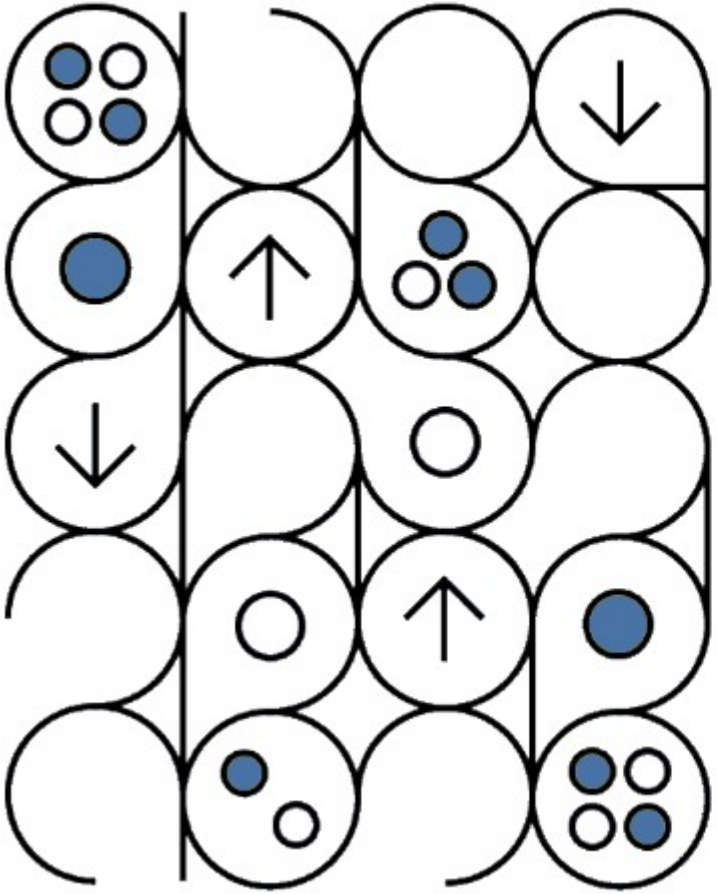


# NextGraph .org

Peer to Peer
Live collaboration
Semantic Web
CRDTs

# Fasten your seatbelt 🚀

- Semantic Web introduction
- Novel CRDT of RDF
- Document model
- End-to-end encryption
- Cryptographic capabilities
- Sync protocol
- Synchronous transactions
- Framework

# Semantic Web - Linked Data

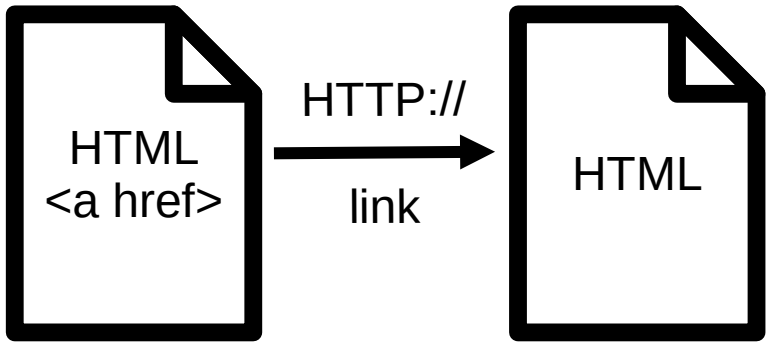
- Tim Berners-Lee :
  - 1999: Semantic Web, 2006: Linked Data, 2007: Giant Global Graph

# Semantic Web - Linked Data

- Tim Berners-Lee :
  - 1999: Semantic Web, 2006: Linked Data, 2007: Giant Global Graph
- We had the Web of pages

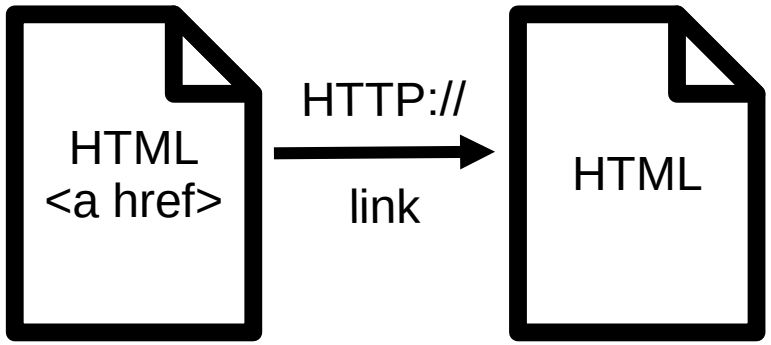
# Semantic Web - Linked Data

- Tim Berners-Lee :
  - 1999: Semantic Web, 2006: Linked Data, 2007: Giant Global Graph
- We had the Web of pages



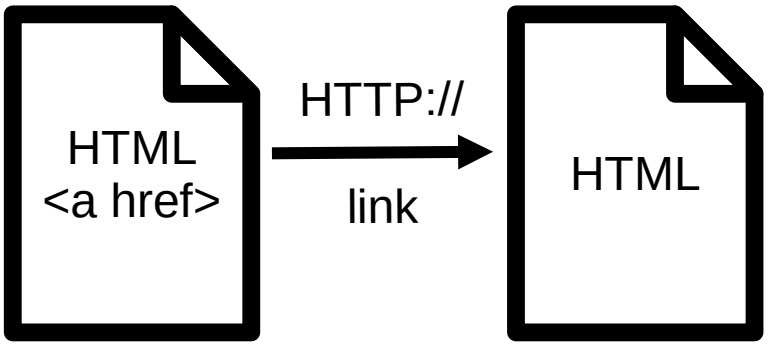
# Semantic Web - Linked Data

- Tim Berners-Lee :
  - 1999: Semantic Web, 2006: Linked Data, 2007: Giant Global Graph
- We had the Web of pages
  - Now we want the Web of Data

