ТЕХНОСФЕРА

Лекция №1

Многопоточное программирование

Дмитрий Калугин-Балашов

Литература

Джеф Элджер. C++: Библиотека пограммиста

Jeff Alger. C++ for Real Programmers

Стандарты C++

- C++98/C++03
  - Boost
- C++11
- C++14

Контейнеры C++

- STL
- STL (C++11)
- Boost

Контейнеры STL

- Последовательные контейнеры
- Ассоциативные контейнеры
- Контейнеры-адаптеры
- Псевдоконтейнеры

Контейнеры STL

- **Последовательные контейнеры**
- **Ассоциативные контейнеры**
- Контейнеры-адаптеры
- Псевдоконтейнеры

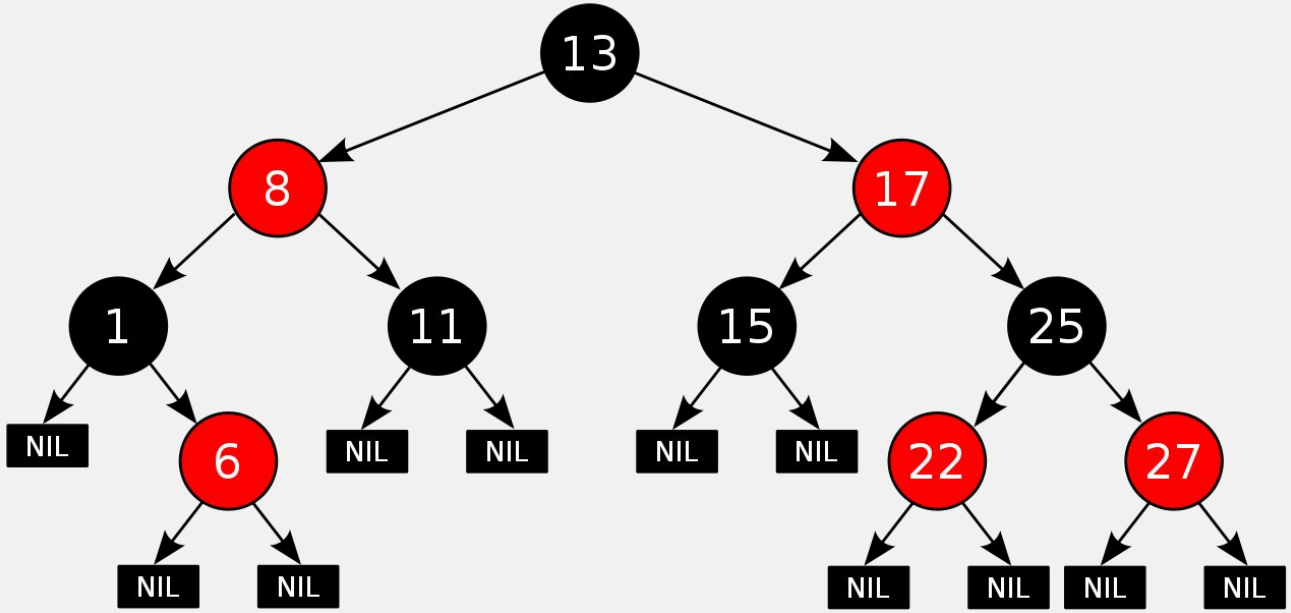
Последовательные контейнеры STL

- std::vector
- std::list
- std::deque

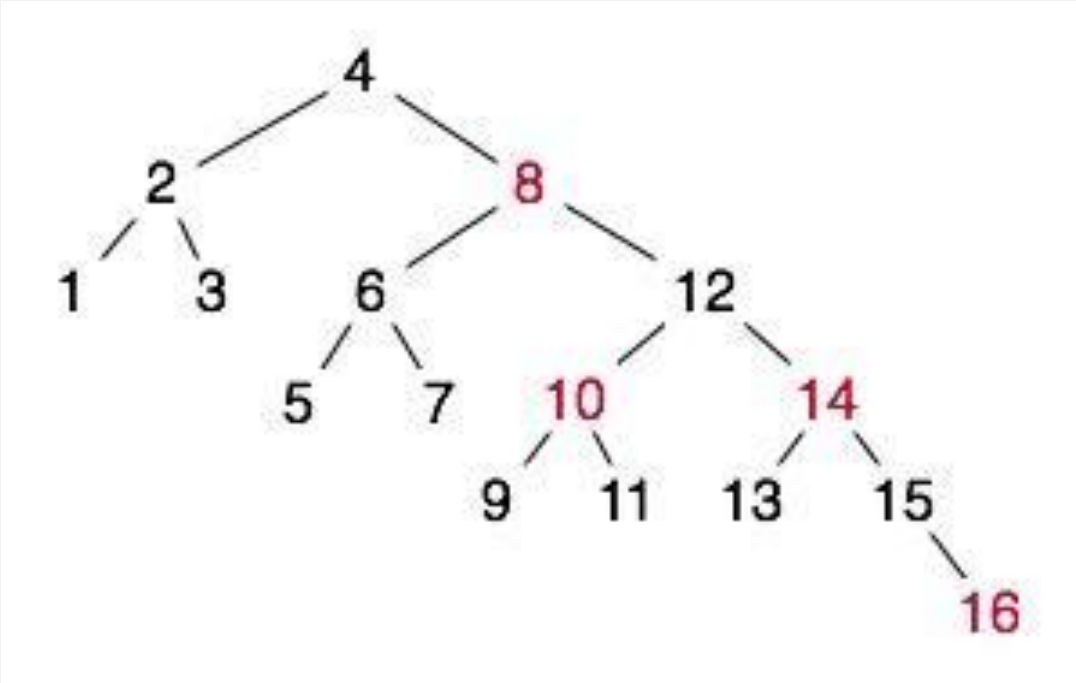
Ассоциативные контейнеры STL

- std::set
- std::map
- std::multiset
- std::multimap

Красно-черные деревья



Красно-черные деревья



Красно-черные деревья

http://www.youtube.com/v/vDHFF4wjWYU

B-деревья

https://code.google.com/p/cpp-btree/

B-деревья

- btree_set
- btree_map
- btree_multiset
- btree_multimap

Контейнеры-адаптеры STL

- std::stack
- std::queue
- std::priority_queue

Псевдоконтейнеры STL

- std::bitset
- std::basic_string
- std::valarray

Последовательные контейнеры STL (C++11)

- std::array
- std::forward_list

std::array vs. std::vector

- std::vector хранит все элементы в куче
- std::array хранит все элементы в себе

- std::array не может изменить свой размер
- std::array должен знать свой размер на этапе компиляции
- std::array работает быстрее

std::forward_list

Итератор может двигаться только в одном направлении.

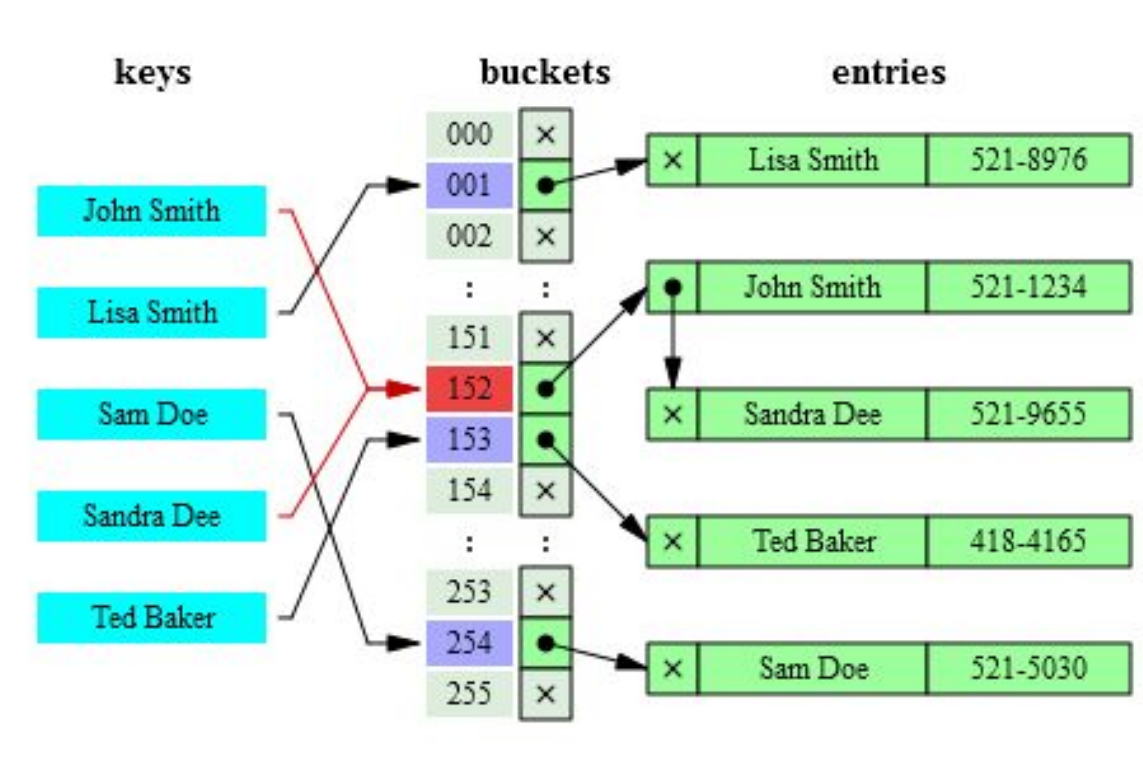
```
1.  #include <iostream>
2.  #include <forward_list>

1.  int main ()
2.  {
3.    std::forward_list<int> mylist = { 34, 77, 16, 2 };
4.    std::cout << "mylist contains:";
5.    for ( auto it = mylist.begin(); it != mylist.end(); ++it )
6.      std::cout << ' ' << *it;
7.    std::cout << '\n';
8.    return 0;
9.  }
```

