Implementing an API in ASP.NET Web API

Module 5: REST Constraints

Shawn Wildermuth
Wilder Minds LLC
http://wilderminds.com

pluralsight
hardcore developer training

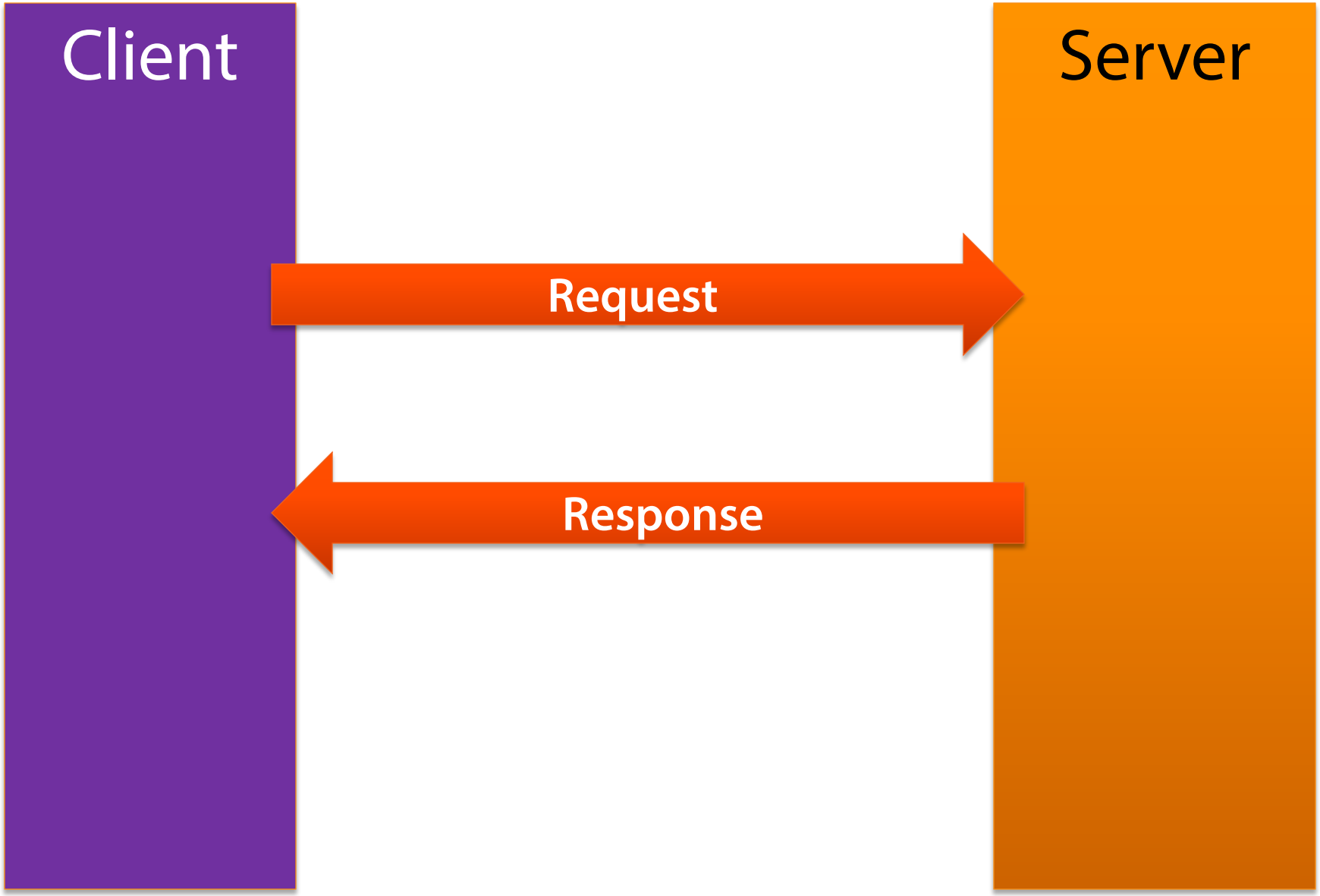
Agenda

- **REST Constraints**
  - What are the Constraints of REST?
  - Client-Server
  - Stateless Server
  - Cache
  - Uniform Interface
  - Layered System
  - Code On-Demand

