NextGraph: Building P2P and E2EE Community Applications

Niko Bonnieure · 2026-09-30 · English

Edited transcript of the presentation and discussion on 30 September 2026. Repetitions and clear recognition errors have been corrected while preserving the order, examples, qualifications and questions. Product status and plans reflect the talk date. Timestamps refer to the edited video.



NextGraph and Niko Bonnieure · 00:00:00

Introduction to NextGraph

00:00:00 · Niko Bonnieure

Hello, everyone. Thank you for inviting me. I don't know if you can see my shared screen. Yes, someone has confirmed that you can, so I can switch slides. I'm Niko, and I'm working on NextGraph, which is sponsored by those European grantmakers, the European Commission. I want to give an introduction to NextGraph first, and then we can dive into the parts you are most interested in. You can tell me which ones.



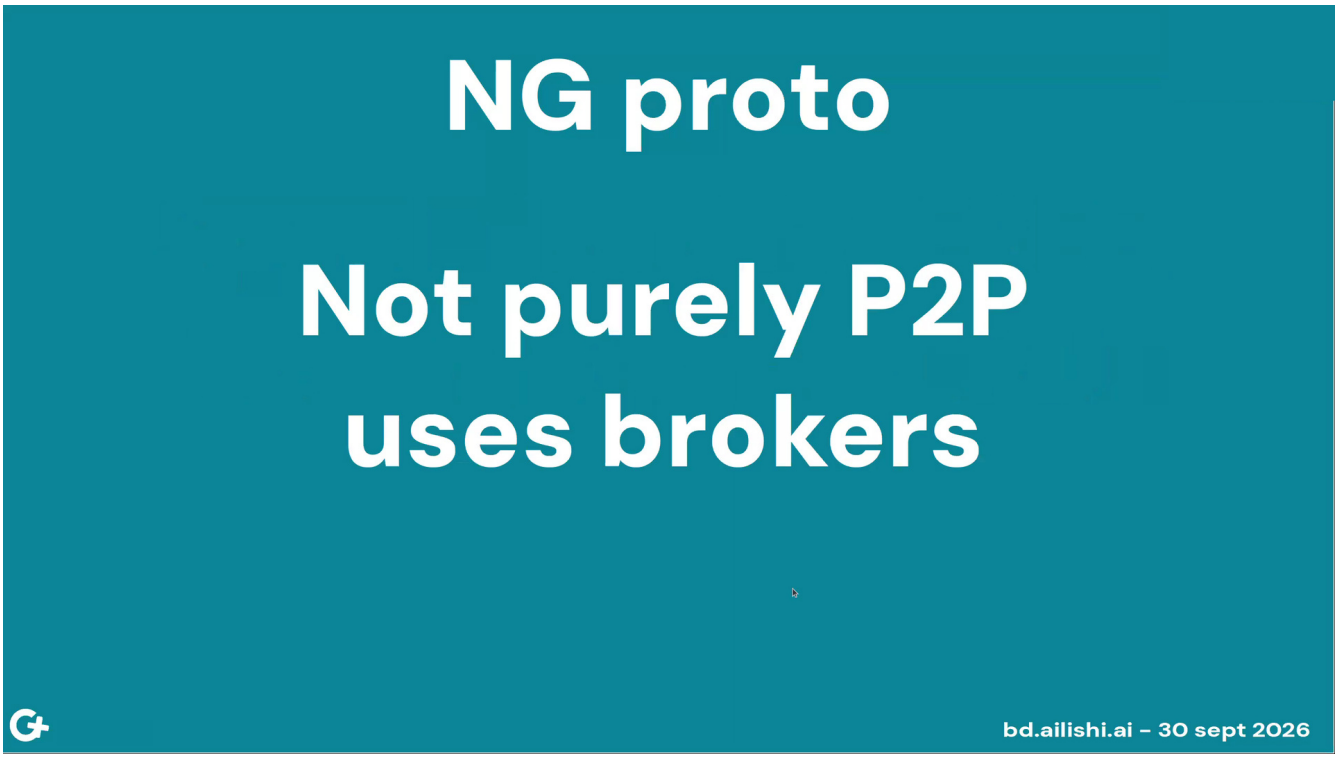
Project sponsors · 00:00:19



E2EE, local-first and CRDTs · 00:00:28

00:00:44 · Niko Bonnieure

First, we have a protocol for synchronizing data. It is end-to-end encrypted: the server that receives the data cannot see it. It is also local-first, meaning it can work without the internet, and it is based on CRDTs, which I think you all know.



Why NextGraph uses brokers · 00:01:15

Why synchronization uses brokers

00:01:17 · Niko Bonnieure

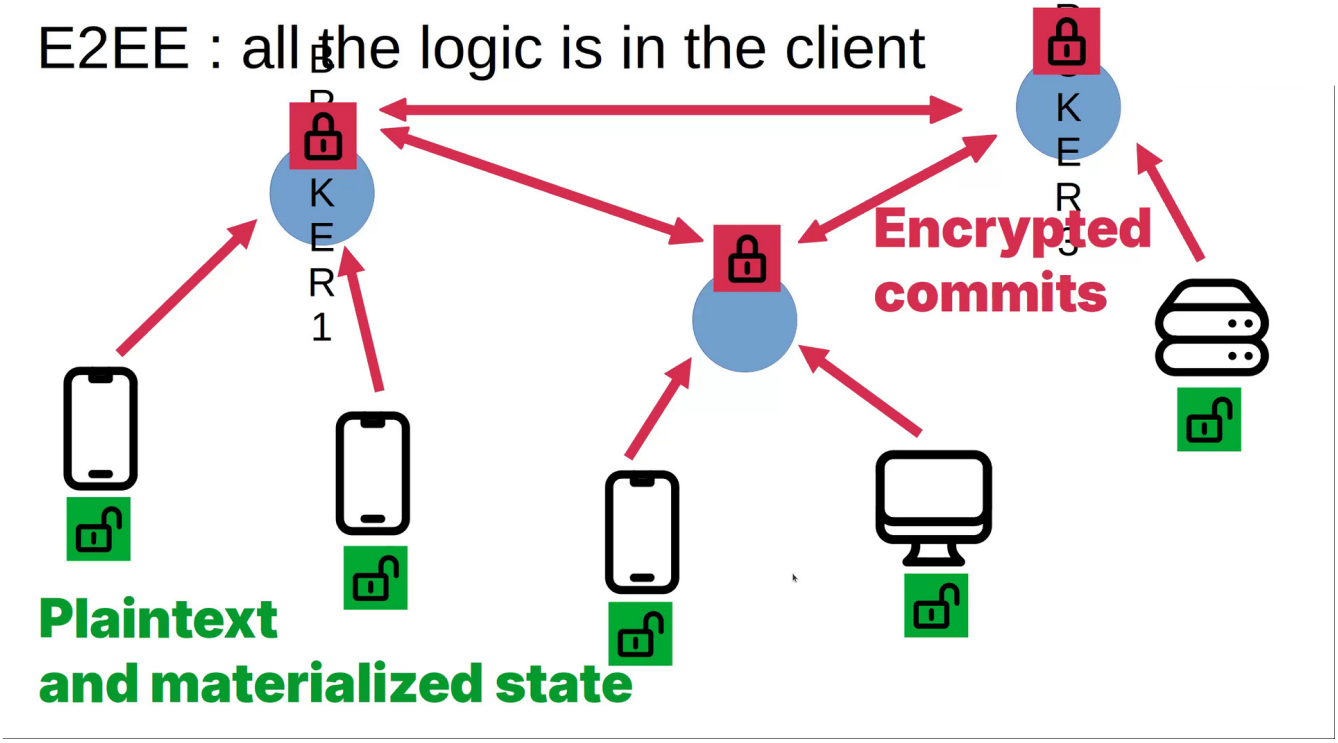
It is not purely peer-to-peer—not really peer-to-peer. That was in the title, so I'm sorry to disappoint you, but there is a good reason. Pure peer-to-peer means devices connect directly to one another; in a browser, for example, that would use WebRTC.

00:01:41 · Niko Bonnieure

That means synchronization is not mediated by a server. When you share something with somebody, their device needs to be switched on and online at the same time as yours so that you can synchronize. That is not always guaranteed. When you want to share with someone, you would prefer to send the data, end-to-end encrypted, to a server. Then you can go offline; that is fine.

00:02:16 · Niko Bonnieure

When the other person comes online, they synchronize through the server. We call these servers brokers. In this diagram, the blue circles are brokers. They talk to each other in a federation of brokers, and they only exchange end-to-end encrypted blocks—encrypted data.



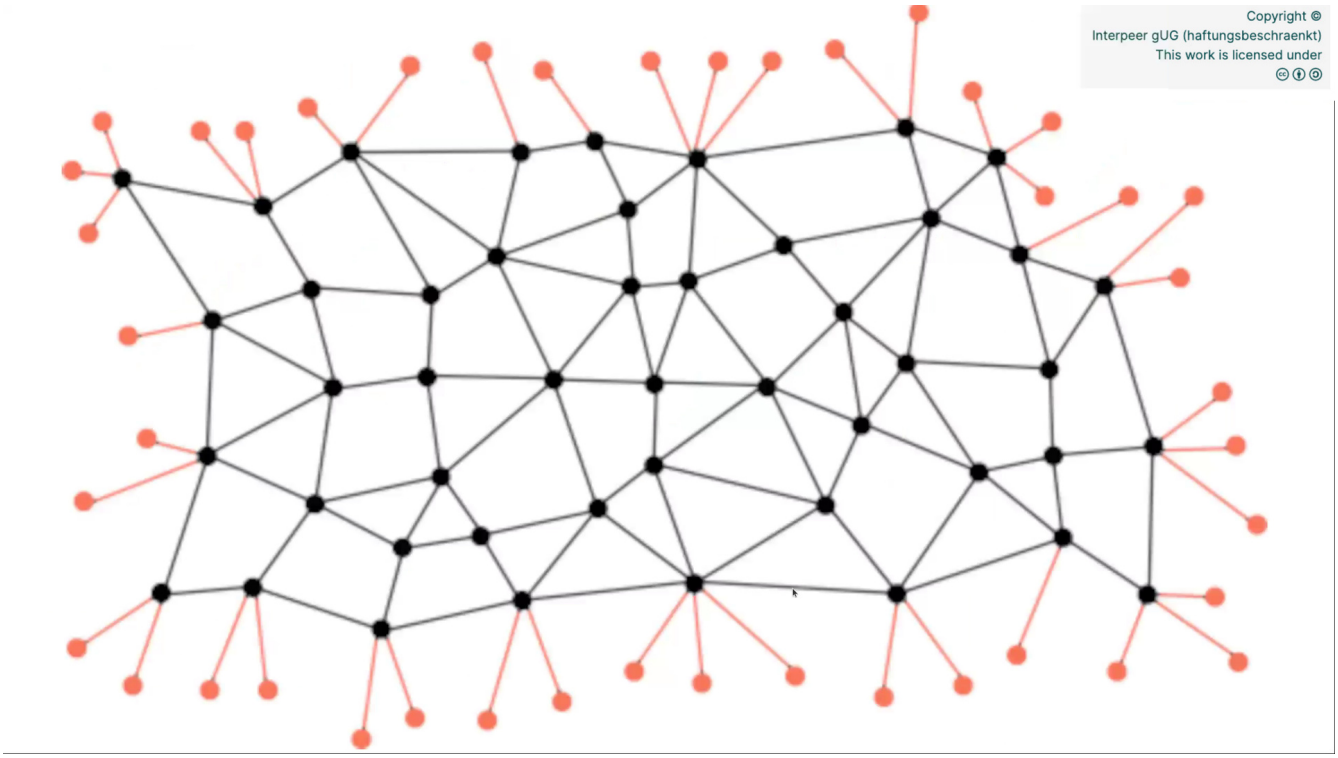
Encrypted data and client-side state · 00:02:25

00:02:48 · Niko Bonnieure

The users' devices, shown below in green, have access to the data. They encrypt and decrypt on the device. We also say that they materialize the state. This is a CRDT concept: after decrypting, they take the CRDT updates and aggregate them. We reconstruct the document locally, one update at a time.

00:03:21 · Niko Bonnieure

This is what the network would look like when deployed. NextGraph is not used in production yet. These black points would be the brokers. Devices connect to brokers, but they do not connect directly to one another. That is what I wanted to make clear.



Broker network topology · 00:03:24

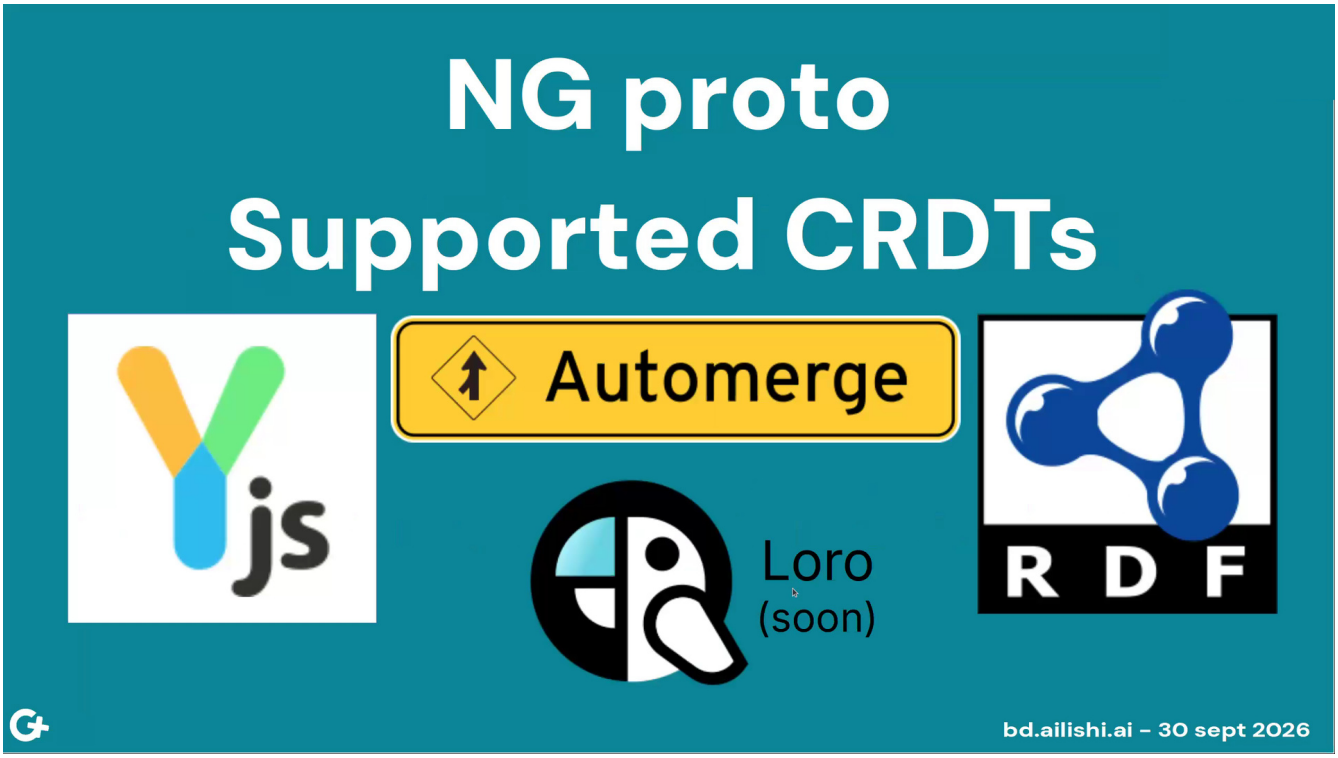


CRDT-agnostic synchronization · 00:03:43

A CRDT-agnostic protocol

00:03:45 · Niko Bonnieure

The protocol can synchronize any kind of CRDT; it is CRDT-agnostic. Different CRDTs can be transferred and synchronized. For now, we support Yjs, Automerge, and a special CRDT for RDF that we developed. RDF is a data description format also used in the Solid project, which I think another speaker will talk about soon in this series.



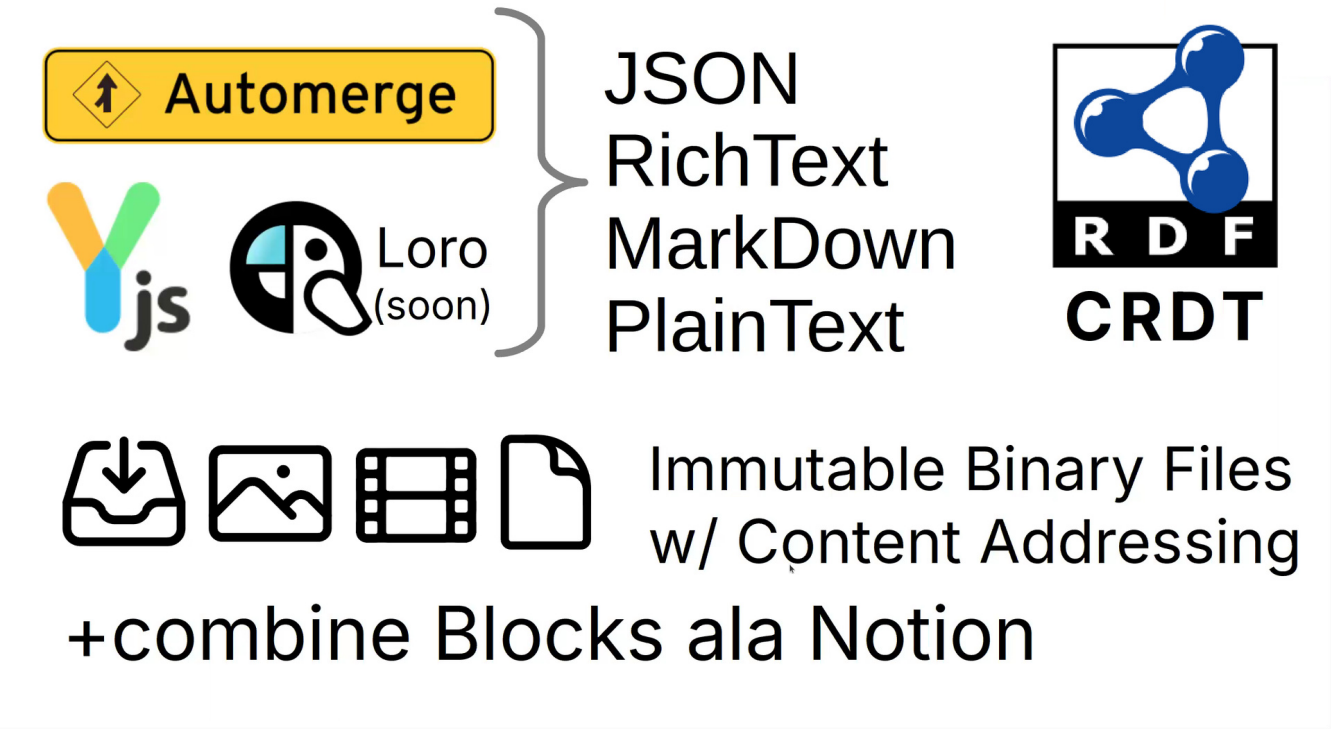
Supported CRDTs · 00:03:58

00:04:23 · Niko Bonnieure

We support RDF too, and I will say more about it later. We have planned to add Loro for two years. I know it comes from China and I know the developer; it is a very cool project, but we have not integrated it yet. Adding new CRDTs would be easy, because the protocol does only the minimum needed to transport a CRDT. I will explain that later. Basically, it preserves the causal ordering—the partial ordering—of the updates.

00:05:09 · Niko Bonnieure

So, Loro and more. We are now adding another CRDT specifically for spreadsheets, which takes advantage of this flexibility in the NextGraph protocol. The three generic CRDTs—Automerge, Yjs and Loro—handle structured data like JSON. They can also store rich text, Markdown and plain text, especially Yjs.



