Python ồ

- لجلمد -

wklken

ㄣ

- Ծ
- ㅗ
- int
- string
- list
- tuple
- dict

„ Ẫ ı Ʊ

Python [illegible] ?

Ʊ - ẩ

- Python ɾ Ʊ ẩ ı ˉ :ˆ̉ (int), ˆ̉ (list/dict)
- ɾˆ̉ מ ı PyObject ˜ PyVarObject
- ˉ :ˆ̉ מ „ : ṛ ᾠ ץ ˆ̉ מ PyObject_HEAD

Ծ - PyObject_HEAD

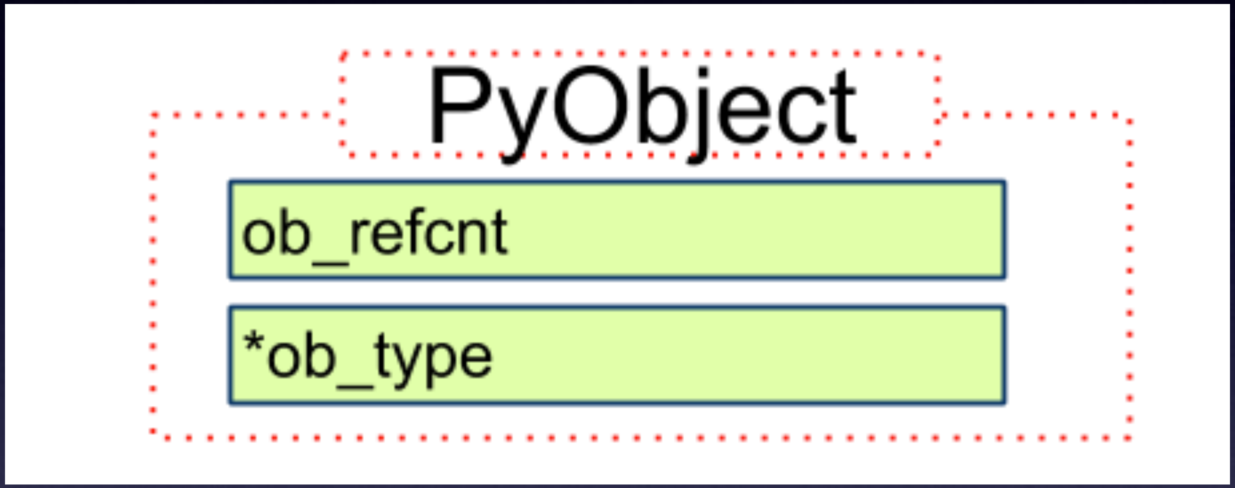
```
#define PyObject_HEAD                  \
        _PyObject_HEAD_EXTRA                  \
        Py_ssize_t ob_refcnt;            \
        struct _typeobject *ob_type;
```

- _PyObject_HEAD_EXTRA ṅ ώ̤ - ל ѵ ◘ طمط ὲ
- ob_refcnt ٹ لمد
- *ob_type بج ώ̤ ┸ Ծ بج , V̤̂ E : Ծ ┸ !

- 获取引用计数的方式，sys.getrefcount，以及为何数字不对?
- 获取类型的函数
-

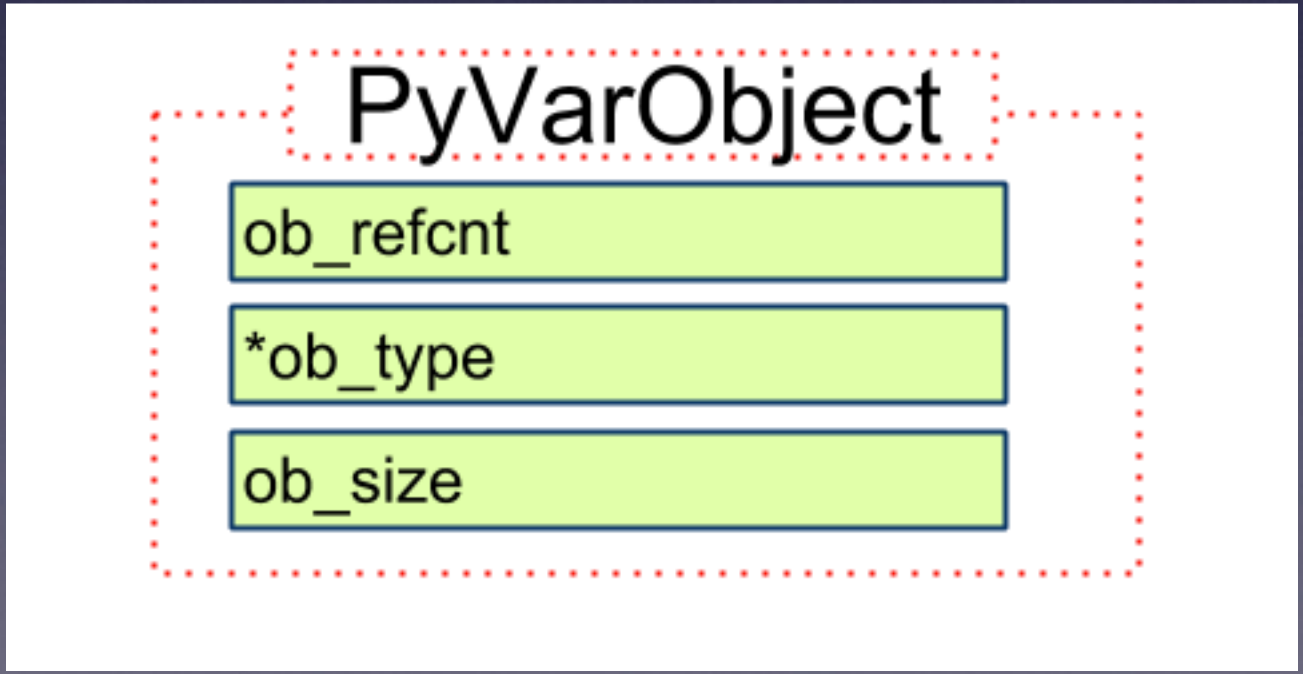
ნ - PyObject

```
typedef struct _object {
    PyObject_HEAD
} PyObject;
```

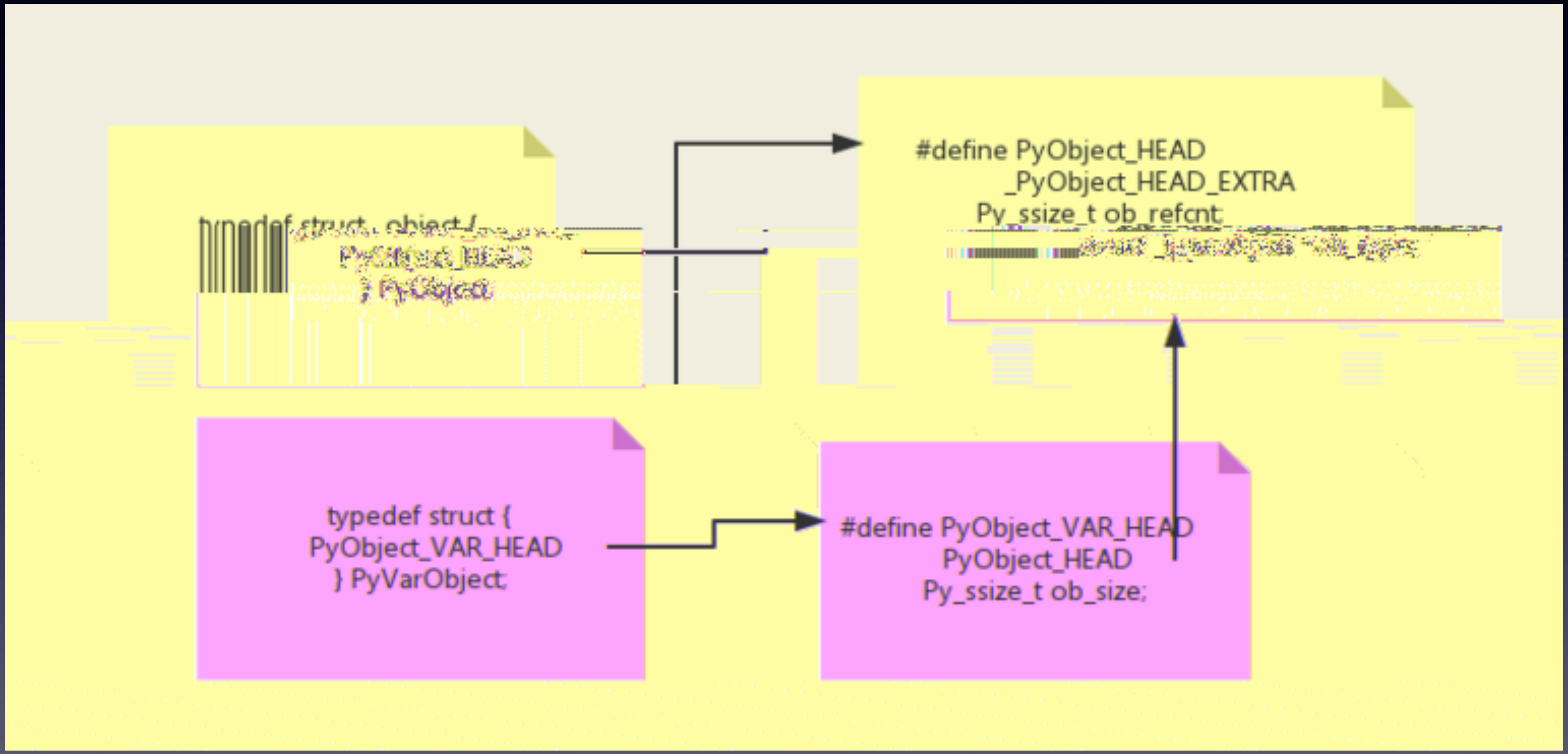


ნ - PyVarObject

```
typedef struct {
    PyObject_VAR_HEAD
} PyVarObject;

#define PyObject_VAR_HEAD               \
  PyObject_HEAD                         \
  Py_ssize_t ob_size; /* Number of items in variable part */
```



Ꝺ - ɐ Ṝ



```
typedef struct _object {
    PyObject_HEAD
} PyObject;
```

```
#define PyObject_HEAD
        _PyObject_HEAD_EXTRA
        Py_ssize_t ob_refcnt;
        struct _typeobject *ob_type;
```

```
typedef struct {
    PyObject_VAR_HEAD
} PyVarObject;
```

```
#define PyObject_VAR_HEAD
        PyObject_HEAD
        Py_ssize_t ob_size;
```

ŏ - Ẑ : تخی

```
#define Py_REFCNT(ob)          (((PyObject*)(ob))->ob_refcnt)
读取引用计数

#define Py_TYPE(ob)            (((PyObject*)(ob))->ob_type)
获取对象类型

#define Py_SIZE(ob)            (((PyVarObject*)(ob))->ob_size)
读取元素个数：(len)
```

Py_INCREF(op). 增加对象引用计数

Py_DECREF(op) 减少对象引用计数, 如果计数为0, 调用_Py_Dealloc

_Py_Dealloc(op) 调用对应类型的 tp_dealloc 方法(每种类型回收行为不一样的, 各种缓存池机制, 后面看)

æ

„ : ვ ĭ

```
>>> a = 1
>>> a
1

>>> type(a)
<type 'int'>

# 等价的两个:
>>> type(type(a))
<type 'type'>
>>> type(int)
<type 'type'>

# 还是等价的两个:
>>> type(type(type(a)))
<type 'type'>
>>> type(type(int))
<type 'type'>
```

- 基本类型对象的 类型是type
- type 的类型是 type
-

⊥ - PyTypeObject

ή ώ̈ ﺒﻬ Ձ,, : intƠ ﺗﻰ ﻨﺠﻲPython ⊥ ề

```
typedef struct _typeobject {
 /* MARK: base, 注意, 是个变长对象*/
  PyObject_VAR_HEAD
  const char *tp_name; /* For printing, in format "<module>.<name>" */ //类型名
  Py_ssize_t tp_basicsize, tp_itemsize; /* For allocation */ // 创建该类型对象时分配的内存空间
大小

 // 一堆方法定义, 函数和指针
  /* Methods to implement standard operations */
  printfunc tp_print;
  hashfunc tp_hash;

  /* Method suites for standard classes */
  PyNumberMethods *tp_as_number;   // 数值对象操作
  PySequenceMethods *tp_as_sequence; // 序列对象操作
  PyMappingMethods *tp_as_mapping; // 关联对象操作

  // 一堆属性定义
  ....
} PyTypeObject;
```

