Анатомия Address Sanitizer

Алексей Веселовский

Align Technology

Что будет

1. Sanitizer – что это? Краткий обзор видов.
2. Основная идея реализации ASan
3. Код ASan – что где лежит
4. Поддержка ASan в компиляторе
5. Runtime поддержка ASan.
6. Знакомим ASan со своим аллокатором
7. Пример из жизни, как ASan встретил C++

Что будет

1. Sanitizer – что это? Краткий обзор видов.
2. Основная идея реализации ASan
3. Код ASan – что где лежит
4. Поддержка ASan в компиляторе
5. Runtime поддержка ASan.
6. Знакомим ASan со своим аллокатором
7. Пример из жизни, как ASan встретил C++

Sanitizer это…

Sanitizer это…

Динамический анализатор кода

Sanitizer это...

Динамический анализатор кода
Runtime проверки для различных типов UB

Sanitizer это…

Динамический анализатор кода
Runtime проверки для различных типов UB

Высокая точность

Sanitizer это...

Динамический анализатор кода
Runtime проверки для различных типов UB

Высокая точность
Минимальный шанс false positive

Sanitizer это...

Динамический анализатор кода
Runtime проверки для различных типов UB

Высокая точность
Минимальный шанс false positive
Накладные расходы во время исполнения программы

Sanitizer это…

Динамический анализатор кода
Runtime проверки для различных типов UB

Высокая точность
Минимальный шанс false positive
Накладные расходы во время исполнения программы
Накладные расходы в compile time (но не всегда заметные)

Sanitizer это…

Make undefined behavior defined again*

* не всегда

Sanitizer это…

Скорость достаточная для полноценного тестирования

Sanitizers - какие бывают

clang

gcc

Sanitizers - какие бывают

clang

Address Sanitizer (ASan) +
Leak Sanitizer (LSan)

gcc

Address Sanitizer (ASan) +
Leak Sanitizer (LSan)

Sanitizers - какие бывают

clang

Address Sanitizer (ASan) +
Leak Sanitizer (LSan)

Thread Sanitizer (TSan)

gcc

Address Sanitizer (ASan) +
Leak Sanitizer (LSan)

Thread Sanitizer (TSan)

Sanitizers - какие бывают

clang

Address Sanitizer (ASan) +
Leak Sanitizer (LSan)

Thread Sanitizer (TSan)

Undefined Behavior Sanitizer (UBSan)

gcc

Address Sanitizer (ASan) +
Leak Sanitizer (LSan)

Thread Sanitizer (TSan)

Undefined Behavior Sanitizer (UBSan)

Sanitizers - какие бывают

clang

Address Sanitizer (ASan) +
Leak Sanitizer (LSan)

Thread Sanitizer (TSan)

Undefined Behavior Sanitizer (UBSan)

Memory Sanitizer (MSan)

gcc

Address Sanitizer (ASan) +
Leak Sanitizer (LSan)

Thread Sanitizer (TSan)

Undefined Behavior Sanitizer (UBSan)

Sanitizers - какие бывают

clang

Address Sanitizer (ASan) +
Leak Sanitizer (LSan)

Thread Sanitizer (TSan)

Undefined Behavior Sanitizer (UBSan)

Memory Sanitizer (MSan)

Data Flow Sanitizer

gcc

Address Sanitizer (ASan) +
Leak Sanitizer (LSan)

Thread Sanitizer (TSan)

Undefined Behavior Sanitizer (UBSan)

Sanitizers - какие бывают

clang

Address Sanitizer (ASan) +
Leak Sanitizer (LSan)

Thread Sanitizer (TSan)

Undefined Behavior Sanitizer (UBSan)

Memory Sanitizer (MSan)

Data Flow Sanitizer

gcc

Address Sanitizer (ASan) +
Leak Sanitizer (LSan)

Thread Sanitizer (TSan)

Undefined Behavior Sanitizer (UBSan)

Sanitizers - какие бывают

MSVS 2019 Version 16.7.0 Preview 3.1:

Address Sanitizer

Sanitizers - какие бывают

MSVS 2019 Version 16.7.0 Preview 3.1:

Address Sanitizer
Нет Leak Sanitizer (его в принципе под Windows нет)

Sanitizers - какие бывают

MSVS 2019 Version 16.7.0 Preview 3.1:

Address Sanitizer
Нет Leak Sanitizer (его в принципе под Windows нет)
Теперь есть под x86_64

Sanitizers - какие бывают

MSVS 2019 Version 16.7.0 Preview 3.1:

Address Sanitizer
Нет Leak Sanitizer (его в принципе под Windows нет)
Теперь есть под x86_64. Работает пока плохо (с MFC не работает совсем, с консольным hello world можно попытаться).

Sanitizers - какие бывают

MSVS 2019 Version 16.7.0 Preview 3.1:

Address Sanitizer
Нет Leak Sanitizer (его в принципе под Windows нет)
Теперь есть под x86_64. Работает пока плохо (с MFC не работает совсем, с консольным hello world можно попытаться).

x86_32 вариант можно аккуратно пробовать использовать в экспериментальном режиме

Address Sanitizer

Лучше всего начинать с него

Address Sanitizer

Лучше всего начинать с него

- можно смешивать код

Address Sanitizer

Лучше всего начинать с него

- можно смешивать код
- умеренная нагрузка на компилятор

Address Sanitizer

Лучше всего начинать с него

- можно смешивать код
- умеренная нагрузка на компилятор
- умеренное падение производительности (2x)

Address Sanitizer

Лучше всего начинать с него

- можно смешивать код
- умеренная нагрузка на компилятор
- умеренное падение производительности (2x)
- поддерживается даже древними компиляторами (clang 3.2, gcc 4.8)

Address Sanitizer

Лучше всего начинать с него

- можно смешивать код
- умеренная нагрузка на компилятор
- умеренное падение производительности (2x)
- поддерживается даже древними компиляторами (clang 3.2, gcc 4.8)
- бонусом идет Leak Sanitizer

Address Sanitizer

Лучше всего начинать с него

- можно смешивать код
- умеренная нагрузка на компилятор
- умеренное падение производительности (2x)
- поддерживается даже древними компиляторами (clang 3.2, gcc 4.8)
- бонусом идет Leak Sanitizer
- ловит самые неприятный и частый класс ошибок

Address Sanitizer

Что ловит

- use after free
- heap buffer overflow
- stack buffer overflow
- global buffer overflow
- use after return
- use after scope
- global initialization order
- memory leaks (LSan)

Address Sanitizer

Что ловит

- use after free
- heap buffer overflow
- stack buffer overflow
- global buffer overflow
- use after return
- use after scope
- global initialization order
- memory leaks (LSan)

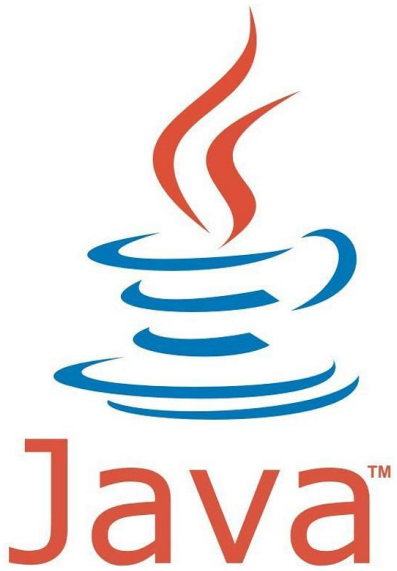
Address Sanitizer

Чтобы избавиться от этих ошибок придумали целый язык

Address Sanitizer

Чтобы избавиться от этих ошибок придумали целый язык

Java



Address Sanitizer

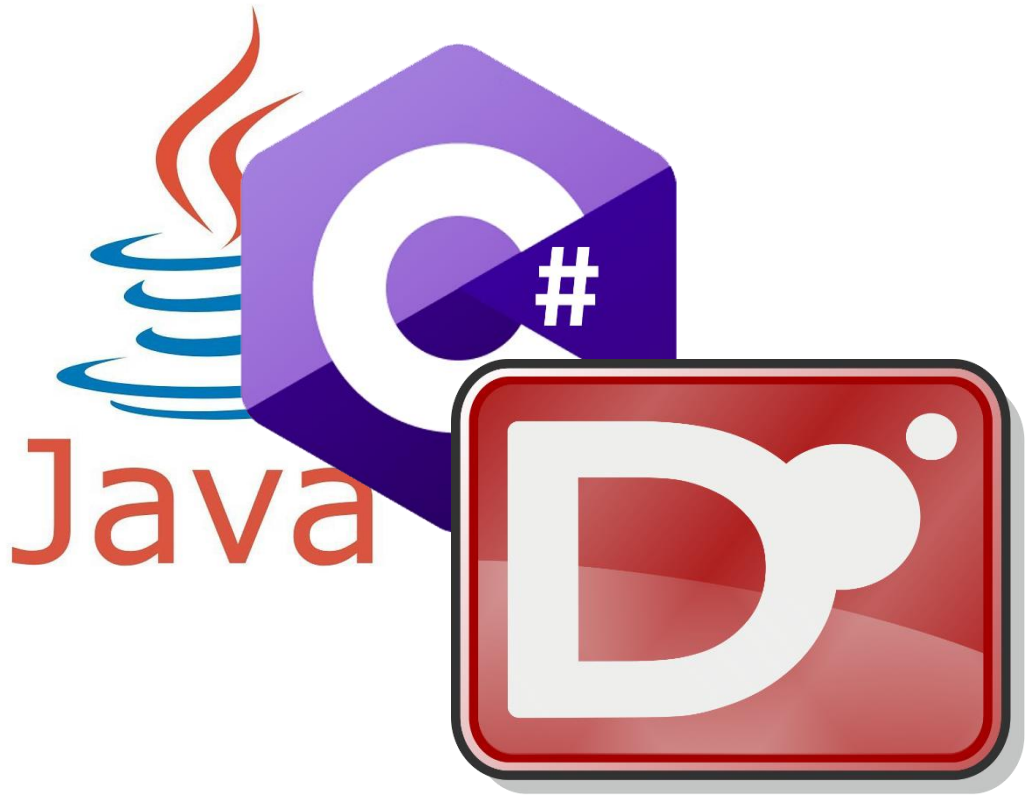
Чтобы избавиться от этих ошибок придумали целый язык

Java
C#

Address Sanitizer

Чтобы избавиться от этих ошибок придумали целый язык

Java
C#
D



Address Sanitizer

Чтобы избавиться от этих ошибок придумали целый язык

Java
C#
D
Go

Address Sanitizer

Чтобы избавиться от этих ошибок придумали целый язык

Java
C#
D
Go
Rust

Address Sanitizer

```
$ ./a.out
=================================================================
==2982==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x602000000011 at pc
0x000000400759 bp 0x7ffcfd5b24b0 sp 0x7ffcfd5b24a0
WRITE of size 1 at 0x602000000011 thread T0
    #0 0x400758 in main /home/build/Projects/atest/test.cpp:4
    #1 0x7f962d6dd494 in __libc_start_main (/lib64/libc.so.6+0x22494)
    #2 0x400658  (/home/build/Projects/atest/a.out+0x400658)

0x602000000011 is located 0 bytes to the right of 1-byte region
[0x602000000010,0x602000000011)
allocated by thread T0 here:
    #0 0x7f962e3871a8 in operator new(unsigned long) (/lib64/libasan.so.4+0xe01a8)
    #1 0x400718 in main /home/build/Projects/atest/test.cpp:3
    #2 0x7f962d6dd494 in __libc_start_main (/lib64/libc.so.6+0x22494)
SUMMARY: AddressSanitizer: heap-buffer-overflow /home/build/Projects/atest/test.cpp:4 in
main
```

Address Sanitizer

```
Shadow bytes around the buggy address:
  0x0c047fff7fb0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c047fff7fc0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c047fff7fd0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c047fff7fe0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c047fff7ff0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x0c047fff8000: fa fa[01]fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c047fff8010: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c047fff8020: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c047fff8030: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c047fff8040: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c047fff8050: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
```

```
 Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
  ASan internal:           fe
  Left alloca redzone:     ca
  Right alloca redzone:    cb
```

Address Sanitizer основная идея

Найдем все обращения к памяти и проверим

```
*address = …;        => if (IsPoisoned(address)) {
                            ReportError(address, …);
                        }
                        *address = …;
```

Address Sanitizer основная идея

Нужна быстрая функция `IsPoisoned`

Address Sanitizer основная идея

Нужна быстрая функция `IsPoisoned`

ShadowMemory – область памяти, где будем хранить состояние основной памяти

Address Sanitizer основная идея

ShadowMemory - область памяти, где будем хранить состояние основной памяти

```
shadow_address = address + kOffset;
if (*shadow_address == kPoisoned) {
    ReportError(address, …);
}
*address = …;
```

Address Sanitizer основная идея

ShadowMemory - область памяти, где будем хранить состояние основной памяти

```
shadow_address = address + kOffset;
if (*shadow_address == kPoisoned) {
    ReportError(address, …);
}
*address = …;
```

Address Sanitizer основная идея

ShadowMemory - область памяти, где будем хранить состояние основной памяти

```
shadow_address = address + kOffset;
if (*shadow_address == kPoisoned) {
    ReportError(address, …);
}
*address = …;
```

Address Sanitizer основная идея

ShadowMemory - область памяти, где будем хранить состояние основной памяти

```
shadow_address = address + kOffset;
if (*shadow_address == kPoisoned) {
    ReportError(address, …);
}
*address = …;
```

Address Sanitizer основная идея

ShadowMemory - область памяти, где будем хранить состояние основной памяти

```
shadow_address = address + kOffset;
if (*shadow_address == kPoisoned) {
    ReportError(address, …);
}
*address = …;
```

Нужно слишком много памяти

Address Sanitizer основная идея

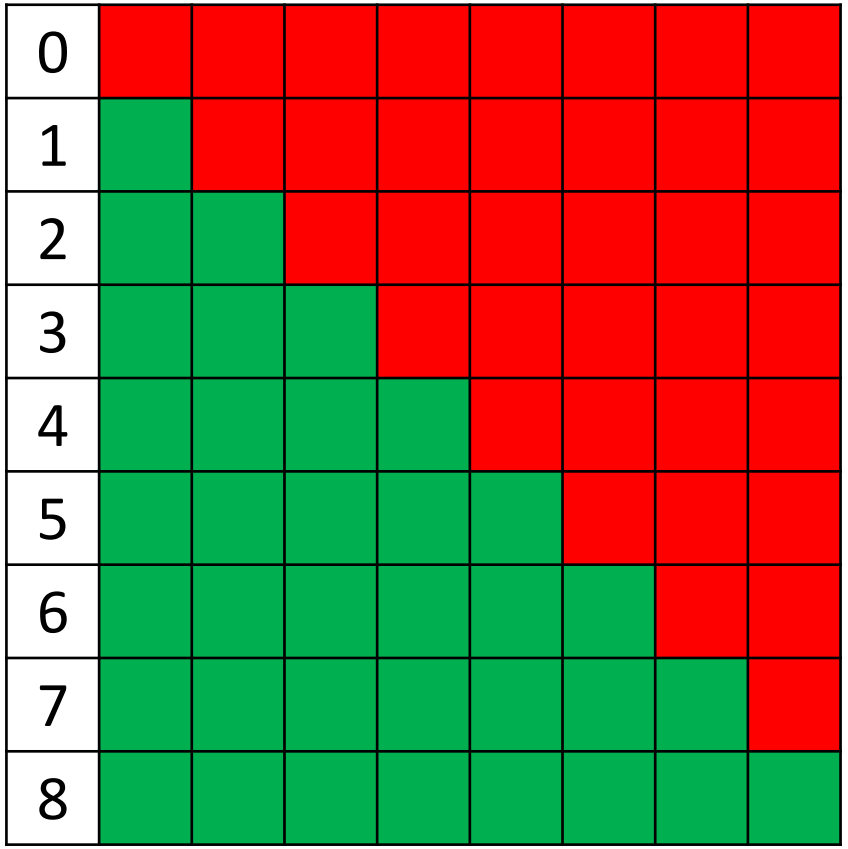
обычно при malloc используется выравнивание

