VNX TA Design, SplunkWeb and Python

Ken Chen
2014/08/07

Agenda

- Python Overview
- SplunkWeb
    - The architecture behind
    - How everything hook up together
- VNX TA
    - Current design
    - Testing
    - Postmortem
- Lessons learned

Life is short,
I use Python

Python is a perfect language!

- either you are a fool or
- a sales man

— Ken

C++ is a perfect language!

- either you are a fool or
- a sales man

— Bjarne

Python Overview

Python Overview

- Python in one slide
  - A (backend) script language
  - Auto memory mgmt by reference counting
  - A Dynamic typing language
  - A Strongly typed language
  - Object oriented/based
  - Platform independent

Python Overview

- Python in one slide (I am lying)
  - One language spec, different implementation
  - CPython (C)
  - Jython (Java)
  - Cython (Written in Python, translate to C)
  - PyPy (RPython/Written in Python/GIT)
  - IronPython (.NET)
  - …

Python Overview

- The good
  - Simple
  - One coding style (important)
  - One and the only best way to solve a problem
    - VS 16 ways to stack a cat (Bjarne)
  - Has good language features
  - Rich standard and 3rd party libraries
  - Community is super good
  - Dynamic typing
  - Productive

Python Overview

- The bad
  - Generally slow (We will see)
  - Single thread due to GIL (We will see)
  - Py3k is not backward compatible with Python 2

Python Overview

- How Slow
  - Let’s do a file system scanning
  - Let’s do a Fibonacci
- Why slow
  - 90 % of the time, we program in a slow way
  - function call is relative expensive in Python
  - The missing high performance APIs
  - Not pythonic

Python Overview

- The missing APIs
  -
```
def unmarshal(filename, ObjectClass):
    objs = []
    with open(filename) as f:
        for lin in f:
            lin = lin.strip():
            if lin:
                objs.append(ObjectClass(lin))
    return objs
```