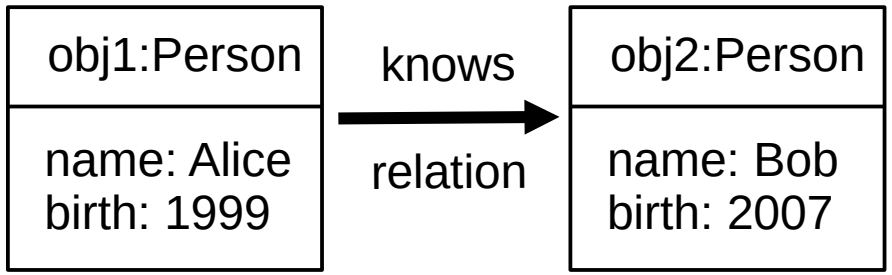


# Semantic Web - Linked Data

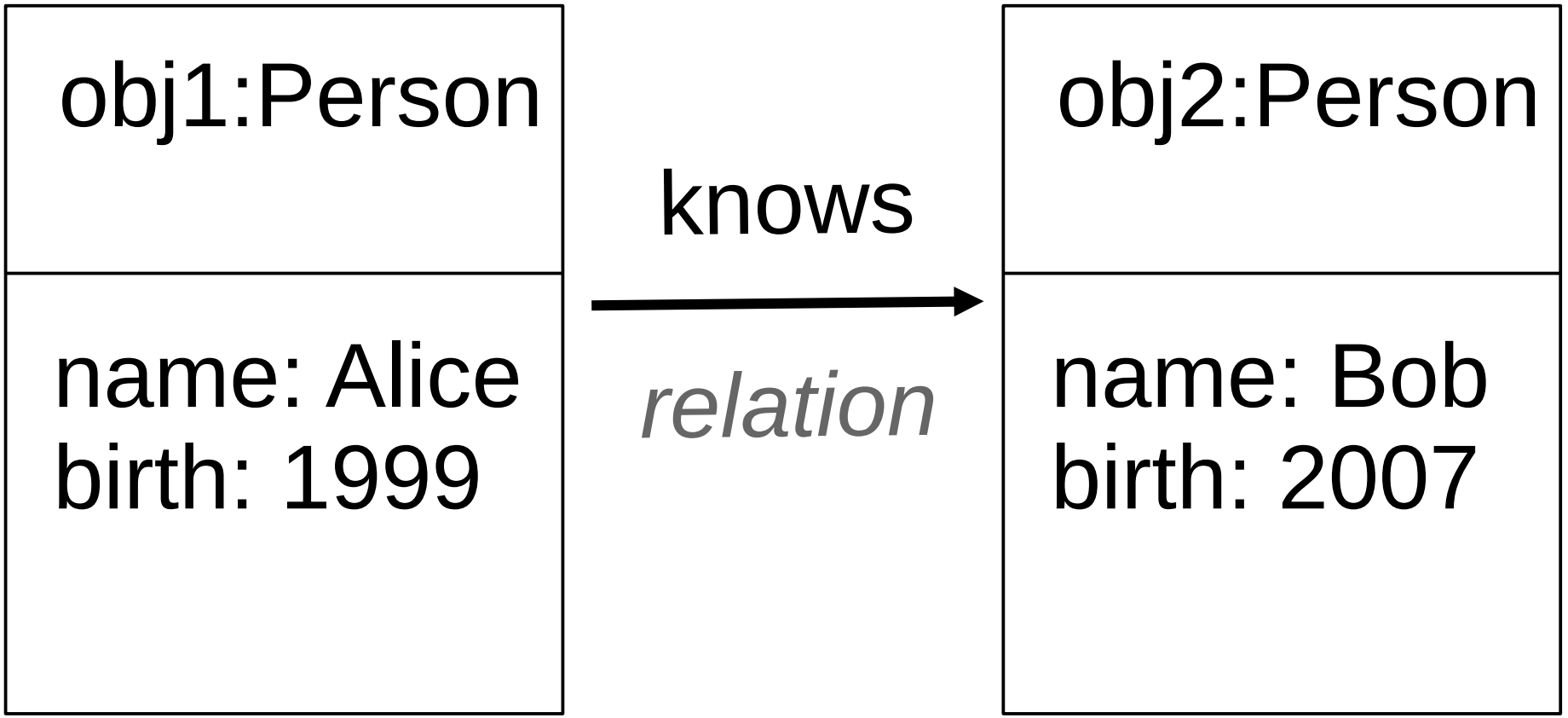
- Tim Berners-Lee :
  - 1999: Semantic Web, 2006: Linked Data, 2007: Giant Global Graph
- We had the Web of pages
  - Now we want the Web of Data




UML objects/classes



# Web of Data : UML



# RDF : Resource Description Framework

- exists since 1999. W3C standard since 2004
- used in Academia, Industry, WikiData
- simple format based on triples
- establish facts about resources

