1 **Interviewer**
So to start, what have been your motivations for learning Rust and choosing to use it?

2

3 **Participant**
A lot of the motivation was just a project I wanted to work on, which was the server project used Rust. But also I really like systems programming, I like programming where I'm really close to what the computer's doing. I like having that level of mechanical sympathy with the computer. When I started using Rust, I had touched C++ a bunch and I found that that was really nice, but also I didn't really feel comfortable doing very large things in it and Rust came along and it let me do a kind of programming that I was overall, did not have the time to get good at otherwise at the time. So it opened up kind of a new world for me and that kind of stayed there. So that's where I started and kind of why I continue.

2:41 B... | Low-Level Access

2:40 When I... | C++ is Difficult

4

5 **Interviewer**
You said that you did not feel comfortable doing large things in C and C++ compared to Rust. Could you describe a bit more what you mean by that and how Rust differs from C++ in that way?

6

7 **Participant**
It's just the memory safety stuff in Rust and that C and C++, if you're working on a large project, if you're working on your own large project, you do sometimes have, I find it possible to do this, but you have to keep track of a lot of things in your mind to make sure your program is safe and the larger it gets, the more you have to do that. There are a lot of best practices to avoid that, but at the time I didn't necessarily know all of them because I'm self-taught, which means that I learned them through experience and there are just a lot of these best practices that you have to know and I was learning them, but part of it is just if you're working on a large project on your own, you have to keep track of everything and what I noticed is if you're contributing to a large project, they all have their own kind of internal practices around this, which are not necessarily what everyone else does, and figuring that out and figuring out what's going on, you really just, it's almost like you're learning a new language. For C++, it's almost like you're learning a new language each time when you work on a large code base because a lot of the subtleties change and they change in ways that are not immediately obvious when you look at the code. You have to just know certain things about how this other part of the program is written and I think over time that's gotten a little bit better for C++, but a language where there are the, I don't really have much against C++ having a lot of dialects that get used in different large code bases, especially like each large code base has its own special dialect usually with its own internal library. That's fine, a lot of people do that, a lot of Rust projects that work on do that, but pair that with the fact that understanding what your code is doing is very very important for safety, that becomes much trickier. I just never felt that comfortable working on large C++ code bases. Over time now I am actually, but because I worked on [browser] as well for a while when I was working on

2:38 part of it is just if you'r... | C++ is Difficult

2:1 You have to just know ce... | C++ is Difficult

... | Web Browser Develop

the Rust integration and wrote quite a bit of C++ there, but overall when I started using Rust, I was definitely not.

⋮ Web Browser Develop

Interviewer
So in that particular integration with [browser], having a project where you had a mix of Rust and C++ interoperating, how did you feel about your confidence with the scale of the code base and memory safety in that environment where you had a mix of the languages versus one?

Participant
7 This was a couple years after I had started writing Rust and since then I had already gotten better at C++, so the same reasons don't apply. At that time I mean I was more comfortable, it would still take a lot more work in the C++ thing. The anecdote I have is when I was, there was this one big change I needed to make at one point that was across both Rust and C++ and I kind of predicted it would take me three or four days with like one day to do the Rust side, one day to do the C++ side, and one day to kind of make sure they're all aligned. It took me around two or three weeks and what happened was partly I had to figure out how to split it up into pieces so that the C++, so that if there was a memory safety error, I'd have a small amount of things to look at and partly it was actually chasing those down. What happened was like the Rust side was still easy, but the C++ side, every turn there were problems due to things I had not realized yet and some of this was because I was not an experienced person in that code base. I knew C++ but it wasn't like I'd been writing [browser] code for five years, so some of that was that and over time I've picked picked up on that and some of that was just, it would have been, anyone else would have taken that amount of time because some of this was memory safety, some of this was thread safety. What we were doing was definitely out of the bounds of what people expected that code to do and since there were no guardrails around it, you kind of just had to do the thing and then see what broke, which was not great because it's impossible to kind of understand for something so deeply integrated in the code base and understand all the effects it might have. We could do it to the extent that once it was working we had reasonable certainty that it was safe, but the transition was a huge pain.

2:39 It took... C++ is Difficult

2:3 What happened was like the Rust... C++ is Difficult

12

13 **Interviewer**
With the transition and with the problems that you were tackling in that code base with the C++ side, were they local to C++ or were they introduced purely by the fact that you had Rust and interoperation? Were there problems that arose specifically because you had to adjust your C++ code base to fit the memory management patterns that you had in Rust?

