РОСНЕФТЬ

ВНУТРЕННЕЕ УСТРОЙСТВО СБОРКИ МУСОРА В CPYTHON 3.14+

МИРЯНОВ СЕРГЕЙ
РН-ТЕХНОЛОГИИ

UFADEVCONF 2025

- Старший эксперт в **РН-ТЕХНОЛОГИИ**
- Мы занимаемся разработкой наукоемкого десктопного ПО
- Для нефтянки и не только

```
https://github.com/python/cpython
```

```
3.15 @ a17c57eee5b5cc81390750d07e4800b19c0c3084
3.14 @ ebf955df7a89ed0c7968f79faec1de49f61ed7cb
3.13 @ 9ab89c026aa9611c4b0b67c288b8303a480fe742
```

РОСНЕФТЬ

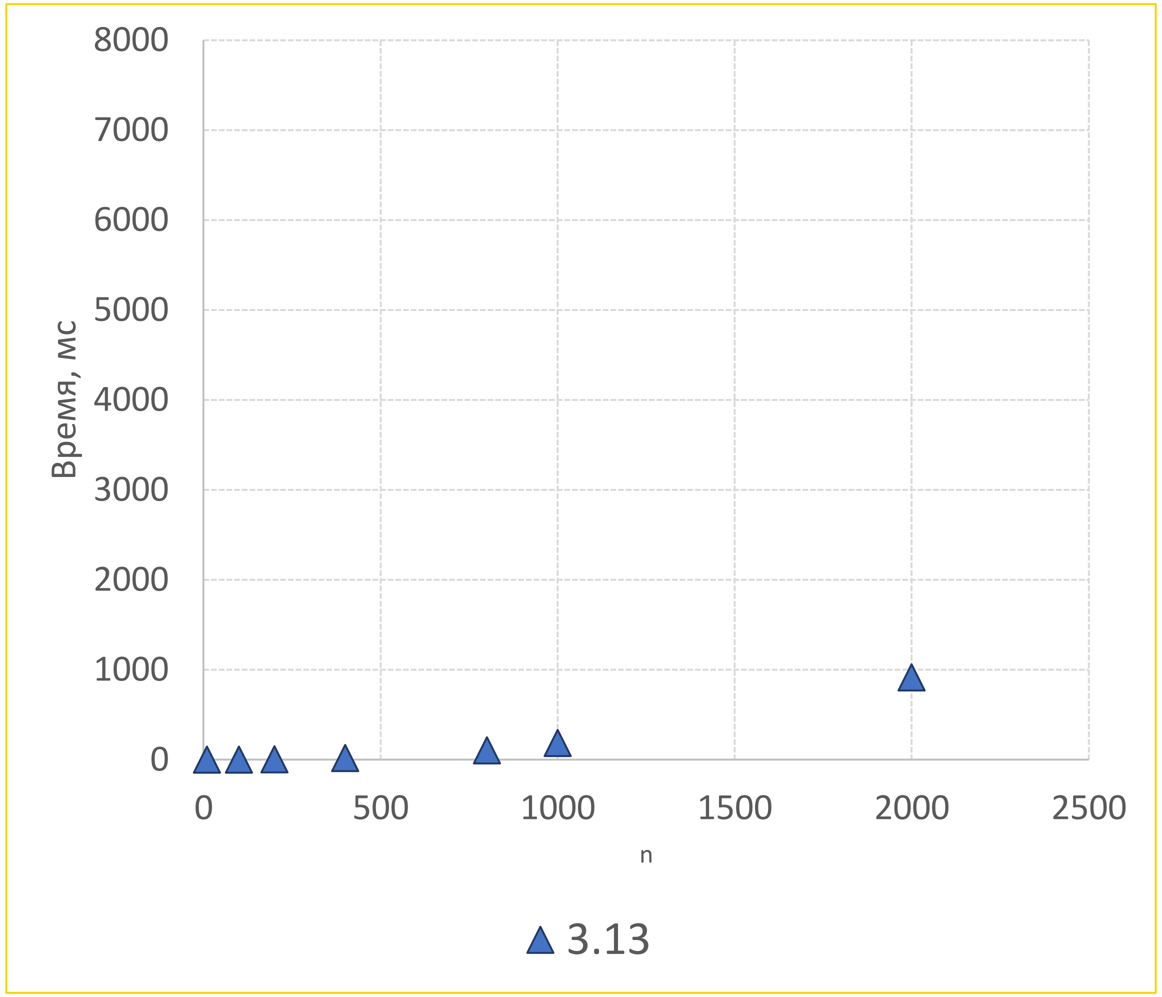
ВИДЕО

СЛАЙДЫ

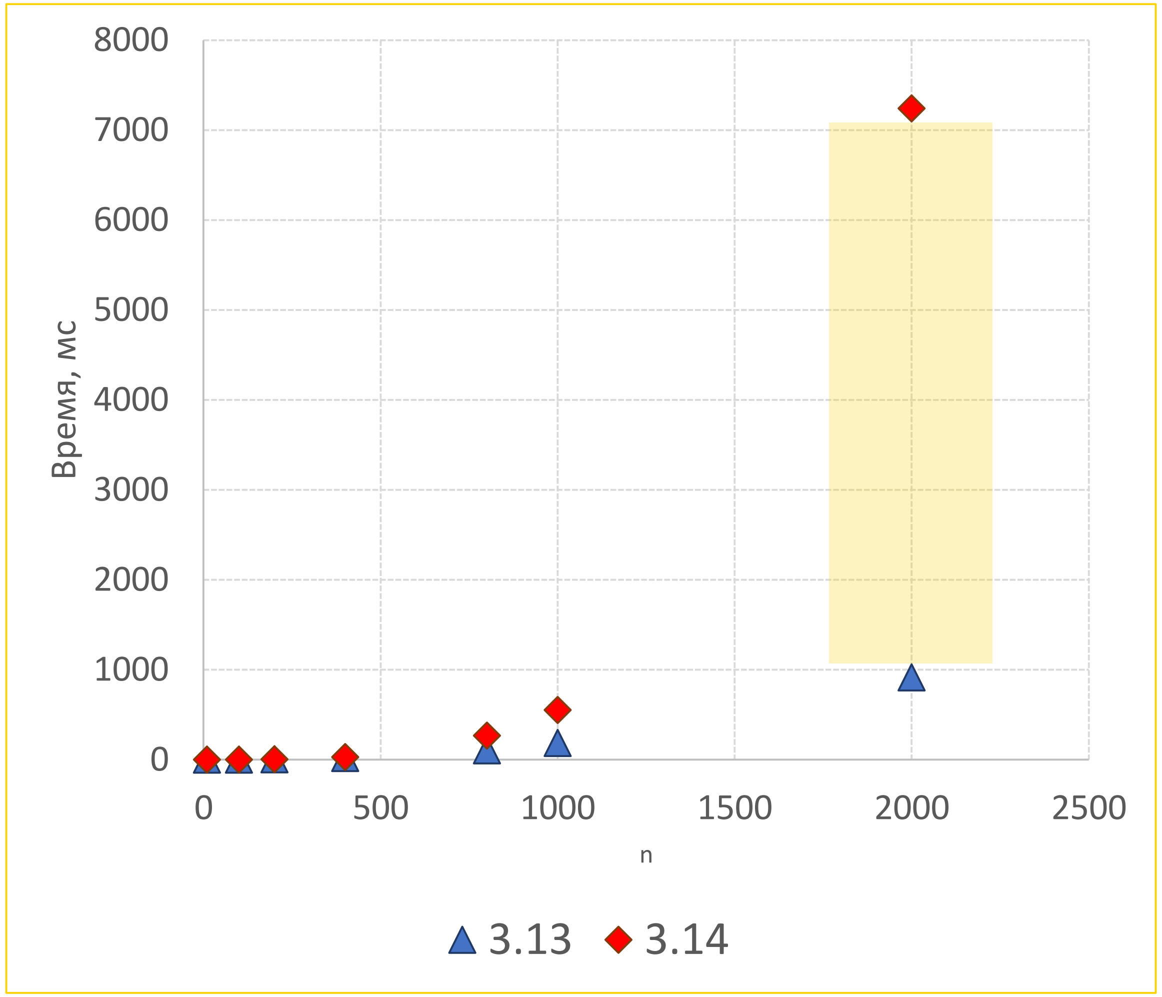
ПРИМЕР

```python
def test(n=2000):
    d = {}
    for i in range(n):
        for j in range(n):
            d[(i,j)] = i
```

```
def test(n=2000):
    d = {}
    for i in range(n):
        for j in range(n):
            d[(i,j)] = i
```



```
def test(n=2000):
    d = {}
    for i in range(n):
        for j in range(n):
            d[(i,j)] = i
```



