COMP9321:
Data services engineering

Semester 1, 2018,

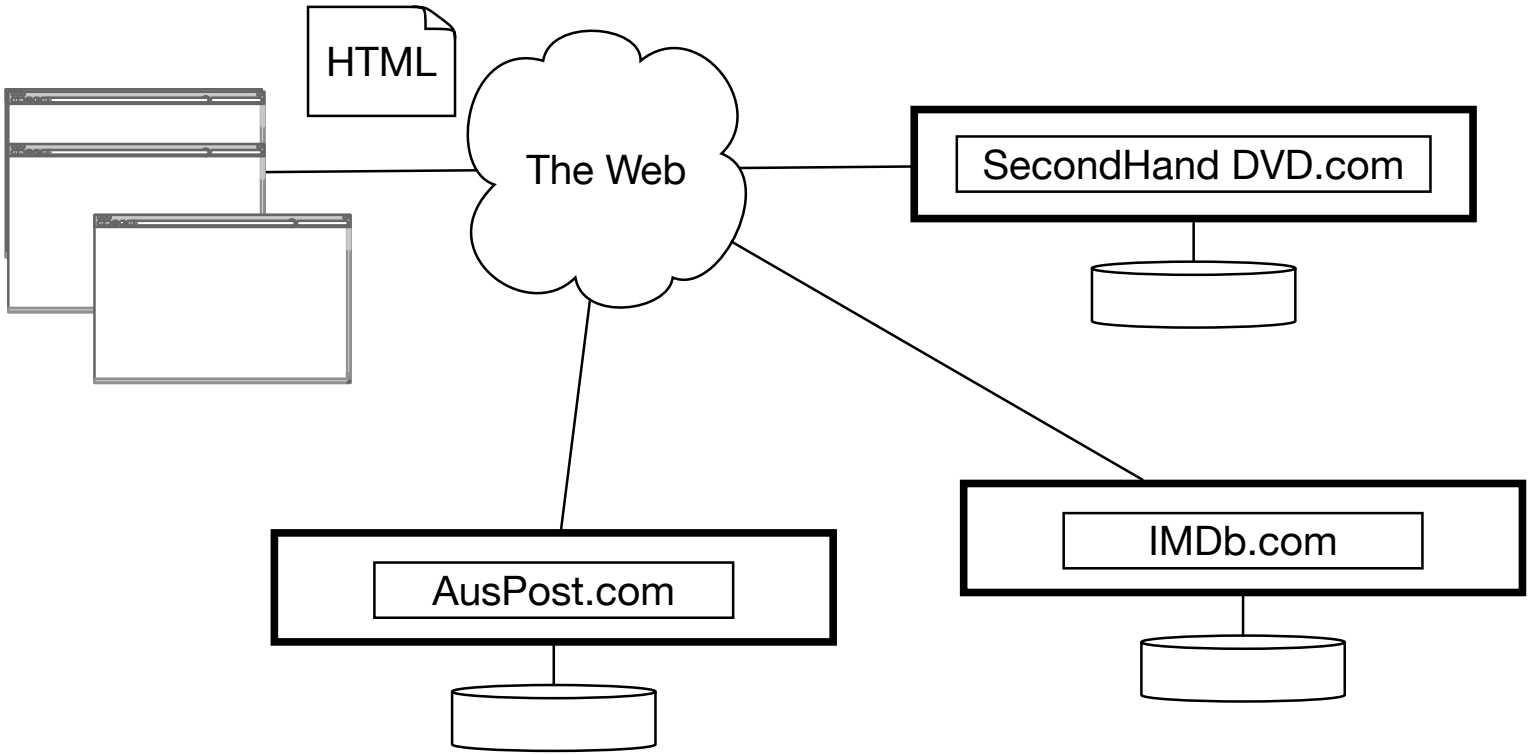
Week 3 RESTful Services

By Helen Paik, CSE UNSW

Back to the story of integrating applications

Over the last 20 years, we have built a lot of Web sites!

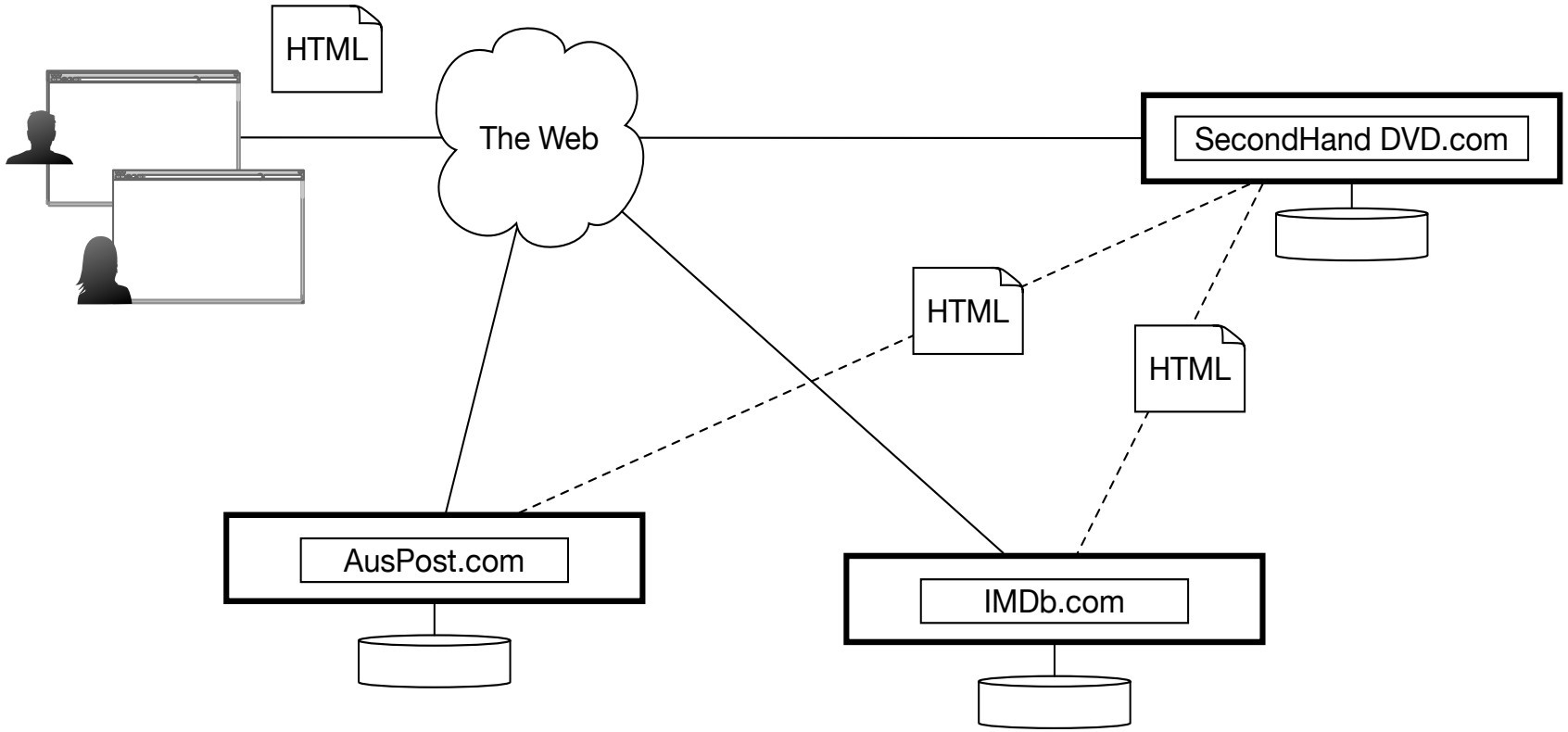
all these now become our legacy system



Story of Web APIs

Web sites in early 2000 were pretty much 1-Tier system from the application integration standpoint

just by using their presentation layer

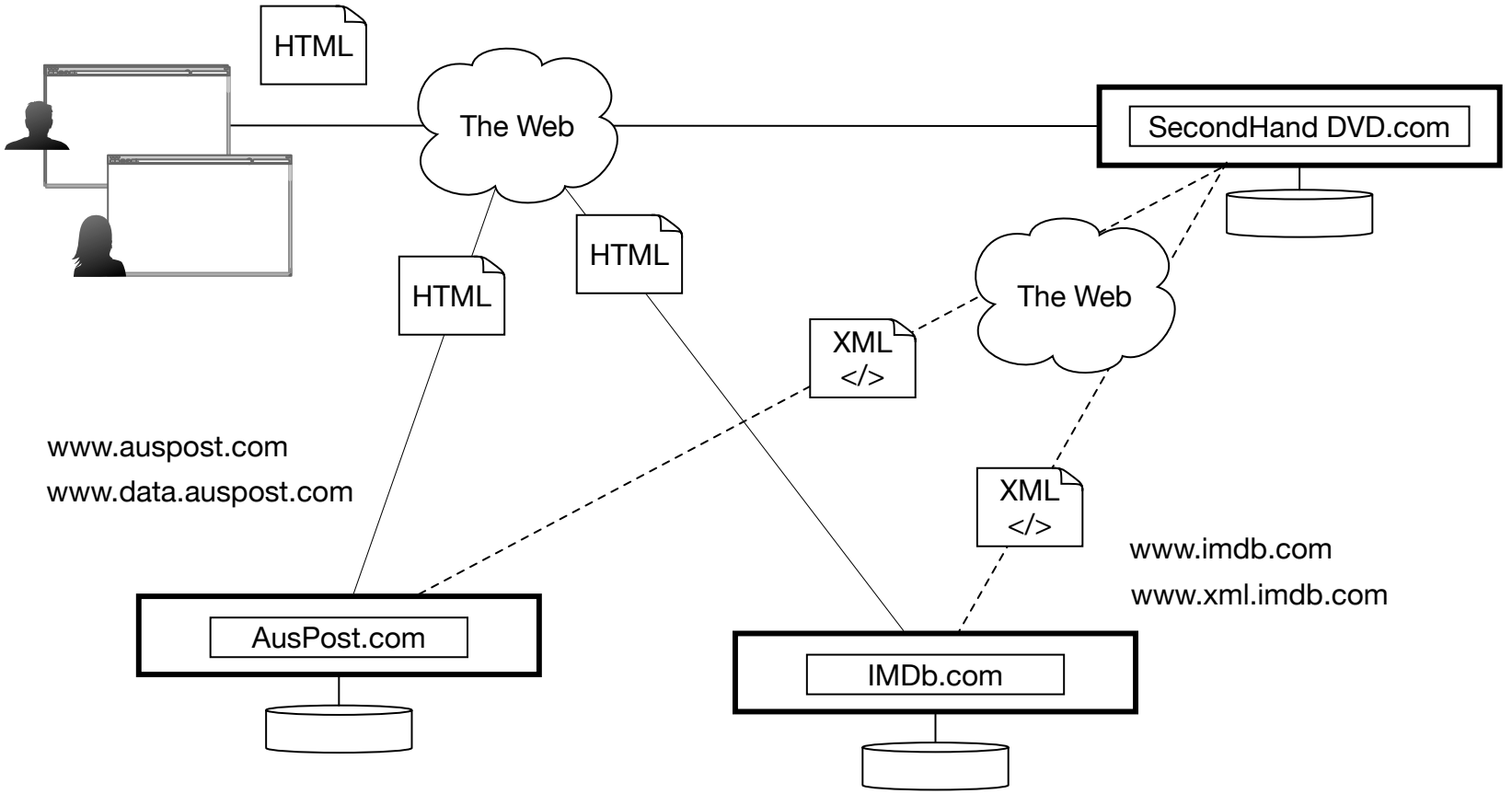


Faking browser clicks (requests) – HTTP/HTML interactions

Brittle … the sources can change without you.

