PYCON CHINA 2018

./evloop-and-coroutine

浅谈 Python 事件循环与协程

Anqur 2018.10.27 中国 深圳

PYCON
CHINA 2018

About Me

PYCON CHINA 2018

Anqur

@anqurvanillapy

PYCON
CHINA 2018

'14 蟒营

PYCON
CHINA 2018

'16 PyCon 观众

PYCON
CHINA 2018

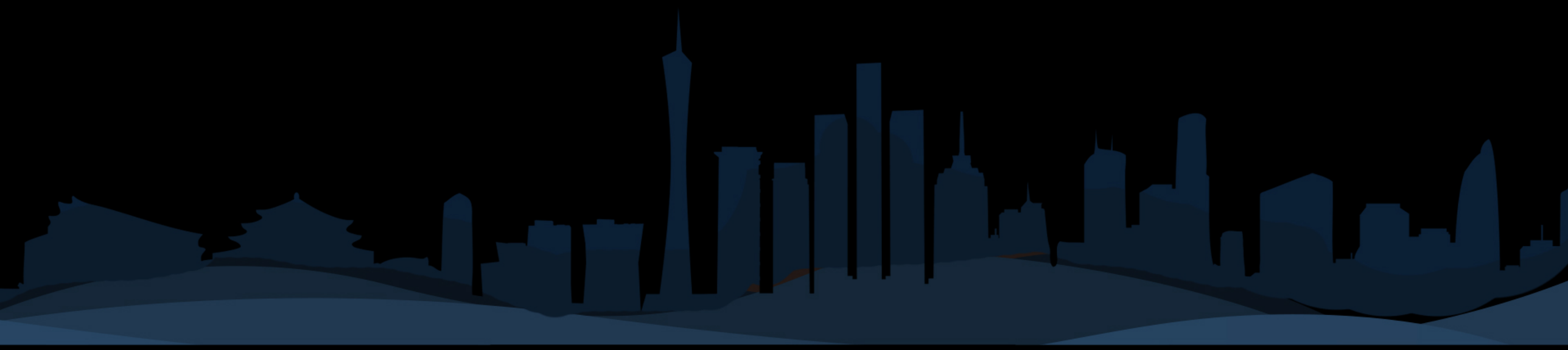
'18 搬砖

PYCON
CHINA 2018

菊厂

PYCON
CHINA 2018

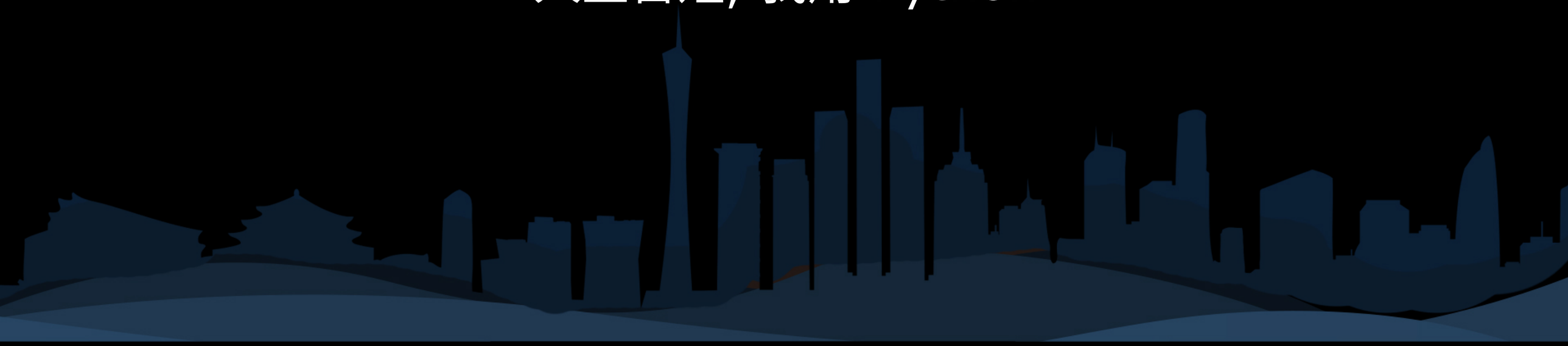
C/Java/Python



PYCON CHINA 2018

Life's short, use Python

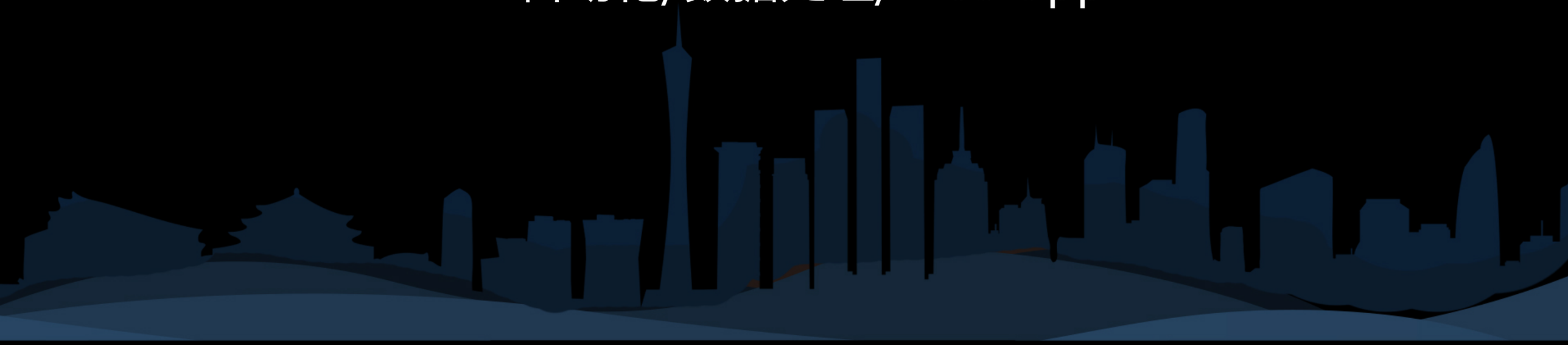
人生苦短, 我用 Python



PYCON CHINA 2018

...As tools

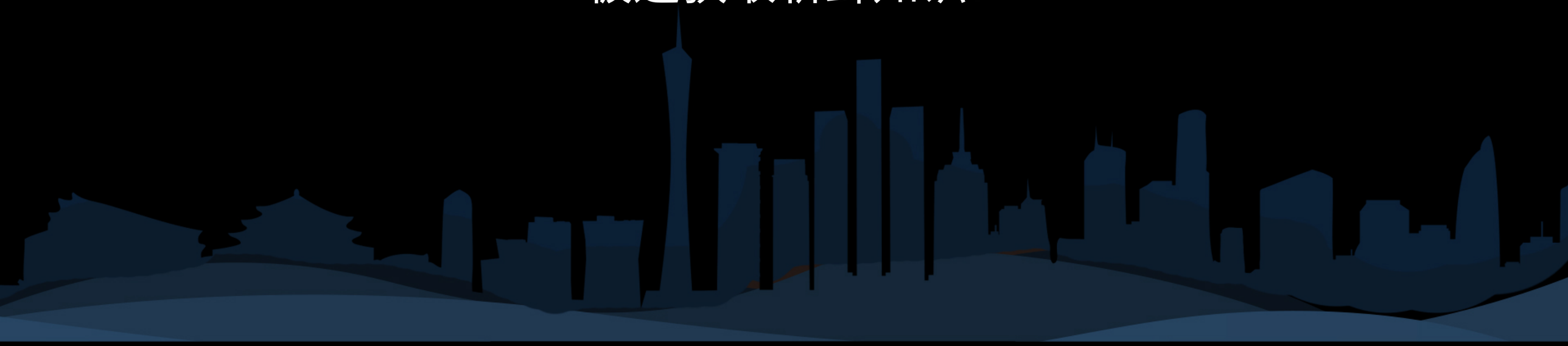
...自动化/数据处理/Web app



PYCON
CHINA 2018

To gain knowledge!

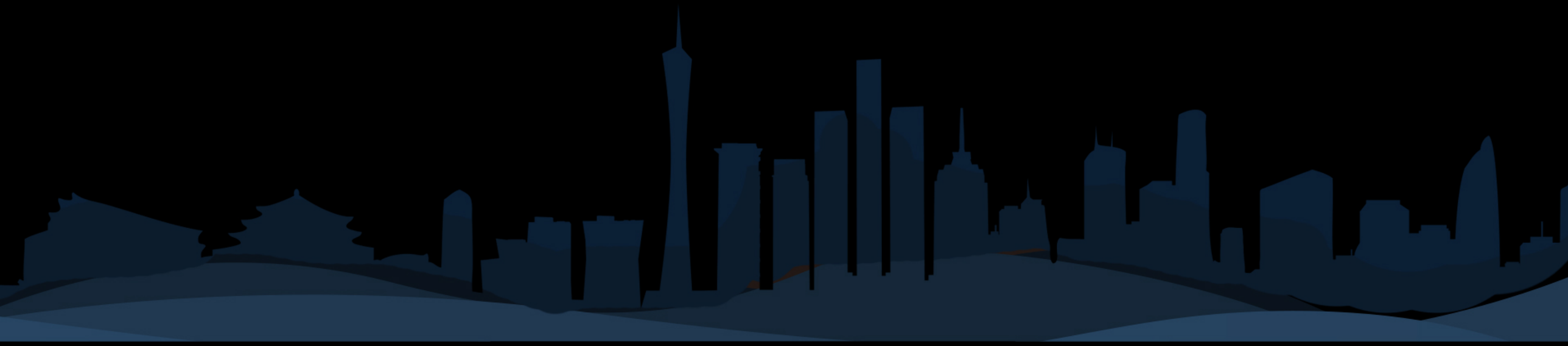
极速获取新鲜知识!



