iroh: Infrastructure for Network Resilience

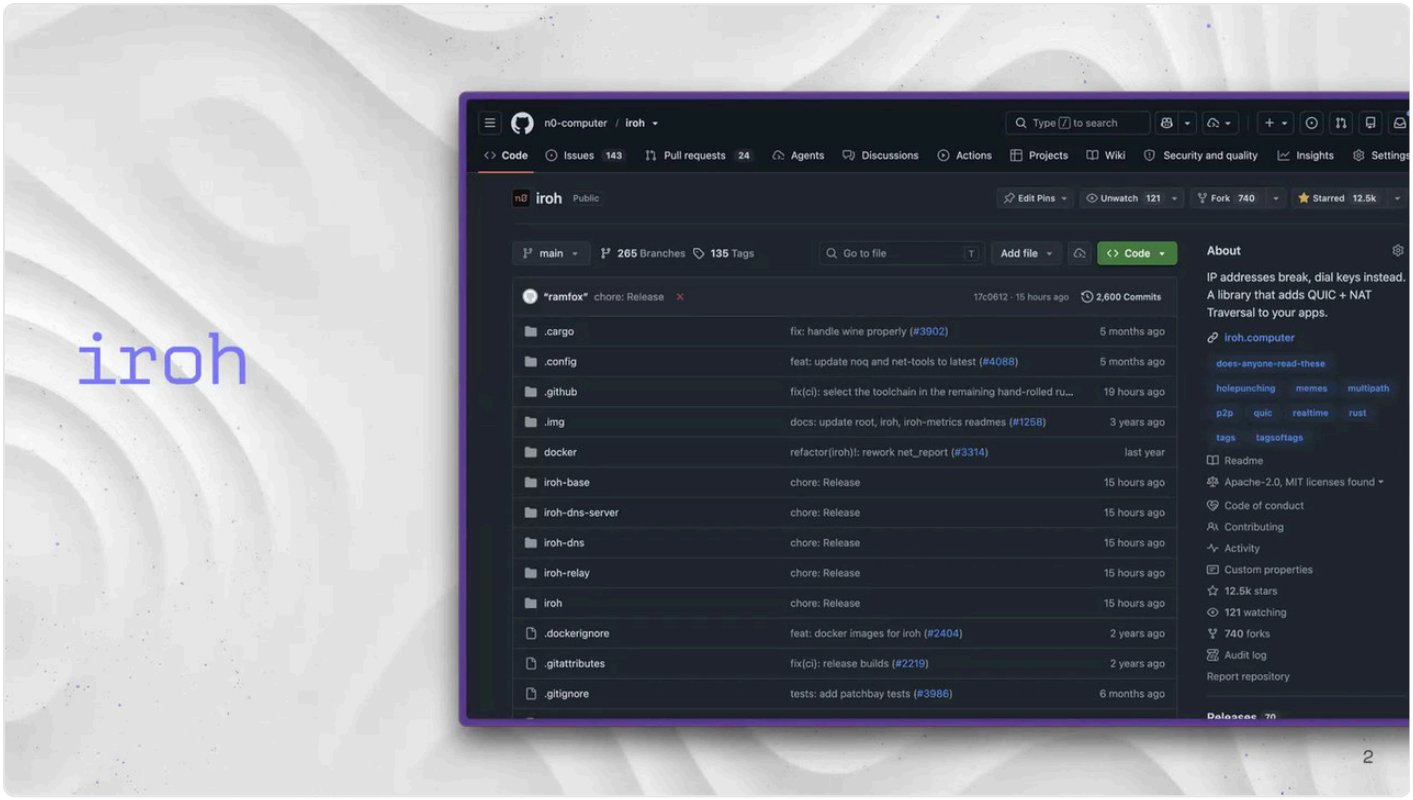
Dial keys, not IPs

First Local-First Meetup · 22 September 2026

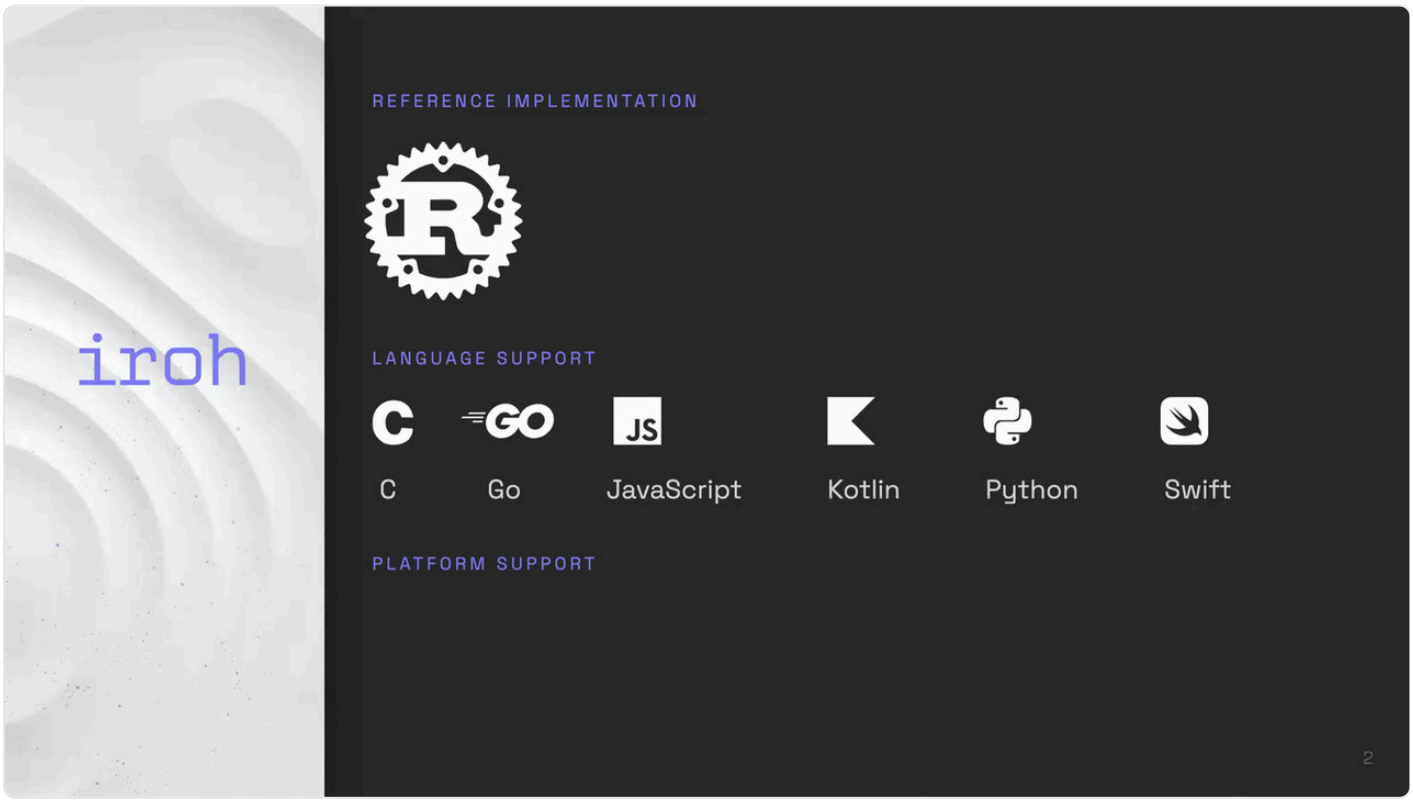
Speaker: Brendan O'Brien (iroh / number 0)



Well, hello everyone. I'm delighted to be here talking with all of you, and I'm here to tell you about iroh. Some of you may have heard of iroh in the past, but I'll take it from the beginning, so if you have no familiarity with what it is or how it applies to local-first, hopefully today is the day where you will learn what all that is.



So, iroh at its core is a networking stack. It is a piece of open-source software. iroh is a library that is on GitHub right now — you can go and pull it and download it and use it. And the thing that you do with iroh is you embed it in your application. This is a library that you use the same way you would import an HTTP stack or something else. It is intended to help you demand more of your networking, and hopefully that will be a theme that you see time and time again.



iroh is written in Rust. We wrote the whole thing in Rust from the jump; the reference implementation is there. The project has been growing for a while, and it's been very nice to see that we actually have our first independent implementation of iroh. It's been written in Go, and that is a ground-up implementation of the actual iroh protocol itself. There are other languages that we also support, so we can get out to C, JavaScript, Kotlin, Python, a whole bunch of languages today, and we can add more if there's something that feels like it's missing. The way we do a lot of this is through bindings, C bindings to the Rust core. The upside of taking that approach is you get a nice, very fast Rust-based foundation for your networking stack, regardless of which language you're using.

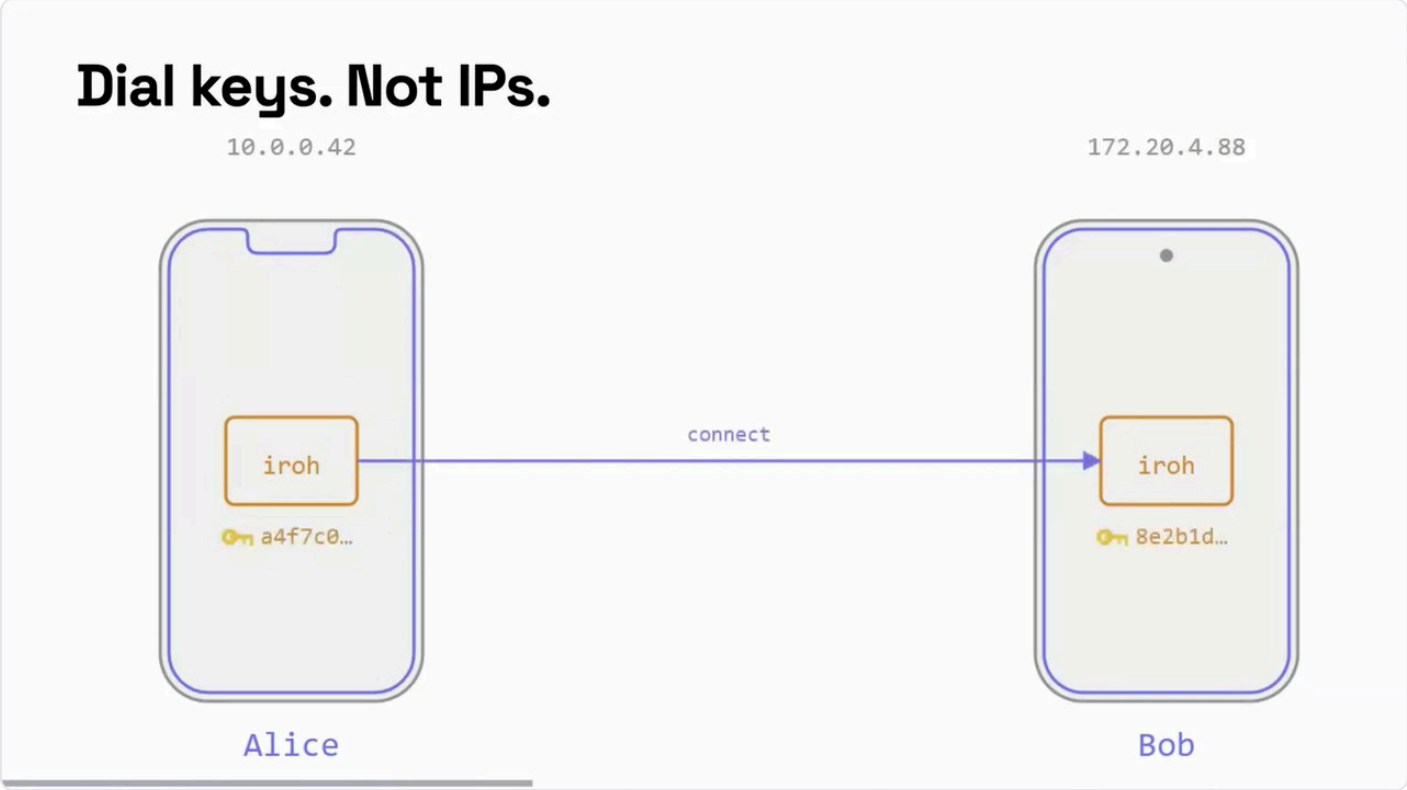
In terms of platforms that this supports: iroh can be compiled to WebAssembly and run in the browser. iroh can run on really, really low-power devices like the ESP32 — the smallest we can get the binary right now is roughly 1.5 megabytes, but we can operate with less than 2 megabytes of RAM, so the whole system itself can run on incredibly tiny devices. We can also run on every major platform that you can think of: Windows, Linux, FreeBSD, macOS. You name it, it works on it, and we test for this continually in prod.



Now, why write a new networking stack? What is the big idea? Our big idea with iroh is that we think that everyone should be dialing cryptographic key pairs instead of IP addresses.

So I want you to think for a moment — imagine the entire internet, and imagine that instead of everywhere that you put an IP address, when you try to establish a connection to another device, you put in a cryptographic public key. In iroh we dial not by IP address, but by public key. Now, we can't change the way the internet works, so under the hood we have a whole bunch of stuff that translates those public keys into IP addresses. But the core primitive that you work with as a developer is public keys. And we use those public keys to give you end-to-end encrypted connections between devices. So you will always have: hey, I want to dial that public key — and when you talk to it, you're going to get a TLS connection that is secured by that key. The really interesting property about that is, as you are communicating, every single packet is being signed with the key that you're talking to. So you have this very exciting new foundation.

