



UNIVERSITÀ
DI TORINO

Communication using Json

Prof. Fabio Ciravegna
Dipartimento di Informatica
Università di Torino
fabio.ciravegna@unito.it

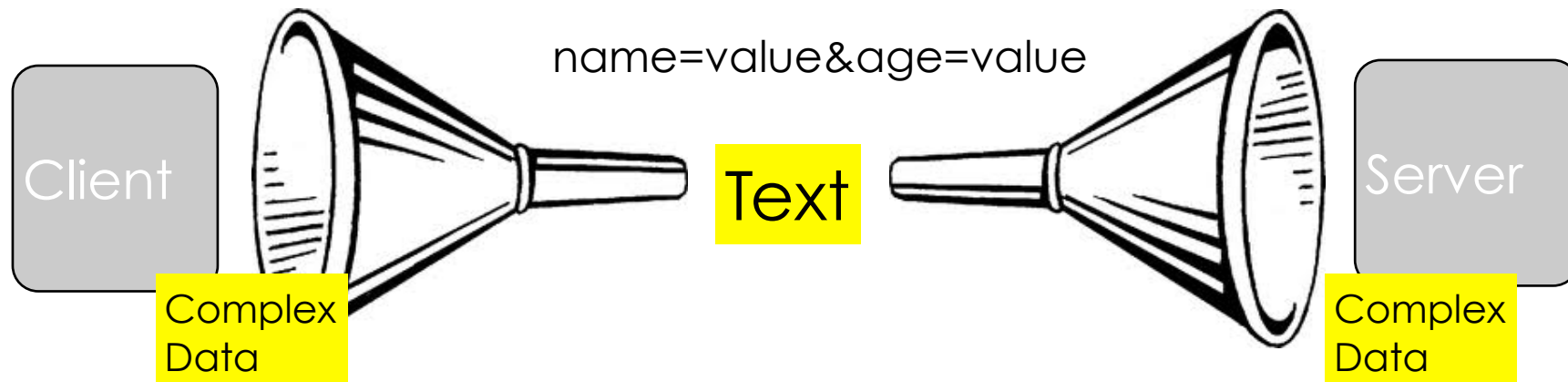
<http://json.org/>




Client-Server communication


- Different environments
 - E.g. JavaScript on browser and server
 - Difficult communication
- Solution?
 - Serialisation/de-serialisation of data into text
 - e.g. used in HTML forms

This approach is really cumbersome




Example: National Rail Enquiry <http://www.nationalrail.co.uk/>

**National Rail Enquiries**

 Register now for instant access to your favourite journeys
Already registered? [Sign in now.](#) [Let's go!](#)

[Home](#) [Train times & tickets](#) [Stations & on train](#) [Changes to train times](#) [Hotels](#) [Search](#) [YAHOO!](#)


 [Travel news](#) > Prepare for winter weather with our information feeds

Find my train times & fares


From to

Leaving


:




at :



21/03/2012




at :





21/03/2012

[GO](#)

 Remove return journey

[Advanced search](#)