Address Sanitizer основная идея

обычно при malloc используется выравнивание
обычно выравнивание как минимум на 8 байт

Address Sanitizer основная идея

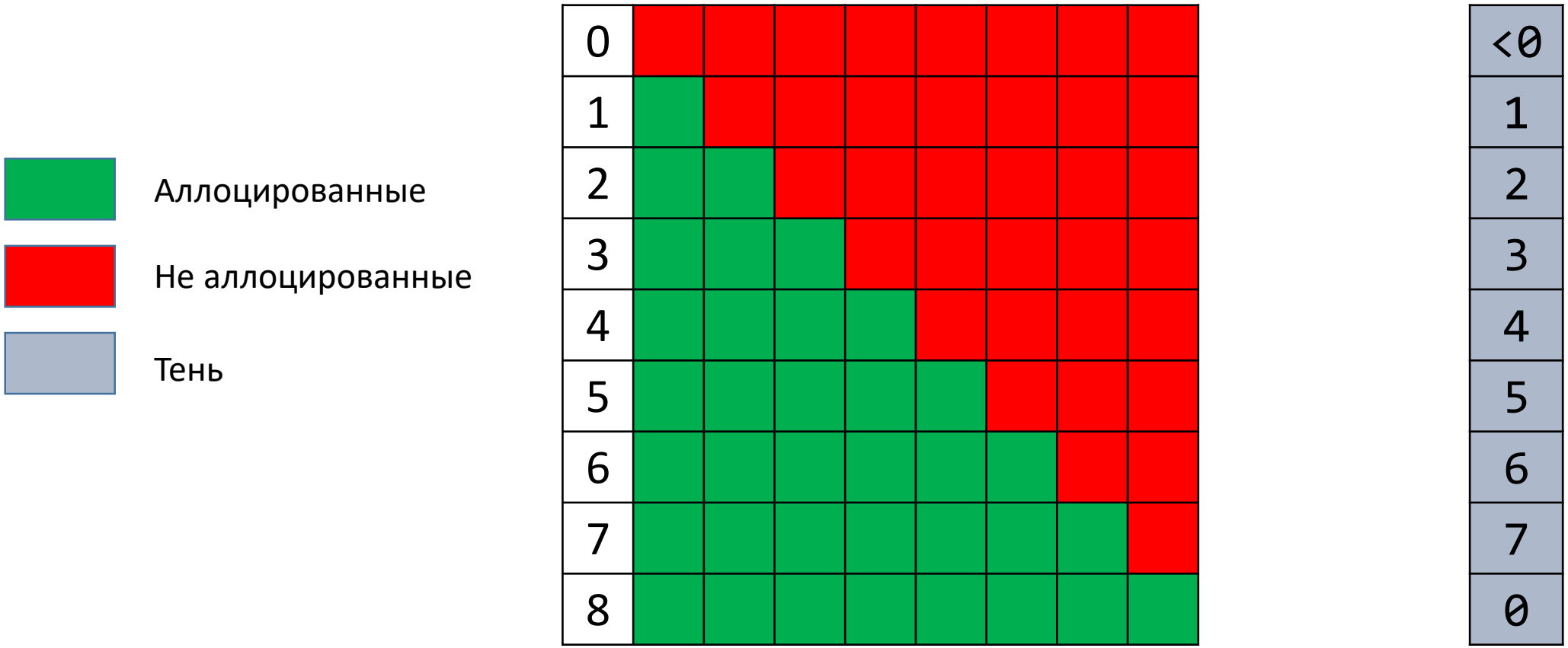
обычно при malloc используется выравнивание
обычно выравнивание как минимум на 8 байт



9 возможных состояний

Address Sanitizer основная идея

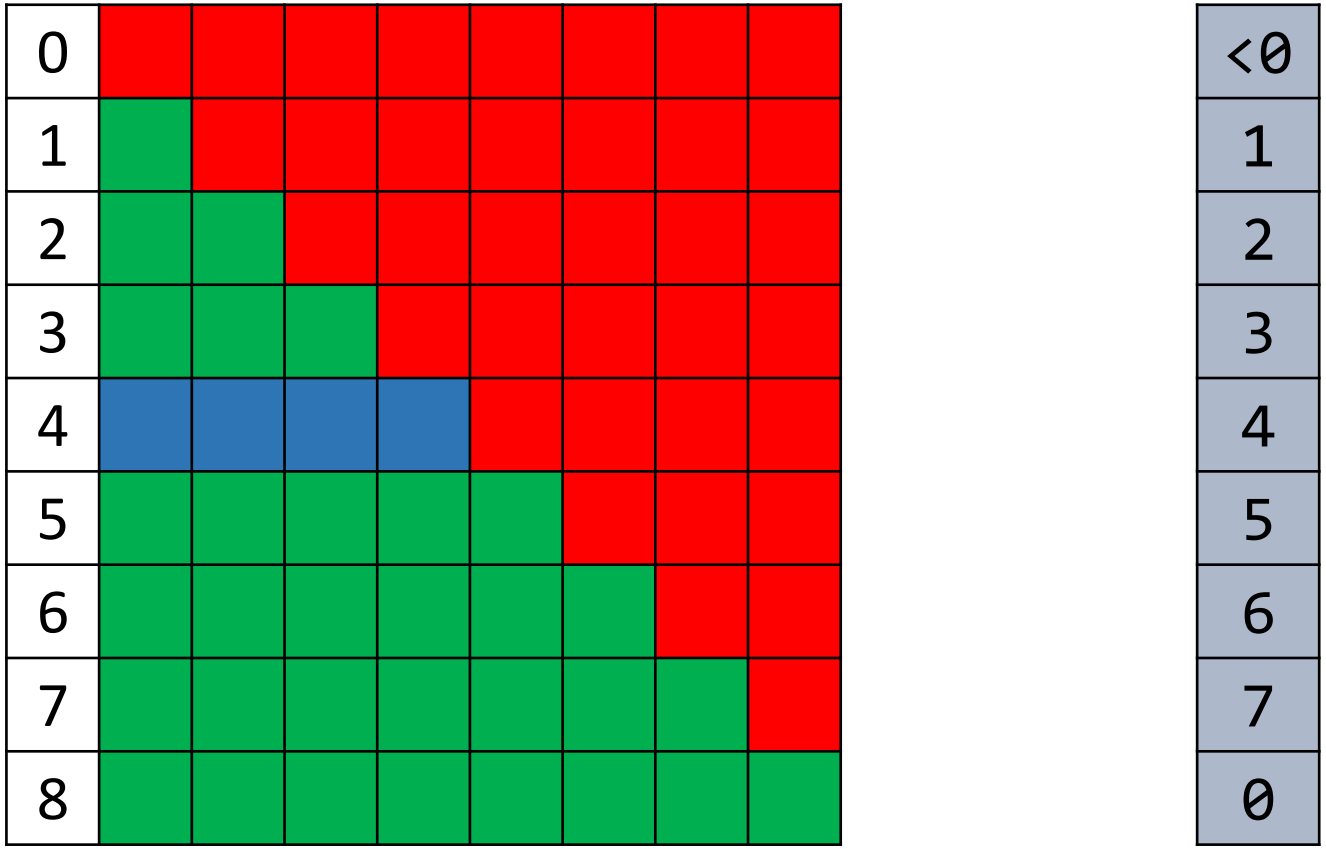
обычно выравнивание как минимум на 8 байт
1 байт в ShadowMemory на 8 байт в основной памяти



Address Sanitizer основная идея

обычно выравнивание как минимум на 8 байт
1 байт в ShadowMemory на 8 байт в основной памяти

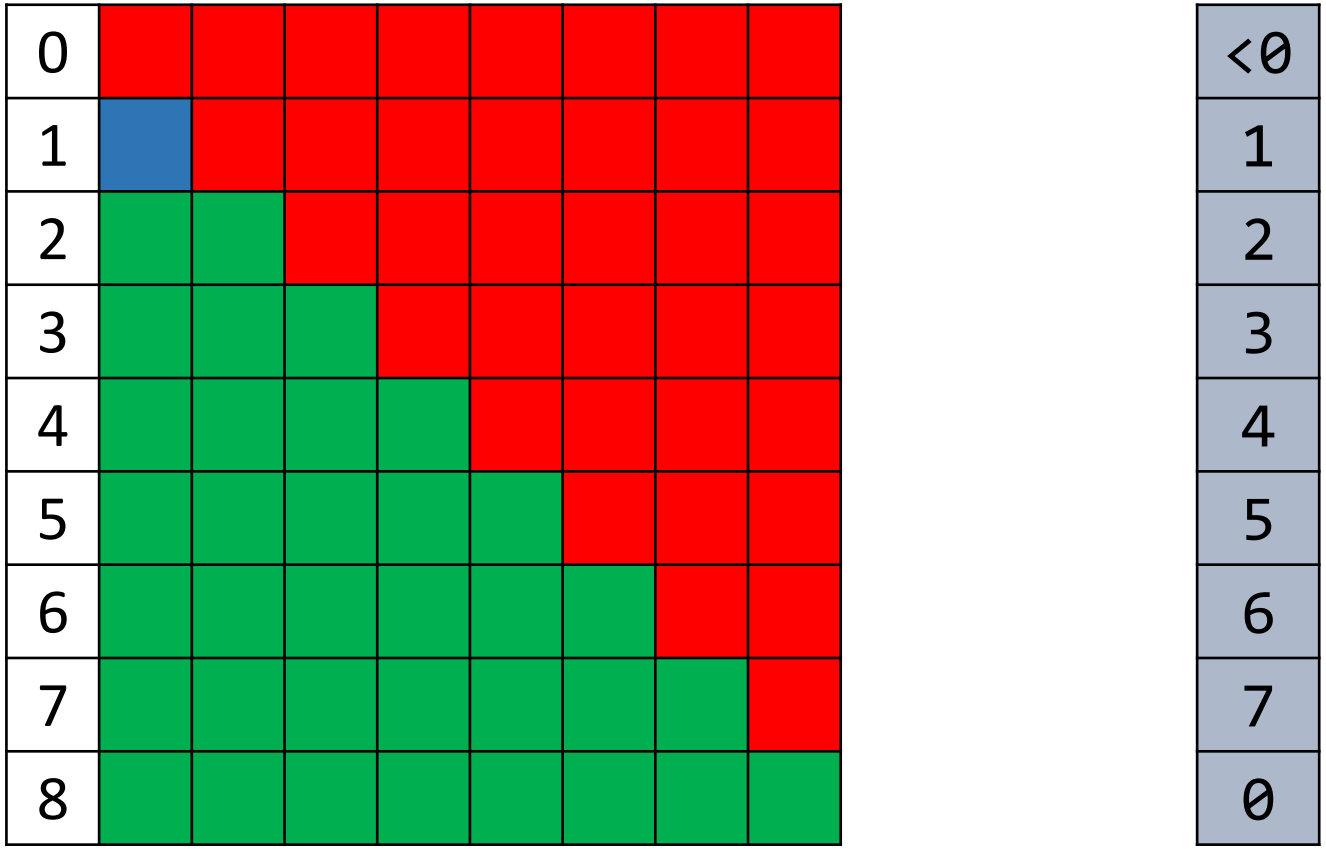
`int32_t`



Address Sanitizer основная идея

обычно выравнивание как минимум на 8 байт
1 байт в ShadowMemory на 8 байт в основной памяти

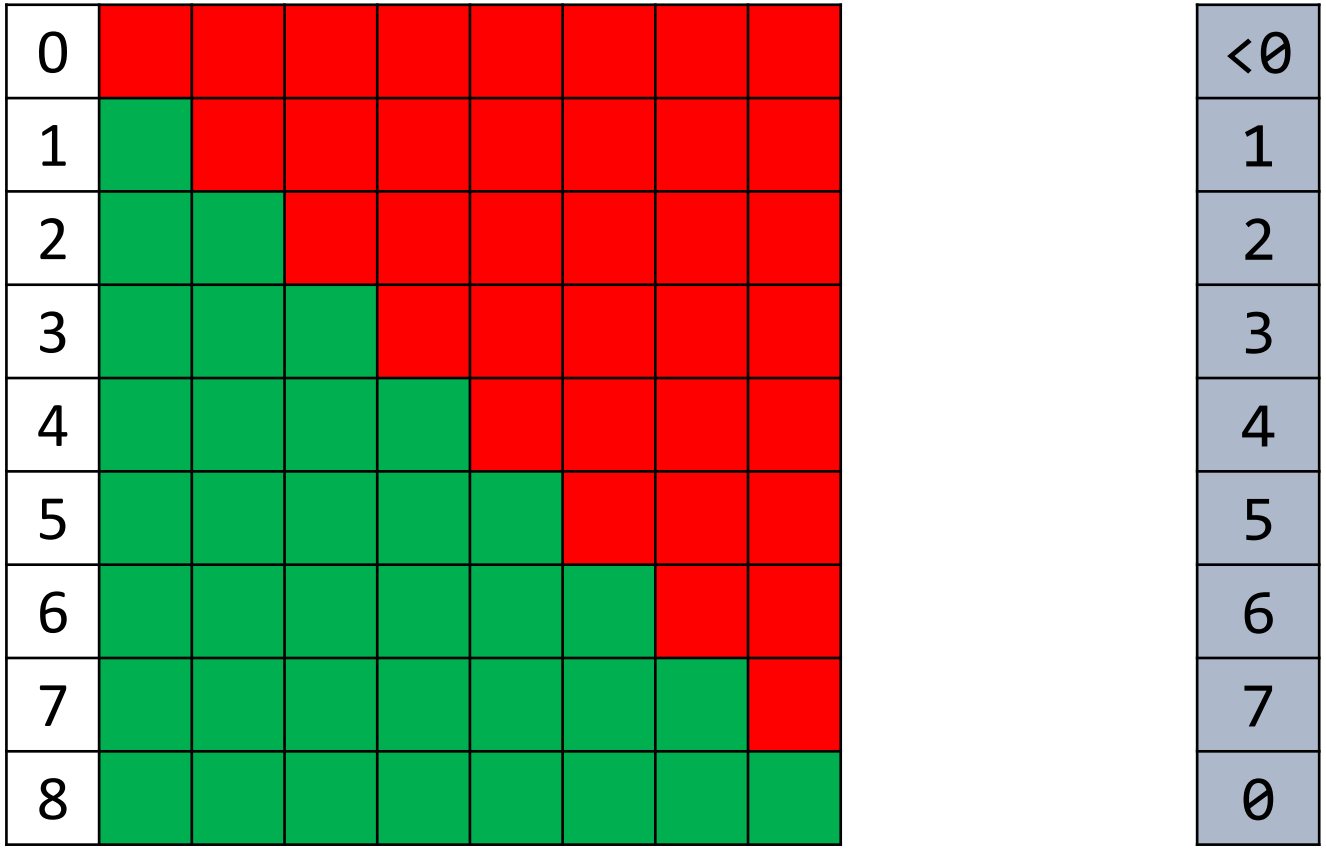
`int8_t`



Address Sanitizer основная идея

~~обычно~~ всегда выравнивание как минимум на 8 байт
1 байт в ShadowMemory на 8 байт в основной памяти

Заменим malloc на свой, гарантирующий выравнивание

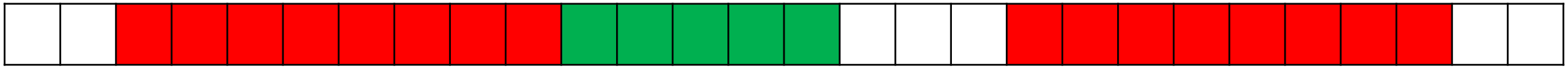


Address Sanitizer основная идея

~~обычно~~ всегда выравнивание как минимум на 8 байт

свой malloc:

- гарантирует выравнивание на 8
- добавляет red zone вокруг аллокаций

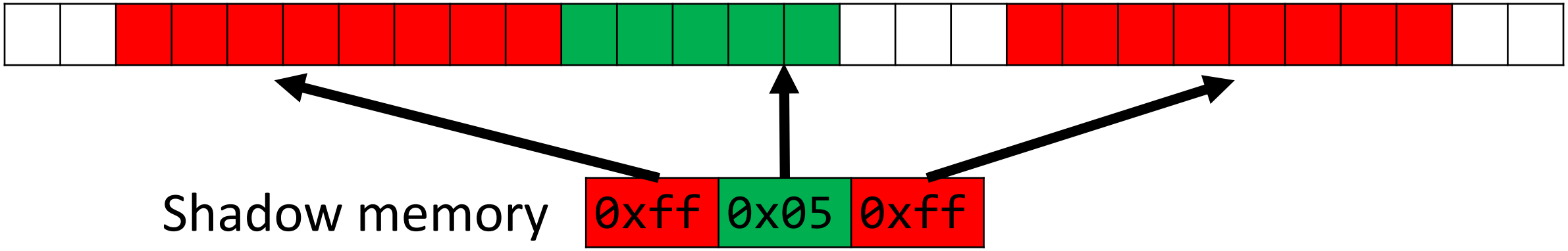


Address Sanitizer основная идея

~~обычно~~ всегда выравнивание как минимум на 8 байт

свой malloc:

- гарантирует выравнивание на 8
- добавляет red zone вокруг аллокаций

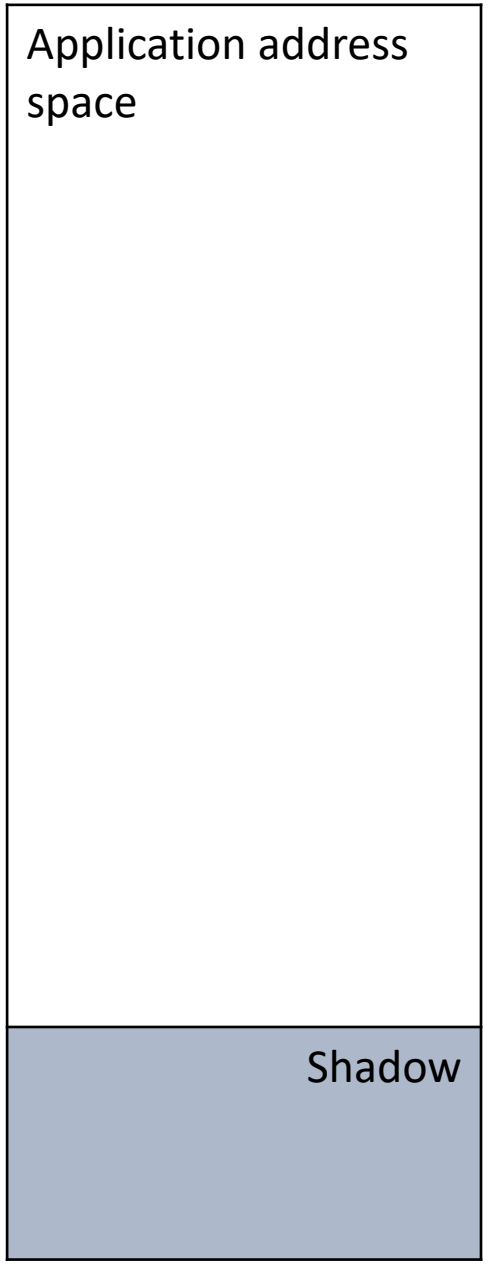


Address Sanitizer основная идея

```
shadow_address = address/8;
```

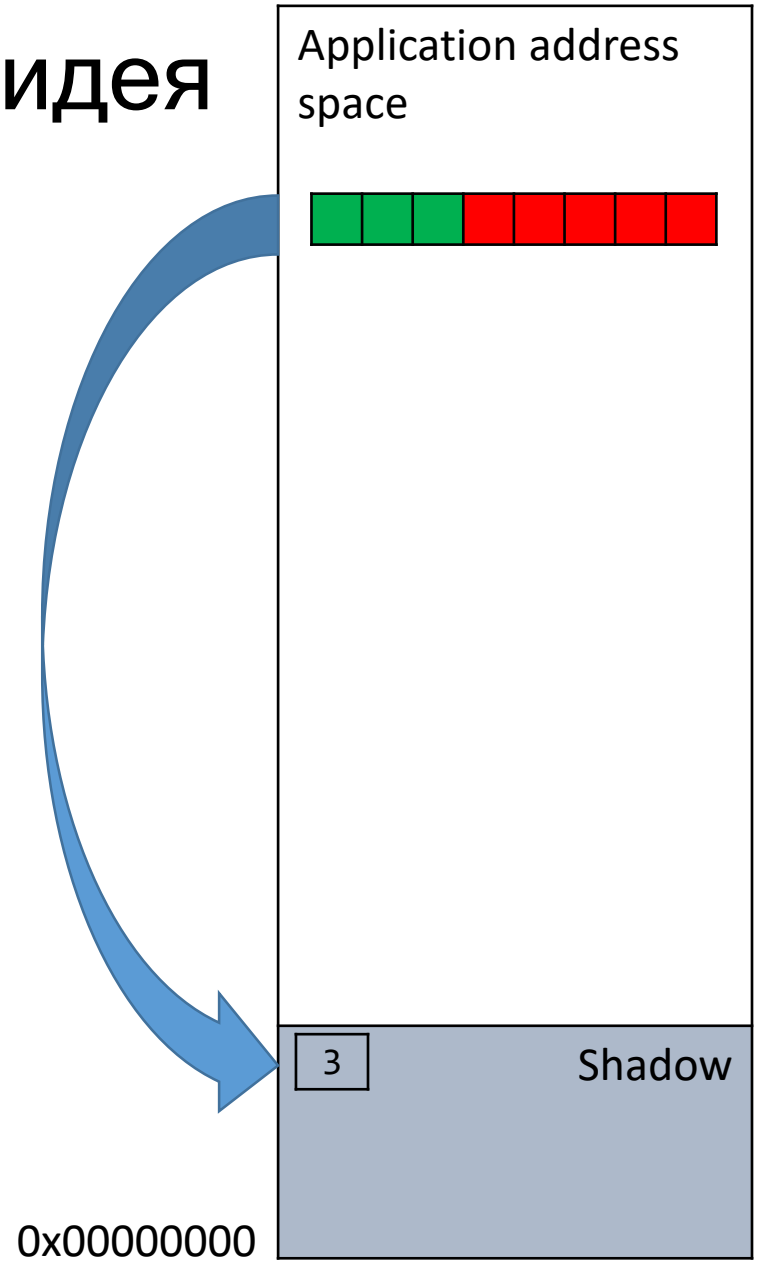
Address Sanitizer основная идея

```
shadow_address = address >> 3;
```



Address Sanitizer основная идея

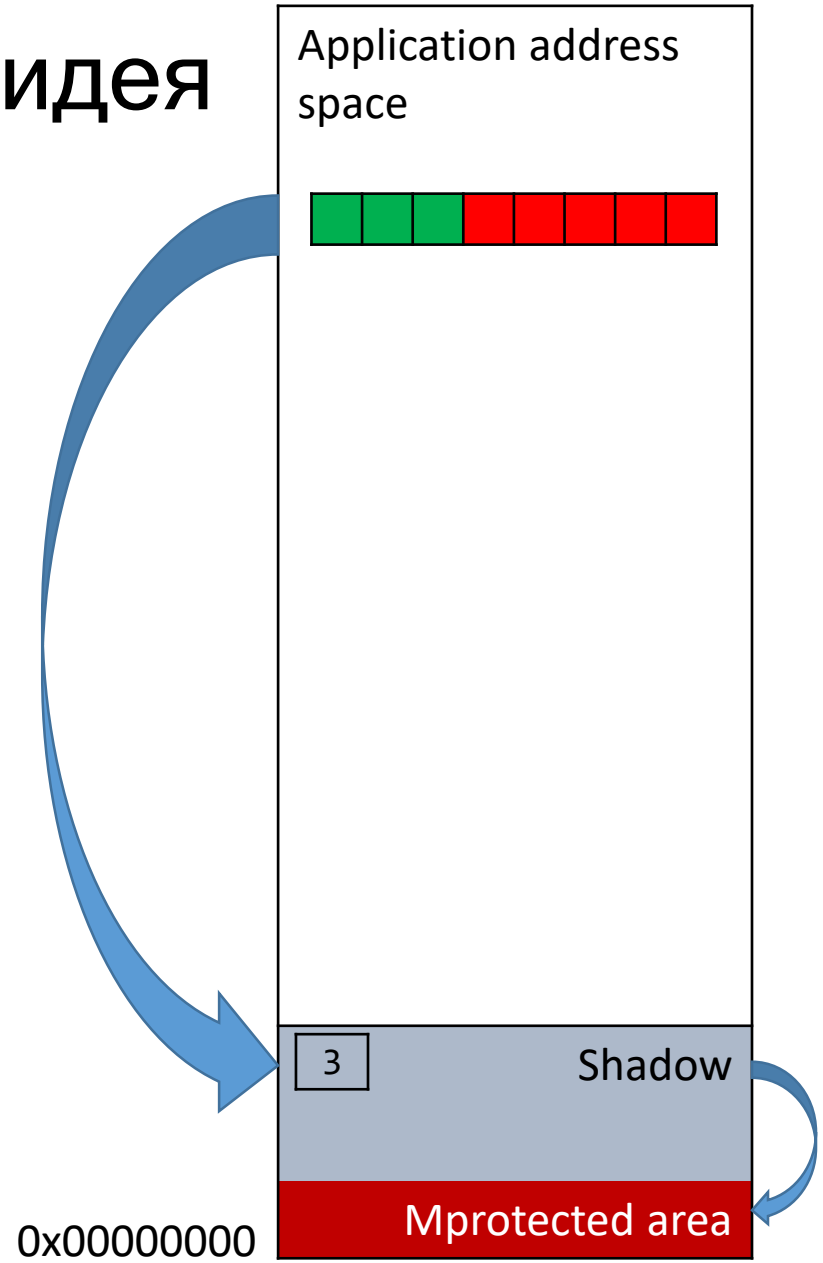
```
shadow_address = address >> 3;
```



Address Sanitizer основная идея

```
shadow_address = address >> 3;
```

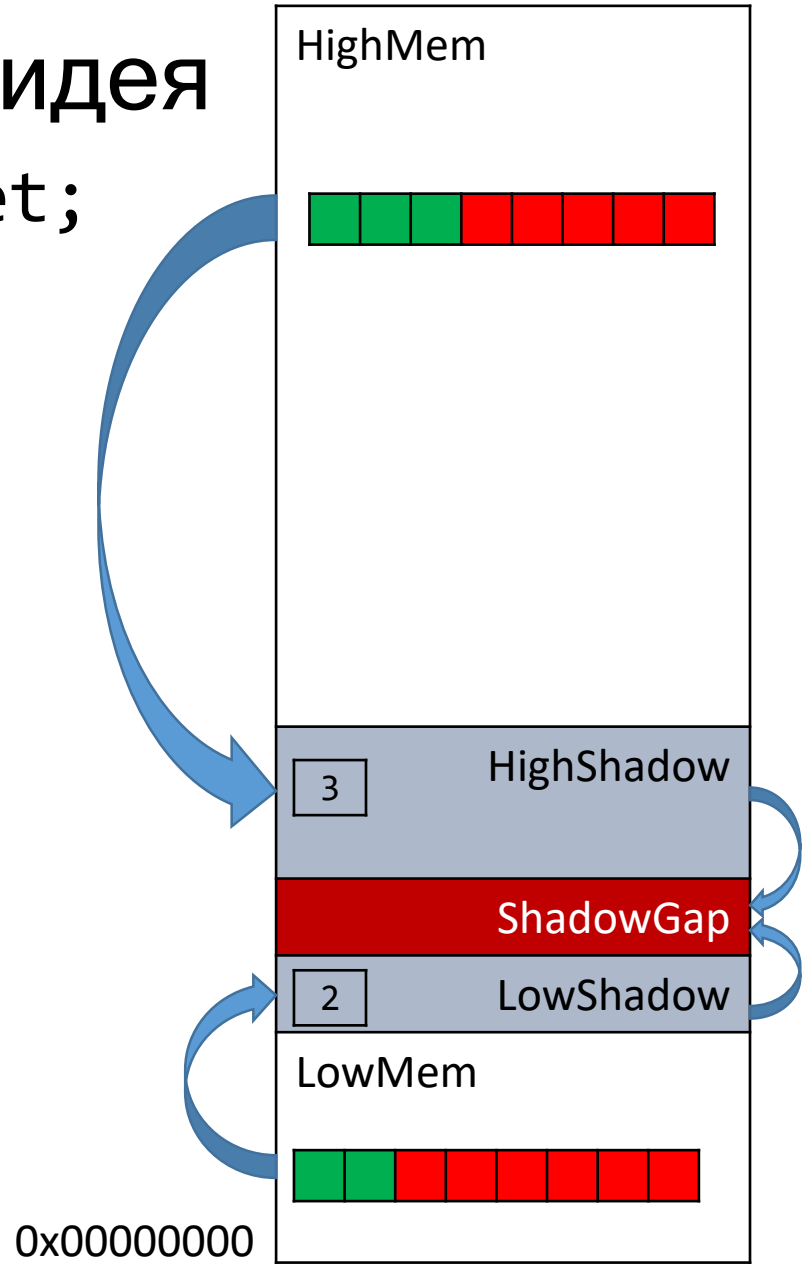
Защитим Shadow memory от доступа пользовательского кода



Address Sanitizer основная идея

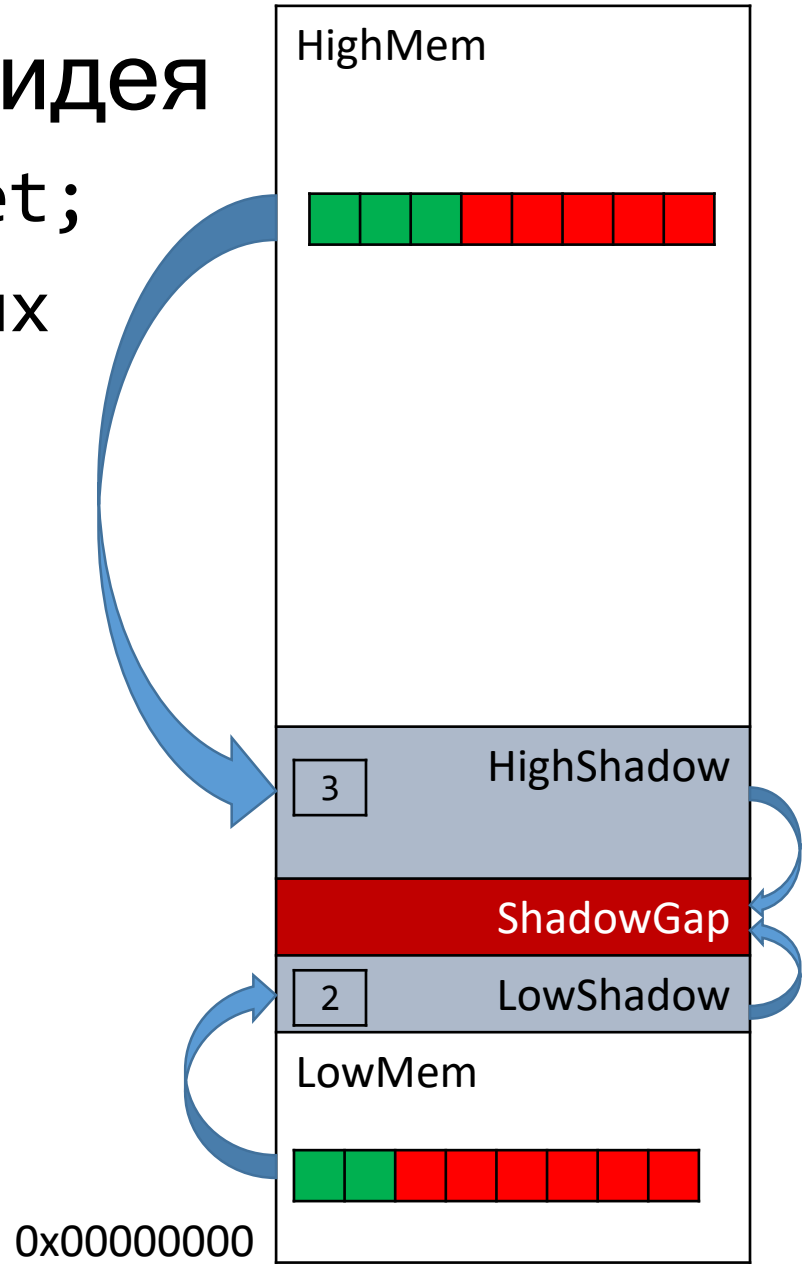
```
shadow_address = (address >> 3) + kOffset;
```

