

Movies Web App

Data Layer

Outline

- Movies and Producers
- Standard approaches
- Semantics and Data
- Data Flow

Inspiration

IMDbAllQ

Movies TV News Videos Community IMDbPro App

IMDb Charts: IMDb Top 250

IMDb Charts

[Main index](#)

IMDb Top 250

[IMDb Bottom 100](#)

US Box Office

[USA Top 10](#)

[USA Archive](#)

UK Box Office

[UK Top 10](#)

Top 250 movies as voted by our users

For this top 250, only votes from regular voters are considered.

Track which films you've seen from the Top 250 [right here!](#)

Rank	Rating	Title	Votes
1.	9.2	The Shawshank Redemption (1994)	756,240
2.	9.2	The Godfather (1972)	562,931
3.	9.0	The Godfather: Part II (1974)	355,742
4.	8.9	Pulp Fiction (1994)	595,124
5.	8.9	The Good, the Bad and the Ugly (1966)	235,566

WIKIPEDIA

English
The Free Encyclopedia
3 907 000+ articles

日本語
フリー百科事典
799 000+ 記事

Español
La enciclopedia libre
879 000+ artículos

Русский
Свободная энциклопедия
838 000+ статей

Italiano
L'enciclopedia libera
905 000+ voci

Português
A enciclopédia livre
718 000+ artigos

Deutsch
Die freie Enzyklopädie
1 383 000+ Artikel

Français
L'encyclopédie libre
1 230 000+ articles

Polski
Wolna encyklopedia
887 000+ haseł

中文
自由的百科全书
429 000+ 條目



It would be great to pull these together to figure out who the most important producers are

It'd be even better if it could keep itself updated as new movies come out

Opportunity



Home
[Interlinking](#)
[Statistics](#)
[Licensing](#)
[About](#)

Statistics

The Linked Movie DataBase ([LinkedMDB](#)) contains hundreds of thousands of high-quality interlinks to several movie-related data sources in the [Linking Open Data](#) project, as well as hundreds of thousands of links to movie-related web pages.

LinkedMDB is one of the densest examples of interlinking among the datasets in the Linking Open Data cloud, with as many interlinks and page references as entities. **We expect these numbers to continue to grow significantly.**



[DBpedia Blog](#) | [Get Involved](#) | [Get Help](#)

[About / News](#)
[Applications](#)
[Use Cases](#)
[Datasets](#)
[Online Access](#)
[DBpedia Live](#)
[Downloads](#)
[Interlinking](#)

DBpedia is a community effort to extract allows you to ask sophisticated queries a it easier for the amazing amount of inform for navigating, linking and improving the e

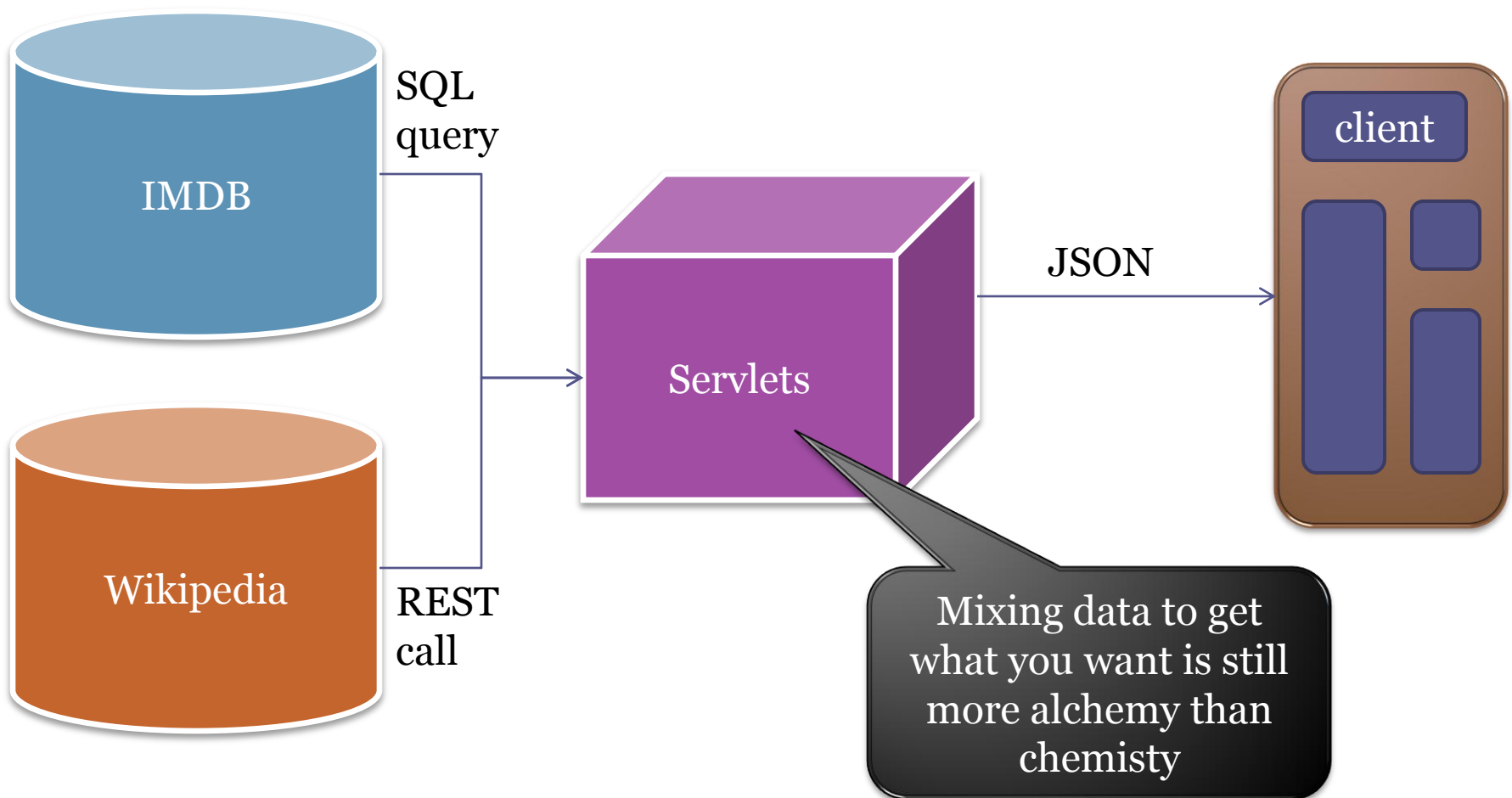
News

[DBpedia Spotlight has been selected for Google](#)
The Google Summer of Code (GSoC) is a global had thousands of participants since the first edit Gnome, Apache Software Foundation, Mozilla, i
[DBpedia 3.7 released, including 15 localized Ed](#)
Hi all, we are happy to announce the release of

