



Указатели

ПЕРЕМЕННЫЙ РАЗМЕР МАССИВА

```
int n;  
cin >> n;  
int arr[n]; // ошибка
```

Взять заведомо большую длину массива:

```
int arr[10000], n;  
cin >> n;
```

Далее используем только первые n элементов

Динамическая
память
(вторая половины
семестра)

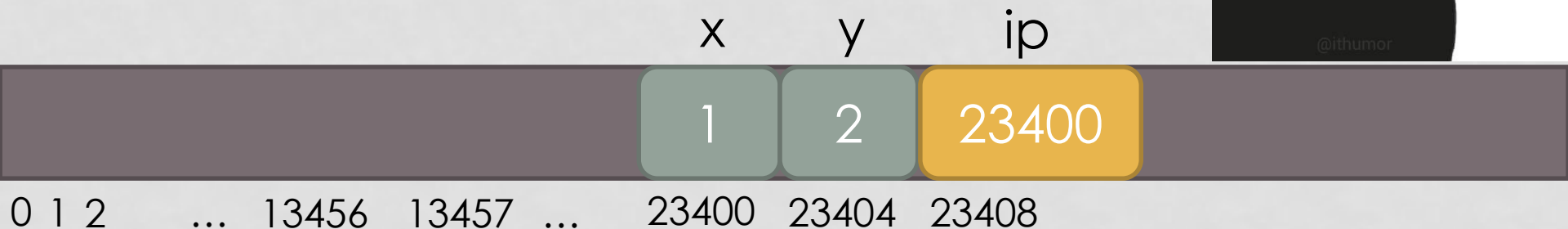
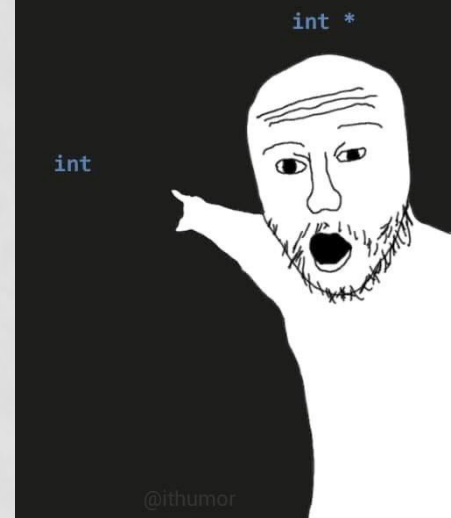
RAM - ЭТО МАССИВ БАЙТОВ

- Память - это массив байтов.
- Каждый байт имеет свой индекс в массиве памяти. Назовем индекс **адресом**.
- **RAM** - random-access memory, запоминающее устройство с произвольным доступом. В нашем случае – оперативная память.



.flat

МОДЕЛЬ ПАМЯТИ



```
int x = 1, y = 2;  
int *ip = &x;
```

АДРЕСА И УКАЗАТЕЛИ

```
int x = 1, y = 2;
```

```
int *ip;           // ip имеет тип int *, ip - указатель на int
ip = &x;           // & - операция взятия адреса, теперь в
                   // ip хранится адрес x; иначе говоря,
                   // ip указывает на x
```

```
y = *ip;           // * - операция разыменования
                   // указателя, теперь y == 1.
```

```
*ip = 0;           // x == 0;
```

- NULL – нулевой указатель, константа.
- NULL == 0

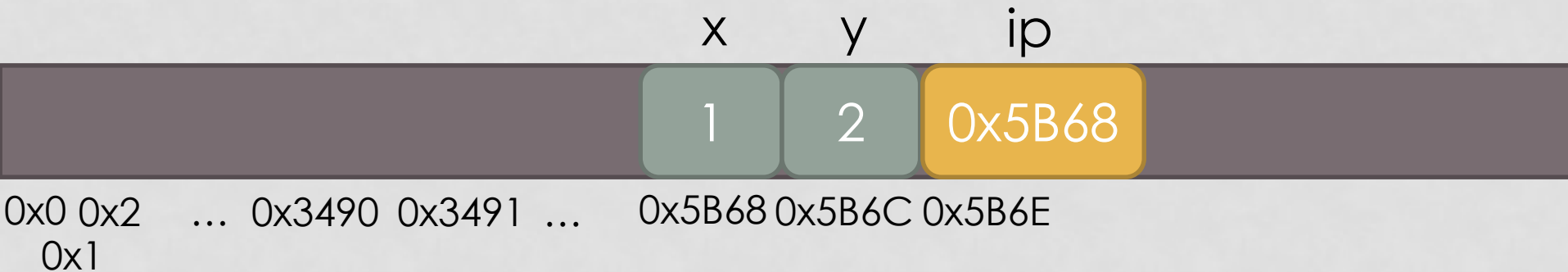


МОДЕЛЬ ПАМЯТИ

16ричная система счисления.

