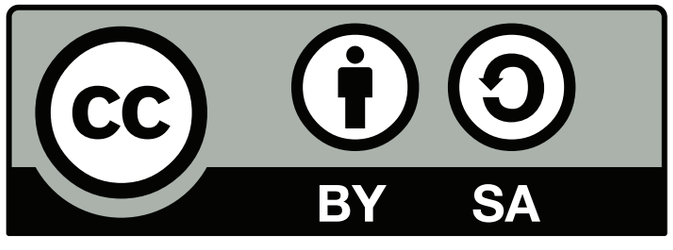
Tecnologia e Applicazioni Internet 2008/9

Lezione 4 - Rest

Matteo Vaccari
http://matteo.vaccari.name/
matteo.vaccari@uninsubria.it



What is REST?

Roy Fielding

*Re*presentational
*S*tate
*T*ransfer

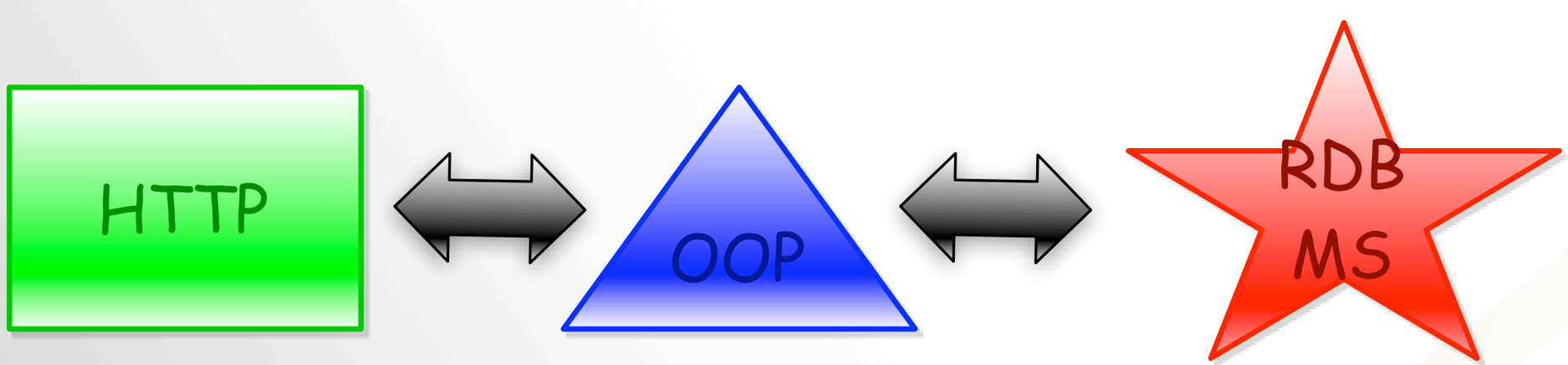
Roy T. Fielding "Architectural Styles and the Design of Network-based Software Architectures", Ph.D dissertation, 2000

REST is a *Software Architecture Style*

a set of *Constraints* on *Component Interaction* that, when obeyed, cause the resulting *Architecture* to have certain *properties*

