A Case for Rust

Meet Tony

What is Rust

Memory safety

Speed

Productivity

Ergonomics

Compile Times

Correctness

What you can’t do

```rust
fn main() {
    let v = 0;
    v = v + 1;
}
```

No mutability by default

```rust
fn main() {
    let mut v = 0;
    let x = &v;
    let y = &mut v;

    println!("{}", x);
}
```

No mutable aliasing

```rust
fn main() {
    let f = Foo::new();
    drop(f);
    bar(f);
}
```

No use after free

```rust
fn main() {
    let x: String;
    println!("{}", x);
}
```

No invalid memory

```
fn main() {
    let val, _ = may_fail();
    val.use();
}
```

No forgotten errors

```rust
fn main() {
    let f = ThreadUnsafe::new();

    std::thread::spawn(||{
        println!("{:?}", f);
    });
}
```

No thread unsafety

What you can do

```rust
pub struct Indexed<K, V> {
    indexes: HashMap<K, usize>,
    items: Vec<V>,
}
```

Generics

```rust
#[cfg(test)]
mod tests {
    #[test]
    fn it_works() {
        let result = 2 + 2;
        assert_eq!(result, 4);
    }
}
```

```
user@host$ cargo test
```

Write Tests

```
/// # Examples
///
/// ```
/// let x = 5;
/// ```

user@host$ cargo test
```

Write Tests

```rust
pub fn lookup<'b>(
    &self,
    id: &ID<'b>,
) -> Option<&Symbol>
where 'b: 'a
{
    self.table.iter()
        .rev()
        .find_map(|s| s.get_info(id))
}
```

Declarative Programming

```rust
#[derive(Parser)]
#[clap(author, version)]
struct Args {
    /// Name of the person to greet
    #[clap(short, long, value_parser)]
    name: String,

    /// Number of times to greet
    #[clap(short, long, value_parser,
default_value_t = 1)]
    count: u8,
}
```

Declare CLI arguments

```rust
use serde_yaml::to_string;

#[derive(Serialize, Deserialize)]
struct Point {
    x: f64,
    y: f64,
}

fn main() {
    let point = Point { x: 1.0, y: 2.0 };
    let yaml = to_string(&point).unwrap();
}
```

(De)serialize

```
#[tokio::main]
async fn main() {
    let (tx, mut rx) = mpsc::channel(100);

    tokio::spawn(async move {
        for i in 0..10 {
            tx.send(i).await.unwrap();
        }
    });

    while let Some(i) = rx.recv().await {
        println!("got = {}", i);
    }
}
```

Async

Component-based WASM Apps

```
impl Component for Model {
    type Message = ();
    type Properties = ();

    fn update(&mut self, _ctx: &Context<Self>, _msg:
Self::Message) -> bool {
        false
    }

    fn view(&self, _ctx: &Context<Self>) -> Html {
        html! {
            <PeerList />
        }
    }
}
```

```rust
extern "C" {
    pub fn syscall(
        syscall: i64,
        futex_addr: *const AtomicU32,
        op: i32,
        val: u32,
        timeout: *const c_timespec,
        uaddr2: *const u32,
        val3: u32,
    ) -> c_long;
}
```

Zero Overhead FFI

```rust
#![no_std]

fn main() {
    // do stuff...
}
```

Run on Bare Metal

What you don’t need to do

```
fn main() {
    let mut v = Vec::new();
    for i in 0.1000 {
        v.push(i);
    }
    free(v);
}
```

Manage memory

```
let locked = shared.lock();
locked.mutate(foo);
locked.unlock();
```

Unlock mutexes

Speak compileese

```
error: unknown start of token: \u{37e}
 --> src/main.rs:2:22
  |
2 |     println!("hello");
  |                      ^
  |
help: Unicode character ';' (Greek Question Mark) looks like ';' (Semicolon), but it is not
  |
2 |     println!("hello");
  |                      ~

error: could not compile `playground` due to previous error
```

```rust
fn main() {
    let vga = 0xb8000 as *mut u8;
    unsafe { *vga = 'c' as u8};
    unsafe { *vga.add(1) = 0};
}
```