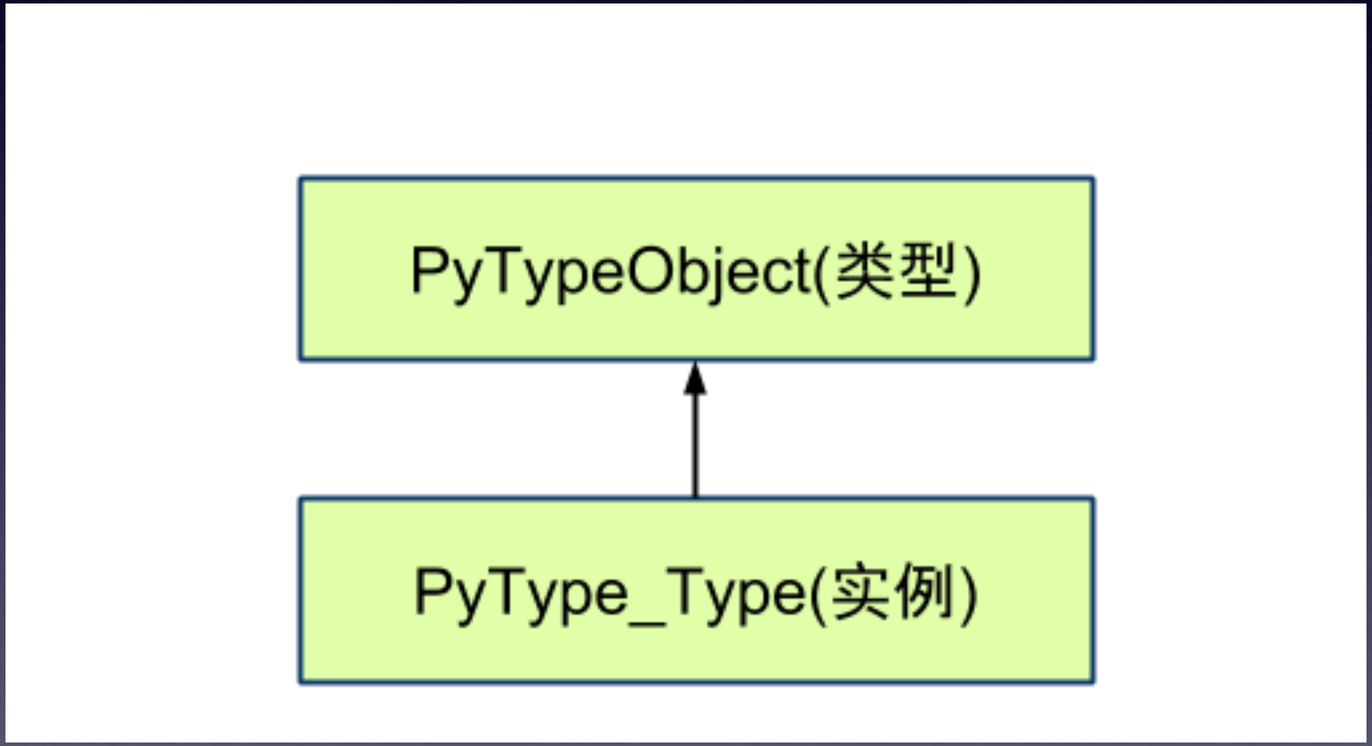
⊥ - PyType_Type

```
PyTypeObject PyType_Type = {
  PyVarObject_HEAD_INIT(&PyType_Type, 0)
  "type",                              /* tp_name */
  sizeof(PyHeapTypeObject),                  /* tp_basicsize */
  sizeof(PyMemberDef),                     /* tp_itemsize */
  (destructor)type_dealloc,                 /* tp_dealloc */

  // type ƃ 象的 تخس 法和 ̈ ƌ 性 ä ặ ⅃ 始化 ɛ̆ ḏ
  .....

};
```

- 实例化, tp_name = ‘type’
- 注意, PyVarObject_HEAD_INIT

ᅩ - PyType_Type

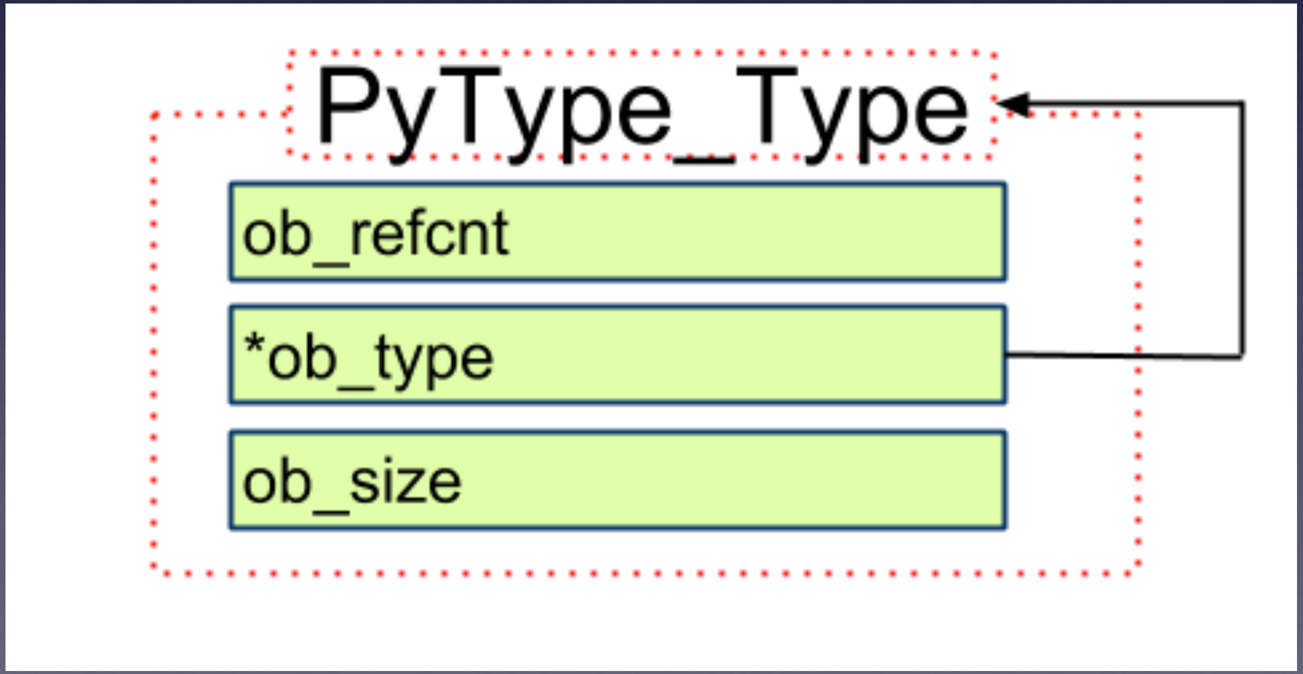
PyType_Type PyTypeObject ̉ ʒ ἕ Ϭ



⊥ - PyType_Type

PyVarObject_HEAD_INIT, 这个方法在 Include/object.h 中，等价于

```
    ob_refcnt = 1
    *ob_type = &PyType_Type
    ob_size = 0
```



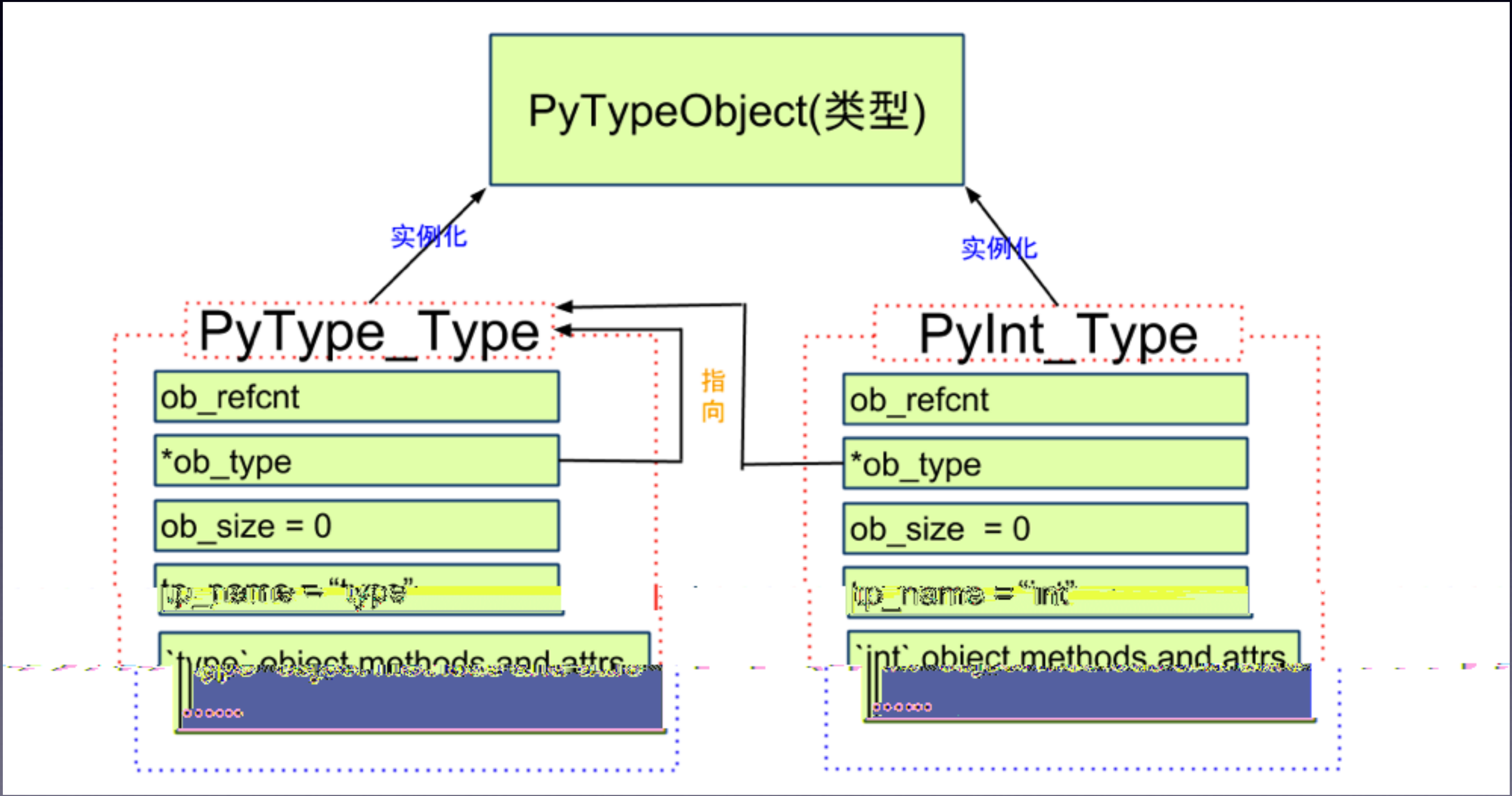
⊥ - PyType_Type

```
# 1. int 的 类型 是`type`
>>> type(int)
<type 'type'>

# 2. type 的类型 还是`type`, 对应上面说明第二点
>>> type(type(int))
<type 'type'>
```

⊥ - PyInt_Type

```
PyTypeObject PyInt_Type = {
  PyVarObject_HEAD_INIT(&PyType_Type, 0)
  "int",
  sizeof(PyIntObject),
  0,

  // int类型的相关方法和属性值
  ....

  (hashfunc)int_hash,                    /* tp_hash */

};
```

⊥ - PyInt_Type


PyTypeObject(类型)
实例化
实例化
指向
PyType_Type
ob_refcnt
*ob_type
ob_size = 0
tp_name = "type"
`type` object methods and attrs
PyInt_Type
ob_refcnt
*ob_type
ob_size = 0
tp_name = "int"
`int` object methods and attrs

