Intro to REST

Joe Gregorio
Google

REST is an Architectural Style

Shaker Architectural Style

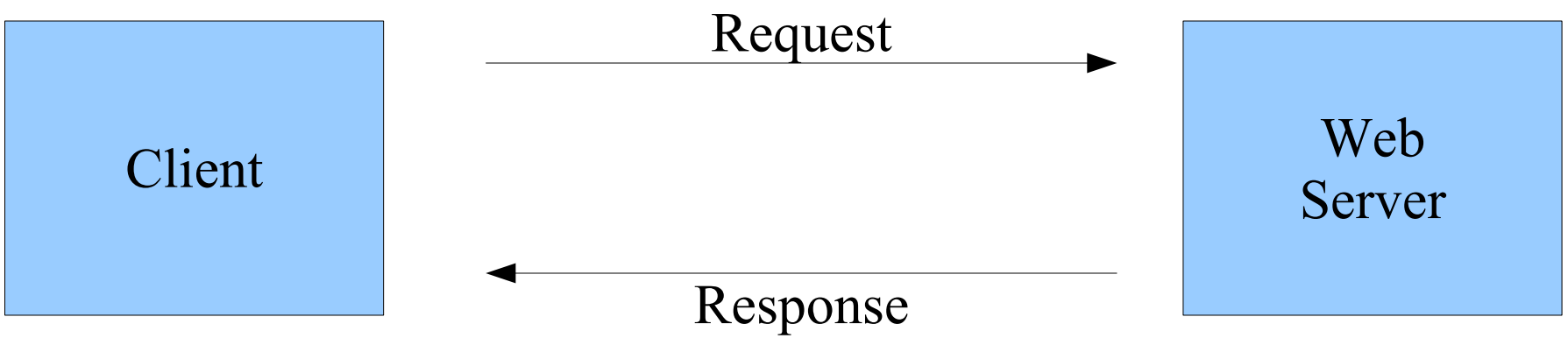
REST Architectural Style

HTTP

HTTP

Why?

The Web



Request

```
GET /news/ HTTP/1.1
Host: example.org
Accept-Encoding: compress, gzip
User-Agent: Python-httplib2
```

Response

```
HTTP/1.1 200 Ok
Date: Thu, 07 Aug 2008 15:06:24 GMT
Server: Apache
ETag: "85a1b765e8c01dbf872651d7a5"
Content-Type: text/html
Cache-Control: max-age=3600

<!DOCTYPE HTML>
...
```

```
GET /news/ HTTP/1.1
Host: example.org
Accept-Encoding: compress, gzip
User-Agent: Python-httplib2
```

Resource = http://example.org/news/

```
GET /news/ HTTP/1.1
Host: example.org
Accept-Encoding: compress, gzip
User-Agent: Python-httplib2
```

Method = GET

Methods

GET – Safe, Idempotent, Cacheable
PUT – Idempotent
DELETE – Idempotent
HEAD – Safe, Idempotent
POST

```
<!DOCTYPE HTML>
<html>
  <head>
    ...
```

Representation

```
...
<body>
    <p><img src="logo.png">
    <a href="/home/”>Home</a>
...
```

Hypertext

```
...
  <head>
    <link href="/css/b/base.css"
      type="text/css"
      rel="stylesheet">
...
```

Hypertext

```
...
<head>
  <script src="utility.js"
    type="text/javascript">
  </script>
...
```

Code on Demand

```
...
Server: Apache
ETag: "85a1b765e8c01dbf872651d7a5"
Content-Type: text/html
Cache-Control: max-age=3600
...
```

Control Data

Characteristics

- Resources
  - URI
  - Uniform Interface
    - Methods
    - Representation
- Protocol
  - Client-Server
  - Stateless
  - Cacheable
  - Layered

Characteristics

- Resources
  - **URI**
  - Uniform Interface
    - Methods
    - Representation
- Protocol
  - Client-Server
  - Stateless
  - Cacheable
  - Layered

Characteristics

- Resources
  - URI
  - **Uniform Interface**
    - Methods
    - Representation
- Protocol
  - Client-Server
  - Stateless
  - Cacheable
  - Layered

Characteristics

- Resources
  - URI
  - Uniform Interface
    - **Methods**
    - Representation
- Protocol
  - Client-Server
  - Stateless
  - Cacheable
  - Layered

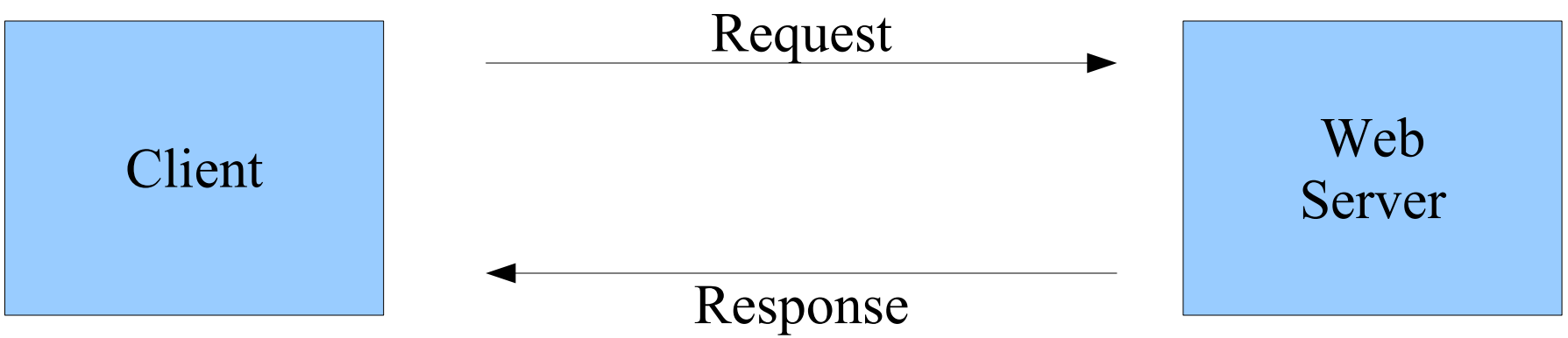
Characteristics

- Resources
  - URI
  - Uniform Interface
    - Methods
    - **Representation**
- Protocol
  - Client-Server
  - Stateless
  - Cacheable
  - Layered