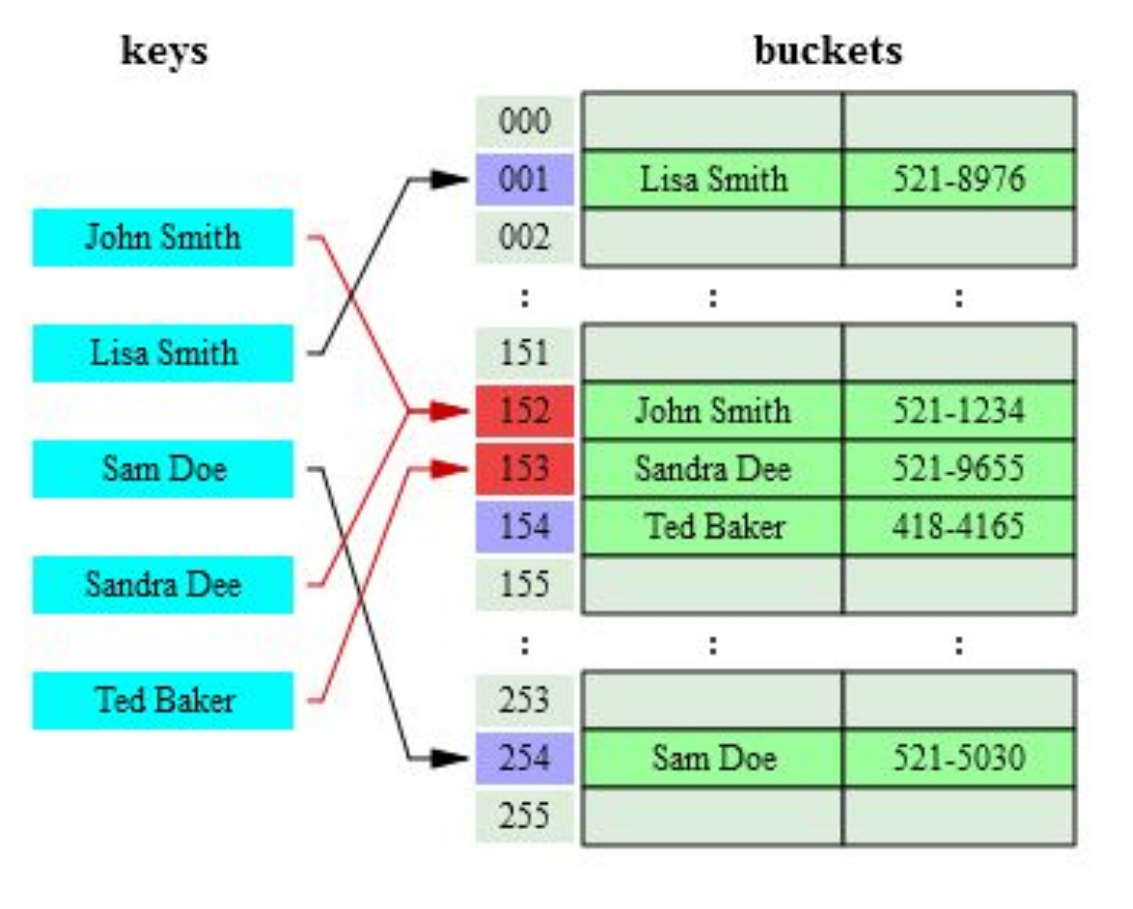
Хэш-таблицы STL (C++11)

- std::unordered_set
- std::unordered_map
- std::unordered_multiset
- std::unordered_multimap

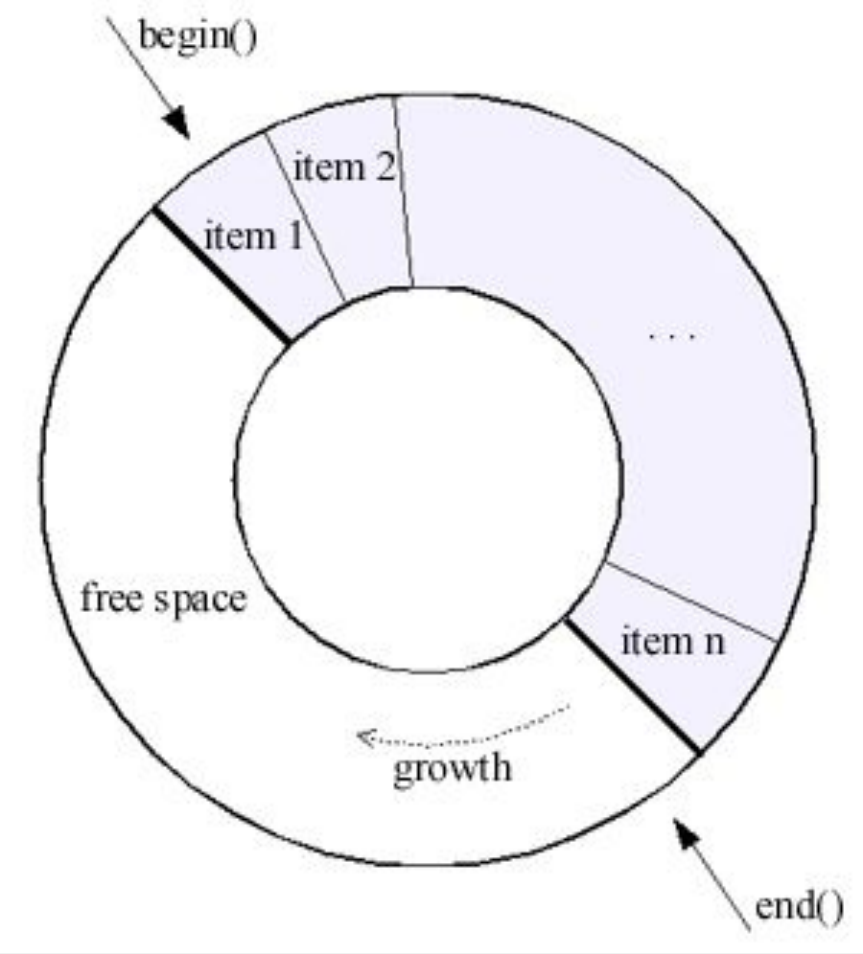
Хэш-таблицы STL (C++11)



