A TALE OF CONCURRENCY THROUGH CREATIVITY IN PYTHON:

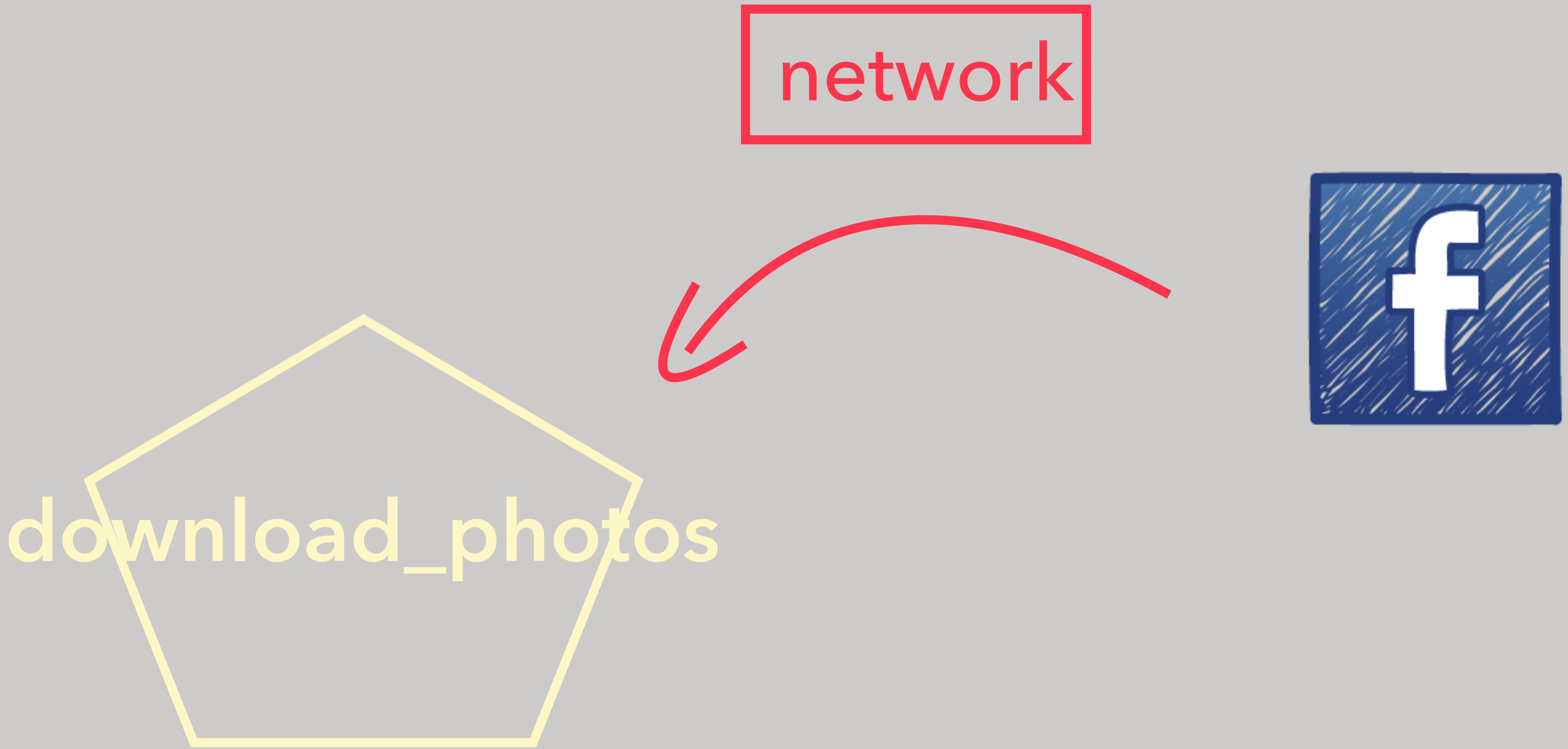
A DEEP DIVE INTO HOW GEVENT WORKS

KAVYA

GEVENT

What is asynchronous I/O?

What is gevent?


network
download_photos

```python
def download_photos(user):
    # Open a connection to the server
    conn = get_authenticated_connection(user)

    # Download all photos
    photos = get_photos(conn)

    # Save for later display
    save_photos(user, photos)
```

```python
def downloader():
    users  = get_users()
    for user in users:
        download_photos(user)
```

network I/O

import

multiprocessing

threading

twisted

green_thread ?

```
import multiprocessing
```

```python
import multiprocessing as mp

def downloader():
    pool = []
    for user in users:
        p = mp.Process(download_photos, user)
        pool.append(p)
        p.start()

    for p in pool:
        p.join()
```

```
import threading
```

```python
import threading

def downloader():
    pool = []
    for user in users:
        t = threading.Thread(download_photos, user)
        pool.append(t)
        t.start()

    for t in pool:
        t.join()
```

```
import twisted
```

```python
import twisted

def download_photos():
    # Modify this to add callbacks

def downloader():
    # Something something loop.run()
```

green threads

- user space –
  the OS does not create or manage them
- cooperatively scheduled –
  the OS does not schedule or preempt them
- lightweight

```
import gevent
import gevent
from gevent import monkey; monkey.patch_all()

def downloader():
    pool = []
    for user in users:
        g = gevent.Greenlet(download_photos,
                            user)
        g.start()
        pool.append(g)
    gevent.joinall(pool)
```

