Memory Management

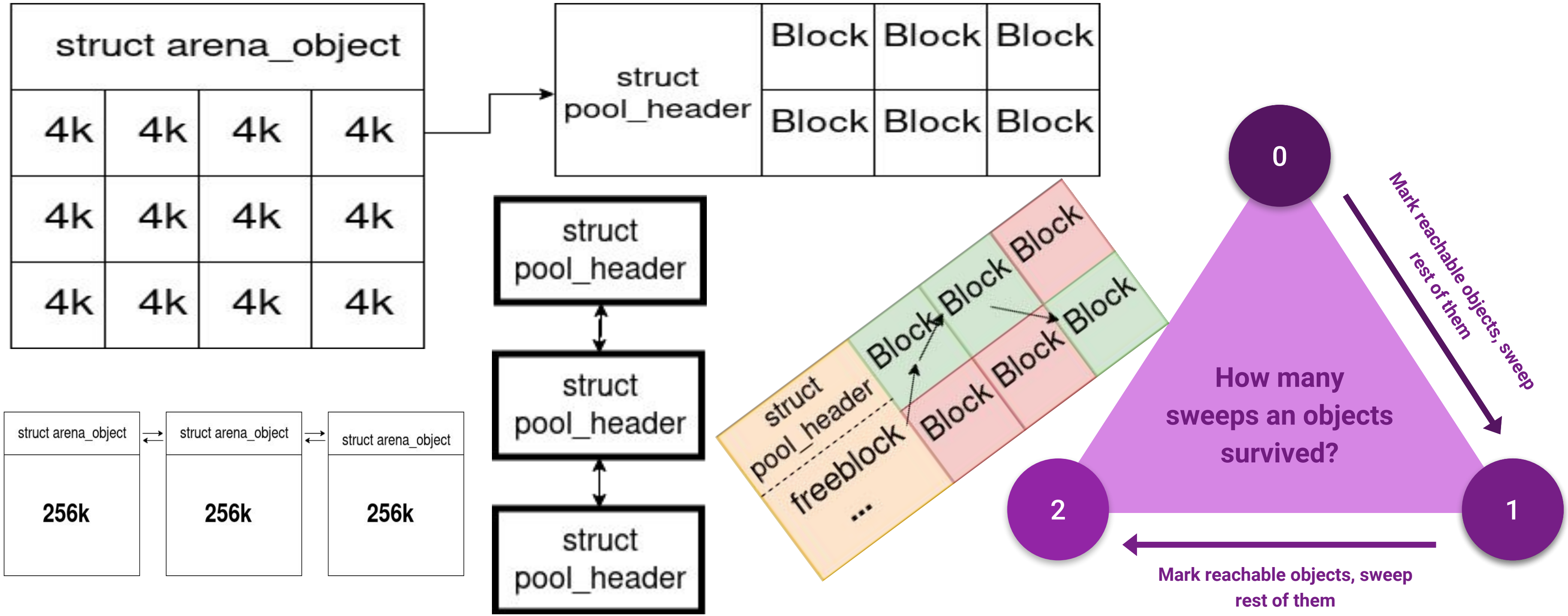
CPython’s Memory Management

Batuhan Taskaya
@exec_emacs on Twitter

Motivation

Why should I learn memory management concepts?

See behind the curtain



Learn how to Control

```python
import gc

my_list = []
my_list.append(my_list)

del my_list

gc.collect()
```

```python
import sys

name = "batuhan"
myname = "batuhan"
ourname = "batuhan"

del name

print(sys.getrefcount("batuhan"))
```

Handle Memory Leaks

```python
import tracemalloc

tracemalloc.start()
run_some_code()
snapshot = tracemalloc.take_snapshot()
top_stats = snapshot.statistics('lineno')
for stat in top_stats[:10]:
    print(stat)
```

Allocation of Memory

- Objects
- Memory Management Model
- Threshold
- Big object allocation
- Small object allocation
- Object Specifics

Everything Is An Object

- In python everything is an object

```c
typedef struct _object {
    _PyObject_HEAD_EXTRA
    Py_ssize_t ob_refcnt;
    struct _typeobject *ob_type;
} PyObject;
```

