Яндекс

Making Java REST with JAX-RS 2.0

April 24, 2014

Dmytro Chyzhykov

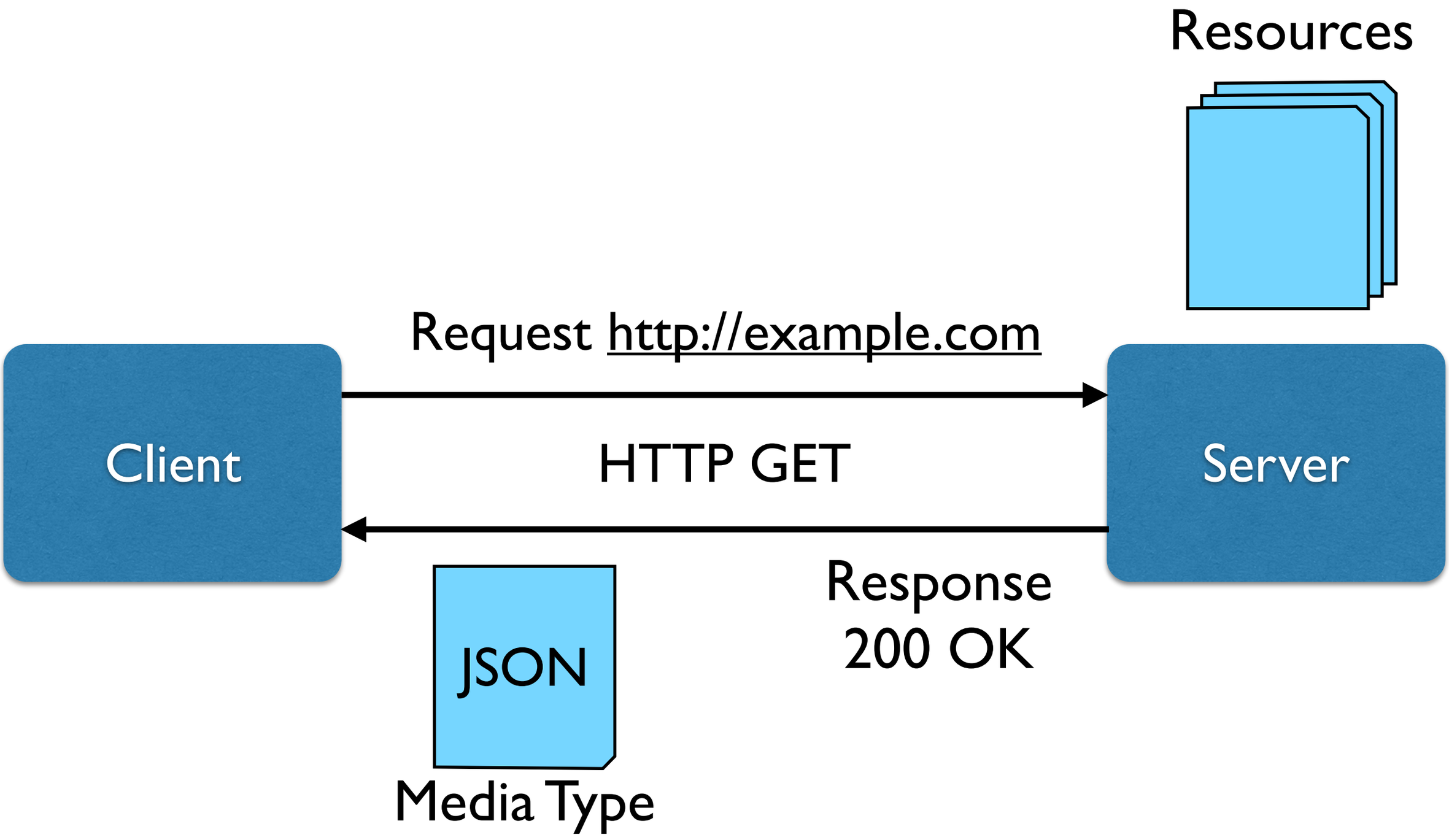
Disclaimer

The views and opinions expressed in this presentation expressed by the speaker are solely his own and do not necessary represent the views and opinions of companies he is working or worked for.

Agenda

- What is REST?
- REST Principles and JAX-RS 2.0
- Q & A

HTTP



Read more Hypertext Transfer Protocol -- HTTP/1.1

What is REST?

REST?

REST JEE7 Definition

REpresentational State Transfer is an architectural style of client-server applications centered around the **transfer** of **representations** of **resources** through requests and responses.

Serves to build loosely coupled, lightweight web services that are particularly well suited for creating APIs for clients spread out across the Internet.

Was introduced and defined by Roy T. Fielding in his doctoral dissertation.

Read more The Java EE 7 Tutorial: 29.1 What Are RESTful Web Services?

REST is...

