Ruby Rogues

155 RR Why Ruby Sucks

JAMES:

Where's David? Did he fall into the water trough?

CHUCK:

How did you know he has a water trough?

[This episode is sponsored by Rackspace. Are you looking for a place to host your latest creation? Want terrific support, high performance all backed by the largest open source cloud? What if you could try it for free? Try out Rackspace at RubyRogues.com/Rackspace and get a $300 credit over six months. That's $50 per month at RubyRogues.com/Rackspace.]

[DevMynd is a software design and development studio in Chicago with expertise in Ruby, JavaScript, and Clojure. We believe that well-crafted software makes life better. And our team of designers and engineers is dedicated to that pursuit. We love our customers, we love our team, and we spend a lot of time and effort making sure that we fit the right projects with the right people. Get in touch at DevMynd.com.]

[This podcast is sponsored by New Relic. To track and optimize your application performance, go to RubyRogues.com/NewRelic.]

CHUCK:

Hey everybody and welcome to episode 155 of the J- [Chuckles] JavaScript J-.

JAMES:

Ouch. Ouch.

[Laughter]

DAVID:

Keep going. Keep going.

[Laughter]

DAVID:

You've made this mistake so many times, Chuck, Mandy, that the episode is on. Just pick it up.

JAMES:

We're going with it. [Chuckles]

DAVID:

Keep going. Keep going.

CHUCK:

Hey everybody and welcome to episode 155 of the Ruby Rogues Podcast. [Chuckles] This week on our panel, we have Avdi Grimm.

AVDI:

Hey, I thought this a C++ show?

CHUCK:

James Edward Gray.

JAMES:

I'm only staying if we're talking about Lua.

CHUCK:

David Brady.

DAVID:

I'm David Brady and this episode really grinds my gears.

CHUCK:

I'm Charles Max Wood from DevChat.TV. And this week we're going to talk about why Ruby is not the Promised Land, or maybe why there are other Promised Lands.

JAMES:

That does it. I'm hanging up.

[Laughter]

DAVID:

Yeah.

CHUCK:

Whose idea was this?

DAVID:

We're going to take some sacred cows and make some delicious, delicious hamburger.

JAMES:

It was Avdi's idea. He's the resident curmudgeon.

[Laughter]

CHUCK:

Yeah, we got him complaining about Erlang in the Rogues channel.

AVDI:

Come on. You all know this language sucks.

JAMES:

[Chuckles] That's right. Ruby sucks.

DAVID:

It's true. It's true.

JAMES:

So, we're going to talk about specific problem points and where they might be better, kind of a grass is greener sort of approach. So, what's a problem point? Throw one out there. DAVID: So, should I start with the really big one that had the disclaimer that we talked about in the preshow?

JAMES:

Disclaimer. Oh, nice.

CHUCK:

I didn't see a disclaimer.

DAVID:

Okay, so my biggest axe to grind right now with Ruby requires a disclaimer. And the disclaimer is this: there's a rule in open source that says don't complain, only fix. And I want to be careful with that rule. Sometimes I want to build cathedrals, not make bricks. And if I bash on about the lack of a particular type of brick, I don't mean to disparage those of us out there toiling away making bricks. Keep up the great work, guys. The one exception I'll make to this disclaimer is that I want to talk about version mismatches between Ruby 1.9 and Ruby 2. And there's no other way to say it than I want our gems to work on both versions of Ruby and sometimes they don't. And I don't have time to fix the entire Ruby ecosystem so I'm just going to complain instead.

JAMES:

[Chuckles]

CHUCK:

Isn't this the point that, I think it was Avdi made a few weeks ago about debuggers?

JAMES:

Yeah. The debugger, it's probably the penultimate example because it seems to break with every…

AVDI:

[Laughs]

JAMES:

Every single version of Ruby?

DAVID:

Yeah. Ultimately what it comes down to is in my opinion there is currently no stable version of Ruby that also has a robust ecosystem. There are widely used tools like MetricFu that aren't ready for Ruby 2. As of a month ago, Ripper still wouldn't rip Ruby 2 syntax. I think that's changed. But there are other widely used tools like guard that have amputated their Ruby 1.9 support. They just don't run. Rb-readline won't work on Ruby 1.2. And Guard has switched to use rb-readline instead of the rb-notify and the different types of readline support that it had. So, in some languages, you want to change things. And you have this trailing edge of really old legacy stuff and then you have this stable platform where everybody gets along great. And then you have this sloping forward bleeding, leading edge of the language. And right now, I feel like Ruby 2 is bleeding edge and not work-y. And we have amputated the tail of 1.9 support in some areas. But we've amputated so close to the bleeding edge that there is no stable platform in the middle.

AVDI:

So, what language has a stable platform?

DAVID:

I'm going to go ahead and eat my own words on this one. I have publically derided Python for freezing core for two years. Guido stopped development on core. He basically stopped

development on the reference language so that everyone else could catch up. And that was right at the time that Ruby was saying, “Hey let’s go from 1.8.7 to 1.9 and let’s not be backwards-compatible. Let’s jump headlong into the future.” And Python at that same time literally was freezing for two years. And I mocked Python ridiculously and this just totally showed how Python were a bunch of stick in the mud backwards fear of progress people. And boy, those words are really hard to chew. But boy golly, I’m going to have to eat them because we’re now reaping a little bit of the dark side of having jumped forward so fast and cut ties with our past so quickly.

CHUCK:

So, is there a way to solve this problem?

JAMES:

Let me say this, because this is interesting. I have not felt this particular pain with Ruby 2. I definitely felt it in the 1.8/1.9 era and I assume that’s no… when we were making that transition I’m pretty sure the whole world felt it. I thought Ruby 2 was fairly smooth. There were some changes. I had one particularly large app that began to segfault on it which was really unfortunate and caused some problems. And then the Ruby 2 to Ruby 2.1, to me that was almost a plug and play. It seemed perfect. So, I find it interesting that you run into that a lot lately and I haven’t. You mentioned Ripper. Ripper is included in Ruby. And I wasn’t aware that there was any point that it couldn’t parse its own syntax. It’s actually really interesting if you look at how it’s implemented. They cover it a little bit in Rum because the parser basically now has if-else’s at almost every node that it parses. So, the if is, am I parsing this for Ruby itself in which case I do this thing to get it ready to go to Ruby, or am I parsing it for Ripper? And so, that, it didn’t do that. So, I feel like the Ripper integration is really amazingly good. But I haven’t seen the…

DAVID:

I may be fingering the wrong gem. I think it’s Ripper. But a month ago…

JAMES:

But Ripper’s not a gem. It’s included.

DAVID:

Sorry, yeah. Wow. I may be actually having a beef with Ruby itself then. All I know is I wrote a program that had keyword arguments. And I want to say optional keyword arguments, so we’re talking 2.1 stuff here. And it just blew up. MetricFu, one of its gems, like Saikuro I think uses… and you know what might have been happening is that Saikuro might have been using an older version. I’m not sure how that’s even possible. But it might have been using an older version Ripper to parse the Ruby syntax. And yeah, I lost all my complexity metrics because the language couldn’t be parsed anymore.

JAMES:

That’s interesting. I haven’t run into that problem but I admit I haven’t used MetricFu in a while. And Guard I think was the other one you named with readline. I haven’t hit that particular problem either. I do use Guard, but I don’t think I’ve used it very recently. So, I may have just dodged that bullet. I don’t know. Avdi, Chuck, have you run into trouble with the Ruby 2 transition?

AVDI:

It’s been smoother than previous ones.