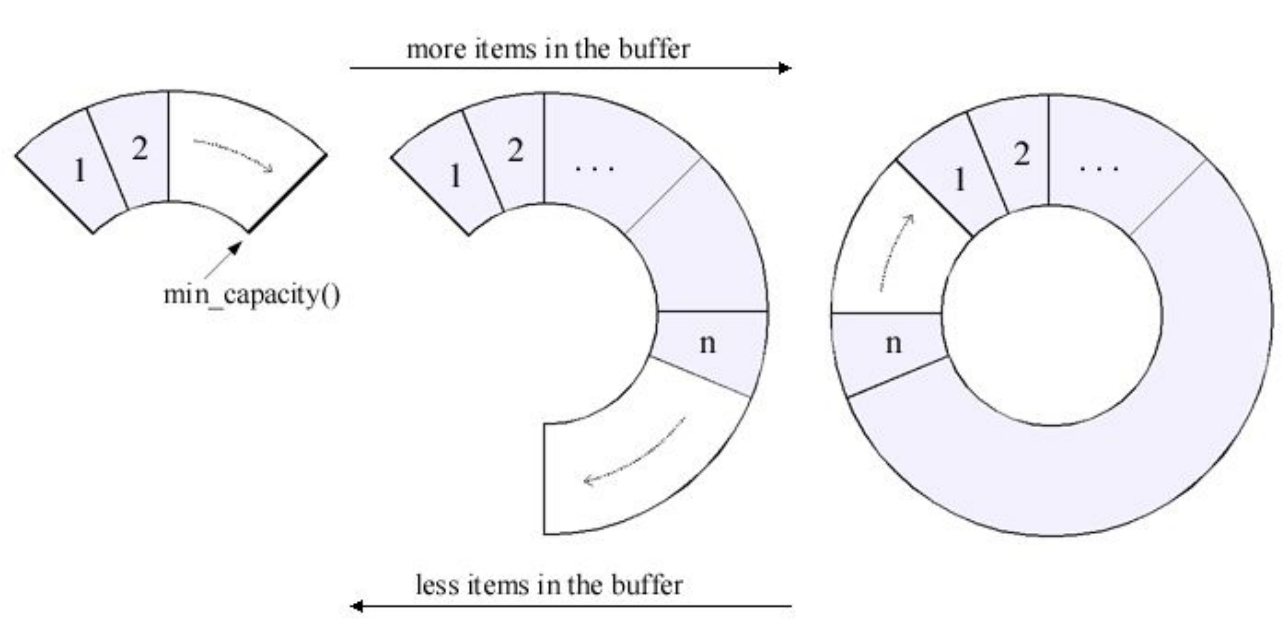
Хэш-таблицы STL (C++11)



boost::circular_buffer



boost::circular_buffer_space_optimized



Умные указатели

Умные указатели

Пример «самодельного» умного указателя.

```
1.  class PFoo {
2.  private:
3.      Foo* foo;
4.  public:
5.      PFoo() : foo(NULL) {}
6.      PFoo(Foo* f) : foo(f) {}
7.      operator Foo*() { return foo; }
8.      Foo* operator->() { return foo; }
9.  };

1.  void f(Foo*);
2.  PFoo pf(new Foo);
3.  f(pf);
4.  pf->MemberOfFoo();
```

Умные указатели

Пример «самодельного» умного указателя.

```
1.  template <class Type>
2.  class SP {
3.  private:
4.      Type* pointer;
5.  public:
6.      SP() : pointer(NULL) {}
7.      SP(Type* p) : pointer(p) {}
8.      operator Type*() { return pointer; }
9.      Type* operator->() { return pointer; }
10. };

1.  void f(Foo*);
2.  Ptr<Foo> pf(new Foo);
3.  f(pf);
4.  pf->MemberOfFoo();
```

std::auto_ptr (C++03)

std::auto_ptr (C++03)

Не использовать!

std::auto_ptr (C++03)

Пример некорректного использования std::auto_ptr

```
1.  #include <memory>
2.  int func()
3.  {
4.  std::auto_ptr<CFoo> PFoo1(new CFoo());
5.  std::auto_ptr<CFoo> PFoo2;
6.  PFoo2 = PFoo1;
7.  }
```

std::unique_ptr (C++11)

Невозможность скопировать std::unique_ptr

```
1.  #include <memory>
2.  int func()
3.  {
4.  std::unique_ptr<CFoo> PFoo1(new CFoo());
5.  std::unique_ptr<CFoo> PFoo2;
6.  PFoo2 = PFoo1; // Ошибка при компиляции
7.  }
```

std::unique_ptr (C++11)

Перемещение std::unique_ptr

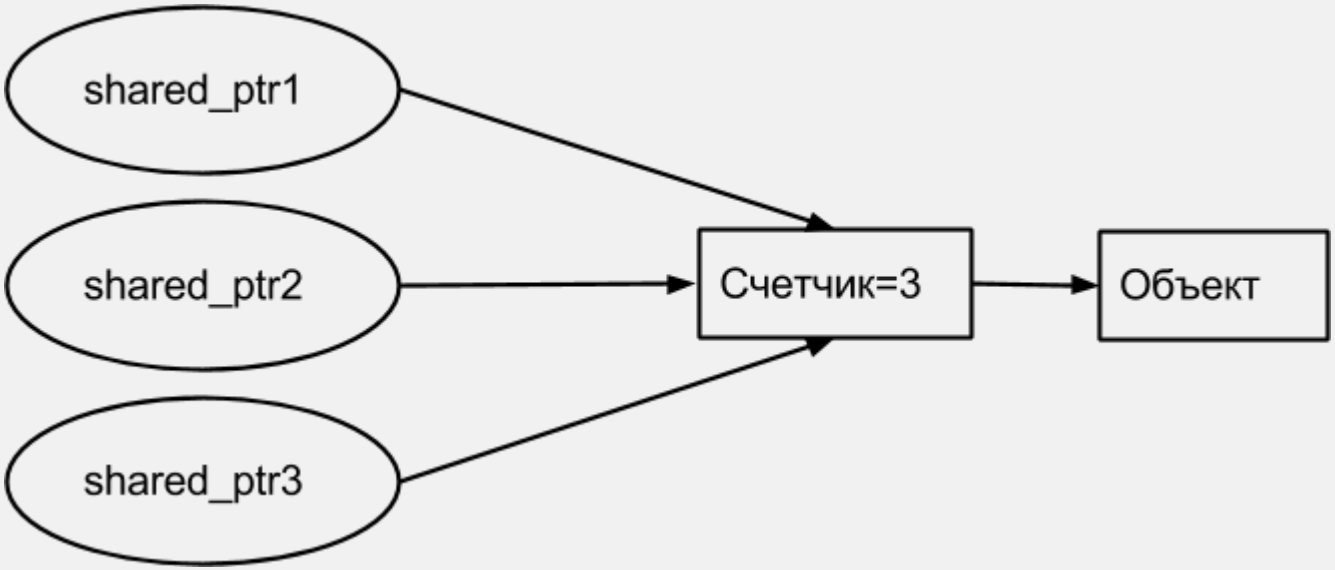
```
1.  #include <memory>
2.  int func()
3.  {
4.  std::unique_ptr<CFoo> PFoo1(new CFoo());
5.  std::unique_ptr<CFoo> PFoo2;
6.  PFoo2 = std::move(PFoo1);
7.  }
```

std::shared_ptr (C++11)

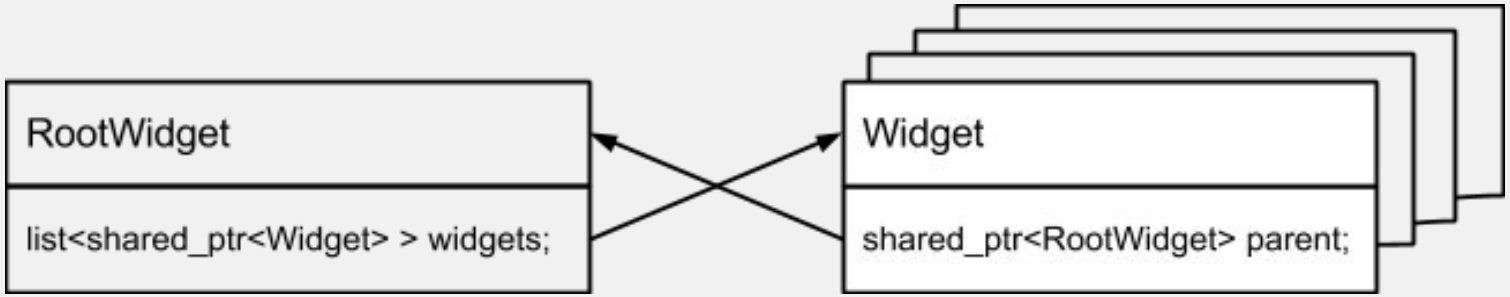
Пример использования std::shared_ptr

```
1.  #include <memory>
2.  int func()
3.  {
4.  std::shared_ptr<CFoo> PFoo1(new CFoo());
5.  std::shared_ptr<CFoo> PFoo2;
6.  PFoo2 = PFoo1;
7.  }
```

