Python網路程式設計

IGS / ACD_RD7
duyhsieh
duyhsieh@igs.com.tw

source file available at:
https://github.com/mantisa1980/pythontutorial

Agenda

- Python
  - 基礎觀念
  - Concurrency & Parallelism
  - pip套件管理
- HTTP
  - 基礎觀念
  - JSON
  - WSGI
- Python Web Server library
  - gunicorn
  - gevent coroutine
  - falcon

Part I: Python基礎

Python Basis

- 偏向腳本語言(Scripting language)
  - 以interpreter執行pyc (bytecode)的指令碼
  - 對照: Compiled language是編譯成native machine code，速度較快
- 強制縮排: 4 space (recommended) / tab (等同C大括號)
- Garbage collection
- 跨平台 (但某些三方套件對Windows不太友善: linux is better)
- 官方標準直譯器: CPython
  - 其他: PyPy (JIT), IronPython (for .NET), Stackless Python…
- 整合IDE: PyCharm

Python Basis (Cont.)

- 2.7.x v.s. 3.x
  - 2.x預計只patch到2020，新功能會出在3.x
  - 語法差異上不大: print 123 v.s. print(123)，對於unicode/byte處理上有些差異
  - Linux系統預設安裝的多為2.x版本（舊系統依賴問題）
  - 新專案可建議直接使用Python3，但不要混用 (ex. pymongo 2.x / 3.x語法不相容)

Python Hello World

- 練習: helloworld.py

- python helloworld.py

- Try to unmark first line

- if __name__ == "__main__":
  …被當輸入腳本時才執行，import不會

Interactive Interpreter

- python
- 可以快速拿來Try語法或測試import路徑，平常用不太到
- 離開: quit()

Package / Module

- 在python的術語中...
  - package: 包含多個modules的資料夾
    - package目錄下需要有__init__.py
      - __init__.py的all=[module1, module2...] 屬性:
        - 指定from package import * 時要連帶import的modules
  - module: single .py file
    - module裡面有function, class...(module attribute)

Import

- 各種可能的import法
  - import lib.foo
    lib.foo.bar()
  - from lib import foo
    foo.bar()
  - **from lib.foo import bar**
    bar() —> 效能最好 (較少 dot reference)

- **project**

```
        |
        —main.py
                |
                lib—foo.py: def bar(): …
                   —__init__.py
```

Import Paths

- Ubuntu 14.04
  - 1. Input script (被執行的腳本) 目錄
  - 2. PYTHONPATH環境變數
    - 會被置入sys.path中; 若無設定預設為input script 目錄
  - 3. /usr/local/lib/python2.7/dist-packages/
    - 3rd-party library default installation path
    - 如果自己make install python，則會變成/usr/local/lib/python2.7/site-packages
- **練習: import_test.py (fix import error)**

Variable

- 變數: 動態型別，指向物件的一個參考(name reference)
- 物件類型
  - Mutable object: list , dictionary…
    - 可透過參考直接修改物件本身
  - Immutable object: string, number, tuple…
    - 無法透過參考修改物件，只能改指到別的地方
- 變數傳入Function只是複製了另一份Variable，指向一樣的地方
- **練習:reference.py**

Reference

Variable

Object

A = 3

A

3

B = 3

B

C = [1,2,3]

C = D

C

[1,2,3]

D

Reference

Variable

Object

A = 3

A

3

B = 4

B

4

C

[1,2,3,4]

D.append(4)

連帶影響C

D

常用資料結構: Dictionary (Hash table)

- dic = {'name':'John', 'coin': 100 } #或用dic = dict()
  dic['age'] = 30
  dic.pop('name')

- 用來快速查找key / value

- 無法保證key的順序性

- dic.keys(), dic.values(): 回傳keys / values

- 尋訪元素

- for k in dic.keys():
      dic[k] = …

- for k,v in dic.items():
      ….

常用資料結構 (Cont.)

