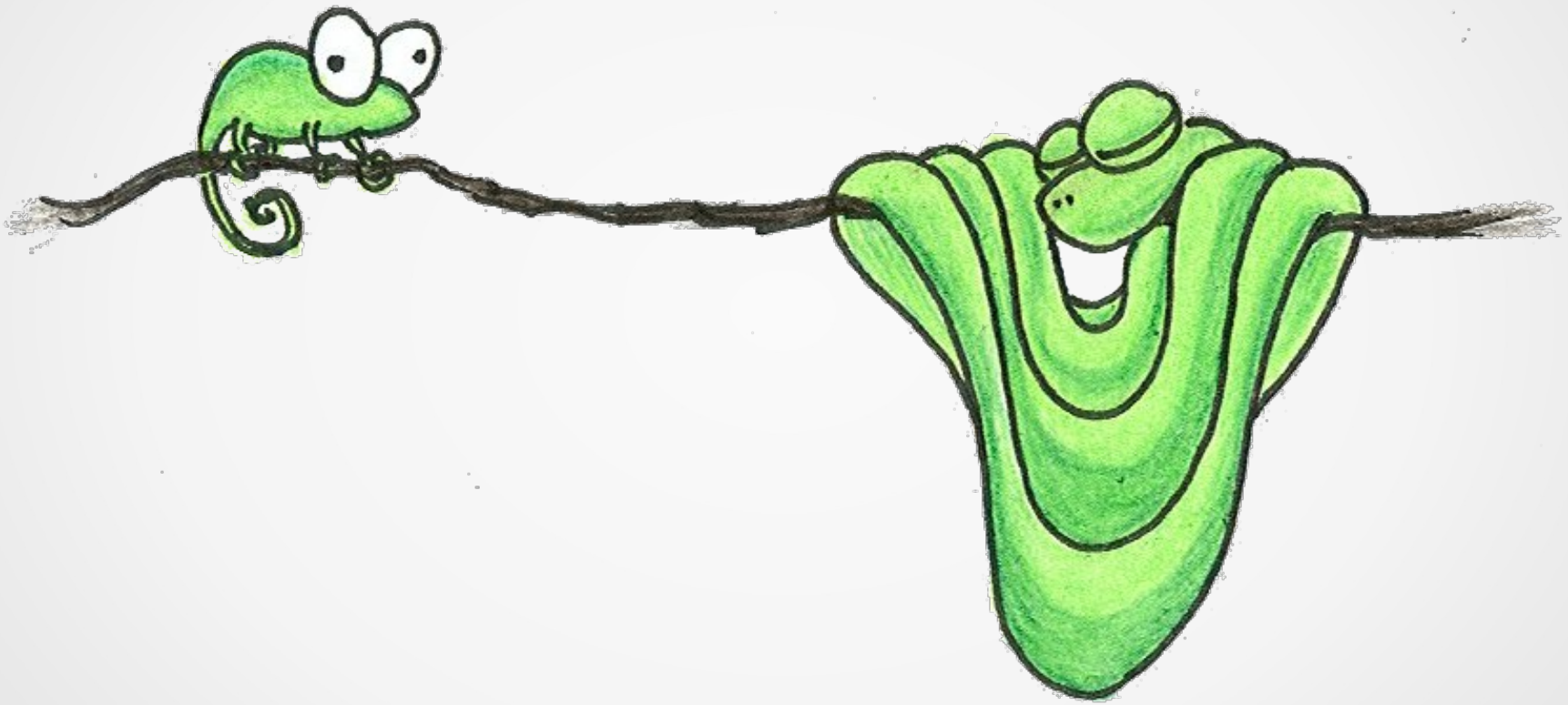
Sleepy



Sleepy – intro+historia

alejandro david weil

a.k.a.: Tenuki / dave 

Sleepy

- Trabajo (‘99-): CORE SECURITY Thinking ahead.
- Algunos proyectos personales:
  - IMFish
  - Juegos varios p/PyWeek..
- ¿?