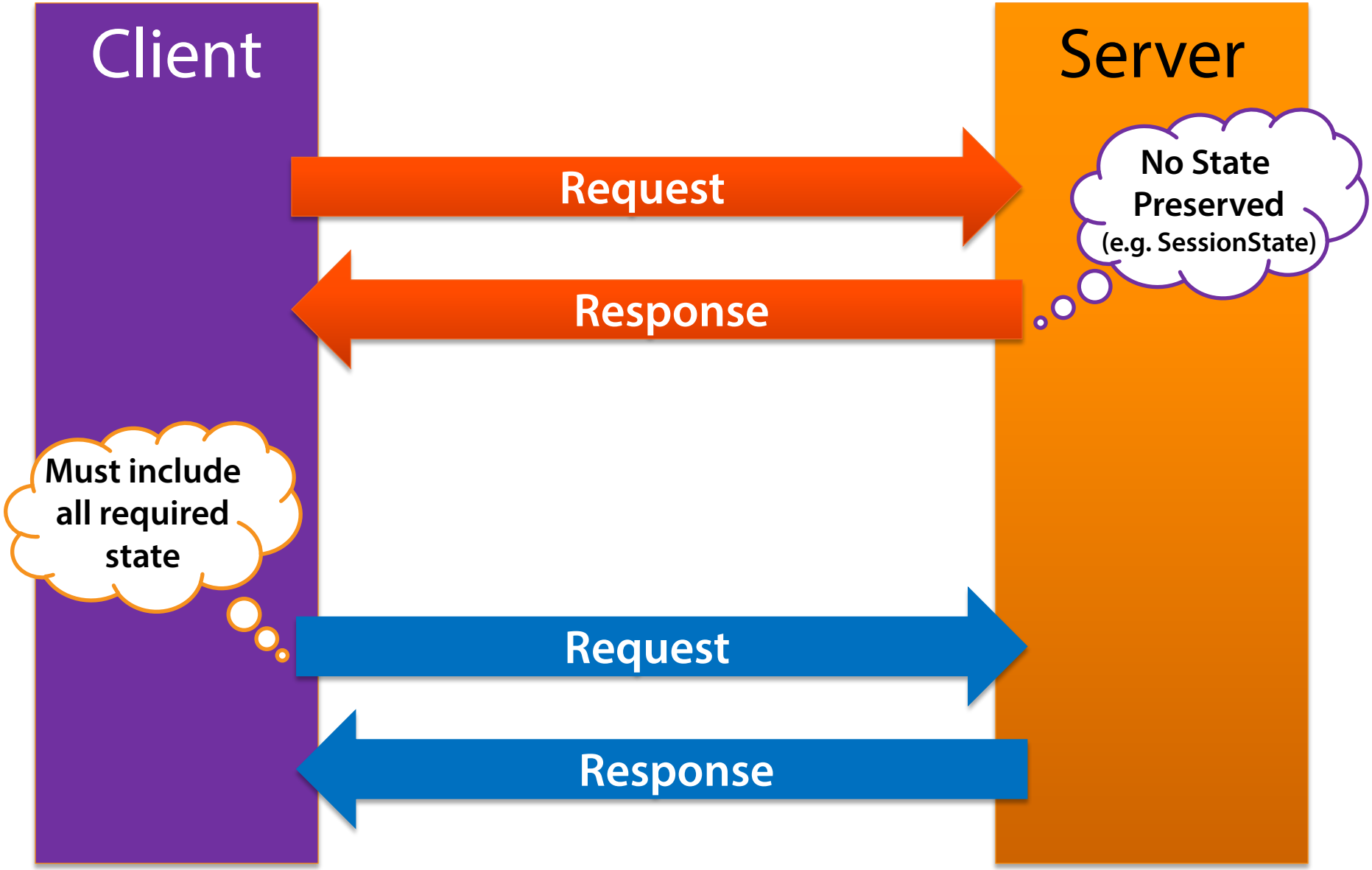
What are the Constraints of REST?

1. **Client-Server**
2. **Stateless Server**
3. **Cache**
4. **Uniform Interface**
    a. Identification of Resources
    b. Representations Supporting Modification
    c. Self-Description
    d. Hypermedia As The Engine Of Application State (HATEOAS)
5. **Layered System**
6. ***Code-On-Demand***

