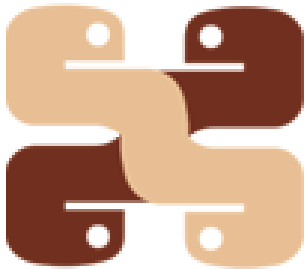
The Story of Stackless Python

Christian Tismer, Hervé Coatanhay

EuroPython 2012

July 4 2012



What is Stackless about?

- it is like CPython
- it can do a little bit more
- adds a single module

```
import stackless
```

- is like an extension
  - but, sadly, not really
  - **but:** there is a solution...

What is Stackless about?

- it is like CPython
- it can do a little bit more
- adds a single module

```
import stackless
```

- is like an extension
  - but, sadly, not really
  - **but:** there is a solution...

What is Stackless about?

- it is like CPython
- it can do a little bit more
- adds a single module

```
import stackless
```

- is like an extension
  - but, sadly, not really
  - **but:** there is a solution...

What is Stackless about?

- it is like CPython
- it can do a little bit more
- adds a single module

```
import stackless
```

- is like an extension
  - but, sadly, not really
  - **but:** there is a solution...

What is Stackless about?

- it is like CPython
- it can do a little bit more
- adds a single module

```
import stackless
```

- is like an extension
  - but, sadly, not really
  - **but:** there is a solution...

Now, what is it really about?

- have tiny little “main” programs
  - `tasklet`
- tasklets communicate via messages
  - `channel`
- tasklets are often called `microthreads`
  - but there are no threads at all
  - only one tasklets runs at any time
- *but see the PyPy STM* approach
  - this will apply to tasklets as well

Now, what is it really about?

- have tiny little “main” programs
  - `tasklet`
- tasklets communicate via messages
  - `channel`
- tasklets are often called `microthreads`
  - but there are no threads at all
  - only one tasklets runs at any time
- *but see the PyPy STM* approach
  - this will apply to tasklets as well

Now, what is it really about?

- have tiny little “main” programs
  - `tasklet`
- tasklets communicate via messages
  - `channel`
- tasklets are often called `microthreads`
  - but there are no threads at all
  - only one tasklets runs at any time
- *but see the PyPy STM* approach
  - this will apply to tasklets as well

Now, what is it really about?

- have tiny little “main” programs
  - `tasklet`
- tasklets communicate via messages
  - `channel`
- tasklets are often called `microthreads`
  - but there are no threads at all
  - only one tasklets runs at any time
- *but see the PyPy STM* approach
  - this will apply to tasklets as well

Cooperative Multitasking ...

```
>>> import stackless
>>>
>>> channel = stackless.channel()

>>> def receiving_tasklet():
...     print "Receiving tasklet started"
...     print channel.receive()
...     print "Receiving tasklet finished"
...

>>> def sending_tasklet():
...     print "Sending tasklet started"
...     channel.send("send from sending_tasklet")
...     print "sending tasklet finished"
...
```

Cooperative Multitasking ...

```
>>> import stackless
>>>
>>> channel = stackless.channel()

>>> def receiving_tasklet():
...     print "Receiving tasklet started"
...     print channel.receive()
...     print "Receiving tasklet finished"
...

>>> def sending_tasklet():
...     print "Sending tasklet started"
...     channel.send("send from sending_tasklet")
...     print "sending tasklet finished"
...
```

Cooperative Multitasking ...

```
>>> import stackless
>>>
>>> channel = stackless.channel()

>>> def receiving_tasklet():
...     print "Receiving tasklet started"
...     print channel.receive()
...     print "Receiving tasklet finished"
...

>>> def sending_tasklet():
...     print "Sending tasklet started"
...     channel.send("send from sending_tasklet")
...     print "sending tasklet finished"
...
```

Cooperative Multitasking ...

```
>>> def another_tasklet():
...     print "Just another tasklet in the scheduler"
...

>>> stackless.tasklet(receiving_tasklet)()
<stackless.tasklet object at 0x00A45B30>
>>> stackless.tasklet(sending_tasklet)()
<stackless.tasklet object at 0x00A45B70>
>>> stackless.tasklet(another_tasklet)()
<stackless.tasklet object at 0x00A45BF0>
```

Cooperative Multitasking ...

```
>>> def another_tasklet():
...     print "Just another tasklet in the scheduler"
...

>>> stackless.tasklet(receiving_tasklet)()
<stackless.tasklet object at 0x00A45B30>
>>> stackless.tasklet(sending_tasklet)()
<stackless.tasklet object at 0x00A45B70>
>>> stackless.tasklet(another_tasklet)()
<stackless.tasklet object at 0x00A45BF0>
```

... Cooperative Multitasking

```
<stackless.tasklet object at 0x00A45B70>
>>> stackless.tasklet(another_tasklet)()
<stackless.tasklet object at 0x00A45BF0>
>>>
>>> stackless.run()
Receiving tasklet started
Sending tasklet started
send from sending_tasklet
Receiving tasklet finished
Just another tasklet in the scheduler
sending tasklet finished
```

Why not just the greenlet ?

- greenlets are a subset of stackless
  - there is no scheduler
  - can emulate stackless
- greenlets are about 5-10x slower to switch using only hard-switching
- but the main difference is ...

Why not just the greenlet ?

- greenlets are a subset of stackless
  - there is no scheduler
  - can emulate stackless
- greenlets are about 5-10x slower to switch using only hard-switching
- but the main difference is ...

Why not just the greenlet ?

- greenlets are a subset of stackless
  - there is no scheduler
  - can emulate stackless
- greenlets are about 5-10x slower to switch using only hard-switching
- but the main difference is ...

Pickling Program State

Example (p. 1 of 2)

```
import pickle, sys
import stackless

ch = stackless.channel()

def recurs(depth, level=1):
    print 'enter level %s%d' % (level*'  ', level)
    if level >= depth:
        ch.send('hi')
    if level < depth:
        recurs(depth, level+1)
    print 'leave level %s%d' % (level*'  ', level)
```

Pickling Program State

Example (p. 2 of 2)

```
def demo(depth):
    t = stackless.tasklet(recurs)(depth)
    print ch.receive()
    pickle.dump(t, file('tasklet.pickle', 'wb'))

if __name__ == '__main__':
    if len(sys.argv) > 1:
        t = pickle.load(file(sys.argv[1], 'rb'))
        t.insert()
    else:
        t = stackless.tasklet(demo)(9)
    stackless.run()

# remember to show it interactively
```

Software archeology

- Around since 1998
  - version 1
    - using only soft-switching
    - continuation-based
    - *please let me skip old design errors :-)*
- Complete redesign in 2002
  - version 2
    - using only hard-switching
    - birth of tasklets and channels
- Concept merge in 2004
  - version 3
    - **80-20** rule:
    - soft-switching whenever possible
    - hard-switching if foreign code is on the stack
  - these 80 % can be *pickled*

Thank you

- `http://pypy.org/`
- You can hire Antonio
- Questions?