Solid: The web the way it was envisioned

World before the web

- Hard to exchange info
  - How/Where can we find other people
- Hard to build
  - Different Software, Different Hardware
- Hard to innovate
  - Distribution Problem. How can software be delivered to people

Creation of the Web

- Universality
- Permissionless Innovation

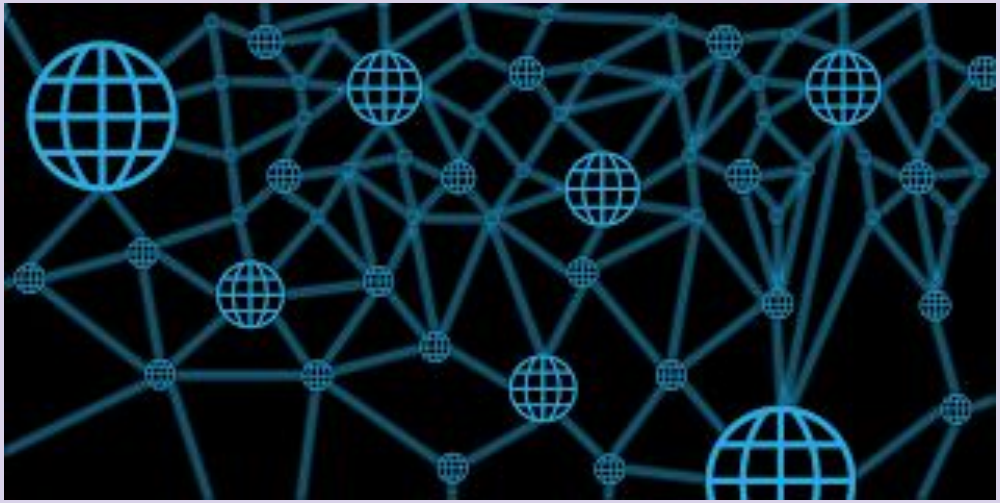
The Web brings: Universality

- Freedom to use
  - Low barrier of entry
  - Everyone can have a blog/site/space
- Interoperable
  - Independent of hardware
  - W3C standards (the paradox of freedom!)
- Interconnected through **links**
  - Share on your own terms

The Web brings: Permissionless Innovation

- Freedom to create
  - “*Anyone can build anything for any reason*”
  - Build for the web (not an OS, or an architecture)
- No permissions needed
  - Anyone can join the Web and build
  - *Is this the case for app stores?*
- We can **link** to other people’s software/ideas

The history of the web is decentralized



The slow Centralization of the Web

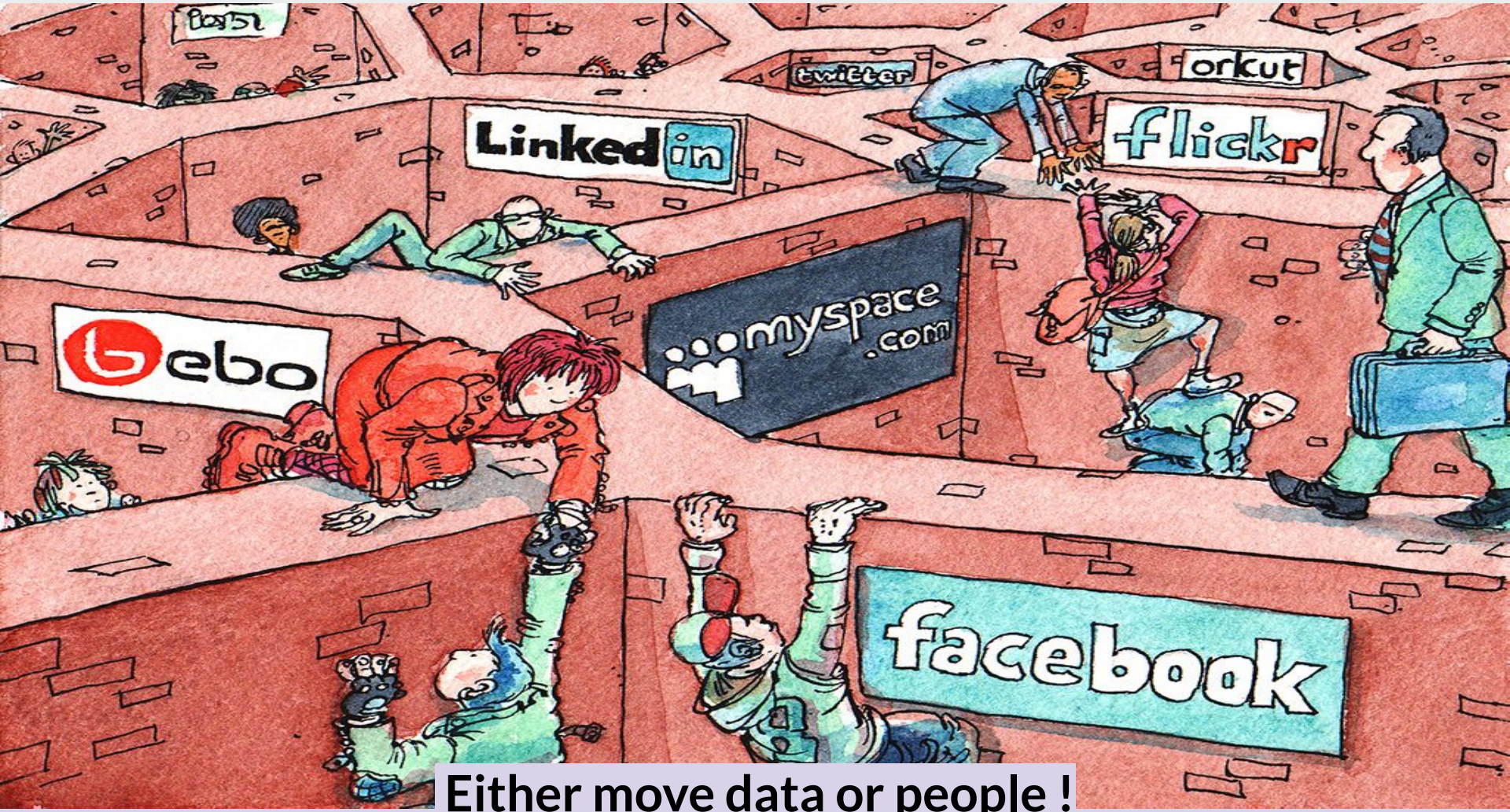
- Browser dependence
  - IE -> Google Chrome. Build for a specific browser
- Search Engine dependence
  - SEO, page ranking instead of content quality
  - Pay Google to be seen
- Social Platform dependence
  - Be on Facebook to have an identity, to see your friends
  - Gather likes to have business
- Computing dependence
  - Amazon, Azure, GCloud for infrastructure