Python Overview

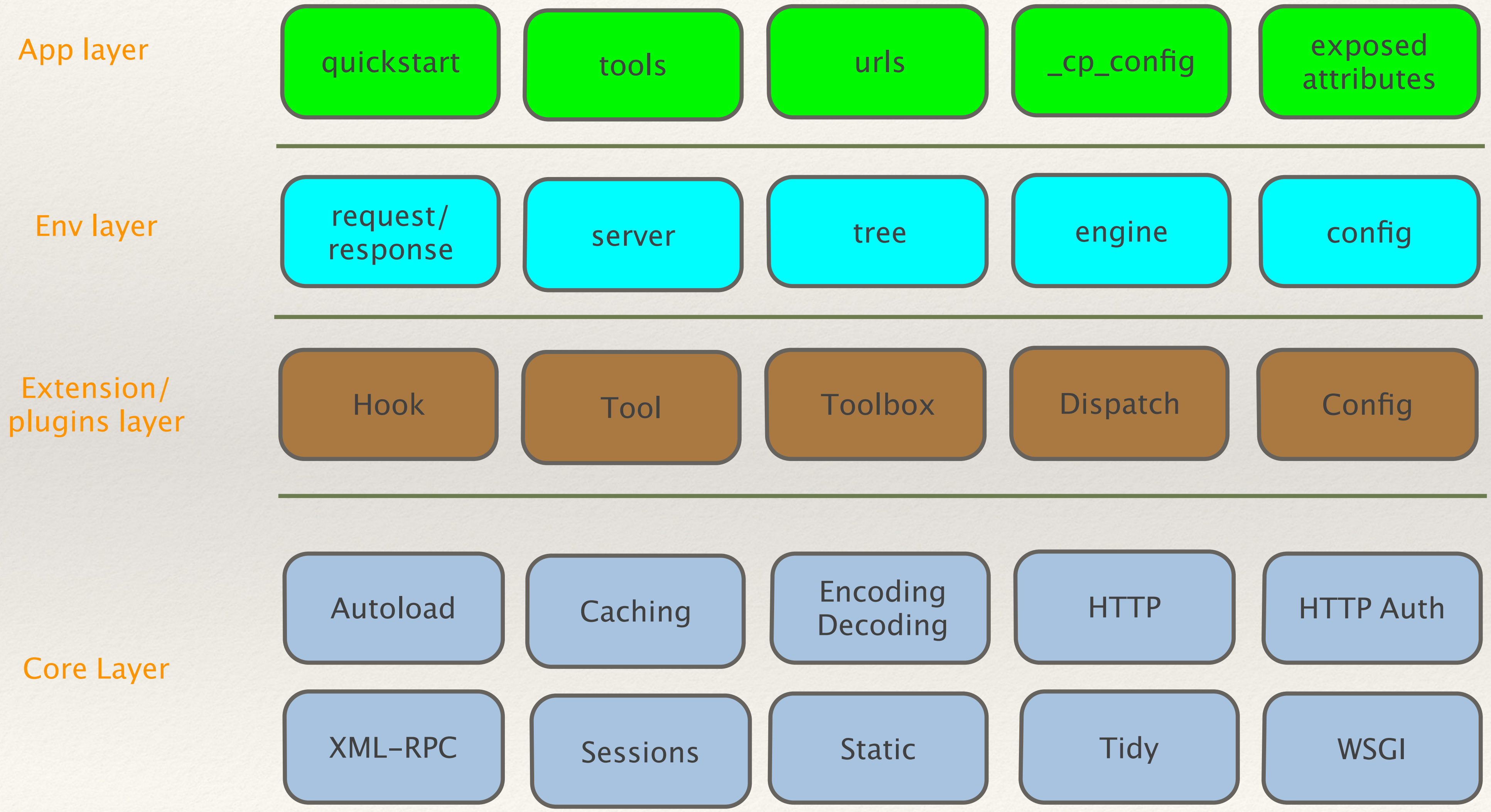
- Pythonic (Python idiom)
  - Code style
    - "if file== None" is not Pythonic
  - Performance
    - "for i in range(100000)" is not Pythonic
  - There are a few guidelines, but far less than the the guidelines in C++
  - Am I talking about micro-optimisation ?