cout << &x; // выведет 0x5B68

.flat



УКАЗАТЕЛИ И МАССИВЫ

массив хранится одним куском памяти

```
int a[10];
```

```
int * p;
```

```
p = &(a[0]);
```

```
p = a;
```

// эти две строчки делают одно
// и то же: переменная a имеет
// тип int* и является указателем
// на нулевой элемент массива

```
*(a + 8) = 1;
```

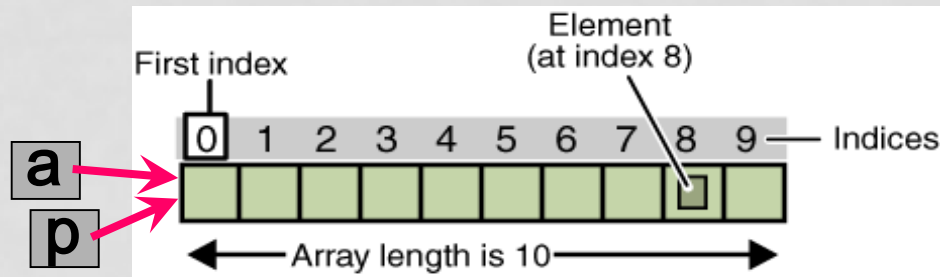
```
a[8] = 1;
```

```
*(p + 8) = 1;
```

```
p[8] = 1;
```

// эти четыре строчки тоже

a = p; //ошибка



АДРЕСНАЯ АРИФМЕТИКА

```
int a[10], x;
```

```
int * p;
```

```
p = a;
```

```
x = *(p + 3);
```

```
x = p[3];
```

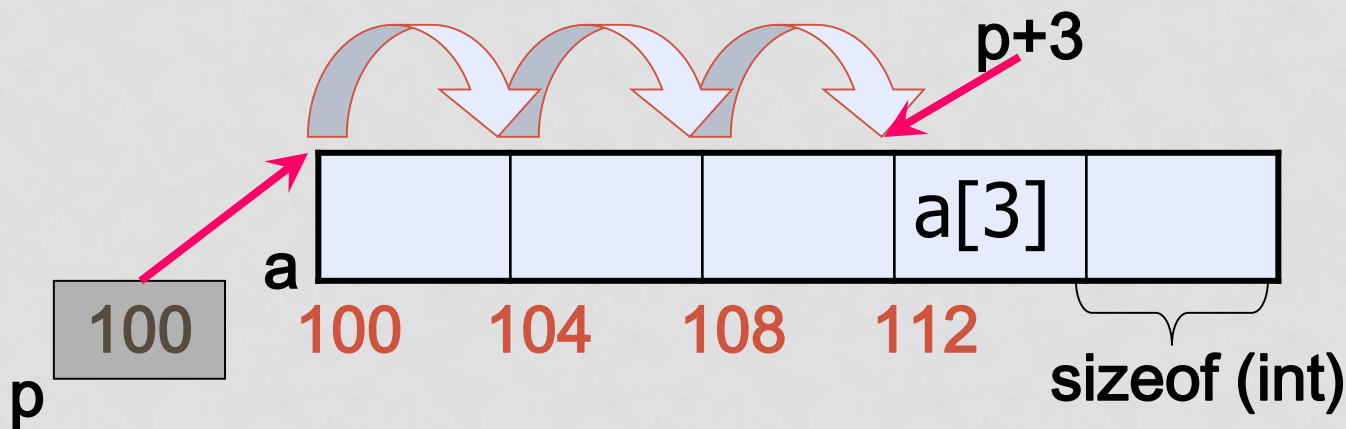
```
// хотим получить a[3]
```

```
// теперь x == a[3]: сдвиг на 3 ячейки,
```

```
// а не байта
```

```
// поэтому массивы нумеруются с нуля
```

```
// a[0] == *a == *(a+0)
```



