Concurrency and Distributed systems

...With Python today.

Jesse Noller

30,000 Foot View

- Introduction
- Concurrency/Parallelism
- Distributed Systems
- Where Python is today
- Ecosystem
- Where can we go?
- Questions

Hello there!

- Who am I?
- Why am I doing this?
- Email: jnoller@gmail.com
- Blog - http://www.jessenoller.com
- Pycon - http://jessenoller.com/category/pycon-2009/

Most of all, it's fun!

No Code, Why?

Bike sheds

Concurrency

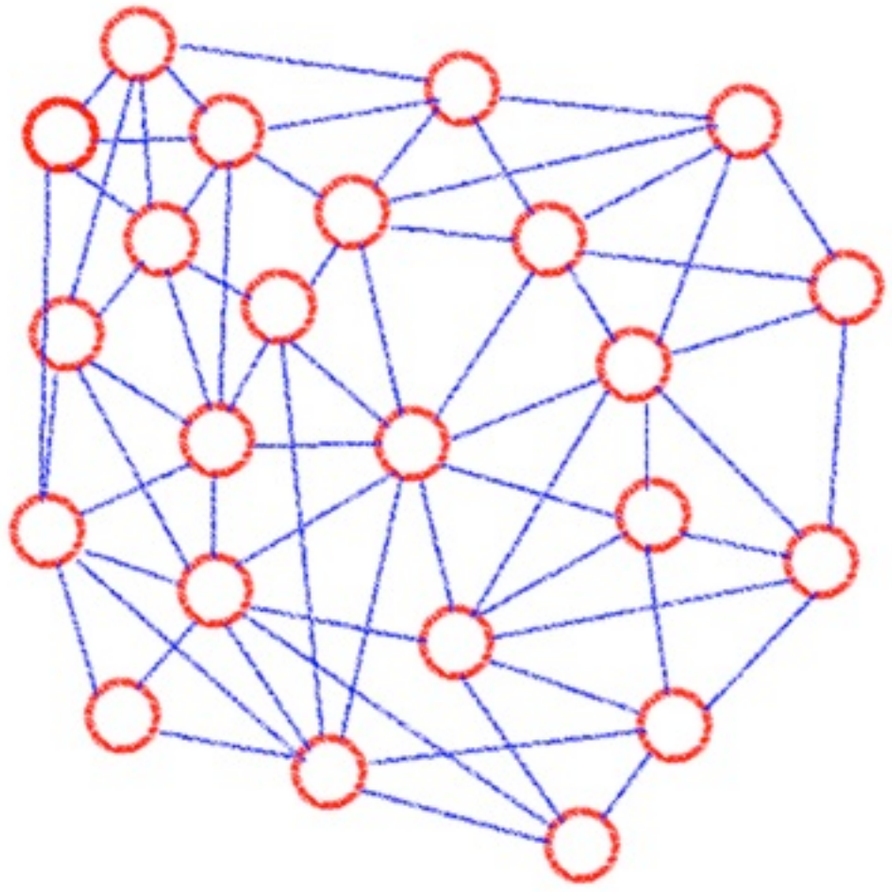
- **What is it?**
  - Doing many things "at once"
  - Typically local to the machine running the app.
- **Implementation Options:**
  - threads / multiple processes
  - cooperative multitasking
  - coroutines
  - asynchronous programming

... vs Parallelism

- **What is it?**
  - Doing many things **simultaneously**
- **Implementation options**:
  - threads
  - multiple processes
  - distributed systems

