How does a web session work ?

machinesaredigging.com/2013/10/29/how-does-a-web-session-work/

Not long ago I had to investigate on a session reset bug that forced me to do some research on sessions. Since I didn't find the subject well covered, I thought I would share what helped me solve my bug, so that it can also help you in time.

I will describe here sessions in web applications, my area of practice.

What is a session ?

"session" is one of those computing terms that refers to seemingly different things : a shell session, a tcp session, a login session, a desktop session, a browser session, a server session etc. This makes it confusing to understand what exactly a session is. Same for the "cache", another confusing term (database cache, browser cache, framework cache, network cache..). But actually what defines those confusing terms is the use they're describing.

So, the first thing to understand for sessions is their use.

Use of a session

Generally you should understand the session as the different states of an application during the time a user is interacting with it, it is specific to the user. I would even say a session is an instance of the interaction of a user with an application, but I'm not sure it clarifies the matter. Now more specifically for a web session, the session is a data structure that an application uses to store temporary data that is useful only during the time a user is interacting with the application, it is also specific to the user.

For example, you could save the user's name in the session so that you don't have to query the database every time you need it or you could store data in the session to save state between pages (between pages of a payment process for example).
Think of it as a volatile memory quickly accessible that is allocated to each user who is using the application, and when the user quits, it is destroyed.

This is the general concept, the storage mechanism and how it is implemented is then specific to the application. This temporary storage could be on the file system in text files, on a database or in the internal memory of the program executing the application.

The second thing to understand is the structure of a session.

Structure of a session

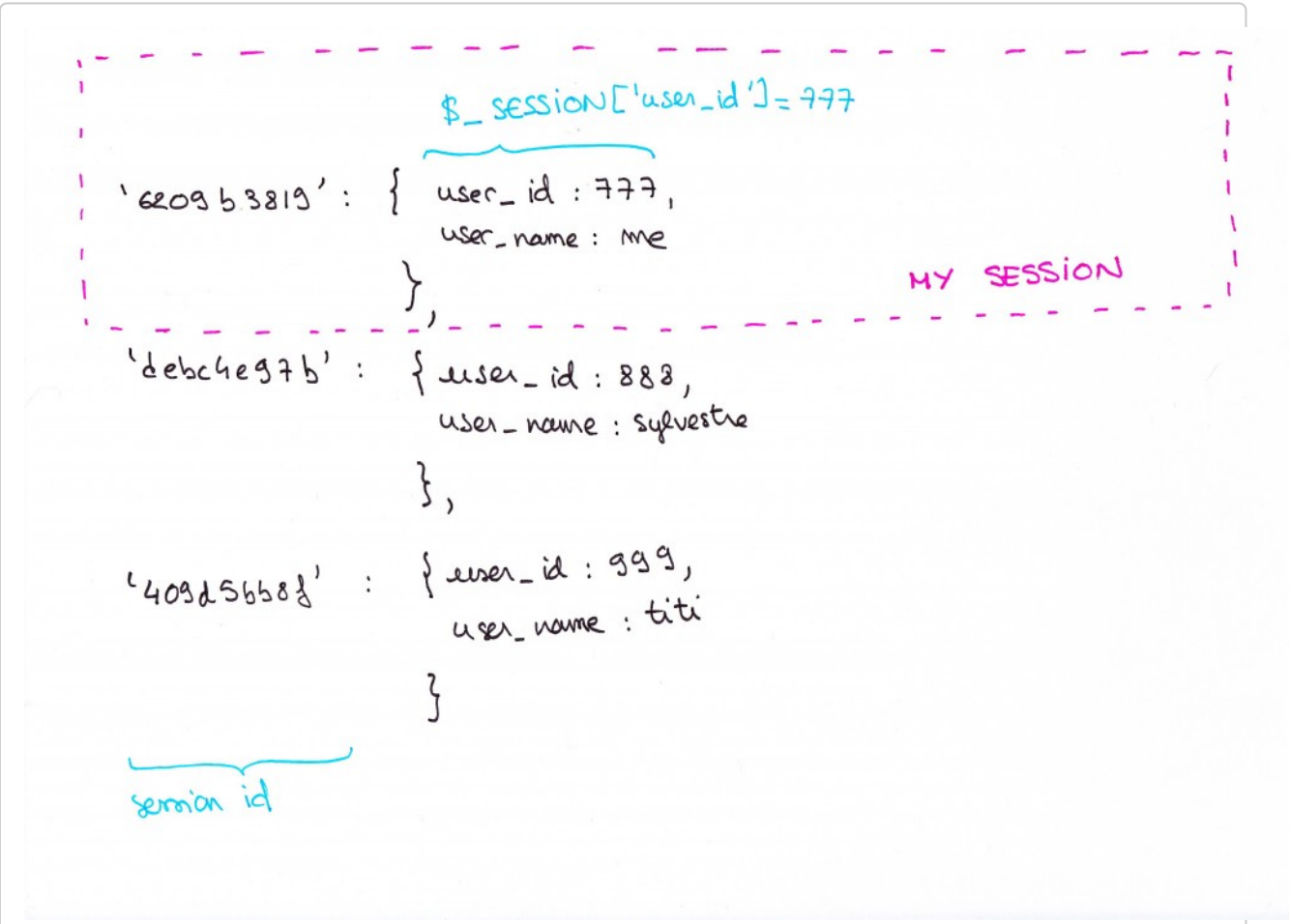
The session is a key-value pair data structure. Think of it as a hashtable where each user gets a hashkey to put their data in. This hashkey would be the "session id". A session data structure would look like this :