- Nació en Core como idea de respuesta a la pregunta:

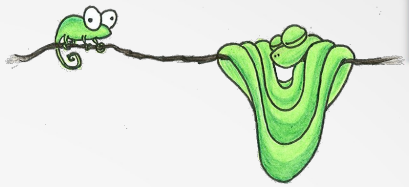
¿cómo correr un código, sin modificarlo, en Google App Engine?

- En 2009 Sleepy v1 con **Fernando Russ**
- Ahora proponemos algunas soluciones a problemas..

¿Por / para qué suspender python?

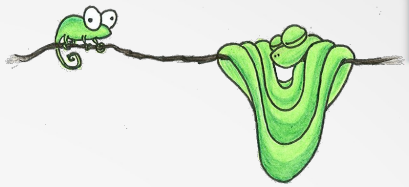
- Poder ejecutar código con un límite de tiempo (por ej.: google app engine)
- Poder suspender un proceso y reanudarlo en otro momento
- Poder reanudar un proceso en otra máquina
  - Distribución de carga
  - Alta disponibilidad
- Poder suspender un proceso para inspeccionarlo
- “continuations”

Soluciones intermedias u otro nivel



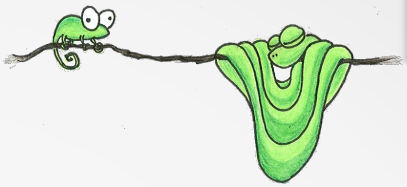
- A nivel S.O. en unix:
  - Ctrl+Z es una solución a algunas de las necesidades presentadas
  - CryoPID *“allows you to capture the state of a running process in Linux and save it to a file”*
- A nivel V.M. en python, seria posible en un momento determinado, recorrer toda la memoria y guardar el estado de todos los objetos y reconstruir el stack de cada thread.
- Pero buscamos una solución “`pure-python`”

¿Es posible en otros lenguajes?



- Smalltalk → image
- Lisp → worlds
- C → Solución semejante a CryoPID:
  - Salvar toda la memoria (heap, stack, código, ..)
  - Salvar estado del cpu:
    - registros
    - program-counter “`pc`”
  - Restaurar memoria
  - Restaurar cpu
- Python? No.. sí.. mas o menos..

¿Qué pasa en Python?



- CPython: no... **:-(**
- Jython: no idea.
- IronPython: no idea.
- Stackless: sí*!
  - "*One of the main features of Stackless is its ability to pickle and unpickle tasklets*"
- PyPy: sí ~ work in progress**:
  - `_continuations: continulets, genlets`

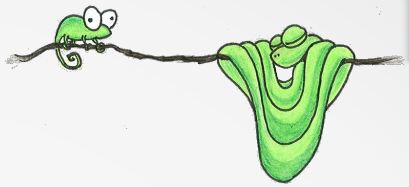
* http://www.stackless.com/wiki/Pickling

** http://doc.pypy.org/en/latest/stackless.html

Sleepy – CPython - idea

- Debería ser suficiente grabar / restaurar todos los objetos en memoria y dependencias, ie:
  - Objetos del “usuario”:
    - Aplicación: clases / instancias de la aplicación
    - Variables
    - Etc.
  - Módulos
  - Objetos **de** la VM:
    - Callstack de cada Thread

Sleepy - problemas

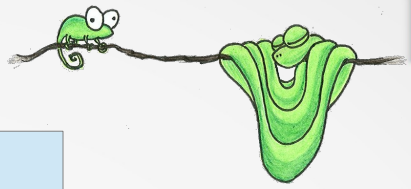


- Objetos no serializables:
  - Para guardarlos: instancias en C que no permiten su serialización
  - Para restaurarlos:
    - Clases en C de las cuales no se pueden crear instancias desde Python
    - Instancias que pueden ser creadas pero no “configuradas” en un estado determinado
- Ejemplos: Módulos, Frames, Objetos C (iteradores, etc)..