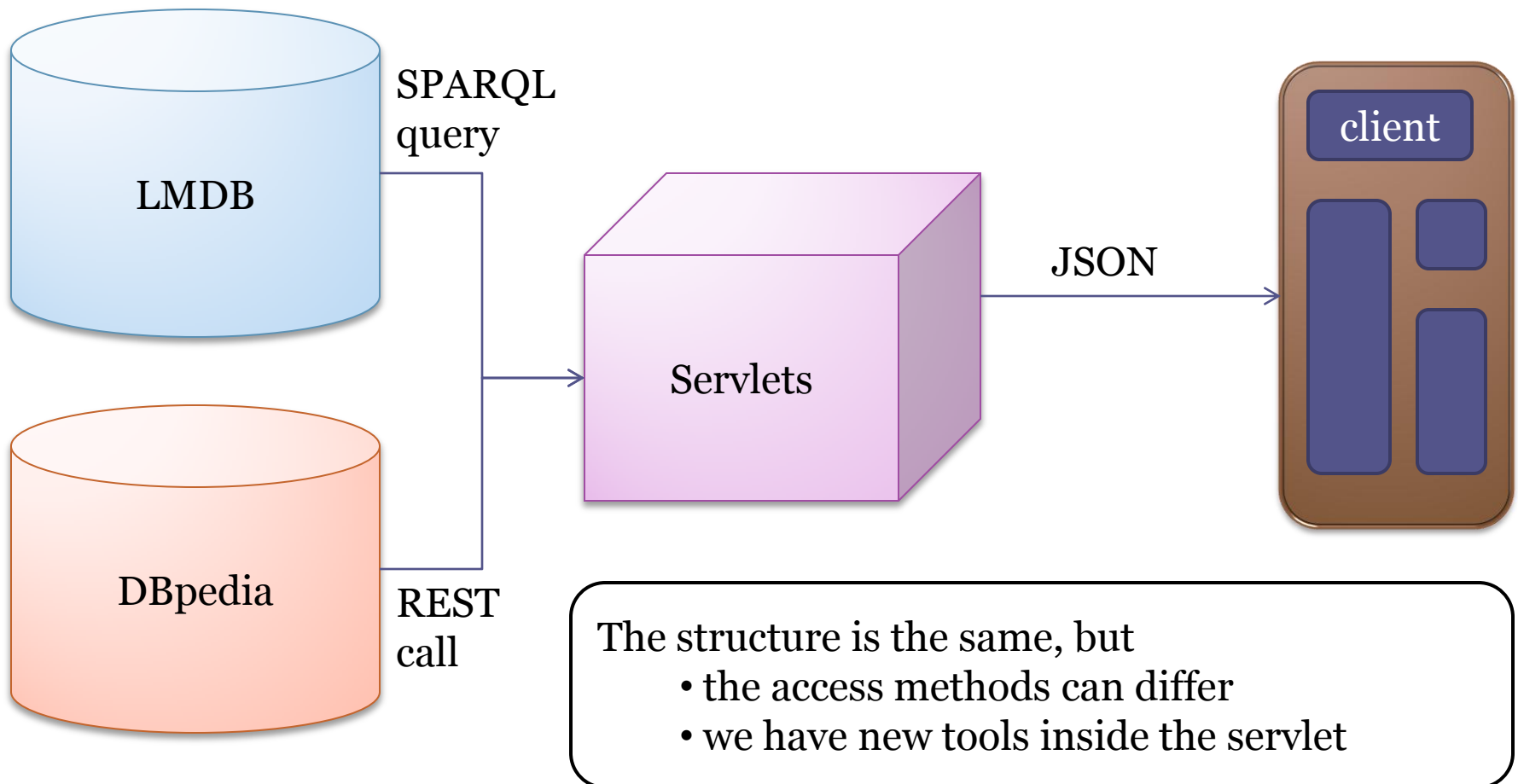
Fortunately, there are open data sources with similar content

