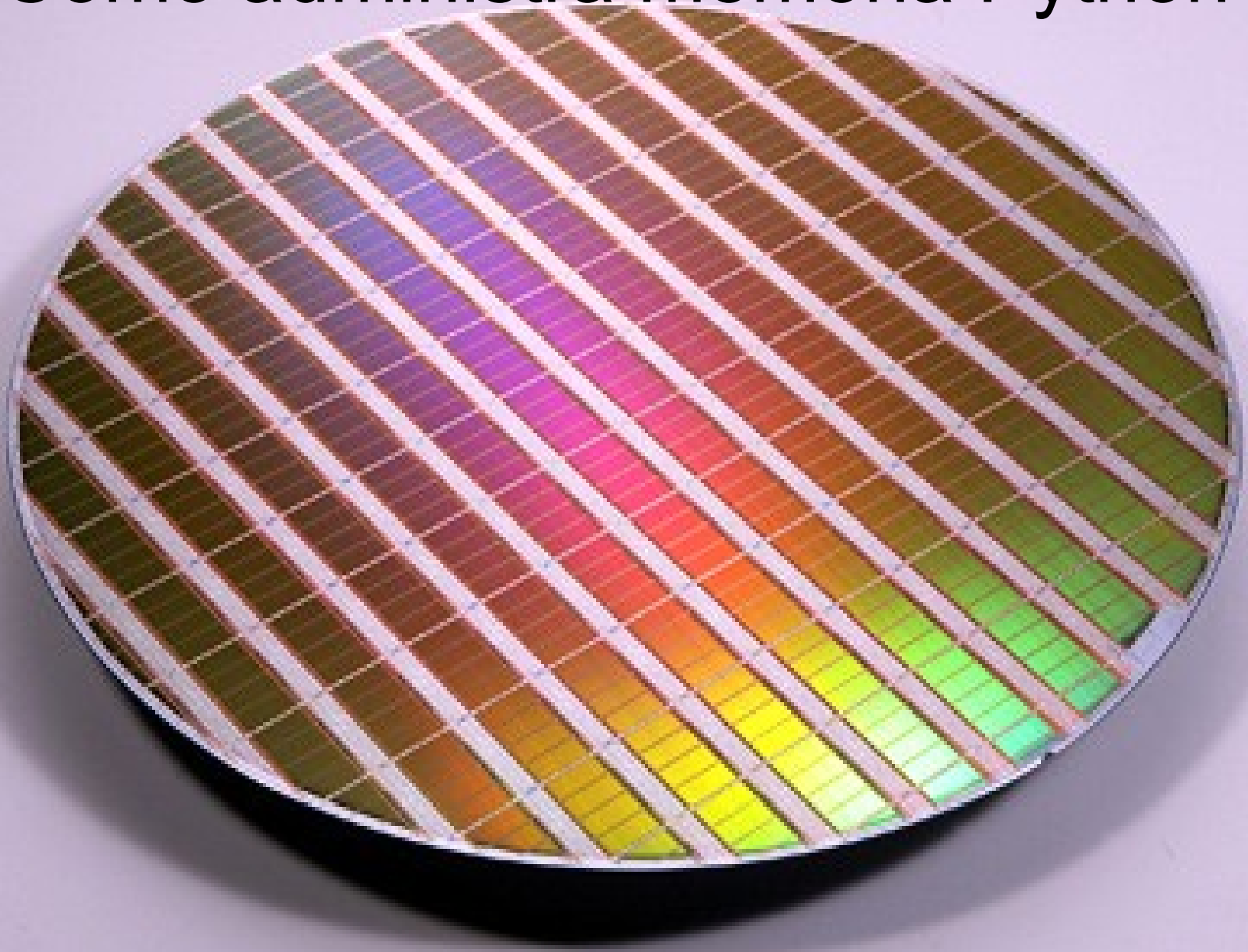
Depuración y Defragmentación de memoria

¿Cómo administra memoria Python?



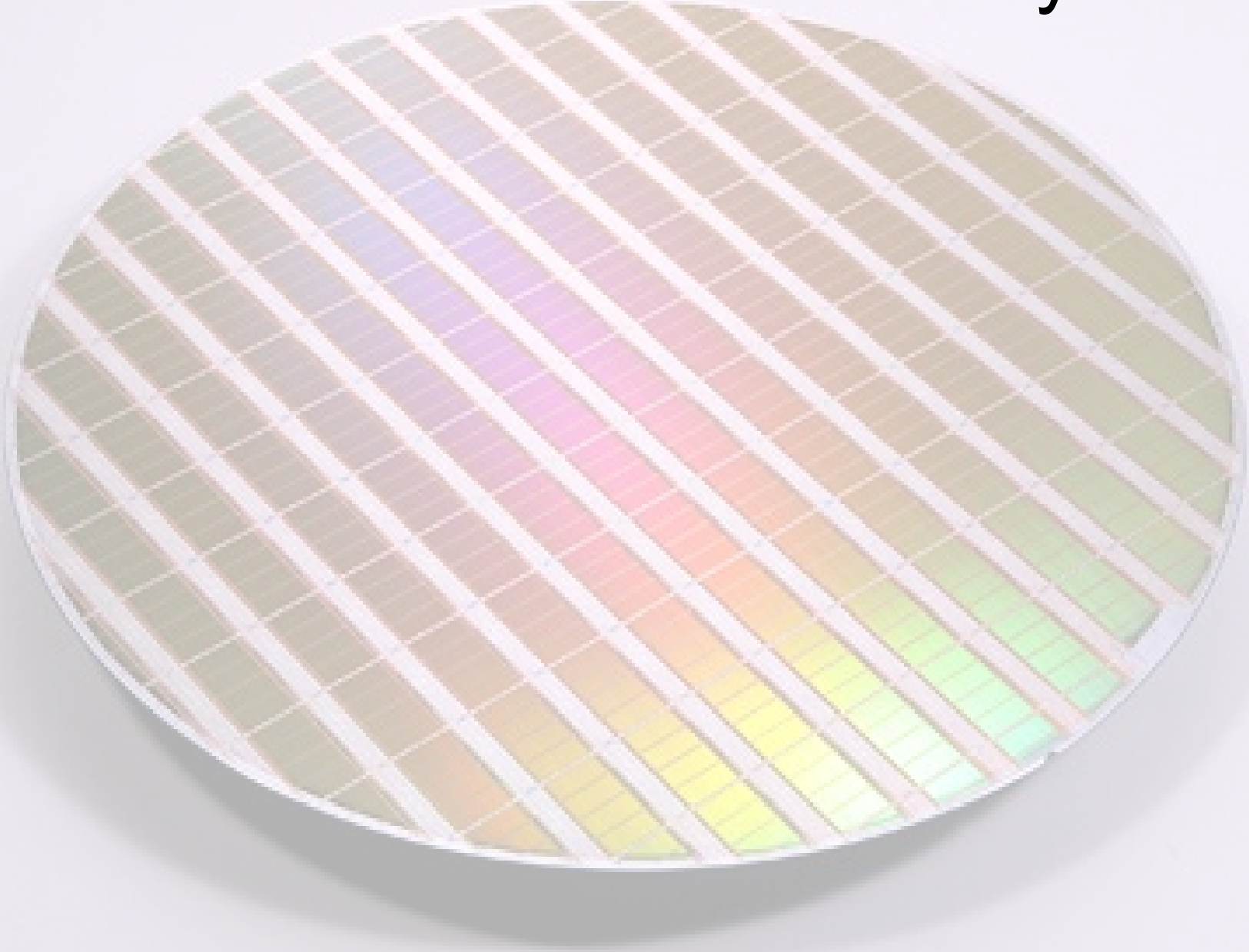
¿Cómo administra memoria Python?

Malloc

¿Cómo administra memoria Python?

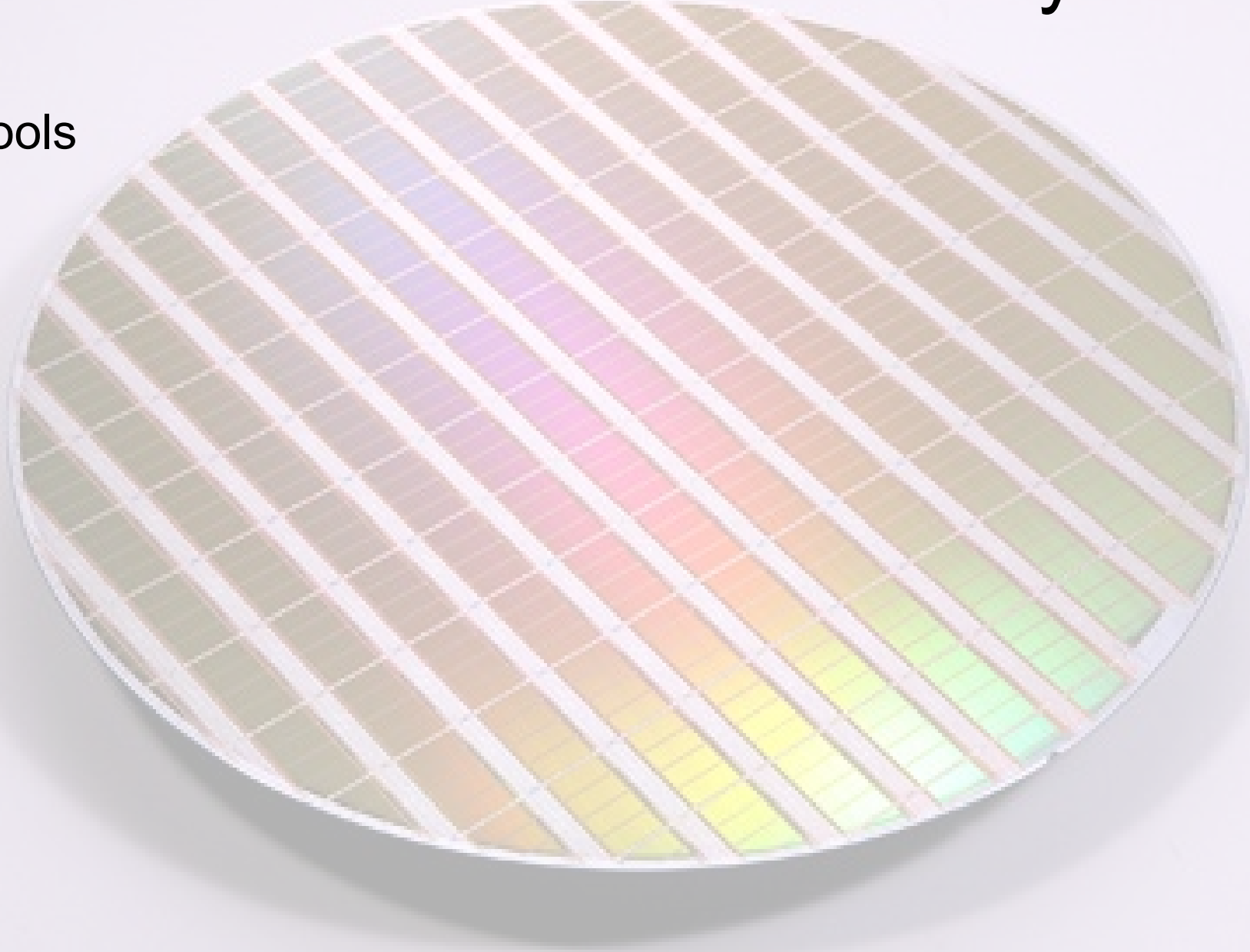
~~Malloc~~

¿Cómo administra memoria Python?



