INFO216:
Knowledge Graphs

Andreas L. Opdahl
<Andreas.Opdahl@uib.no>

(c) Andreas L Opdahl, 2020

Session 4: Application architecture

- Themes:
  - application architecture for the web of data
  - components of web-of-data applications
    - triple stores & Blazegraph
    - *programming against RDFLib in Python*
      - *examples at wiki.uib.no/info216*
    - visualization:
      - front-ends: Google Charts / SGvizler
      - ontologies: VOWL / WebVOWL

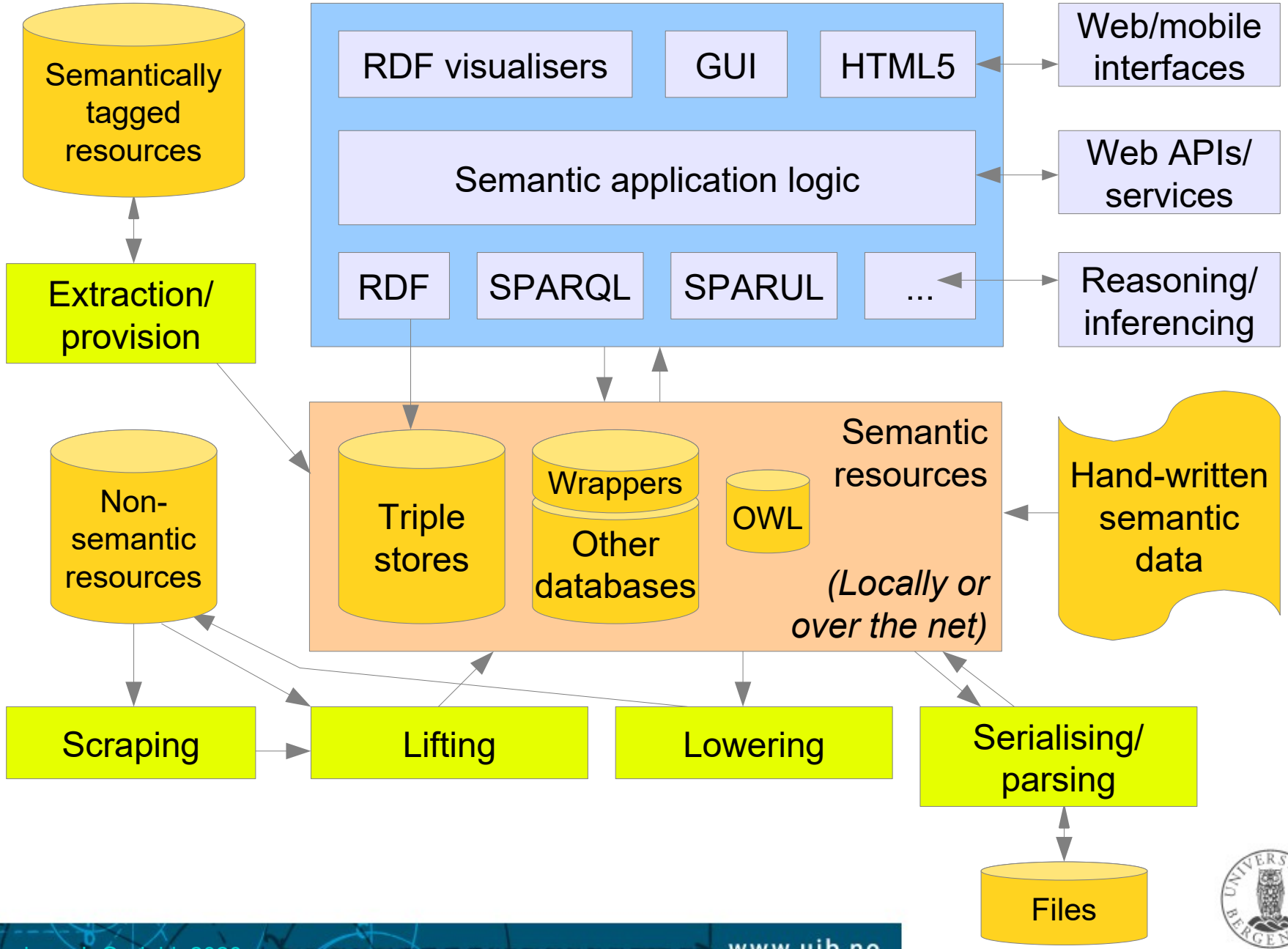
Readings

- Sources:
  - Allemang & Hendler (2011):
    Semantic Web for the Working Ontologist,
    - chapter 4 on application architecture
  - materials at wiki.uib.no/info216

Expectations to the lab presentations

- Prepare slides! Bring your own computer!
- Which data sets you plan to use?
  - do you need to lift them?
  - if so, how?
- Which vocabularies you plan to use?
- What will you use them for?
  - queries (and updates?) you plan to support
  - possible use cases / scenarios
  - possible apps or web pages
- *Friday 14th and Wednesday 19th*

Application architecture for the Web of Data


Semantically tagged resources
RDF visualisers
GUI
HTML5
Web/mobile interfaces
Semantic application logic
Web APIs/ services
RDF
SPARQL
SPARUL
...
Reasoning/ inferencing
Extraction/ provision
Non-semantic resources
Triple stores
Wrappers
Other databases
OWL
Semantic resources
(Locally or over the net)
Hand-written semantic data
Scraping
Lifting
Lowering
Serialising/ parsing
Files

Parsing/serialising

- Reading from (“parsing”) and writing to (“serialising”) standard RDF formats
- Why different formats?
  - compactness, XML-dependency
  - can the same data set be stored in many ways?
  - machine versus human readability, abbreviations
  - CURIEs (“Compact URIs”) with qname/prefix
  - nested resources
  - scope: only basic RDF or also, e.g., quads, rules , OWL...
- Built into all RDF- (and OWL-) programming frameworks
  - e.g. RDFLib, Jena

Example: TURTLE

- *"Terse RDF Triple Language"*
  - *extends the N-Triple format*
  - *restricts the Notation 3 (N3) -format*
  - *not XML-based (like RDF/XML), but simpler to read*
  - *supports prefixes (and bases)*
  - *writing multiple predicates-objects for the same subject*
  - *writing multiple objects for the same subject-predicate*
  - *flexible notations for blank/anonymous nodes: [], [...]*
  - *SPARQL uses TURTLE-like syntax*
  - *OWL is sometimes written in TURTLE*
    - *but OWL also has its own notations!*
  - *TriG extends TURTLE to support named graphs/quads*

Example: TURTLE

N-TRIPLE:

<http://r.e.x/Harald> <http://r.e.x/ektefelle> <http://r.e.x/Sonja> .
<http://r.e.x/Harald> <http://r.e.x/barn> <http://r.e.x/Haakon_Magnus> .
<http://r.e.x/Harald> <http://r.e.x/barn> <http://r.e.x/Martha_Louise> .

