NextGraph：打造 P2P 与 E2EE 的社区应用

Niko Bonnieure · 2026-09-30 · 中文译文

根据 2026 年 9 月 30 日的演讲与问答整理，保留发言顺序、例子、限定条件和交流承接，校订口头重复、断句与明显识别错误。产品状态和后续计划以分享时点为准。时间对应交付视频。



NextGraph 与分享者 · 00:00:00

NextGraph 项目介绍

00:00:00 · Niko Bonnieure

大家好，谢谢你们邀请我。不知道大家能不能看到我的共享屏幕？可以，有人确认看到了，那我可以翻页了。我是 Niko，正在做 NextGraph，项目得到了屏幕上这些欧洲资助机构、欧盟委员会的支持。我想先介绍一下 NextGraph，再深入讲大家最感兴趣的部分。想听哪些内容，可以告诉我。



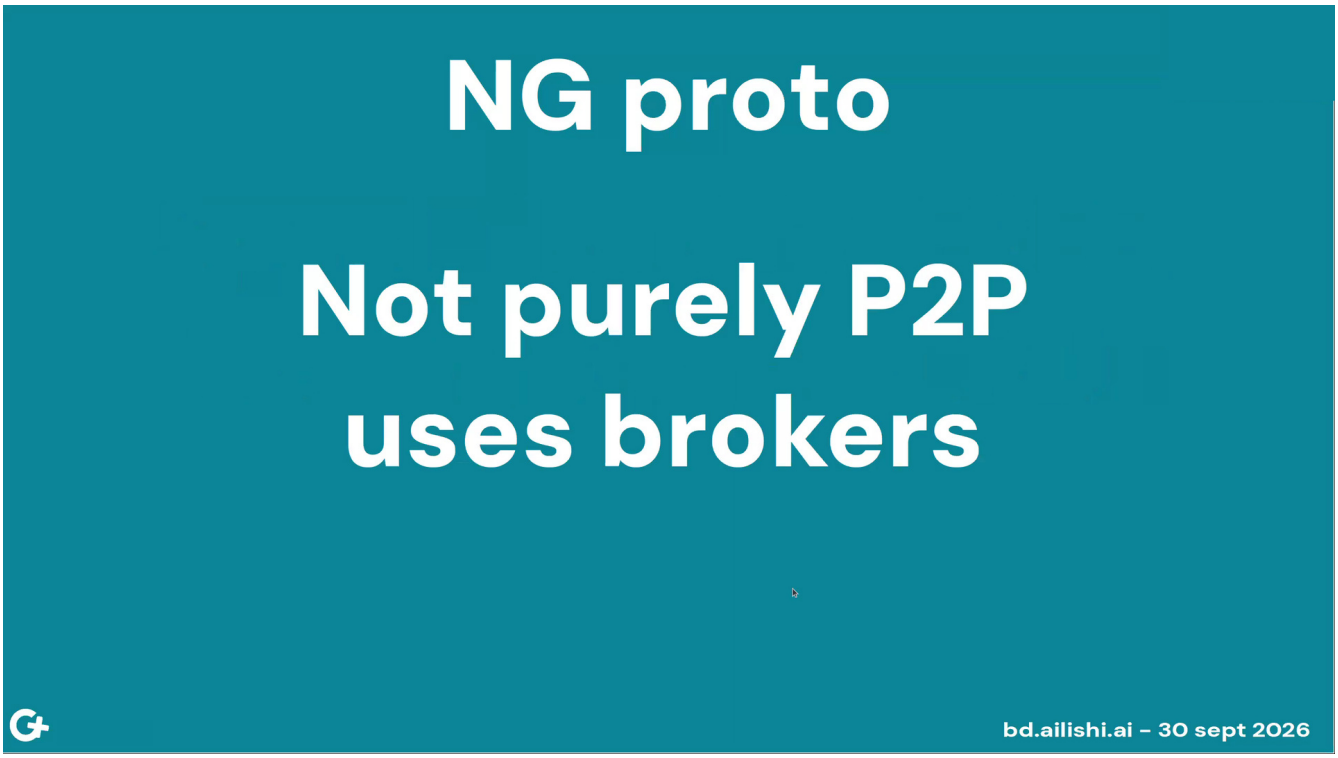
项目资助方 • 00:00:19



端到端加密、本地优先与 CRDT • 00:00:28

00:00:44 • Niko Bonnieure

首先，我们有一个数据同步协议。它采用端到端加密，也就是说，接收数据的服务器看不到数据内容。它也是本地优先的，没有互联网也能工作，并且基于 CRDT。我想大家都知道 CRDT 是什么。



为什么使用 broker • 00:01:15

为什么通过 broker 同步

00:01:17 • Niko Bonnieure

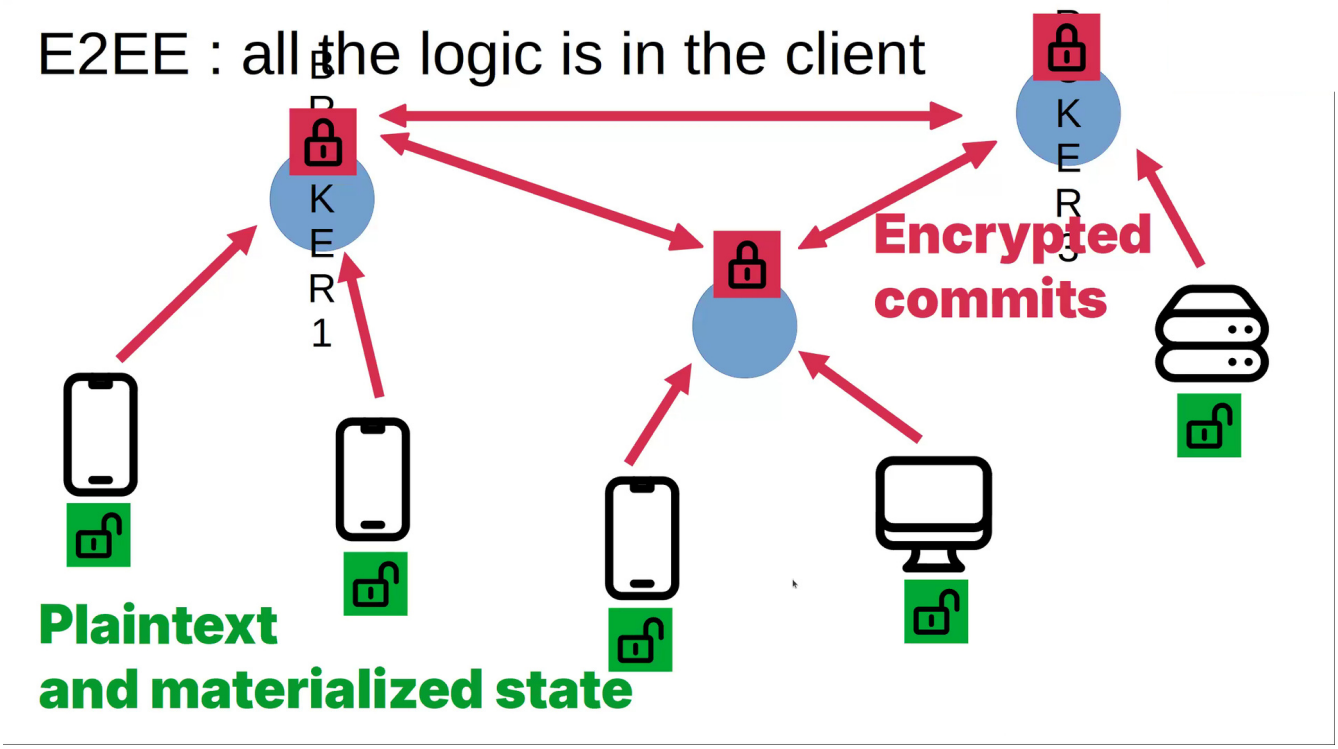
它并不是纯粹的点对点系统，严格来说并不是那种 P2P。题目里写了 P2P，很抱歉让大家失望了，但这样做有充分的理由。纯粹的点对点意味着设备直接互相连接，比如在浏览器里使用 WebRTC。

00:01:41 • Niko Bonnieure

这意味着同步不经过服务器中转。当你和别人分享内容时，对方的设备必须和你的设备同时开机、同时在线，才能同步，而这一点并不总能保证。所以，你想和别人分享时，更希望先把端到端加密的数据发到服务器，然后自己就可以离线了。

00:02:16 • Niko Bonnieure

等对方上线后，再通过服务器同步。我们把这些服务器叫作 broker。图中的蓝色圆圈就是 broker，它们互相通信，组成 broker 联邦，彼此只交换经过端到端加密的数据块。



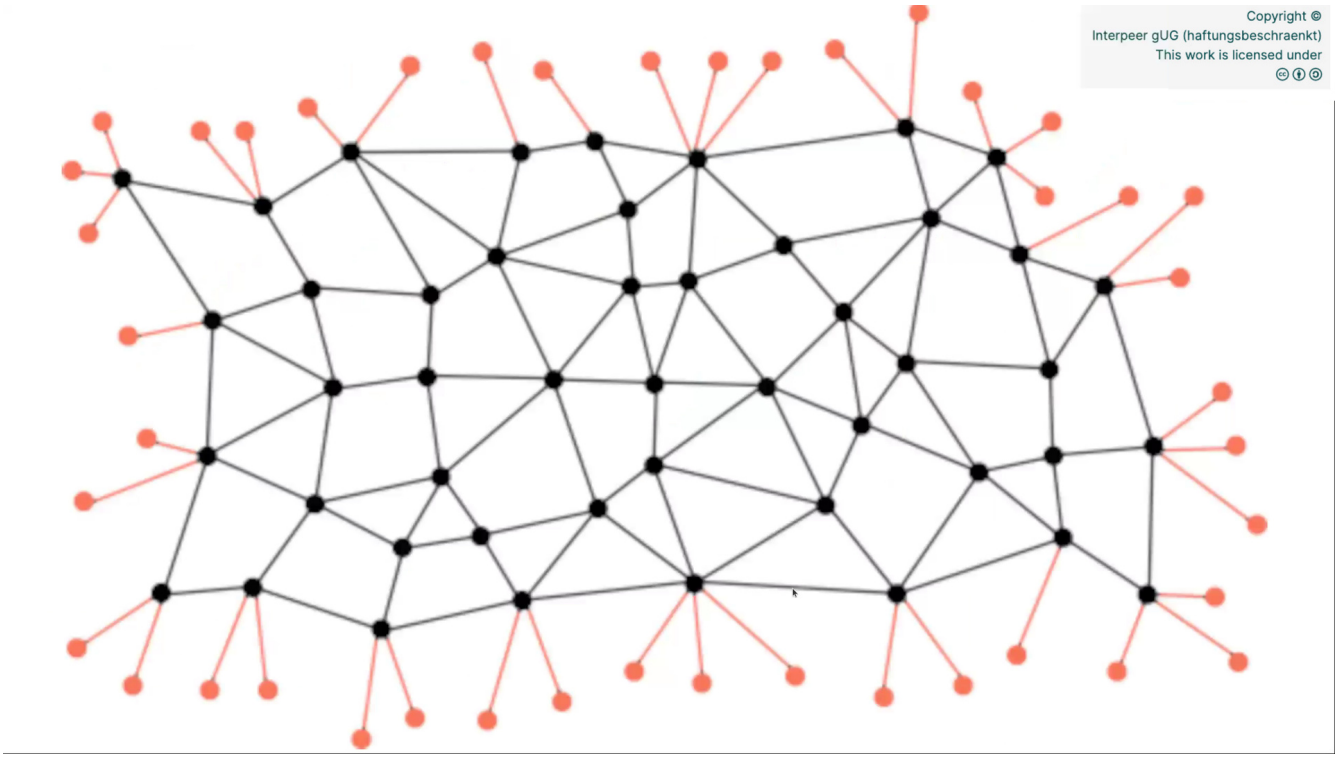
加密数据与客户端状态 · 00:02:25

00:02:48 · Niko Bonnieure

下方绿色的用户设备可以访问数据，加密和解密都在设备上进行。我们还会说，设备负责将状态物化，也就是 materialize the state。这是 CRDT 里的概念：解密之后，把 CRDT 更新聚合起来，逐条应用，在本地重建文档。

00:03:21 · Niko Bonnieure

这张图展示了网络部署之后的样子。NextGraph 目前还没有用于生产环境。这些黑点是 broker，设备连接 broker，而不是设备之间直接连接。这是我想先说明的。



broker 网络拓扑 · 00:03:24

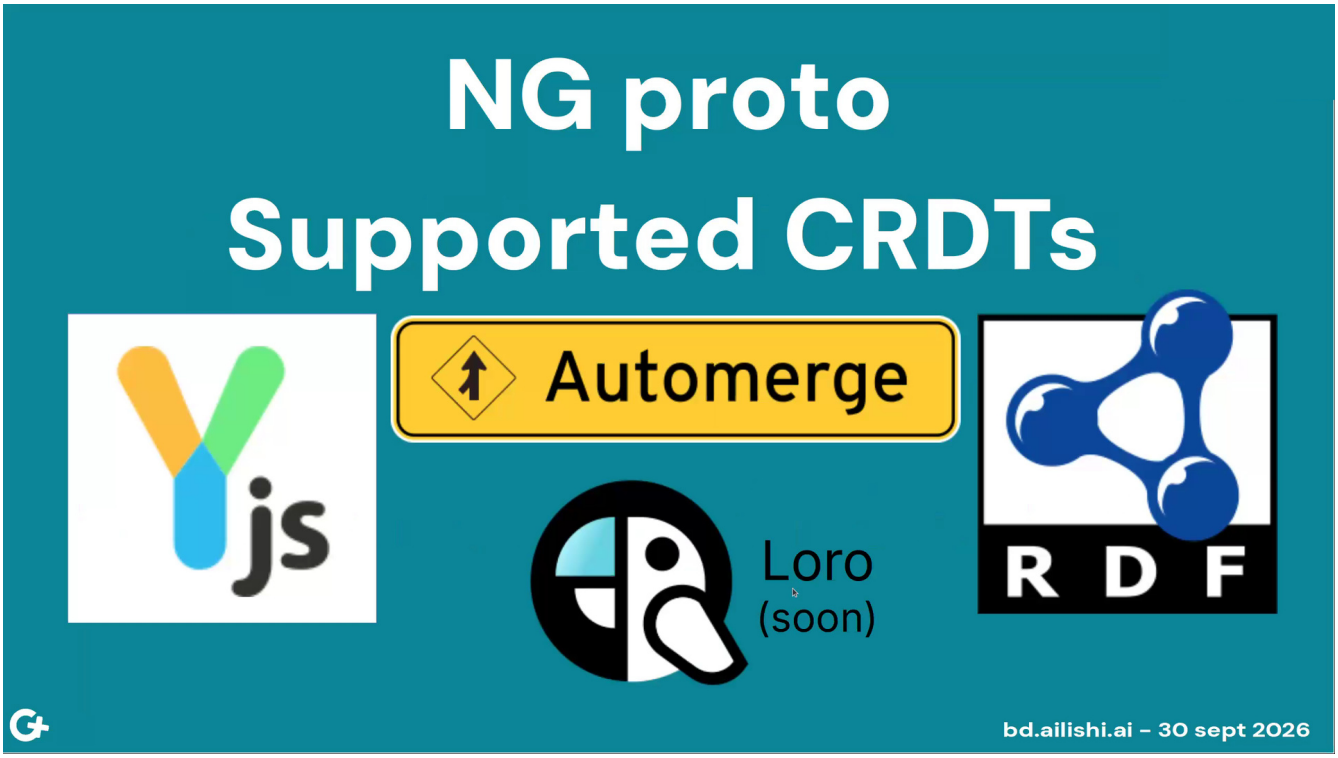


与 CRDT 类型无关的同步 · 00:03:43

不绑定特定 CRDT 的协议

00:03:45 • Niko Bonnieure

这个协议可以同步各种 CRDT，不绑定某一种 CRDT。不同的 CRDT 都可以通过它传输和同步。目前我们支持 Yjs、Automerge，以及我们为 RDF 开发的一种专用 CRDT。RDF 是一种数据描述格式，Solid 项目也在使用。我想另一位分享者很快会在这个系列活动里介绍 Solid。



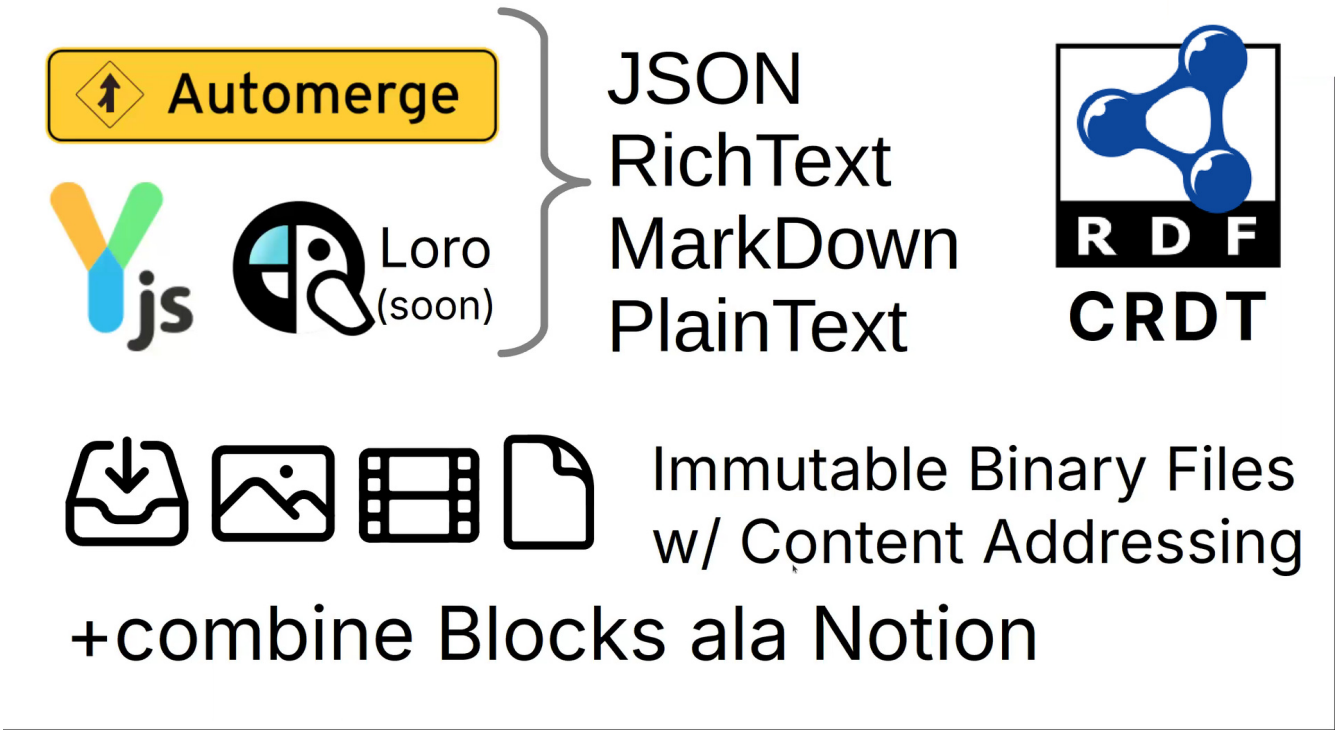
支持的 CRDT • 00:03:58

00:04:23 • Niko Bonnieure

我们也支持 RDF，后面会再展开。我们计划接入 Loro 已经有两年了。我知道它来自中国，也认识开发者，这是个很棒的项目，不过我们还没有把它接进来。增加新的 CRDT 应该比较容易，因为协议只做传输 CRDT 所必需的最少工作，后面会解释。基本上，就是保留更新之间的因果顺序，也就是偏序关系。

