AtomServer : The Power of Publishing for Data Distribution

Chris Berry & Bryon Jacob

- Basics of **Atom** (the **Atom Syndication Format** and the **Atom Publishing Protocol**)

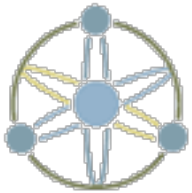
- **Atom Stores**, and how to use them in a Data Distribution Platform

- **AtomServer** – our open-source implementation of an Atom Store

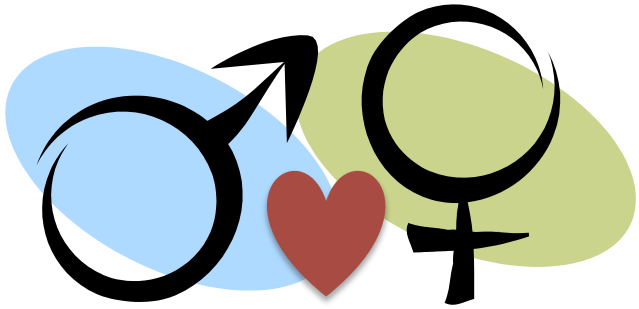
- How Atom and **AtomServer** fit into **SOA**

ONLINE DATING

+



AtomServer



We want to be the place for you to meet your mate – no matter who you are!

We've want grow our business quickly – so we've acquired several popular "niche" dating sites…

LoveOnTwoWheels.com


WineSnob
Match.com

LARPers
InLove.com

Integration plan…

We have four distinct platforms, and three operating businesses with existing customers.

We want to integrate everything for economies of scale, and to get “critical mass” on our newly launched site.

First, we need to assess what we have…

- Chris and I built this from scratch
- Java
- PostgreSQL DB
- Linux

- Hired a Microsoft Certified developer
- C# / .NET / ASP
- SQL Server DB
- Windows Server

- Owners built the site themselves between WOW sessions
- PHP
- MySQL DB
- Linux

LoveOnTwoWheels.com

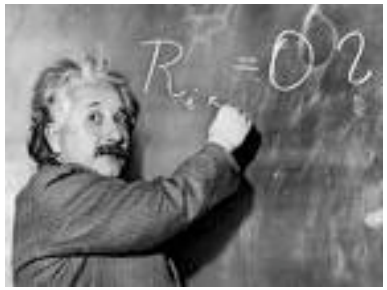
- Built by owner's uncle who once shared a cab with Steve Wozniak in the 70's
- COBOL / FORTRAN / INTERCAL
- Flat-text files
- On a TRS-80
- In the owner's closet
- With a 2400 baud modem

AtomServer

- This kind of integration is exactly the reason we built AtomServer…
- The “agree to disagree” model
- Atom is a RESTful protocol – which means:
  - If you can speak XML…
  - And you can speak HTTP…
  - Then you can access and manipulate Atom data.
  - If we’re talking about a web application – then the answer is already YES to both of these questions!

Business Objectives

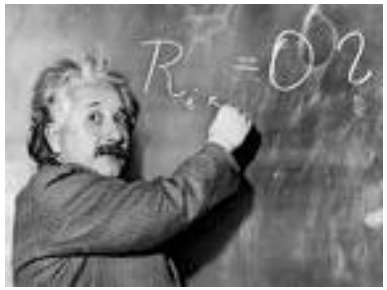
- Get our main site to critical mass, by combining data from our acquisitions
- Increase revenue by “cross-selling” customers between our sites
- Simplify our business processes with a SOA that’s streamlined by Atom and AtomServer



Professor Atom Says:

Atom Basics

- What we commonly call Atom is actually TWO separate specs
  - Atom Syndication Format.
  - Atom Publishing Protocol
- Together they provide a RESTful protocol for interacting with time-stamped data



Professor Atom Says:

Atom Syndication Format

- Provides a basis for a web syndication framework
  - Where syndication == simultaneous publication
- Originally conceived to replace RSS
- Simply an XML vocabulary to describe
  - feeds
  - entries
  - content

Atom XML conveys the semantics of publishing.

```
<feed xmlns=http://www.w3.org/2005/Atom xmlns:as="http://
   atomserver.org/namespaces/1.0/">
  <as:endIndex>7</as:endIndex>
  <author><name>AtomServer APP Service</name></author>
  <title type="text">dogs entries</title>
  <updated>2008-10-06T23:22:27.073Z</updated>
  <entry>
    <id>/atomserver/v1/pets/dogs/fido.xml</id>
    <title type="text"> Entry: dogs fido</title>
    <author><name>AtomServer APP Service</name></author>
    <link href="/atomserver/v1/pets/dogs/fido.xml/3"
   rel="edit" />
    <as:entryId>fido</as:entryId>
    <as:updateIndex>27</as:updateIndex>
    <as:revision>2</as:revision>
    <updated>2008-10-06T23:22:27.071Z</updated>
    <published>2008-10-06T23:22:27.071Z</published>
    <category scheme="urn:pets.owner" term="123" />
    <link href="/pets/dogs/fido.xml" >
</entry>
  <entry>
    <id>/atomserver/v1/pets/dogs/sparky.xml</id>
    .....
```

In Atom, lists are *feeds*

```
<feed xmlns=http://www.w3.org/2005/Atom xmlns:as="http://atomserver.org/
    namespaces/1.0/">
  <as:endIndex>7</as:endIndex>
  <author><name>AtomServer APP Service</name></author>
  <title type="text">dogs entries</title>
  <updated>2008-10-06T23:22:27.073Z</updated>
  <id>tag:atomserver.org,2008:v1:dogs</id>
  <entry>
    <id>/atomserver/v1/pets/dogs/fido.xml</id>
    <title type="text"> Entry: dogs fido</title>
    <author><name>AtomServer APP Service</name></author>
    <link href="/atomserver/v1/pets/dogs/fido.xml/3" rel="edit" />
    <as:entryId>fido</as:entryId>
    <as:updateIndex>27</as:updateIndex>
    <as:revision>2</as:revision>
    <updated>2008-10-06T23:22:27.071Z</updated>
    <published>2008-10-06T23:22:27.071Z</published>
    <category scheme="urn:pets.owner" term="123" />
    <content type="application/xml">
      <pet xmlns="http://schemas.atomserver.org/pets/v1/rev0" name="fido"
   type="dog">
        <breed>Mixed</breed>
        <owner>123</owner>
      </pet>
    </content>
  </entry>
  <entry>
    <id>/atomserver/v1/pets/dogs/sparky.xml</id>
    .....
```

The items in the list are *entries*

```
<feed xmlns=http://www.w3.org/2005/Atom xmlns:as="http://atomserver.org/
    namespaces/1.0/">
  <as:endIndex>7</as:endIndex>
  <author><name>AtomServer APP Service</name></author>
  <title type="text">dogs entries</title>
  <updated>2008-10-06T23:22:27.073Z</updated>
  <id>tag:atomserver.org,2008:v1:dogs</id>
  <entry>
    <id>/atomserver/v1/pets/dogs/fido.xml</id>
    <title type="text"> Entry: dogs fido</title>
    <author><name>AtomServer APP Service</name></author>
    <link href="/atomserver/v1/pets/dogs/fido.xml/3" rel="edit" />
    <as:entryId>fido</as:entryId>
    <as:updateIndex>27</as:updateIndex>
    <as:revision>2</as:revision>
    <updated>2008-10-06T23:22:27.071Z</updated>
    <published>2008-10-06T23:22:27.071Z</published>
    <category scheme="urn:pets.owner" term="123" />
    <content type="application/xml">
      <pet xmlns="http://schemas.atomserver.org/pets/v1/rev0" name="fido"
   type="dog">
        <breed>Mixed</breed>
        <owner>123</owner>
      </pet>
    </content>
  </entry>
  <entry>
    <id>/atomserver/v1/pets/dogs/sparky.xml</id>
    .....
```

Entries contain *content*

```
<feed xmlns=http://www.w3.org/2005/Atom xmlns:as="http://atomserver.org/
    namespaces/1.0/">
  <as:endIndex>7</as:endIndex>
  <author><name>AtomServer APP Service</name></author>
  <title type="text">dogs entries</title>
  <updated>2008-10-06T23:22:27.073Z</updated>
  <id>tag:atomserver.org,2008:v1:dogs</id>
  <entry>
    <id>/atomserver/v1/pets/dogs/fido.xml</id>
    <title type="text"> Entry: dogs fido</title>
    <author><name>AtomServer APP Service</name></author>
    <link href="/atomserver/v1/pets/dogs/fido.xml/3" rel="edit" />
    <as:entryId>fido</as:entryId>
    <as:updateIndex>27</as:updateIndex>
    <as:revision>2</as:revision>
    <updated>2008-10-06T23:22:27.071Z</updated>
    <published>2008-10-06T23:22:27.071Z</published>
    <category scheme="urn:pets.owner" term="123" />
    <content type="application/xml">
      <pet xmlns="http://schemas.atomserver.org/pets/v1/rev0"
               name="fido" type="dog">
        <breed>Mixed</breed>
        <owner>123</owner>
      </pet>
    </content>
  </entry>
  <entry>
    <id>/atomserver/v1/pets/dogs/sparky.xml</id>
    .....
```

Entries and feeds contain *metadata*.

```
<feed xmlns=http://www.w3.org/2005/Atom xmlns:as="http://atomserver.org/
    namespaces/1.0/">
  <as:endIndex>7</as:endIndex>
  <author><name>AtomServer APP Service</name></author>
  <title type="text">dogs entries</title>
  <updated>2008-10-06T23:22:27.073Z</updated>
  <id>tag:atomserver.org,2008:v1:dogs</id>
  <entry>
    <id>/atomserver/v1/pets/dogs/fido.xml</id>
    <title type="text"> Entry: dogs fido</title>
    <author><name>AtomServer APP Service</name></author>
    <link href="/atomserver/v1/pets/dogs/fido.xml/3" rel="edit" />
    <as:entryId>fido</as:entryId>
    <as:updateIndex>27</as:updateIndex>
    <as:revision>2</as:revision>
    <updated>2008-10-06T23:22:27.071Z</updated>
    <published>2008-10-06T23:22:27.071Z</published>
    <category scheme="urn:pets.owner" term="123" />
    <content type="application/xml">
      <pet xmlns="http://schemas.atomserver.org/pets/v1/rev0" name="fido"
   type="dog">
        <breed>Mixed</breed>
        <owner>123</owner>
      </pet>
    </content>
  </entry>
  <entry>
    <id>/atomserver/v1/pets/dogs/sparky.xml</id>
    .....
```

Some of the *metadata* is Atom specific.

```
<feed xmlns=http://www.w3.org/2005/Atom xmlns:as="http://atomserver.org/
    namespaces/1.0/">
  <as:endIndex>7</as:endIndex>
  <author><name>AtomServer APP Service</name></author>
  <title type="text">dogs entries</title>
  <updated>2008-10-06T23:22:27.073Z</updated>
  <id>tag:atomserver.org,2008:v1:dogs</id>
  <entry>
    <id>/atomserver/v1/pets/dogs/fido.xml</id>
    <title type="text"> Entry: dogs fido</title>
    <author><name>AtomServer APP Service</name></author>
    <link href="/atomserver/v1/pets/dogs/fido.xml/3" rel="edit" />
    <as:entryId>fido</as:entryId>
    <as:updateIndex>27</as:updateIndex>
    <as:revision>2</as:revision>
    <updated>2008-10-06T23:22:27.071Z</updated>
    <published>2008-10-06T23:22:27.071Z</published>
    <category scheme="urn:pets.owner" term=”123" />
    <content type="application/xml">
      <pet xmlns="http://schemas.atomserver.org/pets/v1/rev0" name=“fido”
   type="dog">
        <breed>Mixed</breed>
        <owner>123</owner>
      </pet>
    </content>
  </entry>
  <entry>
    <id>/atomserver/v1/pets/dogs/sparky.xml</id>
    .....
```

And some of the *metadata* is extension elements.

```
<feed xmlns=http://www.w3.org/2005/Atom xmlns:as="http://atomserver.org/
    namespaces/1.0/">
  <as:endIndex>7</as:endIndex>
  <author><name>AtomServer APP Service</name></author>
  <title type="text">dogs entries</title>
  <updated>2008-10-06T23:22:27.073Z</updated>
  <id>tag:atomserver.org,2008:v1:dogs</id>
  <entry>
    <id>/atomserver/v1/pets/dogs/fido.xml</id>
    <title type="text"> Entry: dogs fido</title>
    <author><name>AtomServer APP Service</name></author>
    <link href="/atomserver/v1/pets/dogs/fido.xml/3" rel="edit" />
    <as:entryId>fido</as:entryId>
    <as:updateIndex>27</as:updateIndex>
    <as:revision>2</as:revision>
    <updated>2008-10-06T23:22:27.071Z</updated>
    <published>2008-10-06T23:22:27.071Z</published>
    <category scheme="urn:pets.owner" term="123" />
    <content type="application/xml">
      <pet xmlns="http://schemas.atomserver.org/pets/v1/rev0" name="fido"
   type="dog">
        <breed>Mixed</breed>
        <owner>123</owner>
      </pet>
    </content>
  </entry>
  <entry>
    <id>/atomserver/v1/pets/dogs/sparky.xml</id>
    .....
```

And some of the *metadata* is extension elements.

```
<feed xmlns=http://www.w3.org/2005/Atom xmlns:as="http://atomserver.org/
    namespaces/1.0/">
  <as:endIndex>7</as:endIndex>
  <author><name>AtomServer APP Service</name></author>
  <title type="text">dogs entries</title>
  <updated>2008-10-06T23:22:27.073Z</updated>
  <id>tag:atomserver.org,2008:v1:dogs</id>
  <entry>
    <id>/atomserver/v1/pets/dogs/fido.xml</id>
    <title type="text"> Entry: dogs fido</title>
    <author><name>AtomServer APP Service</name></author>
    <link href="/atomserver/v1/pets/dogs/fido.xml/3" rel="edit" />
    <as:entryId>fido</as:entryId>
    <as:updateIndex>27</as:updateIndex>
    <as:revision>2</as:revision>
    <updated>2008-10-06T23:22:27.071Z</updated>
    <published>2008-10-06T23:22:27.071Z</published>
    <category scheme="urn:pets.owner" term="123" />
    <content type="application/xml">
      <pet xmlns="http://schemas.atomserver.org/pets/v1/rev0" name="fido"
   type="dog">
        <breed>Mixed</breed>
        <owner>123</owner>
      </pet>
    </content>
  </entry>
  <entry>
    <id>/atomserver/v1/pets/dogs/sparky.xml</id>
    .....
```

Extensions simply require a separate namespace

Entries and feeds contain metadata.

```
<feed xmlns=http://www.w3.org/2005/Atom xmlns:as="http://atomserver.org/
    namespaces/1.0/">
  <as:endIndex>7</as:endIndex>
  <author><name>AtomServer APP Service</name></author>
  <title type="text">dogs entries</title>
  <updated>2008-10-06T23:22:27.073Z</updated>
  <id>tag:atomserver.org,2008:v1:dogs</id>
  <entry>
    <id>/atomserver/v1/pets/dogs/fido.xml</id>
    <title type="text"> Entry: dogs fido</title>
    <author><name>AtomServer APP Service</name></author>
    <link href="/atomserver/v1/pets/dogs/fido.xml/3" r
    <as:entryId>fido</as:entryId>
    <as:updateIndex>27</as:updateIndex>
    <as:revision>2</as:revision>
    <updated>2008-10-06T23:22:27.071Z</updated>
    <published>2008-10-06T23:22:27.071Z</published>
    <category scheme="urn:pets.owner" term="123" />
    <content type="application/xml">
      <pet xmlns="http://schemas.atomserver.org/pets/v1/rev0" name="fido"
   type="dog">
        <breed>Mixed</breed>
        <owner>123</owner>
      </pet>
    </content>
  </entry>
  <entry>
    <id>/atomserver/v1/pets/dogs/sparky.xml</id>
    .....
```

This metadata is the power of Atom!

Entries contain *content*.

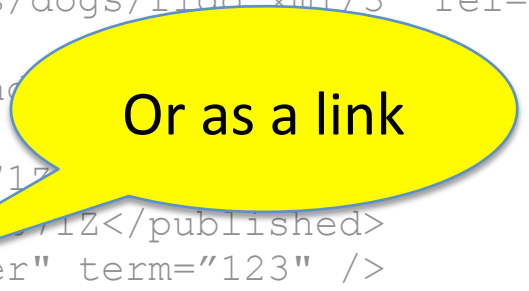
```
<feed xmlns=http://www.w3.org/2005/Atom xmlns:as="http://atomserver.org/
    namespaces/1.0/">
  <as:endIndex>7</as:endIndex>
  <author><name>AtomServer APP Service</name></author>
  <title type="text">dogs entries</title>
  <updated>2008-10-06T23:22:27.073Z</updated>
  <id>tag:atomserver.org,2008:v1:dogs</id>
  <entry>
    <id>/atomserver/v1/pets/dogs/fido.xml</id>
    <title type="text"> Entry: dogs fido</title>
    <author><name>AtomServer APP Service</name></author>
    <link href="/atomserver/v1/pets/dogs/fido.xml/3" rel="edit" />
    <as:entryId>fido</as:entryId>
    <as:updateIndex>27</as:updateIndex>
    <as:revision>2</as:revision>
    <updated>2008-10-06T23:22:27.071Z</updated>
    <published>2008-10-06T23:22:27.071Z</published>
    <category scheme="urn:pets.owner" term="123" />
    <content type="application/xml">
      <pet xmlns="http://schemas.atomserver.org/pets/v1/rev0" name="fido"
   type="dog">
        <breed>Mixed</breed>
        <owner>123</owner>
      </pet>
    </content>
  </entry>
  <entry>
    <id>/atomserver/v1/pets/dogs/sparky.xml</id>
    .....
```

Entries contain *content*.

```
<feed xmlns=http://www.w3.org/2005/Atom xmlns:as="http://atomserver.org/
    namespaces/1.0/">
  <as:endIndex>7</as:endIndex>
  <author><name>AtomServer APP Service</name></author>
  <title type="text">dogs entries</title>
  <updated>2008-10-06T23:22:27.073Z</updated>
  <id>tag:atomserver.org,2008:v1:dogs</id>
  <entry>
    <id>/atomserver/v1/pets/dogs/fido.xml</id>
    <title type="text"> Entry: dogs fido</title>
    <author><name>AtomServer APP Service</name></author>
    <link href="/atomserver/v1/pets/dogs/fido.xml/3" rel="edit" />
    <as:entryId>fido</as:entryId>
    <as:updateIndex>27</as:updateIndex>
    <as:revision>2</as:revision>
    <updated>2008-10-06T23:22:27.071Z</upd
    <published>2008-10-06T23:22:27.071Z</pub
    <category scheme="urn:pets.owner" term
    <content type="application/xml">
      <pet xmlns="http://schemas.atomserver.org/pets/v1/rev0" name="fido"
    type="dog">
        <breed>Mixed</breed>
        <owner>123</owner>
      </pet>
    </content>
  </entry>
  <entry>
    <id>/atomserver/v1/pets/dogs/sparky.xml</id>
    .....
```

Entries contain *content*.

```
<feed xmlns=http://www.w3.org/2005/Atom xmlns:as="http://
   atomserver.org/namespaces/1.0/">
  <as:endIndex>7</as:endIndex>
  <author><name>AtomServer APP Service</name></author>
  <title type="text">dogs entries</title>
  <updated>2008-10-06T23:22:27.073Z</updated>
  <id>tag:atomserver.org,2008:v1:dogs</id>
  <entry>
    <id>/atomserver/v1/pets/dogs/fido.xml</id>
    <title type="text"> Entry: dogs fido</title>
    <author><name>AtomServer APP Service</name></author>
    <link href="/atomserver/v1/pets/dogs/fido.xml/3" rel="edit" />
    <as:entryId>fido</as:entryId>
    <as:updateIndex>27</as:updateIn
    <as:revision>2</as:revision>
    <updated>2008-10-06T23:22:27.071
    <published>2008-10-06T23:22:27
    <category scheme="urn:pets:owner" term="123" />
    <link href="pets/dogs/fido.xml" rel="alternate" />
</entry>
  <entry>
    <id>/atomserver/v1/pets/dogs/sparky.xml</id>
    .....
```

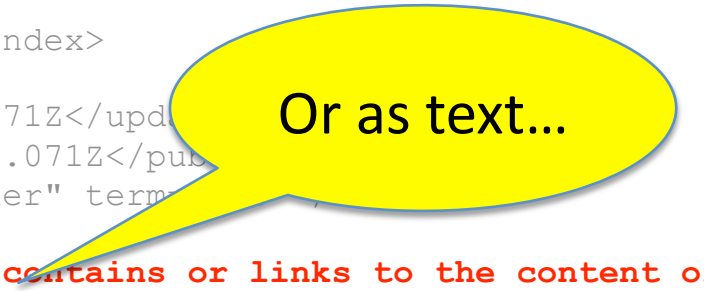


Entries contain *content*.

```
<feed xmlns=http://www.w3.org/2005/Atom xmlns:as="http://atomserver.org/
    namespaces/1.0/">
  <as:endIndex>7</as:endIndex>
  <author><name>AtomServer APP Service</name></author>
  <title type="text">dogs entries</title>
  <updated>2008-10-06T23:22:27.073Z</updated>
  <id>tag:atomserver.org,2008:v1:dogs</id>
  <entry>
    <id>/atomserver/v1/pets/dogs/fido.xml</id>
    <title type="text"> Entry: dogs fido</title>
    <author><name>AtomServer APP Service</name></author>
    <link href="/atomserver/v1/pets/dogs/fido.xml/3" rel="edit" />
    <as:entryId>fido</as:entryId>
    <as:updateIndex>27</as:updateIndex>
    <as:revision>2</as:revision>
    <updated>2008-10-06T23:22:27.071Z</upd
    <published>2008-10-06T23:22:27.071Z</pub
    <category scheme="urn:pets.owner" term
    <content type="application/xml">
      <pet xmlns="http://schemas.atomserver.org/pets/v1/rev0" name="fido"
    type="dog">
        <breed>Mixed</breed>
        <owner>123</owner>
      </pet>
    </content>
  </entry>
  <entry>
    <id>/atomserver/v1/pets/dogs/sparky.xml</id>
    .....
```

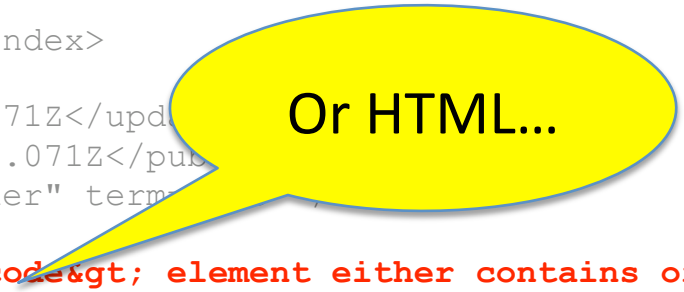
Entries contain *content*.

```
<feed xmlns=http://www.w3.org/2005/Atom xmlns:as="http://atomserver.org/
    namespaces/1.0/">
  <as:endIndex>7</as:endIndex>
  <author><name>AtomServer APP Service</name></author>
  <title type="text">dogs entries</title>
  <updated>2008-10-06T23:22:27.073Z</updated>
  <id>tag:atomserver.org,2008:v1:dogs</id>
  <entry>
    <id>/atomserver/v1/pets/dogs/fido.xml</id>
    <title type="text"> Entry: dogs fido</title>
    <author><name>AtomServer APP Service</name></author>
    <link href="/atomserver/v1/pets/dogs/fido.xml/3" rel="edit" />
    <as:entryId>fido</as:entryId>
    <as:updateIndex>27</as:updateIndex>
    <as:revision>2</as:revision>
    <updated>2008-10-06T23:22:27.071Z</upd
    <published>2008-10-06T23:22:27.071Z</pub
    <category scheme="urn:pets.owner" term
    <content>
The "atom:content" element either contains or links to the content of
the entry.  The content of atom:content is Language-Sensitive. The "atom:content"
element either contains or links to the content of the entry.  The content of
atom:content is Language-Sensitive.
    </content>
  </entry>
  <entry>
    <id>/atomserver/v1/pets/dogs/sparky.xml</id>
    .....
```



