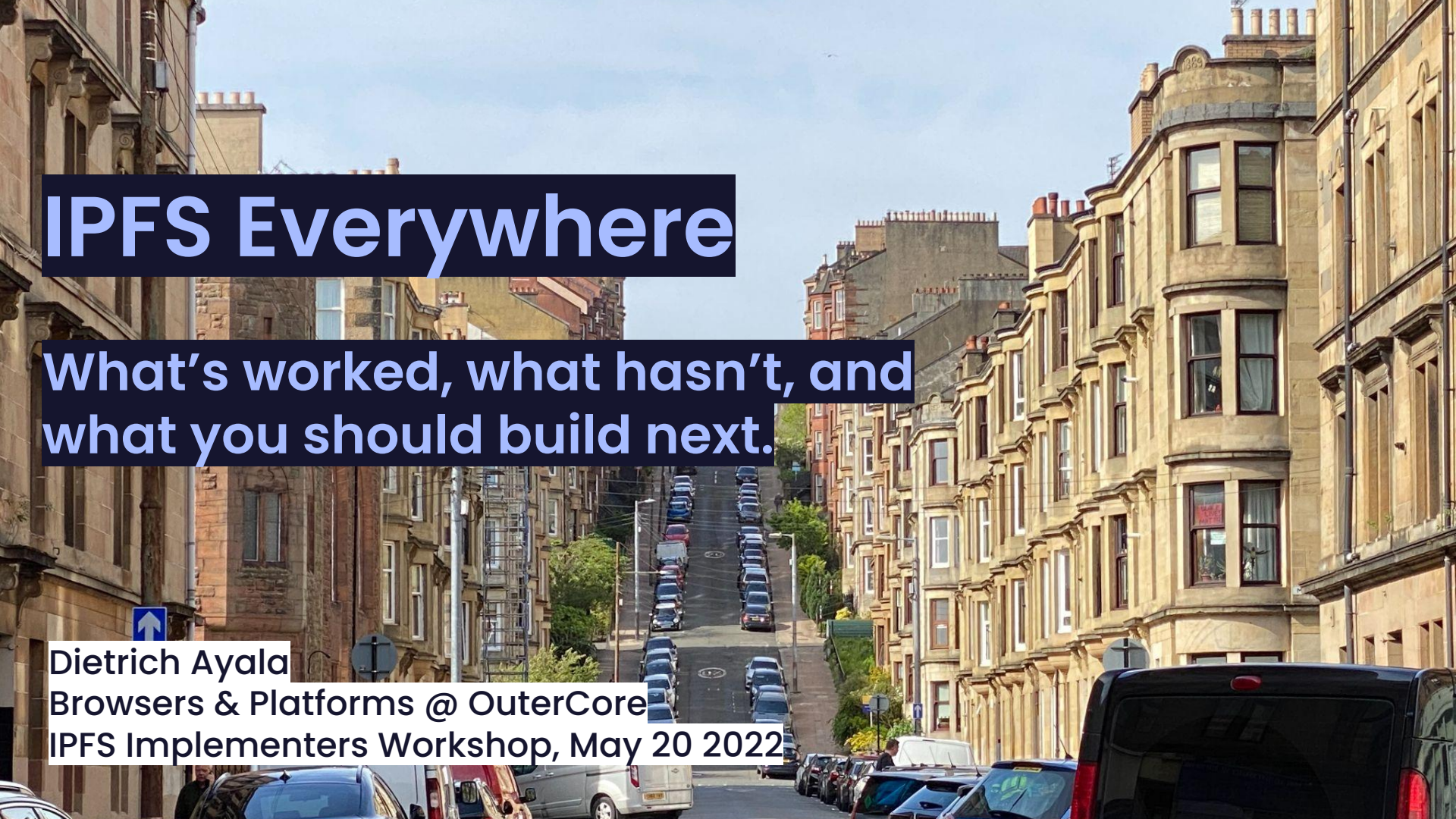


IPFS Everywhere

What's worked, what hasn't, and what you should build next.

Dietrich Ayala
Browsers & Platforms @ OuterCore
IPFS Implementers Workshop, May 20 2022