Python Overview

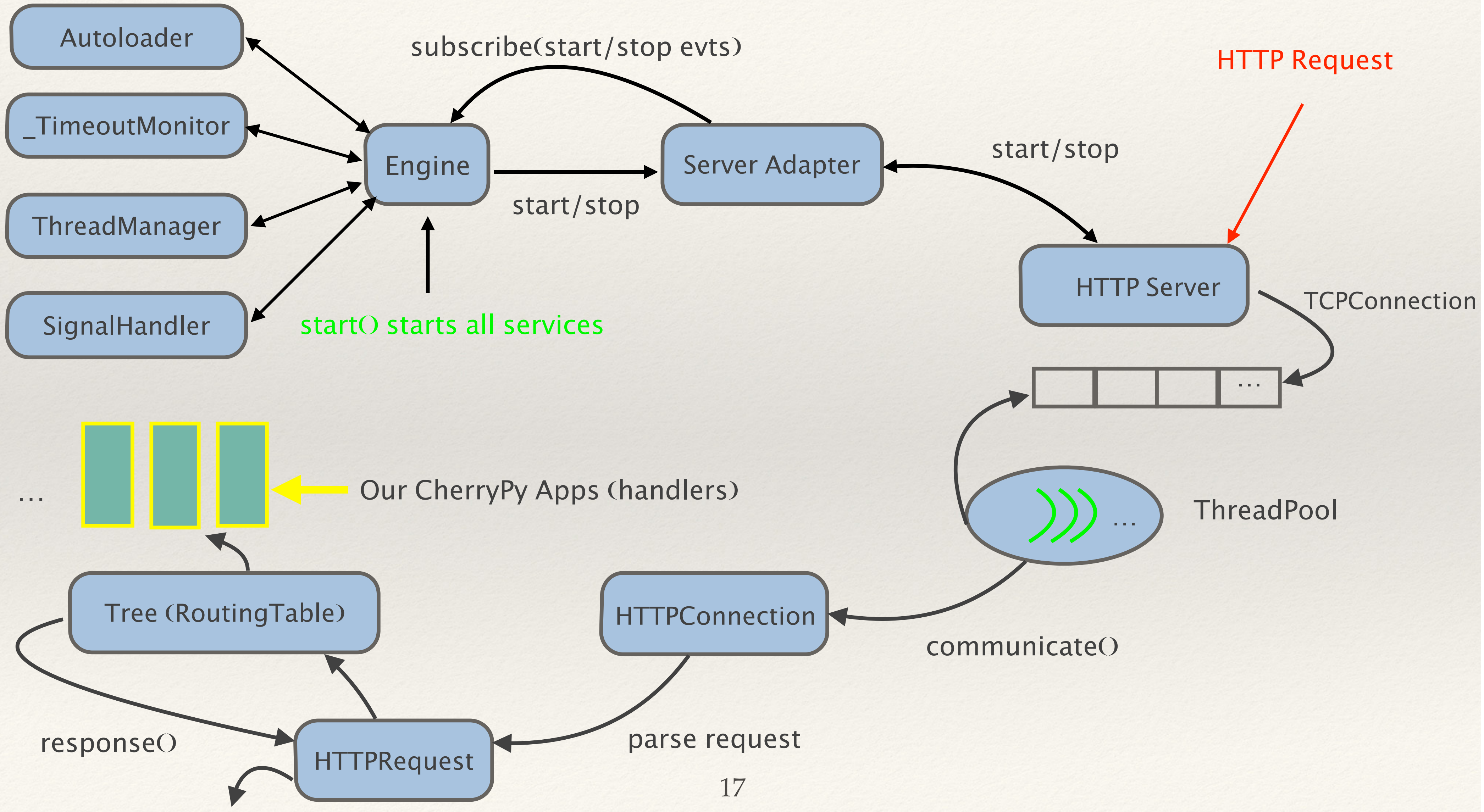
- How to stay Pythonic
  - Read code (Python standard lib)
  - VIM + Plugins (Allow me to show you)
  - pep8.py
  - pylint
  - PyCharm or other IDE