PYCON CHINA 2018

C++20

Coroutine TS



PYCON CHINA 2018

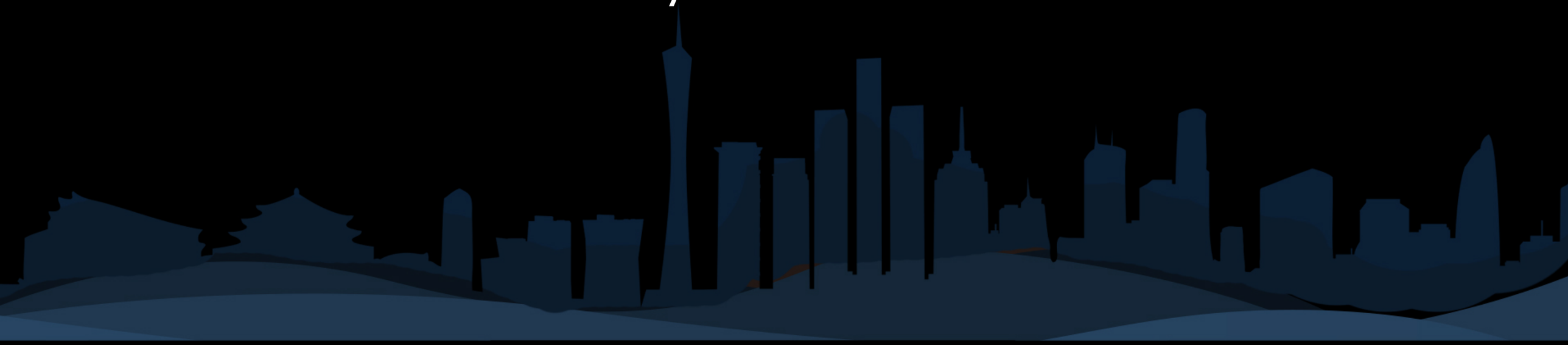
Rust

Borrowing in Async

PYCON CHINA 2018

ECMAScript 6

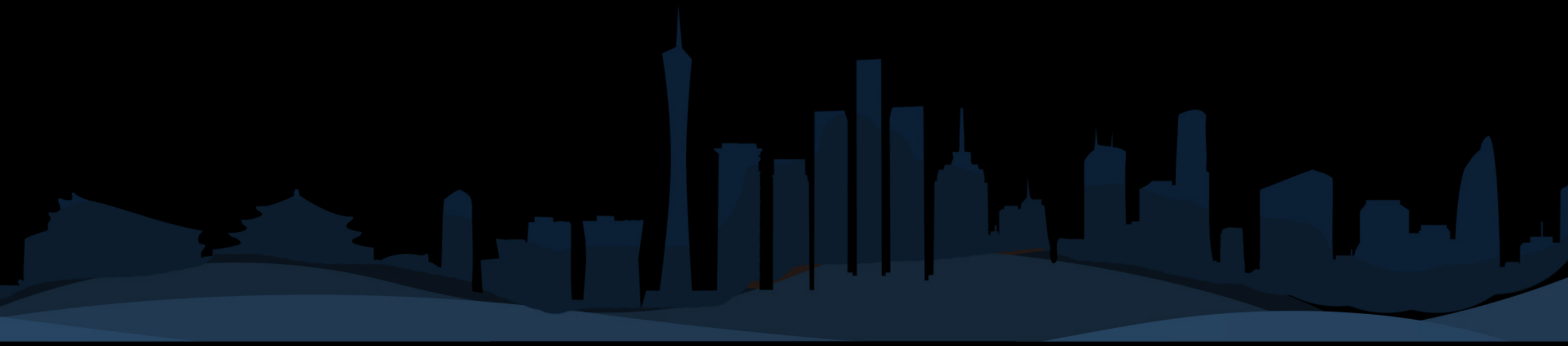
Async Function



PYCON CHINA 2018

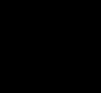
Go

Goroutine



PYCON CHINA 2018

Python


asyncio


PYCON CHINA 2018

Event Loop

and...

Coroutine!

PYCON CHINA 2018

Why?

有何用?

PYCON CHINA 2018

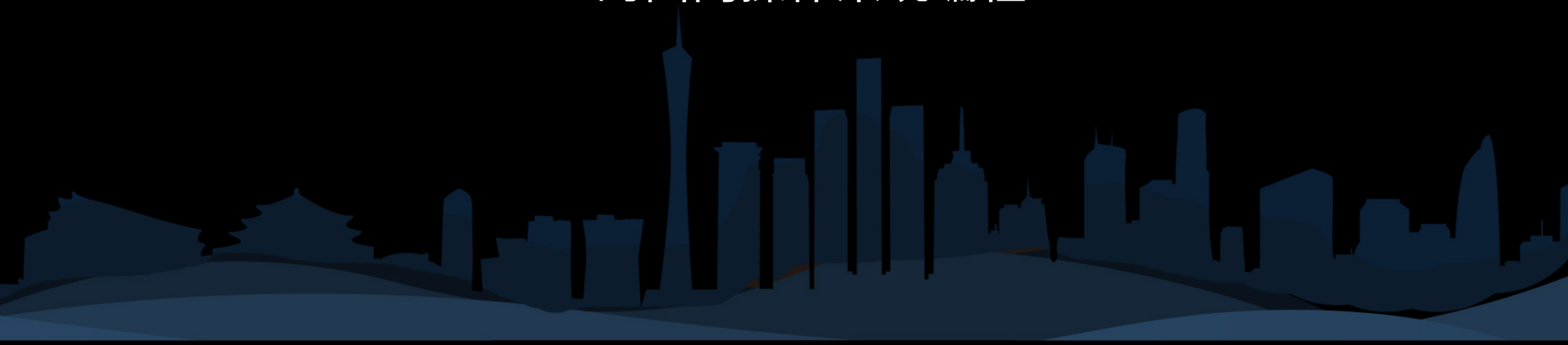
说来话长...

asyncore/concurrent.futures/asyncio...

PYCON CHINA 2018

O(S)OP in C

C 的面向操作系统编程



PYCON
CHINA 2018

Event loop?

PYCON
CHINA 2018

Event?

OS Event

- 套接字
- 计时器
- 信号
- 进程同步

Handle

- sockfd
- timerfd
- signalfd
- pipefd

新连接，连接中断，read readyness，write readyness，端口监听，延时订阅/取消，信号处理订阅/取消...

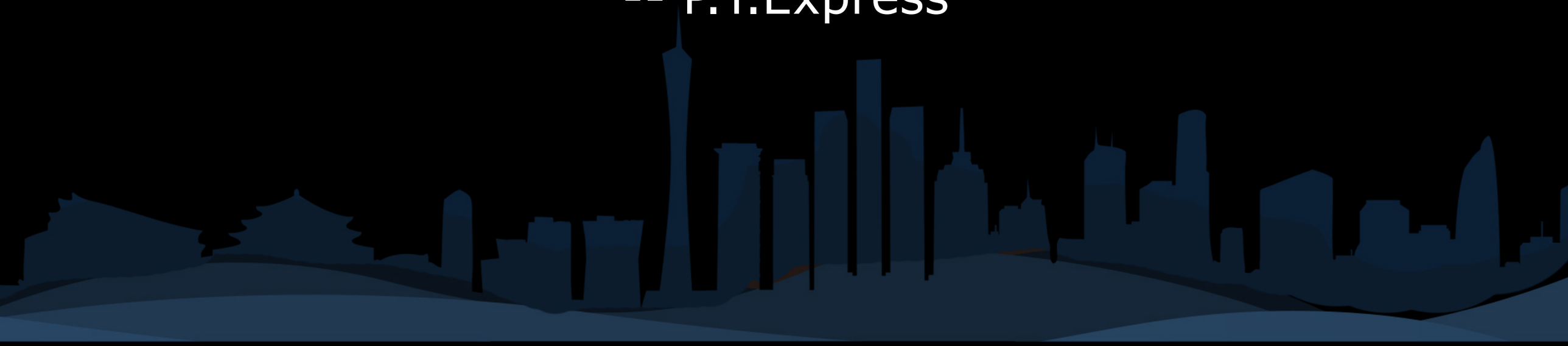
PYCON
CHINA 2018

Loop?

