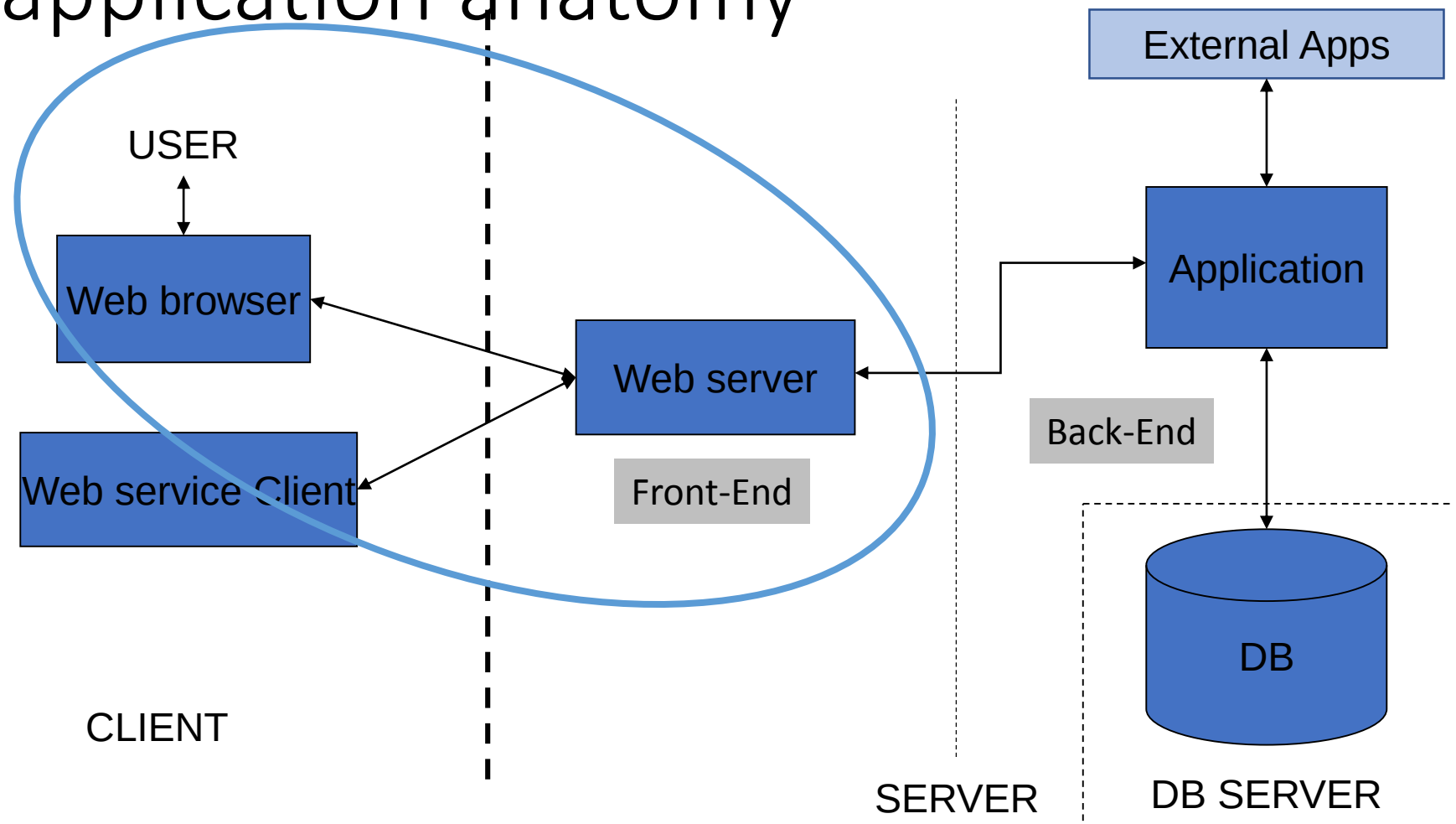
Web & Web applications

DBW

Outline

- Web basics
  - HTTP servers and browsers
  - Languages
  - Software
- Concept and types of Web applications. Web services
- Languages involved
- CGI Protocol.
- Time issues
- Personalization. Cookies & session persistence
- Hints

Web application anatomy



Web (HTTP) Servers

- Computer applications that listen to a TCP port (80 typically), and understand HTTP requests.
- Information served are text or binary files (*resources*) stored locally in the server.
- HTTP servers that implements the appropriate protocols, can run **server-side applications** according to the request.
- Example SimpleHTTPServer (Perl, Python)

