

Nothing is lost, nothing is created,
everything is `<xml:tranformed>`



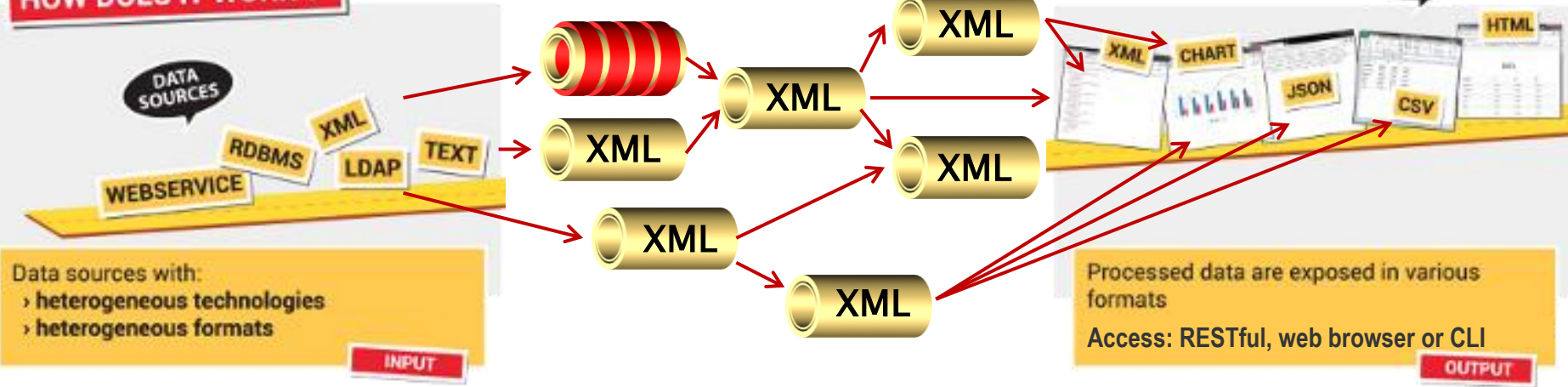
LAVOISIER

DATA AGGREGATION FRAMEWORK

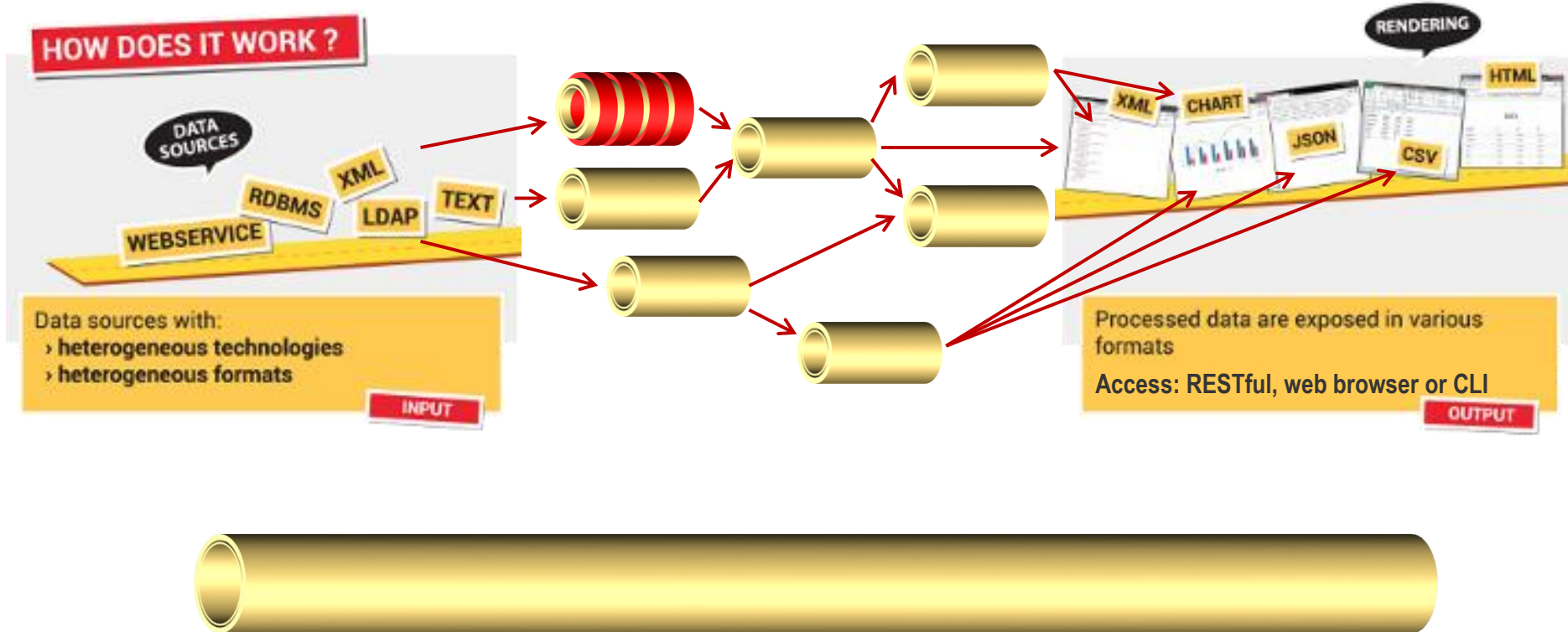
What is Lavoisier ?



