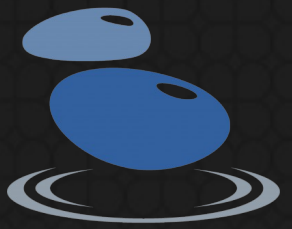
Discover asyncio event loop

Pycon 2014, Lyon

Victor Stinner
victor.stinner@gmail.com

Victor Stinner

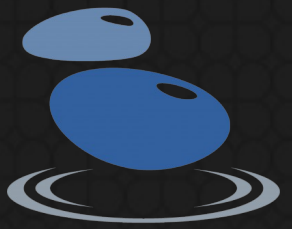


- Python core developer since 2010
- `github.com/haypo/`
- `bitbucket.org/haypo/`
- Working for eNovance

eNovance
Multi-Cloud Experts

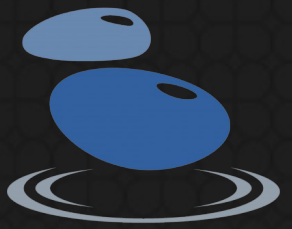
python™

Disclamer



- How asyncio is implemented
- Simplified code snippets close to asyncio, but different
- No error handling nor optimization
- (asyncio handles errors and is optimized)

Agenda

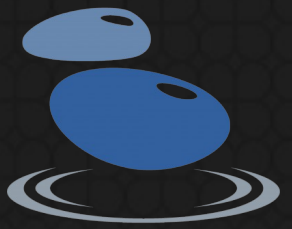


1. Callbacks
2. Timers
3. Selectors
4. Generator
5. Coroutine and BaseTask
6. Future and Task

eNovance
Multi-Cloud Experts

python™

Callbacks

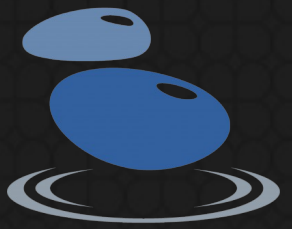


```
class CallbackEventLoop:
    def __init__(self):
        self.callbacks = []

    def call_soon(self, func):
        self.callbacks.append(func)

    def execute_callbacks(self):
        callbacks = self.callbacks
        self.callbacks = []
        for cb in callbacks:
            cb()
```