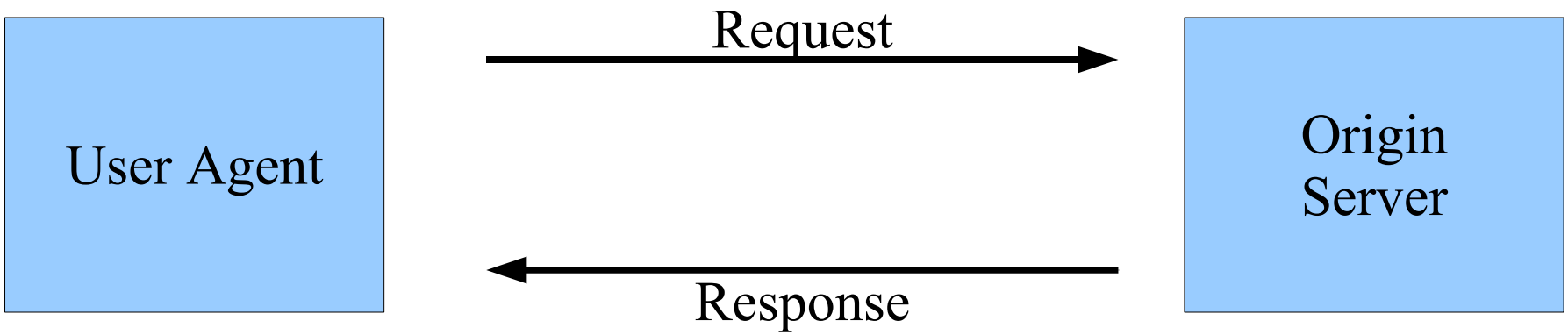
Characteristics

- Resources
  - URI
  - Uniform Interface
    - Methods
    - Representation
- Protocol
  - **Client-Server**
  - Stateless
  - Cacheable
  - Layered

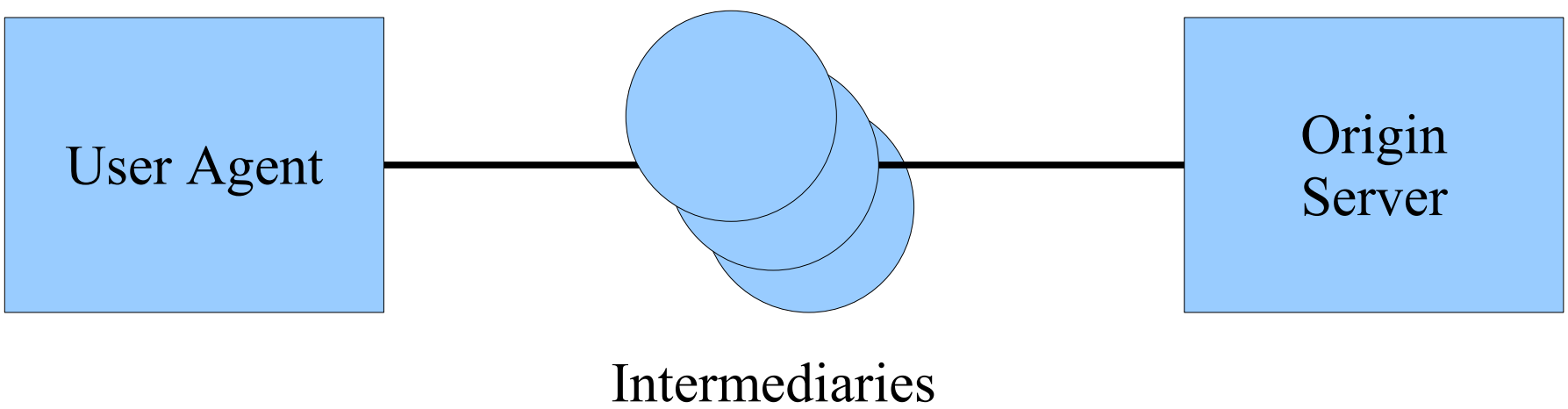
The Web



The Web



Intermediaries



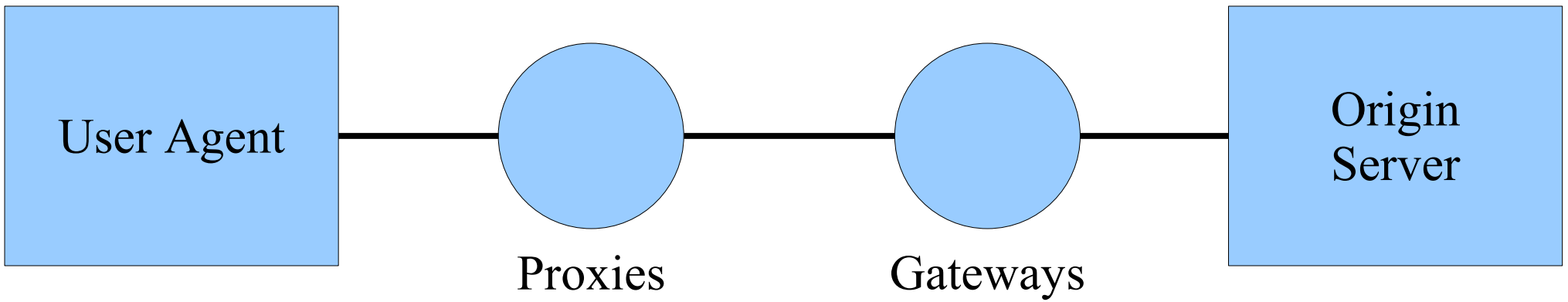
Characteristics

- Resources
  - URI
  - Uniform Interface
    - Methods
    - Representation
- Protocol
  - Client-Server
  - Stateless
  - Cacheable
  - **Layered**

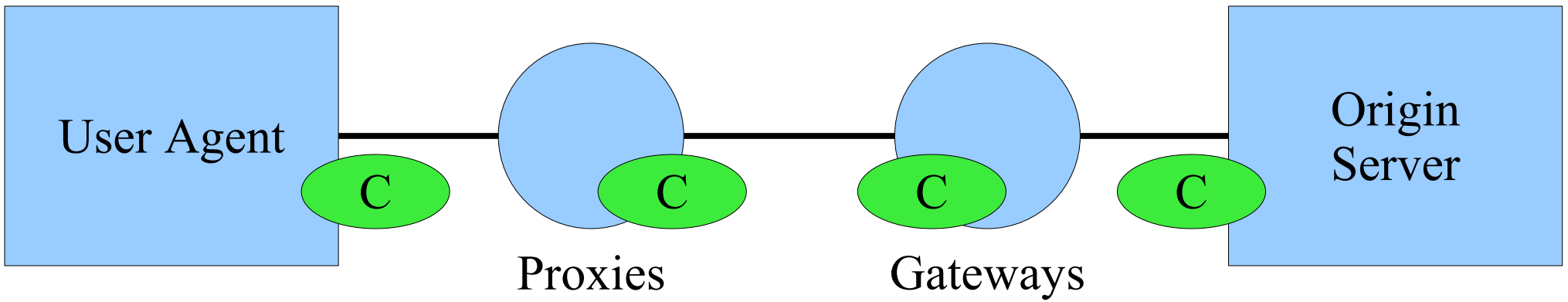
Characteristics

- Resources
  - URI
  - Uniform Interface
    - Methods
    - Representation
- Protocol
  - Client-Server
  - **Stateless**
  - Cacheable
  - Layered

