UPDATE (5/16/2016, 22:05): The official transcription, which is what was projected, as you know, looked different (was cleaner) from what came out here. This was an experiment having both streams going at once and there were definitely some bugs on the Google doc end. The “official” transcription can be found here, so any edits henceforth should be made to the following copy. I tried, during breaks, to copy over changes people were making on this transcript as it was happening live. Thanks to all who participated to help!!

https://docs.google.com/document/d/15QaEghNd1-MH8RqZhf54tsuXA9aaMTS0WbS9GL_2EIE/edit?usp=sharing

STKRURBL STKRURBLT STKRURBLTS give a shout out to the first project I ever contributed to, which is Dreamwidth which is a forbidding-up Livejournal. They’re really super friendly community that basically held my hand through my first patch and they’re really open to beginners. In conclusion open source good but we can make it better if we really tried. Thank you. **Welcome to !!Con 2015**

May 16, 2015

[(ROUGH) LIVE TRANSCRIPT]

***Disclaimer: The translation software running on the tablet to produce this transcript still has bugs and hence there will be more errors on here as compared to the projected copy.**

THANKS FOR HELPING CORRECT, EVERYONE!! - <3 S

MAGGIE: Hello! We need help with something. Has anybody set up a proxy server on a Mac laptop before?

>> A proxy server on a Mac laptop?

MAGGIE: I can try to figure it out but I would love some expert assistance!

>> All right. Everyone should sit down. We're about to start. Right?

>> Whoo! Yeah!
[Applause]
JULIA: all right. Welcome to !!Con, everyone. I'm so excited to see you all here.

>> Introduce yourself. JULIA: could I get the organizer humans to come here, actually, so that you can actually know who we are. So I'm Julie. This is Lindsey. Erty. Yes?

>> A little farther away. To the side. JULIA: over here? Wow, this is ten times better. If you're speaking, this is how you need to do your microphone. This should function as a tutorial. Erty is over there, and he's doing our AV. There are other people who are hiding. Maggie is hiding.

>> Actually I need to figure out how to close the door!

LINDSEY: That's Maggie.

JULIA: Alex is downstairs.

>> Where's Leo? JULIA: downstairs. We're going to introduce ourselves later. I have some administrative notes.

LINDSEY: Do you need me?

JULIA: what?

>> Do you need me?

JULIA: I do but you should sit at the front. There's Wi-Fi information there. The restrooms are somewhat complicated to locate but they're in the back.

>> Off the pantry!

JULIA: off the pantry, and there's, like, a gender-neutral restroom. So, some things about this conference there is a quiet room in the back if you get tired talking to people or want and work by yourself.

>> It's room number 825.
JULIA: the way you get to it is you go all the way to the right, and there's a sign that says, "Quiet room." There is also allowed room which is over there. The loud room is for the unconferencing sessions that we're going to have and the idea is you can talk about the exciting things that you're talking about with all the people that you meet, so you can make an unconferencing session about the thing that there's a talk about and then you're so excited about! And now, you need to really go talk about it! And you can go talk about it over there, past the food room where you have been all enjoying the croissant I would like to thank the MAGNET space for having us here on relatively short notice and I would ask you to just please respect the space and when we leave, you should take stuff with us. Yeah, this is, like, a huge deal and this is such a great space. So I want to talk about one of my favorite things about the Recurse Center? So one of the things that I love is talking about programming, surprise. And one thing that I, like, before I went to the Recurse Center, I was like, I can go talk to people about programming. And I'll say crappy things and things will suck and when I went there, I was, like, wow I can be in a place where we can talk about programming all the time and everything is constantly awesome! Which was kind of like shocking. And I also, like, !!Con want to go a similar place where we can talk about everything all the time so that things are automatically awesome. Things aren't always awesome but things are always better than normal. I'm exaggerating. But one of the ways that they do this, and we have to ways that they're so effective is that they have these amazing social rules which I love, and I'm going to tell you about them just to not be a jerk myself, but also, the having this space has been a really delightful experience for me. So the social rules are, and we're going to be doing them here. And it's going to be the best. No "actually"s. If someone says something that's totally correct except for one minor thing, it's not okay to say, "Actually it was in nineteen eighty two, and not nineteen eighty three,

because it was December and things that are normally not helpful. And another one is no feigning surprise. One of the things that that was helpful to me was that 24%s includes actual surprise that if someone doesn't know something that's basic, it's helpful to not fake being surprised. If you're like, I don't know what quicksort is, you can be like yeah, sure. And just not being surprised is a way better experience. And oh, no, everyone is shocked, I must be really dumb because no. People don't know things. I don't know lots of things. It's the greatest.

>> Jewel, can you speak a little more slowly? JULIA: yeah, I can speak a little more slowly, maybe. Another one is no backseat driving which is a little more -- so the way this one works is, if, like, two people are having a conversation, it's better to, like, engage in their conversation than to, like, lob corrections or advice across the room. And, um, my favorite, which is no subtle sexism, or, like, other "isms" which is about -- so there are, like, comments which are obviously, like, racist or sexist, which most of us generally don't make. And this rule is about, like, more subtle comments. So you might say, like, oh, it's so easy, you could teach, I could teach my mom to do it, which you might say with, like, an attitude of love and respect towards your mom who's an awesome neuroscientists who's not that great with computers and you don't mean for that to be a sexist statement and someone else might interpret that for reasons that are obvious. So, like, the thing about this rule which is interesting to me is that it's about not doing things that you might not realize at the time are a problem, right? And so the way that we deal with that is by, like, someone can either say to you, "Hey, that was kind of subtly sexist." Or say to one of us, or one of the organizers -- is Alex here? So you can come and talk to me or Alex if something happens we can talk about it. And then if someone tells you that they've done something that they didn't feel was great, the best way to deal with it is to be like, "sorry" and then apologizing and moving on and not arguing about it. I've experienced that being a great approach. And then if you have, like, a question about something, you can talk to me or Alex. So all of that is my favorite. And the best. I have two more things to say. One is that, have you seen this?

>> Whoo!

[Applause]

we have captioning for all the talks. This is Stan! The best! I'm sorry to talking so fast.

STAN (CAPTIONER): you're excused.

[Applause]

JULIA: this is basically wizardary at work. And I'm glad that we have it. And the last thing that I want to say is that there isn't any Q&A in any of our talks which is great which means that you can go to talk to the person in the unconferencing space at the great, like so now I'd like to hand this over to Lindsey.

LINDSEY: Thanks. All right. Can you hear me?

>> Yes.

LINDSEY: Am I being amplified?

>> No. What -- sorry.

>> Not as much.

AV DUDE: Hi, everyone.

LINDSEY: Thank you. So what I should be doing is introducing our first keynote speaker. I'm slightly concerned because he just got here and I don't know if we've done an AV check. So maybe, we can -- I'm delighted to introduce our first keynote speaker, of !!Con 2015, by the way, my name is Lindsey Kuper, I'm one of the organizers. And so our keynote speaker for today is David Nolen and I'll give you a little bit of background about this. Some of you may know, especially those of you who were at !!Con last year -- how many of you went to !!Con last year, by the way? A fair number. Cool. So some of you may know -- we originally wanted to have David as our keynote speaker last year, and we were really excited about this, and then at the last minute it turned out that he had a conflict and couldn't make it. So we ended up having Michael Bernstein instead and that was fantastic and everyone loved it. But we were still sad that we couldn't have David and so we invited him back. And I'm so pleased that we are now having David speaking. And he's going to be telling us about inventing the future that we want to live in. And I'm especially excited about this because !!Con is a conference that we invented because it was the conference we wanted to attend. And so, David is going to be telling us about the future that we want to live in and how we can invent it and I'm extremely excited. And with that, I guess we'll turn it over to David.

DAVID: Thanks.
[Applause]

LINDSEY: In just a moment.

DAVID: We can try this one, too… I wasn't sure…

 LINDSEY: While we're waiting -- am I being amplified now?

>> Yes.

 LINDSEY: While we're waiting for this to come on. Oh, it's almost done. Cool.

DAVID: Ready?

 LINDSEY: Feel free to attach this to yourself in whatever way you could.

>> Maybe if that lamp was off, we would see the screen better. JULIA: oh, yeah. I wonder…

DAVID: Check, check, check. Hello, hello. I'll project. Okay. Hi, I'm David Nolen.

>> Hi.

[Applause]

Okay. And so this talk is called, "Invent the future you want to live in" so this is really kind of more of a personal narrative and it's not really about inventing the future for everybody. You can't invent the future for everybody, but perhaps it's worth inventing the future for a small community that you care about. So, again this is kind of a personal narrative but hopefully, you get some ideas or some inspirations, or new perspectives on each and how they use computers and how things come about. So the very first thing that I'm going to start with, though, may not even seem having to do anything with computers at all. Raise your hand if you've ever read -- no -- no -- raise your hand if you've ever heard of Moby Dick. And that's amazing. Nearly every single person in this room raised his hand. When her man Melville in the 1750's, it was universally paneled. So how can a book that people hated -- I can be in a room in New York and everyone raises their hand. And it's been translated into many languages. How does that come to be? Well, one person believed that he didn't want there to exist a future in which people didn't appreciate this book, this is Carl Van Doren, I think in

nineteen twenty he wrote an essay where he said, “This is one of Melville’s greatest works.” And this came back people reassessing it, and people re-evaluating where it sat in American literature. Now people don’t question it. It’s one of the greatest American novels. In fact, Moby, Dick people could say, that it was written too soon. Sometimes you have a big idea and people don’t have -- really, the means to understand it. And that may very well be a cultural that’s at play. So this is one idea in my talk. I’m starting out with a couple of things that frame my talk. A couple of things that’s at the center of my talk is probably one of my first loves which is drawing. I don’t do as much drawing as I used to, but one thing that I do now, or that I appreciate, or that I just involved in, that drawing is a very tactile thing. Everybody that gets into drawing loves the fact that you get your hand on the paper, and you really get to see the result and the other really great thing about drawing is, if it’s something that you want to draw well, it’s beautiful, over time, you can see yourself improve. So that’s something that’s fantastic about drawing that I think a lot of things that involve technology don’t really have this sense yet.