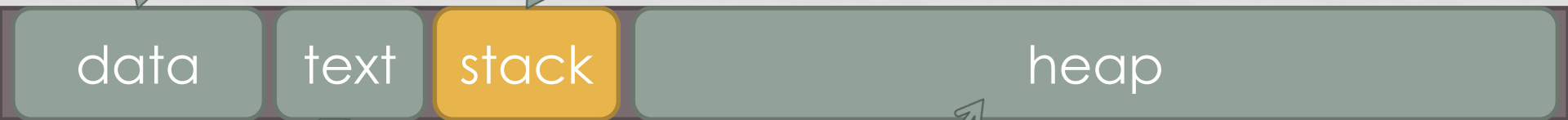
МОДЕЛЬ ПАМЯТИ

сегменты памяти

.flat

глобальные переменные

стек локальных переменных



data

text

stack

heap

текст программы
(машинный код)

куча

main

СТЕК ЛОКАЛЬНЫХ ПЕРЕМЕННЫХ



```
void g(char x, char b) {  
    float mas[100];  
}
```

```
void f() {  
    int a, b;  
    g('a', 48);  
}
```

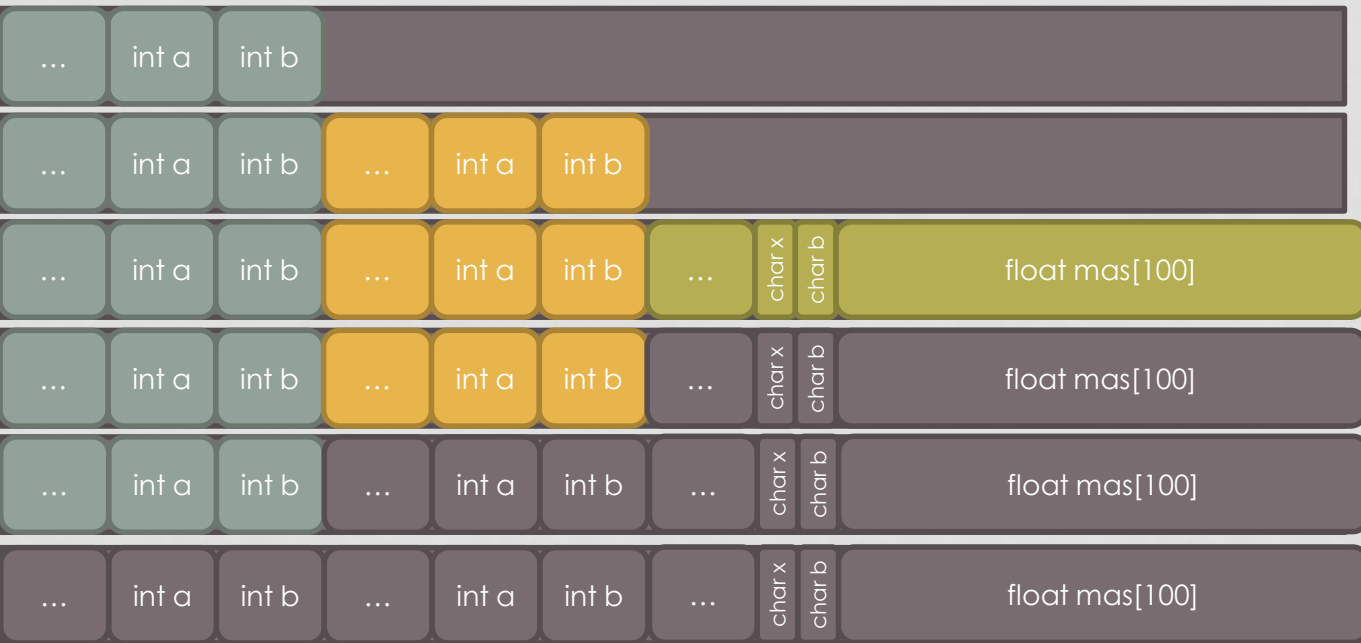
```
int main() {  
    int a, b;  
    f();  
    return 0;  
}
```