Entries contain *content*.

```
<feed xmlns=http://www.w3.org/2005/Atom xmlns:as="http://atomserver.org/
    namespaces/1.0/">
  <as:endIndex>7</as:endIndex>
  <author><name>AtomServer APP Service</name></author>
  <title type="text">dogs entries</title>
  <updated>2008-10-06T23:22:27.073Z</updated>
  <id>tag:atomserver.org,2008:v1:dogs</id>
  <entry>
    <id>/atomserver/v1/pets/dogs/fido.xml</id>
    <title type="text"> Entry: dogs fido</title>
    <author><name>AtomServer APP Service</name></author>
    <link href="/atomserver/v1/pets/dogs/fido.xml/3" rel="edit" />
    <as:entryId>fido</as:entryId>
    <as:updateIndex>27</as:updateIndex>
    <as:revision>2</as:revision>
    <updated>2008-10-06T23:22:27.071Z</upd
    <published>2008-10-06T23:22:27.071Z</pub
    <category scheme="urn:pets.owner" term
    <content type="html">
The &lt;code&gt;atom:content&lt;/code&gt; element either contains or links to the
content of the entry.  The content of &lt;code&gt;atom:content&lt;/code&gt; is
&lt;a href="http://www.ietf.org/rfc/rfc3066.txt"&gt;Language-Sensitive&lt;/a&gt;.
    </content>
  </entry>
  <entry>
    <id>/atomserver/v1/pets/dogs/sparky.xml</id>
    .....
```



Entries contain *content*.

```
<feed xmlns=http://www.w3.org/2005/Atom xmlns:as="http://atomserver.org/
    namespaces/1.0/">
  <as:endIndex>7</as:endIndex>
  <author><name>AtomServer APP Service</name></author>
  <title type="text">dogs entries</title>
  <updated>2008-10-06T23:22:27.073Z</updated>
  <id>tag:atomserver.org,2008:v1:dogs</id>
  <entry>
    <id>/atomserver/v1/pets/dogs/fido.xml</id>
    <title type="text"> Entry: dogs fido</title>
    <author><name>AtomServer APP Service</name></author>
    <link href="/atomserver/v1/pets/dogs/fido.xml/3" rel="edit" />
    <as:entryId>fido</as:entryId>
    <as:updateIndex>27</as:updateIndex>
    <as:revision>2</as:revision>
    <updated>2008-10-06T23:22:27.071Z</upd
    <published>2008-10-06T23:22:27.071Z</pub
    <category scheme="urn:pets.owner" term
    <content type='xhtml'>
       <div xmlns='http://www.w3.org/1999/xhtml'>
            <p>Isn't life grand!?!</p>
            <ul>
              <li>sure is.</li>
            </ul>
       </div>
    </content>
  </entry>
  <entry>
    <id>/atomserver/v1/pets/dogs/sparky.xml</id>
    .....
```

Entries contain *content*.

```
<feed xmlns=http://www.w3.org/2005/Atom xmlns:as="http://atomserver.org/
    namespaces/1.0/">
  <as:endIndex>7</as:endIndex>
  <author><name>AtomServer APP Service</name></author>
  <title type="text">dogs entries</title>
  <updated>2008-10-06T23:22:27.073Z</updated>
  <id>tag:atomserver.org,2008:v1:dogs</id>
  <entry>
    <id>/atomserver/v1/pets/dogs/fido.xml</id>
    <title type="text"> Entry: dogs fido</title>
    <author><name>AtomServer APP Service</name></author>
    <link href="/atomserver/v1/pets/dogs/fido.xml/3" rel="edit" />
    <as:entryId>fido</as:entryId>
    <as:updateIndex>27</as:updateIndex>
    <as:revision>2</as:revision>
    <updated>2008-10-06T23:22:27.071Z</upda
    <published>2008-10-06T23:22:27.071Z</pu
    <category scheme="urn:pets.owner" te      123" />
     <content type="image/png">
09URSu4gzXiKMbxgHgBTzGcfEyirxivitH5LeF1yvIkXuLZptKdw98Q28Sf58y
HlWgfnsIxmEo/NvHRxCB/xgng/ZfkFg1glTBPP4w3h+4agHhOW8TAvuVcpuBV8
rmxl7iKVKm4mn/WDtqA3yOQKuHaSApTAAAAAE      FTkSuQmCC
     </content>
  </entry>
  <entry>
    <id>/atomserver/v1/pets/dogs/sparky.xml</id>
    .....
```

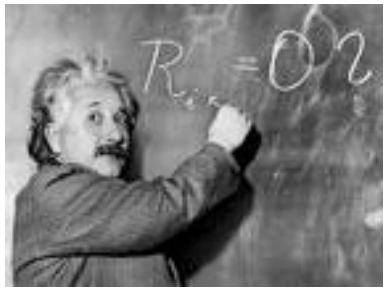
Media is supported

It must be valid base64 encoded

Entries contain *content*.

```
<feed xmlns=http://www.w3.org/2005/Atom xmlns:as="http://
   atomserver.org/namespaces/1.0/">
  <as:endIndex>7</as:endIndex>
  <author><name>AtomServer APP Service</name></author>
  <title type="text">dogs entries</title>
  <updated>2008-10-06T23:22:27.073Z</updated>
  <id>tag:atomserver.org,2008:v1:dogs</id>
  <entry>
    <id>/atomserver/v1/pets/dogs/fido.xml</id>
    <title type="text"> Entry: dogs fido</title>
    <author><name>AtomServer APP Service</name></author>
    <link href="/atomserver/v1/pets/dogs/                el="edit" />
    <as:entryId>fido</as:entryId>
    <as:updateIndex>27</as:updateInde
    <as:revision>2</as:revision>
    <updated>2008-10-06T23:22:27.071                updated>
    <published>2008-10-06T23:22:                .071Z</published>
    <category scheme="urn:pets.owner" term="123" />
    <content src="image.png" type="image/png"/>
  </entry>
  <entry>
    <id>/atomserver/v1/pets/dogs/sparky.xml</id>
    .....
```

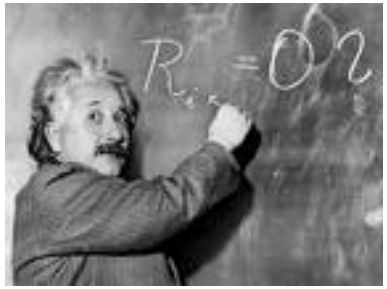
Or media as a link…



Professor Atom Says:

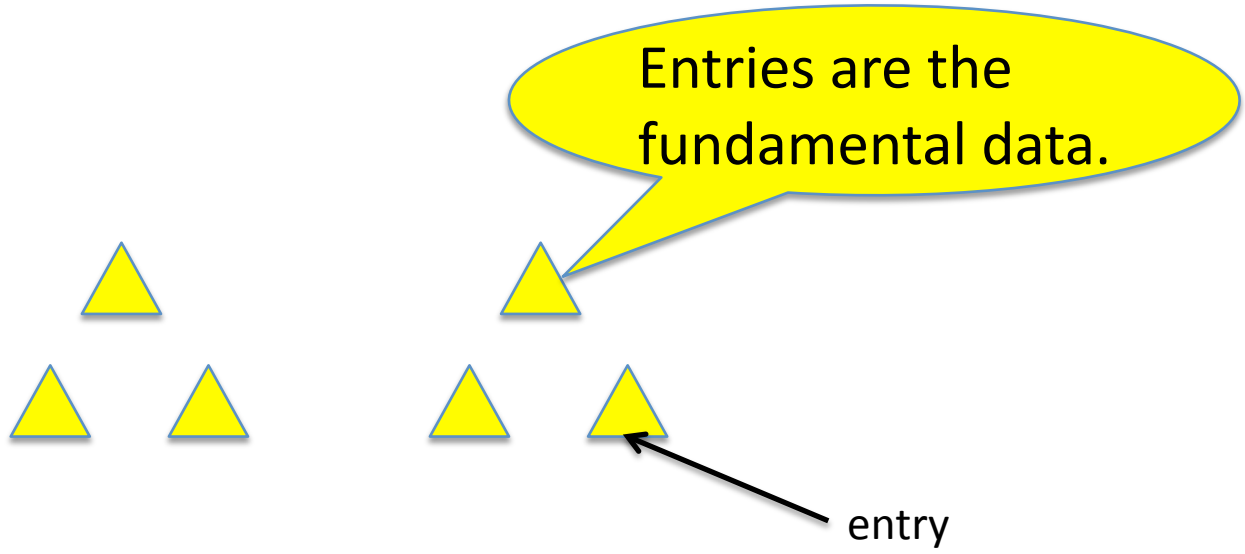
The Atom Publishing Protocol (AtomPub)

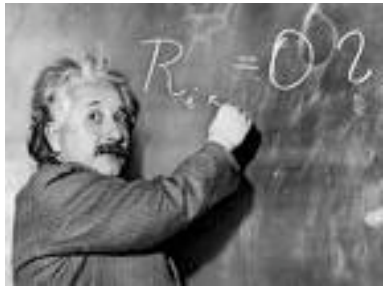
- A simple, RESTful protocol for creating and updating web resources
- Provides
  - A data layout
  - RESTful semantics for interacting with that data, including a mechanism for discovery.



Professor Atom Says:

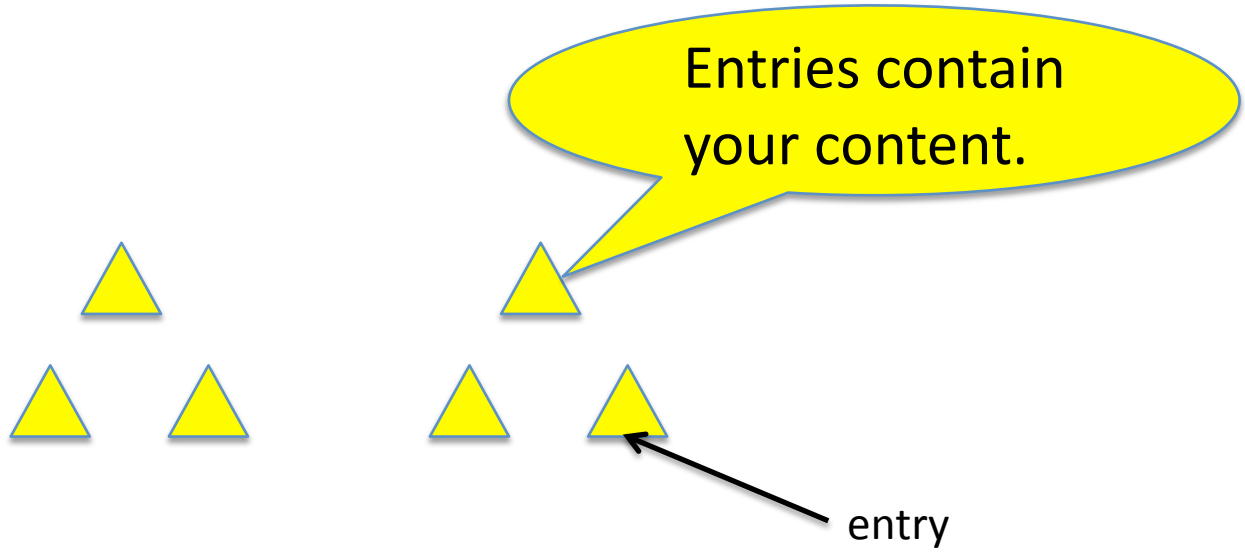
AtomPub organizes data into **services, workspaces, collections,** and **entries**

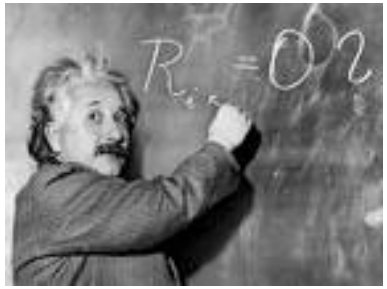




Professor Atom Says:

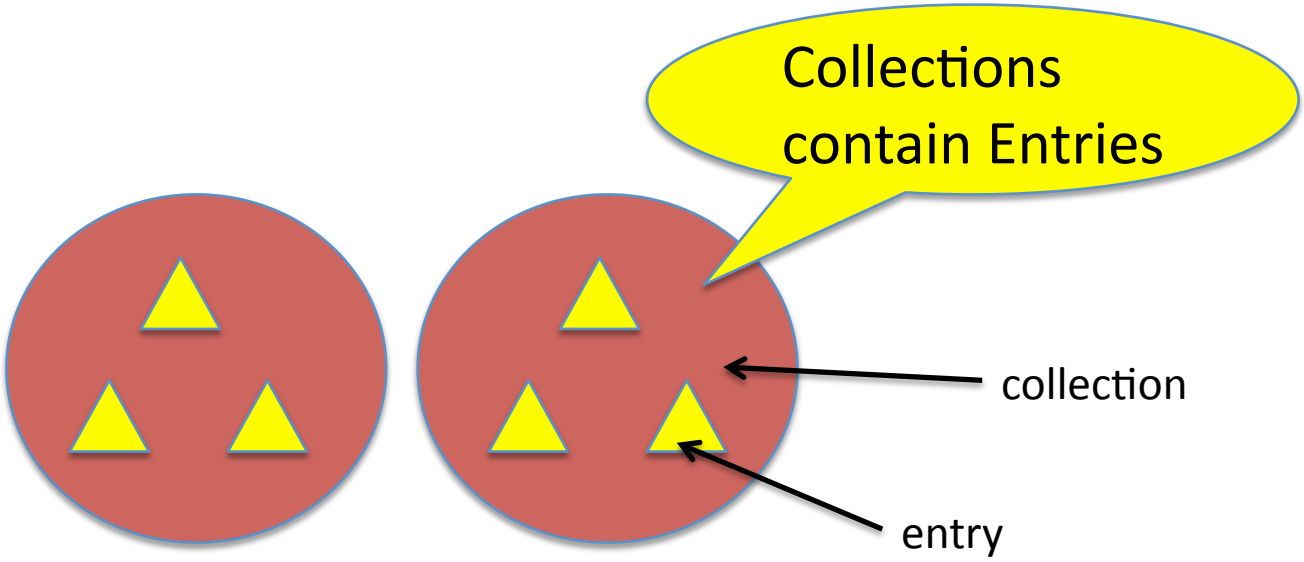
AtomPub organizes data into **services, workspaces, collections,** and **entries**

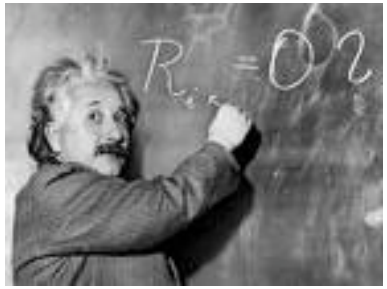




Professor Atom Says:

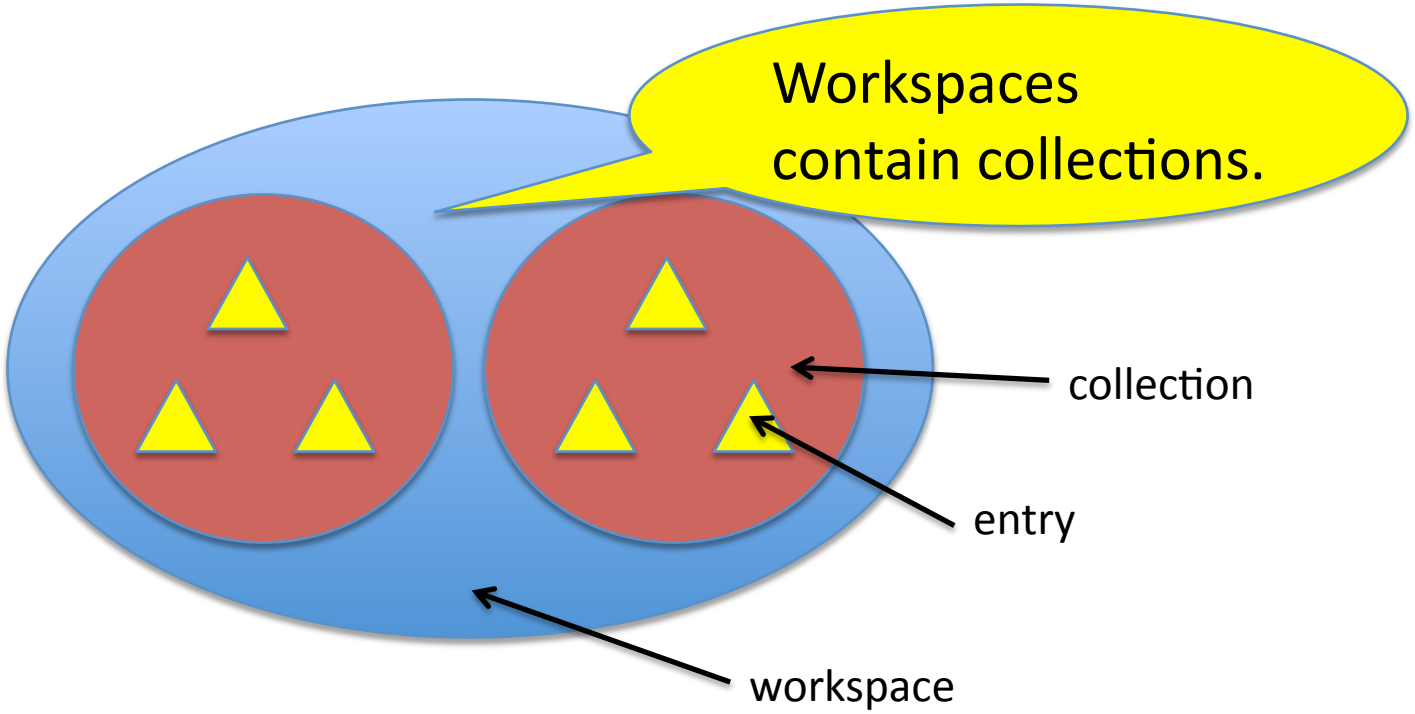
AtomPub organizes data into **services, workspaces, collections,** and **entries**

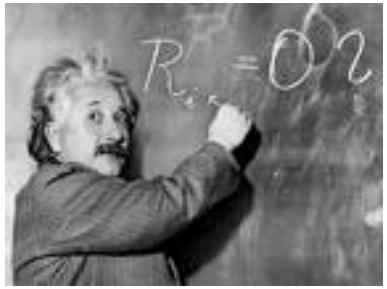




Professor Atom Says:

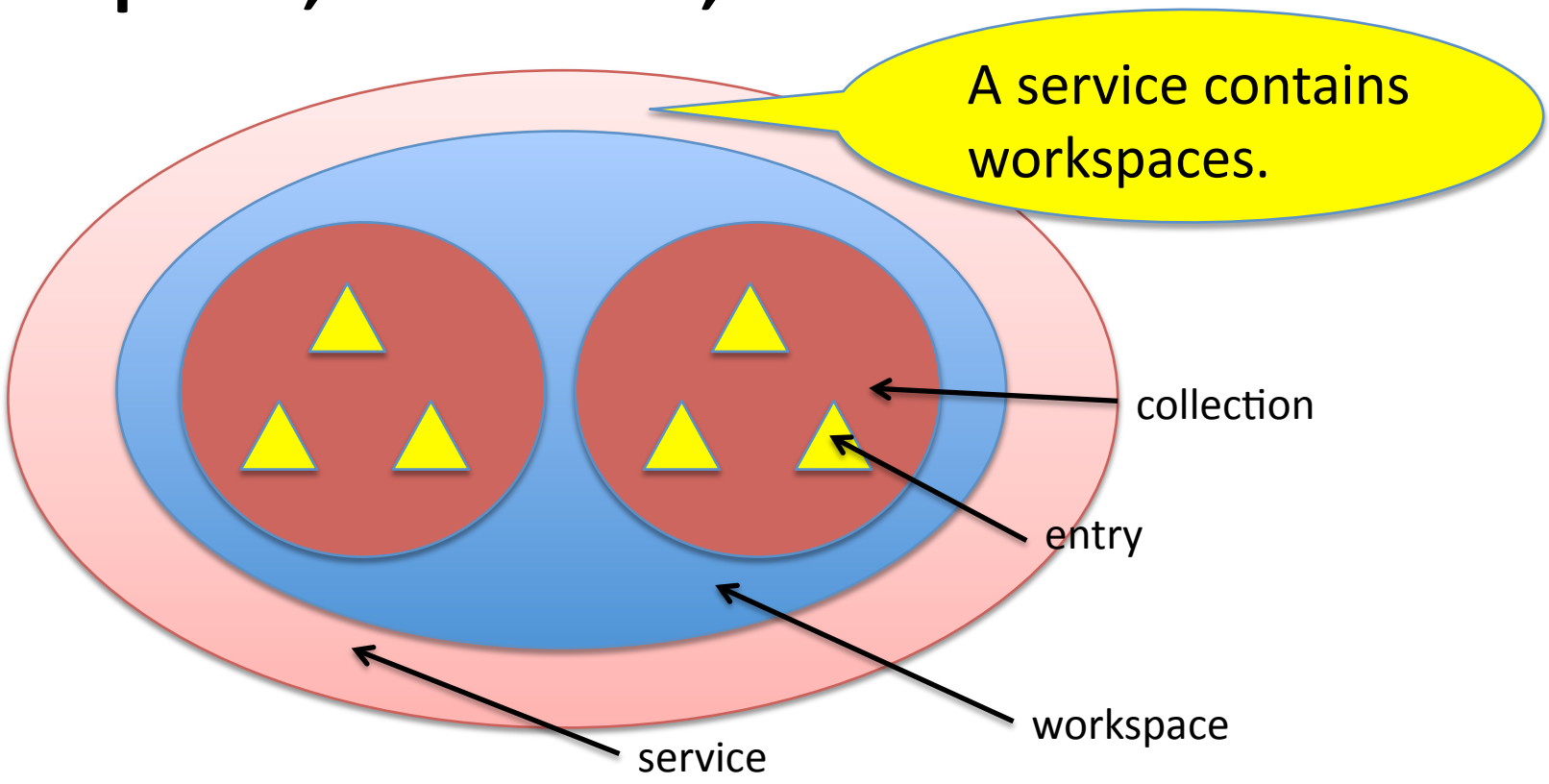
AtomPub organizes data into a **service, workspaces, collections,** and **entries**

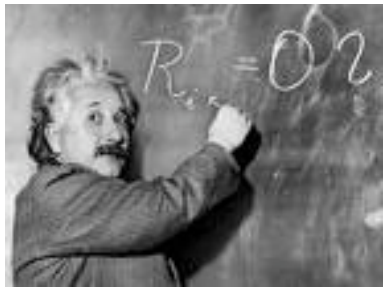




Professor Atom Says:

AtomPub organizes data into a **service, workspaces, collections,** and **entries**





Professor Atom Says:

AtomPub introduces the concept of a *service*

- AtomPub's mechanism for discovery
- How you find out what workspaces and collections are available in an Atom-enabled service.

