ditto

Real World Local-First Deployments

An Experience Report

Borja de Régil
Sr. Engineer @ Ditto

A local-first app wishlist?

GitHub

Social Media icons illustration by Bastian Riccardi on Unsplash

Office illustration by Igor Omilaev on Unsplash

Expanding the local-first vision

Remember your flight here?

One of the last places without ubiquitous connectivity

Need to support commerce
(bought anything recently?)

Plenty of logistics occurring behind the scenes of passengers
(checklists, manage on-flight stock, flight attendant collaboration)

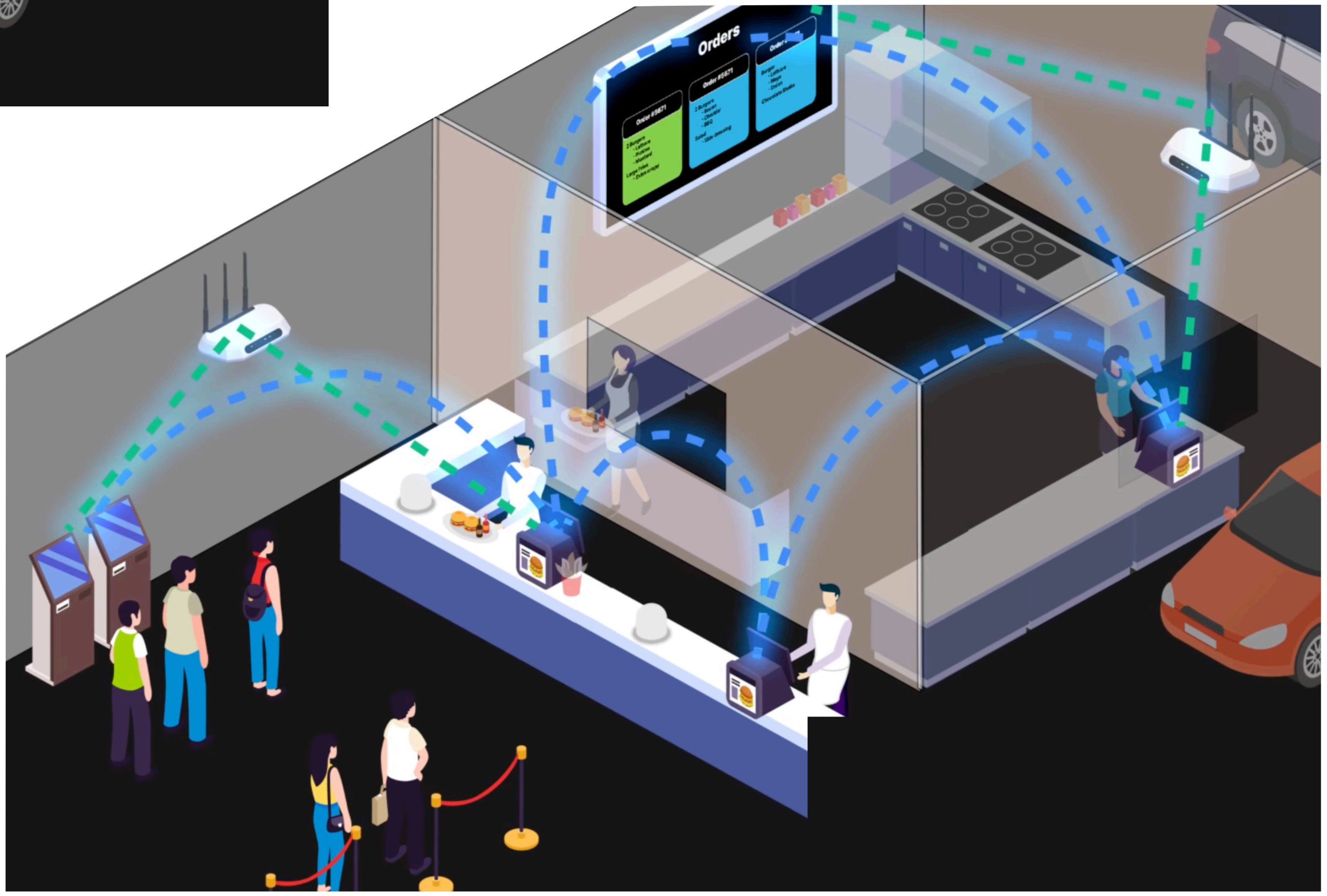
ditto

Restaurants have changed

Multiple points of order and updating displays
(drive-through, self-serve kiosks, waiter tablets, menus, kitchen displays & printers)