Story of Web APIs

A better version of the idea …

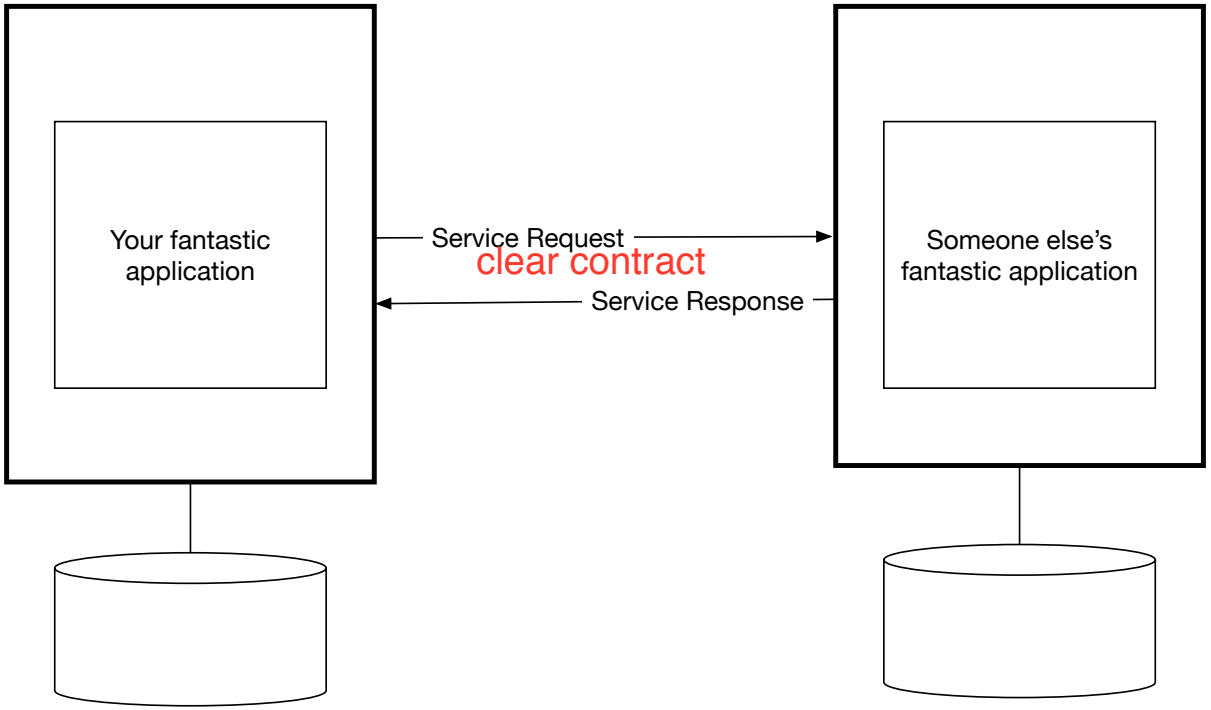


Development of an idea that software-to-software interactions are to be treated differently.

API – Application Programming Interface

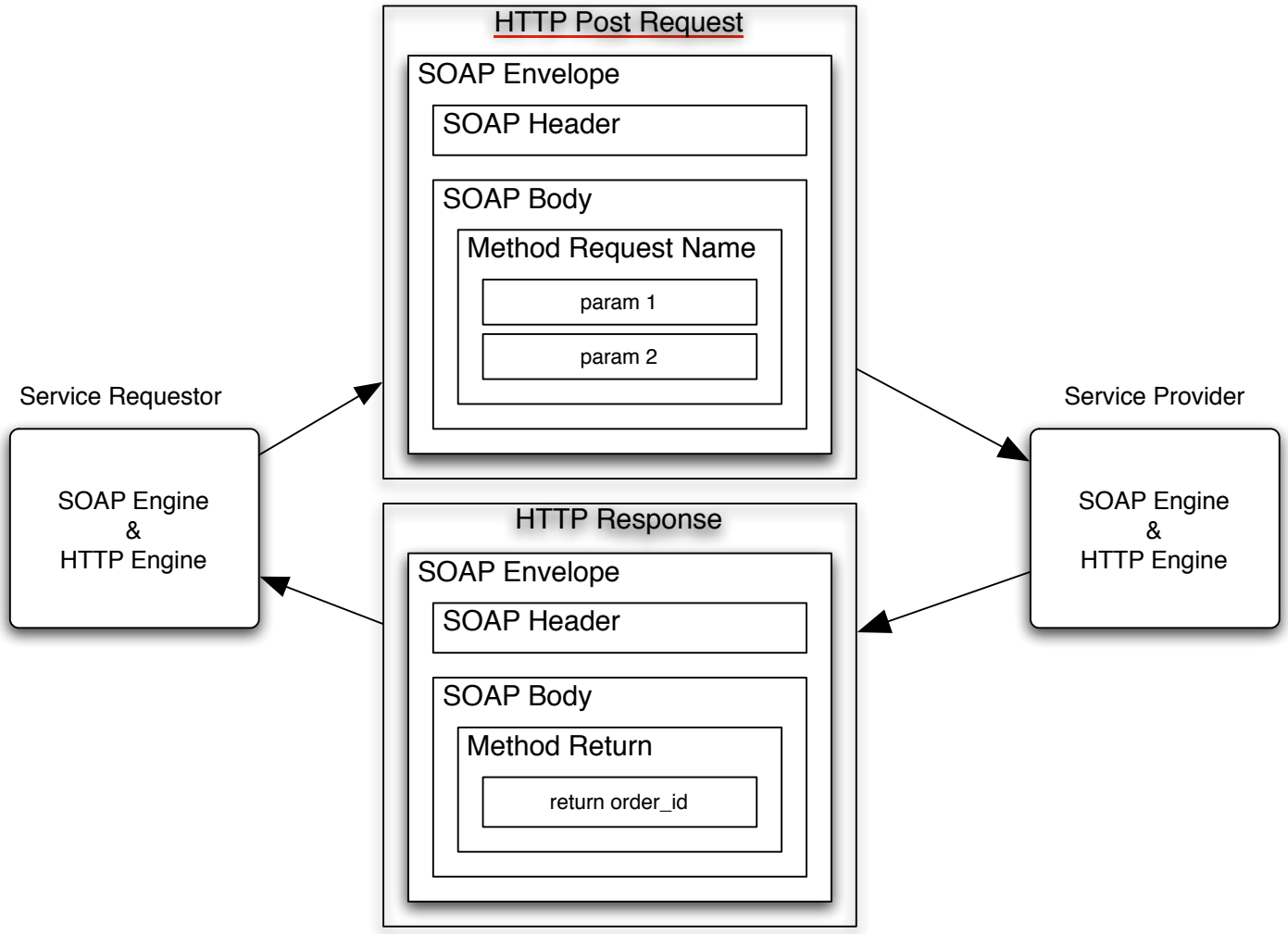
application programming interfaces what it is for software system

talk to other software system to get the required data



- The interface is not meant for human interactions – there is another program on the other side → implication of this: you must have a clear contract (e.g., IOU Alice Bob 100)
- These days companies use APIs internally (private APIs) as well as exposing them externally (public APIs)

XML-based APIs …



Early versions of API utilised XML documents → SOAP protocol (W3C standards), or XmlRPC

XML-based APIs …

Couple of SOAP examples:
http://www.dneonline.com/calculator.asmx
http://www.webservicex.net/stockquote.asmx/

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI"

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <SOAP-ENV:Body>
       <m:GetLastTradePrice xmlns:m="Some-URI">
           <symbol>DIS</symbol>
       </m:GetLastTradePrice>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
   <SOAP-ENV:Body>
       <m:GetLastTradePriceResponse xmlns:m="Some-URI">
         <Price>34.5</Price>
       </m:GetLastTradePriceResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Use HTTP as the 'carrier' of a tightly-constructed messages 使用HTTP作为紧密构造消息的"载体"

http://www.webservicex.net/stockquote.asmx (Web Service Description Language)

Now JSON/REST is ‘preferred’ choice

```
GET /stockquote/DIS HTTP/1.1
Host: www.stockquoteserer.com
Accept: application/json
```

