Variable Length Array

Так ли страшен черт, как его малюют?

Евгений Ерохин

HyperIntegrate

Немного обо мне

Основатель стартапа HyperIntegrate.

Драйвер файловой системы MTP(Media Transfer Protocol) для macOs.
Который позволяет подключать Android устройства как файловую систему.

Немного обо мне

Старший разработчик в DINS

- Разработка сервиса передачи видео в реальном времени
- Семейство протоколов RTP (Real-time Transport Protocol)
- Реализация протоколов по RFC

Немного обо мне

И еще, Paragon Software 10+ лет ведущий разработчик направления, тимлид.

- Драйвера файловых систем под macOS
- Имплементация файловых систем
- Система снепшотинга блочных (дисковых) устройств
- Бут загрузчик
- Реверс инжиниринг
- И еще оч много всего системного =)

Так что это такое?

VLA или Variable Length Array (Стандарт C99).

Как выглядит код с VLA

```
void fn(int count)
{
    char buf[count];
    ...
}
```

Откуда возник вопрос?

А что думает мировое сообщество?

What's so bad about Variable Length Arrays?

My Intermediate C++ professor marked points off for using variable length arrays in my assignments. I use them because I'm testing out smaller files so if I can get a task down, that requires huge data, it could possibly work for the bigger file.

My professor told me that VLA's are bad because the ISO prohibits it and the VLA allows array lengths to be determined by run-time than compile time.

He also linked me to an article where it explains why https://www.phoronix.com/scan.php?page=news_item&px=Linux-Kills-The-VLA

But I'm still not convinced why I shouldn't use VLA's. Can anyone give me a metaphor or example to help me understand? Thanks.

Edit: I'm currently enrolled in college and am learning this stuff as a CS/SWE major I that helps.

https://www.reddit.com/r/cpp/comments/9t6riy/whats_so_bad_about_variable_length_arrays/

raevnos 91 points · 1 year ago

Not being part of C++ isn't enough?

wutdefukk 19 points · 1 year ago

Potentially Large Arrays in runtime stack ,which is stored little space, can be dangerous.

trvlng_ging 10 points · 1 year ago

What, just because they are antithetical to C++ design goals? They have runtime performance problems that can be quite severe, they are not portable, and they open security holes in your application. There are other things, like they're C and not C++, even in the C world they are considered a mistake. Frankly, I can't see trading all that for minor convenience gains.

Так а что за статья?

The Linux Kernel Is Now VLA-Free: A Win For Security, Less Overhead & Better For Clang

Written by Michael Larabel in Linux Kernel on 29 October 2018 at 12:15 AM EDT. 14 Comments

With the in-development Linux 4.20 kernel, it is now effectively VLA-free... The variable-length arrays (VLAs) that can be convenient and part of the C99 standard but can have unintended consequences.

VLAs allow for array lengths to be determined at run-time rather than compile time. The Linux kernel has long relied upon VLAs in different parts of the kernel -- including within structures -- but going on for months now (and years if counting the kernel Clang'ing efforts) has been to remove the usage of variable-length arrays within the kernel. The problems with them are:

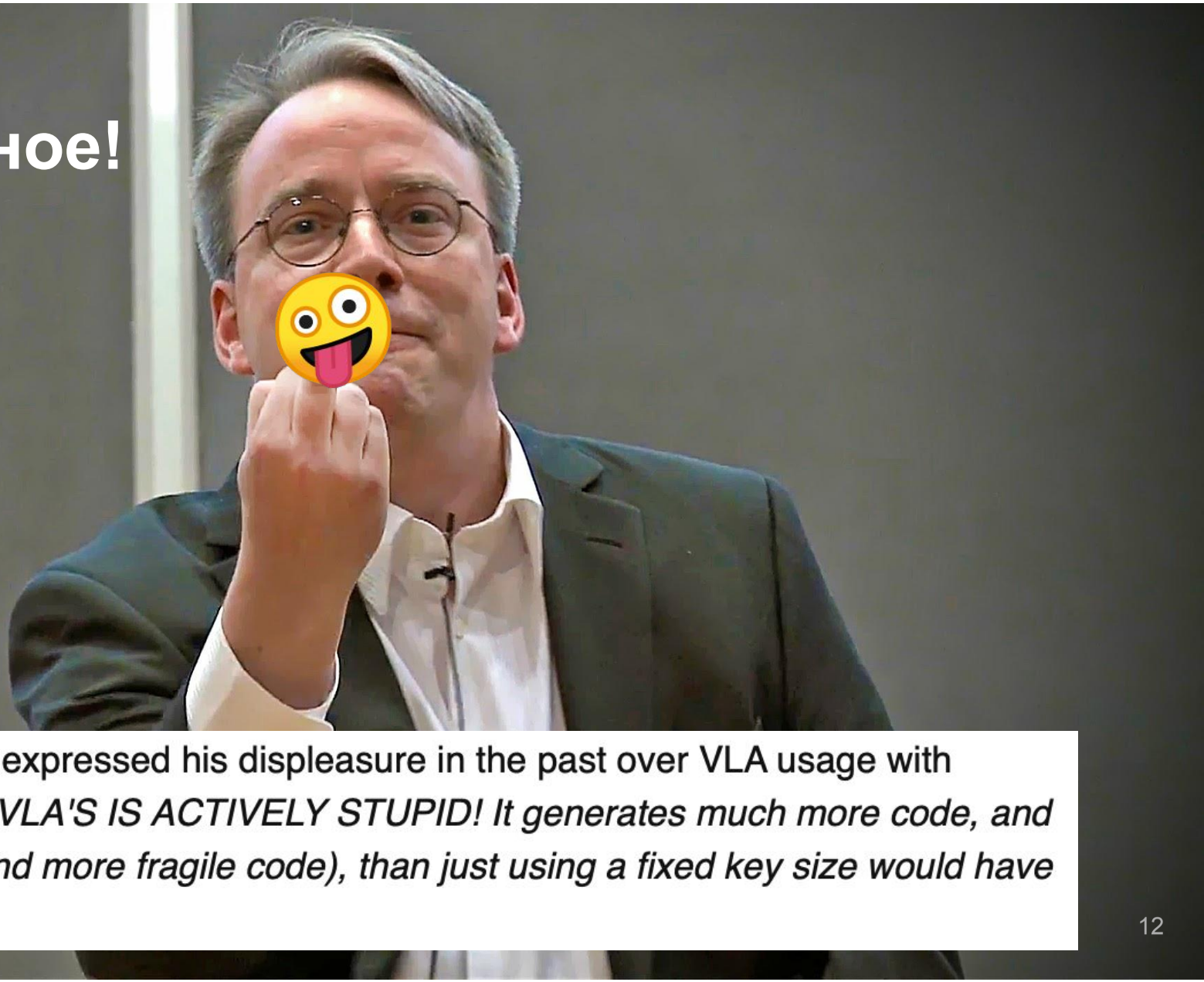
- Using variable-length arrays can add some minor run-time overhead to the code due to needing to determine the size of the array at run-time.

- VLAs within structures is not supported by the LLVM Clang compiler and thus an issue for those wanting to build the kernel outside of GCC, Clang only supports the C99-style VLAs.

- Arguably most importantly is there can be security implications from VLAs around the kernel's stack usage.

https://www.phoronix.com/scan.php?page=news_item&px=Linux-Kills-The-VLA

А самое главное!



