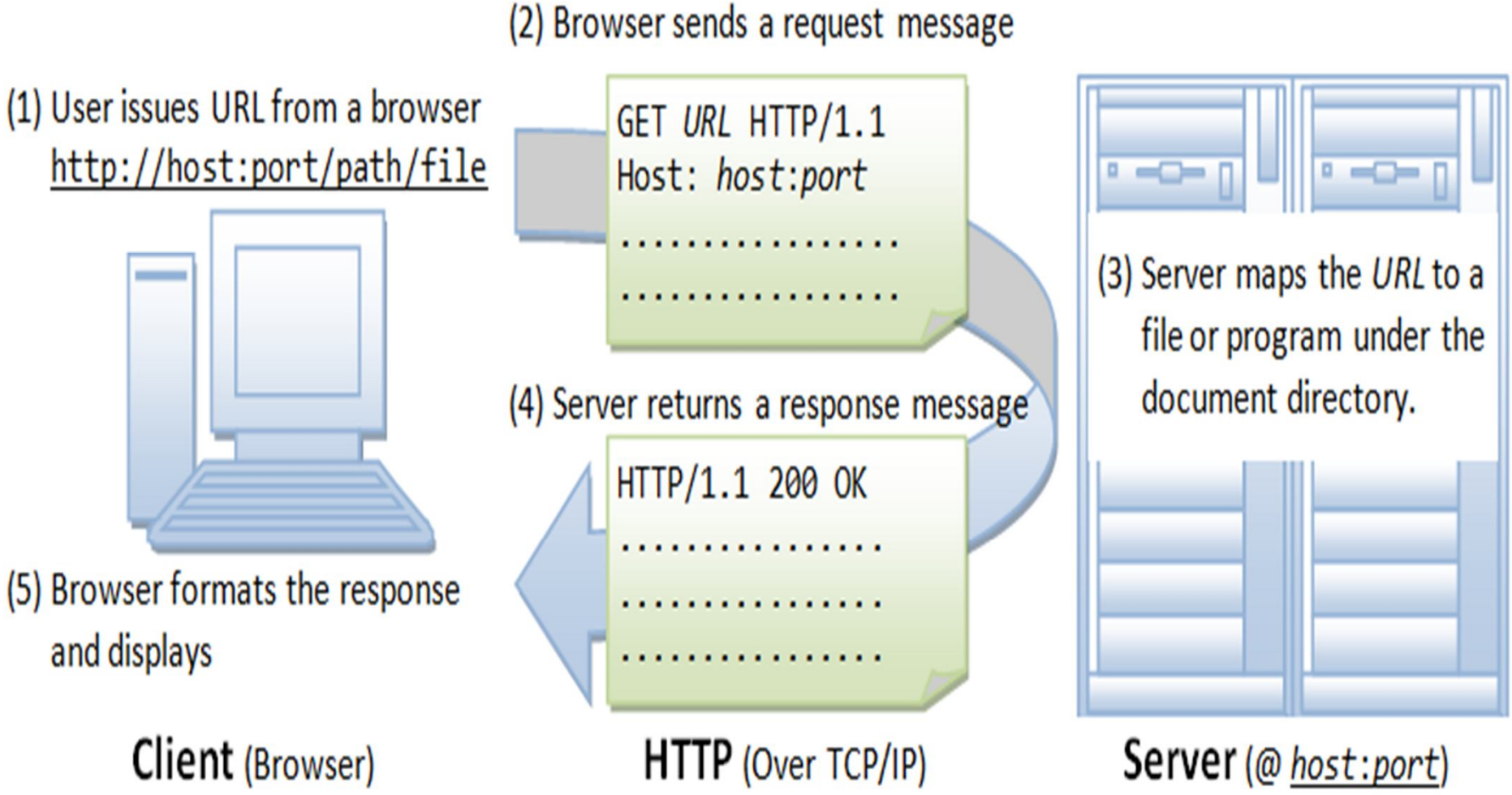
Module 4

Client/Server Communication

- HTTP- Request/Response Model
- HTTP Methods
- RESTful APIs
- AJAX
- AJAX with JSON

HyperText Transfer Protocol (HTTP)