- An architectural style, not a technology
- Everything is a Resource
- Suitable for CRUD (Create/Read/Update/Delete)
- Stateless by nature (excellent for distributed systems)
- Cacheable (naturally supported by HTTP)
- Composable code on demand applications

REST Principles

- Give every thing its own ID
- Link things together (HATEOAS)
- Use standard HTTP methods
- Resources can have multiple representations
- Communicate statelessly
- Support caching

Give every thing its own ID

Individual Resource ID

```
http://habrahabr.ru/users/theshade/
https://twitter.com/fielding
https://api.github.com/teams/46059
```

Collection Resource ID

```
https://api.github.com/teams/
https://twitter.com/fielding/followers
https://api.github.com/user/repos?page=2
http://ajax.googleapis.com/ajax/services/
search/web?v=1.0&q=Paris%20Hilton
```

JAXRS 2.0 resource example

```
GET http://locahost:8080/articles/7
GET http://locahost:8080/articles/
GET http://locahost:8080/articles/?page=1
```

JAXRS 2.0 resource example

```
GET http://locahost:8080/articles/7
GET http://locahost:8080/articles/
GET http://locahost:8080/articles/?page=1
```

```
@Path("articles")
public class ArticleResource {
}
```

JAXRS 2.0 resource example

```
GET http://locahost:8080/articles/7
GET http://locahost:8080/articles/
GET http://locahost:8080/articles/?page=1
```

```
@Path("articles")
public class ArticleResource {
   @GET
   @Path("{id}")
   public Article read(@PathParam("id") int id) {…}
}
```

JAXRS 2.0 resource example

```
GET http://locahost:8080/articles/7
GET http://locahost:8080/articles/
GET http://locahost:8080/articles/?page=1
```

```
@Path("articles")
public class ArticleResource {
   @GET
   @Path("{id}")
   public Article read(@PathParam("id") int id) {…}
   @GET
   public List<Article> list(@QueryParam("page")
                             @DefaultValue("1")
                             int page) {…}
}
```

Article POJO

```
public class Article {
  private Integer id;
  private String title;
  private String content;

  // Getters and setters go here
}
```

Let's try

```
$ curl -X GET http://localhost:8080/articles/7
{
  "id": 7,
  "title": "REST Individual Resource",
  "content": "REST Individual Resource example"
}
```

```
$ curl -X GET http://localhost:8080/articles/
[
  {
    "id": 7,
    "title": "REST Individual Resource",
    "content": "REST Individual Resource example"
  }
]
```

Link things together

HATEOAS

HATEOAS is abbreviation for **Hypermedia as the Engine of Application State**.

A hypermedia-driven site provides information to navigate the site's REST interfaces dynamically by including hypermedia links with the responses.

Read more Why hypermedia APIs? and Understanding HATEOAS

Atom Links Style

```
<article id="123">
  <title>Hypermedia and JAX-RS 2.0</title>
  <content>JAX-RS 2.0 supports Hypermedia …</content>

  <link rel="self"   href=“/articles/123/" />
  <link rel="update" href=“/articles/123/" />
  <link rel="delete" href=“/articles/123/" />
  <link rel="list"   href=“/articles/“     />

</article>
```

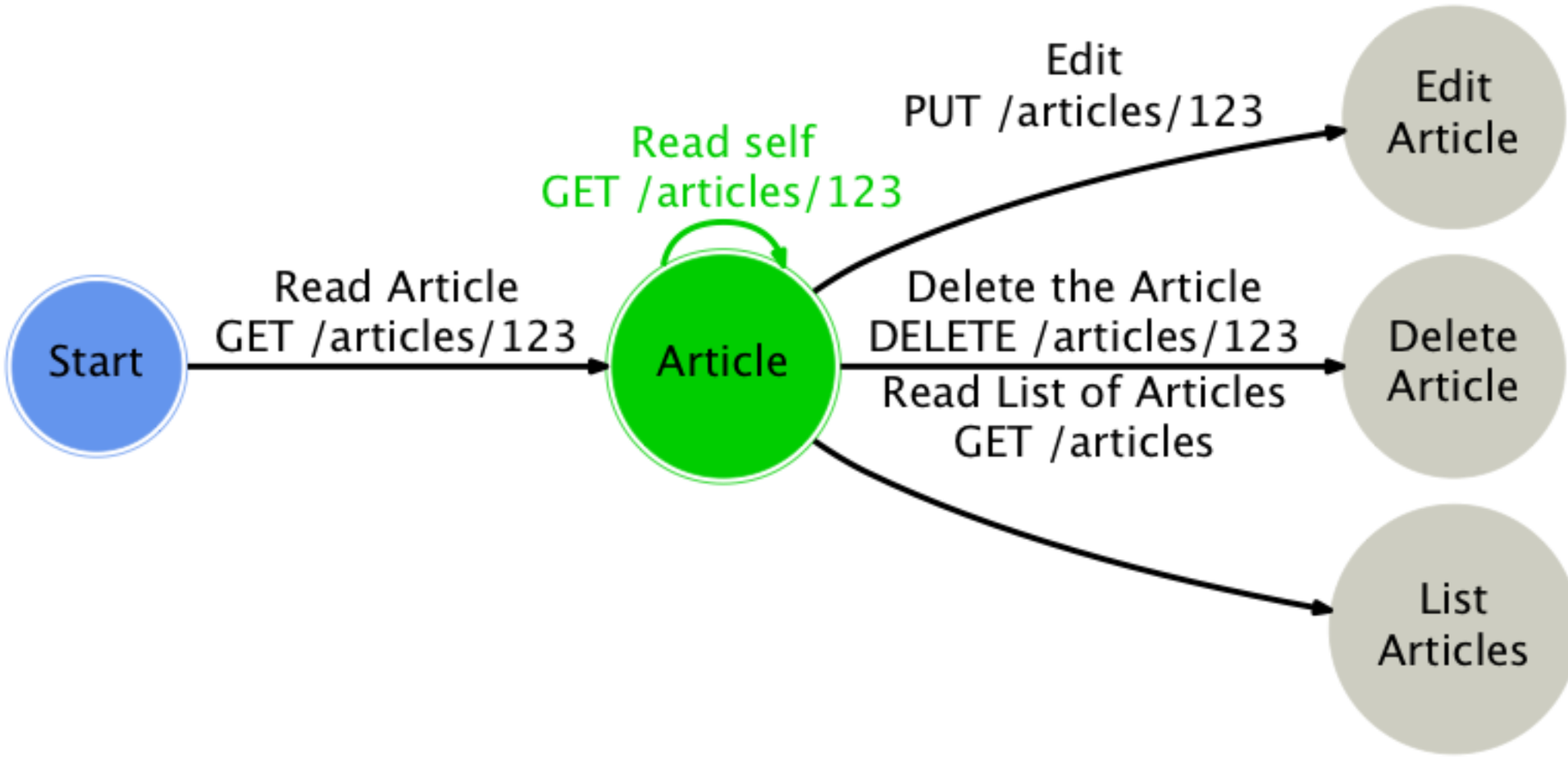
Read more [Atom Format Link Relation Extensions]