DAVID:

I’m just the lone voice crying in the wilderness. [Laughs]

CHUCK:

I've run into things occasionally that don't… For example, I move ahead when there's a new version of Ruby out. So, I'm writing code on Ruby 2.1. And occasionally I'll run into something that doesn't play real nice. There's some little gotcha between 2.0 and 2.1 or 1.9 and 2.1. But it's pretty rare. And most of the time I can find a way around it without too much hassle.

JAMES:

One of the issues Dave brought up is really good I think, and that's keyword arguments. In my opinion, keyword arguments didn't really become fully-baked production-ready this is a go until Ruby 2.1, which was kind of unfortunate. I almost wish they could have shipped in that state originally because it's this weird thing now when I'm using Ruby 2. We have an application at work that's tied to Ruby 2 and it's like, "I'll use keyword arguments," and then I forget, "Oh they still have that one major disadvantage here."

DAVID:

Yeah.

CHUCK:

Yeah.

DAVID:

Last week we talked about games with Megan Fox and I mentioned that I might go off and do Ludum Dare with Ruby Gosu. And I was going to explore the Releasy gem to ship to Windows. And for anybody who's interested, I did actually chicken out largely because the Saturday of Ludum Dare turned out to be my wife's birthday, which I as an ignorant standard bone-headed husband forgot. So, through no fault of Ruby's, that didn't happen. But I did take a minute to install Releasy. And it requires, if you want to build on Linux which I do and you want to ship to Windows which I did, you have to use Ruby 1.9.2. And all sorts of things fell apart. MiniTest was available as a gem in 1.9, wouldn't run. And what happens is Bundler just gives up. It says, "I can't give you the version you need." Even though that version, a version that satisfies the Ruby version number requirement, even though a version of that gem might exist somewhere, Bundler can't find it. It just gives up. And so, you're at that point. You're literally going out to GitHub and digging through the commit history of the gem to find which commit message to clone and build manually and install. I will admit that 95% of the time, Ruby and Bundler are great and they work fantastic and they're seamless. But the other 5% of the time, we have no package management. And our pants are down on the internet.

CHUCK:

So, I have to ask this. At what point is it appropriate to drop support for 1.9 or 1.8?

DAVID:

Never.

JAMES:

Well, 1.8 is dropped.

CHUCK:

So, you drop support when the core team drops support?

DAVID:

My code still runs on 1.8.4.

JAMES:

Wow. That would be awesome.

DAVID:

Not really. Not really. I actually have code that doesn't run. So, I'm a hypocrite here. TourBus doesn't run. We dragged it kick and screaming to Ruby 2. But Migratrix won't run on Ruby 1.8. It just won't. So, this is hard.

JAMES:

I think the core team is usually fairly conservative in Ruby Land as far as support. Ruby 1.8 was only, 1.8.7 was only end-of-lifed was it last year or the year before?

CHUCK:

I think it was last year.

JAMES:

It hasn't been very long, yeah. And to me, that was really conservative. I was long past it then. And I felt like the gems had caught up. I don't know. It's an interesting problem. I do agree that Ruby tends to push forward and occasionally there will be a release that breaks a lot of things. I feel like they're slowing that some since the Ruby 1.9 transition which was brutal, in a word. I don't know. It's an interesting point. There are definitely languages that are more stable, move slower, stuff like that. But I think we should stress, there can be downsides to stuff like that. For example, when did Erlang get Unicode? [Chuckles]

DAVID:

Does it?

JAMES:

Wasn't that also fairly recently?

DAVID:

Does it have it? [Chuckles]

JAMES:

I'm just saying. You can go too slow, right?

AVDI:

Yeah. It's also nice to have more explicit constructs in the language for saying I am going to use something from the future or I swear that I am totally compatible with the past.

JAMES:

Ooh, what languages do that?

AVDI:

Well, we talked a little about Python. Python has the import from future thing where you can import language features that are not, I guess, it's been a while since I've done Python but basically import language features from the future that aren't part of this current release but they're in there experimentally.

CHUCK:

JavaScript does this as well.

AVDI:

And then you can see that in the file. An analyzer can see that in the file and say, “Oh okay. This file probably isn’t going to work on a version that doesn’t even have that in its future libraries.”

DAVID:

Yeah.

CHUCK:

JavaScript solves it in a different way. They actually have transpilers. So, ECMAScript 6 for example is not in current browsers. But the Angular core team is writing Angular 2 in ECMAScript 6 and then transpiling it to ECMAScript 5.

AVDI:

Which is cool. And that kind of thing is incredibly difficult in Ruby.

CHUCK:

Yeah.

DAVID:

Yeah.

JAMES:

Yeah.

AVDI:

Which brings me to one of my complaints.

JAMES:

Bring it.

DAVID:

Wait. Before you do that Avdi, I wanted to give a specific shout-out. Miah Johnson, she is the reason that TourBus runs on Ruby 2. As much as I complain about gems that don’t run on two versions, I didn’t want to maintain my own gem. And she was the one who kept after me and pushed and helped. And we got it running over v2. So, thank you Miah.

JAMES:

It is super cool when people go and take old libraries and fix them.

DAVID:

Yes, yes.

JAMES:

So, that’s awesome. Alright Avdi, you were going to tell us.

AVDI:

Or like when Erik Michaels-Ober took my Naught gem and said, “Hey I want to use this in my gem.

But for that, I need it to be 1.8-compatible."

JAMES:

Yeah.

AVDI:

And moved mountains, did things that amazed me, to make it. There were things in there I didn't think was possible to port, but he managed to make it happen.

JAMES:

Wow.

DAVID:

Yes.

AVDI:

And now it supports 1.8. So, yes Ruby is incredibly difficult to parse. The parser code is a nightmare for anybody, for implementers. And the reason it's incredibly difficult to parse is because it tries to be super friendly to programmers who like writing really expressive code, which is nice. It's one of the reasons that I use it. But here is the thing. You have your static languages. You have your C++ and your Java. And you have your nicer static languages like Haskell, et cetera. And those, they're static so they get compiled. And compilers and other tools that are smart enough can learn a great deal about the code. They can make all kinds of interesting predictions about it. And they can be certain about things at certain points in the code just by analyzing it statically, because they're made to be analyzed statically, which means that you can build really cool refactoring tools for them. You can have tools that can very easily go through and do things like, "Oh hey. You want to extract that method out? Well, I've found three different places that I've determined without a shadow of a doubt are exactly identical to that code. Would you like me to replace those with calls to your new method?" And you have a lot of neat stuff like that. You can do a lot of really advanced refactorings automatically. And an automatic refactoring is nice because it means you can think in terms of domain and semantics. You can say, "Oh I want a new thing," rather than, "I need to type, I need to make 15 different changes in order to have a new thing." And then you have your dynamic languages. And one of the biggest influences on Ruby in this regard is both Smalltalk and Lisp. And they have none of these guarantees. Everything's only known at runtime. An object could have methods added to it at any time. So, you can't make any of these predictions. But they make up for that because in Smalltalk there is only runtime. There is no such thing as not runtime. So, it makes up for the fact that you can't make any static predictions about it by the fact that the system is always running.

DAVID:

[Chuckles]

AVDI:

You can always ask the system, what methods does this object respond to? You can do neat things Smalltalk like given this message send here, this abstract message send. I don't have a receiver, just the message send, what would respond to that in my entire system? And you can ask the system that because the system is running at all times. And Lisp systems, maybe not all of them, but some Lisp systems were very similar where you have a running image and you could ask the image things about its dynamic nature. And so, that took care of the things on the dynamic front. Ruby is in this gutter in between the two.