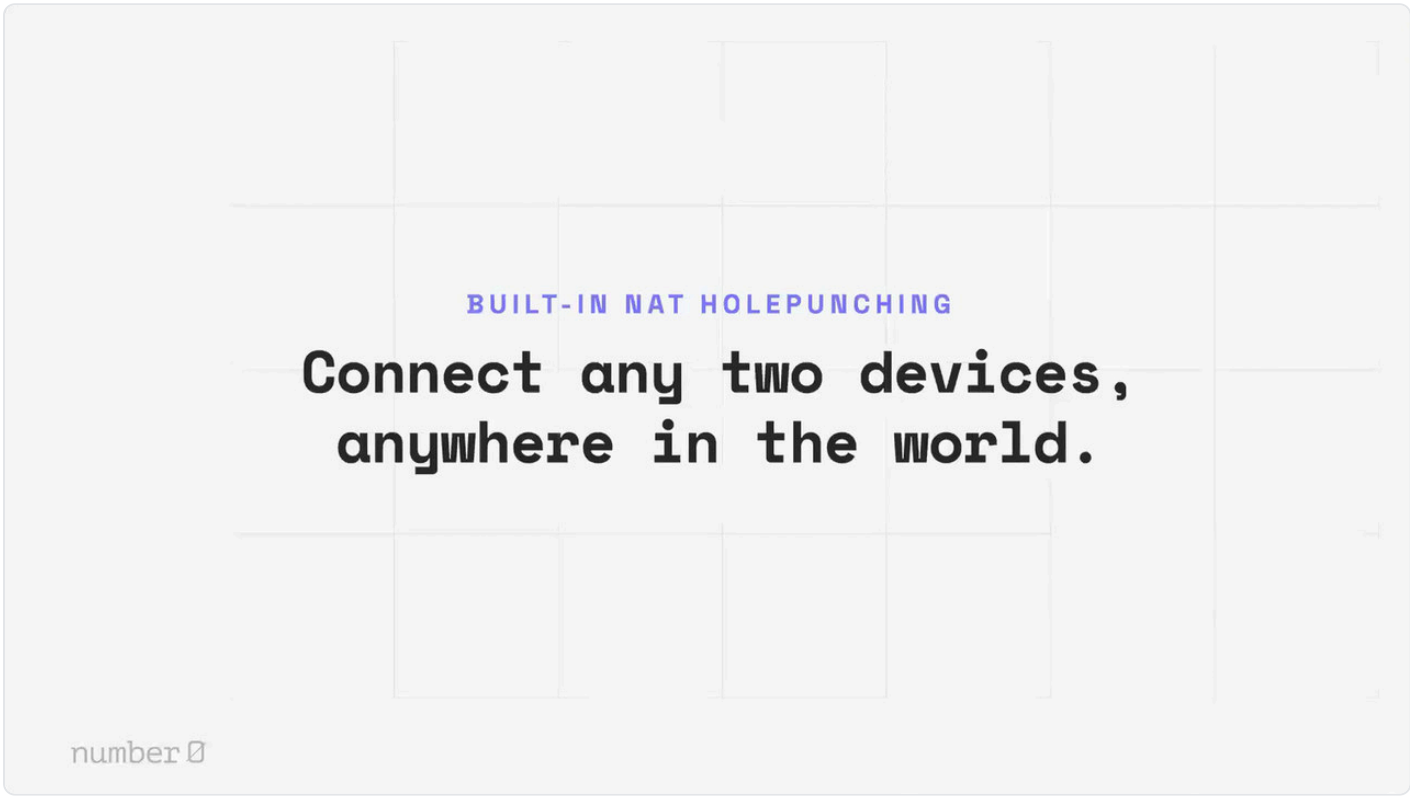
In iroh we combine that with hole punching, so iroh can traverse NATs, which means that you can dial any two devices anywhere on the planet. Doesn't matter. You can be sitting at your phone and your laptop — those can talk to each other. Two servers in different clouds — those can talk to each other, as long as you don't do really ridiculous stuff with your networking policies that prevents that from happening. So iroh is not total magic, but it is quite magical.



The way that this works under the hood is: this IP address is changing over time, so when we try to connect, we do this establishment between keys. The thing to keep in mind here is every key is per

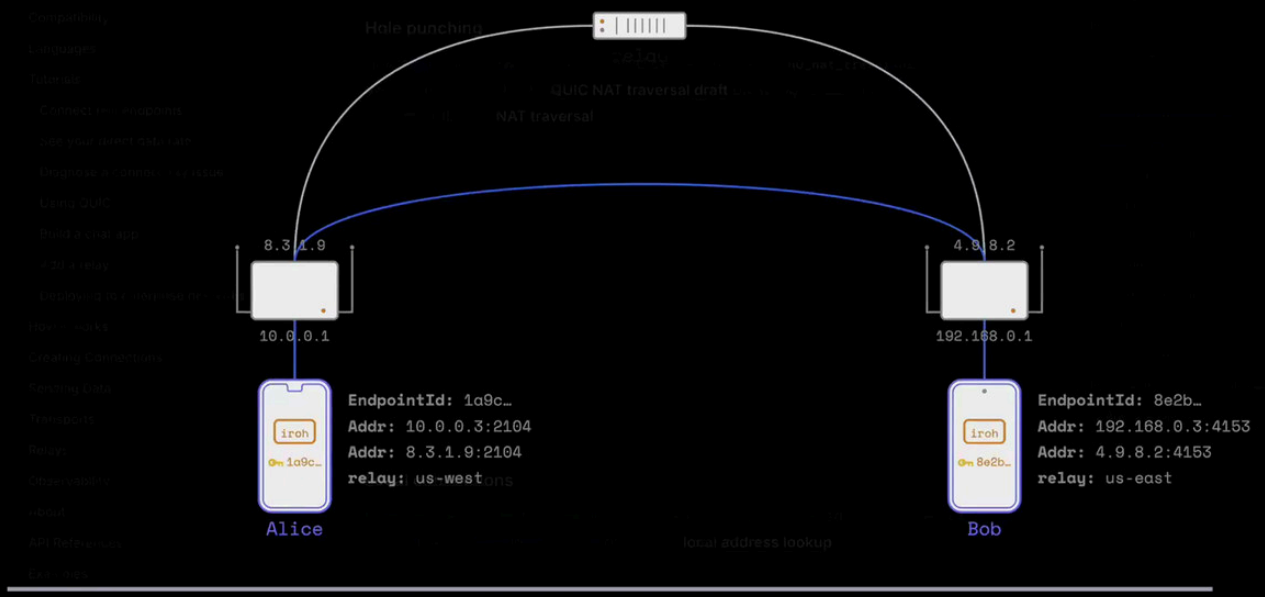
process. You can imagine your app, and one user of your app getting this one key, which allows you to declare a thing that doesn't change over time.

This is the big advantage over using IP addresses as a foundation. With IP addresses, your router can decide what your IP address is at any given point and change it for you. Inside of iroh, that's not the case. We can say: this is the key that I want to be dialed on. It doesn't matter where you are in the world. And as you move around, if you switch between Wi-Fi or cellular connections, or you plug in an Ethernet cable, that connection will stay persistent, and it will work with the key that you're trying to talk to. This is one of the core advantages that you get thinking this way.



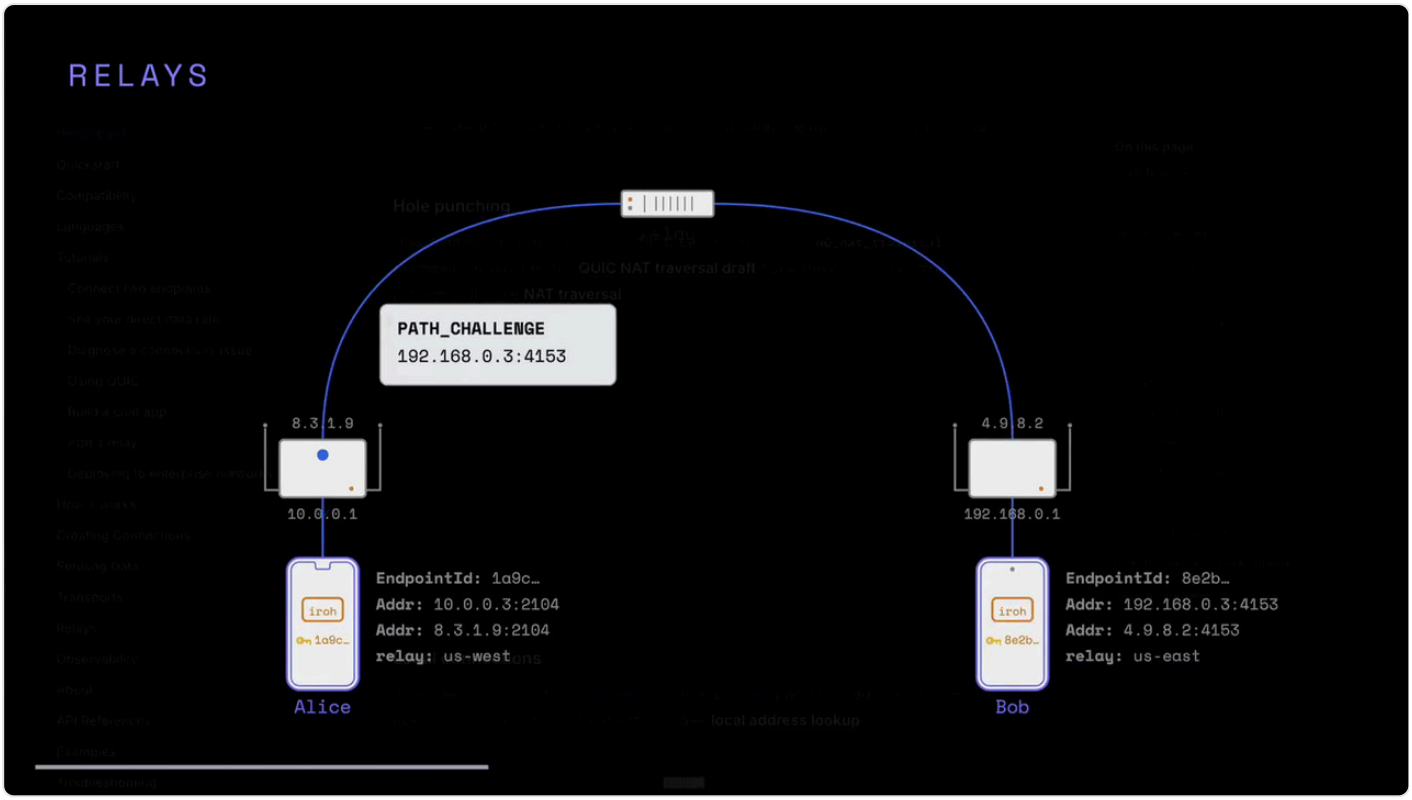
And so the whole idea is: we connect any two devices anywhere on the planet.

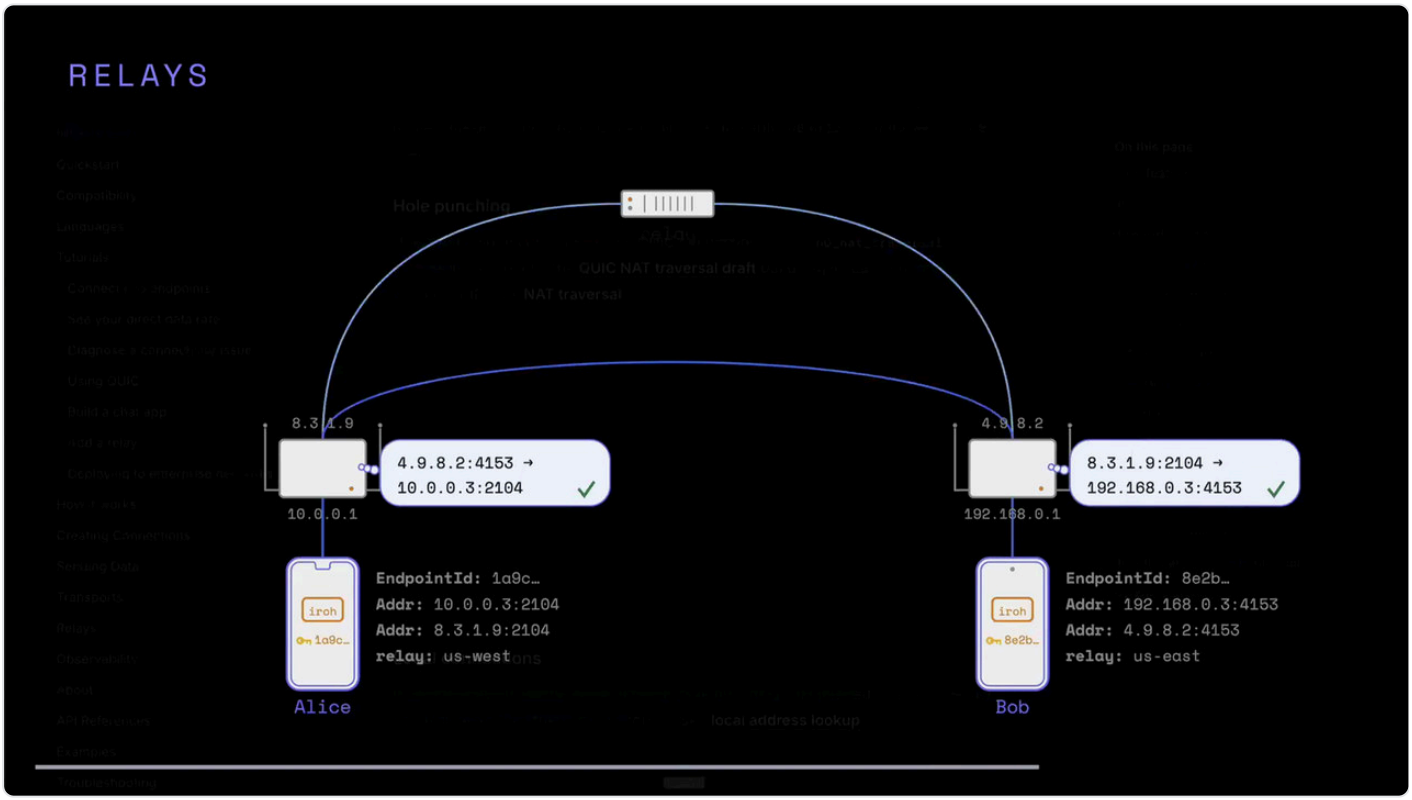
RELAYS



RELAYS







We use a technology called relays that we have written for this. iroh does rely on relay servers to establish these connections, and so every endpoint at any given moment has what we call a home relay. When an iroh endpoint starts up, it will ping any of the relays that it's configured to know about from a list that's given to it in advance, and then it will use that to figure out: this is how I want to be communicated with.

And then when two endpoints want to talk to each other, they start doing this exact process where they dance, and one of them figures out the home relay that the other one is at and says: I would like to connect to you. The relay sends that message down to the device, and then in parallel we start to negotiate a direct connection. So iroh is constantly switching between what is the best connection possible.

Generally, this process that you're seeing shown here in this diagram happens within the first packet. We generally see one packet of data flowing encrypted through the relay. The relays are totally blind — they cannot see any of the communication between the devices. But the first packet tends to flow through the relay to get a fast time to first byte. And then after that, traditionally — or in many of the cases — we will at that point have established a direct connection between the two devices through this path challenge process that you're seeing animated. Once they have that, they swap to it. And then if at any given point your connection changes, we can always just fall back to the relay and then pick back up to direct.