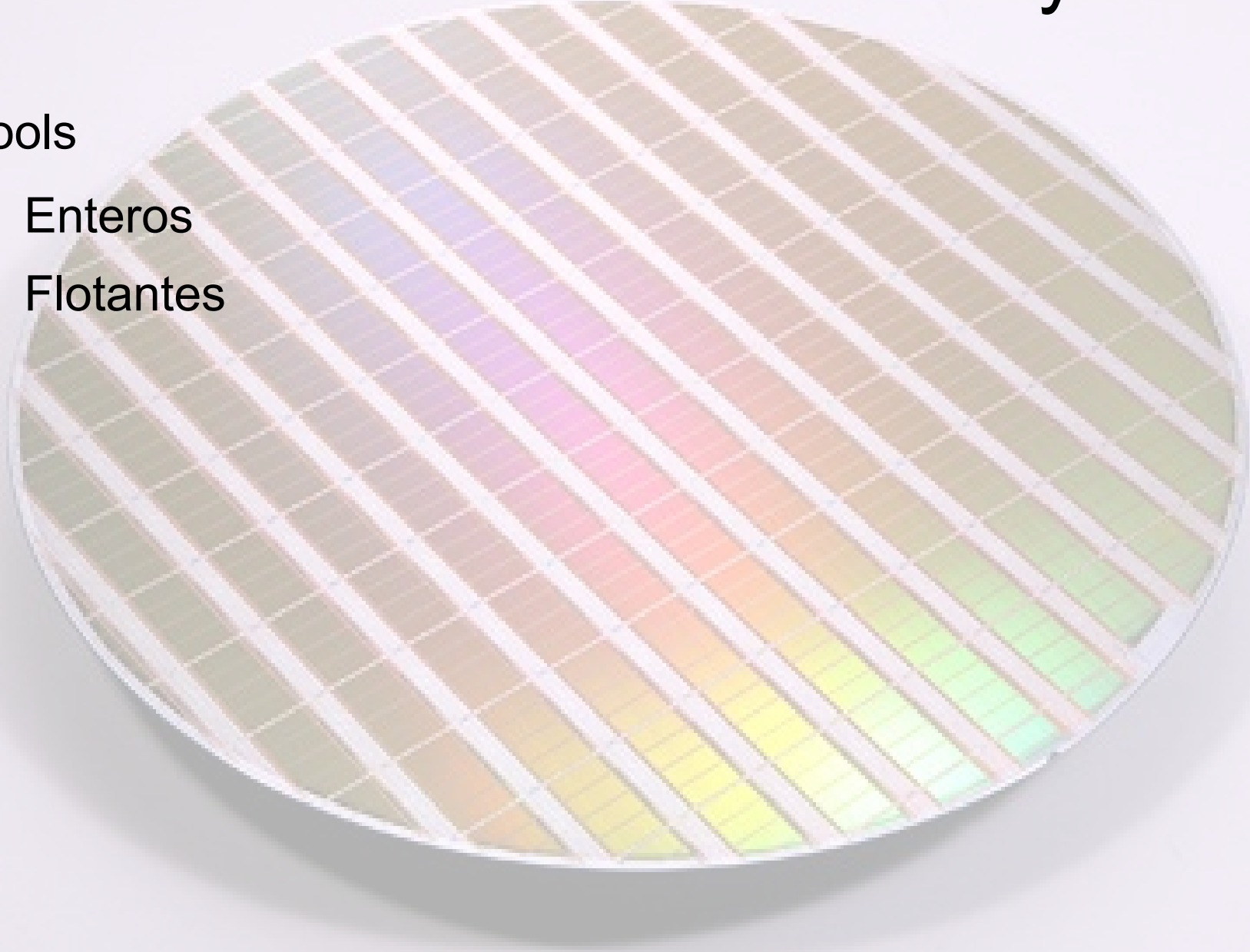
¿Cómo administra memoria Python?

- Pools

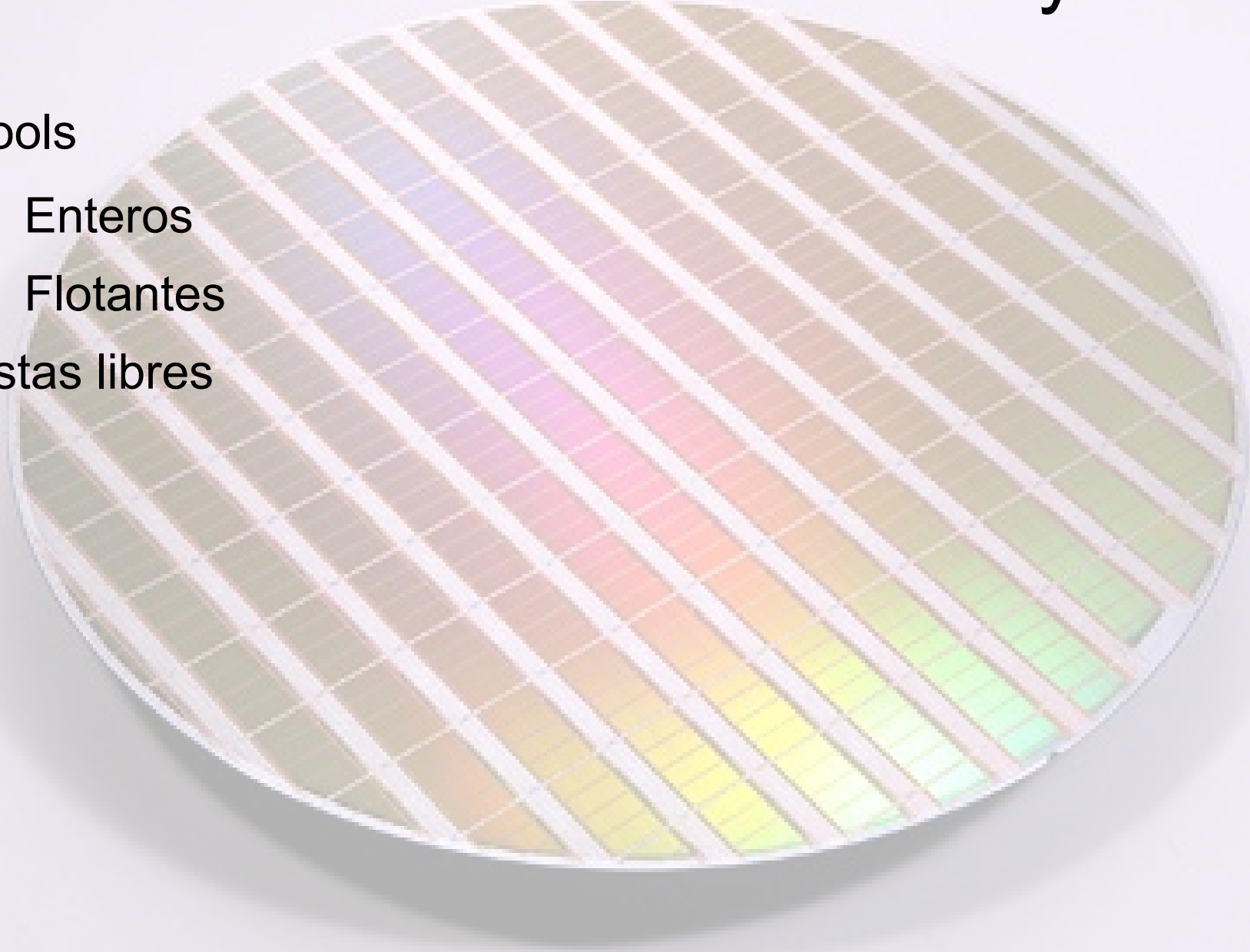


¿Cómo administra memoria Python?