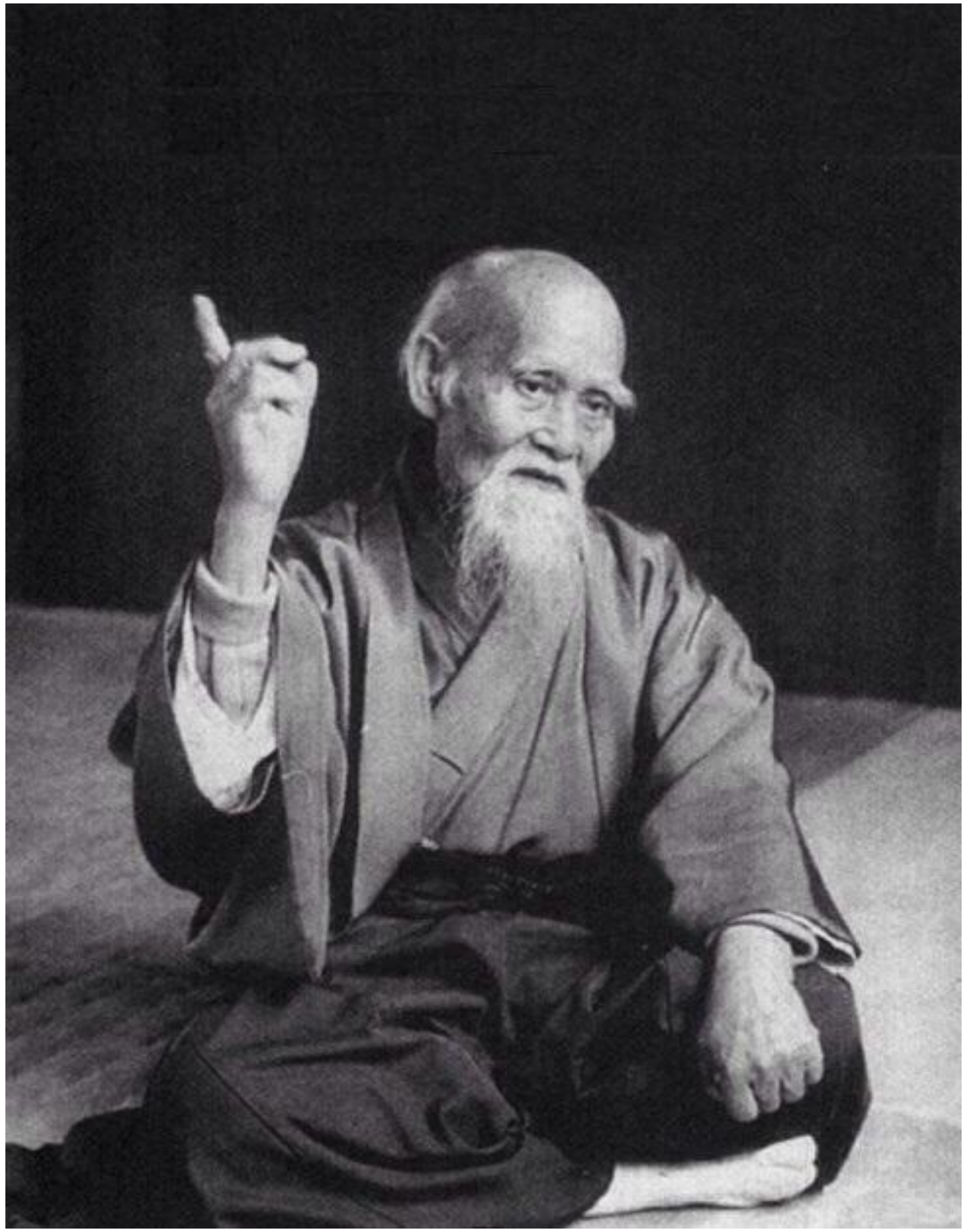
ЧТО ПРОИСХОДИТ?

- Подсчет ссылок

- Подсчет ссылок
- Объекты-контейнеры

- Подсчет ссылок
- Объекты-контейнеры
- Циклические ссылки

- Подсчет ссылок
- Объекты-контейнеры
- Циклические ссылки
- Сборка мусора



Measurements of object lifetimes proved that ***young objects die young*** *and* ***old objects continue to live***

- Подсчет ссылок
- Объекты-контейнеры
- Циклические ссылки
- Сборка мусора **с поколениями**

```
def test(n=2000):
    d = {}
    for i in range(n):
        for j in range(n):
            d[(i,j)] = i
```

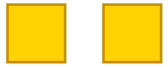
```
gc.set_threshold(
  2000,  // Объектов в молодом поколении
  10,    // Количество сборок молодого поколения
  10     // Количество сборок среднего поколения*
)
```

```
def test(n=2000):
    d = {}
    for i in range(n):
        for j in range(n):
            d[(i,j)] = i
```

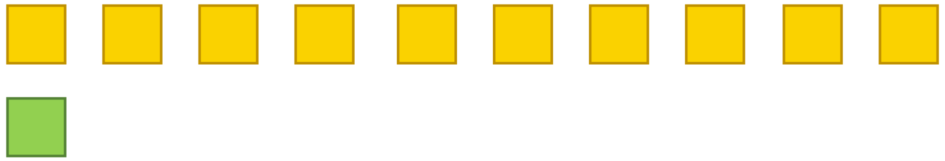
```python
def test(n=2000):
    d = {}
    for i in range(n):
        for j in range(n):
            d[(i,j)] = i
```

🟨 🟩

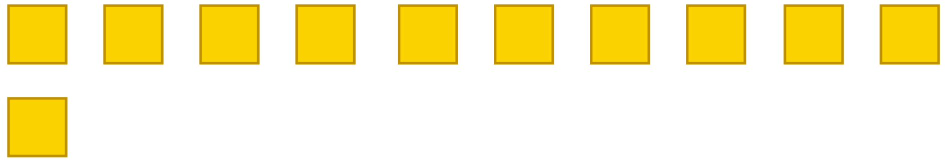
```
def test(n=2000):
    d = {}
    for i in range(n):
        for j in range(n):
            d[(i,j)] = i
```



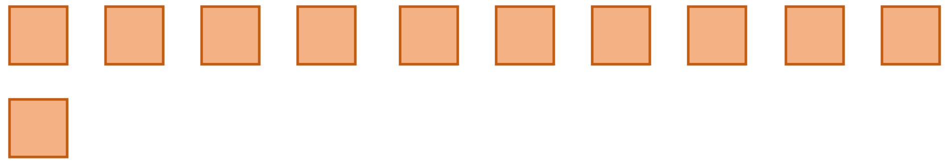
```
def test(n=2000):
    d = {}
    for i in range(n):
        for j in range(n):
            d[(i,j)] = i
```



```
def test(n=2000):
    d = {}
    for i in range(n):
        for j in range(n):
            d[(i,j)] = i
```