I’m a musician as well. And instruments are important. I think people know this intuitively, although, I think sometimes our standard for whether the instruments in terms of the technology we pick or use, whether or not we’re more discerning. As a musician I don’t want to play this. I really want to play something like this, right? Instruments matter a lot. It’s not that -- the beautiful thing about a musician and you have a musical sense, you can play anything. But if you ask anyone who works as a living as a musician there’s a deeper appreciation for well designed instruments. But it’s not just about the instruments, right? Often people confuse that you can get somewhere just by having a great instrument. Music is a social phenomenon, right? It doesn’t matter if it’s just you playing your thing. There’s a whole element of communication at play. So, this is an image of Sun Ra who’s a fantastic free jazz artist. And some of you may be like, “Well, I hate free jazz.” And that’s great. Again I think there’s an obsession in the tech community that this thing needs to scale and everybody needs to use this thing. And I think it’s ridiculous. Free jazz is a beautiful artform and not everybody likes it, and that’s okay. But the people who understood it as artists, we have to create an audience here, that means finding collaborators who believe in your music, making recordings, doing wild performances so that people are blown away. These are all important things to consider in trying to sort of evangelize your idea. So those are the big framing ideas for my talk. Now, let’s get into sort of my personal evolution with computing and you’ll see how this ties back to the earlier slides. So my -- so the very first computer that I ever used was an Apple IIGS. My father bought that in nineteen eighty seven, it was a huge investment. A great little machine. I really loved this machine. Raise your hand -- did anybody else here have an

Apple IIGS? There's like, two, three? Not so many of you but it was a really great machine. And it was relatively fast. I mean, the really great thing about it was the high resolution display but there was all this awesome software that was mind-blowing for a kid. So Fantavision. So flash, it basically morphs vectors. And so, Fantavision had this in '80s hardware. It was awesome. If you have haven't seen Fantavision, check it out. And the other thing that I loved was where in the world was Carmen San Diego. I was obsessed with all these beautiful pixel graphics, I don't know who did this, it was amazing. So for my brother and I, it was a window into a whole universe. In this machine was an entire reality. So as kids, computing was so cool. It seemed like you could do anything. Eventually my father said well, if you really like computers why don't you learn how to program the thing. So this sort of got me, my brother was ten, I was nine. And he started messing around with Basic and this goes back to my slide about drawing. The amazing thing about Apple IIGS was… okay. Whoops, sorry. One second. So if you held the let set button down on a Apple IIGS, what would happen is, you would be immediately dropped into a Basic prompt, right? So when you boot the machine, you would hold down the reset button and you would immediately be able to start programming, right? And so for a kid, it's like, this is like, a piece of paper. My computer's a piece of paper! I can boot it and I can start writing code. Right, so I can go…

```
10 PRINT "HELLO !!CON";
20 GOTO 10;…
```

[Applause]

That's totally awesome. This is how computing should be, right? My brother and I would waste hours, hours, hours, it was so simple. You could set statements. You could change your program. You could write it down if you wanted to. It was so immediate. There was immediacy here. So for me the future of programming that I wanted to live in always involves sort of deep appreciation for immediacy. Okay. So… okay. So I ended up -- I was a teenager and I was, like,, "Well, I love computers." But I didn't really want to be a video game programmer and at the time I didn't know that much about the history of computing, or all this other stuff and so I was, like, well, I sort of catered to the liberal arts side of me and I studied film and I actually have no background in computer science. Besides using them, I never went to school for computer science. And so I spent sometime exploring film. I actually got into experimental film which is not what most of the people in my program was excited about, but this was this amazing world. An underappreciated artform. Quite a few interesting experimental filmmakers in the U.S., but I ended up being involved in something called Cinematexus, that

was a collection of people who said, we don't want to live in a world where experimental cinema is not celebrated and people could come all over the place, all of these criminals, filmmakers would come together. And it would be like !!Con for experimental film. It was awesome. Communities really matter, even if it's something again, maybe a lot of people don't appreciate. That is okay. I got into film, but I realized -- I was excited about experimental film and there's no money in experimental film. It's something that someone do out of love. Not to make money there's no money in it. I loved watching it, but I didn't want to do it. But I got excited. And what got me excited again about computers after my stint in film was because of John Meta (phonetic) who was a well known artist and designer and he was doing great things in the '90s where he started combining computational ideas with a deep aesthetic sense. This sort of kickstarted a whole world of people who were fascinated with combining creativity and computing. And so I got inspired by this and this led me to want to know. I was like, okay, I'm excited about computers again. So Meta went to MIT, and I have to read the book that Meta did. And I didn't know anything about Lisp.

My background was Basic, Pascal, C++. But this book was a shock. Everything that I thought I knew about computers was wrong. I didn't know anything at all. And that's a great feeling. When you find something that says, you thought you knew what computing is about. And it says, you don't know anything at all. And they barely know what it is. And they're showing this stuff. And I distinctly remember opening up this book of code and thinking, "This looks like abstract poetry. This does not look like a computer program." And so I got so excited and I said, I gotta understand this stuff. And at the end, I was so blown away. Because I was, like, this seems like this is so cool with computers but I look,s looked around the industry, and no one was using this scheme. No one was using Lisp. How could something that was so great get to little real world use? So this is me, the seed is planted in my brain. So maybe we could change that. I moved to New York City. I got a graduate degree in new media, and I ended up getting jobs, in various forms of computer programming and while I was do, I did quite a bit of Objective C. For those of you who haven't done Objective C, it's like someone trashed SmallTalk with C. And it's probably in some ways, a little bit antiquated now, but really what was awesome was that OSX was really just a reboot of NeXT. And what was really great about NeXT was that they were trying to get commodity on -- that the Altos were, you need the stuff to work on commodity hardware. So next was bringing the SmallTalk idea.

And if you've never used Objective C and interface builder, it is quite powerful. I think there are tools that surpass it. But this was futuristic stuff. They were taking ideas that were taken less than a decade earlier as Xerox and using them as an industry. What was interesting, Doom, one of the most famous games of all time, the entire Doom editor was built on a nexus

machine, but that's important again, right, the immediacy of the system that NeXT gave developers, it was an important property. So after school, I ended up doing Javascript, this was, like, 2005. Raise your hand if you do any Javascript. Wow that's a lot of people. Almost as many people that heard of Moby Dick. This was in the dark days of Javascript, though. We had to support IE6. Alert debugging was the state of the art. And it really was, like, wow I can't believe that it was so bad. This is like the antithesis of small talk of immediacy and visibility. You could Internet Explorer 6 and you would have no information. This is what you would see. This is it. Now what? How do you debug this program? anyways I think many of you experienced things like this in computer. And it should be very familiar to you. A lot of our systems don't work quite as well as we like. So I did that for you know, four years, and after that, I love Javascript but I gotta, like, take a break,, so in the summer of 2008, I took a three month sabbatical and I said, I'm going to go back to the Lisp thing. I think hack -- there was a lot of enthusiasm for Lisp, and it seemed like a lot of new developers were excited about the potential of Lisp. And so I was, like, I did, you know, the sick pea stuff, and I was like, what was the state of Lisp today. Common Lisp is really cool, but the hard thing was that it was really hard to set up and the community was, I would say, like, I mean, it's -- it's actually gotten a little bit better but the community wasn't taking it seriously. Communicating the benefits of the system and making it sort of easy to get set up with it. So as awesome as it was, it was hard to get going. And then I chanced upon this website, and it had his weird Lisp and was for the JBM. I'm going to download this JAR, and then I had a repo immediately. So I could download it and I could run it. And it was interesting because I was already interested with processing. And processing runs in the JBM. It was a fun, way to experience computer programming. So if you know Lisp, you can see that oh if I take a graphic programming environment, and I add a Lisp to it, I can recover live programming. So, like, live-coding the system. It's not hard if you have late binding the way that Lisp, SmallTalk, and as well. This is verbatim from structured programs. And they say, this is a powerful strategy of synthesis, wishful thinking. I think this is just wanting, and wanting it better, and thinking about it, and even acting on it. So a lot of things started coming together. I feel like I was interested in Lisp and I was able to use Clojure, and take advantage of the JBM. And be part of the ecosystem that people were already invested in, but I could also use all of the stuff that I love from this book at MIT. And this blew me away when Fred Fickter (phonetic) did his talk, what was fascinating about this talk was that I honestly thought that nobody was going to be able to convince people that small talk or the Lisp way of doing things was ever going to come back. People sort of lost that ideal as a goal, and Fred did a great job of saying, this is a huge idea here, and this is an old idea and I'm going update so that it's palatable to

an audience. Old amazing idea, it may seem radical, it's actually not. People know about it. And then he updates it so that people don't forget. I think that's very important. So me, I actually do lots of things but one thing that's been really rewarding and really, really fun is that as a Javascript developer I got involved in Clojurescript because Clojurescript was a dialect that allowed me to Javascript. And again, this allows me to interface with a much broader world for people that write Javascript and produce Javascript and Clojure script is just a way for us to get these nice Lisp semantics back into our development, into our workflow. So to wrap it up. So it took a long time. So I started poking around with Clojure in 2008. Clojure's now eight years old. So I think it actually takes a long time. You may have a wishful thought and it take take you long time to get to where you want to be. But I would say this was in the back of my mind back in 2003. This was my Lisp editor machine. This is an editor here that can understand Lisp syntax and the fact that what was happening with other browsers, you can view the list, and step through it. And this is so cool, right? I had this thought that one day I'll be able to program the computer the way that I want to program the computer. I never lost site of the, sort of, joy and immediacy that I liked for my Basic programming. But that's really all I had. And I'm probably out of time.

>> We can take a couple minutes

LINDSEY: We're not doing Q&A though.

DAVID: Okay. Good. Thanks. Clojure

LINDSEY: All right. So I just told David that we're not doing Q&A, however, in lieu of Q&A we have lots of breaks and time. And so we have a 15 minute break now in which you can ask him all the questions during Q&A had we had it. So with that, we're going to break for 15 minutes there are restrooms back that way, next to the kitchen. And we'll resume in 15 minutes with the first session, which is called, "Traveling swiftly through time."

MAGGIE: While you're taking a break in addition to getting water, using the bathroom and stretching and stuff, if you would like to, you can take a look at your unconference sheets, they're where you got your name tags this morning. There are three unconference sessions that are one hour long where you can say what you're interested in talking to each other about. I've put up an example. And the example I have no idea what it would feel like, but if someone wants to lead an unconference example I will come. Because I'm the conference architect and you! So they're right behind the glass doors. See you in 15 minutes, otherwise.

[Short Break]

MAGGIE: Hello? Seats? Hey, how are y'all? So while you're getting seated, there's a few more notes that we just wanted to make clear. When Julie gave her intro and her opening remarks, she talked a lot about social rules and social rules are great but they're also my favorite thing. Okay? They're probably a lot of ours' favorite things but the reason we've adopted them and the reason just Julie went to such an extent to talk about them is because we think this is a useful way to think of code of conduct. A code of conduct isn't necessarily the most useful active information. It's basically, don't be a jerk, don't break the law, and that's not Bailey the best framework of thinking about it. So the social rule is a great way to observe the conduct amongst each other. And also to emphasize, come to any of the organizers about the code of conduct, or any code of conduct violations, or social rule violations that you would like to talk to someone about, and Julie is Alex are the best people to talk to, but any of us are also great to talk to. So that's that. The second thing is, so how are we going to actually do these sessions? There are four people in each session, and they've all prepared a ten minute talk. Is ten true?

>> Yeah, ten.

MAGGIE: They've all prepared a ten minute talk. And we're not doing Q&A like we did for David as well but instead we're going to have a five minute break in between for the next speaker to set up, so you can talk to your neighbor saying, "That was really great! Who knew check summing worked that way?" So without further ado I'm going to introduce the first set of speakers. The title for this speaker is, "Traveling swiftly through time." And we're going to hear first from Ashley Blewer and her title is, "Don't know about you, but I'm feeling SHA-2, And then Josh will talk to us about GDBs,, and four exclamation points. And speakers, if you would like to be seen on the recording, you have to know to be standing on this side of the lectern. So if you want to hide for personal reasons… I don't know how to deal with that.

ASHLEY: So I'll pin it somewhere.

>> Hold it to the side of your mouth. So you're not breathing.

