Actors

What, Why, and How

Donovan Preston

What Is an Actor?

Actor Model

- http://en.wikipedia.org/wiki/Actor_model
- The Actor model is a mathematical model of concurrent computation that treats "actors" as the universal primitives of computation.

Actor Constraints

An Actor Is a Process

An Actor Can Change its State

An Actor Can Create Another Actor and Get its Address

An Actor Can Send a Message To Any Addresses it Knows

An Actor Can Wait for a Specific Message to Arrive in its Mailbox

Why Use Actors?

Isolation

- Only an Actor can change its own state, simplifying logic
- Locking not required
- Potential for race conditions reduced

Simple Control Flow

- Each Actor's control flow is independent
- Code can be written straight line or with simple loops

Message Passing

- Easy to distribute
- Across cores
- Across UNIX process boundaries
- Across machines

Simplified Error Handling

- Most exceptional conditions occur while waiting for a message
  - Timeouts
  - Network errors
- Isolates error handling code
- Makes it easier to build fault tolerant systems

How Are Actors Implemented?

Existing Actor Systems

- Erlang http://erlang.org/
- Io http://www.iolanguage.com/
- Python
  - PARLEY http://osl.cs.uiuc.edu/parley
  - Dramatis http://pypi.python.org/pypi/dramatis
  - Candygram http://candygram.sourceforge.net/

I Built My Own ...for science!

http://bitbucket.org/fzzzy/python-actors

What Did I Choose?

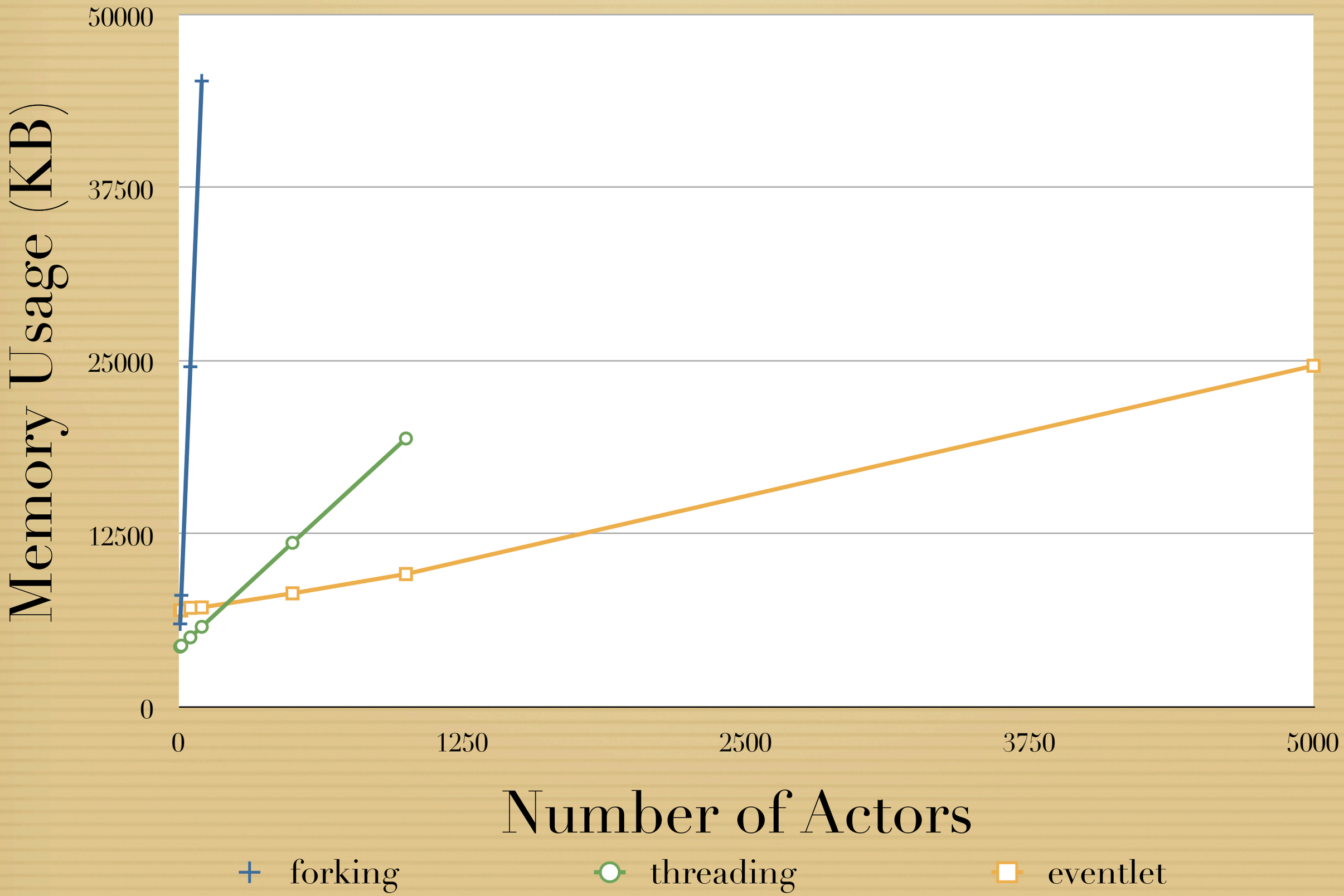
- Green Threads for "Processes"
  - Using eventlet
    - Which uses greenlet
- JSON for message serialization
  - Used to copy messages between in-process actors
- WSGI/HTTP for network protocol

