State Management
Asynchronous Processing

Cookies and Sessions

Writing Concurrent Code in C#

SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.bg

Have a Question?

sli.do

#csharp-web

Table of Contents

HTTP Cookies

Usages and Control

What Are Cookies?

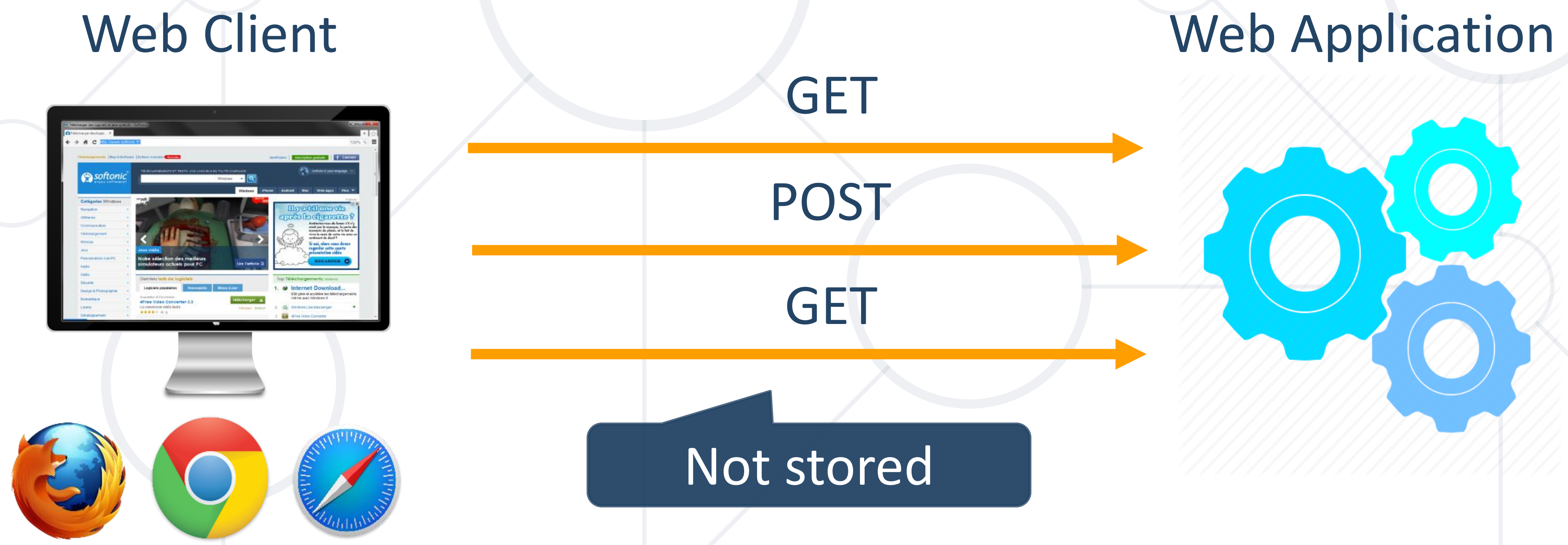
- A small file of plain text with no **executable code**
  - Sent by the server to the client's browser
  - **Stored** by the browser on the **client's device** (computer, tablet, etc.)
  - Hold small piece of data for a **particular client** and a web site

What Are Cookies Used for?

- Session management
  - Logins, shopping carts, game scores or anything else the server should remember
- Personalization
  - User preferences, themes and other custom settings
- Tracking
  - Recording and analyzing user behavior
- Breakfast
  - But that's not what we are currently talking about

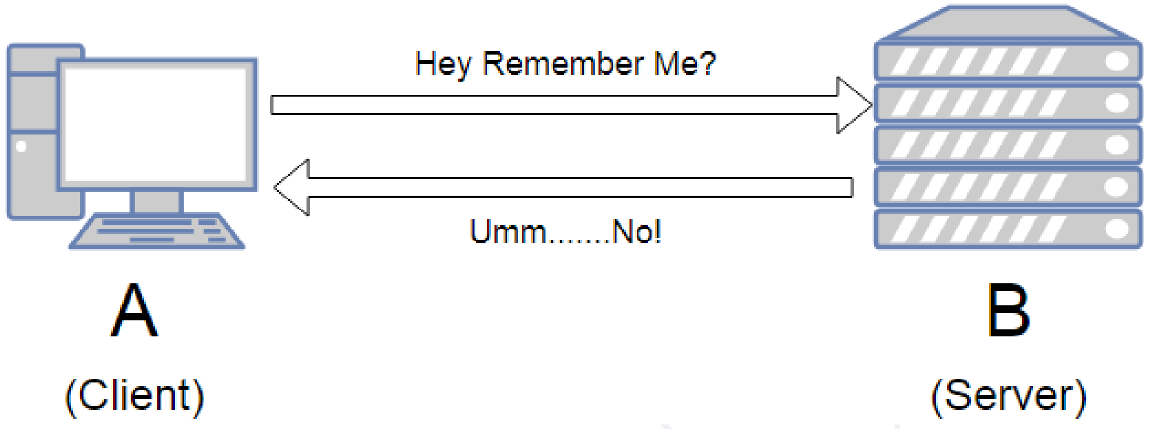
Session Management

- The HTTP object is **stateless**
  - It **doesn't store** information about the requests



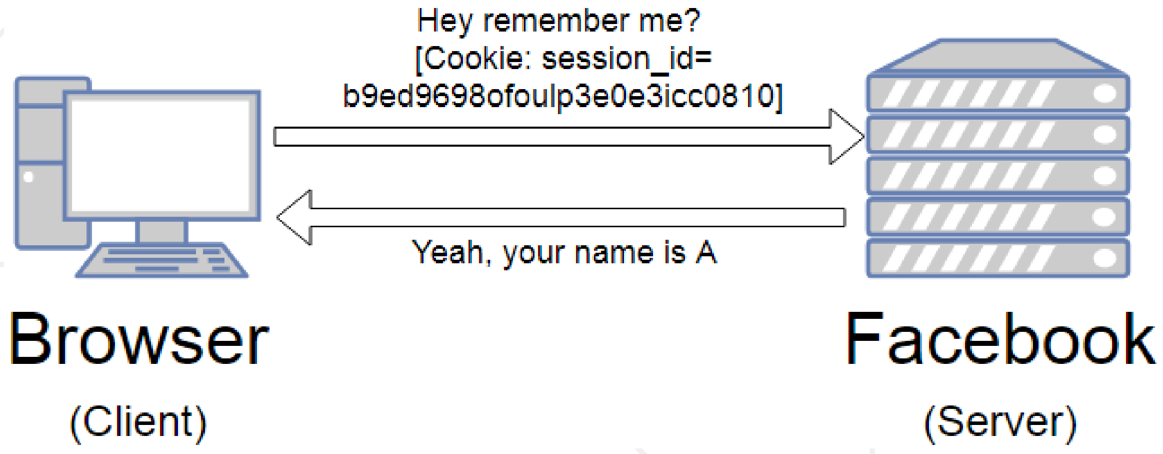
Stateless HTTP – the Problem

- The server **doesn't know** if two requests come from the same **client**
- State management problems
  - **Navigation** through pages requires **authentication** each time
  - **Information** about the pages is lost between the **requests**
  - Harder **personalization** of functionality of pages



Stateless HTTP – the Cookie Solution

- A reliable mechanism for websites to remember **stateful information**
  - To know whether the **user** is **logged** in or **not**
  - To know which account the **user is logged** in with
  - To record the user's **browsing activity**
  - To **remember** pieces of information **previously** entered into form fields (usernames, passwords, etc.)



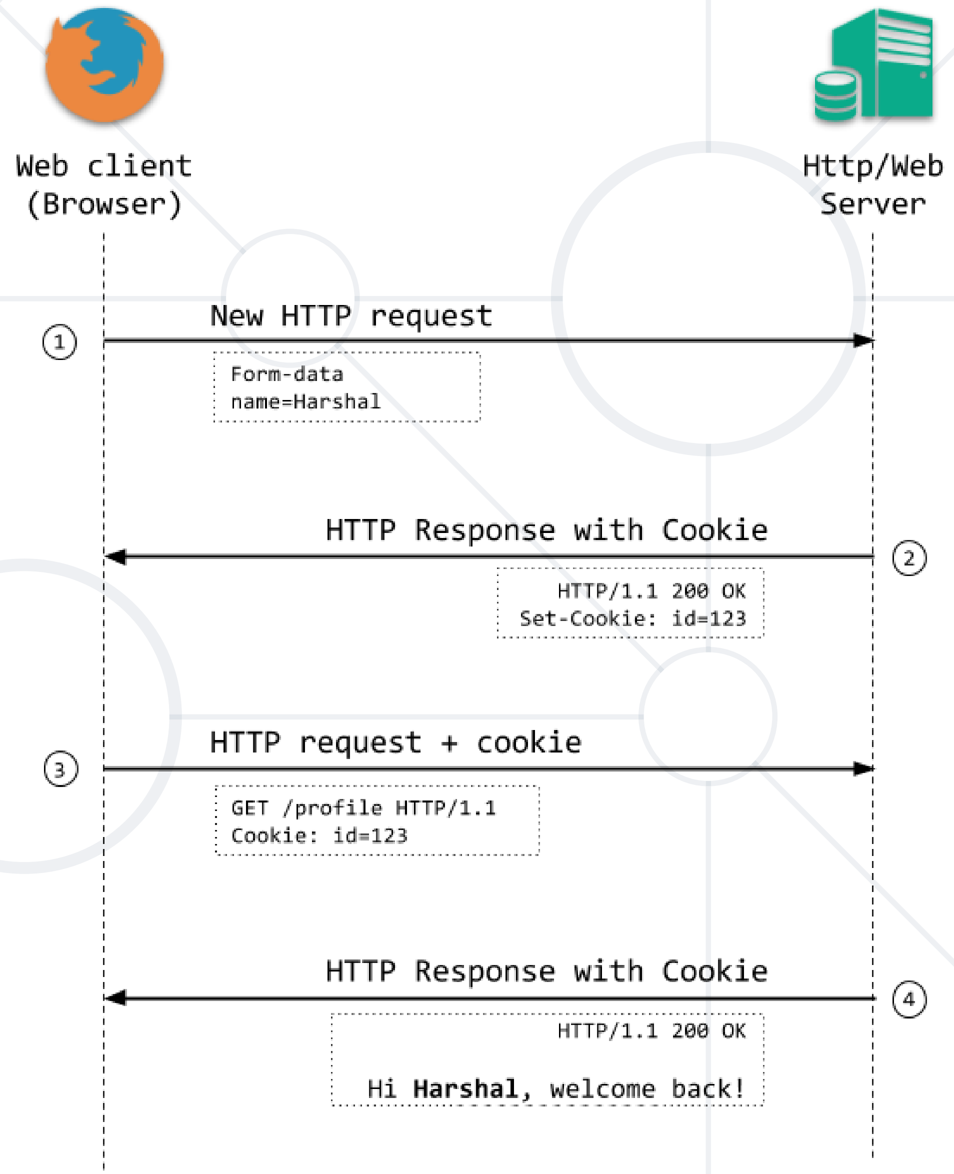
How Are Cookies Used?