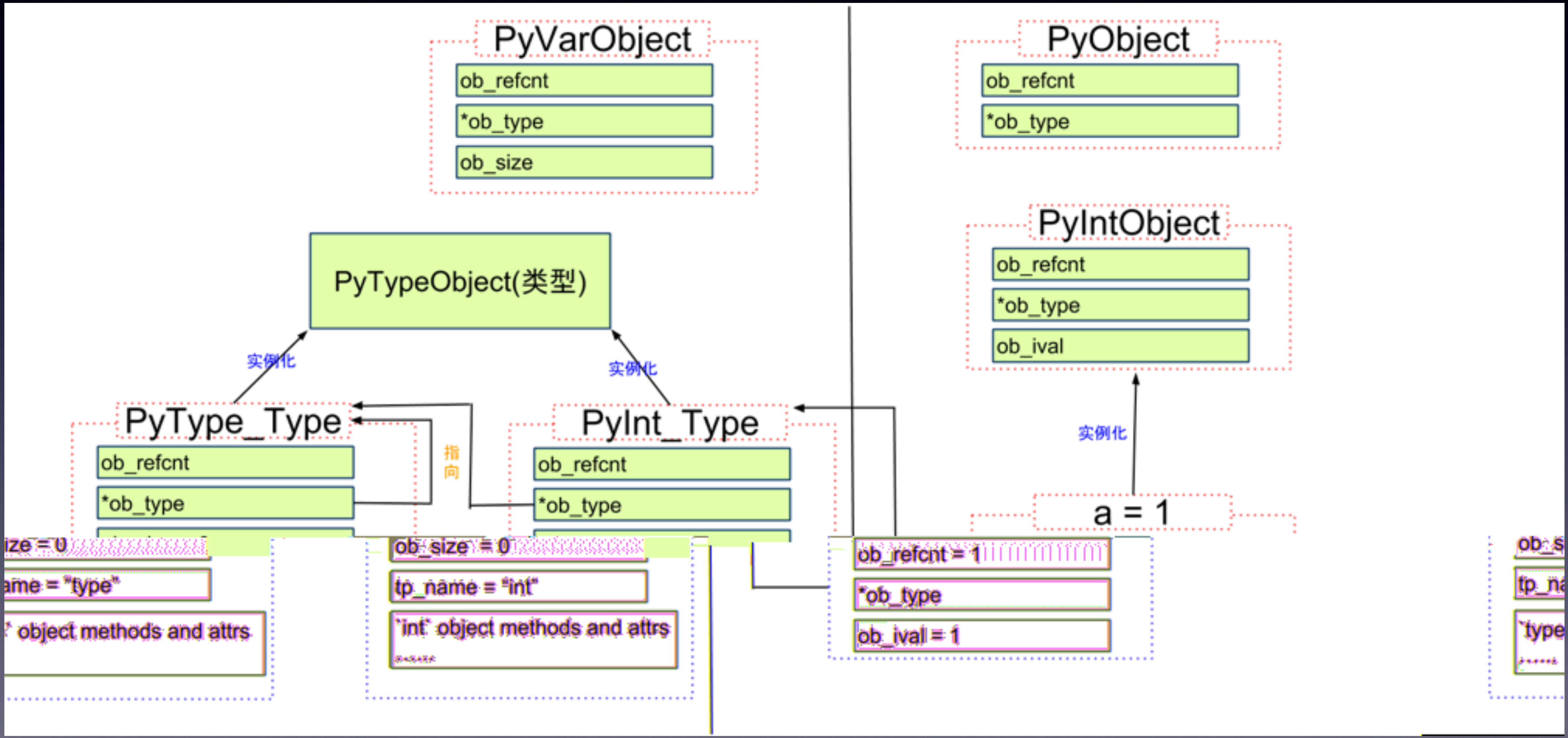
⊥ - PyInt_Type

```
>>> type(1)
<type 'int'>

>>> type(type(1))
<type 'type'>
```

⊥ - PyIntObject

```
typedef struct {
    PyObject_HEAD
    long ob_ival;
} PyIntObject;
```

⊥ - تىPyIntObject



[illegible]

1. 一切都是对象

2. PyType_Type / PyInt_Type / PyString_Type .... 等
这些是"类型"对象，可以认为是同级，都是PyTypeObject这种"类型"的实例！

3. 虽然是同级，
但是其他PyXXX_Type，其类型指向 PyType_Type
PyType_Type 的类型指向自己，它是所有类型的"类型"

4. PyTypeObject 是一个变长对象

5. 每个object，例如PyIntObject都属于一种"类型"
object初始化时进行关联

[illegible]

```
>>> hash(1)
1
>>> hash("abc")
1453079729188098211
```

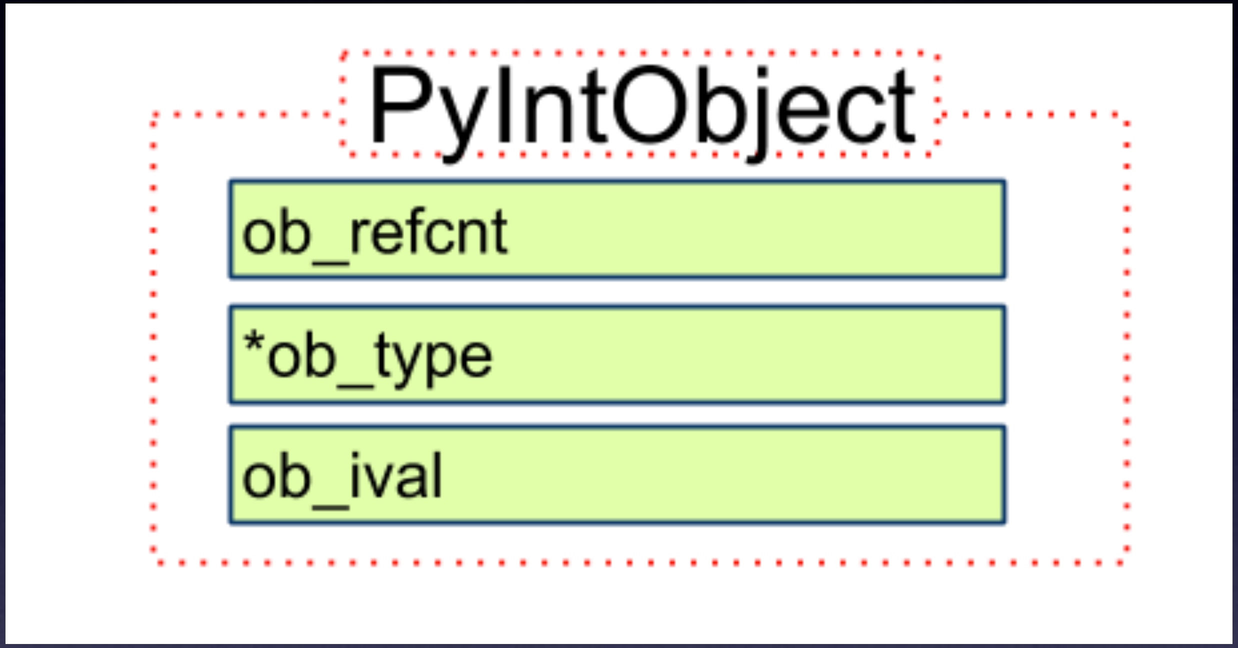
```
PyTypeObject PyInt_Type = {
    ...
    (hashfunc)int_hash,                    /* tp_hash */
    ...
}

PyTypeObject PyString_Type = {
    ...
    (hashfunc)string_hash,                 /* tp_hash */
    ...
}
```

```
object -> ob_type -> tp_hash
```

INT

INT - PyIntObject

```
typedef struct {
    PyObject_HEAD
    long ob_ival;
} PyIntObject;
```



INT - PyIntObject

```
>>> a = -5
>>> b = -5
>>> id(a) == id(b)
True
>>> a = -6
>>> b = -6
>>> id(a) == id(b)
False

>>> a = 256
>>> b = 256
>>> id(a) == id(b)
True
>>> a = 257
>>> b = 257
>>> id(a) == id(b)
False

#在python2.x中，对于大的序列生成，建议使用 xrange(100000) 而不是range(100000), why?
```

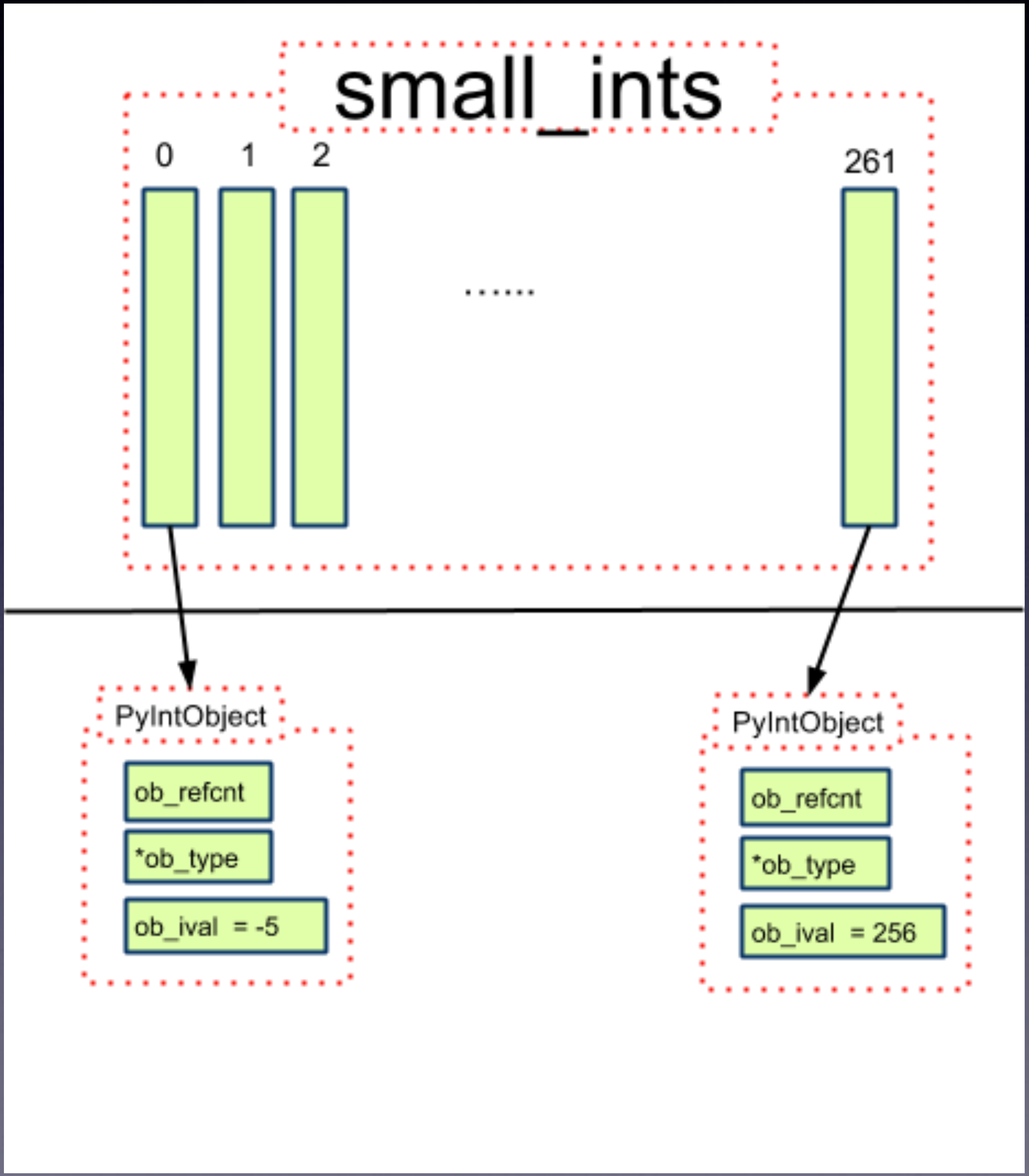
INT - ∩ محه لمد Ծ Ṿ

```
#ifndef NSMALLPOSINTS
#define NSMALLPOSINTS         257
#endif

#ifndef NSMALLNEGINTS
#define NSMALLNEGINTS         5
#endif

#if NSMALLNEGINTS + NSMALLPOSINTS > 0
/* References to small integers are saved in this array
   so that they can be shared.
   The integers that are saved are those in the range
   -NSMALLNEGINTS (inclusive) to NSMALLPOSINTS (not inclusive).
*/

static PyIntObject *small_ints[NSMALLNEGINTS + NSMALLPOSINTS];
#endif
```