DAVID:

[Chuckles]

AVDI:

It is in the worst possible position between the two. It is impossible to statically analyze. Yes, people try but there's just a hard limit on what you can do with it. And it's very hard.

JAMES:

Come on, Avdi. That's what regex is for.

AVDI:

[Laughs] Most of it's just heuristics. It's heuristics like in this case, "I think probably you have access to the following methods, but I can never actually be sure." But at the same time, there's no image. It's very difficult to just have a running Ruby image and develop inside a running Ruby image where you can just ask the image, "Okay. Give me all the things that might respond to this message." And you have things like it's difficult to have an image that reloads things, because reloading things in Ruby is messy and often leaves stuff lying around. Or you have things like, "Oh you can't reload that class because now it has a different superclass and that's just a no-no." And so, reloading is a problem for images. A lot of things make it difficult or impossible to have just a long-running image that you develop in, in Ruby.

DAVID:

Yeah.

AVDI:

So, it's just an awful position to be in. And I've done some, I did a Ruby Tapas episode recently where I was demonstrating how I sometimes flip over to RubyMine and the RubyMine team has gone to great lengths to try to do some nice automated refactorings for Ruby. They're incredibly primitive compared to the ones that are available in Java. But they're there. And they even do a little bit of duplicate code detection and stuff like that. But it's really barely there. You have to hand-hold it a lot of the time. It doesn't always get things right. And there's just no way out, the way things are set up right now. And if you want to see an example of how things are different, check out either of those two sides. Check out the amazing refactorings that are available in IDEs for Java. I think some similar things exist for some of the static functional languages. Or checkout the stuff that you can do in Smalltalk refactor-wise. So, that's my rant.

JAMES:

I have no response to that.

[Laughter]

DAVID:

I love it. I love it. You were going down the Smalltalk thing and I'm like, "Oh dude. Ruby has ObjectSpace. We could walk all of the objects and we could say, do you respond to this? Oh except for we can't really query the arity of a method." We can't know how many arguments it's supposed to take. And then you really nailed it, which is ObjectSpace has no response for objects that are lazily loaded and the file hasn't been loaded into memory yet.

AVDI:

Yeah. It's really honestly been pushing me towards using other languages lately because I've, okay when I'm gluing stuff together, when I'm writing system automation scripts and stuff like that, I'm

gluing stuff together, a language like Ruby is fantastic. When I want to think in more domain terms, I want to think in more domain terms. I don't want to have to do all the work of refactoring, moving things around. A lot of design is moving things around. And I don't want to have to think about that. I want to be able to work at a higher level.

JAMES:

One thing I was thinking about when I was listening to you talk is Light Table. Have you seen that? The editor that…

DAVID:

Oh, yes.

JAMES:

Yeah. That would be really hard to, I don't know if it has Ruby support yet. I haven't really looked at it. But if it does, I'm sure it can't be as thorough as some of the things I've seen in it. And that's largely due to everything Avdi just said. So yeah, it's an interesting point. Ruby almost forces you down to dealing with classes at the characters in a text file level.

DAVID:

Yeah.

AVDI:

Yeah.

JAMES:

So yeah. Oh, I think you're dead on. [Chuckles] Alright.

CHUCK:

So, why else do we hate, hate, hate Ruby?

JAMES:

That's right. While we're getting it all out of our system I have a complaint. My complaint is the Ruby standard library, which I will go ahead and admit is one of my favorite and least favorite things of Ruby at the same time. The reason I say that is that I love having everything and the kitchen sink. I love being able to say I can count on something and know that I can require it and it will be there. And that lets me write Ruby programs that I just know will run and do these things and stuff like that. So, I love that aspect of it. And for the most part, the standard library includes a lot of the things I think are really, really critical with maybe a couple of exceptions like Nokogiri or something, though it does have REXML in defense. But that is the crux of it for me. The standard library I think, and this isn't really a fault of the core team or anything like that, it's that once a library is pulled into the standard library, it's slowed way down in its development cycle. It's expected that it will be very stable and not jump around a lot. And it will always be there and it will always be the same. And so, some of those libraries are great but we have much more modern versions that are leaps ahead. So, it would be great if we could rotate those in and rotate the old ones out, because obviously we don't want six xml parsers in there. Or if we could somehow keep progressing and building toward the future with the libraries that are in there. Sometimes, requiring REXML and using that on XML versus the current state of Nokogiri feels like a major step backwards. Or, WEBrick. WEBrick is awesome and cool and neat. But what about Rack? Rack is a similar. I realize they're not exactly the same thing, but how come WEBrick is in the standard library but Rack's not? Is Rack still evolving too heavily so it can't be moved in? Or whatever. But I think my problem is that they get in there and they just stagnate. And then that makes it difficult to go

forward. That's my complaint.

AVDI:

So, what's a language that handles this better?

JAMES:

So, we were talking about this actually a little bit before the show. And I think the consensus was that Java is probably the gold standard here. It has a cultivation process where things start outside the language and get a certain level of fame and then they go through an improvement, a touch-up process. And then get imported into the language proper. So, I guess by the time they get there they're still pretty good. And then I'm not super familiar. It's been so long since I've been in the Java world. I'm not familiar with how well they do going forward from there. Maybe it's just that they were so good by the time they get in that they're a lot better and it's okay to make them the standard. And I think in Ruby's defense, a lot of these things were probably imported at a younger time in Ruby's life. So, we need an XML parser. Okay, REXML's the best we got. Let's go with that, and great. And REXML doesn't require LibXML which Nokogiri does. So, they went forward with that because it was the best choice at the time. But it's not the best choice at this time anymore. And I pretty much never see code parsing XML with REXML anymore. Am I alone in that?

CHUCK:

I'm going to have to write some.

JAMES:

[Chuckles] Yeah. New goal.

DAVID:

My opinion on XML is that it's a four letter word. The second joke…

JAMES:

No, it's just three.

DAVID:

No, XML's a four letter word. The F is silent.

[Laughter]

DAVID:

And the follow-on joke is the F stands for verbose. [Chuckles]

JAMES:

Yeah. No, that's cool. I love the standard library. I can't count the number of times I've required PStore or something and just blown somebody's mind and gotten ridiculous amounts of work done. I love things like that. But I do wish we could find a way to keep it moving forward and maybe find a good process for rotating things in, rotating things out, still developing in parallel. And I think this is on the core team's radar. There's been talk of gemifying the entire standard library and then just having it be when Ruby installs, certain gems install. But then you could still easily upgrade to newer versions of that gem or whatever if you wanted to. So, it seems like that could maybe mitigate the problem a little bit of things stagnating in there and maybe free their development cycle from being tied to Ruby's development cycle. But I don't know. To me it's a great thing and a bad thing at the same time.

AVDI:

Yeah, agreed.

CHUCK:

So, I'm going to throw one out there. And my experience with this has not been super recent except for some of my coaching clients. Sometimes we run into stuff. And that is Ruby on Windows. The Windows support I have to say has gotten a whole lot better than it used to be. I started programming in Ruby professionally in 2006. And yeah, it was kind of a pain on Windows. And by kind of a pain, it mostly worked but if you had to compile C extensions or anything like that, it just really didn't work. And today, it's gotten a lot better. And so, you can use a lot of the gems that are out there with it. But for some of these you still don't have a great option for some of the things that they require. So for example, they have to rely on DLLs and stuff like that. And if it's not there, then it's not there. And there's not really anything you can do. But we do tend to keep up a little bit. It's just a little bit behind now. But I have to say, RailsInstaller and RubyInstaller have come a long way to making that possible.

