Lightweight SOA

Stefan Tilkov / innoQ / stefan.tilkov@innoq.com

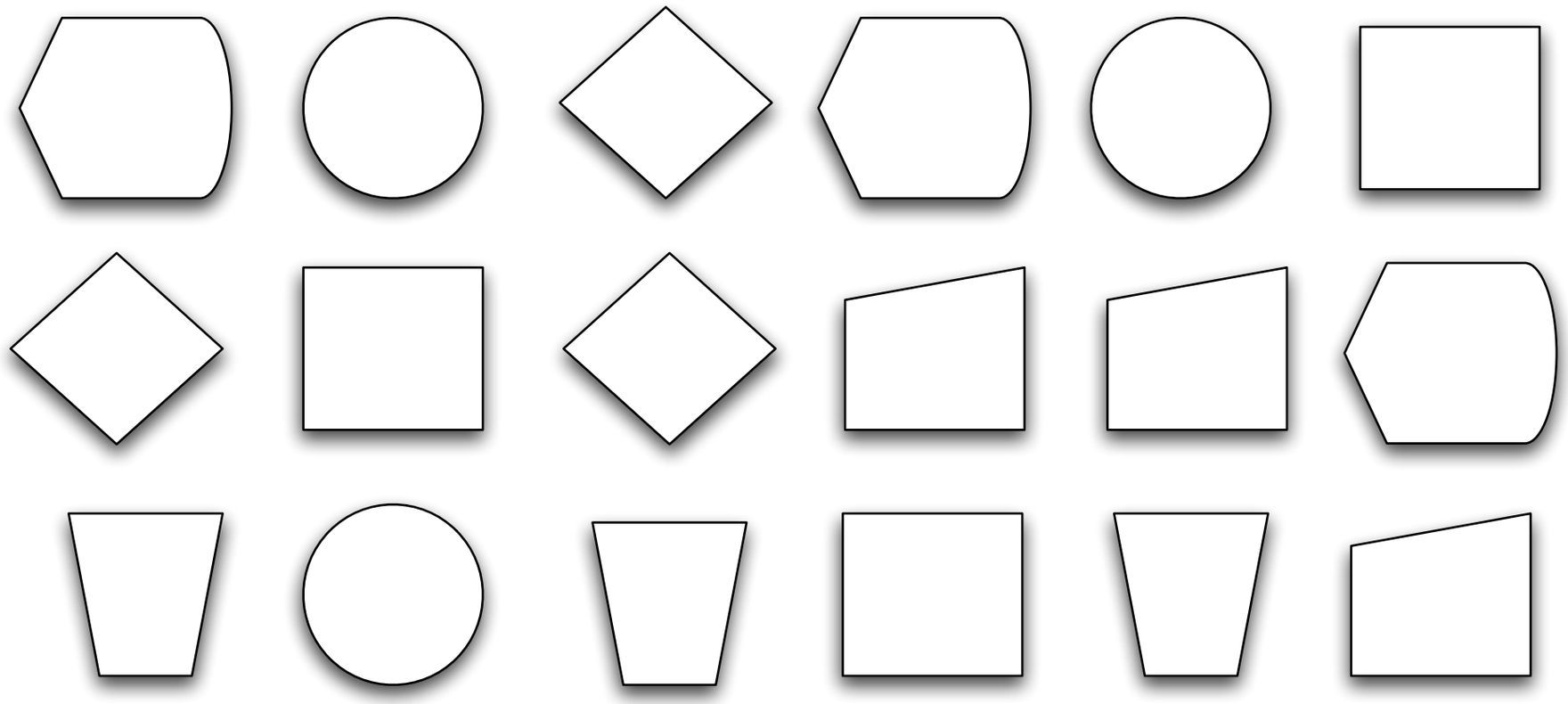
9 Claims

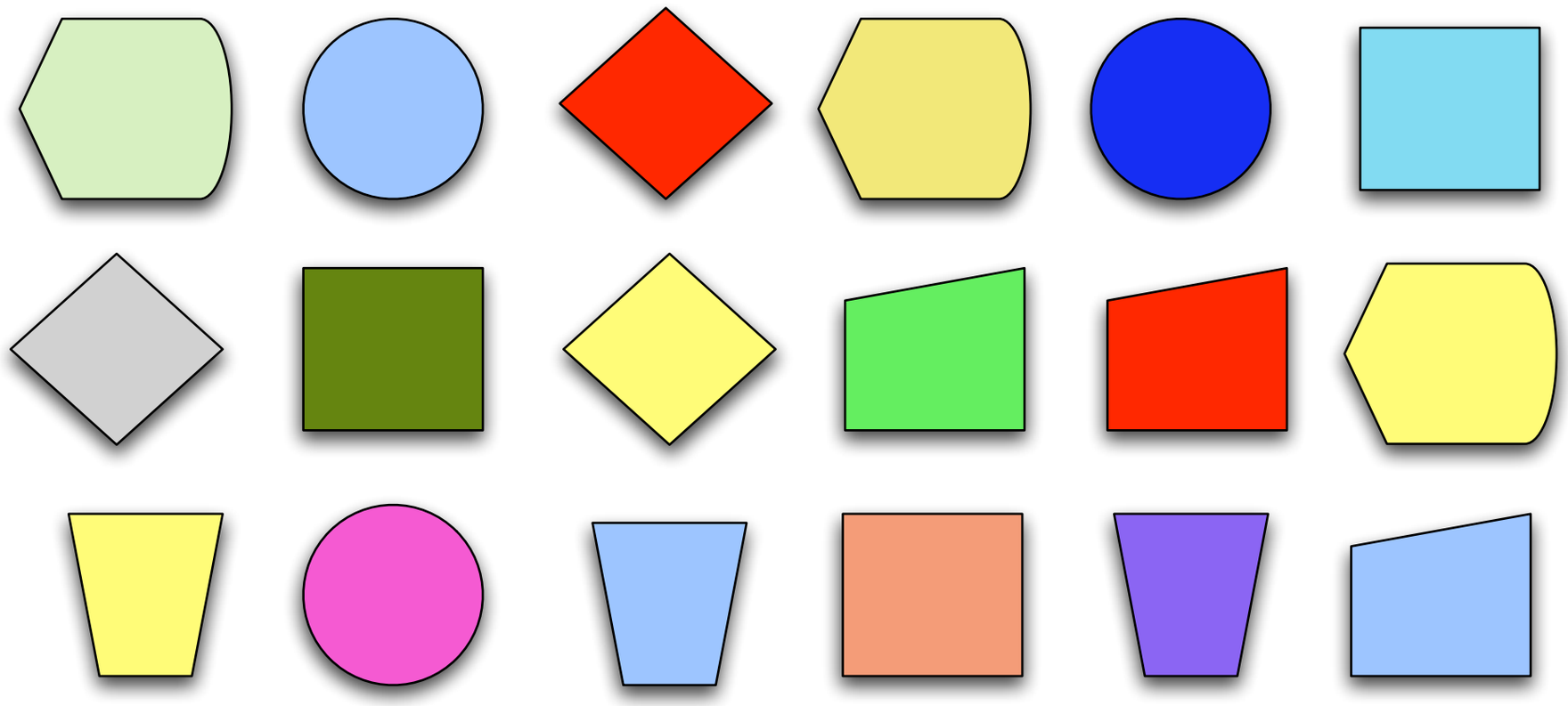
Claim #1/9: Application architecture is irrelevant for your SOA