INT - ᑎ ﻤﺤﻤ ﻠﻤﺪ Ơ Ṿ

ᑎ ﻤﺤﻤ ﻠﻤﺪ Ơ $ „ : PyIntObjectﺑﺠ ﻠﻤﺪ (ﻹ ﺑﺠ ﻠﻤﺪ
), Ⴗ ᑎ =257+5=262, ㄐ [-5, 257) ﻹ ⋆ ῇ ﻳ .
῞O : ﻠﻤﺪ ῎A ᾳ̃ ᴇ 262 : ﺑﺠ ῴ PyIntObject ﺑﺠ

INT - ∩ محم لمد Ŏ Ỵ



INT - ặ ˧ ɛ̋

```
#if NSMALLNEGINTS + NSMALLPOSINTS > 0
if (-NSMALLNEGINTS <= ival && ival < NSMALLPOSINTS) {

   v = small_ints[ival + NSMALLNEGINTS];
    // 引用+1
    Py_INCREF(v);

   ……

    // 返回
    return (PyObject *) v;
}
#endif
```

INT - محم لمد Ʊ Ṿ

∩ محم لمد Ʊ , ῷ „ : بج لمد , ṟ Ʊ ƚ محم لمد
Ʊ

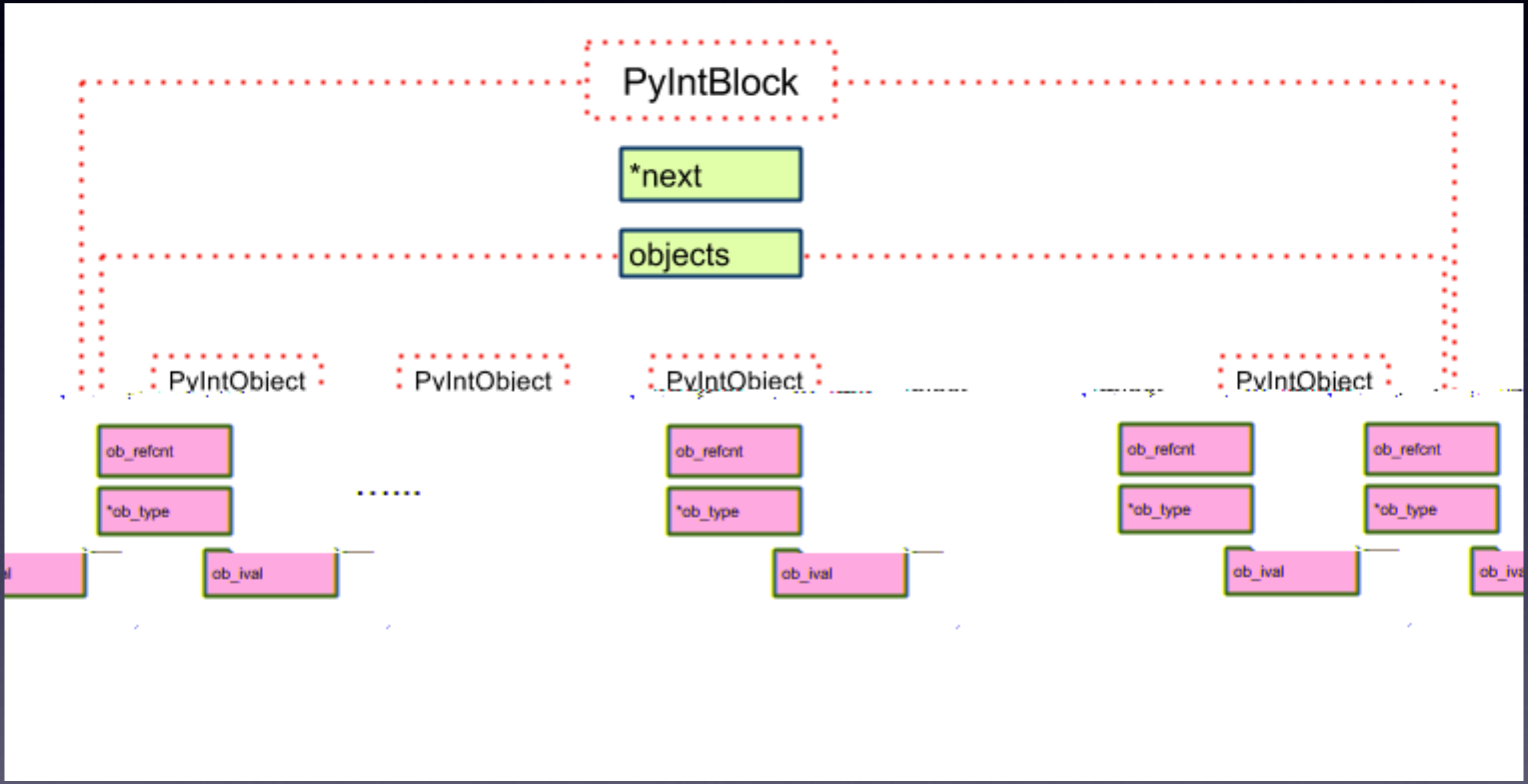
```
// ∩ محم لمد Ʊ象池ặ ⊣ ἕ过程, ه环, 逐„一生ى
for (ival = -NSMALLNEGINTS; ival < NSMALLPOSINTS; ival++) {
      if (!free_list && (free_list = fill_free_list()) == NULL)
              return 0;

   /* PyObject_New is inlined */
    v = free_list;
    free_list = (PyIntObject *)Py_TYPE(v);
    PyObject_INIT(v, &PyInt_Type);
    v->ob_ival = ival;

    // 放 محم ế لمد组里
    small_ints[ival + NSMALLNEGINTS] = v;
}
```

free_list

INT - محم لمد Ŏ Ṿ

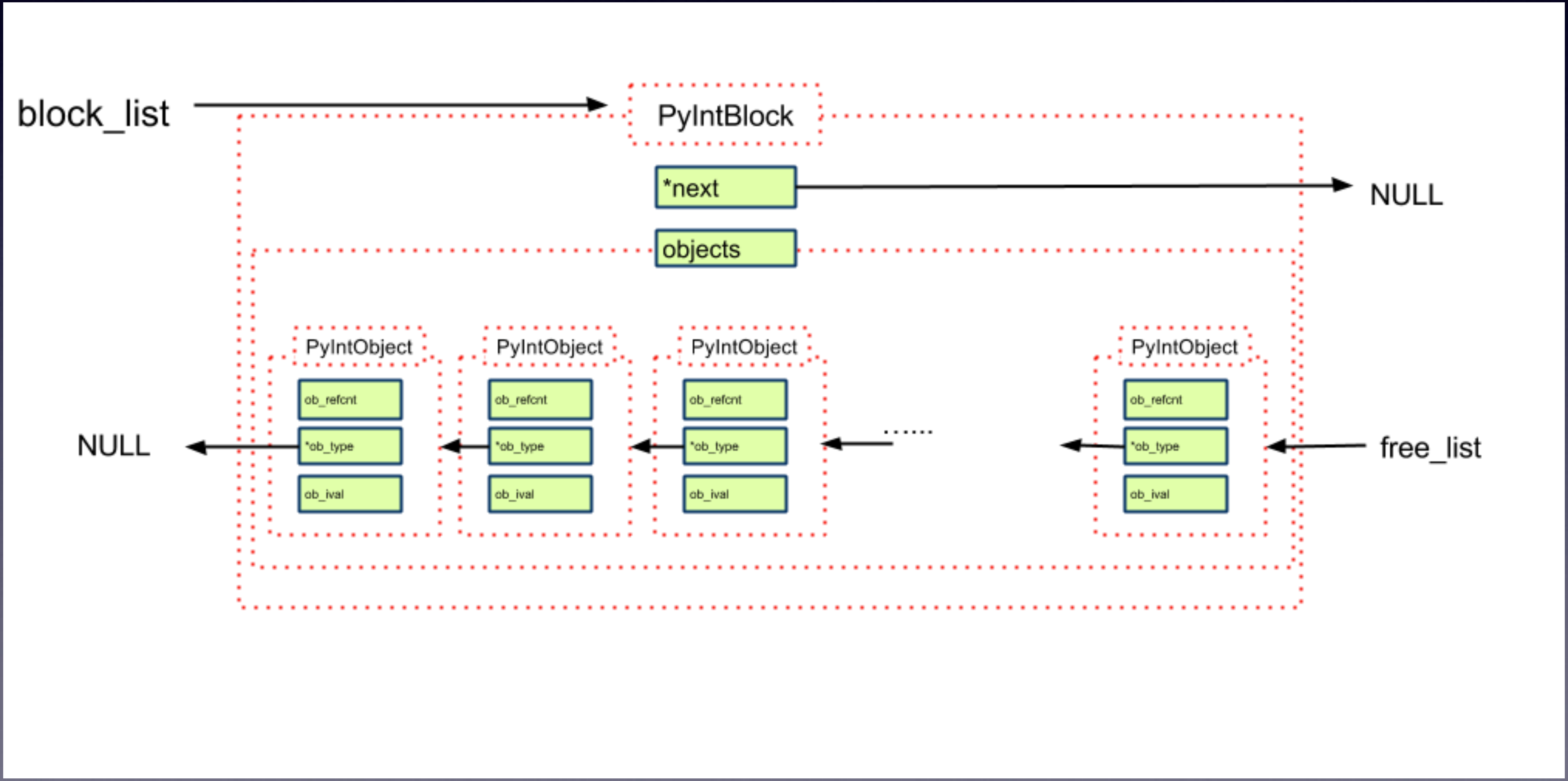
```
#define BLOCK_SIZE      1000    /* 1K less typical malloc overhead */
#define BHEAD_SIZE      8       /* Enough for a 64-bit pointer */
#define N_INTOBJECTS    ((BLOCK_SIZE - BHEAD_SIZE) / sizeof(PyIntObject))


struct _intblock {
    struct _intblock *next;
    PyIntObject objects[N_INTOBJECTS];
};

typedef struct _intblock PyIntBlock;
```

INT - محم لمد Ō Ṿ



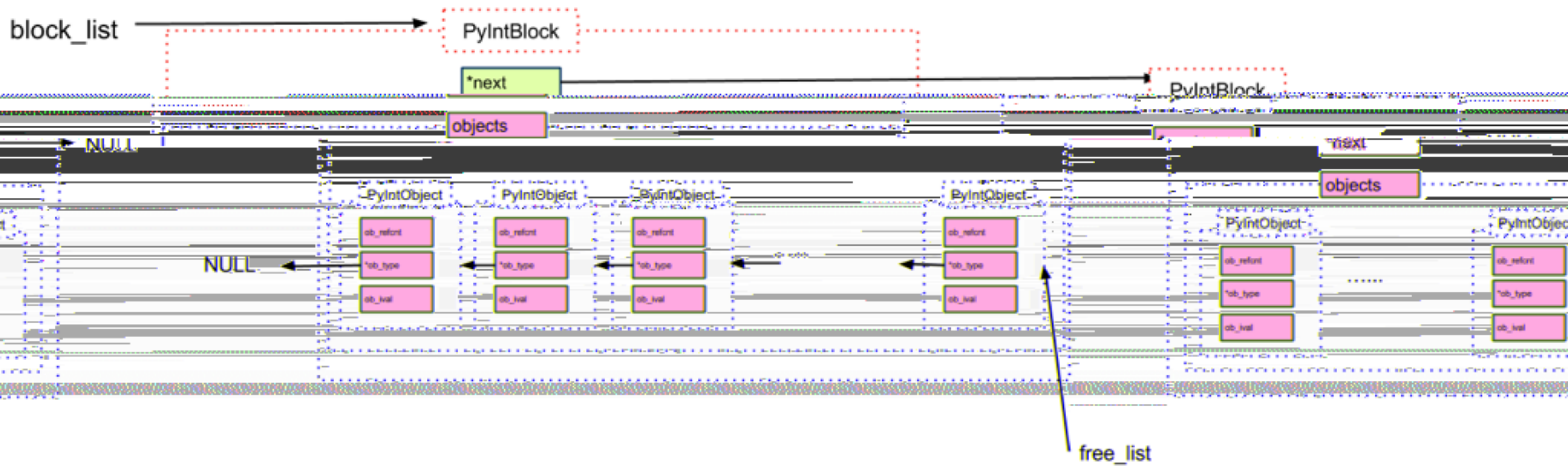
INT - محم لمد ٥ V̤

b جحي : رر پ تخي



INT - محم لمد Ծ Ṿ

ﭽ„ : block ̃ E | ῴ, ῞Ofree_list=NULL, ﺠﻲ ﺗﺨﻲ ﭘ ᾀ„
: PyIntBlock



INT - ׀ л ם xrange

int ◘ طمح جعي ḅ , مي معجم ◘ ế free_list

PyIntBlocks are never returned to the system before shutdown (PyInt_Fini).