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: xxx

{
  “ticker”: “DIS”,
  “price”: 34.5
}
```

Let’s deconstruct this idea in detail …

What is and Why a RESTful Service

Early XML-based API fell out of favour along with the rise of the number of ‘mobile’ devices (and other ‘non-traditional’ client devices) due to the ‘heavy’ data payload and processing load

由于“大量”数据负载和进程负载
早期基于XML的API随着“移动”设备（以及其他“非传统”客户端设备）
数量的增加而失宠

REST is an architectural style of building networked systems - a set of architectural constraints in a protocol built in that style.

REST是构建网络系统的一种体系结构风格 - 以该风格构建的协议中的一组架构约束。

The protocol in REST is HTTP (the core technology that drives the Web)

Popular form of API ... It is popularised as a guide to build modern distributed applications on the Web – let’s work with the components that the Web itself is built in.

它被普及为在WEB上构建现代分布式应用程序的指南 - 让我们使用Web本身构建的组件进行工作。

REST itself is not an official standard specification or even a recommendation. It is just a “design guideline” for building a system (or a service in our context) on the Web

REST – REpresentational State Transfer

(of resources)

- https://www.youtube.com/watch?v=w5j2KwzzB-0
- http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

Resources in REST

In REST, everything starts and ends with resources (the fundamental unit).

What is a Resource?

The key abstraction of information in REST is a resource. Any information that can be named can be a resource: a document or image, a temporal service (e.g. today's weather in Los Angeles), a collection of other resources, a non-virtual object (e.g. a person), and so on. In other words, any concept that might be the target of an author's hypertext reference must fit within the definition of a resource. – Roy Fielding's dissertation.

name / link

Resources and Representational States

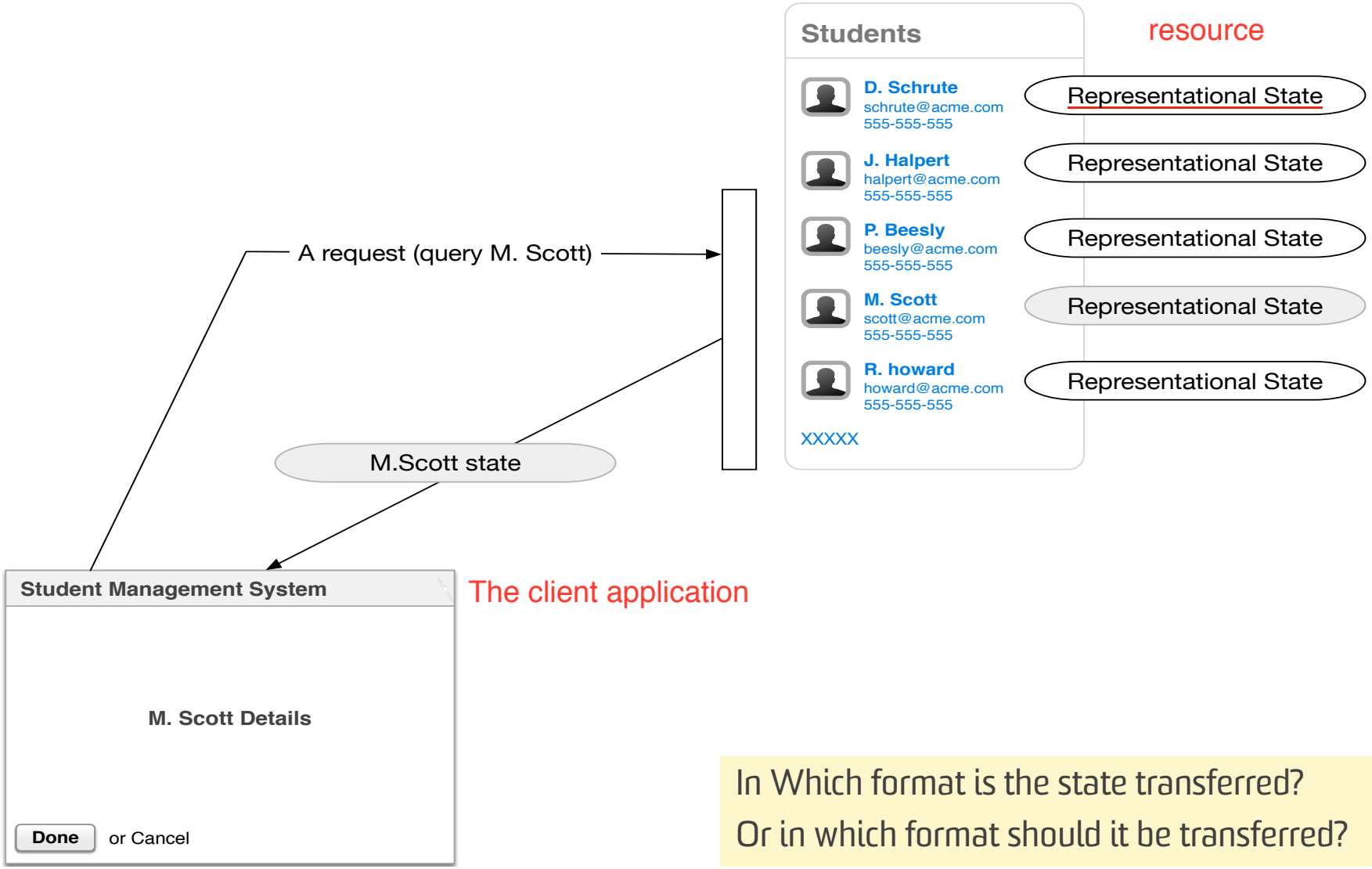
What is a Resource? (more concretely …)

A resource is a thing that:

- is unique (i.e., can be identified uniquely) 可以唯一地标识
- has at least one representation
- has one or more attributes beyond ID
- has a potential schema, or definition 潜在模式或定义洽 some sort of attributes
- can provide context (state) – which can change (updated)
- is reachable within the addressable universe
- collections, relationships

e.g., Students, Courses, Program

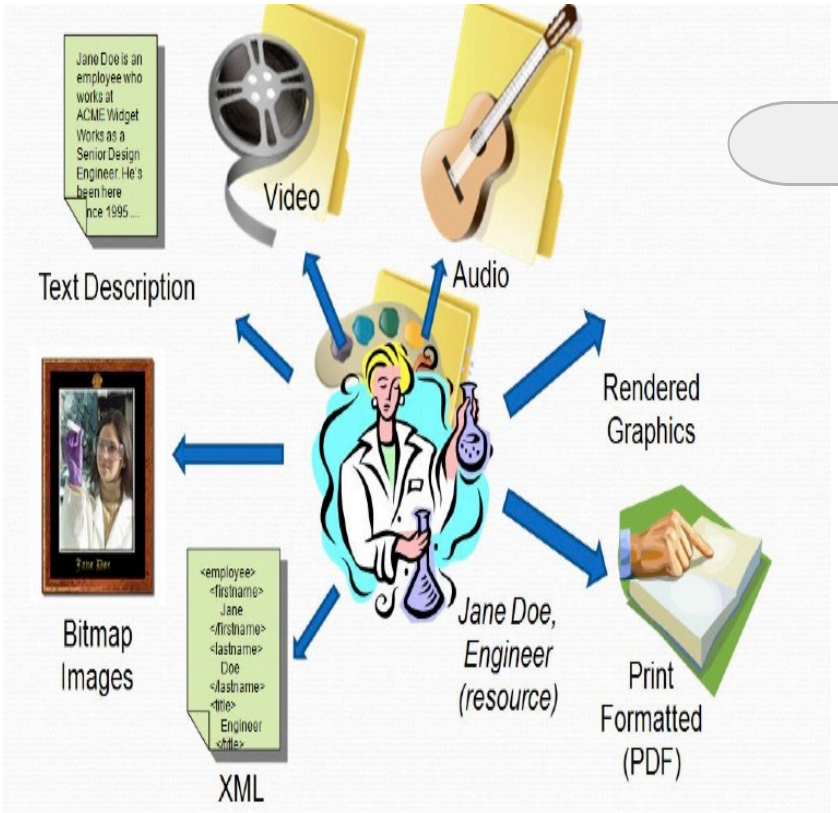
Representational State Transfer

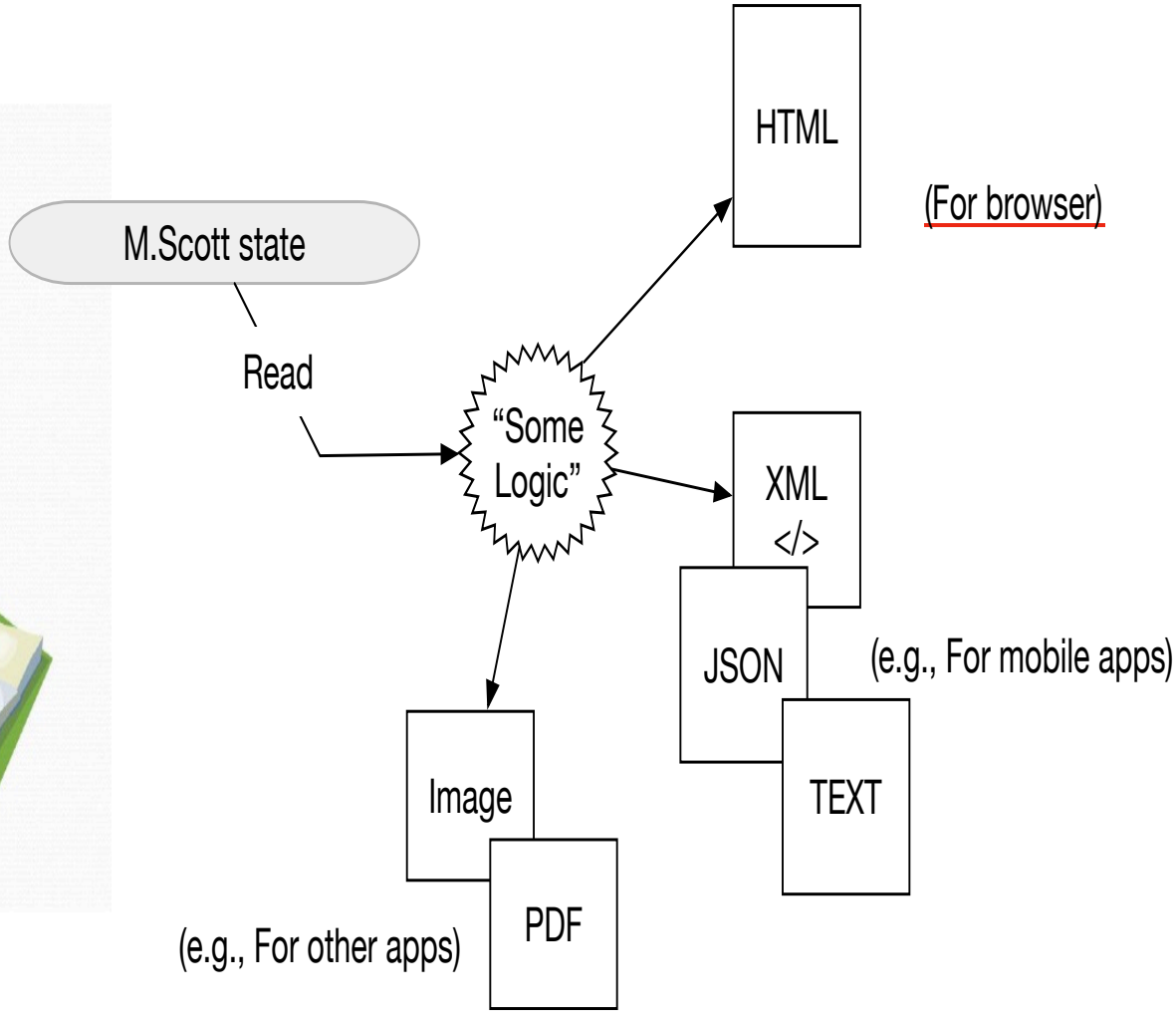


In Which format is the state transferred?

Or in which format should it be transferred?

Resource's Multiple Representations





Based on the needs of the consumer … (REST principles do not specify "standard data format")

What makes a RESTful Service?

重要概念:

REST is an architectural style of building networked systems - a set of architectural constraints in a protocol built in that style.

A RESTful service/API MUST meet the architectural constraints by following the design guide/principles (i.e., you can say what is and is not RESTful)

Architectural Constraints of REST

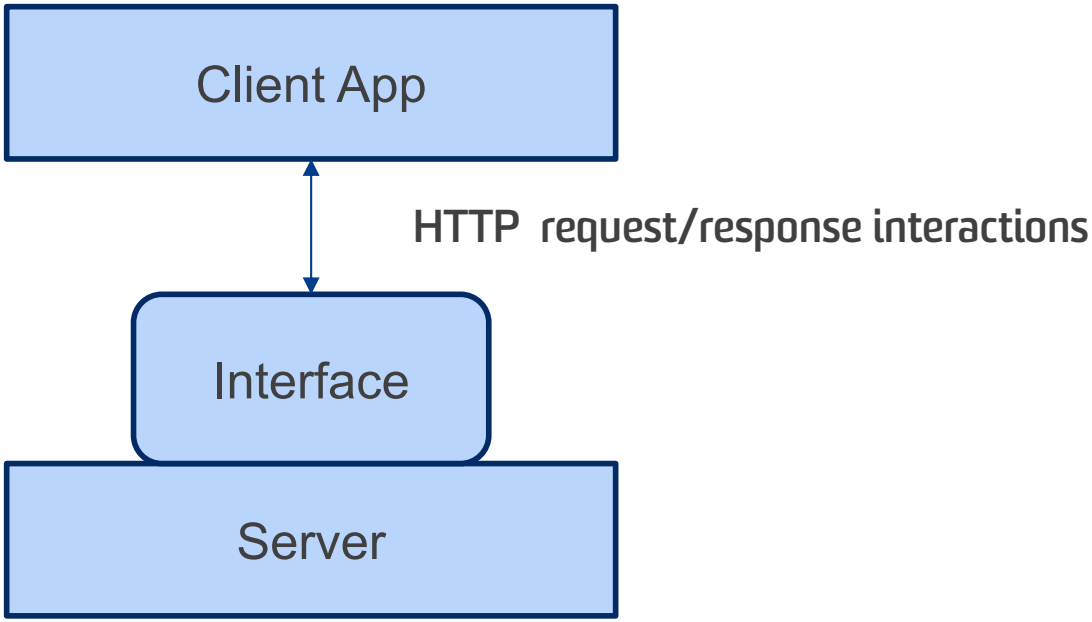
重要概念:

1. Client-Server
2. Uniform Interface
3. Statelessness
4. Caching
5. Layered System
6. Code on demand (optional)

If your design satisfies the first five, you can say your API is 'RESTful'

Client-Server

1. Client-Server
2. Uniform Interface
3. Statelessness
4. Caching
5. Layered System



The same idea as the classic two-tier architecture, foundation of REST architecture.

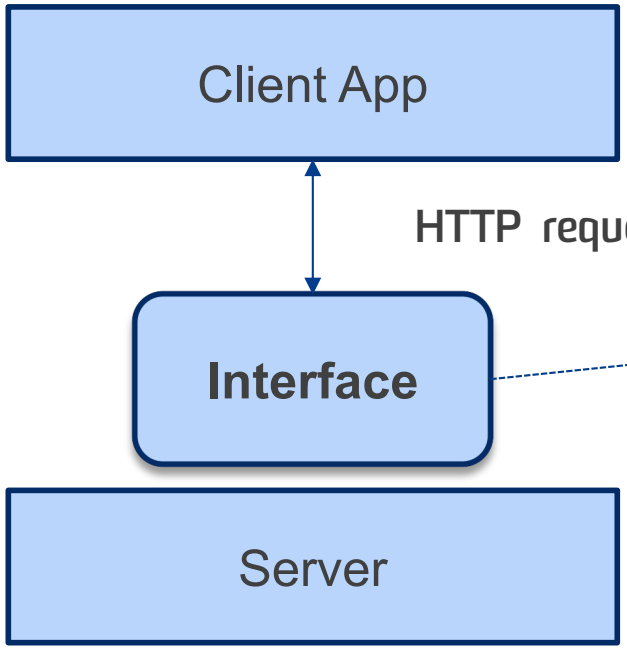
Server-side (business logic layer and beyond) is physically separated from the client app

WHY good thing? – separation of concerns, they can evolve without affecting each other

- Server-side: performance, scaling, data management, data security, etc.
- Client App: user experience, multiple form factor/devices support, etc.

Uniform Interface

1. Client-Server
2. Uniform Interface
3. Statelessness
4. Caching
5. Layered System



- UNIFORM INTERFACE
  - Uniform ≈ "Common" in All RESTful interface
  - Contract for communication btw C-S
  - Described using HTTP methods and Media Type only

REST design principles go into how to design the interface to meet the constraint.

On Uniform Interface – Resource URI ID

Resources are identified by a URI (Uniform Resource Identifier)

A resource has to have at least one URI

Most well-known URI types are URL and URN

- URN (Uniform Resource Name) a standard naming scheme
  - urn:isbn:0-486-27557-4 (Shakespeare's Romeo and Juliet book)
  - urn:isan:0000-0000-9E59-0000-O-0000-0000-2 (2002 spider man film)

- URL (Uniform Resource Locator)
  - file:///home/tommy/plays/RomeoAndJuliet.pdf
  - http://home/tommy/plays/RomeoAndJuliet.html

Every URI designates exactly one resource 每个Uniform Resource Identifier都指定一个资源

- http://www.example.com/software/releases/1.0.3.tar.gz
- http://www.example.com/software/releases/latest

On Uniform Interface - Addressability 寻址

The resources must be addressable

An application is 'addressable' if it exposes its data set as resources (i.e., usually a large number of URIs)

- File system on your computer is addressable system
- The cells in a spreadsheet are addressable (cell referencing)
  电子表格中的单元格是可寻址的（单元格引用）

flickr – a good example of "addressable" Web
you can bookmark, use it as link in a program, you can email, etc.

Systems that are not addressable ?

REST advocates identification and addressability of resources as a main feature of the Uniform Interface constraint 资源的识别和可寻址性 是 统一接口约束的主要特征

On Uniform Interface - Representations

state

A resource needs a representation for it to be sent to the client means what format it should come

a representation of a resource - some data about the 'current state' of a resource

E.g., On some software project, a list of open bugs can have representations in :–

- an XML file,
- a web page,
- comma-separate-values,
- printer-friendly-format, etc.

when a representation of a resource may also contain metadata about the resource (e.g., books: book itself + metadata such as cover-image, reviews, other related books) - relationships.

Representations can flow the other way too: a client send a new or updated 'representation' of a resource and the server creates/updates the resource.

On Uniform Interface - Representations

And … you shall provide multiple representations …

Deciding between multiple representations

Option 1.

Have a distinct URI for each representation of a resource:

- http://www.example.com/release_doc/104.en (English)
- http://www.example.com/release_doc/104.es (Spanish)
- http://www.example.com/release_doc/104.fr (French)

Looks very “addressable” → good!

On Uniform Interface - Representations

Deciding between multiple representations

Option 2.

Use HTTP HEAD (metadata) for content negotiation:

Expose a single URL: http://www.example.com/release_doc/104

Client HTTP request contains Accept-Language

Content Negotiation (part of HTTP spec)

Other types of request metadata can be set to indicate all kinds of client preferences,

e.g., file format, payment information, authentication credentials, IP address of the client, caching directives, and so on.

Option 1 or 2 are both acceptable REST-based solution ...

On Uniform Interface – Description Syntax

Use pure HTTP methods as main operations on resources

What HTTP Spec says about the following methods:

GET: Retrieve a representation of a resource.

PUT: Create a new resource (new URI) or update a resource (existing URI)

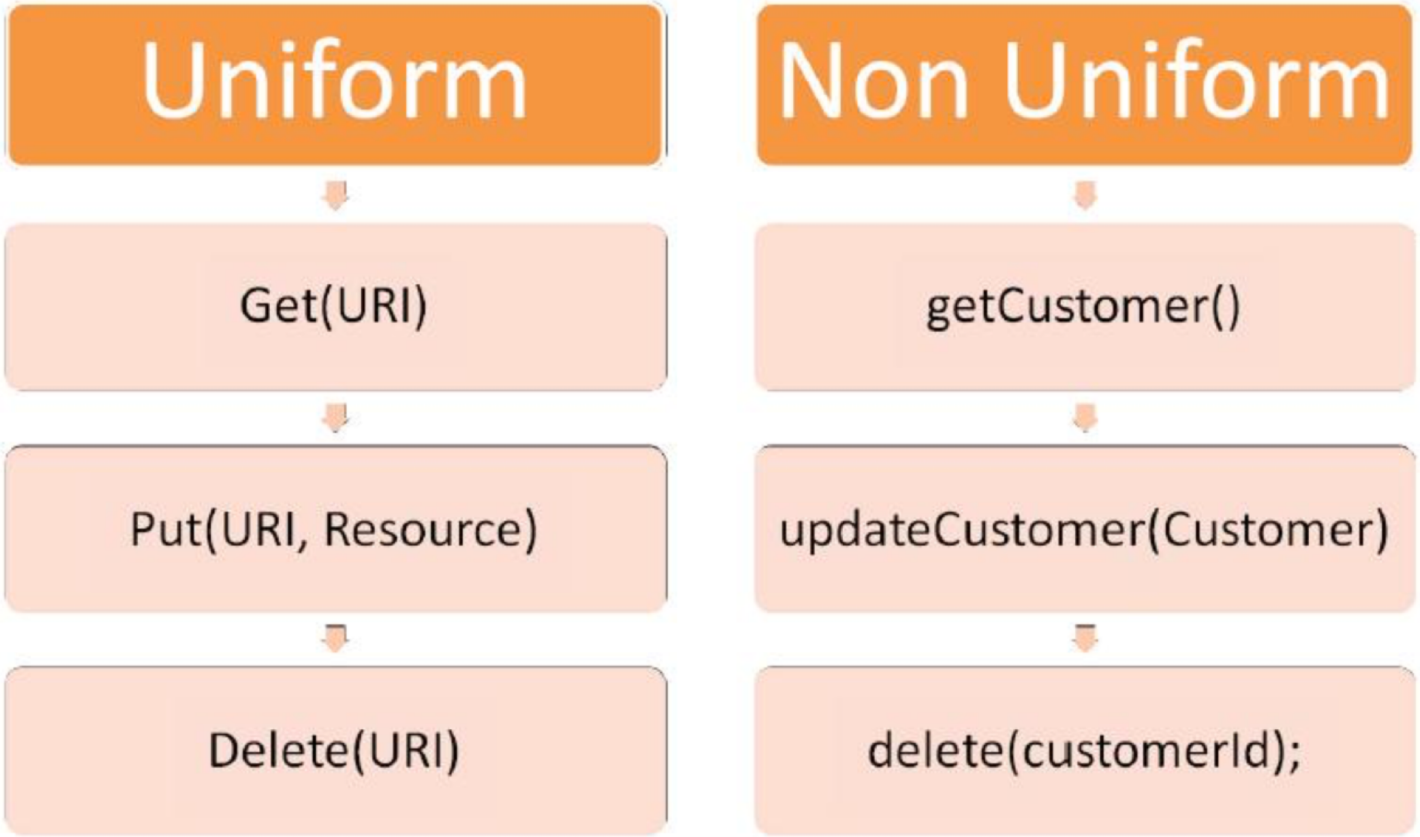
DELETE: Clear a resource, after the URI is no longer valid

POST*: Modify the state of a resource. POST is a read-write operation and may change the state of the resource and provoke side effects on the server. Web browsers warn you when refreshing a page generated with POST.

HEAD, OPTIONS and PATCH

Similar to the CRUD (Create, Read, Update, Delete) databases operations *POST: a debate about its exact best-practice usage ...

On Uniform Interface – “Uniform” Interface



On Uniform Interface – Description Syntax

Given a resource (coffee order): a representation in XML

