Interviewer
So my first question is, what have been your motivations for learning Rust and choosing to use it?

Participant
When I first started learning Rust, I gotta say I wasn't really motivated by anything else than curiosity. In 2019, I started to learn Rust and went through the Rust book and I just enjoyed it. It met some of the things that I didn't like about C++. It satisfied. And I just kept using it for my hobby projects and I've yet to use it in my professional life.

Interviewer
Gotcha. So what type of hobby projects are you typically working on?

Participant
Well, it started with small command line applications just to automate some tasks that I like. Most recently, I did a chess engine and chess website in Rust.

17:... Personal Tools & Toy P

Interviewer
Nice. Gotcha. So then what do you use unsafe Rust for?

Participant
So my use of unsafe Rust in chess engine was mainly... So in a chess engine, you need to track board state. Part of that is the placement of pieces, what pieces are and stuff like that. For the placement, you track the rank and file, the pieces, you know, like having a number of all the pieces. I found I was able to get more performance by using enums for the rank and file because it limited to the zero to seven index of rank and file. And also for a combined rank and file is like a square. Right. And that's zero to sixty four. So as opposed to I guess more I was using unsafe to constrain my types to enums. And then the unsafe bit here is that the casting to enums knowing the discriminant of it.

17:3 I found I w... Contract or Invariant
Increase Performance

Interviewer
Gotcha.

Participant
OK, so I know the discriminant of my square enum, which is like sixty four variants is zero to sixty three. And I know the discriminant of my rank and file are zero to seven each. So I shift one of them over three bits and add them together and use a transmute to get my square enum.

17:39 OK... Transmute

Interviewer
Gotcha. OK, interesting.

Participant

There's a couple of little benefits, right? As opposed to... There's no more bounds checks on fixed size arrays of squares because the compiler is smart enough to figure out this enum has... Its discriminants are zero to sixty three, so no need for a bounds check.

17:4 Ther... Compiler Elidi...ounds C

Interviewer
Yeah, nice.

Participant
I think that was my most recent use of unsafe. I've also written little boot loaders for like x86 computers. And that's just lots of unsafe there interacting with like IO ports, raw pointers in memory and stuff like that.

17:5 I'... Operating & E...dded S

Interviewer
Gotcha. So with the enum transmutation that you were doing, is that goal mostly to increase performance by eliminating bounds checks or also to space efficiency, I guess?

Participant
Space efficiency was not a concern. Enums are still stored on disk with the same size that they would be. I still need a single byte to store them. It was mainly just reducing the bounds checks. And I find that's probably a better solution to get rid of the bounds checks than just to unsafely get rid of the bounds checks at the point where you're indexing. If you can prove that this type can only be zero to sixty three, then the compiler removed the bounds checks for me.

17:6 And I fi... Compiler Elidi...ounds C

Interviewer
Yeah, Yeah.

Participant
I mean, ergonomics also was a factor.

Easier or More Ergono

Interviewer
Yeah, I was going to say the bit shifting you used to hook and like that combination of the bit shift and the transmutation sounds like it was like a really elegant thing to use in that particular situation.

Participant
I mean, in other chess engines like Stockfish, one of the big chess engines written in C++, that's precisely what they do. They have rank and files where it's simply within just shift them together into a single square index zero sixty three. But they don't do bounds checks anyways in C++.

Interviewer
Yeah. Gotcha. Were there any other areas of unsafe within that particular chess engine or is that the main one?

38

39 **Participant**
I think that's the main one. The same thing for pieces, pieces made of color and a kind combine them into a single piece type. You could store them in a struct with like two fields, piece and color. But this one's more for space efficiency. Why store them in two bytes when you can store them in one?

17:40 The sa...
Increase Performance

40

41 **Interviewer**
Gotcha. Yeah. So then for the x86 stuff you mentioned with like boot loaders, was it?

42

43 **Participant**
Yeah.

44

45 **Interviewer**
Gotcha. Can you describe a bit more about those applications?