Listen to the borrow checker

A Deeper Look

Algebraic Data Types

```
struct Foo {
    x: u32,
    y: String,
}
```

```
enum Bar {
    This(Foo),
    That(bool),
    Other(String),
}
```

Error Handling with Types

```
enum Result<T, E> {

    Ok(T),

    Err(E),

}
```

```
match may_fail() {
    Ok(val) => val.use(),
    Err(e) => handle_err(e),
}
```

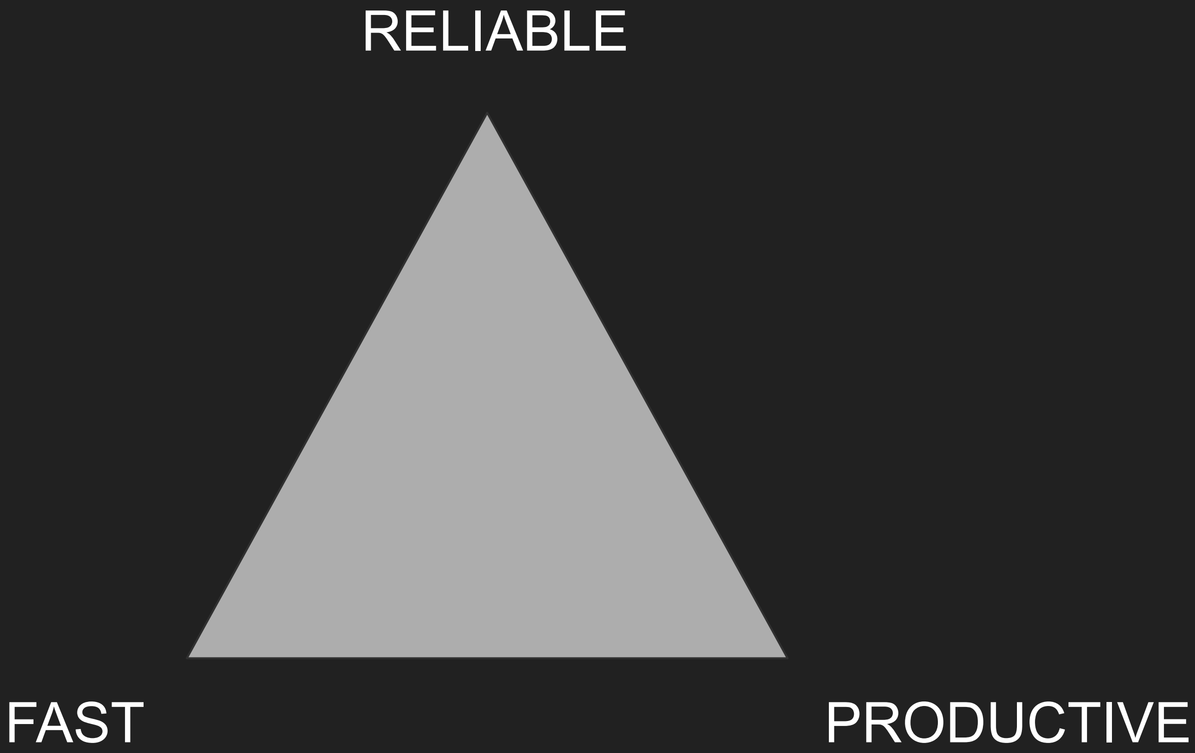
Generics and Traits

```rust
fn largest<T>(list: &[T]) -> &T
where
    T: PartialOrd
{
    let mut largest = &list[0];

    for item in list {
        if item > largest {
            largest = item;
        }
    }

    return largest;
}
```

