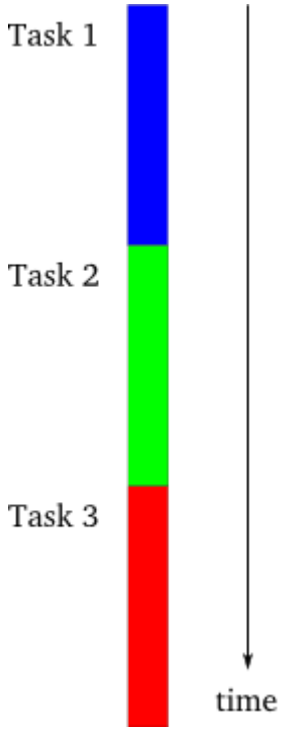
Asynchronous Programming with Twisted

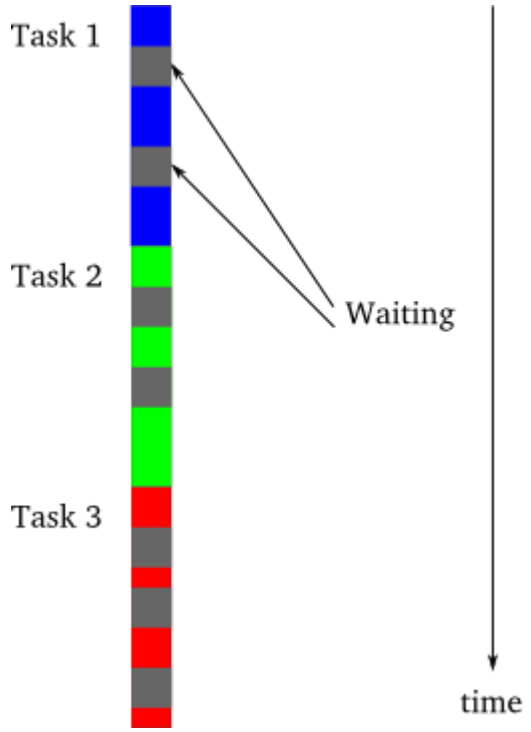
Single Threaded Synchronous Model

- Doing things one at a time, in order.
- Simple to think about and plan.
- Later tasks can assume an earlier task finished.



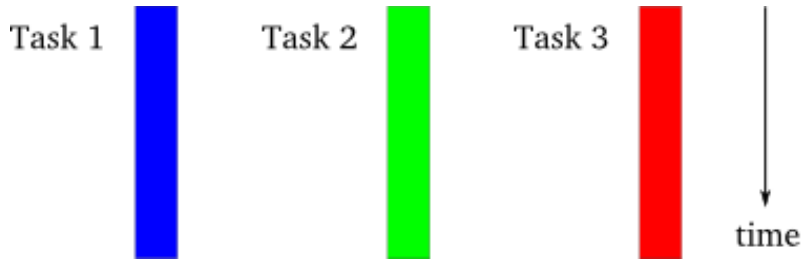
What's wrong with Synchronous?

- The synchronous model is nice and simple so why do anything else?
- If the tasks are non blocking and each task depends on the previous task then **synchronous is best**
- If tasks are largely independent and can be done out of order than synchronous is terrible.
- Tasks that must wait on IO / network / a human being will waste a *lot* of cpu cycles and running time!