Web (HTTP) clients (browsers, …)

- Applications making requests to a server at a given TCP port (typically 80) using HTTP protocol

- Simple (command-line) browsers request for files (using HTTP) in a similar way to FTP. Resources are identified by URL (i.e. wget, curl)

- Normal browsers “understand” the contents of the obtained files and combine information from one or more servers interpreting a given language (usually HTML/CSS) in graphical output.
  - Most **browsers can execute applications** (client-side) obtained from the information server
  - Modern Web sites rely heavily on client-side applications to deliver dynamics content (mostly with Javascript) → “single-page applications”

Languages involved (ordinary web)

- HTML: Contents management language
  - Defines contents and structure of the page, includes the necessary links to all elements
  - Tag formatted language `(…<p>Some text</p>…)`

- CSS: Formatting language
  - Defines how the contents is represented in the user browsers
  - `P {font-family:Times; font-family: 10pt; display:block; background-color:black}`
- Javascript
  - Used for client-side applications
  - Plain or in richer variants like Typescript, and frameworks like Angular, React, …

Languages involved (web services)

- Data exchange formats
  - XML: Most traditionally used by web applications
    - Same structure as HTML, but with no fixed tags
    - Requires XML-schema to specify tags and check coherence

```
<Course id="DBW">
<Acronym>DBW</Acronym>
<Title>Databases and Web applications</Title>
</Course>
```

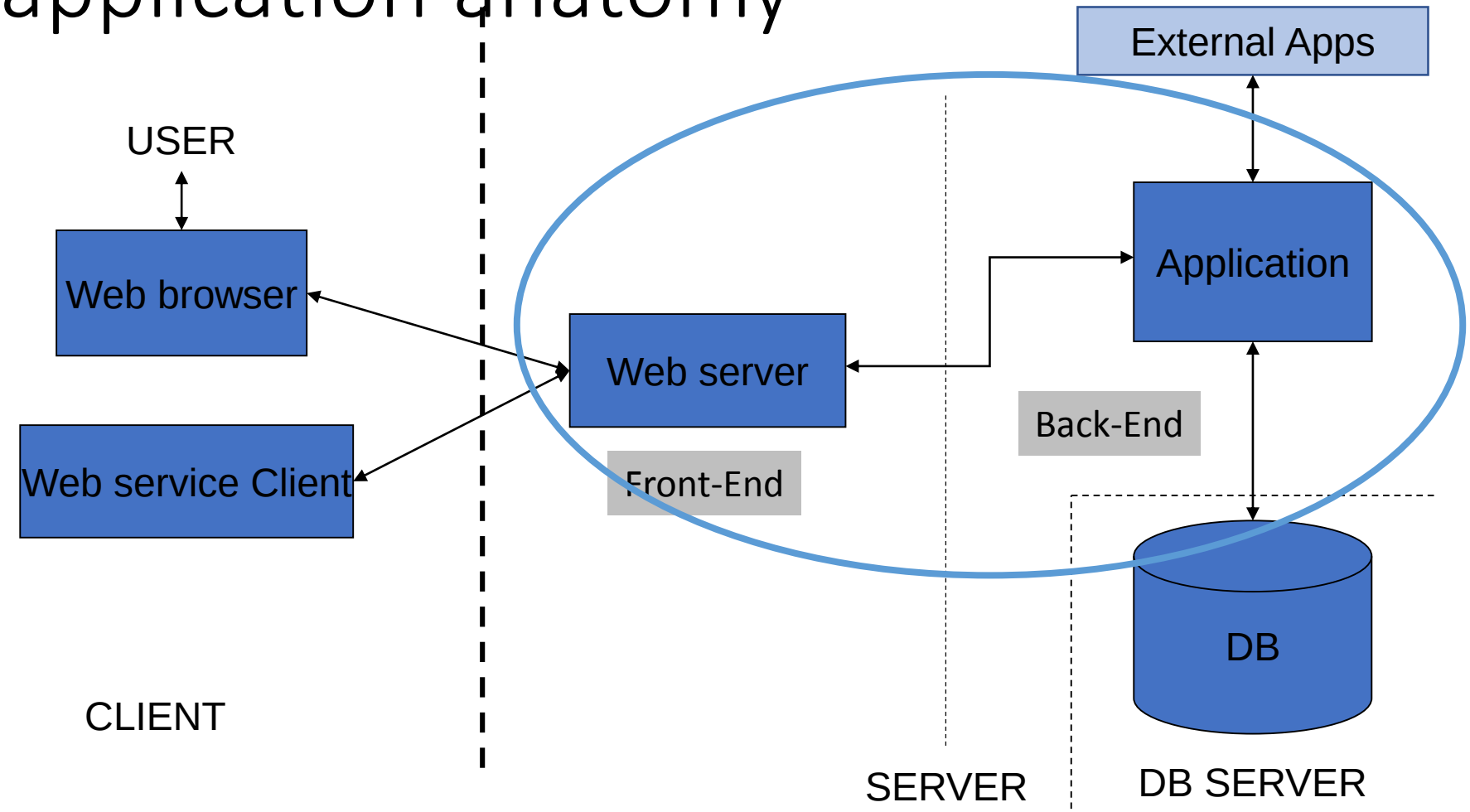
  - JSON: Data interchange format replacing XML
    - Natively understood by Javascript, and of increasing popularity

```
Course: {Acronym: 'DBW', Title: 'Databases and Web applications'}
```

