Computer Network I

Reti di Calcolatori I

Università di Napoli Federico II – Scuola Politecnica e delle Scienze di Base
Corso di Laurea in Informatica

Riccardo Caccavale

(riccardo.caccavale@unina.it)

UNIVERSITÀ DEGLI STUDI DI NAPOLI
FEDERICO II

DIETI

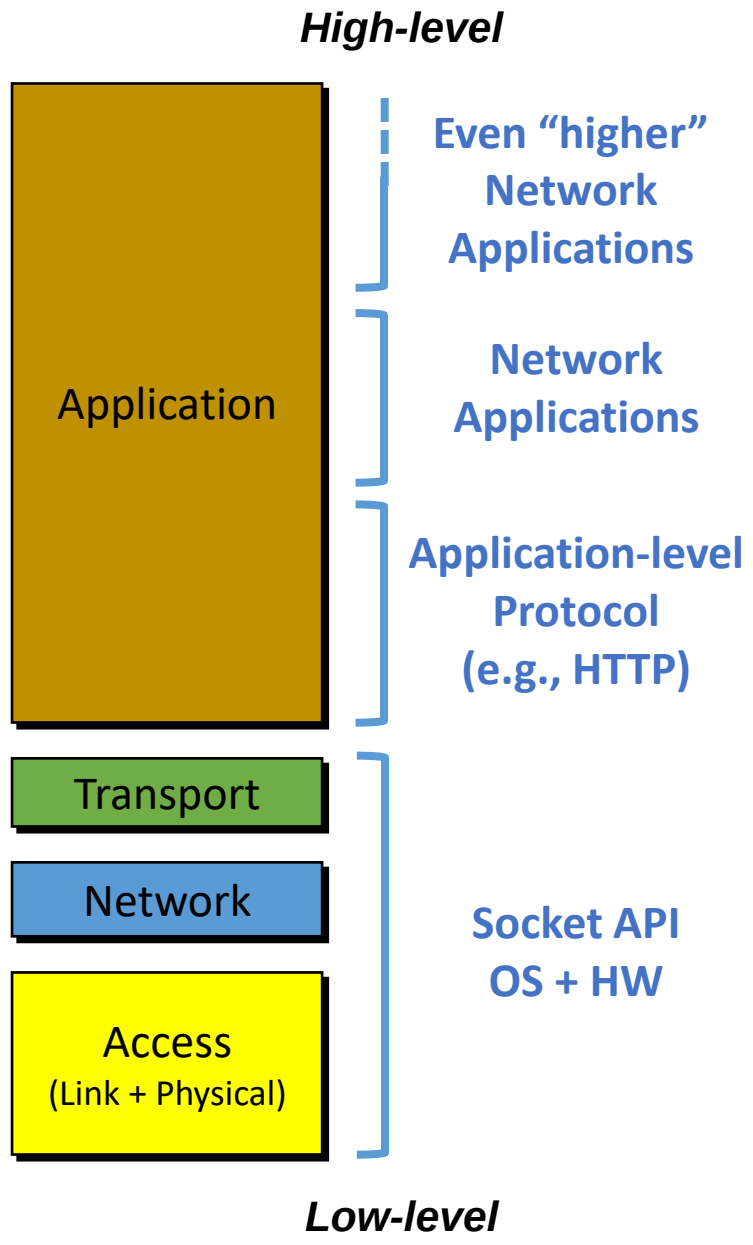
UNINA

Network Programming

Designing Network Applications

- We can design **applications that use UDP or TCP** transport protocols but for the application-level protocol we can choose between 2 alternatives:
  - **Standard protocol**.
  - **Proprietary (custom) protocol**.

- A possible approach (quite common) is to design network applications "on top" of the **standard HTTP protocol** (as web-based applications):
  - **HTTP is well known**: several network elements already works with HTTP (servers, browsers, etc.) and many programming languages already have HTTP-related APIs.
  - **HTTP is versatile**: several functions can be implemented through a **combination of standard HTTP methods** and **header lines**.