Service document

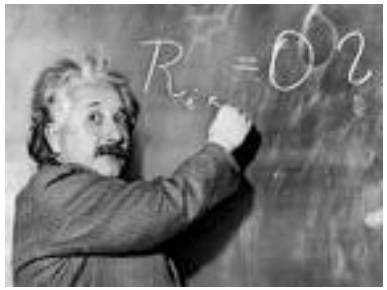
```
<app:service xmlns:app="http://purl.org/atom/app#"
                        xmlns:atom="http://www.w3.org/2005/atom">
    <app:workspace>
         <atom:title>Pets</atom:title>
         <app:collection href="http://localhost:8080/pets/dogs">
              <atom:title>Dogs</atom:title>
              <app:accept>entry</app:accept>
         </app:collection>
         <app:collection href="http://localhost:8080/pets/dog-images">
             <atom:title>Dog pictures</atom:title>
             <app:accept>image/*</app:accept>
         </app:collection>
         …..
    </app:workspace>
    <app:workspace>
            ...
    </app:workspace>
</app:service>
```

Service document

```
<app:service xmlns:app="http://purl.org/atom/app#"
                        xmlns:atom="http://www.w3.org/2005/atom">
    <app:workspace>
        <atom:title>Pets</atom:title>
        <app:collection href="http://localhost:8080/pets/dogs">
            <atom:title>Dogs</atom:title>
            <app:accept>entry</app:accept>
        </app:collection>
        <app:collection href="http://localhost:8080/pets/dog-images">
            <atom:title>Dog pictures</atom:title>
            <app:accept>image/*</app:accept>
        </app:collection>
        …..
    </app:workspace>
    <app:workspace>
            ...
    </app:workspace>
</app:service>
```

Service document

```
<app:service xmlns:app="http://purl.org/atom/app#"
                        xmlns:atom="http://www.w3.org/2005/atom">
    <app:workspace>
        <atom:title>Pets</atom:title>
        <app:collection href="http://localhost:8080/pets/dogs">
            <atom:title>Dogs</atom:title>
            <app:accept>entry</app:accept>
        </app:collection>
        <app:collection href="http://localhost:8080/pets/dog-images">
            <atom:title>Dog pictures</atom:title>
            <app:accept>image/*</app:accept>
        </app:collection>
        …..
    </app:workspace>
    <app:workspace>
            ...
    </app:workspace>
</app:service>
```

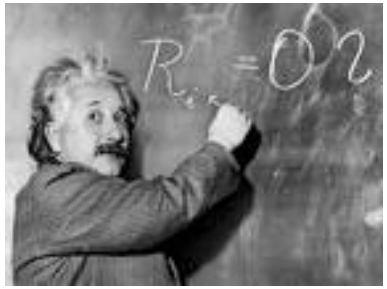


Professor Atom Says:

AtomPub is RESTful

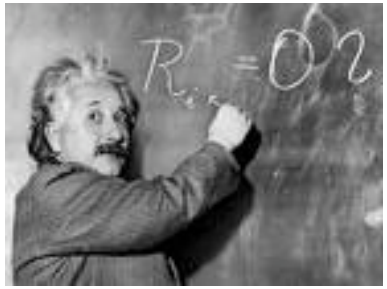
services, workspaces, collections, and entries are all RESTful Resources

- Each resource is uniquely and globally identified by a URI
- Each is identified by a simple unique name within its respective container



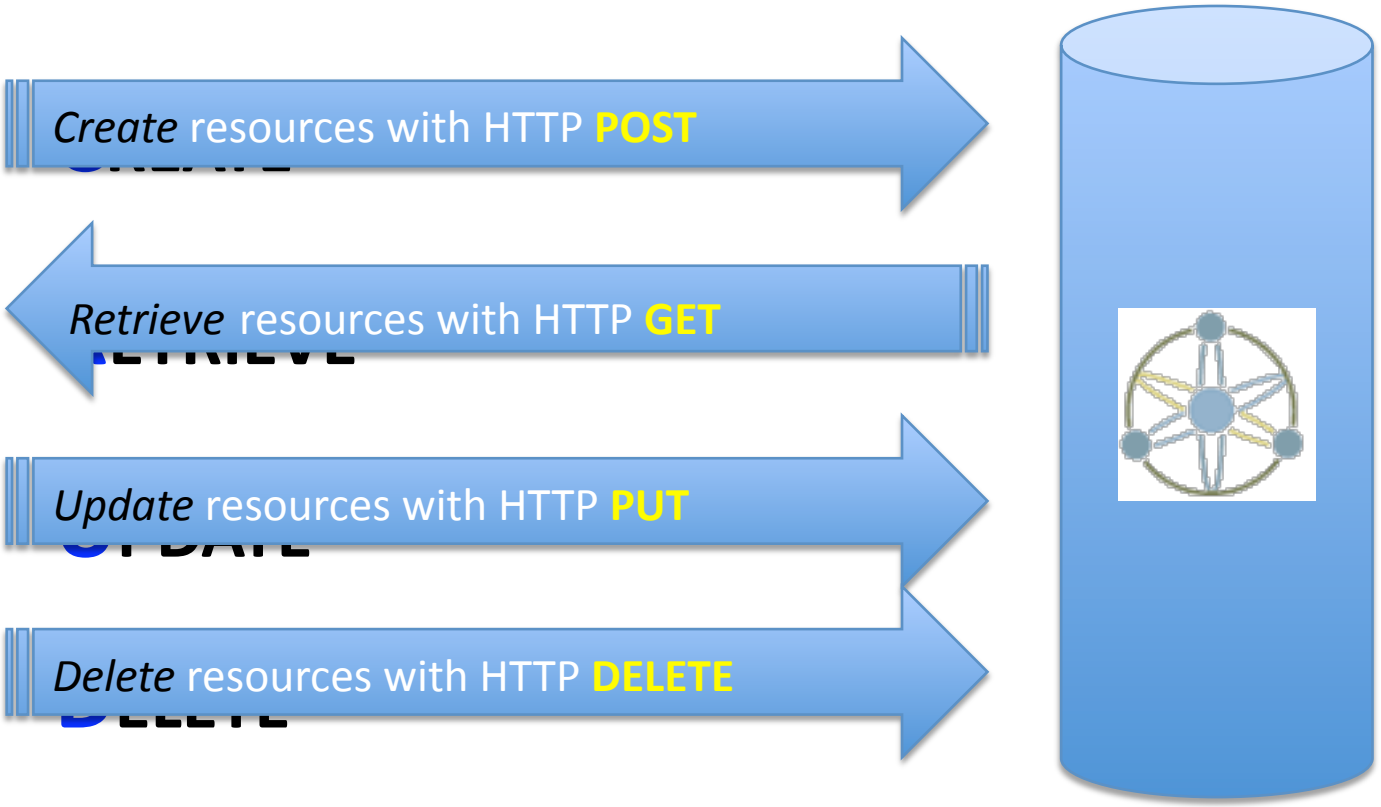
Professor Atom Says:

- Every resource is interacted with using a universally predefined set of "verbs".
- These verbs are not defined by individual resources, but across all resources.
- Each verb has clearly defined semantics that can be relied upon, which is critical so intermediaries can do the right thing.



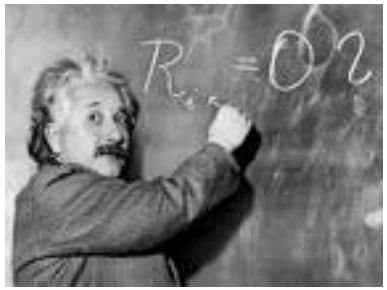
Professor Atom Says:

AtomPub is a RESTful HTTP protocol

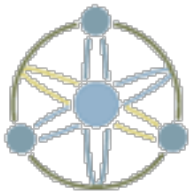


HTTP status codes

Code	Meaning
200 OK	No error
201 CREATED	Creation of a resource was successful.
304 NOT MODIFIED	The resource hasn't changed since the time specified in the request's If-Modified-Since header.
400 BAD REQUEST	Invalid request URI or header, or unsupported nonstandard parameter.
401 UNAUTHORIZED	Authorization required.
403 FORBIDDEN	Unsupported standard parameter, or authentication or authorization failed.
404 NOT FOUND	Resource (such as a feed or entry) not found.
409 CONFLICT	Specified version number doesn't match resource's latest version number.
422 BAD CONTENT	The data within this entry's <content> is not valid. For example, this may indicate not "well-formed" XML
500 INTERNAL SERVER ERROR	Internal error. This is the default code that is used for all unrecognized errors.

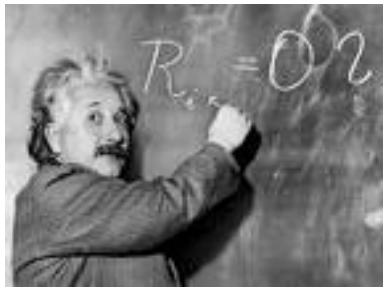


Professor Atom Says:



Extending Atom

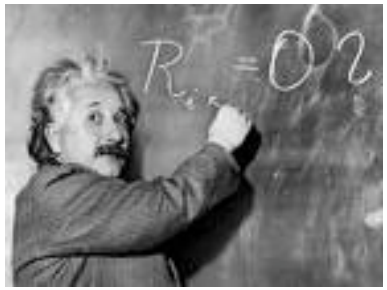
- Atom defines only a few resources and the corresponding operations on those resources.
- This leaves gaps that require extensions to Atom.



Professor Atom Says:

Atom is easily extended.

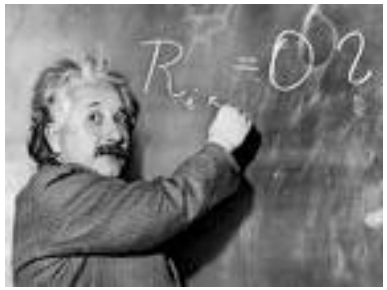
- Atom is an open protocol – it allows XML elements and attributes from other namespaces to be freely intermixed with the Atom building blocks: **feeds**, **entries**, **categories**, etc.
- Extensions can enrich the basic Atom elements with additional semantics not covered by Atom.
- Extensions can be ignored by clients that don't know how to use them – they can still process the basic Atom Elements.



Professor Atom Says:

Two popular Atom Extensions are:

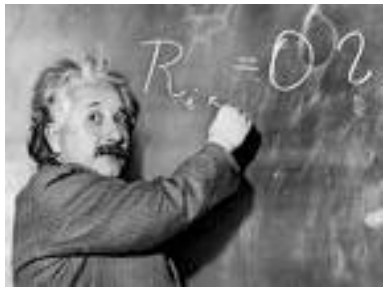
- OpenSearch
- Feed Paging



Professor Atom Says:

OpenSearch

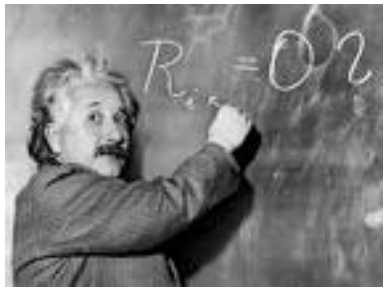
- A joint spec by several industry giants
  - www.opensearch.org
- A RESTful protocol for AtomPub-based searching
  - The search query is embedded in a feed request (in the URL)
  - The query results are returned as an Atom feed, with the individual query results represented as entries



Professor Atom Says:

Feed Paging

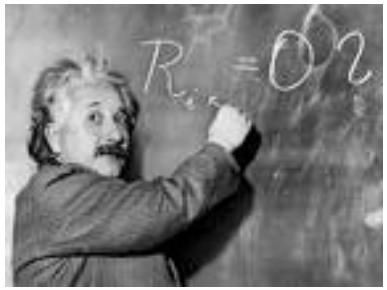
- http://tools.ietf.org/html/rfc5005
- Addresses the paging problem for time-based data.
  - Defines *next* and previous link types for Atom feeds
  - Clients then use these links to page forward and backward through a multi-page feed



Professor Atom Says:

Putting all this together leads us to the *"Atom Store"*

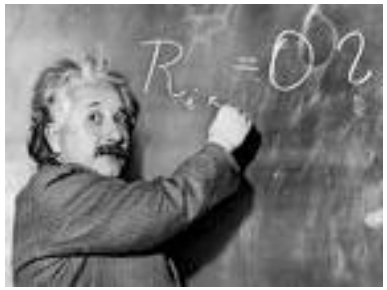
- A generic data store of inter-linked Atom entries, which:
  - You can edit using AtomPub
  - Has searching capabilities (OpenSearch, Category-based feeds)
  - You can page through the feed results using Feed Paging
- The seminal example of this is *GData*.
  - GData is Google's API for accessing their data-oriented services – it consists of the Atom specs, OpenSearch, and several custom extensions.



Professor Atom Says:

And, finally, to AtomServer

- A full-fledged, out-of-the box, scalable, battle-hardened Atom Store
  - It IS NOT an API for bolting Atom onto an existing data store (we use **Apache Abdera** to implement the Atom specs, and it's a great library for doing this.)



Professor Atom Says:

AtomServer

It IS a standalone data service that stores your data and manages the Atom metadata about your data

Implements useful extensions invented by industry leaders (Google/GData) and some custom extensions of our own

So, back to our business objectives – how is this going to help us…

Get our main site to critical mass, by combining data from our acquisitions?

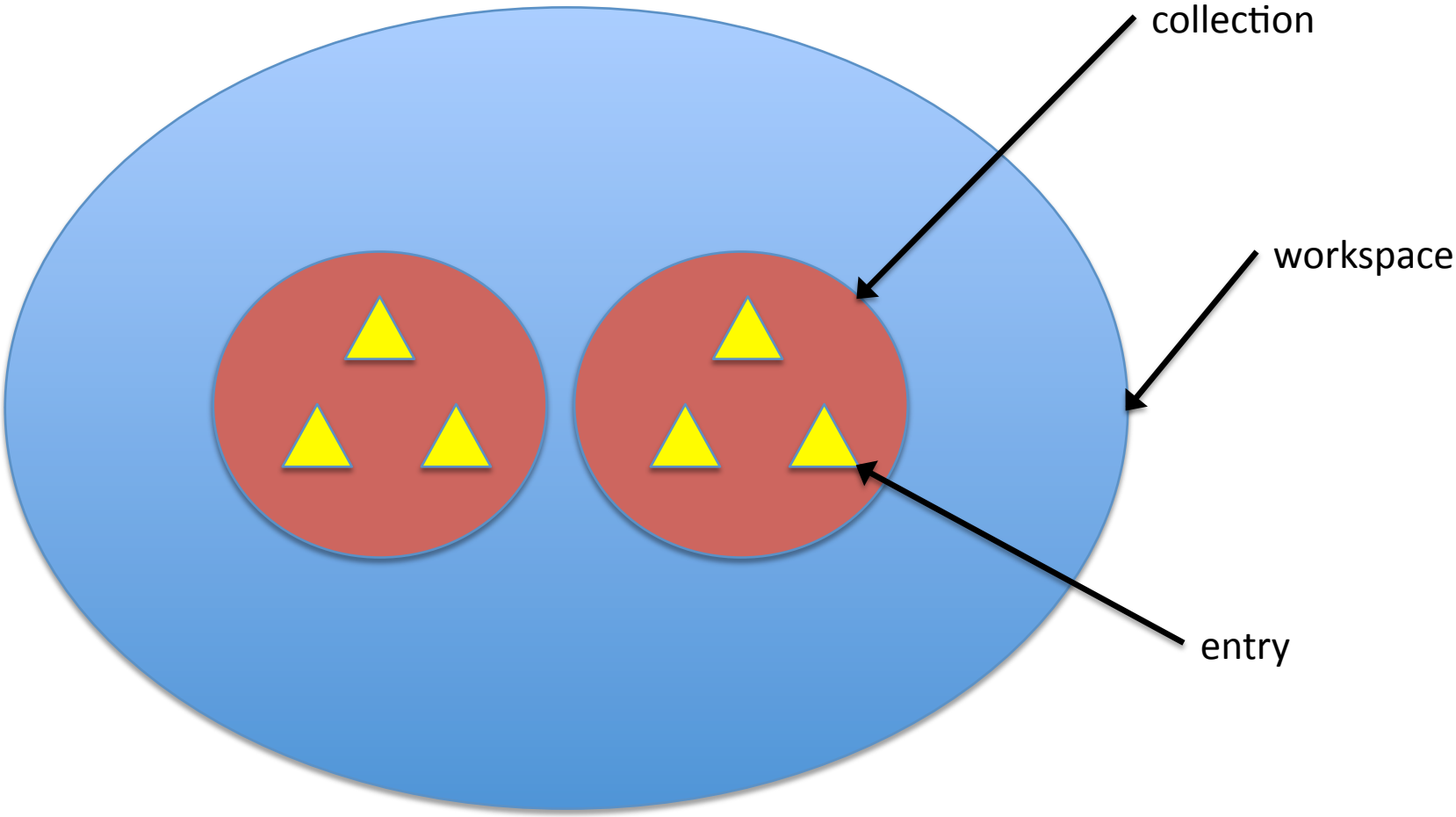
Increase revenue by “cross-selling” customers beteen our sites?

Simplify our business processes with a SOA that’s streamlined by Atom and AtomServer?

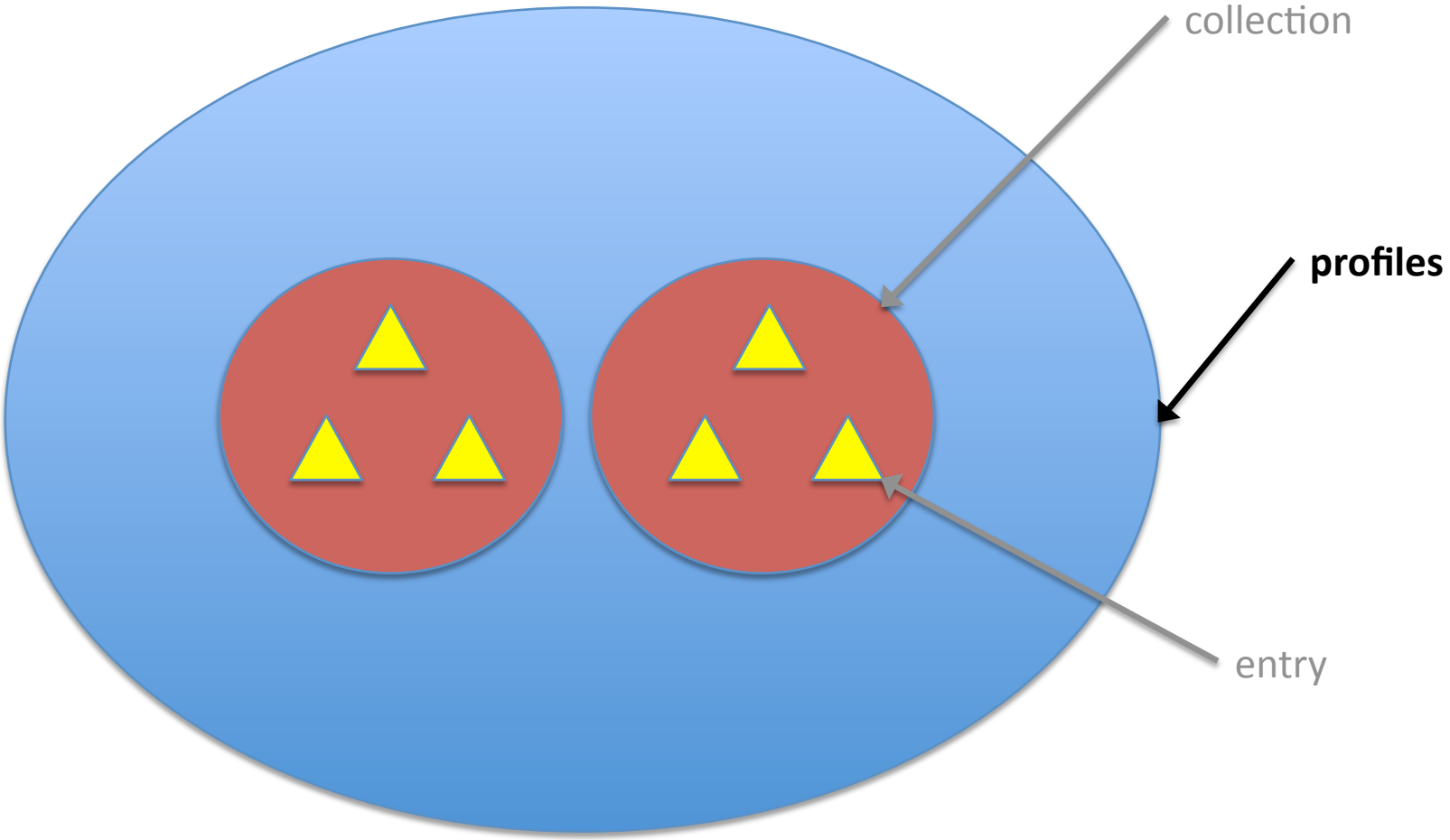
Get our main site to critical mass, by combining data from our acquisitions?