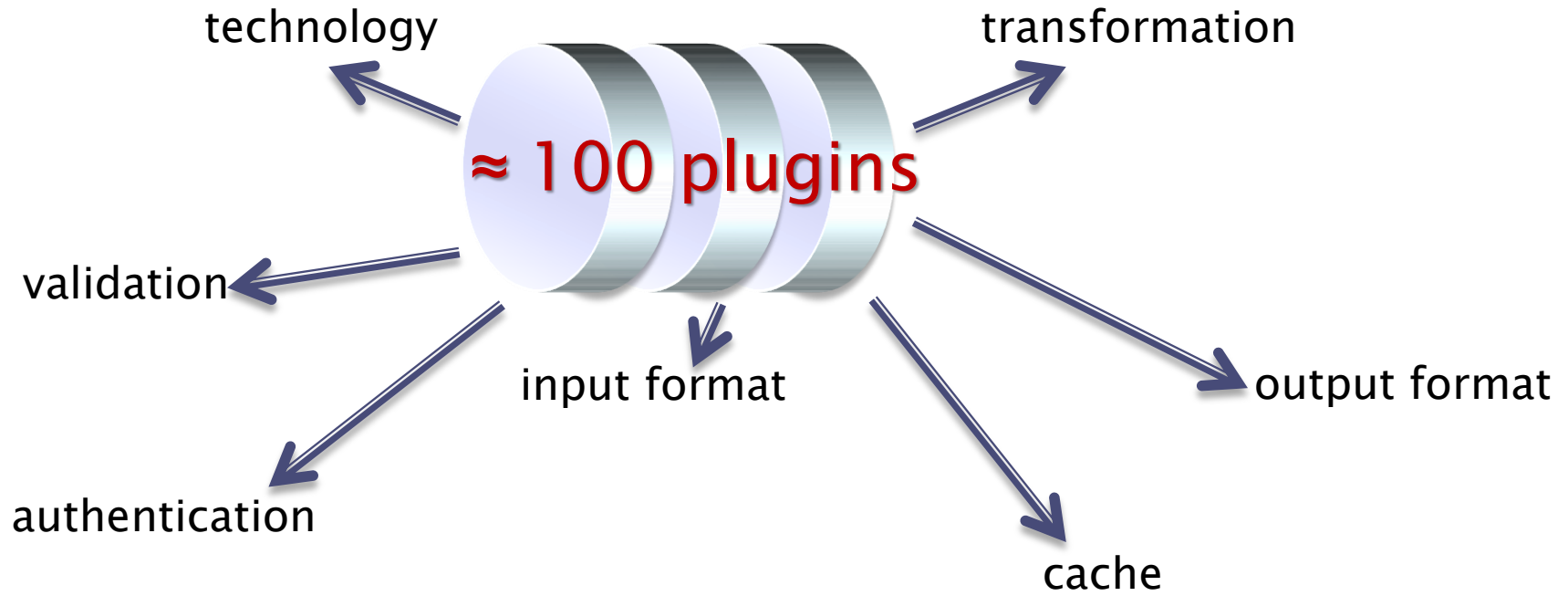
HOW DOES IT WORK ?



What is Lavoisier ?

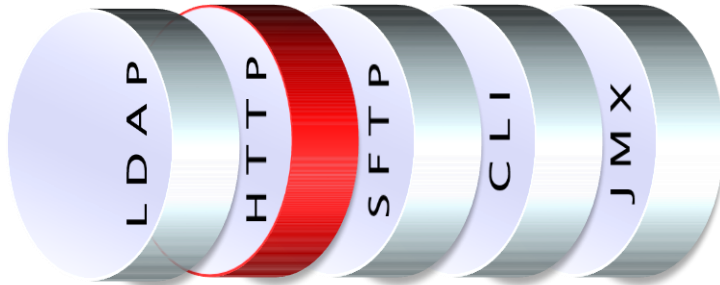


What is a data view ?

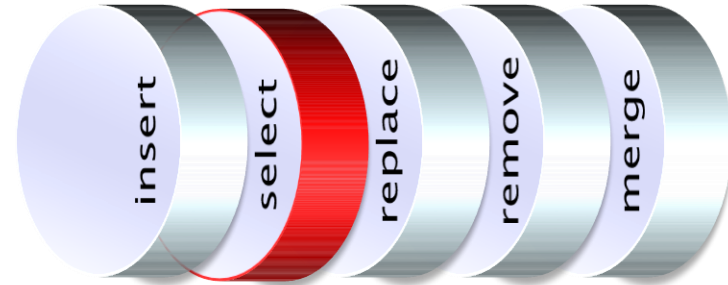


What is a data view ?

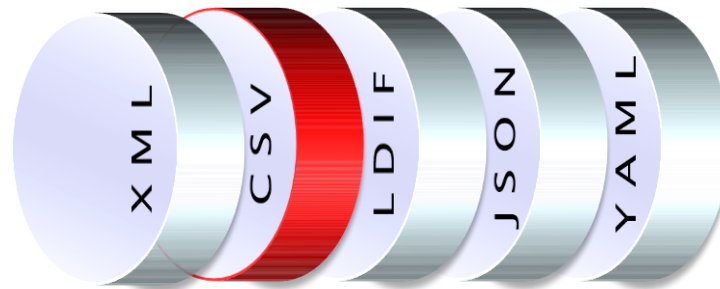
technology



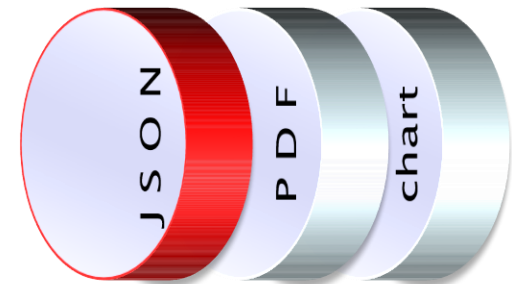
transformation



input format



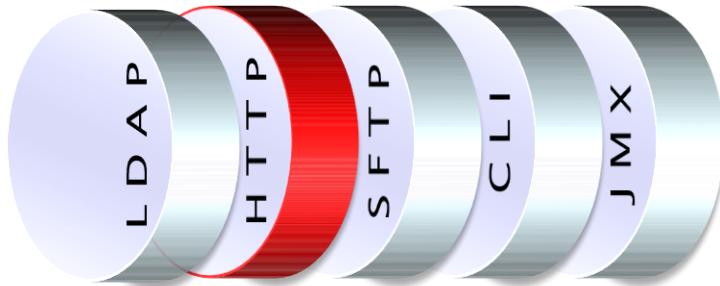
output format



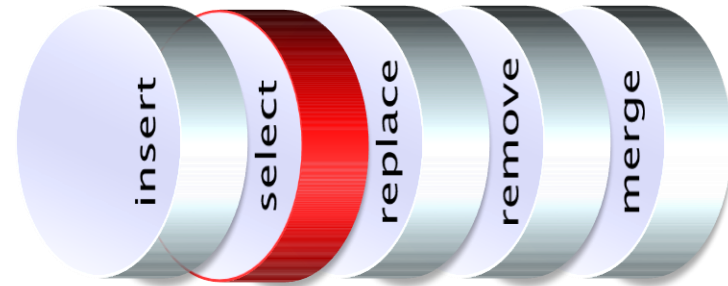
What is a data view ?