00:05:09 • Niko Bonnieure

所以，除了 Loro，还可以接入更多 CRDT。我们现在就在增加一种电子表格专用的 CRDT，用到了 NextGraph 协议的这种灵活性。Automerge、Yjs 和 Loro 这三种通用 CRDT 处理的是类似 JSON 的结构化数据，也能存储富文本、Markdown 和纯文本，尤其是 Yjs。



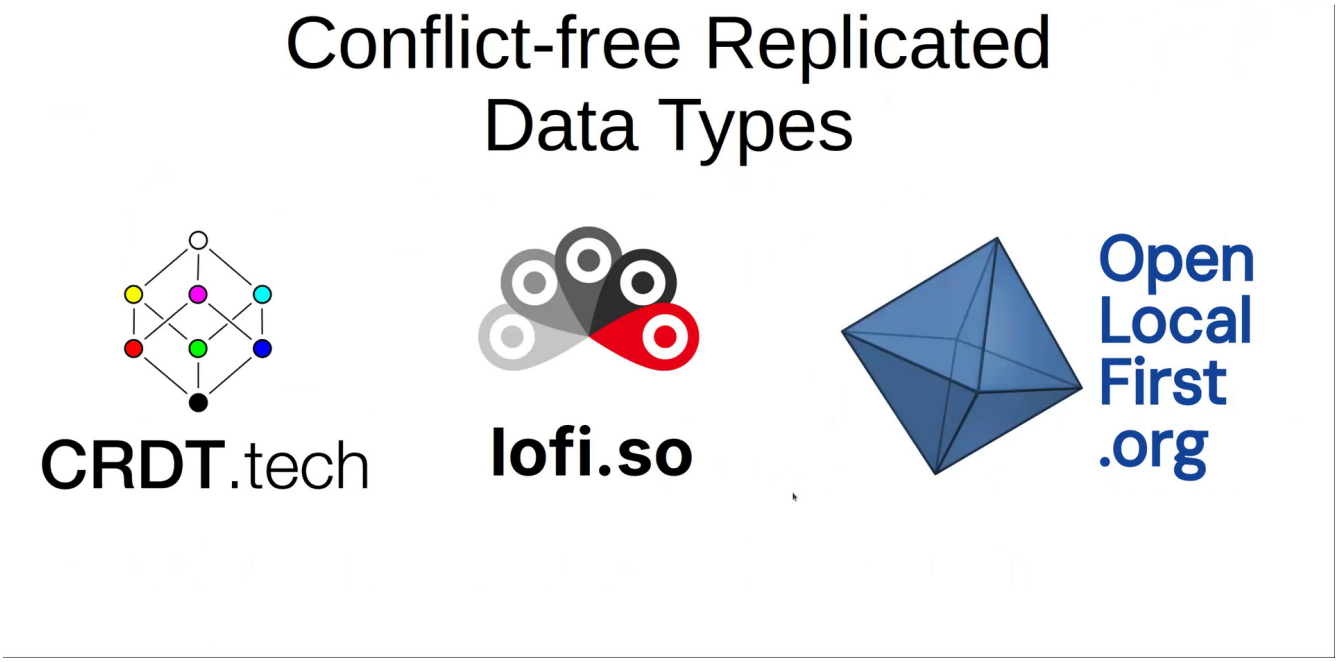
数据类型与二进制文件 • 00:05:22

00:05:43 • Niko Bonnieure

NextGraph 协议还可以同步不可变数据，比如二进制文件。我们使用内容寻址：同一个文件上传两次，不会存两份，因为我们会对内容计算哈希，知道这个文件已经有了。它是一个基于文档的系统，同时也面向数据库，把两者结合在了一起。

00:06:14 • Niko Bonnieure

你有一份份文档，所有文档里存储的数据合在一起，就构成了数据库。文档本身也可以组合，类似 Notion 或其他以内容块为基础的系统。关于 CRDT，幻灯片上的网站有更多资料。协议里另一个重要部分是密码学，以及基于密码学的访问控制机制，所以我们把它叫作密码学能力，也就是 cryptographic capabilities。



CRDT 相关资源 · 00:06:33

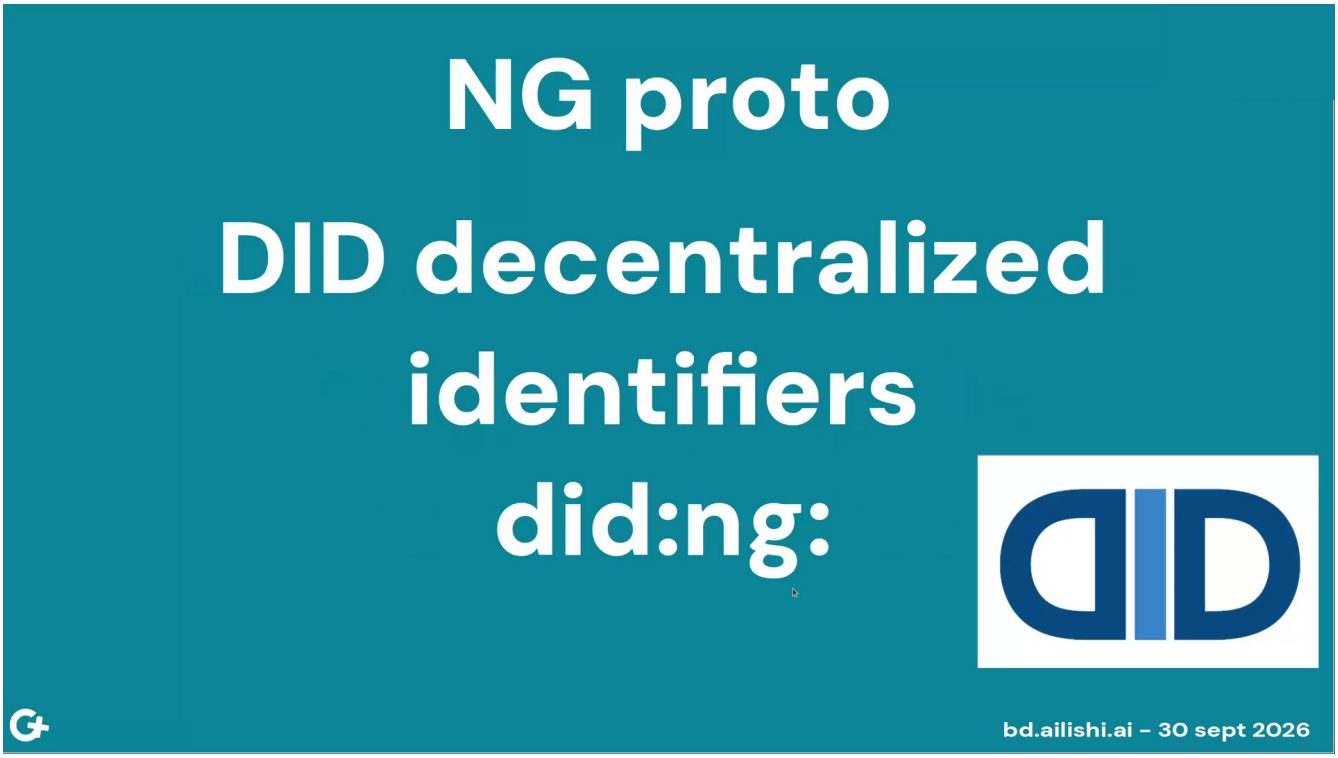


密码学能力与访问控制 · 00:06:42

密码学能力与访问控制

00:07:02 • Niko Bonnieure

后面我会详细讲。如果没有解密密钥，你收到的就只是一团无法使用的加密数据。这就是读取能力：想读，就要有这个密钥。写入能力更多依靠非对称密码学。你需要证明自己是编辑者，这通过签名完成。两者相互配合：读取依靠对称加密，看你是否持有能够解密的对称密钥；其他部分则依靠签名。这是总体思路。



去中心化标识符 • 00:07:47

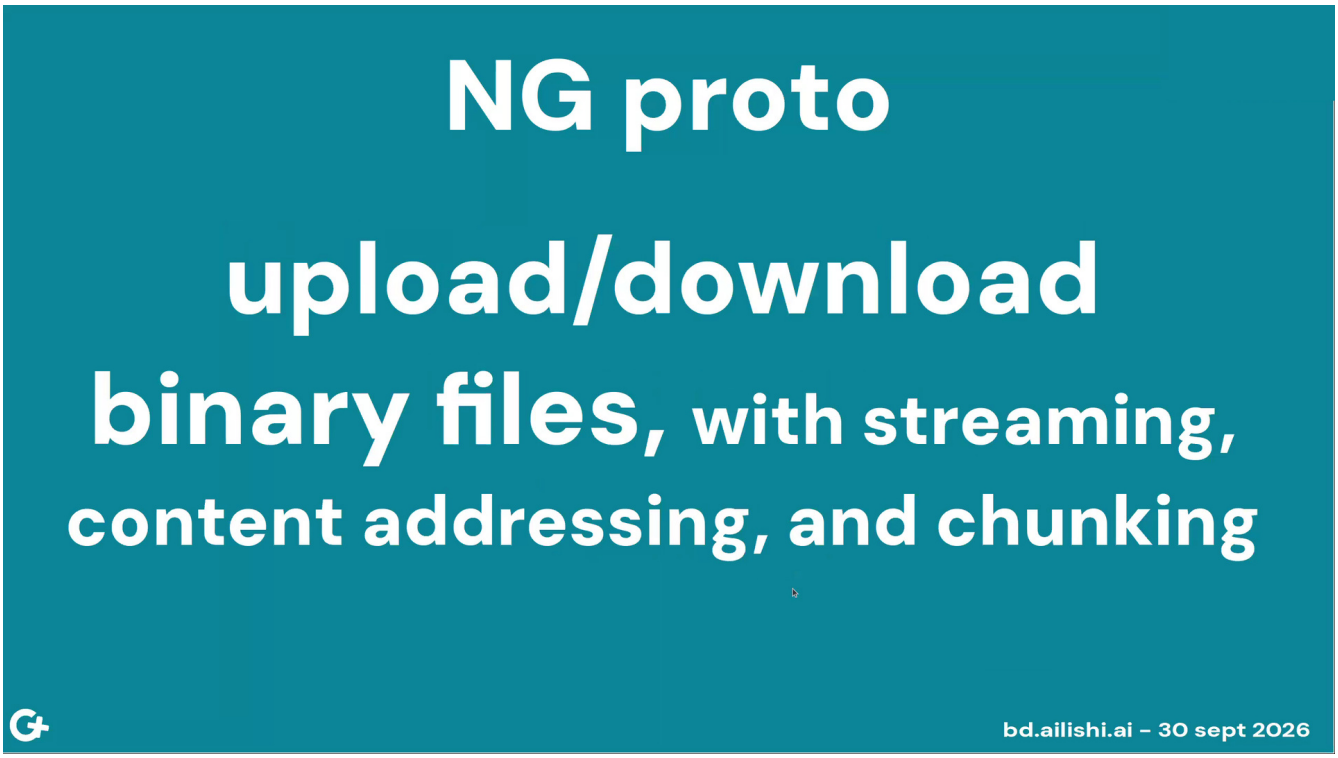
去中心化身份与持久标识符

00:07:48 • Niko Bonnieure

因为整体是去中心化的，这个协议没有单点故障。例如，我们去掉了对域名的依赖。服务器和设备通过 IP 地址以及标识它们的公钥来识别彼此。文档和用户身份都有唯一的去中心化标识，我们使用的是 W3C 的 DID 标准。

00:08:23 • Niko Bonnieure

外面有很多种 DID method，也就是 DID 方法。我们的方法叫 ng，代表 NextGraph。用它可以引用数据库中的任何文档或任何用户。ng 后面跟着标识符，是很长的一串字符，其中有公钥，URI 里还编码了数据位置等信息。

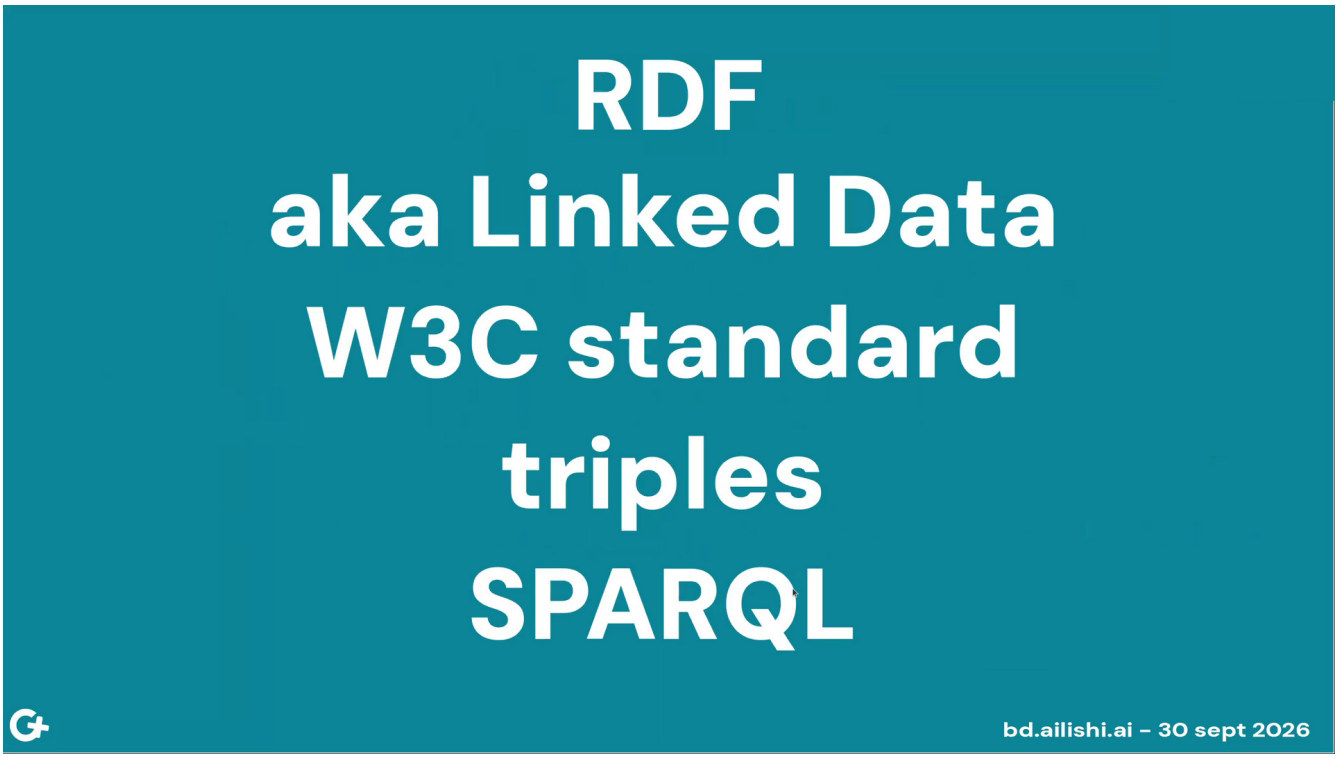


二进制文件的分块与传输 · 00:08:56

RDF、关联数据与 SPARQL

00:08:57 · Niko Bonnieure

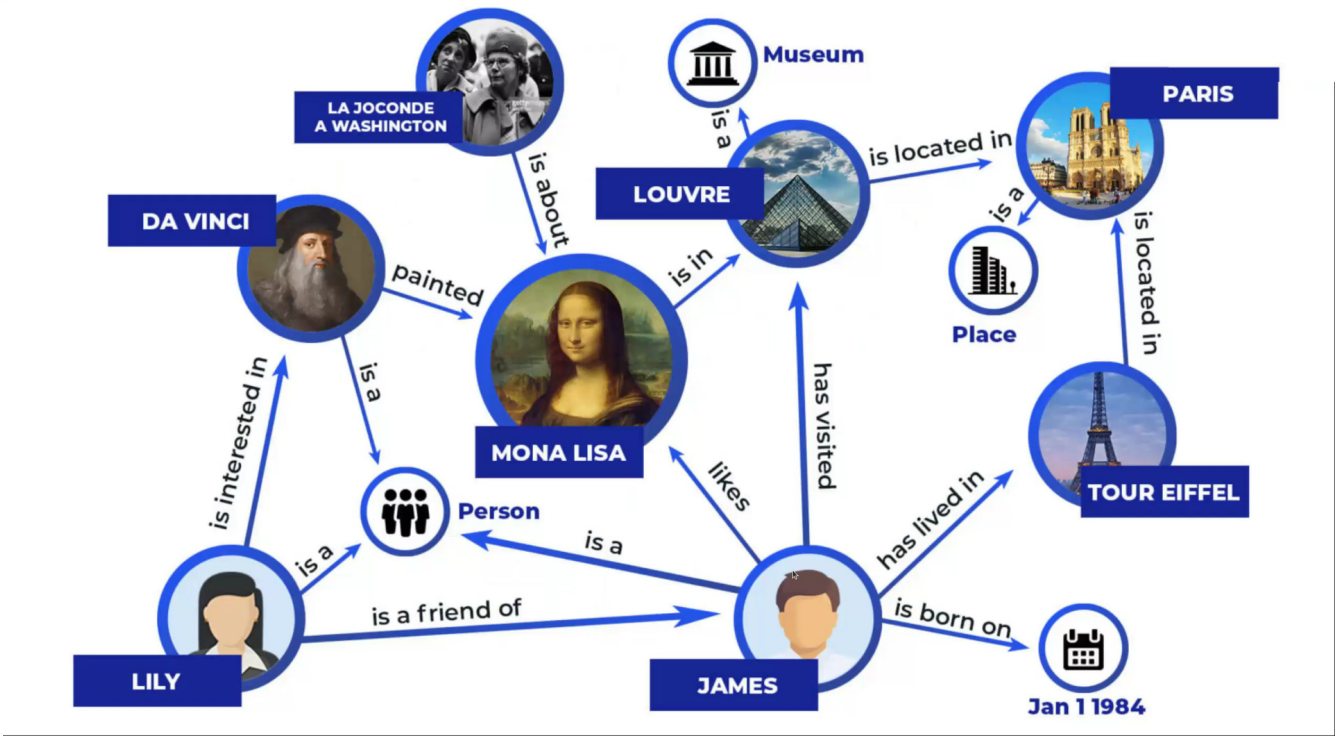
接下来讲讲 RDF，这是我很喜欢的一种数据格式。除了 JSON，我们也提供 RDF，作为补充。它也与关联数据、语义网这些概念有关。基本上，它把数据以三元组的形式存进图里，是一种图数据库。



RDF 与关联数据 · 00:08:59

00:09:30 · Niko Bonnieure

三元组是能够保存的最小数据单元，有点像键值存储中的键值对，也可以看作一个属性。它还能表示对象之间的关系。你可以用 SPARQL 查询它，这和 SQL 类似。图上是一个很简化的例子：这里有一些对象，可以用 RDF 表达对象的属性，也可以表达它们之间的关系。



图中的对象与关系 · 00:09:53

00:10:08 • Niko Bonnieure

右下角有一个叫 James 的人，他的出生日期属性是 1984 年 1 月 1 日，这是 James 的一个属性。Paris 的类型是“地点”，这是 Paris 的一个属性。对象之间还有关系，例如“访问过”“画过”或“感兴趣”。

00:10:39 • Niko Bonnieure

这样就可以把这些点连接起来。关系有类型、有标签，甚至还可以给关系附加属性，这就是 RDF 能做的事情。我很喜欢把自己的所有数据组织成一张图，再连接、引用其他人的数据。这样，散布在互联网上的许多数据库就能够互操作。