TURTLE:

<http://r.e.x/Harald> <http://r.e.x/ektefelle> <http://r.e.x/Sonja> **;**
<http://r.e.x/barn> <http://r.e.x/Haakon_Magnus> **,**
<http://r.e.x/Martha_Louise> **.**

- *semicolon (;) means "new predicate, same subject"*
- *comma (,) means "new object, same subject, predicate"*
- *period (.) means "new subject"*

Example: TURTLE

TURTLE:

```
@prefix rex: <http://r.e.x/> .
rex:Harald  rex:spouse     rex:Sonja ;
            rex:chld       rex:Haakon_Magnus ,
                           rex:Martha_Louise .
```

- *@prefix allows use of Compact URIs (“Curies”)*
- *@base allows use of IRI-fragments*
- *we have looked at blank/anonymous nodes already...*

Example: TriG

TriG:

```
@prefix rex: <http://r.e.x/> .
rex:Royal   { rex:Harald    rex:spouse    rex:Sonja ;
                            rex:child     rex:Haakon_Magnus ,
                                          rex:Martha_Louise .  }
rex:Mine    { reg:Andreas   reg:spouse    reg:Monika ;
                            reg:child     reg:Jens_Christian .  }
```

- *extends Turtle with named graphs wrapped in { ... }*
- *A bit similar to SPARQL, which adds the keyword*

```
GRAPH rex:Royal   {
        rex:Harald rex:spouse rex:Sonja .
}
```

Other RDF serialisations

- RDF/XML *(the original XML serialisation)*
- TriX *(XML-based, experimental, named graphs)*
- N-TRIPLE *(maximally simple format, has "canonical form")*
- NQ, NQUAD *(extends N-TRIPLE with quads)*
- TURTLE ("Terse RDF Triple Language")
  (builds on N-TRIPLE, human readable, SPARQL ++)
- TriG *(TURTLE-extension, named graphs)*
- Notation3, N3 *(builds on TURTLE, supports rules, graphs ++)*
- JSON-LD *("JavaScript Object Notation – Linked Data")*
- Embedded formats:
  - microformats, (eRDF →) RDFa, microdata
- In addition, OWL has its own serialisations...
  - RDF/XML and TURTLE are sometimes used

Scraping

- Making less structured data locally available in a well-structured format
- Typically used on internet data:
  - from less to more explicitly structured formats
  - HTML, PDF, DOCX, TXT, tagged file formats
- Storing the result in, e.g., CSV, XML or JSON
- A useful “technical craft”
  - not our focus
  - using scripts, regular expressions
  - many APIs available
    - BeautifulSoup, Selenium
  - *think continuous process – not once-off conversion!*

Semantic lifting

- Making structured data semantic
  - ...important for us
- Often the next step after scraping
  - ...or in parallel with scraping
  - storing the result in, e.g., RDF, RDFS, OWL...
- Tasks:
  1. creating triples *(make everything (s, p, o)-triples)*
  2. creating graphs *(one or several?)*
  3. selecting IRIs *(standard IRIs as identifiers)*
  4. selecting vocabularies *(standard IRIs as predicates)*
  5. selecting types *(standard IRIs as resource types)*
  6. external linking *(owl:sameAs)*

Extraction

- Retrieving RDF triples from (semantically) tagged resources
  - e.g., microformats, (eRDF ->) RDFa, Microdata
  - RDFLib parses RDFa and Microdata
- Replaces scraping + lifting
  - but is much simpler
    - the tags already do much of the job
  - open-source code is often available

Triple stores

- Basic software for persistent triple stores, or
  - database management systems (DBMSs) for RDF graphs
  - general DBMS properties and behaviours
  - a specialised type of *graph databases*
- Examples:
  - *Apache Jena TDB* (simple, file based, RDF-centric)
  - *Eclipse RDF4J (Sesame)* (much used, RDF-centric)
  - *Ontotext GraphDB (OWLIM)* (RDF4J compatible)
  - *Stardog* (RDF4J compatible)
  - *OpenLink Virtuoso* (much used, supports multiple data models, large)
  - *Blazegraph*...

https://www.w3.org/wiki/LargeTripleStores

Blazegraph

- Native triple store
  - SPARQL queries and updates, incl. federation
  - a simple web-based interface
  - multi-tenancy through namespaces
- Three modes for namespaces
  - plain triples (RDF), optional inference
  - quads (no inference)
  - reification done right (RDR), optional inference
- Many, many options
- *Built to scale for data and processing*

Blazegraph

- Built around Bigdata, a BTree store
  - single-machine journal (< 50B triples)
  - replicated journal (scale-up queries)
  - federated journal (> 8 machines, scale-out data)
- Dual licensing: GPLv2 and commercial
- Used by Wikidata and others
  - dockers are available with nice GUI

Why different triple stores?

- A few important properties:
  - capacity (a trillion triples (norsk: “billion”, 10^{12}))
  - scaling, performance, security
  - SPARQL version (1.0, 1.1, Update, Federation)
  - functionality: inference, quads, reification...
  - SQL dependency, supports other data models?
  - license, price
  - in memory / on file / over DB
  - local server or cloud-hosted
  - single- / multi-thread and -server
  - reasoning, APIs
  - built-in SPARQL or other endpoints?

