REST in Practice

A Tutorial on Web-based Services

Jim Webber

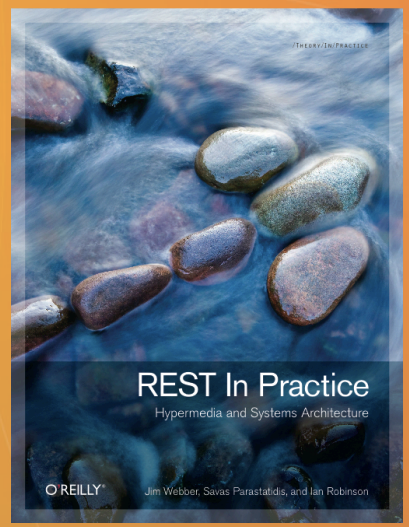
http://jim.webber.name

Savas Parastatidis

http://savas.me

Ian Robinson

http://iansrobinson.com



Motivation

- This follows the plot from a book called *REST in Practice* which is currently being written by:
  - Jim Webber
  - Savas Parastatidis
  - Ian Robinson
- With help from lots of other lovely people like:
  - Halvard Skogsrud, Lasse Westh-Nielsen, Steve Vinoski, Mark Nottingham, Colin Jack, Spiros Tzavellas, Glen Ford, Sriram Narayan, Ken Kolchier, Guilherme Silveira and many more!
- The book deals with the Web as a distributed computing platform
  - The Web as a whole, not just REST
- And so does this tutorial…

Timetable

- 09:00-10:30
  - Web basics
  - URI templates
  - RPC
  - HTTP
  - CRUD
- 10:30-10:45
  - Coffee break
- 10:45-12:00
  - Semantics
  - Hypermedia formats
  - Restbucks DAP
  - Restbucks retrospective
- 12:00-13:00
  - Lunch
- 13:00-14:30
  - Hypermedia design
  - Scalability
  - Security
- 14:30-14:45
  - Coffee break
- 14:45-16:00
  - Atom
  - AtomPub
  - Epilogue

Introduction

- This is a tutorial about the Web
- It's very HTTP centric
- But it's not about Web pages!
- The Web is a middleware platform which is…
  - Globally deployed
  - Has reach
  - Is mature
  - And is a reality in every part of our lives
- Which makes it interesting for distributed systems geeks

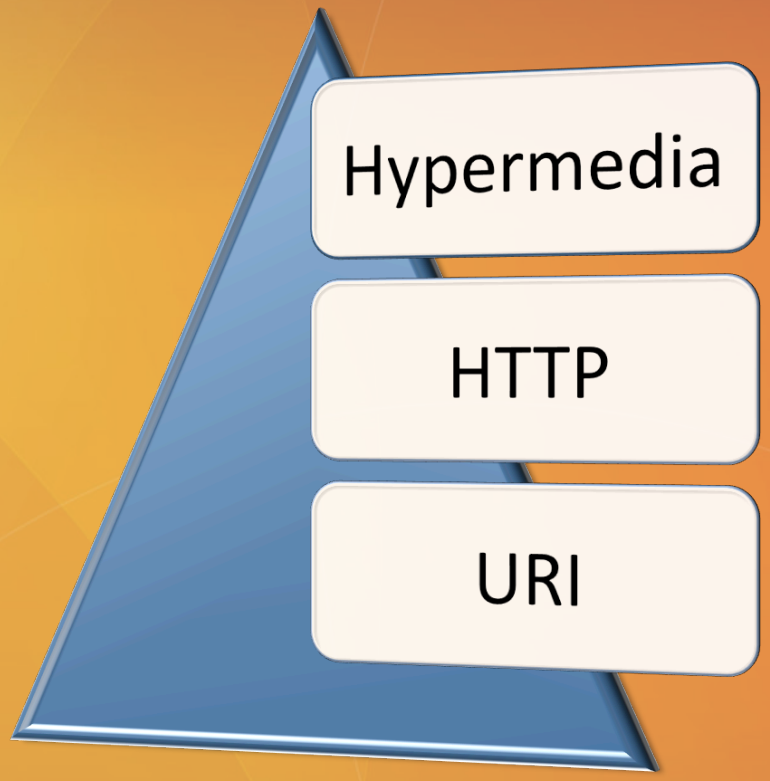
Leonard Richardson's Web service maturity heuristic

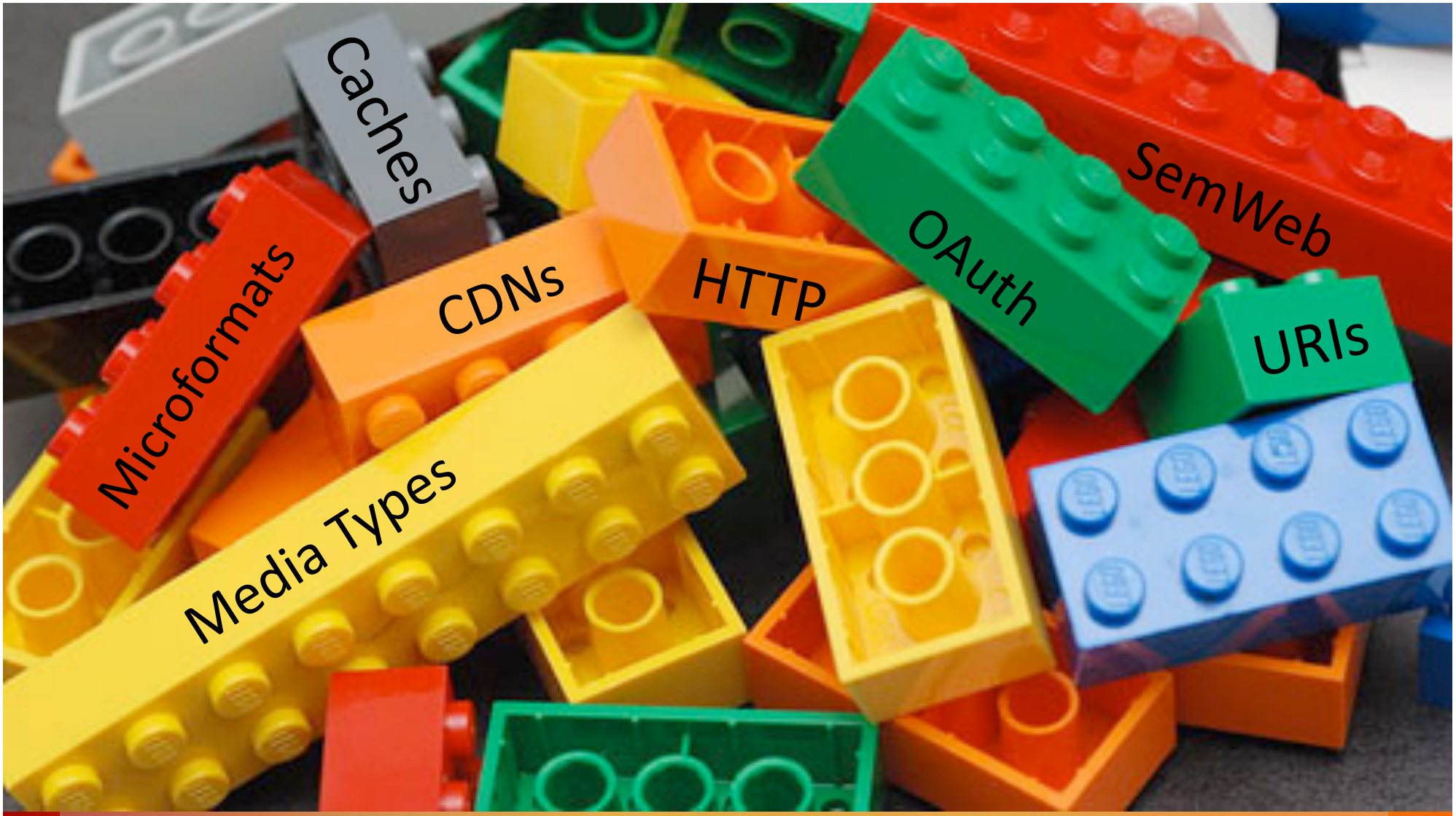
What?	Why?	How?
Divide and conquer	Spreads complexity around	URIs
Refactor (Do the same things in the same way)	Reduces complexity	Uniform interface
Describe special behaviour in a standard way	Makes complexity learnable	Hypermedia

http://www.crummy.com/writing/

Why Web? Why not just REST?

- REST is a brilliant architectural style
- But the Web allows for more than just RESTful systems
- There's a spectrum of maturity of service styles
  - From completely bonkers to completely RESTful
- We'll use the Richardson maturity model to frame these kinds of discussions
  - Level 0 to Level 3
  - Web-ignorant to RESTful!





REST is an Architectural style suited to the Web…

…but it is not mandatory

Why the Web? Why be RESTful?

- Scalable
- Fault-tolerant
- Recoverable
- Secure
- Loosely coupled

- Precisely the same characteristics we want in business software systems!

REST isn't…

- Just pretty URIs
- Just XML or JSON
- URI templates
- AJAX applications

But you could be forgiven for thinking it is!

Web Architecture

Web History

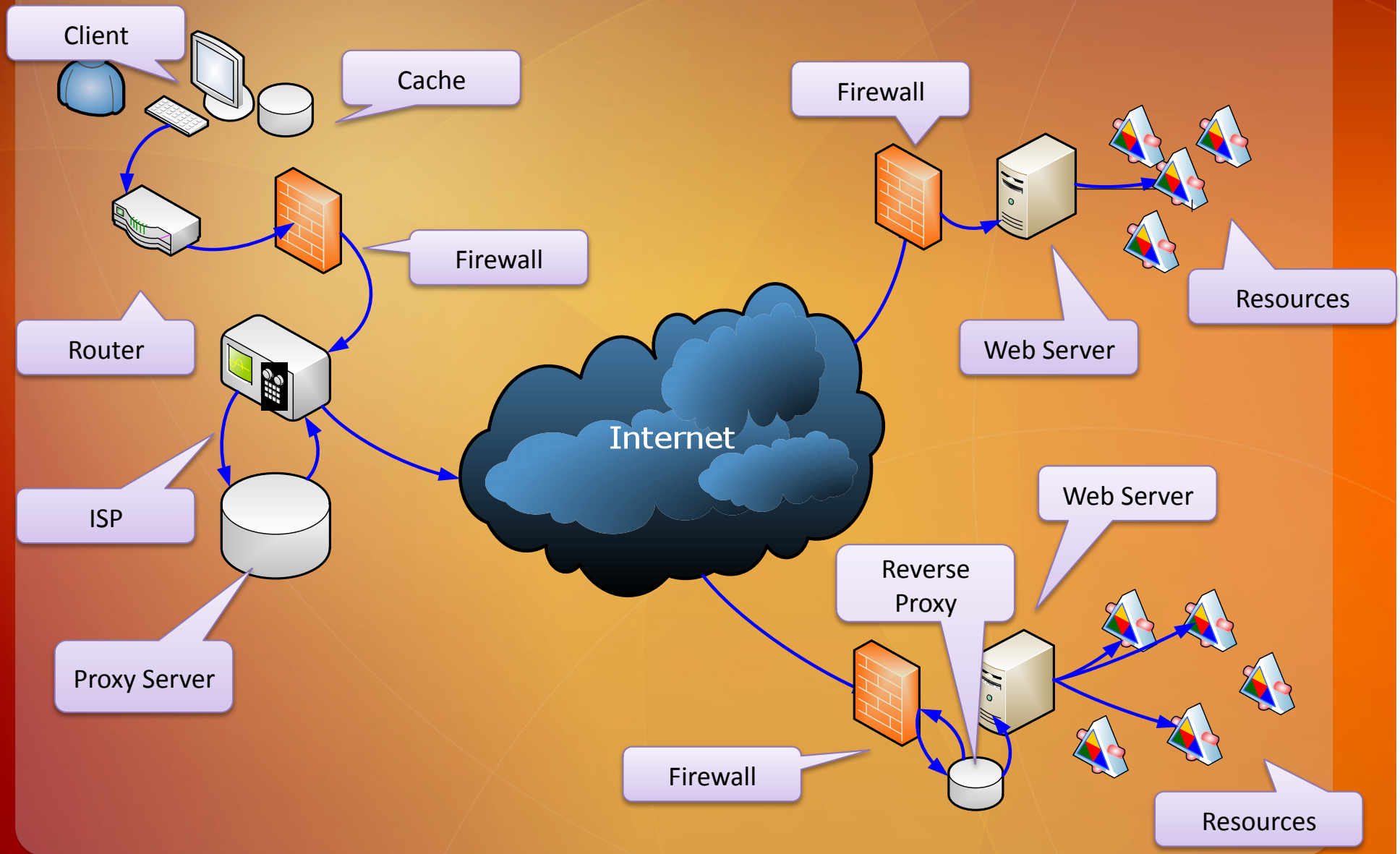
- Started as a distributed hypermedia platform
  - CERN, Berners-Lee, 1990
- Revolutionised hypermedia
  - Imagine emailing someone a hypermedia deck nowadays!
- Architecture of the Web largely fortuitous
  - W3C and others have since retrofitted/captured the Web's architectural characteristics

The Web broke the rules

Web Fundamentals

- To embrace the Web, we need to understand how it works
- The Web is a distributed hypermedia model
  - It doesn't try to hide that distribution from you!
- Our challenge:
  - Figure out the mapping between our problem domain and the underlying Web platform

Key Actors in the Web Architecture



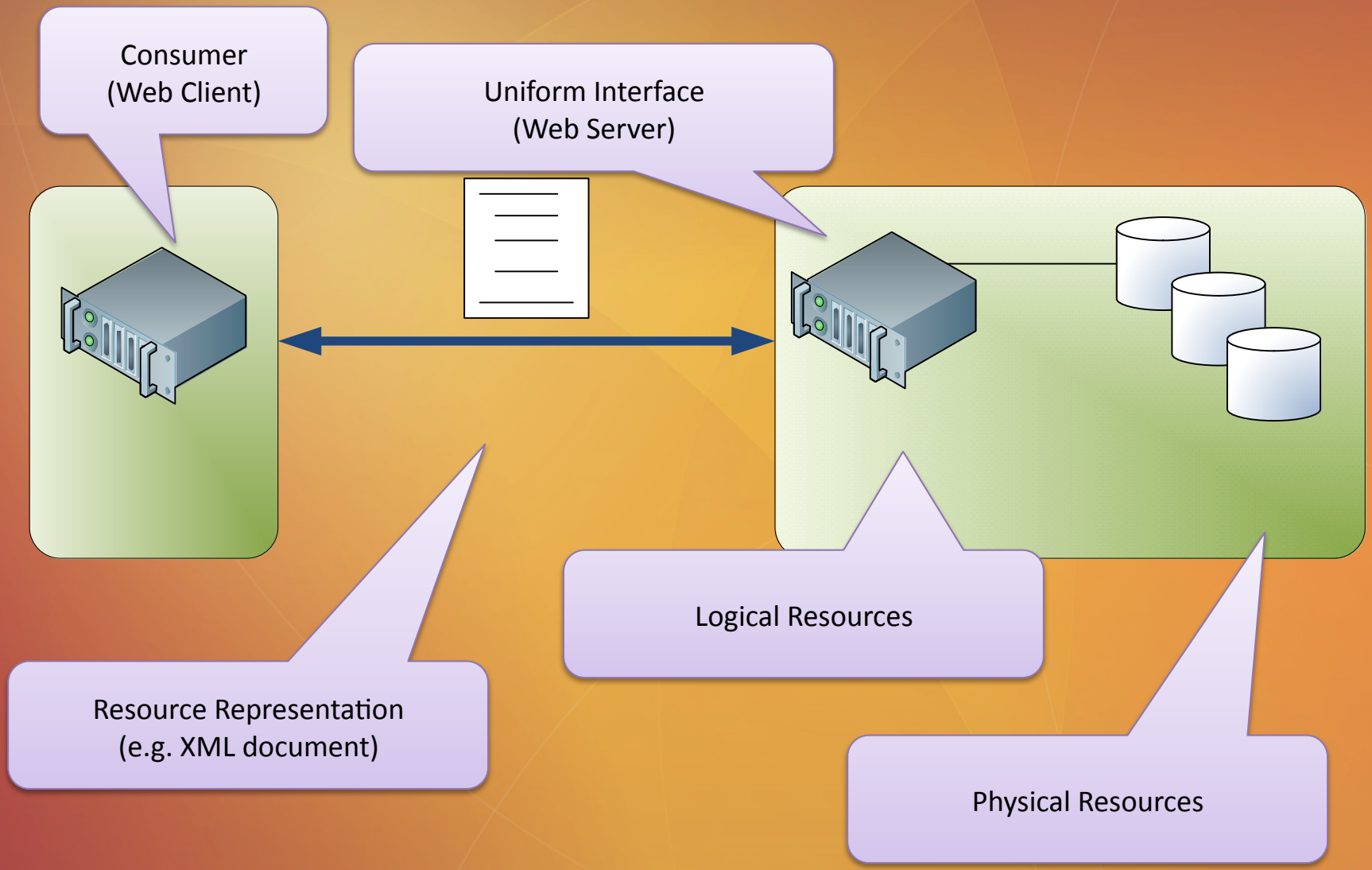
Resources

- A resource is something “interesting” in your system
- Can be anything
  - Spreadsheet (or one of its cells)
  - Blog posting
  - Printer
  - Winning lottery numbers
  - A transaction
  - Others?

Interacting with Resources

- We deal with representations of resources
  - Not the resources themselves
    - "Pass-by-value" semantics
  - Representation can be in any format
    - Any media type
- Each resource implements a standard uniform interface
  - Typically the HTTP interface
- Resources have names and addresses (URIs)
  - Typically HTTP URIs (aka URLs)

Resource Architecture



Resource Representations

- Making your system Web-friendly increases its surface area
  - You expose many resources, rather than fewer endpoints
- Each resource has one or more representations
  - Representations like JSON or XML or good for the programmatic Web
- Moving representations across the network is the way we transact work in a Web-native system

URIs

- URIs are addresses of resources in Web-based systems
  - Each resource has at least one URI
- They identify “interesting” things
  - i.e. Resources ☺
- Any resource implements the same (uniform) interface
  - Which means we can access it programmatically!
- Declarative scheme

URI/Resource Relationship

- Any two resources cannot be identical
  - Because then you've only got one resource!
- But they can have more than one name
  - http://foo.com/software/latest
  - http://foo.com/software/v1.4
- No mechanism for URI equality
- Canonical URIs are long-lived
  - E.g. http://example.com/versions/1.1 versus http://example.com/versions/latest
  - Send back HTTP 303 ("see also") if the request is for an alternate URI
  - Or set the Content-Location header in the response

Scalability

- Web is truly Internet-scale
  - Loose coupling
    - Growth of the Web in one place is not impacted by changes in other places
  - Uniform interface
    - HTTP defines a standard interface for all actors on the Web
    - Replication and caching is baked into this model
      - Caches have the same interface as real resources!
  - Stateless model
    - Supports horizontal scaling

Fault Tolerant

- The Web is stateless
  - All information required to process a request must be present in that request
    - Sessions are still plausible, but must be handled in a Web-consistent manner
      - Modelled as resources!
- Statelessness means easy replication
  - One Web server is replaceable with another
  - Easy fail-over, horizontal scaling

Recoverable

- The Web places emphasis on repeatable information retrieval
  - GET is idempotent
    - Library of Congress found this the hard way!
  - In failure cases, can safely repeat GET on resources
- HTTP verbs plus rich error handling help to remove guesswork from recovery
  - HTTP statuses tell you what happened!
  - Some verbs (e.g. PUT, DELETE) are safe to repeat

Secure

- HTTPs is a mature technology
  - Based on SSL for secure point-to-point information retrieval
- Isn't sympathetic to Web architecture
  - Restricted caching opportunities
- Higher-order protocols like Atom are starting to change this...
  - Encrypt parts of a resource representation, not the transport channel
  - OK to cache!

Loosely Coupled

- Adding a Web site to the WWW does not affect any other existing sites
- All Web actors support the same, uniform interface
  - Easy to plumb new actors into the big wide web
    - Caches, proxies, servers, resources, etc

Tech Interlude

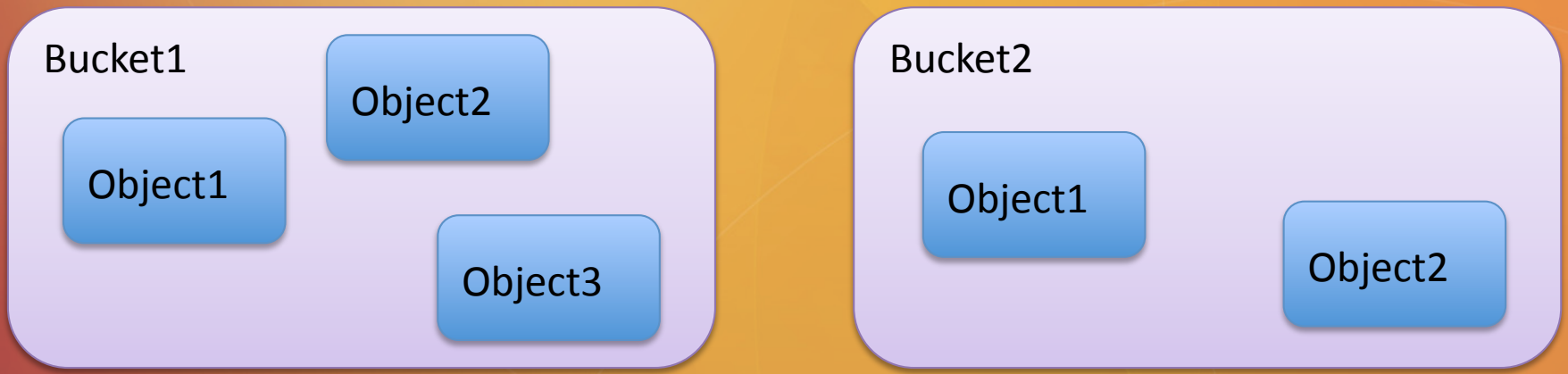
URI Templates

Conflicting URI Philosophies

- URIs should be descriptive, predictable?
  - http://spreadsheet/cells/a2,a9
  - http://jim.webber.name/2007/06.aspx
    - Convey some ideas about how the underlying resources are arranged
      - Can infer http://spreadsheet/cells/b0,b10 and http://jim.webber.name/2005/05.aspx for example
    - Nice for programmatic access, but may introduce coupling
- URIs should be opaque?
  - http://tinyurl.com/6
  - TimBL says "opaque URIs are cool"
    - Convey no semantics, can't infer anything from them
      - Don't introduce coupling

URI Templates, in brief

- Use URI templates to make your resource structure easy to understand
- For Amazon S3 (storage service) it's easy:
  - `http://s3.amazon.com/{bucket-name}/{object-name}`



URI Templates are Easy!

- Take the URI:

```
http://restbucks.com/orders?{order_id}
```

- You could do the substitution and get a URI:

```
http://restbucks.com/orders?1234
```

- Can easily make more complex URIs too
  - Mixing template and non-template sections

```
http://restbucks.com/{orders}/{shop}/{year}/{month}.xml
```

- Use URI templates client-side to compute server-side URIs
  - But beware this introduces coupling!

Why URI Templates?

- Regular URIs are a good idiom in Web-based services
  - Helps with understanding, self documentation
- They allow users to infer a URI
  - If the pattern is regular
- URI templates formalise this arrangement
  - And advertise a template rather than a regular URI

URI Templates Pros and Cons

- Everything interesting in a Web-based service has a URI
- Remember, two schools of thought:
  - Opaque URIs are cool (Berners-Lee)
  - Transparent URIs are cool (everyone else)
- URI templates present two core concerns:
  - They invite clients to invent URIs which may not be honoured by the server
  - They increase coupling since servers must honour forever any URI templates they've advertised
- Use URI templates sparingly, and with caution
  - Entry point URIs only is a good rule of thumb

RPC Again!

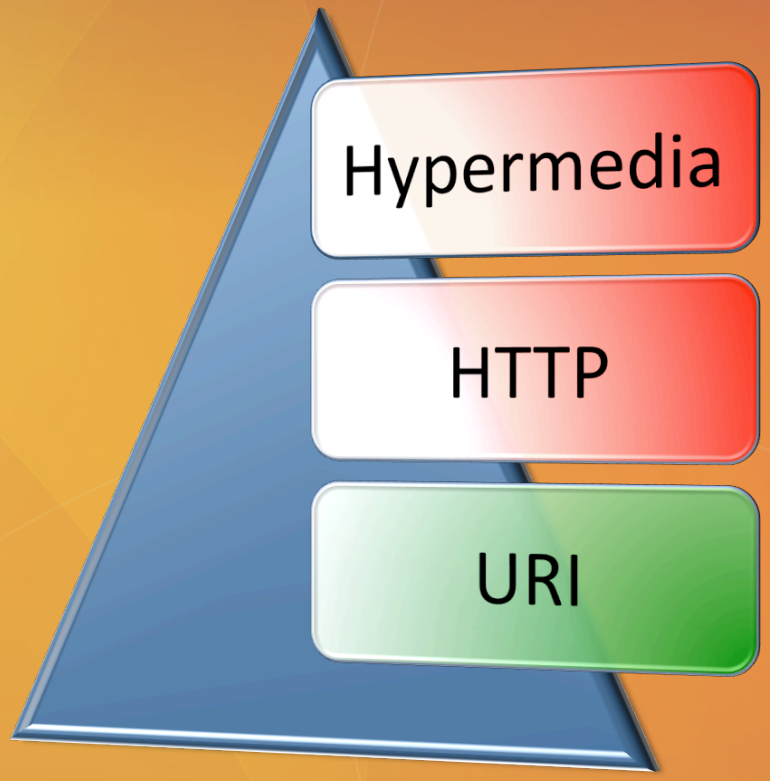
Web Tunnelling

- Web Services tunnel SOAP over HTTP
  - Using the Web as a transport only
  - Ignoring many of the features for robustness the Web has built in
- Many Web people do the same!
  - URI tunnelling, POX approaches are the most popular styles on today's Web
  - Worse than SOAP!
    - Less metadata!

But they claim to be "lightweight" and RESTful

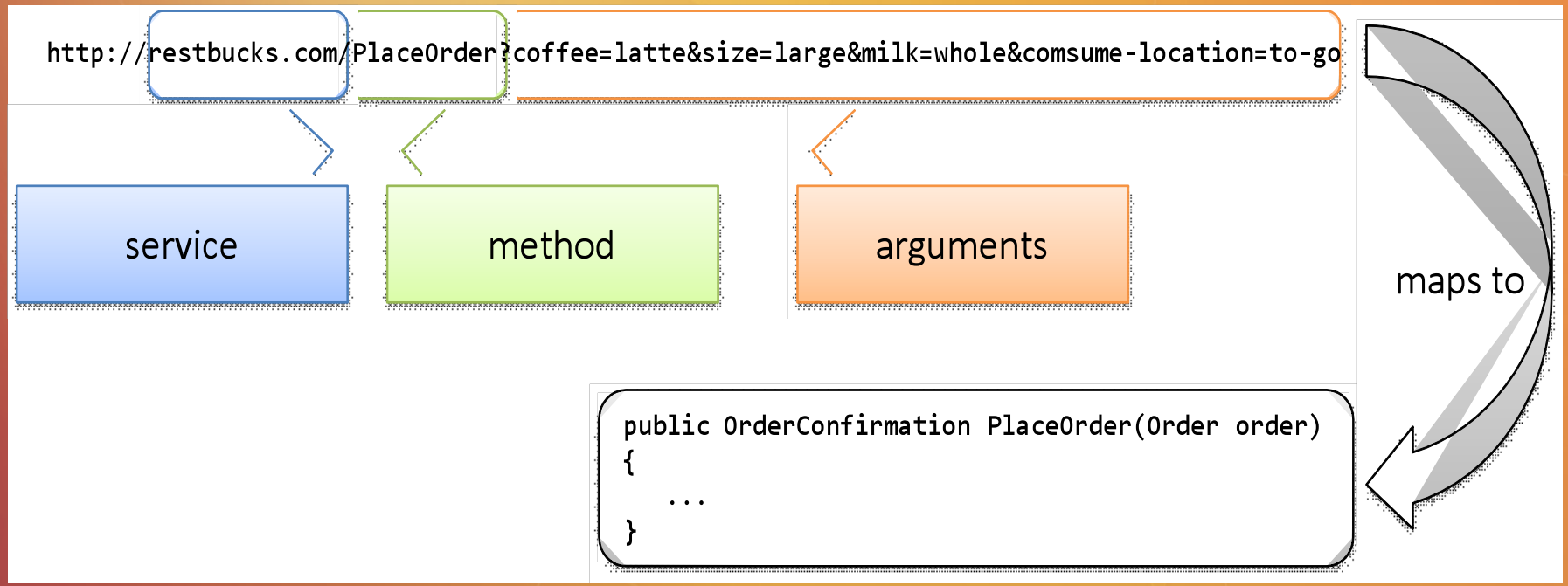
Richardson Model Level 1

- Lots of URIs
  - But really has a more level 0 mindset
- Doesn't understand HTTP
  - Other than as a transport
- No hypermedia



URI Tunnelling Pattern

- Web servers understand URIs
- URIs have structure
- Methods have signatures
- Can match URI structure to method signature



On The Wire

Request

```
GET /PlaceOrder?coffee=latte&size=large&milk=whole&comsume-location=to-go HTTP/1.1
Host: restbucks.com
```

Response

```
HTTP/1.1 200 OK
Content-type: text/plain

OrderId=1234
```

Server-Side URI Tunnelling Example

```
public void ProcessGet(HttpListenerContext context)
{
  // Parse the URI
  Order order = ParseUriForOrderDetails(context.Request.QueryString);

  string response = string.Empty;

  if (order != null)
  {
     // Process the order by calling the mapped method
    var orderConfirmation = RestbucksService.PlaceOrder(order);

    response = "OrderId=" + orderConfirmation.OrderId.ToString();
  }
  else
  {
    response = "Failure: Could not place order.";
  }

  // Write to the response stream
  using (var sw = new StreamWriter(context.Response.OutputStream))
  {
    sw.Write(response);
  }
}
```

Client-Side URI Tunnelling

```
public OrderConfirmation PlaceOrder(Order order)
{
  // Create the URI
  var sb = new StringBuilder("http://restbucks.com/PlaceOrder?");

  sb.AppendFormat("coffee={0}", order.Coffee.ToString());
  sb.AppendFormat("&size={0}", order.Size.ToString());
  sb.AppendFormat("&milk={0}", order.Milk.ToString());
  sb.AppendFormat("&consume-location={0}", order.ConsumeLocation.ToString());

  // Set up the GET request
  var request = HttpRequest.Create(sb.ToString()) as HttpWebRequest;
  request.Method = "GET";

  // Get the response
  var response = request.GetResponse();

  // Read the contents of the response
  OrderConfirmation orderConfirmation = null;
  using (var sr = new StreamReader(response.GetResponseStream()))
  {
    var str = sr.ReadToEnd();

    // Create an OrderConfirmation object from the response
    orderConfirmation = new OrderConfirmation(str);
  }
  return orderConfirmation;
}
```

URI Tunnelling Strengths

- Very easy to understand
- Great for simple procedure-calls
- Simple to code
  - Do it with the servlet API, HttpListener, IHttpHandler, Rails, whatever!
- Interoperable
  - It's just URIs!

URI Tunnelling Weaknesses

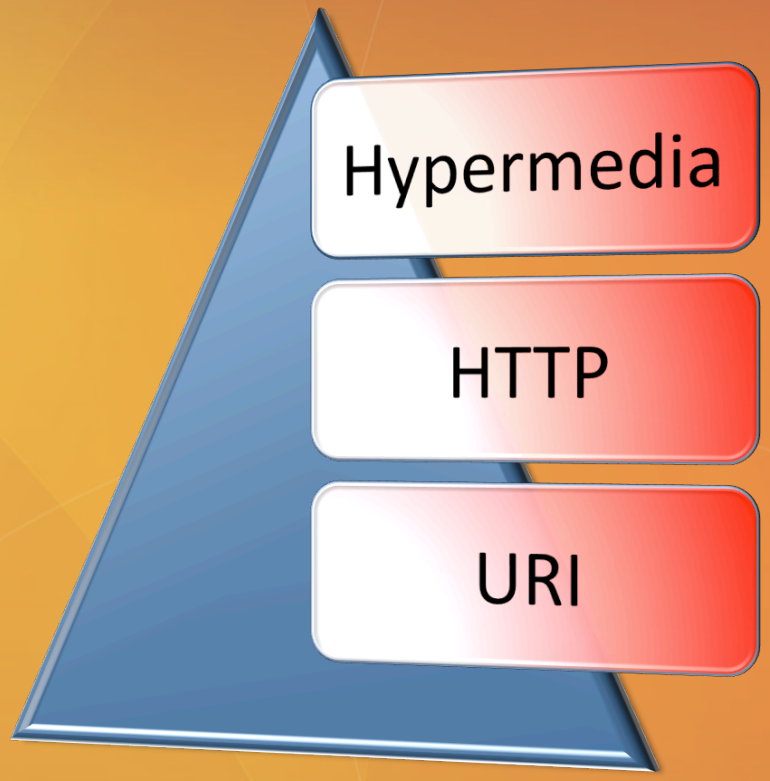
- It's brittle RPC!
- Tight coupling, no metadata
  - No typing or "return values" specified in the URI
- Not robust – have to handle failure cases manually
- No metadata support
  - Construct the URIs yourself, map them to the function manually
- You typically use GET (prefer POST)
  - OK for functions, but against the Web for procedures with side-affects

POX Pattern

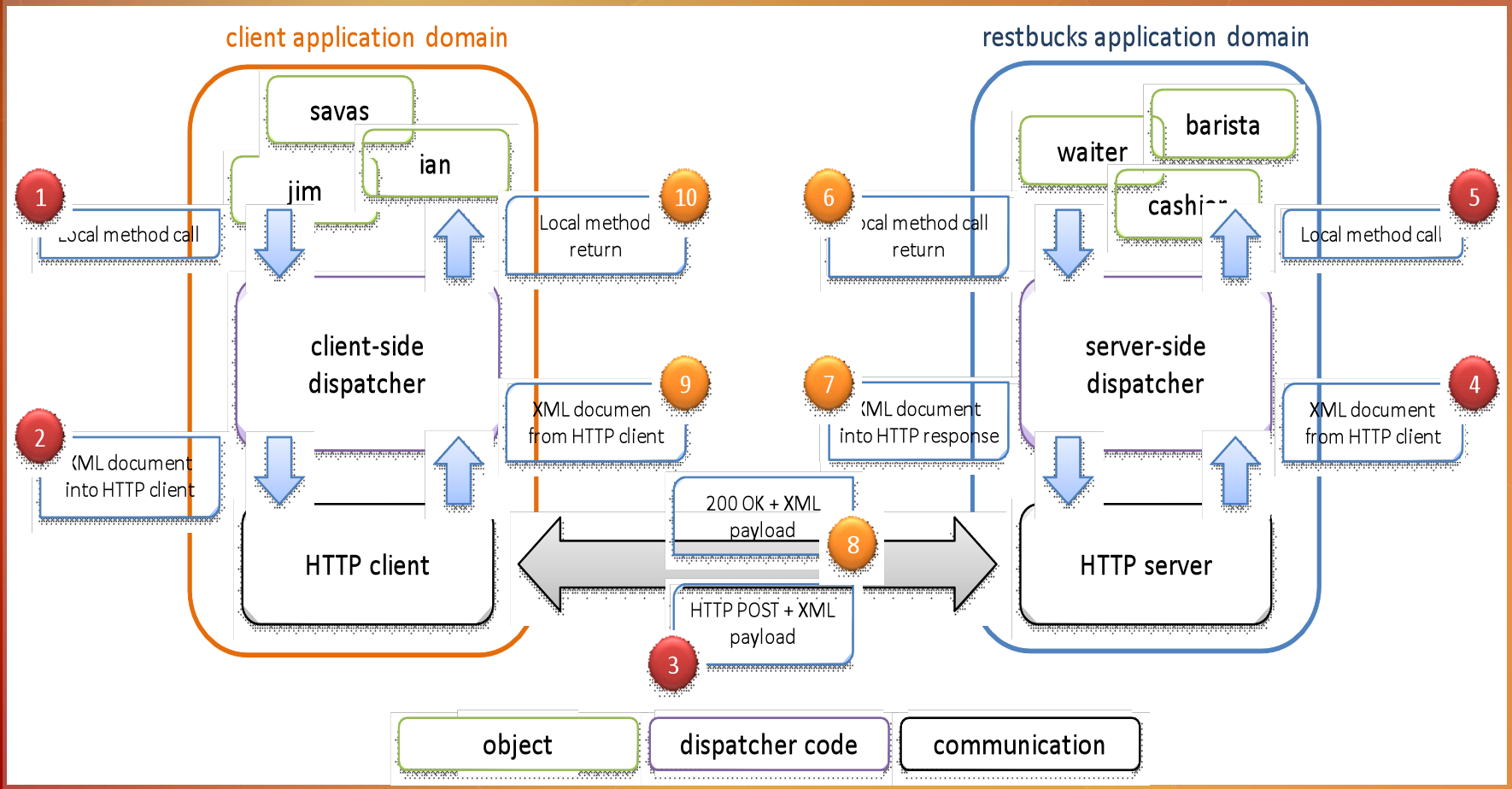
- Web servers understand how to process requests with bodies
  - Because they understand forms
- And how to respond with a body
  - Because that's how the Web works
- POX uses XML in the HTTP request and response to move a call stack between client and server