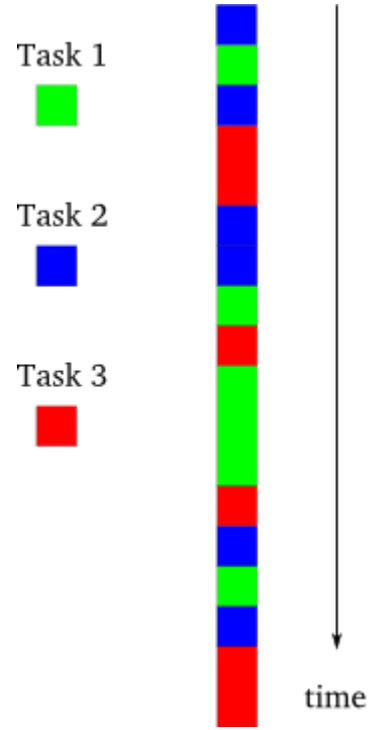
Multi Threaded / Process Model

- Doing tasks in parallel.
- Threads cannot guarantee another task has finished
- More complex to organise
- Communications between threads can be nasty!
- Risk of thread / process lock / deadlocks and a lot of time spent doing busy *nothing*
- OS owns control of thread execution
- Threads can be resource intensive depending on your language and architecture.



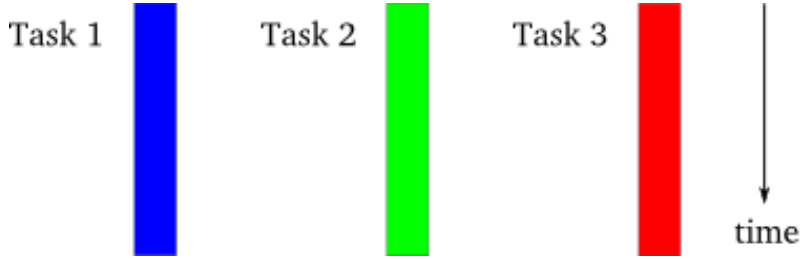
The Asynchronous Model

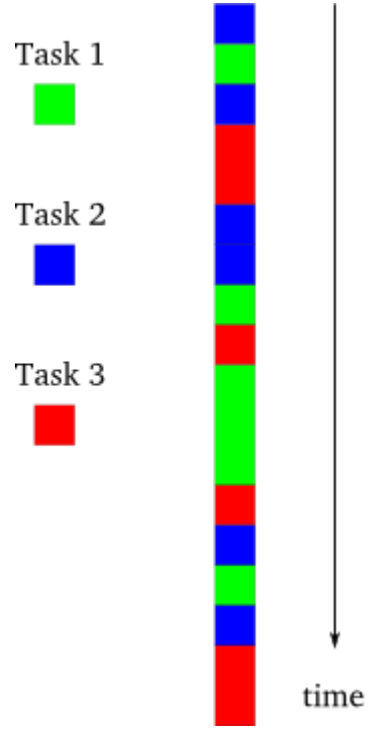
- Interleaving Tasks in the same **single** Thread / Process
- You guarantee that when a task is executing another task **is not.**
- No risk of resource locks / deadlocks on shared resources
- Decision to suspend and start tasks is under the programmer's control.
- Cannot make use of multiple CPU's
- Tasks yield control voluntarily.
- Not a good fit for high CPU workloads



Multi thread vs Asynchronous Model

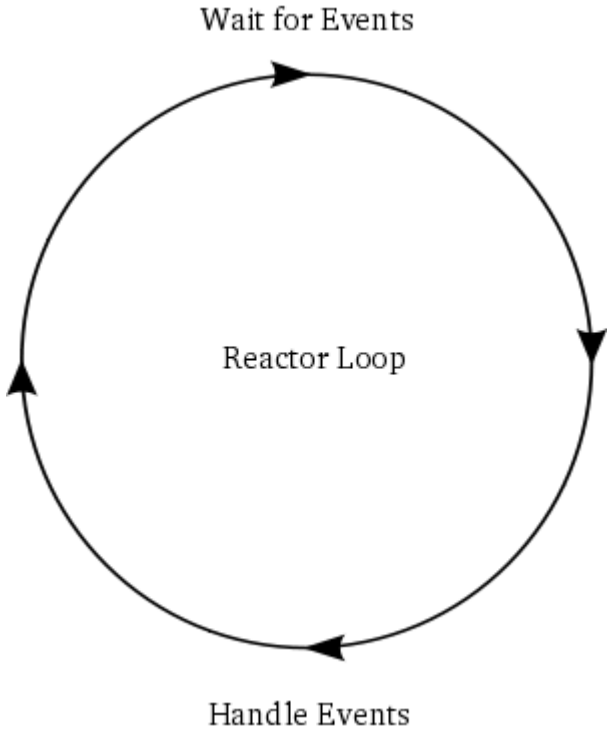
- Threads have overheads!
- Threads are under OS control
- Threads can block each other and contend for the same resources. Asynchronous tasks cannot do this.
- Managing state across threads is a PITA
- When tasks are dominated by IO wait / blocking then a asynch single threaded model can get amazing results compared to multi-thread