So underneath the hood, this is the abstraction that iroh is giving you. You don't have to worry about any of this — this is all happening for you. It's just: give me a connection that works, give me the fastest connection that we have. And if we have to drop back to relay because for some reason I can't get a direct connection to that device, that will still work, and you won't compromise the security or the encryption of the connection.

```rust
use anyhow::Result;
use iroh::endpoint::{Endpoint, presets::N0};

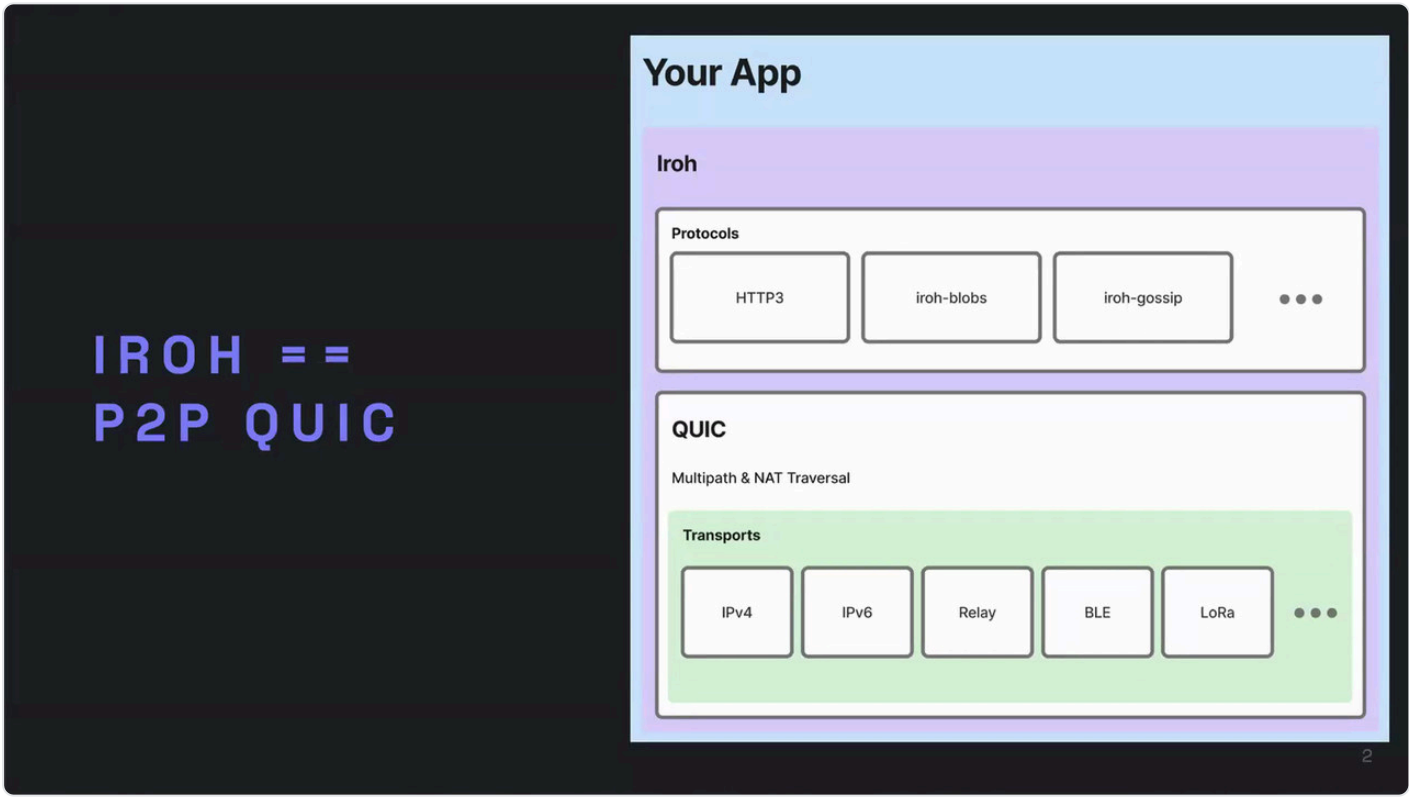
#[tokio::main]
async fn main() -> Result<()> {
    let endpoint: Endpoint = Endpoint::bind(preset: N0).await?;
    println!("my public key: {:?}", endpoint.id());

    tokio::signal::ctrl_c().await?;
    Ok(())
}

```

I'm going to show you a little bit of Rust code. This is going to be the same in lots of other languages, but I want you to get a feel for what this looks like. You do not need to write Rust to use iroh, but this is generally the look that you're going to get.

You're going to get some process where we import the iroh library, and declaring an endpoint basically says: I'm going to make an endpoint, I'm going to bind it with this thing we call a preset. And number 0 — the company — runs a set of public relays that you can use just to connect right out of the box. So this code will work as is. No accounts, no requirements, no anything. At this point, when this function returns, you will have an endpoint that has an identifier that can be dialed. That one line of code is doing all of that process that we saw above. And then in here we're just waiting on exit. So this is just an open endpoint that's waiting and can't really do anything, but this shows you how powerful this library is. It's a very small and fast abstraction.



So that's one endpoint that binds. We import that preset.

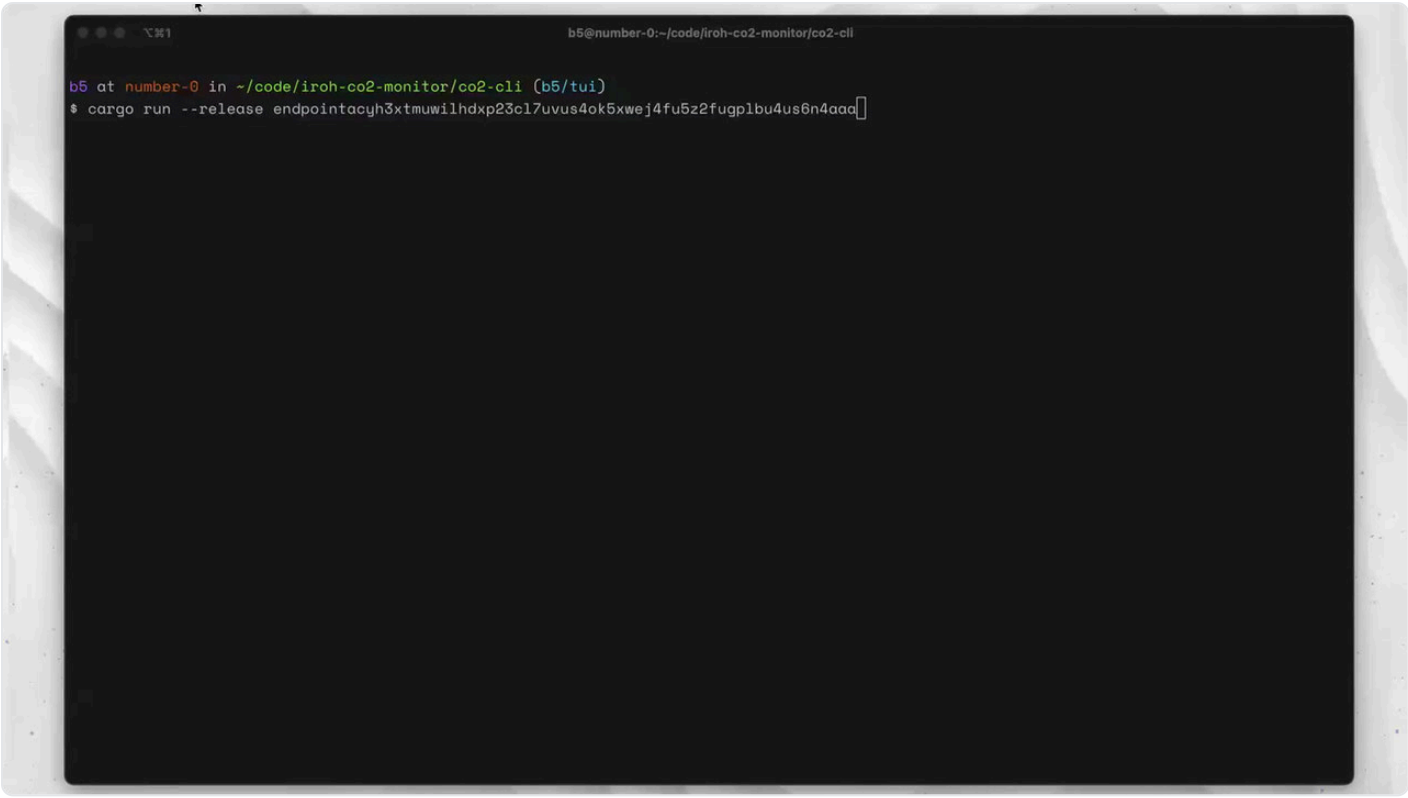
We kind of like to think about iroh as peer-to-peer on top of the QUIC protocol. I don't know if you've ever heard of QUIC, but QUIC is a technology developed by a number of major companies, and currently HTTP/3 is specced on it. A significant portion of internet traffic today runs over QUIC. QUIC is intended to be a replacement for TCP, and we have large companies like Google and Amazon contributing to the QUIC spec.

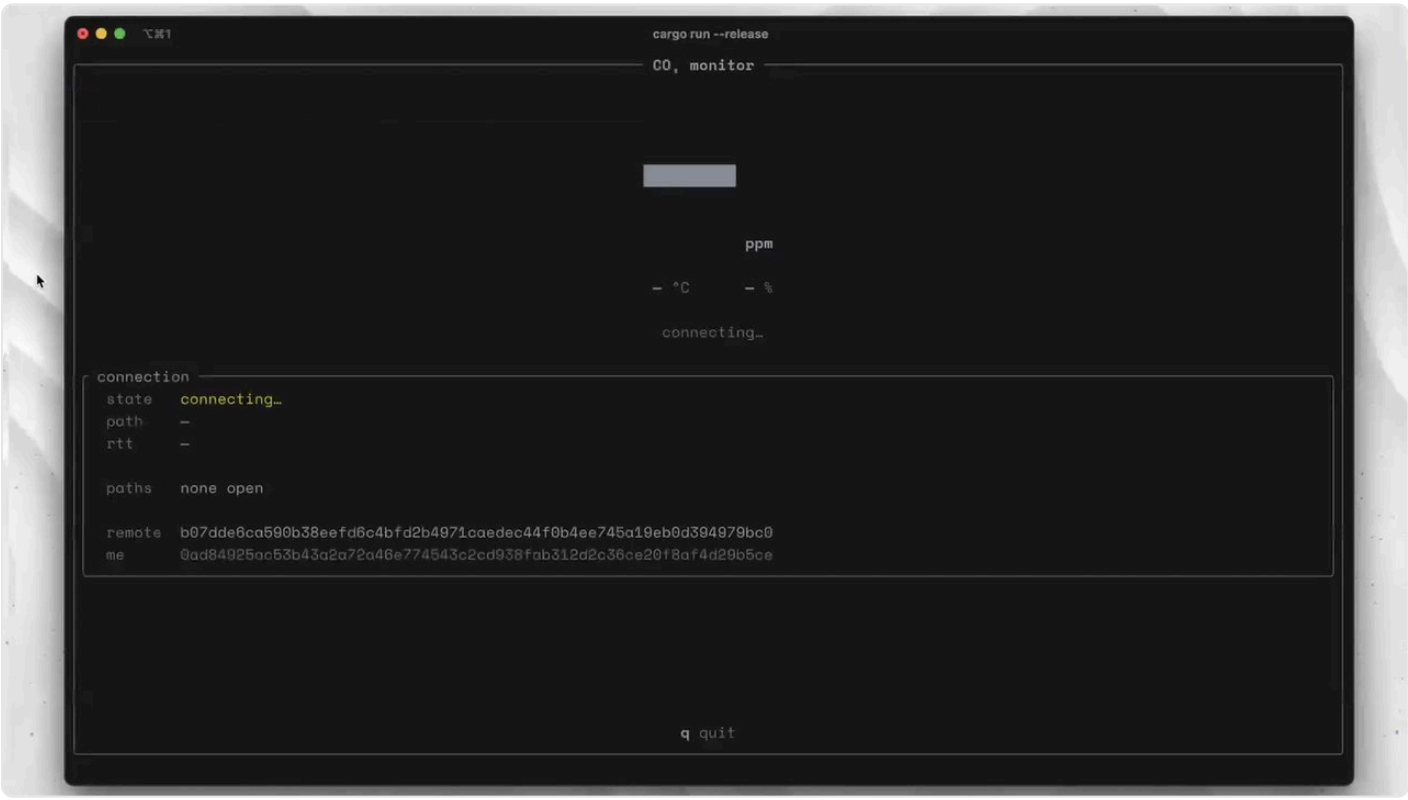
We like to think about iroh as being this kind of shape, where we give you protocols on top of QUIC — some of which is HTTP/3, but we also have all kinds of really interesting protocols that the community has built. iroh-blobs is one, for transferring blob data: if you wanted to move files around, this is kind of like iroh's version of S3. iroh-gossip is great for broadcasting messages: if you want to build swarms of endpoints and say, I want to broadcast one message to all of them from any place, that's iroh-gossip.

And then inside of QUIC, we have this abstraction over transports. iroh actually supports the idea of pluggable transports. Out of the box, it comes with the ability to connect over IPv4, IPv6, and the relay connection. You can also add stuff like Bluetooth Low Energy. You can add stuff like long-range Wi-Fi. All of these are possible extensions that you can plug into iroh, and it will then use those as part of that transport switching process, just picking the best one at any given moment.

This is really important for local-first — this is kind of the first place where you see local-first jump in. The difference between iroh and the regular internet stack is that it is designed from the ground up to not need a wide-area network connection. Everything you're seeing here, we can use mDNS discovery to discover new peers over local connections, and that can establish connections between two devices without any connection to a relay at all. The whole process is from the ground up built around this use case of: we may not have access to the wide-area internet, and we still want to communicate with each other.

And that process is dynamic, in the sense that if partway through you lose your wide-area network access, you can continue that conversation with the direct connection that you already have. But then partway through, if you restore that connection, you would just magically be able to talk to peers on the wider internet again. That would come and go as conditions of the internet change. We think that's a really important characteristic.