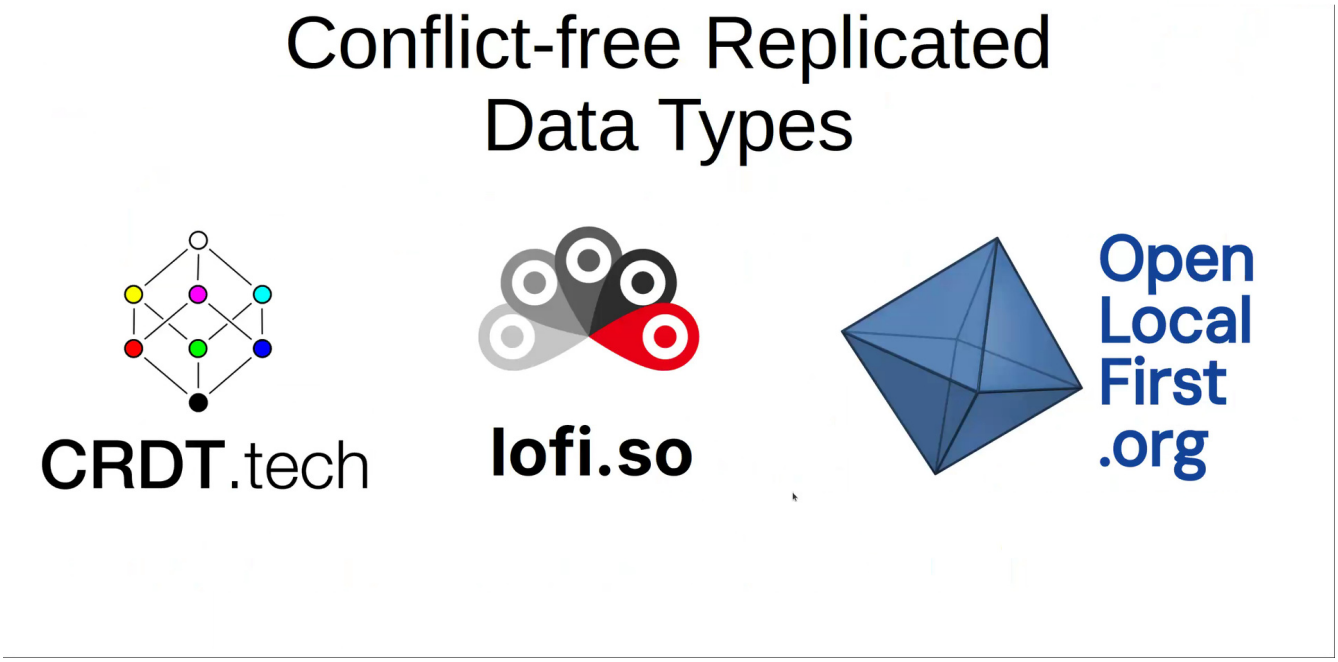
Data types and binary files · 00:05:22

00:05:43 · Niko Bonnieure

The NextGraph protocol also lets you synchronize immutable data, such as binary files. We use content addressing: if you upload the same file twice, it will not be saved twice, because we hash the content and know that we already have it. As you can see, it is a document-based system, but it is also database-oriented. It combines both.

00:06:14 · Niko Bonnieure

You have documents, and all the data stored in those documents together makes a database. The documents themselves can be combined, like in Notion or other block-based systems. You know what CRDTs are; the websites on this slide have more resources. Another important part of the protocol is cryptography and the access-control mechanisms based on it. That is why we call them cryptographic capabilities.



CRDT resources · 00:06:33

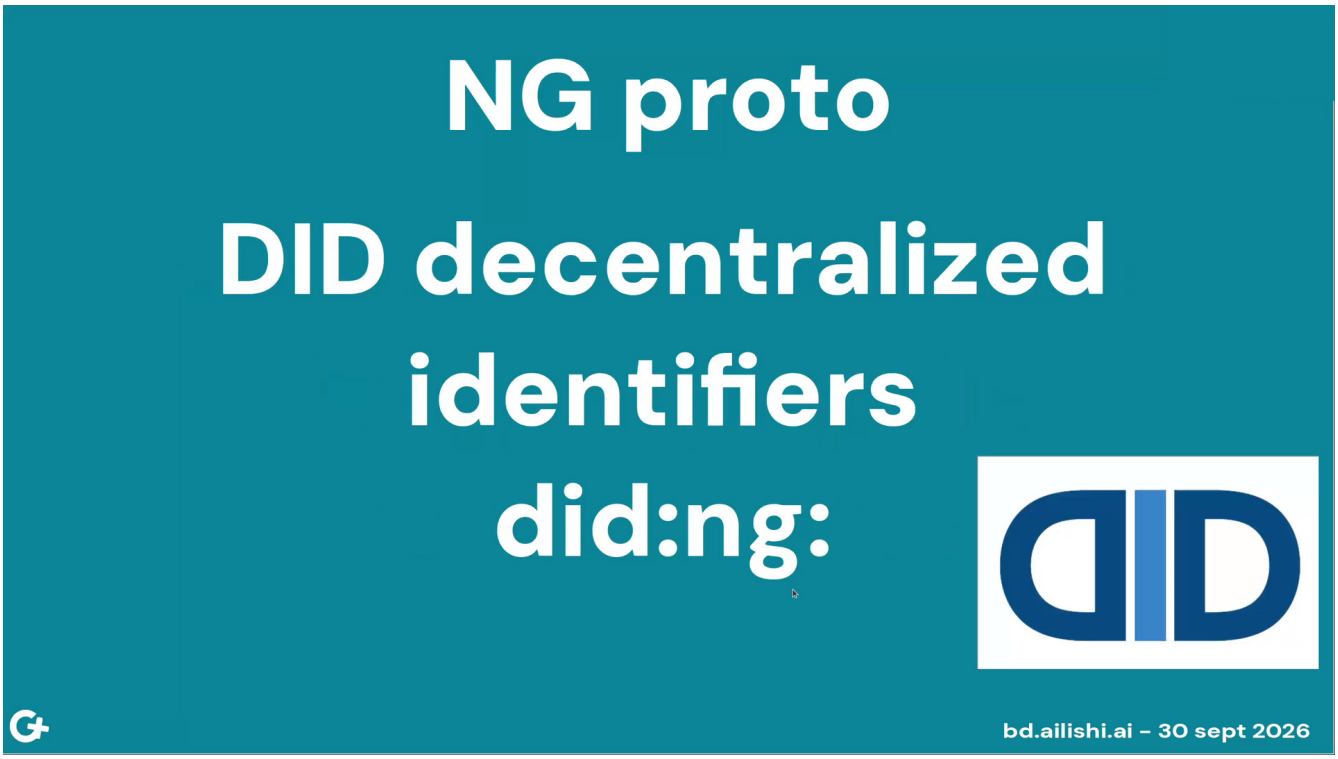


Cryptographic capabilities and access control · 00:06:42

Cryptographic capabilities

00:07:02 · Niko Bonnieure

I will go into more detail later, but if you do not have the decryption key, you receive an encrypted blob of data that you cannot do anything with. That is the read capability: you need the key to read. The write capability is based more on asymmetric cryptography. You need to prove that you are an editor, and this is done with signatures. These are complementary: reading uses symmetric encryption—do I have the symmetric key to decrypt?—and the rest is based on signatures. That is the general concept.



Decentralized identifiers · 00:07:47

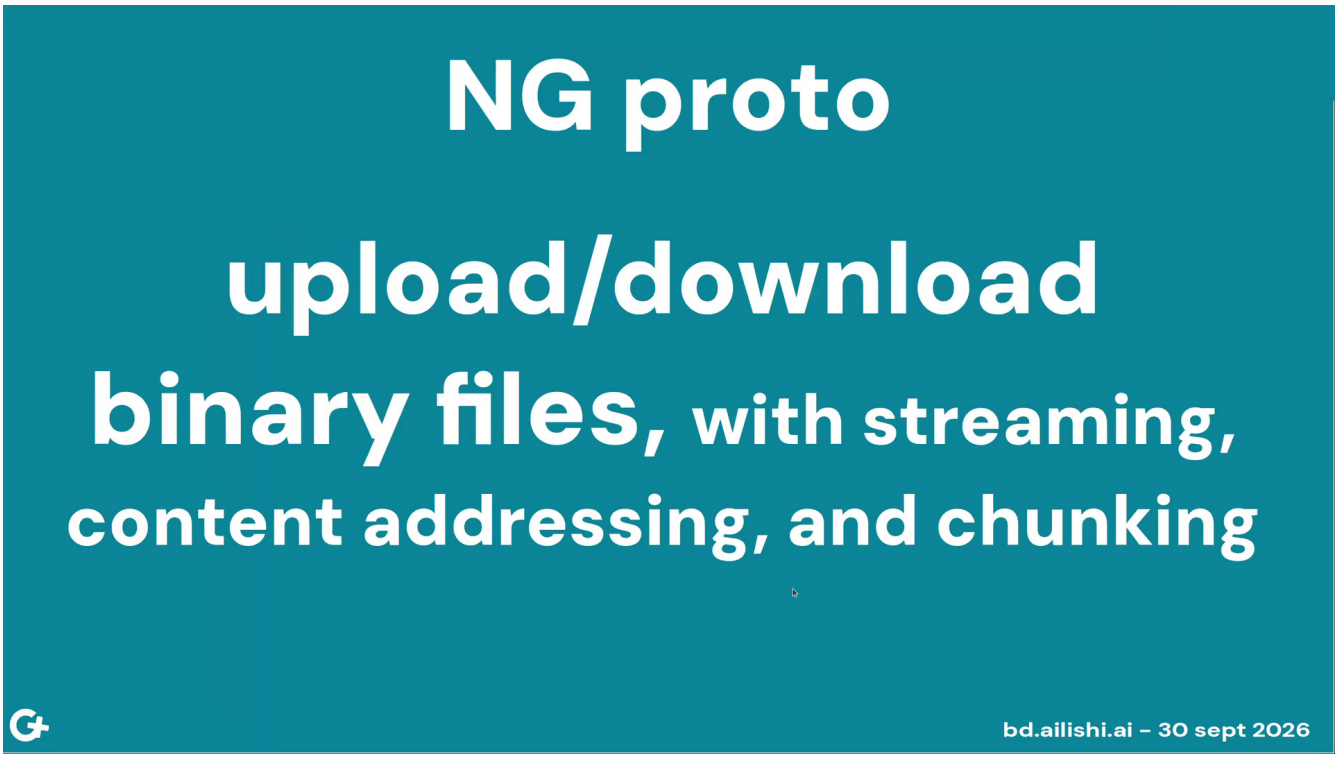
Decentralized identities and durable IDs

00:07:48 · Niko Bonnieure

Because everything is decentralized, there is no single point of failure in this protocol. For example, we have removed the dependency on domain names. Servers and devices know one another only by their IP addresses and identifying public keys. Documents and user identities have unique decentralized IDs. We use the W3C standard called DID.

00:08:23 · Niko Bonnieure

There are many DID methods out there. Ours is called ng, for NextGraph. With it, you can refer to any document in the database or any user. After ng comes the ID, a very long string of characters: a public key, along with more information about the data's location encoded in the URI.

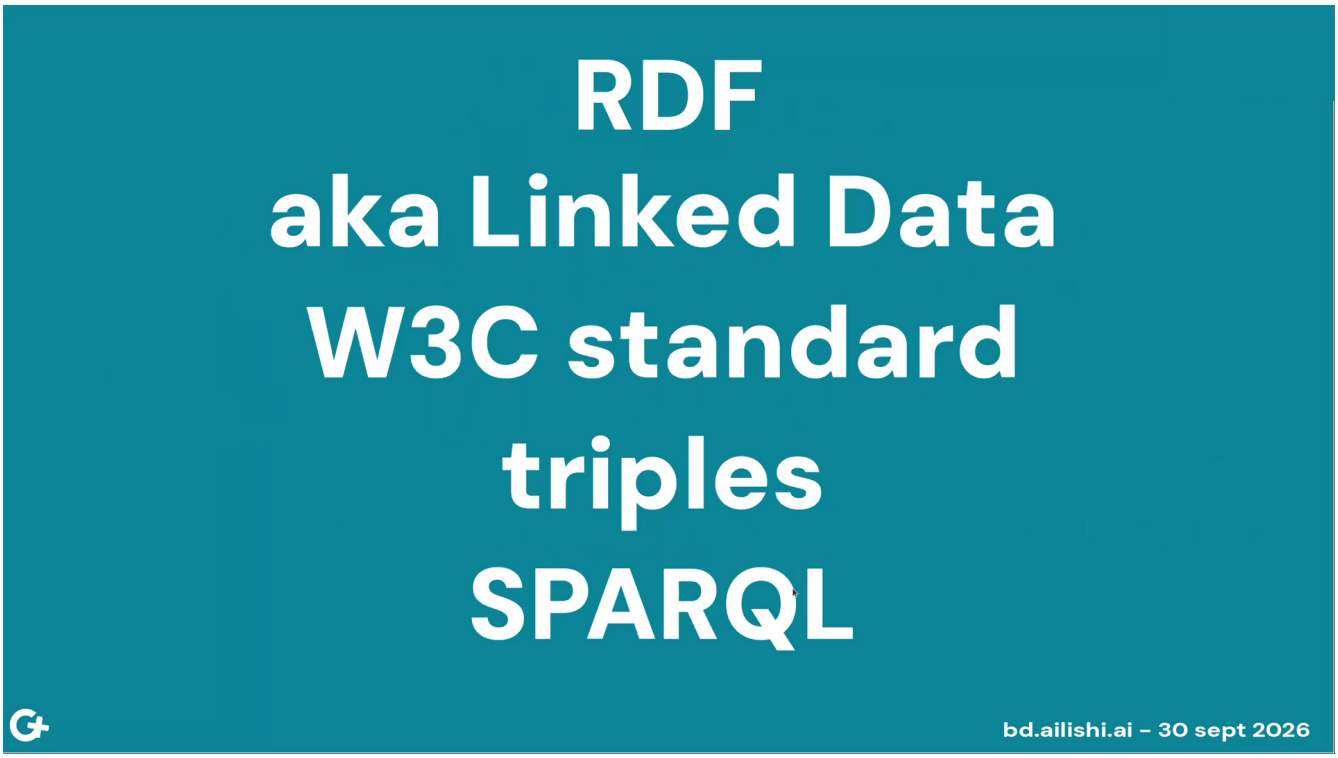


Chunked binary file transfer · 00:08:56

RDF, linked data and SPARQL

00:08:57 · Niko Bonnieure

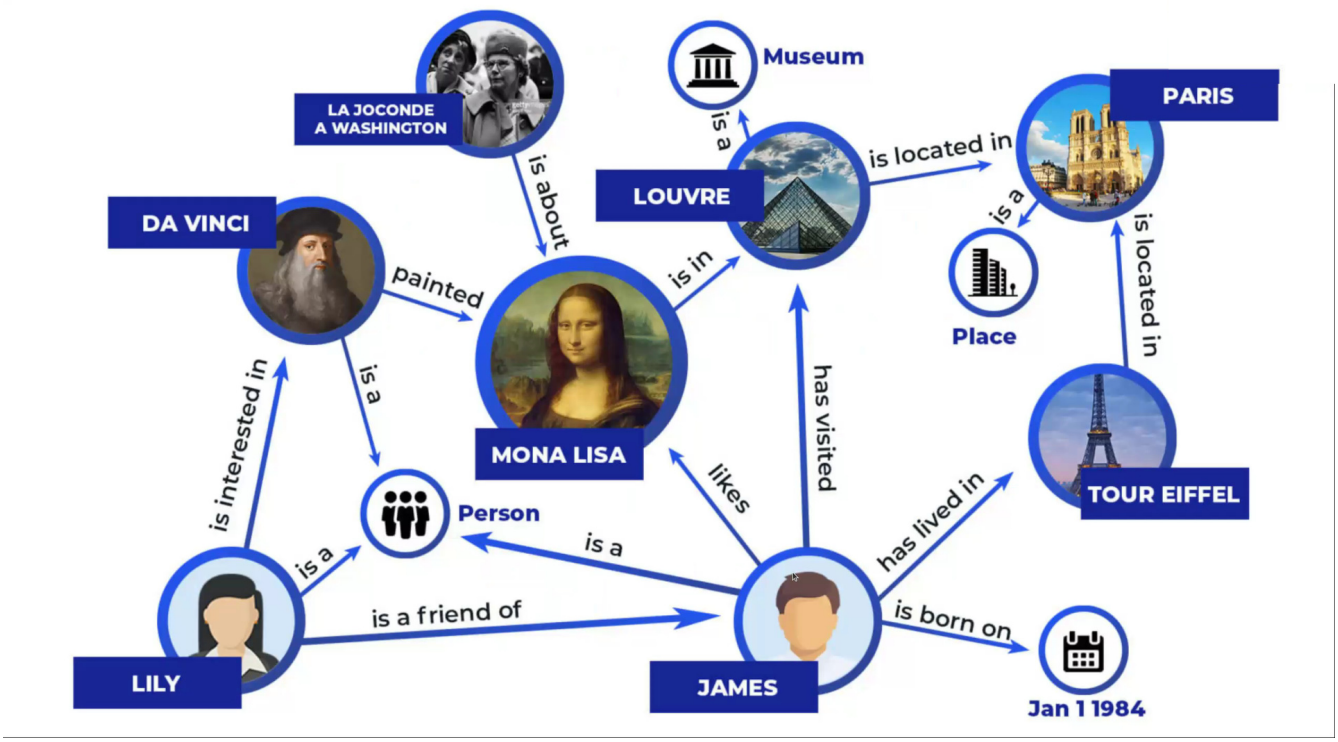
Now I will talk a bit about RDF, a data format I really like. In addition to JSON, it complements what we offer. It is also known under the terms linked data or semantic web. Basically, it stores data as triples inside graphs. It is a graph database.



RDF and Linked Data · 00:08:59

00:09:30 · Niko Bonnieure

A triple is the smallest unit of data you can save. It is a bit like a key-value pair in a key-value store, or a property. It can also encode relationships between objects. You can query it with SPARQL, which is similar to SQL. This is a very schematic representation: you have objects, and RDF encodes properties on those objects or relationships between them.



Objects and relationships in a graph · 00:09:53

00:10:08 · Niko Bonnieure

At the bottom right, there is a person called James with a birth-date property: 1 January 1984. That is a property on James. Paris has a type, a place; that is a property of Paris. There are also relationships between these objects: “has visited,” “painted,” or “is interested in.”

00:10:39 · Niko Bonnieure

You can connect the dots like this. These relationships are typed and have labels; you can even attach properties to relationships. That is what RDF lets you do. I like the idea of keeping a graph of all your data and connecting it to other people's data, referencing others. It brings interoperability between databases scattered across the internet.



Interoperability and malleable software · 00:11:02

Malleable software and data portability

00:11:10 · Niko Bonnieure

It also enables malleable software: you can change the editors and viewers you use for your data. You decouple data storage from applications. If I do not like the software, I can switch it and still keep my data. Where do I keep it? Locally, because in NextGraph and local-first software, the data is always kept locally. RDF also gives you automatic joins.



Automatic joins · 00:11:41

00:11:44 · Niko Bonnieure

Unlike SQL, you do not need to think about foreign keys. The data has global IDs, the did:ng URIs I mentioned. They are globally unique and durable: they do not change even when the data moves. If it is hosted on one broker and you move it to another, the IDs remain the same. That is different from what we are used to on Web 2.0, where URLs depend on domain names. Data moves, the domain changes, the URL breaks, and you need a redirect. Here we have removed all of that.