- The response holds the cookies to be saved within the **Set-Cookie** header

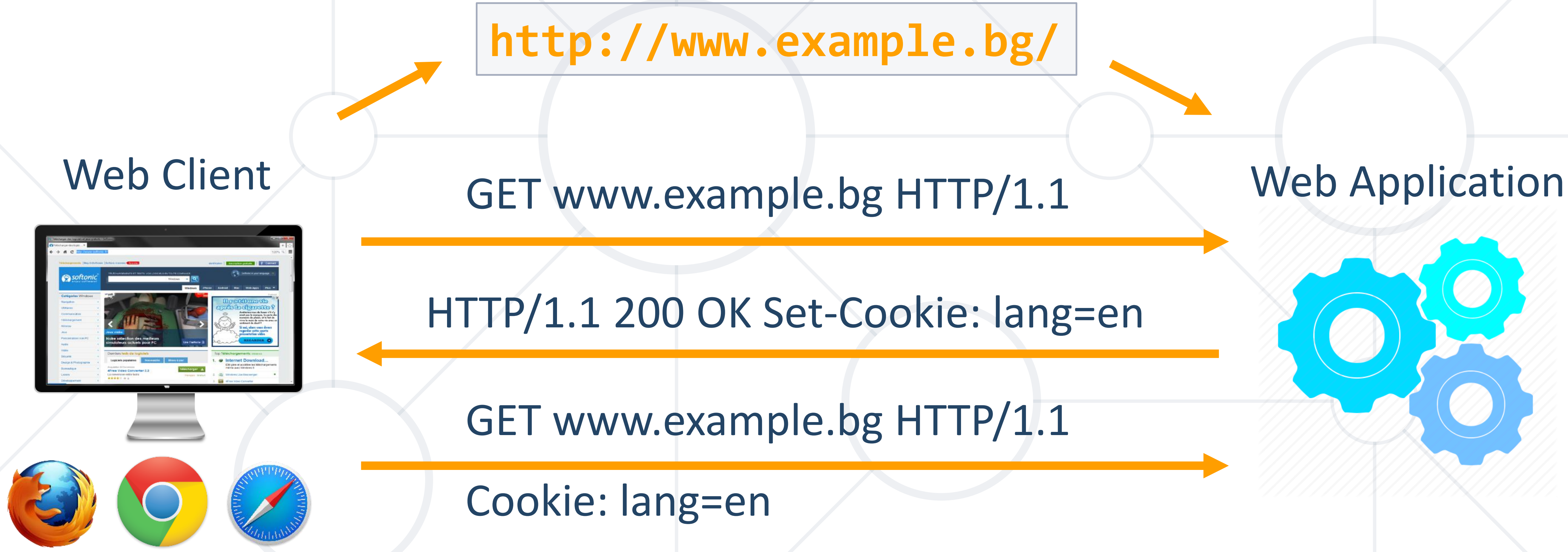
```
HTTP/1.1 200 OK
Set-Cookie: lang=en
```

- The request holds the specific web site cookie within the **Cookie** header

```
GET www.example.bg HTTP/1.1
Cookie: lang=en
```



Server-Client Cookies Exchange


http://www.example.bg/
Web Client
Web Application
GET www.example.bg HTTP/1.1
HTTP/1.1 200 OK Set-Cookie: lang=en
GET www.example.bg HTTP/1.1
Cookie: lang=en

Cookie Structure

- The cookie consists of **Name**, **Value** and **Attributes** (optional)
- The attributes are **key-value pairs** with additional information
- Attributes are **not included** in the requests
- Attributes are used by **the client** to control the **cookies**

Name=Value

Attributes

```
Set-Cookie: SSID=Ap4P…GTEq; Domain=foo.com; Path=/;
Expires=Wed, 13 Jan 2021 22:23:01 GMT; Secure; HttpOnly
```

Scope

- Defined by the attributes **Domain** and **Path**
- **Domain** – defines the website that the cookie belongs to
- **Path** – Indicates a **URL** path that must exist in the requested resource before sending the **Cookie** header

```
Set-Cookie: SSID=Ap4P…GTEq; Domain=foo.com; Path=/;
Expires=Wed, 13 Jan 2021 22:23:01 GMT; Secure; HttpOnly
```

Lifetime

- Defined by the attributes **Expires** and **Max-Age**
- **Expires** – defines the date the browser should delete the cookie
  - By default the cookies are deleted after the end of the session
- **Max-Age** – interval of seconds before the cookie is deleted

```
Set-Cookie: SSID=Ap4P…GTEq; Domain=foo.com; Path=/;
Expires=Wed, 13 Jan 2021 22:23:01 GMT; Secure; HttpOnly
```

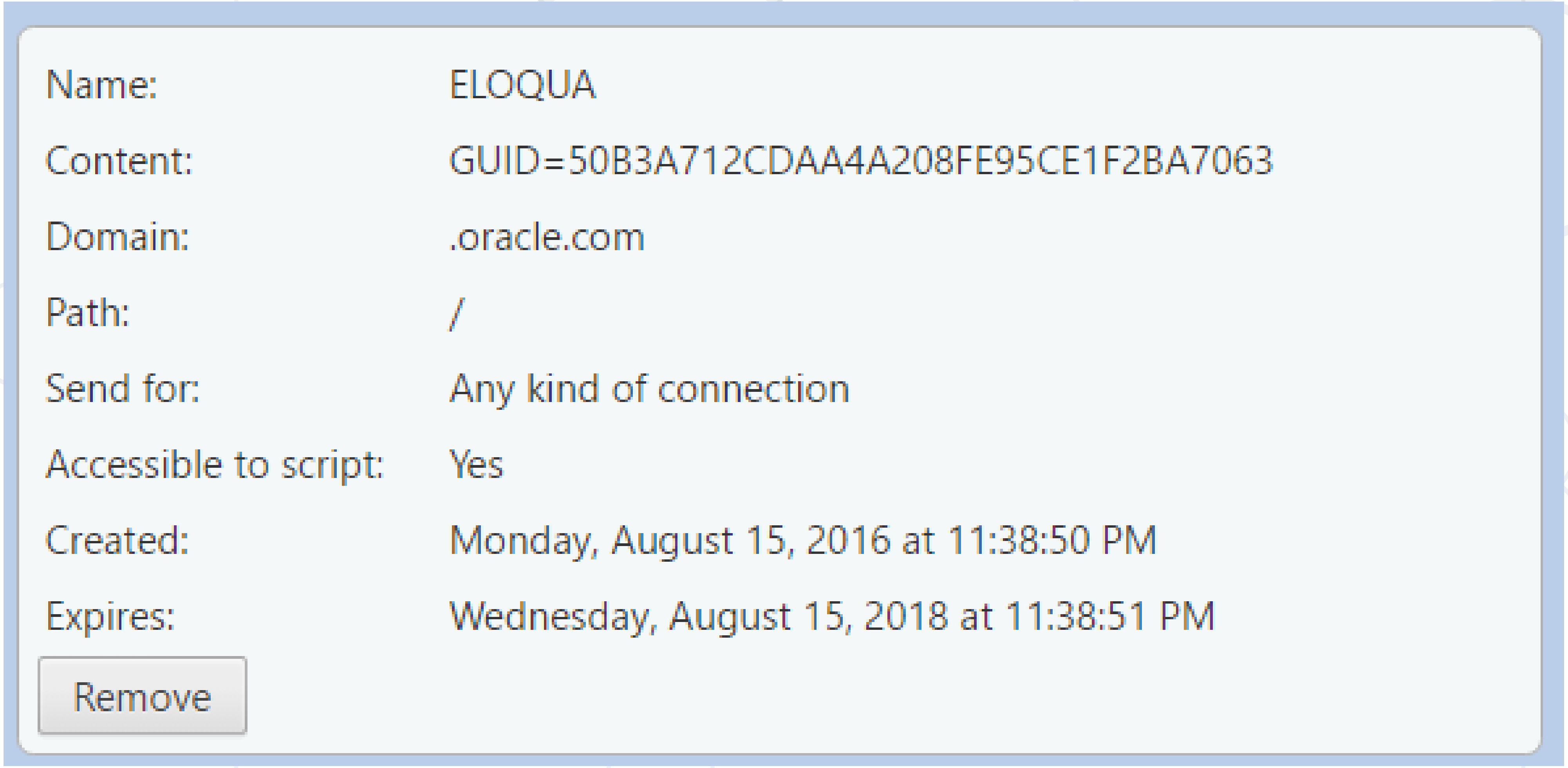
Security

- Security flags do not have associated values
- **Secure** - tells the browser to use cookies only via **secure/encrypted** connections
- **HttpOnly** – defines that the cookie cannot be accessed via client-side scripting languages

```
Set-Cookie: SSID=Ap4P…GTEq; Domain=foo.com; Path=/;
Expires=Wed, 13 Jan 2021 22:23:01 GMT; Secure; HttpOnly
```

What is in the Cookie?