* dietrich ayala

Browsers & Platforms team lead

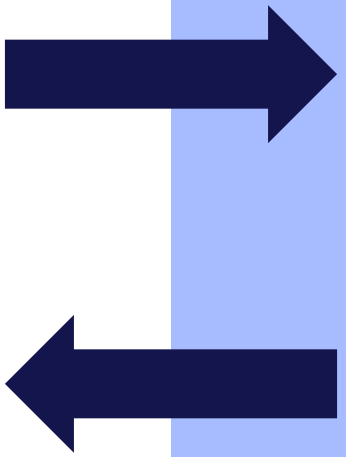
Outercore / Ecosystem / Protocol Labs

browsers & platforms team

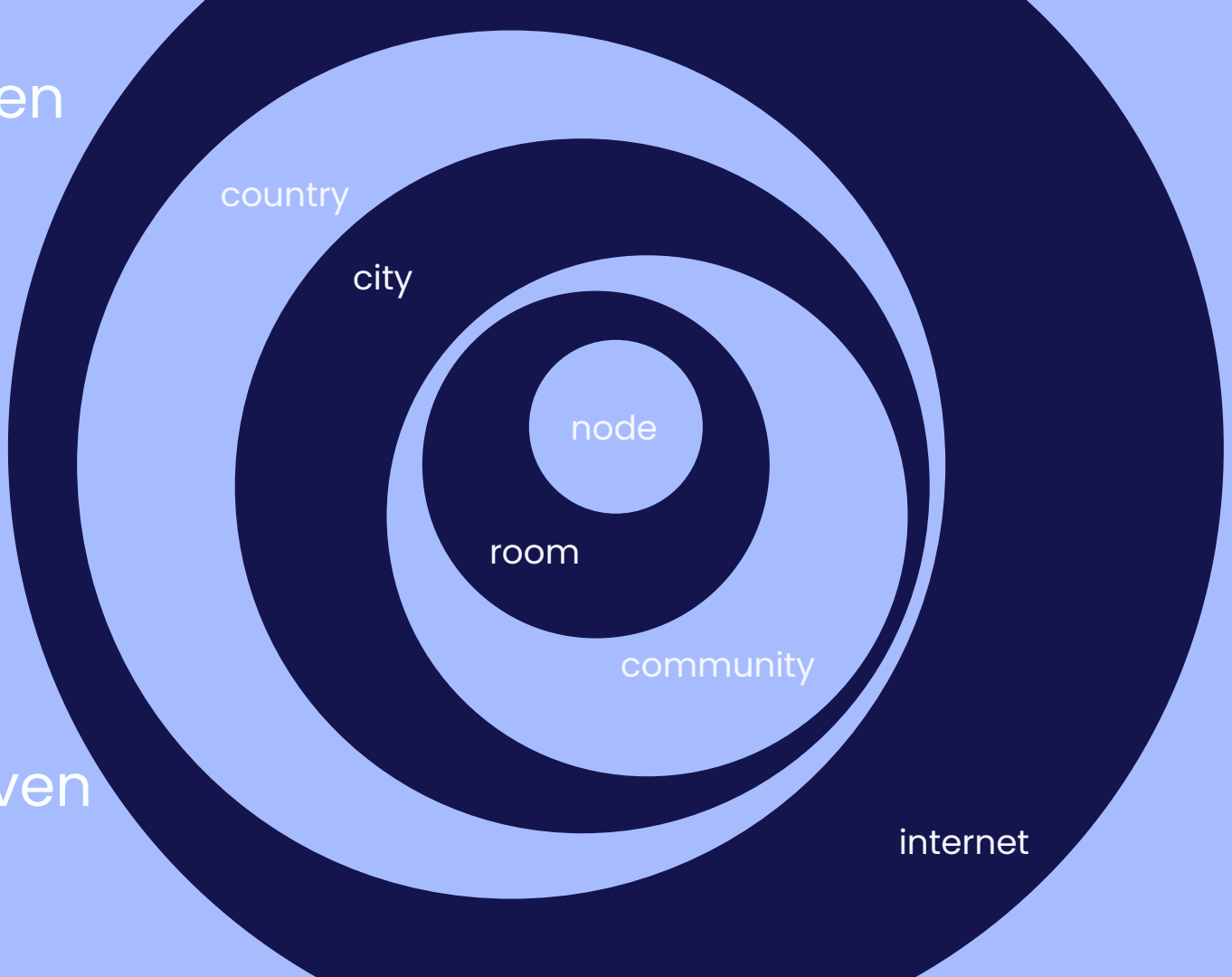
Priority	Effort	Goal
High	Low	Companion Mv3
Low	Medium	Companion design rework
High	Low	IPFS/FIL extension templates
High	Medium	FIL in Brave Wallet
High	Medium	NFT features in Brave
Medium	Low	Chromium protocol handler (Igalia)
Low	Medium	Mobile apps shipped
Low	Medium	Mobile OS demo
Medium	Low	Curl
Low	Medium	WebDAV

HTTP Client

HTTP Server



Context-driven availability



Demand-driven scaling

* par adigma tic collis ion

Domain-level vs keyholder-level privacy

Single- vs multi-source data

Local vs global connectivity

(lack of) application model

Location vs content addressing

CA-rooted trust vs keyholder trust

Transport agnositicism vs HTTP only

Persistent vs transient connections

No idiomatic IPFS usage, more like a “toolkit”

Transport encryption for public networks not just private connections

Point to point vs broadcast

Scale via single entity vs cooperative network

* par adigma
tic collis ion

web://browsers

Browser						
Filecoin	🚧	❌	❌	❌	❌	❌
Extension	✅	✅	✅	✅	✅	❌
ipfs:// handling	✅	✅	🚧	❌	❌	❌
Full IPFS node	✅	❌	❌	❌	❌	❌
Publicly positive	✅	✅	❌	❌	❌	❌
Privately positive	✅	✅	[redacted]			
Internal champions	✅	✅	✅	✅	✅	❌

(it’s complicated.)

Brave browser

Implemented support for ipfs:// and ipns:// in address bar

Implemented ability to run a full go-ipfs node

Bundled IPFS companion

Integration with ENS

Adding NFT features like auto-archive and health levels

Brave Browser

worked:

Willing and enthusiastic partner

Full time engineer embedded

Off by default

Clear privacy warnings

Experimenting

Lite and full options

challenges:

How visualize connectivity?

How to communicate trust?

What does “secure” even mean?

System resource usage?

HTTP & IPFS co-living?

Where’s my “stuff” in external node?

NFTs as a first class browser concept?

