Rust: Systems Programmers Can Have Nice Things

Arun Thomas
arun.thomas@acm.org
@arunthomas
EuroBSDCon 2019

On Systems Programming

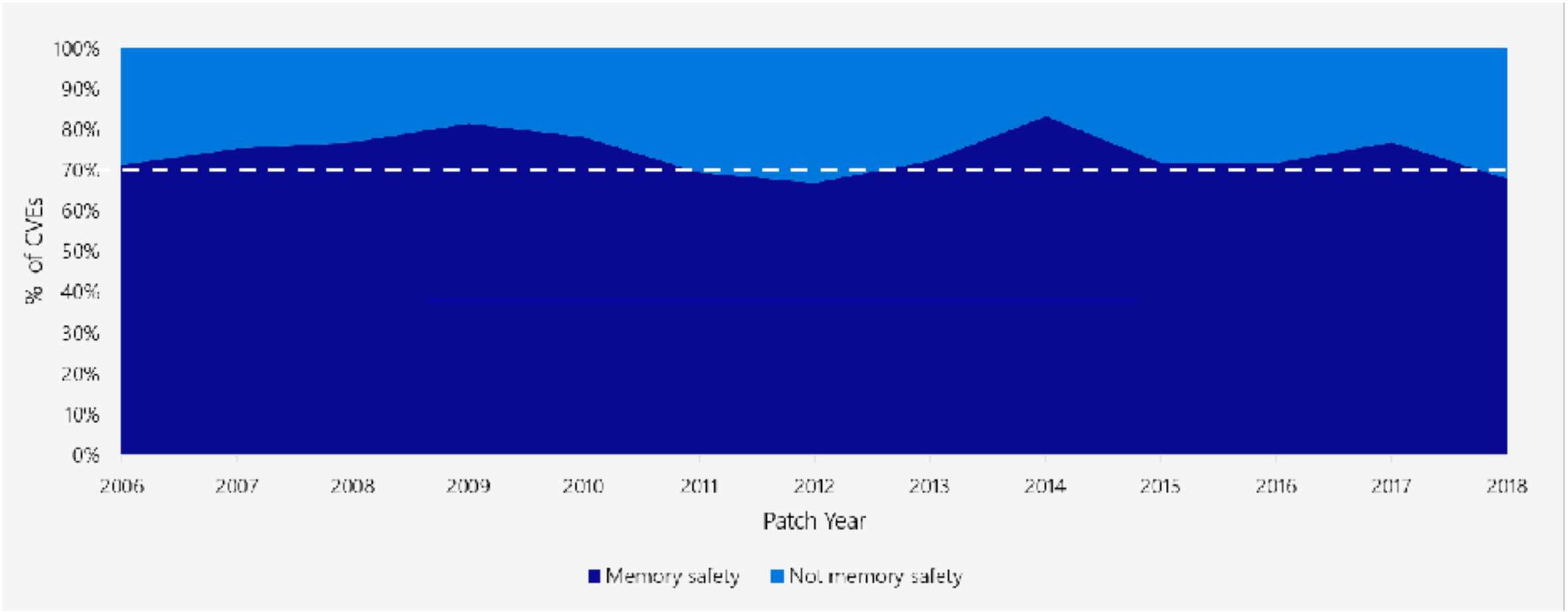
*[A] systems programmer has seen the **terrors** of the world and understood the **intrinsic horror** of existence*

-James Mickens, The Night Watch

What Makes Systems Software Hard?

