



LD Generation with Fintan

Christian Fäth (University of Frankfurt, Germany)
Maxim Ionov (University of Frankfurt, Germany)

LD Generation with Fintan

1. Challenges in LD Generation
2. What is Fintan?
3. Segmentation and Parallelization
4. Design Workflows





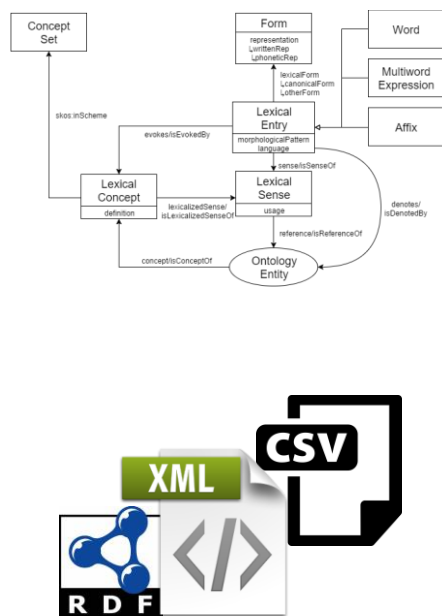
Challenges in LD Generation and Exploitation

Challenges in LD Generation and Exploitation

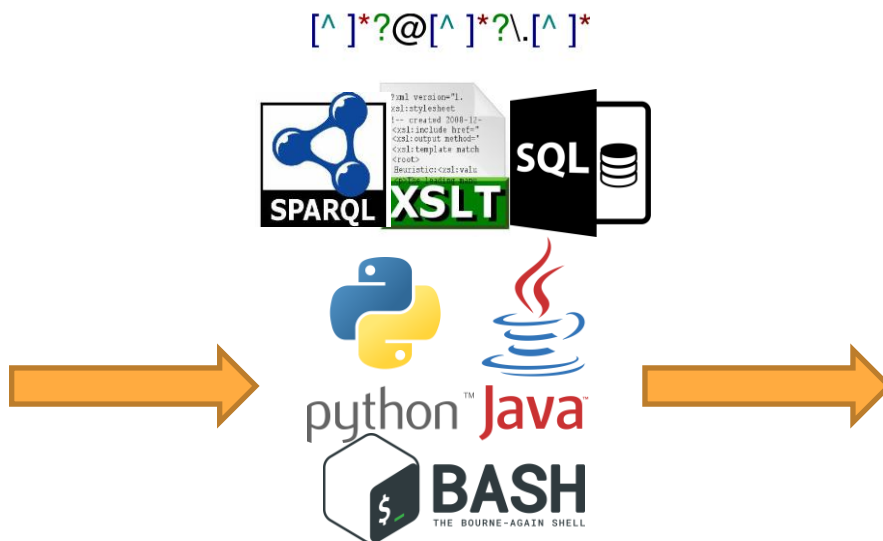
- **Heterogeneity of data**
 - RDF, TSV, XML, Relational, etc.
 - Corpora, dictionaries, taxonomies, etc.
- **Similarity of tasks**
 - E.g. changing annotations in **dictionaries or corpora**
- **Scalability**
 - Segmentation and parallelization
- **Deployment and integration**
 - Integrate into heterogeneous frameworks or pipelines.

A typical pipeline using Language Data

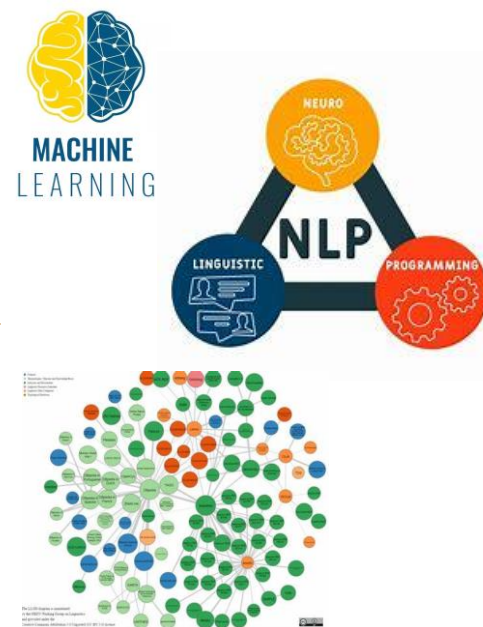
Heterogeneous
Source Data



Merge and Transform



Deploy, integrate, publish



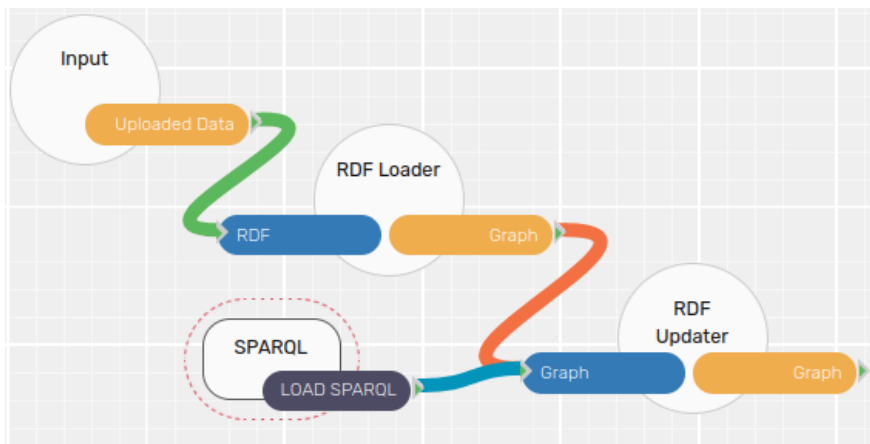


What is Fintan?

Is it a Fish?

- **Fintan mac Bóchra** in Irish folklore was a **shape-shifting sage** who survived a great flood in the shape of a salmon.