The centralization of Data by big platforms

- “Cheap” and easy way to create
  - Blogs are now on Facebook, code on Github, work profiles on LinkedIn etc
- Identity dependence
  - Sign In with Google/Facebook
- Ongoing war for our data

The centralization of Data by big platforms

- Sharing is hard or dependent
  - Data are tied to each provider/app
- Innovation is hard or dependent
  - You either depend on some centralized platform or you try to become one!
- Lost control, privacy
- Walled Garden effect



Either move data or people !

Three challenges for the Web

1. Regain control over our personal data
2. Reduce the spread of misinformation
3. Transparency of (political) advertising

Tim Berners Lee

The Solid Project

Solid is an effort to re-decentralize the Web

- Exists as an idea for many years
- By a team led by Tim Berners Lee
  - Solid MIT Project https://solid.mit.edu/
  - Inrupt Startup and community https://solid.inrupt.com/
- **Ecosystem - Movement - Community**

Timeline

- **2014**: Start of Solid at MIT
- **Oct 2018**: Launch of Inrupt
- **Jan 2019:** developer toolkit and common UX
- **In 2019:** MVP of the ecosystem

Solid is about choice

- Provide a way to build web apps that allows users to control their data
- Separate data and apps
- Built on existing standards
  - **Content**: Linked Data
  - **Id and Authentication**: WebID, WebID-TLS
  - **Access Control:** Web Access Control
  - **Read/Write:**HTTP REST and Websockets API

Users choose where to store Data

Author's name and latest profile picture
stored in author's personal data pod

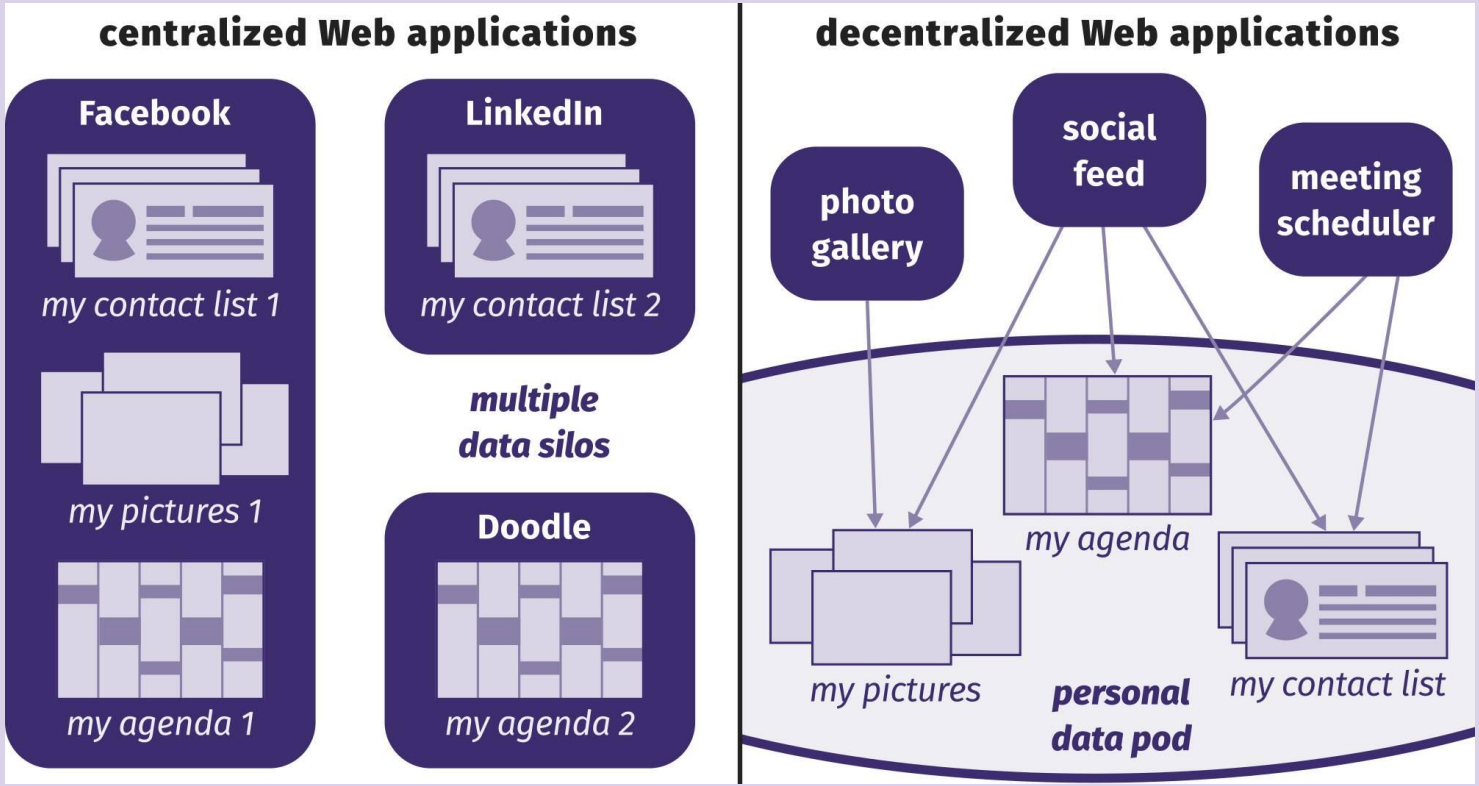
Work-related opinion about an article
stored in data pod of author's company