Richardson Model Level 0

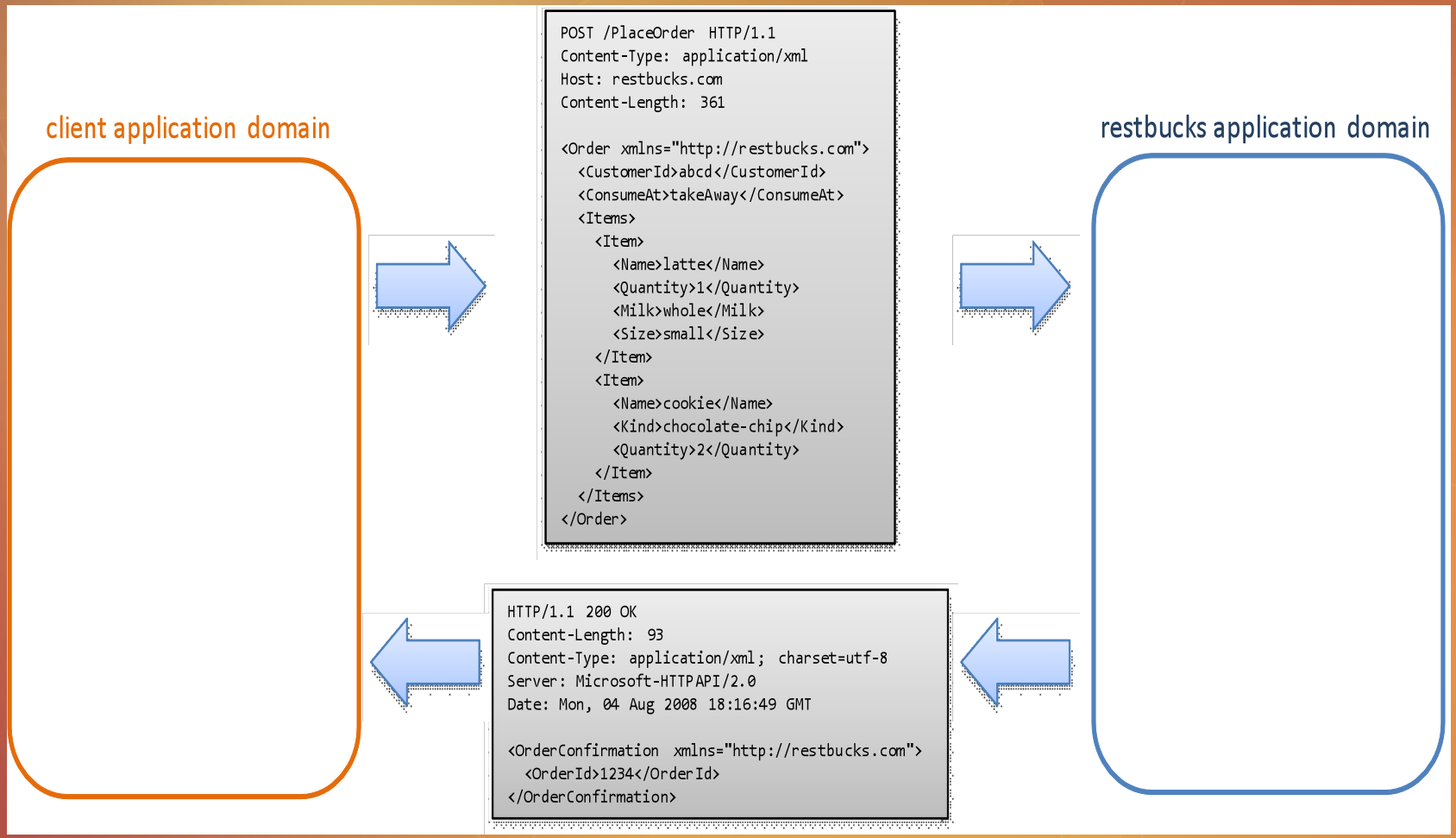
- Single well-known endpoint
  - Not really URI friendly
- Doesn't understand HTTP
  - Other than as a transport
- No hypermedia



POX Architecture



POX on the Wire



.Net POX Service Example

```
private void ProcessRequest(HttpListenerCont
{
  string verb = context.Request.HttpMethod.ToLower().Trim();
  switch (verb)
  {
    case "post'
    {
      // Everyt                    ost in this case
      XmlDocument request = new XmlDocument();
      request.Load(XmlReader.Create
   (context.Request.InputStream));



      XmlElement result = MyApp
   (request.Documen tElement);
      byte[] retu   Value =
        Utils.Co   rtUnicodeString
   (Constants.X   ML_DECLARATION +
          res     terXml);

       cor                    eam.Write(returnValue, 0,
       ret

       break;
  } ...
```

From the Web server

Check HTTP Verb (we want POST)

Dispatch it for processing

XML document for processing

Get XML result, and get bytes

Return XML bytes to client

Java POX Servlet

```
public class RestbucksService extends HttpServlet {

@Override
protected void doPost(HttpServletRequest request,
                      HttpServletResponse response)
                      throws ServletException, IOException {

    // Initialization code omitted for brevity
    try {
        requestReader = request.getReader();
        responseWriter = response.getWriter();

        String xmlRequest = extractPayload(requestReader);

        Order order = createOrder(xmlRequest);

        OrderConfirmation confirmation = restbucksService.placeOrder(order);

        embedPayload(requestWriter, confirmation.toString());

    } finally {
      // Cleanup code omitted for brevity
    }
}
```

C# POX Client Example

```
public OrderConfirmation PlaceOrder(string customerId, Item[] items)
{
  // Serialize our objects
  XmlDocument requestXml = CreateXmlRequest(customerId, items);
  var client = new WebClient();

  var ms = new MemoryStream();
  requestXml.Save(ms);

  client.Headers.Add("Content-Type", "application/xml");

  ms = new MemoryStream(client.UploadData("http://restbucks.com/
   PlaceOrder", null, ms.ToArray()));

  var responseXml = new XmlDocument();
  responseXml.Load(ms);
  return CreateOrderConfirmation(responseXml);
}
```

Java Apache Commons Client

```
public class OrderingClient {

  private static final String XML_HEADING = "<?xml
     version=\"1.0\"?>\n";
  private static final String NO_RESPONSE =
     "Error: No response.";

  public String placeOrder(String customerId,
     String[] itemIds)
    throws Exception {

    // XML string creation omitted for brevity
    // ...

    String response = sendRequestPost(request,
     "http://restbucks.com/PlaceOrder");

    Document xmlResponse =
     DocumentBuilderFactory.newInstance()
                           .newDocumentBuilder
     ().parse(
                            new InputSource(new
     StringReader(response)));

    // XML response handling omitted for brevity
  }
```

```
  private String sendRequestPost(String
     request, String uri)
                                      throws
     IOException, HttpException {
     PostMethod method = new PostMethod
     (uri);
     method.setRequestHeader("Content-
     type", "application/xml");
     method.setRequestBody(XML_HEADING +
     request);
     String responseBody = NO_RESPONSE;
     try {
       new HttpClient().executeMethod
     (method);
       responseBody = new String
     (method.getResponseBody(), "UTF-8");
     } finally {
       method.releaseConnection();
     }
     return responseBody;
   }
}
```

POX Strengths

- Simplicity – just use HTTP POST and XML
- Re-use existing infrastructure and libraries
- Interoperable
  - It's just XML and HTTP
- Can use complex data structures
  - By encoding them in XML

POX Weaknesses

- Client and server must collude on XML payload
  - Tightly coupled approach
- No metadata support
  - Unless you're using a POX toolkit that supports WSDL with HTTP binding (like WCF)
- Does not use Web for robustness
- Does not use SOAP + WS-* for robustness either

Web Abuse

- Both POX and URI Tunnelling fail to take advantage of the Web
  - Ignoring status codes
  - Reduced scope for caching
  - No metadata
  - Manual crash recovery/compensation leading to high development cost
  - Etc
- They're useful in some situations
  - And you can implement them with minimal toolkit support
  - But they're not especially robust patterns

COMMON SENSE

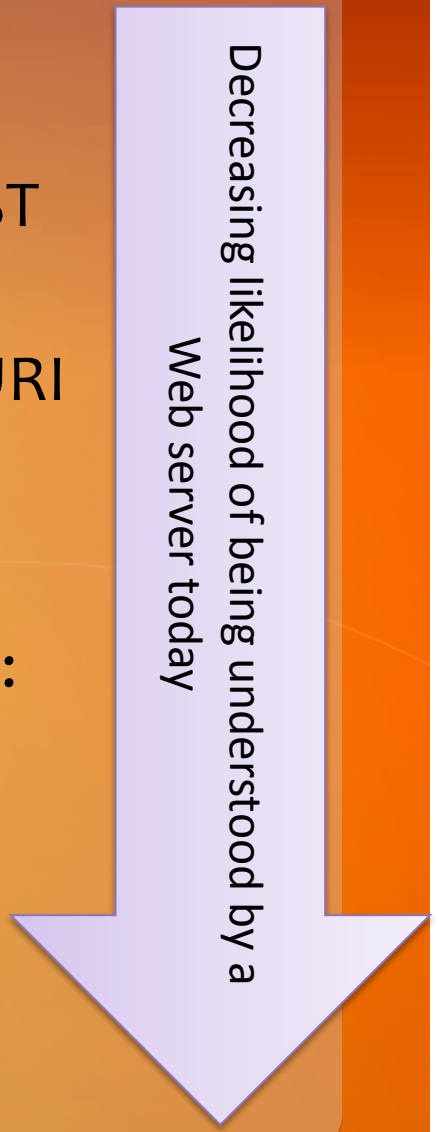
Just because you can, doesn't mean you should.

Tech Interlude

HTTP Fundamentals

The HTTP Verbs

- Retrieve a representation of a resource: GET
- Create a new resource: PUT to a new URI, or POST to an existing URI
- Modify an existing resource: PUT to an existing URI
- Delete an existing resource: DELETE
- Get metadata about an existing resource: HEAD
- See which of the verbs the resource understands: OPTIONS



HEAD Semantics

- HEAD is like GET, except it only retrieves metadata
- Request

```
HEAD /order/1234 HTTP 1.1
Host: restbucks.com
```

- Response

```
200 OK
Content-Type: application/vnd.restbucks
  +xml
Last-Modified: 2007-07-08T15:00:34Z
Etag: aabd653b-65d0-74da-bc63-4bca-
  ba3ef3f50432
```

Useful for caching, performance

OPTIONS Semantics

- Asks which methods are supported by a resource
  - Easy to spot read-only resources for example

- Request

```
OPTIONS /orders/1234 HTTP 1.1
Host: restbucks.com
```

- Response

```
200 OK
Allow: GET,HEAD,POST
```

You can only read and add to this resource, may change over time

HTTP Status Codes

- The HTTP status codes provide metadata about the state of resources
- They are part of what makes the Web a rich platform for building distributed systems
- They cover five broad categories
  - 1xx - Metadata
  - 2xx – Everything's fine
  - 3xx – Redirection
  - 4xx – Client did something wrong
  - 5xx – Server did a bad thing
- There are a handful of these codes that we need to know in more detail

1xx

- 100 – Continue
  - The operation will be accepted by the service
  - The “look before you leap” pattern
    - Use with the Expect header
- Request

```
POST /orders HTTP/1.1
Content-Type: application/xml
Expect: 100-continue
```

- Response

```
100 Continue
```

or

```
417 Expectation Failed
```

2xx

- 200 – OK
  - The server successfully completed whatever the client asked of it
- 201 – Created
  - Sent when a new resource is created at the client's request via POST
  - `Location` header should contain the URI to the newly created resource
- 202 – Accepted
  - Client's request can't be handled in a timely manner
  - `Location` header should contain a URI to the resource that will eventually be exposed to fulfil the client's expectations

More 2xx Codes

- 203 – Non-Authoritative Information
  - Much like 200, except the client knows not to place full trust in any headers since they could have come from 3rd parties or be cached etc.
- 204 – No Content
  - The server declines to send back a representation
    - Perhaps because the associated resource doesn't have one
  - Used like an "ack"
    - Prominent in AJAX applications
- 206 – Partial Content
  - Optimisation used in failure cases to support partial GETs
  - Request `Content-Range` header must specify the byte range of the resource representation it wants
  - Response headers must contain `Date`;
    - `ETag` and `Content-Location` headers must be consistent with the original request not the current values

3xx

- 301 – Multiple Choices
  - Response Location header should contain the preferred URI
  - Message body can contain list of URIs
    - In XHTML possibly
  - In general avoid being ambiguous!
- 301 – Moved Permanently
  - `Location` header contains the new location of the resource
- 303 – See Other
  - `Location` header contains the location of an alternative resource
  - Used for redirection

More 3xx

- 304 – Not Modified
  - The resource hasn't changed, use the existing representation
  - Used in conjunction with conditional GET
  - Client sends the `If-Modified-Since` header
  - Response `Date` header must be set
  - Response `Etag` and `Content-Location` headers must be same as original representation
- 307 – Temporary Redirect
  - The request hasn't been processed, because the resource has moved
  - Client must resubmit request to the URI in the response `Location` header

4xx

- 400 – Bad Request
  - The client has PUT or POST a resource representation that is in the right format, but contains invalid information
- 401 – Unauthorized
  - Proper credentials to operate on a resource weren't provided
  - Response `WWW-Authenticate` header contains the type of authentication the server expects
    - Basic, digest, WSSE, etc
  - Don't leak information!
    - Consider 404 in these situations

More 4xx

- 403 – Forbidden
  - The client request is OK, but the server doesn't want to process it
    - E.g. Restricted by IP address
  - Implies that resource exists, beware leaking information
- 404 – Not Found
  - The standard catch-all response
  - May be a lie to prevent 401 or 403 information leakage

Even more 4xx

- 405 – Method Not Allowed
  - The resource doesn't support a given method
  - The response `Allow` header lists the verbs the resource understands
    - E.g. `Allow: GET, POST, PUT`
- 406 – Not Acceptable
  - The client places too many restrictions on the resource representation via the `Accept-*` header in the request
  - The server can't satisfy any of those representations

Yet More 4xx

- 409 – Conflict
  - Tried to change the state of the resource to something the server won't allow
    - E.g. Trying to DELETE something that doesn't exist
- 410 – Gone
  - The resource has gone, permanently.
  - Don't send in response to DELETE
    - The client won't know if it was deleted, or if it was gone and the delete failed
- 411 – Length Required
  - If a request (POST, PUT) contains a representation, it should set the `Content-Length` header
  - The server *might* demand this, and interrupt the client request

Still more 4xx

- 412 – Precondition Failed
  - Server/resource couldn't meet one or more preconditions
    - As specified in the request header
  - E.g. Using `If-Unmodified-Since` and PUT to modify a resource provided it hasn't been changed by others
- 413 – Request Entity Too Large
  - Response comes with the `Retry-After` header in the hope that the failure is transient

Final 4xx Codes

- 414 – Request URI Too Long
  - You've got a hopeless HTTP server, or ridiculous URI names!
- 415 Unsupported Media Type
  - E.g. If the server resource expects JSON but the client sends XML

5xx Codes

- 500 – Internal Server Error
  - The normal response when we're lazy ☺
- 501 – Not Implemented
  - The client tried to use something in HTTP which this server doesn't support
    - Keep HTTP use simple, get a better HTTP server
- 502 – Bad Gateway
  - A proxy failed
  - Doesn't help us much ☹

More 5xx Codes

- 503 – Service Unavailable
  - The HTTP server is up, but not supporting resource communication properly
  - Server may send a `Retry-After` header, assuming the fault is transient

HTTP Headers

- Headers provide metadata to assist processing
  - Identify resource representation format (media type), length of payload, supported verbs, etc
- HTTP defines a wealth of these
  - And like status codes they are our building blocks for robust service implementations

Must-know Headers

- Authorization
  - Contains credentials (basic, digest, WSSE, etc)
  - Extensible
- Content-Length
  - Length of payload, in bytes
- Content-Type
  - The resource representation form
    - E.g. application/vnd.restbucks+xml, application/xhtml+xml

More Must-Know Headers

- Etag/If-None-Match
  - Opaque identifier – think “checksum” for resource representations
  - Used for conditional GET
- If-Modified-Since/Last-Modified
  - Used for conditional GET too
- Host
  - Contains the domain-name part of the URI

Yet More Must-Know Headers

- Location
  - Used to flag the location of a created/moved resource
  - In combination with:
    - 201 Created, 301 Moved Permanently, 302 Found, 307 Temporary Redirect, 300 Multiple Choices, 303 See Other
- User-Agent
  - Tells the server side what the client-side capabilities are
  - Should not be used in the programmable Web!

Final Must-Know Headers

- WWW-Authenticate
  - Used with 401 status
  - Informs client what authentication is needed
- Date
  - Mandatory!
  - Timestamps on request and response

Useful Headers

- Accept
  - Client tells server what formats it wants
  - Can externalise this in URI names in the general case
- Accept-Encoding
  - Client tells server that it can compress data to save bandwidth
  - Client specifies the compression algorithm it understands
- Content-Encoding
  - Server-side partner of Accept-Encoding

More Useful Headers

- Allow
  - Server tells client what verbs are allowed for the requested resource (used in combination with OPTIONS)
- Cache-Control
  - Metadata for caches, tells them how to cache (or not) the resource representation
  - And for how long etc.
- Content-MD5
  - Cryptographic checksum of body
  - Useful integrity check, has computation cost

Yet More Useful Headers

- Expect
  - A conditional – client asks if it's OK to proceed by expecting 100-Continue
  - Server either responds with 100 or 417 – Expectation Failed
- Expires
  - Server tells client or proxy server that representation can be safely cached until a certain time
- If-Match
  - Used for ETag comparison
  - Opposite of If-None-Match

Final Useful Headers

- If-Unmodified-Since
  - Useful for conditional PUT/POST
    - Make sure the resource hasn't changed while you're been manipulating it
  - Compare with If-Modified-Since
- Range
  - Specify part of a resource representation (a byte range) that you need – aka partial GET
  - Useful for failure/recovery scenarios

Less Often-Used Headers

- Retry-After
  - Resource or server is out of action for the specified time
  - Usually associated with 413 – Request Entity Too Large, or one of the 5xx server error statuses
- Content-Location
  - Header gives the canonical URI of the resource
  - Client might be using a different URI to access that resource

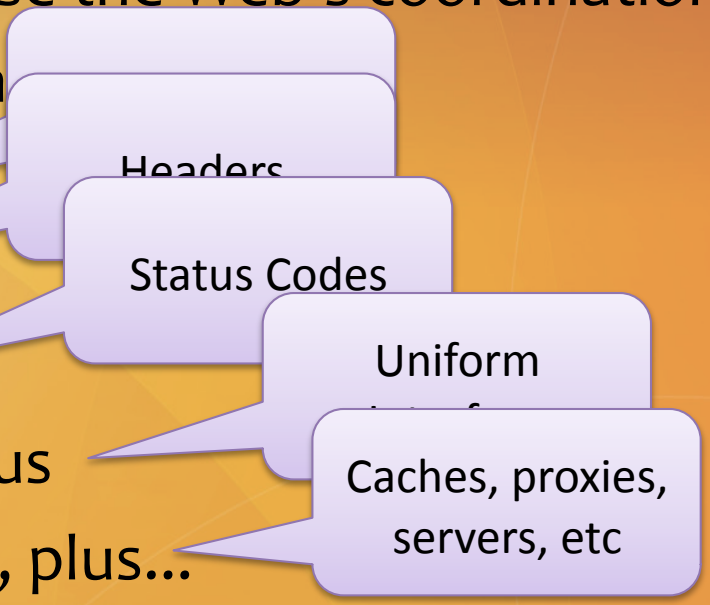
HTTP RCF 2616 is Authoritative

- The statuses and headers here are a sample of the full range of headers in the HTTP spec
- They spec contains more than we discuss here
- It is authoritative about usage
- And it’s a good thing to keep handy when you’re working on a Web-based distributed system!

Embracing HTTP as an Application Protocol

Using the Web

- URI tunnelling and POX use the Web as a transport
  - Just like SOAP without metadata support
- CRUD services begin to use the Web's coordination support
- But the Web is more than
  - Transport, plus
  - Metadata, plus
  - Fault model, plus
  - Component model, plus
  - Runtime environment, plus…

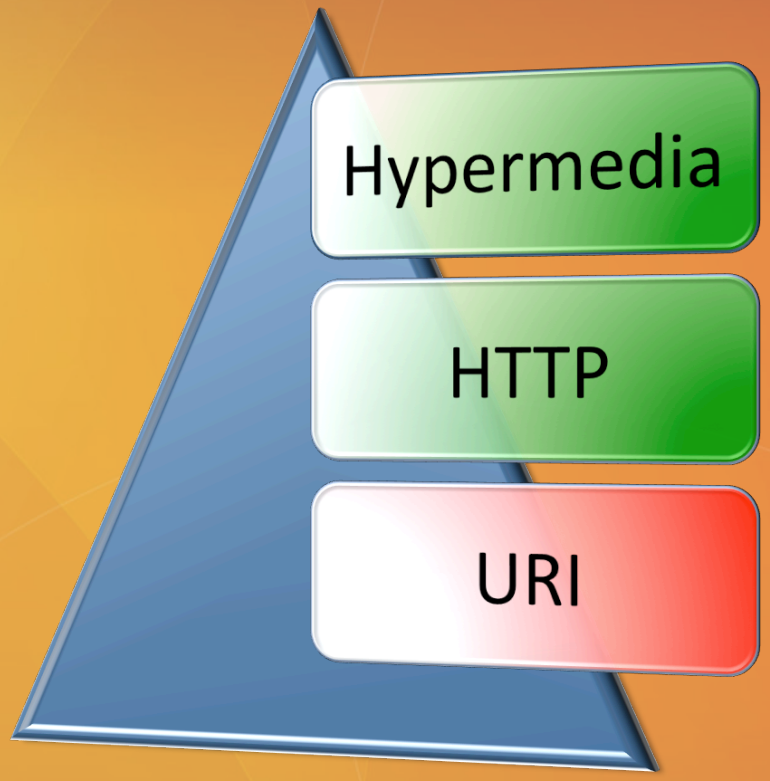


CRUD Resource Lifecycle

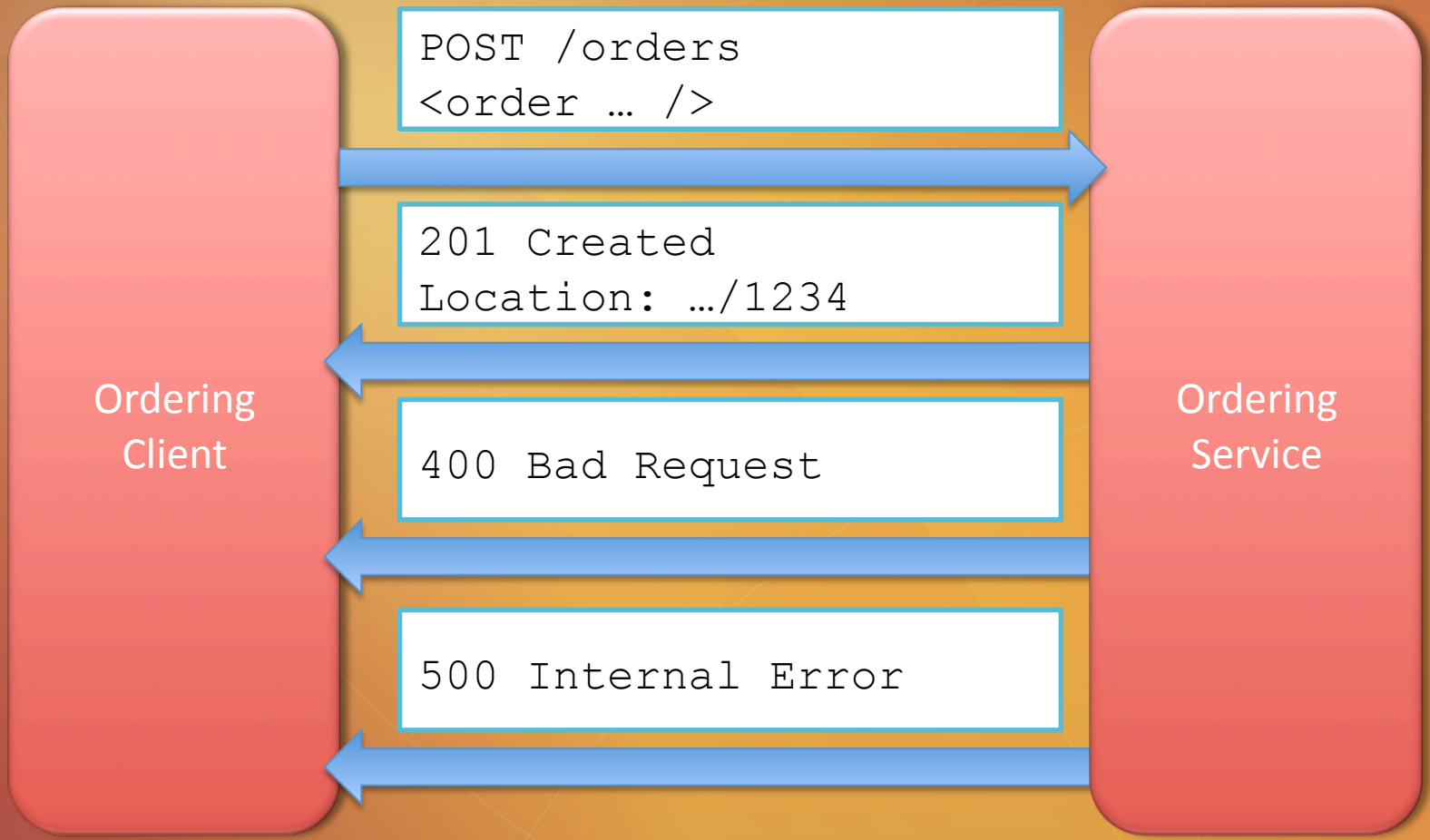
- The resource is created with POST
- It's read with GET
- And updated via PUT
- Finally it's removed using DELETE

Richardson Model Level 2

- Lots of URIs
- Understands HTTP!
- No hypermedia



Create with POST



POST Semantics

- POST creates a new resource
- But the server decides on that resource's URI
- Common human Web example: posting to Web log
  - Server decides URI of posting and any comments made on that post
- Programmatic Web example: creating a new employee record
  - And subsequently adding to it

POST Request

```
POST /orders HTTP/1.1
Host: restbucks.example.com
Content-Type:application/vnd.restbucks+xml
Content-Length: 225

<order xmlns="http://schemas.restbucks.com/order">
  <location>takeAway</location>
  <items>
    <item>
      <name>latte</name>
      <quantity>1</quantity>
      <milk>whole</milk>
      <size>small</size>
    </item>
  </items>
</order>
```

Verb, path, and HTTP version

Restbucks-specific XML content

Content (again Restbucks XML)

POST Response

```
HTTP/1.1 201 Created
Location: /orders/1234
```

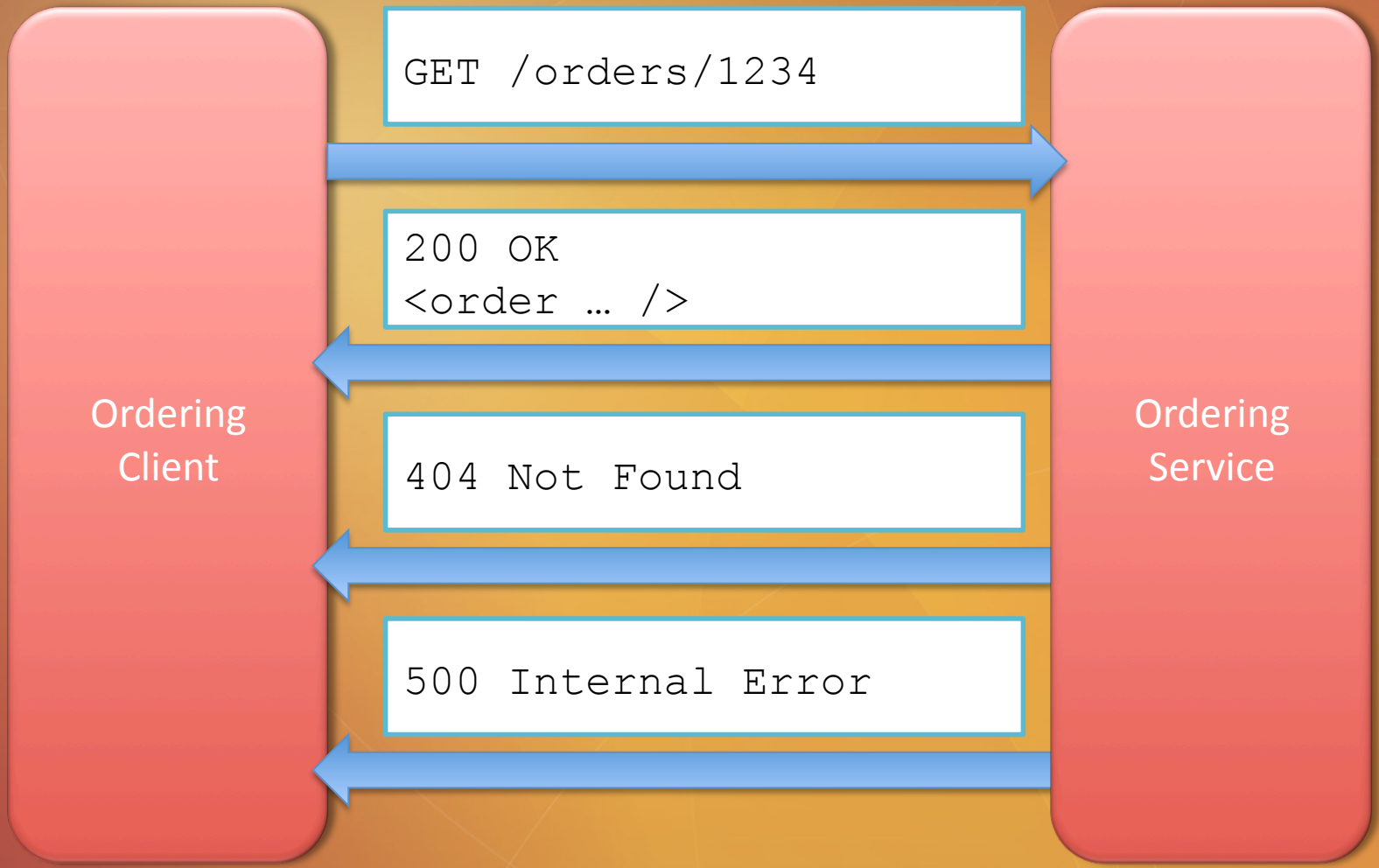
When POST goes wrong

- We may be 4xx or 5xx errors
  - Client versus server problem
- We turn to GET!
- Find out the resource states first
  - Then figure out how to make forward or backward progress
- Then solve the problem
  - May involve POSTing again
  - May involve a PUT to rectify server-side resources in-place

POST Implementation with a Servlet

```
protected void doPost(HttpServletRequest request,
                      HttpServletResponse response) {
  try {
    Order order = extractOrderFromRequest(request);
    String internalOrderId = OrderDatabase.getDatabase().saveOrder(order);
    response.setHeader("Location", computeLocationHeader(request,
                       internalOrderId));
    response.setStatus(HttpServletResponse.SC_CREATED);
  } catch(Exception ex) {
    response.setStatus(HttpServletResponse.SC_INTERNAL_SERVER_ERROR);
  }
}
```

Read with GET



GET Semantics

- GET retrieves the representation of a resource
- Should be idempotent
  - Shared understanding of GET semantics
  - Don't violate that understanding!

Library of congress catalogue incident!

GET Exemplified

```
GET /orders/1234 HTTP/1.1
Accept: application/vnd.restbucks+xml
Host: restbucks.com
```

GET Response

```
HTTP/1.1 200 OK
Content-Length: 232
Content-Type: application/vnd.restbucks+xml
Date: Wed, 19 Nov 2008 21:48:10 GMT

<order xmlns="http://schemas.restbucks.com/order">
  <location>takeAway</location>
  <items>
    <item>
      <name>latte</name>
      <quantity>1</quantity>
      <milk>whole</milk>
      <size>small</size>
    </item>
  </items>
  <status>pending</pending>
</order>
```

When GET Goes wrong

- Simple!
  - Just 404 – the resource is no longer available

```
HTTP/1.1 404 Not Found
Content-Type: application/vnd.restbucks+xml
Content-Length: 952
Date: Sat, 20 Dec 2008 19:01:33 GMT
```

- Are you sure?
  - GET again!
- GET is safe and idempotent
  - Great for crash recovery scenarios!

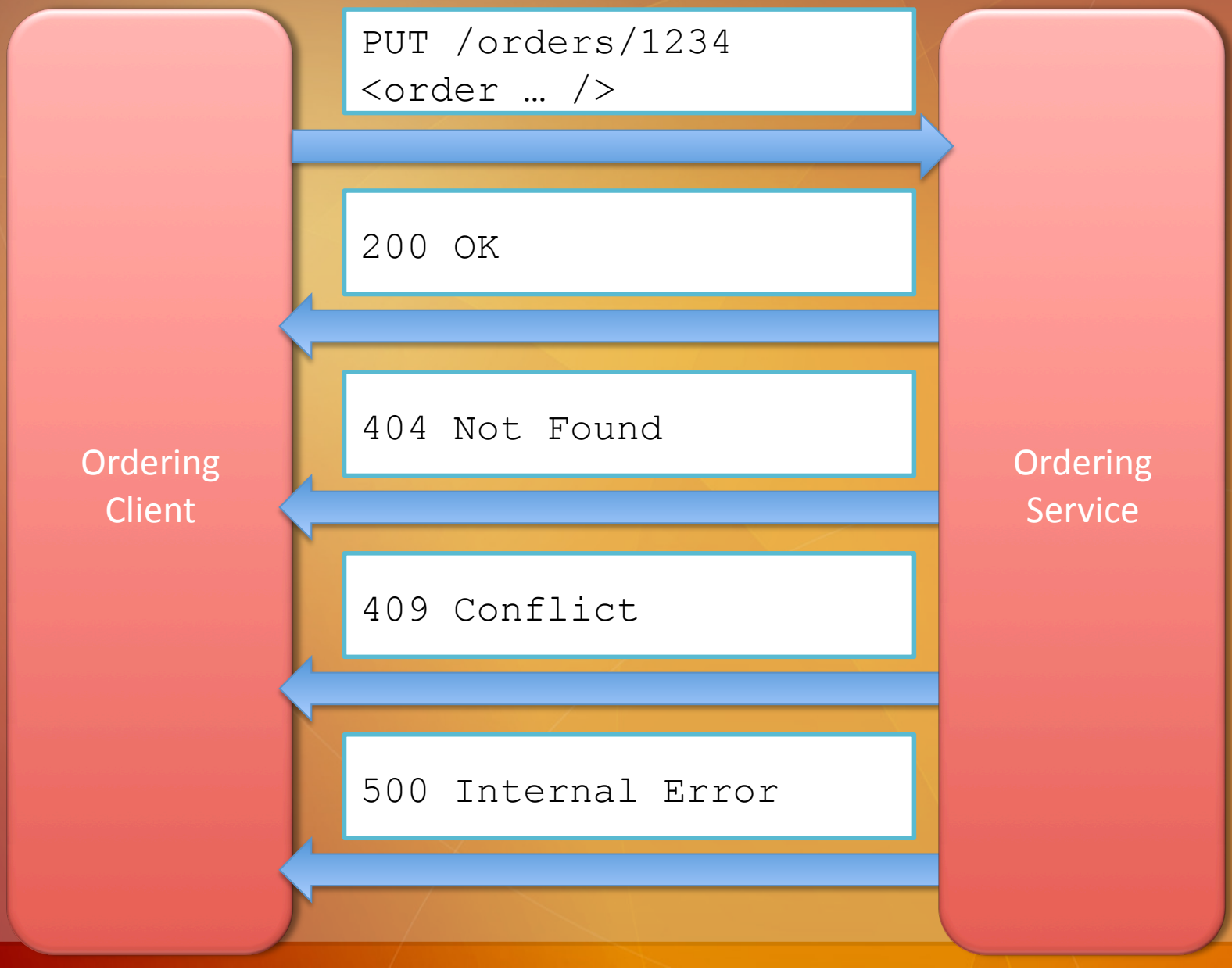
Idempotent Behaviour

- An action with no side affects
  - Comes from mathematics
- In practice means two things:
  - A safe operation is one which changes no state at all
    - E.g. HTTP GET
  - An idempotent operation is one which updates state in an absolute way
    - E.g. x = 4 rather than x += 2
- Web-friendly systems scale because of safety
  - Caching!
- And are fault tolerant because of idempotent behaviour
  - Just re-try in failure cases

GET JAX-RS Implementation

```
@Path("/")
public class OrderingService {
  @GET
  @Produces("application/xml")
  @Path("/{orderId}")
  public String getOrder(@PathParam("orderId") String orderId) {
    try {
      Order order = OrderDatabase.getDatabase().getOrder(orderId);
      if (order != null) {
        return xstream.toXML(order);
      } else {
        throw new WebApplicationException(404);
      }
    } catch (Exception e) {
      throw new WebApplicationException(500);
    }
  }
  // Remainder of implementation omitted for brevity
}
```

Update with PUT



PUT Semantics

- PUT creates a new resource but the client decides on the URI
  - Providing the server logic allows it
- Also used to update existing resources by overwriting them in-place
- PUT is idempotent
  - Makes absolute changes
- But is not safe
  - It changes state!