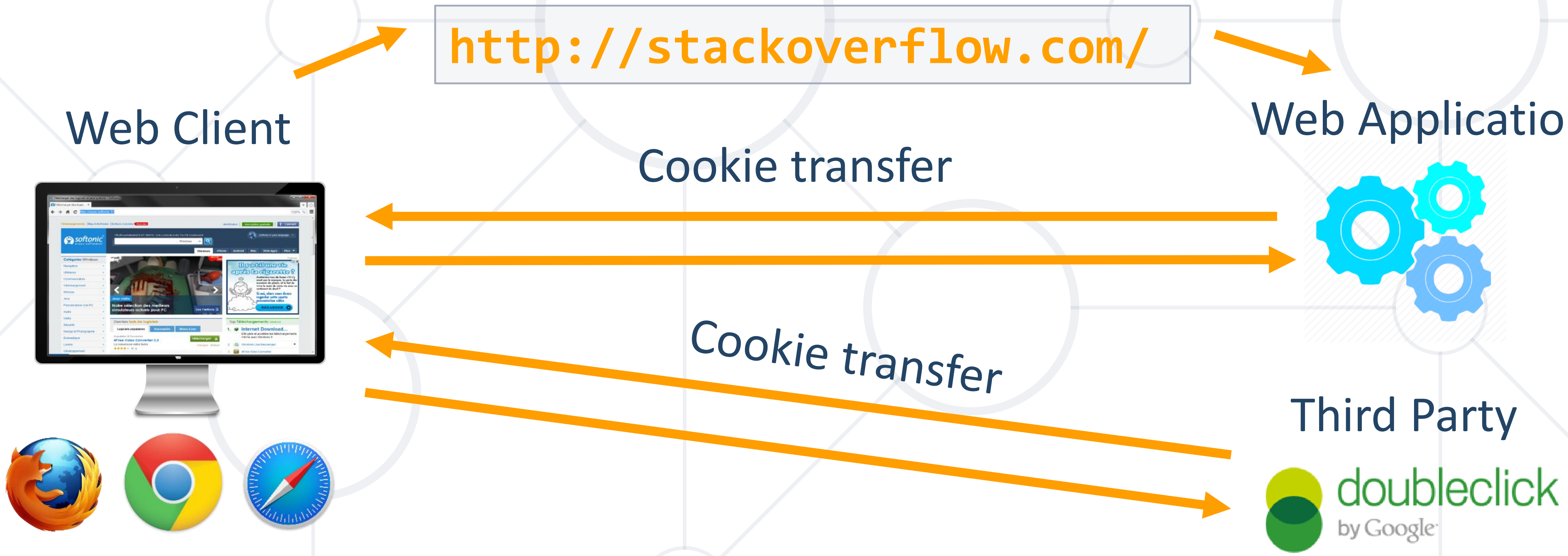
- The cookie file contains a table with **key-value** pairs



Name:	ELOQUA
Content:	GUID=50B3A712CDAA4A208FE95CE1F2BA7063
Domain:	.oracle.com
Path:	/
Send for:	Any kind of connection
Accessible to script:	Yes
Created:	Monday, August 15, 2016 at 11:38:50 PM
Expires:	Wednesday, August 15, 2018 at 11:38:51 PM

Third Party Cookies

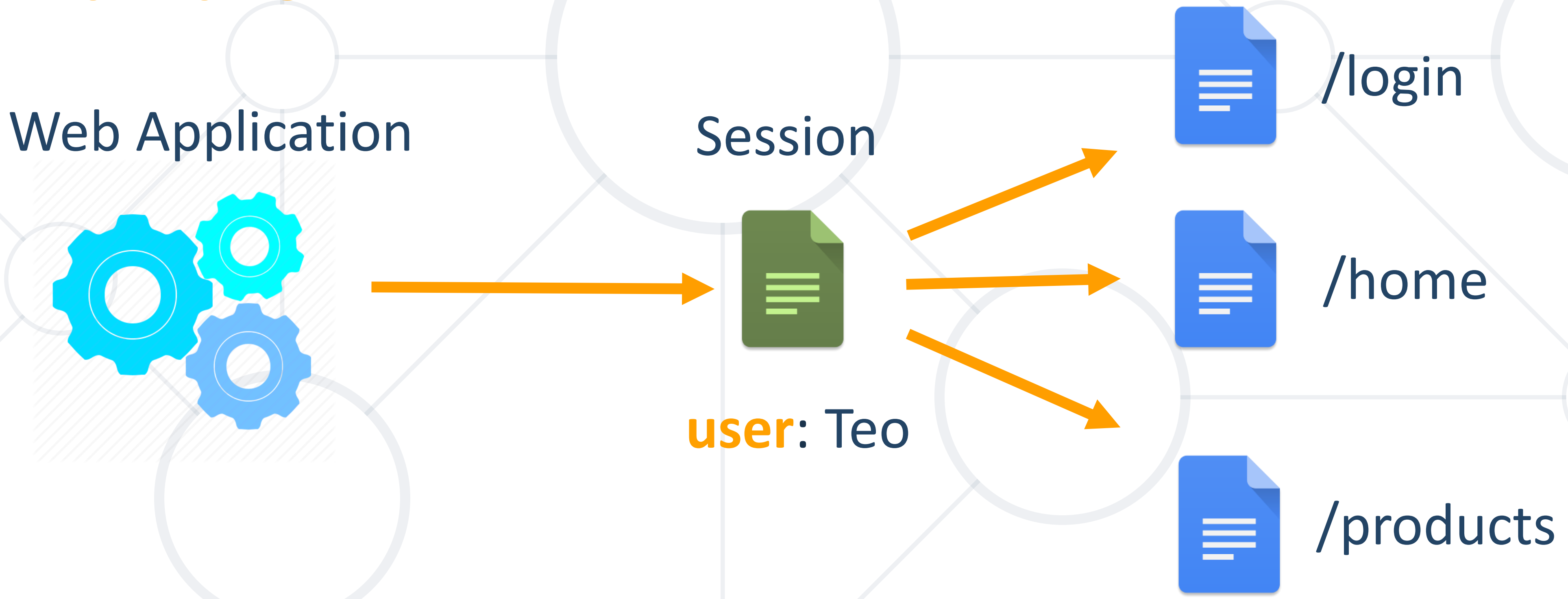
- Cookies stored by an **external party** (different **domain**)
- Mainly used for advertising and tracking across the web



HTTP Sessions

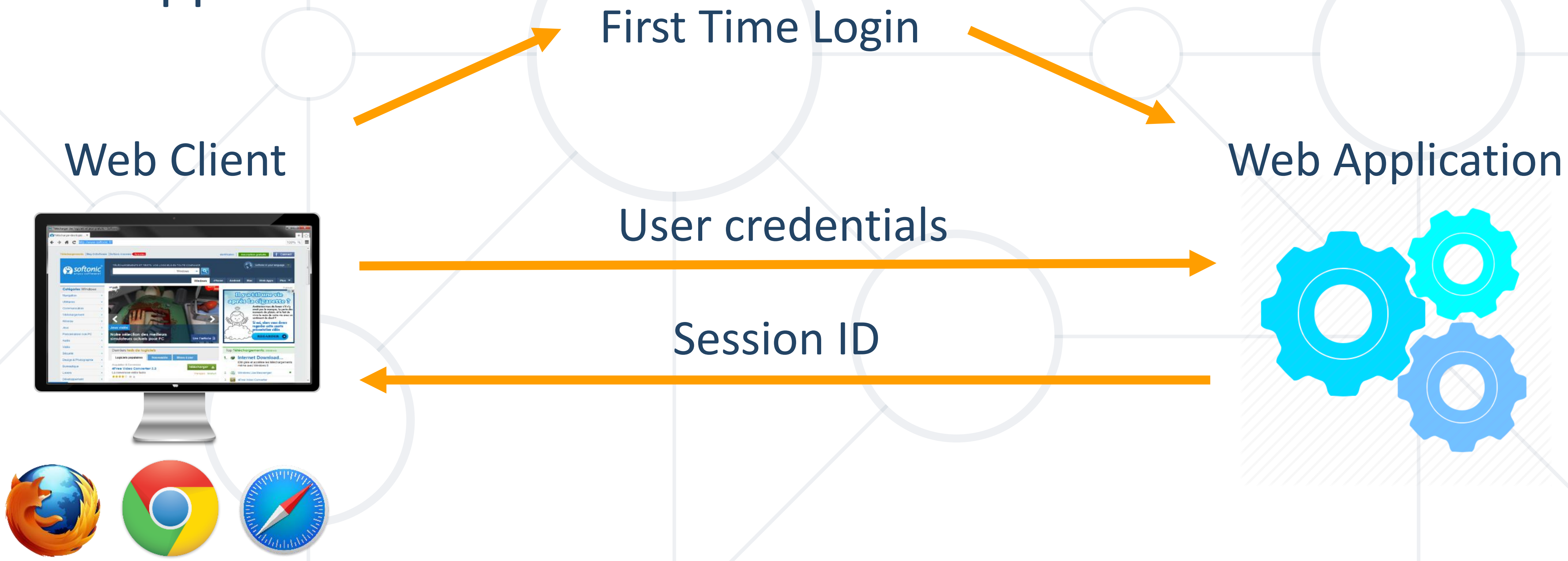
What Are Sessions?

