A Couple of Ways to Skin an Internet-Scale Cat

Jim Webber

http://jim.webber.name

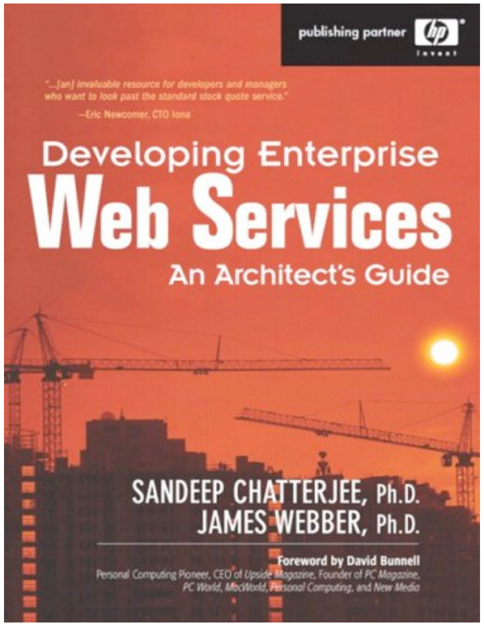
Roadmap

- A little Swedish
- Some home truths
  - About Web Services *and* the Web
- Implementing Workflows
  - The Starbuck's example
- Q&A

Jag heter Jim und kommer du England

- I like Web Services
  - I am a MESTian at heart
- I like the Web
  - I have sympathies that lie with the RESTafarians

- I wrote this book, about WS-*

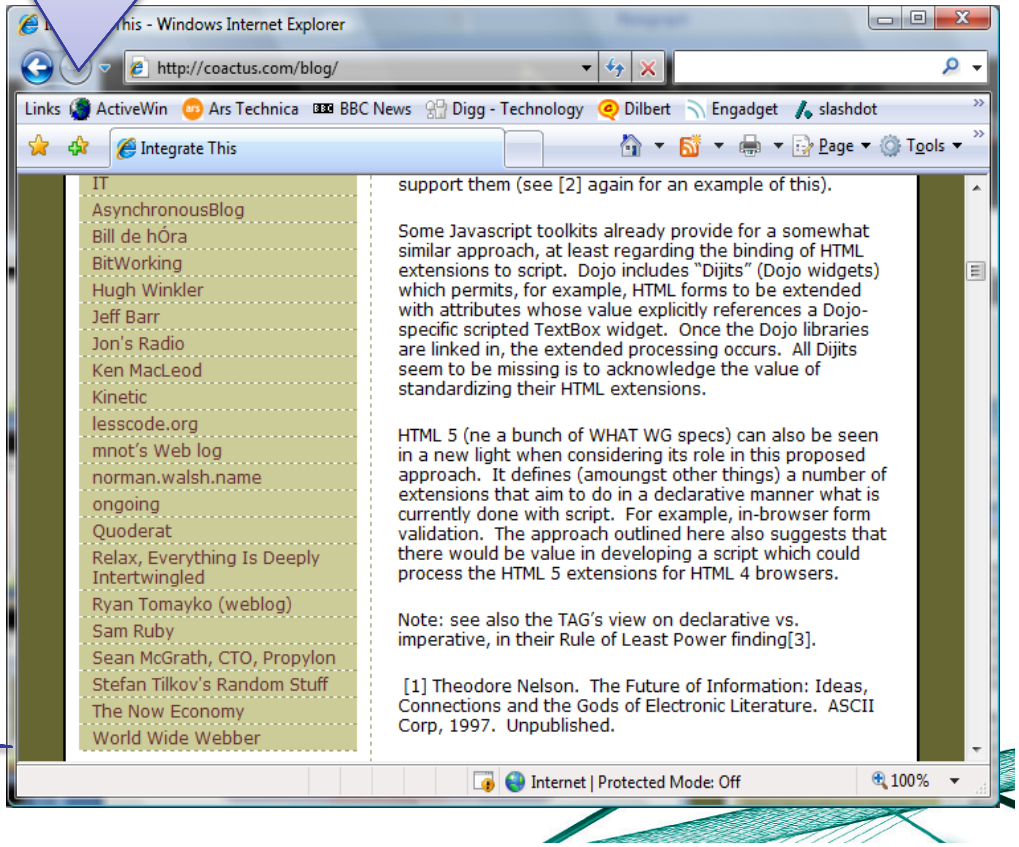

publishing partner
hp invent
"...[an] invaluable resource for developers and managers who want to look past the standard stock quote service."
–Eric Newcomer, CTO Iona
Developing Enterprise
Web Services
An Architect's Guide
SANDEEP CHATTERJEE, Ph.D.
JAMES WEBBER, Ph.D.
Foreword by David Bunnell
Personal Computing Pioneer, CEO of Upside Magazine, Founder of PC Magazine, PC World, MacWorld, Personal Computing, and New Media

Jag heter Jim und kommer du England

- I like Web Services
  - I am a MESTian at heart
- I like the Web
  - I have sympathies that lie with the RESTafarians

Mark Baker's consulting company, Coactus

- I am "similarly minded"



IT
AsynchronousBlog
Bill de hÓra
BitWorking
Hugh Winkler
Jeff Barr
Jon's Radio
Ken MacLeod
Kinetic
lesscode.org
mnot's Web log
norman.walsh.name
ongoing
Quoderat
Relax, Everything Is Deeply Intertwingled
Ryan Tomayko (weblog)
Sam Ruby
Sean McGrath, CTO, Propylon
Stefan Tilkov's Random Stuff
The Now Economy
World Wide Webber

support them (see [2] again for an example of this).

Some Javascript toolkits already provide for a somewhat similar approach, at least regarding the binding of HTML extensions to script. Dojo includes "Dijits" (Dojo widgets) which permits, for example, HTML forms to be extended with attributes whose value explicitly references a Dojo-specific scripted TextBox widget. Once the Dojo libraries are linked in, the extended processing occurs. All Dijits seem to be missing is to acknowledge the value of standardizing their HTML extensions.

HTML 5 (ne a bunch of WHAT WG specs) can also be seen in a new light when considering its role in this proposed approach. It defines (amoungst other things) a number of extensions that aim to do in a declarative manner what is currently done with script. For example, in-browser form validation. The approach outlined here also suggests that there would be value in developing a script which could process the HTML 5 extensions for HTML 4 browsers.

Note: see also the TAG's view on declarative vs. imperative, in their Rule of Least Power finding[3].

[1] Theodore Nelson. The Future of Information: Ideas, Connections and the Gods of Electronic Literature. ASCII Corp, 1997. Unpublished.

That's me

Falling out of Love?