One triple =

Subject → Predicate → Object

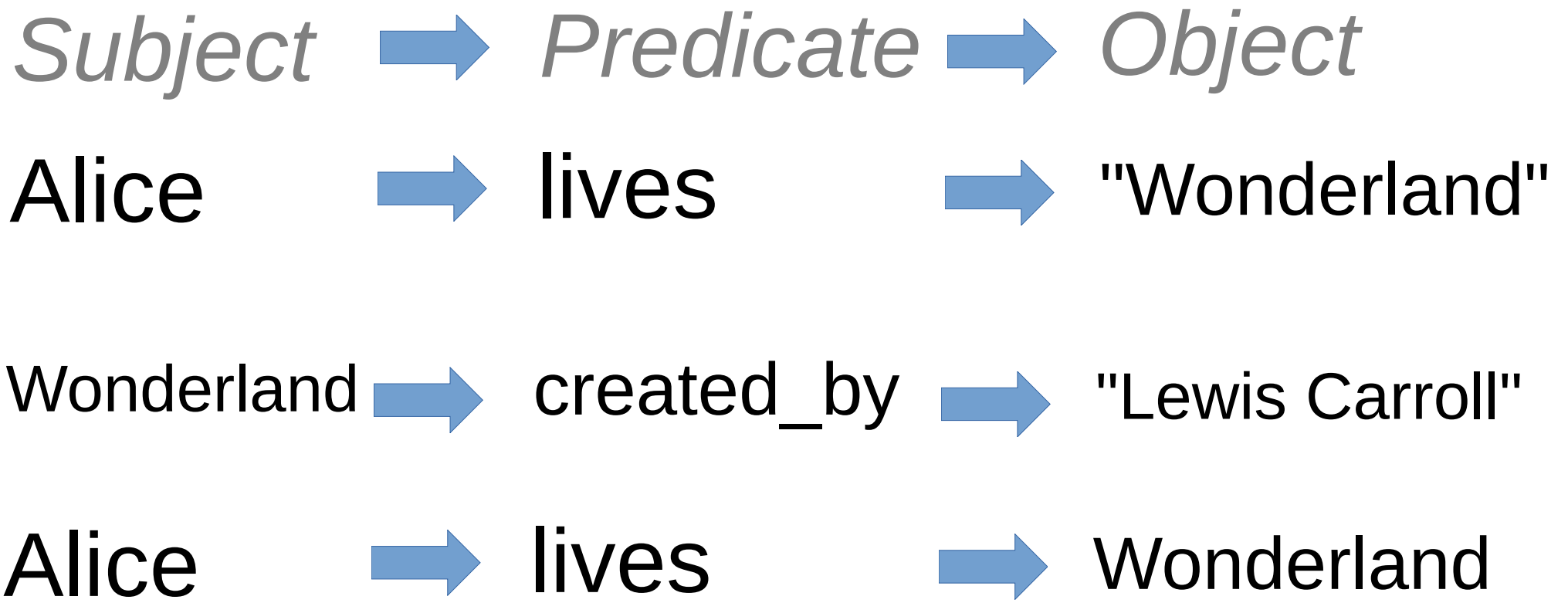
# Triple

| *Subject* | | *Predicate* | | *Object* |
|---|---|---|---|---|
| Alice | ➡ | lives | ➡ | "Wonderland" |

# Triple : from literal to resource

| *Subject* | *Predicate* | *Object* |
| --- | --- | --- |
| Alice | lives | "Wonderland" |
| Wonderland | created_by | "Lewis Carroll" |

# Object can be literal or ref to Resource



# Predicate and Relationship

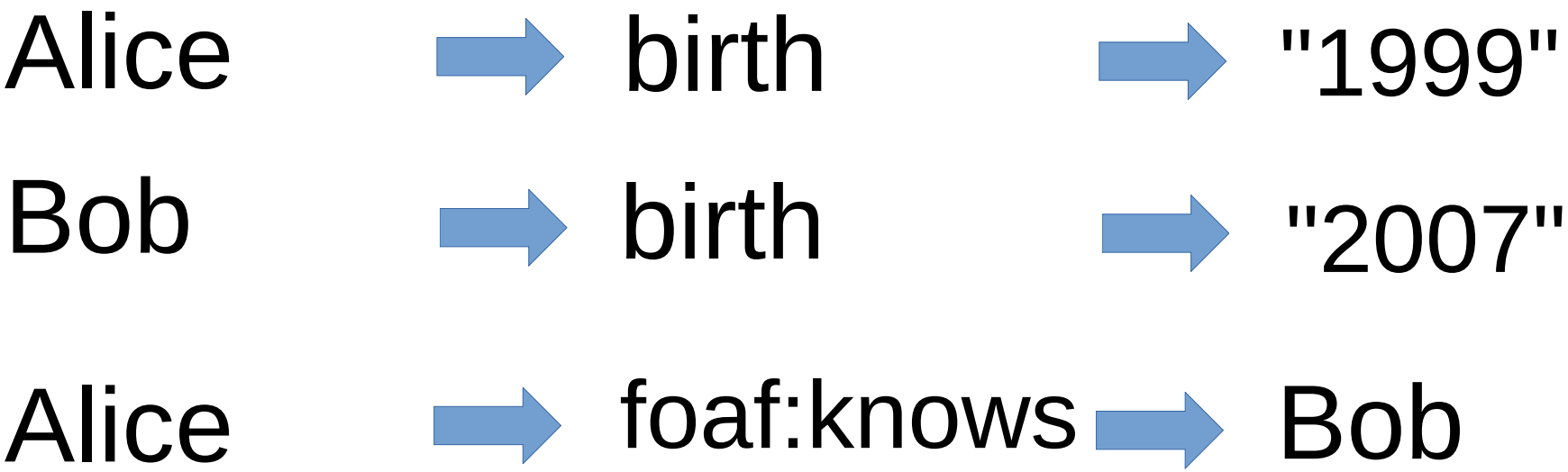
Wonderland → created_by → "Lewis Carroll"

Alice → lives → Wonderland

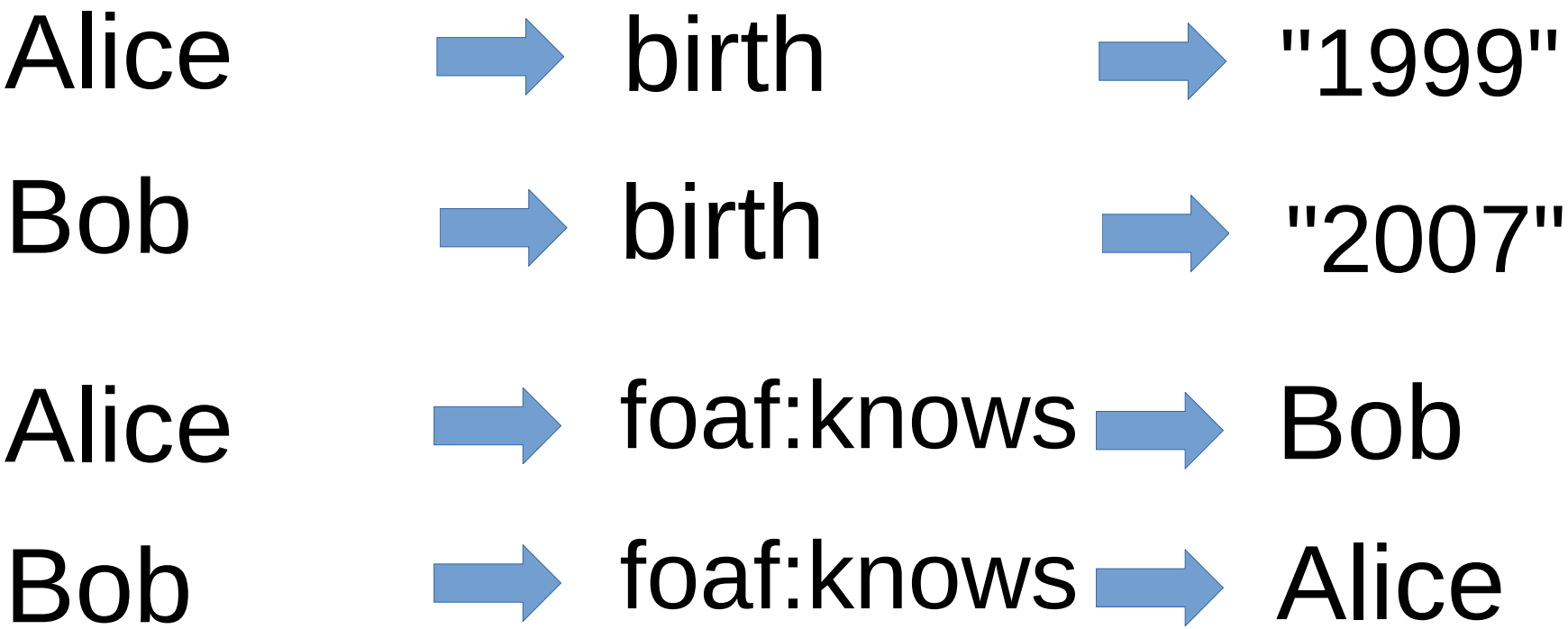
- Subject is always a resource ID (a URI) http://test.com/Alice
- **created_by** and **lives** are predicates
- equivalent to a property, attribute, or key in JSON / KV
- represented with a URI http://xmlns.com/foaf/0.1/**knows**
- shortened to foaf:**knows** if using prefix/context of Ontology