Link Headers Style

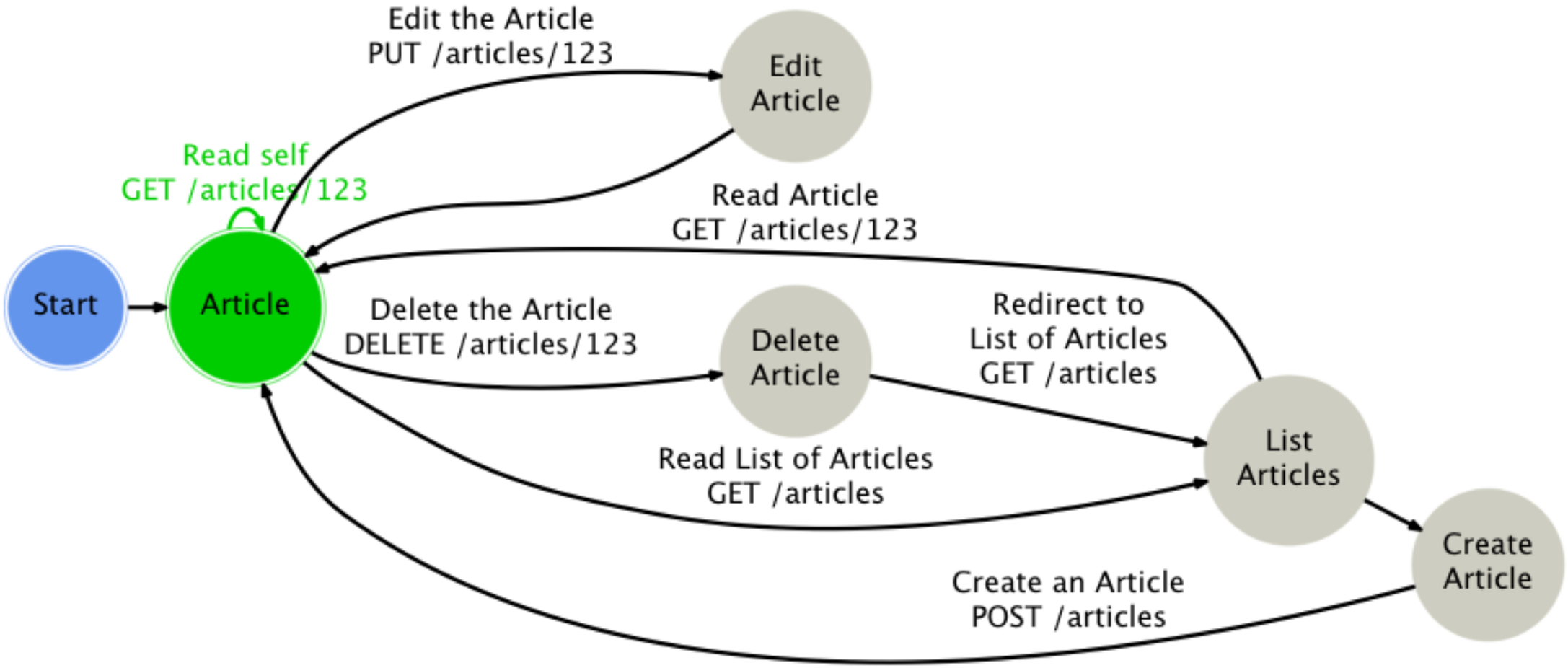
```
HTTP/1.1 200 OK
Content-Type: application/xml
Link: </articles/123/>; rel=self;
Link: </articles/123/>; rel=update;
Link: </articles/123/>; rel=delete;
Link: </articles/>;     rel=list;

<article id="123">
  <title>Hypermedia and JAX-RS 2.0</title>
  <content>JAX-RS 2.0 supports Hypermedia …</content>
</article>
```

