Nutrient

Rust for iOS Developers

More Swifty than Swift

Douglas Hill
Nutrient

iOSDevUK
9 September 2026

Nutrient

NSLondon.com

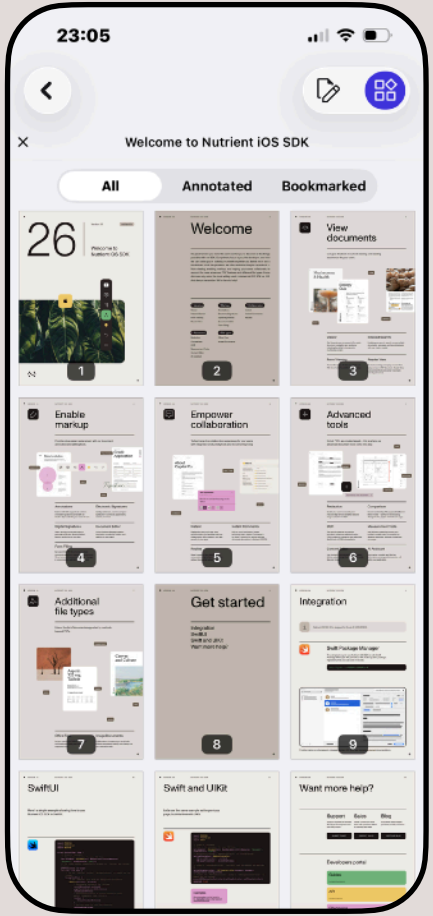





Nutrient

Nutrient.io (PSPDFKit)

The PDF SDK you're looking for



Nutrient

Learning new things

Logo from github.com/rust-lang/rust-artwork

Nutrient

Rust for iOS Developers

More Swifty than Swift

- Introduction
    - Why learn other programming languages?
    - Why Rust?
- Rust and Swift language comparison
    - Arrays
    - Macros
    - Reference types and value types
    - Error handing
- Memory management

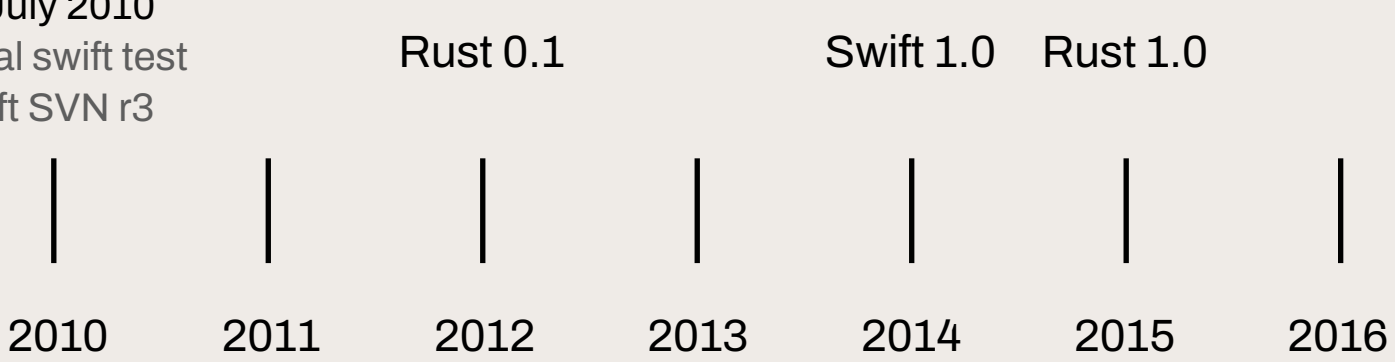
Why learn other programming languages?

Why Rust?