- list: dynamic array，類似C++ vector的東西
  - Direct indexing很快，但搜尋複雜度=O(n)
  - A = [1, 2, 'xyz']
- tuple # immutable
  - a= (2,)　# 注意a=(2)會解讀成2 (整數)
  - a = (2,3) # ok
    a = (2,3,) # ok
    a[0] # 2
- **練習:data_structure.py**

Exception Handling

- import traceback
  try:
  raise Exception("Error message")
  except Exception as e:
  print e # print error message
  print traceback.format_exc() # print call stack

Class

- class Foo(object): # new style class in python
    def __init__(self, name):
        self.name = name
        self.__pdata = 0 # private

    def hello(self):
        print self.name

    @staticmethod
    def bar(param1):
        print param1

- x = Foo('Alice')
  print x.name, x.hello()
  Foo.bar(1234)

GIL

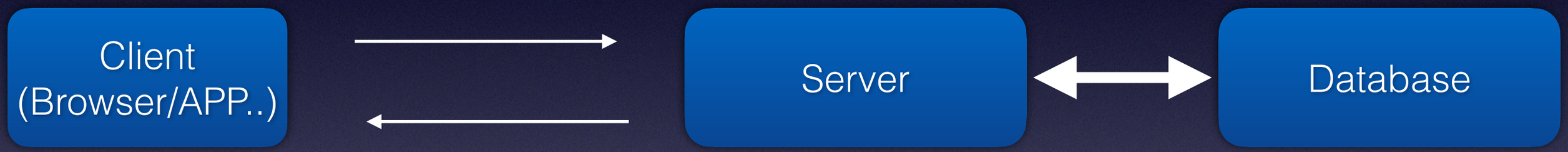
- Python 全域鎖(GIL: Global Interpreter Lock)
  - one active thread per python process
    - No parallelism (even multi-threaded)

pip套件管理

- PyPI (Python Package Index)
    - Python的公開第三方套件庫
    - https://pypi.python.org/pypi
- 安裝第三方套件:
    - pip install package_name
    - 安裝特定版本: pip install pymongo==2.8
- 反安裝:
    - pip uninstall package_name
- 查看安裝套件:
    - pip list