Software to build Web pages

- A simple text editor is enough (Notepad, vi, nano, ...)
- Syntax checking editors are more useful (vim, gedit, ...)
- WYSWYG editors are common (Dreamweaver, Openoffice...). However, they MUST allow to check HTML manually!
- Content Manager Systems (CMSs)
  - Integrated environments to build web sites, general or specialized
  - Can include some useful functionality (user management, email, …)
  - Very useful to build static sites, but difficult to include applications
    - However, web structure and layout made by the CMS can be used
  - Drupal, Joomla, Wordpress, …
- Pre-build environments (CSS & JS)
  - Jquery, Bootstrap, …

Web application anatomy



Definition & types

- A Web application is a dynamic extension of a Web server.
  - Adapts to user input
  - Can serve non-static information (generated in real-time)
  - Uses standard protocols (HTTP, SMTP)
  - Users interact with the application mainly using Web browsers

- Presentation-oriented
  - Generates dynamic Web pages (HTML/CSS/JS) responding to user queries
  - Usual way to provide bioinformatics results

- Service-oriented
  - Interacts with other applications (XML/SOAP, REST)
  - Allows to build automatic workflows for complex analyses

Client side

- Application must be compatible with standard web browsers
  - HTTP protocol: GET, POST, (PUT)
- User input comes from URL's or HTML forms
- Output must be in known languages: HTML, CSS, Javascript, XML, JSON
- Output may invoke other programs (plug-ins) though MIME
  - Almost obsolete, fully replaced by HTML v5

Client side

- HTML v5 include a variety of native functionalities
  - Audio/video, SVG graphics, MathML, Geolocalization, parallel process, …