```
<order>
    <drink>latte</drink>
</order>
```

What HTTP Spec says about the input/output of the following methods. What does the following operations mean and what do they return?

POST /starbucks/orders

GET /starbucks/orders/order?id=1234

PUT /starbucks/orders/order?id=1234

DELETE /starbucks/orders/order?id=1234

On Uniform Interface - POST and PUT

POST creates a new resource

- But ... the server decides on that resource's URI and returns the new URI for the resource in the response
- Common examples: creating a new employee, a new order, a new blog posting, etc.

PUT "creates" or "updates" a resource:

- But ... the URI is given in the request input by client
- if existing, the contained entity is considered as a modified version of the resource ...

Generally, POST to create, PUT to update ...

On Uniform Interface - PUT and PATCH

PATCH is added later in the HTTP spec to support "partial updates" of a resource:

HTTP spec says: In a PUT request, the enclosed entity is considered to be a modified version of the resource stored on the origin server, and the client is requesting that the stored version be replaced. With PATCH, however, the enclosed entity contains a set of instructions describing how a resource currently residing on the origin server should be modified to produce a new version.

在PUT请求中，封闭的实体被认为是存储在源服务器上的资源的修改版本
并且客户端正在请求替换存储的版本
但是，对于PATCH，封闭实体包含一组说明，说明如何当前驻留在源服务器上的资源应该被修改以生成新版本。

PATCH /students/0001 [{ "op": "replace", "path": "/DOB", "value": "12/12/1990" }]

Neither safe nor idempotent ...

即不安全也不幂等

On Uniform Interface – Safe & Idempotent

Uniform Interface must be safe and idemponent

- *Being Safe*

安全 幂等

Read-only operations ... The operations on a resource do not change any server state. The client can call the operations 10 times, it has no effect on the server state.

(like multiplying a number by 1, e.g., 4x1, 4x1x1, 4x1x1x1, ...) 相当于乘以1

- *Being Idempotent*

Operations that have the same “effect” whether you apply it once or more than once. An effect here may well be a change of server state. An operation on a resource is idempotent if making one request is the same as making a series of identical requests.

(like multiplying a number by 0, e.g., 4x0, 4x0x0, 4x0x0x0, ...)

无论您应用一次还是多次应用相同“效果”的操作。

这里的效果可能是服务器状态的改变。
对一个资源的操作是幂等的，如果提出一个请求与提出一系列相同的请求相同。

On Uniform Interface – Safe & Idempotent

Safety and Idempotency

- GET: safe (and idempotent)
- HEAD and OPTION: safe

- PUT: - idempotent
  - If you create a resource with PUT, and then resend the PUT request, the resource is still there and it has the same properties you gave it when you created it.
    If you update a resource with PUT, you can resend the PUT request and the resource state won't change again
- DELETE: - idempotent
  - If you delete a resource with DELETE, it's gone. You send DELETE again, it is still gone !

POST is neither safe nor idempotent

On Uniform Interface – Safety & Idempotent

Why Safety and Idempotency matter:

The two properties let a client make reliable HTTP requests over an unreliable network.
这两个属性让客户端通过不可靠的网络提供可靠的HTTP请求。

Your GET request gets no response? make another one. It's safe.

Your PUT request gets no response, make another one. Even if your earlier one got through, your second PUT will have no side-effect
即使你的前一个通过了，你的第二个PUT也没有副作用

There are many applications that misuse the HTTP uniform interface. e.g.,

- GET https://api.del.icio.us/posts/delete
- GET www.example.com/registration?new=true&uname=John&passwd=123

? Many expose unsafe operations as GET and there are many use of POST operation which is neither safe nor idempotent. Repeating them has consequences ...

On Uniform Interface – Safe & Idempotent

Why Safety and Idempotency matter

Allows the “Uniformity” in REST interface (≈ accepted convention by the community)

The point about REST Uniform Interface is in the ’uniformity’: that every service uses HTTP’s interface the same way.

It means, for example, GET does mean ’read-only’ across the Web no matter which resource you are using it on.

例如，无论您使用哪种资源，GET在整个Web上的含义都是“只读”。

It means, we do not use methods in place of GET like doSearch or getPage or nextNumber.

It is not just having a method called GET in your service, it is about using it the way it was meant to be used.

On Uniform Interface – Linked Resources

Representations are hypermedia: resource (data itself) + links to other resources

e.g., Google Search representation:

Jellyfish ☆
Jellyfish are most recognised because of their jelly like appearance and this is where they get their name. They are also recognised for their bell-like ...
www.reefed.edu.au › ... › Corals and Jellyfish - Cached - Similar

Jellyfish - Wikipedia, the free encyclopedia ☆
Jellyfish (also known as jellies or sea jellies) are free-swimming members of the phylum Cnidaria. **Jellyfish** have several different morphologies that ...
Terminology - Anatomy - Jellyfish blooms - Life cycle
en.wikipedia.org/wiki/**Jellyfish** - Cached - Similar

Searches related to **jellyfish**

jellyfish **facts**	**types of** jellyfish	jellyfish **pictures**	**blue bottle** jellyfish
jellyfish **stings**	jellyfish **photos**	jellyfish **reproduction**	jellyfish **life cycle**

Goooooooooo gle ▶
1 2 3 4 5 6 7 8 9 10 **Next**

On Uniform Interface – Linked Resources

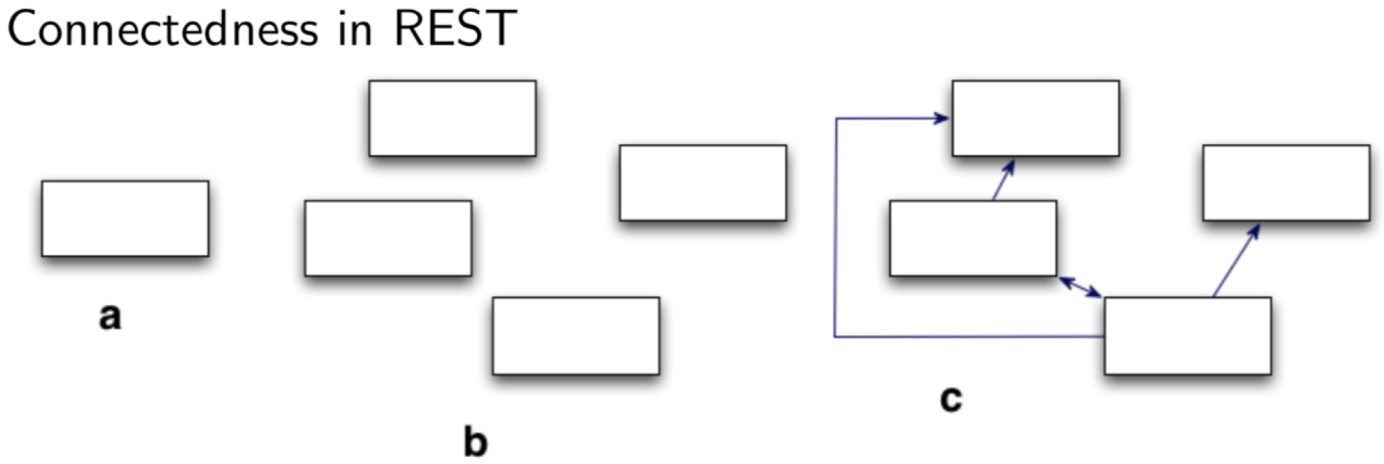
R. Fielding talks about: “Hypermedia as the engine of application state”

The current state of an HTTP ‘session’ is not stored on the server as a resource state, but tracked by the client as an application state, and created by the path the client takes through the Web. The server guides the client’s path by serving hypermedia: links and forms inside resource representations.

HTTP'session'的当前状态不是作为资源状态存储在服务器上，而是作为应用程序状态由客户端跟踪并由客户端通过Web访问的路径创建。
服务器通过服务超媒体来指导客户端的路径：资源表示中的链接和表单。

The server sends the client guidelines about which states are near and current one.

On Uniform Interface – Linked Resources



All three services expose the same functionality, but their usability increases towards the right

- Service A is a typical remote-function style service, exposing everything through a single URI. Neither addressable, nor connected
- Service B is addressable but not connected; there are no indication of the relationships between resources. A hybrid style ...混合风格
- Service C is addressable and well-connected; resources are lined to each other in ways that make sense. A fully RESTful service

Statelessness

1. Client-Server
2. Uniform Interface
3. Statelessness
4. Caching
5. Layered System

REST API must be stateless

All calls from clients are independent

Stateless means every HTTP request happens in a complete isolation. Stateless is good !!
- scalable, easy to cache, addressable URI can be bookmarked (e.g., 10th page of search results)

可缩放，易于缓存，可寻址URI可以被加书签（例如，搜索结果的第10页）

HTTP is by nature stateless. We do something to break it in order to build applications

the most common way to break it is 'HTTP sessions'

the first time a user visits your site, he gets a unique string that identifies his session with the site

用户第一次访问您的网站时，他会得到一个唯一的字符串来标识他与该网站的会话

http://www.example.com/forum?PHPSESSIONID=27314962133

the string is a key into a data structure on the server which contains what the user has been up to.

all interactions assume the key to be available to manipulate the data on the server

On Statelessness

What counts as ‘state’ exactly?

Think a Flickr.com-like web site ... you will add photos, rename photos, share them with friends, etc. – what would ‘being stateless’ mean here?

重要概念

KEY notion: separation of client application state and RESTful resource state.

- consider the application state as data that could vary by client, and per request.
- consider the resource state as data that could is centrally managed by the server. It is the same for every client.
- resource state live on the server
- individual client application states should be kept off the server

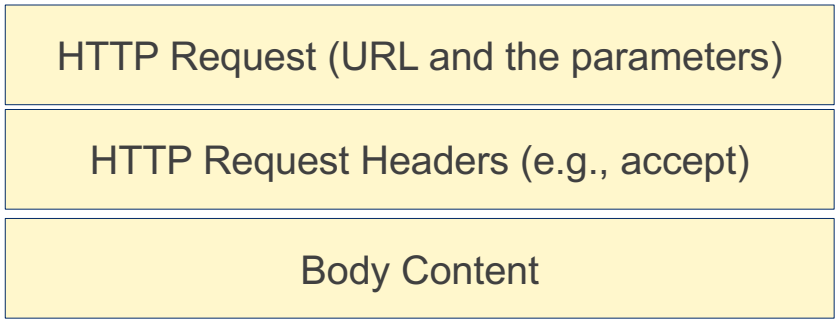
Consider a scenario: a little photo edit app + Flickr and other APIs ...

On Statelessness

Statelessness in REST applies to the client application state (from the server's view point)

What does this mean to the client application?

- every REST request should be totally self-descriptive
- client transmit the state to the server for every request that needs it.



a RESTful service requires that the application state to stay on the client side. Server does not keep the application state on behalf of a client

Consider a scenario: Tracking how many times a user has used your API ...

Caching

1. Client-Server
2. Uniform Interface
3. Statelessness
4. Caching
5. Layered System

Responses must be marked ‘cachable’ or ‘non-cachable’

Well-managed caching partially or completely eliminates some client–server interactions, improving scalability and performance.

Being Statelessness: every action happens in isolation:

- Keeps the interaction protocol simpler
- But the interactions may become ‘chattier’

To scale, RESTful API must be work-shy (only generate data traffic when needed, other times use cache)

This requires ’server-client’ collaboration:

- Client provide guard clauses in requests so that servers can determine easily if there’s any work to be done
- If-Modified-Since, Last Modified, If-None-Match/ETag

Caching

Request