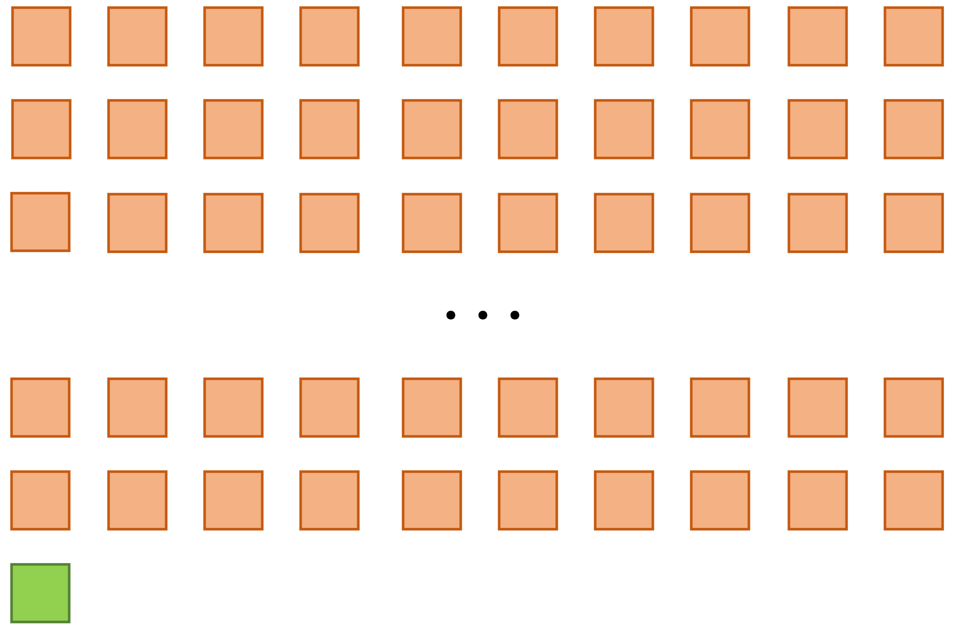


```
def test(n=2000):
    d = {}
    for i in range(n):
        for j in range(n):
            d[(i,j)] = i
```

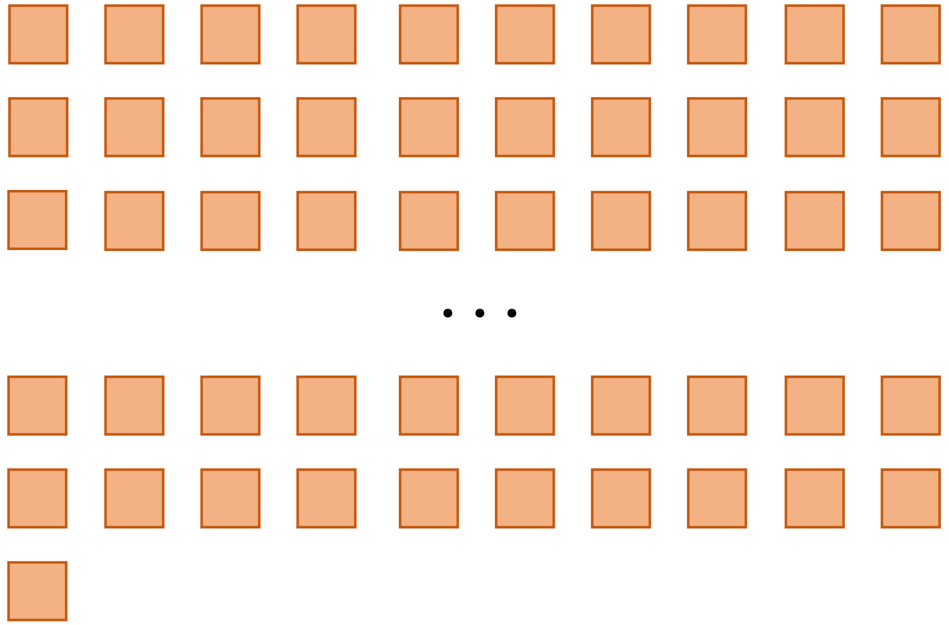


```
def test(n=2000):
    d = {}
    for i in range(n):
        for j in range(n):
            d[(i,j)] = i
```

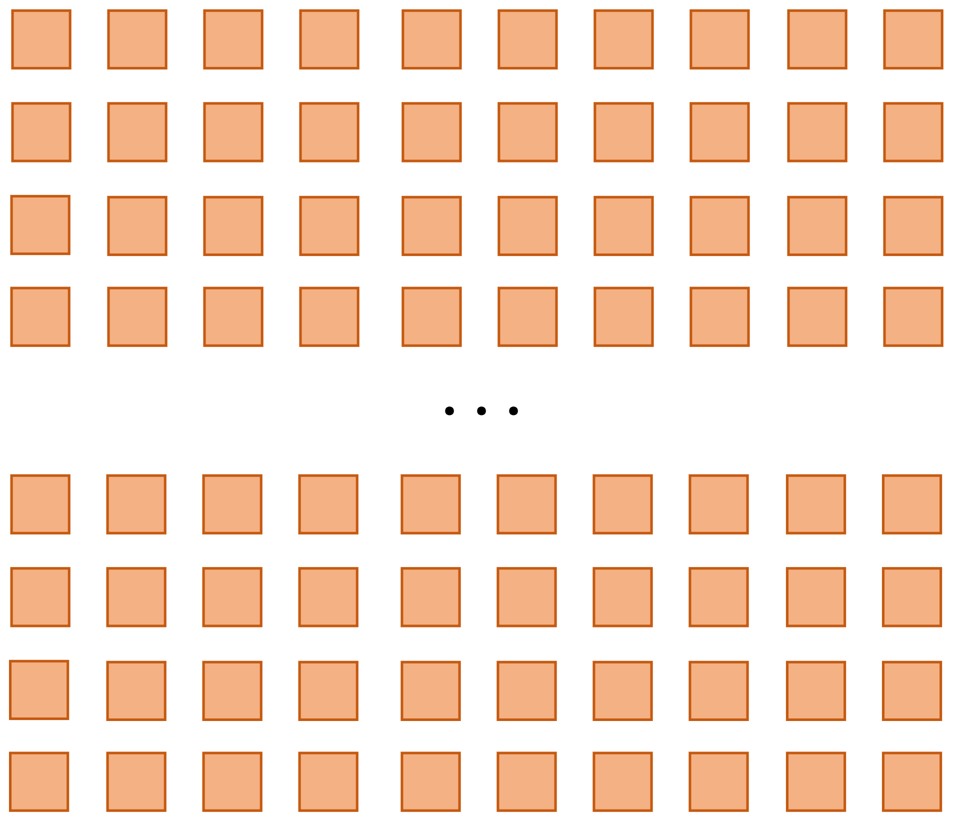
```
def test(n=2000):
    d = {}
    for i in range(n):
        for j in range(n):
            d[(i,j)] = i
```



```
def test(n=2000):
    d = {}
    for i in range(n):
        for j in range(n):
            d[(i,j)] = i
```



```
def test(n=2000):
    d = {}
    for i in range(n):
        for j in range(n):
            d[(i,j)] = i
```



```python
def test(n=2000):
    d = {}
    for i in range(n):
        for j in range(n):
            d[(i,j)] = i
```

```
Disassembly of <line 2>:
  2           RESUME                   0

  3           BUILD_MAP                0
              STORE_FAST               1 (d)

  4           LOAD_GLOBAL              1 (range + NULL)
              LOAD_FAST                0 (n)
              CALL                     1
              GET_ITER
      L1:     FOR_ITER                26 (to L4)
              STORE_FAST               2 (i)

             ...

              JUMP_BACKWARD           28 (to L1)

  4   L4:     END_FOR
              POP_TOP
              RETURN_CONST             0 (None)
```

```python
def test(n=2000):
    d = {}
    for i in range(n):
        for j in range(n):
            d[(i,j)] = i
```