14

15 **Participant**
Not actually that much the latter. Mostly the former, but there's some nuance there because there's a bunch of stuff where it was just like, yeah, to do this refactor I need to change how we use this pointer in C++. Oh, this pointer can be null. That was not clear. Stuff like that. That's a C+

2:42 t... Different FFI Memory

+ problem itself where I'm making basically a pure C++ change. There's Rust involved, but the actual change I'm making is mostly a C++, can be expressed in C++ and then there's like some nuance there. Then there's the other one, which is thread safety stuff. This is like a tricky middle ground because what we were doing, I was working on the [name] project, we were just taking a parallel style engine written in Rust and bolting it onto [browser]. There are two things tricky about that. One is browsers are very monolithic. The [name] browser tried to be less monolithic, but they're still rather monolithic and there are no clean components. You can't just go, oh yeah, that's the style system. It's all integrated with everything else. There's no nice public API. They're all just meshed like that. The way you do this kind of integration is you find a roughly okay part to tear. Then you get this jagged tear and then you glue it together. Because of that, that makes the FFI stuff a bit complicated.

2:42 t... Different FFI Memory

2:4 The [name] br... Difficult FFI

Participant
But also, browsers are like both [browser] was written to be single threaded. It had multiple threads, but it was written to have a concept of main thread where a lot of the bulk of work is done. What we noticed was, of course, one of the things we were doing was working on other threads to parallelize it. However, because like I said, it's a very jagged tear. There's C++ calling it to Rust. Rust calling it to C++. The call stacks get kind of mixed up, which is fine. That's what we want. But it means that sometimes we call it to C++ from the wrong thread, which is not a problem in [browser] because it was very clear in [browser] when you were on non main threads. In Rust, we were throwing threads around all the time. That was the point. So there was a lot of things where we'd have to go back and fix that.

2:32 There's C++ c... Difficult FFI
Fearless Concurrency

Participant
Fortunately, there were asserts all over the place for this. So it wasn't like, we didn't have to break out thread sanitizer, but it was annoying. And what we realized through that is actually, it was a validation of a belief that a lot of people had was that this would have been impossible to do in pure C++ because you we were we had to hit this for the occasional times we had to call out to C++ from Rust and we'd hit this every now and then. But if we were only writing C++ code and we were just using all the internal C++ APIs, we would have been hitting this constantly. And it was the only reason it was even possible to parallelize was because we we could do most of it in Rust and then occasionally there'd be C++ stuff on other threads and we'd have to kind of sit down and specifically handle those cases, which was still a lot of work, but much less work than doing everything.

2:5 Fortunately, there... C++ is Difficult

2:44 And it was... Fearless Concurren

Interviewer
15 Yeah, that makes sense. So this next question gets at a bit of a higher level. What do you currently use unsafe Rust for? Like, what are the sort of use cases and the applications you're currently contributing for?

22

23 **Participant**

Yeah, so I'm doing a bunch of FFI, firstly, which is kind of one of the things I've always been using on safe Rust for talking to, in this case, previously I've done stuff where Rust is calling other code bases. Here, mostly what I'm doing is the main library I work on is something that we want to be callable from other languages. And we're using this tool that I've built that lets you build those APIs using kind of like a shared interface in between. So you say this is what I want my FFI, so I can get to all of these. And of course, there's unsafe code involved there. The other thing I do is…I've been doing zero copy deserialization. In Rust, basically, a very efficient way of doing deserialization that involves not allocating new space for anything you deserialize, but instead, having your deserializations all be stack copies or pointers to the original data, which we use a lot in [library], the internationalization library I work on, because we want internationalization data to be loaded efficiently. And implementing that in Rust is possible. I'm not the first one to implement it in Rust, but the strategies around that, we need some amount of unsafe to do that correctly.

2:8 Here, m… Directionality

2:4… Engaging with…st Com

Interviewer
Gotcha. So it's just the memory access that that level requires unsafe?

Participant
Yes.

Interviewer
Could you describe a bit about the FFI stuff? So you mentioned having this sort of shared API between multiple ...

Participant
So what I've noticed over the years is there are different tools for doing FFI in Rust. There is bindgen, which is about calling C++ or C from Rust. And you give it kind of a C++ or C header, and it gives you stuff so you can call from Rust. You can use it to go the other way around. It's clunky to do that, but [browser] has done that.