PUT Request

```
PUT /orders/1234 HTTP/1.1
Host: restbucks.com
Content-Type: application/xml
Content-Length: 386

<order xmlns="http://schemas.restbucks.com/order">
  <location>takeAway</location>
  <items>
    <item>
      <milk>whole</milk>
      <name>latte</name>
      <quantity>2</quantity>
      <size>small</size>
    </item>
    <item>
      <milk>whole</milk>
      <name>cappuccino</name>
      <quantity>1</quantity>
      <size>large</size>
    </item>
  </items>
  <status>preparing</preparing>
</order>
```

Updated content

PUT Response

```
HTTP/1.1 200 OK
Date: Sun, 30 Nov 2008 21:47:34 GMT
Content-Length: 0
```

Minimalist response contains no entity body

When PUT goes wrong

- If we get 5xx error, or some 4xx errors simply PUT again!
  - PUT is idempotent
- If we get errors indicating incompatible states (409, 417) then do some forward/backward compensating work
  - And maybe PUT again

```
HTTP/1.1 409 Conflict
Date: Sun, 21 Dec 2008 16:43:07 GMT
Content-Length:382

<order xmlns="http://schemas.restbucks.com/
    order">
  <location>takeAway</location>
  <items>
    <item>
      <milk>whole</milk>
      <name>latte</name>
      <quantity>2</quantity>
      <size>small</size>
    </item>
    <item>
      <milk>whole</milk>
      <name>cappuccino</name>
      <quantity>1</quantity>
      <size>large</size>
    </item>
  </items>
  <status>served</status>
</order>
```

WCF Implementation for PUT

```
[ServiceContract]
public interface IOrderingService
{
  [OperationContract]
  [WebInvoke(Method = "PUT", UriTemplate = "/orders/
   {orderId}")]
  void UpdateOrder(string orderId, Order order);

  // …
}
```

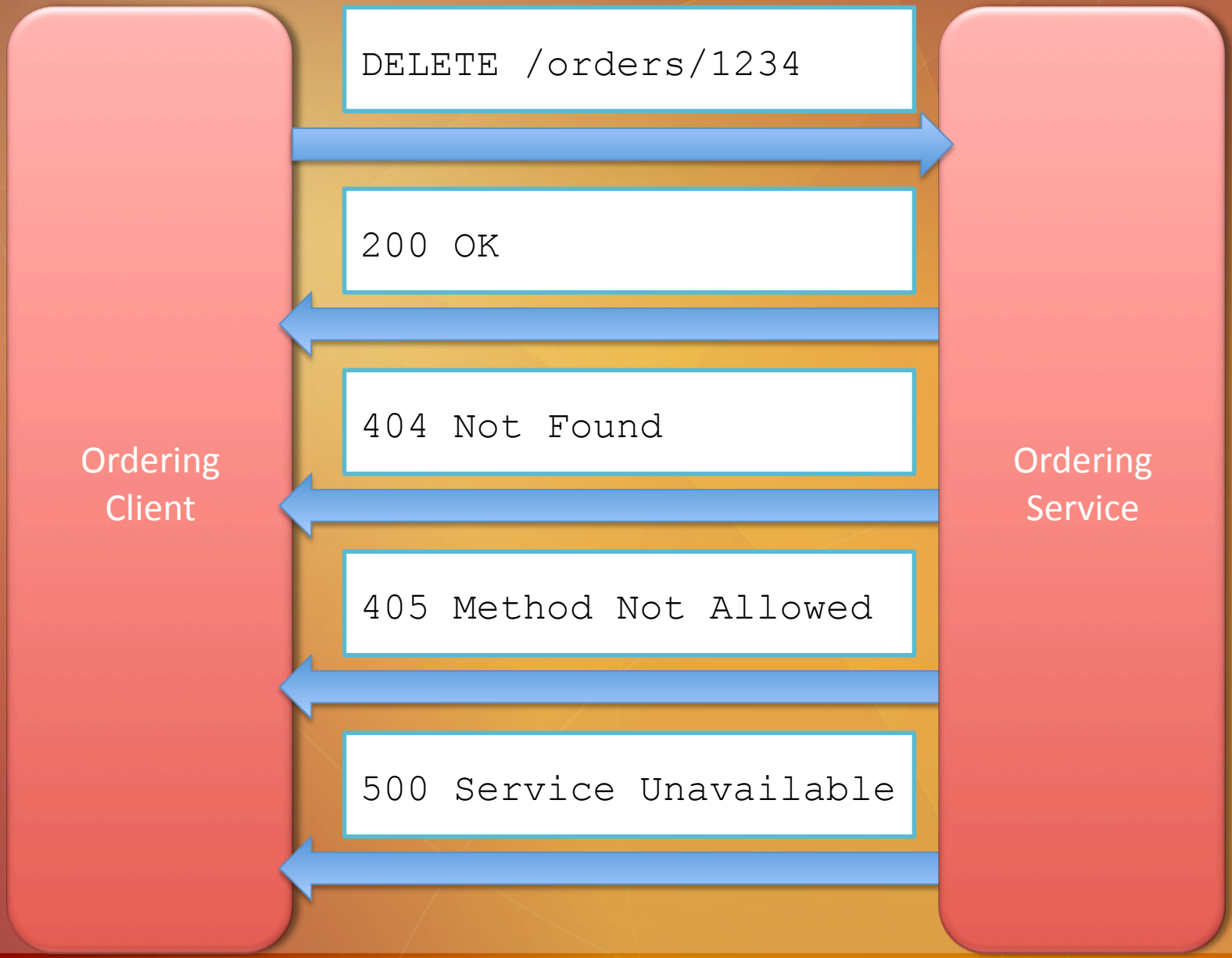
WCF Serializable Types

```
[DataContract(Namespace = "http://schemas.restbucks.com/order", Name = "order")]
public class Order
{
  [DataMember(Name = "location")]
  public Location ConsumeLocation
  {
    get { return location; }
    set { location = value; }
  }

  [DataMember(Name = "items")]
  public List<Item> Items
  {
    get { return items; }
    set { items = value; }
  }

  [DataMember(Name = "status")]
  public Status OrderStatus
  {
    get { return status; }
    set { status = value; }
  }
  // …
}
```

Remove with DELETE



DELETE Semantics

- Stop the resource from being accessible
  - Logical delete, not necessarily physical
- Request

```
DELETE /orders/1234 HTTP/1.1
Host: restbucks.com
```

- Response

```
HTTP/1.1 200 OK
Content-Length: 0
Date: Tue, 16 Dec 2008 17:40:11 GMT
```

This is important for decoupling implementation details from resources

When DELETE goes wrong

- Simple case, DELETE again!
  - Delete is idempotent!
  - DELETE once, DELETE 10 times has the same effect: one deletion

```
HTTP/1.1 404 Not Found
Content-Length: 0
Date: Tue, 16 Dec 2008 17:42:12 GMT
```

When DELETE goes Really Wrong

- Some 4xx responses indicate that deletion isn't possible
  - The state of the resource isn't compatible
  - Try forward/backward compensation instead

```
HTTP/1.1 409 Conflict
Content-Length: 379
Date: Tue, 16 Dec 2008 17:53:09 GMT

<order xmlns="http://schemas.restbucks.com/
    order">
  <location>takeAway</location>
  <items>
    <item>
      <name>latte</name>
      <milk>whole</milk>
      <size>small</size>
      <quantity>2</quantity>
    </item>
    <item>
      <name>cappuccino</name>
      <milk>skim</milk>
      <size>large</size>
      <quantity>1</quantity>
    </item>
  </items>
  <status>served</status>
</order>
```

Can't delete an order that's already served

CRUD does not mean Worthless

CRUD is Good?

- CRUD is good
  - But it's not great
- CRUD-style services use some HTTP features
- But the application model is limited
  - Suits database-style applications
  - Hence frameworks like Microsoft's Astoria
- CRUD has limitations
  - CRUD ignores hypermedia
  - CRUD encourages tight coupling through URI templates
  - CRUD encourages server and client to collude
- The Web supports more sophisticated patterns than CRUD!

Tech Interlude

Semantics

Microformats

- Microformats are an example of little “s” semantics
- Innovation at the edges of the Web
  - Not by some central design authority (e.g. W3C)
- Started by embedding machine-processable elements in Web pages
  - E.g. Calendar information, contact information, etc
  - Using existing HTML features like `class`, `rel`, etc

Semantic versus semantic

- Semantic Web is top-down
  - Driven by the W3C with extensive array of technology, standards, committees, etc
  - Has not currently proven as scalable as the visionaries hoped
    - RDF triples have been harvested and processed in private databases
- Microformats are bottom-up
  - Little formal organisation, no guarantee of interoperability
  - Popular formats tend to be adopted (e.g. hCard)
  - Easy to use and extend for our systems
  - Trivial to integrate into current and future programmatic Web systems

Microformats and Resources

- Use Microformats to structure resources where formats exist
  - I.e. Use hCard for contacts, hCalendar for data
- Create your own formats (sparingly) in other places
  - Annotating links is a good start
  - `<link rel="withdraw.cash" .../>`
  - `<link rel="service.post" type="application/atom+xml" href="{post-uri}" title="some title">`
- The `rel` attribute describes the semantics of the referred resource

Tech Interlude

Hypermedia Formats

Media Types Rule!

- The Web's contracts are expressed in terms of media types and link relations
  - If you know the type, you can process the content
- Some types are special because they work in harmony with the Web
  - We call these "hypermedia formats"

(Hyper) media types

Standardised media type

Processing model

- Hypermedia controls (links and forms)
- Supported operations (methods, headers and status codes)
- Representation formats (may include schemas)

General

Compose application-specific behaviours on top of the handling of standardised media types

Specific

Other Resource Representations

- Remember, XML is not the only way a resource can be serialised
  - Remember the Web is based on **RE**presentational **S**tate **T**ransfer
- The choice of representation is left to the implementer
  - Can be a standard registered media type
  - Or something else
- But there is a division on the Web between two families
  - Hypermedia formats
    - Formats which host URIs and links
  - Regular formats
    - Which don't

Plain Old XML is not Hypermedia Friendly

```
HTTP/1.1 200 OK
Content-Length: 227
Content-Type: application/xml
Date: Wed, 19 Nov 2008 21:48:10 GMT

<order xmlns="http://schemas.restbucks.com/order">
  <location>takeAway</location>
  <items>
    <item>
      <name>latte</name>
      <quantity>1</quantity>
      <milk>whole</milk>
      <size>small</size>
    </item>
  </items>
  <status>pending</pending>
</order>
```

Where are the links?
Where's the protocol?

So what?

- How do you know the next thing to do?
- How do you know the resources you're meant to interact with next?
- In short, how do you know the service's protocol?
  - Turn to WADL? Yuck!
  - Read the documentation? Come on!
  - URI Templates? Tight Coupling!

URI Templates are NOT a Hypermedia Substitute

- Often URI templates are used to advertise all resources a service hosts
  - Do we really need to advertise them all?
- This is verbose
- This is out-of-band communication
- This encourages tight-coupling to resources through their URI template
- This has the opportunity to cause trouble!
  - Knowledge of “deep” URIs is baked into consuming programs
  - Services encapsulation is weak and consumers will program to it
  - Service will change its implementation and break consumers

Bad Ideas with URI Templates

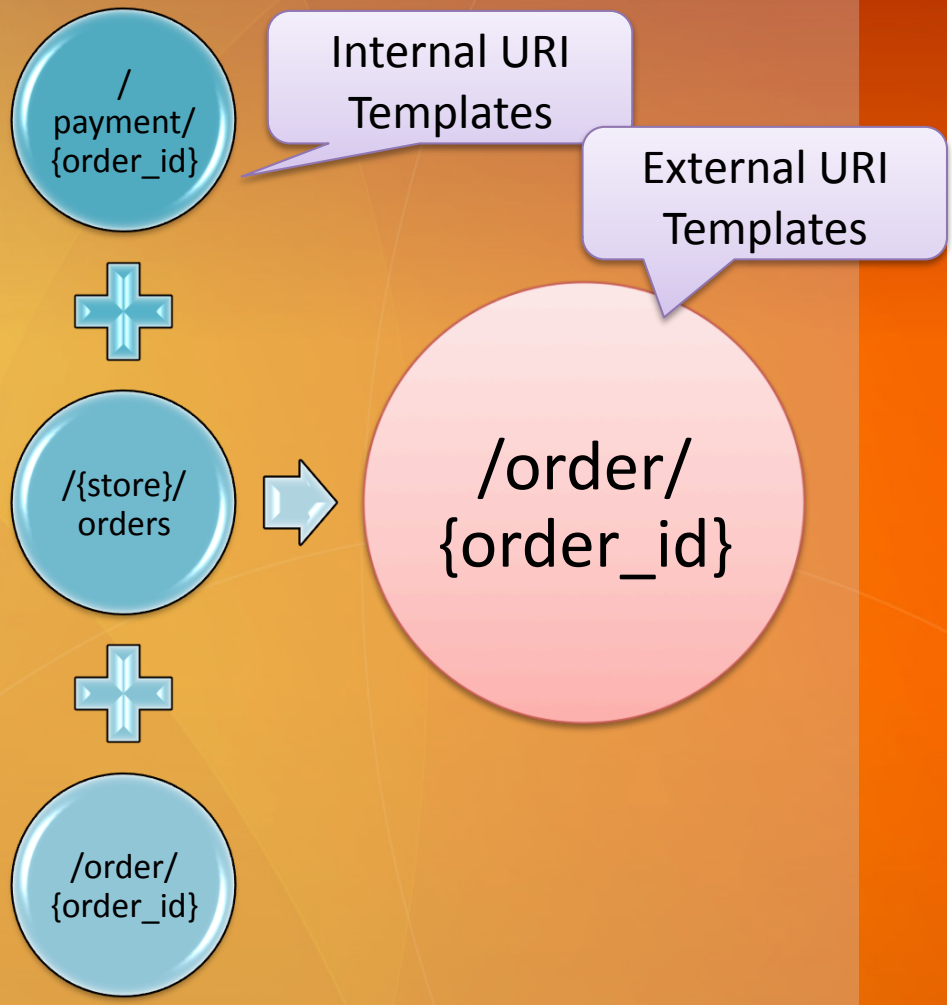
- Imagine we're created an order, what next?
- We could share this URI template:
  - `http://restbucks.com/payment/{order_id}`
- The `order_id` field should match the order ID that came from the restbucks service
  - Sounds great!
- But what if Restbucks outsources payment?
  - Change the URI for payments, break the template, break consumers!
  - D'oh!
- Be careful what you share!

Better Ideas for URI Templates: Entry Points

- Imagine that we have a well-known entry point to our service
  - Which corresponds to a starting point for a protocol
- Why not advertise that with a URI template?
- For example:
  - `http://restbucks.com/signIn/{store_id}/{barista_id}`
- Changes infrequently
- Is important to Restbucks
- Is transparent, and easy to bind to

Better Ideas for URI Templates: Documentation!

- Services tend to support lots of resources
- We need a shorthand for talking about a large number of resources easily
- We can use a URI template for each "type" of resource that a service (or services) supports
- But we don't share this information with others
  - Don't violate encapsulation!



`application/xml` is not the media type you're looking for

- Remember that HTTP is an application protocol
  - Headers and representations are intertwined
  - Headers set processing context for representations
- Remember that `application/xml` has a particular processing model
  - Which doesn't include understanding the semantics of links
- Remember if a representation is declared in the `Content-Type` header, you must treat it that way
  - HTTP is an application protocol – did you forget already? ☺
- We need real hypermedia formats!

Hypermedia Formats

- Standard
  - Wide “reach”
  - Software agents already know how to process them
  - But sometimes need to be shoe-horned
- Self-created
  - Can craft specifically for domain
  - Semantically rich
  - But lack reach

Two Common Hypermedia Formats: XHTML and ATOM

- Both are commonplace today
- Both are hypermedia formats
  - They contain links
- Both have a processing model that explicitly supports links
- Which means both can describe *protocols*…

XHTML

- XHTML is just HTML that is also XML
- For example:

```
<html xmlns="http://www.w3.org/1999/xhtml"
  xml:lang="en" lang="en"
  xmlns:r="http://restbuc
  <head>
    <title> XHTML Example </title>
  </head>
  <body>
    <p>
      ...
```

Default XML namespace

Other XML namespaces

What's the big deal with XHTML?

- It does two interesting things:
  1. It gives us document structure
  2. It gives us links
- So?
  1. We can understand the format of those resources
  2. We can discover other resources!
- How?
  1. Follow the links!
  2. Encode the resource representation as "normal" XML in your XHTML documents
- Contrast this with the Atom and APP approach...similar!

XHTML in Action

```
<html xmlns="http://www.w3.org/1999/xhtml">
<body>
  <div class="order">
    <p class="location">takeAway</p>
    <ul class="items">
      <li class="item">
        <p class="name">latte</p>
        <p class="quantity">1</p>
        <p class="milk">whole</p>
        <p class="size">small</p>
      </li>
    </ul>
    <a href="http://restbucks.com/payment/
  1234"
       rel="payment">payment</a>
  </div>
</body>
</html>
```

Business data

"Hypermedia Control"

`application/xhtml+xml`

- Can ask which verb the resource at the end of the link supports
  - Via HTTP OPTIONS
- No easy way to tell what each link actually does
  - Does it buy the music?
  - Does it give you lyrics?
  - Does it vote for a favourite album?
- We lack semantic understanding of the linkspace and resources
  - But we have microformats for that semantic stuff!
- Importantly XHTML is a hypermedia format
  - It contains hypermedia controls that can be used to describe protocols!

Atom Syndication Format

- We'll study this in more depth later, but for now…
- The application/atom +xml media type is hypermedia aware
- You should expect links when processing such represent
- And be pr
things with them!

```
HTTP/1.1 200 OK
Content-Length: 342
Content-Type: application/atom+xml
Date: Sun, 22 Mar 2009 17:04:10 GMT

<entry xmlns="http://www.w3.org/2005/
    Atom">
  <title>Order 1234</title>
  <link rel="payment" href="http://
    restbucks.com/payment/1234"/>
  <link rel="special-offer"
    href="http://restbucks.com/offers/
    freeCookie"/>
  <id>http://restbucks.com/order/
    1234</id>
  <updated>2009-03-22T16:57:02Z</
    updated>
  <summary>1x Cafe Latte</summary>
entry>
```

Links to other resources, a nascent protocol

`application/atom+xml`

- No easy way to tell what each link actually does
  - But look at the way the `rel` attribute is being used
  - Can we inject semantics there?
- Atom is a hypermedia format
  - Both feeds and entries contains hypermedia controls that can describe protocols

`application/vnd.restbucks+xml`

- What a mouthful!
- The `vnd` namespace is for proprietary media types
  - As opposed to the IANA-registered ones
- Restbucks own XML is a hybrid
  - We use plain old XML to convey information
  - And Atom `link` elements to convey protocol
- This is important, since it allows us to create RESTful, hypermedia aware services

Hypermedia and RESTful Services

Revisiting Resource Lifetime

- On the Web, the lifecycle of a single resource is more than:
  - Creation
  - Updating
  - Reading
  - Deleting
- Can also get metadata
  - About the resource
  - About its (subset of) the verbs it understands
- And as we see, resources tell us about other resources we might want to interact with…

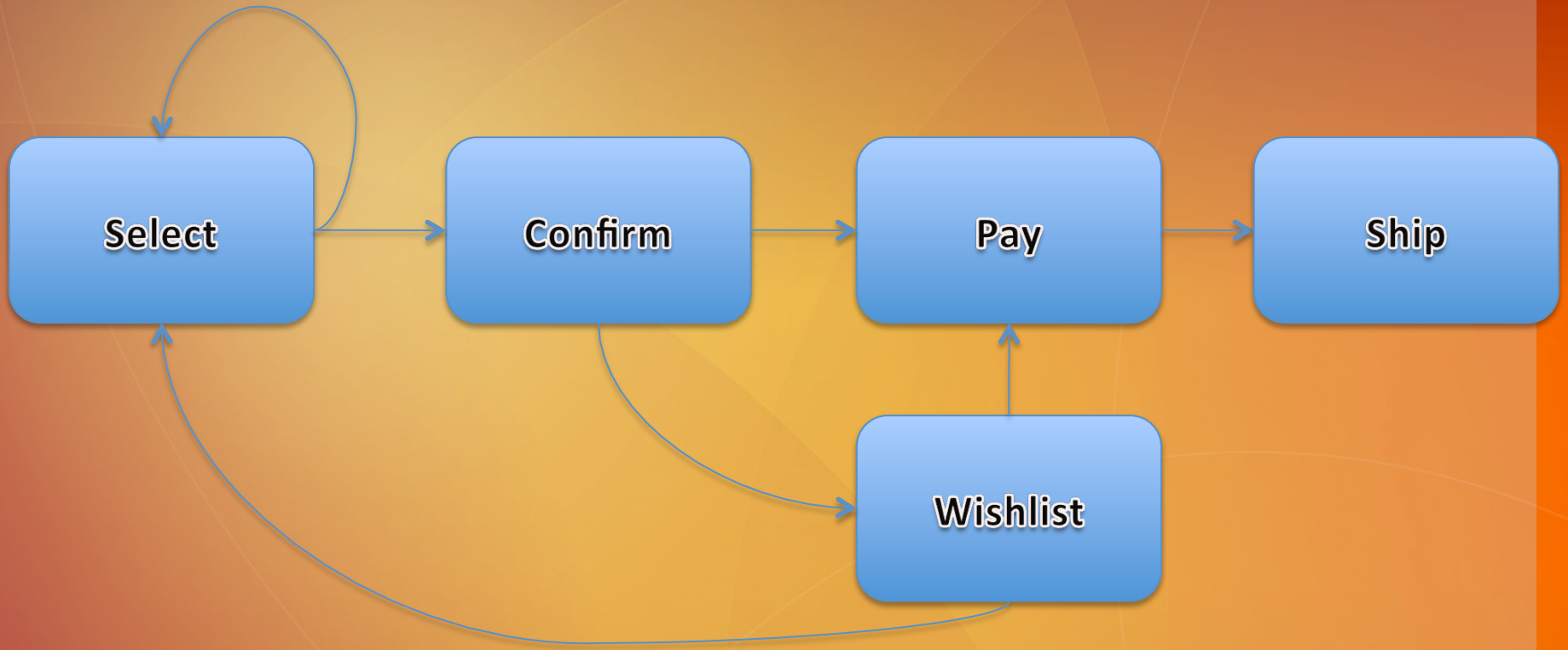
Links

- Connectedness is good in Web-based systems
- Resource representations can contain other URIs
- Links act as state transitions
- Application (conversation) state is captured in terms of these states

Describing Contracts with Links

- The value of the Web is its “linked-ness”
  - Links on a Web page constitute a contractfor page traversals
- The same is true of the programmatic Web
- Use Links to describe state transitions in programmatic Web services
  - By navigating resources you change application state
- Hypermedia formats support this
  - Allow us to describe higher-order protocols which sit comfortably atop HTTP
  - Hence `application/vnd.restbucks+xml`

Links are State Transitions



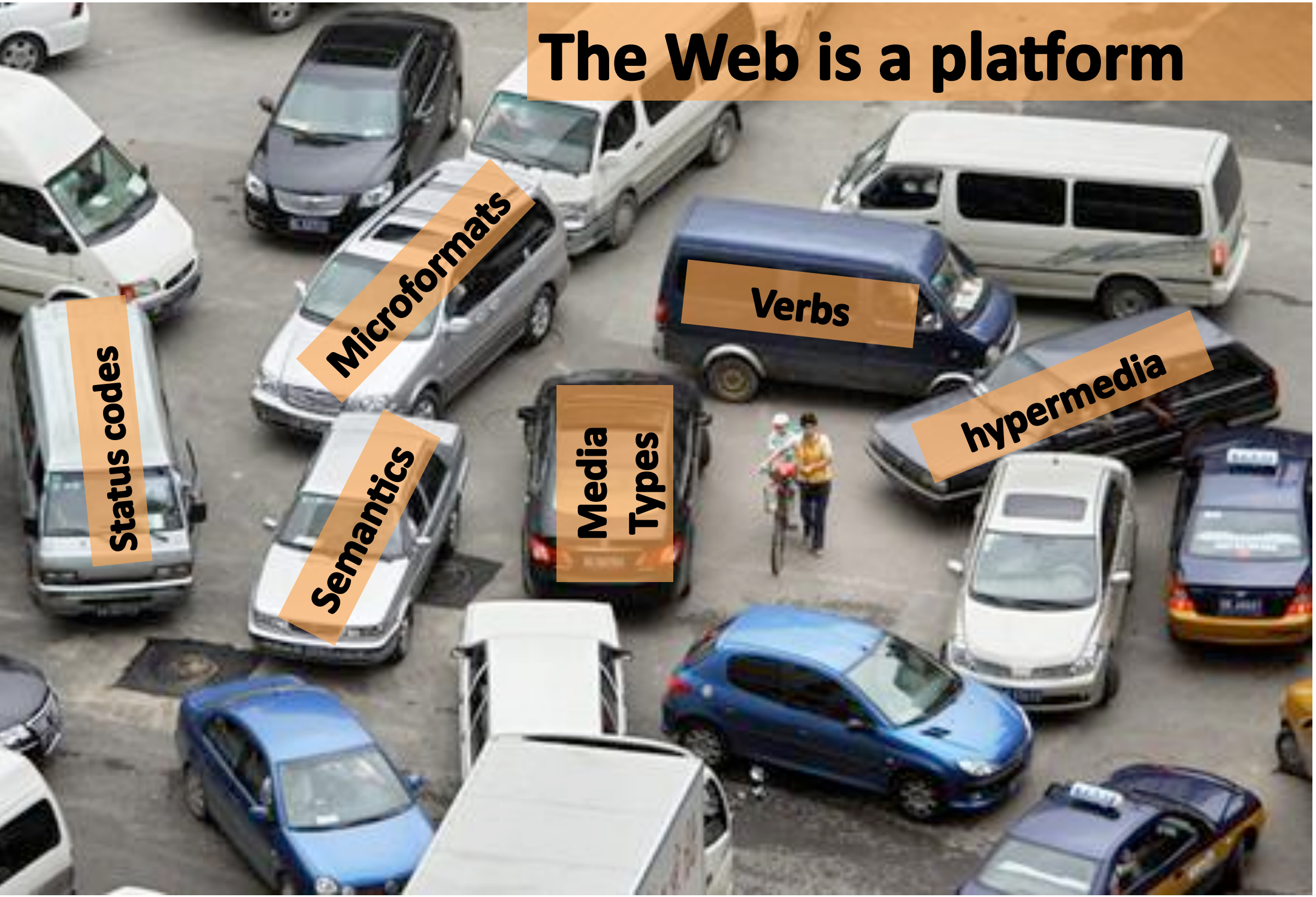
Links as APIs

```
<confirm xmlns="...">
<link rel="payment"
   href="https://pay"
   type="application/xml"/>
<link rel="postpone"
   href="https://wishlist"
   type="application/xml"/>
</confirm>
```

- Following a link causes an action to occur
- This is the start of a state machine!
- Links lead to other resources which also have links
- Can make this stronger with semantics
  - Microformats

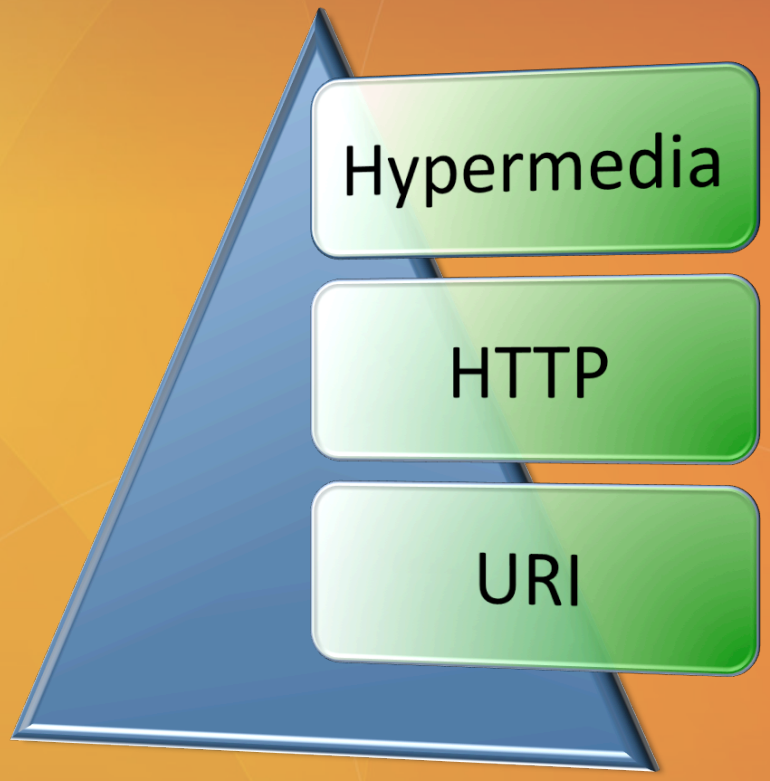
We have a framework!

- The Web gives us a processing and metadata model
  - Verbs and status codes
  - Headers
- Gives us metadata contracts or Web “APIs”
  - URI Templates
  - Links
- Strengthened with semantics
  - Little “s”


The Web is a platform
Microformats
Verbs
hypermedia
Status codes
Media Types
Semantics

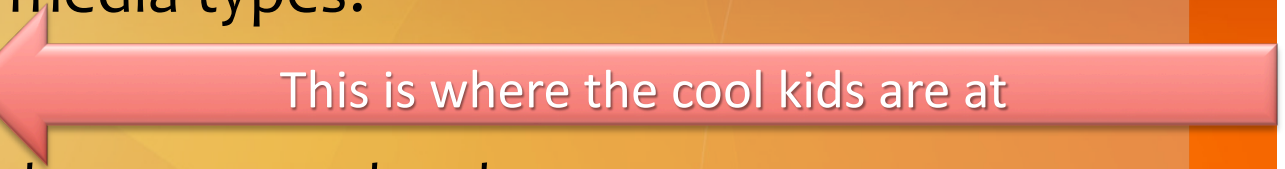
Richardson Model Level 3

- Lots of URIs that address resources
- Embraces HTTP as an application protocol
- Resource representations and formats identify other resources
  - Hypermedia at last!



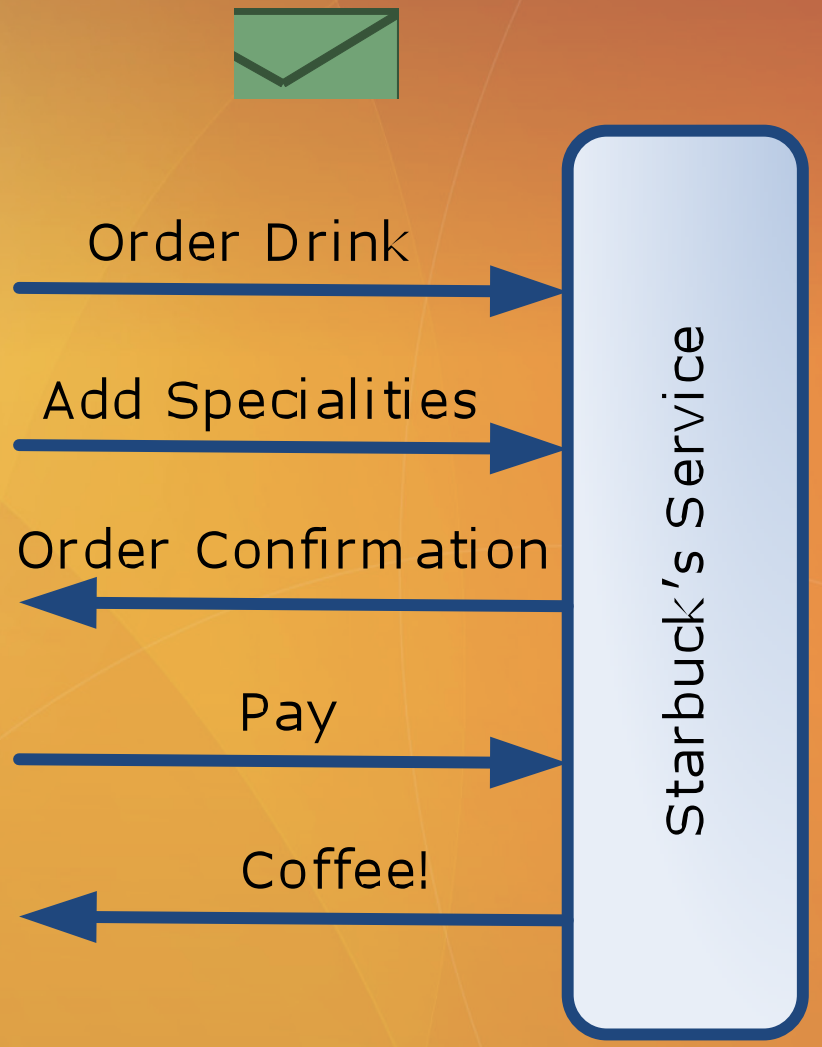
Structural versus Protocol

- "Structural" REST
  - Domain model broken up with URIs
  - Lazy loading, caching over the network
  - Proliferation of media types?
- "Protocol" REST
  - Focus on media types as contracts
  - Protocol state transitions
  - DAPs – Domain Application Protocols



Workflow and MOM

- With Web Services we exchange messages with the service
- Resource state is hidden from view
- Conversation state is all we know
  - Advertise it with SSDL, BPEL
- Uniform interface, roles defined by SOAP
  - No “operations”

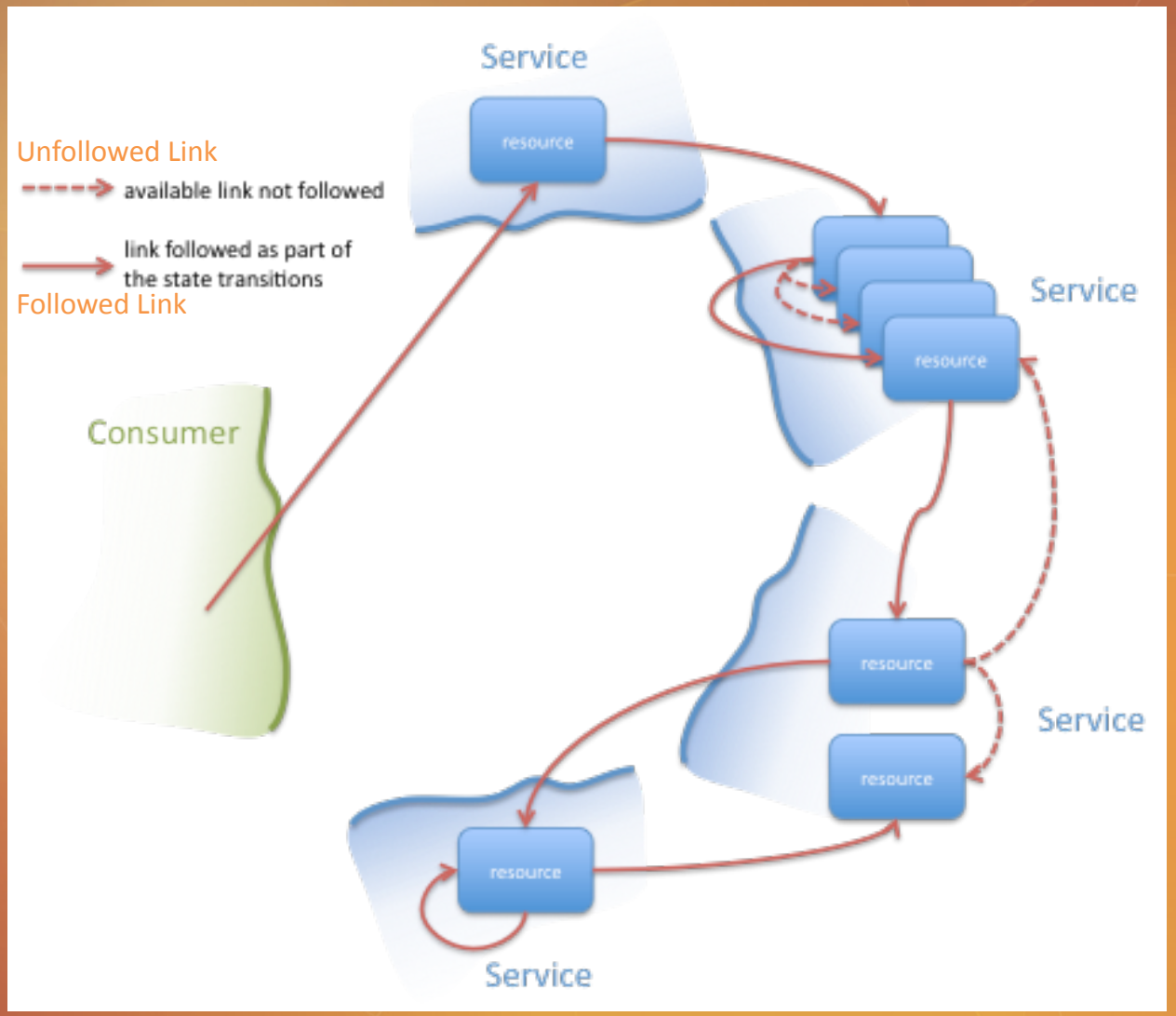


Hypermedia Describes Protocols!

- Links declare next valid steps
- Following links and interacting with resources changes *application state*
- Media types with links define contracts
  - Media type defines processing model
  - Links (with microformats) describe state transitions
- Don't need a static contract description
  - No WSDL, no WADL
- This is HATEOAS!
- So let's see how we order a coffee at Restbucks.com…
  - Based on: http://www.infoq.com/articles/webber-rest-workflow

Domain Application Protocols!