Пользователи, которым
нравится, что Snapchat
не сохраняет сообщения

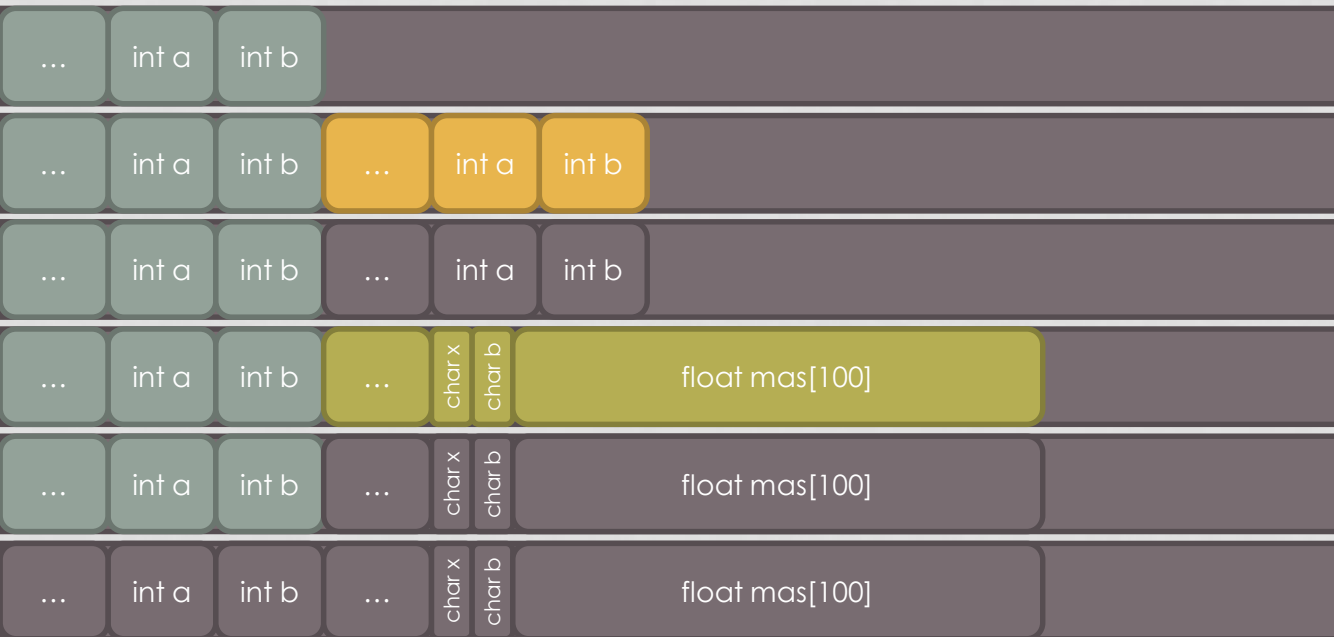
Разработчики,
которые забили
на сервер для хранения
сообщений.

СТЕК ЛОКАЛЬНЫХ ПЕРЕМЕННЫХ



```
void g(char x, char b) {  
    float mas[100];  
}  
  
void f() {  
    int a, b;  
    g('a', 48);  
}  
  
int main() {  
    int a, b;  
    f();  
    return 0;  
}
```

СТЕК ЛОКАЛЬНЫХ ПЕРЕМЕННЫХ



```
void f() {  
    int a, b;  
}  
  
void g(char x, char b) {  
    float mas[100];  
}  
  
int main() {  
    int a, b;  
    f();  
    g('a', 48);  
    return 0;  
}
```

don't touch my garbage!!!!



УКАЗАТЕЛИ И ФУНКЦИИ

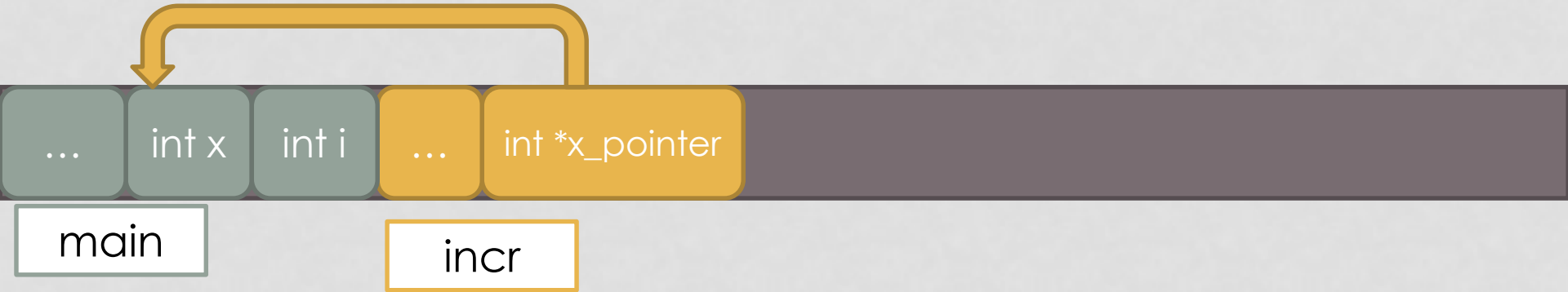
Как менять значение аргументов?