- **Complex transformation workflows**
- **Integrate existing converters**
- **Customizable Scripts (SPARQL, XSL, ...)**
- **Common interface**



Name

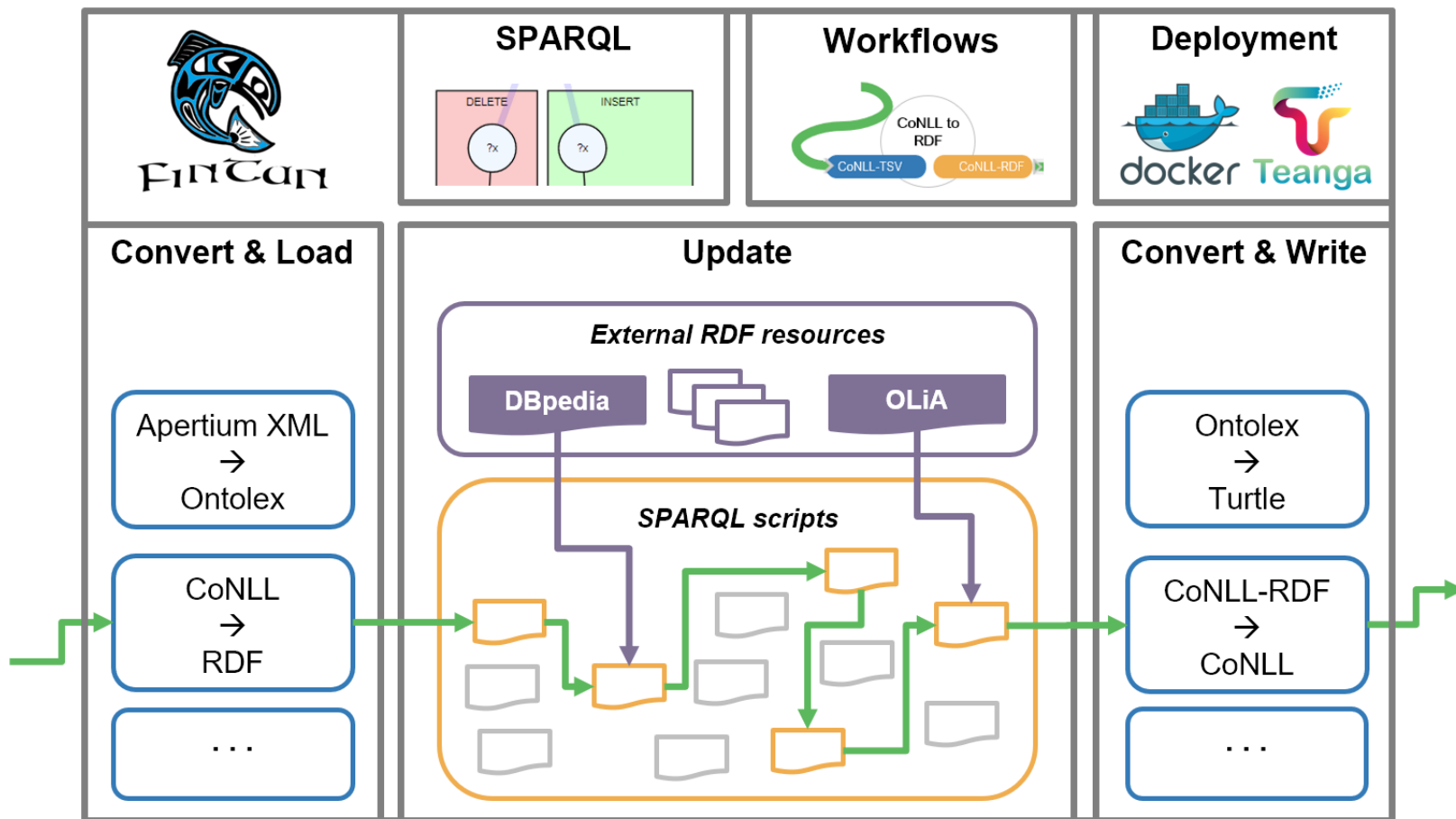
SPARQL

Query

```
1 INSERT {  
2   ?source ?rel  
3   ?target  
4 } WHERE {  
5   ?r a  
6   powla:Relation.  
7   ?r powla:hasSource  
8   ?s.
```