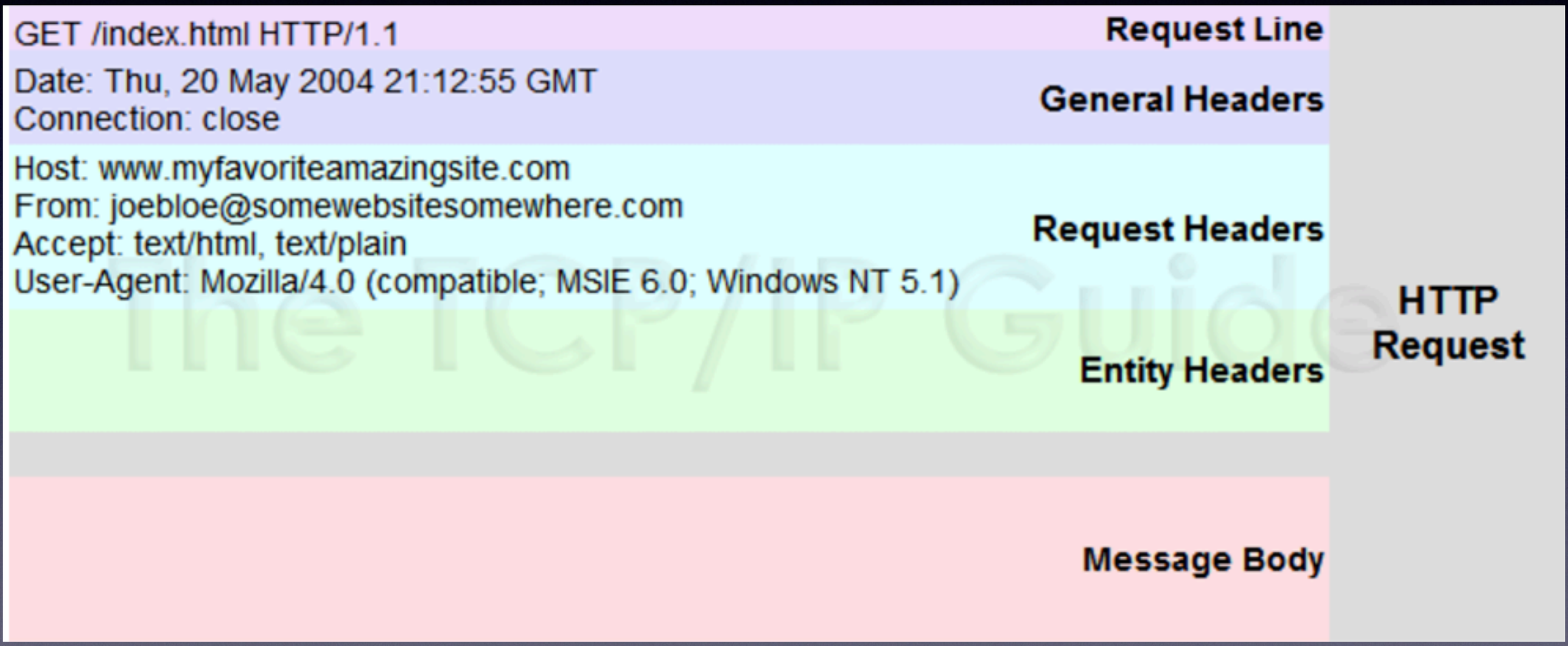
Part II: 基礎HTTP觀念

Common HTTP Architecture



Over TCP/IP
Connection closed after request
Use Keep-alive header to keep connections open

HTTP Request Format



HTTP Response Format

HTTP/1.1 200 OK	**Status Line**
Date: Thu, 20 May 2004 21:12:58 GMT Connection: close	**General Headers**
Server: Apache/1.3.27 Accept-Ranges: bytes	**Response Headers**
Content-Type: text/html Content-Length: 170 Last-Modified: Tue, 18 May 2004 10:14:49 GMT	**Entity Headers**
<html> <head> <title>Welcome to the Amazing Site!</title> </head> <body> This site is under construction. Please come back later. Sorry! </body> </html>	**Message Body**

HTTP Response

HTTP Methods

- HEAD
    - 只取得資源的metadata, 不取得資源本文
- GET
    - 讀取資源.
    - 參數夾帶在url之後. ex. http://serverurl/api?param1=abc&param2=efg
        - 不要用來修改資料; 可能會有爬蟲程式來呼叫
        - 瀏覽器可以cache server response
- POST
    - 修改資源
    - 除非設定特殊Header否則一般是不cache server response
    - 參數夾帶在Message body
- Others
    - PUT, DELETE, TRACE, CONNECT, PATCH

JSON (JavaScript Object Notation)

- key-value pairs
- Values can be: Object: {} or Array: [], or primitive data type (integer, string ...)
  - **{**
  - **"str_key":"bbb",**
  - **"int_key":1,**
  - **"array_key":[**
  - **{"some_key":123 },**
  - **{"some_key":456 }**
  - **],**
  - **"sub_doc_key":{**
  - **"mmm":"nnn",**
  - **"xxx":"yyy"**
  - **}**
  - **}**
- use json.loads(json_string) to dict_object, json.dumps(dict_object) to json_string

WSGI (Web Server Gateway Interface)

- 規範Python web server的request handler格式
- 執行環境繼承了傳統CGI變數，以及新增自定義的變數
  - 傳統CGI環境變數: REQUEST_METHOD, QUERY_STRING
  - 自定義變數: wsgi.version, wsgi.url_scheme…