Intermediaries

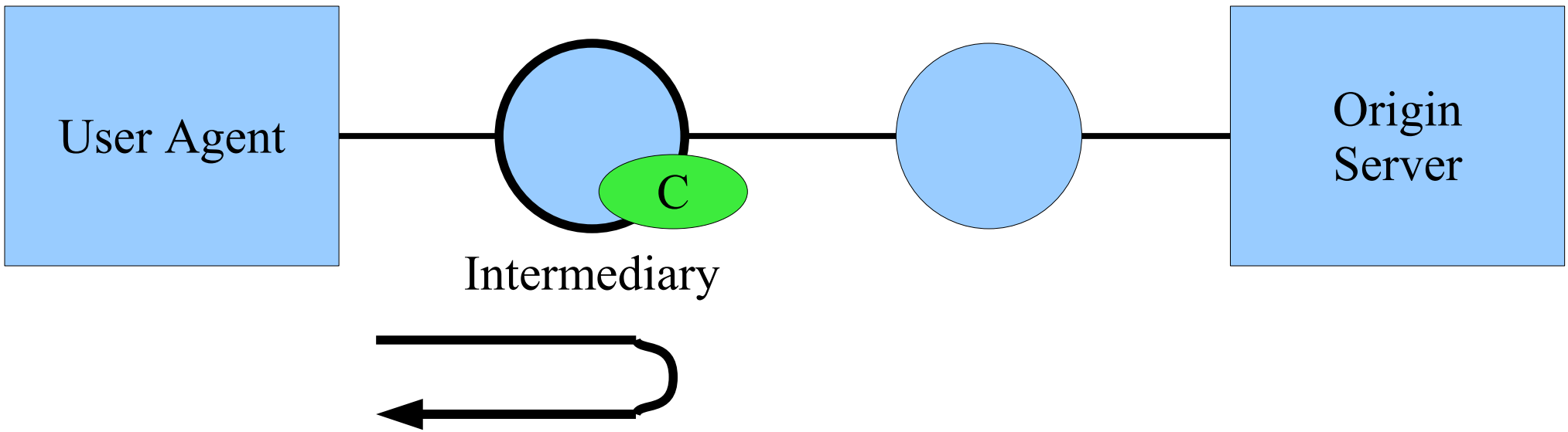


Intermediaries



```
...
Server: Apache
ETag: "85a1b765e8c01dbf872651d7a5"
Content-Type: text/html
Cache-Control: max-age=3600
...
```

Cache Hit



Characteristics

- Resources
  - URI
  - Uniform Interface
    - Methods
    - Representation
- Protocol
  - Client-Server
  - Stateless
  - **Cacheable**
  - Layered

Benefits

Network Performance

- Efficiency
- Scalability
- User Perceived Performance

Benefits

Network Performance

- **Efficiency**
- Scalability
- User Perceived Performance

Benefits

Network Performance

- Efficiency
- **Scalability**
- User Perceived Performance

Benefits

Network Performance

- Efficiency
- Scalability
- **User Perceived Performance**

Other Benefits

- simplicity
- evolvability
- extensibility
- customizability
- configuration
- reusability
- visibility
- portability
- reliability

Benefits

Aren't Free

Comparison

XML-RPC

Atom Publishing Protocol

XML-RPC

It's remote procedure calling using HTTP as the transport and XML as the encoding. XML-RPC is designed to be as simple as possible, while allowing complex data structures to be transmitted, processed and returned.

http://www.xmlrpc.com/

```
POST /RPC2 HTTP/1.0
User-Agent: Frontier/5.1.2 (WinNT)
Host: betty.userland.com
Content-Type: text/xml
Content-length: 181

<?xml version="1.0"?>
<methodCall>
  <methodName>getStateName</methodName>
...
```

```
HTTP/1.1 200 OK
Connection: close
Content-Length: 158
Content-Type: text/xml
Date: Fri, 17 Jul 1998 19:55:08 GMT

<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
      <value>
        <string>Maine</string>
...
```

```
POST /RPC2 HTTP/1.0
User-Agent: Frontier/5.1.2 (WinNT)
Host: betty.userland.com
Content-Type: text/xml
Content-length: 181

<?xml version="1.0"?>
<methodCall>
<methodName>getStateName</methodName>
...
```

```
POST /RPC2 HTTP/1.0
User-Agent: Frontier/5.1.2 (WinNT)
Host: betty.userland.com
Content-Type: text/xml
Content-length: 181

<?xml version="1.0"?>
<methodCall>
<methodName>getStateName</methodName>
...
```

Atom Publishing Protocol

The Atom Publishing Protocol (AtomPub) is an application-level protocol for publishing and editing Web resources. The protocol is based on HTTP transfer of Atom-formatted representations. The Atom format is documented in the Atom Syndication Format.

[RFC 5023]

Service Document

A document that describes the location and capabilities of one or more Collections, grouped into Workspaces.

[RFC 5023]

GET a Service Document

```
GET /collection/ HTTP/1.0
Host: example.com
```

```
HTTP/1.1 200 Ok
Date: Thu, 14 Aug 2008 23:26:31 GMT
Server: Apache
Content-Length: 753
Vary: Accept-Encoding,User-Agent
Content-Type: application/atomsvc+xml

<?xml version="1.0" encoding="utf-8"?>
<service
    xmlns="http://www.w3.org/2007/app"
   ...
```

```
...
<collection href="entry/">
  <atom:title>entry</atom:title>
...
```

GET a Collection

```
GET /collection/entry/ HTTP/1.0
Host: example.com
```

```
HTTP/1.1 200 Ok
Date: Thu, 14 Aug 2008 23:26:31 GMT
Server: Apache
Content-Length: 753
Vary: Accept-Encoding,User-Agent
Content-Type: application/atom+xml

<?xml version="1.0" encoding="utf-8"?>
<feed
   xmlns="http://www.w3.org/2005/Atom"
...
```

```
<?xml version="1.0" encoding="utf-8"?>
<feed
  ...
  <title type="text">Example </title>
  <link href="?page=1" rel="next" />
  ...
```

```
...
<entry>
  <title>Lists I Like</title>
  <link
    href="http://example.com/entry/2"
    rel="edit" />
...
```

As you can see, benefits

Long-lived Images

Set the cache for images to very long time. If you need to update the image, upload a new image to a new URI and change the HTML to point to that new URI.

HTML

```
...
<img src='/image/big-image.png'>
...
```

Image

```
HTTP/1.1 200 Ok
Date: Thu, 15 Aug 2008 23:26:31 GMT
Server: Apache
Content-Length: 50753
Cache-Control: max-age=2592000
...
```

HTML

```
...
<img src='/image/big-image-2.png'>
...
```

Further Reading

- RFC 2616
- RFC 3986
- Architectural Styles and the Design of Network-based Software Architectures
- Caching Tutorial

Hi, I'm Joe Gregorio and I work at Google in Developer Relations. This talk in on REST and in the talk I presume you are familiar with the Atom Publishing Protocol. If you aren't then you can watch my video "An Introduction to the Atom Publishing Protocol" and then come back and watch this video.

So let's begin.

REST is an Architectural Style

What is REST?

==========

You may have seen or heard the term REST, which comes from

Roy Fielding's Thesis and stands for Representational State Transfer.