Visualisation

- APIs:
  - general GUI APIs
  - graph drawing/editing APIs
- Cloud based:
  - graph and general visualisers
  - e.g., embedded in web pages
  - often SPARQL-based
    - a SPARQL query extracts the dataset
    - the SELECTed variables are used to draw
      - graphs, bar charts, pie charts...
    - e.g., *http://mgskjaeveland.github.io/sgvizler/*

Endpoints

- Providing access to semantic resources over the net using standard protocols
  - typically HTTP, SPARQL, RDF, XML, JSON
  - based on
    - pure RDF resources, or
    - "wrapped" resources, e.g., relational databases
- May provide web interfaces for interactive use, e.g.,
  - the web-interface to *Blazegraph*
  - *SNORQL* (http://dbpedia.org/snorql/)
  - *Wikidata Query Service*
    - new and very nice
    - built on top of Blazegraph

Wrappers

- Wrapping existing structured data resources to present them as semantic resources
  - often relational data
    - but also, e.g., spreadsheets, XML, JSON
  - on-demand (live) semantic lifting
    - attributes/columns are mapped to predicates
  - read-only or read+update?
  - handwritten or wrapper software
    - e.g., D2RQ (http://d2rq.org)
  - wrapped resources can be used locally
    - or made accessible through an endpoint

Three-level architecture

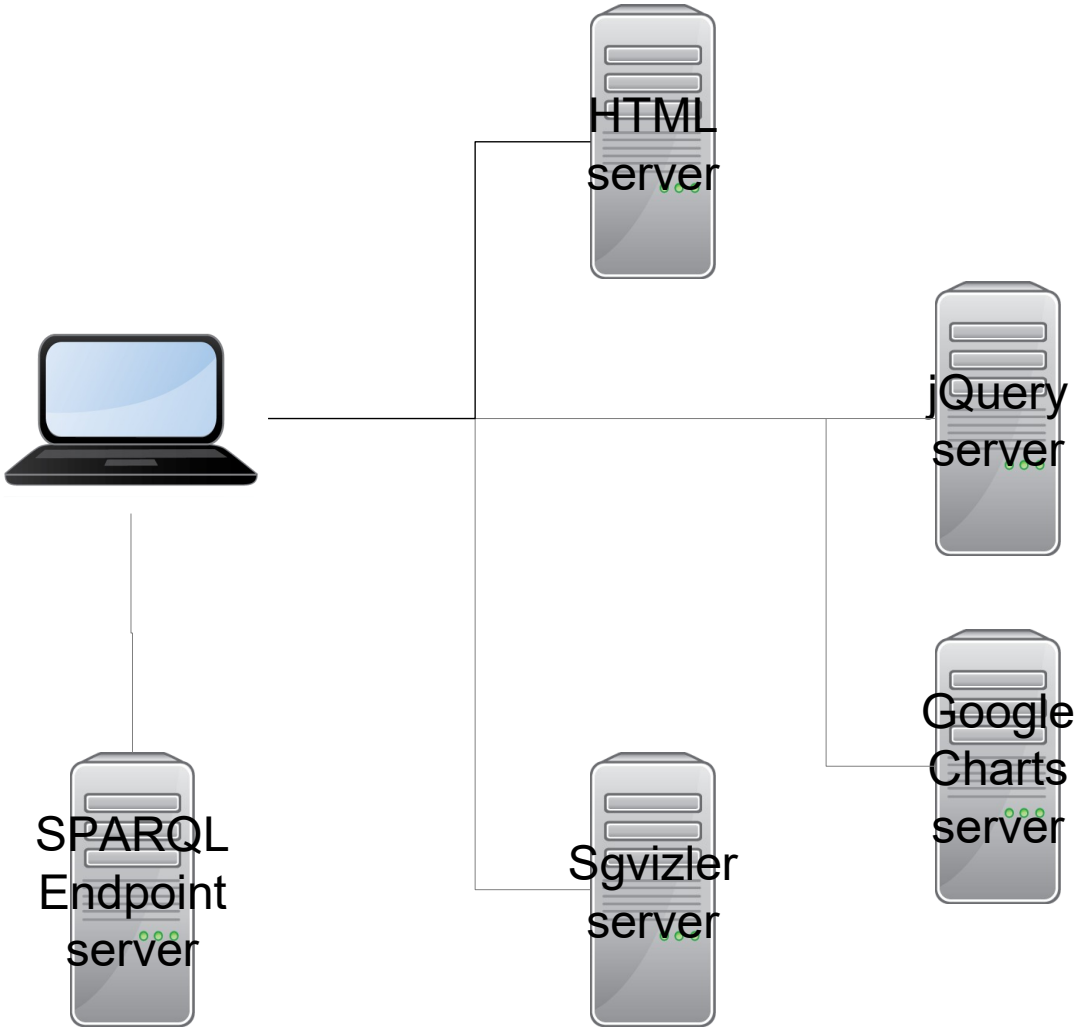
- Raw data sets:
  - available in a standard format
    - perhaps virtually
  - SPARQL end points, RDF files
- Abstract data representation (RDF):
  - graph of nodes and arrows
- Queries:
  - standard query languages
  - based on the abstract data representation
- *Enabled by the semantic technologies*