В реальности нулевые адреса заняты, подвинем Shadow



Address Sanitizer основная идея

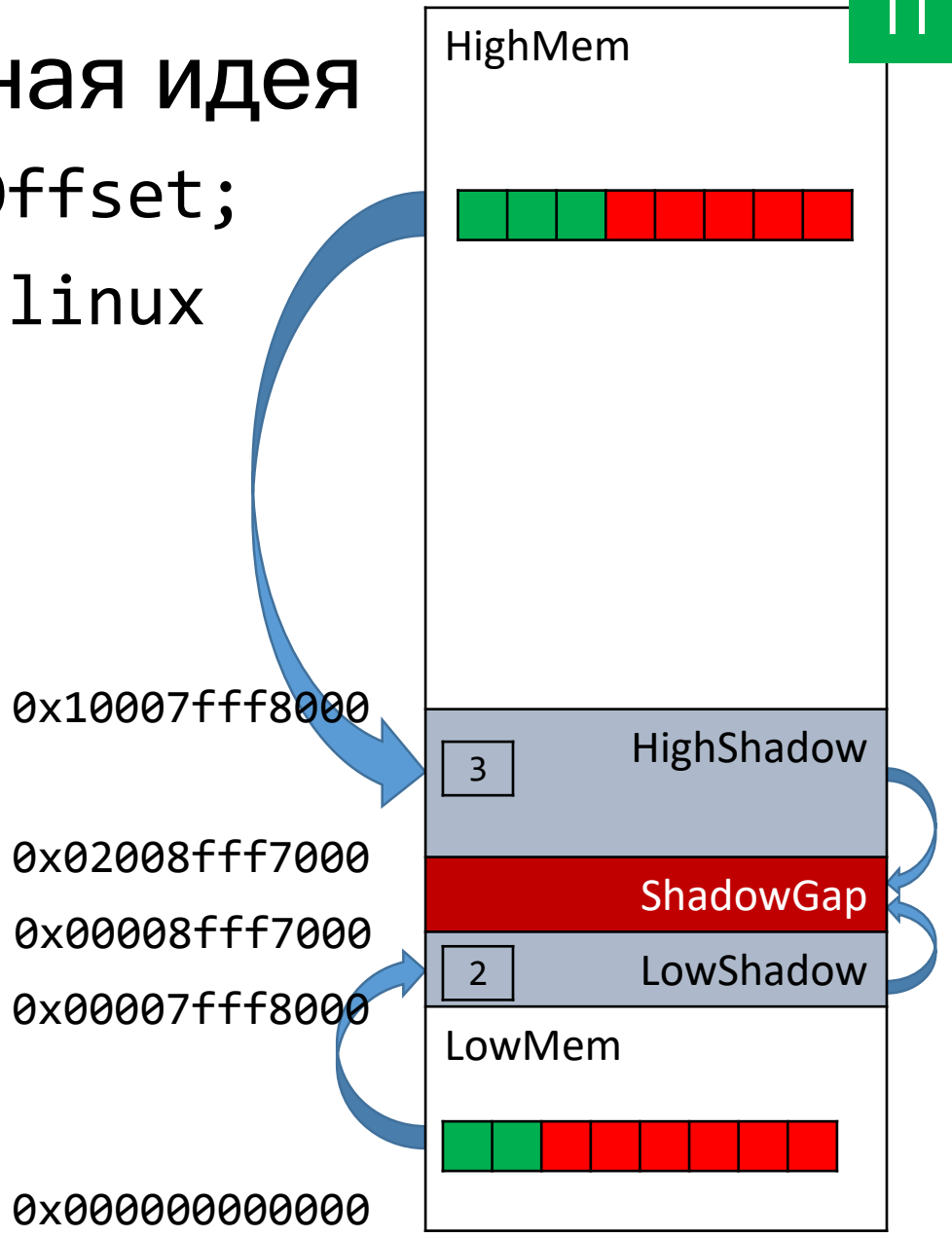
```
shadow_address = (address >> 3) + kOffset;
kOffset = 0x7fff8000; // for x86_64 linux
```



Address Sanitizer основная идея

```
shadow_address = (address >> 3) + kOffset;
kOffset = 0x7fff8000; // for x86_64 linux
```

[0x10007fff8000, 0x7fffffffffff]	HighMem
[0x02008fff7000, 0x10007fff7fff]	HighShadow
[0x00008fff7000, 0x02008fff6fff]	ShadowGap
[0x00007fff8000, 0x00008fff6fff]	LowShadow
[0x000000000000, 0x00007fff7fff]	LowMem



Address Sanitizer основная идея

Защита стека

```
void foo() {
    char a[13];
}
```

Address Sanitizer основная идея

Защита стека

Окружим переменные на стеке редзонами.

```
void foo() {
    char a[13];
}
```

Защита стека

```
void foo() {

    char a[13];

}
```

Защита стека

```
void foo() {
    char redzone1[32]
    char a[13];









}
```

Защита стека

```
void foo() {
    char redzone1[32]
    char a[13];
    char redzone2[32-13];




}
```

Защита стека

```
void foo() {
    char redzone1[32]
    char a[13];
    char redzone2[32-13];
    char redzone3[32];



}
```

Защита стека

```
void foo() {
    char redzone1[32]
    char a[13];
    char redzone2[32-13];
    char redzone3[32];
    sh = MemToShadow(redzone1);









}
```

Защита стека

```
void foo() {
    char redzone1[32]
    char a[13];
    char redzone2[32-13];
    char redzone3[32];
    sh = MemToShadow(redzone1);
    sh[0]= 0xffffffff;
    sh[1]= 0xffff0500;
    sh[2]= 0xffffffff;



}
```

Защита стека

```
void foo() {
    char redzone1[32]
    char a[13];
    char redzone2[32-13];
    char redzone3[32];
    sh = MemToShadow(redzone1);
    sh[0]= 0xffffffff;
    sh[1]= 0xffff0500;
    sh[2]= 0xffffffff;
    …
    sh[0]=sh[1]=sh[2]=0;
}
```

Address Sanitizer детали организации

Address Sanitizer детали организации

Поддержка компиляторами:

Address Sanitizer детали организации

Поддержка компиляторами:

отдельная реализация clang, отдельная в gcc

Address Sanitizer детали организации

Поддержка компиляторами:

отдельная реализация clang, отдельная в gcc

Runtime библиотека:

Address Sanitizer детали организации

Поддержка компиляторами:

отдельная реализация clang, отдельная в gcc

Runtime библиотека:

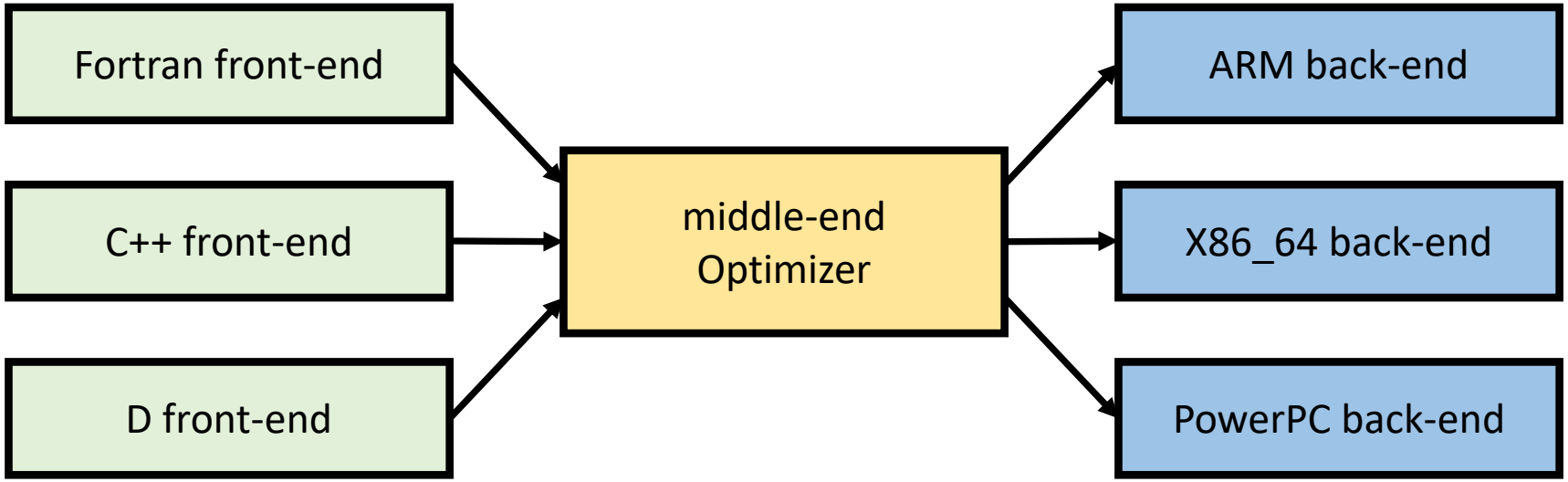
единая реализация, развивается в llvm проекте, бекпортируется в дерево gcc

LLVM: https://github.com/llvm/llvm-project/tree/master/compiler-rt/lib

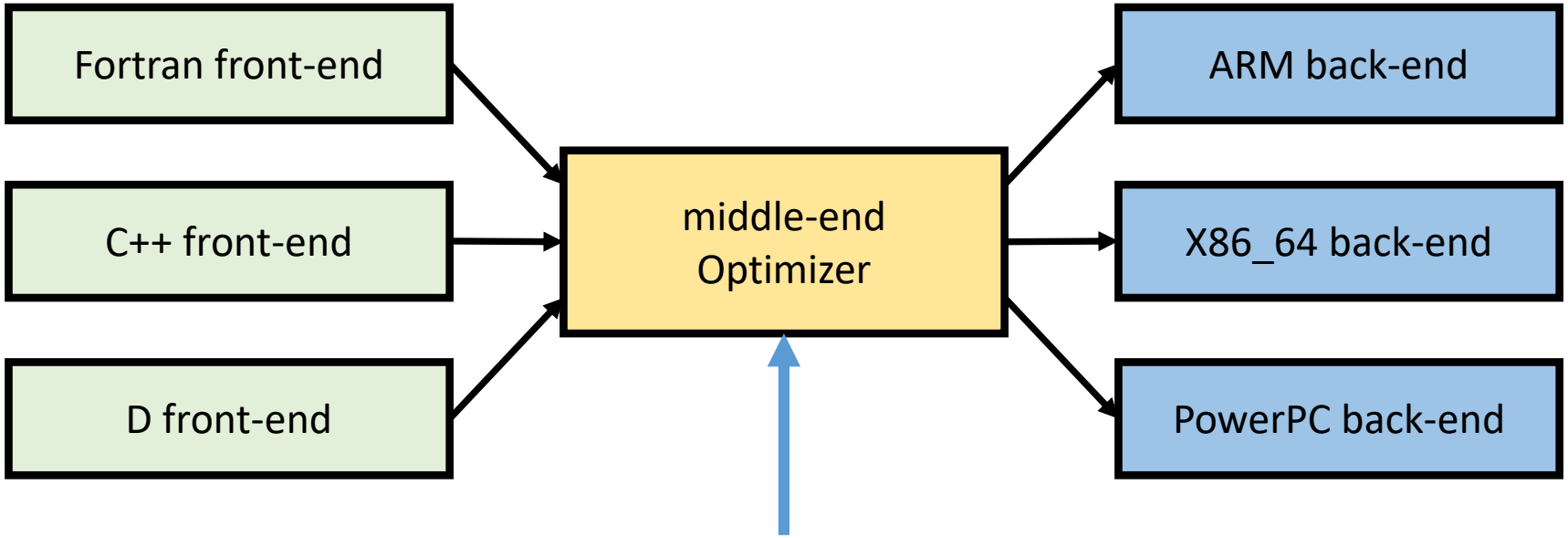
GCC: https://github.com/gcc-mirror/gcc/tree/master/libsanitizer