Oddly, they insist on being Semantic Web sources

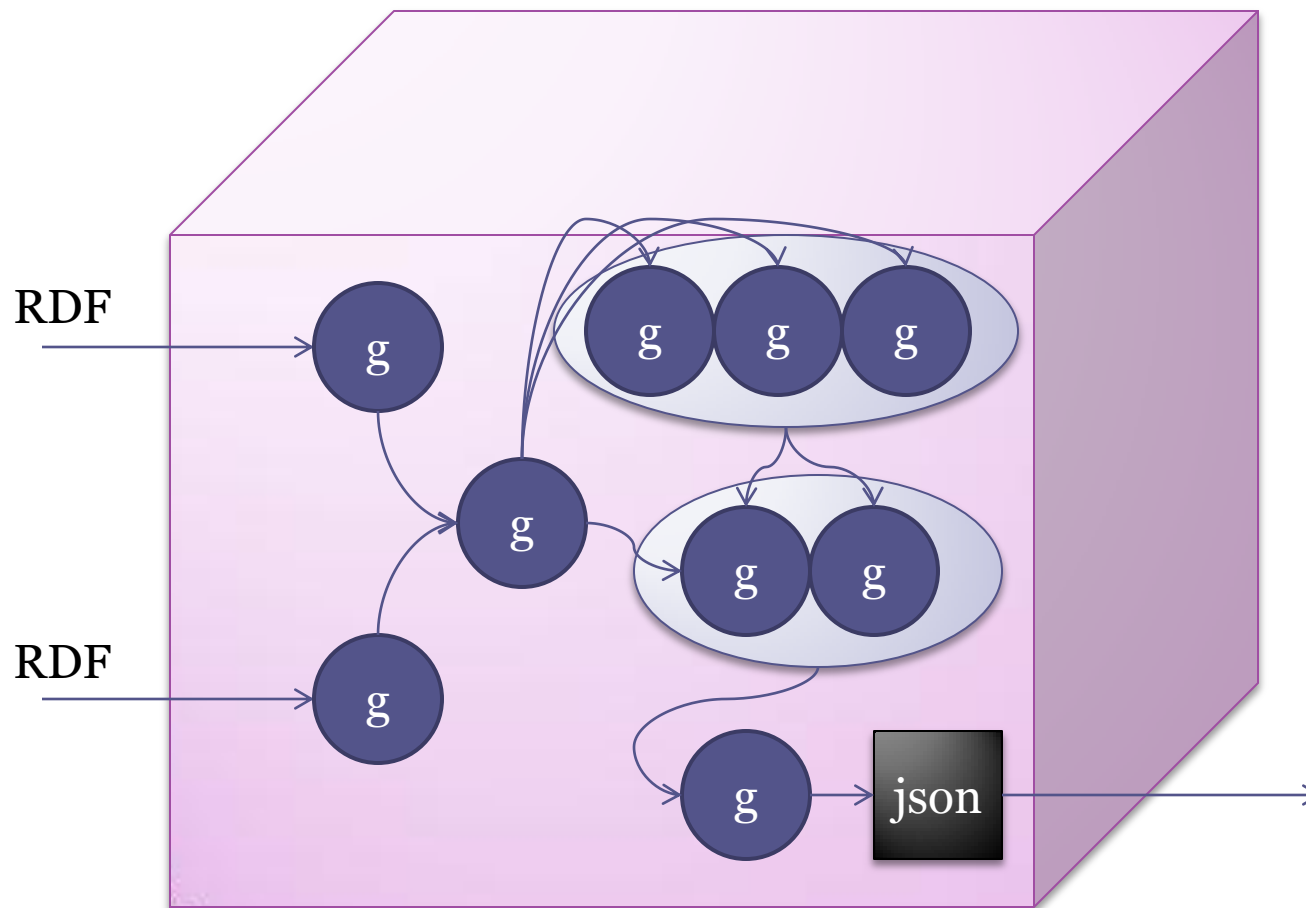
The Standard Approach



Semantic Alchemy



Semantics is odd

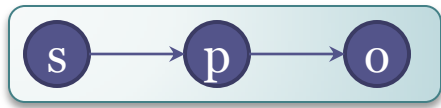


We tend not to rip through XML, JSON, or POJOs

Rather, semantic processing is a cascade of graph transformations

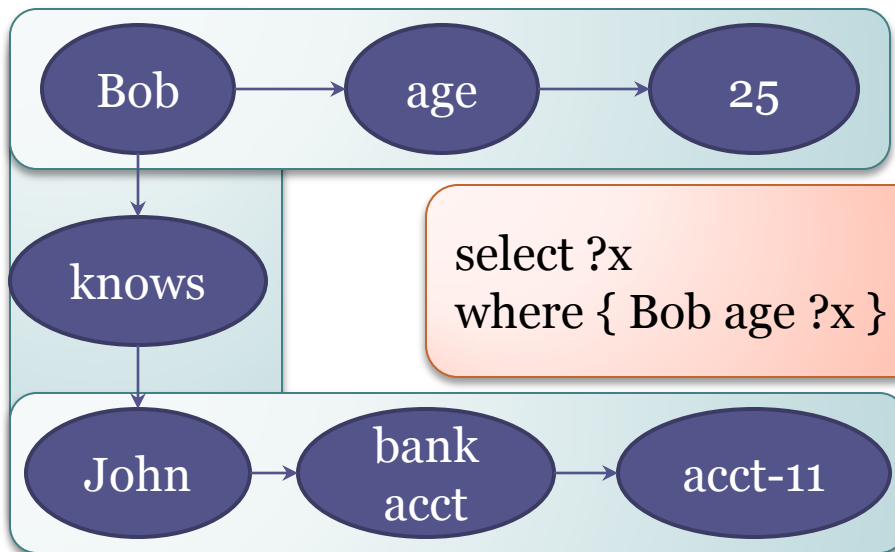
At the end, we still respond to the client with a sensible JSON object

Standardization in Semantics



An RDF fact:

- subject
- predicate
- object



A SPARQL
query against
three facts

It's not down to a
science, but there are
standards to lean on

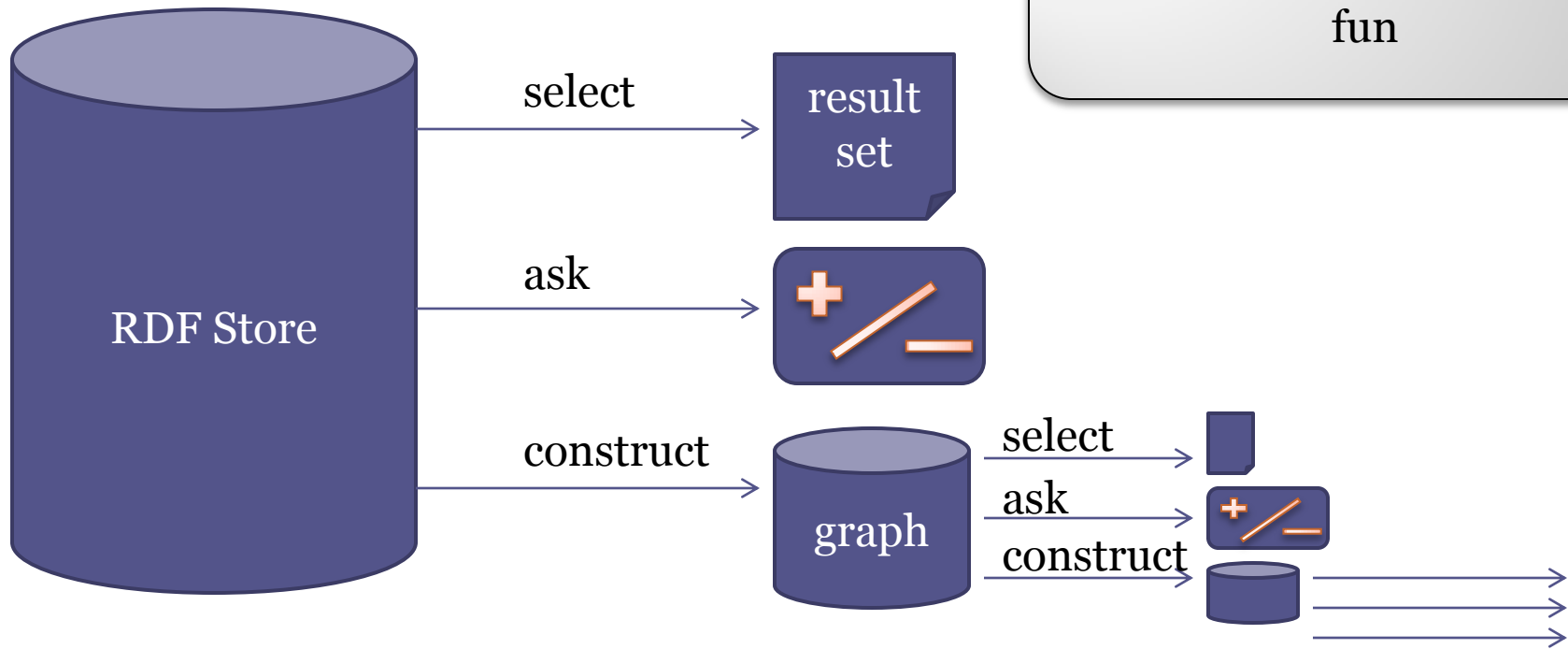
RDF – data language

SPARQL – query
language

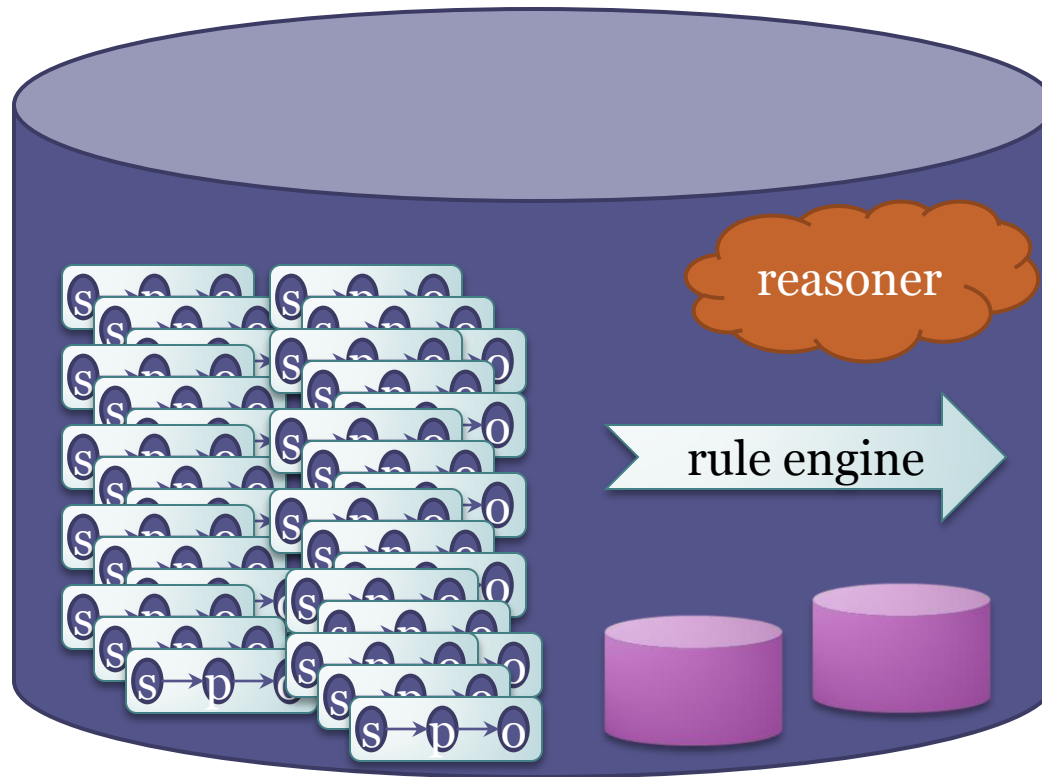
Jena – de facto
standard Java library
for semantics

SPARQL is special

Those constructs give you composition, which is a lot of fun



RDF graphs are special too



Primarily they
are nests of
RDF facts

But we also get
to put in

- reasoners
- rule engines
- sub-graphs

Semantics and Movies

Who are the top 100 movie producers?

- Name
- # of films
- Gravity
- Experience
- Street Cred
- Most frequent film rating
- Associated Production Companies

Some of these properties require a bit of elucidation:

Gravity: How well does the producer attract stars?

Experience: How much in the way of quality film-making has this producer given us?

Street Cred: How many directors and writers are willing to work with the producer?

Tools

- Clojure
- Seabass
- RDF/RDFS (Turtle)
- SPARQL
- JSON

```
(defn lcd [n t]
  (loop [a t]
    (cond (> (* a a) n) n
          (= (rem n a) 0) a
          (= 2 a) (recur 3)
          true (recur (+ a 2)))))
(defn prime? [x] (= x (lcd x 2)))
(defn next-prime [x]
  (loop [a (+ x 2)]
    (cond (= x 2) 3
          (prime? a) a
          true (recur (+ a 2)))))
```

- A Lisp on the JVM
- Handles concurrency very well
- I like it
- Everything done here could be done in Java, Scala, Groovy, Python, Ruby, or even C++

Tools

- Clojure

```
(def q "      select ?x ?y ?z
      where {
          ?x <http://ex.org/foo> ?y .
          ?z <http://ex.org/bar ?y . }")
(bounce q (build "data/my-ont.ttl" "data/your-ont.owl"))
```

- Seabass

- RDF/RDFS (Turtle)

- SPARQL

- JSON

- A Clojure library I wrote around Jena
- Simplified interface:
 - **build** – create a model from local files, remote files via URL, and other models
 - **bounce** – get results from a model with a SELECT query
 - **pull** – get a model from a model with a CONSTRUCT query
 - **ask** – get a boolean from a model with an ASK query
 - **stash** – save a model as a local file as N-triples

Tools

- Clojure
- Seabass
- RDF/RDFS (Turtle)
- SPARQL 1.1
- JSON

```
nhl:hometeam    rdfs:domain nhl:Game ;  
                rdfs:range  nhl:Team ;  
                rdfs:subPropertyOf nhl:team .  
  
nhl:awayteam    rdfs:domain nhl:Game ;  
                rdfs:range  nhl:Team ;  
                rdfs:subPropertyOf nhl:team .
```

```
(def team-names "  
prefix : <http://www.nhl.com/>  
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>  
construct { ?team :name ?name }  
{ select distinct ?team ?name  
  { ?game :team ?team . ?team rdfs:label ?name }  
}  
")
```

