

8. Немного о памяти в Python

Программирование и алгоритмизация

Практические занятия

БИВТ-24-17

Надежда Анисимова

`ms teams m2102039@edu.misis.ru`

Проверка себя

1. В чем разница между `new` и `new[]`?
2. Зачем нужен итератор?
3. Каковы алгоритмы бинарного поиска и метода двух указателей?

Как дела с памятью в Питоне?

01

CPython – дефолтная имплементация питона

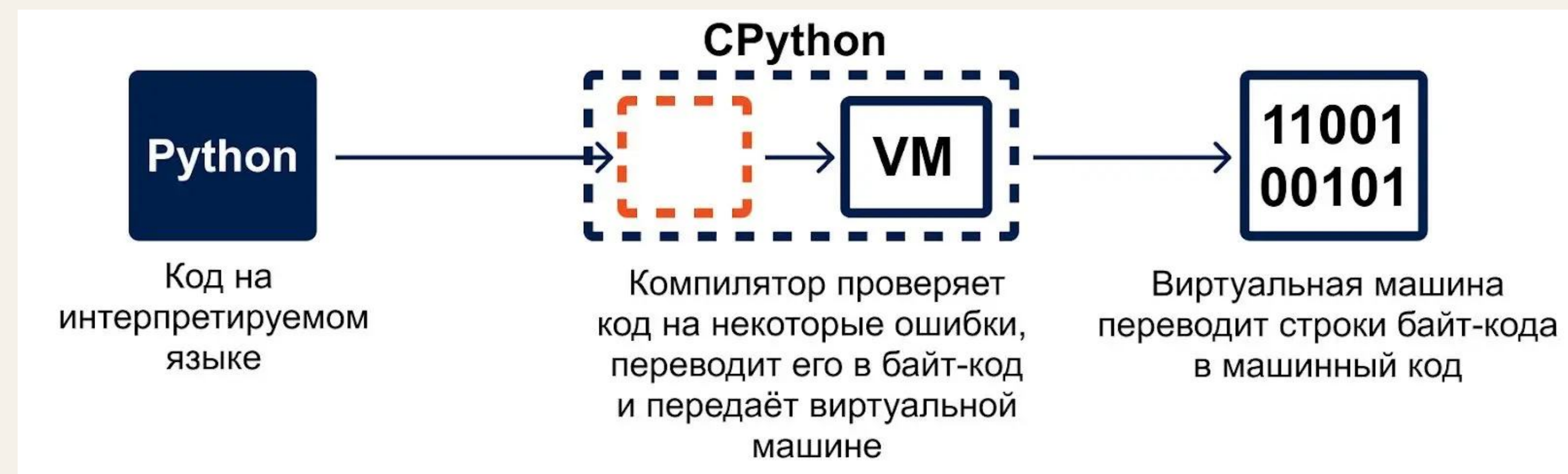
`platform.python_implementation()`

(но есть и другие)

C/C++

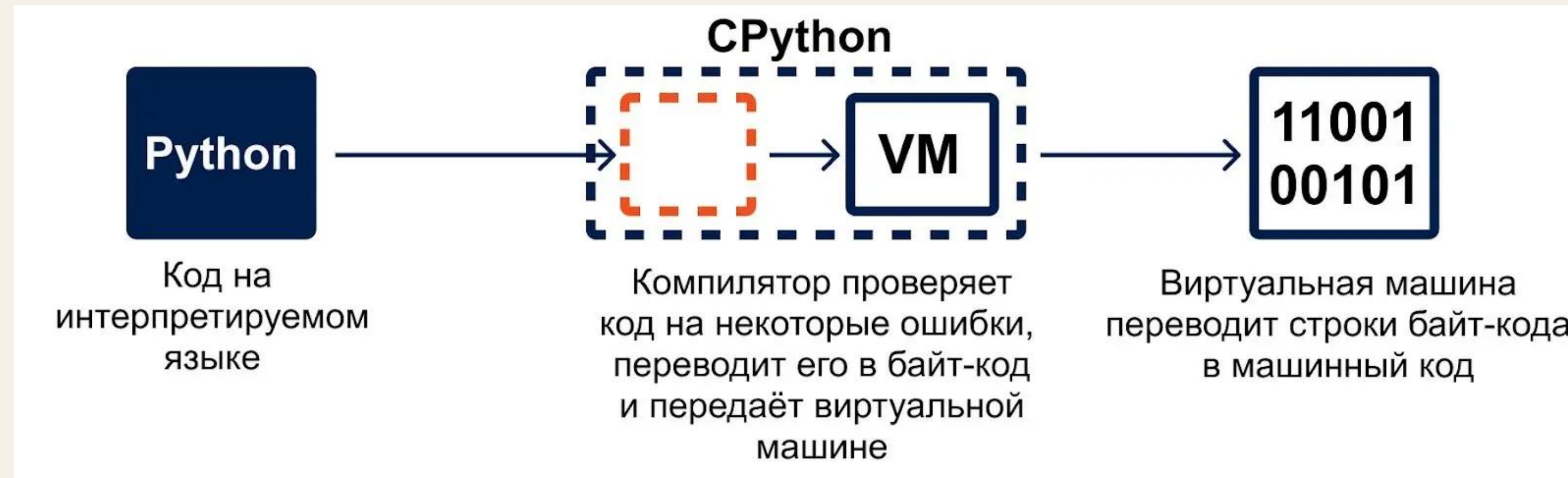


Python



C++ — КОМПИЛИРУЕМЫЙ СТАТИЧЕСКИ ТИПИЗИРОВАННЫЙ ЯП

PYTHON — ИНТЕРПРЕТИРУЕМЫЙ И ДИНАМИЧЕСКИ ТИПИЗИРОВАННЫЙ ЯП



- **Интерпретатор – программа которая исполняет код строка за строкой:**

- сперва исходный код, который написан программистом и хранится в файлах с расширением .py, преобразуется в байт-код (файлы с расширением .pyc, __pycache__ папка)