eNovance Multi-Cloud Experts

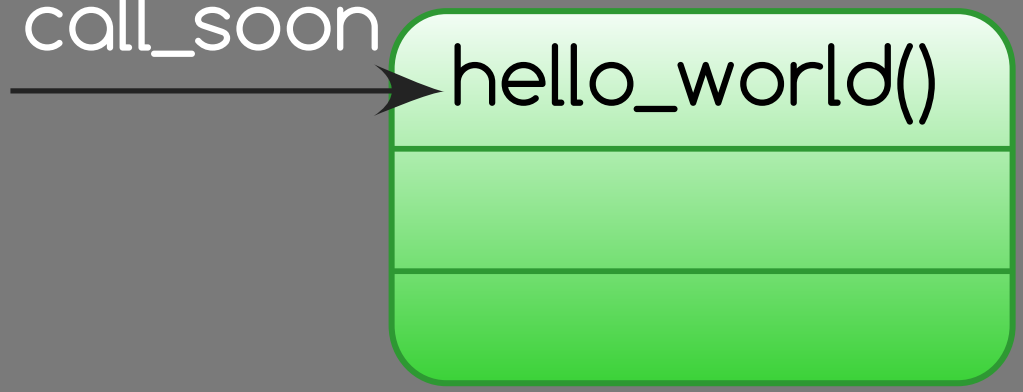
python™

Callbacks



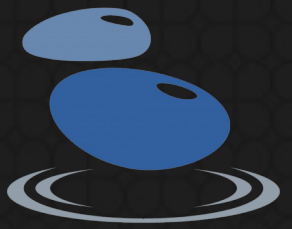
Code

```
.call_soon(hello_world)
```



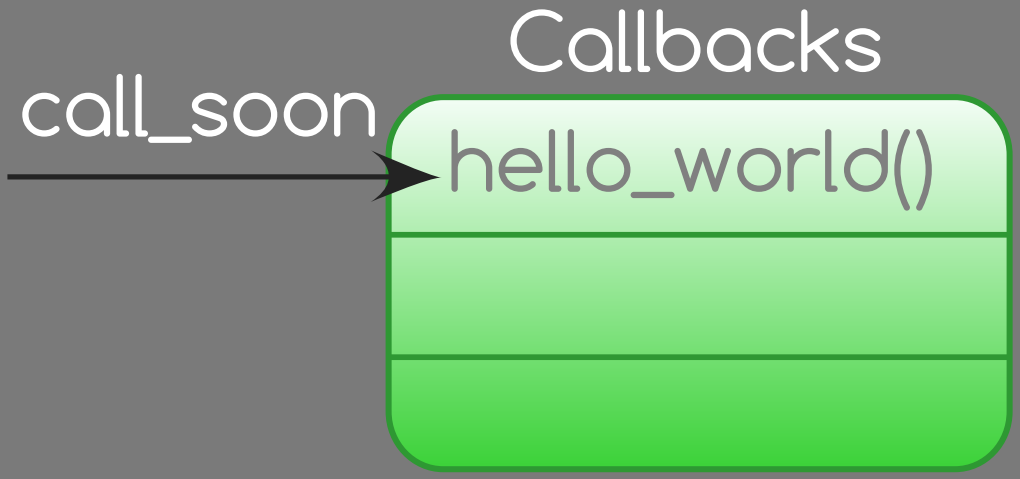
Output

Callbacks



Code

```
.call_soon(hello_world)
.execute_callbacks()
```



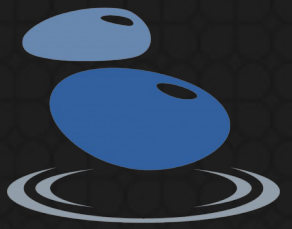
Output

```
Hello World!
```

eNovance Multi-Cloud Experts

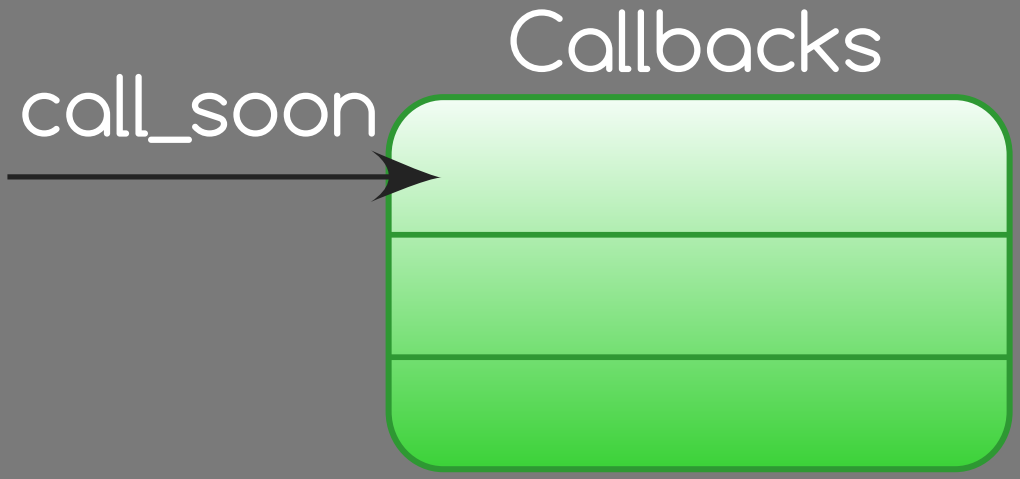
python™

Callbacks



Code

```
.call_soon(hello_world)
.execute_callbacks()
```



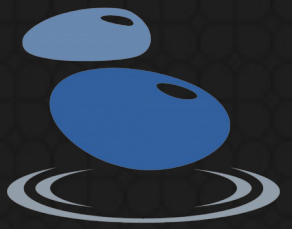
Output

```
Hello World!
```

eNovance Multi-Cloud Experts

python™

Timers

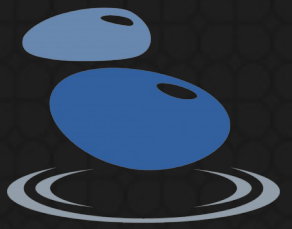


```
class TimerEventLoop(CallbackEventLoop):