互操作与可塑软件 • 00:11:02

可塑软件与数据可迁移性

00:11:10 • Niko Bonnieure

它还支持可塑软件，也就是 malleable software：你可以更换用来编辑、查看数据的软件，把数据存储和应用分开。如果我不喜欢某个软件，可以换一个，数据仍然保留。保留在哪里？在本地，因为在 NextGraph 和本地优先软件里，数据始终保存在本地。RDF 还能提供自动关联。



自动连接数据 • 00:11:41

00:11:44 • Niko Bonnieure

不像 SQL，你不需要考虑外键。数据有全局标识符，就是前面讲的 did:ng URI。它们全局唯一，也能长期保持：即使数据移动，标识符也不会改变。数据从一个 broker 搬到另一个 broker，ID 仍然不变。这不同于我们习惯的 Web 2.0：URL 依赖域名，数据一移动、域名一变，URL 就失效，还要做重定向。我们去掉了这些问题。



记录的全局标识符 • 00:11:48

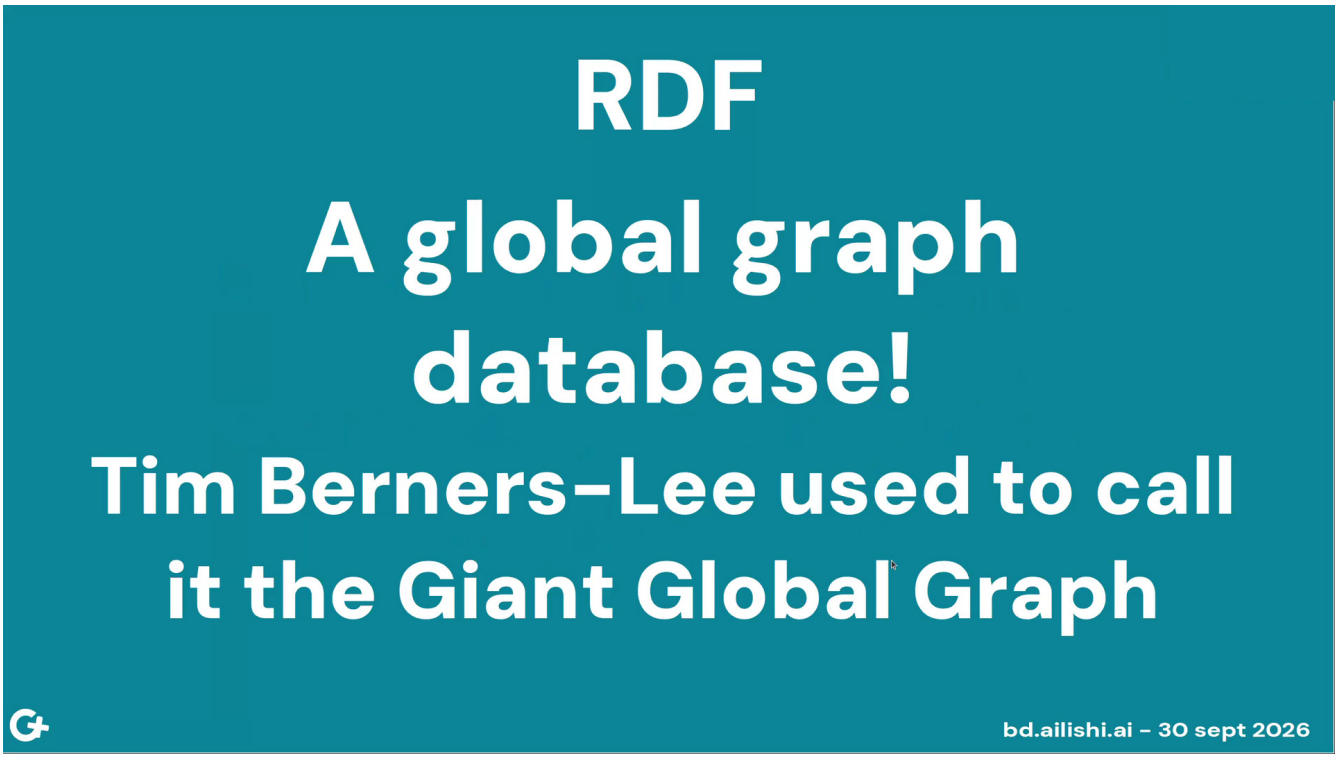


链接与引用其他文档 • 00:12:29

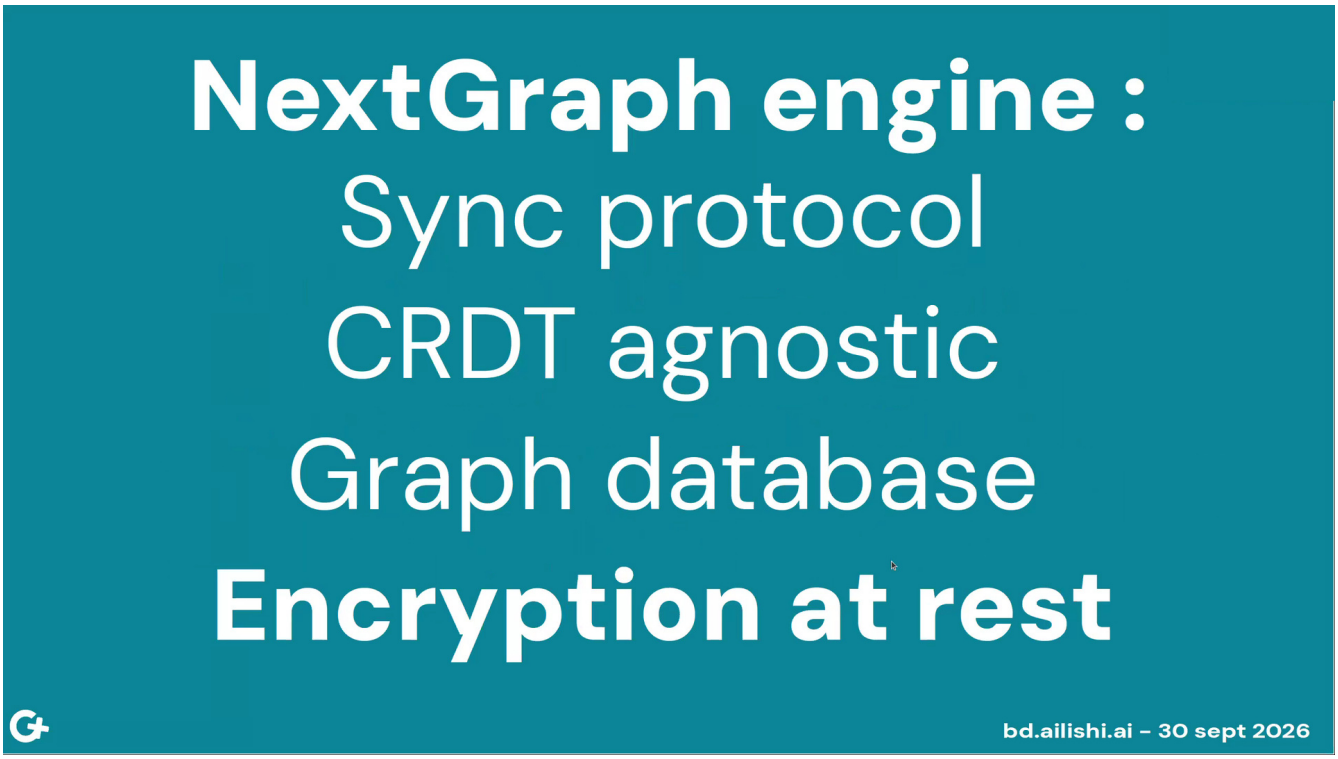
静态加密与 SDK

00:12:30 • Niko Bonnieure

前面说过，它是一个数据库。我还提到了 Tim Berners-Lee，他发明了万维网，也是关联数据概念的提出者。到这里，我们看到了一个不绑定特定 CRDT 的同步协议，以及一个图数据库。除了端到端加密，还有静态加密，也就是 encryption at rest。数据到达本地后，我们解密、物化状态，写进本地数据库，而这个本地数据库本身也是加密存储的。



全局图数据库 • 00:12:33



NextGraph 引擎 • 00:12:42

00:13:06 • Niko Bonnieure

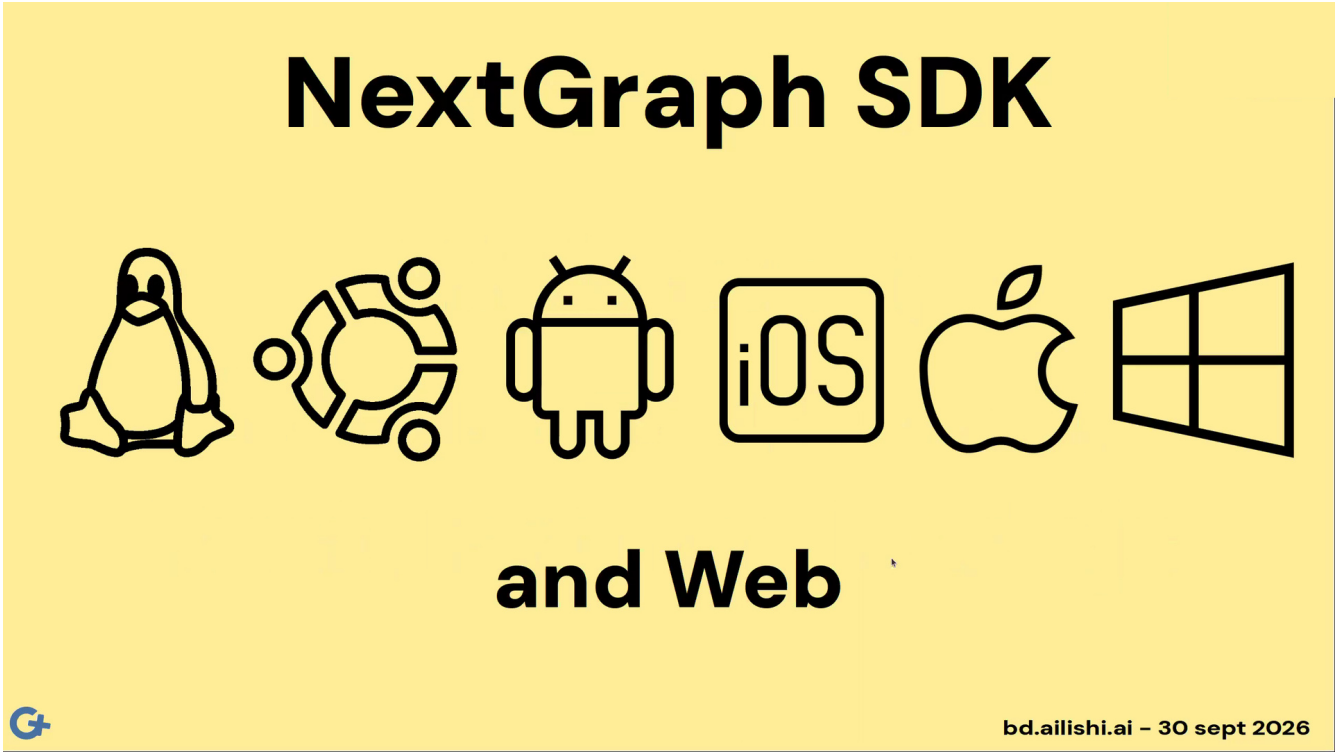
这样，即使设备被盗或被非法访问，数据仍然受到用户打开应用时所用密码的保护。我看一下时间，看看有没有问题。接下来可以讲讲 SDK。我们在协议之上提供 SDK，供大家开发应用，尽量把内部机制隐藏起来，让你专注于自己的应用，不用操心里面怎么工作。



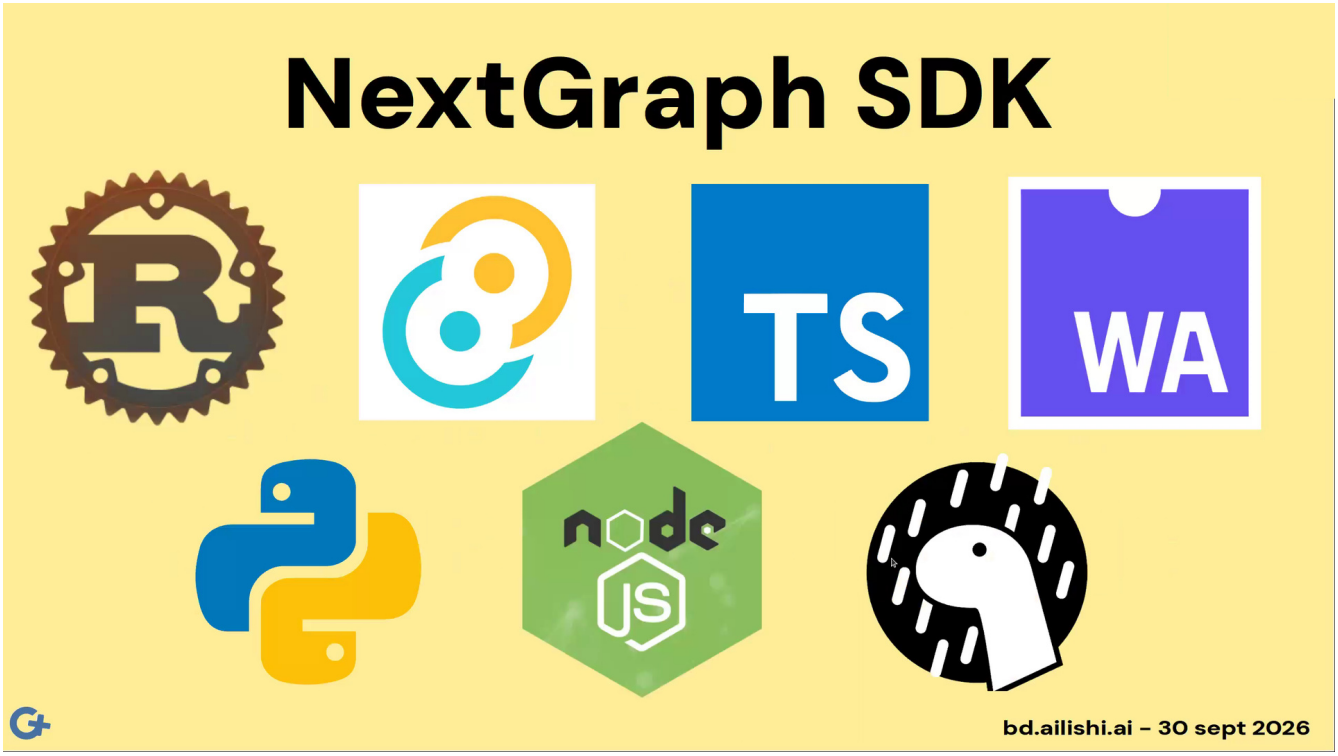
SDK 与运行环境 • 00:13:21

00:13:50 • Niko Bonnieure

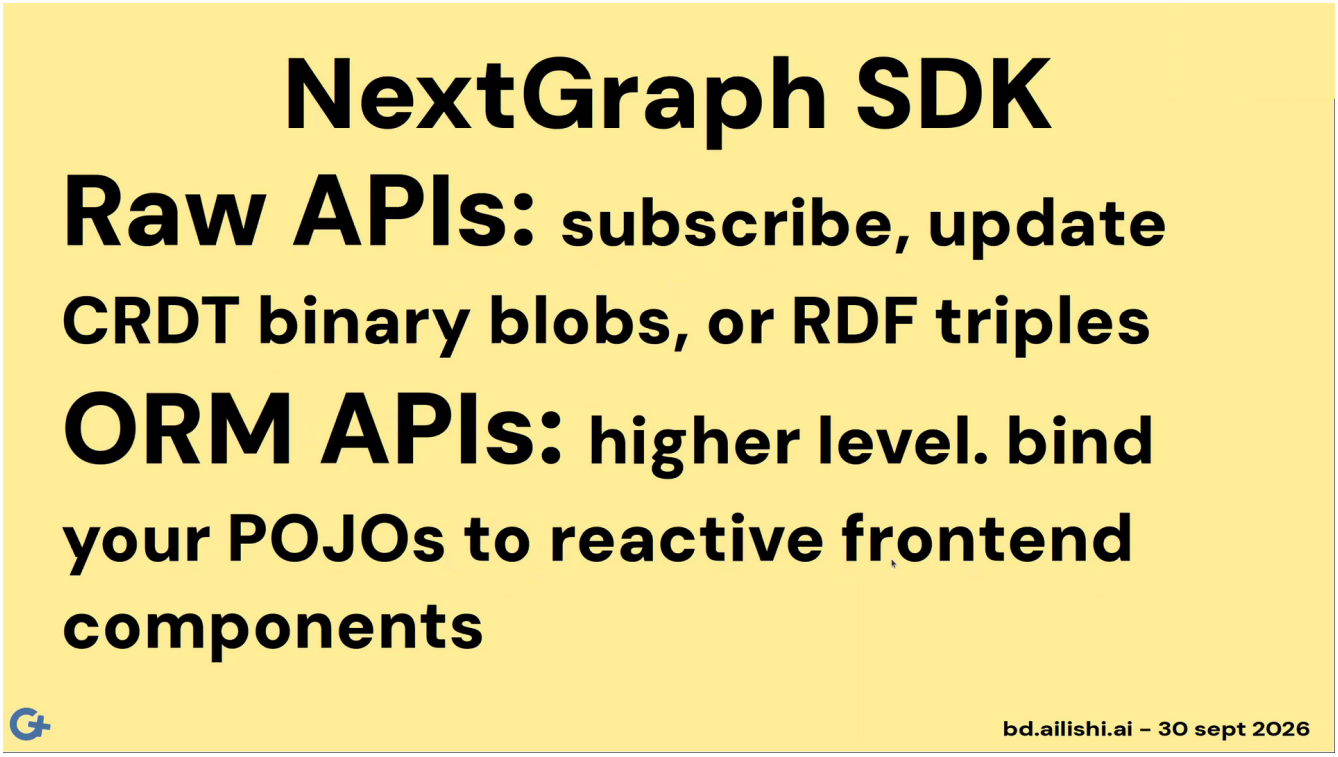
你可以开发 Web 应用，也可以开发原生应用，因为我们使用 Tauri。NextGraph 用 Rust 编写，前端部分使用 TypeScript，可以编译到常见的平台。我们还为 Node.js 和 Python 提供 SDK，用来开发服务、AI 等。幻灯片上就是我提到的五个平台，以及目前支持的语言。