- Перестроить алгоритм
- Глобальные переменные
 - небезопасная передача данных
 - неэффективное использование памяти
 - усложняет повторное использование кода
- Передача аргумента по указателю

ПЕРЕДАЧА В ФУНКЦИЮ УКАЗАТЕЛЯ

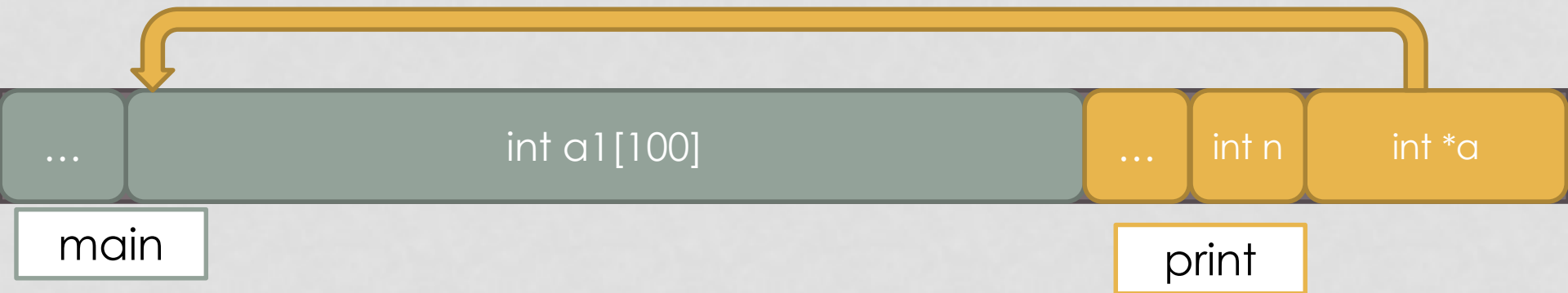
```
void incr (int * x_pointer) {  
    (*x_pointer) ++;  
}  
  
int main ( ) {  
    int x = 1;  
    for (int i = 0; i < 3; i++) {  
        incr(&x);  
        cout << "x = " << x << endl;  
    }  
    return 0;  
}
```

ПЕРЕДАЧА В ФУНКЦИЮ УКАЗАТЕЛЯ



```
void incr (int * x_pointer) {  
    (*x_pointer) ++;  
}  
  
int main ( ) {  
    int x = 1;  
    for (int i = 0; i < 3; i++) {  
        incr(&x);  
    }  
}
```

ПЕРЕДАЧА В ФУНКЦИЮ МАССИВА



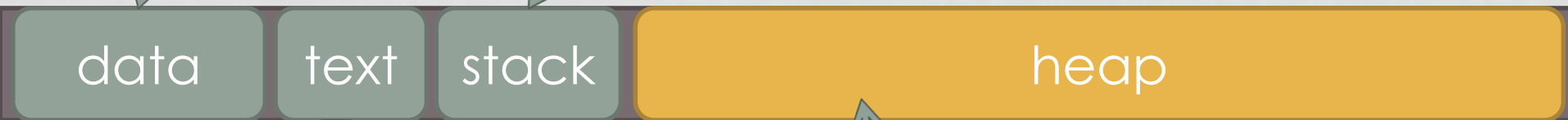
```
void print(int n, int *a) {  
    for (int i = 0; i < n; i++)  
        cout << a[i] << " ";  
    cout << endl;  
}  
...  
int a1[100];  
print(100, a1);
```


КАК ДОБЫТЬ КУСОЧЕК ПАМЯТИ

глобальные переменные

.flat

стек локальных переменных (~8Мб)



data

text

stack

heap

текст программы
(машинный код)

куча (~8-64Гб)

```
int *p = new int;  
delete p;
```

ПЕРЕМЕННЫЙ РАЗМЕР МАССИВА

```
unsigned int n;
```

```
cin >> n;
```

```
int *p = new int[n];
```

```
// выделили память размера n * sizeof(int) байт
```

```
//
```

```
// p - указатель на начало выделенной памяти
```

После этого обращаемся к элементам как в обычном массиве:

```
p[3] = 15; // то же самое: *(p+3) = 15;
```

```
delete[] p; // освобождаем память
```

```
// когда станет не нужной
```

```
// в конце работы программы
```

C++



ПЕРЕМЕННЫЙ РАЗМЕР МАССИВА



```
unsigned int n;  
scanf ("%u", &n);
```

```
int *p = (int *) malloc (n * sizeof (int) ); //stdlib.h
```

```
// выделили память размера n * sizeof(int) байт
```

```
// то же самое: p = calloc (n, sizeof(int) );
```

```
// p - указатель на начало выделенной памяти
```