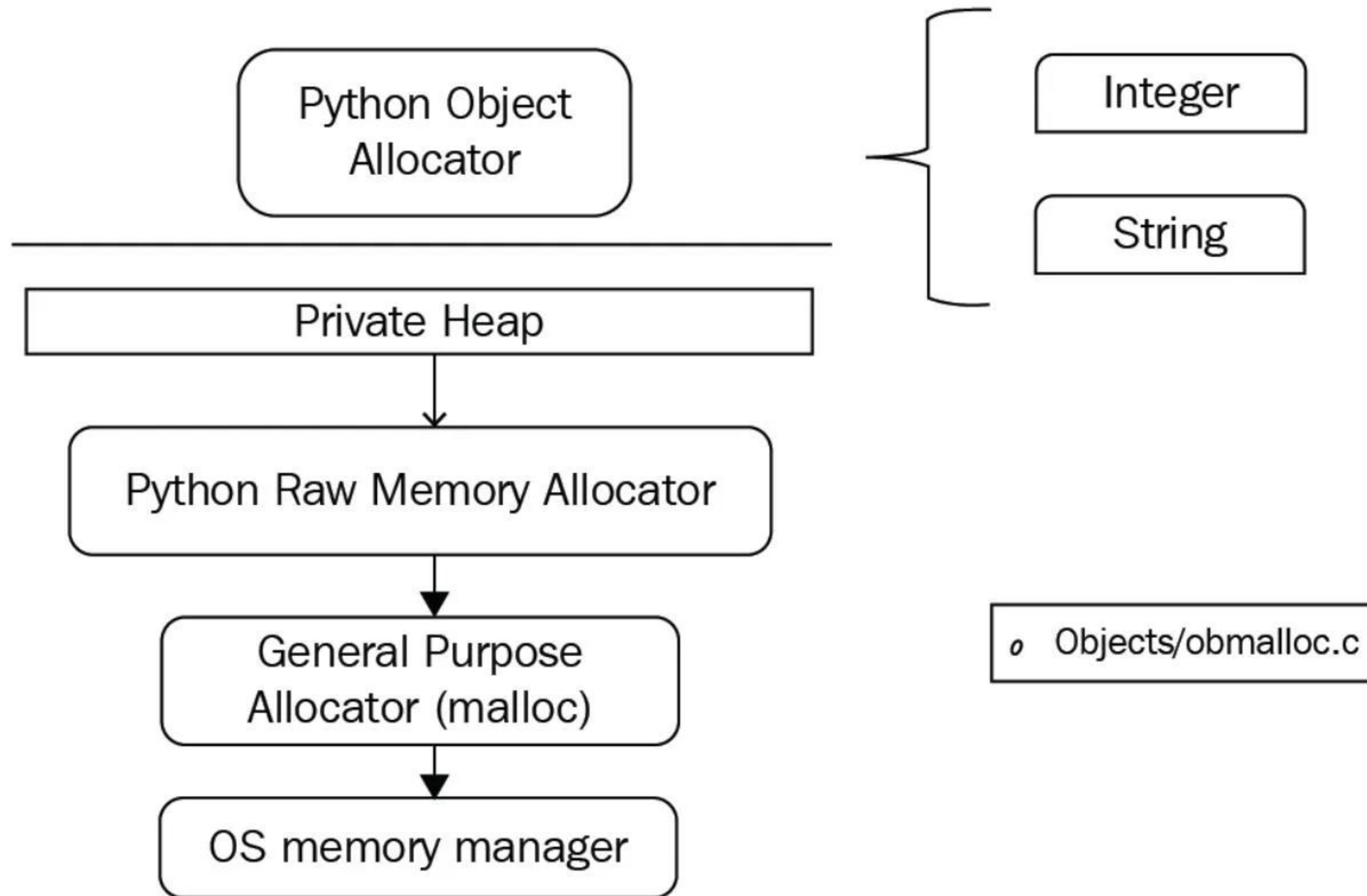
Байт-код – это промежуточный слой между человеко-читаемым кодом и кодом на языке процессора.

- затем байт-код исполняется виртуальной машиной (PVM)

Управление памятью в питоне

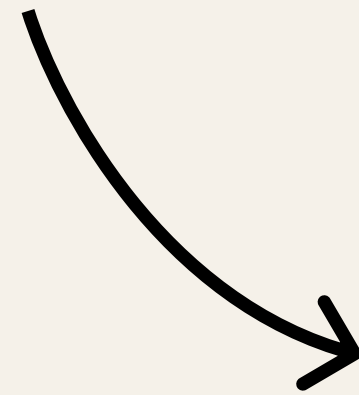
- **Python memory manager** ответственен за управлением памятью, а не программист. У программиста нет контроля над памятью
- Все объекты и структуры данных питона хранятся **в куче (private heap)**
- По требованию python memory manager распределение памяти под внутренние объекты питона выполняется с помощью функций АПИ Python/C

Python Memory Allocator



Всё в питоне – это объект, а точнее PyObject

CPython написан на C, который изначально не поддерживает объектно-ориентированное программирование