 Passengers: 1 Adult

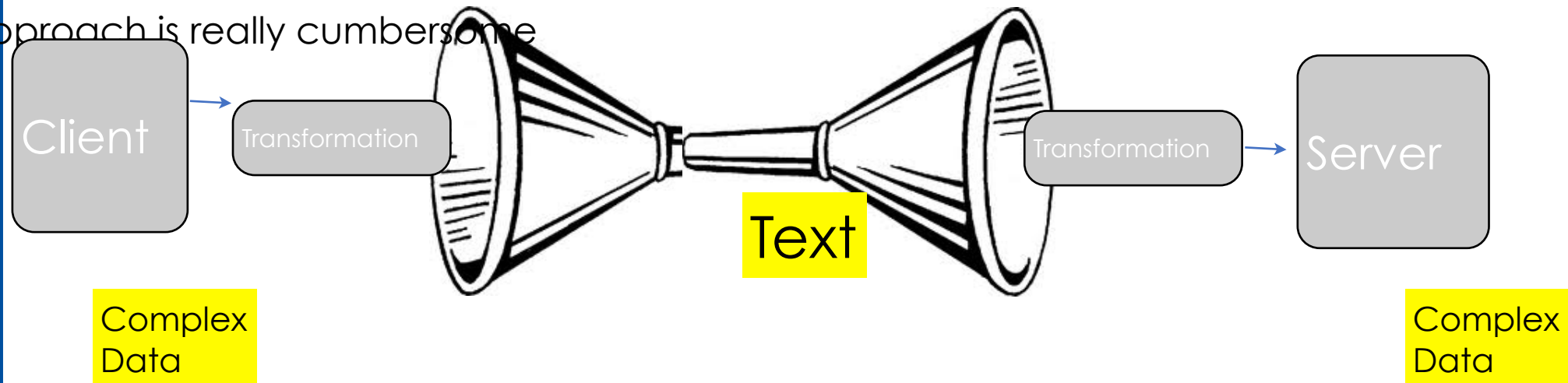
☐ Show only fastest trains 

3

Traditional solution

- Client and server exchange
 - Client to Server (POST)
 - `fromStation="Shf" &toStation="MncAir" &dateOut="Today"&hourOut="9"&minOur="15" &dateIn="21/03/2012"&...`
 - Server to Client: HTML file rendered with parameters

This approach is really cumbersome



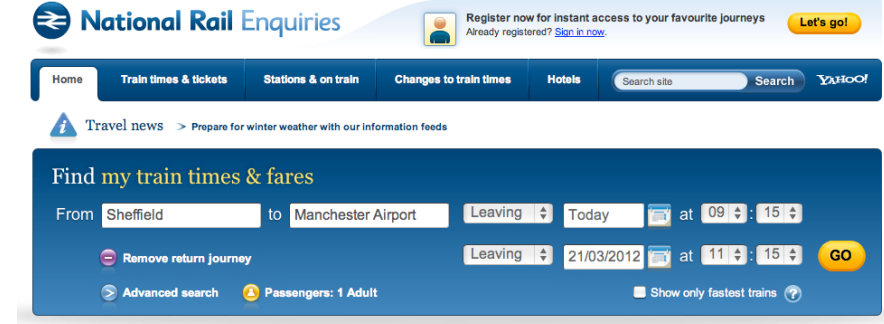
JSON

<http://www.json.org/>

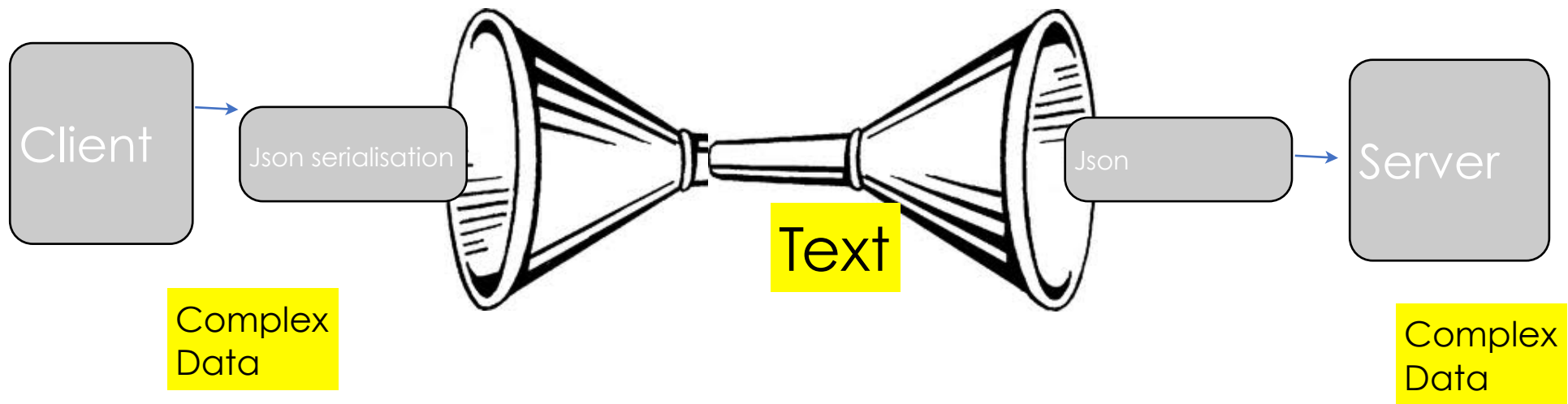
- JSON (JavaScript Object Notation) is a lightweight data-interchange format.
 - It is easy for humans to read and write. It is easy for machines to parse and generate.
 - It is based on a subset of the [JavaScript Programming Language](#)
 - JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others.
 - These properties make JSON an ideal data-interchange language.