- Pools
  - Enteros
  - Flotantes

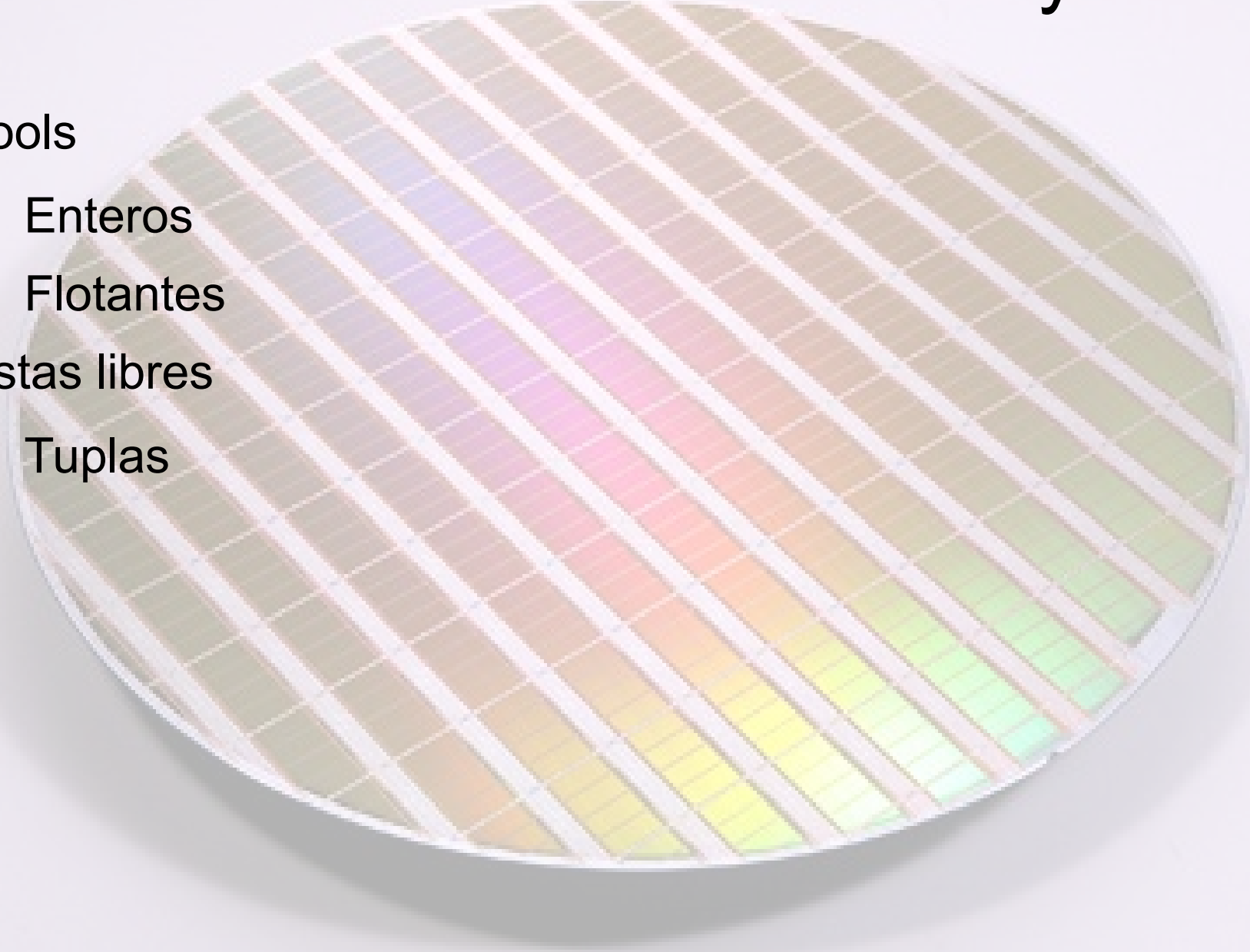


¿Cómo administra memoria Python?



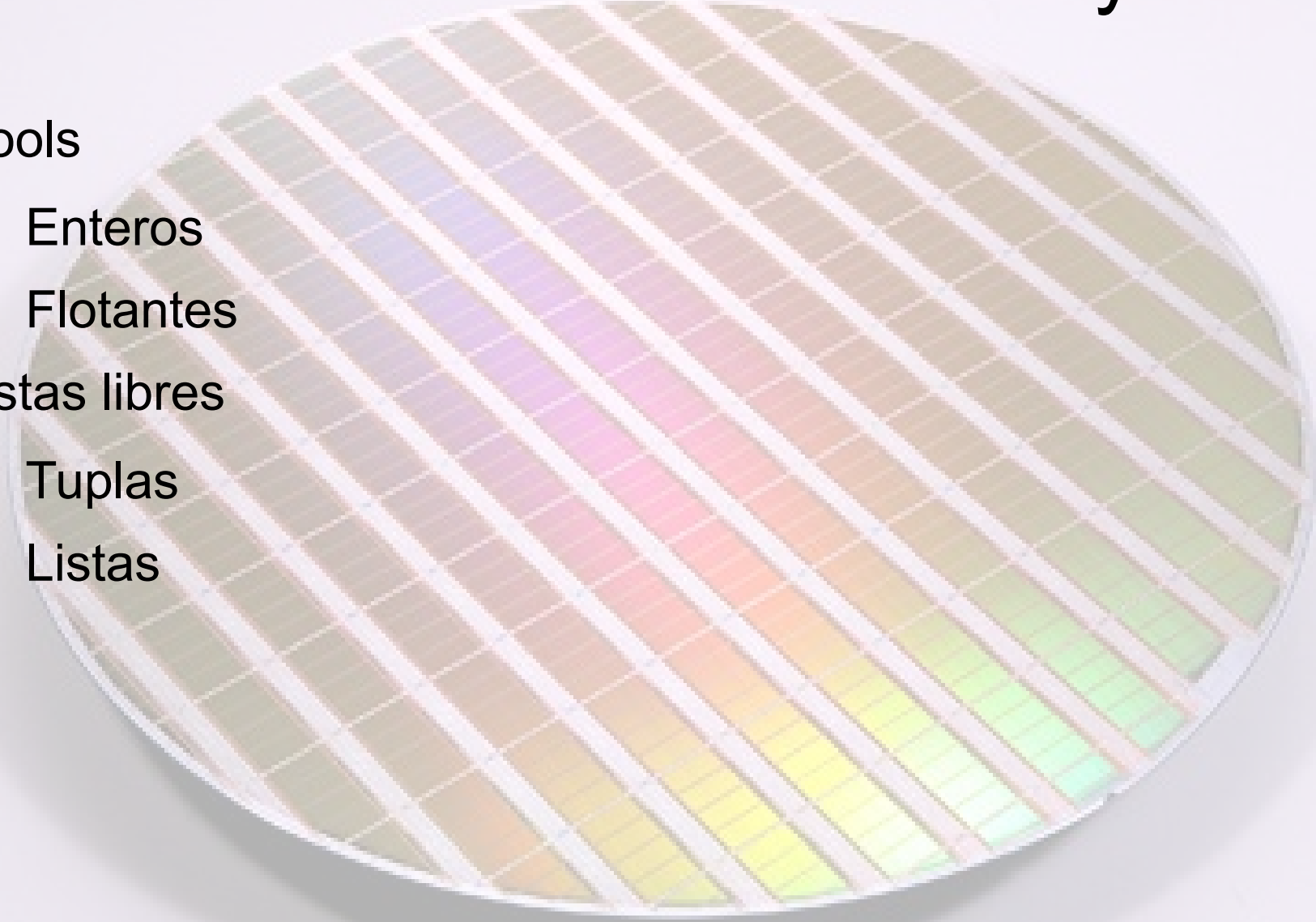
- Pools
  - Enteros
  - Flotantes
- Listas libres

¿Cómo administra memoria Python?



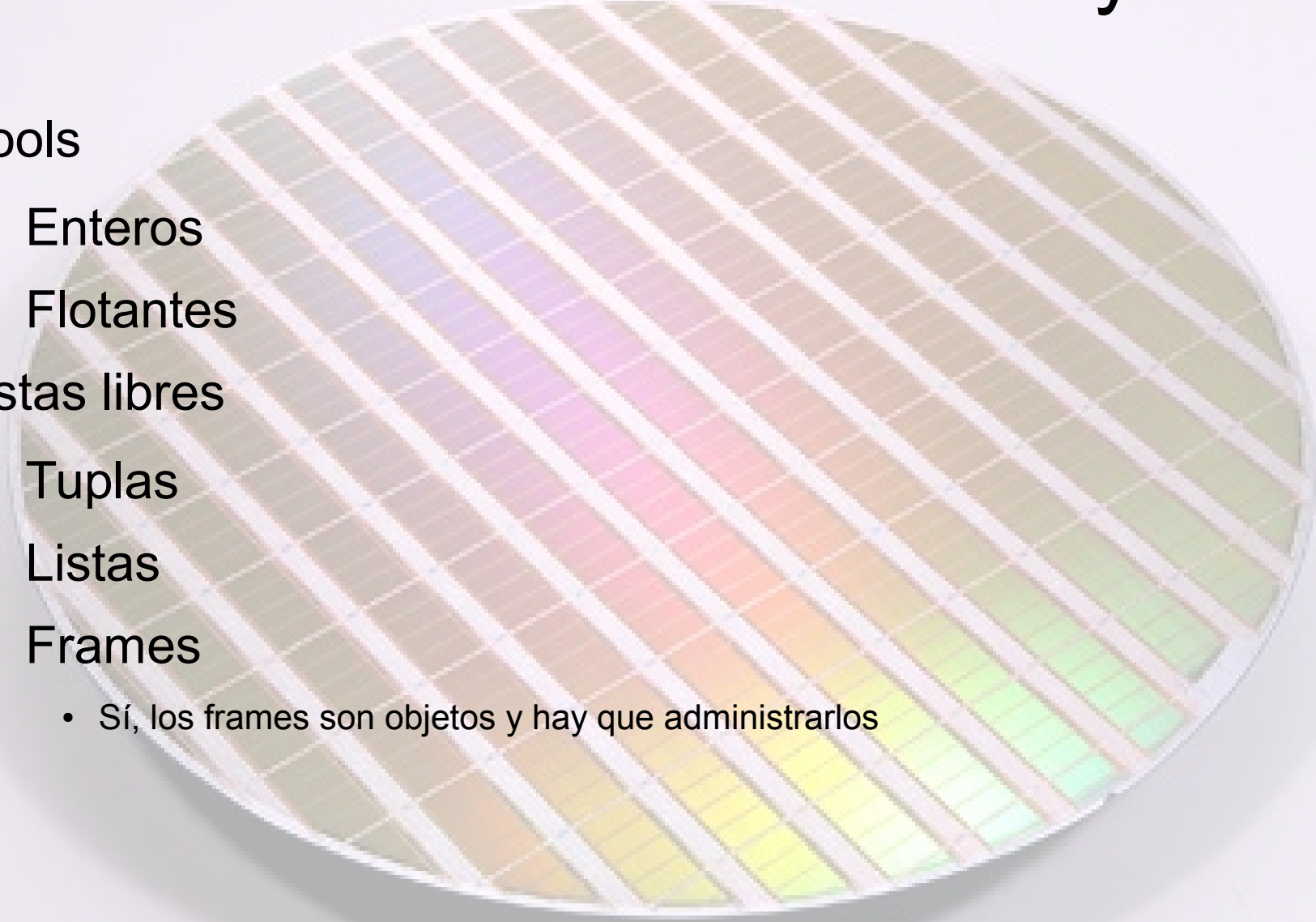
- Pools
  - Enteros
  - Flotantes
- Listas libres
  - Tuplas

¿Cómo administra memoria Python?