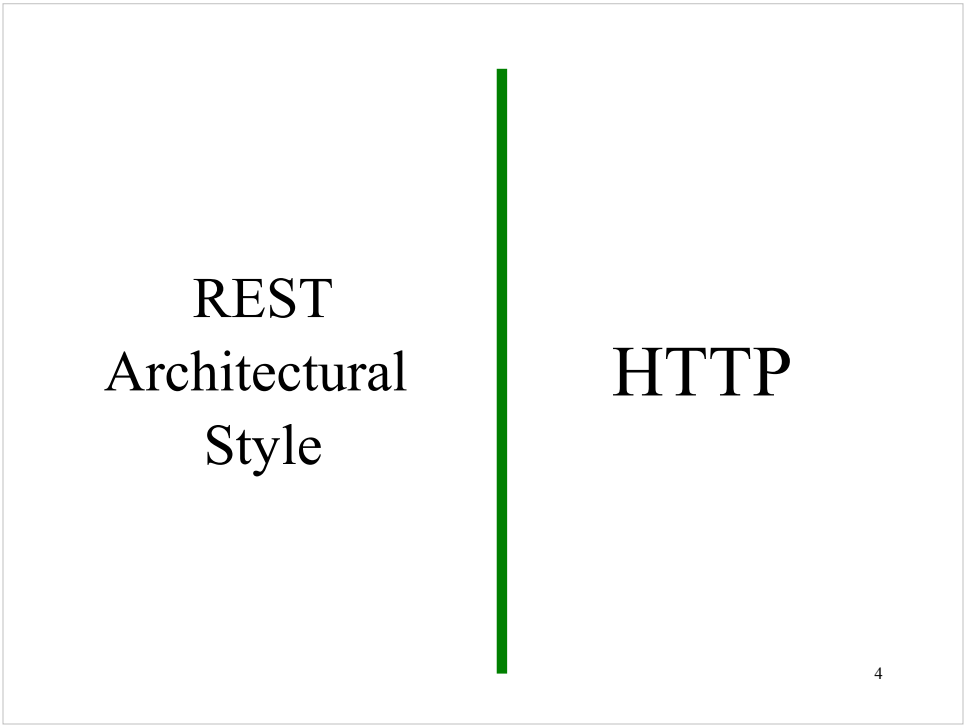
It is an architectural style.

Shaker Architectural Style

3

http://www.flickr.com/photos/worobod/322627448/
CC Attribution

Now an architectural style is an abstraction, as opposed to a concrete thing. For example, this shaker house is different than the Shaker Architectural Style. The "architectural style" of Shaker defines the attributes or characteristics you would see in a house built in that style.



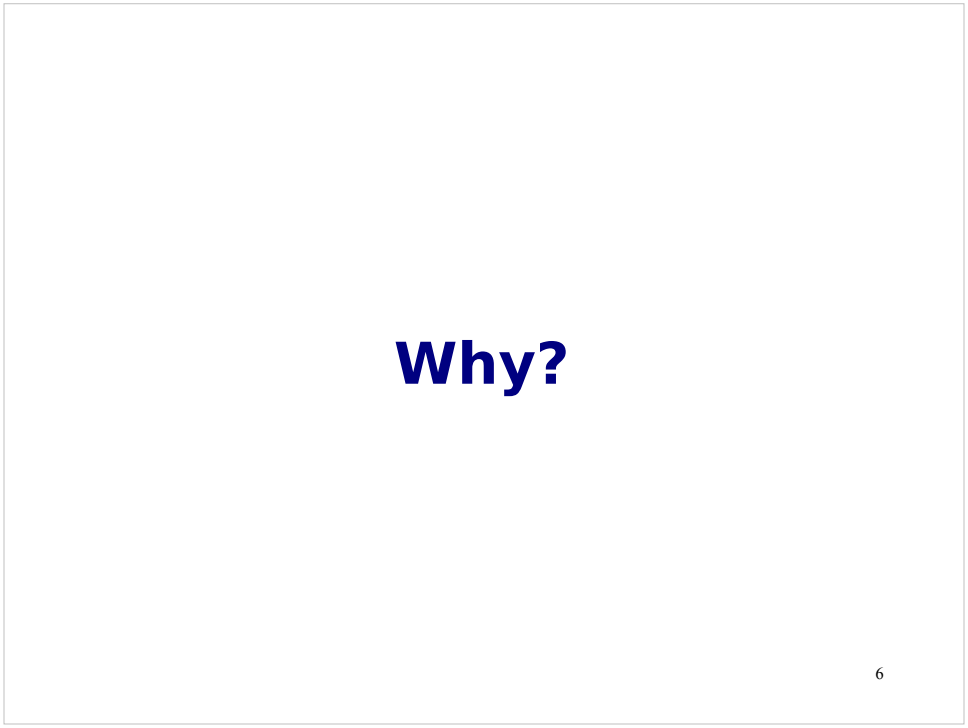
In the same way, the REST Architectural Style is a set of architectural constraints you should see in a protocol built in that style.

HTTP

5

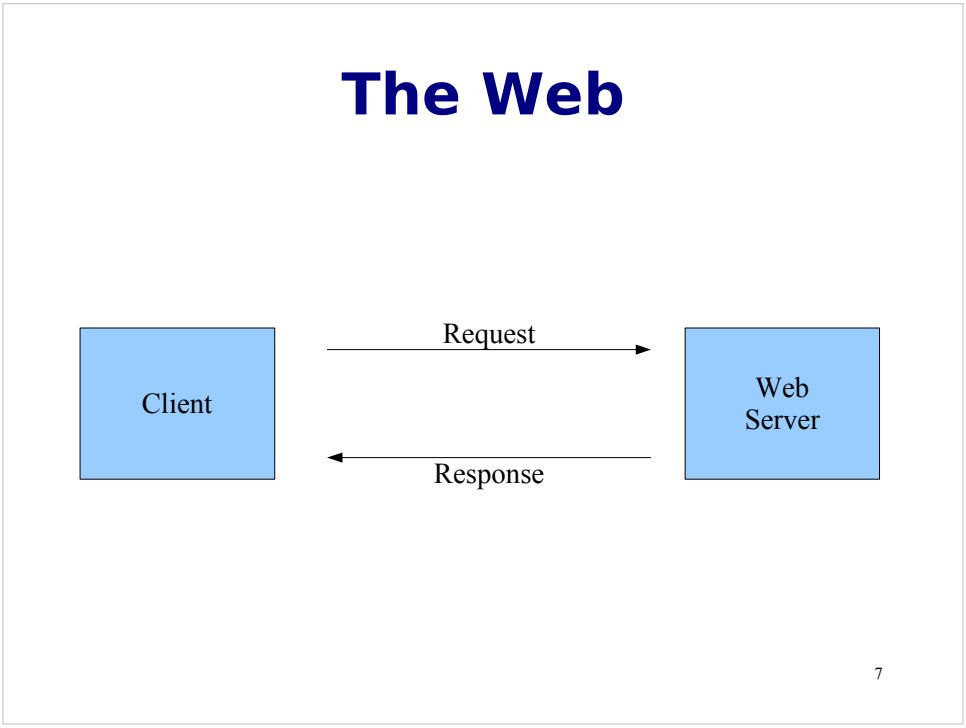
HTTP is one such protocol, and for the remainder of this talk we're going to just talk about HTTP.

Now it's simply not possible to cover every aspect of HTTP so at the end of this presentation there will be a further reading list.



So why should you care about REST? It's the architecture of the web as it works today, and if you're going to be building applications that run on the web, shouldn't you be working *with* that architecture, instead of against it?