- RDF/S is neat
- Turtle is the only RDF syntax
- SPARQL 1.1 is likewise neat

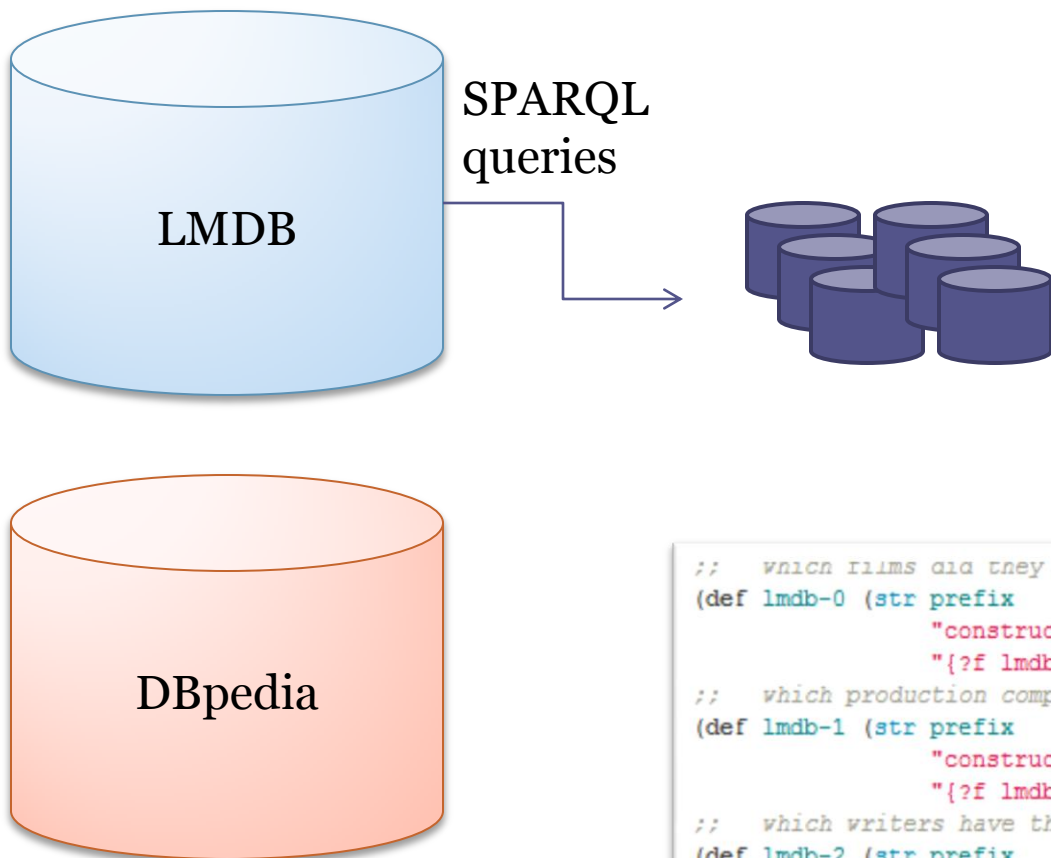
Tools

- Clojure
- Seabass
- RDF/RDFS (Turtle)
- SPARQL
- JSON

```
game: {
  awayteamid: 8,
  awayteamname: "Montreal Canadiens",
  hometeamname: "Toronto Maple Leafs",
  plays: {
    play: [
      {
        sweater: "37",
        localtime: "7:29 PM",
        xcoord: 87,
        desc: "Tim Brent HIT on Josh Gorges",
        teamid: 10,
        strength: 701,
        pid: 8470283,
        formalEventId: "TOR51",
        period: 1,
        type: "Hit",
        p3name: "",
        eventid: 51,
        p2name: "Josh Gorges",
        ycoord: 36,
```

- The very popular data format for web things
- Better than XML in every way
- It's the native object syntax for Javascript

Data Flow

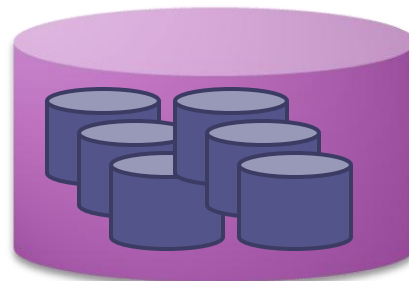
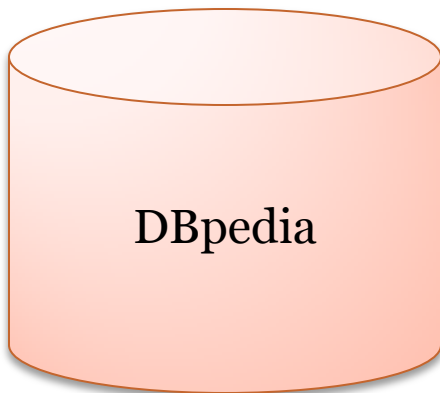
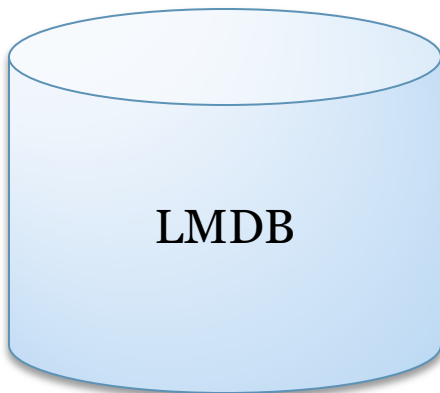


We want a flow that can be run every morning and update our RDF store.