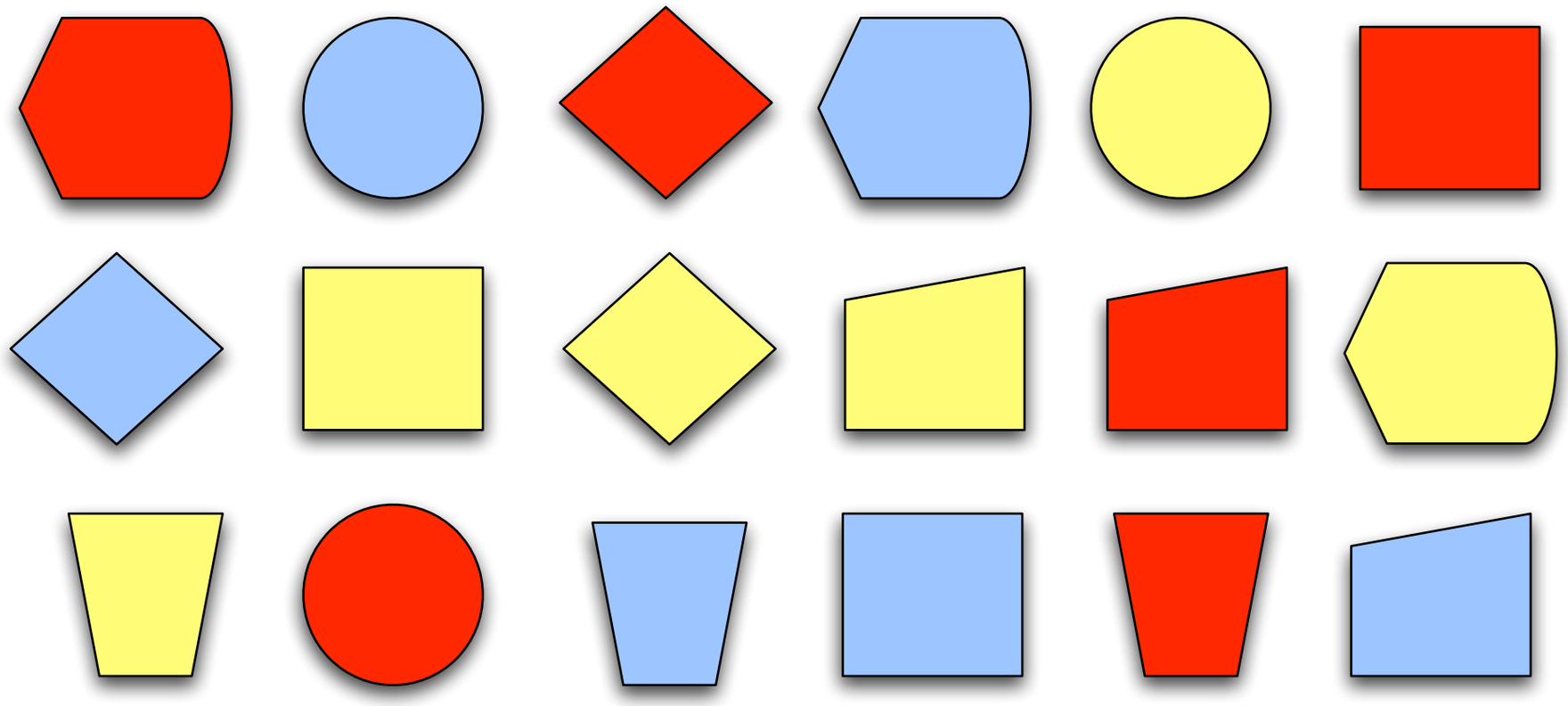
Application Architecture

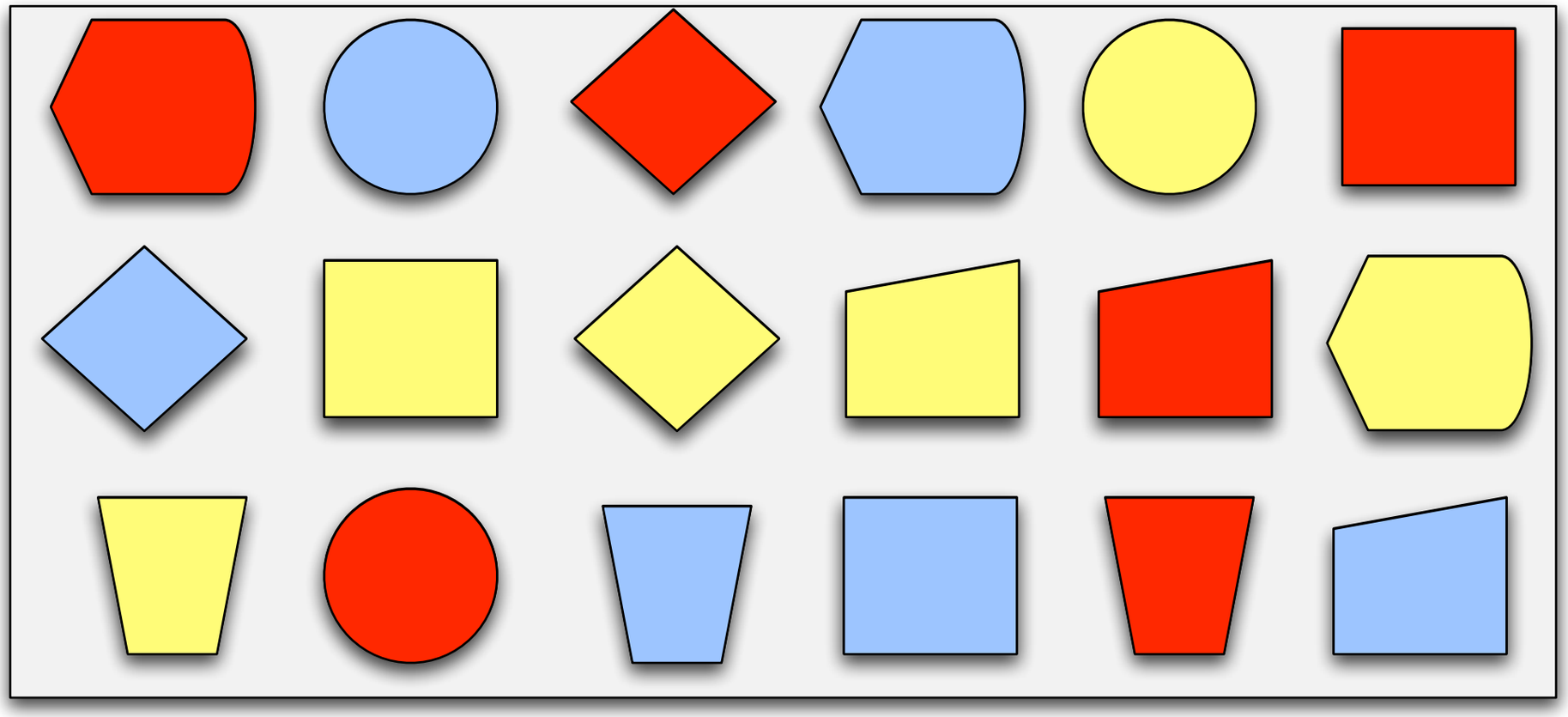
vs.