Hopefully you'll see as we go through this video that there are many opportunities to increase the performance and scalability of your application, and solve some traditionally tricky problems by working with HTTP and taking full advantage of it's capabilities.



Let's get some of the basics down - some nomenclature and the operation of HTTP.

At its simplest HTTP is a simple request-response protocol, your browser makes a request and the server sends a response. The beauty of the web is that it appears very simple, as if your browser talks directly to a single server.

Request

```
GET /news/ HTTP/1.1
Host: example.org
Accept-Encoding: compress, gzip
User-Agent: Python-httplib2
```


let's look in detail at a specific request and response

Here is a GET request to http://example.org/news/

Response

```
HTTP/1.1 200 Ok
Date: Thu, 07 Aug 2008 15:06:24 GMT
Server: Apache
ETag: "85a1b765e8c01dbf872651d7a5"
Content-Type: text/html
Cache-Control: max-age=3600

<!DOCTYPE HTML>
...
```


And here is the response

```
GET /news/ HTTP/1.1
Host: example.org
Accept-Encoding: compress, gzip
User-Agent: Python-httplib2
```

Resource = http://example.org/news/

The request is to a resource identified by a URI.

In this case http://example.org/news/

Resources, or addressability is very important,

```
GET /news/ HTTP/1.1
Host: example.org
Accept-Encoding: compress, gzip
User-Agent: Python-httplib2
```

Method = GET

There is a method, the action on that resource

Methods

GET – Safe, Idempotent, Cacheable
PUT – Idempotent
DELETE – Idempotent
HEAD – Safe, Idempotent
POST

There is a small set of methods and they have specific functions and specific characteristics

```
<!DOCTYPE HTML>
<html>
  <head>
    ...
```

Representation

The representation is the body, in this case an HTML document

```
...
<body>
   <p><img src="logo.png">
   <a href="/home/”>Home</a>
...
```

Hypertext

HTML is a form of hypertext, which means it has links to other resources, here is a tradition link that you would click on

```
...
  <head>
    <link href="/css/b/base.css"
      type="text/css"
      rel="stylesheet">
...
```

Hypertext

15

but there is more than one kind of link, here is a link to a CSS document, which will provide styling for the page

```
...
<head>
  <script src="utility.js"
    type="text/javascript">
  </script>
...
```

Code on Demand

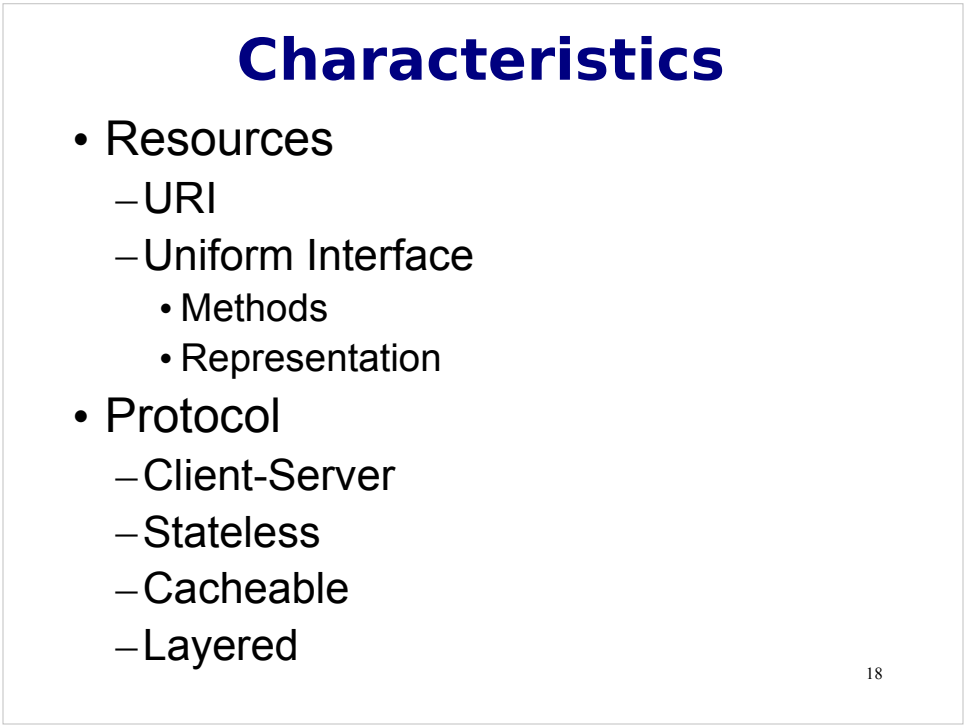
And there is also a link to some JavaScript, also hypertext example.

This one is particularly important as it is Code on Demand, the ability of loading code into the browser to execute on the client.

```
...
Server: Apache
ETag: "85a1b765e8c01dbf872651d7a5"
Content-Type: text/html
Cache-Control: max-age=3600
...
```

Control Data

The response headers show control data, such as this header which controls how long the response can be cached.



So now that we've reviewed those parts of HTTP let's look at the characteristics of a RESTful protocol:

* Resource - Application state and functionality are abstracted into **resources**

* URI - Every resource is uniquely addressable using a universal syntax for use in hypermedia links

* Uniform Interface - All resources share a **uniform interface** for the transfer of state between client and resource, consisting of

o Methods - A constrained set of well-defined **operations**

o Representation - A constrained set of content types, optionally supporting code on demand

* A protocol which is:

o Client-server

o Stateless

o Cacheable

o Layered

Characteristics

- Resources
  - **URI**
  - Uniform Interface
    - Methods
    - Representation
- Protocol
  - Client-Server
  - Stateless
  - Cacheable
  - Layered

19

Now we've already talked about many of these aspects with HTTP, that Resources are identified by URIs

Characteristics

- Resources
  - URI
  - **Uniform Interface**
    - Methods
    - Representation
- Protocol
  - Client-Server
  - Stateless
  - Cacheable
  - Layered

And that those resources have a uniform interface

Characteristics

- Resources
  - URI
  - Uniform Interface
    - **Methods**
    - Representation
- Protocol
  - Client-Server
  - Stateless
  - Cacheable
  - Layered

21

, understanding a limited set of methods such as GET, PUT, POST, DELETE, and HEAD

Characteristics

- Resources
  - URI
  - Uniform Interface
    - Methods
    - **Representation**
- Protocol
  - Client-Server
  - Stateless
  - Cacheable
  - Layered

