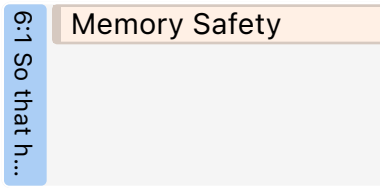
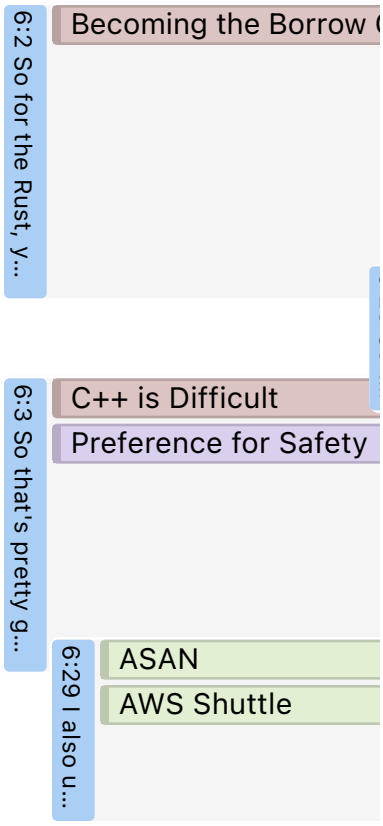
Interviewer
So the first question is, what have been your motivations for learning Rust and choosing to use it?

Participant
So the first thing Rust, I was a C++ programmer. So my undergrad study, my undergrad research was basically based on C++. So we've become pretty low-level stuff like database, very high-performance data structures. And we need to touch the very basic level of the hardware, like interact with hardware a lot. That's true. And we deal with concurrency. So that has a lot of bugs in C++, like it's very error-prone. So at that point, I think it's 2019, and Rust is still relatively new, and this language and look at it, and I found, okay, this is probably, it's really promising, and it has a lot of tool, and it has security, a lot of memory safety guidelines, like a lot of things enforced by the compiler. So I think Rust might be a really good fit because I'm working on the system research, and it is a system programming language.



Interviewer
So could you talk a bit more about your experience writing in C++ compared to Rust? Like, are there any particular ways of thinking or ways of managing memory that you used in C++ that changed when you moved to Rust?

Participant
Yeah, I think, so the first thing I would say is the borrow checker. So for the Rust, you have to manage the lifetime, but in C++, you don't have that thing explicitly enforced by the compiler, but you still have to consider the lifetime, because those things are the critical to make sure that your programming, your memory is correct, but that is kind of optional in C++, and you have to do it after you have refunded bugs, but for the Rust, you have to consider that when you're writing the code, otherwise it won't compile. So that will basically eliminate a very large portion of my bugs that is used like using the stack pointers, like written on a stack pointer of your function, so that will not happen. That turns out to be one of the most frequent bugs that I wrote when I was writing in C++. So that's pretty good. And the other, the second, I think, is the toolchain, I would say, because in C++, you have a lot of choices, but they are not perfect. In Rust, like things are built, I would say the community is more focused on the safety of the programming of the language, so you go to, you have some tools, like easy access for like Miri, and also you have some concurrency checkers that I use in my development, which I found really, really helpful. I also use AddressSanitizer in C++, that's not really helpful because I'm dealing with lock-free developing. So yeah, so that sort of analyzer will not help you with a lock-free programming, but in Rust, you have some tools recently, I think last year, AWS shuttle, which I found extremely helpful. Yeah, I think the tool and compiler is the most helpful.



Interviewer
What was that tool that you mentioned that came out in the last year that

was helpful for you?

Participant
Yes, that is called shuttle from AWS. Yeah, because it is a framework which allows you to check your concurrency code with random permutation.

Interviewer
Gotcha. Yeah, I can see how that would be incredibly useful for that particular set of problems.

Participant
Yes.

Interviewer
So more specifically, what do you use unsafe Rust for?