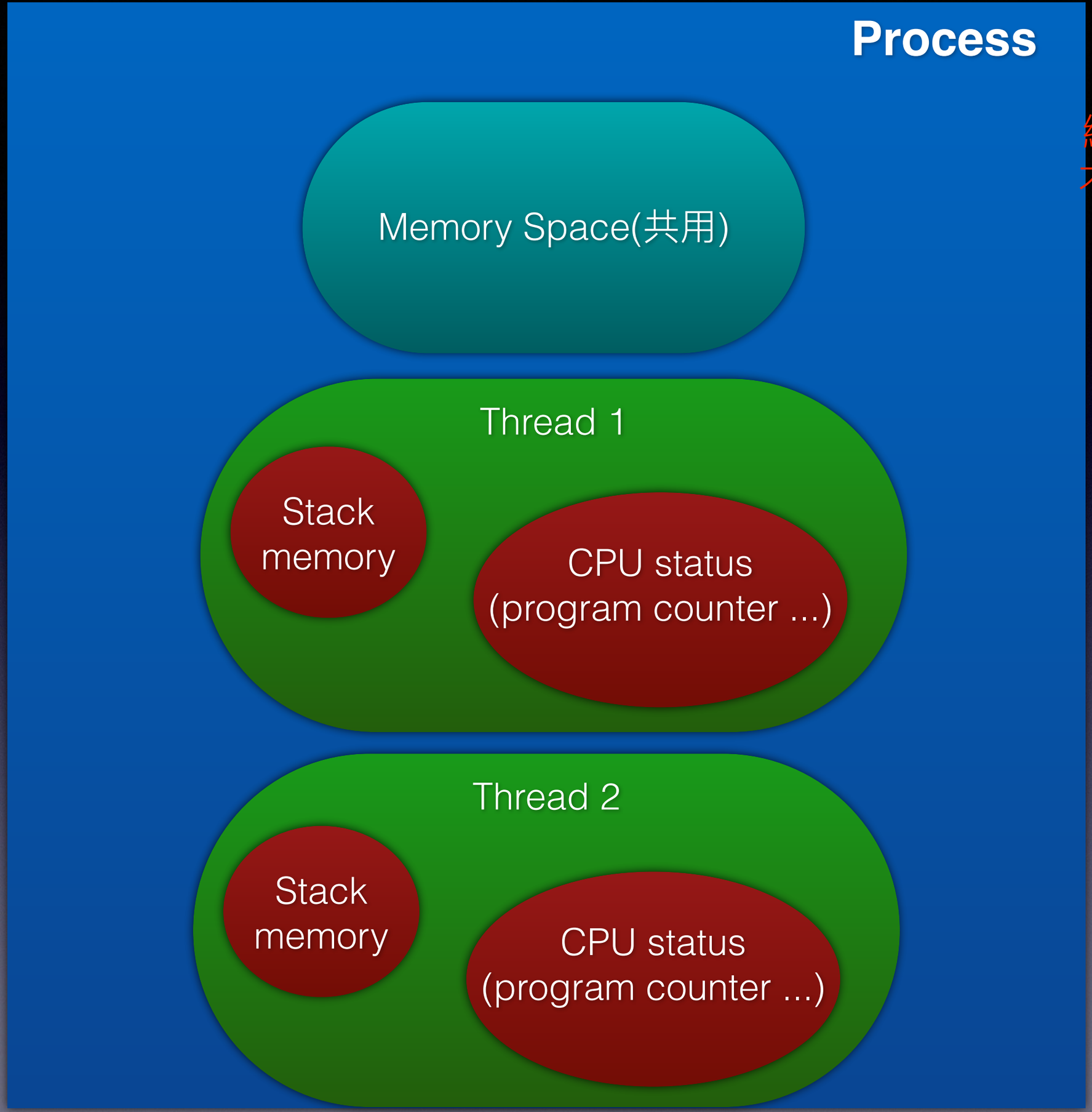
```
def simple_app(environ, start_response):
    status = '200 OK'
    response_headers = [('Content-type','text/plain')]
    start_response(status, response_headers)
    return ['Hello world!n’]
```

- **練習: wsgi.py**

Part III: Python Web Server函式庫

作業系統科普

- program(程式/執行檔):包含一堆電腦指令的集合
- process(程序): program執行後產生的一個執行個體
- Thread(執行緒): 執行process指令的主要角色
  - 每個process至少有一thread (main thread)



Program counter(PC):
紀錄執行到哪一行程式，
不同thread有自己的PC跟
Stack memory
(ex. local variable)