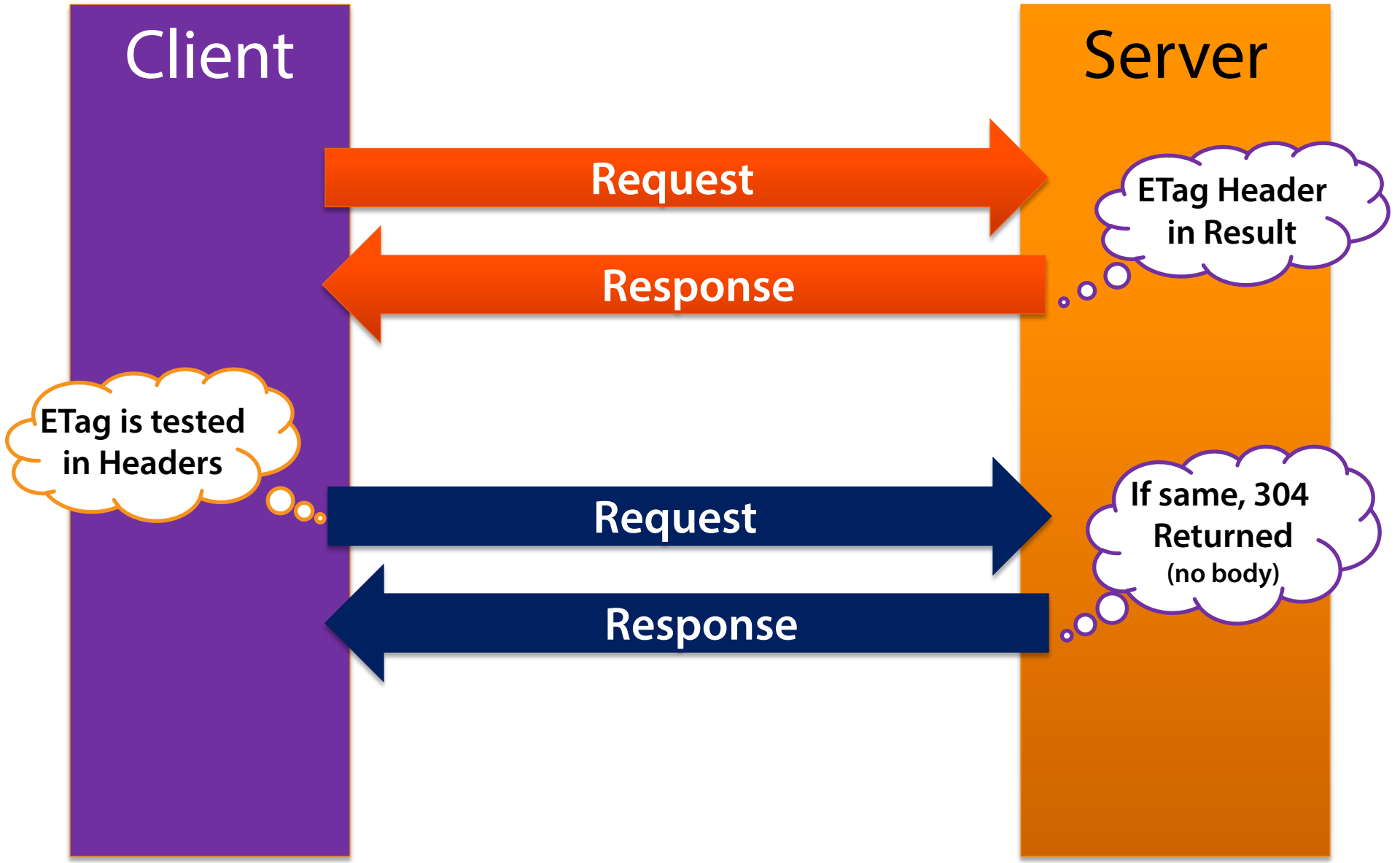
Client-Server



Stateless Server



Cache


Client
Server
Request
ETag Header in Result
Response
ETag is tested in Headers
Request
If same, 304 Returned (no body)
Response

What are ETags?

- **Header used as a Unique Key for Resource**

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Date: Thu, 23 May 2013 21:52:14 GMT
ETag: W/"4893023942098"
Content-Length: 639
```

- **Requests Should Test using If-None-Match:**

```
GET /api/nutrition/foods/2 HTTP/1.1
Accept: application/json, text/xml
Host: localhost:8863
If-None-Match: "4893023942098"
```

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

What are ETags?

- **For PUT/PATCH it is different**

```
PATCH /api/user/diaries/2013-5-12 HTTP/1.1
Accept: application/json, text/xml
Host: localhost:8863
If-Match: "4893023942098"
...
```

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

Implementing ETags

Uniform Interface

- **Defined in four parts:**
  - Identification of resources