- **Stakes are High:** Systems software is **critical to enforcing security and safety**
    - Kernels, hypervisors, firmware, bootloaders, embedded software, language runtimes, browsers, …
- Usually written in C or C++ for **performance**
    - BUT C and C++ are **not memory-safe**
- **Memory corruption vulnerabilities** abound (and are exploited)
    - See recent Microsoft study (next slide)

Memory Unsafety is a Problem



Microsoft found ~70% of CVEs in their products each year continue to be memory safety issues

(Matt Miller, MSRC @ Blue Hat IL 2019)

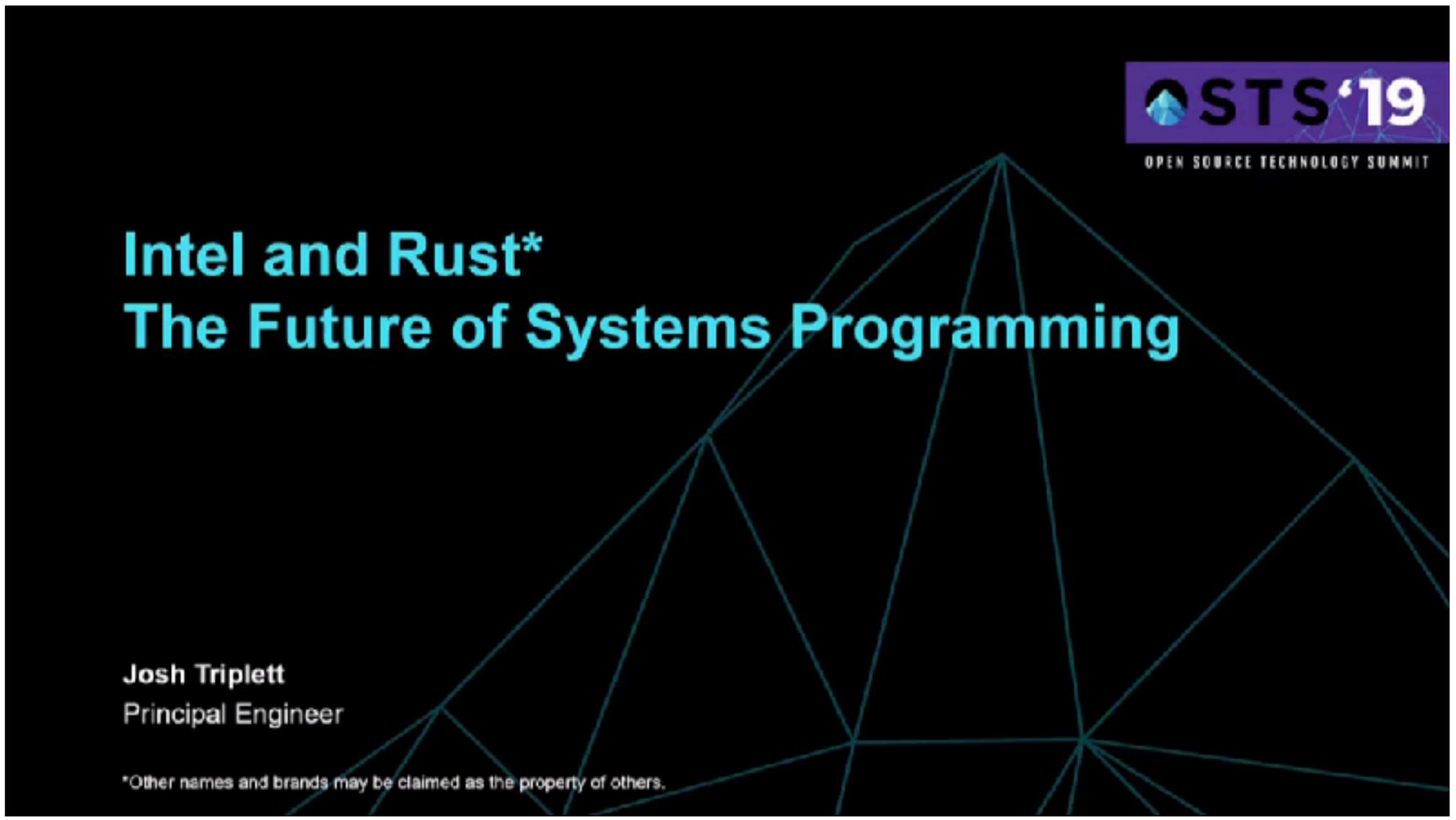
Microsoft and Rust

A bright future

Why Rust for safe systems programming

We believe Rust changes the game when it comes to writing safe systems software. Rust provides the performance and control needed to write low-level systems, while empowering software developers to write robust, secure programs.