JAMES:

I think one of the complaints I heard recently with Windows was that Win32 API was deprecated. Is that right? I think. You're not supposed to use it anymore. You're supposed to use Fiddle I think instead as an acceptable alternative. But I think that's significantly harder. And I may be wrong on all that. This is just what I've heard. But it does seem like the Windows solutions lag a little bit behind the Unix support.

CHUCK:

Yeah. I also remember going back and finding the mwin32 blah, blah, blah version of the gem.

AVDI:

Alright, now I often hear an objection whenever this comes up. "Well, why does it matter? Why do we need support on Windows?"

JAMES:

No, that's not a cool objection.

[Laughter]

DAVID:

It's really not, because Windows still dominates the business marketplace. There is so much, when you start talking big data, you're either talking Java or you're talking .NET.

JAMES:

Schools use Windows.

DAVID:

Schools use Windows.

CHUCK:

Well, most of personal computing is still done on Windows as well, not just business.

DAVID:

Absolutely, absolutely. I spent a couple of years writing device drivers for Windows. So, guess what operating system I had to live on the entire time I was at work?

CHUCK:

[Inaudible]

JAMES:

Yeah. Plus, I just don't think, as those of us probably who listen to this podcast, there's probably a lot of that favor things like Linux and stuff like that. But surely, we haven't liked that attitude of, "Oh we don't have to support Linux." So, we don't ever want to be doing that to any other platform.

CHUCK:

Yeah.

AVDI:

For the record, I learned Ruby primarily on Windows because that's what I had available to me at the time.

JAMES:

How did you feel it was, pain-point-wise?

AVDI:

To be honest, with the stuff that I was doing back then it wasn't really that painful. I wasn't employing a lot of gems. I was actually using it for some Windows automation. Actually, one of the cool things about Ruby is that very early on it shipped with Win32OLE module which enabled you to do things like automate Excel from Ruby which was a lot of fun.

JAMES:

I believe that's the one they recently deprecated. I may be wrong on that.

AVDI:

If you prefer to do that over using Visual Basic for it, for the stuff that I was doing, and I was doing stuff with a lot of core Ruby and YAML and stuff like that and just not a lot of stuff from the ecosystem. So, it wasn't a huge problem. And I certainly wasn't serving web pages, doing a lot of concurrency with it. I think that if I had been doing less automation tasks and more application tasks, I probably would have been frustrated faster.

CHUCK:

Yeah. I have two things that I want to bring up with this as well. One thing is that most of the people that I see at the new Rubyist workshops, they show up with a Dell or an HP machine or something like that.

DAVID:

Yup, yup.

CHUCK:

And they're running Windows. And if you can't get them set up and you can't do it quickly and easily, we lose them right there, right at the doorstep.

DAVID:

Yeah.

CHUCK:

And it’s really kind of sad.

AVDI:

Some people at this point will say, “Well, get them to load up a Linux VM.” No. CHUCK: Yeah.

JAMES:

[Laughs]

DAVID:

Right, exactly.

AVDI:

Let me try and revise that. No.

CHUCK:

Well, that’s hassle. And if you don’t do that then you wind up using one of the bash emulators. And those aren’t great too.

AVDI:

Because what you’re saying is number one you’re saying you have to bring in a Windows machine with sufficient horsepower to run a VM, which somebody might not have. And what you’re also saying it, “Oh to learn this new programming language, you have to learn an entire new operating system.”

JAMES:

[Chuckles]

DAVID:

Yes.

CHUCK:

Yeah.

AVDI:

This argument [inaudible] me out a little bit.

JAMES:

It’s so easy. [Chuckles]

DAVID:

That actually segs well into my next gripe. I can’t write a program, compile it, and give it to my mom in Ruby. In theory, it can be done. RubyMotion, I can build an iOS app. But that really just lets me release to the entire world via the app store, not give a program to a friend without giving them a temporary beta license. There’s the LLVM. There are compilers. There’s the Releasy gem that we’ve already talked about and we’ve talked about it having problems. And ultimately if you are writing in Ruby, your consumers are either people who use websites or other Rubyists. We’re not building solutions in Ruby for people outside of Ruby. That bugs me.

CHUCK:

Yeah. Mac ships with Ruby so technically you can hand something over to them and they can, if

they figure out how to run it, then you can.

DAVID:

Technically, you can.

JAMES:

As long as you target the version that's on their Mac.

DAVID:

Exactly.

JAMES:

And that version changes every single release, so you'd probably be locking yourself to a certain Mac OS and higher.

DAVID:

Right.

CHUCK:

Case in point, I had a neighbor email me and he said, "Do you know a good program that will download the list of my Twitter followers into a text file?" And I emailed him back and I said, "Not off the top of my head, but I could probably write one in 20 minutes." And then he's like, "Oh that would be cool." And I emailed him back and I said, "What kind of machine do you have?" And he emailed me back and told me, "Windows." And so I'd have to have him install Ruby on his machine in order to make it run. And yeah, it's hassle. It's hassle city to give somebody non-technical something like that to run.

JAMES:

And in that case, it's actually a one-liner with the t gem.

CHUCK:

Yeah.

JAMES:

Which is yeah, it's cool. We have all these tools. We want to give them to everybody but when, and this is true and I see this every time I teach somebody Ruby, they get to the point where they're like, "Oh how do I write a program that I can run on my computer at work?" And I'm like, "Yeah, we don't really do that. We build a web app and we hit it through the browser." [Chuckles]

CHUCK:

Yup.

JAMES:

Because that's our best choice.

DAVID:

Yeah.

CHUCK:

Well, and it's a whole added level of complexity that you don't need for something like, "Give me a

list of my Twitter followers," right?

DAVID:

Right, right. And again, there do exist tools. Why the lucky stiff wrote that weird wacky thing that was just…

CHUCK:

Shoes?

DAVID:

Not Shoes. It was a different thing. But it would take, maybe it was shoes, but it would take an entire Ruby script and the entire Ruby interpreter and it would stick them into a, he just laid them down in a binary file. And on Linux he laid them down in the ELF format. And on Windows he laid down the data segment and the code segment in an EXE. And on Mac OS X he laid down the right segments of a DMG file. And when you ran the program, it ran Ruby that it had contained with itself and then it loaded out of the data segment your script. So, this stuff is there.

JAMES:

That's kind of scary awesome. [Chuckles]

DAVID:

It's really scary. I cannot believe for a minute that that wouldn't trip most virus scanners today, especially if you're trying to write data back to the file to store it. But also on Windows, we have the one-click installer which honestly I don't know where it's at because I haven't touched Windows. I mean no disrespect to the guys that maintain it. The OCI fell out of favor for a while. It was unmaintained for a long time. And then it was maintained again. And OCI is great because it also includes Rails. So, you can give somebody a Ruby and a Rails and they can get up and they can get going.

CHUCK:

I tell people to use RailsInstaller now.

DAVID:

Oh, okay.

CHUCK:

And it works seamlessly. Even if they just need Ruby, because it puts Ruby on there and it works.

DAVID:

Right.

JAMES:

So, in a similar vein, let's talk about GUI support.

CHUCK:

Eww. [Chuckles]

JAMES:

Ruby has shipped with a GUI toolkit for forever, Ruby Tk. And my hat's off to the Ruby Tk maintainers because they have had to go above and beyond and then some to maintain things

over the years because of things like mismatches between Ruby using its own threading system and Tk expecting real operating system threads and stuff.

DAVID:

Right.