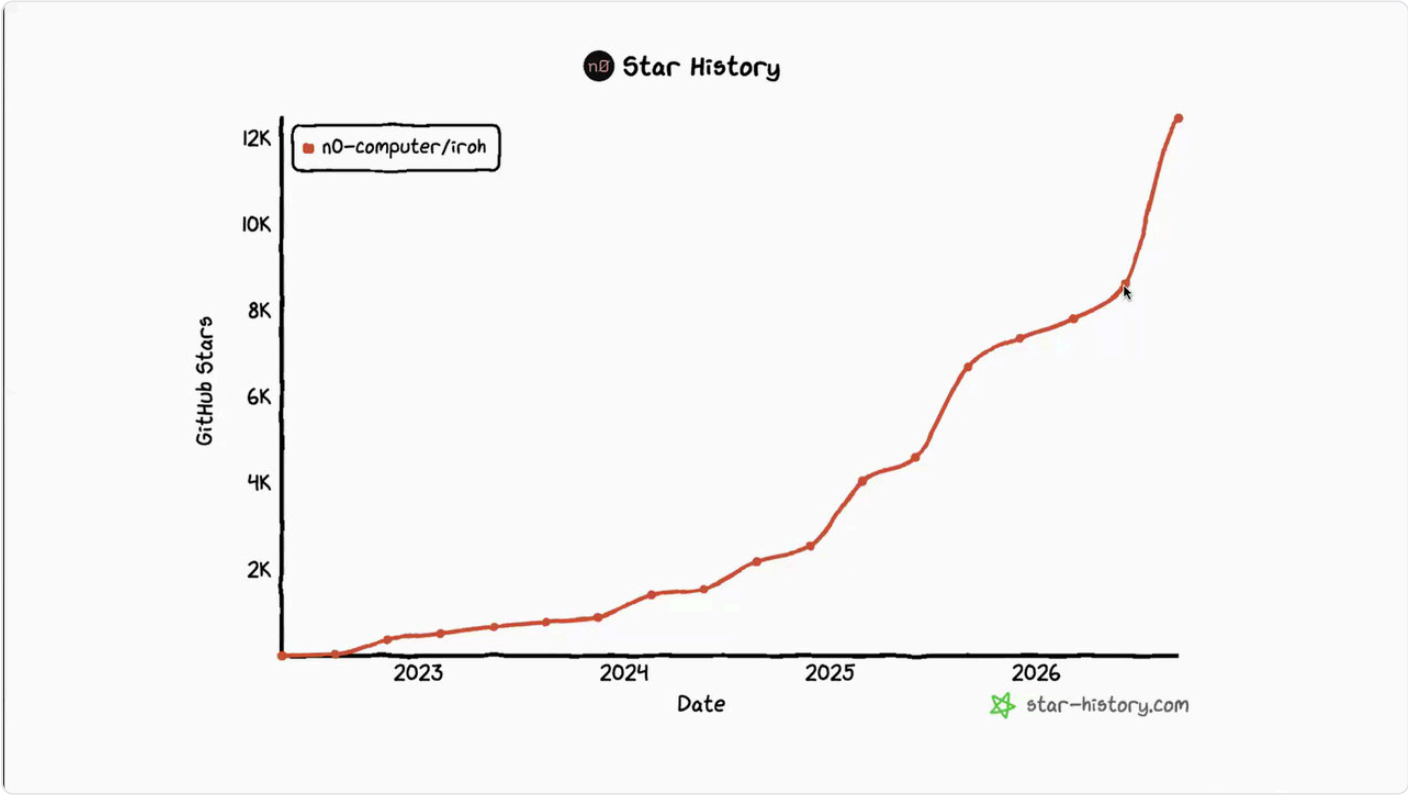




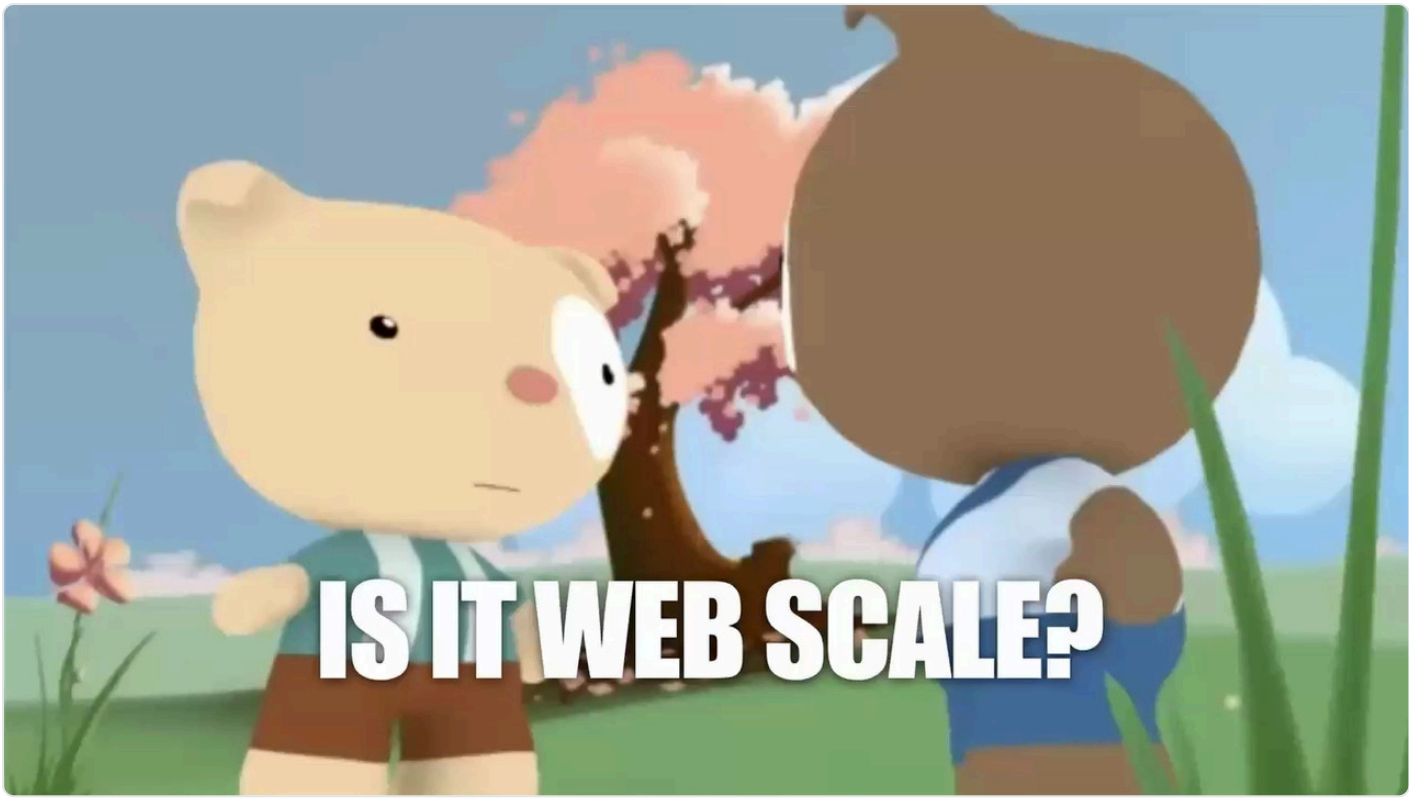
I was going to show a demo here, but instead I'm going to show you just an example of connecting to a CO2 monitor, which is actually an ESP32 that only samples once every two minutes.

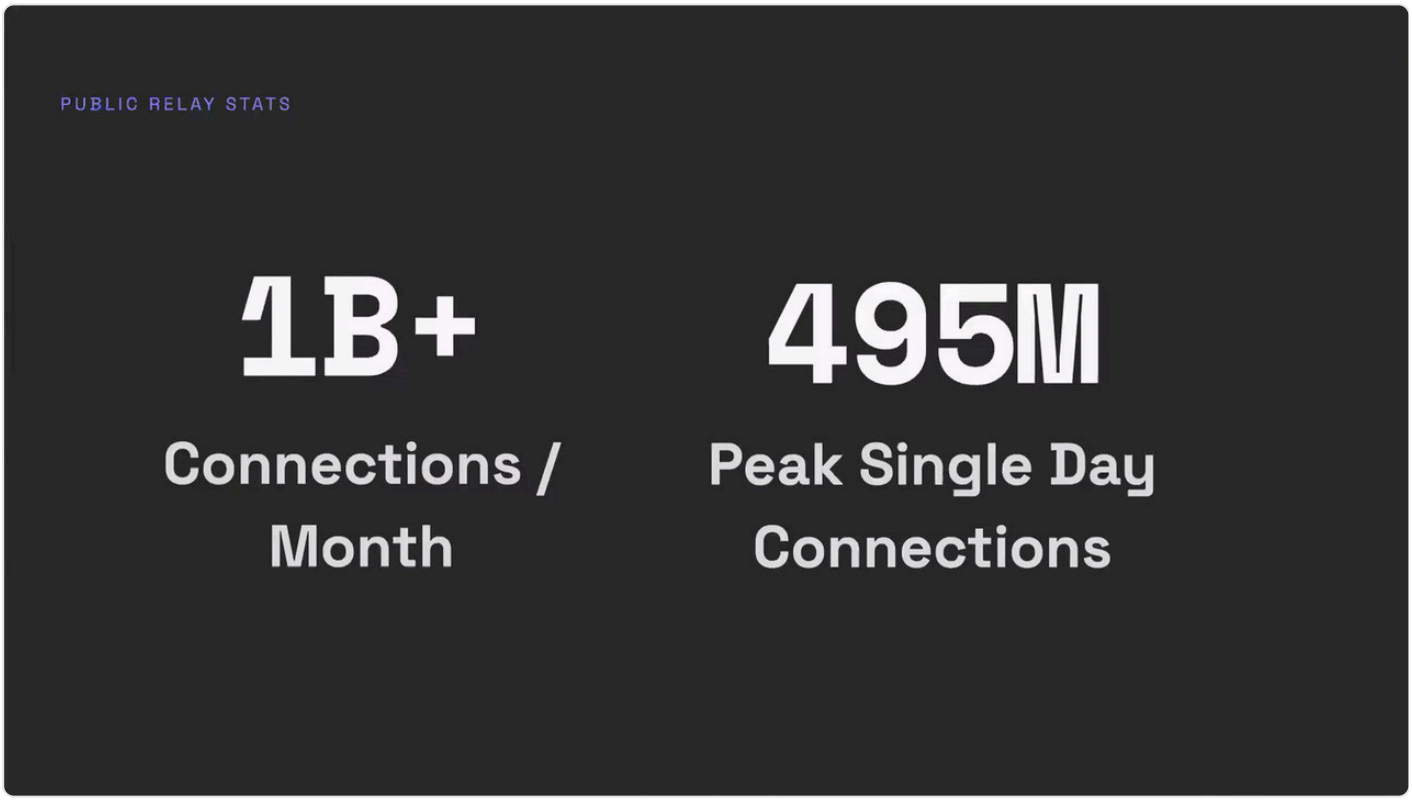


This is what it looks like when you build CLIs and binaries that connect directly between iroh and other things. We have some other examples that are a little more useful.



The project has been growing for a long time. This moment right here was the release of 1.0, and so we've had a nice spike in popularity.





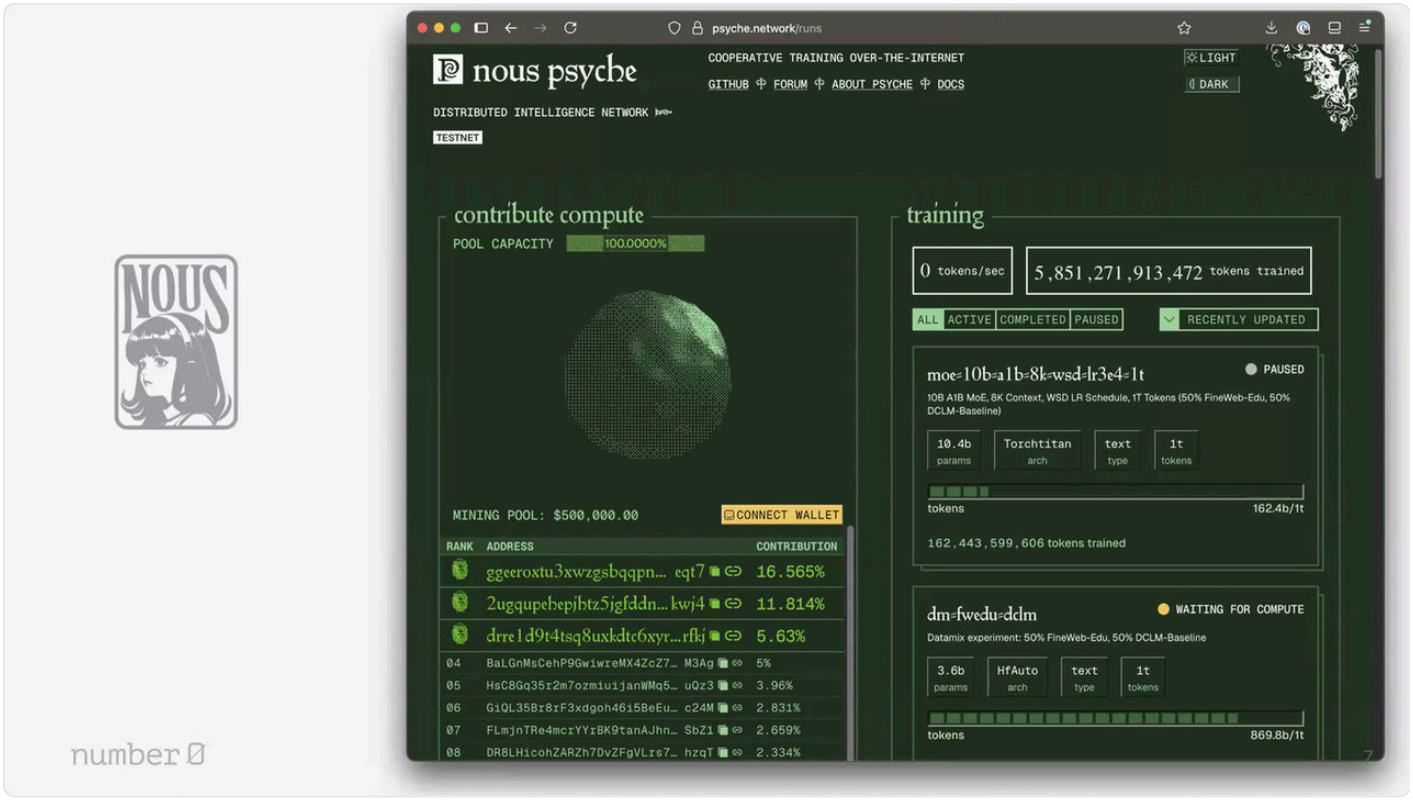
To give you a sense of the scale and use of this thing: we have been at a billion connections a month. On some days, we've seen almost half a billion connections on a single day. So the actual system itself, from the relays that we can see, does a lot. Again, we can't see any of the traffic, but we do get a sense of scale of use. The project is quite widely deployed at this point.