ASHLEY: I'll breathe but just not audiblly. No magic here. Okay. How do I do this multitask? This needs to come over here. All right. Yeah, and just give me -- if I see a thumbs-up that means "volume up" in the back. And maybe volume down and I won't be offended or delighted either way, I'll just know

what to do. So hello. I gave this talk about a month ago and I managed to do 20 slides and 20 lyrics in five minutes and so for this talk, I was shooting for 40 slides and in ten minutes but I'm at 40 for each. So I'll be moving quickly but this time I'm telling you, I'm telling you, side note: That unless you're like me, and you love Taylor Swift and checksums you'll at least be 50% confused. It'll be fun. It'll like Bill Nye the Science Guy. Or Beakman's World. So nice to meet you, I'll show you incredible things by using Taylor Swift to explain cryptographic hashing functions. She's nice. This talk is, I don't know about you but I'm feeling like SHA-2. And I wasn't kidding about the amount of Taylor Swift lyrics. Uh… checksumming. It's difficult but it's real. What do people mean when they're like, bro, are you salting that hash? It's magic madness, heaven, sin, but I think we can get to a common baseline of understanding fairly easily. Taylor Swift helps with that. This is the golden age. And this is the golden age of GIFs in slides, too. So I don't want to know everything about you. It's pretty much a string of nonsense. It's beautiful, it's wonderful, don't you ever change. So as long as the string is the same as it is, when you check up on it it's good. Except for malicious hackers, which is haters are going to hate. Next chapter, that's a thinly veiled reference. But first I want to talk about a concepts at me I want talk about whether this is a heart or an invisible cheeseburger. Talk to me afterwards. I don't know. But anyway, checksums are used to prevent a man in the middle attack. And I've tried this before and it works fine. So you can see in crypto passwords and other sensitive information can pass through a network without that information being compromised. I talk to my friends, talk to me. We broke up, you're a jerk, and I don't want to see you. Anyway. The way this is successful though is information is passed through channels but never directly so it doesn't matter if hackers are standing by you, waiting at your backdoor, your data will be safe because the plaintext information happens safely, and the obfuscation happens transparently. So I mentioned salting. Salt is a random string that's linked to a password hash that's linked again for an extra layer of protection just like you would be protected from bad exes by your friends. And you can encrypt that information but without each other, that information cannot be unencrypted. So you can shake it off and not worry about your passwords being stolen. A collision attack would be when an attempt to find two ash believe them and when that salt certificate reads green, you want to believe it. But you want to look out for security breaks all the time like heartbreakers. Security-wise, these things don't last forever and always. And you'll hear, this format is broken like in SHA one. It can be corrupted. If she l if the algorithm can be decrypted by computers. This is slow and treacherous, under or the path is -- time and money to cause that break to happen. If an algorithm would take a modern computer multiple computers years and years to decrypt, it's still pretty secure but it's

still, like, my Achilles heel, another thinly veiled reference. It follows a sort of Moore's Law pattern, that it gets faster and cheaper every day. Information is sofa moving target. These walls will fall down so when you're shipping this into production or you're investigating what might be a good fit Friday, maybe you were thinking they were forever ever, and I used to say never say never, but these things get broken every day. And I'm just, this is exhausting, you know. Girl, I know. I know. Anyway, we're going to switch it up and talk about some algorithms that we use out in the wild. CRC. Don't make me laugh. Stands for cycle redundancy check. This is serious stuff. We were both young when I first saw you. I say that because CRC was developed by W Wesley Peterson back in nineteen ninety one. And two thirty two, still growing up now. It's a lot older and a lot more basic of a process, and so obviously there's a lot more limitations here, but it is used for file format verification. The tresco files are an example of this. They use CRC32 as a check mechanism. And then we have SHA. It was developed by the standards of information technology as a information processing standard. But it gets a little complicated because there are different numbers and they mean different things. It's a roller coaster kind of rush. The lower, less secure versions like SHA zero and one, -- so get used to the SHA1 so that you have unique commit messages. More complex algorithms are used in other things. HTTPS used to use SHA1, but that's been broken for a while, so how could you not know about baby,-y-y. That algorithm's basically the same, but there are different versions of it and it determines if you're legitimately on the website that you intend to be on. And then we have MD5. We see MD5a lot in archives which is my field. So you look like my next mistake. That's at least how I feel because sometimes it works.

It also uses MD5 for fixity and integrity. I think I know when you belong. MD five stands for message digest algorithm five because computer scientists are great at naming things, by Ronald in 1991 so replace the old MB standard, guess what that stands for. It's also broken. But I work in the archives field for the most part. Security isn't an issue. And so cave sums aren't broken from a security standpoint. Nothing's not going to change not for me and you, not because I knew how much I had to lose. File integrity, file authentication. I love this gif. Are they what they used to be. Are they what they say they are? Isn't this easy? You could also use checksums as a structure for quicker access for validation like when you said you needed space, what? But if you're thinking of using checksums for security you just gotta go in headfirst, fearless but the core principle still stands that regardless if you're working in information security or if you're working in digital preservation you want assurance the file is not changed in any way, even in ways that seem invisible, at least that's what people say -- eh, eh. So that's why this stuff is so important. So from a digital preservation stance, you really want to back up, baby, back up. But for security, things

are more like it’s 2:00 a.m. and I’m cursing your name and you spend more time dreaming instead of sleeping, or maybe weary of people who are doing so. So we’re not out of the woods yet. I don’t have time to get into the details and the problems with checksums but you can summarize by saying that it’s, “Trouble, trouble, trouble.” Or maybe it’s like trying to solve a crossword puzzle and there’s no right answer. So anything in cryptography is incredibly complex mathematically, and it should be left to experts. And it’s complicated stuff. Is this in my head? I don’t know what to think but since you’re already out in the wild, just know that your battle's in your hands now. And that is the end.
[Applause]
MAGGIE: Five minutes! Talk amongst yourselves! hey! You know, it’s funny that I’m louder without it, but I feel like if I were to bellow into this, it might hurt our ears, but maybe we’ll get quieter faster. Here’s Kamal. And Kamal’s going to talk about what the clock.

KAMAL: Hello, am I being amplified? My name is Kamal and I’m here to talk about what the clock. I was with my parents. And I thought I wanted to contribute to something open sourceish and I’m really interested in the Rust language and I have been following around for a while, and I found this list of libraries that hadn’t been implemented yet, and I found this robust date-time library. And I thought, maybe I could do that.

[AUDIENCE LAUGHS UPROARIOUSLY]

In my defense I don’t jump into things without thinking about them a little bit. So I went and read some of the code literature. In literary, people from the Java world -- I’m seeing little clapping hands and nodding -- Joda Time, or JSR-310, or java.time, I think the author’s name is Steven Colbern, and I think he wrote about this topic. This topic is incredibly confusing. And yeah. So the number of libraries I’ve implemented stands at zero, but reading up on this stuff took me down a crazy rabbit hole and I wanted to share some of that with you. So here’s a really fun thing. This happened four weeks ago. This happened on the 20th of April. They announced in Egypt, DST that year was canceled. And DST was going to be the first of May and the final announcement and the decree by the President of the 24th of April. And yesterday someone told me that if you say time zones and people cringe, they’re programmers. This should make you cringe extra double. It’s terrible. So this seems a bit crazy but that was actually the fourth update this year to the time zone database. The fourth update. And we’re what? We’re in May? Every year, there are typically nine or ten updates to the time zone database. So if you have a system that’s showing times to humans somewhere in the world, which most of us do, it’s important that your time zones are

up-to-date. So just as an example requests, this is what the diff looks like, Egypt's minister's cabinet just announced, dot, dot, dot that it will cancel DST for 2015. And above the Emacs Lisp code, it was added 20 days earlier, so it had to be added really fast again at me so it can be frustrating. Reading the time zone database files is really interesting. There are people who are really, really meticulous. Very, very, very meticulous about having everything just right. There are people who call little hamlets in the Northwest Territories every March and November to see whether or not they observe daylight savings time every year. There's some towns somewhere in northern Canada the federal parts are in daylight savings time, and some are not, and they celebrated new year's twice, one hour apart. And I actually going on there and looking at the zone info. But this brings us to how long is a day. And I'm going to specifically be talking about a UTC day. I'm not going to talk about the -- I was going to talk about the civil day which is the kind of day that we look at on our clocks which can be twenty four hours, or twenty-three, or twenty five because of daylight savings time, or twenty two, or twenty six because of weirder daylight savings time, or twenty four, or twenty four hours, and 20 minutes because of still weirder daylight savings time but I'm talking about the UTC day. And the UTC day is the modern Greenwich time. And we know that it has 86400 seconds, except for this day, which has 86401-- it has thus far not had 86,399, but they've not ruled that out. But this is what UTC looks like tonight. So tonight we're going to get get 23:59:57, 23:59:58, 23:59:59, 00:00:00. And then we get beyond the seventeen and we know this. And UTC in June is going to look like this, 23:59:58, 23:59:59, 23:59:60, and then it will be the next day 00:00:00.

There's this whole extra second here and that's the leap second. And this is inserted into UTC to keep the UTC time, which is, read off of atomic clocks in-sync with our, like, you know, the stupid planet and its rotation and the sun and stuff. So if they didn't insert these, then we'd slowly drift. and it continues. Time doesn't stop at midnight. So each seconds are inserted or deleted -- thus far, none have been deleted, to keep UTC close to solar time. Solar time being the time the sun is in the same place in the sky. Since they were introduced in nineteen seventy two, there have been twenty five leap seconds introduced, the next was at the end of June, there'll be some fear mongering articles on the news, and some Internet company will screw this up, and there'll be some embarrassment, guaranteed. But so -- a nicer way to think about this is if you go back to nineteen seventy two and when they first started having leap seconds and you said I'm going to start a countdown to the year 2000, if you didn't take into account leap seconds, you would have been celebrating a whole twenty two seconds earlier than your friends if you didn't count leap seconds. So it adds up. Over the next second, this June not a big deal, but over the next ten, hundred years, you would be a few years. And that would be bad. Astronomers.

So what we use typically is Posix, and what Posix does it steps back compared to the leap second. And so if we compare June second, and Posix is our 1.4 billion-ish seconds since nineteen seventy and then we have the next second -- and then we get the next second which is still the same but actually it's not still the same. If we look in between the seconds, what's actually happening is, it is progressing 200.5 but then it just goes backwards. Yay, Posix. And you know, this could be, you say, what's the big deal, the deal is, you can't translate from Posix time to UTC, there's a second here that's ambiguous. This is what it looks like when you have a stop clock and you're watching your Posix clock, as time progresses linearly along the bottom there's this sudden jump backwards. And that's not the best if you have systems that rely on timestamps for correctness and that happens, there's going to be some incorrectness. This is the stuff to look out for we don't have this ambiguousness but again there's this whole second where nothing's happening. And so there's this whole big issue. So every time the system is asked for time of day, it asks for it a minimal increment, a microsecond or a nanosecond. And then when it gets past the nanosecond, it waits for the next second that the new time catches up and says, "Okay, now we can continue." And so it's like, okay… I have no answers for you here. I have no answers. And this is okay except that now during the second, the times don't match between different systems. And so there's this crypto person called DJB who happens to use without time seconds. Google uses the previous 24 hours that if you think about one in 86,400 adjustment that you spread all over the day. There's actually going to be a vote this November whether the leap second remains with us. So it's going to be decided in November. Except they were going to have this vote in 2012 and they postponed it for three years, and so they may decide the fate in November and this is again, DJB and he writes, the main obstacle is Posix, fortunately the Posix rules are so outrageously dumb, for example they require that twenty-one hundred be asleep year, contradicting the Gregorian calendar that no self-respecting engineer would obey them. And just when you thought that you've solved this, and DJB says, well, Einstein, nineteen 12, nineteen fifteen, general relativity, the worst, the Hafele–Keating experiment confirmed that if you fly around the world, you can gain up to

KIRAN: Hi. I'm Kiran Kiran and I'll be talking about how Wi-Fi has gotten faster over the last 20 years. When the first -- when the IAAA put out the first Wi-Fi standard in nineteen eighty seven, it had a max upper speed of 20 megabits per second. The newest amendment is a new transmission of three.three nine gigabits. Per second. That's a seventeen fold increase in the last twenty years. We talk a lot about Moore's Law, and its two times transitory density. So it's a grab bag of magic tricks that we've thrown together to make this happen. So this is how Wi-Fi works. We get data in, and

we get estimated data out. So the first thing we do is compression. This is how JZITS works and then you run this through channeling algorithms and then this throws in redundancy and error correcting. It's kind of thinking about uncompression where you take your data and think about it three times over. But it's not quite. It's more like complicated checksums that sends the in the end. So once you have the bits you're sending,, you take your bits and you, like, mess with the sine wave to turn it into information you can send. And then you just mess it up completely. You send it over the air which is a totally noisy channel. There are people talking all the time. There's microwaves which are the worst. And then you repeat the whole process in reverse. You demodulate, you decode, you decompress. So I'm going to concentrate on demodulation and compression. Which is the least programmy part of this but this is where we've seen most increases happen. So let's talk about why this is hard. So there's a theorem that says, for a given noise contamination there's a maximum rate of data transmission that you can send error free. So this, the bandwidth, the channel capacity in bits depends on your analogue bandwidth. The number of frequencies that you can talk over and your signal to noise ratio which is the signal that you're transmitting compared to the background noise and so on. So we've been squeezing out both variables. So getting more on with bandwidth. If you have more space, you can encode more information in. And so let's do a quick overview of modulation. You may have heard of AM, or FM radio, amplitude modulation, or frequency modulation. The method Wi-Fi uses, you adjust your phase in amplitude in order to encode a different number of bits in. So RF engineers represent this as a. So the way that we've been doing is to make that information denser. You can have different values for each of these things. So we saw a 8-piece constellation here.

