innoQ

REST ≥ CRUD

Stefan Tilkov | innoQ | stefan.tilkov@innoq.com
http://railsconsulting.de

Stefan Tilkov

innoQ

http://www.innoQ.com

Stefan Tilkov

innoQ http://www.innoQ.com

stefan.tilkov@innoq.com

Stefan Tilkov

innoQ http://www.innoQ.com

stefan.tilkov@innoq.com

http://www.innoq.com/blog/st/

Stefan Tilkov

innoQ http://www.innoQ.com

stefan.tilkov@innoq.com

http://www.innoq.com/blog/st/

http://www.InfoQ.com

Stefan Tilkov

innoQ http://www.innoQ.com

stefan.tilkov@innoq.com

http://www.innoq.com/blog/st/

InfoQ queue
174,255 Sep unique visitors

http://www.InfoQ.com

http://www.soa-expertenwissen.de

Stefan Tilkov

innoQ http://www.innoQ.com

stefan.tilkov@innoq.com

http://www.innoq.com/blog/st/

InfoQ queue
174,255 Sep unique visitors

http://www.InfoQ.com

http://www.soa-expertenwissen.de

http://www.railsconsulting.de

Audience Poll

Audience Poll

How many of you make money doing Rails?

Audience Poll

How many of you make money doing Rails?

Percentage of Rails users developing RESTfully?

Audience Poll

How many of you make money doing Rails?

Percentage of Rails users developing RESTfully?

How many are just learning Ruby/Rails?

Audience Poll

How many of you make money doing Rails?

Percentage of Rails users developing RESTfully?

How many are just learning Ruby/Rails?

How many want to learn what REST is about?

Audience Poll

How many of you make money doing Rails?

Percentage of Rails users developing RESTfully?

How many are just learning Ruby/Rails?

How many want to learn what REST is about?

How many know REST and want to see where I'm wrong?

What is REST?

3 definitions

1

REST: An Architectural Style

One of a number of “architectural styles”

... described by Roy Fielding in his dissertation

... defined via a set of *constraints* that have to be met

... architectural principles underlying HTTP, defined *a posteriori*

... with the Web as one particular instance

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

2

REST: The Web Used Correctly

A system or application architecture

... that uses HTTP, URI and other Web standards “correctly”

... is “on” the Web, not tunneled through it

... also called “WOA”, “ROA”, “RESTful HTTP”

3

REST: XML without SOAP

Send plain XML (w/o a SOAP Envelope) via HTTP

... violating the Web as much as WS-*

... preferably use GET to invoke methods

... or tunnel everything through POST

... commonly called “POX”

Only option 1 is the right one
(because Roy said so)

But we'll go with option 2 (and equate "REST" with "RESTful HTTP usage")

and avoid option 3 like the plague

REST Explained in 5 Easy Steps

1. Give Every “Thing” an ID

```
http://example.com/customers/1234

http://example.com/orders/2007/10/776654

http://example.com/products/4554

http://example.com/processes/sal-increase-234
```

2. Link Things To Each Other

```
<order self='http://example.com/orders/1234'>
  <amount>23</amount>
  <product ref='http://example.com/products/4554' />
  <customer ref='http://example.com/customers/1234' />
</order>
```

3. Use Standard Methods

GET	retrieve information, possibly cached
PUT	Update or create with known ID
POST	Create or append sub-resource
DELETE	(Logically) remove

4. Allow for Multiple “Representations”

```
GET /customers/1234
Host: example.com
Accept: application/vnd.mycompany.customer+xml
```

4. Allow for Multiple “Representations”

```
GET /customers/1234
Host: example.com
Accept: application/vnd.mycompany.customer+xml

<customer>...</customer>
```

4. Allow for Multiple “Representations”

```
GET /customers/1234
Host: example.com
Accept: application/vnd.mycompany.customer+xml
```

```
<customer>...</customer>
```

```
GET /customers/1234
Host: example.com
Accept: text/x-vcard
```

4. Allow for Multiple “Representations”

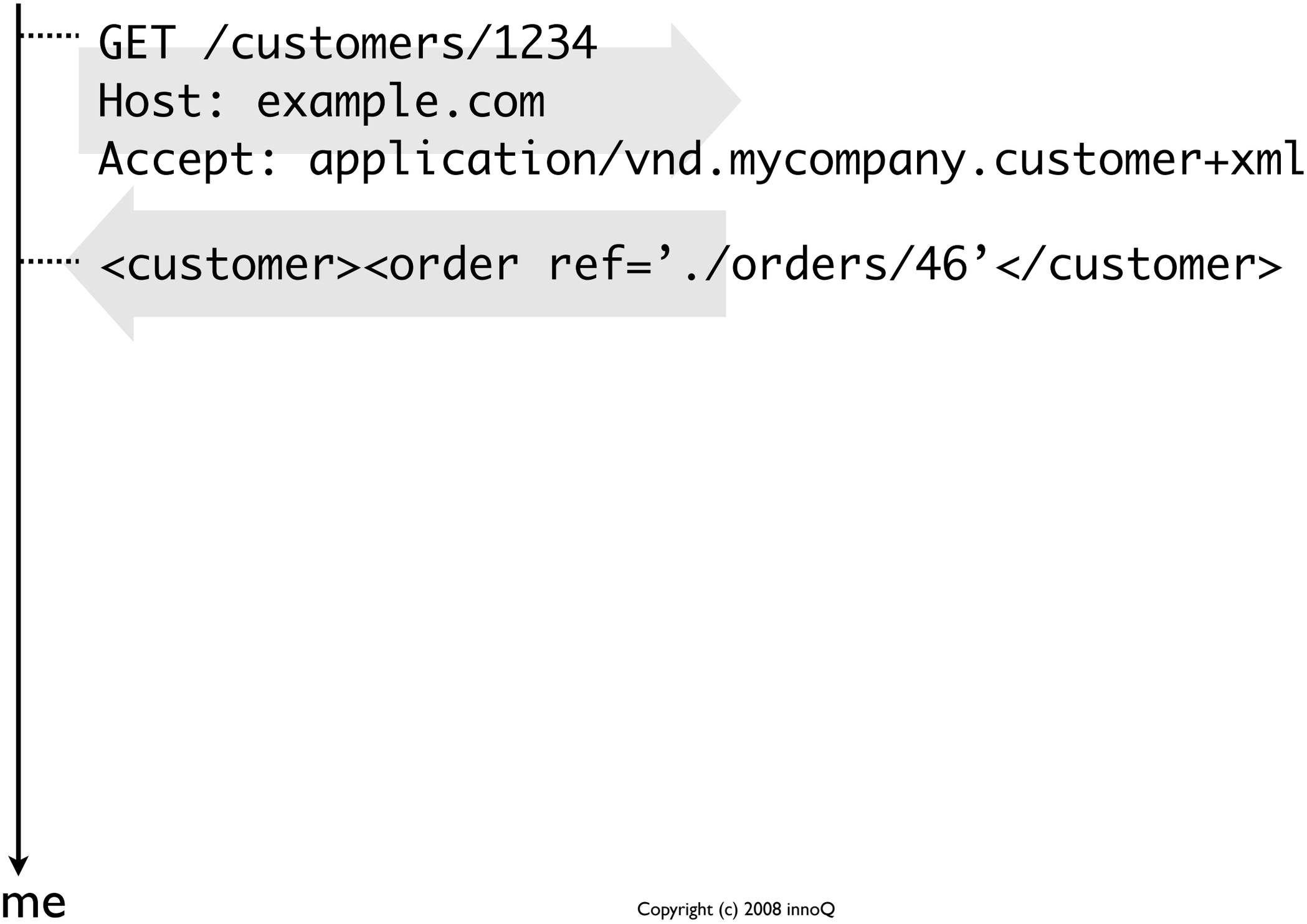
```
GET /customers/1234
Host: example.com
Accept: application/vnd.mycompany.customer+xml

<customer>...</customer>
```

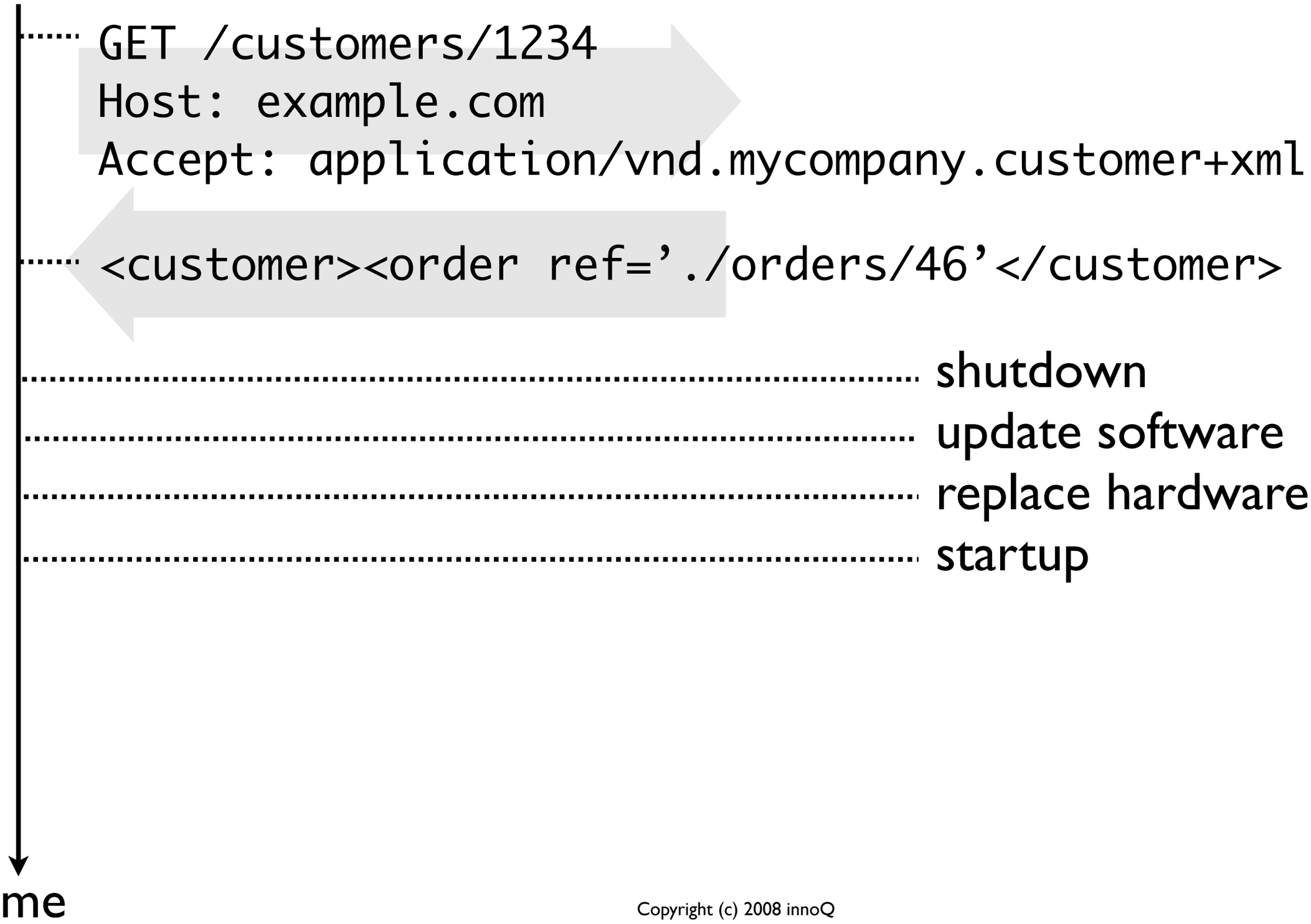
```
GET /customers/1234
Host: example.com
Accept: text/x-vcard

begin:vcard
...
end:vcard
```

5. Communicate Statelessly



5. Communicate Statelessly



5. Communicate Statelessly

```
GET /customers/1234
Host: example.com
Accept: application/vnd.mycompany.customer+xml
```

```
<customer><order ref='./orders/46'</customer>
```

shutdown

update software

replace hardware

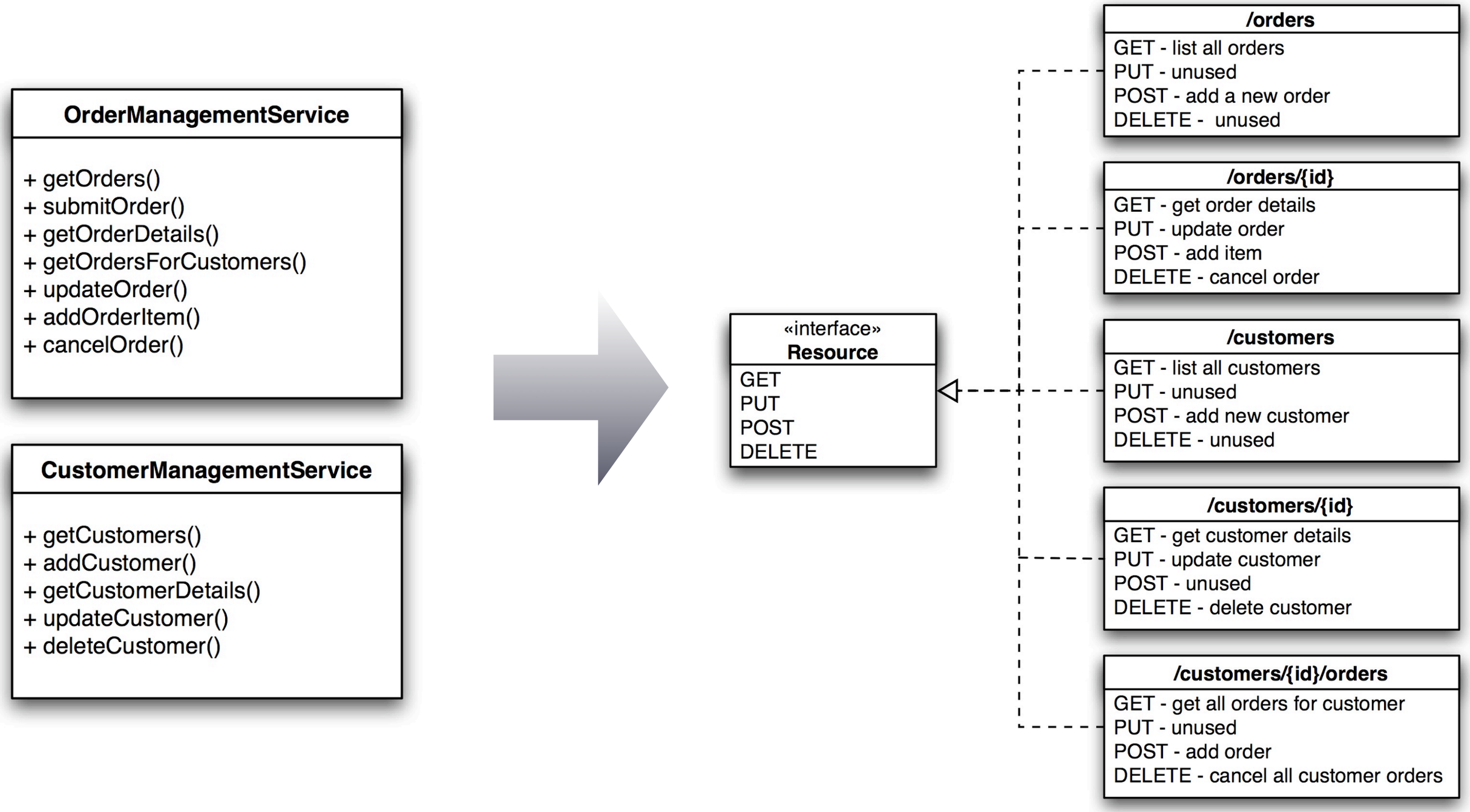
startup

```
GET /customers/1234/orders/46
Host: example.com
Accept: application/vnd.mycompany.order+xml
```

```
<order>...</order>
```

time

Consequences



What's cool about REST?