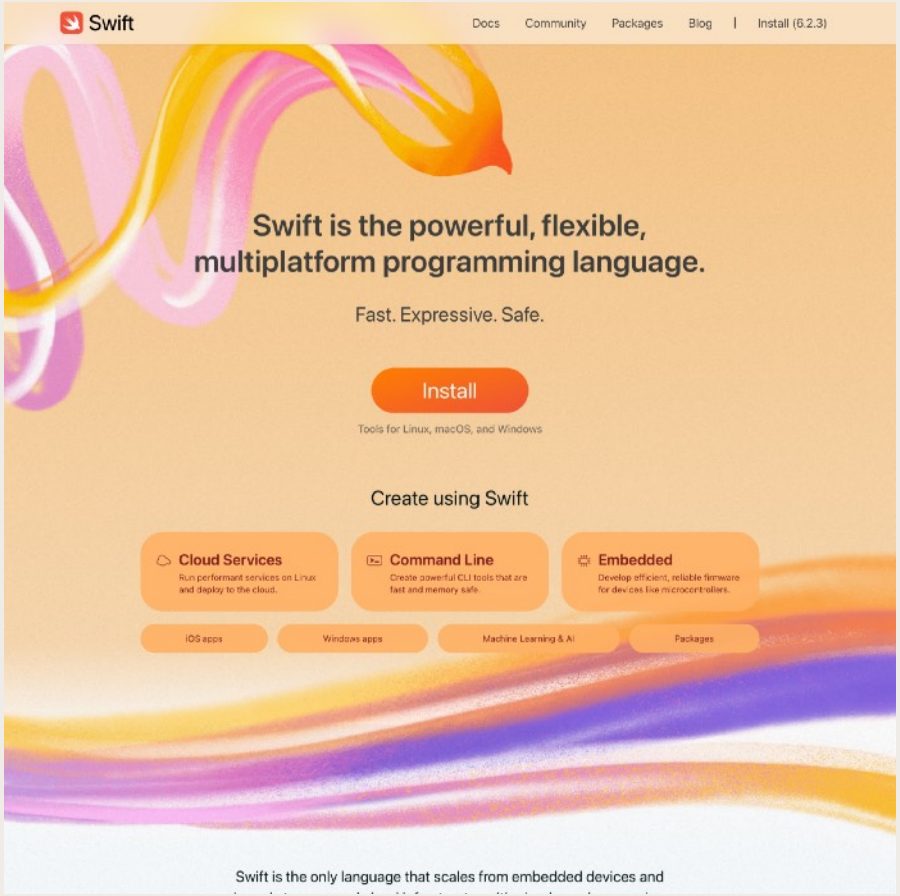
Rust and Swift timeline

`c01efc6`
Graydon Hoare
16 June 2010
Initial git commit.