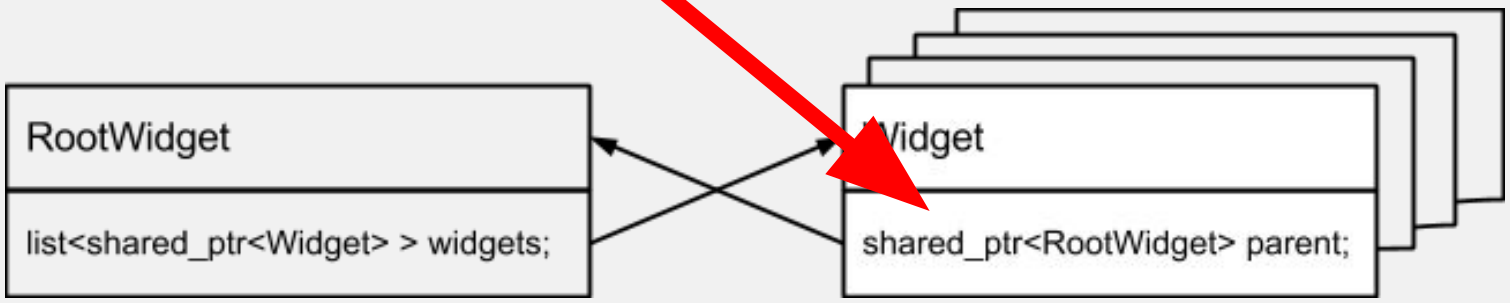
std::shared_ptr (C++11)



std::shared_ptr (C++11)

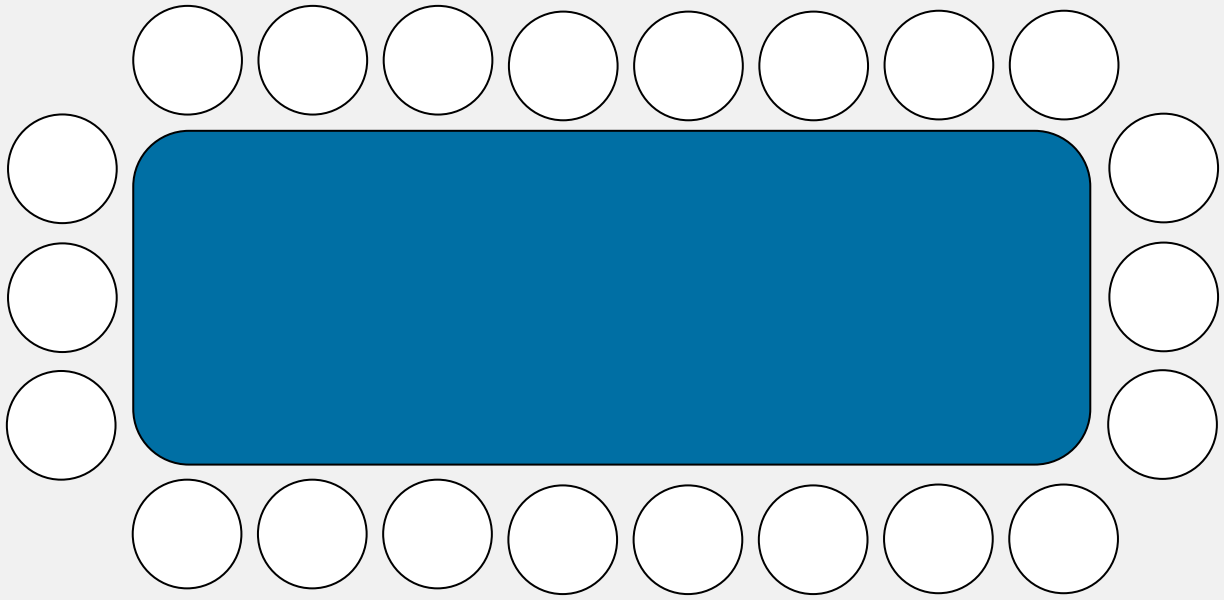


std::weak_ptr (C++11)