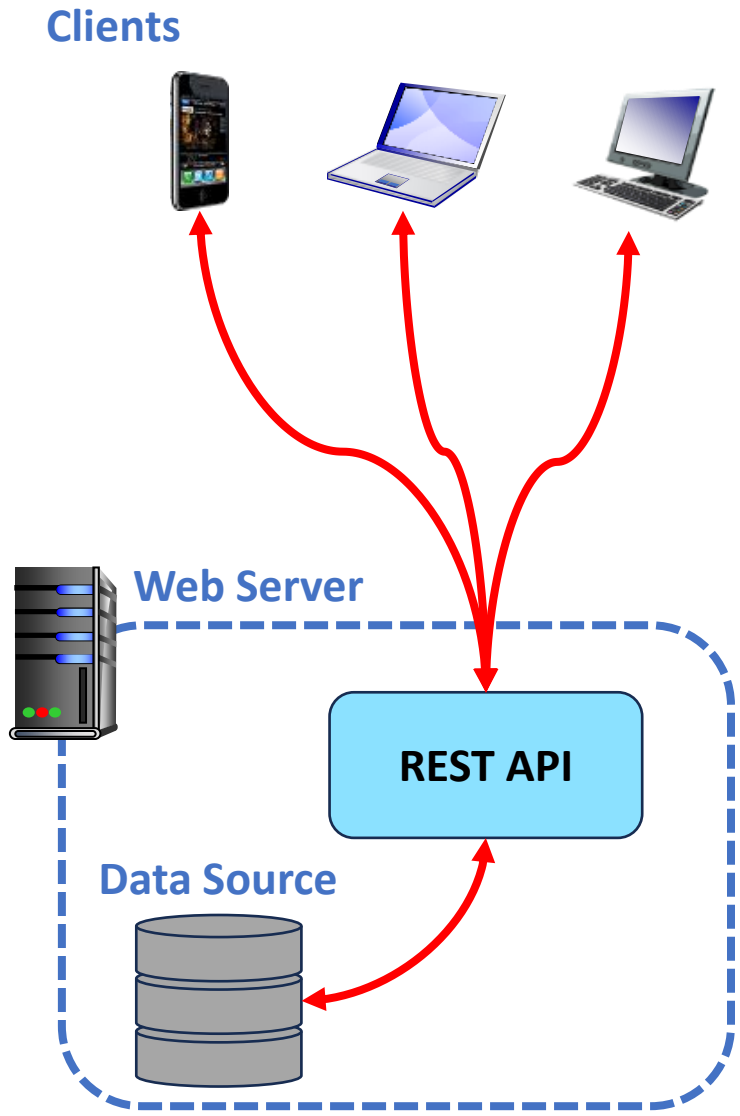
Network Programming

REST

- **REST** (Representational State Transfer) is a **set of architectural principles** that guides programmer in defining **modular, scalable and flexible web-based applications**.

- Web applications following such principles are typically called **RESTful APIs**.

- In theory, **REST principles are not tied to specific standards, programming languages, or technologies**, but these are strongly related to HTTP.