- A way to store information about a user to be used across **multiple pages**



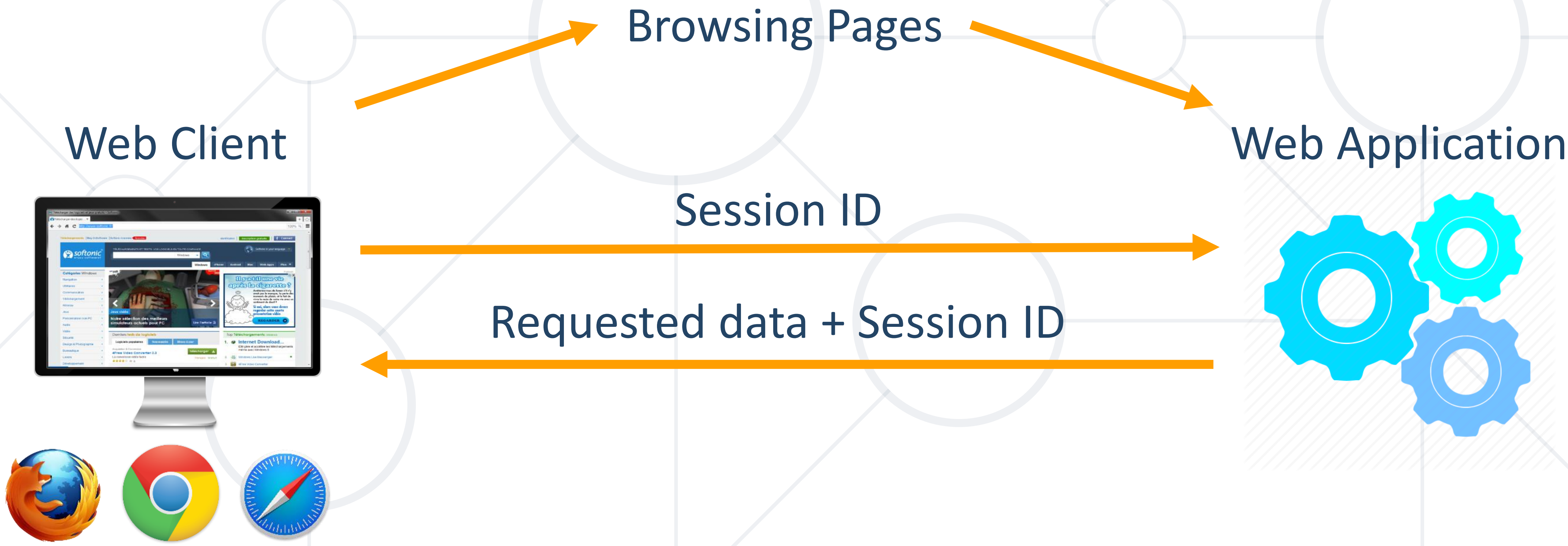
Session Management

- The exchange mechanism be used between the user and the web application



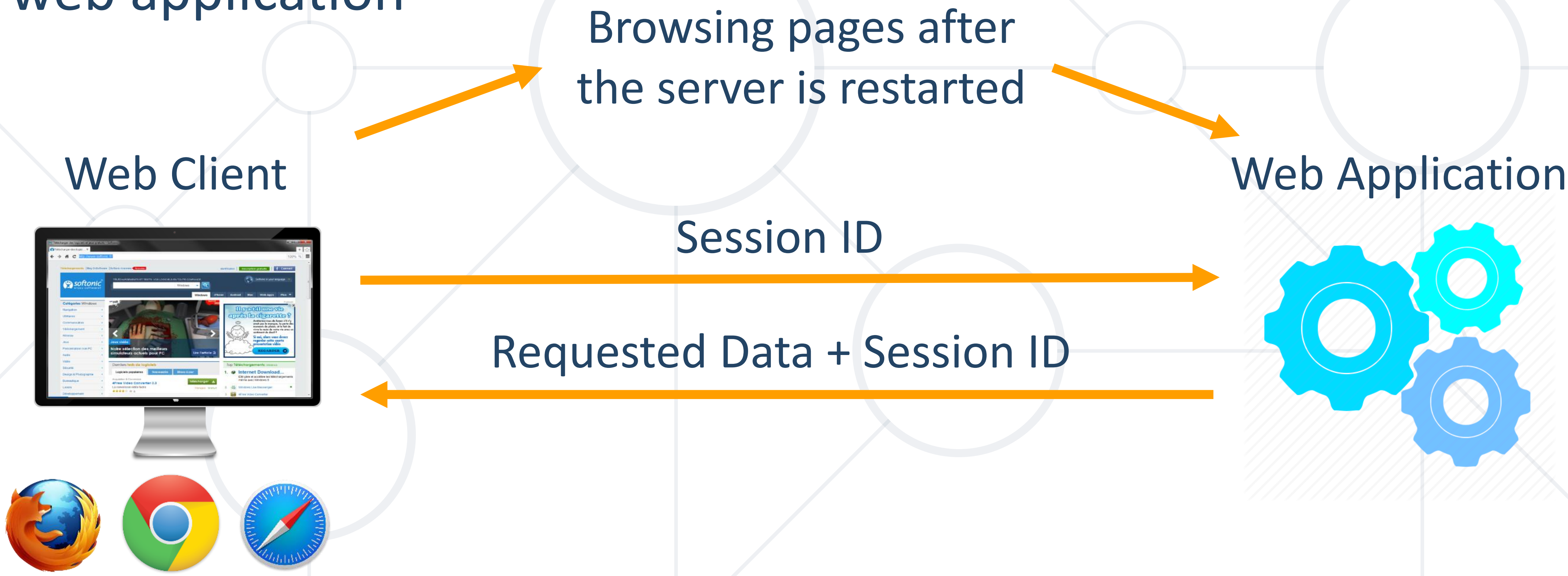
Session Management

- The exchange mechanism be used between the user and the web application



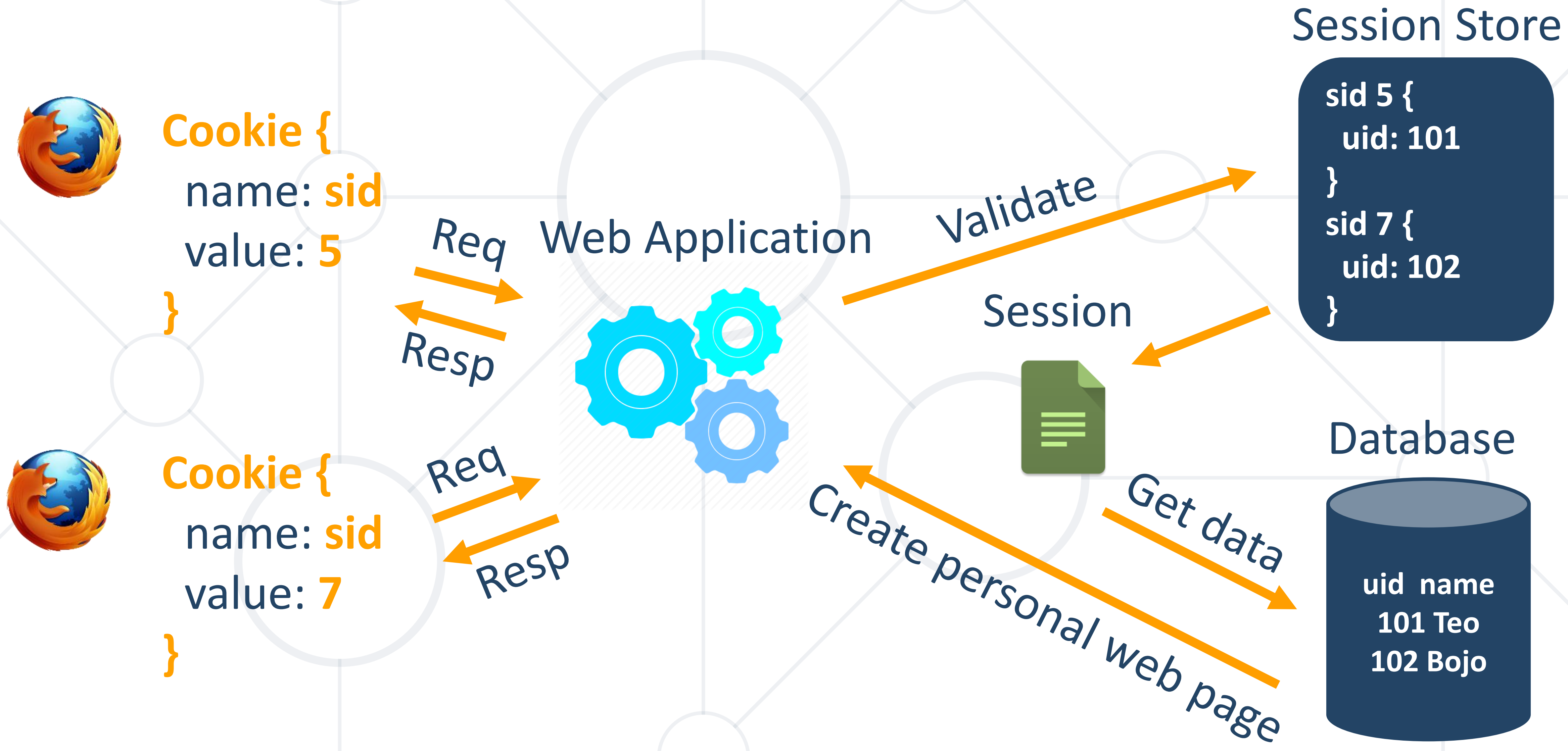
Session Management

- The exchange mechanism be used between the user and the web application

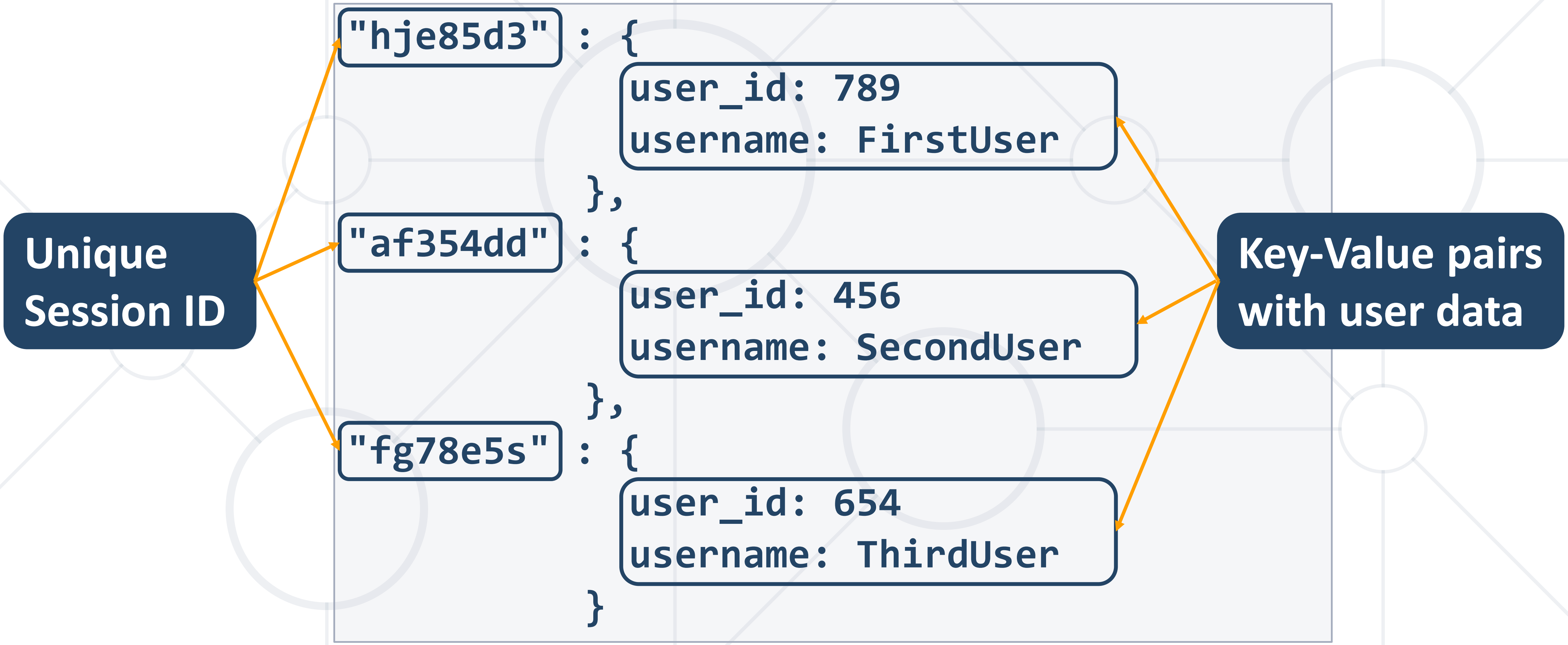


Browsing pages after the server is restarted
Web Client
Web Application
Session ID
Requested Data + Session ID

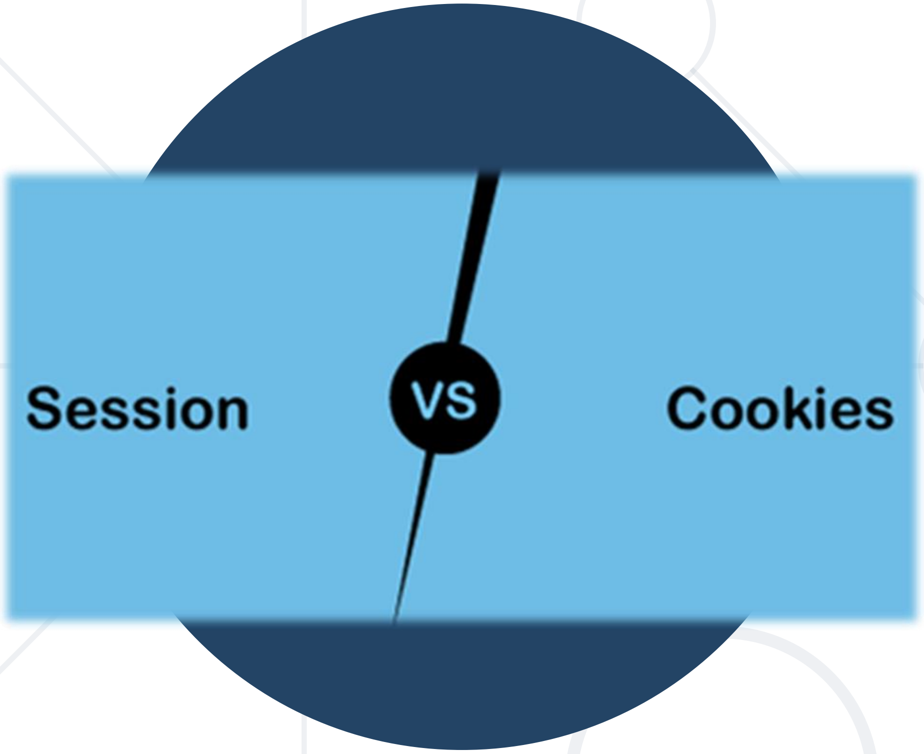
Relation with Cookies



Session Structure



```
"hje85d3" : {
              user_id: 789
              username: FirstUser