46

47 **Participant**
So, I mean, when your computer starts, there's like basically two modes of booting. But the simpler one, the older one, the BIOS boot reads a disk and then finds this 512 byte chunk of memory in a certain space on the disk with a little signature on it and begins executing that code in 16 bit mode. And then the job of that little bit of code is to bring the CPU into 64 bit mode eventually and load the rest of the operating system and let the operating system take over. So what I was doing here is working with like a nightly compiler with like a custom tool chain to get 16 bit code to be generated. And then using a small amount of global assembly to set up like an initial stack and then jump into some Rust code that communicates with the BIOS disk drivers to read the next stage and then jump to that. And all along here there's little bits of unsafety.

17:...
Inline Assembly

48

49 **Interviewer**
Yeah. Okay.

50

51 **Participant**
The unsafe there is this little bit of assembly at the start. Any inline assembly for it to call the BIOS disk functions use interrupts and those are inherently kind of unsafe and to do so you use some inline assembly.

17:9 T...
Inline Assembly
No Other Choice

52

53 **Interviewer**
Gotcha

54

55 **Participant**
And then make some safe abstractions around that. But then it's still unsafe to just write to some random memory locator you have no operating system to tell you to give you memory protections at this stage

17:11...
Interacting with hardv
No Other Choice

right. So, you're just writing to the raw RAM.

Interacting with hardw
No Other Choice

Interviewer
I can see how Rust would be particularly appealing in that situation sense without OS level protections on memory use.

Participant
Having the restrictions of the type system to back you seems appealing. I'm definitely not the first person to think that neither of you. I mean, Redux is an operating system written in Rust for those very reasons, right?

Interviewer
Yeah. So, you mentioned, creating safe abstractions around some of these lower level inline assembly instructions, could you describe what those abstractions would look like and how you would interact with them in that application.

Participant
Yeah, so I mean you can think of the, you can think of these BIOS interactions, these interrupt as function calls in a way. It's simply an interrupt, and then somewhere in memory the BIOS has an interrupt handler configured to run some BIOS code and then return back to you where you were. But the I mean the unsafe part really is having all of the parameters set up. So, like, and especially with BIOS stuff you need to have registers, be very specific values. One of those values will actually be decides what operation the BIOS will perform for you and then like other ones will be like parameters for the operation and then the BIOS, when returns, when it returns control to you will have set a bunch of those things to different stuff for like return values of status and that kind of stuff. So, definitely needs a human eye to look at all of those, make sure they're the same you can't really express all these inputs and outputs in the type system. Luckily, global assembly, so the global assembly you can actually assign these registers values, mark them as an input or an output and continue on that way and then these safe abstractions around them would simply just be a function that it takes those parameters the normal way and returns the results normally.

17:13 So, like, and...
Contract or Invariant

17:12 Luckily...
Inline Assembly
Interacting with hardw

Interviewer
Yeah, gotcha. So, with the situation you're describing, is that where you'd use something like MaybeUninit, where you're like using return pointers that are passed in and then populated by the BIOS?

Participant
Yeah, one case especially that uses that is getting the memory layer computer. So, one of the BIOS interrupt calls will like what's called the E820 memory layout and it basically you don't pass it a pointer, you receive a pointer from it and make sure you don't overwrite all that memory that it gave you. It's supposed to be done very early on, it's in

17:14 Yeah,...
MaybeUninit<T>

some special BIOS area of memory. Yeah, certainly, like reading from the disk, you pass it a pointer to a structure that then that structure has whatever so, like, and even modifies that structure and then you're expected to read through like the result written there. So, like, with that you create some structure in Rust that has all the right stuff, pass a mutable pointer to this function, then this function invokes the interrupt passing that mutable reference as a pointer and then, yeah. You have to kind of mark these things at the inline assembly call, you have to mark them that they modify memory in ways that Rust doesn't know about.

17:43 Ye... Return Pointers

Interviewer
I guess what do you mean by mark them? Do you mean like document them for?

