# The Story of Stackless Python

Christian Tismer, Hervé Coatanhay

EuroPython 2012

July 4 2012

## About This Talk

- first talk after a long break
  - *rst2beamer* for the first time

guest speaker:

- Herve Coatanhay about Nagare
  - PowerPoint (Mac)

Meanwhile I used

- Powerpoint (PC)
- Keynote (Mac)
- Google Docs

poll: What is your favorite slide tool?

## About This Talk

- first talk after a long break
  - *rst2beamer* for the first time

guest speaker:

- Herve Coatanhay about Nagare
  - PowerPoint (Mac)

Meanwhile I used

- Powerpoint (PC)
- Keynote (Mac)
- Google Docs

poll: What is your favorite slide tool?

## About This Talk

- first talk after a long break
  - *rst2beamer* for the first time

guest speaker:

- Herve Coatanhay about Nagare
  - PowerPoint (Mac)

Meanwhile I used

- Powerpoint (PC)
- Keynote (Mac)
- Google Docs

poll: What is your favorite slide tool?

## What is Stackless?

- *Stackless is a Python version that does not use the C stack*
  - really? naah
- Stackless is a Python version that does not keep state on the C stack
  - the stack *is* used but
  - cleared between function calls
- Remark:
  - theoretically. In practice...
  - ... it is reasonable 90 % of the time
  - we come back to this!

## What is Stackless?

- *Stackless is a Python version that does not use the C stack*
  - really? naah
- Stackless is a Python version that does not keep state on the C stack
  - the stack *is* used but
  - cleared between function calls
- Remark:
  - theoretically. In practice...
  - ... it is reasonable 90 % of the time
  - we come back to this!

## What is Stackless?

- *Stackless is a Python version that does not use the C stack*
  - really? naah
- Stackless is a Python version that does not keep state on the C stack
  - the stack *is* used but
  - cleared between function calls
- Remark:
  - theoretically. In practice...
  - ... it is reasonable 90 % of the time
  - we come back to this!

## What is Stackless?

- *Stackless is a Python version that does not use the C stack*
  - really? naah
- Stackless is a Python version that does not keep state on the C stack
  - the stack *is* used but
  - cleared between function calls
- Remark:
  - theoretically. In practice...
  - ... it is reasonable 90 % of the time
  - we come back to this!

## What is Stackless about?

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

```
import stackless
```

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

## What is Stackless about?

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

```
import stackless
```

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

## What is Stackless about?

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

```
import stackless
```

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

## What is Stackless about?

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

```
import stackless
```

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

## What is Stackless about?

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

```
import stackless
```

- is like an extension
    - but, sadly, not really
    - stackless **must** be builtin
    - **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
  - can partially emulate stackless
  - there is no builtin scheduler
  - technology quite close to Stackless 2.0
- greenlets are about 10x slower to switch context because using only hard-switching
  - but that's ok in most cases
- greenlets are kind-of perfect
  - near zero maintenace
  - minimal interface
- but the main difference is ...

## Why not just the greenlet ?

- greenlets are a subset of stackless
  - can partially emulate stackless
  - there is no builtin scheduler
  - technology quite close to Stackless 2.0
- greenlets are about 10x slower to switch context because using only hard-switching
  - but that's ok in most cases
- greenlets are kind-of perfect
  - near zero maintenace
  - minimal interface
- but the main difference is ...

## Why not just the greenlet ?

- greenlets are a subset of stackless
  - can partially emulate stackless
  - there is no builtin scheduler
  - technology quite close to Stackless 2.0
- greenlets are about 10x slower to switch context because using only hard-switching
  - but that’s ok in most cases
- greenlets are kind-of perfect
  - near zero maintenace
  - minimal interface
- but the main difference is ...

## Why not just the greenlet ?

- greenlets are a subset of stackless
    - can partially emulate stackless
    - there is no builtin scheduler
    - technology quite close to Stackless 2.0
- greenlets are about 10x slower to switch context because using only hard-switching
    - but that’s ok in most cases
- greenlets are kind-of perfect
    - near zero maintenace
    - minimal interface
- but the main difference is ...

## Excurs: Hard-Switching

Sorry ;-)

Switching program state “the hard way”:

Without notice of the interpreter

- the machine stack gets hijacked
    - Brute-Force: replace the stack with another one
    - like threads
- stackless, greenlets
    - stack slicing
    - semantically same effect
- switching works fine
- pickling does not work, opaque data on the stack
    - this is more sophisticated in PyPy, another story...

## Excurs: Soft-Switching

Switching program state “the soft way”:
With knowledge of the interpreter

- most efficient implementation in Stackless 3.1
- demands the most effort of the developers
- no opaque data on the stack, pickling does work
  - again, this is more sophisticated in PyPy
- now we are at the main difference, as you guessed ...

## Excurs: Soft-Switching

Switching program state “the soft way”:
With knowledge of the interpreter

- most efficient implementation in Stackless 3.1
- demands the most effort of the developers
- no opaque data on the stack, pickling does work
  - again, this is more sophisticated in PyPy
- now we are at the main difference, as you guessed ...