# relations are directed

# relations are directed



# relations are directed



# TripleStore

- the set of Triples is the database
- we call it TripleStore. several implementations JAVA, 1 Rust
- query language: SPARQL
- similar to SQL: filter with WHERE
- automatic JOINs : traverse the graph, match patterns
- no need for FOREIGN KEYS, normalization
- all the records are **JOINABLE** on all predicates by default
- URI for predicates: no need for migration. **Schema** embedded in the data

# SPARQL : Traverse the Graph

```
SELECT *
WHERE {
    ?person <foaf:knows> <http://test.com/Alice>.
    ?person <birth> “2007”
}
```

will return

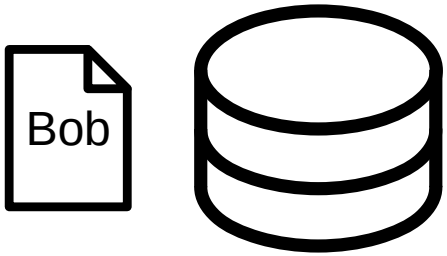
| person |
| --- |
| Bob |

# Web of Data 2.0

test.com


Alice -> birth -> “1999”

example.com

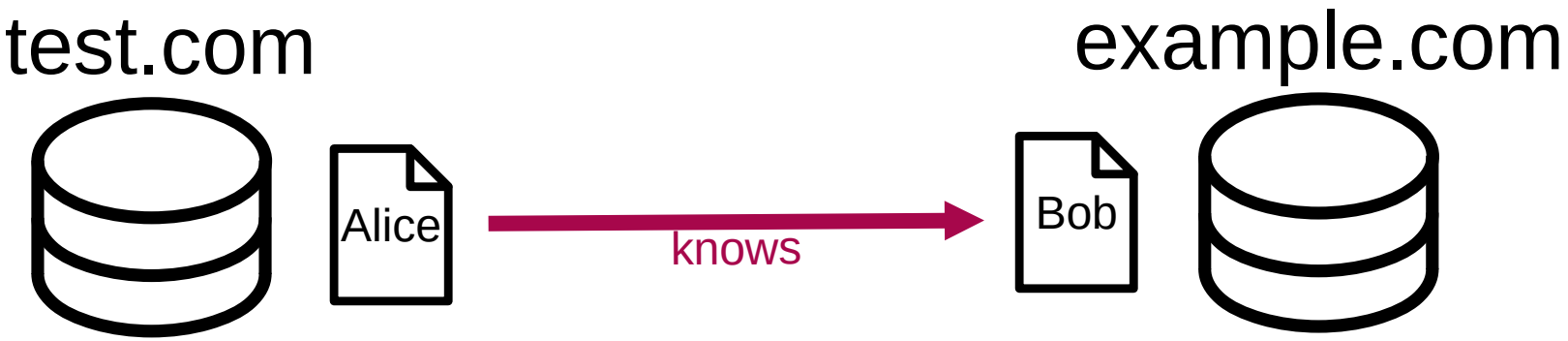


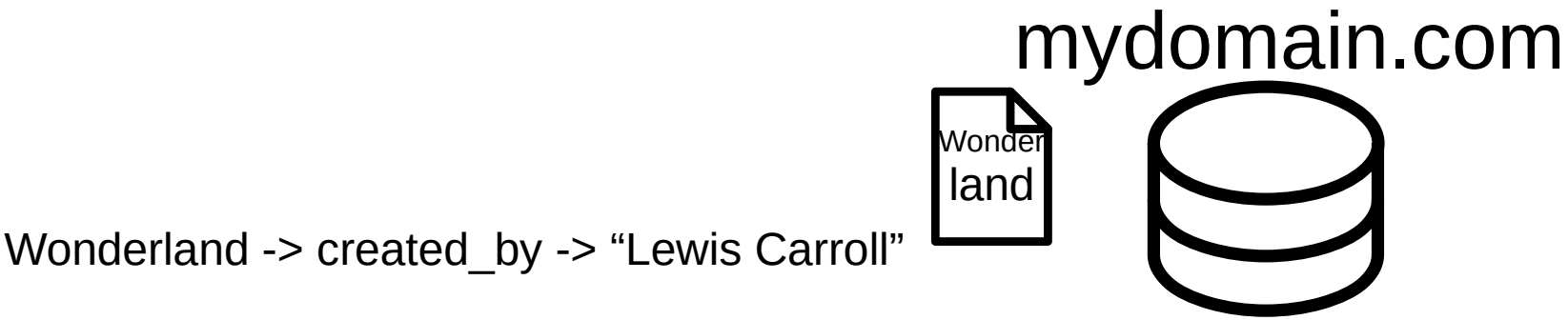

Bob -> birth -> “2007”

mydomain.com

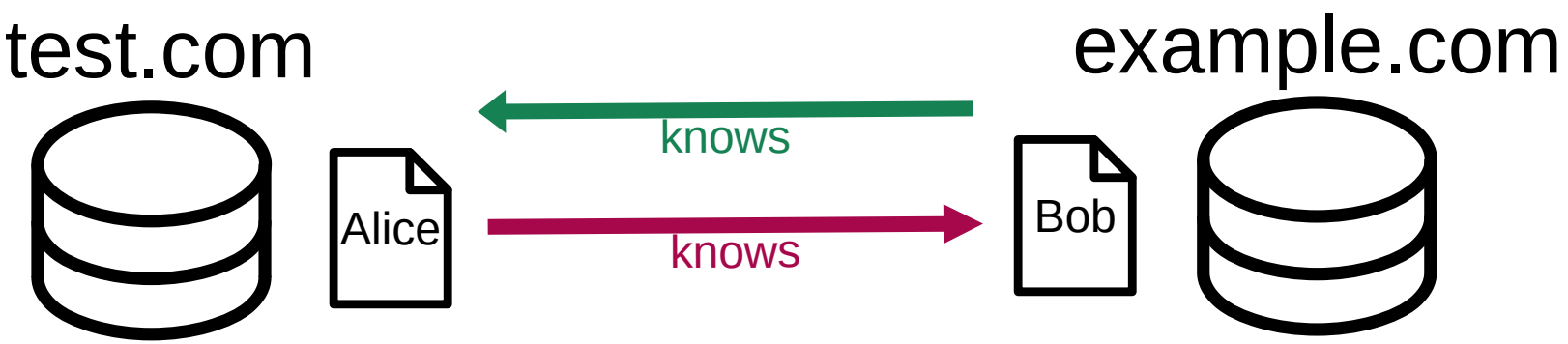

Wonderland -> created_by -> “Lewis Carroll”