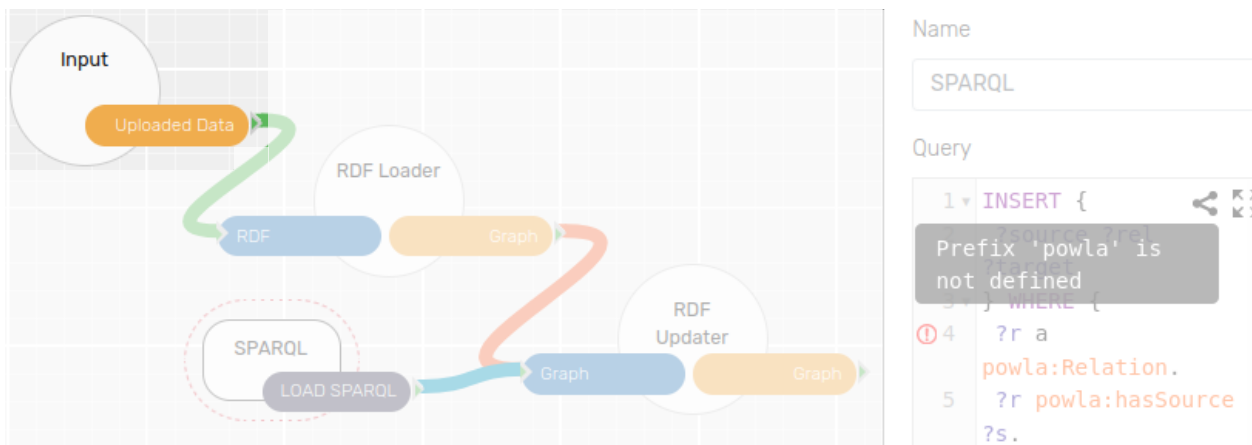



Fintan Platform



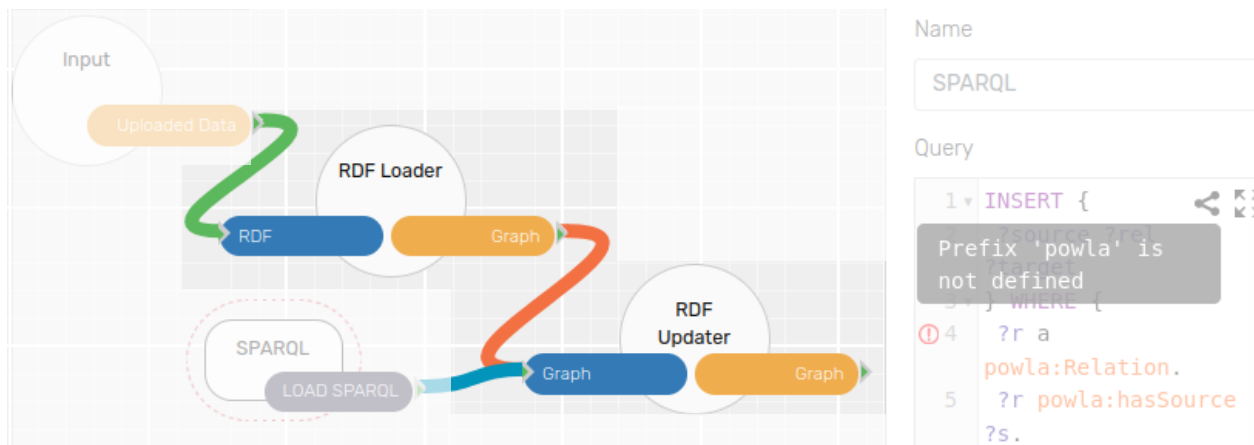
Basic Concepts

- Create complex transformation pipelines.
 - **Data I/O:** Data to be streamed
 - **Components:** Performing transformation steps
 - **Streams:** Streaming data between components
 - **Scripts:** SPARQL, XSL



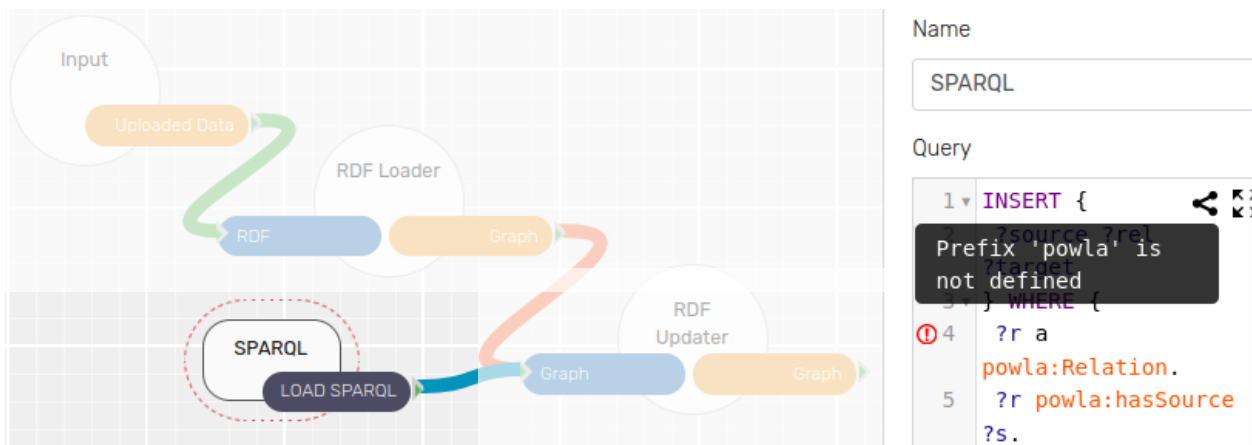
Basic Concepts

- Create complex transformation pipelines.
 - **Data I/O:** Data to be streamed
 - **Components:** Performing transformation steps
 - **Streams:** Streaming data between components
 - **Scripts:** SPARQL, XSL



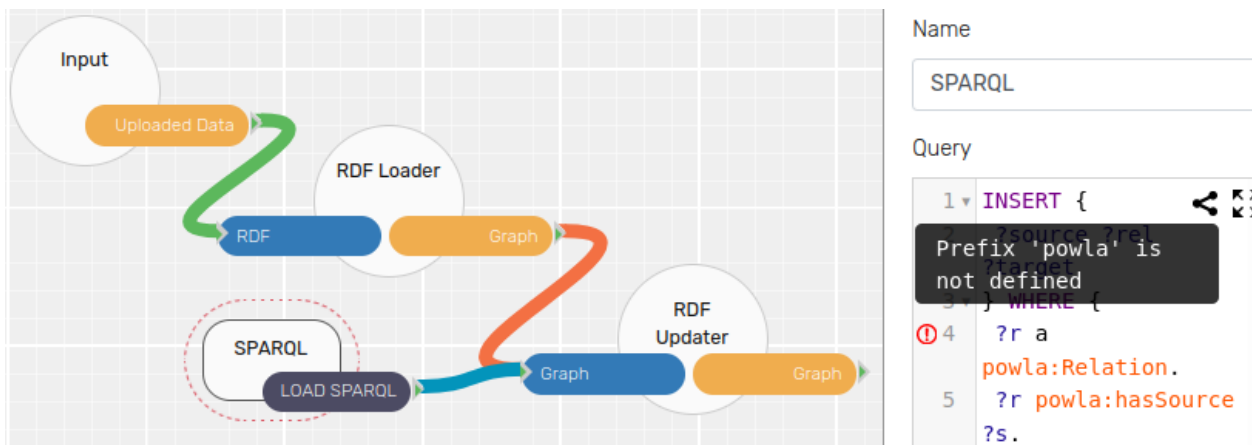
Basic Concepts

- Create complex transformation pipelines.
 - **Data I/O:** Data to be streamed
 - **Components:** Performing transformation steps
 - **Streams:** Streaming data between components
 - **Scripts:** SPARQL, XSL



Basic Concepts

- Create complex transformation pipelines.
 - **Data I/O:** Data to be streamed
 - **Components:** Performing transformation steps
 - **Streams:** Streaming data between components
 - **Scripts:** SPARQL, XSL





Data segmentation

Data segmentation

- **Tackling scalability:**

- **Problem 1:** RDF serialization typically is not context sensitive. When working with RDF data, you need to load the whole dataset in order to operate on it.
- **Problem 2:** Execution time scales exponentially with number of triples.