Participant
It's more a marker so that LLVM knows not to optimize certain things across the call. See, LLVM is the back end for Rust, it ends up compiling stuff right and Rust will basically just pass off a bunch of information to LLVM and let the compilation happen but one of the things that LLVM will do when performing optimization passes is pre-fetch reads or completely negate the need to have a read of memory because it already knows what the value is or thinks it does. But those assumptions could be broken by the BIOS when performing this call. So it's very careful to say, like, I need a stack to belong here, this thing could touch memory that you don't know about and those kind of markers are present in Rust inline assembly.

17:15 one of the thing... Contract or Invariant
Inline Assembly

Interviewer
Gotcha. Okay, so is there a particular case where you, like, how, what's the process that you go about reasoning where to insert these markers? Is it just based on the semantics of ...

Participant
They're on there. When, when to remove them, but the basic thing is, if I'll put them on all, because I can't make any, especially with a BIOS there's so many different implementations out there that you don't know what you're going to do. You should essentially remove all assumptions. And, yeah, and that doesn't make it very hard to optimize the code but yeah. There's some great resources out there for the documents, a bunch of different behaviors for BIOS calls. I was using those during the time.

17:44 An... Engaging with...st Com

Interviewer
Did you find that those resources were accurate to the behavior that you observed or were there cases where you really did have to, like, protect yourself from divergent behavior?

67

78 **Participant**
I didn't, no, I didn't find with my personal computer. Right, I was just running in a virtual machine right, I didn't get any tremendously odd behavior but, yeah.

79

80 **Interviewer**
So, with the markers that you have with the inline assembly, I guess, another interesting aspect I'm curious about would be how Rust's memory restrictions like at a broader scope interact here. Like, the problems that you get when you're taking something that's a borrow and then you're converting it into a raw pointer and using that raw pointer in a way that respects the permissions that were associated with the borrow that it came from. Have you run into any problems like that with use of memory between the BIOS and memory that is in play within this Rust implementation? Or is this usually something that is just handled by ensuring that you have the right markers on your assembly instructions?

81

82 **Participant**
Yeah, I mean, you can, if you can tell the compiler that it can't make certain assumptions then you're all good, right, but certainly you wouldn't want to violate the mutable aliasing properties, you can't, if, like, straight up when you have two mutable pointers that's undefined behavior, right, so even if some BIOS function lets you get around that that would be bad. I certainly wouldn't, I didn't run into any cases where I had to try and skirt around Rust borrowship rules in that case. In, like, for some, like, writing to printed characters on the screen in its simplest form. There's old VGA drivers that have just the VGA text buffer in which you write ASCII characters and color attributes to a defined location of memory. And then that displays on the screen. This is like in the days of CRT monitors, so no, no monitor is going to be configured to do that right now, but it works just great in the virtual machine. But that's like just some location of all the memory, you don't know if the VGA driver is also going to write to that. So taking some mutable pointer to that is, is a no no.

17:16 certainly... | It's not a problem

83

84 **Interviewer**
Gotcha

85

86 **Participant**
But the way to get around that, though, is to not take mutable pointer to perform volatile writes using using a raw pointer. So Rust has just SCDPTR write volatile, right?

17:17... | Volatile Intrinsics

87

88 **Interviewer**
Yeah, gotcha.

89

90 **Participant**
Like, there are ways to, like, if something would have another way of writing to memory that you also want to write to, you should not have a mutable reference to it. And there's ways around that, just using pointers.

91

92 **Interviewer**
Gotcha. So in this case, it's like the places where these dangerous

patterns can occur are well documented because just of the nature of what you're actually accessing and who has access to it. So you're able to use Rust's existing methods like volatile reads with pointer or just the markers that you have around assembly to protect yourself from that.

Participant
Yeah.

Interviewer
Gotcha. OK. And then, so one thing I'm curious about, too, is you mentioned also using a Box in unsafe contexts. Did that show up within the bootloader stuff?

Participant
A Box in unsafe context.

Interviewer
Oh, I'm looking at your survey responses, so.

Participant
Well. OK, so I've used Rust across FFI boundaries before.

Interviewer
Gotcha

Participant
So, you know, writing C wrappers for some of my libraries or consuming C libraries with Rust. And in that case, just Boxing and leaking a pointer and passing the pointer to Rust. And then the unsafe part then is when you receive that pointer back to drop it, you need to convert it to a Box, right? Unsafely.