This is 16 and we've gone up to 256. That's super density. That's 16 bits per symbol that you're sending out in a four nanosecond per second space and it's crazy that we can actually detect those changes at such a small level. So this again, noise. Noise is hard. And noise knocks out the symbols that you see into different places and so the separation between these symbols gives you your noise tolerance and if you can't tell what the symbol originally was, you can't really decode it. So what happens -- we use something called link adaptation to do this. If you see that your channel's especially noisy, you can back up and make your modulation less dense which is one of the reasons that as you back away from your router, things get slower and slower and slower, you go down to a lower modulation. so these conditions are highly dependent on your signal to noise ratio. So another way that you can optimize more bandwidth is by having more bandwidth. So you take a 20 megahertz channel and you can split that into 52 streams that are talking independently at once. The newest draft protocol uses one hundred eighty six megahertz channels so you can send things eight times as quickly.

So your transmission speeds increase almost linearly with your bandwidth because you can just split it up into more channels. The issue here is interference. You can think about slicing off apiece of bandwidth for yourself as getting a table at a restaurant. There's a lot of stuff going on at two.four gigaherz. So move up into the spectrum to five gigahertz where not many people use it, or up to 60 gigahertz, which is the wild west, there's no one up there, partially because there's no one with it. So we might see seeing faster Internet but you can't use it, like, across your five-storehouse which I wish I had. So we'll talk about getting more out of your signal to noise ratio, as we were talking before, your signal to noise ratio on how tightly you can modulate things. So basically if you're speaking louder, you can speak faster and we can't fix background noise. There is noise in the air and our signal power is limited by the FCC at one watt maximum. So you can't really, like, keep sending more information out and turning up the power partially because you might mess up with people in your area, and partially because of the FCC. And so we'll talk about tricky tricks about increasing our power in localized points instead of the average power.

One of these is beamforming. It's kind of like throwing stones in a pond and seeing this warped checkerboard pattern go out with different power levels. And so I'm going to use quantum mechanics to explain this because you may have seen the double slit bit. It's silly relying on that as a tool that people understand… but, you can use this interference to double the intensity at points. So you can see here that there are -- this is the same intensity at both points. On both diagrams but there are some points where you have double the power and some points where you have no power. And so you end up with a watt on average. And this is directional. By changing the phase difference between the two antennas, you can send it out to particular points. This is kind of like how your ears hear direction where one of the transmitters, the receiver sends out a synchronization pulse, and the transmitter listens on multiple antennas and shoots a signal back with a phase offset. They were kind of like yeah, people will figure out how to do feedback for beamforming. But now it's something that's baked wild and the has it is crazy. And this only really works in some very small cases but this is one way where you can send multiple streams out at the same frequency, using the same channel and still be able to resolve it at the end. So this is what exists now in the protocols we have now. In the future, we might see more crazy things. One of the most exciting things to me is multiuser MIMO, which is multiuser using many antennas. This is kind of like the beam forming that we saw before where instead of -- but here instead of talking to one station at a time, you can talk to multiple stations by relying on beam forming to increase intensity at some points and null at others. So I can have four conversations with four people at once by directing who exactly I'm talking to, and expecting it to null out elsewhere, which is insane. We're

going from being able to use one antenna at one time to one person, so utilizing the channel to talk to many, many people. And the other thing that I was alluding to earlier was having more bandwidth. If you go up to 60 gigabits which was what was proposed you can still carve out to two gigabits and still be okay, and that's still a huge bandwidth increase both in the analogue and digital sense. So, thanks.

[Applause]

JOSH: Excellent. Okay. I am first going to remind you how terrible the debugging experience can be and then I'm going to introduce you to a future of rainbows and unicorns and you'll love it. Let's definitely so when you're using a source code debugger. Often you have two main concerns, either you interrupt the program too late and you miss the parts of it that made it important for resolving the problem, in which case you have to restart, or you stop it too early and then you waste time getting to the parts that do matter. And if you end up making a mistake, you still have to restart all over again. This is unpleasant. This is even worse if your error isn't actually consistent because then if you restart there's no guarantee that you'll actually catch the error again. It's a really bad cycle. So let's talk about the specter of non-determinism in programs. So what that means is that your program relies on factors besides the initial state of the program and any initial inputs provided to it in order to decide how it behaves. So here are two simple example programs that both, initialize a random number generator and then take the first random number and then decide what to do based on it. The one on the left is deterministic. That means that if you provide the same seed number on the command line to it, every time it will behave exactly the same way. The one on the right initializes the random number generator based on the current time and that means that if you run the program over and over, it'll have a different seed every time. That is non-deterministic. So other causes include things like user input or the way the kernel schedules your tasks, or, you know, external signals, things like this. So I want to tell you about a tool called rr which stands for "record and replay." And what that does it actually records the non-deterministic behavior of your program, it notices what is non-deterministic. So let's see what that means in practice. So I have a very simple web server which, occasionally, when you refresh it, instead of showing you a directory listing that actually has a bad request and that's no good. But it's also mysterious. So if you look at the output, you notice that it sometimes prints out a useful message saying, "There was a problem." So let's go and take a look at this replay of the recording that I took of this in the past. And this can be hooked up to any GDB front end. So I'm going to do it in emacs. So first I'll set the breakpoint that printed out the error.

>> Bigger font?

JOSH: I don't think I can. It's a video. So we're sitting on the break point. Right there. So this happens after the actual error has occurred. So that doesn't necessarily help us because we want to know what caused it. So let's go backwards in time. We're now at the previous if-statement. so it's based on a variable called error flag. So let's take a look at where that's stored in memory and we get a pointer out of it. And we'll then set a watchpoint on this pointer, which means, "We'll get a breakpoint whenever that value's changed." Now, let's run the program backwards and see what happens. And we go to the line that changes the zero to one. And that's really cool. So now we can move around in time in program state and we can go forwards to see what happens afterwards or we can go backwards again and we can keep going backwards. If we want to dig into what happened before us, we want to step into functions, so we can go back in time, and a bit more. But if we decide we're not actually interested in that, we can reverse finish and go to the color.

So from here, I think we'll `step back a little bit more to see what the problem was. Not the if-statement, that's behavior here ends up happening through the system calls the kernel which means that the program can intercept the system call and notice the resulting output when recording and then when it's replaying, when it has the system call again, ignore it and then just replace the output you would get with the recorded one and then you use something called hardware performance counters to get a specific point in time that you can associate with these events and in this manner, you then have a trace of programming behavior over time that you can just replay over and over, which is neat. So why is rr especially in particular is that it has special low overhead compared to other tools. It's nothing new. Like, this isn't an idea that's been around since the earlier 2000's, and there's a bunch of tools out there. This one is a new one that's aiming for production quality. We use it at FireFox regularly. It's of new use to us if it can't deal with giant programs and also we don't want any kernel modules, we don't want any hardware. There is any stock thirty two or 60 four bit Linux system that can do this with it. And it's all open source, so it's pretty fun. So what else can you do with this? You can travel back in time. It's fun. But imagine if all your testing infrastructure ran in rr. If you have less than two times overhead, often that means one.two times. That means a ten minute suite will now take two times. Every time you notice a flaky test that occasionally fails, you can be just like oh I'll just grab the recording from another server and log in there and take a look whenever I have the time. That's pretty nice. and also this works in optimize builds in that if you're ever debugging something in GDB, and this object that I'm trying to inspect

optimizes out, you can figure out where that that value came more about how it works. Thank you.

MAGGIE: How amazing were all of these?

>> Super amazing!

MAGGIE: So I wanted to announce a slight schedule change. We're going to merge the unconference session that we were going to do before lunch with the one after lunch. Lunch is going to start at 1:30. It is 1:19 and we're going to meet back here in this room to listen to three more delightful talks at 2:30 so you're basically released until 2:30. But lunch looks really good. So you should hang out and have lunch. and if you're one of the speakers in the next session -- you know who you are -- go to that corner of the room and talk to Alex. Alex, wave.

ALEX: Hi. Actually you should talk to Leo. He's the chair. Leo, go to the corner, also.

MAGGIE: Or just go to the corner and we'll figure out who to talk to. But we'll run you through setup beforehand.

[Applause]

>> Yay!

ERTY: Hello, everybody. Let's make our way back towards the main area. It is time. Time has come. Please, make your way back towards the stage.

Awesome. Can everybody hear me.

>> So just quick announcement. These bin cameras are not being-extremelied streamed so I will be editing this footage later. So if I point this camera at you and you're wearing a red or orange lanyard, don't be afraid of the big cameras.. This camera is being live-streamed, so don't walk in front of that if you don't want to be on air. Thank you.

ERTY: Cool. So a couple brief announcements. First of all there is water in the back. There's We've found a black notebook, blue, with a green pen on it?

>> Basically the person was wandering around looking for their notebook.

ERTY: Great. So we found the owner. Awesome. So this section is called analytic philosophizing. I'll let you figure out what that means. I have not named these sections. a water fountain and a sink with, like, a water tap and there are cups. So if you want water that's a good place to go. There's also, the bathrooms are also over there. So brief kind of organizational note while this so we have three talks in this sect, "Contributing to open source should be like roller derby." By Libb called "MissingNo byy Horacek. And the second talk ision. The first one is called Cool,comes up. Oh and one more thing.

>> So open source should be like roller derby. I don't mean this. This is the nineteen seventys derby. In the '70s it was a no holds background, race around the track. Problem no problem, just grab their hair and take them down. A little too fast, knock them in the head. Fortunately a lot -- this version of the sport kind of died out when people stopped being paid to play roller derby. And you can still play it in a few places but generally, skaters aren't interested because it seems really scary and dangerous. And I would say, yeah, there are analogues in open source but no, I'm talking about flat-track. Modern flat-track derby. And it's a real spot. It's fast-paced, full contact, and it has rules to help people stay safe and it also has a really awesome collaborative culture. Derby culture's awesome. Like open source, it's mostly volunteers. It's by the skaters for the skaters. there's no, like, Crowe directing everything. It's always very inclusive. Open source projects can learn from on you roller derby encourages beginners, values teamwork and empowers participants. Roller derby is really great at supporting beginners, and that's something that I started to think about as a beginner to open source. Like open source, too. Pairing is really powerful. In this gif here you can pretend that the girl in black is a really tricky bug. And the skaters in white are managing to isolate here and eliminate her. Unfortunately in derby, that bug's just going to come back around and

re-emerge but unfortunately, that happens in real life, too. Also when you're working with others, you can push your teammates to write clear and maintainable code. But the thing to remember is, in derby, when we're pushing people through the pack like . What does that mean? We shall see.

IGOR: Hi, everyone. My name is Igor and I'm g talk about , specifically, MissingNo, which looks like this. So has everything to do with programming, and here's why. This is my day job. Whoops. There we go. But first of all, what even is, we've practiced pushing each other safely so that you don't just knock someb

ERTY: Okay. So up next is Igor, with MissingNo, my favorite Pokemon over. And you can think about that in open source as not being like,"Wow your code sucks." By instead being like, "Here's how you can improve it." Sorry, I like stretching metaphors so much. You can also give each momentum when you're feeling burnt-out. Roller derby also has teamwork off the track. Skaters share the responsibilities of organizing the league, doing the paperwork and stuff like that that doesn't involve hitting people. But it also means that the bus factor is really high. You can have one of your skaters get pregnant and it's still okay.