- **Common Solutions:**

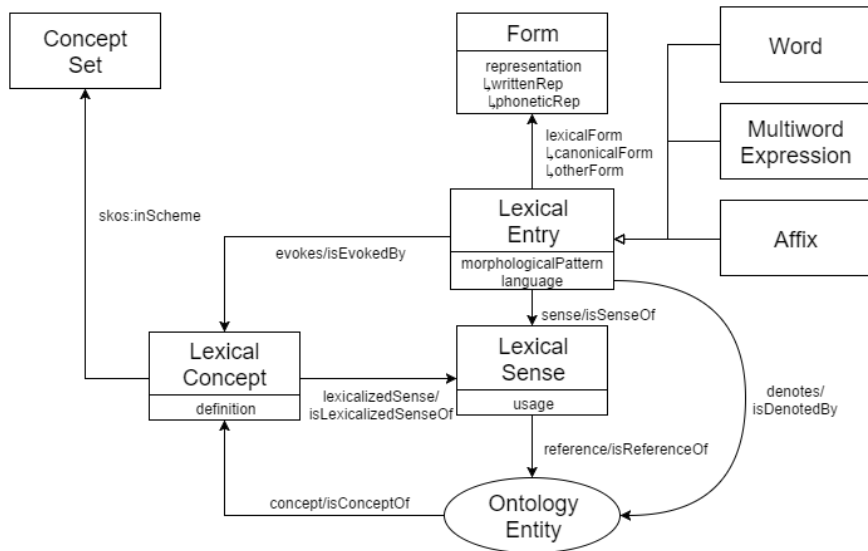
- Powerful servers
- Commercial platforms with GPU acceleration
- DBMS prefetching strategies
- Just wait

- **Fintan:**

- Split data into **contextual segments** (e.g. lexical entries, sentences, words ...)
- Only perform SPARQL Updates on individual segments, **in-memory**
- **Parallelize:** Update multiple segments on independent threads.

Fintan Loader

Structured serialization



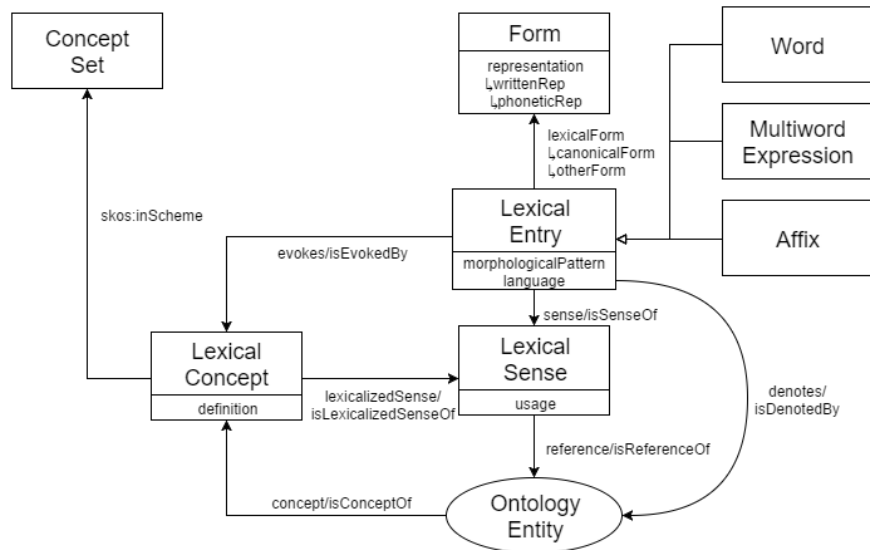
```
:entry1 a ontolex:LexicalEntry ;
      ontolex:lexicalForm :form1_1 ;
      ontolex:sense :sense1_1 .
:form1_1 a ontolex:Form ;
      ontolex:writtenRep "question" .
:sense1_1 a ontolex:LexicalSense .
```

###delimiting line###

```
:entry2 a ontolex:LexicalEntry ;
      ontolex:lexicalForm :form2_1 .
. . .
```


Fintan Splitter - data segmentation

Unstructured serialization



```
:entry1 a ontalex:LexicalEntry ;  
      ontalex:lexicalForm :form1_1 ;  
      ontalex:sense :sense1_1 .
```

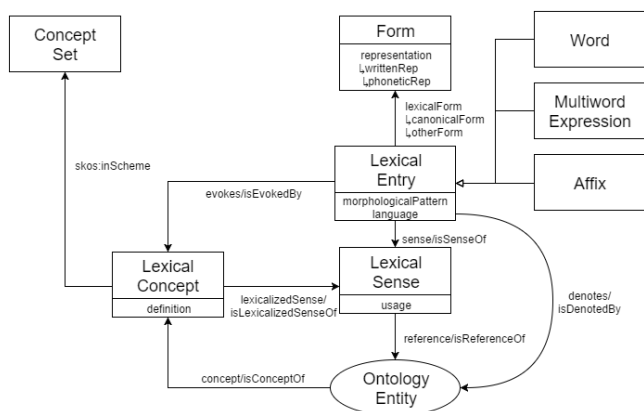
```
:entry82 a ontalex:LexicalEntry ;  
      ontalex:otherForm :form82_1 ;  
      ontalex:sense :sense82_1 .
```

```
:form1_1 a ontalex:Form ;  
      ontalex:writtenRep "question" .
```

. . .