2:46 And… bindgen

Participant
Then there's cbindgen, which is about the other direction, which is calling Rust from C, and it looks at Rust code and generates a C kind of interface in C. And then there's stuff like CXX, which are bi-directional, where CXX is a tool where the point is kind of going both ways. It'll generate C++ headers, but also it can read C++ headers and let you call them from Rust code. It does both ways, but it's for one language

Participant
And the missing piece that I've noticed is there's not really a tool that, for the use case of I'm a library, I never want to call code that's not Rust, but I want everyone to talk to me. I don't want one language to talk to me. I want everyone to be able to use me as a library. And so there are now
23 two tools that do this. There's one by Mozilla called UniFFI, where you

2:47 I… Directionality

… Uniffi

kind of write these IDL files. And what it does is I think it generates traits on the Rust side and headers and stuff on the other, whatever other side, and you implement these traits or you plug into whatever stuff it gives you. And then from the other side, you can call it.

2:48 The…
Uniffi

Participant
And then there's Diplomat…which is you write these. It works similarly to CXX and where you write all of your public kind of FFI APIs go in these bridge modules. You write these bridge modules that have, you can have like types and methods and stuff. And what it does is that it generates additional Rust code using a proc macro to, so that all of the actual underlying C functions exist. And then every backend for different languages calls those same C functions. But when you write your backend, you kind of figure out these Rust APIs, how would they be idiomatic? How would they be idiomatic in this language? Like, oh, yes, you know, a struct in Rust is a class in JavaScript or C++. A result in Rust is an exception in JavaScript, but maybe it's still a result type in C++ because people don't like exceptions in C++. Like, you figure out where you're landing on all of this and you do this translation so that another library can call this Rust API in a roughly idiomatic looking way. And that's kind of what we're using. That's this is a tool generating stuff.

2:49 And then ther…
Diplomat

2:9 But when you writ…
Idiomatic FFI Encap

Interviewer
Gotcha. So to clarify the architecture just a bit, you have the central code base, which is Rust. And then that code base is exposed to any other arbitrary code base through a C API. But then you have this tool generating bindings like through the C API to the Rust library that are idiomatic based on how you'd interact with the Rust library in that language.

Participant
Yes. Yeah. And the tool does not read the C API. This tool also generates the C API. So like we are not the thing that we as humans write is not C API. The thing is we as humans write is something that looks like the Rust API just that it's somewhat restricted in what you can do. You can't do arbitrary things in it. It will like if you start doing generics, it will yell at you. If you start doing start using really complicated types, it will yell at you. But it can do most basic things. And as long as you stick to that subset of Rust, it will generate a C API for you and the tool.

2:50 And…
Diplomat
Generation VS Validati

2:33 You ca…
Limitation of Bindir
Simple FFI

Interviewer
So with encapsulating unsafe Rust or unsafe function calls that you're doing from Rust, there's sort of the practice of reasoning about them in terms of pre and post conditions that you need to establish for those calls.
35 Do you feel that translates for this situation where you have a Rust library that's being called from other languages? Or is it mostly that?

44

45 **Participant**
Gotcha. Sometimes. So and we've done it sometimes. But for I think

…
CXX

there has been controversy about this in the Rust community in the past, because for example, CXX, the tool CXX does not actually mark everything as unsafe. Because it trusts you when you write a C++ header, that that header is correct. And there's a perspective that you're calling into foreign code, that it can do anything. So it's unsafe. But the other perspective is that then you get unsafe absolutely everywhere in your code base, and it stops being useful. And furthermore, I think the point is, if you are linking to a different language, you have already put a cap in the level of safety you're gonna get. I spent like a different unsafe language, if you're linking to a different unsafe language, you've got a cap on the amount of safety you're gonna get. You already have a giant part of your code base that's going, trust me, I know how to do this. It is fine for the user interface, the interface between Rust and C++ to also be trust me, that like, like C++ to be able to go to Rust, trust me, I'm doing this right. Because you're already doing that all over within the C++ code base. You should not do the trust me, I'm right, if you're manually writing the bindings. But if you go to the tool reading the C++, gaining the and understanding what the C++ is trying to do from the headers, it's fine to do that. And that's kind of the position here is, you can put preconditions post conditions, we do this sometimes. Like, for example, we do we don't actually, right now, assume anyone is giving us valid UTF-8, we validate it if you give us a string. Most of our APIs take very small strings, so it doesn't matter. There's some APIs that take large ones, but they also support unvalidated, unvalidated UTF-8 directly and things like that. So even though on the Rust side, our APIs take UTF-8 strings, we validate them before everyone over FFI gives us basically bytes. That's one way we do preconditions and post conditions, we do those manually. But yeah, when it comes to FFI, you can do stuff like that, but there's a limit to how much you can do because you're ultimately just trusting the rest of that code base the same way people within that code base are trusting the rest of that code base. So that that is not the main strategy, I would say.