Splunkweb server CherryPy

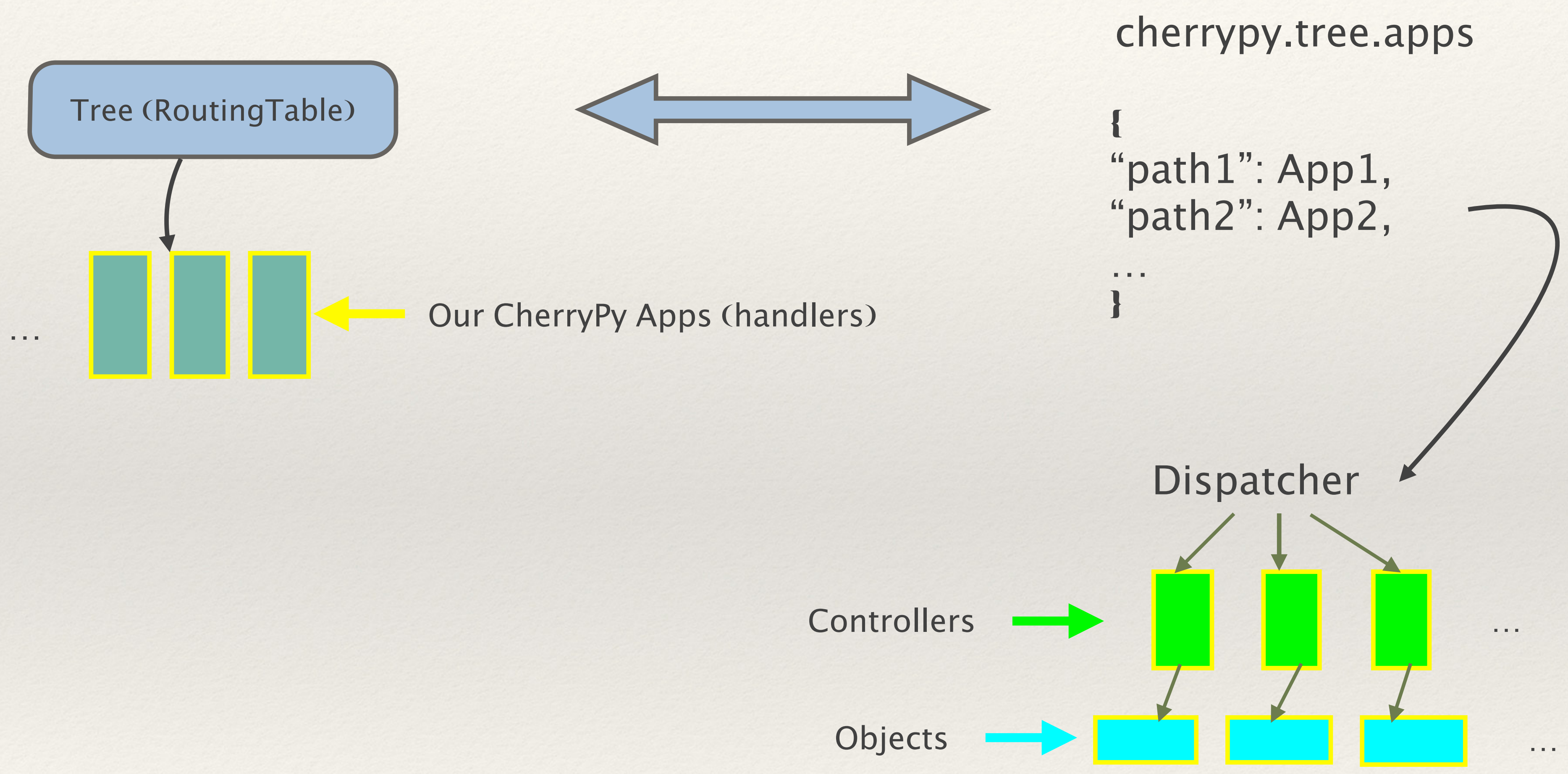
CherryPy


App layer
quickstart
tools
urls
_cp_config
exposed attributes
Env layer
request/ response
server
tree
engine
config
Extension/ plugins layer
Hook
Tool
Toolbox
Dispatch
Config
Core Layer
Autoload
Caching
Encoding Decoding
HTTP
HTTP Auth
XML-RPC
Sessions
Static
Tidy
WSGI