Fintan Splitter - data segmentation

Unstructured serialization

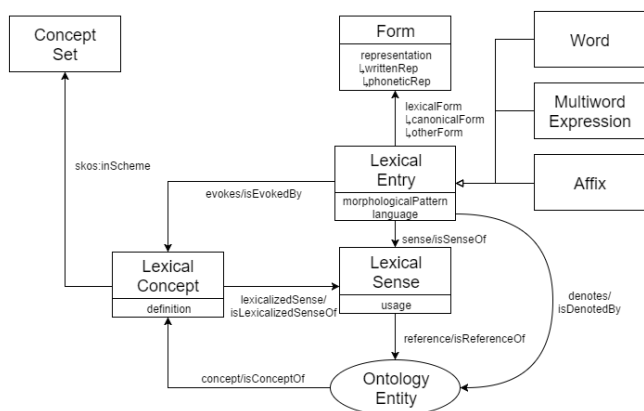


```

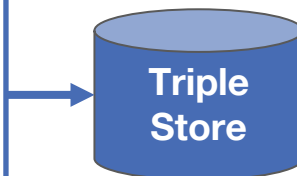
:entry1 a ontolex:LexicalEntry ;
  ontolex:lexicalForm :form1_1 ;
  ontolex:sense :sense1_1 .
:entry82 a ontolex:LexicalEntry ;
  ontolex:otherForm :form82_1 ;
  ontolex:sense :sense82_1 .
:form1_1 a ontolex:Form ;
  ontolex:writtenRep "question" .
. . .
  
```

Fintan Splitter - data segmentation

Unstructured serialization



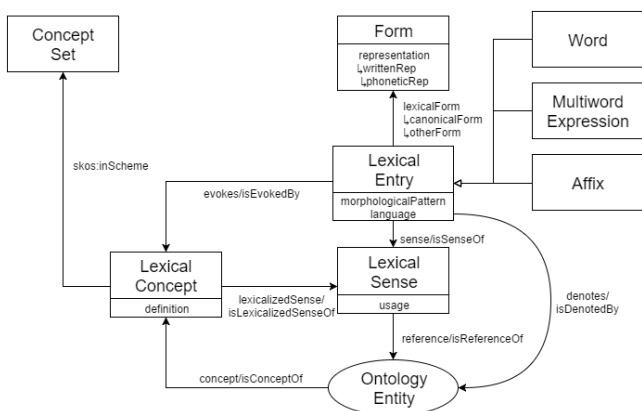
```
:entry1 a ontlex:LexicalEntry ;
  ontlex:lexicalForm :form1_1 ;
  ontlex:sense :sense1_1 .
:entry82 a ontlex:LexicalEntry ;
  ontlex:otherForm :form82_1 ;
  ontlex:sense :sense82_1 .
:form1_1 a ontlex:Form ;
  ontlex:writtenRep "question" .
. . .
```



Triple
Store

Fintan Splitter - data segmentation

Unstructured serialization

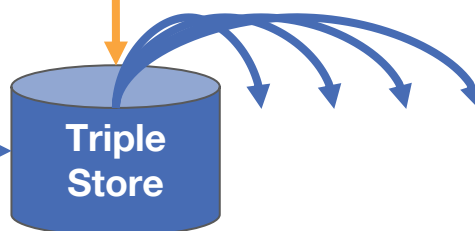


```
:entry1 a ontalex:LexicalEntry ;  
  ontalex:lexicalForm :form1_1 ;  
  ontalex:sense :sense1_1 .  
:entry82 a ontalex:LexicalEntry ;  
  ontalex:otherForm :form82_1 ;  
  ontalex:sense :sense82_1 .  
:form1_1 a ontalex:Form ;  
  ontalex:writtenRep "question" .  
.  
.  
.
```

Segmentation with SPARQL

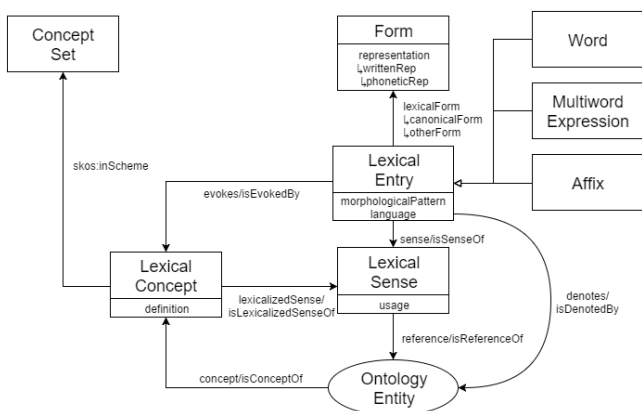
Split into segments

SELECT
iterator
and
CONSTRUCT
per entity



Fintan Splitter - data segmentation

Unstructured serialization



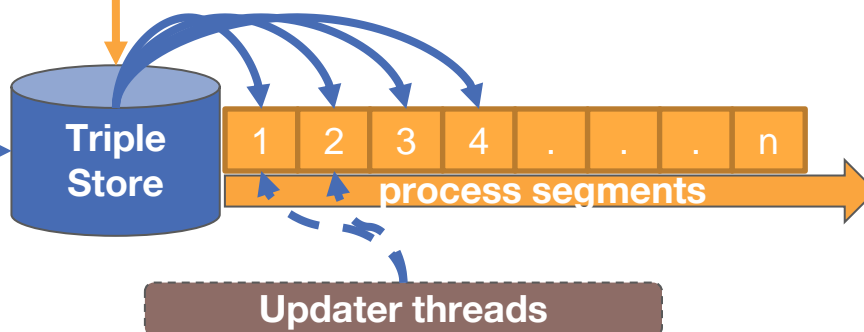
```

:entry1 a ontolex:LexicalEntry ;
  ontolex:lexicalForm :form1_1 ;
  ontolex:sense :sense1_1 .
:entry82 a ontolex:LexicalEntry ;
  ontolex:otherForm :form82_1 ;
  ontolex:sense :sense82_1 .
:form1_1 a ontolex:Form ;
  ontolex:writtenRep "question" .
. . .
  
