

CPP-Summit 2020

张汉东

《Rust编程之道》作者
企业独立咨询顾问

Rust系统级开发 的优势与劣势

议程

1

自我介绍

一个简单的自我介绍

2

时代视角下的系统级开发

从不同时代的角度去看待系统级开发的变迁

3

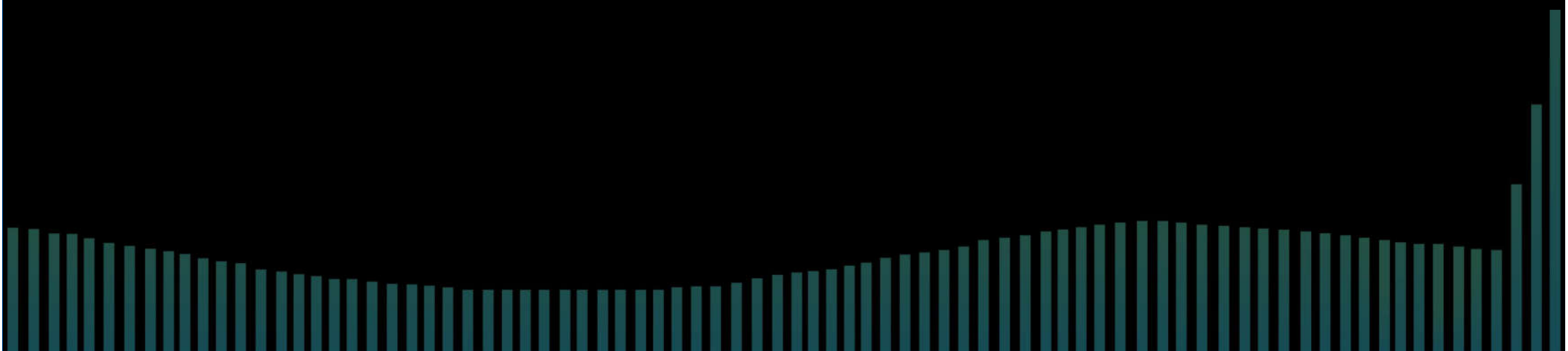
Rust 语言的兴起

介绍 Rust 语言

4

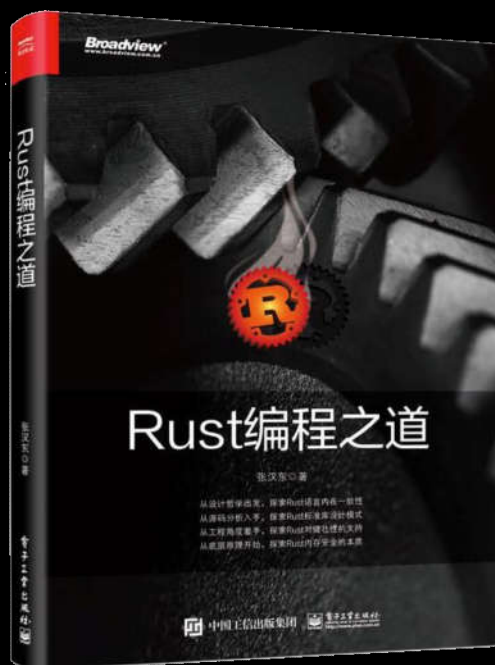
Rust 语言的优势与劣势