Roy T. Fielding
http://roy.gbiv.com/untangled/2008/on-software-architecture

Web Applications Information Flow

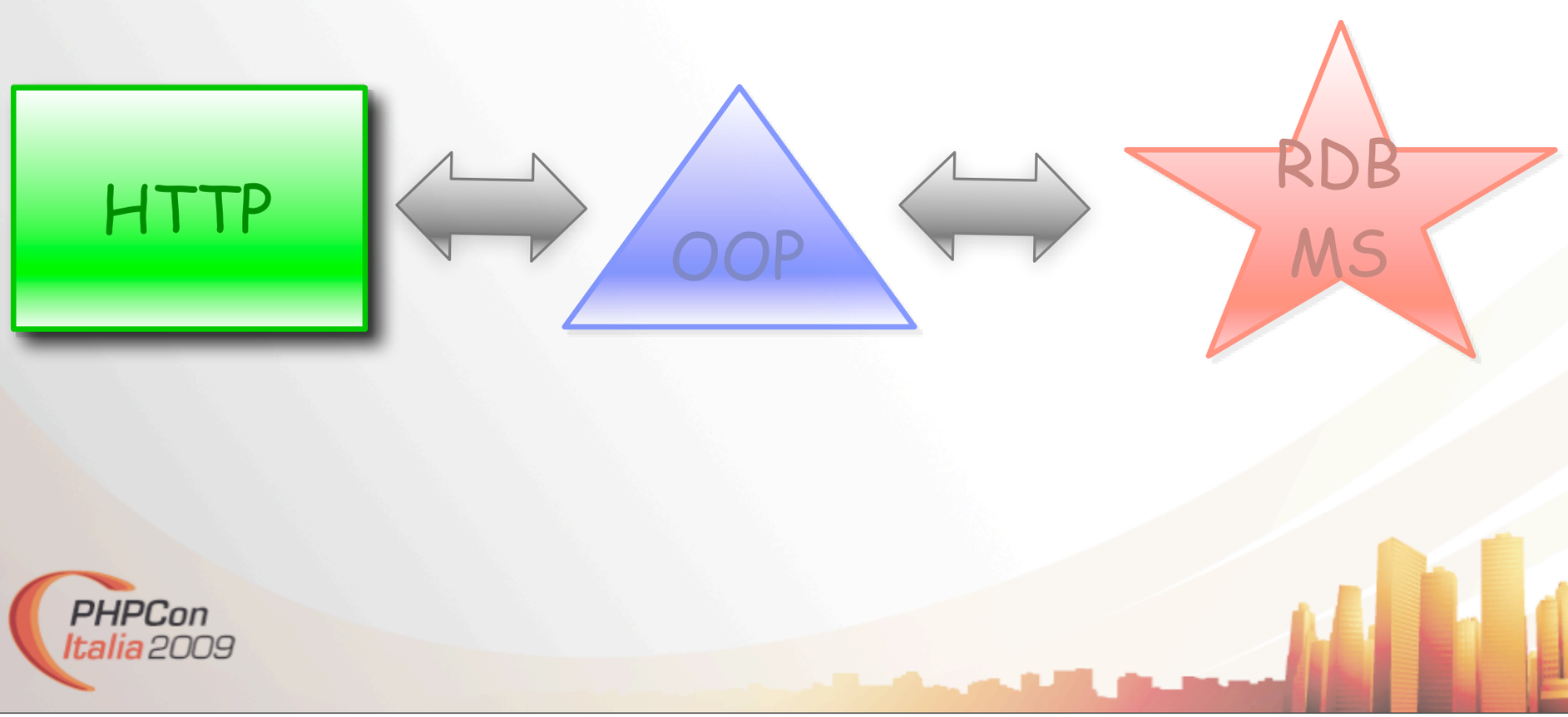


PHPCon Italia 2009

Gabriele Lana, *REST and Resource Oriented Architectures*, PHP Con 2009

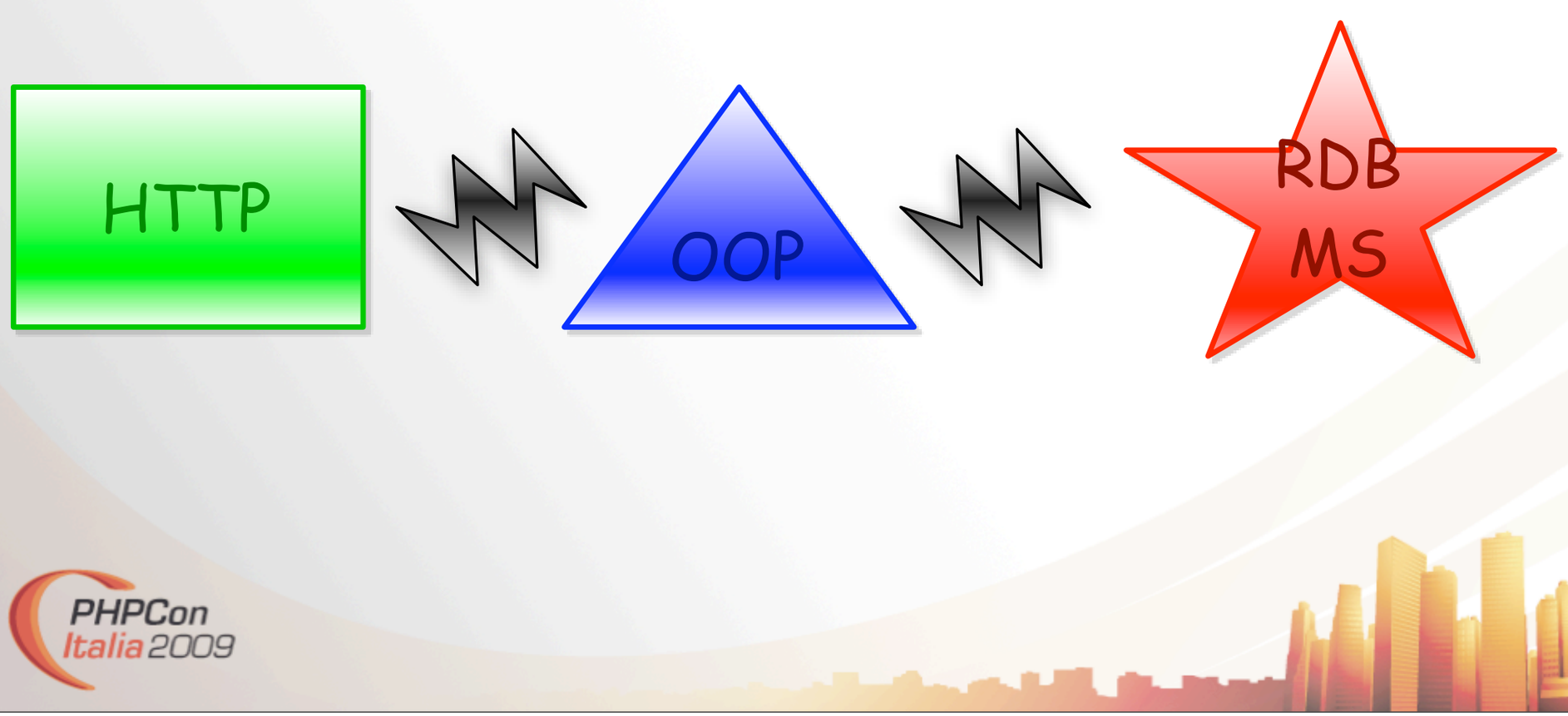
HTTP is REST

the most common *REST architecture*



Gabriele Lana, *REST and Resource Oriented Architectures*, PHP Con 2009

Impedance Mismatch

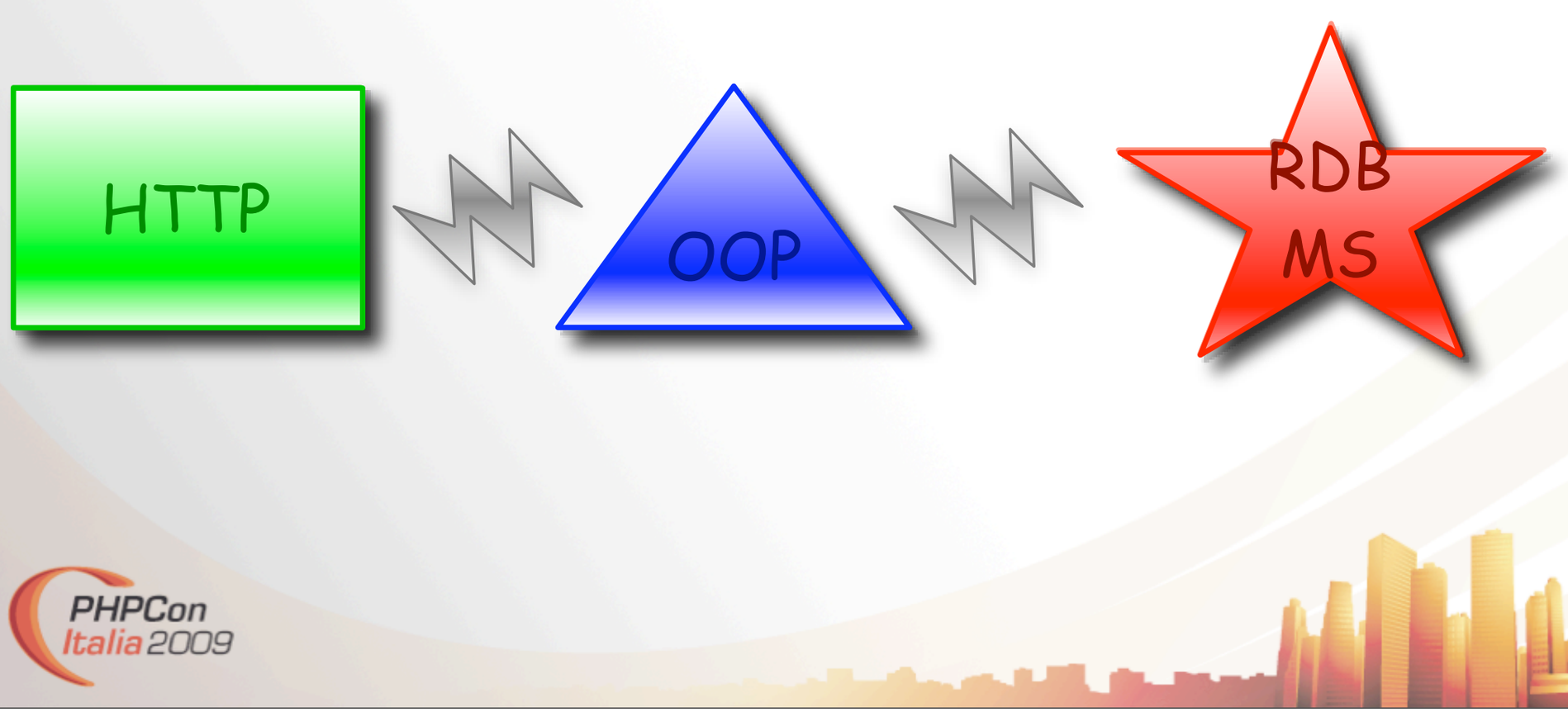


Gabriele Lana, *REST and Resource Oriented Architectures*, PHP Con 2009

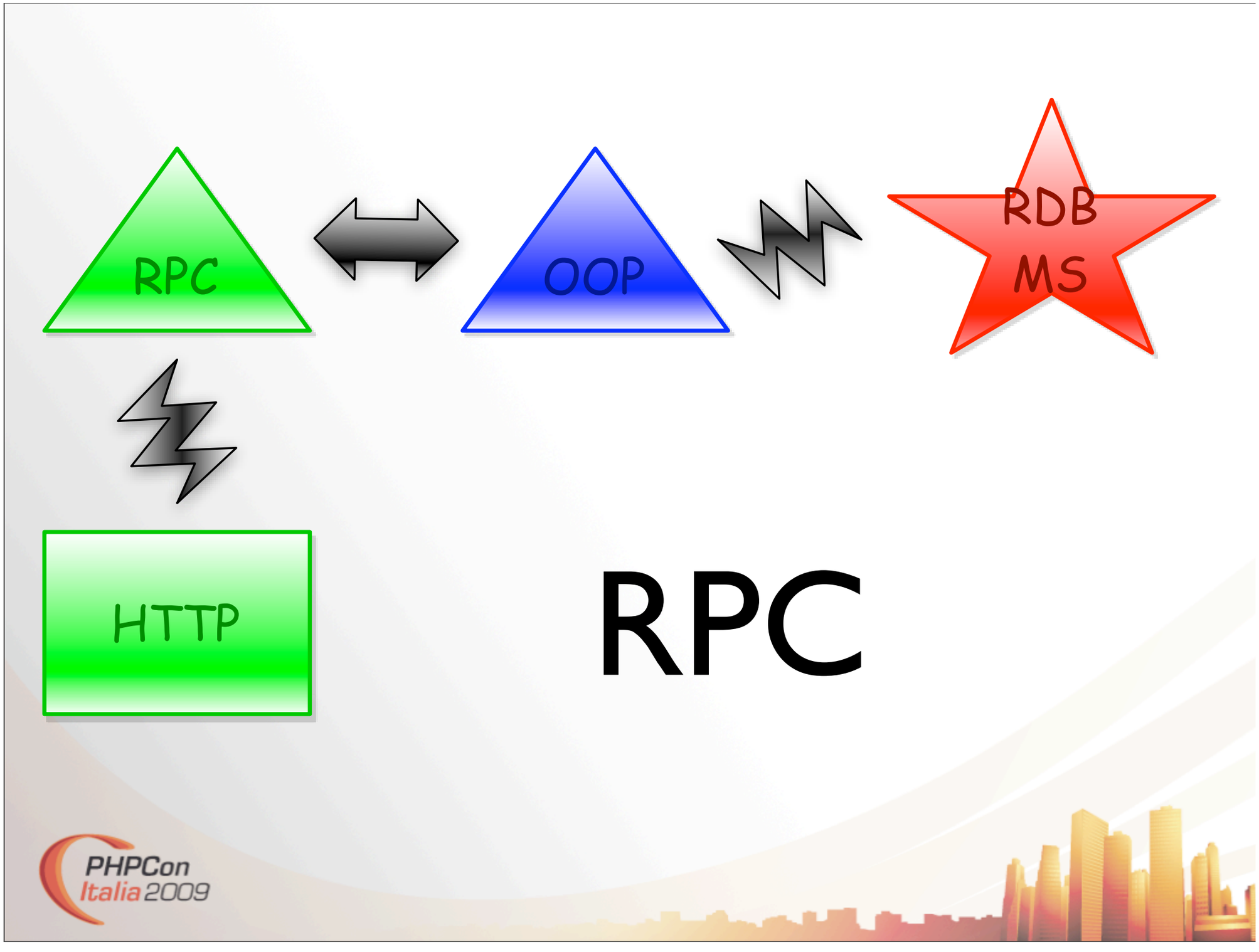
Accidental Complexity

Gabriele Lana, *REST and Resource Oriented Architectures*, PHP Con 2009