- The **Profiles** people list on our sites are our most valuable asset
- We need to configure our AtomServer to hold our **Profile** data

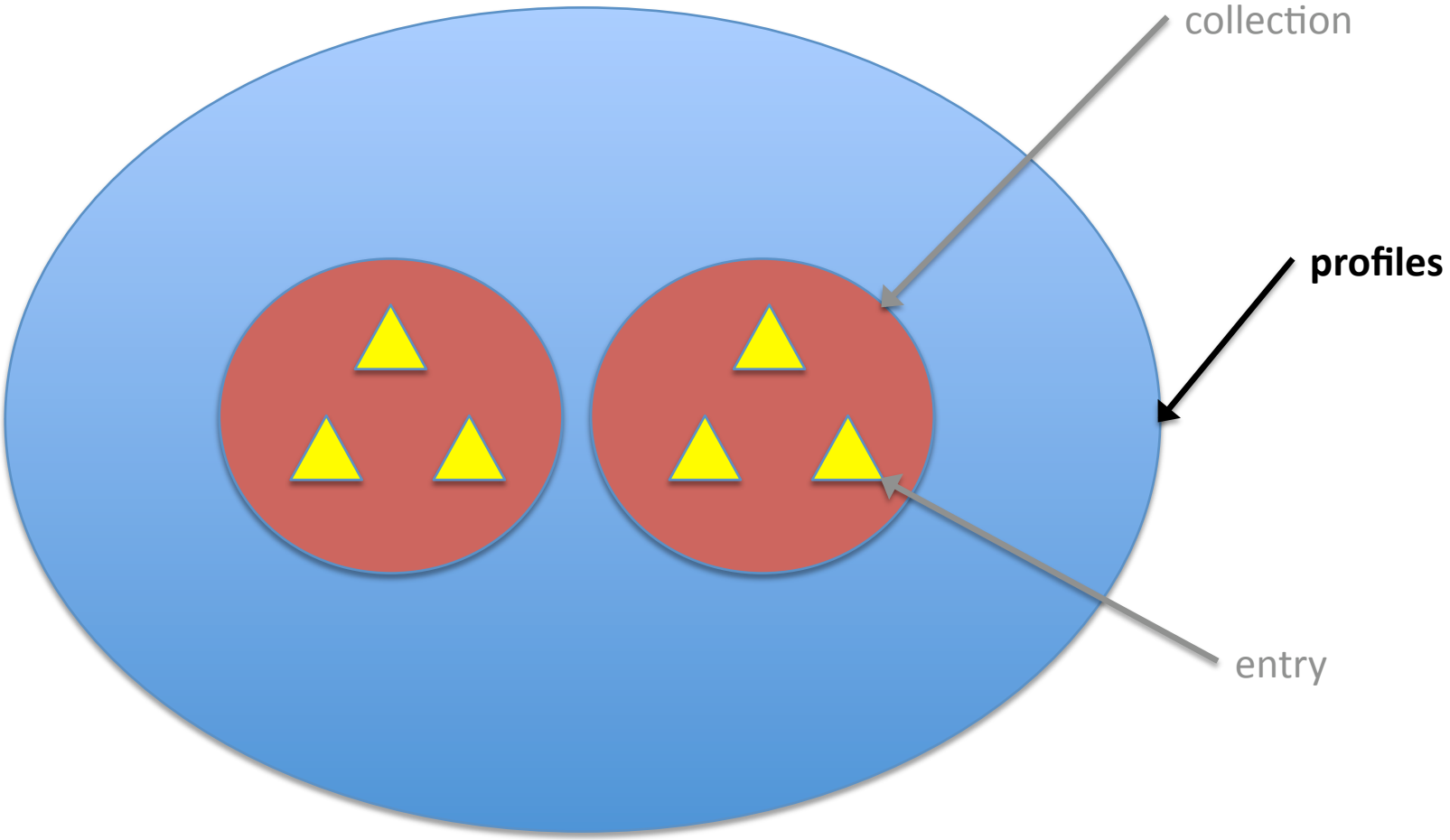
Atom Pub organizes data into **workspaces, collections** and **entries**



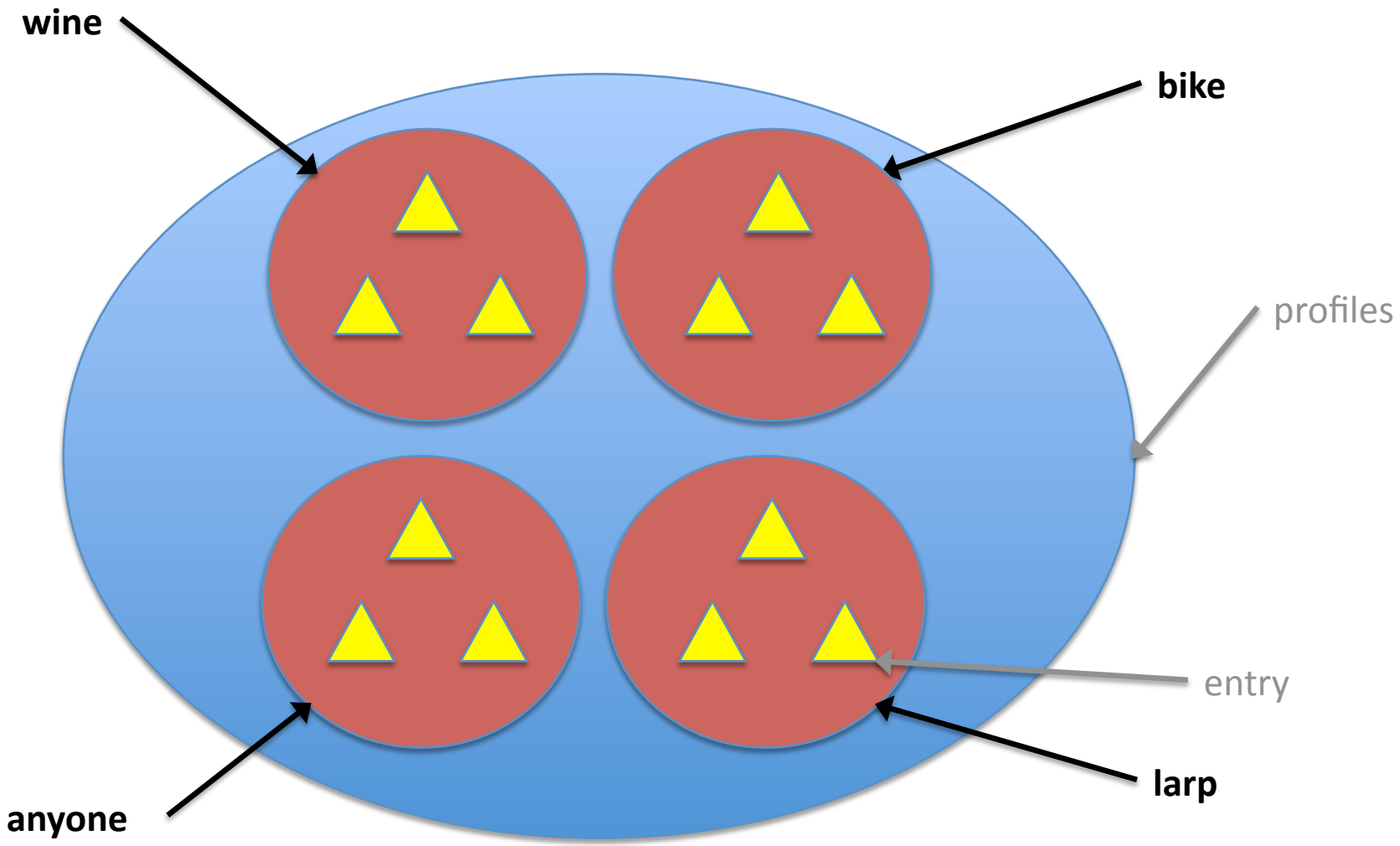
Our workspace will be called **profiles**



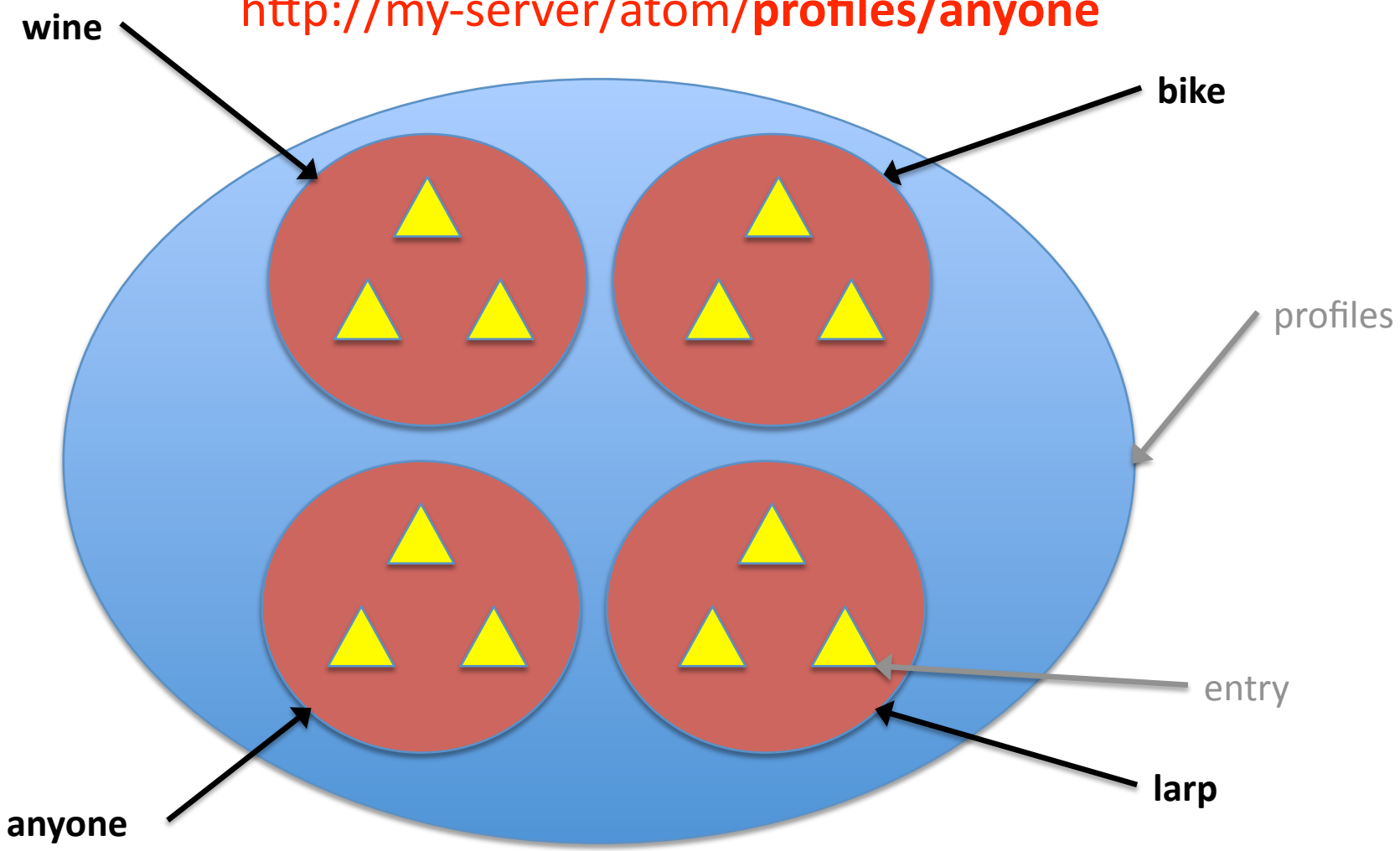
http://my-server/atom/**profiles**



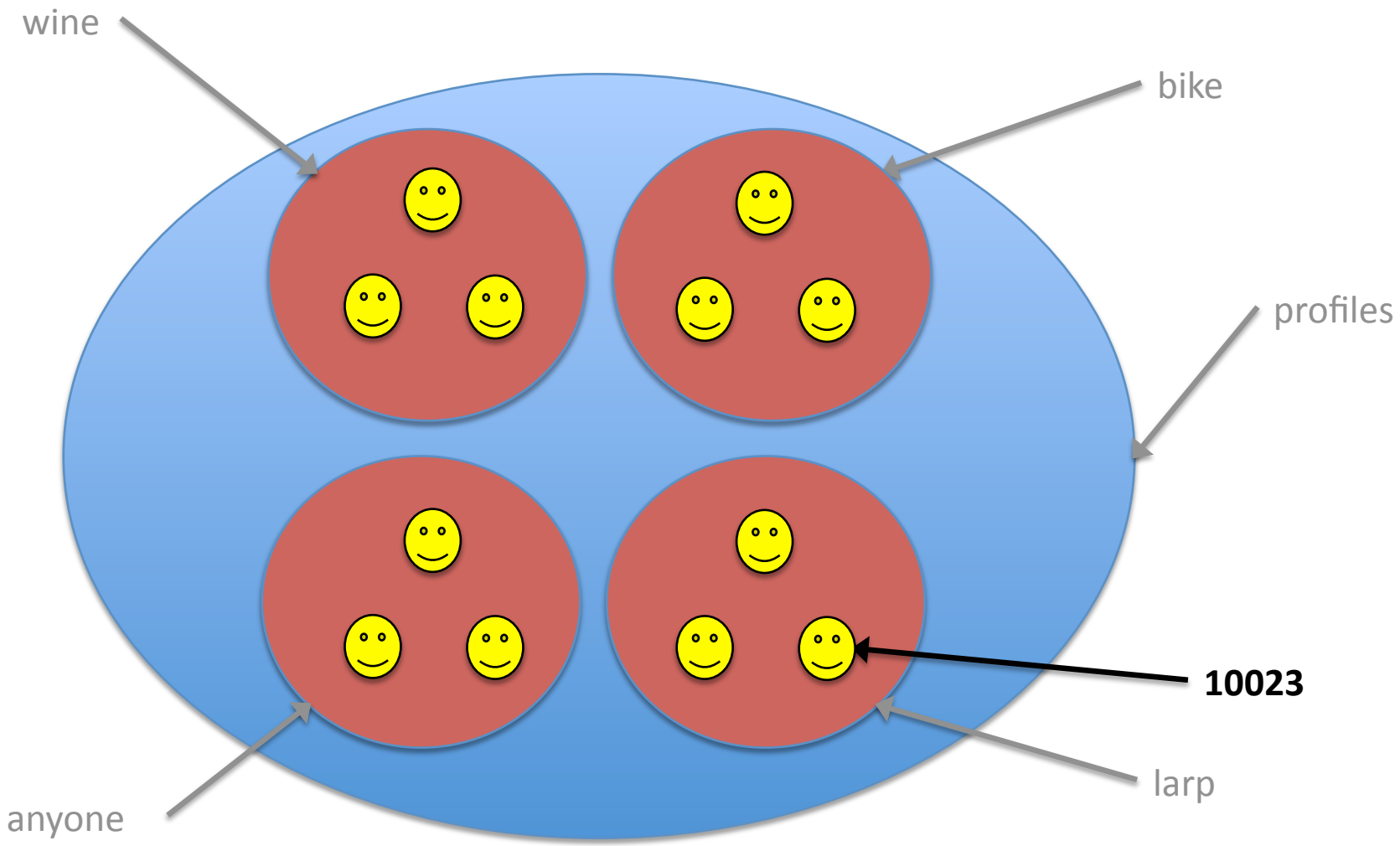
We define a collection for each website (**bike**, **wine**, **larp**, and **anyone**)

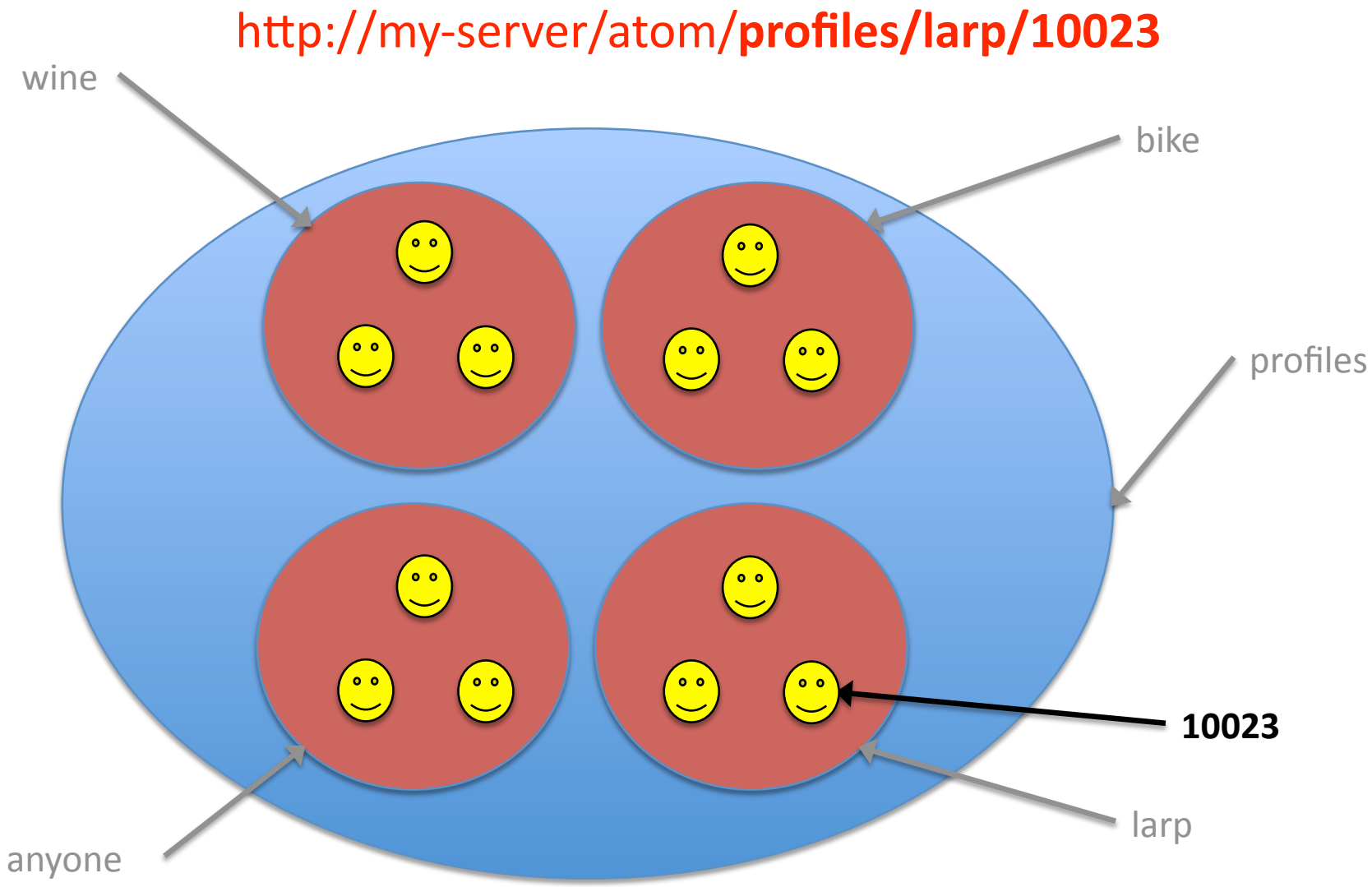


http://my-server/atom/**profiles/bike**
http://my-server/atom/**profiles/wine**
http://my-server/atom/**profiles/larp**
http://my-server/atom/**profiles/anyone**



And finally, each entry will refer to a specific profile…




http://my-server/atom/profiles/larp/10023
wine
bike
profiles
10023
larp
anyone

Service URIs

http://my-server/atom/

- A **GET** returns a **service document** listing all the workspaces and collections in this service.

Workspace URIs

http://my-server/atom/**profiles**

- A **GET** returns a **service document** listing the collections in this particular workspace.

Collection URIs

http://my-server/atom/**profiles/larp**

- The URI for the **larp** collection – a **GET** of this URI returns the first page of results in the feed.
- A **POST** of an entry to this URI creates a new entry, and returns the **entry id** to the caller.

Entry URIs

http://my-server/atom/**profiles/larp/10023**

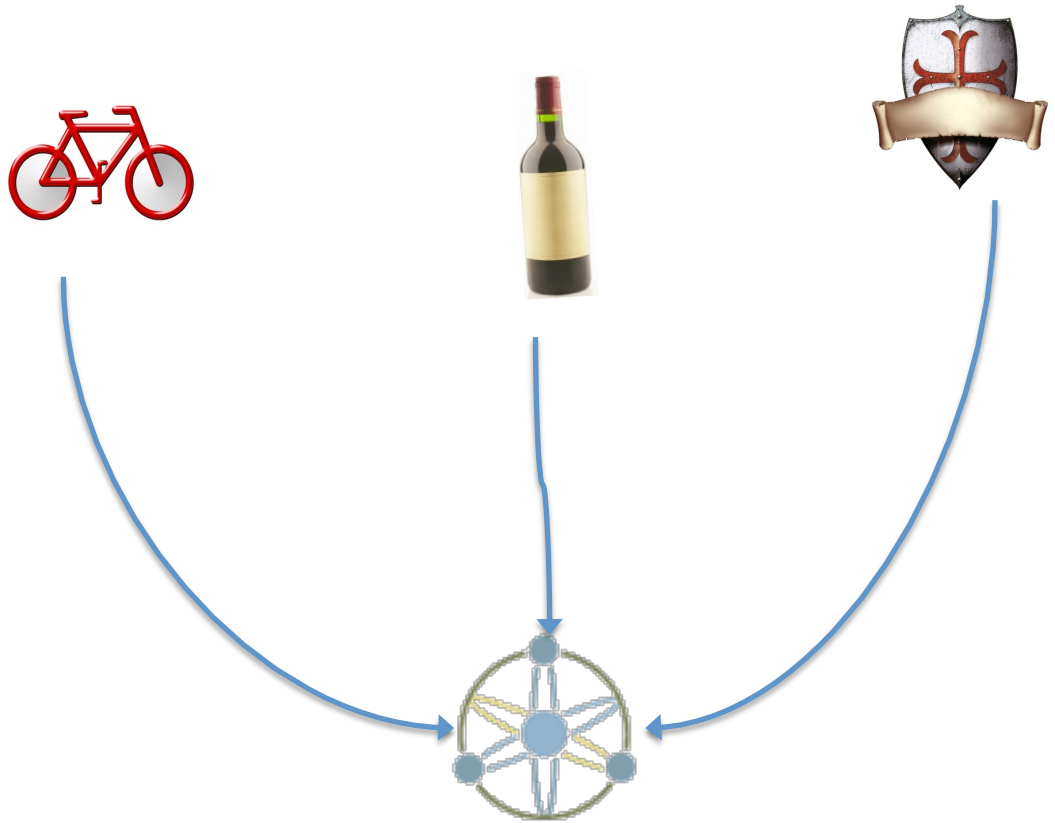
- The **entry URI** for the entry 10023 in the larp collection – **GET**ting this URI returns the entry itself.
- Performing a **PUT** to this URI edits the entry.
- Doing a **DELETE** to this URI deletes the entry.

Normalize Content

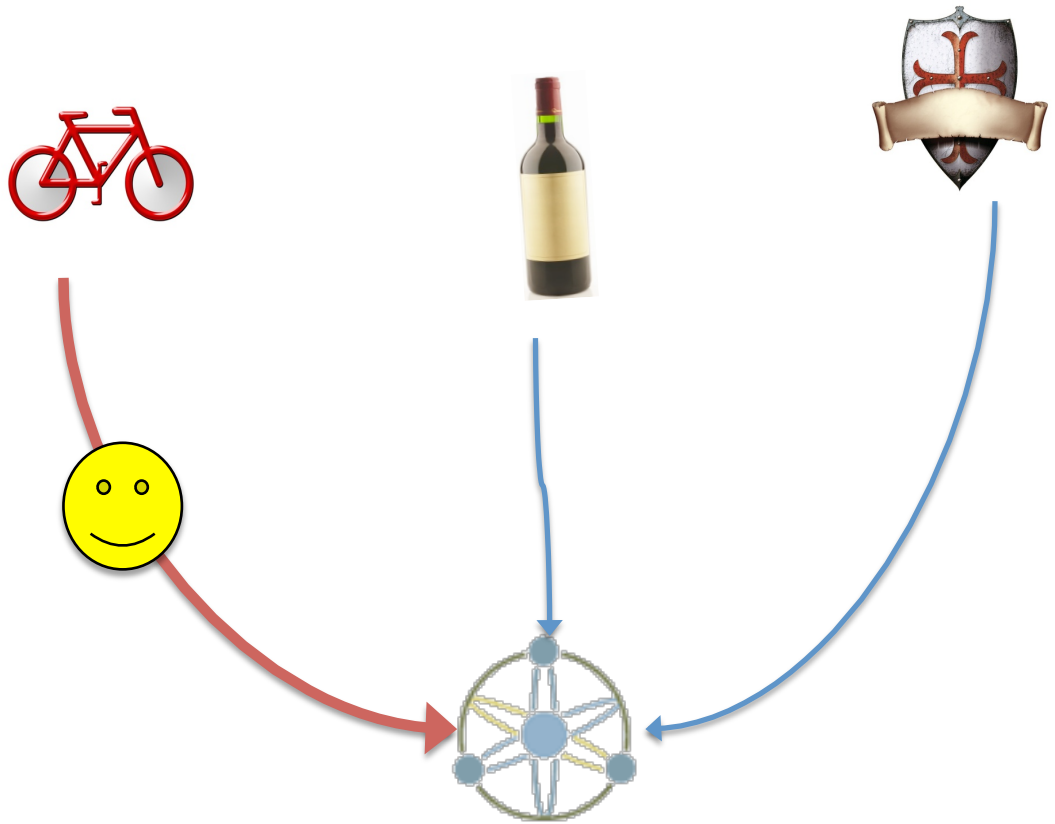
- Each site needs to define a transformation from their local format (probably DB tables) into the normalized format for the content.
- We'll define an XML schema for our profile data:

```
<profile>
  <first-name>Bryon</first-name>
  <last-name>Jacob</last-name>
  <birthdate>6/22/1976</birthdate>
    ...
</profile>
```

Each of the niche sites needs to publish changes to their data into the AtomServer