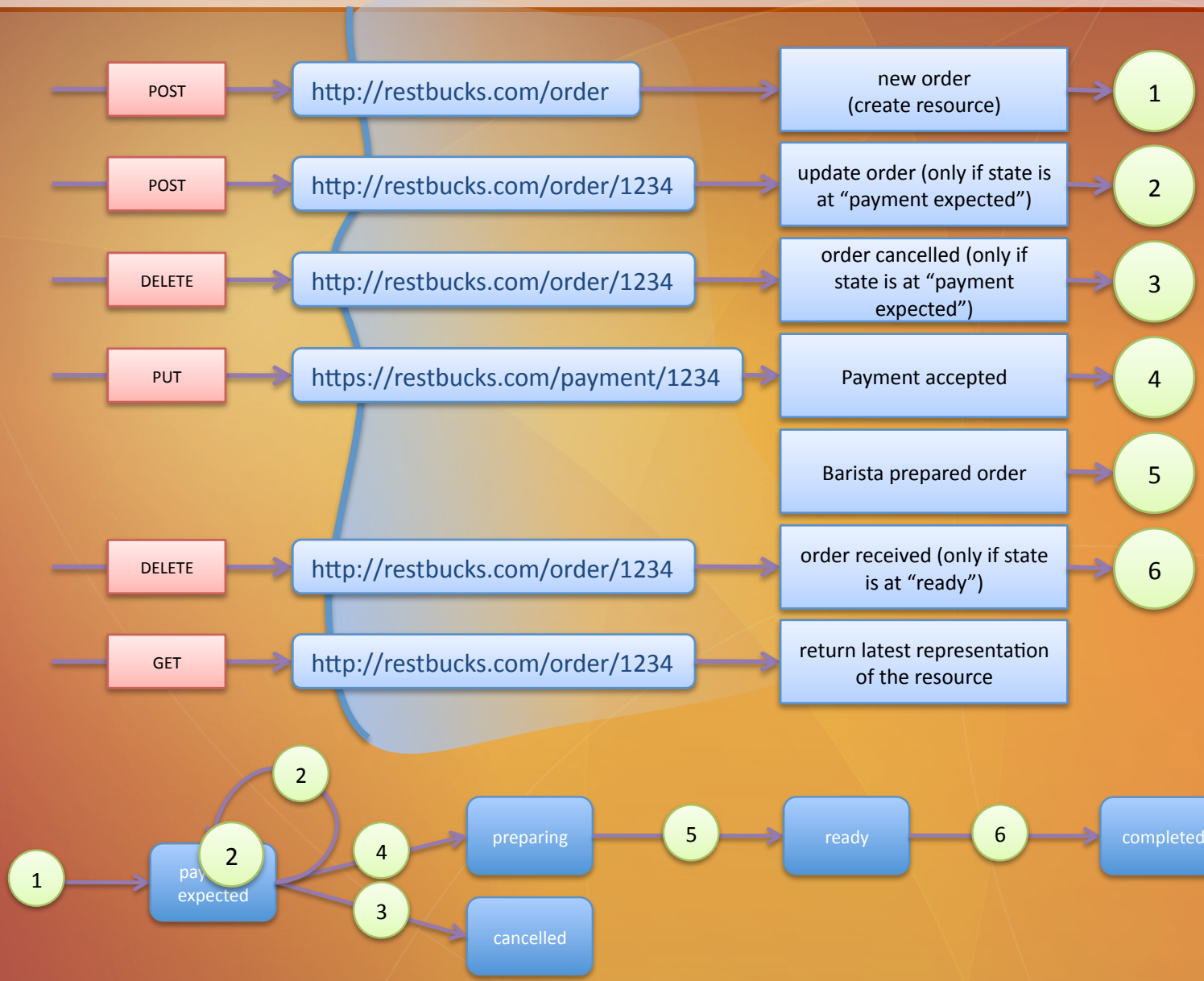
Hypermedia Protocols Span Services



Workflow

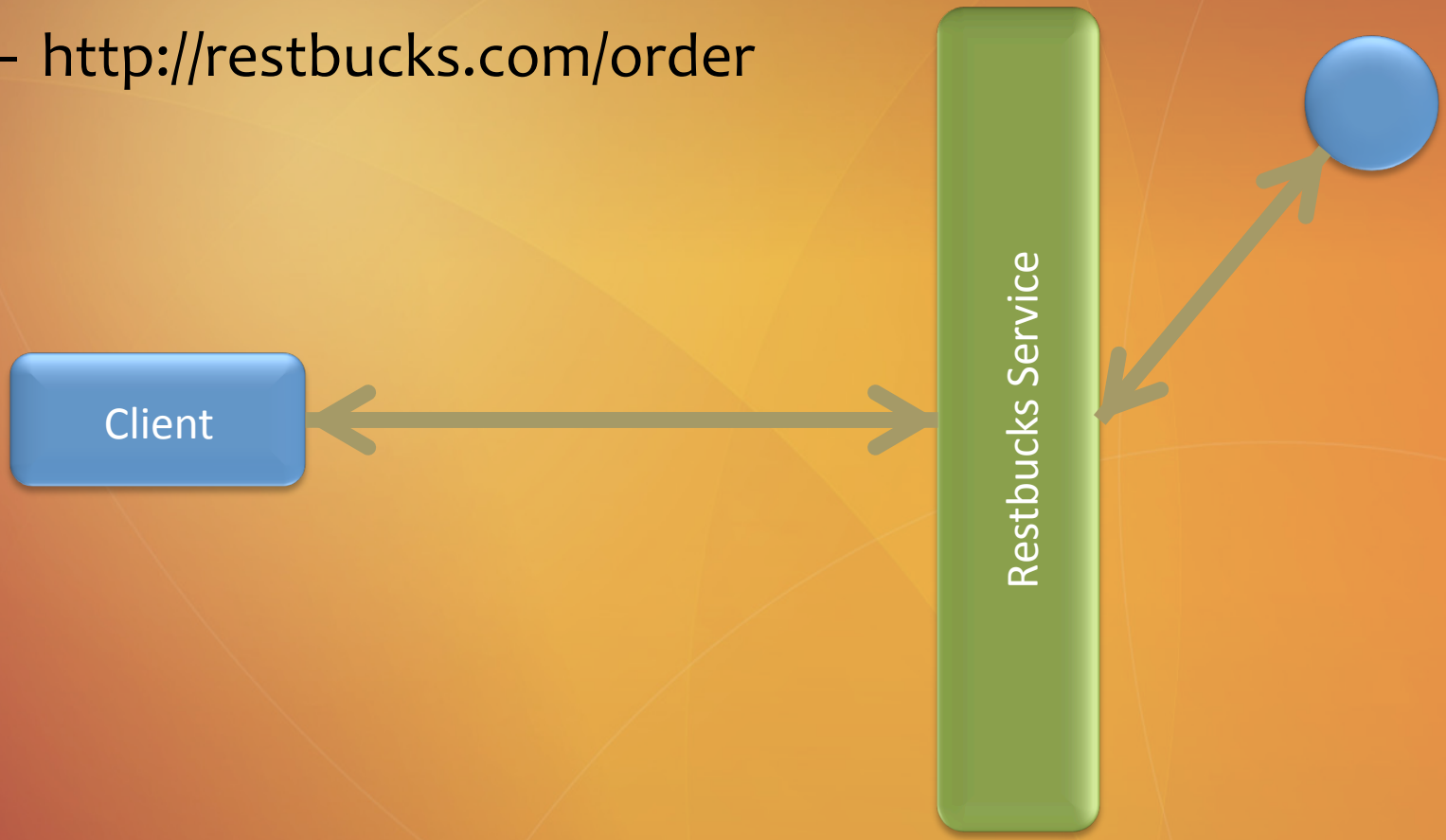
- How does a typical enterprise workflow look when it's implemented in a Web-friendly way?
- Let's take Restbucks ordering servas an example, the happy path is:
  - Make selection
    - Add any specialities
  - Pay
  - Wait for a while
  - Collect drink

Static Interface and State Transitions



Place an Order

- POST to a well-known URI
  - http://restbucks.com/order



Placing an Order

- Request

```
POST /order HTTP/1.1
Content-Type: application/vnd.restbucks+xml
Accept: application/vnd.restbucks+xml
Host: restbucks.com
Connection: keep-alive
Content-Length: 278

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<rb:order xmlns:rb="http://schemas.restbucks.com">
  <rb:item>
    <rb:milk>semi</rb:milk>
    <rb:size>large</rb:size>
    <rb:drink>latte</rb:drink>
  </rb:item>
  <rb:location>takeaway</rb:location>
</rb:order>
```

Placing an Order

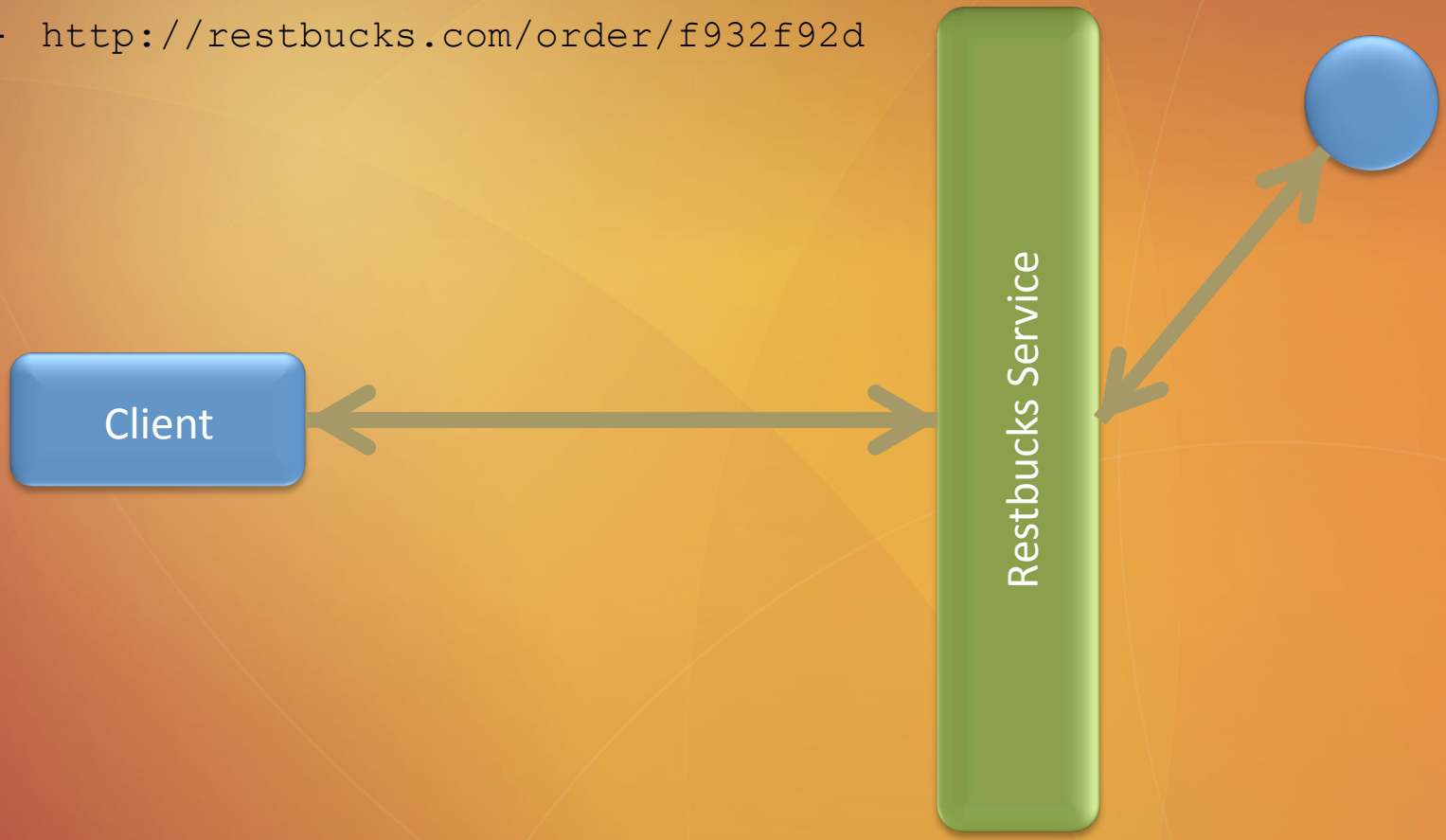
- Response

```
HTTP/1.1 201 Created
Location: http://restbucks.com/order/f932f92d
Content-Type: application/vnd.restbucks+xml
Content-Length: 828
Date: Sun, 06 Sep 2009 06:51:22 GMT

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<rb:order xmlns:rb="http://schemas.restbucks.com" xmlns:dap="http://schemas.restbucks.com/dap">
  <dap:link uri="http://restbucks.com/order/f932f92d" rel="cancel"/>
  <dap:link mediaType="application/vnd.restbucks+xml" uri="http://restbucks.com/payment/
    f932f92d"
        rel="payment"/>
  <dap:link mediaType="application/vnd.restbucks+xml" uri="http://restbucks.com/order/f932f92d"
        rel="update"/>
  <dap:link uri="http://restbucks.com/order/f932f92d" rel="latest"/>
  <rb:item>
    <rb:milk>semi</rb:milk>
    <rb:size>large</rb:size>
    <rb:drink>latte</rb:drink>
  </rb:item>
  <rb:location>takeaway</rb:location>
  <rb:cost>2.0</rb:cost>
  <rb:status>unpaid</rb:status>
</rb:order>
```

Confirm the Order

- GET from the rel="latest" URI
  - `http://restbucks.com/order/f932f92d`



Confirm the Order

- Request

```
GET /order/f932f92d HTTP/1.1
Accept: application/vnd.restbucks+xml
Host: restbucks.com
Connection: keep-alive
```

Confirm the Order

- Response

```
HTTP/1.1 200 OK
Location: http://restbucks.com/order/f932f92d
Content-Type: application/vnd.restbucks+xml
Content-Length: 828
Date: Sun, 06 Sep 2009 06:51:22 GMT

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<rb:order xmlns:rb="http://schemas.restbucks.com" xmlns:dap="http://schemas.restbucks.com/dap">
  <dap:link uri="http://restbucks.com/order/f932f92d" mediaType="application/vnd.restbucks+xml"
    rel="cancel"/>
  <dap:link mediaType="application/vnd.restbucks+xml" uri="http://restbucks.com/payment/f932f92d"
      rel="payment"/>
  <dap:link mediaType="application/vnd.restbucks+xml"
      uri="http://restbucks.com/order/f932f92d" rel="update"/>
  <dap:link mediaType="application/vnd.restbucks+xml" uri="http://restbucks.com/order/f932f92d"
     rel="latest"/>
  <rb:item>
    <rb:milk>semi</rb:milk>
    <rb:size>large</rb:size>
    <rb:drink>latte</rb:drink>
  </rb:item>
  <rb:location>takeaway</rb:location>
  <rb:cost>2.0</rb:cost>
  <rb:status>unpaid</rb:status>
</rb:order>
```

Change the Order

- POST new order to service-generated URI
  - `http://restbucks.com/order/f932f92d`

POST? Not PUT?

- PUT expects the whole resource state to be presented in the request
  - But our clients aren't responsible for generating hypermedia links
- PATCH would be better
  - It allows us to send diffs, but isn't part of the standard yet
- So POST is our only option
  - Compare this to our CRUD protocol in SOA 321

Change the Order

- Request

```
POST order/f932f92d HTTP/1.1
Content-Type: application/vnd.restbucks+xml
Accept: application/vnd.restbucks+xml
Host: restbucks.com
Connection: keep-alive
Content-Length: 278

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<rb:order xmlns:rb="http://schemas.restbucks.com">
  <rb:item>
    <rb:milk>semi</rb:milk>
    <rb:size>large</rb:size>
    <rb:drink>cappuccino</rb:drink>
  </rb:item>
  <rb:location>takeaway</rb:location>
</rb:order>
```

No Hypermedia Controls?

Change the Order

- Response

```
HTTP/1.1 200 OK
Location: http://restbucks.com/order/f932f92d
Content-Type: application/vnd.restbucks+xml
Content-Length: 828
Date: Sun, 06 Sep 2009 06:52:22 GMT

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<rb:order xmlns:rb="http://schemas.restbucks.com"
    xmlns:dap="http://schemas.restbucks.com/dap">
  <dap:link mediaType="application/vnd.restbucks+xml" uri="http://restbucks.com/order/f932f92d"
       rel="cancel"/>
  <dap:link mediaType="application/vnd.restbucks+xml"
       uri="http://restbucks.com/payment/f932f92d" rel="payment"/>
  <dap:link mediaType="application/vnd.restbucks+xml"
       uri="http://restbucks.com/order/f932f92d" rel="update"/>
  <dap:link mediaType="application/vnd.restbucks+xml" uri="http://restbucks.com/order/f932f92d"
       rel="latest"/>
  <rb:item>
    <rb:milk>semi</rb:milk>
    <rb:size>large</rb:size>
    <rb:drink>cappuccino</rb:drink>
  </rb:item>
  <rb:location>takeaway</rb:location>
  <rb:cost>2.0</rb:cost>
  <rb:status>unpaid</rb:status>
</rb:order>
```

Statelessness

- Remember interactions with resources are stateless
- The resource "forgets" about you while you're not directly interacting with it
- Which means race conditions are possible
- Use `If-Unmodified-Since` on a timestamp to make sure
  - Or use `If-Match` and an `ETag`
- You'll get a `412 PreconditionFailed` if you lost the race
  - But you'll avoid potentially putting the resource into some inconsistent state

Warning: Don't be Slow!

- Can only make changes until someone actually makes your drink
  - You're safe if you use `If-Unmodified-Since` or `If-Match`
  - But resource state can change without you!

- Request

```
POST /order/1234 HTTP 1.1
Host: restbucks.com
...
```

- Response

```
409 Conflict
```

Too slow! Someone else has changed the state of my order

- Request

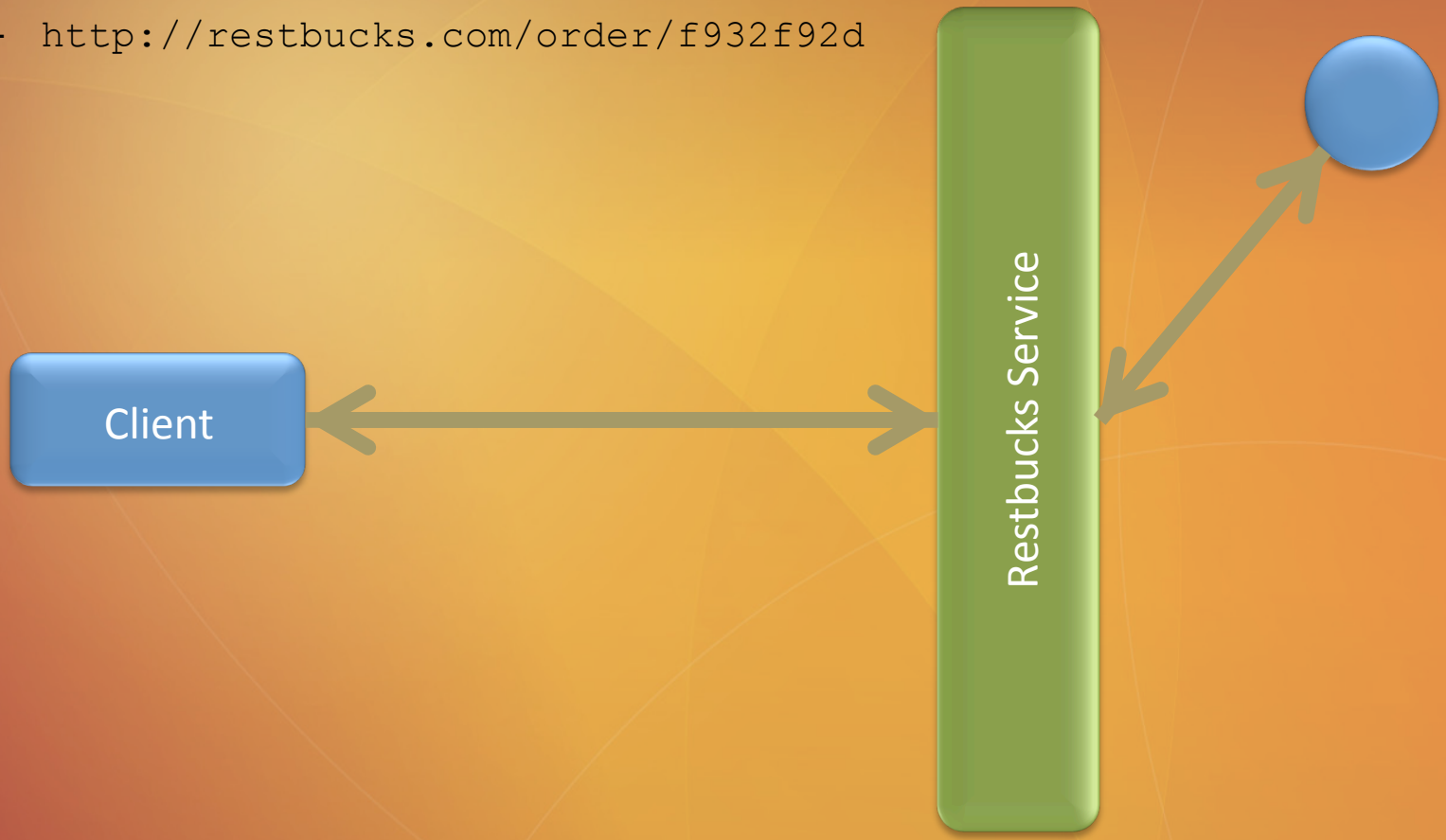
```
OPTIONS /order/1234 HTTP 1.1
Host: restbucks.com
```

- Response

```
Allow: GET
```

What if we want to cancel?

- DELETE order at order URI
  - `http://restbucks.com/order/f932f92d`



What if we want to cancel?

- Request

```
DELETE /order/f932f92d HTTP/1.1
Host: restbucks.com
Connection: keep-alive
```

- Response

```
HTTP/1.1 200 OK
Content-Type: application/vnd.restbucks+xml
Date: Sun, 06 Sep 2009 06:53:22 GMT
```

Order Payment

- PUT to service-generated payment URI
  - `http://restbucks.com/payment/f932f92d`

Why PUT this time?

- It's idempotent
  - Safe to repeat in the event of failures
- For all $ transactions, prefer idempotent verbs

Order Payment

- Request

```
PUT /payment/f932f92d HTTP/1.1
Content-Type: application/vnd.restbucks+xml
Accept: application/vnd.restbucks+xml
Host: restbucks.com
Connection: keep-alive
Content-Length: 269

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<payment xmlns="http://schemas.restbucks.com">
  <amount>2.0</amount>
  <cardholderName>Michael Farraday</cardholderName>
  <cardNumber>11223344</cardNumber>
  <expiryMonth>12</expiryMonth>
  <expiryYear>12</expiryYear>
</payment>
```

Order Payment

- Response

```
HTTP/1.1 201 Created
Location: http://restbucks.com/payment/f932f92d
Content-Type: application/vnd.restbucks+xml
Content-Length: 547
Date: Sun, 06 Sep 2009 06:51:22 GMT

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<rb:payment xmlns="http://schemas.restbucks.com/dap" xmlns:rb="http://
   schemas.restbucks.com">
  <link rel="latest" uri="http://restbucks.com/order/f932f92d" />
  <link rel="receipt" uri="http://restbucks.com/receipt/f932f92d" />
  <rb:amount>2.0</rb:amount>
  <rb:cardholderName>Michael Farraday</rb:cardholderName>
  <rb:cardNumber>11223344</rb:cardNumber>
  <rb:expiryMonth>12</rb:expiryMonth>
  <rb:expiryYear>12</rb:expiryYear>
</rb:payment>
```

What if we want to cancel now?

- Request

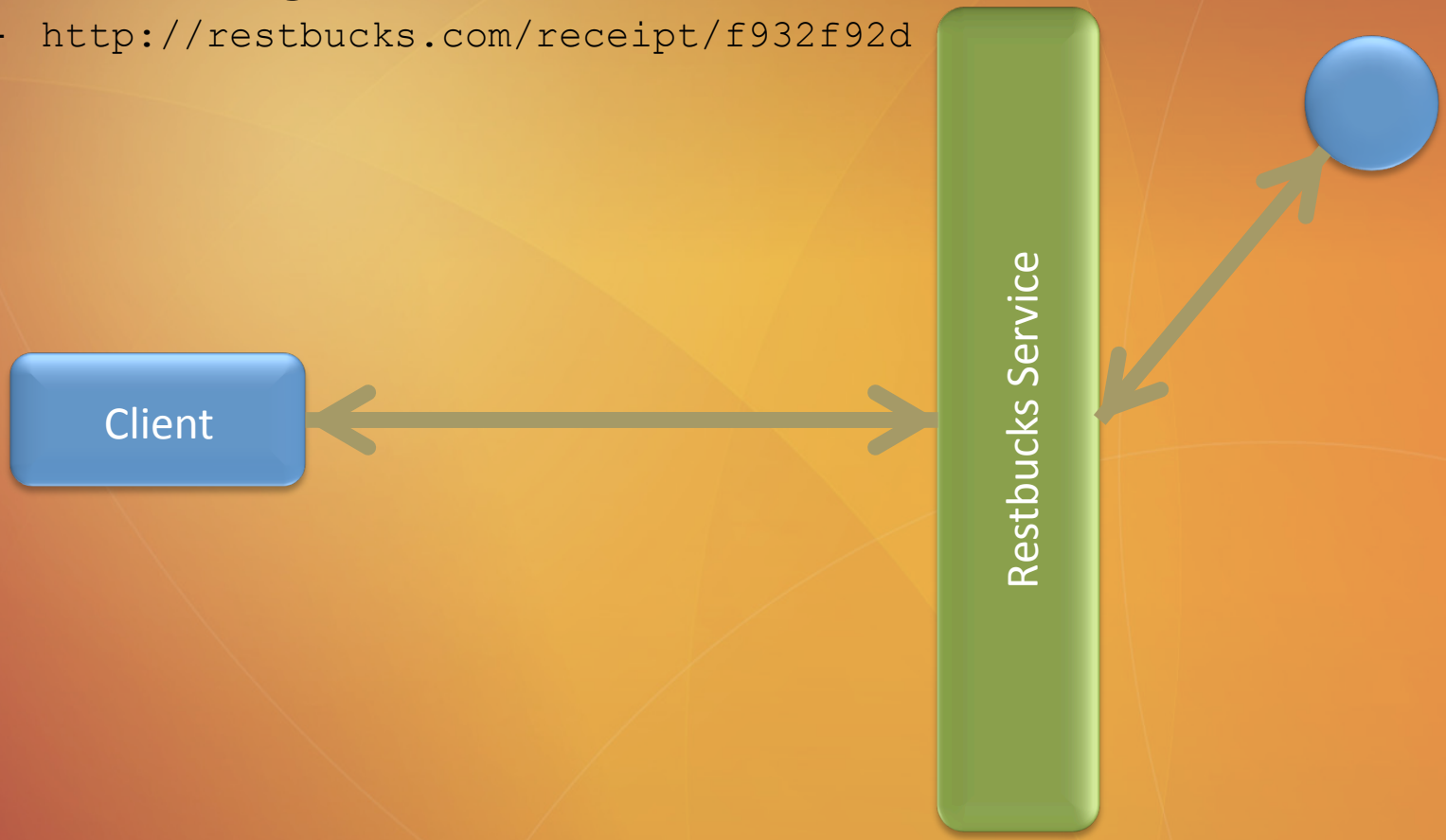
```
DELETE /order/f932f92d HTTP/1.1
Host: restbucks.com
Connection: keep-alive
```

- Response

```
HTTP/1.1 405 Method Not Allowed
Allow: GET
Content-Type: application/vnd.restbucks+xml
Content-Length: 0
Date: Sun, 06 Sep 2009 07:43:29 GMT
```

Grab a Receipt

- GET service-generated receipt URI
  - `http://restbucks.com/receipt/f932f92d`



Grab a Receipt

- Request

```
GET /receipt/f932f92d HTTP/1.1
Accept: application/vnd.restbucks+xml
Host: restbucks.com
Connection: keep-alive
```

Grab a Receipt

- Request

```
HTTP/1.1 200 OK
Content-Type: application/vnd.restbucks+xml
Content-Length: 332
Date: Sun, 06 Sep 2009 06:51:22 GMT

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<rb:receipt xmlns="http://schemas.restbucks.com/dap"
    xmlns:rb="http://schemas.restbucks.com">
  <link rel="order"
    uri="http://restbucks.com/order/f932f92d" />
  <rb:amount>2.0</rb:amount>
  <rb:paid>2009-09-06T16:51:22.814+10:00</rb:paid>
</rb:receipt>
```

Complete the Order

- GET from the rel="order" URI
  - `http://restbucks.com/order/f932f92d`

Complete the Order

- Request

```
GET /order/f932f92d HTTP/1.1
Host: restbucks.com
Connection: keep-alive
```

Complete the Order

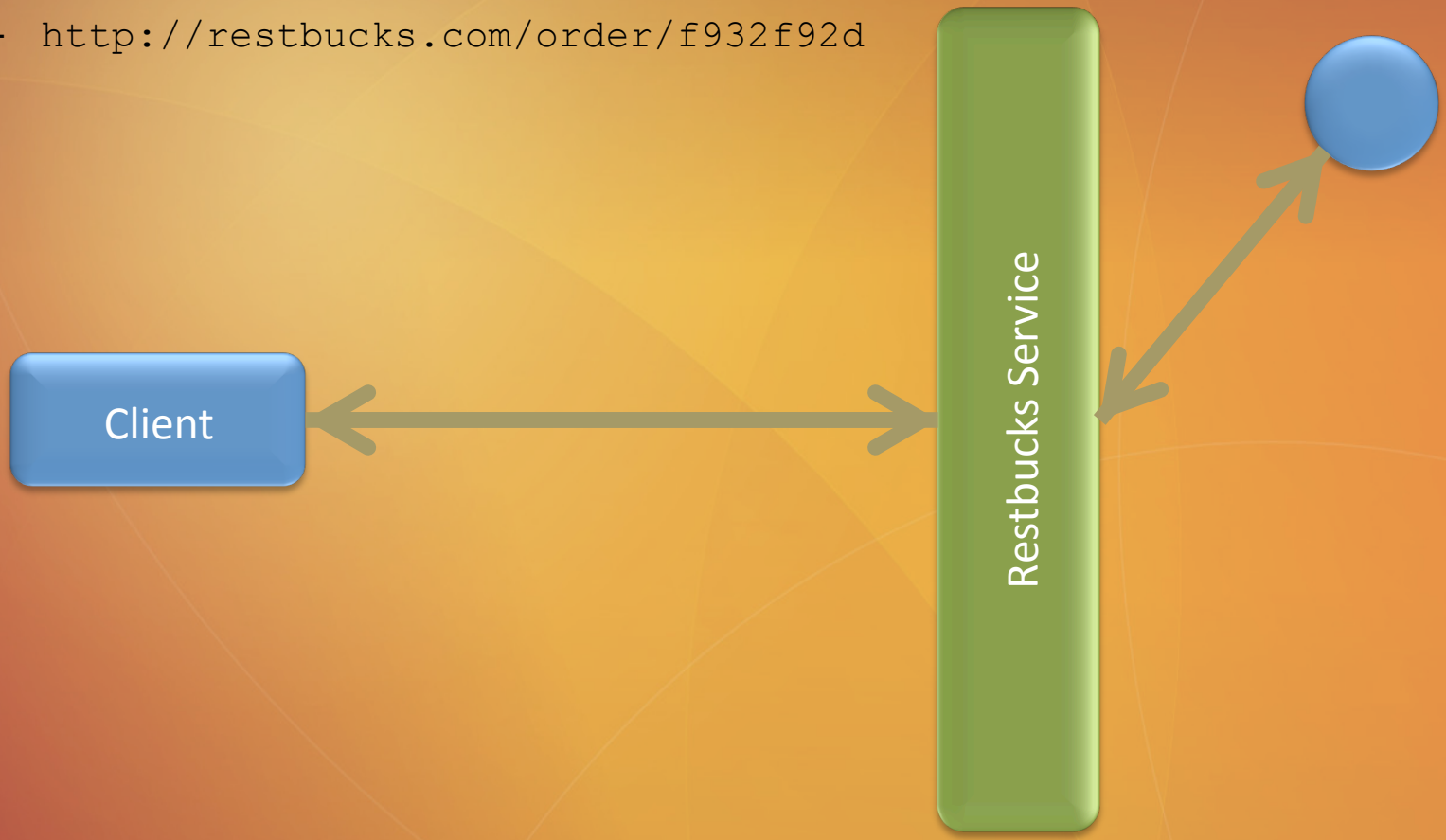
- Response

```
HTTP/1.1 200 OK
Content-Type: application/vnd.restbucks+xml
Content-Length: 455

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<rb:order xmlns:rb="http://schemas.restbucks.com"
     xmlns:dap="http://schemas.restbucks.com/dap">
  <dap:link mediaType="application/vnd.restbucks+xml"
       uri="http://restbucks.com/order/f932f92d" rel="latest"/>
  <rb:item>
    <rb:milk>semi</rb:milk>
    <rb:size>large</rb:size>
    <rb:drink>cappuccino</rb:drink>
    </rb:item>
  <rb:location>takeaway</rb:location>
  <rb:cost>2.0</rb:cost>
  <rb:status>ready</rb:status>
</rb:order>
```

Complete the Order

- DELETE order at rel="latest" URI
  - `http://restbucks.com/order/f932f92d`



Complete the Order

- Request

```
DELETE/order/f932f92d HTTP/1.1
Host: restbucks.com
Connection: keep-alive
```

Complete the Order

- Response

```
HTTP/1.1 200 OK
Content-Type: application/vnd.restbucks+xml
Content-Length: 360
Date: Sun, 06 Sep 2009 23:01:10 GMT

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<rb:order xmlns:rb="http://schemas.restbucks.com"
     xmlns:dap="http://schemas.restbucks.com/dap">
  <rb:item>
    <rb:milk>semi</rb:milk>
    <rb:size>large</rb:size>
    <rb:drink>cappuccino</rb:drink>
  </rb:item>
  <rb:location>takeaway</rb:location>
  <rb:cost>2.0</rb:cost>
   <rb:status>taken</rb:status>
</rb:order>
```

No Hypermedia Controls

Finally drink your coffee...

Source: http://images.businessweek.com/ss/06/07/top_brands/image/starbucks.jpg

What did we learn from Restbucks?