Green Threads?

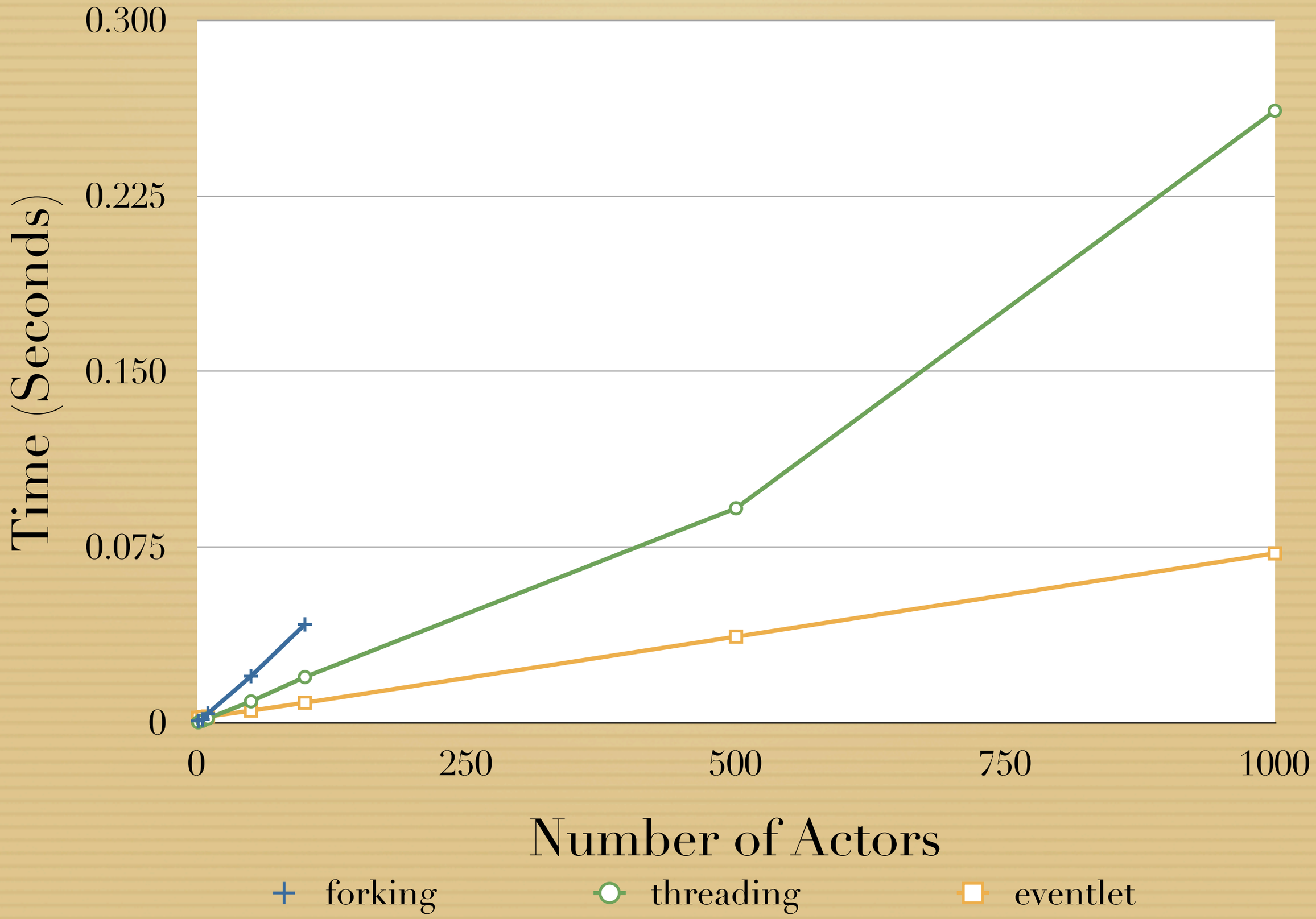
- Why not:
  - POSIX Threads
  - OS Processes