There's also a lot of mentorship in roller derby. Individual skaters, and at least in my league were assigned a derby mom. And so if they had any questions about what their role was in the league or if they had any issues, then they had a single person that they could go to and they didn't have to feel nervous about talking to them, but regional lesion also cool for open source to have one-on-one mentoring not just for contributors but also for maintainers. When people talk about why they play roller derby and why they love roller derby, they often talk about how empowering it is. People play roller derby who never felt comfortable playing a sport in their lives. One thing that you hear over and over again is that the reason they feel empowered is that they believe able to do things that they never thought they could do. So here ? So is a super violent video game, it's a dystopia where humans trap , and trap them in Pokeballs for their own amusement. Pok can only say their own name. One of the cutest and probably the most popular is Pikachu which has given us academy award winning lines like "Pikachu," "pikipi." The first generation ran on the Nintendo Game Boy console which was my first computer. Which sounds like this when you start the game. Give me a second. Can you hear that?

>> That's the best sound ever.

IGOR: Let me just turn that off. There we go. So at the beginning of the game, you must choose between one of these three , Bulbasaur, Charmander, and Squirtle which is regarded as the most important decision of your life. It's a generational thing. So there's one hundred fifty one in the first iteration of the game but there's this spectacular glitch that allows you to encounter that doesn't exist. So you go to the old man who explains the , after the tutorial sequence, you fly to Cinnebr Island, and soon enough, you'll encounter MissingNo, which is this strange . You're not going to believe this. The Nintendo is a computer, it has a CPU and memory, well, not that much memory. It's only eight K and so when you start the tutorial sequence, the name of the player is temporarily replaced with the string, "Old man." And for this, the player's name has to be backed up into memory somewhere, and so it's stored normally in the place where you can find the in grass. So this data is not cleared so after you finished the tutorial sequence, it stays there, and this is known as the "old man glitch." So next you fly to kin kin Island, which has no grass tiles, so the name stays in memory, and there's a glitch called the left-facing shore tile glitch which makes the east shore tiles behave like grass. so normally surfing in this area, you would encounter the that you would find in the grass of the previous area that you visited but because of the old man glitch, that information's actually been replaced by the player's name, which means that the name, or the game will interpret your name as the index numbers of the levels of the that you can encounter, which means that depending on the name that you picked at the beginning of the game, you'll be able to find MissingNo and other strange . There are other different types of that you can encounter this way. There's MissingNo and there's M. On examining the dumped ROM, there's a table of , the table that stores all of the in the game, so in addition to the one hundred fifty one that are officially part of the game, you have thirty nine extra in there that all have the name "MissingNo." and incidentally, MissingNo stands for "missing number." So these were supposed to be actual but they were then deleted or just replaced with blank data. So they're really just placeholder values that you wouldn't be able to get normally. So the player's name will index into these tables and hit one of these forgotten entries, right? But what's even more interesting is that it's also possible to index outside of the table. So you can overflow and get way more glitched-up and one example of that is M, which looks like MissingNo but behaves very differently. So I mean, the first thing is the name, which looks interesting. But when you trade this to the Yellow Edition, it will actually transform to a called 3trainerpokedollar. A nice name. But the best part of this glitch, though is that it will actually -- so if you overflow the table it will set the encounter bit in a memory area that stores the user's items because right at the table of the table of are the user's items which means that every time you see one of these M , you

will actually get one hundred twenty eight of a particular item, which means you can duplicate very rare items, which is pretty useful in the game. So that's MissingNo. The second glitch I want to share is glitch city so there's a special area in the game known as the Safari Zone, and upon entering this area there's auto step counter that starts and after 500 steps, you get teleported back to the entrance of the zone. However, flying outside of the zone will not reset the counter which means that it will keep running, so after 500 steps, even if you're no longer in the Safari Zone, it will attempt to warp to the warp location four, which is contextual. So the Safari Zone, that location is the entrance. But for other areas of the map, that can be anything, right? So there are different values for this, and it turns out that a lot of cities don't actually have a warp location four, which means that when attempting to warp from one of those cities, you will -- it will execute some code that will transform the current city into glitch city and just mess up the entire map and rearrange the tiles. But you can actually still walk around in it, which is pretty sweet.

Now, the collision map is completely different to the tiles. And if you -- if you look closely you see that there's just numbers lying around on the floor, which I find most profound.

[Laughter] so if you actually walk into, if you collide into something and you can't actually see the things that you're colliding with, then you will get stuck and you can't move anymore. So you can do at that point is pause the game, at which point all of your surroundings will turn into water and then you'll start swimming around but you're not going to get anywhere. So the only way to actually escape is to use a flying ability to fly out. Also, in the Yellow Edition of the game, you have this Pikachu that follows you around everywhere. And in Glitch City, it simply turns around and walks away.

[Laughter]

So that's Glitch City. Now in the first generation of the games was released as two editions, red and blue, and they're mostly the same, but there's certain that you can only find in either one exclusively which is really only just a way to get you to buy both of them. So to "catch them all" you would have to trade between the games by connecting two Game Girls using a link cable. So this was my first distributed system, two-node cluster. And so during the trading process, it's possible to disconnect the link table which cancels the transaction. But with good timing, it's possible to disconnect it right after one side has acknowledged. Resulting in the trade succeeding on one side but not on the other. So this way you get to actually clone the and keep it on both sides. So this was my first network partition. It resulted in losing rights. So this is an example of a fundamental problem in instrumental

consensus known as the two generals problem where you have two generals who want to attack a city but they need to know when to do so. And so because it's impossible to know when one side has received an acknowledgement, you have this constant regress of acknowledging acknowledgement, not reaching the consensus status. You might say cool story, bro, why should I care? So what is the significance of this? And I would say, well, we depend on technology, right? If we want to own our devices and if we want to be creators, we need to understand how they work. And taking things apart is just a really good way of doing that. And of learning about the inner workings and limitations of the computing machines that we use every single day. And you can subvert the system which is great. So in closing I would like to thank these amazing sources that I have documented all of these glitches in way too much detail. And thanks to all of you for listening.

[Applause]

ERTY: Okay. So the last one of this series is quine + tar = Quinine. Peter Boothe.

PETER: Thanks. So if I clip it here, am I still coming through?

>> Yup

PETER: Excellent. So okay. In my talk proposal, I wrote a lot of checks. And so I first gotta tell you what I promised, and then I gotta do what I promised, and then at the end, hopefully we can all agree. Quine populist tar = Quinine. Quines are self-printing programs. And every program can be made to print out its own source code. And storage has become exceedingly cheap. So we've introduced Quinine, a library that uses fixed points, analytic philosophy and tape storage tools to make analytic programs that carry source code themselves. It's available in my repo in GitHub under Quinine. I promised to talk about quines. I promised to talk about storage being cheap. Fixed points, analytic philosophy and all this other stuff. so let's see if I can do this. Quinine named after philosopher, Willard Van Orman Quinine. There is really good for analytic philosophers because they use logic to represent the world, which we know in computer science kind of works. And quines allow them to talk about -- and since cognition, a large part of cognition is thinking about thinking, it would be great if we could have logic that represented itself without ever using the word "this." Right, because this is problematic. Which "this" are you referring to? Oh gosh. So quines are a way of doing that because they are things that reproduce themselves. So, they are fixed points in the compile and run function, that is, they are things when you put into the compile and run function, you get out exactly what you've put in. So it's a fun challenge, actually to write a

program that prints out on its own source code. Usually the quotation marks are the tricky part depending on which language you're working in, but it's a fun challenge and honestly the first time I did it, I probably have seen examples before, but I still had to reinvent it, because I don't remember the examples. But to spoil the challenge a little bit, but not to spoil it too much. So in English, you want to say the following quotation twice, the second time in quotes. Notice that at no point, do I say "this" these are all backreferences here. They're all forward references. This is embodied in this program which is actually valid Python and will print out itself which is kind of neat. And let's make sure that it does.

I totally thought I had -- there we go. So, I have this program, pyquine.py. It's exactly what you saw up there. So if I run it, I should get myself out, right? So let me clear the screen and we will cat pyquine.py, so that's just looking at the file and then we're going to run it. And it looks pretty much exactly the same. We started off with the quotes, we started -- it works, hurray! But let's, rather than using your eyes which lie to us, let's use technology. And so we will run pyquine.py and we will pipe the output through Python. Pipe that output. And we get it back TPW-PBL so it really is… and we can actually pipe this through diff and we will look at the difference between dash, which is a fancy name meaning standard input and we will look at the difference between standard input and pyquine.py, and there is no difference. So it really does reproduce itself. It really is a fixed point in the analytic cycle. Let's do some bad hacks. Every program can be quinified. Said my theoretical computer science teacher a long time ago. You can care about this, but mostly I just wanted this slide so I can tell you how to pronounce kleene. We never know how to pronounce it, is it clean star?"Clain star?" we never know how to pronounce his name. It turns out, it's "clean-y. " Now we all know. It's disconnect in all languages, according to his son. So let's talk about cheater quines. Things that are kind of quines, but you don't feel good about having them be quines, maybe. So for languages which print out their final value. They evaluate to something and then they print it out automatically. All values are quines, right, because the whole program's just that one value. So some variants of scheme print out their final value. So the string "hello" or the symbol hello, or nil, these are quines in their languages but you don't feel good about that. It doesn't feel like you did the tricky thing. Similarly for compiled languages, I said it was the fixed point in the compile and run cycle. Well, let's hijack the compile part and put error messages in and we'll just keep putting the same error message in until we get the same error message out that people put in, and then that's a quine! That's fun. So in -- in a clang on OSX, this is a quine. These undefined symbols for architecture. You don't feel good about it but it kind of counts. And now in Python because I did something in Python, this is a cheater quine because Python indents its error messages which means

that the first thing that the Python sees are the error messages. So no indentation, error.

So it's a quine. And for interpreted languages you can cheat because interpreted languages often carry their source code with them so you can just read the file that's in rv zero, and that doesn't feel particularly great because you're using operating system stuff. We're going to cheat a little bit but we're not going to cheat this much. Storage has become super cheap. This is the price per byte of storage. A dollar per megabyte of storage and mostly what I just want you to see is it's a log scale and it's going hella down. And "hella down" in scale is "really down." And this particular curve going down it doesn't actually show the very, very, very cheapest form of storage, which is tape. Tape is so friggin' cheap. It's a giant pain to work with, but it is super cheap. So tapes can only store one big file. One long file. So we built a tool called the "tape archiver" which takes many files and turns them into one file, now you know what "tar" stands for. And so we just write one big file to the tape but it's a tar file and so it contains multitudes so that's nice. So let's talk about Quinine. Quinine is a program that quinifies your entire code base. It will produce Quinine.h, Quinine.c if you're asking for C code, it will produce Quinine.py, if you're in a Python context. It creates a string that reproduces the surrounding code base so it's an automatic thing for creating that library function so if you do your command line processing right, you can say hey, if I get the dash, dash sourced option, print out the source code. That's kind of cool. So how do we make one string represent multiple files? Tar! Okay. I said the problem was usually quotes, right? How do we deal with quotes? We could do it that fancy way with the Python making it all small and elegant or we can cheat. And this is where I get this feeling of cheating. That's where base64 comes in. Base sixty four will encode arbitrary binary files and put them into a thing with no quotes, with printable characters. We've turned one file into one file of tar. We can turn problematic quotes into just data with base64 code, so let's do it. And by the way, this is all in pursuit of a Middling pun at best. Quinine can be produced from coal tar, it was a big thing in the mid-twentieth century, because at first you had to get it from bark and stuff. So if you look in this repository, I have a program called program.c. Program.c doesn't do much. So when I say that, will it fit? It will fit. And you can say, this is actually almost the entire file. There's a usage message that says if you print help, if you print this message, if you print source it will print the source for the program. And then all there is in main, and this is the whole file now, it just says, "Get the options and then if it has to print out the usage do that, if it has to print out the source, otherwise, print out "hello world." It really is that silly of a program. Well, as part of the make process. These are the only files. There's Quinine.sh, and program.c. Notice Quinine.sh. So now, when I run it, let me… so now, I've

created, like, I just said, Quinine.c, Quinine.h, if I run program, it will print out hello world. If I run program, I think it was dash -- source -- dash, dash, source. Program-source, this is not helpful. But remember, base64, antar. So base64, dash D for D code. And then pipe it through tar. And wait, you can't see this. Let's fix that. So let's program base64, dash D for D code, and for now, just look at the table and you can see yeah, it contains the makefile, and Quinine.sh, and -- so that's true, let's make a directory temp. So now, we'll run, and we'll pipe it through tar xv.