Discussed article title and photo
stored in news website's data pod

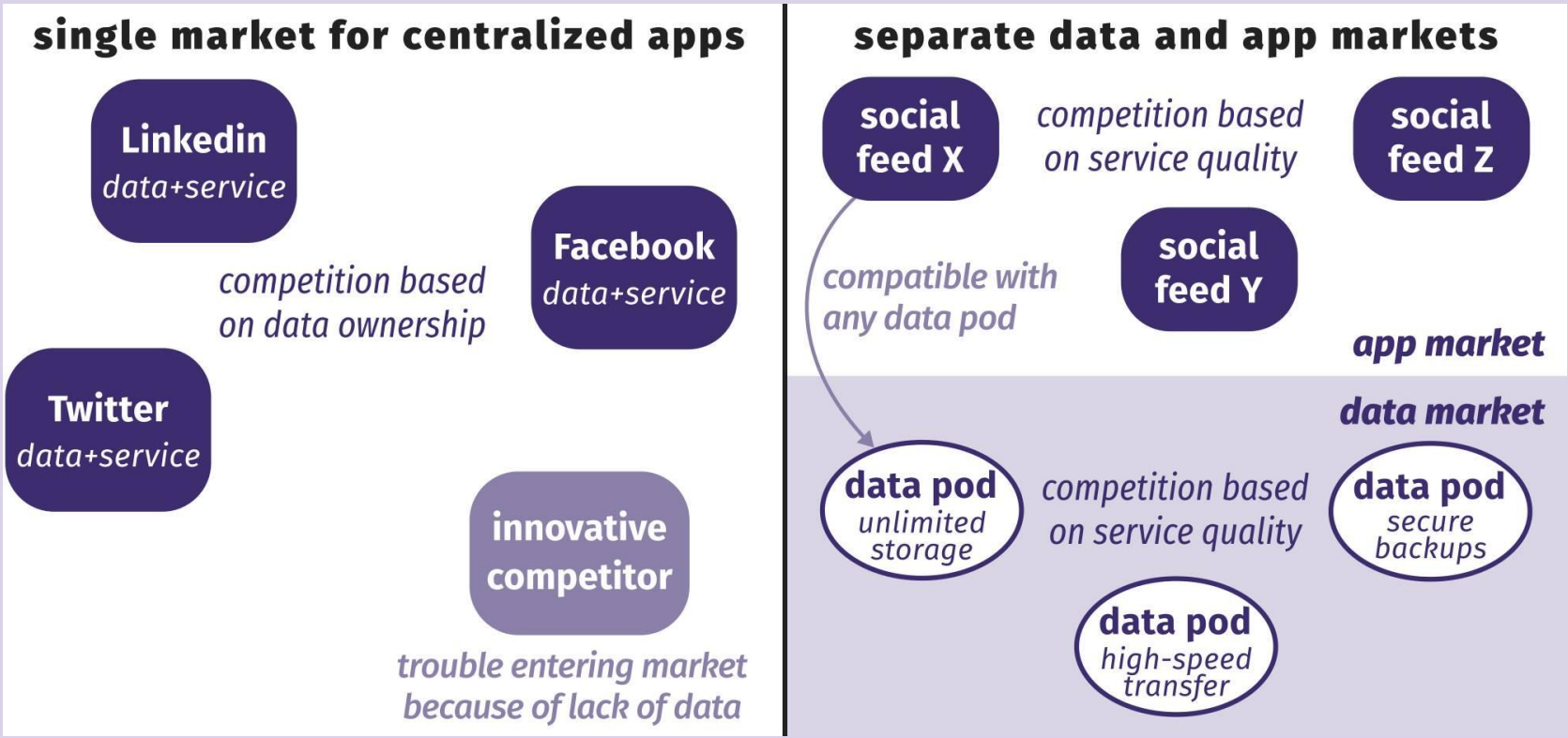
Likes on this post
each one in different individuals' data pods