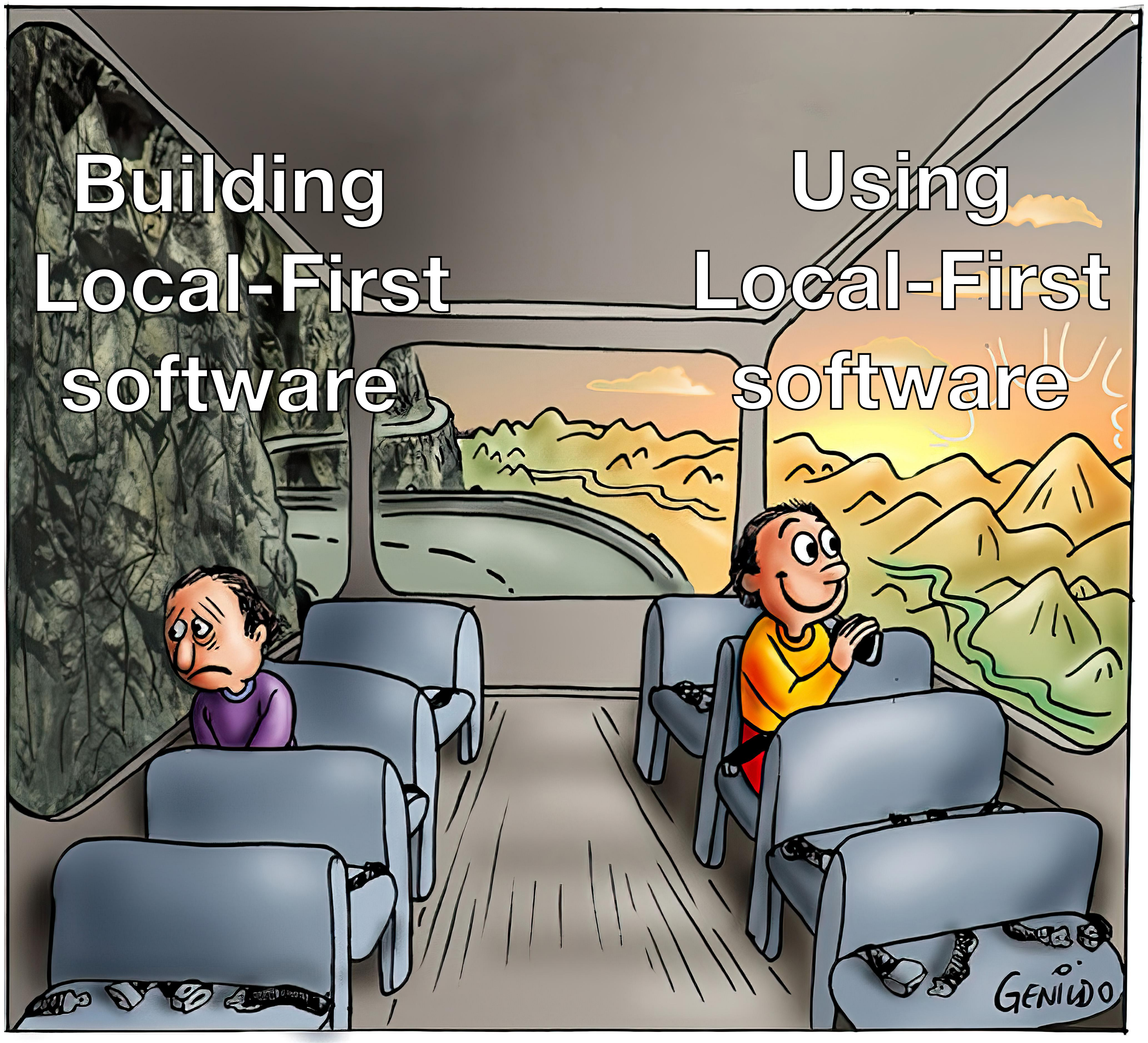
All connected by brittle networks, all needed to be up for smooth operation

Why are all these devices talking to the cloud?



P2P connectivity is *key*

Why don't we see more local-first software?



ditto

Expectations of users

Most interaction happens on hand-held devices

Usage "on the go", data always available

Cross device, your data should move seamlessly

Expectations of users

Most interaction happens on hand-held devices

Usage "on the go", data always available

Cross device, your data should move seamlessly

Challenges for developers

Multiple transport protocols. Which to use & how to combine?

Discovery. How do you form a mesh of devices?

Replication. How do you move data around?

Conflicts. How do you reconcile cross-device conflicts?

Auth & Security. Who has access to your data?

Usability. How do you expose all of this to end developers?

ditto

Challenges for developers

Multiple transport protocols. Which to use & how to combine?

Discovery. How do you form a mesh of devices?

Replication. How do you move data around?

Conflicts. How do you reconcile cross-device conflicts?

Auth & Security. Who has access to your data?

Usability. How do you expose all of this to end developers?

