a > b:
        return a
    else:
        return b

import dis
dis.dis(maximum)

```

```
2           0 LOAD_FAST                0 (a)
            2 LOAD_FAST                1 (b)
            4 COMPARE_OP               4 (>)
            6 POP_JUMP_IF_FALSE       12

3           8 LOAD_FAST                0 (a)
           10 RETURN_VALUE

5     >>   12 LOAD_FAST                1 (b)
           14 RETURN_VALUE
           16 LOAD_CONST               0 (None)
           18 RETURN_VALUE
```

Python Bytecode

```python
def maximum(a, b):
    if a > b:
        return a
    else:
        return b

import dis
dis.dis(maximum)
```

```
2        0 LOAD_FAST            0 (a)
         2 LOAD_FAST            1 (b)
         4 COMPARE_OP           4 (>)
         6 POP_JUMP_IF_FALSE 12

3        8 LOAD_FAST            0 (a)
        10 RETURN_VALUE

5 >>    12 LOAD_FAST            1 (b)
        14 RETURN_VALUE
        16 LOAD_CONST           0 (None)
        18 RETURN_VALUE
```

Python Bytecode

```python
def maximum(a, b):
    if a > b:
        return a
    else:
        return b

import dis
dis.dis(maximum)
```

```
2        0 LOAD_FAST            0 (a)
         2 LOAD_FAST            1 (b)
         4 COMPARE_OP           4 (>)
         6 POP_JUMP_IF_FALSE   12

3        8 LOAD_FAST            0 (a)
        10 RETURN_VALUE

5   >>  12 LOAD_FAST            1 (b)
        14 RETURN_VALUE
        16 LOAD_CONST           0 (None)
        18 RETURN_VALUE
```

Python Bytecode

```
2      0 LOAD_FAST             0 (a)
       2 LOAD_FAST             1 (b)
       4 COMPARE_OP            4 (>)
       6 POP_JUMP_IF_FALSE    12

3      8 LOAD_FAST             0 (a)
      10 RETURN_VALUE

5 >>  12 LOAD_FAST             1 (b)
      14 RETURN_VALUE
      16 LOAD_CONST            0 (None)
      18 RETURN_VALUE
```

Python Bytecode

```
  2         0 LOAD_FAST                0 (a)
            2 LOAD_FAST                1 (b)
            4 COMPARE_OP               4 (>)
            6 POP_JUMP_IF_FALSE       12

  3         8 LOAD_FAST                0 (a)
           10 RETURN_VALUE

  5   >>   12 LOAD_FAST                1 (b)
           14 RETURN_VALUE
           16 LOAD_CONST               0 (None)
           18 RETURN_VALUE
```

Python Bytecode

```
2       0 LOAD_FAST                0 (a)
        2 LOAD_FAST                1 (b)
        4 COMPARE_OP               4 (>)
        6 POP_JUMP_IF_FALSE       12

3       8 LOAD_FAST                0 (a)
       10 RETURN_VALUE

5 >>   12 LOAD_FAST                1 (b)
       14 RETURN_VALUE
       16 LOAD_CONST               0 (None)
       18 RETURN_VALUE
```

Python Bytecode

```
2       0 LOAD_FAST            0 (a)
        2 LOAD_FAST            1 (b)
        4 COMPARE_OP           4 (>)
        6 POP_JUMP_IF_FALSE   12

3       8 LOAD_FAST            0 (a)
       10 RETURN_VALUE

5 >>   12 LOAD_FAST            1 (b)
       14 RETURN_VALUE
       16 LOAD_CONST           0 (None)
       18 RETURN_VALUE
```

Bytecode Modification

```
2       0 LOAD_FAST           0 (a)
        2 LOAD_FAST           1 (b)
        4 COMPARE_OP          4 (>)
        6 POP_JUMP_IF_FALSE  12

3       8 LOAD_FAST           0 (a)
       10 RETURN_VALUE

5 >>   12 LOAD_FAST           1 (b)
       14 RETURN_VALUE
       16 LOAD_CONST          0 (None)
       18 RETURN_VALUE
```

```
breakpoint()
```

Bytecode Modification

- Insert breakpoint’s code
- Update arguments and offsets

Bytecode Modification

- Insert breakpoint’s code
- Update arguments and offsets
- 200 lines in Python

Bytecode Modification

```
2       0 LOAD_FAST           0 (a)
        2 LOAD_FAST           1 (b)
        4 COMPARE_OP          4 (>)
        6 POP_JUMP_IF_FALSE  12

3       8 LOAD_FAST           0 (a)
       10 RETURN_VALUE

5 >>   12 LOAD_FAST           1 (b)
       14 RETURN_VALUE
       16 LOAD_CONST          0 (None)
       18 RETURN_VALUE
```

`breakpoint()`

?!

Breakpoint Bytecode