ИХ НЕНАВИДИТ

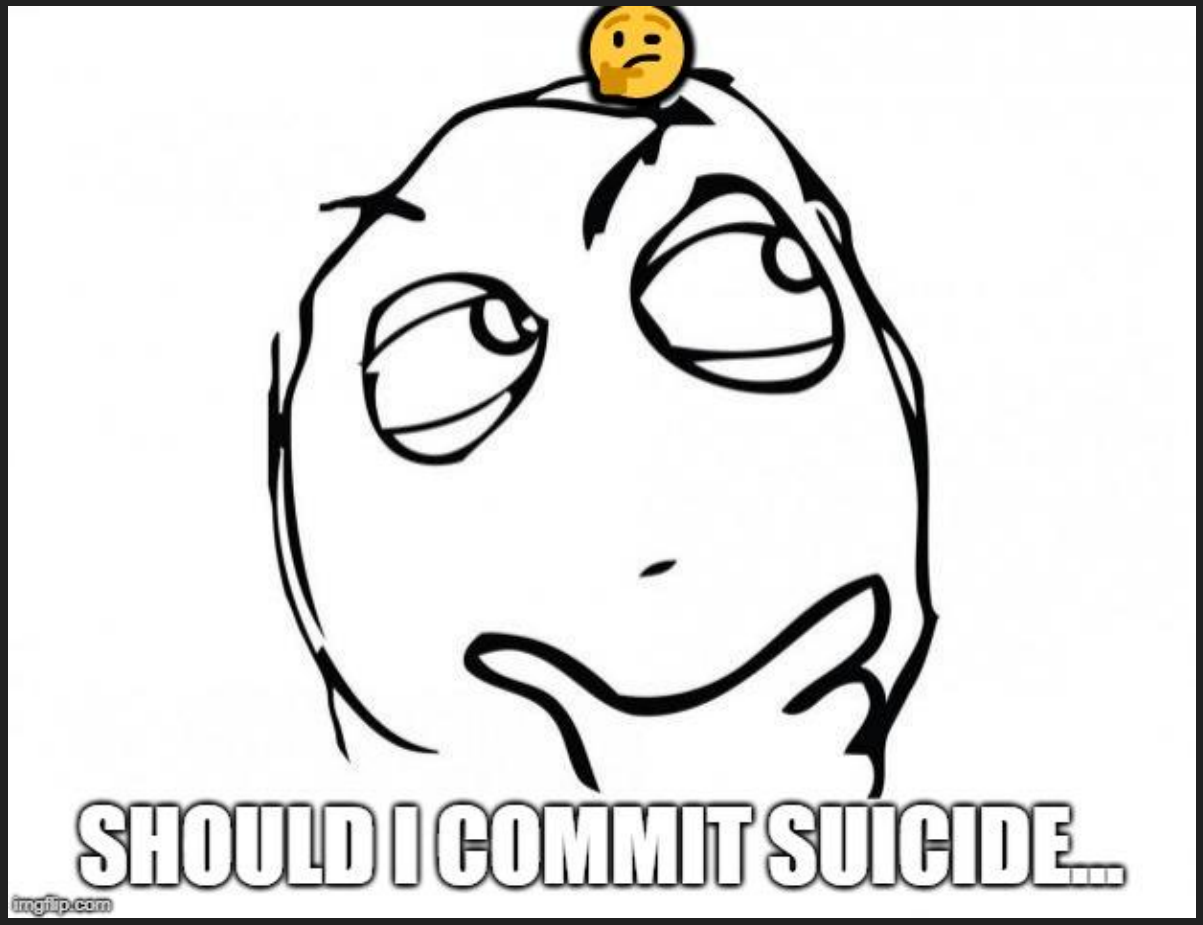
ЛИНУС ТОРВАЛЬДС

Linus Torvalds has also expressed his displeasure in the past over VLA usage with comments like "*USING VLA'S IS ACTIVELY STUPID! It generates much more code, and much _slower_ code (and more fragile code), than just using a fixed key size would have done.*"


И никогда,
никогда!


Не пишите на
C++!

А может не все так плохо?



Где могут пригодиться VLA

- Сборочные буфера
- Массив для получения значений из функций когда количество известно только в рантайме
- И еще много вариантов, когда размер не известен заранее

x86-64-psABI-1.0

An array uses the same alignment as its elements, except that a local or global array variable of length at least 16 bytes or a C99 variable-length array variable always has alignment of at least 16 bytes.[4]

[4]The alignment requirement allows the use of SSE instructions when operating on the array. The compiler cannot in general calculate the size of a variable-length array (VLA), but it is expected that most VLAs will require at least 16 bytes, so it is logical to mandate that VLAs have at least a 16-byte alignment.

Пример

Snippet 1

https://godbolt.org/z/T1Ezj4

А что еще есть в VLA?

1.
```
void
foo (int n)
{
  struct S { int x[n]; };
}
```

2.
```
struct entry
tester (int len, char data[len][len])
{
  /* ... */
}
```

или даже

```
struct entry
tester (int len; char data[len][len], int len)
{
  /* ... */
}
```

https://gcc.gnu.org/onlinedocs/gcc/Variable-Length.html

Как можно решать проблему длины известной в рантайме?

Типичная задача - сборочный буфер для (пакета, структуры для записи на диск и т.п.). Как можно организовать:

- std::vector
- Массив фиксированной длины равной “максимально возможному значению” или std::array
- Массив переменной длины VLA