THE BUILDING BLOCKS

PUTTING IT TOGETHER

WRAP-UP/ Q&A

THE BUILDING BLOCKS

```
g = gevent.Greenlet(download_photos, user)
```

```
from greenlet import greenlet
...


class Greenlet(greenlet):
    """
    A light-weight cooperatively-scheduled
    execution unit.

    """

    ...
```

```
from greenlet import greenlet

gr1 = greenlet(print_red)
gr2 = greenlet(print_blue)
gr1.switch()
```

```
def print_red():
    print 'red'
    gr2.switch()
    print ‘red done!’
```

```
def print_blue():
    print 'blue'
    gr1.switch()
    print ‘blue done!’
```

```
red

blue

red done!
```

.switch()

- pause current + yield control flow
- resume next.switch()

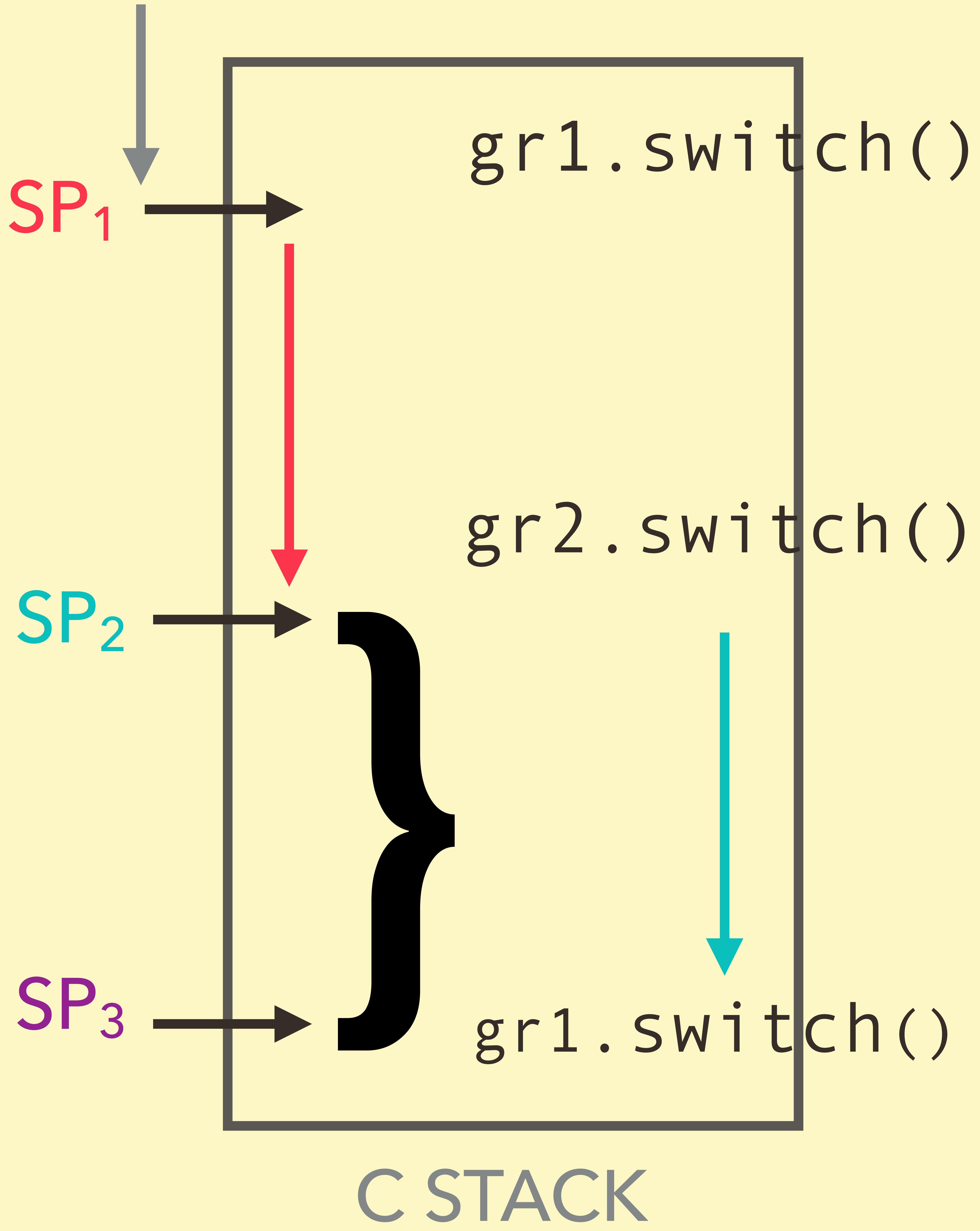
↓

coroutine

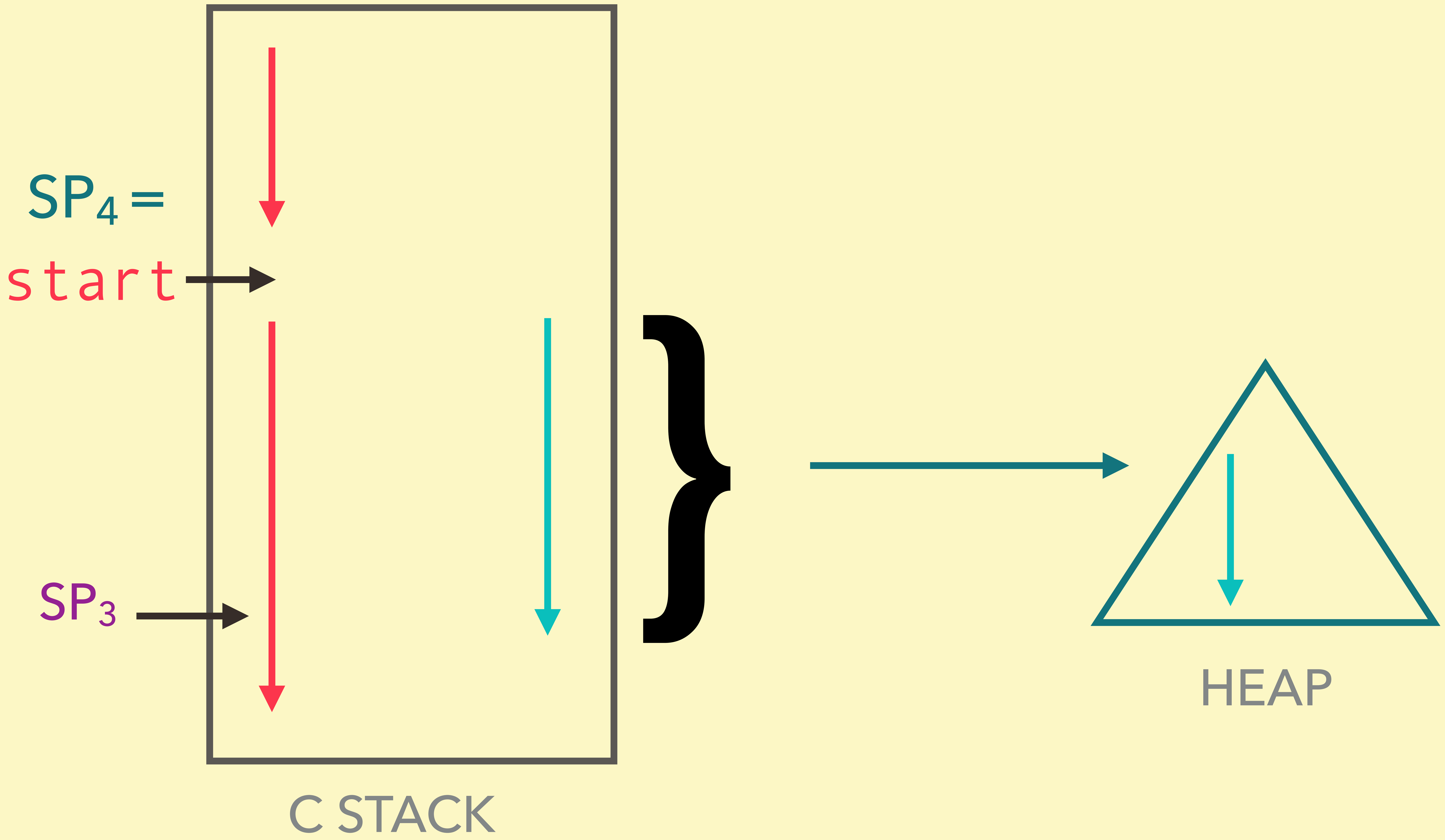
```
gr1 = greenlet(run_fn)
```

$\longrightarrow$

```
{
  run_fn
  parent
  …
}
```