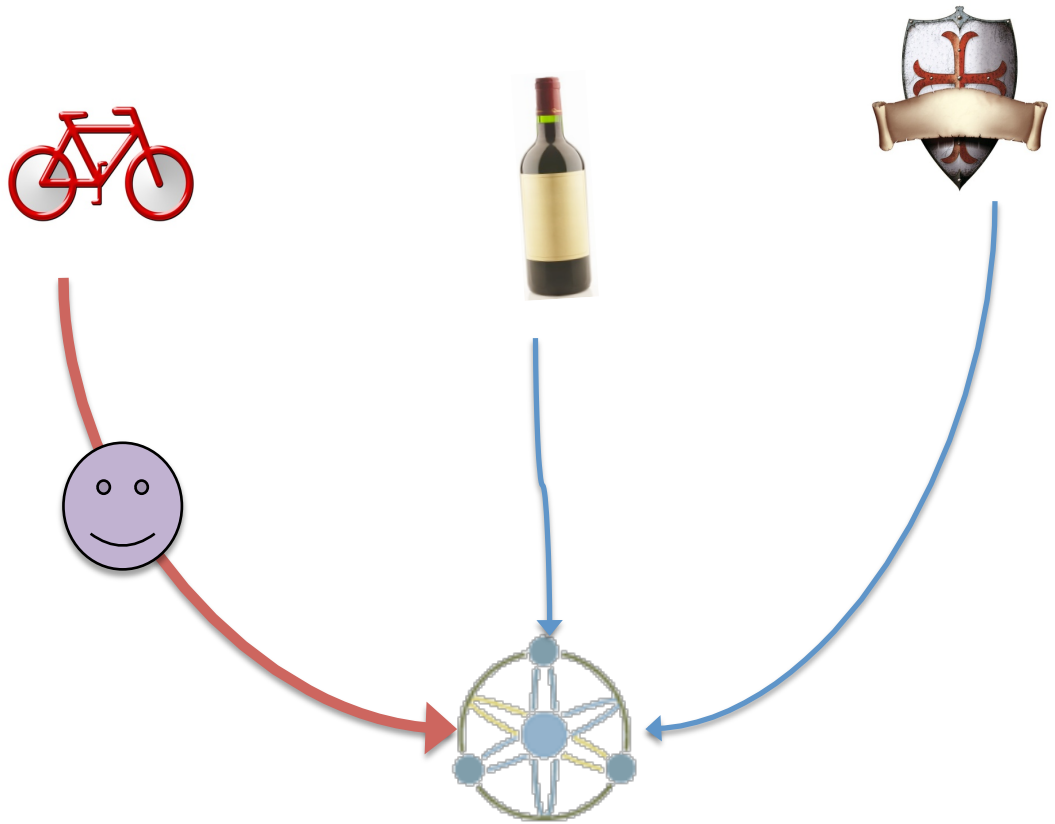


When a new profile is ***created***, the site should **POST** it to its collection URI:



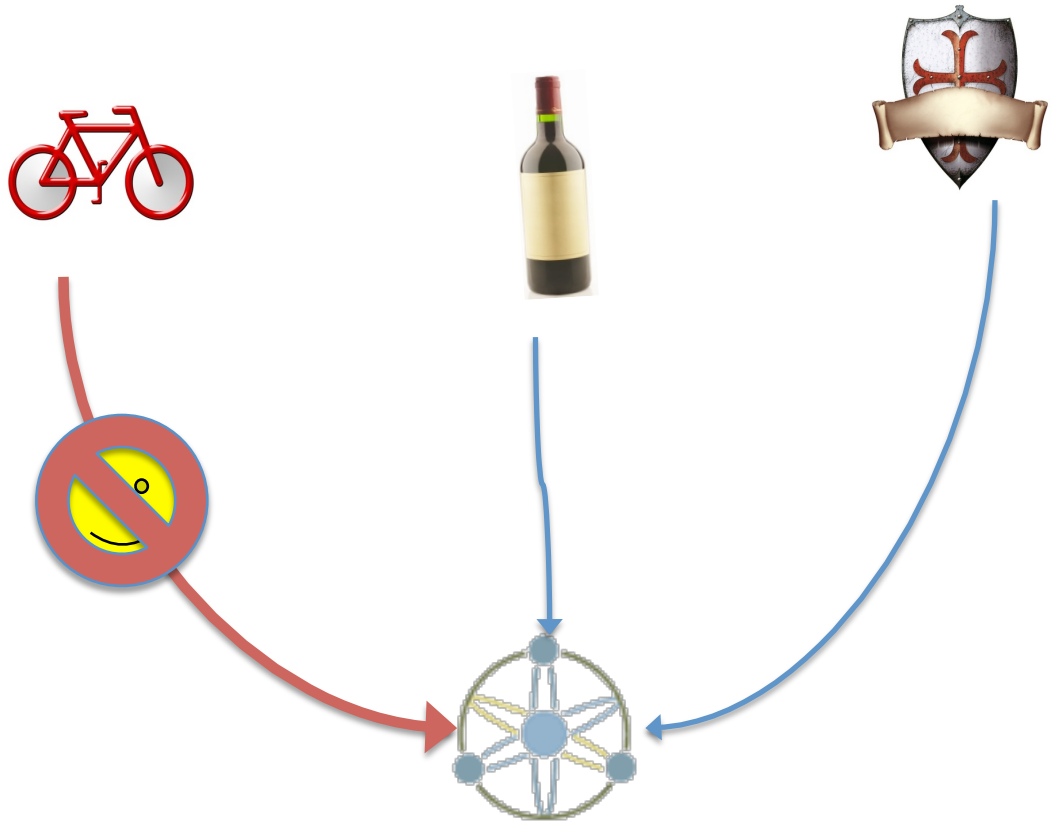
POST
http://myatomserver/profiles/bike

When a profile is ***edited***, the site should **PUT** it to its entry URI:



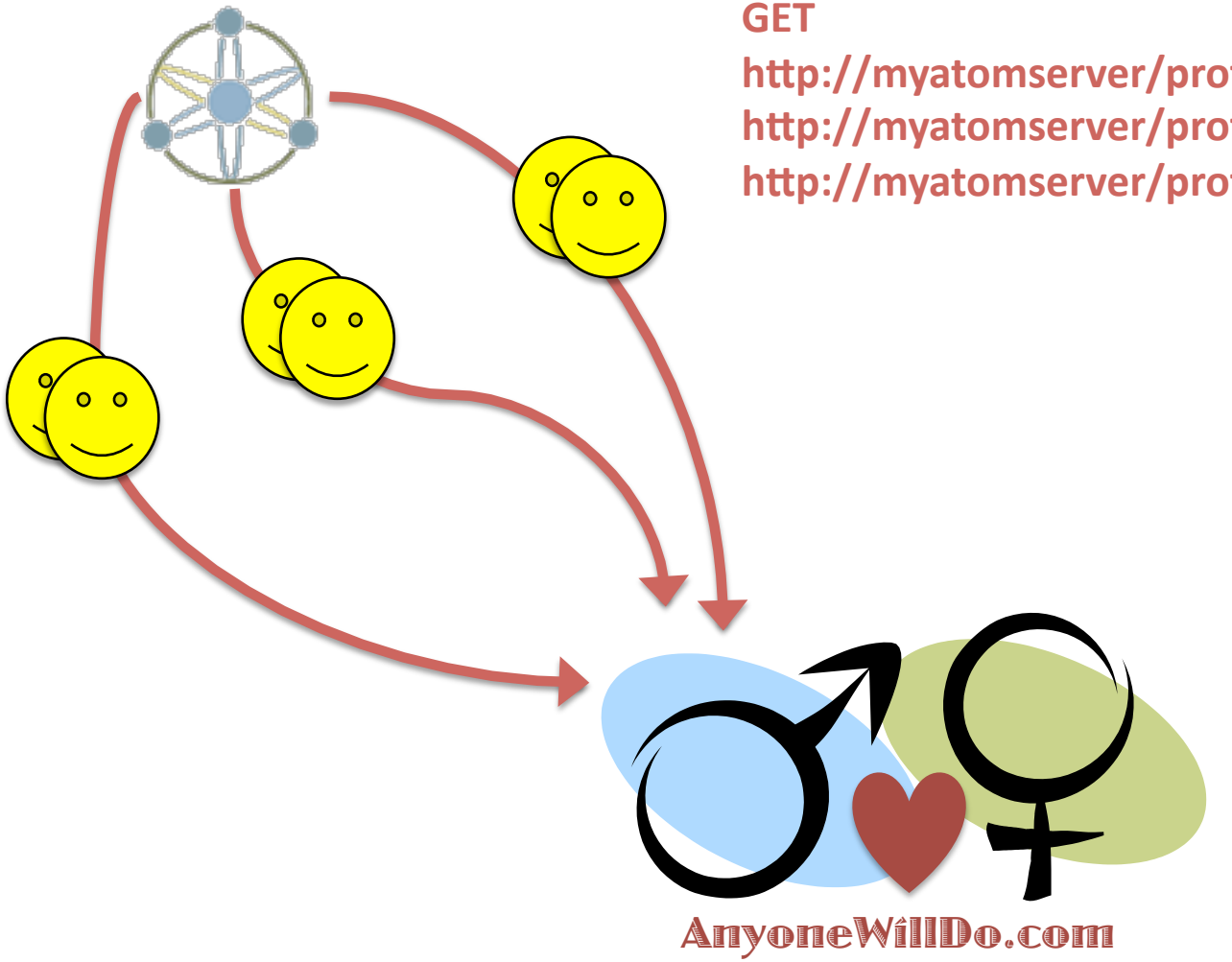
PUT
http://myatomserver/profiles/bike/bob

When a profile is ***removed***, the site should **DELETE** it from its entry URI:



DELETE
http://myatomserver/profiles/bike/bob

Now, our main site can pull feeds of the profiles from the individual niche sites:



“following” feeds

“Following” or “pulling” a feed consists of these steps:

1. Retrieve the first page of the feed
2. Follow the “next” links in the feed pages to iterate through the pages of the feed, until you reach the end
3. Store a “bookmark” from where you left off
4. At some polling interval, start again at step 1 – but start at the stored “bookmark” rather than the first page

“following” feeds

A **GET** of a collection’s URI pulls the first page of the feed associated with that collection:

http://my-server/atom/profiles/larp

“following” feeds

Every time an entry is ***touched*** *(**inserted**, **updated**, or **deleted**)*, it is assigned a new **index** number, which is guaranteed to be larger than ***all*** previously assigned index numbers.

This index number is a **unique**, **discrete** point in time – two entries could have the same updated time, but never the same index value.

These index numbers provide a mechanism to page through a feed. You cannot, in general, make any other assumptions about the index values.

“following” feeds

Collection URIs accept a **start-index** query parameter that uses these index values to drive this paging process – the default value is zero, so:

http://my-server/atom/profiles/larp

is equivalent to

http://my-server/atom/profiles/larp?start-index=0

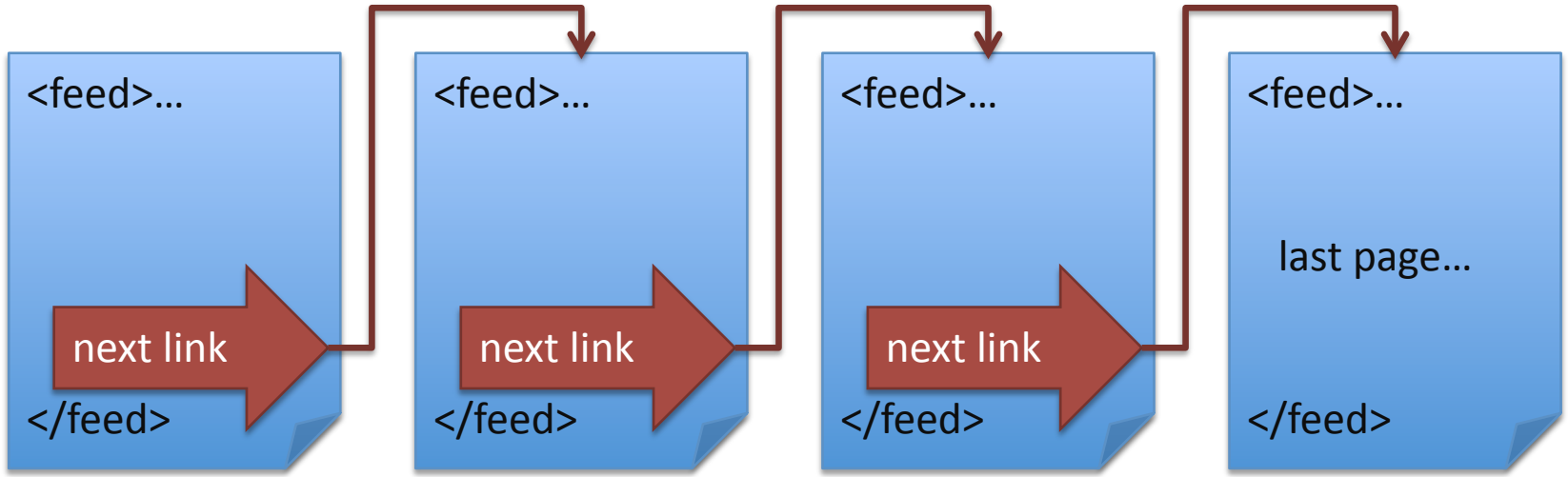
and subsequent pages would look like:

http://my-server/atom/profiles/larp?start-index=73

“following” feeds

Each feed page before the last embeds a “next” link that has the URI for the next page prepared for you:

```
<link rel="next" href="http://my-server/atom/profiles/larp?start-index=73"/>
```



“following” feeds

When you reach the end of a feed (for now) you just keep track of the end index of the last page:

```
<as:endIndex>1034</as:endIndex>
```

*AtomServer specific feature

“following” feeds

Then you supply that as the **start-index** value the next time you wake up to poll the server.

http://my-atom-server/**profiles/larp?start-index=1034**

So, each client to the atom server has to keep track of the index where it left off in each feed.

If there are no new entries, you receive a **304 NOT MODIFIED**, otherwise you get a **200 OK** and body of the response is the next page in the feed.

* **AtomServer specific feature**

“following” feeds

Feed pages are not stable, nor are they neccessarily reproducible.

For example, with a page size of 5, we pull the page at:
http://my-server/atom/profiles/larp?start-index=82&max-entries=5

Which returns entries **A**, **B**, **C**, **D**, and **E**. If someone changes entry **C**, and we pull the same URL again, this time we might get entries **A**, **B**, **D**, **E**, and **F**. This is because, since **C** was changed, it was assigned a higher index value and will now appear on a later page.

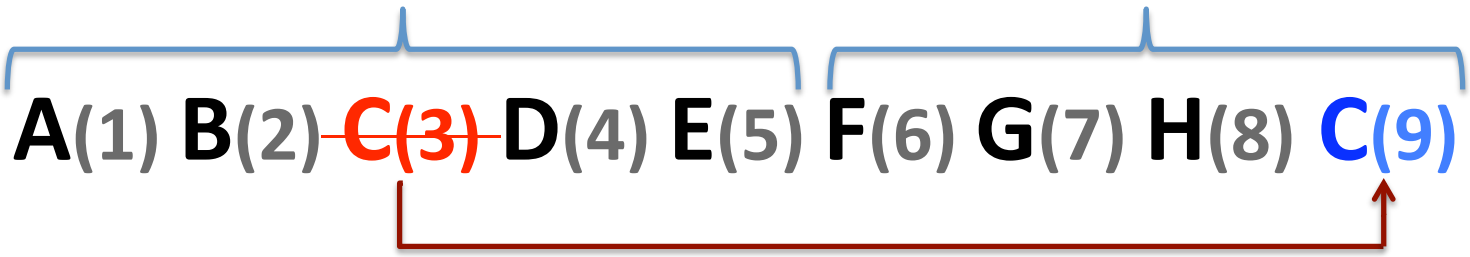
“following” feeds

You might also see the same entry multiple times in the same feed pull – assume the following entries and corresponding index values:

A(1) **B**(2) **C**(3) **D**(4) **E**(5) **F**(6) **G**(7) **H**(8)

“following” feeds

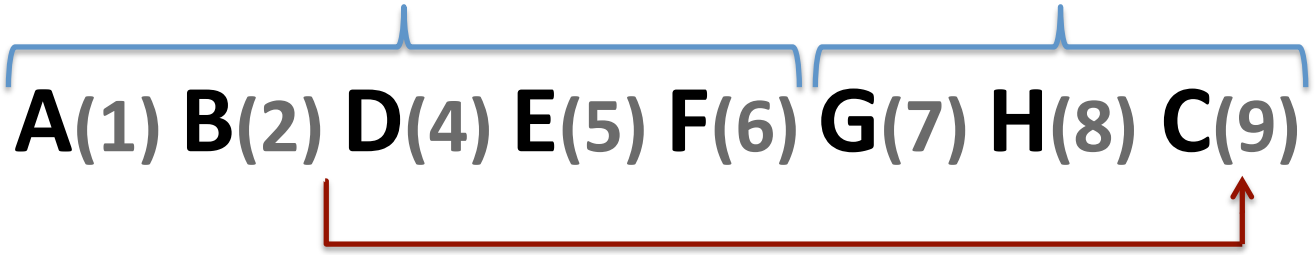
On the first page, you get **ABCDE**. If someone edits **C** while you’re processing that page, it will be assigned a higher index than **H**’s – let’s say **9**.



Now, when you pull the next page, you will see entries **FGHC**. You will have seen **C** twice.

“following” feeds

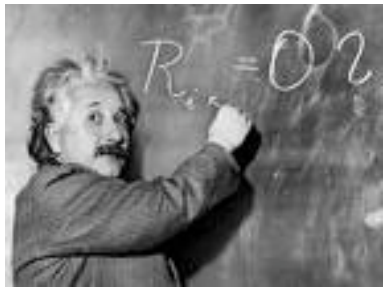
If someone else was to start at the beginning of this feed at this point, they would see **ABDEF**, followed by **GHC**. They will only get the **second** version of **C**, which is fine – because that version **overwrites** the first.



“following” feeds

In general, you are not guaranteed to see **all** changes made to an entry – but you are guaranteed that if you follow a feed to the end, you will see the **last** edit of every entry up to that point in time.

This means that you will be “in sync” with the Atom Store every time you follow a particular feed to the end.



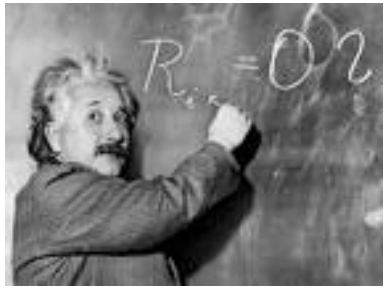
Professor Atom Says:

Beyond the basics. *Atom Categories*

- One of the most powerful, yet overlooked features in Atom
- Atom provides for the arbitrary categorization of *entries*.
- Atom **categories** are **tags** that can be applied to an entry (and we will use those two words interchangably in this talk)

Entries may contain *categories*

```
<feed xmlns=http://www.w3.org/2005/Atom xmlns:as="http://atomserver.org/namespaces/1.0/">
 <as:endIndex>7</as:endIndex>
 <author><name>AtomServer APP Service</name></author>
 <title type="text">dogs entries</title>
 <updated>2008-10-06T23:22:27.073Z</updated>
 <id>tag:atomserver.org,2008:v1:dogs</id>
 <entry>
  <id>/atomserver/v1/pets/dogs/fido.xml</id>
  <title type="text"> Entry: dogs fido</title>
  <author><name>AtomServer APP Service</name></author>
  <link href="/atomserver/v1/pets/dogs/fido.xml/3" rel="edit" />
  <as:entryId>fido</as:entryId>
  <as:updateIndex>27</as:updateIndex
  <as:revision>2</as:revision>
  <updated>2008-10-06T23:22:27.071Z</updated>
  <published>2008-10-06T23:22:27.071Z</published>
  <category scheme="urn:pets.owner" term=”123" />
  <category scheme="urn:pets.breed" term=”Mixed" />
  <link href=“pets/dogs/fido.xml”  rel=“alternative”>
</entry>
 <entry>
  <id>/atomserver/v1/pets/dogs/sparky.xml</id>
  .....
```



Professor Atom Says:

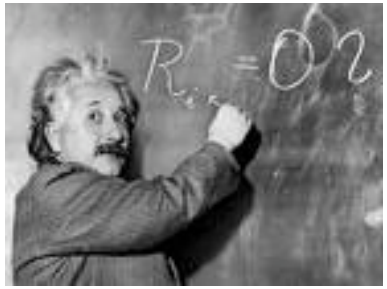
- A **category** has three parts:
  - A **scheme** (this is a URN, and is used as a **namespace**)
  - A **term** (the data of the tag)
  - An optional **label** (free-text about the category)

In an entry's XML representation:

```
<category scheme="urn:sites" term="wine" />
```

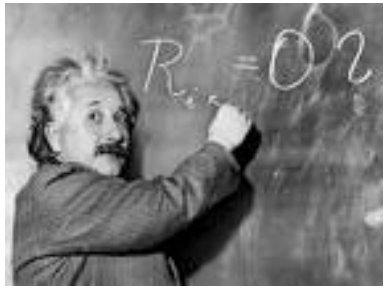
In a URI:

*http://my-server/atom/ws/coll/-/***(urn:scheme)term**



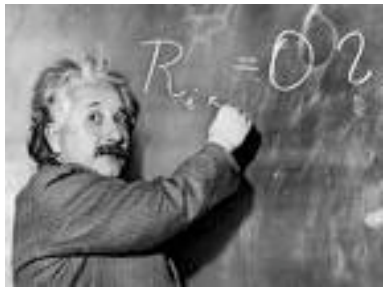
Professor Atom Says:

- An entry can have an arbitrary set of categories
- The Atom specification assigns no meaning to the content of the category element – leaves an opportunity for implementers to assign any meaning they choose.
- There is no notion of mutual exclusion within a scheme – Several categories with the same scheme can be repeated within an entry



Professor Atom Says:

- However, you can *choose* to treat categories more restrictively, say, as key/value pairs.
- Enables the ability to assign new meaning to your data without altering the data itself – by attaching it as metadata on the entry.
- Think of this as a “mash up” for data.

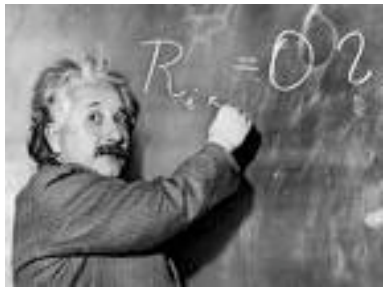


Professor Atom Says:

How do we *apply* categories?

1) By inserting them directly into an entry's XML and then **POST** or **PUT** that back into the collection.

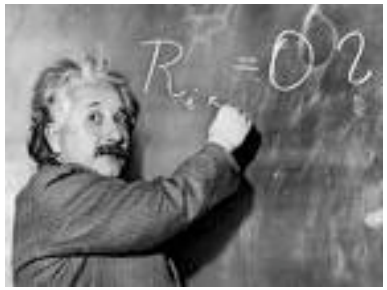
   The Atom server will then apply each category it finds in the entry XML to that entry's metadata.



Professor Atom Says:

PUT http://myserver/pets/dogs/fido

```
<entry xmlns="http://www.w3.org/2005/Atom" >
   <id>/pets/dogs/fido.xml</id>
   <category scheme="urn:owner" term="cberry" />
   <category scheme="urn:breed" term="Mixed" />
   <content  type="application/xml" >
  <pet type="dog" name="fido">
     <breed>Mixed</breed>
     <owner>cberry</owner>
  <pet>
   </content>
</entry>
```



Professor Atom Says:

2) Construct a **categories** document and then **PUT** it to "virtual ***tags:*** *workspace*" for a given entry

– This is an AtomServer specific feature, although the <categories/> document is defined by Atom.