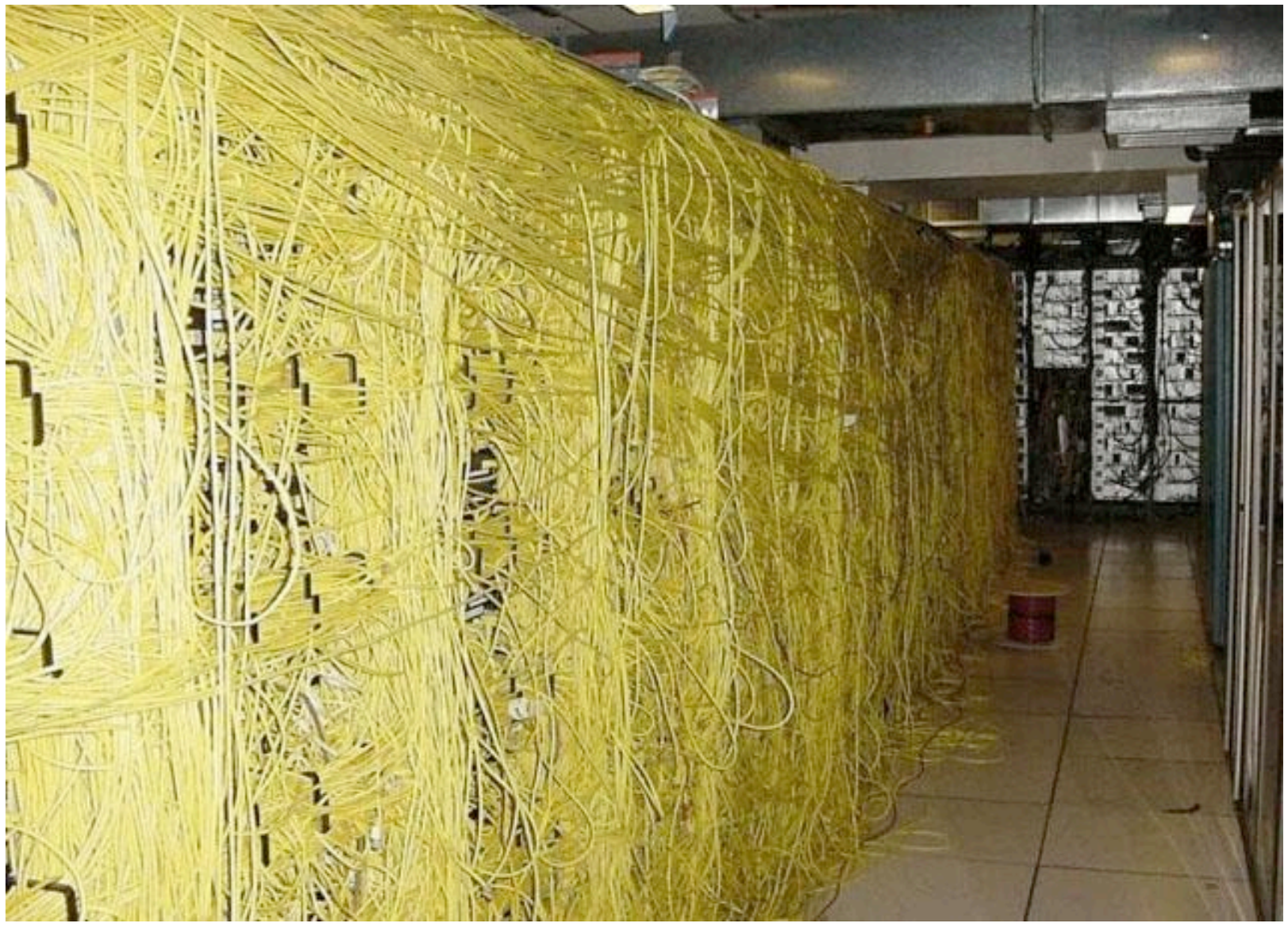
... vs Distributed Systems

- **What is it?**
  - Doing many things, across **multiple** machines, **simultaneously**
    - Many cores, on many machines
- **There are many designs**
  - There are eight fallacies...



