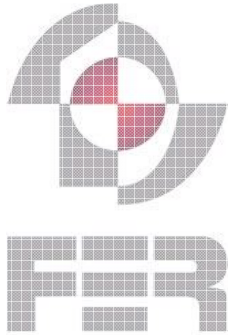
UNIZG-FER 86518

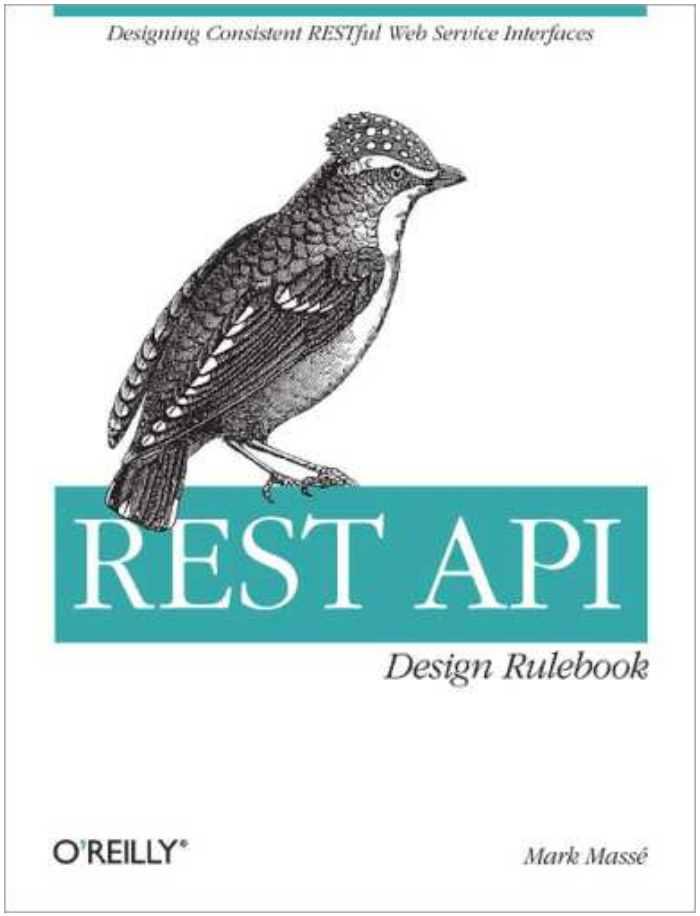
Service-Oriented Computing



RESTful API Design

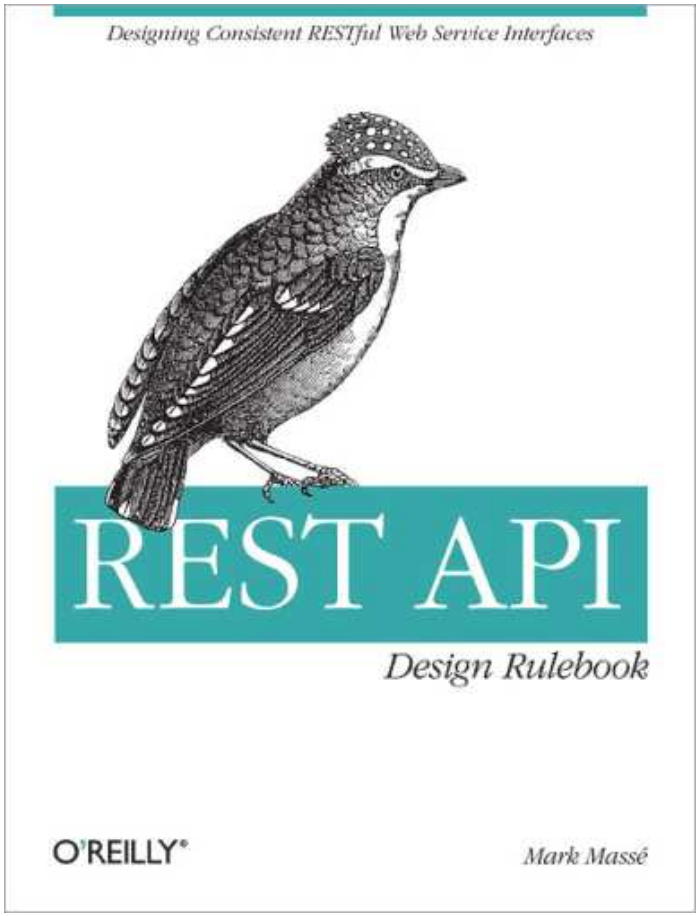
RESTful API Design

- Resource & URI design
- Interaction design
  - Request methods
  - Response codes
- Metadata design
- Representation design
- Advanced feature design
  - Versioning
  - Security
  - Response representation composition



RESTful API Design

- Resource & URI design
- Interaction design
  - Request methods
  - Response codes
- Metadata design
- Representation design
- Advanced feature design
  - Versioning
  - Security
  - Response representation composition



General Rules for URI Design

Lowercase letters should be preferred in URI paths

- RFC 3986 defines URIs as case-sensitive except for the scheme and host components
- Differentiation between URIs should not rely on letter capitalization, but on spelling

`http://api.example.com/my-folder/my-doc` ✓

`HTTP://API.EXAMPLE.COM/my-folder/my-doc` ⊘ } =

`http://api.example.com/My-Folder/my-doc` ⊘ } ≠

General Rules for URI Design

Hyphens (-) should be used to improve the readability of URIs

- Spaces makes long strings readable, but are not allowed in URIs
- Anywhere you would use a space in spoken language, you should use a hyphen in a URI

`http://api.example.com/blog/entries/this-is-my-first-post` ✓

`http://api.example.com/blog/entries/thisismyfirstpost` ⊘

`http://api.example.com/blog/entries/thisIsMyFirstPost` ⊘

`http://api.example.com/blog/entries/this_is_my_first_post` ⊘

General Rules for URI Design

Underscores (_) should not be used in URIs

- Client applications (browsers, editors, etc.) often underline URIs to provide a visual cue that they are clickable
  - The underscore (_) character can either get partially obscured or completely hidden by this underlining
- To avoid this confusion, use hyphens (-) instead of underscores (_)

`http://api.example.com/my-folder/my-doc` 

`http://api.example.com/my_folder/my_doc` 

General Rules for URI Design

File extensions should not be included in URIs

- File extension in URI indicates the format of a message body
- The format of a message body should rely on the media type communicated through the `Content-Type` header

http://api.college.com/students/32482/transcripts/2005/fall.json 

http://api.college.com/students/32482/transcripts/2005/fall.xml 

http://api.college.com/students/32482/transcripts/2005/fall 

URI Authority Design

Consistent subdomain names should be used for your APIs

- The top-level domain and first subdomain names should identify the service owner
- The full domain name of an API should add a subdomain named `api`

`http://www.example.com`

Service owner

Website for human consumption (prefixed with `www`)

`http://api.example.com`

Service owner

REST API for programmable web (prefixed with `api`)

URI Authority Design

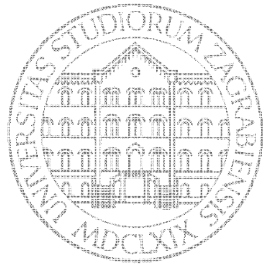
Consistent subdomain names should be used for your developer portal

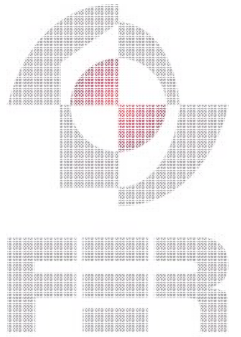
- Many REST APIs have an associated website, known as a *developer portal*
  - API documentation
  - API access keys provisioning
  - Forums, FAQs, download zones, etc.