A very rough analogy (in pseudocode)

```
interface Resource {
      Resource(URI u)
      Response get()
      Response post(Request r)
      Response put(Request r)
      Response delete()
}
```

```
interface Resource {
      Resource(URI u)
      Response get()
      Response post(Request r)
      Response put(Request r)
      Response delete()
}
```

```
interface Resource {
      Resource(URI u)
      Response get()
      Response post(Request r)
      Response put(Request r)
      Response delete()
}
```

```
class CustomerCollection : Resource {
      ...
      Response post(Request r) {
            id = createCustomer(r)
            return new Response(201, r)
      }
      ...
}
```

```
interface Resource {
      Resource(URI u)
      Response get()
      Response post(Request r)
      Response put(Request r)
      Response delete()
}
```

Any HTTP client
(Firefox, IE, curl, wget)

```
class CustomerCollection : Resource {
      ...
      Response post(Request r) {
            id = createCustomer(r)
            return new Response(201, r)
      }
      ...
}
```

```
interface Resource {
      Resource(URI u)
      Response get()
      Response post(Request r)
      Response put(Request r)
      Response delete()
}
```

Any HTTP client
(Firefox, IE, curl, wget)

Any HTTP server

```
class CustomerCollection : Resource {
      ...
      Response post(Request r) {
            id = createCustomer(r)
            return new Response(201, r)
      }
      ...
}
```

```
interface Resource {
      Resource(URI u)
      Response get()
      Response post(Request r)
      Response put(Request r)
      Response delete()
}
```

Any HTTP client
(Firefox, IE, curl, wget)

Any HTTP server

Caches

```
class CustomerCollection : Resource {
      ...
      Response post(Request r) {
            id = createCustomer(r)
            return new Response(201, r)
      }
      ...
}
```

```
interface Resource {
      Resource(URI u)
      Response get()
      Response post(Request r)
      Response put(Request r)
      Response delete()
}
```

Any HTTP client
(Firefox, IE, curl, wget)

Any HTTP server

Caches

Proxies

```
class CustomerCollection : Resource {
      ...
      Response post(Request r) {
            id = createCustomer(r)
            return new Response(201, r)
      }
      ...
}
```

```
interface Resource {
      Resource(URI u)
      Response get()
      Response post(Request r)
      Response put(Request r)
      Response delete()
}
```

Any HTTP client
(Firefox, IE, curl, wget)

Any HTTP server

Caches

Proxies

Google, Yahoo!, MSN

```
class CustomerCollection : Resource {
      ...
      Response post(Request r) {
            id = createCustomer(r)
            return new Response(201, r)
      }
      ...
}
```

```
interface Resource {
      Resource(URI u)
      Response get()
      Response post(Request r)
      Response put(Request r)
      Response delete()
}
```

Any HTTP client
(Firefox, IE, curl, wget)

Any HTTP server

Caches

Proxies

Google, Yahoo!, MSN

```
class CustomerCollection : Resource {
      ...
      Response post(Request r) {
            id = createCustomer(r)
            return new Response(201, r)
      }
      ...
}
```

Anything that knows
your app

```
interface Resource {
      Resource(URI u)
      Response get()
      Response post(Request r)
      Response put(Request r)
      Response delete()
}
```

```
class CustomerCollection : Resource {
      ...
      Response post(Request r) {
            id = createCustomer(r)
            return new Response(201, r)
      }
      ...
}
```

Any HTTP client
(Firefox, IE, curl, wget)

Any HTTP server

Caches

Proxies

Google, Yahoo!, MSN

Anything that knows
your app

```
interface Resource {
      Resource(URI u)
      Response get()
      Response post(Request r)
      Response put(Request r)
      Response delete()
}
```

```
class CustomerCollection : Resource {
      ...
      Response post(Request r) {
            id = createCustomer(r)
            return new Response(201, r)
      }
      ...
}
```

generic

Any HTTP client
(Firefox, IE, curl, wget)

Any HTTP server

Caches

Proxies

Google, Yahoo!, MSN

Anything that knows
your app

```
interface Resource {
      Resource(URI u)
      Response get()
      Response post(Request r)
      Response put(Request r)
      Response delete()
}
```

generic

Any HTTP client
(Firefox, IE, curl, wget)

Any HTTP server

Caches

Proxies

Google, Yahoo!, MSN

Anything that knows
your app

```
class CustomerCollection : Resource {
      ...
      Response post(Request r) {
            id = createCustomer(r)
            return new Response(201, r)
      }
      ...
}
```

specific

generic

Anything that understands HTTP

```
interface Resource {
      ...
}
```

```
class CustomerCollection : AtomFeed {
      ...
}
```

Anything that knows your app

specific

generic

Anything that understands HTTP

```
interface Resource {
      ...
}
```

```
class AtomFeed : Resource {
      AtomFeed get()
      post(Entry e)
      ...
}
```

```
class CustomerCollection : AtomFeed {
      ...
}
```

Anything that knows your app

specific

generic

Anything that understands HTTP

```
interface Resource {
      ...
}
```

```
class AtomFeed : Resource {
      AtomFeed get()
      post(Entry e)
      ...
}
```

Any feed reader

```
class CustomerCollection : AtomFeed {
      ...
}
```

Anything that knows your app

specific

generic

```
interface Resource {
      ...
}
```

Anything that understands HTTP

```
class AtomFeed : Resource {
      AtomFeed get()
      post(Entry e)
      ...
}
```

Any feed reader

Any AtomPub client

```
class CustomerCollection : AtomFeed {
      ...
}
```

Anything that knows your app

specific

generic

```
interface Resource {
      ...
}
```

Anything that understands HTTP

```
class AtomFeed : Resource {
      AtomFeed get()
      post(Entry e)
      ...
}
```

Any feed reader

Any AtomPub client

Yahoo! Pipes

```
class CustomerCollection : AtomFeed {
      ...
}
```

Anything that knows your app

specific

REST & Rails

Rails < 2.0

```
ActionController::Routing::Routes.draw do |map|
  # ...
  map.connect ':controller/:action'
end
```

http://localhost:3000/demo/read_something?value1=...&value2=...

```
class DemoController < ApplicationController

  def read_something
    # retrieve some result using params[:value1], params[:value2], ...
  end

  def change_something
    # update backend using params[:value1], params[:value2], ...
  end
end
```

Rails < 2.0

```
ActionController::Routing::Routes.draw do |map|
  # ...
  map.connect ':controller/:action'
end
```

http://localhost:3000/demo/read_something?value1=...&value2=...

```
class DemoController < ApplicationController

  def read_something
    # retrieve some result using params[:value1], params[:value2], ...
  end

  def change_something
    # update backend using params[:value1], params[:value2], ...
  end
end
```

Rails < 2.0

```
ActionController::Routing::Routes.draw do |map|
  # ...
  map.connect ':controller/:action'
end
```

http://localhost:3000/demo/read_something?value1=...&value2=...

```
class DemoController < ApplicationController

  def read_something
    # retrieve some result using params[:value1], params[:value2], ...
  end

  def change_something
    # update backend using params[:value1], params[:value2], ...
  end
end
```

Rails < 2.0

```
ActionController::Routing::Routes.draw do |map|
  # ...
  map.connect ':controller/:action'
end
```

http://localhost:3000/demo/read_something?value1=...&value2=...

```
class DemoController < ApplicationController

  def read_something
    # retrieve some result using params[:value1], params[:value2], ...
  end

  def change_something
    # update backend using params[:value1], params[:value2], ...
  end
end
```

Rails < 2.0

```ruby
ActionController::Routing::Routes.draw do |map|
  # ...
  map.connect ':controller/:action'
end
```

http://localhost:3000/demo/read_something?value1=...&value2=...

```ruby
class DemoController < ApplicationController

  def read_something
    # retrieve some result using params[:value1], params[:value2], ...
  end

  def change_something
    # update backend using params[:value1], params[:value2], ...
  end
end
```

Rails < 2.0

Default (incl. scaffolding) unRESTful

URIs identify *actions*

No difference between POST and GET by default

Typical PHP/Java Web programming model

Rails ≥ 2.0

RESTful scaffolding as default

REST considered best practice

Uniform resource interface mapped to standard-style controllers

Routing

Simple:

```
map.resources :orders
```

Nested:

```
map.resources :customers do |customers|
  customers.resources :orders
end
```

Prefixed:

```
map.resources :comments, :path_prefix => '/articles/:article_id'
```

Scaffolding

```
$ script/generate scaffold Order customer_id:integer description:string
      exists  app/models/
      exists  app/controllers/
      exists  app/helpers/
      create  app/views/orders
      exists  app/views/layouts/
      exists  test/functional/
      exists  test/unit/
      create  app/views/orders/index.html.erb
      create  app/views/orders/show.html.erb
      create  app/views/orders/new.html.erb
      create  app/views/orders/edit.html.erb
      create  app/views/layouts/orders.html.erb
   identical  public/stylesheets/scaffold.css
  dependency  model
      exists    app/models/
      exists    test/unit/
      exists    test/fixtures/
      create    app/models/order.rb
      create    test/unit/order_test.rb
      create    test/fixtures/orders.yml
      exists    db/migrate
      create    db/migrate/002_create_orders.rb
      create  app/controllers/orders_controller.rb
      create  test/functional/orders_controller_test.rb
      create  app/helpers/orders_helper.rb
       route  map.resources :orders
```

Controller Code

```
class OrdersController < ApplicationController
  # GET /orders
  # GET /orders.xml
  def index
  end

  # GET /orders/1
  # GET /orders/1.xml
  def show
  end

  # GET /orders/new
  # GET /orders/new.xml
  def new
  end

  # GET /orders/1/edit
  def edit
  end

  # POST /orders
  # POST /orders.xml
  def create
  end

  # PUT /orders/1
  # PUT /orders/1.xml
  def update
  end

  # DELETE /orders/1
  # DELETE /orders/1.xml
  def destroy
  end
end
```

Generated Routes

Output of rake routes:

```
            orders GET    /orders                {:controller=>"orders", :action=>"index"}
 formatted_orders GET    /orders.:format        {:controller=>"orders", :action=>"index"}
                  POST   /orders                {:controller=>"orders", :action=>"create"}
                  POST   /orders.:format        {:controller=>"orders", :action=>"create"}
        new_order GET    /orders/new            {:controller=>"orders", :action=>"new"}
 formatted_new_order GET    /orders/new.:format    {:controller=>"orders", :action=>"new"}
       edit_order GET    /orders/:id/edit       {:controller=>"orders", :action=>"edit"}
formatted_edit_order GET    /orders/:id/edit.:format {:controller=>"orders", :action=>"edit"}
            order GET    /orders/:id            {:controller=>"orders", :action=>"show"}
 formatted_order GET    /orders/:id.:format    {:controller=>"orders", :action=>"show"}
                  PUT    /orders/:id            {:controller=>"orders", :action=>"update"}
                  PUT    /orders/:id.:format    {:controller=>"orders", :action=>"update"}
                  DELETE /orders/:id            {:controller=>"orders", :action=>"destroy"}
                  DELETE /orders/:id.:format    {:controller=>"orders", :action=>"destroy"}
```

ActiveResource

Client for RESTful Rails Resources

Basic support for ActiveRecord-style code

```
class Person < ActiveResource::Base
  self.site = "http://api.people.com:3000/"
end

# Find a person with id = 1
ryan = Person.find(1)
Person.exists?(1)  #=> true
```

REST ≠ CRUD

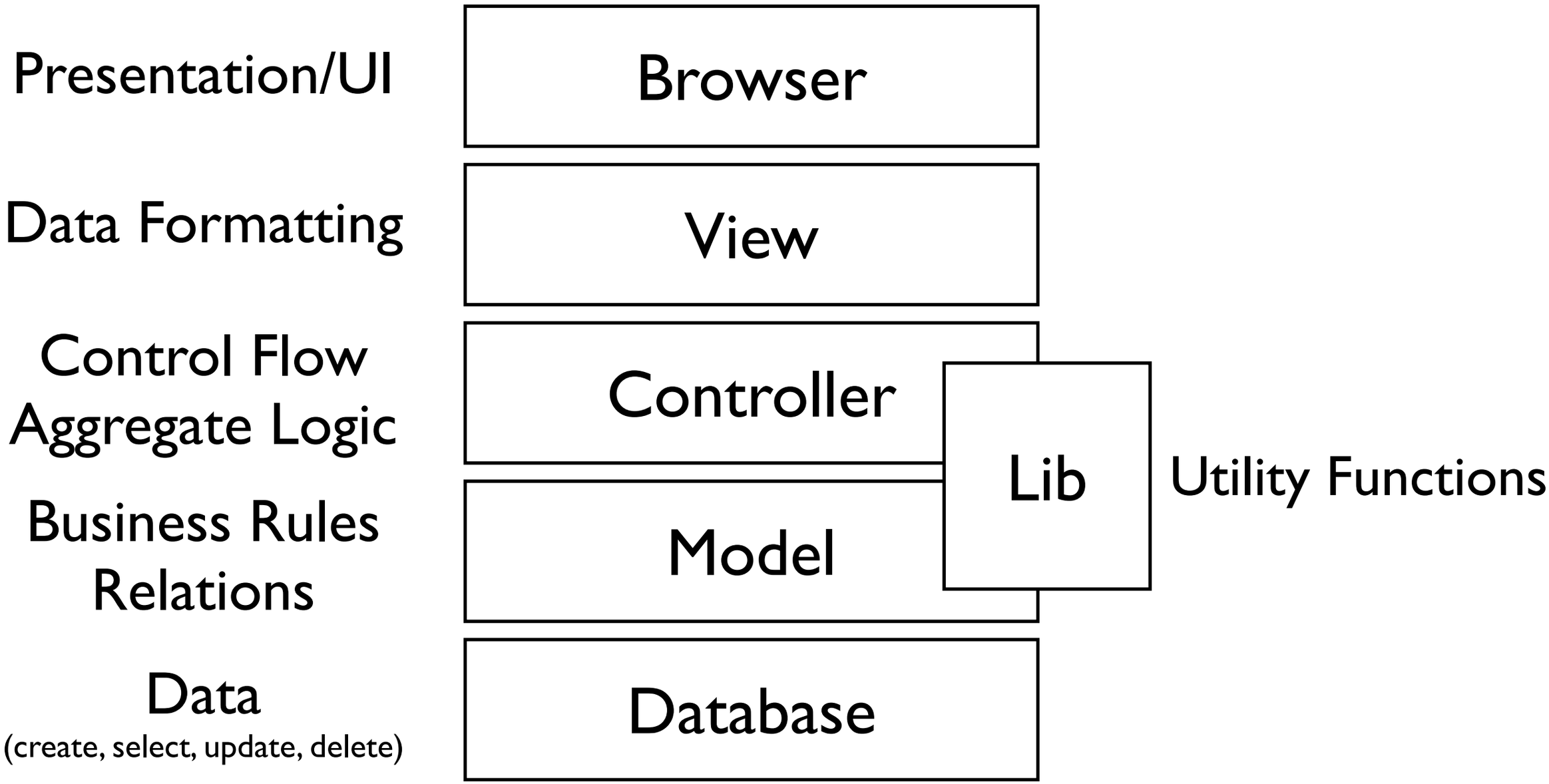
?