Example : download CSV data, extract column "Profit" and render to JSON

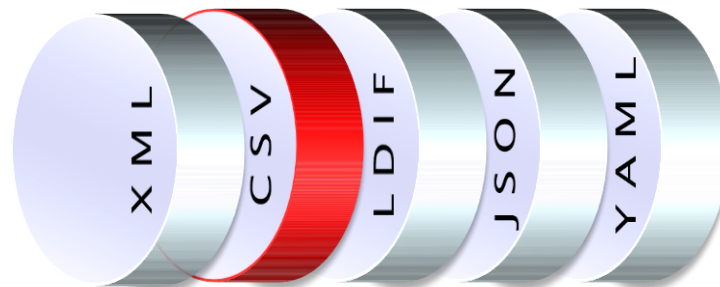
technology



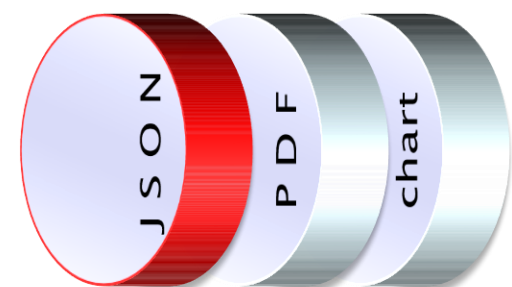
transformation



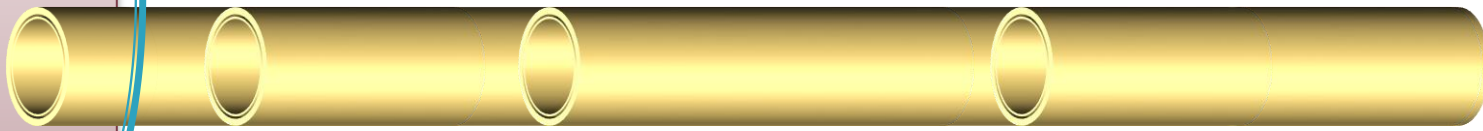
input format



output format



Month	Sales	Profit
March	28	89
April	432	1587
May	267	529
June	622	2103
...		



JSON

What is a data view ?

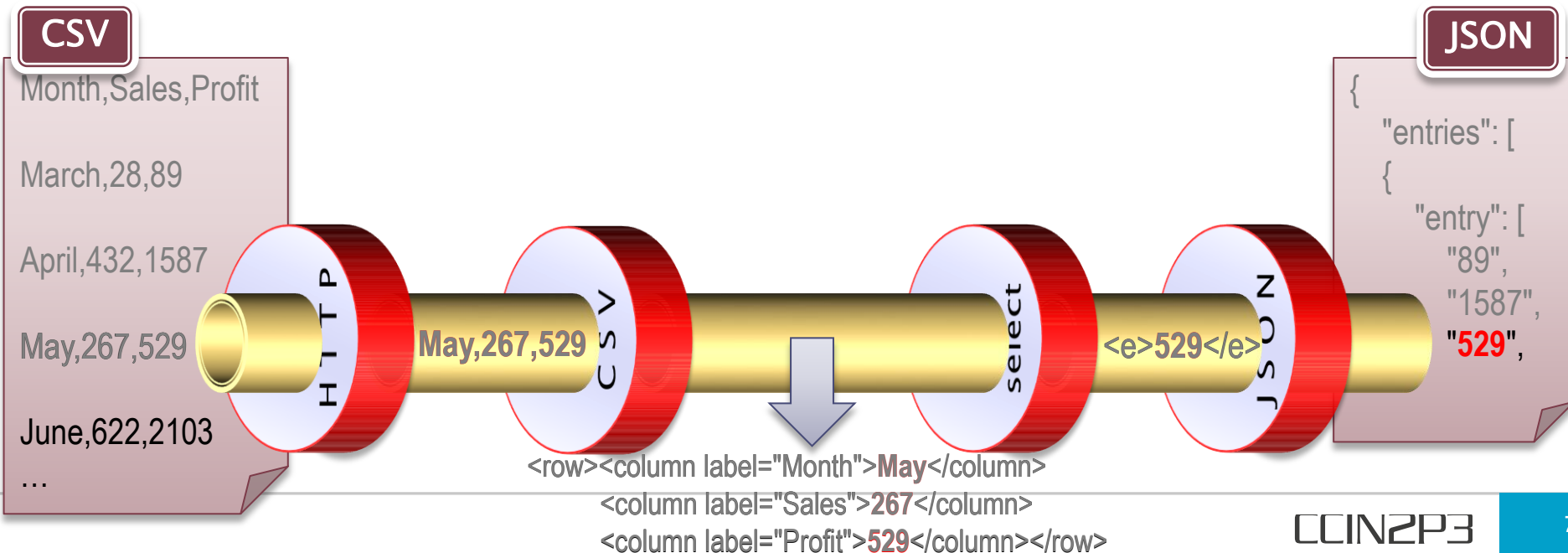
Example : download CSV data, extract column "Profit" and render to JSON

technology

transformation

input format

output format

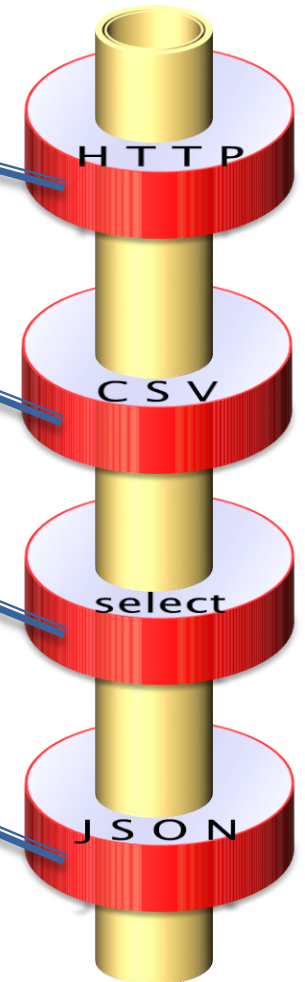


Example : download CSV data, extract column "Profit" and render to JSON

The screenshot shows a web browser window with the URL `ccsyre7:8080/lavoisier/debug/profit`. The page displays a configuration interface for three components:

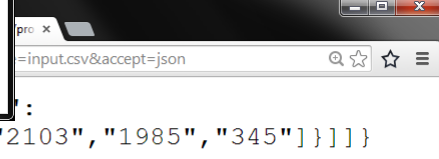
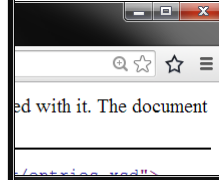
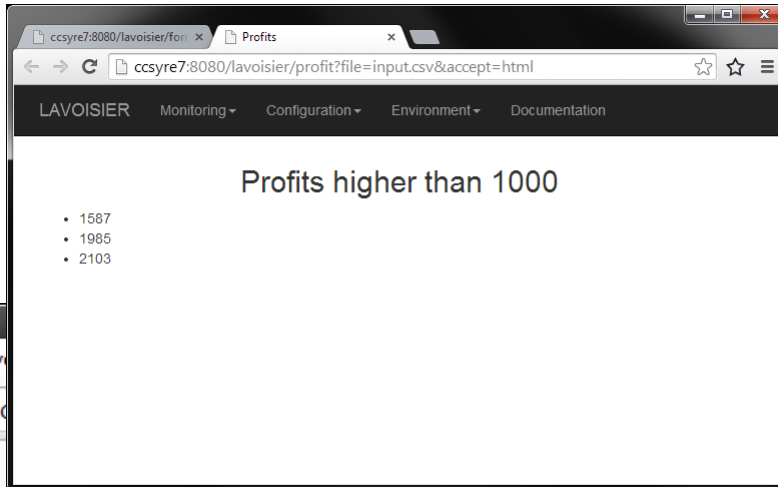
- HTTPConnector**:
 - url: `concat('http://software.in2p3.fr/lavoisier/', $file)`
- CSVSerializer**:
 - header: `true`
 - Rendering XML:

```
<rendering xmlns="http://software.in2p3.fr/lavoisier/rendering.xsd">
  <rows>
    <column_labels>
      <column_label label="Month">Month</column_label>
      <column_label label="Sales">Sales</column_label>
      <column_label label="Profit">Profit</column_label>
    </column_labels>
    <row>
      <column label="Month">March</column>
      <column label="Sales">28</column>
      <column label="Profit">89</column>
    </row>
  </rows>
</rendering>
```
 - Selected XPath: `/ns_0:rendering/ns_0:rows/ns_0:row/ns_0:column/@label`
- SelectProcessor**:
 - match: `/ns_0:rendering/ns_0:rows/ns_0:row/ns_0:column[@label='Profit']/text()`



Rendering

Example : download CSV data, extract column "Profit" and render to JSON



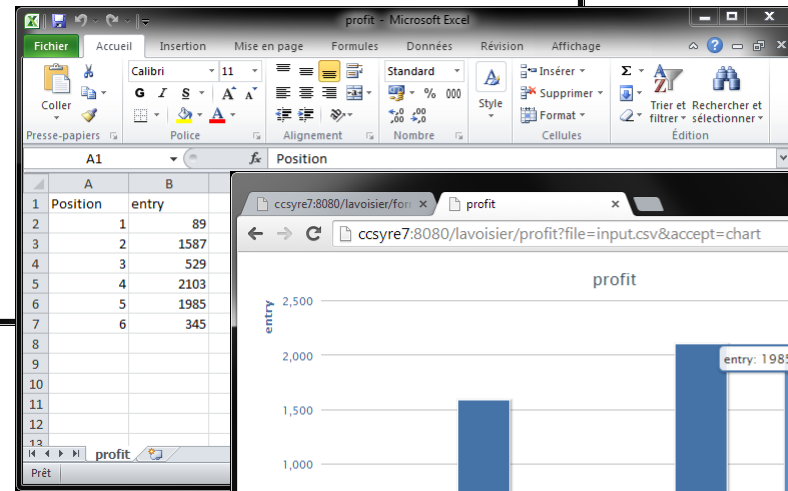
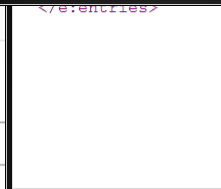
file input.csv

Rendering

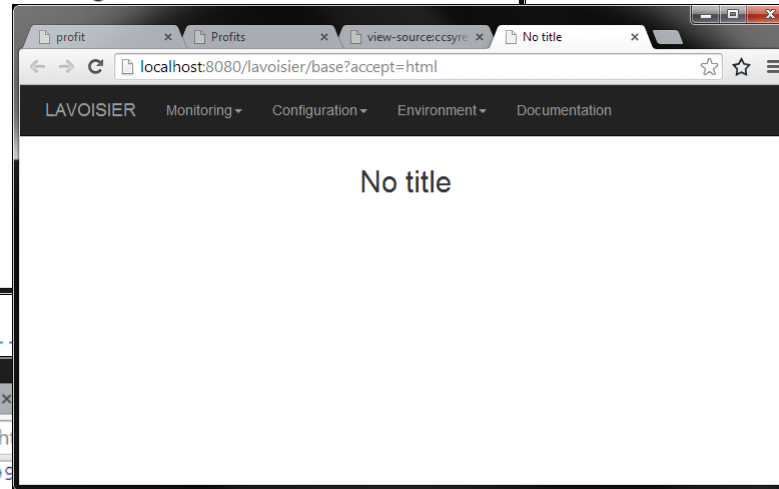
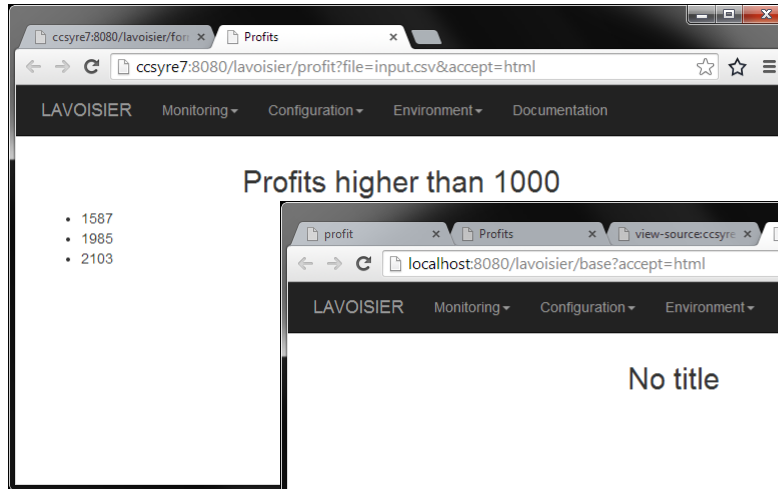
XPath filter

Range

- Default rendering
- Default rendering
- rowcol
- chart
- html
- gz
- pdf
- ldif
- shell
- zip
- json
- txt
- xml
- sql
- csv