- If an API provides a developer portal, by convention it should have a subdomain labeled `developer`

```
http://developer.example.com
```

Resource Modeling





- Resource archetypes
  - Document
  - Collection
  - Store
  - Controller

REST API example

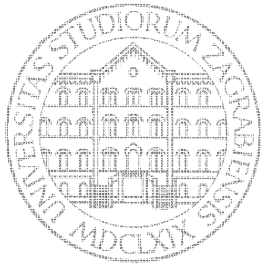
```
http://api.soccer.com
```

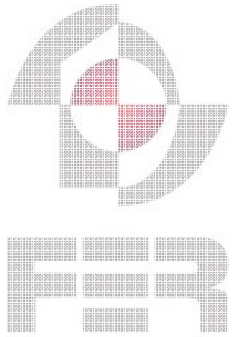
root resource (*docroot*)

```
http://api.soccer.com/leagues
http://api.soccer.com/leagues/seattle
http://api.soccer.com/leagues/seattle/teams
http://api.soccer.com/leagues/seattle/teams/sonic
http://api.soccer.com/leagues/seattle/teams/sonic/players
http://api.soccer.com/leagues/seattle/teams/sonic/players/mike
```

subordinate resources

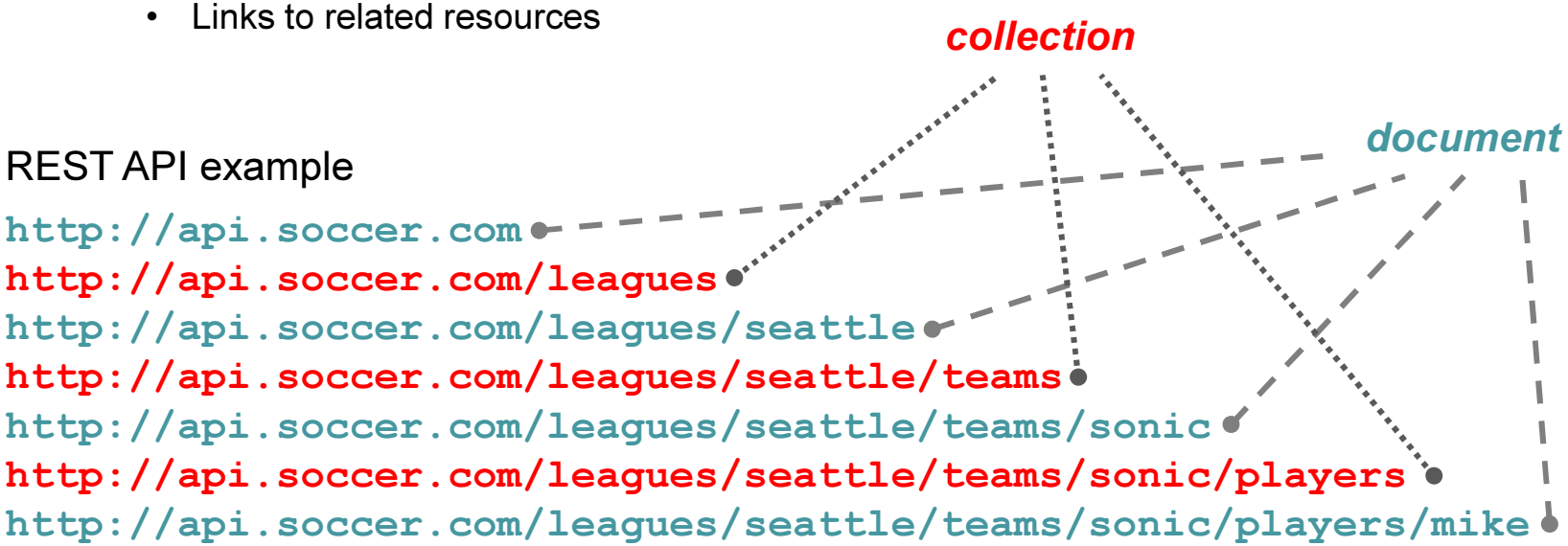
Resource Modeling



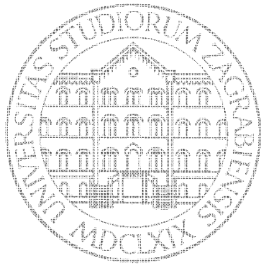


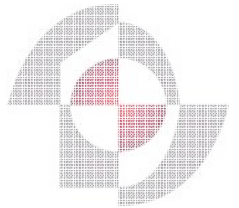
- Document
  - Base archetype of the other resource archetypes
    - The three other resource archetypes can be viewed as specializations of the document archetype
  - Resource are usually classified as document if
    - represents individual resource
    - does not fit to any of the other resource archetypes
  - Document contains
    - Resource representation
    - Links to related resources

REST API example



Resource Modeling





- Collection
  - Server-managed directory of resources
  - Server may add new resources into collection on their own
  - Clients may propose new resources to be added to a collection, but it is up to the collection to choose to create a new resource or not
  - A collection decides what it wants to contain and also generates the URI of each contained resource

Collection URI

`http://api.news.com/articles/croatia-in-eu/comments`

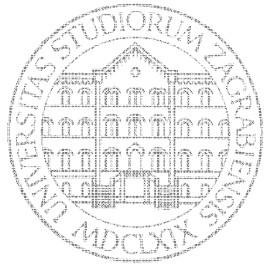
Client request

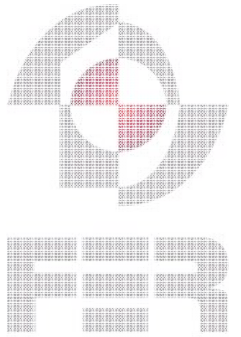
```
POST /articles/croatia-in-eu/comments
Host: api.news.com

comment text in request body
```

New server-created resource with ***server-autogenerated URI***

`http://api.news.com/articles/croatia-in-eu/comments/89`

Resource Modeling





- Store
  - Client-managed resource repository
  - Server never adds new resources into store, but lets clients to put resources in, get them back out, and delete them
  - Store do not generates new URIs, it is a client responsibility
  - Each stored resource has a URI that was chosen by a client when it was initially put into the store

Store URI

`http://api.encyclopedia.com/articles`

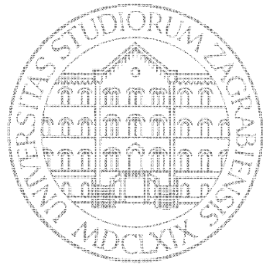
Client request

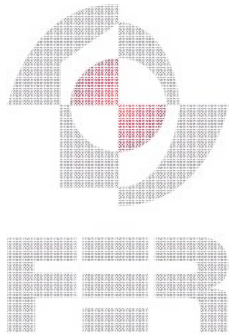
```
PUT /articles/croatia-in-eu
Host: api.encyclopedia.com

article content in request body
```

New server-created resource with ***client-defined URI***

`http://api.encyclopedia.com/articles/croatia-in-eu`

Resource Modeling





- Controller
  - Algorithmic resources
  - Expose executable functions with parameters and return values
  - Used to perform application-specific actions that cannot be logically mapped to any of the standard CRUD methods (*create*, *read*, *update*, *delete*)
  - Clients use query variables to supply input arguments
  - Server returns output value as resource representation after performing requested operation

Controller URI examples

`http://www.google.com/search`

`http://api.communicator.com/sms/send`

Client request examples

```
GET /search?q=zagreb
Host: www.google.com
```

```
POST /sms/send
Host: api.communicator.com

message text in request body
```

URI Path Design

Forward slash separator (/) must be used to indicate a hierarchical relationship

- Use path variables to encode hierarchy between resources using pattern `/parent/child`

```
http://api.canvas.com/shapes/polygons/quadrilaterals/rectangle

http://api.canvas.com/shapes/polygons/quadrilaterals/square

http://api.canvas.com/shapes/ovals/circle


http://api.maps.com/france/paris

http://api.maps.com/croatia/zagreb/trnje/ulica-grada-vukovara
```

URI Path Design

No hierarchy between resources? Use commas (,) or semicolons (;)

- Use commas (,) when the order of the scoping information is important

`http://api.maps.com/coordinates/45.8,16` (Zagreb, Croatia)

≠

`http://api.maps.com/coordinates/16,45.8` (Yemen, Asia)

- Use semicolons (;) when the order does not matter

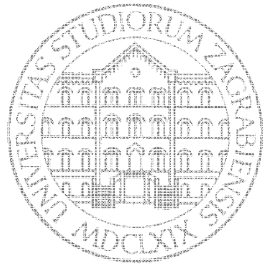
`http://api.maps.com/color-blends/red;blue`

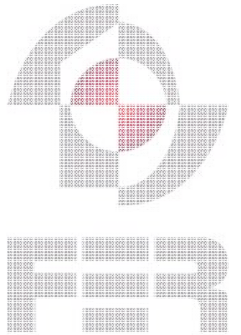
=

`http://api.maps.com/color-blends/blue;red` (purple)

URI Path Design

A trailing forward slash (/) should not be included in URIs

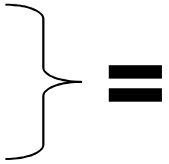




- As the last character within a URI's path, a forward slash (/) adds no semantic value