Parallelism & Concurrency

- Parallelism: 硬體在”單一時間點”多核同時執行: 描述的是物理意義上的平行運算
- Concurrency: 一段時間內交叉執行多項任務
  - 任務A執行到一半時，可以切換到任務B執行，不會讓CPU閒置而效能低落
    - 一般討論web server concurrency會限於單一應用程式(process)內，和OS的multi-tasking區隔(同時執行多個process)

Parallelism & Concurrency in Python

- 由於GIL的限制，即使開多執行緒，單一process內同一時間也只有一個執行緒可以運作

- 若有多核心，如何做到Parallelism?
  - Multi-processing
- 對於每個單一process，又如何做到concurrent？
  - Multi-threading: A thread等待IO時可以換B thread執行
  - Single thread + asynchronous library (see next page)

Web development stack

- gunicorn + gevent + falcon
  - falcon: web API development framework
  - gevent: asynchronous coroutine library
    - concurrency
  - gunicorn: web server binary
    - parallelism

gunicorn

- Python本身或部分python web framework 內建的web server不適合用來正式環境 (註解有寫)
  - 安全性不佳
  - 不能處理GIL對於多核心執行的限制
- gunicorn:
  - ported from Ruby’s Unicorn project
  - 1 master process + N worker processes (pre-fork model)
    - 用multi process處理python GIL無法利用多核心的問題

run gunicorn server

- 練習測試
  - gunicorn -w 5 guni:app
    - app: a WSGI compatible handler
- test workers
  - curl localhost:8000 some times to check worker pids

Synchronous server

- 一個request完全處理完才處理下一個. 例如: 剪頭髮
  - gunicorn default mode
    - Good: CPU-bound applications (scientific computation)
    - Bad: High I/O bound applications (ex. DB read/write)

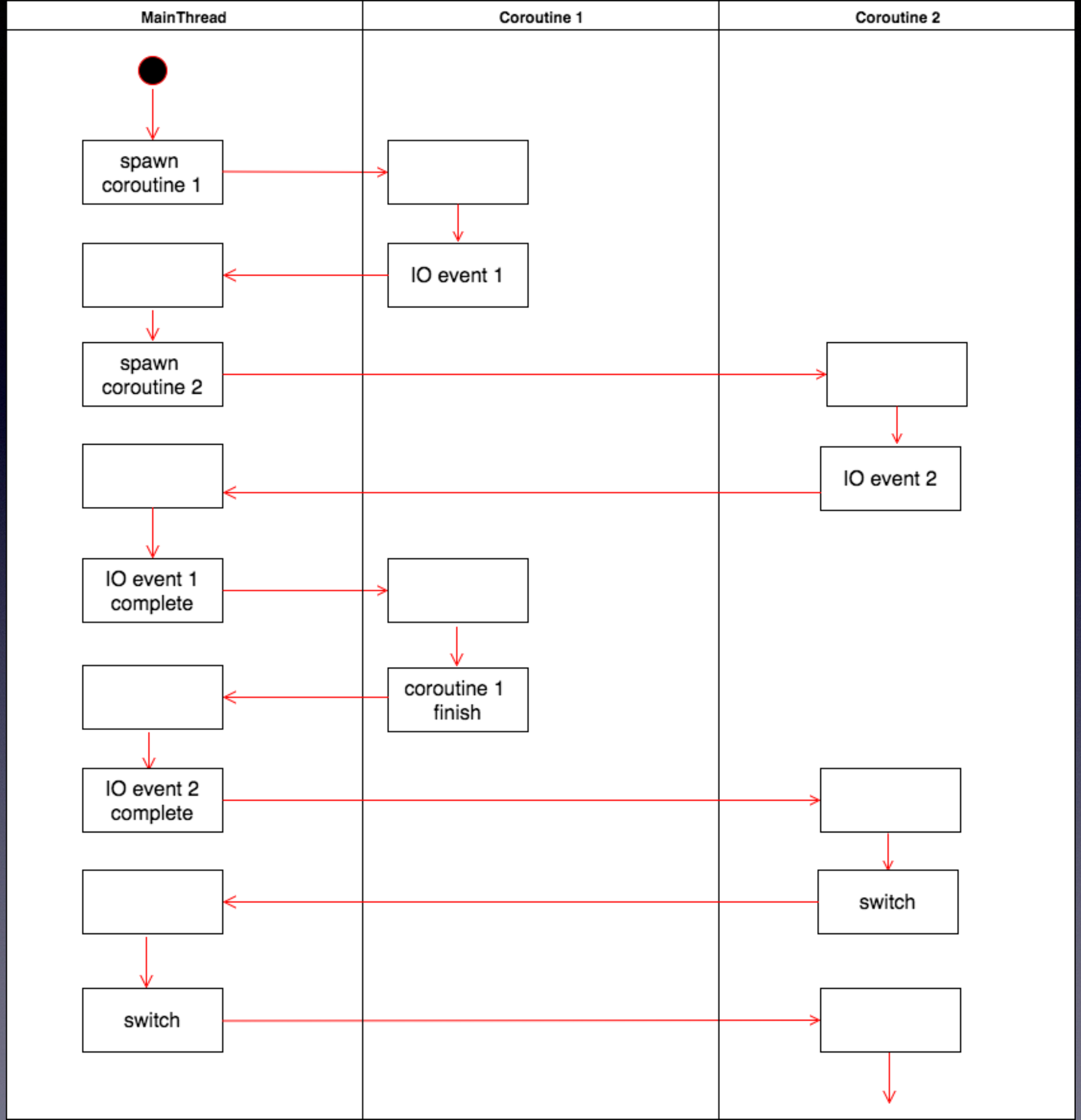
- 練習測試: Sleep 10秒的response
  - gunicorn -w 1 guni_sleep:app
    - 可測試同時開2個terminal (滑鼠中鍵) request觀察response時間
    - 測試指令: date && curl localhost:8000 && date
  - gunicorn -w 2 guni_sleep:app
    - 同時下兩個curl測量時間, 可比較一下有哪裡不同