После этого обращаемся к элементам как в обычном массиве:

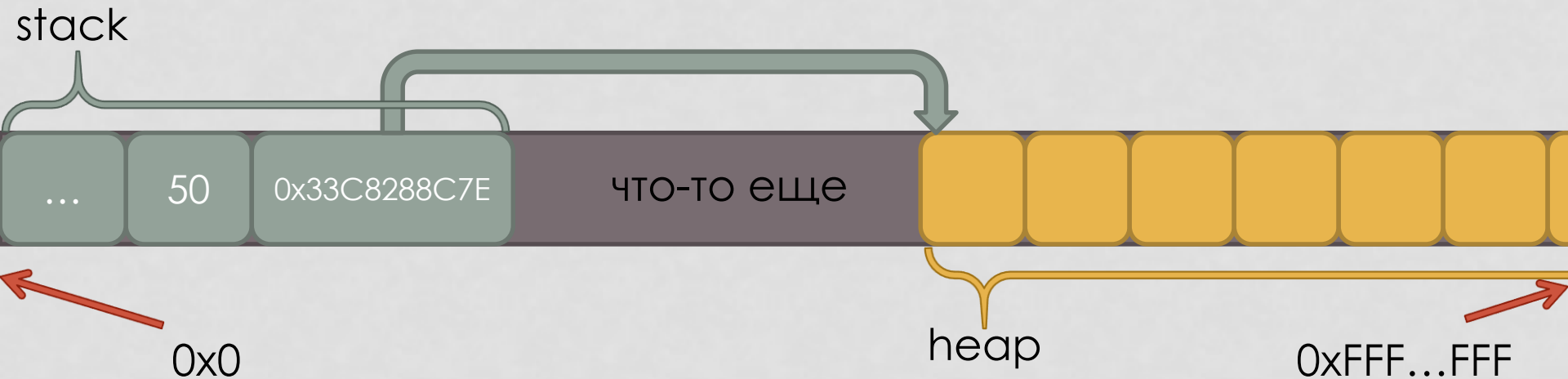
```
p [3] = 15; // то же самое: *(p+3) = 15;
```

```
free (p); // освобождаем память
```

```
// когда станет не нужной
```

```
// в конце работы программы
```

КАК ВЫГЛЯДЯТ УКАЗАТЕЛИ



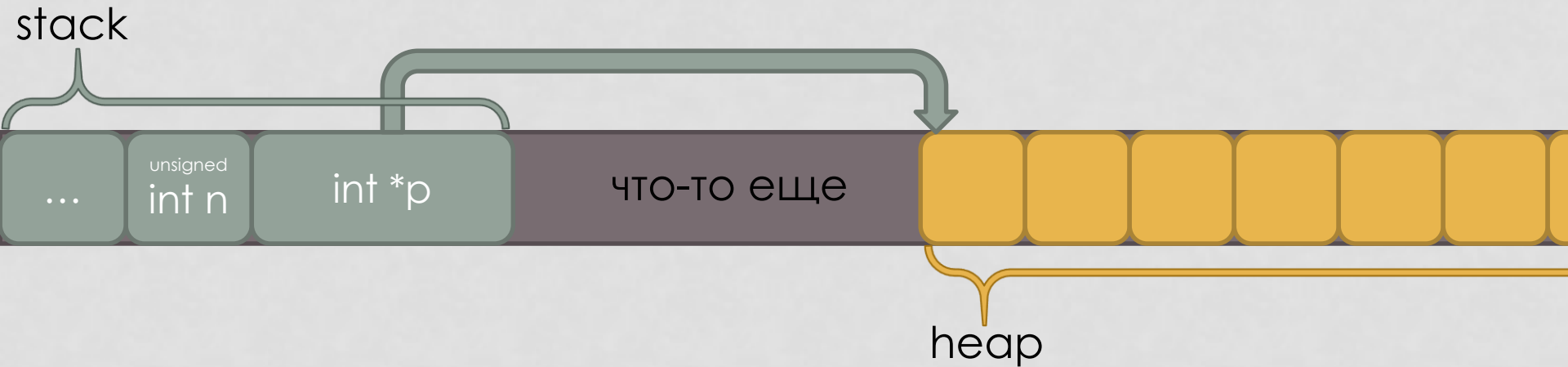
Размер указателя == битность системы.