Sleepy – restaurar un módulo

- No se puede serializar un módulo cargado en memoria, pero no es un gran problema, se lo puede volver a cargar!
- Las variables globales “de módulo”, deberían ser serializadas / restauradas?
  - Variables dependiendo de la arquitectura donde se ejecuta, por ejemplo: ***`os.path.sep`***
  - Variables que tienen un estado, podría ser, por ejemplo: ***`random._inst`*** - ***`random.getstate()`***

Sleepy - Callstack

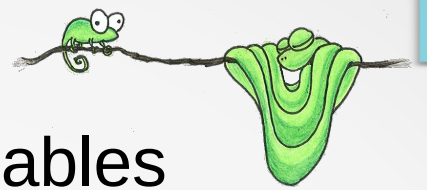


```
(Pdb) where
  ../recipe578278.py(167)<module>()
-> main()
  ../recipe578278.py(162)main()
-> myNN.train(pat)
  ../recipe578278.py(134)train()
-> self.BP(patterns, N, M,i)
  ../recipe578278.py(73)BP()
-> self.runNN(inputs)
 ../recipe578278.py(24)runNN()
-> self.ai = inputs
(Pdb) _
```

frames

Sleepy - Frames

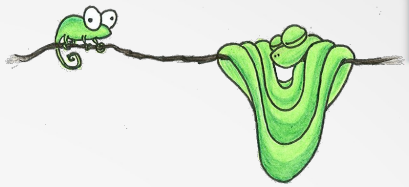
- Una lista enlazada de frames
- No todos los atributos son modificables



Frame Fields

```
f_back            →   <frame object at 0x2ca1d60>
f_builtins        →   {..}  # namespace de builtins del frame
f_code            →   <code object interact at 0x7f7cfe904430,
                        file "../interactiveshell.py", line 327>
f_exc_traceback   →   None
f_exc_type        →   None
f_exc_value       →   None
f_globals         →   {..} # globales del frame
f_lasti           →   676  # ultima instruccion bytecode ejecutada
f_lineno          →   409  # nro. linea en el fuente python
f_locals          →   {..} # locales del frame !!!
f_restricted      →   False
f_trace           →   None
```

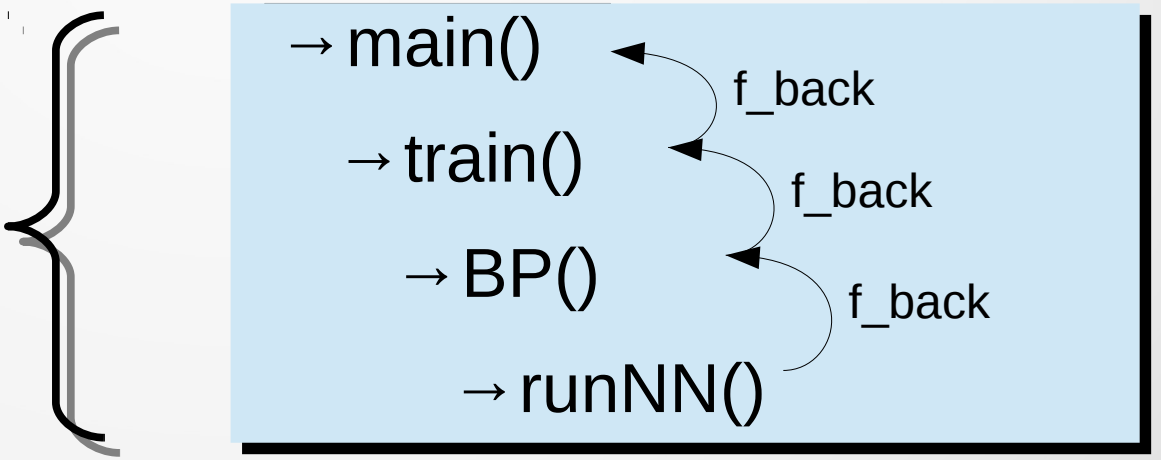
Sleepy - Frames



- “No se puede crear **`frames`**”
  - A medias. Invocar una función, crea un frame:

    `f()` → El llamado creará un frame!

¿Cómo restaurar un callstack?