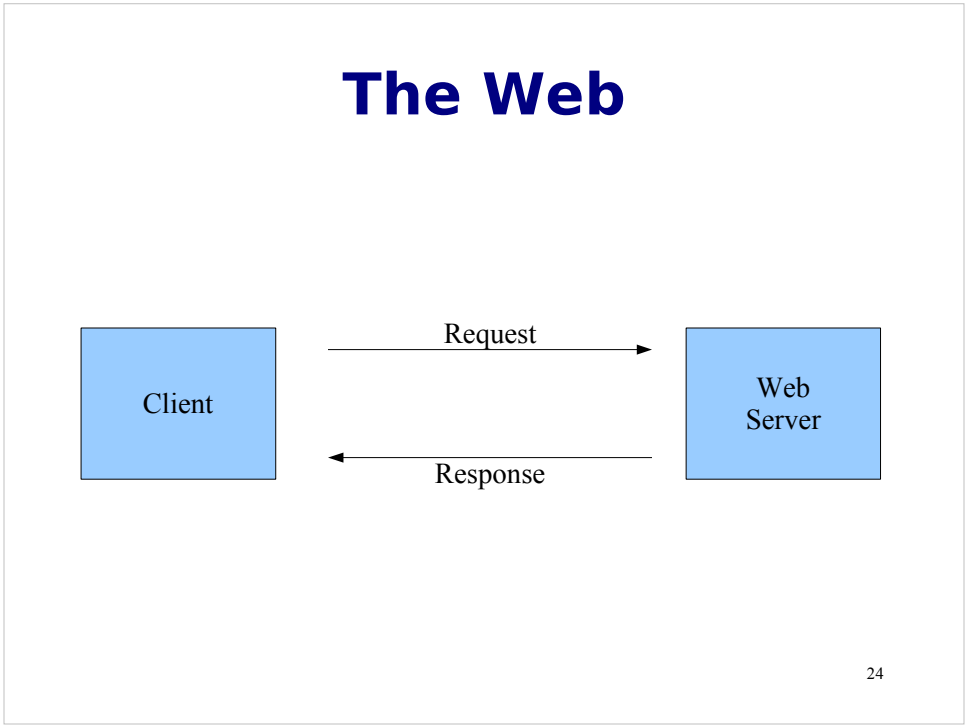
22

And that the representations sent are self-identified, a constrained set of content types, that might not only be hypertext, but could also include Code on Demand, such as the example we saw with JavaScript.

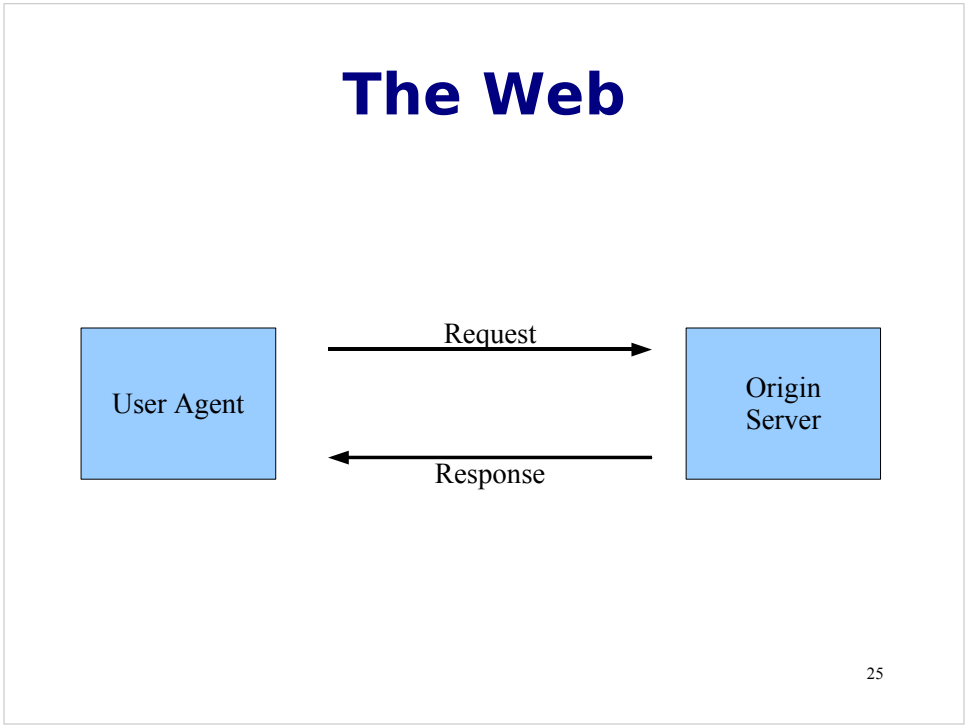
Characteristics

- Resources
  - URI
  - Uniform Interface
    - Methods
    - Representation
- Protocol
  - **Client-Server**
  - Stateless
  - Cacheable
  - Layered

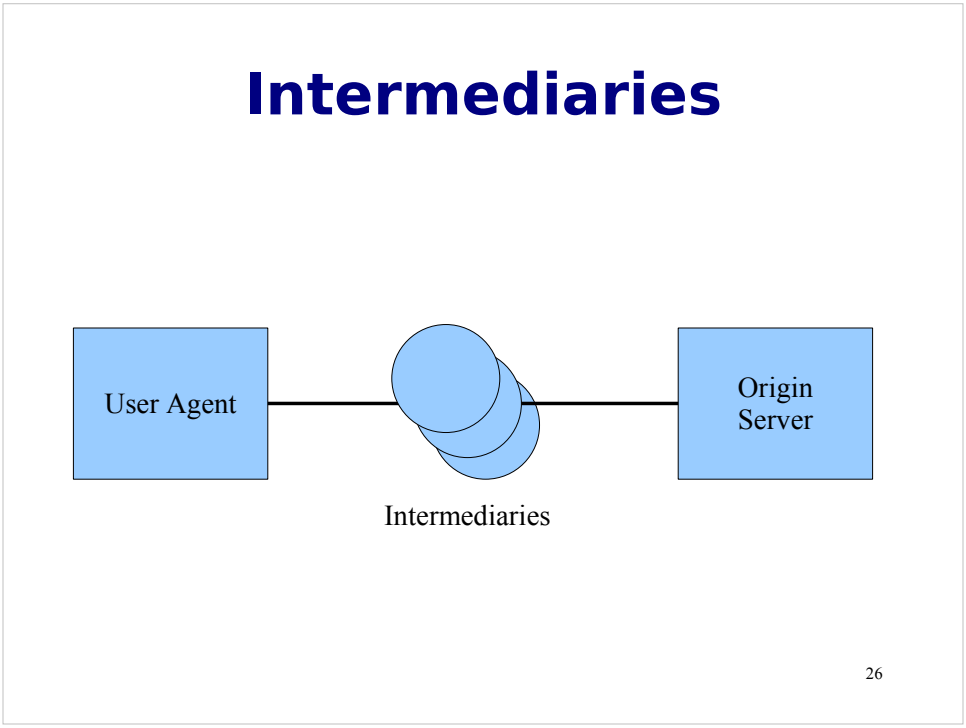
And we've even seen that HTTP is a client-server protocol. to discuss the remainder of the characteristics of the protocol we need to look at the underlying structure of the web.



We originally started out with this simplified example of how the web appears to a client. Let's switch to using the right names for each of these pieces.



They are the User Agent and the Origin Server.



But the reality is more complicated than that.

There can be many intermediaries between you and the server you're connecting to. By "intermediaries" we mean "HTTP intermediaries", which doesn't include devices at lower levels in the protocol stack like routers, modems, and access points.