Global identifiers for records · 00:11:48

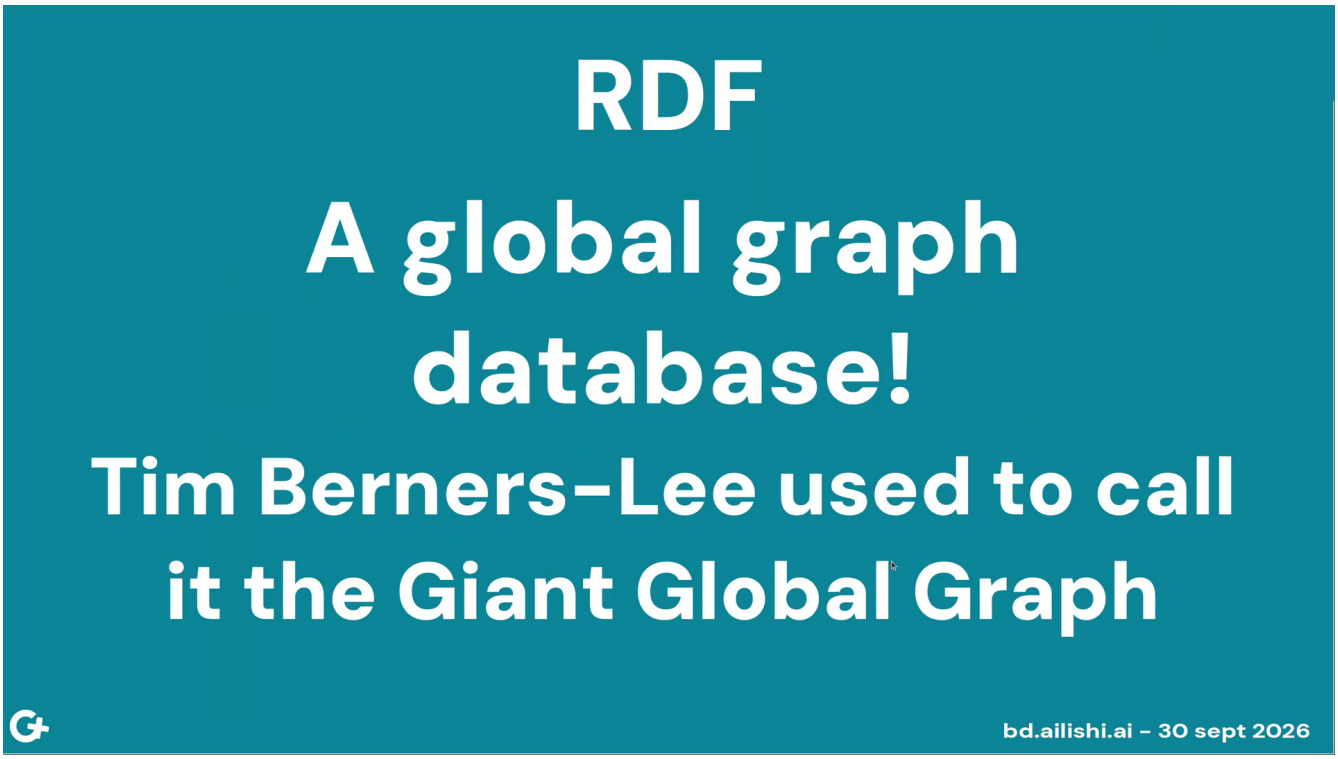


Linking and referencing documents · 00:12:29

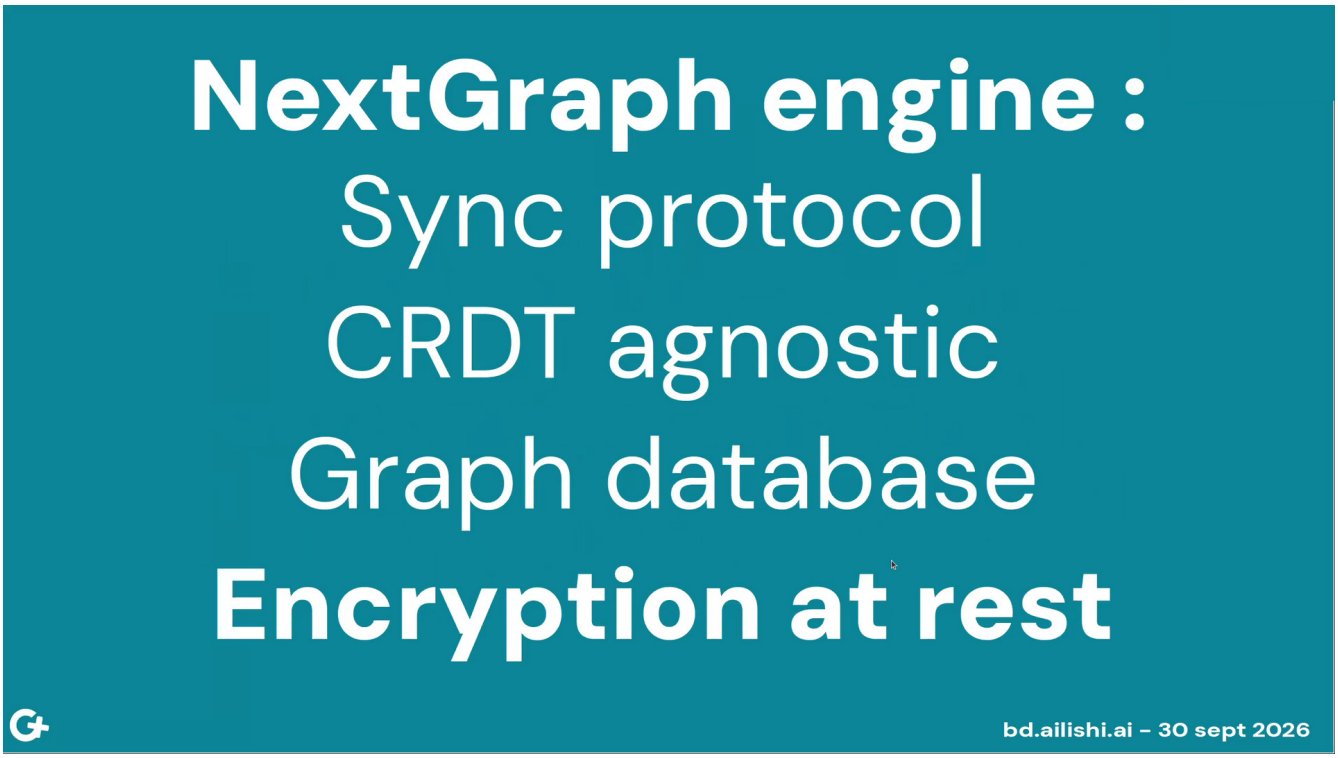
Encryption at rest and the SDK

00:12:30 · Niko Bonnieure

As I explained, it is a database. I also mentioned Tim Berners-Lee, who invented the web and is the father of linked data. So we have seen a CRDT-agnostic synchronization protocol and a graph database. There is encryption at rest as well as end-to-end encryption. When we receive data locally, we decrypt it, materialize the state and put the data into the local database. That local database is also encrypted at rest.



A global graph database · 00:12:33



NextGraph engine · 00:12:42

00:13:06 · Niko Bonnieure

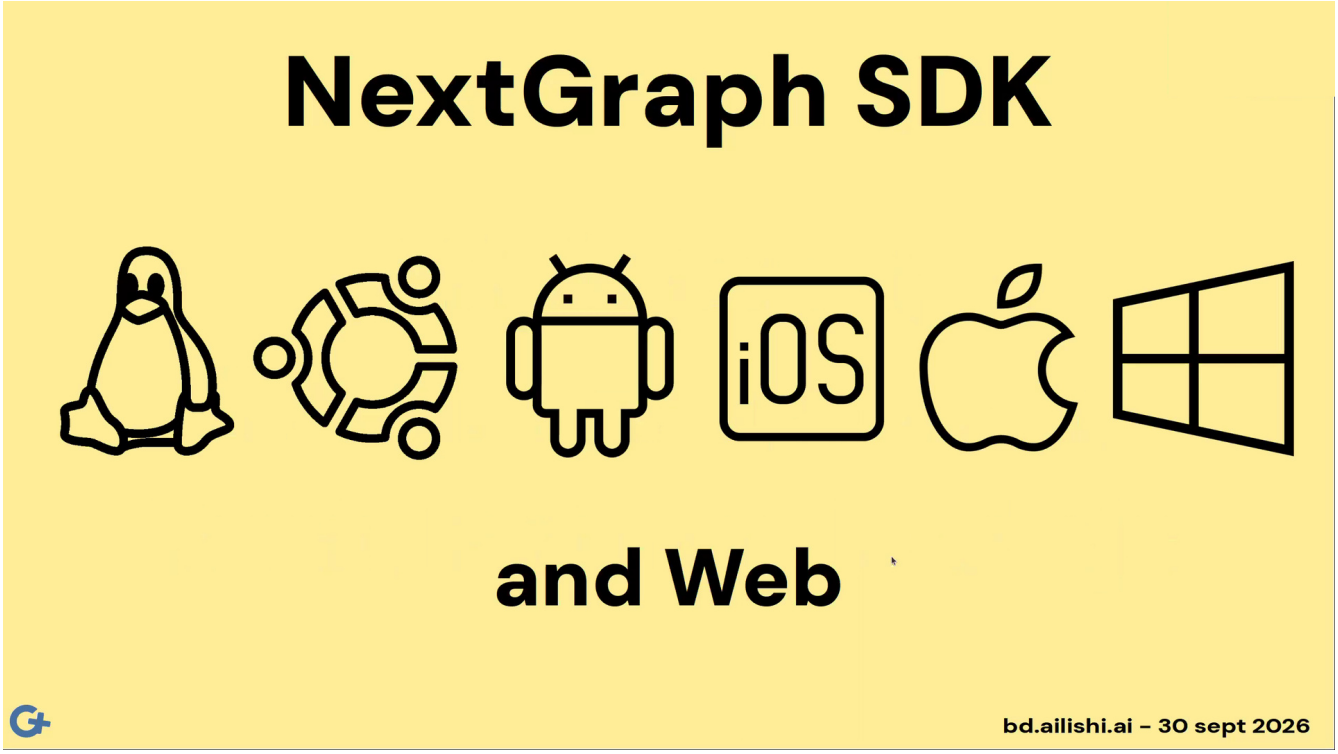
If the device is stolen or accessed illegally, it is still protected by the password the user enters when opening the app. Now, let me check the time and see whether there are questions. I can talk a little about the SDK. On top of the protocol, we have an SDK for developing applications. We try to hide the internal workings so you can focus on your app without worrying about what is inside.



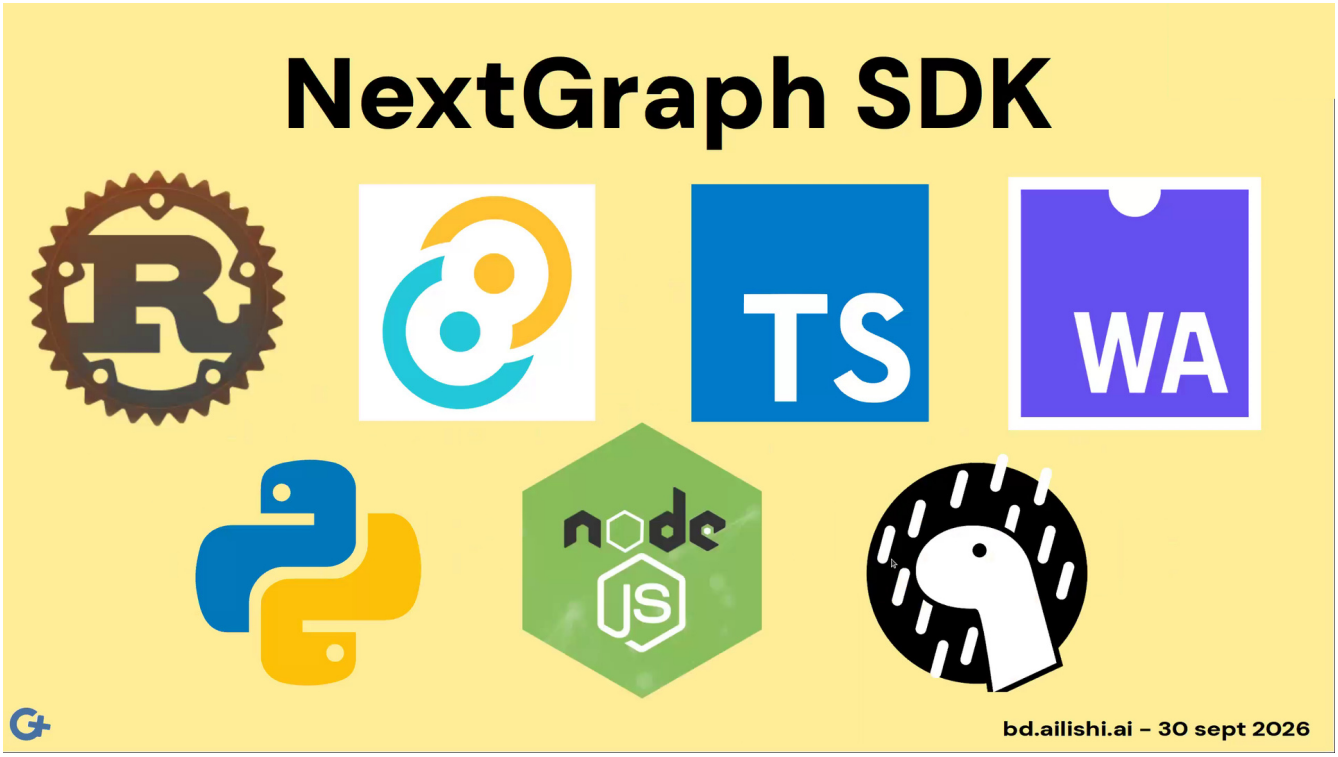
SDK and runtimes · 00:13:21

00:13:50 · Niko Bonnieure

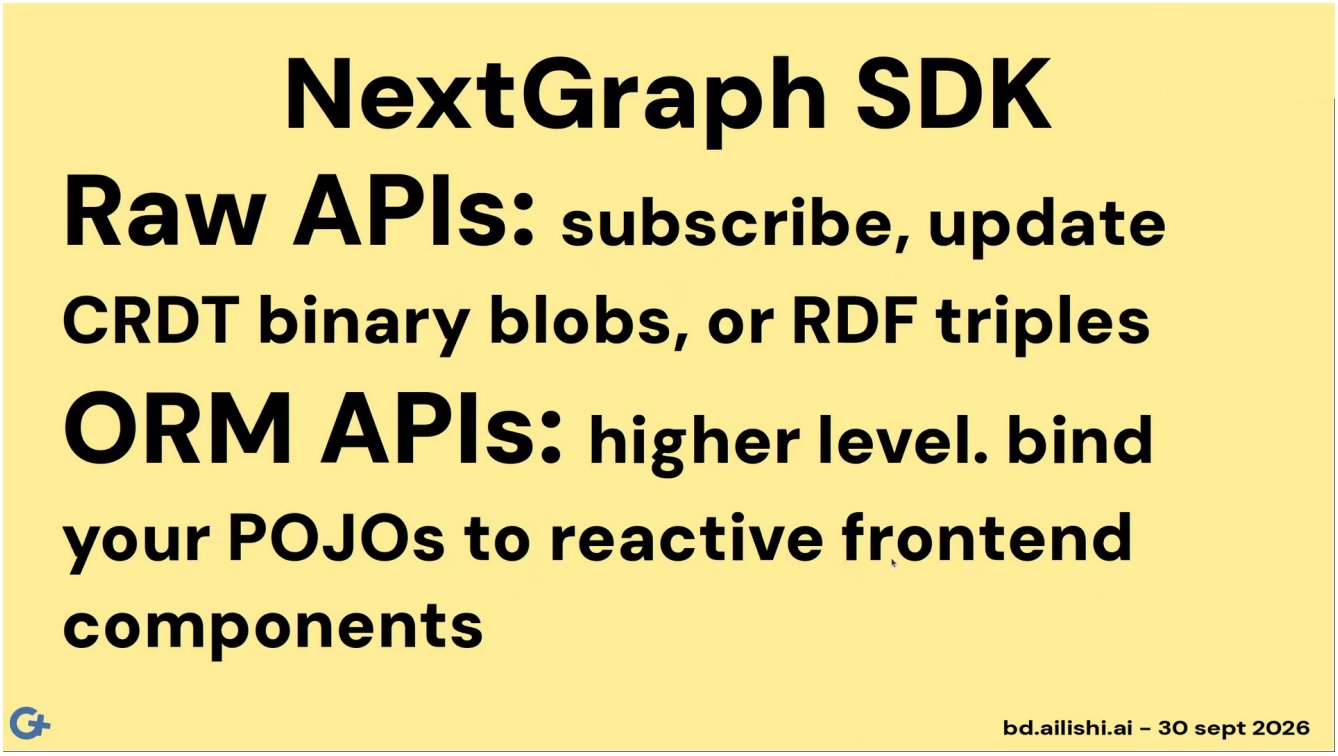
You can build web applications and native applications, because we use Tauri. NextGraph is written in Rust, with TypeScript for the front end. You can compile it for the usual platforms. We also have SDKs for Node.js and Python, for services, AI and things like that. These are the five platforms I mentioned, and the languages we currently support.



Supported platforms · 00:14:17



Languages and runtimes · 00:14:20



Raw APIs and ORM APIs · 00:14:24

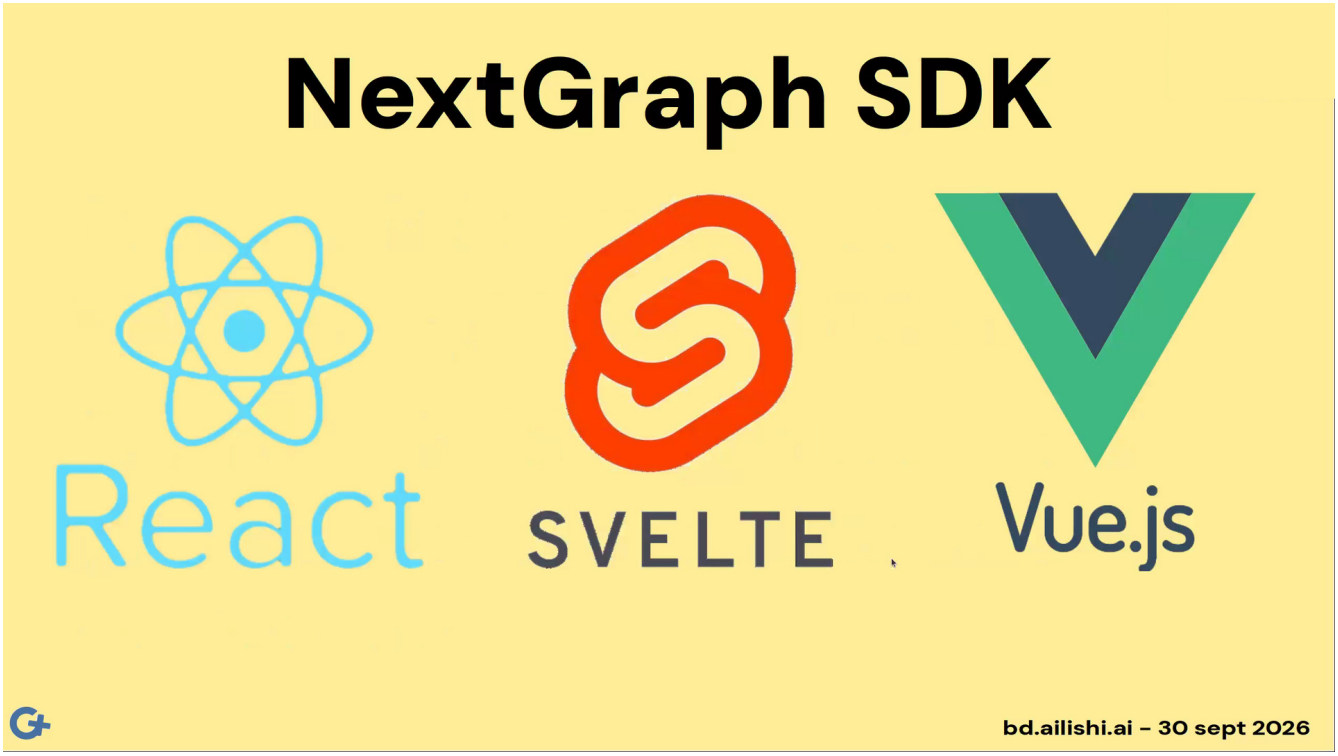
00:14:25 · Niko Bonnieure

The SDK has two levels of API. With the raw APIs, you subscribe to documents or create new ones. Subscribing means your data stays up to date. If an update happens somewhere on the network, in another tab, your local data in the browser stays current. It is always based on web technology; even the native apps are WebViews. You subscribe, and that keeps all kinds of data updated.