>> Dot dot.

>> Oh, cd temp. Thank you. We've made it. There we are. We've got program.c again. Just so… ctrl + L. We've got program.C, Quinine.sh, and makefile just as promised. Ta-da, we've made a quinifier. So Quinine is quines made from tar. Thank you very much.

[Applause]

ERTY: Okay. So now, it's time for some unconferencing. It's now approximately 3: 10 and we're going to start with the next set of talks at four thirty:. So we have a short break and about an hour of unconferences time. There are a bunch of sign-ups in the back with sticky notes on those white pieces of paper with cool things that you can go talk about with other sided people. So mingle, talk, have a great time. Four thirty:, we'll be right back here.

>> Tell the speakers to go in the corner.

ERTY: So if you're in the next set of speakers, so that means, Lea, Jon, Bonnie, and Kevin, please go check out in your stuff for AV in that corner. Thanks. One more thing. There is coffee. There is coffee over there in the lunch room for those who want coffee.

[Short Break]

>> AV dude: Everyone here, I just want you to know that everything is streaming live. So if you don't want to be broadcasted.

>> There is some stuff that I have to say and ostensible know stuff. So this stuff is real as it gets. So the commonality is that they involve something physical and they're cool. So I was thinking how I would present it so that you guys don't ignore it. We have sponsors. Don't ignore them. So I decided

I’m going to intersperse sponsors surprisingly throughout this presentation. All right so the first talk is by Lea aAlbaugh. and by the way, the amazing sponsors are NYU, Mandrill, and TheLadders. So the second talk is light painting with robots with robots and if you don’t know about light painting. The excellent sponsors, Hacker School -- I’m sorry, rubbering center. I’m so sorry. It’s actually on our website as Hacker School. I didn’t think about this before I said it. Third talk is music,, programming, arduino, or building musical interfaces to create awesome. And that is -- it’s not part of the title. That’s what Bonnie Eisenman -- sorry, I forgot to ask you how you pronounce your name.

>> “Eiz-man.”

>> And so the great sponsor is BuzzFeed. So the amazing ones gave us, like, too much money and then the excellent ones gave us, like, also too much money, but less than the “amazing “people. Now, I feel bad. I’ve sort of locked myself into a corner here. They also gave us too much money but they’re still great. Okay. So following that we’ll have closing remarks and then the optional part of this is you can coalesce into dinner groups. There was -- we couldn’t organize people ahead of time because not enough people responded but we had some suggestions if you would like food and people you should perhaps coalesce and toing to one of the following restaurants: Shake Shack. Han Koes (phonetic). Chipotle. I don’t know, I guess we should just meet in the lobby and figure it out. So I’ve got the sponsor stuff. Remind people to show up tomorrow. Yes. So the space opens up at nine forty five:as usual. You will receive breakfast for free thanks to our sponsors. Talks begin at 10:30:we hope. Actually no. Yes, no. Yes. Yes. Yes. Yes. Yes. I said that correctly. I’m going to leave it intentionally ambiguous so that you show up on time. Okay. So with that I will hand it off to our first talk, which is Lea. Or “lay-ah” sorry.

LEA: Mmm, technology. All right.
LEO: You have an HDMI port over here, right?

LEA: You think I do. This goes to wherever the HDMI goes, if you think that is… if I put it here people will hear me?

>> Yes!

LEA: Awesome. It doesn’t sound to me like you can hear me but that’s okay.

>> I can’t hear you.

LEA: You can’t hear me? Well, that’s not good. Okay give me a moment. Can you hear me?

>> Do you want me to hold it awkwardly next to you?

>> There you go!

LEA: This hair has to be good for something, right? No. Thicker hair. There’s a new body image issue to have. Yeah? Yeah? In the back?

>> No.

>> Most…

>> No.

>> Nose.

LEA: No. That sounds like it would hurt.

>> Do we have a stick? I bet we could put a stick in this podium.

LEA: I stick to physical things in the real world. Okay, how about that?

>> Can it go higher?

LEA: That’s a really great sound.. Okay. How about now? in the back?

ALEX: Maybe lean in? Sorry --
[Applause]
[Laughter] all right. On that note. Okay. Hi. I’m Lea. I do a lot of stuff with code and textiles and so sometimes I I do stuff with computation mixed into the final product. So that’s a cocktail dress mixed with microcontrollers on the inside. But right now I work with a research lab, so I work with industrial machines that make the fabric itself and then the machines are controlled by the code that I write. One of the awesome machines that I work with with a knitting machine. Knitting, for those of you who haven’t met it, it’s a way of turning a yarn into a surface through careful intermeshing. The path of the yarn zig zags from the bottom to the top and then the loops are connected by being knit continuously by a continuous string and then the loops stack vertically by align and then each loop is

held in place by another line that's held above it. That's important. Another loop that's not put through it is thought of as alive or unstable. So this one is held by friction and you could unravel it with the thread as you walk away. In handknitting. The lives are held next to each other by one needle and then the new ones are -- and in machine knitting each loop is allocated to its own hook-shaped needle which is capable of pulling through a single new loop like the animation. Here's the formation of a new loop. I also have this sweet hand-crank version if you want to come play with it after the talk. You turn the crank. It knits. So the knit stitch is the most important atomic operation that it can do. The second most important atomic operation is the tuck stitch. Pulling the loop through without dropping the previous one. You can use these for textural effects, for adding elastic, or doing color work if you're using two different colored yarns for example. Just to throw it out there topologically, the first row you knit is a tough row because there's no loop there. A third atomic operation is the transfer. There are thousands of these hook-shaped needles and they're tiny and they're called beds. So there are both beds in the picture and they meet at a ninety degree angle. And any of those can transfer to the needle that's directly on the other bed and so we can transfer from the front bed to its corresponding needle on the the back bed and vice versa. We can also change the alignment, and this is called racking the back bed and it moves the back bedside ways. And so those are four things that we can do. And how are they useful? Four things. Let's look at some swatches. Rectangles are really easy. we can just make some stitches with one row, headache another row and note this slide we're reading from the bottom to the top because that's how the stitches are formed. Gravity makes it so.

Okay. How about lace, though. That's kind of complicated looking, right? A lace is any fabric with a pattern made of holes. So there are actually lace making methods for all of the textile processes. In knitting when we want to make a hole we can't just drop a stitch, right, because every stitch has to be held by at least one other stitch because all of its predecessors will drop. You can move a stitch out of the way by transferring it to its neighbor. We move the bed one way or the other and then we transfer back up to its original bed and then note the stitch above, that hole is a tucked stitch because it doesn't have anything to pull through. Okay what if we move a whole chunk of stitches over? And like, maybe all the way to the edge? We have a fabric that is now slightly narrower. Or conversely we could move some fabric outwards and we would have a fabric that's slightly wider. Okay, this is kind of my Julie, this sounds kind of boring but I tell you it's really amazing! [Laughter] because fabric that is slightly narrower or wider is how we make complicatedly three-dimensionally shaped fabrics for our complicated three-dimensionally shaped bodies. So we've got the ability to make these arbitrarilily shaped surfaces out of these four atomic operations

and these are machines that exist right now in the industry. And I think we can probably assume there's some really cool UI that fancy designers and law spaces that are telling what these machines what to do? No. Right. Here's the proprietary programming language that is used by the company that made the machine that I use. It's a two D array of color values, AKA, pixel art. You read it from the bottom to the top, and each horizontal line of the pattern corresponds to a pass of the carriage. The carriage is what actual articulates the needles. Because they're so tiny there's not like a solenoid hooked up to each one, they follow a cam path in this physical hurdle that articulates from side to side and articulates them. This is just one pattern because they get tall very quickly. So let's zoom in. Okay, stuff in the middle is essentially stitch-by-stitch instructions. Most of stitch combinations are shorthands for the combinations that we went over before. So fifty one and fifty two pale pink and pale green, correspond to, knitting at two, again, obviously and then the over purply colors lower down are transfers forward and back. The streaky lines up the side are called option lines and they carry machine instructions for that entire pass and so the rack value that's moving side to side set by that line which is ally because it's set by two lines, it's set by the mauve and purple line, and they interact in kind of an inexplicable way. And and you can do repeat blocks as well. Had so yeah, this is pretty easy to read but it gets complicated really quickly when we're talking about something more like a glove, for example. I've spent about six months learning to read these. This is a real system that we really actually use in industry. Pretty sure that we can do better. A large part of doing better is doing physical simulations of the object because that's kind of one of the better ways to do UI in my opinion but we also need to think about what kind of abstractions we're going to build on top. So one thing that we can right off the bat, we can do some loops as a joke, because knitting's all about loops. So this is straightforward. This is a nested loop and it's drawing a rectangle and this is oversimplified because it turns out we care about directionality we care whether the SKWRARPB going leftwards, you are right wards, back to the front, or front to back. And back to front and front to back is determined by which loop we're using, so we really want something more like this because we're alternating rows going left and right, we need two at a time. So this is actually what it looks like to knit a rectangle actually. So how about that business with moving the stitches around with all the transfers and the wraps. So this is basically a pseudo-code version of what I showed you earlier. We're transferring back, and then coming forward and then -- and it's a good idea to set your rack back to zero so that you're set up for the next time around. This is great. This is all we need if we're transferring just one stitch around. Something that we haven't talked about yet is knit time efficiency. Yeah. Something -- one of the nice things about this format is that they make visible something

that’s actually really important especially in industry. I said earlier that each horizontal line corresponds to a physical pass of the carriage. It’s a thing that takes time to do. So there’s actually a direct correlation between the height of this drawing and how long it’s going to take to knit something. And one of the things that will get you really quickly is the transfers because the transfers have to happen in a row all by themselves and remember we can only set the rack value per line. So this thing that I just code up takes a minimum of two passes which is great if you only want to do one transfer, right, or one sideways movement but if we’re talking about a lace where we’ve got three hundred stitches in a row that all want to go over by one, and we just naively call this thing three times, we’re talking about six hundred passes for a single, this tall row of knitting and it’s probably a bad idea. So we want to do something more like this instead. Transfer all our stitches back at the same time and rack it, and then transfer them all forward at the same time again. Yeah, this looks really good. But what if the stitches are going to go to different locations; they’re not all going plus one. Maybe one of them’s going minus one, and one of them is going plus one. Or maybe they’re swapping places if it’s a cable knit or maybe we want to knit it evenly across its width. The ones in the middle are moving quite a lot and the ones in the end are moving just a little bit. Here’s a fun one. The has a maximum amount that it can rack over. It’s a physical piece of metal that has to stay inside a bigger physical piece of metal. So what if we wanted to rack over more than the bed can actually move, and what if we wanted to do all of these things at the same time? Ehh -- ah, mmm. I’ve actually written all of these cases specifically but actually haven’t written a function that combines them altogether because I actually have no idea. This is one of those talks where I’m like, “I have no idea but we’re accepting interns.” so in the meantime, though, I’m having a really good time turning this nonsense, which kind of looks familiar to you into something that looks more like this. Look at this awesome thing that is -- these look the same, right? Yeah! All right. I’m out of time. Come talk to me. We can talk about modular arithmetic in short rows.