PYCON CHINA 2018

"是你的快递"

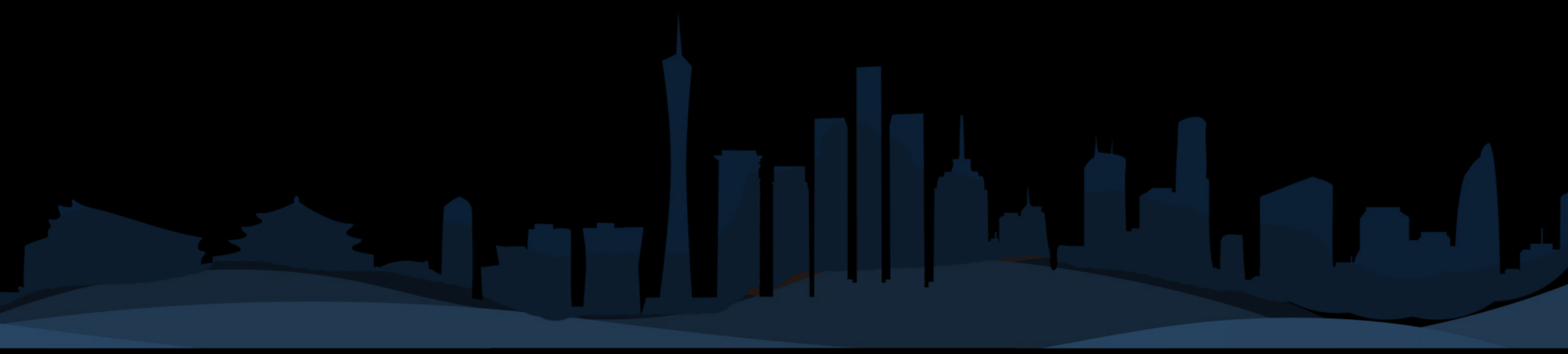
-- P.Y.Express

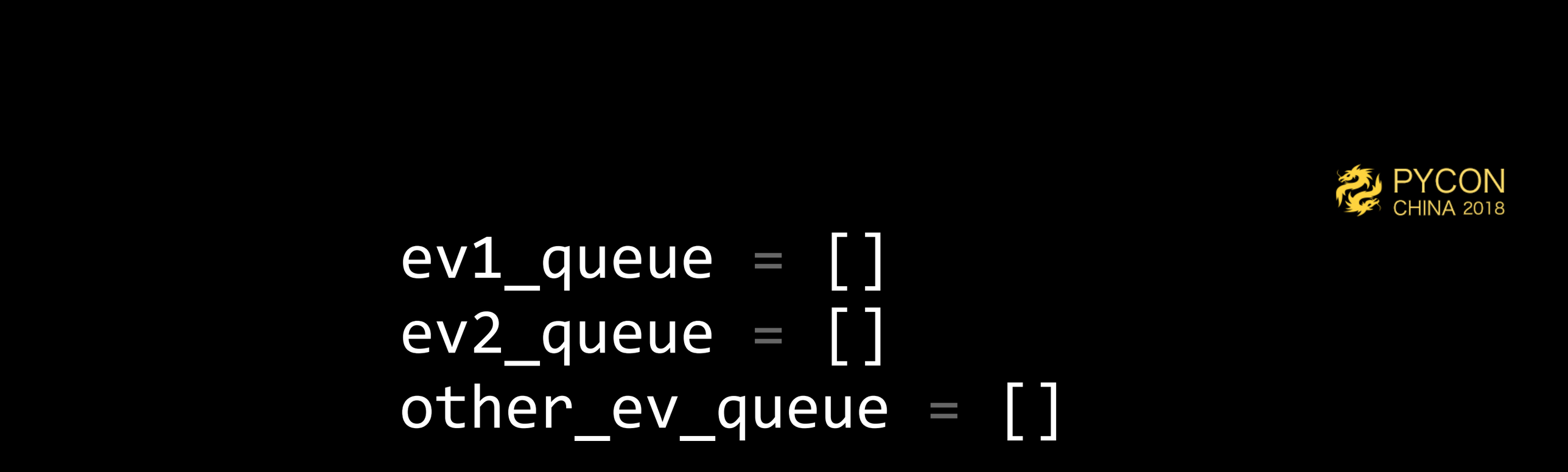



```
events = wait_events()

for ev in events:
    if ev == EVENT1:
        do_use_event1(ev)
    elif ev == EVENT2:
        do_use_event2(ev)
    else:
        do_use_others(ev)
```

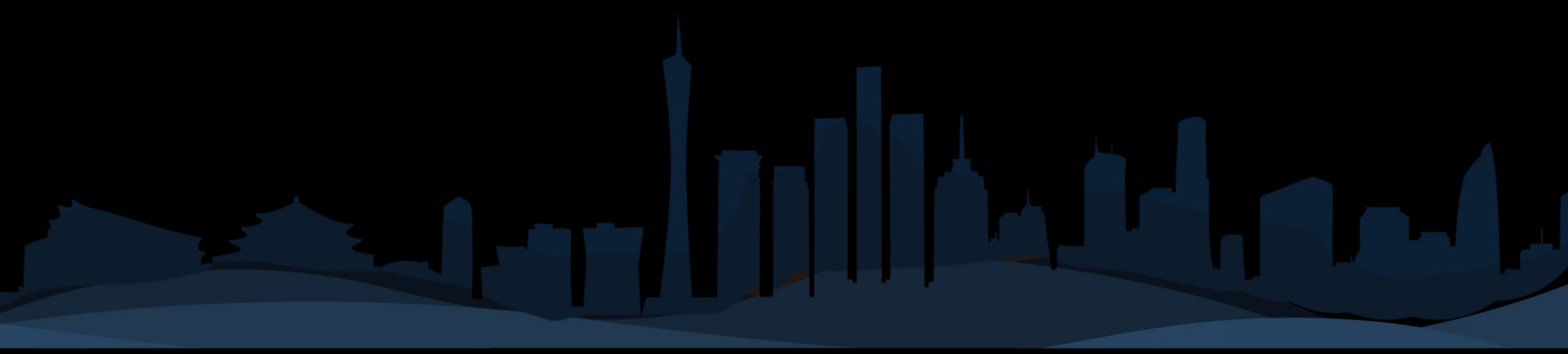
PYCON
CHINA 2018

"我双十一买了好多"




PYCON
CHINA 2018
ev1_queue = []
ev2_queue = []
other_ev_queue = []
events = wait_events()
for ev in events:
# ...

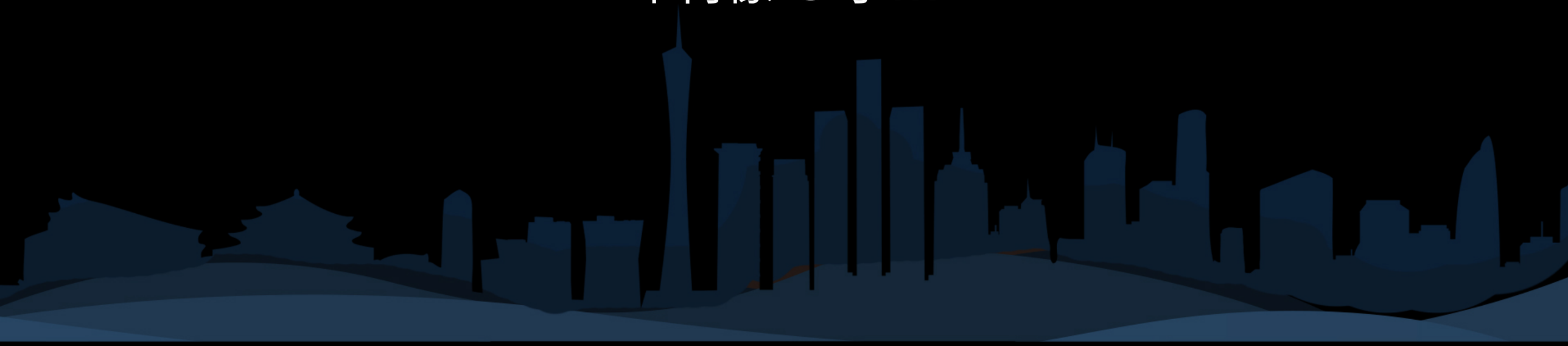
副作用与耦合 :(



PYCON CHINA 2018

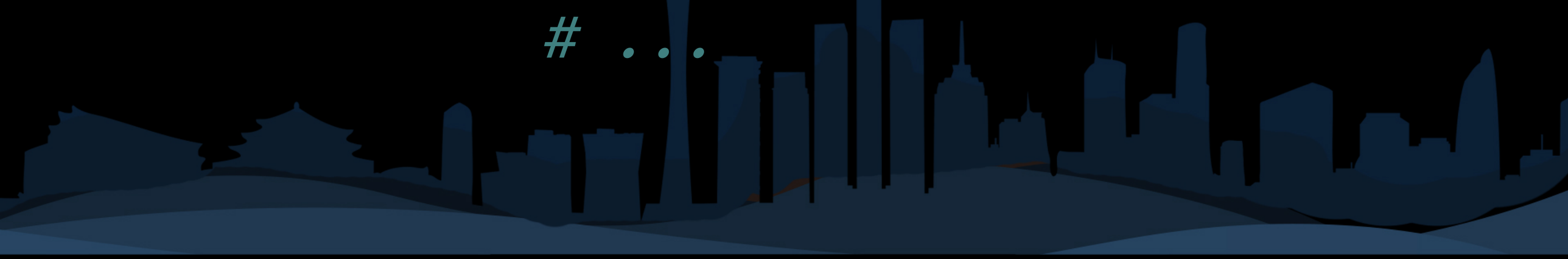
Generator

不再像 C 了...



```python
def event1_gen():
    ev_queue = []

    for _ in range(3):
        # 通过 g.send() 发送子事件.
        ev = yield
        ev_queue.append(ev)

    # 处理 ev_queue...
```

```
ev1_g = event1_gen()

events = wait_events()

for ev in events:
    if ev == EVENT1:
        ev1_g.send(ev)
    # ...
```



