MORE THAN A WSGI SERVER
WHAT IS GUNICORN
- PEP 3333
- pre-fork worker model
- Python > 2.6x
- handle Django & Paste configuration
- support any concurrency framework

A WSGI SERVER
ALREADY_HANDLED: take control of the socket

'gunicorn.socket' environ

'gunicorn.raw_uri' environ

`transfer-encoding: chunked`
PREFORK WORKER MODEL

- listen
- handle signals
- supervise

WORKER
- accept
- WSGI server
- handle signals

share socket

ARBITER
SHARE NOTHING

- share only file descriptors
- Privilege separation
- Worker: simple process with a run function
- sockets
- configuration
- log
- parent pid
- tempfile
- setup signals

```python
class MyWorker(WorkerBase):
    def __init__(..):
        ...

    def init_process(self):
        ...

    def run(self):
        ...
```
run the application
need to notify the arbiter
tempfile updates
handle application errors
but should crash on errors most of the time

LET IT CRASH
class MyApp(BaseApplication):
    def load_config(self):
        ...
    def load(self):
        ...
        return self.application
DO MORE >
▸ improve listening in the sync worker
▸ remove the tempfile
▸ prepare for the new protocols
▸ improve logging
▸ going further

DO MORE. BE SAFER.
place the listening sockets in an event loop

on read, start to accept

also have a timer

ACCEPT WHEN NEEDED
IMPORT GUNCORN.IMSG
- communicate between processes
- use UNIX socket
- sendmsg/recvmsg + buffer
- 2 methods: send, receive
- zero copy memory
fds = socket.socketpair()

pid = os.fork()

if pid != 0:
    # parent
    socket.close(fds[1])
    imsg = IMSG(fds[0])
    return main(imsg)

# child
socket.close(fds[0])
var imsg = IMSG(fds[1])
return main(imsg)
main(imsg):

    imsg.send("sometype", "somemsg"),
    dispatch_msg(imsg)

dispatch_msg(imsg):

    while True:
        msg = imsg.recv()
        if msg is None:
            sleep(1)
            continue
        print(msg.data)
- A socket pair is opened for each worker.
- An `imsg` object is created for each.
- `imsg` objects are placed in an event loop to read asynchronously.
- Maybe extracted from the code as a simple library.
- rework ALREADY_HANDLED
- socket control is given to app
- but still supervised
- simple API
def app(environ, start_response):
    ...
    start_response(status, headers)
    proc = MyProc(environ)
    return proc

class MyProc(gunicorn.Proc):
    def status(self):
        return STATUS_ALIVE
    
    def terminate(self, Reason):
        ...

• simple proc abstraction

• return a proc instance

EXAMPLE
- logging will now be handled in its own process
- log infos passed via imsg and dispatched by the arbiter
- fix privilege separation issues

IMPROVE LOGGING
GOING FURTHER

- HTTP 2
- static file server?
- fastcgi?
Gunicorn

HTTP://GUNICORN.ORG

Enki Multimedia

HTTP://ENKIM.EU