Essential Complexity

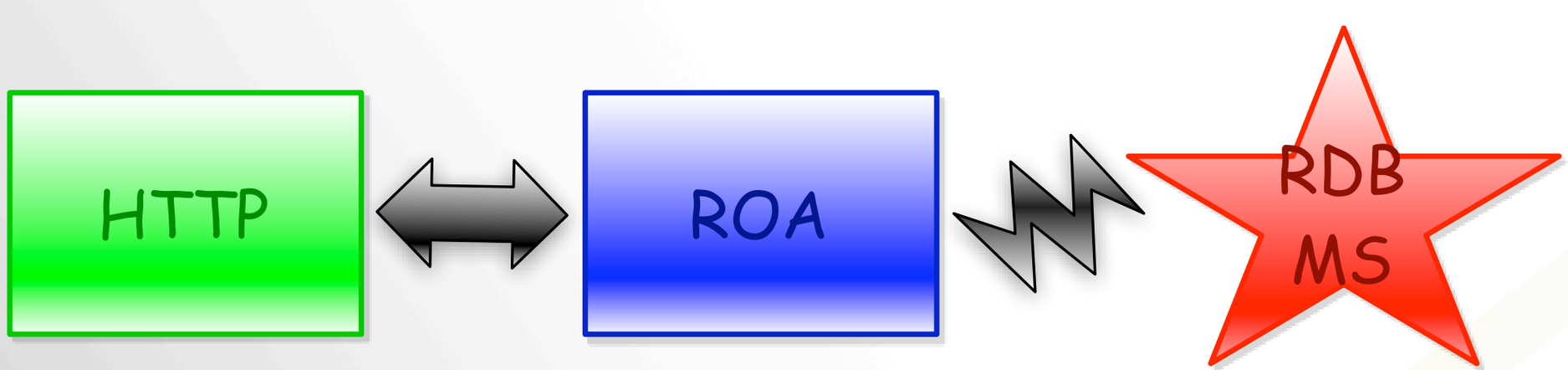


Gabriele Lana, *REST and Resource Oriented Architectures*, PHP Con 2009



Gabriele Lana, *REST and Resource Oriented Architectures*, PHP Con 2009

Resource Oriented Architectures



PHPCon Italia 2009

Gabriele Lana, *REST and Resource Oriented Architectures*, PHP Con 2009

Not Today But...
Yes, we Can

PHPCon Italia 2009

Gabriele Lana, *REST and Resource Oriented Architectures*, PHP Con 2009

Uniform

Ryan: We need to be able to talk to all machines about all the stuff that's on all the other machines. So we need some way of having one machine tell another machine about a resource that might be on yet another machine.