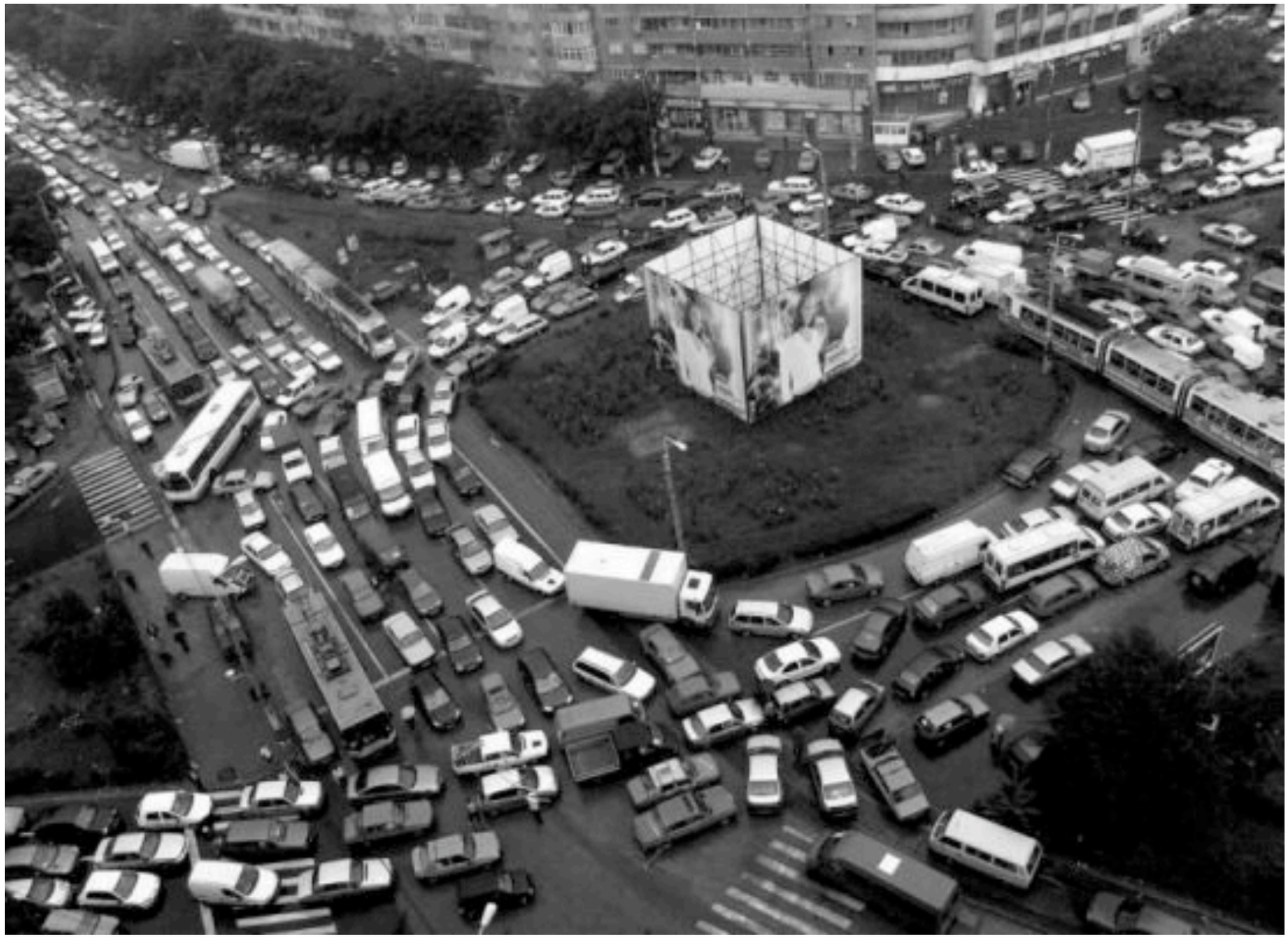
8 fallacies of distributed systems

The network is reliable



Latency is zero

Bandwidth is infinite



The network is secure

Topology doesn't change

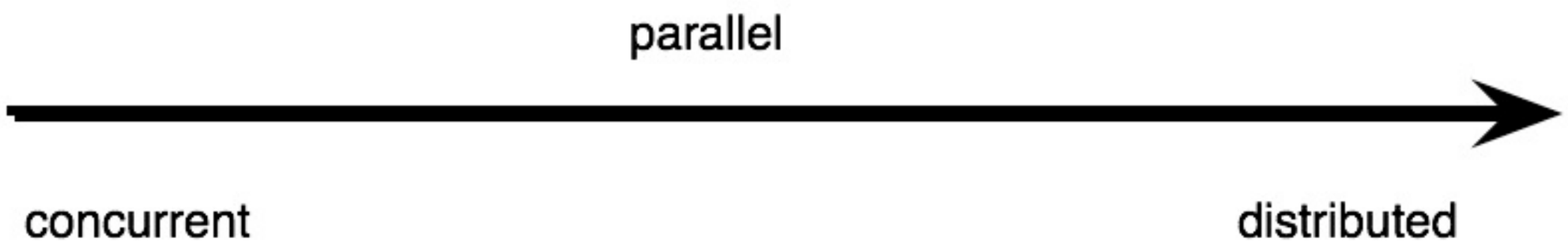
There is only one administrator

Transport cost is zero

The network is homogenous

Summary

- All 3 are related to one another, the fundamental goals of which are to:
  - Decrease latency
  - Increase throughput
- Applications start simple, progress to concurrent systems and evolve into parallel, distributed systems
- As the system evolves, the fallacies become more pertinent, you have to account for them early




python™

Where is (C)Python?