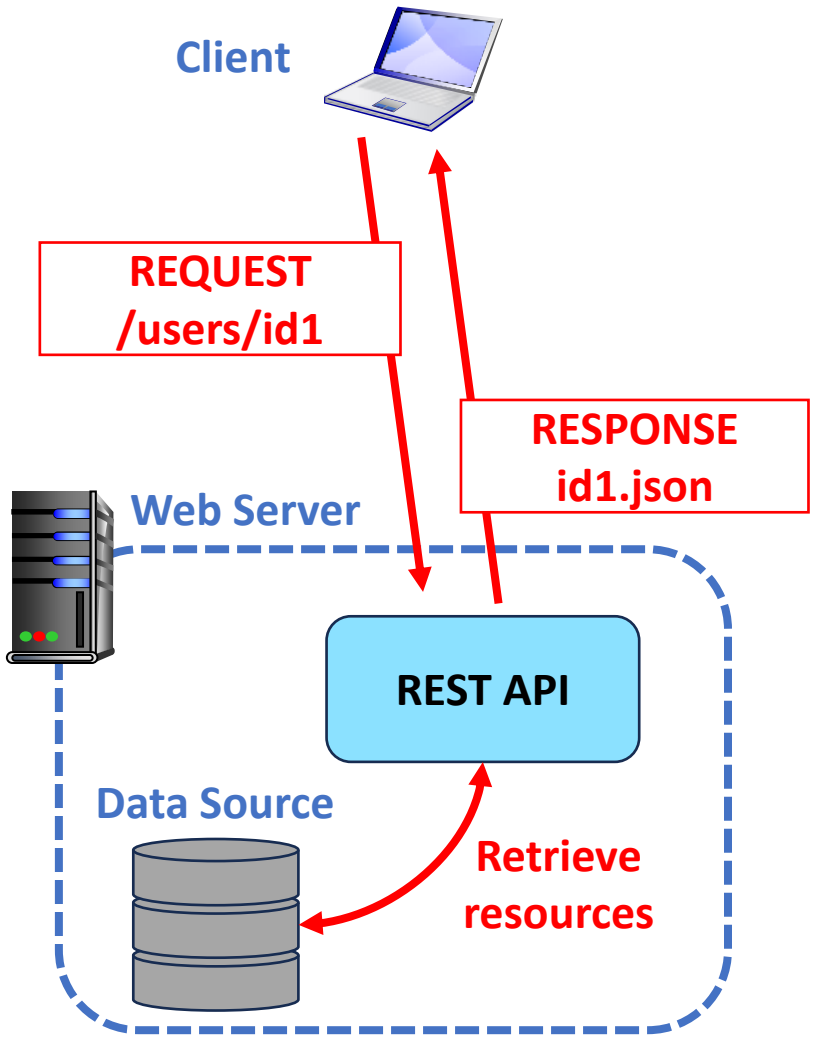


Network Programming

REST Architectural Elements

- A **REST API works as an interface** between multiple (and different) **clients** and the shared **resources** (e.g., a database).

- The **main REST elements** to be identified are:
  - **Resources**: the **pieces of information** that hosts are willing to manipulate. Resources are **identified by URIs** (Uniform Resource Identifiers).
    - We can see **URI as a generalized form of URL** that also include resource names.
  - **Representations**: how resources are structured and represented in a **specific format** (e.g., plain text, XML, YAML, JSON, etc.).



Network Programming

The Six REST Principles

1. **Client-server**. REST rely on client-server applications where the two **hosts are independent**:
   - **Client** application **only knows the URI** of resources to be requested.
   - **Server** application only **passes resources** via HTTP.
2. **Statelessness**. The state of the conversation is not maintained, each request must be self-contained.
3. **Cacheability**. Resources **should be cacheable** on the client or server side.
   - Responses should also contain information about whether caching is allowed or not for a specific resource.
4. **Uniform interface**. All API requests for the **same resource should look the same**, no matter where the request comes from.
5. **Layered system architecture**. RESTful APIs should consider that **messages can be forwarded through different intermediaries** (there is no direct link between client and server), but this should not be perceived by the client nor the server.
6. **Code on demand** (optional). Responses can also contain **executable code** (such as Java applets). In these cases, the code should only **run on-demand**.