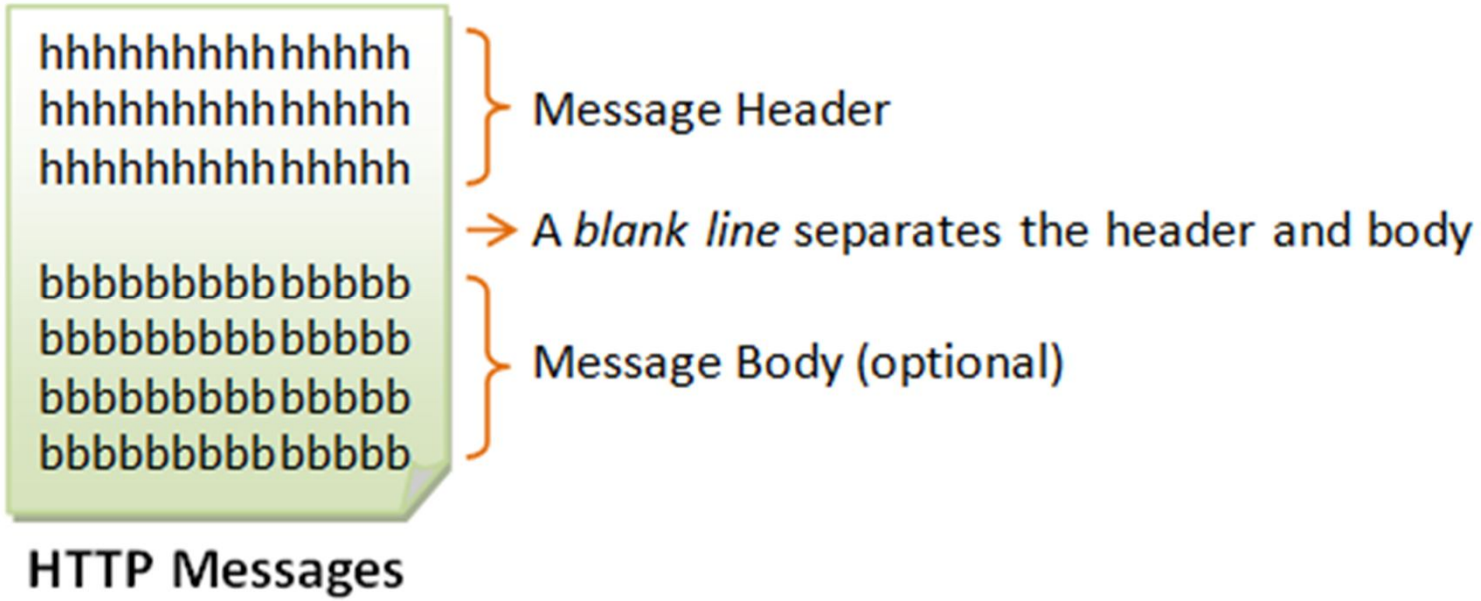


HTTP

- HTTP is an *asymmetric request-response client-server* protocol. An HTTP client sends a request message to an HTTP server. The server, in turn, returns a response message. In other words, HTTP is a *pull protocol*, the client *pulls* information from the server (instead of server *pushes* information down to the client).
- HTTP is a stateless protocol. In other words, the current request does not know what has been done in the previous requests.
- HTTP permits negotiating of data type and representation, so as to allow systems to be built independently of the data being transferred.
- It is an application-level protocol for distributed, collaborative, hypermedia information systems.