Resource ≠ Entity

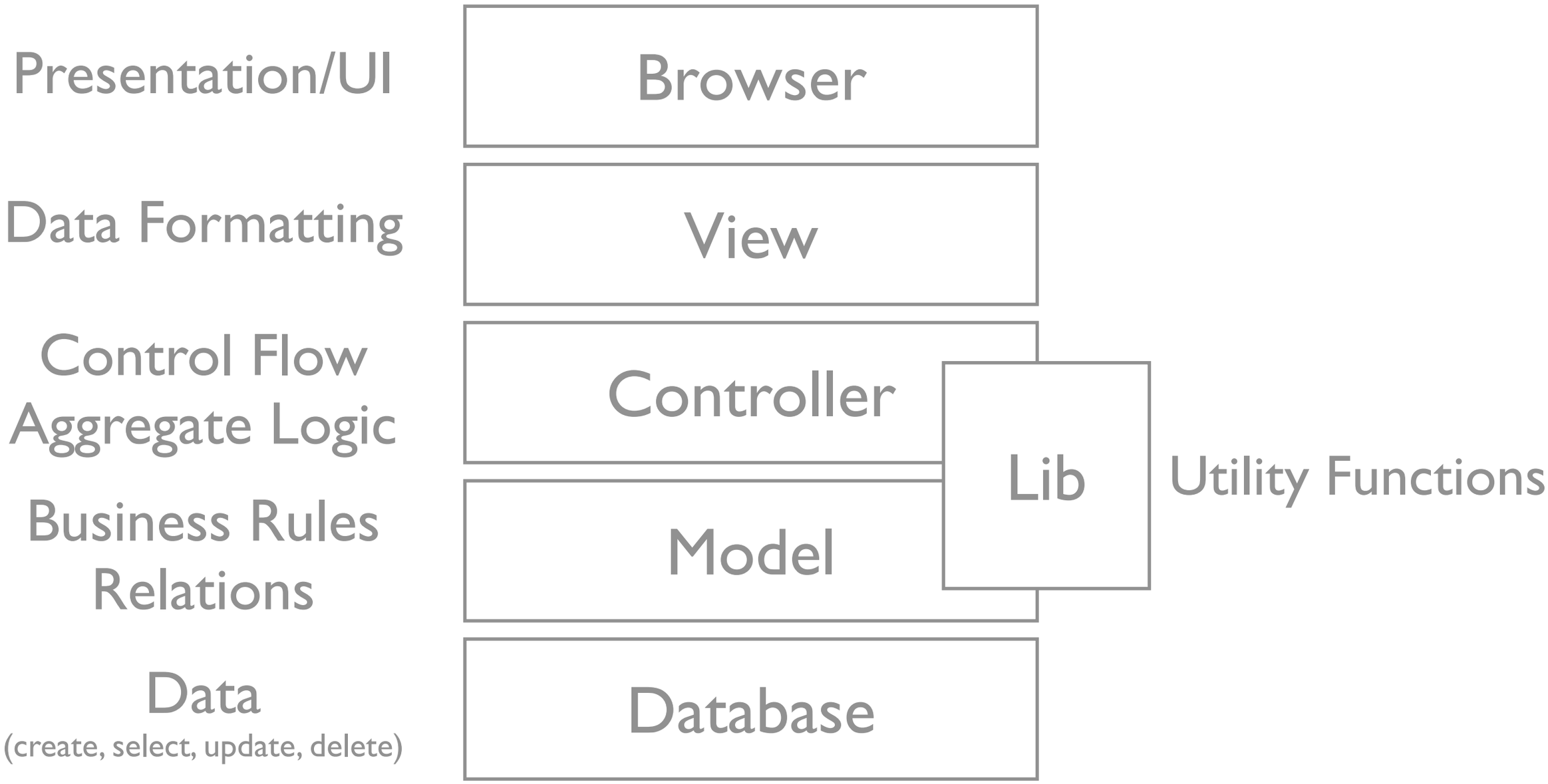
Resource ≈ Model

Resource ≈ Controller

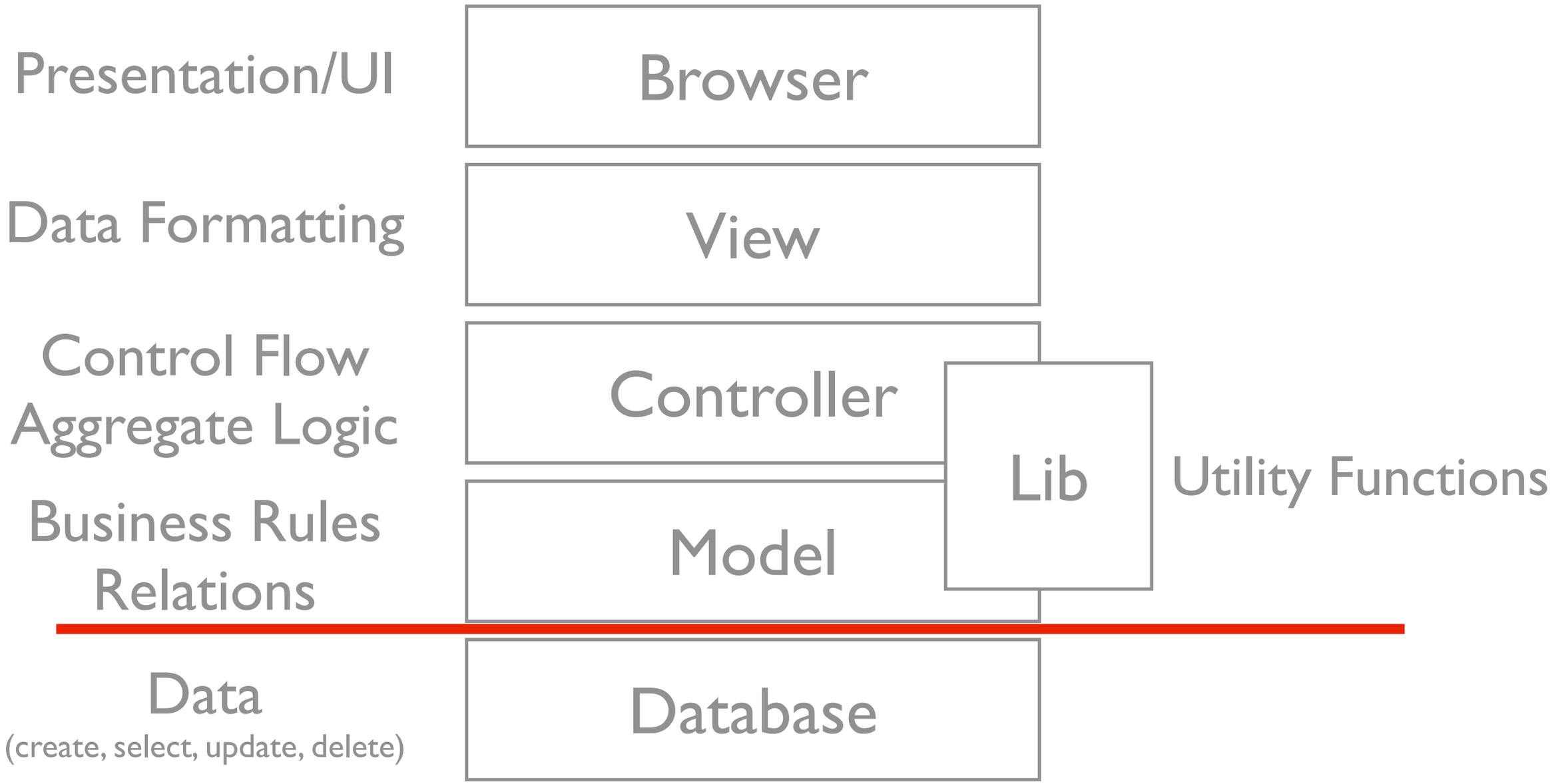
Rails App Layers



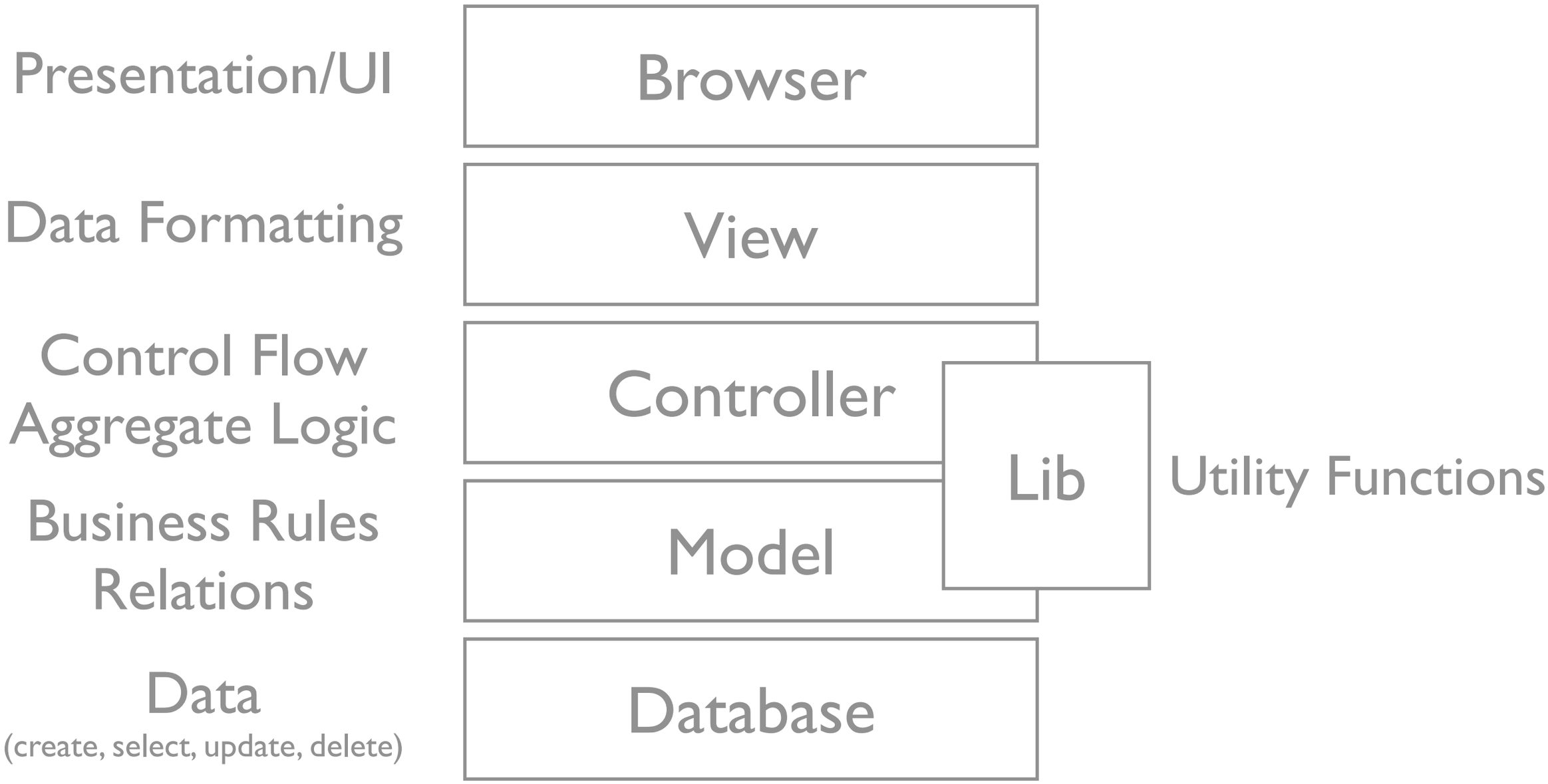
Rails App Layers & Resources



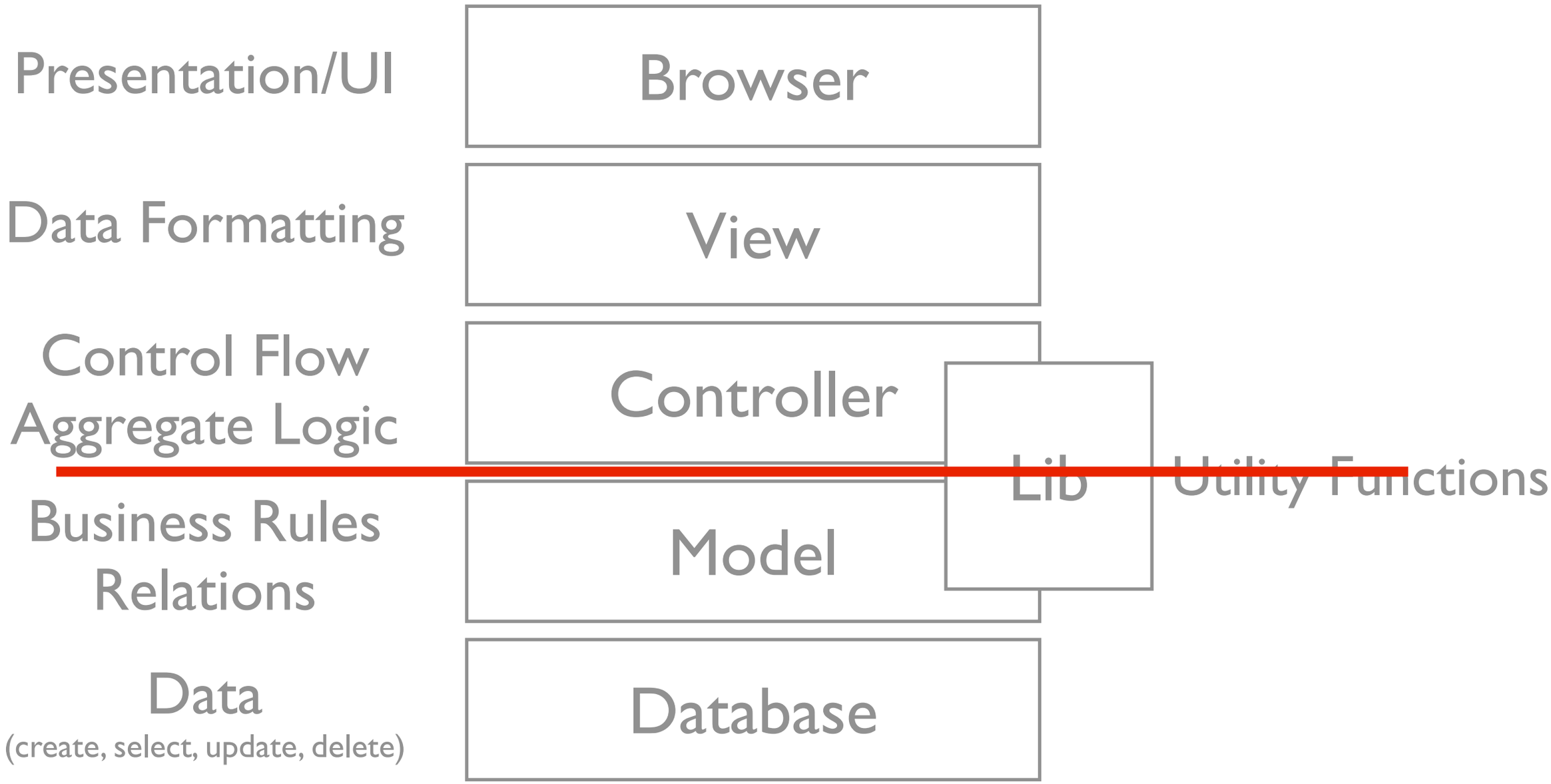
Rails App Layers & Resources



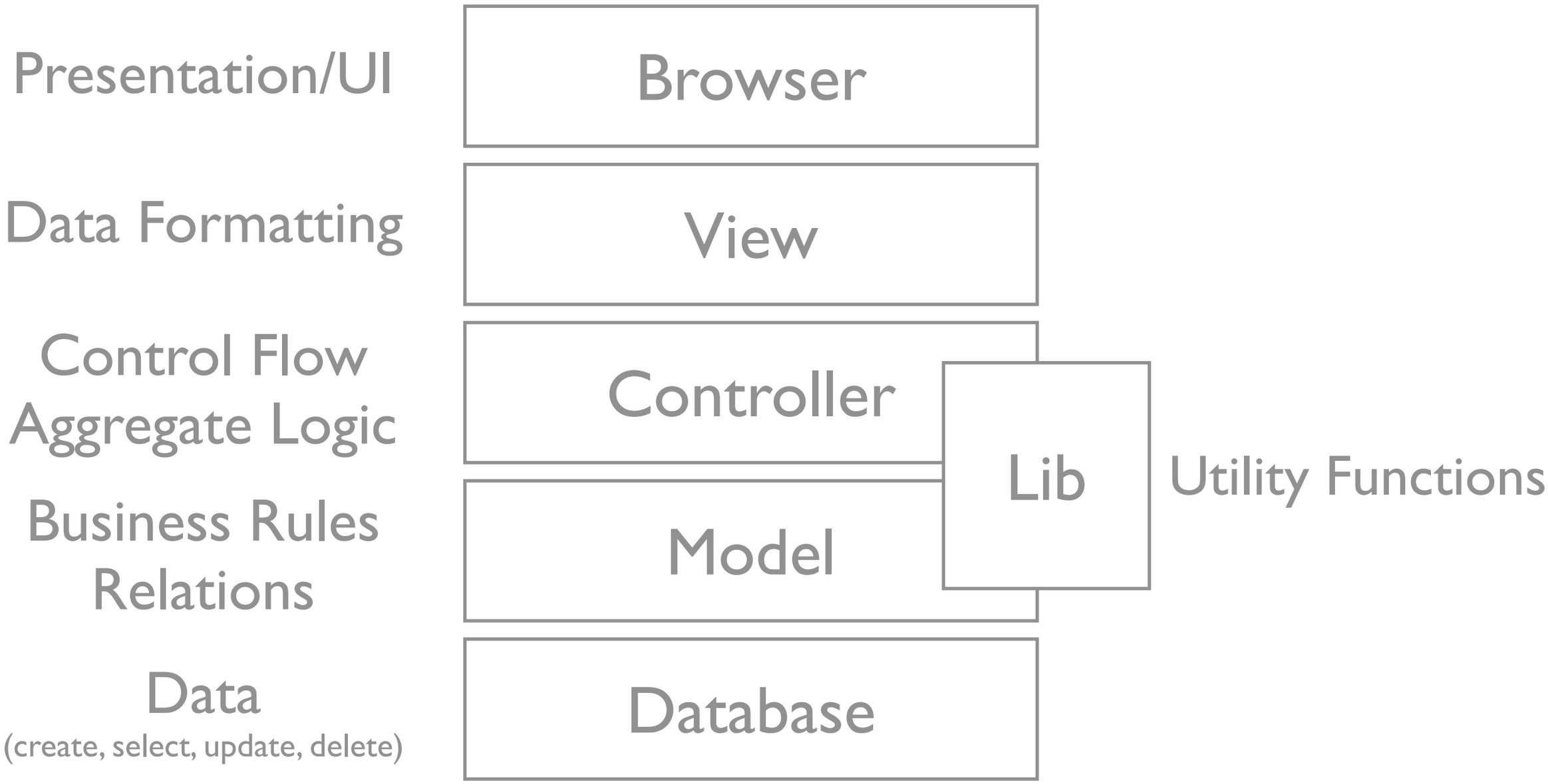
Rails App Layers & Resources



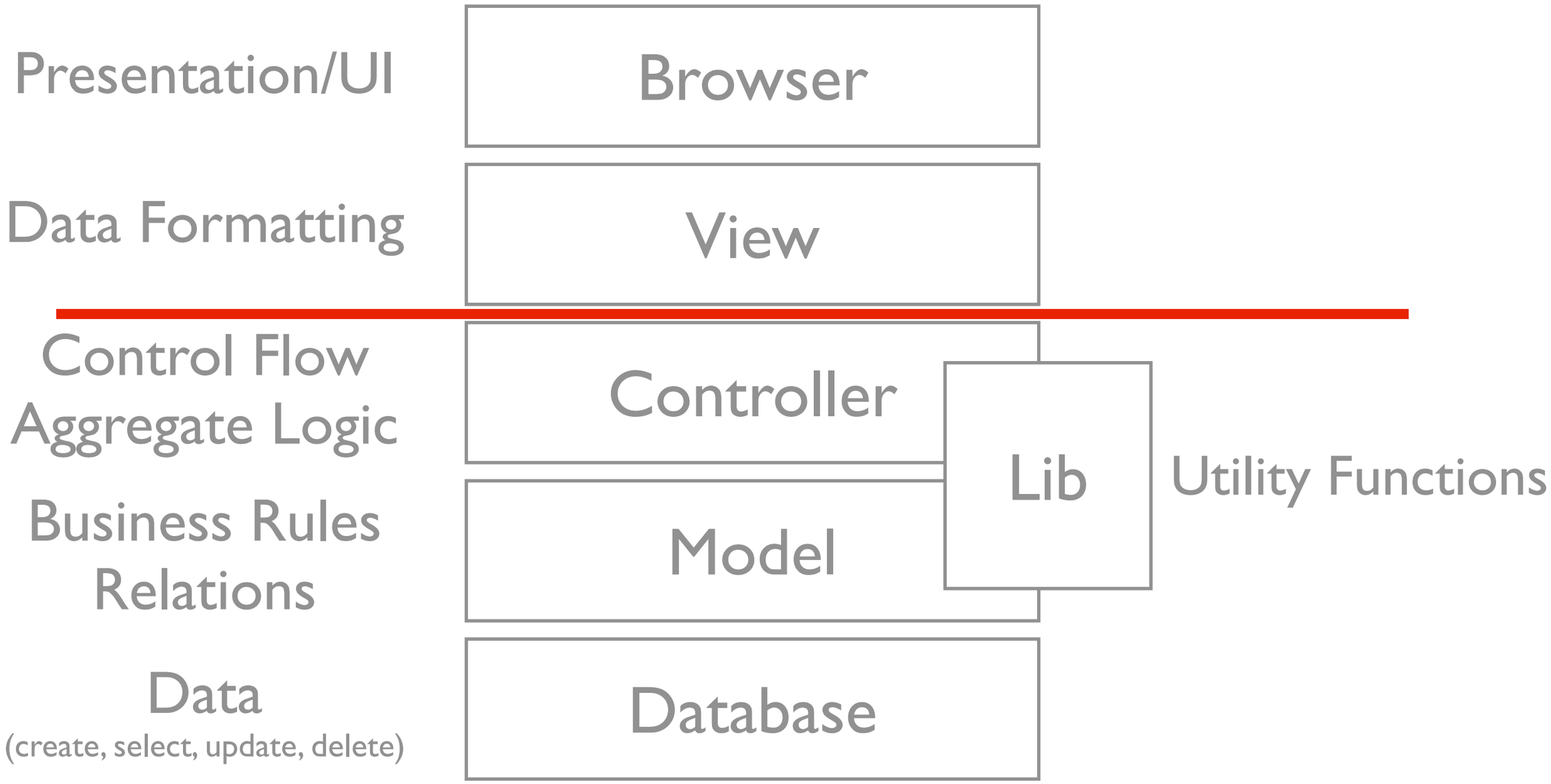
Rails App Layers & Resources



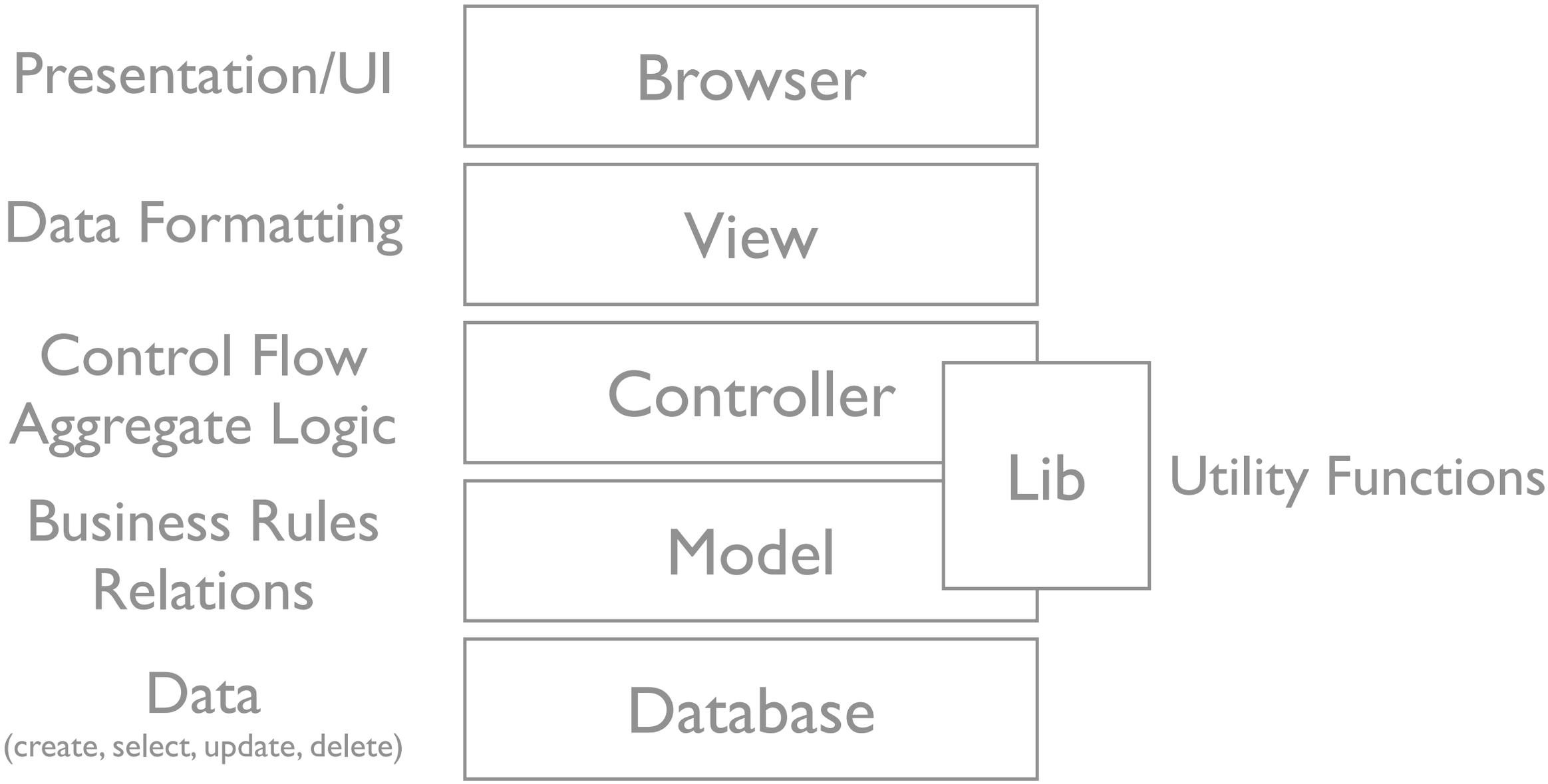
Rails App Layers & Resources



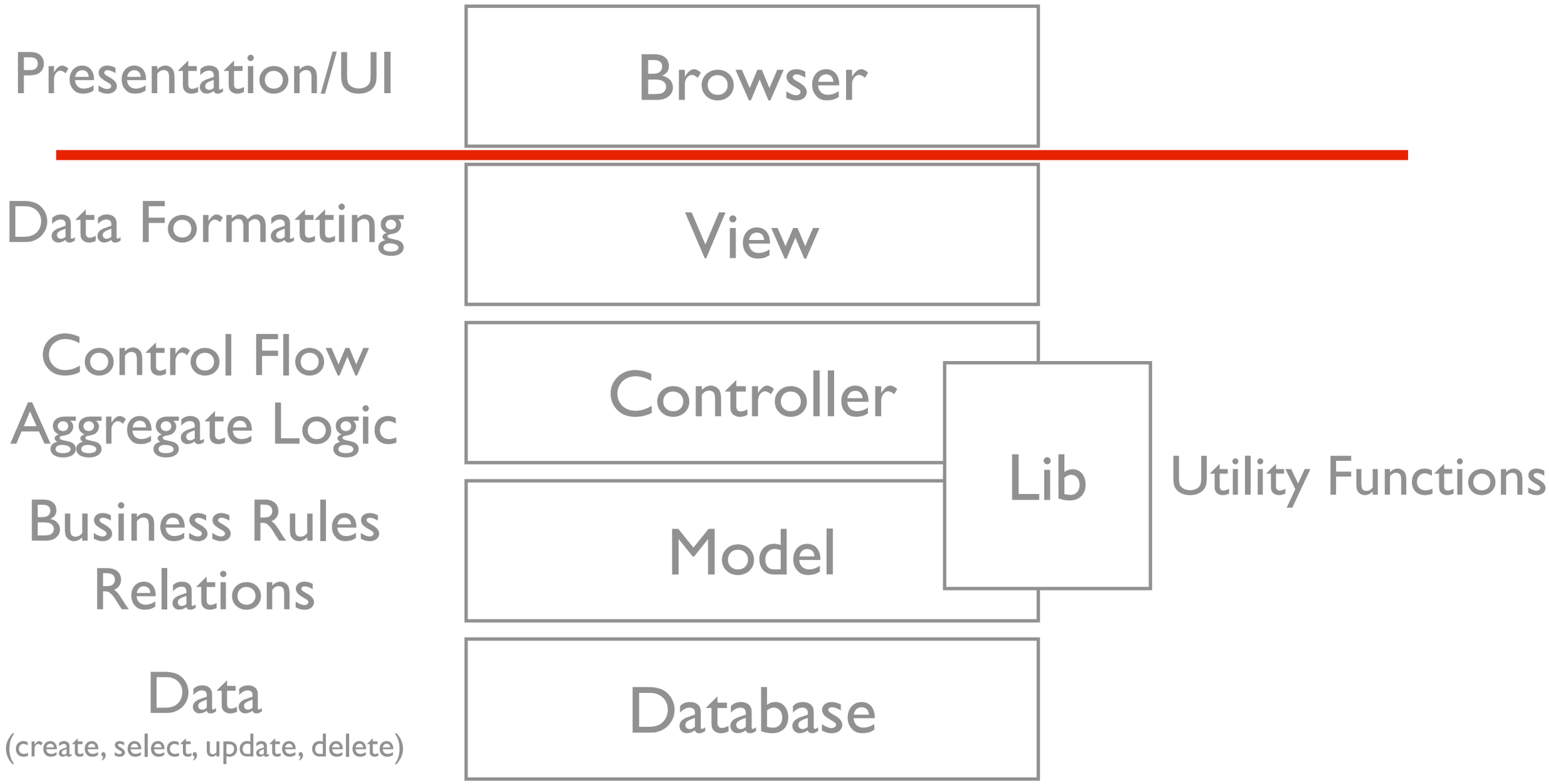
Rails App Layers & Resources



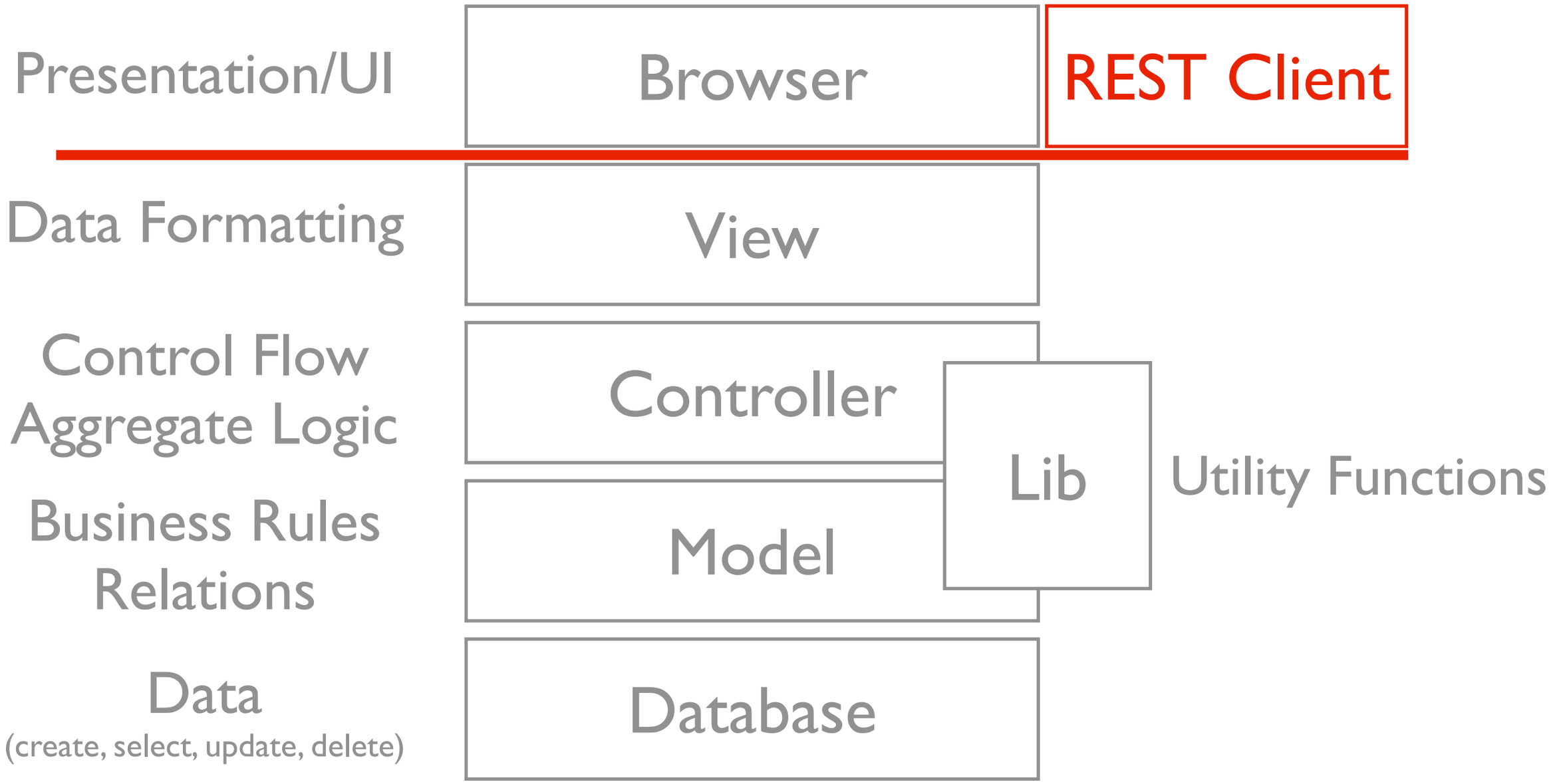
Rails App Layers & Resources



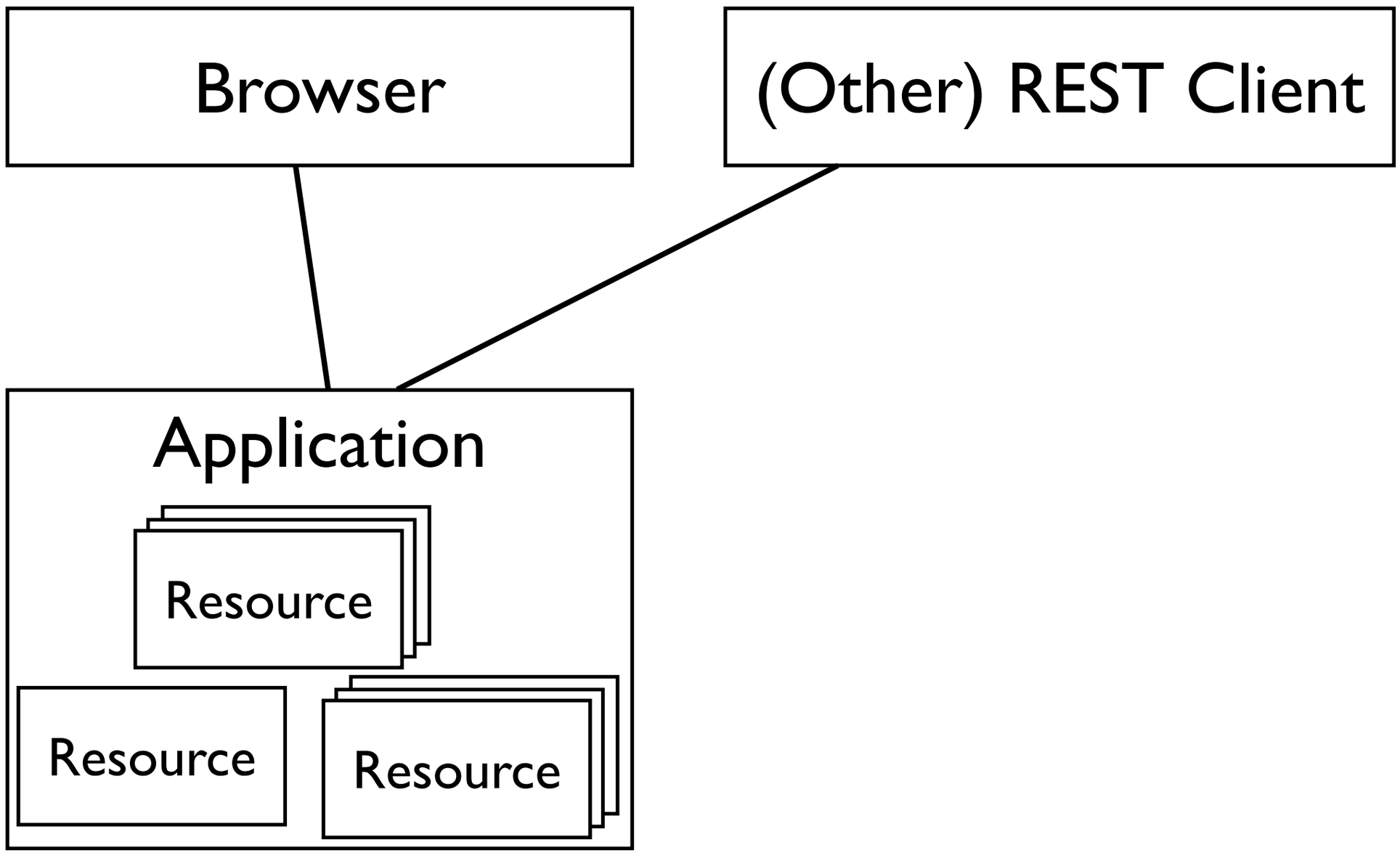
Rails App Layers & Resources



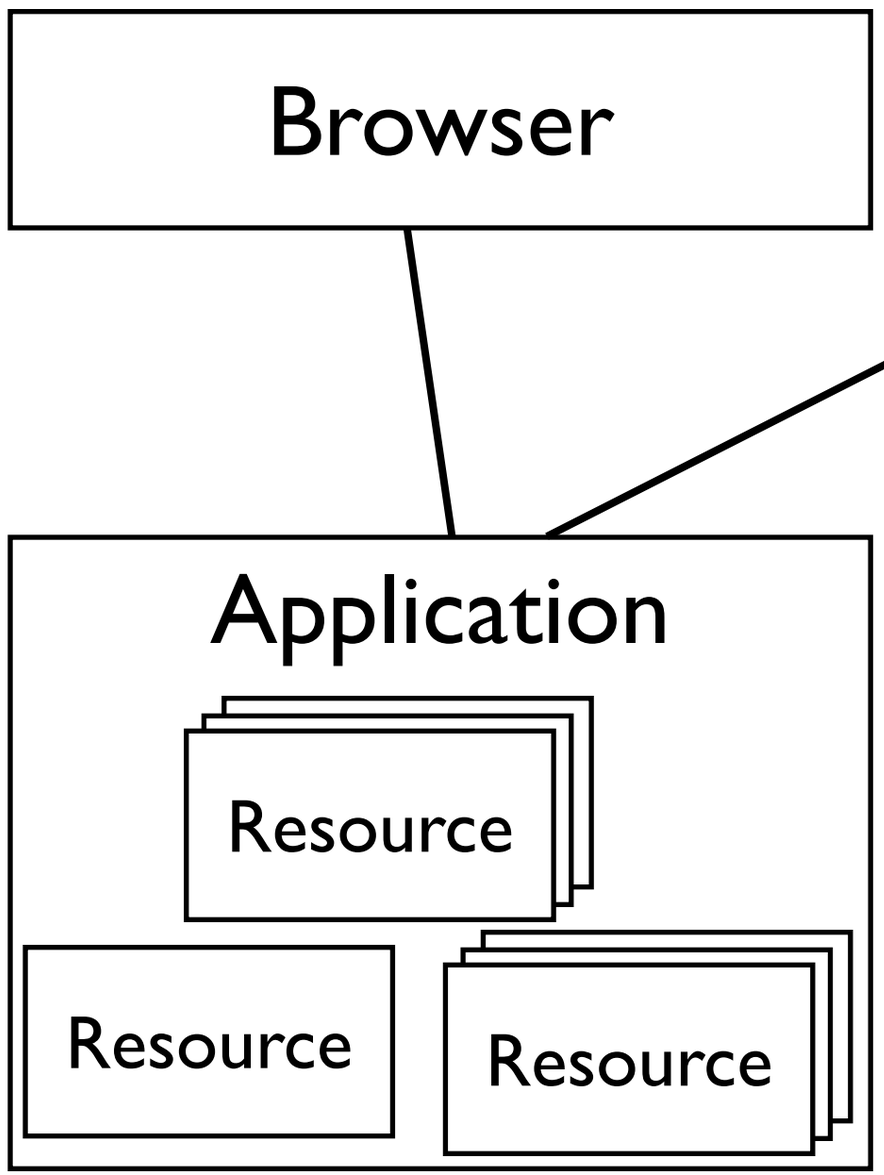
Rails App Layers & Resources



Single Resource Model

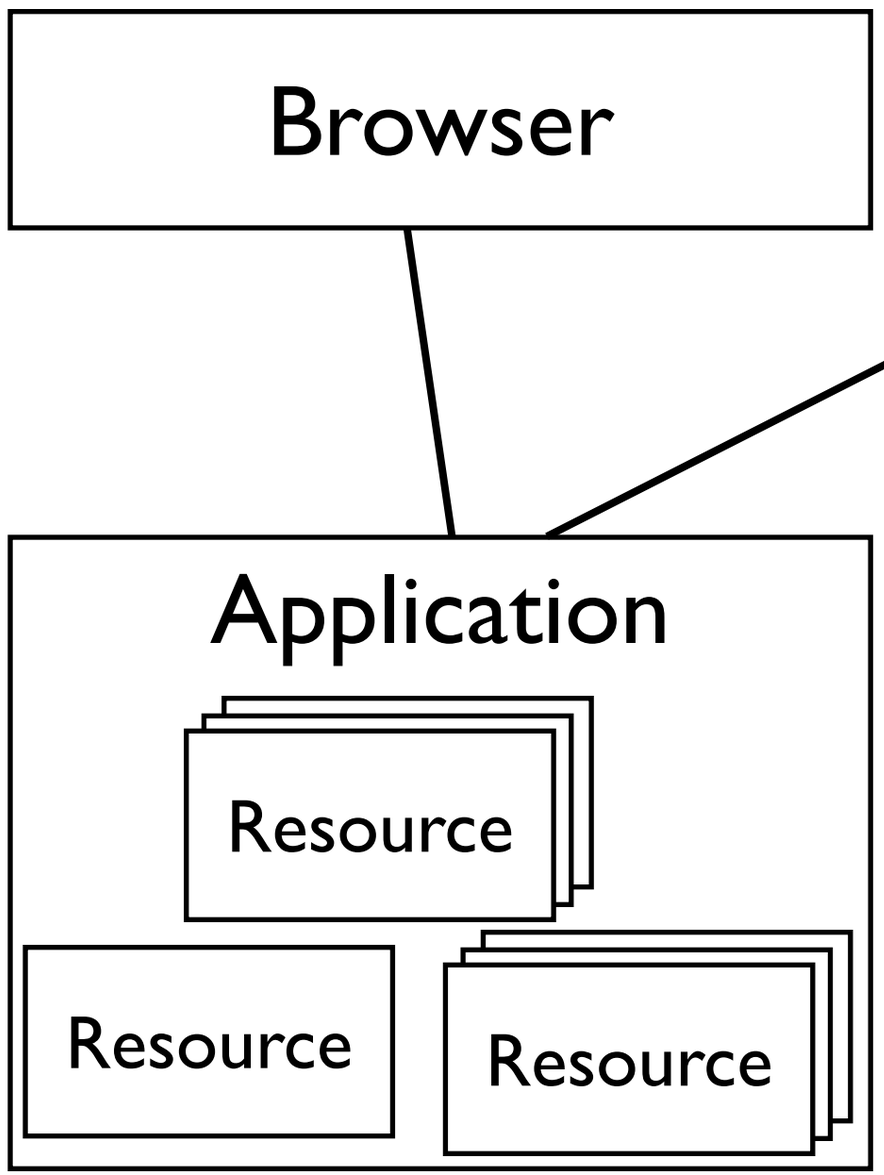


Single Resource Model



Everything doable via UI …

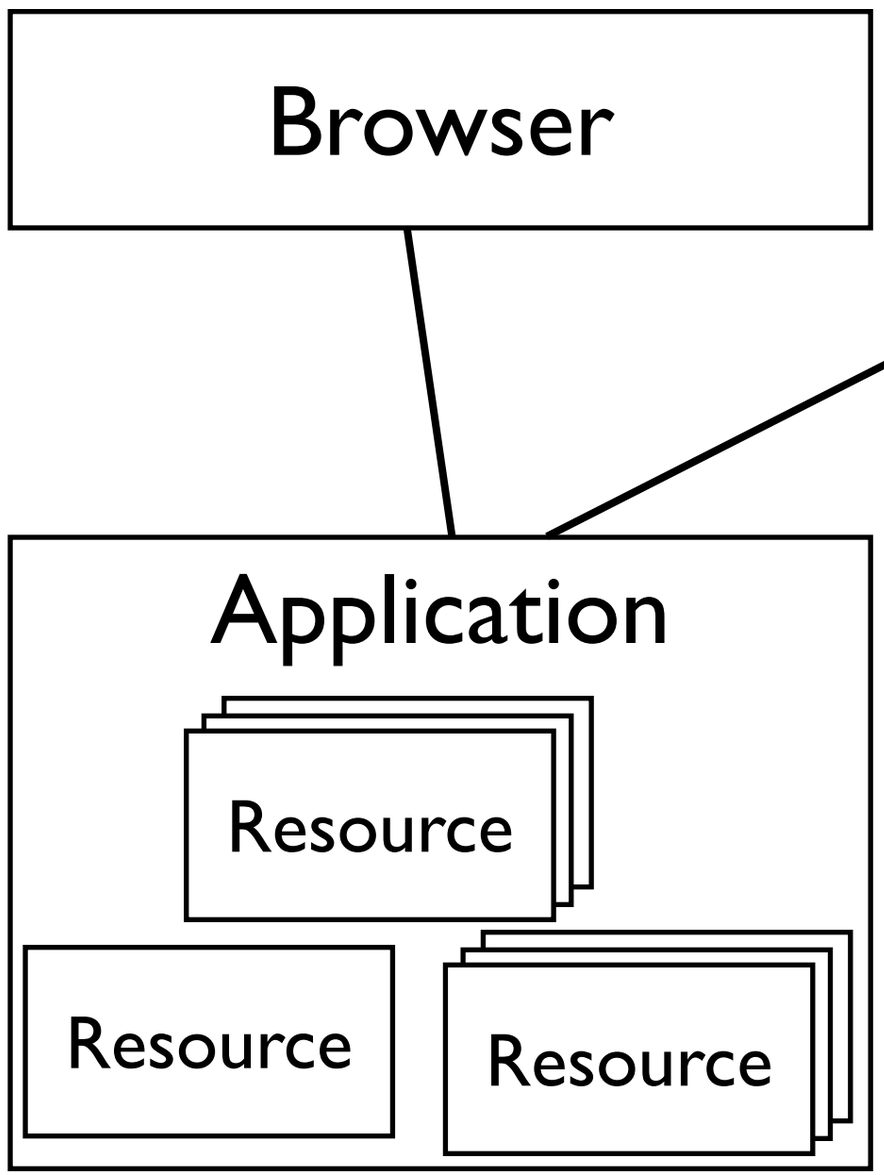
Single Resource Model



Everything doable via UI …

… becomes doable via API

Single Resource Model

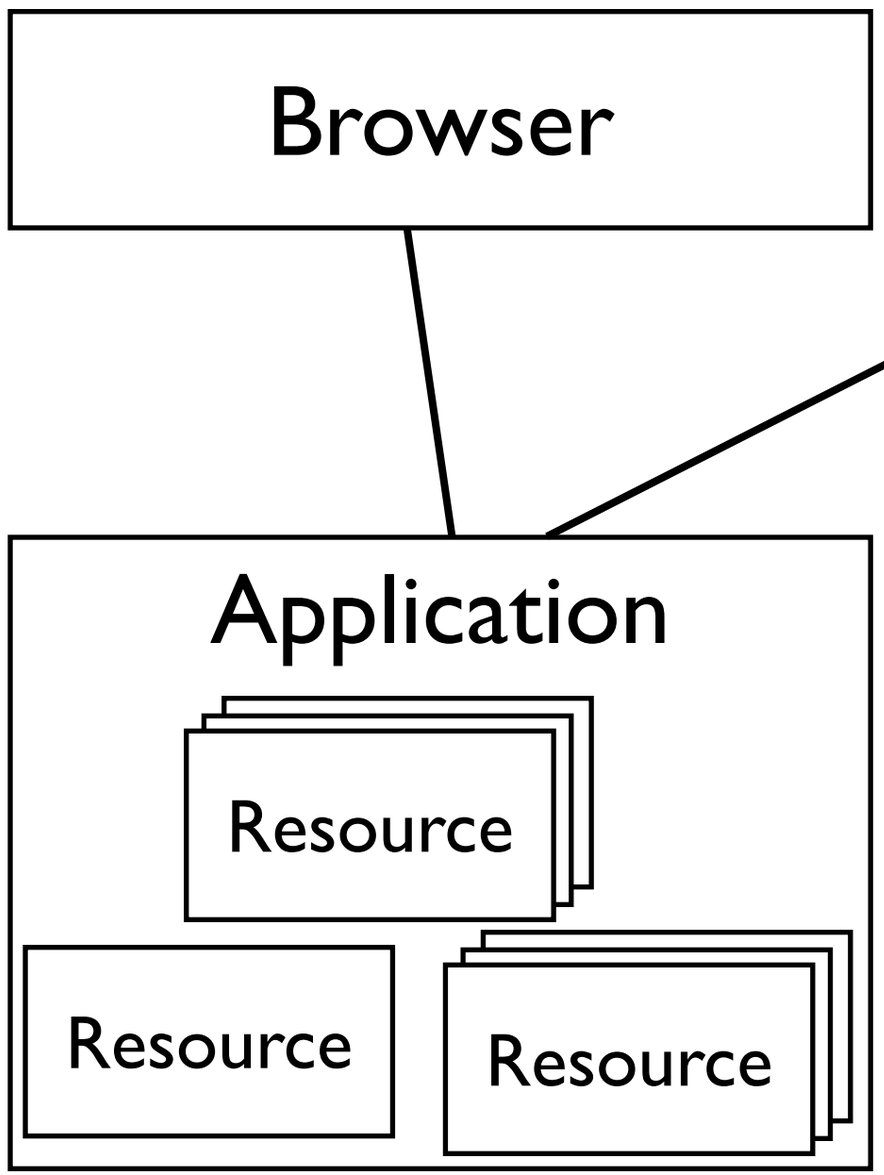


Everything doable via UI …

… becomes doable via API

UI Backend = API

Single Resource Model



Everything doable via UI …

… becomes doable via API

UI Backend = API

RESTful APIs

RESTful APIs don’t expose low-level details

Same layer – different abstraction

Value through uniformity and hypermedia

Mapping necessity: “Implement” HTTP base interface

Mapping Examples

Mapping Examples

getFreeTimeSlots(Person)