```

Segmentation with SPARQL

Split into segments

SELECT
iterator
and
CONSTRUCT
per entity



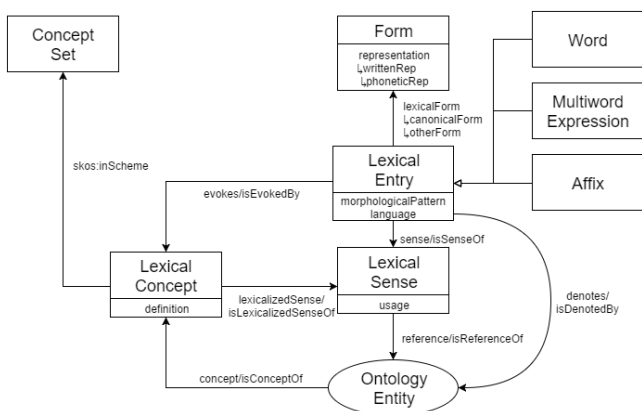
Merging

RESULTING DATA

Segment 1
Segment 2
Segment 3
Segment 4
...
Segment n

Fintan Splitter - data segmentation

Unstructured serialization



```

:entry1 a ontalex:LexicalEntry ;
  ontalex:lexicalForm :form1_1 ;
  ontalex:sense :sense1_1 .
:entry82 a ontalex:LexicalEntry ;
  ontalex:otherForm :form82_1 ;
  ontalex:sense :sense82_1 .
:form1_1 a ontalex:Form ;
  ontalex:writtenRep "question" .
. . .
    
```

Segmentation with SPARQL

Split into segments

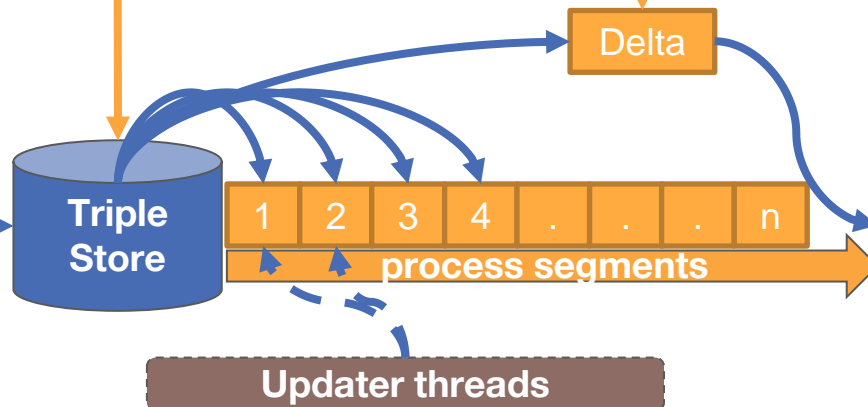
```

SELECT
  iterator
  and
  CONSTRUCT
  per entity
    
```