2:10 But for I think... CXX
Shifting Ground
2:51 It is fine... C++ is Difficult
2:13 And that's kind of th... Runtime Assertion
2:52 But yea... Difficult FFI

Interviewer
Gotcha. So would it be accurate to say that in terms of pre and post conditions, it's usually lower hanging fruit like format of data that's entering or leaving the interface and not larger memory safety problems because you sort of assume a certain cap or are there more complex things that you would in some cases want to verify?

Participant
45 A lot of larger memory problems, you just can't as well, like verify at runtime. So like, what are you going to do about a borrow error when if C++ is holding it wants to hold the thing past its lifetime, there's nothing you can do about that. You can't test that test for that at runtime, unless you are linked in the sanitizer, and then you've got performance issues. So yeah, so for unsafe code, unsafe library code, pre and post conditions work quite often quite quite well, sometimes, not for everything that they work often. But for unsafe, FFI code, they really don't because to do this kind of thing, you've got to maintain a bunch of state that you're not going to be maintaining on the other side.

2:14 So yeah... Difficult to Encapsulat

50

51 **Interviewer**
So you mentioned several different reference and memory container types, including Box and arc, especially, that you have either used to contain raw pointers or converted to and from raw pointers. So in this case, how have you reasoned about memory safety in those situations? And like, what's been the use cases surrounding having that container in Rust and then unwrapping it and exposing it to the foreign code or unsafe code that you've used?

52

53 **Participant**
For foreign code, it's basically you have to. If you have a Rust allocation, you want a foreign code to be able to read it. You really don't want to send Rust's structs over FFI by move. You really don't want to return a box over FFI because there are some ABI concerns around things like that. In particular, I think MSVC gets very annoyed about that. And the bindings change across different platforms and stuff like that. It basically is a mess. So you don't want to do that. So you want to just turn it into a raw pointer first and return it. That's one thing. So if you're doing FFI, a lot of these things you have to convert. The other thing is if you are implementing some low level unsafe thing and you're using Box or Vec or Rc as a kind of like a spring off point, you want them to do their thing. You want them to do the allocation for you or whatever. And then you want to do some of your own things on it. And you want to store it as your own thing. Then sometimes it's useful converting to a raw pointer. I think one of the entries there was UnsafeCell. If you want to use that if you're doing anything with custom interior mutability, which is rare because there are a bunch of existing good cell APIs, but I've been around long enough that not all of them existed at the time. So I've had to do this. And if you're using UnsafeCells, you have to use raw pointers. That is how that API works.

2:15 You really don... Simple FFI

2:16 The other t... Allocation in Rust

2:53 I think one of... UnsafeCell<T>

54

55 **Interviewer**
Yeah. So for cases in which you've used like Box or Rc and unwrap that and convert it into a raw pointer, have you had problems in this case with like memory leaks due to the semantics of those types? Or are there ways that you practice development when you're doing that to ensure that you are actually like re-wrapping that pointer and having it being deallocated properly?

56

57 **Participant**
Um, so we've mostly done that. Like, I mean, honestly, I don't think we've had that many problems around that. Or I've had, I don't really do this kind of thing. Well, okay, Diplomat does this. But in holistic code bases, I don't think I've actually had a problem there just because usually when you do that, you're handing it off to a type on the other side, FFI or whatever, where the rest of that logic is written. Like where, where, where you're turning it into like a unique pointer or something in C++. So the other half of that logic exists somewhere. It might if you might forget to hook it up. That has happened. I recent last year, I think the last time I

2:54 I don't think... It's not a problem

2:17 S... Memory Leak Memory Leaks

had this problem was last year, I refactored some of the zero copy deserialization code to be slightly different. And I forgot to write a destructor. So it ended up not. It ended up like having a memory leak. But it was it was kind of like, that was also just easily caught and easily found. We don't have a sanitizer here for actually, no, we do have a way of detecting this. But yeah, I, for [browser] and stuff like that, we have used tools like Valgrind and whatever to do this. In general, I haven't found this to be that big an issue. Leaks are possible there. You can get them by accident. But I think it's not as likely to have a leak.