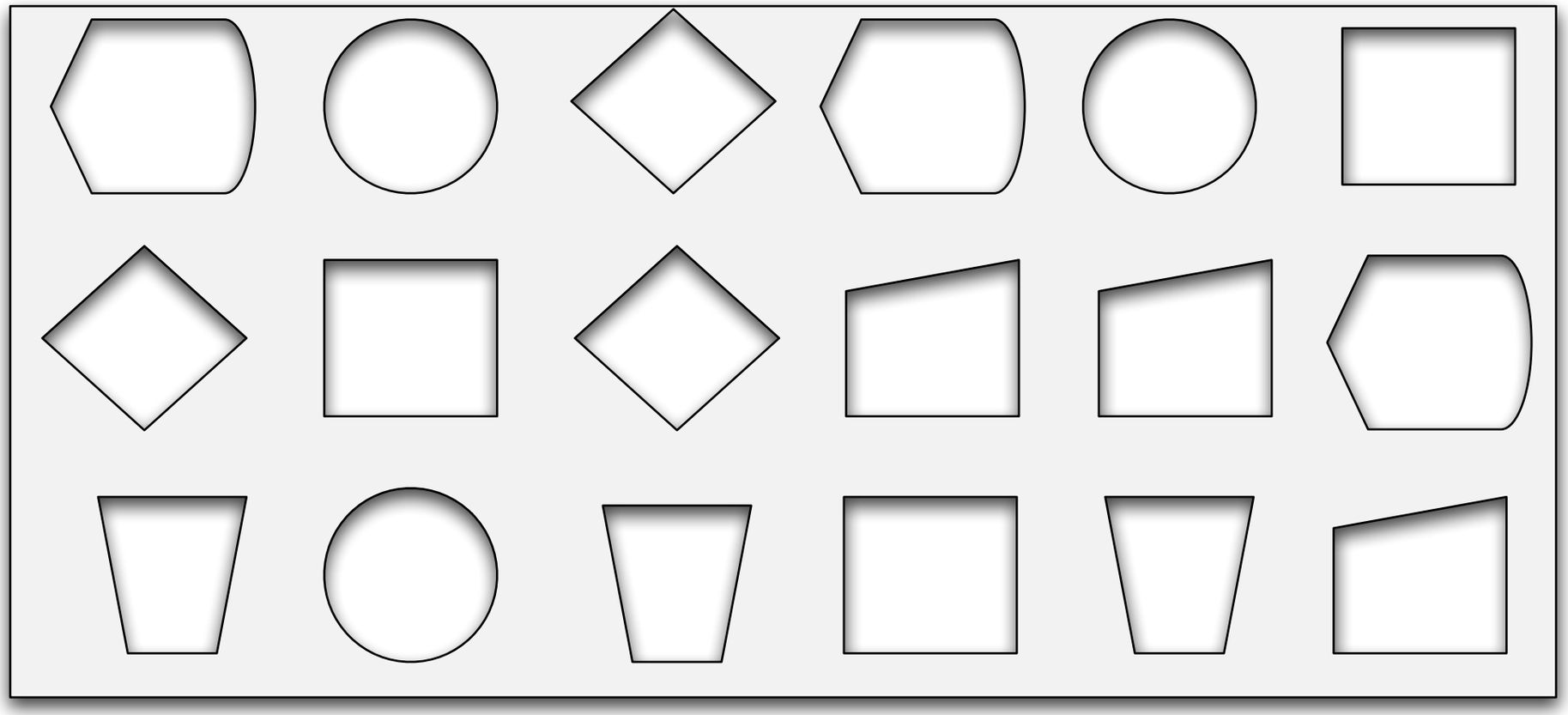
Integration Architecture

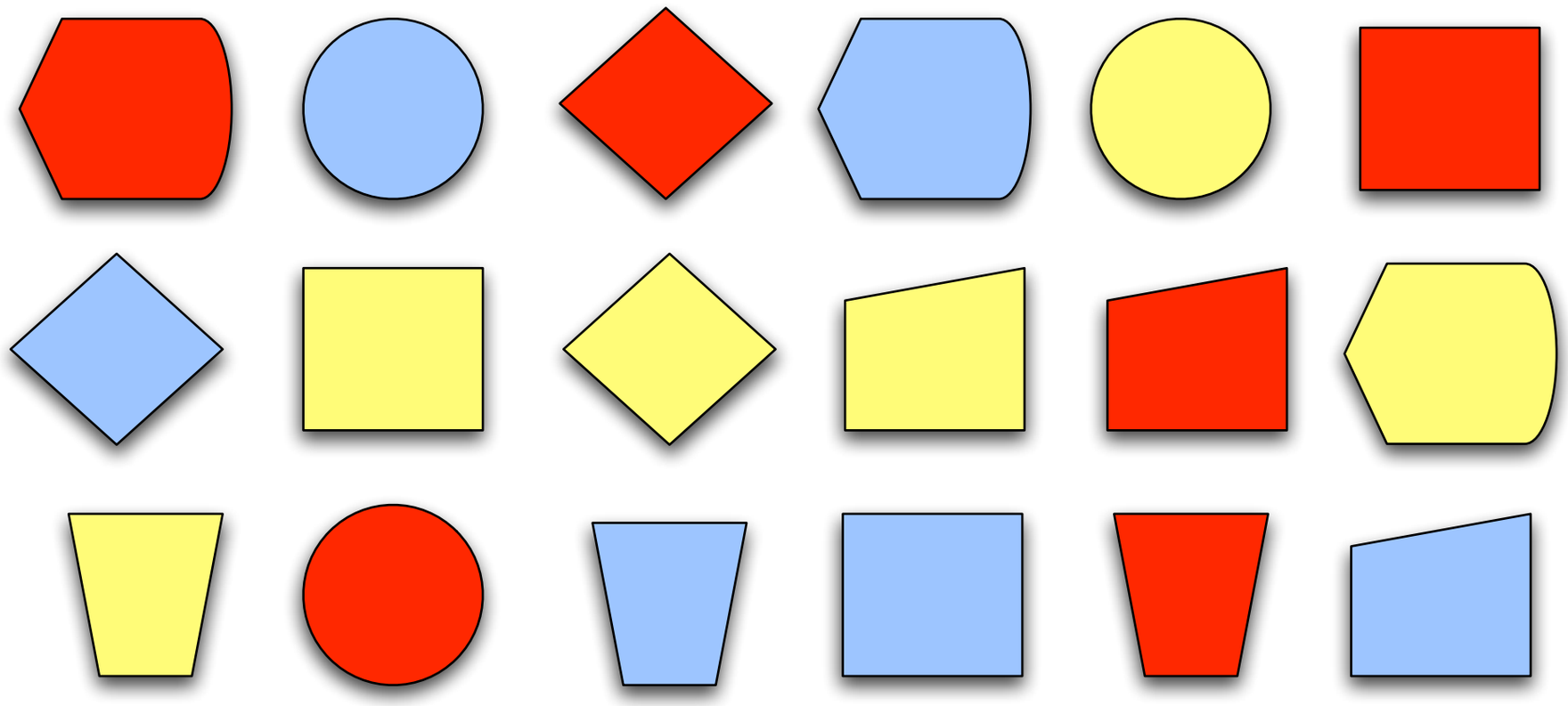


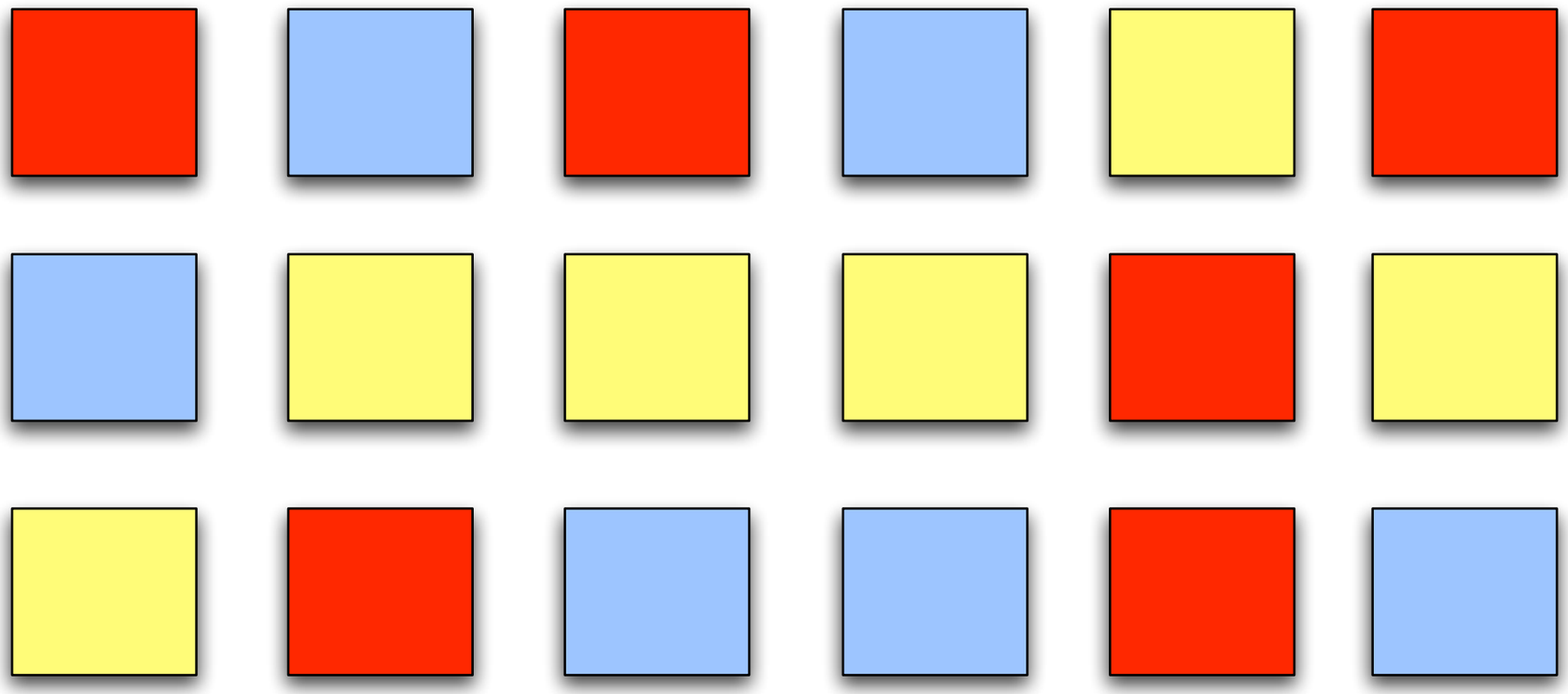


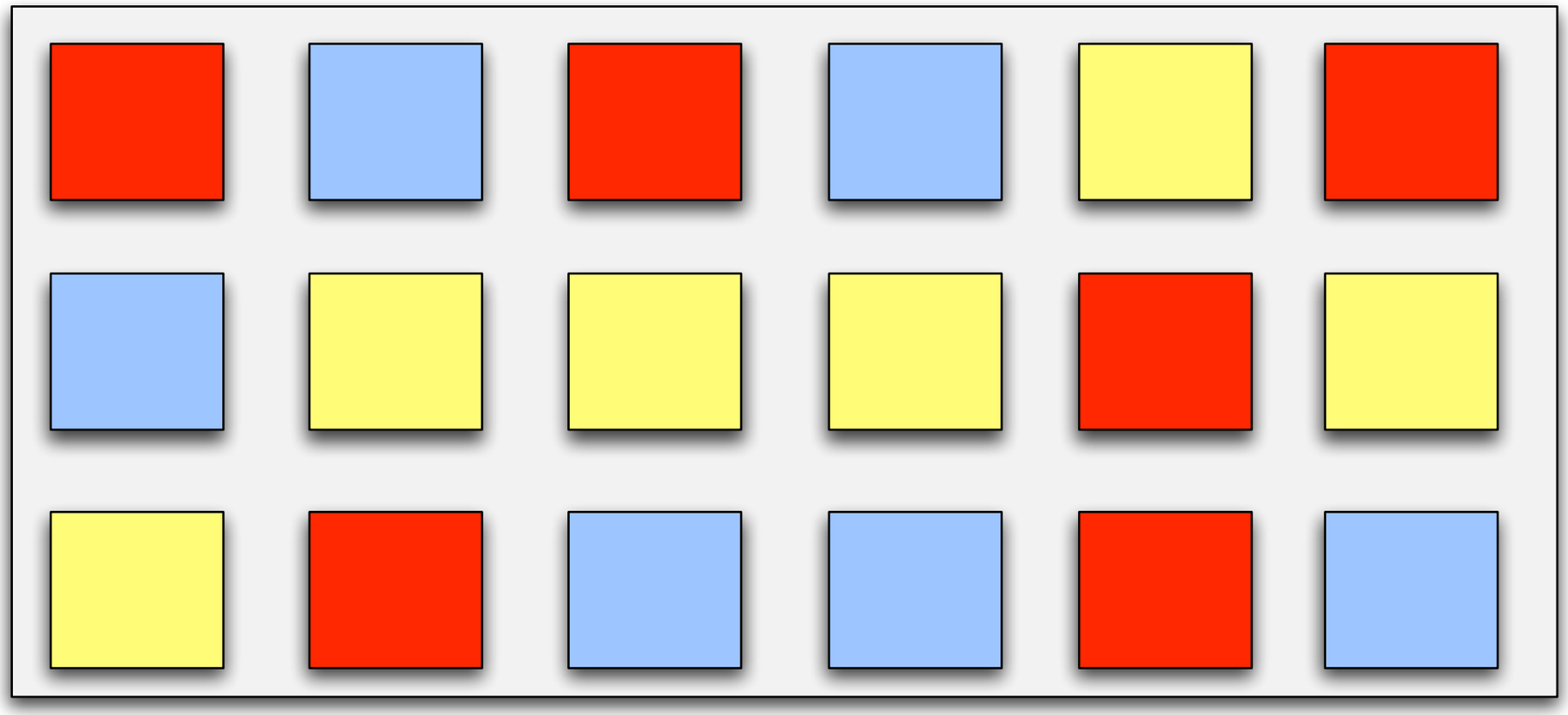