Intel and Rust



Talk Overview

"Systems Programmers Can Have Nice Things"
-Robert O'Callahan
Random Thoughts On Rust

- Why Rust?
- Rust for Systems Software
- Getting Started with Rust on BSD

Why  ust?

I like C.

But it turns out programming languages have **evolved** in the last **50 years**.

Rust is a **safe, fast,**
productive systems
programming language.

Mozilla and Rust

- Rust was originally created by Mozilla Research
  - Initial use case: Servo browser engine
- Mozilla began shipping Rust components in Firefox 48 in 2016
  - **Oxidation** is Mozilla's term for "Rusting out" components
- Rust code has improved Firefox's security and performance
  - **Security**: Safe parsers (MP4 metadata parser)
  - **Performance**: New parallel CSS engine for faster page loads

Systems Programmers Can Have Nice Things: Speed and Safety

- Performance on par with C
- Memory safety without garbage collection overheads
  - Suitable for low-level systems software
- Thread safety (“Fearless Concurrency”)
  - No concurrency bugs associated with multi-threaded code
- Rust’s type system enforces memory/thread safety

Systems Programmers Can Have Nice Things: Productivity

- Good interoperability with C
    - Important for systems software
- “Fancy” language features
    - Type inference, algebraic data types, pattern matching, traits, generics
    - Modules, hygienic macros, handy literals (*0b1010*, *0x8000_0000)*
- Great tooling
    - ***rustc*** - “friendly compiler with useful error messages”
    - ***cargo*** - Great package manager and build system
    - ***rustfmt*** - No bike shedding over coding styles

Hello, EuroBSDCon!

```
fn main() {
    println!("Hello, EuroBSDCon!");
}
```

Factorial

```
fn fact (n:int) -> int {
    if n == 0 { 1 }
    else { n * fact(n-1) }
}
```

Variables

```
fn main() {
    let x = 2; // Immutable
    x = 3;     // Error
}
```

Mutable Variables

```
fn main() {
    let mut x = 2;
    x += 1;
    println!{“x is {}", x} // 3
}
```

References

- v2 “borrows” an immutable reference to v1

```
fn main() {
    let v1 = vec![1, 2, 3]; // vector
    let v2 = &v1;           // reference
    println!{"v1 is {:?}", v1} // [1, 2, 3]
    println!{"v2 is {:?}", v2} // [1, 2, 3]
}
```

Mutable References

- Rust allows only one mutable reference in a given scope

```
fn main() {
    let mut s = String::from("hello");

    let r1 = &s; // OK, immutable ref
    let r2 = &s; // OK, immutable ref
    let r3 = &mut s; // Error

    println!("{}, {}, and {}", r1, r2, r3);
}
```

```
error[E0502]: cannot borrow `s` as mutable because it is also borrowed as
immutable
```

Ownership Overly Simplified

- Ownership model is powerful, but takes time to fully grok
- Rust compiler tracks ownership (and borrowing/moves)
    - Determines object lifetimes
    - Restrict mutation of shared state (Allow mutation **OR** allow sharing)
- Ownership rules:
    - Each value has a variable that’s the owner
    - Only one owner at a time
    - When owner goes out of scope, value will be dropped

Rust for Systems Software

Gaining Popularity for Systems Software (1/2)