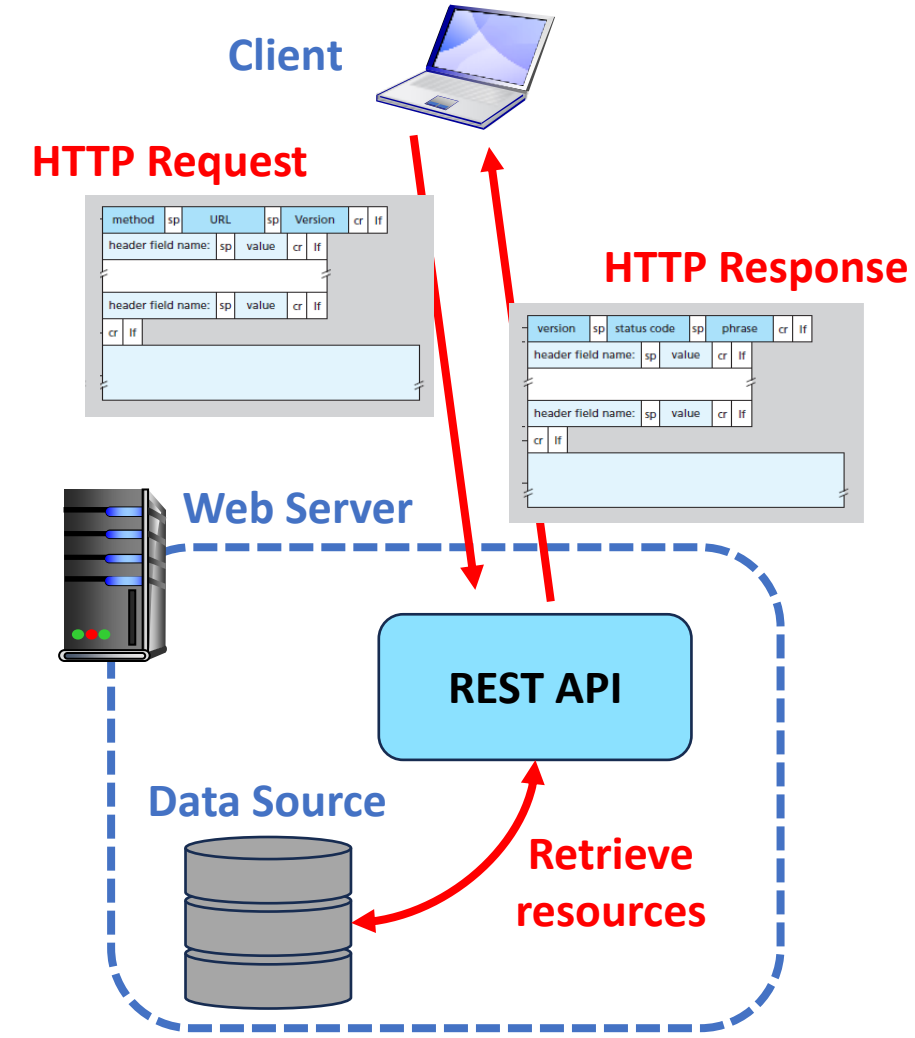
Network Programming

REST API in Practice

- It is possible to notice that **REST principles closely resemble some HTTP features**. In a way, REST defines how to “boldly” use HTTP in a network application.

- The general idea is to manipulate standardly-represented resources through standardly-represented messages.

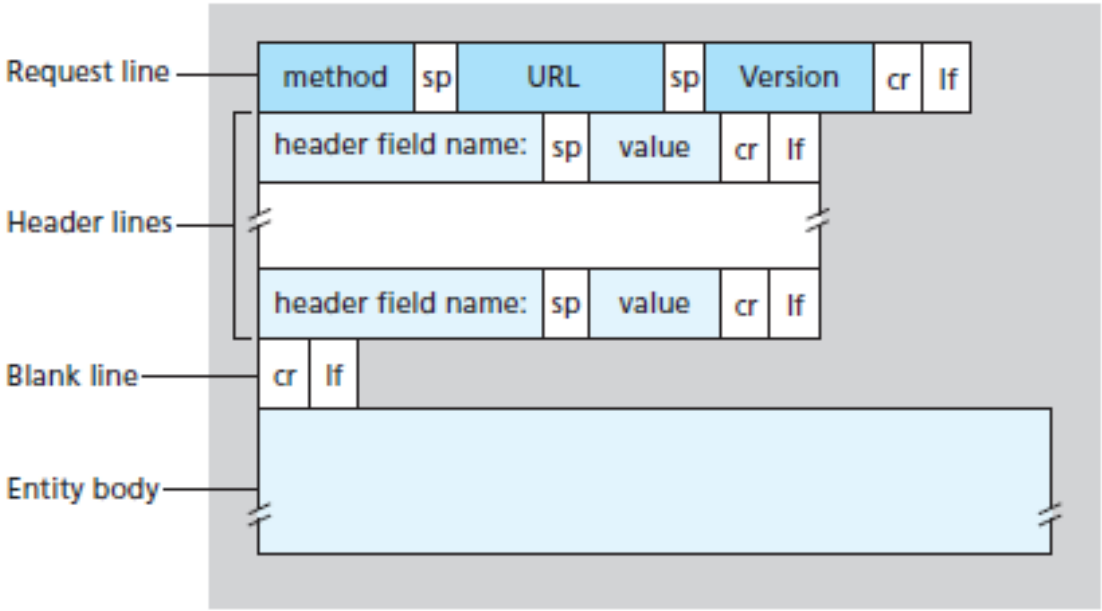
- We rely on **HTTP** to implement client/server communication:
  - The client access resources through HTTP **requests**.
  - The server provides answers through HTTP **responses**.