TypeScript ORM and reactive front ends

00:15:04 · Niko Bonnieure

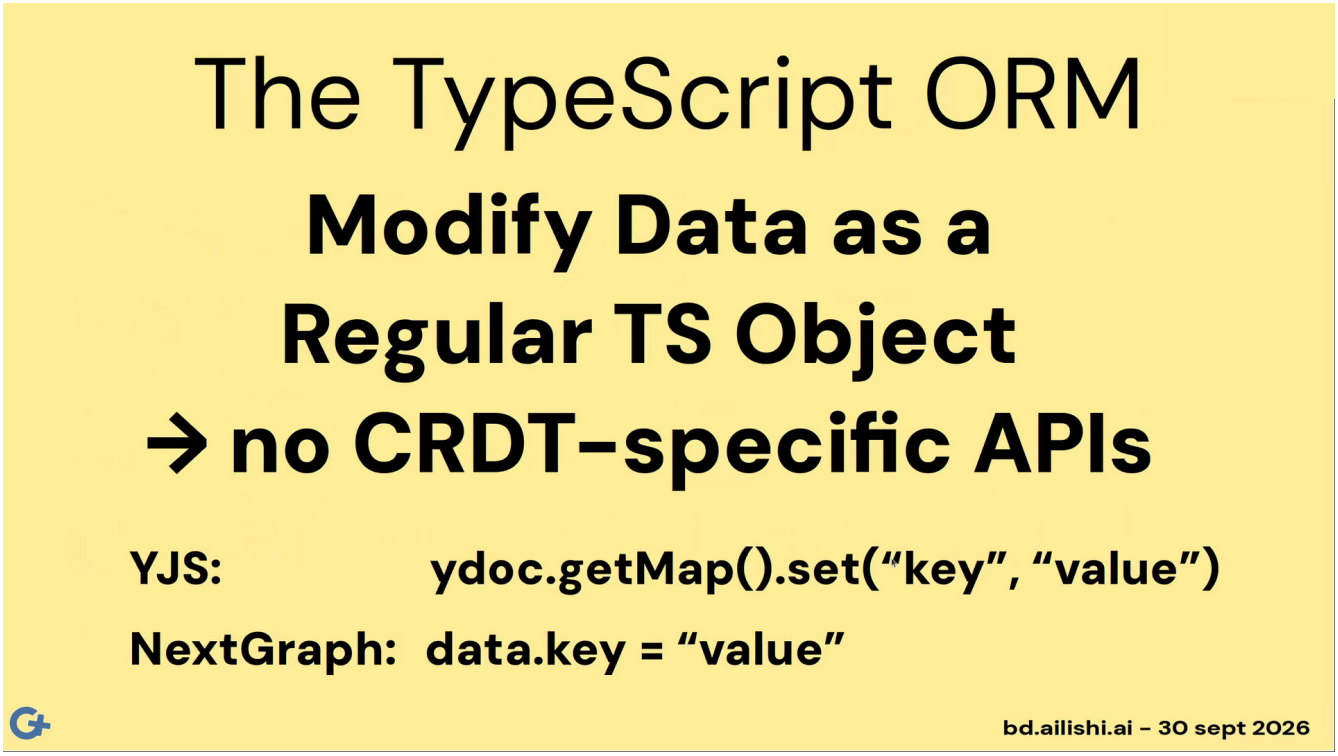
Whatever CRDT is behind a document, you can subscribe and receive its updates. We also have a higher-level API, the ORM. Data is mapped to TypeScript objects, with or without a schema, depending on the case. You can bind that data directly to your front-end framework, such as Vue, Svelte or React. We recently added SolidJS too, which should not be confused with the Solid project I mentioned earlier.



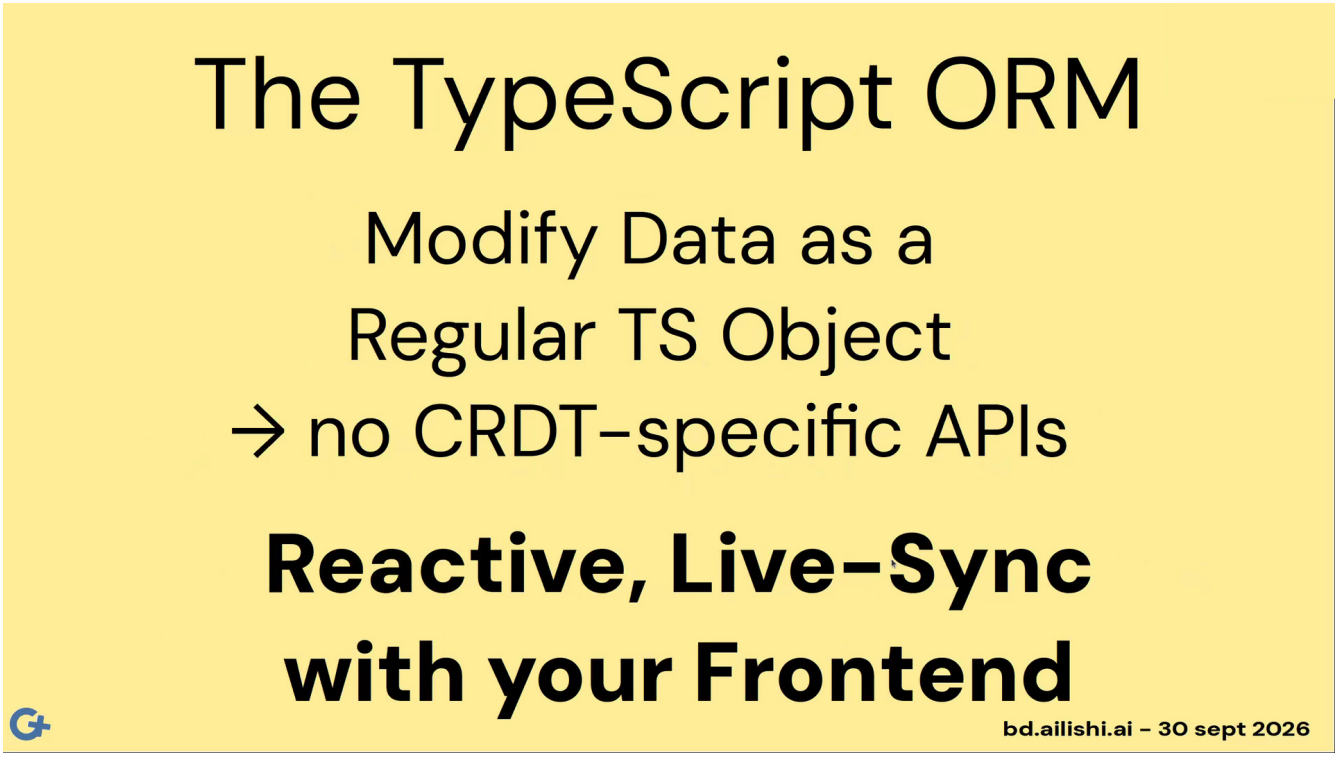
Frontend frameworks · 00:15:29

00:15:39 · Niko Bonnieure

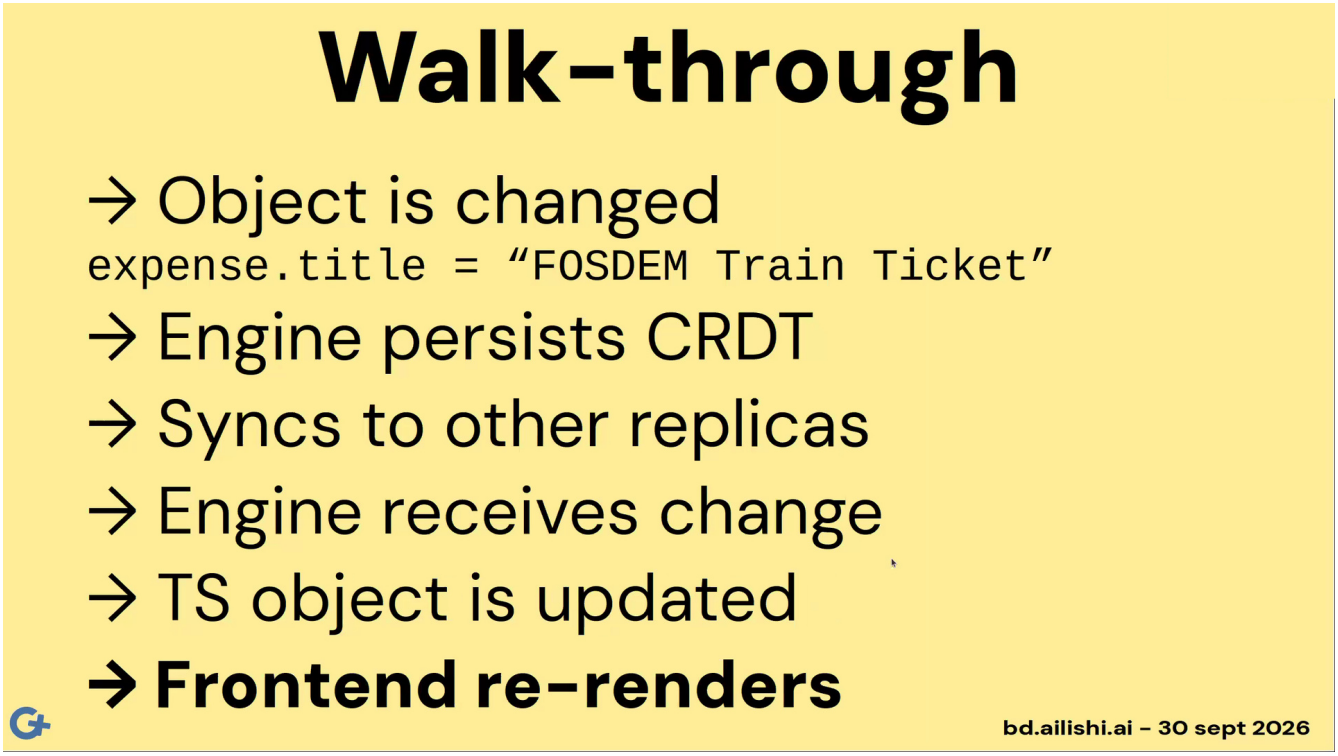
With the TypeScript ORM, you do not need to worry about CRDTs. In the example below, Yjs is one of the libraries I mentioned. Subscribing or modifying data requires knowing Yjs. Here, it is just plain TypeScript: object.key = value. You modify the data, and that immediately creates an end-to-end encrypted update. If you are online, it goes out over the network and others receive it. This gives you front-end reactivity and live document collaboration. You change a property, and the local engine persists it as a CRDT operation. It synchronizes if you are online; otherwise, it keeps the change until the next time you connect.



Modify data as a regular object · 00:15:40



Reactive frontend and live sync · 00:16:19



Data updates and synchronization · 00:16:30

00:16:48 · Niko Bonnieure

The other replicas receive those changes in their own local engines. Their objects update locally, and then their React, Svelte or Vue.js front ends update too. Here is front-end code using our two main APIs, useDiscrete and useShape. useDiscrete is for documents backed by Yjs, Automerge, or Loro if it were integrated. useShape is for RDF, and you give it a shape. I will explain that later. Whichever document type the data comes from, using it is the same: it is reactive. Here it is a React hook that you connect to your front end and templates. I mentioned a schema; that gives you types and type safety in TypeScript.

Frontend Code (React)

```
function ExpenseInputs() {
    // For discrete (JSON) CRDTs
    const { doc } = useDiscrete(documentId);
    const expenses = doc.expenses;

    // Or for graph ORM across all documents
    const expenses = useShape(ExpenseShape, scope);

    return expenses.map((expense) => (
        <input
            value={expense.title}
            onChange={(event) => {
                // Modifying data triggers rerender.
                expense.title = event.target.value;
            }}
        ></input>
    ));
}
```



bd.ailishi.ai – 30 sept 2026

React frontend code · 00:17:08

Schemas & Typing

For RDF Data



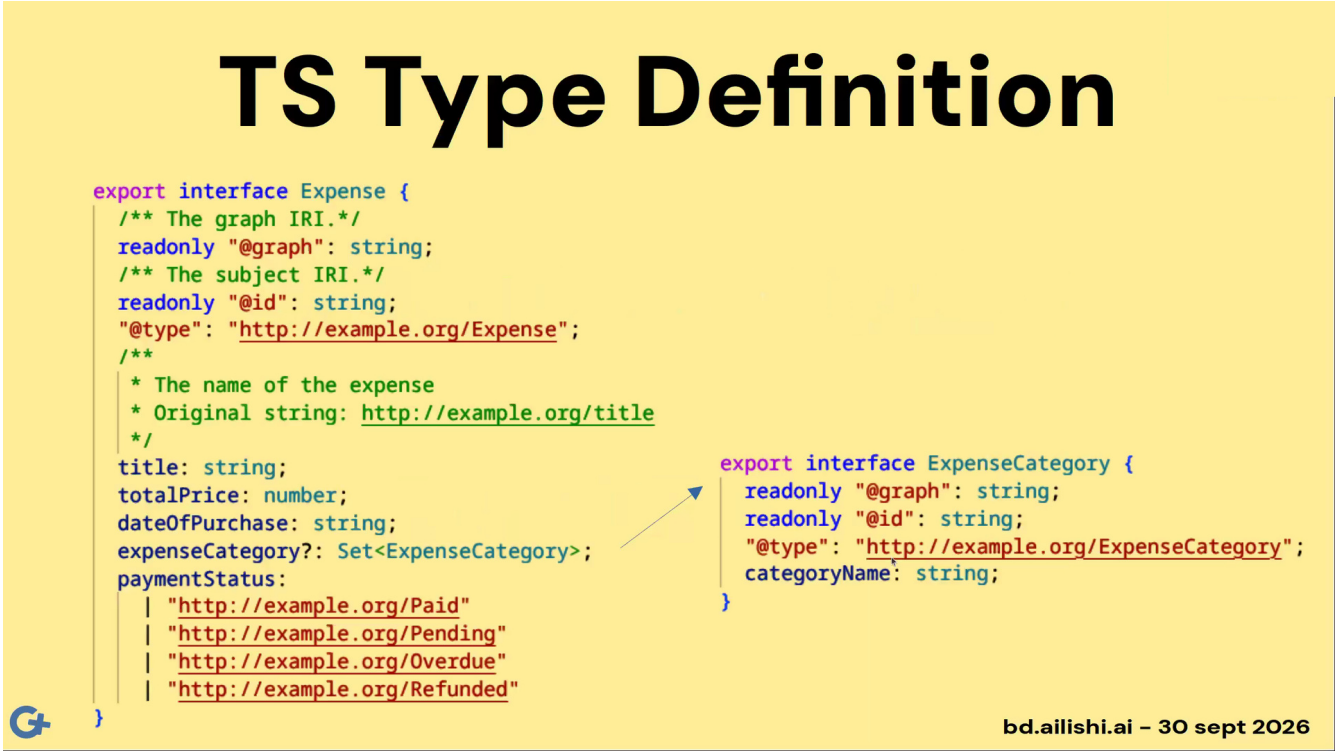
bd.ailishi.ai – 30 sept 2026

Schemas and typing for RDF · 00:17:47

ShEx schemas and type safety

00:17:56 · Niko Bonnieure

That only works with RDF documents, because the other documents are JSON and we do not support schemas for them yet. We use the ShEx format to describe the schema. It is a usual schema definition: the properties, subproperties and their types, including nested objects. You can find these examples in our Git repositories, in the expense-tracker example. It works, and it is continuously updated and tested, so it is a good place to start.



TypeScript type definitions · 00:18:06

SHEX Schema Definition

```
PREFIX ex: <http://example.org/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

ex:ExpenseShape {
  a [ex:Expense] ;

  ex:title xsd:string
    // rdfs:comment "The name of the expense" ;

  ex:totalPrice xsd:float
    // rdfs:comment "The total price" ;

  ex:amount xsd:integer
    // rdfs:comment "The number of items bought" ;

  ex:dateOfPurchase xsd:date
    // rdfs:comment "The date of purchase" ;

  ex:expenseCategory @ex:ExpenseCategoryShape *
    // rdfs:comment "The use category of the product" ;

  ex:paymentStatus [ex:Paid ex:Pending ex:Overdue ex:Refunded] ;
    // rdfs:comment "Status of payment" ;
}
```

```
ex:ExpenseCategoryShape {
  a [ ex:ExpenseCategory ] ;

  ex:categoryName xsd:string
    // rdfs:comment "Name of expense category" ;

  ex:description xsd:string
    // rdfs:comment "Human-readable description of category" ;
}
```



bd.ailishi.ai – 30 sept 2026

ShEx schema definitions · 00:18:21

Summary

Modify CRDTs in TS like any other object

ORM for RDF (and Automerge and YJS)