- HTTP has a header/status combination for every occasion
- APIs are expressed in terms of links, and links are great!
  - APP-esque APIs
- APIs can also be constructed with URI templates and inference
  - But beware tight coupling outside of CRUD services!
- XML is fine, but we could also use formats like Atom, JSON or even default to XHTML as a sensible middle ground
- State machines (defined by links) are important
  - Just as in Web Services…

Tech Interlude

Design Exercise: Adding HTTP Idioms to Restbucks

Ordering and Payments Protocol Weaknesses

- Lacks HTTP idioms
  - ETags
  - Cache metadata
  - Etc
- No circuit breaker
  - E.g. 413 at busy times

Ordering and Payments Protocol Retrospective

- Scenario:
  - The system is in production, and has delivered revenue successfully. What design decisions should we change for our next version?
- Four categories:
  - Introduce
    - New things we should do
  - Do more of
    - Good things we should do more of
  - Do less of
    - Poor things we should do less of
  - Remove
    - Stupid things we should cease

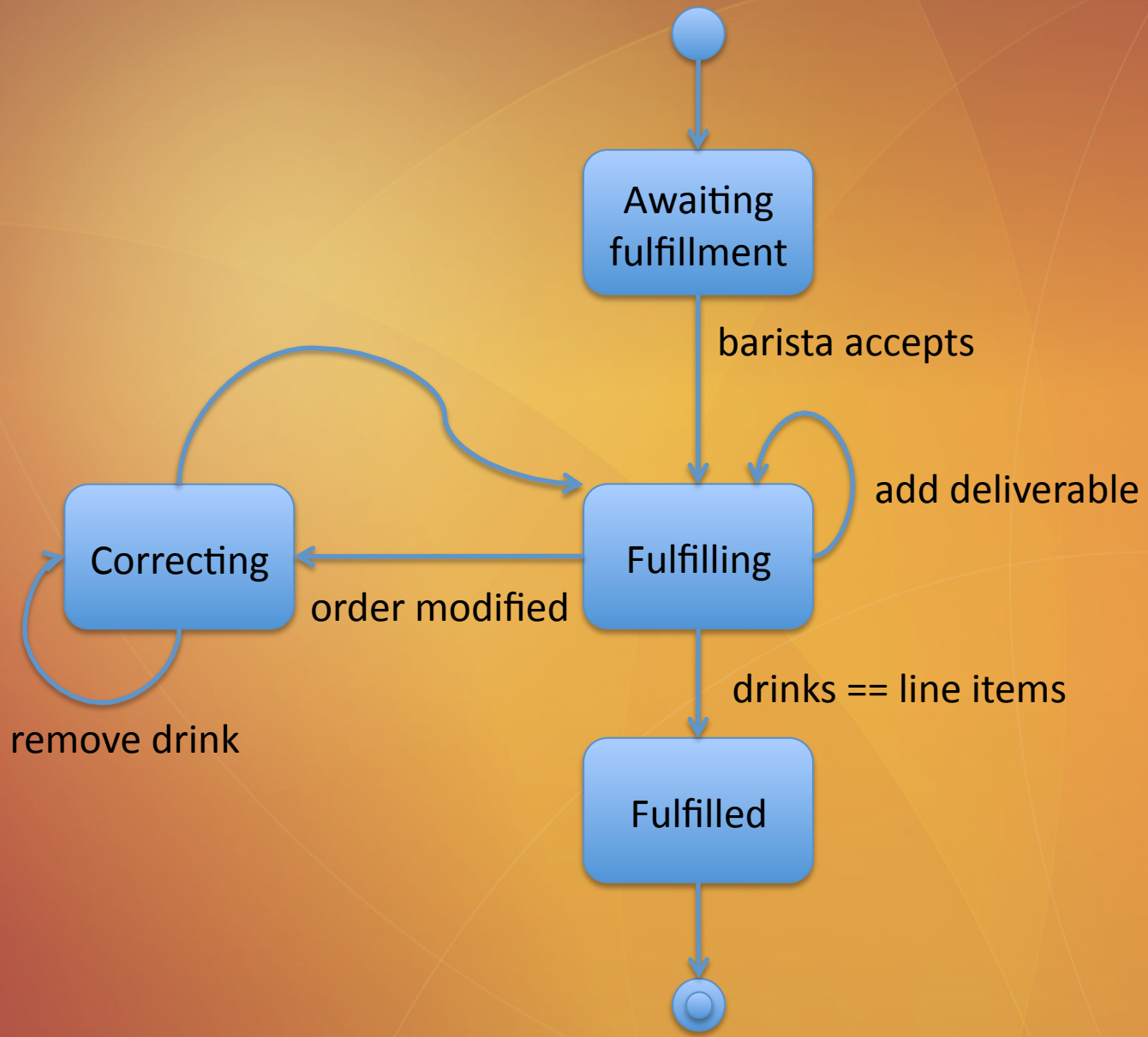
Tech Interlude

Hypermedia Design Walkthrough

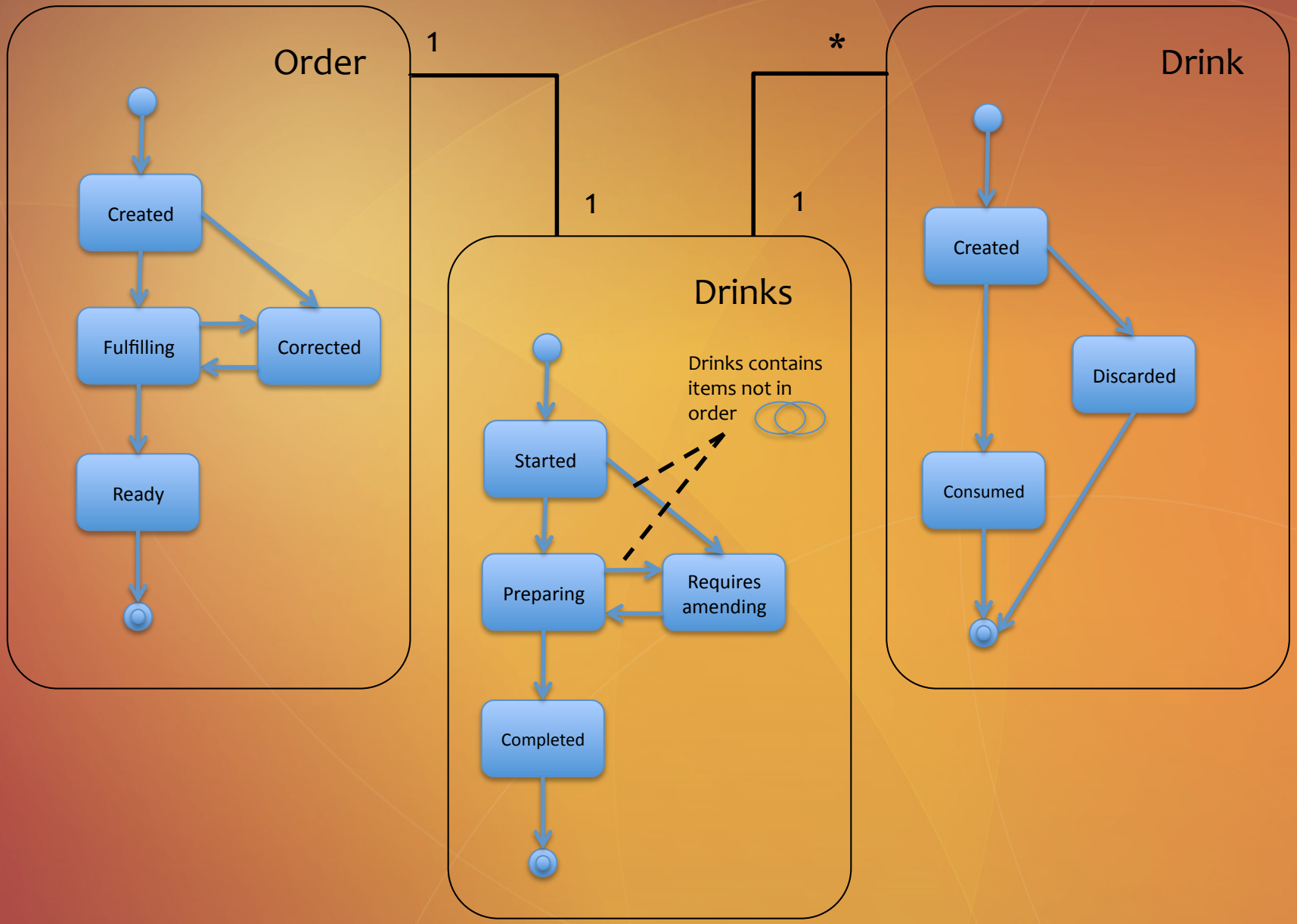
Problem

- What's the solution for integrating ordering and barista-ing systems?
- Guidelines:
  - Design applications in terms of application protocol state machines
  - Implement them in terms of resource lifecycles
  - Advertise/document them using media types, link relation values and HTTP idioms

Fulfillment domain application protocol



Resource lifecycles



Barista reviews outstanding orders

```
GET /orders HTTP/1.1
Host: restbucks.com
```

```
HTTP/1.1 200 OK
Date: ...
Content-Length: ...
Content-Type: application/vnd.restbucks+xml
```

```
<orders xmlns="http://schemas.restbucks.com/orders">
  <order xmlns="http://schemas.restbucks.com/order">
    <location>takeAway</location>
    <item>
      <name>latte</name>
      <quantity>1</quantity>
      <milk>whole</milk>
      <size>small</size>
    </item>
    <item>
      <name>cappuccino</name>
      <quantity>2</quantity>
      <milk>whole</milk>
      <size>small</size>
    </item>
    <status>created</status>
    <link
      rel="edit"
      type="application/vnd.restbucks+xml"
      href="http://restbucks.com/orders/123"/>
    <link
      rel="http://relations.restbucks.com/fulfillment"
      type="application/vnd.restbucks+xml"
      href="http://internal.restbucks.com/orders/123/drinks"/>
  </order>
  ...
</orders>
```

Barista reviews drinks status of first order

```
GET /orders/123/drinks HTTP/1.1
Host: internal.restbucks.com
```

```
HTTP/1.1 200 OK
Date: ...
Content-Length: ...
Content-Type: application/vnd.restbucks+xml
ETag: "1"
```

```
<drinks xmlns="http://schemas.restbucks.com/drinks">
  <link
    rel="http://relations.restbucks.com/fulfillment"
    type="application/vnd.restbucks+xml"
    href="http://internal.restbucks.com/orders/123/drinks"/>
  <link
    rel="http://relations.restbucks.com/order"
    type="application/vnd.restbucks+xml"
    href="http://internal.restbucks.com/orders/123"/>
  <status>started</status>
</drinks>
```

Barista creates latte

```
POST /orders/123/drinks HTTP/1.1
Host: internal.restbucks.com
If-Match: "1"
```

```
<drink xmlns="http://schemas.restbucks.com/drink">
  <name>latte</name>
  <milk>whole</milk>
  <size>small</size>
</drink>
```

```
HTTP/1.1 201 Created
Location: http://internal.restbucks.com/orders/123/drinks/1
Date: ...
Content-Length: ...
Content-Type: application/vnd.restbucks+xml
```

```
<drink xmlns="http://schemas.restbucks.com/drink">
  <name>latte</name>
  <milk>whole</milk>
  <size>small</size>
  <link
    rel="edit"
    type="application/vnd.restbucks+xml"
    href="http://internal.restbucks.com/orders/123/drinks/1"/>
</drink>
```

Barista reviews drinks status

```
GET /orders/123/drinks HTTP/1.1
Host: internal.restbucks.com
```

```
HTTP/1.1 200 OK
Date: ...
Content-Length: ...
Content-Type: application/vnd.restbucks+xml
ETag: "2"
```

```
<drinks xmlns="http://schemas.restbucks.com/drinks">
  <drink xmlns="http://schemas.restbucks.com/drink">
    <name>latte</name>
    <milk>whole</milk>
    <size>small</size>
    <link
      rel="edit"
      type="application/vnd.restbucks+xml"
      href="http://internal.restbucks.com/orders/123/drinks/1"/>
  </drink>
  <link
    rel="http://relations.restbucks.com/fulfillment"
    type="application/vnd.restbucks+xml"
    href="http://internal.restbucks.com/orders/123/drinks"/>
  <link
    rel="http://relations.restbucks.com/order"
    type="application/vnd.restbucks+xml"
    href="http://internal.restbucks.com/orders/123"/>
  <status>preparing</status>
</drinks>
```

Barista creates first cappuccino

```
POST /orders/123/drinks HTTP/1.1
Host: internal.restbucks.com
If-Match: "2"
```

```
<drink xmlns="http://schemas.restbucks.com/drink">
  <name>cappuccino</name>
  <milk>whole</milk>
  <size>small</size>
</drink>
```

```
HTTP/1.1 201 Created
Location: http://internal.restbucks.com/orders/123/drinks/2
Date: ...
Content-Length: ...
Content-Type: application/vnd.restbucks+xml
```

```
<drink xmlns="http://schemas.restbucks.com/drink">
  <name>cappuccino</name>
  <milk>whole</milk>
  <size>small</size>
  <link
    rel="edit"
    type="application/vnd.restbucks+xml"
    href="http://internal.restbucks.com/orders/123/drinks/2"/>
</drink>
```

Barista reviews drinks status

```
GET /orders/123/drinks HTTP/1.1
Host: internal.restbucks.com
```

```
HTTP/1.1 200 OK
Date: ...
Content-Length: ...
Content-Type: application/vnd.restbucks+xml
ETag: "3"
```

```
<drinks xmlns="http://schemas.restbucks.com/drinks">
  <drink xmlns="http://schemas.restbucks.com/drink">
    <name>latte</name>
    <milk>whole</milk>
    <size>small</size>
    <link rel="edit"
      type="application/vnd.restbucks+xml"
      href="http://internal.restbucks.com/orders/123/drinks/1"/>
  </drink>
  <drink xmlns="http://schemas.restbucks.com/drink">
    <name>cappuccino</name>
    <milk>whole</milk>
    <size>small</size>
    <link rel="edit"
      type="application/vnd.restbucks+xml"
      href="http://internal.restbucks.com/orders/123/drinks/2"/>
  </drink>
  <link rel="http://relations.restbucks.com/fulfillment"
    type="application/vnd.restbucks+xml"
    href="http://internal.restbucks.com/orders/123/drinks"/>
  <link rel="http://relations.restbucks.com/order"
    type="application/vnd.restbucks+xml"
    href="http://internal.restbucks.com/orders/123"/>
  <status>preparing</status>
</drinks>
```

Cashier reviews order status

```
GET /orders/123 HTTP/1.1
Host: restbucks.com
```

```
HTTP/1.1 200 OK
Date: ...
Content-Length: ...
Content-Type: application/vnd.restbucks+xml
```

```
<order xmlns="http://schemas.restbucks.com/order">
  <location>takeAway</location>
  <item>
    <name>latte</name>
    <quantity>1</quantity>
    <milk>whole</milk>
    <size>small</size>
  </item>
  <item>
    <name>cappuccino</name>
    <quantity>1</quantity>
    <milk>whole</milk>
    <size>large</size>
  </item>
  <status>fulfilling</status>
  <link
    rel="edit"
    type="application/vnd.restbucks+xml"
    href="http://restbucks.com/orders/123"/>
  <link
    rel="http://relations.restbucks.com/fulfillment"
    type="application/vnd.restbucks+xml"
    href="http://internal.restbucks.com/orders/123/drinks"/>
</order>
```

Cashier changes order

```
POST /orders/123 HTTP/1.1
Host: restbucks.com
Content-Type: application/vnd.restbucks+xml
Content-Length: ...
```

```
<order xmlns="http://schemas.restbucks.com/order">
  <location>takeAway</location>
  <item>
    <name>cappuccino</name>
    <quantity>2</quantity>
    <milk>whole</milk>
    <size>small</size>
  </item>
</order>
```

```
HTTP/1.1 200 OK
Date: ...
Content-Length: ...
Content-Type: application/vnd.restbucks+xml
```

```
<order xmlns="http://schemas.restbucks.com/order">
  <location>takeAway</location>
  <item>
    <name>cappuccino</name>
    <quantity>2</quantity>
    <milk>whole</milk>
    <size>small</size>
  </item>
  <status>fulfilling</status>
  <link
    rel="edit"
    type="application/vnd.restbucks+xml"
    href="http://restbucks.com/orders/123"/>
  <link
    rel="http://relations.restbucks.com/fulfillment"
    type="application/vnd.restbucks+xml"
    href="http://internal.restbucks.com/orders/123/drinks"/>
</order>
```

Barista continues making drinks

```
POST /orders/123/drinks HTTP/1.1
Host: internal.restbucks.com
If-Match: "3"
```

```
<drink xmlns="http://schemas.restbucks.com/drink">
  <name>cappuccino</name>
  <milk>whole</milk>
  <size>small</size>
</drink>
```

```
HTTP/1.1 412 Precondition Failed
```

Barista reviews drinks status

```
GET /orders/123/drinks HTTP/1.1
Host: internal.restbucks.com
```

```
HTTP/1.1 200 OK
Date: ...
Content-Length: ...
Content-Type: application/vnd.restbucks+xml
ETag: "4"
```

```
<drinks xmlns="http://schemas.restbucks.com/drinks">
  <drink xmlns="http://schemas.restbucks.com/drink">
    <unwanted/>
    <name>latte</name>
    <milk>whole</milk>
    <size>small</size>
    <link rel="edit"
      type="application/vnd.restbucks+xml"
      href="http://internal.restbucks.com/orders/123/drinks/1"/>
  </drink>
  <drink xmlns="http://schemas.restbucks.com/drink">
    <name>cappuccino</name>
    <milk>whole</milk>
    <size>small</size>
    <link rel="edit"
      type="application/vnd.restbucks+xml"
      href="http://internal.restbucks.com/orders/123/drinks/2"/>
  </drink>
  <link rel="http://relations.restbucks.com/fulfillment"
    type="application/vnd.restbucks+xml"
    href="http://internal.restbucks.com/orders/123/drinks"/>
  <link rel="http://relations.restbucks.com/order"
    type="application/vnd.restbucks+xml"
    href="http://internal.restbucks.com/orders/123"/>
  <status>requires-amending</status>
</drinks>
```

Barista discards latte

```
DELETE /orders/123/drinks/1
Host: internal.restbucks.com
```

```
HTTP/1.1 204 No Content
Date: ...
```

Barista reviews drinks status

```
GET /orders/123/drinks HTTP/1.1
Host: internal.restbucks.com
```

```
HTTP/1.1 200 OK
Date: ...
Content-Length: ...
Content-Type: application/vnd.restbucks+xml
ETag: "5"
```

```
<drinks xmlns="http://schemas.restbucks.com/drinks">
  <drink xmlns="http://schemas.restbucks.com/drink">
    <unwanted/>
    <name>cappuccino</name>
    <milk>whole</milk>
    <size>small</size>
    <link
      rel="edit"
      type="application/vnd.restbucks+xml"
      href="http://internal.restbucks.com/orders/123/drinks/2"/>
  </drink>
  <link
    rel="http://relations.restbucks.com/fulfillment"
    type="application/vnd.restbucks+xml"
    href="http://internal.restbucks.com/orders/123/drinks"/>
  <link
    rel="http://relations.restbucks.com/order"
    type="application/vnd.restbucks+xml"
    href="http://internal.restbucks.com/orders/123"/>
  <status>preparing</status>
</drinks>
```

Barista creates another cappuccino

```
POST /orders/123/drinks HTTP/1.1
Host: internal.restbucks.com
If-Match: "5"
```

```
<drink xmlns="http://schemas.restbucks.com/drink">
  <name>cappuccino</name>
  <milk>whole</milk>
  <size>small</size>
</drink>
```

```
HTTP/1.1 201 Created
Location: http://internal.restbucks.com/orders/123/drinks/3
Date: ...
Content-Length: ...
Content-Type: application/vnd.restbucks+xml
```

```
<drink xmlns="http://schemas.restbucks.com/drink">
  <name>cappuccino</name>
  <milk>whole</milk>
  <size>small</size>
  <link
    rel="edit"
    type="application/vnd.restbucks+xml"
    href="http://internal.restbucks.com/orders/123/drinks/3"/>
</drink>
```

Barista reviews drinks status

```
GET /orders/123/drinks HTTP/1.1
Host: internal.restbucks.com
```

```
HTTP/1.1 200 OK
Date: ...
Content-Length: ...
Content-Type: application/vnd.restbucks+xml
ETag: "6"
```

```
<drinks xmlns="http://schemas.restbucks.com/drinks">
  <drink xmlns="http://schemas.restbucks.com/drink">
    <name>cappuccino</name>
    <milk>whole</milk>
    <size>small</size>
    <link rel="edit"
      type="application/vnd.restbucks+xml"
      href="http://internal.restbucks.com/orders/123/drinks/2"/>
  </drink>
  <drink xmlns="http://schemas.restbucks.com/drink">
    <name>cappuccino</name>
    <milk>whole</milk>
    <size>small</size>
    <link rel="edit"
      type="application/vnd.restbucks+xml"
      href="http://internal.restbucks.com/orders/123/drinks/3"/>
  </drink>
  <link rel="http://relations.restbucks.com/fulfillment"
    type="application/vnd.restbucks+xml"
    href="http://internal.restbucks.com/orders/123/drinks"/>
  <link rel="http://relations.restbucks.com/order"
    type="application/vnd.restbucks+xml"
    href="http://internal.restbucks.com/orders/123"/>
  <status>completed</status>
</drinks>
```

Cashier reviews order status

```
GET /orders/123 HTTP/1.1
Host: restbucks.com
```

```
HTTP/1.1 200 OK
Date: ...
Content-Length: ...
Content-Type: application/vnd.restbucks+xml
```

```
<order xmlns="http://schemas.restbucks.com/order">
  <location>takeAway</location>
  <item>
    <name>cappuccino</name>
    <quantity>2</quantity>
    <milk>whole</milk>
    <size>small</size>
  </item>
  <status>ready</status>
</order>
```

The documented protocol

Entry point
http://restbucks.com/orders

Media type
Application/vnd.restbucks+xml

Resources

Orders	http://schemas.restbucks.com/orders	<status> element indicates the order status. Values are: created, fulfilling, ready.
Order	http://schemas.restbucks.com/order	
Drinks	http://schemas.restbucks.com/drinks	<status> element indicates the progress of the prepared drinks. Values are: started, requires-amending, preparing, completed.
Drink	http://schemas.restbucks.com/drink	

Fulfilment is complete when the order status is ready, and the drinks status is completed.

Link relations

edit	Refers to a resource that can be used to edit the link's context. To edit an order, POST a new order to the edit link. To remove a drink from a list of drinks, DELETE the linked resource.
http://relations.restbucks.com/fulfillment	Drinks that fulfil an order. The linked resource is accompanied by an ETag. Creating new drinks should be done by conditionally POSTing to the linked resource. If the list of drinks requires amending before a new drink can be added, the service responds with 412 Precondition Failed.
http://relations.restbucks.com/order	The order to which a list of drinks belongs.

Scalability

Enterprise Scalability?


Web Scale!

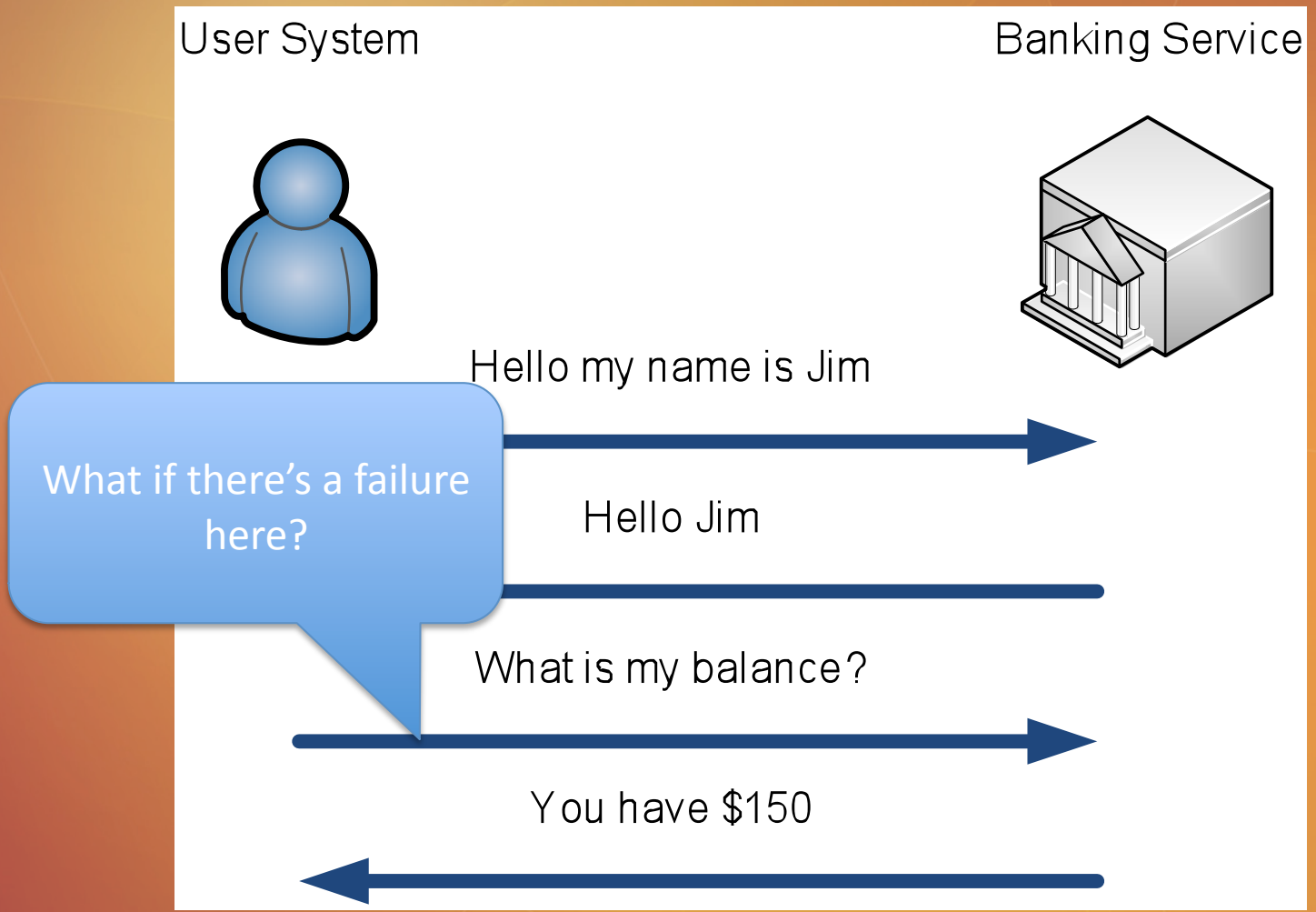
Statelessness

- Every action happens in isolation
  - This is a good thing!
- In between requests the server knows nothing about you
  - Excepting any state changes you caused when you last interacted with it.
- Keeps the interaction protocol simpler
  - Makes recovery, scalability, failover much simpler too
  - Avoid cookies!

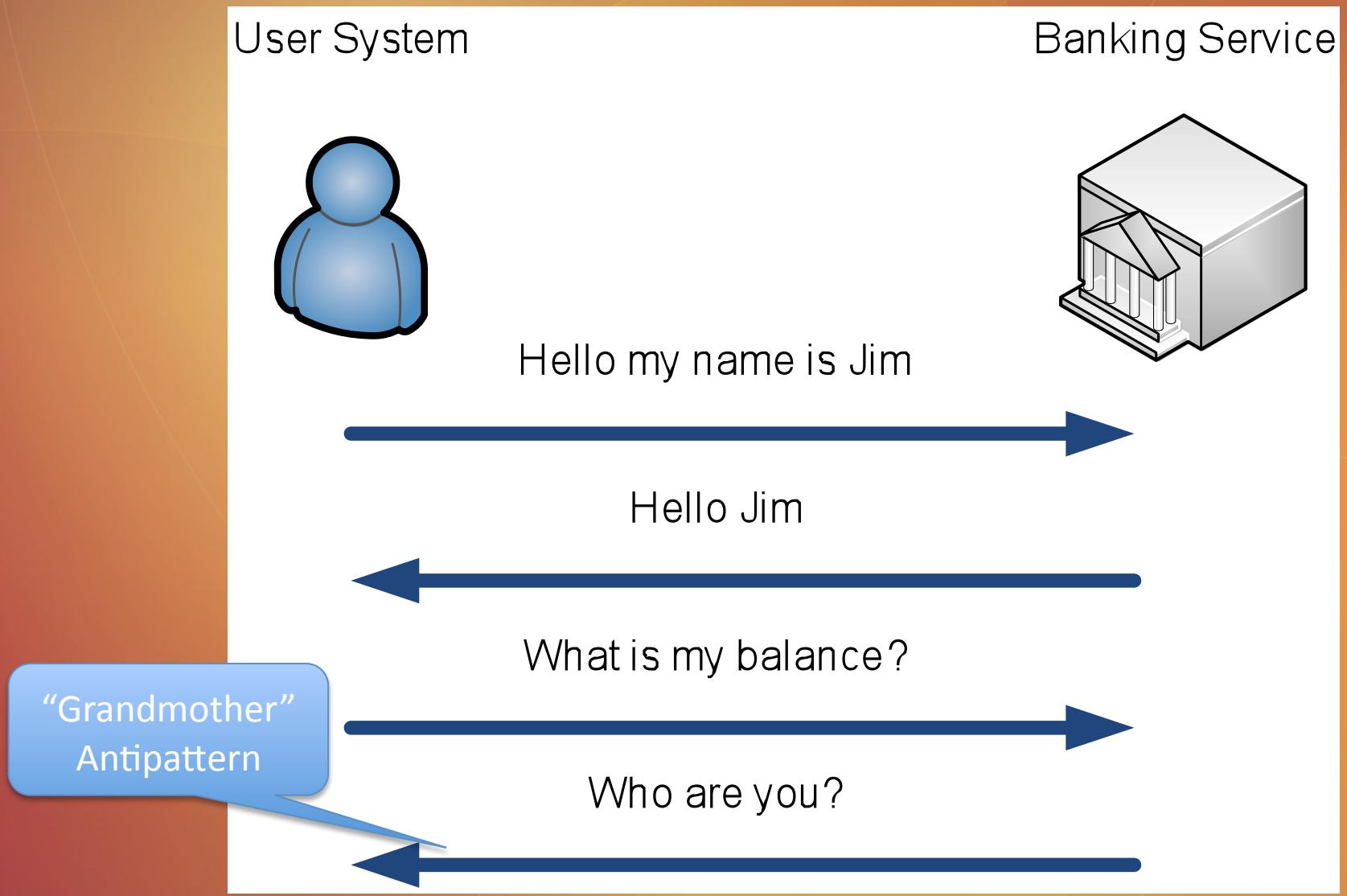
Application vs Resource State

- Useful services hold persistent data – Resource state
  - Resources are buckets of state
  - What use is Google without state?
- Brittle implementations have application state
  - They support long-lived conversations
  - No failure isolation
  - Poor crash recovery
  - Hard to scale, hard to do fail-over fault tolerance
- Recall stateless Web Services – same applies in the Web too!

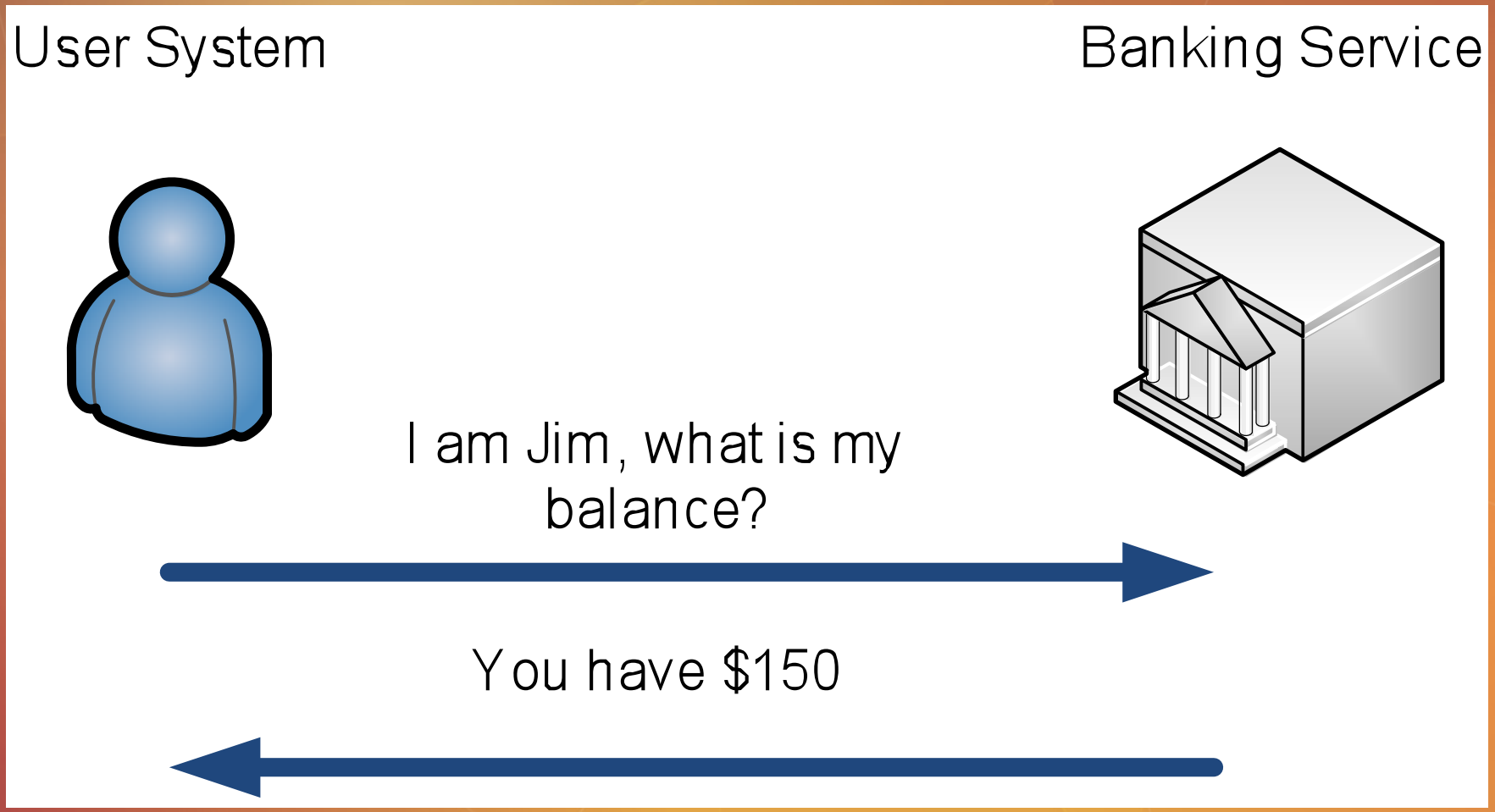
Stateful Example



Stateful Failure



Stateless System Tolerates Intermittent Failures

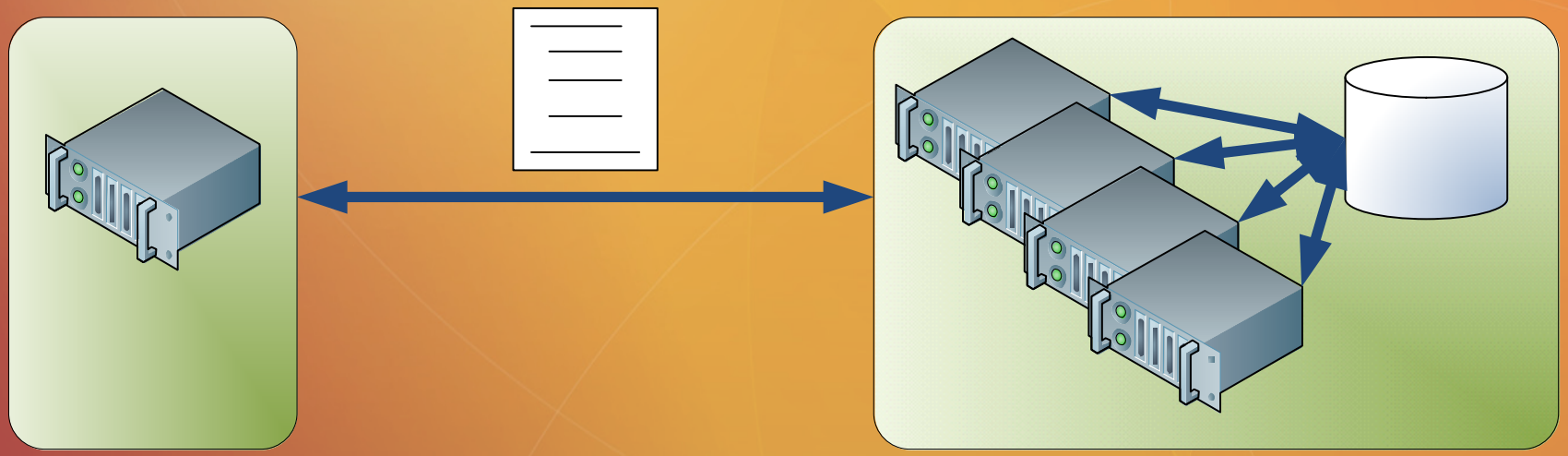


Scaling Horizontally

- Web farms have delivered horizontal scaling for years
  - Though they sometimes do clever things with session affinity to support cookie-based sessions
- In the programmatic Web, statelessness enables scalability
  - Just like in the Web Services world

Scalable Deployment Configuration

- Deploy services onto many servers
- Services are stateless
  - No cookies!
- Servers share only back-end data



Scaling Vertically… without servers

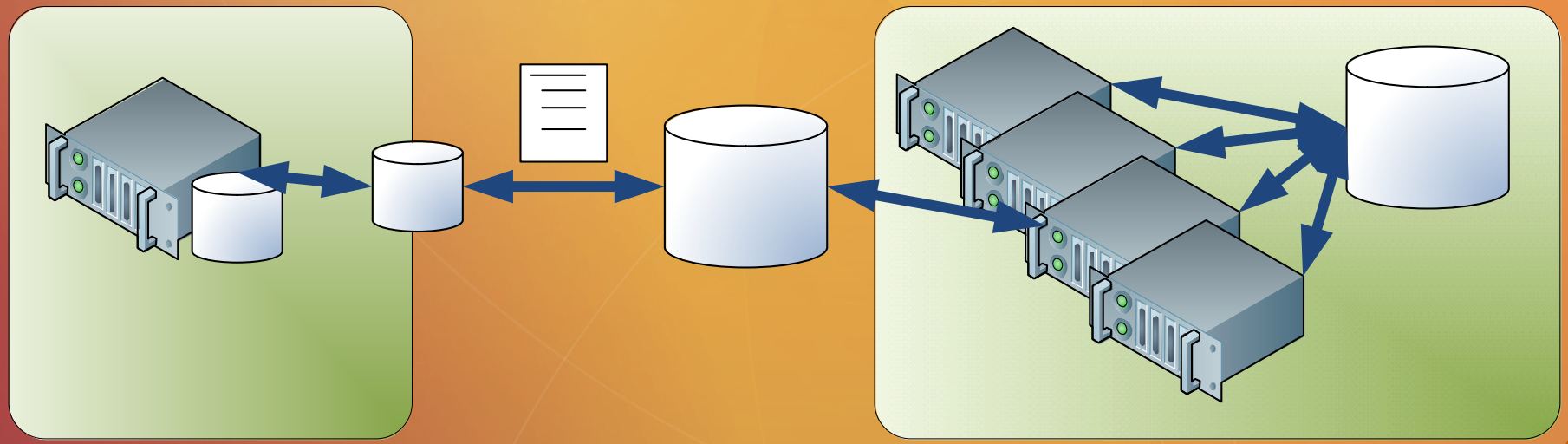
- The most expensive round-trip:
  - From client
  - Across network
  - Through servers
  - Across network again
  - To database
  - And all the way back!
- The Web tries to short-circuit this
  - By determining early if there is any actual work to do!
  - And by caching

Caching

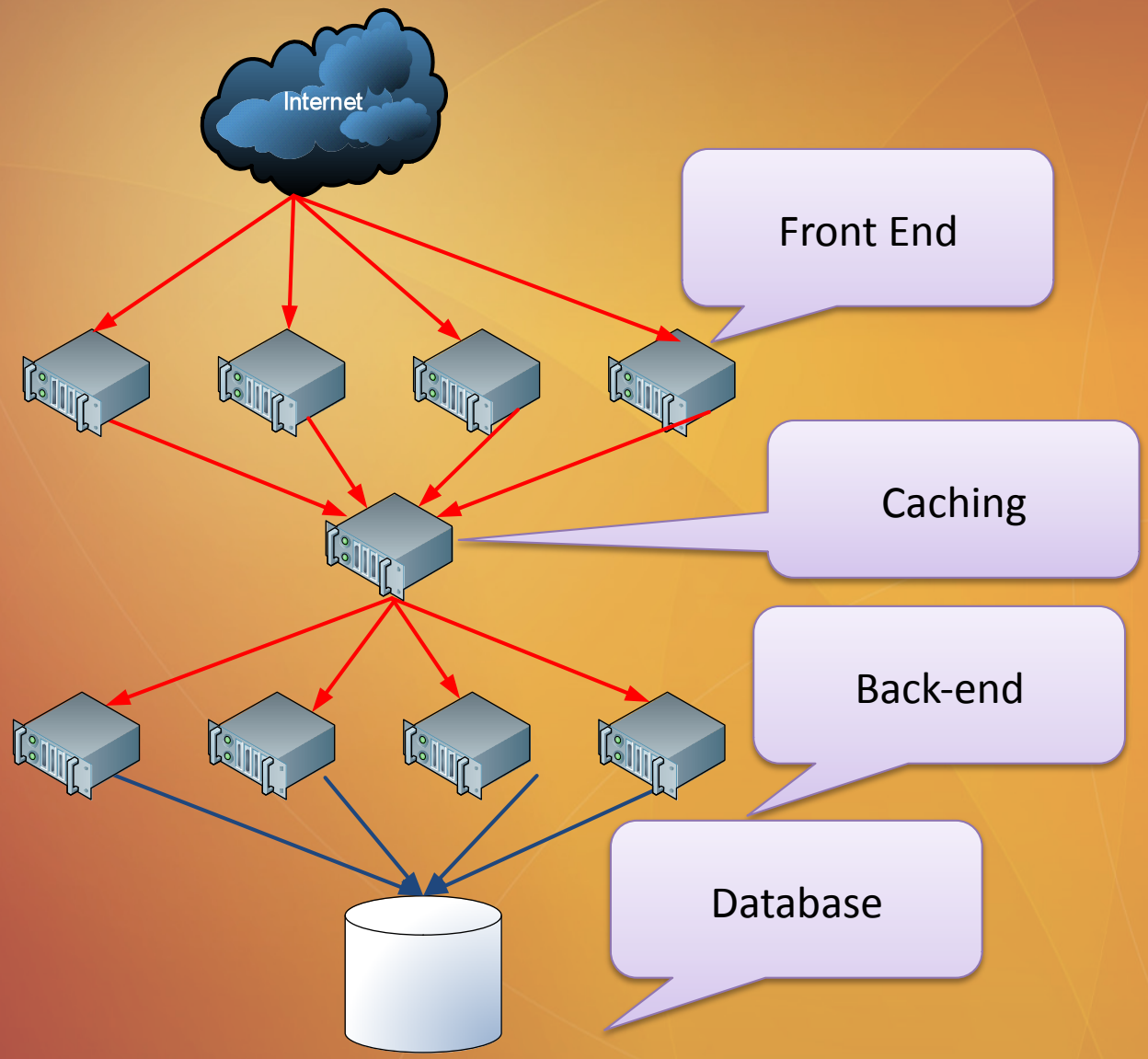
- Caching is about scaling vertically
  - As opposed to horizontally
- Making a single service run faster
  - Rather than getting higher overall throughput
- In the programmatic Web it's about reducing load on servers
  - And reducing latency for clients