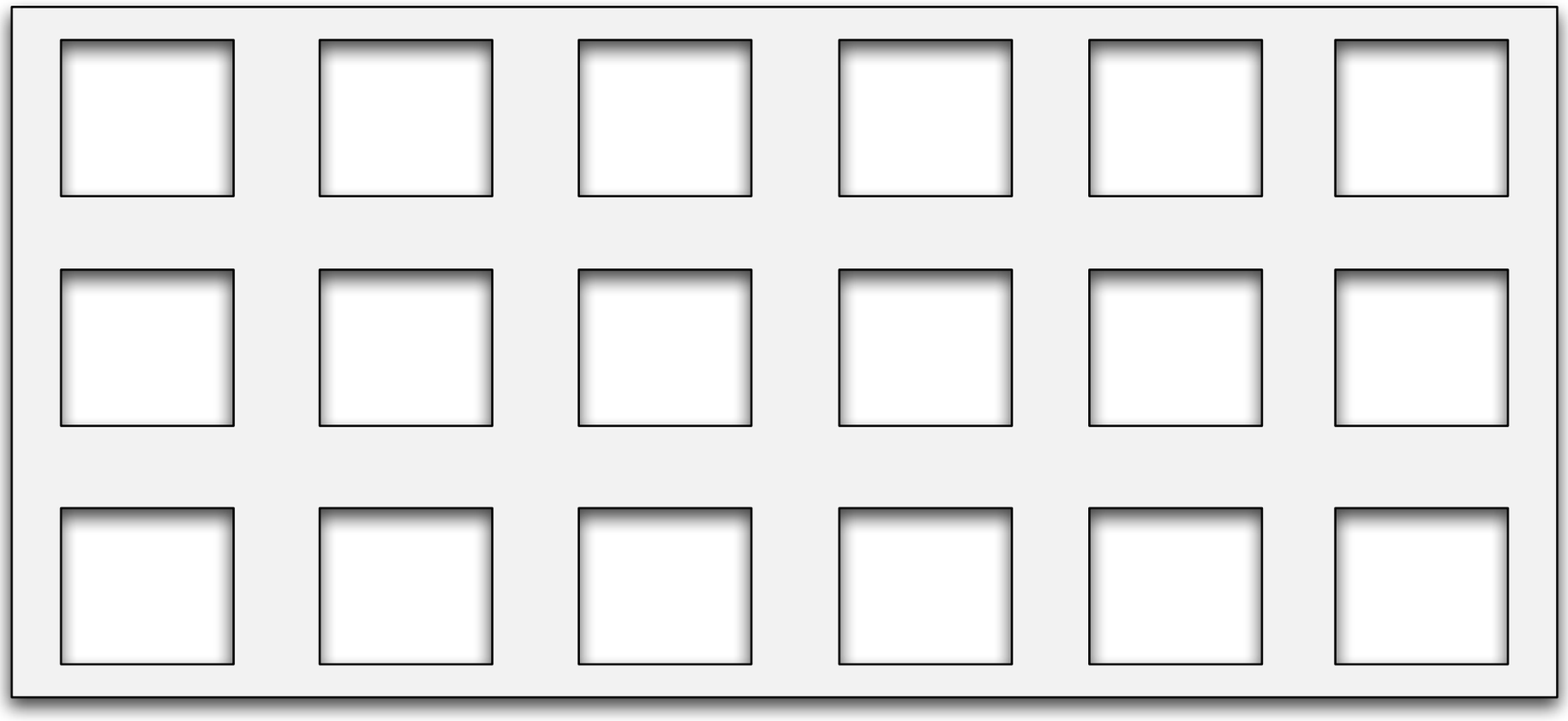




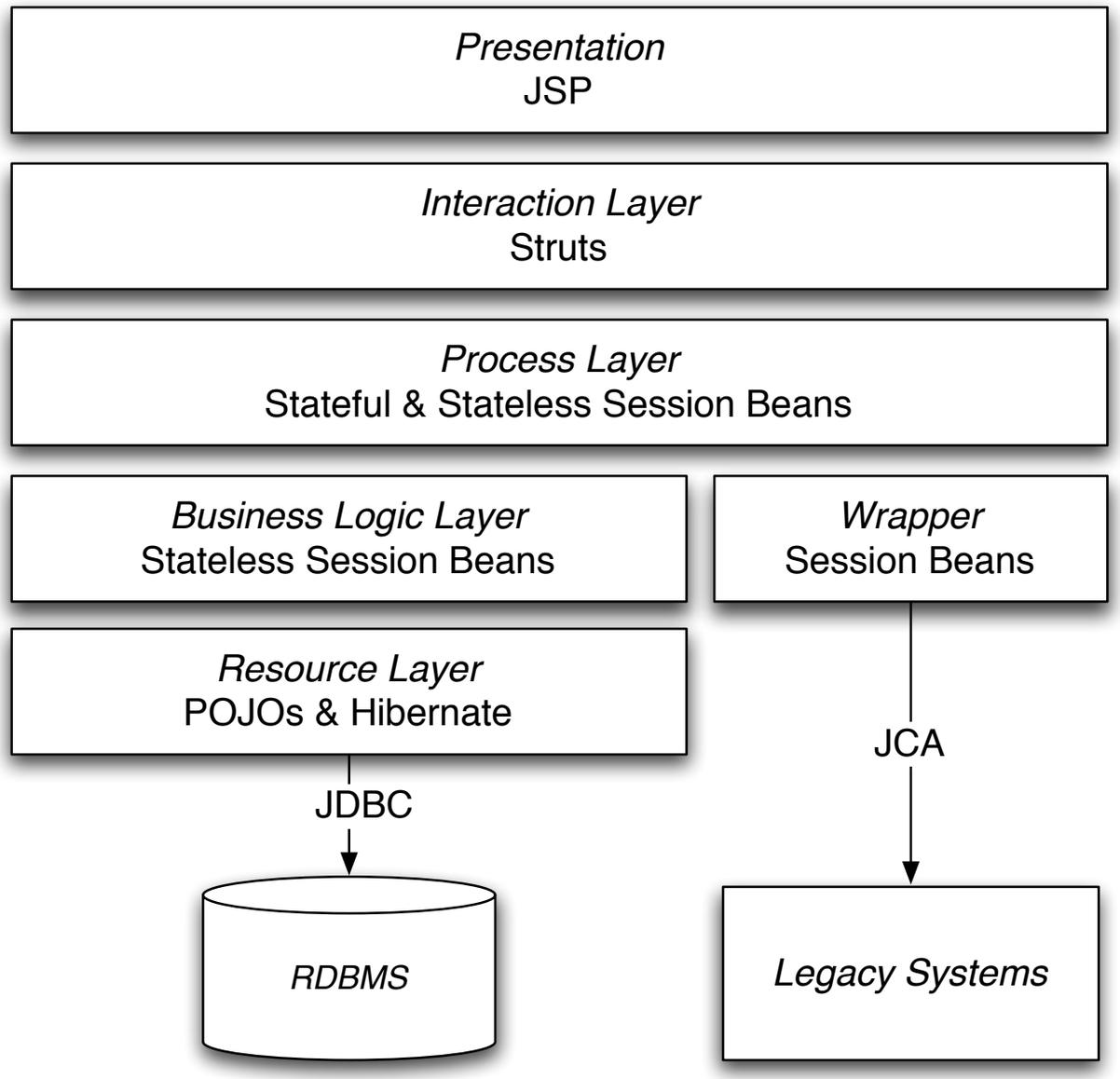






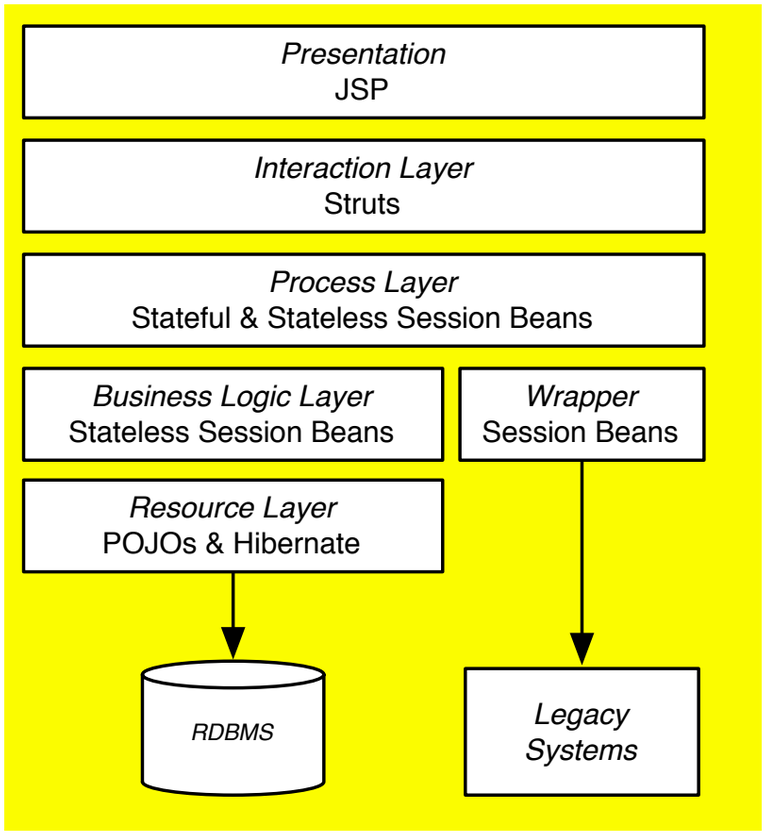


Application Architecture
(Example)