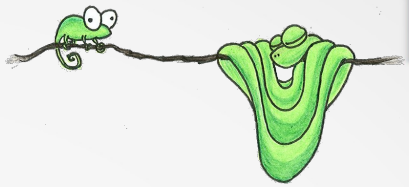


Sleepy – restaurar el callstack

- Llamando a la función original, en el ejemplo, **`main`**, y luego **`train`** y así sucesivamente, sería posible recrear el call-stack original

- Utilizando el debugger (**`bdb.py`** / **`pdb.py`**) es posible:
  1. Llamar una función, que se cree su frame (**`<frame f>`**)
  2. Configurar las variables locales (**`f->f_locals`**)
  3. Actualizar el "**`pc`**" de la VM para ejecutar inmediatamente el siguiente call y crear el siguiente frame (**`f->lastno = saved_lineno`**) y volver a 1.

Sleepy – restaurando un frame



```
(Pdb) r
> /home/aweil/test_for.py(2)fail_pc()
-> for x in xrange(100):
(Pdb) l
  1 B   def fail_pc():
  2  →      print 0
  3          for x in xrange(100):
  4              print 1
  5              print 2
  6              print 3
  7
  8    fail_pc()
```

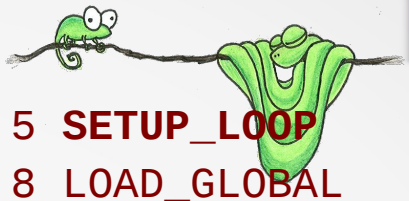
linea actual

X

Si "**pc**" (→) se encuentra dentro de un bloque, no podrá cambiarse a una linea en otro!

```
(Pdb) jump 4
*** Jump failed: can't jump into the middle of a block
```

Sleepy – bytecode & blocks

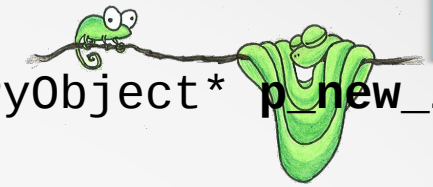


```
1 def fail_pc():
2     print 0
3     for x in xrange(100):
4         print 1
5         print 2
6         print 3
```

```
3           5 SETUP_LOOP              35 (to 43)
            8 LOAD_GLOBAL              0 (xrange)
           11 LOAD_CONST               2 (100)
           14 CALL_FUNCTION            1
           17 GET_ITER
      >>   18 FOR_ITER                21 (to 42)
           21 STORE_FAST               0 (x)

4          24 LOAD_CONST               3 (1)
           27 PRINT_ITEM
           28 PRINT_NEWLINE

5          29 LOAD_CONST               4 (2)
           32 PRINT_ITEM
           33 PRINT_NEWLINE

6          34 LOAD_CONST               5 (3)
           37 PRINT_ITEM
           38 PRINT_NEWLINE
           39 JUMP_ABSOLUTE           18
      >>   42 POP_BLOCK
```

frame.f_lineno = X

```
int frame_setlineno(PyFrameObject *f, PyObject* p_new_lineno) (*)
```



Setter for f_lineno - you can set f_lineno from within a trace function in order to jump to a given line of code, subject to some restrictions. Most lines are OK to jump to because they don't make any assumptions about the state of the stack (obvious because you could remove the line and the code would still work without any stack errors), but there are some constructs that limit jumping:

- Lines with an '**except**' statement on them can't be jumped to, because they expect an exception to be on the top of the stack.
- Lines that live in a '**finally**' block can't be jumped from or to, since the END_FINALLY expects to clean up the stack after the 'try' block.
- **'try'/'for'/'while' blocks** can't be jumped into because the blockstack needs to be set up before their code runs, and for 'for' loops the iterator needs to be on the stack.