Sgvizler

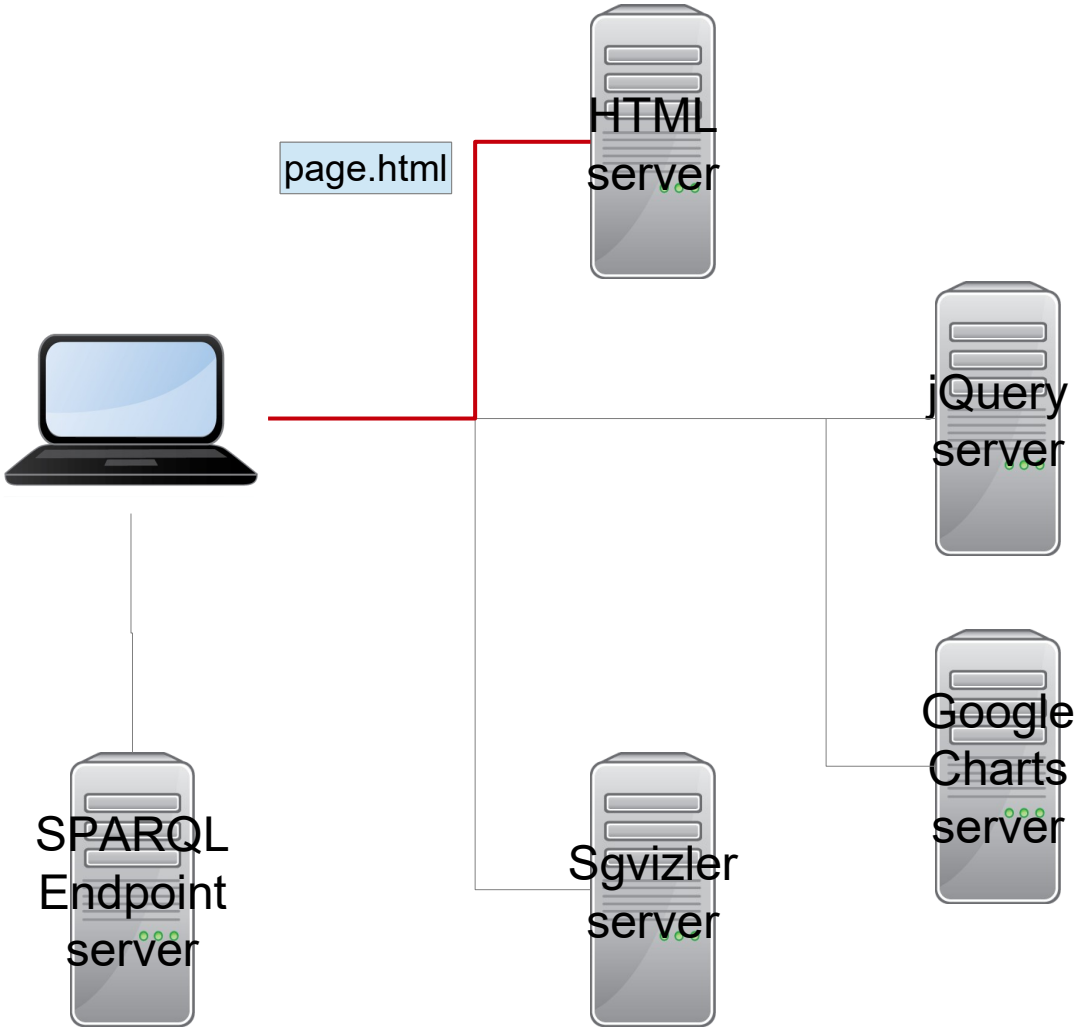
Sgvizler

- An open-source JavaScript library
  - for embedding in web pages
  - submits SELECT queries to SPARQL endpoints
  - transforms the (JSON or XML) results into data tables
  - visualises the data tables as charts or in other ways
  - the charts appear as part of the web pages
- SPARQL SELECT queries can be:
  - hard-coded into the HTML web page
  - input by the user through HTML forms (fully or partially)
  - invoked from JavaScripts in the page (Sgvizler's API)
- Based on
  - Google Charts/Visualisation API (or other similar, e.g., D3.js)
  - jQuery – JavaScript utility library