Говоря о памяти, нельзя обойти 2 абстракции

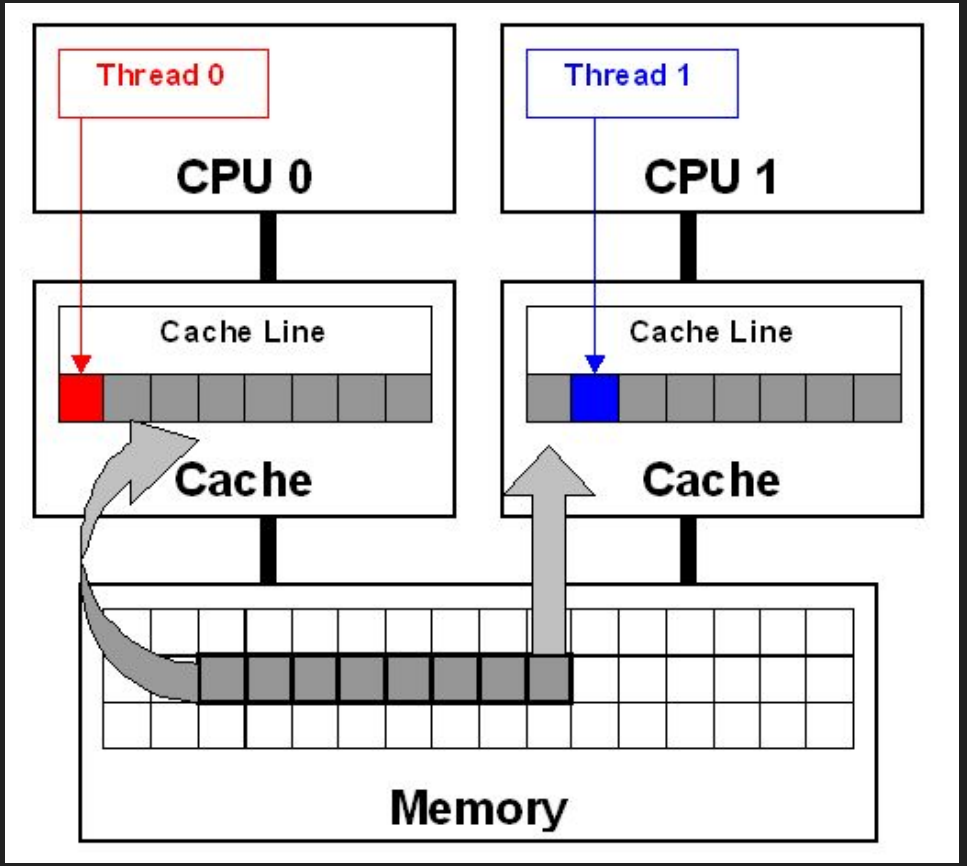
- Кеш
- Стек

Кеш

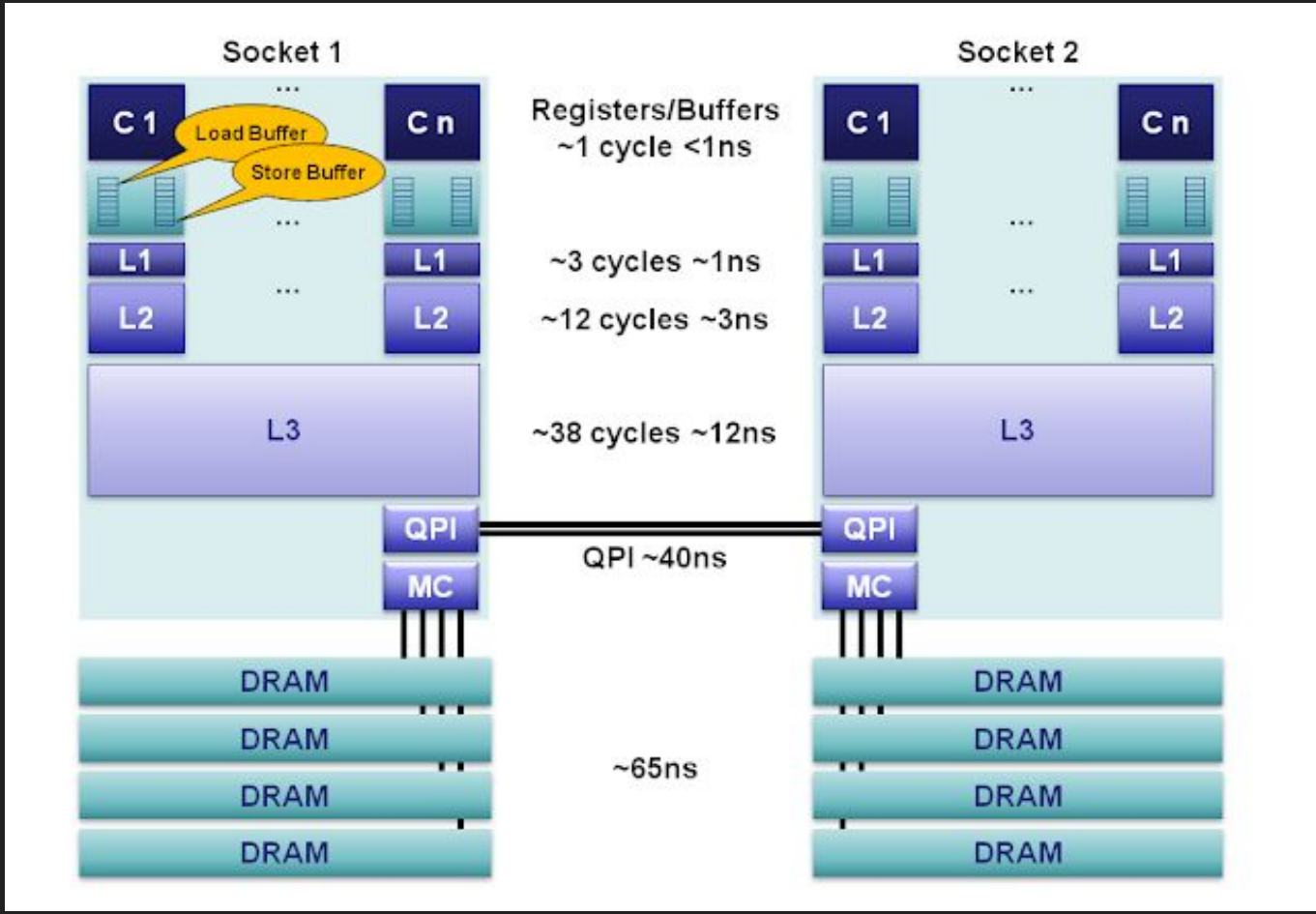
Основное понятие - Cache line.