17:18 So, yo...
Allocation in Rust
Box<T>

Interviewer
Gotcha

Participant
So I guess I do have a lot.

Interviewer
Gotcha. Like, are you comfortable giving examples of the specific libraries that you worked on?

Participant
Well, that would have been years ago. I don't even know if I remember.

116

117 **Interviewer**
Gotcha. No worries. So is this a case where you need to allocate memory for the particular FFI function, so you're just using a Box as like a standard allocator?

118

119 **Participant**
Yeah.

120

121 **Interviewer**
Gotcha. OK.

122

123 **Participant**
It could just as easily be ellipse, malloc.

124

125 **Interviewer**
Yeah. Gotcha.

126

127 **Participant**
Although if it's some structure you have in Rust and it's going to be passed back to you to free it up, you might as well use Box.

17:... Allocation in Rust
Box<T>

128

129 **Interviewer**
Gotcha. That makes sense. So then, I guess, was that both like the foreign bindings that you worked with in your survey you mentioned, C, JavaScript, TypeScript, and Python? Were those all just different libraries within that category or were there other projects where the JavaScript and Python?

130

131 **Participant**
The test stuff had JavaScript and TypeScript.

132

133 **Interviewer**
Gotcha.

134

135 **Participant**
That was using WebAssembly.. So you have to pre-selfify there a WebAssembly module, just like some, let's call it some Wasm code, and the TypeScript or JavaScript calls into that. Yeah.

136

137 **Interviewer**
And then with Python?

138

139 **Participant**
I've run through examples with Python. I haven't actually programmed

anything. So that seemed like PyO3 to generate things.

Interviewer
Gotcha. Yeah. Gotcha. Gotcha. Did you find in any of these cases that there were challenges in, like, I guess this could include the bootloader stuff as well, because you sort of have different memory models in play there. But in any situation where you've had the Rust memory model and some foreign memory model, are there particular challenges that you've encountered in getting those two to line up? I'm thinking of things along the lines of potentially breaking the aliasing rules, like we talked about with volatile reads. Are there any other problems like that that you can think of?

Participant
I mean, certainly if you're calling into some C code, you have no idea whether it's going to keep that pointer or get rid of it, right? So you can't make the guarantees. You just have to kind of verify trust.

17:20... Difficult FFI

Interviewer
And I suppose especially with the JavaScript interrupt, right?

Participant
JavaScript has complete control over the Wasm memory. In fact, it's just some buffer to JavaScript, and it can read and write freely. I've never, usually you control both sides of the interface, so you never really run into cases where that's an issue.

17:21 Jav... It's not a problem

Interviewer
Gotcha, gotcha. And then with foreign bindings in particular, you mentioned using, like you've both written them manually as well as use different flavors of bindgen. Could you describe your experiences with those? Like what was great about those tools?
What was maybe problematic or incorrect about the results?

Participant
Probably my biggest use of bindgen. I remember the library now, by the way.

Interviewer
Oh, gotcha.

Participant
Yeah, it was [tool], which is a Django tool that runs a kernel driver and then has a library to interact with that driver and use it to communicate with PCI and USB devices.

17:22... Operating & E...dded S

139

158 **Interviewer**

Oh, gotcha.

Participant
So if you want to write a PCI device in user space, you can use this as a stepping stone.

Interviewer
Cool.

Participant
So most of my bindgen use has been generating bindings and consuming them in that library.

Interviewer
Gotcha.

Participant
I found actually there was no issues with bindgen. It did some weird stuff for anonymous unions and structs that are in the C header, but nothing strange, nothing you couldn't change with just the bindgen options to rename those anonymous ones to something more sensible. I never had any issues with the layout of those types.

17:24 I foun...

Generation VS Validati

Interviewer
Gotcha. Okay, gotcha. So it just, no particular problems? Did you ever need to modify the output at all to fit a particular use case or was it just good to go?

Participant
I was scared to because it was very long.