Chromium

Igalia working on various projects

Protocol registration API support

Refactoring protocol handling to support non-HTTP protocols

Be able to define scheme->handler at compile-time

Gateways first

Then bundle native IPFS support

Chromium

worked:

Going slow

Credible partner

Building up trust through compat fixes

Not-HTTP instead of IPFS

Backchannels

Broader standards environment

challenges:

HTTP up and down the stack

Major refactorings required

Taking a long time

Can’t bundle go-ipfs

Need new vision of “native IPFS”

Chromium support for Rust is not production ready

IPFS Companion

In most top browsers

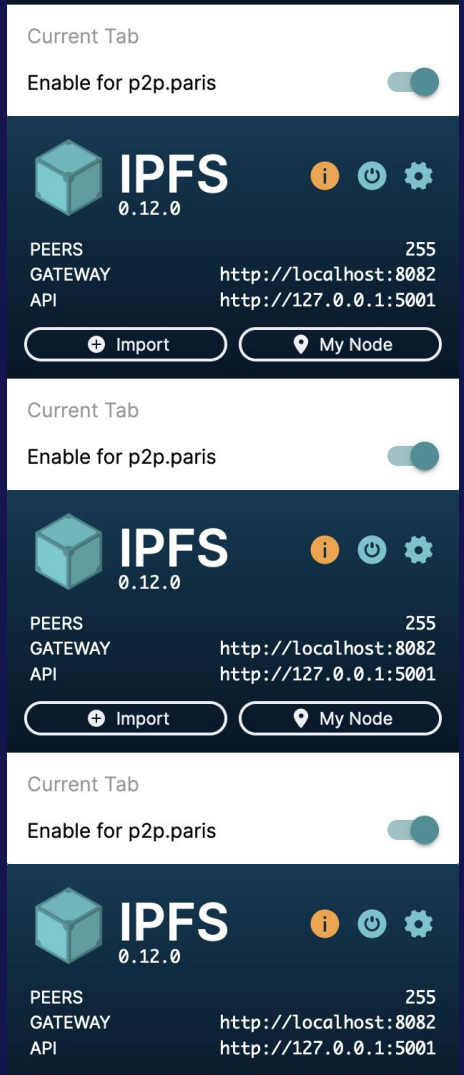
Connects to local node

And other browser UX features

Highly constrained

And Manifest v3 constrains further

And requires complete rewrite



IPFS Companion

worked:

Don’t fight it. Be like water.

Don’t depend on it in order to work - must be additive.

Make your voice heard in bug trackers for browsers.

Use it as a user acquisition channel.

challenges:

We can’t do the same stuff as before 😅

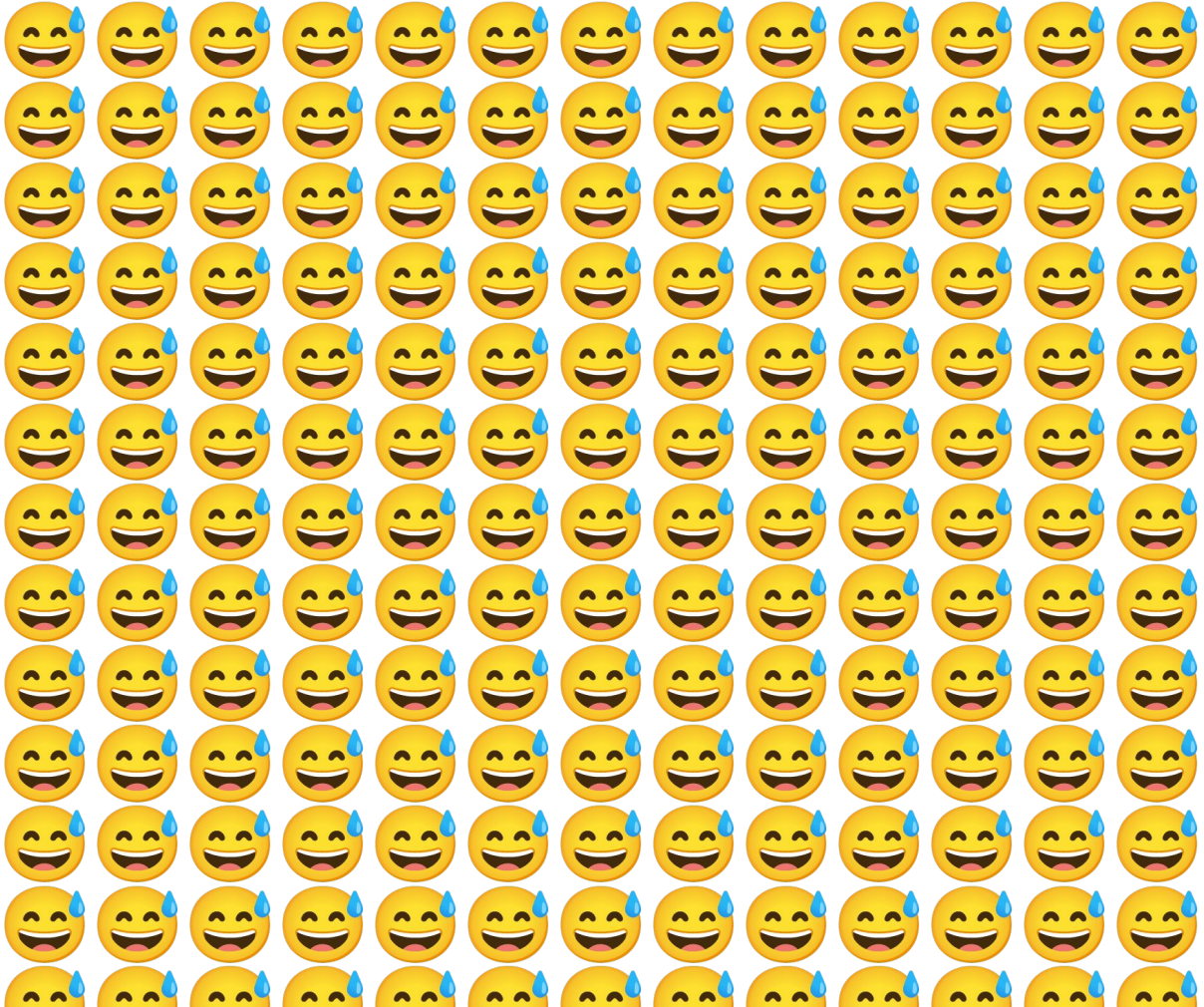
API surface area even smaller

Connectivity model even further from “persistent node” of IPFS

Again need to revisit IPFS core architecture

And yeah, we’re ready… but Google likely is pushing this back, since they’re not ready.

* mobile

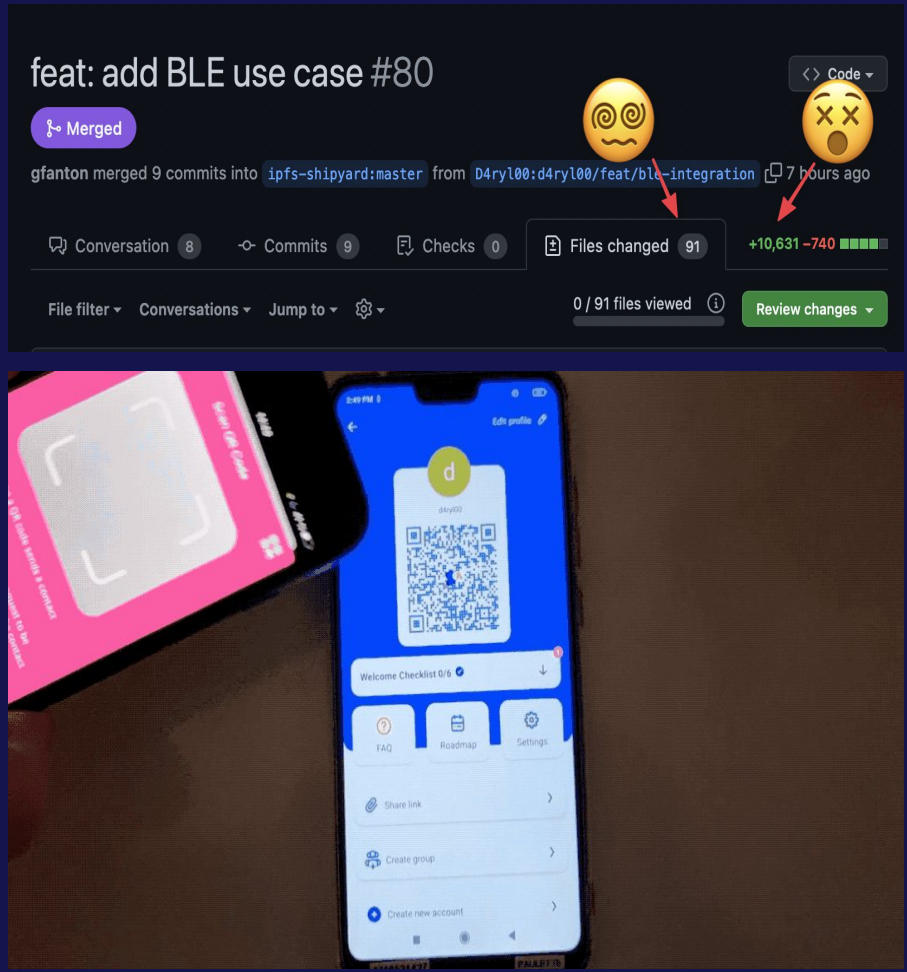


Berty

embedded go-ipfs with native bindings 🤯

BLE connectivity abstraction as a libp2p transport impl 🤯

cross-platform local connectivity 🤯



Berty

what worked:

Dedicated team

Single use-case

Focus on local/offline

Long development lead time

Proved the concept

challenges:

Native builds really tough to configure

Chews device resources

Peer-aggregation model not aligned with OS stack, let alone mobile network capabilities