...

Wife: So how do the machines tell each other where things are?

Ryan: The URL, of course. If everything that machines need to talk about has a corresponding URL, you've created *the machine equivalent of a noun.*

http://tomayko.com/writings/rest-to-my-wife

What is REST?

(a) Give a name to every *Thing*

(b) Link *Things* together

(c) Use standard *methods*

(d) Communicate *statelessly*

(a) Give a name to every *Thing*

- A *resource* is any interesting concept in your domain
- Every *resource* should be identified by a *URI*
- Resources are the important parts of the application *domain*
- *URIs* are the *nouns*

Le più importanti invenzioni - IV

http://www.example.org/

URI
Uniform Resource Identifier

URI

Uniform Resource Identifier

Il *nome* di una *risorsa*

rfc2396

URI

Uniform Resource Identifiers (URI) provide a simple and extensible means for identifying a resource.

Tim Berners-Lee, Roy Fielding, rfc2396

Esempi di URI

ftp://ftp.is.co.za/rfc/rfc1808.txt

gopher://spinaltap.micro.umn.edu/00/Weather/California/Los%20Angeles

http://www.math.uio.no/faq/compression-faq/part1.html

mailto:mduerst@ifi.unizh.ch

news:comp.infosystems.www.servers.unix

telnet://melvyl.ucop.edu

What is a *resource*?

A resource can be *anything that has identity.*

Familiar examples include an electronic document, an image, a service (e.g., “today's weather report for Los Angeles”), and a collection of other resources.

Not all resources are network “retrievable”; e.g., human beings, corporations, and bound books in a library can also be considered resources.

Tim Berners-Lee, Roy Fielding, rfc2396

Altri esempi di *risorse*

La versione 22.1 del software rilasciato

L'ultima versione del software rilasciato

Il primo post del blog del 2 ottobre 2006

Il post dedicato a "RESTful Web Services"

Una mappa di Varese

Informazioni sui facoceri

Il primo numero primo > 1024

I primi 5 numeri primi > 1024

La lista dei bachi aperti

Richardson and Ruby, *RESTful Web Services*

Esempi di URI

http://www.example.com/software/releases/22.1.tgz
http://www.example.com/software/releases/latest.tgz
http://www.example.com/blog/2007/10/2/0
http://www.example.com/blog/restful-web-services
http://www.example.com/maps/IT/VA
http://www.example.com/search?q=facoceri
http://www.example.com/nextprime/1024
http://www.example.com/next-5-primes/1024
http://www.example.com/bugs/by-state/open

Richardson and Ruby, *RESTful Web Services*

E che ci facciamo con le *risorse*?

Having identified a resource, a system may perform a variety of operations on the resource, as might be characterized by such words as 'access', 'update', 'replace', or 'find attributes'

Tim Berners-Lee, Roy Fielding, rfc2396

URI e URL

URI: Universal Resource *Identifier*

URL: Univeral Resource *Locator*

URL refers to the *subset of URI* that identify resources via a representation of their primary access mechanism (e.g., their network "location")

Tim Berners-Lee, Roy Fielding, rfc2396

Addressability

Un'applicazione Web è *indirizzabile* se espone gli aspetti interessanti dei suoi dati come *risorse*

Google Mail è indirizzabile?

https://mail.google.com/mail/#inbox/11dbe2460af15fe6

https://mail.google.com/mail/#label/aaa-agire

https://mail.google.com/mail/#search/marco

Wordpress è indirizzabile?

http://matteo.vaccari.name/blog/archives/138

http://matteo.vaccari.name/blog/archives/date/2008/10

http://matteo.vaccari.name/blog/archives/category/agile

(b) Link *Things* together

- Humans use the web by starting with a URL, and then following links
- Machines should do the same

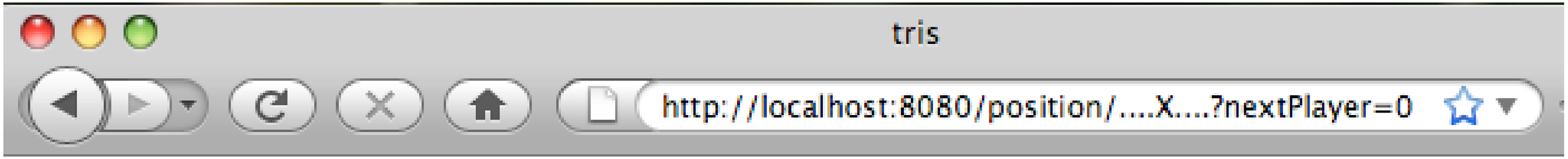
Welcome to TRIS!

- Start play as "0"
- Start play as "X"

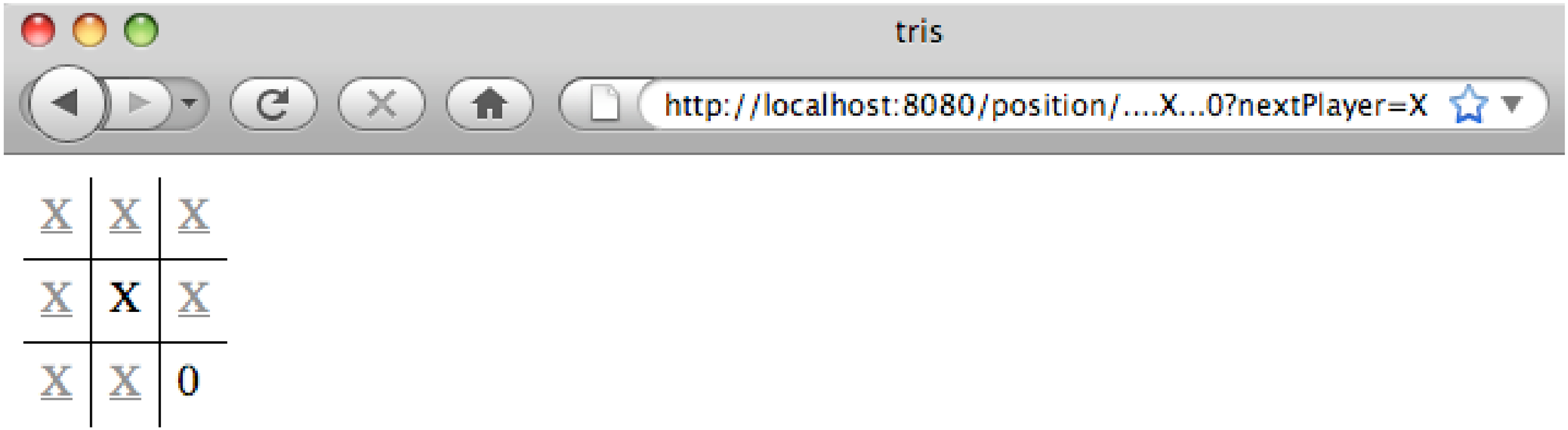
```
<ul>
  <li><a href="/position/.........?nextPlayer=0">Start play as "0"</a></li>
  <li><a href="/position/.........?nextPlayer=X">Start play as "X"</a></li>
</ul>
```