```
{
  base = SP₁
}

{                    {
  base = SP₁           base = SP₂
  start = SP₂        }
}
```


SP_4 =
start
SP_3
C STACK
HEAP

greenlets

for

coroutines

via

assembly-based stack-slicing

```python
import gevent
from gevent import monkey; monkey.patch_all()

def downloader():
    pool = []
    for user in users:
         g = gevent.Greenlet(download_photos,
                             user)
        g.start()
        pool.append(g)
    gevent.joinall(pool)
```

```
g.start()
```

```
def start(self):
    """ Schedule the greenlet to run in this
    loop iteration. """
    if self._start_event is None:
        self._start_event = \
            ...loop.run_callback(self.switch)
```

libev

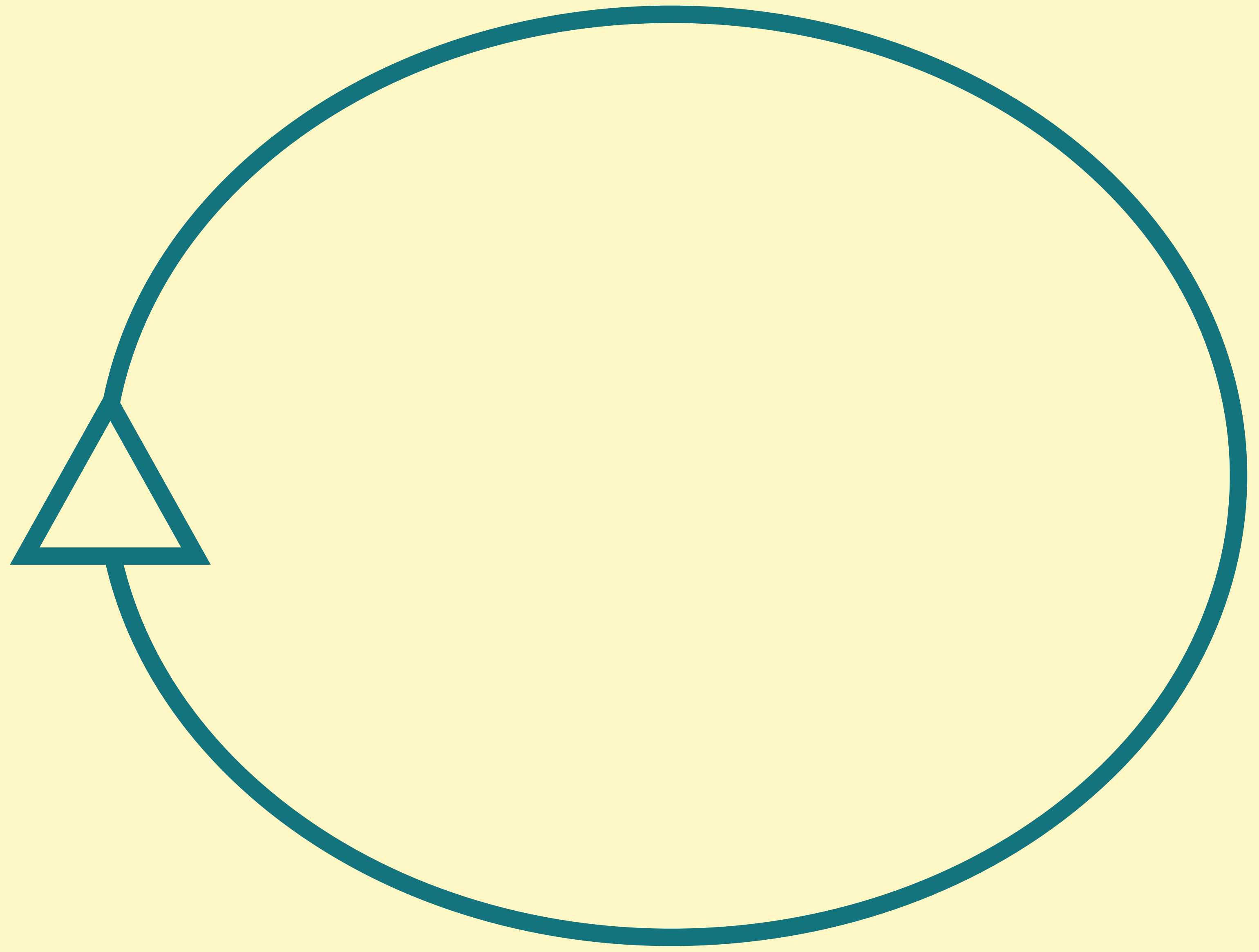
- API to register event_handler callbacks
- watches for events
- calls registered callbacks