Python Object

num = 10



type	int
value	10
reference	1

PyObject – это структура struct, можно сказать класс без функций

<https://github.com/python/cpython/blob/main/Objects/object.c>

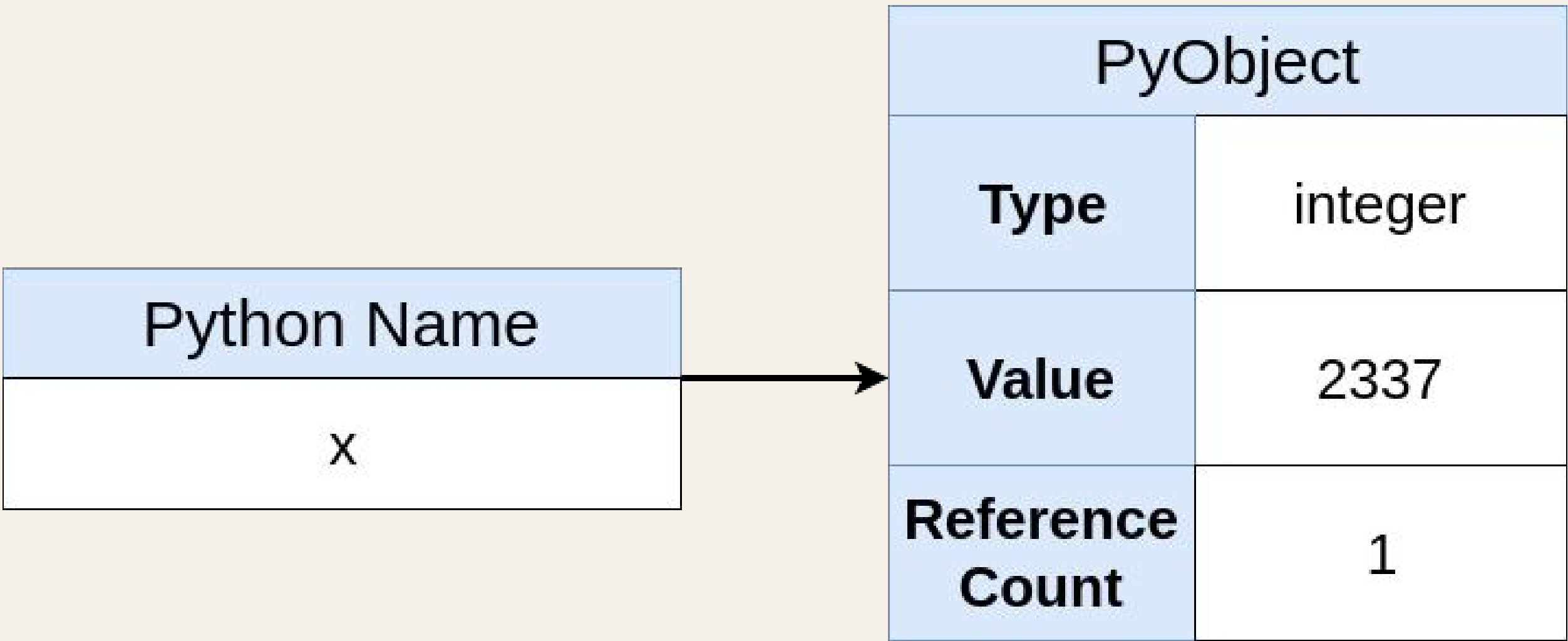
В Python **НЕТ ПЕРЕМЕННЫХ** в стандартном понимании
этого термина

В Python переменные – это ссылки на объекты в памяти

C++

X	
Location	0x7f1
Value	2338

python



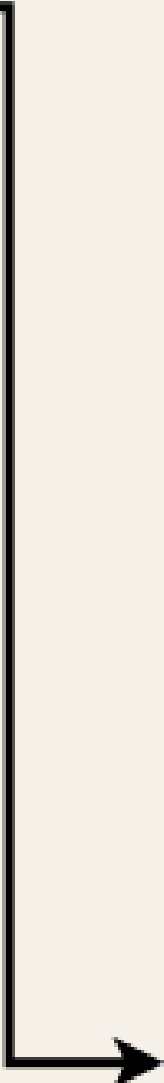
Каждый раз
создается новый
объект

```
x = 0
x += 1
```

Python Name
x

PyObject	
Type	integer
Value	2337
Reference Count	0

PyObject	
Type	integer
Value	2338
Reference Count	1



Значение не копируется - теперь
оба имени ссылаются на единый
блок памяти

```
x = y
```

Python Name
x

Python Name
y

PyObject	
Type	integer
Value	2337
Reference Count	0

PyObject	
Type	integer
Value	2338
Reference Count	2



1. `id()` возвращает адрес памяти объекта;
2. `is` возвращает `True`, если и только если два объекта имеют одинаковый адрес памяти.