Let's make local-first useful for everyone

Chick-fil-A

DELTA

DNERO

Alaska AIRLINES

JAPAN AIRLINES

Logipad aero

Lufthansa

SPONT

sharemyhealth

U.S. AIR FORCE

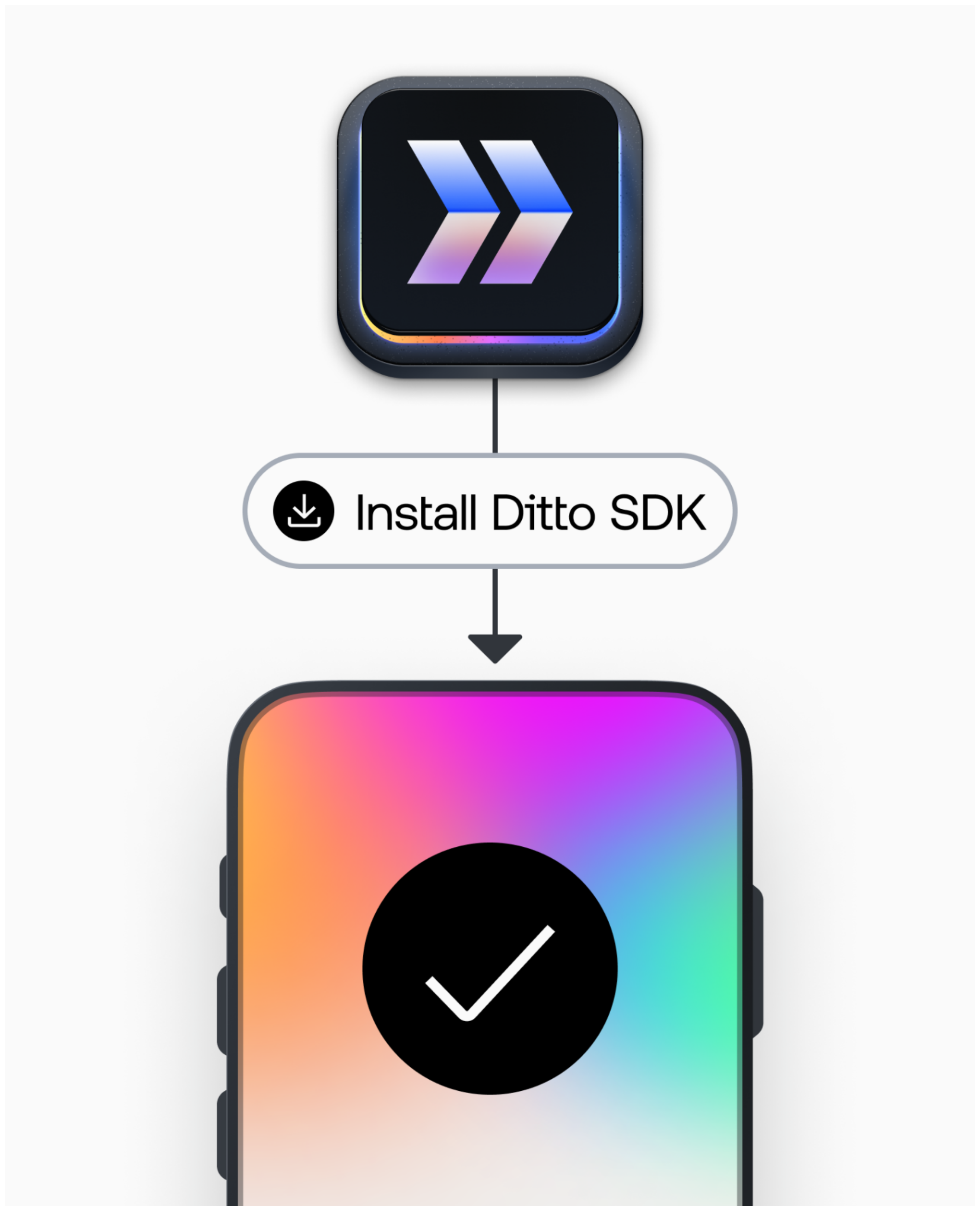
ditto

Topics

Ditto: making it work (choices, choices)

Deploying local-first: lessons learned

Onwards: challenges ahead



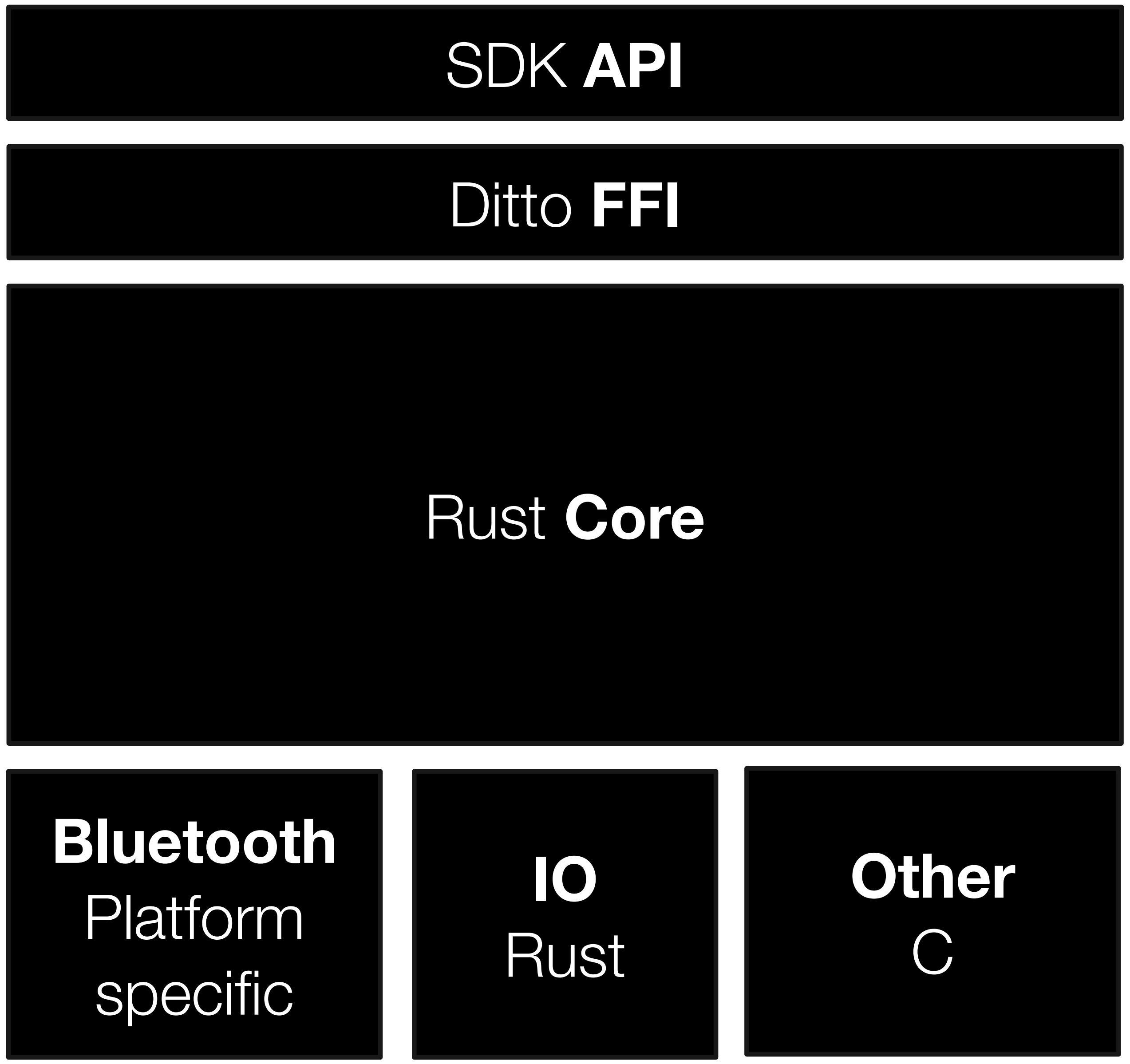
An SDK that allows you to write apps supporting P2P and cloud