- Speed
- Memory Usage

Memory Usage



Time


0.300
0.225
0.150
0.075
0
Time (Seconds)
0
250
500
750
1000
Number of Actors
forking
threading
eventlet

Spawning an Actor

```
from pyact import actor


class Hello(actor.Actor):
    def main(self, what):
        print "Hello,", what


actor1 = Hello.spawn("world")
actor2 = Hello.spawn("pycon")
```

Message Copying

- Messages are serialized into JSON
- Messages to in-process actors are thus copied
  - Preserves isolation
- JSON is ready to go over the network

Sending a Message

```
from pyact import actor

class Send(actor.Actor):
    def main(self, address):
        print "Sending message"
        address.cast("hello")

class Receive(actor.Actor):
    def main(self):
        print "Receiving message"
        pattern, message = self.receive()
        print "Message:", message

receiver = Receive.spawn()
sender = Send.spawn(receiver)
```

Pattern Matching

- “An Actor Can Wait for a Specific Message to Arrive in its Mailbox”
  - In Erlang this is called “Selective Receive”
  - Also known as “Pattern Matching”
  - Python doesn’t have this
    - My implementation is called “shaped”

Possible Message Contents

- dict
- list
- string
- int
- float

```
{'hello': str}

[str]

(int, str, float)

{'nested': {str: str}}

{'exact': 'match'}
```

Matching a Message

```
class Send(actor.Actor):
    def main(self, address):
        print "Sending message"
        address.cast("This does not match")
        address.cast({u'hello': u'world'})

class Receive(actor.Actor):
    def main(self):
        print "Receiving message"
        pattern, message = self.receive({u"hello": unicode})
        print "Greeting Target:", message['hello']

receiver = Receive.spawn()
sender = Send.spawn(receiver)
```

Network Protocol

- I chose:
  - HTTP
    - wsgi application
  - JSON
  - REST

REST Interface

- PUT spawns an actor
  - Redirects to an Address
    - POST <Address>
    - sends a message to actor

Problem

Python modules contain global state

Module Global Problem

- Possibility:
  - Keep a unique copy of sys.modules for every actor
    - Seal modules in wrapper objects to prevent modification
- Reality:
  - Just don’t use global module state

Q&A

Tell me what I don't know about Actors or Python