# Web of Data 2.0



# Web of Data 2.0

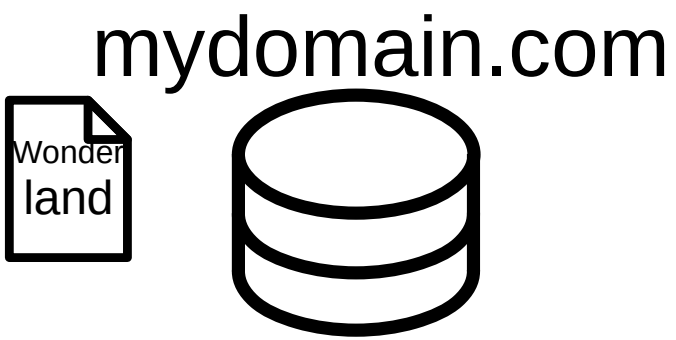



Alice -> birth -> “1999”

Alice -> foaf:knows -> http://example.com/Bob

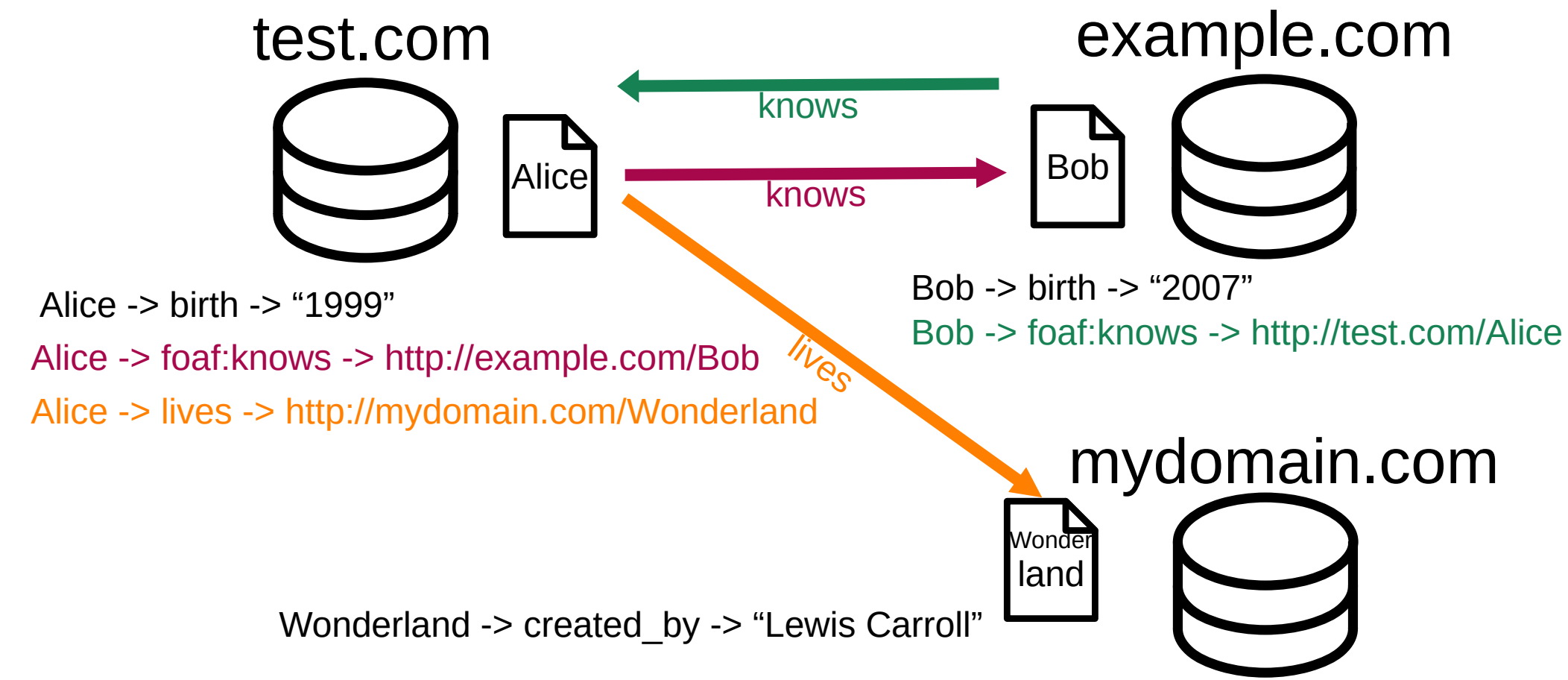
Bob -> birth -> “2007”

Bob -> foaf:knows -> http://test.com/Alice




Wonderland -> created_by -> “Lewis Carroll”

# Web of Data 2.0



# Web of Data: kind of...

- Links Entities' data across Files, Databases, Servers
- Solid project is one attempt
- There are several others (ActivityPods, AtomicData, etc...)
- But that's **not DECENTRALIZED** (DNS, HTTP)
- that's **not PORTABLE** data neither
- If I change the name of Alice's resource, the URL changes
- If I move my data to another domain, the URL changes
- 25 years later, we are still looking for our

**decentralized WEB OF DATA !!!**

# Web of Data 3.0

- NextGraph brings decentralization, CRDT and E2EE
- Important to stick to **RDF**, but **JSON** is also here to stay
- NextGraph forgoes of HTTP altogether, uses **DIDs** instead
- Decentralized Identifiers is a W3C standard that is seeing adoption (BlueSky, ActivityPub...)
- All RDF URIs become <did:...> using a randomly generated public key as **unique and permanent identifier**
- Self-Sovereign Identities : also used for user IDs