Comments on this post
each one in different individuals' data pods

Users control access to their Data



A new wave of permissionless innovation



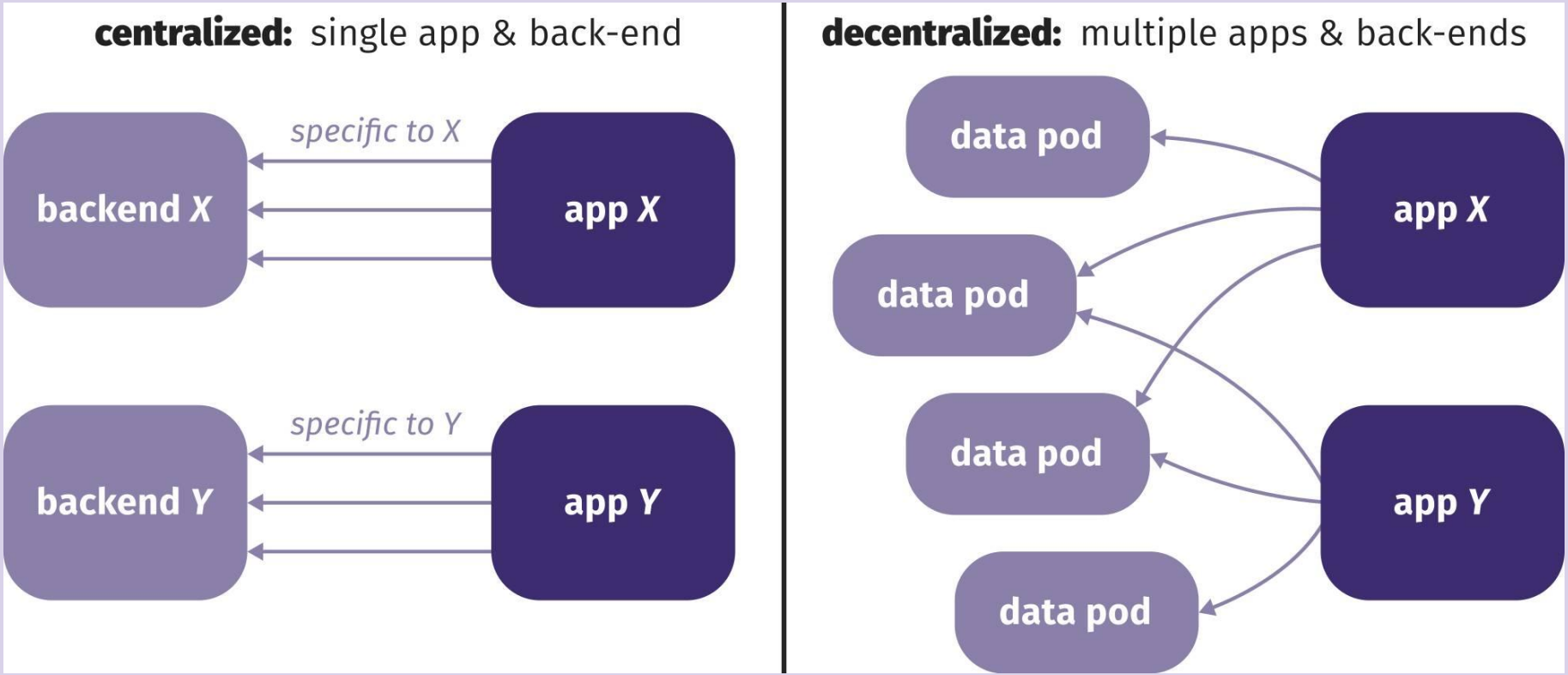
A Solid server is a Data Pod

- A regular web server
  - Support for Access Control
  - Support for Linked Data
- Application agnostic
  - Exposes an interface
  - App-specific logic on clients
- A Solid Pod is essentially a website
  - With data
  - That can be opened by any app

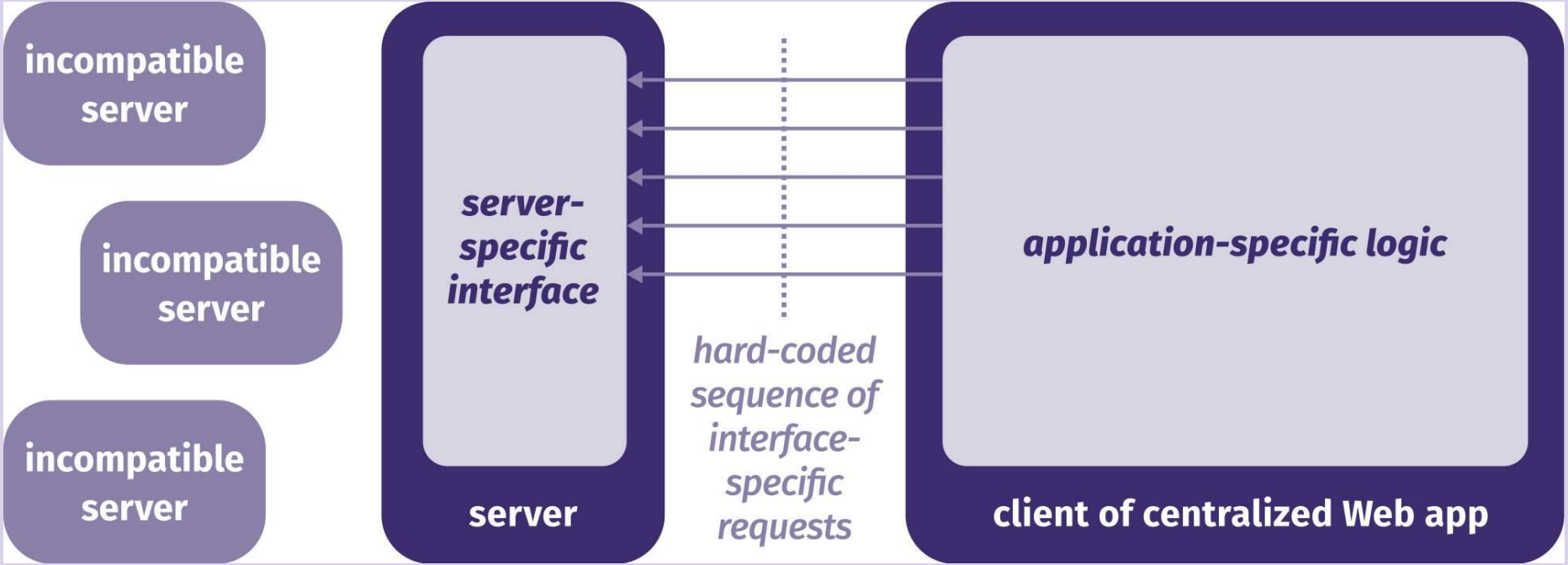
Solid clients are apps

- Browser or Native Apps
  - That read from and write to data pods
- Users give precise access to their data
  - Choose what to share
- Unified experience
  - I.e. Browse your friends pictures together with yours