Type Safety for RDF

Have reactivity in React, Vue, Svelte, solid-js

Live-Sync with other users and devices



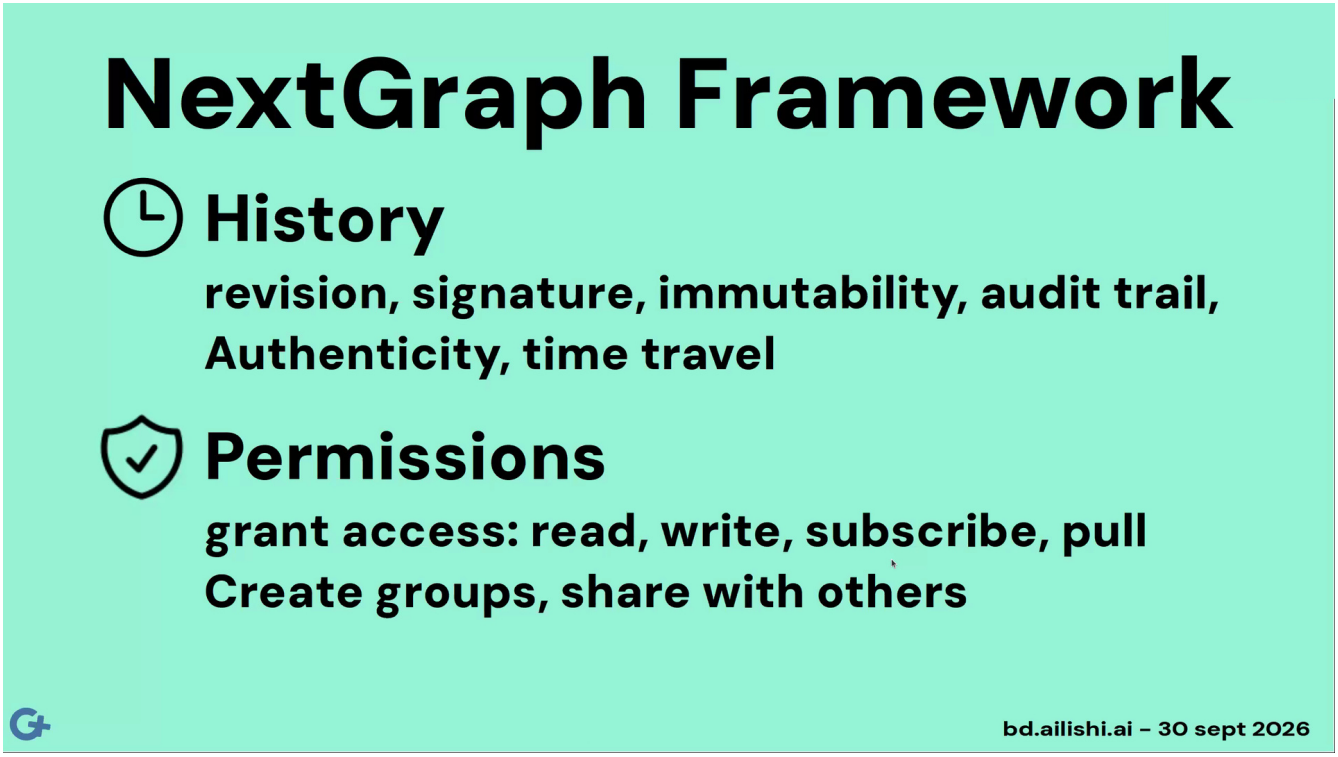
bd.ailishi.ai – 30 sept 2026

ORM feature summary · 00:18:25

History, permissions and reactive queries

00:18:44 · Niko Bonnieure

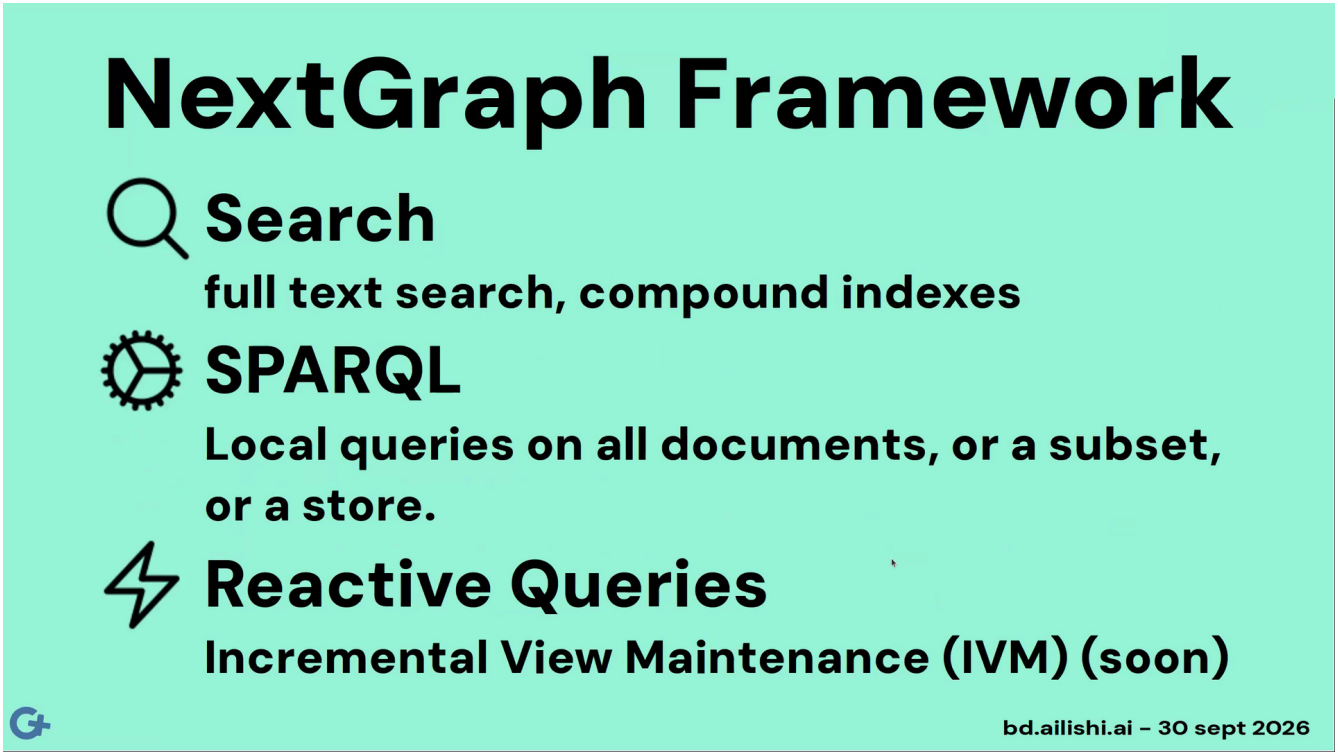
On top of the SDK, we are also working on a framework. When applications need common features, we provide higher-level APIs for them, such as document history. You can travel back in time and see a previous revision. There is also an audit trail: everything is signed.



History and permissions · 00:18:50

00:19:15 · Niko Bonnieure

Every commit is signed by its author. This history of changes is tamper-proof and immutable. To be precise, it is not really a log but a DAG, a directed acyclic graph. There are also permission mechanisms. You may not want to rebuild the interface for changing document permissions, so we can provide that if you need it. And, as I said, SPARQL gives you a local query engine.



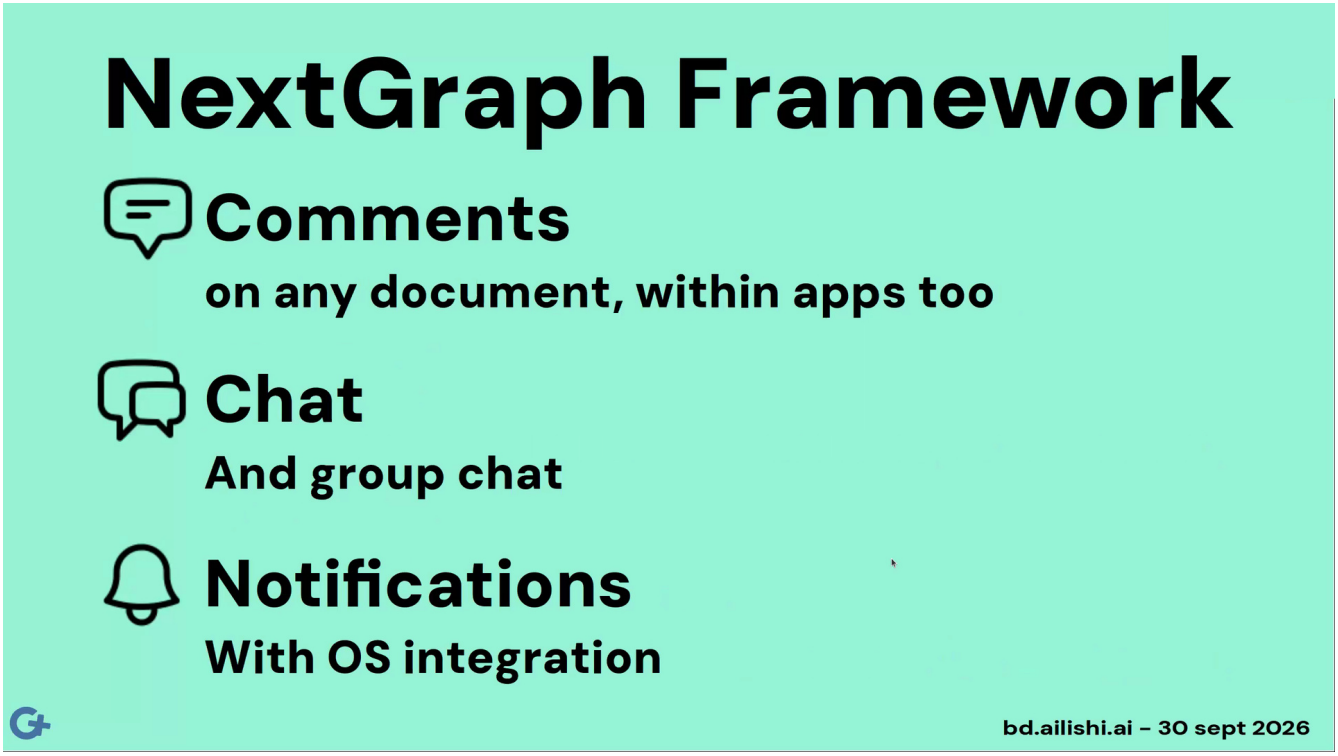
Search and queries · 00:19:44

00:19:50 · Niko Bonnieure

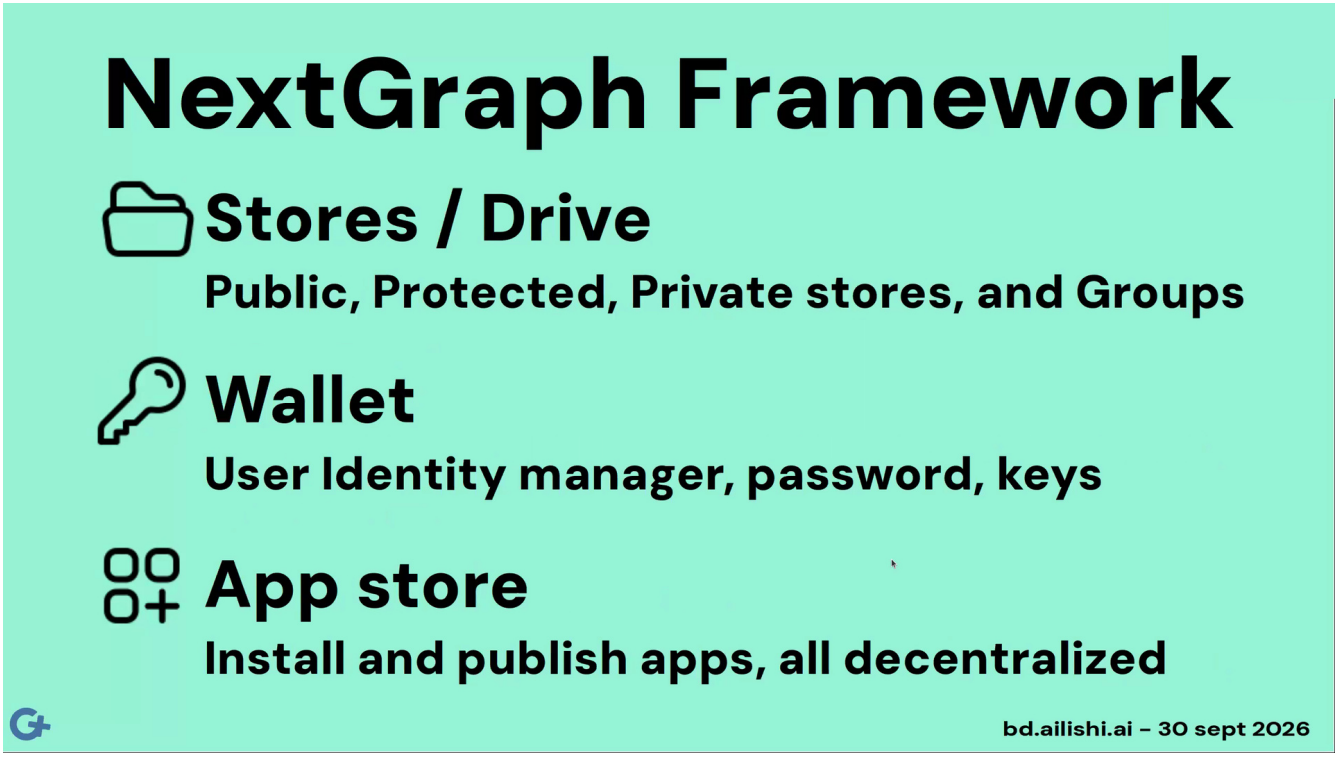
Reactive queries are a work in progress, but going well. You start a SPARQL query and keep a handle on it, so the query stays alive. First, you receive the initial results. If the underlying data changes while you are still displaying or using those results in a view, a callback gives you the updated results.

00:20:24 · Niko Bonnieure

For example, if a record is edited or deleted, you see that change automatically. Those are reactive queries. Comments, chats and notifications are other useful things the framework supports. Document sharing and saving data are the foundation; these extra APIs sit on top.



Comments, chat and notifications · 00:20:36



Stores, wallet and app store · 00:20:56

00:20:58 · Niko Bonnieure

The wallet is very important. It stores the master keys, is always kept locally, and is never seen by the server. There is also an app store and a drive. We have plans for decentralized, private social networks, using end-to-end encryption and providing a lot of privacy. If we have time, I can mention ACID transactions and smart contracts later. To recap, this is an open platform made up of brokers.

NextGraph Framework

Social Network

P2P web of trust private social network

Smart contracts

ACID transactions

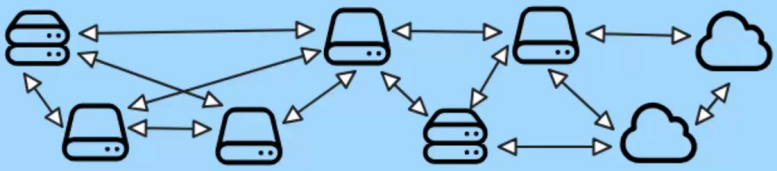
with strong consistency. Paxos coordination

bd.ailishi.ai – 30 sept 2026

Social network and transactions · 00:21:12

Recap: NextGraph stack

OPEN PLATFORM : a federation of brokers



At home/office, On Premise, Self-hosted, community hosting, hosters, integrators, edge, cloud

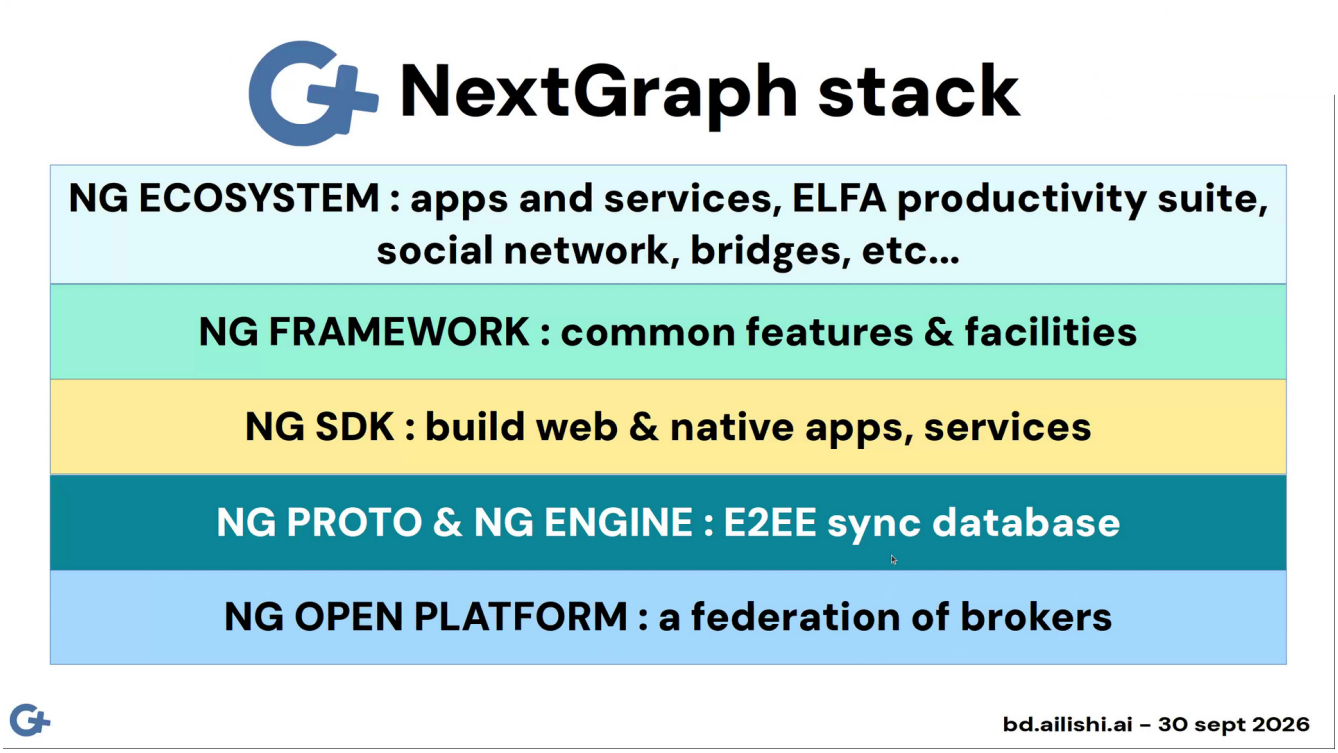
bd.ailishi.ai – 30 sept 2026

A federation of brokers · 00:21:27

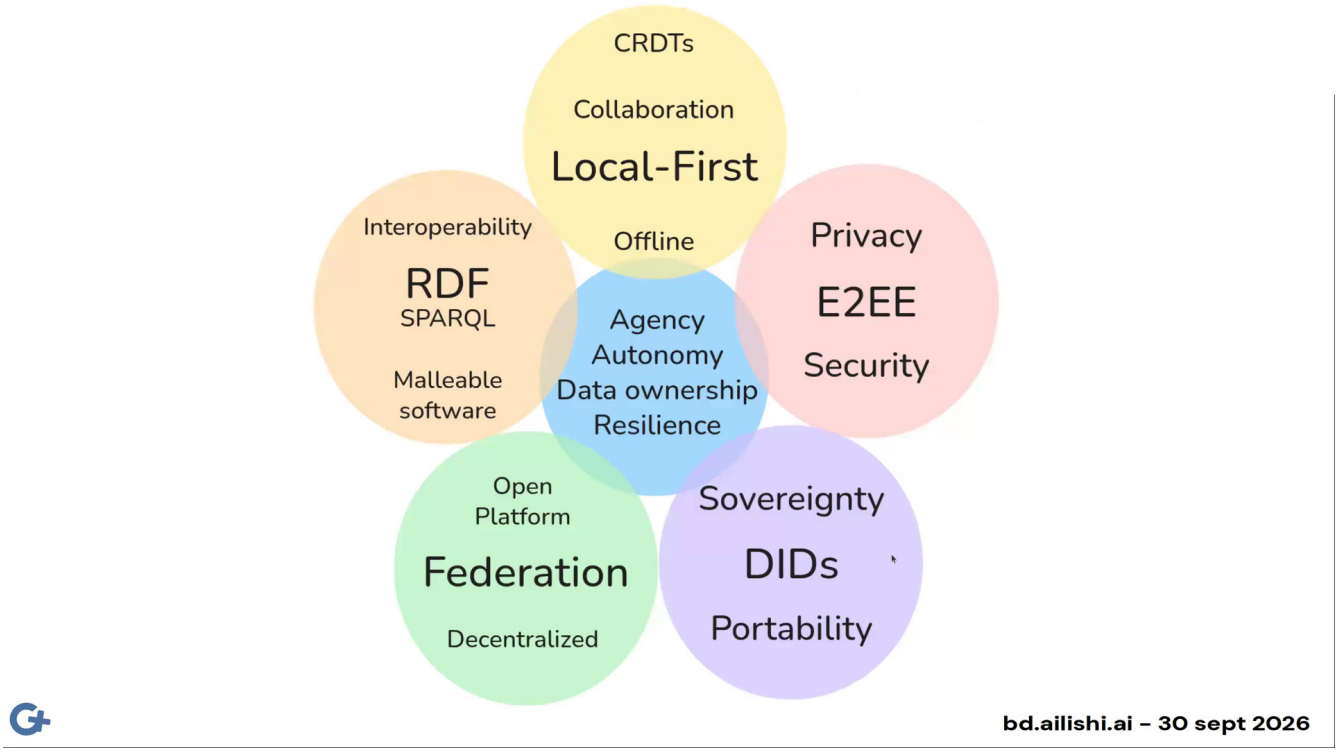
ELFA and the local-first ecosystem

00:21:31 · Niko Bonnieure

It is open source and self-hostable: you can deploy it in the cloud, on a rented VPS or on your own machine. There is the protocol, the SDK, the framework and an ecosystem of apps. I want to mention ELFA. I will skip these general properties and give you the PDF later. ELFA is a suite of apps we started working on four months ago. It is a complete workspace, trying somewhat to compete with Google Workspace, with the apps and documents you need: documents, spreadsheets, email, calendar, contacts and the rest.