Read more [Web Linking: The Link Header Field]

How to Imaging HATEOAS



How to Imaging HATEOAS



JAX-RS 2.0 HATEOAS

```
@POST
@Consumes({"application/json", "application/xml"})
@Produces({"application/json", "application/xml"})
public Response create(Article article) {
  Article created = articleDao.create(article);
  return Response
          .ok(created)
          .link("link-URI", “link-rel")
          .links(produceLinks(created))
          .build();
}

private Link[] produceLinks(Article article) {...}
```

Read more javax.ws.rs.core.Link

Jersey 2.x HATEOAS

```
public class Article {
  @InjectLinks({
    @InjectLink(resource = ArticleResource.class,
                rel = "self", method = "read"),
    @InjectLink(resource = ArticleResorce.class,
                rel = "edit", method = "update"),
    @InjectLink(resource = ArticleResource.class,
                rel = "delete", method = "delete"),
    @InjectLink(resource = ArticleResource.class,
                rel = "list", method = “list")
  })
  private List<Link> links;
```

Read more Jersey 2.x Chapter 12. Declarative Hyperlinking

Let's try

```
$ curl -X GET “http://localhost:8080/articles/123"
{
  "id": 123,
  "title": "Hypermedia and JAX-RS 2.0",
  "content": "JAX-RS 2.0 supports Hypermedia …",
  "links": [
    { "uri": "/articles/",
      "rel": "self" },
    { "uri": "/articles/",
      "rel": "edit" },
    { "uri": "/articles/",
      "rel": "delete" },
    { "uri": "/articles",
      "rel": “list"}]
}
```

Use standard HTTP methods

Method	Purpose
GET	Read representation of the specified resource
POST	Create or Update without a know ID
PUT	Update or Create with a know ID
DELETE	Remove
HEAD	GET with no response, just metadata
OPTIONS	Supported methods for the specified URL.

HTTP methods and JAX-RS 2.0

Method	Annotation
POST	@POST
GET	@GET
PUT	@PUT
DELETE	@DELETE
HEAD	@HEAD
OPTIONS	@OPTIONS

Multiple Representations

Content Negotiation

```
GET http://example.com/stuff
Accept: application/xml, application/json, text/*
```

Give me resource contents as (in priority):

1. XML
2. or JSON
3. or any text content type you can
4. or return **406 Not Acceptable Error** otherwise

Content Negotiation Example

```
curl -X POST "http://localhost:8080/articles"          \
     -H "Content-Type: application/xml"                \
     -d "<article>                                     \
           <title>Conneg and JAX-RS 2.0</title>        \
           <content>                                   \
             JAX-RS 2.0 supports Conneg…               \
           </content>                                  \
         </article>"                                   \
      -H "Accept: application/json"
{
  "id": 7,
  "title": "Conneg and JAX-RS 2.0",
  "content": "JAX-RS 2.0 supports Conneg …"
}
```

Content Negotiation Example

```
curl -X POST "http://localhost:8080/articles"          \
     -H "Content-Type: application/xml"                \
     -d "<article>                                     \
           <title>Conneg and JAX-RS 2.0</title>        \
           <content>                                   \
             JAX-RS 2.0 supports Conneg…               \
           </content>                                  \
         </article>"                                   \
     -H "Accept: application/json"
{
  "id": 7,
  "title": "Conneg and JAX-RS 2.0",
  "content": "JAX-RS 2.0 supports Conneg …"
}
```

Content Negotiation Example

```
@POST
@Consumes({"application/json", "application/xml"})
@Produces({"application/json", "application/xml"})
public Article create(Article article) {
  return articleDao.create(article);
}
```