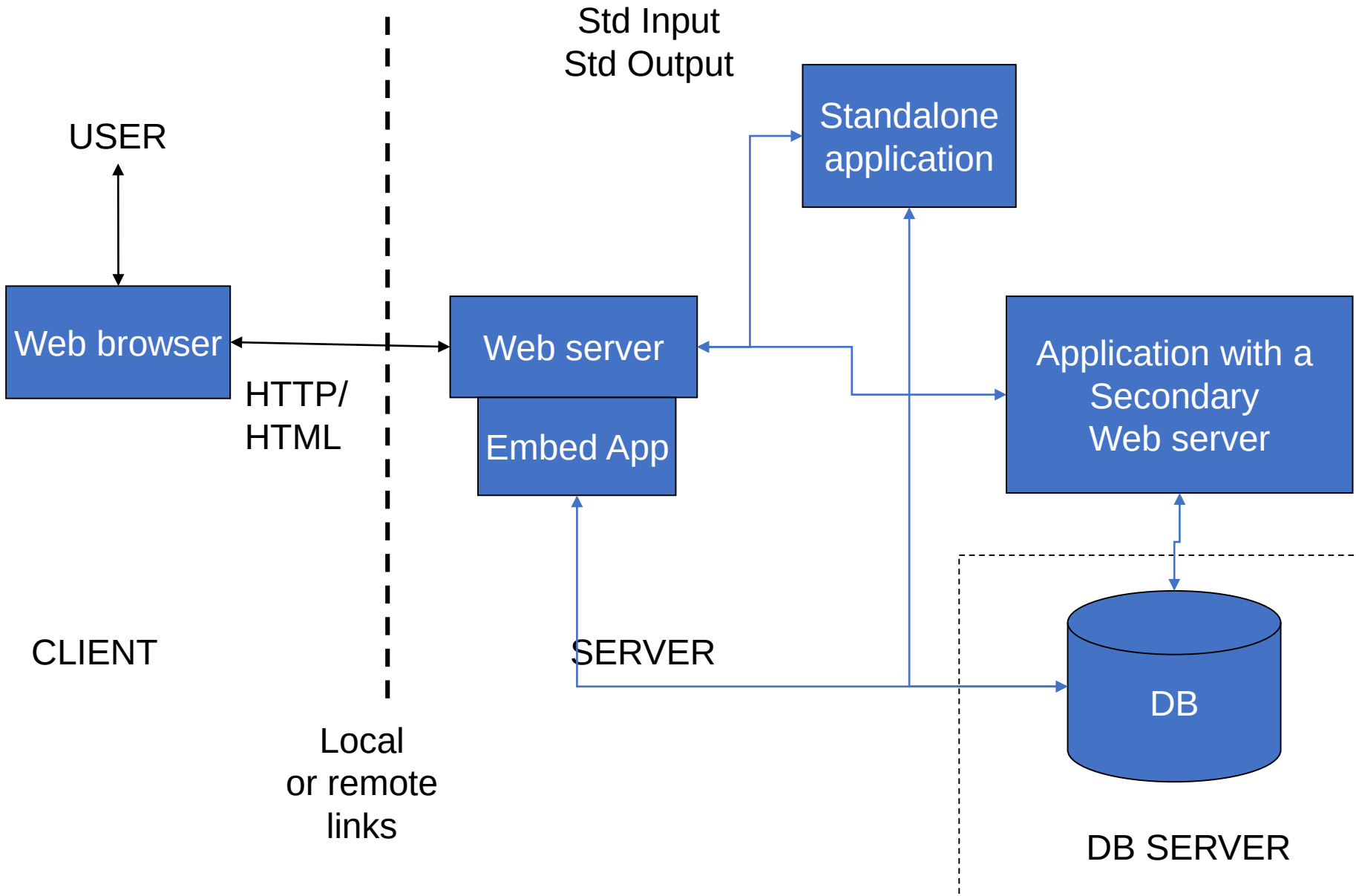
- Modern browsers are able to run client-side apps (Java applets, and Javascript)
  - Client-side applications are fully qualified applications, served as static files, and run in the browser
  - Java applets are almost obsolete (still seen in bioinformatics) due to security issues
  - Javascript is behind dynamic behaviour of modern web sites.
    - Asynchronous interaction with server (new request do not require reload)
    - JsMol, NGL
    - Jquery, AngularJS ReactJS, VueJS, …
    - Component libraries for bioinformatics start to be available (https://ebi-webcomponents.github.io/nightingale/#/)

Server side

- An application is invoked by web server on receiving the request
  - External application (CGI)
    - Executable running in the server machine.
      Can be written in any language.
    - Get input from standard input and writes in the standard output. Web server redirects both.

  - Server embedded.
    - Web server is able to execute the application as a child process (may require a driver)
    - Usual languages: Python, NodeJS, Perl, Java
    - “Web oriented” languages: PHP, ASP, .NET, JSP

  - Java, python, JS applications require special servers
    - Installed normally as a secondary web server (port != 80)
    - The main server acts as a “proxy”