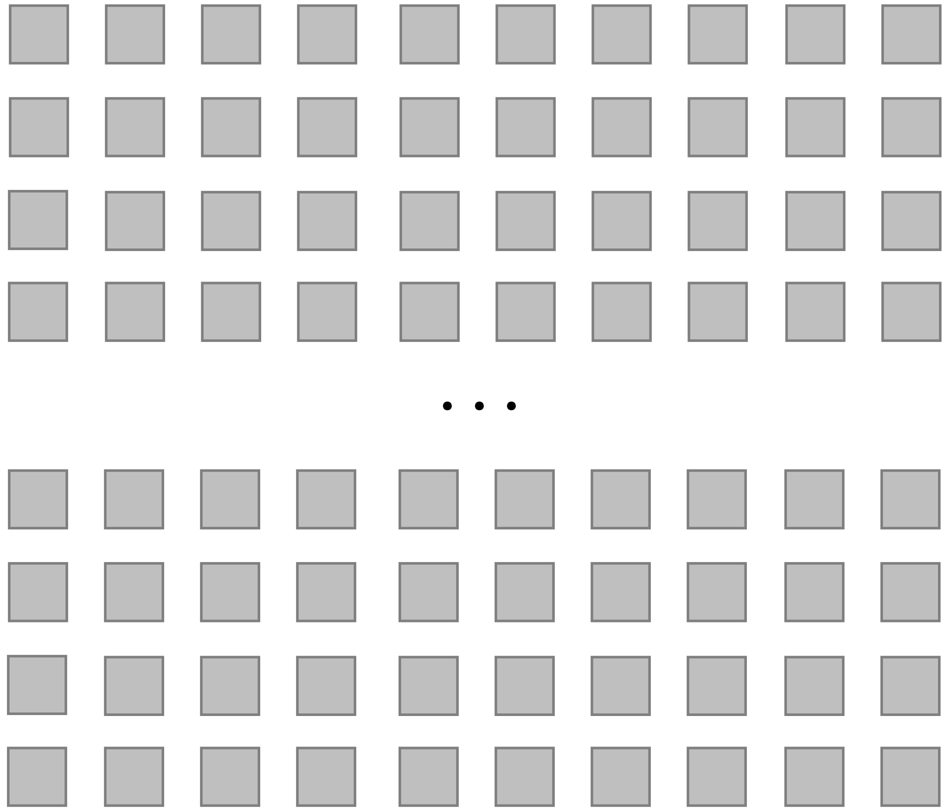
```
Disassembly of <line 2>:
  2           RESUME                  0

  3           BUILD_MAP               0
              STORE_FAST              1 (d)

  4           LOAD_GLOBAL             1 (range + NULL)
              LOAD_FAST               0 (n)
              CALL                    1
              GET_ITER
        L1:   FOR_ITER               26 (to L4)
              STORE_FAST              2 (i)

             ...

              JUMP_BACKWARD          28 (to L1)

  4     L4:   END_FOR
              POP_TOP
              RETURN_CONST            0 (None)
```

```python
def test(n=2000):
    d = {}
    for i in range(n):
        for j in range(n):
            d[(i,j)] = i
```

```c
inst(INSTRUMENTED_RETURN_CONST, (--)) {
    PyObject *retval = GETITEM(FRAME_CO_CONSTS, op
    int err = _Py_call_instrumentation_arg(
            tstate, PY_MONITORING_EVENT_PY_RETURN,
            frame, this_instr, retval);
    if (err) ERROR_NO_POP();
    Py_INCREF(retval);
    _PyFrame_SetStackPointer(frame, stack_pointer)
    _Py_LeaveRecursiveCallPy(tstate);
    _PyInterpreterFrame *dying = frame;
    frame = tstate->current_frame = dying->previou
    _PyEval_FrameClearAndPop(tstate, dying);
    _PyFrame_StackPush(frame, retval);
    LOAD_IP(frame->return_offset);
    goto resume_frame;
}
```

```python
def test(n=2000):
    d = {}
    for i in range(n):
        for j in range(n):
            d[(i,j)] = i
```

```c
void
_PyFrame_ClearLocals(_PyInterpreterFrame *frame)
{
    assert(frame->stacktop >= 0);
    int stacktop = frame->stacktop;
    frame->stacktop = 0;
    for (int i = 0; i < stacktop; i++) {
        Py_XDECREF(frame->localsplus[i]);
    }
    Py_CLEAR(frame->f_locals);
}
```



```
void
_PyFrame_ClearLocals(_PyInterpreterFrame *frame)
{
    assert(frame->stacktop >= 0);
    int stacktop = frame->stacktop;
    frame->stacktop = 0;
    for (int i = 0; i < stacktop; i++) {
        Py_XDECREF(frame->localsplus[i]);
    }
    Py_CLEAR(frame->f_locals);
}
```

```python
def test(n=2000):
    d = {}
    for i in range(n):
        for j in range(n):
            d[(i,j)] = i
```

```
200 000
400 000
600 000
...
3 600 000
3 800 000
4 000 000
```

→ **Поиск недостижимых**
→ Определение несобираемых
→ Обработка слабых ссылок
→ Удаление объектов
→ Определение воскрешенных объектов
→ Очистка слабых ссылок
→ Очистка циклов
→ Перенос несобираемых

```
200 000
400 000
600 000
...
3 600 000
3 800 000
4 000 000
```

ИНКРЕМЕНТАЛЬНАЯ СБОРКА МУСОРА

```
gc.set_threshold(
  2000,  // Объектов в молодом поколении
  10,    // Доля старшего поколения
  0      // Не используется
)
```

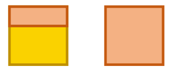
```
def test(n=2000):
    d = {}
    for i in range(n):
        for j in range(n):
            d[(i,j)] = i
```

```
def test(n=2000):
    d = {}
    for i in range(n):
        for j in range(n):
            d[(i,j)] = i
```

```
def test(n=2000):
    d = {}
    for i in range(n):
        for j in range(n):
            d[(i,j)] = i
```

🟨 🟩

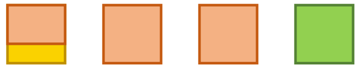
```
def test(n=2000):
    d = {}
    for i in range(n):
        for j in range(n):
            d[(i,j)] = i
```



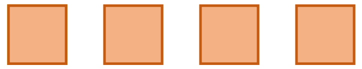
```
def test(n=2000):
    d = {}
    for i in range(n):
        for j in range(n):
            d[(i,j)] = i
```

```
def test(n=2000):
    d = {}
    for i in range(n):
        for j in range(n):
            d[(i,j)] = i
```

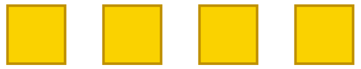
```
def test(n=2000):
    d = {}
    for i in range(n):
        for j in range(n):
            d[(i,j)] = i
```



```
def test(n=2000):
    d = {}
    for i in range(n):
        for j in range(n):
            d[(i,j)] = i
```

