ELEC 424/553

Mobile & Embedded Systems

Lecture 19

Rust

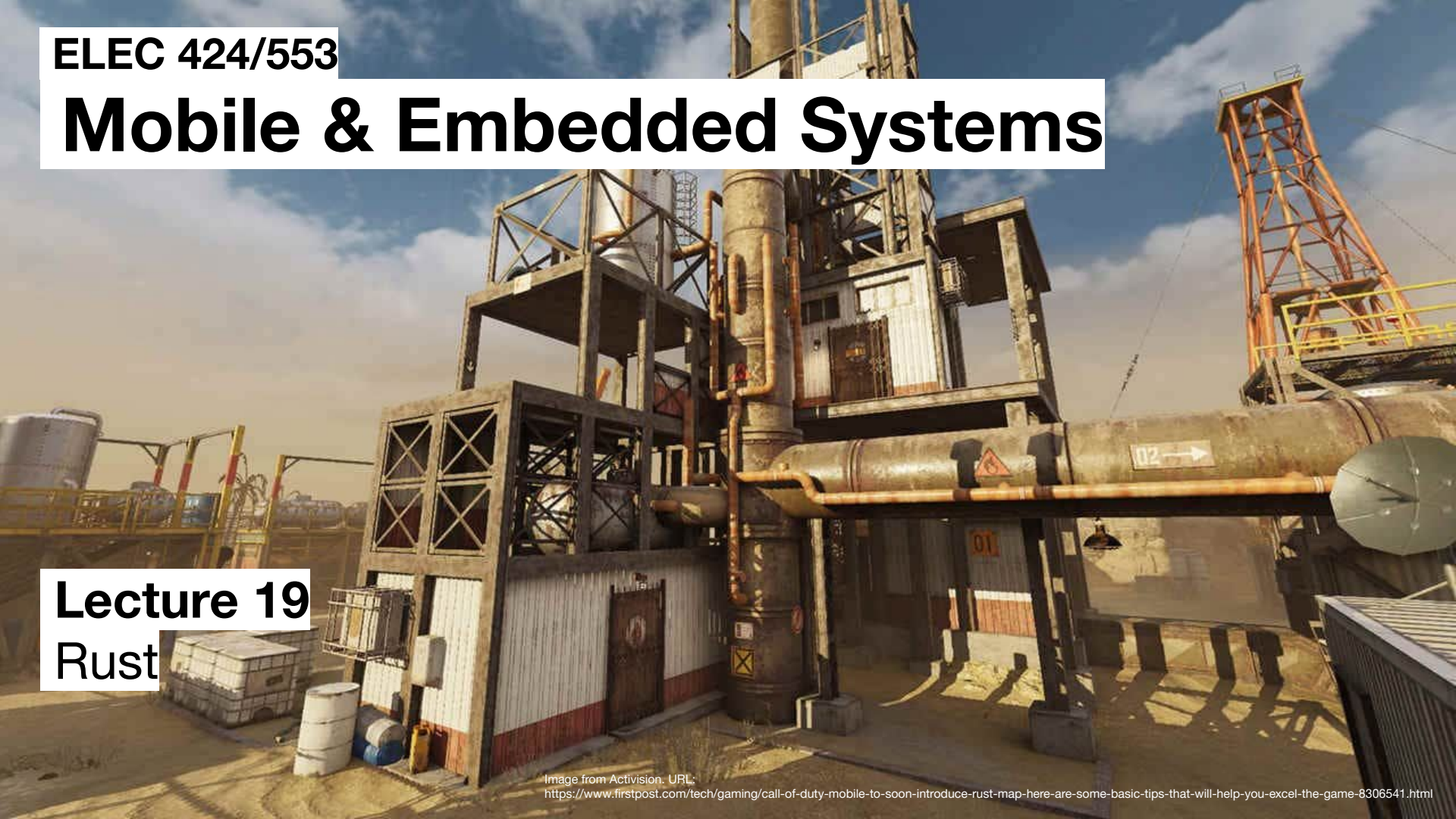
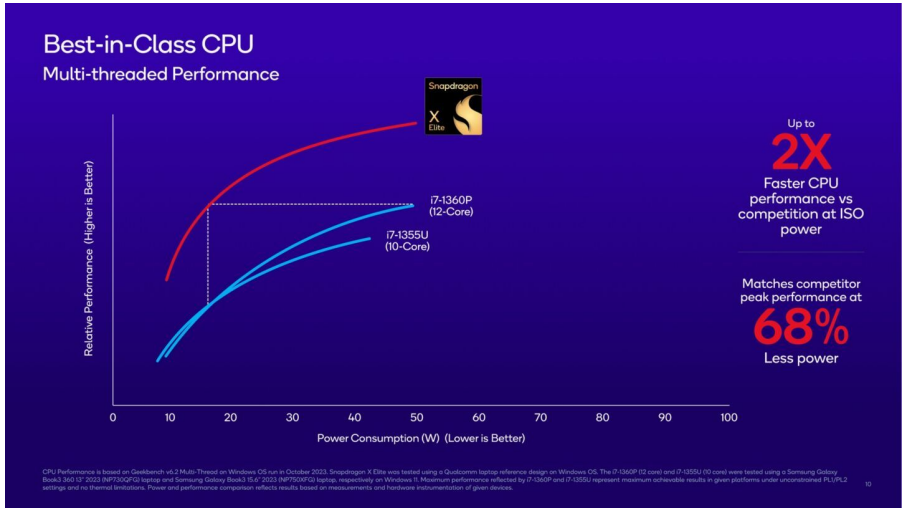


Image from Activision. URL:
https://www.firstpost.com/tech/gaming/call-of-duty-mobile-to-soon-introduce-rust-map-here-are-some-basic-tips-that-will-help-you-excel-the-game-8306541.html

SoC’s Are a Big Deal: Qualcomm Snapdragon X Elite (2023)

“Nearly three years ago, **Qualcomm bought** a company called **Nuvia** for $1.4 billion. Nuvia was mainly working on server processors, but the company's founders and many of its employees had also been **involved in developing the A- and M-series Apple Silicon** processors that have all enabled the iPhone, iPad, and Mac to achieve their enviable blend of performance and battery life. Today, **Qualcomm is formally announcing the fruit of the Nuvia acquisition**: the Qualcomm Snapdragon X Elite is a **12-core, 4 nm chip** that will compete directly with Intel's Core processors and AMD Ryzen chips in PCs—and, less directly, Apple's M2 and M3-series processors for Macs.”



