Rust for the rest of us

John Downey | @jtdowney

~~Rust for the rest of us~~

Why I am excited about Rust

8:30 8:35	**Opening Remark - Eric Kepes College Center Square**									
8:35 9:05	**Keynote - Jennifer Davis College Center Square**									
9:20 10:20	Rust out your C Carol (Nichols || Goulding)	Herding the Cats: Client Management for the Rest of Us Nick Stewart	Why Does Automated Testing Fail Joel Mason	The Alchemy of User Experience Benjamin Bykowski	Performance Management in an Agile World Sudhir Thaikootathil/Jay Anderson	Be the Center of the Universe with REST & ASP.NET Web API James Bender	Property Based Testing in Javascript using JSVerify Michael Gilliland	Azure Machine Learning for the Absolute Beginner Joel Cochran	Real-world microservices with Docker Chris Winters	React Native To The Rescue Josh Gretz
10:30 11:00	Deep Dive Into Chrome's DevTools Gabriel Obregon	Why do I need to know how to write? Ronald Stone		Building "serverless" software with AWS Lambda Jonathan Knapp	Delivering value with distributed scrum teams Gary Greenwood	Rust for the rest of us John Downey	Building A Package For Elixir Onorio Catenacci	Coming Out Of Your Shell: A Comparison of *Nix Shells Kel Cecil	First, Let's Automate All The Lawyers: Smart Contracts With Ethereum Bill Laboon	Of Bandwagons and Builds: Creating a Product with Ember 2.x Eli Flanagan
11:10 12:10	Mind the Gaps...Or Finding the Stories You Need to Build the Product You Want Jeremy Jarrell	Dealing with system failure – Hystrix, Netflix OSS, Spring Cloud and other gems Stuart Ingram	Learn how to make the jump to Angular 2 with Wijmo's JavaScript UI controls! Ross Dederer	Developing Android Apps with Gradle and Groovy Billy Conner	Threads, Processes and the Death of Moore's Law André Henry	**Lunch Part 1**				
12:10 1:10	**Lunch Part 2**					Cross-Browser Testing with Man AND Machines Shawn Summers and Brandon Yee	Connecting UI and UX: how to speak for the user without killing your development Stephanie Butler	Machine Learning (ML) in Healthcare For the Rest of Us Mohinder Dick	Pragmatic Process: Realistic Software Development Methods Richard Goforth	Angular 2 via the CLI Mike Brocchi
1:20 2:20		Integrating Python into the CLR with Python for .NET Hussein Farran	Lessons a tester learned when writing production code Anthony Lamorte	Micro-datacenter chaos monkeys! Raspberry Pi & Kubernetes Steve Sloka	So You Want to be an Agile Coach? Kim Hardy	Release Management with Team Services Paul Hacker	From Evergreen to Evergreat: Leveling up Seattle's Parks with Microcontrollers! Walé Ogundipé	Functional Reactive Programming for Natural User Interface Riccardo Terrell		A Little Bit of Category Theory Goes a Long Way Keith Pinson
2:00 3:00		***Coffee Break - 2:00-3:00***								
	Training Jennifer Davis	How to Hire Programmers	Agile and Automated Testing in a High Pressure	Empathy marketing: why	What Makes a	Software	ElasticSearch For		Revolution [illegible]	Rebooting the