`a = 10`
`id(a)` # адрес памяти

Каждый раз
– новый
объект

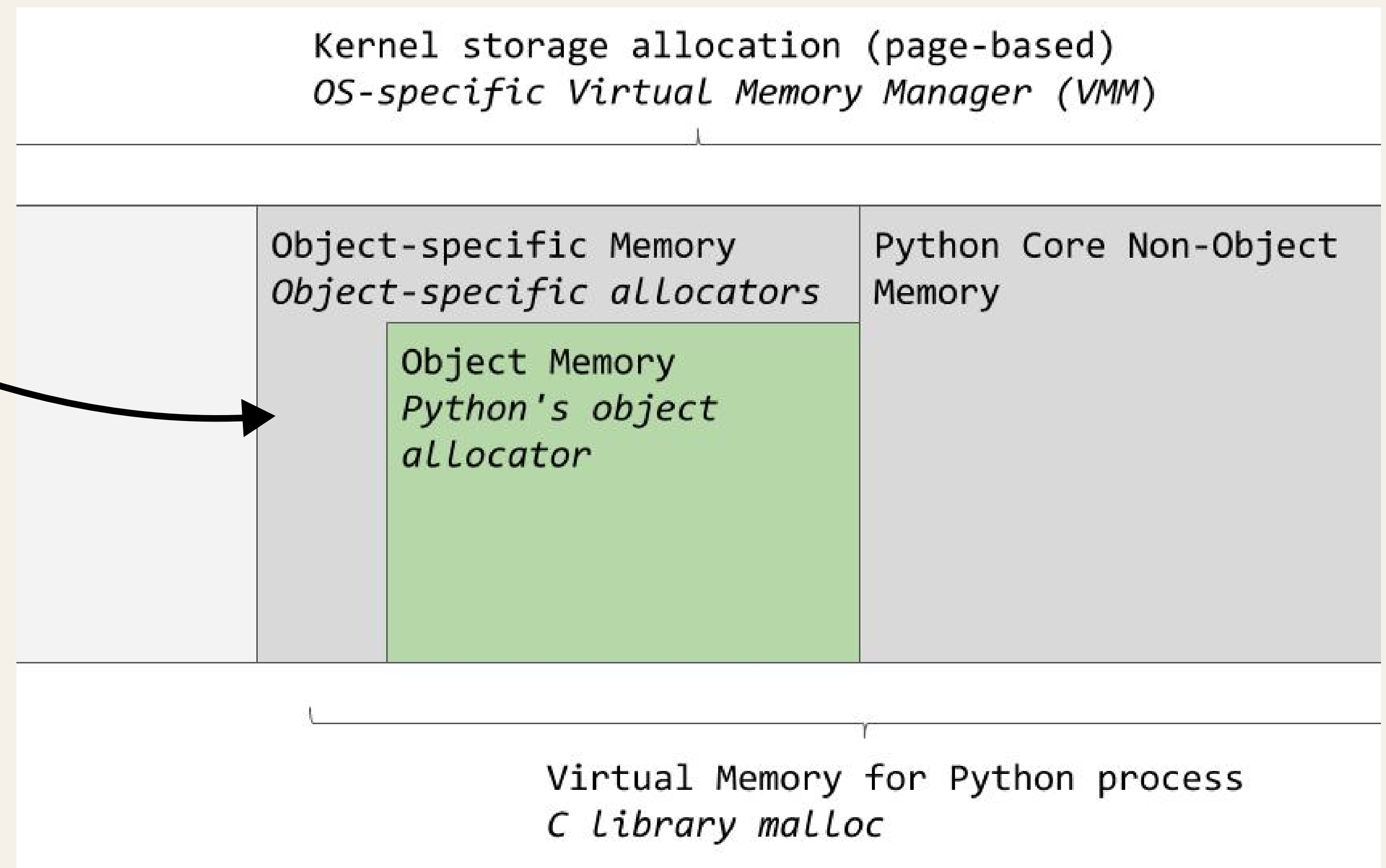
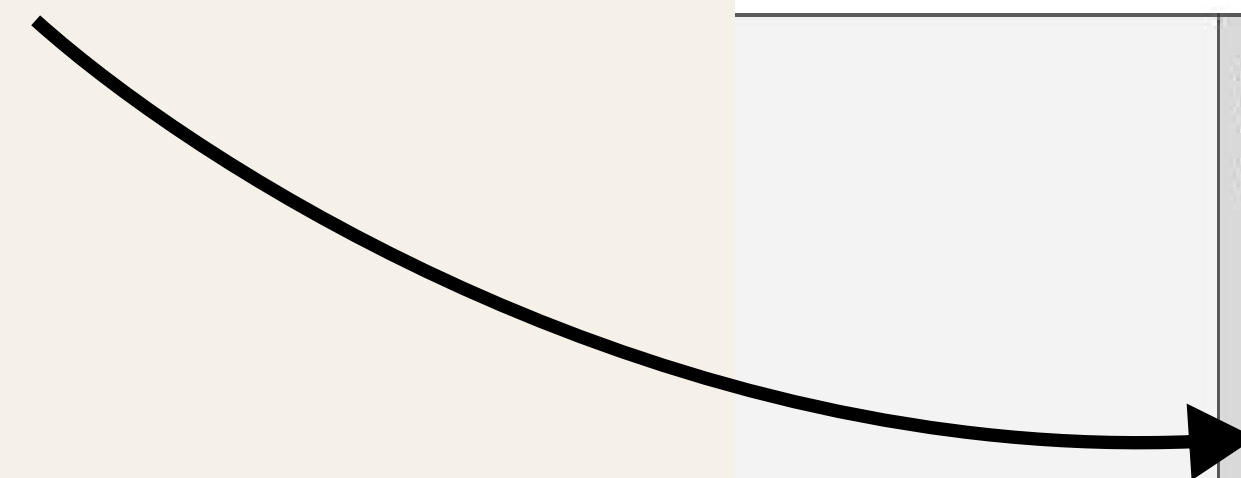