The Reactor / Select loop

- A loop that listens and **reacts** to events
- Informs you repeatedly when events are ready
- *Ideally – abstracts away the underlying horror of networking / socket IO on different systems*
- *Ideally – provides protocols for you to use.*
- **Twisted is a robust cross platform implementation of the reactor pattern with lots of extra stuff thrown in**

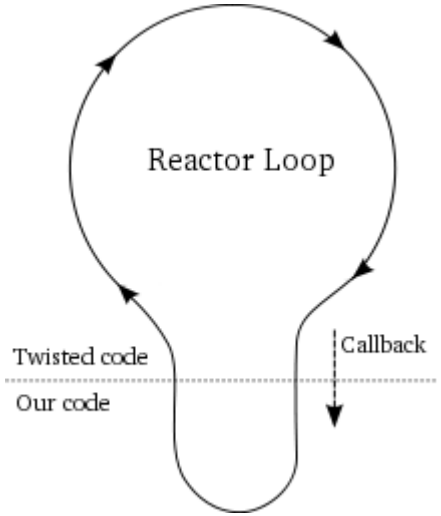


Python Twisted

- Twisted is an event-driven networking engine written in Python and licensed under the open source MIT license.
- Twisted is very mature with lots of functionality out of the box
- Lots of community libraries – database connectors, network clients, etc
- The reactor in twisted is a **singleton** if you import the reactor in a module you will get the same instance as elsewhere in your code.
- Twisted calls code using a **callback** mechanism
- Twisted official documentation can be quite challenging to read.
- In general: Twisted can be a bit of a brain ****

Callbacks in Twisted

- Callbacks are triggered by events determined by the programmer
  - Data ready on a socket for example
- The reactor must call our code to handle that event
- Callbacks are passed the value from the event as an input – whatever that is.
- When our code is running the reactor **is not running**
  - Our code must not block!
- The reactor loop resumes when our code has finished handling the event



Twisted Callback Example

```
class Countdown(object):

    def __init__(self, counter):
        self.counter = counter

    def decrement_count(self, decrement):
        if self.counter < 1:
            reactor.stop()
        else:
            print self.counter
            self.counter -= decrement
            reactor.callLater(1, self.decrement_count, decrement)

def main():
    from twisted.internet import reactor
    c = Countdown(10)
    reactor.callWhenRunning(c.decrement_count, 1)

    print "Reactor Starting..."
    reactor.run()
    print "Reactor Stopped!"

if __name__ == "__main__":
    main()
```

```
Reactor Starting...
10
9
8
7
6
5
4
3
2
1
Reactor Stopped!
```

Introducing Deferreds

- What do we do if our callback needs to block?
  - Maybe it needs to make an IO call based on information if got previously?
- We can't do it in our code or that would block the whole reactor!
- We'd like to define a chain of connected callbacks that will run, in order, when data becomes available from a previous step.
- Twisted calls these **deferreds**
- Think of deferreds as a list or **chain** of callbacks that will be executed, in order, when the time is right.
- Another common name is **The Callback Chain**
- Each callback passes the result of it's task into the next step of the chain as an input.