Participant
Oh, I think I have a few uses. The first obvious use is that I use the raw pointers, because I actually don't, I actually am thinking about how to make those raw pointers into safe Rust, but I haven't figured out a way. So my specific scenario is I have a B-Tree, I can say we have a B-Tree end pointer, and we are going to point to, the pointer will point to either a leaf node or internal node. So there are different data structures. So only when you're reading that memory, you know what type it is. So in that case, I think there's no way in Rust that you can express them without any overhead. So I have to use raw pointers pointing to that place and then reading the first byte of the data structure, which tells me what the type it is. It is an internal node or a leaf node. And after that, I will do a transmute so that I know what exactly, how exactly I should interpret the rest of the memory. Yeah, that's the one. Yeah. Because we have to.

6:4 So my speci... — Increase Performance

6:5 A... — Transmute

Interviewer
No, sorry, you go

Participant
Yeah, so I was saying, for these kind of data structures, you really don't want to pay the extra overhead of, for example, dynamic dispatch, divide in Box, divide in something, because that's the eight byte point, that doesn't become like from eight byte pointer to a 16 byte pointer, which is not great for the high performance computing, et cetera. So that's not great. So you have to do some hack and do some, this kind of stuff. But this is really common in C++, so I kind of got used to it.

6:6 Yeah, so I was... — Increase Performance, Shared Experiences

Interviewer
Gotcha. So to sort of summarize, the issue is you have a scenario with, with your B-tree where you could have an implementation in safe rust,

maybe that would let you represent the idea that your, your node can be like one of those two types. But in order to do that, you would have to use a 16 byte pointer, which would give you performance overhead that you can't accommodate for that use case. So instead, you use unsafe and you read the first byte to tell the type and then use transmute to figure out the rest.

Participant
Perfect.

Interviewer
Gotcha. Okay.

Participant
That's the first kind. I think the second kind is I have to use UnsafeCell. So I use UnsafeCell because I have to get internal mutability. And I have to do that because, so we're writing lock free code. So lock-free code basically use atomic operations to do the concurrency, to ensure the concurrency, to, to protect the multiple writers, et cetera. So there are case, I'll just say a very specific case. That is, we have a mechanism called optimistic locking. So optimistic locking says, if you create a data, you just read it and without acquiring a read lock. And if you have a write, if you have a write operation, you need to first acquire the write lock and then write it. So how do you protect a reader to read data that is being read, concurrently? It will use a version number. And so before you read the data, you'll read it or version number. And after you read the data, you'll check that version number. If that version number changed, that means someone has modified it. So we have to try again. So in this scenario, you're going to read data that is unprotected. But you have to use a customized logic to make sure your data is, is consistent, is an intricate, but that kind of allow you to read the, uh, like dirty data. So in that case, you have to do the UnsafeCell to wrap the data so that you, so that the compiler will not do stupid things with data.

6:7 I have to... Data Structur...and Op
UnsafeCell<T>

6:8 So in this scen... UnsafeCell<T>

Interviewer
Gotcha. So to prevent the default optimizations that you would have from causing undefined behavior, um, since you have that situation, yeah, yeah. Gotcha.

Participant
That's where, yeah.

Interviewer
Oh, sorry. You go.

25 **Participant**
I was saying in the, like, I actually feel the recipe is not ideal for this kind of concurrent sequential case where, where users have really compact,

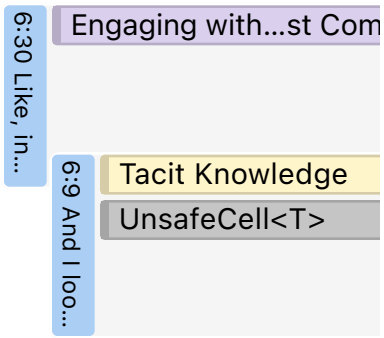
complex, threading model, really compact, correctness model. So like a lot of physics saying if someone's writing is, nobody can read it, but that is not ideal for some scenarios.

Interviewer
Gotcha

Participant
So that's why we have unsafe.

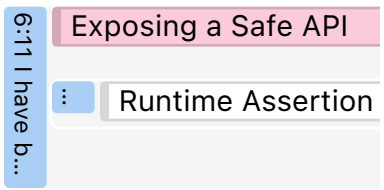
Interviewer
Gotcha. So in, in general, the use cases that you find in supporting this type of concurrency will break the aliasing and mutability invariants of rust. So you're required to use things like UnsafeCell to prevent undefined behavior from happening. Was there a situation there where you initially attempted to implement it without UnsafeCell and then ran into undefined behavior? Or did you just start with that choice from the beginning?