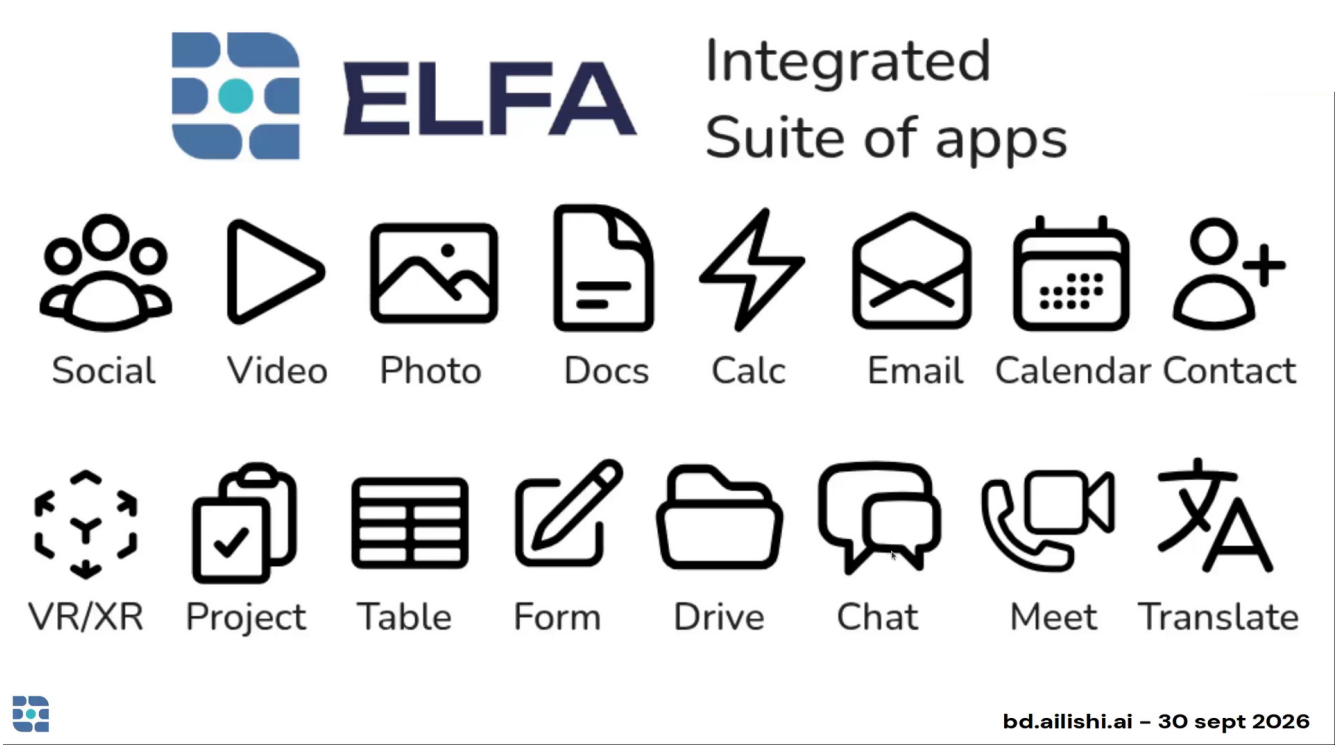
NextGraph stack · 00:21:39



NextGraph design principles · 00:21:47



ELFA consortium · 00:21:52



ELFA application suite · 00:22:06

00:22:14 · Niko Bonnieure

It is based on NextGraph, so it is end-to-end encrypted and local-first. It should be ready in a year or two, with an alpha version coming soon. These are the partners working with me on it.



Partner projects · 00:22:25

00:22:30 · Niko Bonnieure

Recently I also created a larger coalition with all the partners shown here; there is even a second slide. It is a new venture involving ELFA, NextGraph and these partners. They are not all using NextGraph, but there will be synergies. People will work with me to bring local-first to the general public, operating systems, mobile operating systems and more apps—to the wider open-source ecosystem.



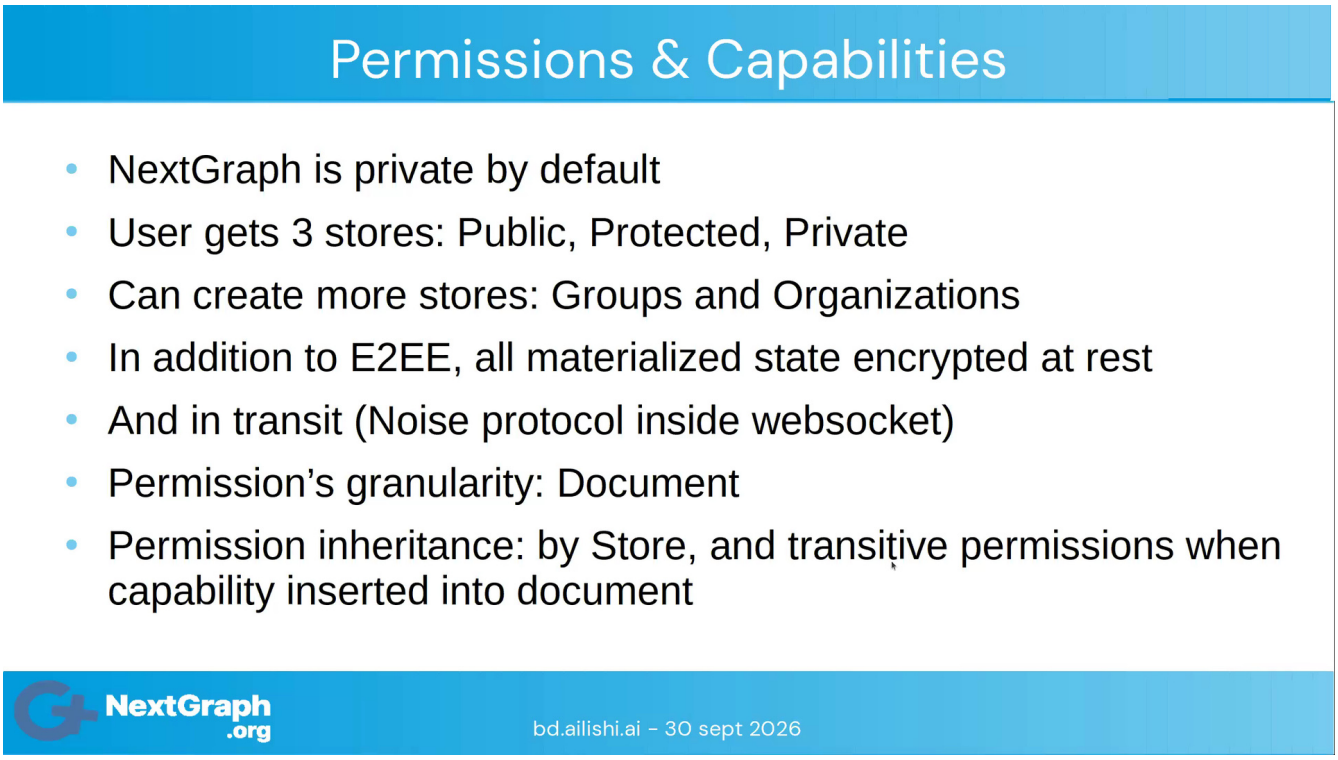
SUPER Stack coalition (1) · 00:22:31



SUPER Stack coalition (2) · 00:22:38

00:23:10 · Niko Bonnieure

We can now go into more detail on encryption, permissions and so on. I will pause and ask whether you have questions, and which parts you want me to dive into.



Permissions and capabilities · 00:23:13

00:23:30 · Host

Cryptography, perhaps?

00:23:32 · Niko Bonnieure

Yes, cryptography. I cannot find the chat. Do you see any questions there? I do not see where it is. Is there a chat here?

00:23:55 · Host

Yes, but I do not think there are any questions.

Stores and permission boundaries

00:23:59 · Niko Bonnieure

Perfect. Tell me if you see a question; you can interrupt me. I will go into capabilities and the permission system. All the data is organized into stores, which group documents together. A store is like a container. There are several types. A private store contains data that synchronizes only between your own devices, not between users.

00:24:34 · Niko Bonnieure

You keep your secrets there; you cannot share those documents with others. In the protected space, you can share documents with other people using different permissions. Then there is the public store, whose data anyone can read. You just give them the link. We do not check read permissions there, but we do check write permissions. Basically, only you can write, unless you delegate that permission to others.

00:25:07 · Niko Bonnieure

Those are the three stores. In addition, we have group stores. You create a group and invite people into it. Members can edit and create documents in the group. All these stores have a document hierarchy: you can put folders and subfolders inside and organize your documents as you want. As I mentioned, we also encrypt data in transit.

00:25:40 · Niko Bonnieure

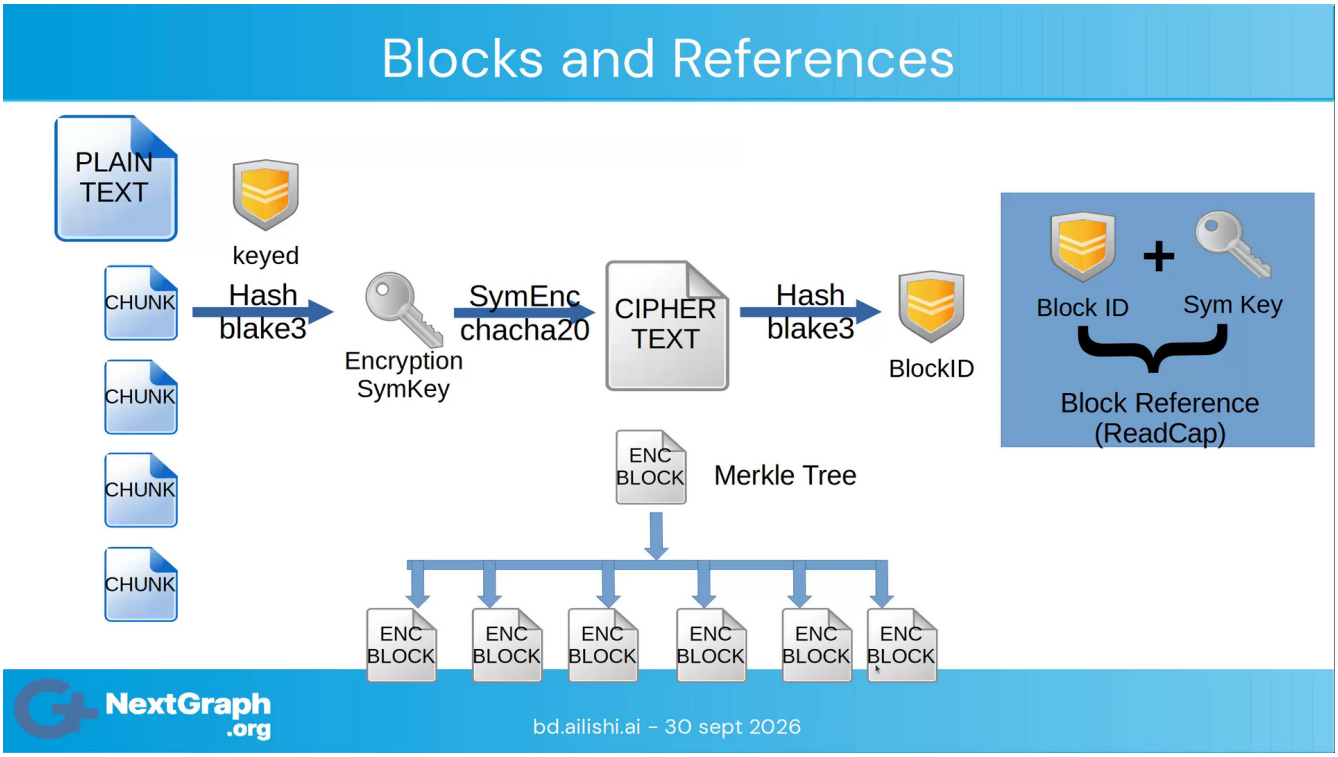
We have three different kinds of encryption. We use the Noise Protocol for all communications between devices and between brokers.

00:25:52 · Niko Bonnieure

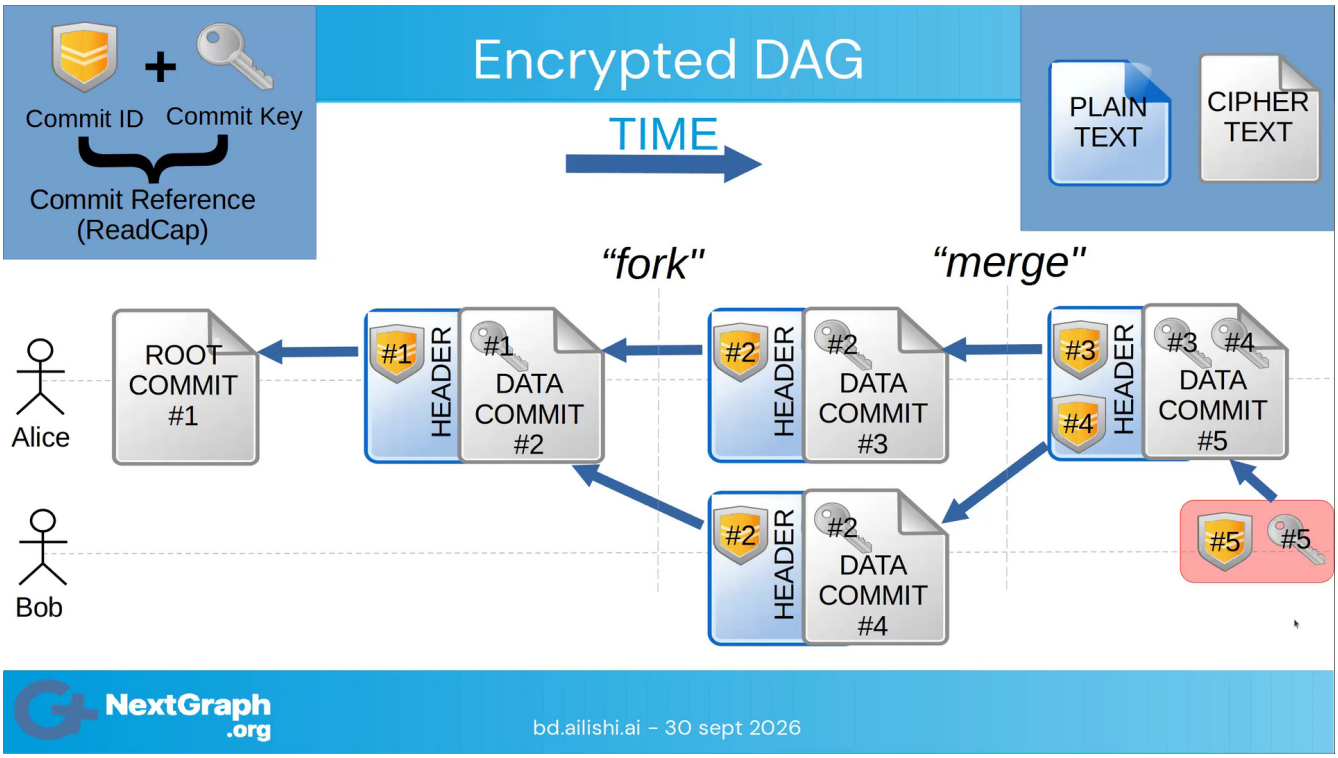
Permission granularity is at the document level, but not exactly. Write permissions are at the document level: you specify who can edit a document. Read permissions are more granular, down to what we call a branch. You can divide the data inside a document into branches, so some parts of a document are readable and others are hidden.

00:26:25 · Niko Bonnieure

That does not apply to write permission, which remains at the document level. There is inheritance: documents inherit permissions from their store, but you can also override that and change permissions on a particular document. Now we get into the technical details of encryption inside the protocol. I think this is important to explain. Let me start with the second slide.



Blocks and references · 00:26:44



Encrypted DAG · 00:26:59

Encrypted blocks and commits

00:27:01 · Niko Bonnieure

This is how we do end-to-end encryption and connect it with CRDTs. I think this is very important when combining the two. Every update is atomic: we store it in what we call a block, and that block is encrypted separately, because an update can happen at any time.

00:27:33 · Niko Bonnieure

Suppose you open a text document and change three characters. That edit is encrypted as a block. Only the change—the patch, the CRDT update—is encrypted. We use symmetric encryption, so there is a key. At the top left, you can see that we call a block containing an update a commit. It has an ID and a key.

00:28:06 · Niko Bonnieure

The key is a symmetric key, and the ID is the hash of the encrypted block. If I give someone this commit ID and key, they can obtain the update.

00:28:25 · Niko Bonnieure

We need to transfer that ID and key to the other peers or replicas with whom we want to share the update—the ones subscribing to this document. With the ID, they can request the block from the broker. The broker keeps all these end-to-end encrypted blocks. Give it the ID, and it returns the encrypted data.

00:28:59 · Niko Bonnieure

With the key, they can decrypt it. The key is not stored on the server, of course. We start with the first commit in a document—the root commit, or genesis commit if you want a blockchain analogy, although here we are only talking about a document. That first commit contains some metadata. Then the update shown here is data commit number two.

00:29:31 · Niko Bonnieure

It contains the first update and refers to the first commit by its commit ID. The yellow element with the number one is that reference, the ID of its parent, a bit like Git. There is still no commit key here. We continue in this way as more commits are added.

Concurrent commits and the DAG

00:30:07 · Niko Bonnieure

In the third column, commits three and four are concurrent. They were created by two different users, or two replicas or tabs—even by the same user—but without being connected to the internet at the time. They happen concurrently and both refer to the same past, commit number two. Both generated commits point to number two.

00:30:44 · Niko Bonnieure

When they go online, they send those commits to the broker. Other subscribers receive them and see that they are concurrent. For the data itself, that is fine: CRDTs handle concurrency well. Here, though, we are only talking about encryption.

00:31:11 · Niko Bonnieure

Those two commits are unpacked, decrypted and materialized locally, and that works. We stay in that state. The next step on the slide, labelled “merge,” has not happened yet. We have received the data and updated the local document. Then another commit occurs, number five. This is the one that, in a sense, merges the two previous commits. It points to both previous commits, three and four, as shown in the last column.

00:31:48 · Niko Bonnieure

Suppose commit five is the latest commit on this document. What I need to share is its ID and key, as I said earlier. You can see them in the red box: ID number five and key number five.

Sharing keys through pub/sub

00:32:17 · Niko Bonnieure

Let me check that you are still following. Good. How do we share this without giving it to the server? We encrypt it and send it through a pub/sub, a publish-subscribe mechanism. Other replicas receive it, but it is encrypted.

00:32:48 · Niko Bonnieure

It is encrypted with another key that I will not go into here. The replicas can decrypt it because they belong to this document's pub/sub: they are authorized readers. Only those replicas can decrypt the red item. Then they know, “We are at commit five.” They request the block from the broker, receive it, and use the key to decrypt it.

00:33:19 · Niko Bonnieure

When you are already subscribed, we push these blocks to you; you do not have to request them. If you subscribe later, you need to request the missing commits, but the protocol and library handle that for you. There is also a reconciliation mechanism, because you may already have part of a document but be missing some commits.

Reconciliation and broker federation

00:33:56 · Niko Bonnieure

You talk only to your broker, not to the other replicas. As I said, this is not direct peer-to-peer. As a replica, you tell your broker, “I have these blocks.” It says, “You are missing two or three more,” and sends them to you. That is why the yellow elements in this diagram, the block IDs, are not encrypted.