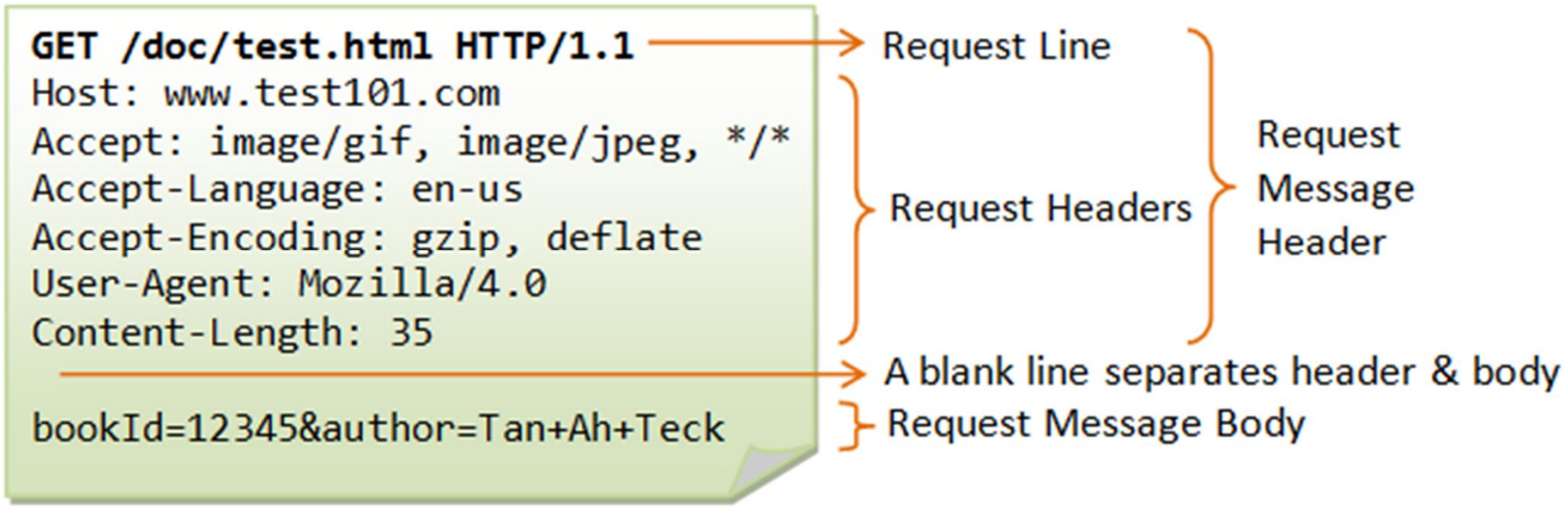
HTTP Request and Response Messages

- An HTTP message consists of a *message header* and an optional *message body*, separated by a *blank line*.



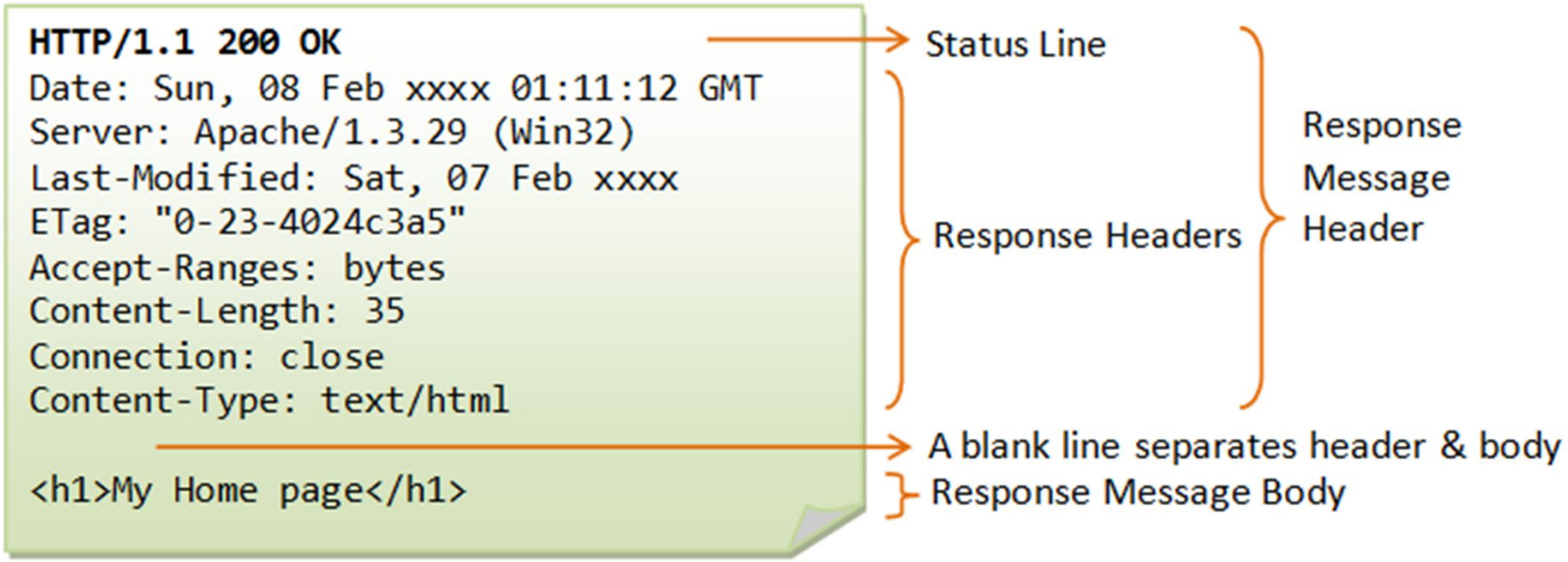
HTTP Request Message

- ***request-method-name request-URI HTTP-version***
- *request-method-name*: HTTP protocol defines a set of request methods, e.g., GET, POST, HEAD, and OPTIONS.
- The client can use one of these methods to send a request to the server.
- *request-URI*: specifies the resource requested.
- *HTTP-version*: HTTP/1.0 and HTTP/1.1.



HTTP Response Message

- The response message body contains the resource data requested.



Status Line

- The status code is a 3-digit number:
- 1xx (Informational): Request received, server is continuing the process.
- 2xx (Success): The request was successfully received, understood, accepted and serviced.
- 3xx (Redirection): Further action must be taken in order to complete the request.
- 4xx (Client Error): The request contains bad syntax or cannot be understood.
- 5xx (Server Error): The server failed to fulfill an apparently valid request.

