Building Highly Scalable Web Services with Gevent

Aditya Manthramurthy

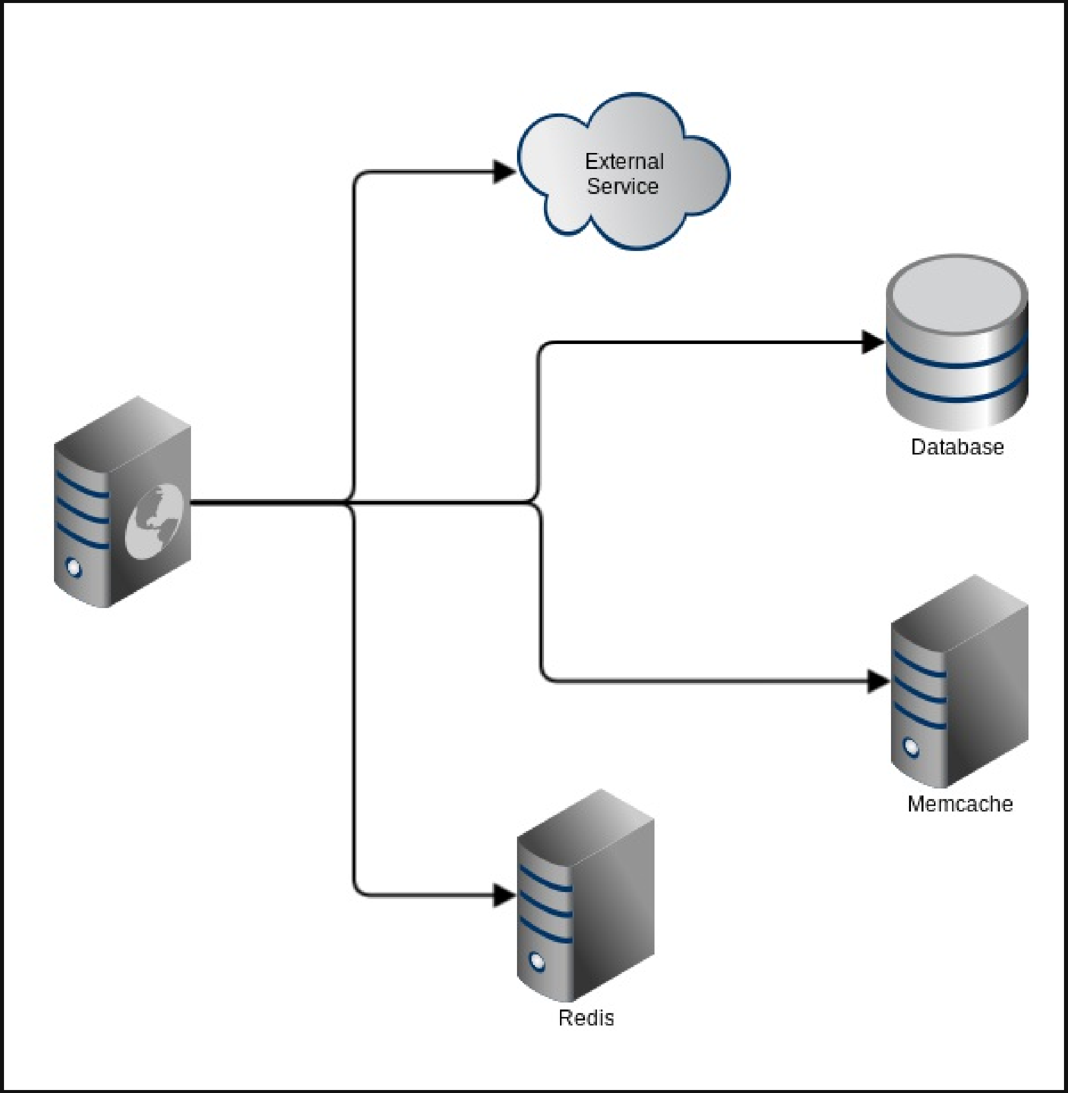
Plivo Inc

donatello imdonatello

Background

- Infrastructure Engineer at Plivo
- Programmed in Python for six years
- Gevent used everywhere in Plivo

Blocking IO is terrible!