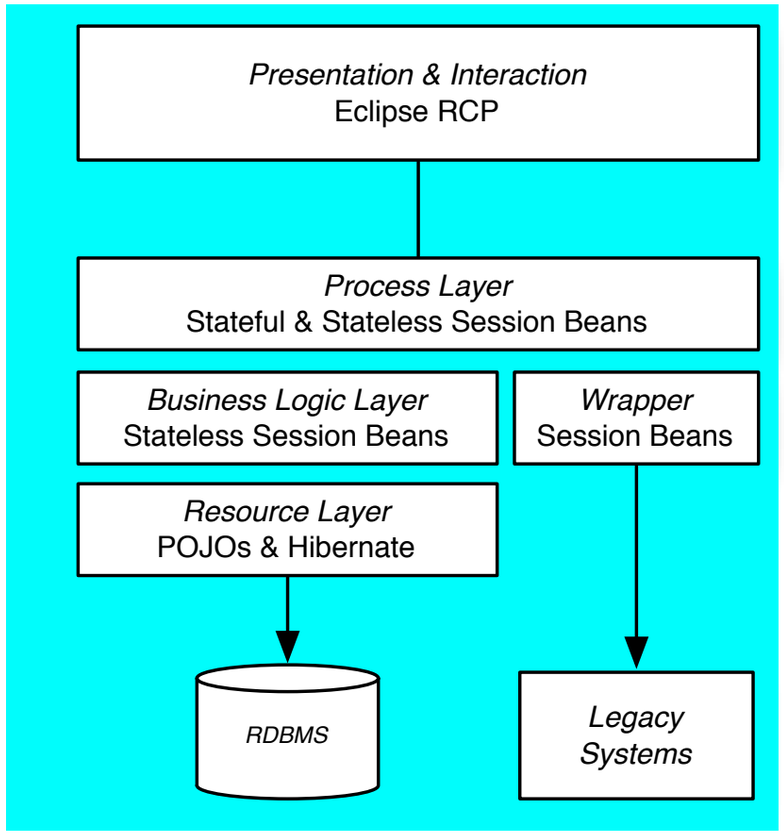


Application Classes

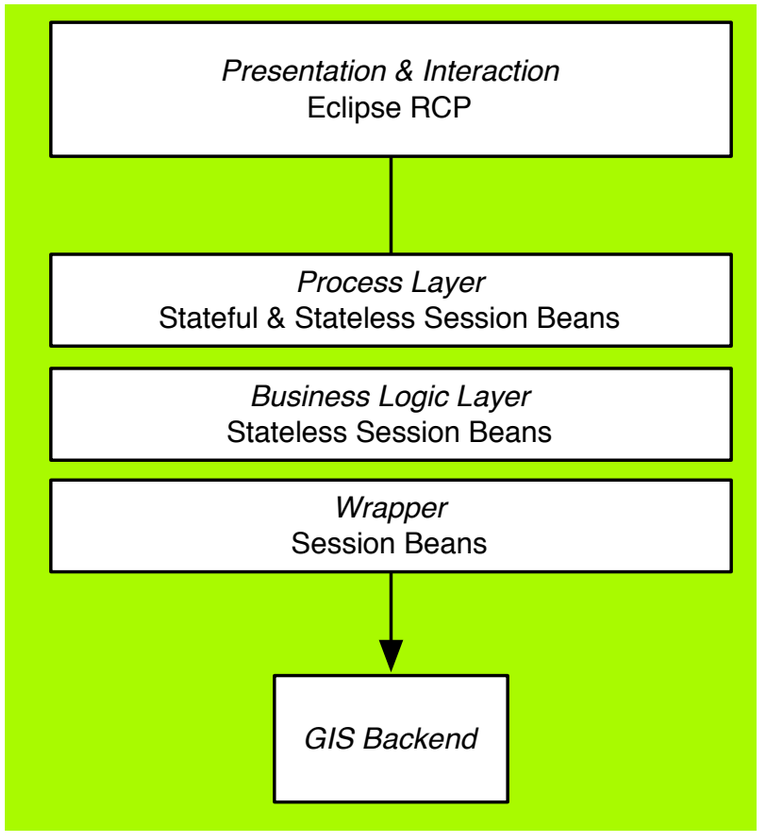
Class A



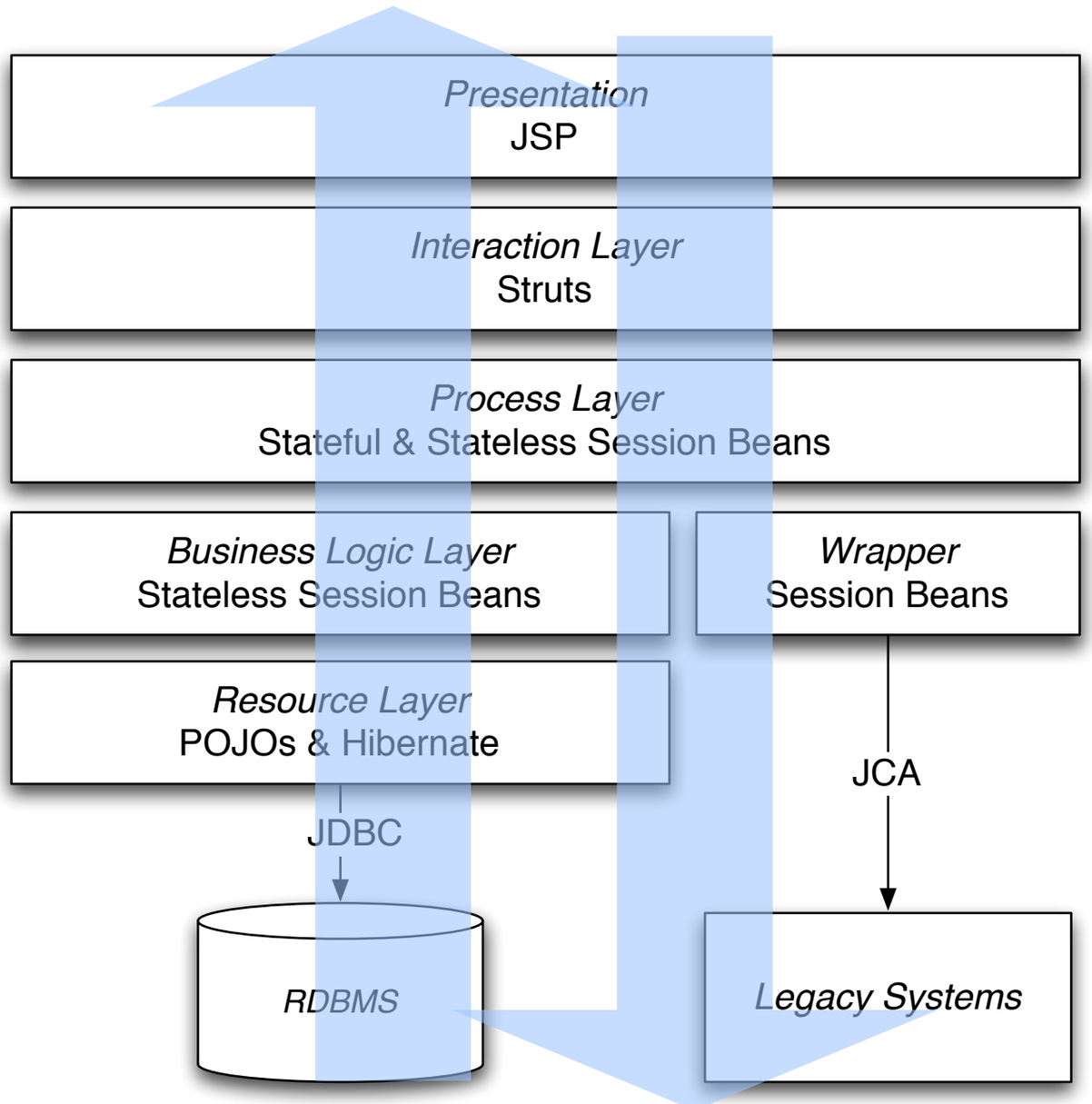
Class B



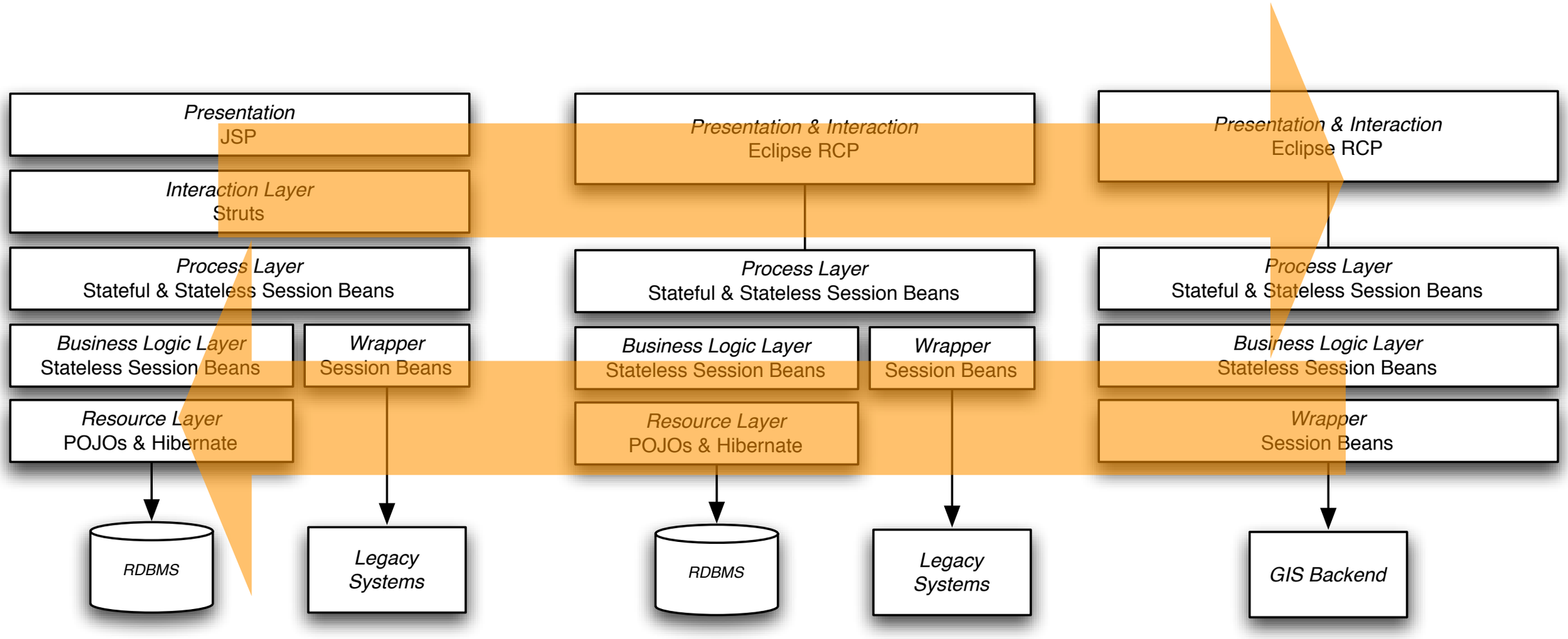
Class C



Focus: Portability



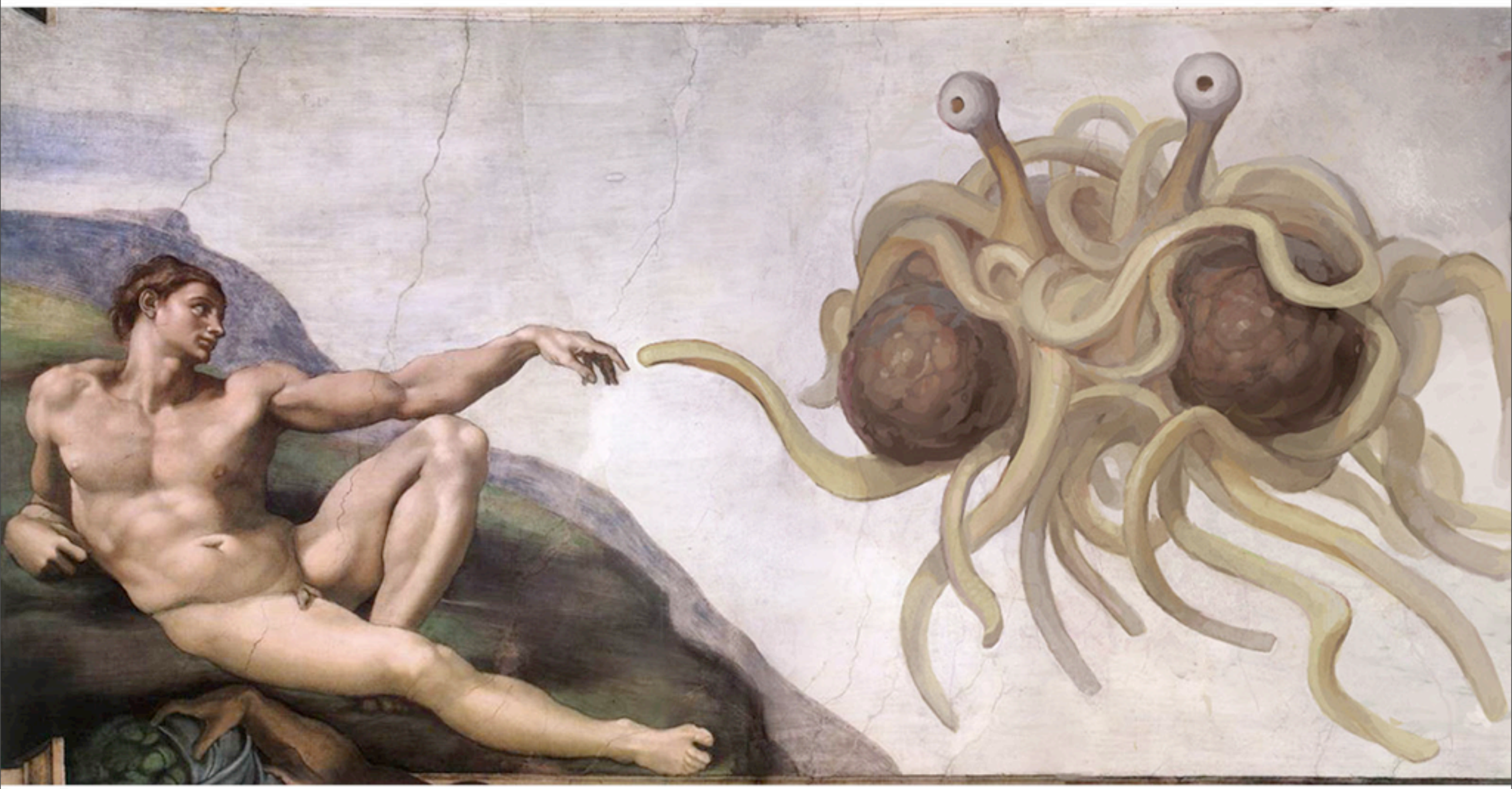
Focus: Interoperability



Claim #2/9:
An ESB should not be at the core of your SOA

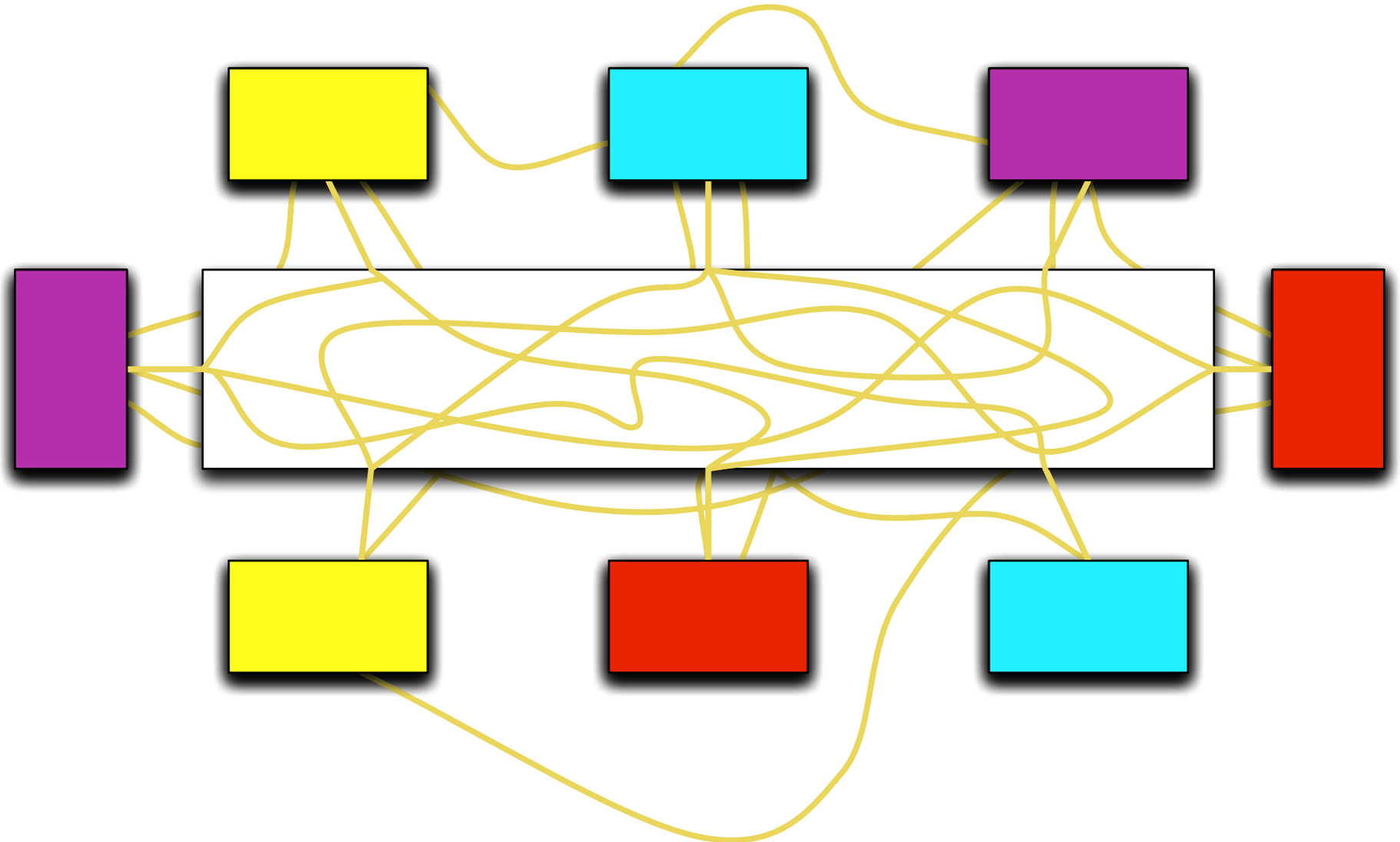
ESB Spaghetti Monster

THE FLYING SPAGHETTI MONSTER



TOUCHED BY HIS NOODLY APPENDAGE

ESB Spaghetti



Shamelessly stolen from Jim Webber: http://tinyurl.com/ywm5sj

EOA

ESB-oriented Architecture

Proprietary tooling (“doodleware”)