Desktop/server-class software running on mobile device

Embedded golang high friction w/ native platforms

Durin

Native iOS and Android apps for experimenting with IPFS patterns and use-cases

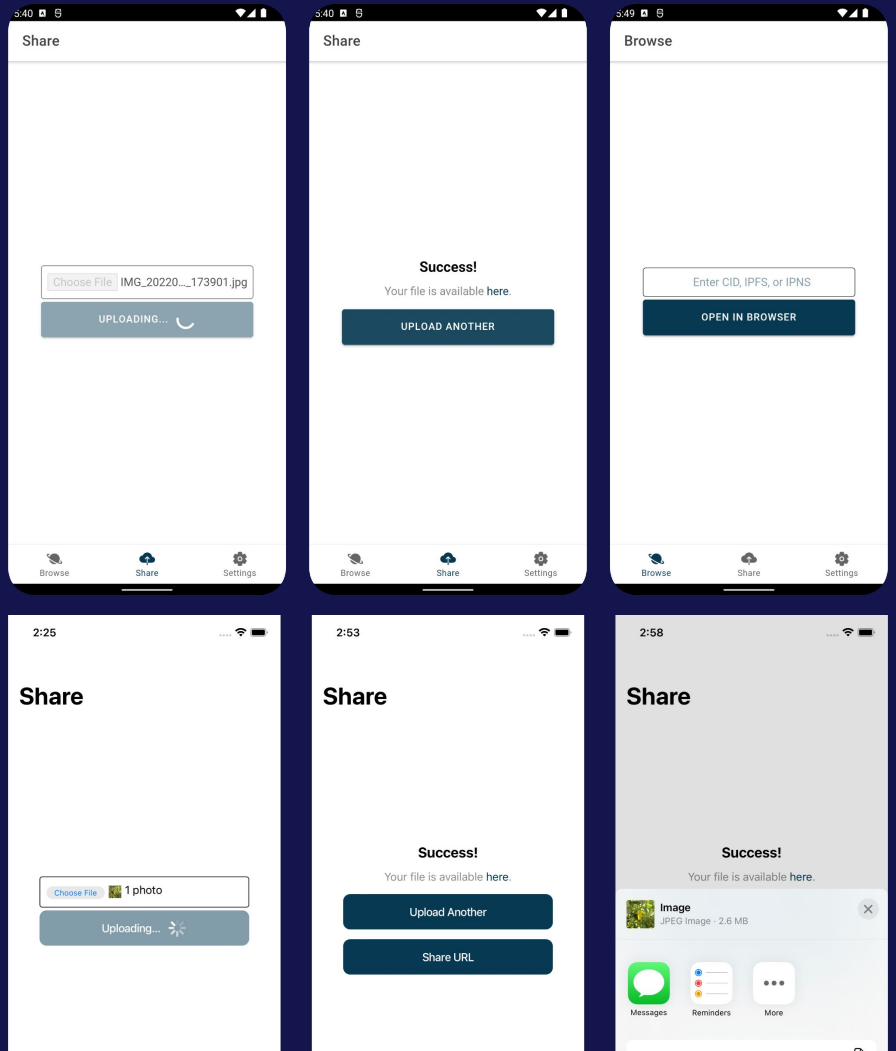
Built by Trigram and Outercore

Protocol handling, routing to renderers

Uploads, views of “my stuff”

UX exploration

Performance and device resource consumption testing



Durin

what worked:

Focus on native platform UX

Focus on end-user tasks

Start small and expand on learnings

Don’t go full IPFS

challenges:

Platform native integrations - eg photos and file storage, duplicate? copy?

Defining core user actions and use-cases, eg moving files to IPFS for local storage, to IPFS for sharing, from IPFS to native storage, etc

How to “share” to IPFS without centralizing?

How to not end up building a browser...

Agregore

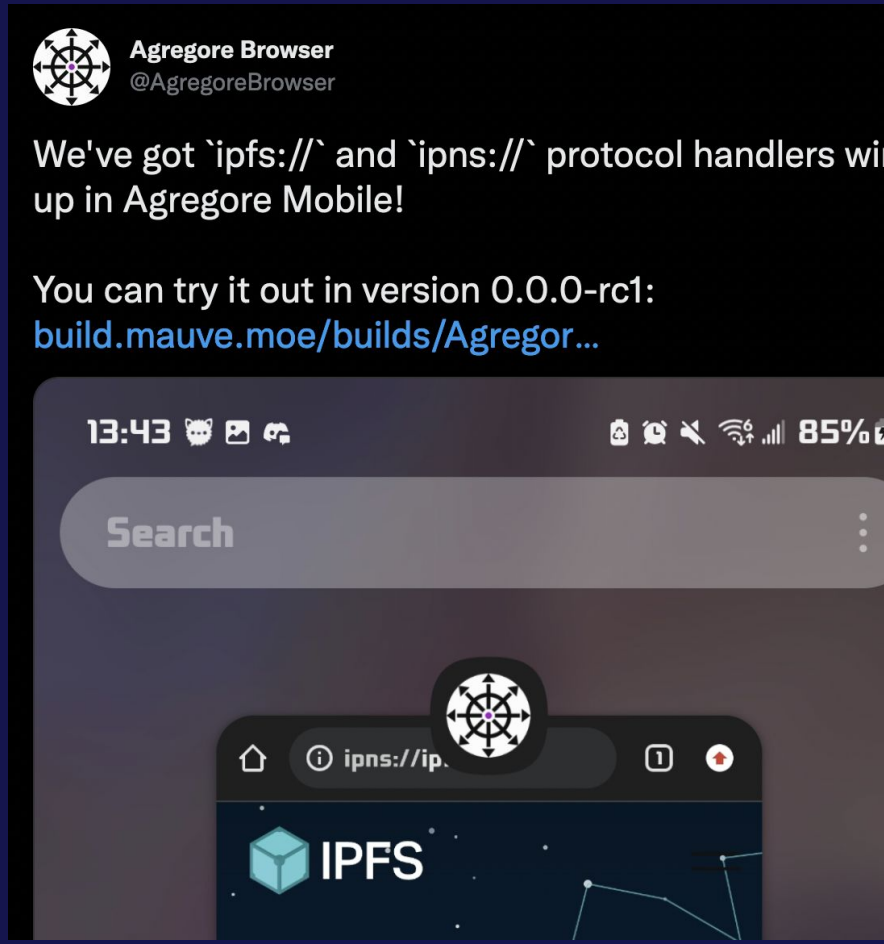
Designed as a multiprotocol browser

Received grant to develop Android version with IPFS support

Focused on a specific community need for mesh applications in South Africa

Tested a few approaches as a base, went with Chromium/Bromite

Experimenting in web platform integrations and mesh networking



Agregore

what worked:

Learn on desktop, port to mobile

Build on existing browser base

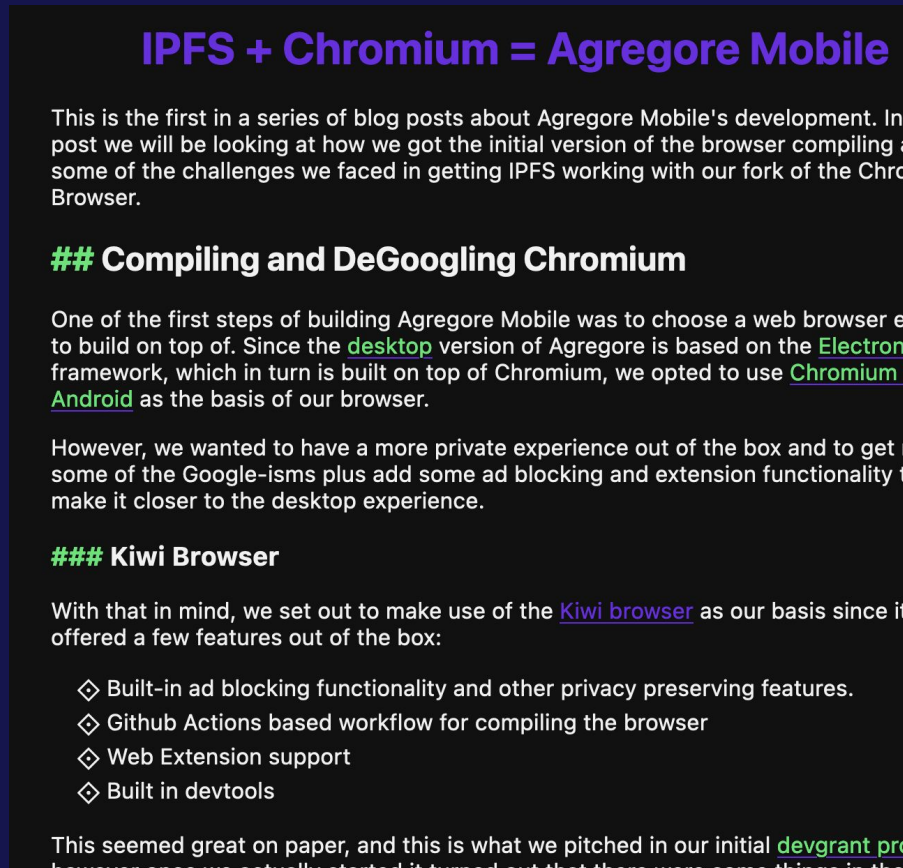
Specific use-case / community

Partner w/ other experts

Event-driven deadlines

Application model learnings

challenges:



IPFS + Chromium = Agregore Mobile

This is the first in a series of blog posts about Agregore Mobile's development. In post we will be looking at how we got the initial version of the browser compiling some of the challenges we faced in getting IPFS working with our fork of the Chro Browser.

Compiling and DeGoogling Chromium

One of the first steps of building Agregore Mobile was to choose a web browser e to build on top of. Since the desktop version of Agregore is based on the Electron framework, which in turn is built on top of Chromium, we opted to use Chromium Android as the basis of our browser.

However, we wanted to have a more private experience out of the box and to get some of the Google-isms plus add some ad blocking and extension functionality make it closer to the desktop experience.

Kiwi Browser

With that in mind, we set out to make use of the Kiwi browser as our basis since it offered a few features out of the box:

- Built-in ad blocking functionality and other privacy preserving features.
- Github Actions based workflow for compiling the browser
- Web Extension support
- Built in devtools

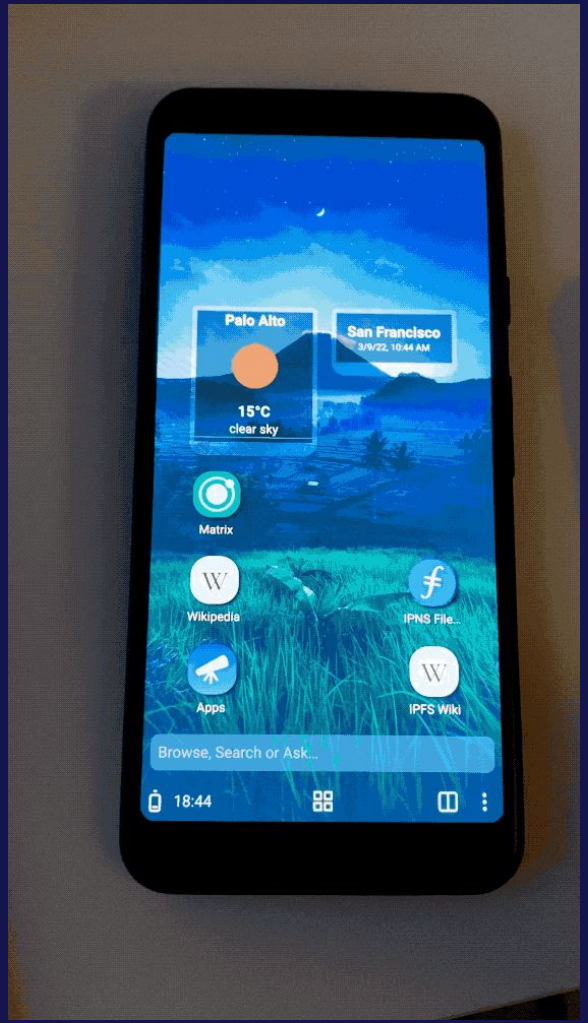
This seemed great on paper, and this is what we pitched in our initial devgrant pr

Capyloon.org

Capyloon is a mobile operating system (based on KaiOS (based on B2G OS (based on Firefox OS)))

Filecoin Devgrant to support development of IPFS and Filecoin features, from core end-user needs to OTA updates, build system... everything.