JAMES:

So, I can't even imagine how they've done so well with that. But it's generally not, when we talk GUI, Tk is nice but probably not our first thought when we think of making a good, fits in everywhere kind of GUI.

DAVID:

My programming background was years and years in Visual Basic and then Visual C++ in Windows. Then Visual C++ in Windows, so .NET and MFC and all that good stuff. And so, I naturally think, when I want to give somebody a utility to work on something, I naturally think in terms of a dialog application that opens up a dialog box, gives you some control, so you can click on them. And Ruby will let you do that by packing things using the Tk libraries. And there are people out there right now going, "What about FOX windows, what about wxWindows?" And my answer is [pbbbt].

CHUCK:

[Laughs]

DAVID:

They all stink. They're all lousy.

JAMES:

The poor transcriptionist.

DAVID:

Yeah.

JAMES:

That's all you got to feel sorry for.

DAVID:

Yup. Using wxWindows and using Tk from the programmer's side of things, basically no different. You're still playing games with packing widgets from source code. So, you have to read basically a document and then compile it in your head to try and figure out what this dialog box is going to look like. We had a thread on Parley months ago where somebody said I wish we had the Visual Basic rad tool where you could just drag dialog boxes and drag buttons on and drag list controls and then double click on things to go right to a code editor and write the handler for on click and that sort of thing. And it was funny that one person actually responded and I believe it was out of just a vehement hatred of all things Microsoft, but they actually responded with, "I don't want this tool to ever exist in Ruby." And I love you Parley members. I love all of you like my children, but whichever child you were that said that, you're wrong.

[Laughter]

JAMES:

And go to your room.

DAVID:

And go to your room. I still love you, but go to your room.

CHUCK:

Well, build it in a gem and if you don't want it, don't install it.

DAVID:

Yeah. And I don't know. I have thought long and hard about building a rad tool for Ruby that all it does is, because I can, in Developer Studio it spits out this, I can't remember what the name of the file is, but it spits out basically an XML document or a binary document that describes what the layout of the window is after you're done drawing it. And I really just don't see any reason why we couldn't build a graphical environment, drag and drag stuff into Ruby, and have it spit out this data blob. And then have a Ruby gem that reads that data blob or loads it from file and throws it over to Tk or wxWindows or whatever. Actually, I see a whole bunch of reasons why we couldn't do it. One is integrating with Tk and wxWindows and FOX or whatever window library means you need different adapters. And the other one is, again I want to build cathedrals, not make bricks. And so, I'm bemoaning the lack of bricks here. I'm not complaining, this is one of those cases where don't complain, fix it. Well, I don't want to fix it because this would be an epic project, right?

JAMES:

Those rapid development tools, they're cool any time you see them. I've even seen one in Ruby. Ryan Davis had a thing, I'm pretty sure it was zenspider, had a thing that using OmniGraffle Pro, he could use the UML stuff included with OmniGraffle Pro and UML design some classes. And then he integrated that. It had some API or something. And he integrated that where it would pop out the Ruby classes and the methods would already be in them and stuff that was said would be there. And it was neat. Not a, this is going to change how you program forever, but it was still very cool. And you see this in lots of places like Unity and stuff we've talked about, have lots of tools, Unreal Engine, where some decisions in programming don't have to be thought out from, let's start assigning some variables and write some code and blah, blah, blah. And you can start at higher levels, which is actually what Avdi was sort of talking about before, that in game stuff where you can connect certain environments just by manipulating these graphic nodes and then drop into the code when you want to change something. Or if you use something like Blender for 3D imaging or something, similar thing, graphic environment. You drag these nodules around and you create procedural textures for your 3D objects. Or we talked to Jacqui a while back at The New York Times and they have that Streamtools, which is designed for streaming information in to various systems. And they were literally using a visual drag and drop system for that, to set that up. It's amazing stuff.

DAVID:

Yeah.

JAMES:

It's like programming in the future.

DAVID:

Almost, except that it started in 1996 and we've lost it. [Laughs]

JAMES:

[Laughs] So, now it's like programming in the past.

DAVID:

Yeah.

AVDI:

I will say there's a good version of that and there's a bad version of that. And I think what some of the pain points, what some of the remembered pain might come from is the bad version of that, which manifested back in the day in tools that would write a whole lot of code for you.

JAMES:

Yes.

AVDI:

And it was usually one of two things. Either there were these huge sections of code that you weren't allowed to touch, hands off. If you touch things here things will break, and the tool would maintain that. Or, they would go a little further. They would try to do the round-tripping thing where you could modify the code and then it would be reflected back in the tool. And that was an interesting kettle of fish. And the flipside of this is tools that don't actually write code for you. So, my understanding is that most modern GUI frameworks have some sort of GUI definition language as far as I know, like Qt has a GUI builder which is really independent of any programming language. You build a UI in it and it's basically building up some sort of, I don't know, XML representation or something along those lines, something standardized. And then the various language bindings to that GUI framework, they can all load up one of these definitions and that gives you at runtime, that gives you something that you could hook your events into regardless of what language you're in. So, yeah I agree with people that have been burned by tools that tried to generate a lot of code. But there are other ways of handling it. And that might actually be an option in Ruby. I haven't really played with this. But I understand the Qt support is pretty decent these days. And building up a UI using their GUI builder and then hooking Ruby into it might be workable.

JAMES:

The prags had a short book on it one time, Qt. And I read it back then but it seems to have passed, I don't know, that phase. People didn't get into GUI development in Ruby for some reason I'm not totally clear on.

DAVID:

Well, Ruby started out as a peer to Perl almost, even more than to Python. Ruby and Python came out at the same time as noble successors to Perl.

AVDI:

Right.

DAVID:

And Perl was this text graphic language that could do anything because everything was text. And so, Perl even had the Tk bindings to give you crappy looking windows from the get go. And so, Ruby has Ruby Tk to give you crappy looking windows right from the get go. And then Python got some cleaner windowing tools and Ruby got some cleaner windowing tools. Ruby seems to have not been as interested, I guess, as an ecosystem. We as developers just haven't been as concerned. Like with Python, very clear that game developers wanted to move forward. So, Pygame and those things are a lot more robust and a lot more stable and a lot more solid. And here I'm using games as an extension of graphics and windowing support. But in Ruby, I think we, there's a few of us that really, really want it but everyone else is kind of happy either writing Rails apps or the occasional transformation script that just munges files on the disk system.

JAMES:

So, let's talk about one more before we call it a day because it's interesting. Dave, you had problems with concurrency, right?

CHUCK:

He's the only one.

DAVID:

Yeah, this is one that I mentioned in the preshow and you said you wanted to argue about it. I'm excited about this one. So yeah, I'm glad we've saved this one for last because it is the sacred cow with Ruby, the biggest number one sacred cow. Standard MRI Ruby is slow, single-threaded, and uses a lot of memory. There, I said it. It's out on the table.

JAMES:

[Laughs]

DAVID:

We can get around all of these problems.

JAMES:

No! [Chuckles]

CHUCK:

Ruby can't scale.