Generation VS Validati

Interviewer
Gotcha, gotcha. Yeah, all right. Yeah, that fits. I've definitely had that experience as well. So, then, let me just pull this up quick. This is a much broader question, but describe a bug that you faced that involved unsafe rust in some way. How you encountered it, what you did to fix it, how you could avoid it.

Participant
Lots of bugs with the bootloader.

Interviewer
Gotcha.

Participant
Partly because although the documentation is very good, it's very easy to

miss things. So, for example, setting up. Oh, geez. Probably more about x86 program than it is about Rust, but setting up the global, setting up memory protections using a global descriptor table, GDT, has to have a certain layout, certain bit flags in places. That's one that I struggled with for probably a week. And all around unsafe Rust. So, it wasn't so much an issue with Rust, but just my understanding of x86. I don't think I've broken enough rules with unsafe Rust to get an error because I've broken the rules.

17:45 So, for ex...
Logical Error
Requirements Bug

17:46...
It's not a problem

Interviewer
Gotcha.

Participant
I've certainly seen lots of cases in libraries that I just, I'll choose not to use the library if I see something that I know that's immediately undefined behavior. Some programs still using, like some libraries still using std mem uninitialized, which should have been phased out in the 2018 edition, but it was only deprecated instead. It's just immediately undefined behavior to call that function.

17:25 I've certai...
Code Smells

Interviewer
Gotcha.

Participant
And certainly that can lead to bugs that would be very difficult to track down.

Interviewer
Yeah, yeah. So, is the, so that particular pattern, like, just in, like, how would you, so when you choose a particular code to use.

Participant
You're looking for more, if I had a bug, how would I go around solving it? Is that what you want?

Interviewer
Yeah, no, I mean that, but I'm also kind of curious about, like, what your process is to decide whether or not a crate is something that's good to use or not.

Participant
Because like, so when you're, when you found that particular use case of mem uninitialized. That tells me that somebody either hasn't maintained the code or disabled the warning.

180 **Interviewer**
Gotcha. Yeah. So are you just like looking through the source code of

dependencies that you choose and then auditing them for unsafe regularly?

Participant
Occasionally I'll audit, occasionally I won't. If I'm interested in how something works, I'll take a peek, right?

Interviewer
I see. And then when you're doing that sort of exploratory thing, that's when you start finding these cases.

Participant
Yeah.

Interviewer
What are some of the other patterns that you've seen where you're just like, I'm not so sure about that?

Participant
Well, using old idioms. Like there's some people that still want to try and use the 2015 edition of Rust. I mean, it's just a non-maintained library, so I'm not going to use it.

17:26... Code Smells

Interviewer
Gotcha.

Participant
I'll do some other patterns. I don't really think there's a whole lot that really turns me off from a Rust library besides just doing something blatantly silly.

Interviewer
Gotcha. Gotcha. Yeah. So I guess with other bugs, the one that you mentioned, setting up this table and setting the right flags for it, that sounds like something that is almost more dependent on the architecture of the BIOS and the system that you're running on in Rust, right?

Participant
Yeah. It's not really that specific. Yeah. For example, I didn't have any bugs with it, but to write this one for chess that did this transmuting to enums, I verified it with Miri afterwards. I used the tooling that's available to verify stuff and write tests. I've never run into bugs because of my unsafe, but even if I have bugs around my unsafe, run it through Miri or Valgrind or something, right?

17:27... Transmute

17:28... Dynamic Analysis Saf

Interviewer

Gotcha. So with Miri, I guess you were able to use it for the chess application, but my guess is because it doesn't support non-Rust code very well, were there any parts of the bootloader that you were able to run through Miri?

Participant
No.

Interviewer
Gotcha. Yeah. So I guess do you have any ...