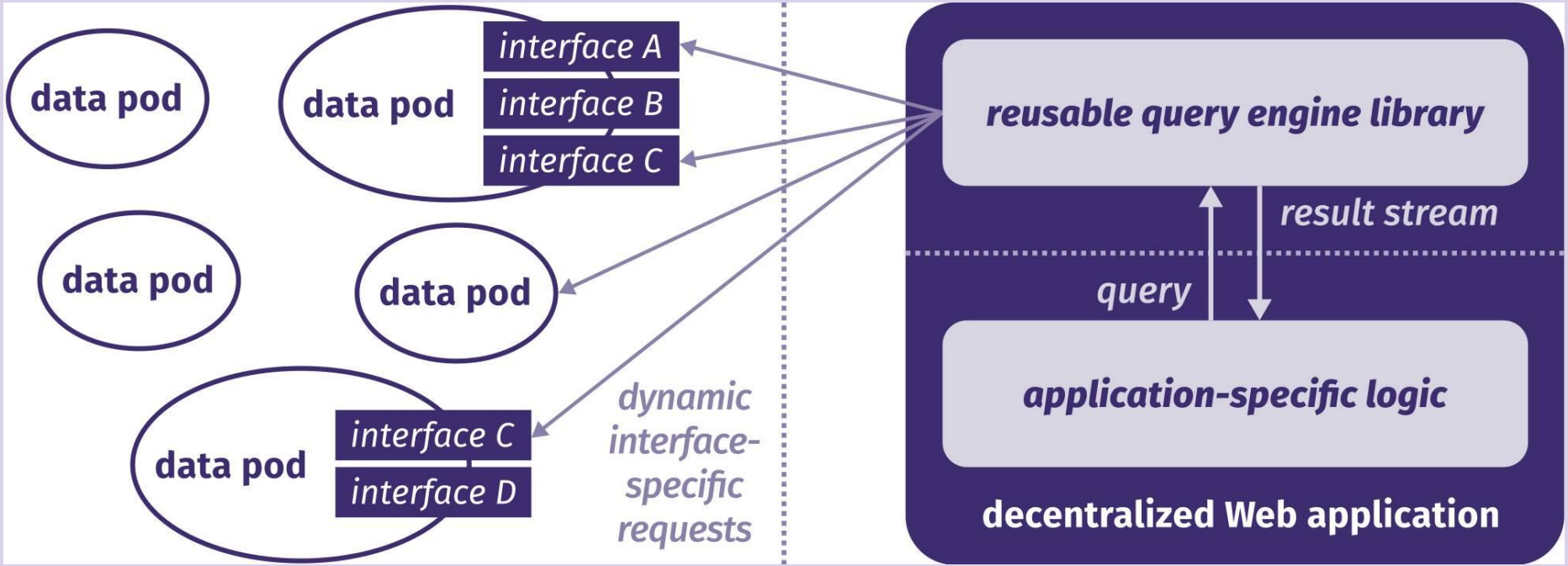
Many-to-many servers and clients



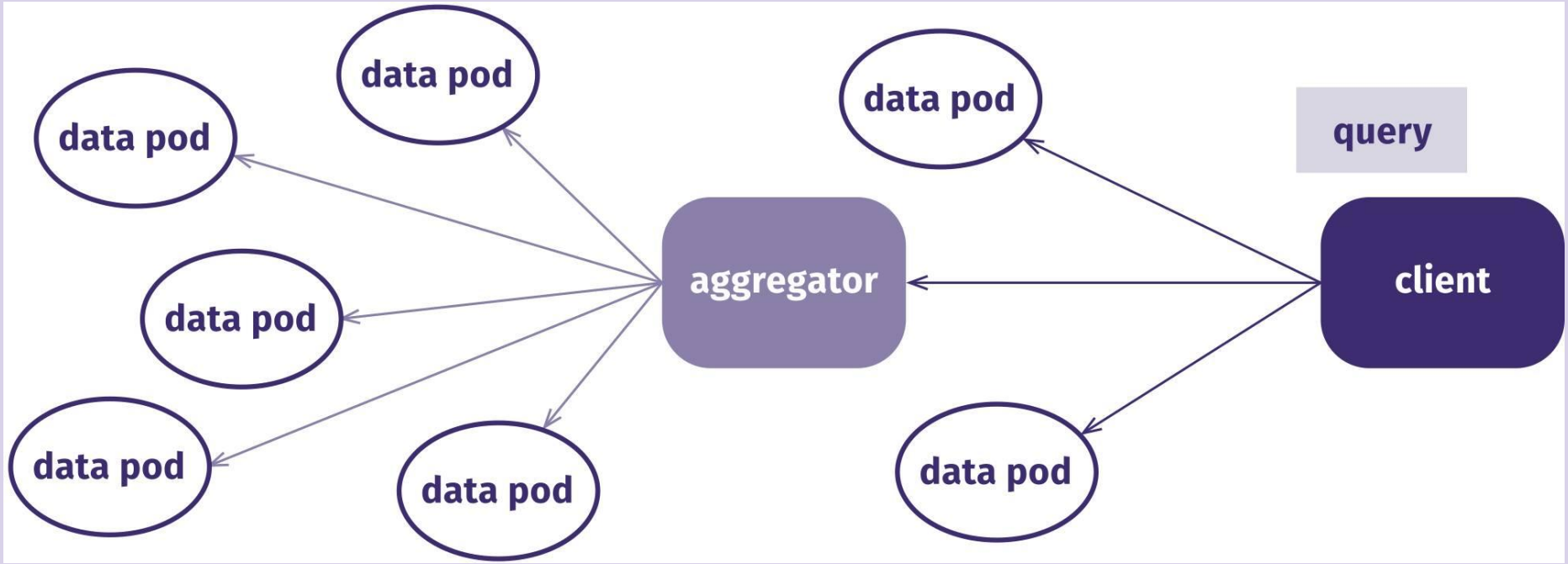
A new way to build APIs



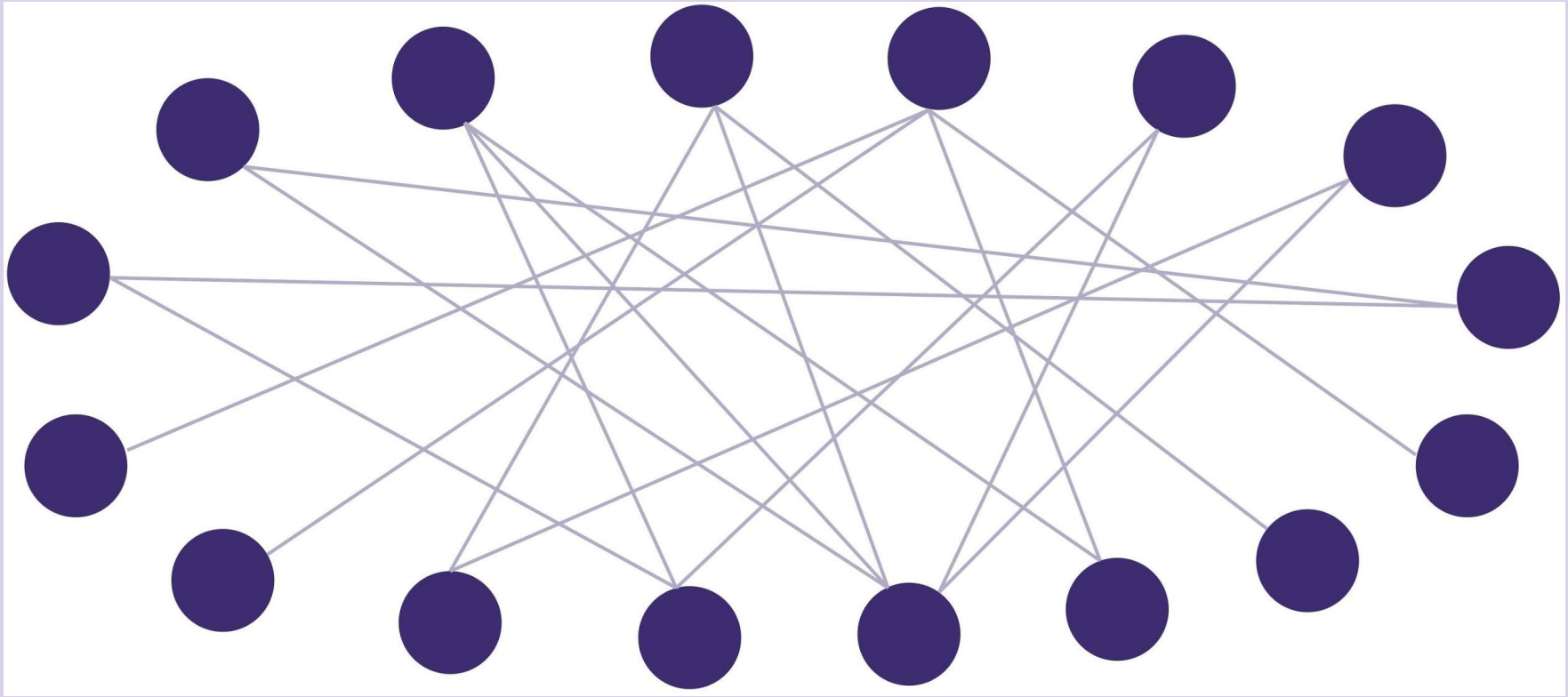