高屋建瓴式分析 Rust 语言的优势与劣势



01

自我介绍



《Rust 编程之道》作者
企业独立咨询顾问

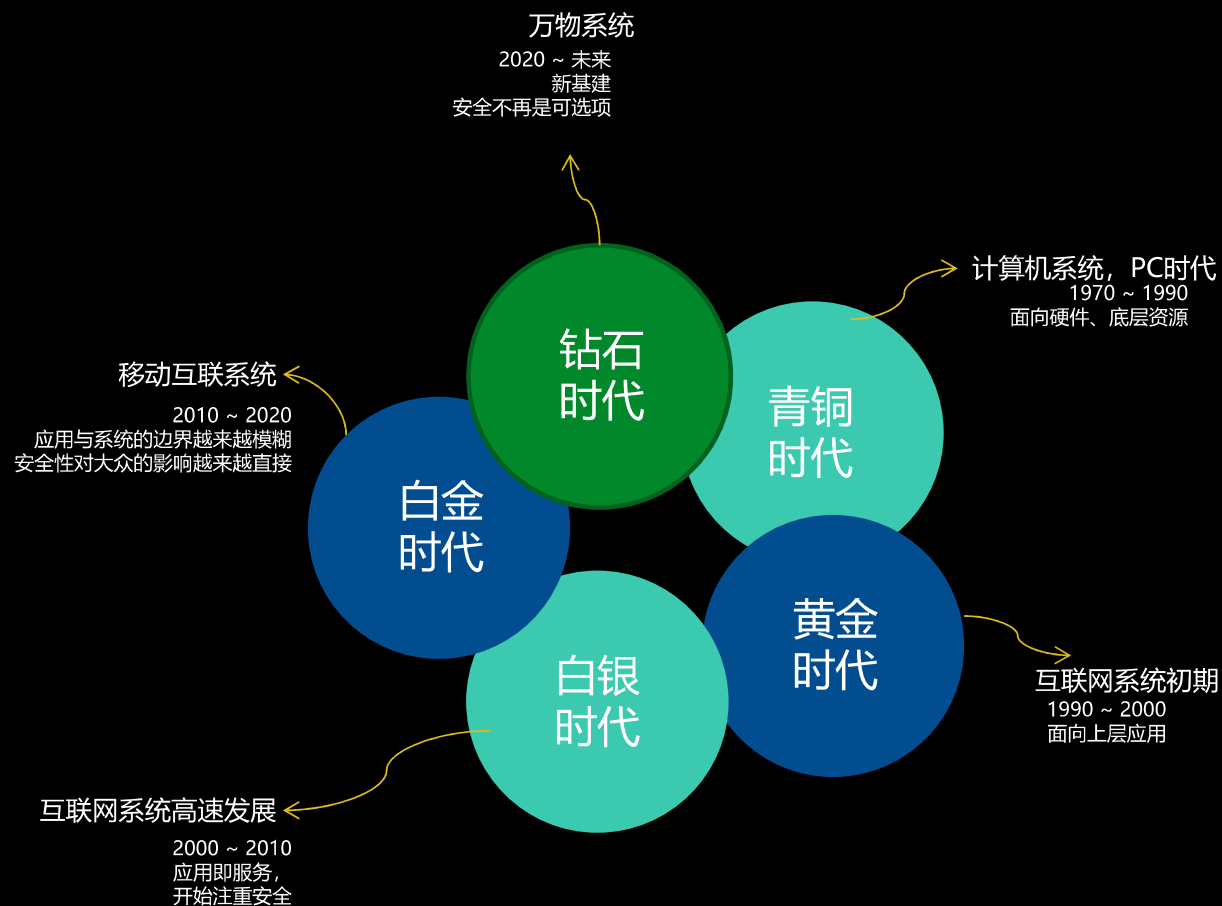
Rust 中文社区联合发起者

布道 Rust 过程中的小烦恼：

喜欢我的人调侃我 “Rust 之父”
不喜欢我的人喷我 “Rust 之父”

02

时代视角下 系统级开发



语言大师们的看法



LangNext 2014 (C++, Rust, D, Go)