00:34:32 · Niko Bonnieure

We can see the IDs, and the broker can see them. The header containing them is plaintext, as noted at the top right of the slide. That lets the broker see the shape of the DAG, the graph of all the commits. It can reconstruct that shape in advance, so when a replica arrives and asks for only the latest missing commits, it knows which ones to send.

00:35:12 · Niko Bonnieure

Brokers also synchronize with each other, because this is a federated system. I can host my data on one broker and somebody else on another, and we can still collaborate on the same documents. The brokers exchange data even while our devices are offline. That is useful: when I come online, the data I need to synchronize is already on my broker.

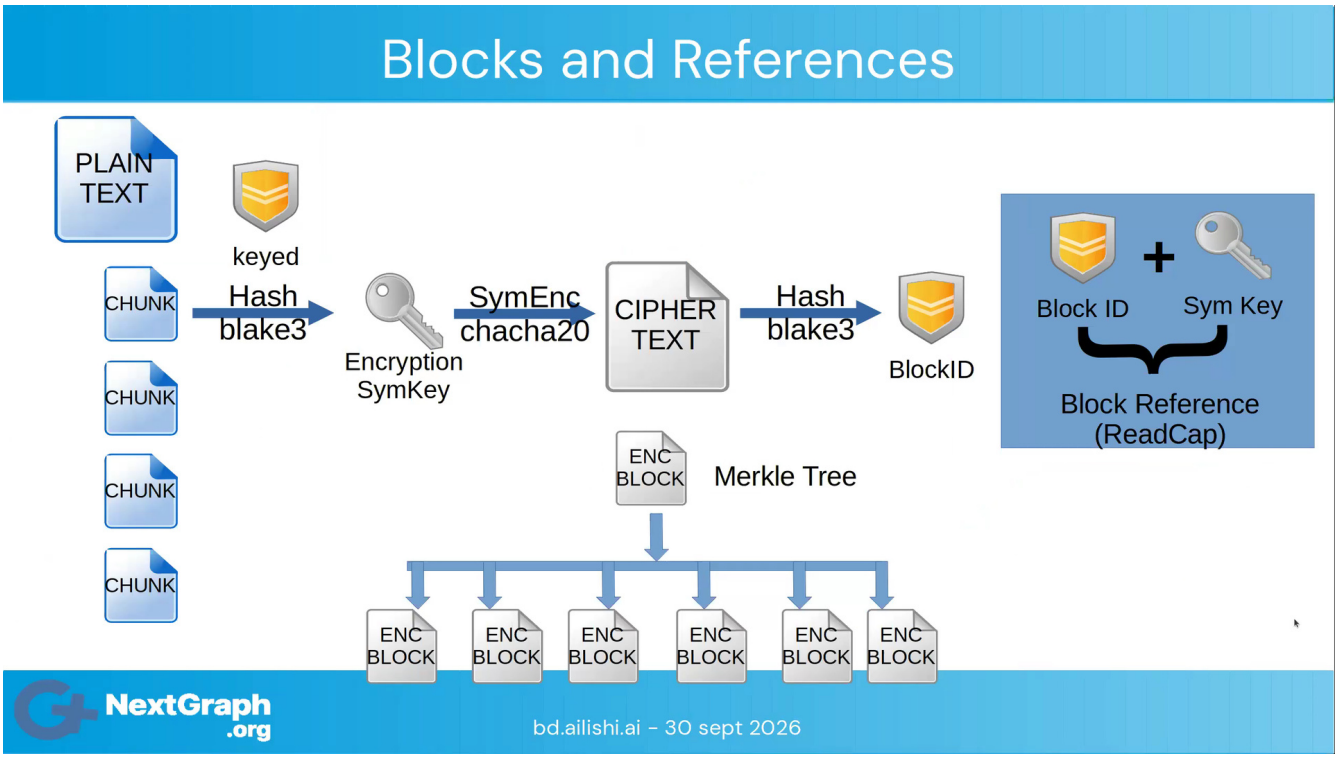
00:35:49 · Niko Bonnieure

That makes it quicker. The broker has already prepared the latest commits I am missing. When I connect, I receive only those, and in the correct partial order, which can be determined from the DAG. You know what came earlier. That helps the CRDT afterwards. I believe this is the right way to encrypt local-first data.

Merkle trees and chunking

00:36:24 · Niko Bonnieure

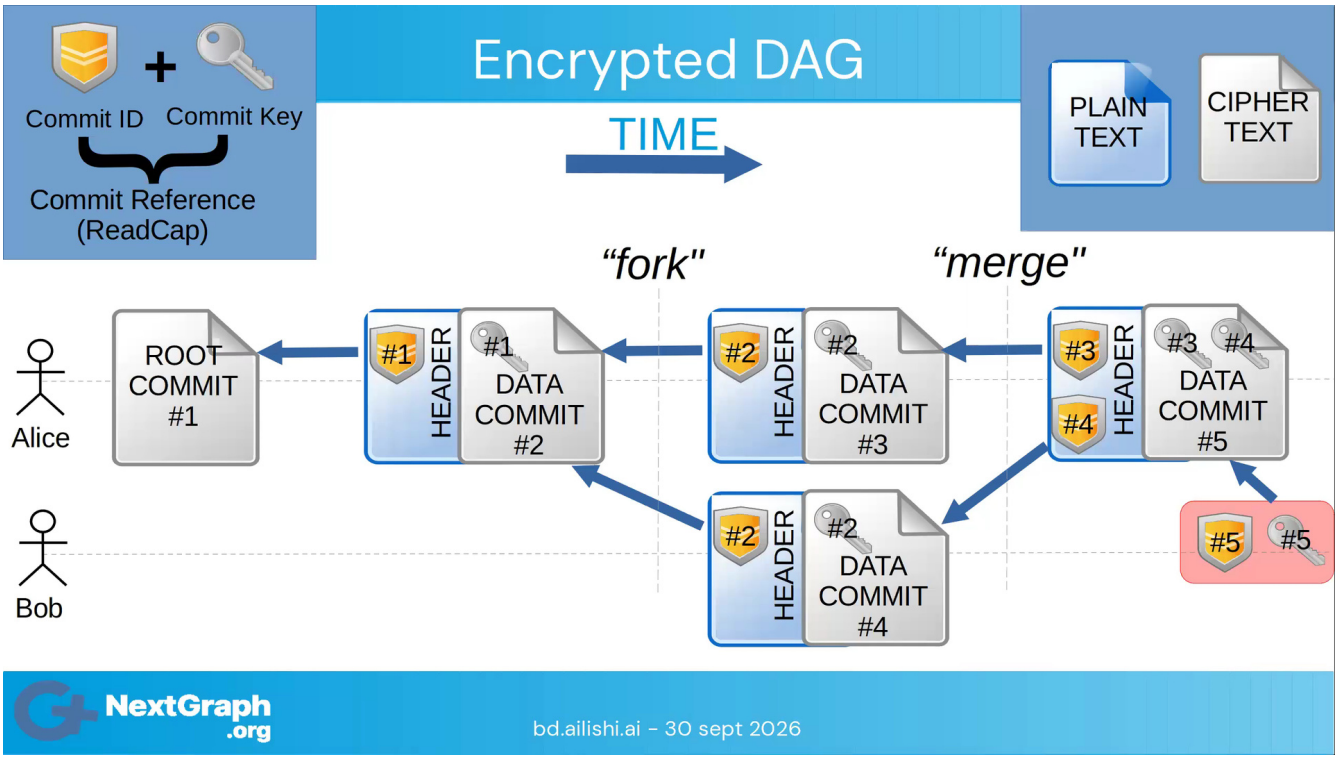
Other libraries also do it this way. Returning to the previous slide, this shows exactly how we handle blocks containing data, block IDs and symmetric keys. There is a Merkle tree inside the blocks. If an update is very large, more than one megabyte, we split it into chunks, a bit like BitTorrent.



Blocks and references, revisited · 00:36:28

00:36:55 · Niko Bonnieure

That is what this slide explains, along with the symmetric encryption and hashing I mentioned. That is the technical part. Please ask if you have questions about it now, and then I can move on. Is that clear?



Encrypted DAG and Q&A · 00:37:07

00:37:29 · Niko Bonnieure

Sorry, I did not hear your question or comment.

00:37:34 · Host

He left a question in the chat.

00:37:37 · Niko Bonnieure

Where is the chat? I cannot see it.

Q&A: an encrypted Git?

00:37:43 · Host

He is asking whether the structure is like an encrypted Git.

00:37:54 · Niko Bonnieure

Yes, it is similar. The main difference is that in Git you cannot have more than two parents. You can have two when you merge two branches: the commit has two parents. Here, you can have three, four or five. If three devices are editing concurrently without the internet, there will be three parents. That is the difference in the analogy with Git.

> Editor's note: Git also supports octopus merges with more than two parents. The speaker's wording is retained here; see the Git merge documentation: https://git-scm.com/docs/git-merge .

Q&A: scale and benchmarks

00:38:37 · Audience

Hi, Niko. Thanks for your presentation. I have two questions. First, about NextGraph's capacity: how many users and devices do you estimate the system can currently support for real-time, end-to-end encrypted collaboration?

00:39:02 · Niko Bonnieure

We do not have benchmarks for that. I'm sorry to disappoint you. I expect it to work well with a small number of participants, but I do not know how it scales. The CRDT part will depend on the performance of the CRDT libraries. For the network itself, we do not have benchmarks yet, but we should start doing them.

Q&A: OrbitDB, snapshots and coordination

00:39:40 · Audience

My second question is about comparison with other open-source projects, such as OrbitDB and Delta. How does your roadmap differ from theirs?

00:39:58 · Niko Bonnieure

Can you name the projects again? I did not understand which ones you mentioned.

00:40:05 · Audience

OrbitDB.

00:40:07 · Niko Bonnieure

OrbitDB, very good. I know it. What is the other one?

00:40:11 · Audience

Delta.

00:40:12 · Niko Bonnieure

Delta, I am not sure I know it. OrbitDB is a database, a key-value store based on a log. It needs the operation log. The last time I tried it, which was a long time ago, it would replay the whole log to access the data. I do not know whether that is still the case. I tried to address that in NextGraph by really saving the database locally, so you do not have to replay everything all the time.

00:40:58 · Niko Bonnieure

Every time you start the app or access a document, there is a database available locally. For RDF, for example, we use Oxigraph. It is encrypted at rest, but queries are fast because the data is local. We do not have to replay everything. Replaying and rematerializing the state each time is very costly with CRDTs, and we try to avoid that cost.

00:41:31 · Niko Bonnieure

The main problem is a newcomer. You add somebody to a document they have never seen before, and that document is a year old, with perhaps 10,000 commits or updates. To replay everything, they first have to fetch it all. The broker can help by sending it all at once, but they still have to decrypt the commits one by one. As you have seen, this is a chained DAG, so you start at the latest commit and work backwards into the past.

00:42:04 · Niko Bonnieure

That is slow at the encryption level, and then the CRDT also takes time to rematerialize everything. To help, we can take a snapshot: compact the past, the tombstones and things no longer needed, and produce a document that can be ingested in one go without replaying everything. It is encrypted just once too.

00:42:41 · Niko Bonnieure

That is a snapshot, and we have it in NextGraph. The caveat is that taking one requires a little coordination with others. The protocol handles that; you do not do it by hand. But it is not local-first in that respect: we need at least your broker's cooperation, so you need to be online. Your broker can handle it, but you need to connect to a broker to take that snapshot.

00:43:13 · Niko Bonnieure

Others are trying to make coordination-free snapshots, such as the Automerge and Keyhive people at Ink & Switch. These are trade-offs. They have done great work, but I have not integrated it into NextGraph. Our snapshots still need coordination, because taking a snapshot is not something you do every day.

00:43:45 · Niko Bonnieure

We are not going to work offline forever; we need to synchronize. These are intermittent disconnections. If you spend a month without connecting, you are not collaborating with others; you simply have a local file. In that situation, when you are only editing locally and never online, we actually do take a snapshot locally. We already compact everything because we know you are offline and there is no risk. So I think it is an acceptable trade-off.

00:44:29 · Audience

That keeps it simpler.

Q&A: coding agents and AI access to local data

00:44:31 · Audience

I am also curious about agentic coding tools such as Claude Code and Codex. There are many features to improve or incorporate. In your experience, how do these coding tools improve productivity?

00:44:55 · Niko Bonnieure

You are talking about agents and AI, right?

00:44:58 · Audience

Yes.

00:45:01 · Niko Bonnieure

First, in NextGraph and my own work, I am not really using AI. Some developers working with me do use it for coding, but very little. If you mean how to connect AI to the data, the database, or NextGraph, perhaps that is more your question?

00:45:26 · Audience

You can interpret it both ways. I am interested in both.

00:45:29 · Niko Bonnieure

Which one did you mean?

00:45:33 · Audience

What kinds of AI features do you plan to incorporate?

00:45:39 · Niko Bonnieure

We do not have plans for that, but some people already did it last year. They took our APIs and connected them to AI.

00:45:54 · Niko Bonnieure

We have a demo I can send you. I think it works well because of the SPARQL query language. We used Qwen, by the way, a powerful model from China that knows SPARQL. If you put instructions in the context explaining how to access the data, the LLM can query your local data using SPARQL.

00:46:29 · Niko Bonnieure

In the demo, we had a contact manager storing contact details, phone numbers and so on. We could ask, “What is Niko's phone number, or another contact's?” and it would find it.

00:46:48 · Niko Bonnieure

It used SPARQL, even without MCP. Those APIs are available in Python and Node.js, so it was easy to query from the LLM.

00:47:06 · Niko Bonnieure

Of course, that compromises the end-to-end encryption somewhat, because the LLM runs on a server and needs access to the documents or database it reads. You need to grant it access, and then it can query. If you self-host Qwen 3, you have a bit more privacy. If you give the data to OpenAI, it might be exfiltrated.

00:47:35 · Audience

Cool.

00:47:36 · Niko Bonnieure

There is a repository on NextGraph called something like LLM router. Contact me and I can guide you through it.

00:47:47 · Audience

Thanks.

00:47:50 · Niko Bonnieure

Thank you. Are there any other questions?

00:48:02 · Host

Do you have any other questions?

00:48:07 · Niko Bonnieure

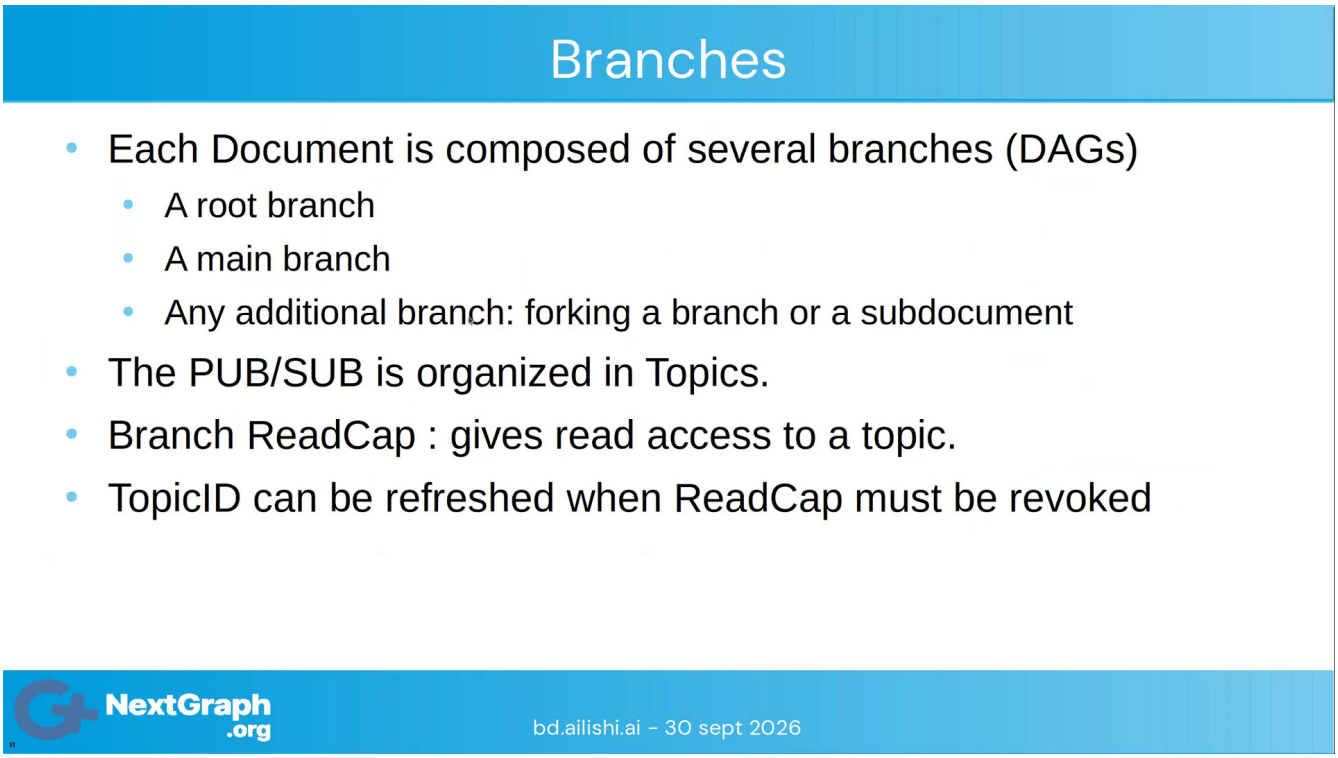
Sorry?

00:48:13 · Host

It seems there are no more questions. Perhaps you can keep going.

00:48:19 · Niko Bonnieure

I can go back to the slides. There are many more; I did not know how much time we would spend. The synchronization protocol, the branches I mentioned—we can wrap up soon unless you see something you really want to know. I have explained this and mentioned pub/sub a little. I will give you the slides afterwards, so you can go through them and ask me questions on our forum, which is also on Zulip.



Documents and branches · 00:48:28

Sync Protocol

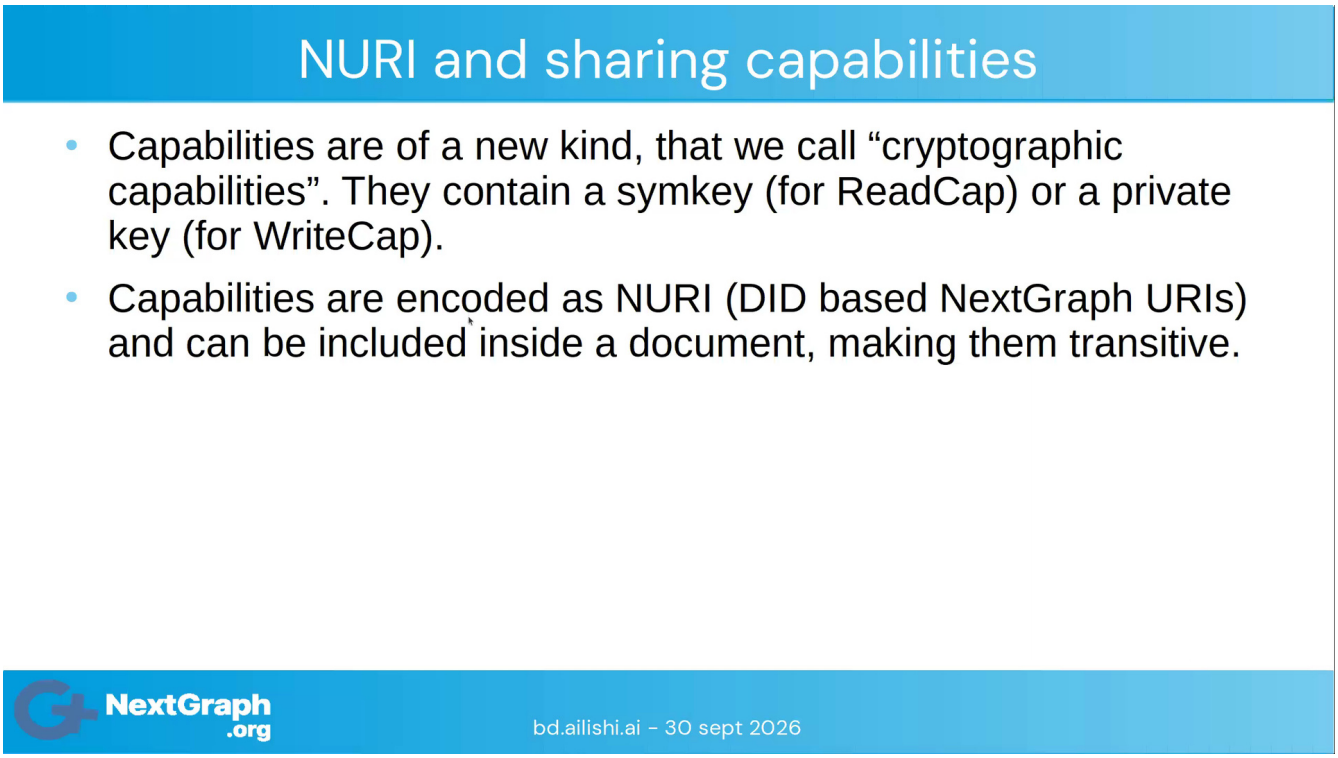
- Sync Protocol is based on BEC paper of Martin Kleppmann et al. adapted to the Broker architecture. Uses bloom filters.
- Causal partial order is preserved => Pub/Sub of NextGraph is a reliable causal broadcast.
- Brokers maintain connectivity between each other and reroute paths if needed, in a general undirected graph network topology. They maintain routing tables.