Participant
Like, in the beginning, I was not using UnsafeCell and I believed it caused problems. But, but later I, like, I searched around because at that time I'm still like a new learner. So I didn't know how to correctly use that. And I read some document and look at some referencing presentation or call you to the low level concurrency. And I look at, and at that point I learned that there's something called UnsafeCell. Okay, I feel, okay, I need to use it here to protect undefined behavior. But I don't know. I actually don't know if this actually works. Yeah, I mean, the UnsafeCell, like, whether it has an effect or if I just do it, it will probably work fine.



Interviewer
Gotcha. So from how you're describing this, would you say that you generally are like consuming or using unsafe APIs to accomplish certain tasks more or are there situations where you are also like writing unsafe APIs that are consumed by other people?

Participant
I have been working on some open source roster applications for the most of the cases, I don't expose unsafe APIs, like, I'll say most of them are safe. So I have some internal checks and I'll put some wrappers around the unsafe code and we kind of reluctant to expose unsafe code, like API because that's kind of dangerous.



Interviewer
Gotcha. So I guess when you are writing wrappers in those situations, are there certain pre and post conditions that you need to establish to make sure that the interior unsafe code that you're using is going to execute properly?

Participant
I think most of the cases we can do that, but like, there are certain cases we cannot, like, for example, if we are calling the FFI, we're calling C functions, and if that C function is, it's not, like, if that C function itself is unsafe, then we cannot do that. But most of the cases, we will have some wrappers, for example, like in C API, you probably have some lock and unlock, and that is unsafe because you're directly operating on the lock. So we're going to develop, we're probably going to add a wrapper for them like the lock guard, like on top of the raw lock and unlock manually.

6:31 But mo... Exposing a Safe API

Interviewer
Gotcha. Okay, so it, it seems like there, like in some situations, there will be things like, for example, accessing a vector or other array like structure with an integer, indice, or index, where your, the borrower checker really can't help you, and you need to put the burden on the users to make sure that they're like the index that they pass is safe. But from what you're describing, it seems like there aren't situations like that. It's more that you are just taking these unsafe operations, like, you know, more that you are just taking these unsafe operations and encapsulating them, and then presenting them to the user. So the user assumes that if they use the interface, it will be safe. But there isn't like an additional obligation on them. It's just all handled by you.

Participant
Yeah, but I was, they just have some overhead like runtime. I'll also add the check.

6:3... Runtime Assertion

Interviewer
Oh, okay, gotcha.

Participant
So, that's not a deal.

Interviewer
Gotcha. So you do add runtime checks to ensure that certain conditions are met.

Participant
Right.

Interviewer
Gotcha. Like, what, what, what, kind of conditions would you usually be inserting runtime checks for?

Participant
Like a bounds check, like, what is that? And also, sometimes I would do

6:1... Runtime Assertion

alignment check, because we're dealing with, we sometimes ensure that your pointer passing in this, like, properly aligned.

6:1... Runtime Assertion

Interviewer
Gotcha.

Participant
And, and we also have some, I think that's the most common case.

Interviewer
Gotcha. Yeah, no, that makes sense. So just moving on to the next section here. I need to pull up your responses to the survey quick. So, we've discussed a bit about UnsafeCell, but you also mentioned using Box, MaybeUninit and ManuallyDrop. So could you describe a bit about your use of those three additional types in unsafe contexts?

Participant
Yeah, so the ManuallyDrop, I think is I want to prevent. And yeah, so I think ManuallyDrop is basically associated with use of MaybeUninit. So you have a, so you have, in a struct, you have a field that may not be initialized at the time of the construction. And I forgot exactly how to use that. But yeah, there's a scenario where I have to do that, because in initialization time, there's not available. And, and I, at that time, I also cannot use Option something, because Option something will make it larger.

6:13 Yeah, so the Manuall... Increase Performance, ManuallyDrop<T>, MaybeUninit<T>, Option<T>

Interviewer
Gotcha.