- We have threads. Shiny, real OS ones
  - Except for the Global Interpreter Lock
- The GIL makes the interpreter easier to maintain
  - ...And it simplifies extension module code

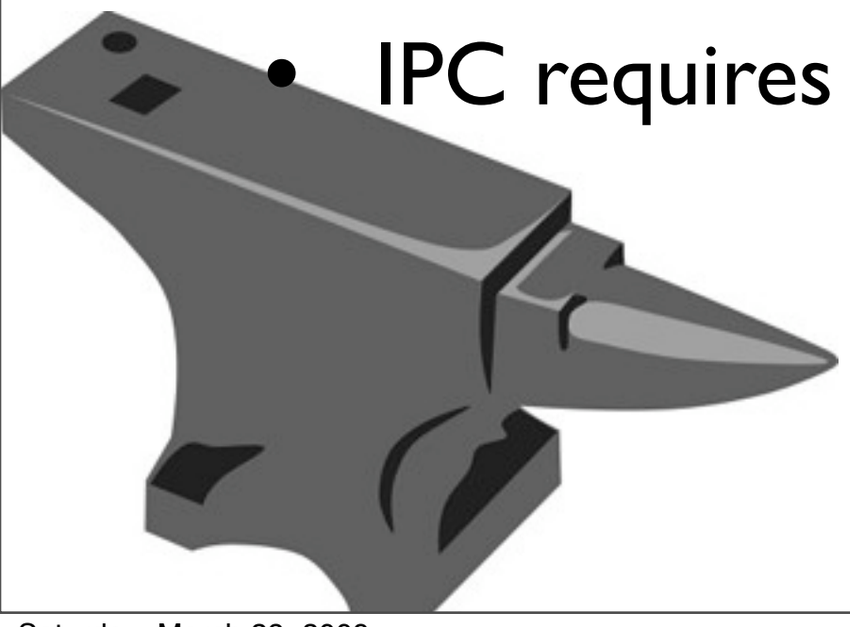
Is the GIL a problem?

- Yes. Sorta. Maybe. It depends.
  - I/O Bound / C extensions release it!
    - Most applications **are** I/O bound
      - The GIL still has non-zero overhead
- The GIL is not going away*
- **You can build concurrent applications regardless of the GIL**

* ... more on this in a moment, dun dun dun.