Extract Delta-Segment

```

{full dataset}
MINUS
{segments}
    
```



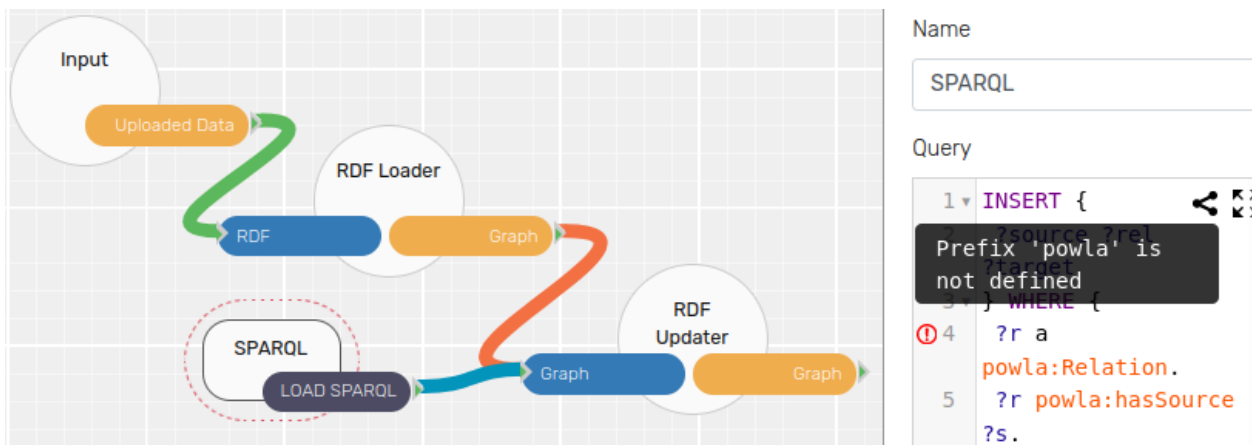
Merging

RESULTING DATA

Segment 1
Segment 2
Segment 3
Segment 4
...
Segment n
Delta-Segment

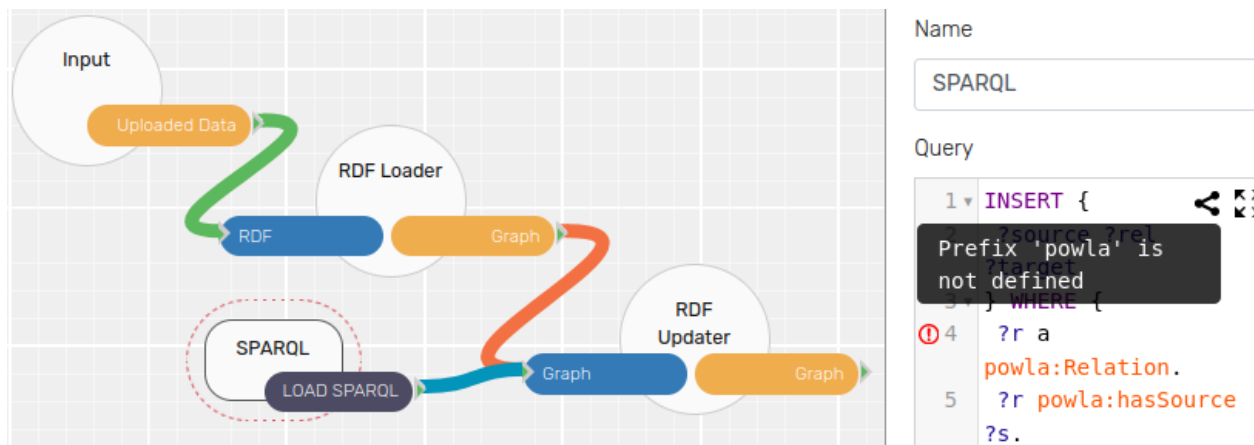
Data segmentation in the UI

- Types of streams:
 - **RED:** Segmented RDF streams, in-memory models
 - **GREEN:** Serialized RDF or general text streams
 - **BLUE:** Side-loaded data or scripts (non-transformable assets)



Data segmentation in the UI

- Types of components:
 - **LOADER / SPLITTER:** read serialized data and produce segmented RDF
 - **UPDATER:** Transform segmented RDF
 - **WRITER:** create serialized data from segmented RDF
 - **Other Transformers:** directly transform serialized data

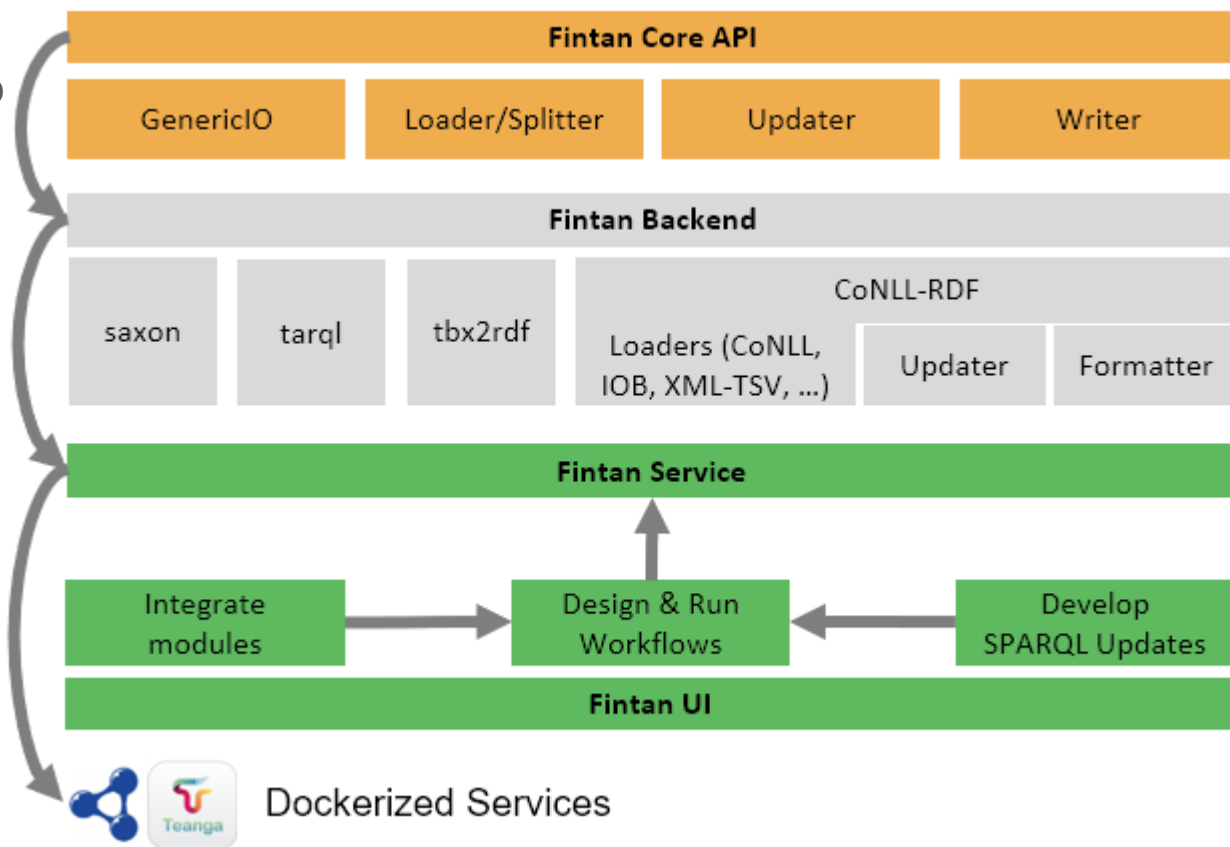




Integrate Fintan into your workflows

Architecture

- Integrate parallel stream processing into your own toolchain
- Host and integrate existing converters
- Build workflows and deploy them for
 - CLI
 - OpenAPI
 - Docker