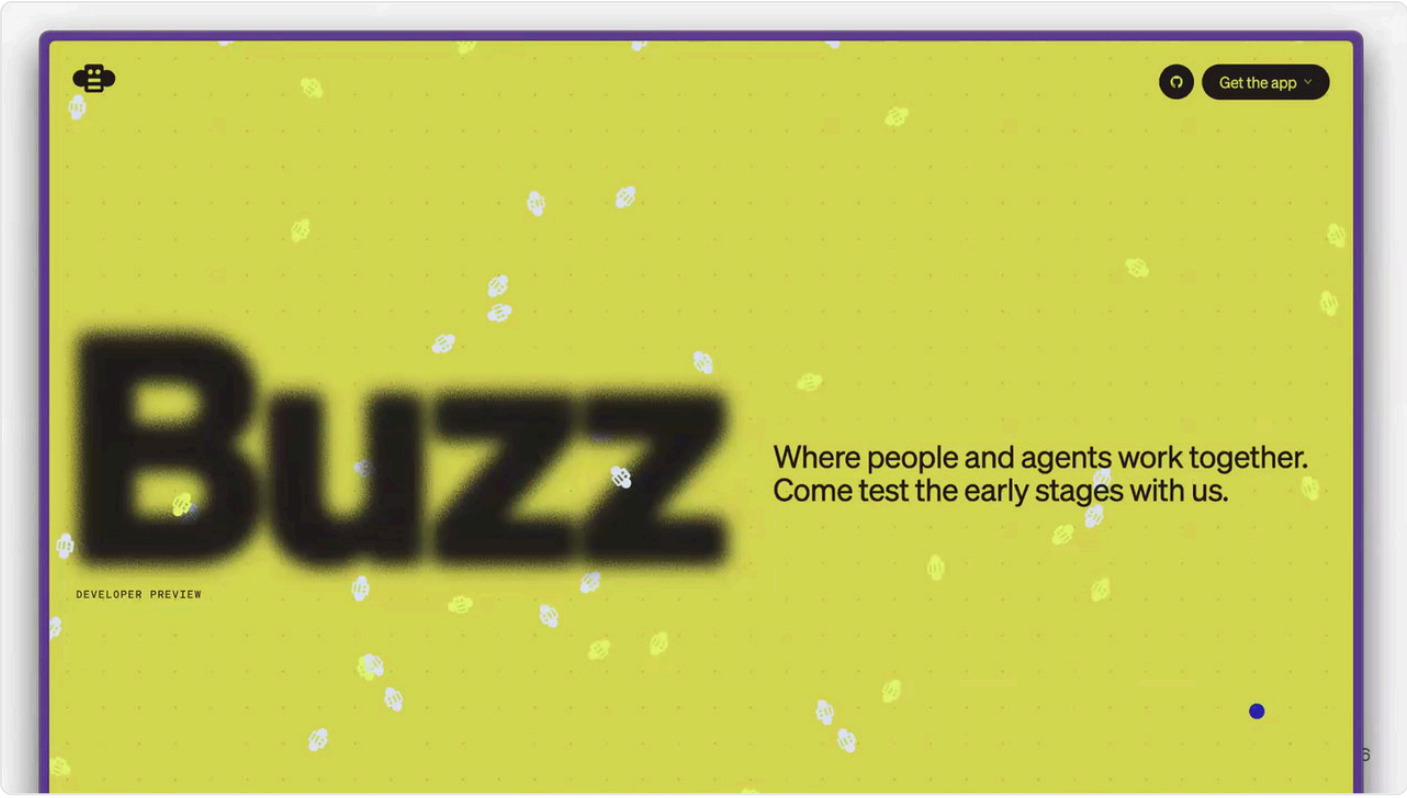
For the use of the actual library itself, we've seen around 2 million downloads from crates.io. For a number of the other package managers, I think we're in the hundreds of thousands for the JavaScript and Python ecosystems. And we have at least 700 projects building on top of this thing. There's lots of people filing bugs against it, there's lots of people using it.



It's used for all kinds of really interesting stuff.



We worked with Nous Research. They trained a base model, an LLM, on the broad internet using this. They actually wired together a number of H100s to do foundational model training, and to have compute clusters that could flex and join and leave over time, where each endpoint was an actual H100 GPU cluster talking to other GPU clusters. This is one of the most intense compute projects we ever worked on.



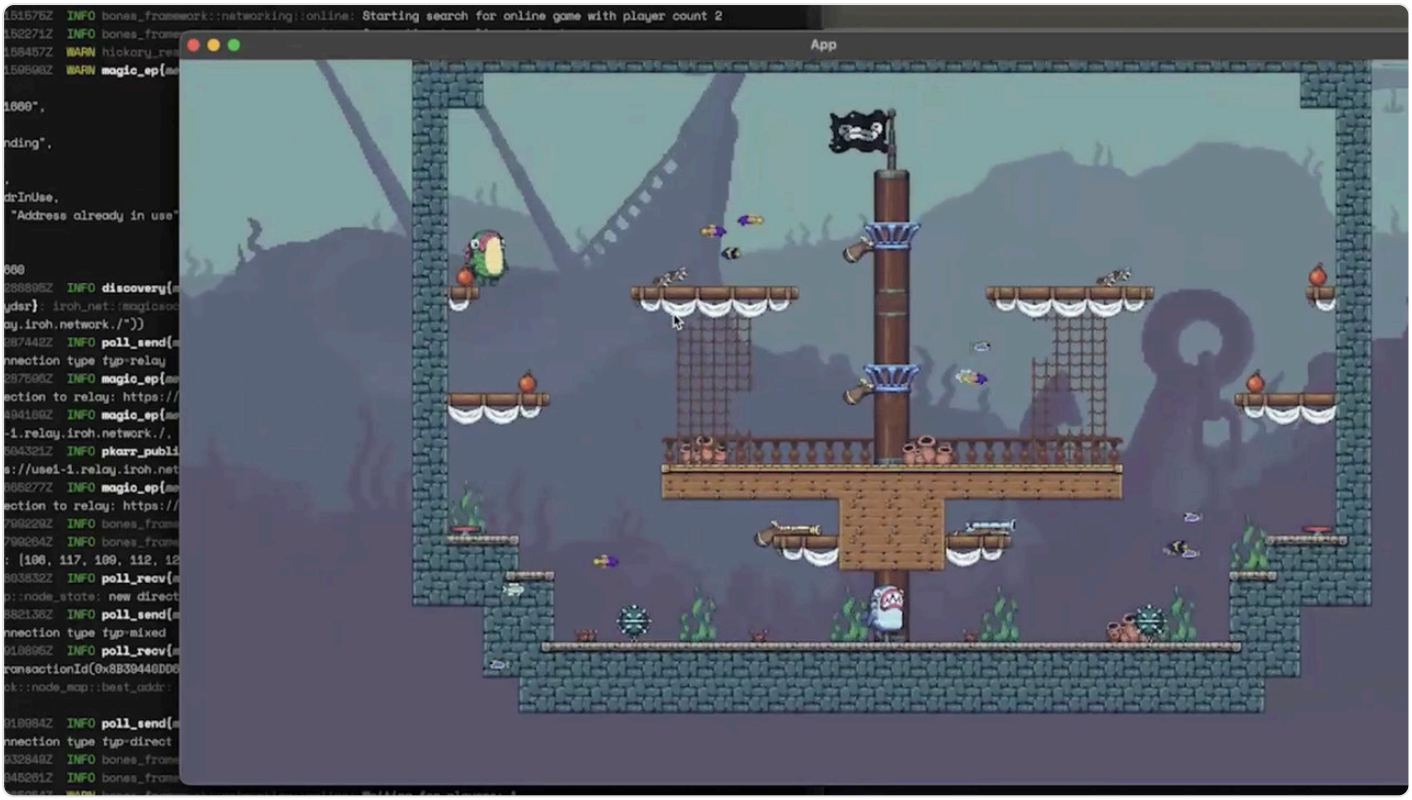
We've had Buzz, which is an agentic interaction framework — or, like, a chat app — built on top of iroh, and it uses that for communication between devices and inside of rooms.

This is an example of a video game being played where the actual rendering of the game is happening on the laptop, and the controller input is coming from the phone. We can get 60 FPS, sub-50-millisecond latency, where the actual app itself is written in C++. So we're doing C++ down into Rust, sending this over the wire, and able to communicate with this. We've done demonstrations of this working over 300 kilometers away.



So this process works. You can build very fast, very high-performance full video and audio streaming with iroh. That's a very common use case.

We can also build game engines. Here's an example of a game that is fully powered with iroh as the networking stack. The really cool property of this, if you come from the local-first world, is that you can make a game where the game server doesn't ever have to exist. Now the game can just continue to work even though the developers aren't running the game servers anymore. If you play video games, you know the tragedy of losing access to a game just because someone turned off the server.



That's no longer necessary. You can just have your LAN parties back.

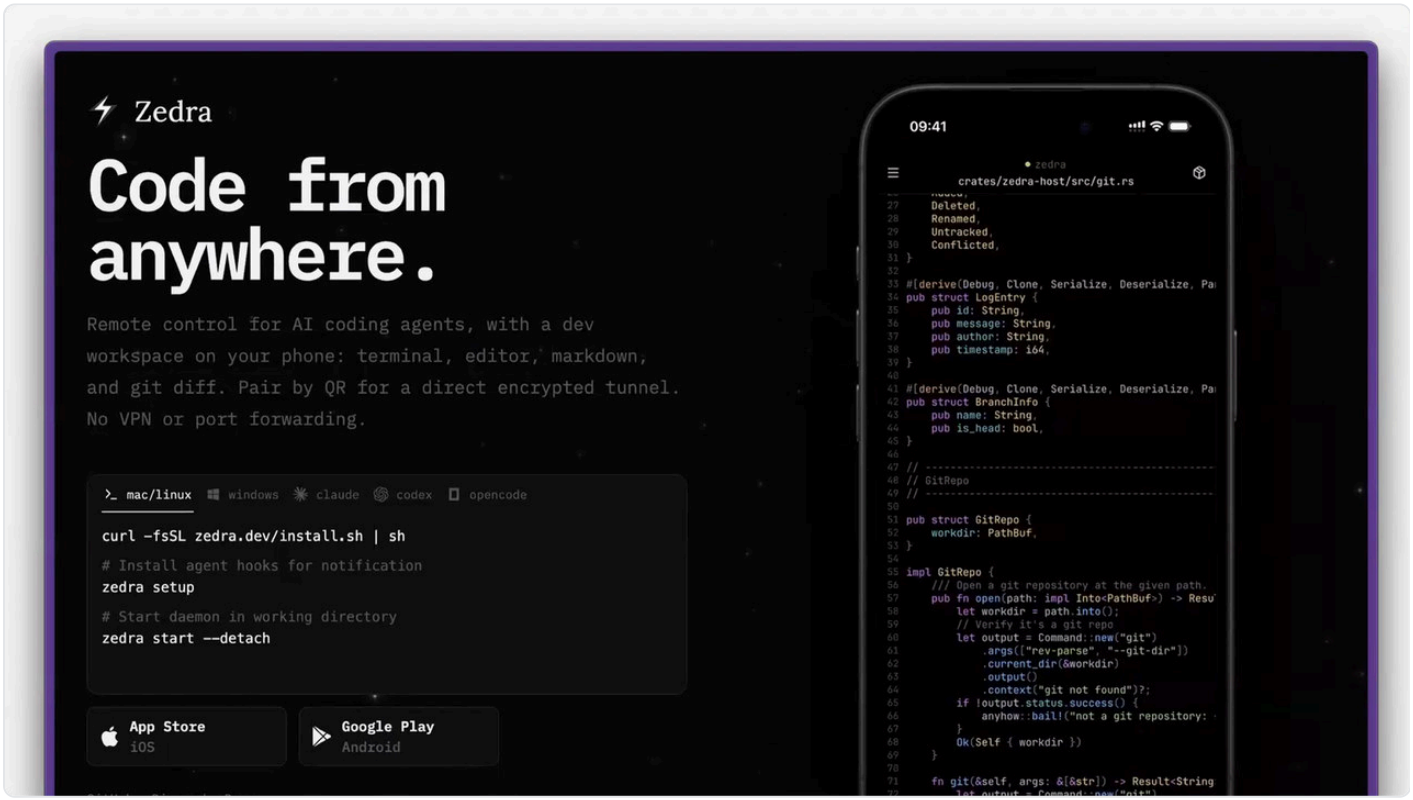
We've also been used a bunch in device backup. One of our first use cases was: how do I get my encrypted messages from one device to another?



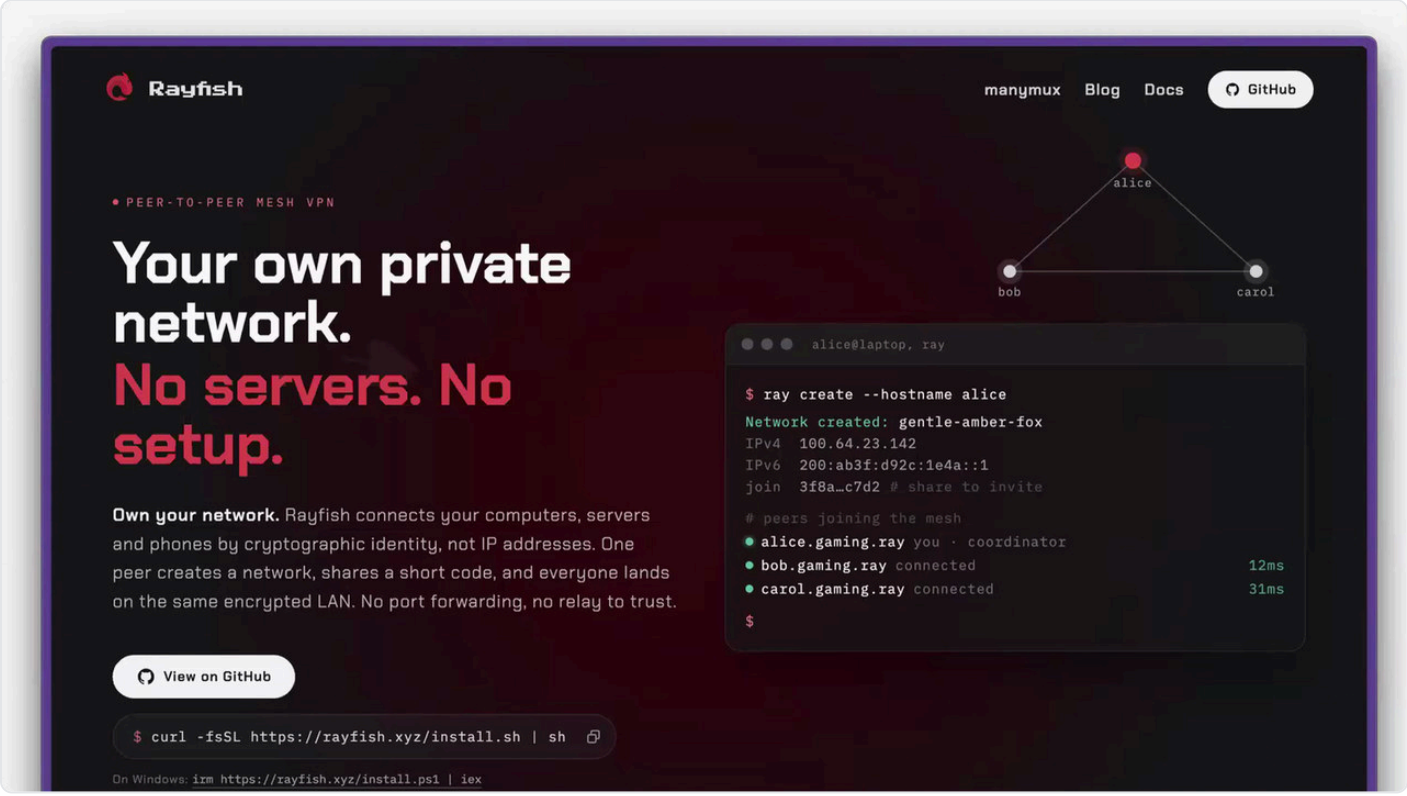
I just bought a new iPad, and I need to get all of my stuff off my iPhone.