* Link to frameobject.c#L78 setlineno() function

Sleepy – entrando en loops y bloques

Principalmente encontramos tres maneras de hacerlo para:

```
for x in range(100):
    print x
return x
```

1. Duplicar el código de bloque “fuera” de la función:

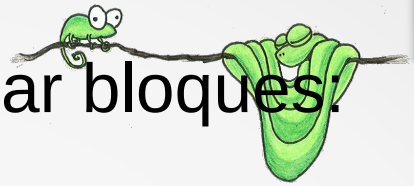
```
for x in range(100):
    print x
return x
print x
```

2. Manejar el loop desde una función provista:

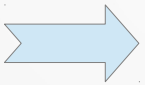
```
for x in sleepy.chkIt(someId):
    print x
return x
```

Sleepy – `for`

3. Re-escribir el bytecode para no usar bloques:



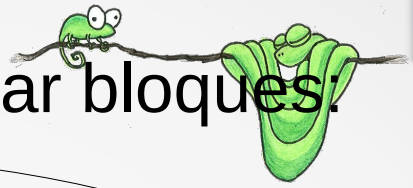
```
for x in xrange(100):
    print x
return x
```



```
_it = iter(xrange(100))
:next
try:
    X = _it.next()
except StopIteration:
   goto @end

print x

goto next
:end

return x
```

- Casos especiales: **`break`**, **`continue`**, **`else`**

Sleepy – `for`



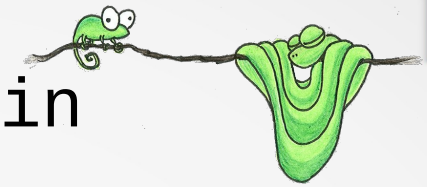
3. Re-escribir el bytecode para no usar bloques:

```
for x in xrange(100):
    print x
return x
```

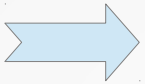
```
_it = iter(xrange(100))
:next
try:
    x = _it.next()
except StopIteration:
    goto @end
print x

goto next
:end

return x
```

- Casos especiales: **`break`**, **`continue`**, **`else`**

Sleepy - **while**



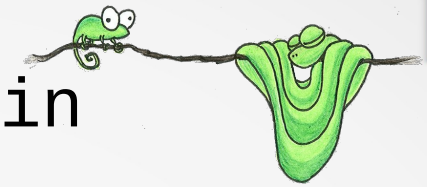
```
while condicion:
    bloque
```



```
:begin
if not condicion:
    goto end
bloque
goto begin
:end
```

- Casos especiales: **`break`**, **`continue`**, **`else`**.

Sleepy - **while**

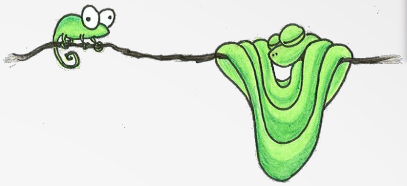


```
while condicion:
    bloque
```

```
:begin
if not condicion:
    goto end
bloque
goto begin
:end
```

- Casos especiales: **break**, **continue**, **else**.

Sleepy – `try / except`

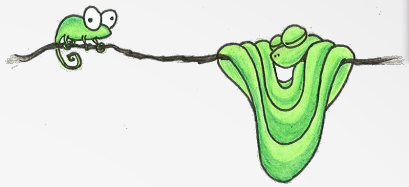


```
try:
    statement_1
     ...
    statement_n
except SomeExc:
    bloque1
except OtherExc, e:
    bloque2
except:
    bloque3
else:
    bloque4
```

```
Temp = None
try:
    statement_1
except SomeExc:
    Temp = SomeExc
except OtherExc ex:
    Temp = OtherExc
except:
    Temp = True

            ...
```

```
if Temp is SomeExc:
    bloque1
if Temp is OtherExc:
    bloque2
if Temp == True:
    bloque3
if Temp is None:
    bloque4
```

Sleepy – `try / except`



```
try:
    statement_1
     ...
    statement_n
except SomeExc:
    bloque1
except OtherExc, e:
    bloque2
except:
    bloque3
else:
    bloque4
```

```
Temp = None
try:
    statement_1
except SomeExc:
    Temp = SomeExc
except OtherExc ex:
    Temp = OtherExc
except:
    Temp = True
```

statements *1 .. n*

```
try:
    statement_n
...
```

```
if Temp is SomeExc:
    bloque1
if Temp is OtherExc:
    bloque2
if Temp == True:
    bloque3
if Temp is None:
    bloque4
```

* antes de cada **`try`** hay una verificacion:
`Temp is None`

Sleepy – recompilando loops

- El módulo **`compiler.pycodegen`*** incluye un *compilador python* implementado *en python*: **`CodeGenerator`**!
- Para cambiar los loops es suficiente redefinir:
  - **`visitBreak, visitContinue, visitWhile`**
  - **`visitGoto, visitTryExcept, visitFor`**
  - **`visitTryFinally`**
- También cambiamos **`visitModule`** para poder automaticamente importar un módulo auxiliar (**`sleepy`**).