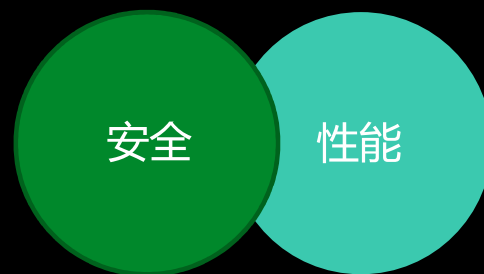
当下什么是系统级开发

能直接面向硬件，解决底层资源调度和性能问题

能保证基本的安全性，尽最大可能避免系统安全漏洞

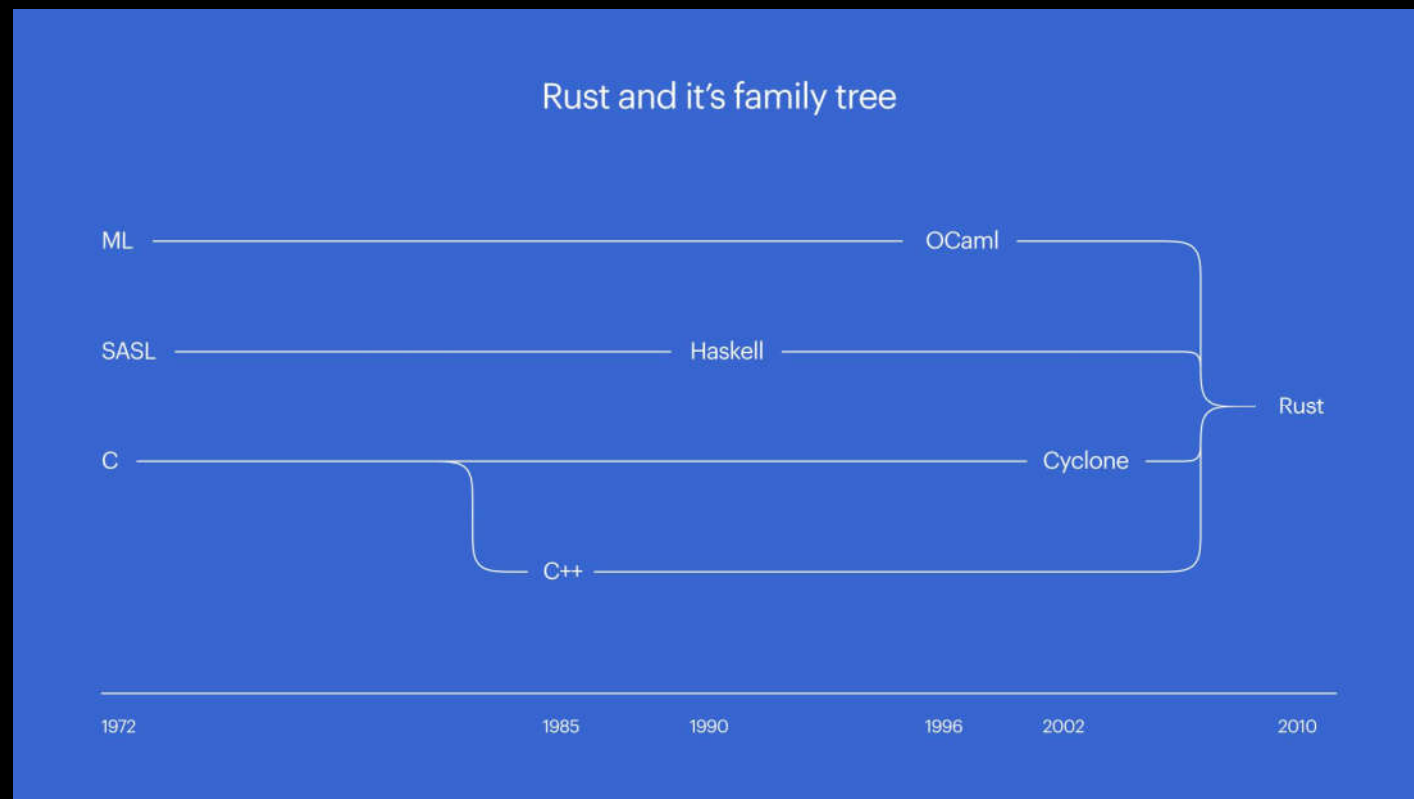
能保证系统的健壮性，保证程序中的错误都得到合理的处理

能保证系统灵活的可扩展性和易维护性，保证系统可以长期稳定提供服务



03

Rust 语言的兴起



04

Rust 语言 的优势与劣势

1

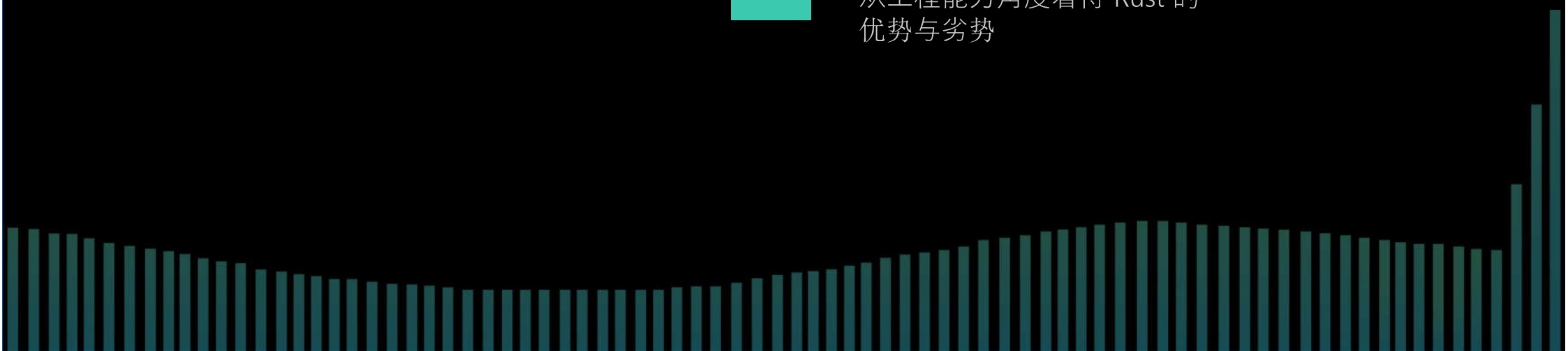
语言设计

从语言设计角度看待 Rust 的优势与劣势

2

工程能力

从工程能力角度看待 Rust 的优势与劣势



语言设计

1

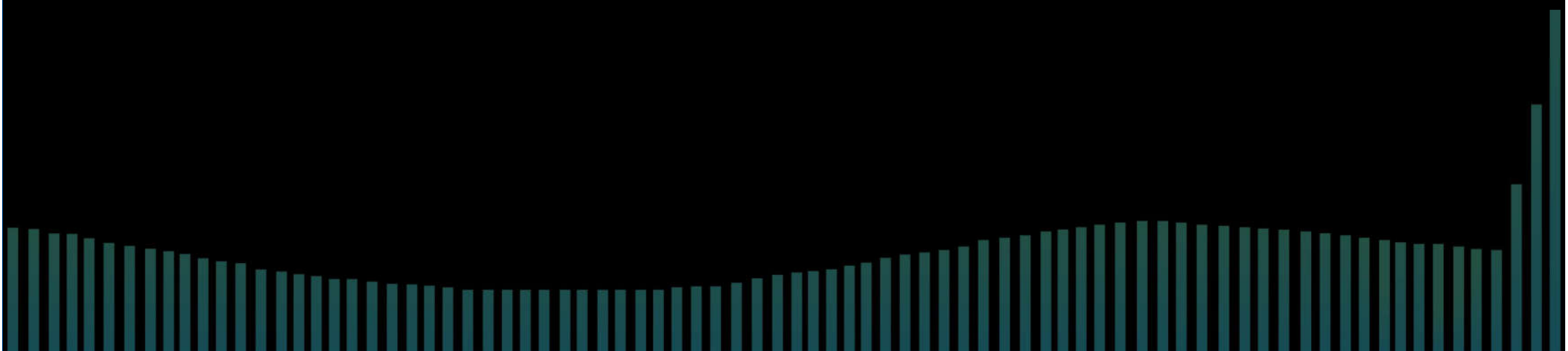
类型系统

虽然类型安全，但类型系统还不够完善

2

安全模型

所有权机制保证安全，但带来心智负担



语言设计

类型系统·角色

所有权语义

工程能力

类型系统

资源管理



语言设计

类型系统·优势

一切皆类型

简洁的内核：类型 + 行为

强大的静态类型检查

支持零成本抽象

基于类型的多范式特性

```
fn main(){
    let s = "abc?d";

    let mut chars = s.chars().collect::<Vec<char>>();

    for (i, c) in chars.iter_mut().enumerate() {
        let mut words = ('a'..'z').into_iter();

        if chars[i] == '?' {
            let left = if i==0 {None} else { Some(chars[i-1]) };