̏O, PyIntBlock طح ṩ ű̋ , Ł Python ░ ׀ ỏ ,
˚ δ معجم

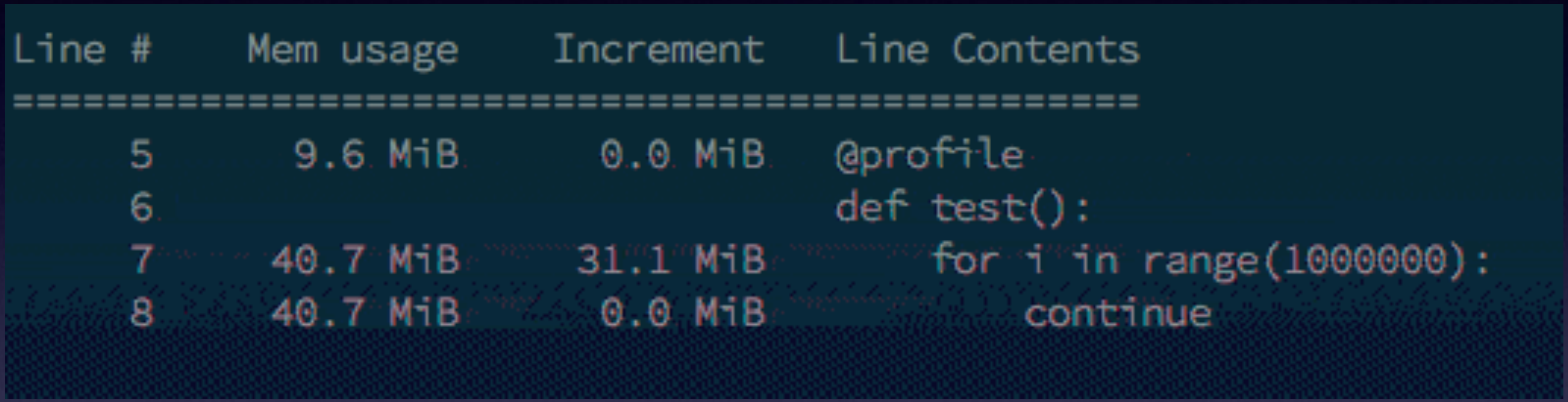
طح æ , ə̢ range(100000), ̦̦ ة ̦̦تى : ˥ Ả ,
ώ̤ , ن E , ŧ لمحمحمد ̏l Ł .
xrange, ̦̦ ̦̦ : , ą̃̏ ̖ δ ̏l ̦̦ : intobject, ׀ ώ̤ ◘ طمح,
ţ ̦̦ : ....

为什么python3直接使用range

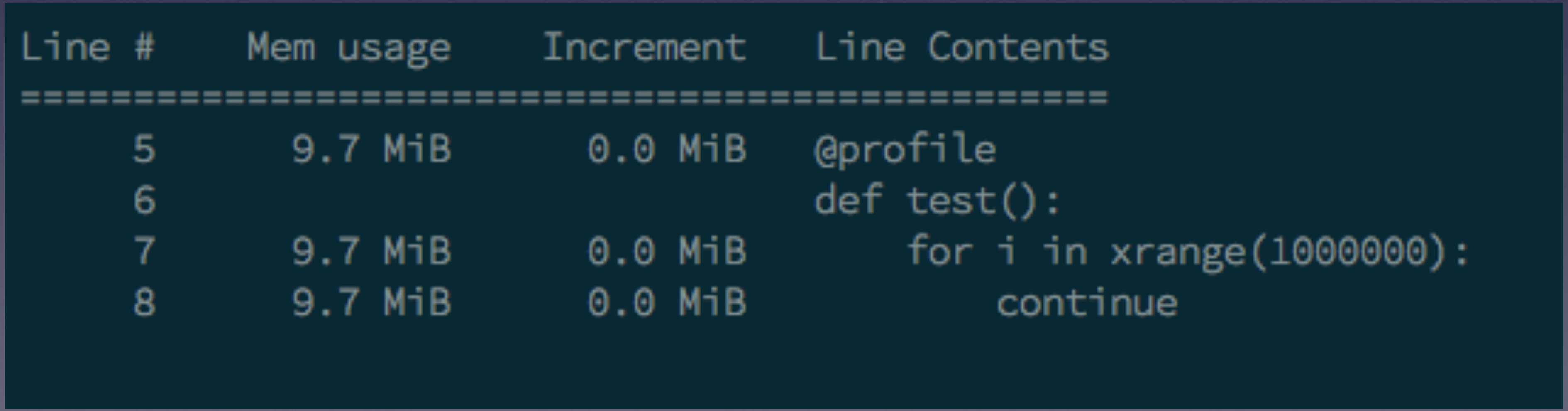
`xrange` was not removed: it was renamed to `range`, and the 2.x `range` is what was removed.

INT - ı л □ xrange

```
python -m memory_profiler test.py
```



```
Line #    Mem usage    Increment   Line Contents
================================================
     5      9.6 MiB      0.0 MiB   @profile
     6                             def test():
     7     40.7 MiB     31.1 MiB       for i in range(1000000):
     8     40.7 MiB      0.0 MiB           continue
```



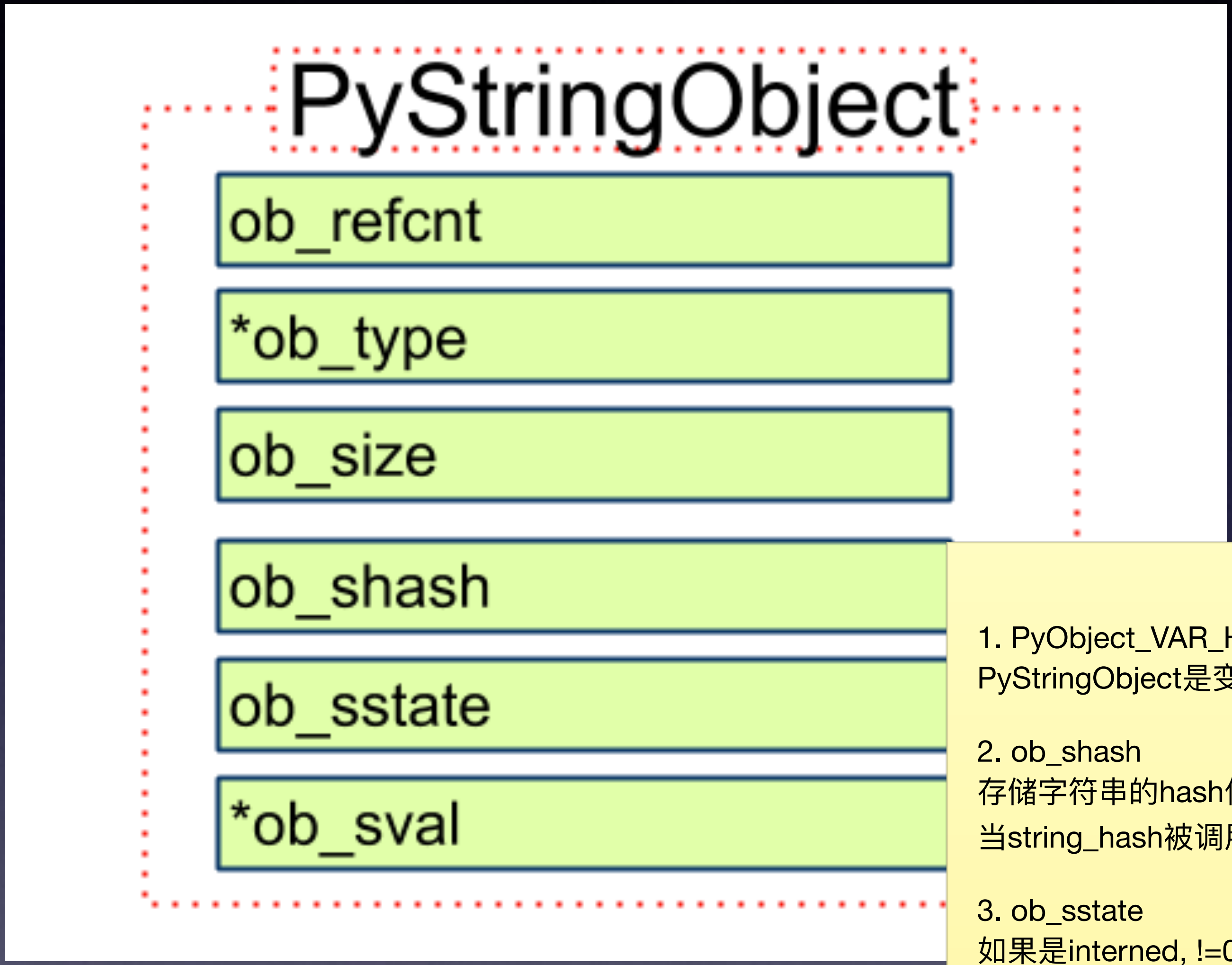
```
Line #    Mem usage    Increment   Line Contents
================================================
     5      9.7 MiB      0.0 MiB   @profile
     6                             def test():
     7      9.7 MiB      0.0 MiB       for i in xrange(1000000):
     8      9.7 MiB      0.0 MiB           continue
```

String

String - PyStringObject

```
typedef struct {
  PyObject_VAR_HEAD
  long ob_shash;
  int ob_sstate;
  char ob_sval[1];

  /* Invariants:
   *     ob_sval contains space for 'ob_size+1' elements.
   *     ob_sval[ob_size] == 0.
   *     ob_shash is the hash of the string or -1 if not computed yet.
   *     ob_sstate != 0 iff the string object is in stringobject.c's
   *       'interned' dictionary; in this case the two references
   *       from 'interned' to this object are *not counted* in ob_refcnt.
   */
} PyStringObject;
```

String - PyStringObject



1. PyObject_VAR_HEAD
PyStringObject是变长对象, 比定长对象多了一个ob_size字段

2. ob_shash
存储字符串的hash值, 如果还没计算等于-1
当string_hash被调用, 计算结果会被保存到这个字段一份, 后续不再进行计算

3. ob_sstate
如果是interned, !=0, 否则=0
interned后面说

4. char ob_sval[1]
字符指针指向一段内存, char数组指针, 指向一个ob_size+1大小数组(c中字符串最后要多一个字符`\0`表字符串结束)

String - interned ề

```
>>> a = "hello"
>>> b = "hello"
>>> id(a) == id(b)
True
>>>
>>> c = ''.join(['h', 'ello'])
>>> id(a) == id(c)
False
>>> a = "hello world"
>>> b = "hello world"
>>> id(a) == id(b)
False
>>>
>>> a = intern("hello world")
>>> b = intern("hello world")
>>> id(a) == id(b)
True
```

String - interned ề

```
/* This dictionary holds all interned strings.  Note that references to
strings in this dictionary are *not* counted in the string's ob_refcnt.
When the interned string reaches a refcnt of 0 the string deallocation
function will delete the reference from this dictionary.

Another way to look at this is that to say that the actual reference
count of a string is:  s->ob_refcnt + (s->ob_sstate?2:0)
*/
static PyObject *interned; //بج , بج ώ̰PyDictObject
```

String - interned ề

```
// ƚ interned ȕ ṡ ɾ پ ű ƚ , ˜ 修 ﻄﻤﻣ , 返 ◘ intern 独 w
t = PyDict_GetItem(interned, (PyObject *)s);
if (t) {
    Py_INCREF(t);
    Py_DECREF(*p);
    *p = t;
    return;
}

// ƚ interned ȕ ṡ ɾ ˚ 不 ű ƚ , ﻣﻌﻤ 放进 ˝Ω
if (PyDict_SetItem(interned, (PyObject *)s, (PyObject *)s) < 0) {
    PyErr_Clear();
    return;
}
```

„ ﻗﻤﺪ ȕ ℵ intern, ᵟ python ᶻ ű ế ȕ ṡ ɾ , ﻣﺤﻤ : python , ɾ

. ᴰ „ : Ƃ . ˛ ὦ ȕ ℵ ţ Ṕ, ˚ ᵟ ¯ Ặ ﭘ Ƃ .