Caching in a Scalable Deployment

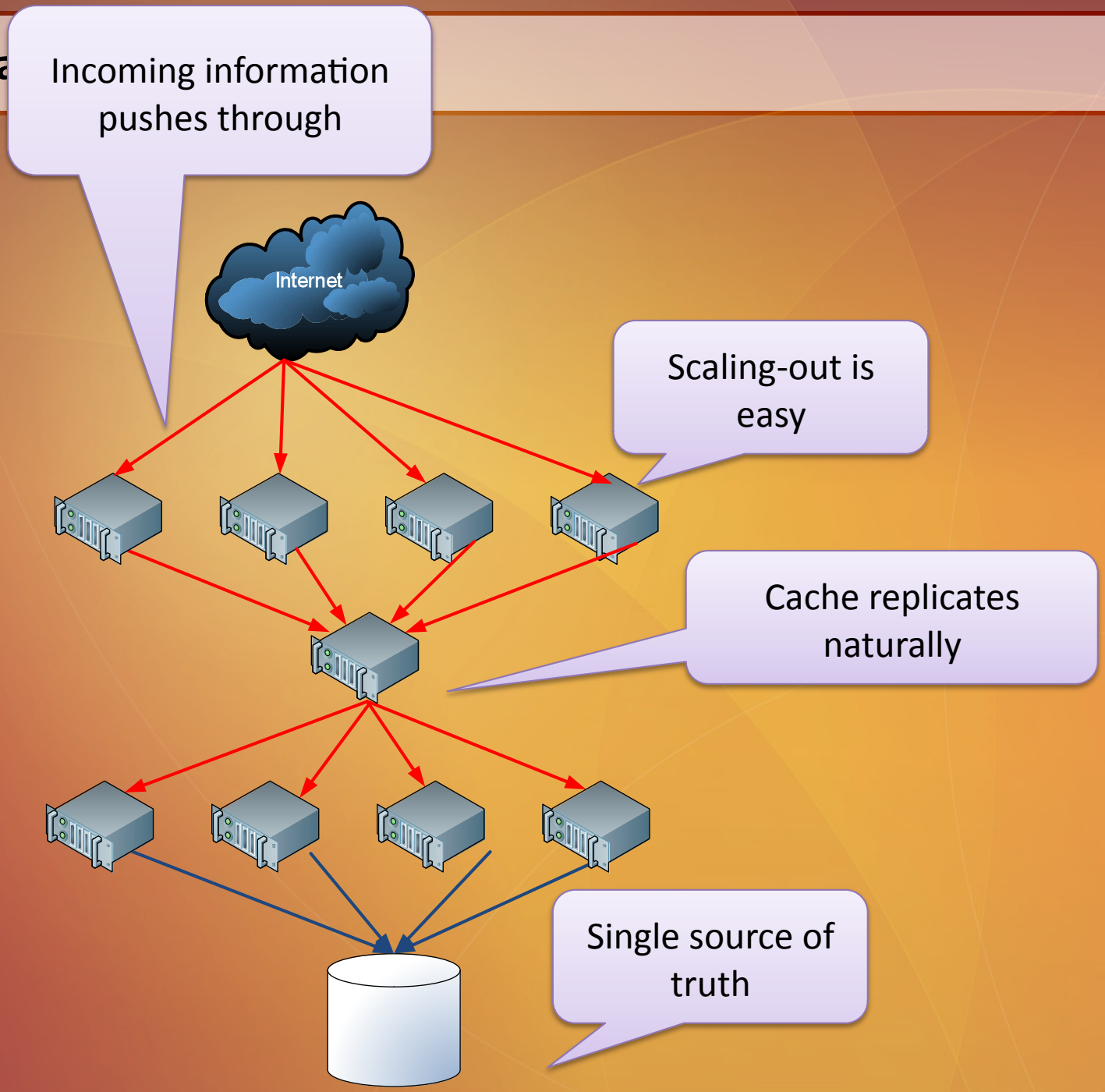
- Cache (reverse proxy) in front of server farm
  - Avoid hitting the server
- Proxy at client domain
  - Avoid leaving the LAN
- Local cache with client
  - Avoid using the network



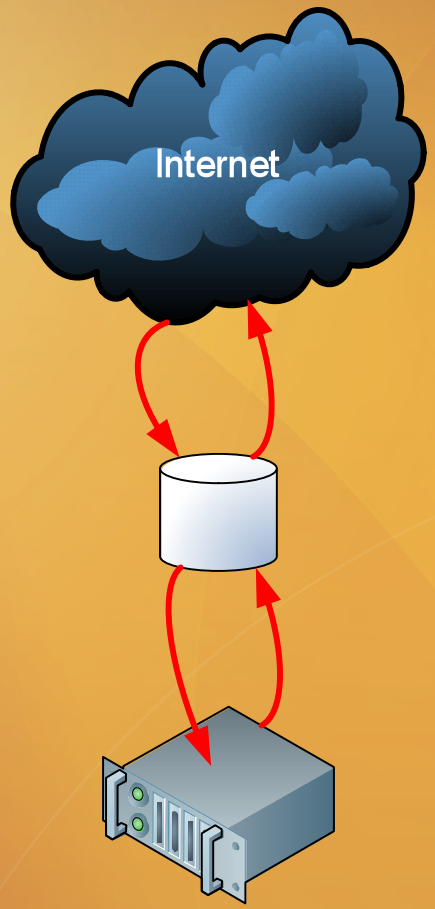
Web-Inspired Architecture



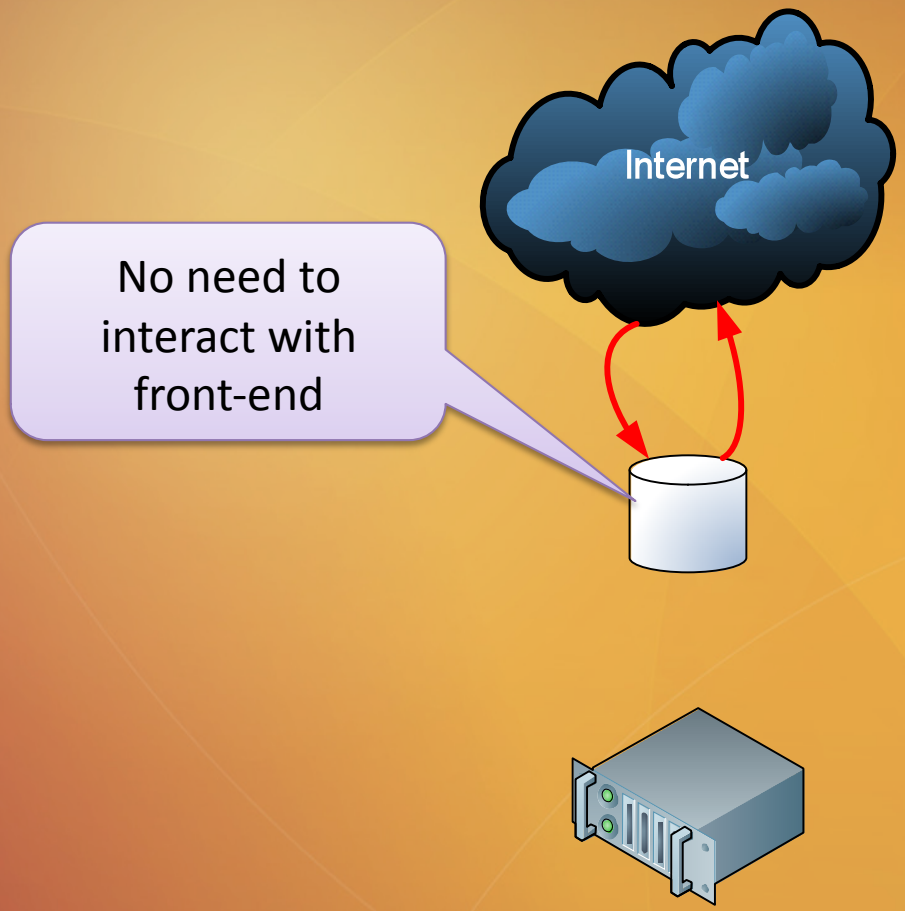
Cha


Incoming information pushes through
Internet
Scaling-out is easy
Cache replicates naturally
Single source of truth

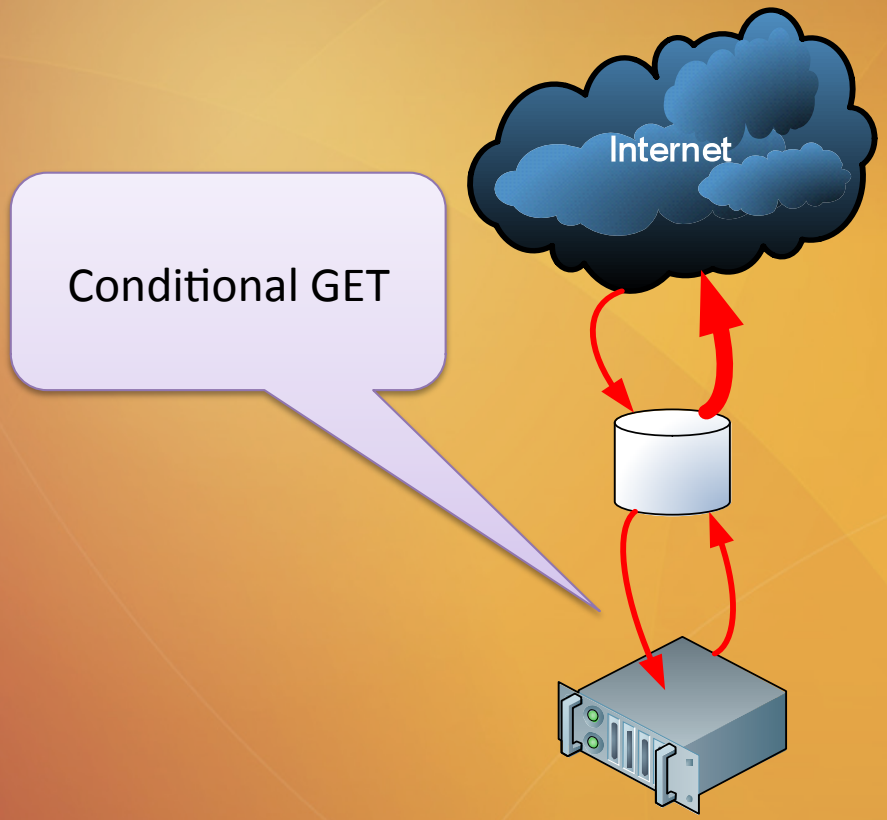
Improving Performance with Caching



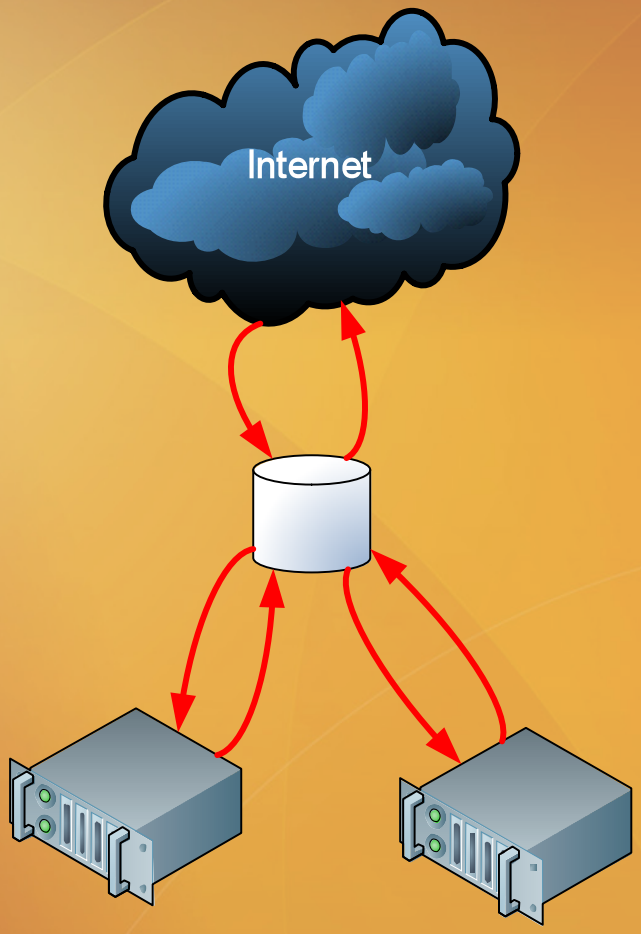
Freshness



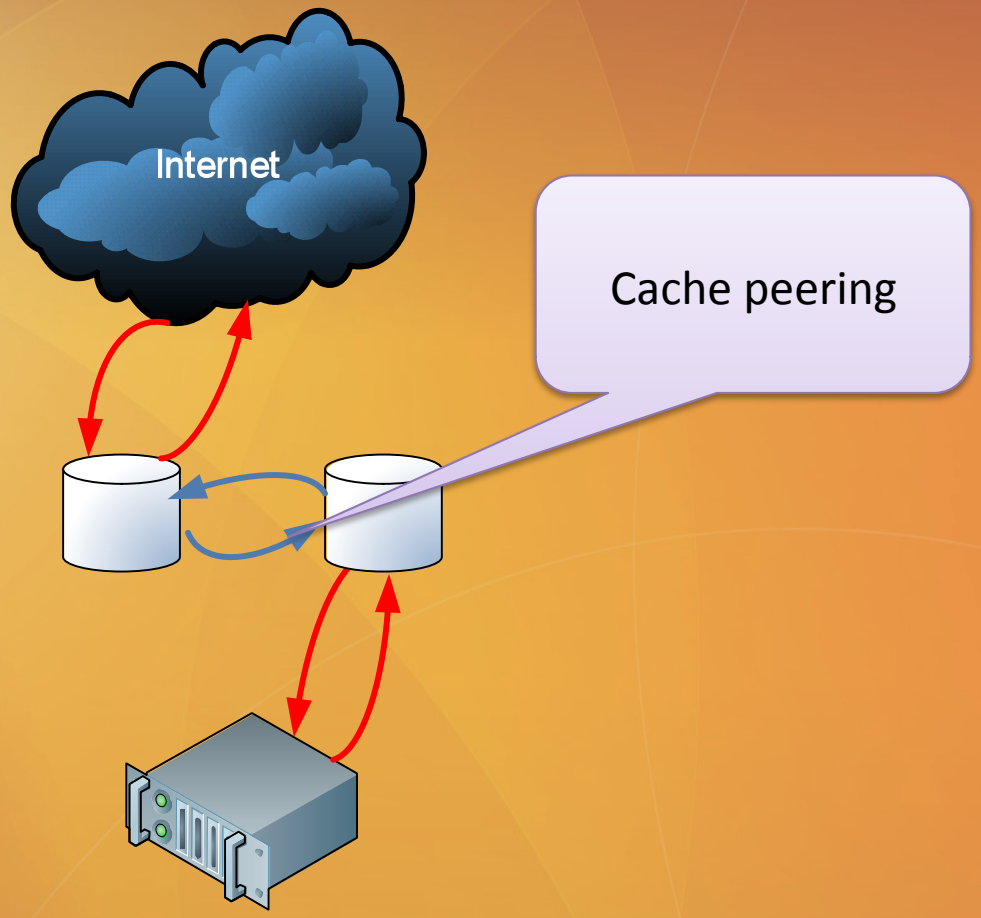
Validation



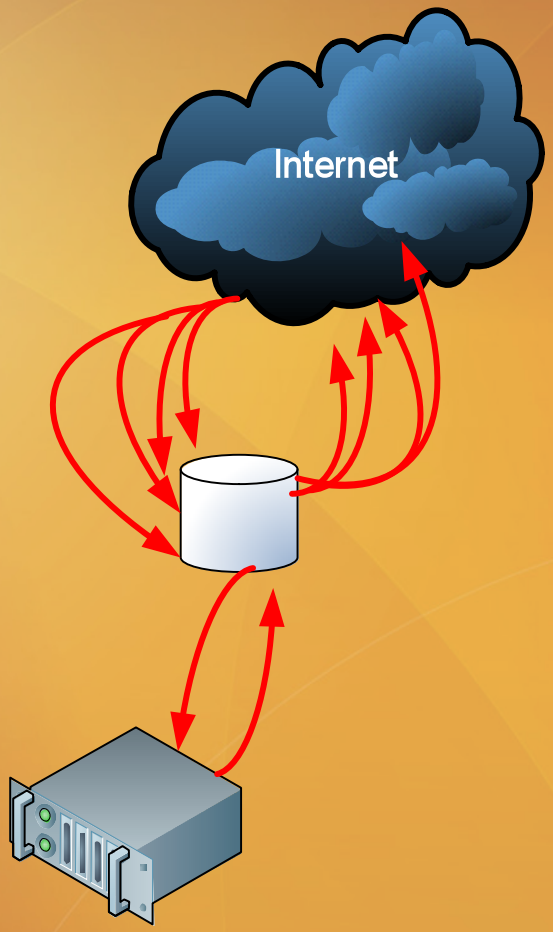
Scale out!



Scale the cache out!



Collapsed Forwarding



Being workshy is a good thing!

- Provide guard clauses in requests so that servers can determine easily if there's any work to be done
  - Caches too
- Use headers:
  - `If-Modified-Since`
  - `If-None-Match`
  - And friends
- Web infrastructure uses these to determine if its worth performing the request
  - And often it isn't
  - So an existing representation can be returned

Conditional GET Avoids Work!

ETag is an opaque identifier for specific resource state

- Bandwidth-saving pattern
- Requires client and server to work together
- Server sends `Last-Modified` and/or `ETag` headers with representations
- Client sends back those values when it interacts with resource in `If-Modified-Since` and/or `If-None-Match` headers
- Server responds with a 200 an empty body if there have been no updates to that resource state
- Or gives a new resource representation (with new `Last-Modified` and/or `ETag` headers)

Retrieving a Resource Representation

- Request

```
GET /orders/1234 HTTP 1.1
Host: restbucks.com
Accept: application/vnd.restbucks+xml
If-Modified-Since: 2009-01-08T15:00:34Z
If-None-Match: aabd653b-65d0-74da-bc63-4bca-ba3ef3f50432
```

- Response

```
200 OK
Content-Type: application/vnd.restbucks+xml
Content-Length: ...
Last-Modified: 2009-01-08T15:10:32Z
Etag: abbb4828-93ba-567b-6a33-33d374bcad39

<order … />
```

Not Retrieving a Resource Representation

- Request

```
GET /orders/1234 HTTP 1.1
Host: restbucks.com
Accept: application/vnd.restbucks+xml
If-Modified-Since: 2009-01-08T15:00:34Z
If-None-Match: aabd653b-65d0-74da-
  bc63-4bca-ba3ef3f50432
```

- Response

```
HTTP/1.1 304 Not Modified
```

Client's representation of the resource is up-to-date

Works with other verbs too

```
PUT /orders/1234 HTTP 1.1
Host: restbucks.com
Accept: application/vnd.restbucks+xml
If-Modified-Since: 2007-07-08T15:00:34Z
If-None-Match: aabd653b-65d0-74da-
  bc63-4bca-ba3ef3f50432

<order …/>
```

PUT Results in no Work Done

```
200 OK
Content-Type: application/xml
Content-Length: ...
Last-Modified: 2007-07-08T15:00:34Z
Etag: aabd653b-65d0-74da-bc63-4bca-
  ba3ef3f50432
```

Security

Good Ole' HTTP Authentication

- HTTP Basic and Digest Authentication: IETF RFC 2617
- Have been around since 1996 (Basic)/1997 (Digest)
- Pros:
  - Respects Web architecture:
    - stateless design (retransmit credentials)
    - headers and status codes are well understood
  - Does not prohibit caching (set `Cache-Control` to `public`)
- Cons:
  - Basic Auth must be used with SSL/TLS (plaintext password)
  - Not ideal for the human Web – no standard logout
  - Only one-way authentication (client to server)

HTTP Basic Auth Example

1. Initial HTTP request to protected resource

```
GET /index.html HTTP/1.1
Host: example.org
```

2. Server responds with

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Basic realm="MyRealm"
```

3. Client resubmits request

```
GET /index.html HTTP/1.1
Host: example.org
Authorization: Basic Qm9iCnBhc3N3b3JkCg==
```

Further requests with same or deeper path can include the additional `Authorization` header preemptively

HTTP Digest Difference

- Server reply to first client request:

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Digest
     realm=myrealm@example.org,
     qop="auth,auth-int",
     nonce="a97d8b710244df0e8b11d0f600bfb0cdd2",
     opaque="8477c69c403ebaf9f0171e9517f347f2"
```

- Client response to authentication challenge:

```
Authorization: Digest
     username="bob",
     realm=myrealm@example.org,
     nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
     uri="/index.html",
     qop=auth, nc=00000001, cnonce="0a6f188f",
     response="56bc2ae49393a6589745097850 7ff442",
     opaque="8477c69c403ebaf9f0171e9517f347f2"
```

WSSE Authentication

- Driven from the Atom community
- Use the WS-Security Username Token profile mapped to HTTP headers
- Doesn’t pass sensitive data in clear text
- Does require both sides to know a shared secret

Man-in-the-Middle

- All HTTP Authentication schemes can be hijacked by a man-in-the-middle attack
- Can intercept a Digest response from a service and change it into a Basic challenge
- Basic is easy to crack, attacker learns the password
- **Transport level security considered mandatory when you're using HTTP authentication of any variety**

SSL / TLS

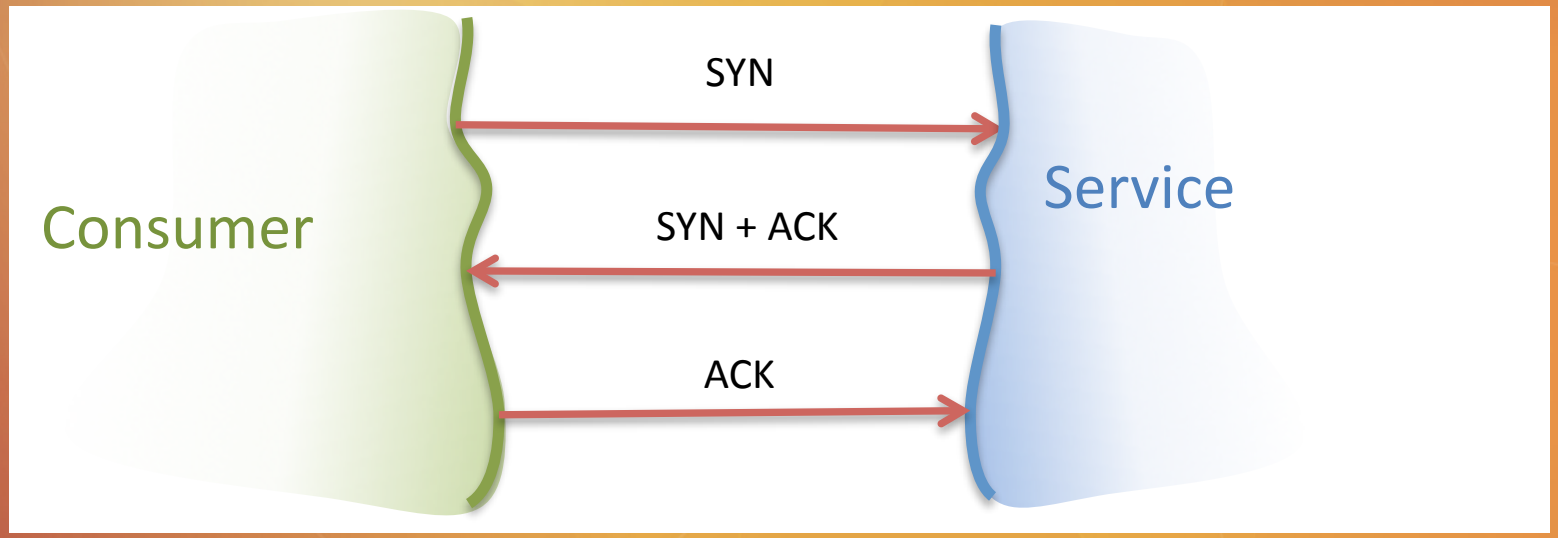
- “Strong” server and optional client authentication, confidentiality and integrity protection
- The only feasible way to secure against man-in-the-middle attacks
- Not broken! Even if some people like to claim otherwise
- Not very cache friendly though…

Transport Level Security

- TLS is the successor to Netscape's original SSL
- Tightened up some security loopholes
- Now under IETF's stewardship
  - RFC 2246
  - RFC 5246
- Provides a secure channel between client and server
  - Authenticated
  - Identified (bilateral too)
  - Confidential
  - Integrity assured

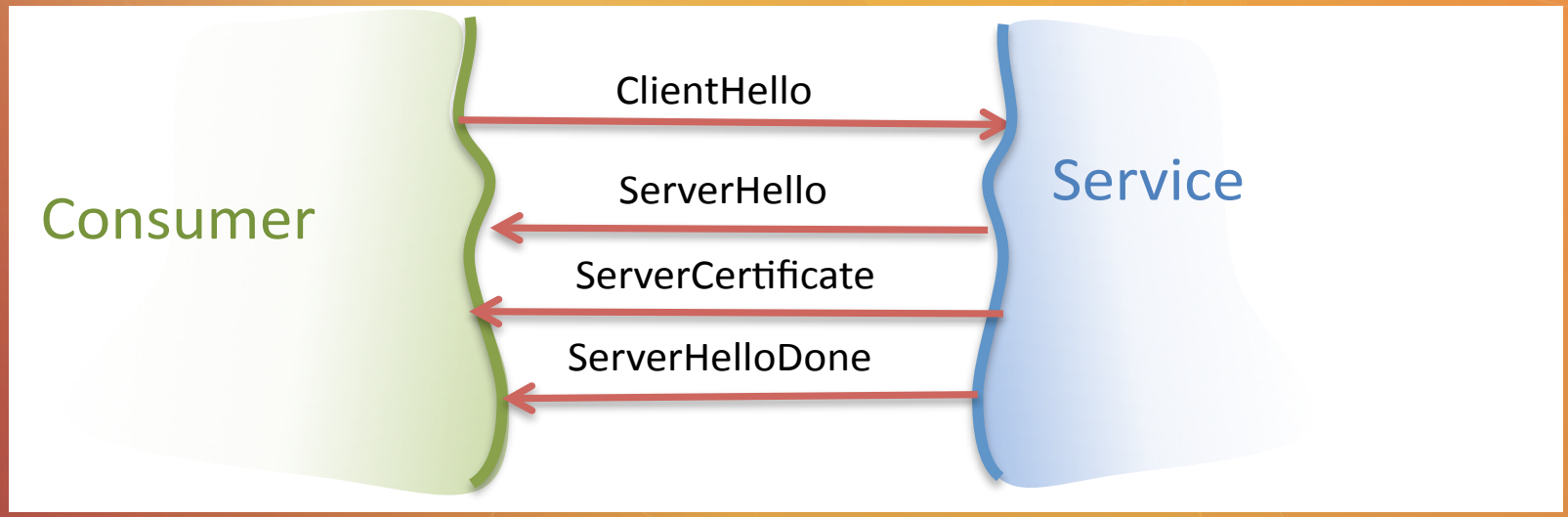
Start with a TCP Handshake

- Classic handshake the underpins HTTP connections
- Not secure at this point, obviously



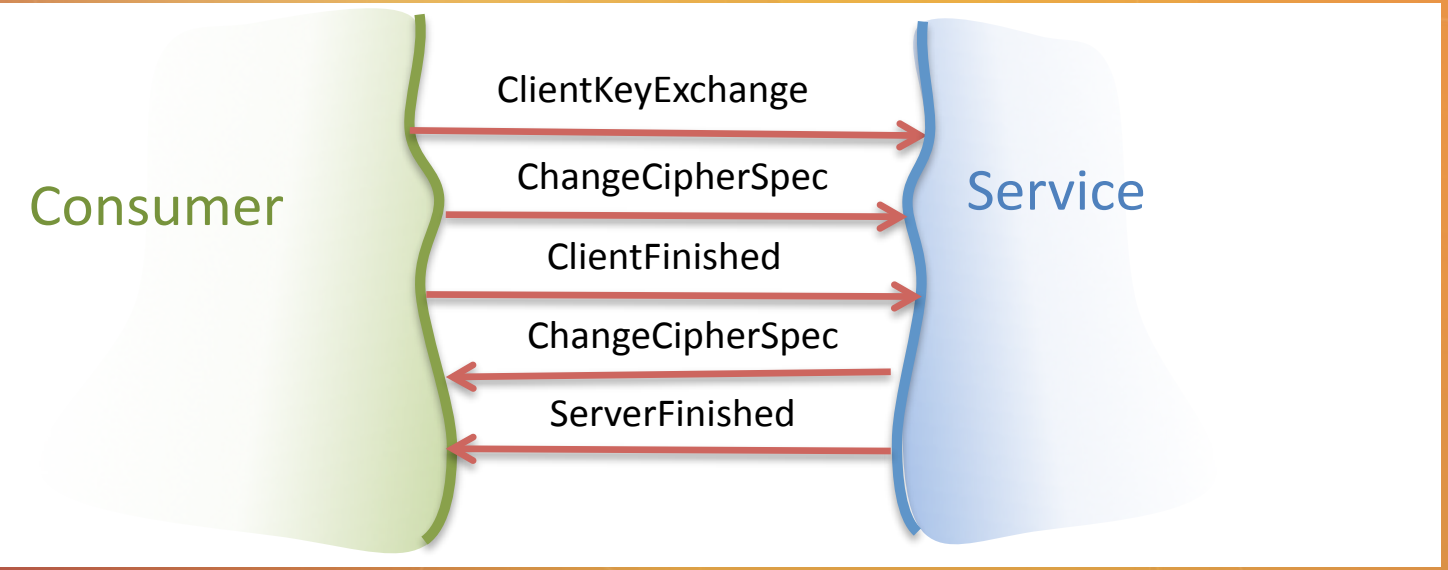
Negotiate some cryptographic options

- Client sends its hashing/encryption capabilities and preferences to the server
- Server responds with its choices from those the client presented
  - Preferring highest levels of security
- Server sends its certificate
  - Public key, CA
- Server indicates that it's complete



Switch on the crypto

- Client uses server's public key to send a PreMasterKey to the server
  - If the server is authenticated, it can decrypt this secret with its private key
- Client ChangeCipherSpec flips client onto the secure channel
- ClientFinished message sends a hash of the entire conversation
  - To alleviate any possibility of missing/tampered messages
- Server ChangeCipherSpec flips server onto the secure channel
- ServerFinished sends a hash of the entire conversation to the client



Network and performance considerations

- Just use HTTP everywhere?
- Reduces options for caching
  - Reverse proxies and client-side caching only
- Expensive to set up connections
  - Though relatively cheap to maintain, if you have enough sockets
- Securing a channel on the risk/value profile of a resource
  - Secure channels only for high value/risk resources
- Can use a hybrid approach…

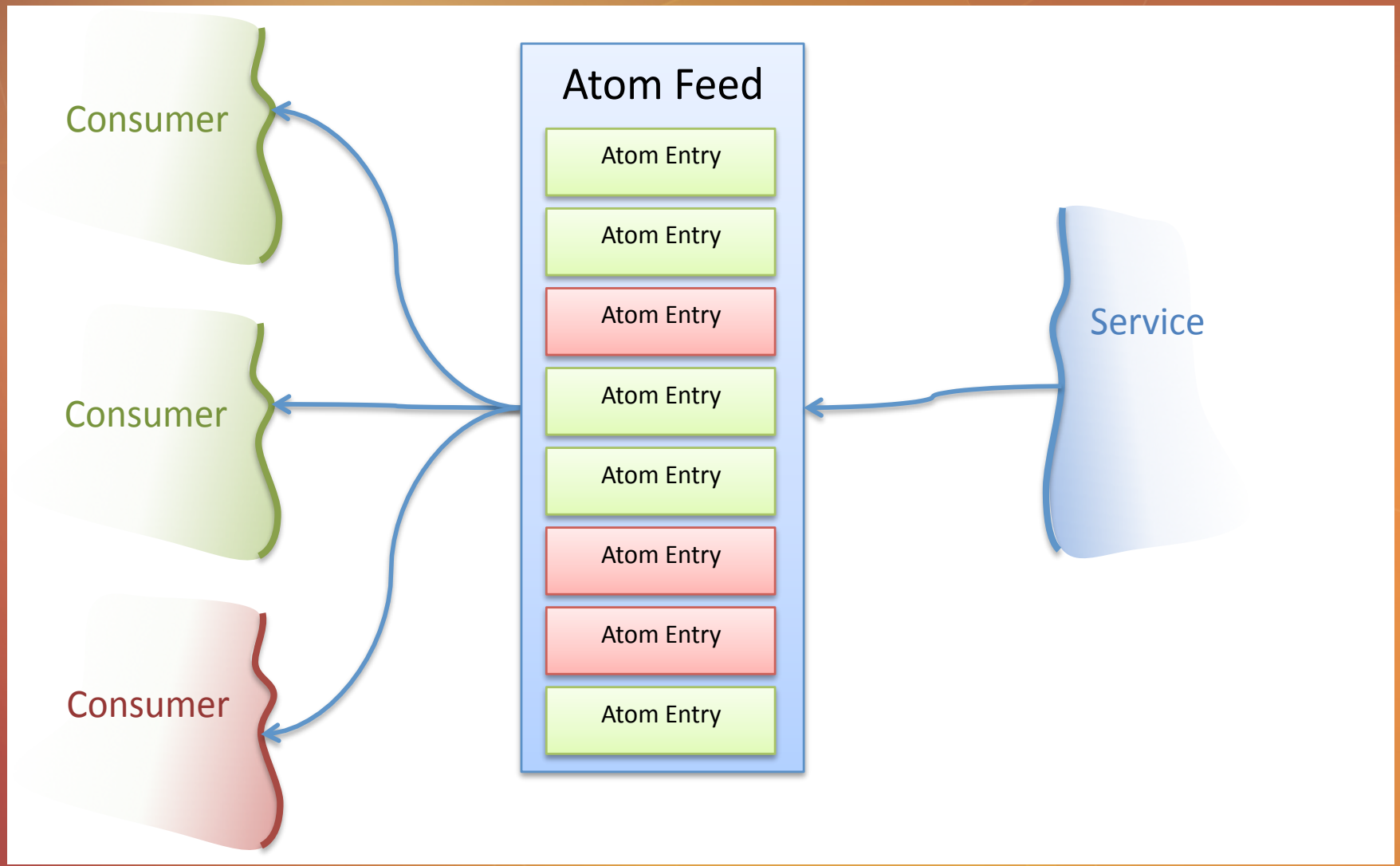
Publish secret data

- Here are my bank account details, please feel free to use them:

```
bb0ff34c3ab9c2a478cb7b8b61139a787bab5de1b4b
5ec463db070e1b72c502114758f1afd44c09b799207
3ccf00b43dc991579ddc5cbb91ea6984cbda08be9f
```

- Useless to you, or anyone else, unless you know the decryption key

Widely publish secret data



Secure messaging with Atom

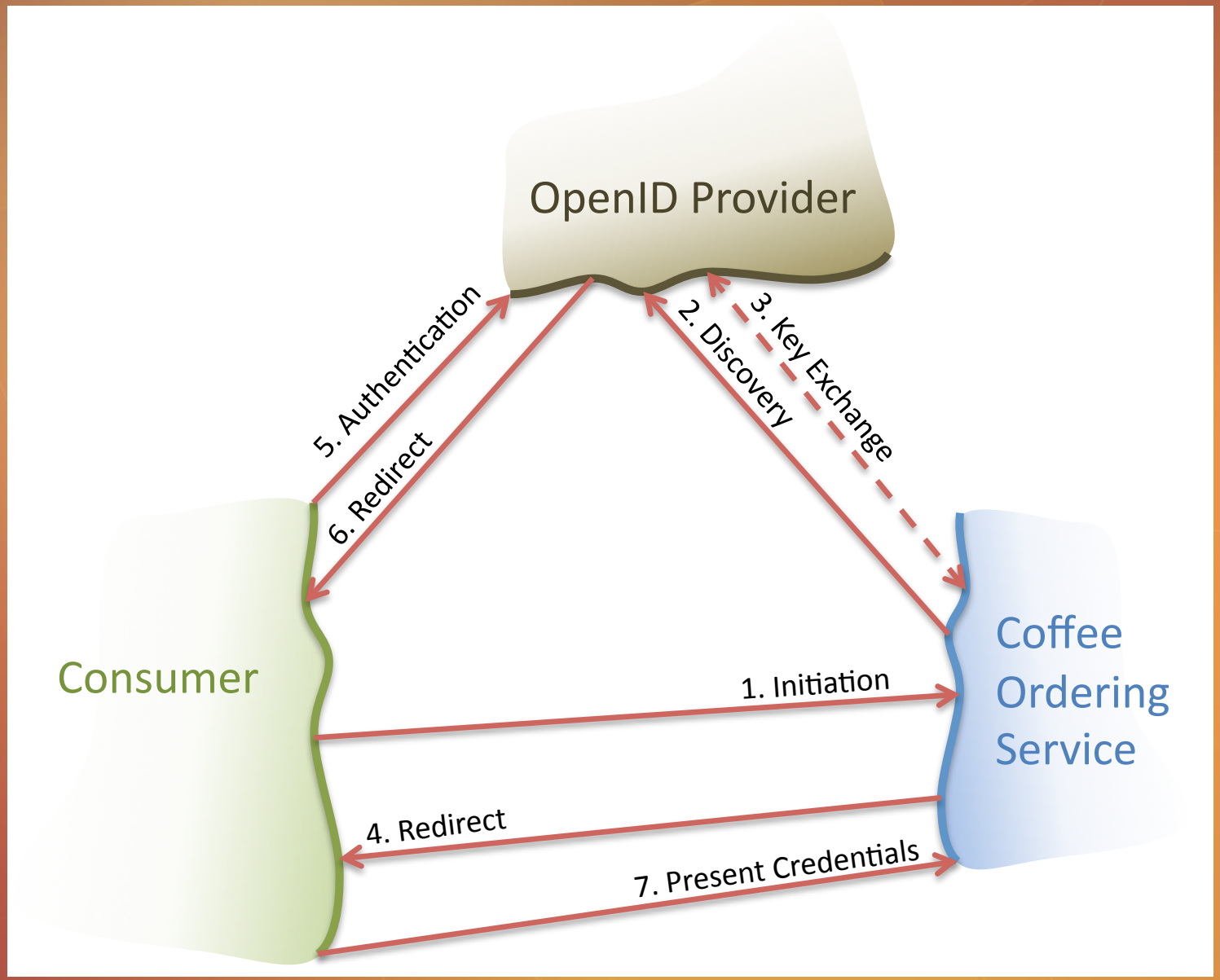
- The contents of individual atom:entry elements can be encrypted with public/shared keys for specific consumers
- Only consumers who know the corresponding private/ shared key can make sense of the content
  - To anyone else, it's gibberish
- Keep the crypto strong!
  - This will be in the public domain, beware brute force on weak algorithms
- Can cache this widely, reduced performance hit
- But beware coupling via keys!