:(

- 副作用还在
- 调度逻辑暴露

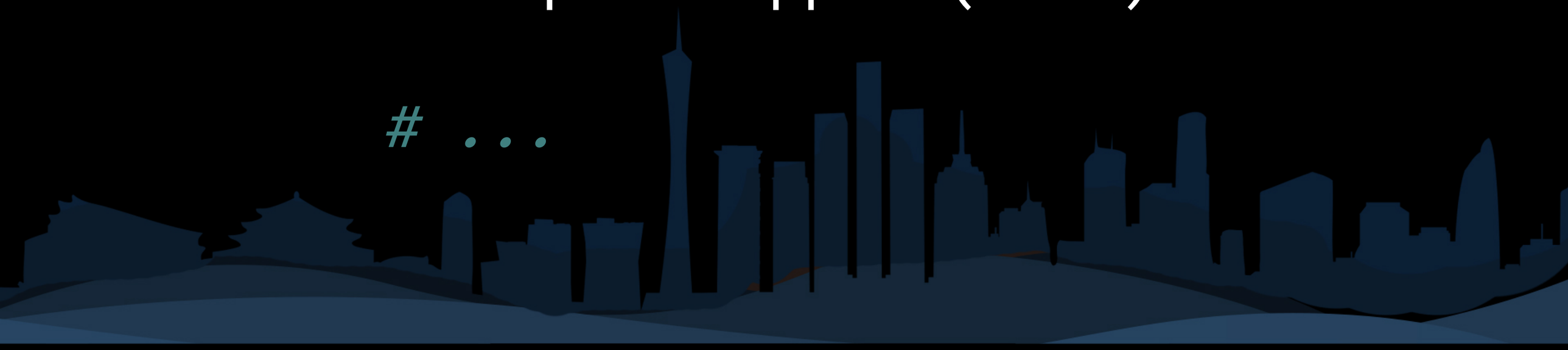
PYCON CHINA 2018

Coroutine

终于...

```
async def main():
    s = pyconio.create_server(
        do_use_event)
    await with s:
        await s.serve_forever()

pyconio.run(main())
```

```
async def do_use_event(reader):
    queue = []

    for _ in range(3):
        data = await reader()
        queue.append(data)

    # ...
```



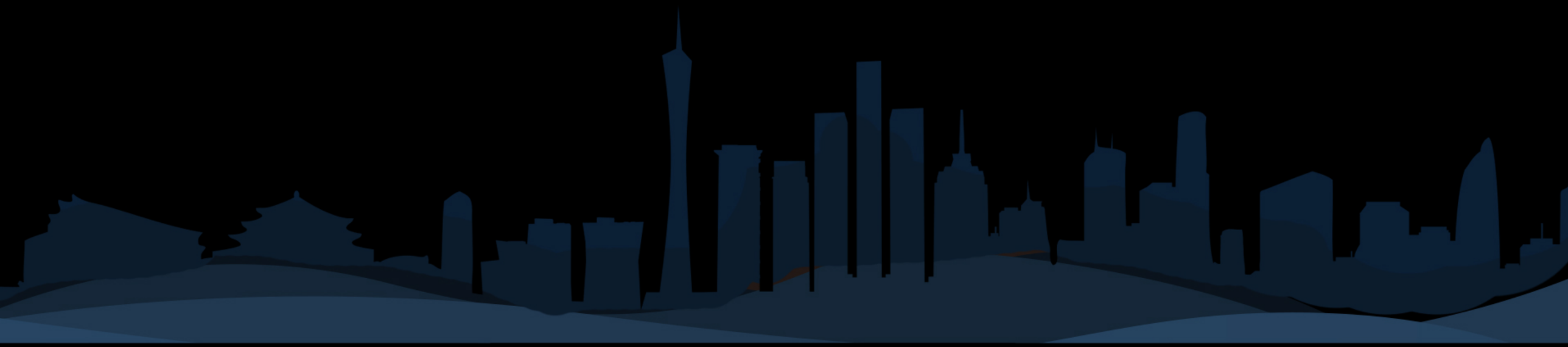
- 隐藏调度细节
- 减少副作用
- 语法支持的类型检查
- 事件良好归类

PYCON CHINA 2018

Revisited

- yield
- async/await

How?



PYCON
CHINA 2018

Learn Py by Py

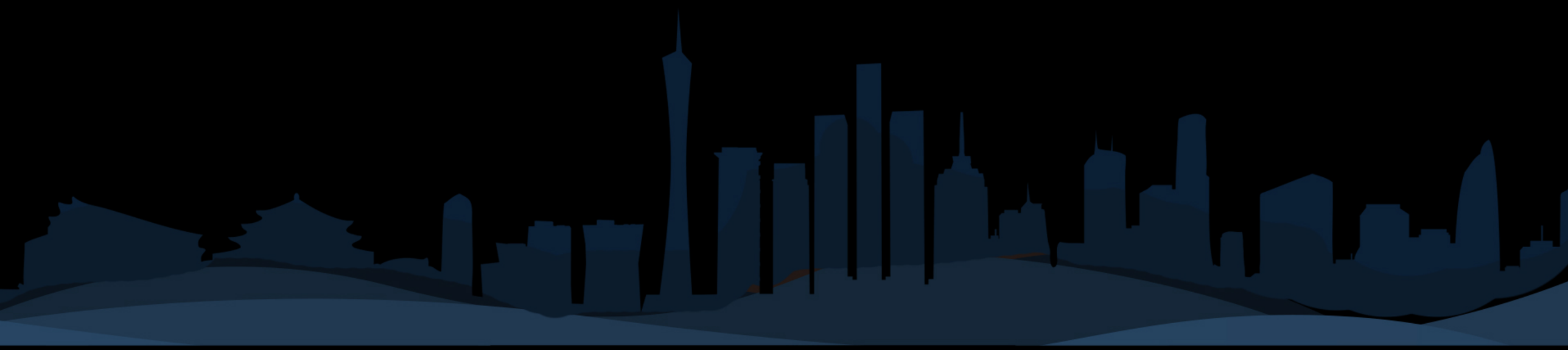
PYCON CHINA 2018

说来话长...