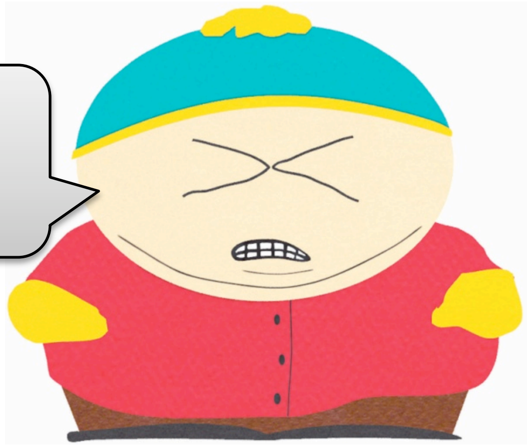


Photo: Comedy Central

- Two things:
  - WSDL
    - It’s an XML IDL for RPC
    - Therefore ill-suited for Internet scale
  - All the superfluous WS-* standards and politics
    - Too many dumb WS-KitchenSink standards
      - Not everything needs to be an OASIS standard!
    - Too many useful tools spent too long in standards wars
      - 3 transactions specs? Anyone heard of consistency???
- Toolkits hide messaging model, provide leaky abstractions over a distributed system

Why Web Services Rock could My World

- Good Web Services/SOA are message-oriented
  - TCP/IP is message-y and has scaled really well!
  - SOAP Service Description Language (SSDL) provides message-oriented metadata for services
    - WSDL must die, die, die!
- Business processes tend to be message-oriented
  - Easy to map workflows onto
- Loose coupling by default
- End-to-end processing model
  - Defined by SOAP, not WSDL!
- Composable model
  - You can ignore all the dumb stuff in the WS-* stack
    - Except WSDL because the toolkits embrace it ☹

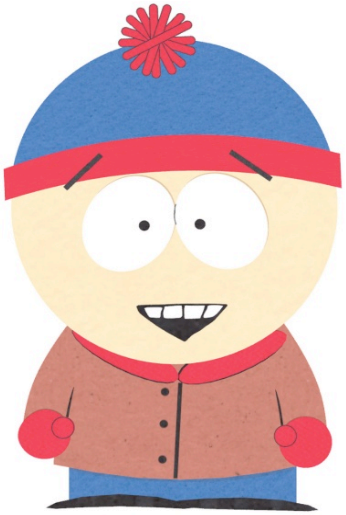


Photo: Comedy Central

Web Abuse

- Two lo-fi approaches to "Web" integration
  - URI tunnelling
  - POX
- Both models treat HTTP as a transport
  - More or less
- Yet some of the Web jihadists don't see this
- Both of these approaches overlay the Web with their own (weak) models...

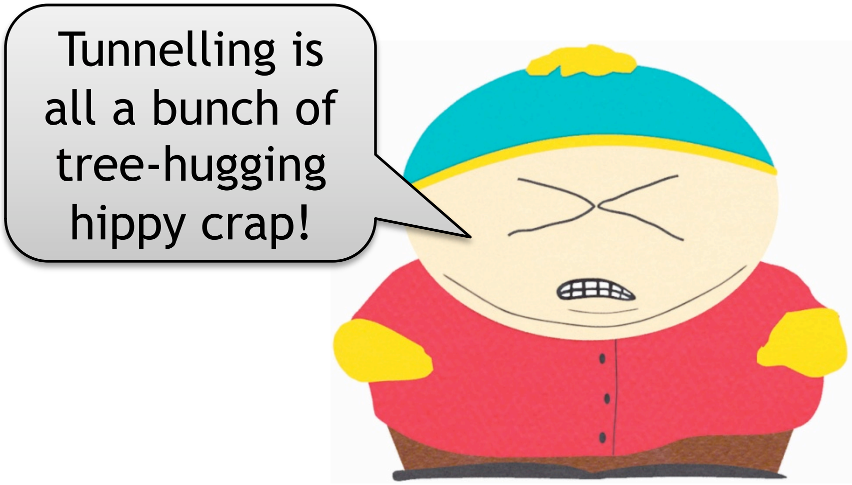


Photo: Comedy Central

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
- Lots of Web people doing 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

URI Tunnelling Pattern

- Web servers understand URIs
- URIs have structure
- Methods have signatures
- Can match URI structure to method signature
- E.g.
  - `http://example.com/addNumbers?p1=10&p2=11`
  - `int addNumbers(int i, int j) { return i + j; }`

URI Tunnelling Strengths

- Very easy to understand
- Great for simple procedure-calls
- Simple to code
  - Do it with the servlet API, HttpListener, IHttpHandler, Rails controllers, whatever!
- Interoperable
  - It’s just URIs!
- Cacheable – providing you don’t abuse GET

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 can use GET (but also POST)
  - OK for functions, but contrary to the Web for functions 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

POX Strengths

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

RPC is Commonplace Today

- To err is human, to really mess things need a computer
- To really, really mess things up you ne distributed system
  - “A Note on Distributed Computing”
- Bad Web Services and Web integratior have much in common
  - It’s RPC!
  - With latencies and nasty partial failure characteristics

</rant>

Web Fundamentals

- To embrace the Web, we need to understand how it works
  - Which means understanding RFC 2616
- 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

Why the Web was Inevitable

Tim Berners-Lee is a physicist

(Sir Tim is also a knight, but that’s not important right now)

Why the Web was Inevitable

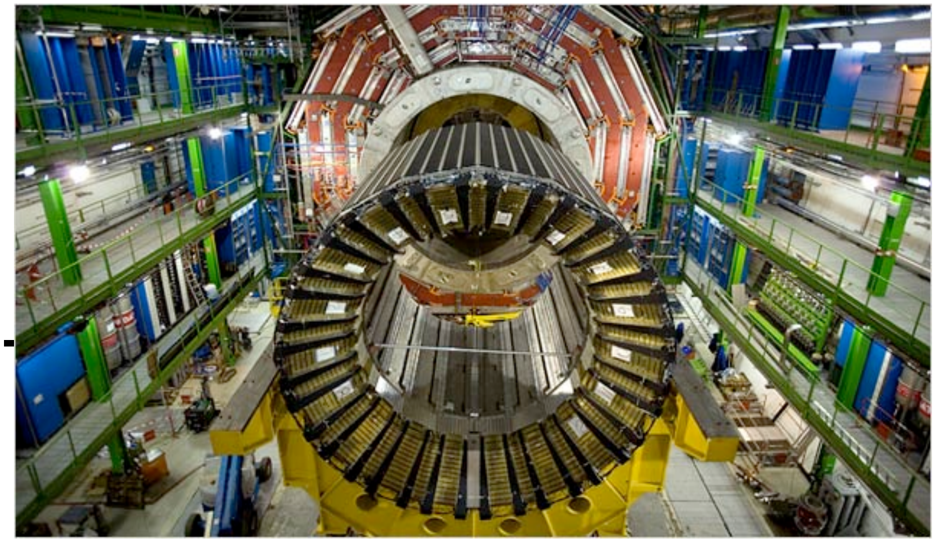
He lived in a hole in the ground

Underneath a big mountain

(in Switzerland)

Why the Web was Inevitable

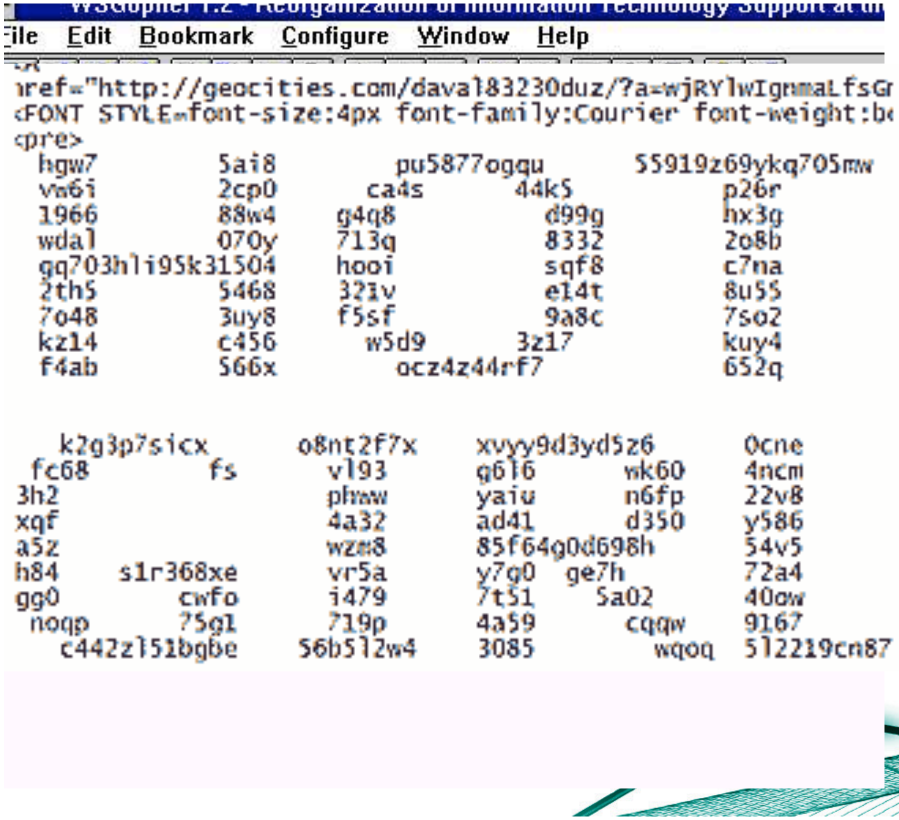
And because he was a physicist (and not yet a knight)...