OpenID

- OpenID is a decentralised framework for digital identities
  - Not trust, just identity!
- Your have an OpenID provider or one is provided for you
  - It has a URI
- Services that you interact with will ask for that URI
- Your OpenID provider will either:
  - Accept the request for processing immediately
  - Ask whether you trust the requesting site (e.g. via email with hyperlinks)
- Once your OpenID server OK's the login, then you are authenticated against the remote service
  - With your canonical credentials

Authenticating doesn't mean you're authorised to do anything!
This is not a trust system!

OpenID Workflow



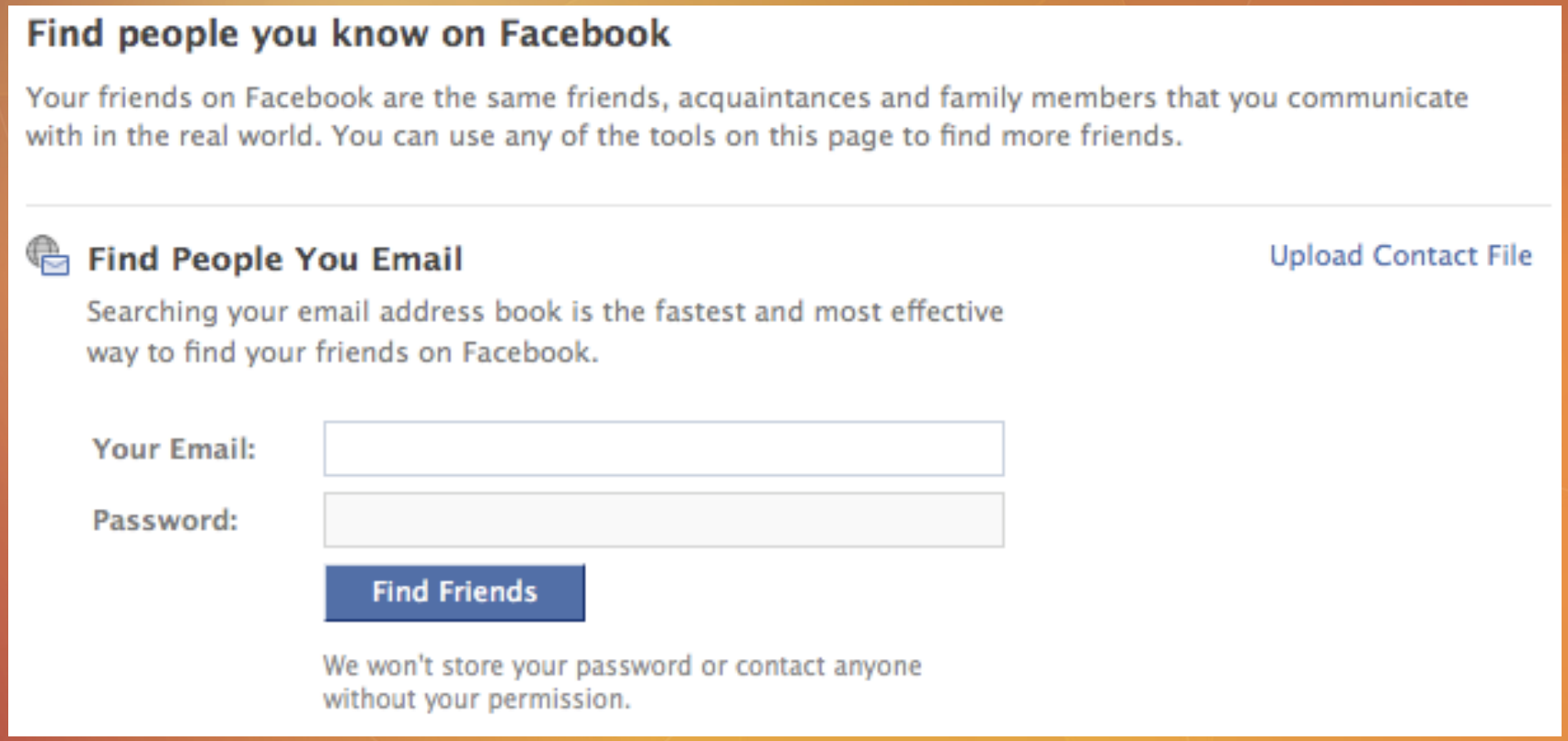
Not-So-OpenID?

- There's no trust between OpenID providers
- Your Web service might not accept my OpenID provider
  - In general it won't!
- Trusted providers centralise control
  - Against the philosophy of decentralised ID!
- Federated providers won't interoperate
  - Need a hybrid "signing" model like CAs?

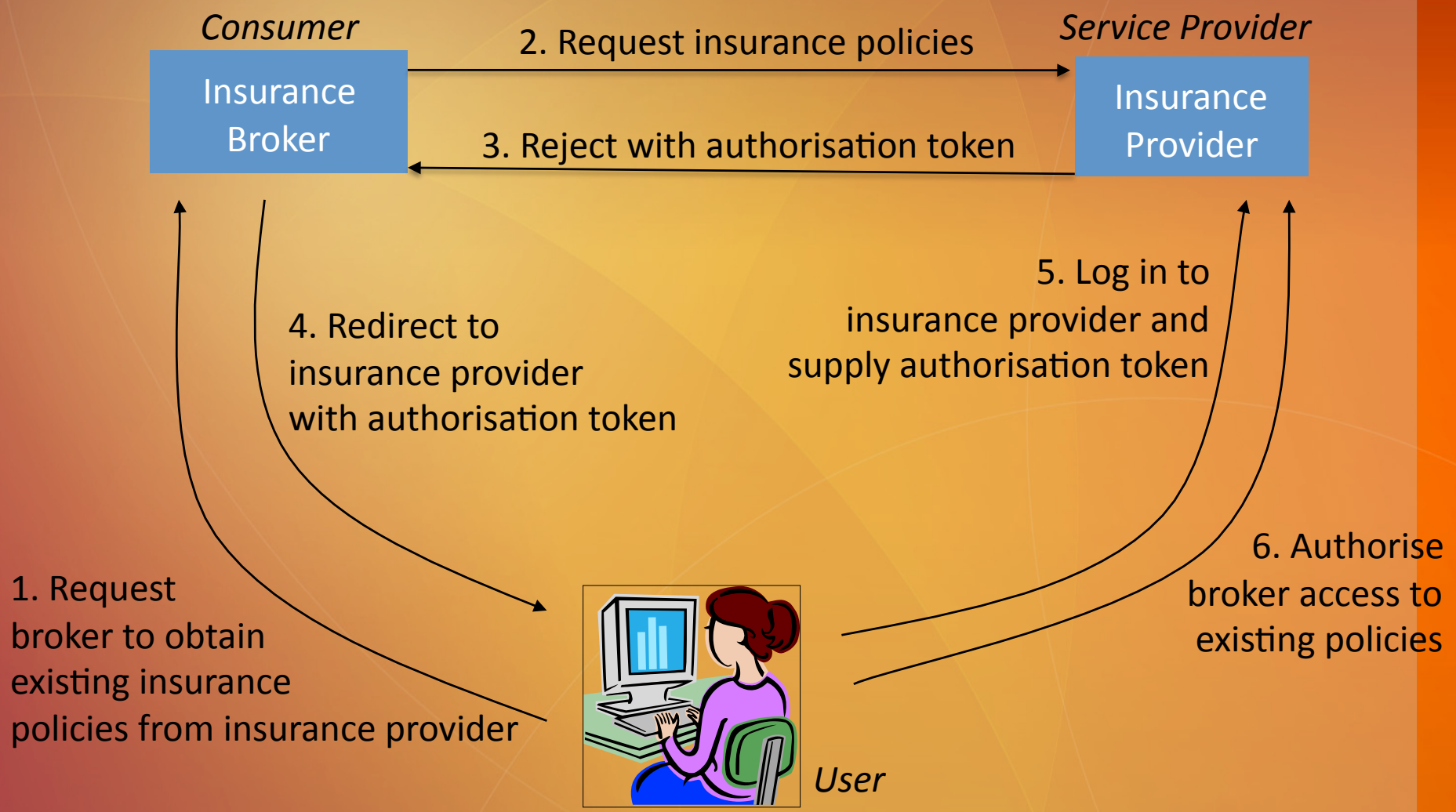
OAuth

- Web-focused access delegation protocol
- Give other Web-based services access to some of your protected data without disclosing your credentials
- Simple protocol based on HTTP redirection, cryptographic hashes and digital signatures
- Extends HTTP Authentication as the spec allows
  - Makes use of the same headers and status codes
  - These are understood by browsers and programmatic clients
- Not dependent on OpenID, but can be used together

Why OAuth?


Find people you know on Facebook
Your friends on Facebook are the same friends, acquaintances and family members that you communicate with in the real world. You can use any of the tools on this page to find more friends.
Find People You Email
Upload Contact File
Searching your email address book is the fastest and most effective way to find your friends on Facebook.
Your Email:
Password:
Find Friends
We won't store your password or contact anyone without your permission.

OAuth Workflow



Tech Interlude

Service Hacks and Defences

Denial of Service

- Large incoming representations can cause problems
  - DoS through memory consumption on the server
- Use the `Content-Length` header strictly
  - No header, bin payload
    - 400 Bad Request
  - Stop processing payload after the number of bytes in the header
  - Bin payloads for suspiciously large headers
    - Who wants a million cappuccinos?
- Swallow `OutOfMemoryError` or `OutOfMemoryException`
  - Don't let them know they won!

```
POST /order HTTP/1.1
Host: restbucks.com
Content-Type:application/vnd.restbucks
    +xml

<order xmlns="http://
    schemas.restbucks.com/order">
  <location>takeAway</location>
  <item>
    <name>latte</name>
    <quantity>1</quantity>
    <milk>whole</milk>
    <size>small</size>
  </item>
<!-- Millions more item elements -->
  <item>
    <name>latte</name>
    <quantity>1</quantity>
    <milk>skim</milk>
    <size>small</size>
  </item>
</order>
```

Keep Secrets, Secret

- You can be a good guy, and help attackers
  - E.g. 401
  - Says there's something interesting here!
- Or you can be less helpful
  - 404
  - Says there's nothing to see here, even if there really is
- Don't use easily guessable URIs – they can be hijacked
  - UUID is your friend
- Think carefully about what attackers can learn from probing you
  - You don't always want to be a good guy!

Act Defensively

- Validate the content of representations
  - Just over 2^{16} café lattes would be quite lucrative
  - But is likely a ruse to get a large negative number into our workflow
    - Integer overflow?
- Don't forget anti-corruption layering between your resources and you domain model
  - REST is not mindlessly exposing a domain over HTTP!

```
<order xmlns="http://
   schemas.restbucks.com/
   order">
   <location>takeAway</
   location>
   <item>
      <name>latte</name>
      <quantity>
        2147483648
      </quantity>
      <milk>whole</milk>
      <size>small</size>
   </item>
   ...
</order>
```

Don’t be gamed

```
GET /order/../../../../etc/passwd HTTP/1.1
Host: restbucks.com
```

- Oh oh
- We just gave up the password file
- And rainbow tables cracked it in no time

```
GET /order/../../../../dev/random HTTP/1.1
Host: restbucks.com
```

- Oh oh
- We just generated a never-ending stream of bytes
- And now we’re going to spend all our time serving them

Frameworks help avoid these problems

Less is best

- Bugs hide in code
- Bugs cause security breaches
- Less code, less places for bugs to hide
- So build only what you *need*
  - And keep your software *soft* so you can grow it over time

Defend in Depth

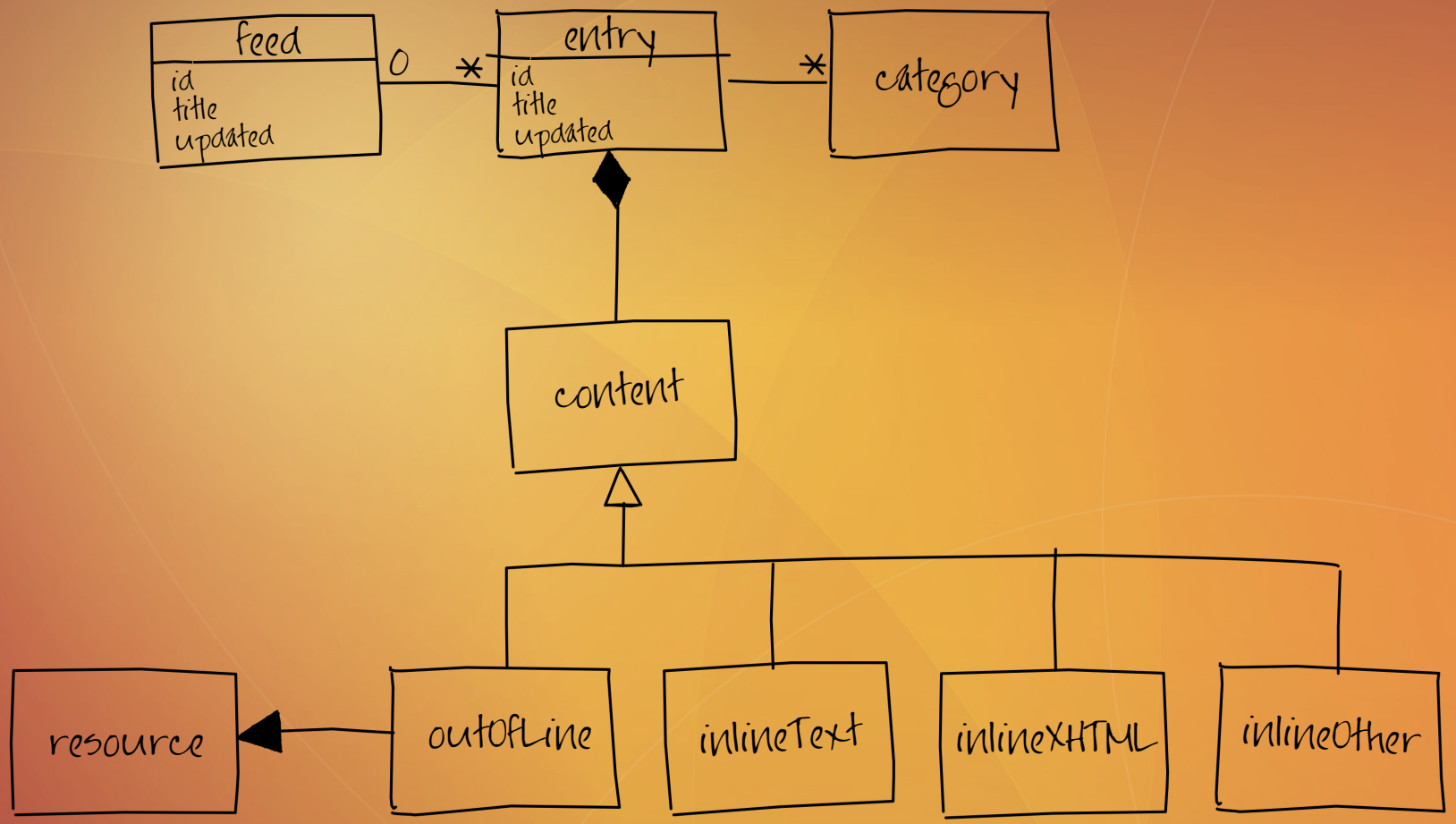
- Use firewalls
  - On the network, and on the server
- Do open ports 80 and 443
  - Do not open other ports
- Do not mistake HTTPS for security
  - It's not enough!
- Run at least privilege
  - Never run your service as *root* or *administrator*
- Keep good deployment hygiene
  - No lingering artifacts that attackers can grab hold of
    - Deploy only the config files, DBs, etc that you need
    - And remove what you don't
- And remember that social engineering is still effective!

Atom and AtomPub

Atom

```
<feed xmlns="http://www.w3.org/2005/Atom">
  <title type="text">Inventory</title>
  <id>urn:uuid:426830d2-ab1d-11dd-a9c5-c85155d89593</id>
  <updated>2008-09-10T14:50:00Z</updated>
  <author>
    <name>Leicester Square</name>
    <uri>http://restbucks.com/stores/1234</uri>
  </author>
  <link rel="self" href="http://restbucks.com/stores/1234/inventory"/>
  <entry>
    <id>urn:uuid:95506d98-aae9-4d34-a8f4-1ff30bece80c</id>
    <title type="text">Chocolate Chip Cookies</title>
    <updated>2008-09-10T14:45:32Z</updated>
    <content type="application/vnd.restbucks+xml">
      <inventory xmlns="http://schemas.restbucks.com/inventory">
        <product xmlns:a="http://www.w3.org/2005/Atom">
          <a:link href="http://restbucks.com/product/9876" type="application/vnd.restbucks+xml"/>
        </product>
        <quantity>678</quantity>
       </inventory>
    </content>
  </entry>
  <entry>
    <id>urn:uuid:7fb82319-b190-46d2-bb88-c9fcce240643</id>
    <title type="text">Fairtrade Coffee</title>
    <updated>2008-09-10T13:55:02Z</updated>
    <content type="application/vnd.restbucks+xml">
      <inventory xmlns="http://schemas.restbucks.com/inventory">
        <product xmlns:a="http://www.w3.org/2005/Atom">
          <a:link href="http://restbucks.com/product/211" type="application/vnd.restbucks+xml"/>
        </product>
        <quantity>407</quantity>
       </inventory>
    </content>
  </entry>
</feed>
```

Atom model



Based on diagrams by Stefan Tilkov

Entries can be standalone

```
<entry xmlns="http://www.w3.org/2005/Atom">
  <id>urn:uuid:95506d98-aae9-4d34-a8f4-1ff30bece80c</id>
  <title type="text">Chocolate Chip Cookies</title>
  <updated>2008-09-10T14:45:32Z</updated>
  <content type="application/vnd.restbucks+xml">
    <inventory xmlns="http://schemas.restbucks.com/inventory">
      <product xmlns:a="http://www.w3.org/2005/Atom">
        <a:link href="http://restbucks.com/product/9876" type="application/vnd.restbucks+xml"/>
      </product>
      <quantity>678</quantity>
    </inventory>
  </content>
</entry>
```

AtomPub

Application protocol

for

editing Web resources

using

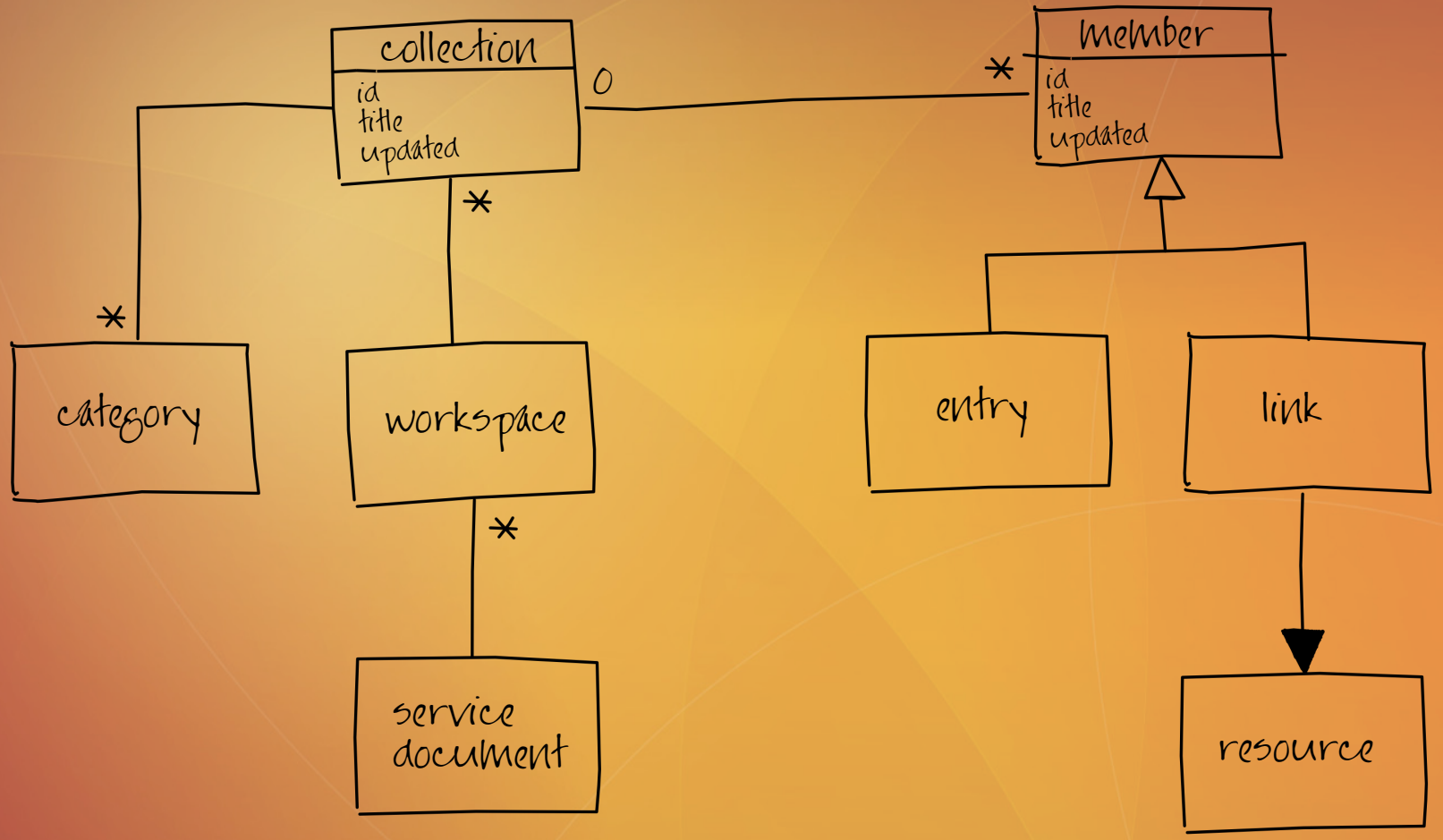
HTTP transfer of Atom-formatted representations

Methods, status codes, headers

Not just Atom resources

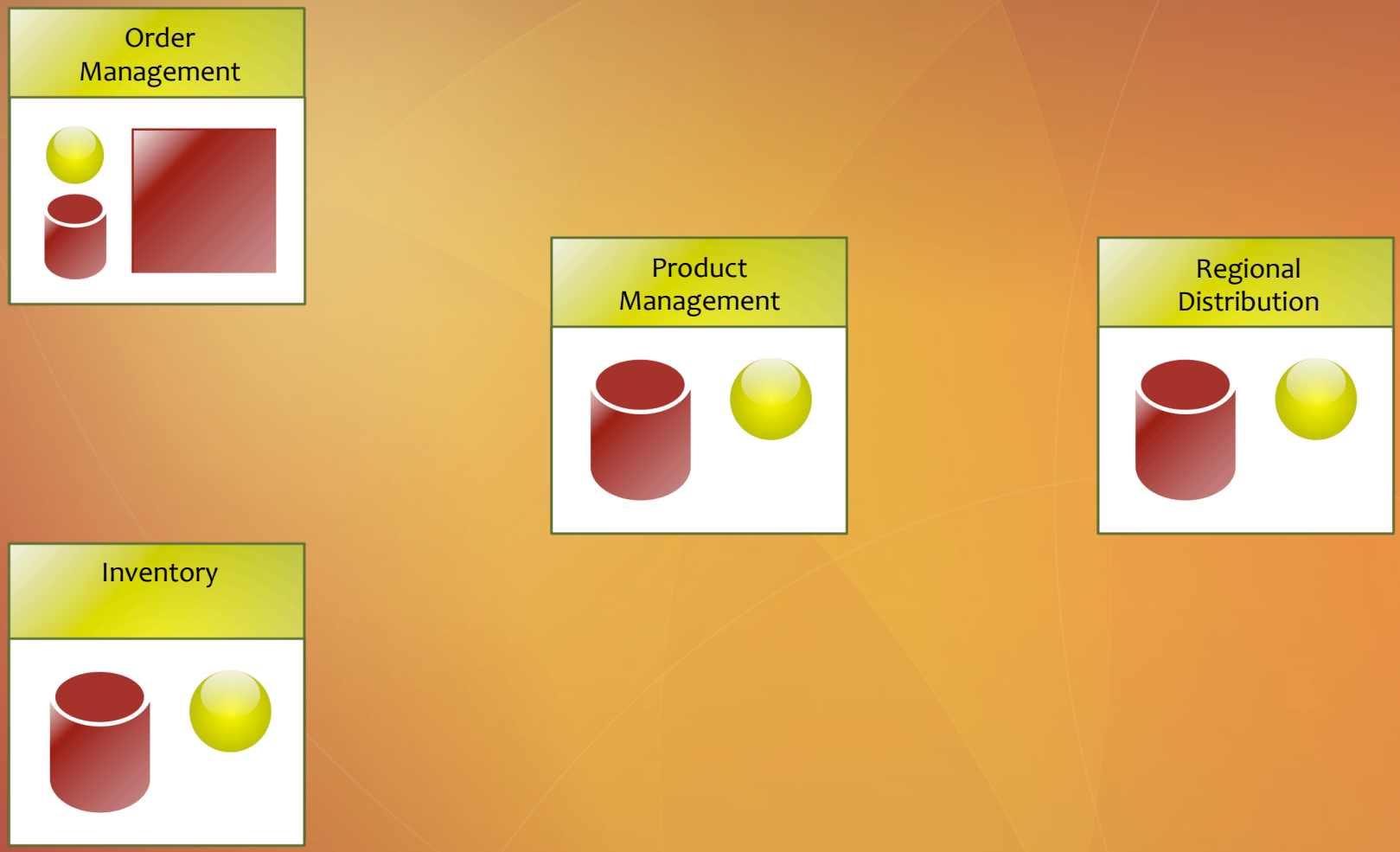
Server envelope

AtomPub model

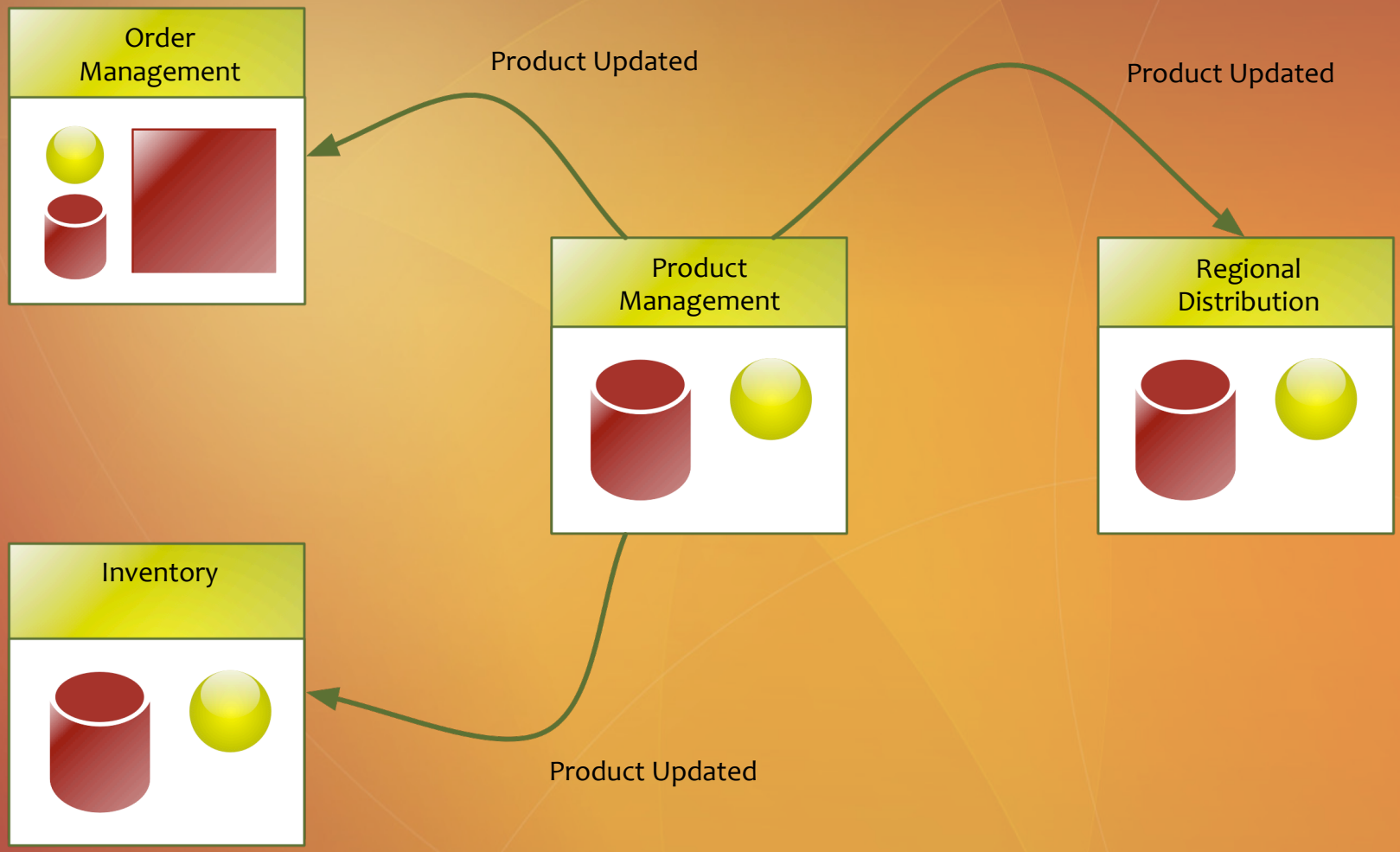


Based on diagrams by Stefan Tilkov

Restbucks


Order Management
Product Management
Regional Distribution
Inventory

Who needs to know what?



Implementation options

Point-to-point

- Publisher maintains subscriber list
- Queues to reduce temporal coupling

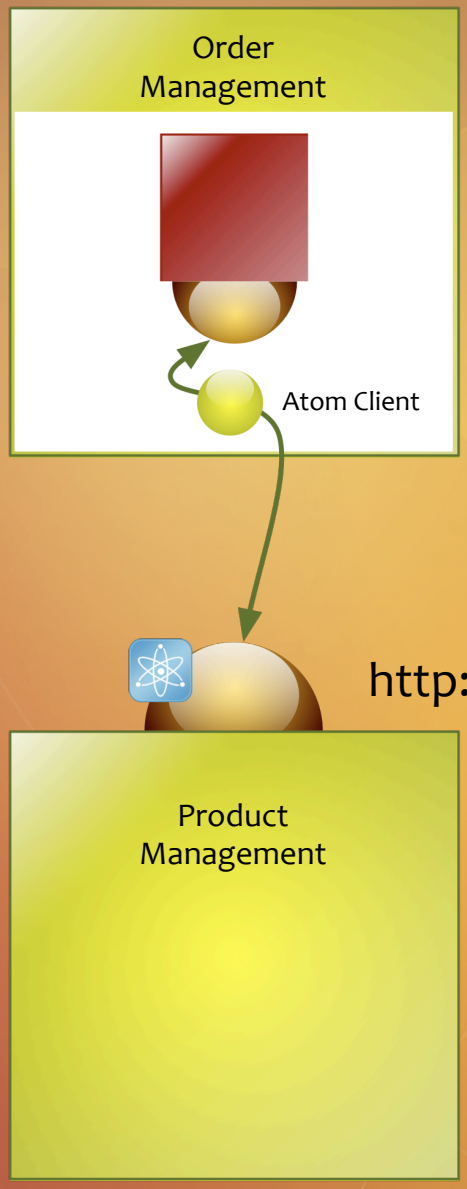
Bus

- Subscriptions and guaranteed delivery delegated to middleware
- Reduced location and temporal coupling

Consumers pull events

- Consumers poll publishers
- Guaranteed delivery delegated to consumers
- No list of subscribers to maintain

Polling an Atom feed



http://restbucks.com/products/notifications

Atom icons from http://sniker.blogage.de/

Atom feed represents an event stream

```
<feed xmlns="http://www.w3.org/2005/Atom">
  <id>urn:uuid:be21b6b0-57b4-4029-ada4-09585ee74adc</id>
  <title type="text">Product Notifications</title>
  <updated>2008-09-10T14:50:32+01:00</updated>
  <author>
    <name>Product Management</name>
    <uri>http://restbucks.com/products</uri>
  </author>
  <link rel="self" href="http://restbucks.com/products/notifications/2008/9/10/13"/>
  <link rel="prev-archive" href="http://restbucks.com/products/notifications/2008/9/10/12"/>
  <entry>
    <id>urn:uuid:95506d98-aae9-4d34-a8f4-1ff30bece80c</id>
    <title type="text">product created</title>
    <updated>2008-09-10T14:45:32+01:00</updated>
    <link rel="self"
       href="http://restbucks.com/products/notifications/95506d98-aae9-4d34-a8f4-1ff30bece80c"/>
    <category term="product"/>
    <category term="created"/>
    <content type="application/xml">
      <product xmlns="http://restbucks.com/products" xmlns:atom="http://www.w3.org/2005/Atom">
        <atom:link type="application/xml" rel="alternate" etag="1"
            href="http://restbucks.com/products/527"/>
        <id>527</id>
        <name>Fairtrade Roma Coffee Beans</name>
        <size>1kg</size>
        <price>10</price>
      </product>
    </content>
  </entry>
  ...
</feed>
```

On the wire

Request

```
GET /products/notifications HTTP/1.1
Host: restbucks.com
```

Response

```
HTTP/1.1 200 OK
Cache-Control: max-age=60
Content-Length: 12230
Content-Type: application/atom+xml;charset="utf-8"
Content-Location: http://restbucks.com/products/notifications/2008/9/10/13
Last-Modified: Wed, 10 Sep 2008 13:50:32 GMT
ETag: "6a0806ca"
Date: Wed, 10 Sep 2008 13:51:03 GMT

<feed xmlns="http://www.w3.org/2005/Atom"><id>urn:uuid:be21b6b0-57b4-4029-
ada4-09585ee74adc</id><title type="text">Product Notifications</
title><updated>2008-09-10T14:50:32+01:00</updated><author><name>Product Management</
name><uri>http://restbucks.com/products</uri></author><link rel="self" href="http://
restbucks.com/products/notifications/2008/9/10/13"/><link rel="prev-archive"
href="http://restbucks.com/products/notifications/2008/9/10/12"/><entry><id>urn:uuid:
95506d98-aae9-4d34-a8f4-1ff30bece80c</id><title type="text">product created</
title><updated>2008-09-10T14:45:32+01:00</updated><link rel="self" href="http://
restbucks.com/products/notifications/95506d98-aae9-4d34-a8f4-1ff30bece80c"/><category
term="product"/><category term="created"/><content type="application/xml"><product
xmlns="http://restbucks.com/products"
...
```

Retrieving the archive by following links

Request

```
GET /products/notifications/2008/9/10/12 HTTP/1.1
Host: restbucks.com
```

Response

```
HTTP/1.1 200 OK
Cache-Control: max-age=2592000
Content-Length: 9877
Content-Type: application/atom+xml;charset="utf-8"
Last-Modified: Wed, 10 Sep 2008 12:57:14 GMT
Date: Wed, 10 Sep 2008 13:51:46 GMT

<feed xmlns="http://www.w3.org/2005/Atom"><id>urn:uuid:4cbc0acf-a211-40ce-a50e-
a75d299571da</id><title type="text">Product Notifications</
title><updated>2008-09-10T13:57:14+01:00</updated><author><name>Product Management</
name><uri>http://restbucks.com/products</uri></author><link rel="self" href="http://
restbucks.com/products/notifications/2008/9/10/12"/><link rel="current" href="http://
restbucks.com/products/notifications/2008/9/10/13"/><link rel="prev-archive"
href="http://restbucks.com/products/notifications/2008/9/10/11"/
><entry><id>urn:uuid:b436fda6-93f5-4c00-98a3-06b62c3d31b8</id><title
type="text">promotion cancelled</title><updated>2008-09-10T13:57:14+01:00</
updated><link rel="self" href="http://restbucks.com/products/notifications/
b436fda6-93f5-4c00-98a3-06b62c3d31b8"/><category term="promotion"/><category
term="cancelled"/><content type="application/xml"><promotion xmlns="http://
restbucks.com/products" xmlns:atom="http://www.w3.org/2005/Atom"><atom:link
type="application/xml"
...
```

Archive

```
<feed xmlns="http://www.w3.org/2005/Atom" xmlns:fh="http://purl.org/syndication/history/1.0">
  <id>urn:uuid:4cbc0acf-a211-40ce-a50e-a75d299571da</id>
  <title type="text">Product Notifications</title>
  <updated>2008-09-10T13:57:14+01:00</updated>
  <author>
    <name>Product Management</name>
    <uri>http://restbucks.com/products</uri>
  </author>
  <link rel="self" href="http://restbucks.com/products/notifications/2008/9/10/12"/>
  <link rel=”next-archive" href="http://restbucks.com/products/notifications/2008/9/10/13"/>
  <link rel="prev-archive" href="http://restbucks.com/products/notifications/2008/9/10/11"/>
  <fh:archive/>
  <entry>
    <id>urn:uuid:b436fda6-93f5-4c00-98a3-06b62c3d31b8</id>
    <title type="text">promotion cancelled</title>
    <updated>2008-09-10T13:57:14+01:00</updated>
    <link rel="self"
      href="http://restbucks.com/products/notifications/b436fda6-93f5-4c00-98a3-06b62c3d31b8"/>
    <category term="promotion"/>
    <category term="cancelled"/>
    <content type="application/xml">
      <promotion xmlns="http://restbucks.com/products" xmlns:atom="http://www.w3.org/2005/Atom">
        <atom:link type="application/xml" rel="alternate" etag="3"
          href="http://restbucks.com/products/391"/>
        <regions>
          <atom:link type="application/atom+xml;type=feed" etag="14"
            href="http://restbucks.com/regions/london"/>
        </regions>
        <products>
          <atom:link type="application/xml" etag="1" href="http://restbucks.com/products/7642"/>
        </products>
      </promotion>
    </content>
  </entry>
  ...
</feed>
```