Mapping Examples

getFreeTimeSlots(Person)	→GET /people/{id}/timeslots?state=free

Mapping Examples

getFreeTimeSlots(Person)	→GET /people/{id}/timeslots?state=free
rejectApplication(Application)	

Mapping Examples

getFreeTimeSlots(Person)	→GET /people/{id}/timeslots?state=free
rejectApplication(Application)	→POST /rejections↵ <application>http://...</application>↵ <reason>Unsuitable for us!</reason>

Mapping Examples

getFreeTimeSlots(Person)	→GET /people/{id}/timeslots?state=free
rejectApplication(Application)	→POST /rejections↵ <application>http://...</application>↵ <reason>Unsuitable for us!</reason>
performTariffCalculation(Data)	

Mapping Examples

getFreeTimeSlots(Person)	→GET /people/{id}/timeslots?state=free
rejectApplication(Application)	→POST /rejections↵ <application>http://...</application>↵ <reason>Unsuitable for us!</reason>
performTariffCalculation(Data)	→POST /calculations↵ Data ←Location: http://.../calculations/4711 →GET /calculations/4711 ←Result

Mapping Examples

getFreeTimeSlots(Person)	→GET /people/{id}/timeslots?state=free
rejectApplication(Application)	→POST /rejections↵ <application>http://...</application>↵ <reason>Unsuitable for us!</reason>
performTariffCalculation(Data)	→POST /calculations↵ Data ←Location: http://.../calculations/4711 →GET /calculations/4711 ←Result
shipOrder(ID)	

Mapping Examples

getFreeTimeSlots(Person)	→GET /people/{id}/timeslots?state=free
rejectApplication(Application)	→POST /rejections↵ <application>http://...</application>↵ <reason>Unsuitable for us!</reason>