`18844bc`
Chris Lattner
18 July 2010
initial swift test
Swift SVN r3



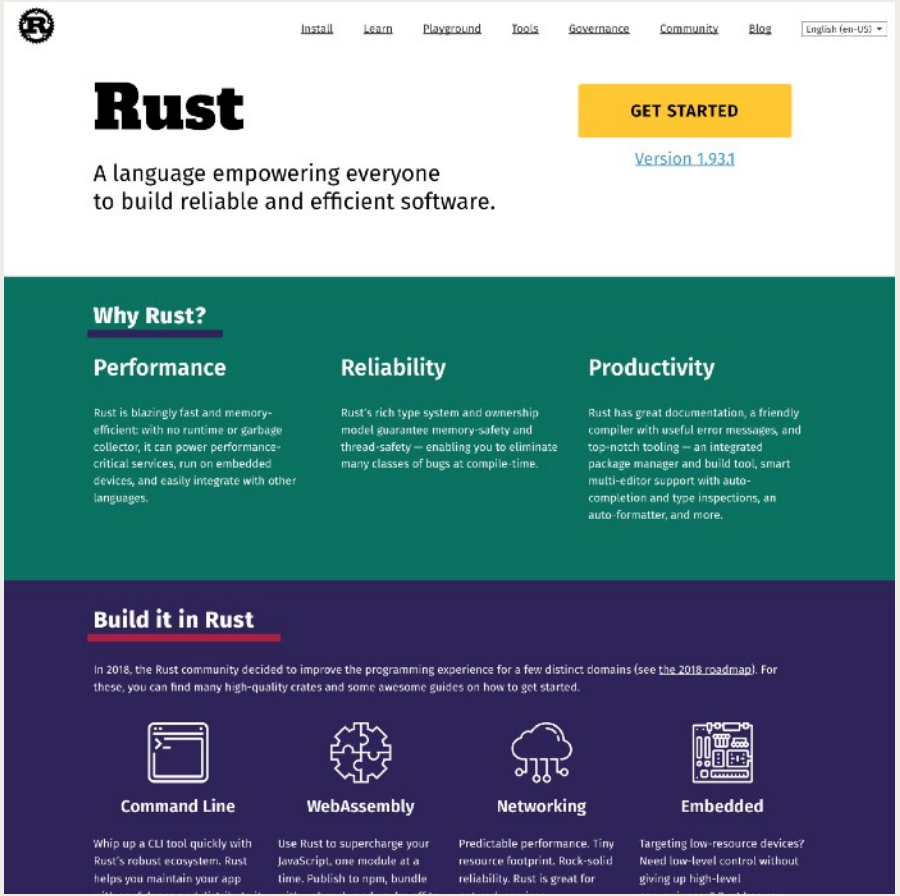
Nutrient



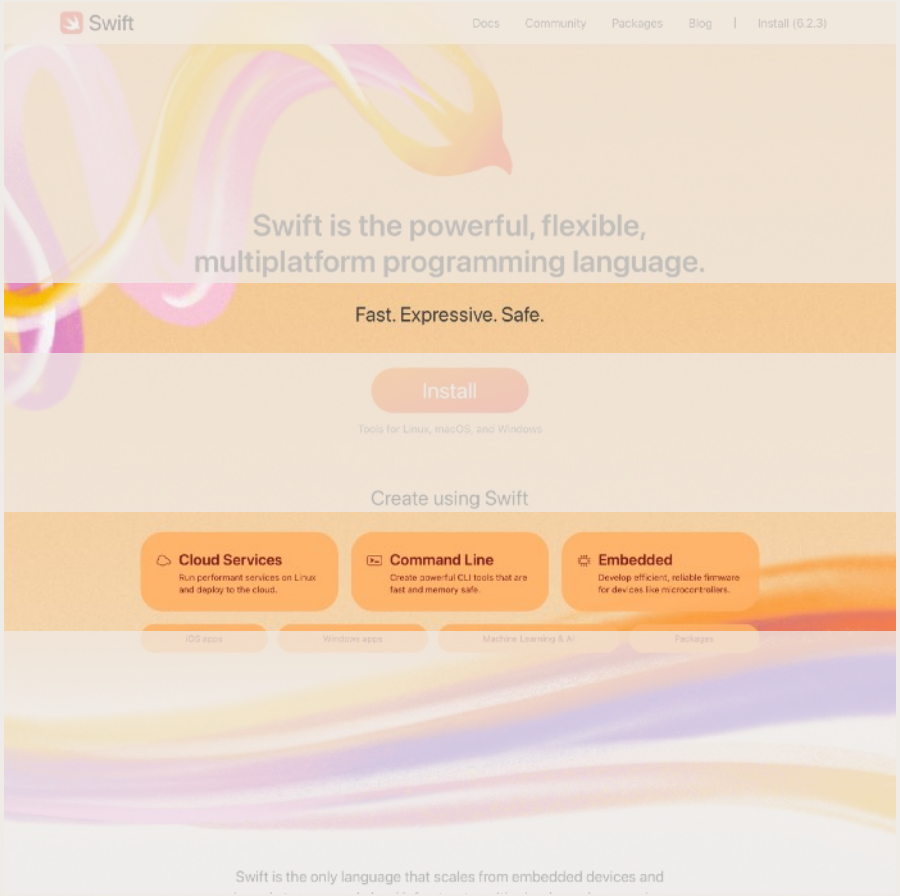
swift.org

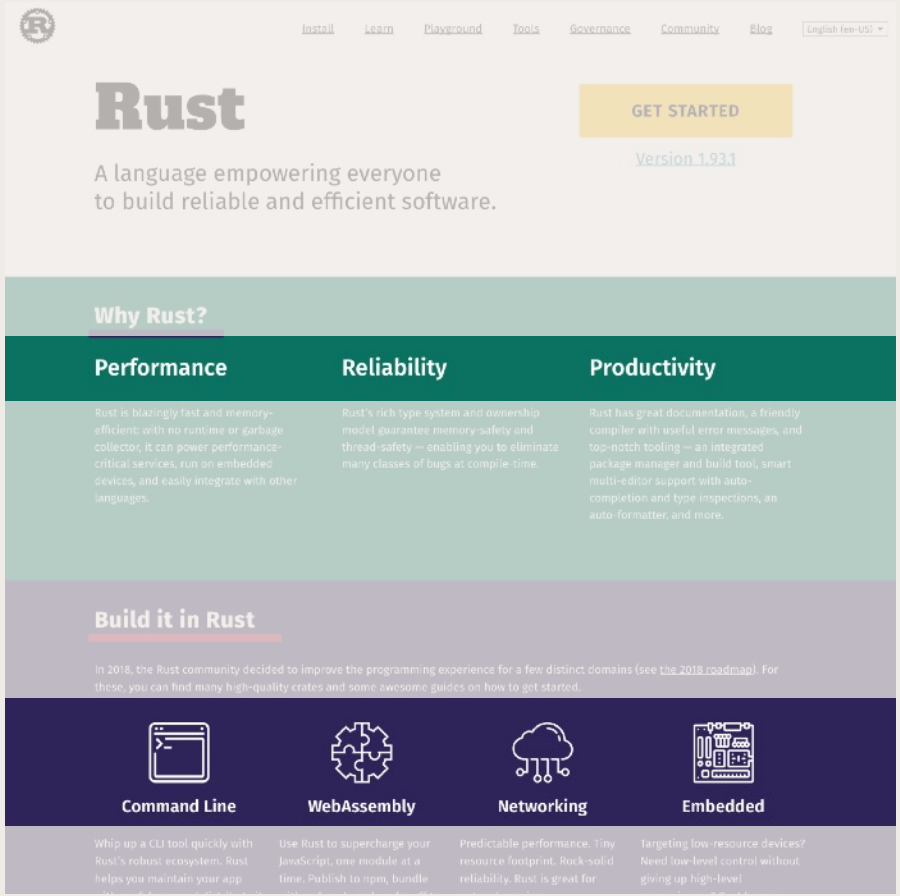
Nutrient

rust-lang.org



Nutrient


Swift
Docs
Community
Packages
Blog
Install (6.2.3)
Swift is the powerful, flexible, multiplatform programming language.
Fast. Expressive. Safe.
Install
Tools for Linux, macOS, and Windows
Create using Swift
Cloud Services
Run performant services on Linux and deploy to the cloud.
Command Line
Create powerful CLI tools that are fast and memory safe.
Embedded
Develop efficient, reliable firmware for devices like microcontrollers.
iOS apps
Windows apps
Machine Learning & AI
Packages
Swift is the only language that scales from embedded devices and


Install
Learn
Playground
Tools
Governance
Community
Blog
English (en-US)
Rust
GET STARTED
Version 1.93.1
A language empowering everyone to build reliable and efficient software.
Why Rust?
Performance
Reliability
Productivity
Rust is blazingly fast and memory-efficient: with no runtime or garbage collector, it can power performance-critical services, run on embedded devices, and easily integrate with other languages.
Rust's rich type system and ownership model guarantee memory-safety and thread-safety — enabling you to eliminate many classes of bugs at compile-time.
Rust has great documentation, a friendly compiler with useful error messages, and top-notch tooling — an integrated package manager and build tool, smart multi-editor support with auto-completion and type inspections, an auto-formatter, and more.
Build it in Rust
In 2018, the Rust community decided to improve the programming experience for a few distinct domains (see the 2018 roadmap). For these, you can find many high-quality crates and some awesome guides on how to get started.
Command Line
WebAssembly
Networking
Embedded
Whip up a CLI tool quickly with Rust's robust ecosystem. Rust helps you maintain your app
Use Rust to supercharge your JavaScript, one module at a time. Publish to npm, bundle
Predictable performance. Tiny resource footprint. Rock-solid reliability. Rust is great for
Targeting low-resource devices? Need low-level control without giving up high-level

Nutrient

Swift Rust

Expressive.

Fast.

Safe.

Performance

Reliability

Productivity

Cloud Services
Run performant services on Linux and deploy to the cloud.

Networking

Command Line
Create powerful CLI tools that are fast and memory safe.