Network Programming

HTTP remind

- The **request message** format include the following fields:
  - The **method**: specifies the requested command to be executed by the server.
  - The **URL**: is used to identify the object on which we want to operate.
  - The **version**: specifies the HTTP version (e.g., HTTP/1.1).
  - The **header lines**: contain the parameters of the request, the number and the type of these lines are not fixed. Each line include the **name** and the **value** of the parameter.
  - The **body**: is method-specific and contains data that are potentially associated with the command.



The fields are separated by special characters:

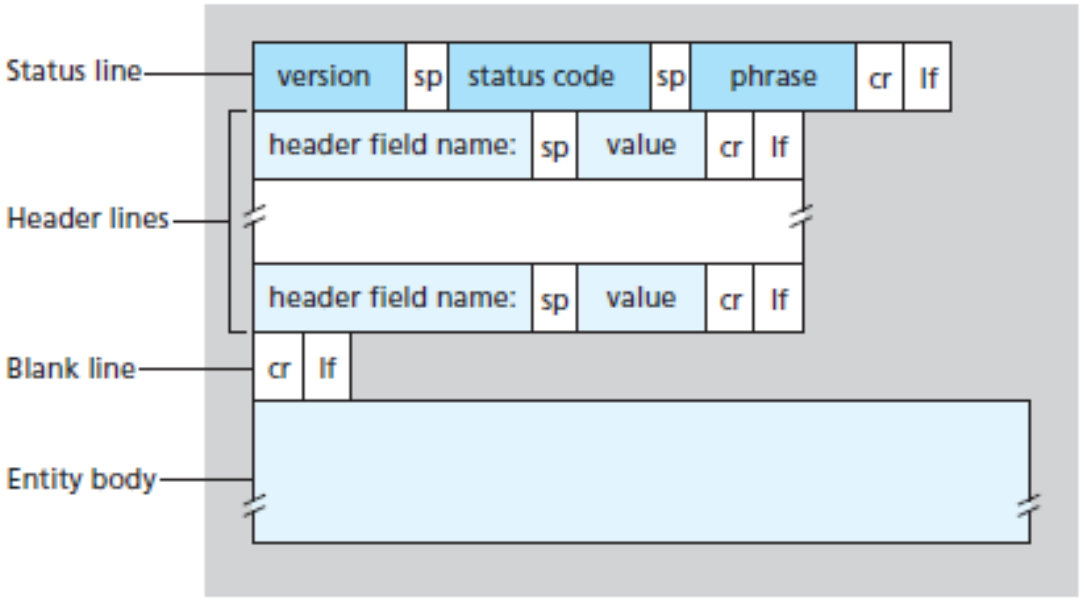
- The sp is space character.
- The cr is carriage return (\r).
- The lf is line feed (\n).

Network Programming

HTTP remind

- The **response format message** is similar to the request one.

- Instead of a request line, we have a **status line** that reports the outcome of the command that includes:
  - The **version**: reports the HTTP version of the server's response.
  - The **status code**: a code (number) that specifies the outcome of the command.
  - The **phrase**: contains the result of the request.