Quote from *Ars Technica* and picture from Qualcomm but found on *Ars Technica*. Bolding in quote is my own. URL for both: https://arstechnica.com/gadgets/2023/10/qualcomm-snapdragon-x-elite-looks-like-the-windows-worlds-answer-to-apple-silicon/

Intel: Don’t Worry, Trust Us

ars TECHNICA SUBSCRIBE SIGN IN

DISARMED —

Intel doesn’t think that Arm CPUs will make a dent in the laptop market

"They've been relegated to pretty insignificant roles in the PC business."

ANDREW CUNNINGHAM - 10/27/2023, 4:58 PM

Intel

Enlarge / Intel's Meteor Lake laptop CPUs launch this December, and they'll be facing competition from more high-end Arm processors.

Screenshot of article from *Ars Technica*. URL: https://arstechnica.com/gadgets/2023/10/intels-ceo-doesnt-seem-worried-about-arm-chips-from-qualcomm-nvidia-or-amd/

Steve Ballmer: Don’t Worry About The iPhone

Video from YouTube channel smugmacgeek. URL: https://www.youtube.com/watch?v=eywi0h_Y5_U

Steve Ballmer - Maybe Should Have Worried More

Related *MacRumors* article: https://www.macrumors.com/2016/11/07/former-microsoft-ceo-steve-ballmer-wrong-iphone/

Video from YouTube channel Charlie Rose. URL: https://www.youtube.com/watch?v=v9d3wp2sGPI

M3

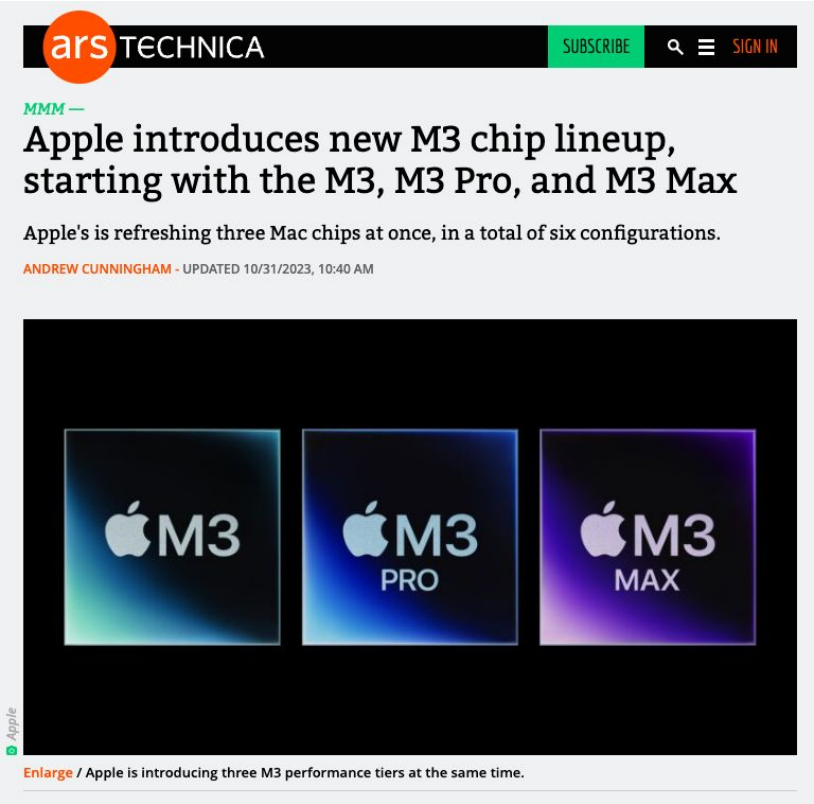
3 nm

Left image from comment by user randolorian on *MacRumors*. URL: https://www.macrumors.com/2020/10/13/apple-magsafe-iphone-12/

Middle image from *MacRumors*. URL: https://www.macrumors.com/2012/05/30/tim-cook-at-d10-loves-that-customers-rumor-sites-and-the-media-care-about-apple/

Right image from *Imgflip*. URL: https://imgflip.com/memegenerator/78785463/Tim-Cook-Laughing

M3


ars TECHNICA SUBSCRIBE SIGN IN

MMM —

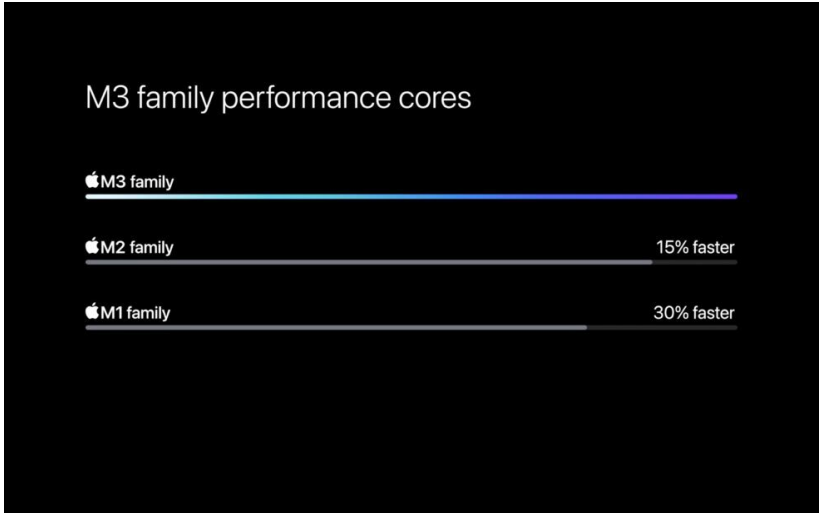
Apple introduces new M3 chip lineup, starting with the M3, M3 Pro, and M3 Max

Apple's is refreshing three Mac chips at once, in a total of six configurations.