An Atom entry represents an event

```
<entry xmlns="http://www.w3.org/2005/Atom">
  <id>urn:uuid:b436fda6-93f5-4c00-98a3-06b62c3d31b8</id>
  <title type="text">promotion cancelled</title>
  <updated>2008-09-10T13:57:14+01:00</updated>
  <link rel="self"
    href="http://restbucks.com/products/notifications/b436fda6-93f5-4c00-98a3-06b62c3d31b8"/>
  <category term="promotion"/>
  <category term="cancelled"/>
  <content type="application/xml">
    <promotion xmlns="http://restbucks.com/products" xmlns:atom="http://www.w3.org/2005/Atom">
      <atom:link type="application/xml" rel="alternate" etag="3"
        href="http://restbucks.com/products/391"/>
      <regions>
        <atom:link type="application/atom+xml;type=feed" etag="14"
          href="http://restbucks.com/regions/london"/>
      </regions>
      <products>
        <atom:link type="application/xml" etag="1" href="http://restbucks.com/products/7642"/>
      </products>
    </promotion>
  </content>
</entry>
```

Handling eager re-polling

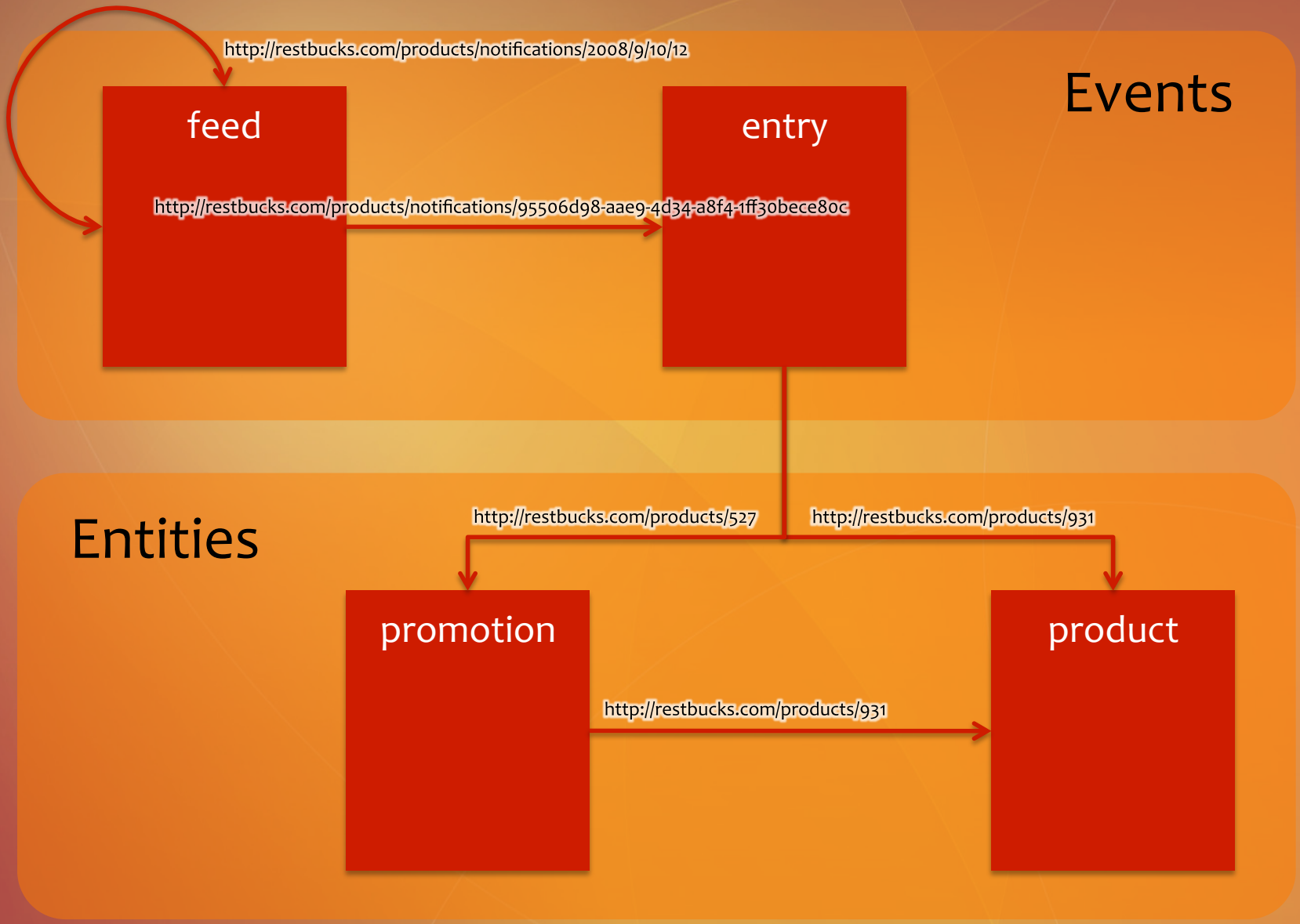
Request

```
GET /products/notifications HTTP/1.1
Host: restbucks.com
If-None-Match: "6a0806ca"
```

Response

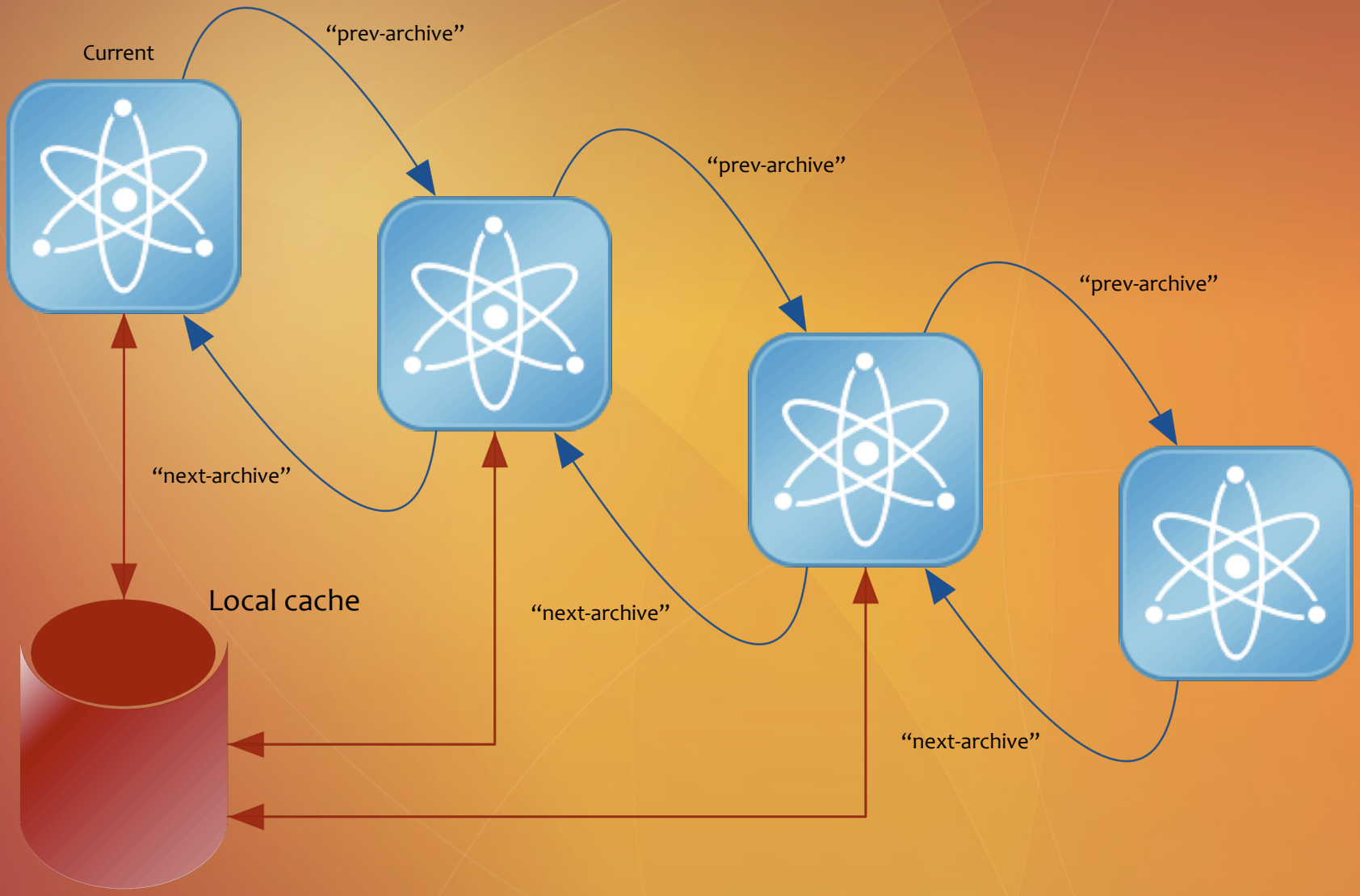
```
HTTP/1.1 304 Not Modified
Date: Wed, 10 Sep 2008 13:57:20 GMT
```

Divide and conquer

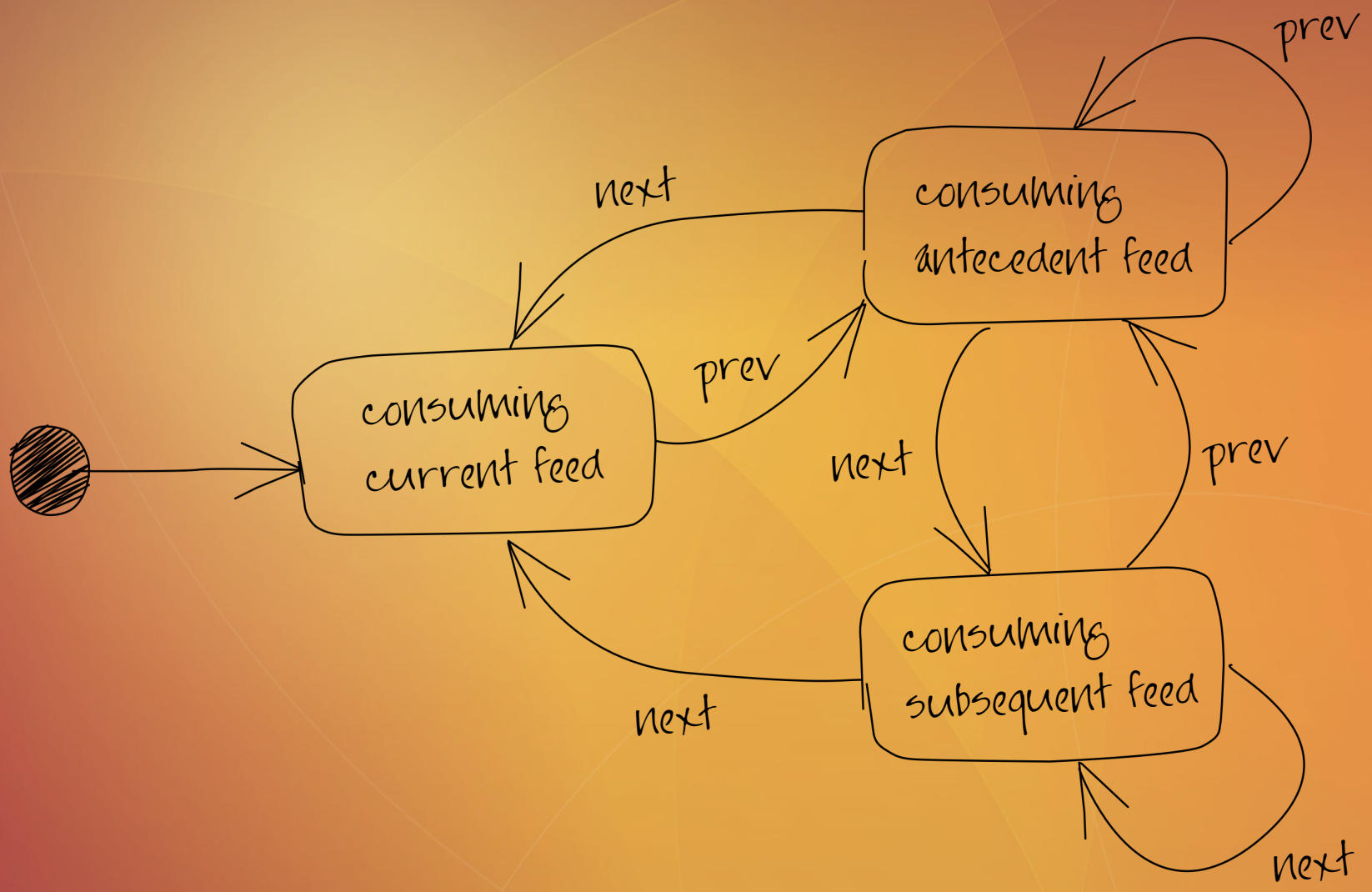


An Atom- and AtomPub-enabled service

Caching archived feeds



Domain application protocol



"application/atom+xml"

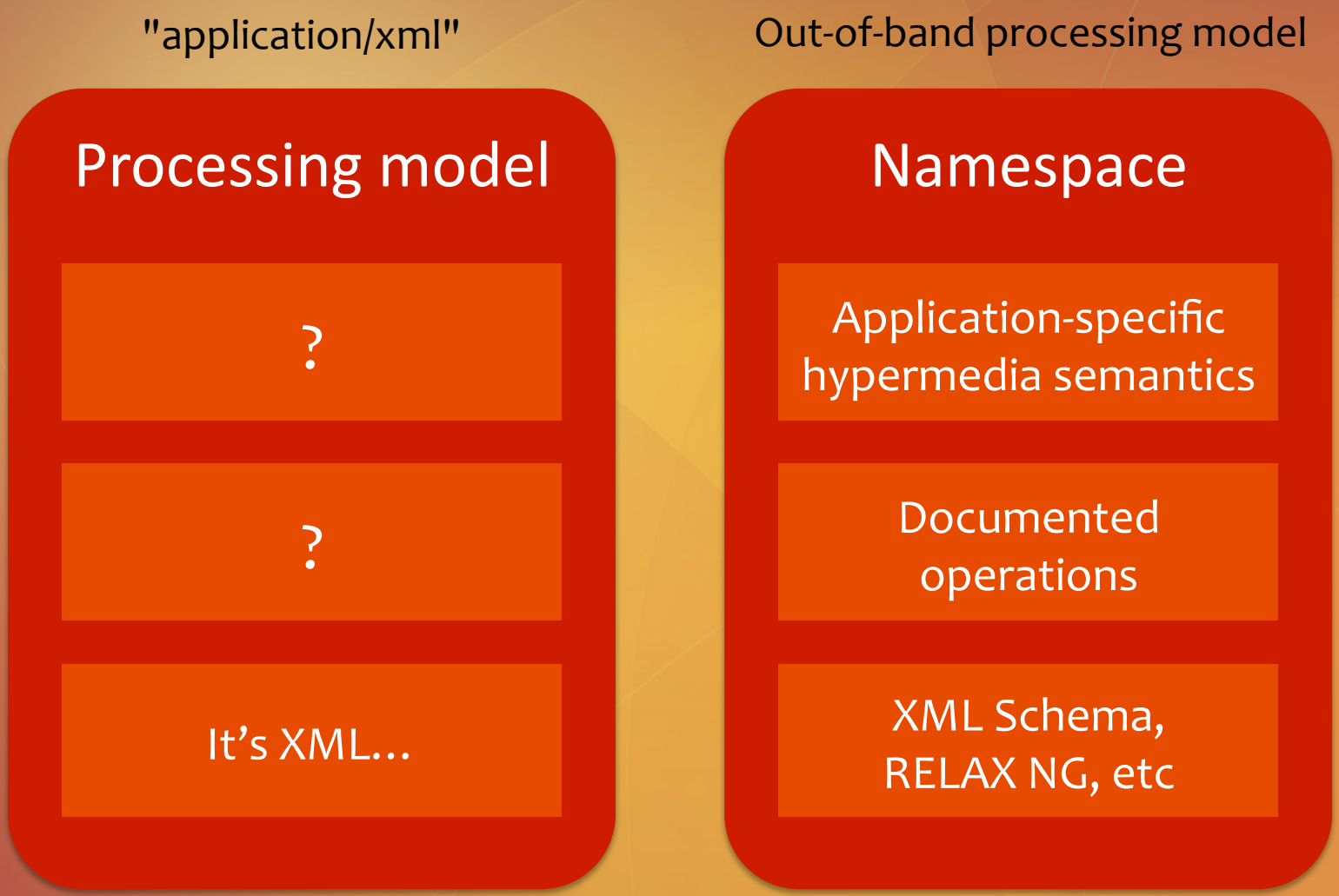
Atom Publishing Protocol

"An IRI of an editable Member Entry. When appearing within an atom:entry, the href IRI can be used to retrieve, update and delete the Resource represented by that Entry."

```
<link
    rel="edit"
    href="http://restbucks.com/products/notifications/2008/9/10/13"
    type="application/atom+xml;type=entry"/>
```

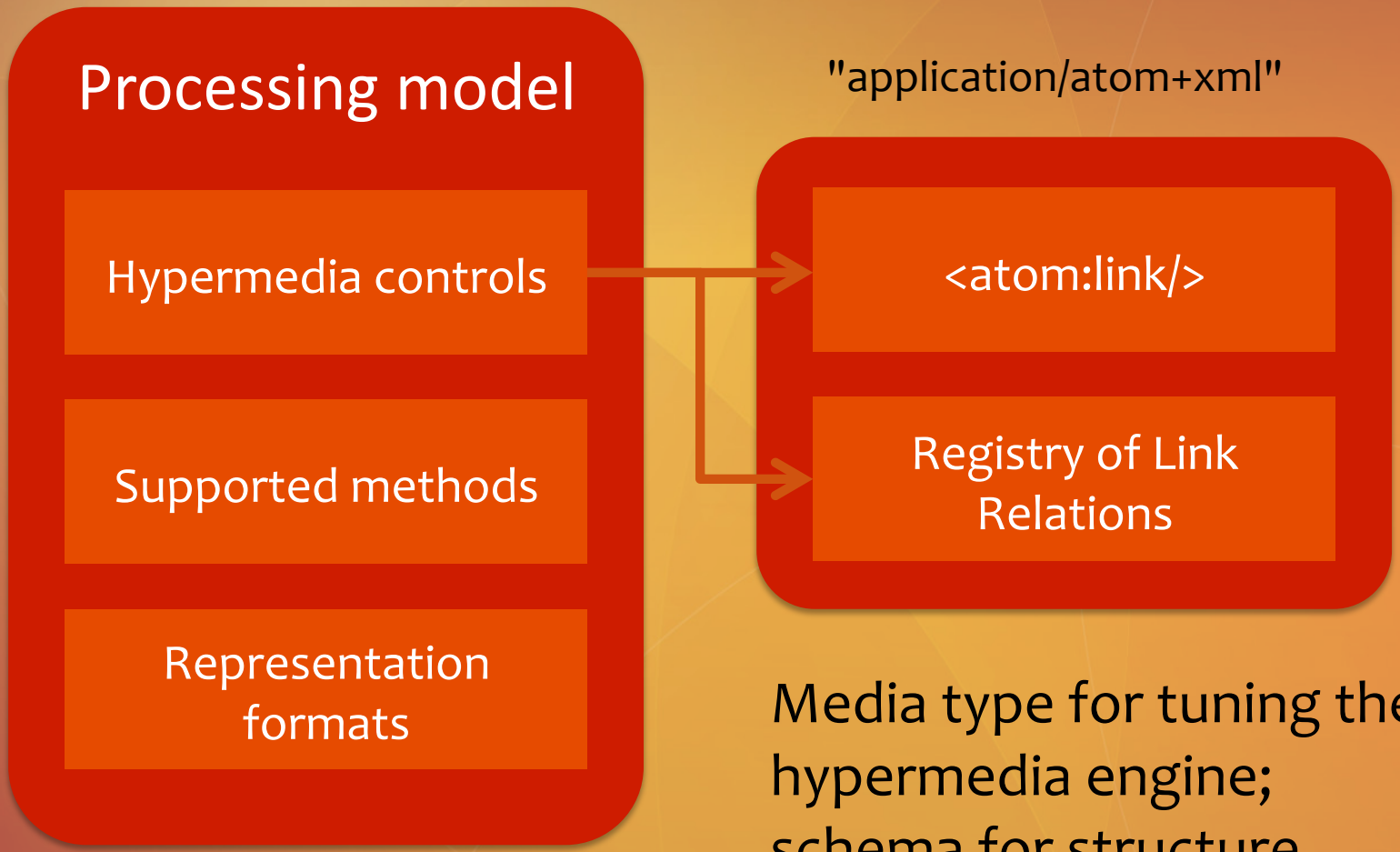
Atom Syndication Format

Remember: "application/xml" is not your friend



Custom media types

"application/vnd.restbucks+xml"

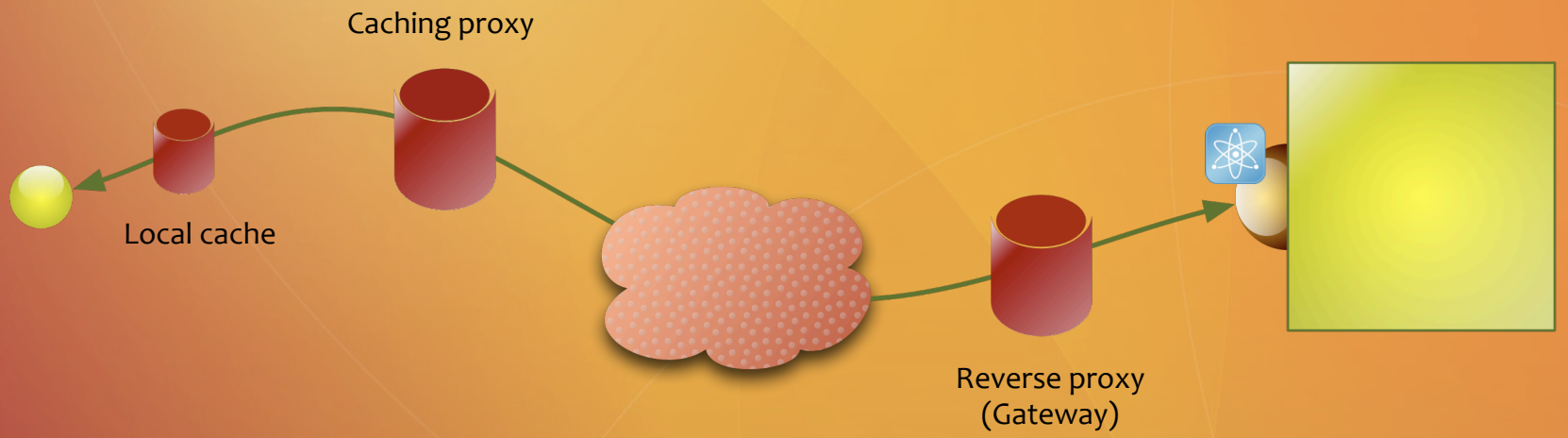


Media type for tuning the hypermedia engine; schema for structure

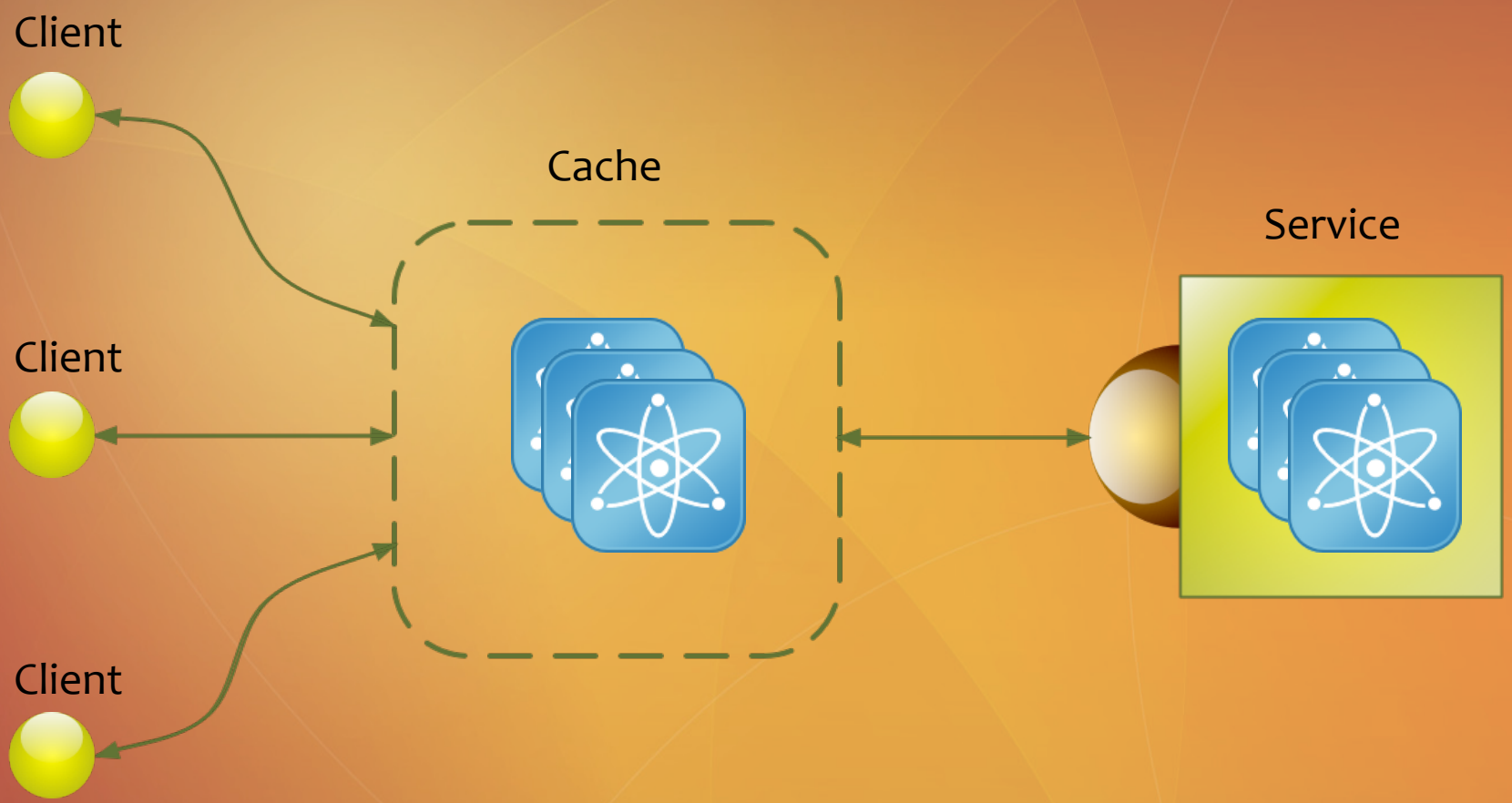
Caching

Uri	Description	Caching
`/products/notifications`	Current	Short
`/products/notifications/{year}/{month}/{day}/{hour}`	Archive	Long
`/products/notifications/{entry-id}`	Notification	Long
`/products/{product-id}`	Product	Varies
`/products/{hardware-id}`	Promotion	Varies

Caches



Caching the bus



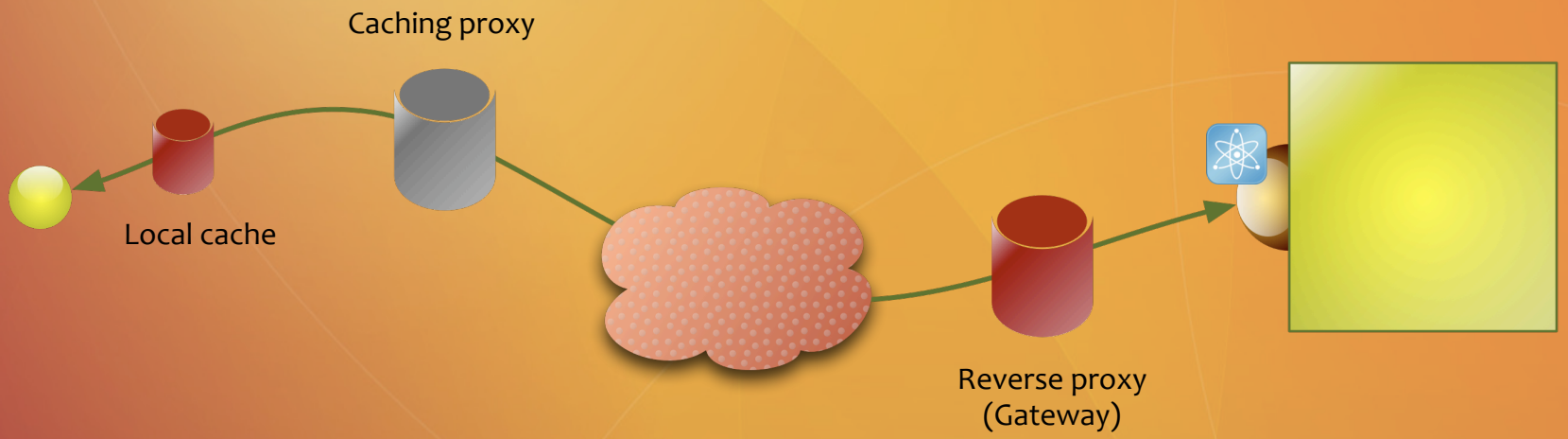
Protecting feeds

Request

```
GET /products/notifications HTTP/1.1
Host: restbucks.com
Authorization: Basic aWFuc3JvYmluc29uOlBhdHQzcm41==
```

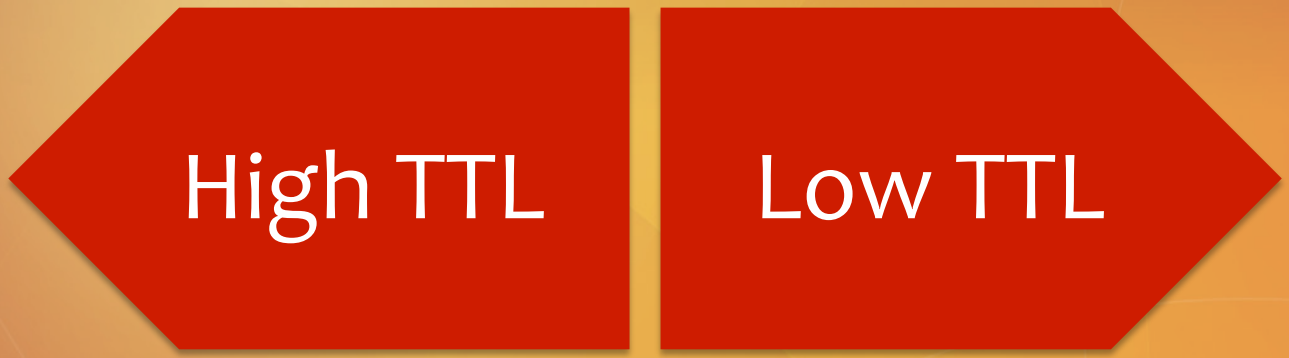
Response

```
...
Cache-control: public, no-cache
```



Caching dilemma

Efficient use of network resources



Publisher controls freshness of data

Cache channels

Mark Nottingham, Yahoo

- http://www.mnot.net/cache_channels/

Use Atom to extend the freshness of cached responses

Response

```
Cache-Control: max-age=60, channel="http://restbucks.com/products/channel/index",
channel-maxage
```

Response remains fresh as long as:

- Cache polls channel at least as often as "precision" specified by channel
- Channel doesn’t issue stale event

Epilogue

Web Architecture

- Ubiquitous, global on-ramp
- Connects everything to everything, based on URI-addressable resources
  - With a uniform interface
- Also provides standard coordination mechanism
  - Status codes!
- And is ambivalent about content
    - Media types!

URI Tunnelling

- Map URIs to methods and GET those URIs
  - Easy, ubiquitous
- Not very Web-friendly
  - Breaks expectations
  - Remember the library of congress incident?

POX

- Treats HTTP as a synchronous transport protocol
  - Great because it gets through firewalls
- But again breaks expectations
  - HTTP is not MOM!
- Misses out on all the good stuff from the Web
  - Status codes for coordination
  - Caching for performance
  - Loose coupling via hypermedia
  - Etc
- Not as good as proper message-oriented middleware
  - Which are low-latency, reliable, etc.

CRUD Services

- The simplest kind of Web-based service
- Embraces HTTP and Web infrastructure
  - Four verbs, status codes, formats
  - Cacheable!
- Can easily describe them
  - URI templates
  - WADL
- But tightly couples client and server
  - Might not be a problem in some domains

Hypermedia

- It's all about media types and link relations!
  - Describe state machines with lots of lovely links
- Constrain what you can do to resources with the uniform interface
- Loosely coupled
  - The server mints URIs to resources, clients follow them
  - Easily spans systems/domains (URIs are great!)
- Embraces the Web for robustness
  - Verbs, status codes, caching
- Design and implementation:
  - Design application protocol state machines;
  - Implement resource lifecycles;
  - Document using media types, link relation values and HTTP idioms.


Contracts


Reliability

Transactions

Scalability

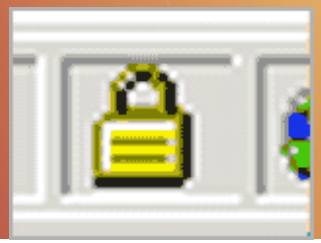
- Everything you know still applies
  - Stateless is good
  - Horizontal is good
- Yet everything you know no longer applies!
  - Text-based synchronous protocol is scalable???
- Do as little work as possible
  - Make interactions conditional
    - ETags and if-modified etc are your friends
- And cache!

Security

- HTTPS is still our friend!
  - But it inhibits caching
- OpenID support waning on the human Web
- OpenAuth now finding its feet
  - Likely to become dominant approach
- Other approaches like SAML, mature but yet to be widely deployed


Enterprise security is
awesome, but…

…you wouldn't use it at home!

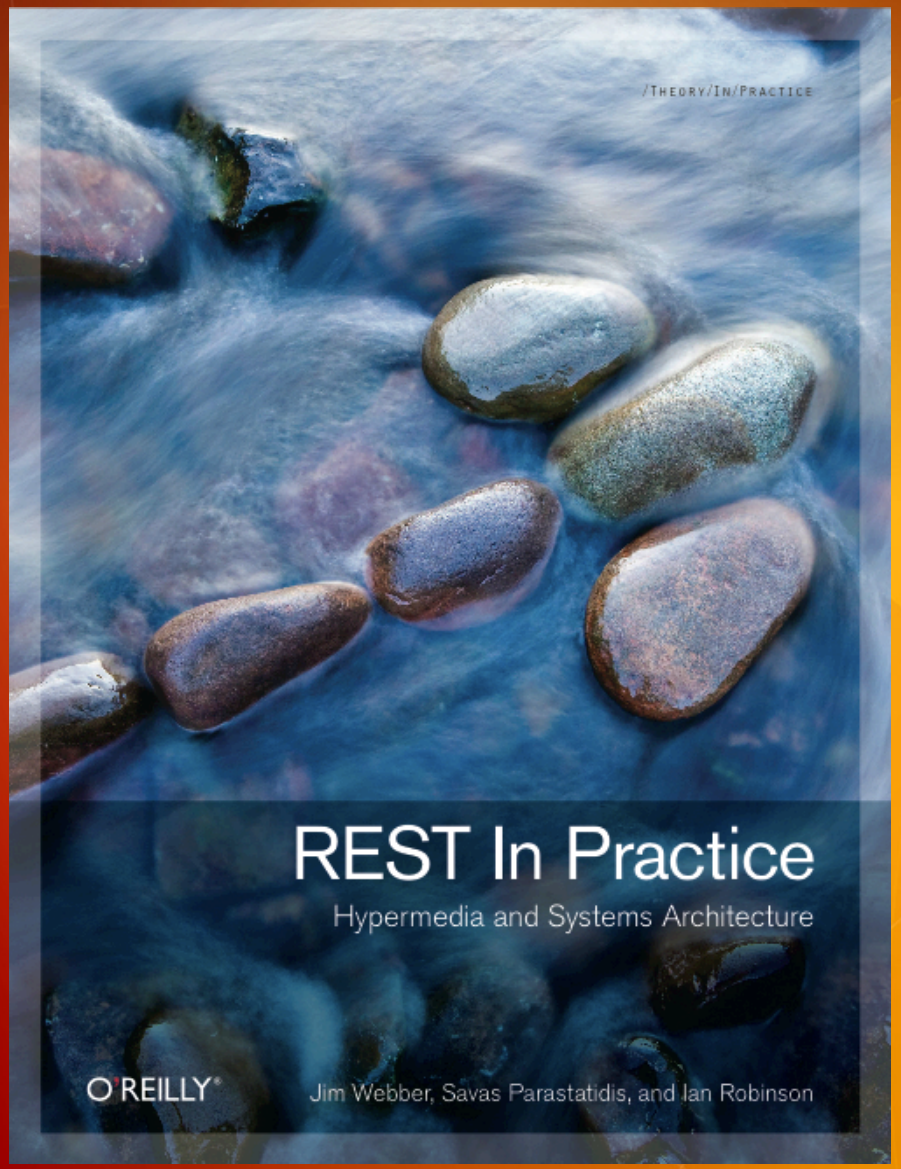


OpenID

OAUTH

Atom and AtomPub

- Atom is format that describes list of things
  - In terms of *feeds* and *entries*
- AtomPub is a protocol defined in terms Atom entries and links
- Together they can be used for very scalable pub/sub
  - But latency is very high compared to enterprise pub/sub
  - Caching enables massive scalability
    - But causes latency



Liked the Tutorial?
GET /theBook

204 No Content
(until early 2010!)