Participant
Yeah, make it larger. So I have to add a vericing or wrapper. So I found that that should not be MaybeUninit. And because of that, oh, I try not to. Okay. Because of that, we have to make sure that that that field is not automatically destructed. So it will not be dropped. Yeah, so I think MaybeUninit is inherently ManuallyDrop. But yeah, the case that we have to make sure that, oh, actually, if I think of like a ManuallyDrop case that I use, is that when I have a data structure that is a linked list, something like a link list. And if you, if you automatically drop, then if the link list is the chain of link list is really long, then you're going to drop from the end of the chain. So you start, you start from distracting, distracting, and you push every, every data structure, like every data structure on the stack. And until you reach the end of the chain, that drop will happen. And then you're going to pop up all all the items, all the strats, like pop all the strats, and we call the drop. So in this case, we want to enforce like a drop order, so that it will not have some stack overflow. In that case, we just manually drop.

6:14 if I think of like a Manually... Increase Performance, ManuallyDrop<T>

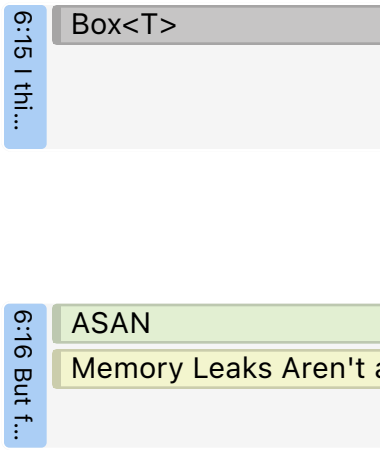
71
85 **Interviewer**

Gotcha. Okay. So just overriding Rust's default drop order to improve performance in cases where you have a very large data structure.

Participant
Yeah, yeah, kind of.

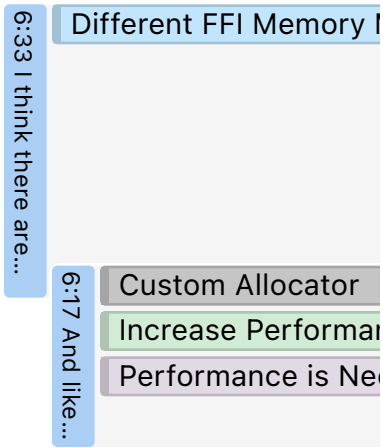
Interviewer
Gotcha. Gotcha. And then with Box, I know that type can be a bit tricky because once you unwrap it, and expose it as a raw pointer, you have to remember to rewrap it so that it can be dropped. Has that ever been a problem?

Participant
I think, well, I forgot to say that my use case is unwrap a Box. So my case is, yeah, so sometimes we need to get the raw pointer and you can do the allocation a lock, but you can also do a Box and then unwrap it. I think they're the same. So yeah, I use them interchangeably. And so what you're saying is you can only, so when your Box is structured, you get the pointer and when you deallocate, you have to make it a Box again. And if you forget to do that, you're in trouble. And if you do that with the wrong pointer, you're also in trouble. That basically is the case of like a C, something like a yeah, you have to do. But for me, I think it's not a big, I haven't found bugs involved with that property because I use address sanitizer. Because I think this kind of bug is quite easy to check because if you leave memory, then you don't immediately be checked. And yeah.



Interviewer
So would you say that there are, so in your use of Box, is that typically cases where you would, like is that allocation only staying on the Rust side, or are there situations where you allocate memory using a Box and then pass that memory across a foreign function barrier?

Participant
I think there are cases when we need to handle string in front of another language. For example, when I do a string between Rust and Python. So Python string, like you pass a Python string to Rust and you use it and you cannot deallocate it because otherwise you just break Python. So you have to give back, like give back to Python. And I think that's the case where we have allocation where, because the memory is not from Rust, it's from another language. And you have to make sure that you're not accidentally deallocated. And like in our application, we actually have our customized allocator. So it's not like we call allocator all the time. We will have, because allocator, like the system allocator is really slow. And in the very high performance applications, you don't want to call the system allocator all the time. So what you do is you will allocate a bunch of memory, like probably one gigabyte of memory from the system as the initialization fits. And after that, you're kind of manually, like you kind of get our one gigabyte of memory pool and then you will manually specify, okay, I'll use this region for this purpose. And after that, maybe it's a very



simple allocator, like say, each slot is only like a cache line. So you use that and you frequently get memory from that and return it. So it's faster. But in that case, you cannot use Box, because once you bought it, if you deallocate that portion of the memory, then I think that's a good thought. And so basically, that essentially says that we cannot use Box in our application because we use our own allocator. Yeah. And to do that, we actually have some like a wrapper around this kind of thing, like a similar to Box, on Box.