除去细节，保留框架

PYCON
CHINA 2018

Asyncio 和 async/await

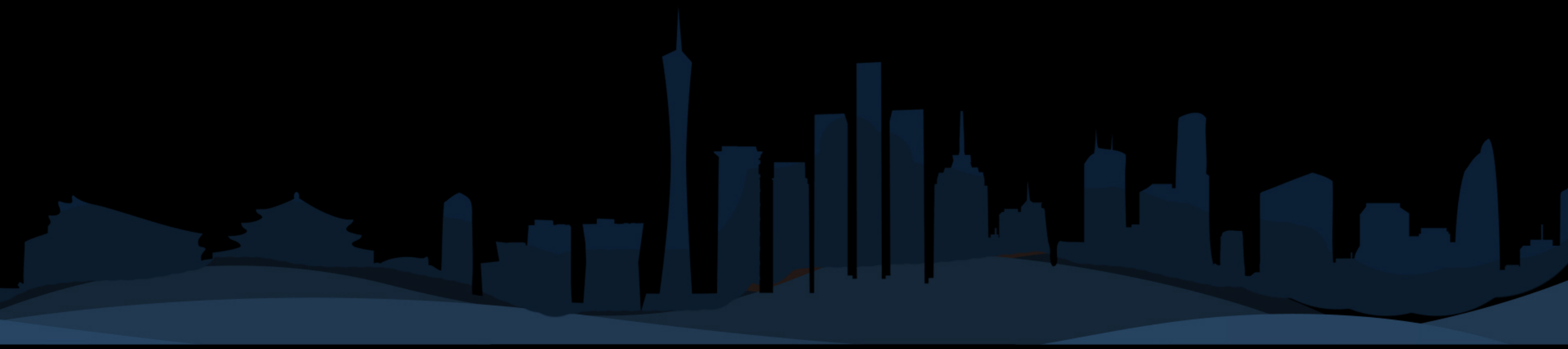


PYCON
CHINA 2018

~~As~~To~~o~~yncio ~~和 async/await~~

PYCON
CHINA 2018

Toyncio



PYCON
CHINA 2018

因为这些坑

PYCON CHINA 2018

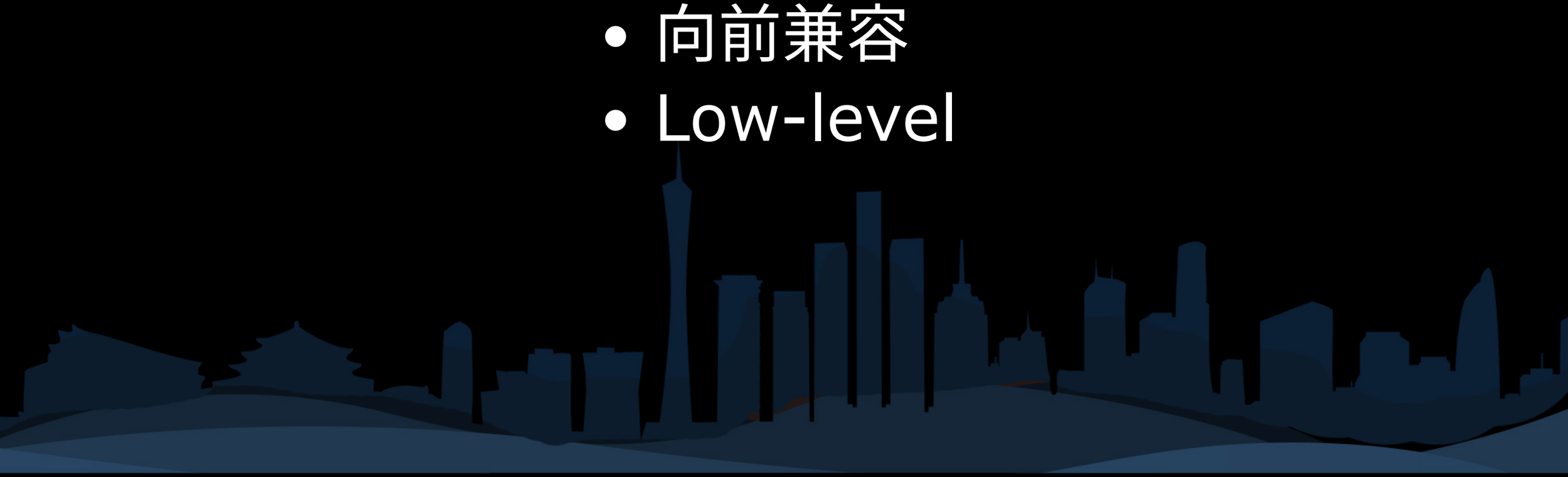
Asyncio

- Task (~~广义的~~)
- ~~Future~~
- Coroutine

PYCON CHINA 2018

Future

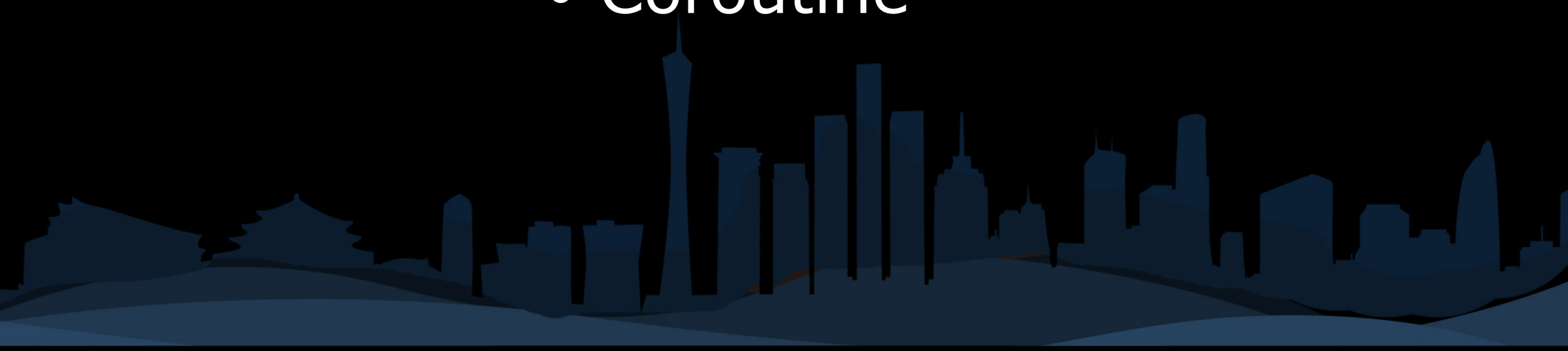
- 向前兼容
- Low-level



PYCON CHINA 2018

Asyncio

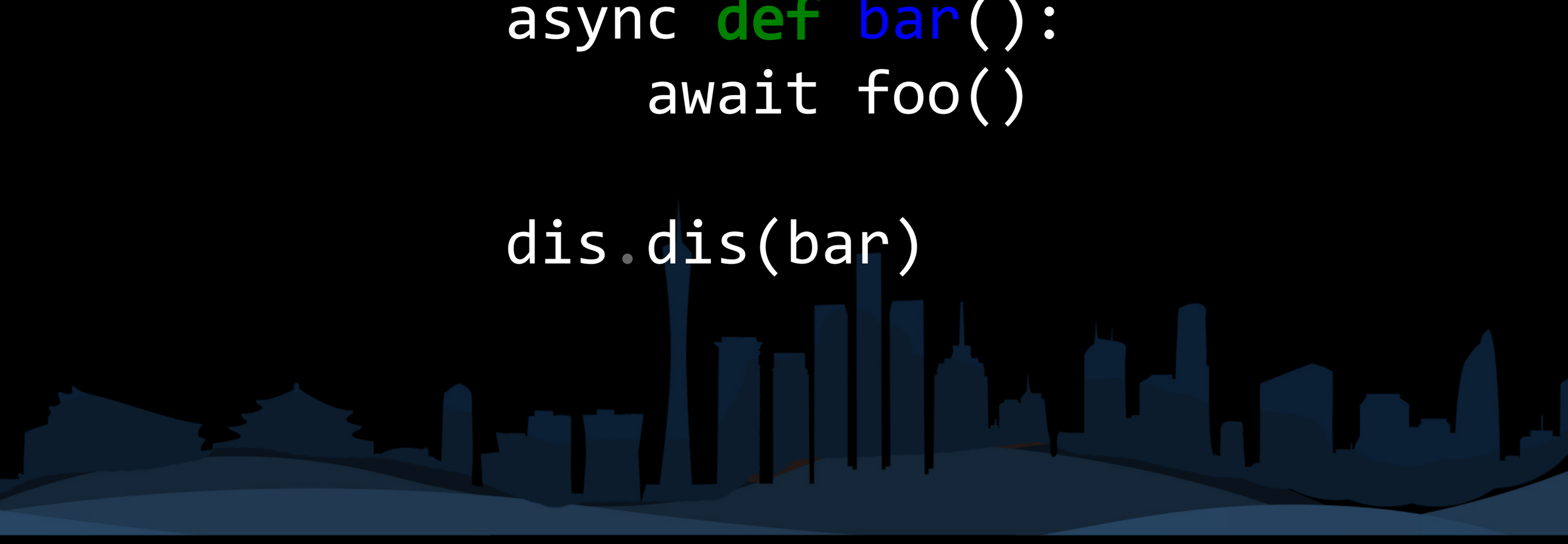
- Task (狭义的)
- Coroutine



PYCON
CHINA 2018

async/await

```
import dis

async def bar():
    await foo()

dis.dis(bar)
```



```
0 LOAD_GLOBAL              0 (foo)
2 CALL_FUNCTION            0
4 GET_AWAITABLE
6 LOAD_CONST               0 (None)
8 YIELD_FROM
# Return function...
```

调用 `foo`

```
0 LOAD_GLOBAL              0 (foo)
2 CALL_FUNCTION            0
```