支持的平台 • 00:14:17



语言与运行时 • 00:14:20



原始 API 与 ORM API • 00:14:24

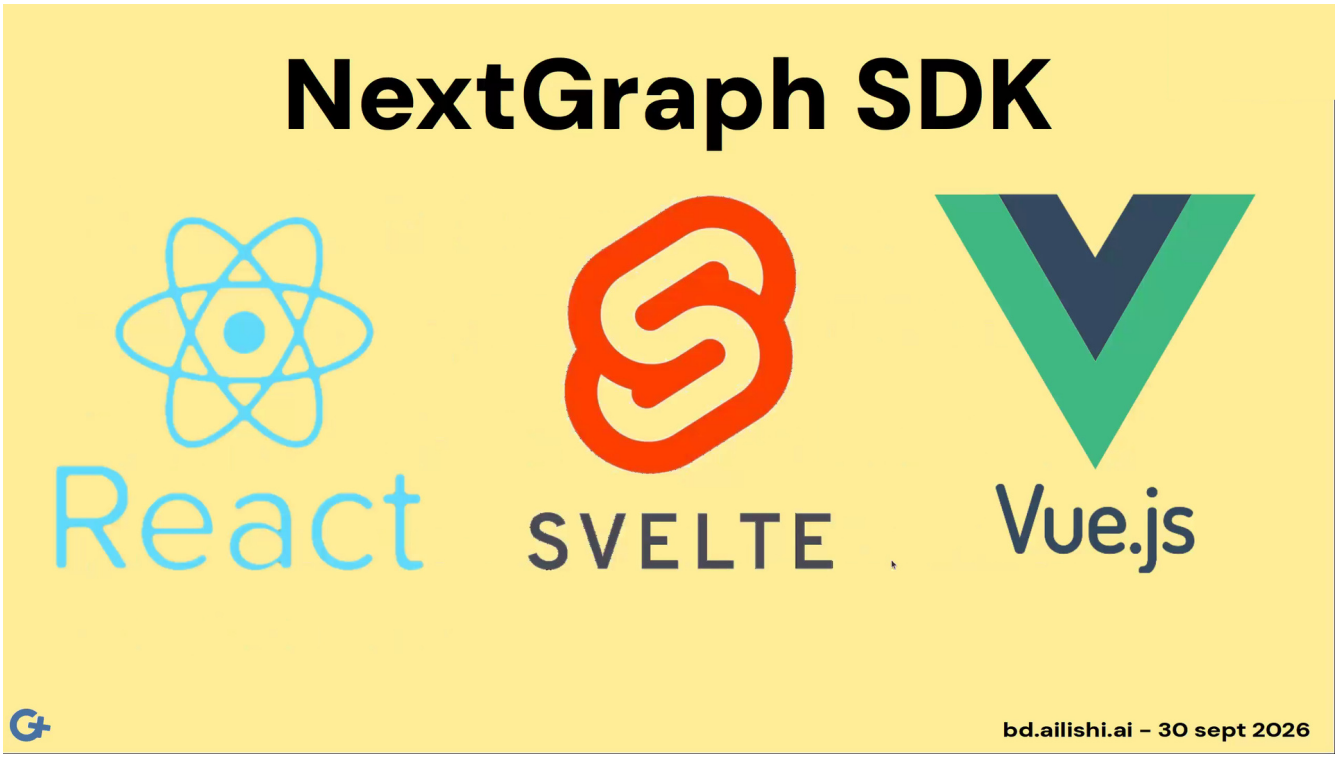
00:14:25 • Niko Bonnieure

SDK 有两个层次的 API。用底层 API，你可以订阅文档或者创建文档。订阅意味着数据会持续保持最新状态。网络中某处、某个标签页发生更新时，你在浏览器里的本地数据也会更新。它始终基于 Web 技术，实际上连原生应用也是 WebView。通过订阅，各类数据都能保持更新。

TypeScript ORM 与响应式前端

00:15:04 • Niko Bonnieure

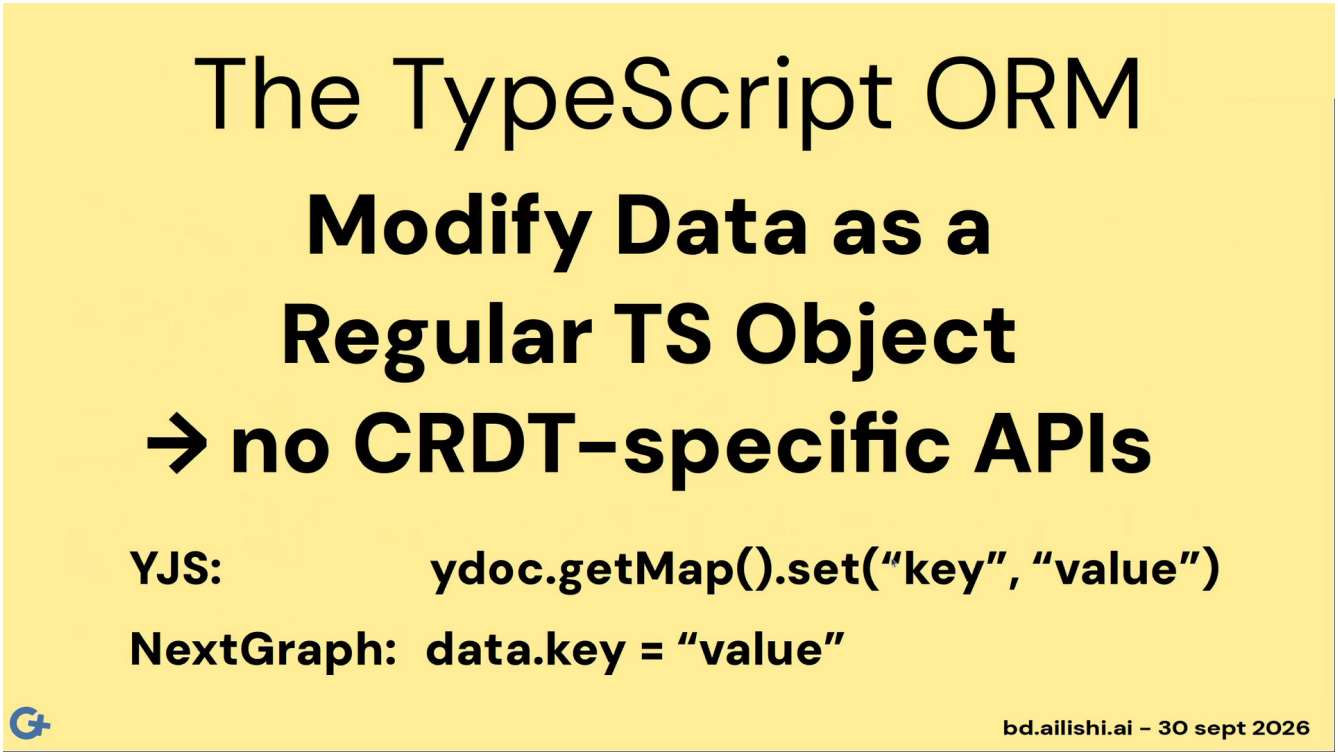
无论文档背后用的是哪种 CRDT，你都可以订阅并收到更新。我们还提供了更高层的 ORM API，把数据映射成 TypeScript 对象。对象可以带 schema，也可以不带，取决于具体情况。你可以把数据直接绑定到 Vue、Svelte 或 React 等前端框架。最近我们还加入了 SolidJS，注意不要和前面提到的 Solid 项目混淆。



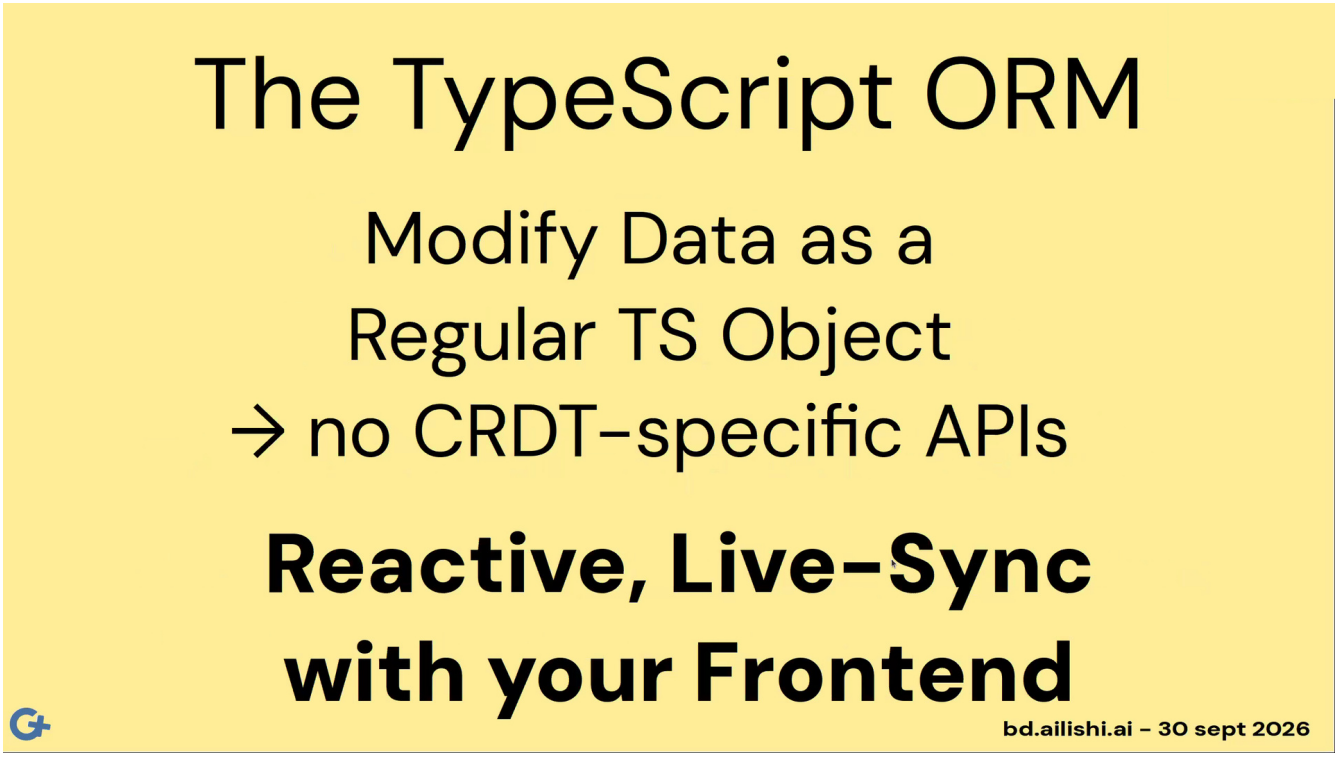
前端框架 • 00:15:29

00:15:39 • Niko Bonnieure

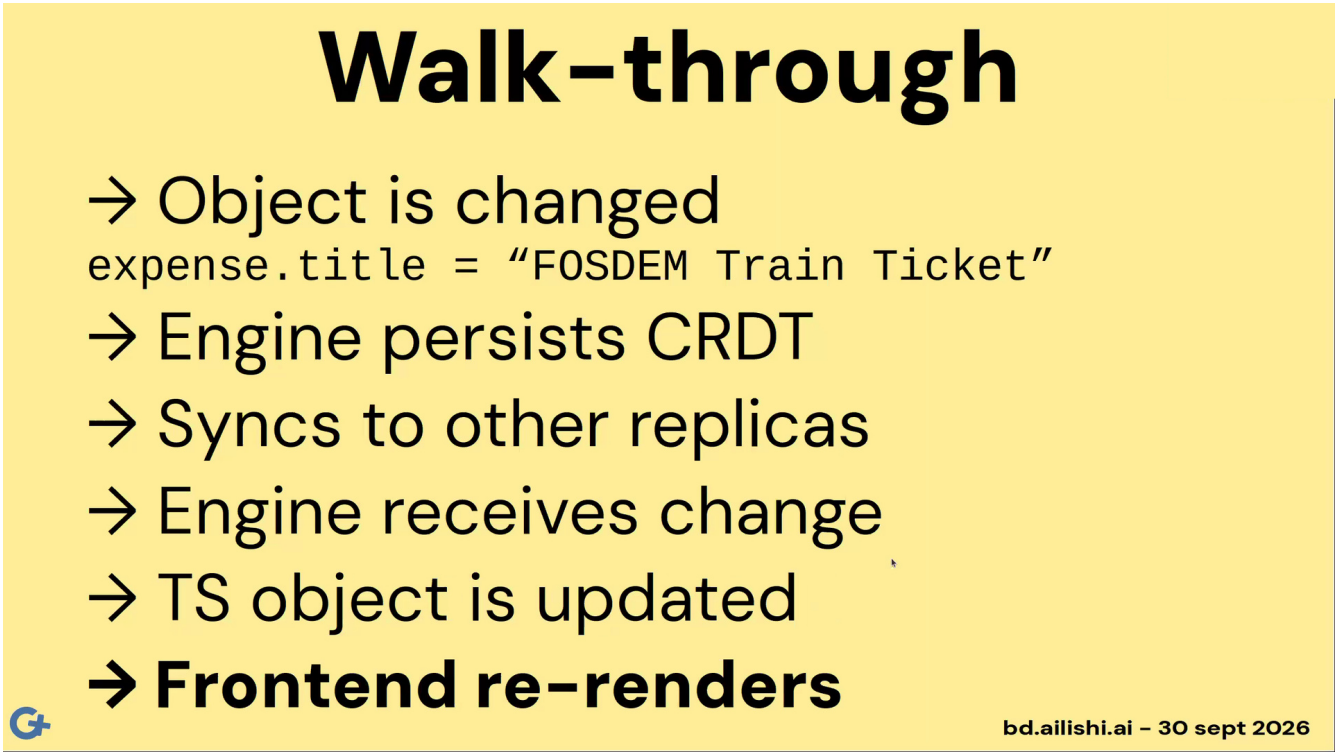
有了 TypeScript ORM，你就不用操心 CRDT 了。下面的例子里，Yjs 是我前面提到的 CRDT 库。直接使用它，订阅或修改数据需要了解 Yjs 的用法；在这里，你写的只是普通 TypeScript：object.key = value。修改数据后，会立即生成一个端到端加密的更新。如果在线，就通过网络发出去，其他人收到更新。这样就能获得前端响应性，也能实时协作文档。你改一个属性，本地引擎就把它作为 CRDT 操作持久化；在线时同步，不在线就保留到下次联网。



像普通对象一样修改数据 · 00:15:40



响应式前端与实时同步 · 00:16:19



数据修改与同步流程 · 00:16:30

00:16:48 · Niko Bonnieure

其他副本的本地引擎收到这些变化后，本地对象会更新，随后 React、Svelte 或 Vue.js 前端也会更新。这里展示了使用两个主要 API 的前端代码：useDiscrete 和 useShape。useDiscrete 用于基于 Yjs、Automerge 的文档；如果将来接入 Loro，也可以用于 Loro。useShape 用于 RDF，需要传入一个 shape，后面会解释。不管数据来自哪一种文档，使用方式最终都一样，都是响应式的。这里是一个 React hook，把它接到前端和模板上即可。刚才提到的 schema，可以为 TypeScript 提供类型和类型安全。

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 前端代码 · 00:17:08

Schemas & Typing

For RDF Data



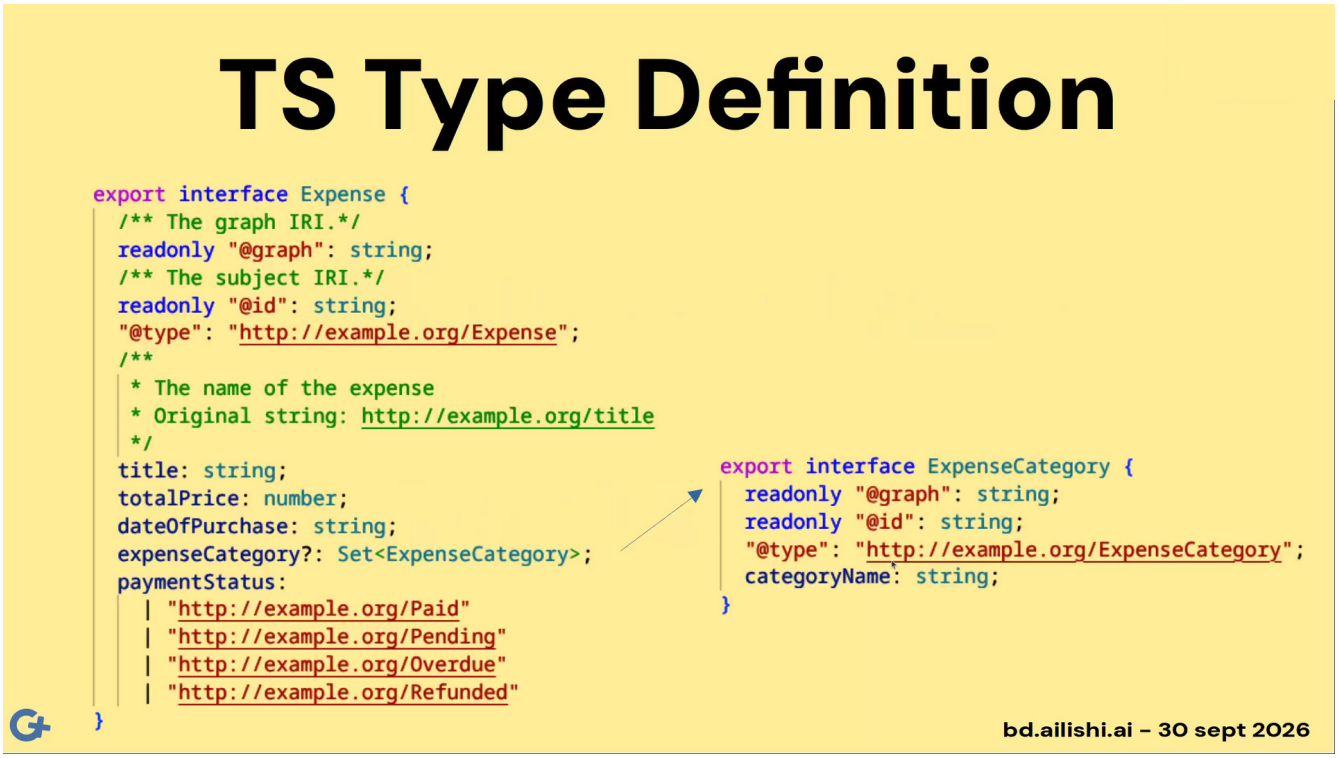
bd.ailishi.ai – 30 sept 2026

RDF 的模式与类型 · 00:17:47

ShEx 模式与类型安全

00:17:56 • Niko Bonnieure

这目前只适用于 RDF 文档，因为其他文档是 JSON，我们还不支持它们的 schema。我们使用 ShEx 格式描述 schema，就是常见的模式定义：有哪些属性、子属性、各自是什么类型，也可以有嵌套对象。大家可以在我们的 Git 仓库中找到这个费用追踪应用的例子。它可以运行，并且持续更新和测试；想上手的话，可以从这里开始。



TypeScript 类型定义 • 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 模式定义 · 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