# DID URI

- http://test.com/Alice becomes

  did:ng:o:JQ5gCLoX_jalC9diTDCvx-Wu5ZQUcYWEE821nhVRMcEA
- **durable** and **portable**: the URI will never change, no matter what the triples are, or where the data is stored.
- the human-readable name of the resource is stored in a triple (as it also is in HTTP-based RDF)
- <did:ng:o:...> <rdfs:label> “Alice”
- <did:ng:o:...> <birth> “1999”
- <did:ng:o:...> <foaf:knows> <did:ng:o:TheLongIdOfBob>

# CRDT of RDF

- NextGraph is Local-First and CRDT based
- Each replica has locally access to a SPARQL endpoint
- UPDATING the graph is done with SPARQL UPDATE, like usual

```
INSERT DATA {
  <> <https://schema.org/name> "Wonderland".
  <> <https://schema.org/description> "A fictional country".
}
```

# UPDATE RDF

- replacing the value of a predicate is done with a DELETE+INSERT operation
- This represents one transaction (called a commit)

```
DELETE DATA {
  <> <https://schema.org/description> "A fictional country".
};
INSERT DATA {
  <> <https://schema.org/description> "A fictional place".
}
```

# Concurrent Updates

- Because NextGraph has implemented its own CRDT of RDF (based on the excellent SPARQL engine of Oxigraph), concurrent updates on the same resource **never conflict**.
- We use the Observed-remove SET mechanism as described in the SU-SET paper by Luis Daniel Ibáñez et al.
- If two concurrent DELETE and INSERT commits happen on the same triple, the INSERT wins.
- Access the **history**: We can query each revision separately to go back in time, by using the Commit ID.

# CRDTs agnostic

- A Document in NextGraph is primarily an RDF resource
- RDF is used to link Documents between each other
- SPARQL is used to traverse the Graph, cross-replica.
- Documents can also have some additional “side-car” data in the form of JSON. We call this the “**discrete**” part.
- It is optional and is based on **Yjs** or **Automerge** libraries.
- More CRDT could be added (Loro, Diamond types, etc...)
- NextGraph protocol can transport any kind of CRDT

# Document: graph, discrete, binary

- The discrete part (JSON/XML) is useful for some data usage (key/value, rich text documents, list/arrays) and complements well the RDF part.
- For now only the RDF part can be queried with SPARQL, but we have plans to extend it to the JSON part.
- GraphQL will also be made available, on top of SPARQL
- In addition, a document can have an arbitrary number of **attached binary files**

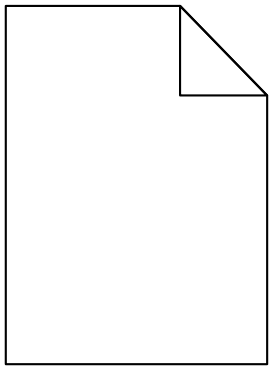
# NextGraph Document: Rich Text

NURI
did:ng:o:xxx

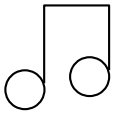
binary files

graph

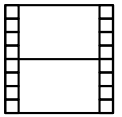
```
<did:ng:o:xxx> <rdfs:label> “my first post”.
<did:ng:o:xxx> <as:image> <did:ng:j:yyy>.
<did:ng:o:xxx> <as:summary> “my first
   thoughts on something futile”.
<did:ng:o:xxx> <dcterms:created>
  "2015-01-01T00:00:59Z".
```



<did:ng:j:yyy>



<did:ng:j:zzz>



<did:ng:j:www>

discrete: Yjs markdown

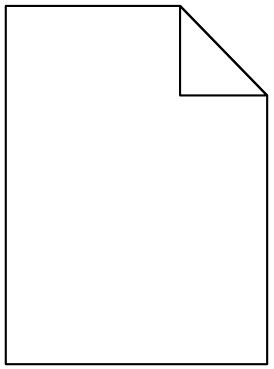
```
## My important title
- a list item in markdown
**some bold text**
![image](did:ng:j:yyy)
```

# NextGraph Document: JSON

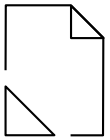
## graph

```
<did:ng:o:xxx> <rdfs:label> “some data”.
<did:ng:o:xxx> <prov:wasDerivedFrom>
                        <did:ng:j:yyy>.
<did:ng:o:xxx> <as:summary> “some
    very important data derived from CSV”.
<did:ng:o:xxx> <dcterms:created>
  "2015-01-01T00:00:59Z".
```

## NURI
## did:ng:o:xxx



## binary file



<did:ng:j:yyy>

by example containing the CSV source imported into the JSON

## discrete: Automerge JSON

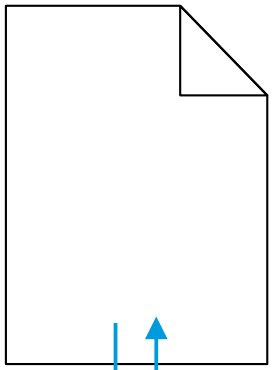
```
{
      ”key”: “value”,
      ”list” : [ 1, 2, 3]
}
```

# 2 NextGraph Documents: Social Network

## Profile of Alice
## did:ng:o:xxx

### graph

```
<did:ng:o:xxx> <rdfs:label> “Alice”.
<did:ng:o:xxx> <vcard:photo> <did:ng:j:yyy>.
<did:ng:o:xxx> <foaf:knows>
  <did:ng:o:aaa>
```



### binary file

<did:ng:j:yyy>

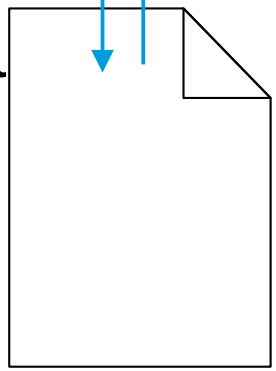
### discrete: Yjs Markdown (bio)

Hi I am Alice, a well known computer **geek**

## Profile of Bob
## did:ng:o:aaa

### graph

```
<did:ng:o:aaa> <rdfs:label> “Bob”.
<did:ng:o:aaa> <vcard:photo> <did:ng:j:zzz>.
<did:ng:o:aaa> <foaf:knows>
  <did:ng:o:xxx>
```



### binary file

<did:ng:j:zzz>

### discrete: Yjs Markdown (bio)

Hi I am Bob, the classical buddy of Alice. I dislike Eve who is nosy

# Schema & Primary Class

- RDF lets you specify for each resource, one or more classes (data types) with the predicate rdf:type
- Reusing predicates and classes (defined in **ontologies**) across data models fosters **interoperability**. There exist already a multitude of ontologies defining the most common predicates and classes.
- Each NextGraph Document needs to have at least one class called “primary class” that will be used to know which **apps** can open the document.