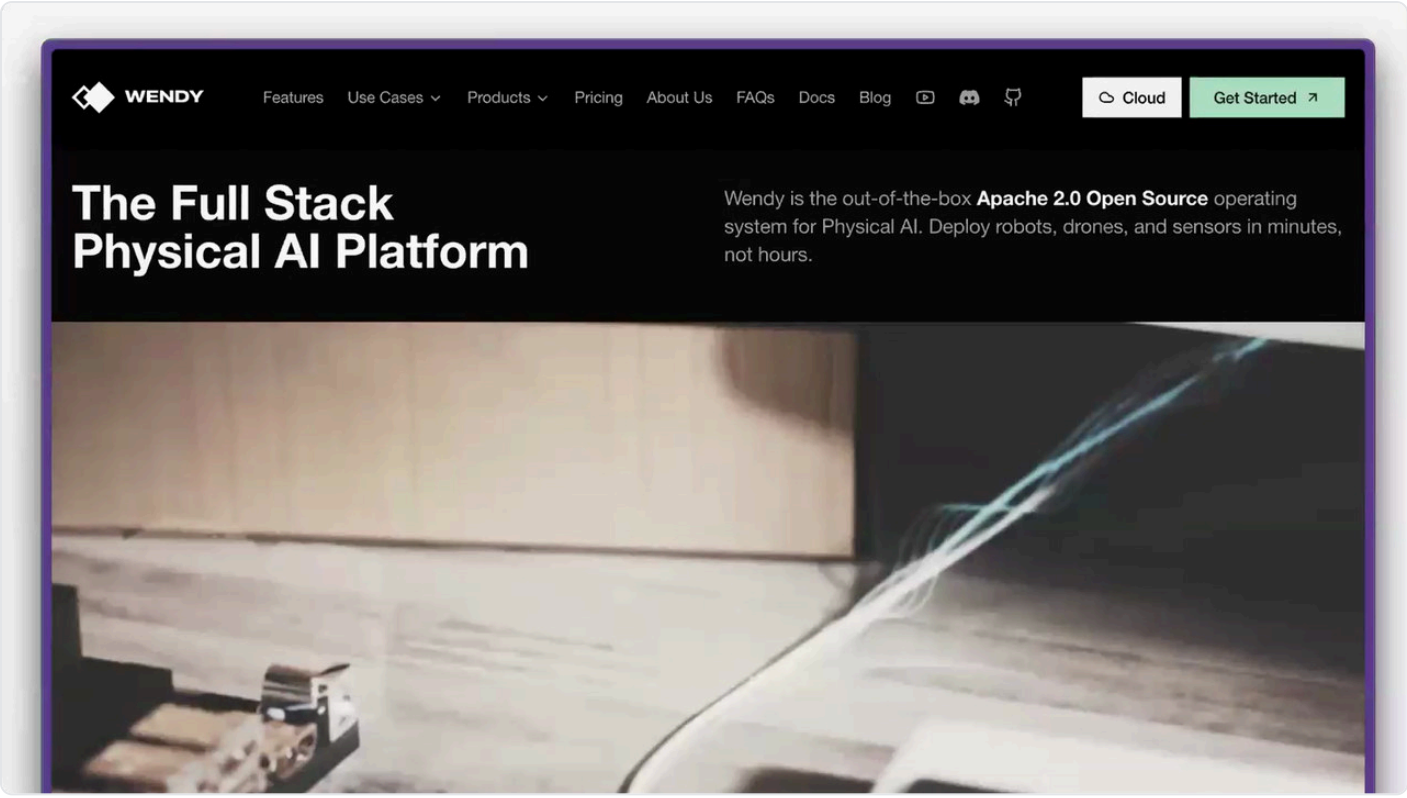
This has been a really magical method of getting direct encrypted connections between each other.



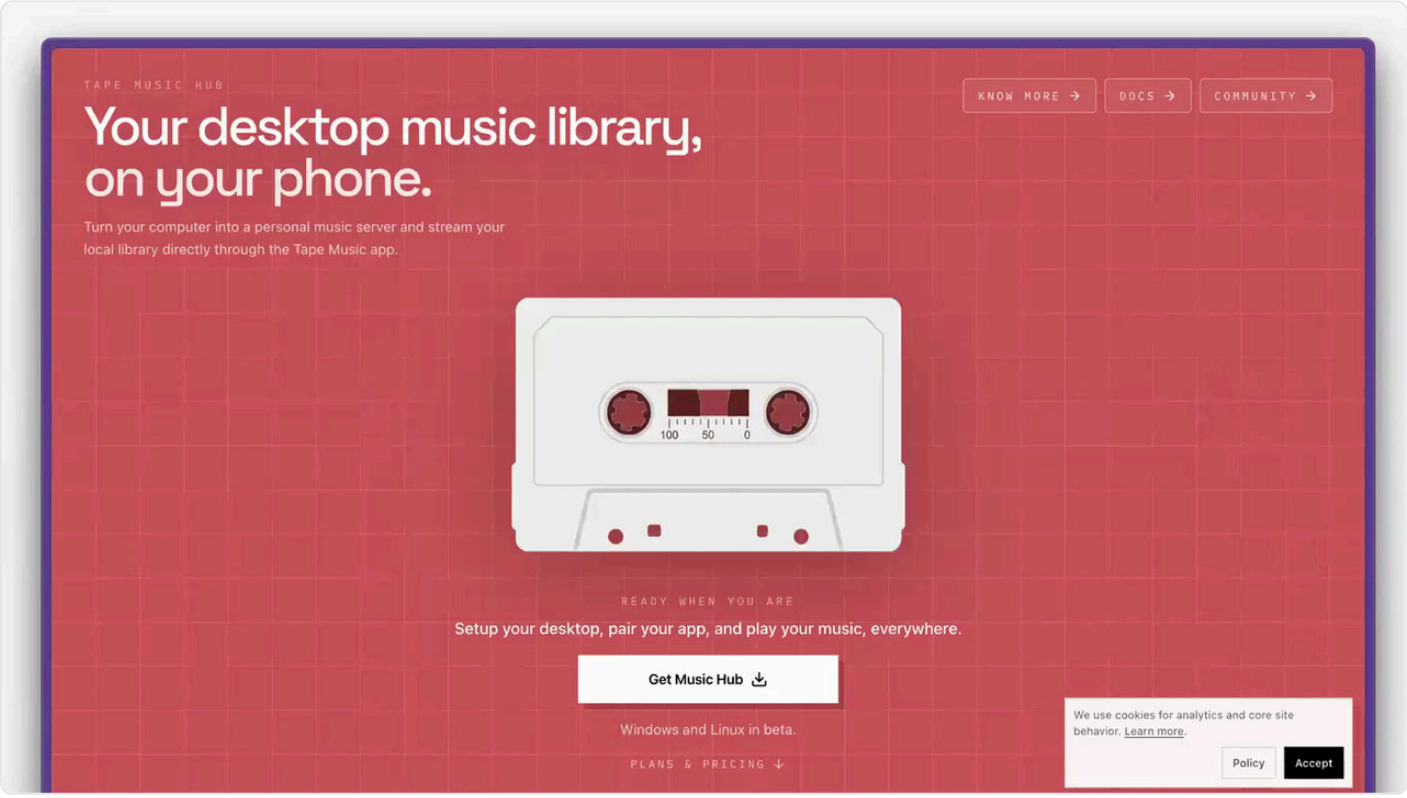
Code editors. This one has really started to take off: I want to be able to edit code from my phone that is running on my desktop, where you have a computer plugged in somewhere and you want to be able to talk to it. This has been a really powerful, fast-growing technique.



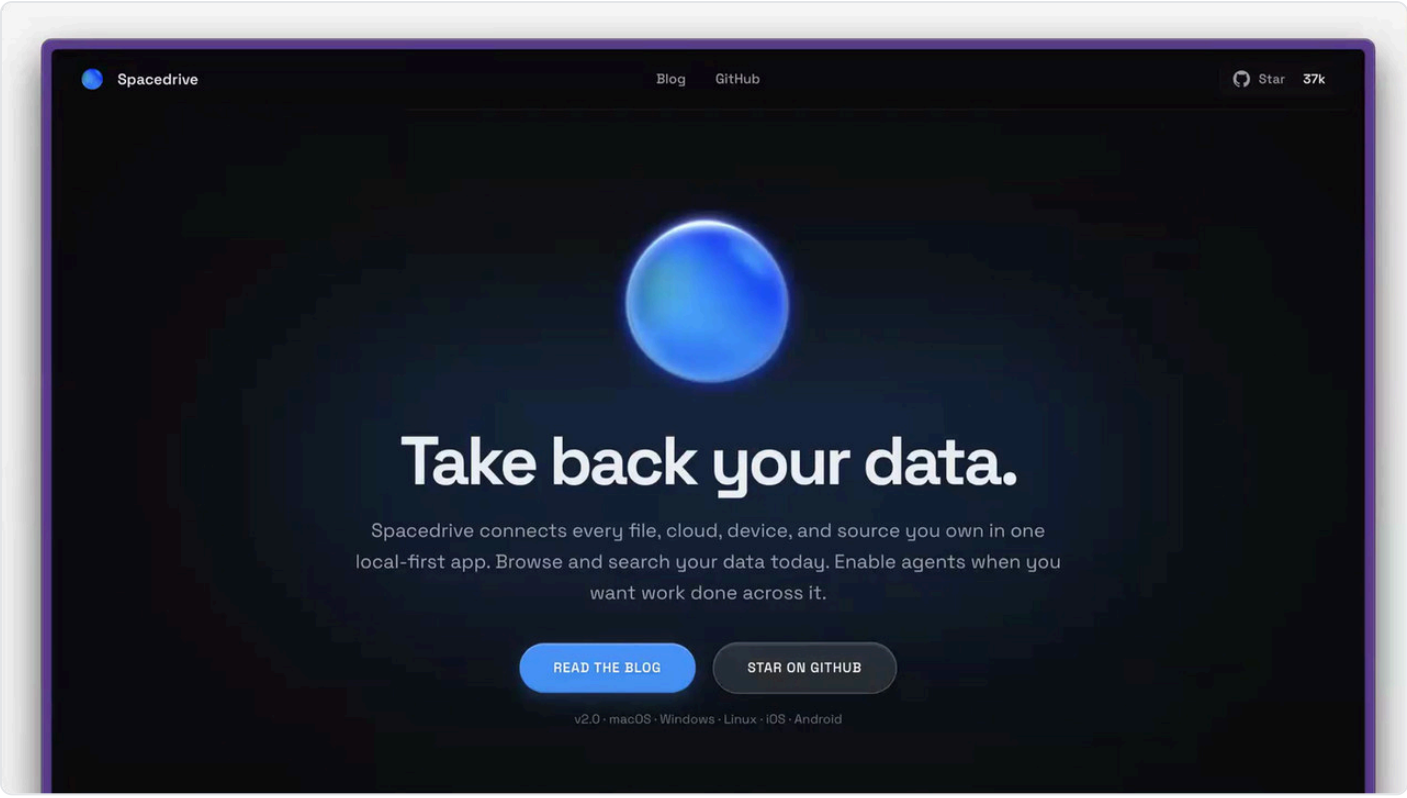
VPNs. We've seen a lot of people build VPNs where they say: I want to have a mesh network between a bunch of devices, and I want to be able to port forward everything and control that over time.



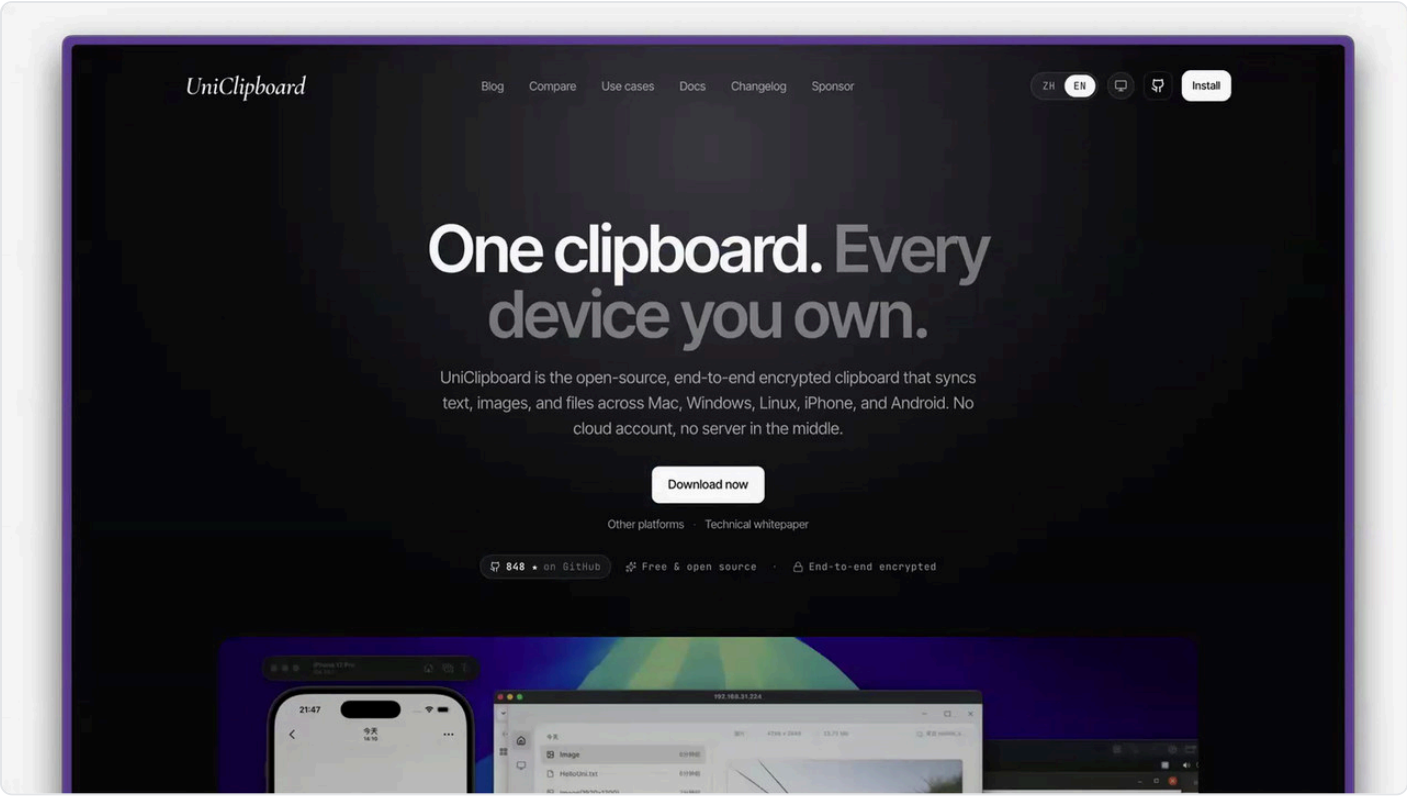
We've seen a lot of physical compute — robots use iroh a lot, where people want to have devices in their home that they're able to automate and talk to in safe and encrypted ways.