6:18 But in that...
Box<T>
Increase Performance
Unsafe Data Structure

Interviewer
Gotcha. So you avoid use of Box in that situation because using your own allocator gives you better performance, but you still need to create your own version of Box to be able to have the same usage patterns?

Participant
Right.

Interviewer
Gotcha. Okay. So you also mentioned using some different methods to create bindings to external code written in C and C++. You specified that you use c-bindgen and CXX. Could you speak a bit more about your experience using those tools to create bindings?

Participant
So I forgot what I did is I tried to generate the Rust bindings for C application and also tried generate C bindings for Rust application. So I forgot which is which. So to generate the bindings, I think it mostly works fine because C is really simple and it's pretty straightforward. But what you get is very unsafe functions. You have to do all the stuff that's unsafe and it's pretty error prone because you always have to use the C types and use the C function unsafe every time. That's not great. But that's not my major use case. My major use case was previously was to generate C++ bindings for Rust and that is really challenging because C++ is really complex. And at that time, CXX was still in the early stage. But I don't know what the current state of the CXX is. At that time, I feel that library has a really smart idea and it really works. But when you scale up, so when you have to deal with a really complex C++ library, that can be quite challenging even with CXX. And I think the major goal of CXX is not to generate binding for C++ projects. It's more of you develop some part of your coding C++ and also you develop some part of a coding Rust so that they can coexist. In the end, I would not be using both of them now because I feel it's just not like the experience I had. I think it's one year ago. A year ago, it's not great. And now I would rather just use regular rewriting Rust.

6:19 So to g...
Bindings are a Fiddly M
Generation VS Validati
Preference for Safety

6:20 My major use case was...
CXX
Limitation of Binding T

Interviewer
Gotcha. So just to summarize, tools like CXX has a great idea and has a good experience if you are working equally in Rust and C++ and you just need interoperation between two evolving code bases. But it isn't a great

idea if you are just taking a C++ library that's like an existing thing and trying to create Rust bindings for it.

Participant
Gotcha. Yeah. That's my question. Gotcha

Interviewer
So you tend to write bindings yourself. Has that led to any problems relating to alignment or proper type usage when you write those bindings?

Participant
I think I was concerned about this. So I was clear and that's why I didn't go that way because I tried to write bindings and I found there are so many complicated things I have to deal with. For example, the allocation, the deallocation, the alignment and the different typing system. And yeah, a lot of things that are really complex and challenging. So yeah, I was scared. So I didn't do that way. I tried and I have some samples working. But to the scale of my application, which is database system, I feel that's too much overhead, too much mental burden for me.

6:21 I think I was concern...

Bindings are a Fiddly M

Interviewer
Gotcha. So did you end up just using C bindings?

Participant
No. I tried to use C bindings. And I also tried C bindings. And I think I also feel that is not great. So in the end, I just used the Rust implementation.

Interviewer
Gotcha. So you just moved away from foreign functions entirely because using these tools, the tools weren't adequate and writing it yourself would be too much extra mental burden to think about. So you just decided to do it all in Rust.

Participant
Okay. Gotcha. And also, I want to say about FFIs, it's really difficult to get correct. Even to get a real working demonstration because C and C profiles are really complex building systems. You can do Bazel, you can do CMake, you can do make, and to compile things and to deal with the name mangling. And you need to correctly link things and stuff. That's just not great. I think I have a demo to set up this kind of linking. And that's really complex. And the people have been found useful, but I personally think very few people can really get it correct. And another thing about FFIs, when you call a shared library, you basically prevent you from doing the linking time optimization. Oh, sorry. So when you call a shared library, you basically have to use the linking time optimization to make sure that your function call is your overhead. Otherwise, every function, every shared library call will be a real function. So you have to pay the

6:34 And also, I want...