* **Cuidado!** Solo en Python 2.x! Deprecado en python 3k! :-(

Sleepy – iteradores & co.

- En el caso de estar re-compilando un **`for`** tenemos que considerar, además, los distintos tipos de iteradores posibles.
- En general, no son persistibles:

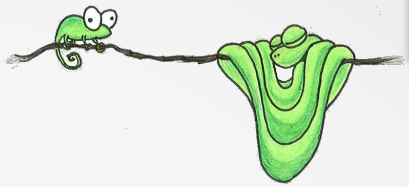
```
>>> pickle.dumps(iter([])) → TypeError: can't pickle listiterator objects
```

- Para resolver esto, utilizamos una función que importaremos de manera ad-hoc, que:
  - obtiene el iterador
  - lo transforma en un iterador "**`pure-python`**"
- De esta forma, queda código "reanudable"

Sleepy – control

- Diferentes formas de especificar como/cuando persistir una ejecución, por ahora, solamente:
  - Por tiempo:
    - La forma mas sencilla por ahora de implementar la “interrupción” de la ejecución por ahora, chequeo del tiempo de ejecución en una función de tracing.
      - Lento
      - No asegura la interrupción en el momento indicado
  - Asistido:
    - Mediante llamados auxiliares regulares a *sleepy* para verificar el tiempo de ejecución:
      - **`sleepy.CheckSuspend()`**

Sleepy – componentes



- Compilador:
  - genera código reanudable
- CLI: Permite al usuario elegir:
  - crear una nueva instancia de ejecución
  - reanudar una ejecución suspendida
    - en ambos casos, especificar nuevos parametros para controlar la ejecución iniciada
- Utilidades para serializar:
  - objetos + código-en-ejecución
- Funcionalidad / control:
  - brinda desde código la posibilidad de controlar la ejecución (cuando/como suspender, etc)

Sleepy – contras, problemas y más..

- Genera bytecode no-óptimo para la VM → Por ejemplo, es mas rápido el bytecode: "**`SETUP_LOOP + .. + FOR_ITER`**" qué:

  `try: x=next(iter); except StopIteration..`

- El nuevo compilador no está, tampoco, para nada optimizado, podemos encontrar por ejemplo, cosas como:

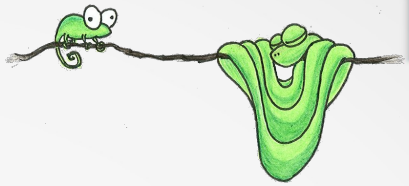
```
123 JUMP_ABSOLUTE   129
126 JUMP_FORWARD      0 (to 129)
129 JUMP_ABSOLUTE    21
```

  - Se ejecuta a 0.5X la velocidad del bytecode de CPython
- Aparentemente **`compiler.pycodegen`** no soporta python 2.7 al 100%
- Permite debuggear con mayor flexibilidad, al posibilitar entrar en bloques!

Sleepy - links

- **Repositorio:** https://bitbucket.org/tenuki/sleepy/
- **Design of the CPython Compiler:** http://www.python.org/dev/peps/pep-0339/
- **An implementation of import importlib:** http://docs.python.org/dev/library/importlib.html
- **Python Module of the Week: “dis”:** http://www.doughellmann.com/PyMOTW/dis/
- **New Import Hooks:** http://www.python.org/dev/peps/pep-0302/
- **The compiler module:** http://docs.python.org/2/library/compiler.html
- http://docs.python.org/library/imputil.html
- http://docs.python.org/library/imp.html
- http://www.stackless.com/wiki/Pickling
- http://doc.pypy.org/en/latest/stackless.html
- **Artwork from:** http://www.bluebison.net

Sleepy – Gracias



¿Preguntas?