Some common HTTP status codes

Status Range	Description	Examples
100	Informational	100 Continue
200	Successful	200 OK
201	Created	
202	Accepted	
300	Redirection	301 Moved Permanently
304	Not Modified	
400	Client error	401 Unauthorized
402	Payment Required	
404	Not Found	
405	Method Not Allowed	
500	Server error	500 Internal Server Error
501	Not Implemented	

HTTP Request Methods

- A client can use one of these request methods to send a request message to an HTTP server. The methods are:
- GET: A client can use the GET request to get a web resource from the server.
- HEAD: A client can use the HEAD request to get the header that a GET request would have obtained. Since the header contains the last-modified date of the data, this can be used to check against the local cache copy.
- POST: Used to post data up to the web server.
- PUT: Ask the server to store the data.
- DELETE: Ask the server to delete the data.
- TRACE: Ask the server to return a diagnostic trace of the actions it takes.
- OPTIONS: Ask the server to return the list of request methods it supports.
- **Refer more on this:** https://restfulapi.net/http-methods/

URL and URI

- **URL (Uniform Resource Locator)** is used to uniquely identify a resource over the web.
- *protocol*://*hostname*:*port*/*path-and-file-name*
- ***Eg:***//www.ftp.org/docs/test.txt,mailto:user@test101.com, news:soc.culture.Singapore, telnet://www.nowhere123.com/
- The URL after encoding is called *encoded URL*. (%20 or '+' for blank space)
- **URI (Uniform Resource Identifier)** is more general than URL, which can even locate a *fragment* within a resource.
- http://host:port/path?request-parameters#nameAnchor
- The request parameters, in the form of name=value pairs, are separated from the URL by a '?'. The name=value pairs are separated by a '&'.
- The #nameAnchor identifies a fragment within the HTML document, defined via the anchor tag ....
- URL rewriting for session management, e.g., "...;sessionID=xxxxxx".

REST

- **Representational state transfer (REST)** or **RESTful** Web services are one way of providing interoperability between computer systems on the Internet. It was introduced and defined in 2000 by Roy Fielding in his doctoral dissertation.
- REST is *an architecture style* for designing networked applications. It is a lightweight that, rather than using complex mechanisms such as CORBA, RPC or SOAP to connect between machines, simple HTTP is used to make calls between machines. In many ways, the World Wide Web itself, based on HTTP, can be viewed as a REST-based architecture.

- RESTful applications use HTTP requests to post data (create and/or update), read data (e.g., make queries), and delete data.
- Thus, REST uses HTTP for all four CRUD (Create/Read/Update/Delete) operations.
- REST is not a "standard". There will never be a W3C recommendation for REST, for example.
- DNS is another valid discovery mechanism. FWIW, UDDIv3 does specify a REST interface.
- A REST service is a web service, but a web service is not necessarily a REST service.

REST Constraints

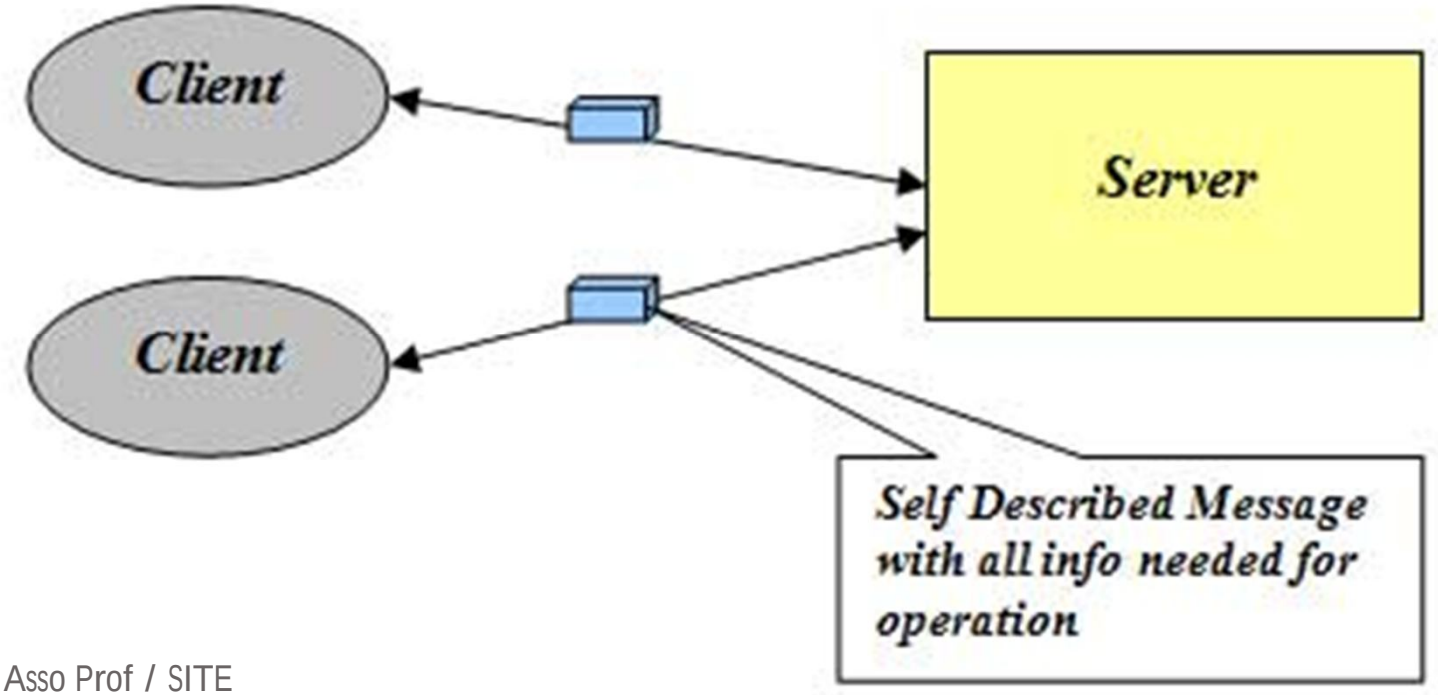
- If you follow all constraints (design rules) designed by the REST architectural style your systems is considered RESTful. These constraints don't dictate what kind of technology to use; they only define how data is transferred between components and what benefits we get following the guidelines.
  - Client-Server
  - Stateless
  - Cacheable
  - Uniform Interface
  - Layered System
  - Code On Demand (Optional)