Scaling Challenges in Python Webservers

- Flask/Django on Gunicorn - multiple procs, each doing a request
- Each request - many RPCs, process blocked on I/O
- Max requests handled at a time = No. of webserver procs
- Try to run with more procs --- unpredictable server load!!

Non-blocking I/O

gevent

- Existing code becomes non-blocking on IO ops!
- Standard libs are monkey patched.
- On an IO op. thread switching happens

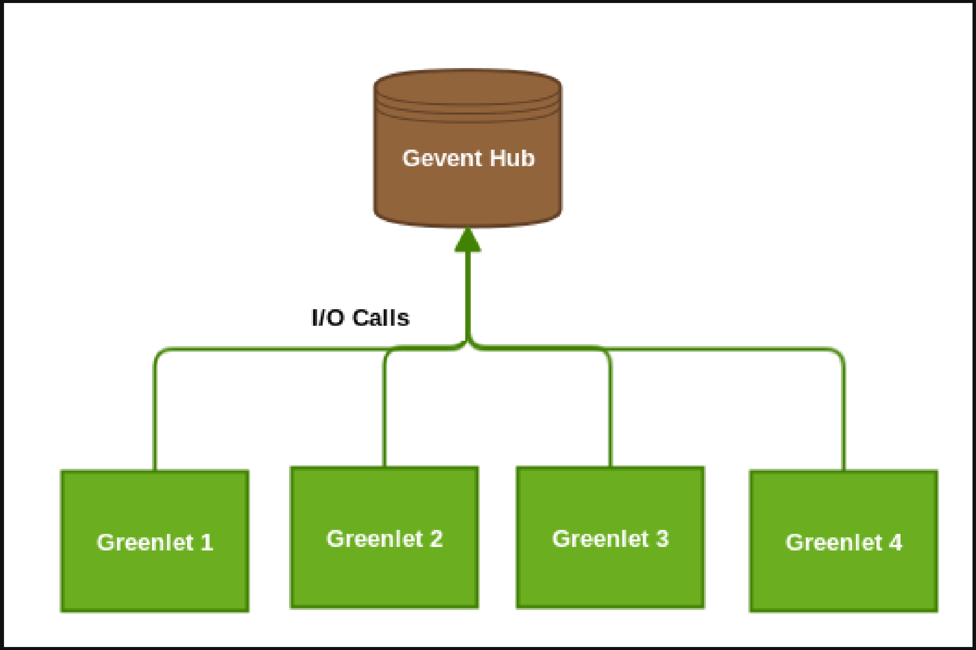
Twisted or Others

- Need to rewrite code in a callbacks based model.
- Or litter code with lots of uninteresting async annotations
- Think about pure CPU / pure IO components?!
- The machinery can do that too!

What is Gevent?

Gevent is a ***co-routine*** based networking library that uses light-weight execution units, called ***greenlets***, to provide a high-level synchronous API on top of a fast event loop provided by ***libev***.

- Co-operative threading model
- Monkey patching of standard libs
- Thread switching on I/O or explicit yield