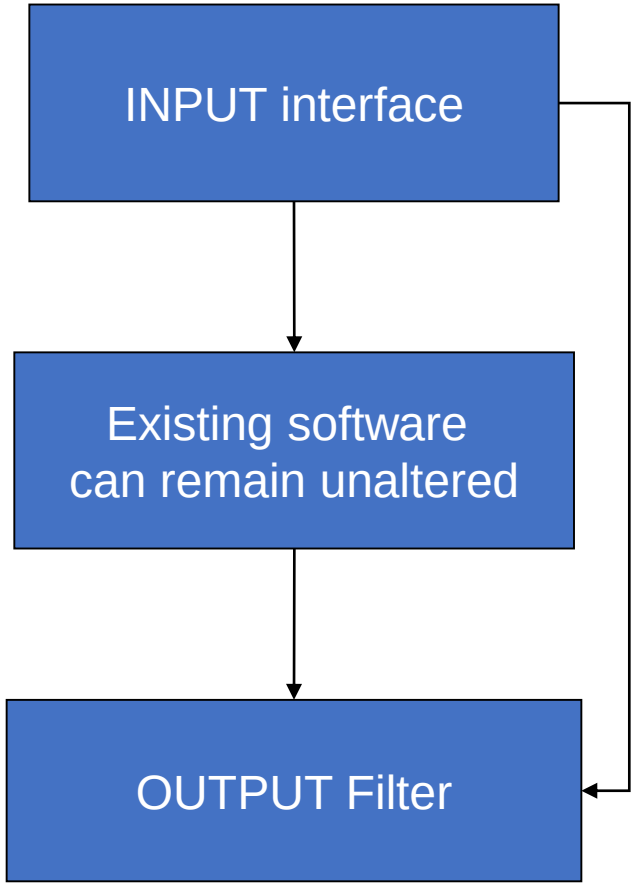

Std Input
Std Output
USER
Standalone application
Web browser
Web server
Application with a Secondary Web server
HTTP/
HTML
Embed App
CLIENT
SERVER
DB
Local or remote links
DB SERVER

CGI Protocol & strategies

- **Common Gateway Interface (CGI)**
  - **Formal interface** between Web server and external applications
  - CGI interface provides
    - Environment variables including information from the browser-server conversation
    - POST input data, as standard input
    - Redirection of application standard output & error to Web stream.
- **For External applications**
  - Read Input information from Environment variables, and standard input
  - Provides results and error as standard output
  - Are executed by the operative system as usual command-line executables (security issues!!)
- **For Embedded applications**
  - Still use input and output standard and CGI variables, but data is processed internally in a web server's subprocess

- Practical CGI use
  - Languages have specific extensions to deal (Hide) CGI from the development process
    - Input GET and POST, and CGI variables available at predefined variables/objects
    - No need to process HTTP
      - Cookies, authentication, etc.
  - Web programming frameworks
    - Slim, Symphony (PHP), Flask, Django (Python)
    - Programming helpers acting as an interface between CGI and the programming language
    - Provide "easy" web applications