Language Negotiation

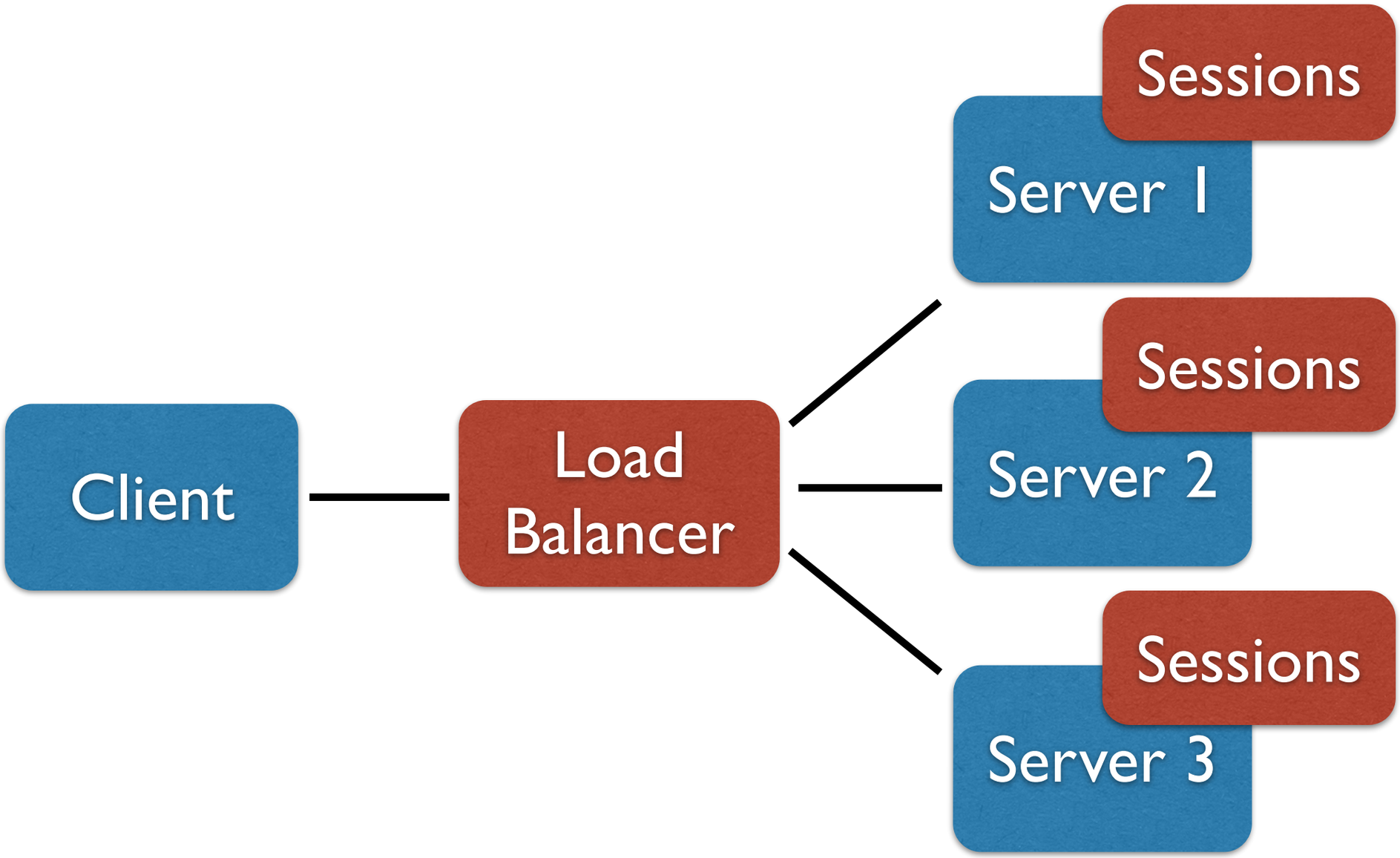
```
GET http://example.com/stuff
Accept-Language: en-us, en, ru
```