```
http://.../api/nutrition/foods/12345
```

```
http://.../api/user/diaries/2013-5-24
```

```
http://.../api/user/diaries/2013-5-24/entries/12
```

Uniform Interface

- **Defined in four parts:**
  - Manipulation of resources through representations
    - Same Structures can be sent back as are received through the API
    - Using the standard HTTP verbs to represent the operation

Uniform Interface

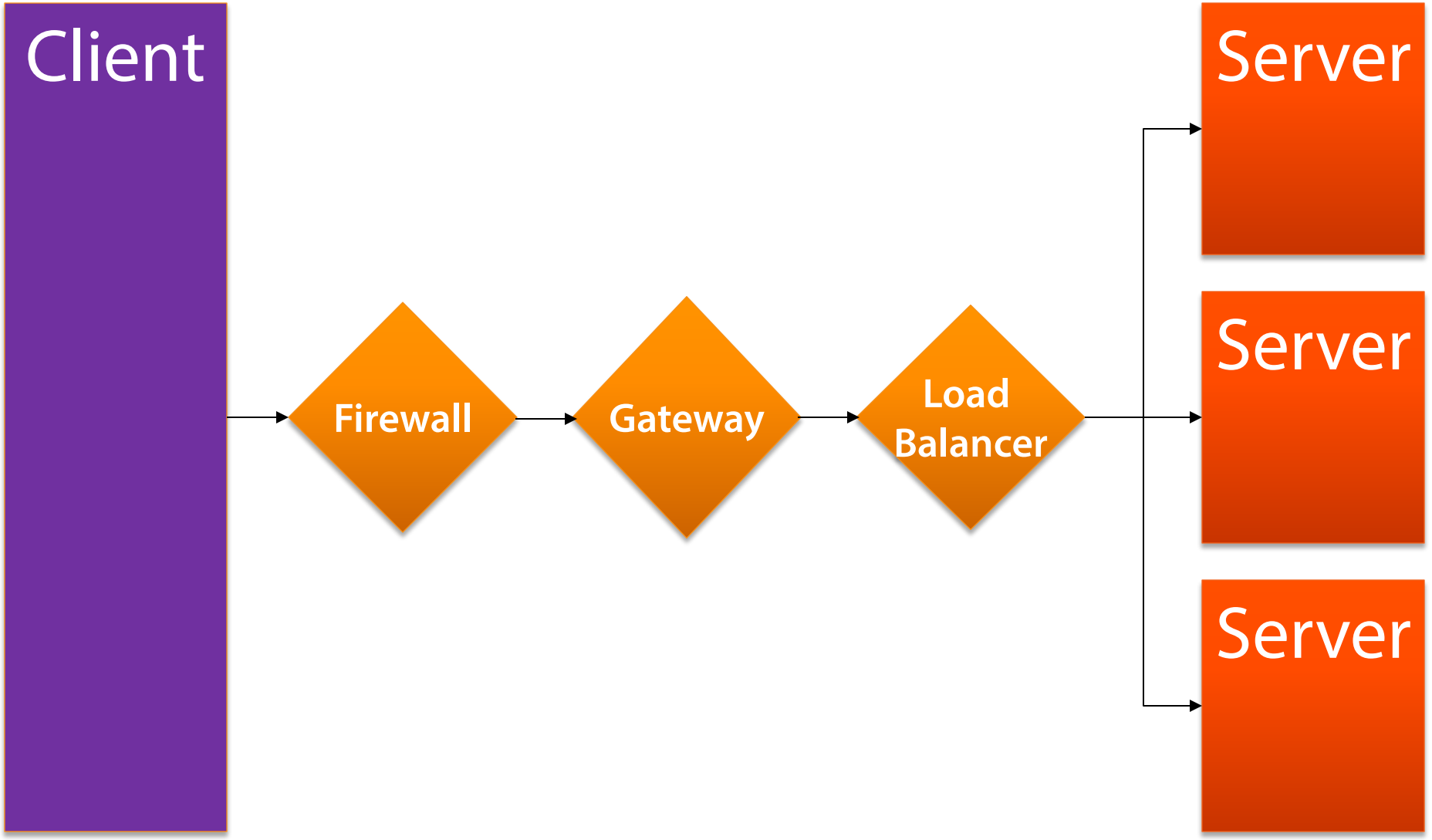
- **Defined in four parts:**
  - Self-descriptive messages
  - Hypermedia as the engine of application state
    - Represented by links in the results

Links