`http://api.canvas.restapi.org/shapes` 

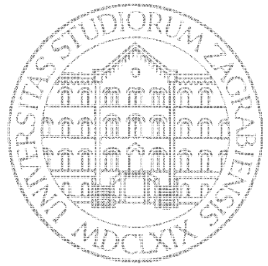
`http://api.canvas.restapi.org/shapes/` 

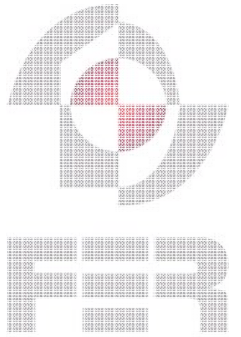


- If the URIs differ, then so do the resources, and vice versa
  - Requests to URIs with a trailing forward slash (/) are redirected to URIs without a trailing forward slash using response code `301 Moved Permanently`

URI Path Design

A singular noun should be used for document names





- For document, use a singular noun or singular noun phrase as its path segment

```
http://api.soccer.com/leagues/seattle
http://api.soccer.com/leagues/seattle/teams/sonic
http://api.soccer.com/leagues/seattle/teams/sonic/players/mike
```

```
http://api.canvas.com/shapes/polygons/quadrilaterals/rectangle
http://api.canvas.com/shapes/polygons/quadrilaterals/square
http://api.canvas.com/shapes/ovals/circle
```

URI Path Design

A plural noun should be used for collection names

- For collection of resources, use a plural noun or plural noun phrase as its path segment

```
http://api.soccer.com/leagues
http://api.soccer.com/leagues/seattle/teams
http://api.soccer.com/leagues/seattle/teams/sonic/players
```

```
http://api.canvas.com/shapes
http://api.canvas.com/shapes/polygons/quadrilaterals
http://api.canvas.com/shapes/ovals
```

URI Path Design

A plural noun should be used for store names

- For store of resources, use a plural noun or plural noun phrase as its path segment

```
http://api.encyclopedia.com/articles

http://api.music.com/users/8463759/playlists

http://api.maps.com/users/8463759/custom-maps
```

URI Path Design

A verb should be used for controller names

- For controller, use a verb or verb phrase as its path segment
  - Like a computer program's function, a URI identifying a controller resource should be named to indicate its action

```
http://www.google.com/search

http://api.communicator.com/sms/send

http://api.fer.unizg.hr/courses/soc/register

http://api.fer.unizg.hr/courses/soc/students/36489/login
```

URI Path Design

Variable path segments may be substituted with identity-based values

- Some URI path segments are *static*
  - Have fixed names chosen by REST API designer
- Other URI path segments are *variable*
  - They are automatically filled in with some unique identifiers (e.g. from database)

- *URI Template* syntax

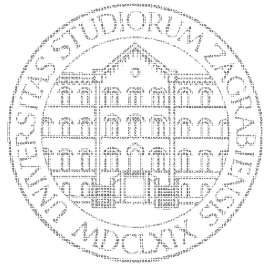
http://api.soccer.com/leagues/{*league-id*}/teams/{*team-id*}/players/{*player-id*}

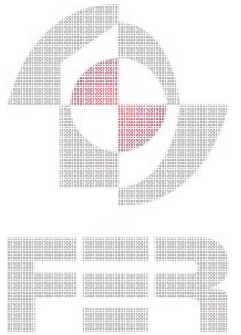
framework-generated IDs
(*names*, *custom numeric IDs*, *UUID*)

http://api.soccer.com/leagues/seattle/teams/sonic/players/mike
http://api.soccer.com/leagues/seattle/teams/sonic/players/john
http://api.soccer.com/leagues/atlanta/teams/hawk/players/tom

URI Path Design

CRUD function names should not be used in URIs





- URIs should be used to uniquely identify resources, not operations over them
- HTTP request methods should be used to indicate which CRUD function is performed over the resource

```
DELETE /users/1234
```

```
GET /deleteUser/1234

GET /deleteUser?id=1234

POST /users/1234/delete

DELETE /deleteUser/1234
```

URI Query Design

The query component of a URI ***may*** be used to provide input arguments to controllers

Controller resource that sends an SMS message
`http://api.communicator.com/sms/send`

Controller resource that sends an SMS message with text `hello`
`http://api.communicator.com/sms/send?text=hello`

Controller resource that sends an SMS message with text `hello world`
`http://api.communicator.com/sms/send?text=hello%20world`

3 different resources (although related)

- As a component of a URI, the query contributes to the unique identification of a resource
- Resource = Controller + input arguments
- Different input arguments ⇒ different resources

URI Query Design

The query component of a URI ***should*** be used to filter collections or stores

- URI's query component is a natural fit for supplying search criteria to a collection or store

```
GET /users
```

The response message's state representation contains a listing of all the users in the collection

```
GET /users?role=admin
```

The response message's state representation contains a filtered list of only those users in the collection with a "role" value of `admin`

Search through URI query makes search results cacheable

URI Query Design

The query component of a URI ***should*** be used to paginate collection or store results

- `pageSize` parameter specifies the maximum number of elements to return in the response
- `pageStartIndex` parameter specifies the zero-based index of the first element to return in the response

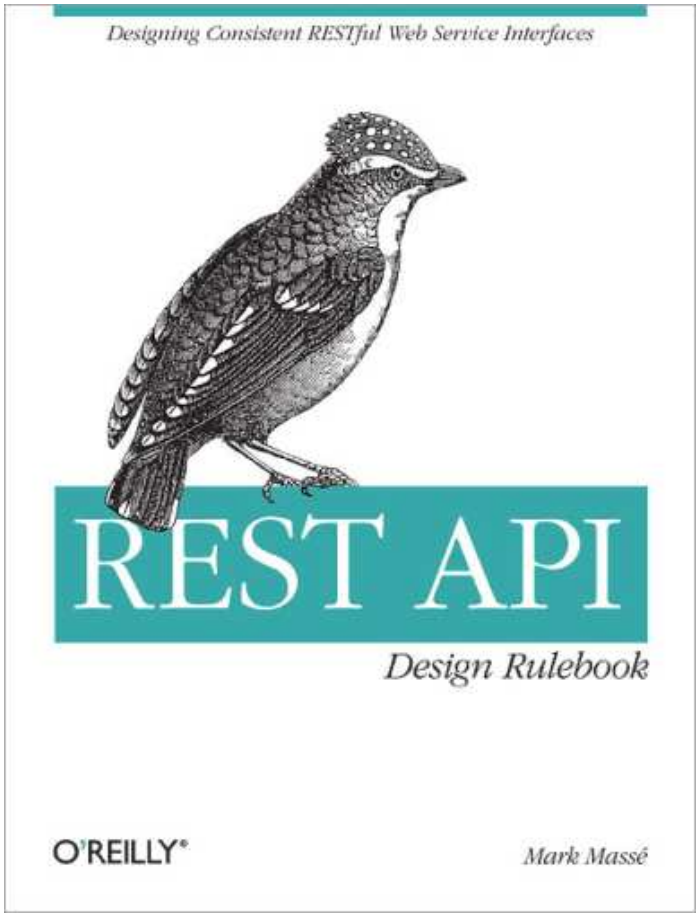
```
GET /users?pageSize=25&pageStartIndex=50
```

The response message's state representation contains a listing of 25 users ranging from position 50 to 74

Pagination through URI query makes pagination results cacheable

RESTful API Design

- Resource & URI design
- Interaction design
  - Request methods
  - Response codes
- Metadata design
- Representation design
- Advanced feature design
  - Versioning
  - Security
  - Response representation composition



Interaction Design

HTTP methods should be used in accordance with their defined semantics within the HTTP protocol

Method	Purpose
`GET`	To retrieve a representation of a resource state
`HEAD`	To retrieve the metadata associated with the resource state
`PUT`	a) To add a new resource to a store b) To update a resource
`POST`	a) To create a new resource within a collection b) To execute controllers
`DELETE`	To remove a resource from its parent
`OPTIONS`	To retrieve metadata that describes a resource's available interactions

Interaction Design

GET and POST must not be used to tunnel other request methods

- *Tunneling* refers to any abuse of HTTP that masks or misrepresents a message's intent and undermines the protocol's transparency

- REST API must not compromise its design by misusing HTTP request methods in an effort to accommodate clients with limited HTTP vocabulary

```
GET /deleteUser/1234
GET /updateAccount/789?value=100
POST /getOptions?id=1234
```

```
DELETE /users/1234
PUT /accounts/789?value=100
OPTIONS /users/1234
```