Длина кратна степени 2.

В основном используется длина 64 байта.



Подсистема памяти

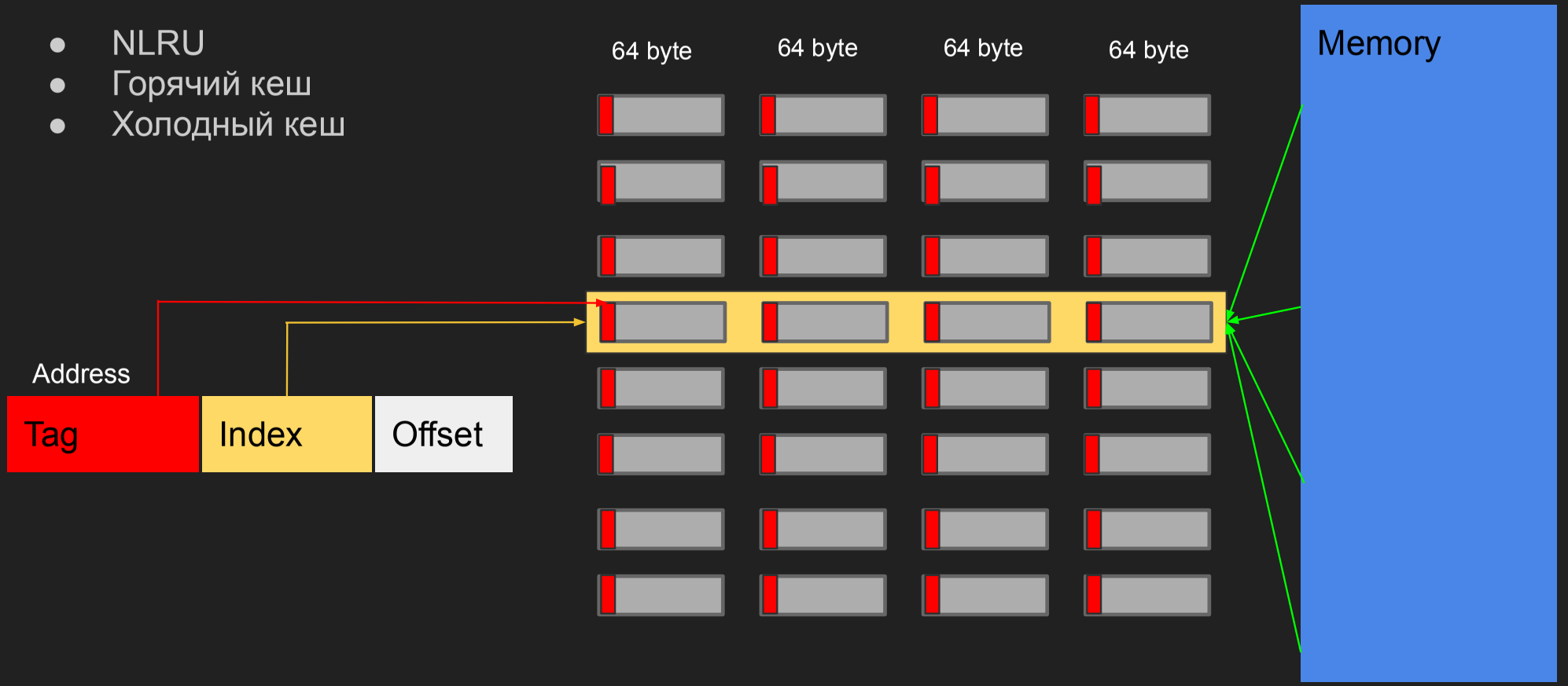


Кеш с реальными цифрами

```
Core i7 Xeon 5500 Series Data Source Latency (approximate)                [Pg. 22]

local  L1 CACHE hit,                                    ~4 cycles (   2.1 -  1.2 ns )
local  L2 CACHE hit,                                   ~10 cycles (   5.3 -  3.0 ns )
local  L3 CACHE hit, line unshared                     ~40 cycles (  21.4 - 12.0 ns )
local  L3 CACHE hit, shared line in another core       ~65 cycles (  34.8 - 19.5 ns )
local  L3 CACHE hit, modified in another core          ~75 cycles (  40.2 - 22.5 ns )

remote L3 CACHE (Ref: Fig.1 [Pg. 5])          ~100-300 cycles ( 160.7 - 30.0 ns )

local  DRAM                                                      ~60 ns
remote DRAM                                                     ~100 ns
```