Command Line

Embedded
Develop efficient, reliable firmware for devices like microcontrollers.

Embedded

WebAssembly

Rust is similar to Swift

Rust overview

- Compiled
- Safe
    - type safety
    - memory safety
    - data race safety
- Thinner than Swift
- Progressive disclosure => more explicit

Rust ecosystem

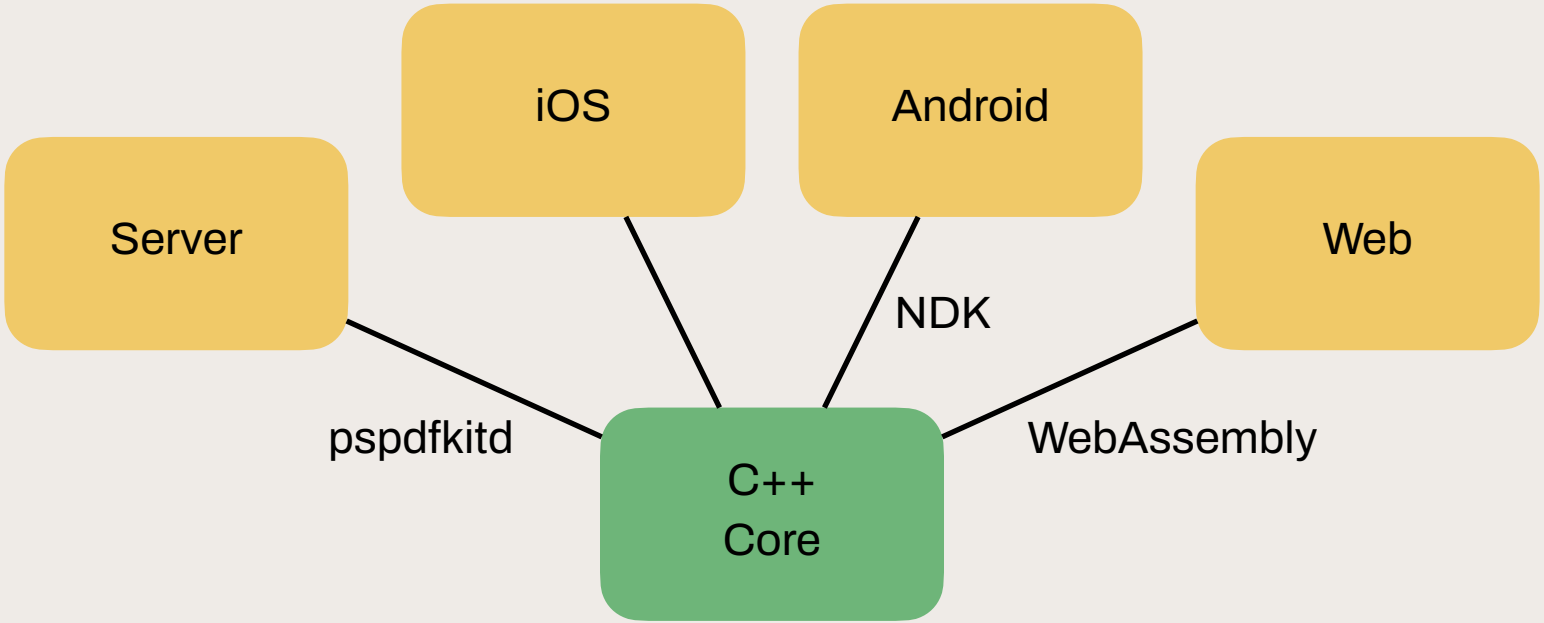
- Packages => **crates**
- Swift Package Manager => **Cargo**
- Swift Package Index => **crates.io** / **docs.rs**

Rust is more established outside of iOS development

Rust on iOS?