CherryPy architecture



CherryPy the routing table

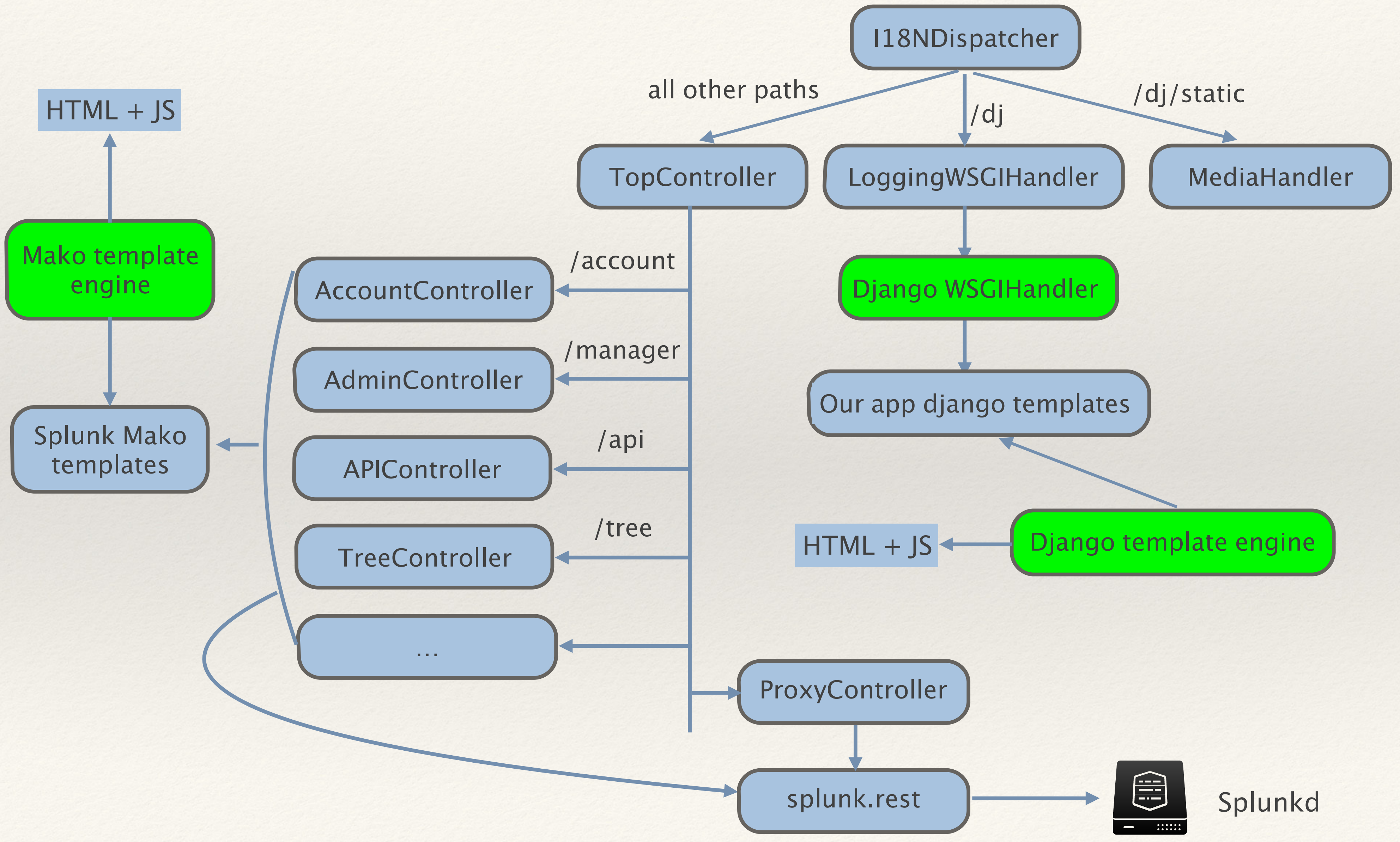


Splunk Apps (Django/CherryPy)

Glue up everything

- Discover CherryPy handlers
  - ControllerMetaLoader discovers all apps by matching "etc/apps/<app>/appserver/controllers" folder
  - CustomController attaches handler's endpoints
  - app's CherryPy controller inherits from "BaseController"
- Discover Django apps
  - $SPLUNK_HOME/etc/apps/framework/server/splunkdj/loaders/apps_finder.py
  - Matching "etc/apps/<app>/django/<app>" folder

The routing tree



SplunkWeb backend in 2 slides

- The third parties
  - CherryPy
  - Django
  - Mako lib
  - OpenSSL
  - httplib2
  - lxml
  - Others - Babel, beaker, reportlab

SplunkWeb backend in 2 slides

- In house components
  - App Controllers
  - clilib
  - Django templates/tags
    - Bridge Django CORE and Django App
  - Mako templates

VNX TA Design

VNX TA main functionalities

- Data collection
  - VNX File, inventory/perf metrics
  - VNX Block, inventory/status/perf
  - Output parsing
  - Storage objects unmarshaling
- CIM Mapping
- Eventgen sampling