And when you say, "my session" you would refer to your entry in the session object. Every user is able to access only their session. The session can be stored on the server, or on the client. If it's on the client, it will be stored by the browser, most likely in cookies and if it is stored on the server, the session ids are created and managed by the server. So if there are a million users connected to the server, there will also be a million session ids for those users on the server.

From now, I will focus only on server side sessions.

How does a session work ?

So how exactly do users access their session?
For a single user application, like a desktop application, there is only one user, so there is also one session, it is not difficult for the application to



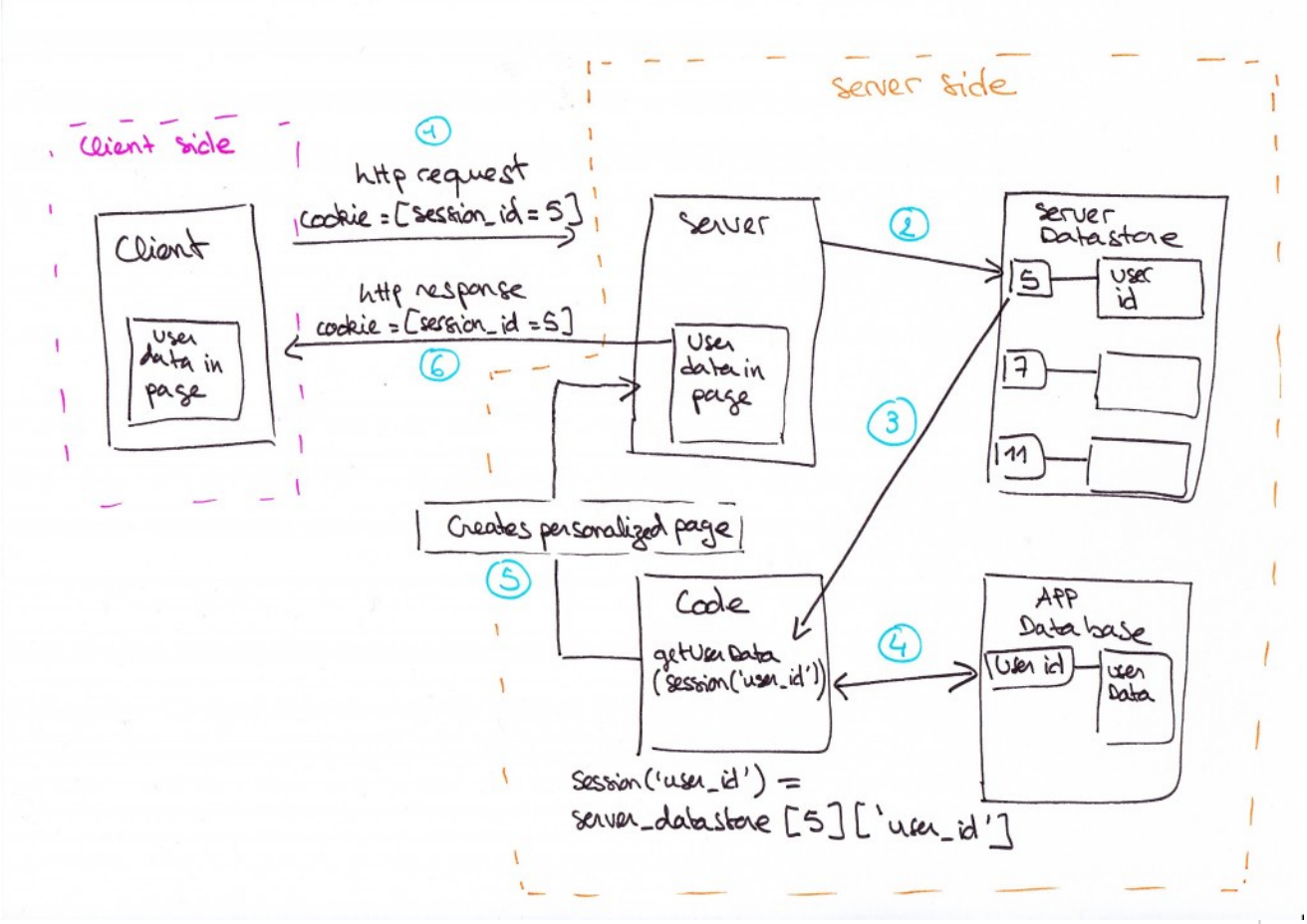
Session datastructure

make the connection between the user and their session data. However, for a web application, a server has multiple clients, how does it know which session is yours? That's where the session id comes into play.

The general principle is that you, as the client, give the server your session id, and in return the server grants you access to your session data if it finds your session id stored in its session datastore. The session structure is like a data locker for users, and the key for the locker is the session id, the server is the guy who shows you which one is your locker.

Let's look more in details how it works :

Let's start from the moment when you land on a webpage. When you receive a webpage from the server, along with the page content itself, the server sent you (in general, in a cookie) the session id that it set to

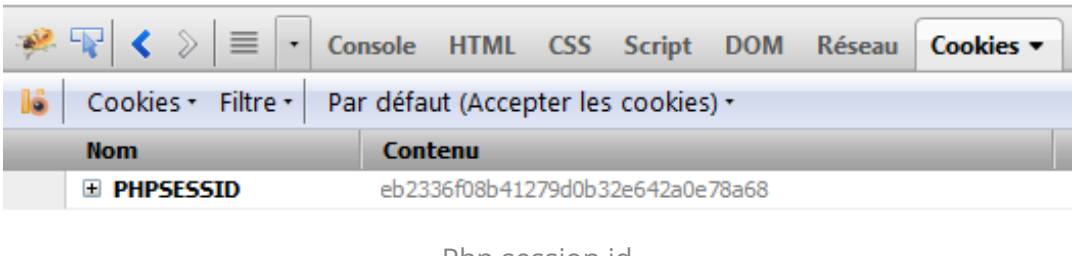


How does a session work ?

identify your connection among all the requests that it gets.

Make the experiment, open your console and check the cookies, you will see something that looks like :

JSP will send you



Php session id

JSESSIONID, and ASP ASPSESSIONID, here the back end is PHP.