Interaction Design

GET and POST must not be used to tunnel other request methods (cont'd)

- `GET-POST` misuse
  - `GET` used to retrieve a representation of a resource state
  - `POST` used to execute controllers **Absolutelly always?**

```
GET /users/search?role=admin
GET /accounts/sort?by=ammount
```

```
POST users/1234/send-sms?text=hello
POST timer/start
```

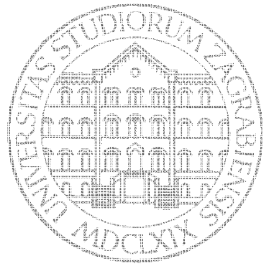
execution of controller

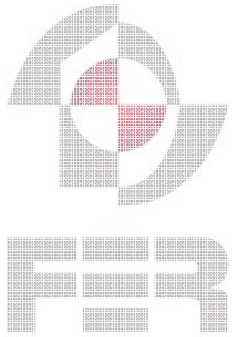
safe & idempotent

POST?

not safe & not idempotent

Interaction Design





- Response status codes
  - Part of the status line of an HTTP response message
  - Inform clients of their request's overarching result
- HTTP defines forty standard status codes
  - Divided into the five categories

- Response status code categories

Category	Description
1xx: Informational	Communicates transfer protocol-level information.
2xx: Success	Indicates that the client's request was accepted successfully.
3xx: Redirection	Indicates that the client must take some additional action in order to complete their request.
4xx: Client Error	This category of error status codes points the finger at clients.
5xx: Server Error	The server takes responsibility for these error status codes.

Interaction Design

200 (“OK”) should be used to indicate nonspecific success

- In most cases, `200` is the code the client hopes to see
- Indicates that the REST API successfully carried out the requested action, and that no more specific code in the `2xx` series is appropriate
- Should include a response body
  - unlike the `204` status code, which is used with empty response body

```
HTTP/1.1 200 OK
Date: Sun, 18 Oct 2009 08:56:53 GMT
Server: Apache/2.2.14 (Win32)
Content-Length: 44
Content-Type: text/html

<html><body><h1>Hello World</h1></body></html>
```

Interaction Design

200 (“OK”) must not be used to communicate errors in the response body

- Use the HTTP response status codes as specified by their semantics defined within the HTTP protocol
- REST API must not be compromised in an effort to accommodate less sophisticated HTTP clients
- Use codes from `4xx` and `5xx` series to communicate errors to the clients

```
HTTP/1.1 200 OK
Date: Sun, 18 Oct 2009 08:56:53 GMT
Server: Apache/2.2.14 (Win32)
Content-Length: 44
Content-Type: text/html

<html><body><h1>Resource not found</h1></body></html>
```

Interaction Design

201 (“Created”) must be used to indicate successful resource creation

- A REST API responds with the `201` status code whenever a collection creates (`POST`), or a store adds (`PUT`), a new resource at the client’s request
- Also may be used when a new resource is created as a result of some controller action
- The `Location` response header should contain the URI to the new resource

Interaction Design

201 (“Created”) must be used to indicate successful resource creation (cont’d)

Client request

```
POST /articles/croatia-in-eu/comments
Host: api.news.com

comment text in request body
```

Server response

```
HTTP/1.1 201 Created
Date: Sun, 18 Oct 2009 08:56:53 GMT
Server: Apache/2.2.14 (Win32)
Location: http://api.news.com/articles/croatia-in-eu/comments/89
Content-Length: 44
Content-Type: text/html

optional response body
```

Interaction Design

202 (“Accepted”) must be used to indicate successful start of an asynchronous action

- A `202` response indicates that the client’s request will be handled asynchronously
- Typically used for actions that take a long while to process
- Tells the client that the request appears valid, but it still may have problems once it’s finally processed
- Controller resources may send `202` responses, but other resource types should not

Interaction Design

202 (“Accepted”) must be used to indicate successful start of an asynchronous action (cont’d)

Client request to start long-running operation

```
POST /factorization/calc-primes
Host: api.math.com

100.000-digit number
```

Server response

```
HTTP/1.1 202 Accepted
Date: Sun, 18 Oct 2009 08:56:53 GMT
Server: Apache/2.2.14 (Win32)
Location: http://api.math.com/factorization/calc-primes/job-7245
```

Client request to get result of long-running operation once completed

```
GET /factorization/calc-primes/job-7245
Host: api.math.com
```

Interaction Design

204 (“No Content”) should be used when the response body is intentionally empty

- The `204` status code is usually sent out in response to a `PUT`, `POST`, or `DELETE` request, when the REST API declines to send back any status message or representation in the response message’s body
- An API may also send `204` in conjunction with a `GET` request to indicate that the requested resource exists, but has no state representation to include in the body

Interaction Design

301 (“Moved Permanently”) should be used to relocate resources

- The `301` status code indicates that the REST API resource model has been significantly redesigned and a new URI has been assigned to the requested resource
- Server did not process the request and the client is expected to resubmit the request to the new URI
- The new URI is returned to the client in the `Location` response header
- The `301` status code is used to keep old URI from breaking after URI model has been changed

Interaction Design

301 (“Moved Permanently”) should be used to relocate resources (cont’d)

Client request to an old, no more existing URI

```
GET / HTTP/1.1
Host: www.google.com
```

Server response

```
HTTP/1.1 301 Moved Permanently
Date: Sun, 18 Oct 2009 08:56:53 GMT
Server: Apache/2.2.14 (Win32)
Location: http://www.google.hr/
```

Client request to new URI

```
GET / HTTP/1.1.
Host: www.google.hr
```

Interaction Design

307 (“Temporary Redirect”) should be used to tell clients to resubmit the request to another URI

- Similar to `301`, a `307` response indicates that the REST API is not going to process the client’s request
- The client should resubmit the request to the URI specified by the response message’s `Location` header
- Used in case when a resource URI changes just for a limited period of time and recovers to its original form at some point later
  - Server maintenance or upgrade
  - Server out of service
  - Server overload (load balancing)

Interaction Design

307 (“Temporary Redirect”) should be used to tell clients to resubmit the request to another URI (cont’d)

Client initial request

```
GET /breaking-news/new-iphone-released HTTP/1.1
Host: www.news.com
```

server overload

Server response

```
HTTP/1.1 307 Temporary Redirect
Date: Sun, 18 Oct 2009 08:56:53 GMT
Server: Apache/2.2.14 (Win32)
Location: http://mirror.news.com/breaking-news/new-iphone-released
```

Client resubmits the request to new URI

```
GET /breaking-news/new-iphone-released HTTP/1.1
Host: mirror.news.com
```

Interaction Design

303 (“See Other”) should be used to refer the client to a different URI

- A controller resource **has finished its work**
- Instead of sending a potentially unwanted response body, it sends the client the URI of a response resource
- Allows a REST API to send a reference to a resource without forcing the client to download its state

Interaction Design

303 (“See Other”) should be used to refer the client to a different URI (cont’d)

Client request

```
GET /world?type=satellite&level=street HTTP/1.1
Host: www.maps.com
```

resource state extremely large

Server response

```
HTTP/1.1 303 See Other
Date: Sun, 18 Oct 2009 08:56:53 GMT
Location: http://www.maps.com/downloads/456-agd-786
Content-Length: 84
Content-Type: text/html

<html><body>
  Map size: 700 TB
  <a href="/downloads/456-agd-786">Download</a>
</body></html>
```

Interaction Design

304 (“Not Modified”) should be used to preserve bandwidth

- Status code `304` is used to support caches
- Always returns a response with **empty body**
- Similar to `204` (`"No Content"`) in that the response body must be empty, but
  - `204` is used when there is **nothing to send** in the body
  - `304` is used when **there is state information** associated with a resource **but the client already has the most recent version of the representation**

Interaction Design

400 (“Bad Request”) may be used to indicate nonspecific failure

- `400` is the generic client-side error status
- Used when no other `4xx` error code is appropriate
- The response body may contain a document describing the client’s error (unless the request method was `HEAD`)

Interaction Design

400 (“Bad Request”) may be used to indicate nonspecific failure (cont’d)

Client request (misspelled HTTP method, wrong HTTP version number)

```
ADD /users/1234 HTTTTP/1.5
Host: www.example.com
```

Server response

```
HTTP/1.1 400 Bad Request
Date: Sun, 18 Oct 2009 08:56:53 GMT
Server: Apache/2.2.14 (Win32)
Content-Length: 76
Content-Type: text/html

<html><body>
  <p>The request line contains invalid characters</p>
</body></html>
```