```
GET /transactions/debit/1234 HTTP 1.1
Host: bank.example.org
Accept: application/xml
If-None-Match: aabd653b-65d0-74da-bc63-4bca-ba3ef3f50432
```

Response

```
200 OK
Content-Type: application/xml
Content-Length: ...
Last-Modified: 2010-21-04T15:10:32Z
Etag: abbb4828-93ba-567b-6a33-33d374bcad39
<t:debit xmlns:t="http://bank.example.com">
<t:sourceAccount>12345678</t:sourceAccount>
<t:destAccount>987654321</t:destAccount>
<t:amount>299.00</t:amount>
<t:currency>GBP</t:currency>
</t:debit>
```

Caching

Request

```
GET /transactions/debit/1234 HTTP 1.1
Host: bank.example.org
Accept: application/xml
If-Modified-Since: 2010-21-04T15:00:34Z
```

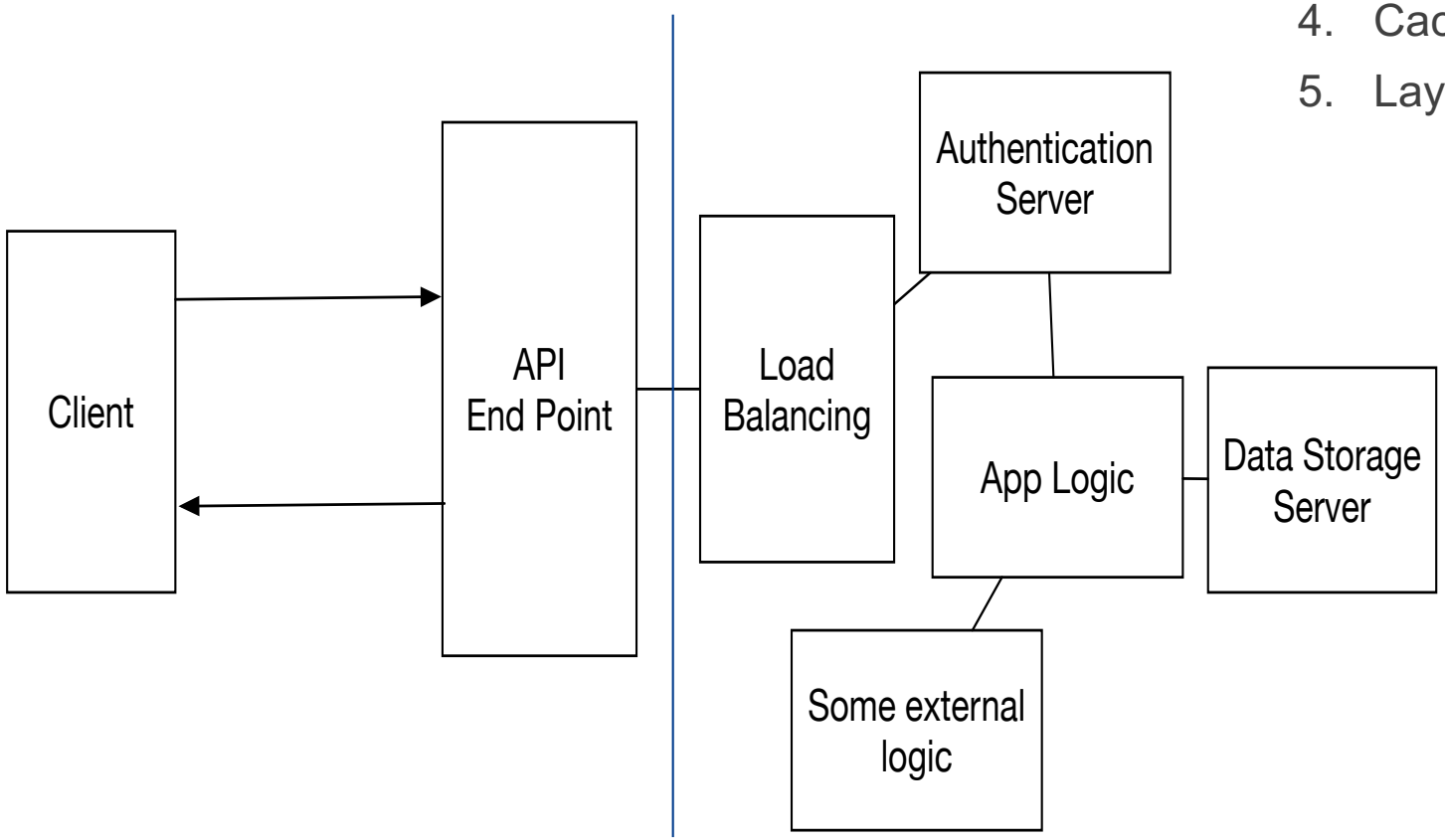
Response