After you logged in, the application validated your password and login and saved your user id in the session so that every time you will make a request, you won't have to log in again (this will be detailed later).

Now let's review the diagram above to understand what is going on when you make another request to get more data. For example, let's say you landed on Gmail inbox after you logged in, and now you want to navigate to your drafts page.

1 – You send a http request to the server asking for the drafts page. Along with this http request you send your session id to tell the server "hey, it's me from before, give me my drafts page now". The session id is usually sent in cookies, but it can also be sent in GET or POST parameters, whatever the technique the session id just needs to be sent to the server.

2 – The server receives your request. Before it gives you your drafts page, it checks your session id, looks it up in its session datastore, it finds 5, your session id, so it makes the data in entry 5 available to the code engine (php, java, ruby...).

3 -The server then executes the code corresponding to your request "give me the drafts page".

4 – The code starts by getting your user id from the session made available by the server earlier, then it uses it to ask the database "give me the drafts of the user who has this user id".

5 – Finally when the code got your drafts from the database, it creates an html page, puts your drafts in it, and hands it to the server.

6 – The server sends you your drafts page, along with your session id.

Logged in state

In this exchange, you could have just sent your user id in your request, and told the server you want the drafts of this user id. But that would mean that anybody who knows your user id would also be able to get your drafts and you don't want that for your private data. You prefer that the application sends this data only to you. So to protect your data the application makes you log in first to make sure the person asking for the data is really you. And normally for any request for private data, it should ask you who you are first.