Memory Management Model of Python

```
    Python's Memory Management Model

    _____   ______   ______       ________
   [ int ] [ dict ] [ list ] ... [ string ]       Python core         |
+3 | <----- Object-specific memory -----> | <-- Non-object memory --> |
    _______________________________       |                           |
   [   Python's object allocator   ]      |                           |
+2 | ####### Object memory ####### | <------ Internal buffers ------> |
    ______________________________________________________________    |
   [          Python's raw memory allocator (PyMem_ API)          ]   |
+1 | <----- Python memory (under PyMem manager's control) ------> |   |
    __________________________________________________________________
   [    Underlying general-purpose allocator (ex: C library malloc)   ]
 0 | <------ Virtual memory allocated for the python process -------> |
   =========================================================================
    _______________________________________________________________________
   [                OS-specific Virtual Memory Manager (VMM)               ]
-1 | <--- Kernel dynamic storage allocation & management (page-based) ---> |
    __________________________________   __________________________________
   [                                  ] [                                  ]
-2 | <-- Physical memory: ROM/RAM --> | | <-- Secondary storage (swap) --> |
*/
/*==========================================================================*/
```

Small Object Threshold

```
obj size > 512 bytes = Big

obj size < 512 bytes = Small
```

Big Objects

- Not our concern
- Uses system allocator

Small Objects

- Managed with 3 level of abstractions
- Blocks encapsulates objects
- Pools contains same sized blocks
- Arena’s contains pool

Blocks

First level of abstractions

- 8-byte-alignment Notation
- Implementation

8-byte-alignment Notation

- The block size can be range(**8**, **512**+1, **8**).
- The size idx value can be found with (allocated space / 8) - 1

Object Size (bytes)	Allocated Space (bytes)	Size Idx
1-8	8	0
9-16	16	1
17-24	24	2
25-32	32	3
33-40	40	4
505-512	512	63

Implementation

- They designed for containing python objects
- Uses 8-byte-alignment notation for better management over free blocks
- Marked as free and linked to free blocks when their object deallocated.

Pools

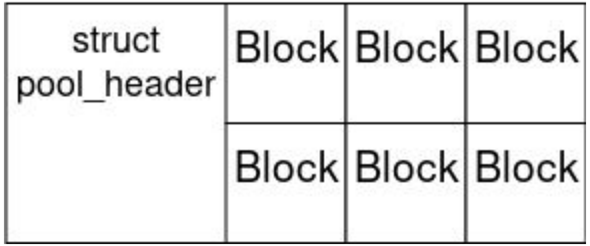
encapsulates same sized blocks.

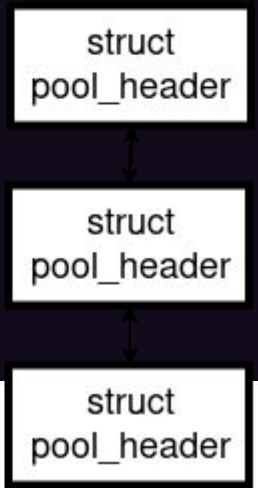
- Implementation
- States

Implementation

- Contains same sized blocks
- 4K Size
- Every pool has a pool_header overhead for meta information.
- Every pool linked together with nextpool & prevpool ptrs.

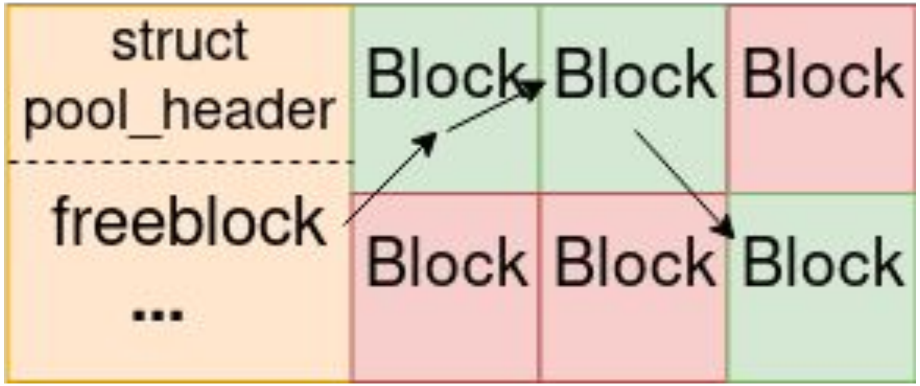
```c
struct pool_header {
    union { block *_padding;
            uint count; } ref;
    block *freeblock;
    struct pool_header *nextpool;
    struct pool_header *prevpool;
    uint arenaindex;
    uint szidx;
    uint nextoffset;
    uint maxnextoffset;
};

```





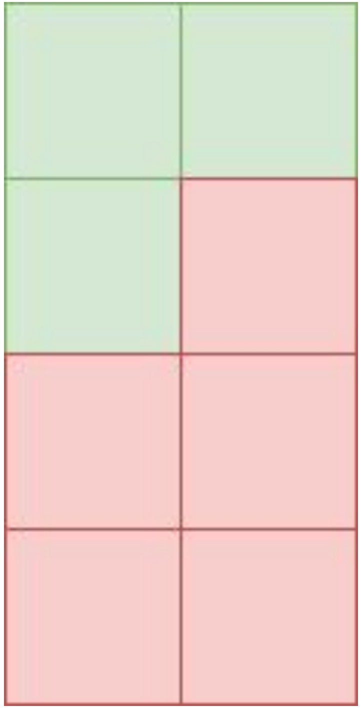
Implementation - Free Block

- Linked List of Blocks
- Blocks inserted whenever they freed.