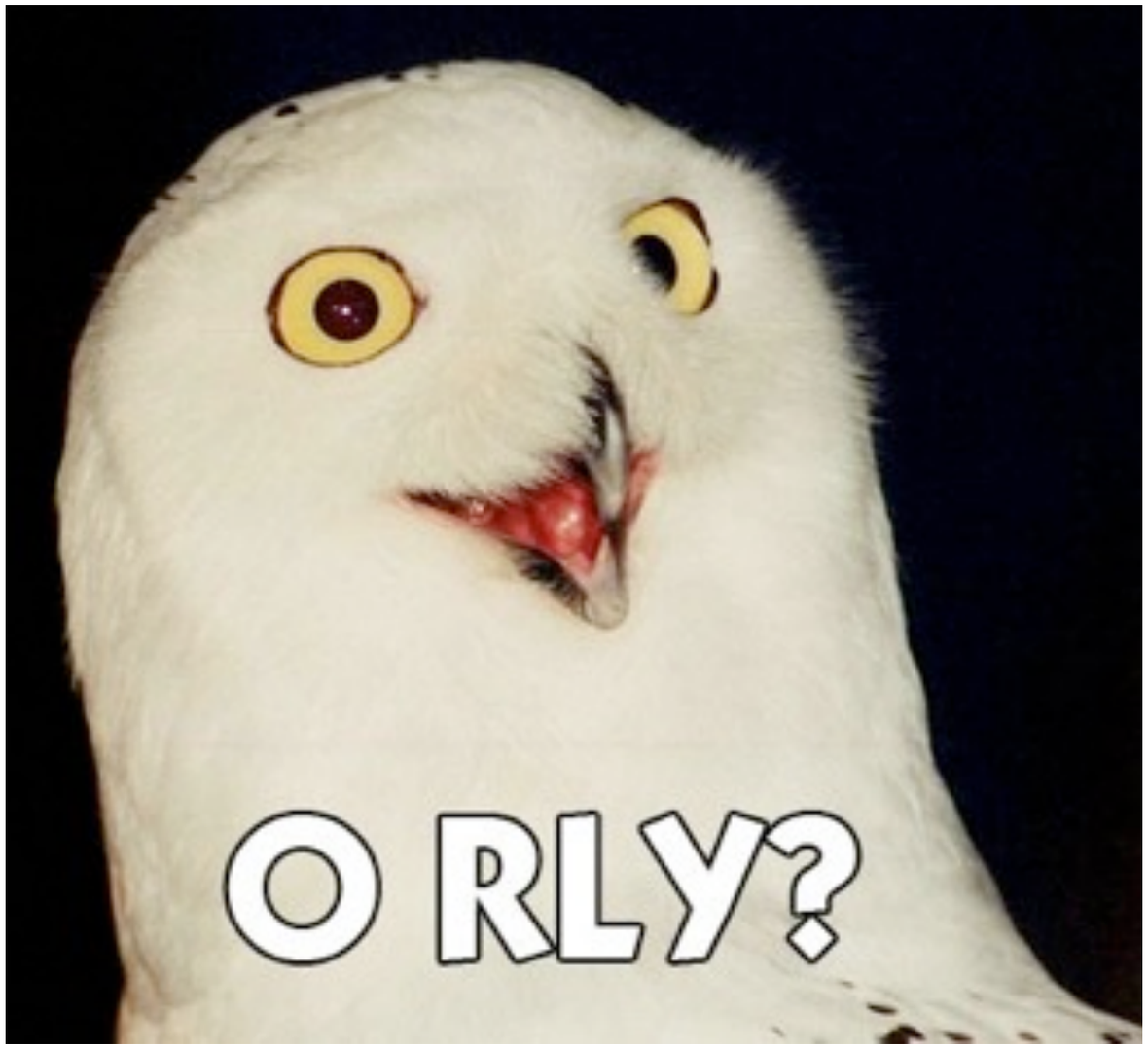
Multiprocessing!

- Added in the 2.6/3.0 timeline, PEP 371
- Processes and IPC (via pipes) to allow parallelism
    - Same(ish) API as threading and queue
    - Includes Pool, remote Managers for data sharing over a network, etc
- Multiprocessing "outperforms" threading
- IPC requires pickle-ability. Incurs overhead



Summary

- We have the Global Interpreter Lock
  - We also have multiprocessing (no GIL)
- Threads (as an approach) **are** good for some problems
  - They're not impossible to use correctly
  - While hampered, python threads are still useful
- Python still allows you to leverage other approaches to concurrency



(remember that asterisk?)*

- Python on the JVM (in Java)
- 2.5-Compatible
- Frank and the others are **awesome** for resurrecting this project
  - May allow python in the Java door

- Pros:
  - Unrestricted threading
  - Hooray java.util.concurrent!
- Cons:
  - No C extensions

IronPython

- Python on the .NET CLR
- 2.5.2 Compatible
- Matured rapidly, highly usable
- Great for windows environments

- Pros:
  - Unrestricted threading
  - Some C extensions via ironclad
- Cons:
  - Mostly windows only, barring mono