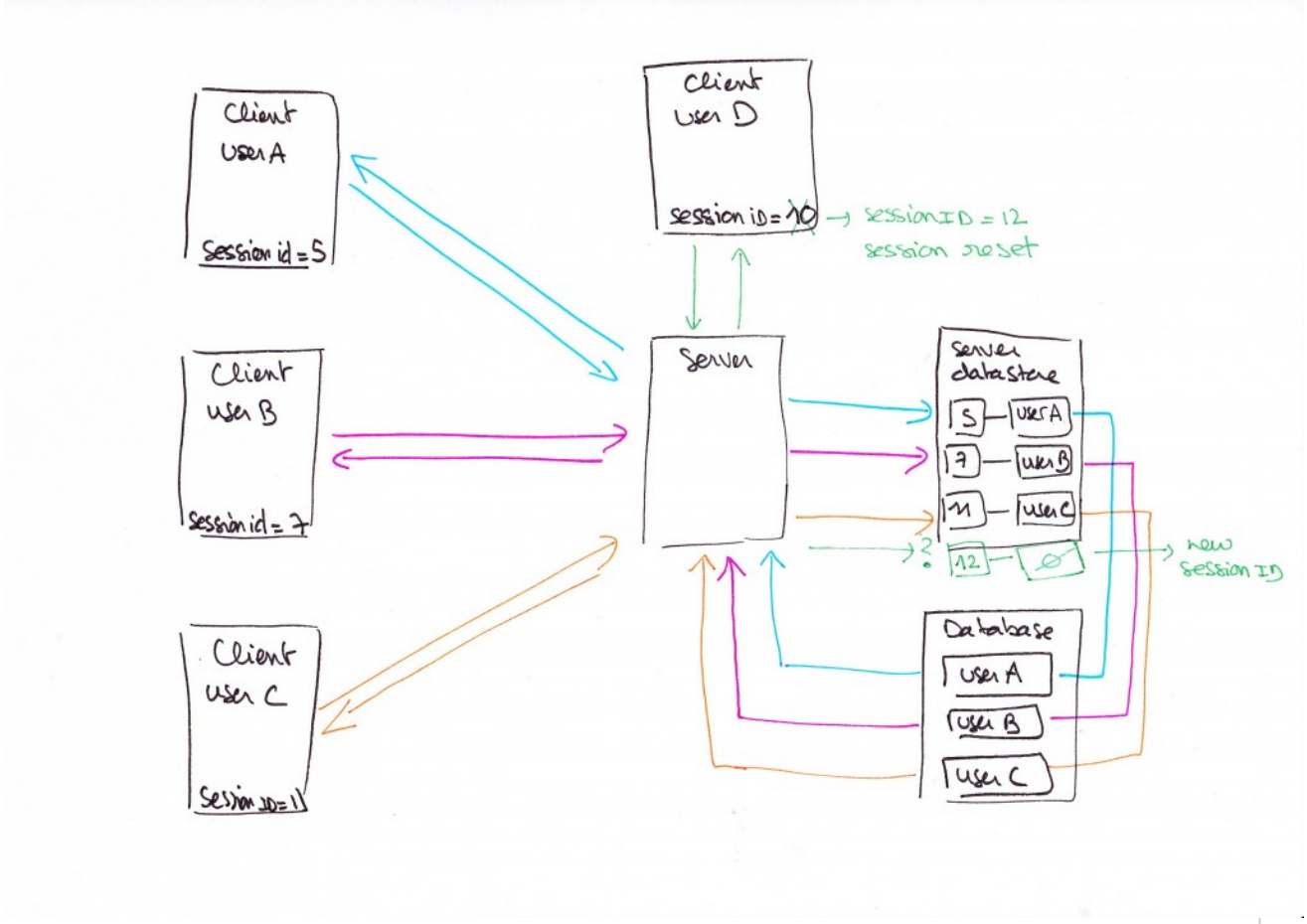
If there was no session id during this exchange, when you asked your drafts page, the server would not know the drafts of who you're asking and it would ask you to log in first. HTTP is a stateless protocol so it doesn't save the fact that you are already logged in. At each request, HTTP doesn't know anything about what happened before, it just carries the request. So for any request for private data you would have to log in again to make sure the application knows this is really you. This would be very annoying.

That's the problem that sessions solve. To avoid logging in all the time after the first time, sessions keep you logged in while you are connected to the server. Basically, after you logged in the first time, the server remembers in the session that this is really you and lets you ask for more data without asking again who you are. That's how sessions are very useful.

So if we zoom out from the preceding diagram, we can observe how user connections are identified and maintained by the server thanks to session ids :

Session management is a feature of the server, you need to activate it. For a static website that doesn't serve private data for example you would not need to activate session management on your server.

Keeping you logged in is one main use of the session, but sessions can also be used to save temporary data that are completely independent of



Simultaneous identified connections

the logged in state. You could decide to put some data in session just because it is quicker to access.