DAVID:

Ruby can't scale. It's true. We can get around all of these problems. James, you gave a great talk at LoneStar years ago about Ruby is not slow. It can be as fast as you want and here is why, and you showed all of the things that we had available in Ruby. And single-threading, you can go to JRuby, you can go to EventMachine. Here is my beef with this. Going to something like JRuby or EventMachine, surmounting these problems requires introducing an inordinate amount of technical hassle, like deploying. We have an entire book by Joe Kutner called 'Deploying JRuby' because if you switch to JRuby now you have to know which version of Java to use, Java 6 or Java 7. You have to know. Java tools are going to creep in. now, you're going to be making your project with Ant instead of Rake. Even knowing which version of the JDK or the JRE to install, do I install OpenJDK? Do I install the default Linux Java 7 JDK? Do I go to Oracle's website and download their JDK installer or RPM, Debian file, whatever? These technical hassles. EventMachine is a great gem for Ruby that lets us write evented Ruby. But EventMachine also has a lot of problems of its own. And more importantly, writing evented Ruby looks wildly different than writing a regular Ruby script and just adding in concurrency where you need it. It's like you have to turn your whole program on its head. And I feel like Ruby, yeah, it's slow, single-threaded, and uses a ton of resources. And surmounting these problems introduces a whole bunch of other problems that are annoying and a hassle.

CHUCK:

Can I pile on for a second? Because even if you're writing evented code…

JAMES:

No, they're all against me.

DAVID:

[Laughs]

CHUCK:

Even if you're writing evented code you're still blocked by the GIL. The threads don't go to other processors, or the cores on your processor. So yeah, it will switch threads and work on something else but only if the other thread has given up the Global Interpreter Lock so that the other one can do its thing.

JAMES:

Avdi, you want to get in a few slaps?

[Laughter]

AVDI:

I think they covered it pretty well. The concurrency story on Ruby sucks.

CHUCK:

We didn't talk about memory.

JAMES:

[Chuckles] Well, he said resources, uses an inordinate amount of resources.

CHUCK:

Yeah.

JAMES:

Alright. So, since I feel like I'm the lone defender here, I will make my best attempt at defending some of these things. But yeah, I think it's not like I can save it. Slow, I kind of disagree with in that I do believe in the past Ruby has been almost criminally slow. I really feel like they've made massive headway once YARV was integrated.

DAVID:

I have to give you that one because I got beat up regularly at my day job that Ruby was slower than PHP. And in 1.9.1 or 1.9.2, YARV is now faster than PHP. And all the PHP guys now no longer care about comparative benchmarks.

CHUCK:

[Laughs]

JAMES:

Yeah, I did feel Ruby's speed in some areas. Dave mentioned that talk. I basically went through and showed a lot of cheats to get Ruby to go faster. And there are a lot of cheats. I'll put the link to the slides into the show notes. Using things like NArrays so that you can manipulate a whole bunch of C integers in tandem and stuff, you can go crazy fast on things like graphic manipulation and stuff like that. So, there have been cheats to get around it in the past. I got to tell you, I'm feeling that need a lot less these days now that Ruby 2's really come of age. And I feel like it's pretty smoking fast on a lot of things. There have been the occasional stumbling blocks like how Ruby gems would basically file stack everything when it was trying to require stuff. And that was a big slowdown. There have definitely been bumps in the road. But as far as, the way I'm seeing speed improve, all I can say is thumbs up core team, looks awesome. Looks great to me.

AVDI:

That's definitely true. But you can't get around the fact that if you want to do something that's math-heavy, in other words just engages the processor a lot in Ruby, even though it uses native threads nowadays, MRI still is not going to parallelize that math-heavy processing because anything that's actually in native Ruby itself is going to take the GIL.

JAMES:

So, yes. That is basically Dave's second point, single-threaded. So, let's talk about that one because I think it's the biggie.

CHUCK:

Can I say something on speed really quickly?

JAMES:

Yeah, sure.

CHUCK:

Because speed is two concerns for me. One is does it tell me the answer fast enough? And I think that's what we're talking about here where the NArrays or these other tricks, or not having to use them anymore, yeah it tells me the answer fast enough. But the other thing that comes up is that it's also in some ways a measure of how much work it had to do to get the answer. And so, there are other systems out there that could take advantage of, C is an example of this, where it's just really wicked fast. And it doesn't take as much processing to get the answer because it's a little bit lower level language. And so, there's that as well where when you scale it up, you have to scale it up to handle a little bit more work.

JAMES:

Yeah, I think that's true. To some extent, that's impossible to mitigate. Calling a Ruby method will always be more expensive than calling a C function because there's so much context involved and stuff like that.

DAVID:

Yeah.

CHUCK:

Yeah. But for the most part, you could scale it. It's not terribly painful to do it. But at the same time, I just wanted to point that out, that there are processes out there that are a little bit more efficient than Ruby is.

DAVID:

Yeah. Well, and to be fair, if you're worrying about speed in an interpreted, byte compiled, garbage collected language, you are already in a state of sin.

[Laughter]

AVDI:

Eh, I don't know. There have been so much… first of all, it's not, okay it's either interpreted or it's byte compiled but not really both.

DAVID:

Right, right.

AVDI:

And there's been so much work to speed up byte compiled code these days. You've got V8. The Smalltalk people put loads and loads of effort into making that very, very fast. And a lot of that got rolled into Java. A lot of that research got rolled into Java, which is, okay granted it's a static language. But it's pretty freaking fast these days.

JAMES:

I'm surprised at all this because right now it seems like we're complaining about garbage collectors and stuff. Am I the only one that realized Ruby just got generational garbage collecting?

AVDI:

Woo!

JAMES:

We're catching up.

[Laughter]

CHUCK:

Yay.

JAMES:

We're getting faster. We have a real garbage collector. We have bitmap generational garbage collecting. So, it's friendly with copy-on-write. I feel like they're making massive strides in these areas recently.

DAVID:

Yeah.

CHUCK:

Oh, totally props to the team.

DAVID:

Absolutely.

AVDI:

I think there's an elephant in the room though, with this concurrency stuff.

JAMES:

Okay. What's that?

AVDI:

Which is that because Ruby has not had a concurrency focus for basically its entire life, nobody has been thinking about writing thread-safe or otherwise concurrency-safe code in the gem ecosystem.

DAVID:

Right.

AVDI:

And that to me is one of the biggest elephants in the room, is that there is a ton of code in the Ruby ecosystem that is not thread-safe. I don't know. It might not even be EventMachine safe.

JAMES:

Yeah, so let's talk about that a sec, because actually some of what Chuck said about EventMachine is confusing, I think. Let's address that. So, in something like Node where the entire thing is evented, period, from the bottom up, everything people write is evented-friendly. And in Ruby, that's not the case. We have lots of gems, like Avdi said, that do things that just can't be used in that kind of environment. So, you can totally use EventMachine on a single thread. That's actually the point. And to do that, you need to make sure that any libraries you use are event type approach. So, you switch which database driver you're using and stuff like that so that you get something that's evented layer, meaning that they'll take their little slices of time as they come available and return in a reasonable amount of time and not do something that just brings everything to a screeching halt.

DAVID:

Right.

JAMES:

So, I believe EventMachine can be used in situations like that. But as Avdi said, you're going to have to be careful which things you have to grab and stuff like that. It's a good point, that we haven't been aware of the threading thing in the past so we haven't done a good job prepping for that. So, getting back to the single-threaded thing which seems to be our primary complaint. MRI is no longer single-threaded but because it evokes a GIL to make C extensions work smoothly, and again that had to be done because otherwise they would have broken almost every single C extension out there because we haven't prepared like Avdi said, it's effectively still single-threaded because you can only be in one at the same time. That said, I see people get this wrong all the time about Ruby and it makes me mad. [Chuckles]

JAMES:

So, I'm going to go up on my soapbox here.

DAVID:

James smash!