A new way to build APIs



Replication for performance



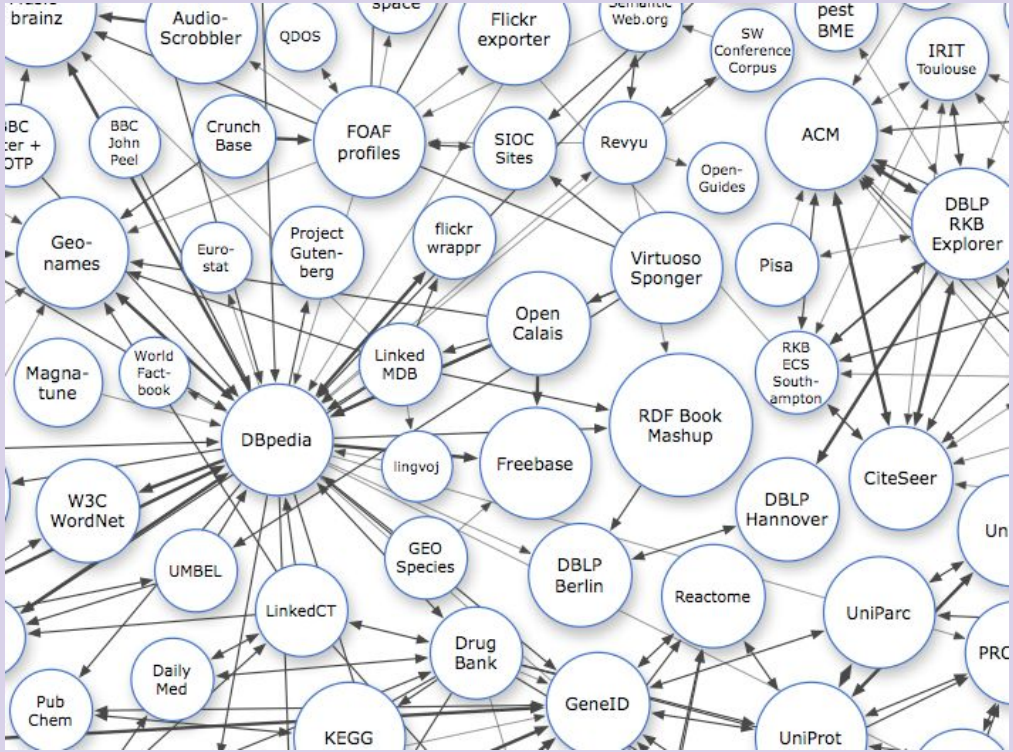
Aggregators that lead to a fully decentralized network



Interoperability?