## Pickling Program State

### Persistence (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)
```

*# remember to show it interactively*

## Pickling Program State

### Persistence (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*

# Script Output 1

```
$ ~/src/stackless/python.exe demo/pickledtasklet.py
enter level   1
enter level     2
enter level       3
enter level         4
enter level           5
enter level             6
enter level               7
enter level                 8
enter level                   9
hi
leave level                   9
leave level                 8
leave level               7
leave level             6
leave level           5
leave level         4
leave level       3
leave level     2
leave level   1
```

# Script Output 2

```
$ ~/src/stackless/python.exe demo/pickledtasklet.py tasklet.pickle
leave level                    9
leave level                  8
leave level                7
leave level              6
leave level            5
leave level          4
leave level        3
leave level      2
leave level   1
```

## Greenlet vs. Stackless

- Greenlet is a pure extension module
    - but performance is good enough
- Stackless can pickle program state
    - but stays a replacement of Python
- Greenlet never can, as an extension
- *easy installation* lets people select greenlet over stackless
    - see for example the *eventlet* project
    - *but there is a simple work-around, we'll come to it*
- *they both have their application domains and they will persist.*

## Greenlet vs. Stackless

- Greenlet is a pure extension module
  - but performance is good enough
- Stackless can pickle program state
  - but stays a replacement of Python
- Greenlet never can, as an extension
- *easy installation* lets people select greenlet over stackless
  - see for example the *eventlet* project
  - *but there is a simple work-around, we'll come to it*
- *they both have their application domains and they will persist.*

## Greenlet vs. Stackless

- Greenlet is a pure extension module
    - but performance is good enough
- Stackless can pickle program state
    - but stays a replacement of Python
- Greenlet never can, as an extension
- *easy installation* lets people select greenlet over stackless
    - see for example the *eventlet* project
    - *but there is a simple work-around, we'll come to it*
- *they both have their application domains and they will persist.*

## Greenlet vs. Stackless

- Greenlet is a pure extension module
  - but performance is good enough
- Stackless can pickle program state
  - but stays a replacement of Python
- Greenlet never can, as an extension
- *easy installation* lets people select greenlet over stackless
  - see for example the *eventlet* project
  - *but there is a simple work-around, we'll come to it*
- *they both have their application domains and they will persist.*

## Greenlet vs. Stackless

- Greenlet is a pure extension module
    - but performance is good enough
- Stackless can pickle program state
    - but stays a replacement of Python
- Greenlet never can, as an extension
- *easy installation* lets people select greenlet over stackless
    - see for example the *eventlet* project
    - *but there is a simple work-around, we'll come to it*
- *they both have their application domains and they will persist.*

## Why Stackless makes a Difference

- Microthreads ?
  - the feature where I put most effort into
  - can be emulated: (in decreasing speed order)
    - generators (incomplete, “half-sided”)
    - greenlet
    - threads (even ;-)
- Pickling program state ! ==
- **persistence**

## Why Stackless makes a Difference

- Microthreads ?
  - the feature where I put most effort into
  - can be emulated: (in decreasing speed order)
    - generators (incomplete, “half-sided”)
    - greenlet
    - threads (even ;-)
- Pickling program state ! ==
- **persistence**

## Why Stackless makes a Difference

- Microthreads ?
  - the feature where I put most effort into
  - can be emulated: (in decreasing speed order)
    - generators (incomplete, “half-sided”)
    - greenlet
    - threads (even ;-)
- Pickling program state ! ==
- **persistence**

## Why Stackless makes a Difference

- Microthreads ?
  - the feature where I put most effort into
  - can be emulated: (in decreasing speed order)
    - ★ generators (incomplete, “half-sided”)
    - ★ greenlet
    - ★ threads (even ;-)
- Pickling program state ! ==
- **persistence**

## Persistence, Cloud Computing

- freeze your running program
- let it continue anywhere else
    - on a different computer
    - on a different operating system (!)
    - in a cloud
- migrate your running program
- save snapshots, have checkpoints
    - without doing any extra-work

## 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* (90?)
- This stayed as version 3.1

# 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* (90?)
- This stayed as version 3.1

## 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* (90?)
- This stayed as version 3.1

## Status of Stackless Python

- mature
- Python 2 and Python 3, all versions
- maintained by
  - Richard Tew
  - Kristjan Valur Jonsson
  - me (a bit)

# The New Direction for Stackless

- `pip install stackless-python`
  - will install `slpython`
  - or even `python` (opinions?)
- drop-in replacement of CPython *(psssst)*
- `pip uninstall stackless-python`
  - Stackless is a bit cheating, as it replaces the python binary
  - but the user perception will be perfect