```python
def _stop_at_break():
    # a lot of code here

def breakpoint():
    _stop_at_break()
```

```
0 LOAD_GLOBAL   0
2 CALL_FUNCTION 0
4 POP_TOP
6 LOAD_CONST    0
8 RETURN_VALUE
```

Build Python Debugger

- Breakpoints
- Stepping

Stepping

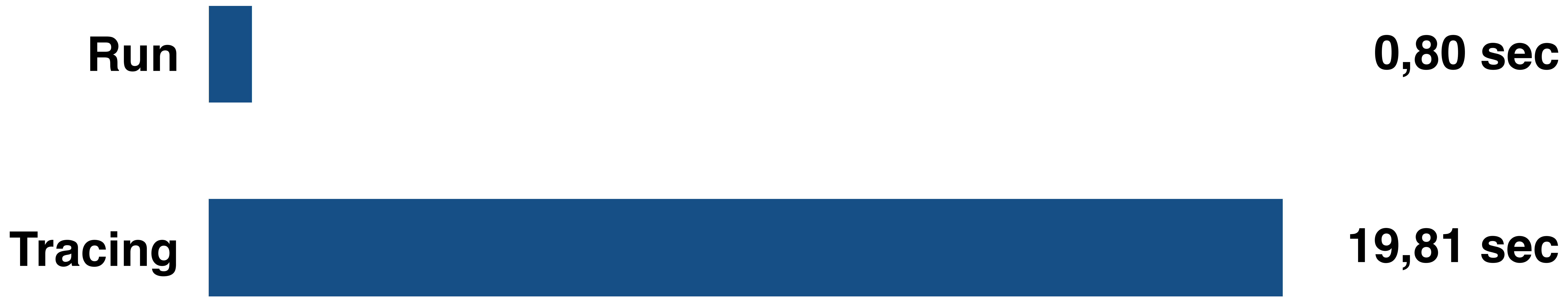
- Inserting temporary breakpoint on every line
- Use old tracing function

Frame evaluation debugger is ready!

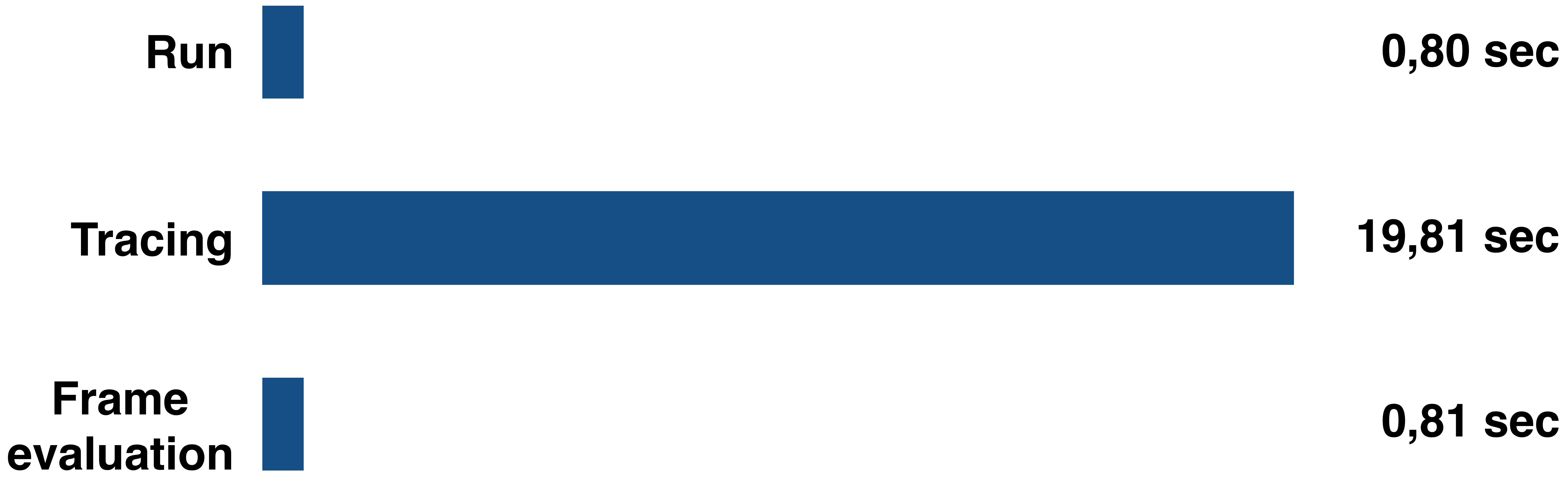
Example 1