performTariffCalculation(Data)	→POST /calculations↵ Data ←Location: http://.../calculations/4711 →GET /calculations/4711 ←Result
shipOrder(ID)	→PUT /orders/0815↵ <status>shipped</status>

Mapping Examples

getFreeTimeSlots(Person)	→GET /people/{id}/timeslots?state=free
rejectApplication(Application)	→POST /rejections↵ <application>http://...</application>↵ <reason>Unsuitable for us!</reason>
performTariffCalculation(Data)	→POST /calculations↵ Data ←Location: http://.../calculations/4711 →GET /calculations/4711 ←Result
shipOrder(ID)	→PUT /orders/0815↵ <status>shipped</status>
shipOrder(ID) [variation]	

Mapping Examples

getFreeTimeSlots(Person)	→GET /people/{id}/timeslots?state=free
rejectApplication(Application)	→POST /rejections↵ <application>http://...</application>↵ <reason>Unsuitable for us!</reason>
performTariffCalculation(Data)	→POST /calculations↵ Data ←Location: http://.../calculations/4711 →GET /calculations/4711 ←Result
shipOrder(ID)	→PUT /orders/0815↵ <status>shipped</status>
shipOrder(ID) [variation]	→POST /shipments↵ Data ←Location: http://.../shipments/4711

RESTful HTTP's Role for Rails

Application Size	Modularization
0-50 LOC	1 file
50-500 LOC	few files, many functions
500-1000 LOC	library, class hierarchy
1000-2000 LOC	framework + application
>2000 LOC	more than one application

How RESTful is Rails?

Positive:

Consistent and clean CRUD mapping

Use of URIs for resource identification

Support for content negotiation

Reasonable Status codes

ETags (!)

How RESTful is Rails?