Deferred Example

```
from twisted.internet.defer import Deferred

def to_upper(s):
    return s.upper()

def print_foo(foo):
    print foo
    return foo

def main():
    d = Deferred()

    d.addCallback(print_foo)
    d.addCallback(to_upper)
    d.addCallback(print_foo)

    # fire the chain with a normal result
    d.callback("yo, uppercase me!")

    print "Finished"

if __name__ == "__main__":
    main()
```

```
yo, uppercase me!
YO, UPPERCASE ME!
Finished
```

Introducing Errbacks

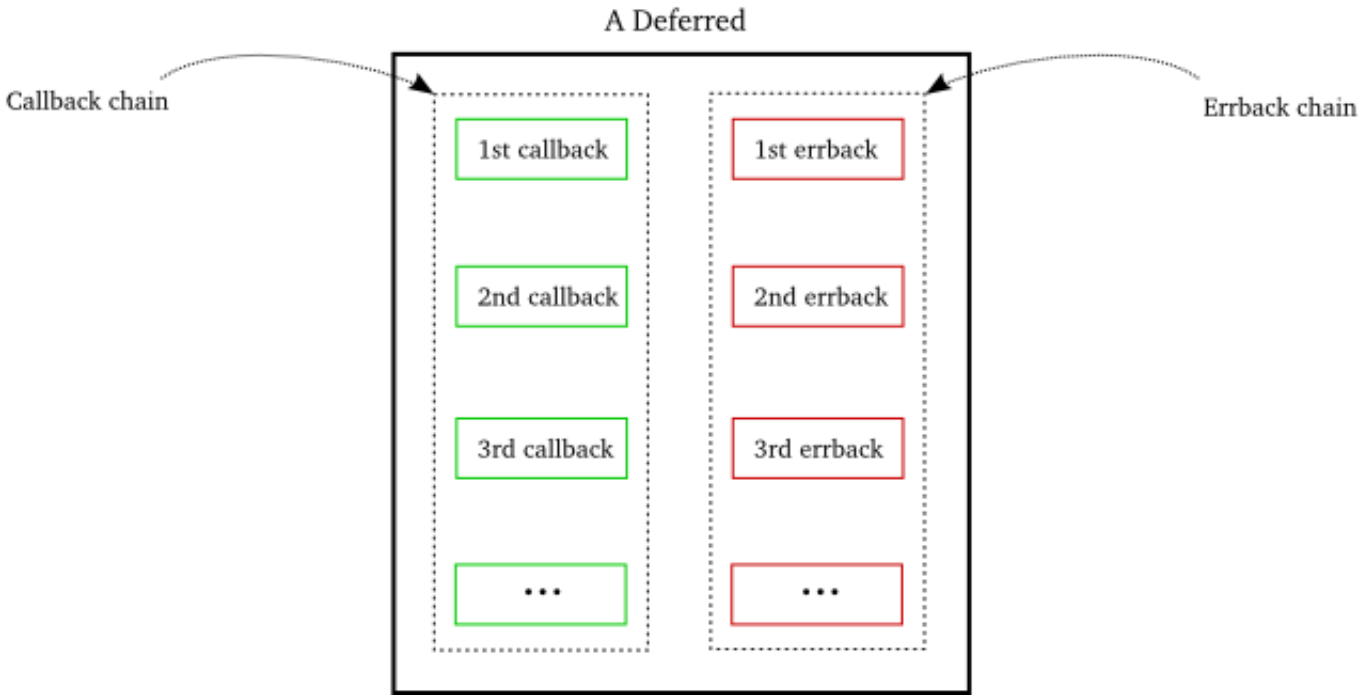
- IF SOMETHING CAN GO WRONG, SOMETHING WILL GO WRONG
- So what happens if one our callbacks errors (throws an exception or fails to return a result?)
- **Callbacks** further down the chain will **not** be called.
- **Twisted will continue running if we hit an exception and log it.**
  - You probably don't want a server to die just because you hit an untrapped exception!
- The deferred chain will look for an **Errback to call** with the value returned by the failure
- Twisted will pass a **Failure** instance (think of this as a wrapped exception)
- If the **ERRBACK** returns a result it will be passed to the next **CALLBACK** in the chain.
- If the **ERRBACK** throws an error (returns a Failure) the next **ERRBACK** in the chain will be called.