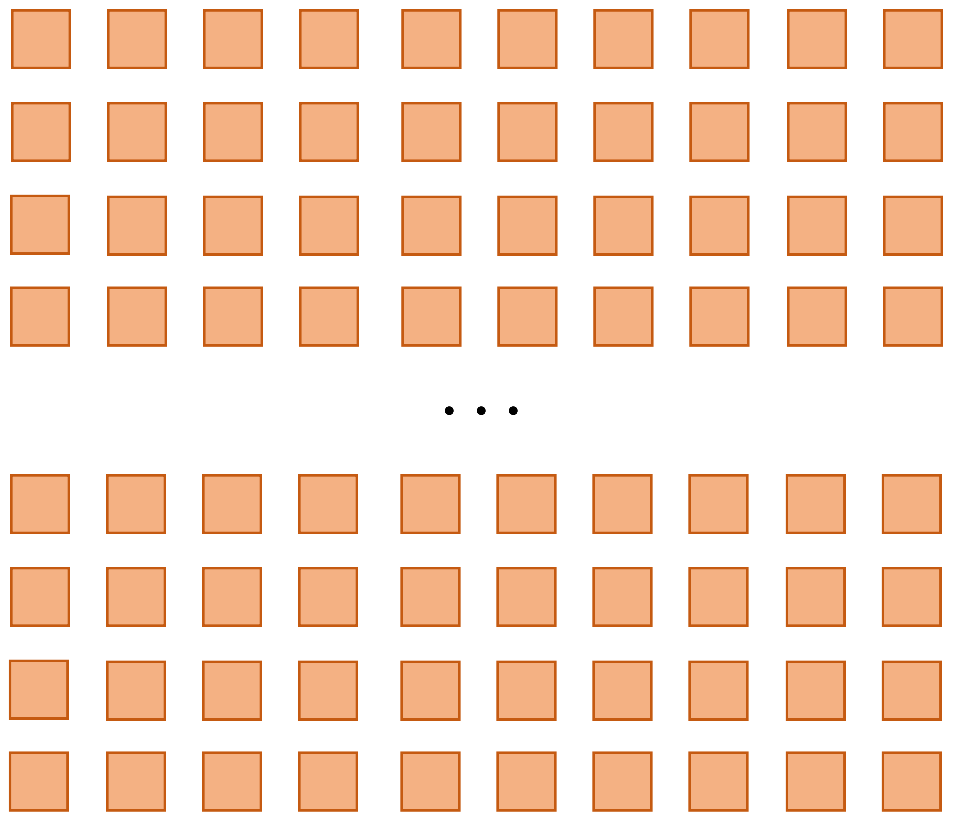


```
def test(n=2000):
    d = {}
    for i in range(n):
        for j in range(n):
            d[(i,j)] = i
```

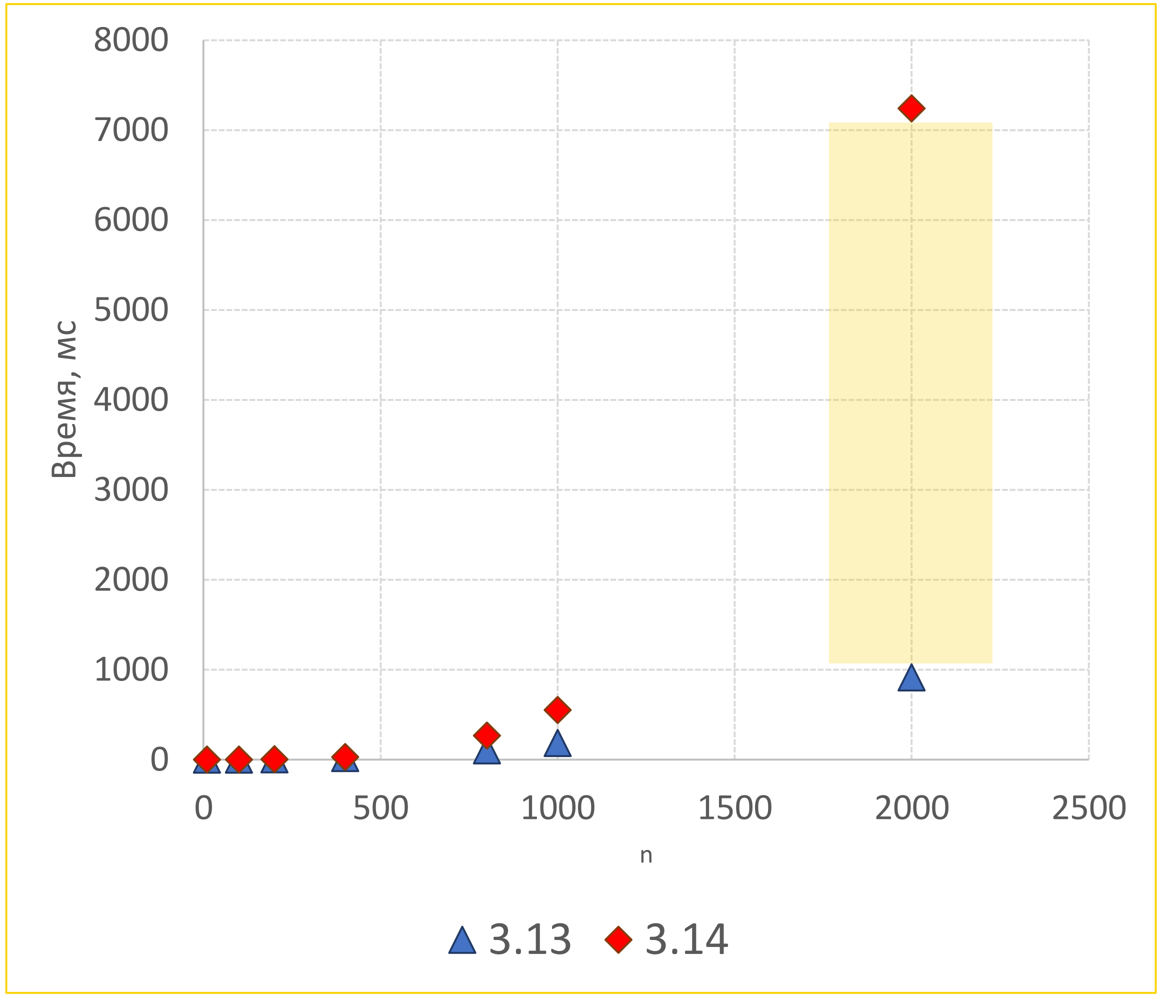


```
def test(n=2000):
    d = {}
    for i in range(n):
        for j in range(n):
            d[(i,j)] = i
```

```
def test(n=2000):
    d = {}
    for i in range(n):
        for j in range(n):
            d[(i,j)] = i
```



```python
def test(n=2000):
    d = {}
    for i in range(n):
        for j in range(n):
            d[(i,j)] = i
```



РОСНЕФТЬ

```c
static void
gc_collect_increment(PyThreadState *tstate, struct gc_collection_stats *stats)
{
    GCState *gcstate = &tstate->interp->gc;
    gcstate->work_to_do += assess_work_to_do(gcstate);
    if (gcstate->work_to_do < 0) {
        return;
    }
    untrack_tuples(&gcstate->young.head);
    if (gcstate->phase == GC_PHASE_MARK) {
        Py_ssize_t objects_marked = mark_at_start(tstate);
        gcstate->work_to_do -= objects_marked;
        return;
    }
    PyGC_Head *not_visited = &gcstate->old[gcstate->visited_space^1].head;
    PyGC_Head *visited = &gcstate->old[gcstate->visited_space].head;
    PyGC_Head increment;
    gc_list_init(&increment);
    int scale_factor = gcstate->old[0].threshold;
    if (scale_factor < 2) {
        scale_factor = 2;
    }
    intptr_t objects_marked = mark_stacks(tstate->interp, visited, gcstate->visited_space, false);
    gcstate->work_to_do -= objects_marked;
    gc_list_set_space(&gcstate->young.head, gcstate->visited_space);
    gc_list_merge(&gcstate->young.head, &increment);
    gc_list_validate_space(&increment, gcstate->visited_space);
    Py_ssize_t increment_size = gc_list_size(&increment);
    while (increment_size < gcstate->work_to_do) {
        if (gc_list_is_empty(not_visited)) {
            break;
        }
        PyGC_Head *gc = _PyGCHead_NEXT(not_visited);
        gc_list_move(gc, &increment);
        increment_size++;
        gc_set_old_space(gc, gcstate->visited_space);
        increment_size += expand_region_transitively_reachable(&increment, gc, gcstate);
    }
    PyGC_Head survivors;
    gc_list_init(&survivors);
    gc_collect_region(tstate, &increment, &survivors, stats);
    gc_list_merge(&survivors, visited);
    gcstate->work_to_do -= increment_size;
    if (gc_list_is_empty(not_visited)) {
        completed_scavenge(gcstate);
    }
}
```