Sync protocol (1) · 00:48:33

Sync Protocol

- The Commit Reference #5, the last one in the DAG, is sent in the Pub/Sub Topic, in an Event encrypted with the Branch ReadCap
- Only the readers of this Topic/Branch can decrypt this event. The brokers can't.
- Clients subscribe to Topics and tell their Home Broker to maintain the subscription for when they go offline.
- When back online, Clients use sync protocol with Broker to get all the missing events.
- Thanks to Brokers, data is available even if no peer is online.

Sync protocol (2) · 00:48:42



NURI and sharing capabilities · 00:48:46

00:48:55 · Niko Bonnieure

As I said, the read capability is based on symmetric encryption, and the write capability on asymmetric encryption.

Pull control and post-quantum plans

00:49:06 · Niko Bonnieure

I mentioned DIDs. I will not go deeply into overlays, but they are for pull control. We even want to prevent people from obtaining the encrypted data. You cannot simply go to a broker and request data using a block ID or commit ID, as my earlier explanation might suggest. There is no public API for that. You have to belong to an overlay, which also uses encryption and security. Only if you have been granted access to that overlay can you contact the broker and request specific block IDs or commit IDs.

Pull Control & Overlays

- Having access to encrypted events is a risk. We limit to the minimum the capability to obtain encrypted blocks from brokers.
- Users that interact within a Store are isolated at the network level inside an Overlay
- Readers need access to the Outer Overlay
- While Writers need access to the Inner Overlay.
- Access to both is refreshed when needed

NextGraph .org bd.ailishi.ai – 30 sept 2026

Pull control and overlays · 00:49:06

00:49:51 · Niko Bonnieure

We do not know what may happen to encrypted data in the future; somebody might be able to decrypt it one day. We have plans for post-quantum cryptography, to be added next year, and have received more grants for it. At present, the protocol's cryptography is based on elliptic curves and is not resistant to quantum computing, but it will be.

External Signatures

- Signatures of individual commits are only useful for other editors or for readers that have access to the root branch, which is not the case of external readers.
- In order to prove authenticity of data to external readers, we have implemented a threshold signature mechanism
- Signers gather asynchronously in a quorum in order to compute a group signature for each repo's commit
- This signature is verifiable from the Document's ID (DID) and following a chain of certificates

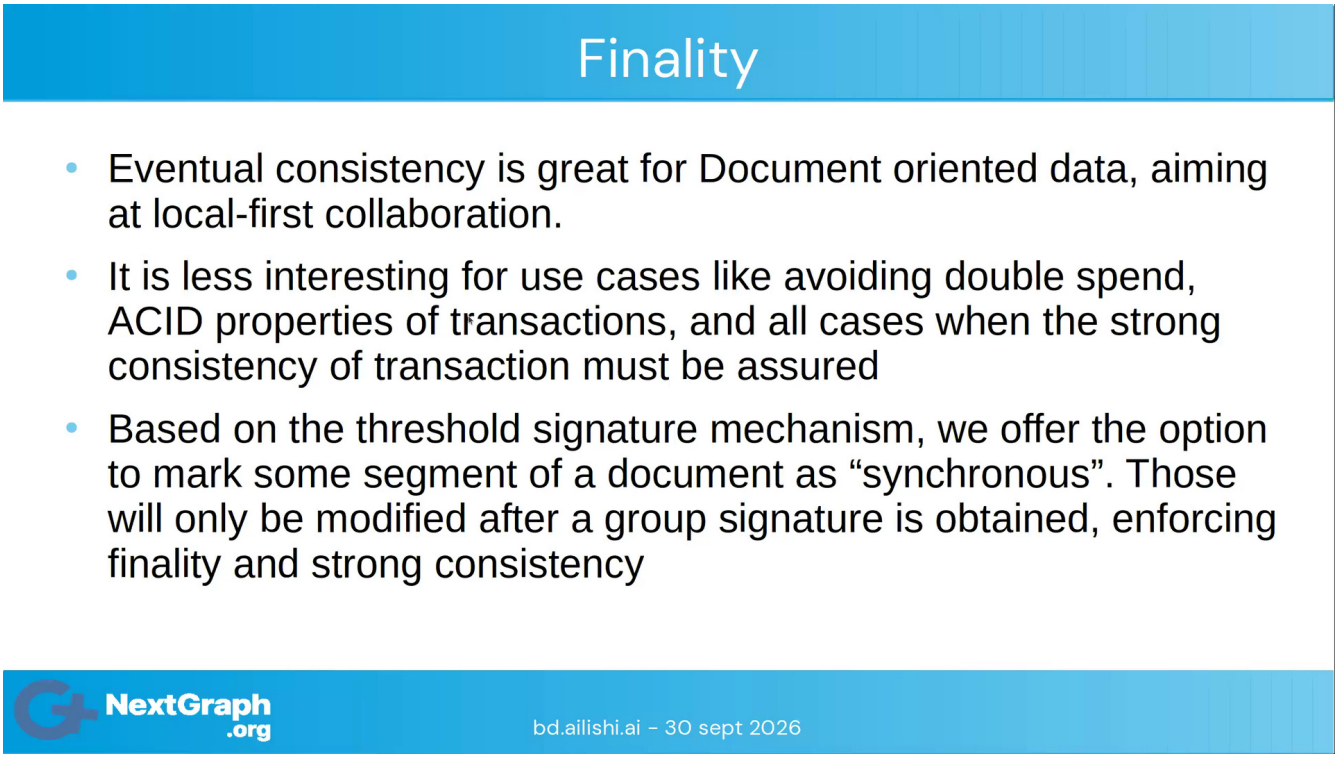
NextGraph .org bd.ailishi.ai – 30 sept 2026

External signatures · 00:50:23

ACID properties and the planned Paxos layer

00:50:24 · Niko Bonnieure

I mentioned signatures on all the data, the immutable audit trail and data authenticity. Finality is what I wanted to discuss in relation to ACID commits and databases. In the local-first paradigm, there is no coordination for data changes. I can modify data while offline.



Finality and strong consistency · 00:50:29

00:50:57 · Niko Bonnieure

That breaks the ACID properties we are used to in databases such as a server-hosted PostgreSQL database. When I connect and perform a transaction, there may be several changes in it. When I commit, either all the data is updated or nothing is updated; there is no halfway state. That is very important for managing money or e-commerce.

00:51:31 · Niko Bonnieure

For an order, you do not want to sell the same item twice or spend the same money twice. CRDTs and local-first do not solve that. You can modify data here and there without coordination or internet access, but when those changes are merged or reconciled, the CRDT does not care whether you sold something twice or spent the money twice.

00:52:02 · Niko Bonnieure

These are two orthogonal problems; you cannot solve that with CRDTs. We want to bring ACID properties back into NextGraph as an add-on. Most of the time, CRDTs and local-first are enough, but sometimes you want database properties such as atomicity and consistency.

00:52:24 · Niko Bonnieure

We have not done it yet. It is in the design but has not been coded. We will use coordination mediated by brokers, still with end-to-end encryption. The brokers will not know what you are doing, but they will help you coordinate. It is based on Paxos, the coordination algorithm. Then you can build smart contracts on top and do what you need.

Synchronous Transactions

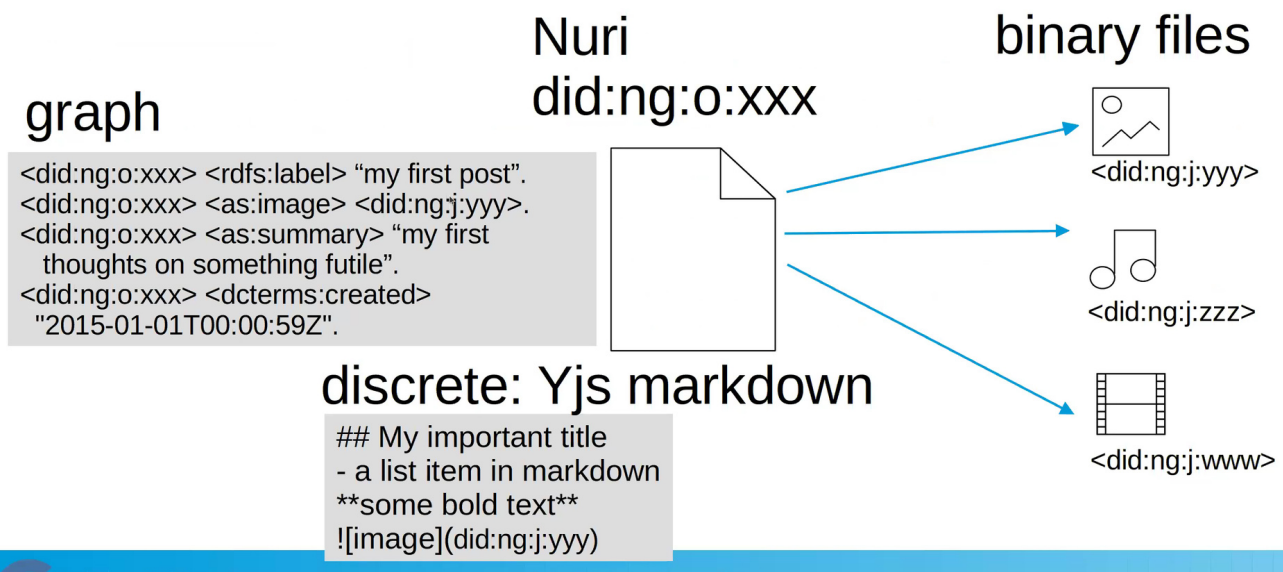
- Coordination and strong consistency with Paxos
- Synchronous transactions can implement Smart Contracts, cross-document transactions, atomic transactions, decentralized naming system, and e-commerce applications
- It complements well the CRDT/eventual consistent model. Each document/application can benefit from both world.

NextGraph .org

bd.ailishi.ai - 30 sept 2026

Synchronous transactions · 00:52:49

NextGraph Document: Rich Text



NextGraph .org

bd.ailishi.ai - 30 sept 2026

Rich-text document example · 00:52:54

Document structure: graph, discrete and files

00:52:57 · Niko Bonnieure

Here is the document structure. A document has an ID called a NURI, a NextGraph URI. It is a did:ng DID. Inside, there can be a graph, the RDF part. Here is an RDF example: each line under the graph is a triple with a subject, predicate and object. You can learn more about RDF if you want; I love it.

00:53:27 · Niko Bonnieure

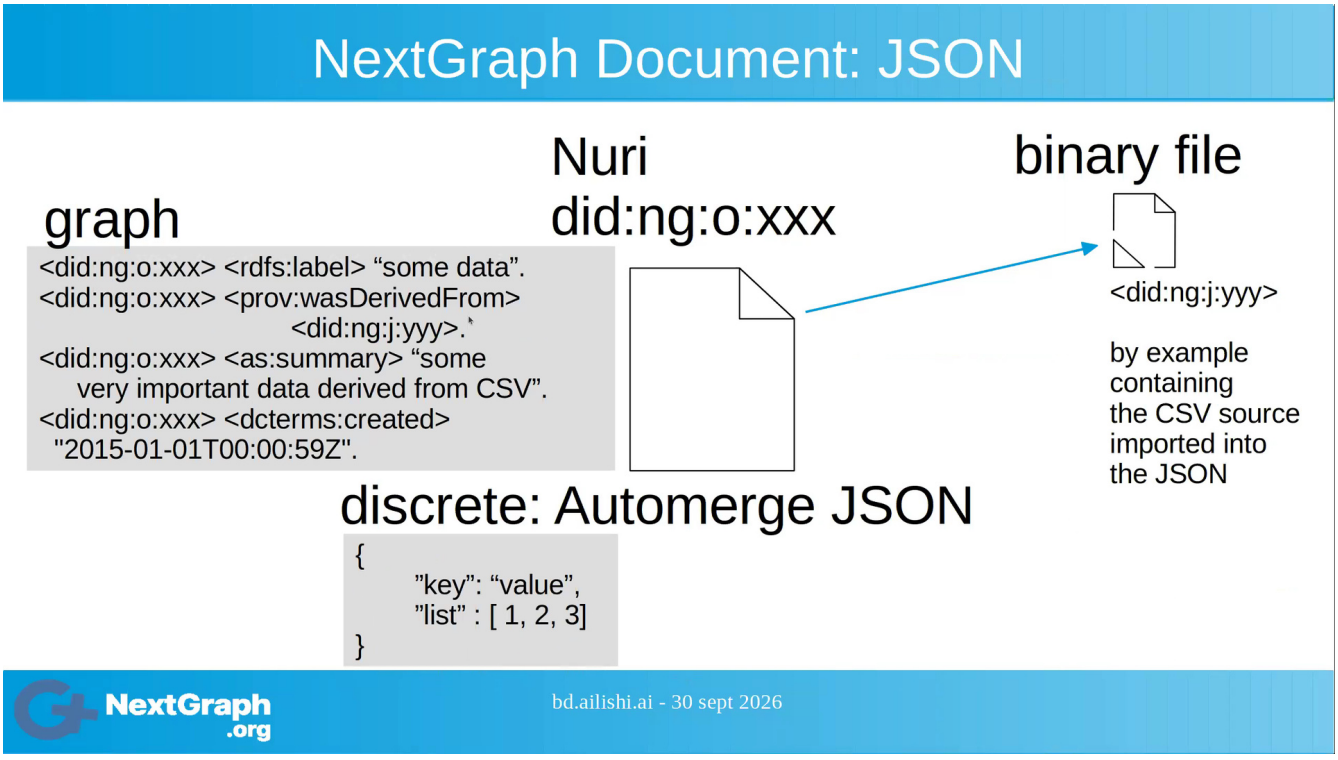
Another speaker here likes it too, so you can ask him questions in the coming days. When is your talk? Tomorrow?

00:53:37 · Audience

On Friday.

00:53:38 · Niko Bonnieure

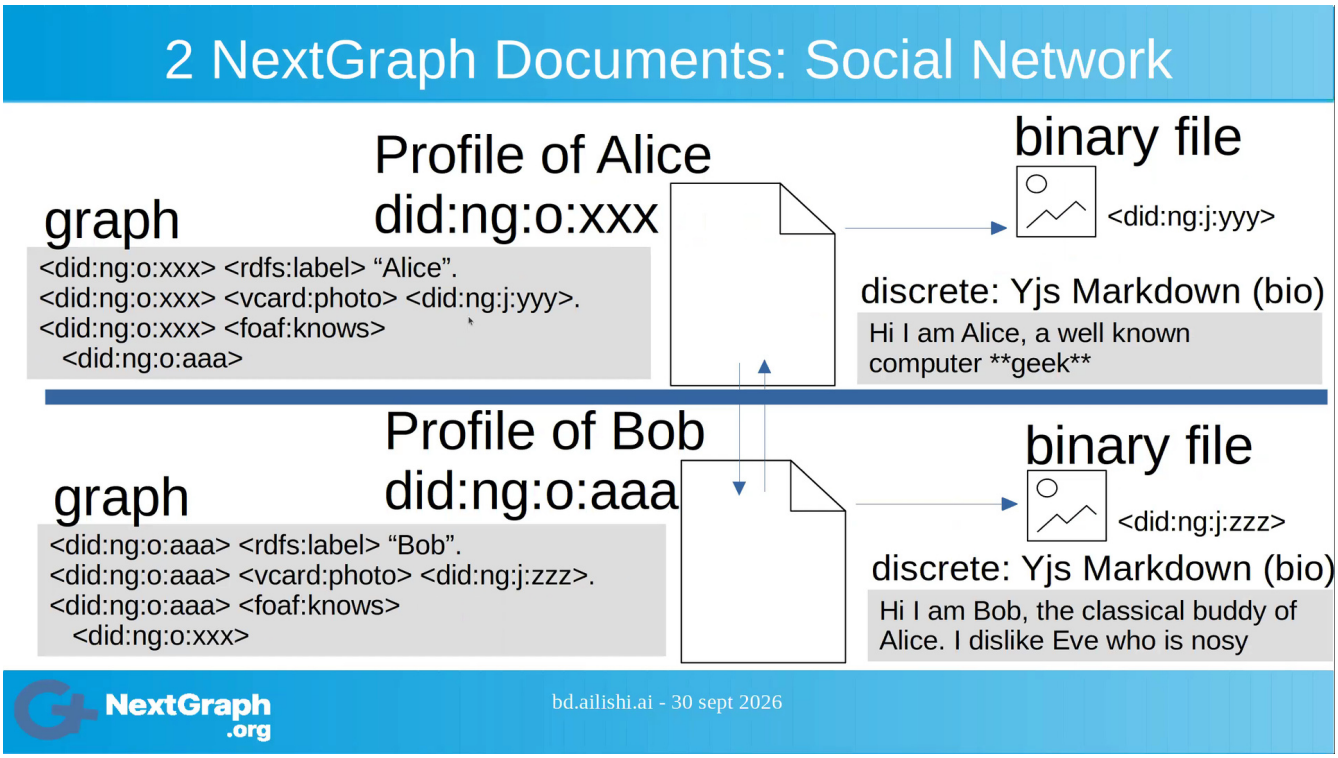
On Friday. Come to his talk; it will be interesting, and you can ask about RDF there. The discrete part is the other side of a document, or branches inside it, where you can put Yjs data such as Markdown or rich text. It could also be JSON data.



JSON document example · 00:54:04

00:54:09 · Niko Bonnieure

The graph and discrete parts are both inside the same document. You can also attach binary files. They are uploaded to the broker and made available, and you get a URI for the binary data. You can put that link in the graph or in the JSON to keep it.



Social-network document example · 00:54:36

00:54:37 · Niko Bonnieure

For social networking, there are user profiles and social data, direct messages and chat. It is possible to build a social network on NextGraph. That brings us to the end. Here are my contact details; you can also find me on the website. Thank you, everyone.



Contact and discussion · 00:54:54

00:55:09 · Niko Bonnieure

Thank you for this time together. I hope you build many apps on NextGraph. If you need help, tell me. If you want to know more about local-first, encryption or these concepts in general, I am always happy to answer questions.

NextGraph: https://nextgraph.org/ · Documentation: https://docs.nextgraph.org/