Можно
добавить
элемент

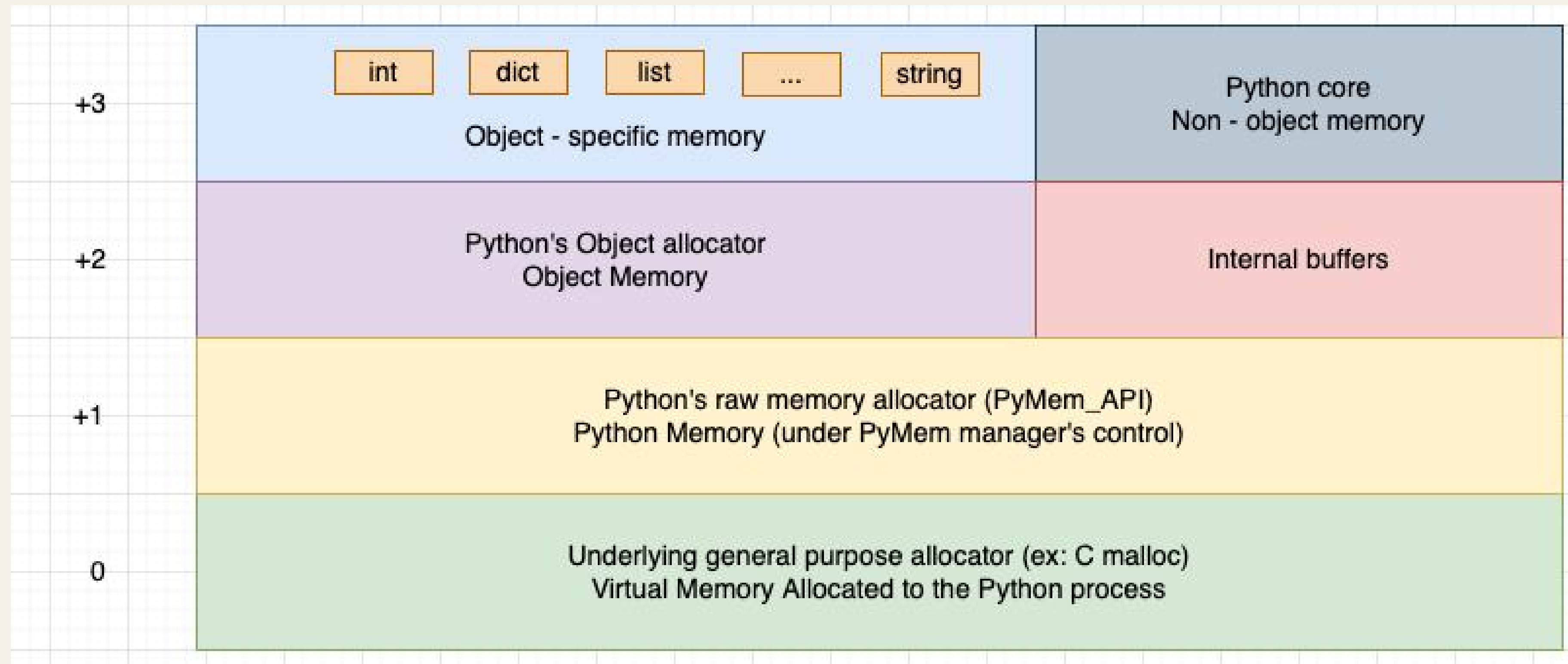
Пул, арена

02

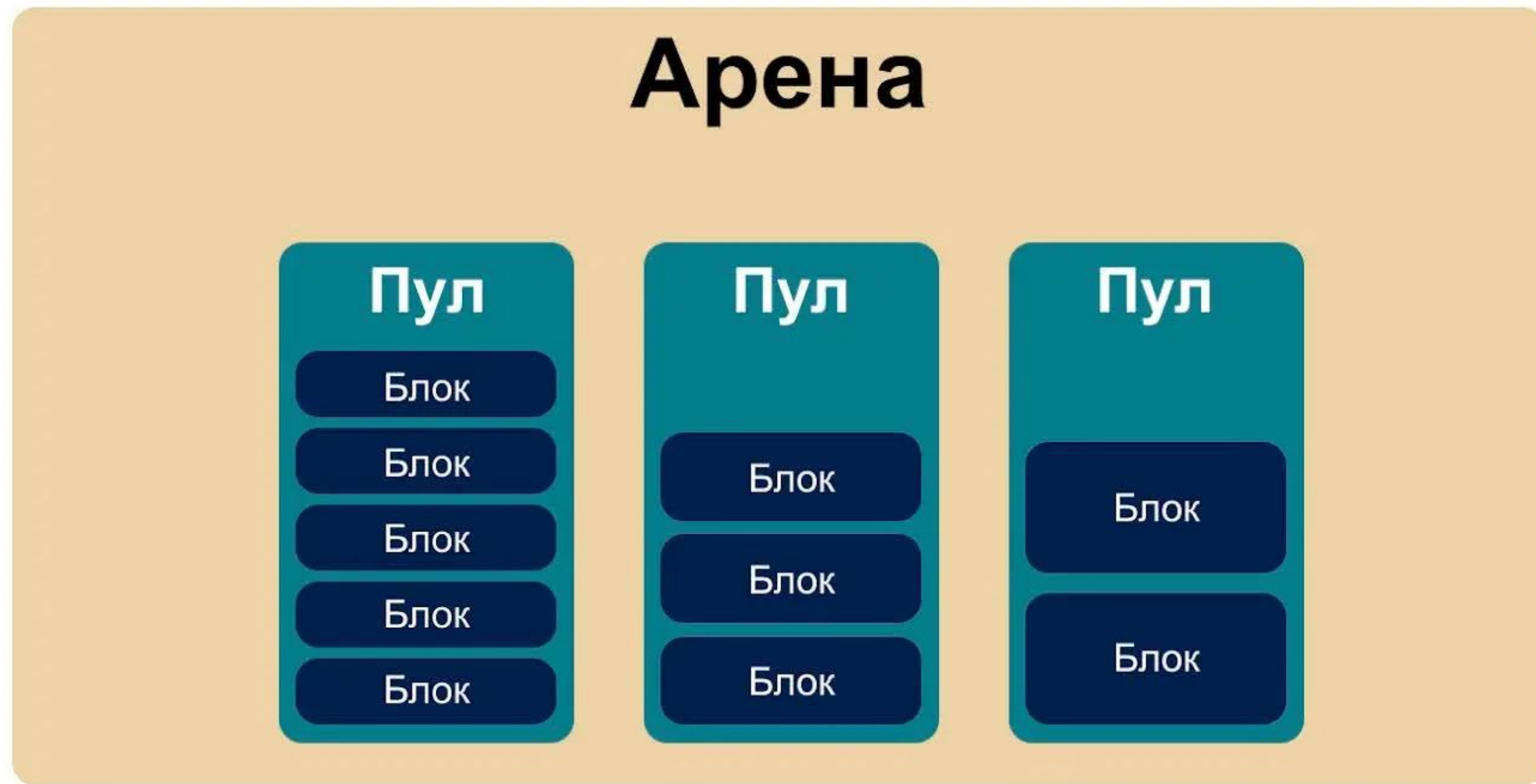
Менеджер виртуальной памяти, в зависимости от операционной системы, выделяет часть памяти для процесса Python



- Внутренний аллокатор `pymalloc`. Он работает только с теми данными, объём которых не превышает 512 байт памяти
- Если для работы объекта нужно больше 512 байт, запрос отправляется в системный аллокатор `malloc`. Это стандартный механизм работы с памятью в операционной системе.



руталлос работает с объектами: ареной, пулом и блоком



Блоки — основные единицы работы румаллос. Это ячейки памяти, кратные 8 байтам. На каждый запрос до 512 байт выделяется один из блоков.

Блок может быть выделен только одному объекту, а объект может быть выделен только одному блоку.

Если требуется 14 -> выделяется блок на 16

Бывают:

- untouched — блок еще не использовался для хранения данных
- free — блок использовался механизмом памяти, но больше не содержит
- allocated — блок хранит данные, необходимые для выполнения программы.