Encoding Negotiation

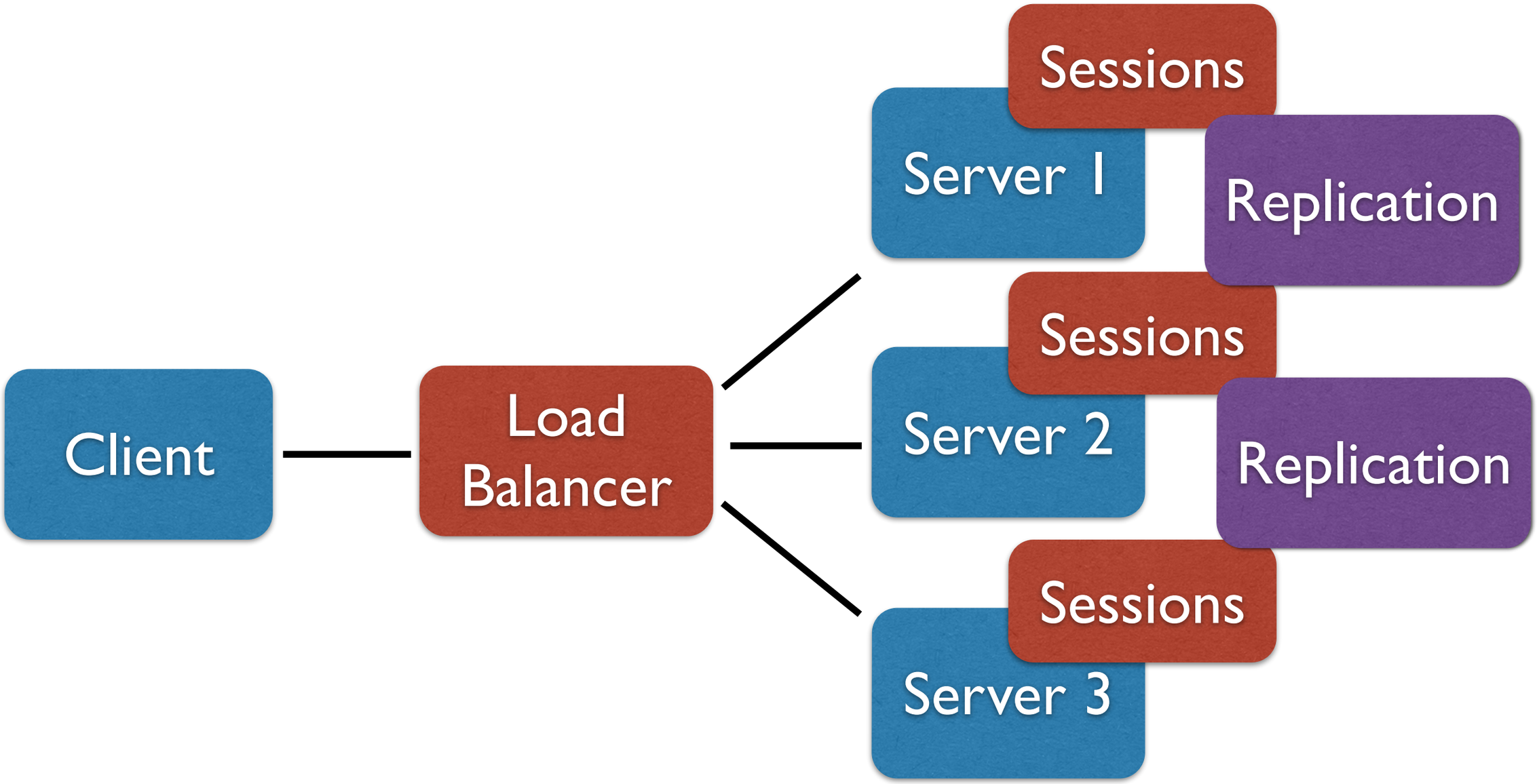
```
GET http://example.com/stuff
Accept-Encoding: gzip, deflate
```