Interaction Design

400 (“Bad Request”) may be used to indicate nonspecific failure (cont’d)

Client request (wrong request URI)

```
GET users/1234 HTTP/1.1
Host: www.example.com
```

Server response

```
HTTP/1.1 400 Bad Request
Date: Sun, 18 Oct 2009 08:56:53 GMT
Server: Apache/2.2.14 (Win32)
Content-Length: 50
Content-Type: text/html

<html><body>
  <p>Wrong request URI</p>
</body></html>
```

Interaction Design

400 (“Bad Request”) may be used to indicate nonspecific failure (cont’d)

Client request (missing `Host` header in HTTP/1.1)

```
GET /users/1234 HTTP/1.0
```

Server response

```
HTTP/1.0 200 OK
Date: Sun, 18 Oct 2009
Server: Apache/2.2.14 (Win32)
Content-Length: 50
Content-Type: text/html

resource representation
```

```
GET /users/1234 HTTP/1.1
```

Server response

```
HTTP/1.1 400 Bad Request
Date: Sun, 18 Oct 2009
Server: Apache/2.2.14 (Win32)
Content-Length: 50
Content-Type: text/html

<html><body>
  <p>Missing Host header</p>
</body></html>
```

Interaction Design

401 (“Unauthorized”) must be used when there is a problem with the client’s credentials

- A `401` error response indicates that the client tried to operate on a protected resource without providing the proper authorization
  - Client may have provided the wrong credentials
  - Client may have provided no credentials
- It is common for a client to make a request and accept `401` response just to find out what kind of credentials to send and in what format
  - Basic
  - Digest
  - WSSE

Interaction Design

403 (“Forbidden”) should be used to forbid access regardless of authorization state

- A `403` error response indicates that the client’s request is formed correctly, but the REST API refuses to honor it
- Not a case of insufficient client credentials (that would be `401 Unauthorized`), but insufficient application-specific privileges at the time of making a request
- A `403` error response is used for fine-grained application-level permissions

Interaction Design

403 (“Forbidden”) should be used to forbid access regardless of authorization state (cont’d)

- Examples
  - A client may be authorized to interact with some, but not all of a REST API resources. If the client attempts a resource interaction that is outside of its permitted scope, the REST API should respond with `403`
  - A client may be permitted to access the resource from certain IP addresses only, for example within the corporate intranet or within the country
  - REST API may allow access to resources during the day or working hours only, for example because of regular nightly upgrade of resources or law enforcement (online banking)

Interaction Design

404 (“Not Found”) must be used when a client’s URI cannot be mapped to a resource

- Along with `200 OK`, this is the most often used and probably the most famous HTTP status code
- Informs client that the requested resource does not exists
  - Never existed
  - May exist before, but was deleted
- For store resources, may signal to the client that the URI is free
  - The client can then create a new resource by sending a `PUT` request to that URI

Interaction Design

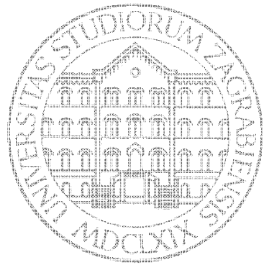
405 (“Method Not Allowed”) must be used when the HTTP method is not supported

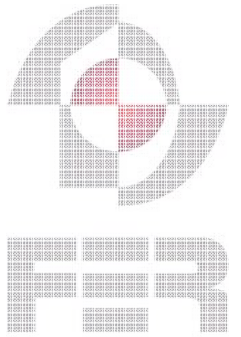
- A `405` error is an indication that the client tried to use an HTTP method that the resource does not allow
  - Resource exists
    - don’t use `404 Not Found`
  - Client provided proper credentials
    - don’t use `401 Unauthorized`
  - Client is granted a full control over the resource
    - don’t use `403 Forbidden`
  - The resource just does not support requested operation
  - Bad request? Actually is. But...
    - don’t use `400 Bad Request` since it communicates a nonspecific error
    - use more specific `405 Method Not Allowed`

Always use as precise response code as possible

Interaction Design

405 (“Method Not Allowed”) must be used when the HTTP method is not supported (cont’d)





Client request (read-only resource)

```
DELETE /predmet/rznu HTTP/1.1
Host: www.fer.unizg.hr
```

```
OPTIONS /predmet/rznu HTTP/1.1
Host: www.fer.unizg.hr
```

Server response

```
HTTP/1.1 405 Method Not Allowed
Date: Sun, 18 Oct 2009
Server: Apache/2.2.14 (Win32)
Allow: GET
```

Server response

```
HTTP/1.1 200 OK
Date: Sun, 18 Oct 2009
Server: Apache/2.2.14 (Win32)
Allow: GET
```

Interaction Design

406 (“Not Acceptable”) must be used when the requested media type cannot be served

- Used when the API is ***not able to generate*** resource representation in any of the client’s preferred media types, as indicated by the `Accept` request header

Client request

```
GET /users/1234 HTTP/1.1
Accept: application/json
```

Server response

```
HTTP/1.1 200 OK
Content-Length: 33
Content-Type: application/json

{"name": "John Smith", "age": 25}
```

Client request

```
GET /users/1234 HTTP/1.1
Accept: application/xml
```

Server response

```
HTTP/1.1 406 Not Acceptable
Date: Sun, 18 Oct 2009
Server: Apache/2.2.14 (Win32)
```

Interaction Design

415 (“Unsupported Media Type”) must be used when the media type of a request’s payload cannot be processed

- Used when the API is ***not able to accept*** the client’s supplied resource representation in request body

Client request

```
PUT /users/1234 HTTP/1.1
Content-Length: 33
Content-Type: application/json

{ "name": "John Smith",
  "age": 38 }
```

Server response

```
HTTP/1.1 200 OK
Date: Sun, 18 Oct 2009
Server: Apache/2.2.14 (Win32)
```

Client request

```
GET /users/1234 HTTP/1.1
Content-Length: 58
Content-Type : application/xml

<person><name>John Smith</name>
<age>38</age><person>
```

Server response

```
HTTP/1.1 415 Unsupported Media Type
Date: Sun, 18 Oct 2009
Server: Apache/2.2.14 (Win32)
```

Interaction Design

500 (“Internal Server Error”) should be used to indicate API malfunction

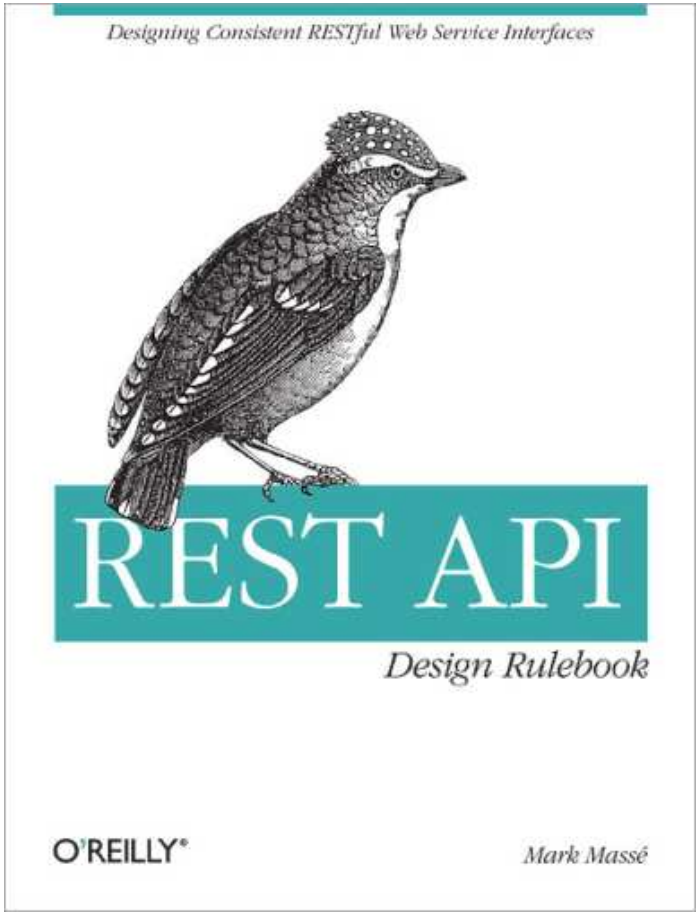
- 500 is the generic REST API error response for unexpected server-side error
  - Most web frameworks automatically respond with this response status code whenever they execute some request handler code that raises an exception