```
200 OK
Content-Type: application/xml
Content-Length: ...
Last-Modified: 2010-21-04T15:00:34Z
```

Server needs cache management policy and implementation …

Layered System

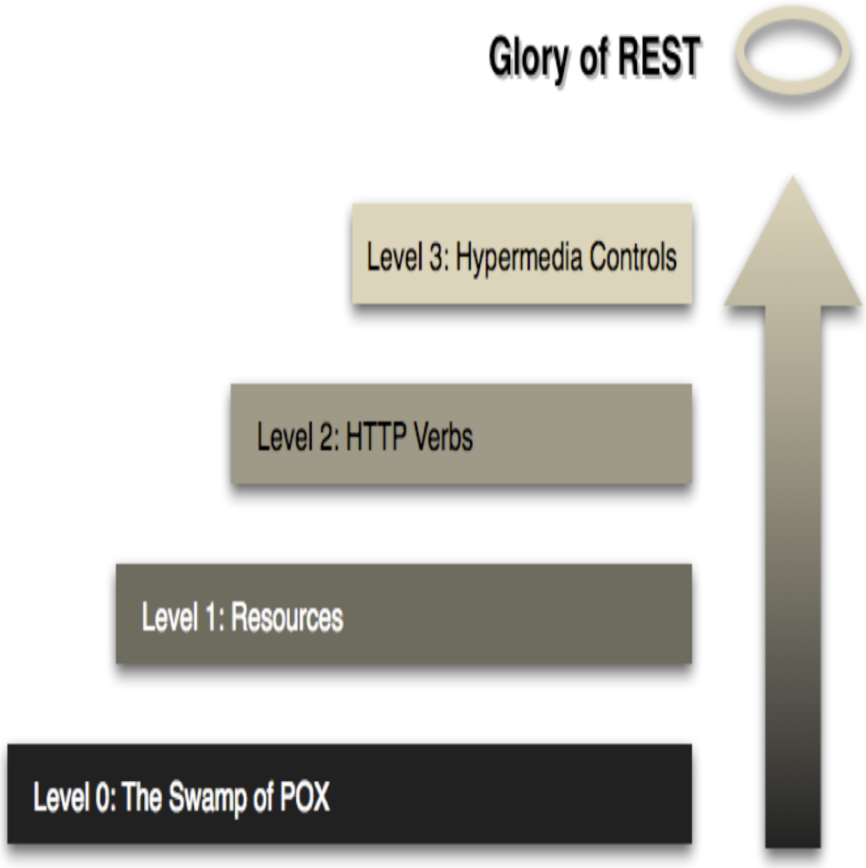
1. Client-Server
2. Uniform Interface
3. Statelessness
4. Caching
5. Layered System



A client cannot ordinarily tell whether it is connected directly to the end server, or to an intermediary along the way.

Again, de-coupling allows the components I the architecture to evolve independently

The Richardson Maturity Model



Leonard Richardson: can we measure to what level your service is RESTful?

Level 0: One URI (single endpoint) exposed, requests contain operation details

Level 1: Expose resource URIs - individual URIs for each resource. Requests could still contain some operation details

Level 2: HTTP Methods - use the standard HTTP methods, status codes with the resource URIs,

Level 3: HATEOAS - self-documenting responses, responses include links that the client can use

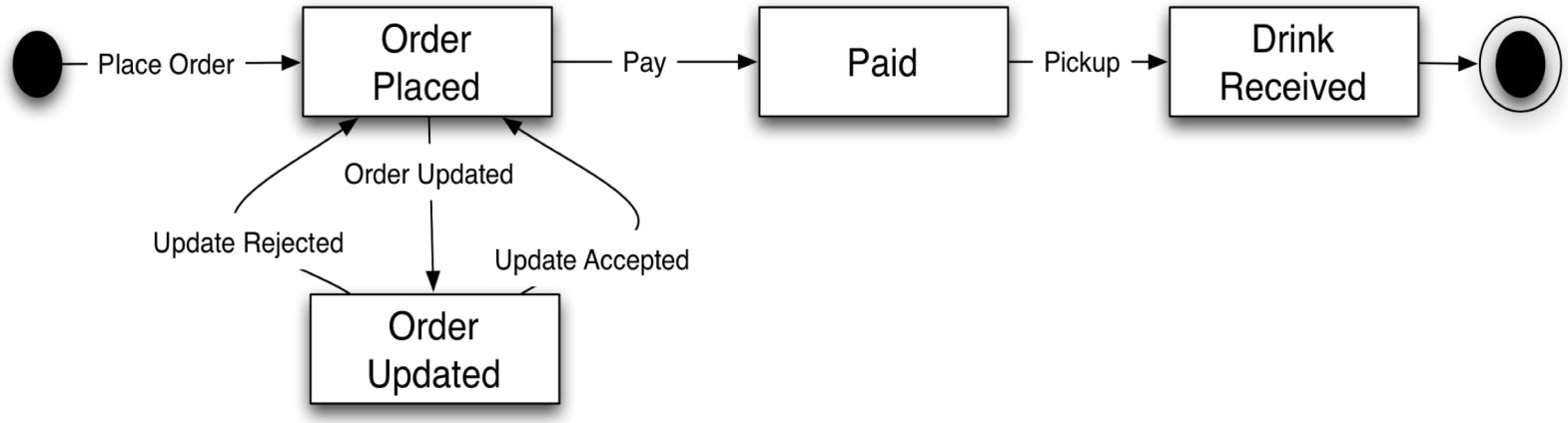
A good read: https://martinfowler.com/articles/richardsonMaturityModel.html

Interacting with RESTful API (workflow)

What would it be like to have a stateless conversation with API vs. stateful conversation with API? (e.g., POST -> return ID, forget vs. POST-> return no ID, plant Cookies)

Take the Coffee Order Process from Jim Webber as example ...

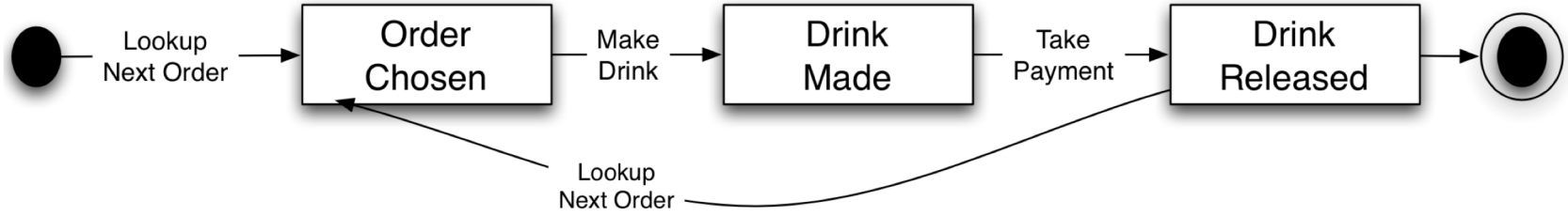
The customer workflow:



customers advance towards the goal of drinking some coffee by interacting with the Starbucks service, the customer orders, pays, and waits for the drink, between ‘order’ and ‘pay’, the customer can update (asking for skimmed milk)

Interacting with RESTful API (workflow)

The barista workflow:



the barista loops around looking for the next order to be made, preparing the drink, and taking the payment,

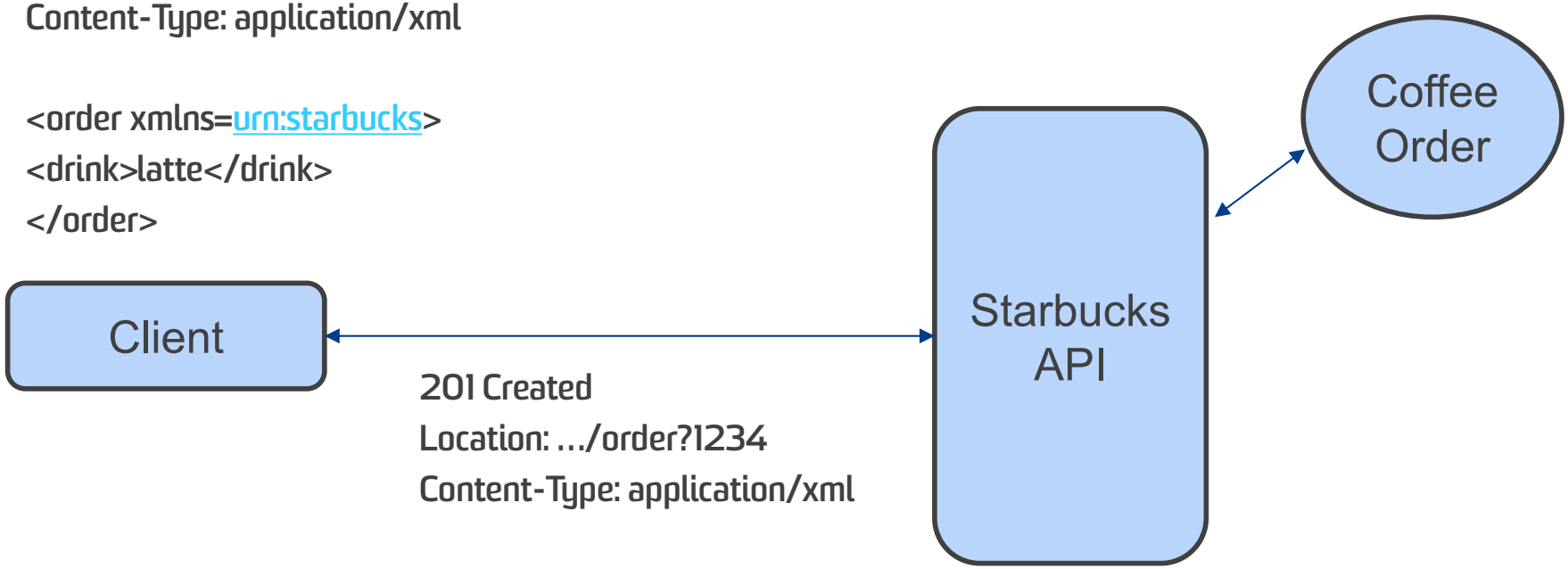
The outputs of the workflow are available to the customer when the barista finishes the order and releases the drink

Points to Remember: We will see how each transition in two state machines is implemented as an interaction with a Web resource. Each transition is the combination of a HTTP verb on a resource via its URI causing state changes.

Customer's View Point

Place an order: POST-ing on http://api.starbucks.com/orders

```
POST /orders HTTP/1.1
Host: xxx
Content-Type: application/xml

<order xmlns=urn:starbucks>
<drink>latte</drink>
</order>
```

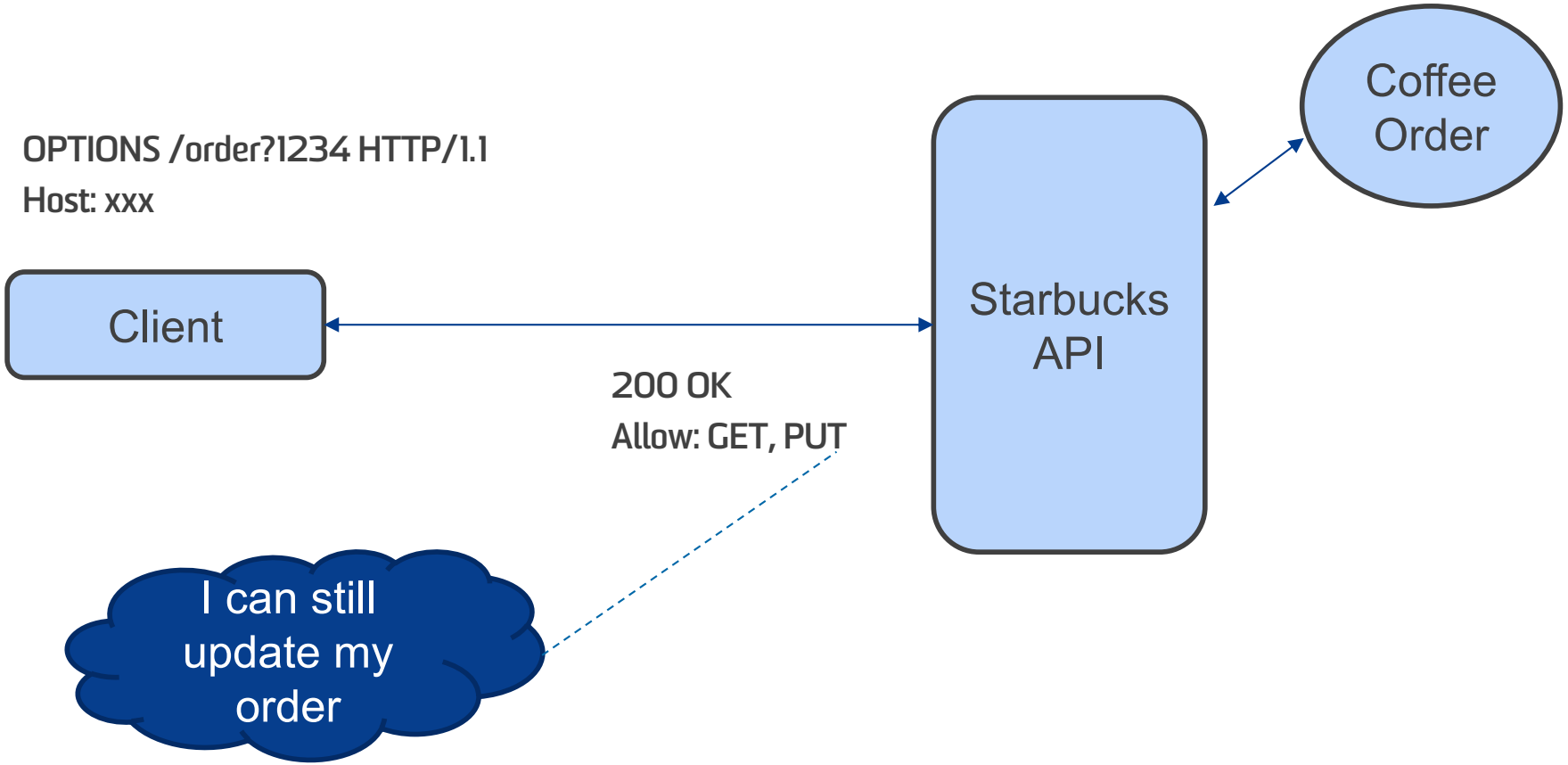