X	X	X
X	X	X
X	X	X

```
<table cellspacing="0" cellpadding="0">
  <tr>
    <td class="right bottom"><a href='X........?nextPlayer=0'>X</a></td>
    <td class ="right bottom"><a href='.X.......?nextPlayer=0'>X</a></td>
    <td class ="bottom"><a href='..X......?nextPlayer=0'>X</a></td>
  </tr>
  <tr>
    <td class="right bottom"><a href='...X.....?nextPlayer=0'>X</a></td>
    <td class ="right bottom"><a href='....X....?nextPlayer=0'>X</a></td>
    <td class ="bottom"><a href='.....X...?nextPlayer=0'>X</a></td>
  </tr>
  <tr>
    <td class="right"><a href='......X..?nextPlayer=0'>X</a></td>
    <td class ="right"><a href='.......X.?nextPlayer=0'>X</a></td>
    <td ><a href='........X?nextPlayer=0'>X</a></td>
  </tr>
</table>
```



0	0	0
0	X	0
0	0	0


tris
http://localhost:8080/position/....X...0?nextPlayer=X
X X X
X X X
X X 0

(c) Use standard *methods*

- GET
- POST
- PUT
- DELETE

```
GET /some/resource HTTP/1.1
```

```
HTTP/1.1 200 OK
```

- Dammi una *rappresentazione* di questa risorsa
- Safe & Idempotent

```
PUT /some/resource HTTP/1.1
\n
some-representation-of-my-resource
```

```
HTTP/1.1 201 Created
```

- Crea una *nuova* risorsa alla URI data
- Not Safe; Idempotent

```
PUT /some/resource HTTP/1.1
\n
some-representation-of-my-resource
```

```
HTTP/1.1 205 Reset Content
```

- Modifica una risorsa presistente alla URI data
- Not Safe; Idempotent

```
DELETE /some/resource HTTP/1.1
```

```
HTTP/1.1 204 No Content
```

- Cancella una risorsa
- Not Safe; Idempotent

```
POST /articles HTTP/1.1
\n
some-representation-of-my-article
```

```
HTTP/1.1 201 Created
Location: /articles/12345
```

- Crea una nuova risorsa *sotto* la URI data
- Not Safe; Not Idempotent
- La URI della risorsa creata viene *decisa dal server*

Create Read Update Delete

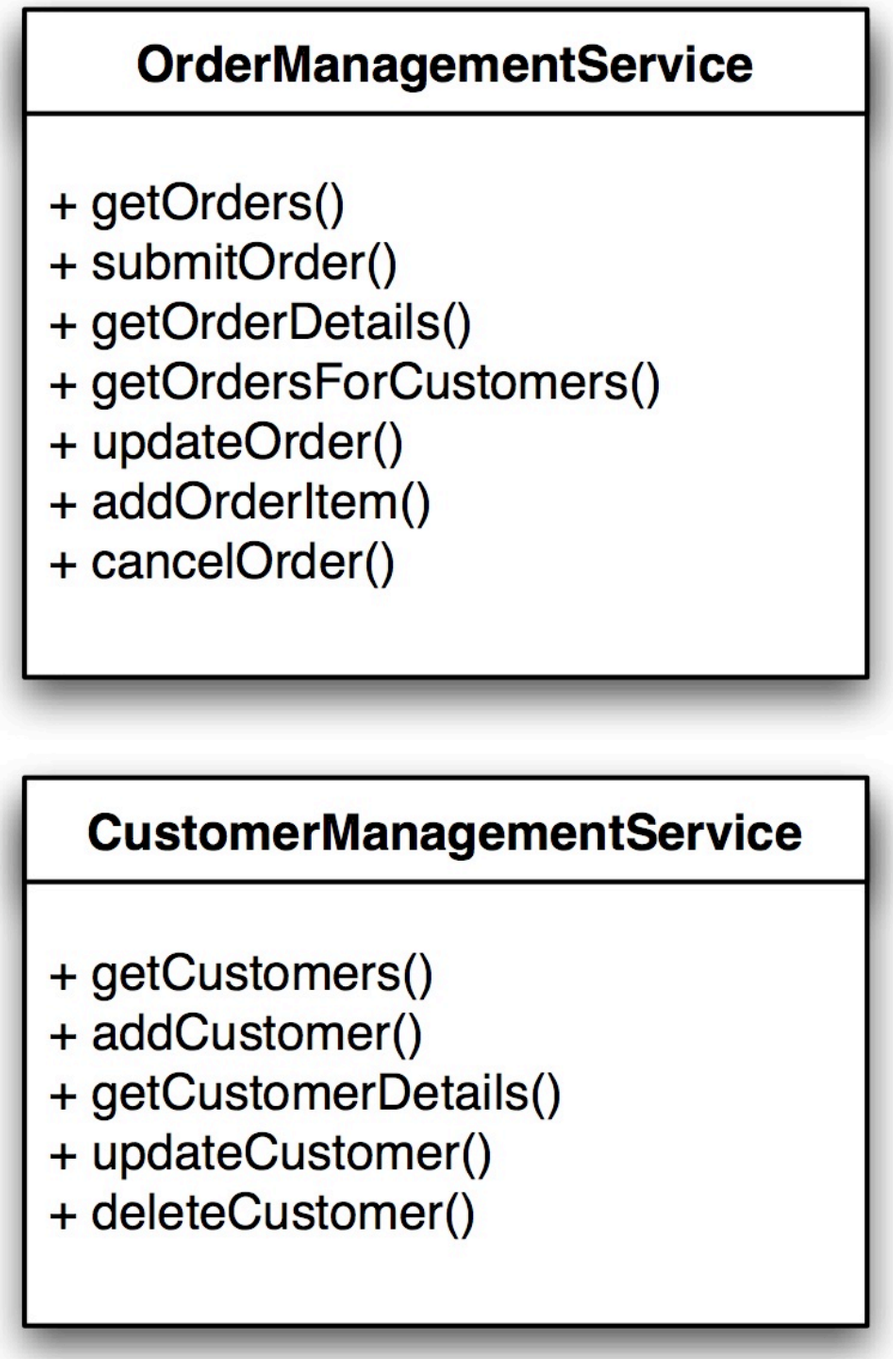
find	create	update	destroy
SELECT	INSERT	UPDATE	DELETE