VNX TA main functionalities

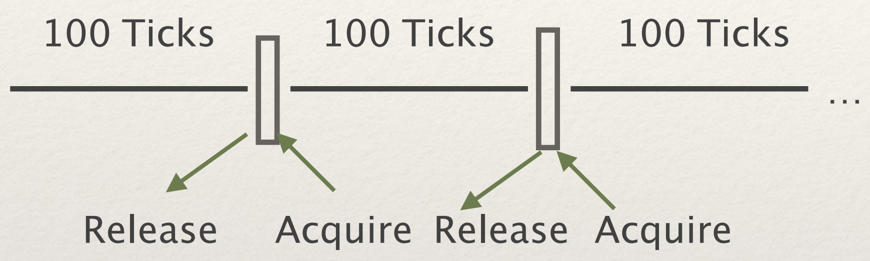
- Key challenges
  - VNX itself is hard hard to manage, not a unified system, no unified APIs
  - Multiple machines
  - Blocking call and many calls
  - VNX mgmt path is not designed for high concurrency and perf
  - Python language limitation (GIL)
  - Testing env ($$$)

GIL

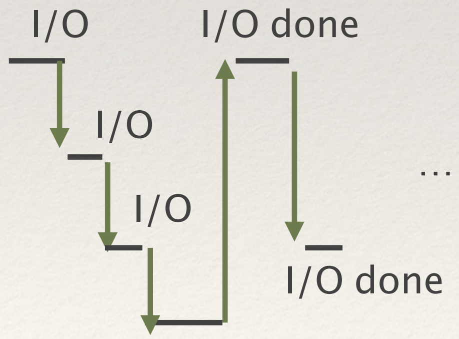
- The Global Interpreter Lock
  - Only one thread is able to run
  - Parallel executing the code is not possible
  - Concurrency is still possible
  - Concurrency is not parallelism (Rob Pike, Go)