Communicate Statelessly

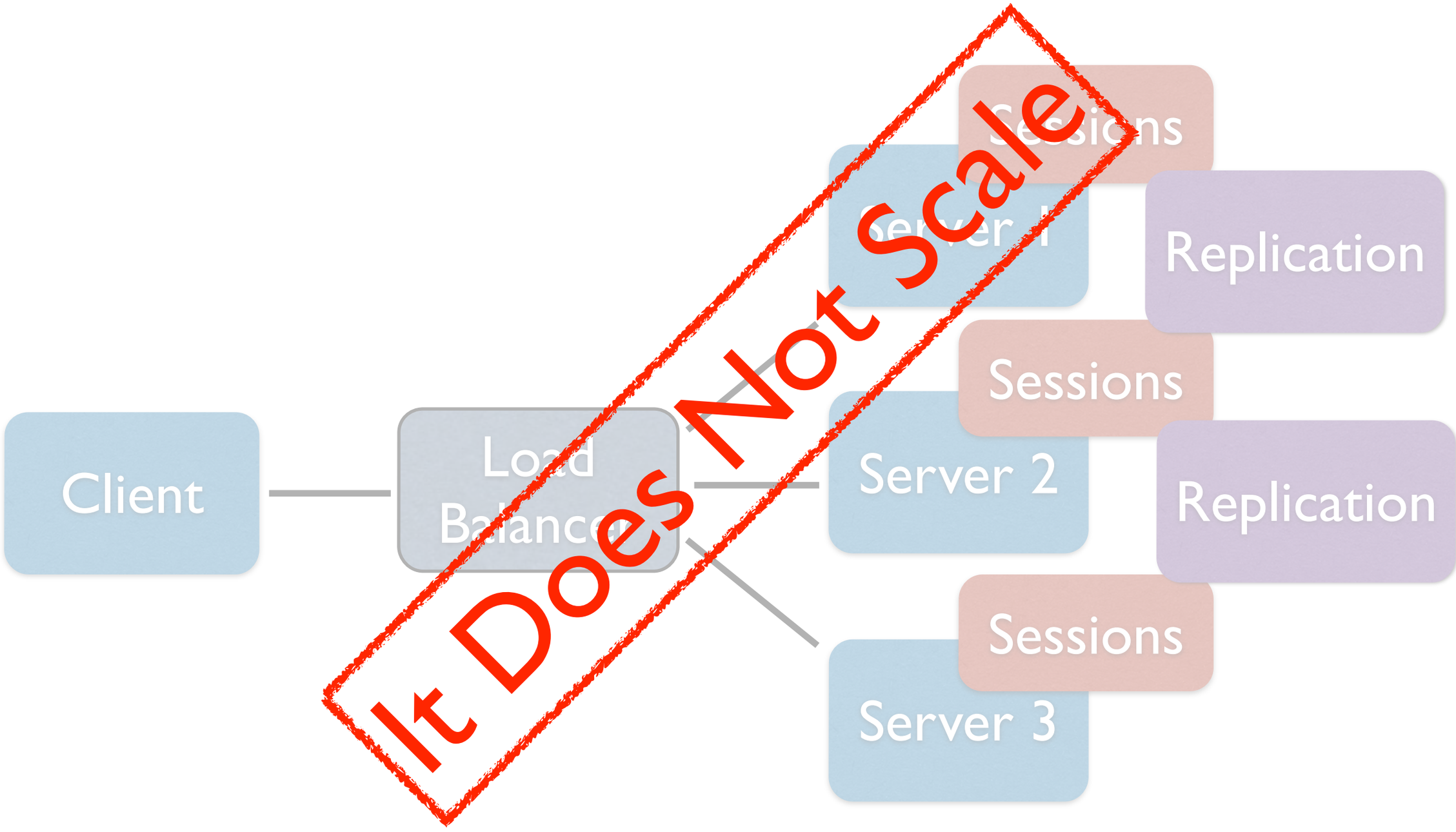
State is Bad


Sessions
Server 1
Sessions
Server 2
Client
Load Balancer
Sessions
Server 3

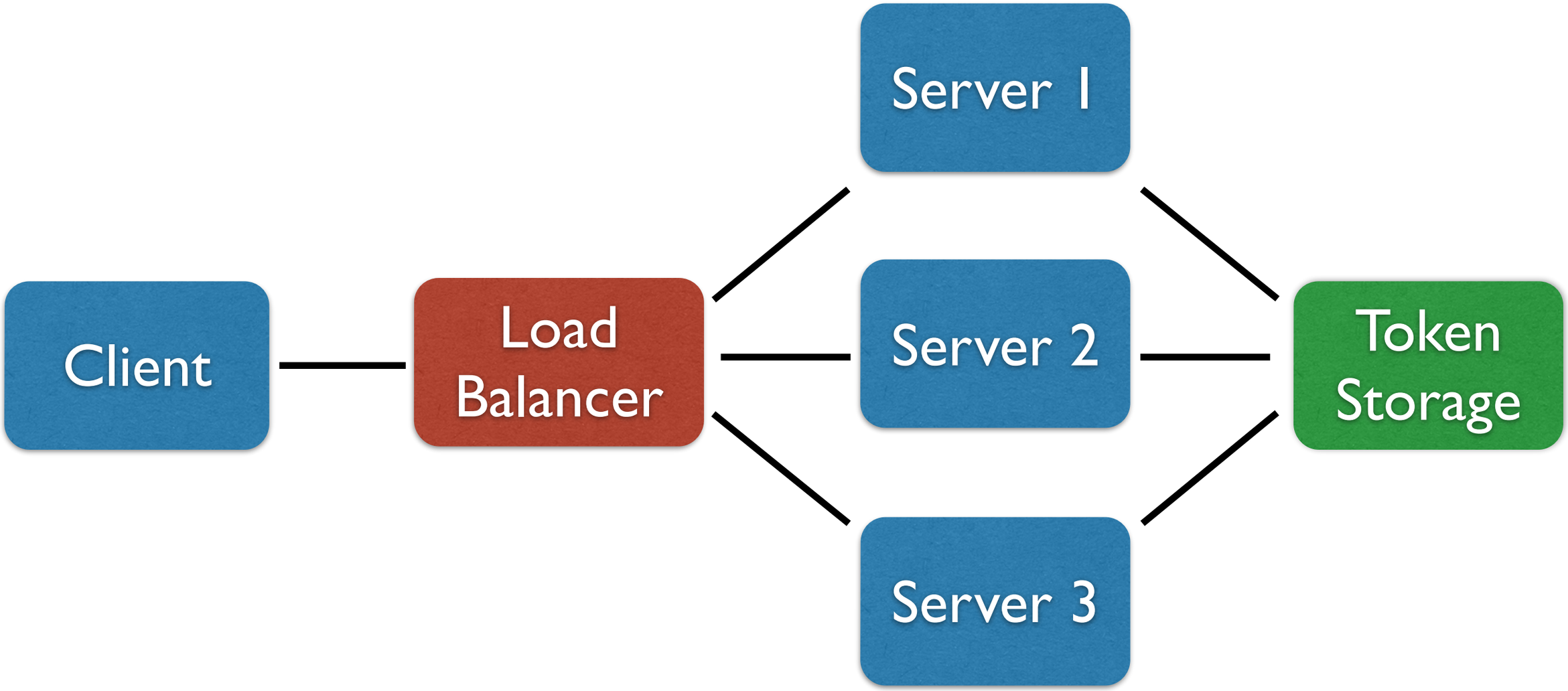
State is Bad


Client
Load Balancer
Server 1
Sessions
Replication
Server 2
Sessions
Replication
Server 3
Sessions

State is Bad


Client
Load Balancer
Server 1
Server 2
Server 3
Sessions
Sessions
Sessions
Replication
Replication
It Does Not Scale