Supports ipfs/ipns schemes, bundles MetaMask, images distributed on IPFS.



Capyloon

what worked:

Full control of OS stack

Full control of UI

IPFS at work in making the OS

Interested partners

A production base

challenges:

Maybe embedded Webkit or Chromium a better long term option?

Daily use immediately uncovered gateway issues in regular web content

Distribution of build artifacts over IPFS uncovered bugs in web3.storage

What are IPFS apps?

How to do things like OS-level mobile backups on IPFS? So much private data

ffmpeg

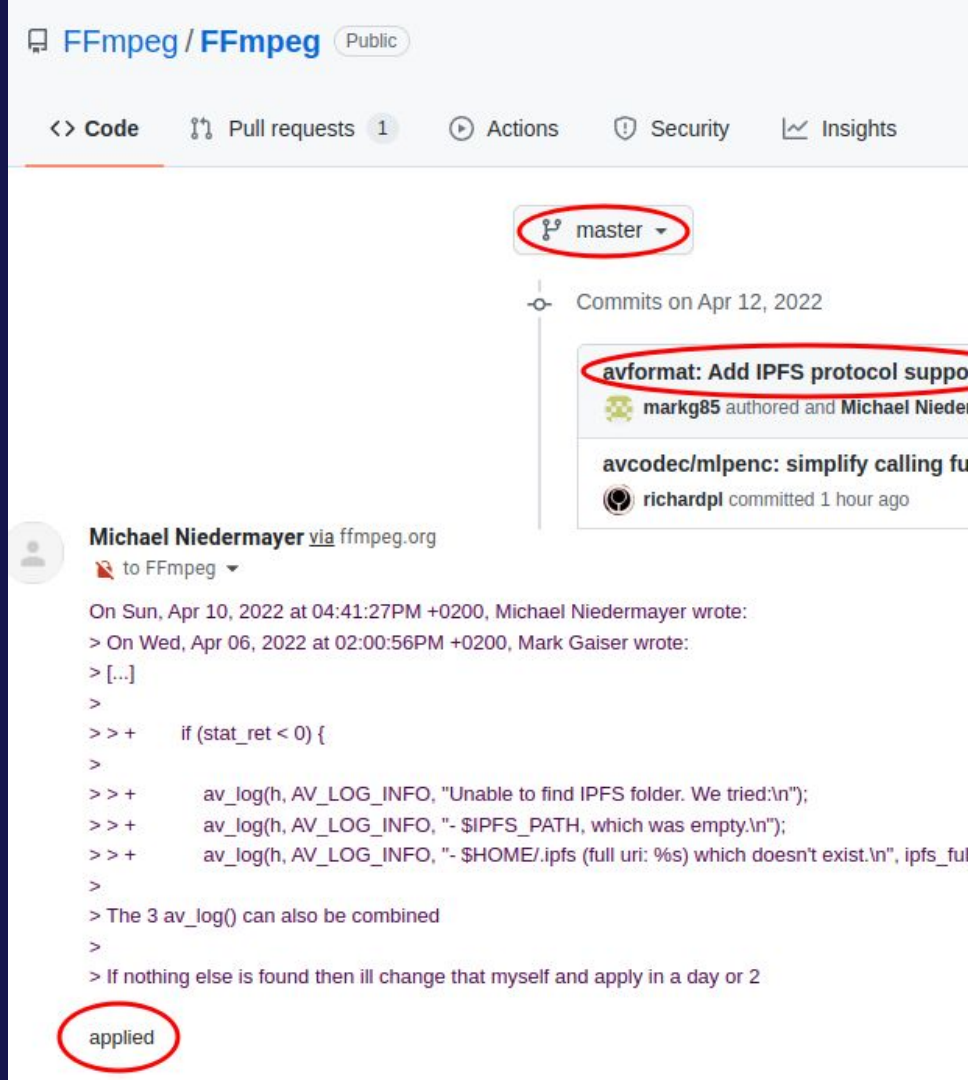
mark5891 starts poking at it

we meet across a crowded chat room

writes up a microgrant proposal

(months of back and forth on ffmpeg mailing list)

boom, landed!



ffmpeg

what worked

Meet community where they’re at

Take your time

Explain a lot

Don’t go full IPFS

Compromise

challenges

Defaults configuration!

Fallback connectivity schemes!

Don’t accidentally centralize!

OMGSTRINGS!

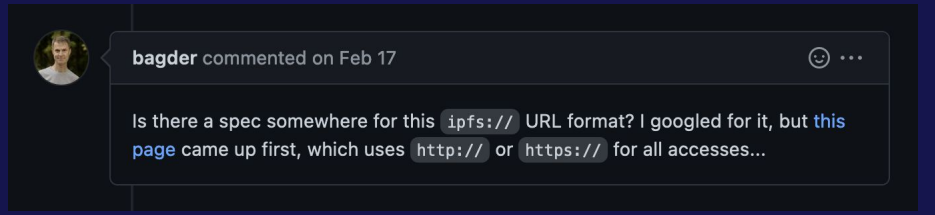
curl

jchris prototyped, submitted PR

badger is interested!