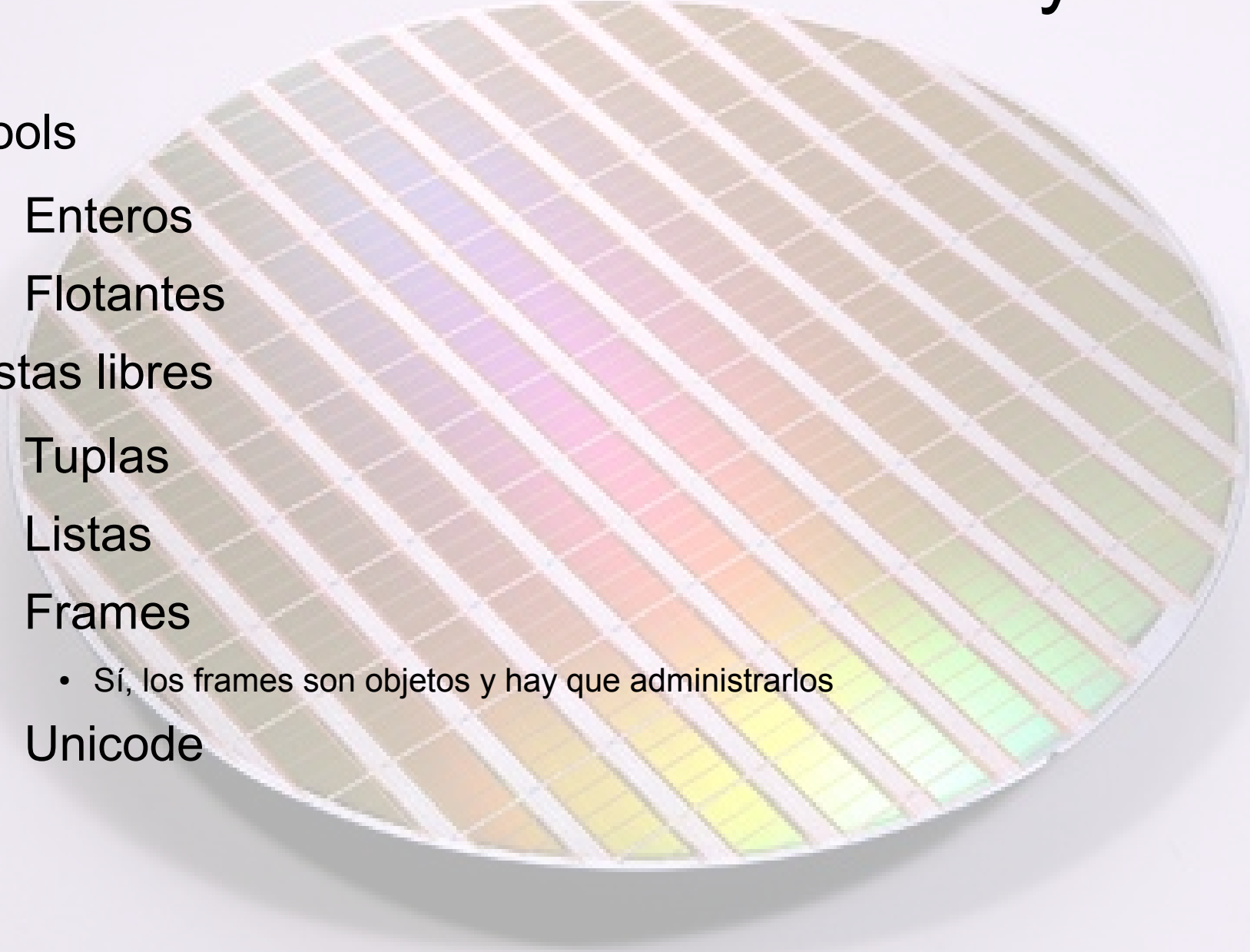
- Pools
  - Enteros
  - Flotantes
- Listas libres
  - Tuplas
  - Listas

¿Cómo administra memoria Python?



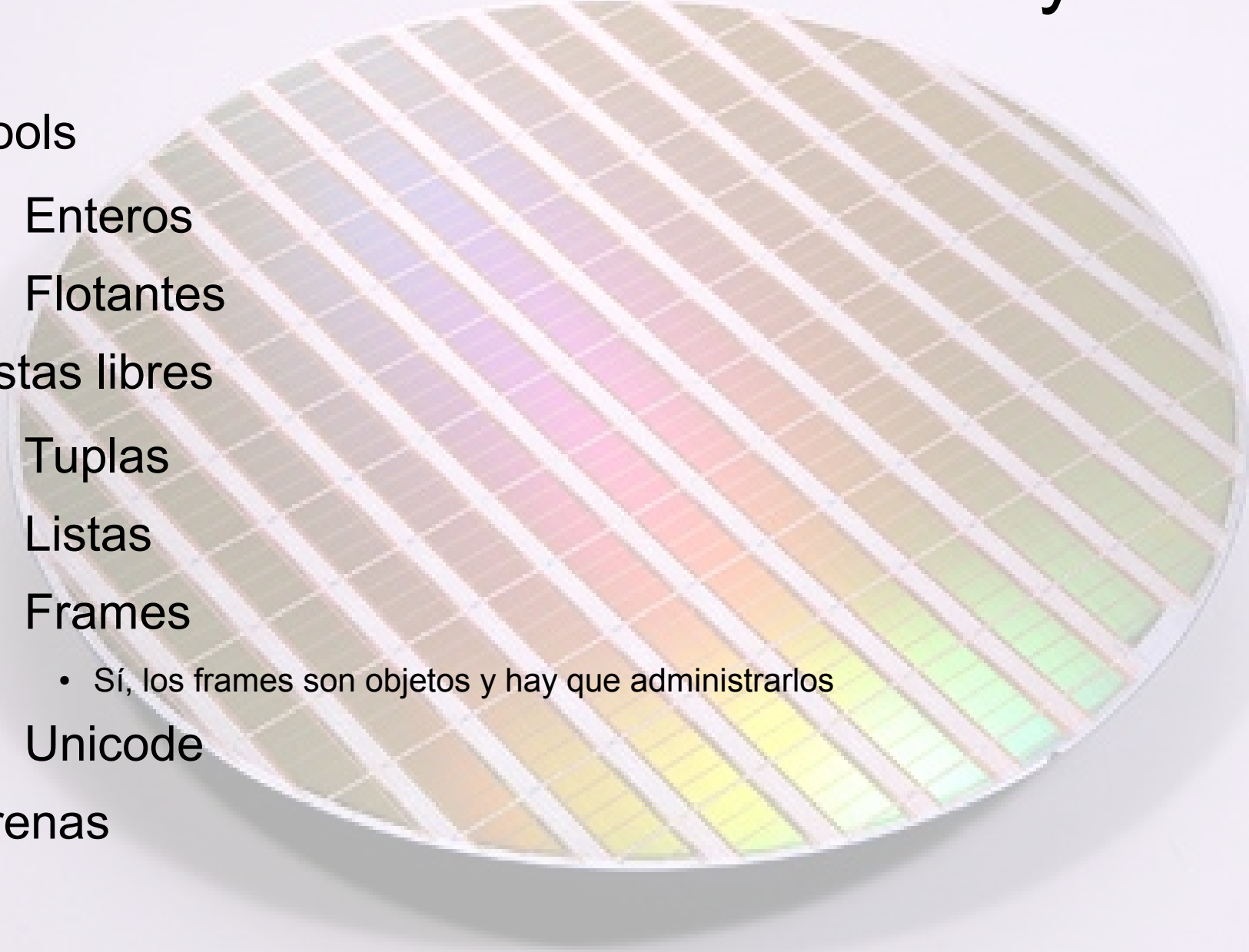
- Pools
  - Enteros
  - Flotantes
- Listas libres
  - Tuplas
  - Listas
  - Frames
    - Sí, los frames son objetos y hay que administrarlos

¿Cómo administra memoria Python?



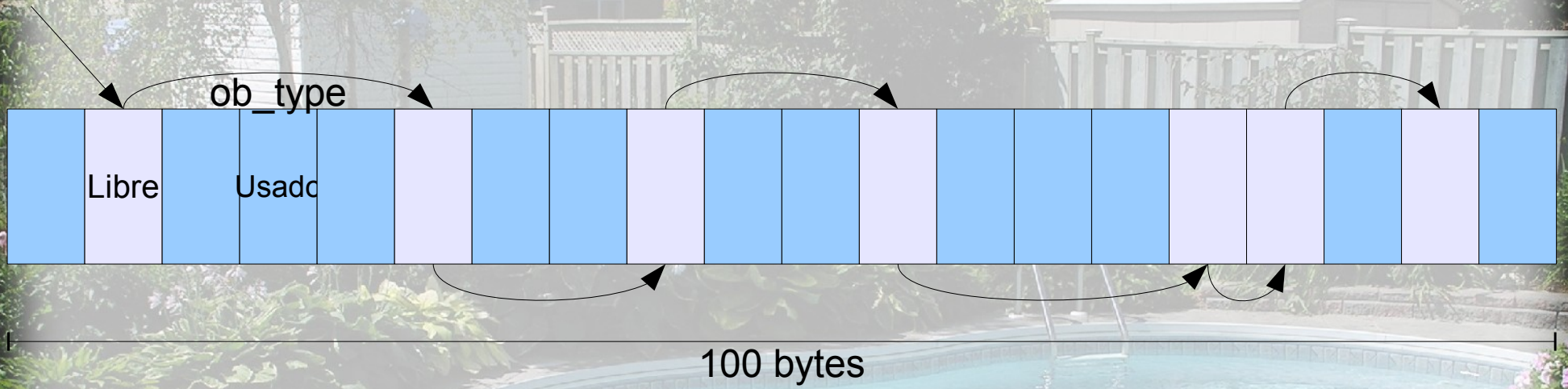
- Pools
  - Enteros
  - Flotantes
- Listas libres
  - Tuplas
  - Listas
  - Frames
    - Sí, los frames son objetos y hay que administrarlos
  - Unicode

¿Cómo administra memoria Python?