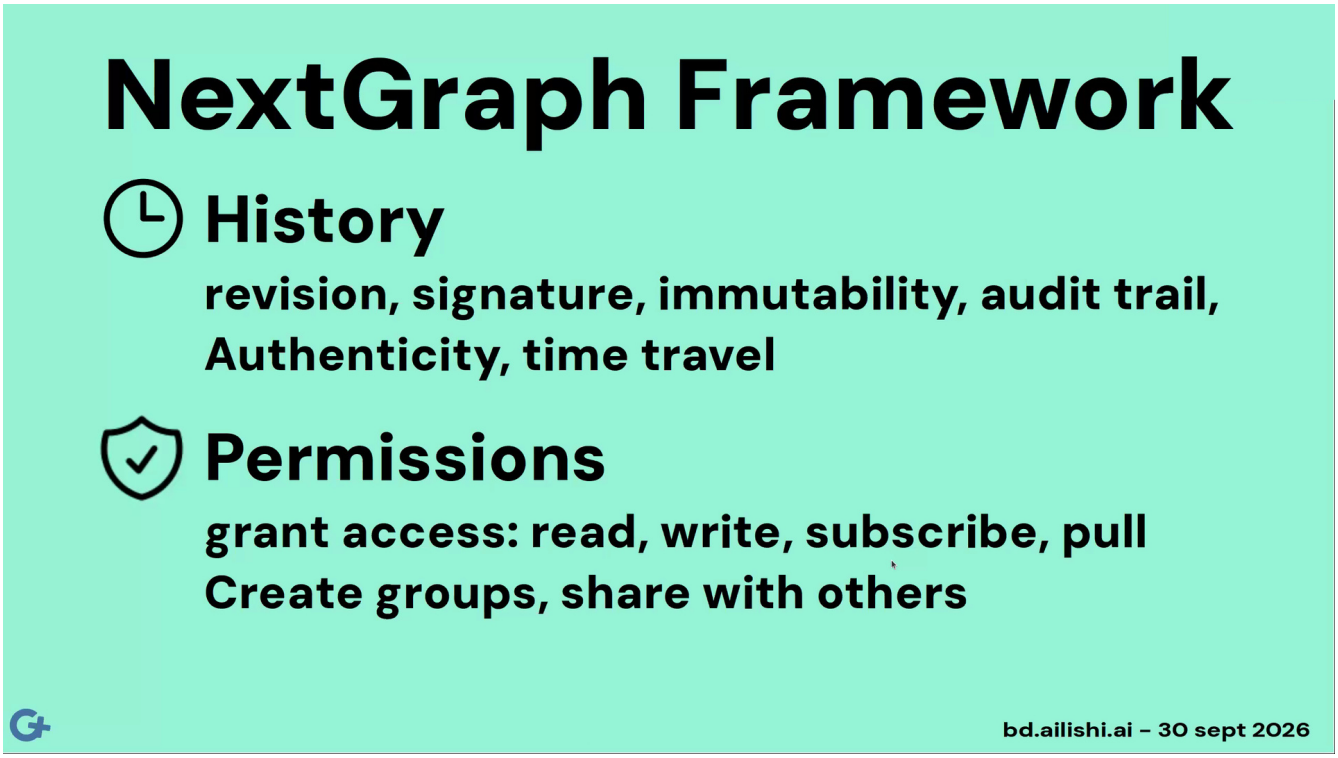
bd.ailishi.ai – 30 sept 2026

ORM 功能概览 · 00:18:25

历史记录、权限与响应式查询

00:18:44 • Niko Bonnieure

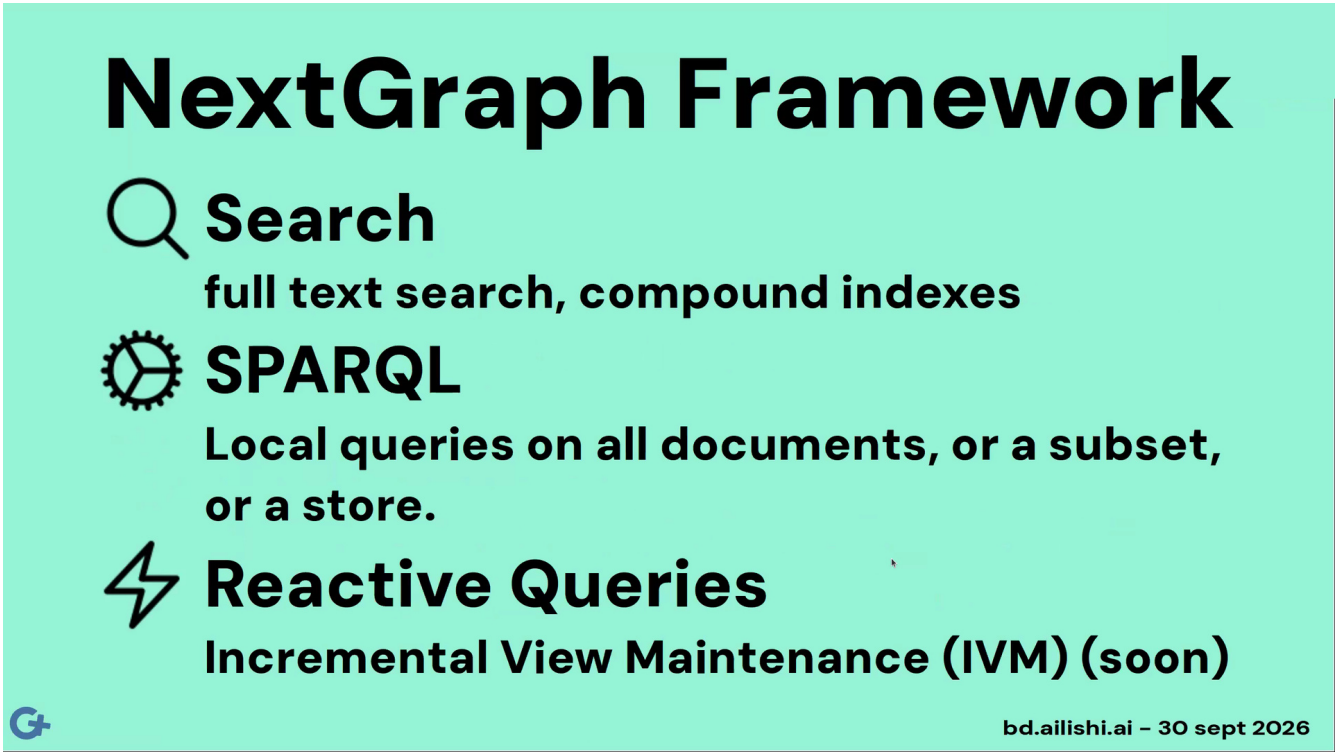
在 SDK 之上，我们还在开发一个框架。如果不同应用需要一些共通功能，我们就通过更高层的 API 提供出来，例如文档历史。你可以回到过去，查看以前的版本，也有审计轨迹，因为所有内容都有签名。



历史与权限 • 00:18:50

00:19:15 • Niko Bonnieure

每个提交都由作者签名，这份变更历史防篡改、不可变。准确地说，它并不是线性日志，而是 DAG，也就是有向无环图。还有权限机制：你可能不想重新做一套修改文档权限的界面，我们可以按需提供。前面提到的 SPARQL 则提供了本地查询引擎。



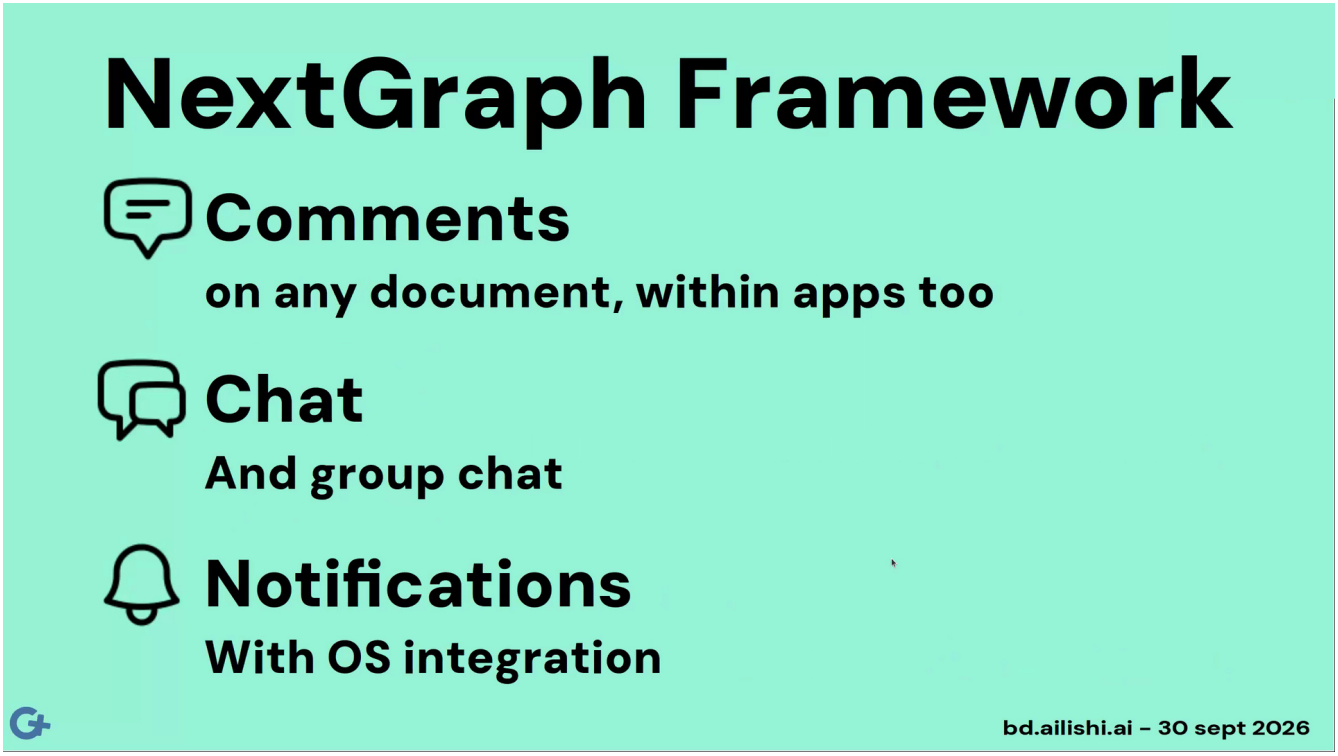
搜索与查询 • 00:19:44

00:19:50 • Niko Bonnieure

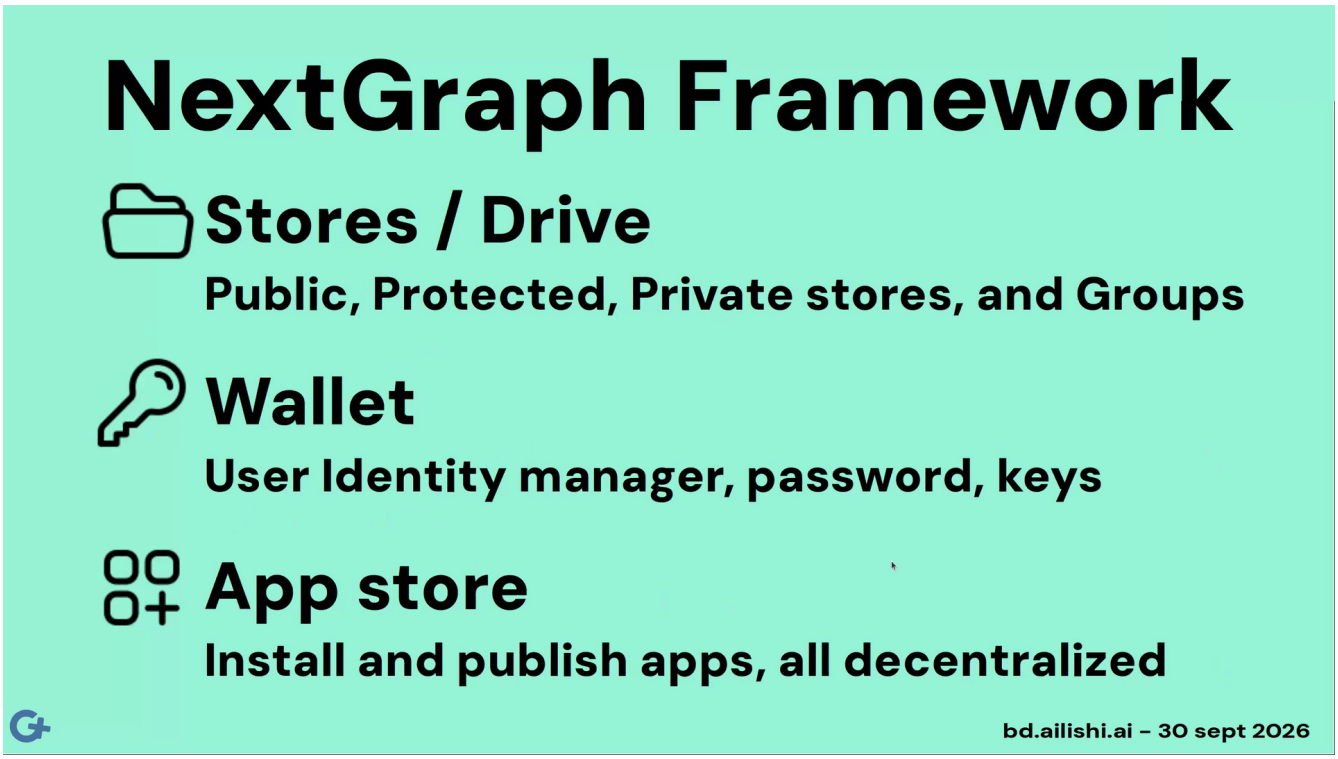
响应式查询还在开发中，不过进展不错。你发起一个 SPARQL 查询，并保留它的句柄，让查询持续存在。首先会收到初始结果；如果你还在显示这些结果、或者在视图中使用它们时，底层数据发生了变化，就会通过回调收到更新后的结果。

00:20:24 • Niko Bonnieure

例如，一条记录被编辑或者删除，你会自动看到变化，这就是响应式查询。评论、聊天、通知也是框架支持的实用功能。文档共享和数据保存是基础，上面还可以提供这些额外 API。



评论、聊天与通知 · 00:20:36



存储空间、钱包与应用商店 · 00:20:56

00:20:58 · Niko Bonnieure

钱包非常重要，主密钥保存在里面。钱包始终放在本地，服务器永远看不到它。还有应用商店和网盘。我们也计划做去中心化、注重隐私的社交网络，使用端到端加密。如果有时间，后面可以讲讲 ACID 事务和智能合约。概括一下，这是一个由 broker 组成的开放平台。

NextGraph Framework

Social Network

P2P web of trust private social network

Smart contracts

ACID transactions

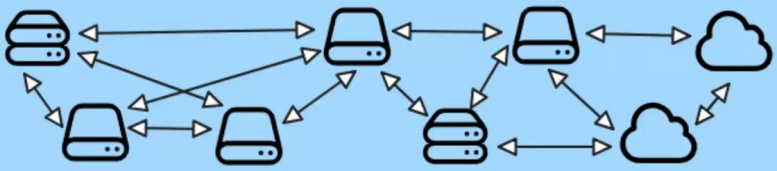
with strong consistency. Paxos coordination

bd.ailishi.ai – 30 sept 2026

社交网络与事务 · 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

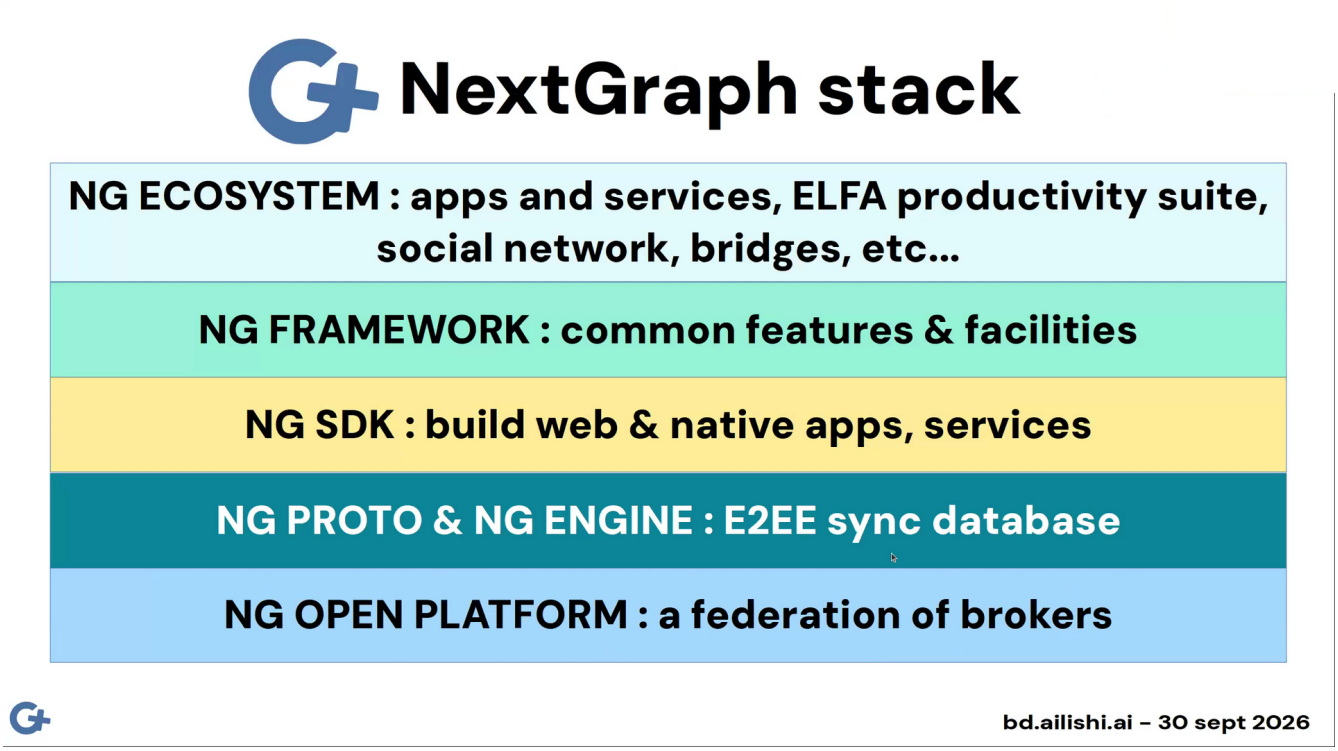
bd.ailishi.ai – 30 sept 2026

broker 联合网络 · 00:21:27

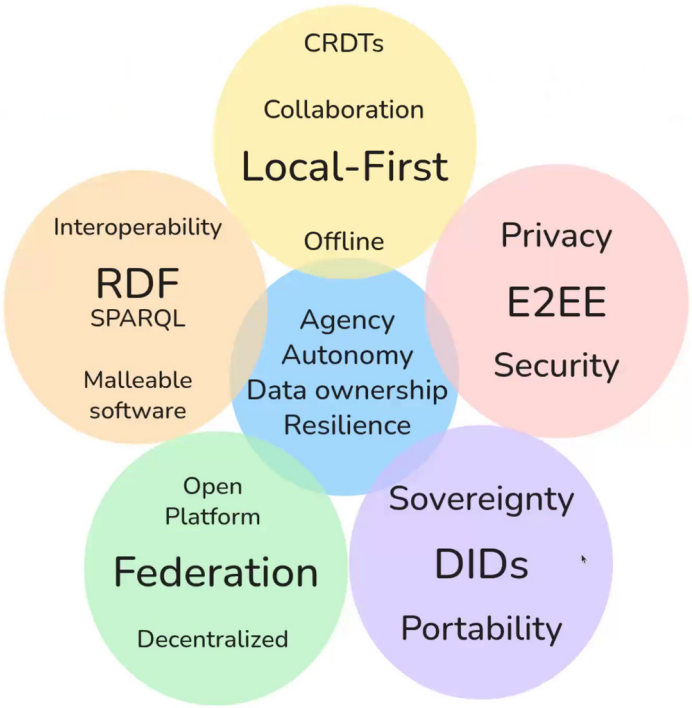
ELFA 与本地优先生态

00:21:31 • Niko Bonnieure

它是开源的，可以自行托管：部署在云上、租来的 VPS 上，或者自己的机器上都可以。它包含协议、SDK、框架，以及应用生态。我想提一下 ELFA。这些一般特性先跳过，之后会把 PDF 给大家。ELFA 是一套应用，我们四个月前开始做。它是一个完整的工作空间，有点像是在尝试与 Google Workspace 竞争，提供你需要的各种应用：文档、电子表格、电子邮件、日历、联系人等。