Also as a side note, from the server's point of view : one connection = one session id. So if you connect from two different browsers, the server will create two session ids. You should remember that the session id only identifies your connection to the server. All the user identification logic is handled by the application.

Debugging session problems

The bug I was trying to resolve was a session reset. I couldn't explain why the user was suddenly logged out. This bug drove me totally crazy and I couldn't find any useful help on the Internet because there are so many different kinds of sessions and no one describes the web session as we

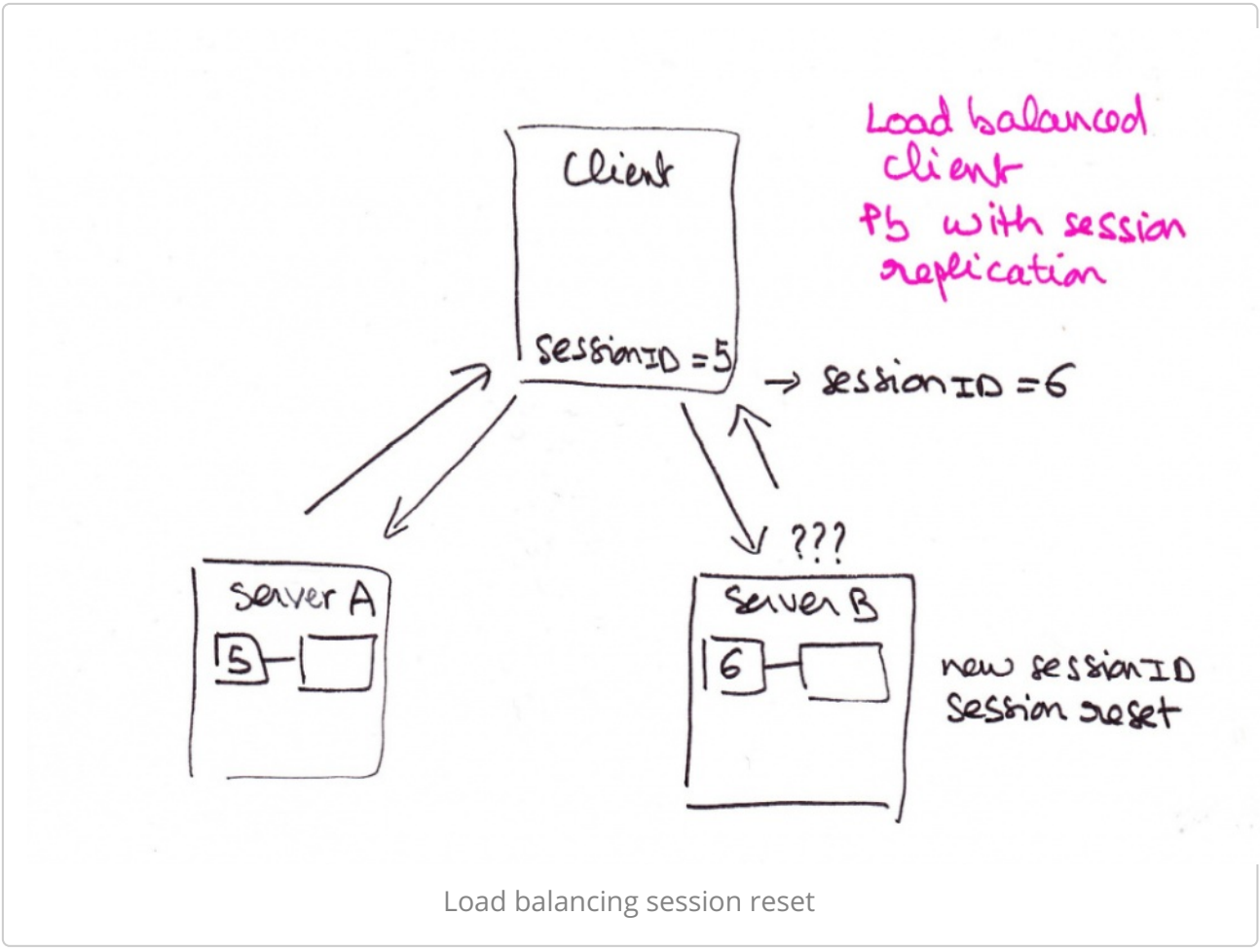
know it. So I thought I would spare you the pain of searching in vain by summing up what you should look for when debugging a session problem.

From the preceding sections, you have seen that the logged in state is maintained by the presence of the session id on the server. So if you are logged out, it means the server doesn't know who you are anymore because it didn't find your session id in its datastore. Most likely your session id never got to your server, you need to find where it got lost, or maybe it's just not on your server, you need to find out why.