获取一个 coroutine

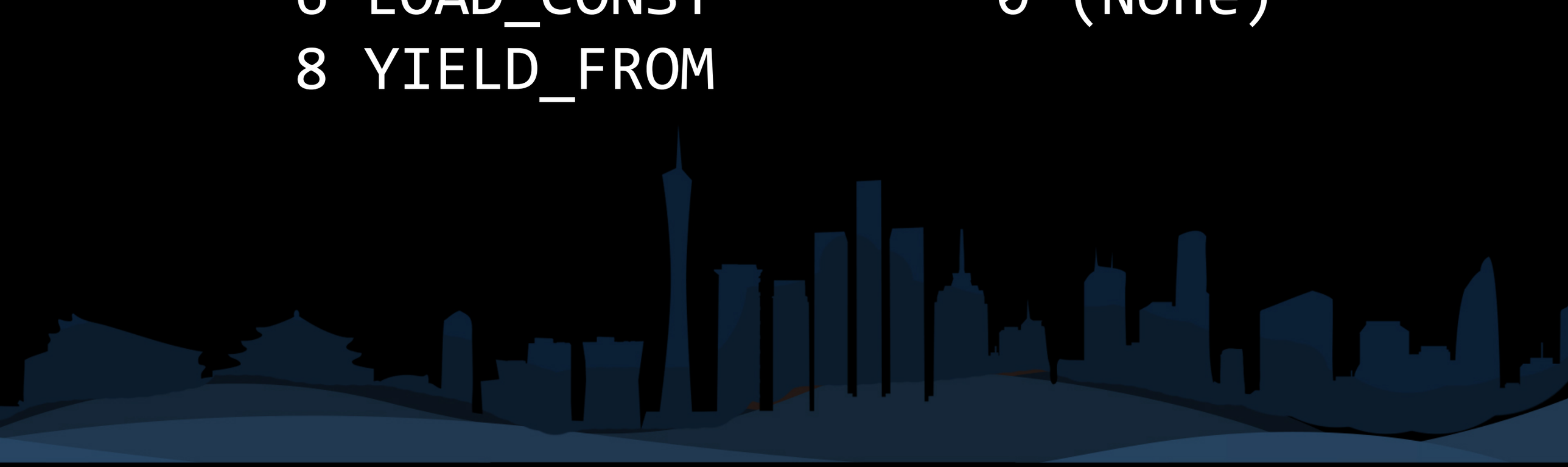
4 GET_AWAITABLE

Awaitable

- 协程对象
- `CO_ITERABLE_COROUTINE` 生成器
- `o.__await__`

执行 yield from

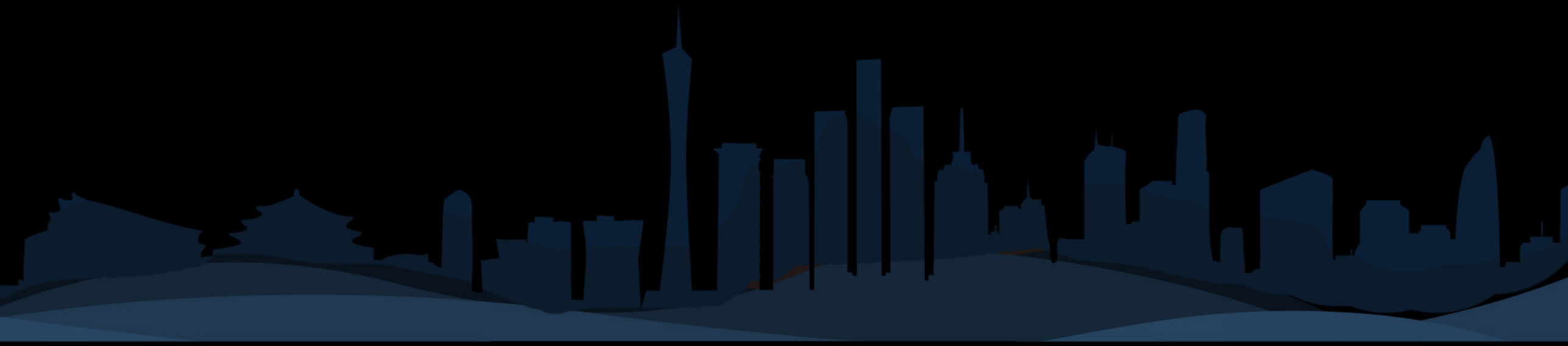
```
6 LOAD_CONST               0 (None)
8 YIELD_FROM
```



PYCON CHINA 2018

Essentially

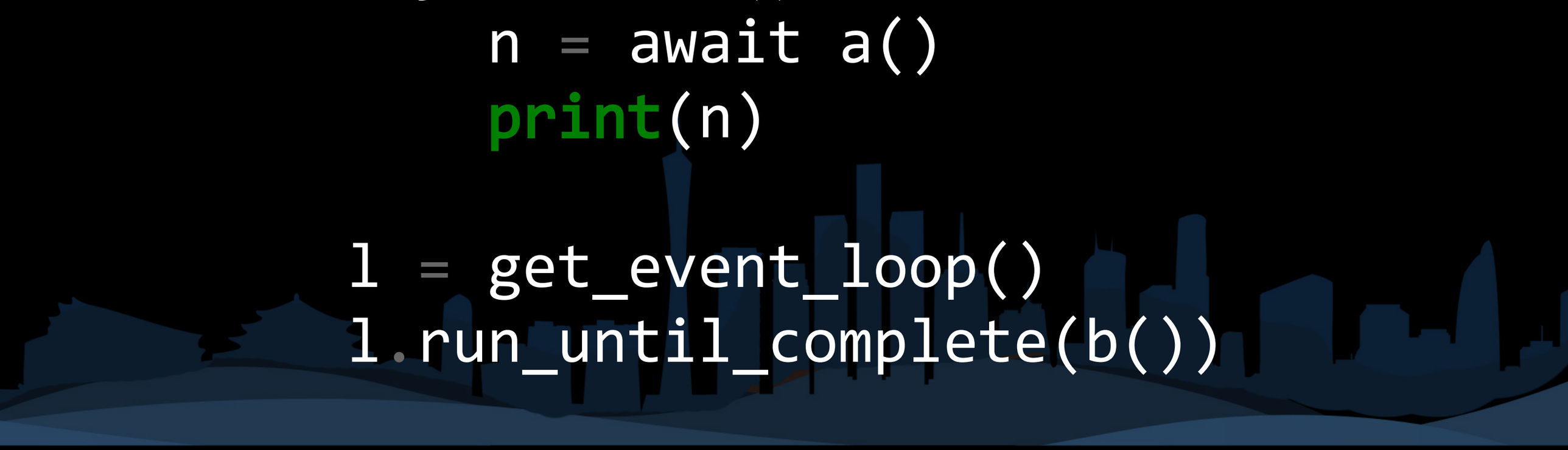
解释器标记的生成器



PYCON
CHINA 2018

基本场景

```
from asyncio import *

asnyc def a():
    return 42

async def b():
    n = await a()
    print(n)

l = get_event_loop()
l.run_until_complete(b())
```



PYCON CHINA 2018

Pdb 猴哇！

Pdb for the win.

```
$ python3 -m pdb pycon.py
(Pdb) s
```

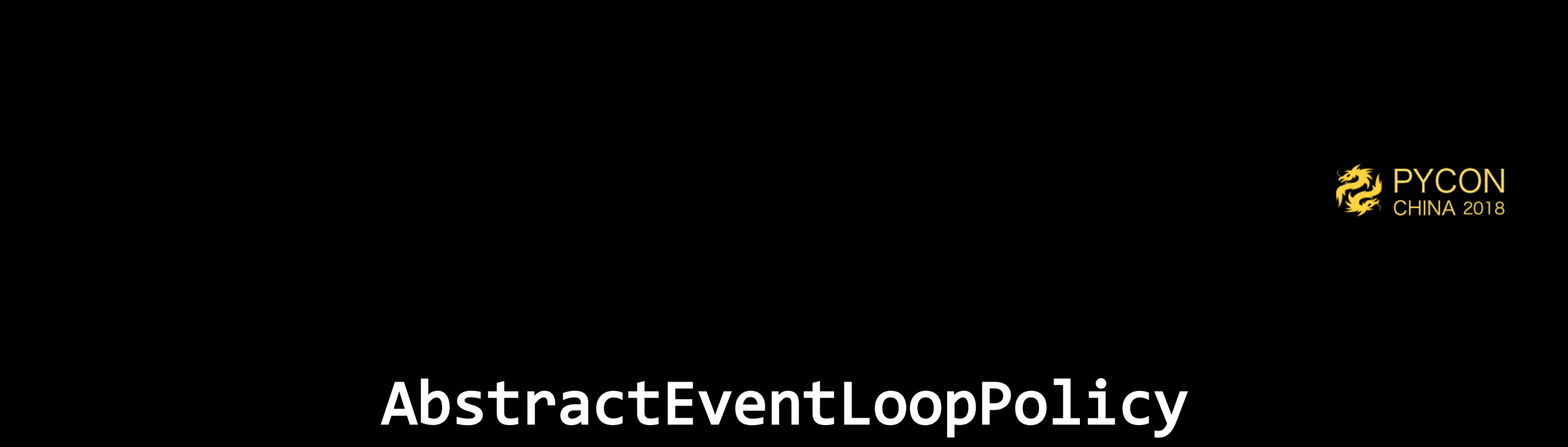
```
l = get_event_loop()
```

获得 policy

```
-> def get_event_loop_policy():
```

AbstractEventLoopPolicy

- `{get|set|new}_event_loop`
- `{get|set}_child_watcher`
- 进程唯一



获得 **event loop**

```
# Thread-local instance.
-> def get_event_loop(self):
```

- `_WindowsSelectorEventLoop`
- `BaseSelectorEventLoop`
- `BaseEventLoop`

BaseSelectorEventLoop

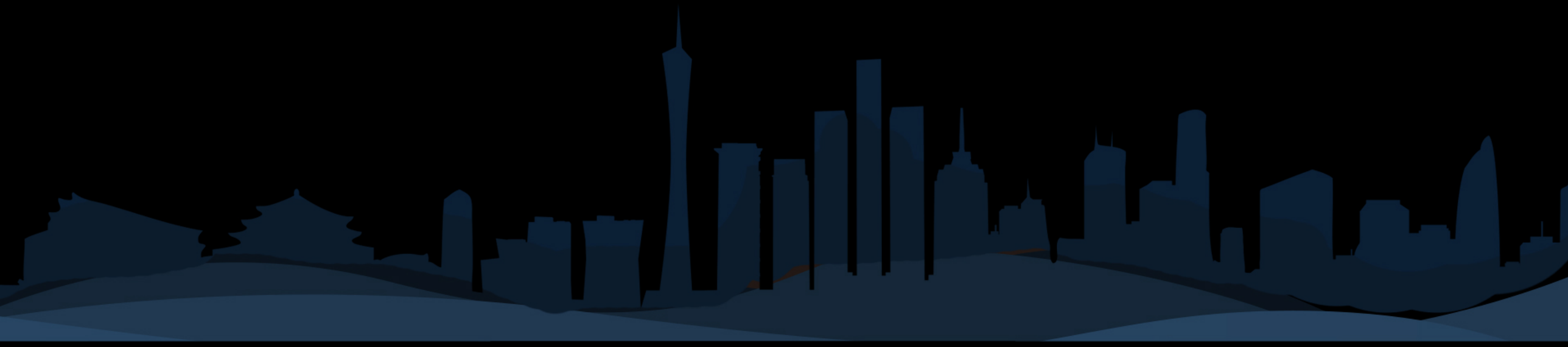
- `{add|remove}_{reader|writer}`
- `sock_*`
- `_make_self_pipe`

BaseEventLoop

- `create_{task|future}`
- `run_{forever|until_complete}`
- `create_server}
- ...