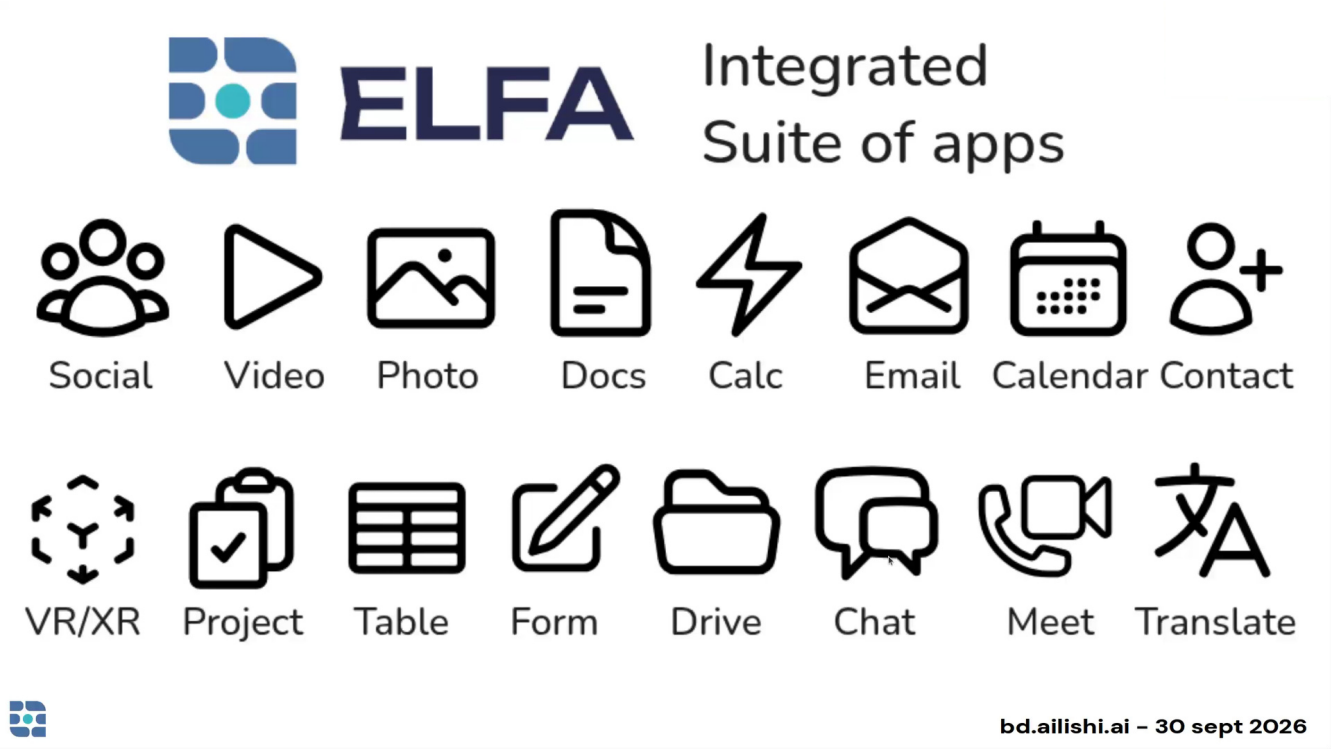
NextGraph 技术栈 • 00:21:39



NextGraph 的设计要素 · 00:21:47



ELFA 联盟 · 00:21:52



ELFA 应用套件 • 00:22:06

00:22:14 • Niko Bonnieure

它基于 NextGraph，所以具备端到端加密和本地优先的特性。预计一两年后准备好，近期会有一个 alpha 版本。这里是和我一起参与这个项目的合作伙伴。



合作项目 • 00:22:25

00:22:30 • Niko Bonnieure

最近，我还和屏幕上这些伙伴组建了一个更大的联盟，名单甚至有第二页。这是一个新的合作，包括 ELFA、NextGraph 和这些伙伴。他们并不全都使用 NextGraph，但会相互协作。大家将一起把本地优先带给普通用户，带进操作系统、移动操作系统和更多应用，推广到更广泛的开源生态里。



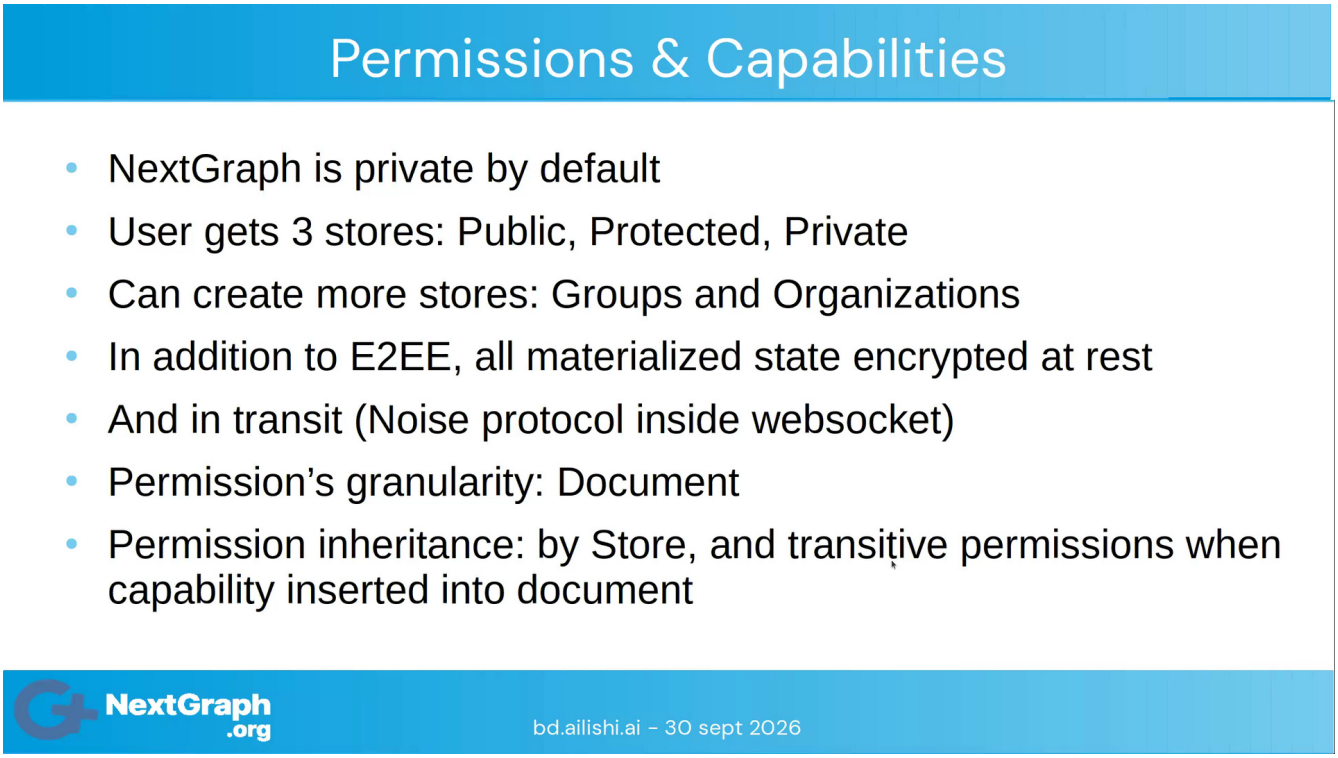
SUPER Stack 联盟（一） • 00:22:31



SUPER Stack 联盟（二） • 00:22:38

00:23:10 · Niko Bonnieure

接下来可以更深入地讲加密、权限这些内容。我先停一下，看看大家有没有问题，以及希望我展开哪一部分。



权限与密码学能力 · 00:23:13

00:23:30 · 主持人

也许可以讲讲密码学？

00:23:32 · Niko Bonnieure

好，密码学。我找不到聊天窗口。你们看到里面有问题吗？我没看到它在哪里，这里有聊天窗口吗？

00:23:55 · 主持人

有，不过我想目前还没有问题。

Store 与权限边界

00:23:59 · Niko Bonnieure

好的，看到问题就告诉我，可以随时打断。接下来讲能力和权限系统。所有数据都会组织进 store，文档按 store 分组，你可以把它理解为一个容器。store 有几种类型。private store，也就是私有存储区，数据只在你自己的设备之间同步，不在不同用户之间同步。

00:24:34 · Niko Bonnieure

你可以在里面保存秘密，其中的文档不能分享给别人。在 protected space，也就是受保护空间里，则可以通过不同权限和别人分享文档。还有 public store，即公共存储区，其中的数据任何人都能读，只要把链接给他们即可。这里不检查读取权限，但仍会检查写入权限。基本上，只有你自己能写，除非你把写入权限委托给别人。

00:25:07 • Niko Bonnieure

这是三种 store。此外，还有 group store，也就是群组存储区。你创建一个群组，邀请其他人加入，成员可以在群组里编辑和创建文档。所有这些 store 都有文档层级，可以放文件夹、子文件夹，按自己的方式组织文档。前面也说过，我们还对传输中的数据加密。

00:25:40 • Niko Bonnieure

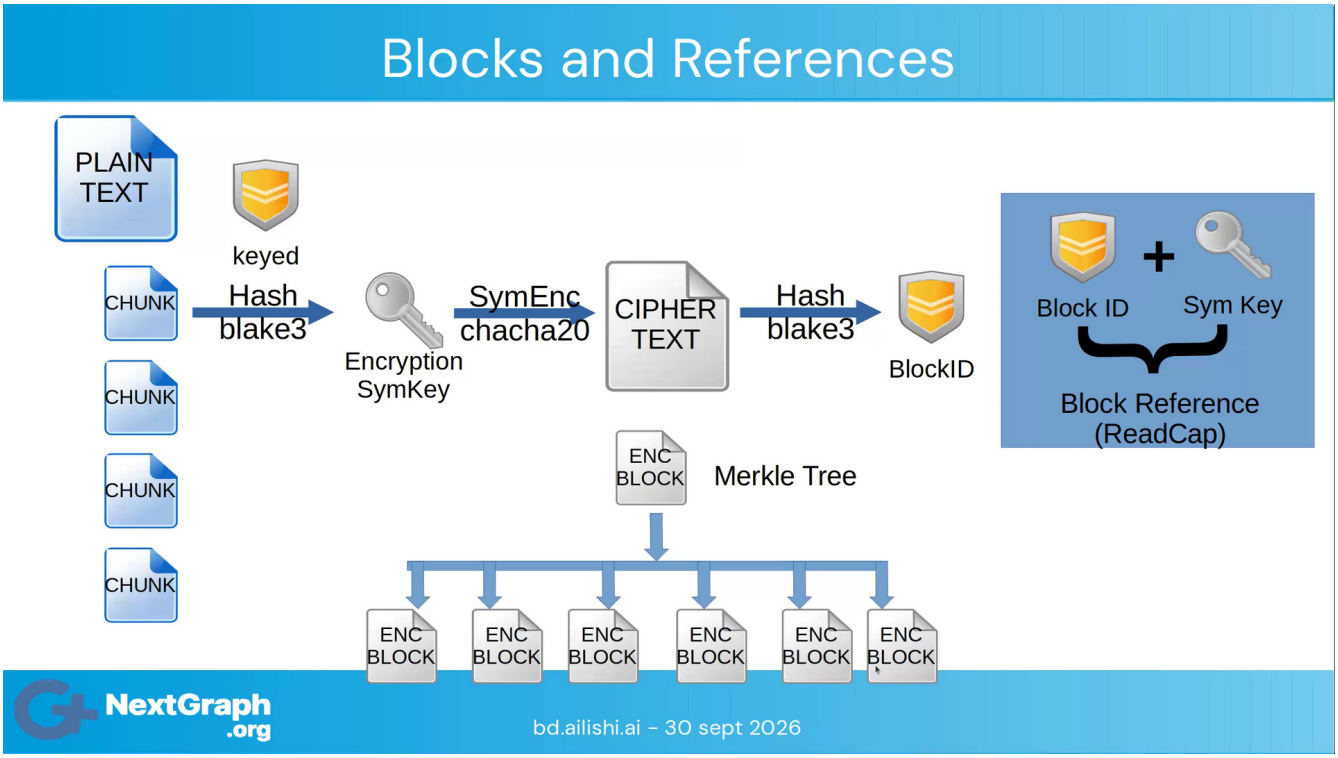
我们有三种不同层面的加密。设备之间、broker 之间的所有通信都使用 Noise Protocol。

00:25:52 • Niko Bonnieure

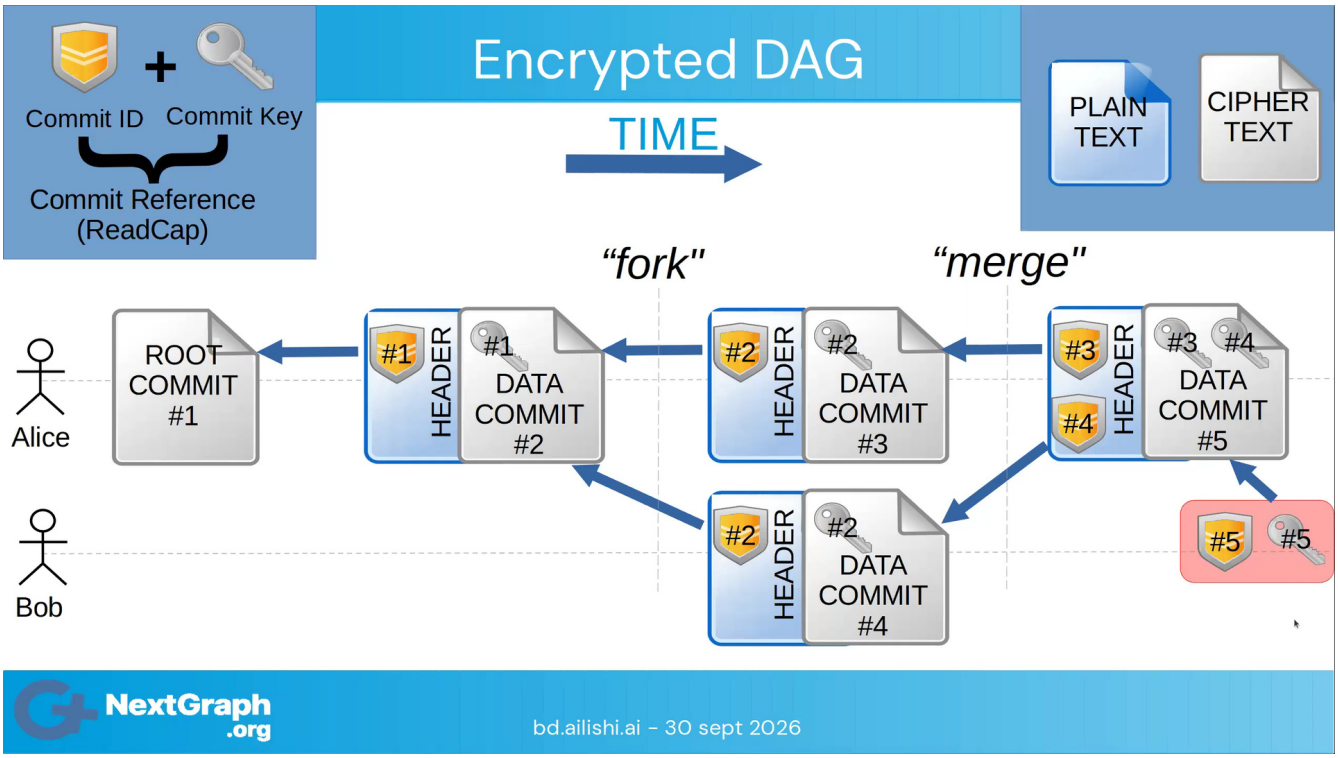
权限的粒度是在文档层面，不过也不完全是。写入权限按文档设置，你可以指定谁能编辑这份文档。读取权限则更细，可以细到我们所说的 branch。你可以把文档中的数据分成不同分支，让某些部分可读，而另一些部分隐藏起来。

00:26:25 • Niko Bonnieure

写入权限不细分到这个程度，它仍在文档层面。权限有继承关系：文档继承所在 store 的权限，但你也可以不沿用继承结果，单独修改某份文档的权限。下面进入协议内部加密的技术细节。我觉得这部分有必要讲清楚，先从第二张幻灯片开始。



数据块与引用 • 00:26:44



加密的有向无环图 · 00:26:59

加密数据块与提交

00:27:01 · Niko Bonnieure

这一部分讲的是我们如何做端到端加密，以及它怎样与 CRDT 结合。我认为，把两者结合起来时，这一点非常重要。每次更新都是一个原子单元，我们把更新放进一个数据块，并单独加密，因为更新随时可能发生。

00:27:33 · Niko Bonnieure

比如，你打开一个文本文档，修改了三个字符，这次编辑就会作为一个数据块被加密。加密的只是这次变化，也就是补丁、CRDT 更新。我们采用对称加密，所以会有一个密钥。看左上角，我们把包含更新的这个块叫作 commit，也就是提交。每个提交都有 ID 和密钥。

00:28:06 · Niko Bonnieure

这里的密钥是对称密钥，ID 是加密后数据块的哈希。如果我把提交 ID 和密钥给某个人，他就可以获取这次更新。

00:28:25 · Niko Bonnieure

我们需要把 ID 和密钥传给想要分享更新的其他节点、其他副本，也就是订阅这份文档的人。有了 ID，他们就能向 broker 请求数据块。broker 保存着所有这些端到端加密的数据块，你给它 ID，它就返回相应的密文数据。

00:28:59 · Niko Bonnieure

有了密钥，就能解密。当然，密钥不会存放在服务器上。先从文档的第一个提交开始，我们叫它根提交；如果借用区块链的说法，也可以叫创世提交，不过这里说的只是文档。第一个提交里有一些元数据，然后图中的第二个数据提交包含第一次更新。

00:29:31 • Niko Bonnieure

第二个提交包含第一次更新，并通过提交 ID 引用第一个提交。这个标着数字 1 的黄色部分就是引用，表示父提交的 ID，有点像 Git。注意，这里还没有提交密钥。之后再增加提交时，也是沿着这个方式继续。

并发提交与 DAG

00:30:07 • Niko Bonnieure

第三列里，提交 3 和提交 4 是并发的。它们可能由不同用户创建，也可能是同一个用户的两个副本、两个标签页创建的，但当时没有联网。所以它们并发发生，并且都指向同一段过去，也就是提交 2。生成的这两个提交，都以提交 2 为父提交。

00:30:44 • Niko Bonnieure

等它们上线后，就会把这些提交发给 broker。其他订阅者收到两个提交，就能看出它们是并发的。数据本身没有问题，因为基于 CRDT，并发更新可以由 CRDT 处理好。不过，这里我们主要讨论加密这一层。

00:31:11 • Niko Bonnieure

两个提交会被解包、解密，并在本地物化，这样就可以工作了。此时我们停留在这个状态，图中后面标注的 merge 还没有发生。数据已经收到，本地文档也已经更新。之后又发生了一次提交，也就是提交 5。某种意义上，它把前面两个提交合并起来，指向提交 3 和提交 4，正如最后一列所示。

00:31:48 • Niko Bonnieure

假设提交 5 是这份文档当前最新的提交，那么我需要分享的就是它的 ID 和密钥。红色框里画的就是提交 5 的 ID，以及提交 5 的密钥。

通过发布订阅分享密钥

00:32:17 • Niko Bonnieure

我确认一下大家有没有跟上。好的。那么，怎样分享这些信息，又不把它们直接交给服务器呢？我们先加密，再通过 pub/sub，也就是发布订阅机制发送。其他副本收到的仍然是密文。

00:32:48 • Niko Bonnieure

这里用的是另一个密钥，暂时不展开。副本之所以能够解密，是因为它们属于这份文档的发布订阅范围，是经过授权的读者。只有这些副本能解开红色框中的信息。解开后，就知道“现在到了提交 5”，于是向 broker 请求对应的数据块，再用密钥解密。

00:33:19 • Niko Bonnieure

如果你已经订阅，我们会直接把数据块推送给你，不需要另行请求。如果晚一些才订阅，就需要获取缺失的提交，但这些工作由协议和库自动完成，不用你手动处理。还有一个协调差异的机制，因为你可能已经有文档的一部分，只缺少其中某些提交。