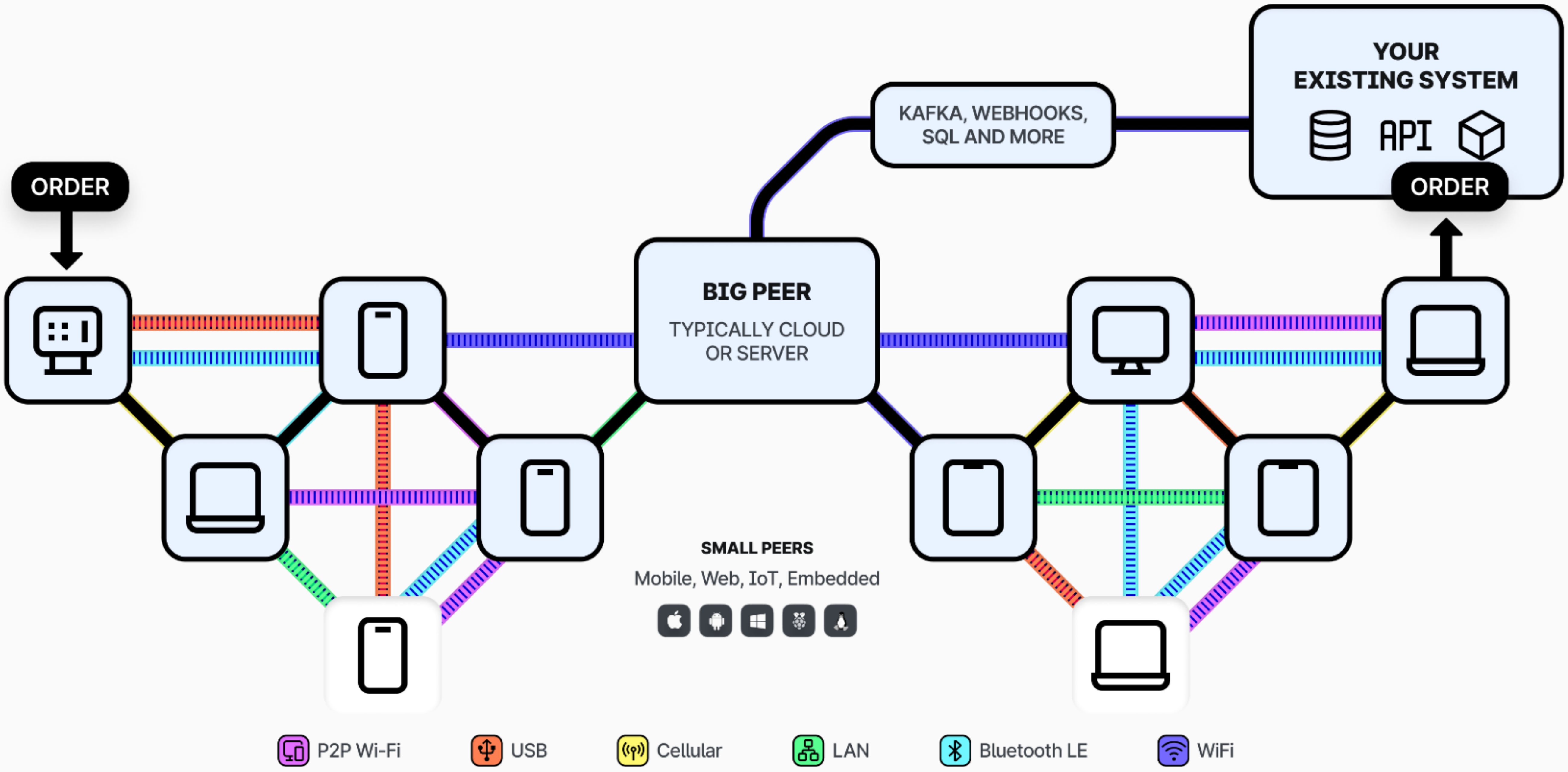
Automatically connects devices in ad-hoc mesh networks

Data moves P2P, but the overall ecosystem feels traditional

Puts tested local-first research into practice

ditto


SDK **API**
Ditto **FFI**
Rust **Core**
Bluetooth Platform specific
IO Rust
Other C