x32 => 4 байта

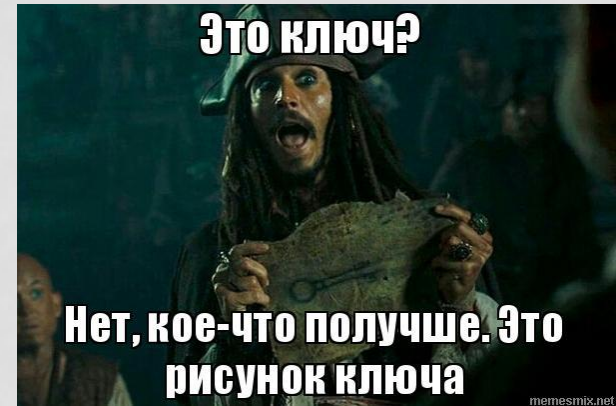
x64 => 8 байтов

`sizeof(int*) == sizeof(char*) == sizeof(void*)`

КАК «ВЫГЛЯДЯТ» УКАЗАТЕЛИ



```
int main() {  
    unsigned int n;  
    cin >> n;  
    int *p = new int[n];  
    ...  
}
```



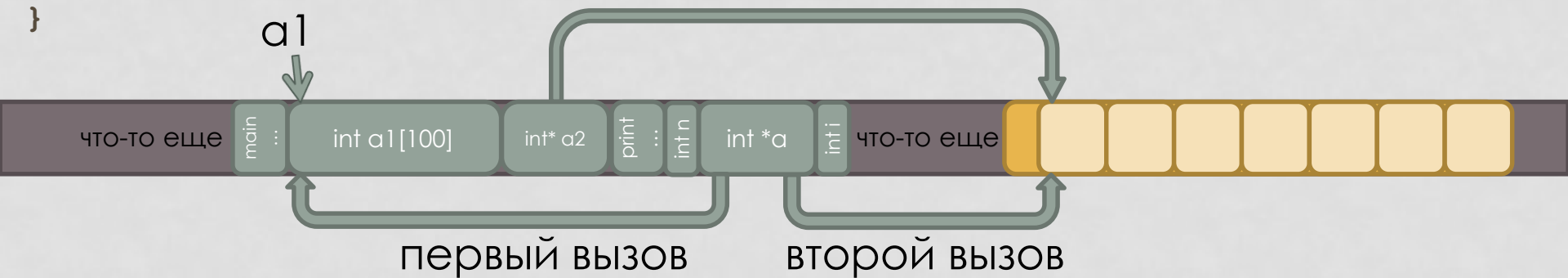
МАССИВЫ И ФУНКЦИИ

```
void print(int n, int *a) {  
    for (int i = 0; i < n; i++)  
        cout << a[i] << " ";  
    cout << endl;  
}
```

```
...  
int a1[100], *a2 = new int[100];  
print(100, a1);  
print(100, a2);
```

МАССИВЫ И ФУНКЦИИ

```
void print(int n, int *a) {  
    for (int i = 0; i < n; i++)  
        cout << a[i] << " ";  
    cout << endl;  
}
```



```
...  
int a1[100], *a2 = new int[100];  
print(100, a1);  
print(100, a2);
```

МАССИВЫ И ФУНКЦИИ

```
void print(int n, const int *a) {  
    for (int i = 0; i < n; i++)  
        cout << a[i] << " ";  
    cout << endl;  
}
```

...

```
int a1[100], *a2 = new int[100];  
print(100, a1);  
print(100, a2);
```


МАССИВЫ И ФУНКЦИИ

```
void scan(int n, int *a) {  
    for (int i = 0; i < n; i++)  
        cin >> a[i];  
}
```

```
...  
int a1[100], *a2 = new int[100];  
scan(100, a1);  
scan(100, a2);
```

МАССИВЫ И ФУНКЦИИ

```
void scan(int n, const int *a) {  
    for (int i = 0; i < n; i++)  
        cin >> a[i];  
}
```

```
...  
int a1[100], *a2 = new int[100];  
scan(100, a1);  
scan(100, a2);
```

ОПЕРАТОР СТРЕЛКА

```
struct Plot {  
    float data[1000];  
    unsigned int n;  
};
```

...

```
Plot *p1 = new Plot;  
p1->n = 100; // (*p1).n = 100;  
p1->data[314] = 42;  
delete p1;
```

МАССИВ В СТРУКТУРЕ

```
struct Array {  
    int *data, n;  
};
```

...