Nutrient

Shared code at Nutrient



Language comparison

Similarities between Swift and Rust

`let` and `var`	✔ `var` is `mut let`
Type inference or explicit types	✔
Tuples	✔ no named members
`if`, `else`, `while`, `for`	✔ also `loop`
Exhaustive `switch`	✔ called `match`
Structs	✔ have update syntax
Enums	✔
Protocols	✔ called traits
Generics	✔
`Optional`	✔ called `Option`, no `nil` shorthand
Closures	✔ use |

Macros

Nutrient

Swift

```swift
let year = 2026
print("iOSDevUK \(year)")
```

Rust

```rust
let year = 2026;
println!("iOSDevUK {year}");
```

Swift

```swift
#expect(conference.year == 2026)
```

Rust

```rust
assert_eq!(conference.year, 2026);
```

Nutrient

Swift

```swift
struct Talk: Equatable {
    let title: String
    let duration: Double
}
```

Rust

```rust
#[derive(PartialEq)]
struct Talk {
    title: String,
    duration: f64,
}
```

Swift

```swift
@Observable class Talk {
    let title: String
    let duration: Double
}
```

Rust

```rust
#[derive(PartialEq)]
struct Talk {
    title: String,
    duration: f64,
}
```

Arrays

Nutrient

Swift

```swift
let array = [8, 2]
let first = array[0]
let crash = array[99]

var array = [8, 2]
array.append(5)
```

Rust

```rust
let array = [8, 2];
let first = array[0];
let crash = array[99];

let mut vector = vec![8, 2];
vector.push(5);
```

Nutrient

Swift

```swift
let array: InlineArray = [8, 2]
let first = array[0]
let crash = array[99]

var array = [8, 2]
array.append(5)
```

Rust

```rust
let array = [8, 2];
let first = array[0];
let crash = array[99];

let mut vector = vec![8, 2];
vector.push(5);
```

Nutrient

Reference types

Value types only

- Rust has **no classes** or reference types
- Closest thing to subclassing is traits (like protocols)
- For reference counting, use wrapper types like `Arc<Mutex<T>>`

Error handling

Nutrient

Error handling in Rust

```rust
fn fetch_speakers(year: i32) -> Result<Speakers, ConfError> {
    if year == 2025 {
        return Err(ConfError::Skipped);
    }

    let conference = Conference::new(year)?.speakers();

    Ok(conference)
}
```

Memory management

Nutrient

Reference types in Swift

```
let conf = Conference(name: "iOSDevUK") // This is a class
let otherConf = conf // Cheap pointer copy + ARC
```

Value types in Swift

```
var count = 5
var otherCount = count // Cheap value copy
```

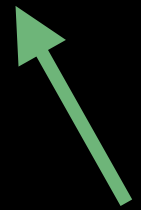
Value types in Swift

```
var string = "Really big string"
var otherString = string
string.append("more text") // Possibly expensive copy
```

Nutrient

Noncopyable types in Swift

```
struct NoncopyableString: ~Copyable {
    let value: String
}
```



Nutrient

Noncopyable types in Swift

```
struct NoncopyableString: ~Copyable {
    let value: String
}

let noncopyableString = NoncopyableString(value: "Really big string")
let other = noncopyableString

print("String is \(noncopyableString.value).")
```

❌ used after consume

Nutrient

Noncopyable parameters in Swift

```
struct NoncopyableString: ~Copyable {
    let value: String
}

func doSomething(with noncopyableString: NoncopyableString) {
}
```

parameter of noncopyable type must specify ownership

(consuming, borrowing, or inout)

Nutrient

Noncopyable parameters in Swift

```
struct NoncopyableString: ~Copyable {
    let value: String
}

func doSomething(with noncopyableString: consuming NoncopyableString) {
}
```

Nutrient

Noncopyable parameters in Swift

```
struct NoncopyableString: ~Copyable {
    let value: String
}

func doSomething(with noncopyableString: consuming NoncopyableString) {
}

let noncopyableString = NoncopyableString(value: "Really big string")
doSomething(with: nonCopyableString)
print("String is \(noncopyableString.value).")
```

❌ used after consume

Nutrient

Noncopyable parameters in Swift

```
struct NoncopyableString: ~Copyable {
    let value: String
}

func doSomething(with noncopyableString: borrowing NoncopyableString) {
}

let noncopyableString = NoncopyableString(value: "Really big string")
doSomething(with: nonCopyableString)
print("String is \(noncopyableString.value).")
```