Still some work to spec out configuration and behavior, but the road forward is clear enough at this point.

A big win in itself, when popular open source tools say “yeah we know IPFS, let’s talk” ⚡⚡⚡



> Using a scheme called 'IPFS' when in reality it is doing HTTPS
> is going to cause confusion. Are we adding IPFS support or are we not?

There are three ways a user agent can support resolution of ipfs:// addresses

(A) Fetch from a trusted HTTP gateway. Payload integrity check happens on the gateway. Note: HTTP consumers often run their own gateway on localhost for better performance and to remove this need for trusting a third-party gateway.

(B) Fetch from a "trustless" HTTP gateway: data is fetched block-by-block or as a CAR (https://ipld.io/specs/transport/car/carv1/), both being opaque byte streams. Then, the deserialization and integrity verification happens on the client.

(C) Full P2P node (integrity verification + libp2p stack with DHT/mDNS and P2P transports).

I believe in case of curl (tool/library) we would be discussing something between A and B.
See my proposal below.

curl

what worked

Meet community where they’re at

Take your time

Explain a lot

Don’t go full IPFS

Compromise

challenges

No easy to find spec (headsmack)

What even is IPFS?! Native vs gateway’d usage vs native vs http schemes… just so much that is so confusing, esp for new folks

Where to implement? Curl tool vs libcurl?

Again with the environment config b/c no native IPFS - can apply FFmpeg learnings/approach

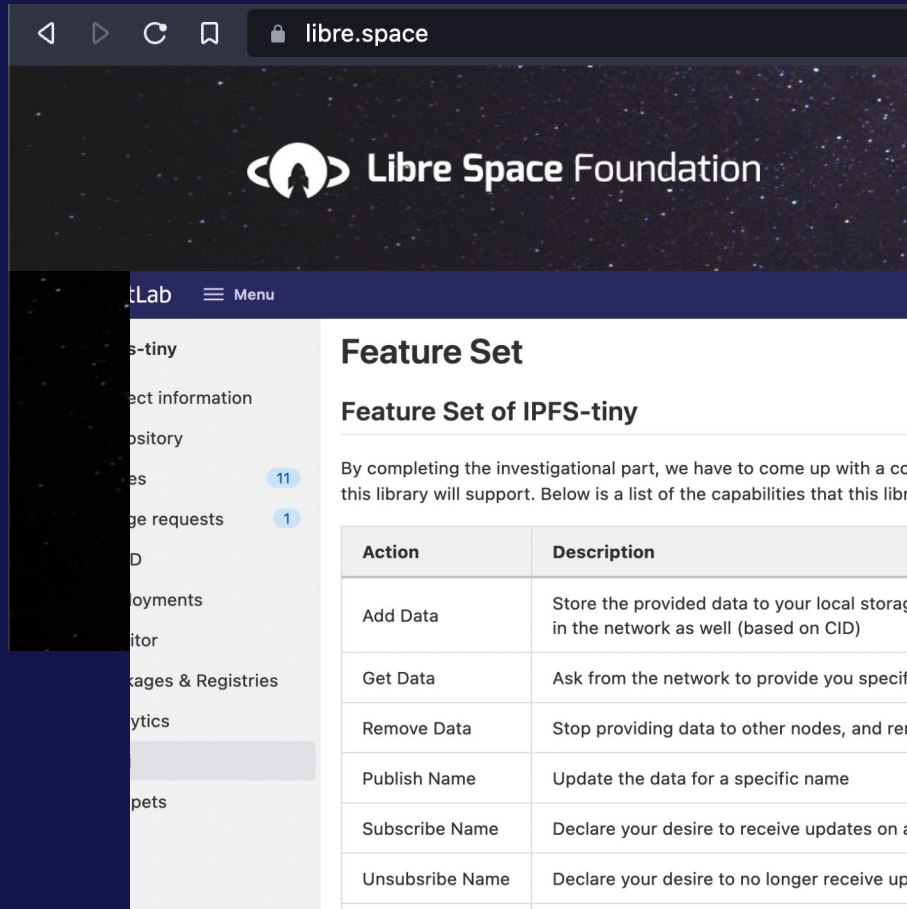
* ipfs-tiny

C++ implementation of a subset of IPFS core

Designed for emitting content-addressed sensor data

Run on embedded devices, microcontrollers, etc

Push data using whatever available network stack, out to a specialized gateway



ipfs-tiny

what worked

TBD

challenges

“WHY C++?!” 😐

Some IPFS C++ work in the past, but not designed for embedded systems

Highly constrained network stack on device

Mesh layout, with a collection “portal”… again, what is “IPFS”?

Space

Explorations of IPFS for use-cases in...

⤵️ Space to ground, aggregating satellite-emitted data across multiple ground stations

↔️ Space to space, defining open standards for content-addressed communication between disparate space-based platforms

🌕 Lunar communications infrastructure

Space

what worked

TBD

challenges

Highly constrained environments

Proprietary platforms

Secrecy

IPFS Everywhere
Gaps Identified / Lessons Learned

➡️ IPFS does not have application model

➡️ IPFS implementation is inverted - libraries not nodes

➡️ Not enough libraries *from **their** communities*

➡️ Language elitism == smaller adoption vector (no best tool, only choosable tools)

➡️ Transport-agnostic **cannot** === libp2p

➡️ Transient networks vs persistent peer aggregation