Example – PUTTING a categories document to the URI:

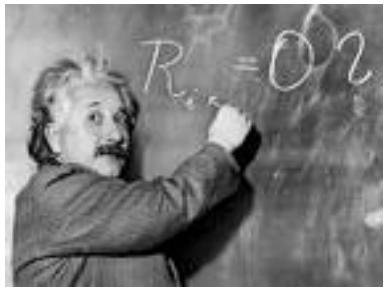
/tags:pets/dogs/fido

changes categories on the Entry with the URI:

/pets/dogs/fido

– Doing this will modify ONLY the categories, without changing the entry content

*AtomServer **specific feature**

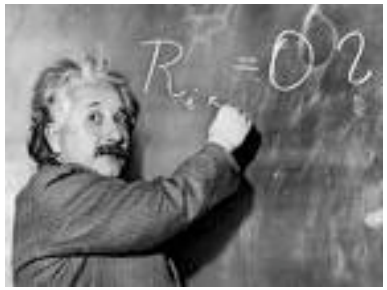


Professor Atom Says:

PUT http://myserver/tags:pets/dogs/fido

```
<categories xmlns="http://www.w3.org/2005/Atom" >
    <category scheme="urn:owner" term="cberry" />
    <category scheme="urn:breed" term="Mixed" />
</categories>
```

*AtomServer specific feature

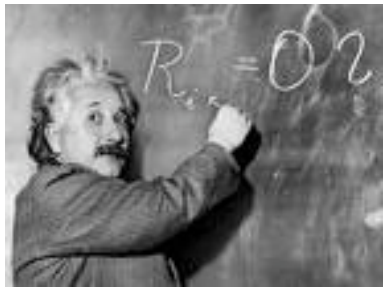


Professor Atom Says:

3) Create ***autotagger*** rules to tag entries based on values extracted from the content

- Another AtomServer specific feature.
- Automatically computes categories, based on the <content> of each entry that is PUT/POST.
- AtomServer comes with **XPathAutoTagger**

*AtomServer **specific feature**

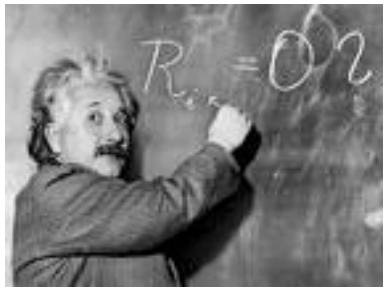


Professor Atom Says:

PUT http://myserver/pets/dogs/fido

```
<entry xmlns="http://www.w3.org/2005/Atom" >
    <id>/pets/dogs/fido.xml</id>
    <content  type="application/xml" >
   <pet type="dog" name="fido">
      <breed>Mixed</breed>
      <owner>cberry</owner>
   <pet>
    </content>
</entry>
```

*AtomServer specific feature



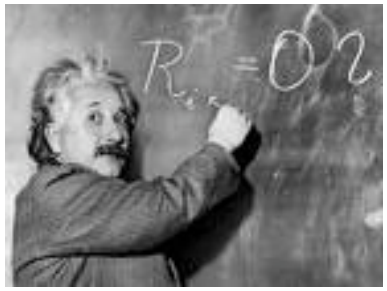
Professor Atom Says:

PUT http://myserver/pets/dogs/fido

```
<entry xmlns="http://www.w3.org/2005/Atom" >
    <id>/pets/dogs/fido.xml</id>
    <content  type="application/xml" >
   <pet type="dog" name="fido">
      <breed>Mixed</bre
      <owner>cberry</ow
   <pet>
    </content>
</entry>
```

XPathAutoTagger:
/pet/owner

* **AtomServer specific feature**



Professor Atom Says:

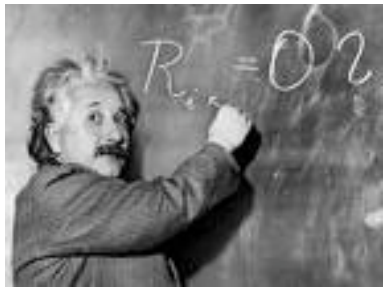
PUT http://myserver/pets/dogs/fido

```
<entry xmlns="http://www.w3.org/2005/Atom" >
    <id>/pets/dogs/fido.xml</id>
    <content  type="application/xml" >
   <pet type="dog" name="fido">
      <breed>Mixed</bre
      <owner>cberry</ow
   <pet>
    </content>
</entry>
```

XPathAutoTagger:
/pet/owner

Category Template:
`(urn:owner)$value`

* **AtomServer specific feature**



Professor Atom Says:

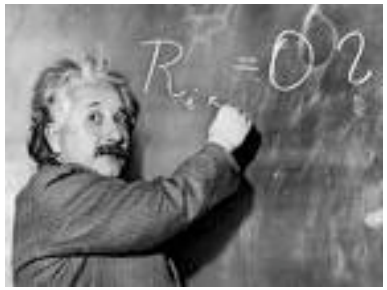
PUT http://myserver/pets/dogs/fido

```
<entry xmlns="http://www.w3.org/2005/Atom" >
    <id>/pets/dogs/fido.xml</id>
    <content  type="application/xml" >
   <pet type="dog" name="fido">
      <breed>Mixed</bre
      <owner>cberry</ow
   <pet>
    </content>
</entry>
```

XPathAutoTagger:
/pet/owner

Category:
(urn:owner)cberry

* **AtomServer specific feature**

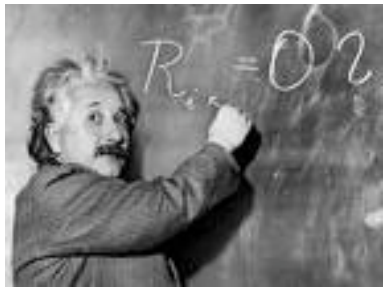


Professor Atom Says:

Autotaggers take the burden off the clients to explicitly create and manage categories.

This is crucial for categories that are critical to the system's integrity.

Usable anywhere that the information in an entry's categories can be ***deduced*** from it's content.

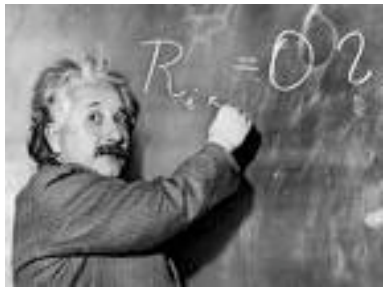


Professor Atom Says:

Category Feeds

- A special Feed syntax that lets clients filter a Feed by the Categories applied to Entries
- AtomServer feature – modeled after the Category Feeds in GData (with slightly different syntax)
- A Category Feed is by appending "/-/" followed by the categories in the query to a standard feed URI

*AtomServer specific feature



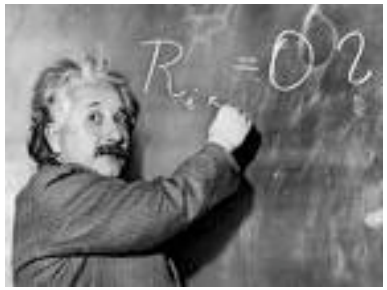
Professor Atom Says:

The simplest form of a category query is a single category:

http://my-server/atom/pets/dogs/-/(urn:owner)cberry

"retrieve all the dogs tagged with ***(urn:owner)cberry****"*

*AtomServer **specific feature**



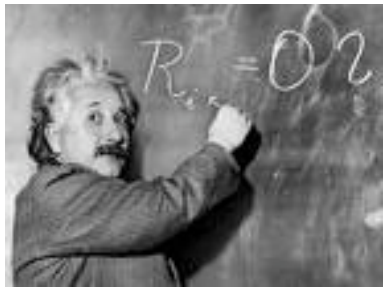
Professor Atom Says:

A list of several categories is treated as a **conjunction**:

http://my-server/atom/pets/dogs/-/(urn:owner)cberry/(urn:breed)Mixed

"retrieve all the dogs tagged with ***(urn:owner)cberry*** *AND* ***(urn:breed)Mixed****"*

* **AtomServer specific feature**



Professor Atom Says:

Or, you can do more complex Boolean logic with AND and OR, in prefix notation:

http://my-server/atom/pets/dogs/-/AND/(urn:owner)cberry/OR/(urn:breed)Mixed/(urn:breed)Beagle

*"retrieve all the dogs tagged with **(urn:owner)cberry** AND either **(urn:breed)Mixed** or **(urn:breed)Beagle**"*

* **AtomServer specific feature**

Business objectives again…

✓Get our main site to critical mass, by combining data from our acquisitions?

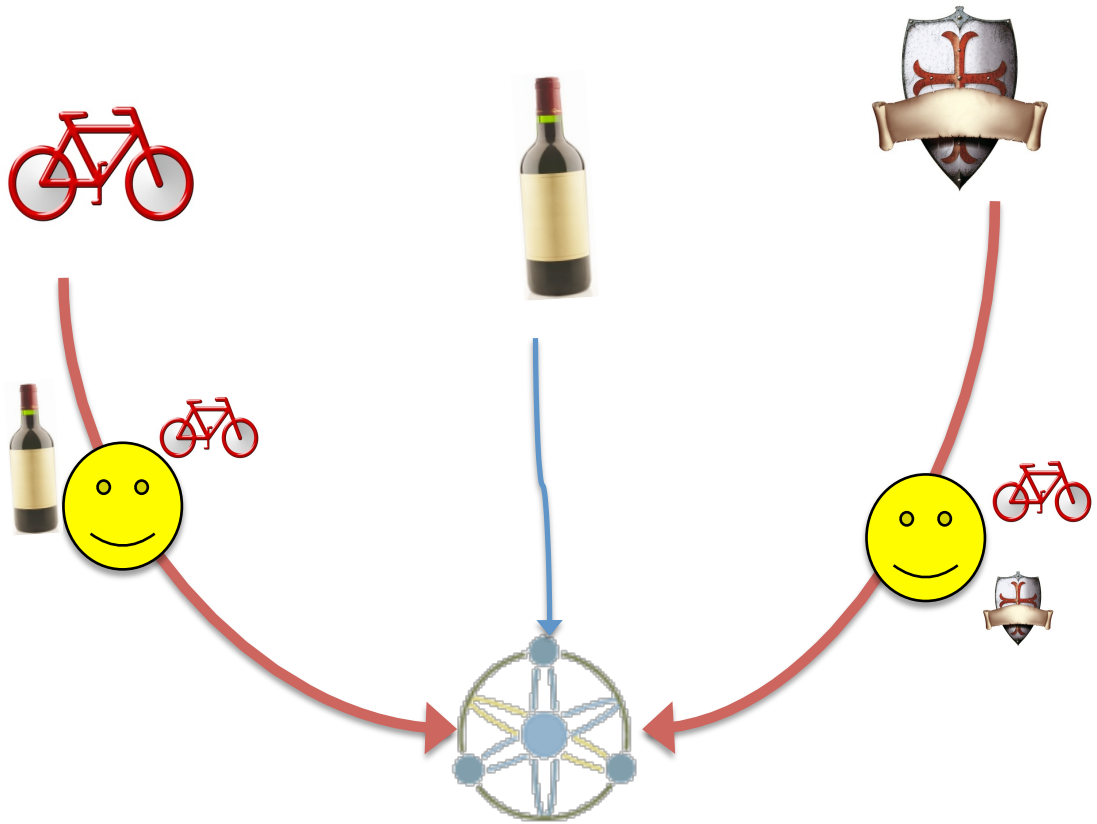
Increase revenue by “cross-selling” customers beteen our sites?

Simplify our business processes with a SOA that’s streamlined by Atom and AtomServer?

Increase revenue by “cross-selling” customers beteen our sites?

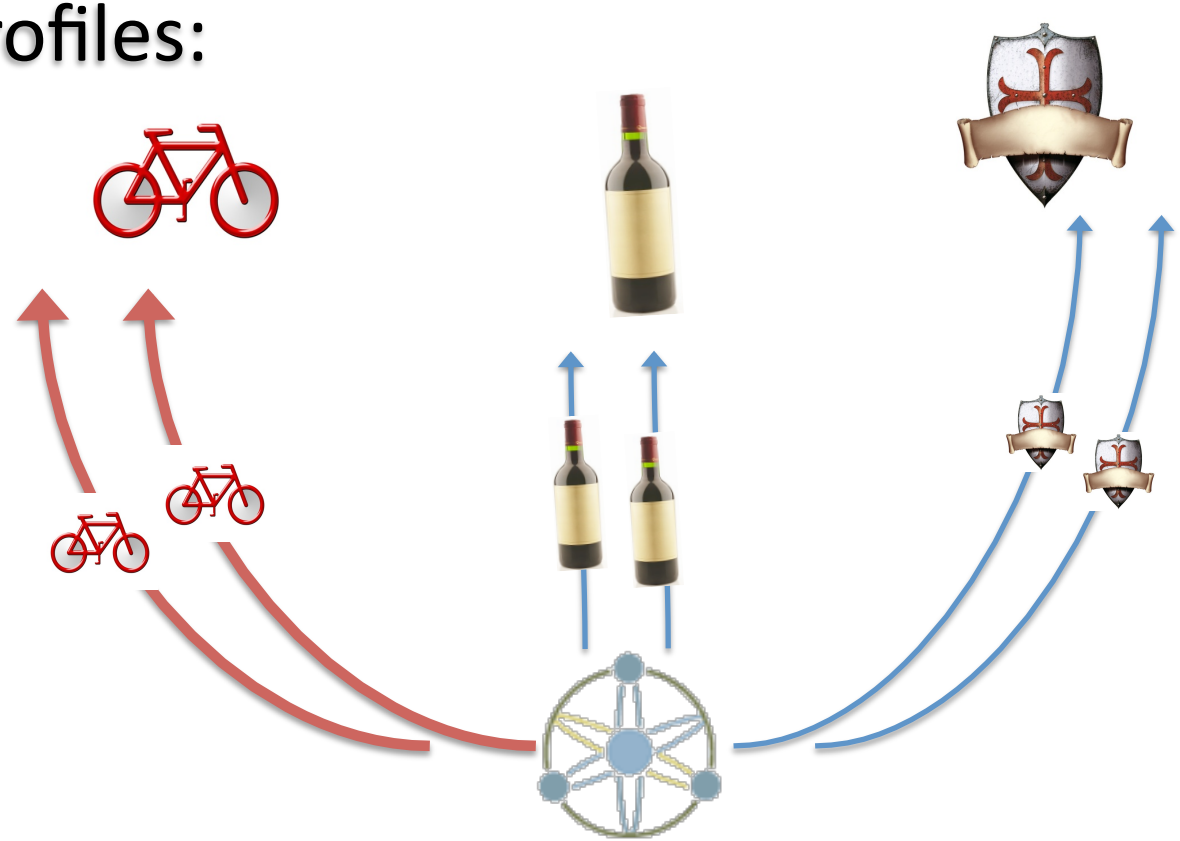
We could use **categories** on our profile objects to indicate which of our sites customers want their profiles to appear on.

We'd like to leverage our sites by **cross-selling** our customers who have multiple interests:



each profile can be **tagged** with one or more **categories** – which will indicate which sites they've paid to show up on.

Then, each site can then pull categorized feeds from the other sites with relevant tagged profiles:



GET
http://myatomserver/profiles/larp/-/(urn:site)bike
http://myatomserver/profiles/wine/-/(urn:site)bike

- Now we can move our profiles around between all of our sites easily
- Is there other data that we'd like to distribute?
- **Rendezvous** objects represent a potential "date" between two people.
- If I have an account across multiple sites, I would like to see all my rendezvous on any of the sites when I log in

Same as before, we define an XML format for a rendezvous:

```
<date:rendezvous>
  <requestor>/profiles/bike/bob</requestor>
  <requestee>/profiles/bike/sue</requestee>
  <where>Slanted Door</where>
  <when>2008-12-15 19:00:00</when>
  <status>REQUESTED</status>
</date:rendezvous>
```

```
<date:rendezvous>
  <requestor>/profiles/bike/bob</requestor>
  <requestee>/profiles/wine/sue</requestee>
  <where>Slanted Door</where>
  <when>2008-12-15 19:00:00</when>
  <status>REQUESTED</status>
</date:rendezvous>
```

other Atom objects are referenced by URI

```
<atom:entry>
  <content>
    <date:rendezvous>
      <requestor>/profiles/bike/bob</requestor>
      <requestee>/profiles/wine/sue</requestee>
      <where>Slanted Door</where>
      <when>2008-12-15 19:00:00</when>
      <status>REQUESTED</status>
    </date:rendezvous>
  </content>
  <category scheme="urn:site" term="bike"/>
  <category scheme="urn:site" term="wine"/>
  <category scheme="urn:site" term="anyone"/>
</atom:entry>
```

Rendezvous are **categorized** for which sites they should appear on, just like profiles

We set up a new workspace, **rendezvous**, for our rendezvous objects:

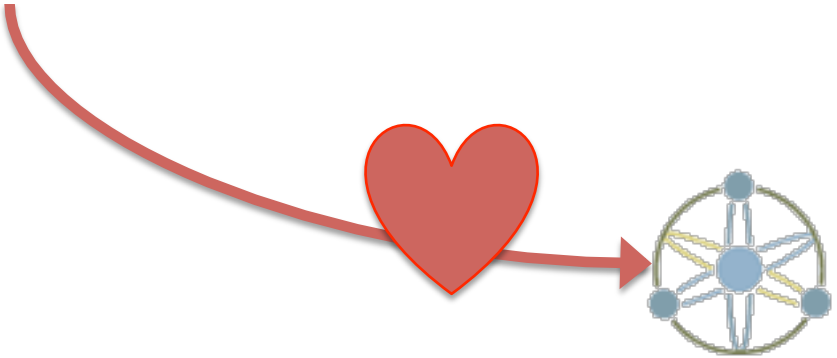
http://my-server/atom/rendezvous

We **could** have a separate collection for each site (as with profiles), or we can put them all in one collection and use **categories** to shepherd the rendezvous objects to the right places.

Bob requests a **rendezvous** with Sue from **loveontwowheels.com**…

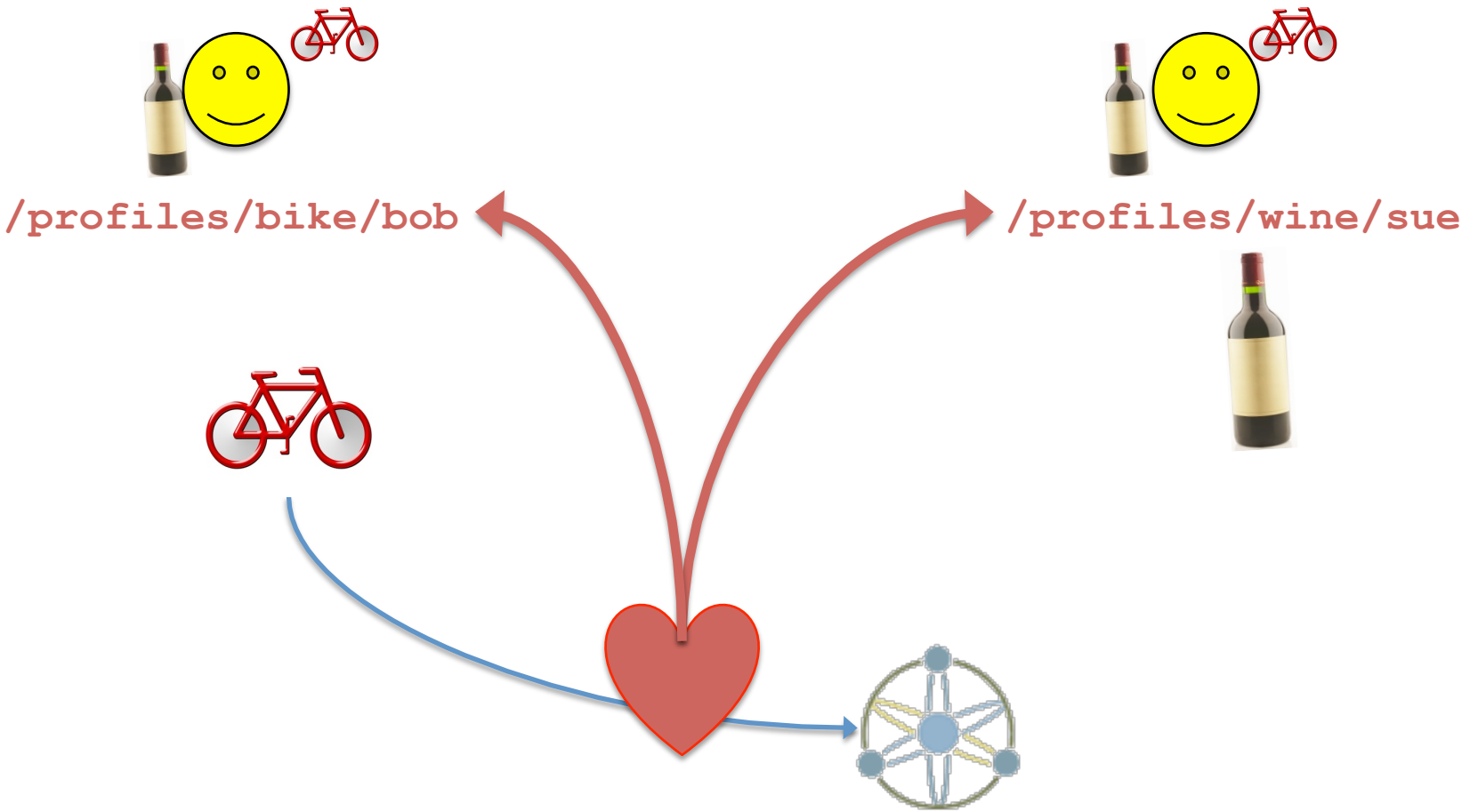
`/profiles/bike/bob`

`/profiles/wine/sue`



POST
http://my-server/atom/rendezvous/all

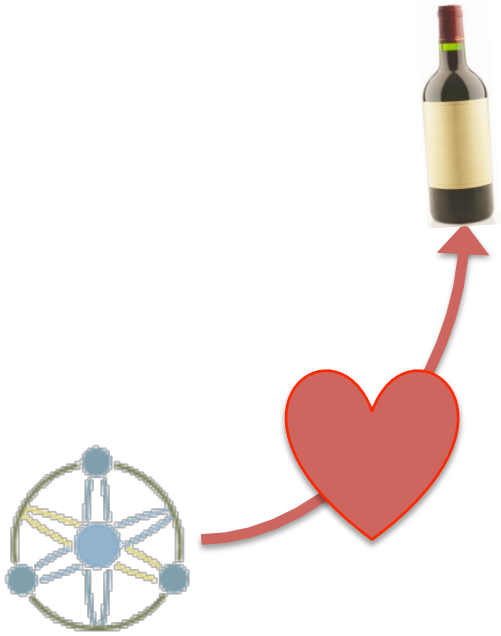
The **rendezvous** object ***references*** both Bob's and Sue's profiles



The rendezvous object is pulled from the AtomServer for Sue on **winesnobmatch.com**

`/profiles/bike/bob`

`/profiles/wine/sue`



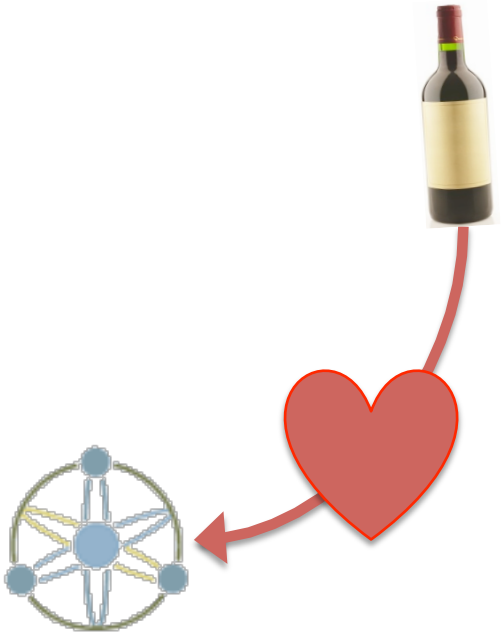
GET

http://my-server/atom/rendezvous/all/-/(urn:site)wine

Sue wants to accept Bob's request, so she updates the **status** to ACCEPTED.