ANDREW CUNNINGHAM - UPDATED 10/31/2023, 10:40 AM

Enlarge / Apple is introducing three M3 performance tiers at the same time.

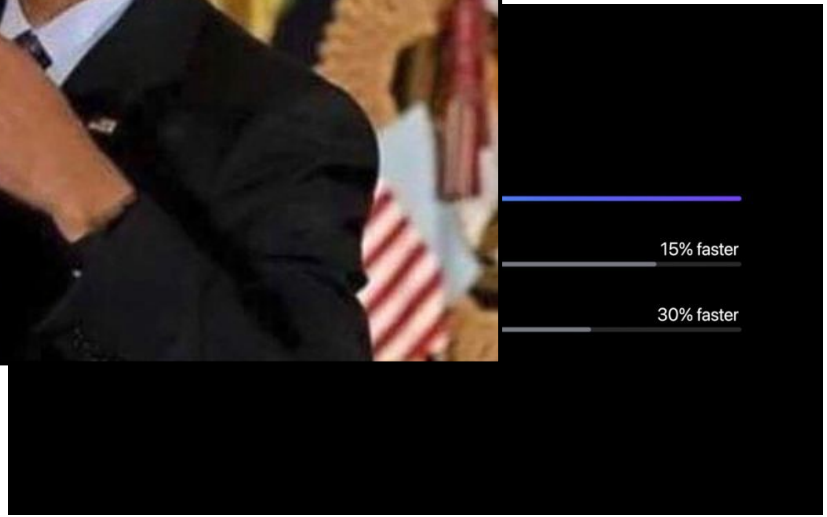
"The M3, M3 Pro, and M3 Max all share the same underlying CPU and GPU architectures, the same ones used in the iPhone 15 Pro's A17 Pro chip. Also like the A17 Pro, all M3 chips are manufactured using a new 3 nm process from Taiwan Semiconductor (TSMC). Let's dive into everything we know about the M3 family's capabilities, plus the differences between each performance tier."



Screenshot of and quote from article from *Ars Technica*. Picture comparing performance from Apple, found in same *Ars Technica* article. URL: https://arstechnica.com/gadgets/2023/10/everything-to-know-about-apples-new-m3-m3-pro-and-m3-max-processors

M3

“The M3, M3 Pro, and M3 Max all share the same
underlying CPU and GPU architectures, the same
Pro's A17 Pro chip. Also
are manufactured
om Taiwan
s dive into everything
's capabilities, plus the
rformance tier.”



Screenshot of and quote from article from *Ars Technica*. Picture comparing performance from Apple, found in same *Ars Technica* article. URL: https://arstechnica.com/gadgets/2023/10/everything-to-know-about-apples-new-m3-m3-pro-and-m3-max-processors

Obama image from *Imgflip*. URL: https://imgflip.com/memegenerator/89853322/obama-medal

Beginning of this year:

ars TECHNICA SUBSCRIBE SIGN IN

AMPUTATING ARM —

Google wants RISC-V to be a “tier-1” Android architecture

Google's keynote at the RISC-V Summit promises official, polished support.

RON AMADEO - 1/3/2023, 3:14 PM

Getty Images

Enlarge / Google HQ.

Over the holiday break, the footage from the recent "RISC-V Summit" was posted for the world to see, and would you believe that Google showed up to profess its love for the up-and-coming CPU architecture?

Screenshot of article from *Ars Technica*. URL: https://arstechnica.com/gadgets/2023/01/google-announces-official-android-support-for-risc-v/

Yesterday:

ars TECHNICA

BIZ & IT TECH SCIENCE POLICY CARS GAMING & CULTURE STORE

EVERY YEAR IS THE "YEAR OF RISC-V?" GOING FORWARD —

Google plans RISC-V Android tools in 2024, wants developers to “be ready”

We've got RISC-V OS support, incoming chips, and soon, an app ecosystem.

RON AMADEO - 10/31/2023, 1:55 PM

Google

Enlarge

51

Android is slowly entering the RISC-V era. So far we've seen Google say it wants to give the up-and-coming CPU architecture "tier-1" support in Android, putting RISC-V on equal footing with Arm. Qualcomm has announced the first mass-market RISC-V Android chip, a still-untitled Snapdragon Wear chip for smartwatches. Now Google has announced a timeline for developer tools via the Google Open Source Blog. The last post is titled "Android and RISC-V: What you need to know to be ready."

Screenshot of article from *Ars Technica*. URL: https://arstechnica.com/gadgets/2023/10/google-plans-risc-v-android-tools-in-2024-wants-developers-to-be-ready/