The simplest application



HTML Forms
Data validation
Error handling

Output to HTML. EMail

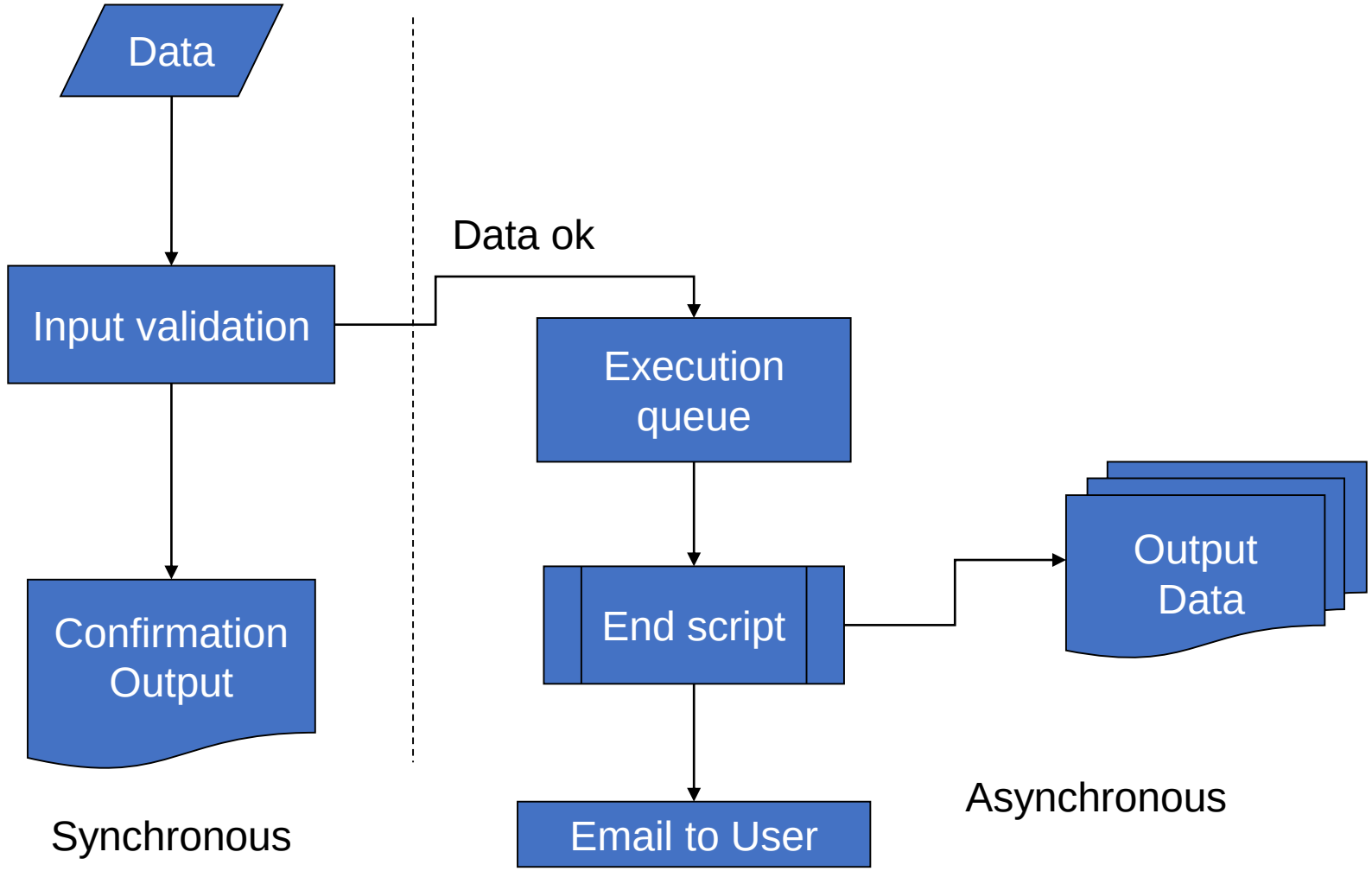
Known issues in web applications

- Time issues
  - Web users require “instant” responses
  - Most web browsers (and network helpers) may have a short “timeout”
  - Application that lasts more that 1-2 mins must be asynchronous

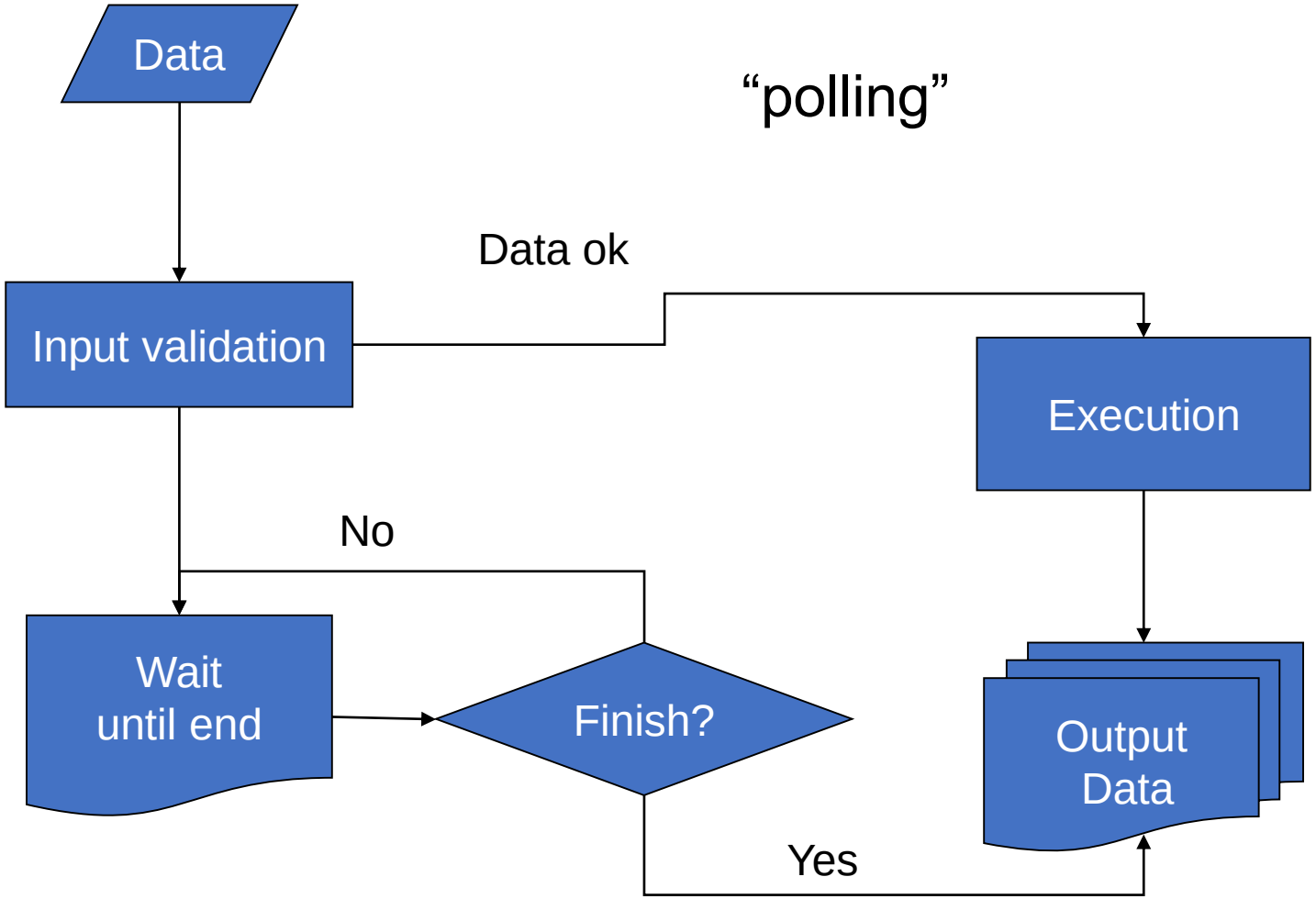
- Persistence, and User recognition
  - HTTP protocol is not persistent: Connection closes short time (~20s) after the server answers