2:17 So t…
Memory Leak
Memory Leaks

2:18 But…
Memory Leaks Aren't a

Interviewer
Gotcha. Gotcha. Okay. So you've mentioned using with specific specifically with the foreign bindings. You were speaking a bit about Diplomat. We've already discussed some of CXX and other CXX as well as the different different versions of bindgen for C and for WASM. Could you speak a bit more about the usability of these tools or some of the problems that you've had in using them to create bindings and sort of evaluate which approaches you feel are the most effective?

Participant
Yeah. So I mean…I've been writing FFI for ages and have kind of a lot of opinions on this. So I think I mean, I think the thing is, it's hard to some of these tools are unidirectional, some of these tools are unidirectional, the other directional and other direction and some of these tools are bidirectional. So that makes it hard to compare. One thing I've noticed is like one opinion I hold is for it's actually if you're if your FFI tool is reading Rust code, it's actually not great for it to read all of the Rust code you have, you should give it a small subset of your Rust code to read and tell it to stick to that. Because what you don't want is you make some internal Rust change and then your FFI suddenly changes. You want to notice that. And also what you don't want is your FFI tool to have to deal with all the complexities of Rust because Rust is a complicated language. I've used cbindgen in the past, it is great for what [organization] wants to use it for. But it, it has a mode for where you can kind of have it just read everything. Basically, cbindgen looks at the extern C functions in your library, and it just generates everything based on that. If you use a type in that, it'll go find that type and generate something for that type. And there's a mode when cbindgen where it can do that and also chase down crates, if you can tell it, look for look for this type and other crates and stuff, it's not from here. And that when it works, it works. But when it doesn't work, it's a mess. Because cbindgen does not understand all of
57 Rust, and it can't generate FFI for all of Rust anyway. So my general opinion is that if you're building a tool where your source of truth is Rust code, make the source of truth be very explicitly tagged Rust code, like a separate crate, or separate modules, and both CXX and diplomat do this. And that's part of the design. You need FFI instead of the source of truth being Rust code, the source of truth is an IDL file, so it's not the same problem. And bindgen works in the other direction. So it doesn't really have that problem. In bindgen's case, the source of truth is instead the entire C++ code base. But worth noting about bindgen is, is very configurable. And the way everyone kind of uses it is, you point it at a C+

2:59 So I…
Directionality

2:55 if you're if…
Generate Bind...From a

2:56 I've used cbindgen in the…
cbindgen

2:57 So…
CXX
Diplomat
Generate Bind...Fro

2:58 I…
bindgen

+ code base, and then you manually kind of tell it what headers are important, and which headers are not, because what you don't want is for it to generate FFI for everything you wanted to generate FFI for the things you care about, because that's faster, and also less, less chances of it trying to generate FFI for some weird messed up template thing that you don't care about in the first place. So I think that's one thing is that if you FFI tools should not have to read the entire code base on the other side, they should read a subset that you clearly define.

2:58 In bindgen... bindgen

2:19 S... Generate Bind...Fro

Interviewer
Gotcha. That makes sense.

Participant
And I think being very clear about unidirectionalness versus bidirectionalness is important.

Interviewer
Gotcha. That makes sense. Just a moment, I need to pull up your
61 responses to the survey to inform my next question. So you mentioned earlier, there's an issue with moves in particular, in general, that MSVC was not, was not a, like it wasn't, it didn't handle cases where you move a Box type across a foreign boundary correctly. Could you describe, I guess, that particular issue, as well as any other similar issues with types of foreign boundaries that tend to happen?