```
{
  "links":[
    {
      "href":"http://.../api/nutrition/foods/4479",
      "rel":"self"
    },
    {
      "href":"http://.../api/nutrition/foods/4479/measures",
      "rel":"getmeasures"
    }
  ],
  "description":"Abalone, Mixed Species, Raw",
  "measures":[
    {
      "url":"http://.../api/nutrition/foods/4479/measures/7269",
      "description":"3 Oz",
      "calories":89.2
    }
  ]
}
```

Implementing Links

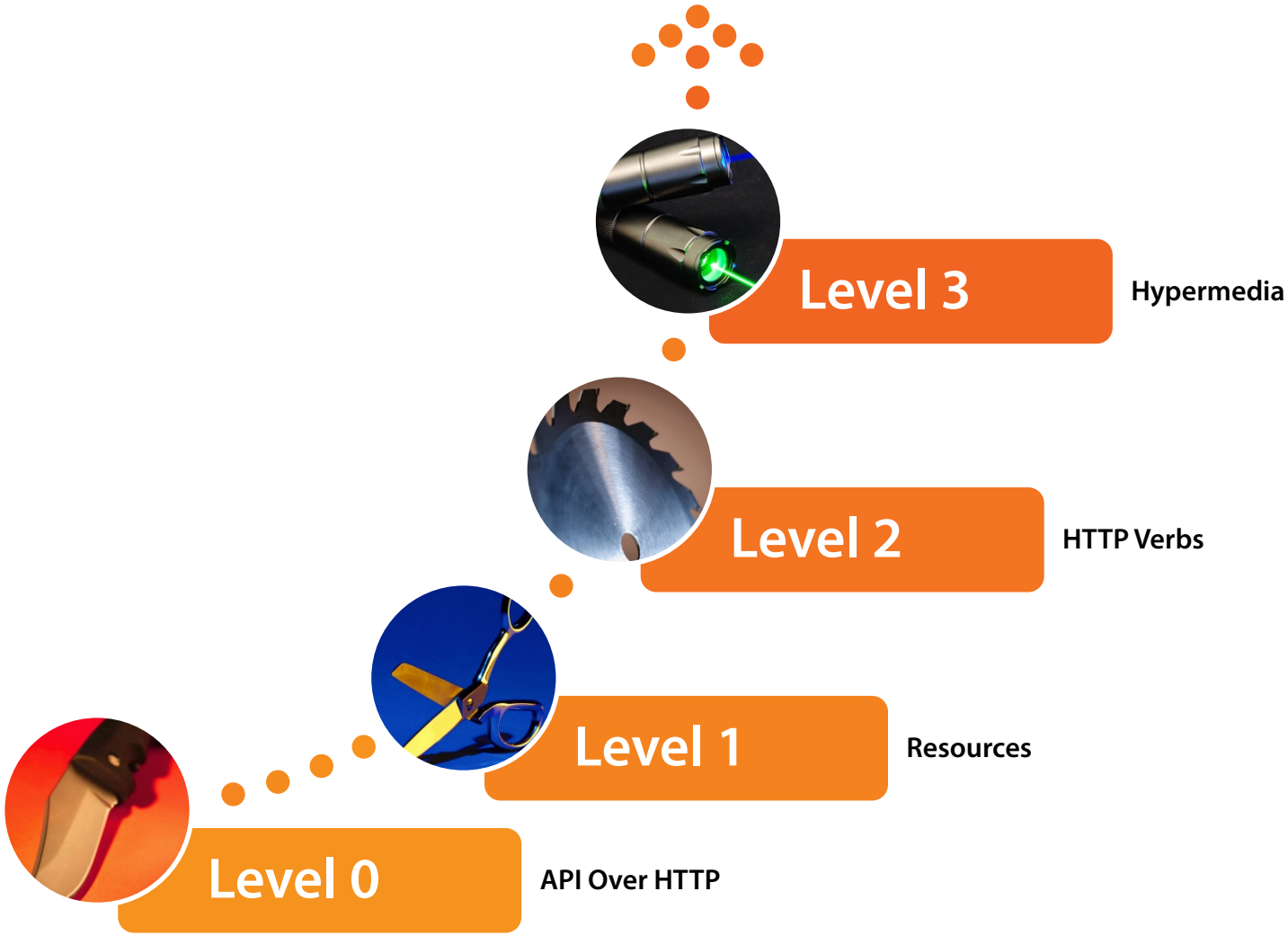
Layered System



Code On-Demand

- **Mentioned as Optional by Original REST Paper**
  - The ability to deliver code from the server for the client to execute
  - Very rarely adopted
  - Possible in the JavaScript world, but rarely would clients trust the server

A REST Maturity Model*



* Leonard Richardson's Model of REST 'Completeness'
http://shawnw.me/restrmm

Pragmatism

- **Richardson Maturity Model is used as a Cudgel**
  - Build "Enough" maturity for your API
  - Over-Building is worse than Under-Building
  - Remember your users
  - Don't get caught up in the Dogma Trap

Summary

- **REST Constraints**
  - Understanding the REST Constraints can help you design a great API
  - Building a scalable, stateless server is crucial
  - You should supporting Caching using ETags wherever possible
  - Building self-describing messages using Links where appropriate
  - Ultimately, being pragmatic about what level of maturity build is best