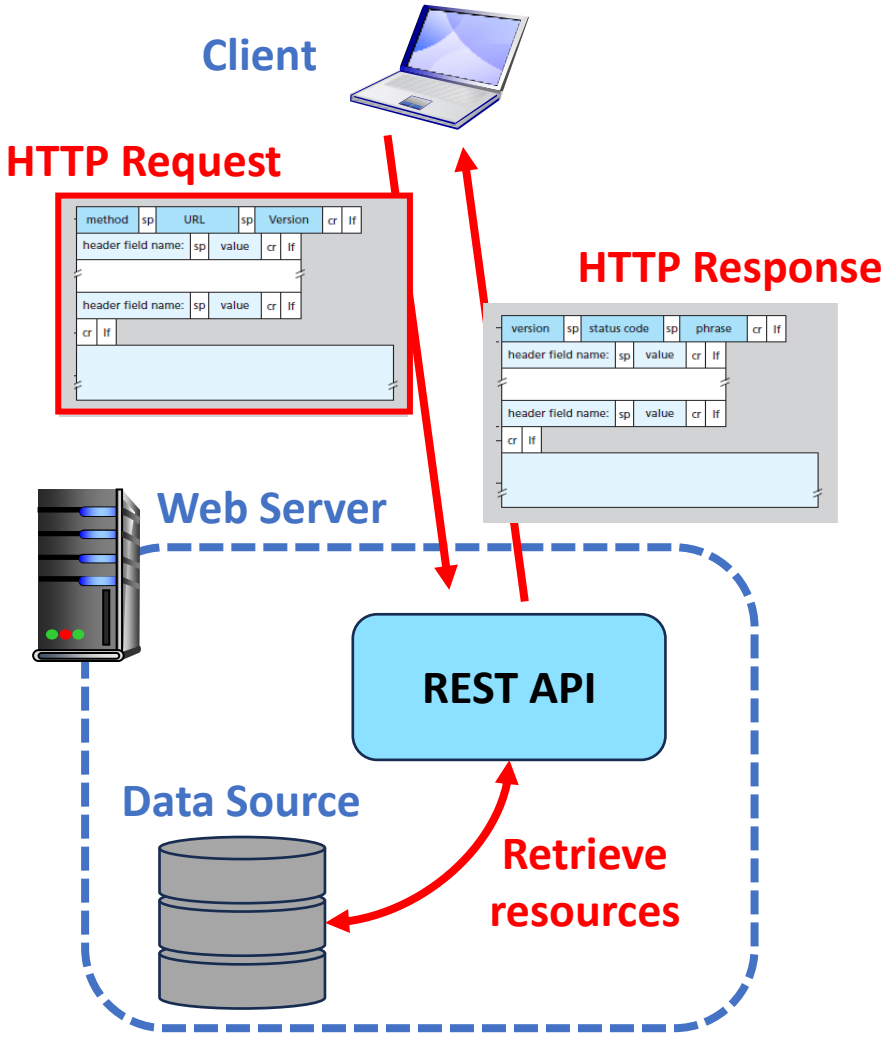
The fields are separated by special characters:

- The sp is space character.
- The cr is carriage return (\r).
- The lf is line feed (\n).