Set-associative cache

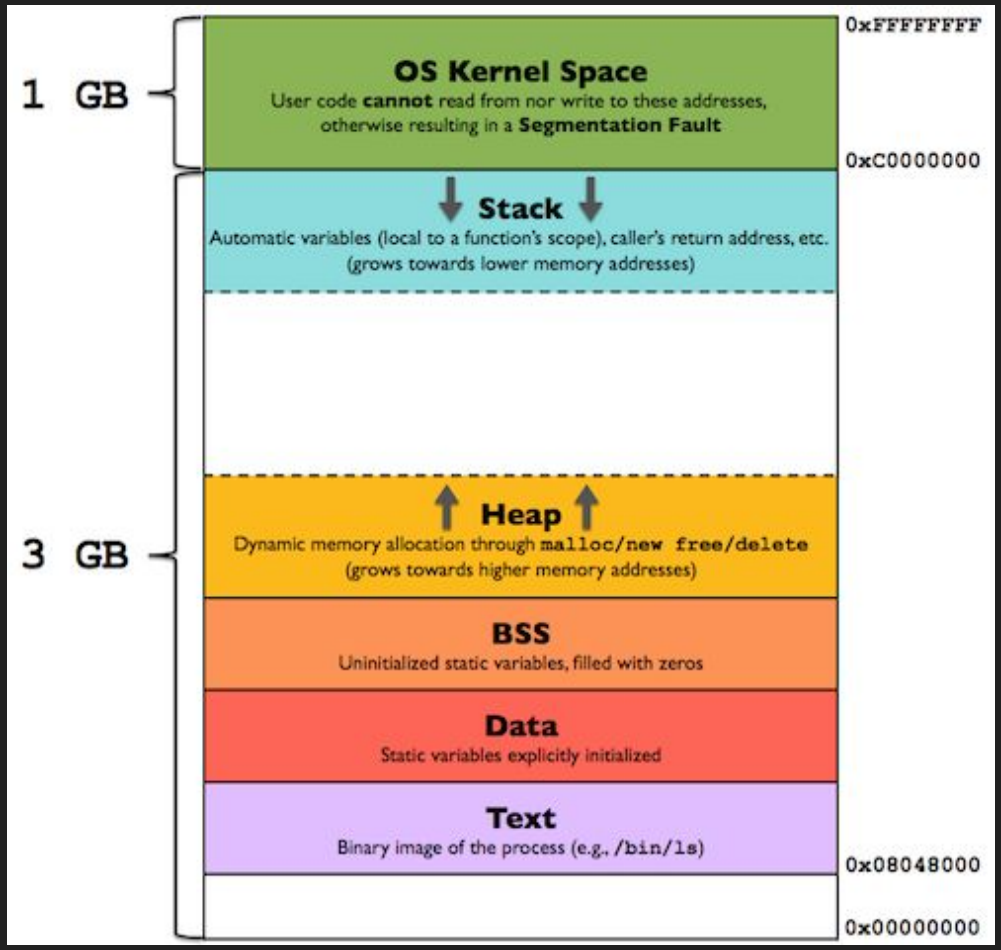
- NLRU
- Горячий кеш
- Холодный кеш



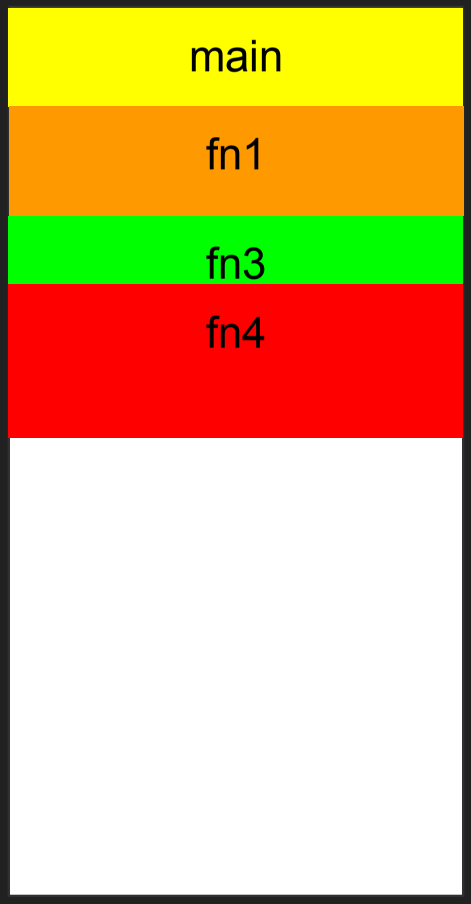
Два ключевых понятия

- Spatial locality
- Temporal locality

Стек


1 GB
3 GB
0xFFFFFFFF
OS Kernel Space
User code cannot read from nor write to these addresses, otherwise resulting in a Segmentation Fault
0xC0000000
Stack
Automatic variables (local to a function's scope), caller's return address, etc.
(grows towards lower memory addresses)
Heap
Dynamic memory allocation through malloc/new free/delete
(grows towards higher memory addresses)
BSS
Uninitialized static variables, filled with zeros
Data
Static variables explicitly initialized
Text
Binary image of the process (e.g., /bin/ls)
0x08048000
0x00000000