            },
"af354dd" : {
              user_id: 456
              username: SecondUser
            },
"fg78e5s" : {
              user_id: 654
              username: ThirdUser
            }
```



Session vs Cookies

Differences and Usage

Session vs Cookies

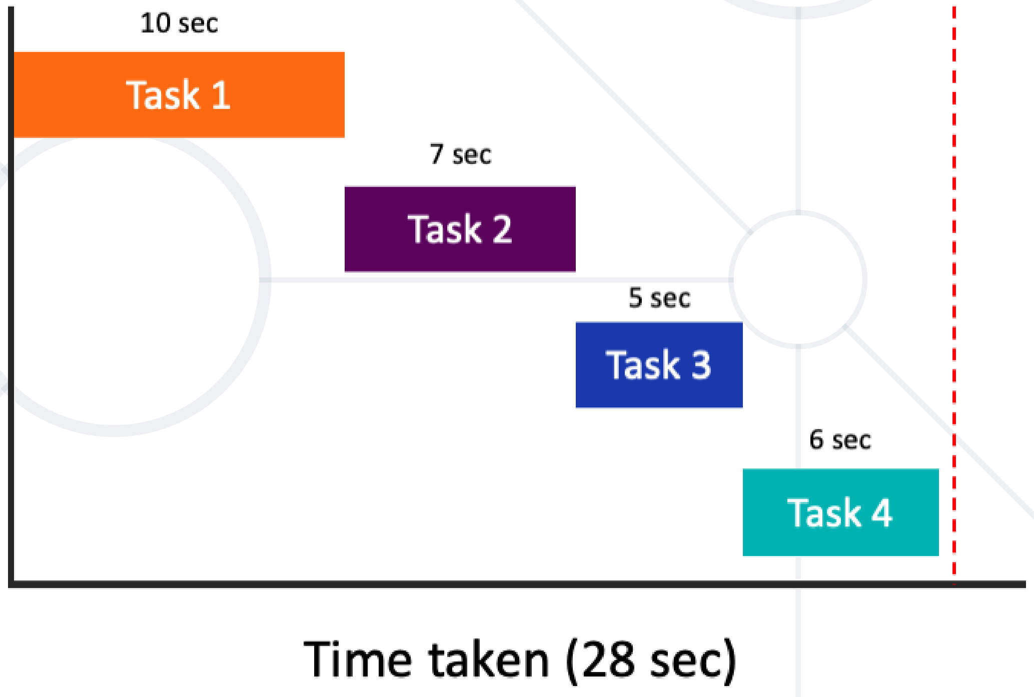
- **Session**
  - Stored on the server
  - Expires when the user closes the browser
  - It can store an unlimited amount of data
  - Depends on the cookie
  - Secure –saves data in encrypted form and cannot be accessed by anyone easy
- **Cookies**
  - Stored on the user's computer as a text file
  - Expires on its expiration date
  - It can store only limited data
  - Does not depend on the session
  - Have security issues, as data is stored in a text file and it can be accessed by anyone easily