- How do we **connect** each others data
  - Since they are kept in our pods
- How can apps **share** data
- How do we **integrate** data
  - From multiple pods

Linked Data


brainz
Audio-Scrobbler
QDOS
Flickr exporter
Web.org
SW Conference Corpus
pest BME
IRIT Toulouse
BBC John Peel
Crunch Base
FOAF profiles
SIOC Sites
Revyu
ACM
Open-Guides
DBLP RKB Explorer
Geo-names
Euro-stat
Project Guten-berg
flickr wrappr
Virtuoso Sponger
Pisa
Open Calais
Linked MDB
Magna-tune
World Fact-book
RKB ECS South-ampton
DBpedia
lingvoj
Freebase
RDF Book Mashup
CiteSeer
W3C WordNet
DBLP Hannover
UMBEL
GEO Species
DBLP Berlin
Reactome
UniParc
LinkedCT
Drug Bank
Daily Med
Pub Chem
KEGG
GeneID
UniProt

4 principles to publish Linked Data

1. Use URIs as names for things
2. Use HTTP URIs so people can look up those things
3. When someone looks up a URI, provide useful information, using standards
4. Include links to other things, so people can discover more

Linked Data are already available

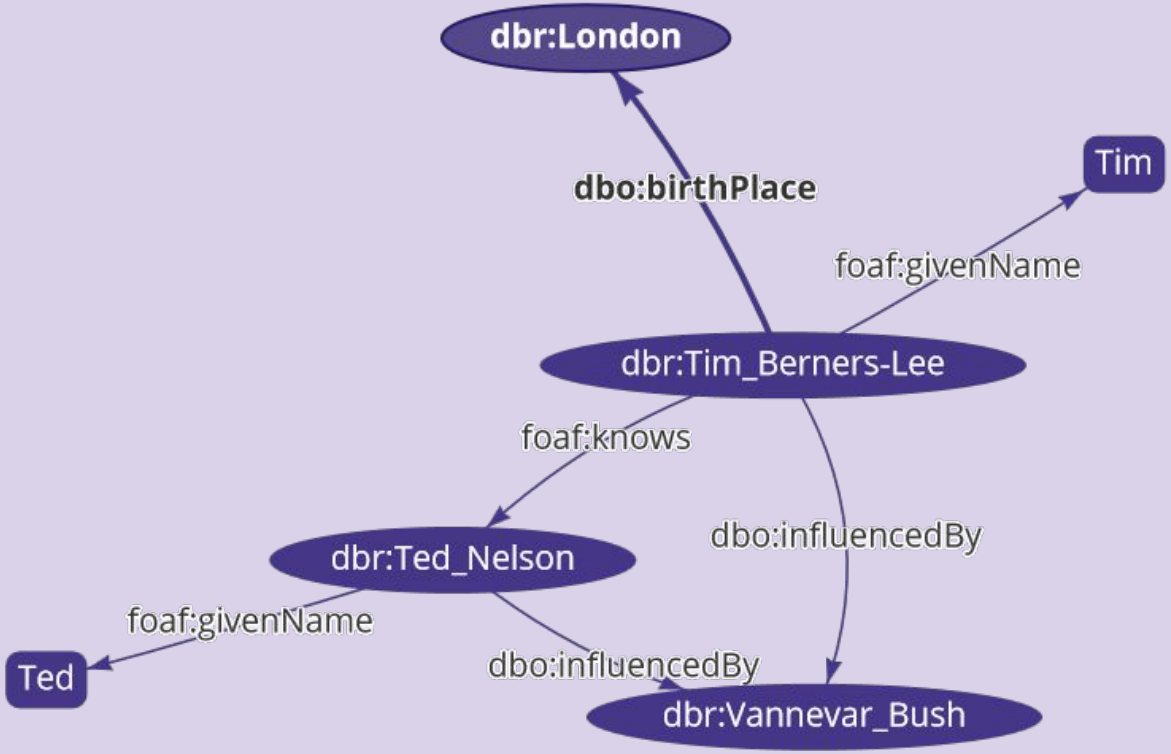
- Hundreds of vocabularies to model data
- Thousands of datasets to map
- Open-World Assumption
  - No dataset is ever complete
  - Other sources *might always* have more data on a subjects

How to represent Linked Data

- Resource Description Framework (**RDF**)
- RDF Graphs as sets of RDF Triples
  - Subject: IRI – dbr:Tim_Berners-Lee
  - Predicate: IRI – foaf:knows
  - Object: IRI – dbr:Ted_Nelson
- Syntaxes
  - Triple-based, JSON-based, XML-based

- Semantic Web, Owl, Web ontologies etc

An RDF Graph


dbr:London
dbo:birthPlace
Tim
foaf:givenName
dbr:Tim_Berners-Lee
foaf:knows
dbo:influencedBy
dbr:Ted_Nelson
foaf:givenName
Ted
dbo:influencedBy
dbr:Vannevar_Bush