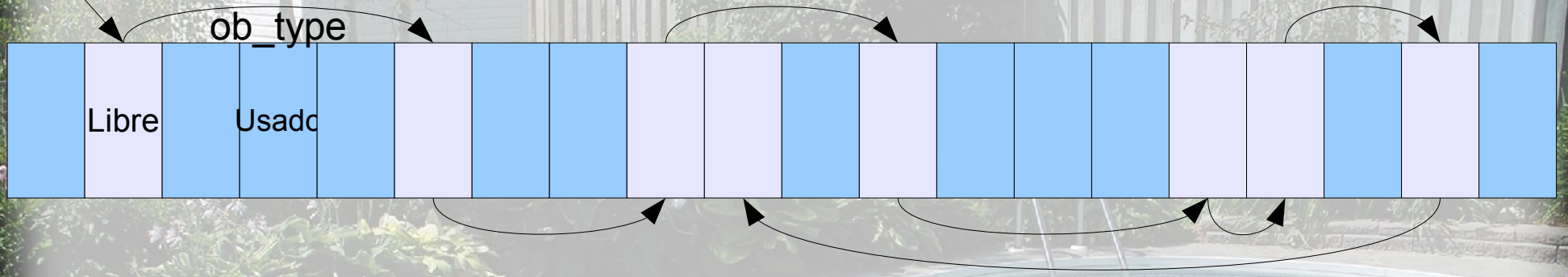
- Pools
  - Enteros
  - Flotantes
- Listas libres
  - Tuplas
  - Listas
  - Frames
    - Sí, los frames son objetos y hay que administrarlos
  - Unicode
- Arenas

Pools



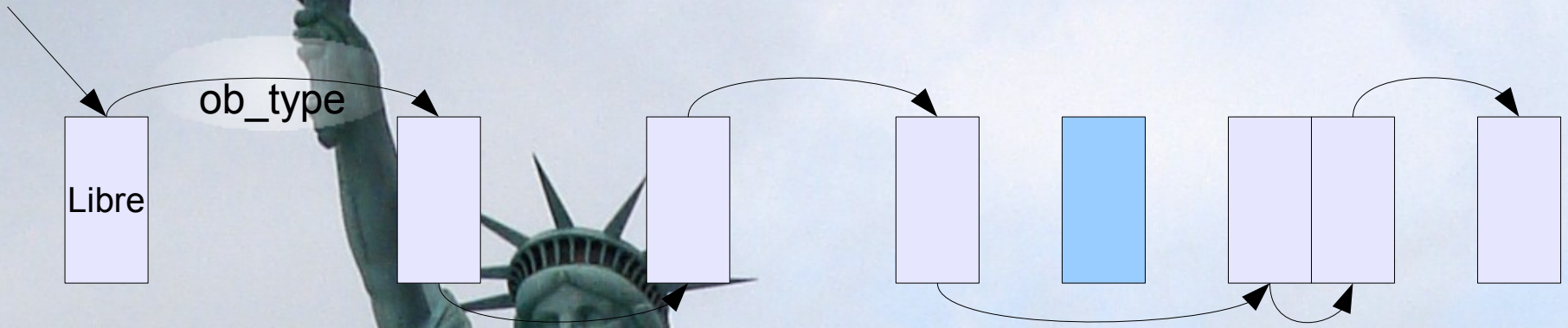
Se usa el campo ob_type del objeto para formar una lista enlazada de objetos libres.

Pools



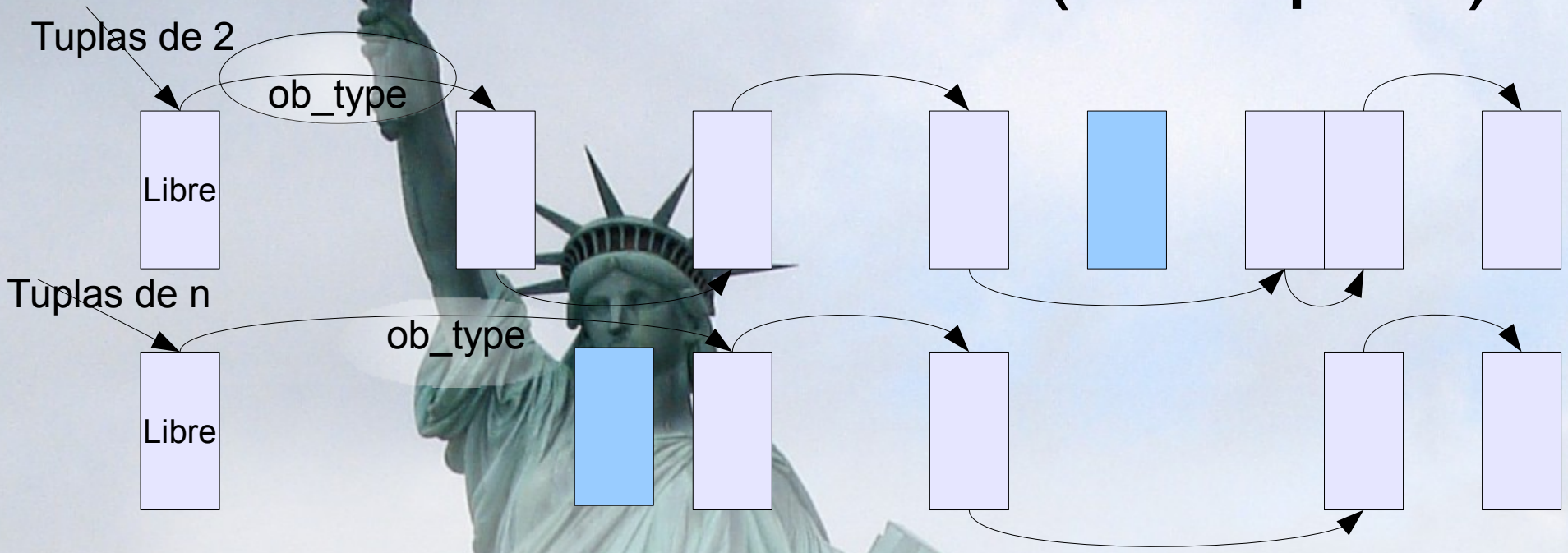
Se usa el campo ob_type del objeto para formar una lista enlazada de objetos libres.

Listas libres



Se usa el campo ob_type del objeto para formar una lista enlazada de objetos libres...
...pero desperdigados por toda la memoria

Listas libres (de tuplas)



Para las tuplas, que son demasiaaaaado comunes, hay varias listas libres agrupadas por tamaño de la tupla

¿Arenas?

¿Arenas?

- Objetos grandes:
  - Van directo a malloc

¿Arenas?

- Objetos grandes:
  - Van directo a malloc
- Objetos pequeños:
  - Para cada tamaño, una lista de pools
  - Cada pool con su lista libre
  - Cada pool de 4K


Negro es usado
Fragmentación
Mucho espacio libre pero discontiguo
no entran objetos grandes
Blanco es libre

Fragmentación

- El tamaño virtual de la aplicación crece desproporcionadamente al tamaño efectivo
- Baja la performance por peor localidad de referencia.
- En casos extremos thrashing, y hasta OOM

Fragmentación

Python

```
>>> l = []
>>> for i in xrange(1,100):
...     ll = [ " " * i * 16 for j in xrange(1000000 / i) ]
...     ll = ll[::2]
...     l.extend(ll)
...     print sum(map(len,l))
...
8000000
16000000
…
792005616
>>>
```

top

```
10467 claudiof  20   0 1676m 1.6g 1864 S    0 82.7   1:17.07 python
```

Fragmentación

Python

```
>>> del l
```

top

```
10467 claudiof  20   0 1532m 1.5g 1864 S    0 75.6   1:17.96 python
```

Fragmentación

Python

```
>>> l = []
>>> for i in xrange(1,100):
...     ll = [ " " * i * 16 for j in xrange(1000000 / i) ]
...     ll = ll[::2]
...     l.extend(ll)
...     print sum(map(len,l))
...
8000000
16000000
…
792005616
>>>
```

top

```
10467 claudiof  20   0 1676m 1.6g 1864 S    0 82.8   2:33.39 python
```

Fragmentación

Python

```
>>> del l
```

top

```
10467 claudiof  20   0 1664m 1.6g 1876 S    0 82.1   5:12.27 python
```

Fragmentación

Python

```
>>> del l
>>> del ll
```

top

```
10467 claudiof  20   0 1664m 1.6g 1876 S    0 82.1   5:12.28 python
```

Fragmentación

Python

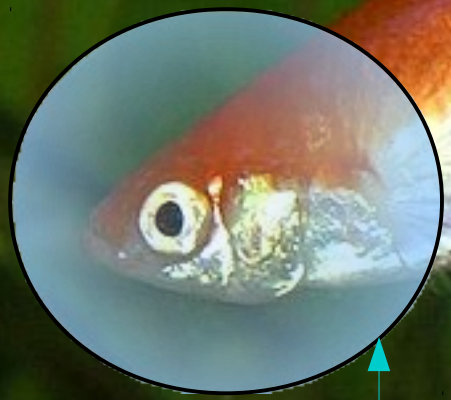
```
>>> print “WTF!\n” * 5

WTF!
WTF!
WTF!
WTF!
WTF!

>>>
```

top

```
10467 claudiof  20   0 1664m 1.6g 1876 S    0 82.1   5:12.28 python
```

From guppy import heapy

From guppy import heapy



Ésto es un guppy

foto by Przemysław Malkowski

From guppy import heapy

Python

```
>>> from guppy import hpy
>>> hp = hpy()
>>> heap1 = hp.heap()
```

Guppy.heapy

```
>>> hp.heap()
Partition of a set of 23778 objects. Total size = 1760692 bytes.
 Index  Count   %     Size   % Cumulative  % Kind (class / dict of class)
     0  10642  45   696652  40     696652  40 str
     1   5432  23   196864  11     893516  51 tuple
     2    345   1   112968   6    1006484  57 dict (no owner)
     3   1546   7   105128   6    1111612  63 types.CodeType
     4     66   0   104592   6    1216204  69 dict of module
     5    174   1    93168   5    1309372  74 dict of type
     6    194   1    86040   5    1395412  79 type
     7   1472   6    82432   5    1477844  84 function
     8    124   1    67552   4    1545396  88 dict of class
     9   1027   4    36972   2    1582368  90 __builtin__.wrapper_descriptor
```

foto by Przemysław Małkowski

From guppy import heapy

Python

```
>>> from guppy import hpy
>>> hp = hpy()
>>> heap1 = hp.heap()
```

Guppy.heapy

```
>>> hp.heap()
Partition of a set of 23778 objects. Total size = 1760692 bytes.
 Index  Count   %     Size   % Cumulative  % Kind (class / dict of class)
     0  10642  45   696652  40     696652  40 str
     1   5432  23   196864  11     893516  51 tuple
     2    345   1   112968   6    1006484  57 dict (no owner)
                              6    1111612  63 types.CodeType
                              6    1216204  69 dict of module
                              5    1309372  74 dict of type
                              5    1395412  79 type
                              5    1477844  84 function
     8    124   1    67552   4    1545396  88 dict of class
     9   1027   4    36972   2    1582368  90 __builtin__.wrapper_descriptor
```