            let right = if i==s.len()-1 {None} else {Some(chars[i+1])};

            chars[i] = words.find(
                |&w| Some(w) != left && Some(w) != right
            ).unwrap();
        }
    }

    let s = chars.into_iter().collect::<String>();
    println!("{:?}", s);
}
```

语言设计

类型系统•劣势

高阶类型不支持,
表达力不足

不支持泛型特化

CTFE 支持不够完善

```

○○○

#![feature(specialization)]

trait Count {
    fn count(self) -> usize;
}

impl<T> Count for T {
    default fn count(self) -> usize {
        1
    }
}

impl<T> Count for T
where
    T: IntoIterator,
    T::Item: Count,
{
    fn count(self) -> usize {
        let i = self.into_iter();

        i.map(|x| x.count()).sum()
    }
}

fn main() {
    let v = vec![1, 2, 3];
    assert_eq!(v.count(), 3);

    let v = vec![vec![1, 2, 3], vec![4, 5, 6]];
    assert_eq!(v.count(), 6);
}

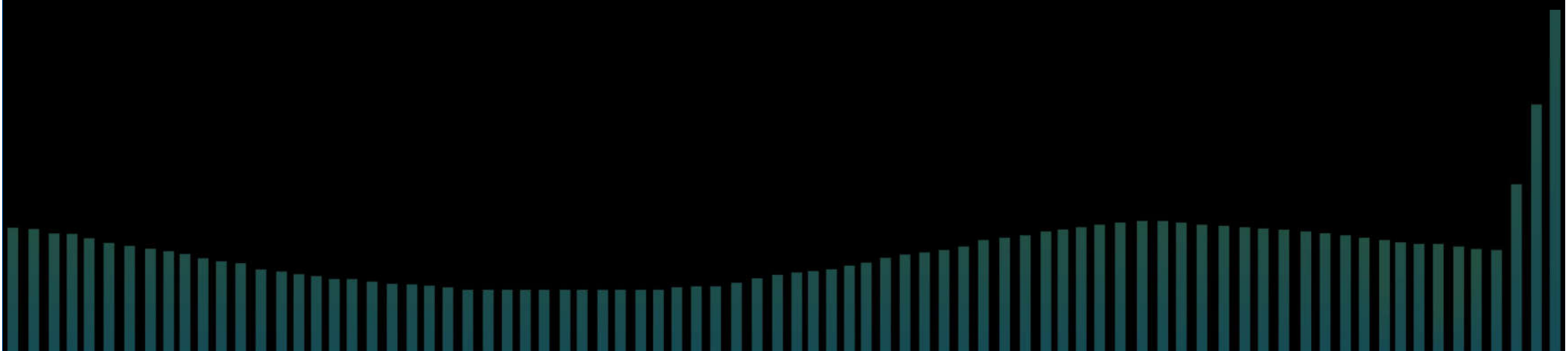
```