```python
def calculate():
    sum = 0
    for i in range(10 ** 7):
        sum += i
    return sum
```

Example 1



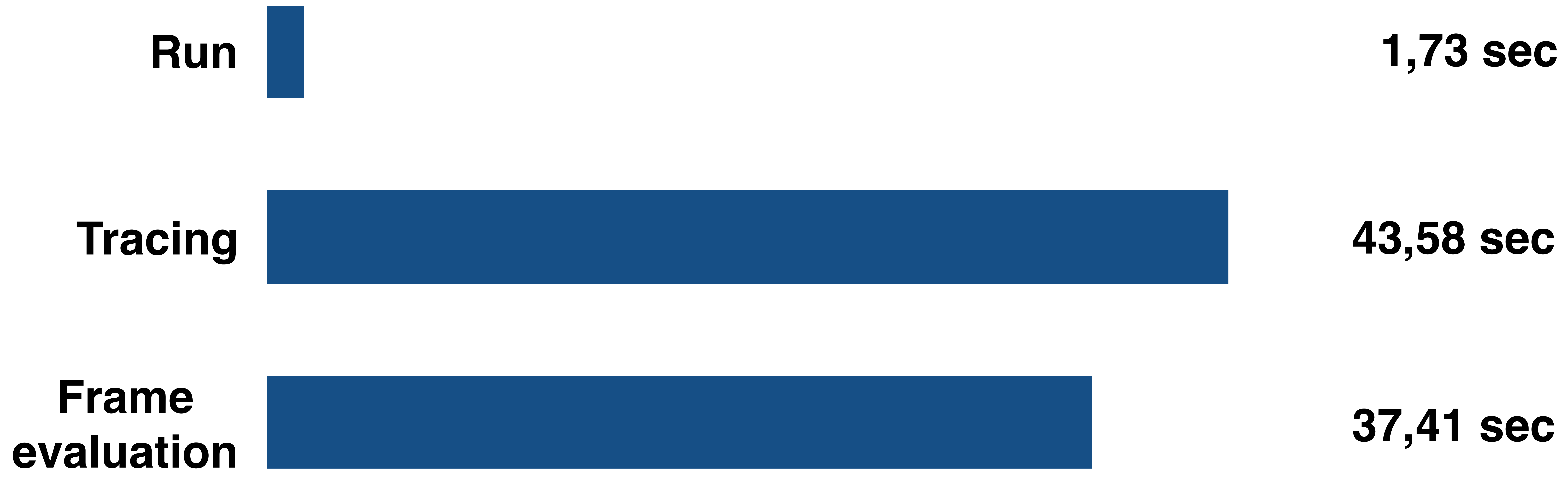
Example 1



Example 2

```
def foo():
    pass

def calculate():
    sum = 0
    for i in range(10 ** 7):
        foo()
        sum += i
    return sum
```

Example 2



PEP 523

- Handle evaluation of frames
- Add a new field to code objects

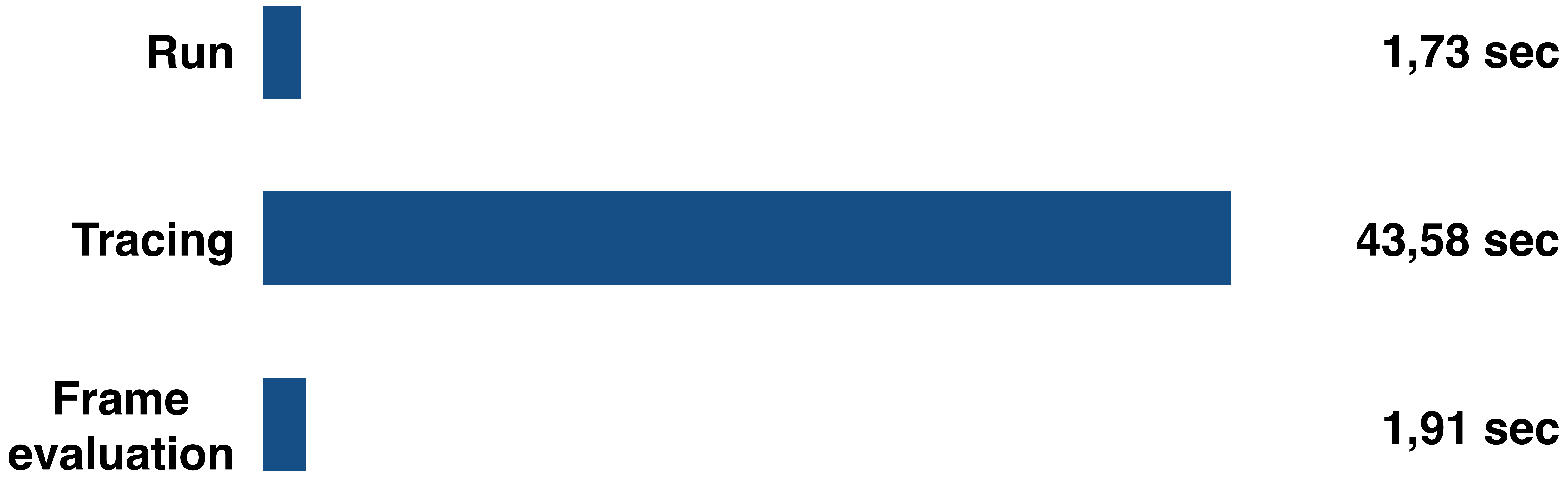
PEP 523

- Expand `PyCodeObject` struct
- `co_extra` - “scratch space” for the code object
- Mark frames without breakpoints

Mark Frames