Address Sanitizer детали реализации
поддержка компилятором

Address Sanitizer детали реализации
поддержка компилятором

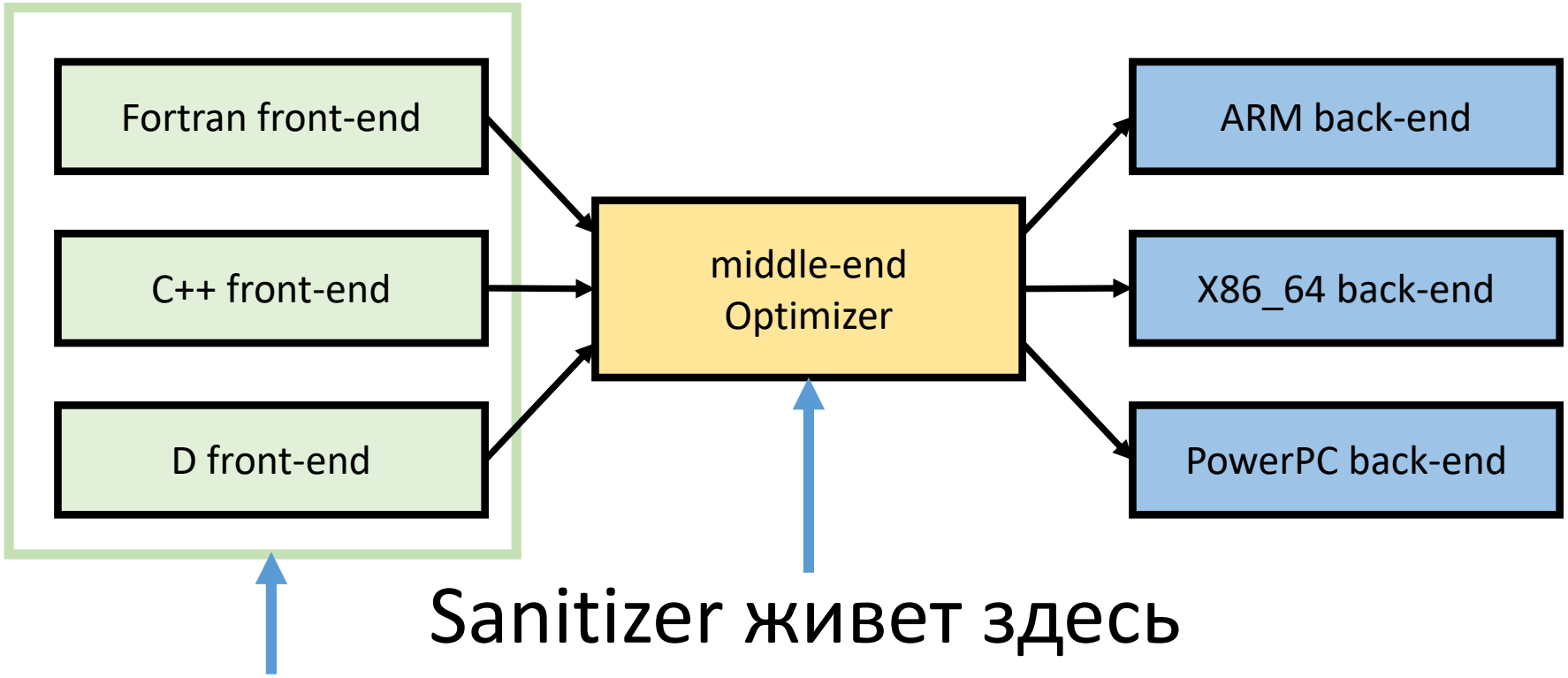


Address Sanitizer детали реализации
поддержка компилятором



Sanitizer живет здесь

Address Sanitizer детали реализации
поддержка компилятором



Sanitizer живет здесь

Но front-end может конфигурировать sanitizer

Address Sanitizer детали реализации

санитайзеры не только для C++

gcc

Address Sanitizer детали реализации

санитайзеры не только для C++

gcc

Ada

Address Sanitizer детали реализации

санитайзеры не только для C++

gcc

Ada

D

Address Sanitizer детали реализации

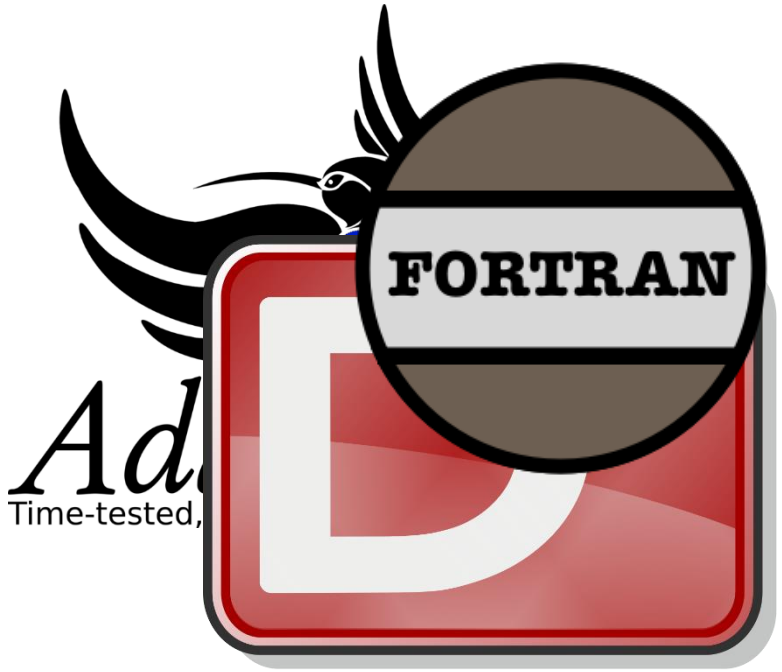
санитайзеры не только для C++

gcc

Ada

D

Fortran



Address Sanitizer детали реализации

санитайзеры не только для C++

gcc

Ada

D

Fortran

Go

Address Sanitizer детали реализации

санитайзеры не только для C++

gcc

Ada

D

Fortran

Go

LLVM

Go

Address Sanitizer детали реализации

санитайзеры не только для C++

gcc

Ada

D

Fortran

Go

LLVM

Go

D

Address Sanitizer детали реализации

санитайзеры не только для C++

gcc

Ada

D

Fortran

Go

LLVM

Go

D

Rust

Address Sanitizer детали реализации

санитайзеры не только для C++

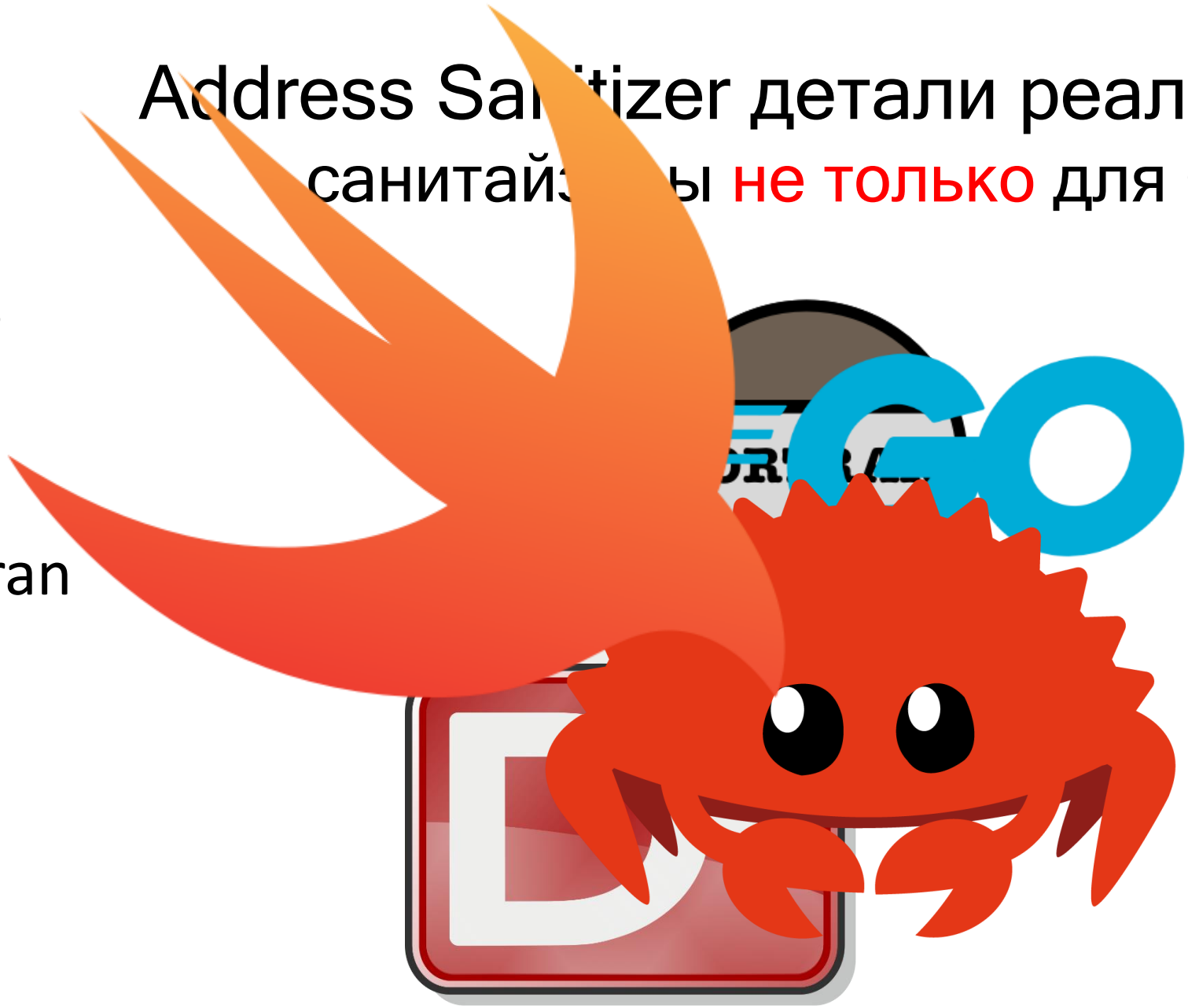
gcc

Ada

D

Fortran

Go



LLVM

Go

D

Rust

Swift

Address Sanitizer детали реализации
поддержка компилятором

GCC

Address Sanitizer детали реализации
поддержка компилятором

GCC

- master/gcc/asan.h
- master/gcc/asan.c
- master/gcc/sanopt.c

Address Sanitizer детали реализации
поддержка компилятором

GCC

master/gcc/asan.h
master/gcc/asan.c
master/gcc/sanopt.c

Оформлены как оптимизационные проходы по GIMPLE:

- `pass_sanopt`
- `pass_asan`
- `pass_asan_O0`

Address Sanitizer детали реализации
поддержка компилятором

LLVM

master/llvm/include/llvm/Transforms/Instrumentation/AddressSanitizer.h

master/llvm/lib/Transforms/Instrumentation/AddressSanitizer.cpp

Address Sanitizer детали реализации
поддержка компилятором

LLVM

master/llvm/include/llvm/Transforms/Instrumentation/AddressSanitizer.h

master/llvm/lib/Transforms/Instrumentation/AddressSanitizer.cpp

Оформлено в виде 2 проходов: `AddressSanitizerPass`
`ModuleAddressSanitizerPass`

Address Sanitizer детали реализации
поддержка компилятором

LLVM

master/llvm/include/llvm/Transforms/Instrumentation/AddressSanitizer.h

master/llvm/lib/Transforms/Instrumentation/AddressSanitizer.cpp

`AddressSanitizerPass` – инструментирует тела функций

Address Sanitizer детали реализации
поддержка компилятором

LLVM

master/llvm/include/llvm/Transforms/Instrumentation/AddressSanitizer.h

master/llvm/lib/Transforms/Instrumentation/AddressSanitizer.cpp

`AddressSanitizerPass` – инструментирует тела функций

`ModuleAddressSanitizerPass` – создает глобальные переменные для поддержки ASan в единицах компиляции

Address Sanitizer детали реализации

```
1  #include <cstdint>
2  void foo(uint64_t* a) {
3      *a = 42;
4  }
```

```
1   foo(unsigned long*):
2           mov     rax, rdi
3           shr     rax, 3
4           cmp     BYTE PTR [rax+2147450880], 0
5           jne     .L7
6           mov     QWORD PTR [rdi], 42
7           ret
8   .L7:                                0x7fff8000
9           push    rax
10          call    __asan_report_store8
```

Address Sanitizer детали реализации

```
1  #include <cstdint>
2  void foo(uint64_t* a) {
3      *a = 42;
4  }
```

```
mov a,b ⇔ a = b
shr a,x ⇔ a >> x
cmp a,b ⇔ if (a==b) flag=1
jne L   ⇔ if (!flag) goto L
```

rdi – наш аргумент функции (a)

```
1  foo(unsigned long*):
2          mov     rax, rdi
3          shr     rax, 3
4          cmp     BYTE PTR [rax+2147450880], 0
5          jne     .L7
6          mov     QWORD PTR [rdi], 42
7          ret
8  .L7:                                  0x7fff8000
9          push    rax
10         call    __asan_report_store8
```

```
shadow_address = (address >> 3) + 0x7fff8000;
```

Address Sanitizer детали реализации

```
1  #include <cstdint>
2  void foo(uint64_t* a) {
3      *a = 42;
4  }
```

```
mov a,b ⇔ a = b
shr a,x ⇔ a >> x
cmp a,b ⇔ if (a==b) flag=1
jne L   ⇔ if (!flag) goto L
```

rdi – наш аргумент функции (a)

```
1  foo(unsigned long*):
2          mov     rax, rdi
3          shr     rax, 3
4          cmp     BYTE PTR [rax+2147450880], 0
5          jne     .L7
6          mov     QWORD PTR [rdi], 42
7          ret
8  .L7:                                  0x7fff8000
9          push    rax
10         call    __asan_report_store8
```

```
shadow_address = (address >> 3) + 0x7fff8000;
```

Address Sanitizer детали реализации

```
1  #include <cstdint>
2  void foo(uint64_t* a) {
3      *a = 42;
4  }
```

```
mov a,b ⇔ a = b
shr a,x ⇔ a >> x
cmp a,b ⇔ if (a==b) flag=1
jne L   ⇔ if (!flag) goto L
```

rdi – наш аргумент функции (a)

```
1  foo(unsigned long*):
2          mov     rax, rdi
3          shr     rax, 3
4          cmp     BYTE PTR [rax+2147450880], 0
5          jne     .L7
6          mov     QWORD PTR [rdi], 42
7          ret
8  .L7:                                  0x7fff8000
9          push    rax
10         call    __asan_report_store8
```

```
shadow_address = (address >> 3) + 0x7fff8000;
```

Address Sanitizer детали реализации

```
1  #include <cstdint>
2  void foo(uint64_t* a) {
3      *a = 42;
4  }
```

```
mov a,b ⇔ a = b
shr a,x ⇔ a >> x
cmp a,b ⇔ if (a==b) flag=1
jne L   ⇔ if (!flag) goto L
```

rdi – наш аргумент функции (a)

```
1  foo(unsigned long*):
2          mov     rax, rdi
3          shr     rax, 3
4          cmp     BYTE PTR [rax+2147450880], 0
5          jne     .L7
6          mov     QWORD PTR [rdi], 42
7          ret
8  .L7:                                  0x7fff8000
9          push    rax
10         call    __asan_report_store8
```

```
shadow_address = (address >> 3) + 0x7fff8000;
```

Address Sanitizer детали реализации

```
1  #include <cstdint>
2  void foo(uint64_t* a) {
3      *a = 42;
4  }
```

```
mov a,b ⇔ a = b
shr a,x ⇔ a >> x
cmp a,b ⇔ if (a==b) flag=1
jne L   ⇔ if (!flag) goto L
```

rdi – наш аргумент функции (a)

```
1  foo(unsigned long*):
2          mov     rax, rdi
3          shr     rax, 3
4          cmp     BYTE PTR [rax+2147450880], 0
5          jne     .L7
6          mov     QWORD PTR [rdi], 42
7          ret
8  .L7:                                  0x7fff8000
9          push    rax
10         call    __asan_report_store8
```

```
shadow_address = (address >> 3) + 0x7fff8000;
```

Address Sanitizer детали реализации

```
1  #include <cstdint>
2  void foo(uint64_t* a) {
3      *a = 42;
4  }
```

```
mov a,b ⇔ a = b
shr a,x ⇔ a >> x
cmp a,b ⇔ if (a==b) flag=1
jne L   ⇔ if (!flag) goto L
```

rdi – наш аргумент функции (a)

```
1  foo(unsigned long*):
2          mov     rax, rdi
3          shr     rax, 3
4          cmp     BYTE PTR [rax+2147450880], 0
5          jne     .L7
6          mov     QWORD PTR [rdi], 42
7          ret
8  .L7:                                  0x7fff8000
9          push    rax
10         call    __asan_report_store8
```

```
shadow_address = (address >> 3) + 0x7fff8000;
```

Address Sanitizer детали реализации

```
1  #include <cstdint>
2  void foo(uint64_t* a) {
3      *a = 42;
4  }
```

```
mov a,b ⇔ a = b
shr a,x ⇔ a >> x
cmp a,b ⇔ if (a==b) flag=1
jne L   ⇔ if (!flag) goto L
```

rdi – наш аргумент функции (a)

```
1  foo(unsigned long*):
2          mov     rax, rdi
3          shr     rax, 3
4          cmp     BYTE PTR [rax+2147450880], 0
5          jne     .L7
6          mov     QWORD PTR [rdi], 42
7          ret
8  .L7:                                  0x7fff8000
9          push    rax
10         call    __asan_report_store8
```

```
shadow_address = (address >> 3) + 0x7fff8000;
```

Address Sanitizer детали реализации

```
1  #include <cstdint>
2  void foo(uint64_t* a) {
3      *a = 42;
4  }
```

```
1   foo(unsigned long*):
2           mov     rax, rdi
3           shr     rax, 3
4           cmp     BYTE PTR [rax+2147450880], 0
5           jne     .L7
6           mov     QWORD PTR [rdi], 42
7           ret
8   .L7:                                0x7fff8000
9           push    rax
10          call    __asan_report_store8
```

Address Sanitizer детали реализации

```
1  int main()
2  {
3          uint64_t* p = uint64_t;
4          delete p;
5          foo(p);
6  }
```

Address Sanitizer детали реализации

```
$ gcc -fsanitize=address main.cpp
$ ./a.out
==45087==ERROR: AddressSanitizer: heap-use-after-free on
address 0x602000000010 at pc 0x000000400774 bp 0x7ffc48da31e0
sp 0x7ffc48da31d0
WRITE of size 8 at 0x602000000010 thread T0
    #0 0x400773 in foo(unsigned long long*)
(/home/build/Projects/mm/a.out+0x400773)
    #1 0x4007b4 in main
(/home/build/Projects/mm/a.out+0x4007b4)
```

Address Sanitizer детали реализации

```
1  #include <cstdint>
2  void foo(uint32_t* a) {
3      *a = 42;
4  }
```

```
1   foo(unsigned int*):
2           mov     rax, rdi
3           shr     rax, 3
4           movzx   edx, BYTE PTR [rax+2147450880]
5           mov     rax, rdi
6           and     eax, 7
7           add     eax, 3
8           cmp     al, dl
9           jl      .L2
10          test    dl, dl
11          jne     .L13
12  .L2:
13          mov     DWORD PTR [rdi], 42
14          ret
```