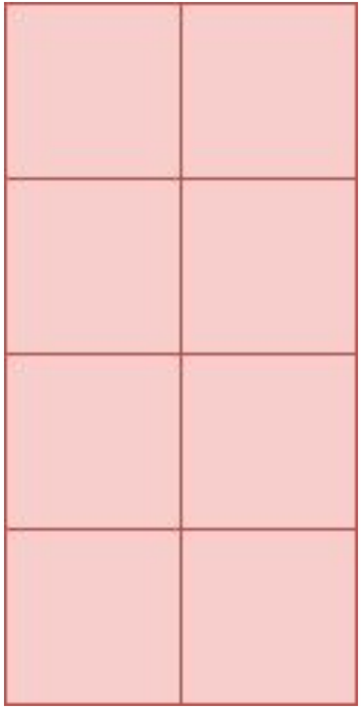


States of Blocks

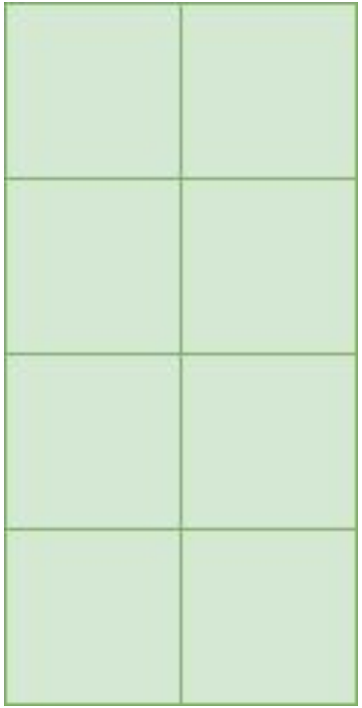
Used



Full



Empty



Arenas

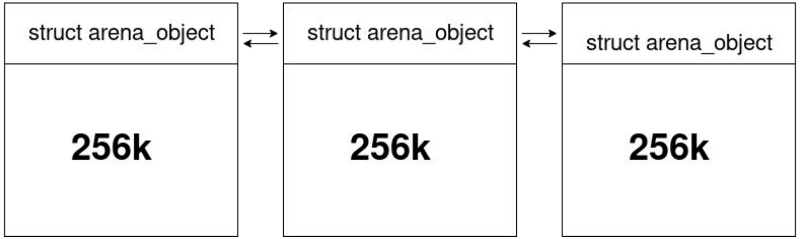
Encapsulates pools

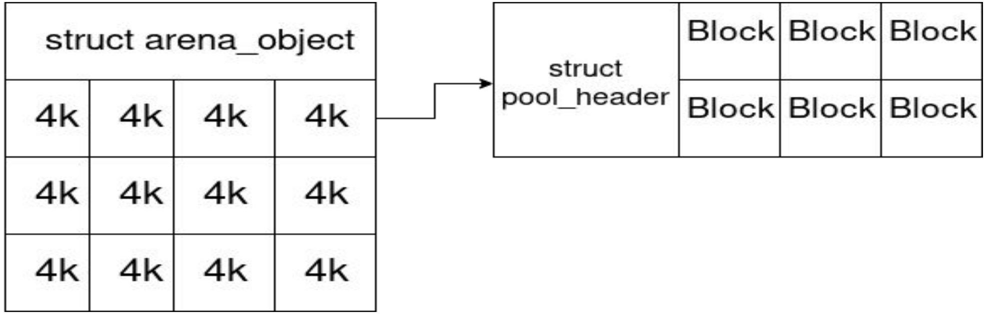
- Implementation

Implementation

- Contains 64 pools.
- Size is 256kb. A big block of memory.
- System allocator only allocates space for arenas. The other abstractions uses this space.
- Also they are linked together like pools.

```
struct arena_object {
    uintptr_t address;
    block* pool_address;
    uint nfreepools;
    uint ntotalpools;
    struct pool_header* freepools
    struct arena_object* nextarena;
    struct arena_object* prevarena;
};
```