Example : download CSV data, extract column "Profit" and render to JSON



Generated

```
1 <html xmlns="http://www.w3.org/1999/xhtml" xmlns:e="http://software.in2p3.fr/lavoisier/entries.xsd"
2     xmlns:tpl="http://software.in2p3.fr/lavoisier/template.xsd"
3     tpl:extends="http://localhost:8080/lavoisier/base?accept=html">
4 <head>
5   <title>Profits</title>
6 </head>
7 <body>
8   <div id="content" tpl:variable.threshold="1000">
9     <h2 id="title">Profits higher than {{ $threshold }}</h2>
10    <ul>
11      <li tpl:foreach="/e:entries/e:entry[text()> $threshold]"
12          tpl:with-order="text()">{{ text() }}</li>
13    </ul>
14  </div>
15 </body>
16 </html>
```

HTML Template

- XML/XPath support
- Client-side execution
- "Natural templating"
- Inheritance

Data aggregation

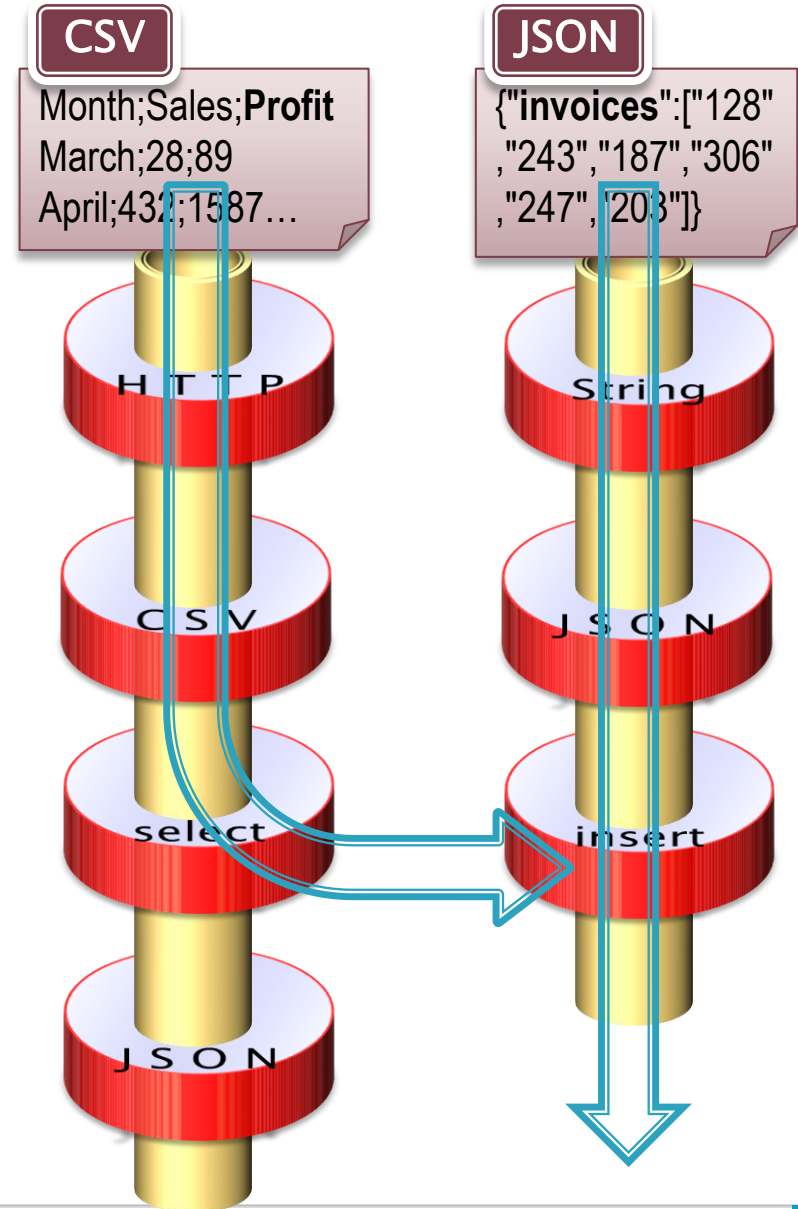
Example : aggregate profit with number of invoices for each month

The left side of the slide shows a sequence of data processing steps:

- XML View:** A browser window showing an XML document with a message: "This XML file does not appear to have any style information associated with it. The document tree is shown below." The tree structure is:

```
<object>
  <invoices>
    <item profit="89">128</item>
```
- JSON View:** A browser window showing the same data as JSON:

```
{
  "object": {
    "invoices": [
      {
        "profit": "89",
        "#text": "128"
      },
      {
        "profit": "1587",
        "#text": "243"
      },
      {
        "profit": "529",
        "#text": "187"
      },
      {
        "profit": "2103",
        "#text": "306"
      }
    ]
  }
}
```
- Excel View:** A Microsoft Excel spreadsheet with columns 'profit' and 'invoices'. The data rows are: (89, 128), (1587, 243), (529, 187), (2103, 306).
- Chart View:** A browser window showing a combined bar and line chart. The x-axis represents months, the left y-axis represents profit (0 to 2,500), and the right y-axis represents the number of invoices (100 to 350). Blue bars represent profit, and a red line represents the number of invoices.



content

```
{"invoices":["128","243","187","306","247","203"]}
```

JSONSerializer



InsertProcessor



match

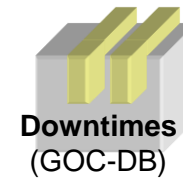
```
1 /object/invoices/item
```

nodes

```
1 new_attribute('profit', view('profit')/**[position()=position(match())])
```

```
<object>
  <invoices>
    <item profit="89">128</item>
    <item profit="1587">243</item>
    <item profit="529">187</item>
    <item profit="2103">306</item>
    <item profit="1985">247</item>
    <item profit="345">203</item>
  </invoices>
</object>
```

Click on XML to get XPATH



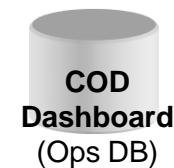
▶ Previous example

- helps to understand how to use Lavoisier
- but Lavoisier is overkill for this !