...he only had a big atom-smashing thing for company

Why the Web was Inevitable

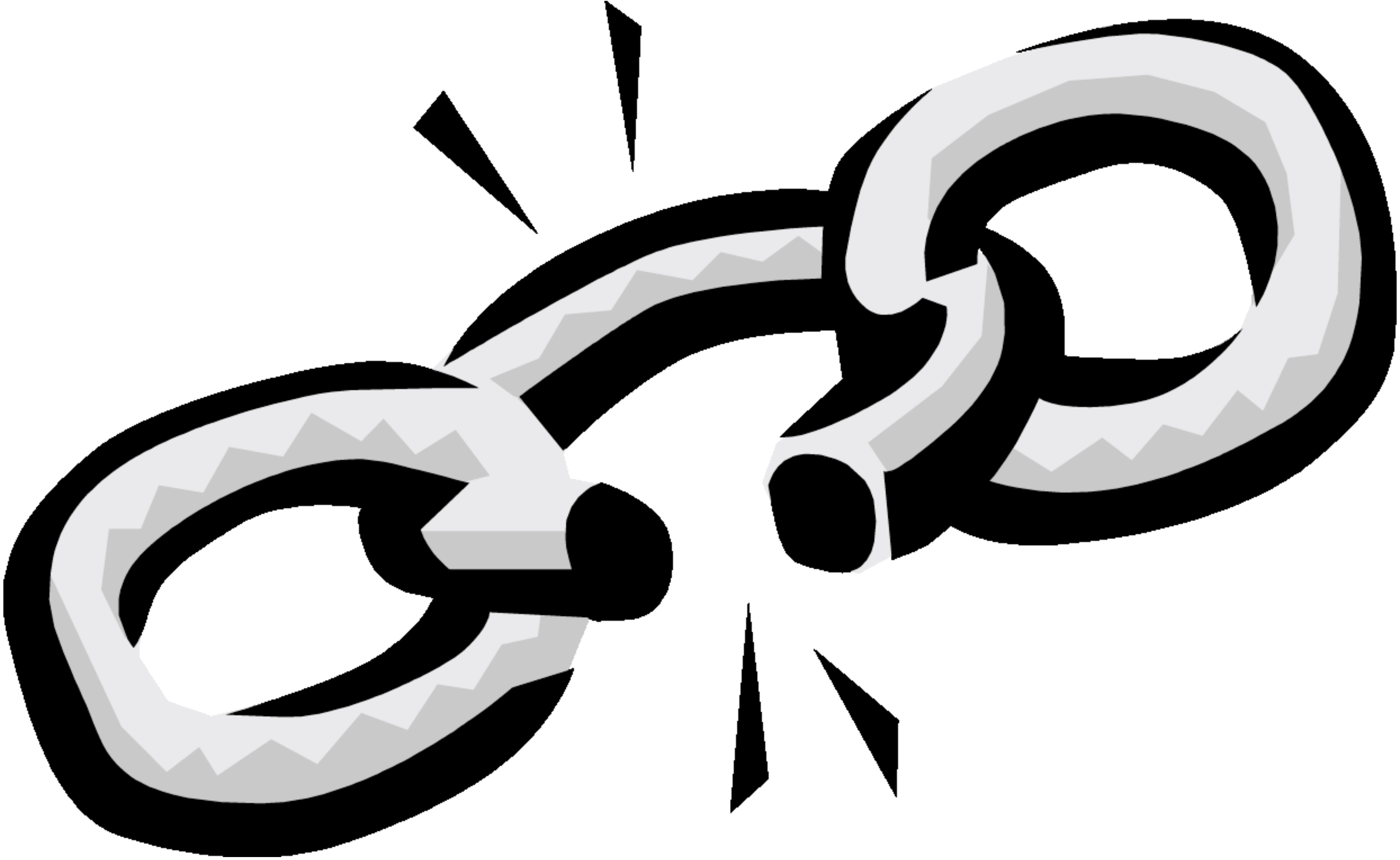
And for a lonesome physicist stuck underground with smashed up atoms for company...


WSGopher 1.2 - Reorganization of Information Technology Support at th
File
Edit
Bookmark
Configure
Window
Help

```
href="http://geocities.com/daval83230duz/?a=wjRYlwIgnmaLfsGr
<FONT STYLE=font-size:4px font-family:Courier font-weight:b
<pre>
  hgw7        5ai8        pu5877oggu      55919z69ykq705mw
  vw6i        2cp0      ca4s      44k5          p26r
  1966        88w4    g4q8          d99g        hx3g
  wdal        070y    713q          8332        2o8b
  gq703hli95k31504    hooi          sqf8        c7na
  2th5        5468    321v          e14t        8u55
  7o48        3uy8    f5sf          9a8c        7so2
  kz14        c456      w5d9      3z17          kuy4
  f4ab        566x        ocz4z44rf7            652q

   k2g3p7sicx      o8nt2f7x    xvyy9d3yd5z6      0cne
 fc68        fs      v193      g616      wk60    4ncm
3h2                  phww      yaiu      n6fp    22v8
xqf                  4a32      ad41      d350    y586
a5z                  wzm8      85f64g0d698h      54v5
h84    s1r368xe      vr5a      y7g0  ge7h        72a4
gg0        cwfo      i479      7t51    5a02      40ow
 noqp      75gl      719p      4a59      cqqw    9167
   c442z151bgbe    56b512w4    3085        wqoq  512219cn87
```

...gopher just wasn't going to cut it!

The Web broke the rules



The REST Architectural Style

- Fielding captured his interpretation of the WWW architecture in his 2000 thesis
  - REpresentational State Transfer (REST)
- Since then the Web community has been working on ways to make distributed systems behave more like the Web
  - Championed by some very vocal people!

RESTafarians?

Bob Marley
Photo by PanAfrican.tv

Mark Baker,
Photo by Paul Downey

Web Characteristics

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

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

Tenets for Web-based Services

- Resource-based
  - Rather than service-oriented (the Web is not MOM!)
- Addressability
  - Interesting things should have names
- Statelessness
  - No stateful conversations with a resource
- Representations
  - Resources can be serialised into representations
- Links
  - Resources
- Uniform Interface
  - No plumbing surprises!