- ***A 500 error is never the client’s fault*** and therefore it is reasonable for the client to retry the exact same request that triggered this response, and hope to get a different response
  - Retry immediately

    or
  - Wait some time until server or API recovers from error

RESTful API Design

- Resource & URI design
- Interaction design
  - Request methods
  - Response codes
- Metadata design
- Representation design
- Advanced feature design
  - Versioning
  - Security
  - Response representation composition



Metadata Design

`Content-Type` header ***must*** be used whenever a message contains a body

- The `Content-Type` header specifies the type of data found within a request or response message's body (*media type*)
- Clients and servers rely on this header's value to find out how to process the sequence of bytes in a message's body

Client request

```
PUT /users/1234 HTTP/1.1
Content-Length: 33
Content-Type: application/json

{ "name": "John Smith",
  "age": 38 }
```

Server response

```
HTTP/1.1 200 OK
Content-Length: 567
Content-Type: image/gif

GIF89aX\x002X\x002⎪,\x006\x00Cü
\x002\x008\x011"LáPí┤à╚¿Qùª0!4ä
\x007\t\x00E\x010╚î┘≥▒Lw'δ}G1Oö
```

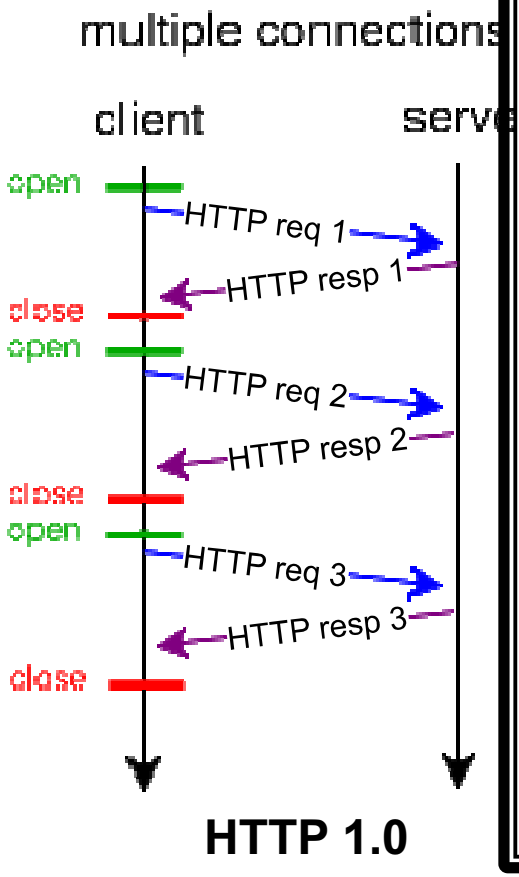
Metadata Design

`Content-Length` header ***should*** be used whenever a message contains a body

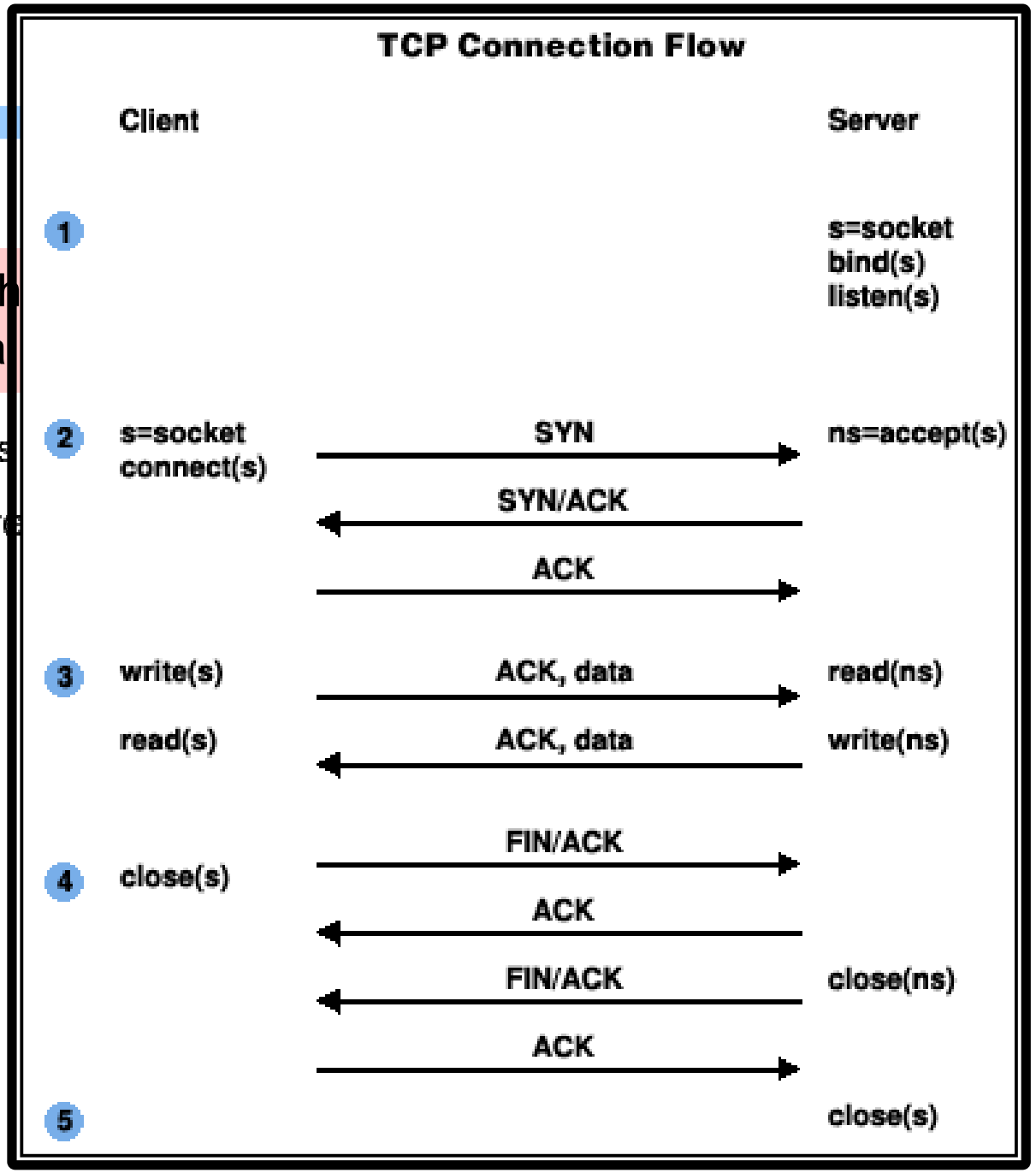
- The `Content-Length` header gives the size of the entity-body in bytes
- In responses, this header is important for two reasons
  - First, a client can know whether it has read the correct number of bytes from the connection
  - Second, a client can make a HEAD request to find out how large the entity-body is, without downloading it

Metadata Design

`Content-Length` h
message contains a

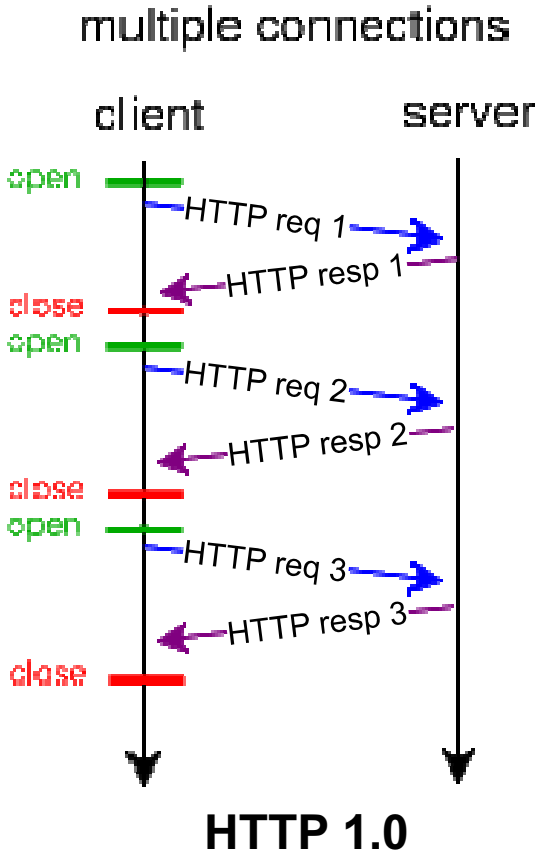


HTTP 1.0



Metadata Design

`Content-Length` header ***should*** be used whenever a message contains a body (cont'd)