Client-Server

- A client component that sends requests and a server component that receives requests. After receiving a request the server may also issue a response to the client.
- Applying separation of concerns: Client-Server as in the diagram :
  - Separates user interface concerns from data storage concerns.
  - Improves portability of interface across multiple platforms.
  - Improves scalability by simplifying server components.
  - Allows the components to evolve independently.

Stateless

- The constraint says that the server should not remember the state of the application. As a consequence, the client should send all information necessary for execution along with each request, because the server cannot reuse information from previous requests as it didn't memorize them.

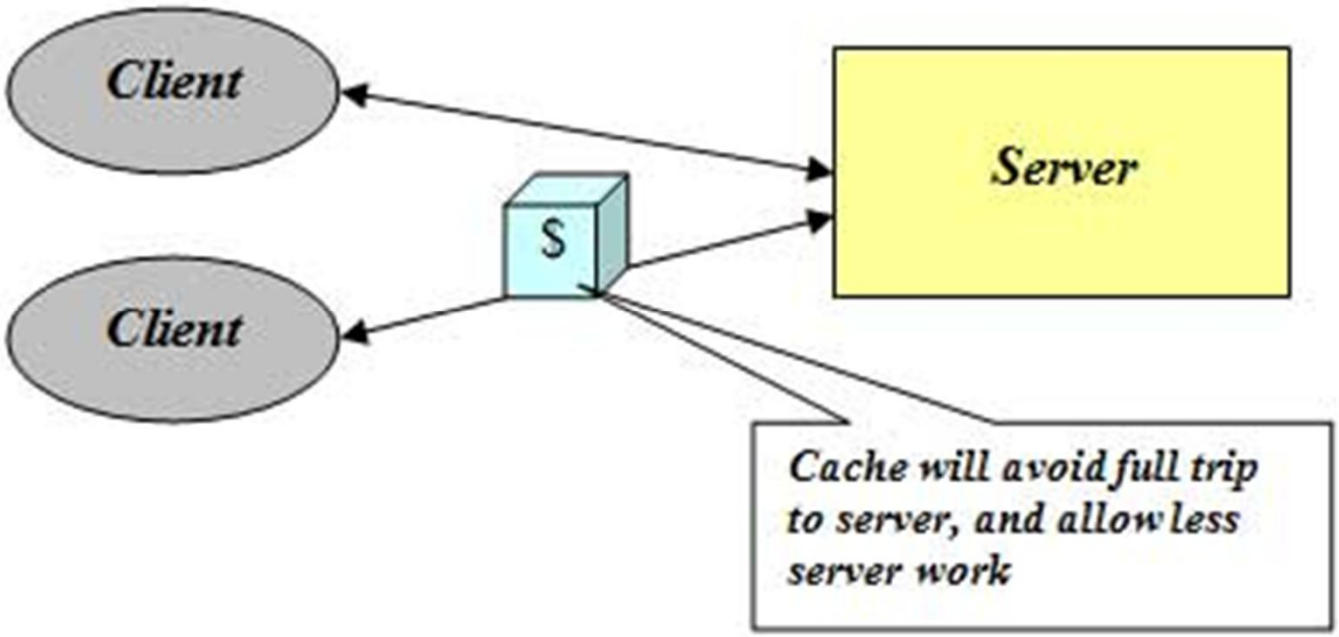


"State Transfer" in REST

- Services in REST do not maintain the state of an interaction with a requester; that is, if an interaction requires a state, all states must be part of the messages exchanged.
- By being stateless, services increase their reliability by simplifying recovery processing. (A failed service can recover quickly.)
- Furthermore, scalability of such services is improved because the services do not need to persist state and do not consume memory, representing active interactions.
- Both reliability and scalability are required properties of the Web. By following the REST architectural style, you can meet these requirements.