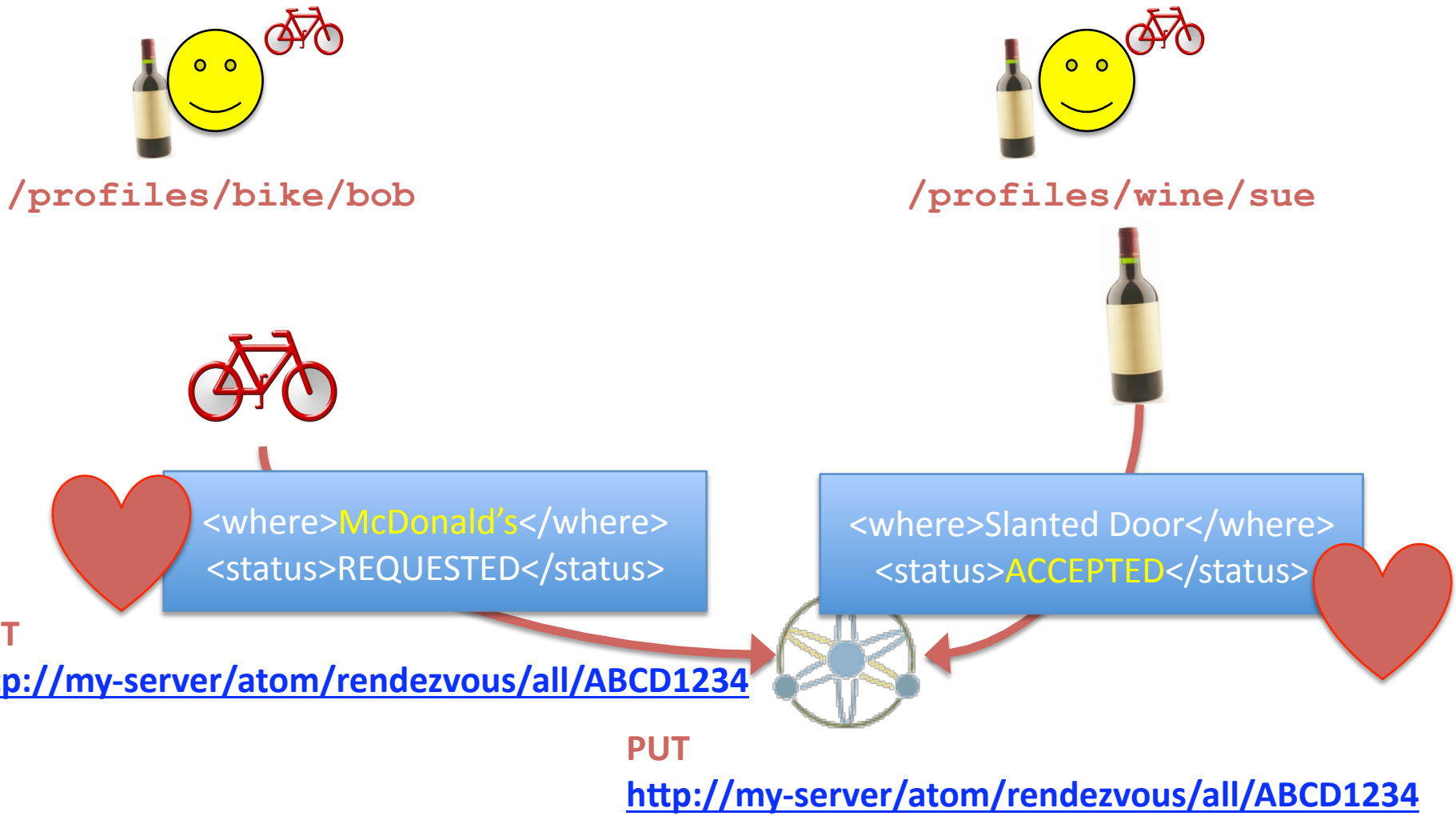
`/profiles/bike/bob`

`/profiles/wine/sue`



PUT
http://my-server/rendezvous/all/ABCD1234

But what if Bob changed his mind about the location at the same moment Sue clicked?



Object state is kept consistent between multiple writers using **Optimistic Concurrency**

- Each entry has a **revision number**, which is incremented on every update
- Edit operations (**PUT** and **DELETE**) must have the revision number appended to the URI:

```
http://my-server/atom/rendezvous/all/ABCD1234/3
```

*AtomServer specific feature

If the **PUT** or **DELETE** is to the correct revision, the update happens, the revision number is incremented, and a **200 OK** is returned.

If one tries to **PUT** or **DELETE** to the wrong revision, it rejects the update and returns a **409 CONFLICT** HTTP error code.

* **AtomServer specific feature**

Entry elements embed an **edit link** that has the edit URI with the correct revision number:

```
<link rel="edit" href="/rendezvous/all/ABCD1234/3"/>
```

* **AtomServer specific feature**

All updates are done **serially**, so if two writers try to edit a resource simultaneously, one is always done **first**, and that one succeeds.

On success, the resource's revision number is incremented.

The other one will now be trying to edit with an out-of-date revision number, and so will fail.

* **AtomServer specific feature**

The loser should then do a **GET** of the entry to refresh it's most current state, make sure to merge the failed changes with the updated entry, and re-try the edit to the new edit URI

```
<link rel="edit" href="/rendezvous/all/ABCD1234/4"/>
```

*AtomServer specific feature

So, with **Optimistic Concurrency**, Bob and Sue are each trying to **PUT** to revision **1**

`/profiles/bike/bob`

`/profiles/wine/sue`

<where>McDonald's</where>
<status>REQUESTED</status>

<where>Slanted Door</where>
<status>ACCEPTED</status>

PUT
http://myatomserver/rendezvous/all/ABCD1234/1

PUT
http://myatomserver/rendezvous/all/ABCD1234/1

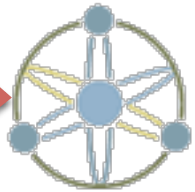
If Bob's happens to be processed first, his venue change is saved, and he gets a **200 OK**

`/profiles/bike/bob`

✔200 OK

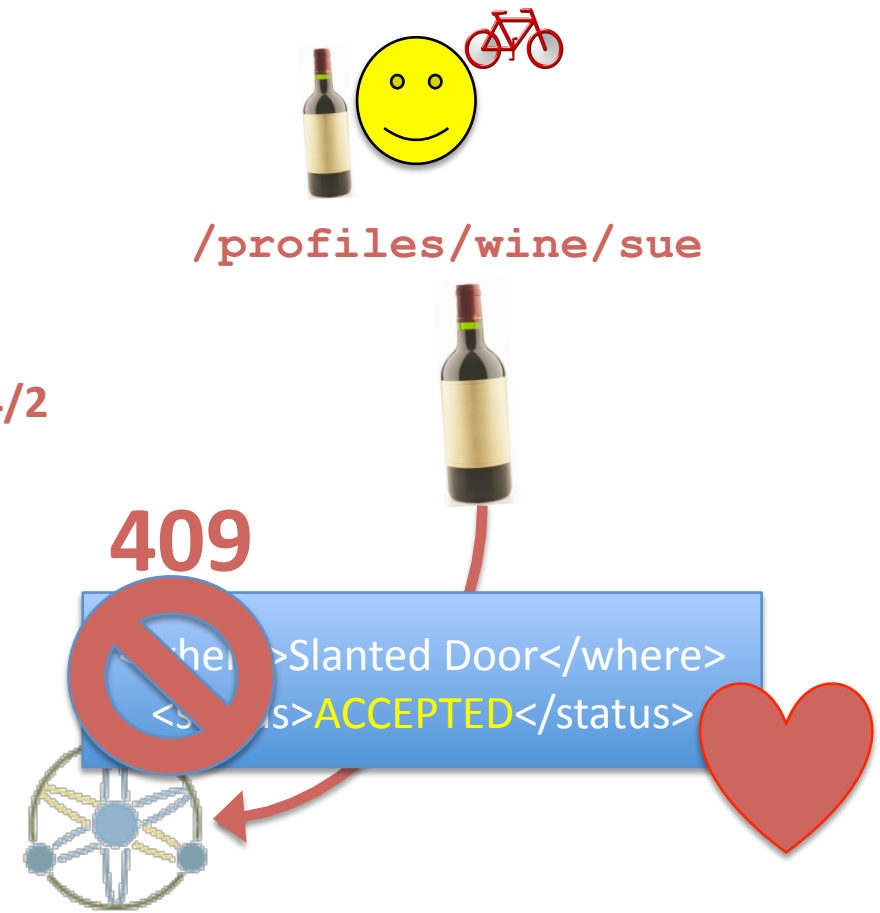
<where>McDonald's</where>
<status>REQUESTED</status>

PUT
http://myatomserver/rendezvous/all/ABCD1234/1



And Sue's request is rejected with the error code

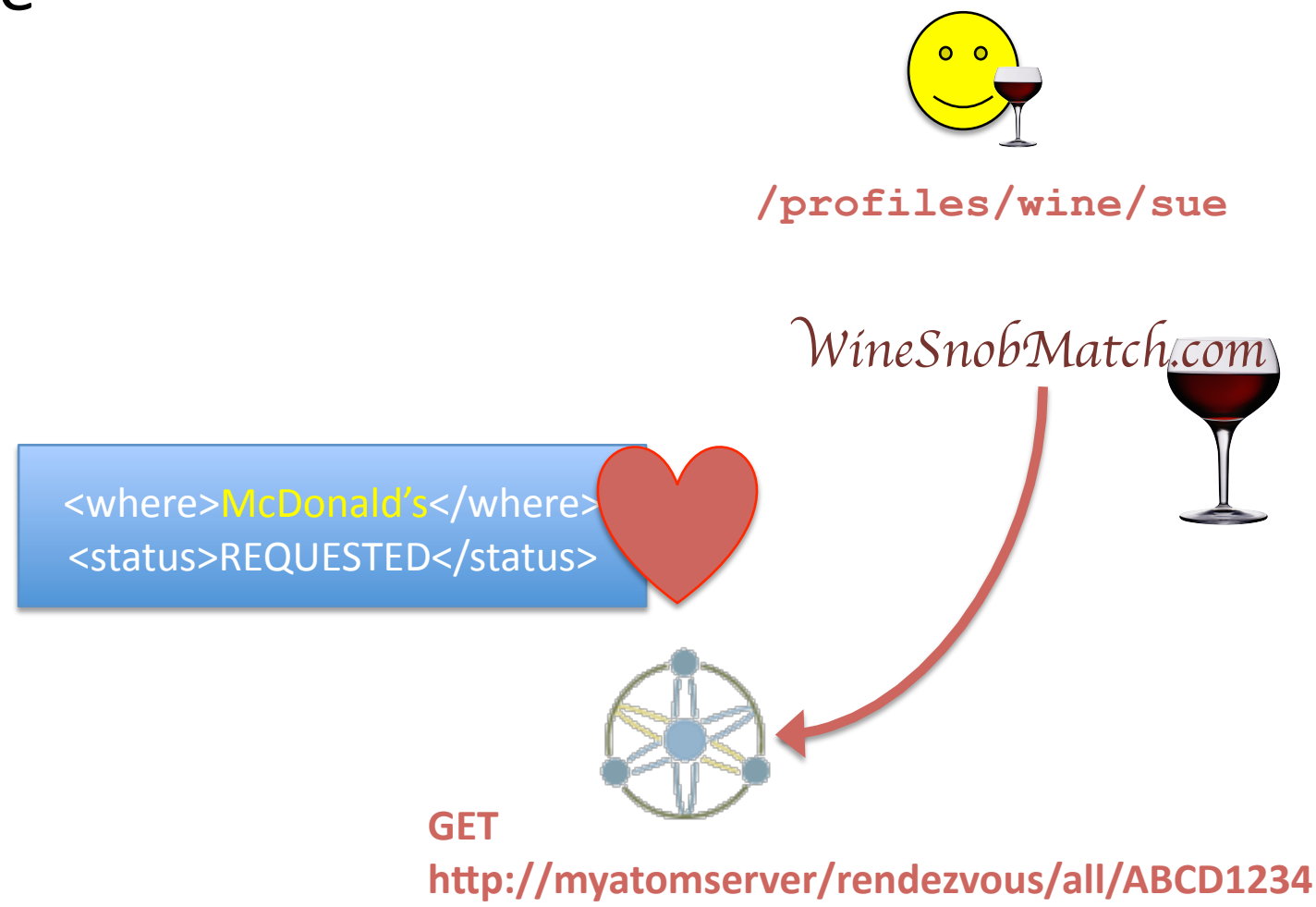
409 CONFLICT



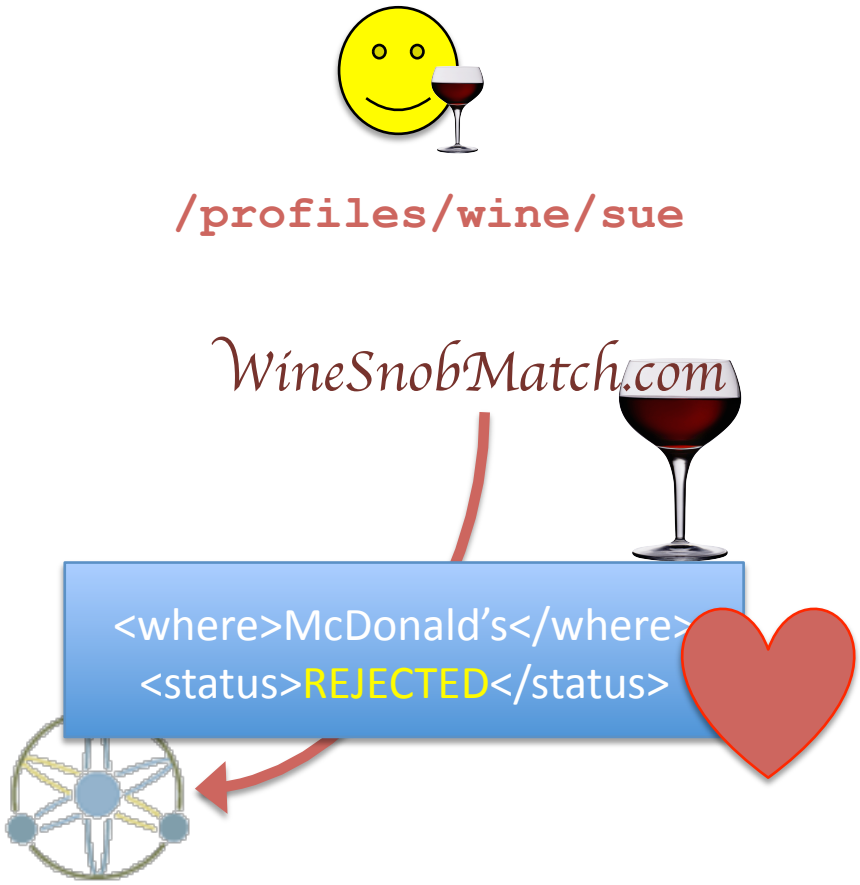
http://myatomserver/rendezvous/all/ABCD1234/2

the response lets her know that she needs to get the updated rendezvous, and it returns her the updated **edit link**

Sue does a **GET** to retrieve the newest revision of the rendezvous, and can see Bob's venue change



Now, she can re-submit her decision on the rendezvous (to the updated edit link):



Which will now succeed.

One last time…

✓Get our main sites to critical mass, by combining data from our acquisitions?

✓Increase revenue by “cross-selling” customers beteen our sites?

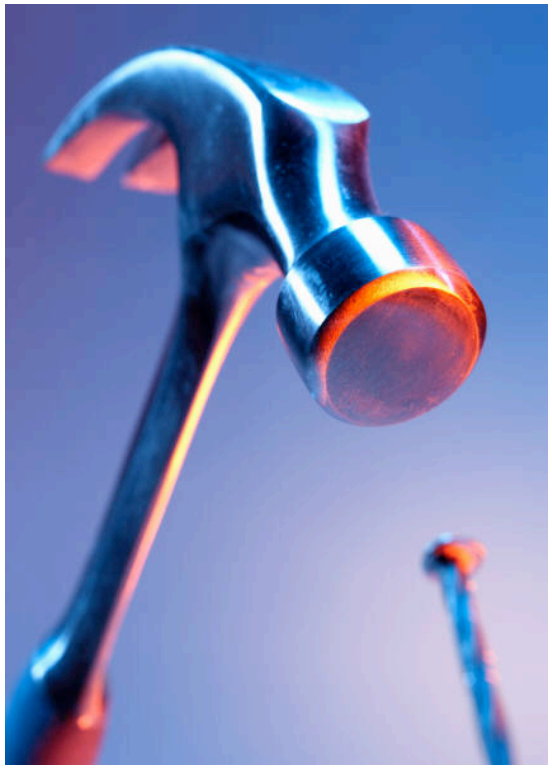
Simplify our business processes with a SOA that’s streamlined by Atom and AtomServer?

Atom Stores

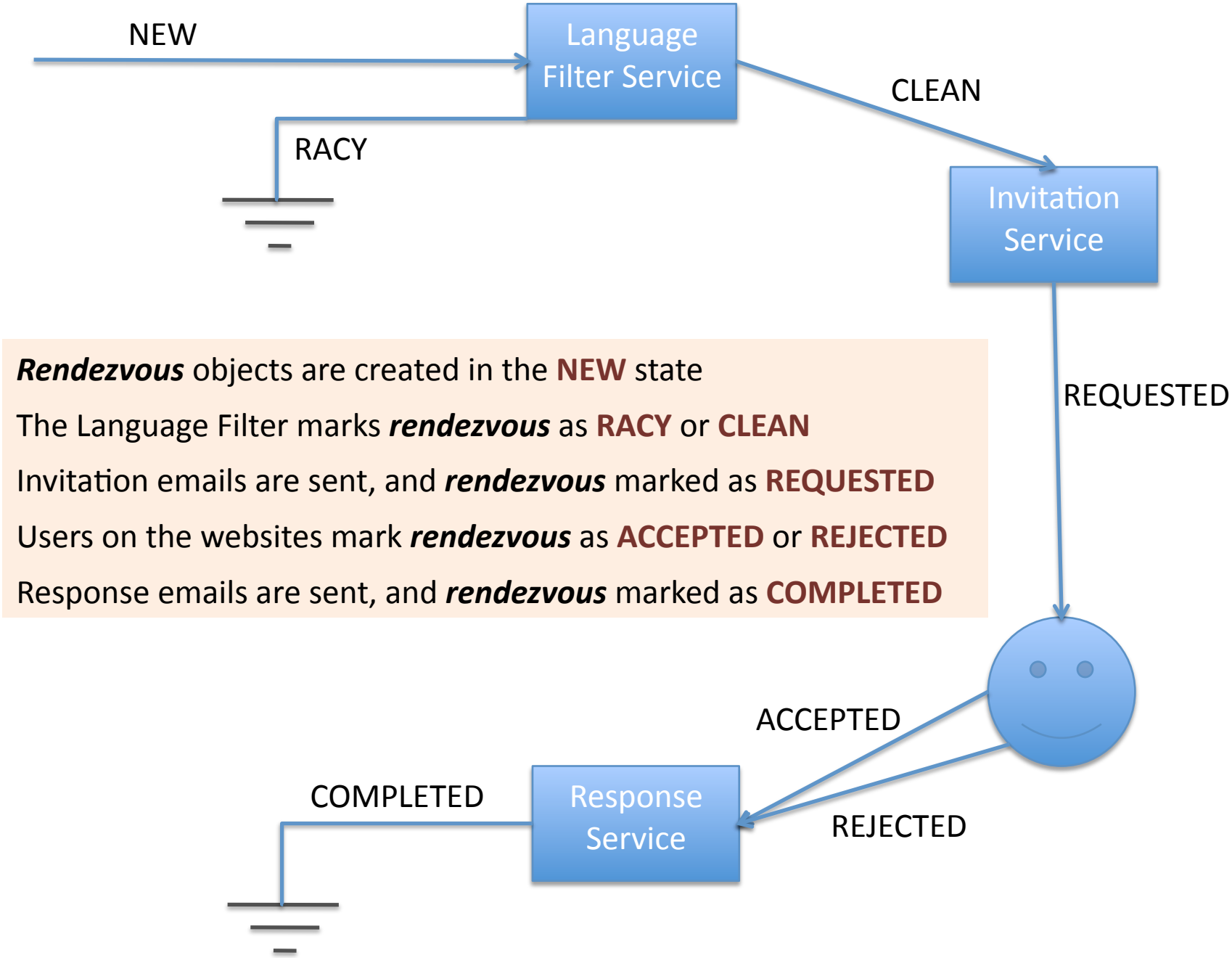
- **Atom Stores** are powerful because of Atom's **simplicity**
  - ✓ RESTful XML over HTTP - easy to integrate
  - ✓ Persistent data store with CRUD operations
  - ✓ Changes are automatically propogated via feeds
  - ✓ Attach arbitrary metadata with categories

- **AtomServer** is an **Atom Store**
  - ✓ Data integrity with optimistic concurrency
  - ✓ Category queries provide searching capabilities

Atom Store Powered SOA



- As our business process gets more complex, we need a strategy to tie together components of our SOA tha implement our business processes
- An **AtomServer** instance is already configured to hold our domain content
- Many types of services can leverage this, and build out a sort of **Atom-Oriented Service Architecture**
- **Example :** let's model the life cycle of a rendezvous object as a state machine, and look at the components that move the rendezvous through the state machine…



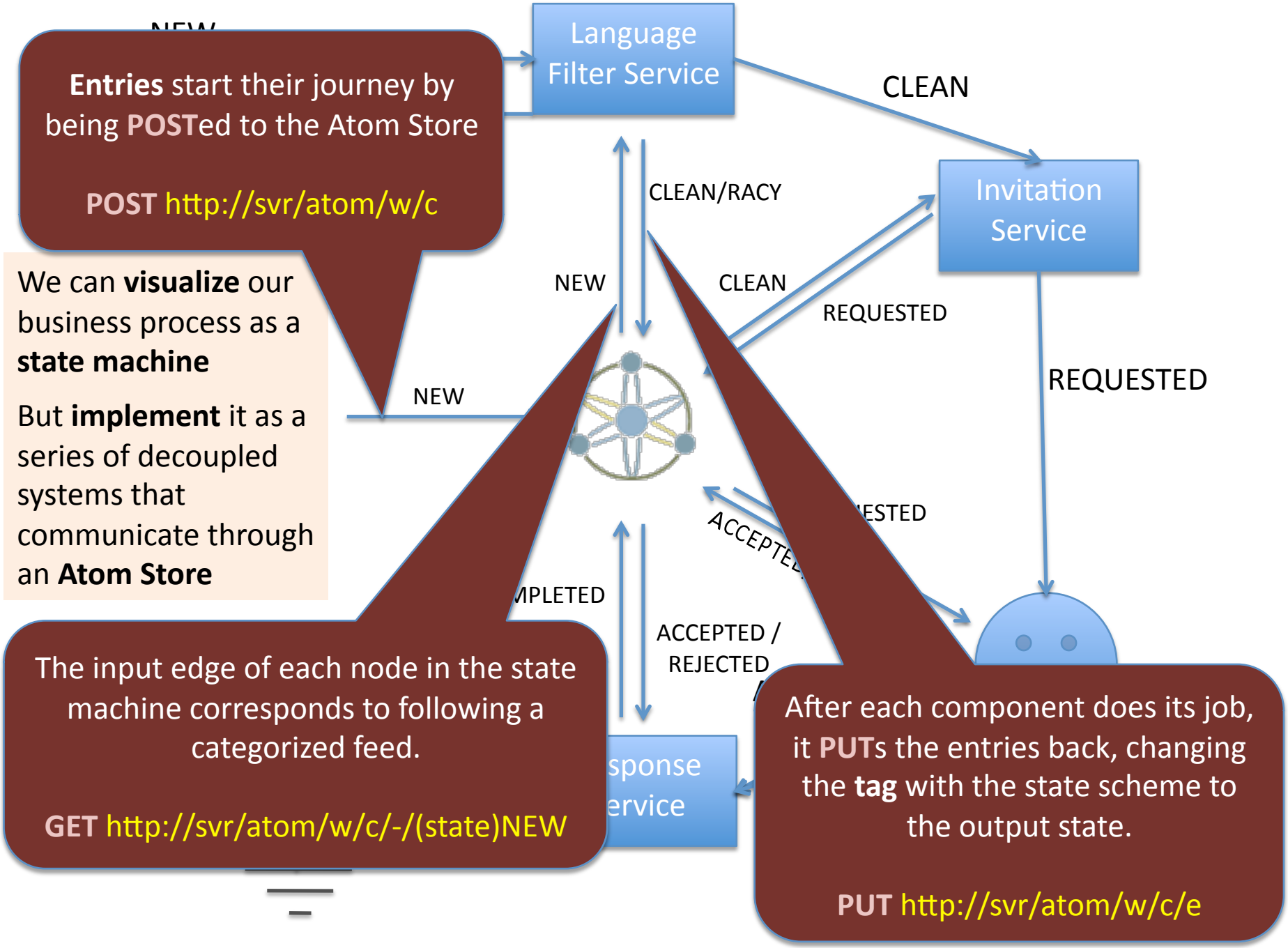
Rendezvous objects are created in the **NEW** state