Participant
Yeah. That's where you go. I think I've probably... I've used Miri before to bug hunt, and I'm thinking of an example now. I was toying around with an idea to constrain the bounds on a new type around a primitive, basically to come up with arbitrary bit-sized integers in Rust. There's an RFC for this, and it's not yet ready for comments or anything. If using a nightly compiler, you can use some flags in Rust to... I think they're called like Rust-C layout minimum or maximum, and you can set the minimum and maximum types, and this just gets passed down to LLVM. And then when constructing this new type around a permanent view, you have to use unsafe to even build it now because you have to verify that it fits within these bounds. This is the same way that the non-zero types in the standard library are made using this. It's a compiler attribute. You're not meant to use it yourself, but you can on nightly if you get around it. So using this to give some bounds, I guess, for a one-bit integer or a five-bit integer, it's still going to take the same amount of storage. However, Rust knows at compile time, because you've told it it has to be, Rust knows that it has a certain minimum and maximum. So then you can use the extra space for niches or that kind of thing and it avoids bounds checks. One of the bugs I had with this was an off-by-one thing. My maximum or minimum was off-by-one because I didn't understand how this Rust internal attribute worked, whether it was off-by-one. It's a classic one.

17:47 There'... Shifting Ground

17:30... Off-by-one
Requirements Bug

Interviewer
Yeah, I've been there.

Participant
Using Miri helped me find what the issue was there because Miri verifies those types when you assign something to them. It verifies their bounds. So then it was able to tell me, oh, the reason why your compiler is making bad assumptions is because you constructed an invalid value here. You told me it was between this and this and it was actually one over.

17:48 Using... Miri found a bug

Interviewer
Gotcha. Were there any other aspects? Did you find that the stacked borrows or tree borrows checkers were helpful? Or was it mostly just the bounds checking that you, I guess, because beyond the safe use, just most of the bit manipulation, right?

233

234 **Participant**
And the transmutation. It was the bounds checking. I've never, I don't think I've built anything that would really need the tree borrows and stacked borrows.

17:31 I... It's not a problem

235

236 **Interviewer**
Gotcha, gotcha. I guess with, do you feel that Miri, like a version of Miri that could interpret or handle the, I guess, more inline assembly instructions and potentially foreign code for the bootloader and BIOS implementation, would that be of use to you? Or do you feel like...

237

238 **Participant**
Part of the reason there is a difference in architecture between the host that Miri is running on and the target that the bootloader is meant to run on, right? Yeah. So Miri is designed to run in 16 bit mode assembly, right?

239

240 **Interviewer**
I guess, let's say, hypothetically, you had a version of Miri that could ...

241

242 **Participant**
Hypothetically, if it was there, it would be useful, but I don't think... I tended not to not do too many pointer manipulations that would pointer to reference, reference to pointer, and that's the kind of thing where Miri would come in handy there, I guess.

17:32 Hy... It's not a problem

243

244 **Interviewer**
Gotcha

245

246 **Participant**
Really, I just didn't get far enough in my bootloader implementation. I could write stuff to the screen, but I wasn't doing much else.

247

248 **Interviewer**
Gotcha. Okay. Interesting. So have there been cases... I guess within that bootloader, when you're writing these abstractions, is this a project that you documented? Or wrote in a way so that you're like... I guess what I'm thinking of is, it seems like a common best practice is for particular functions, especially unsafe ones, to write documentation about like, okay, as a caller, what do you need to establish? And what do you need to expect? Did you end up writing that kind of documentation or doing that kind of reasoning?

249

250 **Participant**
Yeah, because I'm going to be the next one to read it, and I won't forget it.

Audience-Dep...ocume

251

252 **Interviewer**
Gotcha. Could you describe, I guess, some of the types of like invariants that you need to maintain that you would document in that case?

253

254 **Participant**
Yeah, I mean, well... Let's see here. See, usually I also come up with a safe interface around it, so it's hard to break the invariants, I guess.

17:... "Safe" API

255

256 **Interviewer**
257 Gotcha. All right,

258

259 **Participant**
260 So for reading a disk from the BIOS. My safe abstraction around it was to pass in a raw pointer to what's called a disk access packet. It's just a struct that has a bunch of fields in it. And I guess you also pass in a mutable pointer to a location in memory to write the thing to. But I think in that case, so there's some alignment guarantees that need to be upheld there, but I think I was able to express that in the type system.