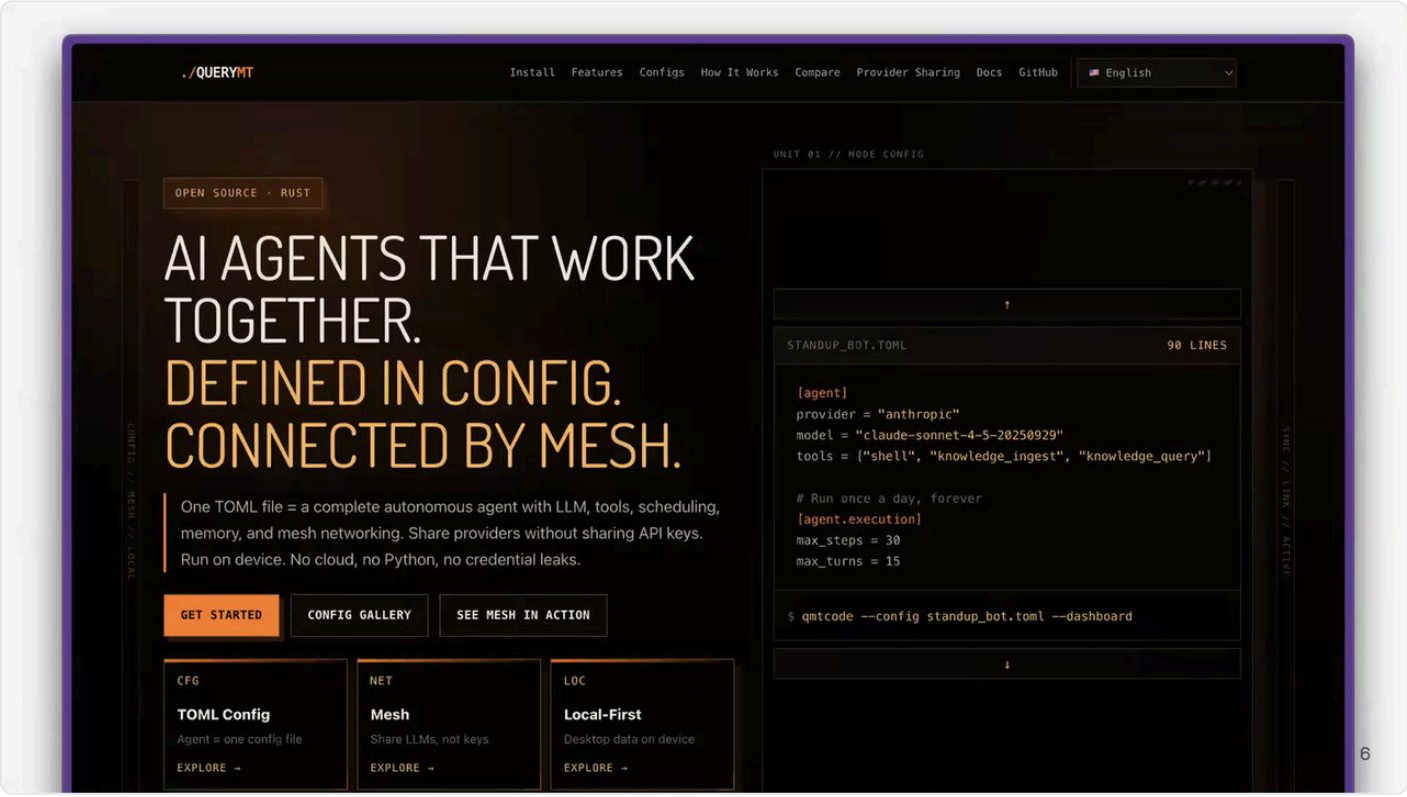
This is, I think, one of my personal favorites: I've always had a bunch of music sitting on my desktop, and so it's nice to be able to get that and talk to it from anywhere.



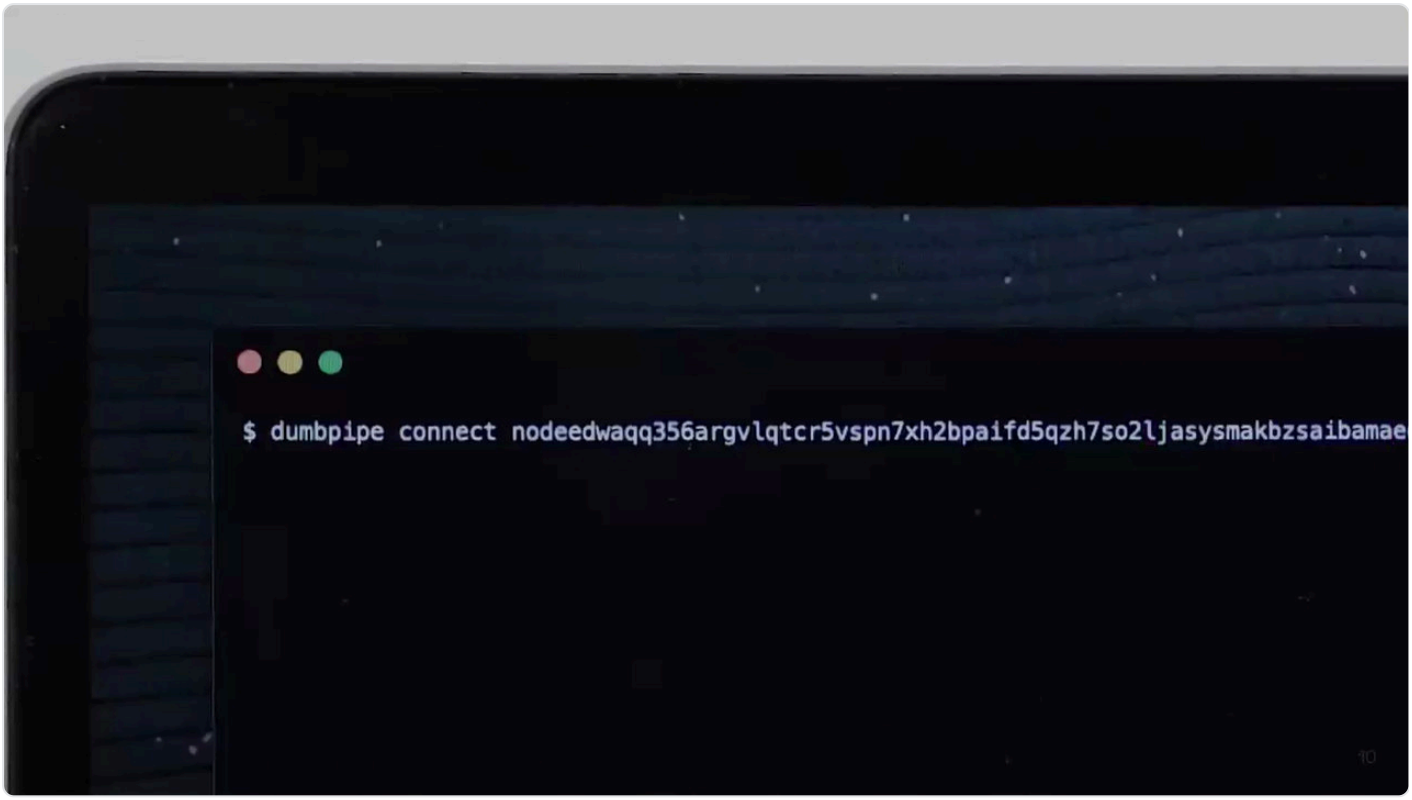
And then a lot of stuff that starts to get much closer to the local-first manifesto approach of: we want to repatriate ownership of data. This is where we start to plug in directly. If you take a local-first approach in principle, it's important to be able to say: no, actually, the data is going to originate on my device. And now with iroh, I have the capacity to network that in a way that I can control, because we can ship all of that networking capability directly to the device itself as part of the library. This is the whole idea — we think that you should be able to drive stuff like this.



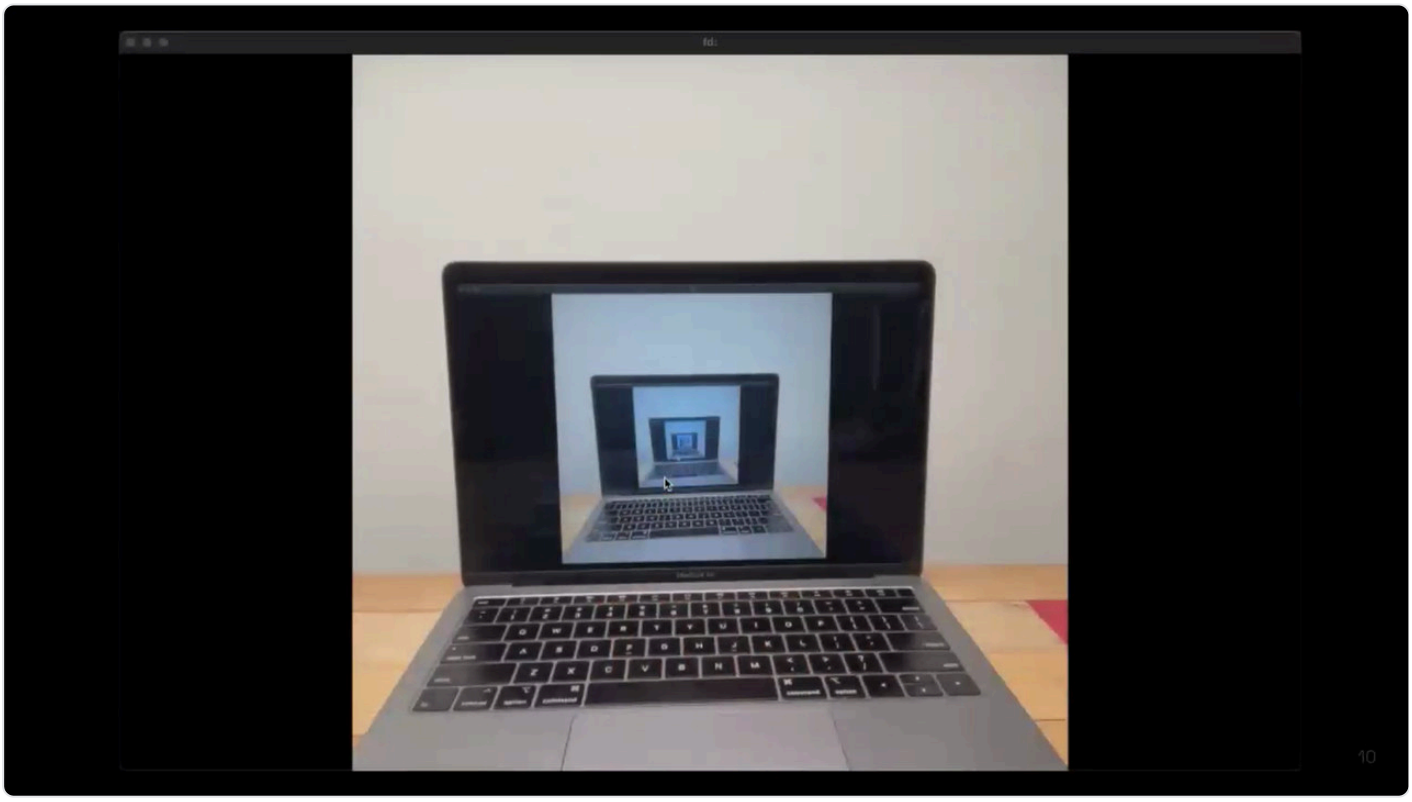
We also have some really great projects, like a unified clipboard that works across different platforms. So finally you can do Ctrl-C on your Mac and paste on your Android device, and that could work.



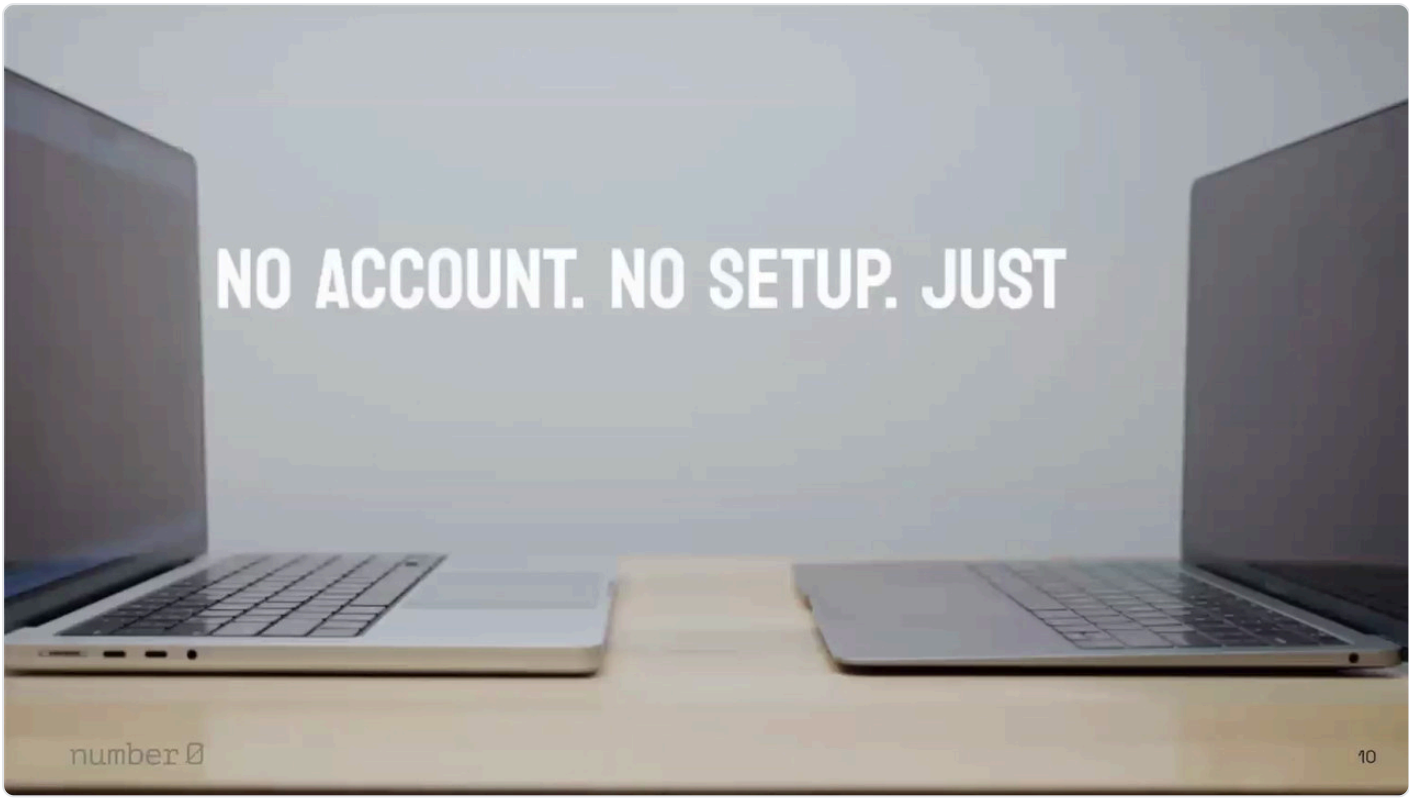
Lots of multi-agent configuration and agentic orchestration, where you can do mesh networking for agents and agents executing each other.