We want to be good netizens, so we shouldn't ask for more than we need.

```
;; which films did they produce?
(def imdb-0 (str prefix
  "construct {?x :produced ?f . ?f :title ?t . ?f :date ?d} "
  "{?f imdb:producer ?x . ?f dc:title ?t . ?f dc:date ?d}"))
;; which production companies have they worked with?
(def imdb-1 (str prefix
  "construct {?x :worked-with ?y . ?y a :Production-Company} "
  "{?f imdb:producer ?x . ?f imdb:production_company ?y}"))
;; which writers have they worked with?
(def imdb-2 (str prefix
  "construct {?x :worked-with ?y . ?y a :Writer} "
  "{?f imdb:producer ?x . ?f imdb:writer ?y}"))
```

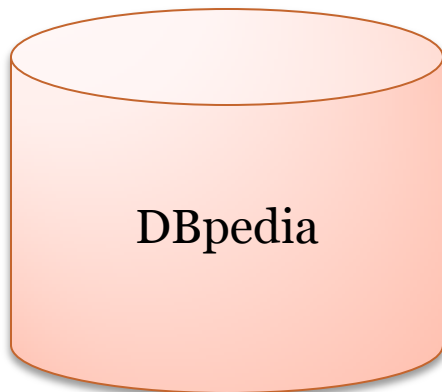
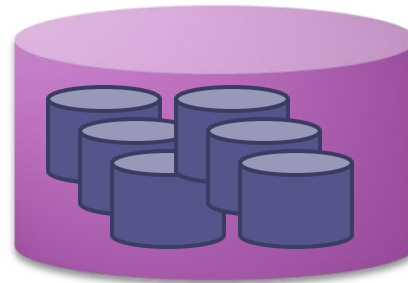
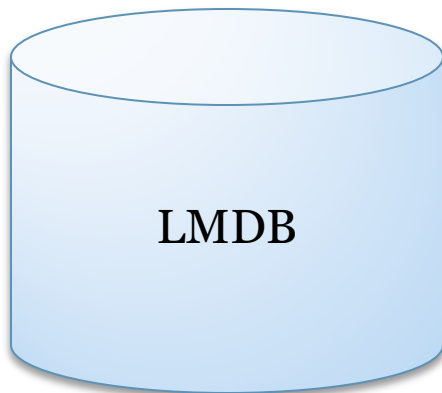

Data Flow



Another nice thing
about RDF is that it is
- very easy –
to combine graphs...

just smoosh them
together

Data Flow



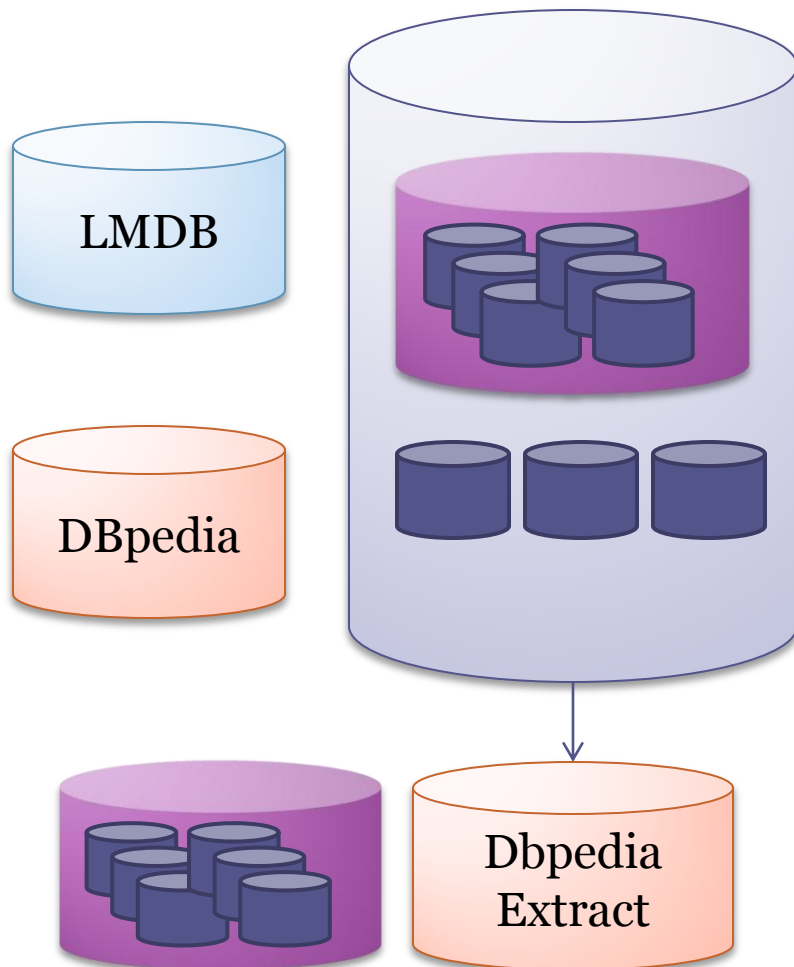
SPARQL
queries

Up to now, we've
ignored Dbpedia

That's because we're
using it as icing on our
LMDB cake

```
;; what were these films' budgets? (dbp)
(def dbp-0 (str prefix
  "construct {?f :dbp-budget ?b} "
  "{?f dbp:budget ?b}"))
;; what were these films' grosses? (dbp)
(def dbp-1 (str prefix
  "construct {?f :dbp-gross ?g} "
  "{?f dbp:gross ?g}"))
;; who were these films' stars? (dbp)
(def dbp-2 (str prefix
  "construct {?f :dbp-starring ?a . ?a :name ?name} "
  "{?f dbp:starring ?a . ?a rdfs:label ?name}"))
```

Data Flow



```
;; the integration queries clean up the ties betw the sources
(def int-0 (str prefix
  "construct {?f :budget ?v} "
  "{ select ?f ?v "
  " { ?f owl:sameAs/:dbp-budget ?b "
  "   bind (xsd:float(?b) as ?v) "
  " } "
  "}")
  "}")

(def int-1 (str prefix
  "construct {?f :gross ?v} "
  "{ select ?f ?v "
  " { ?f owl:sameAs/:dbp-gross ?g "
  "   bind (xsd:float(?g) as ?v) "
  " } "
  "}")
  "}")

(def int-2 (str prefix
  "construct {?f :starring ?a} "
  "{?f owl:sameAs/:dbp-starring ?a}"))
```

Integration Time

LMDB uses some Dbpedia resources

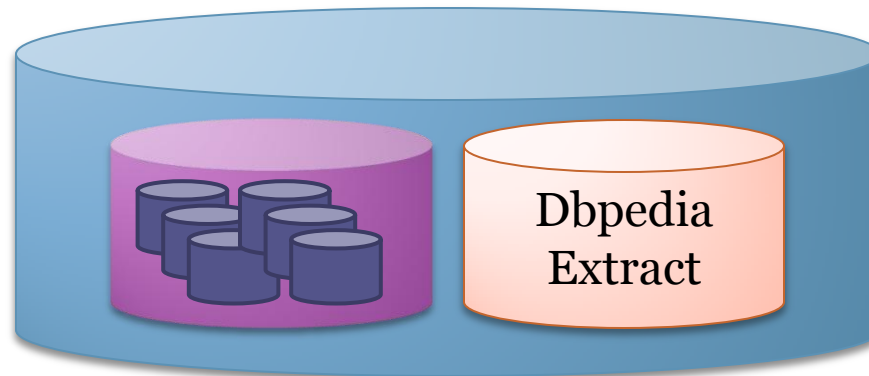
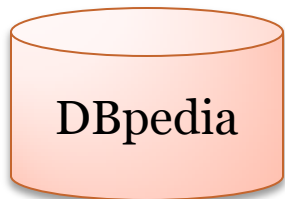
So we want to pull out of our Dbpedia results only those resources that we're using