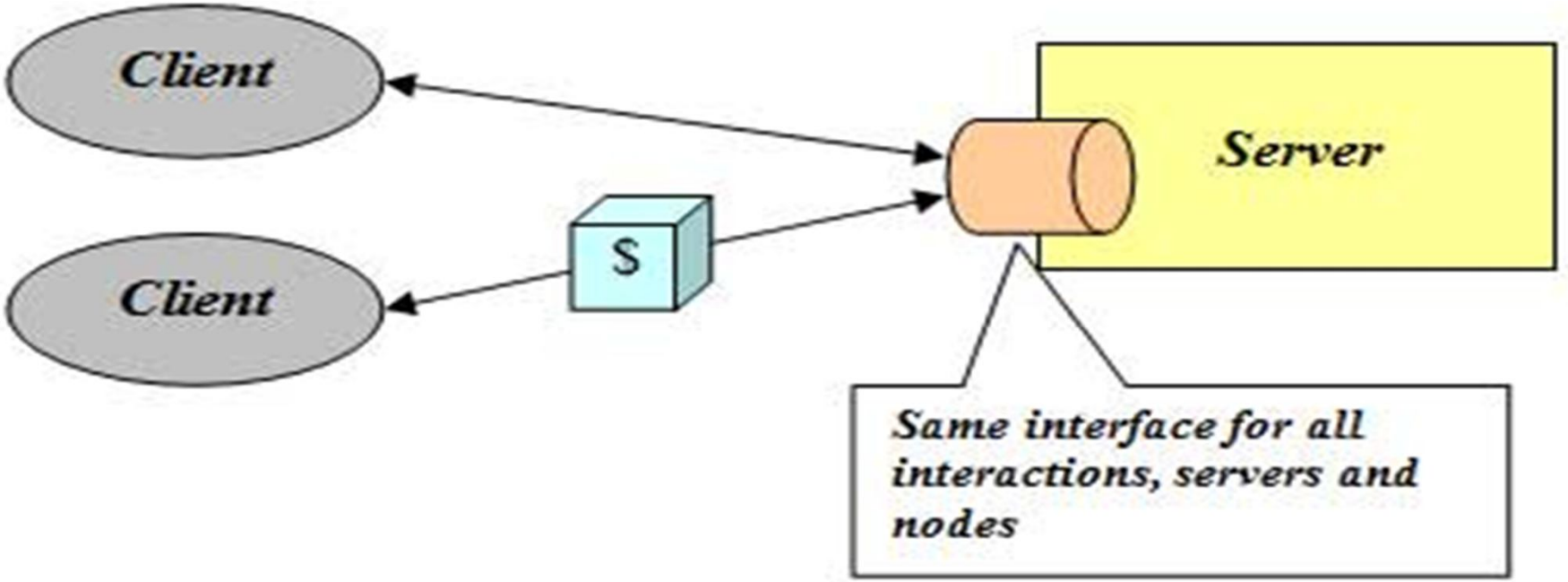
Cacheable

- REST includes cache constraints so that the "second fetch" doesn't have to be made at all if the data is already sitting in your local cache.
- If your data can be designed in such a way to take advantage of this, you can reduce total network traffic by orders of magnitude.



Uniform Interface

- REST assumes a simple interface to manipulate resources in a generic manner. This interface basically supports the create, retrieve, update, and delete (CRUD) method.



- REST is defined by four interface constraints: identification of resources; manipulation of resources through representations; self-descriptive messages; and, hypermedia as the engine of application state.
- Messages are self-describing.
- **Example :** Each request can be interpreted by itself

1. /search-results?q=to-**do**
2. /search-results?q=todo&page=2
3. /search-results?q=todo&page=3