  - Applications need to recognize returning users
    - Authentication (user only must write the login/password once)
    - Keep personal preferences, and private data
    - Grant access to given resources according to previous requests
    - Avoid requesting known data more than once
    - Avoid “reloads”

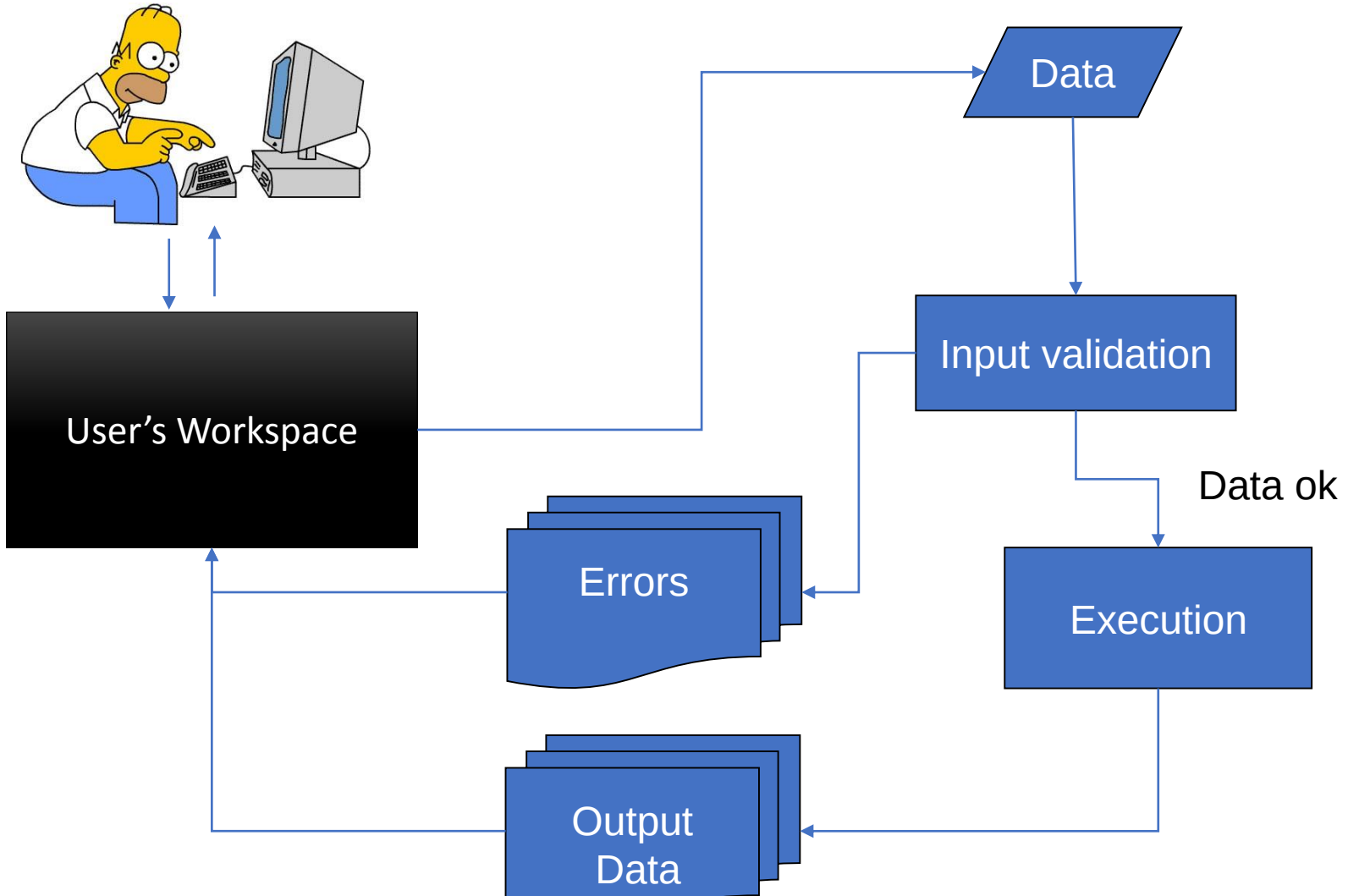
Time issues. Usual strategies (1)