YOUR EXISTING SYSTEM
API
KAFKA, WEBHOOKS, SQL AND MORE
ORDER
ORDER
BIG PEER
TYPICALLY CLOUD OR SERVER
SMALL PEERS
Mobile, Web, IoT, Embedded
P2P Wi-Fi
USB
Cellular
LAN
Bluetooth LE
WiFi

Tackling the challenges

Multiple transport protocols

Discovery

Replication

Conflicts

Auth & Security

Usability

Tackling the challenges

Multiple transport protocols

Discovery

Replication

Conflicts

Auth & Security

Usability

ditto

Usability: the mesh is a database

```
ditto.sync.register_subscription("""
      SELECT * FROM cars
       WHERE color = 'blue'
""")

ditto.store.execute("""
    INSERT INTO cars DOCUMENTS({
        '_id': 'Honda',
        'color': 'blue',
        ...
    })
""")
```

Conflicts: everything is a Δ-CRDT

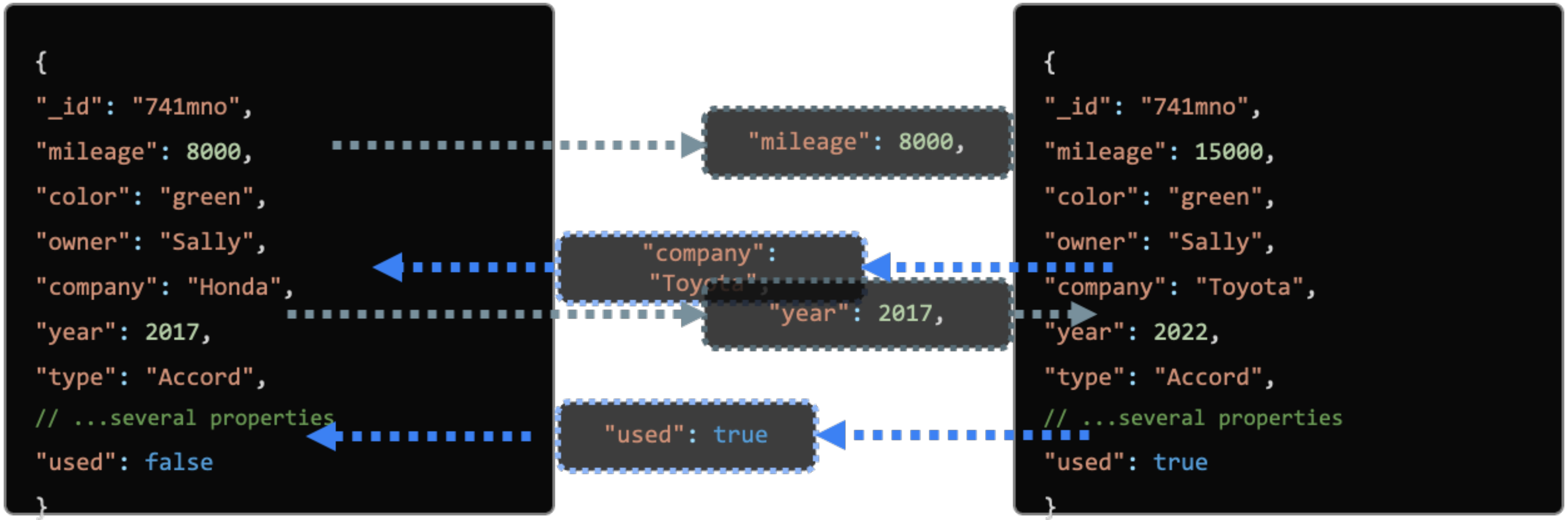
Every row in the database is a CRDT map

Multiple types: maps, registers and counters

Diffs (deltas) prevent the state accumulation problem
(but require more metadata per document)

An inside look at Ditto's Delta State CRDTs: https://ditto.live/blog/dittos-delta-state-crdts
Albert van der Linde, João Leitão, and Nuno Preguiça. Δ-CRDTs: making δ-CRDTs delta-based

Diffs represent the minimal set of changes to be exchanged



```
{
"_id": "741mno",
"mileage": 8000,
"color": "green",
"owner": "Sally",
"company": "Honda",
"year": 2017,
"type": "Accord",
// ...several properties
"used": false
}
```

```
{
"_id": "741mno",
"mileage": 15000,
"color": "green",
"owner": "Sally",
"company": "Toyota",
"year": 2022,
"type": "Accord",
// ...several properties
"used": true
}
```

ditto

Replication: subscription-based sync

```
ditto.sync.register_subscription("""
      SELECT * FROM cars
       WHERE color = 'blue'
""")
```

Devices replicate only what they're interested in

A mesh forms around overlapping subscriptions

Push-based partial replication allows efficient bandwidth and disk usage (polling is inefficient, flooding wastes network resources)

ditto

Replication: Attachments

```
// Peer A
token = ditto.store.new_attachment(data)

// Peer B
ditto.store.fetch_attachment(token)
```

Attachments allow to move binary blobs through the mesh

Tokens can be shared through normal replication

Transfers can be stopped, resumed and cancelled

ditto

Auth & Security

Bring your own identity provider and role-based permission system

Device fleets tend to have centralized control: devices must authenticate online before going offline, and re-authenticate periodically

Offline auth possible, but requires manual work

Permissions are data-oriented: what am I allowed to read and write?