No (or little) team development

No unit-testing, test-driven development

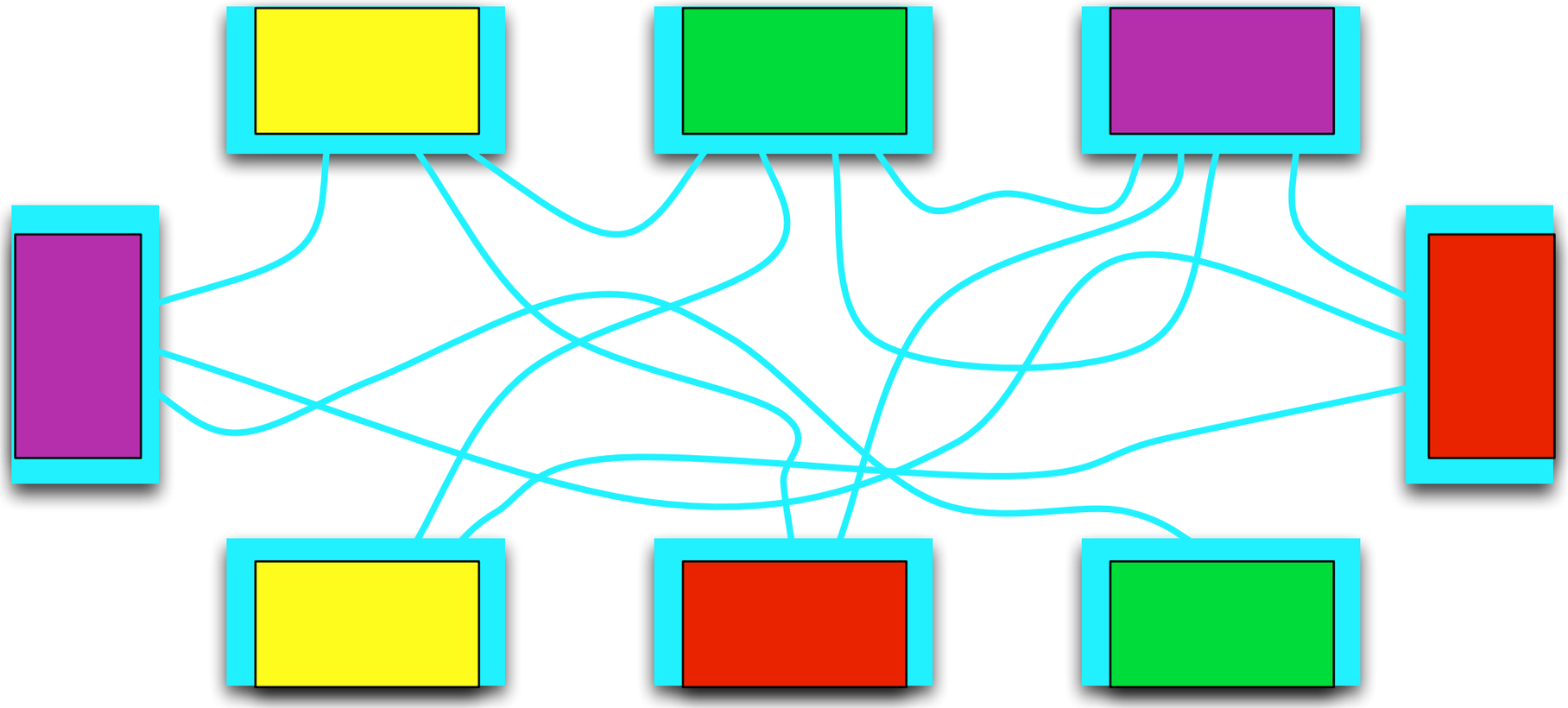
No (or little) modularization

No (or little) versioning

Barrier to entry

Prolonged existence of proprietary interfaces

“Virtual” ESB



Claim #3/9: Point-to-point communication is perfectly fine

Mainstream + Lightweight

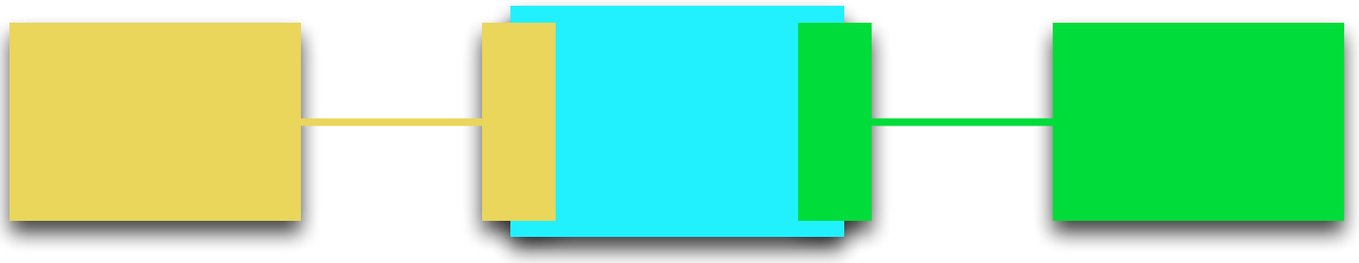
Pick a decent platform or toolkit for SOAP/WS-* support

.NET WCF, Spring Web Services, Apache CXF, Apache Axis2, JBossWS, Sun Metro

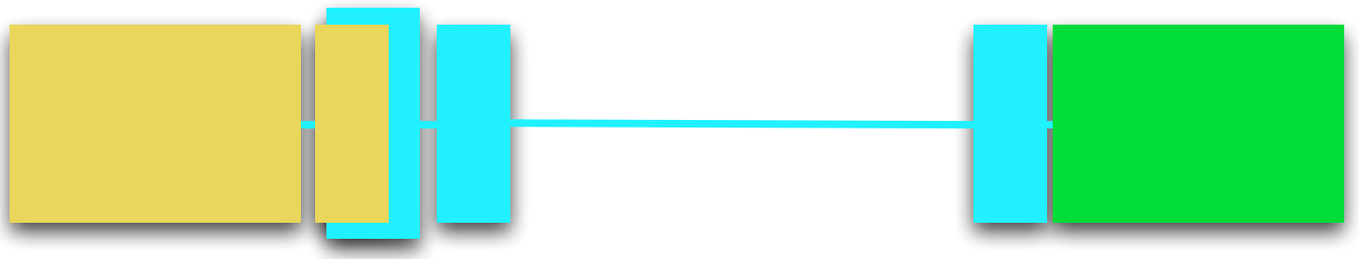
Standardize on standards, not products, not single points of failure

If you Use an ESB ...