1

2

3

4

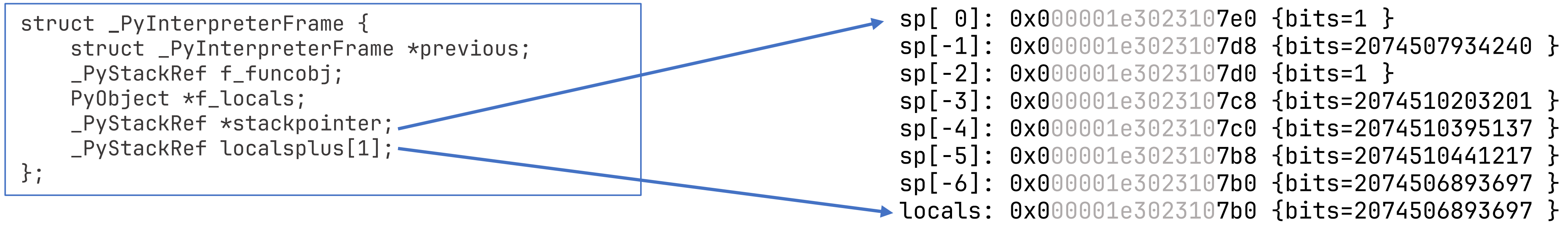
```c
static intptr_t
mark_at_start(PyThreadState *tstate)
{
    // TO DO -- Make this incremental
    GCState *gcstate = &tstate->interp->gc;
    PyGC_Head *visited = &gcstate->old[gcstate->visited_space].head;
    Py_ssize_t objects_marked = mark_global_roots(tstate->interp, visited, gcstate->visited_space);
    objects_marked += mark_stacks(tstate->interp, visited, gcstate->visited_space, true);
    gcstate->work_to_do -= objects_marked;
    gcstate->phase = GC_PHASE_COLLECT;
    validate_spaces(gcstate);
    return objects_marked;
}
```

```c
static intptr_t
mark_global_roots(PyInterpreterState *interp, PyGC_Head *visited, int visited_space)
{
    PyGC_Head reachable;
    gc_list_init(&reachable);
    Py_ssize_t objects_marked = 0;
    objects_marked += move_to_reachable(interp->sysdict, &reachable, visited_space);
    objects_marked += move_to_reachable(interp->builtins, &reachable, visited_space);
    objects_marked += move_to_reachable(interp->dict, &reachable, visited_space);
    // ...
    objects_marked += mark_all_reachable(&reachable, visited, visited_space);
    return objects_marked;
}
```

```
static intptr_t
mark_global_roots(PyInterpreterState *interp, PyGC_Head *visited, int visited_space)
{
    PyGC_Head reachable;
    gc_list_init(&reachable);
    Py_ssize_t objects_marked = 0;
    objects_marked += move_to_reachable(interp->sysdict, &reachable, visited_space);
    objects_marked += move_to_reachable(interp->builtins, &reachable, visited_space);
    objects_marked += move_to_reachable(interp->dict, &reachable, visited_space);
    // ...
    objects_marked += mark_all_reachable(&reachable, visited, visited_space);
    return objects_marked;
}
```

```c
static intptr_t
mark_stacks(PyInterpreterState *interp, PyGC_Head *visited, int visited_space, bool start)
{
    PyGC_Head reachable;
    gc_list_init(&reachable);
    Py_ssize_t objects_marked = 0;
    // Move all objects on stacks to reachable
    _PyRuntimeState *runtime = &_PyRuntime;
    HEAD_LOCK(runtime);
    PyThreadState* ts = PyInterpreterState_ThreadHead(interp);
    HEAD_UNLOCK(runtime);
    while (ts) {
        _PyInterpreterFrame *frame = ts->current_frame;
        while (frame) {
            if (frame->owner >= FRAME_OWNED_BY_INTERPRETER) {
                frame = frame->previous;
                continue;
            }
            _PyStackRef *locals = frame->localsplus;
            _PyStackRef *sp = frame->stackpointer;
            objects_marked += move_to_reachable(frame->f_locals, &reachable, visited_space);
            PyObject *func = PyStackRef_AsPyObjectBorrow(frame->f_funcobj);
            objects_marked += move_to_reachable(func, &reachable, visited_space);
            while (sp > locals) {
                sp--;
                if (PyStackRef_IsNullOrInt(*sp)) {
                    continue;
                }
                PyObject *op = PyStackRef_AsPyObjectBorrow(*sp);
                objects_marked += move_to_reachable(op, &reachable, visited_space);
            }
            if (!start && frame->visited) {
                // If this frame has already been visited, then the lower frames
                // will have already been visited and will not have changed
                break;
            }
            frame->visited = 1;
            frame = frame->previous;
        }
        HEAD_LOCK(runtime);
        ts = PyThreadState_Next(ts);
        HEAD_UNLOCK(runtime);
    }
    objects_marked += mark_all_reachable(&reachable, visited, visited_space);
    assert(gc_list_is_empty(&reachable));
    return objects_marked;
}
```

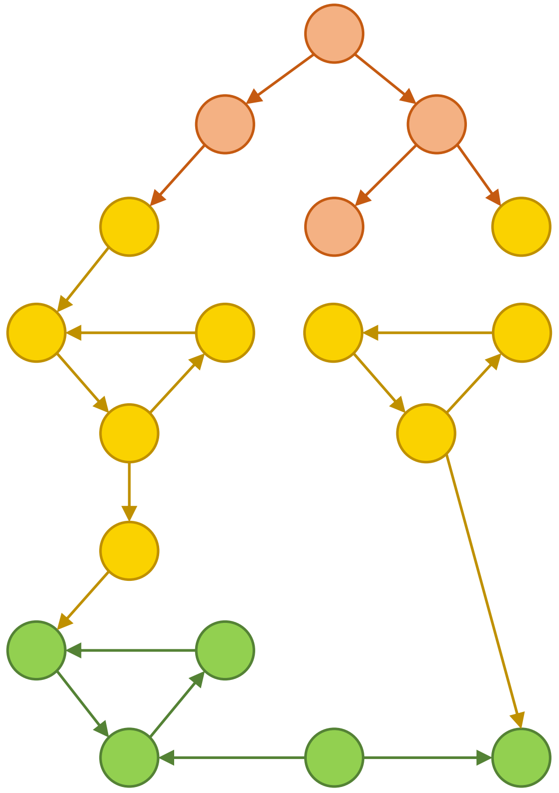
```c
static intptr_t
mark_stacks(PyInterpreterState *interp, PyGC_Head *visited, int visited_space, bool start)
{
    // ...
    _PyStackRef *locals = frame->localsplus;
    _PyStackRef *sp = frame->stackpointer;
    objects_marked += move_to_reachable(frame->f_locals, &reachable, visited_space);
    PyObject *func = PyStackRef_AsPyObjectBorrow(frame->f_funcobj);
    objects_marked += move_to_reachable(func, &reachable, visited_space);
    while (sp > locals) {
        sp--;
        if (PyStackRef_IsNullOrInt(*sp)) {
            continue;
        }
        PyObject *op = PyStackRef_AsPyObjectBorrow(*sp);
        objects_marked += move_to_reachable(op, &reachable, visited_space);
    }
    // ...
}
```