Mesh devices will only talk to devices with matching roots of trust

Deploying local-first: lessons learned

Iterating safely and quickly in an open system

Your degree of control is limited

Software updates going through the App store take time and can get rejected

Deployments will run multiple versions most of the time

Iterating safely and quickly in an open system

Changes in behavior require cross-version coordination

Users can be in control, but at some point you will need to make the new thing the default

Epidemic P2P protocol to spread changes through the mesh

ditto

Understanding your data model
(knowing how much to abstract away)

Tried to emulate JSON too much, after document databases

The default choices you make to model JSON will haunt you
(ask me about using RGAs for arrays)

Basing your system on JSON means finding ways to:
- Map between JSON and internal CRDT representation
- Let users specify type definitions, we went through a few
- Ties your query language to a JSON structure, hard to extend

ditto

Understanding your data model

(knowing how much to abstract away)

Almost everyone understands last-write-wins semantics

Documents end up being "flat" most of the time

A relational model brings you several advantages:
- Everyone is familiar with SQL
- Better support for type definitions, schemas* and operations
- Lets users express more complex queries

Deletes are hard

(especially in a subscription-based world)

```
ditto.sync.register_subscription("""
      SELECT * FROM cars
       WHERE color = 'blue'
""")
```

ditto

Deletes are hard

(and some kinds of deletes are harder)

CRDTs offer different possibilities for removal

Conflict resolution for deletes can be confusing

A remove-wins world means you will be forever looking for the device that fell behind the couch (which is overwriting your other data)

An add-wins world brings you problems of causal vs temporal conflicts.

Everyone understands the soft-delete pattern. Lean on it.

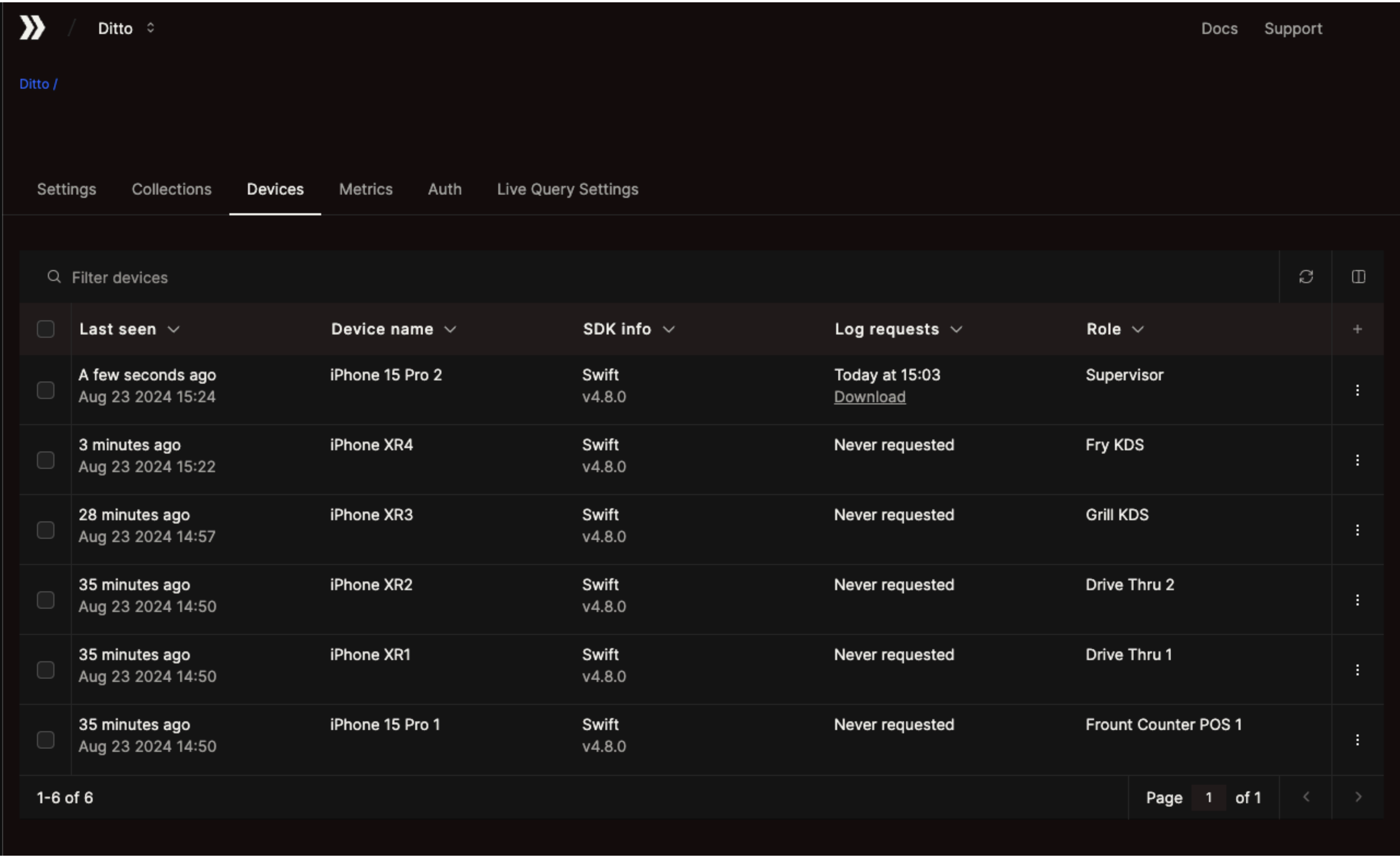
ditto

Observability & Introspection

Network managers want know about devices in the field

- Mesh network status
- Last time a device was online
- Device names and unique identifiers
- Device logs
- Custom metadata