“Hey loop,

Wait for a write on this socket and

call parse_recv() when that happens.”

```
fd = make_nonblocking(socket_fd)
loop.io_watch(fd, write, callback_fn)
loop.run()
```



```
while True:
    call all pre_block_watchers
    block for I/O
    call pending io_watchers
    call all post_block_watchers
```

```
always call pre_block_watchers
```

Hook to integrate other event mechanisms into the loop.

$\downarrow$

"Hey loop,
If there are coroutines ready to run,
run them. Then, block for a write on..."

libev

for an

event loop

PUTTING IT TOGETHER

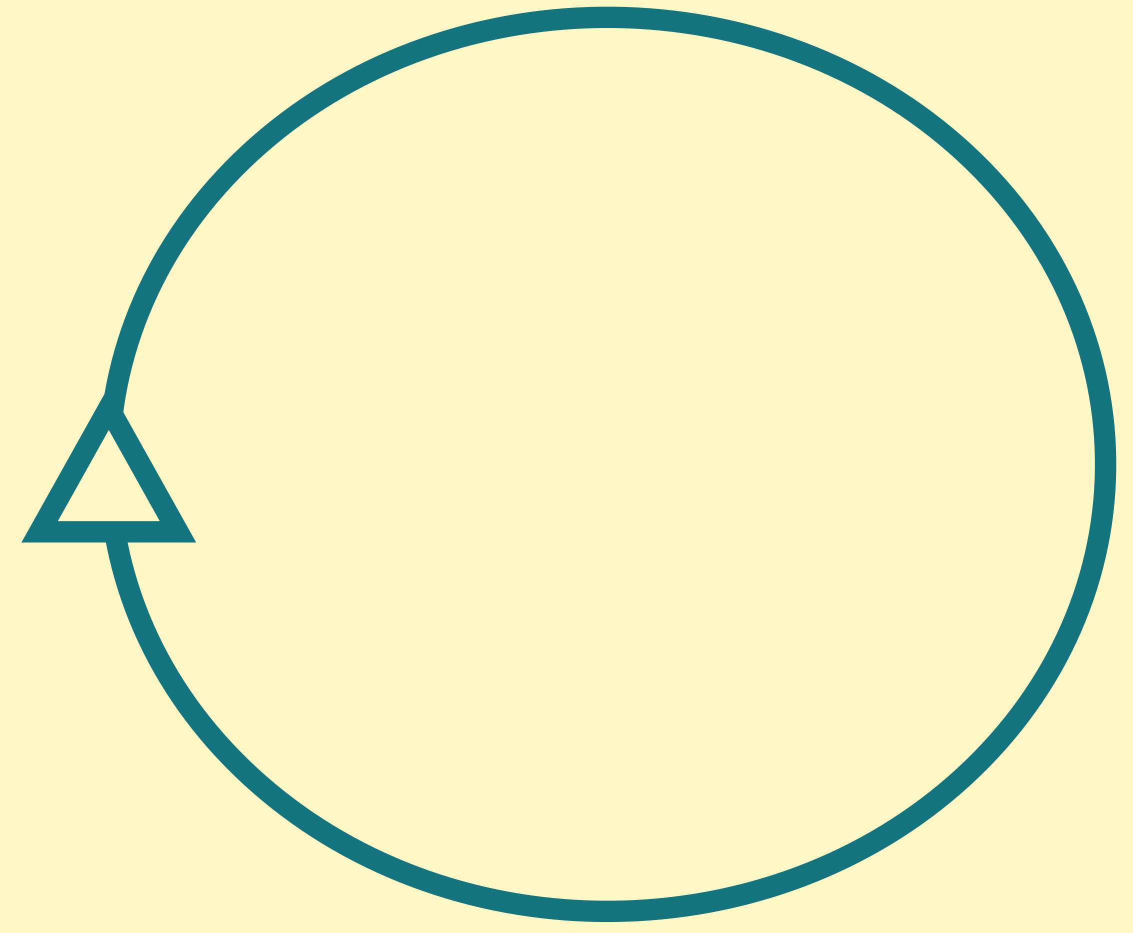
```
import gevent
from gevent import monkey; monkey.patch_all()

def downloader():
    pool = []
    for user in users:
        g = gevent.Greenlet(download_photos,
                            user)
        g.start()
        pool.append(g)
    gevent.joinall(pool)
```