语言设计

安全模型·优点

高度一致性的所有权语义模型

Safe 与 Unsafe 界限分明



语言设计

安全模型·优点

高度一致性的所有权语义模型

Safe 与 Unsafe 界限分明

○○○

```
use std::thread;
fn main() {
    let mut s = "Hello".to_string();
    for _ in 0..3 {
        thread::spawn(move || {
            s.push_str(" Rust!");
        });
    }
}
```

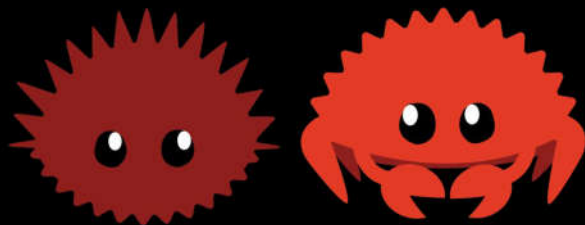
in Tuple

语言设计

安全模型·优点

高度一致性的所有权语义模型

Safe 与 Unsafe 界限分明



```
    ) {  
  
    /// Safety:  
    /// 小心传入的 index 超过 数组长度导致数组越界  
    pub unsafe fn insert(&mut self, index: usize, element: T) {  
        let len = self.len();  
  
        // 通过判断长度是否达到容量极限来决定是否进行扩容  
        if len == self.buf.cap() {  
            self.reserve(1);  
        }  
        unsafe {  
            {  
                let p = self.as_mut_ptr().offset(index as isize);  
                ptr::copy(p, p.offset(1), len - index);  
                ptr::write(p, element);  
            }  
            self.set_len(len + 1);  
        }  
    }  
}
```

语言设计

安全模型·劣势

学习曲线高

开发者会有心智负担

○ ○ ○

```
fn main() {
    let v = vec![1,2,3, 4, 5,6];
    let mut buf = Buffer::new(&v);
    let b1 = buf.read_bytes();
    let b2 = buf.read_bytes();
    print(b1,b2)
}

fn print(b1 :&[u8], b2: &[u8]) {
    println!("{:#?} {:#?}", b1, b2)
}

struct Buffer<'a> {
    buf: &'a [u8],
    pos: usize,
}

impl<'a> Buffer<'a> {
    fn new(b: &'a [u8]) -> Buffer {
        Buffer {
            buf: b,
            pos: 0,
        }
    }

    fn read_bytes(&mut self) -> &'a [u8] {
        self.pos += 3;
        &self.buf[self.pos-3..self.pos]
    }
}
```

```
&'a [u8] {
}
```

it work

工程能力

1

可扩展性和易维护性

Rust 语言天生具备可扩展性和易维护性

2

健壮性

Rust 具有优雅的错误处理方式

3

简洁的抽象方式

Rust 是混合范式语言，但抽象方式和 C 语言一样简洁

4

现代化工具链

Rust 现代化的工具链，就是为工程而生

工程能力

可扩展性和易维护性

Rust 天生面向接口编程

```
enum Knob {
    Linear(LinearKnob),
    Logarithmic(LogarithmicKnob),
}

impl KnobControl for Knob {
    fn set_position(&mut self, value: f64) {
        match self {
            Knob::Linear(inner_knob) => inner_knob.set_position(value),
            Knob::Logarithmic(inner_knob) => inner_knob.set_position(value),
        }
    }

    fn get_value(&self) -> f64 {
        match self {
            Knob::Linear(inner_knob) => inner_knob.get_value(),
            Knob::Logarithmic(inner_knob) => inner_knob.get_value(),
        }
    }
}
```

```
//use the knobs
}
```