JAMES:

Yeah, smash. So, I've seen people say forever that that means you can't do two things in Ruby at once. And oh, that just makes me so mad. So, first of all, at least back when I started Ruby, in Ruby 1.8.2-ish or something like that, from that time forward, Ruby has always been super smart about switching thread context or releasing the GIL when you do something that's going to pause. So, make a slow system call, I/O is the biggest one, you have always been able to in a loop make a bunch of Ruby threads, even when they were green threads, have all of them fetch some webpage, and that would work in parallel, all the way back to Ruby 1.8.2. And the reason is that most of that time was spent waiting on [inaudible] from the other side. And when Ruby sees a thread is about to go into waiting mode, it would suspend it, put it to sleep, jump context to the other thread. So, they would all do that. If you did it in a loop and you fetch ten websites, you would have some big amount of time if you did it in a loop and put them all on threads, even back on Ruby 1.8.2. You would have noticed a huge drop in time.

DAVID:

Yes.

JAMES:

That has worked as long as I've been in Ruby. So, Avdi has the right example. Anything that's processor-intensive is where you're going to get into trouble because you can't pause the thread and then [inaudible] on the CPU. Well, you can but then work won't be done. Ruby has another solution for that, and that's fork. That's been, in my opinion, the core team's idea of how we should go forward. Obviously, there are plusses and minuses about that. But you've been able to fork another process. Two processes will be scheduled by the operating system, not by the Ruby interpreter. Therefore at that point you can easily chew on multiple CPUs or whatever, cores even. So, I would say that's the Ruby answer. It's definitely not a perfect answer. And I think this was primarily Dave's complaint of one, we start forking processes, you have to worry about a lot of things like how are they going to communicate to each other? How are they going to pass data back and forth? What are you going to do to reap the zombie processes so that they don't build up? All of these things.

DAVID:

Or the fact that each process is 64 megabytes in size.

JAMES:

Only if you're forking Rails. That's not…

DAVID:

Oh, fair point, fair point.

JAMES:

But yeah, I agree.

CHUCK:

Fork Rails.

AVDI:

It's still a huge amount of resources compared to shared resource threads.

DAVID:

Right.

JAMES:

Compared just with threads, sure.

AVDI:

And here's the thing about that. It's not just strictly a resource consideration. You have to structure your entire architecture around that decision. You have to figure out, hopefully fairly early on, where is a good place to chop, to divide the system into separate processes? And you have to make it high enough level that you don't later find yourself wishing that you could start up 10,000 of these fork processes. And I compare this in my mind to something like the Erlang process model where an application is basically built of many, many Erlang processes. And they're scheduled independently, and they're very, very, very lightweight. And you don't have to think about, oh, does it makes sense to fork this off as a process or should I try to combine it into another process because we don't want to have too many processes? And of course, Erlang takes it to the next level where you can even decide, should this be even part of the same system basically, or do we want to push it out off somewhere else? But the process model forces you to make really, really big

architectural decisions. And that also comes out in Ruby doesn't have a really great way of communicating between processes either, at least not baked in, not like this is the accepted way.

DAVID:

Right.

AVDI:

and so, you also have to spend a lot of time thinking about, okay now that I've decided I've got a good seam where I'm going to split it into a few different processes, now how do I make them talk? Again, in a language like Erlang, all of that's already wired up for you. You're just sending message around always.

JAMES:

Yeah, that's a good point. I think when I finally got into Linux and learning how the process model works there, I really internalized a lot of that and went down a lot of those rabbit holes. To me, it's totally natural to pull a pipe, do a fork, close one end on both sides and communicate across that. I do recognize it takes a little code and then you have to decide what protocol you're talking over that socket or pipe or whatever you're using. So, to me that's always felt not too unnatural, kind of the Unix-y way. But I recognize that you're right and things like Erlang where they just have operators for sending messages to other processes and stuff. I'm not sure I totally agree with, even with a language that does it better, even JRuby which gets rid of the GIL, then you have to think about different things in my opinion that potentially two threads could be in this code at the same time. Is that okay? And I feel like that makes me make different structural changes and decisions. So, I don't know. I think planning for concurrency to me always seems like planning for concurrency. Then it's something you have to do and think about the problems that are going to be involved and it's complicated. I will give you that I agree that Erlang is probably the lightest, it's so ready for you, we're trying to make this as painless as humanly possible thing.

DAVID:

Yeah.

JAMES:

There are other problems to fork by the way. I didn't cover them all, Windows being one of them. I don't believe fork is implemented in Ruby on Windows.

AVDI:

Nope.

JAMES:

Which is probably one of the reasons that people aren't super big on it. But man, it's amazing [inaudible].

DAVID:

Well, that's not Ruby's fault though, right? The Win32 or, okay that shows my age doesn't it, but the Windows operating system doesn't have a fork, does it?

JAMES:

Not in the Unix sense of the word. You can obviously start up other processes and you can emulate some behaviors. So, hats off to the Perl team on this one. Perl fork works on Windows and it does it through emulation using Pthreads and them arranging for the two processes [inaudible]. It's the most epic hack I think I've ever seen. And I'm sure it was hard. But it does, it is possible to

build things like that, I think, on Windows. But Ruby doesn't go that far. But I would say that traditionally processes have been the Ruby way to achieve real concurrency. And it's not without its faults. I'm not saying that it is. But you can make some stuff go really fast. Resources is maybe the last point Dave had that we should touch on a little. I'm not sure how much of that to actually pin on Ruby. Ruby I guess uses a fair bit of resources. I think a Ruby process is fairly small if you just fire it up and chunk it out. I seem to remember, 2 meg sticks in my head. Maybe it's 4, which I guess is big-ish but not gargantuan. Rails is different. If you fire up Rails and then you check in the process table, it's much, much bigger. But they've made lots of progress there. I know in the hash, they made it ordered in Ruby 1.9 I think. And so, that required hash to get bigger. But they made so many memory improvements in the move to the new architecture that hash was awash or got smaller. So, it didn't really hurt it. So, I think they've done, again made some pretty amazing improvements there. And I think it has a lot to do with how you use it. I think people tend to think Ruby leaks memory and things like that, which I think Ruby Under the Microscope, RUM again covered so well, just showing how the way we use it and passing blocks around can caused this context to be saved on the heap, that it looks like it sticks around for a long time and that's what makes you think things like that, I think.

DAVID:

With every language, there are things that you can learn and go, okay I'm going to close the door on this and never think about it again. And then there are other things that you have to, it's the minefield. You have to learn, oh I have to remember I can't take a hash in from the Rails controller that is the URL parameters that came from user land, I can't call symbolized keys on that because users can put in any keys they want on that string and I've just created a symbol that won't ever go away. Although I hear a rumor now that symbols can be garbage collected.

JAMES:

No, they can't.

DAVID:

They can't? Oh.

JAMES:

It's been talked about but they don't.

DAVID:

Was that an April Fool's Day hoax, then?

JAMES:

[Laughs] It's been talked about, making them garbage collectable. But there are lots of… the VM uses symbols internally for its symbol table. So, there are issues with that. I'm pretty sure they're still not garbage collected. That said, I would encourage us in this discussion to separate Ruby and Rails as much as possible.

DAVID:

Yeah.

JAMES:

Rails does a lot of things. You just mentioned having a hash that doesn't know the difference between symbols and keys. While that may seem really cool from a programming standpoint, it has performance implications that I don't think were properly considered. Rails is just heavyweight with all it includes. And if you have your Rails process and you at some point do a moderately big query

forcing a lot of objects into Ruby's memory, that's going to balloon up your process and then it's never going to go back down. So, that's I think where people come from when they see these leaks and ballooning processes and stuff like that.

DAVID:

Yup.