```
201 Created
Location: .../order?1234
Content-Type: application/xml

<order xmlns=urn:starbucks>
<drink>latte</drink>
<link rel="payment" href="".../payment/order?1234>
</order>
```

Oops ... A mistake!

I like my coffee to be strong

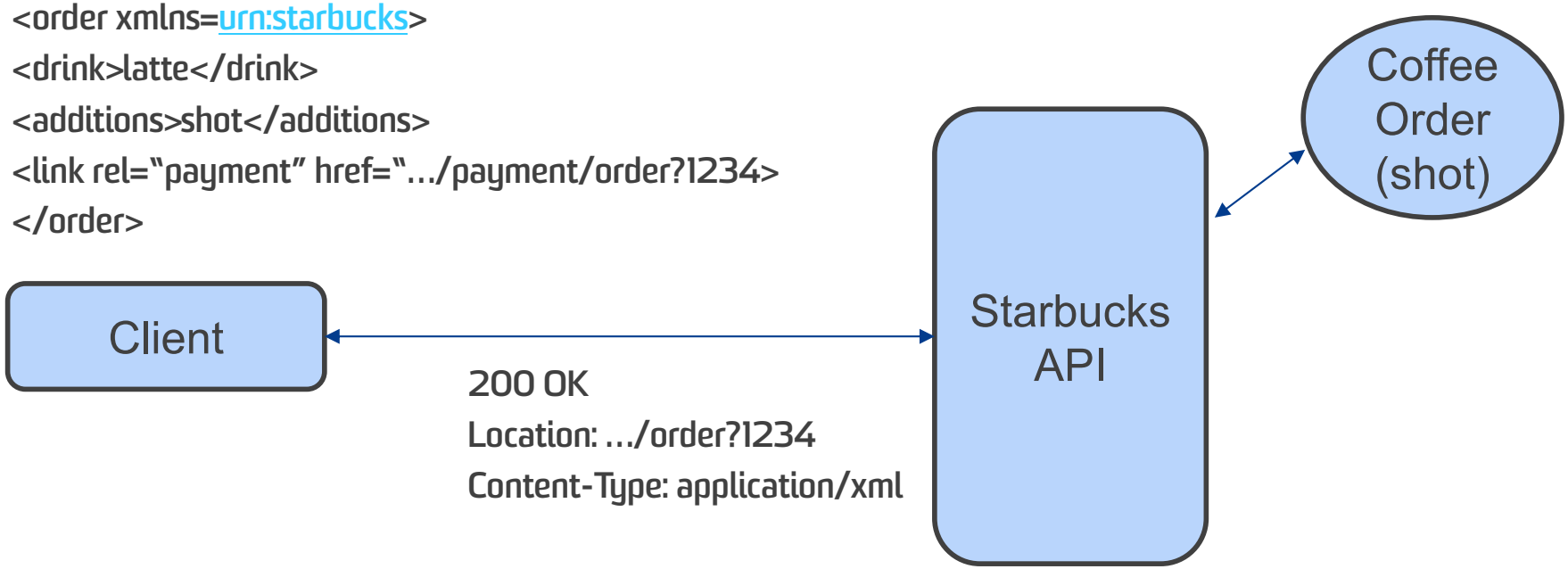
Need another shot of espresso, what are my OPTIONS?



Update the order

PUT /order?1234 HTTP/1.1
Host: xxx
Content-Type: application/xml

<order xmlns=urn:starbucks>
<drink>latte</drink>
<additions>shot</additions>
<link rel="payment" href="…/payment/order?1234>
</order>



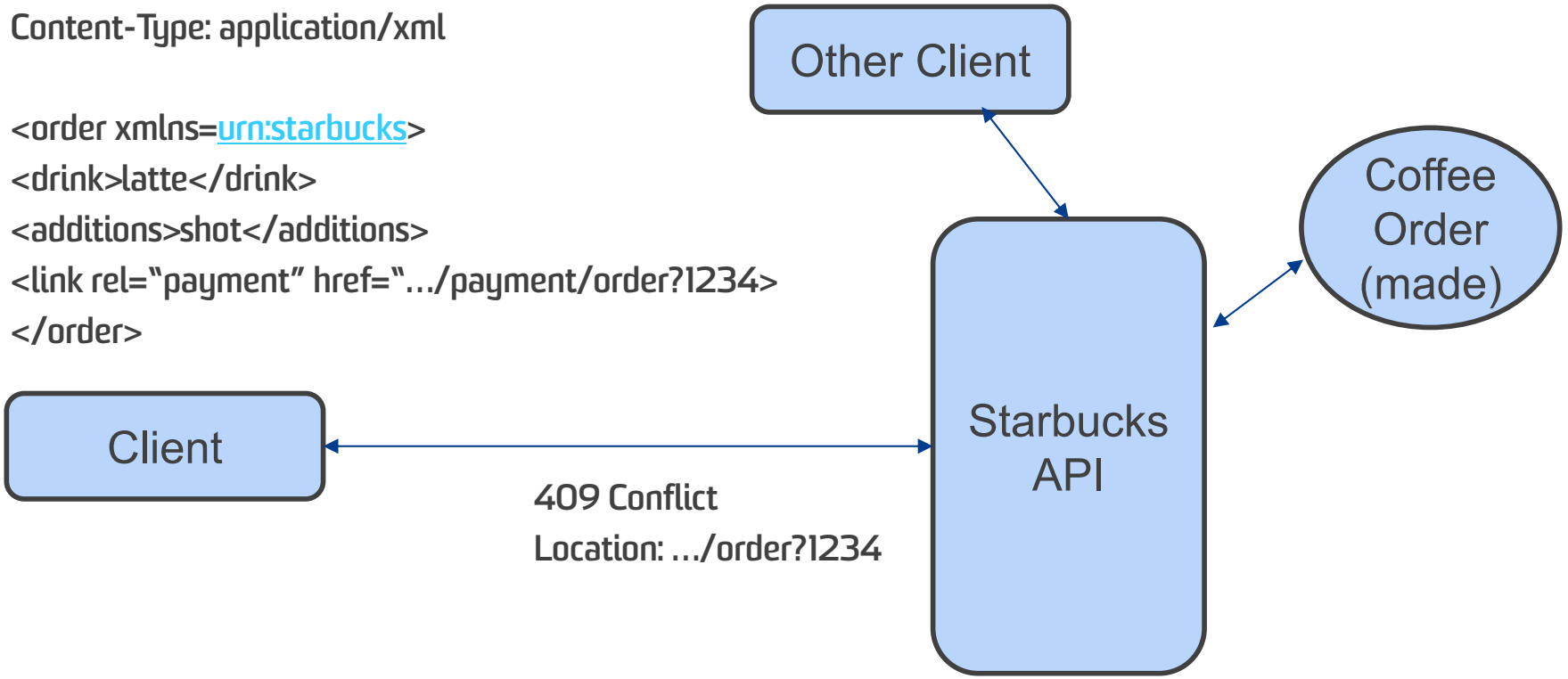
200 OK
Location: …/order?1234
Content-Type: application/xml

<order xmlns=urn:starbucks>
<drink>latte</drink>
<additions>shot</additions>
<link rel="payment" href="…/payment/order?1234>
</order>

Possible conflict with another workflow

The resource state can change without you … (before your PUT-ing getting to the server)

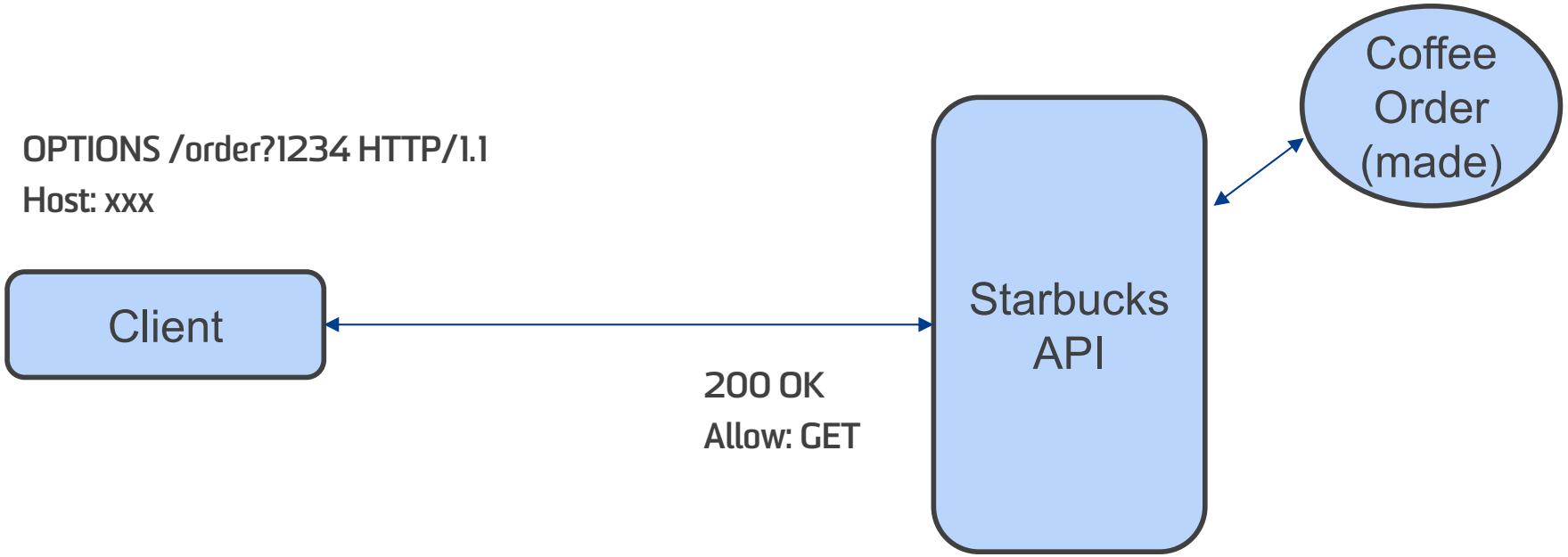
```
PUT /order?1234 HTTP/1.1
Host: xxx
Content-Type: application/xml