68

69 **Participant**
So, okay. So there are two things here. There's reprs or layouts, and there's ABI. So a repr / layout, they mean the same thing to me. In Rust, basically, we use repr to talk about layouts, our EPR representation. But the repr of a type is important to FFI in the sense that it is how that type is laid out in memory. If you're using this type from two languages, you should make sure both languages understand it's laid out the same way in memory. This is not that hard to achieve in Rust. You slap on repr C on that thing, and I think in C++ as well, you have some control over repr. So you can, you do that, and then it's the same. This is great as long as the type is being a pointer, right? When the type is being a pointer, all you care about is the memory representation being the same. But there are other ways of accessing a type across FFI. And one way is a function might return a type or accept a type as a parameter. Again, if it's a pointer, it's fine. If it's primitives, they're fine. The primitives work the same. And if they don't, you should use the primitives that do. But the moment you start doing something that's not a primitive, then, so structs and enums and stuff like that do have additional impacts on this. And what the impact is, is that whether or not the type may be represented the same way, but when a function, when a function ABI is defined when they're like calling convention and stuff happens, whether or not a type from the parameters or return values goes in a register or on the stack or be had a pointer or whatever, those different things differ based on the kind of thing it is. So if you've got primitives on either side, they'll work the same. But if you've got a struct on one side and a struct on the other side, it's not just that it

2:34... Shared Experiences

2:61 But the moment you sta... Difficult FFI had, it's a struct, like it can be a struct, it can have the same layout. But if one of them has a destructor, and one doesn't, they may not behave the same. And it depends on the platform. You can cause problems on this lunging and you platforms, but you can also like you can MSVC, like MSVC is definitely the one where I've seen more of these problems. And it is well within the scope of what C++ and Rust are allowed to do to differ on this. And to some extent, like C++ and Rust don't have the same constant of destructors in the first place. So when you say a type with a destructor works this way, you can't necessarily mean the same thing on the Rust side in the first place. But also it is very much like C++ has choices on how to do this and Rust has choices on how to do this. And they haven't necessarily picked the same thing. There's no standard on this. There can't be a standard on this because different languages kind of have different notions of destructors and stuff like that. So yeah, so this is a bit of a mess. And the end result of this is if you're doing FFI, the only things you should pass by value are copy types. Or the only things you should pass by value are primitives. And you can pass structs by value sometimes. They do this in Diplomat, but they only do structs without destructors.

2:61 B... Difficult FFI

2:36 And it is well within the sc... Different FFI Memory

2:6... Diplomat

Interviewer
Gotcha. Yeah. So I guess following from that related to structs, but related to just any situation with foreign bindings, could you describe some of the challenges you've had navigating the difference between Rust's model of memory and the C and C++ model of memory?

69 **Participant**
Yeah, a memory. Honestly, one thing that I remember, the C and C++ model of memory is not that different. It's just that you have to keep track of it in your head. And I remember when we were doing the [browser] stuff, we had a bunch of experienced C++ programmers start to learn Rust when we were doing this. And I was helping them do that and kind of also very interested to know where their challenges were. And the thing that I remembered was they did not have problems with what was widely known as the most complicated part of Rust. They were like, yeah, the borrow, that's fine. And their reasoning was this is how we think anyway. We're working on a large C++ code base. This is what we're doing in our head. So now it's doing it for us. Great. There were definitely concepts Rust had that C++ didn't have that they took time to pick up. And there are things like RefCell, for example, is not a concept C++ has. And it took a while for people to wrap their heads around that. But what people consider to be the complicated parts of Rust were not there. And overall, the memory model, not the memory model, the way people reason about memory is very similar in the two languages. Except Rust does it for you by default. One thing that C++, for example, has that Rust does not is it has a far finer distinction of different kinds of value types, value kinds, I forgot what they're called, like l values, x values, they're basically C++ can has a way of talking about temporaries at a type level, which is how C++ implements moves. C++ has a bunch of other things there. That gets complicated sometimes. That's Rust in Rust, you kind of just have, you do have a distinction between l values and r values are what we call places

2:21 Honestly, one thing that I r... Becoming the Borrow

2:2... Becoming the Borrow

2:24 One thing tha... Feature Disparity

and temporaries. But that distinction is basically internal. And you need to care about it in a couple places when it comes to borrowing. But the fixes are straightforward. So it's not it's not as in your face as it is in C++, where I've often had to think about these things. So that's like one difference. But overall, like the memory stuff is still similar, the way you deal with memory in Rust is very much inspired by the way people manually do it in C++.

2:24 One... Feature Disparity

2:23... Becoming the Borrow

Interviewer
Gotcha. You mentioned specifically there are a couple cases where you have to care about it in terms of borrowing. Could you describe that just a bit more?