ditto



Last seen	Device name	SDK info	Log requests	Role
A few seconds ago Aug 23 2024 15:24	iPhone 15 Pro 2	Swift v4.8.0	Today at 15:03 Download	Supervisor
3 minutes ago Aug 23 2024 15:22	iPhone XR4	Swift v4.8.0	Never requested	Fry KDS
28 minutes ago Aug 23 2024 14:57	iPhone XR3	Swift v4.8.0	Never requested	Grill KDS
35 minutes ago Aug 23 2024 14:50	iPhone XR2	Swift v4.8.0	Never requested	Drive Thru 2
35 minutes ago Aug 23 2024 14:50	iPhone XR1	Swift v4.8.0	Never requested	Drive Thru 1
35 minutes ago Aug 23 2024 14:50	iPhone 15 Pro 1	Swift v4.8.0	Never requested	Frount Counter POS 1

1-6 of 6 Page 1 of 1

Observability & Introspection

Device operators want to know about local status

- Who is this device connected to (presence viewer)
- Online sync status
- Local data browser
- Disk usage reporting
- Advanced configuration via configuration language

ditto

Devices don't start from scratch

Need to quickly import external sources of data

Many devices will probably need the same set of data on startup

Initial import needs to avoid conflicts, otherwise you risk a flood of replication and needless reconciliation

Ditto performs initial pre-load with a content-derived causal context to prevent conflicts

Devices have small disks and data accumulation is real

Devices dynamically change which partial data-set they're interested in

Need to decouple deleting everywhere (tombstone) vs deleting locally (evict)

Evicting is forgetting, so be sure to respect causal order

Support ways to determine when data is replicated offsite

ditto

Onwards: challenges ahead

Challenges ahead

Beyond the mesh

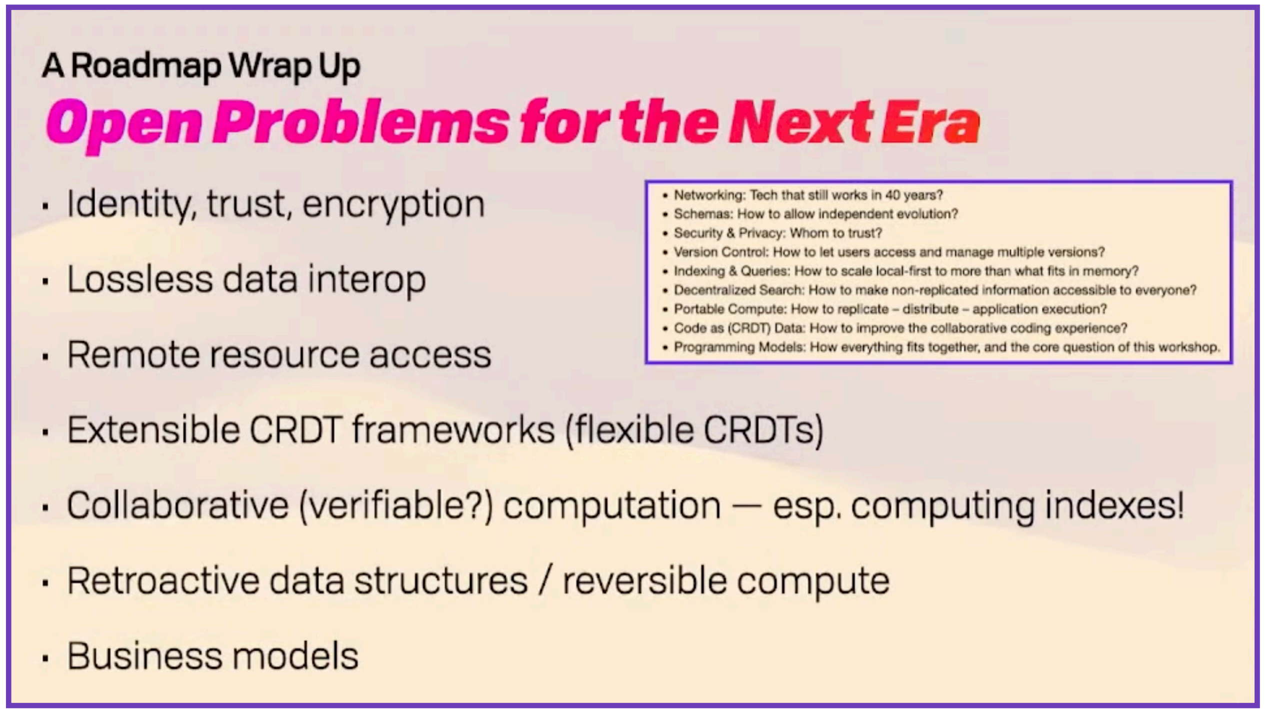
Schema support and evolution

Offline auth, trust, delegation

Dealing with erratic or buggy peers

Flexible reconciliation policies

Metadata growth in large meshes



A Roadmap Wrap Up

Open Problems for the Next Era

- Identity, trust, encryption
- Lossless data interop
- Remote resource access
- Extensible CRDT frameworks (flexible CRDTs)
- Collaborative (verifiable?) computation — esp. computing indexes!
- Retroactive data structures / reversible compute
- Business models