We'll have to make a temporary copy of our LMDB extract to do this

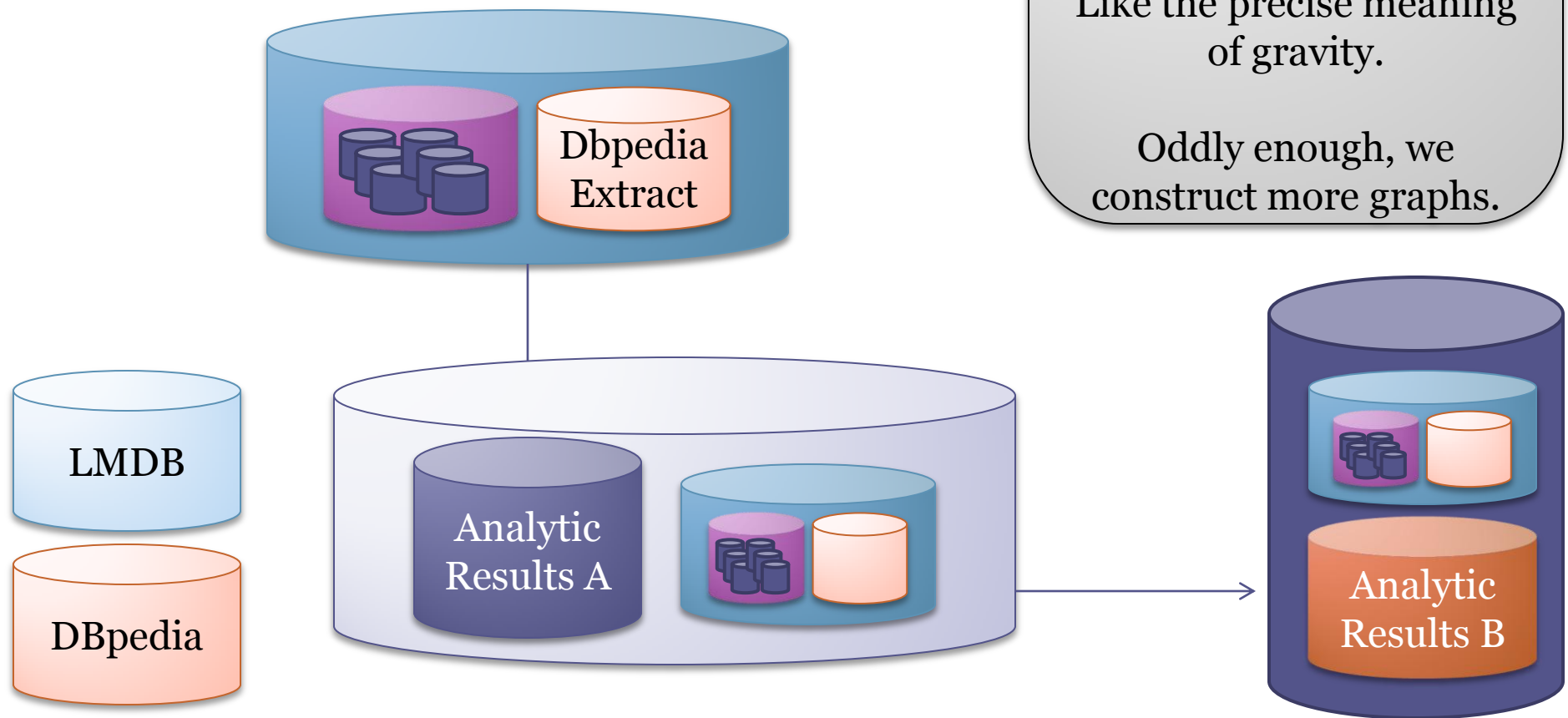
Data Flow

Our Integration Graph is done

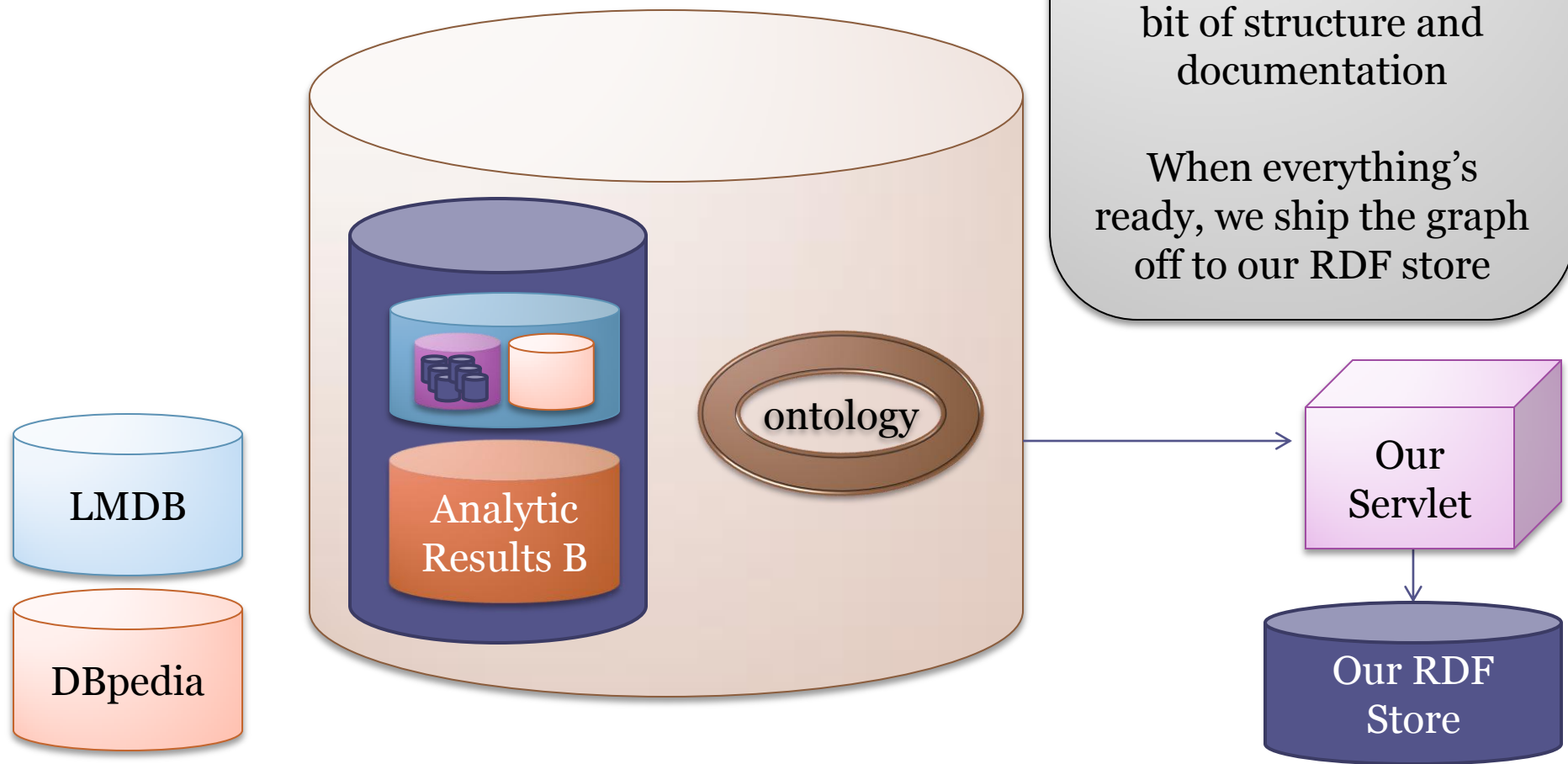
This represents the base data set drawn from our sources, with minimal interpretation done.



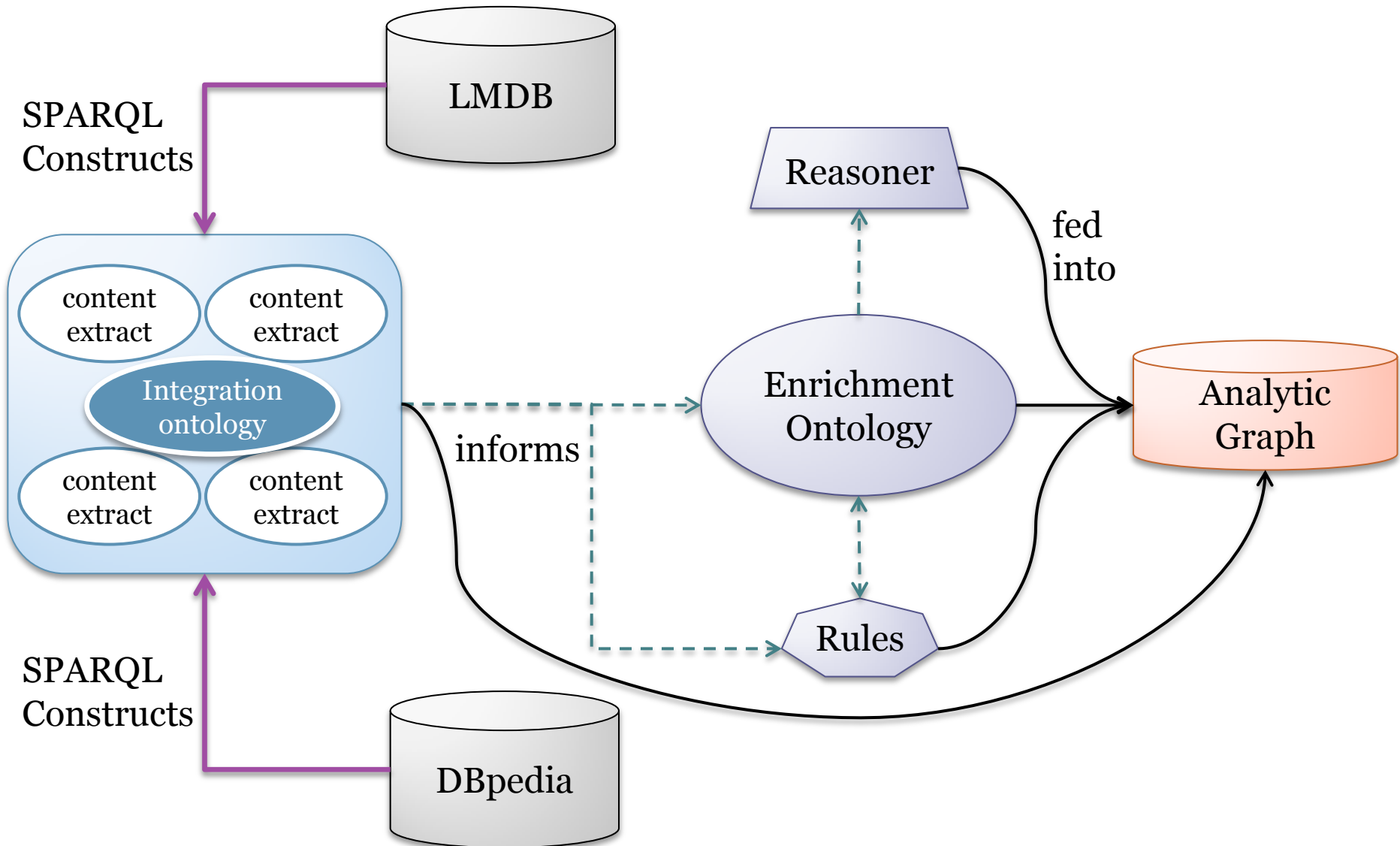
Data Flow



Data Flow



Overview of the Data Flow



What's the payoff for this oddity?

- The algorithms are fairly resilient
 - Only W3C standards are used, so rewriting in Python, Ruby, Groovy, or even R is somewhat straightforward
 - The approach works for pretty much any RDF endpoint you want to touch
 - You can encapsulate all this semantic logic in a single place, so the rest of your code looks normal
 - It's fun



Questions?