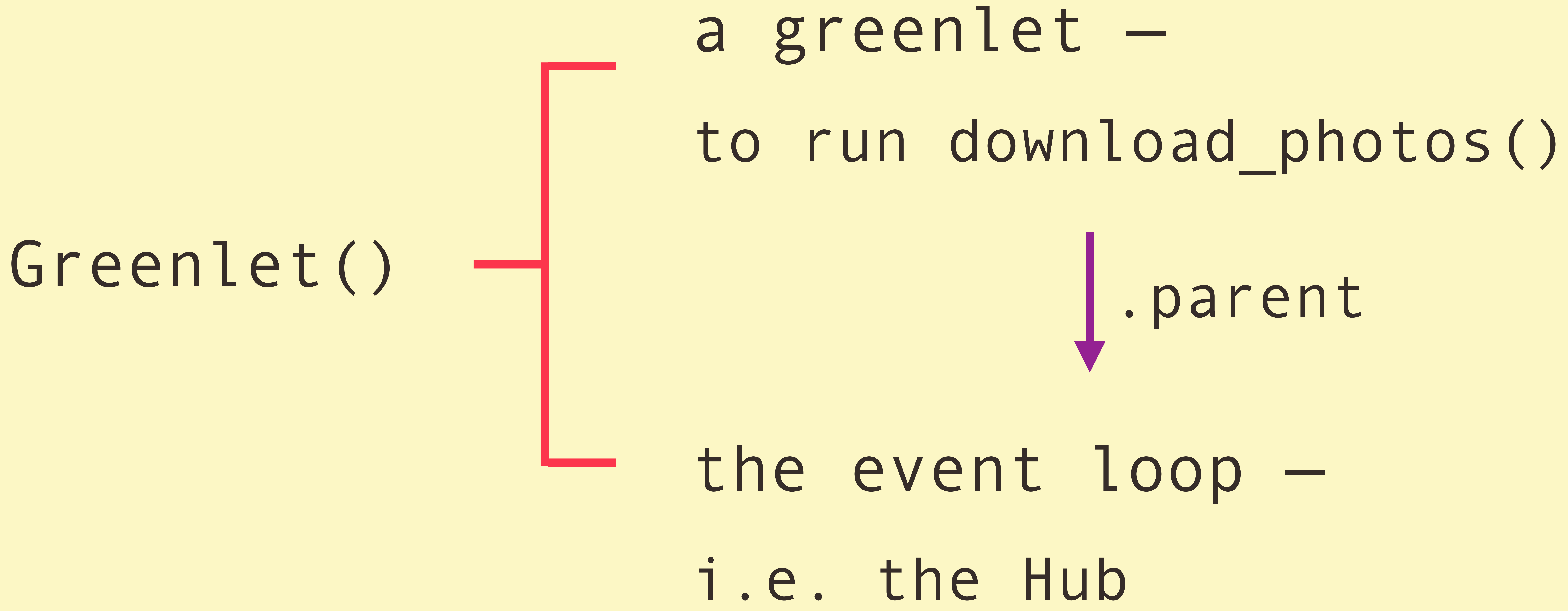
```python
for user in users:
    g = gevent.Greenlet(download_photos,user)
```

```python
g = gevent.Greenlet(download_photos,user)


class Greenlet(greenlet):
    def __init__(self, run=None,...):
        greenlet.__init__(self, None, get_hub())
```

```
g.parent = Hub
```

```
class Greenlet(greenlet):
    greenlet.__init__(self, None, get_hub())

                                  g.parent = Hub

class Hub(greenlet):
    def __init__(self):
        greenlet.__init__(self)
        self.loop = ...
```