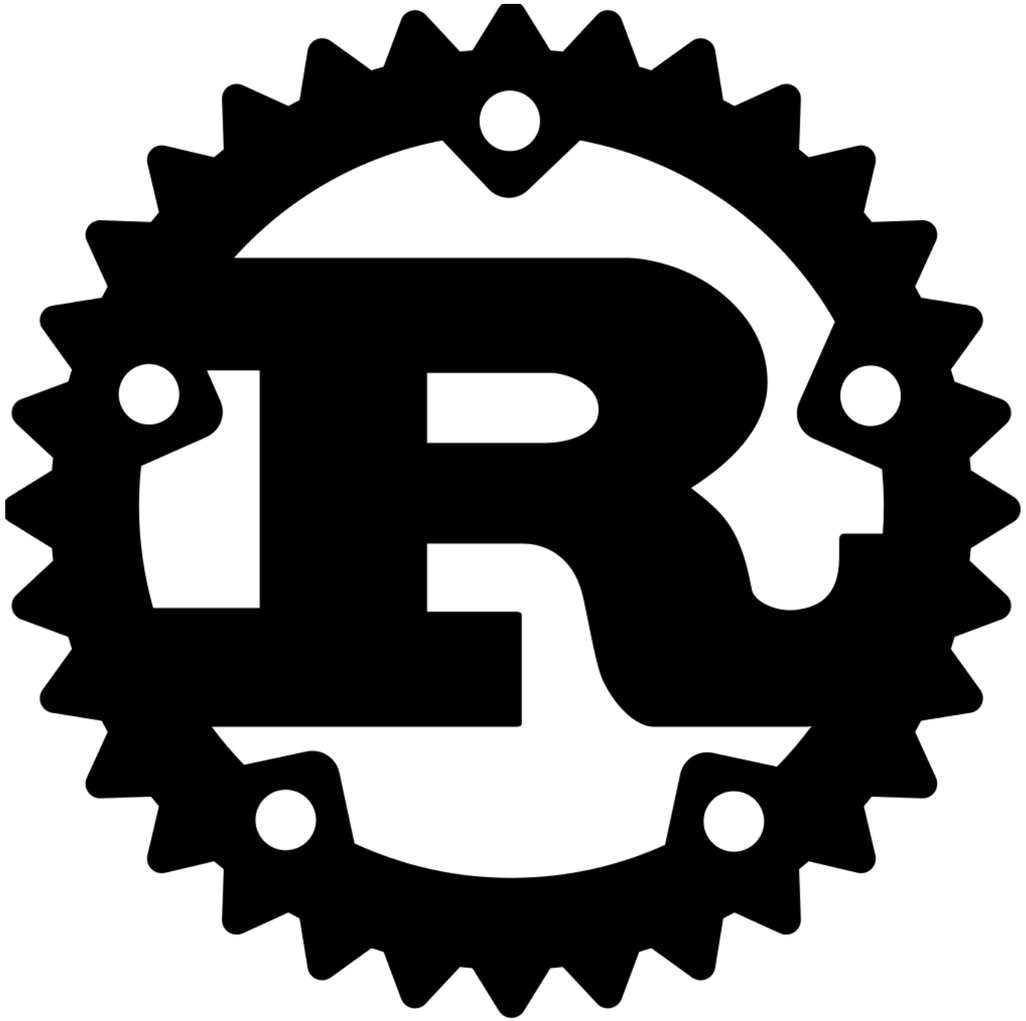
Braintree


PayPal

The views expressed in this presentation are my own, and not those of PayPal or any of its affiliates.

Rust

- Systems language
- From Mozilla Research
- Goals
  - Safe
  - Concurrent
  - Fast



Rust is a systems programming language that runs blazingly fast, prevents segfaults, and guarantees thread safety.

Systems Programming

Systems Programming

- Operating Systems
- Video Games
- Web Browsers

Docs Forum Github News Screenshots

Redox is a Unix-like Operating System written in **Rust**, aiming to bring the innovations of Rust to a modern microkernel and full set of applications

Pull from GitHub

Download ISO (WIP)