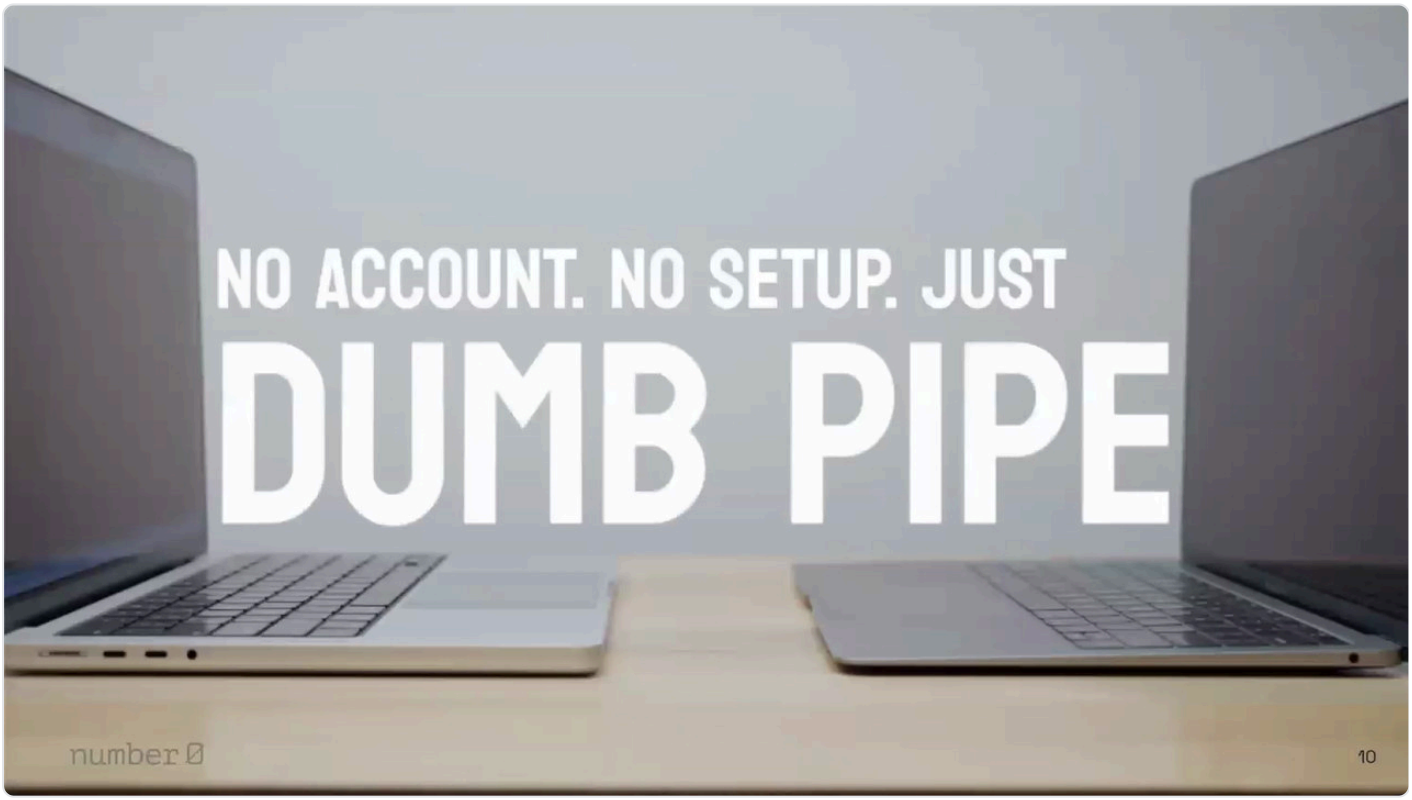


And then some of the other ones that we've built, just to demonstrate this process.



Dumbpipe is one that we've built a bunch. This is just literally a Unix pipe between two devices, and we wired it up to FFmpeg so you can just have live streaming.





And again, you don't have any accounts. It just wires up and connects.



And this is the core thing that we want to say to most folks. In the local-first world, we have long believed that the shortcomings of the networking stack have really been holding the entire community back. If you aren't able to have a really reliable, working peer-to-peer networking library that is intended to support the local-first use case, we've kind of not been able to get to some of the more exciting places. And we think that those more exciting places are where you can build an app that realizes some of these local-first ideals. This, we think, is where iroh is really a lovely match made in heaven for working with local-first ideas, because you now have a networking stack that hopefully delivers on some of the things that you actually need to be able to make a proper, full-fledged local-first app.

So we hope that you will demand more of your networking stack. We've seen a lot of uptake of iroh in the local-first space, and we're really excited — it's honestly some of the most exciting stuff that we see. A lot of the folks building in this world really understand the necessity of advancing the state of the art and building from a different set of principles. To not be afraid of CRDTs — a lot of folks need to find their way to that level of ambition. iroh is a project that hopefully matches that ambition, where you can actually say: no, we do need a networking stack that can do this, that can work anywhere, that can communicate naturally and seamlessly, and that can let us paper over what transports we're using and give us full control over the capacity to design protocols that express what we need them to.



That's kind of it. The website is iroh.computer, and the repo is on GitHub. Any questions?

Q&A

Host · 16:37

Thank you so much — that was amazing. 大家如果有问题的话，现在就可以提出来。For our audience, you can raise any questions now. 也可以把你的中文问题放在聊天区，或者我可以帮你把它念出来。

Audience · 17:05

I have a question. Should I just ask directly?

Host · 17:08

Yeah, yes, please.

Audience · 17:11

Thanks, Brendan, for the amazing presentation. I have a question regarding the history or positioning of iroh. I first saw it, I think, two years ago, and I thought it was an implementation of an IPFS node in Rust. Then it kind of turned into its own thing, but it was based on libp2p. Now I feel like its positioning is almost like a replacement, or a better version, of libp2p. Am I understanding that correctly? I guess the question behind it is: what's the vision or positioning of iroh? Does it want to become what libp2p used to want to become — the infrastructure layer, the unifying protocol on top of whatever other protocols happen to be at the transport layer? Is that kind of what it wants to be?

Brendan · 18:05

Yeah, I think that's right. We're a bunch of ex-IPFS devs, so I'm going to post the link in chat. As you said, we started off — or did actually begin our journey — as an implementation of IPFS. If you want the full long story, I actually did recently tell it in that YouTube video, so you can watch if you have any interest.

But yes, I think that is the right answer, with one distinction. The one big difference between iroh and libp2p is that it's intended to be a very internet-standards-driven project. We really want to be in a place where we can say: 90, 95 percent of iroh is just specs from the Internet Engineering Task Force, and not us inventing new ideas. We're trying really hard to think like software carpenters who just make something that really works, because we want to position iroh as an internet technology. We're hoping that one day the destiny of this thing is to say: a lot of stuff runs this, let's just land it with a standards body and have that be a piece of infrastructure that we can all rely on. It's very interesting to build that out.

I think libp2p had a lot of those ambitions, but was more focused on decentralization as the core animating purpose. iroh is very highly distributed, and it prefers decentralization over centralization whenever possible. But we have swapped that for the goal of being a pragmatic, internet-first technology. We will not make changes to iroh that will slow down the time to first byte. It needs to be competitive with regular web technology. And we're willing to make ideological compromises to make that happen. I think that's a core difference in the project — still open source, still so much shared DNA, but for us, 100 percent of possible connections need to work. If the device is air-gapped, yeah, we can't do anything. But it should be expected that at any moment, if you can get a connection to that other device, iroh is going to do it for you, and it's going to keep it consistently working. If we

have to use a relay, then we have to use a relay. That's not extremely decentralized — that's a distributed system, which is a different set of guarantees. So it's really, really close; this is very hairsplitting. They're very much in the same large vision of: we can improve the internet, we can demand more of our networking stacks.

Audience · 20:31

I think everyone was kind of sad to see IPFS being deserted by Protocol Labs, as well as libp2p, but I feel like iroh is the best spiritual successor to that. I've always been curious about what lessons can be learned from libp2p, and why it doesn't work. I guess you kind of answered most of that question — it's the pragmatic side, that we sometimes have to make compromises in order to make the system work, and also, as you said, the internet spec: instead of deviating from the Internet Engineering Task Force, you follow it more closely. Do you think that's the most important lesson that we can learn from libp2p?

Brendan · 21:13

There's one more that I would add. Two things I would say. First, there would be no iroh without libp2p and IPFS, period. I think they showed that there was demand for this and that there was interest in this. We came from that world, we love that world, and we're also sad to see work on libp2p slow down — but I don't think it's totally over. There's lots of projects still building on it.

The one thing that we really did deliberately change about iroh is that we heavily believe iroh should be self-configuring whenever possible. One of the things that we saw as a real point of challenge when you're building with libp2p is the configuration matrix of what you can possibly do. You can choose your encryption layer — you could use Noise or TLS. You could choose your transport, your networking layer — you could use QUIC, or you could use TCP. All of that optionality at first seems really good, until you realize that there are certain configurations that, if held incorrectly, can actually become security nightmares. You can get into a place where, oh, actually, if you combine Noise with a TCP socket and advertise this specific thing, certain combinations really aren't giving you what you think — and particularly if you choose an encryption cipher that isn't well complemented to that, all of a sudden you can end up in a place where a cryptographer might say: no, don't do that. And you would never really know.

So with iroh, we really try to take away a lot of those footguns by making iroh self-configuring. By self-configuring, we mean we expect iroh, the software, to understand whether your router supports UPnP — which is a low-level technology that you probably shouldn't have to care about, but iroh knows about it. iroh should know when you're on an Android device and your network conditions change, and it should react to that and change itself to match those changes and differences.

That's kind of a fundamental departure. I think libp2p is designed to let you have full control over all of those things, and iroh is trying to tell you: you know what, you probably just want really robust end-to-end encrypted connections, you want them to survive at all times, and you want the best one that you can have. We're going to make that happen for you. So it's a little less — we're increasingly being asked to add more knobs to iroh to let people reach into that, and as the project evolves, I think that's where we're gaining back a lot of that control. But that control is coming through this kind of engine we've built that makes iroh as self-configuring as possible. When iroh is running in a data center, it knows that it's in a data-center context — it doesn't "know," but we have if-statements that handle that branch. That means certain things will start to flip on and off, and we make the whole thing work a little bit better.

We think this is really critical from a performance perspective, from a security perspective, and from giving you back mindshare as a developer. If you can stop worrying about this stuff and just say: okay, iroh gives me connections, it works, and I dial keys — and then my job is to figure out what protocols I want and what I want to talk about — we think that gives you back your innovation budget and lets you focus on building your app. We think that's really important, because we spent a lot of time holding libp2p. So we really wanted to make sure that we made iroh very self-configuring. I don't know if that helps.

Audience · 24:30

That's a great answer. Thank you, Brendan.

Brendan · 24:33

Yeah, of course.

Host · 24:46

Any other questions? 有其他的问题吗？大家可以提出。There's a question — he's asking, is it possible… the conflict between public keys, encrypted?

Brendan · 25:27

Could they collide? I think that's what you're asking. Is that right?

Host · 25:31

Yes, I think so.

Brendan · 25:33

Yes. So the key space is so incredibly large that the probability of a collision is vanishingly small. It is theoretically possible, but it's so incredibly improbable that we don't really worry about it on a day-to-day basis. We do have on the order of billions of keys created all the time, and the key space of the Edwards-curve key which we use — we use Ed25519 keys — is very, very, very large. It's intended for the idea of machines just continually running in tight loops and never hitting collisions. So yes, I don't think that you need to worry about a collision. That would be the short answer. Hopefully that puts your mind at ease. I don't worry about it; I have never worried about it.

Audience · 26:21

If there's an adversary attack, can the user specify the key that he wants to use to create a tunnel? And when that happens, can it conflict with the original key?

Brendan · 26:39

You are right to ask this question. I think the question is getting at this idea of: where does the security of iroh emerge from? Not sharing the key is the origin of the entire security model. When iroh starts up, by default — those code examples I showed you — it actually just creates a random key out of nowhere and only stores it in memory. It doesn't put it anywhere. That's on purpose, to prevent exactly this type of idea.

If you somehow gain access to somebody's key, you can impersonate them, and you can direct traffic towards yourself. That's real. The whole system does depend on that, and this is true of all cryptographic systems: if the private key is shared, it is equivalent to having the account. So it is

worth keeping in mind that yes, if you were somehow able to get access to somebody's private key, that's a big problem, because you could basically be them in the eyes of iroh.

There are lots of ways that we deal with this on a day-to-day basis: key rotation, only using ephemeral keys. Ephemeral keys are basically keys you don't write down — when you start the app, you create a key, you use that for that connection, then you just throw it away. It's a really common pattern; it's actually the default in iroh. You have to store the key and reuse it if you want to have a persistent key.

But yes, it is true, it's very possible, and so it is worth making sure that you code against that correctly. But again, iroh tries really hard not to give you any footguns in this process. It is very difficult to construct a key that you don't have access to. It would be far faster to just try to get the key by some other means. Trying to guess the actual private side of a key is so incredibly computationally expensive — even with our modern understanding of quantum computing and stuff, we think we have at least a few years before this is a major problem.

I hope that answers your question. It is a good question, because thinking about security is a good idea. And it all boils down to: keep the private key private. Don't move it off the device, just leave it there. As long as you don't move it and you don't let it escape, things are good. SSH is built this way, a number of things are built this way. It's a very simple principle, but it is important to get right.

Brendan · 29:06

Cool. Anybody else? This has been super fun.

Host · 29:16

有其他的问题吗？如果不方便开麦的话，也可以放在聊天区。

Brendan · 29:49

Okay, well, then maybe we can leave it at that. If anyone has any other questions, the best place is iroh.computer/discord. We are all there, we're hanging out, happy to chat. And we'd love to see what you build. If you build something, jump in that Discord and post it — or we have an awesome list here, awesome-iroh. We'd love to see what you make.

Host · 30:22

Thank you so much.

Brendan · 30:26

Thank you so much for inviting me, it's really appreciated. And thank you for your time, everybody. Hopefully this is useful. Looking forward to seeing you all on the internet.

Audience · 30:35

Thank you, Brendan. It's great.

Brendan · 30:38

Thanks a lot. I'm going to log off now. Thanks, bye.

Host · 30:41

Have a nice rest of the day. Bye.