Stackless

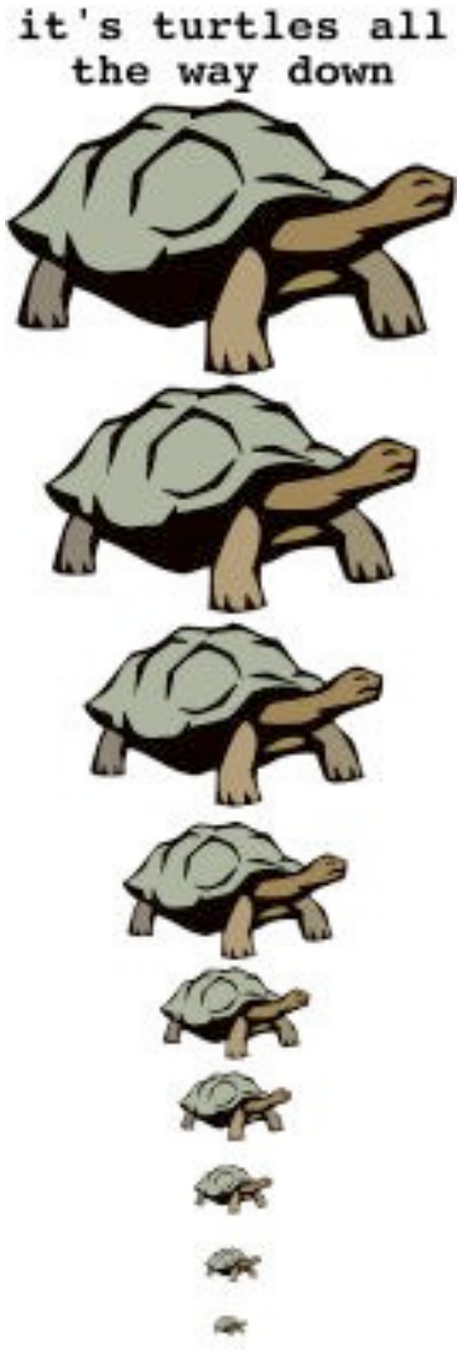
- Modified CPython interpreter
- Offers Coroutines, Channels - “lightweight threads”
- Cooperative multitasking (single thread executes)
- (mostly) Still alive courtesy of CCP Games
- Still has a GIL
- “Stackless is dead, long live PyPy”

- Python written in (R)Python
- Getting close to 2.5-Compatibility
- Complete “rethink” of the interpreter
- Focusing on JIT/interpreter speed right now
- Still has the GIL
- Some Stackless features (e.g. coroutines, channels)
- Not mature

The Ecosystem

That's a lot of nuts!

- When I started, I had around **40** libraries on my list
  - Coroutines, messaging, frameworks, etc
- Python has a huge ecosystem of "stuff"
  - Unfortunately, much of is long in the tooth, or of beta quality
- New libraries/frameworks/approaches are coming out every week



Concurrency Frameworks

Twisted

- "OK, who **hasn't** tried twisted?"
- Asynchronous, Event Driven multitasking
- **Vast** networking library, large ecosystem
- Supports thread usage, but **twisted** code may not be thread safe
- Supports using processes (not mprocessing).
- Can be mind-bending

Kamaelia

- Came out of BBC Research
- Uses an easy to understand "components talking via mailboxes" approach
- Cooperative multitasking via generators by default.
- **Honkin**' library of **cool things**
- Supports thread-based components as well
- **Very** easy to get up and running
- Abstracts IPC, Process, Threads, etc "away"

Frameworks

- Both **kamaelia** and **twisted** have nice networking support
- Both use schedulers which allow scheduled items to schedule other items
- Two different approaches to **thinking** about the problem
- Both can be used to build **distributed** apps
- Like all frameworks, you adopt the methodology

New: Concurrence

- New on the scene ('09) version 0.3
- Lightweight tasks-with-message passing
- Has a main scheduler/dispatcher
- Built on top of stackless/greenlets/libevent
- Network-oriented (HTTP, WSGI servers)
- Still raw (more docs please)
- **Very** promising (minus compilation problems)

Coroutines

- Coroutines are essentially light-weight threads of control, Think micro/green threads
- Typically use explicit task switching (cooperative)
- Most implementations have a scheduler, and some communications method (e.g. pipes)
- Not **parallel** unless used in a distributed fashion
- Both Kamaelia and Twisted "fit" here
- Enhanced generators make these easy to build

Coroutine libraries

- **Fibra**: microthreads, tubes, scheduler
- **Greenlet**: C based, microthreads, no scheduler
  - **Eventlet**: Network "framework" layer on top of *greenlet*. Has an Actor implementation \o/
- **Circuits**: Event-based, components/microthreads
- **Cogen**: network oriented, scheduler, microthreads
- **Multitask**: microthreads, no channels (it's dead jim)

Actors

- Isolated, self reliant components
- Can spawn other Actors
- Communicate via message passing **only** (by value)
- Operate in **parallel**
- Communication is asynchronous
- A good model to overcome the **fallacies**
- See also: Erlang, Scala