Negative:

No hypermedia

No deep ETags

CRUD-centric

Proprietary protocol for ActiveResource

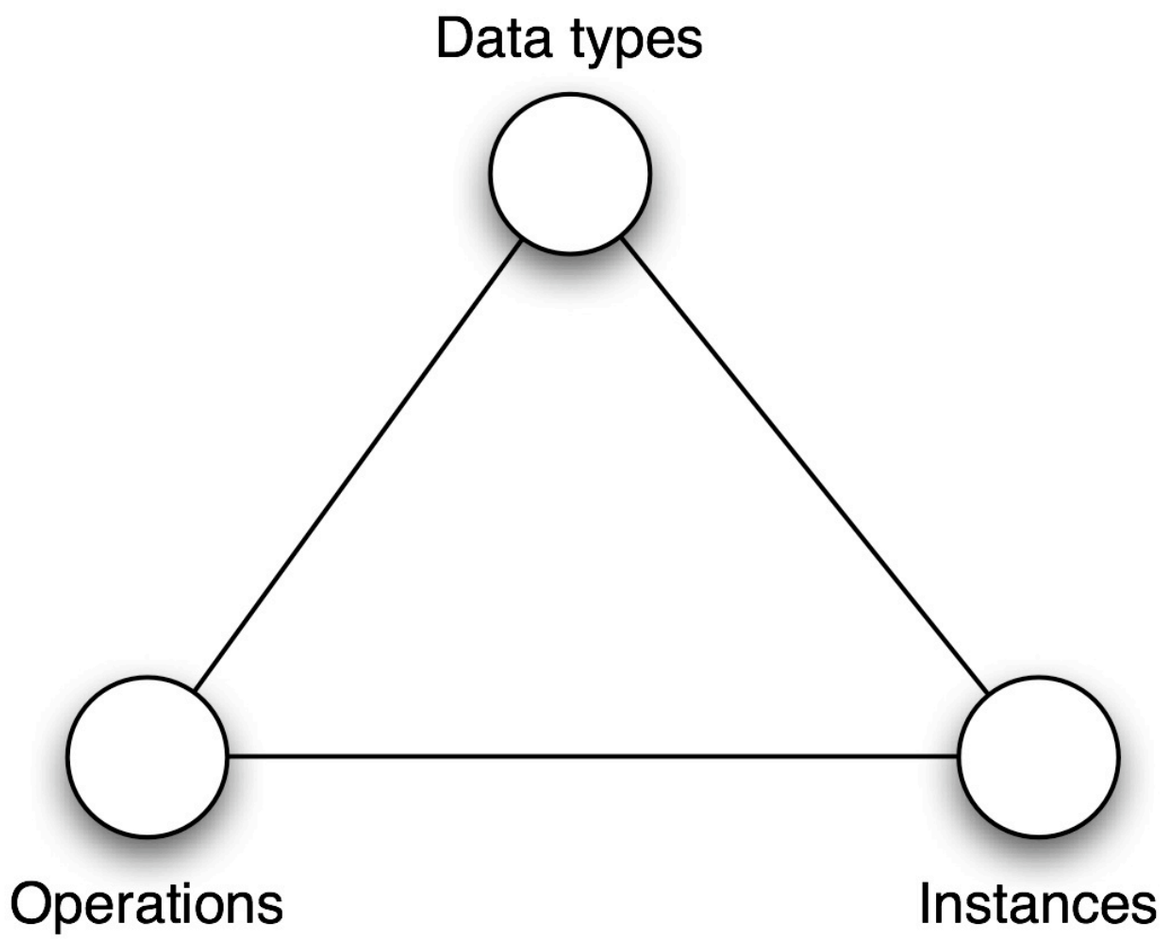
My Rails/REST Wishlist

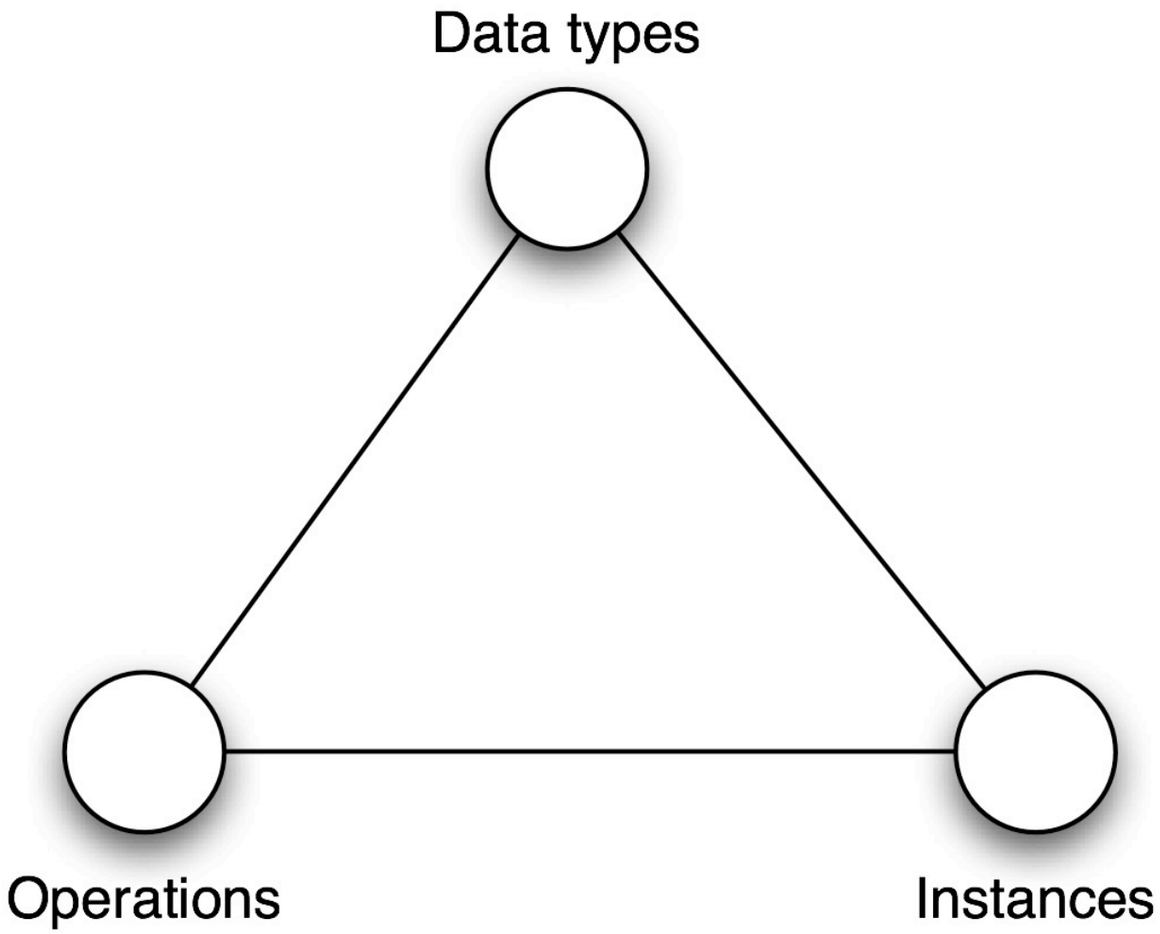
A really cool, meta-driven hypermedia programming model

for both client and server (w/o coupling)

Atom Syndication and Atom Pub Support

Backup



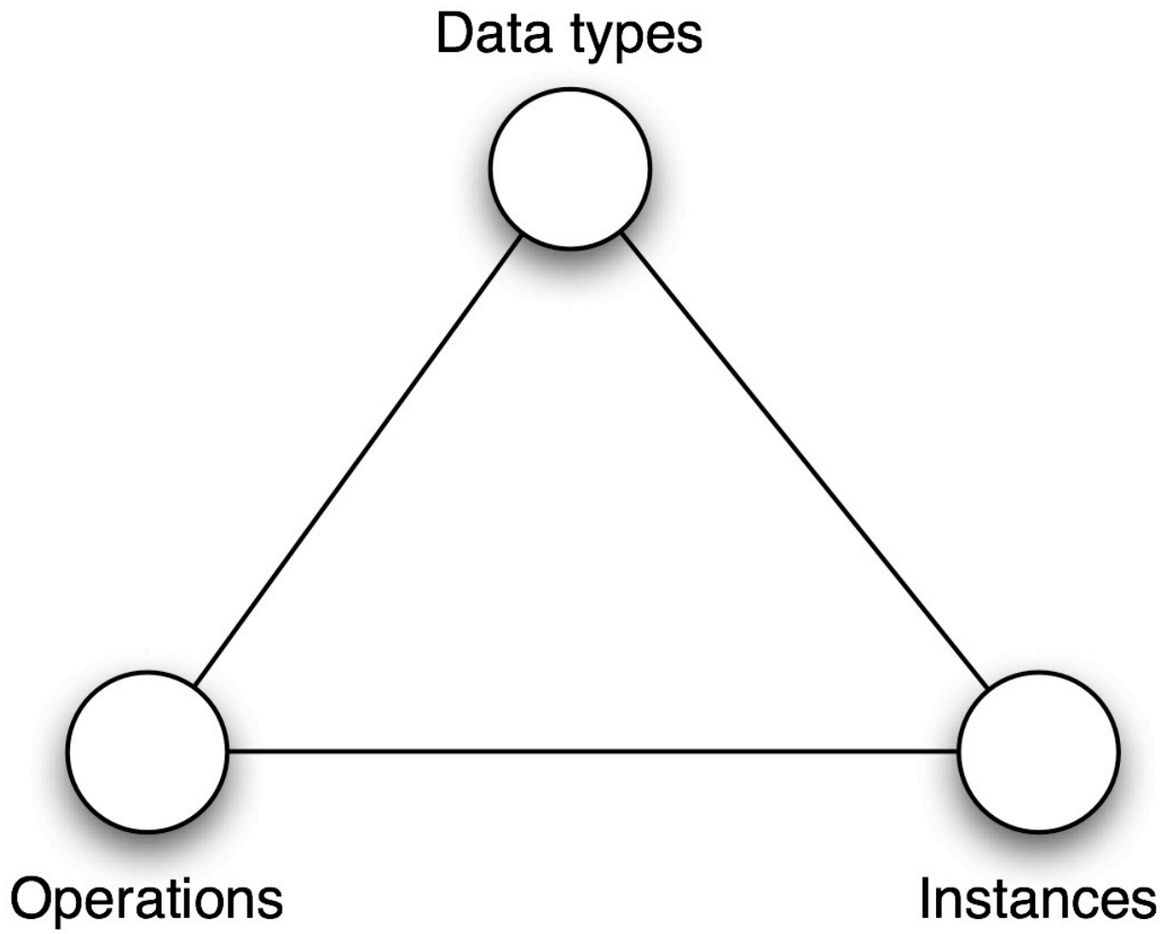


OrderManagementService

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

CustomerManagementService

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



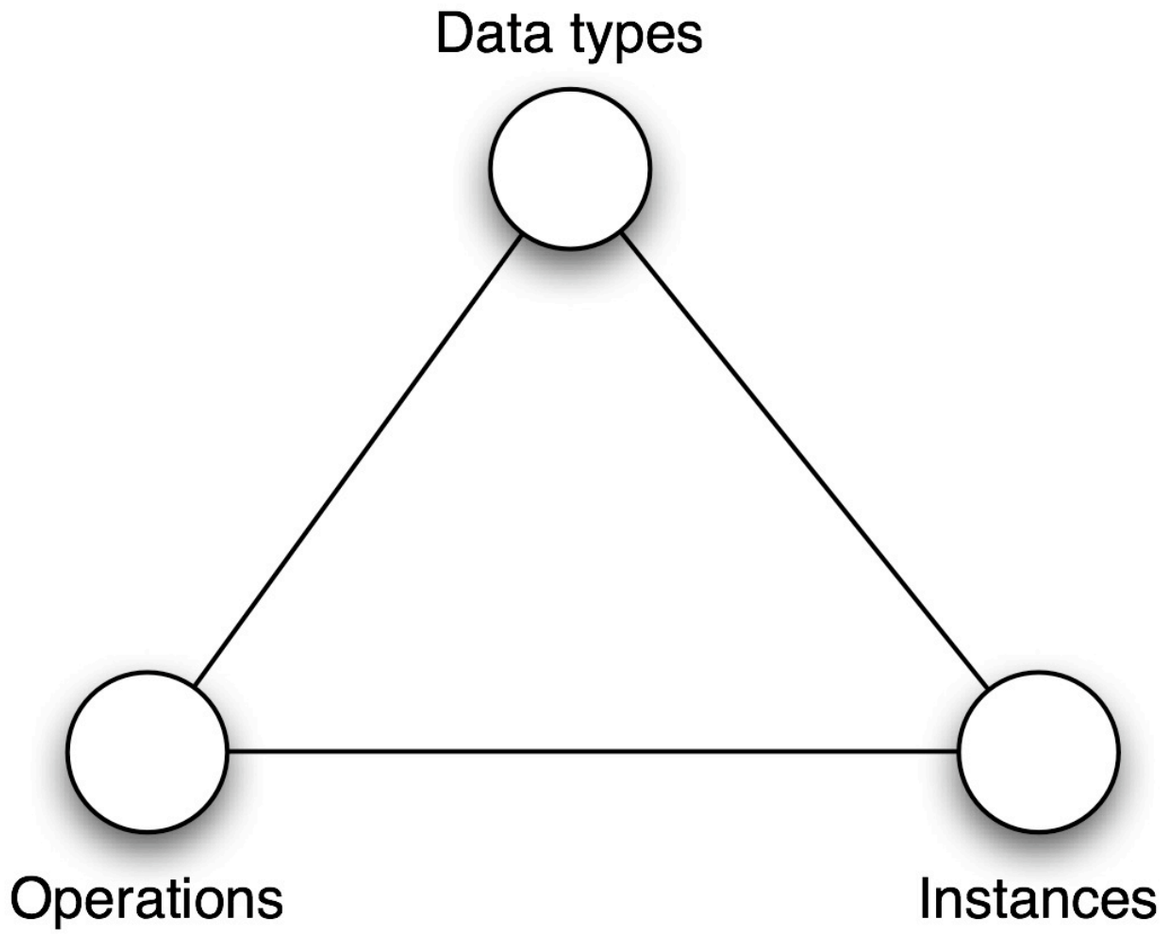
many

OrderManagementService

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

CustomerManagementService

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



many

very few

(one per service)