    def __init__(self):
        super().__init__()
        self.timers = []

    def call_at(self, when, func):
        timer = (when, func)
        self.timers.append(timer)

    ...
```

eNovance Multi-Cloud Experts

python™

Timers

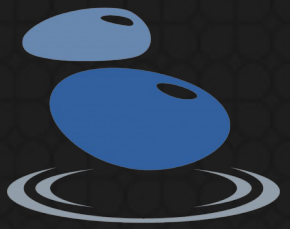


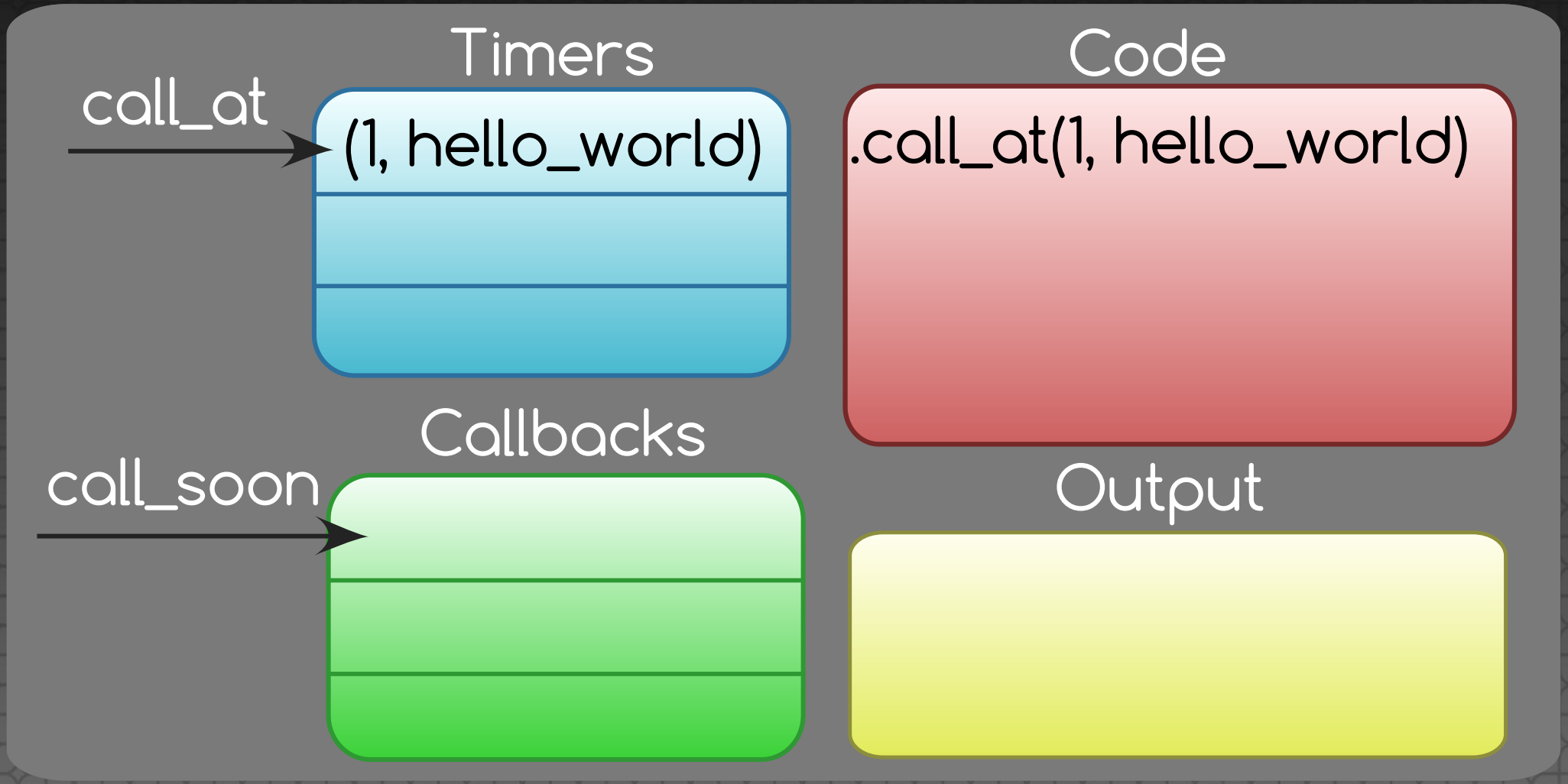
```
class TimerEventLoop(CallbackEventLoop):
    ...
    def execute_timers(self):
        now = time.time()
        new_timers = []
        for when, func in self.timers:
            if when <= now:
                self.call_soon(func)
            else:
                new_timers.append((when, func))
        self.timers = new_timers

        self.execute_callback()
```