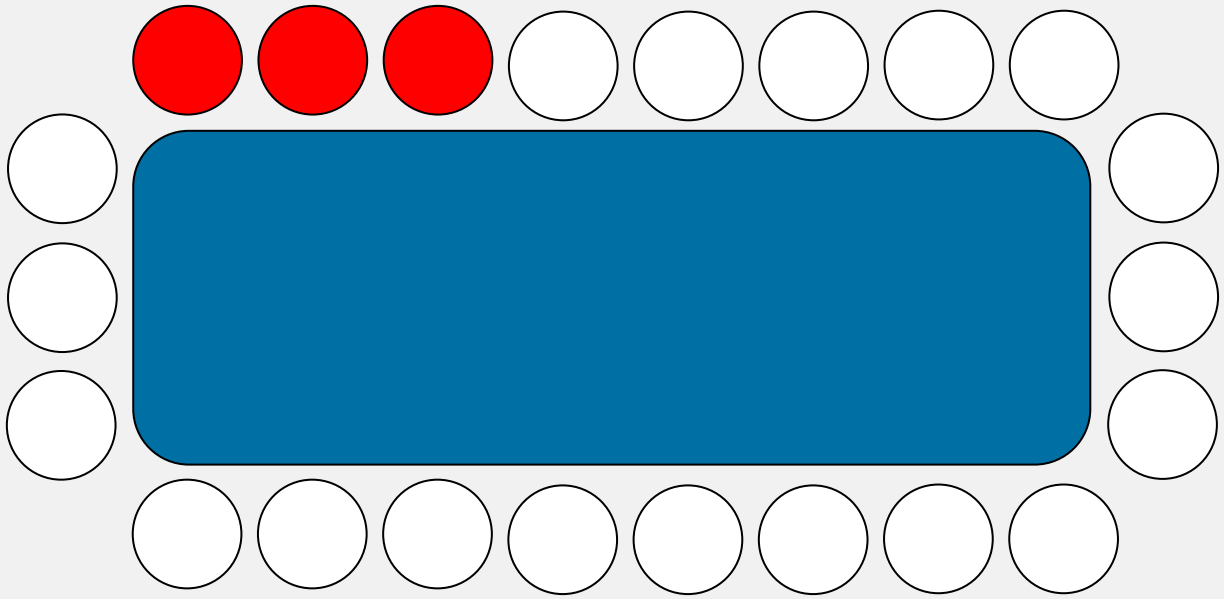


Аллокаторы

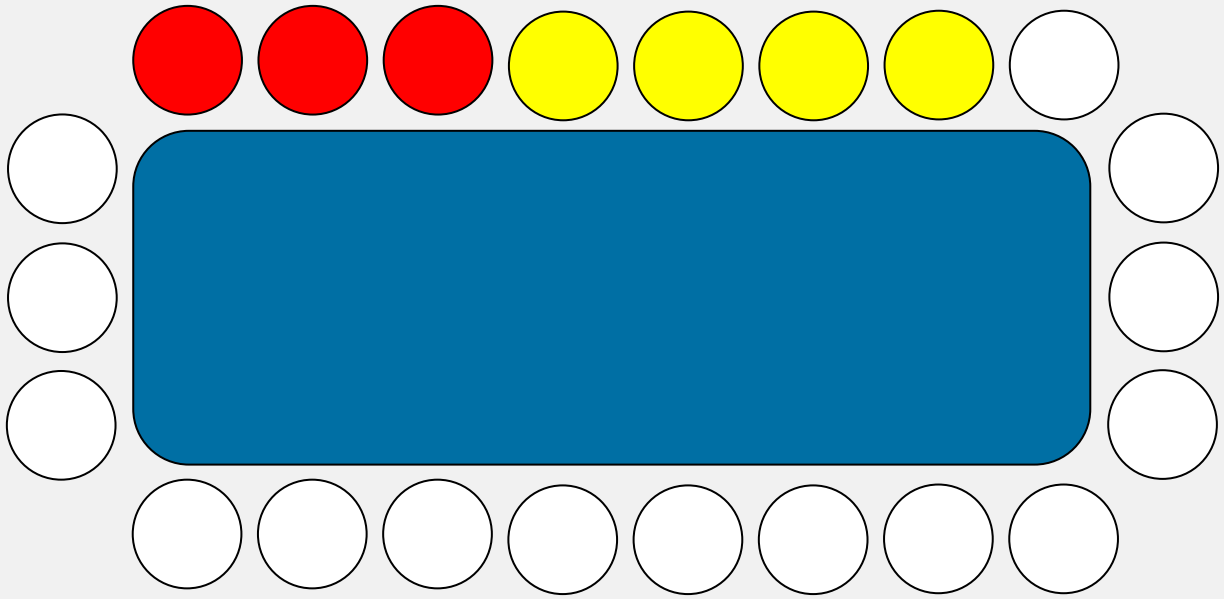
Аллокаторы



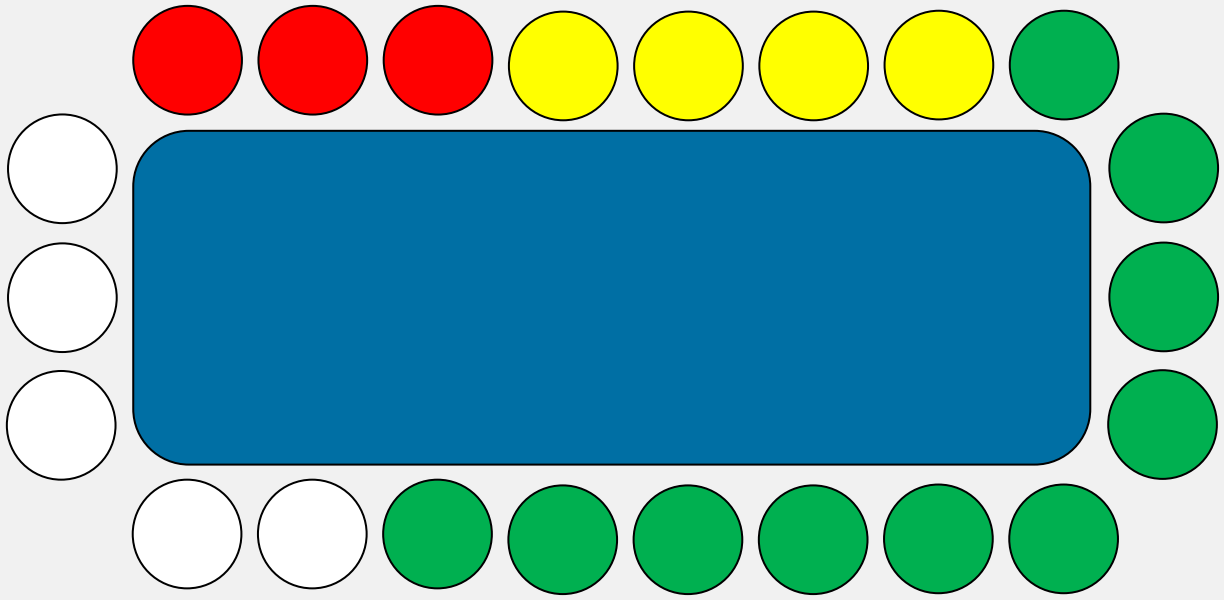
Аллокаторы



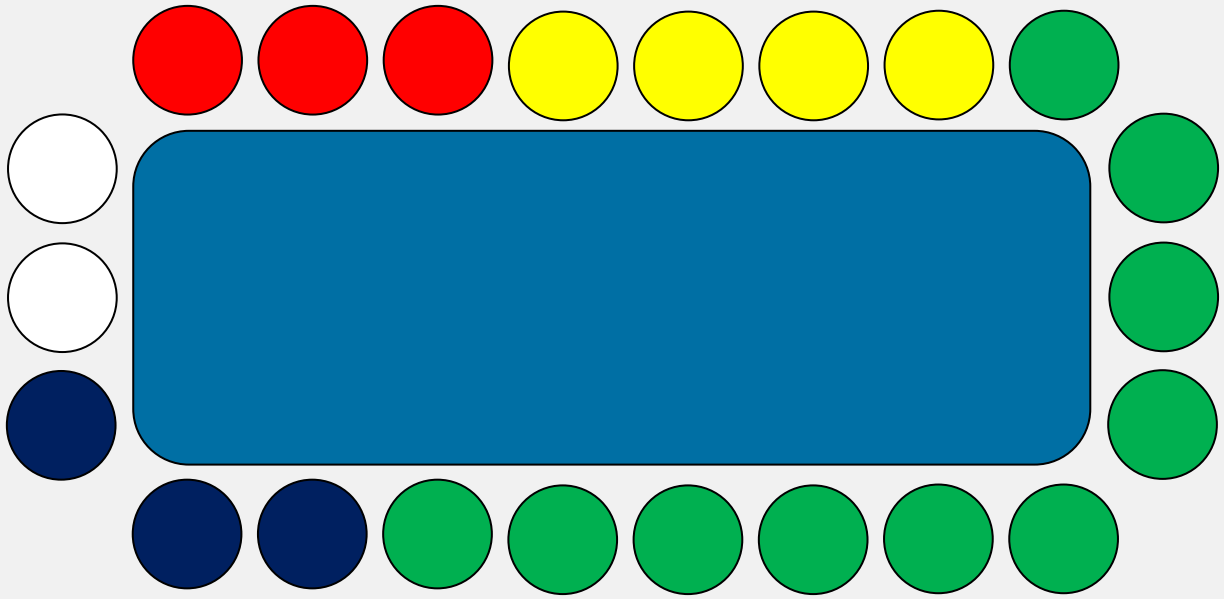
Аллокаторы



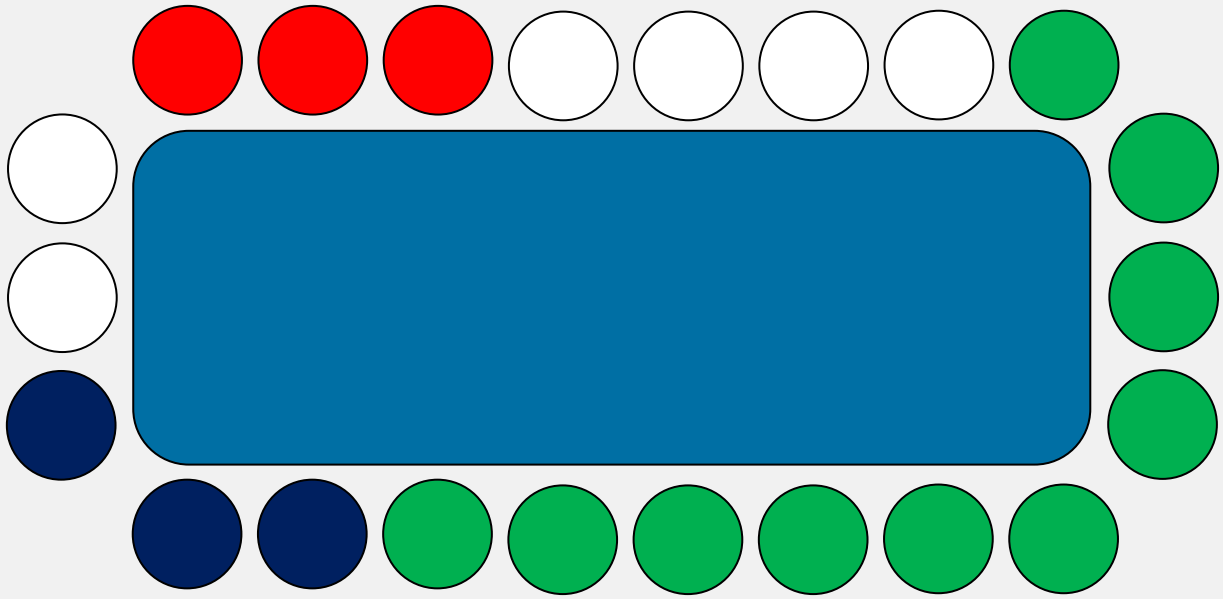
Аллокаторы



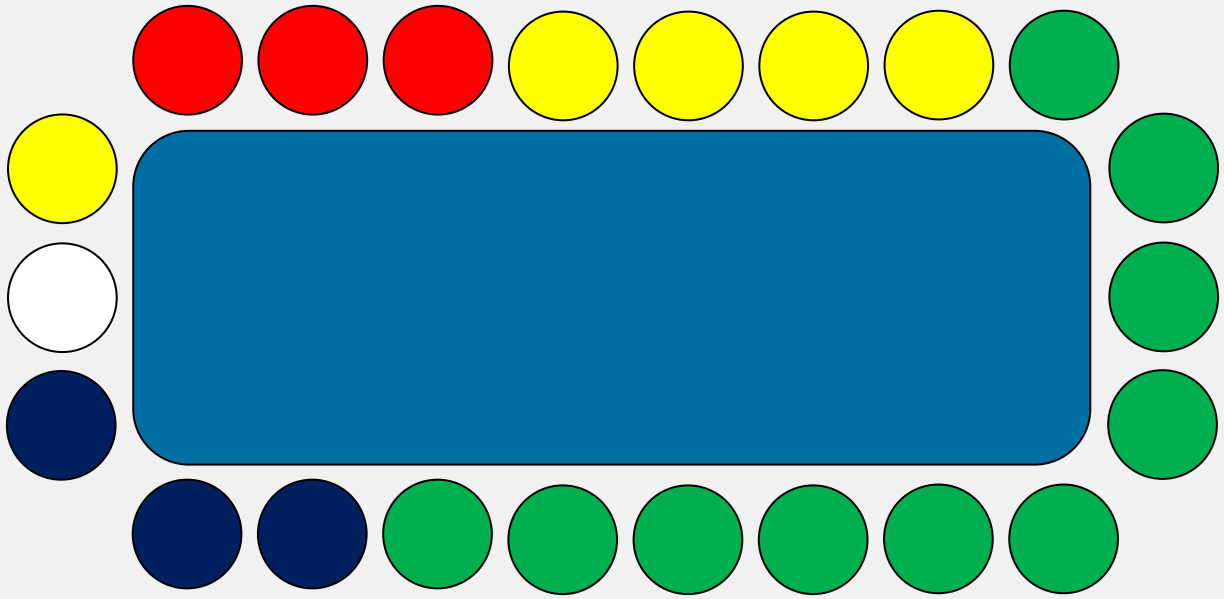
Аллокаторы



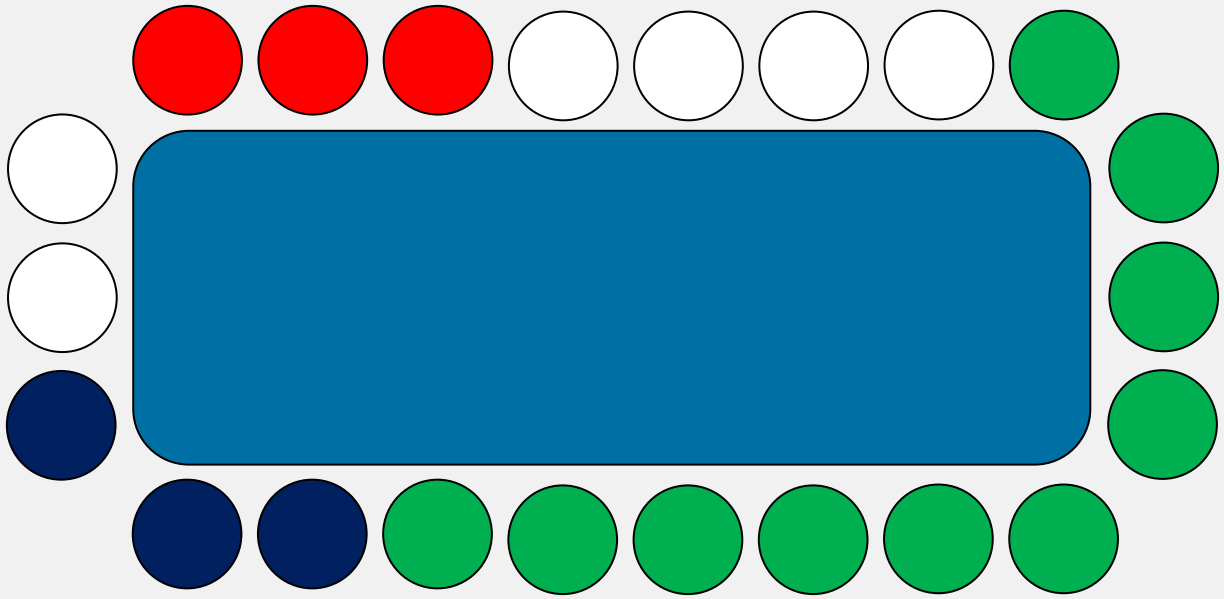
Аллокаторы



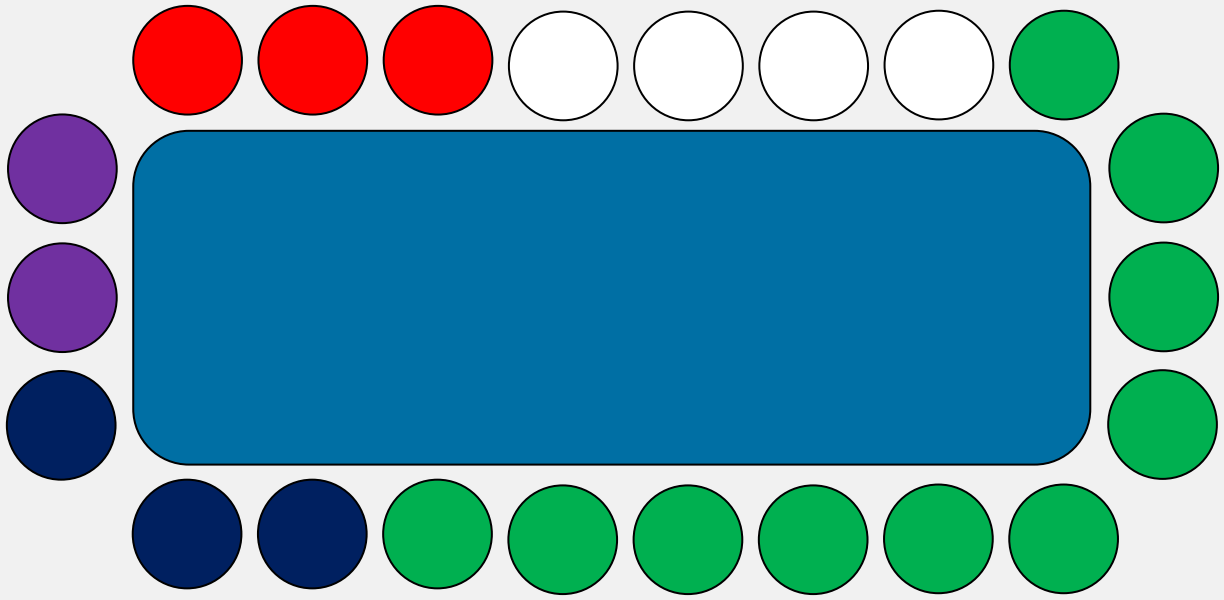
Аллокаторы



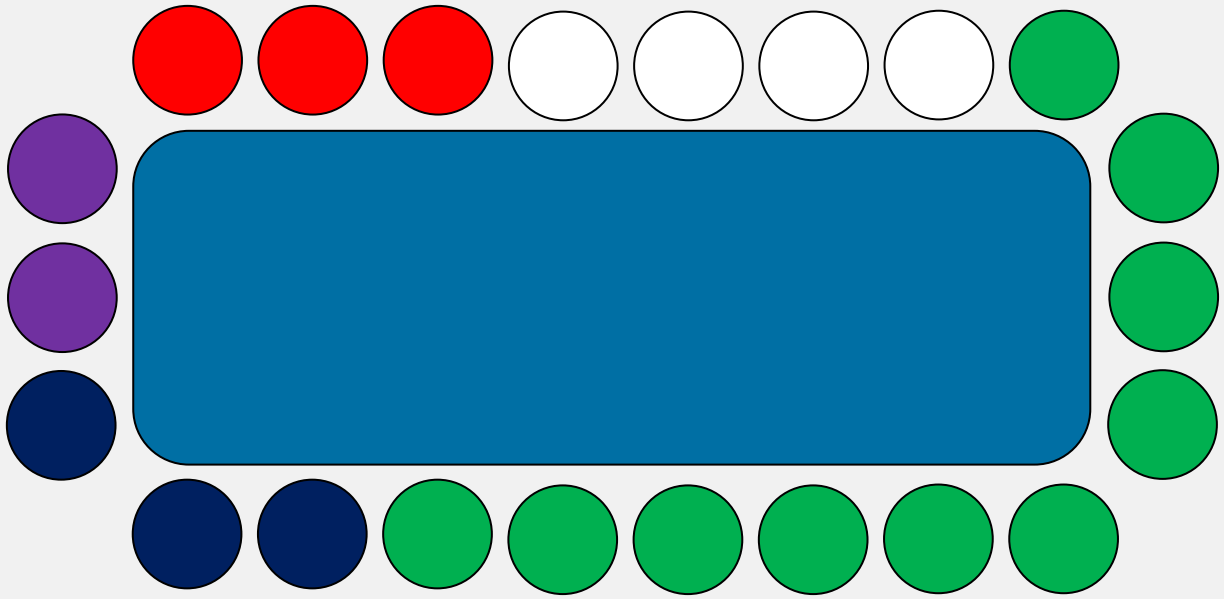
Аллокаторы



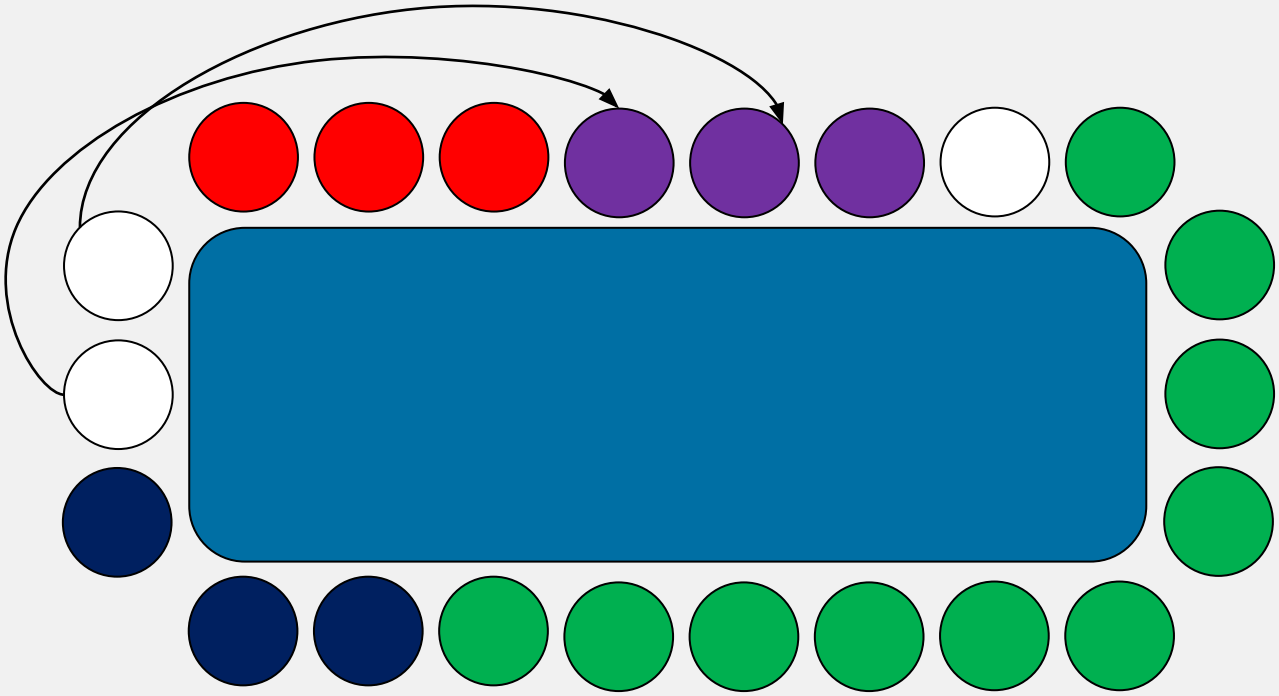
Аллокаторы



Аллокаторы



Аллокаторы



Аллокаторы

- malloc/calloc/realloc/free
- new/delete
- new[]/delete[]

Аллокаторы

- ccmalloc
- dmalloc
- tcmalloc

ccmalloc

Делаем утечки.

```
1.  void Leak(char *inStr)
2.  {
3.    char *str = (char *) malloc(strlen(inStr));
4.    memcpy(str, inStr, strlen(inStr));
5.  }

1.  char *AvoidLeak(char *inStr)
2.  {
3.    char *str = (char *) malloc(strlen(inStr));