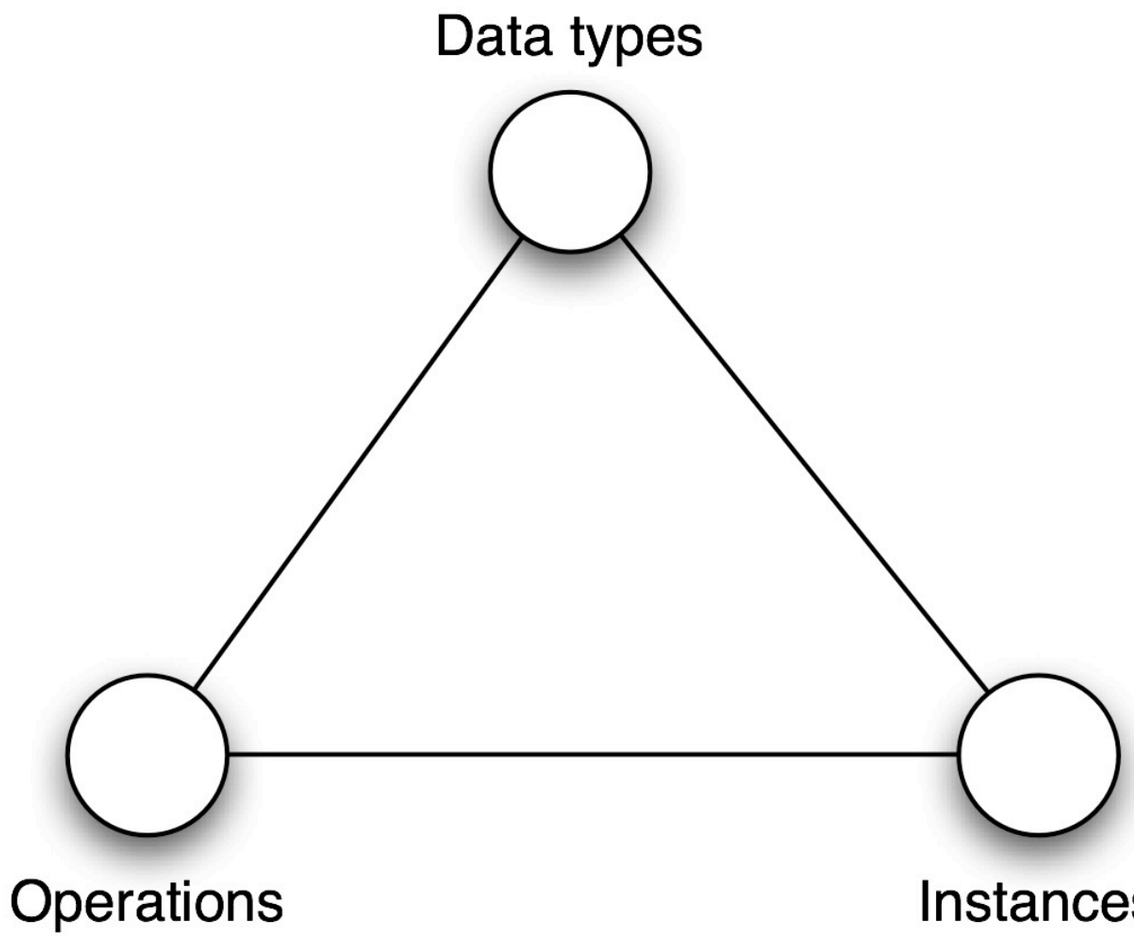
OrderManagementService

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

CustomerManagementService

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

many



many

very few

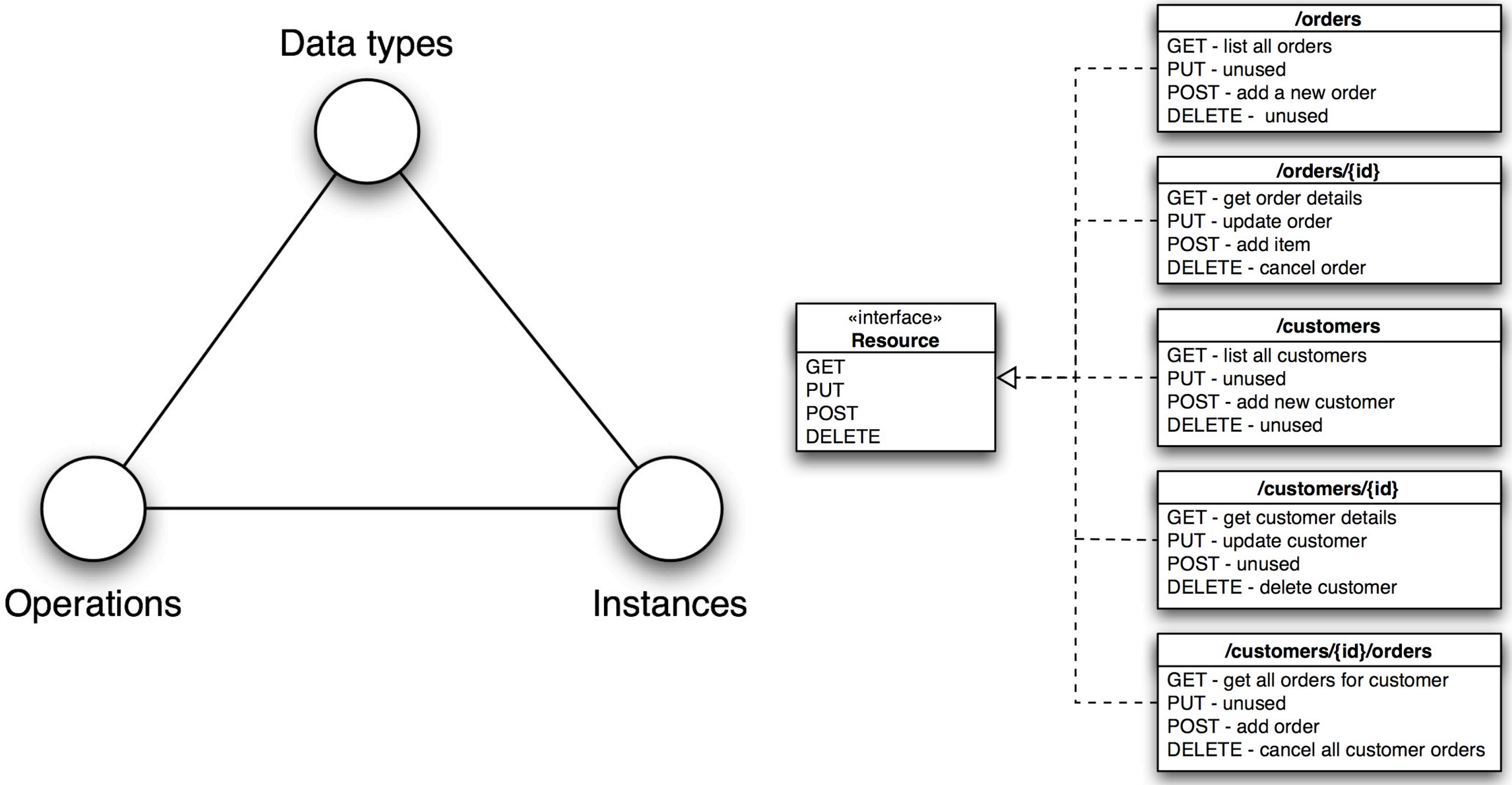
(one per service)

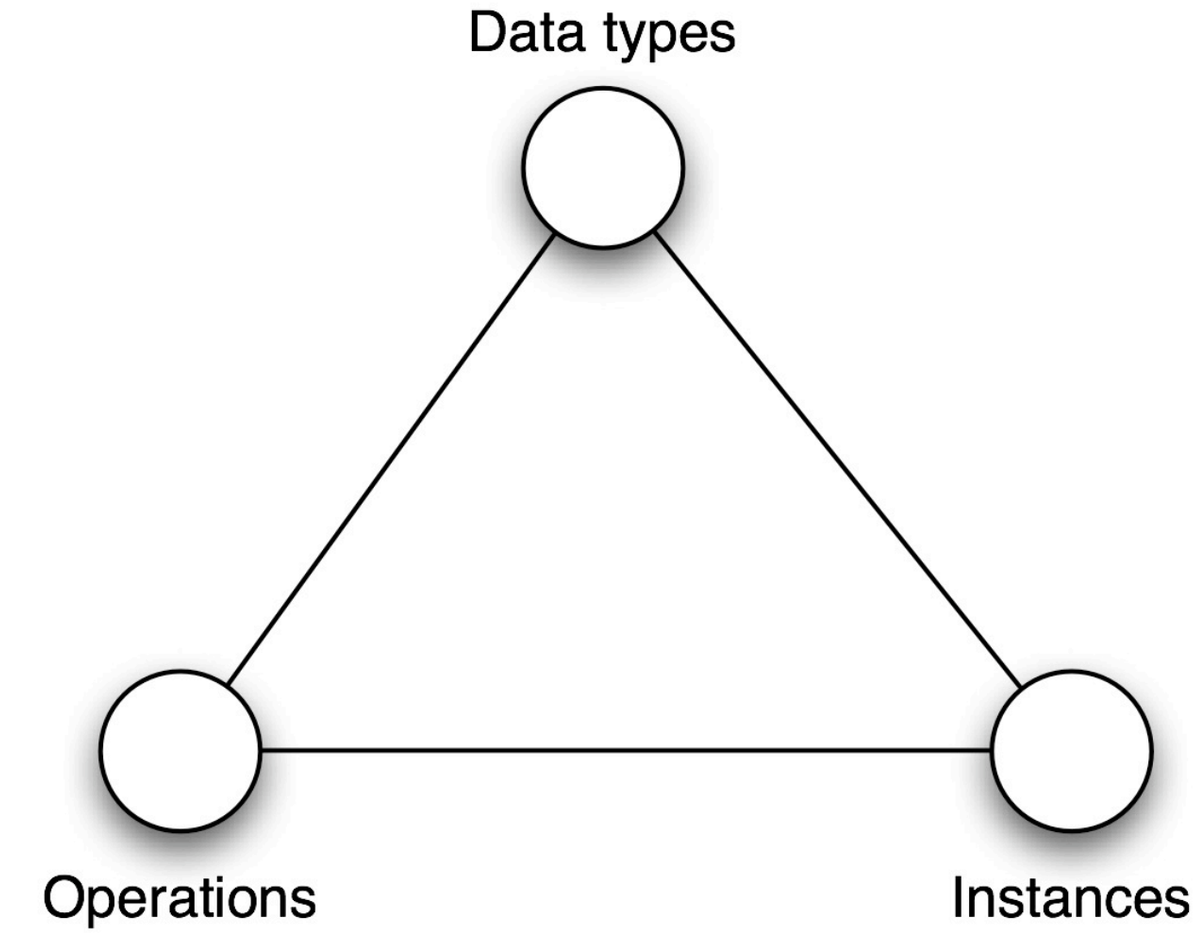
OrderManagementService

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

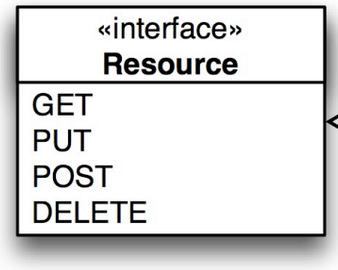
CustomerManagementService

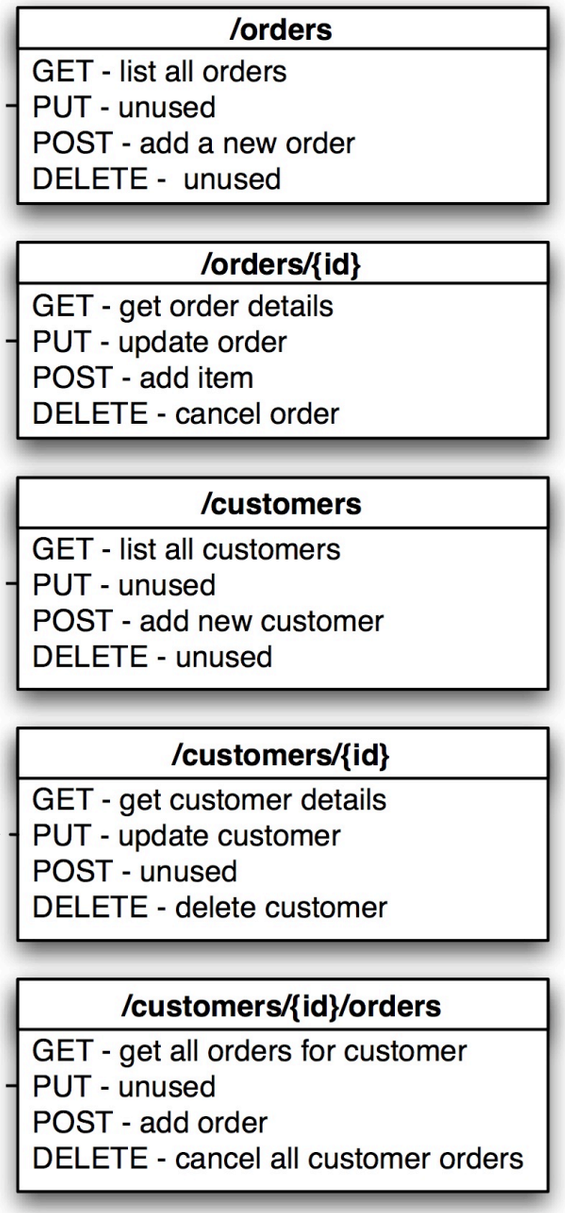
+ getCustomers()
+ addCustomer()
+ getCustomerDetails()
+ updateCustomer()
+ deleteCustomer()

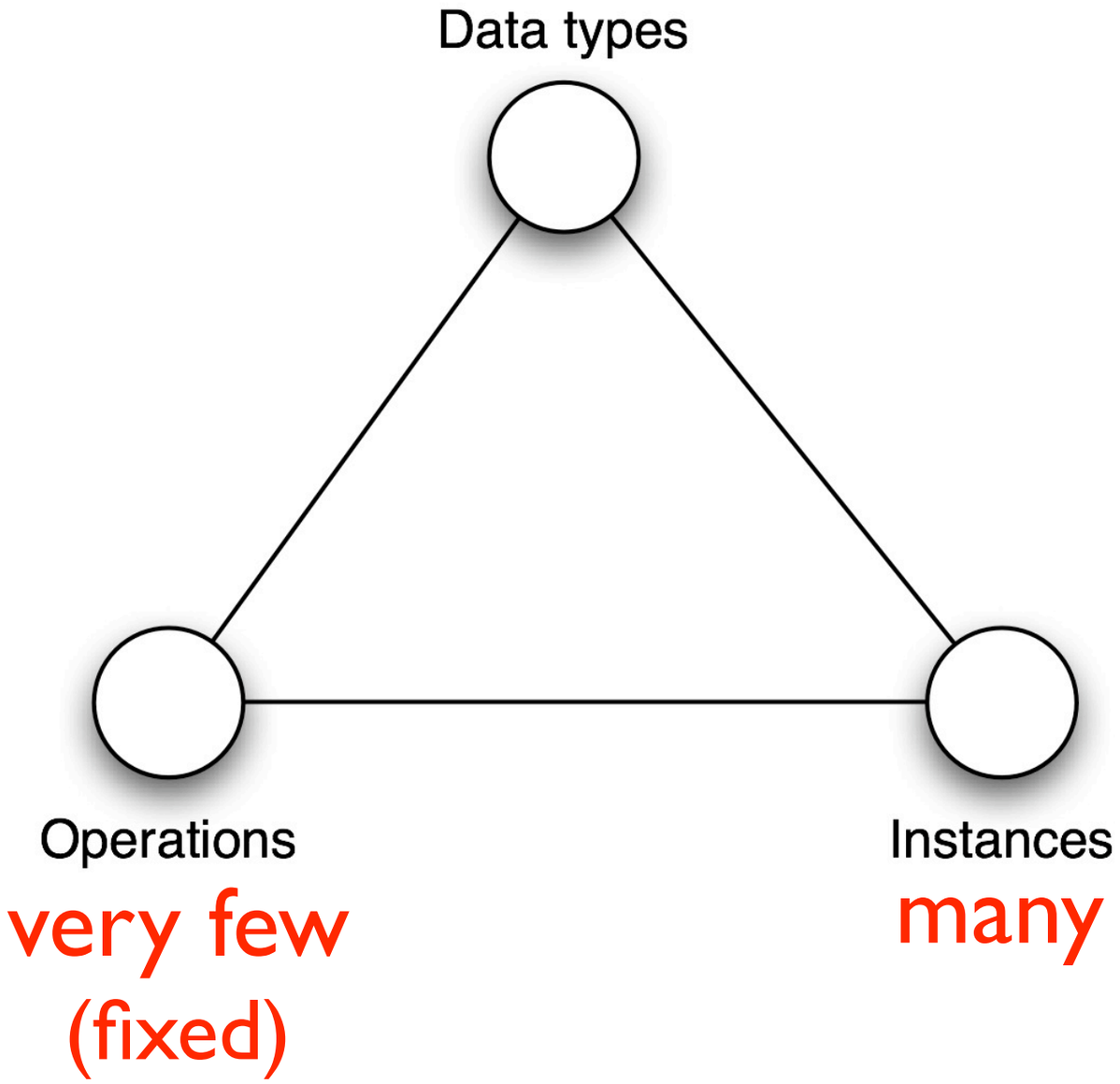


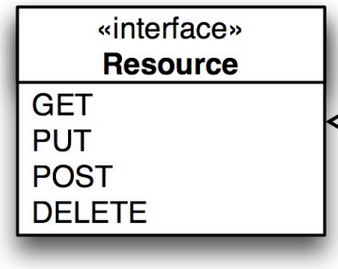


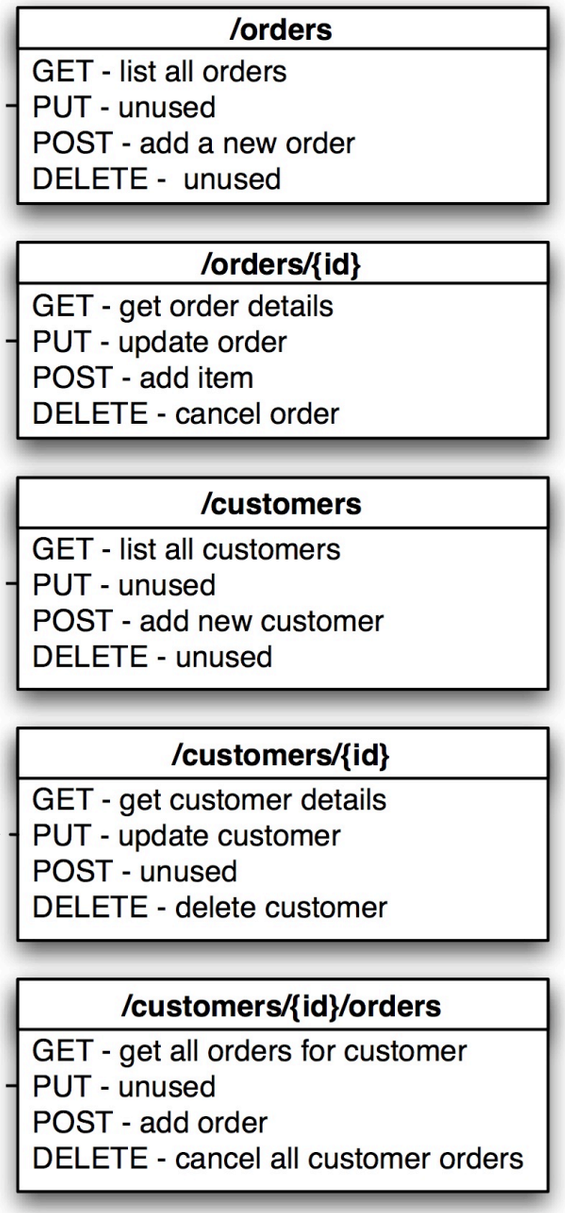
very few
(fixed)