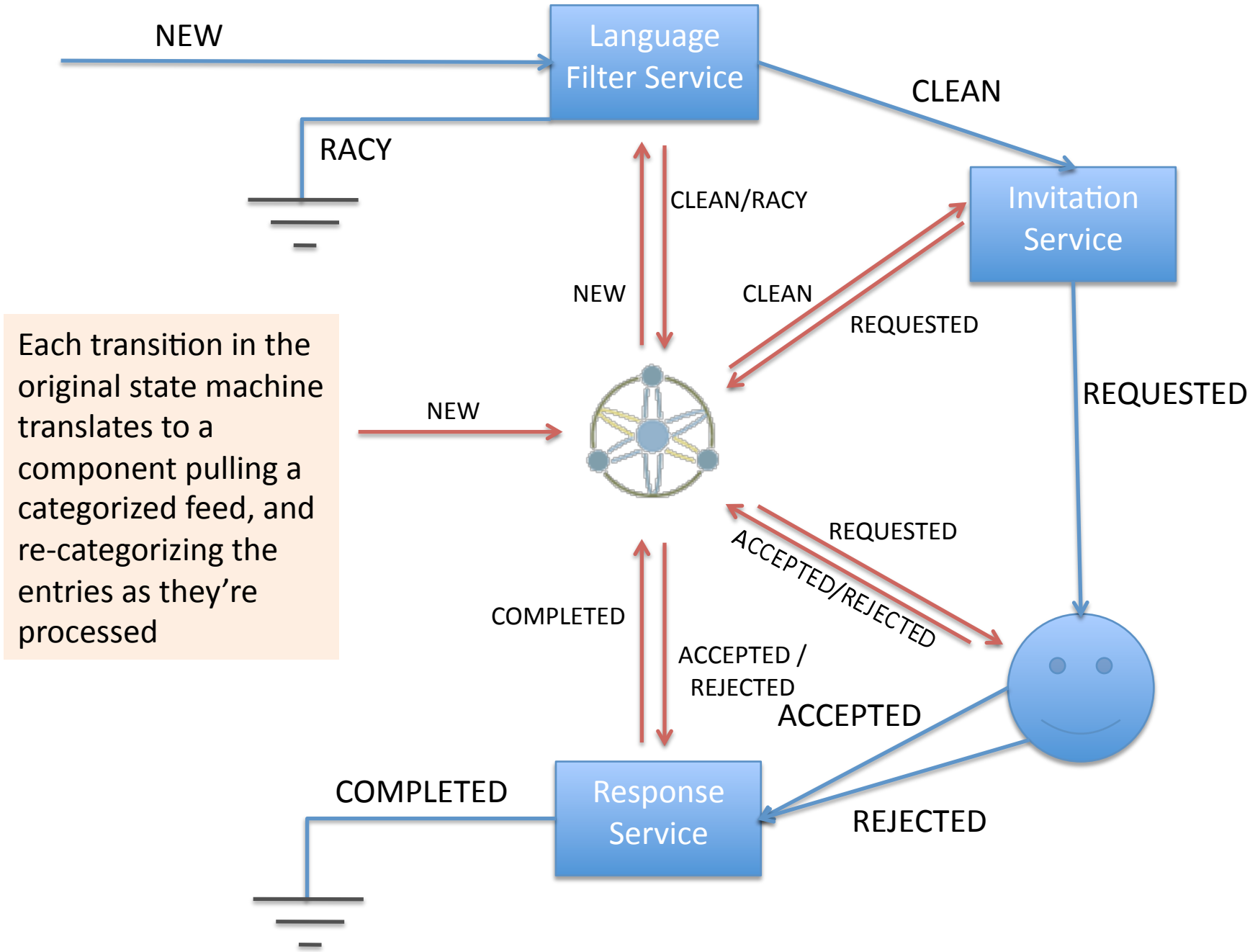
The Language Filter marks ***rendezvous*** as **RACY** or **CLEAN**

Invitation emails are sent, and ***rendezvous*** marked as **REQUESTED**

Users on the websites mark ***rendezvous*** as **ACCEPTED** or **REJECTED**

Response emails are sent, and ***rendezvous*** marked as **COMPLETED**


Language Filter Service
CLEAN
Invitation Service
CLEAN/RACY
NEW
CLEAN
REQUESTED
REQUESTED
NEW
ACCEPTED /
REJECTED
sponse
ervice
Entries start their journey by being POSTed to the Atom Store
POST http://svr/atom/w/c
We can visualize our business process as a state machine
But implement it as a series of decoupled systems that communicate through an Atom Store
The input edge of each node in the state machine corresponds to following a categorized feed.
GET http://svr/atom/w/c/-/(state)NEW
After each component does its job, it PUTs the entries back, changing the tag with the state scheme to the output state.
PUT http://svr/atom/w/c/e

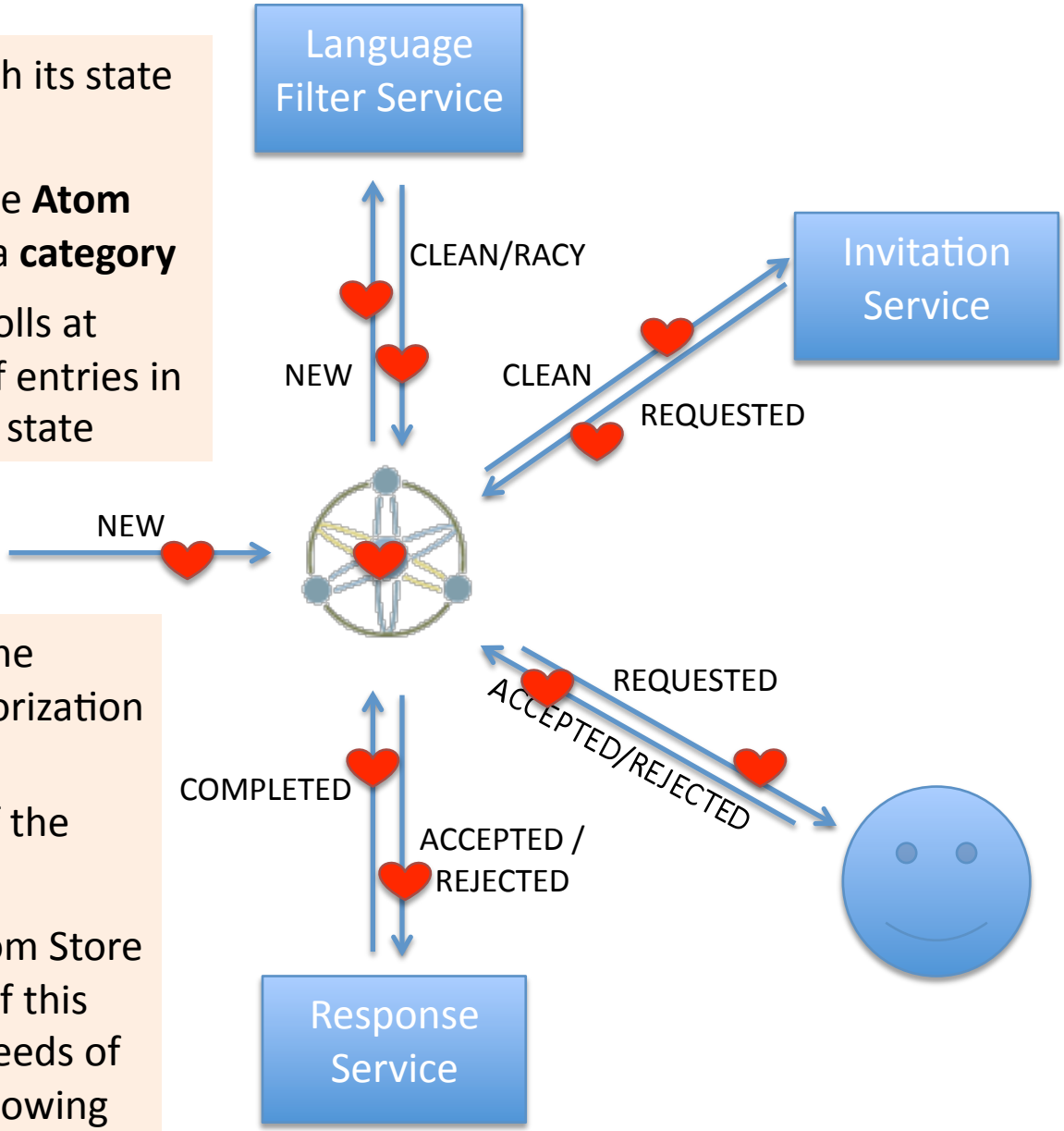


Each transition in the original state machine translates to a component pulling a categorized feed, and re-categorizing the entries as they're processed

Each entry is tagged with its state each time it is written

The entry is stored in the **Atom Store** with the state as a **category**

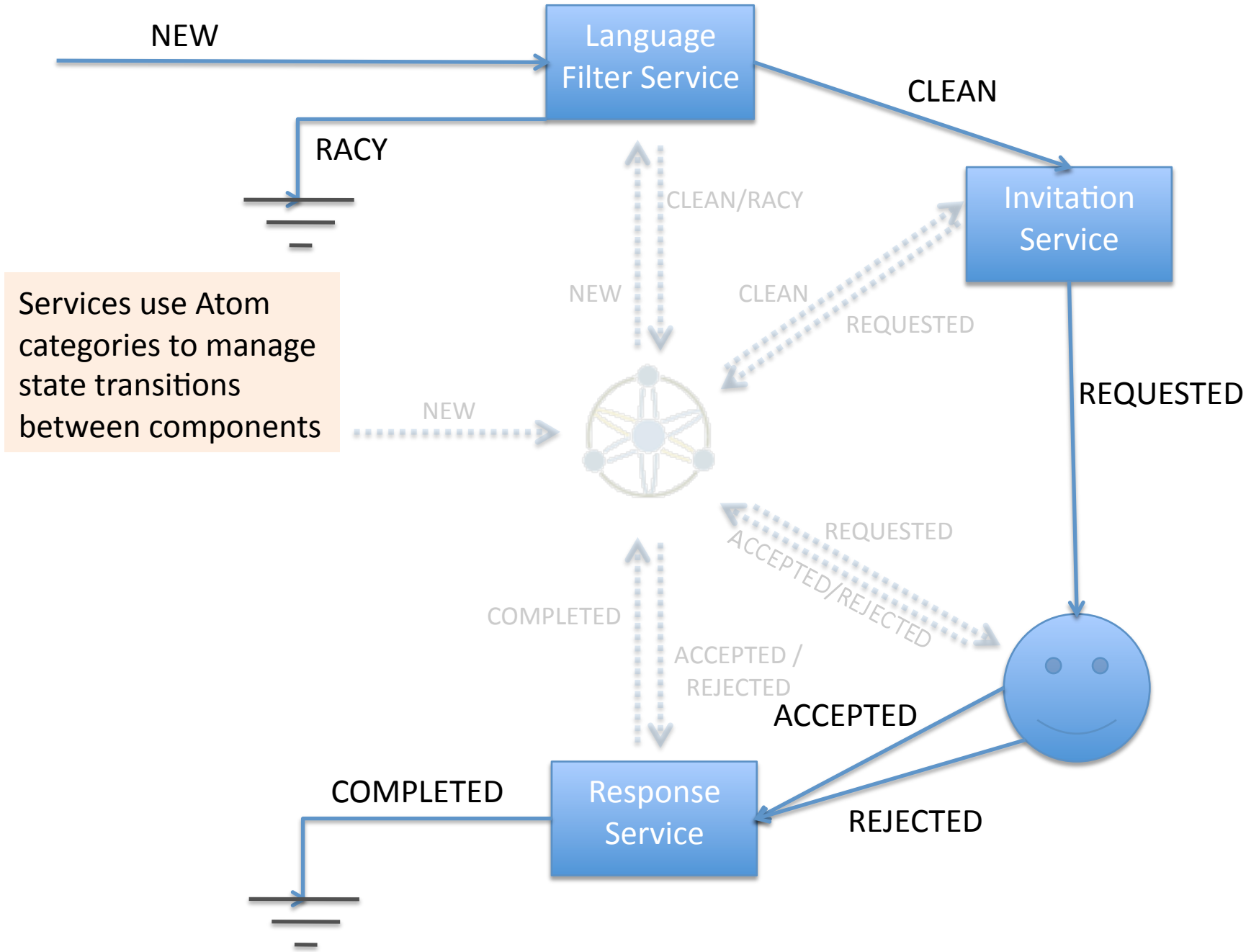
And each component polls at intervals to pull feeds of entries in that component's input state

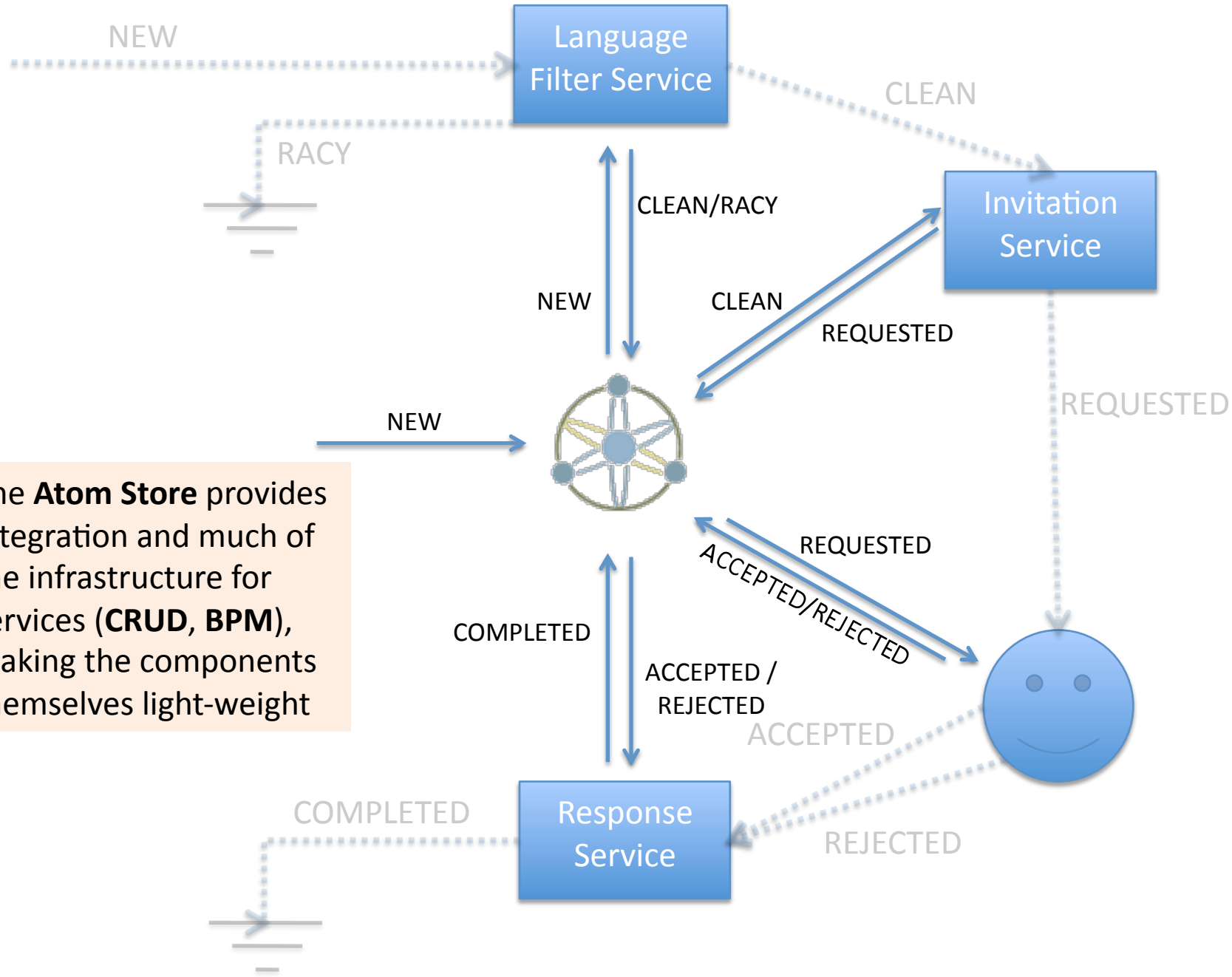


Entries **move** through the system via the re-categorization of entries on each write

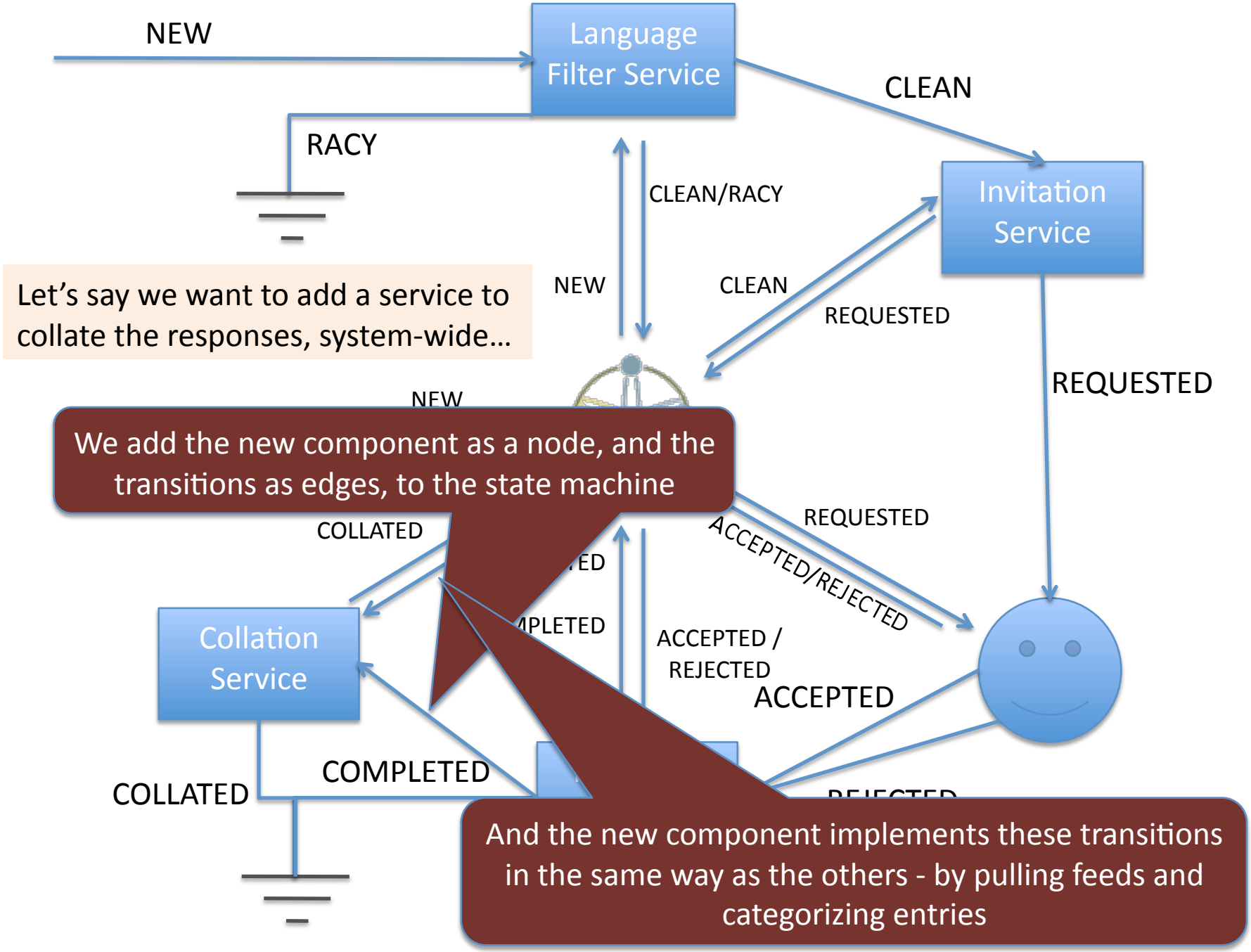
And finally reach one of the **terminal** states…

which remain in the Atom Store – applications outside of this workflow can just pull feeds of **COMPLETED** entries, knowing that they've been through the whole system


NEW
Language Filter Service
CLEAN
RACY
CLEAN/RACY
Invitation Service
NEW
CLEAN
REQUESTED
Services use Atom categories to manage state transitions between components
NEW
REQUESTED
REQUESTED
ACCEPTED/REJECTED
COMPLETED
ACCEPTED / REJECTED
ACCEPTED
COMPLETED
Response Service
REJECTED



The **Atom Store** provides integration and much of the infrastructure for services (**CRUD**, **BPM**), making the components themselves light-weight


NEW
Language Filter Service
CLEAN
RACY
CLEAN/RACY
Invitation Service
NEW
CLEAN
REQUESTED
REQUESTED
Let's say we want to add a service to collate the responses, system-wide…
NEW
We add the new component as a node, and the transitions as edges, to the state machine
COLLATED
REQUESTED
ACCEPTED/REJECTED
Collation Service
ACCEPTED / REJECTED
ACCEPTED
COMPLETED
COLLATED
And the new component implements these transitions in the same way as the others - by pulling feeds and categorizing entries

Atom-oriented Service Architecture.

- Similar to architectures built on queues (JMS, etc.) – but simple, RESTful, and already there
- Service components need very little infrastructure (Atom does most of the heavy lifting)
- Services become cookie-cutter
- We can get on to more important things, like the functionality of the service components, or…

Future planned work

- **ETags** – HTTP ETags are the preferred way to transfer parameters for Atom queries
  - We want to support both query parameters, for simplicity, and ETags going forward

Advanced AtomServer functionality

- **Aggregate Feeds** - the ability to "join" multiple related workspaces into a single feed of "aggregate entries"
  - Join entries on their **categories**
  - http://my-server/atom/$join(profiles,rendezvous)/urn:person

*AtomServer specific feature

Advanced AtomServer functionality

- **Batch Updates** – pack many **PUT**, **POST**, and **DELETE** operations into a single **feed** document, and send it all in one operation
  - Minimize round-trips to the server

*AtomServer specific feature

Advanced AtomServer functionality

- **Custom Content Storage** – like most parts of AtomServer, content storage is pluggable, so you can roll your own:
  - Unmarshal XML docs and store in a RDBMS for advanced searching
  - Store entry content in Amazon S3, or SVN to have a record of every change ever made

*AtomServer specific feature

Q & A

http://atomserver.org

http://infoq.com/articles/atomserver
http://infoq.com/articles/atomserver2

chris@atomserver.org
bryon@atomserver.org