Synchronous Programming

Benefits and Drawbacks

Synchronous Programming

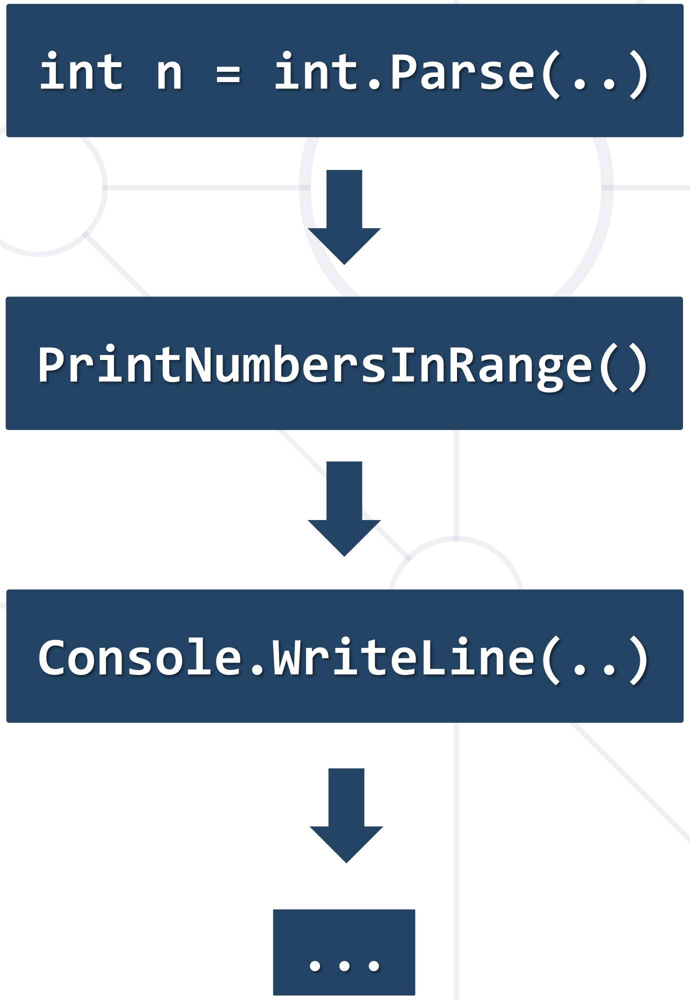
- Executing program components **sequentially**
  - "Sequential programming"
  - Actions happen one after another
- Components **wait** for previous components to finish
- Program resources are accessible at all points



Synchronous Code

- Synchronous code is executed **step by step**

```
static void Main()
{
    int n = int.Parse(Console.ReadLine());
    PrintNumbersInRange(0, 10);
    Console.WriteLine("Done.");
}

static void PrintNumbersInRange(int a, int b)
{
  for (int i = a; i <= b; i++)
  {
    Console.WriteLine(i);
  }
}
```



Synchronous Code – Long Running Operation

```
Console.Write("Enter your name: ");
string name = Console.ReadLine();

for (int i = 0; i < int.MaxValue; i++)
{
  // Execute some operations here
}

Console.WriteLine($"Hello, {name}!");
```

You will have to wait for the long-running operation to finish before you can see the greeting

Synchronous Programming Drawbacks

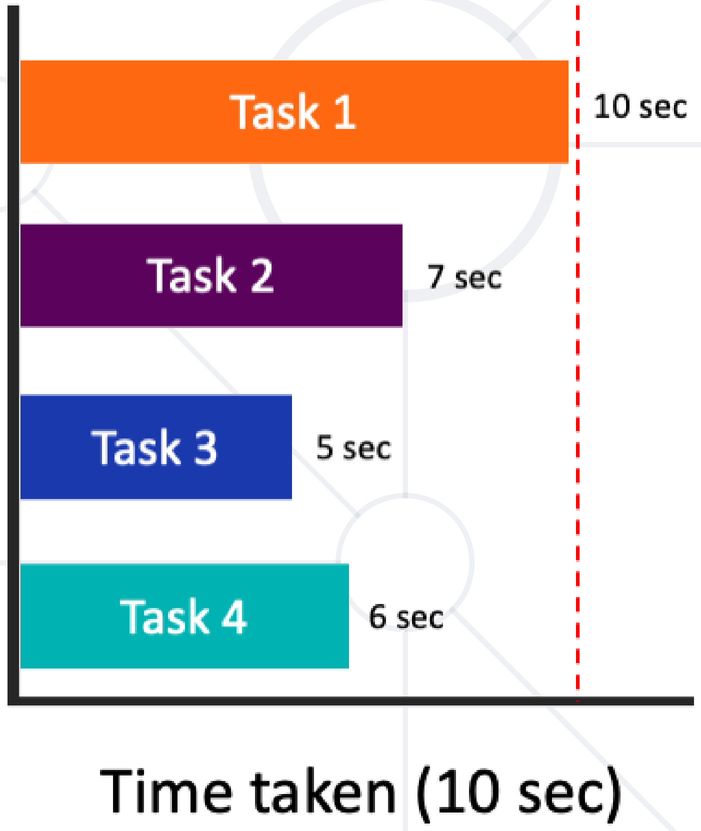
- If one **component is blocked**, the **entire program is blocked**
- UI may become **unresponsive**
- No utilization of multi-core systems
- CPU-demanding tasks **delay execution** of all other tasks
- **Accessing resources** blocks entire program
  - Especially problematic with web resources

Asynchronous Programming

Benefits and Drawbacks

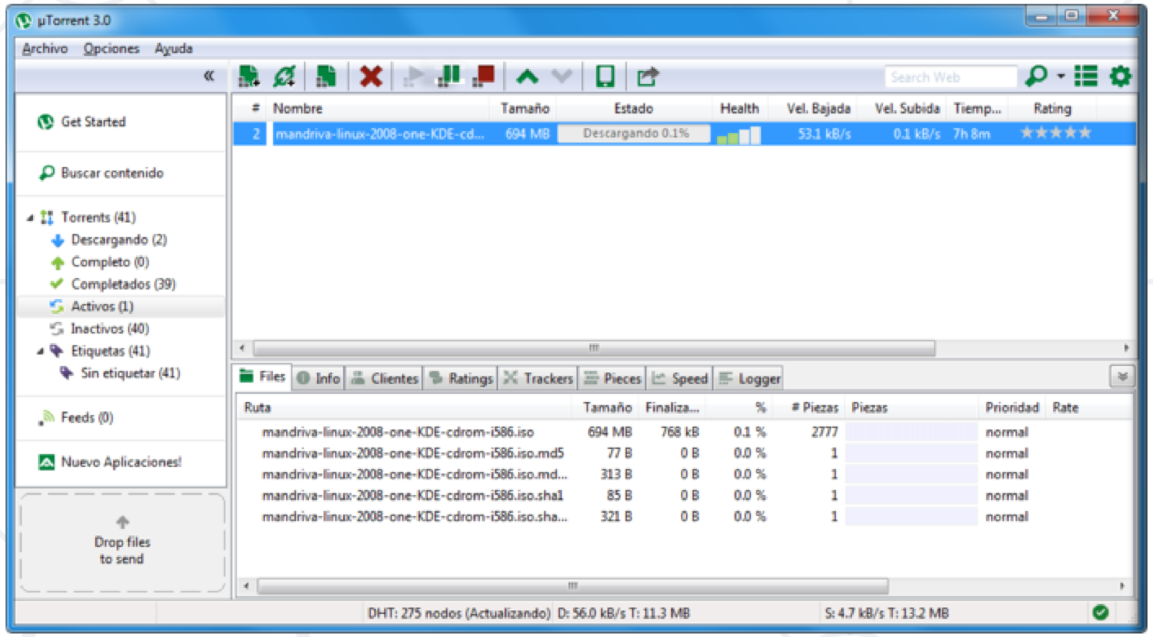
Asynchronous Programming

- Program components can execute in **parallel**
  - Some actions run alongside other actions
  - Each action can happen in a **separate** thread
- **Independent** components don't wait for each other
- Program resources shared between threads
  - If one thread uses a resources, others shouldn't use it



Asynchronous Programming – Benefits

- If a component is blocked, other **components still run**
  - UI runs separately and always remains responsive
- Utilization of multi-core systems
  - Each core executes **one or more** threads
- CPU-demanding tasks run on "**background**" threads
- Resource access runs on "**background**" threads