eNovance Multi-Cloud Experts

python™

Timers

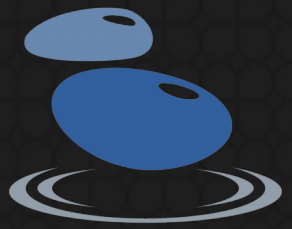


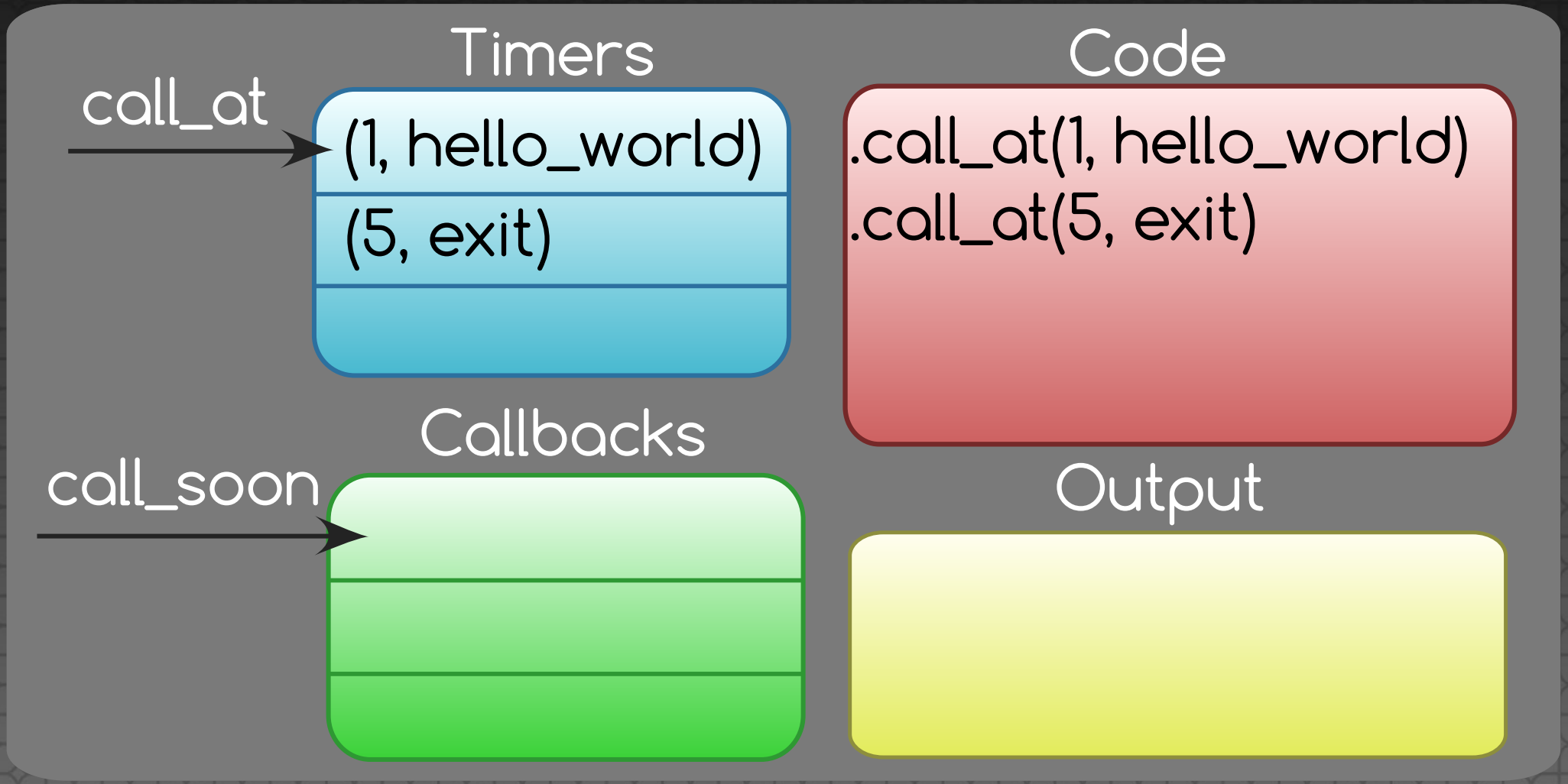


eNovance
Multi-Cloud Experts

python™

Timers

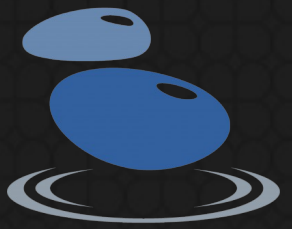


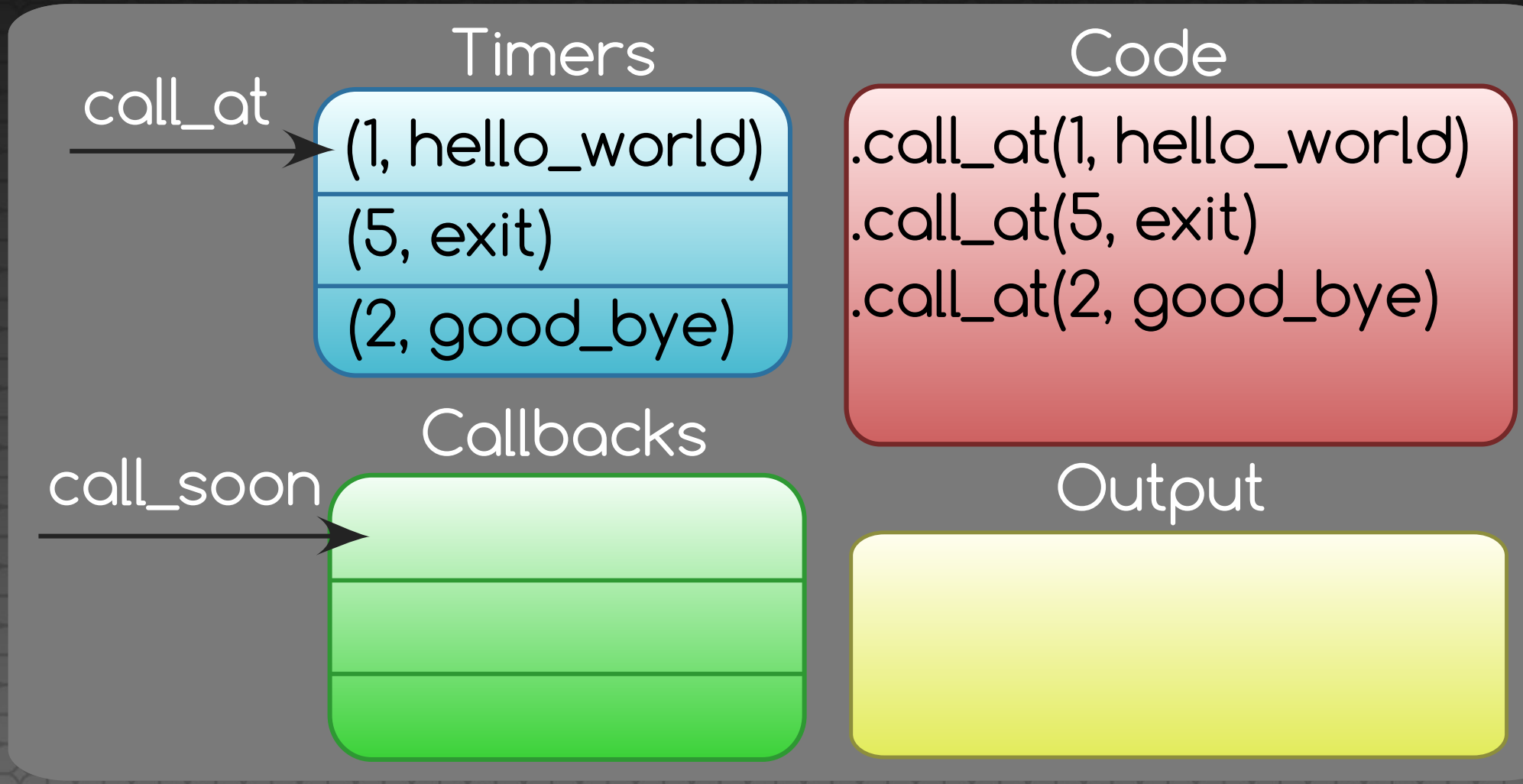


eNovance Multi-Cloud Experts

python™

Timers

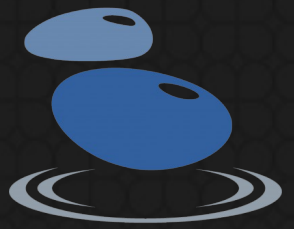


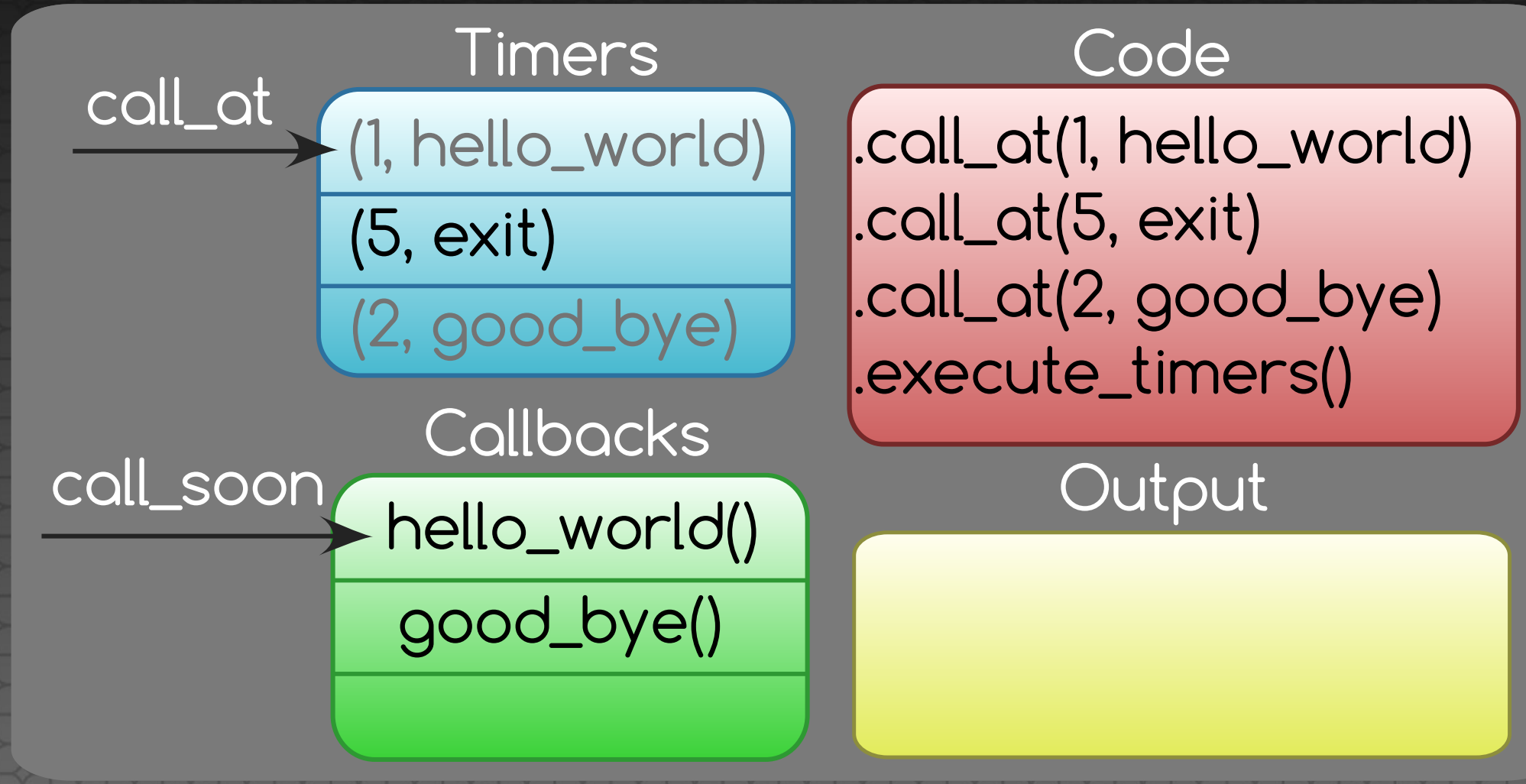


eNovance Multi-Cloud Experts

python™

Timers

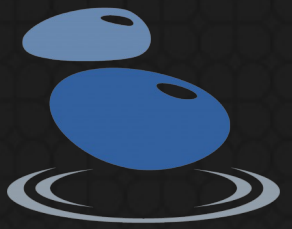




eNovance Multi-Cloud Experts

python™

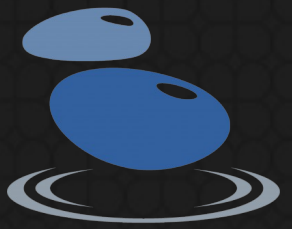
Timers



eNovance Multi-Cloud Experts

python™

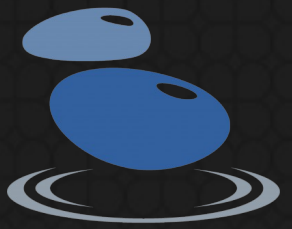
Timers



eNovance Multi-Cloud Experts

python™

Selector

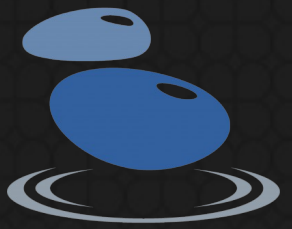


```
class SelectorEventLoop(TimerEventLoop):

    def __init__(self):
        super().__init__()
        self.selector = selectors.Selector()

    def add_reader(self, sock, func):
        self.selector.register(sock,
                               selectors.EVENT_READ, func)

    ...
```

eNovance Multi-Cloud Experts

python™

Selector

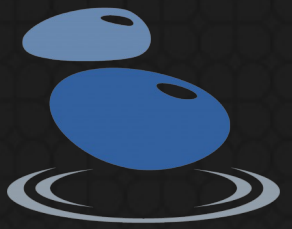


```
class SelectorEventLoop(TimerEventLoop):
    ...
    def select(self):
        timeout = self.compute_timeout()

        events = self.selector.select(timeout)
        for key, mask in events:
            func = key.data
            self.call_soon(func, key.fileobj)

        self.execute_timers()
```