Asynchronous Server

- 一個request處理過程中，還可以處理其他request
  - 例如: 老闆幫顧客A炸雞排，同時幫顧客B裝飲料
- Asynchronous是達成concurrency的一個手段
- gunicorn async mode:
  - **gevent: coroutine**
  - Others: tornado(event-loop) ...

Coroutine

- Light-weight thread (user space thread)
- Coroutine切換的過程是user決定，不是OS！
  - OS不知道coroutine的存在
  - No context-switch overhead like thread，開好開滿
- Execute multi coroutines concurrently with just one single thread


MainThread
Coroutine 1
Coroutine 2
spawn coroutine 1
IO event 1
spawn coroutine 2
IO event 2
IO event 1 complete
coroutine 1 finish
IO event 2 complete
switch
switch

gevent

- Coroutine network library
  - based on greenlet library
    - *synchronous-style code that runs asynchronously*
    - IO events trigger coroutine switch
      - Socket / file / database operatons

Gevent & coroutine

- **測試練習**

- gr_test.py: greenlet coroutine library
- gevent_test.py: gevent based on greenlet

Asynchronous server with Gevent

- 每個request都是一個greenlet coroutine
- **練習測試**
  - gunicorn -w 1 asyn:app --worker-class gevent
    - 同時開多terminal 觀察 response time是否都是10 seconds
      - date && curl localhost:8000 && date

Falcon

- a fast / minimal python web framework to build backend applications
- building web API easier
  - on_get / on_post / on_delete / on_patch …

Falcon API

- Falcon API initialize:
  api_router = falcon.API()
  api_router.add_route('/', RootHandler())
    - add_route parameters:
      - 1. API路徑
      - 2. 實作GET/POST...的類別實體
- API implementation format: (post / patch…亦同)
    - on_get(self, req, resp): # req: dict型態，包含WSGI環境變數
      resp.body ="Hello World!"

練習

- 修改homework/server.py
- 使用falcon + gevent + gunicorn架一個後端伺服器，並新增下列web API
- /account（GET，POST)
  - get自帶兩個參數account, nickname
  - post自帶兩個參數account, nickname並轉成json