Time issues. Usual strategies (2)

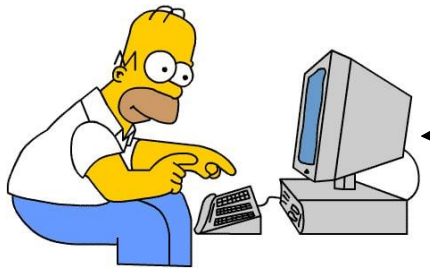


Time issues. Usual strategies (3)



Personalization (Authentication/Authorization)

Authentication: Identify Users
Authorization: Manage users' privileges



Web browser

Web server

Application

OpenID Connect

Identity providers

Third party AA

User/Session DB

CLIENT

SERVER

A DB stores history of user connections and activities

Authentication/Authorization schemes

- **Server based authentication**
  - Based on unix-like passwd files (login / passwd)
  - Protects folders and sub-folders (.htaccess files)
  - Identity via CGI variable (REMOTE_USER)
  - May require access to server configuration
  - Environment managed by local DB

- **Application based (both authentication and authorization)**
  - Do not require access to server configuration
  - Authentication and environment managed by the application itself (via local DB)
  - Full control from the application (login / passwd, SSL Keys, …)
  - Persistence via Cookies or language specific constructs (PHPSessionID / Session)

- **Third party authentication (single sign-on, SSO)**
  - Authentication is done by identity providers (Google, openID, ELIXIR, ….), or other apps (eGroupWare, Drupal, …)
  - **Users can have a single point for authentication (SSO)**
  - Protocols
    - OAuth, OpenID Connect
  - Bioinformatics world
    - Bonda fide Researchers
    - European Life Sciences Id / ELIXIR

User identification from the application: cookies

- Small amount of text information stored by the server in the users' web browser as key /value pairs
- Do not require user/password (user do not need to be aware of)
- Limited to 4Kb
- Retrieved automatically as part of CGI handshake
- Cookies do not identify persons but browsers!!

- ID: a unique ID generated by the server
- Origin: server URL. Browsers send back cookies to the servers that created them (no other servers can get the data)
- Expiration date. Cookies can last for a single session or till a specified date

Web application layout hints

- Static contents (text, images, etc.) stored as normal web resources
  - For optimization, some servers keep them separated from scripts
- Dynamic pages managed by server scripts
  - No general rule, depends on language and programming style
  - The easy way: Each different screen is managed by a specific script.
  - Web frameworks use normally a “routing” mechanism to associate code to incoming URLs
- Global variables
  - Each script acts in a separated HTTP transaction!
  - All scripts should load the same global environment, usually included from a single file
- Protected/public data
  - Protected data should be stored outside of the web directory tree, and be accessed only programmatically
  - Output data may be placed inside the web tree if it is already HTML/CSS

Web application layout hints

- Temporary data
  - Can be stored anywhere
  - Most languages provide automatic temporary directories and file names.
  - Be aware that applications can be executed by the “web” user, check permissions.
  - Should be deleted after use!!
- Beware of multiple concurrent users
  - Use request-specific file names for temporary data and results
  - Use user-based or process-based directories
  - Think in a queueing system for lengthy operations
- Collect statistics of use!!