# NG Apps: Viewers and Editors

- Developers can create Apps in NextGraph by defining a set of **Viewers** and **Editors** that can open a list of Primary Classes.
- NextGraph comes with a set of “official” apps that deal with the most common Document types and use-cases.
- The user can switch and use other apps as needed
- Apps can run inside NextGraph or **standalone**.
- The framework for App developers will be ready within 2 months (Svelte, React, VueJS, nodeJS, Deno, Rust)

# Example of Official Apps

- Post / Article in Markdown or plain text
- Data editor (JSON, XML, Table)
- Code editor (Rust, JS, TS, Svelte, React, etc...)
- Email
- Diagram, Chart, Visualisation
- PDF, video viewer, audio player
- Contact, Event, Calendar, Scheduler, Chatroom
- Form, Spreadsheet, Slides, Poll, Task, Project

# More apps :)

- More apps can be created on top of NextGraph
- Notion alternative
- Gmail copycat
- Instagram copycat
- Personal Knowledge/Graph Management
- productivity suite
- **Services**: all the data is accessible from nodejs/Deno

# Permissions & Capabilities

- NextGraph is private by default.
- User gets 3 **stores** by default: Public, Protected, Private
- Can create more stores: Groups and Organizations
- All data is encrypted at **rest** (convergent encryption for DAG, and AES for materialized state)
- And in **transit** (Noise protocol inside websocket)
- Permission's granularity: Document
- Permission **inheritance**: by Store, and transitive permissions when capability inserted into document

# End to End Encryption

- NextGraph is decentralized. No SPOF. Replication of data
- NextGraph is **almost P2P** ;) Async interactions between peers that not always online
- Need for **Brokers** and **2-tier** network topology, inspired from LoFi.Re design by T.G. of P2Pcollab
- Brokers are blind to the data they exchange. E2EE
- They only deal with encrypted blocks and maintaining the Pub/Sub
- Can be **self-hosted**, and we will encourage this
- No need for domain nor TLS cert. Works on bare IP

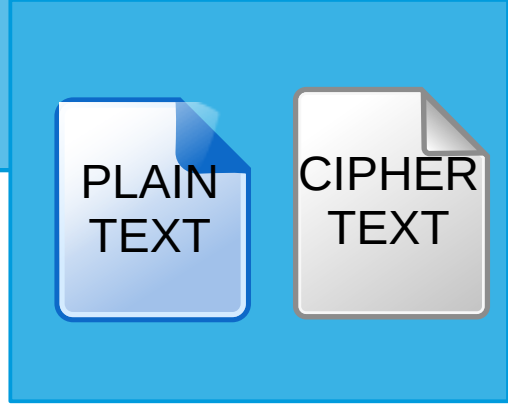
# Convergent Encryption

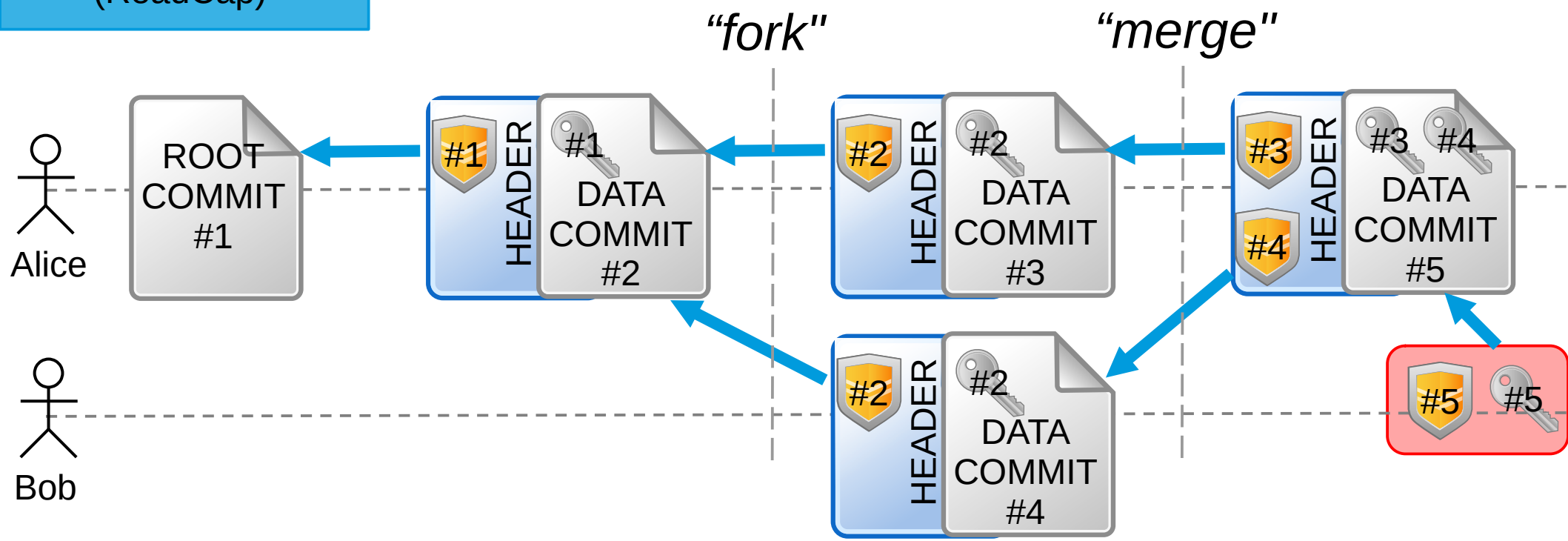
- Commits and Binary data are chunked into fixed size blocks
- Block’s plaintext content is hashed (Blake3). Hash becomes the encryption symkey. (confirmation of a file attack: keyed hashing using the inner overlay read cap)
- content is encrypted with symkey. ChaCha20
- hash of the ciphertext becomes the blockID
- the pair (blockID,symkey) is the Read Capability of the block, also called an ObjectReference.

# Encrypted DAG of commits

- For data that is bigger than one block size (1MB), a Merkle tree is formed to represent all the blocks.
- The DAG (directed acyclic graph) that contains all the CRDT operations (commits) of a document, is E2EE. Each commit references its causal past (one or more commits that happened before). The references are split in two. the IDs are not encrypted, so the brokers can read them and reconstruct the DAG. But the symkeys are encrypted, so only those who possess the symkey of the current HEADs, can recursively decrypt the whole DAG.