2021: Rust & Android/Linux Kernel

Google Security Blog

The latest news and insights from Google on security and safety on the Internet

Rust in the Linux kernel

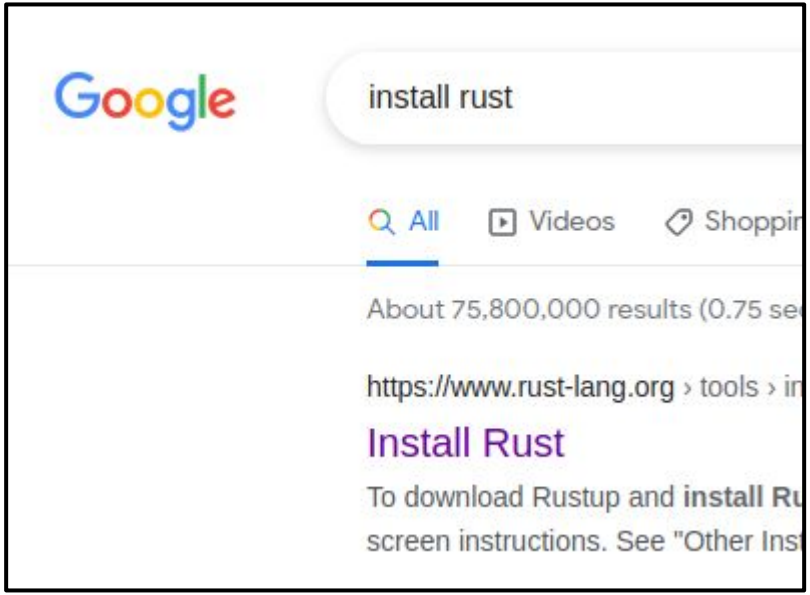
April 14, 2021

Posted by Wedson Almeida Filho, Android Team

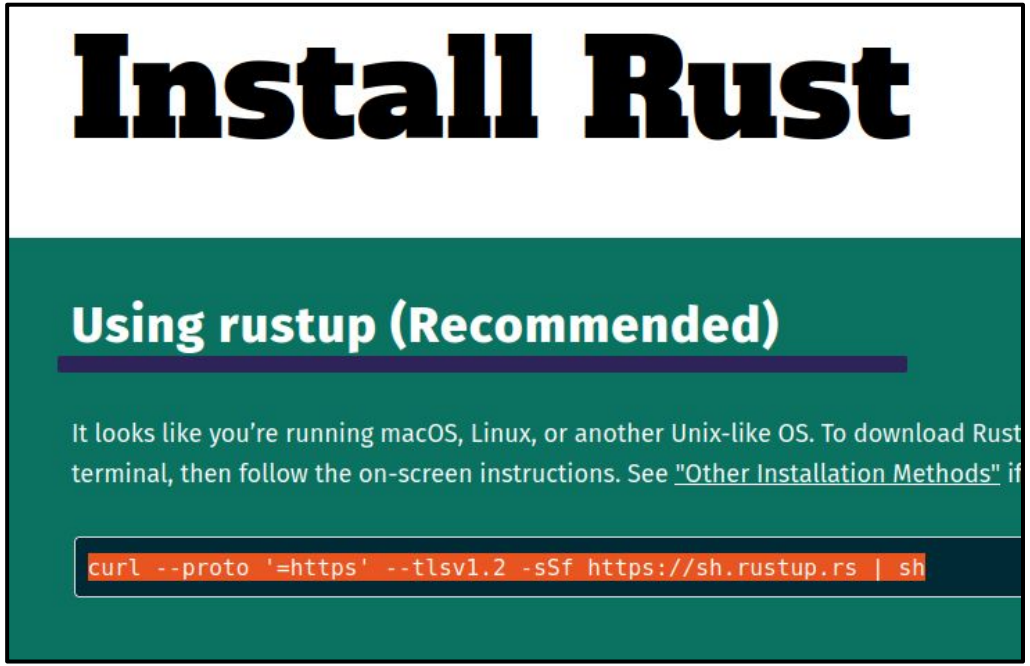
In our previous post, we announced that Android now supports the Rust programming language for developing the OS itself. Related to this, we are also participating in the effort to evaluate the use of Rust as a supported language for developing the Linux kernel. In this post, we discuss some technical aspects of this work using a few simple examples.

“Rust in the Linux kernel”. Google. URL: https://security.googleblog.com/2021/04/rust-in-linux-kernel.html

Before We Get Started - Do This On Laptop



Screenshot of google.com (Google)



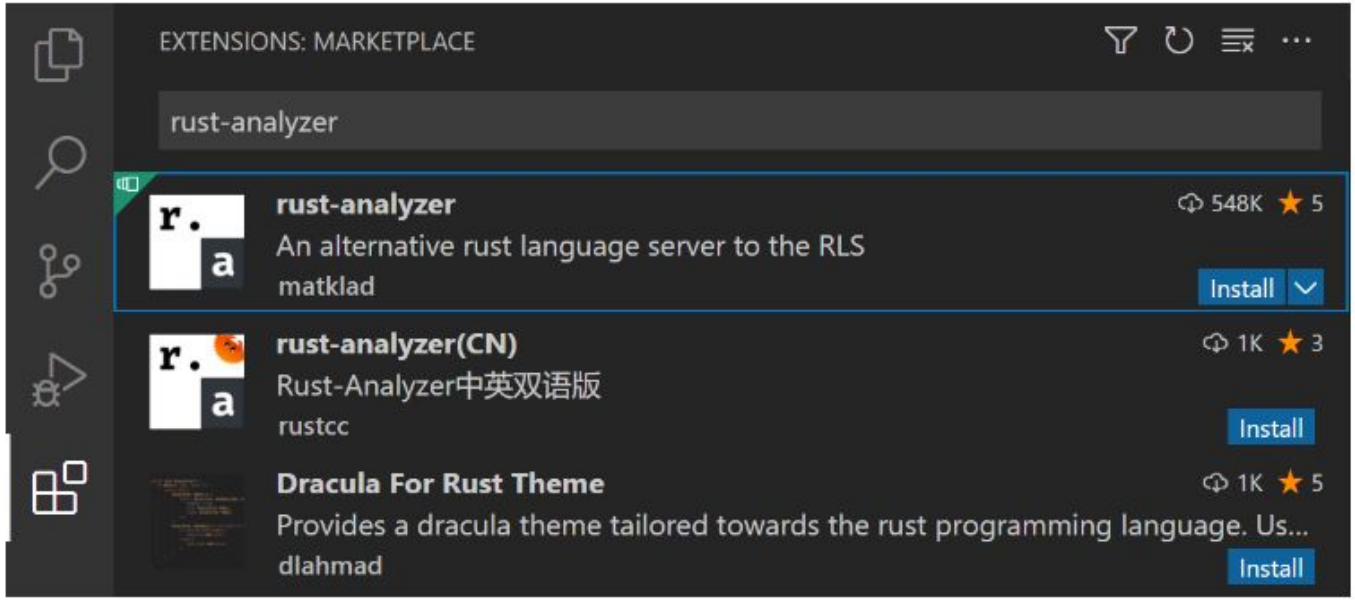
Screenshot from Rust site. URL: https://www.rust-lang.org/tools/install

Also install vs code:
https://code.visualstudio.com/

Install Rust-Analyzer in VS Code

2. Install the rust-analyzer extension

You can find and install the rust-analyzer extension from within VS Code via the Extensions view (⇧⌘X) and searching for 'rust-analyzer'. You should install the **Release Version**.