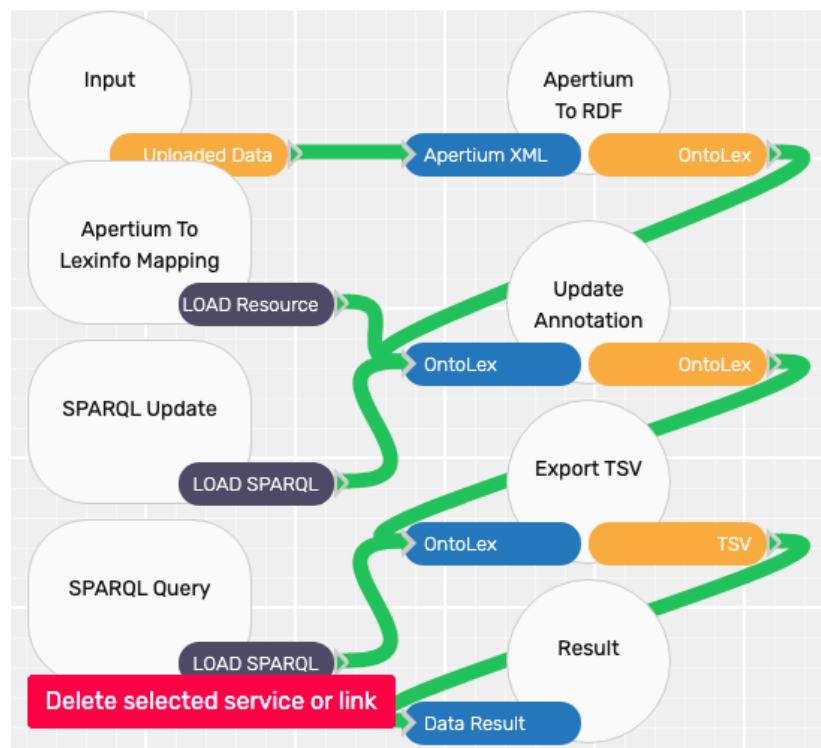
How to use Fintan

- **Java API:**
Directly use Fintan as a Java application **CLI** or as an **API** in your software
- **Integrated Workflow Manager:**
Use Fintan UI as a dockerized service for **building and running pipelines**
- **Fintan-SaaS:** Deploy Integrated Services to other applications
(export Services as **Docker** Containers with **OpenAPI** interfaces)
- **Integrate your own services** in one of two ways:
 - Directly implement the **Java API with Maven** dependencies
 - Directly run **OpenAPI** requests against your transformer services



Design a pipeline

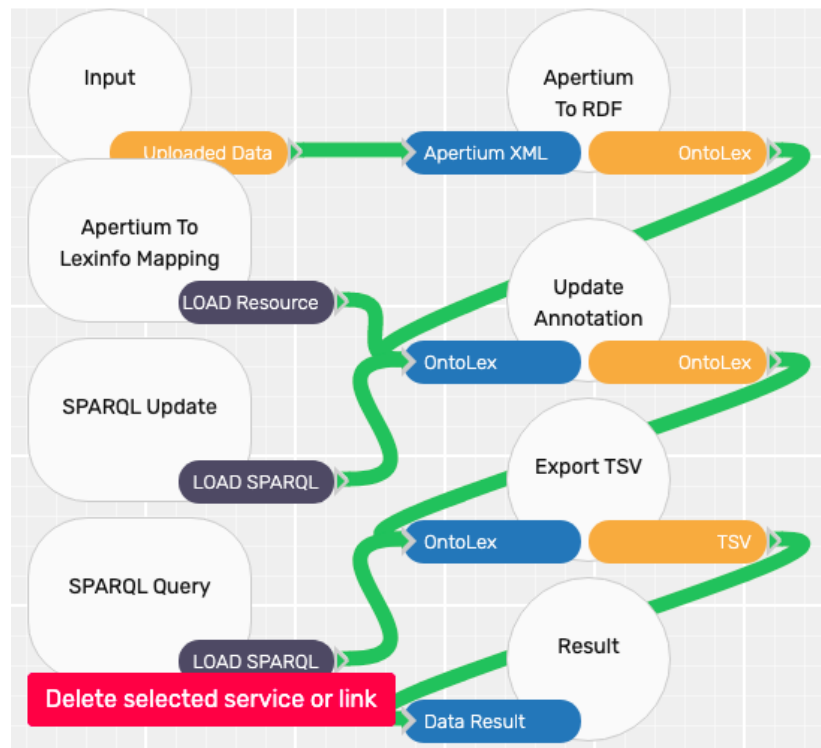
Designing pipelines



```

{
  "input": "System.In",
  "output": "System.Out",
  "pipeline": [
    {
      "class": "ApertiumXMLTransformer",
      "models": [],
      "updates": [
        {}
      ]
    },
    {
      "class": "CoNLLRDFUpdater",
      "models": [
        {
          "source": "apertium-lexinfo-enrichment-update.ttl",
          "graph": "http://linguistic.linkeddata.es/id/apertium/"
        }
      ],
      "updates": [
        { "path": "apertium-pos.sparql", "iter": "1" }
      ]
    },
    {
      "class": "CoNLLRDFFormatter",
      "modules": [
        { "mode": "SPARQLTSV", "select": "apertium-tiad.sparql" }
      ]
    }
  ]
}
  
```

Designing pipelines



- Create Transformation Pipelines
- Export as Docker Containers
- Export as JSON for CLI

Designing pipelines

- Internally, the pipeline is a JSON object
 - with a list of steps
 - and information about resources
- This JSON configuration can be executed by the backend.

```
{
  "input": "System.In",
  "output": "System.Out",
  "pipeline": [
    {
      "class": "ApertiumXMLTransformer",
      "models": [],
      "updates": [
        {}
      ]
    },
    {
      "class": "CoNLLRDFUpdater",
      "models": [
        {
          "source": "apertium-lexinfo-enrichment-update.ttl",
          "graph": "http://linguistic.linkeddata.es/id/apertium/"
        }
      ],
      "updates": [
        { "path": "apertium-pos.sparql", "iter": "1" }
      ]
    },
    {
      "class": "CoNLLRDFFormatter",
      "modules": [
        { "mode": "SPARQLTSV", "select": "apertium-tiad.sparql" }
      ]
    }
  ]
}
```



THANK YOU!

Please fill in google form:

<https://forms.gle/RfFAc83tXMrQC2VL9>