Network Programming

REST API in Practice

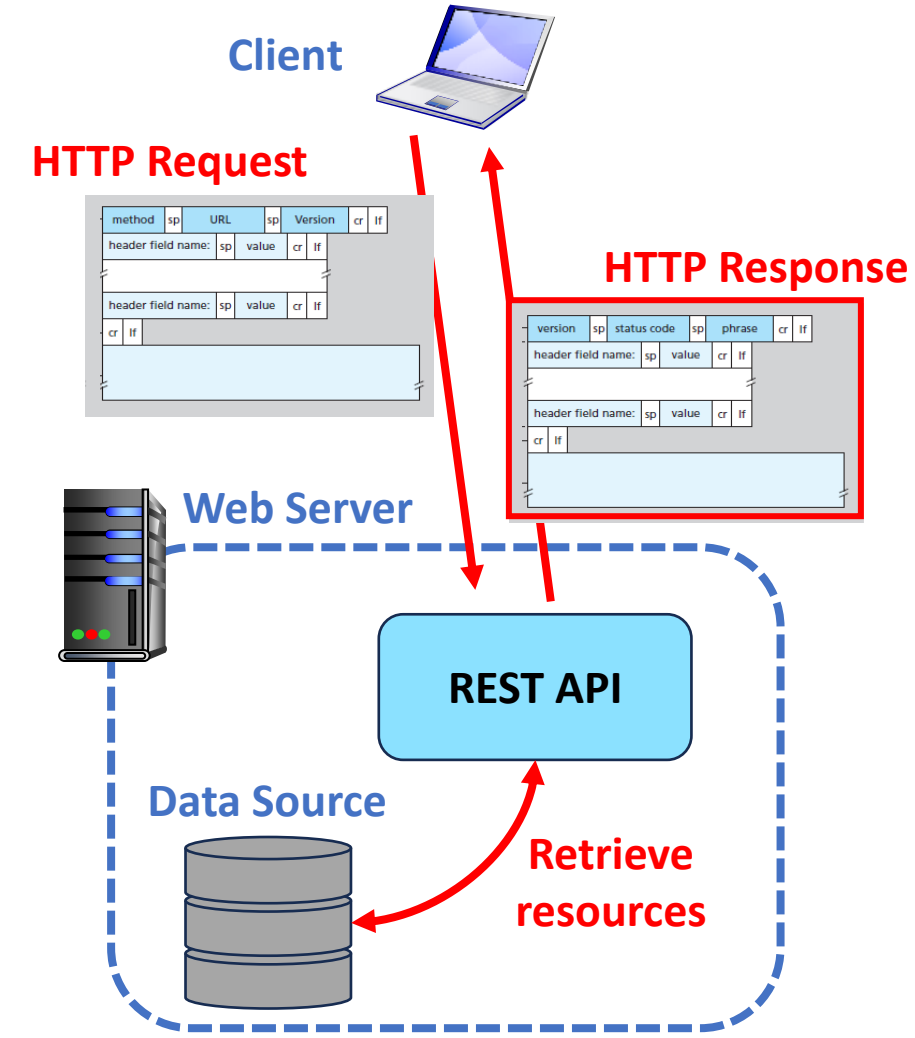
- HTTP **requests** in REST:
  - Resources can be **identified** through the **[URL]** field of the request.
  - The **operations on resources** can be specified into the **[Method]** field of a request.
    - **CRUD** (Create, Read, Update, Delete) **operations are typically considered**, which can be implemented through the 4 HTTP methods:
      - **Get**: retrieve the resource (read).
      - **Post**: modify the resource (update).
      - **Put**: add a new resource (create).
      - **Delete**: remove a resource (delete).
  - Possible **resource and options** can be specified into the **[Body] and [Header]** respectively.



Network Programming

REST API in Practice

- HTTP **response** in REST:
  - The **information about the request** are provided via the **[Status code]** and **[Phrase]** fields. Commonly used messages are:
    - 200 for successfully handled resource,
    - 201 for successful creation of a new resource,
    - 404 for resource not found,
    - 405 for method not supported,
    - etc.
  - Possible **resource and options** can be specified into the **[Body] and [Header]** respectively.



Network Programming

Resource Representation

- **HTTP URL is used to identify resources**. Common formats are:
  - [http(s)]://[Domain name of REST API][:Port]/[API version]/[Path to resource]
    - Example: http://myapi.example.com/v1/users/1
  - [http(s)]://[Domain name][:Port]/[REST API]/[API version]/[Path to resource]
    - Example: http://example.com/myapi/v1/users/1

- Paths for **resources and sub-resources** are commonly specified using the following pattern:
  - /[resource name]/[resource id]/[sub-resource name]/[sub-resource id] ..
    - Example: /users/1/nickname

- Resources can be **represented in different data exchange formats**. Most common is JSON, but **multiple languages can be used contemporary**.
  - For example, selected **format can be specified** by the “Content-type” header line.

Network Programming

REST Example (Java)

Outsourced…

Network Programming

Guidelines

- Resource naming and usage **guidelines**:
  - Use **plural nouns not verbs**:
    - Good: /users
    - Bad: /doSomethingOnUsers
  - Use **HTTP methods instead of CRUD terms** (or similar) into names:
    - Good: /users (with GET method)
    - Bad: /getAllUsers
  - Identify **unique resources** with IDs:
    - Good: /users/1
    - Bad: /users?id=1
  - You may use **queries** to select or sort resources:
    - Good: /users?nickname=Bob&sort=age
    - Bad: /usersNamedBobSortedByAge