GIL

- CPU Bound



- IO Bound

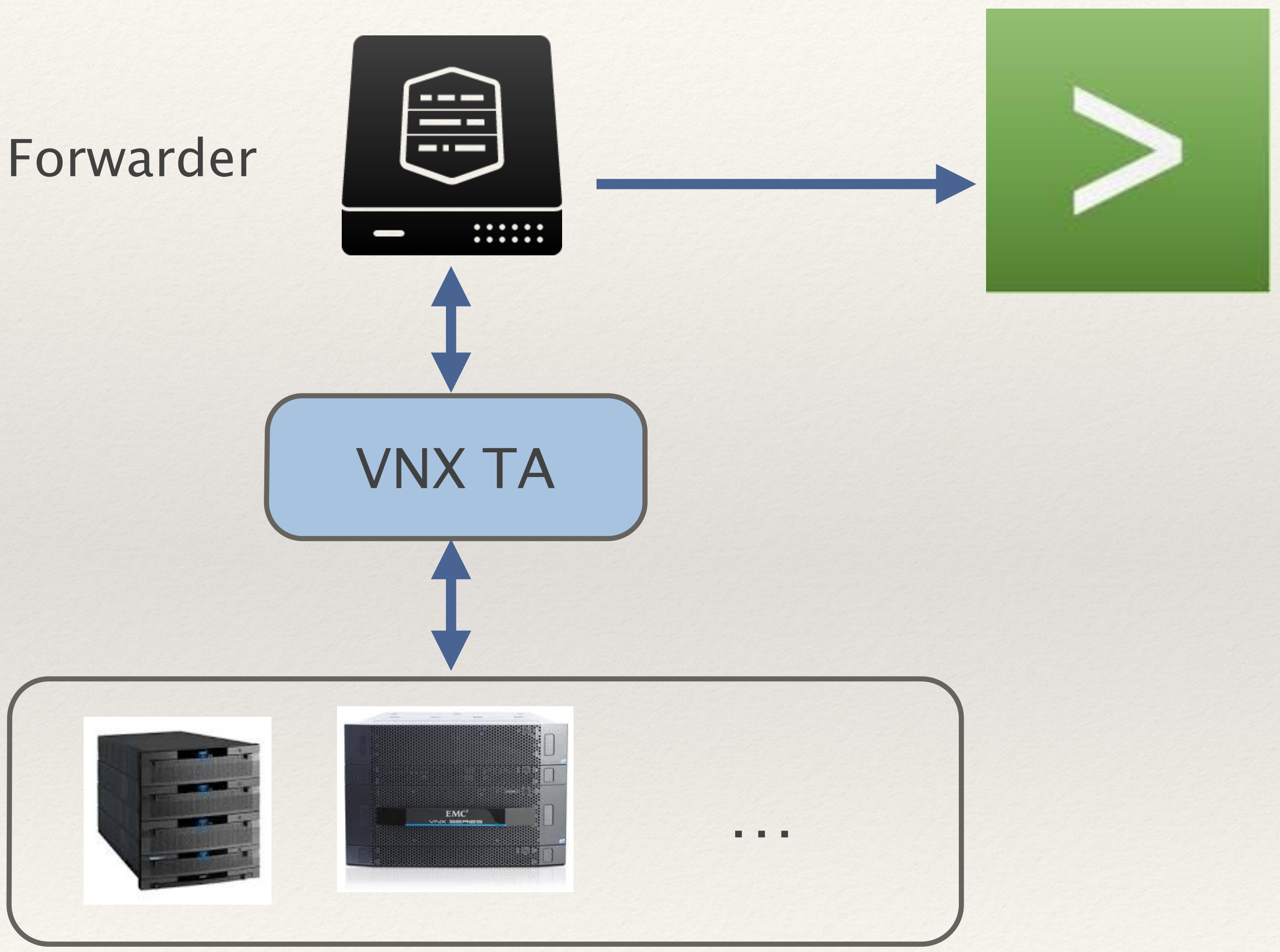


GIL

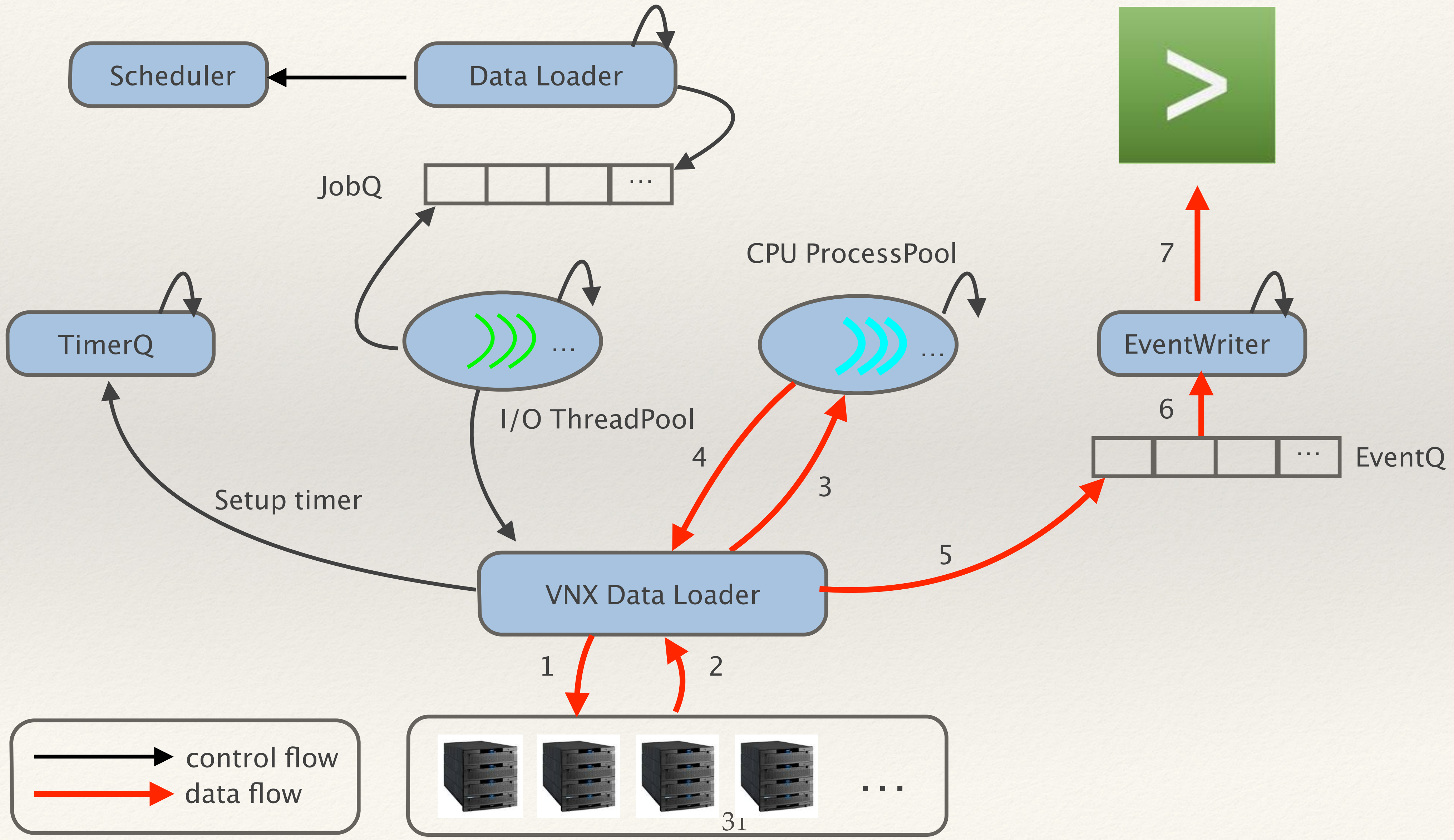
- Python/ceval.c (CPython 2.7.8)

```
if (--_Py_Ticker < 0) {                              Counting down
 ...
 _Py_Ticker = _Py_CheckInterval;                     Reset the counter
 ...

 if (pendingcalls_to_do) {                           Handle signals
     Py_MakePendingCalls();
     ...
 }

 if (interpreter_lock) {
    /* Give another thread a chance */
    ...
    PyThread_release_lock(interpreter_lock);
                                                     Release and reacquire the lock
    /* Other threads may run now */

    PyThread_acquire_lock(interpreter_lock, 1);

    ...
    /* Check for thread interrupts */
    ...
  }
}
```

VNX TA high level design



VNX TA architecture



Pros and cons

- Pros
  - Long lived, no startup overhead
  - Gracefully handle network/load jitter
  - No checkpoint. Maintain status in memory
  - Multithread for I/O, multiprocess for computing, effective workaround GIL
  - Time service handles hung requests
  - Graceful tear down (pay attention to you signal handler)
- Cons
  - SHELL is hard to scale up as splunkd does(async I/O is the right way). Randomization relieves the problem somehow
  - Relatively Complicate

Testing

- Unit test is an absolute need
  - It is hard to test the parser (different output in different version)
  - Need setup test strategy
  - The TA employs replaying outputs from different VNX version
- Performance test
  - Mocks up CLI, simulate verbose output

Postmortem

- Logging
  - Separate the logs (Generic ones and BLOs)
  - Timestamp and thread id is necessary
  - Support debug level switch
    - Verbose logging
    - All raw outputs are captured and saved

Lessons learned

- Embrace GIL
- Embrace threading (I/O)
- Embrace multiprocessing (computation)
- Embrace pythonic
- Embrace pdb (cover little in this session)
- Embrace logging
- Program at high level and think at low level

Life is short,
You need Python

Thread termination

Destroy me !