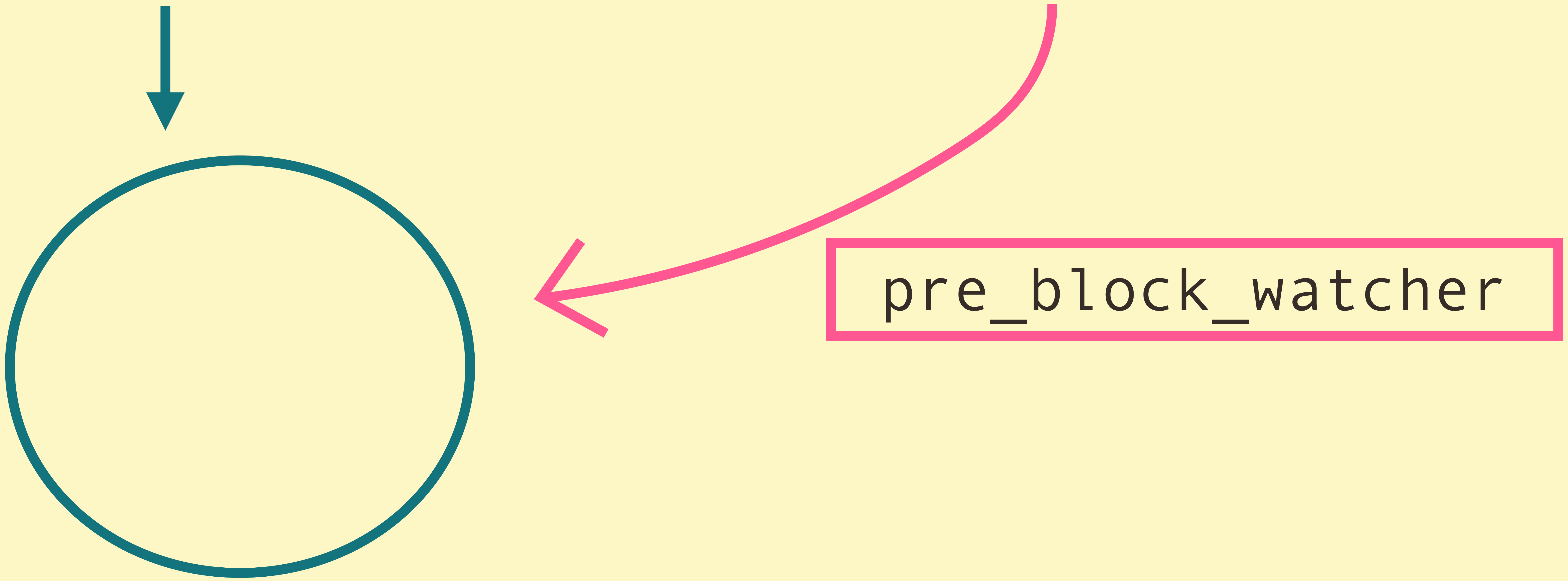



Greenlet()
a greenlet –
to run download_photos()
.parent
the event loop –
i.e. the Hub

```
for user in users:
    g = gevent.Greenlet(download_photos,user)
    g.start()
```

```
g.start()
```

```
    Hub
~~self.parent~~.loop.run_callback(self.switch)
```



```
loop.run()

  while True:

      call all pre_block_watchers  = g.switch

      block for I/O

      ...
```

```
            “Hey loop,
.start() —— This coroutine is ready to run.
            Run it before you block...”
```

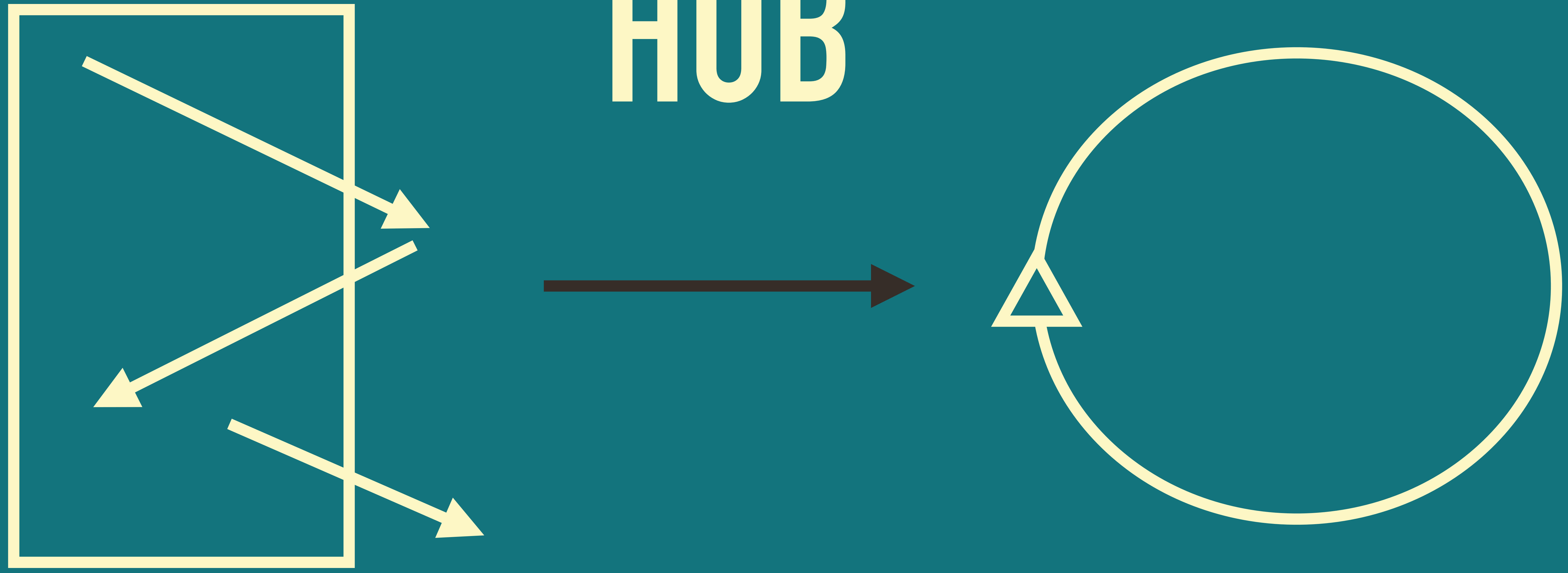
```python
for user in users:
    g = gevent.Greenlet(download_photos,user)
    g.start()
    pool.append(g)
gevent.joinall(pool)
gevent.joinall()
gevent.g.join()
```