Как работает стек

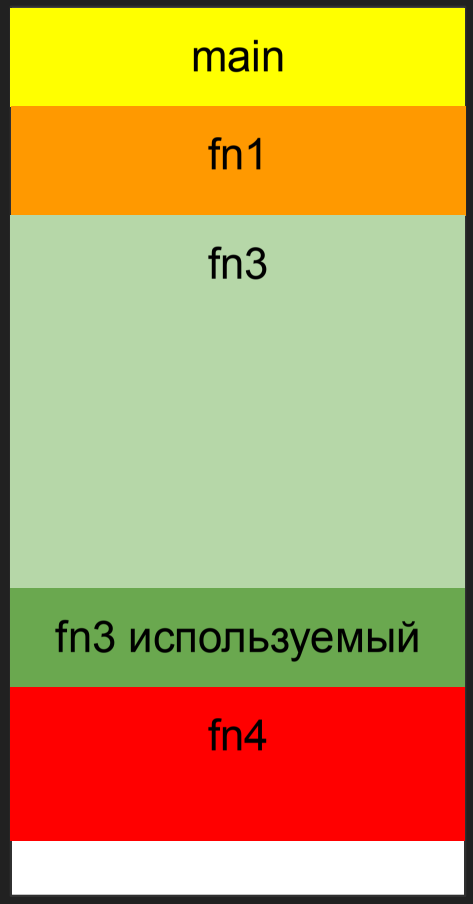


0xc00...000

0x000...000

А как выглядит буфер под максимальный размер

- Где мы теряем производительность?
- А что ожидать от профилировщика?
- Есть ли эти проблемы у VLA?



0xc00...000

0x000...000

А что нам здесь даст VLA?

Большая локальность. И как следствие с большей вероятностью

- работа с буфером самой функции попадает на горячие кеши
- вызовы функций внутри попадают тоже на горячие кеши

Alternatives for Array Extensions (n3810)

Alternatives for Array Extensions

Bjarne Stroustrup

The most important aspect of an “array alternative” is that it enables simple and efficient use of a run-time-specified amount of stack storage and provides an alternative to the traditional problems of “lost” size and “lost” type that make direct use of arrays error-prone.

Недостатки VLA для C++ (n3810)

- В таком виде нельзя использовать с Range-based for loop
- sizeof выдает размер в байтах а не элементах

Все то что послужило мотивацией для std::array.

Альтернативы для VLA (n3810)

- The Array of Run-Time Bounds (ARBs, N3497 Runtime-sized arrays with automatic storage duration by Jens Maurer)
- An class interface for stack memory std::dynarray и std::bs_array

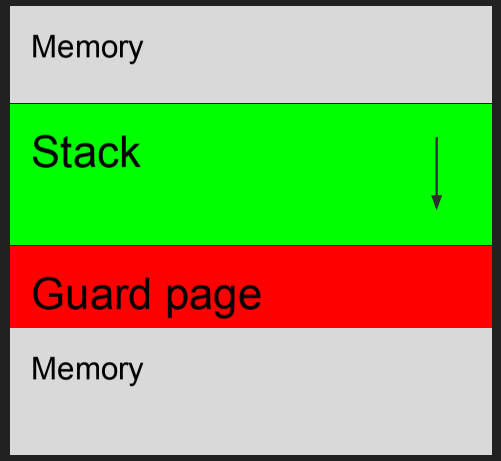
Общая проблема для хранения в стеке

Все методы использования стека имеют общие проблемы:

- Stack clash
- Стек не может расти “бесконечно”

Stack clash

- На самом деле мы не так хорошо контролируем рост стека
- Частично нам помогает Guard page



А что же делать?

Нужна дополнительная магия

- -fstack-check пробация страниц памяти после эллокации (устарела)
- -fstack-clash-protection эллокация постранично с пробацией

https://developers.redhat.com/blog/2020/05/22/stack-clash-mitigation-in-gcc-part-3/

https://godbolt.org/z/3WWGaM

Пример

Snippet 2

https://godbolt.org/z/3WWGaM

А решаема ли проблема с размером стека

По умолчанию рантайм выделяет нам небольшой кусок памяти под стек. И он не увеличивается.

Но можно решить проблему опцией -fsplit-stack. Цена:

- дополнительные проверки на входе в функцию
- сильное увеличение кода
- возможны эллокации

Но стоит ли использовать?

Что можно использовать при отладке

Есть дополнительные флаги для проверки выхода за пределы массива постфактум

-fstack-protector

-fstack-protector-all

-fstack-protector-strong

-fno-stack-protector

Инструментация

Что нам нужно чтобы жить было легко и непринужденно:

- Проверка размера массива на входе
- Если мы хотим все прелести STL, мы всегда можем использовать “паттерн” View std::span, std::string_view, PMR memory resource и т.п.