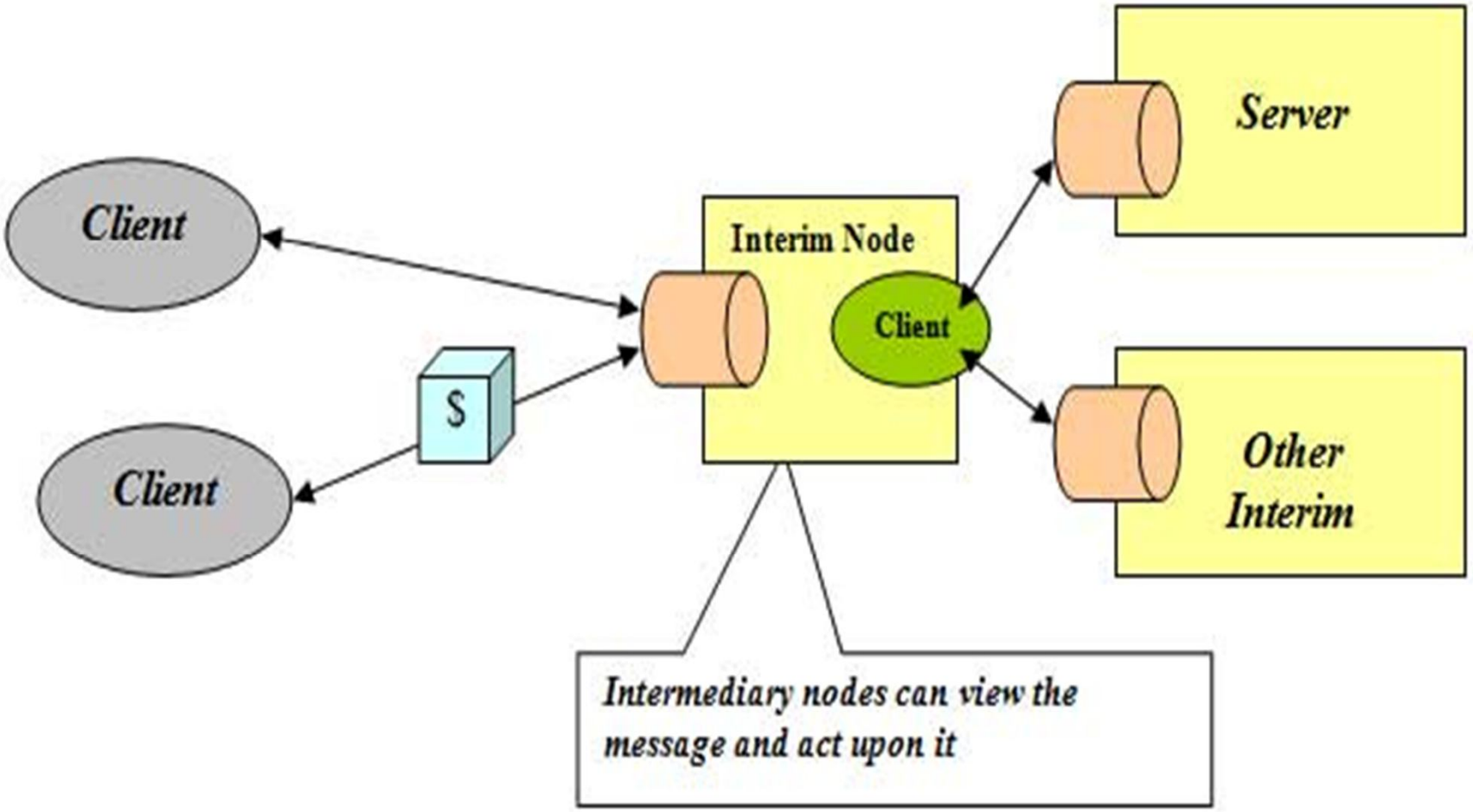
"Representational" in REST

- The basic concept of the REST architecture is that of a resource.
- A resource is any piece of information that you can identify uniquely.
- In REST, requesters and services exchange messages that typically contain both data and metadata.
  - The data part of a message corresponds to the resource in a particular representation as described by the accompanying metadata (format), which might also contain additional processing directives.
- You can exchange a resource in multiple representations. Both communication partners might agree to a particular representation in advance.

- For **example**, the data of a message might be information about the current weather in New York City the resource. The data might be rendered as an HTML document, and the language in which this document is encoded might be German.
- In the REST architectural style, servers only offer resources. Resources are conceptual things about which clients and servers communicate.
  - **Example** for REST resource: /todolists/7/items/35/
- This above thing is not a command, it is the address of a resource, a thing.
- Representation might be in any format: HTML, JSON,XML, Pdf, Video,…
  - **Eg Video Resource:**
  - https://www.youtube.com/watch?v=t27jSNvcotQ