[Applause]

ALEX: Okay. Maggie’s on top of it. Maggie’s going to help us shut the windows. By the way, Lea designed the shirts that you will receive tomorrow so that is why you should come back. So next up is light painting with Jon Williams. And we’re going to shut these. So just hang tight.

ALEX: You should consider talking amongst yourselves.

>> Your ten minutes are running out!

ALEX: Um, yes. So you should turn to your neighbor and consider asking insightful questions such as: Nice shirt, where did you get that? Okay. You ready? guys, guys -- I mean, people.
JON: Hi, my name is Jon Williams. I'm going to talk about light painting with robots in Javascript. and there's going to be a live demo. So say your prayers to whichever deity's in charge of live demos for you. Can you hear me in the back?

>> Yes!
JON: All right. If I get to seven minutes, -- person that's in charge of time… and I'm still -- I was going to ask -- okay, great. I'm Jon Williams. I've been doing online front end something or other under shove media for fifteen, twenty years, it seems like. When I was a kid, I had one of these guys. Anybody have one of these? Hell yeah! This is the Armatron. The Armatron was a badass robot. So I had dreams of hooking this up to a robot, or a remote controlled car and I took this apart and found out that these enjoy sticks are not electrical. There's a series of concentric gears and when you press on a joystick it engages a clutch into that thing, and there's one motor in there that makes all this noise and it's the most ingenious thing that I've ever seen and you can't hook it up to a computer. So this is the U-arm. This thing showed up on Kickstarter. 250 dollars, I think. That's a suction cup on the end and it's able to pick up about a pound. This is sped way up. And you can control it with a mouse. I can't get to -- how do I go back? Not that far back. You gotta watch the intro again. I'm sorry. All right. So, I bought this thing, decided what am I going to do with it? I'm going to try to paint with light. So this is an early test. I'm actually filming with my phone. The computer screen, I did this in After Effects. I just threw in a green LED on the end of this thing. And used After Effects to get the paint effect 'cause I didn't have a proper camera and this was sort of the first result. The first program that I wrote, I'm attempting to draw ado nut. And you will see that theme. This is a still using a different After Effects technique. It's adding up all the brightest pixels that it was able to find. It was awful, and it was at this point that I decided I needed a proper camera. The blue and red at the back, is the lights on the back of the robot. And so it's like swiveling on its base. So you're seeing it sweep out that arc in the back. `. Yeah, no. You can see that, first of all it's really tweaky and it's drunk. And if you look closely, you can see that it's sort of pinched at the back on the right-hand side. It's not properly doing the geometry. This was before I had the polar coordinates math sorted out. This is the same program, different lighting conditions. And then this, I've been joking that all electronic artists are required, by law, to use the Mona

Lisa. And so this was the first picture that I made that you could, sort of, tell hey, this is working.

So at this point, I had not built the iris that's on the front of this thing that allows me to control the size of the dot. It was just kind of like a raw LED just kind of floating out front. This is one of the best shots I got. So lower your expectations to about here. You can see that it leaves gaps. The mechanical addressability is not very awesome. We're going to come back to that. Trying to figure out a way to deal with those gaps in a minute. So one of my first tests, though, to try to deal with that was to give the arm a little wiggle to get it across. That totally didn't work. I got this, though. And this is my favorite shot of all of them. It's like "Tweaky Lisa." this is Starry Evening. Which did not come out at all like what I wanted but there you go. All right. So then I decided, like, okay, let's gets serious, let's build an iris. Let's control our light a little bit better and I drilled a hole and stuck it there.

This, I used one of the leftover plastic parts. And this is black construction paper, and, sort of,, like, early prototyping. This is me gluing pieces of bent metal and I'm using the quarters as a weight. As a vice because I don't have proper anything. This is an early working prototype. This is, like, prototype number three. This actually worked. And I was really excited but this didn't actually work when it attached to the servo. And you see like hot-melt glue and wooden dowels and I've drilled the holes for those -- I'm losing my words because I've been up too late. Anyway, yes. So this is one of the earlier tests like, can we draw with a controlled iris? Yes we can. Yes, we can. This is supposed to be a spiral but you can see, like, again, that mechanical addressability is not quite what we would like. `

This is a -- if you go on Wikipedia and look for a Vogel Spiral, Vogel figured out this mathematical arrangement of sunflower seeds, I'm using that to create a couple of images. `And again with that. Sorry. I was going to show you code. We're running out of time. So all right. How does this work? I put all this on GitHub so we're going to just really quickly talk about. It's a serial port. This is on the arduino side. And it's a little state machine and we look for certain bits. Whoa, that's not… so we're looking for a 255 here. The basic code here was part of the code that came with the robot arm. But when I tried to talk to it with node, I couldn't figure out how to get those numbers to actually go down the wire and so I did want to show you the secret sauce. And the secret sauce is buffer. So I build these arrays of data that I want to go down the line. And you can see me making a new buffer that, where concat, if we're talking about the position we're going to send the robot next plus the positions where I built up here… right, we're just doing not too fancy. come talk to me if you want to know how this works. If we write that out to the U-arm, and if there's an error, we log it. You're

looking at the state-of-the-art in error-handling right there. All right. So then I wrote some more programs.

[Applause]

And this was within twenty four, forty eight hours of finding out that my acceptance to !!Con had gone to my spam folder. This is another one. I'm reading the fill color and that's what's setting the -- so this is coming out of Illustrator, so it's an SVG file, which is awesome! Let's back. I tried to draw planet Earth. This is an SVG file from Wikipedia that I hand-drew very meticulously and the robot screwed it up. That's Africa. I don't know if you could tell. It's painting the front and the back, so some of the mess-up is your perception is, like, you can't tell the front from the back. North Carolina -- sorry, that's where I'm from. North America is up top, South America sort of fills up the bottom right quadrant. That's, like, India is top center, I think. And I decided to animate.

So this is actually a STL file where I'm pulling the coordinates and I'm doing matrix rotations using an awesome node library called Sylvester. But again, the mechanical repeated of the revise is really bad. And so I wanted to try to figure out a way to get rid of those rectilinear rows in the Mona Lisa and so I'm painting the red channel at one angle and then the green channel at another angle and the blue channel at another angle. And I spent a lot of time trying to get this to work, and I thought this was a really good idea -- and this is the best one of that series. So… all right. So the same thing with the sunflower seeds. I was like, you know, it would be sort of the artistic thing to do would be able to use the sunflower seed layout routine to paint van Gogh's sunflowers. This is van Gogh's sunflowers did not work. And they're telling me I'm out of time.

GitHub, Shout Media. And can I play this awesome video of other work that's totally not mine? This is the guy, Yurg Lenny did Hector in 2002. This is a spray paint robot, and he also did paper.js, which allowed me to do appoint some distance percentage along a path. It's awesome same -- Yuri Lenny also did Victor in 2006. I'm totally jealous of the quality here. This next one is my favorite. Has anybody seen this guy? This is Robo Rainbow. I want to be this guy when I grow up. He's connected his robot to a bicycle trailer. Those are spray paint cans.

>> Oh, my God!

JON: Yeah! Yeah!

[Applause]

if anybody knows who this guy is, like, I want to buy him all the drinks. And if you've not seen Batin district manager ali (phonetic), this is industrial robots moving. So this is projection mapping, this is 3D animation, this is, like, mind-blowing stuff. Yes, there are mechanical repeatability is a lot better than mine. So this is really cool. `they take

Maya and they make virtual CAT scans of 3D objects, turn that into a Quicktime movie, sweep the iPad through physical space and this is the pixel stick. It's a rod of LEDs which they upload an image or an made the gif and they open up the shutter on the camera and they wave it through. And I was supposed to do the demo. I was supposed to do the demo while I was doing this. So did anybody see it running a second ago? All right. Anybody see this?
LEO: Yeah.
JON: The camera just opened, it's painting. Can anybody guess what's drawing?

>>

Bang-bang logo?
JON: This will take thirty seconds.

>> It's so cute!

JON: We're going to transfer that back to the file system. Please let it be there, it is.
[Applause]

>> Aww!

>> That's so cool!
JON: Thank you guys so much. If you guys have any questions, come find me. I'll talk about it forever, as you can tell.

BONNIE: All right. Hi, everyone.

>> Hi!

>> Hello!
BONNIE: I'm going to talk to you about music programming and arduino. all the SKHRAEUPBLGS points, also known as building electronic interfaces to create awesome. So before we begin, a little bit about myself. Hi, I'm Bonnie, you can find me on the interwebs there. I'm a software engineer at Cubaing. And so I work in software in my day job, but I have a bit of a hardware habit. I started arduino in 2013, and then I started music about a year ago and I find this stuff really cool and really fun. So I like building electronic musical instruments. As I say that, people are usually not along with me. And they usually look at me funny because we're used to two of the phrases in here but usually not all of them together, right? Everyone has an idea of what

electronic music is, yeah, we all have an idea of what musical instruments are, whether that's the violin, the drums, the human voice, we all have an idea of what musical instruments are, or should be. But we usually don't see them put together, and part of that is the way of creating electronic music right now, a lot of people the way that they perform it, interact with it is through interfaces like with buttons and sliders, and knobs and all this stuff that's weirdly unexpressive. And I find that very strange because electronic music, right, like you're literally shaping sound out of nothing. You have this huge huge range of things that you can produce. And yet, we don't usually get the same expressive interface that you get from a normal musical instrument--"normal right? So I like building electronic musical instruments. I'm going to show you some two quick videos of some of mine. I did not do the live demo route because the live demo gods do not always smile upon me. So here's one. This is called, "Love Music."

Yeah!

[Applause]

thanks. So as you can see, that one uses what sing a pretty natural gesture right: Touching the water. And you get sound. and there's no magic there. This is safe. And that's doing touch sensing. This is another one. This is my lumiphone and this is alight based one. Also stolen from a coffee house. Okay. I'm going to stop there. so, this one is, you know, it uses light and it lets you control pitch, volume, and vibrato based on how you wave your hands to the air. So you can literally reach out and grab the sound and move it around, right? So with both of these projects I've had some fun leaving them in public places and watching people interact with them. I left this on a cafeteria table and just saw what people did, which was great and I've also posted some videos on YouTube and got some great reactions. this is something that I kind of want to frame up on my wall which is weird. I could literally wave my hands over rocks to this music and make it look like I was playing RCKS! Which, as far as what YouTube comments, it was not what I was expecting. So, when I build these kinds of things, I really like them because they get, partly because they get good reactions. People react with excitement or disbelief. And, you know, I think it's kind of magical.