Nutrient

Noncopyable parameters in Swift

```
struct NoncopyableString: ~Copyable {
    let value: String
}

func doSomething(with noncopyableString: inout NoncopyableString) {
}

let noncopyableString = NoncopyableString(value: "Really big string")
doSomething(with: nonCopyableString)
print("String is \(noncopyableString.value).")
```

Nutrient

Ownership and borrowing

Ownership in Rust

```
let string = String::from("Really big string");
let other_string = string;

println!("The string is {string}.");
```

borrowed after move

Nutrient

Ownership in Rust

```
let string = String::from("Really big string");
let other_string = string.clone();

println!("The string is {string}.");
```

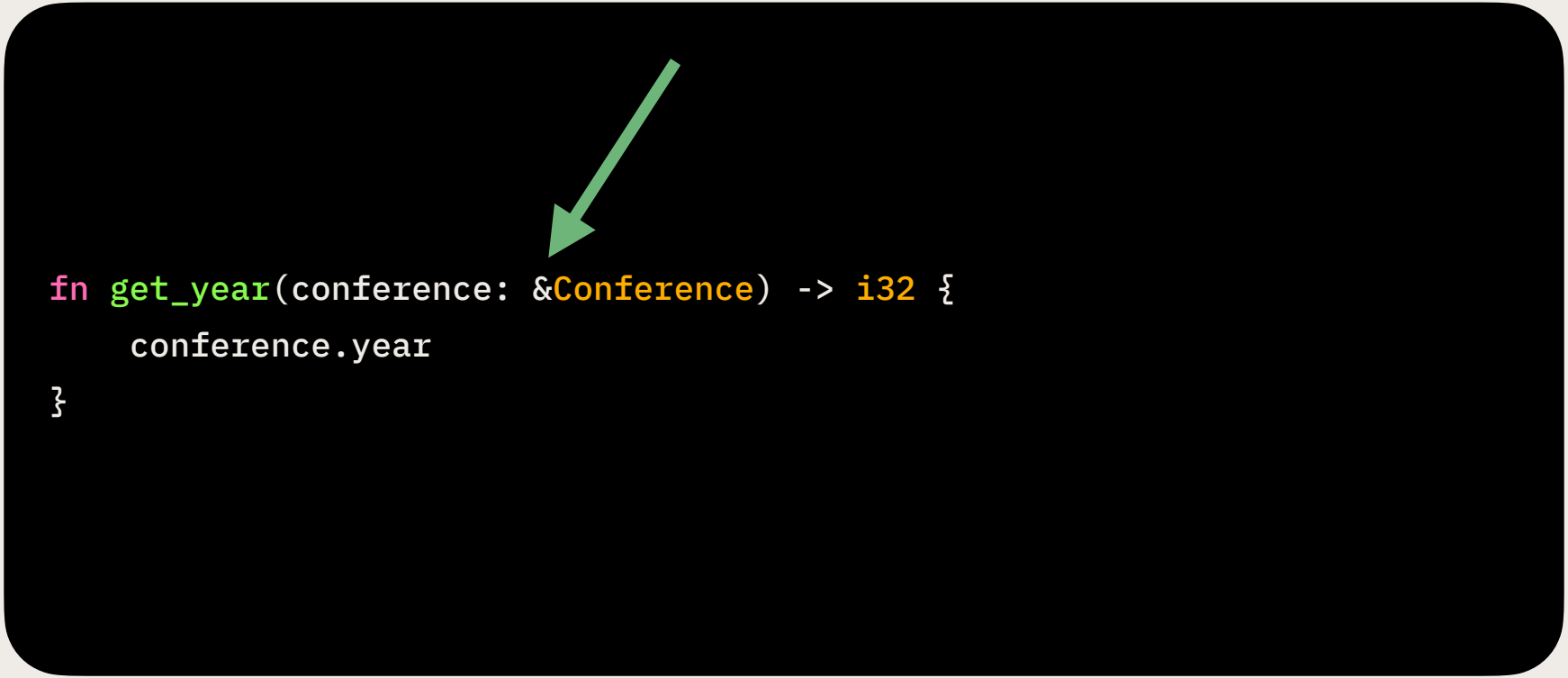
Nutrient

Ownership in Rust

```
fn debug_print(conference: Conference) {
    println!("The conference is {conference:?}.");
}
```