Communicate statelessly



Caching Support

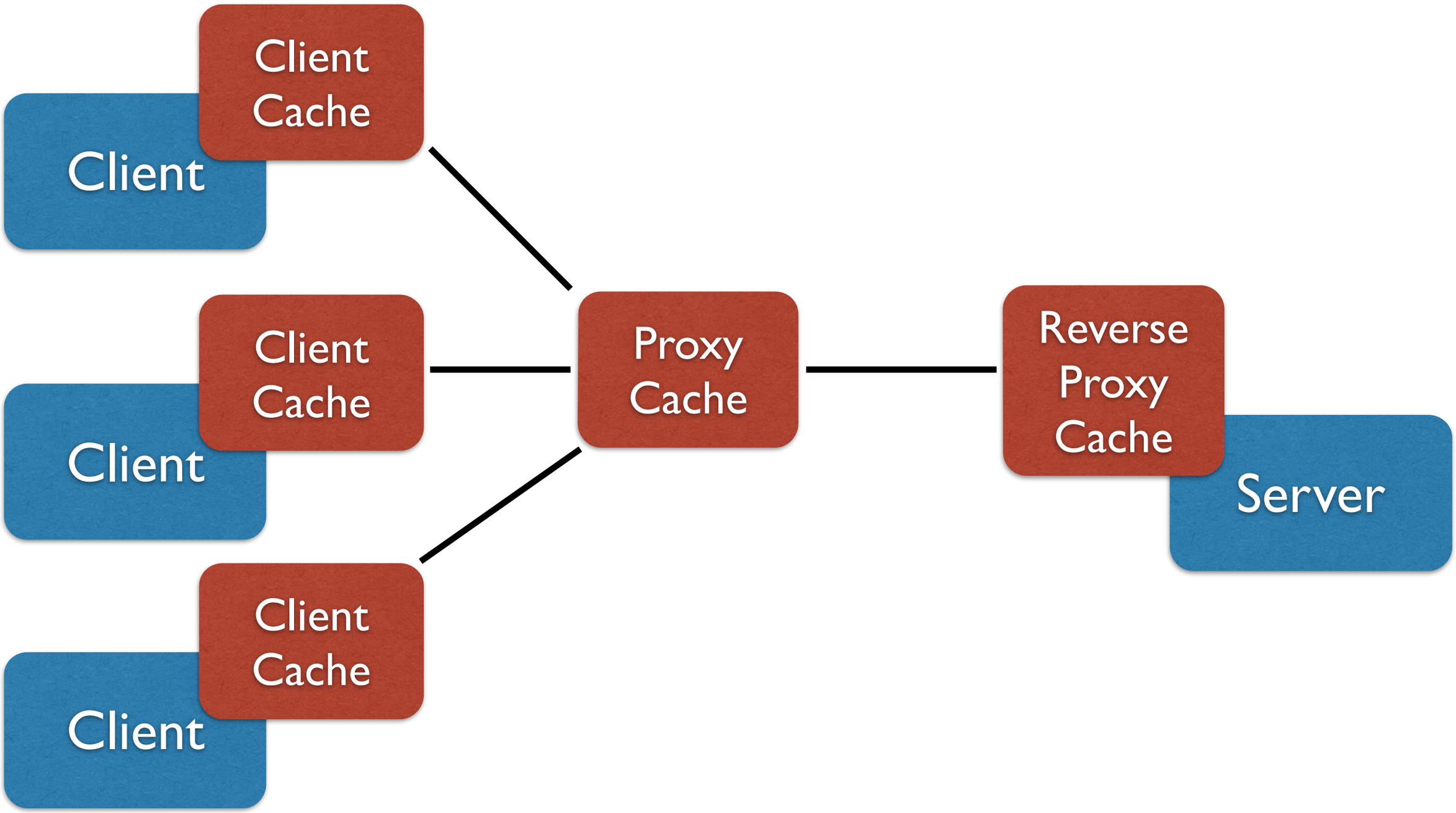
"*The best requests are those that not even reach me*"

- Anonymous overloaded Server

Caching Benefits

- Reduce bandwidth
- Reduce latency
- Reduce load on servers
- Hide network failures

Caching


Client
Client Cache
Client
Client Cache
Client
Client Cache
Proxy Cache
Reverse Proxy Cache
Server

HTTP Caching

```
GET http://example.com/stuff

< HTTP/1.1 200 OK
< Cache-Control: max-age=60
```

Cache-Control: max-age=<delta-seconds>
| s-max-age=<delta-seconds>
| no-cache
| no-store
| …

Read more Caching in HTTP

JAX-RS 2.0 Cache Control

```
CacheControl cache = new CacheControl();
```

Directive	Usage
max-age	cache.setMaxAge(int maxAge)
s-max-age	cache.setSMaxAge(int maxAge)
no-cache	cache.setNoCache(boolean noCache)
no-store	cache.setNoStore(boolean noStore)
…	…

Read more javax.ws.rs.core.CacheControl

Caching and JAX-RS 2.0

```
@GET
@Path("{id}")
public Response read(@PathParam("id") int id) {
  Article article = articleDao.findById(id);
  CacheControl cacheControl = new CacheControl();
  cacheControl.setMaxAge(60);

  return Response.ok(article)
                 .cacheControl(cacheControl)
                 .build();
}
```

Let's try

```
curl -X GET "http://localhost:8081/rest/articles/8"
     -H "Accept: application/json" -v
< HTTP/1.1 200 OK
< Cache-Control: max-age=60
< Content-Type: application/json
<
{
  "id": 8,
  "title": "HTTP Caching and JAX-RS 2.0",
  "content": "JAX-RS 2.0 supports HTTP Caching"
}
```

Wrapping Up

JAX-RS 2.0 is a POJO-based HTTP-centric annotation-driven specification for for RESTful Web Services.

Makes the developer focus on URLs, HTTP methods and Media Types.

Implementations:

- Apache CXF
- Jersey
- RESTeasy
- Restlet
- others

Q & A

Thank you!

Яндекс

Дмитрий Чижиков

Старший Разработчик
Киноплатформы

ffbit

dmytro.chyzhykov@yandex.ru