Bindings are a Fiddly M

overhead of, for example, callee save and caller save and saving the same thing on stack. So that can be slow. And so calling foreign function is not their overhead. And you have to use linking time optimization, which is really slow itself, like compile time, double or triple or compile time. It's not great. And it's really difficult to make it correct. I think Firefox has a blog post about this, how did you make linking time optimization? And they just take on probably a whole year of the entire FFOS team to figure out how to do it. So yeah, you have performance penalty, and it's really difficult to get it correct. And yeah. Yeah, I think so.

6:22 And... Linking Time...timizatio

Interviewer
So in terms of, I guess, performance too, but in terms of the memory models of C and C++ and Rust, were there any challenges that you faced more specifically in, like aside from the bindings, were there challenges that you faced in reconciling the differences in the memory models of Rust in these languages?

Participant
In that memory model, you mean, how do we manage our allocation on the allocation? Or how do we do the correctness concurrency?

Interviewer
I guess I'm thinking of sort of three different things here. Two of them are what you just said, the allocation, deallocation, and the concurrency. I guess the third is also Rust's model of having like a Rust borrowing model and the optimizations that it applies to borrows versus raw pointers in C and unsafe Rust.

Participant
So you're saying when we interact with like those two different languages, I call it the language boundary, will that be like any difference or anything?

Interviewer
Yeah, like have there been problems that you've found because of the fact that across this language boundary, there are different assumptions being made?

Participant
I see, I see. Okay, so I think the first thing is allocator. allocator is really complex because theoretically, you cannot assume that you will see application and Rust application are using the same allocator. And that is definitely you cannot do that. But in practice, sometimes you have to, in practice, our high-performance applications have used some runtime overrides of the allocator. So they will override your default system allocator with some more high-performance allocator, for example, jemalloc. In that case, because the C and the Rust are both compared to the same machine code. And when you reference to the allocator, they actually reference the same allocator. So when you replace that allocator, you actually point to the same allocator. So yeah, that caused confusion

6:23 Okay, so I think the first th... Allocation & Alignmen Deallocate For...emory

to people. When you say, okay, you are not a different allocator, but in fact, it is.

6:2... Allocation & Alignmen
Deallocate For...emory

Interviewer
Gotcha. So to clarify there, it's by default, if you allocate something in Rust and then free it and see, that would be undefined behavior, because you can't guarantee that the allocators are the same. But in your domain where you're working with high-performance computing, they end up being the same, because you just ensure that everything is using, okay, gotcha. So having to manage that is trickier.

Participant
Yeah, that's tricky. But we basically just don't want to do that. Yeah, because that is too tricky and great. And also, I think most of the pairs we interact with see is that we have to interact with our main system. So we have to do the libc. Okay, I was talking about this, and we have to interact with our main system and the libc. So go to the library. And luckily, the libc is like a world design, I would say, so that most of cases, also, Rust has official support for libc. So they have good documentation if you call this function, you have to make sure several things in Rust. That's really helpful. And for example, there's a libc called memory map. So that function is really complex. It'll take a lot of flags, a lot of parameters, and using that in Rust, yeah, I forgot. That function basically says, okay, I'm going to acquire some memory map system. And I'll say it's actually similar in Rust. And actually similar in C and Rust, because in C, you call that function. And in Rust, you also call that function. And that's basically the only way you can do it, because in Rust, when you interact with the system, how to do it with libc, and yeah, that's it.

6:3... Simple FFI

6:36 Rus... Engaging with...st Com

Interviewer
Gotcha. So the only difference there would be that when you're calling these libc functions from Rust, you need to ensure certain preconditions to make sure you're doing it safely because of all of the things that Rust needs to guarantee. But the documentation is very good. So if you just read the documentation, that's enough for you to do it safely.

Participant
Yeah, yeah.

131

141 **Interviewer**
Gotcha.

142

143 **Participant**
Yeah, I think so.

144

145 **Interviewer**
Gotcha. So the next question is sort of broad and applies to any case where you've used unsafe Rust. So I want you to describe a bug that you

faced that involved unsafe Rust in some way. And for that bug, talk a bit about how you found it, like how did it manifest as well as how you fixed it?