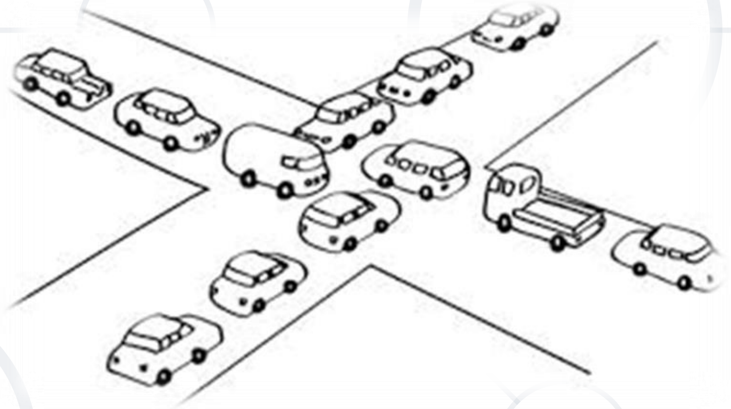
#	Nombre	Tamaño	Estado	Health	Vel. Bajada	Vel. Subida	Tiemp...	Rating
2	mandriva-linux-2008-one-KDE-cd...	694 MB	Descargando 0.1%		53.1 kB/s	0.1 kB/s	7h 8m	★★★★★

Ruta	Tamaño	Finaliza...	%	# Piezas	Piezas	Prioridad	Rate
mandriva-linux-2008-one-KDE-cdrom-i586.iso	694 MB	768 kB	0.1 %	2777		normal	
mandriva-linux-2008-one-KDE-cdrom-i586.iso.md5	77 B	0 B	0.0 %	1		normal	
mandriva-linux-2008-one-KDE-cdrom-i586.iso.md...	313 B	0 B	0.0 %	1		normal	
mandriva-linux-2008-one-KDE-cdrom-i586.iso.sha1	85 B	0 B	0.0 %	1		normal	
mandriva-linux-2008-one-KDE-cdrom-i586.iso.sha...	321 B	0 B	0.0 %	1		normal	

DHT: 275 nodos (Actualizando) D: 56.0 kB/s T: 11.3 MB S: 4.7 kB/s T: 13.2 MB

Asynchronous Programming – Drawbacks

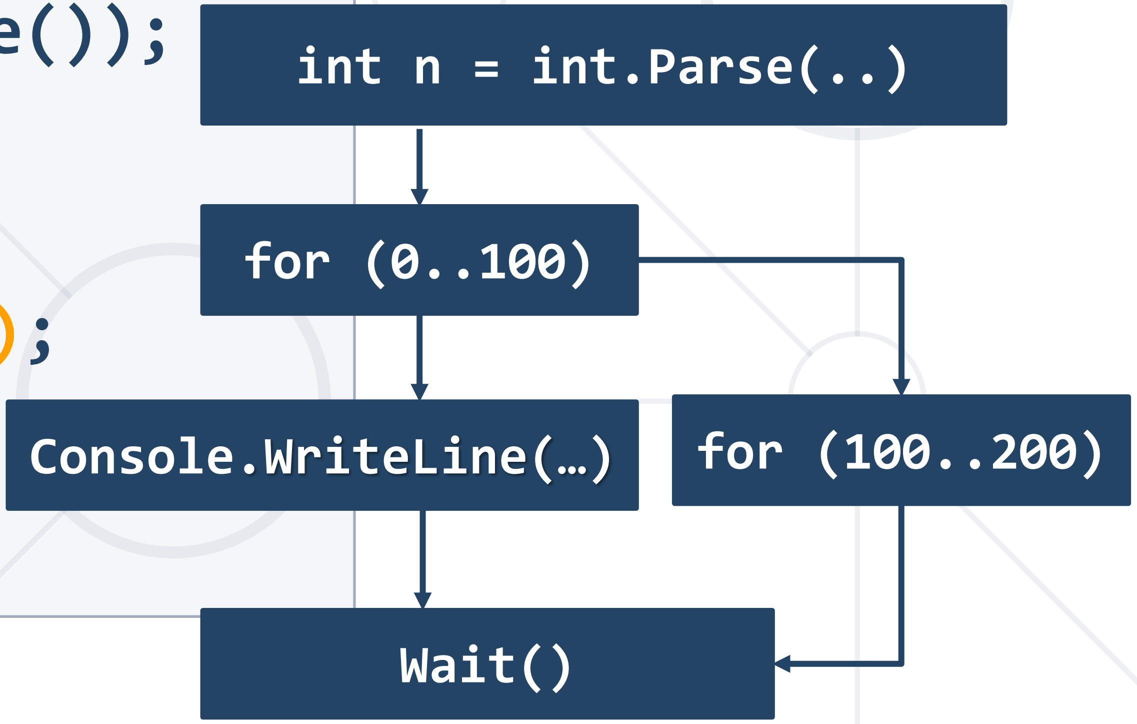
- Hard to know which code parts are running at a specific time
- Harder than usual to **debug**
- Have to **protect resources**
  - One thread uses a resource
  - Other threads must wait for the resource
- **Hard to synchronize** resource access
  - **Deadlocks** can occur



Asynchronous Code

- Asynchronous programming allows the execution of code **simultaneously**

```
int n = int.Parse(Console.ReadLine());

PrintNumbersInRange(0, 100);
var task = Task.Run(() =>
    PrintNumbersInRange(100, 200));

Console.WriteLine("Done.");
task.Wait();
```



Threads

Call Stack, Thread-Safety, Exception Handling

Instruction Execution

- Each program's code is translated to CPU instructions

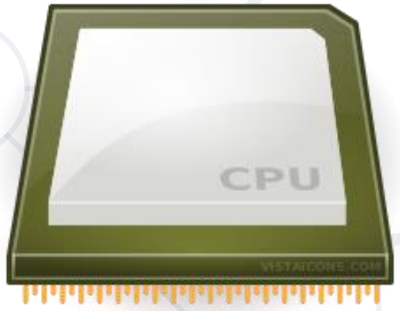
Program.cs

```
int a = 5;
int b = 4;
Console.WriteLine(a + b);
```

Compilation

Program.exe

```
00DA2655  mov   dword ptr [ebp-40h],5
00DA265C  mov   dword ptr [ebp-44h],4
00DA2663  mov   ecx,dword ptr [ebp-40h]
00DA2666  add   ecx,dword ptr [ebp-44h]
00DA2669  call  73B5A920
00DA266E  nop
```

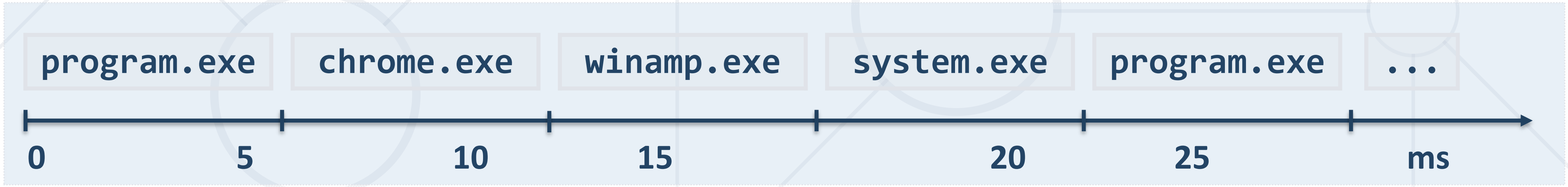


Single-Core CPU

Instructions are executed one by one

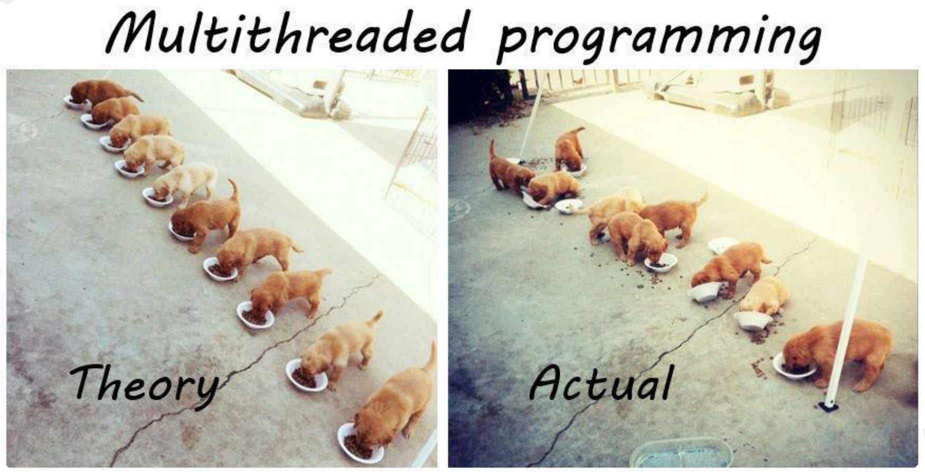
Multi-Tasking

- A computer can run **many processes** (applications) at once
  - But each CPU core can only execute one instruction at a time
  - **Parellelism** is achieved by the operating system's **scheduler**
    - Grants each **thread** a small interval of time to run



Threads

- A **thread** is a fundamental unit of code execution
- Commonly, processes (programs) use more than one thread
  - In .NET, there is always more than one thread (e.g. GC)
- Each thread has a **memory area** associated with it known as a **Call Stack**
  - Stores **local variables**
  - Stores the **currently invoked methods** in order of invocation



Threads in C#

- Threads in C# can be created using the **`System.Thread`** class
- Constructor accepts a **method** (delegate) to execute on a separate thread

```
Thread thread = new Thread(() =>
  {
    for (int i = 0; i < 10; i++)
    {
      Console.WriteLine(i);
    }
  });
```