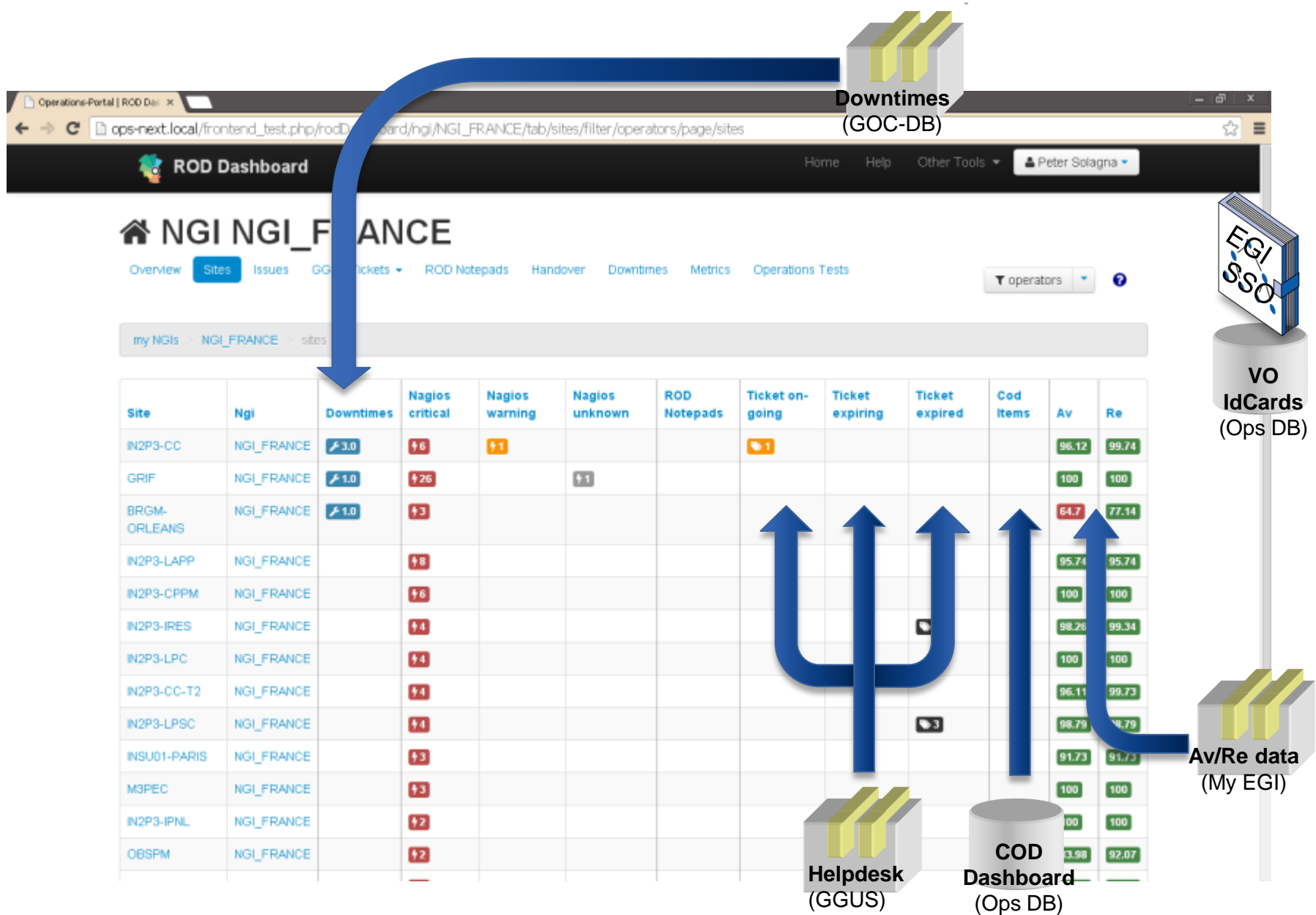


▶ A real use-case : the EGI Operations Portal

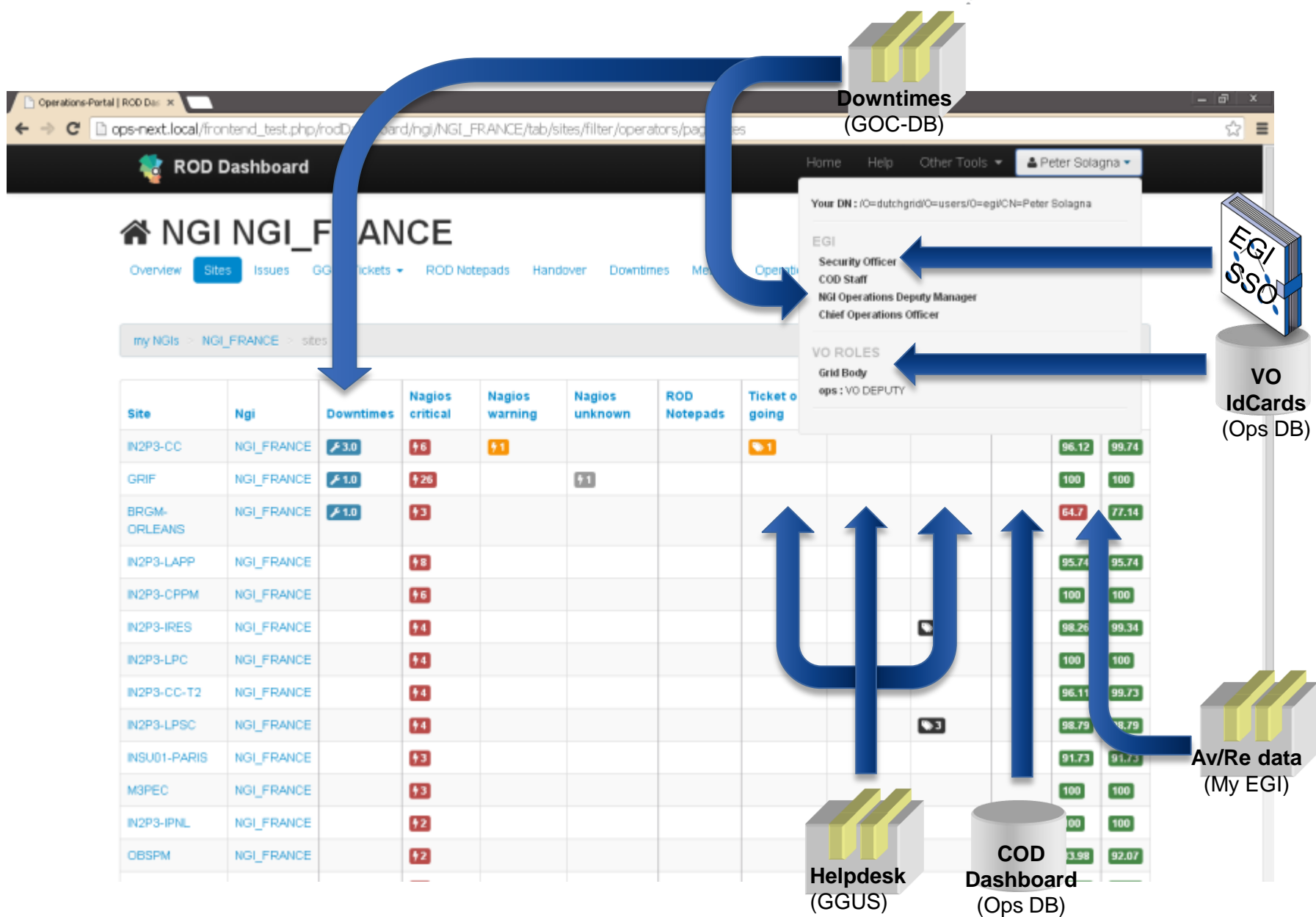
- 15 heterogeneous data sources
- 100 data views
- 2000 users