Participant
In what context? Did I mention that?

Interviewer
So in terms of the difference between, like if you're borrowing memory that is, I believe, coming from C or C++. But there are certain things you have to consider. And you mentioned that there are fixes for that that can help you.

Participant
If you're borrowing memory from C++, yeah, you just consider things like, it might be null in C++. You have to consider things like C++ might have
73 some internal lifetime relationship that is not reflected in your API. And you can figure that out by reading the C++ and understanding or documenting it and then going to the Rust side and putting the right lifetimes there. Like, if you're doing complicated FFI things you are going to care about, does this does who borrows from who? Is this its own allocation? Is this a borrow? If it's a borrow, where is it from? And again, this is also how people are thinking in terms of C++, they may not they may not be using the same terminology of borrows, but they are still thinking about this pointer comes from here and needs to needs to live this long.

2:25 If you're b... Different FFI Memory

2:37 Like, if you're... Becoming the Borr

82

83 **Interviewer**
Gotcha. So the final questions that I have are related to the bug finding tools that you listed in the survey. So you mentioned using both Miri and Clippy as well as a few of the sanitizers. Could you describe your experience using these tools and sort of compare and contrast them as well as the types of problems that you've found in in code bases using Rust or Rust and mixes of other languages?

84

85 **Participant**
Yeah. Yeah, so Miri is great for looking for testing on safe code. I actually haven't had the time to write Miri tests for the zero copy deserialization things I've written. I mean, we have tests and we can run them under Miri and they pass but we haven't put them in CI yet because I want to write

2:65 Yea... Running tests through

some more tests and stuff and I just haven't gotten around to that. But Miri is pretty great because one of the things about unsafe Rust is if you're doing FFI, you're mostly fine. There are some caveats, but if you're doing low level abstractions, a lot of what is and isn't undefined behavior in Rust hasn't been pinned down yet. And that's, I think, one of unsafe Rust's biggest problems so far is that they're still working on that, which means when you write unsafe Rust, there is kind of this mix of community best practice. Knowing what it's very unlikely that the unsafe code guidelines will break, and also just knowing where their heads at right now, knowing what's coming soon, you have to kind of be in that space, which makes it a much harder proposition. We are moving, one thing I'm moving towards is definitely I'm working on with a friend of mine working on this little mini book to teach people about this. And it will contain a bunch of the things the unsafe code guidelines group is up in the air about, not saying that do it this way, but saying it's in the air about it. Also, here's probably the best way you can avoid having problems with this. And we're all understanding that if they do, if they end up breaking it, they will make sure if they end up changing something that affects people, then they will make sure that they do it in a way that doesn't break too much and gives people time. But yeah, Miri is useful because it helps catch some of these things as they add them, it catches them. It's also sometimes buggy. It's not perfect, but it won't catch everything. But it catches quite a few things. And it's quite good for that. And I think part of the thing is a very nice thing they've done is they have tried to define their unsafe model in terms of Miri, or in terms of something Miri can do. So even though I mentioned that there are some things it doesn't catch, these are mostly things that they have, they have decided on, but has not been implemented or have been fully implemented. But in theory, it should be able to catch. And in the long run, I think Miri should be able to catch every instance of undefined behavior in not in not every instance of broken unsafety in your code, but every instance of undefined behavior that actually happens in a fully compiled binary on an execution path, because it's still it's a runtime test, it only does execution paths, it doesn't look at your code and go, oh, that's phishing. So yeah, Miri is useful for that. Clippy is just a general useful for catching things, making sure API is clean. I've heard it's like, I've gotten great feedback that it's very useful for people learning the language. It did not exist when I was learning the language. So I can't, I can't say but I mean, yeah, it is. It's pretty good for that. But it's also as someone who uses it as a mature tool in a code base worked on by Rust, mostly people who already know Rust. We, yeah, it works pretty nicely to catch some small things. It's not too annoying. And it's pretty good. I like it. It catches bugs. It doesn't catch that many bugs overall. Like it's, well, I mean, actually, no, it does catch quite a few bugs. But what it the vast majority of stuff it catches is just more style and API niceness things, which are I guess, API niceness stuff is I would consider that a bug in some sense, like if you're not following community norms on API is maybe that's not great. But yeah, it does catch bugs. Like I have, and it's pretty good at that. It's also just the more experienced you get in Rust, a lot of the bugs that Clippy can catch, you also notice so you end up just having it help with your API for people new people new to Rust working on our code bases, I've definitely noticed that like Clippy is very helpful for them, because like, sometimes they come to me, they're 85

... Running tests through

2:26 There are some cav... Shifting Ground

2:27 And I think pa... Shifting Ground

2:28 Clip... Clippy for Learning Ru

2:29 It's als... Clippy for Learning Ru

they're like, hey, so like, I saw this error earlier. And like, I did what it told me to, but can you explain more or whatever? So like, it's they do experience that a bunch. Kind of like, yeah.