Screenshot from Microsoft's site. URL: https://code.visualstudio.com/docs/languages/rust

2022: Linus on Rust in the Linux kernel

https://youtu.be/sLimmpZWRNI?si=x0Bi-wMdxiiCsNYf&t=1146

Rust In Linux Kernel

DEVELOPMENT

Linux 6.1 Officially Adds Support for Rust in the Kernel

DEC 20, 2022 • 1 MIN READ

by

Sergio De Simone

After over two years in development, support for using Rust for kernel development has entered a stable Linux release, Linux 6.1, which became available a couple of weeks ago.

Article from InfoQ written by Sergio De Simone. URL: https://www.infoq.com/news/2022/12/linux-6-1-rust/

What Underlies Many Vulnerabilities?

- “Estimated 49% of Chrome security vulnerabilities in 2019 had memory unsafety as a root cause”
  - 72% Firefox
- “Estimated 88% of macOS kernel space vulnerabilities in the 10.14 series had memory unsafety as a root cause”
  - 70% lower bound for Microsoft
  - 65% Ubuntu around 2019

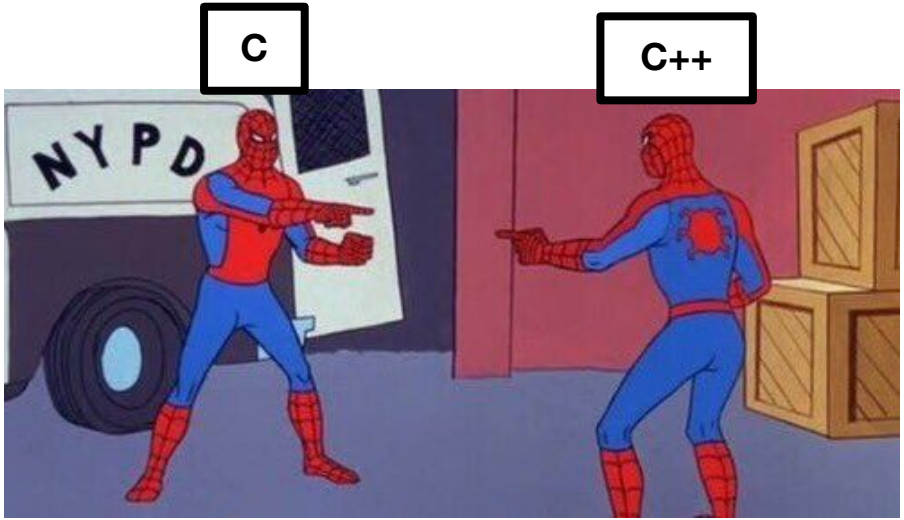
“Writing Linux Kernel Modules in Safe Rust - Geoffrey Thomas & Alex Gaynor”. The Linux Foundation. URL: https://www.youtube.com/watch?v=RyY01fRyGhM

Entertaining Related Presentations

- https://events19.linuxfoundation.org/wp-content/uploads/2017/11/Syzbot-and-the-Tale-of-Thousand-Kernel-Bugs-Dmitry-Vyukov-Google.pdf
- https://www.yecl.org/presentation/2018-04-IoTDI.pdf

Who Did This?

- Memory unsafety of C and C++
- We chose C because it’s close to hardware
- Hardening C is insufficient
- Isolation
  - Microkernels
    - Overhead
  - The problem is within, not only between



https://knowyourmeme.com/memes/spider-man-pointing-at-spider-man

“Writing Linux Kernel Modules in Safe Rust - Geoffrey Thomas & Alex Gaynor”. The Linux Foundation. URL: https://www.youtube.com/watch?v=RyY01fRyGhM

Why Rust?

- We want to solve memory unsafety problems like:
  - Dangling pointer/wild pointer
  - Buffer overflow
  - Playing with memory that has not been initialized

https://en.wikipedia.org/wiki/Dangling_pointer

```
{
   char *dp = NULL;
   /* ... */
   {
       char c;
       dp = &c;
   }
     /* c falls out of scope */
     /* dp is now a dangling pointer */
}
```

```
void vulnerableFunc(char* input) {
    char buffer[80];
    strcpy(buffer, input);
}
int main(int argc, char** argv) {
    if (argc != 2) {
        printf("Arguments: <buffer input>\n");
        exit(1);
    }
    vulnerableFunc(argv[1]);
     printf("Exiting...\n");
    exit(0);
}
```

https://www.tallan.com/blog/2019/04/04/exploring-buffer-overflows-in-c-part-two-the-exploit/

“Writing Linux Kernel Modules in Safe Rust - Geoffrey Thomas & Alex Gaynor”. The Linux Foundation. URL: https://www.youtube.com/watch?v=RyY01fRyGhM

Why Rust? (2)

- Memory safety
- No unwind-based exception handling
- Simpler OO
- Don't "hide things like memory allocations behind your back"
- No garbage collector
- No runtime / thread manager
- Performant FFI to C / assembly

Good but unsuitable safe languages:

- Haskell: GC + runtime
- Go: GC + runtime + overhead for C calls
- D: GC
- Ada: static memory allocations