GET	POST	PUT	DELETE
find	create	update	destroy
SELECT	INSERT	UPDATE	DELETE

```
POST /people/create
GET  /people/show/1
POST /people/update/1
POST /people/destroy/1
```

```
POST   /people
GET    /people/1
PUT    /people/1
DELETE /people/1
```

RPC style



Steven Tilkov, http://www.infoq.com/articles/rest-introduction

OrderManagementService

+ getOrders()
+ submitOrder()
+ getOrderDetails()
+ getOrdersForCustomers()
+ updateOrder()
+ addOrderItem()
+ cancelOrder()

CustomerManagementService

+ getCustomers()
+ addCustomer()
+ getCustomerDetails()
+ updateCustomer()
+ deleteCustomer()

«interface»
Resource

GET
PUT
POST
DELETE

/orders

GET - list all orders
PUT - unused
POST - add a new order
DELETE - unused

/orders/{id}

GET - get order details
PUT - update order
POST - add item
DELETE - cancel order

/customers

GET - list all customers
PUT - unused
POST - add new customer
DELETE - unused

/customers/{id}

GET - get customer details
PUT - update customer
POST - unused
DELETE - delete customer

/customers/{id}/orders

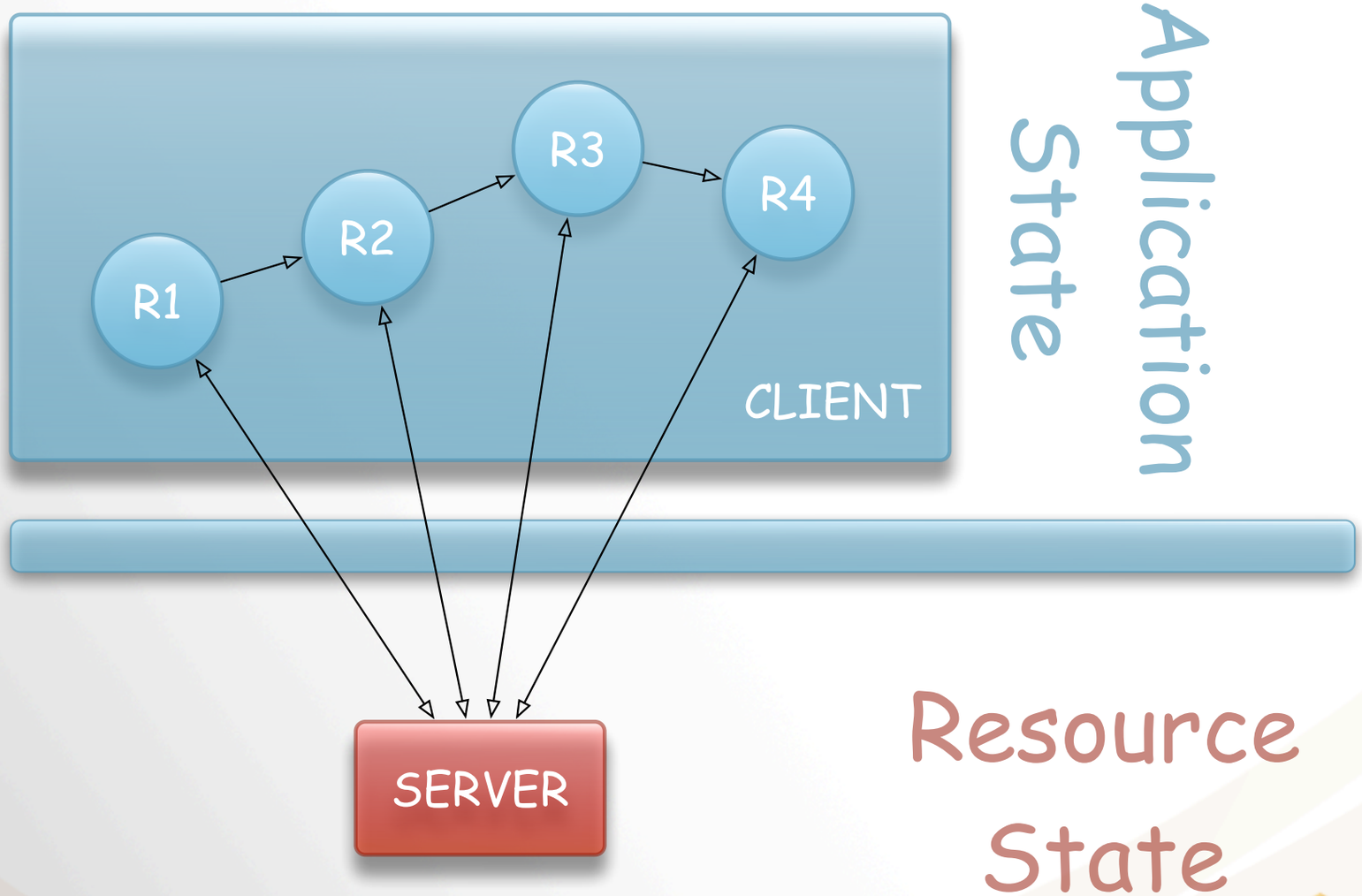
GET - get all orders for customer
PUT - unused
POST - add order
DELETE - cancel all customer orders

Steven Tilkov, http://www.infoq.com/articles/rest-introduction

(d) Communicate *Statelessly*

- Two kinds of *state*:
  - Resource state (on the server)
  - Application state (on the client)

*Re*presentational *S*tate *T*ransfer



PHPCon Italia 2009

Gabriele Lana, *REST and Resource Oriented Architectures*, PHP Con 2009

REST design quiz

Un *ordine* può essere in stato *aperto* oppure *chiuso*

(a) PUT a new state to the resource

(b) Move (logically) the resource from the *open orders* collection to the *closed orders* one

(c) Create a new *OrderClosure* resource

REST design quiz (ii)

Se un cliente POSTa un *ordine*, e non riceve una risposta, non sa se l'ordine è stato ricevuto oppure no. Se lo POSTa di nuovo, rischia di mandare due ordini uguali (infatti POST non è idempotente).

(a) Esponi la risorsa "l'ultimo ordine che ti ho mandato"

(b) Crea gli ordini con PUT (ma a quale URI?)

(c) Includi nella POST un ID unico (ma come lo ottengo?)

So, what *is* REST?

REST = The Web as it *should* work

"... the motivation for developing REST was to create an architectural model for *how the Web should work*, such that it could serve as the *guiding framework* for the Web protocol standards"

http://www.ics.uci.edu/~fielding/pubs/dissertation/evaluation.htm

REST = the Web for *humans* and the Web for *robots* should work the same way

REST = Design your *domain model* around CRUD

"Before refactoring, IconBuffet had 10 controllers and 76 actions:... Now, without adding or removing any features, IconBuffet has 13 controllers and 58 actions.