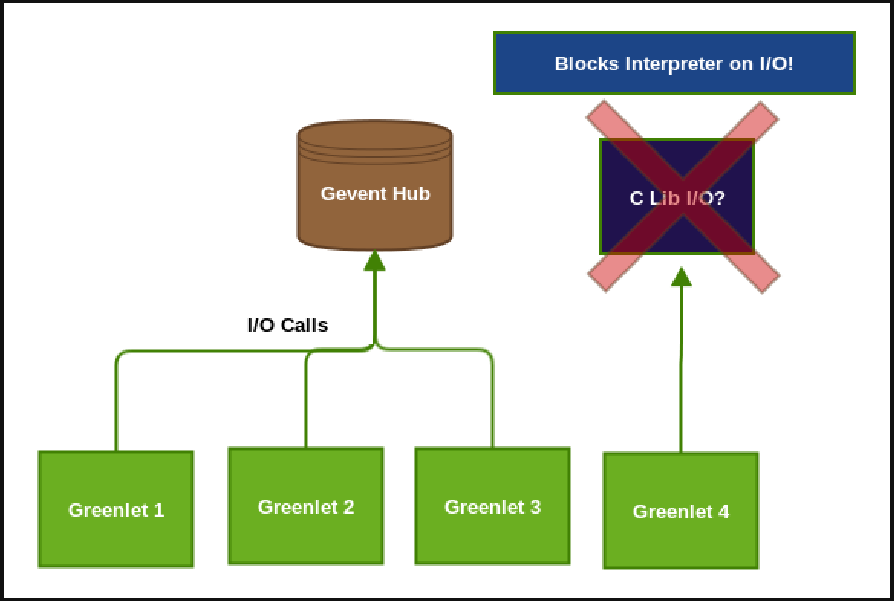
Gevent with Flask/Django?

- Gevent wsgi server - gevent.wsgi
- **Better**: uwsgi/gunicorn with gevent worker type
- Don't forget to ***monkey-patch***!

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

import time
from flask import Flask

app = Flask(__name__)

@app.route('/hello/')
def hello():
    time.sleep(1) # yeah work being done here!
    return "Hello World!"
```

Will Gevent work for my app?

- Is your app CPU bound or IO bound?
- Non-std. libs doing IO (e.g. DB driver) - can they be greened?
- Or can you work around non-std libs doing IO?



DB Driver - How To

Python **DB drivers** usually link to C libraries and do IO in the C layer - they are not ***greenlet-aware*** by default.

But most of them have green alternatives!

```
# EXAMPLE: Green the PostgreSQL driver psycopg2

# monkey patching
from gevent.monkey import patch_all
patch_all()
# patch the DB driver too!
from psycogreen.gevent import patch_psycopg
patch_psycopg()
```

Working with certain C libs

Help on function parse in module lxml.html:

parse(filename_or_url, parser=None, base_url=None, **kw)
    Parse a filename, URL, or file-like object into an HTML document
    tree. ....

```
import lxml.html

def myparse(url):
    return lxml.html.parse(url)
```

Do the IO bit out-of-band!

```
from gevent.monkey import patch_all
patch_all()
import urllib
import lxml.html

def myparse(url):
    page = urllib.urlopen(url)
    return lxml.html.parse(page)
```

Avoid Starvation

The one-off high CPU function?

```python
def my_view_func(request):
    db_results = get_results(request)

    final_result = []
    for result in db_results:
        final_result.append(
            my_expensive_func(result)
        )