JAMES:

And Ruby has given us some tools to combat things like that, even recently with the lazy operators and stuff that let us chew over data in a lazy fashion and things like that. So, I don't know. I feel like massive thread inroads have been made here. The threading is definitely an issue and still is. But processes can be an answer sometimes.

DAVID:

Yeah. I guess we're nearing the point to wrap up. And I want to take all of the kvetching that I've done in this episode and turn it into the biggest compliment that I can give Ruby, which is I see all the warts in Ruby and I see all the things that we've complained about, and Ruby is still hands down orders of magnitude my favorite language over every other language. No other language makes me happy when I program in it.

JAMES:

I am just thrilled nobody brought up m17. I thought for sure I'd have to defend that with my dying breath. [Chuckles]

AVDI:

Well said, Dave.

CHUCK:

Yup.

JAMES:

Yeah, I think it's cool. They've done lots of awesome things. They've made lots of progress. And these things that we're complaining about are largely first world problems, right?

DAVID:

Yeah.

CHUCK:

Yeah.

DAVID:

Yeah, they totally are.

CHUCK:

Yeah, so the moral of the story is go use Python, Java, Node, Erlang.

JAMES:

Erlang. Yup, that's it. We're switching.

DAVID:

And then come back. Yeah, we're switching to everything else.

CHUCK:

Yeah.

[Laughter]

AVDI:

Smalltalk.

CHUCK:

Smalltalk.

DAVID:

Smalltalk.

AVDI:

Haskell.

CHUCK:

Yeah. Mash them all up and you get everything that you ever wanted and everything you didn't want.

JAMES:

From here on we will be the non-Ruby Rogues.

CHUCK:

There you go.

DAVID:

Yup, the Everything-But-Ruby Rogues. [Chuckles]

CHUCK:

The Polyglot, something else.

JAMES:

Polyglot Rogues.

DAVID:

Well, Polyglot minus one.

CHUCK:

Yeah.

DAVID:

[Laughs]

JAMES:

But I tell you what, if Avdi does one episode on Smalltalk, that's it. I'm going.

AVDI:

[Laughs]

CHUCK:

Do Elixir instead.

JAMES:

Yeah. That's close, right?

AVDI:

Have that one already.

JAMES:

You did?

CHUCK:

Alright, cool. Well, should we do the picks?

JAMES:

Let's do it.

CHUCK:

Alright, David I'm going to make you start.

DAVID:

Okay, alright. So, last week we talked video games with Megan and I apologize that I haven't had any really seriously in-depth picks. And I have no intention of changing that today. So, last week we talked about games. I want to start by picking Megan's game, Glass Bottom Games, Jones On Fire. This is a silly, stupid, it's Canabalt with a firefighter where you rescue kitties. It's just a run and jump game. That's all it is. And she talks in the last episode, she talks about Juice It or Lose It. And just the little twitches that they add to the game make this game. You can play it, how to say this delicately, you can play it in the room that people prefer that I not talk about me using my phone while I'm in that room.

[Laughter]

DAVID:

It echoes in there. People hate it when I call. But in the time it takes you to use that room, the library if you will, you can play a game of Jones On Fire. And it's fun. And you rescue some kitties and it's just fun. It is incredibly simple. You'll look at it and you'll go, "I can't believe this game is going to suck me in." But it absolutely is. It's absolutely worth the 99 cents it is. There's a light version that you can play for free. Don't even waste your time. Just give Megan the dollar and get the full game. You'll get your money's worth out of it. The second game had me crying laughing. I installed it last night. It's called Inflation RPG. Have you ever played an RPG where you grind? Where you just try to get more, one more hit-point, try to get one more level, just 50,000 more experience points until I get one more point of charisma or whatever. Inflation RPG is a little mobile game that lampoons grinding. So, you start off and you walk around in this field. And the first thing you fight is a chicken. And you gain so much experience after killing the chicken that you are now 22nd level.

[Laughter]

DAVID:

And so, you have to immediately leave the home area because now there's nothing worth fighting. You've just added, your attack score has gone from 100 to 300 after one battle. So, you move to the next area and you fight something. And my first game of Inflation RPG, I ended. You're limited to 25 battles. And I ended the game with, I was 749th level. And then it's over at that point. And it's just hilarious.

JAMES:

That's awesome.

DAVID:

To just see, they use the JRPG style, Japanese Anime style. When you win a battle you just see this level up, level up, level up, level up, level up, over your character. I've gained 70 levels in one battle at one point. It's just absolutely insane. It's hilarious. It's fun. It's well-made. And it's silly and stupid. You can put it down and walk away from it. There is a little bit of longevity to it which is that you can buy equipment. And anything you purchase you get to keep forever so that your next character starts with things that boost attributes and that sort of thing. So, you can go even further in your next game. And so, that's Inflation RPG. I know it's out on Android. I don't know what other platforms it's on. I know Jones on Fire is on, in Megan's own word, on every mobile device ever. Inflation RPG, I'm not sure on. But I recommend them both. They are lots of fun. And that's my picks.

CHUCK:

Avdi, what are your picks?

AVDI:

I'm just going to pick a bunch of blog posts and talks that I've liked recently. So, let's see, doing down the list. Oh, there's a talk called 'OOP: You're Doing It Completely Wrong' about how you're doing object-oriented programming wrong by Kevin Berridge. I enjoyed that one a lot. There is, oh speaking of Smalltalk Noel Rappin did a talk at MountainWest RubyConf about why you should really check out Smalltalk. And he uses the Pharo Smalltalk in some live demonstrations. And incidentally, Pharo Smalltalk just dropped their 3.0 release. So, that was a really enjoyable talk. And it gives you a nice quick glimpse into what you're missing out on when you don't write code within an image. And let's see. There's an article that I got a huge kick out of recently called 'Programming Sucks'. And that's exactly what it's about. And it's just one of the better rants that I've read recently. So, I think I'll just leave it at that, some talks and articles.

CHUCK:

Awesome. James, what are your picks?

JAMES:

I've got two. For a programming related one, I didn't know that there were other ways to monitor a file other than tail -f, but it turns out there is. You can use less +F. And the advantage of doing this is it does the tail thing. And then when you want to stop it to look at what's going by and examine that, you can. You can hit Ctrl+C.

AVDI:

Cool.

JAMES:

And it takes you into the normal less interface. So then, you can move around or search using that. It's pretty cool. So, it's like tail but you can pause it and go into a less environment and poke around

and then restart it if you need to. It's pretty cool. And then, I just saw this for fun tweeted the other day, 33 Amazingly Useful Websites. And I was surprised by how many of them I wasn't familiar with. And there are all kinds of things in here. Sites that will tell you what to eat or what to watch or how everybody's flipping their name upside down on Twitter, this site will do that for you. You can type in some text and it will give you the characters flipped upside down. Just lots of random things like that. I think one of my favorites is 'Can I Stream It?' was on there, which you can go in and, oh I want to watch movie X, Y, Z and you just type it in there. And it's like, oh you can stream it from Netflix or Amazon or Hulu or whatever. It tells you where you can go find it on the internet. Lots of cool sites in there and then some that are just silly, [inaudible] fun, but cool stuff. That's it.

CHUCK:

Awesome. I've only got one pick today. It is 'Smart Money, Smart Kids' by Dave Ramsey and Rachel Cruze. And it's a terrific book about teaching your kids how to handle money. And you can start them at age three and it gives you guidelines all the way up. So, just a terrific book, really enjoying it. So, that's my pick. We are coming up on our episode with Rebecca Wirfs-Brock. We're going to be talking about 'Object Design'. And so, if you haven't had a chance to pick up the book, go get it. And we'll catch you all next week.