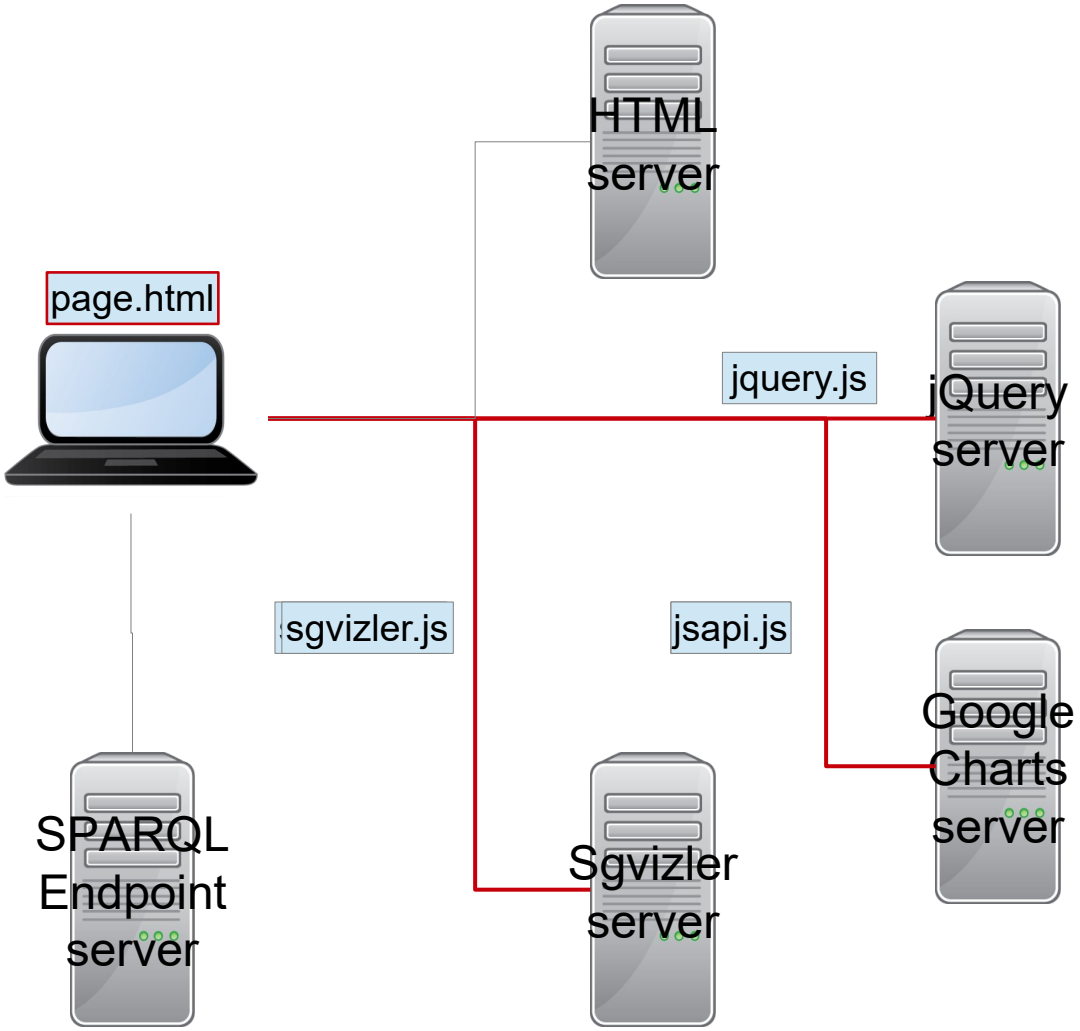
Sgvizler architecture



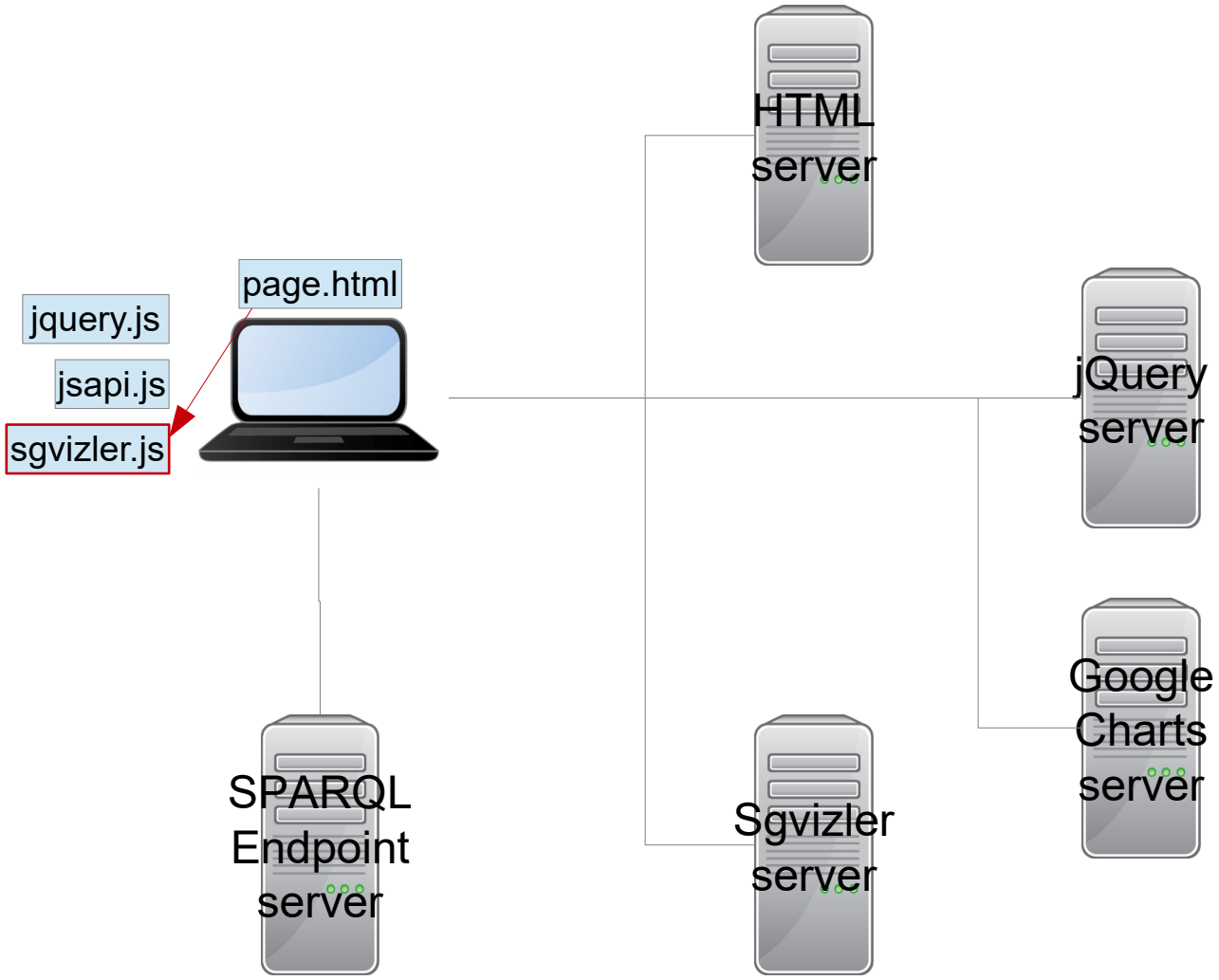
Sgvizler architecture



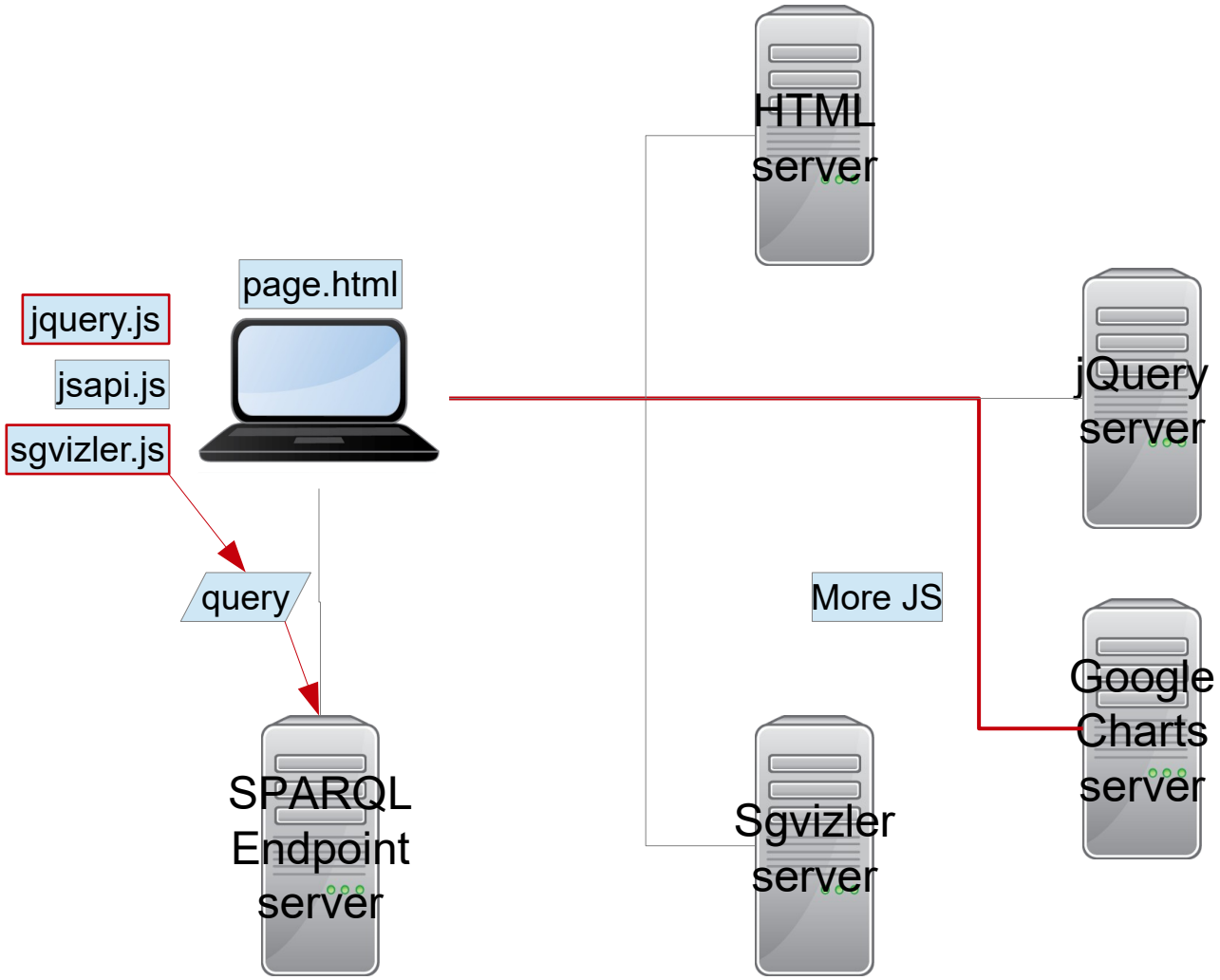
Sgvizler architecture



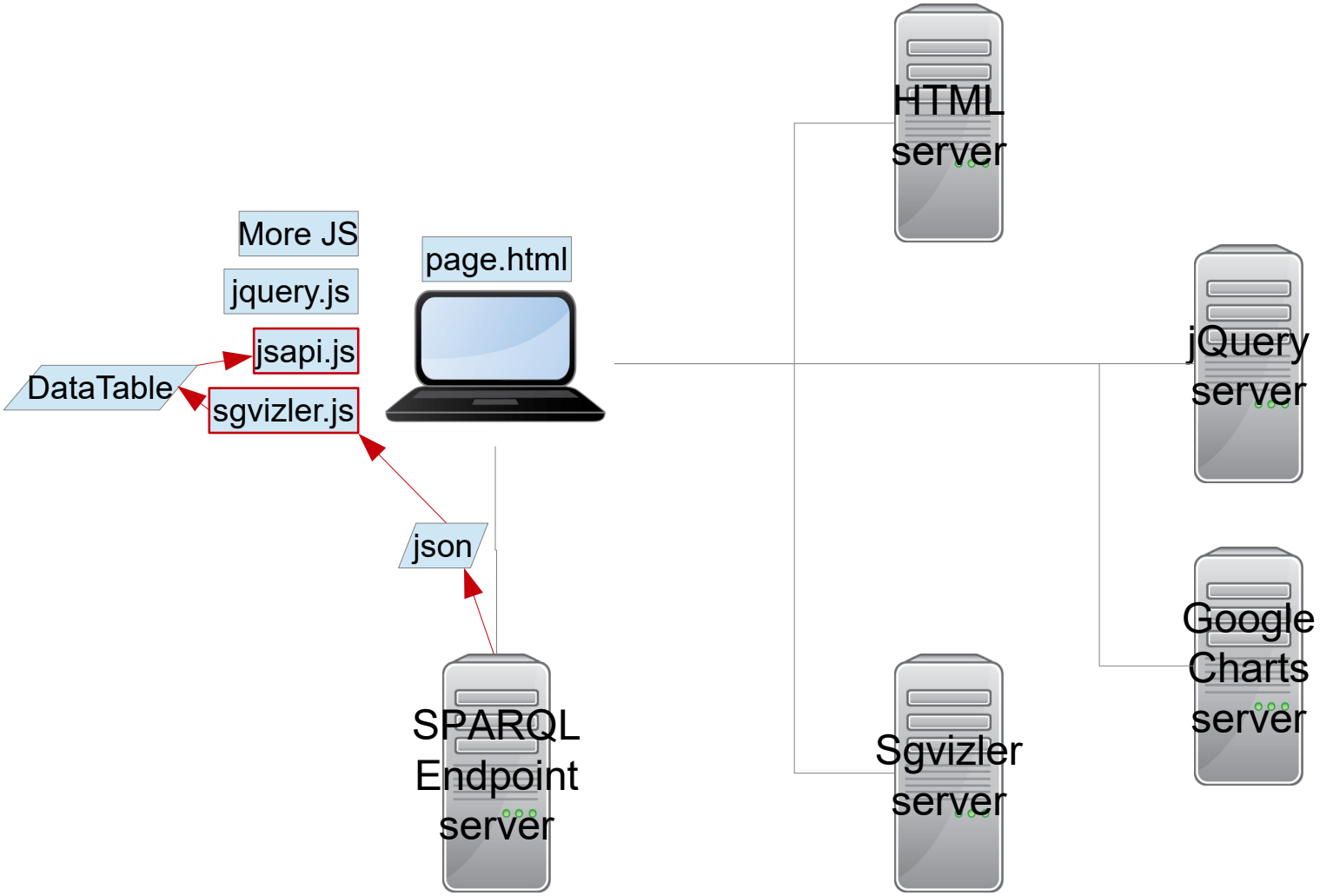
Sgvizler architecture



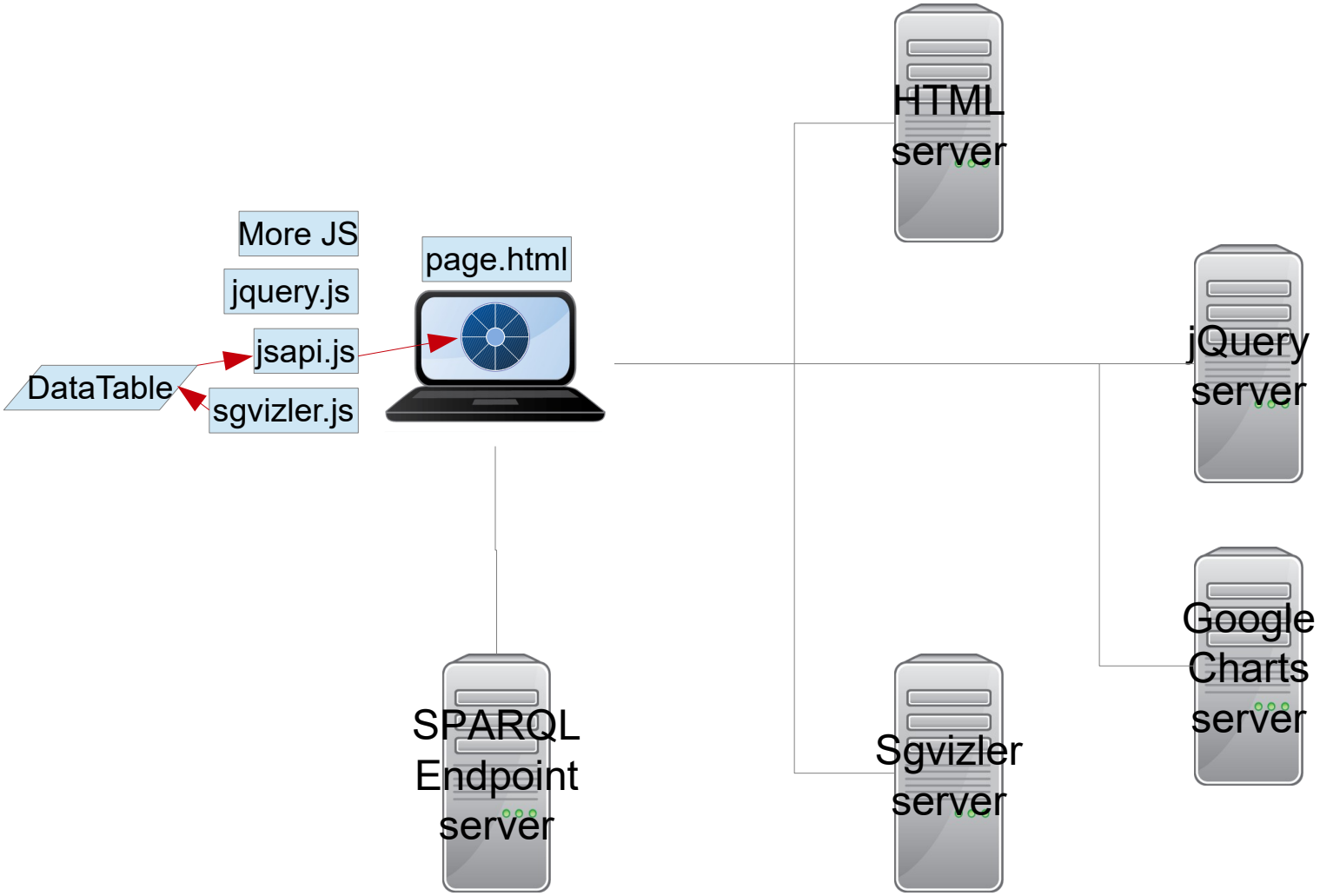
Sgvizler architecture



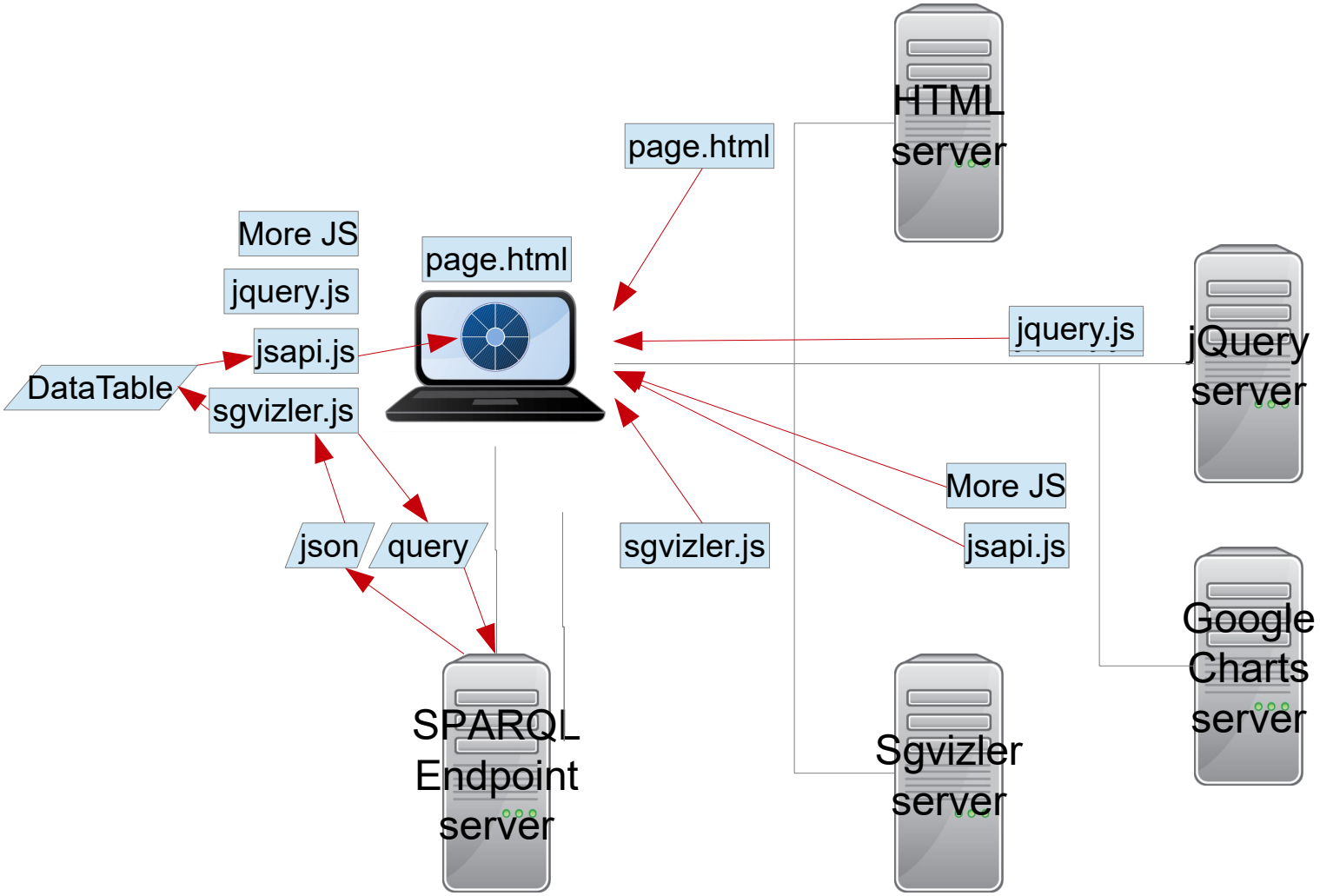
Sgvizler architecture



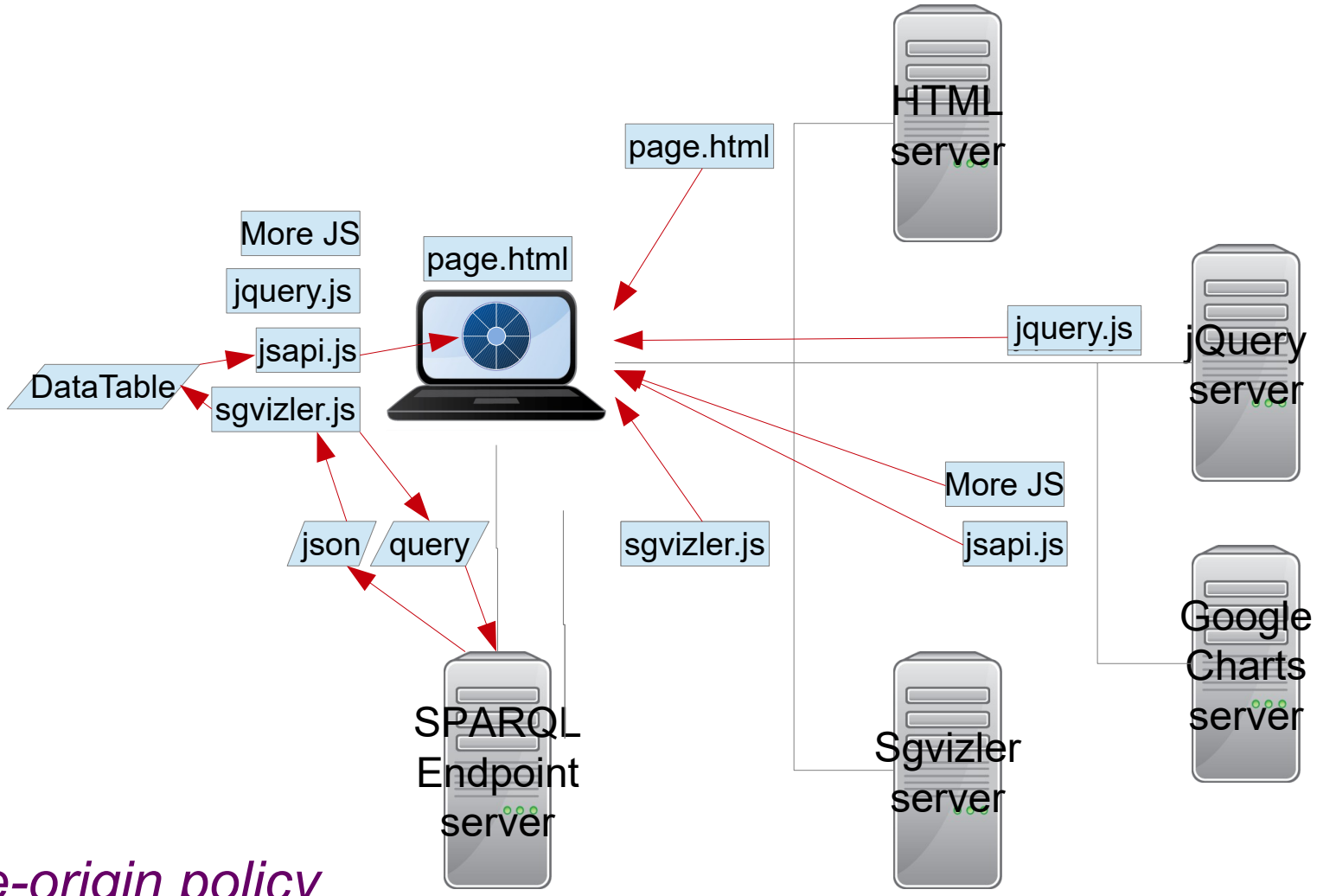
Sgvizler architecture



Sgvizler architecture



Sgvizler architecture



Same-origin policy may cause problems!

Sgvizler markup

```
<html>
   <head>
      <script type="text/javascript"
              src="http://cdnjs.cloudflare.com/ajax/libs/jquery/1.9.0/jquery.js"></script>
      <script type="text/javascript"
              src="https://www.google.com/jsapi"></script>
      <script type="text/javascript"
              src="http://mgskjaeveland.github.io/sgvizler/v/0.6/sgvizler.js"></script>