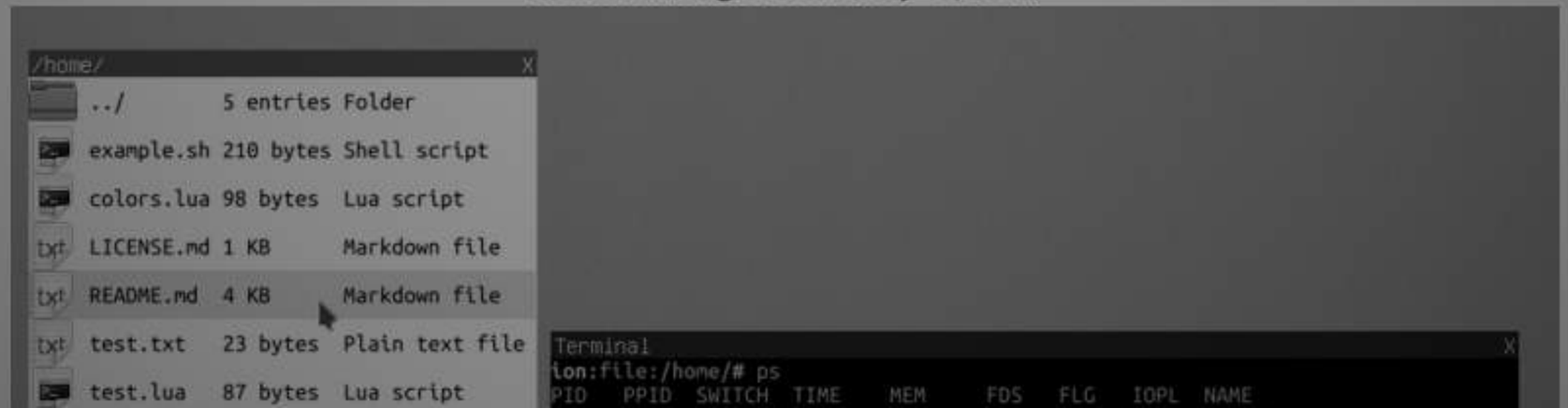
Other Downloads

Redox

- Microkernel Design
- Most features are implemented in Rust
- Includes optional GUI - Orbital
- Ports provided for C programs
- MIT Licensed
- Drivers run in Userspace
- Includes common Unix commands
- ZFS File System Support (WIP)

http://www.redox-os.org/

Redox running Orbital (May 18, 2016)



Piston

http://www.piston.rs

Piston *A modular game engine written in Rust*

Home | Getting Started | Documentation | Blog | Contribute

Quick example

```
extern crate piston_window;

use piston_window::*;

fn main() {
    let mut window: PistonWindow =
        WindowSettings::new("Hello Piston!", [640, 480])
        .exit_on_esc(true).build().unwrap();
    while let Some(e) = window.next() {
        window.draw_2d(&e, |c, g| {
            clear([1.0; 4], g);
            rectangle([1.0, 0.0, [illegible]
                      [0.0, 0.0, [illegible] 100.0
                      c.transform, g);
        });
    }
}
```

More examples »

Piston - a modular open source game engine

The bricks are out of the box! Take a bite of some nice modular libraries for the real world, such as games and interactive applications, written for performance, [illegible] and cross platform development.

Piston was started in 2014 by Sven Nilsen to test back-end agnostic design of 2D graphics in Rust. The project ignited several ambitious projects across the Rust ecosystem. The PistonDevelopers organization is a place where everyone who wants to contribute have write access. This makes it easier to share maintenance, integrate projects and pursue personal goals. Today the Piston project is developing 2D, 3D, [illegible] Visual Studio plugin for Rust, Minecraft clone client/server, sprite animation, AI, and meta parsing for domain specific languages and text formats. We share research and are part of a greater community. You can be a part of it.

Getting started | Source code

The Book of Mozilla, 11:9 - Servo

Mammon slept. And the beast reborn spread over the earth and its numbers grew legion. And they proclaimed the times and sacrificed crops unto the fire, with the cunning of foxes. And they built a new world in their own image as promised by the sacred words, and spoke of the beast with their children. Mammon awoke, and lo! it was [illegible] but [illegible] ever.

from ***The Book of Mozilla,*** *11:9*
(10th Edition)

Servo

https://servo.org/

Fixed in Firefox 35

2015-09	XrayWrapper bypass through DOM objects
2015-08	Delegated OCSP responder certificates failure with id-pkix-ocsp-nocheck extension
2015-07	Gecko Media Plugin sandbox escape
2015-06	Read-after-free in WebRTC
2015-05	Read of uninitialized memory in Web Audio
2015-04	Cookie injection through Proxy Authenticate responses
2015-03	sendBeacon requests lack an Origin header
2015-02	Uninitialized memory use during bitmap rendering
2015-01	Miscellaneous memory safety hazards (rv:35.0 / rv:31.4)