```python
def frame_eval(frame, exc):
  flag = _PyCode_GetExtra(frame.f_code, index)
  if flag == NO_BREAKS_IN_FRAME:
    return _PyEval_EvalFrameDefault(frame, exc)

  # check for breakpoints
    ...


```

Example 2



PEP 523

- Handle evaluation of frames
- Add a new field to code objects

Contents

- Tracing debugger
- Python 3.6
- **Frame evaluation debugger**
- Results

Contents

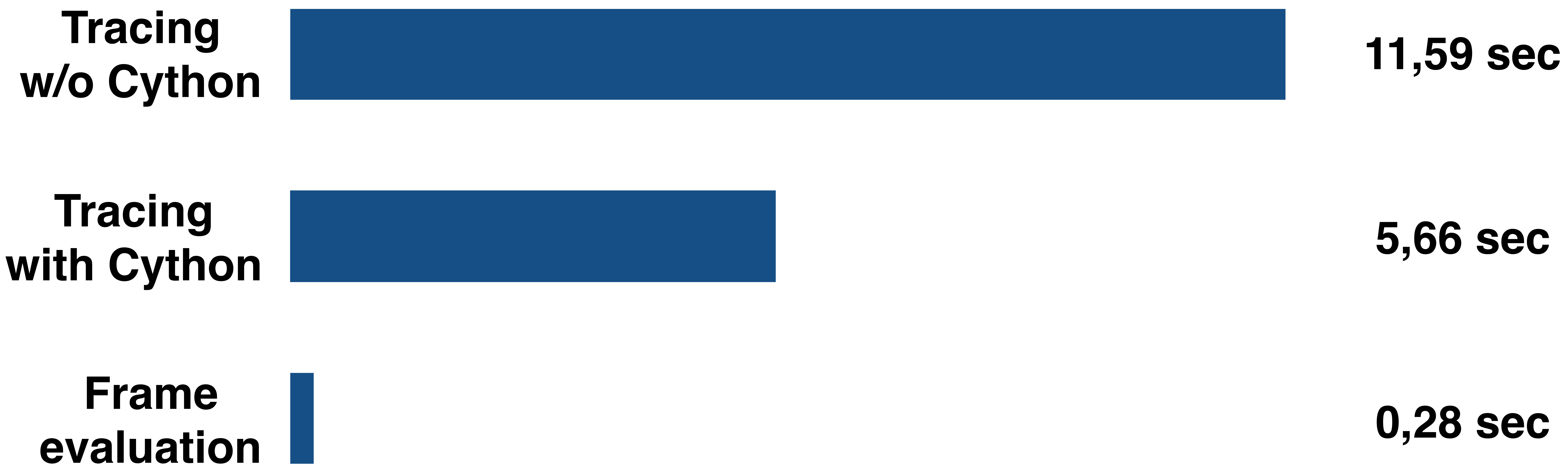
- Tracing debugger
- Python 3.6
- Frame evaluation debugger
- **Results**

Real Life Example

Real Life Example

- Included into PyCharm 2017.1
- Works in production

PyCharm


Tracing w/o Cython
11,59 sec
Tracing with Cython
5,66 sec
Frame evaluation
0,28 sec

Frame evaluation rocks!

Disadvantages

- More complicated
- Only with CPython
- Only with Python 3.6

Frame Evaluation

- **Let’s move to Python 3.6!**

Frame Evaluation

- Let’s move to Python 3.6!
- Let’s find another use cases!

Use cases

```python
def maximum(a, b):
    if a > b:
        return a  # breakpoint
    else:
        return b
```

```python
breakpoint()
```

PEP 523

- Pyjion project
- JIT for Python

Frame Evaluation

- Let’s move to Python 3.6!
- Let’s find another use cases!

Links

- Prototype: https://github.com/Elizaveta239/frame-eval
- PyCharm Community Edition source code
- `bytesinsert` on PyPi

Questions?

- Prototype: https://github.com/Elizaveta239/frame-eval
- PyCharm Community Edition source code
- `bytesinsert` on PyPi

@lisa_shashkova