Conclusion

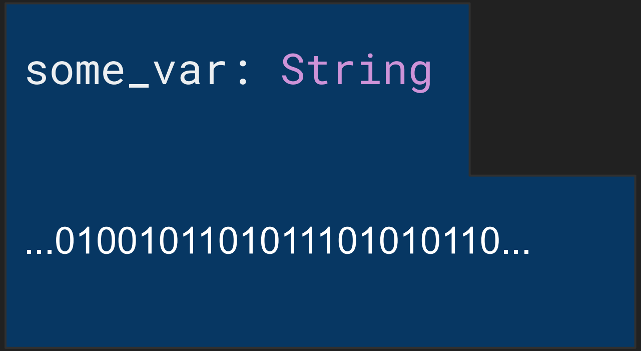

RELIABLE
FAST
PRODUCTIVE

Pick 3

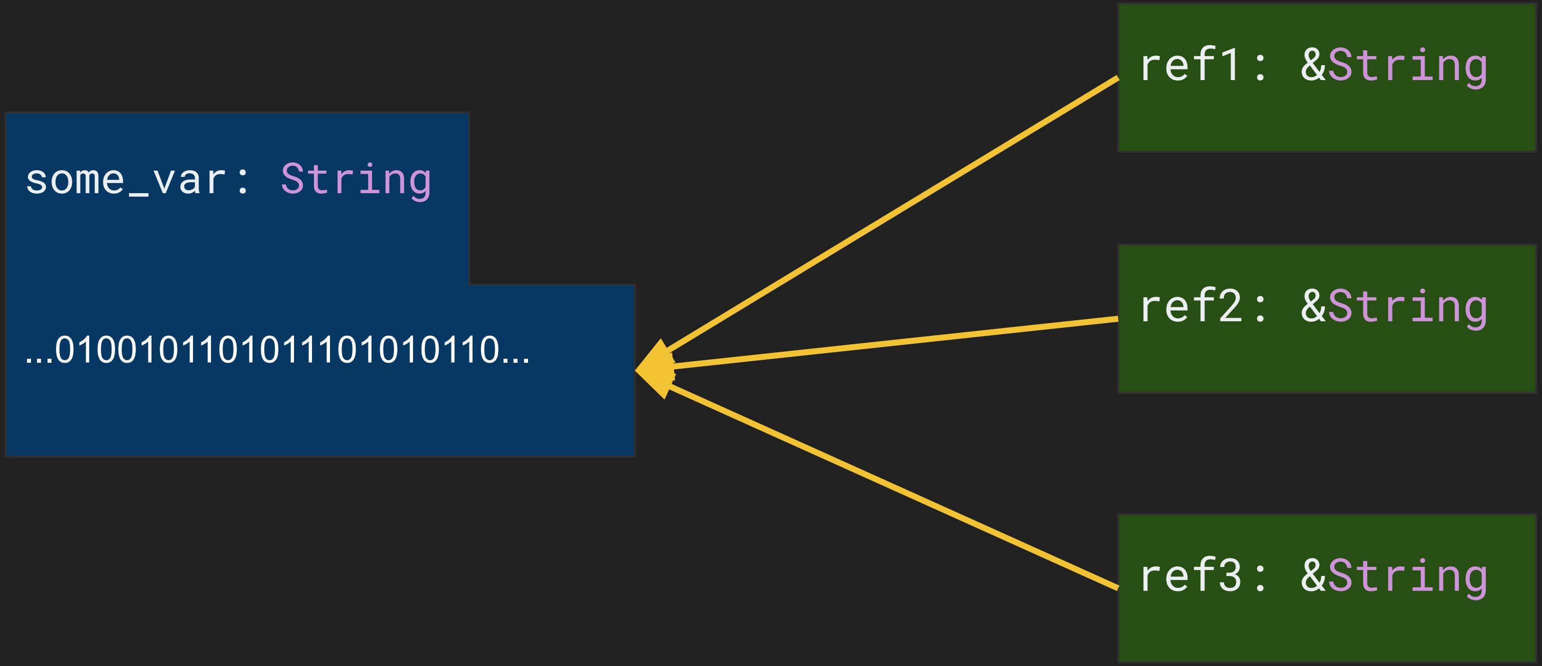
Thanks!

citations:
How rust views tradeoffs: https://www.youtube.com/watch?v=2ajos-0OWts
The Rust Book: https://doc.rust-lang.org/book/foreword.html