eNovance Multi-Cloud Experts

python™

Selector

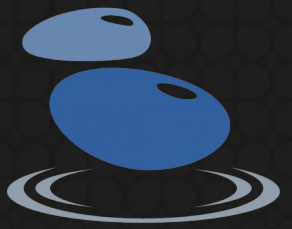


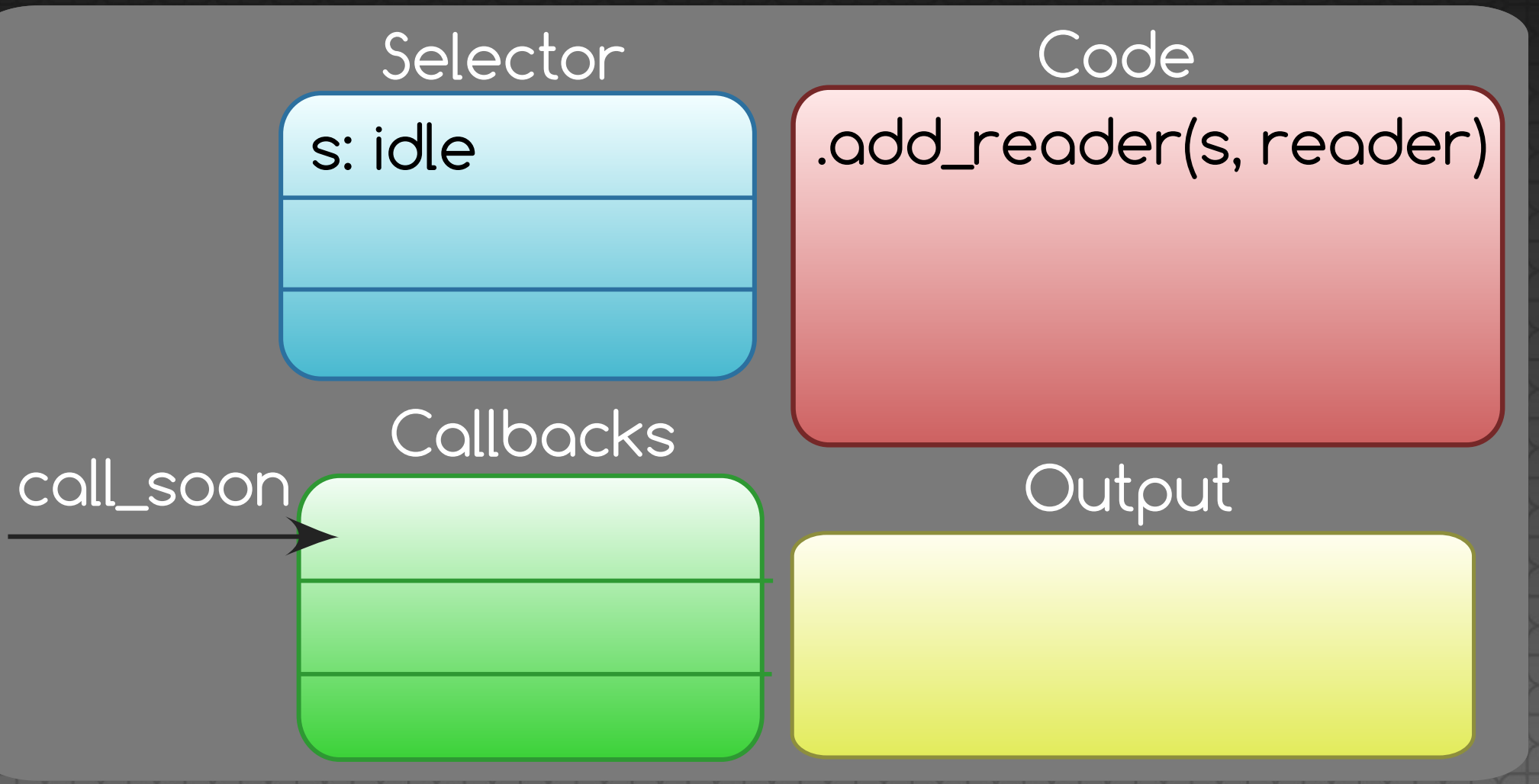
```
class SelectorEventLoop(TimerEventLoop):
    ...
    def compute_timeout(self):
        if self.callbacks:
            # already something to do
            return 0
        elif self.timers:
            next_timer = min(self.timers)[0]
            timeout = next_timer - time.time()
            return max(timeout, 0.0)
        else:
            # blocking call
            return None
```