String - interned　ề

```
#define NAME_CHARS \
    "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ_abcdefghijklmnopqrstuvwxyz"
```

```
        if (!all_name_chars((unsigned char *)PyString_AS_STRING(v)))
            continue;
        PyString_InternInPlace(&PyTuple_GET_ITEM(consts, i));
```

什么情况才会走interned, 什么情况不interned TODO: find the code

只包含下划线、数字、字母的字符串才会被intern. 拼接产生的字符串不算

String - ů̈ Ṿ

```
UCHAR_MAX ک ̈Ạ 相 Ṛ

static PyStringObject *characters[UCHAR_MAX + 1];
```

ů̈ Ṿ ƚ ə ɾ ặ ⊣ ɛ̆ (ǘ ƚ ۂ ◘, ˚ ǘ ƚ ۻ „ : , ﻤﻌ interned ů̈ ṡ ɾ , ặ ⊣ ɛ̆ ů̈ Ṿ Ϭ ڽ ƶ)

```
PyObject *t = (PyObject *)op;
// 走 intern, ω̈ 面说
PyString_InternInPlace(&t);
op = (PyStringObject *)t;

// ặ ⊣ ɛ̆ ů̈ 字符缓冲 Ṿ 池 Ϭ ڽ ƶ 置
characters[*str & UCHAR_MAX] = op;
```

String - ä Ṝ

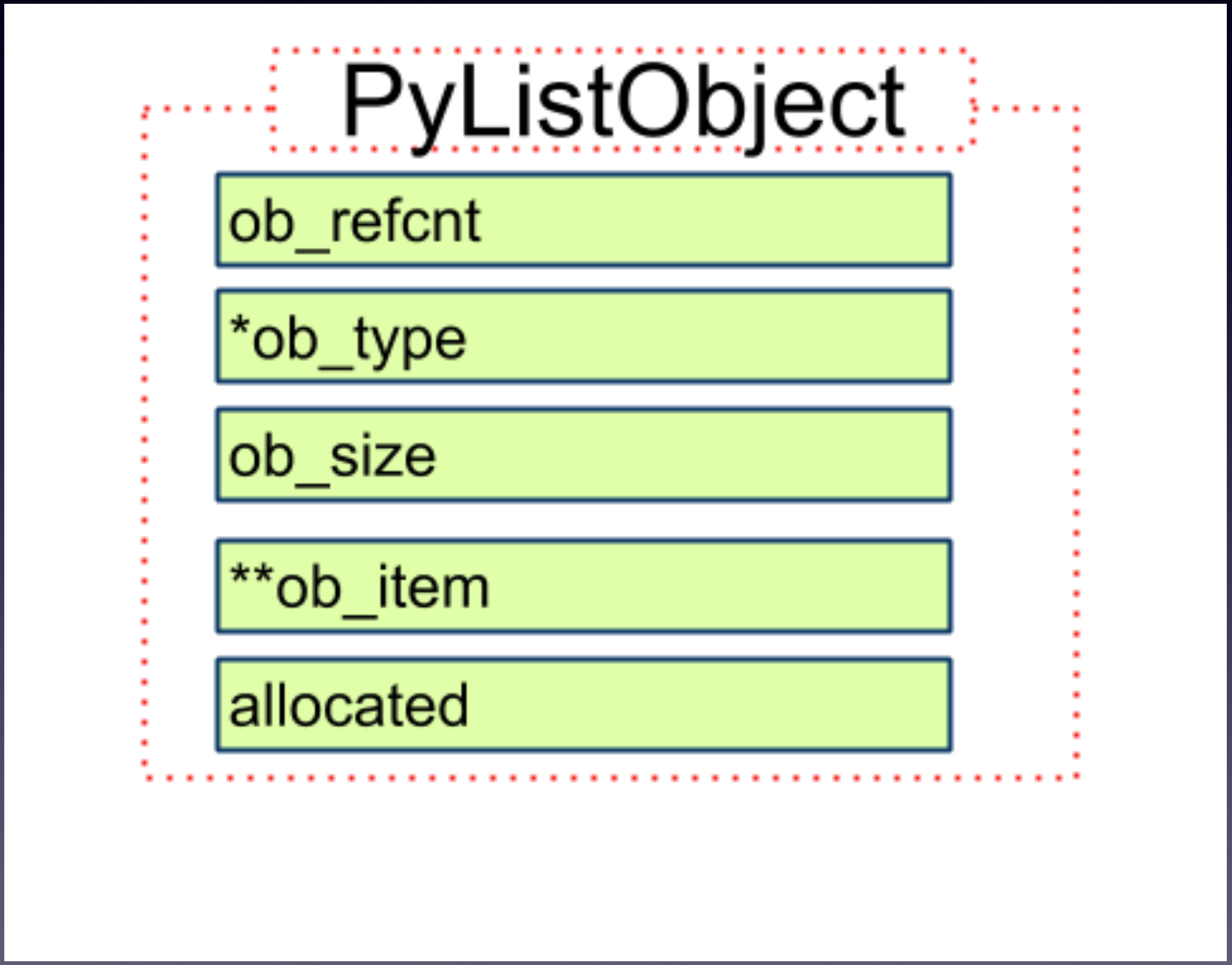
```
'a' + 'b' + 'c'

or

''.join(['a', 'b', 'c'])
```

string_concat, „ Ứ = ẩ „ ṩ ữ , يى ˉ . N ئه , N-1 ṩ ữ ẩ .
string_join, ن Ẳ طح ṅ ṻ א ﮋ · ,
PyString_FromStringAndSize((char*)NULL, sz) ẩ ṩ ữ , ὢ „ يى . „ ṩ
ữ سجد ʄ .

List

List - PyListObject

```
typedef struct {
    PyObject_VAR_HEAD

    PyObject **ob_item;

    Py_ssize_t allocated;
} PyListObject;
```



1. PyObject_VAR_HEAD
PyListObject是变长对象

2. PyObject **ob_item;
指向列表元素的指针数组, list[0] 即 ob_item[0]

3. Py_ssize_t allocated;
allocated列表分配的空间, ob_size为已使用的空间
allocated 总的申请到的内存数量
ob_size 实际使用内存数量

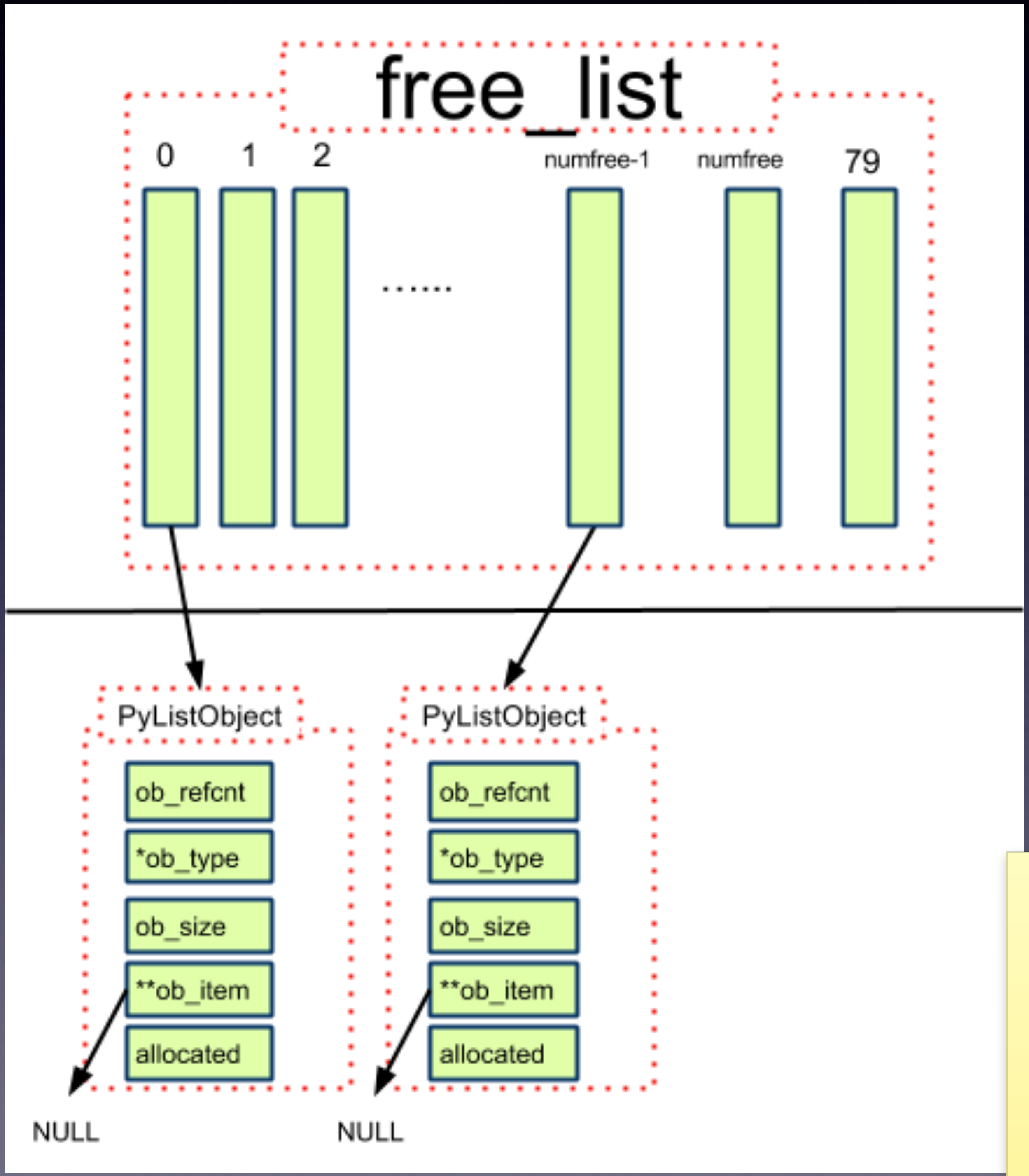
等式:

$0 <= ob\_size <= allocated$

List - Ծ Ṿ

```
/* Empty list reuse scheme to save calls to malloc and free */
#ifndef PyList_MAXFREELIST
#define PyList_MAXFREELIST 80
#endif

// 80 :
static PyListObject *free_list[PyList_MAXFREELIST];

static int numfree = 0;
```