Why would a page reset a session? Reasons for session reset?

Here are some scenarios that will help you narrow down your problem.

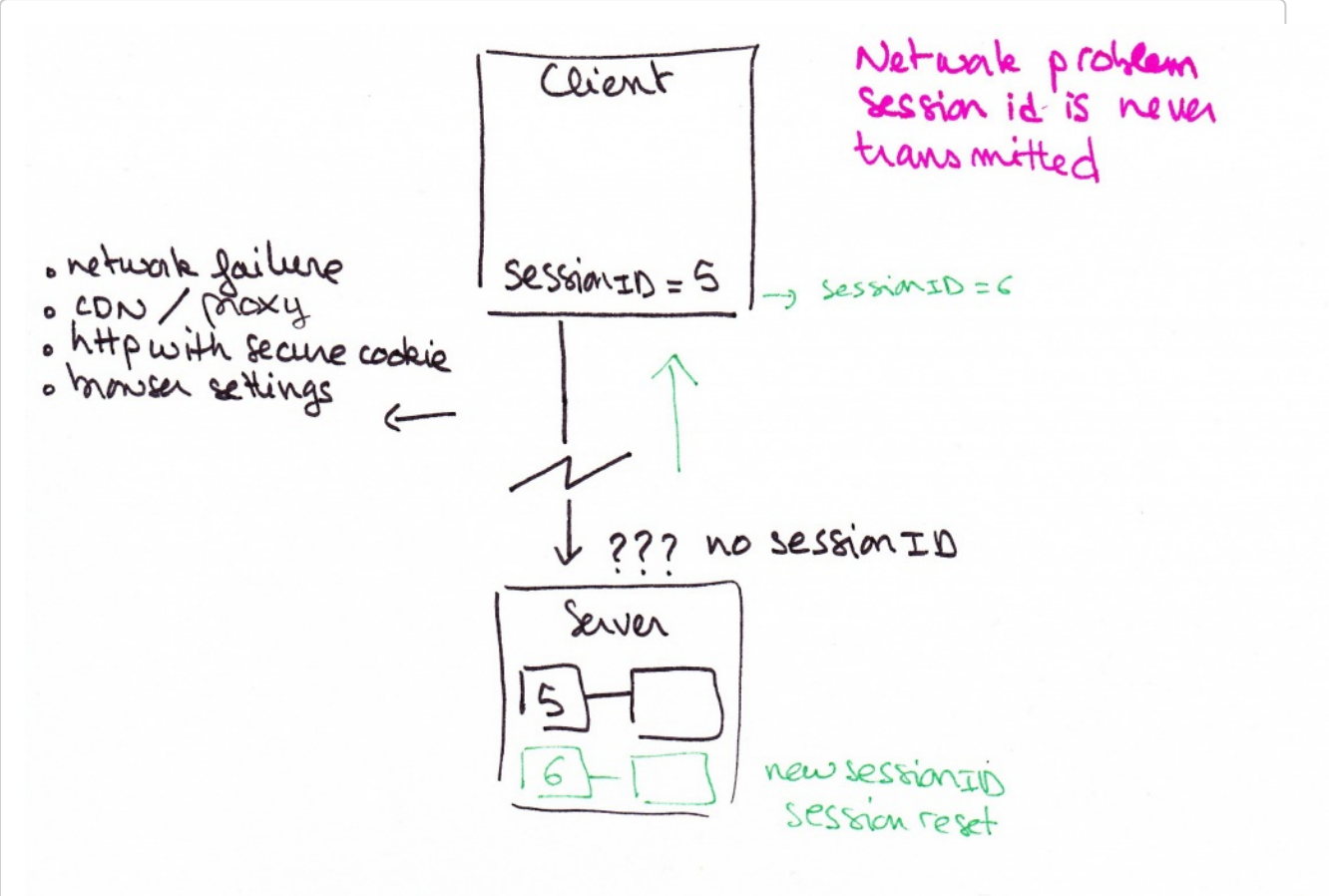
1. Bad session replication



Load balancing session reset

To debug this, you need to know your architecture well. In big architectures, there are several servers on the front arranged in clusters and users are load balanced on one of them at each request. You need to make sure that you share the session data correctly across those servers. Here, it is possible that your session id was set by server A, but when you're load balanced to server B, for all that server B knows, it has never seen you, so it doesn't have the session id you're presenting and will reset your session.

2. Transmission problem



Session id is lost

The main reason for a server to reset your session is because it doesn't recognize the session id you're presenting and therefore creates a new one to identify your connection. The session id is missing from your

request, you need to find where it got lost.