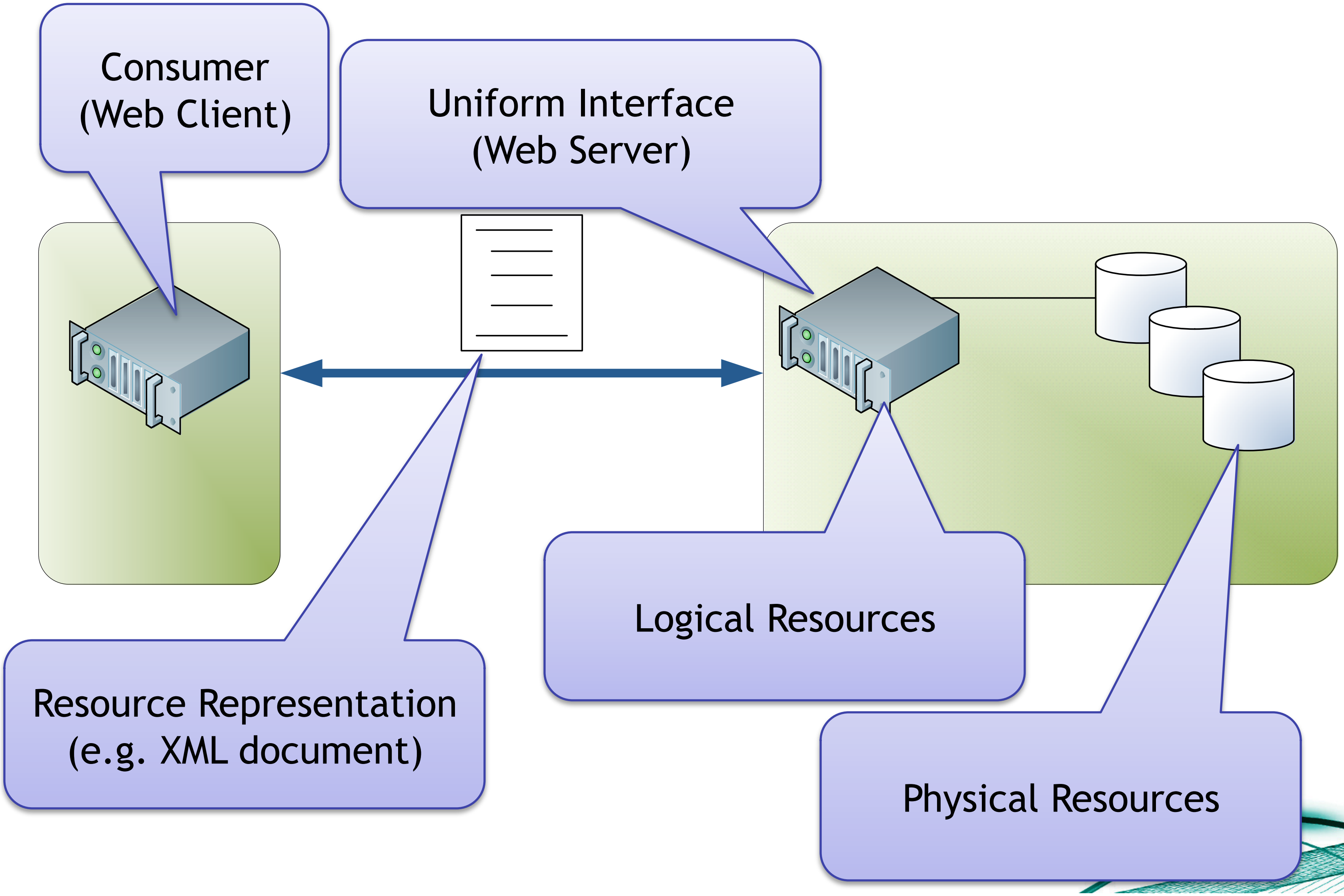
Resources

- 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?
- Making your system Web-friendly increases its surface area
  - You expose many resources, rather than fewer endpoints

Resource Representations

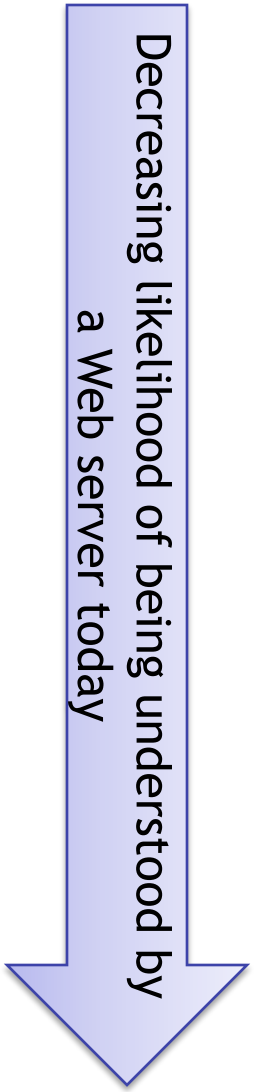
- We deal with representations of resources
  - Not the resources themselves
    - "Pass-by-value" semantics
  - Representation can be in any format
    - Any media type
- Each resource has one or more representations
  - Representations like JSON or XML are good for Web-based services
- Each resource implements the uniform HTTP interface
- Resources have standard addresses (URIs)

Resource Architecture



The HTTP Verbs

- Retrieve a representation of a resource: GET
- Get metadata about an existing resource: HEAD
- 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
- See which of the verbs the resource understands: OPTIONS



Decreasing likelihood of being understood by a Web server today

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

Common Status Codes

- 100 – Continue
- 200 – OK
- 201 – Created
- 301 – Moved Permanently
- 303 – See Other
- 304 – Not Modified

- 400 – Bad Request
- 401 – Unauthorised
- 403 – Forbidden
- 404 – Not Found
- 405 – Method Not Allowed
- 409 – Conflict
- 412 – Precondition Failed
- 500 – Internal Server Error

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

Some Useful Headers

- Authorization
  - Contains credentials (basic, digest, WSSE, etc)
  - Extensible
- Content-Type
  - The resource representation form
    - E.g. application/xml, application/xhtml+xml
- ETag/If-None-Match
  - Opaque identifier – think "checksum" for resource representations
  - Used for conditional operations, GET optimisation
- If-Modified-Since/Last-Modified
  - Used for conditional operations, GET optimisation
- 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
- WWW-Authenticate
  - Used with 401 status
    - Tells client what authentication is needed

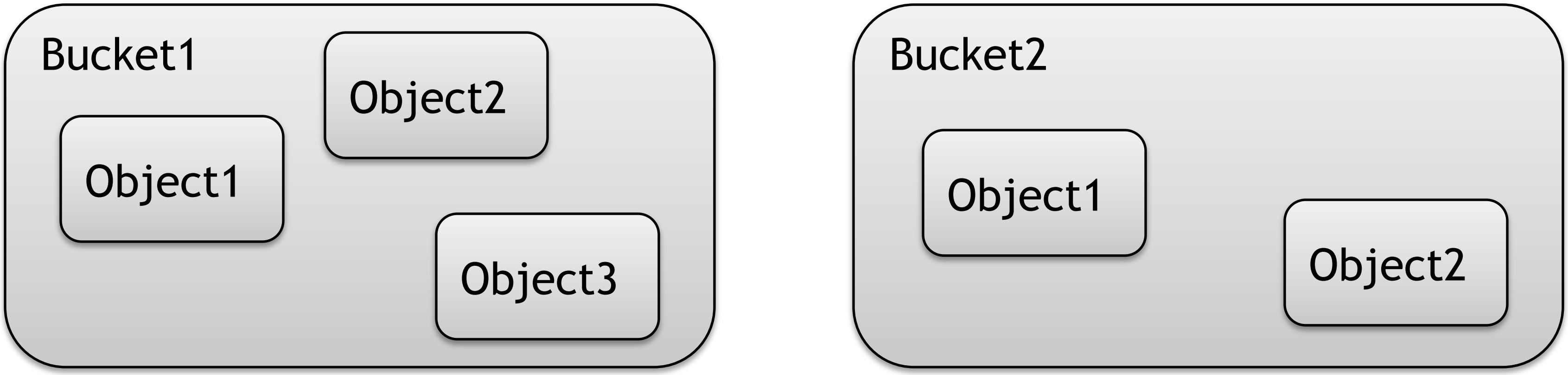
URIs

- Resource 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
- URIs should be opaque?
  - http://tinyurl.com/6
  - TimBL says “opque URIs are cool”
    - Convey no semantics, can’t infer anything from them
      - Can’t introduce coupling

Newsflash: TAG decrees that transparent URIs are OK after all. Use with care!