Address Sanitizer детали реализации

```
1  #include <cstdint>
2  void foo(uint32_t* a) {
3      *a = 42;
4  }
```

```
shAddr=(a>>3 + kOffset);
n = (a % 8) + 3;
if (n < *shAddr) *a = 42;
else if (*shAddr==0) *a=42;
else ReportError(a);
```

```
1   foo(unsigned int*):
2           mov     rax, rdi
3           shr     rax, 3
4           movzx   edx, BYTE PTR [rax+2147450880]
5           mov     rax, rdi
6           and     eax, 7
7           add     eax, 3
8           cmp     al, dl
9           jl      .L2
10          test    dl, dl
11          jne     .L13
12  .L2:
13          mov     DWORD PTR [rdi], 42
14          ret
```

Address Sanitizer детали реализации

```
1  #include <cstdint>
2  void foo(uint32_t* a) {
3      *a = 42;
4  }
```

```
shAddr=(a>>3 + kOffset);
n = (a % 8) + 3;
if (n < *shAddr) *a = 42;
else if (*shAddr==0) *a=42;
else ReportError(a);
```

```
1  foo(unsigned int*):
2          mov     rax, rdi
3          shr     rax, 3
4          movzx   edx, BYTE PTR [rax+2147450880]
5          mov     rax, rdi
6          and     eax, 7
7          add     eax, 3
8          cmp     al, dl
9          jl      .L2
10         test    dl, dl
11         jne     .L13
12 .L2:
13         mov     DWORD PTR [rdi], 42
14         ret
```

Address Sanitizer детали реализации

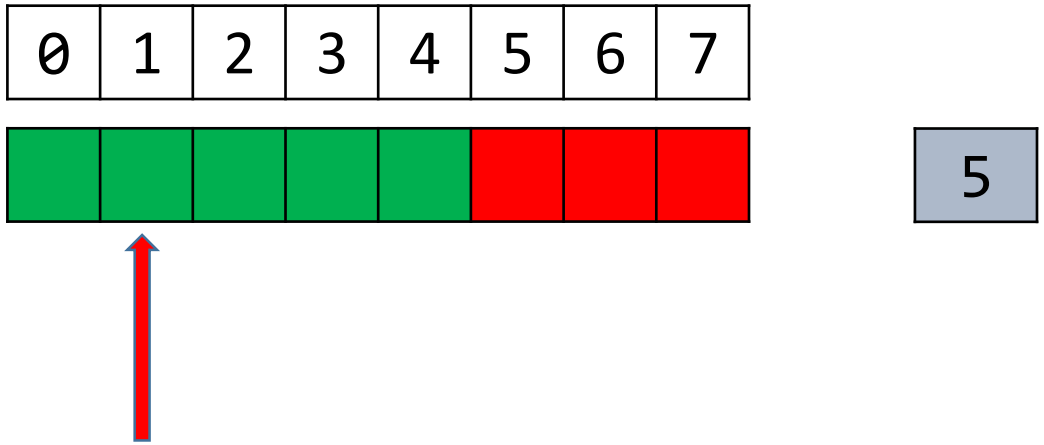
```
1  #include <cstdint>
2  void foo(uint32_t* a) {
3      *a = 42;
4  }
```

```
shAddr=(a>>3 + kOffset);
n = (a % 8) + 3;
if (n < *shAddr) *a = 42;
else if (*shAddr==0) *a=42;
else ReportError(a);
```

```
1  foo(unsigned int*):
2          mov     rax, rdi
3          shr     rax, 3
4          movzx   edx, BYTE PTR [rax+2147450880]
5          mov     rax, rdi
6          and     eax, 7
7          add     eax, 3
8          cmp     al, dl
9          jl      .L2
10         test    dl, dl
11         jne     .L13
12 .L2:
13         mov     DWORD PTR [rdi], 42
14         ret
```

Address Sanitizer детали реализации

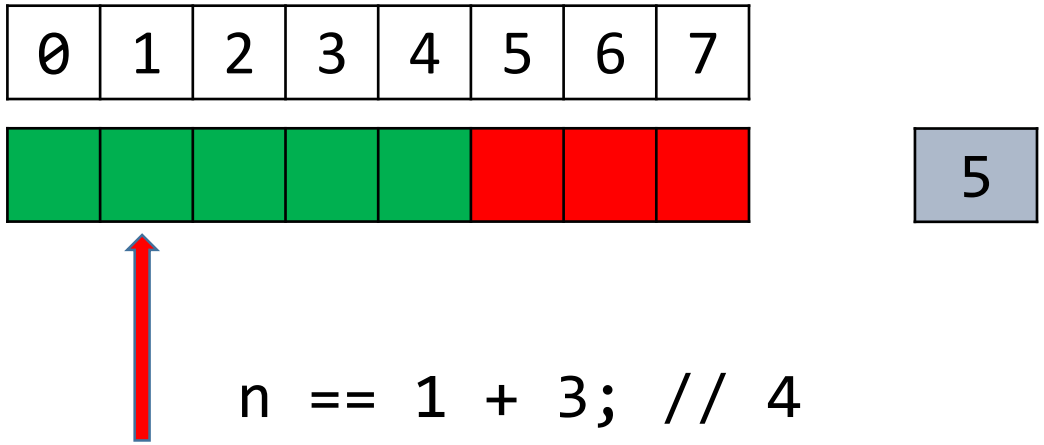
```
1  #include <cstdint>
2  void foo(uint32_t* a) {
3      *a = 42;
4  }
```




```
shAddr=(a>>3 + kOffset);
n = (a % 8) + 3;
if (n < *shAddr) *a = 42;
else if (*shAddr==0) *a=42;
else ReportError(a);
```

Address Sanitizer детали реализации

```
1  #include <cstdint>
2  void foo(uint32_t* a) {
3      *a = 42;
4  }
```

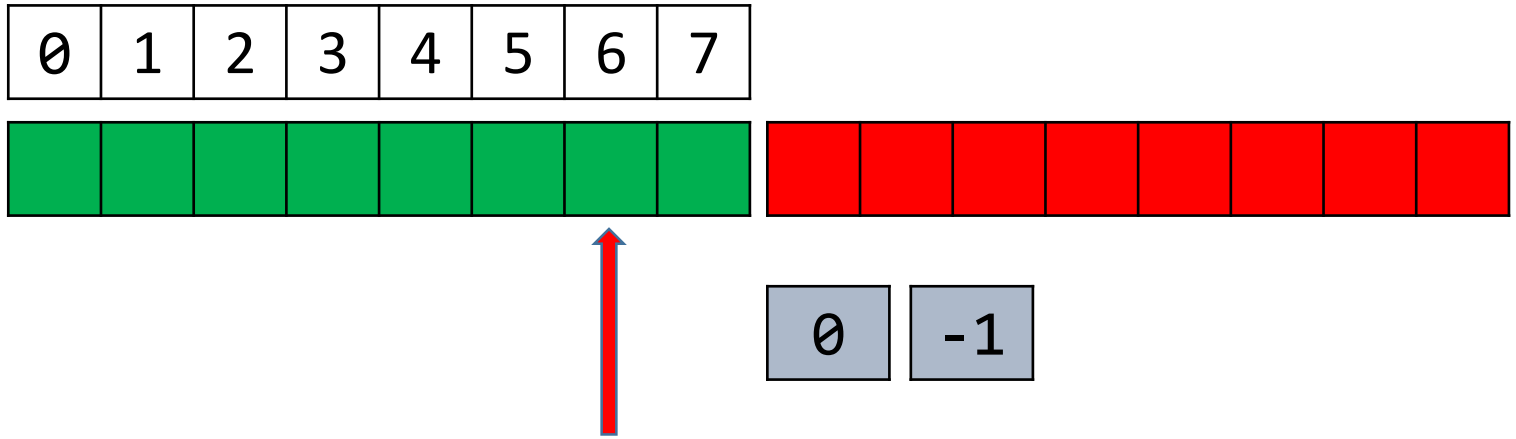



```
shAddr=(a>>3 + kOffset);
n = (a % 8) + 3;
if (n < *shAddr) *a = 42;
else if (*shAddr==0) *a=42;
else ReportError(a);
```

```
n == 1 + 3; // 4
*shAddr == 5;
4 < 5; // ok
```

Address Sanitizer детали реализации

```
1  #include <cstdint>
2  void foo(uint32_t* a) {
3      *a = 42;
4  }
```




```
shAddr=(a>>3 + kOffset);
n = (a % 8) + 3;
if (n < *shAddr) *a = 42;
else if (*shAddr==0) *a=42;
else ReportError(a);
```

Что будет?

Address Sanitizer детали реализации

Давайте найдем эту тройку в компиляторе

```
1   foo(unsigned long*):
2           mov     rax, rdi
3           shr     rax, 3
4           cmp     BYTE PTR [rax+2147450880], 0
5           jne     .L7
6           mov     QWORD PTR [rdi], 42
7           ret
8   .L7:
9           push    rax
10          call    __asan_report_store8
```

Address Sanitizer детали реализации
поддержка компилятором

gcc (asan.h)

```
47   /* Shadow memory is found at
48      (address >> ASAN_SHADOW_SHIFT) + asan_shadow_offset ().  */
49   #define ASAN_SHADOW_SHIFT       3
50   #define ASAN_SHADOW_GRANULARITY (1UL << ASAN_SHADOW_SHIFT)
```

LLVM (AddressSanitizer.cpp)

```
93   static const uint64_t kDefaultShadowScale = 3;
3026     size_t Granularity = 1ULL << Mapping.Scale;
```

Address Sanitizer детали реализации
поддержка компилятором

LLVM (AddressSanitizer.cpp)

```
93    static const uint64_t kDefaultShadowScale = 3;
```

```
3026      size_t Granularity = 1ULL << Mapping.Scale;
```

Address Sanitizer детали реализации
поддержка компилятором

LLVM (AddressSanitizer.cpp)

```
93    static const uint64_t kDefaultShadowScale = 3;
3026      size_t Granularity = 1ULL << Mapping.Scale;
115    static const uint64_t kMyriadShadowScale = 5;
```

Address Sanitizer детали реализации
поддержка компилятором

LLVM (AddressSanitizer.cpp)

```
93    static const uint64_t kDefaultShadowScale = 3;
3026      size_t Granularity = 1ULL << Mapping.Scale;
115    static const uint64_t kMyriadShadowScale = 5;


326   static cl::opt<int> ClMappingScale("asan-mapping-scale",
327                                      cl::desc("scale of asan shadow mapping"),
328                                      cl::Hidden, cl::init(0));
```

Address Sanitizer детали реализации
поддержка компилятором

LLVM (AddressSanitizer.cpp)

```
326  static cl::opt<int> ClMappingScale("asan-mapping-scale",
327                                     cl::desc("scale of asan shadow mapping"),
328                                     cl::Hidden, cl::init(0));

330  static cl::opt<uint64_t>
331      ClMappingOffset("asan-mapping-offset",
332                      cl::desc("offset of asan shadow mapping [EXPERIMENTAL]"),
333                      cl::Hidden, cl::init(0));
```

Address Sanitizer детали реализации
поддержка компилятором

LLVM (AddressSanitizer.cpp)

```
326  static cl::opt<int> ClMappingScale("asan-mapping-scale",
327                                     cl::desc("scale of asan shadow mapping"),
328                                     cl::Hidden, cl::init(0));


449    Mapping.Scale = IsMyriad ? kMyriadShadowScale : kDefaultShadowScale;
450    if (ClMappingScale.getNumOccurrences() > 0) {
451      Mapping.Scale = ClMappingScale;
452    }
```

Address Sanitizer детали реализации

поддержка компилятором

LLVM (AddressSanitizer.cpp)

```
326   static cl::opt<int> ClMappingScale("asan-mapping-scale",
327                                      cl::desc("scale of asan shadow mapping"),
328                                      cl::Hidden, cl::init(0));
```

Попробуем применить

Address Sanitizer детали реализации
поддержка компилятором

Было:

```
1  #include <cstdint>
2  void foo(uint64_t* a) {
3      *a = 42;
4  }
```

```
1  foo(unsigned long*):
2          push    rax
3          mov     rax, rdi
4          shr     rax, 3
5          cmp     byte ptr [rax + 2147450880], 0
6          jne     .LBB0_1
7          mov     qword ptr [rdi], 42
8          pop     rax
9          ret
```

Address Sanitizer детали реализации
поддержка компилятором

Стало: -mllvm -asan-mapping-scale=2

```
1  #include <cstdint>
2  void foo(uint64_t* a) {
3      *a = 42;
4  }
```

```
1  foo(unsigned long*):
2          push    rax
3          mov     rax, rdi
4          shr     rax, 2
5          cmp     word ptr [rax + 2147467264], 0
6          jne     .LBB0_1
7          mov     qword ptr [rdi], 42
8          pop     rax
9          ret
```

Address Sanitizer детали реализации
поддержка компилятором

Опция компиляции рантайма:

```
COMPILER_RT_ASAN_SHADOW_SCALE=2
```

Address Sanitizer детали реализации защита стека

Соберем без ASan

```
1  #include <cstdint>
2  void foo() {
3      uint64_t a = 3;
4  }
```

Address Sanitizer детали реализации защита стека

Соберем без ASan

```
1  #include <cstdint>
2  void foo() {
3      uint64_t a = 3;
4  }
```

```
1  foo():
2          push    rbp
3          mov     rbp, rsp
4          mov     QWORD PTR [rbp-8], 3
5          nop
6          pop     rbp
7          ret
```

Address Sanitizer детали реализации защита стека

Соберем с ASan: -fsanitize=address

```
1  #include <cstdint>
2  void foo() {
3      uint64_t a = 3;
4  }
```

```
1  foo():
2          push    rbp
3          mov     rbp, rsp
4          mov     QWORD PTR [rbp-8], 3
5          nop
6          pop     rbp
7          ret
```

Address Sanitizer детали реализации защита стека

Соберем с ASan: -fsanitize=address, добавим volatile

```
1  #include <cstdint>
2  void foo() {
3      volatile uint64_t a = 3;
4  }
```

5 инструкций превратились в 36

```
3   foo():
4   .LASANPC0:
5           push    rbp
6           mov     rbp, rsp
7           push    r12
8           push    rbx
9           sub     rsp, 64
10          lea     rbx, [rbp-80]
11          mov     r12, rbx
12          cmp     DWORD PTR __asan_option_detect_stack_use_after_return[rip], 0
13          je      .L1
14          mov     edi, 64
15          call    __asan_stack_malloc_0
16          test    rax, rax
17          je      .L1
18          mov     rbx, rax
19  .L1:
20          lea     rax, [rbx+64]
21          mov     rdx, rax
22          mov     QWORD PTR [rbx], 1102416563
23          mov     QWORD PTR [rbx+8], OFFSET FLAT:.LC0
24          mov     QWORD PTR [rbx+16], OFFSET FLAT:.LASANPC0
25          mov     rax, rbx
26          shr     rax, 3
27          mov     DWORD PTR [rax+2147450880], -235802127
28          mov     DWORD PTR [rax+2147450884], -202116352
29          mov     QWORD PTR [rdx-32], 3
30          nop
31          cmp     r12, rbx
32          je      .L2
33          mov     QWORD PTR [rbx], 1172321806
34          movabs  rcx, -723401728380766731
35          mov     QWORD PTR [rax+2147450880], rcx
36          jmp     .L3
```

Address Sanitizer детали реализации защита стека

Добавим оптимизацию: -O2 -fsanitize=address

```
1  #include <cstdint>
2  void foo() {
3      volatile uint64_t a = 3;
4  }
```