Actor Libraries

- Dramatis (**alpha** quality)
  - Great start, excellent base to start working with them
- Parley (**alpha** quality)
  - Another excellent start, supports actors in threads, greenlets or stackless tasklets
- Candygram (2004)
  - Old, implements erlang primitives, spawns in threads
- Kamaelia components can fit here(ish)

(local) Parallelism

- Multiprocessing
  - Processes and IPC via the threading API, in Python-Core as of 2.6
- Parallel Python
  - Allows local parallelism, but **also** distributed parallelism in a "full" package
- pprocess
  - Another easy to use fork/process based package
  - Has IPC mechanisms

Distributed Systems

- Lots of various technologies to help build something
  - communications libraries
  - socket/networking libraries
  - message queues
  - some shared memory implementations
- No “full stack” approach
  - Most users end up rolling their own, using some combinations of libraries and tools

Distributed Processing

- Frameworks:
  - Parallel Python is the closest for a processing cluster
  - The Disco Project is an erlang-based (with python bindings) map-reduce framework

RPC/Messaging

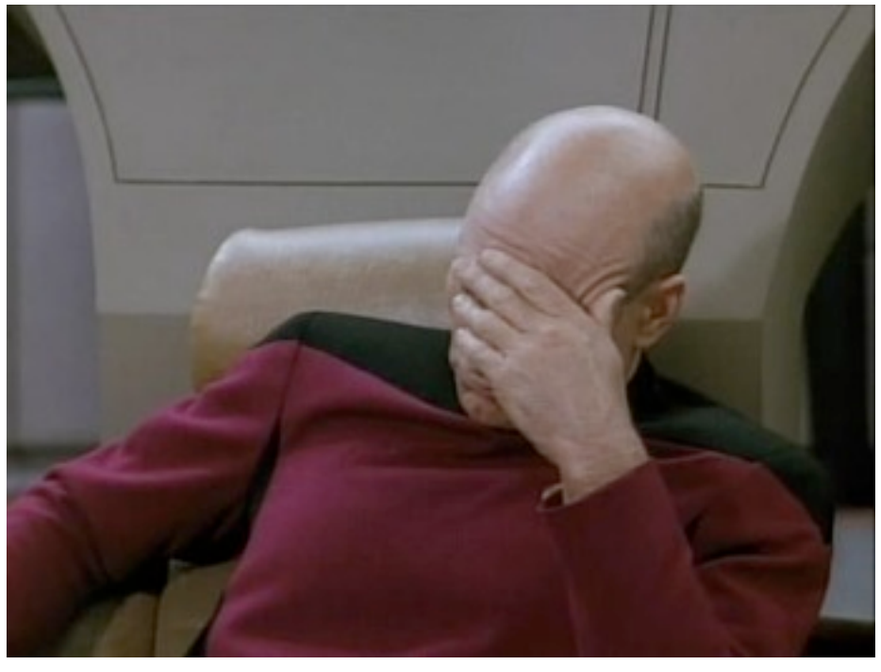
- Messaging:
  - pySage
  - python-spread
  - XMPP
  - Protocol Buffers
- RPC:
  - Pyro
  - rPyc
  - Thrift

Shared Memory/Message Qs

- Shared Memory
  - Posh (**dead**)
  - Memcached
  - posix_ipc
- Message Queues
  - Apache ActiveMQ
  - RabbitMQ
  - Stomp
  - MemcacheQ
  - Beanstalkd

So...

- Where the hell do we point new users?
- While *good*, Twisted and Kamaelia have a documentation problem
- The rest is a mish-mash of technologies
- Concurrency is hard let's go shopping!



Where does this leave us?

- The GIL is here for the foreseeable future
  - Not entirely a bad thing (extensions!)
- Python-Core is **not** the right place for much of this, but can provide some basics
  - Actor implementation
  - Java.util.concurrent-like abstractions
- Anything going in must make this work **safe**

Where does this leave us?

- Lots of great community work
- Continued room for growth, adoption of other language's technologies
- If we can build a stack of reusable, swappable components for all three areas: **everyone wins**
- Anyone for a "distributed Django"?
  - "loose coupling and tight cohesion"
  - Must take the fallacies into account

Django?

- The point of a framework is to make the easy things easy, and the hard things easier
- The abstractions must be **leaky**
  - **Go see abstractions as leverage!**
- It must be **safe**
- It can not ignore the fallacies
- I shall call it Mustaine (Megadeth)

Questions?


Fin.