eNovance Multi-Cloud Experts

python™

Selector

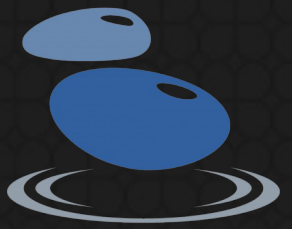


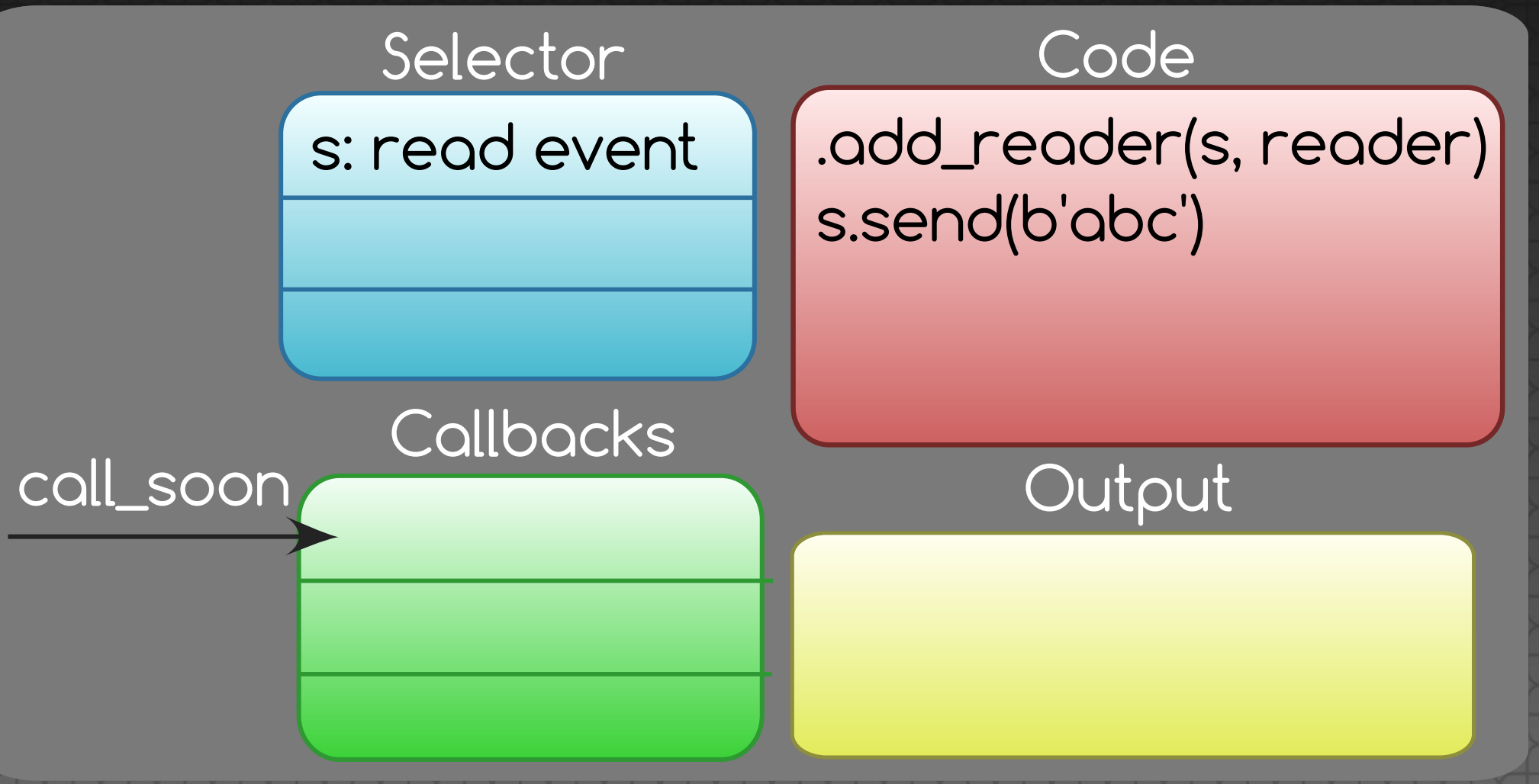


eNovance
Multi-Cloud Experts

python™

Selector

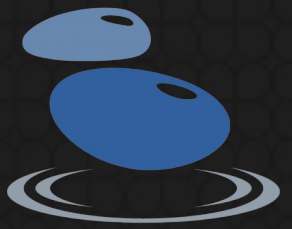




eNovance Multi-Cloud Experts

python™

Selector



Selector

s: read event

Code

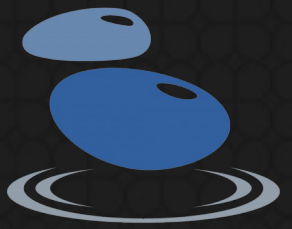
```
.add_reader(s, reader)
s.send(b'abc')
.select()
```

Callbacks

call_soon → reader()

Output

Selector



Selector

s: idle

Code

```
.add_reader(s, reader)
s.send(b'abc')
.select()
```

Callbacks

call_soon → reader()

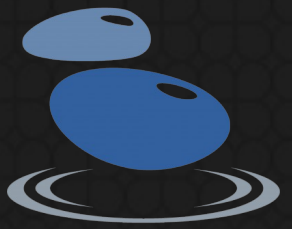
Output

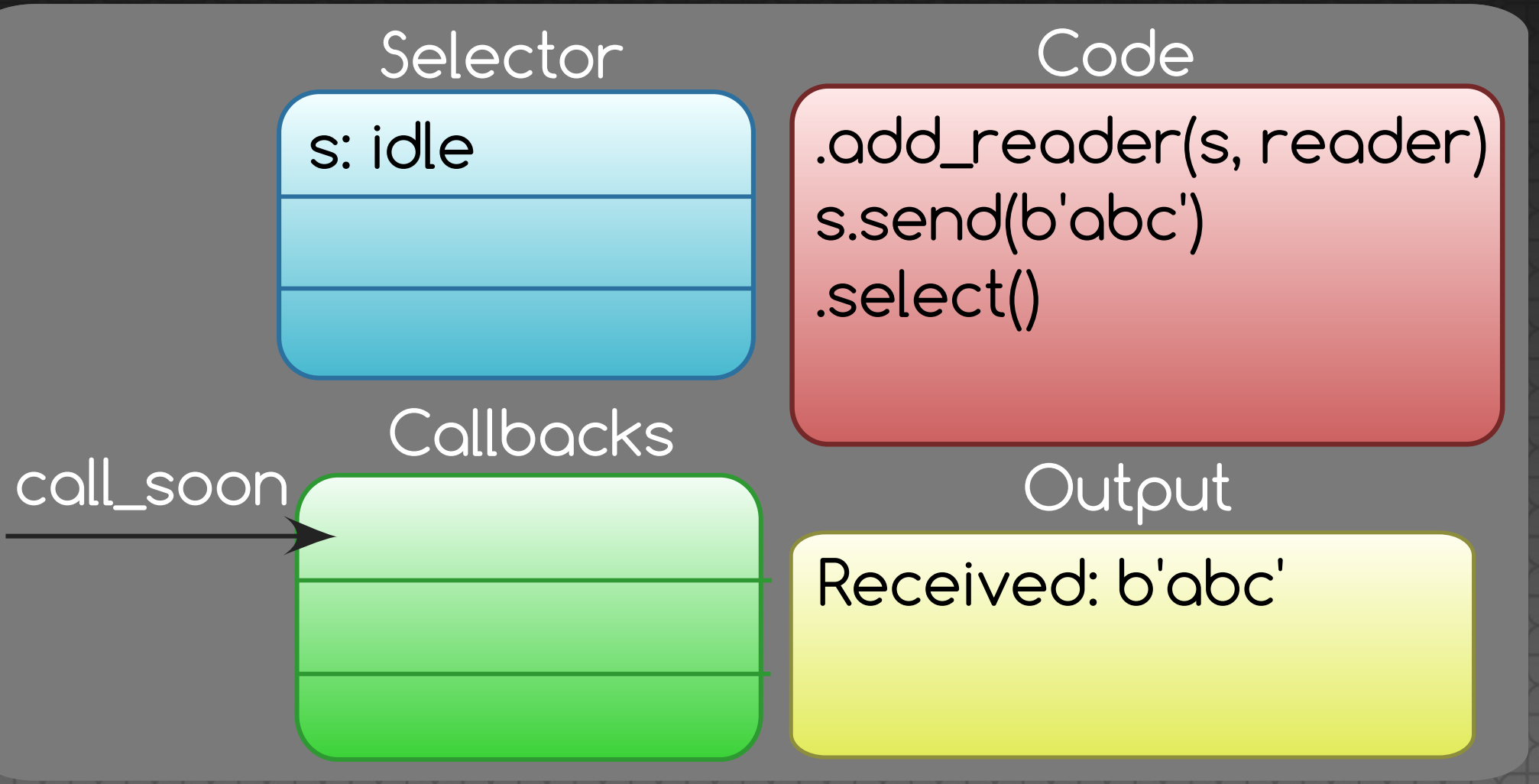
Received: b'abc'