- Rust Operating Systems: Tock, Redox, “Writing an OS in Rust”, …
  - Tock is particularly interesting for deeply embedded applications
  - *See Multiprogramming a 64 kB Computer Safely and Efficiently* (SOSP’17)
- Google, Amazon, and Intel developing Rust-based hypervisors
  - crosvm, Firecracker, Cloud Hypervisor
  - Created rust-vmm initiative to increase sharing/collaboration
- Coreboot developers created oreboot - “coreboot, with C removed”
  - See Open Source Firmware Conference 2019 talk

Gaining Popularity for Systems Software (2/2)

- Microsoft using Rust for Azure IoT Edge Security Daemon
- Google using Rust for Fuchsia components
- Facebook using Rust for Libero cryptocurrecy and parts of HHVM PHP runtime
- Intel contributed VxWorks target
- Mozilla and Fastly writing Webassembly runtimes (wastime, Lucet) in Rust
- CloudFlare and Dropbox using Rust for backend services

Is it time to rewrite *BSD in Rust?

Is it time to rewrite the operating system in Rust?

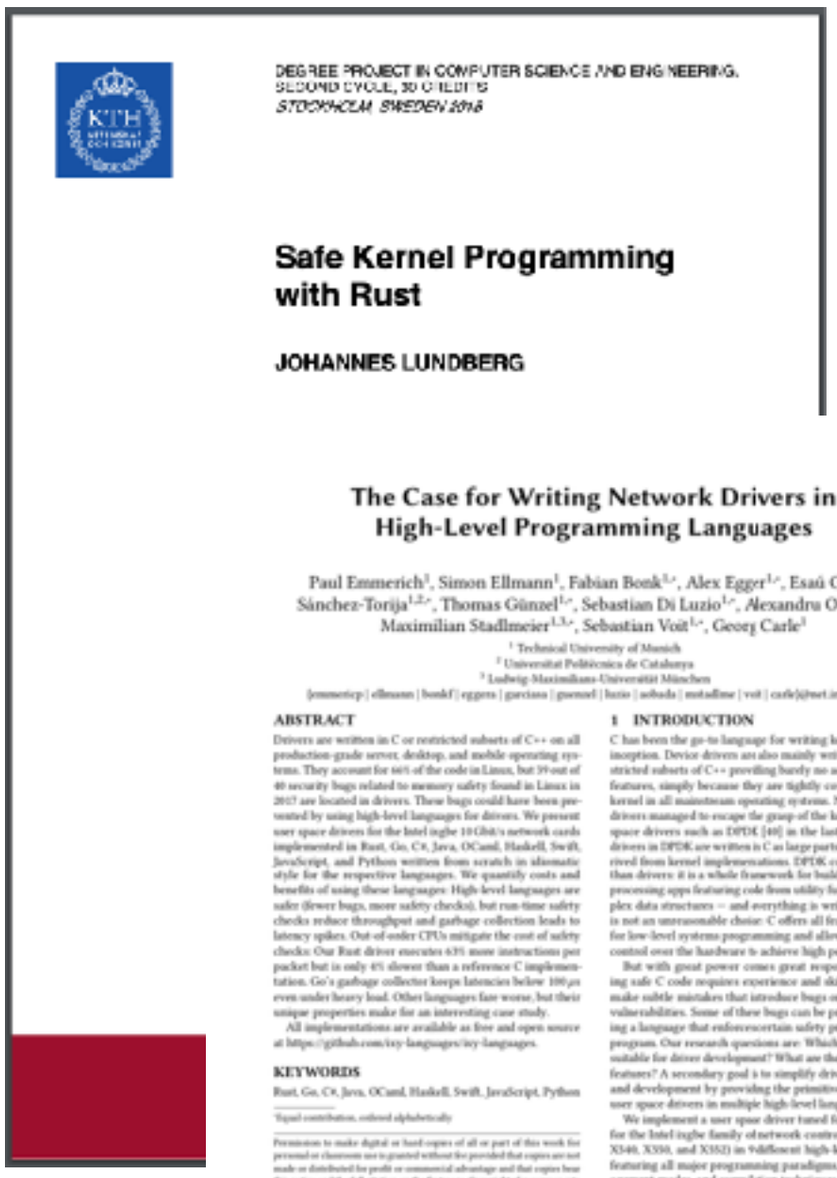
Bryan Cantrill
CTO

bryan@joyent.com
@bcantrill

Mostly no, but maybe…

Oxidation BSD-style?