17:51... Contract or Invariant
Exposing a Safe API

261

262 **Interviewer**
Gotcha. Oh, interesting.

263

264 **Participant**
Most of the unsafe I use, it's very often that I leave an exposed unsafe function that... I don't know, it's very infrequent that I leave invariants just up to like a consumer to worry about.

17:33... Exposing a Safe API

265

266 **Participant**
Gotcha, gotcha. So you were able to successfully encode all of the invariants that you would need to maintain in types that you'd have to have using safe abstraction?

267

268 **Participant**
Yeah, yeah.

269

270 **Interviewer**
I was curious about that alignment type. Could you describe a bit more about how that was laid out?

271

272 **Participant**
The disk is read in sectors of 512 bytes, and memory you write it to should be aligned to 512 bytes.

273

274 **Interviewer**
Gotcha, gotcha.

275

276 **Participant**
The pointer that you pass in is a slice of some type that I've made up that has 512-byte alignment.

17:... Contract or Invariant

277

278 **Interviewer**
Ahh, gotcha, gotcha.

279

280 **Participant**
You'd have to construct that slice unsafely, but in constructing that slice, you'd have to verify the alignment criteria.

281

282 **Interviewer**
Gotcha. That makes sense. Yeah. So then with... let me just double check my... So you mentioned using Miri as well as Clippy, I guess. That's less of a bug-finding tool, more of a linter, but were there any particular things that you found to catch that you didn't realize were... I guess problematic isn't the right word, but how is Clippy useful to you, I guess, in these projects?

283

284 **Participant**
Well, Clippy's helped me catch unnecessary clones. There's just some patterns that are probably bad that Clippy catches and warns you about. Can't think of any off the top of my head. I've had it complained to me before.

17:35 We... Clippy for Clean Code

285

286 **Interviewer**
Gotcha, gotcha. And then are there any problems that you found with the bootloader or just really any Rust application that you've worked on that the development tools you have can't really solve very well?

287

288 **Participant**
Well, certainly for the bootloader, I was using lots of non-Rust development tools. So like a virtual machine with debugging support. Yeah, honestly, the Rust, the ecosystem around Rust tooling is just so fantastic. It's one of the reasons I love it. And if there's something missing, I'd just make it.

17:36... Rust Has Great Tooling

289

290 **Interviewer**
Yeah, that's awesome.

291

292 **Participant**
There's definitely stuff around, like some of the Wasm tooling isn't quite what I'd like it to be. So like programming for WebAssembly. Wasm-bindgen is pretty great, but doesn't yet cover all use cases. Part of the reason for that is the specs for the web stuff aren't ready. The Wasm code that you write can't access the memory of the JavaScript VM. It can only be the other way around. So you need to, if you want to read some

17:37 Wasm... Different FFI Memory
Limitation of Binding T
wasm-bindgen

JavaScript memory, you need to have JavaScript tell you that it has a pointer ready for you. Rust has to return a pointer to its own memory that JavaScript can then write to you and then call back into Rust. And then Rust can finally read it. Wasm bindgen handles all that pretty smoothly, but there's some limitations. There's definitely some more stuff to happen around that space that's going to grow.

17:37 Wasm...
Different FFI Memory
Limitation of Binding T
wasm-bindgen

Interviewer
Okay, gotcha. So you, mostly just things that are limited to the state of WebAssembly and less problems that are necessarily Rust specific.

Participant
I mean, crates.io could use an update.

Interviewer
What would you change?

Participant
I mean, they now have sparse indexing, which is great. So pulling down the most recent index is faster. But it probably needs to be some restructuring and create renaming. I'd love to see create namespaces. And protections against name squatting and stuff like that.

17:38...
Wants Solution

Interviewer
Yeah. Well, I think that covers all of my, all of my questions for today. Really appreciate your time. It's been great talking to you. Really interesting learning about.

Participant
Yeah, right.

Interviewer
Yeah. Yeah, I have worked. I'm more of a compiler's guy. So I haven't worked with anything as low as the, like a bootloader. So it's really interesting to hear about that stuff because it's like the stuff that you're working with like on the daily for that are things that I've like never touched before.