HTTP 1.0

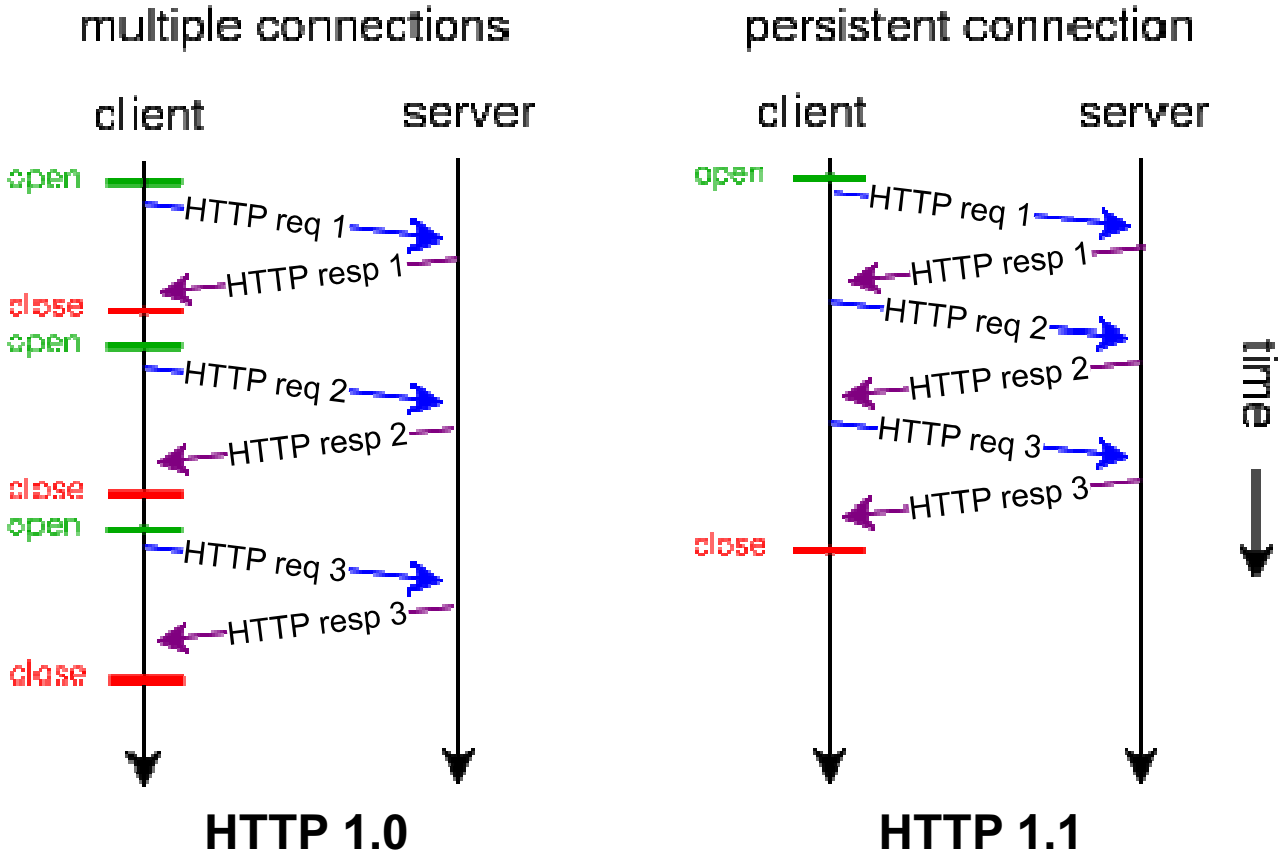
One solution (*does not work in general*):

Client reads bytes from a TCP connection until server closes the connection

- Easy in HTTP 1.0
  - New TCP connection for each HTTP request
- Does not work in HTTP 1.1
  - Persistent TCP connections

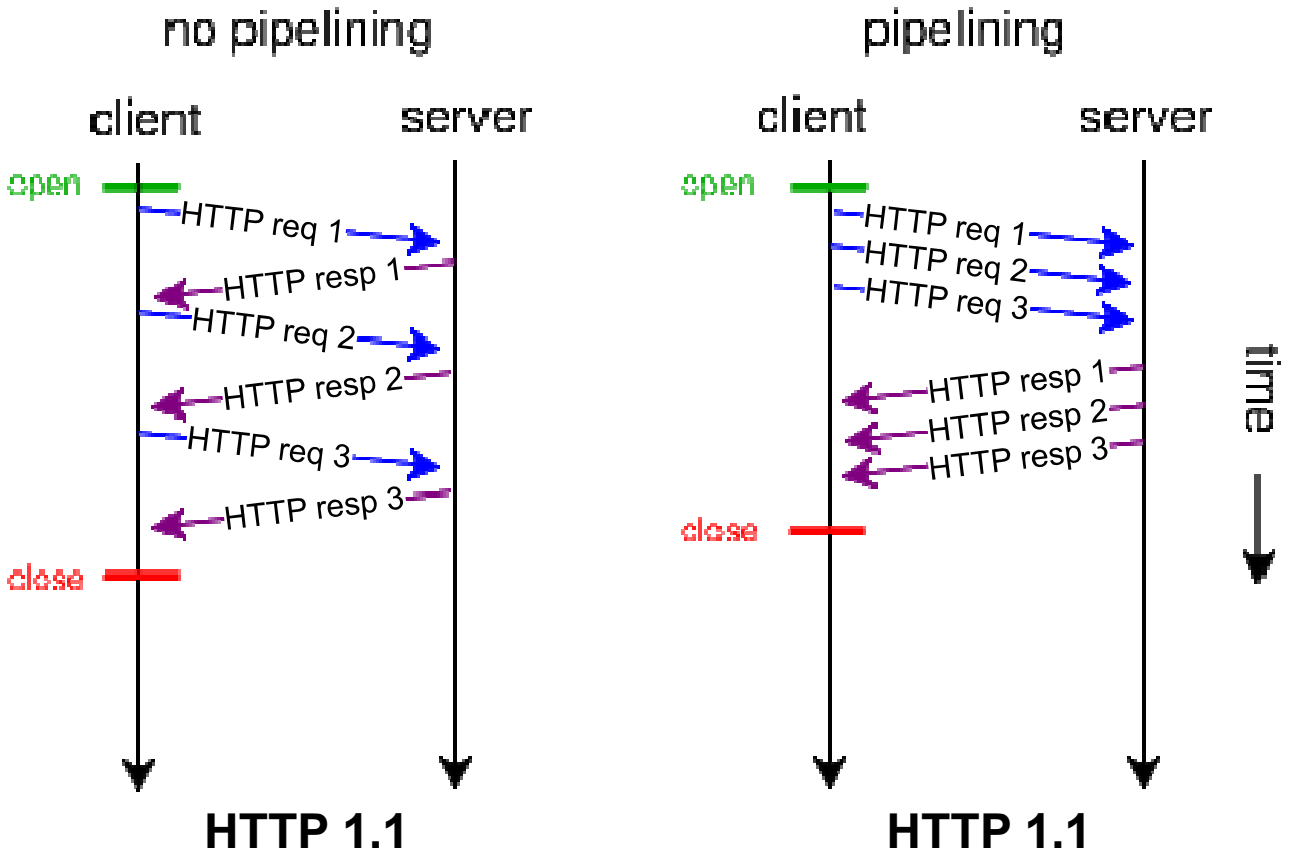
Metadata Design

`Content-Length` header ***should*** be used whenever a message contains a body (cont'd)



Metadata Design

`Content-Length` header ***should*** be used whenever a message contains a body (cont'd)



Metadata Design

`Content-Length` header ***should*** be used whenever a message contains a body (cont'd)

Client requests

```
1) GET /image1 HTTP/1.1
   Host: api.example.com

2) GET /image2 HTTP/1.1
   Host: api.example.com

3) GET /image3 HTTP/1.1
   Host: api.example.com
```

Server parses data from network

```
GET /image1 HTTP/1.1\r\nHost: api.example.c
om\r\n\r\nGET /image2 HTTP/1.1\r\nHost: api
.example.com\r\n\r\nGET /image3 HTTP/1.1\r\
nHost: api.example.com\r\n\r\n
```

Metadata Design

`Content-Length` header ***should*** be used whenever a message contains a body (cont'd)