URI Templates, in brief

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



Bucket1

Object2

Object1

Object3

Bucket2

Object1

Object2

URI Templates in Action

- Once you can reason about a URI, you can apply the standard HTTP techniques to it
  - Because of the uniform interface
- You have metadata for each resource
  - OPTIONS, HEAD
  - Which yield permitted verbs and resource representations
- Can program against this easily using Web client libraries and regular expressions

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

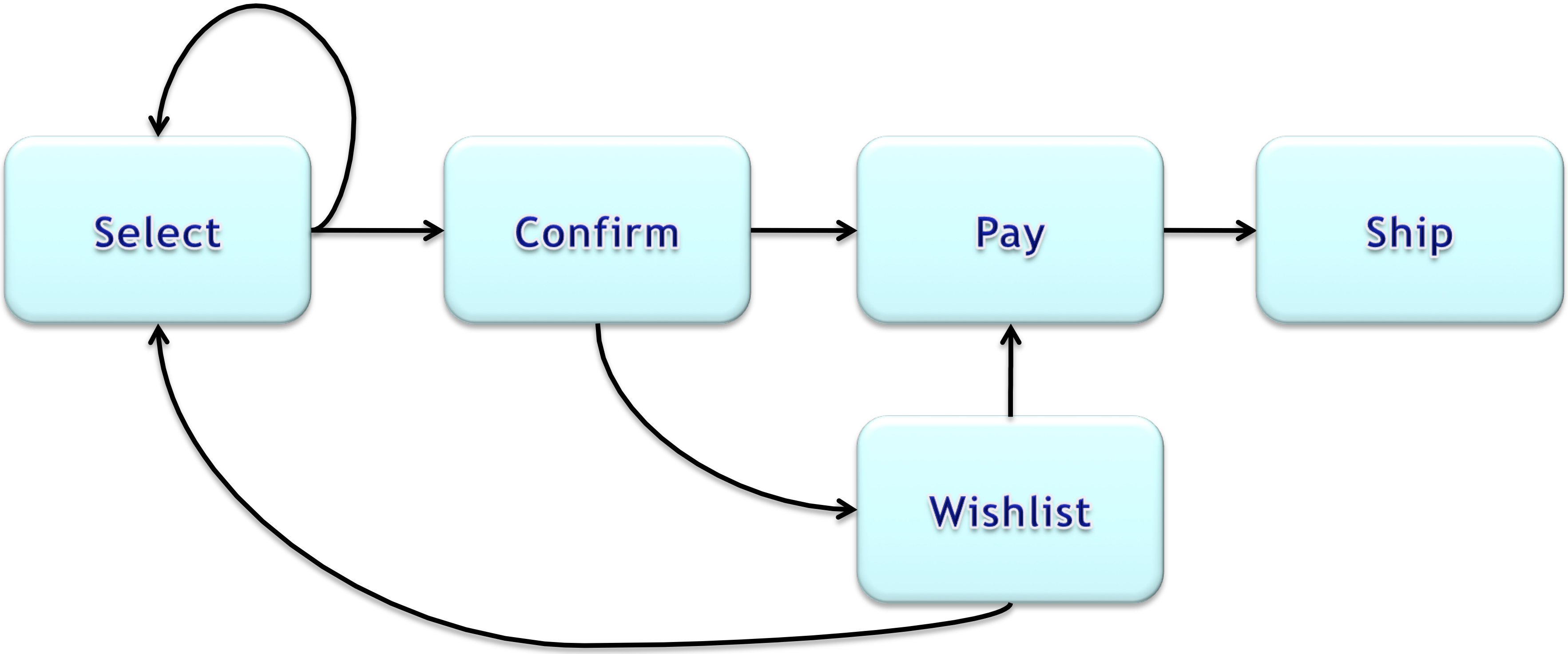
We have a comprehensive model for distributed computing…

… but we still need a way of programming it.

Describing Contracts with Links

- The value of the Web is its “linked-ness”
  - Links on a Web page constitute a contract for 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

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

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

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/x.atom+xml" href="{post-uri}" title="some title">`
- The `rel` attribute describes the semantics of the referred resource

“Subjunctive Programming”

- With changing contracts embedded as part of a resource, we can’t be too imperative anymore
- Think “subjunctive”
- Code for Web integration by thinking “what if” rather than “if then”
  - The Web is declarative!

We have a framework!

- The Web gives us a processing and metadata model
  - Verbs and status codes
  - Headers

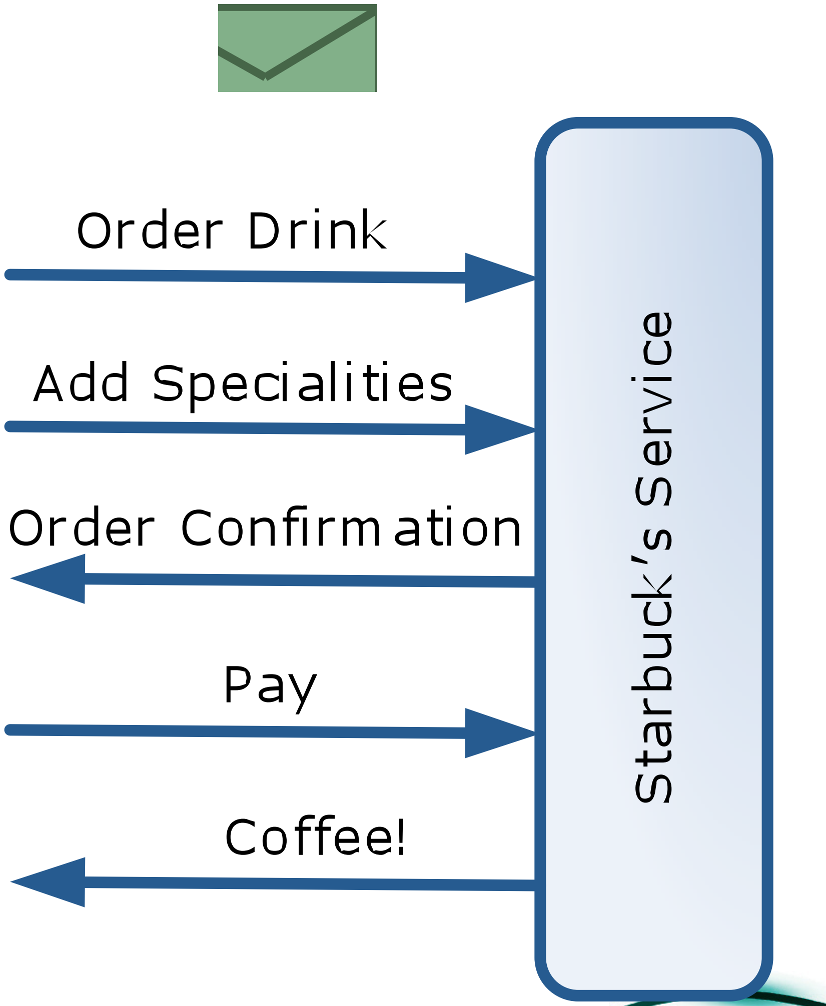
- Gives us metadata contracts or Web “APIs”
  - URI Templates
  - Links

Workflow

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

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"