If you're like me, maybe you like programming because you think it's kind of magical, right? It's like, "I can make my computer do things." That's definitely why I like programming. And if that's true, I think hardware projects, at least for me, are doubly magical. Because with these kinds of things, you're literally taking in the real world, like, calling it into your computer through sensors, doing something processing through it and then pushing sound back out. And so you get this fun feedback loop where suddenly,, like, yeah, the programming is really awesome, but then it, kind of, just disappears and suddenly I have this interaction where I'm just like literally waving my hands in the air, I could be playing rocks! And, like, I

really like that. So I didn’t get up here today, though, to say that makes me feel like a wizard. I got up here to be like, “And you can do it too!” So ingredients in these spells. Nursing home, arduino. Raise your hand if you’ve heard of arduino before. So arduino has been massively successful over the last few years. The first being that it’s affordable and this is huge for barrier of entry. Your average arduino board costs between ten and thirty dollars now and you can just get started. The sensors that I used. In that mug, that’s two dollars worth of sensors. In the cup, that’s six dollars worth of sensors. These not -- you can actually afford to play with these. They’re great toys. And the other two, which are super, super important and echo with some of the keynotes today is that it’s accessible. So the arduino community has put so much effort in making these things approachable. You don’t need electrical experience. You don’t even really need programming experience. when I do workshops, we’ve taken people with zero knowledge and gotten them to the blinking lights stage at the end, which is great. So arduino, it’s wonderful. And what it does, it’s a physical computing platform so it lets you deal with input and output, right, in the real world. And so you have all these sensors for musical instruments. This is of course, super important. You want to build an interface. So you’ve got light, touch, water, GPS, motion, lots of stuff. And I included this gif because, you know, kind of feels like you too, can master the four elements. Except I guess instead of bringing balance to the world or making music, or whatever -- minor details. So that’s great. And then the second one, and this is one that I think far fewer people will have heard of. Raise your hand if you’ve heard of ChucK. Oh yay. That’s exciting. So ChucK is a musical programming language and it was developed at Princeton and it’s designed for use by musicians and that means, like, the primitive things that you work with in ChucK relate to things like time and sound and so you can get started making things pretty quickly. It’s free. And it’s cross-platform. And it’s not some fancy software that you’re going to pay money for `. So making sound in ChucK is pretty simple. This is “hello world.” Yeah, you have to start the virtual machine. That’s kind of weird. Sorry about that. That was kind of loud. So yeah, this is all it takes to get started with ChucK. You say I have a sine wave, send it to my speakers and then wait one second. That’s what you will that says. And then if you want to do something like input. I’ll do something with 440 unless anyone objects. Yeah, and then I can mess around with this, right? So we all know frequency, doubling make octaves happen. Yay! So that’s, like, ChucK very, very briefly. I really like ChucK because the fact that it’s so easy to now program sound means that you can just like use whatever parameters you want, right? We’re programmers, we know how to take things and make them do other things. So how do you combine these? Music, and programming, and arduino somehow work together? It’s actually more straightforward than you might think. So we take a sensor, in this case a

light sensor, it costs a dollar. Plug it into an arduino board, and now it gives you a value. Let's say, zero through one hundred. Now you've got the state of it, what do you do with it? Well, you can literally if you're lazy like me, you can take a USB cable and plug your arduino into your computer and send it over serial and it turns out that that's really, really straightforward and easy and cheap. I like the combination of these things. And then you can just process the data in ChucK, right? And then again that means that you can make it do whatever you want.

I'm obviously handwaving a lot of stuff away here. If you want to try making this stuff and see what actually goes into it, and want to confirm for yourself that I'm not, like, telling lies… these are three of my projects including the first two, the ones that I showed off on YouTube.

I also think they retweeted the slide deck so you can get that later. And I have all the code, I have all the circuit diagrams if you build them yourself. Piano stairs are something that I didn't show off but they're a really good beginner project. So go make things. And I'm going to end on this great meme:, anything is possible if you use your imagination. Because the cool thing about providing these ingredients is that anything is possible. Like the lumiphone that you could control like this, you could use it for vibrato for music. Or shadow, so you make a piano key. You could also, like, imitate the Apple watch and, like, stick one on a wristband, right, and get a really poor man's pulse sensor, and get one that changes a piece of music with how agitated the user is with a sensor that costs a dollar. And there's a lot more that you can do. So literally, the barrier of entry keeps falling and it's gotten to the point where you can just mix and match these things and come up with wacky stuff. So on that note, thank you.

ALEX: What's up? It is at this point that you might be thinking the astute observer, Alex mentioned three talks. There are three talks, we are done. But it is at this point that I embarrassingly admit to you that I forgot to mention a fourth talk. Like finding a twenty dollars in the wash, here's Kevin Lynaugh to tell you how to build cell phone

KEVIN:

KEVIN: All right. Cool. All right. Can you hear me okay? Yeah? So for a front endlong time, for a couple years I had this great phone that Motorola made for the Third World called the Motofone F3. So you can't actually make exclamation points on there so that's why it says II con there there. But

that’s the only downside. It lasted for several weeks. It’s indestructible. But unfortunately, AT&T shut down their 2G network so I had to get a new phone. So my options at the time were to get some kind of smartphone but I find these distracting and annoying and stuff, or to get another dumb phone but all of these are sort of cheap and plastic and stuff. And so I decided to pursue the obvious solution, which is to build my own phone. And in particular what I wanted was a phone that matched the rest of the stuff that that I have or use every day. Things like a bike, sunglasses, or a watch. All of these things have a purpose but they have all sort of -- they had this aesthetic quality was that they were nice, and in ten years, they’re still going to be nice. It’s not just like a gadgety technology thing. So that’s what I wanted to have out of a phone. There’s three things that I’m going to talk about. There’s some electronics stuff, some software stuff, and then some industrial design stuff. All right. So it’s a phone. Obviously you’re going to need some electronics. Luckily while I was in college I had a one-semester lab about electronics. And -- [Laughter] -- and as an example from my work at the time, I don’t know if you can see my work down here, it says G minus. So G minus is the kind of grade you get at a liberal arts college. It stands for “good minus.” and its scale, it went through VG for “very good” and E for “excellent.” So G minus is basically a “D.” so this is actually how I made the electronics for this phone. I knew that I needed some kind of cellular computer chip thingy to talk to the cell network and so I Googled around. And I found this guy. This is actually a photo from the press release. So I mean, it’s obviously going to be sick! [Laughter] so it came with, like, hundreds of pages of documentation. So I’m kind of scrolling through this stuff and you can see that there’s a lot of words…

There’s, like, these diagrams and these chip have all these pins and colors and stuff. But you keep diving in, there’s actually some kind of example, helpful example circuits. So this is an example schematic for connecting to a SIM card. And so the module is on the left and there’s some resistors and stuff, and then some capacitors and then you connect them to the SIM card and I saw this schematic and was like, “looks good!” And then the same thing for the other stuff. This is the antenna documentation. This one’s cooler because it has these blob things. I have no idea what the fuck those are. But you scroll down. And you find this schematic. So that’s how I made those schematics. I just copy and pasted everything. I need a speaker and I, like, media microphone and here’s an example of an application and I was like,”Looks good.” Of course that’s only half the battle. Once you have the schematic you have to have a circuit board and lay it out schematically. So how I did this, I played Daft Punk really loud. This is actually fun. You lay all these things out on a grid, and draw all these wires and connect them up and then you pay nice people fifty bucks and you get in the mail, some nice circuit boards. So this was wire, this was just copper and fiberglass.

This was tricky because the components are really small. This is a picture of a resistor on a quarter. So the way that I did this, I got some tweezers. So this is -- I just printed out, like, a map of the circuit board and then I kind of took all the components out of the really tiny packages and then put them on this map and then I sort of unpacked everything. And it was really important, “Do not sneeze!” I’ll never find these things again. They’re so small and what you do is, you get, like, this lead paste and it’s actually like in this syringe. So you syringe this on the exposed pads. And this stuff is like this gooey stuff. And it’s sticky. And you place them on lead glue. But of course you have to solder it. There’s two schools of thought with this. I went with the frying pan school of thought. Some people use toaster ovens. It was, like, super ghetto but it totally worked. So that’s how I made the hardware. You just connect a bunch of things. They don’t necessarily do anything yet. To do anything, you have to of course, add a tiny computer. So the tiny computer has various responsibilities, detecting the button presses, having these chats about this other tiny computer like there’s incoming calls and stuff like that. So this is a tiny computer that I used. It’s actually related to a lot of the arduino stuff. I’ll point out a couple of things. They’re eight dollars in quantities of one. If you buy a thousand, it’s only four dollars. I run it at eleven megahertz, which if you’re running physical stuff, it’s different. You’re running eleven million things. It’s not slow in computer world but if you’re thinking, I need to blink these lights on and off…. “ And then I needed one hundred twenty eight kilobytes of programming space, which is enough for about one.four jQueries. Of course it’s small, there’s no operating system. There’s no thread. There’s no garbage collection unless you write your own garbage collector. You can’t run sweep because of course, you know, nothing else will happen. You won’t have call-backs. So it’s kind of a hostile, hostile environment. So before I started programming I sort of planned all this stuff out using state charts. This is a little picture of my notebook and I was basically just thinking through, if you’re familiar with finance statement sheet diagrams it’s a very similar kind of thing. Just planning out, okay, the phone is idle. And maybe an incoming phone happens, and it’s going to transition into this lit state and then the ringing happens and then I press a button. Or whatever. So that’s how I land in out. There’s a great book I found about writing it out. So if you’re interested in this, I highly recommend this book. So once you’re at the software, you don’t have a phone. This is just a machine that will get you turned out of an airport. You gotta put, like, this in a box, right? So that’s the last part of my talk which is the industrial design stuff. And again I started by just sketching out some stuff that I want in a phone and I don’t actually have woodworking skills or anything like that. And I said I wanted to keep it simple and basically I had a little box that would be made out of walnut and then would wrap the buttons with leather and to draw that

out, I used this parametric design software. And it's really cool because you make a two D sketch and then you extrude that sketch to make a 3D object. And then paste new on that face. And then you can extrude that again or cut that again and so you kind of go through this process of sketching and then building out features until you make a full 3D model of what you want. And this stuff is really cool. This was the first time that I played with this. But it sort of makes emacs and Vim look really like kid's toys. So anyway, I made this model. But, you know, this is just a model on a computer. I actually need to make this thing but you know, before anyone says, "3D printer." I want to make this out of wood, which you can't print…

If this is -- if there's any nineteen ninety this project where I went to full on "crazy town" it would be here because this is when I bought a CNC. It's basically like a robot drill. I guess the speakers are not plugged in. But it sounds exactly what you think it would -- it's like "wrrl, wrr, wrr." And supposedly it's accurate to, like, six microns which is insane. And of course, I'm working with wood. But the best part of this machine was that it came with a hat! I got this model and there's this whole nother world which is really cool about cam software about how you actually move this tool around to, like, rip out the different parts of material and, you know, if you have different sizes of tools there's different constraintses about the stuff you can do. It took me awhile to sort of figure this stuff out. So a lot of busting through different kinds of wood and I broke some of these really expensive drill bits but eventually eventually I made a molds of this thing. And this is MDF, which is -- and I was working with MDF because it was cheap and then I started working with bamboo and walnut and I think last week this was basically to the point where I was at but thanks to the motivational power of `conference talks, I managed to laser print, or, like, laser cut the surface and do some stuff out of nut. So if you want to hiss to see this come see this, and I have some other busted pieces and stuff. But I'm about three hundred fifty hours into this project, which, according to Malcom Gladwell that makes me a three.5% expert. Artisanal cell phones. And from this position of authority, I want to leave y'all with one thing, which is that, making stuff is hard. Making stuff is really hard. And it's been really cool KPWHRURPBG all these different things 'cause, like, now out in the world I kind of look at stuff in a different way about how did they make this thing? How did they make this do that, and whatever. And it's easy to get lost in the weeds with whatever. There's all this stuff here, all this stuff there. Things break and fly out of machines really fast. But when you're out and deep and lost in the weeds one thing that I want to remind you guys to do is to just celebrate your victories. So occasionally just try and step back, and, you know, look around and be pleased with yourself. I was so pleased when I made those lights blinked. That was the best thing ever. I used this machine and I used it to cut this larger piece of wood to three smaller

pieces of wood. It’s awesome. So I want everyone here to try and go make something crazy and reach far but when that you do, when you get lost in the weeds, make sure you take time and step back and pat yourself on the back. Thanks.

[Applause]

ALEX: Maggie, do you want to chew people out thousand? I will do the schtick for you. I want everyone to look to the left, please. Now to the right, now look underneath your chair. Do you see that trash? Pick it up! Okay. So… thank you. I just wanted to say something in closings since there’s a section marked “closing remarks” and we didn’t have a plan now I have to do it. So first of all, thank you and I’m shocked that you all showed up. This continues tomorrow in case you weren’t aware.

So we have T-shirts for you. Are we planning on getting those out now because you don’t want to, like, carry them --

LEO: Tomorrow morning.

ALEX: Tomorrow morning, yes. So that is your incentive to come. So I think the dinner groups are going to be congregating back there. So that’s what I had for you today. All right. Thanks very much. Cool.

[Applause]

. If you’re not here tomorrow, please come find us and we’ll give you a shirt. PA*ERPLT PA*ERPLTS