CPython, Grumpy, PyPy
When, How, Why?
About Me

- Itay Weiss
- 27 Years - Alive
- 9 Years - Software Developer
- 7 Years - Army Intelligence
- 2 Years - Iguazio
CPython $\neq$ Python

CPython = A Language Engine

Python = Programming Language
• CPython:
  ○ Is used by the majority of developers and users.
  ○ Compiles your python code into bytecode (transparently) and interprets that bytecode in an evaluation loop.
  ○ Performance is mainly limited by the GIL
GIL (Global Interpreter Lock)

- A binary semaphore that prevents multiple native threads from executing Python bytecodes at once.
- Must be held by the current thread before it can safely access Python objects.
- In order to emulate concurrency of execution, the interpreter regularly tries to switch threads (sys.setswitchinterval).

GIL ≠ Mutex
Simple Solution
<table>
<thead>
<tr>
<th>MultiProcessing</th>
<th>MultiThreading</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PROS</strong></td>
<td><strong>PROS</strong></td>
</tr>
<tr>
<td>● Code is usually straightforward</td>
<td>● Shared memory</td>
</tr>
<tr>
<td>● Takes advantage of multiple CPUs &amp; cores</td>
<td>● Low memory footprint</td>
</tr>
<tr>
<td>● Avoids GIL limitations for CPython (basically each process has its own GIL)</td>
<td></td>
</tr>
<tr>
<td>● Child processes are interruptible/killable</td>
<td></td>
</tr>
<tr>
<td><strong>CONS</strong></td>
<td><strong>CONS</strong></td>
</tr>
<tr>
<td>● Separate memory space</td>
<td>● CPython - subject to the GIL</td>
</tr>
<tr>
<td>● Costly to run a huge number of processes</td>
<td>● Code is usually harder to understand and to get right - the potential for race conditions increases dramatically</td>
</tr>
<tr>
<td>● Costly to create and destroy processes frequently</td>
<td></td>
</tr>
<tr>
<td>● IPC a little more complicated with more overhead</td>
<td></td>
</tr>
<tr>
<td>● Larger memory footprint</td>
<td></td>
</tr>
</tbody>
</table>
Not So Simple Solution

but it’s haaaaard.
Twisted/async.io (Asynchronous Programming/Event driven programing)
When Should I Use CPython

- GUI applications
- Network servers
PyPy

- Supports all of the core language (passing Python test suite)
- Supports most of the commonly used Python standard library modules (just check if it imports. If it imports, it should work):
  - __builtin__, __pypy__, __ast, _cffi_backend, _codecs, _collections, _continuation, _csv, _file, _hashlib, _io, _locale, _lsprof, _md5, _minimal_curses, _multibytecodec, _multiprocessing, _numpypy, _pickle_support, _pypyjson, _random, _rawffi, _sha, _socket, _sre, _ssl, _struct, _testing, _warnings, _weakref, array, binascii, bz2, cStringIO, cmath, cppyy, cpyext, crypt, errno, exceptions, fcntl, gc, imp, itertools, marshal, math, mmap, operator, parser, posix, pwd, pyexpat, pypyjit, select, signal, symbol, sys, termios, thread, time, token, unicodedata, zipimport, zlib
- Built using the RPython language that was co-developed with it:
  - A framework for producing implementations of dynamic languages (like python)
  - Able to automatically generate a Just-in-Time compiler for any dynamic language
- Implements a JIT compiler in Python.
- Support for Stackless and Lightweight concurrent programming (Greenlets).
How Are PyPy Getting Rid Of The GIL

```python
def f(list1, list2):
    x = list1.pop()
    list2.append(x)
```

```python
def f(list1, list2):
    global_lock.acquire()
    x = list1.pop()
    list2.append(x)
    global_lock.release()
```
def f(list1, list2):
    while True:
        t = transaction()
        x = list1.pop(t)
        list2.append(t, x)
        if t.commit():
            break
JIT (Just In Time)

- Done during execution (runtime).
- Continuously analyses the code being executed and identifies parts of the code where the speedup gained from compilation would outweigh the overhead of compiling that code.
- In theory JIT compilation can yield faster execution than static compilation.
Demo
Performance + How to use PyPy
When Should I Use PyPy

- Long running processes
- When running mostly Python code.
Grumpy

- Google has YouTube which is:
  - Written In CPython (2.7)
  - Serves millions of requests per second
- Google decides to Implement an alternative runtime optimized for real-time serving.
- Google decides to use Go.
- Grumpy was born!
Grumpy

- Python to Go source code transcompiler and runtime.
- Grumpy has no VM.
- Has no GIL.
- Uses Go’s garbage collection instead of counting references.
- Uses Go’s goroutines
- No support for C extension modules.
Grumpy Features

Missing features
Dylan Trotter edited this page on Apr 3 - 10 revisions

Language features
- decorators
- lambdas
- pow operator
- relative imports
- importing module members
- wildcard import

Types
- buffer
- BufferError
- bytearray
- FloatingPointError
- GeneratorExit
- KeyboardInterrupt
- memoryview

Functions
- apply()
- coerce()
- filter()
- format()
- globals()
- input()
- intern()
- locals()
- map()
- pow()
- reduce()
- reload()
- reversed()
- round()
- sorted()
- vars()
Demo

Performance + How to use Grumpy
It All Comes Down To This
When Should I Use Grumpy

- When you’re using a lot of threads, each executing a very small task.
- When you need to use any of the go libraries.
<table>
<thead>
<tr>
<th>CPython</th>
<th>PyPy</th>
<th>Grumpy</th>
</tr>
</thead>
</table>
| - The reference implementation of Python  
- Written in C  
- Compiles Python code to intermediate bytecode which is then interpreted by a virtual machine | - Supports all of the core language (passing Python test suite)  
- Supports most of the commonly used Python standard library modules  
- Implements a JIT compiler in Python | - Python to Go source code transcompiler.  
- Grumpy has no VM.  
- Uses Go’s garbage collection instead of counting references. |
## Performance Comparison

<table>
<thead>
<tr>
<th>GIL</th>
<th>JIT</th>
<th>Go</th>
</tr>
</thead>
</table>
| ● A binary semaphore that prevents multiple native threads from executing Python bytecodes at once.  
   ● Must be held by the current thread before it can safely access Python objects | ● Done during execution (runtime).  
   ● In theory JIT compilation can yield faster execution than static compilation. | ● Has no GIL.  
   ● Statically compiled.  
   ● Uses Go’s garbage collection instead of counting references.  
   ● Uses Go’s goroutines |
Thanks

- David Beazley (http://www.dabeaz.com/) - https://www.youtube.com/watch?v=ph374fJqFPE
- Emmanuel Klinger - http://emmanuel-klinger.net/python-100-times-faster-than-grumpy.html
- PyPy blog: https://morepypy.blogspot.co.il/2011/06/global-interpreter-lock-or-how-to-kill.html
- Stackoverflow
  - https://stackoverflow.com/questions/18946662/why-shouldnt-i-use-pypy-over-cpython-if-pypy-is-6-3-times-faster
  - https://stackoverflow.com/questions/3044580/multiprocessing-vs-threading-python
More Questions

me@itayweiss.tech