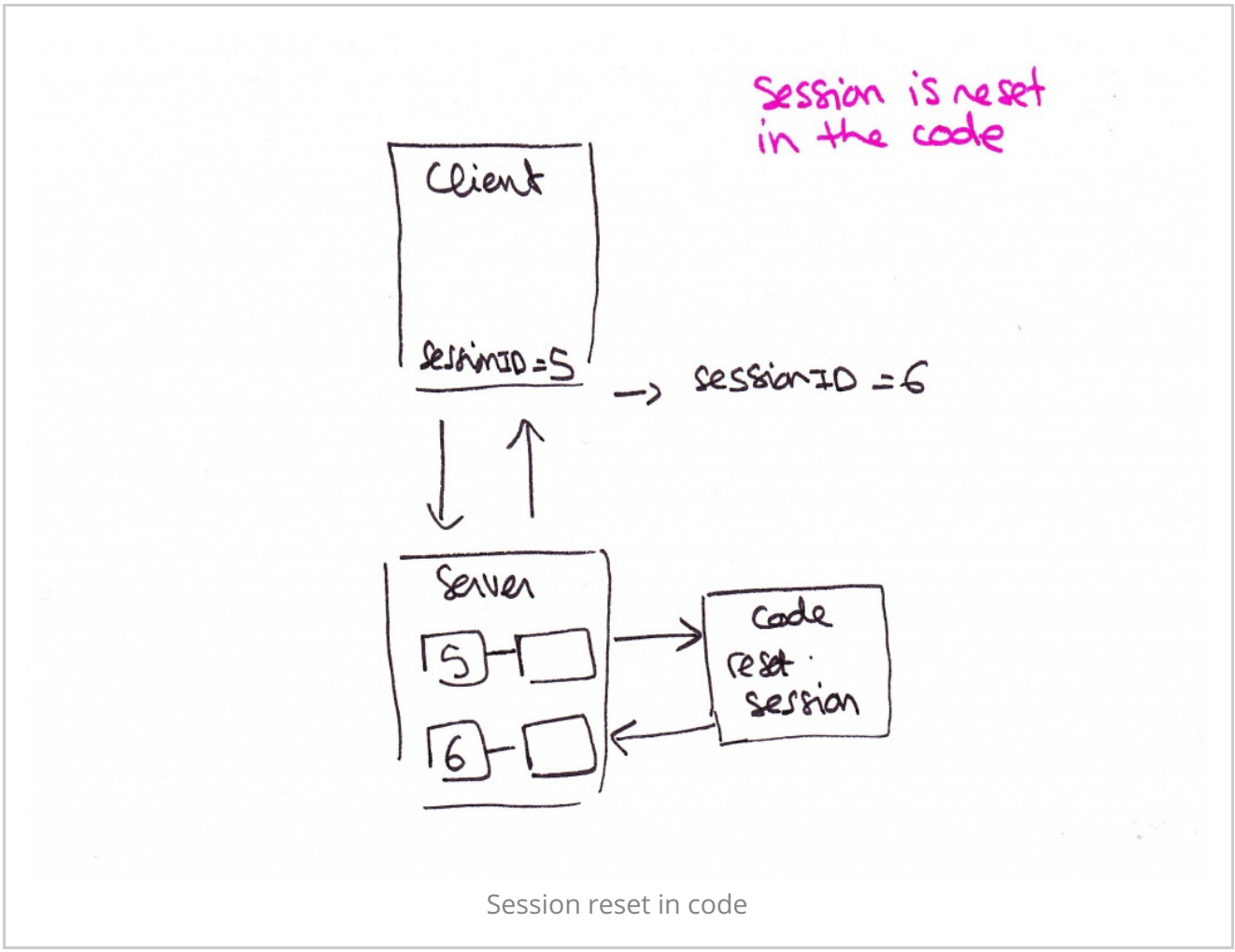
On the way from the browser to the server, there are several places where the session id could have gotten lost :

– on the browser : check how the session id is transmitted, if it's in the cookies check that the cookie transmission is ok with your browser. Maybe the browser doesn't allow cookies in its settings for example. Also, be careful to not have http links on a https website with secure cookies. This was my bug. It seems a classic now, but it was not that easy to detect. Because the cookies were secure, the browser never transmitted the cookie over the http connection, because it only allowed cookie transmission over https, hence session reset. I advise you to look out for : https, secure cookies and redirects, those will be good pointers to the cause of your bug.

– on the network : check that you don't have any CDNs or other proxy that trim your cookies on the way.