# Encrypted DAG

# Branches => blocks & partial sync

- Each Document is composed of several branches (DAGs)
  - A root branch containing the permissions and admin details
  - A main branch containing the main content of doc
  - Any additional branch created by forking a branch or in order to store a standalone **block** => a block as in Notion, holding partial content, with its own primary class and CRDT. => **Partial Sync!**
- The PUB/SUB is organized in Topics. Each branch maps to a unique Topic. Brokers only see TopicIDs, not branchIDs.
- Branch ReadCap : gives read access to a topic.
- TopicID is refreshed when ReadCap must be revoked

# 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 causal broadcast. Thanks to LoCaPs paper of Filipa Salema et al.
- Brokers maintain connectivity between each other and reroute paths if needed, in a general undirected graph network topology. They maintain the routing tables.
- Fallback to P2P reconciliation directly between replicas is possible if needed

# Write Capability

- Publishers (editors) permission is encoded in 2 complementary ways:
  - the repository (root branch of a document) maintains the list of public keys of authorized peers with write permission. As each commit is signed by its author, this allows each replica to verify independently that the edit is authorized.
  - in order to publish events in a topic, the editor needs the private key of the topic in order to sign the event. Brokers only accept Events that are correctly signed. This shared private key is rotated when needed (revocation or compromise).
- Uses Curve25519 (Edwards and Montgomery form for signatures and key exchange)

# Pub/Sub Topic & Events

- 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 them 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 replica is online.

# NURI and sharing capabilities

- Capabilities are of a new kind, that we call “cryptographic capabilities”. They contain a symkey (for ReadCap) or a private key (for WriteCap). Possession of the capability is all that is needed in order to invoke the capability. There are no controllers.
- Capabilities are encoded as NURI (DID based NextGraph URIs) and can be included inside a document, making them transitive.
- They are transient. If not used before a refresh occurs, they become unusable. PermaCap are the solution for durable capabilities (similar to z-caps).

# Pull Control

- Having access to encrypted events is a risk. We limit to the minimum the capability to obtain encrypted blocks from brokers.
- This is especially true because of convergent encryption
- 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.
- Capability for access to both is refreshed when needed

# 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) which is a public key, and following a chain of **certificates**

# Finality & Total order

- **Eventual consistency** is great for Document-oriented data, aiming at local-first collaboration.
- It is less interesting for use-cases like accounting, avoiding **double spend**, **ACID** properties of transactions, and all cases when the **finality** of transaction must be assured
- Based on the threshold signature mechanism, we offer the option to mark some segment of a document as “synchronous”. Those will only be modified after a group signature is obtained, enforcing finality, immediate consistency and **total order** on this segment of data

# Synchronous Transactions

- Synchronous transactions can be used to implement **Smart Contracts**, cross-document transactions, **atomic transactions**, decentralized naming system, and **e-commerce** applications
- Quorum is configurable and different setups are possible depending on your application's needs (unique signer, 50%, up to 100%)
- We believe synchronous transactions are an important feature that complements well the CRDT/eventual consistent model. Each document can use both models.

# Framework

- Developing applications in NextGraph will be possible within 2 months, once the Framework will be released
- Frontend API will be based on **reactive stores** of **React, VueJS and Svelte**, for both read and write access. With also access via SPARQL local endpoint (read and write)
- Backend access to data, according to permissions given by user, in **nodeJS and Deno**, with reactive data store too, probably based on Valtio.
- APIs also available in Rust, and for Tauri framework
- **CLI** to access data. NPM package for **headless** CMS (Astro, ...)

# Current Status

- App for Linux, macOS, windows, android (and soon iOS) and web is released in alpha version since 2 months already. You can try it.
- Very limited features for now in the apps. Hints at what the features will be in the beta release (beginning of next year)
- Framework will be ready before the end of 2024
- **Open platform, open sync protocol, open ecosystem**
- permissive licensing MIT/Apache2. **Free and always will be.**
- Several **collaboration** with other projects in negotiation!
- ActivityPods collab for Solid & ActivityPub compatibility

# Goals

- **Homomorphic encryption** of SPARQL queries in order to complete our full privacy preservation goal, even in a social network environment with a lot of shared personal data.
- Develop “**good default**” apps for basic needs of end-users in terms of productivity suite, daily use, social network, collaborative tools, etc...
- towards a **unified UX** and highly integrated and interoperable system, forming a “best of its kind” suit of products that can **compete with Big Tech** captive ones
- **regain total control over data & software ownership**

# Demo

Demo time 

(2 mins)

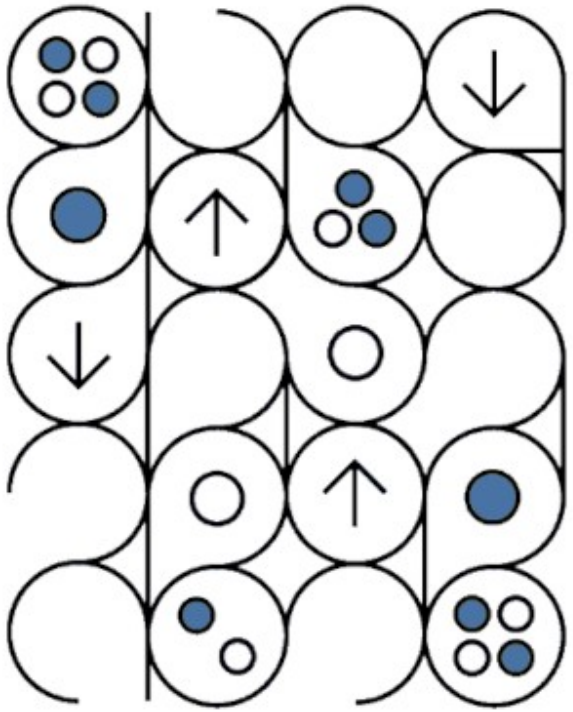
# Join the community

- Thank you for your time ❤️
- Join the community in our forum https://forum.nextgraph.org
- Subscribe to our newsletter
- Spread the word on Mastodon or X
- Contribute or donate ;)  give us a star on GitHub!
- A big thank you to NLnet foundation and the NGI Assure fund who made this project possible !



**NextGraph.org**

# Stay tuned for more!



open source

Free and always will be

NextGraph.org

## A new platform for a better internet

Decentralized - Secure - Private - Encrypted

with Social Network
Shared Documents
& Productivity Tools

APP