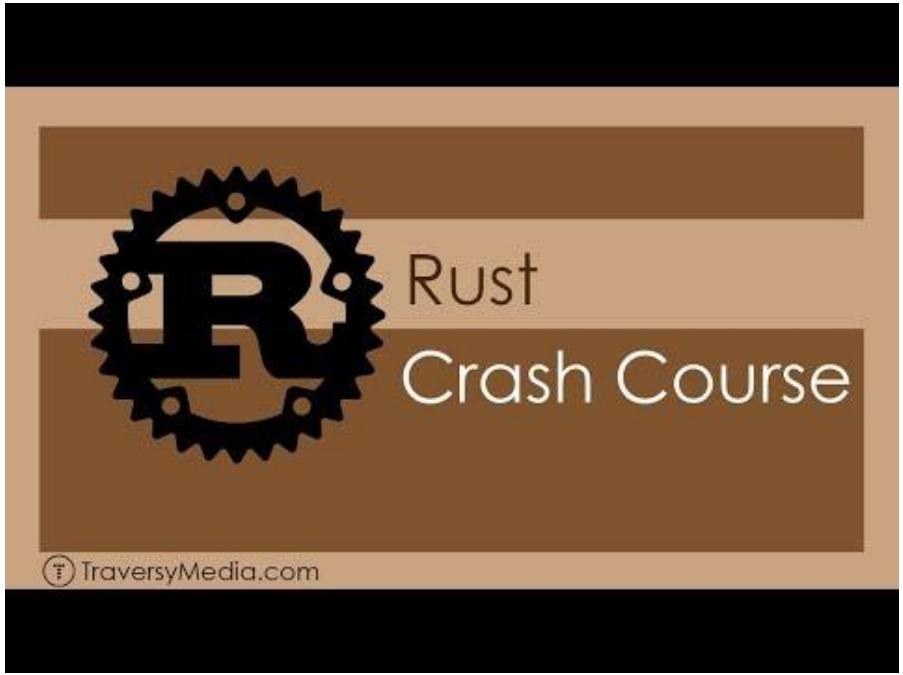
Screenshot from: “Writing Linux Kernel Modules in Safe Rust - Geoffrey Thomas & Alex Gaynor”. The Linux Foundation. URL: https://www.youtube.com/watch?v=RyY01fRyGhM

Rust In a Nutshell

- Compiled language intended for systems programming
- Sponsored by Mozilla as a better / more secure language for Firefox (C++)
- Drop-in replacement for C for incremental rewrites
- Memory safety and thread safety
- No GC
- OS threading
- C-compatible calling convention

Screenshot from: “Writing Linux Kernel Modules in Safe Rust - Geoffrey Thomas & Alex Gaynor”. The Linux Foundation. URL: https://www.youtube.com/watch?v=RyY01fRyGhM

The Following Is Drawn From This Excellent Rust Tutorial



“Rust Crash Course | Rustlang”. Traversy Media. URL: https://www.youtube.com/watch?v=zF34dRivLOw&t=2904s

Installing Rust & Hello World

- https://www.rust-lang.org/tools/install
- Compiler: rustc
- Must have a main function
- `nano hello.rs`
- `rustc hello.rs`
- `./hello`
- `!` used for macros in Rust

```rust
fn main() {
    println!("Hello World!");
}
```

Rust - Cargo

- Package manager for Rust
- Like pip for Python
- `cargo new rhello`
- `cd rhello`
- `cat Cargo.toml`
- `cat .gitignore`
- `cat src/main.rs`

```rust
fn main() {
    println!("Hello World!");
}
```

Compiling With Cargo

- **`cargo run`**
  - Compiles & runs
  - Only compile: **`cargo build`**
    - debug folder
  - Release: **`cargo build --release`**
    - release folder
- Where is the compiled code?
  - `target/debug`
- Output:

```
   Compiling rhello v0.1.0 (/home/joe/rhello)
    Finished dev [unoptimized + debuginfo] target(s) in 0.44s
     Running `target/debug/rhello`
Hello, world!
```

Source Files Go In `rhello/src`

- Print from a different file
- **`nano print.rs`**
- pub - public function
    - Can use this function in other files

```
pub fn run() {
    // Print to console
    println!("Hello world from print.rs!");
}
```

“Rust Crash Course | Rustlang”. Traversy Media. URL: https://www.youtube.com/watch?v=zF34dRivLOw&t=2904s

Back To `main.rs`

- Let's have the main function call our print function in the print file
- **`mod`** - module
- Includes other modules (files)
- Can also do `use`

```
use print::run();
```

- Then can call **`run()`** without **`print::`**

```
// Use name of file
mod print;

fn main() {
    print::run();
}
```


"Rust Crash Course | Rustlang". Traversy Media. URL: https://www.youtube.com/watch?v=zF34dRivLOw&t=2904s

Formatting Printed Output

- In Python we can just do print(1)
- Not in Rust!

```
   Compiling rhello v0.1.0 (/home/joe/rhello)
error: format argument must be a string literal
 --> src/print.rs:6:11
  |
6 |   println!(1);
  |            ^
  |
help: you might be missing a string literal to format with
  |
6 |   println!("{}", 1);
  |           +++++
error: could not compile `rhello` due to previous error
```

Help?? Where has this been all our lives?