Nutrient

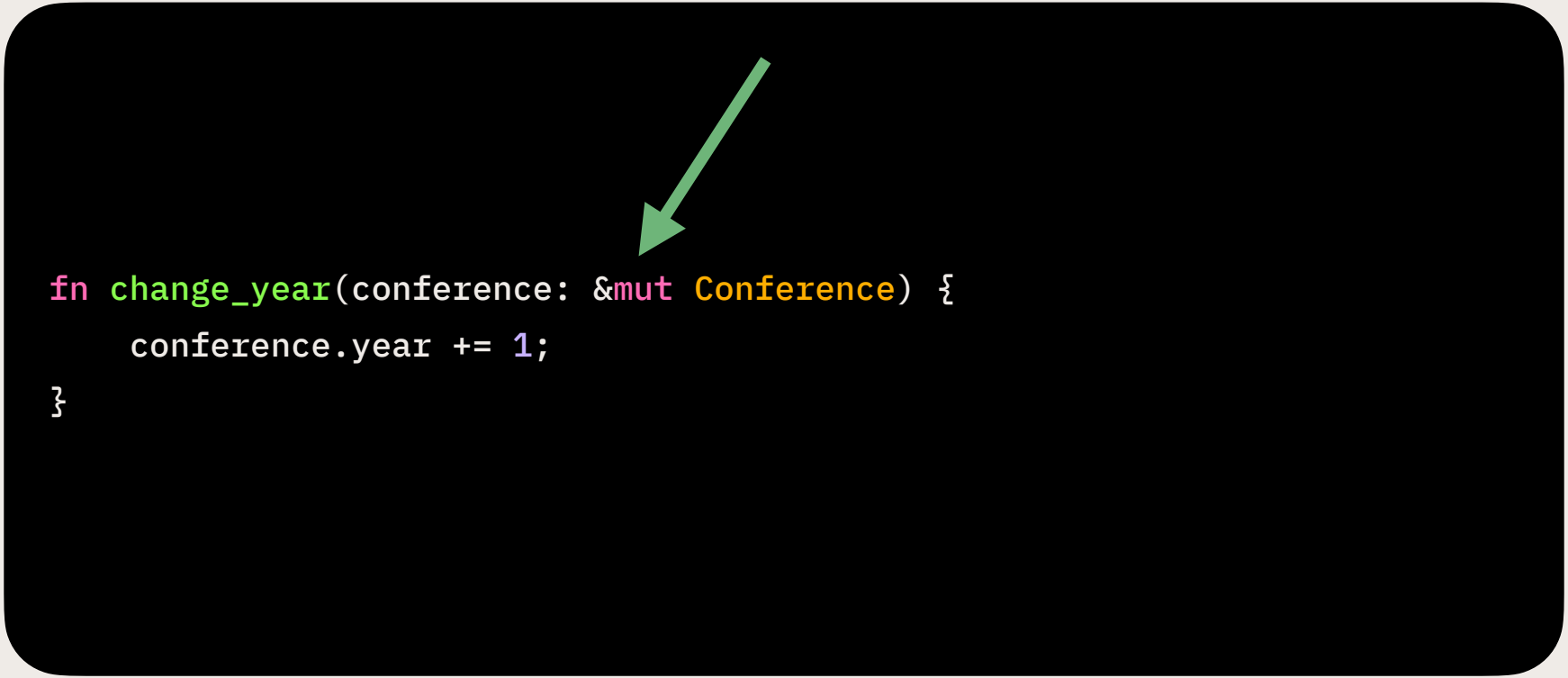
Borrowing in Rust



```rust
fn get_year(conference: &Conference) -> i32 {
    conference.year
}
```

Nutrient

Mutable borrowing in Rust



```rust
fn change_year(conference: &mut Conference) {
    conference.year += 1;
}
```

Nutrient

Ownership and borrowing

Closing

- Rust and Swift have a lot of similarities
    - Swift is sometimes Rusty
    - Rust is sometimes more Swifty than Swift
- Next steps:
    - Rust book
    - Build slowly with AI