补齐缺失提交与 broker 联邦

00:33:56 • Niko Bonnieure

你只需要和自己的 broker 通信，不必联系其他副本。前面说过，这不是设备直接相连的 P2P。作为一个副本，你告诉 broker：“我有这些块。”broker 会说：“你还缺两三个块。”然后把缺失的块发过来。这也解释了为什么图中黄色的部分，也就是数据块 ID，没有加密。

00:34:32 • Niko Bonnieure

我们能看到这些 ID，broker 也能看到。承载这些信息的头部是明文，幻灯片右上角也注明了。这让 broker 能看到 DAG 的形状，也就是所有提交组成的图。它可以提前重建这个结构，等某个副本来请求“只给我缺失的最新提交”时，就知道该发哪些。

00:35:12 • Niko Bonnieure

broker 之间也会同步，因为这是一个联邦系统。我可以把数据放在一个 broker 上，另一个人放在另一个 broker 上，仍然可以协作同一份文档。即使我们的设备离线，这些 broker 也会继续交换数据。这样，当我上线时，需要同步的数据就已经到了我的 broker 上。

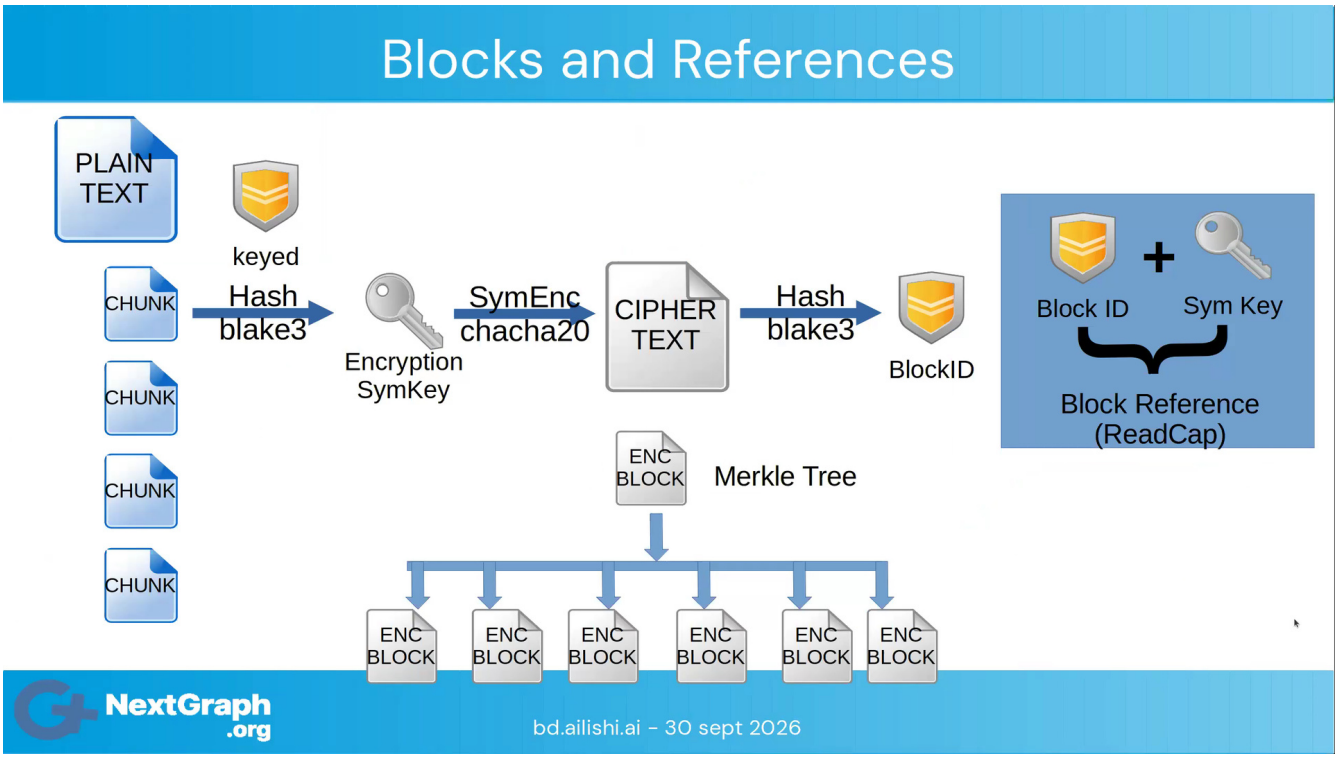
00:35:49 • Niko Bonnieure

这样就会更快。broker 已经准备好了我缺失的最新提交，连接后只需要接收这些，而且会按正确的偏序顺序收到。这个顺序可以从 DAG 看出来，知道什么发生在先，也有助于后续处理 CRDT。我认为，这是对本地优先数据进行加密的正确方式。

Merkle 树与分块

00:36:24 • Niko Bonnieure

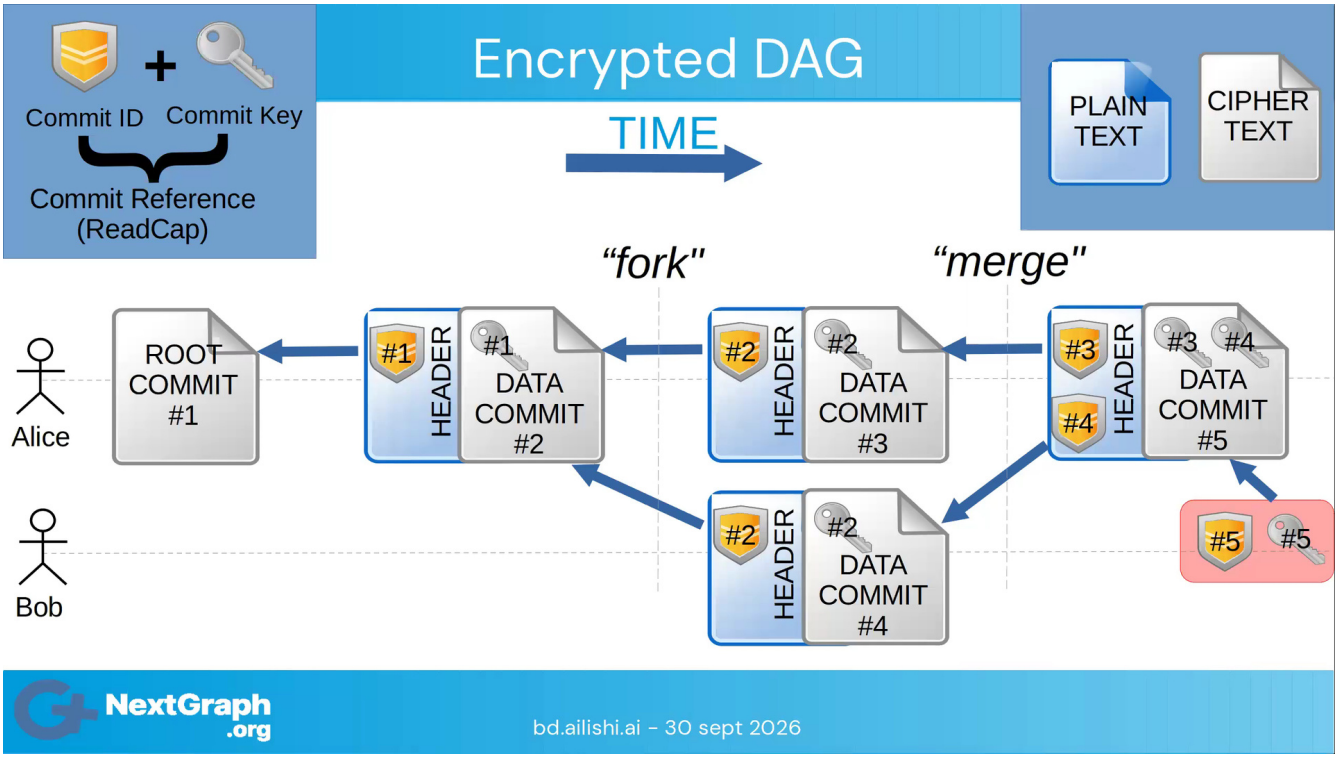
也有其他库采用相同方式。回到前一张幻灯片，这里展示了我们如何具体处理包含数据的数据块、块 ID 和对称密钥。数据块内部有 Merkle 树。如果一次更新很大，超过一兆字节，就会拆成块，有点像 BitTorrent。



数据块与引用：回看 · 00:36:28

00:36:55 · Niko Bonnieure

这张幻灯片讲的就是这些，以及前面提到的对称加密和哈希。技术部分先讲到这里，如果有问题可以现在问，然后我再往下讲。这部分清楚吗？



加密 DAG 与现场问答 · 00:37:07

00:37:29 · Niko Bonnieure

抱歉，我没有听清你的问题或评论。

00:37:34 · 主持人

他在聊天里留了一个问题。

00:37:37 · Niko Bonnieure

聊天窗口在哪里？我看不到。

问答：像一个加密的 Git?

00:37:43 · 主持人

他在问，这个结构是不是像一个加密的 Git?

00:37:54 · Niko Bonnieure

是的，是类似的。主要区别是，Git 里不能有两个以上的父提交。合并两个分支时，一个提交可以有两个父提交。这里则可以有三个、四个或五个。如果三台设备在断网时并发编辑，就会有三个父提交。这是与 Git 类比时的区别。

> 编者注：Git 也支持有两个以上父提交的 octopus merge。这里保留讲者原意，相关机制见 Git 官方文档：https://git-scm.com/docs/git-merge 。

问答：规模与基准测试

00:38:37 · 观众

你好，Niko，谢谢你的分享。我有两个问题。第一个是关于 NextGraph 的容量：你估计目前这个系统可以支持多少用户、多少设备进行实时的端到端加密协作?

00:39:02 · Niko Bonnieure

我们还没有这方面的基准测试，抱歉让你失望了。我预计参与人数较少时应该能运行得不错，但不知道规模扩大后会怎样。CRDT 部分取决于所用 CRDT 库的性能；网络本身还没有做基准测试，不过我们应该开始做了。

问答：OrbitDB、快照与协调

00:39:40 · 观众

第二个问题是，和 OrbitDB、Delta 这样的其他开源项目相比，你们的路线有什么不同?

00:39:58 · Niko Bonnieure

能再说一下项目名字吗？我没听清你提到的是哪几个。

00:40:05 · 观众

OrbitDB。

00:40:07 · Niko Bonnieure

OrbitDB，好的，我知道这个项目。另一个呢?

00:40:11 · 观众

Delta。

00:40:12 • Niko Bonnieure

Delta，我不确定自己是否了解。OrbitDB 是一个基于日志的数据库、键值存储，需要操作日志。上一次我试用它是很久以前，当时需要重放整个日志才能访问数据。我不知道现在是否还是这样。我在 NextGraph 里尝试解决这个问题，把数据库真正保存在本地，就不用每次都重放一切。

00:40:58 • Niko Bonnieure

每次启动应用或者访问文档，都有一个可以直接访问的本地数据库。例如，RDF 部分使用 Oxigraph。它有静态加密，但查询仍然很快，因为数据就在本地，不需要重放全部历史。对于 CRDT，每次重放、重新物化状态的开销很大，我们想避免这部分成本。

00:41:31 • Niko Bonnieure

主要问题出现在新加入的人身上。你把某人加进一份他从没见过的文档，而文档已经用了整整一年，可能有一万个提交、一万次更新。要重放全部历史，首先得把它们都取回来。broker 可以帮忙，一次发送所有数据，但接收方仍需逐个解密提交。刚才也看到了，这是一个链式的 DAG，要从最新提交开始，往历史回溯。

00:42:04 • Niko Bonnieure

这在加密层面就很慢，之后 CRDT 重新物化全部状态也需要时间。要改善这一点，可以做快照：压缩历史、墓碑标记和不再有用的内容，生成一个可以一次载入的文档，不必重放所有操作。这个快照也只需要加密一次。

00:42:41 • Niko Bonnieure

这就是快照，NextGraph 里已经有了。需要注意的是，做快照时需要与其他参与方进行一点协调。协议会负责，你不需要手动操作，但这一点并不是本地优先的：至少需要自己的 broker 配合，所以必须在线。broker 可以处理这件事，不过要生成这样的快照，你得先连接 broker。

00:43:13 • Niko Bonnieure

还有一些人在尝试不需要协调的快照，比如 Ink & Switch 那些做 Automerge、Keyhive 的人。这些都是取舍。他们做了很好的工作，但我还没有把这些成果整合进 NextGraph。我们的快照仍然需要协调，因为它不是一件每天都要做的事。

00:43:45 • Niko Bonnieure

我们并不会永远离线工作，总要同步。这里说的是间歇性的断网。如果你一个月都不联网，那就没有和别人协作，只是在本地放着一个文件。在这种情况下，你只在本地编辑，从不上线，我们实际上会在本地做快照，已经把内容压缩起来了，因为知道你没有联网，不存在这方面的风险。所以我认为，这是可以接受的取舍。

00:44:29 • 观众

这样也更简单。

问答：编程智能体与 AI 访问本地数据

00:44:31 • 观众

我还很好奇，像 Claude Code、Codex 这样的智能体编程工具，现在有很多功能需要改进或加入。根据你的经验，这些工具对开发效率有什么帮助？

00:44:55 · Niko Bonnieure

你是在说智能体，也就是 AI，对吧？

00:44:58 · 观众

是的。

00:45:01 · Niko Bonnieure

首先，在 NextGraph 和我自己的工作中，我其实不怎么用 AI。有一些和我合作的开发者会用它写代码，但用得很少。如果你想问的是，怎样把 AI 接到这些数据、数据库或者 NextGraph 上，也许你更想问这一点？

00:45:26 · 观众

两方面都可以理解，我都感兴趣。

00:45:29 · Niko Bonnieure

你原来想问哪一方面？

00:45:33 · 观众

我是想问，你们计划加入什么样的 AI 功能？

00:45:39 · Niko Bonnieure

我们没有这方面的计划，不过去年已经有人这么做了。他们使用我们的 API，把它接到了 AI 上。

00:45:54 · Niko Bonnieure

我们有一个演示，可以发给你。我觉得效果不错，原因在于 SPARQL 这个查询语言。顺便说一下，我们用的是中国的 Qwen 模型，它很强，也懂 SPARQL。只要在上下文中放入如何访问数据的说明，大模型就能用 SPARQL 查询你的本地数据。

00:46:29 · Niko Bonnieure

演示里做了一个联系人管理器，存储联系人资料、电话号码等。我们问："Niko 或另一位联系人的电话号码是多少？"它就能查出来。

00:46:48 · Niko Bonnieure

它使用 SPARQL，甚至没有用 MCP，也不需要 MCP。因为这些 API 在 Python 和 Node.js 里都可以调用，所以从大模型发起查询很方便。

00:47:06 · Niko Bonnieure

当然，这在一定程度上打破了端到端加密的边界，因为大模型运行在服务器上，需要访问它要读取的文档和数据库。你得授予它访问权限，它才能查询。如果自行托管 Qwen 3，隐私会好一些；如果把数据交给 OpenAI，数据就可能外泄。

00:47:35 · 观众

不错。

00:47:36 · Niko Bonnieure

NextGraph 那边有一个代码仓库，名字大概是 LLM router。你可以联系我，我可以带你看一下。

00:47:47 · 观众

谢谢。

00:47:50 • Niko Bonnieure

谢谢，还有其他问题吗？

00:48:02 • 主持人

你还有问题吗？

00:48:07 • Niko Bonnieure

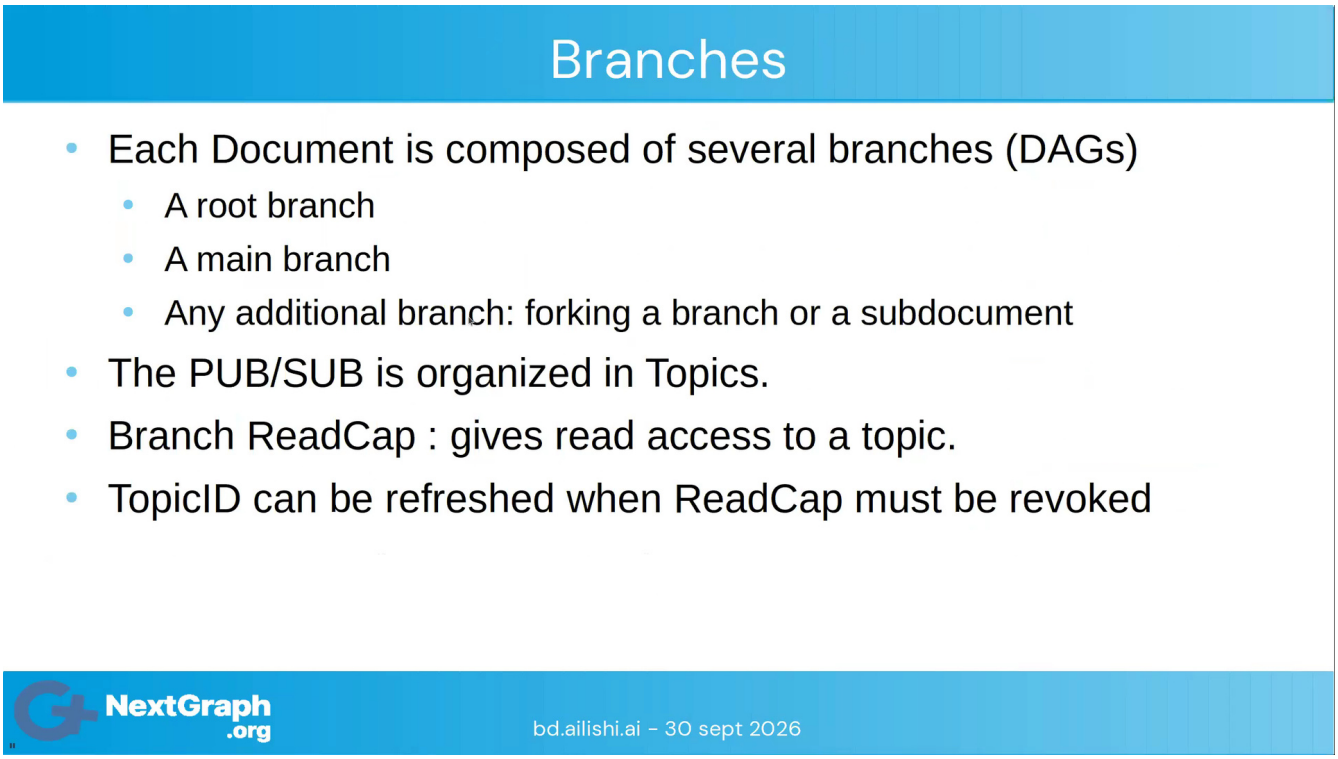
什么？

00:48:13 • 主持人

看来没有其他问题了，那你可以继续。

00:48:19 • Niko Bonnieure

我可以回到幻灯片。后面还有很多页，我之前不知道会花多少时间。这里有同步协议，以及提过的分支。如果没有大家特别想了解的内容，我们也可以很快收尾。这些前面讲过了，发布订阅也稍微提到了一点。之后我会把幻灯片给大家，你们可以慢慢看，到我们的论坛上提问。顺便说，论坛用的也是 Zulip。



文档与分支 • 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.