Participant

Yeah, I'm thinking about a lot of bugs. I mean, most of the bugs aren't difficult. And I think the challenging bugs are those that deal with concurrency, because that kind of, it just breaks the Rust assumption about like the reference model. Because you're basically saying, you can read the data that somebody is writing. And yeah, I think most of them are like the native to the UnsafeCell have to ensure the correctness of my concurrency code have to do the, for example, when I read, whenever I read the data, I need to first check the version number. And after read it, I need to check the version again, make sure that it's correct. And then on top of time, I just forget to check that again. And when I forget to check it, it's definitely a bug. Yeah. And this, this kind of stupid error, like, like stupid bug happens a part in my code. And I'm considering doing something about this, like have some, some buffers over this behavior. But I haven't found out, like, there are, there are, there are some optimistic concurrency control library in Rust, but they are not really, they're overhead. So you actually pay some overhead with, like, all kinds of stuff. So I cannot use them because our system really cares, cares performance about that. So, yeah, actually, I haven't figured out a way to make me to like, for example, check the version number before reading and then check the version number again after reading. This kind of bug happens a lot. Actually, it happens. It takes a lot of time because concurrency bug is really difficult to find. And also, the case that Yeah, like, similarly, like, when I check the, when I read the code, okay, I'm just doing the same example, because I'm doing commerce and like, optimistic read, I read the version number and I read the data. And then when the data is being changed, so I read data, the first byte data is inconsistent with the second byte of data. And, but I was not aware of this. So I end up with a really random value. And it will end up executing some very, like, random things. And that will lead to a second fault. And yeah, because that data is not protected. And I was using the, like, the shuttle, the shuttle library had talked before. I used that to detect that, okay, this is not, this is bug. I cannot do it myself, because it's really difficult to reason about. And I was not totally unaware of this kind of unprotected read.

6:24 when I... Concurrency Bug
Data Races

6:25 But I ha... Increase Performance

6:37 So,... Contract or Invaria

6:38 And I w... AWS Shuttle
Unsafe is Diffi...to Und

Interviewer

Gotcha. So, to summarize a bit, it seems like because of your performance needs, a lot of the Rust standard library types, that deal of concurrency, as well as some of the other libraries that you could use are sort of off limits. So, you end up running into these very challenging concurrency bugs relating to, in this case, unprotected reads, like you're talking about, or freeing data check for something. And your main tool for finding those is the AWS tool that you mentioned.

145

151 **Participant**

Yeah. But And also, the AddressSanitizer.

Interviewer
Gotcha. Gotcha. So those, those two tools are, would you say that those two are your main, like the main two tools you use?

Participant
Yeah, those are main tools I use to detect concurrency bugs. And also use LLVM's libfuzzer. Yeah.

Interviewer
Oh, gotcha. You also use libfuzzer.

Participant
Yeah.

Interviewer
Could you describe a bit more about how you use that to find bugs in your code?

Participant
So my code, like, I just, one example, my code is like a B-Tree, a data structure, you put data, and then you can read the data later. So you want to enforce some invariance, for example, oil operation will be will not stick for you and not be killed by system. And also, the data you, you read before, you can read it back. And so what I do is I have a reference B-Tree in the standard library, which is low, but it's, I believe it is correct. So whenever I do an insert operation, I will insert my post to standard library B-Tree and my B-Tree. And I will check if their, their return value is correct. Like, if they're return value is the same. So if they behave differently than them, definitely take a bug. So the, the fuzzing, the fuzzing loop will generate input to my B-Tree. For example, in the read, update, and then just scan, etc. Yeah. And basically have a standard library, the oracle, the standard, the standard library B-Tree, and then compare the, their values.

6:26 So whe...
Fuzzing
libfuzzer

Interviewer
Gotcha. Nice. That's a really clever way of testing. So you also mentioned you have, so we spoke a bit about address sanitizer. You also use Clippy and memory sanitizer, at least you, you mentioned that your survey response. Have those tools been particularly useful for you?