    return final_result
```

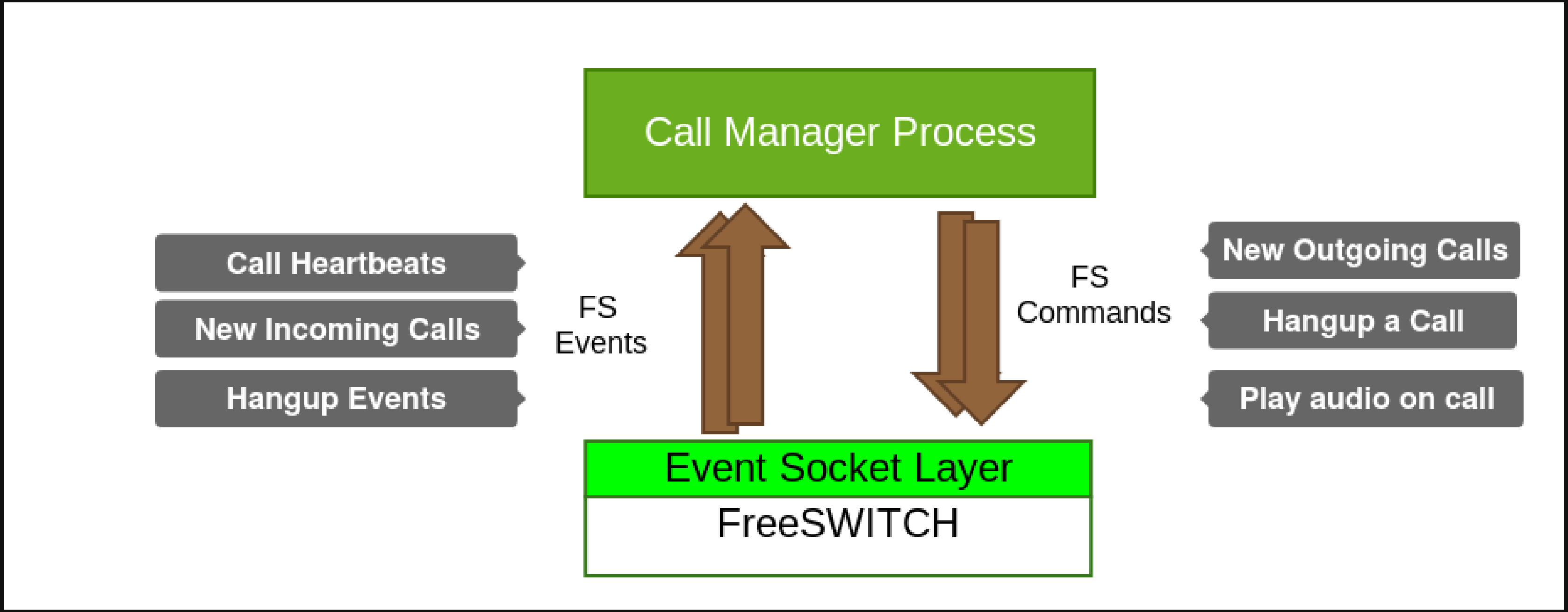
Give other greenlets a chance!

```python
import gevent

def my_view_func(request):
    db_results = get_results(request)

    final_result = []
    for result in db_results:
        final_result.append(
            my_expensive_func(result)
        )
        gevent.sleep(0)