Esto consume python
Simplemente al iniciar
(en objetos)

foto by Przemysław Małkowski

From guppy import heapy

Python

```
>>> l = []
>>> for i in xrange(1,100):
...     …
```

Guppy.heapy

```
>>> hp.heap()
Partition of a set of 2612542 objects. Total size = 866405844 bytes.
 Index  Count   %     Size   % Cumulative  % Kind (class / dict of class)
     0 2599386  99 854833304  99 854833304  99 str
     1     132   0 10516640   1 865349944 100 list
     2    5433   0   197064   0 865547008 100 tuple
     3     351   0   113784   0 865660792 100 dict (no owner)
     4    1547   0   105196   0 865765988 100 types.CodeType
     5      67   0   105112   0 865871100 100 dict of module
     6     174   0    93168   0 865964268 100 dict of type
     7     194   0    86040   0 866050308 100 type
     8    1472   0    82432   0 866132740 100 function
     9     124   0    67552   0 866200292 100 dict of class
```

foto by Przemysław Małkowski

From guppy import heapy

Python

```
>>> del l
>>> del ll
```

Guppy.heapy

```
>>> hp.heap()
Partition of a set of 23844 objects. Total size = 1765236 bytes.
 Index  Count   %     Size   % Cumulative  % Kind (class / dict of class)
     0  10690  45   698996  40     698996  40 str
     1   5433  23   197068  11     896064  51 tuple
     2    351   1   113784   6    1009848  57 dict (no owner)
     3   1547   6   105196   6    1115044  63 types.CodeType
     4     67   0   105112   6    1220156  69 dict of module
     5    174   1    93168   5    1313324  74 dict of type
     6    194   1    86040   5    1399364  79 type
     7   1472   6    82432   5    1481796  84 function
     8    124   1    67552   4    1549348  88 dict of class
     9   1027   4    36972   2    1586320  90 __builtin__.wrapper_descriptor
```

foto by Przemysław Małkowski

From guppy import heapy

WTF?

Heapy nos dice que python ocupa de nuevo 1.7MB
Sucede que de hecho, el resto es espacio “libre”
(libre para python, no para el sistema operativo)

Guppy.heapy

```
>>> hp.heap()
Partition of a set of 23844 objects. Total size = 1765236 bytes.
 Index  Count   %     Size   % Cumulative  % Kind (class / dict of class)
     0  10690  45   698996  40     698996  40 str
     1   5433  23   197068  11     896064  51 tuple
     2    351   1   113784   6    1009848  57 dict (no owner)
     3   1547   6   105196   6    1115044  63 types.CodeType
     4     67   0   105112   6    1220156  69 dict of module
     5    174   1    93168   5    1313324  74 dict of type
     6    194   1    86040   5    1399364  79 type
     7   1472   6    82432   5    1481796  84 function
     8    124   1    67552   4    1549348  88 dict of class
     9   1027   4    36972   2    1586320  90 __builtin__.wrapper_descriptor
```

foto by Przemysław Małkowski

From guppy import heapy

WTF?

Heapy nos dice que python ocupa de nuevo 1.7MB
Sucede que de hecho, el resto es espacio “libre”
(libre para python, no para el sistema operativo)

Confirmando la teoría

```
>>> from guppy import hpy
>>> hp = hpy()
>>> heap1 = hp.heap()
>>> # experimento
>>> heap2 = hp.heap()
>>> cosas_nuevas = heap2 - heap1
>>> del l, ll
>>> basura = heap3 - heap1
```

foto by Przemysław Malkowski

From guppy import heapy

WTF?

Vemos lo que creó nuestro experimento

Guppy.heapy

```
>>> cosas_nuevas
Partition of a set of 2588725 objects. Total size = 864642976 bytes.
 Index  Count   %     Size   % Cumulative  % Kind (class / dict of class)
     0 2588706 100 854134668  99 854134668  99 str
     1       2   0 10506304   1 864640972 100 list
     2       6   0      816   0 864641788 100 dict (no owner)
     3       2   0      676   0 864642464 100 types.FrameType
     4       2   0      272   0 864642736 100 dict of guppy.etc.Glue.Owner
     5       1   0       68   0 864642804 100 types.CodeType
     6       2   0       64   0 864642868 100 guppy.etc.Glue.Owner
     7       2   0       64   0 864642932 100 tuple
     8       1   0       32   0 864642964 100 exceptions.KeyboardInterrupt
     9       1   0       12   0 864642976 100 int
```

foto by Przemysław Malkowski

From guppy import heapy

WTF?

¿Sólo 850M de cadenas?
¿y los otros 800M para completar los 1.6G?

Guppy.heapy

```
>>> cosas_nuevas
Partition of a set of 2588725 objects. Total size = 864642976 bytes.
 Index  Count   %     Size   % Cumulative  % Kind (class / dict of class)
     0 2588706 100 854134668  99 854134668  99 str
     1       2   0 10506304   1 864640972 100 list
     2       6   0      816   0 864641788 100 dict (no owner)
     3       2   0      676   0 864642464 100 types.FrameType
     4       2   0      272   0 864642736 100 dict of guppy.etc.Glue.Owner
     5       1   0       68   0 864642804 100 types.CodeType
     6       2   0       64   0 864642868 100 guppy.etc.Glue.Owner
     7       2   0       64   0 864642932 100 tuple
     8       1   0       32   0 864642964 100 exceptions.KeyboardInterrupt
     9       1   0       12   0 864642976 100 int
```

foto by Przemysław Malkowski

From guppy import heapy

WTF?

¿Sólo 850M de cadenas?
¿y los otros 800M para completar los 1.6G?

Así se ve nuestra memoria

foto by Przemysław Malkowski

From guppy import heapy

WTF?

A ver qué pasa luego de dereferenciar todo...

Guppy.heapy

```
>>> basura
Partition of a set of 29 objects. Total size = 2520 bytes.
 Index  Count   %     Size   % Cumulative  % Kind (class / dict of class)
     0      6  21      816  32        816  32 dict (no owner)
     1      2   7      748  30       1564  62 types.FrameType
     2     10  34      364  14       1928  77 str
     3      2   7      272  11       2200  87 dict of guppy.etc.Glue.Owner
     4      2   7       80   3       2280  90 __builtin__.weakref
     5      1   3       68   3       2348  93 types.CodeType
     6      2   7       64   3       2412  96 guppy.etc.Glue.Owner
     7      2   7       64   3       2476  98 tuple
     8      1   3       32   1       2508 100 exceptions.KeyboardInterrupt
     9      1   3       12   0       2520 100 int
```

foto by Przemysław Malkowski

From guppy import heapy

¡Ahá!

¡Esto es importante!

Esos 29 objetos evitan que se pueda achicar el heap.

Guppy.heapy

```
>>> basura
Partition of a set of 29 objects. Total size = 2520 bytes.
 Index  Count   %     Size   % Cumulative  % Kind (class / dict of class)
     0      6  21      816  32        816  32 dict (no owner)
     1      2   7      748  30       1564  62 types.FrameType
     2     10  34      364  14       1928  77 str
     3      2   7      272  11       2200  87 dict of guppy.etc.Glue.Owner
     4      2   7       80   3       2280  90 __builtin__.weakref
     5      1   3       68   3       2348  93 types.CodeType
     6      2   7       64   3       2412  96 guppy.etc.Glue.Owner
     7      2   7       64   3       2476  98 tuple
     8      1   3       32   1       2508 100 exceptions.KeyboardInterrupt
     9      1   3       12   0       2520 100 int
```

foto by Przemysław Malkowski

From guppy import heapy

¡Ahá!

¡Esto es importante!

Esos 29 objetos evitan que se pueda achicar el heap.

Normalmente el heap varía así:

foto by Przemysław Malkowski

From guppy import heapy

¡Ahá!

¡Esto es importante!

Esos 29 objetos evitan que se pueda achicar el heap.

Normalmente el heap varía así:

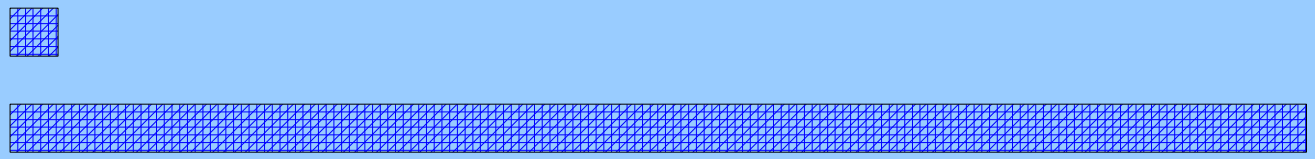


foto by Przemysław Malkowski

From guppy import heapy

¡Ahá!

¡Esto es importante!

Esos 29 objetos evitan que se pueda achicar el heap.

Normalmente el heap varía así:

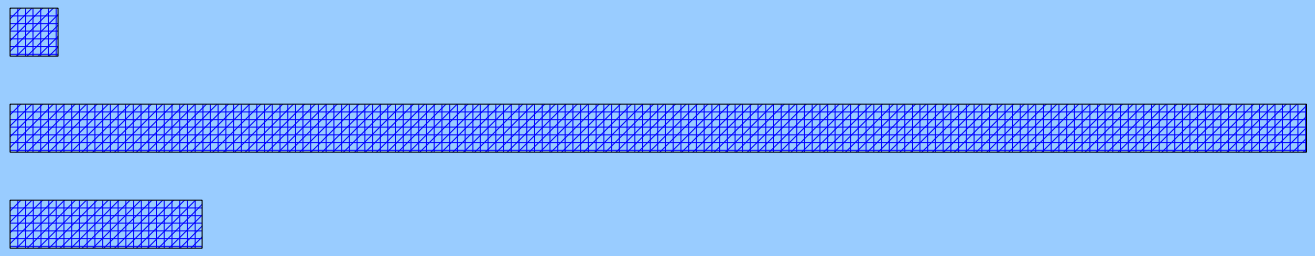


foto by Przemysław Malkowski

From guppy import heapy

¡Ahá!

¡Esto es importante!

Esos 29 objetos evitan que se pueda achicar el heap.