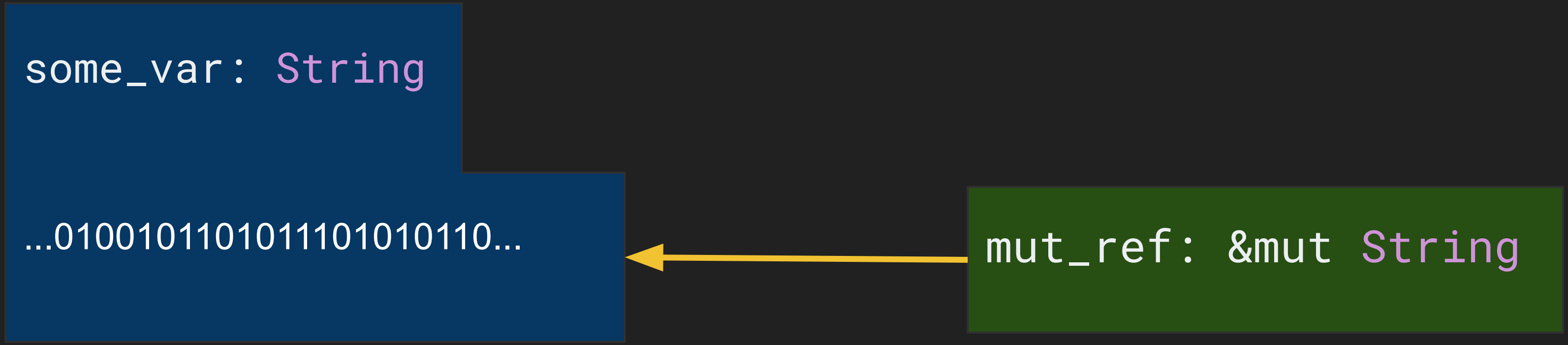
Ownership and Borrowing



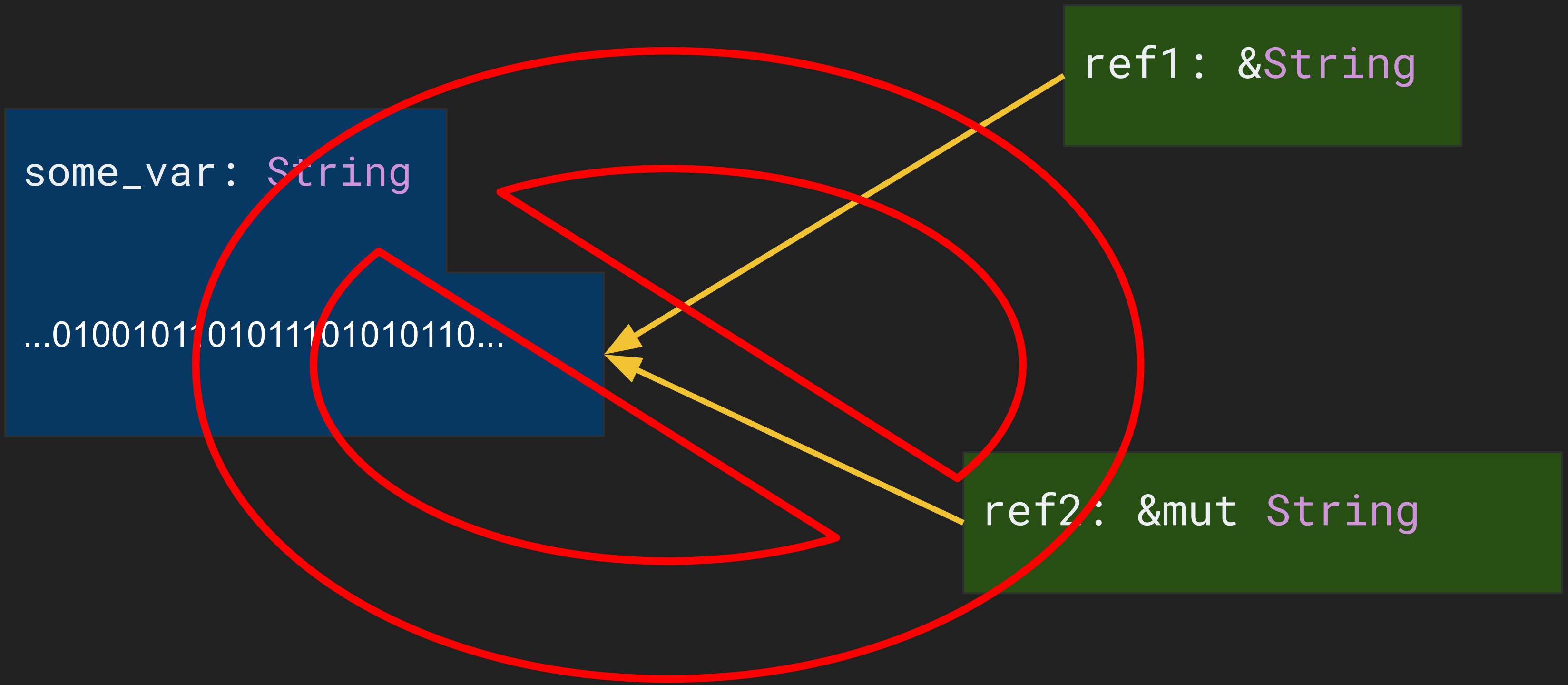
Ownership and Borrowing



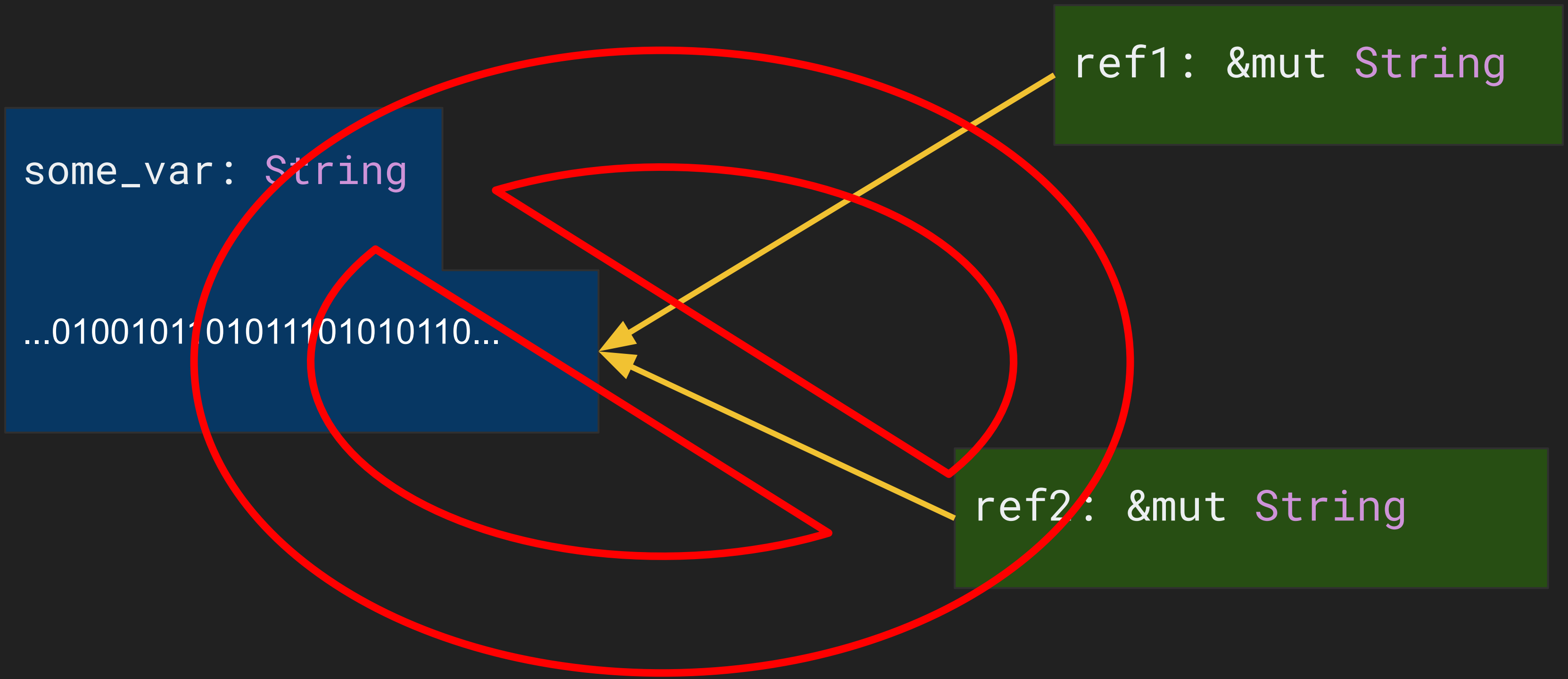
Ownership and Borrowing



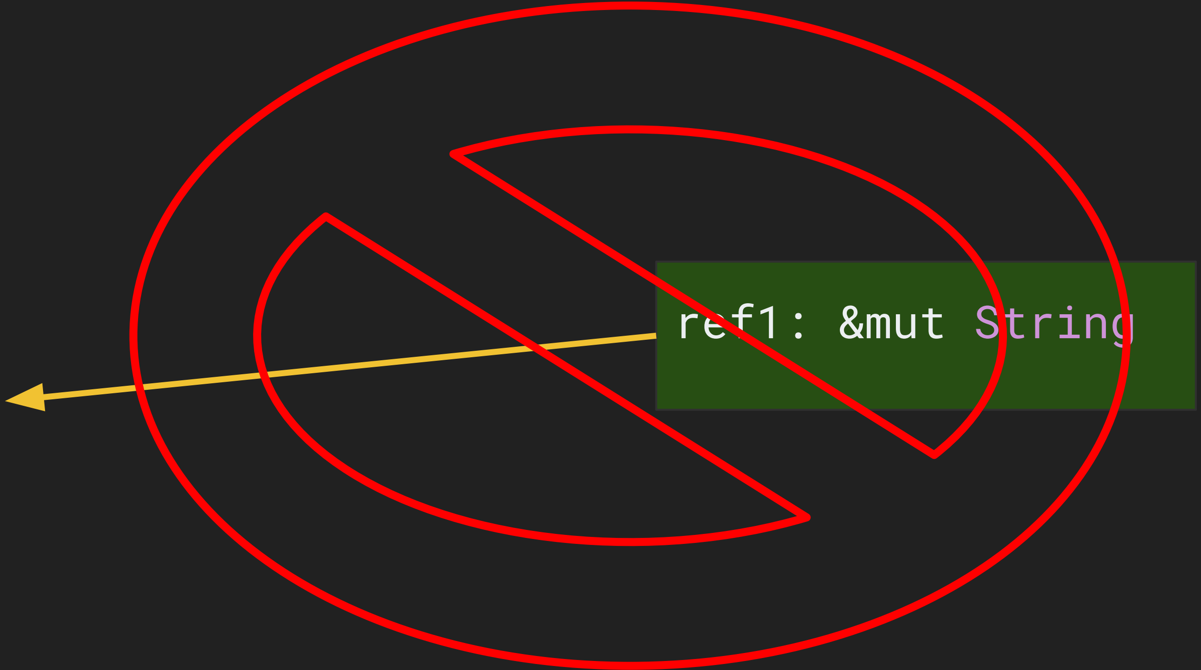
Ownership and Borrowing



Ownership and Borrowing



Ownership and Borrowing



```
ref1: &mut String
```

Memory management

```rust
fn foo() {
    let mut v1 = Vec::new();
    let mut v2 = Vec::new();
    add_many_elements(&mut v1);
    add_many_elements(&mut v2);
}
```

Memory management

```rust
fn foo() {
    let mut v1 = Vec::new();
    let mut v2 = Vec::new();
    add_many_elements(&mut v1);
    add_many_elements(&mut v2);
    drop(v2);
    drop(v1);
}
```

Memory management

`OMG drop is magic?!`

Memory management

```rust
pub fn drop<T>(_x: T) {}
```