System.Thread

- `Start()` – schedules the thread for execution
- `Join()` – waits for the thread to finish its work (blocks the calling thread)

```
Thread primes = new Thread(() =>
    PrintPrimesInRange(1, 10000));
primes.Start();

Console.WriteLine("Waiting for thread to finish work...");
primes.Join();
```

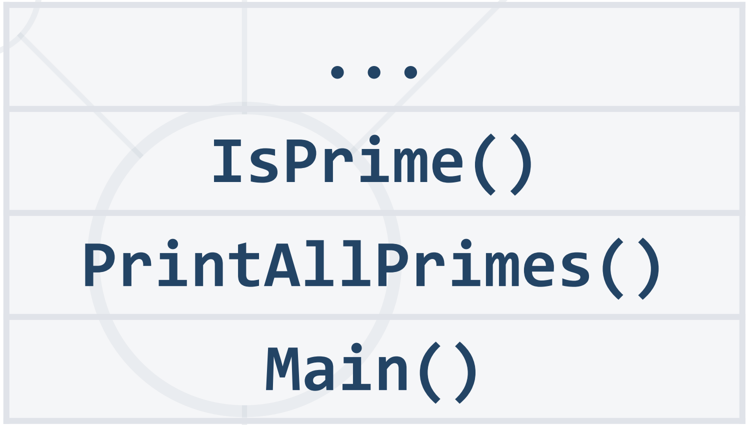
Thread – Example

```
List<long> numbers = new List<long>();
Thread t = new Thread(() =>
  SumOddNumbers(numbers, 10, 100000000L));
t.Start();

Console.WriteLine("What should I do?");
while (true)
{
  string command = Console.ReadLine();
  if (command == "exit") break;
}

t.Join();
```

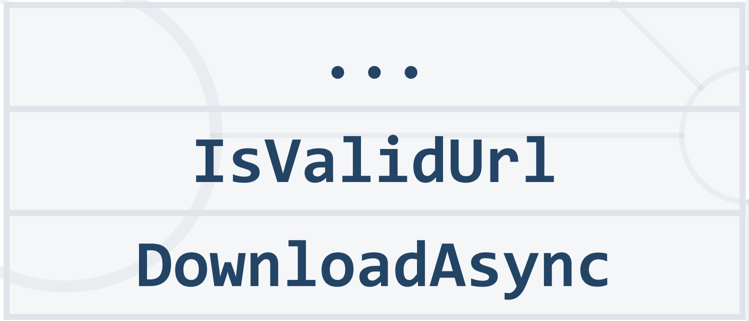
Console interface remains unblocked

Thread Stack

- Each thread has its own **stack**
  - The start (bottom) of the stack is the method from which the thread began execution
  - Each method (frame) stores local variables



main thread



background thread

Thread Race Conditions

- A **race condition** occurs when two or more threads access shared data and they try to change it at the same time

```
List<int> numbers = Enumerable.Range(0, 10000).ToList();
for (int i = 0; i < 4; i++)
{
  new Thread(() =>
  {
    while (numbers.Count > 0)
      numbers.RemoveAt(numbers.Count - 1);
  }).Start();
}
```

Thread Safety

- A thread-safe resource can be safely accessed by multiple threads
- **lock** keyword grants access to only one thread at a time
  - Avoids race conditions
  - Blocks any other threads until the lock is released

```
lock (numbers)
{
  if (numbers.Count == 0) break;
  int lastIndex = numbers.Count - 1;
  numbers.RemoveAt(lastIndex);
}
```

Exception Handling

- Exceptions cannot be handled outside a thread

```
try
{
  new Thread(DoWork).Start();
}
catch (Exception ex)
{
  Console.WriteLine("Exception!");
}
```

This part will never be reached

```
public static void DoWork()
{
  throw new ArgumentNullException();
}
```

Exception Handling – the Right Way

```
new Thread(DoWork).Start();
```

```
public static void DoWork()
{
  try
  {
    throw new ArgumentNullException();
  }
  catch (Exception ex)
  {
    Console.WriteLine("Exception handled!");
  }
}
```

Exceptions should be handled inside the executed method(s)

Tasks in C#

Task Parallel Library

Tasks in C#

- A task is a high-level representation of concurrent work
  - Runs in **parallel** with the main thread
  - May not run on a new thread (the CLR decides)
  - Offers several operations
    - Creating, running and **returning** result
    - Continuing with another task (**chaining several operations**)
    - Proper exception handling
    - Progress/state reports

Creating Tasks in C#

- Creating tasks can be done in several ways
  - Initialize a new **Task** object

```
Task task = new Task(() => { Console.WriteLine(""); });
```

  - **Task.Run()**

```
Task.Run(() => TraverseMatrix());
```

  - **Task.Factory.StartNew()** – enables additional task customization

```
Task.Factory.StartNew(() => CopyFileContents("got-s03ep1.avi"),
    TaskCreationOptions.LongRunning);
```

Generic Tasks

- **Task<T>** is a task that will return a result sometime in the future

```
Task<long> task = Task<long>.Run(() =>
{
  long sum = 0;
  for (int i = 0; i < 10000; i++) sum += i;
  return sum;
});

Console.WriteLine(task.Result);
```

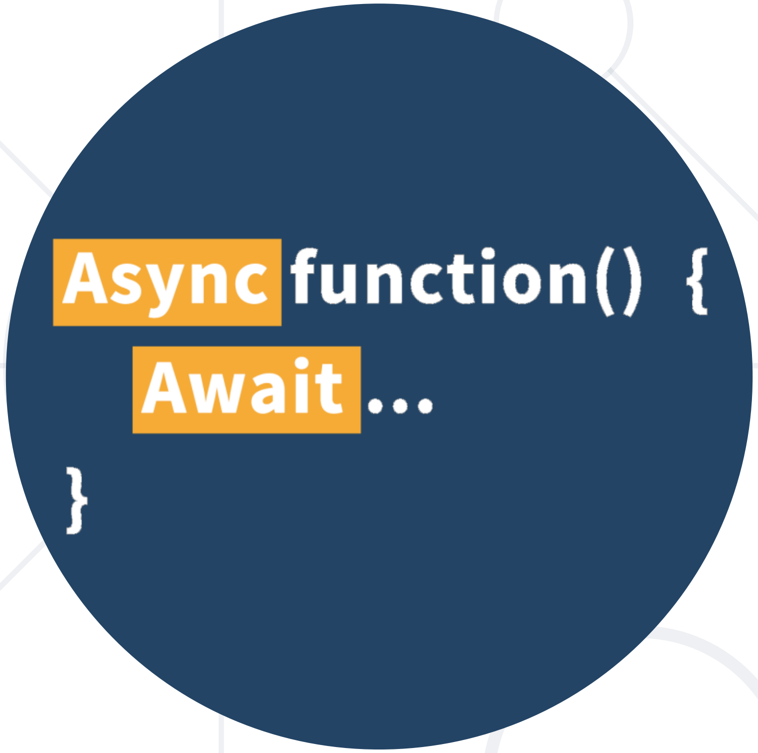
Blocks the calling thread until the task returns a result

Task Exception Handling

- **Exceptions** that have occurred within the body of a **Task** can be captured and handled outside of it

```
var task = SliceAsync(VideoPath, DestinationPath, 5);
try
{
  task.Wait();
}
catch (AggregateException ex)
{
  // Handle exception...
}
```

You can use the **AggregateException** to wrap all exceptions thrown by different threads



```
Async function() {
    Await...
}
```

Async and Await

Keywords for Asynchronous Operations

Tasks with Async and Await

- The keywords **async** and **await** are **always** used together
- **async** hints the compiler that the method might run in parallel
  - Does not make a method run asynchronously (**await** makes it)

```
static async void SliceFileAsync(string file, int parts)
```

  - Tells the compiler "**this method could wait for a resource or operation**"
    - If it starts waiting, return to the calling method
    - When the wait is over, go back to called method

Tasks with Async and Await

- `await` is used in a method which has the `async` keyword
  - Saves the context in a state machine
  - Marks waiting for a resource (a task to complete)
    - Resource should be a `Task<T>`
    - Returns `T` result from `Task<T>` when it completes

```
await DownloadStringAsync("https://softuni.org");
```

Returns `Task<string>`

Async and Await – Example

```
static void Main()
{
  DownloadFileAsync(FileUrl, "book.pdf");
  // Do some other work
}
static async void DownloadFileAsync(string url, string fileName)
{
  Console.WriteLine("Downloading...");
  await Task.Run(() =>
  {
    // Download the file
  });
  Console.WriteLine("Download successful.");
}
```

After the method is over the calling thread gets back to the calling method

The calling thread exits the method on await

When the waiting is over, the calling thread proceeds with method execution

Summary

- State management
  - Cookies are client based stored information
  - Sessions are server-based information
- Asynchronous processing
  - A thread is a unit of code execution
  - Multithreading
  - Tasks facilitate the work with multithreading
    - async and await keywords

Software University

Questions?

SoftUni

Software University

SoftUni Creative

SoftUni Digital

SoftUni Foundation

SoftUni Kids

Finance Academy

SoftUni Diamond Partners

Software University

SmartIT

Coca-Cola HBC Bulgaria

SOFTWARE GROUP

DXC TECHNOLOGY

createX

Postbank

Решения за твоето утре

SUPER HOSTING .BG

DRAFT KINGS

VIVACOM

INDEAVR

Serving the high achievers

AMBITIONED

PHAR VISION

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, about.softuni.bg
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg

License