eNovance Multi-Cloud Experts

python™

Selector

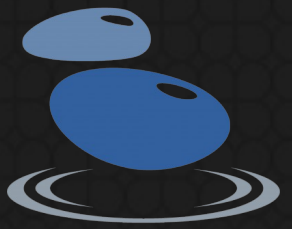




eNovance Multi-Cloud Experts

python™

Generator



Code

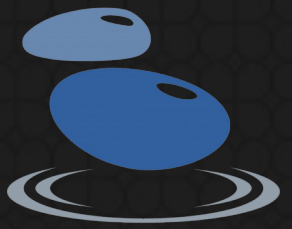
```
gen = producer()
```

producer() generator

```
yield "start"
return "stop"
```

Output

Generator



Code

```
gen = producer()
print(gen.next())
```

producer() generator

```
yield "start"
return "stop"
```

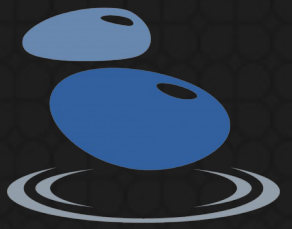
Output

```
start
```

eNovance Multi-Cloud Experts

python™

Generator



Code

```
gen = producer()
print(gen.next())
try:
    gen.next()
except StopIteration as e:
    print(e.value)
```

producer() generator

```
yield "start"
return "stop"
```

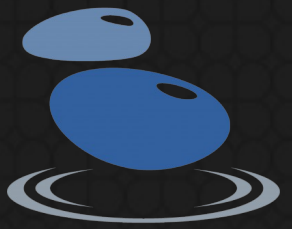
Output

```
start
stop
```

eNovance Multi-Cloud Experts

python™

Coroutine

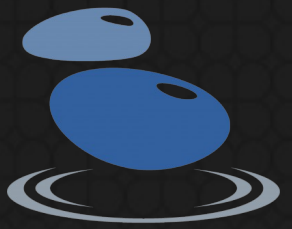


```
@asyncio.coroutine
def my_coroutine(future):
    res = yield from future
    return res
```

eNovance Multi-Cloud Experts

python™

BaseTask

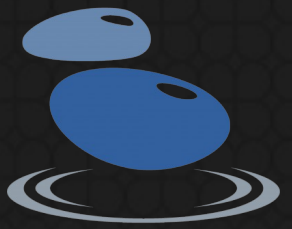


```
class BaseTask:
    def __init__(self, coro):
        self.coro = coro

    def step(self):
        try:
            next(self.coro)
        except StopIteration:
            pass
```

eNovance Multi-Cloud Experts

python™

BaseTask



Code

```
coro = test_coro()
task = BaseTask(coro)
```

test_coro() coroutine

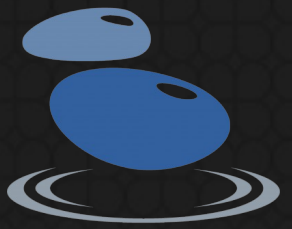
```
print("begin")
yield from ["hack"]
print("end")
```

Output

eNovance Multi-Cloud Experts

python™

BaseTask



Code

```
coro = test_coro()
task = BaseTask(coro)
task.step()
```

test_coro() coroutine

```
print("begin")
yield from ["hack"]
print("end")
```

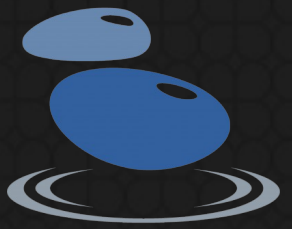
Output

```
begin
```

eNovance Multi-Cloud Experts

python™

BaseTask



Code

```
coro = test_coro()
task = BaseTask(coro)
task.step()
task.step()
```

test_coro() coroutine

```
print("begin")
yield from ["hack"]
print("end")
```

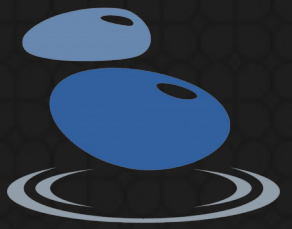
Output

```
begin
end
```

eNovance Multi-Cloud Experts

python™

Future

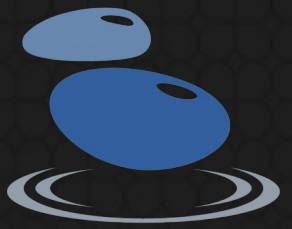


```
class Future:

    def __init__(self, loop):
        self.loop = loop
        self.callbacks = []
        self._result = None

    def add_done_callback(self, func):
        self.callbacks.append(func)

    def result(self):
        return self._result
    ...
```

eNovance Multi-Cloud Experts

python™

Future

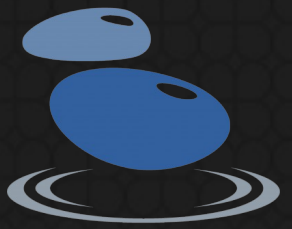


```
class Future:
    ...
    def set_result(self, result):
        self._result = result
        for func in self.callbacks:
            self.loop.call_soon(func, self)

    def __iter__(self):
        # “yield from future” yields future
        yield self
```

eNovance Multi-Cloud Experts

python™

Task

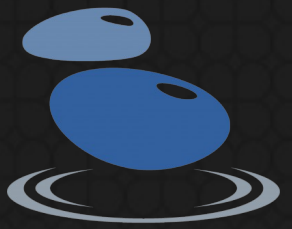


```
class Task:

    def __init__(self, coro, loop):
        self.coro = coro
        loop.call_soon(self.step)
    ...
```

eNovance Multi-Cloud Experts

python™

Task



```
class Task:
    ...
    def step(self):
        try:
            result = next(self.coro)
        except StopIteration:
            pass
        else:
            if isinstance(result, Future):
                result.add_done_callback(
                                    self.step)
```

eNovance Multi-Cloud Experts

python™

Questions?

http://docs.python.org/dev/library/asyncio.html

http://github.com/haypo/
http://bitbucket.org/haypo/

Victor Stinner
victor.stinner@gmail.com

Thanks David Malcom for the LibreOffice model

http://dmalcolm.livejournal.com/