- Explore Rust for:
  - Device drivers
  - Filesystems
  - Userland utilities
  - Networked services



DEGREE PROJECT IN COMPUTER SCIENCE AND ENGINEERING,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2018

Safe Kernel Programming with Rust

JOHANNES LUNDBERG

The Case for Writing Network Drivers in High-Level Programming Languages

Paul Emmerich[1], Simon Ellmann[1], Fabian Bonk[1,*], Alex Egger[1,*], Esaú García Sánchez-Torija[1,2,*], Thomas Günzel[1,*], Sebastian Di Luzio[1,*], Alexandru Obada[1,*], Maximilian Stadlmeier[1,3,*], Sebastian Voit[1,*], Georg Carle[1]

[1] Technical University of Munich
[2] Universitat Politècnica de Catalunya
[3] Ludwig-Maximilians-Universität München

{emmericp | ellmann | bonkf | eggers | garciass | guenzel | luzio | obada | mstadlme | voit | carle}@net.in.tum.de

ABSTRACT

Drivers are written in C or restricted subsets of C++ on all production-grade server, desktop, and mobile operating systems. They account for 66% of the code in Linux, but 39 out of 40 security bugs related to memory safety found in Linux in 2017 are located in drivers. These bugs could have been prevented by using high-level languages for drivers. We present user space drivers for the Intel ixgbe 10 Gbit/s network cards implemented in Rust, Go, C#, Java, OCaml, Haskell, Swift, JavaScript, and Python written from scratch in idiomatic style for the respective languages. We quantify costs and benefits of using these languages: High-level languages are safer (fewer bugs, more safety checks), but run-time safety checks reduce throughput and garbage collection leads to latency spikes. Out-of-order CPUs mitigate the cost of safety checks: Our Rust driver executes 63% more instructions per packet but is only 4% slower than a reference C implementation. Go's garbage collector keeps latencies below 100 µs even under heavy load. Other languages fare worse, but their unique properties make for an interesting case study.

All implementations are available as free and open source at https://github.com/ixy-languages/ixy-languages.

KEYWORDS

Rust, Go, C#, Java, OCaml, Haskell, Swift, JavaScript, Python

*Equal contribution, ordered alphabetically

ANCS'19, 2019

Getting Started with Rust on BSD

Installing Rust

- FreeBSD
  - $ pkg install rust
  - $ pkg install rust-nightly
- NetBSD and OpenBSD
  - $ pkg_add rust
- DragonFly BSD
  - $ pkg install rust