List - ნ Ṿ

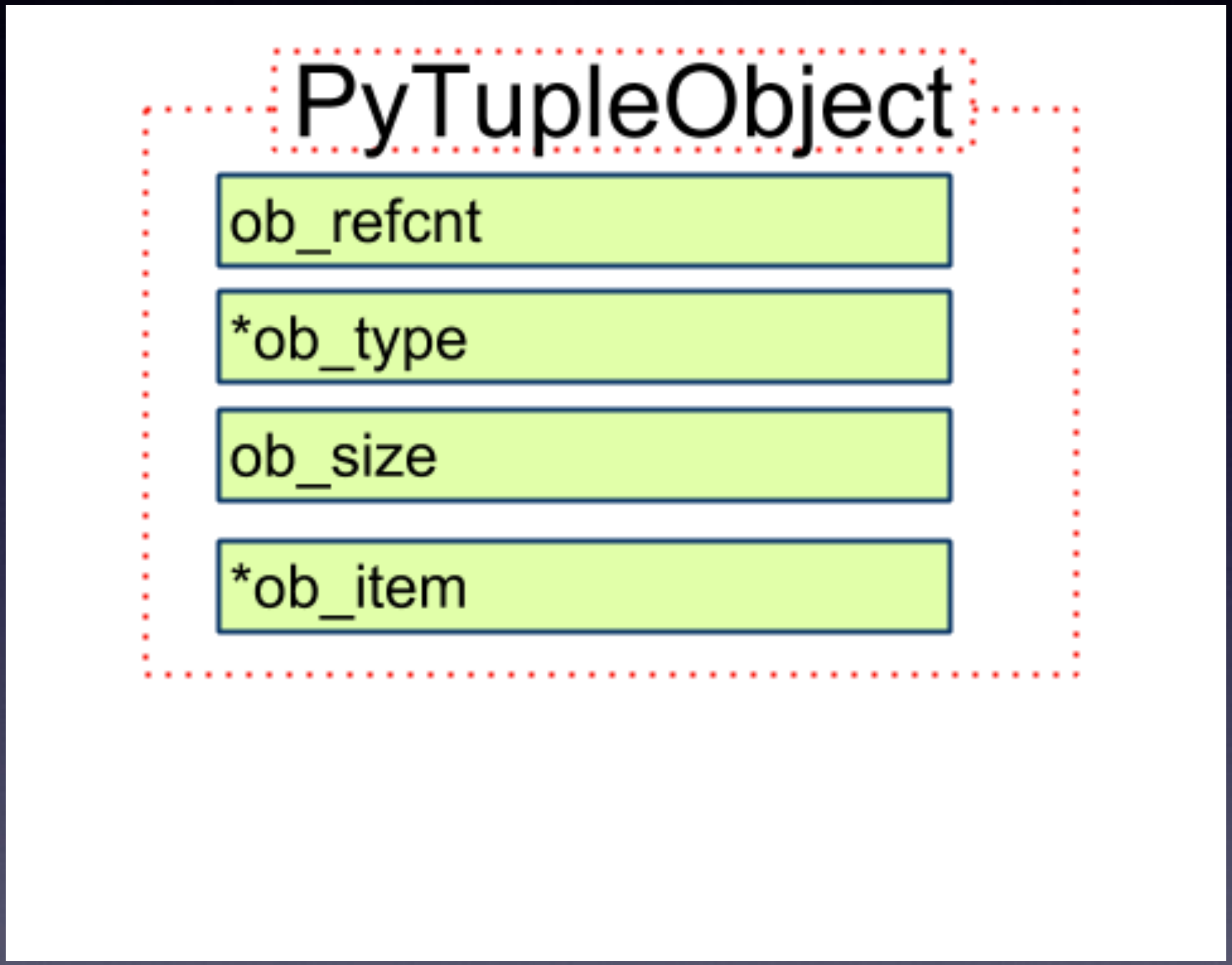


对一个列表对象PyListObject, 回收时, ob_item会被回收, 其每个元素指向的对象引用-1.
但是PyListObject对象本身, 如果缓冲池未满, 会被放入缓冲池, 复用

Tuple

Tuple - PyTupleObject

```
typedef struct {
    PyObject_VAR_HEAD
    PyObject *ob_item[1];

} PyTupleObject;
```



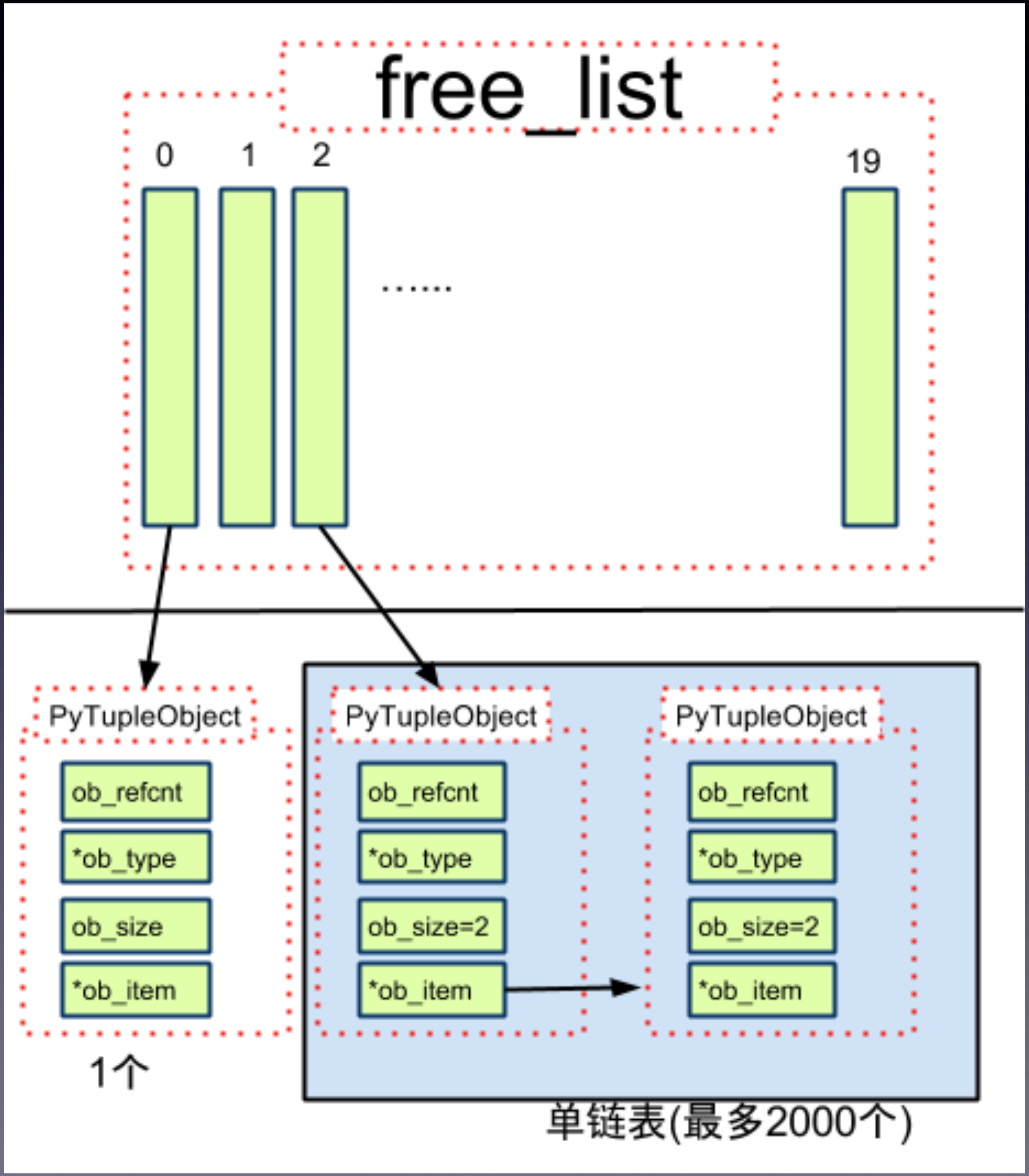
1. PyObject_VAR_HEAD
PyTupleObject在底层是个变长对象(需要存储列表元素个数).
虽然, 在python中, tuple是不可变对象

2. PyObject *ob_item[1];
指向存储元素的数组

Tuple - tuple Ϭ Ṿ

```
/* Speed optimization to avoid frequent malloc/free of small tuples */
#ifndef PyTuple_MAXSAVESIZE
#define PyTuple_MAXSAVESIZE     20
#endif

#ifndef PyTuple_MAXFREELIST
#define PyTuple_MAXFREELIST  2000
#endif

#if PyTuple_MAXSAVESIZE > 0

static PyTupleObject *free_list[PyTuple_MAXSAVESIZE];
static int numfree[PyTuple_MAXSAVESIZE];
#endif
```

Tuple - tupleԾ Ṿ



1. 作用: 优化小tuple的mall/free

2. PyTuple_MAXSAVESIZE = 20
会被缓存的tuple长度阈值, 20, 长度<20的, 才会走对象缓冲池逻辑

3. PyTuple_MAXFREELIST 2000
每种size的tuple最多会被缓存2000个

4. PyTupleObject *free_list[PyTuple_MAXSAVESIZE]
free_list, 指针数组, 每个位置, 存储了指向一个单链表头的地址

5. numfree[PyTuple_MAXSAVESIZE]
numfree, 一个计数数组, 存储free_list对应位置的单链表长度

6. free_list[0], 指向空数组, 有且仅有一个

Dict

Dict - ǚ Ḭ

1. ə̢ لجا Ẳ ǚ Ḭ

2. ə̢ يٍ عجم ٓ Ѡ ˜ Ṿ

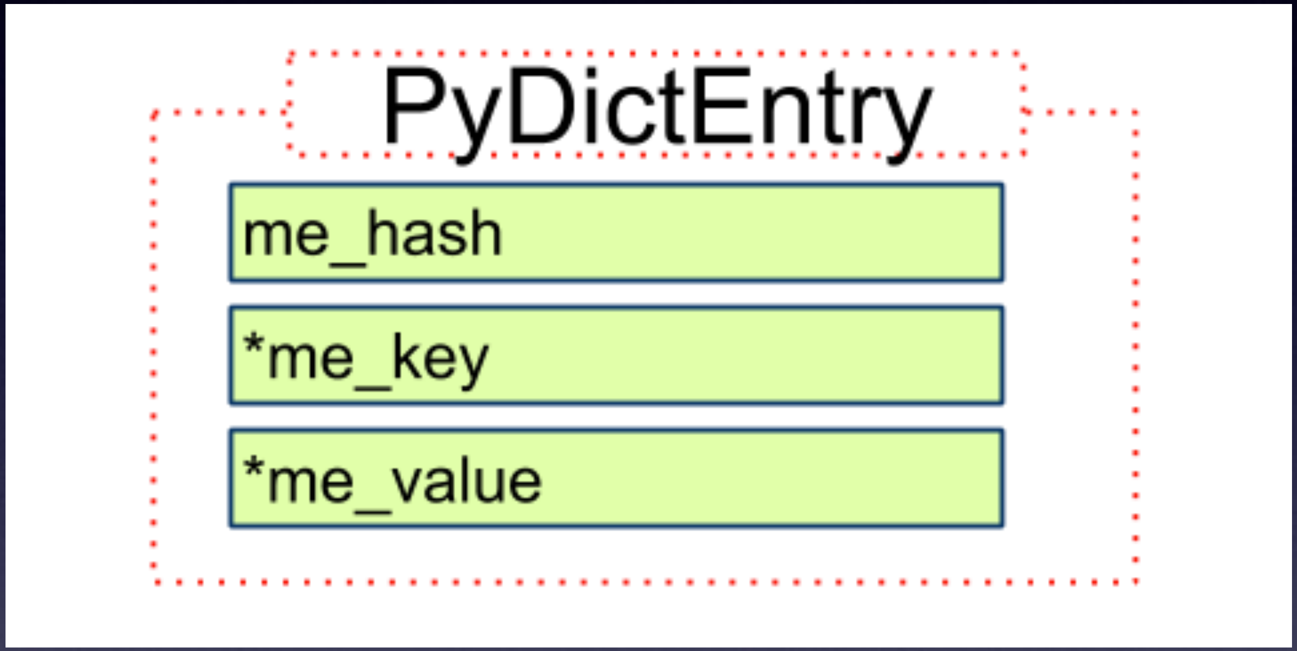
   2.1 غي Ṕ , ì Ṿ , ĸ يه , ᐸ مج ¸ ˛˛ : ƻ , ế مج ế ̏Ậ ƻ , عجم Ṕ (ي تی ˛˛ Ṿ يه)