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

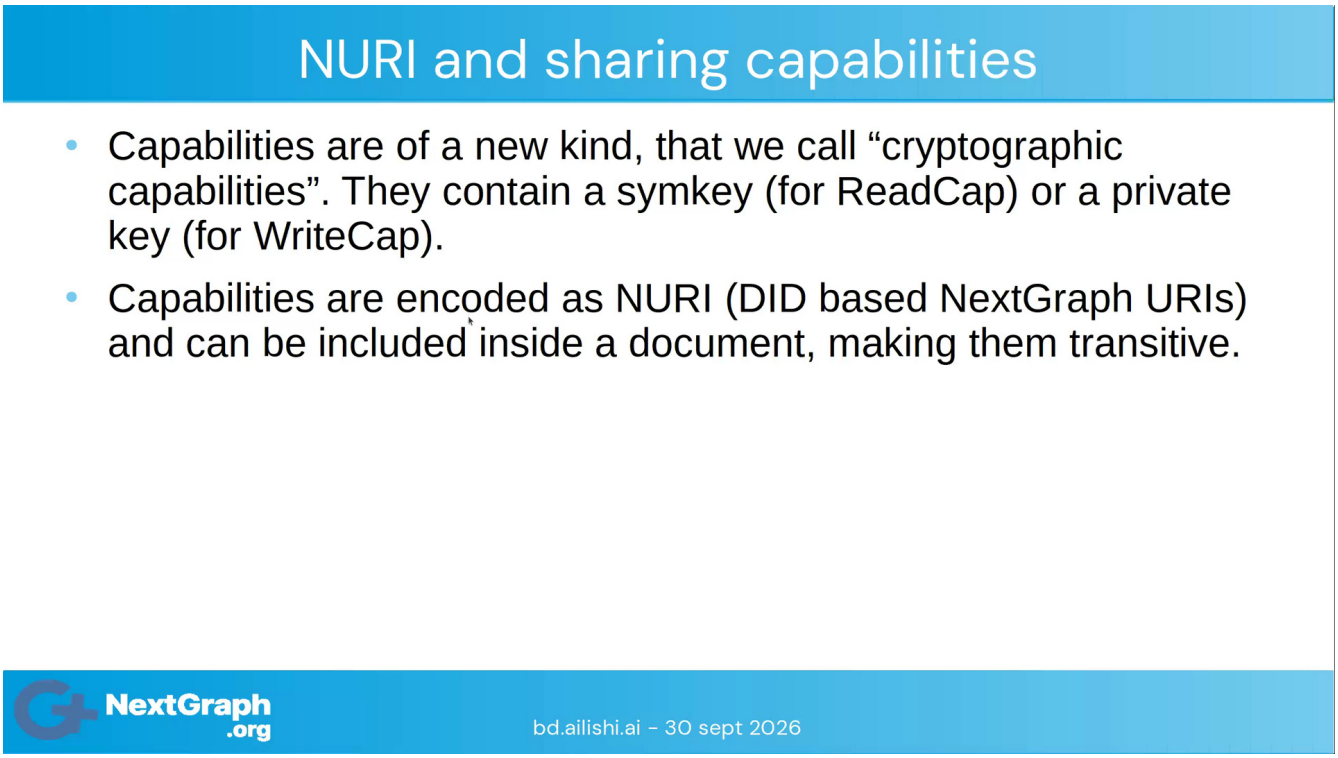
同步协议（一）・00:48:33

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.

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

同步协议（二）・00:48:42



NURI 与共享能力 • 00:48:46

00:48:55 • Niko Bonnieure

前面说过，读取能力基于对称加密，写入能力基于非对称加密。

拉取控制与后量子密码学计划

00:49:06 • Niko Bonnieure

DID 前面讲过了。overlay 这里不深入展开，它用于拉取控制，也就是 pull control。我们甚至希望阻止未授权的人拿到加密数据。并不是像前面简化解释的那样，只拿着一个数据块 ID 或提交 ID，就能直接向 broker 索取数据；没有这样的公开 API。你必须属于一个 overlay，这一层同样有加密和安全机制。只有获得了这个 overlay 的访问权限，才能联系 broker，请求特定 ID 对应的数据块或提交。

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

拉取控制与覆盖网络 · 00:49:06

00:49:51 · Niko Bonnieure

因为我们不知道加密数据将来会怎样，也许有一天会有人能够解开。我们已经有后量子密码学的计划，准备明年加入，也为此获得了更多资助。目前协议中的密码机制基于椭圆曲线，还不能抵抗量子计算，之后会支持。

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

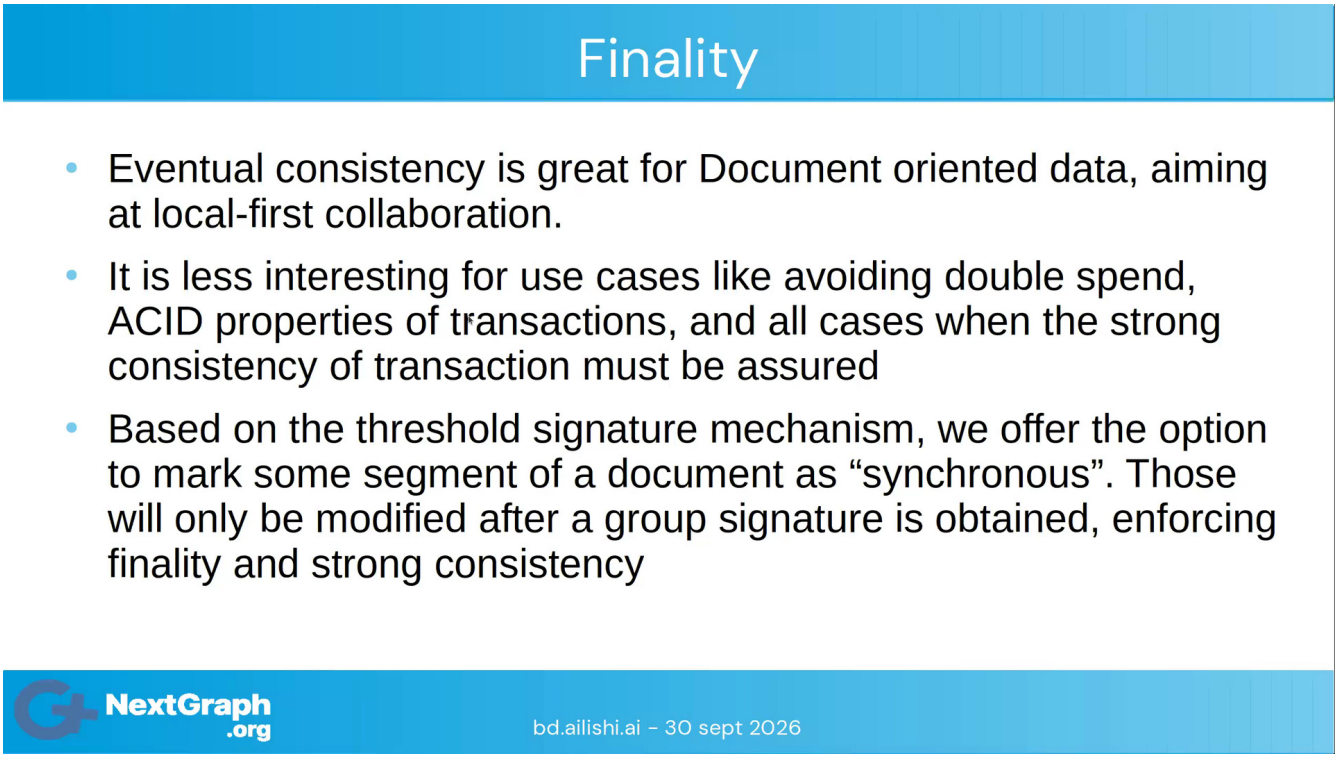
NextGraph .org bd.ailishi.ai - 30 sept 2026

外部签名 · 00:50:23

ACID 属性与计划中的 Paxos 协调层

00:50:24 • Niko Bonnieure

前面还讲了所有数据上的签名、不可变的审计轨迹，以及数据真实性。接下来是 finality，也就是最终确定性，这和我想讲的 ACID 提交、数据库有关。在本地优先模式下，修改数据不需要协调，我在离线时也可以修改。



最终性与强一致性 • 00:50:29

00:50:57 • Niko Bonnieure

这打破了我们习惯的数据库 ACID 属性。例如 PostgreSQL 这样的数据库部署在服务器上，我连接后执行一个事务，事务里可能包含多项修改。提交时，要么所有数据全部更新，要么完全不更新，没有中间状态。这对资金管理或者电商非常重要。

00:51:31 • Niko Bonnieure

比如一笔订单，你不希望把同一件东西卖两次，也不希望把同一笔钱花两次。CRDT 和本地优先本身解决不了这个问题。你可以在这里改数据、在那里也改数据，不需要协调，也不需要互联网。但等到合并、协调这些变化时，CRDT 并不关心你是不是把东西卖了两次，或者把钱花了两次。

00:52:02 • Niko Bonnieure

这是两个相互独立的问题，不能靠 CRDT 来解决。所以，我们想把 ACID 属性作为附加能力带回 NextGraph。大多数时候，你只需要 CRDT 和本地优先；但有些时候，你希望数据库具备原子性、一致性等 ACID 属性。

00:52:24 • Niko Bonnieure

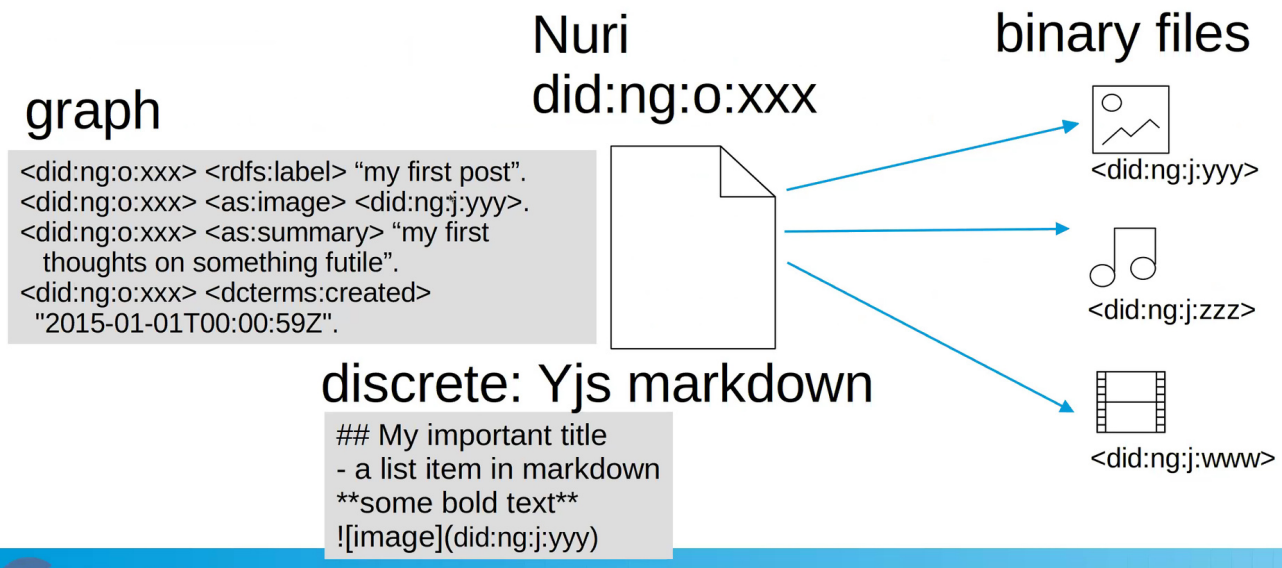
这部分我们还没有做出来，已有设计，但尚未编写代码。我们会通过 broker 中介进行协调，同时仍然采用端到端加密。broker 不知道你在做什么，但会帮助协调。它基于 Paxos 这个协调算法。在此基础上，就可以做智能合约等需要的功能。

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.

同步事务・00:52:49

NextGraph Document: Rich Text



富文本文档示例・00:52:54

文档结构：图、discrete 与文件

00:52:57 • Niko Bonnieure

这里展示文档的结构。文档有一个 ID，叫 NURI，意思是 NextGraph URI，它是 did:ng 形式的 DID。文档内部可以有 graph，也就是 RDF 部分。这里是一个 RDF 例子，graph 下的每一行都是一个三元组，由主语、谓语和宾语组成。感兴趣的话可以多了解 RDF，我很喜欢它。

00:53:27 • Niko Bonnieure

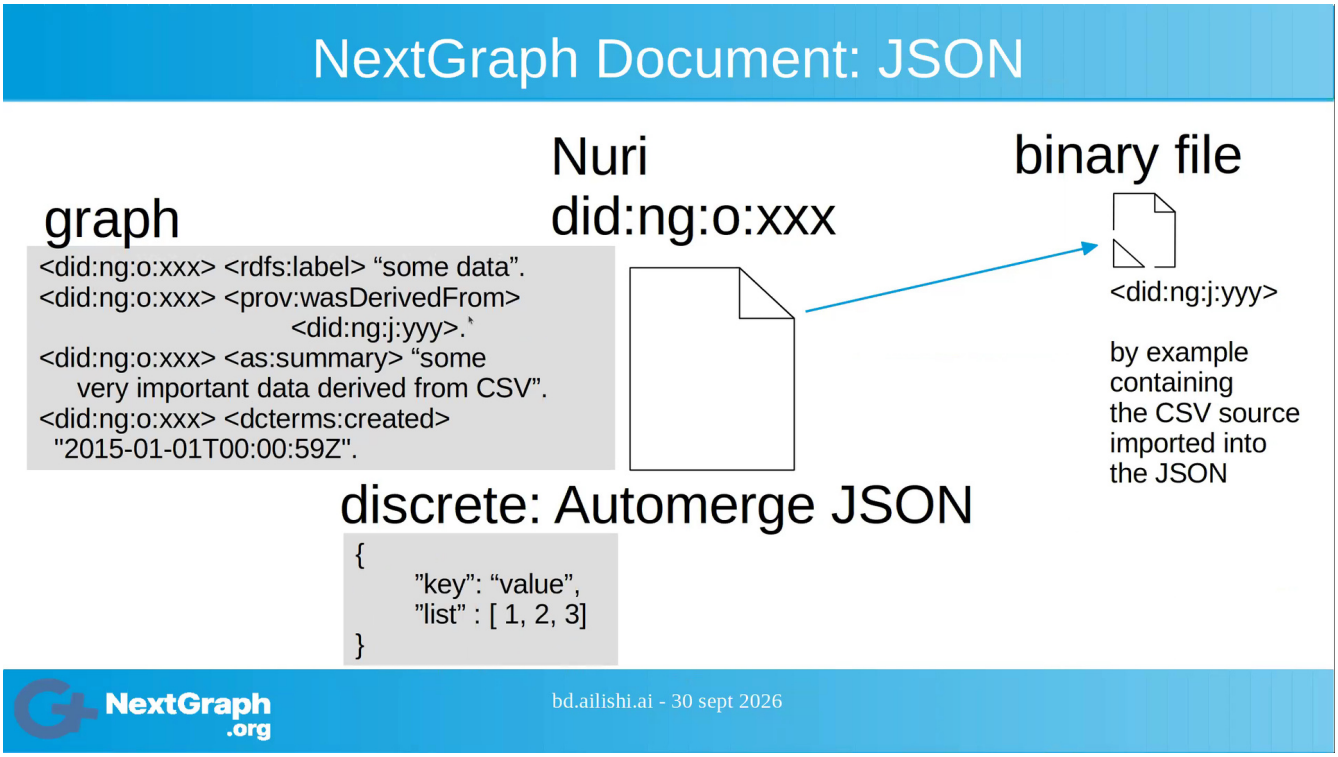
在场的另一位分享者也很喜欢，接下来几天大家可以问他。你什么时候分享？明天吗？

00:53:37 • 观众

周五。

00:53:38 • Niko Bonnieure

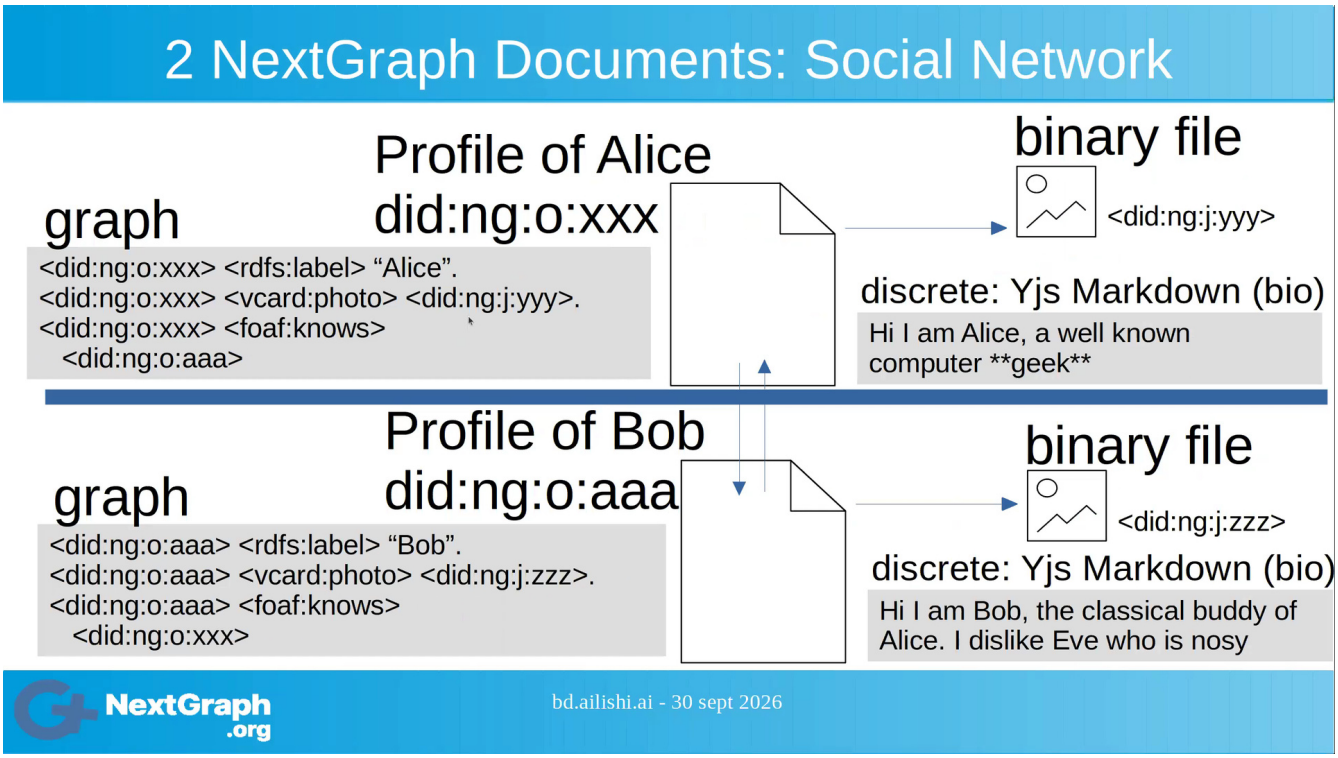
周五，好的。也欢迎大家来听他的分享，会很有意思，还可以在那里问 RDF 的问题。discrete 是文档的另一部分，或者说文档中的那些分支，你可以在里面放 Yjs 数据，比如 Markdown、富文本，也可以放 JSON 数据。



JSON 文档示例 • 00:54:04

00:54:09 • Niko Bonnieure

graph 和 discrete 这两部分可以同时存在于同一份文档里，这一点很重要。还可以附加二进制文件，文件上传到 broker 后可供访问，你会得到一个指向二进制数据的 URI，可以把这个链接存到图里或者 JSON 里。



社交网络文档示例 · 00:54:36

00:54:37 · Niko Bonnieure

如果做社交网络，可以有用户资料、社交数据、私信和聊天。基于 NextGraph 构建社交网络是可行的。我们到这里就讲完了，屏幕上是我的联系方式，也可以到网站上找到我。谢谢大家。



联系方式与交流 · 00:54:54

00:55:09 · Niko Bonnieure

谢谢大家一起度过这段时间。希望你们基于 NextGraph 做出很多应用。需要帮助的话就告诉我。如果想进一步了解本地优先、加密或者相关概念，我也一直很愿意回答问题。

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