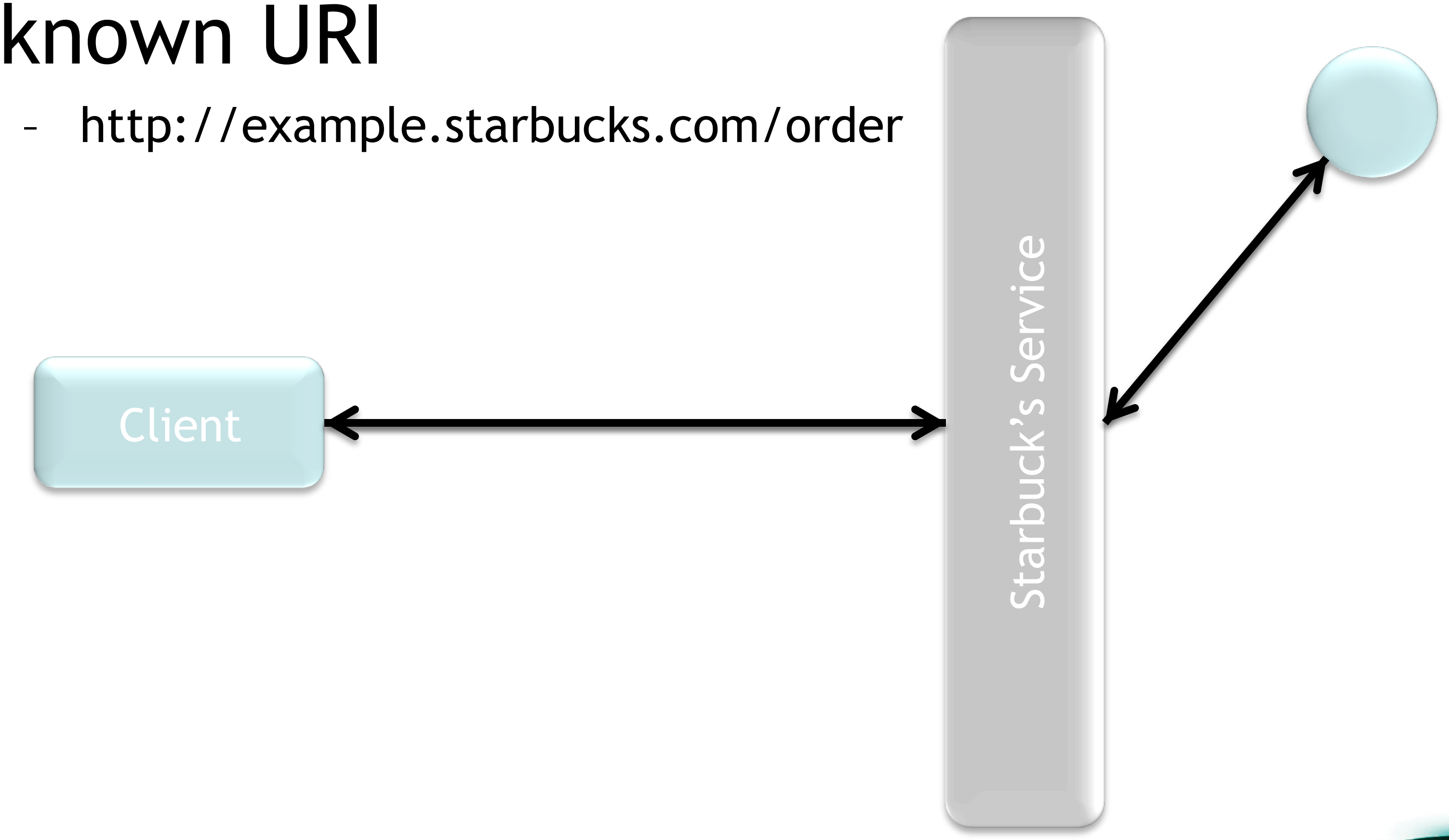


Web-friendly Workflow

- What happens if workflow stages are modelled as resources?
- And state transitions are modelled as hyperlinks or URI templates?
- And events modelled by traversing links and changing resource states?
- Answer: we get Web-friendly workflow
  - With all the quality of service provided by the Web

Placing an Order

- Place your order by POSTing it to a well-known URI
  - http://example.starbucks.com/order



Placing an Order: On the Wire

- Request

```
POST /order HTTP 1.1
Host: starbucks.example.com
Content-Type: application/xml
Content-Length: ...

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

- Response

```
201 Created
Location: http://
   starbucks.example.com/order?
   1234
Content-Type: application/xml
Content-Length: ...

<order xmlns="urn:starbucks">
  <drink>latte</drink>
 <link rel="payment"
   href="https://
   starbucks.example.com/
   payment/order?1234"
  type="application/xml"/>
</order>
```

If we have a (private) microformat, this can become a neat API!

Whoops! A mistake

- I like my coffee to taste like coffee!
- I need another shot of espresso
  - What are my OPTIONS?

Request

```
OPTIONS /order?1234 HTTP 1.1

Host: starbucks.example.com
```

Response

```
200 OK

Allow: GET, PUT
```

Phew! I can update my order, for now

Optional: Look Before You Leap

- See if the resource has changed since you submitted your order
  - If you’re fast your drink hasn’t been prepared yet

- Request