“Rust Crash Course | Rustlang”. Traversy Media. URL: https://www.youtube.com/watch?v=zF34dRivLOw&t=2904s

Formatting Printed Output

- Fixing it: **`println!("{}", 1); // {} - placeholder`**

```
    Compiling rhello v0.1.0 (/home/joe/rhello)
     Finished dev [unoptimized + debuginfo] target(s) in 0.15s
      Running `/home/joe/rhello/target/debug/rhello`
Hello world from print.rs!
1
```

“Rust Crash Course | Rustlang”. Traversy Media. URL: https://www.youtube.com/watch?v=zF34dRivLOw&t=2904s

Formatting Printed Output - Multiple Items

Standard

```
println!("{} hi {}", "Oh", "Mark");
```

Positional arguments:

```
println!("{1} hi {0}", "Oh", "Mark");
```

Named arguments:

```
println!("{noun1} over {noun2}", noun1="Game", noun2="man");
```

Placeholder traits:

```
println!("Binary: {:b} Hex: {:x}", 16, 16);
```


“Rust Crash Course | Rustlang”. Traversy Media. URL: https://www.youtube.com/watch?v=zF34dRivLOw&t=2904s

Formatting Printed Output - Debug

Placeholder for debug trait (good for printing array):

```
println!("{:?}", ("This", "is a tuple", 10, false));
```

Math:

```
println!("10+10={}", 100+20);
```


“Rust Crash Course | Rustlang”. Traversy Media. URL: https://www.youtube.com/watch?v=zF34dRivLOw&t=2904s

Variables

Immutable by default!

```
nano src/vars.rs
```

Triggering immutability error:

```
error[E0384]: cannot assign twice to immutable variable `age`
  --> src/vars.rs:10:2
   |
7  |  let age = 37;
   |        ---
   |        |
   |        first assignment to `age`
   |        help: consider making this binding mutable: `mut age`
...
10 |  age = 38;
   |  ^^^^^^^^ cannot assign twice to immutable variable
```

```rust
pub fn run(){

      // We don't expect this to change
(see exception - Prince)
      let name = "Brad";

      // We do expect this to change, but
it cannot as currently stated
      let age = 37;

      // Try to change it anyways
      age = 38;

      // Print name
      println!("My name is {} and I am
{}", name, age);
}
```

“Rust Crash Course | Rustlang”. Traversy Media. URL: https://www.youtube.com/watch?v=zF34dRivLOw&t=2904s

Variables

Add **mut** to age

Will get warning:

```
warning: value assigned to `age` is never read
 --> src/vars.rs:7:10
  |
7 |  let mut age = 37;
  |           ^^^
  |
  = note: `#[warn(unused_assignments)]` on by default
  = help: maybe it is overwritten before being read?
```

```rust
pub fn run(){

      // We don't expect this to change
(see exception - Prince)
      let name = "Brad";

      // We do expect this to change, but
it cannot as currently stated
      let mut age = 37;

      // Try to change it anyways
      age = 38;

      // Print name
      println!("My name is {} and I am
{}", name, age);
}
```


“Rust Crash Course | Rustlang”. Traversy Media. URL: https://www.youtube.com/watch?v=zF34dRivLOw&t=2904s

Variables - `const`

Worked with at compile time (not really a variable)

Must declare data type

First time we've mentioned a data type!

```
pub fn run(){

      // We don't expect this to change
(see exception - Prince)
      let name = "Brad";

      // We do expect this to change, but
it cannot as currently stated
      let mut age = 37;

      // Try to change it anyways
      age = 38;

      // Print name
      println!("My name is {} and I am
{}", name, age);

      // Constant
      const ID: i32 = 007;
      println!("Hi my name is 00{}", ID);
}
```


"Rust Crash Course | Rustlang". Traversy Media. URL: https://www.youtube.com/watch?v=zF34dRivLOw&t=2904s

Uninitialized Variables

```rust
fn main() {
  let mut x: i32;
  println!("Hello world! x = {}", x);
  x = 5;
}
```

C will give a warning
Rust will give an error

Screenshot from: “Writing Linux Kernel Modules in Safe Rust - Geoffrey Thomas & Alex Gaynor”. The Linux Foundation. URL: https://www.youtube.com/watch?v=RyY01fRyGhM

Initialize Multiple Variables

Tuple syntax

```
pub fn run(){

      // We don't expect this to change (see exception - Prince)
      let name = "Brad";

      // We do expect this to change, but it cannot as currently stated
      let mut age = 37;

      // Try to change it anyways
      age = 38;

      // Print name
      println!("My name is {} and I am {}", name, age);

      // Constant
      const ID: i32 = 007;
      println!("Hi my name is 00{}", ID);

      // Multiple variables
      let (description, value) = ("net worth", -10);
      println!("My {} is {}", description, value);
}
```

“Rust Crash Course | Rustlang”. Traversy Media. URL: https://www.youtube.com/watch?v=zF34dRivLOw&t=2904s

Data Types (file from: https://github.com/bradtraversy/rust_sandbox/blob/master/src/types.rs)

```
/*
Primitive Types--
Integers: u8, i8, u16, i16, u32, i32, u64, i64, u128, i128 (number of bits they take in memory)
Floats: f32, f64
Boolean (bool)
Characters (char)
Tuples
Arrays
*/

// Rust is a statically typed language, which means that it must know the types of all variables at
compile time, however, the compiler can usually infer what type we want to use based on the value and
how we use it.
```


“Rust Crash Course | Rustlang”. Traversy Media. URL: https://www.youtube.com/watch?v=zF34dRivLOw&t=2904s

Data Types (2) (file from: https://github.com/bradtraversy/rust_sandbox/blob/master/src/types.rs)

```rust
pub fn run() {
  // Default is "i32"
  let x = 1;
  // Default is "f64"
  let y = 2.5;
  // Add explicit type
  let z: i64 = 4545445454545;
  // Find max size
  println!("Max i32: {}", std::i32::MAX);
  println!("Max i64: {}", std::i64::MAX);

  // Boolean
  let is_active: bool = true;
  // Get boolean from expression
  let is_greater: bool = 10 < 5;

  let a1 = 'a';
  let face = '\u{1F600}';

  println!("{:?}", (x, y, z, is_active, is_greater, a1, face));
}
```

“Rust Crash Course | Rustlang”. Traversy Media. URL: https://www.youtube.com/watch?v=zF34dRivLOw&t=2904s