Participant
Um, Clippy is really useful. I, I, I have it with my, you know, like the code CI. So every time I commit something, it will run on Clippy. And I think Clippy is a very low-high influence you can get because it's really helpful. But I think that is like the remaining of our synthetic, like a static level. So

6:27...
Clippy Doesn't Find Bu
Clippy for Clean Code

cannot do a lot of things, but it definitely checks some patterns, your code that is not great. It's not for readable or, I don't think it can capture bugs, but it definitely makes your code more readable. And memory sanitizer, oh, I forgot. I actually use address sanitizer more. And the memory sanitizer is mostly about, I think I'm leaking, memory leak. Yeah. Because those sanitizers are incorporated into the fuzzer. So when I run the fuzzing, like libfuzzer, it will like enable the sanitizer by default. Yeah, I also use Miri before, but Miri is more limited because I have to deal with a lot of IO operations and a lot of functions that are not supported by Miri. At least the time I'm using it. And it's pretty useful. So it's both the speed and the fact that Miri is not, like it just doesn't support the types of functions that you need.

6:27... Clippy Doesn't Find Bu
Clippy for Clean Code

6:40 Miri is... Miri doesn't s...ort this
Miri is slow

Interviewer
Gotcha. Yeah. Would you be able to send me a link to the AWS bug funding tool that you mentioned?

Participant
Sure. I'm going to do that. Send to the chat.

Interviewer
Awesome. That's great. Could you describe some of, like another example of some of the bugs that you find using shuttle?

Participant
Yeah, I was trying to find another bug. Most of the bugs are unprotected read. So read some data. And only with a particular scheduling, the bug can be triggered. So in a lot of cases, you just cannot reproduce the bug if you are using the system schedule. So what shuttle does is it kind of hijacks or intercepts your code such that the stress schedule is controlled by shuttle. So it will promote your way of scheduling. And we'll find out. Okay, I think a country bug is, I have, I have an integer overflow bug, which, in my case, I have a counter. And the counter to save space, I use U8. So it's a very tiny one byte data. And I think that should be enough for the most of cases. And it is enough for the most of cases. But with shuttle, you kind of will test with some extreme scheduling. And in that case, it will trigger the bugs. That's a, in my case, it actually will, I think the overall flow. So that, I think that is a case where it's almost impossible for me to test during my development. It can only be unveiled using this kind of tools.

6:41 I have an integer ov... Concurrency Bug
Integer Bound...Overflo
Integer Overflow

Interviewer
Gotcha. So have there been cases where you've had bugs that your development tools haven't been able to handle? Like are there problems that you face when writing on safe rust that you don't have a clear solution to in your tools?

Participant
Yeah, I think that's a lot of fun. Because shuttle only handles the

concurrency permutations things. But in order to use shuttle, you have to change your code. And also because shuttle is not the most intelligent, like, oh, okay, I'll say this. Because concurrency, there's millions of ways to do the same, to run the execute the same code. And there's no way for you to, like, do the model checking to try every combination. Because there's way too many combinations. And I think that's NP-hard. Because of that, when you're code, when you're like code-based, when you're testing large scale systems, you have to shuttle around and like forever. It will not take stuff. And the chance of you having you be finding a bug is really low because you're not running a lot of permutations.

Interviewer
Gotcha. So it seems like it's cases where the scale of your program can lead to having problems with shuttle. And then you also mentioned with Miri that due to just the nature of your development, Miri won't, similar in terms of the slowdown.

Participant
Yeah, slowdown. The different combination you have to try versus limited time.

Interviewer
Gotcha. Yeah. So it's, I can see how fuzzing is like a natural solution to that, given just the necessity of trying as much as you can.

Participant
And also, the fuzzing has the problem that you can only run single thread code.

Interviewer
Oh, okay.

Participant
So yeah, at least for my version, I don't think fuzzing can do the permutation for me. So I can only test with single thread. Like, I mean, the single thread version, but you can run multiple instance of the testing, but they don't interact with that. They don't have concurrency things happening. So I use fuzzing tool to make sure my single thread version is correct. And I use a shuttle to make sure that, at least to get the idea of how correct it is for my multi thread version.

6:42 So yeah, at le...
AWS Shuttle
Fuzzing