```
PUT /order?1234 HTTP 1.1

Host: starbucks.example.com

Expect: 100-Continue
```

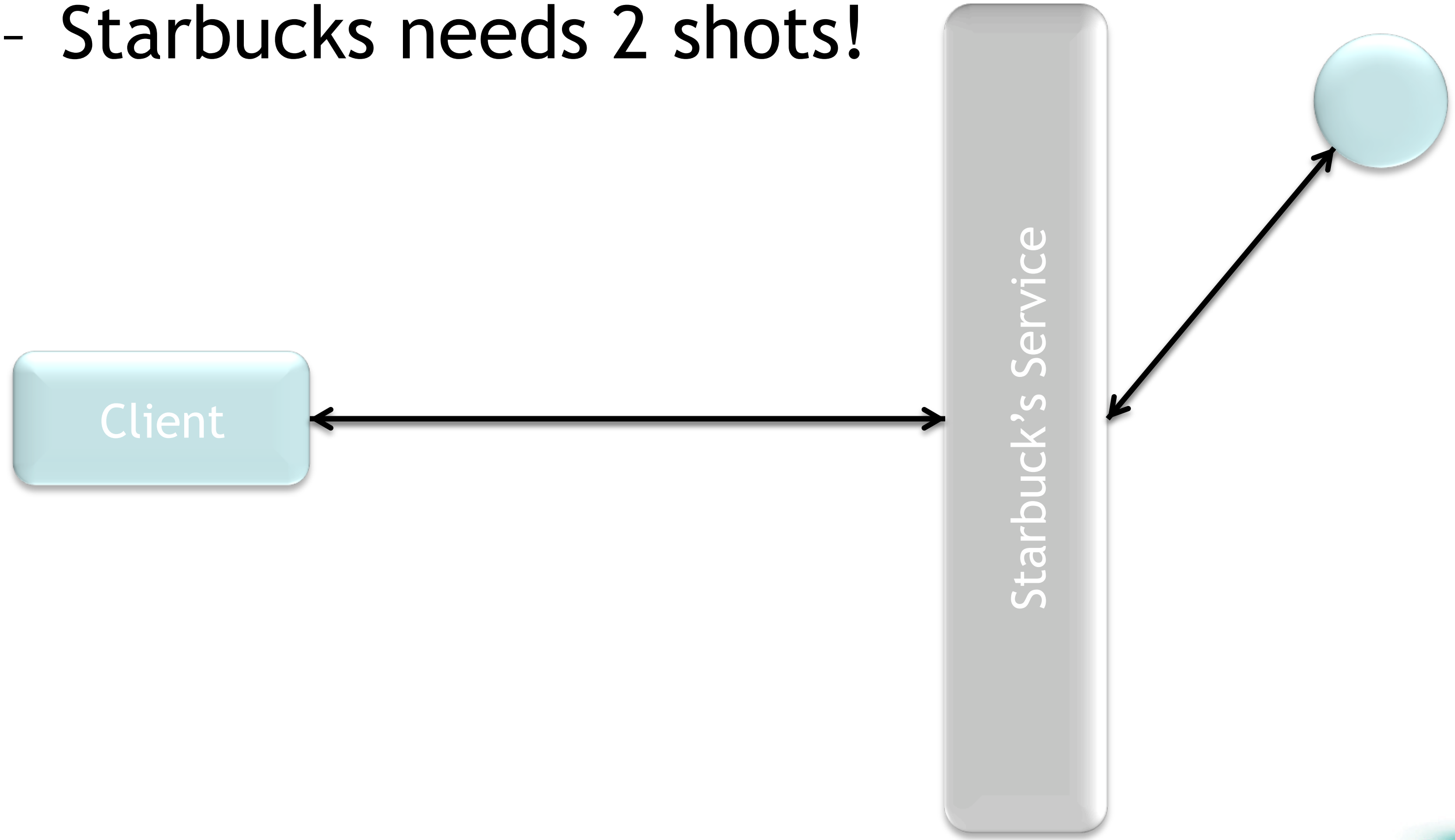
- Response

```
100 Continue
```

I can still PUT this resource, for now. (417 Expectation Failed otherwise)

Amending an Order

- Add specialities to you order via PUT
  - Starbucks needs 2 shots!



Amending an Order: On the Wire

- Request

```
PUT /order?1234 HTTP 1.1
Host: starbucks.example.com
Content-Type: application/xml
Content-Length: ...

<order xmlns="urn:starbucks">
  <drink>latte</drink>
  <additions>shot</additions>
  <link rel="payment"
   href="https://
   starbucks.example.com/payment/
   order?1234"
  type="application/xml"/>
</order>
```

- Response

```
200 OK
Location: http://
   starbucks.example.com/order?
   1234
Content-Type: application/xml
Content-Length: ...

<order xmlns="urn:starbucks">
  <drink>latte</drink>
  <additions>shot</additions>
 <link rel="payment"
   href="https://
   starbucks.example.com/payment/
   order?1234"
  type="application/xml"/>
</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 Precondition Failed` 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

```
PUT /order?1234 HTTP 1.1
Host: starbucks.example.com
...
```

○ Response

```
409 Conflict
```

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

○ Request

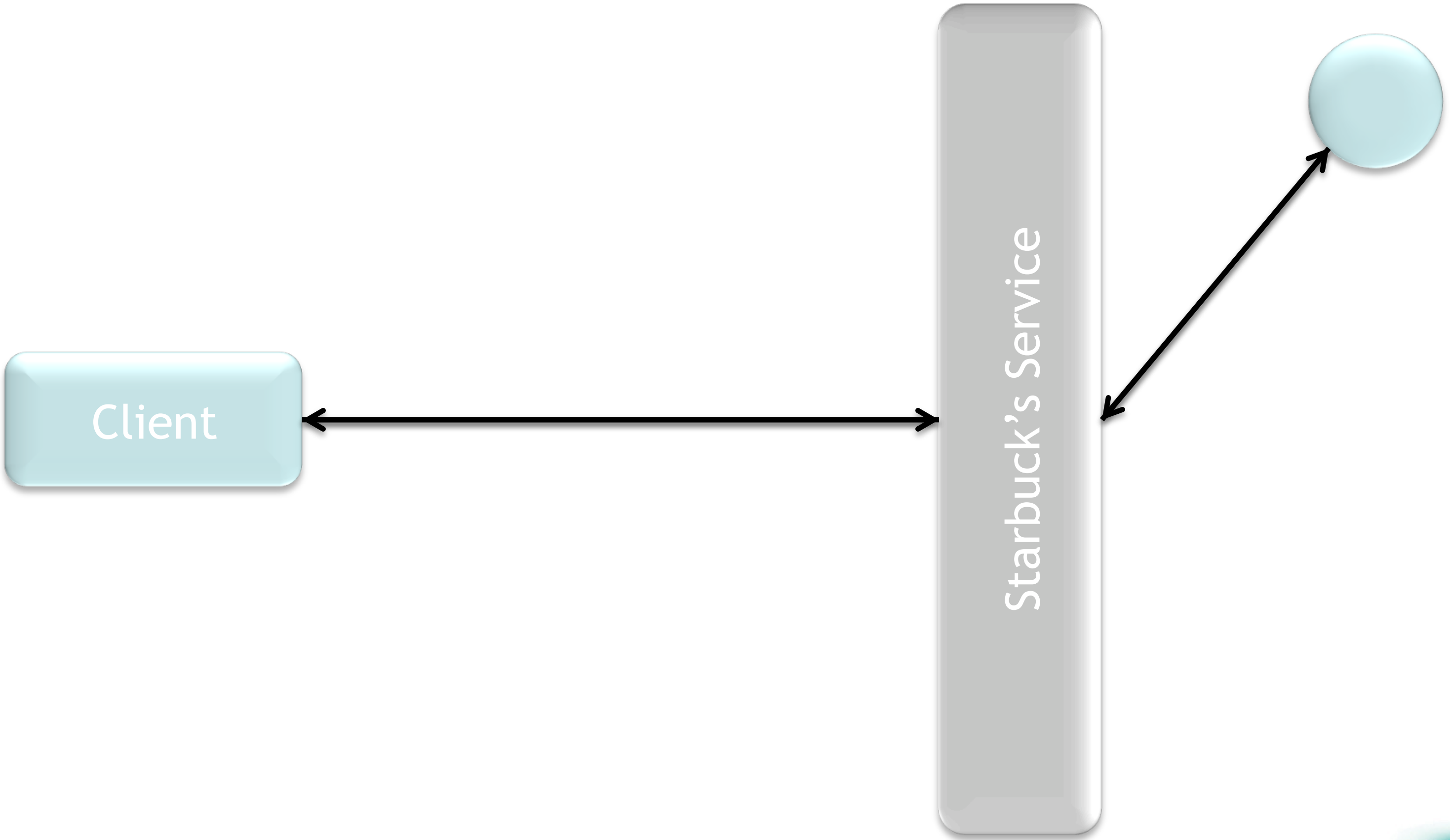
```
OPTIONS /order?1234 HTTP 1.1
Host: starbucks.example.com
```

○ Response

```
Allow: GET
```

Order Confirmation

- Check your order status by GETing it



Order Confirmation: On the Wire

- Request

```
GET /order?1234 HTTP 1.1
Host: starbucks.example.com
Content-Type: application/xml
Content-Length: ...
```