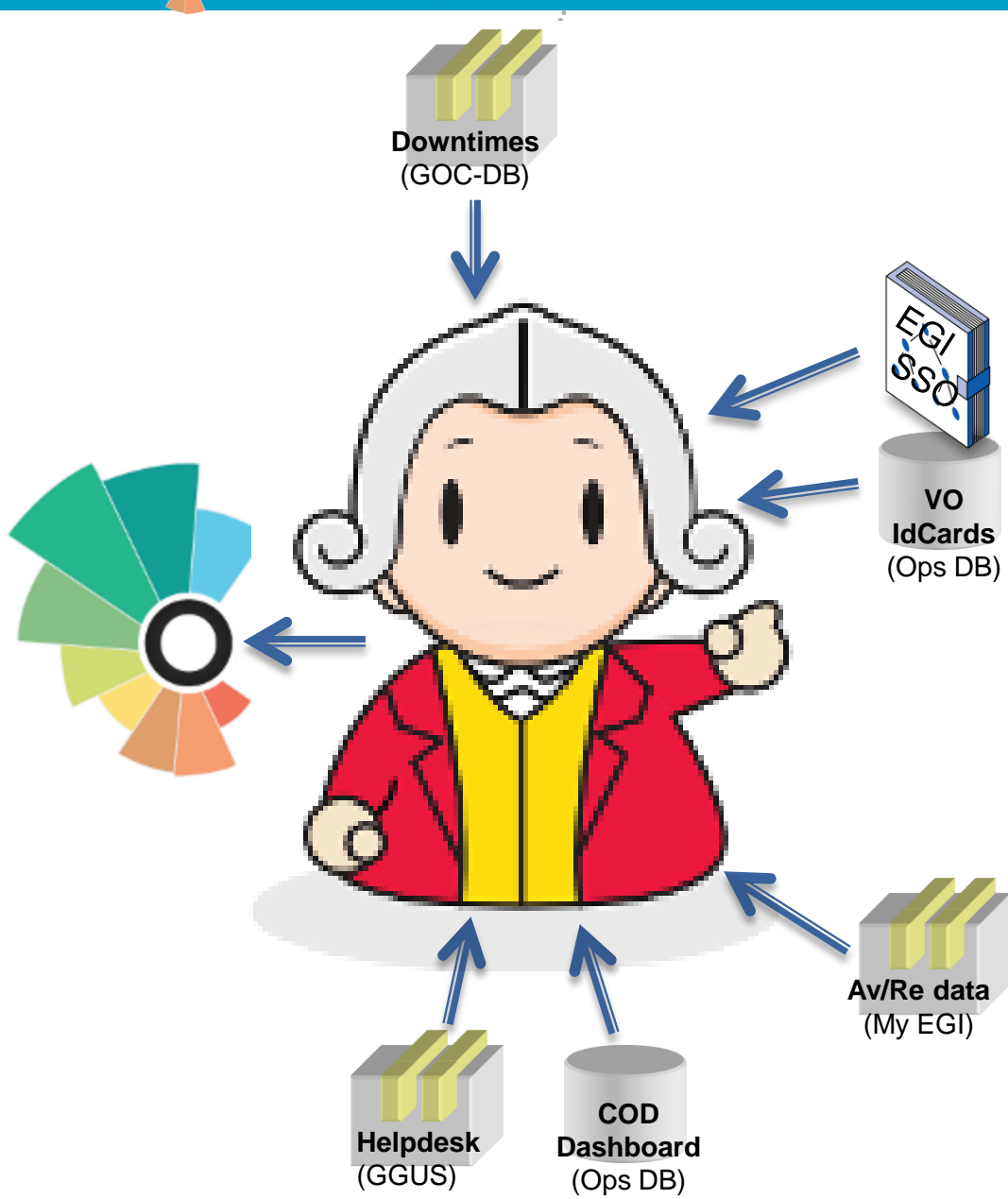
A real example :



A real example :



- Maintainability**
 - Modularity
 - Single language
- Performance**
 - Memory usage
 - Latency
- Availability**
 - Robustness
 - Monitorability
- Security**
 - Authentication
 - Authorization



Maintainability

- **Modularity**
- **Single language**

Performance

- **Memory usage**
- **Latency**

Availability

- **Robustness**
- **Monitorability**

Security

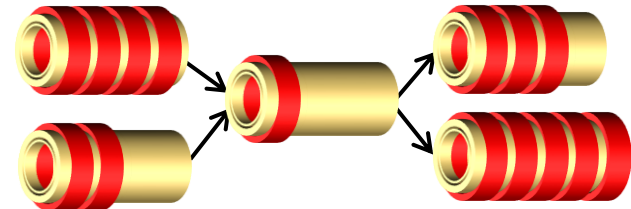
- **Authentication**
- **Authorization**

Ensured by the configuration language

- ▶ **technology/format integration**
 - view = **chain of plugins**



- ▶ **business code**
 - application = **graph of data views**



Maintainability

- Modularity
- Single language

Performance

- Memory usage
- Latency

Availability

- Robustness
- Monitorability

Security

- Authentication
- Authorization

Single language (XPath) to :