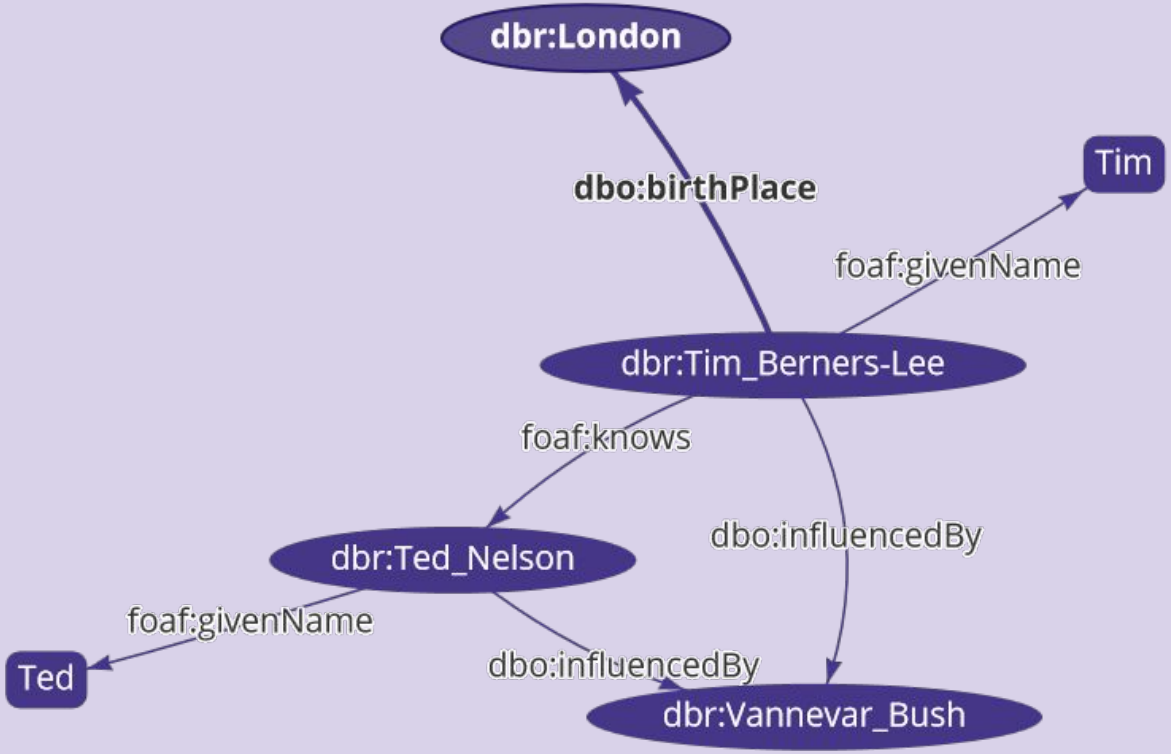
RDF: N-Triples

```
# Every non-empty line represents a triple or comment.
# IRIs are enclosed in angular brackets (< and >).
<http://dbpedia.org/resource/Tim_Berners-Lee> <http://xmlns.com/foaf/0.1/knows>
<http://dbpedia.org/resource/Ted_Nelson>.
# Literals are enclosed in double quotation marks (")
# and optionally end with @ and a language tag.
<http://dbpedia.org/resource/Tim_Berners-Lee> <http://xmlns.com/foaf/0.1/givenName> "Tim"@en.
# Alternatively, they end with ^^ and a datatype IRI.
<http://dbpedia.org/resource/Tim_Berners-Lee> <http://dbpedia.org/ontology/birthDate>
"1955-06-08"^^<http://www.w3.org/2001/XMLSchema#date>.
```

RDF: Turtle

```
# Declare prefixes before use
PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
# The predicate a abbreviates rdf:type.
# A semi-colon ; reuses the subject.
# A comma , reuses the subject and predicate.
dbr:Tim_Berners-Lee a foaf:Person;
                    foaf:knows dbr:Ted_Nelson,
                               dbr:Wendy_Hall.
# There are 3 triples above.
```

An RDF Graph


dbr:London
dbo:birthPlace
Tim
foaf:givenName
dbr:Tim_Berners-Lee
foaf:knows
dbo:influencedBy
dbr:Ted_Nelson
foaf:givenName
Ted
dbo:influencedBy
dbr:Vannevar_Bush

JSON-LD

```
{
  "@context": "http://schema.org/",
  "@id": "http://dbpedia.org/resource/Tim_Berners-Lee",
  "givenName": "Tim",
  "knows": [{
    "@id": "http://dbpedia.org/resource/Ted_Nelson",
    "givenName": "Ted"
  }]
}
```

- @context gives meaning by mapping json terms to IRIs
- @id points to a resource identifier (if not provided by context)

JSON-LD

```
{
  "@context": "http://schema.org/",
  "@id": "http://dbpedia.org/resource/Tim_Berners-Lee",
  "givenName": "Tim",
  "knows": [{
    "@id": "http://dbpedia.org/resource/Ted_Nelson",
    "givenName": "Ted"
  }]
}
```

```
# These triples are equivalent to the JSON-LD example.
PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX schema: <http://schema.org>

dbr:Ted_Nelson schema:givenName "Ted".
dbr:Tim_Berners-Lee schema:givenName "Tim";
                    schema:knows dbr:Ted_Nelson.
```

JSON-LD: A more complicated example

```
{
  "@context":  "https://www.w3.org/ns/activitystreams",
  "@graph": [{
    "type":      "Like",
    "actor":     "https://ruben.verborgh.org/profile/#me",
    "object":    "https://www.ugent.be/#this",
    "published": "2019-04-25T08:00:00Z"
  },{
    "type":      "Like",
    "actor":     "https://example.org/people/silvia#me",
    "object":    "https://www.ugent.be/#this",
```

Working with JSON APIs

1. Gather input data
2. Send a specific API call
3. Parse the response as JSON
4. Traverse the JSON tree structure
5. Update the DOM

Working with RDF

1. Gather input data into a query
2. Send ~~a specific API call~~ the query
3. Parse the response as ~~JSON~~ RDF
4. Traverse the ~~JSON tree structure~~ RDF graph
5. Update the DOM

LDFlex: A DSL for Javascript

```
<LoggedIn>
  <p>Welcome, <Value src="user.firstName"/></p>
  <Image src="user.image" defaultSrc="profile.svg"/>
  <ul>
    <li><Link href="user.inbox">Your inbox</Link></li>
    <li><Link href="user.homepage">Your homepage</Link></li>
  </ul>
  <h2>Your friends</h2>
  <List src="user.friends.firstName"/>
</LoggedIn>
```

LDFlex skips RDF complexity

- Every expression is valid Javascript
  - data.user.friend.firstName
- Feels like a local object
  - But uses Proxy
- Behaves like a Web query
  - await data.user.friend.firstName
  - fetch friends from my data pod
  - fetch their names from **their** data pods

Demo Time!

“*The web as I envisioned it, we have not yet seen*”

Tim Berners Lee

- https://solid.mit.edu/
- https://solid.inrupt.com/
- http://rubenverborgh.github.io/WebFundamentals/semantic-web/
- http://rubenverborgh.github.io/WebFundamentals/decentralization/
- https://archive.fosdem.org/2019/schedule/event/solid_web_decentralization/

References