– on the server : the server doesn't read or write session ids correctly.

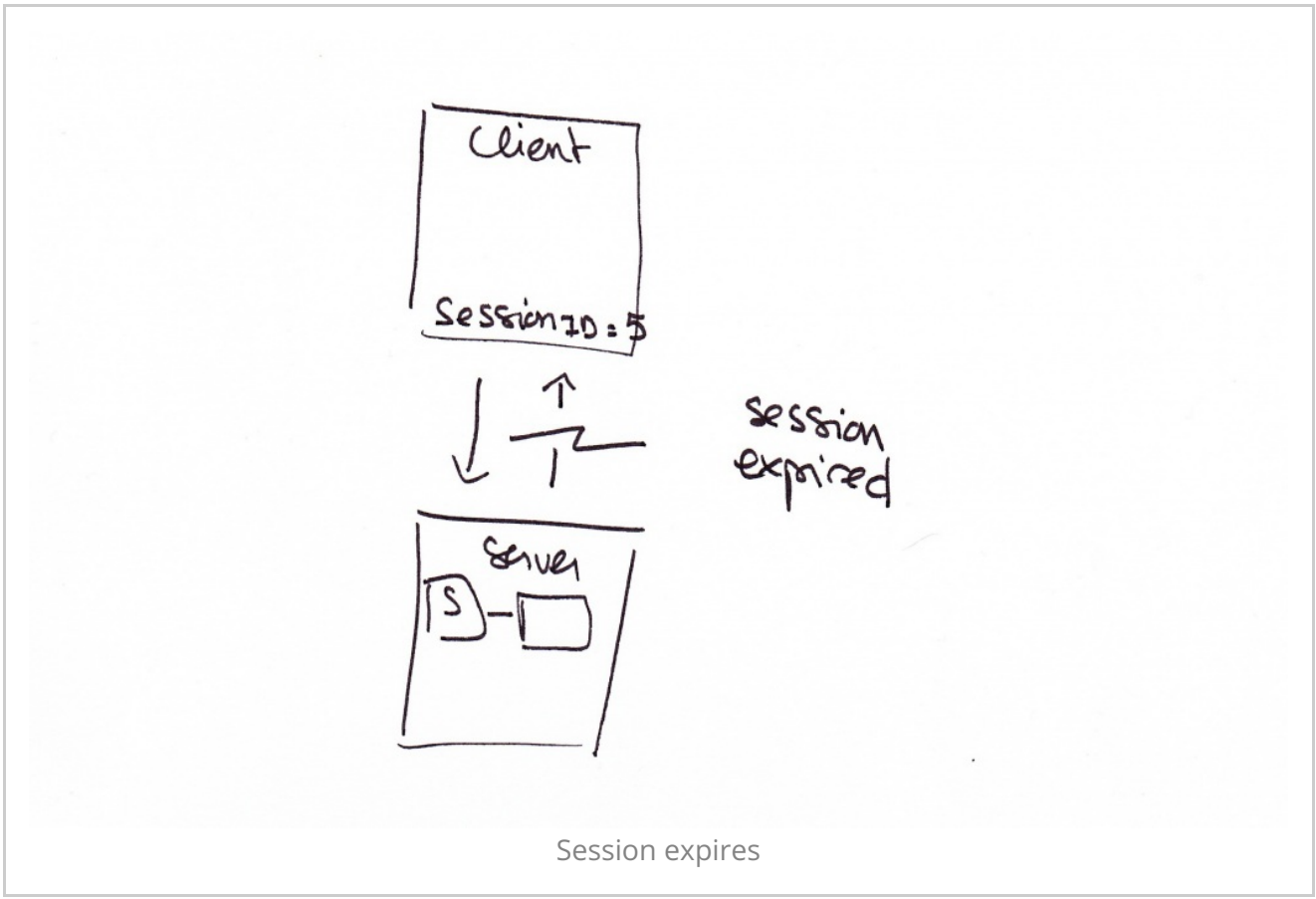
3. Session reset in code



Session reset in code

Of course you should not rule out this possibility. Maybe some code is intentionally asking your server to reset the session, like when you say $user->logout(). This one should not be very difficult to find in the code.

4. Session expired



Session expires

Usually your session is destroyed only when you close the connection, so when you close your browser. Some servers might have specific directives to reset the session after a timeout before you close the connection, you should check that out.

***

So this is what I wish I could find when I was investigating my bug, here it is now. Those are the main reasons I can find for session reset, I have tried most of them before finding out about the secure cookies... Go ahead and smile. You can always read about the full story of my bug here : Rings, bells and victory.