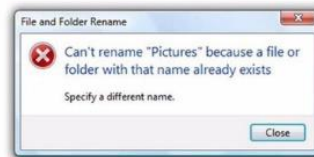
```
Array* a1 = new Array;  
cin >> a1->n;  
a1->data = new int[a1->n];  
delete[] a1->data;  
delete a1;
```

VALGRIND И ОСНОВНЫЕ ОШИБКИ

valgrind – инструмент
поиска ошибок памяти

- утечка памяти (memory leak)
- ошибка сегментации (segmentation fault)
- неинициализированные данные (uninitialized value)

ОШИБКИ В ПРОГРАММИРОВАНИИ



Обычная ошибка



Ошибка
сегментирования



Ошибка при работе
с ядром

Смотришь мемы,
чтобы не ботать

Твоя ошибка

UNINITIALIZED VALUE

==24547== Conditional jump or move depends on uninitialised value(s)

==24547== at 0x33C8288C7E: std::ostreambuf_iterator<char, std::char_traits<char> > std::
std::ostreambuf_iterator<char, std::char_traits<char> > >::_M_insert_int<long>(std::ostreambuf_iterator<char,
std::char_traits<char> >, std::ios_base&, char, long) const (in /usr/lib64/libstdc++.so.6.0.19)

==24547== by 0x33C828925C: std::num_put<char, std::ostreambuf_iterator<char, std::char_traits<char> >
>::do_put(std::ostreambuf_iterator<char, std::char_traits<char> >, std::ios_base&, char, long) const (in
/usr/lib64/libstdc++.so.6.0.19)

==24547== by 0x33C829502D: std::ostream& std::ostream::_M_insert<long>(long) (in /usr/lib64/libstdc++.so.6.0.19)

==24547== by 0x4016AA: main (002686.cpp:25)

==24547== Uninitialised value was created by a stack allocation

==24547== at 0x4015D5: main (002686.cpp:7)

==24547==

==24547== Use of uninitialised value of size 8

==24547== at 0x33C8288B63: ??? (in /usr/lib64/libstdc++.so.6.0.19)

==24547== by 0x33C8288CA5: std::ostreambuf_iterator<char, std::char_traits<char> > std::num_put<char,
std::ostreambuf_iterator<char, std::char_traits<char> > >::_M_insert_int<long>(std::ostr
std::char_traits<char> >, std::ios_base&, char, long) const (in /usr/lib64/libstdc++.so.
std::char_traits<char> >, std::ios_base&, char, long) const (in /usr/lib64/libstdc++.so.6.0.19)

==24547== by 0x33C828925C: std::num_put<char, std::ostreambuf_iterator<char, std::cha
>::do_put(std::ostreambuf_iterator<char, std::char_traits<char> >, std::ios_base&, char,
/usr/lib64/libstdc++.so.6.0.19)

...




MEMORY LEAK

==19472== 40 bytes in 1 blocks are definitely lost in loss record 1 of 1

==19472== at 0x4A06F70: operator new[](unsigned long) (in
/usr/lib64/valgrind/vgpreload_memcheck-amd64-linux.so)

==19472== by 0x4011DD: main (004848.cpp:3)

```
int main()
{
    int *p = new int[10];
    return 0;
}
```




==19829== 120 (80 direct, 40 indirect) bytes in 1 blocks are definitely lost in
loss record 2 of 2

==19829== at 0x4A06F70: operator new[](unsigned long) (in
/usr/lib64/valgrind/vgpreload_memcheck-amd64-linux.so)


==19829== by 0x4011DA: main (004849.cpp:3)

```
int main()
{
    int **p = new int*[10];
    p[0] = new int[10];
    return 0;
}
```



SEGMENTATION FAULT

```
==20172== Invalid write of size 4
==20172==    at 0x4011DE: main (004853.cpp:4)
==20172==   Address 0x4e34068 is 0 bytes after a block of size 40 alloc'd
==20172==    at 0x4A06F70: operator new[](unsigned long) (in /usr/lib64/valgrind/vgpreload_memcheck-amd64-linux.so)
==20172==    by 0x4011DD: main (004853.cpp:3)
==20172==
==20172== 40 bytes in 1 blocks are definitely lost in loss record 1 of 1
==20172==    at 0x4A06F70: operator new[](unsigned long) (in /usr/lib64/valgrind/vgpreload_memcheck-amd64-linux.so)
==20172==    by 0x4011DD: main (004853.cpp:3)
```



```
int main()
{
    int *p = new int[10];
    p[10] = 5; // p[-1000] ошибок не выдает)))
    return 0;
}
```