4.    memcpy(str, inStr, strlen(inStr));
5.    return str;
6.  }
```

ccmalloc

Функция main с утечками.

```
1.  int main()
2.  {
3.    char *str;

1.    Leak("This leaks 19 bytes");
2.    str = AvoidLeak("This is not a 26 byte leak");
3.    free(str);
4.    str = AvoidLeak("12 byte leak");
5.    exit(0);
6.  }
```

ccmalloc

Результат ccmalloc (1).

```
1.   * 61.3% = 19 Bytes of garbage allocated in 1 allocation
2.   |        |
3.   |        |       0x40047306 in <???>
4.   |        |
5.   |        |       0x080493eb in <main>
6.   |        |                   at test1.c:20
7.   |        |
8.   |        |       0x0804935c in <Leak>
9.   |        |                   at test1.c:5
10.  |        |
11.  |        `-----> 0x08052fb7 in <malloc>
12.  |                            at src/wrapper.c:318
```

ccmalloc

Результат ccmalloc (2).

```
1.   |
2.   * 38.7% = 12 Bytes of garbage allocated in 1 allocation
3.   |       |
4.   |       |       0x40047306 in <???>
5.   |       |
6.   |       |       0x0804941e in <main>
7.   |       |                   at test1.c:23
8.   |       |
9.   |       |       0x080493a4 in <AvoidLeak>
10.  |       |                   at test1.c:11
11.  |       |
12.  |       `-----> 0x08052fb7 in <malloc>
13.  |                           at src/wrapper.c:318
14.  |
15.  `-------------------------------------------------
```

dmalloc

Результат dmalloc.

```
1.   not freed: '0x45008' (12 bytes) from 'ra=0x1f8f4'
2.   not freed: '0x45028' (12 bytes) from 'unknown'
3.   not freed: '0x45048' (10 bytes) from 'argv.c:1077'
4.     known memory not freed: 1 pointer, 10 bytes
5.   unknown memory not freed: 2 pointers, 24 bytes
```

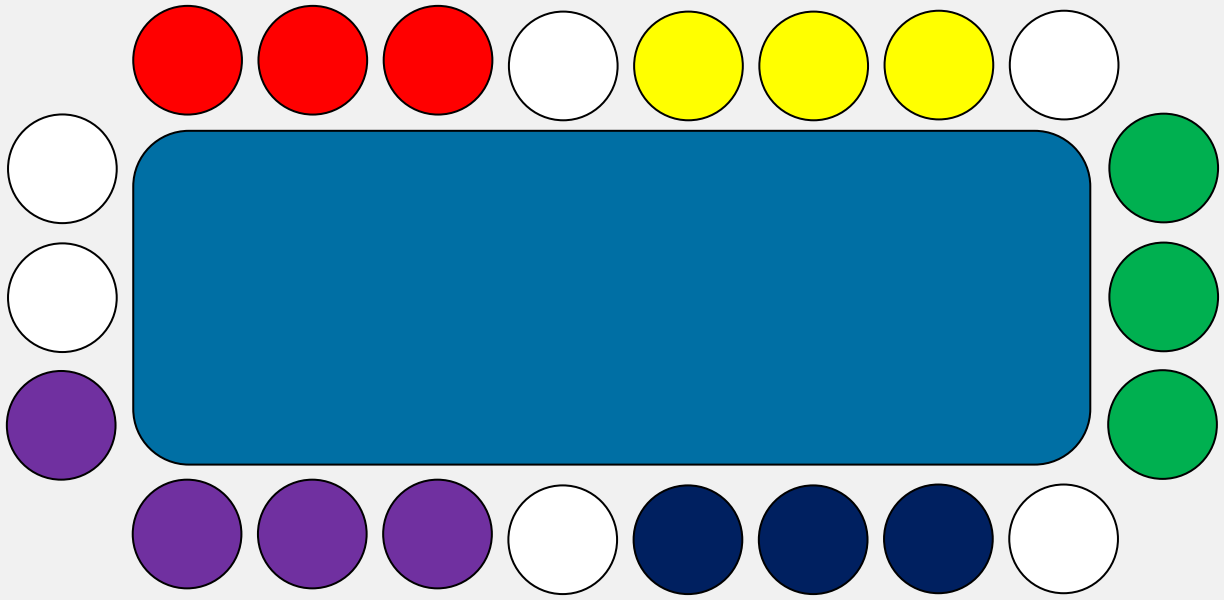
tcmalloc

Работает быстрее, чем malloc из glibc

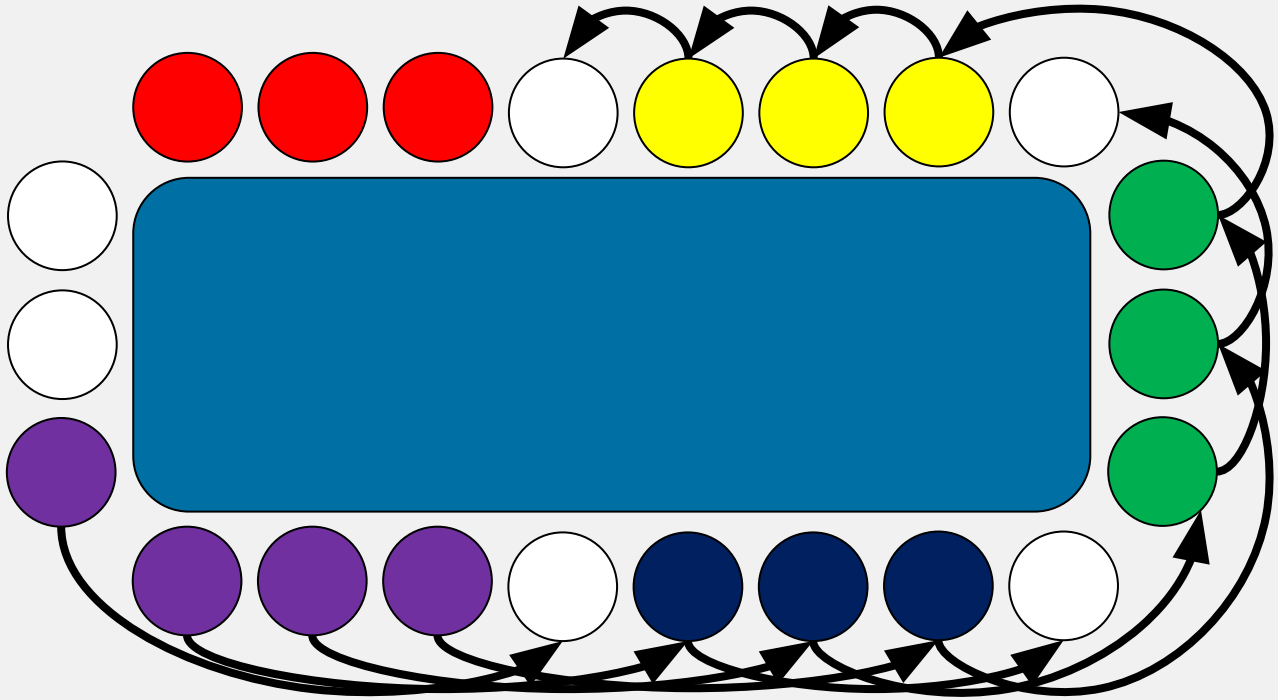
LD_PRELOAD="/usr/lib/libtcmalloc.so"