There are seven standard Rails actions: index, new, create, show, edit, update, and destroy. Everything else—oddball actions—are usually a clue that you're doing RPC."

http://scottraymond.net/2006/7/20/refactoring-to-rest/

REST = Make your *domain model* a part of the Web

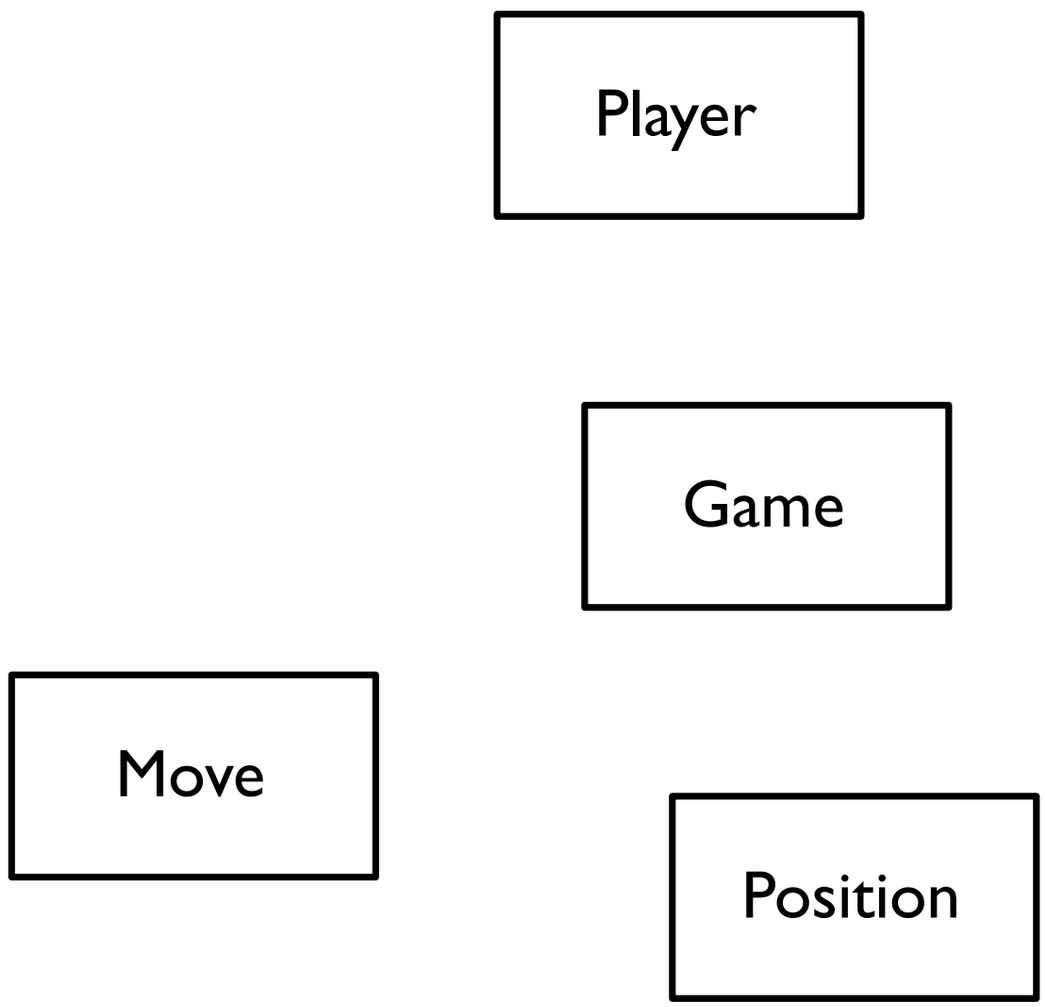
Extend *Tris* RESTfully

How to design a REST service?

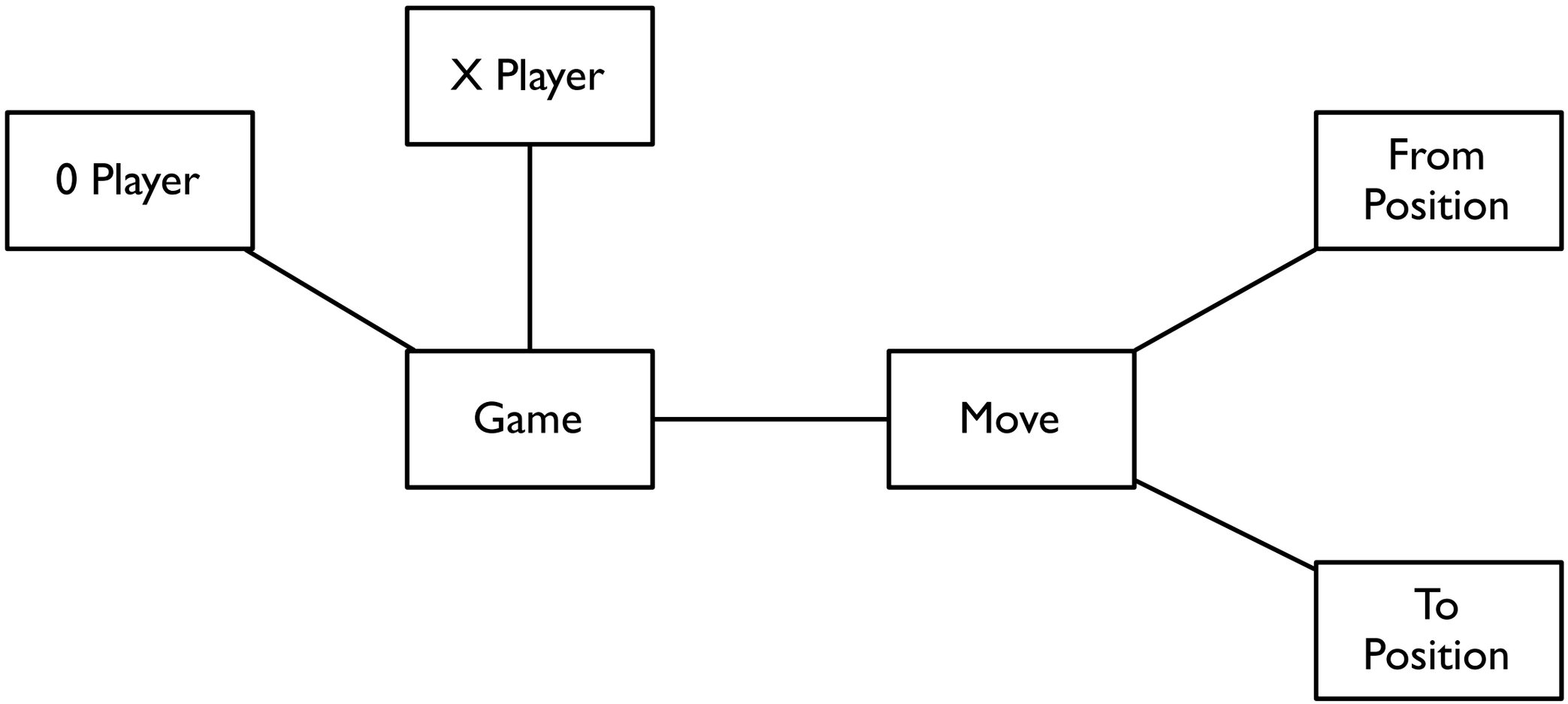
1. Identify *Resources* in your *Domain* (Tris)
2. Design the URIs
3. For each *Resource* expose a subset of the Uniform Interface (GET, PUT, POST, DELETE)
4. For each *Resource* design the *Representations*
5. Design typical *flows* and *error conditions*