工程能力

可扩展性和易维护性

Rust 天生面向接口编程

显式哲学，几乎无隐式行为

```
○○○

trait KnobControl {
    fn set_position(&mut self, value: f64);
    fn get_value(&self) -> f64;
}

struct LinearKnob {
    position: f64,
}

struct LogarithmicKnob {
    position: f64,
}

impl KnobControl for LinearKnob {
    fn set_position(&mut self, value: f64) {
        self.position = value;
    }

    fn get_value(&self) -> f64 {
        self.position
    }
}

impl KnobControl for LogarithmicKnob {
    fn set_position(&mut self, value: f64) {
        self.position = value;
    }

    fn get_value(&self) -> f64 {
        (self.position + 1.).log2()
    }
}

fn main() {
    let v: Vec<Box<dyn KnobControl>> = vec![];
    //set the knobs

    //use the knobs
}
```

工程能力

健壮性

失败 (Failure)

错误 (Error)

恐慌 (Panic)



```
use std::panic;

fn main() {
    let result = panic::catch_unwind(
        || { println!("hello!"); }
    );
    assert!(result.is_ok());
    let result = panic::catch_unwind(
        || { panic!("oh no!"); }
    );
    assert!(result.is_err());
    println!("{}", sum(1, 2));
}
```

工程能力

简洁的抽象方式

基于类型设计，OOP和FP只是语言特性，不需要纠结用哪个编程范式

```
trait Colorize {
    fn red(self) -> ColoredString;
    fn on_yellow(self) -> ColoredString;
}

impl<'a> Colorize for ColoredString {
    fn red(self) -> ColoredString {
        ColoredString { fgcolor: String::from("31"), ..self }
    }
    fn on_yellow(self) -> ColoredString {
        ColoredString { bgcolor: String::from("43"), ..self }
    }
}

impl<'a> Colorize for &'a str {
    fn red(self) -> ColoredString {
        ColoredString {
            fgcolor: String::from("31"),
            input: String::from(self),
            ..ColoredString::default()
        }
    }
    fn on_yellow(self) -> ColoredString {
        ColoredString {
            bgcolor: String::from("43"),
            input: String::from(self),
            ..ColoredString::default()
        }
    }
}

impl fmt::Display for ColoredString {
    fn fmt(&self, f: &mut fmt::Formatter) -> fmt::Result {
        let mut input = &self.input.clone();
        try!(f.write_str(&self.compute_style()));
        try!(f.write_str(input));
        try!(f.write_str("\x1B[0m"));
        Ok(())
    }
}

fn main() {
    let hi = "Hello".red().on_yellow();
    println!("{}", hi);
    let hi = "Hello".on_yellow();
    println!("{}", hi);
    let hi = "Hello".red();
    println!("{}", hi);
    let hi = "Hello".on_yellow().red();
    println!("{}", hi);
}
```

工程能力

简洁的抽象方式

基于类型设计，OOP和FP只是语言特性，不需要纠结用哪个编程范式

```
- let mut started = false;
-
+ let mut ssload = State::init() ;
+
+ if let Some(_) = () {
+     self.restore_heap()?;
-     started = true;
+     ssload.start();
+ } else {
+     self.
+ }
@@
- if self.get_start_func()?.is_some() && started == false {
+ if self.get_start_func()?.is_some() && ssload.no_started() {
+     self.state = State::NotStarted;
+ } else {
+     self.state = State::Ready;
```