Central Model



Enabling Model



... know what you're doing

Consider Open Source ESBs

Apache Synapse/WSO2 ESB

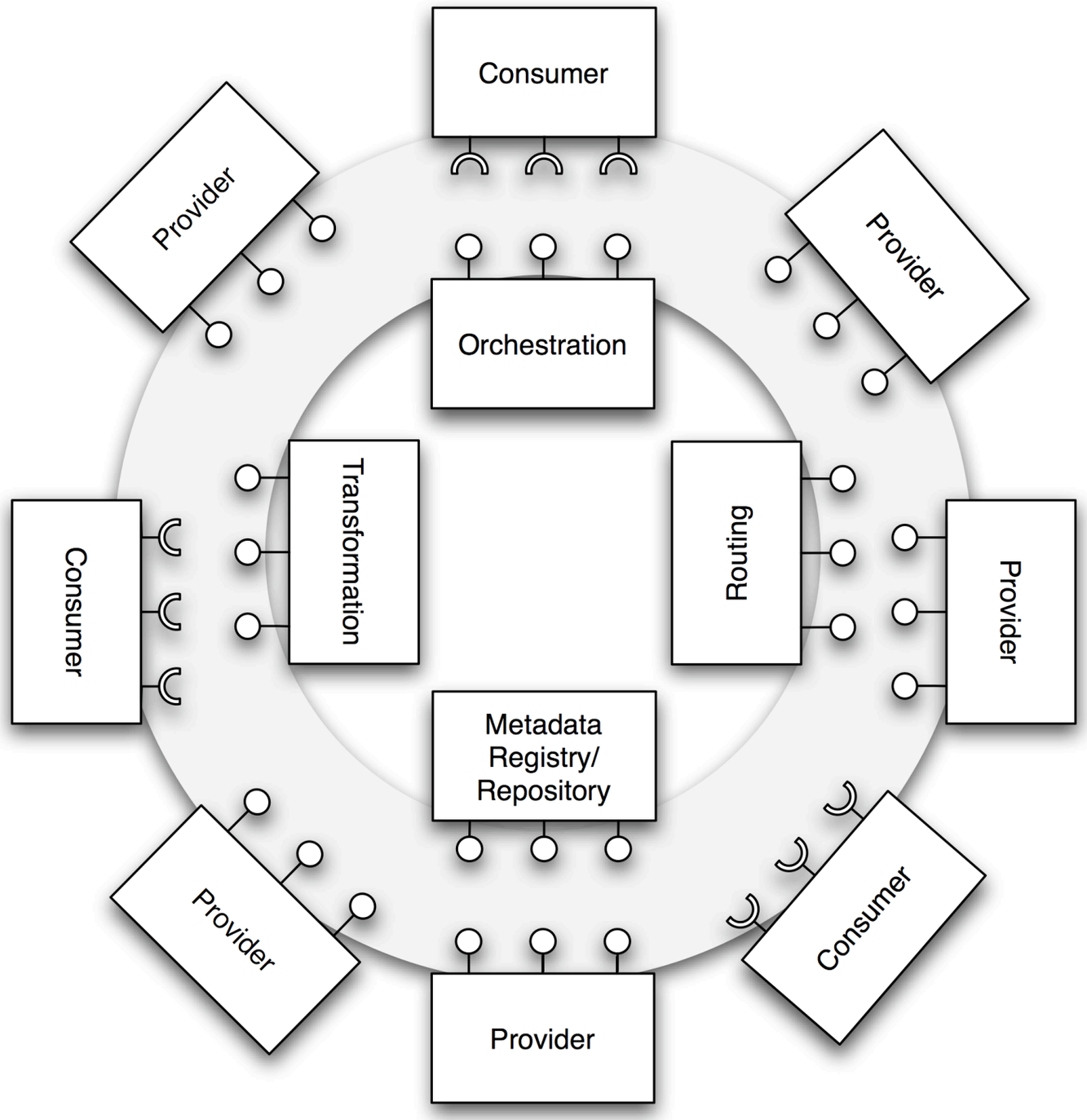
Mule ESB

ServiceMix

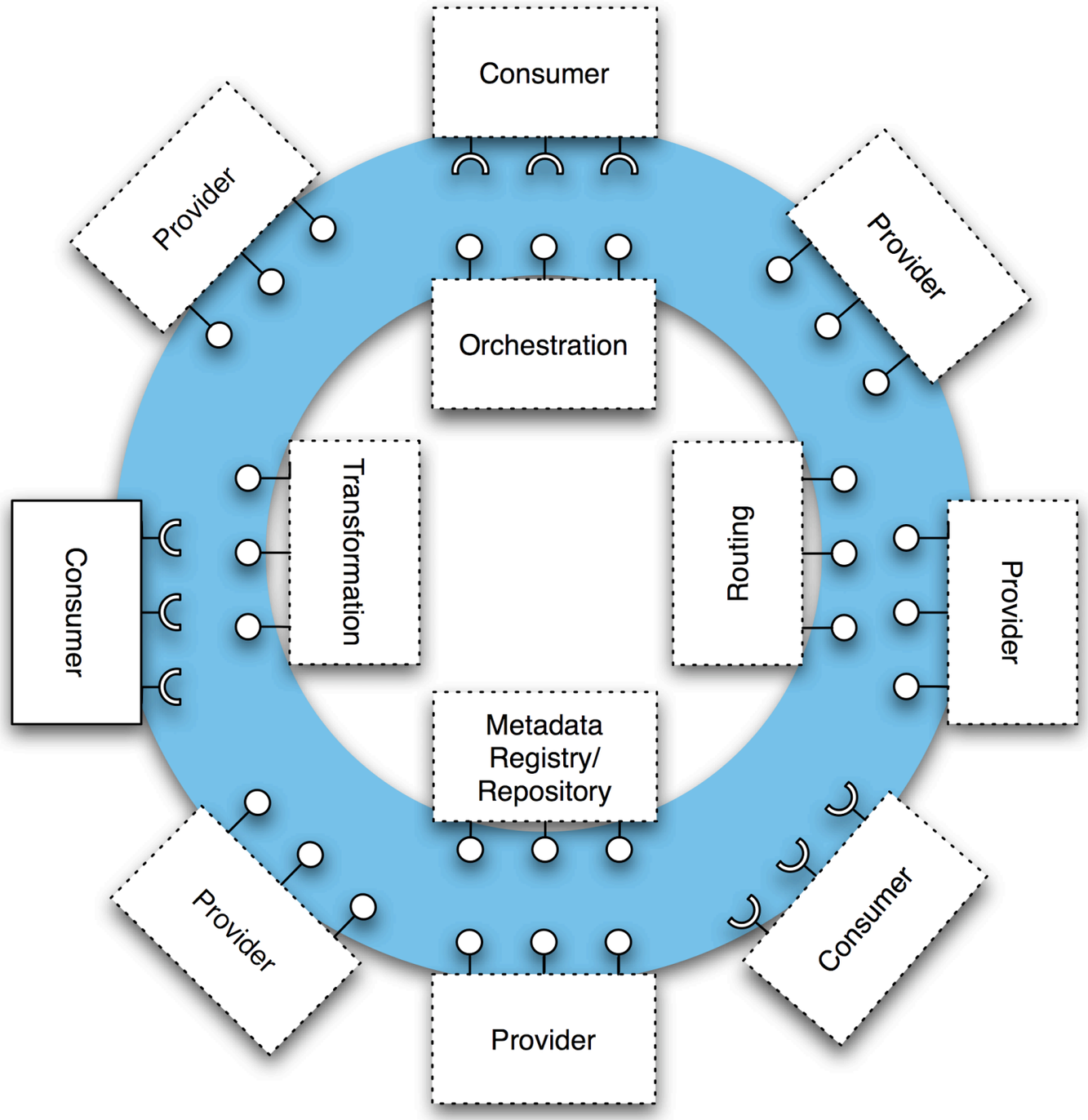
Sopera

Claim #4/9:
Participants in a given SOA must depend on wire formats only

“Virtual” ESB


Consumer
Provider
Provider
Orchestration
Transformation
Routing
Consumer
Provider
Metadata
Registry/
Repository
Provider
Provider
Consumer

“Virtual” ESB


Consumer
Provider
Provider
Orchestration
Transformation
Routing
Consumer
Provider
Metadata
Registry/
Repository
Provider
Provider
Consumer

Claim #5/9:

Web services (SOAP/WSDL) are not the only way

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. Offer Multiple Formats

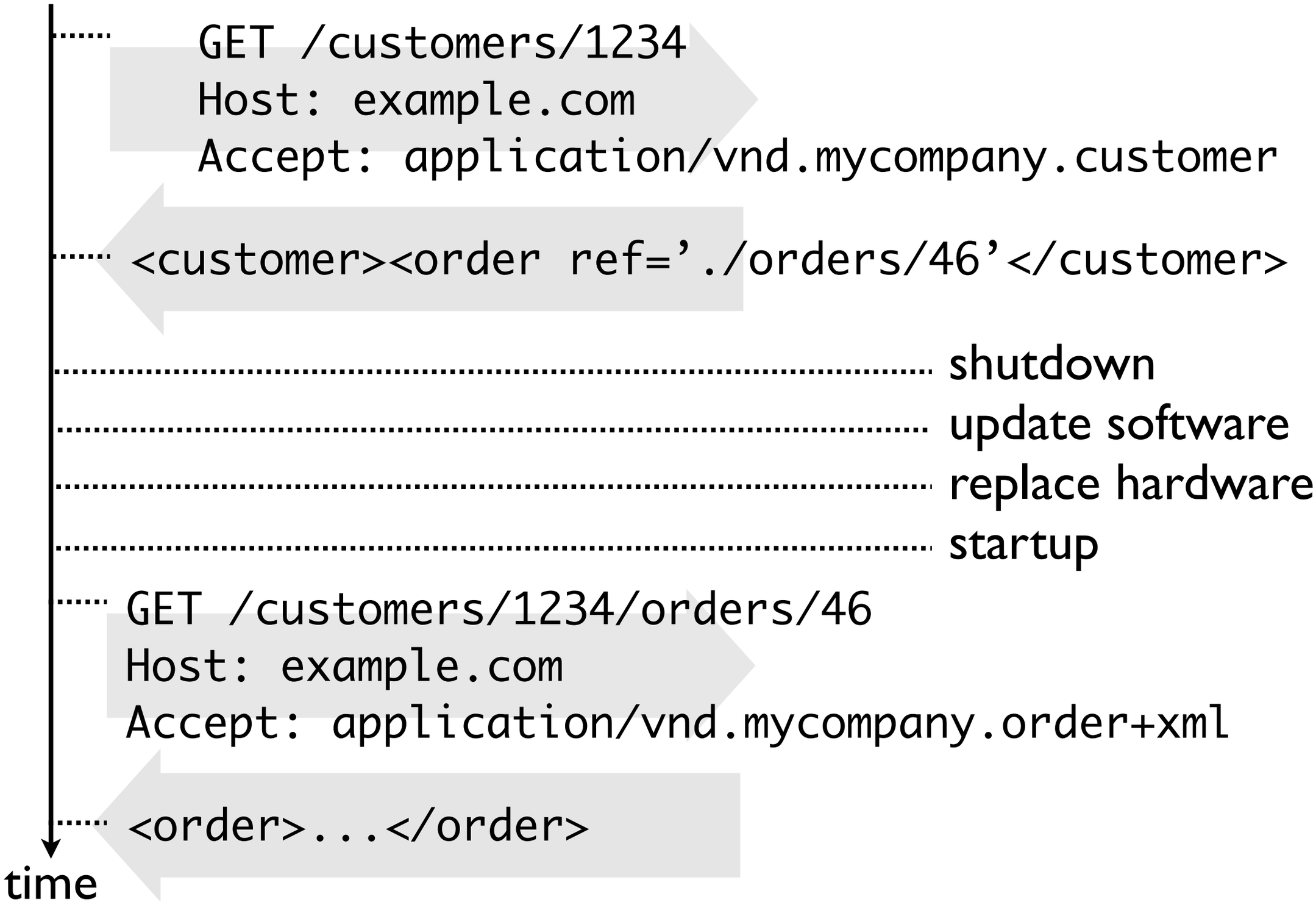
```
GET /customers/1234
Host: example.com
Accept: application/vnd.mycompany.customer+xml
```

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

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

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

5. Communicate Statelessly



What's cool about REST?

```
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

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

Anything that knows
your app

specific

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

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

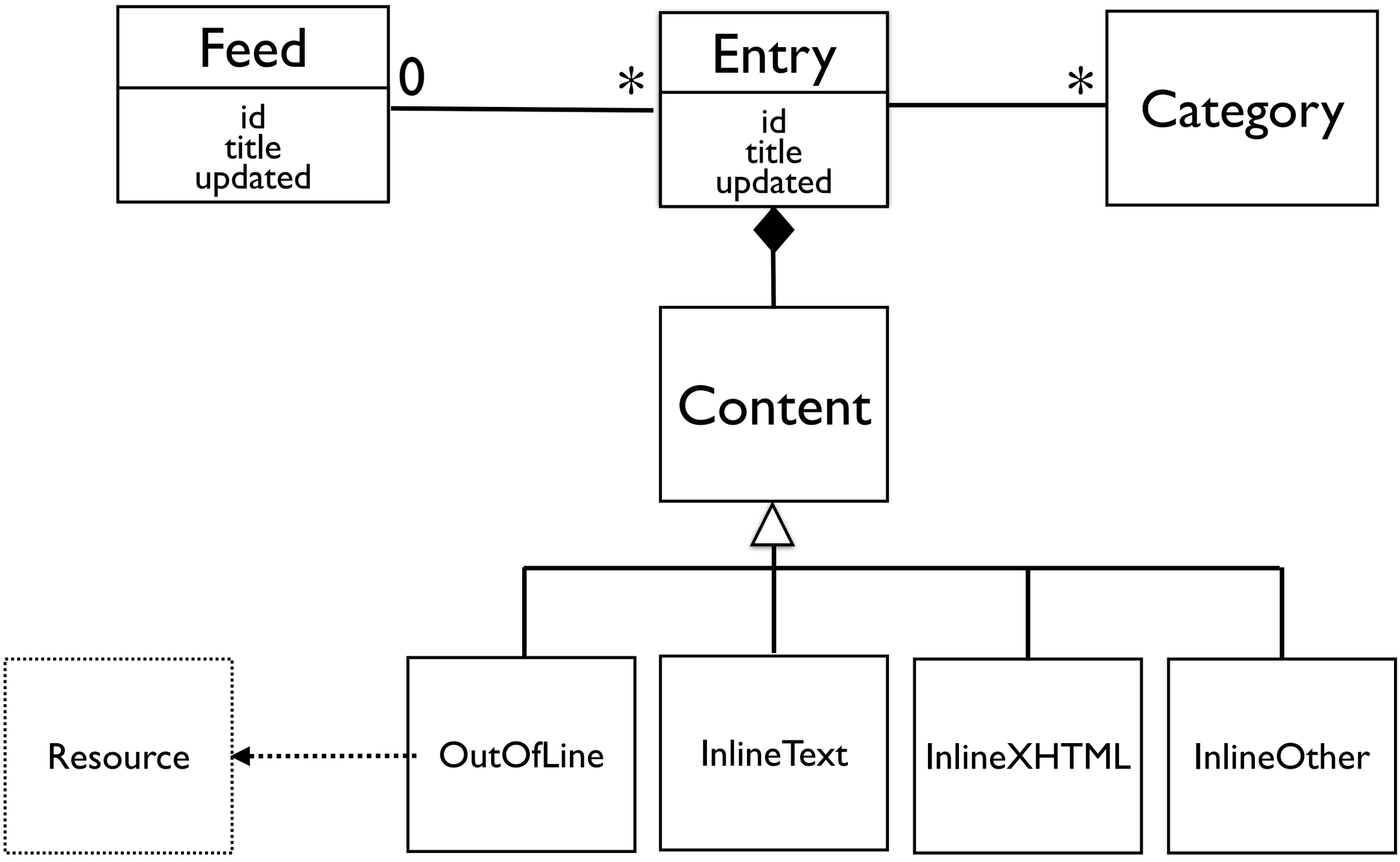
Atom & AtomPub

Atom Syndication Format

Standardized in: RFC 4287
MIME Type: application/atom+xml
Namespace: http://www.w3.org/2005/Atom

RSS Done Right

Atom Model


Feed
id
title
updated
0
*
Entry
id
title
updated
*
Category
Content
Resource
OutOfLine
InlineText
InlineXHTML
InlineOther

```
<?xml version="1.0" encoding="utf-8"?>
<feed xmlns="http://www.w3.org/2005/Atom">
  <title>Example Feed</title>
  <link rel="alternate" type="text/html" href="http://example.org/"/>
  <link rel="self" type="application/atom+xml" href="http://example.org/feeds/23.atom"/>
  <updated>2003-12-13T18:30:02Z</updated>
  <author><name>John Doe</name></author>
  <id>http://example.org/feeds/23</id>