    return final_result
```

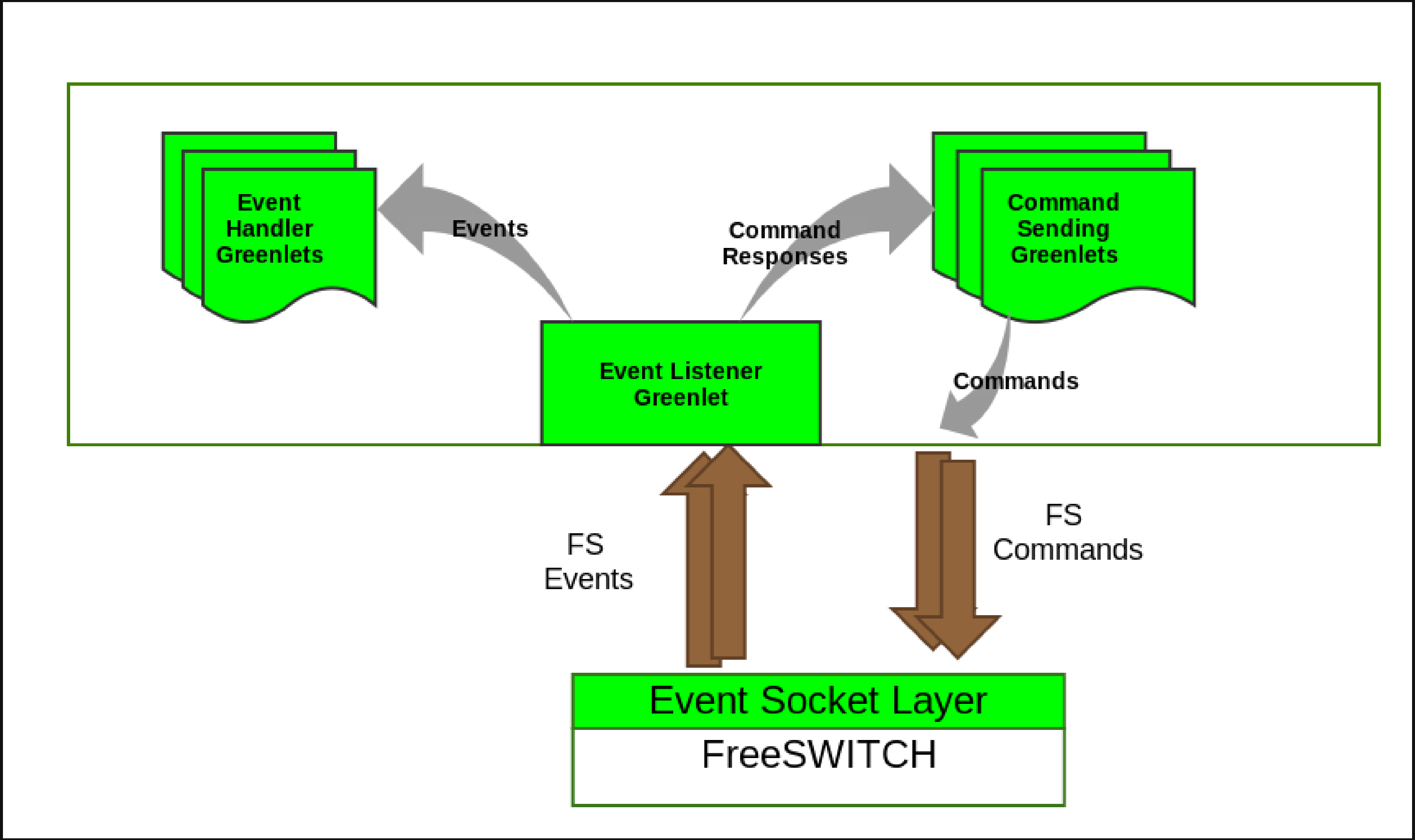
What else with Gevent?

- Greenlets are cheap - start 100s or 1000s of them - no problem!
- Gevent has some standard synchronisation primitives: Events, Queues, and Locks
- Multi-producer-consumer queues are easy to do!
- Let's look at a Events and Locking example

Locks and AsyncResult - Plivo Use Case



Thread Model - FS ESL



Sending Commands to FS

```
COMMANDS = collections.deque()
LOCK = gevent.lock.RLock()

def commander(t, cid, cmd):
    async_res = gevent.event.AsyncResult()
    with LOCK:
        COMMANDS.append((cid, async_res))
        send_command(t, cid, cmd)
    # command sent to ESL - now block until response is got.
    _cid, resp = async_res.get()
    if cid != _cid:
        raise Exception('Commands out of sync!')
    return resp
```

The **lock** ensures that commands and their responses stay in ***sync***

Handling FS Events and Responses

```
def handle_responses(t):
    while True:
        resp = read_response(t)
        if resp.startswith('Event:'):
            gevent.spawn(handle_event, resp)
        else:
            # we got a command response
            cid, async_res = COMMANDS.popleft()
            # wake up waiting command
            async_res.set((cid, resp))
```

See a full runnable example at -
https://github.com/donatello/pycon2014/

Gevent Caveats

- Breaks profiling tools like cProfile - alternatives are of varying quality; YMMV with NewRelic, etc
- C libs doing IO - need green alternatives/workarounds
- Need to be careful with CPU intensive operations - other greenlets will be starved

Benchmarks?

Comprehensive one at -

http://nichol.as/benchmark-of-python-web-servers

Thank you!

Non-Blocking IO Demo

```
@app.route('/sleep/python/')
def sleep_python():
    """ This handler sleeps for 5s. Does it block?
    """
    time.sleep(5)
    return Account.jsonify_all()
```

Demo Code: https://github.com/donatello/pycon2014

Non-Blocking IO Demo

```
@app.route('/sleep/postgres/')
def sleep_postgres():
    """ This handler asks Postgres to sleep. Does it block?
    """
    db.session.execute('SELECT pg_sleep(5)')
    return Account.jsonify_all()
```

Demo Code: https://github.com/donatello/pycon2014