Characteristics

- Resources
  - URI
  - Uniform Interface
    - Methods
    - Representation
- Protocol
  - Client-Server
  - Stateless
  - Cacheable
  - **Layered**

27

Those intermediaries are the layered part of the protocol, and that layering allow intermediaries to be added atvarious points in the request-response path without changing the interfaces between components, where they do things to passing messages such as translation or improving performance with caching

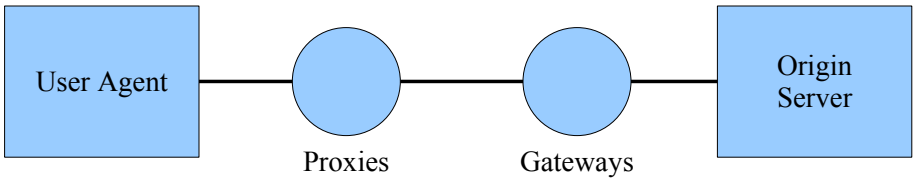
Characteristics

- Resources
  - URI
  - Uniform Interface
    - Methods
    - Representation
- Protocol
  - Client-Server
  - **Stateless**
  - Cacheable
  - Layered

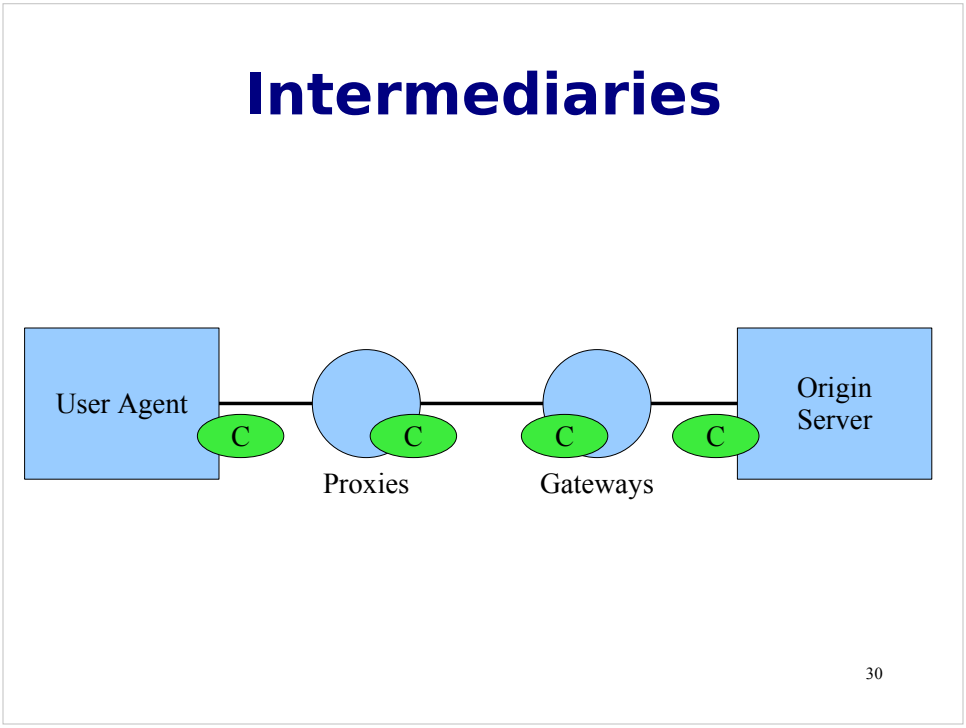
28

This is also why its important that interaction between requests is stateless, that is, each request is independent from the others, allowing the intermediaries to work on a single interaction w/o knowing the entire topology, and since different requests may travel through different intermediaries there may be no chance of visibility between interactions.

Intermediaries



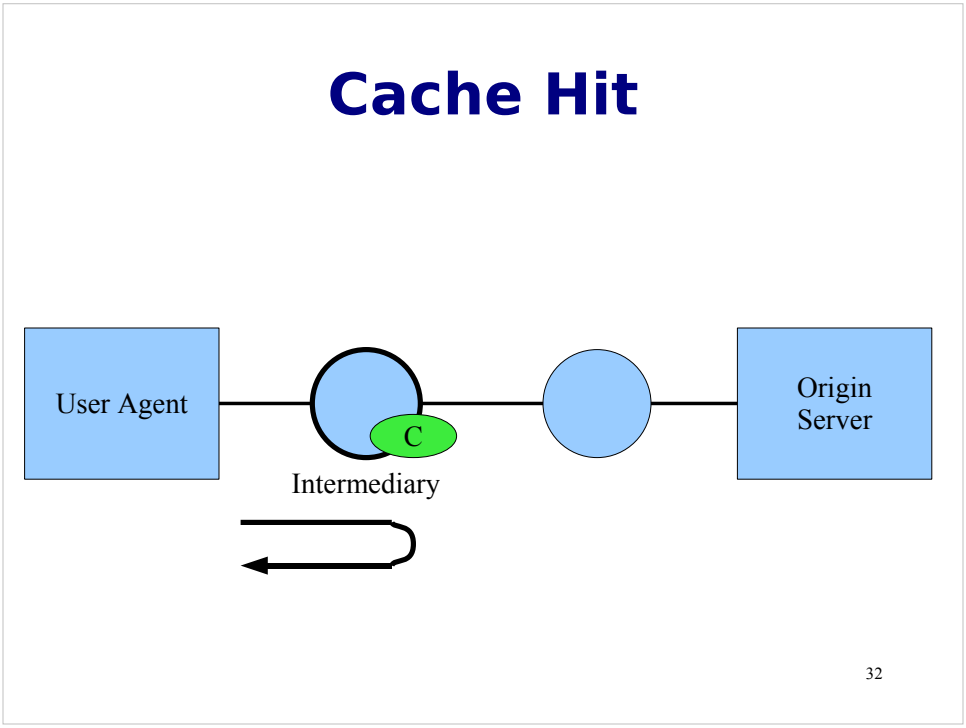
Intermediaries include proxies and gateways. Proxies are chosen by the client, while gateways are chosen by the origin server or are imposed by the network. Despite the slide showing only one proxy and gateway realize there may be several proxies and gateways between a user-agent and origin server, or there may be none.



And finally, every actor in the chain, from the user-agent, through the proxies, and to the origin server, may have a cache.

```
...
Server: Apache
ETag: "85a1b765e8c01dbf872651d7a5"
Content-Type: text/html
Cache-Control: max-age=3600
...
```


If an intermediary does caching and a response indicates that the response can be cached, in this case, for an hour then



if a new request for that resources comes within the hour, then the cached response will be returned.

Characteristics

- Resources
  - URI
  - Uniform Interface
    - Methods
    - Representation
- Protocol
  - Client-Server
  - Stateless
  - **Cacheable**
  - Layered

33

those caches finish out the major characteristics of our REST protocol.

Benefits

Network Performance

- Efficiency
- Scalability
- User Perceived Performance

now we said this architecture had benefits, what are some of those? Let's first look at some performance benefits, which include efficiency, scalability, and user perceived performance

Benefits

Network Performance

- **Efficiency**
- Scalability
- User Perceived Performance

35

HTTP is efficient because of all those caches, your request may not have to reach all the way back to the origin server, or in the case of a local user-agent cache, may never hit the network to begin with.

Control data allows the singaling of compression, so responses can be gzip'd before being sent to user-agents that can handle them.