Request in bytes	Size of allocated block	Size class idx
1 - 8	8	0
9 - 16	16	1
17 - 24	24	2
25 - 32	32	3
33 - 40	40	4
41 - 48	48	5
49 - 56	56	6
.....
497 - 504	504	62
505 - 512	512	63

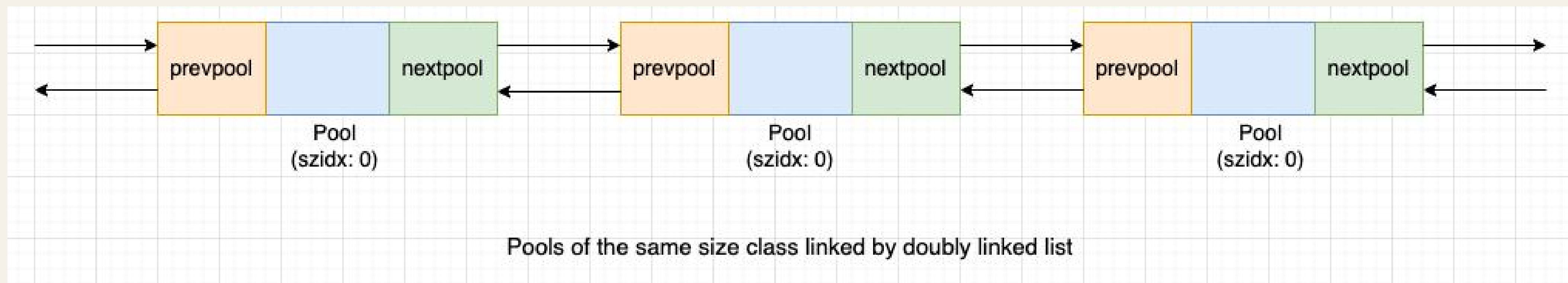
Пул – состоит из блоков одного размера

Пулы нужны для быстрого поиска, выделения и освобождения памяти

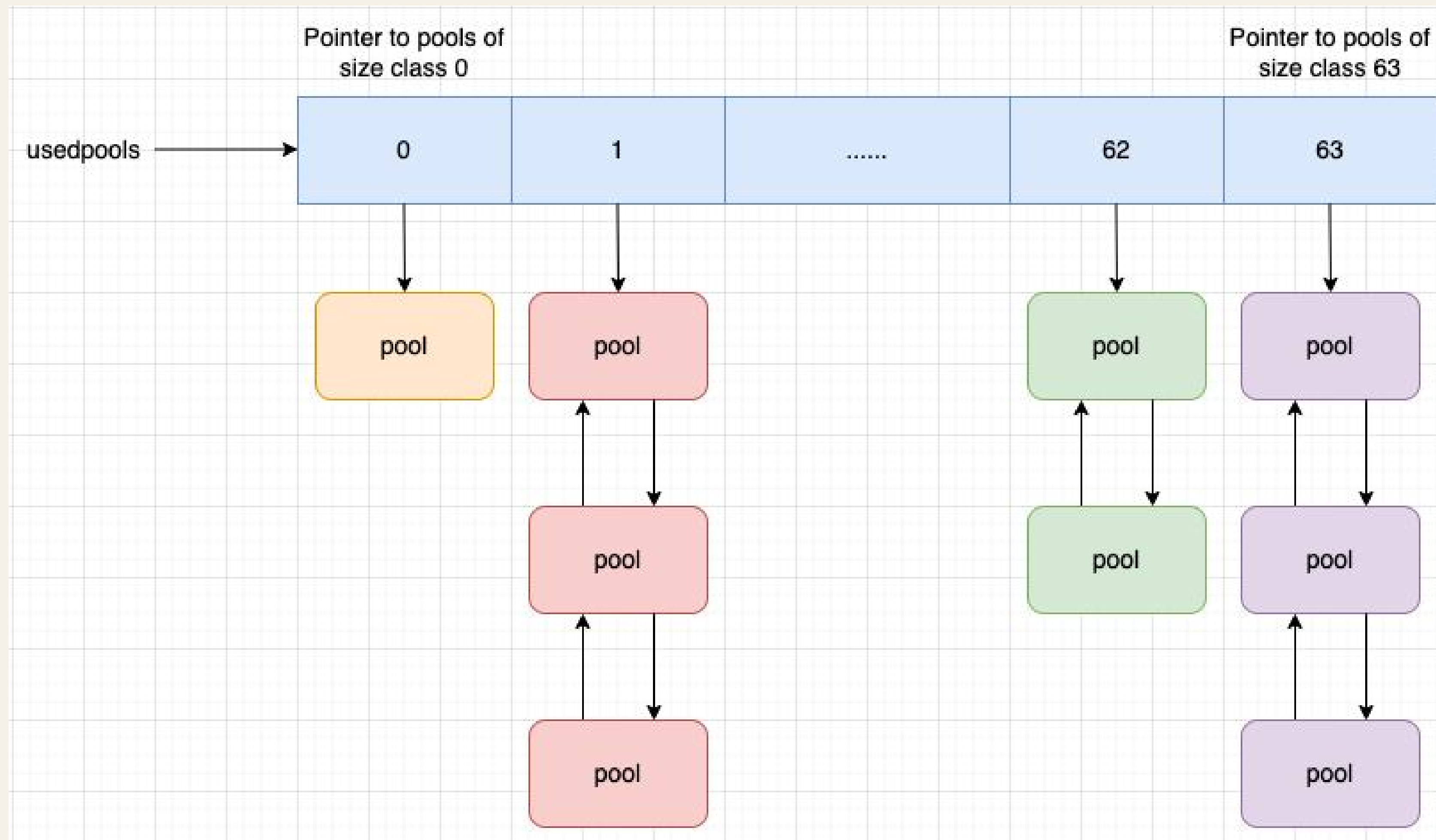
Бывают:

- Used – есть свободные блоки
- Full
- Empty

```
/* Pool for small blocks. */
struct pool_header {
    union { block *_padding;
           uint count; } ref;           /* number of allocated blocks */
    block *freeblock;                   /* pool's free list head */
    struct pool_header *nextpool;       /* next pool of this size class */
    struct pool_header *prevpool;       /* previous pool */
    uint arenaindex;                   /* index into arenas of base adr */
    uint szidx;                        /* block size class index */
    uint nextoffset;                   /* bytes to virgin block */
    uint maxnextoffset;                /* largest valid nextoffset */
};
```