Object Specifics

- String Interning
- Small Integers

String Interning

- One object and multiple names assigned to it
- Happens in Compile Time
- By default basic strings

```python
>>> a = "batuhan"
>>> b = "batuhan"
>>> assert a is b
>>>
>>> a = "b@tuhan"
>>> b = "b@tuhan"
>>> assert a is not b
>>>
>>> a = "batuhan"
>>> b = "".join(a)
>>> assert a is not b
```

Small Integers

classification

Title:	small int optimization		
Type:	performance	Stage:	
Components:	Interpreter Core	Versions:	Python 3.4

process

```
>>> a = 200
>>> b = 200
>>> assert a is b
>>> a = 270
>>> b = 270
>>> assert a is not b
```

- between **-5 … 256**
- has internal references

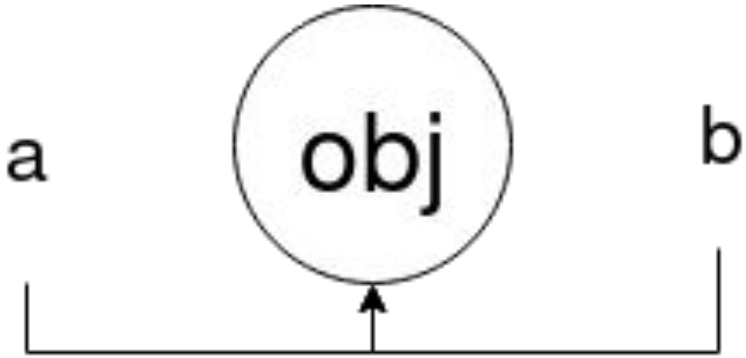
Garbage Collection

Deallocation of Memory

- Reference Counting
- Generational GC

What is Reference Count?

- Reference
- Ref Count




```python
import sys

a = 2018
b = [2018]
c = dict(a=2017, b=2018, c=2019)

sys.getrefcount(a)
```

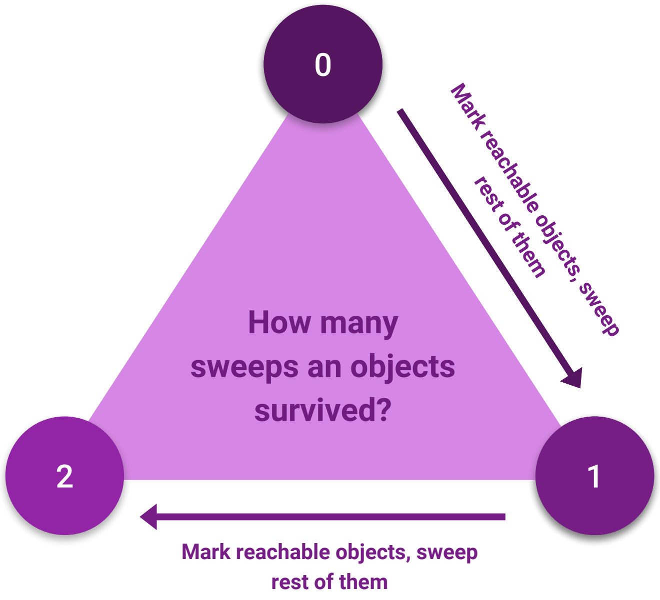
Good Sides vs Bad Sides

- Easy to find unused object
- No need for marking

- Overhead
- No support cyclical references.
- One of the reasons of GIL

What is Generational GC?

- A GC type powered by a tracing algorithm called Mark&Sweep
- Has generations and the generations helps GC to find cyclic references.



How To Track / Manage It

- gc
- tracemalloc

Garbage Collector Interface - GC

```python
import gc

def dummyf(i):
  a = []
  a.append(a)

gc.disable()
if not gc.isenabled():
  for _ in range(10):
    dummyf(_)
  collected = gc.collect()
  print(f"Total {collected} objects freed")

print(f"Track status of 'a': {gc.is_tracked('a')}")
print(f"Track status of []: {gc.is_tracked([])}")
```

```
Total 10 objects freed
Track status of 'a': False
Track status of []: True
```

Trace memory allocations - TraceMalloc

```
import tracemalloc

tracemalloc.start(2)

a = dict(a='b', c='d')
b = list(range(5))

snapshot = tracemalloc.take_snapshot()

stat = snapshot.statistics('traceback')[0]
print(f"{stat.count} memory blocks: {stat.size / 1024} KiB")
for line in stat.traceback.format():
    print(line)
```

Debug Malloc Stats

```python
import sys
sys._debugmallocstats()
```

PYTHONMALLOC

- debug
- malloc_debug
- pymalloc_debug

Questions?