Benefits

Network Performance

- Efficiency
- **Scalability**
- User Perceived Performance

36

Scalability comes from many areas. The use of gateways allows you to distribute traffic among a large set of origin servers based on method, URI or content-type, or any other visible control data or meta-data in the request headers.

Caching helps scalability also as it reduces the actual number of requests that hit the origin server.

Statelessness allows requests to be routed through different gateways and proxies, thus avoiding introducing bottlenecks, allowing more intermediaries to be added as needed.

Benefits

Network Performance

- Efficiency
- Scalability
- **User Perceived Performance**

37

User Perceived Performance in increased by having a reduced set of known media types, that allows browsers to handle known types much faster, for example, partial rendering of HTML documents as they download. Also, Code on Demand allows computations to be moved closer to the client, or closer to the server, depending on where the work can be done fastest. For example, having JavaScript code to do form validation before a request is even made to the network is obviously much faster than round-tripping the form values to the server and having the server return any validation errors.

Caching also helps here, as requests may not need to go completely back to the origin server, or even leave the user-agent if there is a hit in the local cache.

Also, since GET is idempotent and safe a user-agent could pre-fetch results before they are needed, thus increasing user perceived performance.

Other Benefits

- simplicity
- evolvability
- extensibility
- customizability
- configuration
- reusability
- visibility
- portability
- reliability

Lots of other benefits we won't cover, but they are enumerated in Roy's thesis.

Benefits

Aren't Free

39

But all of these benefits aren't free, you actually have to structure your application or service to take advantage of them, if you don't then you won't get any benefits.

Comparison

XML-RPC

Atom Publishing Protocol

To see how the structuring helps, lets look at two protocols XML-RPC and the Atom Publishing Protocol.

XML-RPC

It's remote procedure calling using HTTP as the transport and XML as the encoding. XML-RPC is designed to be as simple as possible, while allowing complex data structures to be transmitted, processed and returned.

http://www.xmlrpc.com/

```
POST /RPC2 HTTP/1.0
User-Agent: Frontier/5.1.2 (WinNT)
Host: betty.userland.com
Content-Type: text/xml
Content-length: 181

<?xml version="1.0"?>
<methodCall>
  <methodName>getStateName</methodName>
...
```


this is what and XML-RPC request looks like

```
HTTP/1.1 200 OK
Connection: close
Content-Length: 158
Content-Type: text/xml
Date: Fri, 17 Jul 1998 19:55:08 GMT

<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
      <value>
        <string>Maine</string>
...
```


And here's the example response.

```
POST /RPC2 HTTP/1.0
User-Agent: Frontier/5.1.2 (WinNT)
Host: betty.userland.com
Content-Type: text/xml
Content-length: 181

<?xml version="1.0"?>
<methodCall>
<methodName>getStateName</methodName>
...
```


All requests are POSTs

So what do the intermediaries see of this request/response?

Is it safe? No.

It is idempotent? No.

Cacheable? No.

```
POST /RPC2 HTTP/1.0
User-Agent: Frontier/5.1.2 (WinNT)
Host: betty.userland.com
Content-Type: text/xml
Content-length: 181

<?xml version="1.0"?>
<methodCall>
<methodName>getStateName</methodName>
...
```


And all ll requests go to the same URI, which means that if you were going to distribute many such calls among a group of origin servers you would have to look inside the body for the methodName.

This gives the least amount of information to the web, and thus doesn't get any help from intermediaries, and doesn't scale with off the shelf parts.

Atom Publishing Protocol

The Atom Publishing Protocol (AtomPub) is an application-level protocol for publishing and editing Web resources. The protocol is based on HTTP transfer of Atom-formatted representations. The Atom format is documented in the Atom Syndication Format.

[RFC 5023]

Service Document

A document that describes the location and capabilities of one or more Collections, grouped into Workspaces.

[RFC 5023]

47

For authoring to commence, a client needs to discover the capabilities and locations of the available Collections. Service Documents are designed to support this discovery process.

GET a Service Document

```
GET /collection/ HTTP/1.0
Host: example.com
```


To retrieve a service document we send a GET to its URI

This is good, GET == Safe, idempotent, cacheable, gzippable,

and as we shall see, the response is hypertext

```
HTTP/1.1 200 Ok
Date: Thu, 14 Aug 2008 23:26:31 GMT
Server: Apache
Content-Length: 753
Vary: Accept-Encoding,User-Agent
Content-Type: application/atomsvc+xml

<?xml version="1.0" encoding="utf-8"?>
<service
    xmlns="http://www.w3.org/2007/app"
   ...
```


first, the response is self-identifying via the content-type,

```
...
<collection href="entry/">
  <atom:title>entry</atom:title>
...
```

50

And it is hypertext as it contains the URIs for each of the collections. What's highlighted is a relative URI for the collection.

Once we have a collection URI we can POST an entry to create a new member, and then GET/PUT/DELETE the members at their own URIs.

GET a Collection

```
GET /collection/entry/ HTTP/1.0
Host: example.com
```


To retrieve the representation of the collection we send a GET to its URI

Again, all the same goodness, This is good, GET == Safe, idempotent, cacheable, gzippable,

and as we shall see, the response is hypertext

```
HTTP/1.1 200 Ok
Date: Thu, 14 Aug 2008 23:26:31 GMT
Server: Apache
Content-Length: 753
Vary: Accept-Encoding,User-Agent
Content-Type: application/atom+xml

<?xml version="1.0" encoding="utf-8"?>
<feed
   xmlns="http://www.w3.org/2005/Atom"
...
```


Here is an example response

```
<?xml version="1.0" encoding="utf-8"?>
<feed
  ...
  <title type="text">Example </title>
  <link href="?page=1" rel="next" />
  ...
```


And this again has hypertext, in a couple forms

The first is the “next” link, which points to the set of next entries in the collection.

```
...
<entry>
  <title>Lists I Like</title>
  <link
    href="http://example.com/entry/2"
    rel="edit" />
...
```


Lastly is the “edit” URI for the entry. This identifies a resource where the entry can be edited.

We send a GET to that URI to retrieve the full representation

We send a PUT to update it

We send a DELETE to remove it from the collection

PUTs and DELETEs can invalidate caches along the way.

Click to add title

As you can see, benefits

Long-lived Images

Set the cache for images to very long time. If you need to update the image, upload a new image to a new URI and change the HTML to point to that new URI.

A strategy for keeping large items, such as images, in caches

HTML

```
...
<img src='/image/big-image.png'>
...
```

Image

```
HTTP/1.1 200 Ok
Date: Thu, 15 Aug 2008 23:26:31 GMT
Server: Apache
Content-Length: 50753
Cache-Control: max-age=2592000
...
```


30 days

HTML

```
...
<img src='/image/big-image-2.png'>
...
```


Is we need to change the image, put it at a new URI also with long caching and update the HTML to use the new image

Further Reading

- RFC 2616
- RFC 3986
- Architectural Styles and the Design of Network-based Software Architectures
- Caching Tutorial

60

So there you go, a high level view of REST and how it relates to HTTP. Here is the list of further reading

You can learn more about caching from Mark Nottingham's Caching Tutorial

http://www.mnot.net/cache_docs/

Thanks, and have fun