JSON (2)

- JSON is built on two structures:
 - A collection of name/value pairs. In various languages, this is realized as an
 - object, record, struct, dictionary, hash table, keyed list, or associative array
 - An ordered list of values. In most languages, this is realized as an
 - array, vector, list, or sequence.
 - These are universal data structures. Virtually all modern programming languages support them in one form or another. It makes sense that a data format that is interchangeable with programming languages also be based on these structures.



- Client and server exchange
 - Client to Server: JSON
 - Server to Client: JSON (or HTML code)



We do not need to find a clever way to encode the data structure. Json does it for you

```
{ fromStation: 'Shf', toStation: 'MncAir', dateOut: 'Today', hourOut: '9', minOur: '15',
  dateIn: '21/03/2012' }
```


Sender Side (Javascript)

- Serialisation

- in the context of data storage and transmission, serialization is the process of converting a data structure or object state into a format that can be stored (for example (...)) transmitted across a network connection link) and "resurrected" later in the same or another computer environment (wikipedia)

```
var myJSONText = JSON.stringify(myObject, replacer);
```


Receiving side (Javascript)

<http://www.json.org/js.html>

- Deserialisation
 - The opposite operation: extracting a data structure from a series of bytes

```
var people = JSON.parse(myJSONText) ;  
>>> people:  
[{name: "joe",    height: 185, weight: 79},  
  {name: "paula", height: 175, weight: 59}]
```

Stringify additional param

- The `stringify` method can take an optional replacer function.
 - called after the `toJSON` method on each of the values in the structure.
 - It will be passed each key and value as parameters, and this will be bound to object holding the key.
 - The value returned will be stringified.

```
function replacer(key, value) {  
    if (typeof value === 'number' && !isFinite(value)) {  
        return String(value);  
    }  
    return value;  
}
```

Receiving side (Javascript)

<http://www.json.org/js.html>

- Deserialisation

- The opposite operation: extracting a data structure from a series of bytes

```
var myObject = JSON.parse(myJSONtext, reviver);
```

- The optional *reviver* function will be called for every key and value at every level of the final result.
 - Each value will be replaced by the result of the *reviver* function.
 - This can be used e.g. to transform date strings into Date objects.

Why use it?

- Client server architecture can now return data structures as opposed to strings or HTML code
 - Client can send complex objects (as opposed to just variable-value pairs)
 - Client is no longer passive: now it interprets the code and displays it as required



How to return JSON data

On a server's route:

```
router.get('/index', function (req, res, next) {
```

```
  res.setHeader('Content-Type', 'application/json');  
  res.send(JSON.stringify({ a: 1 }));  
}
```

declare that you are returning JSON
in header

Stringify the data before sending back

For the client side we need instead to send and receive JSON instead of just getting files and posting forms

we will use AJAX/Axios (soon screened here)



UNIVERSITÀ
DI TORINO

Heads Up: JSON in Java

(Gson)



GSON

A Google library for JSON in Java

- Gson is a Java library that can be used to convert Java Objects into their JSON representation. It can also be used to convert a JSON string to an equivalent Java object. Gson can work with arbitrary Java objects including pre-existing objects that you do not have source-code of
 - Download the gson library in order to use it (it is not in the standard java distribution)

<http://code.google.com/p/google-gson/>

Serialisation

- Serialisation:

```
/* create Gson object */
Gson gson = new Gson();
/* create the object to serialise (any Java object)*/
class BagOfPrimitives {
    private int value1 = 1;
    private String value2 = "abc";
    private transient int value3 = 3;
    BagOfPrimitives() {
        // no-args constructor
    }
}
BagOfPrimitives obj = new BagOfPrimitives();
String json = gson.toJson(obj);
```

Deserialisation

```
BagOfPrimitives obj2 =  
    gson.fromJson(jsonString,  
        BagOfPrimitives.class);
```

Expected Object class



UNIVERSITÀ
DI TORINO

Questions?