      <script>
         $(document).ready(
            function () { sgvizler.containerDrawAll();      }
         );
      </script>
   </head>
   <body>
      ...
   </body>
</html>
```

Sgvizler markup

```
<html>
   <head> ... </head>
   <body>
      ...

<div id="example"
    data-sgvizler-endpoint="http://sws.ifi.uio.no/sparql/npd"
    data-sgvizler-query="
                    SELECT ?class (count(?instance) AS ?noOfInstances)
                    WHERE{ ?instance a ?class }
                    GROUP BY ?class
                    ORDER BY ?class"
    data-sgvizler-chart="google.visualization.PieChart"
    style="width:800px; height:400px;"></div>

      ...
   </body>
</html>
```

Same-origin policy

- Part of the web application security model, in our case:
- *when a web resource (page) contains a script, the script can access a second web resource, but only if the two resources have the same origin, i.e., only if they have the same:*
  - *IRI scheme, hostname, and port number*
- But http://mgskjaeveland.github.io/sgvizler/v/0.6/sgvizler.js received JSON from http://sws.ifi.uio.no/sparql/npd – how come?
  - Cross-Origin Resource Sharing (CORS)
  - used JSONP (JSON with Padding)
  - co-locate your SPARQL endpoint with a copy of sgvizler.js
  - use federated queries for external data
- ...we will just use localhost (127.0.0.1) in the lab

Sgvizler through HTML forms

```
<html>
  <head>
    ... load jQuery, jsapi, and Sgvizler here ...
    <script>
        sgvizler
            .prefix("ex", "http://example.org#")
            .defaultEndpointURL("http://dbpedia.org/sparql")
            .defaultQuery("SELECT * { ?a ?b ?c ?d ?e } LIMIT 7")
            .defaultChartFunction("sgvizler.visualization.Table")
            .defaultChartWidth(500).defaultChartHeight(500);
    </script>
  </head>
  <body>
    <div id="myForm"></div>
    <script type="text/javascript">
      $(document).ready(function() { sgvizler.formDraw("myForm"); });
    </script>
  </body>
</html>
```

http://mgskjaeveland.github.io/sgvizler/example/usage-query-form.html

Sgvizler's API

```
<script>
var Q = new sgvizler.Query();                          // Create a Query instance.

// Values may also be set in the sgvizler object - but will be overwritten here.
Q.query("SELECT * {?s ?p ?o} LIMIT 10")
    .endpointURL("http://dbpedia.org/sparql")
    .endpointOutputFormat("json")                      // Either'xml', 'json', 'jsonp'.
    .chartFunction("google.visualization.Table")       // Function to draw the chart.
    .draw("myElementID");                              // Draw chart in HTML element.
</script>

<div id="myElementID"></div>
```

Google's Chart Tools

https://google-developers.appspot.com/chart/interactive/docs/