- Networking: Tech that still works in 40 years?
- Schemas: How to allow independent evolution?
- Security & Privacy: Whom to trust?
- Version Control: How to let users access and manage multiple versions?
- Indexing & Queries: How to scale local-first to more than what fits in memory?
- Decentralized Search: How to make non-replicated information accessible to everyone?
- Portable Compute: How to replicate – distribute – application execution?
- Code as (CRDT) Data: How to improve the collaborative coding experience?
- Programming Models: How everything fits together, and the core question of this workshop.

ditto

Challenges ahead

Beyond the mesh

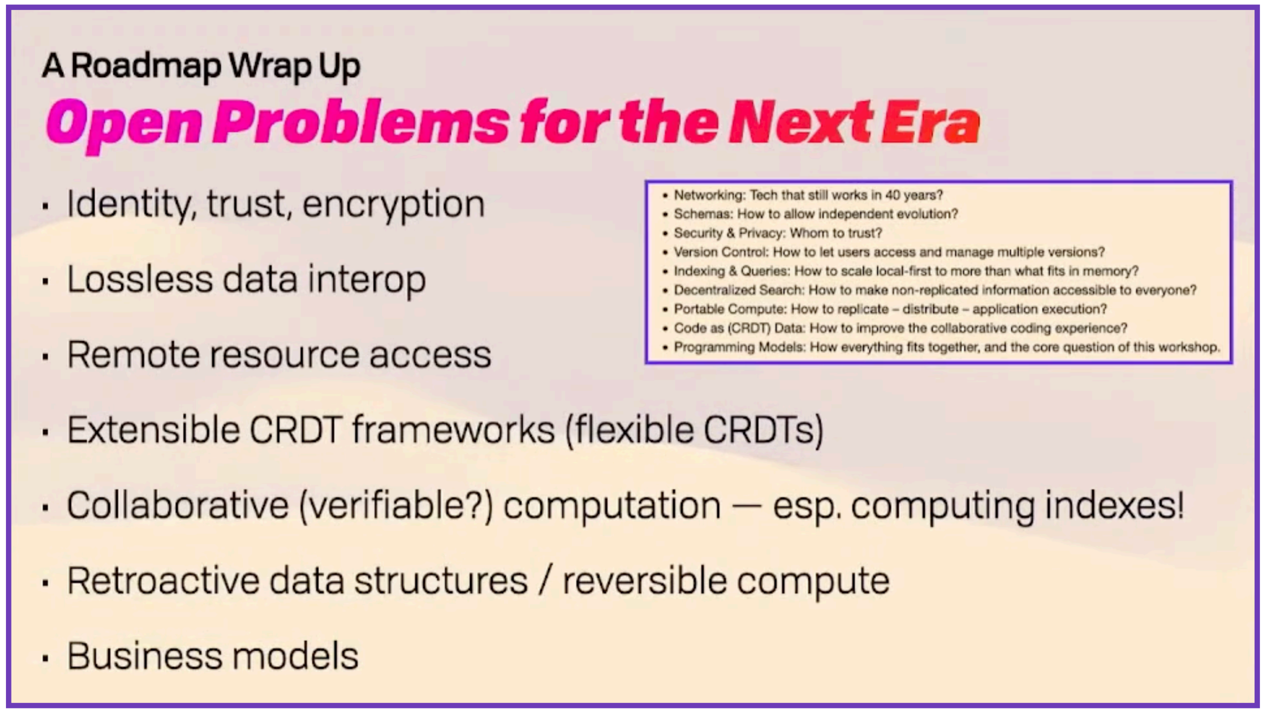
Schema support and evolution

Offline auth, trust, delegation

Dealing with erratic or buggy peers

Flexible reconciliation policies

Metadata growth in large meshes



A Roadmap Wrap Up

Open Problems for the Next Era

- Identity, trust, encryption
- Lossless data interop
- Remote resource access
- Extensible CRDT frameworks (flexible CRDTs)
- Collaborative (verifiable?) computation — esp. computing indexes!
- Retroactive data structures / reversible compute
- Business models

- Networking: Tech that still works in 40 years?
- Schemas: How to allow independent evolution?
- Security & Privacy: Whom to trust?
- Version Control: How to let users access and manage multiple versions?
- Indexing & Queries: How to scale local-first to more than what fits in memory?
- Decentralized Search: How to make non-replicated information accessible to everyone?
- Portable Compute: How to replicate – distribute – application execution?
- Code as (CRDT) Data: How to improve the collaborative coding experience?
- Programming Models: How everything fits together, and the core question of this workshop.

ditto

Beyond the mesh

Pre-existing systems already have a data store

Mesh devices need to import and export data to the outside world

Metadata and provenance is lost when data leaves the mesh

Track mutations occurring outside of our conflict-free world

ditto

Managing and evolving schemas

Are schemas a first-class concept?

How do peers communicate about schemas?

How to safely change and evolve them?

ditto

Offline auth, trust, delegation

Need permission systems that evolve with time

Auth delegation and revocation in a fully offline manner

Can you build a fully offline Slack clone? And make it encrypted?

Dealing with buggy peers

You will introduce a bug at some point

Are your bugs viral? Do they infect data across the mesh?

Revocation, but also reversible data

Imagine a world where all software is Local-First

ditto

Thank you!

Borja de Régil

borja@ditto.live

Chat offline at PLF!

GET IT ON
Google Play

Download on the
App Store