Normalmente el heap varía así:

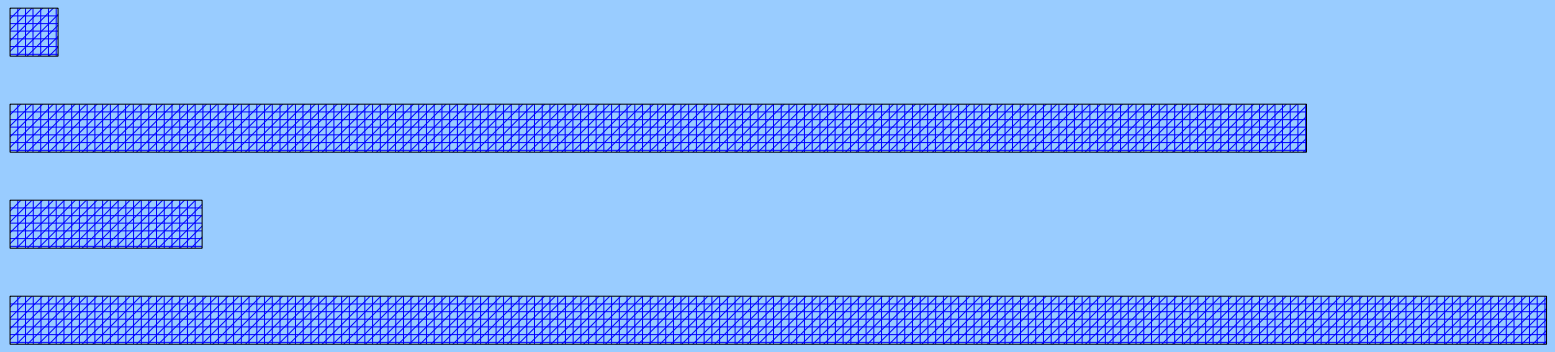


foto by Przemysław Malkowski

From guppy import heapy

¡Ahá!

¡Esto es importante!

Esos 29 objetos evitan que se pueda achicar el heap.

Normalmente el heap varía así:

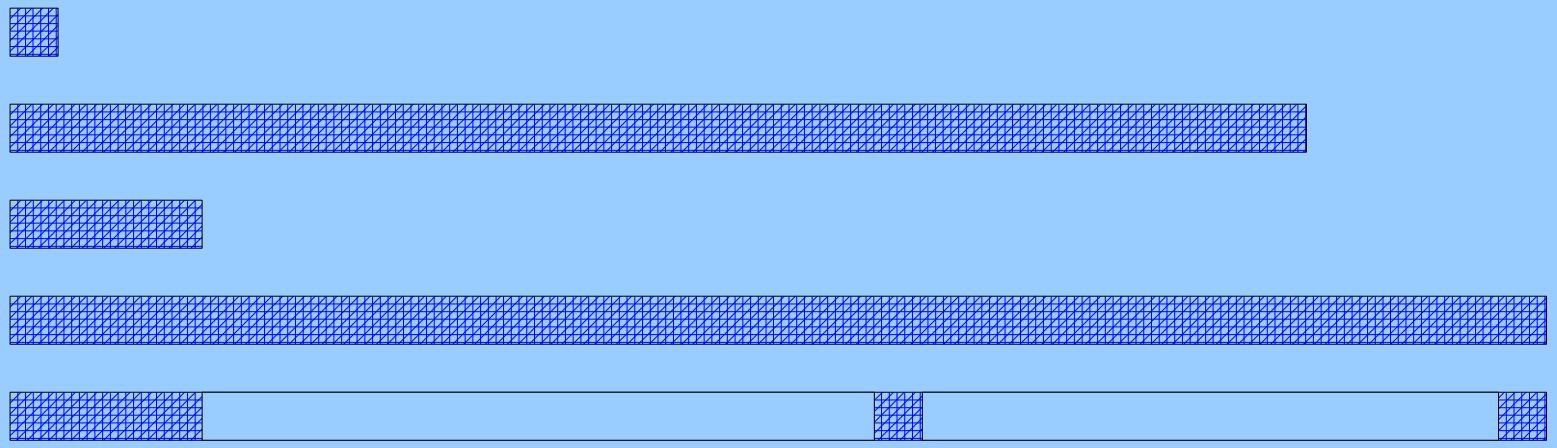


foto by Przemysław Malkowski

From guppy import heapy

¡Ahá!

¡Esto es importante!

Esos 29 objetos evitan que se pueda achicar el heap.

Normalmente el heap varía así:

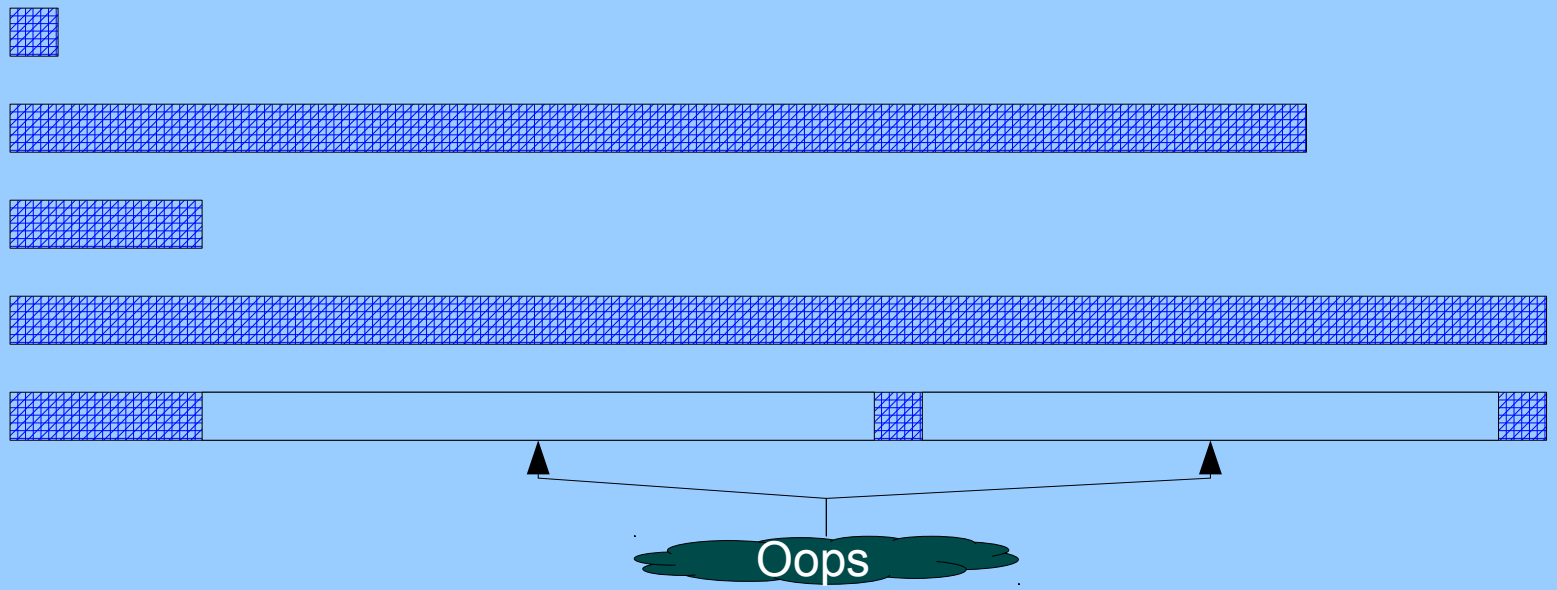
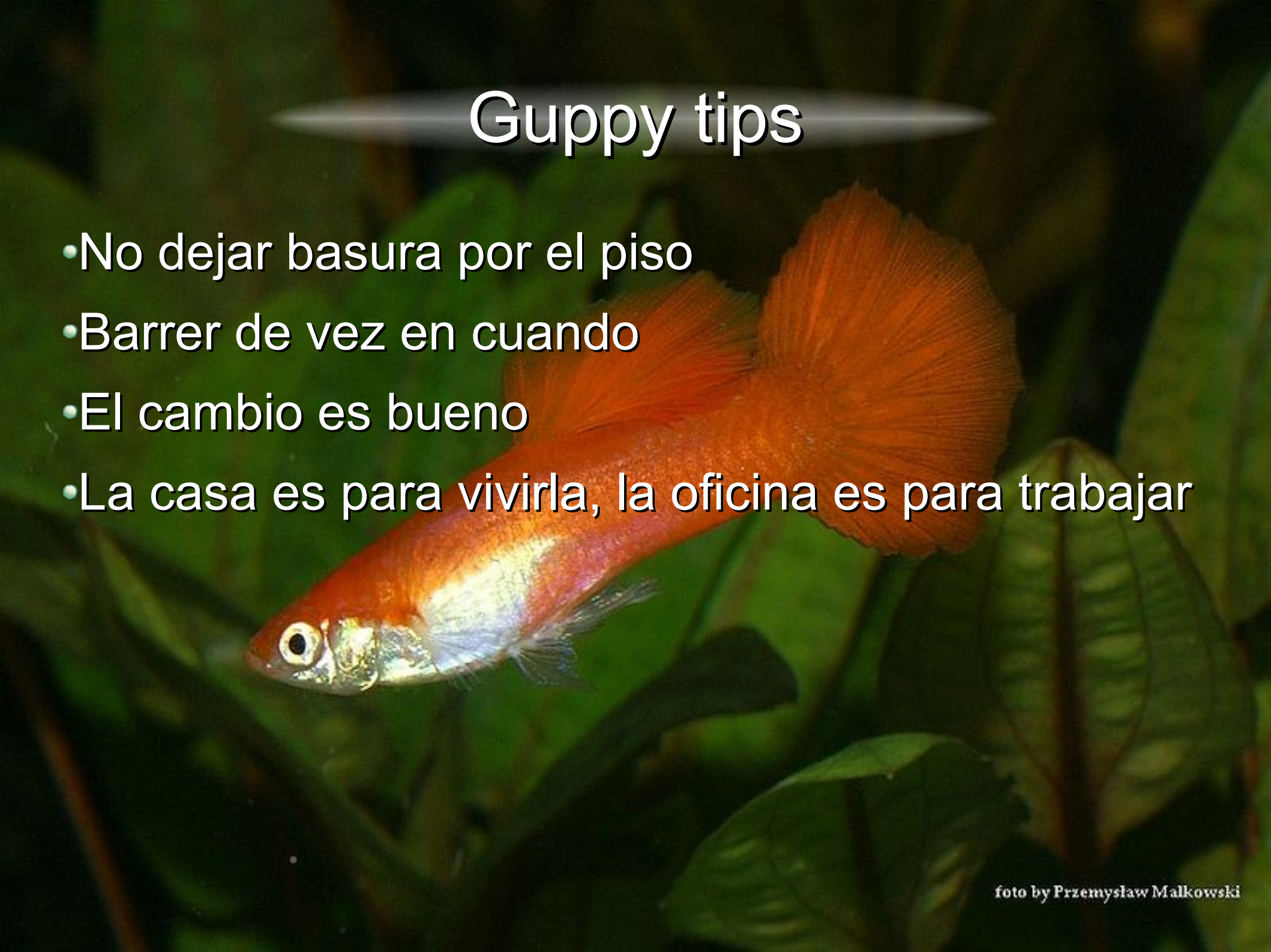


foto by Przemysław Malkowski

Guppy tips

- No dejar basura por el piso
- Barrer de vez en cuando
- El cambio es bueno
- La casa es para vivirla, la oficina es para trabajar