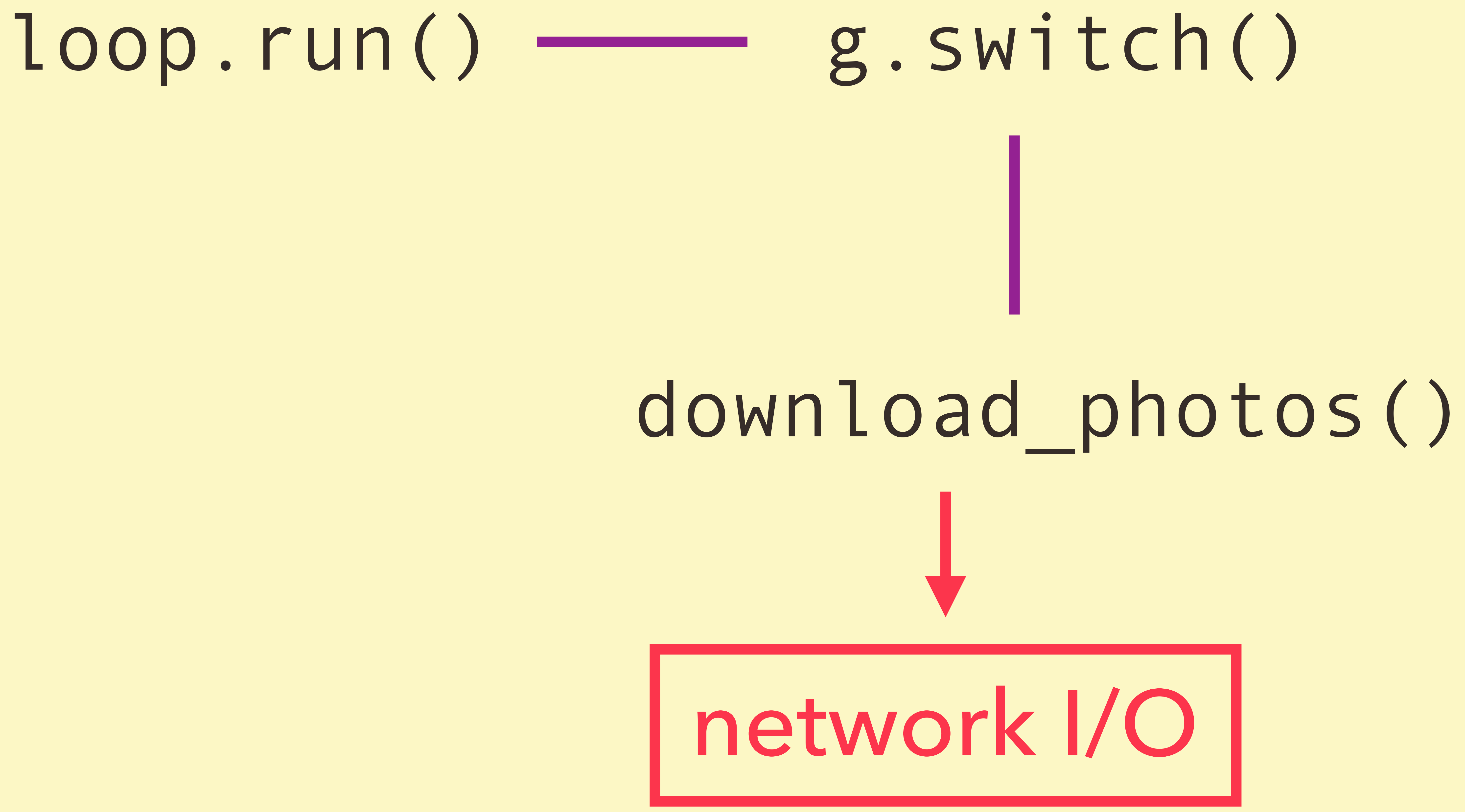
```
                Hub
result = ~~self.parent~~.switch()

class Hub(greenlet):
    def run(self):
        while True:
            self.loop.run()
```

`.join()` — runs the loop

```
.join()
= loop.run()        while True:
                        call pre_block_watchers
                        ...          = g.switch

                               download_photos()
```

HUB




loop.run()
g.switch()
download_photos()
network I/O

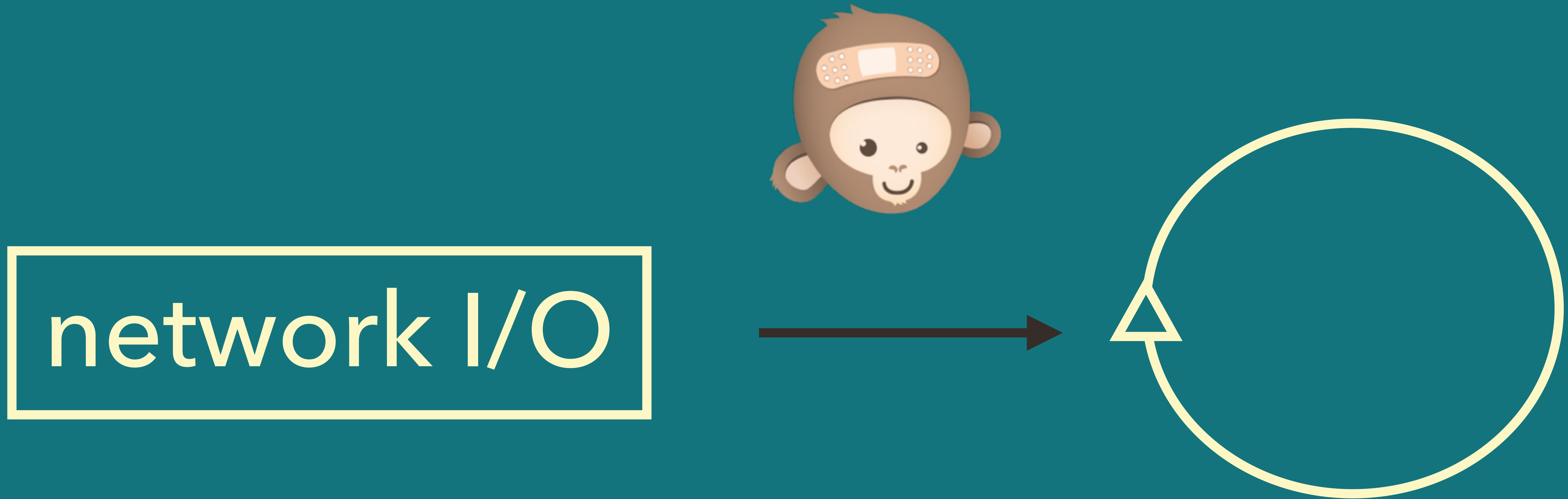
```python
import gevent
from gevent import monkey; monkey.patch_all()

def downloader():
    ...
```

```
       gevent.socket
import ~~socket~~
```