Server responses

```
1) HTTP/1.1 200 OK
   Content-Length: 1234
   Content-Type: image/gif

   GIF89aX\x002X\x002...

2) HTTP/1.1 200 OK
   Content-Length: 567
   Content-Type: image/gif

   GIF89aX\x002X\x002...

3) HTTP/1.1 200 OK
   Content-Length: 1285
   Content-Type: image/gif

   GIF89aX\x002X\x002...
```

Client parses data from network

```
HTTP /1.1 200 OK\r\nContent-Lengt
h: 1234\r\nContent-Type: image/gi
f\r\n\r\nGIF89aX\x002X\x002...HT
TP/1.1 200 OK\r\nContent-Length:
567\r\nContent-Type: image/gif\r\
n\r\nGIF89aX\x002X\x002...HTTP/1
.1 200 OK\r\nContent-Length: 1285
\r\nContent-Type: image/gif\r\n\r
\nGIF89aX\x002X\x002...
```

Metadata Design

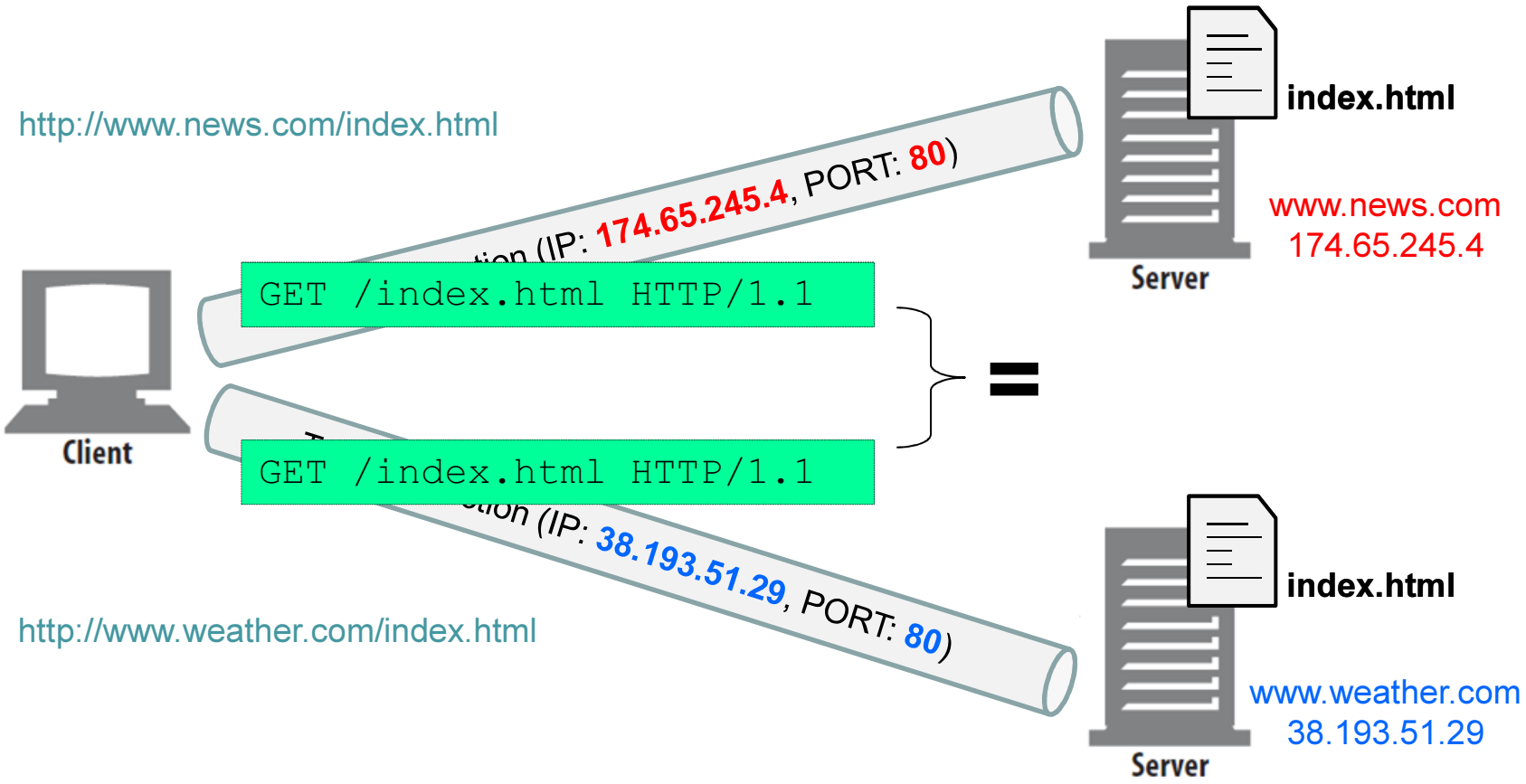
`Host` header ***must*** be used to support virtual server hosting

- The `Host` header is a request header
  - Optional in HTTP 1.0
  - Mandatory in HTTP 1.1

- Allows hosting of multiple domains on a single server machine (*single IP address*)

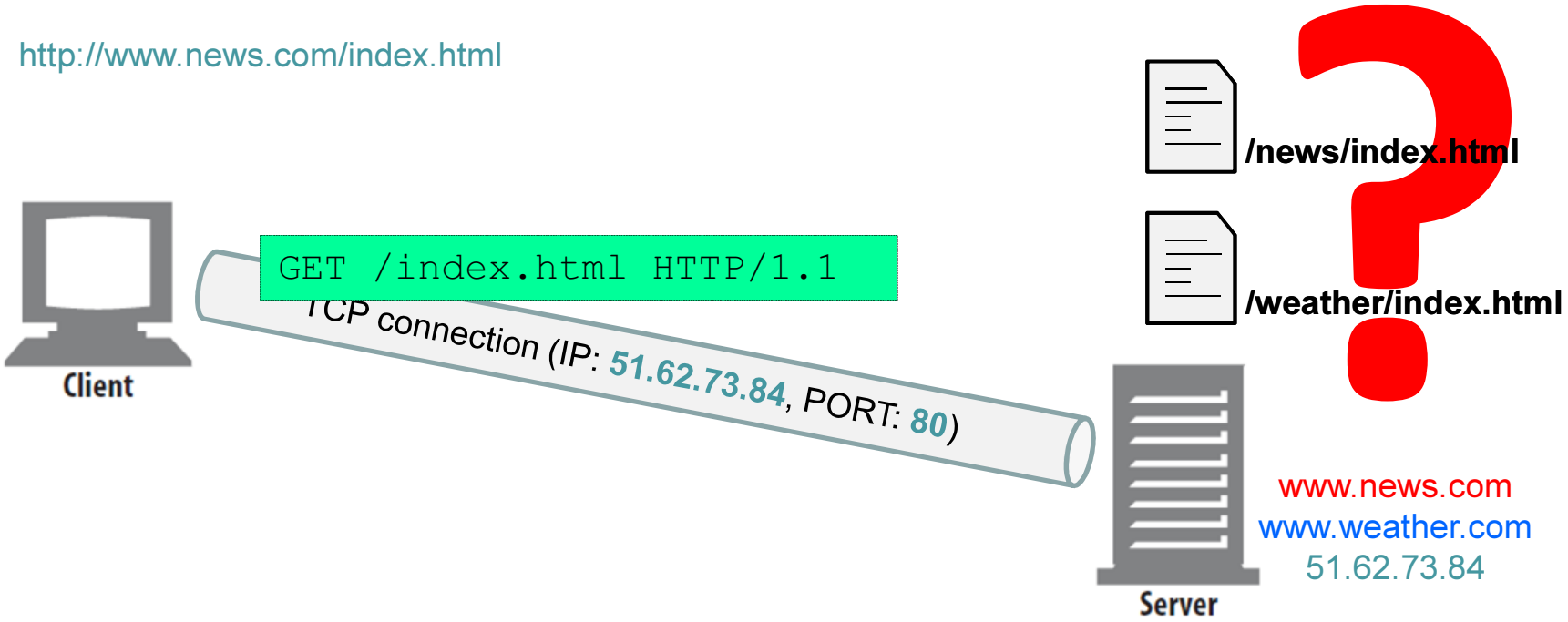
Metadata Design

`Host` header ***must*** be used to support virtual server hosting (cont'd)



Metadata Design

`Host` header ***must*** be used to support virtual server hosting (cont'd)



Metadata Design

`Host` header ***must*** be used to support virtual server hosting (cont'd)