Fixed in Firefox 34

2014-91	Privileged access to security wrapped protected objects
2014-90	Apple CoreGraphics framework on OS X 10.10 logging input data to /tmp directory
2014-89	Bad casting from the BasicThebesLayer to BasicContainerLayer
2014-88	Buffer overflow while parsing media content
2014-87	Use-after-free during HTML5 parsing
2014-86	CSP leaks redirect data via violation reports
2014-85	XMLHttpRequest crashes with some input streams
2014-84	XBL bindings accessible via improper CSS declarations
2014-83	Miscellaneous memory safety hazards (rv:34.0 / rv:31.3)

Fear of C

Rust is not alone in this space

- Go
- D
- Nim

Safety

Immutable by Default

```
fn main() {
    let a = 1;
    let mut b = 1;

    b += 1;
    a += 1;
//  ^ this is an error, `a` isn't mutable
}
```

Checked ranges

```
fn main() {
    let mut buf = [0u8; 10];
    for i in 0..11 {
        buf[i] = 1;
// thread '<main>' panicked at 'index out of
// bounds: the len is 10 but the index is 10'
    }
}
```

Uninitialized memory check

```
fn main() {
    let mut a;

    println!("{}", a);
    // ^ error: use of possibly uninitialized variable: `a`
    a = 5;
    println!("{}", a);
}
```

Ownership, Moving, and Borrowing

```rust
fn my_print(numbers: Vec<u32>) {
    //      ^ taking ownership here

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

    // this is automatically inserted
    // free(numbers); memory freed
}

fn main() {
    let mut numbers = Vec::new(); // memory allocated
    numbers.push(1);
    numbers.push(2);
    numbers.push(3);

    my_print(numbers);

    println!("numbers: {:?}", numbers);
//  ^ this is an error, `numbers` has been moved
}
```

Owned values have one owner and they are dropped when that owner goes out of scope

```rust
fn my_print(numbers: &Vec<u32>) {
    //      ^ borrowing here

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

fn main() {
    let mut numbers = Vec::new(); // memory allocated
    numbers.push(1);
    numbers.push(2);
    numbers.push(3);

    my_print(&numbers);

    println!("numbers: {:?}", numbers);
//  ^ this works, `numbers` was borrowed
}
```

Automatic Memory Management

- Ownership and borrow checking
- Know the lifetime of objects
- Heap memory is freed automatically, no GC!
- All enforced by the compiler

Type System

- No notion of `null`
  - Replaced with `Option<T>` type
  - No null pointer exception
- Errors reported with `Result<T, E>`
  - Choose to handle error or throw it away

Speed

LLVM

- Uses LLVM for code generation
- Allows for cross-compilation

Zero cost abstractions

```rust
use std::hash::{Hash, Hasher, SipHasher};

fn print_hash<T: Hash>(t: T) {
    let mut hasher = SipHasher::new();
    t.hash(&mut hasher);
    println!("The hash is {}", hasher.finish());
}

fn main() {
    print_hash("hello");
    print_hash(42);
}
```

Inlined code

```
#[inline]
fn print_larger(a: i32, b: i32) {
    if a > b {
        println!("{}", a);
    } else {
        println!("{}", b);
    }
}

fn main() {
    print_larger(1, 2);
    print_larger(5, 3);
}
```

Specialization

```
impl<T: fmt::Display + ?Sized> ToString for T {
    #[inline]
    default fn to_string(&self) -> String {
        use core::fmt::Write;
        let mut buf = String::new();
        let _ = buf.write_fmt(format_args!("{}", self));
        buf.shrink_to_fit();
        buf
    }
}
```

Specialization

```rust
#[stable(feature = "str_to_string_specialization", since = "1.9.0")]
impl ToString for str {
    #[inline]
    fn to_string(&self) -> String {
        String::from(self)
    }
}
```

Inline Assembly (feature gated)

```
#![feature(asm)]

#[cfg(any(target_arch = "x86", target_arch = "x86_64"))]
fn add(a: i32, b: i32) -> i32 {
    let sum: i32;
    unsafe {
        asm!("add $2, $1; mov $1, $0" : "=r"(sum) : "r"(a), "r"(b));
    }
    sum
}
```