- Response

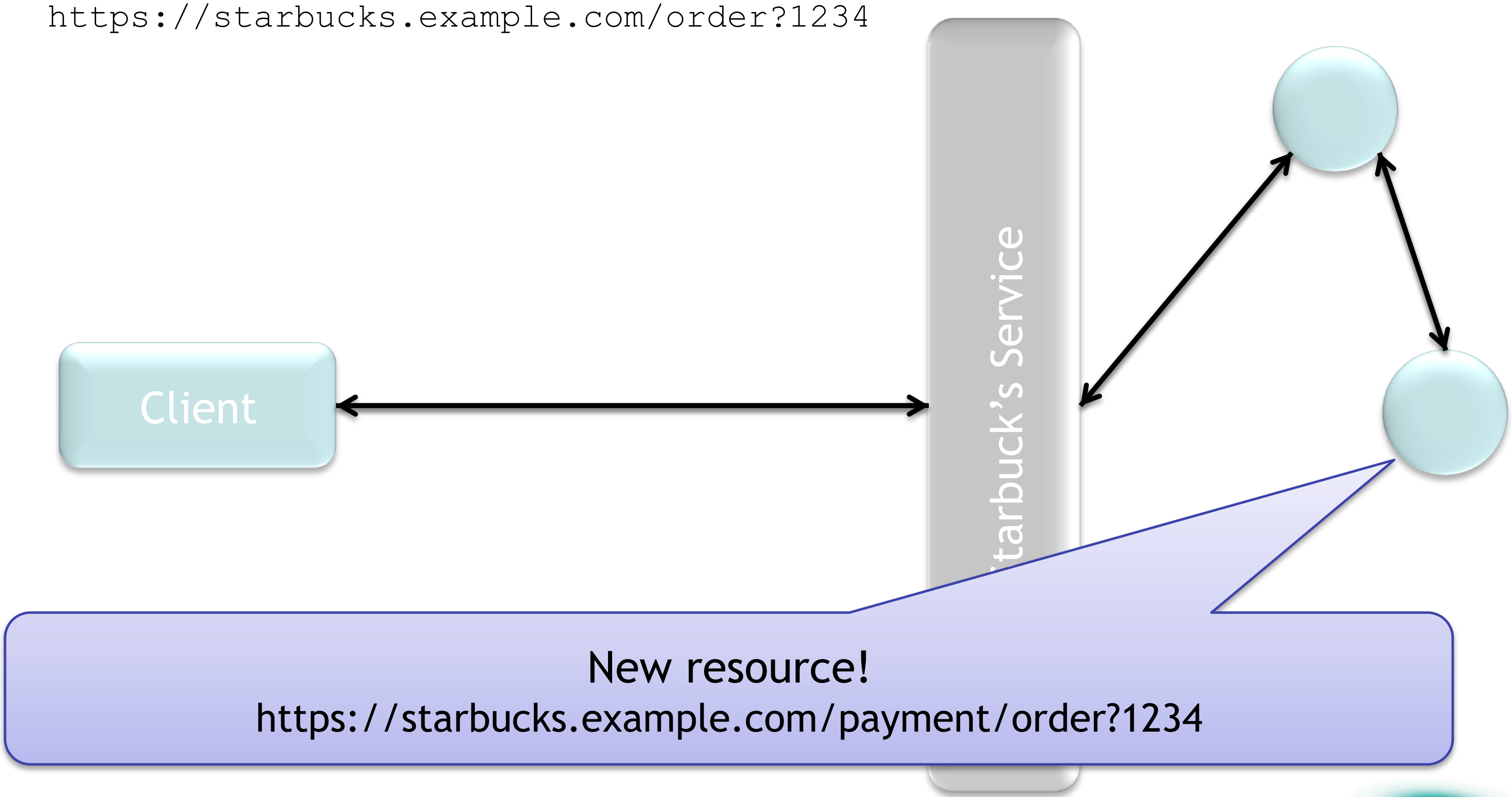
```
200 OK
Location: http://
   starbucks.example.com/order?1234
Content-Type: application/xml
Content-Length: ...


<order xmlns="urn:starbucks">
  <drink>latte</drink>
  <additions>shot</additions>
 <link rel="payment" href="https://
   starbucks.example.com/order?1234"
  type="application/xml"/>
</order>
```

Are they trying to tell me something?

Order Payment

- POST your payment to the order resource

`https://starbucks.example.com/order?1234`



New resource!
https://starbucks.example.com/payment/order?1234

How did I know to POST?

- The client knew the URI to POST to from the link
- Verified with OPTIONS
  - Just in case you were in any doubt ☺

Request

```
OPTIONS /order?1234 HTTP 1.1

Host: starbucks.example.com
```

Response

```
Allow: GET, POST
```

Order Payment: On the Wire

- Request

```
POST /order?1234 HTTP 1.1
Host: starbucks.example.com
Content-Type: application/xml
Content-Length: ...

<payment xmlns="urn:starbucks">
  <cardNo>123456789</cardNo>
  <expires>07/07</expires>
  <name>John Citizen</name>
  <amount>4.00</amount>
</payment>
```

- Response

```
201 Created
Location: https://
   starbucks.example.com/
   payment/order?1234
Content-Type: application/xml
Content-Length: ...


<payment xmlns="urn:starbucks">
  <cardNo>123456789</cardNo>
  <expires>07/07</expires>
  <name>John Citizen</name>
  <amount>4.00</amount>
</payment>
```

Check that you’ve paid

- Request

```
GET /order?1234 HTTP 1.1
Host: starbucks.example.com
Content-Type: application/xml
Content-Length: ...
```

My “API” has changed, because I’ve paid enough now

- Response

```
200 OK
Content-Type: application/xml
Content-Length: ...


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

What Happened Behind the Scenes?

- Starbucks can use the same resources!
- Plus some private resources of their own
  - Master list of coffees to be prepared
- Authenticate to provide security on some resources
  - E.g. only Starbuck's are allowed to view payments

Payment

- Only Starbucks systems can access the record of payments
  - Using the URI template: `http://.../payment/order?{order_id}`
- We can use HTTP authorisation to enforce this

Request

```
GET /payment/order?1234 HTTP 1.1
Host: starbucks.example.com
```

Response

```
401 Unauthorized
WWW-Authenticate: Digest
realm="starbucks.example.com",
qop="auth", nonce="ab656...",
opaque="b6a9..."
```

Request

```
GET /payment/order?1234 HTTP 1.1
Host: starbucks.example.com
Authorization: Digest username="jw"
realm="starbucks.example.com“
nonce="..."
uri="payment/order?1234"
qop=auth
nc=00000001
cnonce="..."
reponse="..."
opaque="..."
```

Response

```
200 OK
Content-Type: application/xml
Content-Length: ...

<payment xmlns="urn:starbucks">
  <cardNo>123456789</cardNo>
  <expires>07/07</expires>
  <name>John Citizen</name>
  <amount>4.00</amount>
</payment>
```

Master Coffee List

- `/orders` URI for all orders, only accepts GET
  - Anyone *can* use it, but it is only *useful* for Starbuck's
  - It's not identified in any of our public APIs anywhere, but the back-end systems know the URI

Request

```
GET /orders HTTP 1.1

Host: starbucks.example.com
```

Atom feed!

Response

```
200 OK
Content-Type: application/xml
Content-Length: ...

<?xml version="1.0" ?>
<feed xmlns="http://www.w3.org/2005/Atom">
 <title>Coffees to make</title>
 <link rel="alternate" href="http://
example.starbucks.com/order.atom"/>
 <updated>2007-07-10T09:18:43Z</updated>
 <author><name>Johnny Barrista</name></author>
 <id>urn:starkbucks:45ftis90</id>

 <entry>
  <link rel="alternate" type="application/xml"
href="http://starbucks.example.com/order?1234"/>
  <id>urn:starbucks:a3tfpfz3</id>
 </entry>
 ...
</feed>
```

Finally drink your coffee...

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

What did we learn from Starbuck's?

- 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
- 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…

Summary

- Both Web and WS-* are about externalising state machines when done well
  - Conversation state machines for Web Services
  - Hypermedia state machines for Web
- Use Web for massive scalability, fault tolerance
  - If you can tolerate higher latencies
- The Web is now starting to feel the love from middleware vendors too – beware!

Questions?

Developing Web-based Services
(working title)

Jim Webber
Savas Parastatidis
Ian Robinson

Coming 2008…

Blog:

http://jim.webber.name