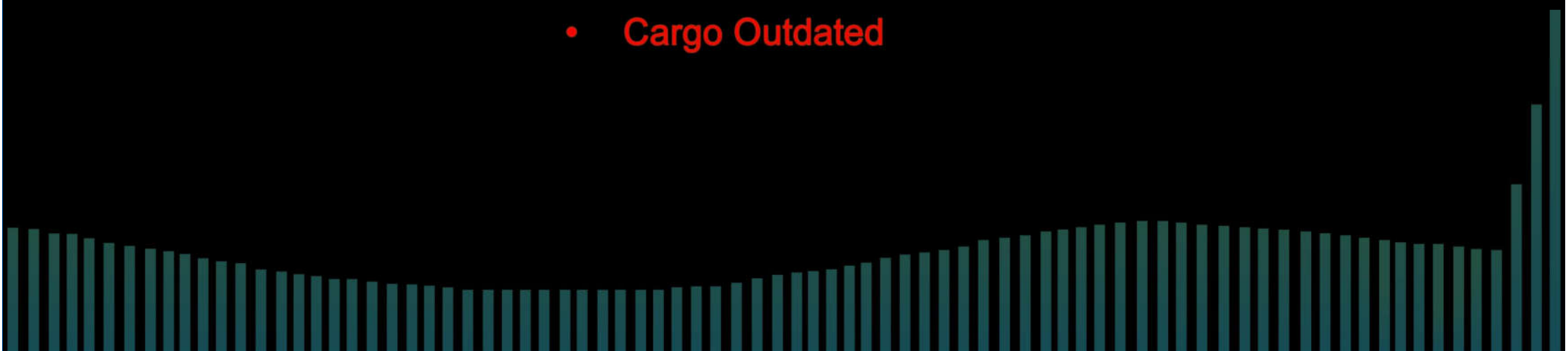
工程能力

现代化工具链

强大的 Cargo 包管理器与模块化支持

高质量的第三方库

- Cargo Clippy
- Cargo Audit
- Cargo Deny
- Cargo Outdated
- Cargo Expand
- Cargo Bloat

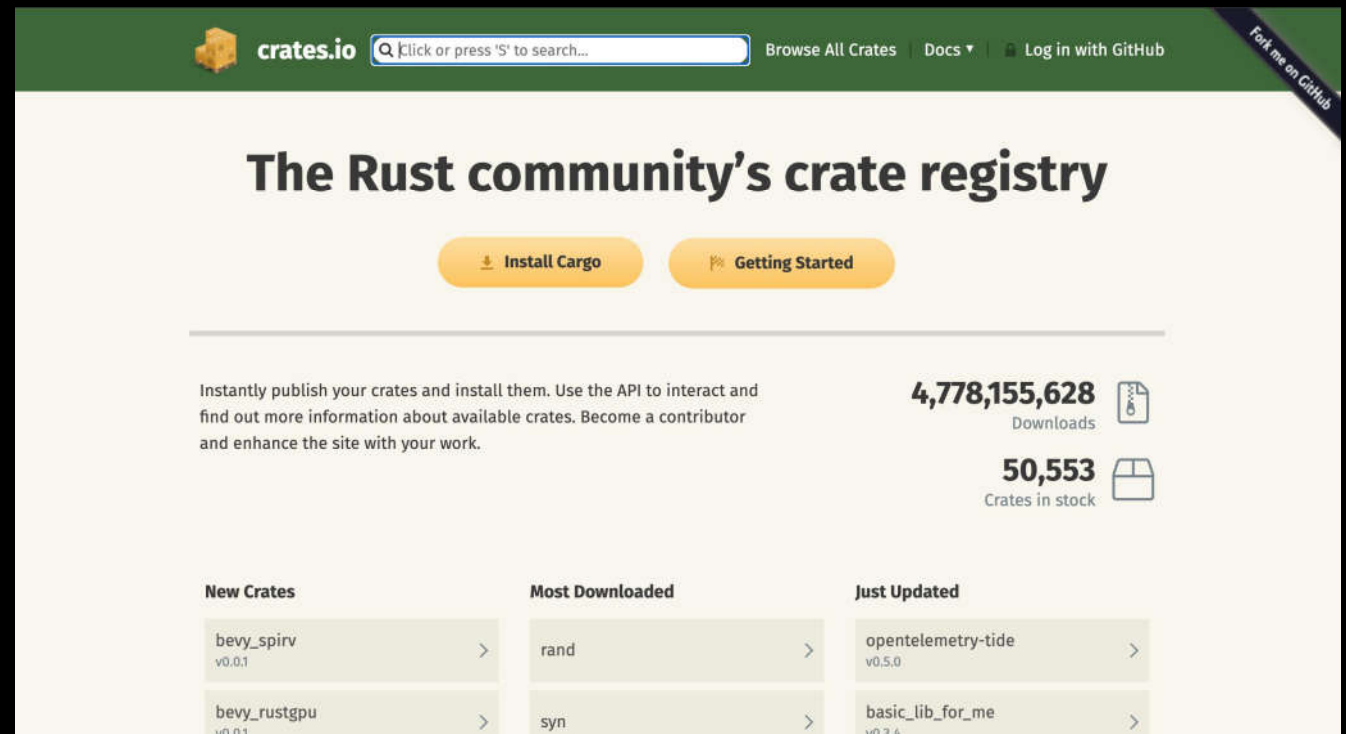


工程能力

现代化工具链

强大的 Cargo 包管理器与模块化支持

高质量的第三方库(crates.io)





crates.io [Browse All Crates](#) [Docs](#) [Log in with GitHub](#) [Fork me on GitHub](#)

The Rust community's crate registry

[Install Cargo](#) [Getting Started](#)

Instantly publish your crates and install them. Use the API to interact and find out more information about available crates. Become a contributor and enhance the site with your work.