- *trying stackless made easy!*

## The New Direction for Stackless

- `pip install stackless-python`
  - will install `slpython`
  - or even `python` (opinions?)
- drop-in replacement of CPython *(psssst)*
- `pip uninstall stackless-python`
  - Stackless is a bit cheating, as it replaces the python binary
  - but the user perception will be perfect
- *trying stackless made easy!*

## The New Direction for Stackless

- `pip install stackless-python`
  - will install `slpython`
  - or even `python` (opinions?)
- drop-in replacement of CPython *(psssst)*
- `pip uninstall stackless-python`
  - Stackless is a bit cheating, as it replaces the python binary
  - but the user perception will be perfect
- *trying stackless made easy!*

## New Direction (cont'd)

- first prototype yesterday from Anselm Kruis *(applause)*
  - works on Windows
  - OS X
    - I'll do that one
  - Linux
    - soon as well
- being very careful to stay compatible
  - python 2.7.3 installs stackless for 2.7.3
  - python 3.2.3 installs stackless for 3.2.3
  - python 2.7.2 : *please upgrade* - or maybe have an over-ride option?

## New Direction (cont'd)

- first prototype yesterday from Anselm Kruis *(applause)*
  - works on Windows
  - OS X
    - I'll do that one
  - Linux
    - soon as well
- being very careful to stay compatible
  - python 2.7.3 installs stackless for 2.7.3
  - python 3.2.3 installs stackless for 3.2.3
  - python 2.7.2 : *please upgrade* - or maybe have an over-ride option?

# New Direction (cont'd)

- first prototype yesterday from Anselm Kruis *(applause)*
  - works on Windows
  - OS X
    - I'll do that one
  - Linux
    - soon as well
- being very careful to stay compatible
  - python 2.7.3 installs stackless for 2.7.3
  - python 3.2.3 installs stackless for 3.2.3
  - python 2.7.2 : *please upgrade* - or maybe have an over-ride option?

## New Direction (cont'd)

- first prototype yesterday from Anselm Kruis *(applause)*
  - works on Windows
  - OS X
    - I'll do that one
  - Linux
    - soon as well
- being very careful to stay compatible
  - python 2.7.3 installs stackless for 2.7.3
  - python 3.2.3 installs stackless for 3.2.3
  - python 2.7.2 : *please upgrade* - or maybe have an over-ride option?

## Consequences of the Pseudo-Package

The technical effect is almost nothing.
The psycological impact is probably huge:

- stackless is easy to install and uninstall
- people can simply try if it fits their needs
- the never ending discussion
  - "Why is Stackless not included in the Python core?"
- **has ended**
  - "Why should we, after all?"
  - hey Guido :-)
  - what a relief, for you and me

## Consequences of the Pseudo-Package

The technical effect is almost nothing.
The psycological impact is probably huge:

- stackless is easy to install and uninstall
- people can simply try if it fits their needs
- the never ending discussion
  - “Why is Stackless not included in the Python core?”
- **has ended**
  - “Why should we, after all?”
  - hey Guido :-)
  - what a relief, for you and me

## Consequences of the Pseudo-Package

The technical effect is almost nothing.
The psycological impact is probably huge:

- stackless is easy to install and uninstall
- people can simply try if it fits their needs
- the never ending discussion
  - “Why is Stackless not included in the Python core?”
- **has ended**
  - “Why should we, after all?”
  - hey Guido :-)
  - what a relief, for you and me

## Consequences of the Pseudo-Package

The technical effect is almost nothing.
The psycological impact is probably huge:

- stackless is easy to install and uninstall
- people can simply try if it fits their needs
- the never ending discussion
  - “Why is Stackless not included in the Python core?”
- **has ended**
  - “Why should we, after all?”
  - hey Guido :-)
  - what a relief, for you and me

## Consequences of the Pseudo-Package

The technical effect is almost nothing.
The psycological impact is probably huge:

- stackless is easy to install and uninstall
- people can simply try if it fits their needs
- the never ending discussion
  - “Why is Stackless not included in the Python core?”
- **has ended**
  - “Why should we, after all?”
  - hey Guido :-)
  - what a relief, for you and me

## Consequences of the Pseudo-Package

The technical effect is almost nothing.
The psycological impact is probably huge:

- stackless is easy to install and uninstall
- people can simply try if it fits their needs
- the never ending discussion
  - “Why is Stackless not included in the Python core?”
- **has ended**
  - “Why should we, after all?”
  - hey Guido :-)
  - what a relief, for you and me

## Status of Stackless PyPy

- was completely implemented before the Jit
    - together with greenlets coroutines
    - not Jit compatible
- was “too complete” with a 30% performance hit
- new approach is almost ready
    - with full Jit support
    - but needs some fixing
    - this *will* be efficient

## Applications using Stackless Python

- The Eve Online MMORPG
  `http://www.eveonline.com/`
  - based their games on Stackless since 1998
- science + computing ag, Anselm Kruis
  `https://ep2012.europython.eu/conference/p/anselm-kruis`
- The Nagare Web Framework
  `http://www.nagare.org/`
  - works because of Stackless Pickling
- today's majority: persistence

## Thank you

- the new Stackless Website
  `http://www.stackless.com/`
  - a **great** donation from Alain Pourier, *Nagare*
- You can hire me as a consultant
- Questions?