Уплотнение памяти

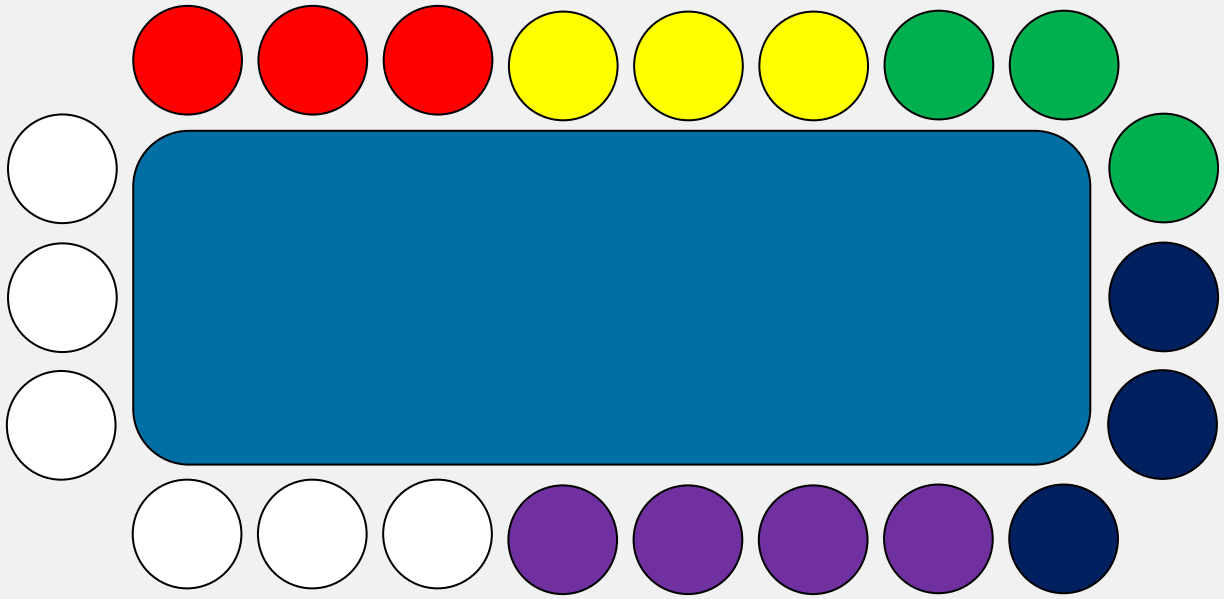
Уплотнение памяти



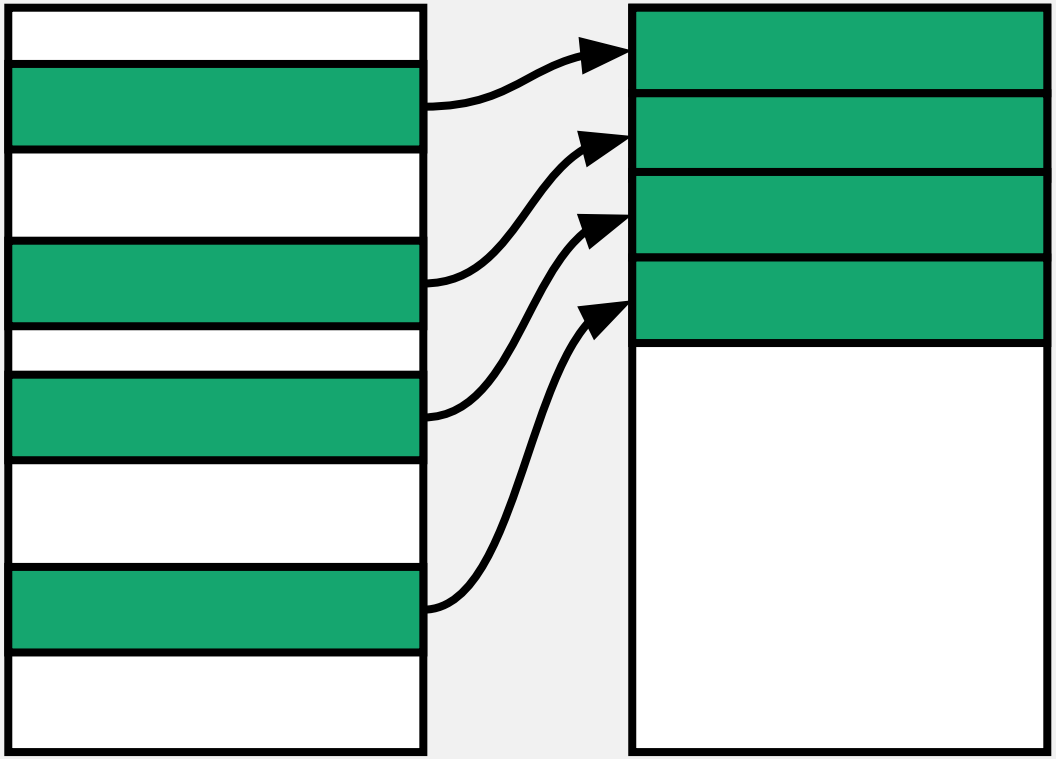
Уплотнение памяти



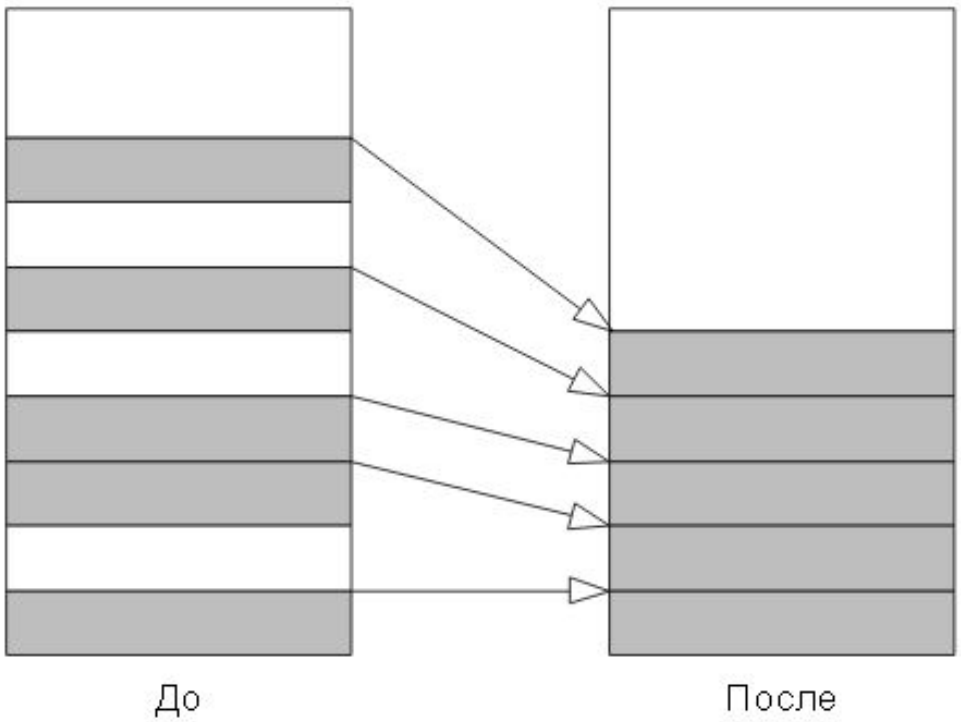
Уплотнение памяти



Алгоритм Бейкера



Уплотнение на месте



Разобраться самостоятельно

- git
- make

ТЕХНОСФЕРА

Спасибо за внимание!

Дмитрий Калугин-Балашов

rvncerr@rvncerr.org