Interviewer
Gotcha. And would you say, I guess, do the development tools you use, you feel that they handle all of the problems that you face when you're writing in safe rust? Or are there particular issues that you feel are not handled by this combination that you describe?

Participant
Clippy is not really trying to help with unsafe Rust. Like it is, it's not like it doesn't help with unsafe for us. But it, it doesn't have that many lints for unsafe Rust, because unsafe Rust is part of the problem of writing a linter…a lot of the things about making a linter is understanding user intent. And the real tricky thing is programming languages are already the tool that computers use to understand user intent. So when you're writing a linter, you are looking at, you're looking for user intent in a situation where they're already using a clear way of expressing intent, but it may have made mistakes. So you need to like find out their actual intent. This is often possible. Often there are just patterns where because we know how we know kind of how people think when it comes to code, we know what kinds of mistakes they make, so we can look at the mistake and be like, Oh, yeah, that's a common mistake. When you make this kind of mistake, your intent was almost certainly to do this, that you can do. The problem with unsafe is when you're when you're statically checking unsafe, you have already been you've already stepped out of the bounds of rust, or normal rust, you're already in a situation where you're doing something weird. It already becomes harder to glean intent from code from broken code. If the code is correct, that is its intent. If the code is not like, if the point of Clippy is to catch brokenness, so it can't catch that. But it can definitely catch things like, oh, if you do this, instead of this, this is going to be more resilient, like there, there are some patterns in unsafe code. So it catches a bunch of things, but it can't be, it can't be the main tool for unsafe rust for this reason. So like, yeah, clip, Miri is pretty good. Overall, I think it's going to need to just be like, I think the good tool here is lots of tests in Miri. Like run all your tests into Miri, and that should be fine. But yeah. And more, more static analysis and Clippy, but ultimately, I don't think it can be perfect. And really, the actual thing that's missing here is more human analysis than static analysis, which is, there isn't really a good way to like a clearly agreed upon way to review unsafe code. And that's one of the things we're building with the book. Like one of the rules that we're hoping to settle on is not just that your unsafe code should be documented, but your unsafe code should be documented so that people reviewing the unsafe code do not need to hold your entire code base in their head to review it, they should be able to go block by
85 block, and maybe like, read the comments at the top of an individual file, but not have to jump around the code base to understand because they can trust, they can trust your comments. Like as to why things are unsafe, the comments say enough about the rest of the effects to know this. And then, then they have to just verify that the comments are correct, which is

2:2… Clippy for Learning Ru
2:66 The proble… Unsafe is Diffi…to Unde
2:30 And… Shifting Ground
2:31 Like one of the rules th… Audience-Dep…ocu

a much easier process. Like the comments are saying that, oh, yeah, that struck there has that feel, then you can go and check that. But yeah, so that that is the norm. I'm hoping to build it is a high bar. It is not a bar I have myself followed that consistently in my code bases, because I haven't had time. But over time, I'm kind of hoping to build that kind of bar. And I'm really excited for more work in the review space.

Audience-Dep...ocume

Interviewer
89 Yeah. And I guess so, in particular, with some of the dynamic tools you mentioned, like Miri and the sanitizers, do you feel that the bug finding capability of those is adequate how they are, I guess, assuming that it will be extended by these unsafe code guidelines that you're working on? Or are there existing issues that have yet to be solved with unsafe code that you think that these tools should be extended to address?

92

93 **Participant**
I think like, basically, a lot of progress here is just blocked on the unsafe code guidelines finishing. And typically, the unsafe code guidelines are the official rust rules around what is and is not defined behavior. Whereas what I'm working on is not that I talked to those people all the time, what I'm working on is kind of like a book that is not a set of rules, but rather teaching you unsafe. And teaching you like how to best practices for read and good unsafe.

2:6... Shifting Ground