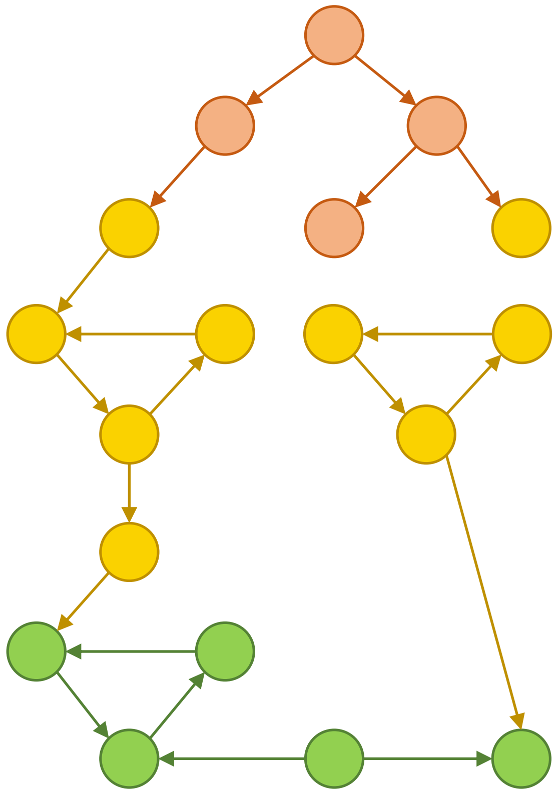
```c
struct _PyInterpreterFrame {
    struct _PyInterpreterFrame *previous;
    _PyStackRef f_funcobj;
    PyObject *f_locals;
    _PyStackRef *stackpointer;
    _PyStackRef localsplus[1];
};
```

```
sp[ 0]: 0x000001e3023107e0 {bits=1 }
sp[-1]: 0x000001e3023107d8 {bits=207450793424 }
sp[-2]: 0x000001e3023107d0 {bits=1 }
sp[-3]: 0x000001e3023107c8 {bits=2074510203201 }
sp[-4]: 0x000001e3023107c0 {bits=2074510395137 }
sp[-5]: 0x000001e3023107b8 {bits=2074510441217 }
sp[-6]: 0x000001e3023107b0 {bits=2074506893697 }
locals: 0x000001e3023107b0 {bits=2074506893697 }
```

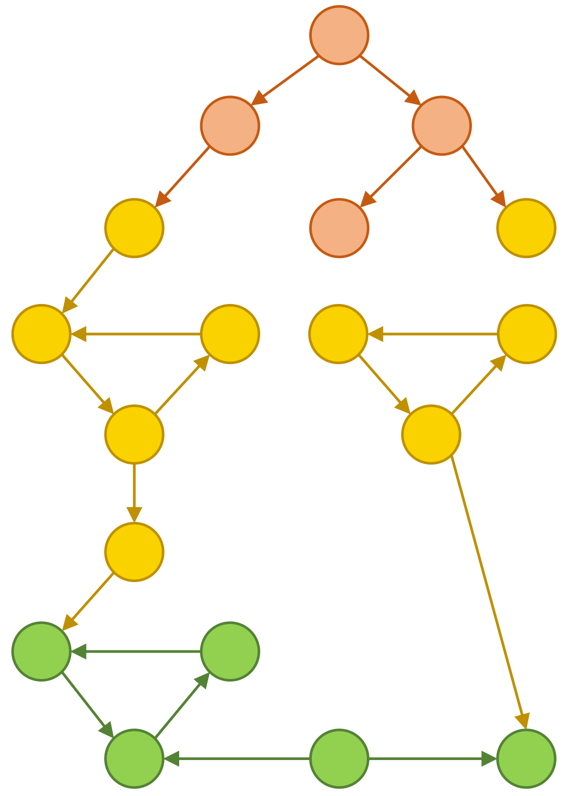
```
static intptr_t
mark_stacks(PyInterpreterState *interp, PyGC_Head *visited, int visited_space, bool start)
{
    // ...
    objects_marked += mark_all_reachable(&reachable, visited, visited_space);
    assert(gc_list_is_empty(&reachable));
    return objects_marked;
}
```



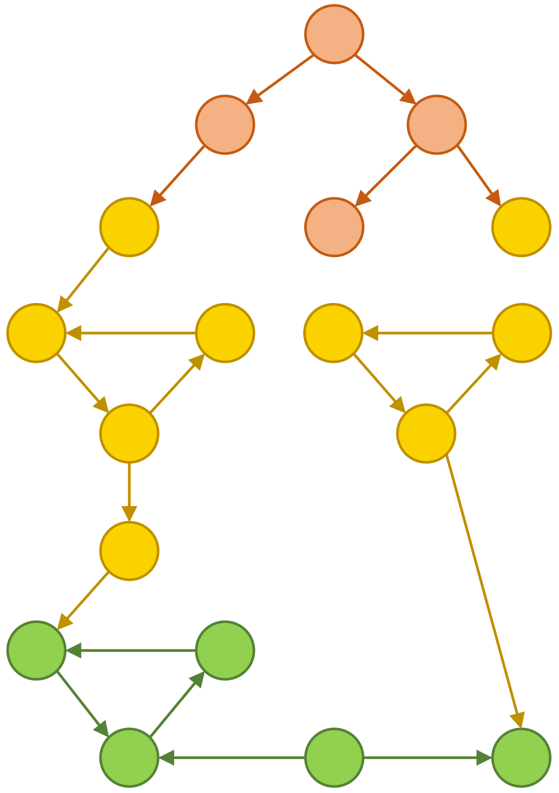
```
static intptr_t
mark_all_reachable(PyGC_Head *reachable, PyGC_Head
*visited, int visited_space)
{
    struct container_and_flag arg = {
        .container = reachable,
        .visited_space = visited_space,
        .size = 0
    };
    while (!gc_list_is_empty(reachable)) {
        PyGC_Head *gc = _PyGCHead_NEXT(reachable);
        gc_list_move(gc, visited);
        PyObject *op = FROM_GC(gc);
        traverseproc traverse = Py_TYPE(op)->tp_traverse;
        (void) traverse(op, visit_add_to_container, &arg);
    }
    return arg.size;
}
```