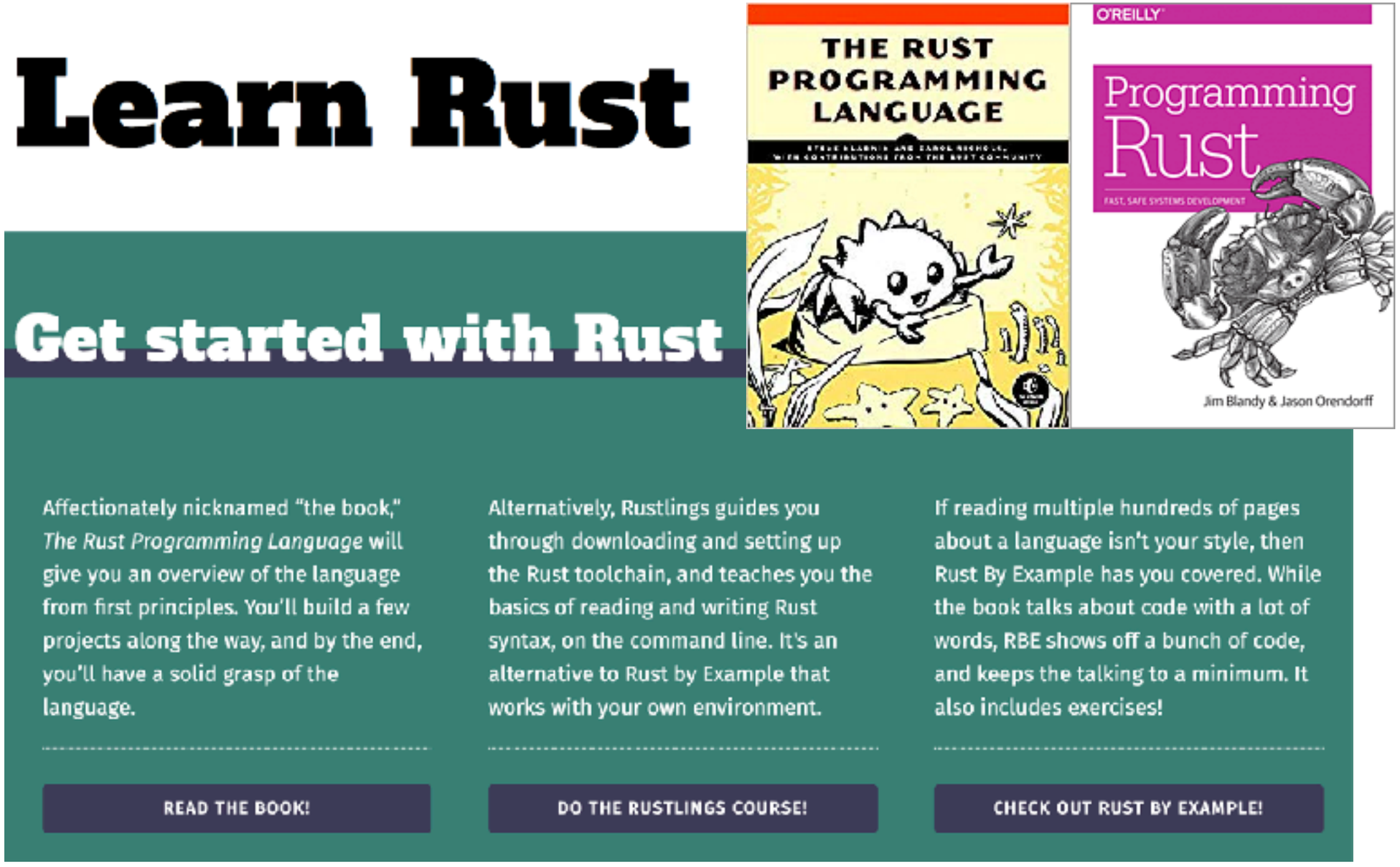
Rustup

- Alternatively, use rustup for FreeBSD and NetBSD
- Handy when managing multiple Rust toolchains
- $ curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
- ‘curl | sh’ pattern not required

Hello World w/ Cargo

```
$ cargo new --bin hello
 Created binary (application) `hello` package
$ cd hello
$ cargo run
   Compiling hello v0.1.0 (/private/tmp/hello)
    Finished dev [unoptimized + debuginfo]
target(s) in 1.52s
     Running `target/debug/hello`
Hello, world!
$ vi src/main.rs # Edit source
$ vi Cargo.toml # Edit pkg dependencies
$ cargo run
…
Hello, EuroBSDCon!
```

Rust Resources



Summary

- **Rust: Systems Programmers Can Have Nice Things**
  - Higher-level language safety and productivity with low-level language performance
  - Growing industry use for systems programming
  - Easy to get hacking Rust on BSD
  - The Future: Oxidation of BSD?