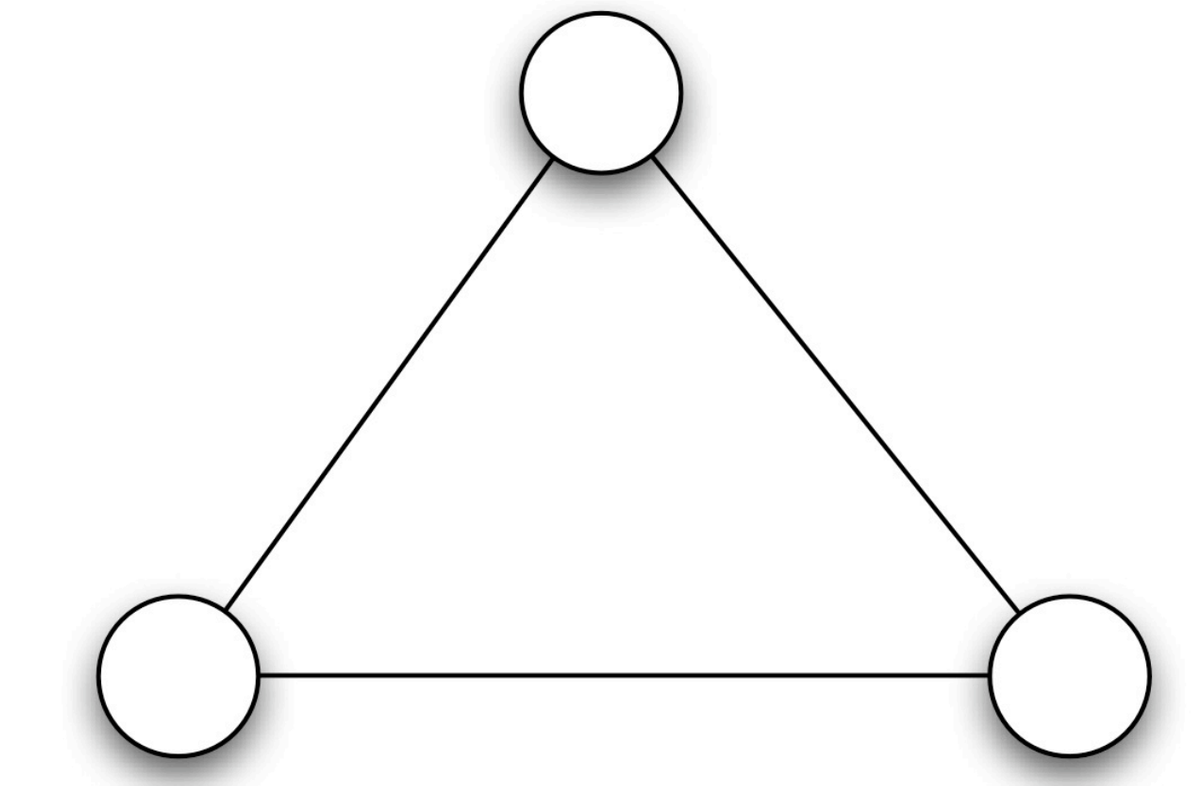






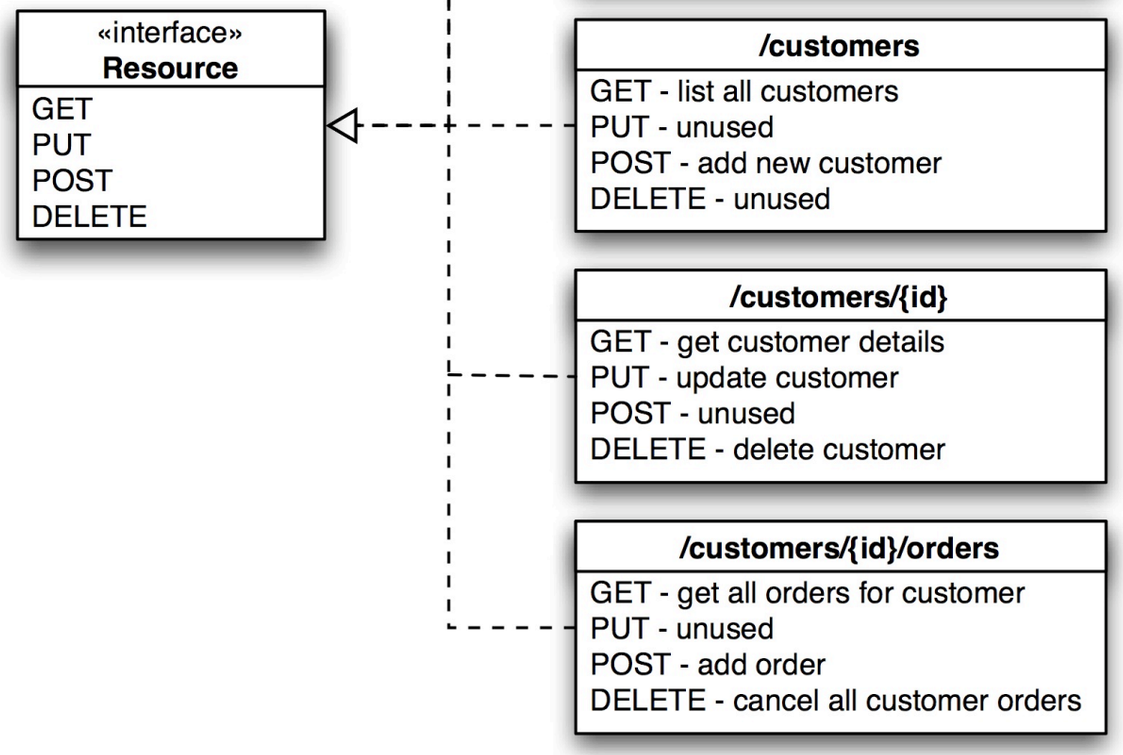


many



very few
(fixed)

many



Designing a RESTful Application

Identify resources & design URIs

Select formats (or create new ones)

Identify method semantics

Select response codes

See: http://bitworking.org/news/How_to_create_a_REST_Protocol

Some HTTP Features

Verbs (in order of popularity):

GET, POST

- PUT, DELETE
- HEAD, OPTIONS, TRACE

Standardized (& meaningful) response codes

Content negotiation

Redirection

Caching (incl. validation/expiry)

Compression

Chunking

RESTful HTTP Advantages

Universal support (programming languages, operating systems, servers, ...)

Proven scalability

Real web integration for machine-2-machine communication

Support for XML, but also other formats

Asynchronous Communication

HTTP is always synchronous request/ response

For lengthy interactions, the server should return `202 Accepted` and a URI for the result

Poll or pass a URI to be notified

WS-Addressing is just URIs used badly

Reliable Messaging w/ HTTP

First question: What does “reliable” mean?

Often solved at the application level

Existing proposals:

- Bill de hÓra’s HTTPLR
- Yaron Goland’s SOA-Reliability (“SOA-Rity”)
- Mark Nottingham’s POE (Post Once Exactly)

Distributed Transactions

Distributed Transactions

Distributed Transactions

The Holy Grail of Distributed Computing

Distributed Transactions

In practice, not used as often as you think

2PC and loose coupling don’t work together very well

Compensating transactions are business logic anyway

A light-weight protocol could be created, but no one has cared so far

UDDI

420-page specification

Finding and maintaining (meta-)model objects

Inquiry

- find_binding
- find_business
- find_relatedBusinesses
- find_service
- find_tModel
- get_bindingDetail
- get_businessDetail
- get_operationalInfo
- get_serviceDetail
- get_tModelDetail

Publication

- save_binding
- save_business
- save_service
- save_tModel
- delete_binding
- delete_business
- delete_publisherAssertions
- delete_service
- delete_tModel
- add_publisherAssertions
- set_publisherAssertions
- get_assertionStatusReport
- get_publisherAssertions
- get_registeredInfo

UDDI *(contd.)*

UDDI could be greatly simplified by using plain HTTP

It would no longer be protocol-independent

- Who cares?

Atom (Syndication Format & Protocol) would be a great match

See: http://www.xml.com/pub/a/ws/2002/02/06/rest.html?page=2

UDDI *(contd.)*

UDDI could be greatly simplified by using plain HTTP

It would no longer be protocol-independent

- Who cares?

Atom (Syndication Format & Protocol) would be a great match

See: http://www.xml.com/pub/a/ws/2002/02/06/rest.html?page=2

REST-based WS-* Registries

MuleSource - Galaxy
http://www.mulesource.com/products/galaxy.php

WSO2 - Registry
http://wso2.org/projects/registry

Attachments

Attachments in WS-*

SOAP with Attachments (MIME)

DIME (supported by Microsoft)

XOP/MTOM

Attachments in REST

<a href="..." />

<link rel="..." href="..." />

alternatively: use a different representation

Security

Web services security is message-based

HTTP relies on

- transport level security (SSL/TLS)
- basic and digest authentication
- access control based on resources and methods

WSS concepts would be a great value-add for HTTP-based systems

Description

What's the WSDL equivalent in REST?

There is none ...

XSD (95% of WSDL) is available to you, anyway

Of the remaining 5%, 90% is just silly

Why would you want to describe the uniform interface over and over again?

... unless you insist

WADL (Web Application Description Language)
https://wadl.dev.java.net/

Use URI Templates to define resource behavior

WADL Example

```
<resources base="http://api.search.yahoo.com/NewsSearchService/V1/">
    <resource path="newsSearch">
        <method name="GET" id="search">
            <request>
                <param name="appid" type="xsd:string" style="query" required="true"/>
                <param name="query" type="xsd:string" style="query" required="true"/>
                <param name="type" style="query" default="all">
                    <option value="all"/>
                    <option value="any"/>
                    <option value="phrase"/>
                </param>
                <param name="results" style="query" type="xsd:int" default="10"/>
                <param name="start" style="query" type="xsd:int" default="1"/>
                <param name="sort" style="query" default="rank">
                    <option value="rank"/>
                    <option value="date"/>
                </param>
                <param name="language" style="query" type="xsd:string"/>
            </request>
            <response>
                <representation mediaType="application/xml" element="yn:ResultSet"/>
                <fault status="400" mediaType="application/xml" element="ya:Error"/>
            </response>
        </method>
    </resource>
</resources>
```

What You Should Do

What You Should Do

(in my very humble opinion)

Be skeptical of WS-*

Be skeptical of WS-*
Learn more about REST

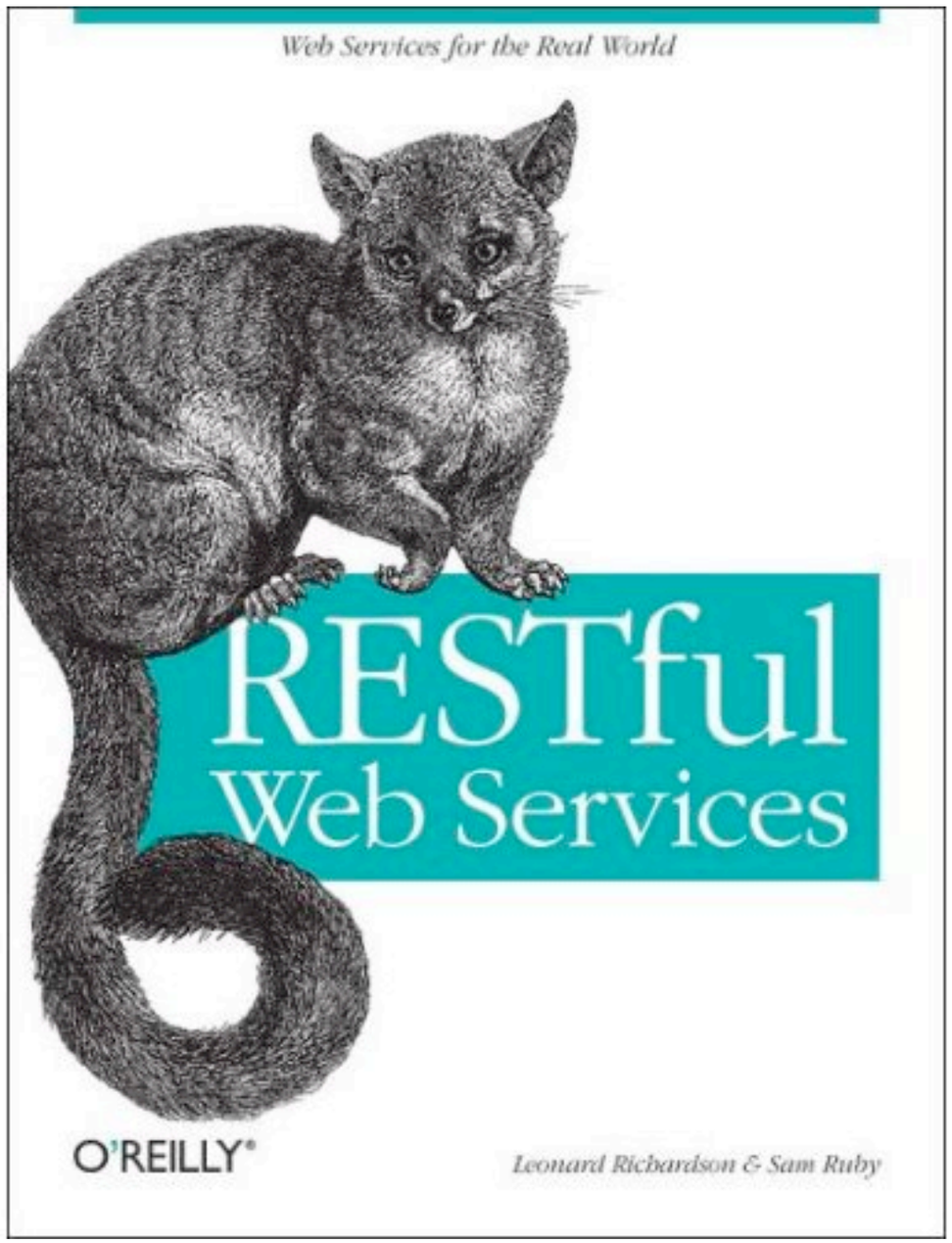
Be skeptical of WS-*
Learn more about REST
Learn to love the URI

Be skeptical of WS-*
Learn more about REST
Learn to love the URI

Appreciate the Web

If You Want to Know More

http://www.innoq.com/resources/REST
http://railsconsulting.de



http://www.oreilly.com/catalog/9780596529260/

http://www.infoq.com/

Random Stuff | innoQ | InfoQ | SOA-Expertenwissen | Txp › SOA-E...ertenwissen | innoQ | Partner | banking | awstats | js | 1&1 | O1BC.DE | AR | RailsDocs

InfoQ

Tracking change and innovation in the enterprise software development community

Version 1.4 | En | 中文 | 日本語

174,255 Sep unique visitors

Welcome, Stefan!

Sign out
Preferences
About us
Personal feed
Home

Your Communities

- [x] Java
- [ ] .NET
- [x] Ruby
- [x] SOA
- [ ] Agile
- [x] Architecture

Search

Featured Topics

Performance & Scalability

SOA Governance

Public Beta
Now Available

New & Notable Written for InfoQ by the Community

Contribute News

DynamicJasper: Runtime generation of Jasper Reports

Community Java Topics Open Source

DynamicJasper, an open-source API which provides runtime generation of Jasper Reports, recently released version 1.3. InfoQ took the opportunity to learn more about this product, and what it provides for users. By Ryan Slobojan on Oct 08 Discuss

Presentation: Architecture Evaluation in Practice

Community Architecture Topics Delivering Quality, Enterprise Architecture

Dragos Manolescu shares insights gained from growing ThoughtWorks' architecture evaluation practice and evaluating several architectures for Global 1000 companies. These insights aim at preparing people interested in commiss… participating in, or an… evaluation to tackle th… Marinescu on Oct 08 D…

Ruby and the hype cycle

Community Ruby Topics Performance & Scalability, Ruby on Rails, Stories & Case Studies

A recent blog post on a failed Rails project caused a big debate about the viability of Ruby on Rails. A closer look at the post paints a different picture, though. We take a look at the reactions in the Ruby community, and compare this discussion with the upheaval about Twitter earlier this year. By Werner Schuster on Oct 08 3 comments

Adobe Max 2007 North America - Wrap Up

Community Java Topics Rich Client / Desktop, Acquisitions, Rich Internet Apps

Adobe was busy this week showing off their latest work at the 2007 Max Conference. Adobe continues to cater to developers with many of their efforts. The conference came with a number of interesting and exciting announcements for the developer community including: By Jon Rose on Oct 05 Discuss

Exclusive Content

All | Articles | Presentations | Interviews | Books

Steve Sloan on BizTalk Server 2006 R2

InfoQ talked to Steve Sloan, Senior Product Manager, about the BizTalk Server 2006 R2 in the context of SOA. SOA, Oct 04, 2007,

Open Source WS Stacks for Java - Design Goals and Philosophy

InfoQ spoke to the lead developers of the most important open source Java Web-services stacks about their design goals, standards, data binding, XML, interoperability, REST support, and maturity. SOA, Java, Oct 04, 2007, 7

Creating dynamic web applications with JSF/DWR/DOJO

JSF, DWR, and Dojo are all popular technologies in their own right. This article looks at how they can be integrated together in a portal environment. Java, Oct 04, 2007, 1

Architecture Evaluation in

Recommendations

Learn more about REST

Learn more about REST

Learn to love the URI

Learn more about REST

Learn to love the URI

Adopt REST principles for UIs ...

Learn more about REST

Learn to love the URI

Adopt REST principles for UIs ...

...APIs will follow on their own

Learn more about REST

Learn to love the URI

Adopt REST principles for UIs ...

...APIs will follow on their own

Appreciate the Web

Thank you!
Any questions?

http://www.innoq.com
http://railsconsulting.de

Stefan Tilkov

http://www.innoq.com/blog/st/

innoQ

Architectural Consulting

SOA	WS-*	REST
MDA	MDSD	MDE
J(2)EE	RoR	.NET

innoQ Deutschland GmbH
Halskestraße 17
D-40880 Ratingen
Phone +49 2102 77162-100
info@innoq.com · www.innoq.com

innoQ Schweiz GmbH
Gewerbestrasse 11
CH-6330 Cham
Phone +41 41 743 01 11

Photo Credit

http://en.wikipedia.org/wiki/Image:Sangreal.jpg