<order xmlns=urn:starbucks>
<drink>latte</drink>
<additions>shot</additions>
<link rel="payment" href="…/payment/order?1234>
</order>
```



Possible conflict with another workflow

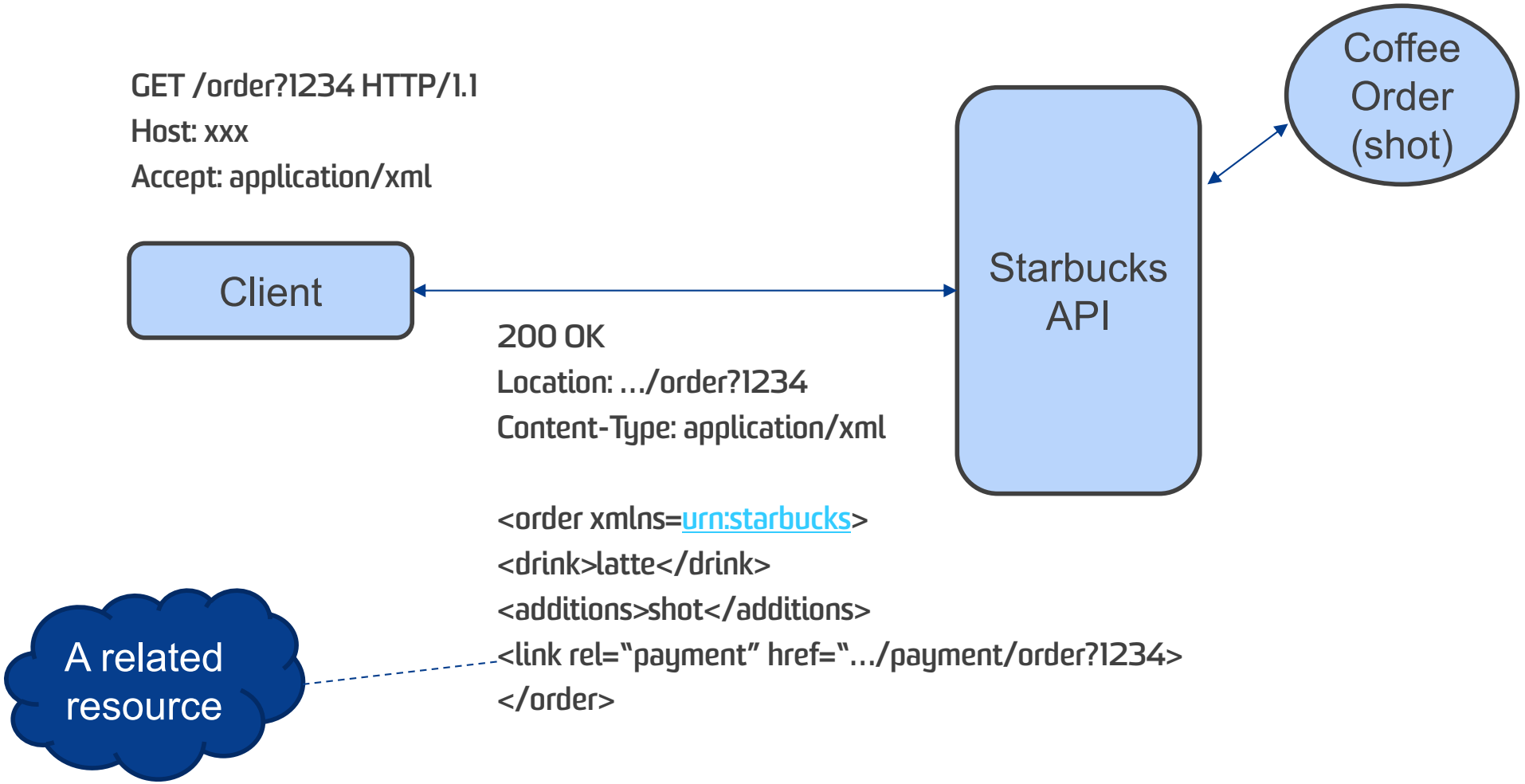
What are my OPTIONS now?

How do I recover?



OK, update successful, what now?

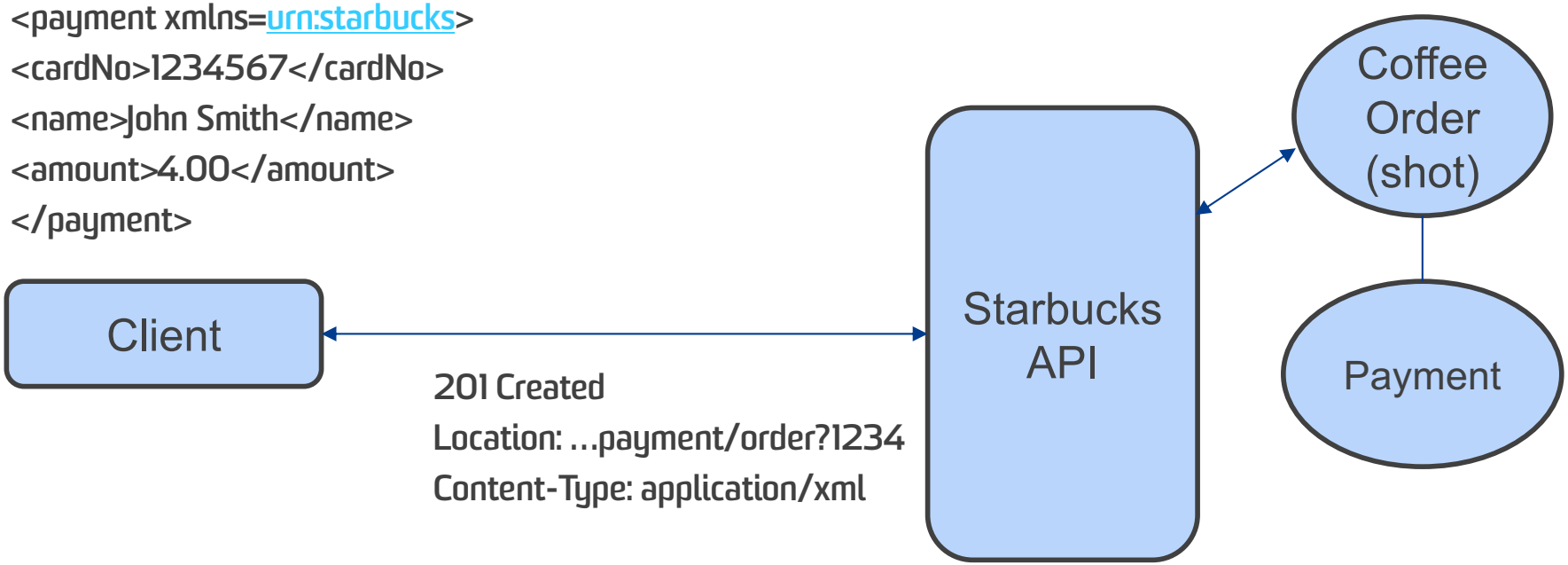
Idea floated here is … FOLLOW THE LINK.



Pay for the order (PUT = idempotent)

PUT /payment/order?1234 HTTP/1.1
Host: xxx
Content-Type: application/xml

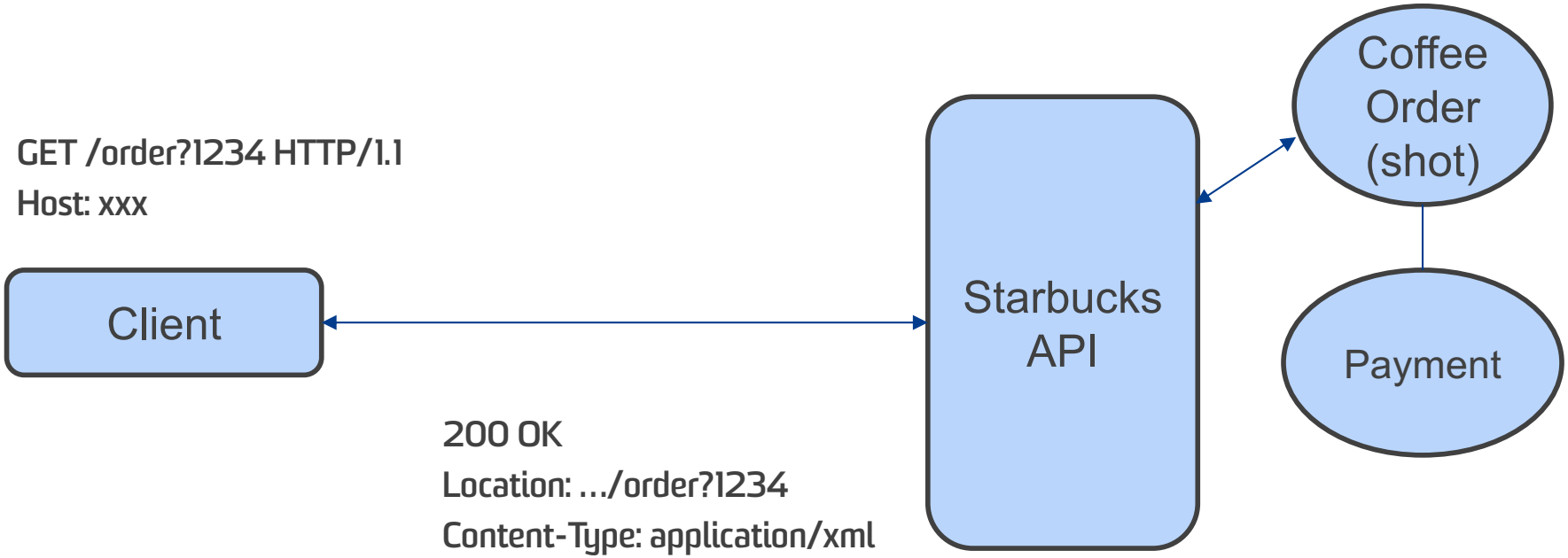
<payment xmlns=urn:starbucks>
<cardNo>1234567</cardNo>
<name>John Smith</name>
<amount>4.00</amount>
</payment>



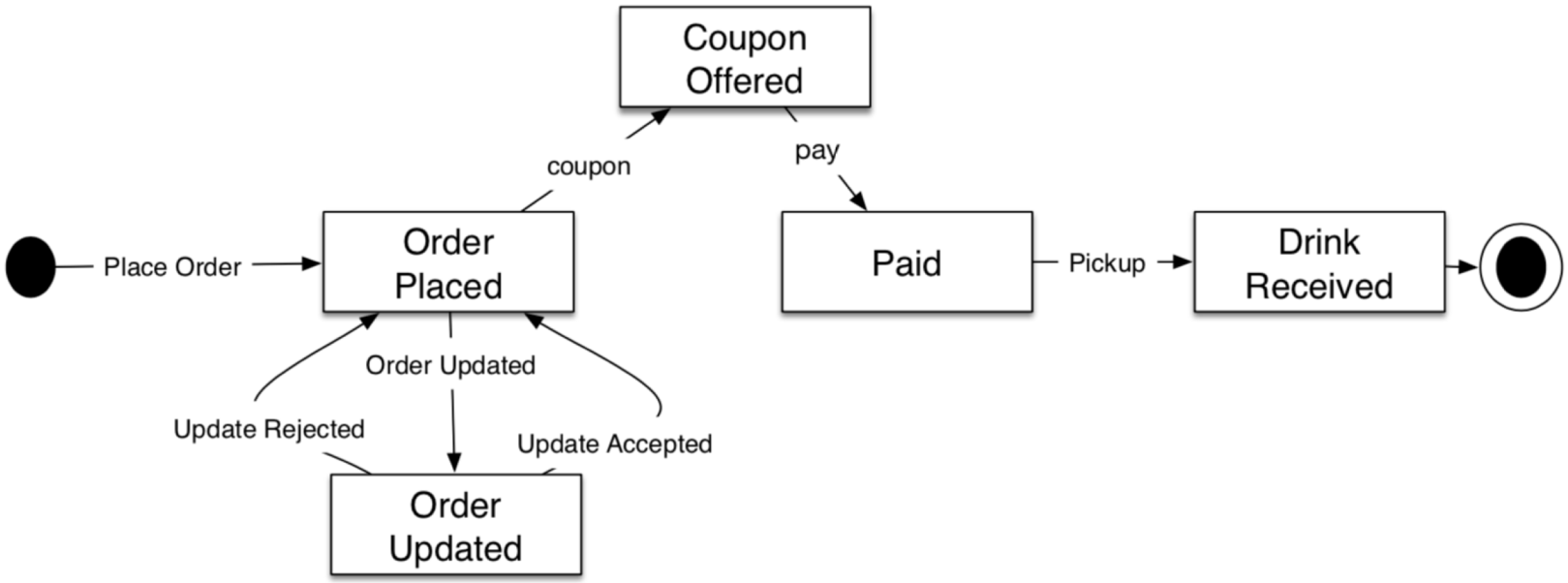
201 Created
Location: …payment/order?1234
Content-Type: application/xml

<payment xmlns=urn:starbucks>
<cardNo>1234567</cardNo>
<name>John Smith</name>
<amount>4.00</amount>
</payment>

Check that you have paid?



Changing the API conversation?



If the self-describing nature of the workflow (i.e., links) is well-respected, the client should not be surprised by the changes !

Non-blocking request/response REST APIs?

HTTP is synchronous by nature ...

How could we design an asynchronous request/response using REST actions (e.g., when a POST will take a long time to complete)?