  <entry>
    <title>Atom-Powered Robots Run Amok</title>
    <link href="http://example.org/2003/12/13/atom03"/>
    <id>urn:uuid:1225c695-cfb8-4ebb-aaaa-80da344efa6a</id>
    <updated>2003-12-13T18:30:02Z</updated>
    <summary>Some text.</summary>
  </entry>

  <entry>
    <title>A Second Contrived Example</title>
    <link href="http://example.org/2003/12/13/atom03"/>
    <id>urn:uuid:1225c695-cfb8-4ebb-aaaa-80da344efa6a</id>
    <updated>2003-12-13T18:30:02Z</updated>
    <summary>Some text.</summary>
    <content type="xhtml" xml:base="http://example.org/">
         <div xmlns="http://www.w3.org/1999/xhtml">
           <p><i>Very fine text.</i></p>
         </div>
    </content>
  </entry>
</feed>
```

generic

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

Anything that understands HTTP

```
class AtomFeed : Resource {
      AtomFeed get()
      ...
}
```

Any feed reader

Yahoo! Pipes

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

Anything that knows your app

specific

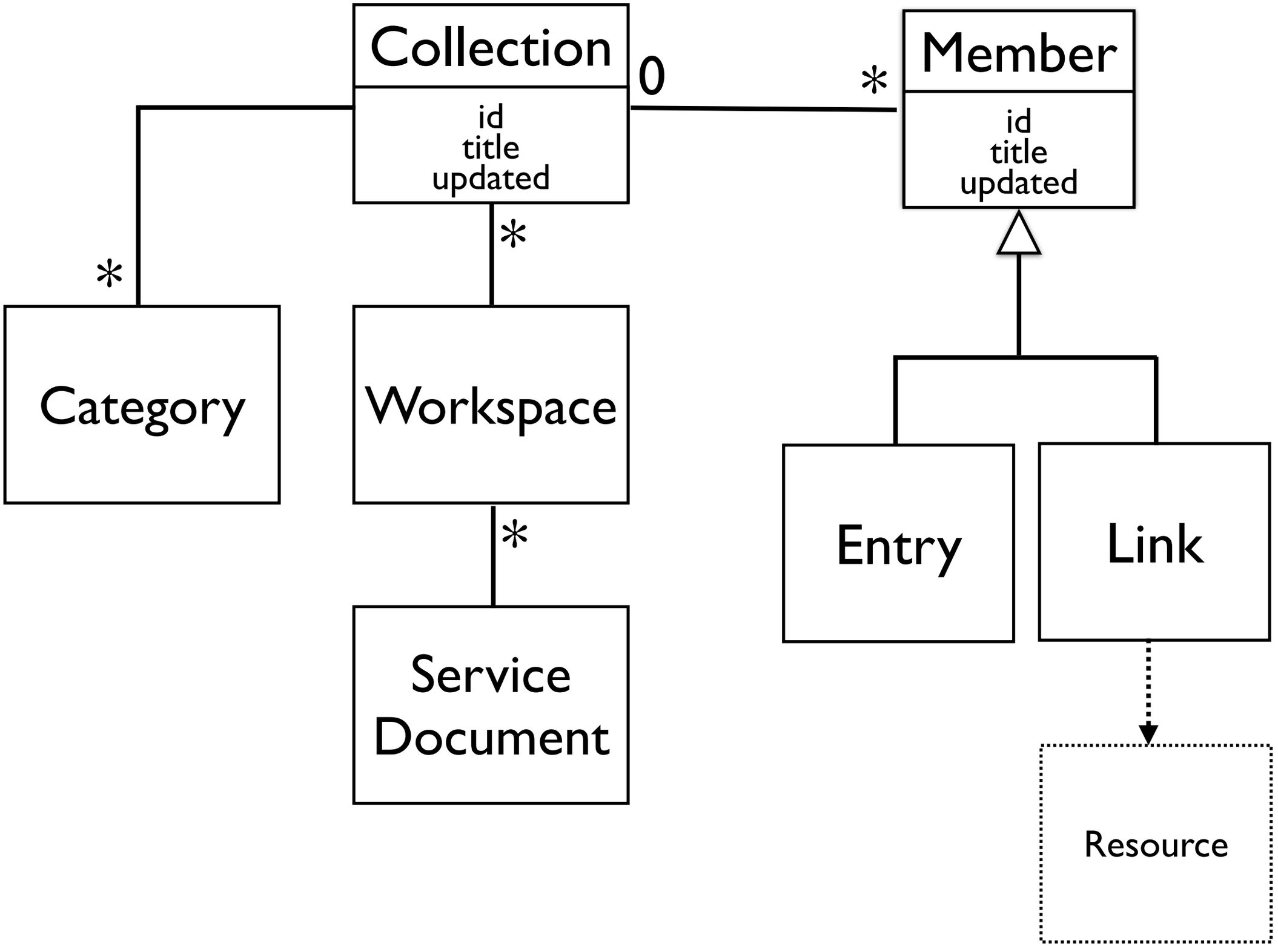
Atom Publishing Protocol

Standardized in: RFC 5023
Namespace: http://www.w3.org/2007/app

RESTful Collections Handling:

- Discovery, Description,
- Retrieval,
- Creation, Editing, Deletion

of Resources


Collection
id
title
updated
0
*
Member
id
title
updated
*
*
Category
Workspace
*
Service
Document
Entry
Link
Resource

generic

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

Anything that understands HTTP

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

Any feed reader

Any AtomPub client

Yahoo! Pipes

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

Anything that knows your app

specific

Claim #6:
RESTful HTTP moves you closer to SOA goals than WS-*

SOA and RESTful HTTP

Loose Coupling
Low Resistance to Change
Unexpected Re-use
Interoperability
Vendor Independence
Available Skills
Tooling for any Language & Platform

... all served better by RESTful HTTP than WS-*

Claim #7:
Advanced WS-*
standards are overrated

WS-* Theory & Practice

WS-Addressing and WS-ReliableMessaging are not yet widely interoperable

Message-bases security is way too expensive

UDDI is a solution looking for a problem

Even WS-based SOAs often don't use them

REST + WS-*

If you choose SOAP, WSDL, WS-*, go with lightweight/OSS tools

Consider adopting RESTful HTTP instead of WS-*

Adopt REST for SOA Governance (!)

Consider WS-* + HTTP GET

Claim #8:
Governance can be greatly simplified

UDDI

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

420-page specification

Finding and maintaining (meta-)model objects

UDDI *(contd.)*

UDDI could be greatly simplified by using plain HTTP

It would no longer be protocol-independent - but who cares?

Atom (Syndication Format & Protocol) are a great match

Approach taken in Mule Galaxy, WSO2 Registry, HP Systinet 2

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

Lightweight Governance

Simple solution (Webserver, WebDAV, AtomStore) for Document Storage

Atom feeds for update notifications

HTTP GET for lookups

(Final) Claim #9: The most important architectural guide is your intelligence

homogeneity
vs.
right tool for the job

integrated solutions
vs.
no dependency on single vendors

central control
vs.
no single point of failure

mainstream
vs.
technical optimum

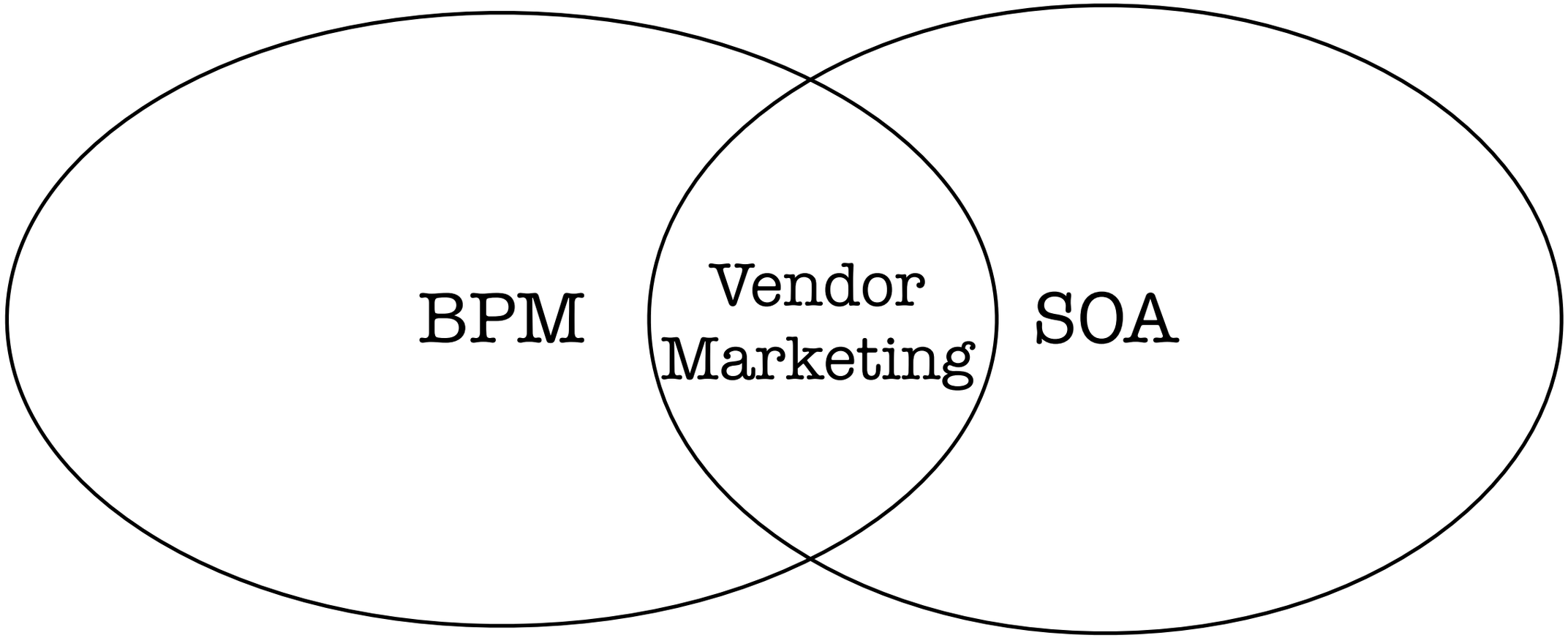
CSOA

Common Sense Oriented Architecture

Q&A

stefan.tilkov@innoq.com
http://www.innoq.com/blog/st/

What about BPM?

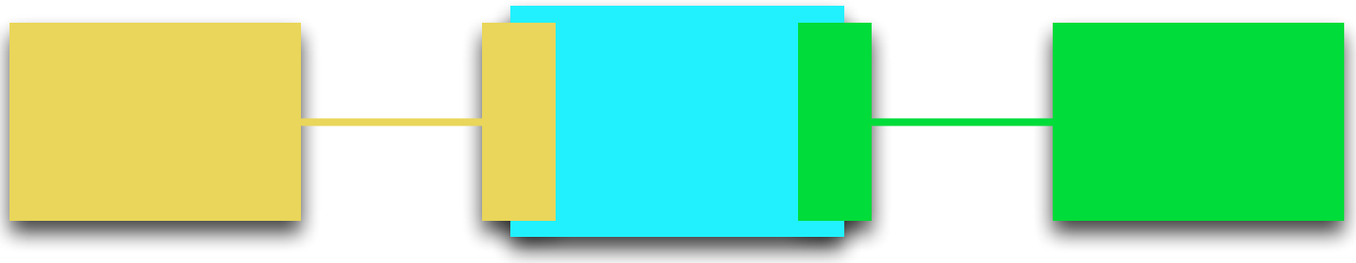


Valid Options:

1. SOA
2. BPM
3. SOA + BPM

Orchestration Options

Central Model



BPM as Impl Model