Arrays (file from: https://github.com/bradtraversy/rust_sandbox/blob/master/src/arrays.rs)

```rust
// Arrays - Fixed list where elements are the same data types
use std::mem;
pub fn run() {
  let mut numbers: [i32; 4] = [1, 2, 3, 4];

  // Re-assign value
  numbers[2] = 20;
  println!("{:?}", numbers);

  // Get single val
  println!("Single Value: {}", numbers[0]);

  // Get array length
  println!("Array Length: {}", numbers.len());

  // Arrays are stack allocated
  println!("Array occupies {} bytes", mem::size_of_val(&numbers));

  // Get Slice
  let slice: &[i32] = &numbers[1..3];
  println!("Slice: {:?}", slice);
}
```

“Rust Crash Course | Rustlang”. Traversy Media. URL: https://www.youtube.com/watch?v=zF34dRivLOw&t=2904s

Vectors (file from: https://github.com/bradtraversy/rust_sandbox/blob/master/src/vectors.rs)

```rust
// Vectors - Resizable arrays

use std::mem;

pub fn run() {
  let mut numbers: Vec<i32> = vec![1, 2, 3, 4];

  // Re-assign value
  numbers[2] = 20;

  // Add on to vector
  numbers.push(5);
  numbers.push(6);

  // Pop off last value
  numbers.pop();

  println!("{:?}", numbers);

  // Get single val
  println!("Single Value: {}", numbers[0]);
```

“Rust Crash Course | Rustlang”. Traversy Media. URL: https://www.youtube.com/watch?v=zF34dRivLOw&t=2904s

Vectors (2) (file from: https://github.com/bradtraversy/rust_sandbox/blob/master/src/vectors.rs)

```rust
    // Get vector length
    println!("Vector Length: {}", numbers.len());

    // Vectors are heap allocated
    println!("Vector occupies {} bytes", mem::size_of_val(&numbers));

    // Get Slice
    let slice: &[i32] = &numbers[1..3];
    println!("Slice: {:?}", slice);

    // Loop through vector values
    for x in numbers.iter() {
        println!("Number: {}", x);
    }

    // Loop & mutate values
    for x in numbers.iter_mut() {
        *x *= 2;
    }

    println!("Numbers Vec: {:?}", numbers);
}
```

“Rust Crash Course | Rustlang”. Traversy Media. URL: https://www.youtube.com/watch?v=zF34dRivLOw&t=2904s

Functions (file from: https://github.com/bradtraversy/rust_sandbox/blob/master/src/functions.rs)

```rust
// Functions - Used to store blocks of code for re-use

pub fn run() {
  greeting("Hello", "Jane");

  // Bind function values to variables
  let get_sum = add(5, 5);
  println!("Sum: {}", get_sum);

  // Closure
  let n3: i32 = 10;
  let add_nums = |n1: i32, n2: i32| n1 + n2 + n3;
  println!("C Sum: {}", add_nums(3, 3));
}

fn greeting(greet: &str, name: &str) { // Must specify argument data types
  println!("{} {}, nice to meet you!", greet, name);
}

fn add(n1: i32, n2: i32) -> i32 {
  n1 + n2  // Implicit return - no semicolon
}
```


“Rust Crash Course | Rustlang”. Traversy Media. URL: https://www.youtube.com/watch?v=zF34dRivLOw&t=2904s

Conditionals (file from: https://github.com/bradtraversy/rust_sandbox/blob/master/src/conditionals.rs)

```rust
// Conditionals - Used to check the condition of something and act on the result

pub fn run() {
  let age: u8 = 22;
  let check_id: bool = true;
  let knows_person_of_age = true;

  // If/Else
  if age >= 21 && check_id || knows_person_of_age {
      println!("Bartender: What would you like to drink?");
  } else if age < 21 && check_id {
      println!("Bartender: Sorry, you have to leave");
  } else {
      println!("Bartender: I'll need to see your ID");
  }

  // Shorthand If
  let is_of_age = if age >= 21 { true } else { false };
  println!("Is Of Age: {}", is_of_age)
}
```

“Rust Crash Course | Rustlang”. Traversy Media. URL: https://www.youtube.com/watch?v=zF34dRivLOw&t=2904s

Loops (file from: https://github.com/bradtraversy/rust_sandbox/blob/master/src/loops.rs)

```
// Loops - Used to iterate until a condition is met

pub fn run() {
  let mut count = 0;

  // Infinite Loop
  // loop {
  //   count += 1;
  //   println!("Number: {}", count);

  //   if count == 20 {
  //  break;
  //   }
  // }

  // While Loop (FizzBuzz)
  // while count <= 100 {
  //   if count % 15 == 0 {
  //  println!("fizzbuzz");
  //   } else if count % 3 == 0 {
  //  println!("fizz");
  //   } else if count % 5 == 0 {
  //  println!("buzz");
  //   } else {
```

"Rust C[illegible] Traversy Media. URL: https://www.youtube.com/watch?v=zF34dRivLOw&t=2904s

Structs

```rust
struct Rectangle {
  length: f64,
  width: f64,
}

impl Rectangle {
  fn area(&self) -> f64 {
    self.length * self.width
  }
}
```


Screenshot from: “Writing Linux Kernel Modules in Safe Rust - Geoffrey Thomas & Alex Gaynor”. The Linux Foundation. URL: https://www.youtube.com/watch?v=RyY01fRyGhM

Dangling Pointer

```rust
fn main() {
  let mut y: &i32;
  for i in 1..5 {
      y = &i;
  }
  println!("{}", y);
}
```

```
error[E0597]: `i` does not live long enough
 --> src/main.rs:4:11
  |
4 |       y = &i;
  |           ^^ borrowed value does not live long enough
5 |   }
  |   - `i` dropped here while still borrowed
6 |   println!("{}", y);
  |                  - borrow later used here
```


Screenshot from: “Writing Linux Kernel Modules in Safe Rust - Geoffrey Thomas & Alex Gaynor”. The Linux Foundation. URL: https://www.youtube.com/watch?v=RyY01fRyGhM