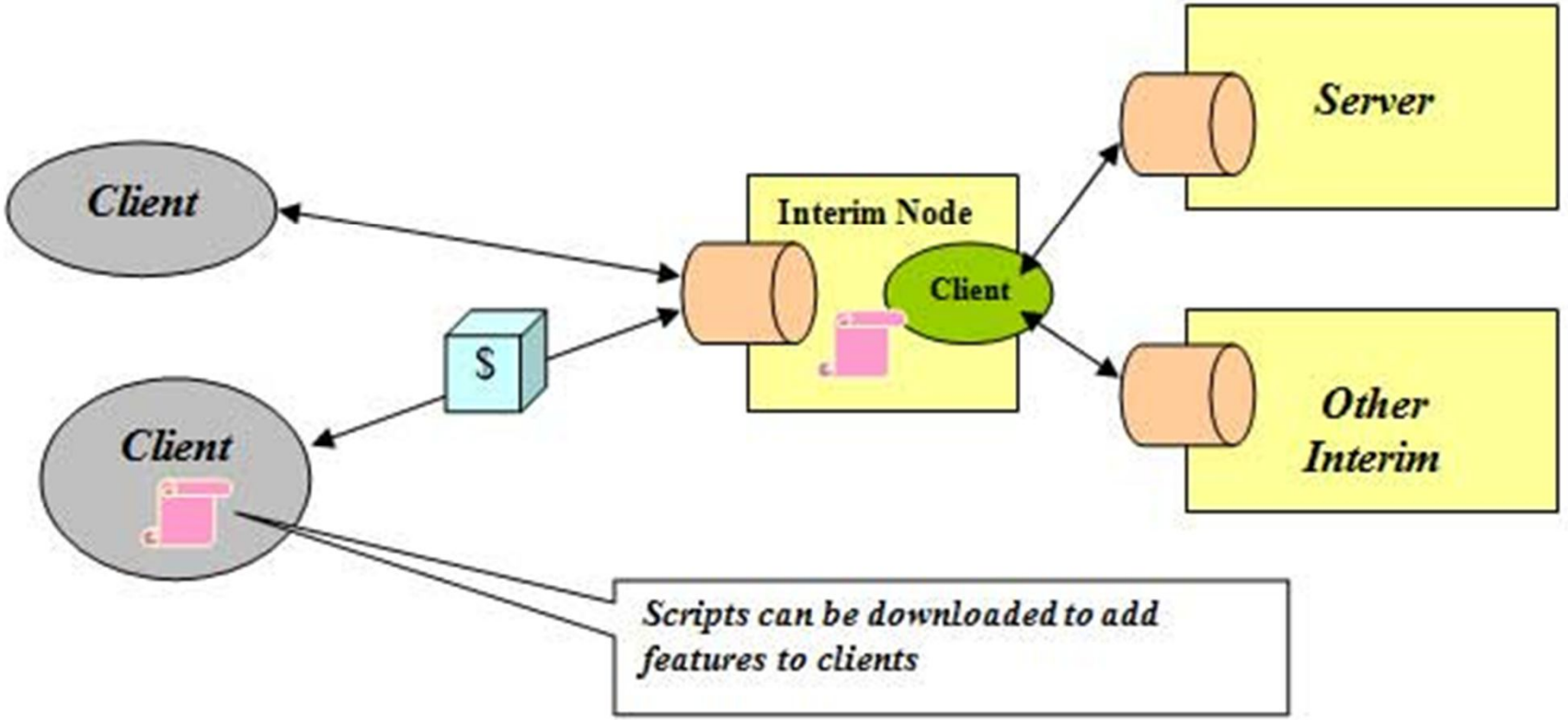
Layered System

- There are many layers between the client and the server. These are called intermediaries and can be added at various 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.
- Intermediaries include proxies (Chosen by Client) and gateways (Chosen by Server).
- This contributes to the scalability of the overall environment.


Client
Client
S
Interim Node
Client
Server
Other
Interim
Intermediary nodes can view the
message and act upon it

Code on Demand (Optional)

- It allows a client to download and execute code from a server.
- For example, if all of the client software within an organization is known to support Java applets , then services within that organization can be constructed such that they gain the benefit of enhanced functionality via downloadable Java classes.
- At the same time, however, the organization's firewall may prevent the transfer of Java applets from external sources, and thus to the rest of the Web it will appear as if those clients do not support code-on-demand.
- This simplifies clients by reducing the number of features required to be pre-implemented.


Client
Interim Node
Client
Server
S
Client
Other
Interim
Scripts can be downloaded to add
features to clients

REST and HTTP are not same !!

- **REST != HTTP**
- In simplest words, in the REST architectural style, data and functionality are considered resources and are accessed using Uniform Resource Identifiers (URIs). The resources are acted upon by using a set of simple, well-defined operations. The clients and servers exchange representations of resources by using a standardized interface and protocol – typically HTTP.

Real Time Examples

- webservices.washington.edu →

https://itconnect.uw.edu/service/enterprise-web-services-and-events/

Enterprise Web Services are a set of consistent, secure, REST Web services that enable UW developers to integrate applications and Web pages with essential UW business systems.

- **Service Options**
- Enterprise Web Services are fully documented in the UW Web Services Registry (webservices.washington.edu). They include:
- Student Web Service (SWS) -SWS v5.4.1 Released 01/25/2017
- HR/Payroll Web Service (HRPWS)
- Financial Web Service (FWS)
- Person Web Service (PWS)
- Space Web Service
- ID Card Web Service