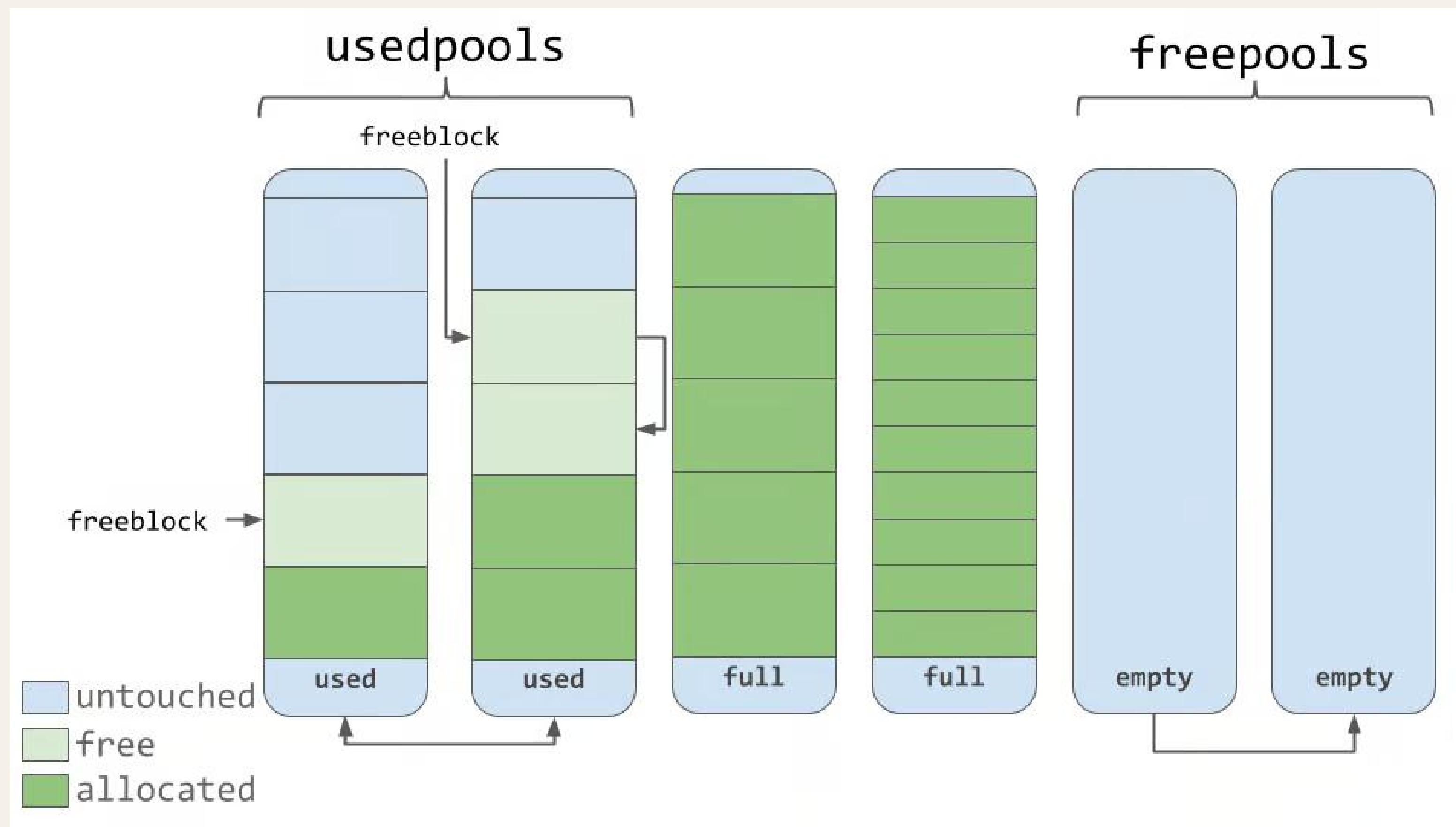


Массив `usedpools` используется для отслеживания заполненных пулов каждого размера



Когда выполняется запрос на выделение памяти, а доступных пулов с блоком требуемого размера нет, CPython создает новый пул из арены.

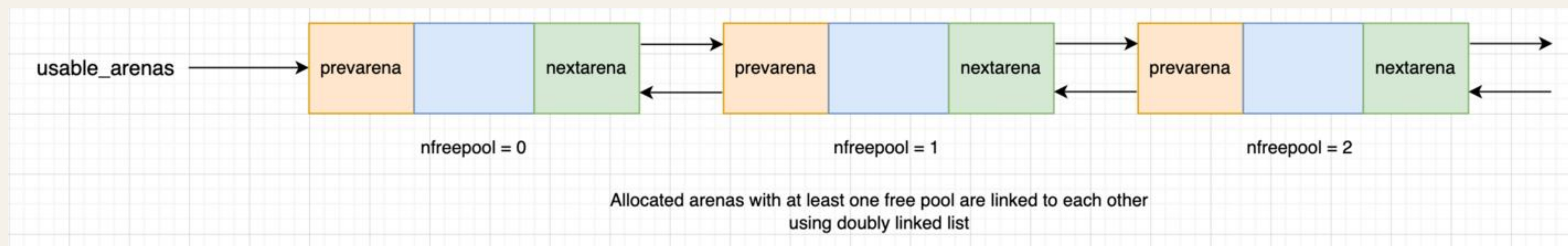
Когда создается новый пул, весь пул не разбивается сразу на блоки. Блоки вырезаются из бассейна по мере необходимости



Арена — фрагмент памяти,
кратный 4 килобайтам
и состоящий из более мелких
фрагментов — пулов

Когда выделенный блок в пуле
освобождается, CPython не возвращает
память обратно операционной системе.
Эта память продолжает принадлежать
процессу Python, и CPython использует
этот блок для выделения памяти новым
объектам.

```
struct arena_object {  
    /* The address of the arena, as returned by malloc */  
    uintptr_t address;  
  
    /* Pool-aligned pointer to the next pool to be carved off. */  
    block* pool_address;  
  
    /* The number of available pools in the arena: free pools + never-  
     * allocated pools.  
     */  
    uint nfreepools;  
  
    /* The total number of pools in the arena, whether or not available. */  
    uint ntotalpools;  
  
    /* Singly-linked list of available pools. */  
    struct pool_header* freepools;  
  
    struct arena_object* nextarena;  
    struct arena_object* prevarena;  
};
```



Типизация

03

Типизация – динамическая

~~Типизация~~
Аннотация типов



```
a: int = 10  
b: float = 1.0  
c: str = "Hi"
```