Errback Example

```
from twisted.internet.defer import Deferred

def handle_error(error): return "CAUGHT: {e}".format(e = error.getErrorMessage())

def to_upper(s): return s.upper()

def print_foo(foo):
    print foo
    return foo

def main():
    d = Deferred()

    # Add a callback and an errback to fire if something went wrong
    d.addCallback(to_upper)
    d.addErrback(handle_error)

    d.addCallback(print_foo)

    # fire the chain with a integer rather than a string
    d.callback(1)

if __name__ == "__main__":
    main()
```

CAUGHT: 'int' object has no attribute 'upper'

Deferred Callback / Errback Chain



Insult Server & Client

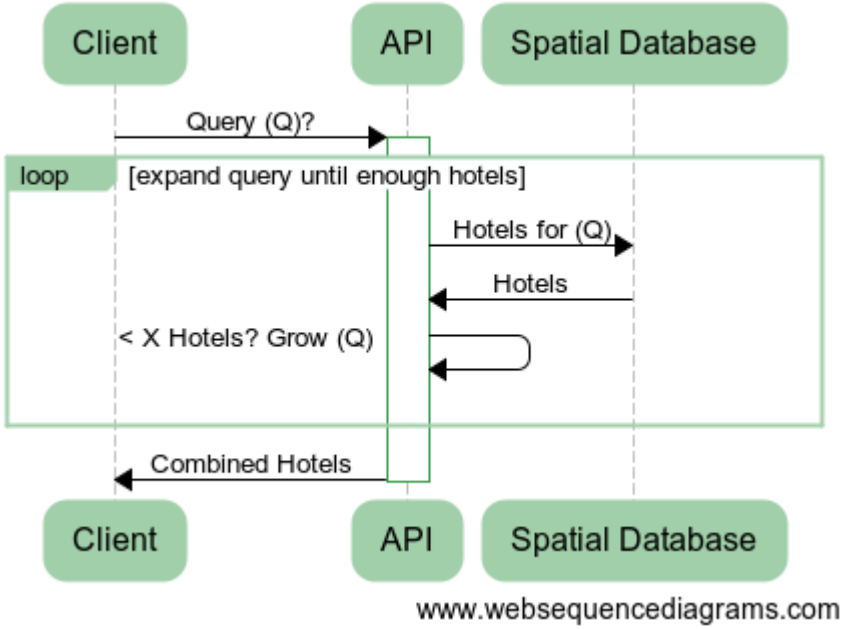
- Code at: https://github.com/TransactCharlie/twisted-intro.git
- Server is pretending to be a slow HTTP API that can take up to 5 seconds to return a random insult
- Client makes 500 simultaneous connections and combines the results
- Both use raw twisted using the provided twisted HTTP server / client objects

Spaghetti Code / Callback Hell

- You can **very, very easily** write incredibly spaghetti code with twisted.
- It can be very hard to understand what is happening in a twisted program
- Especially if that program was written by someone else
- Passing a deferred around effectively means you are returning a mystery pipe – at some point this pipe will have a value in it.
- You should try **very hard** to make sure that value is defined up front!
- Twisted provides some nice abstractions – you should make sure you use them rather than writing your own versions of these
- Use Additional libraries – such as Klein / txRequests to make your life easier.

Twisted at Skyscanner

- Query Expansion of Spatial Search



References

- Original white paper on reactor pattern:
  - http://www.dre.vanderbilt.edu/~schmidt/PDF/reactor-siemens.pdf
- Most awesome introduction to twisted I've ever found:
  - http://krondo.com/blog/?page_id=1327
  - Thanks to Dave Peticolas for the awesome help his blog was.
- The maintainers of Twisted
  - http://twistedmatrix.com/trac/wiki/Documentation