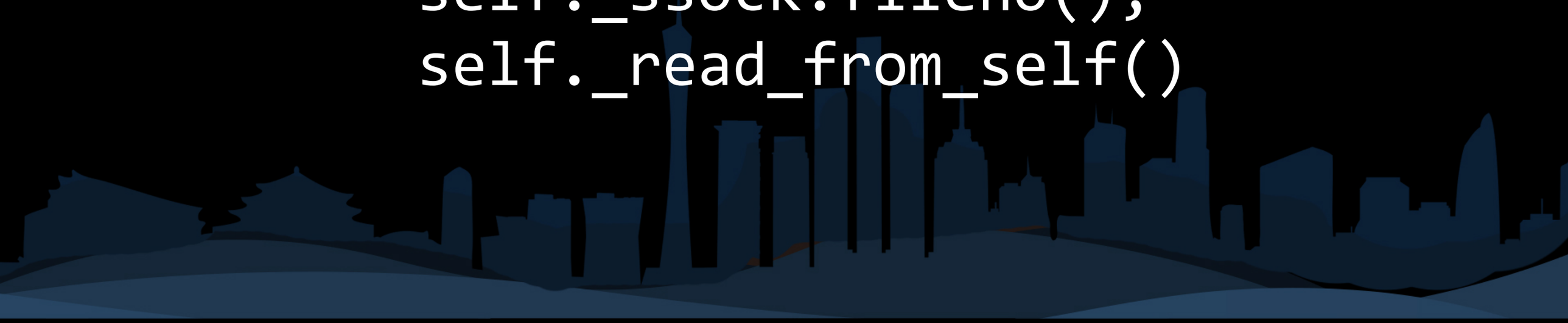
创建 **self pipe**

```
# 跨线程的 event loop 交互
-> def _make_self_pipe(self):
```



socketpair

```
-> self._ssock, self._csock =
     socket.socketpair()
-> # Set non-blocking...
  -> self._add_reader(
       self._ssock.fileno(),
       self._read_from_self()
```



Handle

```
-> handle =
     events.Handle(cb, args, self, None)
-> # When init-ing events.Handle.
-> context =
     contextvar.copy_context()
```

Transports

```
-> self._transports =
   weakref.WeakValueDictionary()
```

Run!

```
-> l.run_until_complete(bar())
-> # Check if it is a coroutine...
```

Task

```
-> task = loop.create_task(coro)
-> tasks.Task(coro, loop=self)
-> self.loop.call_soon(self.__step,
     context=self._context)
-> handle =
     events.Handle(cb, args, self, None)
-> self._ready.append(handle)
```

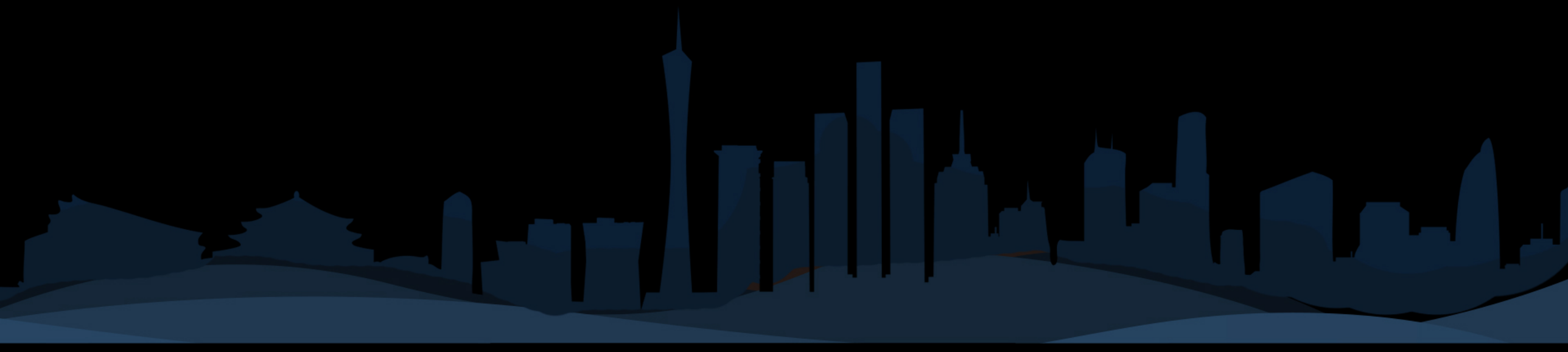
contextvars

- ~~寄存器值~~
- 线程信息: frames, exc...
- GC, GIL, Warnings...
- 参阅 `struct _PyRuntime`

PYCON CHINA 2018

Run forever!

```
-> def run_forever(self):
```

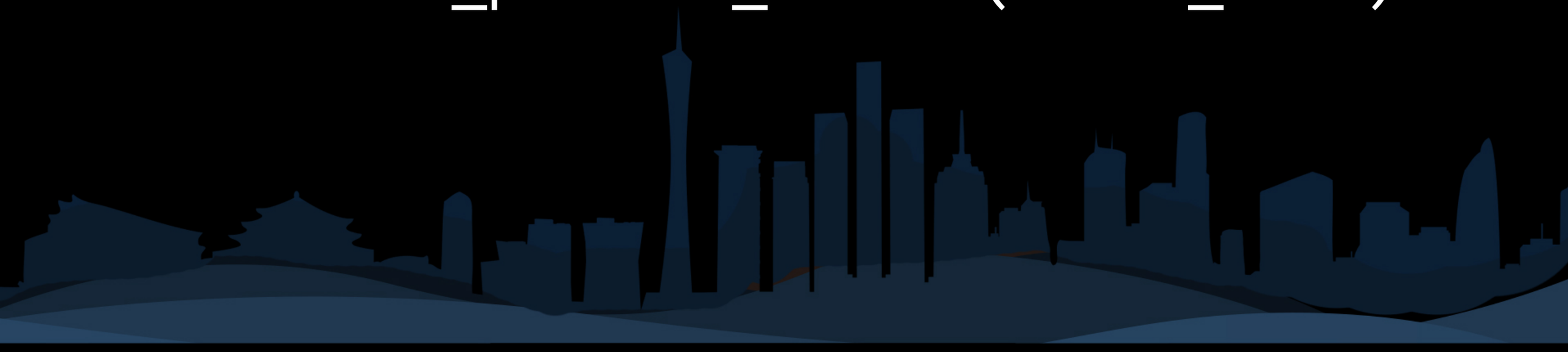


```
events._set_running_loop(self)
while True:
    self._run_once()
    if self._stopping:
        break
```

PYCON CHINA 2018

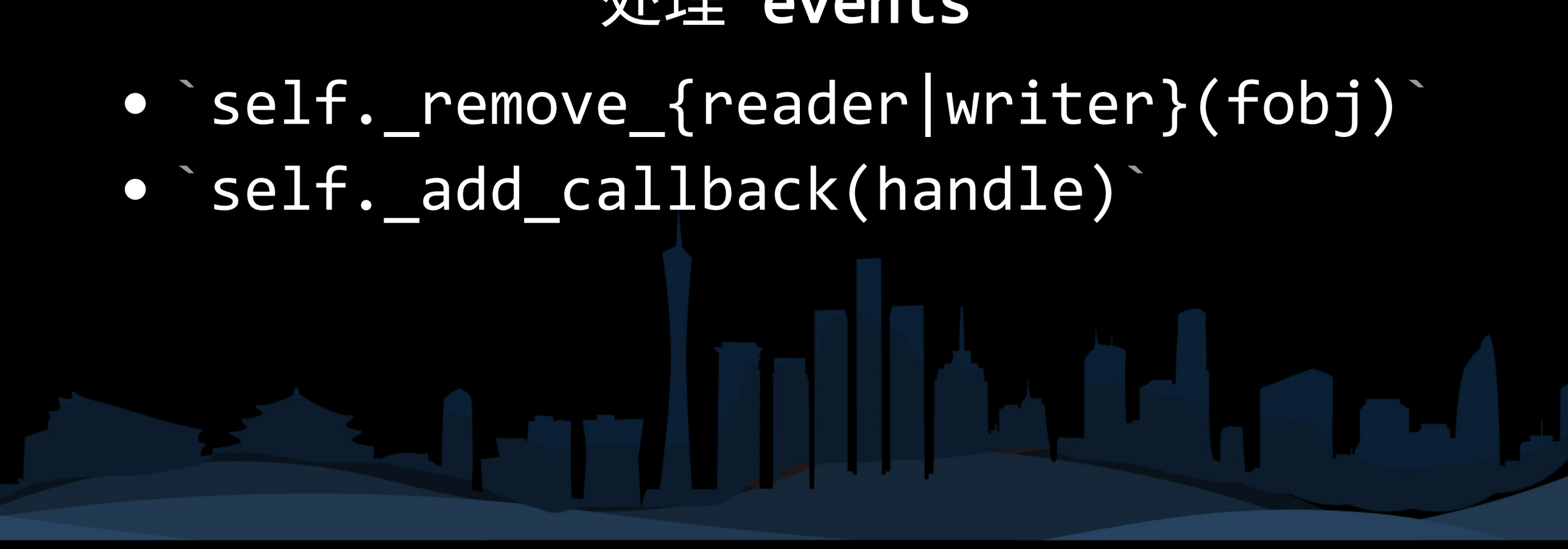
Select!

```
-> event_list =
    self._selector.select(to)
-> self._process_events(event_list)
```