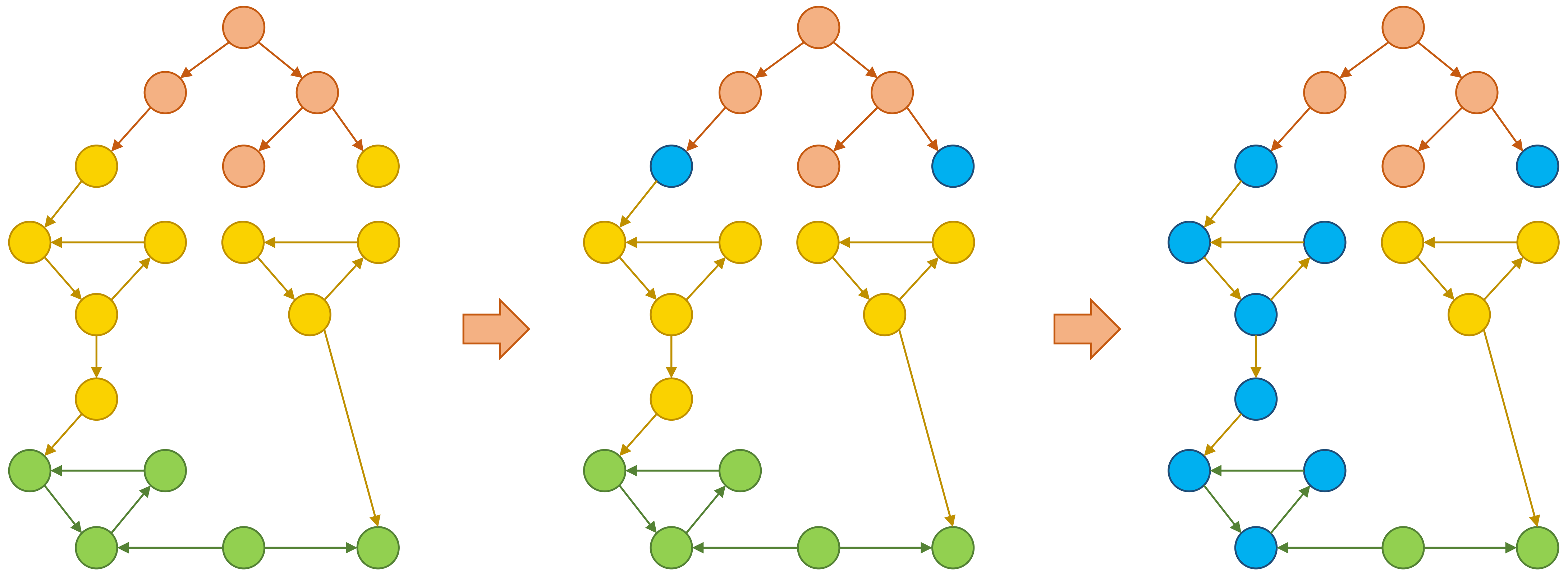
```
static intptr_t
mark_all_reachable(PyGC_Head *reachable, PyGC_Head
*visited, int visited_space)
{
    struct container_and_flag arg = {
        .container = reachable,
        .visited_space = visited_space,
        .size = 0
    };
    while (!gc_list_is_empty(reachable)) {
        PyGC_Head *gc = _PyGCHead_NEXT(reachable);
        gc_list_move(gc, visited);
        PyObject *op = FROM_GC(gc);
        traverseproc traverse = Py_TYPE(op)->tp_traverse;
        (void) traverse(op, visit_add_to_container, &arg);
    }
    return arg.size;
}
```



```
static int
visit_add_to_container(PyObject *op, void *arg)
{
    container_and_flag *cf = (container_and_flag *)arg;
    int visited = cf->visited_space;
    if (!_Py_IsImmortal(op) && _PyObject_IS_GC(op)) {
        PyGC_Head *gc = AS_GC(op);
        if (_PyObject_GC_IS_TRACKED(op) &&
            gc_old_space(gc) != visited) {
            gc_flip_old_space(gc);
            gc_list_move(gc, cf->container);
            cf->size++;
        }
    }
    return 0;
}
```



```
static intptr_t
mark_all_reachable(PyGC_Head *reachable, PyGC_Head
*visited, int visited_space)
{
    struct container_and_flag arg = {
        .container = reachable,
        .visited_space = visited_space,
        .size = 0
    };
    while (!gc_list_is_empty(reachable)) {
        PyGC_Head *gc = _PyGCHead_NEXT(reachable);
        gc_list_move(gc, visited);
        PyObject *op = FROM_GC(gc);
        traverseproc traverse = Py_TYPE(op)->tp_traverse;
        (void) traverse(op, visit_add_to_container, &arg);
    }
    return arg.size;
}
```

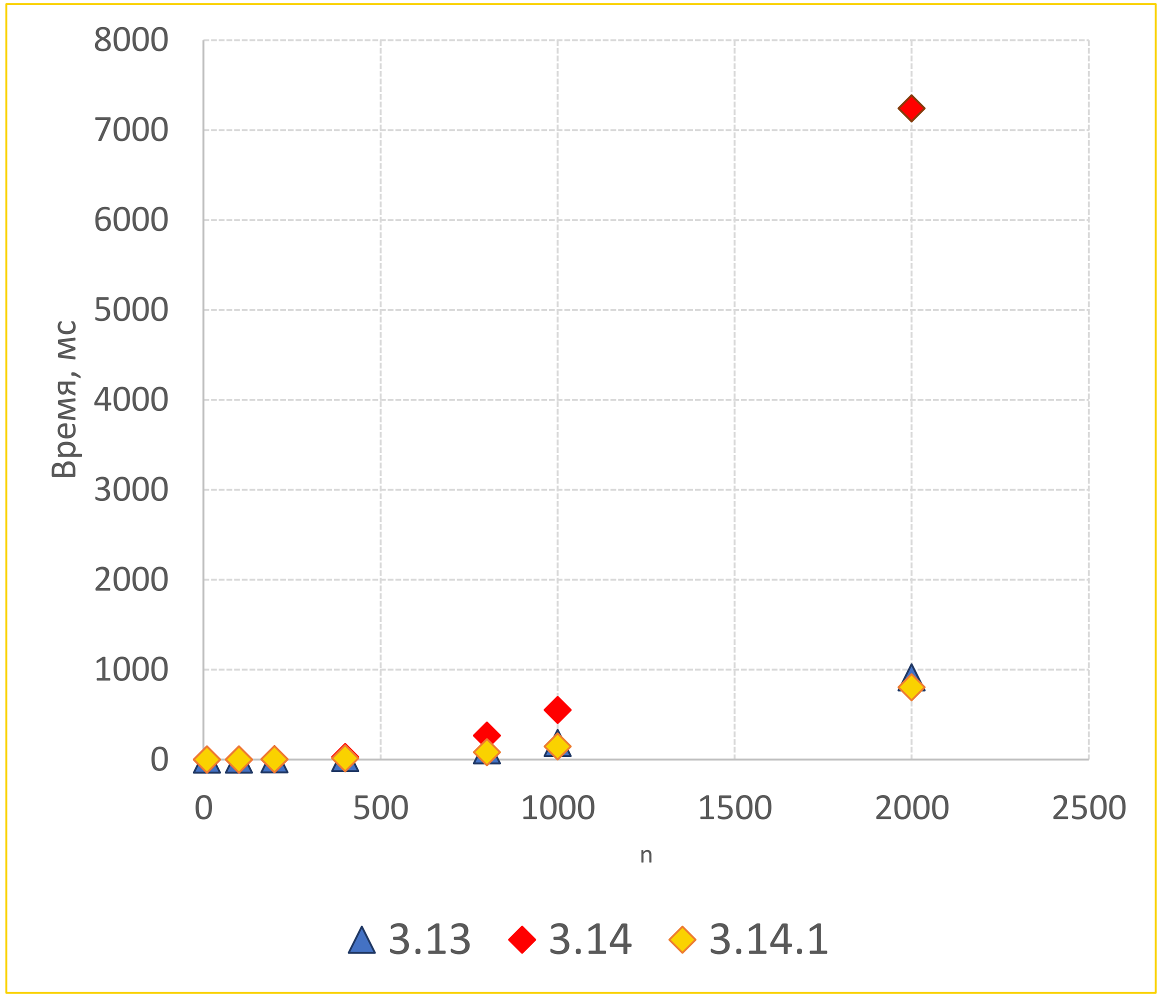


FIN

```python
def test(n=2000):
    d = {}
    for i in range(n):
        for j in range(n):
            d[(i,j)] = i
```

```c
static int
visit_add_to_container(PyObject *op, void *arg)
{
    container_and_flag *cf = (container_and_flag *)arg;
    int visited = cf->visited_space;
    if (!_Py_IsImmortal(op) && _PyObject_IS_GC(op)) {
        PyGC_Head *gc = AS_GC(op);
        if (_PyObject_GC_IS_TRACKED(op) &&
            gc_old_space(gc) != visited) {
            gc_flip_old_space(gc);
            gc_list_move(gc, cf->container);
            cf->size++;
        }
    }
    return 0;
}
```

```
def test(n=2000):
    d = {}
    for i in range(n):
        for j in range(n):
            d[(i,j)] = i
```



РОСНЕФТЬ

СПАСИБО ЗА ВНИМАНИЕ!

По всем возникающим вопросам просьба обращаться к
Мирянову Сергею Николаевичу

по адресу электронной почты: snmirianov1@rn-t.ru