Guppy tips

- No dejar basura por el piso
  - Si se van a crear muchos objetos pequeños, crear los *persistentes* primero, y los *transientes* al final.
  - Siempre que sea posible, preferir pocos objetos grandes a muchos objetos pequeños:
    - Listas de strings → strings separados por comas*
    - Listas de números → array.array o numpy
- Barrer de vez en cuando
- El cambio es bueno
- La casa es para vivirla, la oficina es para trabajar

foto by Przemysław Malkowski

* o pipes, o enter, o lo que sea

Guppy tips

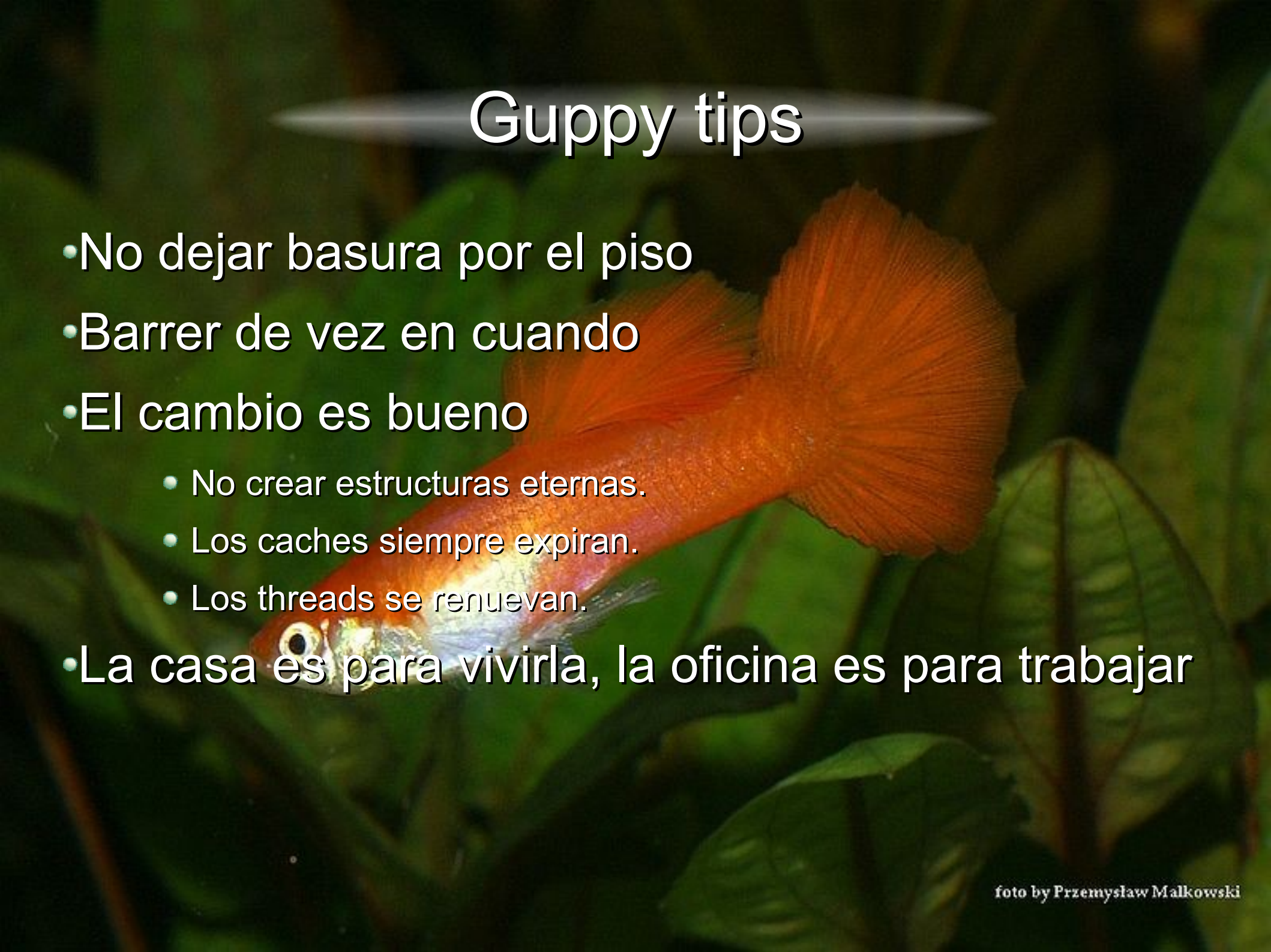
- No dejar basura por el piso
- Barrer de vez en cuando
  - Si se mantienen caches con expiración, limpiar el caché regularmente para quitar elementos expirados
  - A veces se puede “desfragmentar” la memoria, reconstruyendo estructuras *persistentes* como los cachés
- El cambio es bueno
- La casa es para vivirla, la oficina es para trabajar

foto by Przemysław Malkowski

* o pipes, o enter, o lo que sea

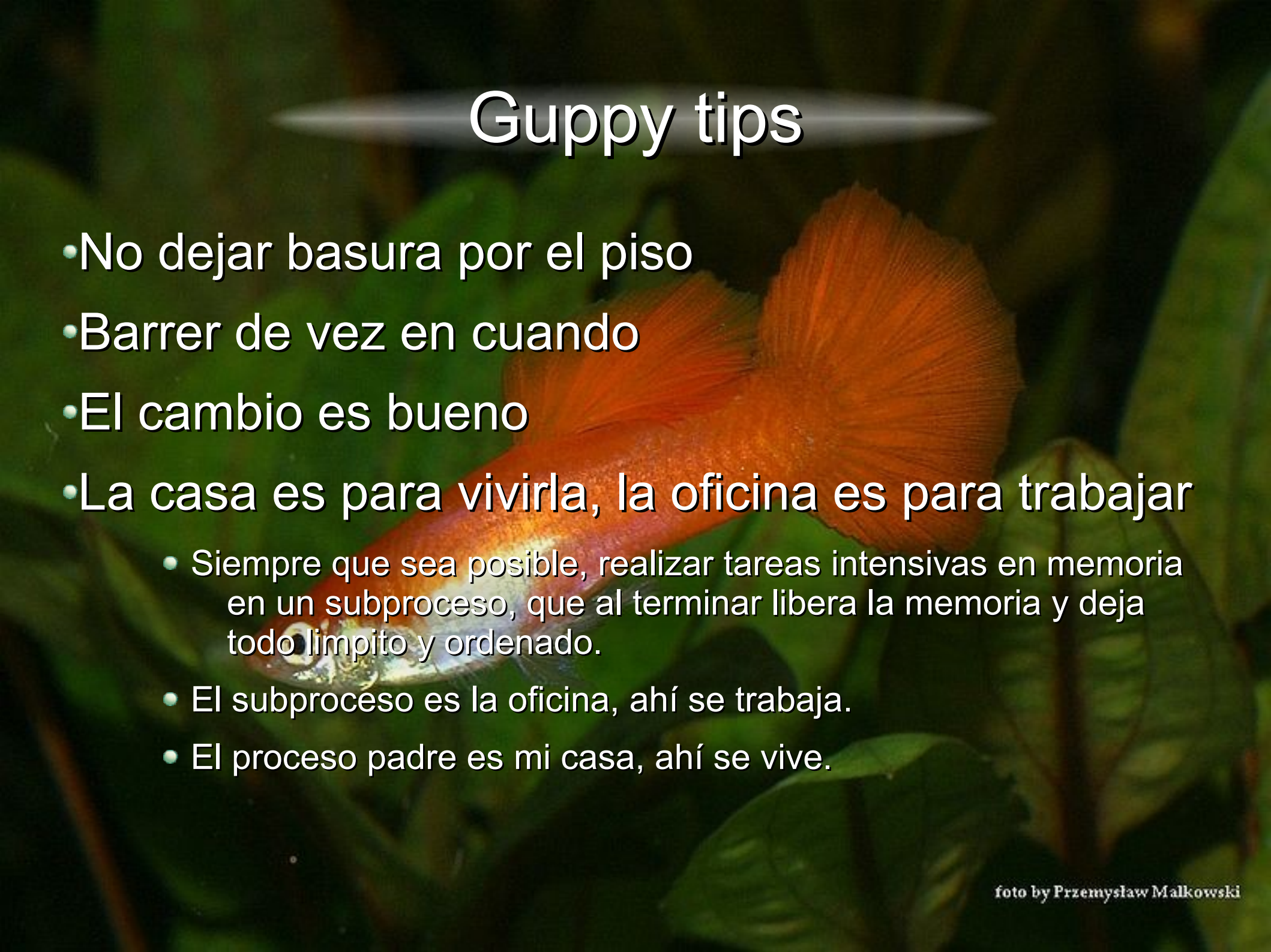
Guppy tips

- No dejar basura por el piso
- Barrer de vez en cuando
- El cambio es bueno
  - No crear estructuras eternas.
  - Los caches siempre expiran.
  - Los threads se renuevan.
- La casa es para vivirla, la oficina es para trabajar



Guppy tips

- No dejar basura por el piso
- Barrer de vez en cuando
- El cambio es bueno
- La casa es para vivirla, la oficina es para trabajar
  - Siempre que sea posible, realizar tareas intensivas en memoria en un subproceso, que al terminar libera la memoria y deja todo limpito y ordenado.
  - El subproceso es la oficina, ahí se trabaja.
  - El proceso padre es mi casa, ahí se vive.




¿Preguntas?