Так в каких случаях можно ожидать выгоду?

Очевидно что VLA могут давать прирост производительности

- если размер “максимального” буфера будет значительным

И будут скорее всего проигрывать

- если “максимальный” размер относительно маленький

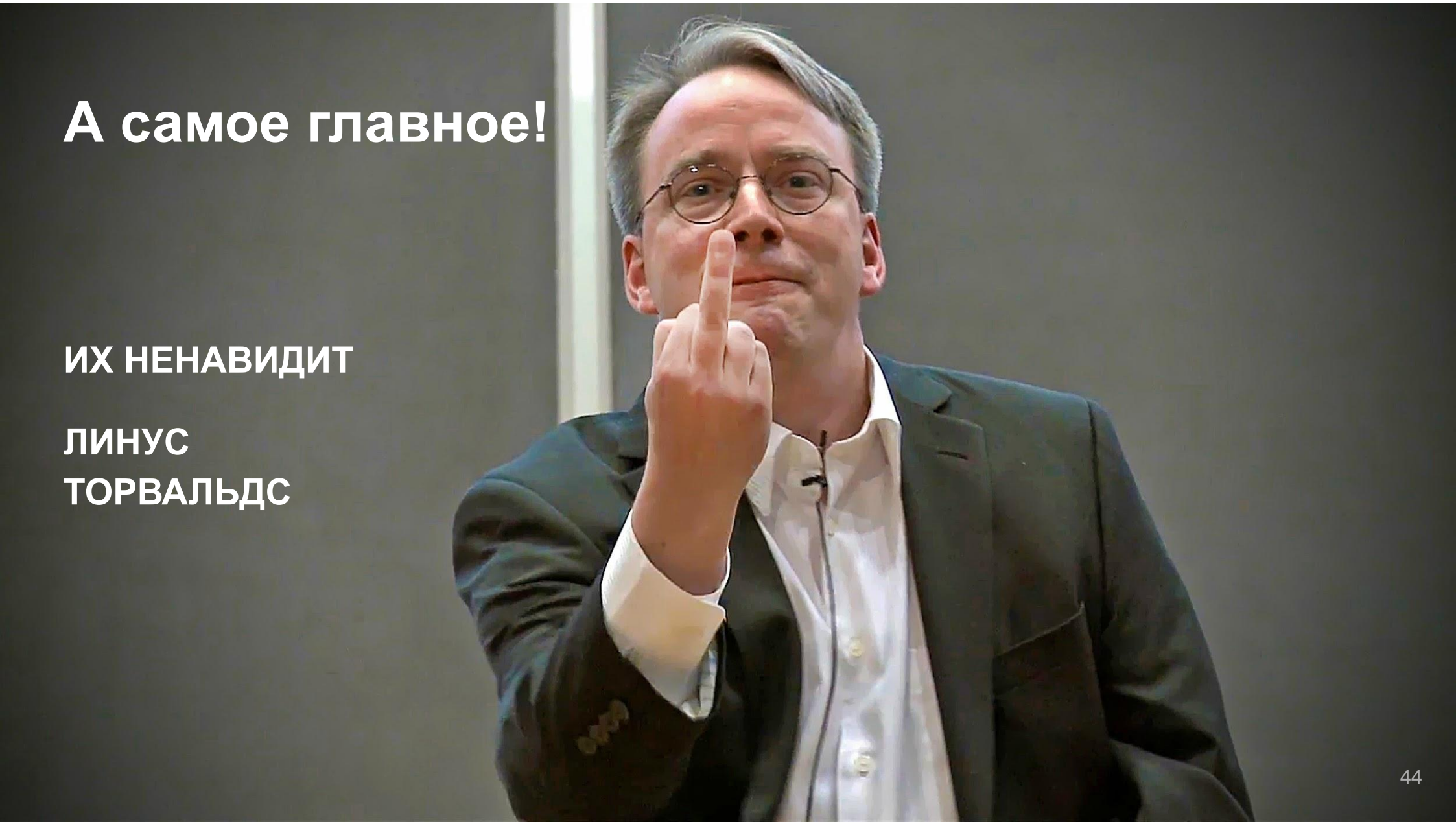
А в чем проблема в Linux?

- Размер стека в ядре Linux 4 страницы = 16К для 64 битного режима
- Глубина стека вызовов может быть достаточно большой (Это НЕ микроядро)
- На чем написан Linux?
- Но самый главный недостаток!!!

А самое главное!

ИХ НЕНАВИДИТ

ЛИНУС ТОРВАЛЬДС



И в заключение

Так что теперь можно сказать о механизме VLA!?

- Потенциально может увеличить производительность
- Производительность может сильно зависеть от профиля программы
- Всегда надо профилировать

Спасибо за внимание

Евгений Ерохин

mailto: the_gabber@mail.ru

telegram: @the_gabber

Вопросы?