- ▶ Define authorization rules
 - `user()='127.0.0.1'`
- ▶ Set parameter dynamically
 - `concat('http://host:8080/lavoisier/', $file)`
- ▶ Select parts of the data view
 - `//row/column[@label='Profit']/text()`
- ▶ Interact with
 - user request : `arguments(), path()`
 - environment : `property(), document()`
 - views : `view(), match()`
- ▶ Configure data rendering
 - row/columns
 - HTML template



Maintainability

- Modularity
- Single language

Performance

- Memory usage
- Latency

Availability

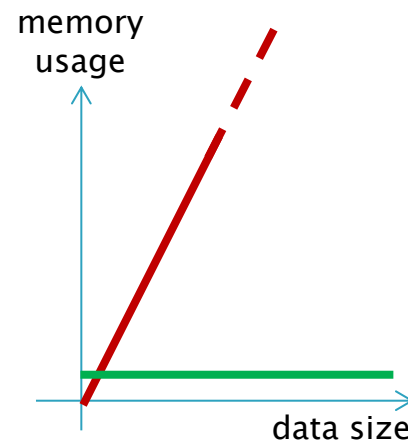
- Robustness
- Monitorability

Security

- Authentication
- Authorization

▶ Plugins process streams

- of bytes
- of XML events



▶ XPath optimizer

- avoid building big data structures
 - `/absolute/xpath` is transformed to a set of instructions processing the stream
 - `./relative/xpath` is evaluated on small data structures
- detect factorizable function calls

Transparent to developers

Maintainability

- Modularity
- Single language

Performance

- Memory usage
- Latency

Availability

- Robustness
- Monitorability

Security

- Authentication
- Authorization

Configure the cache mechanism:

▶ Per data view

- choose cache plugin
- combine refresh triggering rules
 - when view is notified by user
 - when dependency is refreshed
 - when time is elapsed
 - ...

▶ According to the constrains of

- data : size, time-to-live, dependencies
- technology : latency, throughput, availability
- users: patience, access frequency

Transparent to end-users

Maintainability

- Modularity
- Single language

Performance

- Memory usage
- Latency

Availability

- Robustness
- Monitorability

Security

- Authentication
- Authorization

Web console features

- ▶ **Monitoring**
 - Views build status, statistics...
- ▶ **Configuration**
 - Data views inter-dependencies
 - XPath expression optimizations
- ▶ **Environment**
 - OS, JVM, service...
- ▶ **Debugging**
 - Breakpoint on a XML stream
 - Breakpoint on a plugins chain
 - Skeleton of a data view

Developed with Lavoisier language

Maintainability

- Modularity
- Single language

Performance

- Memory usage
- Latency

Availability

- Robustness
- Monitorability

Security

- Authentication
- Authorization

Define security role per data view

- ▶ Authentication plugins
 - Chainable
 - Extensible
 - CAS, password, X509, IP, OAuth
 - Based on JAAS standard
- ▶ Authorization rules (XPath)
 - For each authentication plugin
 - Depending on argument value
`view('authZ')/*/*[@user=user() and @key=$arg]`

Maintainability

- Modularity
- Single language

Performance

- Memory usage
- Latency

Availability

- Robustness
- Monitorability

Security

- Authentication
- Authorization

Lavoisier does all this for you...



...enabling you to focus on
business code !

- ▶ New plugins
- ▶ On-the-fly partial reconfiguration of the framework
- ▶ GUI for building your own data aggregation application

The screenshot displays the Lavoisier Editor interface. The top navigation bar includes 'Home', 'Link', and 'Server'. The main workspace is divided into several panels:


- Configuration tree:** A hierarchical tree on the left showing a 'demo' project with sub-items like 'parking', 'parking_lyon', 'HTTP', 'Zip', 'Insert' (highlighted), 'File', 'renderers', and 'default'.
- Edit:** The central panel shows the configuration for an 'InsertProcessor'. It contains a table with the following data:

Name	Type	Eval	Value
match	String	<input checked="" type="checkbox"/>	1 /kml:kml/kml:Document/kml:Placemark/kml:Point/kml:coordinates[text()]
destination_as	String	<input type="checkbox"/>	last_child
nodes *	XPath	<input checked="" type="checkbox"/>	new_attribute('lng', substring-before(text(),',')) new_attribute('lat', substring-after(text(),','))

Below the table is a 'save' button.
- Configuration graph:** A vertical stack of components on the right, enclosed in a dashed box. From top to bottom: 'parking_lyon' (text), 'HTTP' (blue box), 'Zip' (green box), 'Insert' (grey box), 'File' (orange box), 'distance' (text), 'HTTP' (blue box), and 'Remove' (grey box).
- Shared Components:** A tree on the bottom left showing 'post-processors chains', 'authenticators chains', 'administrator', 'IPAddress', and 'user'.
- Input/Output:** A text area on the bottom right containing XML code:

```
<kml xmlns="http://www.opengis.net/kml/2.2">
  <Document>
    <Name>Parking</Name>
    <Style id="Style_@_pvo_patrimoine_voirie.pvoparking">
      <IconStyle>
        <scale>0.5</scale>
      <Icon>
        <href>https://download.data.grandlyon.com/kml//grandlyon/?request=icon&typename=pvo_patrimoine_vo
```

Lavoisier 2 - x
software.in2p3.fr/lavoisier/

 **LAVOISIER**
DATA AGGREGATION FRAMEWORK

CCIN2P3

Last Published: 2014-05-06 | Version: 2.1.1-SNAPSHOT CNRS | CC-IN2P3

LAVOISIER

- Project home
- Features
- Download
- Release notes
- Submit a bug
- END USERS DOC
- Install guide
- User guide
- ADVANCED USERS DOC
- Language guide
- Language reference
- Adaptors reference
- Configuration example

Nothing is lost, nothing is created, all is `<xml:transformed>`

Lavoisier is a framework which enables to retrieve, transform, merge and query heterogeneous data sources.

HOW DOES IT WORK ?

DATA SOURCES
WEBSERVICE, RDBMS, XML, LDAP, TEXT

LAVOISIER

RENDERING
XML, CHART, JSON, CSV, HTML

INPUT
Data sources with:
- heterogeneous technologies
- heterogeneous formats

AGGREGATION
Input data are converted to a common representation (XML), and then processed by plugins.

OUTPUT
Processed data are exposed in various formats through a RESTful web service.
<http://host:8080/lavoisier/example?accept=text/plain>

The resulting data is exposed as XML views through a RESTful web service interface (see [User guide](#)).

These XML data views are configured by defining chains of plugins (see [DataView Construction Language](#)).