Richardson & Ruby, *RESTful Web Services*, O'Reilly

1. Identify *resources* in your *domain*

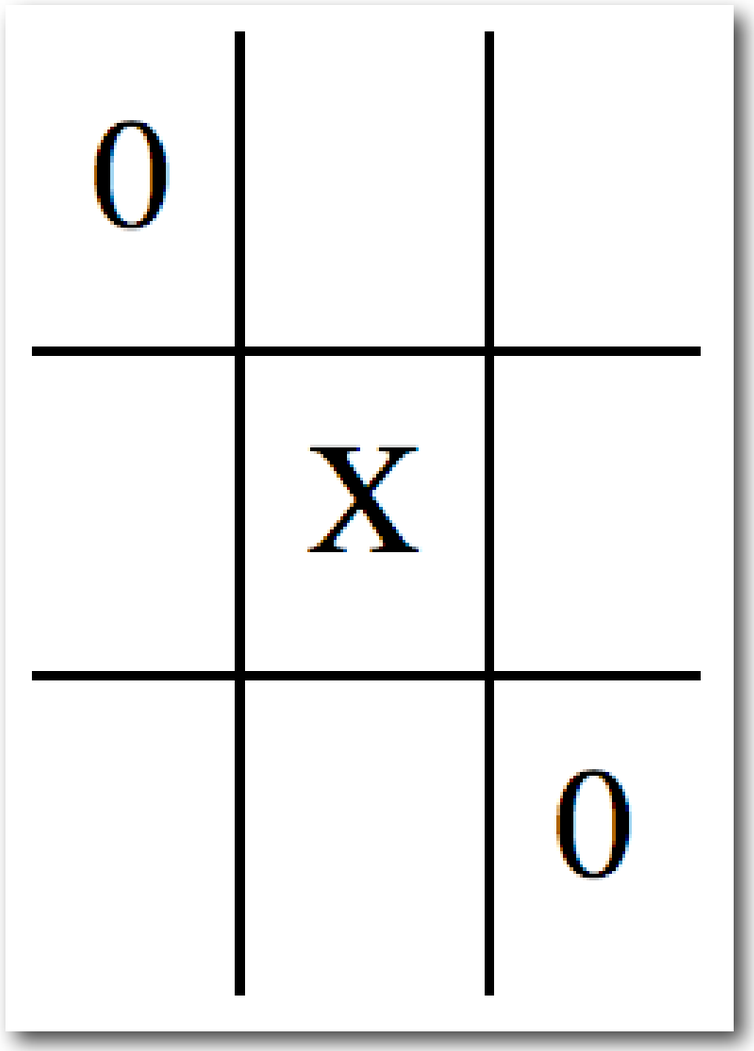


1. Identify *resources* in your *domain*



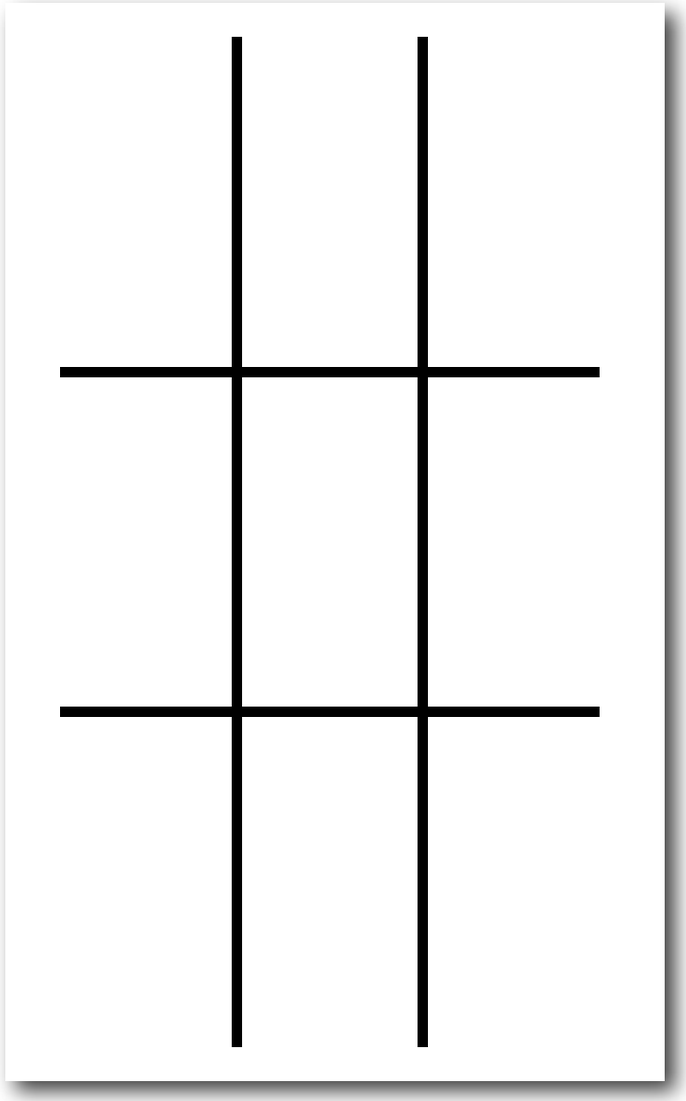
2. Design the URIs

/position/0...X...0



2. Design the URIs

/position/initial



2. Design the URIs

/player/1737514

3. Expose the *Uniform Interface*

```
POST /position/....0.... HTTP/1.1
\n
move=X8
```

```
HTTP/1.1 302 Found
Location: /position/....0...X
```

POST to let resources *process information*

3. Expose the *Uniform Interface*

```
POST /game HTTP/1.1
\n
X=/player/10304
0=/player/99383
```

```
HTTP/1.1 201 Created
Location: /game/42
```

POST to create new resources