   2.2 مج , ǚ Ṿ يه

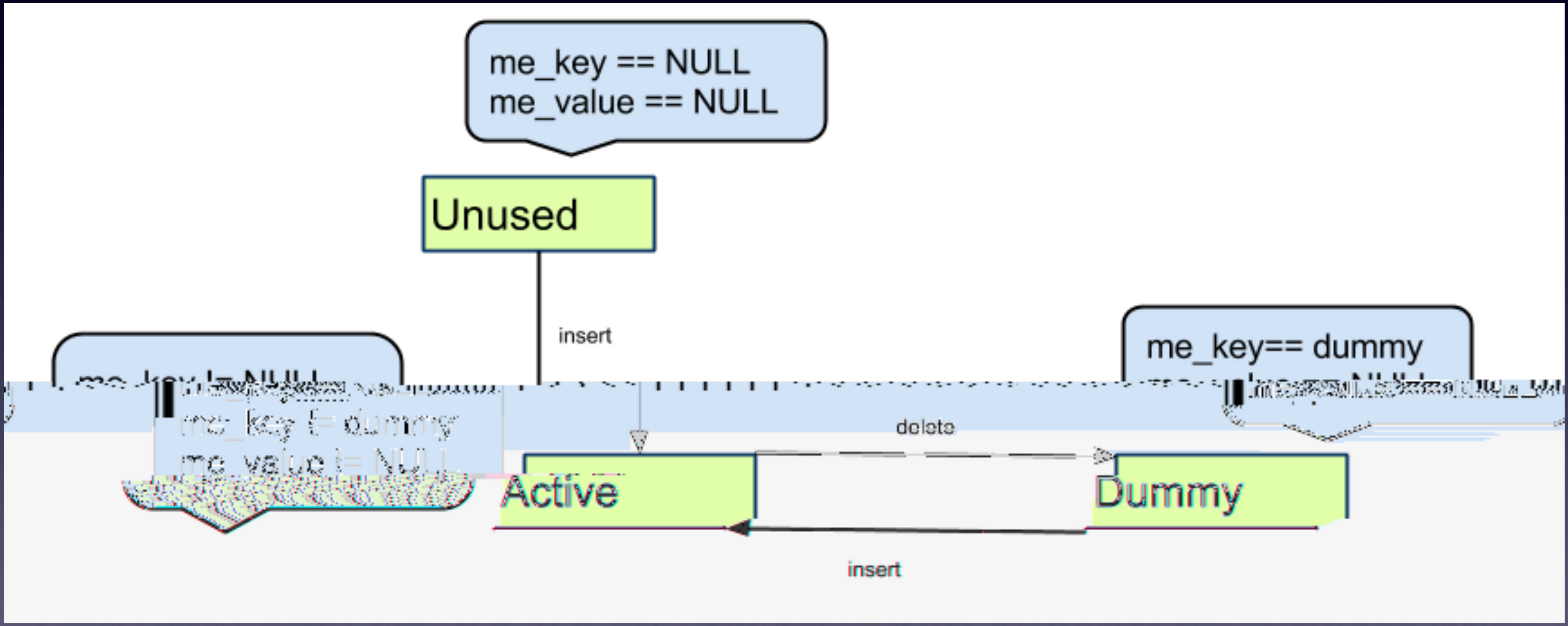
   2.3 Ẹ , ʮ Ʊ Ƚ يه ˩ , ˚ ئه Ẹ , ợ Ẵ δ ˈ محم : (ط æ˚ Ẹ)

Dict - PyDictEntry

```
typedef struct {
    Py_ssize_t me_hash;
    PyObject *me_key;
    PyObject *me_value;
} PyDictEntry;
```



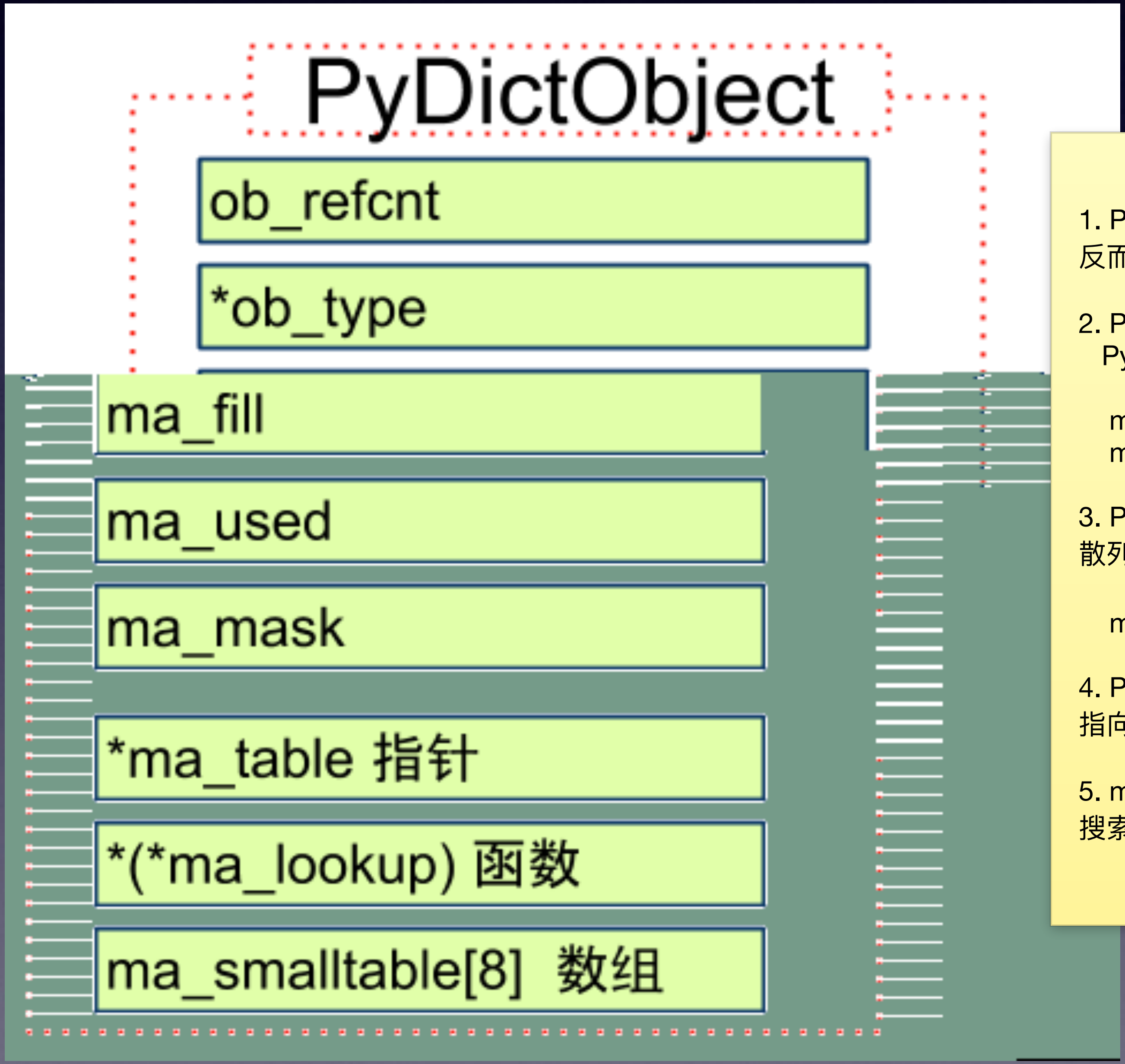
Dict - PyDictEntry



Dict - PyDictObject

```
typedef struct _dictobject PyDictObject;
struct _dictobject {
    PyObject_HEAD

    Py_ssize_t ma_fill;
    Py_ssize_t ma_used;
    Py_ssize_t ma_mask;

    PyDictEntry *ma_table;
    PyDictEntry *(*ma_lookup)(PyDictObject *mp, PyObject *key, long hash);
    PyDictEntry ma_smalltable[PyDict_MINSIZE];
};
```

Dict - PyDictObject



1. PyObject_HEAD
反而声明为定长对象, 因为ob_size在这里用不上, 使用ma_fill和ma_used计数

2. Py_ssize_t ma_fill;
   Py_ssize_t ma_used;

   ma_fill = # Active + # Dummy
   ma_used = # Active

3. Py_ssize_t ma_mask;
散列表entry容量 = ma_mask + 1, 初始值ma_mask = PyDict_MINSIZE - 1 = 7

   ma_mask + 1 = # Unused + # Active + # Dummy

4. PyDictEntry *ma_table;
指向散列表内存, 如果是小的dict(entry数量<=8). 指向ma_smalltable数组

5. ma_lookup
搜索函数

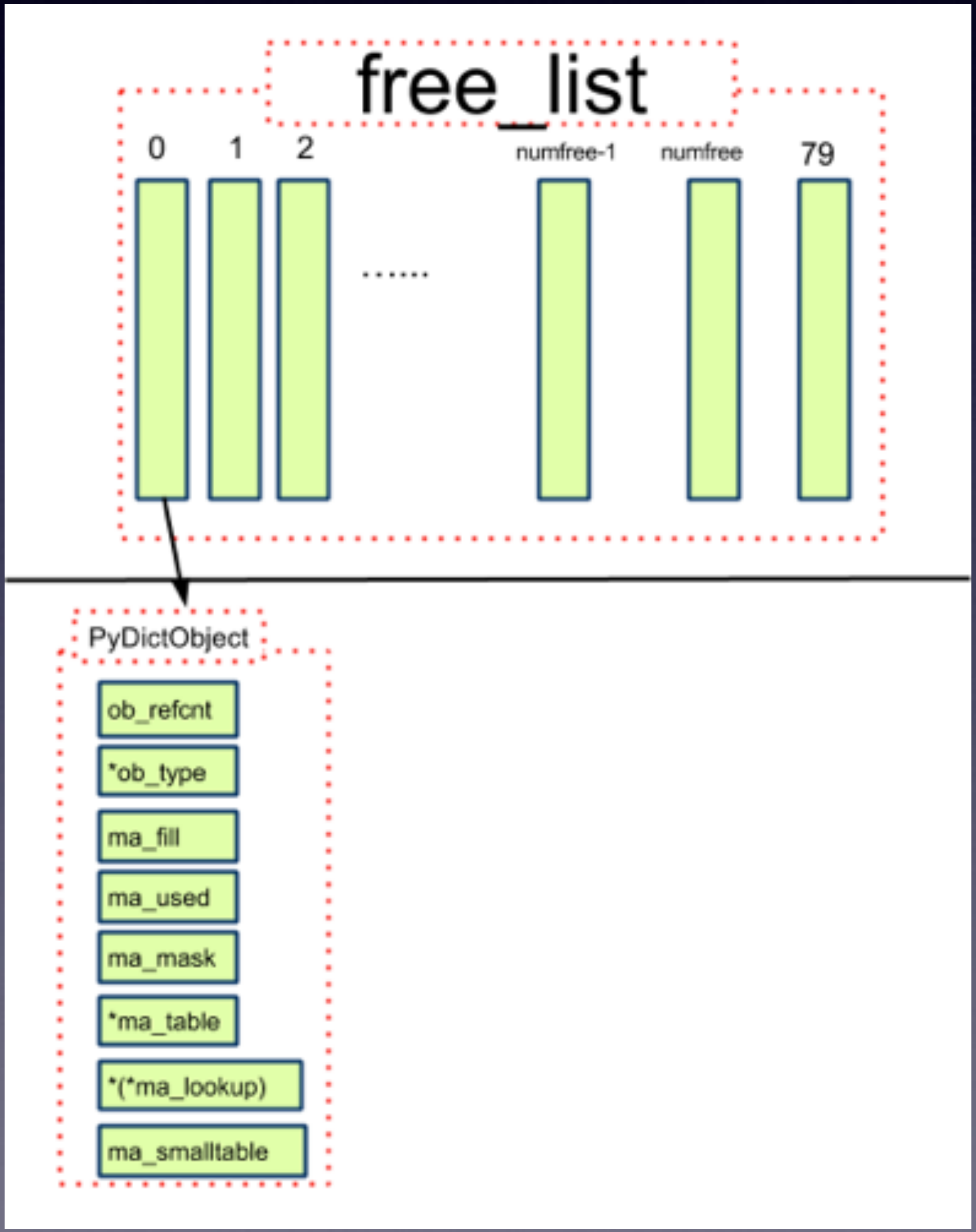
Dict - PyDictObject

1. PyDictObject Ł ǚ ̏ ṩ , يم ma_fill/ma_used/ ma_mask ˂ : ﻠﻤﺤ d̬

2. ặ ˧ ɛ̋ Ặ ﭙ ma_smalltable, ˥ ∩ ῴ̣ , ᵟ ɚ ̄ ẩ

Dict - Ϭ Ṿ

Ϭ Ṿ (PyListObjectϬ Ṿ , „)



ὯΩ

̃Ω

- Python ồ
- Python2.7.8

Q & A

ῴ̀

- ϦVˆ◘ﻄﻤﺢ ề
- ṡ ṻ ề
- ΆͅͿ ̽
- ﻜﻲ
- ﻜﻲ -ٍٍ ﺗ/ﺒﻤ ề /ả ﻠﻤ ề/ ề
- ̉ ặ ˧ ḝ
- ề
- ﻜﺞ ề
- ˥ ┸
- ˩ Ứ ề

Thanks