36 => 31 инструкция

```
3   foo():
4   .LASANPC0:
5           push    rbp
6           push    rbx
7           sub     rsp, 72
8           mov     eax, DWORD PTR __asan_option_detect_stack_use_after_return[rip]
9           mov     rbx, rsp
10          mov     rbp, rbx
11          test    eax, eax
12          jne     .L7
13  .L1:
14          mov     rax, rbx
15          mov     QWORD PTR [rbx], 1102416563
16          shr     rax, 3
17          mov     QWORD PTR [rbx+8], OFFSET FLAT:.LC0
18          mov     QWORD PTR [rbx+16], OFFSET FLAT:.LASANPC0
19          mov     DWORD PTR [rax+2147450880], -235802127
20          mov     DWORD PTR [rax+2147450884], -202116352
21          mov     QWORD PTR [rbx+32], 3
22          cmp     rbp, rbx
23          jne     .L8
24          mov     QWORD PTR [rax+2147450880], 0
25  .L3:
26          add     rsp, 72
27          pop     rbx
28          pop     rbp
29          ret
30  .L7:
31          mov     edi, 64
32          call    __asan_stack_malloc_0
33          test    rax, rax
34          cmovne  rbx, rax
35          jmp     .L1
36  .L8:
37          movabs  rcx, -7234017283807667 31
38          mov     QWORD PTR [rbx], 1172321806
```

Address Sanitizer детали реализации защита стека

Уберем volatile, добавим вызов функции

```
1  #include <cstdint>
2  void bar(uint64_t&);
3  void foo() {
4      uint64_t a = 3;
5      bar(a);
6  }
7
```

```
3   foo():
4   .LASANPC0:
5           push    r12
6           push    rbp
7           push    rbx
8           sub     rsp, 64
9           mov     eax, DWORD PTR __asan_option_detect_stack_use_after_return[rip]
10          mov     rbx, rsp
11          mov     r12, rbx
12          test    eax, eax
13          jne     .L8
14  .L1:
15          lea     rdi, [rbx+32]
16          mov     rbp, rbx
17          mov     QWORD PTR [rbx], 1102416563
18          lea     rax, [rbx+64]
19          mov     rdx, rdi
20          shr     rbp, 3
21          mov     QWORD PTR [rbx+8], OFFSET FLAT:.LC0
22          shr     rdx, 3
23          mov     QWORD PTR [rbx+16], OFFSET FLAT:.LASANPC0
24          mov     DWORD PTR [rbp+2147450880], -235802127
25          mov     DWORD PTR [rbp+2147450884], -202116352
26          cmp     BYTE PTR [rdx+2147450880], 0
27          jne     .L9
28          mov     QWORD PTR [rax-32], 3
29          call    bar(unsigned long&)
30          cmp     r12, rbx
31          jne     .L10
32          mov     QWORD PTR [rbp+2147450880], 0
33  .L3:
34          add     rsp, 64
35          pop     rbx
36          pop     rbp
37          pop     r12
38          ret
39  .L8:
40          mov     edi, 64
41          call    __asan_stack_malloc_0
42          test    rax, rax
43          cmovne  rbx, rax
44          jmp     .L1
```

Address Sanitizer детали реализации защита стека

В ASan используется escape analysis

Address Sanitizer детали реализации защита стека

В ASan используется escape analysis

Определяет может ли переменная “сбежать” из текущей области видимости

Address Sanitizer детали реализации защита стека

В ASan используется escape analysis

Определяет может ли переменная “сбежать” из текущей области видимости

Escape analysis используется для оптимизации, работает даже при -O0

Address Sanitizer детали реализации защита стека

Нет “сбежавших” переменных - больше безопасность и производительность

Address Sanitizer детали реализации: runtime

Address Sanitizer детали реализации: runtime

Нужно перехватывать все вызовы malloc, free, new, delete…

Address Sanitizer детали реализации: runtime INTERCEPTORS

```
2  int main() {
3  }
```

a.out

```
linux-vdso.so.1 (0x00007ffe8
libstdc++.so.6 => /opt/compi
libm.so.6 => /lib/x86_64-lir
libgcc_s.so.1 => /opt/compil
libc.so.6 => /lib/x86_64-lir
/lib64/ld-linux-x86-64.so.2
```

Address Sanitizer детали реализации: runtime INTERCEPTORS

```
2  int main() {
3  }
```

a.out

```
linux-vdso.so.1 (0x00007ffe8
libstdc++.so.6 => /opt/compi
libm.so.6 => /lib/x86_64-lir
libgcc_s.so.1 => /opt/compil
libc.so.6 => /lib/x86_64-lir
/lib64/ld-linux-x86-64.so.2
```

Address Sanitizer детали реализации: runtime INTERCEPTORS

```
2  int main() {
3  }
```

a.out

```
linux-vdso.so.1 (0x00007ffe8
libstdc++.so.6 => /opt/compi
libm.so.6 => /lib/x86_64-lir
libgcc_s.so.1 => /opt/compil
libc.so.6 => /lib/x86_64-lir
/lib64/ld-linux-x86-64.so.2
```

Address Sanitizer детали реализации: runtime INTERCEPTORS

```
2  int main() {
3  }
```

a.out

```
linux-vdso.so.1 (0x00007ffe8
libstdc++.so.6 => /opt/compi
libm.so.6 => /lib/x86_64-lir
libgcc_s.so.1 => /opt/compil
libc.so.6 => /lib/x86_64-lir
/lib64/ld-linux-x86-64.so.2
```

Address Sanitizer детали реализации: runtime INTERCEPTORS

```
2  int main() {
3  }
```

a.out

```
linux-vdso.so.1 (0x00007ffe8
libstdc++.so.6 => /opt/compi
libm.so.6 => /lib/x86_64-li
libgcc_s.so.1 => /opt/compil
libc.so.6 => /lib/x86_64-lir
/lib64/ld-linux-x86-64.so.2
```

Address Sanitizer детали реализации: runtime INTERCEPTORS

```
2  int main() {
3  }
```

a.out

```
linux-vdso.so.1 (0x00007ffe8
libstdc++.so.6 => /opt/compi
libm.so.6 => /lib/x86_64-lir
libgcc_s.so.1 => /opt/compil
libc.so.6 => /lib/x86_64-lir
/lib64/ld-linux-x86-64.so.2
```

Address Sanitizer детали реализации: runtime INTERCEPTORS

```
2  int main() {
3  }
```

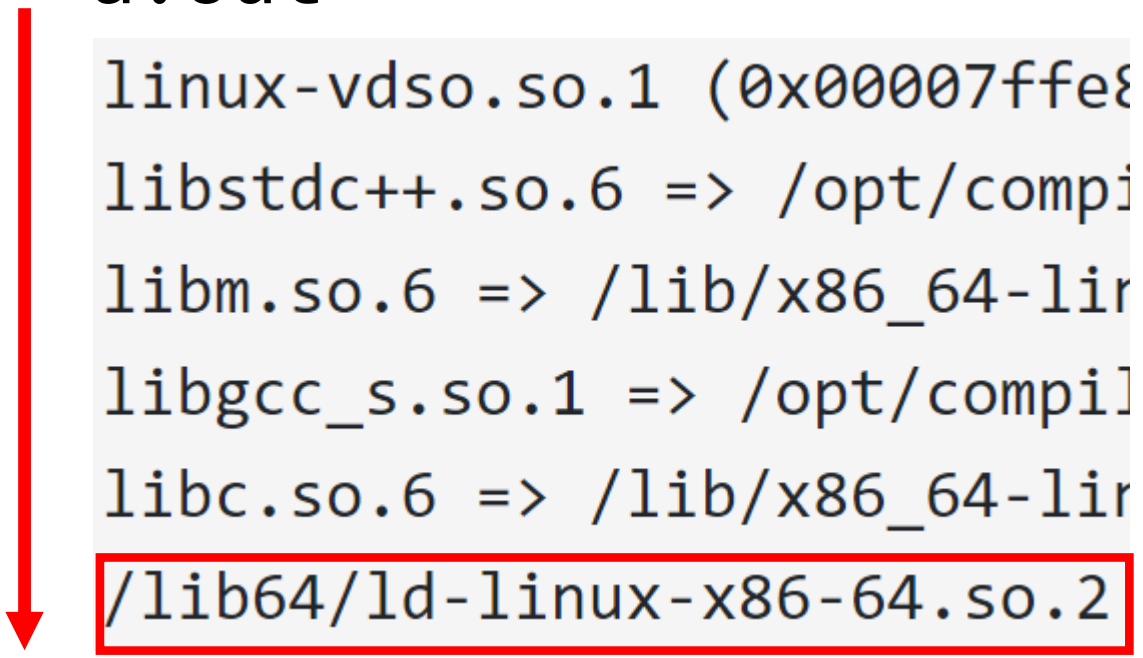
a.out

```
linux-vdso.so.1 (0x00007ffe8
libstdc++.so.6 => /opt/compi
libm.so.6 => /lib/x86_64-lir
libgcc_s.so.1 => /opt/compil
libc.so.6 => /lib/x86_64-li
/lib64/ld-linux-x86-64.so.2
```

Address Sanitizer детали реализации: runtime INTERCEPTORS

```
2  int main() {
3  }
```

a.out



Address Sanitizer детали реализации: runtime INTERCEPTORS

```
1  #include <cstdlib>
2  #include <cstdio>
3  void* malloc(size_t) {
4      puts("hello\n");
5      return nullptr;
6  }
7  int main() {
8      malloc(10);
9  }
```

a.out

```
linux-vdso.so.1 (0x00007ffe8
libstdc++.so.6 => /opt/compi
libm.so.6 => /lib/x86_64-lir
libgcc_s.so.1 => /opt/compil
libc.so.6 => /lib/x86_64-lir
/lib64/ld-linux-x86-64.so.2
```

```
$ ./a.out
hello
```

Address Sanitizer детали реализации: runtime INTERCEPTORS

```
3   #include <dlfcn.h>
4   using real_malloc_t = void*(*)(size_t);
5   void* malloc(size_t arg) {
6       puts("hello");
7       void* func = dlsym(RTLD_NEXT, "malloc");
8       auto real_malloc = real_malloc_t(func);
9       return real_malloc(arg);
10  }
11  int main() {
12      malloc(10);
13  }
```

a.out

linux-vdso.so.1
libdl.so.2
libstdc++.so.6
libm.so.6
libgcc_s.so.1
/lib64/ld-linux-x86-64.so.2

Address Sanitizer детали реализации: runtime INTERCEPTORS

https://github.com/llvm/llvm-project/blob/master/compiler-rt/lib/interception/interception_linux.cpp

```
59  bool InterceptFunction(const char *name, uptr *ptr_to_real, uptr func,
60                         uptr wrapper) {
61    void *addr = GetFuncAddr(name, wrapper);
62    *ptr_to_real = (uptr)addr;
63    return addr && (func == wrapper);
64  }
```

Address Sanitizer детали реализации: runtime INTERCEPTORS

https://github.com/llvm/llvm-project/blob/master/compiler-rt/lib/interception/interception_linux.cpp

```
59  bool InterceptFunction(const char *name, uptr *ptr_to_real, uptr func,
60                         uptr wrapper) {
61    void *addr = GetFuncAddr(name, wrapper);
62    *ptr_to_real = (uptr)addr;
63    return addr && (func == wrapper);
64  }
```

Address Sanitizer детали реализации: runtime INTERCEPTORS

https://github.com/llvm/llvm-project/blob/master/compiler-rt/lib/interception/interception_linux.cpp

```
59  bool InterceptFunction(const char *name, uptr *ptr_to_real, uptr func,
60                         uptr wrapper) {
61    void *addr = GetFuncAddr(name, wrapper);
62    *ptr_to_real = (uptr)addr;
63    return addr && (func == wrapper);
64  }
```

Address Sanitizer детали реализации: runtime INTERCEPTORS

https://github.com/llvm/llvm-project/blob/master/compiler-rt/lib/interception/interception_linux.cpp

```
59  bool InterceptFunction(const char *name, uptr *ptr_to_real, uptr func,
60                         uptr wrapper) {
61    void *addr = GetFuncAddr(name, wrapper);
62    *ptr_to_real = (uptr)addr;
63    return addr && (func == wrapper);
64  }
```

Address Sanitizer детали реализации: runtime INTERCEPTORS

https://github.com/llvm/llvm-project/blob/master/compiler-rt/lib/interception/interception_linux.cpp

```
59  bool InterceptFunction(const char *name, uptr *ptr_to_real, uptr func,
60                         uptr wrapper) {
61    void *addr = GetFuncAddr(name, wrapper);
62    *ptr_to_real = (uptr)addr;
63    return addr && (func == wrapper);
64  }
```

Address Sanitizer детали реализации: runtime INTERCEPTORS

https://github.com/llvm/llvm-project/blob/master/compiler-rt/lib/interception/interception_linux.cpp

```
36   static void *GetFuncAddr(const char *name, uptr wrapper_addr) {
42     void *addr = dlsym(RTLD_NEXT, name);
43     if (!addr) {
49       addr = dlsym(RTLD_DEFAULT, name);
53       if ((uptr)addr == wrapper_addr)
54         addr = nullptr;
55     }
56     return addr;
57   }
```

Address Sanitizer детали реализации: runtime INTERCEPTORS

https://github.com/llvm/llvm-project/blob/master/compiler-rt/lib/interception/interception_linux.cpp

```
36   static void *GetFuncAddr(const char *name, uptr wrapper_addr) {
42     void *addr = dlsym(RTLD_NEXT, name);
43     if (!addr) {
49       addr = dlsym(RTLD_DEFAULT, name);
53       if ((uptr)addr == wrapper_addr)
54         addr = nullptr;
55     }
56     return addr;
57   }
```

Address Sanitizer детали реализации: runtime INTERCEPTORS

https://github.com/llvm/llvm-project/blob/master/compiler-rt/lib/interception/interception_linux.cpp

```
36   static void *GetFuncAddr(const char *name, uptr wrapper_addr) {
42     void *addr = dlsym(RTLD_NEXT, name);
43     if (!addr) {
49       addr = dlsym(RTLD_DEFAULT, name);
53       if ((uptr)addr == wrapper_addr)
54         addr = nullptr;
55     }
56     return addr;
57   }
```

Address Sanitizer детали реализации: runtime INTERCEPTORS

https://github.com/llvm/llvm-project/blob/master/compiler-rt/lib/interception/interception_linux.cpp

```
36   static void *GetFuncAddr(const char *name, uptr wrapper_addr) {
42     void *addr = dlsym(RTLD_NEXT, name);
43     if (!addr) {
49       addr = dlsym(RTLD_DEFAULT, name);
53       if ((uptr)addr == wrapper_addr)
54         addr = nullptr;
55     }
56     return addr;
57   }
```

Address Sanitizer детали реализации: runtime INTERCEPTORS

https://github.com/llvm/llvm-project/blob/master/compiler-rt/lib/interception/interception_linux.cpp

```
36   static void *GetFuncAddr(const char *name, uptr wrapper_addr) {
42     void *addr = dlsym(RTLD_NEXT, name);
43     if (!addr) {
49       addr = dlsym(RTLD_DEFAULT, name);
53       if ((uptr)addr == wrapper_addr)
54         addr = nullptr;
55     }
56     return addr;
57   }
```

Address Sanitizer детали реализации: runtime INTERCEPTORS

https://github.com/llvm/llvm-project/blob/master/compiler-rt/lib/interception/interception_linux.cpp

```
36   static void *GetFuncAddr(const char *name, uptr wrapper_addr) {
42     void *addr = dlsym(RTLD_NEXT, name);
43     if (!addr) {
49       addr = dlsym(RTLD_DEFAULT, name);
53       if ((uptr)addr == wrapper_addr)
54         addr = nullptr;
55     }
56     return addr;
57   }
```

Address Sanitizer детали реализации: runtime INTERCEPTORS

https://github.com/llvm/llvm-project/blob/master/compiler-rt/lib/interception/interception_linux.cpp

```
36   static void *GetFuncAddr(const char *name, uptr wrapper_addr) {
42     void *addr = dlsym(RTLD_NEXT, name);
43     if (!addr) {
49       addr = dlsym(RTLD_DEFAULT, name);
53       if ((uptr)addr == wrapper_addr)
54         addr = nullptr;
55     }
56     return addr;
57   }
```

Address Sanitizer детали реализации: runtime INTERCEPTORS

Перехватываются malloc, free, new, delete, стандартные posix-функции, printf & co, strcmp & co, swapcontext, __cxa_throw, pthread_create, pthread_join, etc.