Подсказки программисту, которые очень упрощают работу

Базовые аннотации:

```
a: int = 10  
b: float = 1.0  
c: str = "Hi"
```

```
obj_dict: dict[str, int] = {  
    "key" : 12,  
    "key_2": 13  
}
```

```
obj_list: list[int] = [1, 2, 3]
```

```
obj_tuple: tuple[str, int, str] = ("str", 10, "str2")
```

```
obj_set: set[str] = set("snt1", "str2", "str3")
```

Аннотации простых объектов, списков, словарей, кортежей и сетов

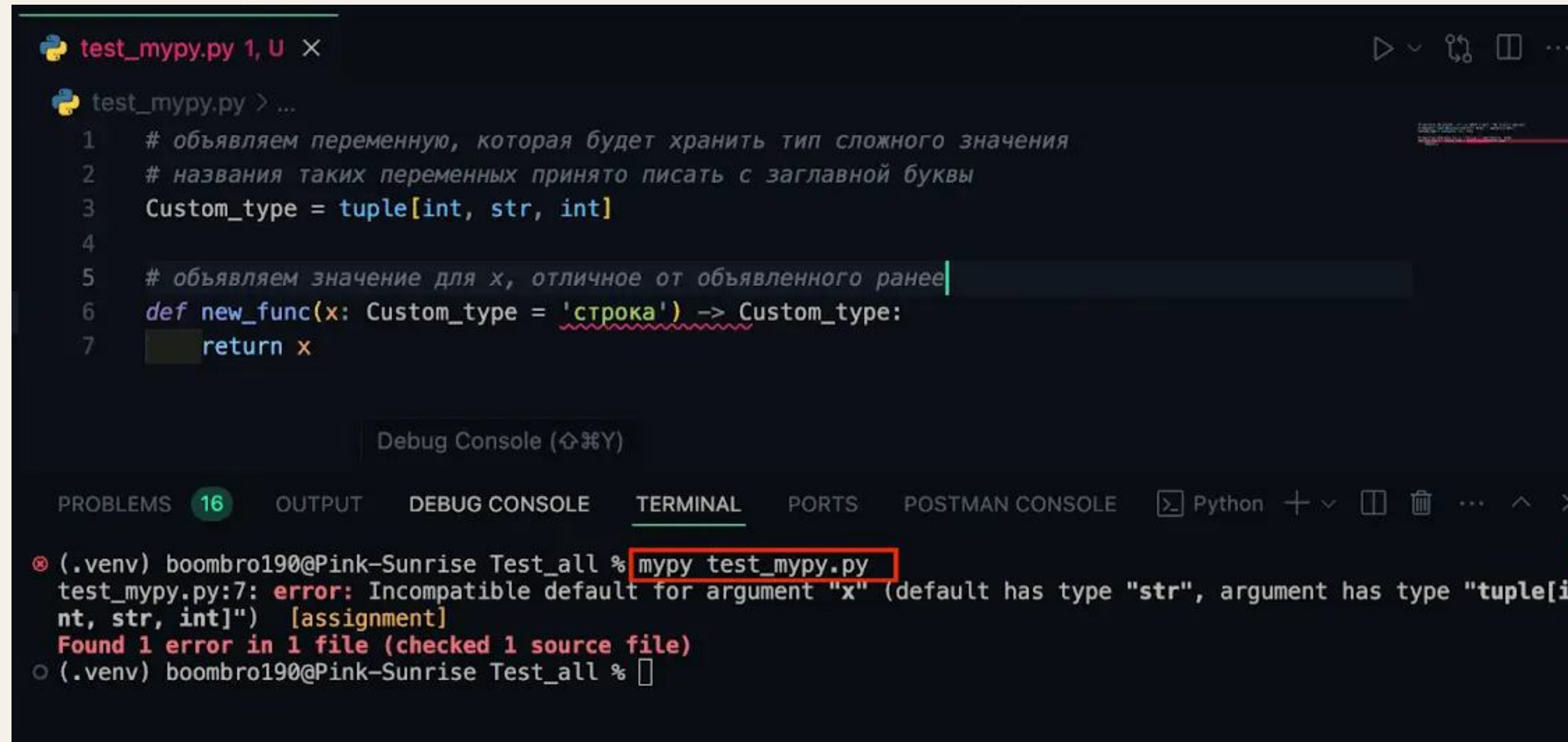
- Из библиотеки `typing` можно импортировать другие аннотации
- Через стрелочку можно указать `return type`

```
from typing import Callable, Optional

def call_func(
    inner_func: Callable[..., float],
    opt_param: Optional[int] = None
) -> float:
    pass
```


Интерпретатор, конечно же не ругается на неверные типы, даже если прописать аннотации

Можно использовать другие инструменты, например, мур



```
test_mypy.py 1, U x
test_mypy.py > ...
1  # объявляем переменную, которая будет хранить тип сложного значения
2  # названия таких переменных принято писать с заглавной буквы
3  Custom_type = tuple[int, str, int]
4
5  # объявляем значение для x, отличное от объявленного ранее
6  def new_func(x: Custom_type = 'строка') -> Custom_type:
7      return x

Debug Console (⇧⌘Y)
PROBLEMS 16 OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE Python + - [ ] [ ] ... ^
⊗ (.venv) boombro190@Pink-Sunrise Test_all % mypy test_mypy.py
test_mypy.py:7: error: Incompatible default for argument "x" (default has type "str", argument has type "tuple[int, str, int]") [assignment]
Found 1 error in 1 file (checked 1 source file)
○ (.venv) boombro190@Pink-Sunrise Test_all %
```

Зачем прописывать аннотации типов?

- Супер упрощают жизнь и уменьшают боль при работе с нетипизированным питоном
- Делают код более читабельным
- Способствуют удобной поддержке кодовой базы проекта
- Помогают быстрее понять код и/или вспомнить о чем свой же давно написанный код

Кроме того, можно использовать датаклассы, простые структуры данных, которые также можно типизировать, обращаться к объекту через точку

```
from dataclasses import dataclass

@dataclass
class Coordinate:
    x: int
    y: int
    z: int

coord = Coordinate(
    x=1,
    y=2,
    z=3
)

print(coord.x)
```

На этом всё)

На следующем
занятии начнем ООП
на C++