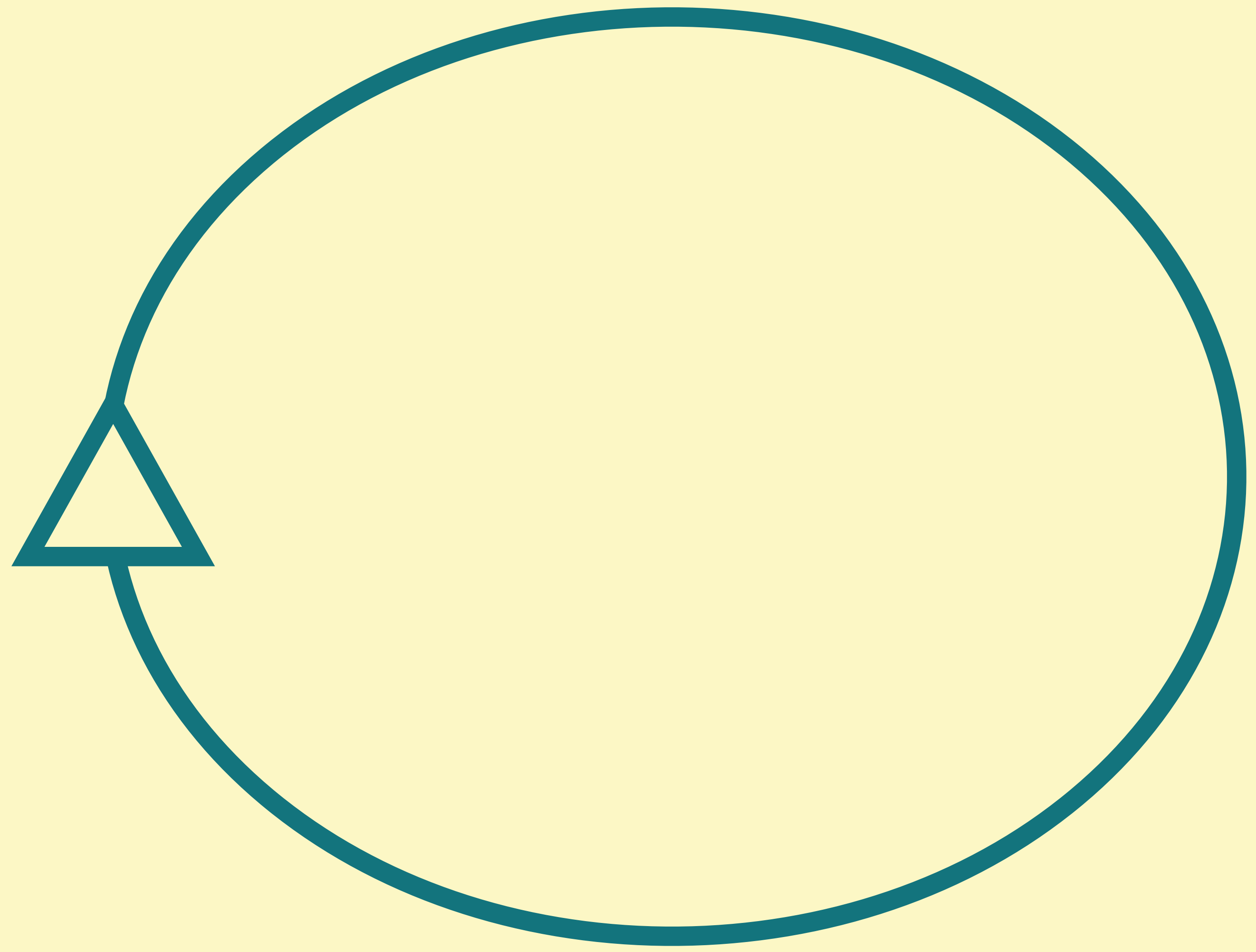
```
create:  fd = make_nonblocking(socket_fd)
send:    loop.io_watch(fd, write, callback_fn)
         loop.run()
```

Hub.switch

g.switch


network I/O

```
for user in users:
    g = gevent.Greenlet(download_photos,user)
    g.start()
```



```
pre_block_watchers = [g1.switch,
                      g2.switch]
```

```
   gevent.joinall()

Hub  loop.run()
       call pre_block_watchers = [g1.switch, ...]
     g1.switch()


g1    download_photos(user1)
         network_request   io_watchers = [g1.switch]
                           Hub.switch()
```

```
Hub  loop.run()
       call pre_block_watchers = [g2.switch]
     g2.switch()

g2   download_photos(user2)
         network_request  io_watchers = [g2.switch,
                                         g1.switch]
                          Hub.switch()
```

```
Hub   loop.run()
        call pre_block_watchers = []
        block for I/O
        call pending io_watchers = [g1.switch]
       g1.switch()

g1     resumes download_photos(user1)

                    ...
```

WRAP-UP

minuses

- no parallelism
- non-cooperative code will block the entire process:
  - C-extensions –> use pure Python libraries
  - compute-bound greenlets –> use gevent.sleep(0)

  –> use greenlet blocking detection
- monkey-patch may have confusing implications
  - order of imports matters

…but

- excellent for workloads that are:
  I/O bound, highly concurrent –> 20-30k concurrent connections!

- Used at "web scale" at:
  Pinterest, Facebook, Mixpanel, PayPal, Disqus, Nylas…

greenlet

libev

Hub

monkeypatching

KAVYA

@KAVYA719

greenlet

libev

Hub

monkeypatching