Address Sanitizer детали реализации: runtime INTERCEPTORS

Перехватываются malloc, free, new, delete, стандартные posix-функции, printf & co, strcmp & co, swapcontext, __cxa_throw, pthread_create, pthread_join, etc.

ASan не перехватывает mmap.

Address Sanitizer детали реализации: runtime

```
1  #include <sys/mman.h>
2  int main() {
3          auto addr = mmap(nullptr, 4096,
4                           PROT_READ|PROT_WRITE,
5                           MAP_PRIVATE|MAP_ANONYMOUS, -1, 0);
6          char* str = (char*)addr;
7          str[0] = 42;
8  }
```

Address Sanitizer детали реализации: runtime

В shadow memory по умолчанию нули. ASan никак не видит эту память => нет false positive

```
1  #include <sys/mman.h>
2  int main() {
3          auto addr = mmap(nullptr, 4096,
4                           PROT_READ|PROT_WRITE,
5                           MAP_PRIVATE|MAP_ANONYMOUS, -1, 0);
6          char* str = (char*)addr;
7          str[0] = 42;
8  }
```

Address Sanitizer детали реализации: runtime

Сделаем, чтобы увидел

Address Sanitizer детали реализации: runtime

Сделаем, чтобы увидел

```
sanitizer/asan_interface.h

ASAN_POISON_MEMORY_REGION(addr, size)
ASAN_UNPOISON_MEMORY_REGION(addr, size)
```

Address Sanitizer детали реализации: runtime

Сделаем, чтобы увидел

```
1   #include <sys/mman.h>
2   #include <sanitizer/asan_interface.h>
3   int main() {
4       auto addr = mmap(nullptr, 4096,
5                        PROT_READ|PROT_WRITE,
6                        MAP_PRIVATE|MAP_ANONYMOUS, -1, 0);
7       ASAN_POISON_MEMORY_REGION(addr, 8);
8       char* str = (char*)addr;
9       str[0] = 42;
10  }
```

Address Sanitizer детали реализации: runtime

Сделаем, чтобы увидел

```
WRITE of size 1 at 0x7f9c68bda000 thread T0
    #0 0x400a01 in main (/home/build/Projects/san/a.out+0x400a01)
    #1 0x7f9c66e4e494 in __libc_start_main (/lib64/libc.so.6+0x22494)
    #2 0x400818  (/home/build/Projects/san/a.out+0x400818)

Address 0x7f9c68bda000 is a wild pointer.
SUMMARY: AddressSanitizer: use-after-poison
```

Address Sanitizer детали реализации: runtime

Сделаем, чтобы увидел

```
Shadow bytes around the buggy address:
  0x0ff40d1733b0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0ff40d1733c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0ff40d1733d0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0ff40d1733e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0ff40d1733f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x0ff40d173400:[f7]00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0ff40d173410: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0ff40d173420: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0ff40d173430: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0ff40d173440: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0ff40d173450: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

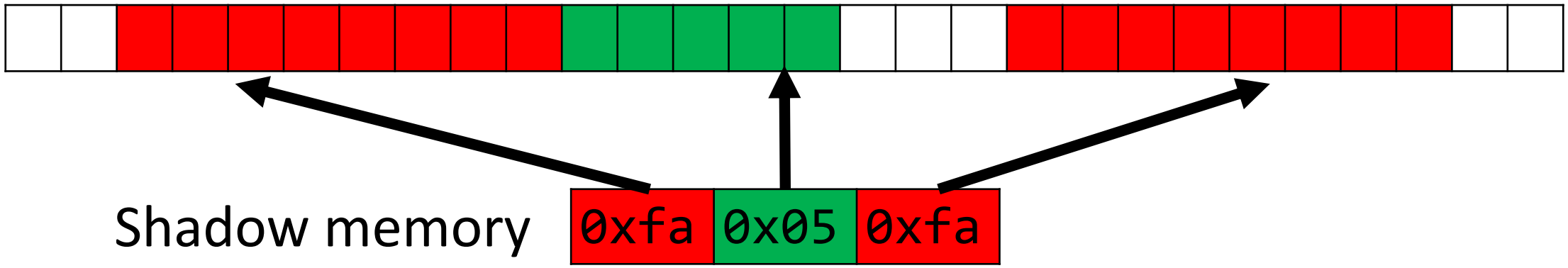
Address Sanitizer детали реализации: runtime

Сделаем, чтобы увидел

```
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
  ASan internal:           fe
  Left alloca redzone:     ca
  Right alloca redzone:    cb
```

Address Sanitizer детали реализации: runtime

- гарантирует выравнивание на 8
- добавляет red zone вокруг аллокаций



Address Sanitizer пример из жизни

Address Sanitizer пример из жизни

```
1   struct DataHolder {
2       auto beginDict() {return dict.begin();}
3       void deleteWord(const std::string& key) {
4           auto n = dict.erase(key);
5           std::cerr << "key '" << key  << "' "
6                     << n << " elements was deleted.\n";
7       }
8       std::map<std::string, std::string> dict;
9   };
10
11  dh.deleteWord(dh.beginDict()->first);
```

Address Sanitizer пример из жизни

```
1   struct DataHolder {
2       auto beginDict() {return dict.begin();}
3       void deleteWord(const std::string& key) {
4           auto n = dict.erase(key);
5           std::cerr << "key '" << key  << "' "
6                     << n << " elements was deleted.\n";
7       }
8       std::map<std::string, std::string> dict;
9   };
10
11  dh.deleteWord(dh.beginDict()->first);
```

Address Sanitizer пример из жизни

```
1   struct DataHolder {
2       auto beginDict() {return dict.begin();}
3       void deleteWord(const std::string& key) {
4           auto n = dict.erase(key);
5           std::cerr << "key '" << key  << "' "
6                     << n << " elements was deleted.\n";
7       }
8       std::map<std::string, std::string> dict;
9   };
10
11  dh.deleteWord(dh.beginDict()->first);
```

Address Sanitizer пример из жизни

```
1   struct DataHolder {
2       auto beginDict() {return dict.begin();}
3       void deleteWord(const std::string& key) {
4           auto n = dict.erase(key);
5           std::cerr << "key '" << key  << "' "
6                     << n << " elements was deleted.\n";
7       }
8       std::map<std::string, std::string> dict;
9   };
10
11  dh.deleteWord(dh.beginDict()->first);
```

Address Sanitizer пример из жизни

```
1   struct DataHolder {
2       auto beginDict() {return dict.begin();}
3       void deleteWord(const std::string& key) {
4           auto n = dict.erase(key);
5           std::cerr << "key '" << key  << "' "
6                     << n << " elements was deleted.\n";
7       }
8       std::map<std::string, std::string> dict;
9   };
10
11  dh.deleteWord(dh.beginDict()->first);
```

Address Sanitizer пример из жизни

```
1   struct DataHolder {
2       auto beginDict() {return dict.begin();}
3       void deleteWord(const std::string& key) {
4           auto n = dict.erase(key);
5           std::cerr << "key '" << key  << "' "
6                     << n << " elements was deleted.\n";
7       }
8       std::map<std::string, std::string> dict;
9   };
10
11  dh.deleteWord(dh.beginDict()->first);
```

Address Sanitizer пример из жизни

```
$ g++ --fsanitize=address main.cpp
$ ./a.out
for key 'ab
```

Address Sanitizer пример из жизни

```
$ g++ --fsanitize=address main.cpp
$ ./a.out
for key 'ab
```

O_o

WAT?!

Address Sanitizer пример из жизни

```
$ g++ --fsanitize=address -O2 main.cpp
$ ./a.out
==40383==ERROR: AddressSanitizer:
heap-use-after-free
READ of size 8 at 0x604000000030 thread T0
    #0 0x402a56 in main
    #1 0x7f67d408f494 in
    #2 0x402c39
```

Address Sanitizer пример из жизни

```
$ g++ --fsanitize=address -O2 main.cpp
$ ./a.out
==40383==ERROR: AddressSanitizer:
heap-use-after-free
READ of size 8 at 0x604000000030 thread T0
    #0 0x402a56 in main
    #1 0x7f67d408f494 in
    #2 0x402c39
```

O_o

WAT?!

O_O

WAT?!

Address Sanitizer пример из жизни

```
1   struct DataHolder {
2       auto beginDict() {return dict.begin();}
3       void deleteWord(const std::string& key) {
4           auto n = dict.erase(key);
5           printf("%s\n", key.c_str());
6       }
7       std::map<std::string, std::string> dict;
8   };
9
10  dh.deleteWord(dh.beginDict()->first);
```

Address Sanitizer пример из жизни

```
$ g++ --fsanitize=address main.cpp
$ ./a.out
ab
$ g++ --fsanitize=address main.cpp -O2
$ ./a.out
==44382==ERROR: AddressSanitizer:
heap-use-after-free
on address 0x604000000030
```

Address Sanitizer пример из жизни

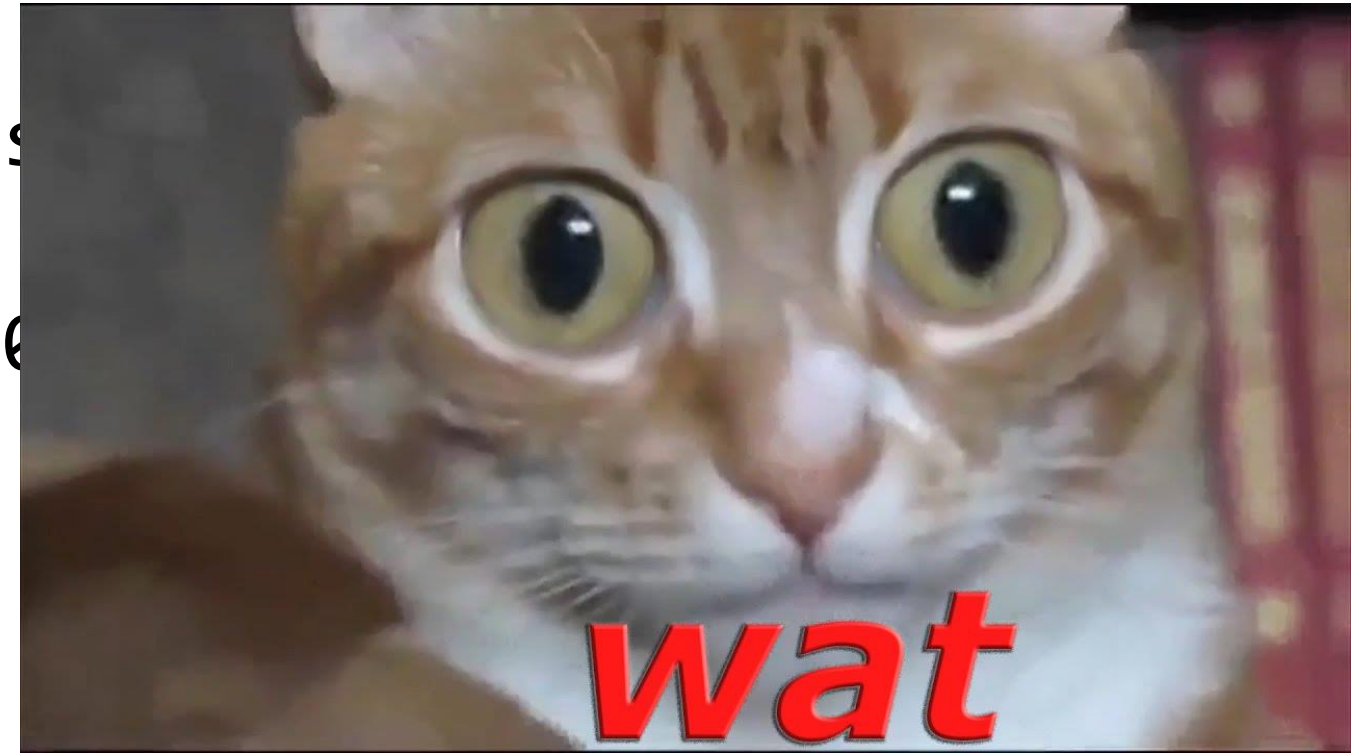
```
1   struct DataHolder {
2       auto beginDict() {return dict.begin();}
3       void deleteWord(const std::string& key) {
4           auto n = dict.erase(key);
5           printf("for key %s ...", key.c_str());
6       }
7       std::map<std::string, std::string> dict;
8   };
9
10  dh.deleteWord(dh.beginDict()->first);
```

Address Sanitizer пример из жизни

```
$ g++ --fsanitize=address main.cpp
$ ./a.out
==44496==ERROR: AddressSanitizer:
heap-use-after-free
on address 0x603000000088
```

Address Sanitizer пример из жизни

```
$ g++ --fsanitize=address main.cpp
$ ./a.out
==44496==ERROR: Addres
heap-use-after-free
on address 0x603000000
```



Address Sanitizer пример из жизни

```
1  #include <stdio.h>
2  void foo() {
3      printf("%s\n", "hello");
4  }
```

```
3  foo():
4          push    rbp
5          mov     rbp, rsp
6          mov     edi, OFFSET FLAT:.LC0
7          call    puts
8          nop
9          pop     rbp
10         ret
```

Address Sanitizer пример из жизни

```
1  #include <stdio.h>
2  void foo() {
3      printf("%s\n", "hello");
4  }
```

```
3  foo():
4          push    rbp
5          mov     rbp, rsp
6          mov     edi, OFFSET FLAT:.LC0
7          call    puts
8          nop
9          pop     rbp
10         ret
```

GCC: gimple-fold.c

Функция: gimple_fold_builtin_printf

Address Sanitizer пример из жизни

```
1   #include <string>
2   const char* foo(std::string& s) {
3       return s.c_str();
4   }
```

--fsanitize=address

```
4   foo(std::__cxx11::basic_string<char, std::
5           push    rbp
6           mov     rbp, rsp
7           sub     rsp, 16
8           mov     QWORD PTR [rbp-8], rdi
9           mov     rax, QWORD PTR [rbp-8]
10          mov     rdi, rax
11          call    <char> >::c_str() const
12          leave
13          ret
```

Address Sanitizer пример из жизни

```
1  #include <string>
2  const char* foo(std::string& s) {
3      return s.c_str();
4  }
```

```
1  foo(std::__cxx11::basic_string<char, std::ch
2          mov     rax, rdi
3          shr     rax, 3
4          cmp     BYTE PTR [rax+2147450880], 0
5          jne     .L7
6          mov     rax, QWORD PTR [rdi]
7          ret
```

```
--fsanitize=address -O2
```

Address Sanitizer пример из жизни

```
1  #include <string>
2  const char* foo(std::string& s) {
3      return s.c_str();
4  }
```

```
--fsanitize=address -O2
```

```
1  foo(std::__cxx11::basic_string<char, std::ch
2          mov     rax, rdi
3          shr     rax, 3
4          cmp     BYTE PTR [rax+2147450880], 0
5          jne     .L7
6          mov     rax, QWORD PTR [rdi]
7          ret
```

Address Sanitizer пример из жизни

bits/basic_string.tcc:1607

```
extern template class basic_string<char>;
```

Address Sanitizer пример из жизни

```
1
2  template <typename T>
3  struct S {
4      void foo(){}
5  };
6
7  extern template struct S<char>;
8
9  int main()
10 {
11     S<char> s;
12     s.foo();
13 }
14
```

-O0

```
1  main:
2          push    rbp
3          mov     rbp, rsp
4          sub     rsp, 16
5          lea     rax, [rbp-1]
6          mov     rdi, rax
7          call    S<char>::foo()
8          mov     eax, 0
9          leave
10         ret
```

-O2

```
1  main:
2          xor     eax, eax
3          ret
```

Спасибо!

Алексей Веселовский

Align Technology

Telegram: @I_vlxy_I
E-mail: aveselovskiy@aligntech.com