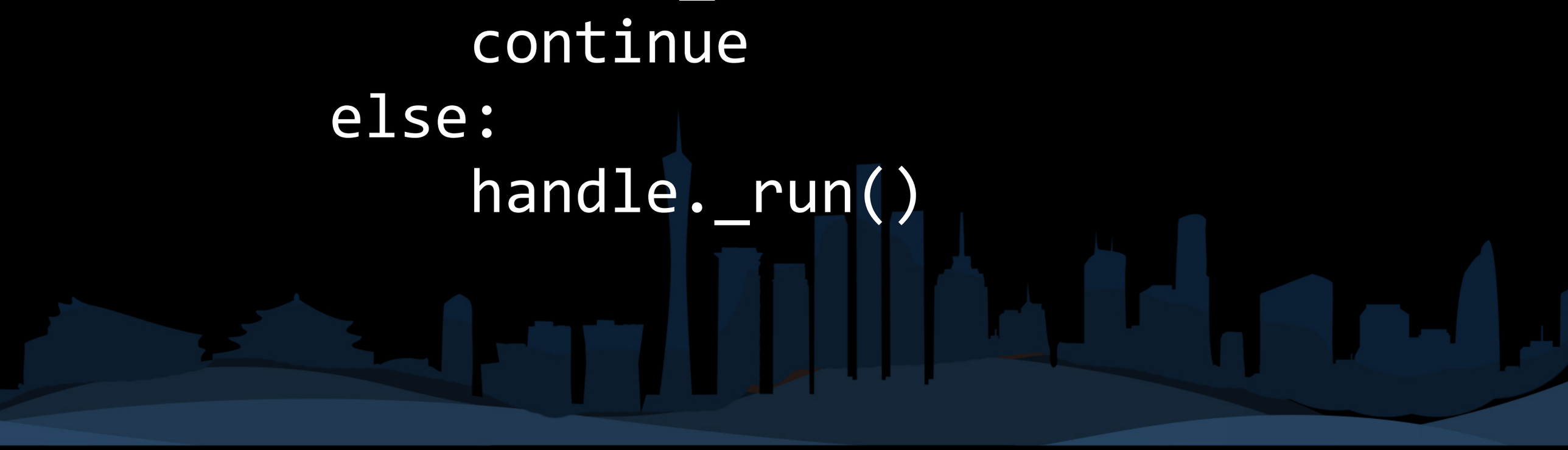
处理 events

- `self._remove_{reader|writer}(fobj)`
- `self._add_callback(handle)`



`self._ready`

```
for i in range(len(self._ready)):
    handle = self._ready.popleft()
    if handle._cancelled:
        continue
    else:
        handle._run()
```



`handle._run()`

```
try:
    self._context.run(self._cb,
                      *self._args)
except Exception:
    # ...
```

```
def __step(self, exc=None):
    try:
        result = self._coro.send(None)
    except StopIteration as e: # ...
    else: # '_asyncio_future_blocking'
    if blocking is not None: # ...
    elif result is None:
        self._loop.call_soon(
            self.__step,
            context=self._context)
```

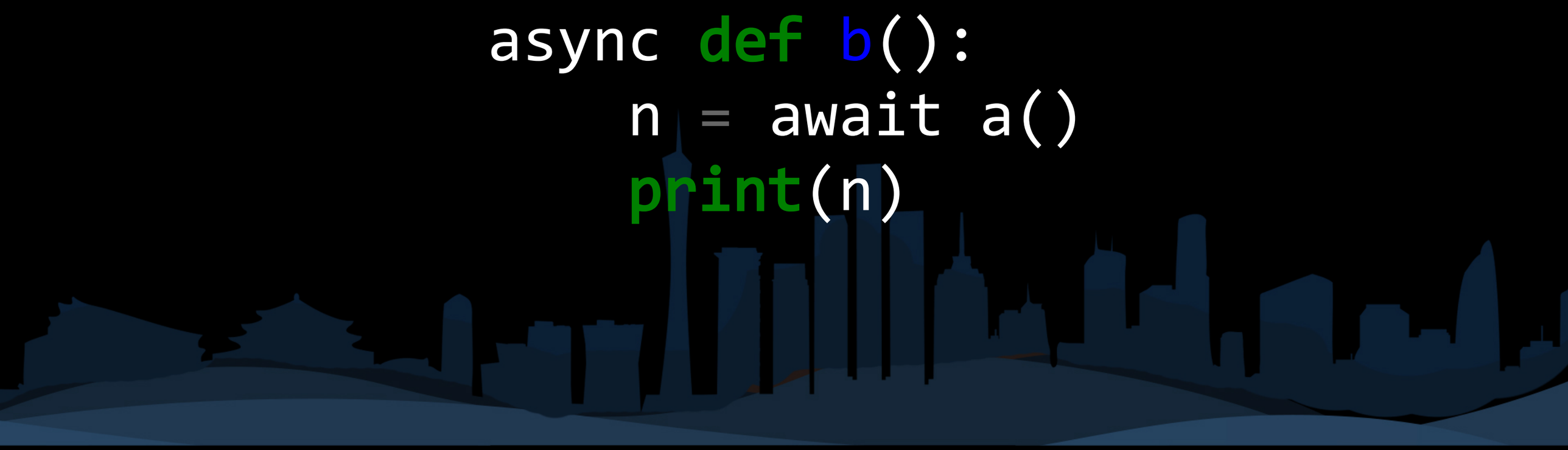
PYCON CHINA 2018

Really?

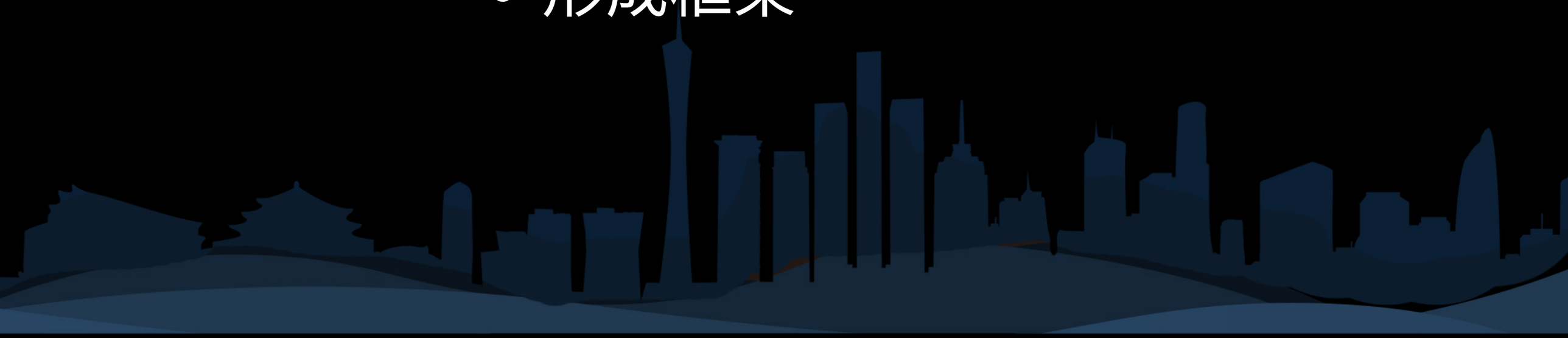
真的只是个生成器吗?

b 输出后, loop 停止了吗?

```
asnyc def a():
    return 42

async def b():
    n = await a()
    print(n)
```



- Task: 事件所有者
- Handle: 调度单位
- 形成框架



其他特性

- Transports
- `call_later`/`self._scheduled`
- `heapq` 优先级
- `self_pipe` 跨线程/进程
- `drain` water mark

PYCON CHINA 2018

```
>>> Anqur.wechat()
anqurmoss
>>> Talk.run_qna()
<Q&A [status: Running]>
>>> quit()
```