4,778,155,628 Downloads 

50,553 Crates in stock 

New Crates	Most Downloaded	Just Updated
bevy_spirv v0.0.1	rand	opentelemetry-tide v0.5.0
bevy_rustgpu v0.0.1	syn	basic_lib_for_me v0.3.4

工程能力

劣势

编译速度慢

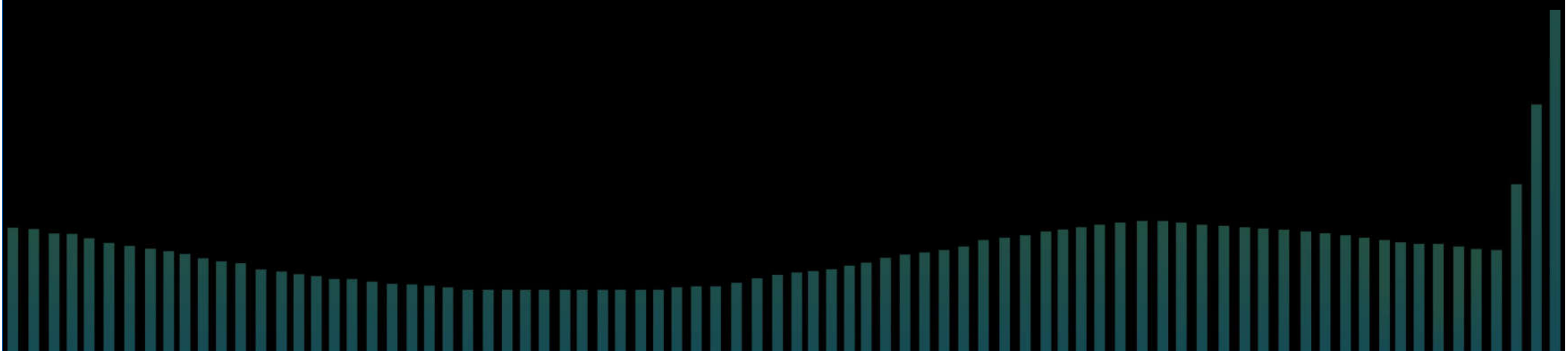
产出的文件尺寸较大

Unsafe Rust UB 检查不够完善

生态中开箱即用的库不多

IDE 支持不够完善

Rust 专用的调试工具不够完善



工程能力

没有展开提到的 Rust 其他优势

出众的元编程能力

跨平台支持

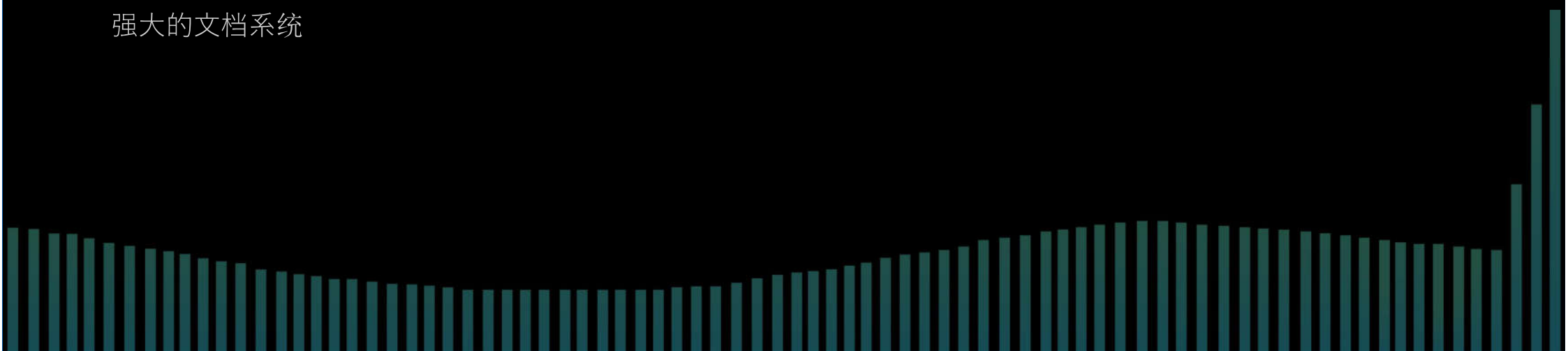
安全易用的异步编程模型

强大的文档系统

成熟的开源社区

无缝与C-ABI 接口调用

内建单元测试和性能测试支持



谢谢