Single Instruction Multiple Data

```rust
extern crate simd;

use simd::f32x4;

fn main() {
    // create simd vectors
    let x = f32x4::new(1.0, 2.0, 3.0, 4.0);
    let y = f32x4::new(4.0, 3.0, 2.0, 1.0);

    // simd product
    let z = x * y;

    println!("z: {:?}", z);
}
```

Interoperability with C

Rust -> C

Rust <- C

Rust <- {Ruby, Node.js, Python}

RustBridge

https://github.com/rustbridge

Ecosystem

Community

- This week in _____
  - This Week in Rust - http://this-week-in-rust.org/
- Users forum - https://users.rust-lang.org/
- Reddit - http://www.reddit.com/r/rust/
- IRC - `#rust` on `irc.mozilla.org`

Regular release schedule

- Push for backwards compatibility
- Broken down into channels
  - Stable (every 1-2 months)
  - Beta
  - Nightly

Cargo - https://crates.io

- A crate is a unit of compilation (binary/library)
- Mozilla hired developers of Bundler
- Manages dependencies and versions
- Builds code
- Runs tests

Rustup - https://www.rustup.rs/

- Easy to get started
- Allows for tracking the different branches
- Makes cross-compilation simpler
  - `rustup target add ...`

Clippy

https://github.com/Manishearth/rust-clippy

Static Linking

- Rust code will be combined in one binary
  - Will still rely on system libc
- True static linking is possible with musl
  - https://github.com/emk/rust-musl-builder

Downsides/Frustrations 😫

- Ownership system is complicated
- Compiler is smart, but not forgiving
- Built-in test framework can be annoying

The Periodic Table of Rust Types

- Supported as of 1.0.0-alpha
- Provided by the standard library
- Impossible

- `'r` Lifetime
- `K` Trait bounds
- `T...` Function arguments
- `-> U` Function return
- `extern "ABI"` ABI definition

	Immutable Pointer	Mutable Pointer	Owned Pointer	Bare	Unsized
Raw	`*const T` Immutable raw pointer	`*mut T` Mutable raw pointer	*Raw pointers do not have ownership, `*const T` or `*mut T` should be used as appropriate. See also `Unique<T>`.*		
Simple	`&T` Immutable borrowed reference	`&mut T` Mutable borrowed reference	`Box<T>` Owned pointer	`T` Primitive type, struct, enum and so on	
Trait	`&Trait` Immutable borrowed trait object	`&mut Trait` Mutable borrowed trait object	`Box<Trait>` Owned trait object		`Trait` Unsized trait type

Array	`&[T]`
	Immutable borrowed slice
		`&mut [T]`
		Mutable borrowed slice
			`Box<[T]>`
			Owned array
			`Vec<T>`
			Owned growable vector
				`[T; n]`
				Fixed-size array
					`[T]`
					Unsized array type

String	`&str`
	Immutable borrowed string slice
		`&mut str`
		Mutable borrowed string slice (not that useful)
			`Box<str>`
			Owned string (theoretical)
			`String`
			Owned growable string
				Fixed sized storage is impractical for the variable length UTF-8 encoding.	`str`
					Unsized string type

Callable	`Fn(T...) -> U` Closure with immutable environment	`FnMut(T...) -> U` Closure with mutable environment	`FnOnce(T...) -> U` Closure with owned environment	`fn(T...) -> U` Bare function type	

Rust Belt Rust

October 27th & 28th, 2016
Pittsburgh, PA, USA

RUST BELT RUST

October 27th & 28th, 2016

Pittsburgh, PA, USA

http://www.rust-belt-rust.com/

Doubletree Pittsburgh Downtown

Rust Belt Rust is a conference for people of any level of Rust experience-- you're welcome even if you're just interested in Rust!

Getting started

- Book - http://doc.rust-lang.org/book/
- Rust by Example - http://rustbyexample.com/
- Rust Playground - https://play.rust-lang.org/

Questions