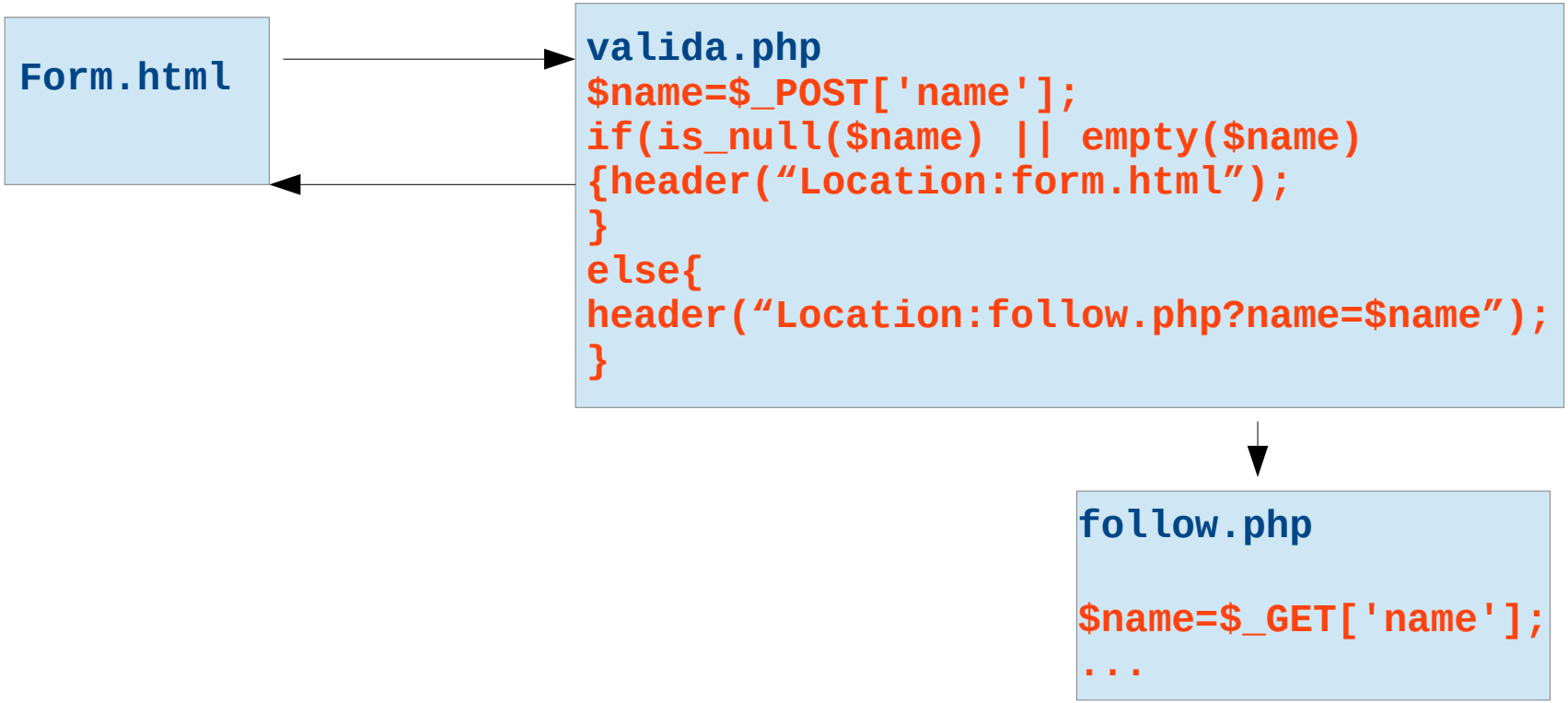
FORM VALIDATION: THREE WAYS

1- FIRST WAY: THREE PAGES

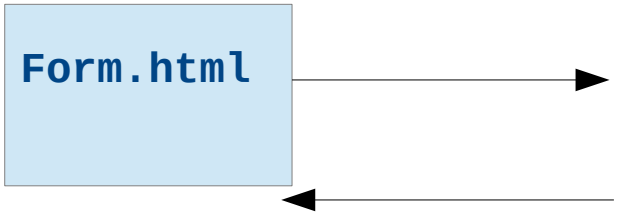
The idea is to have a specific file for validation, and then redirect navigation to the corresponding page, with header(...)



FORM VALIDATION: THREE WAYS

2-SECOND WAY: TWO PAGES

We can also do the validation on the page we want to go to and if the data are not correct, return to the initial form.



follow.php

```
$name=$_POST['name'];
if(is_null($name) || empty($name)
{
Error message
Return to the form in a Return button
}
else
{
Do what you want
}
```

3-THIRD WAY: ONE PAGE

Finally, we can make the validation in the same form.

FORM VALIDATION FIRST WAY

We can do the redirection with the “header” instruction and the “location” clause

```
header("Location:index.html");
```

It is important that the redirection page does not show anything on the screen

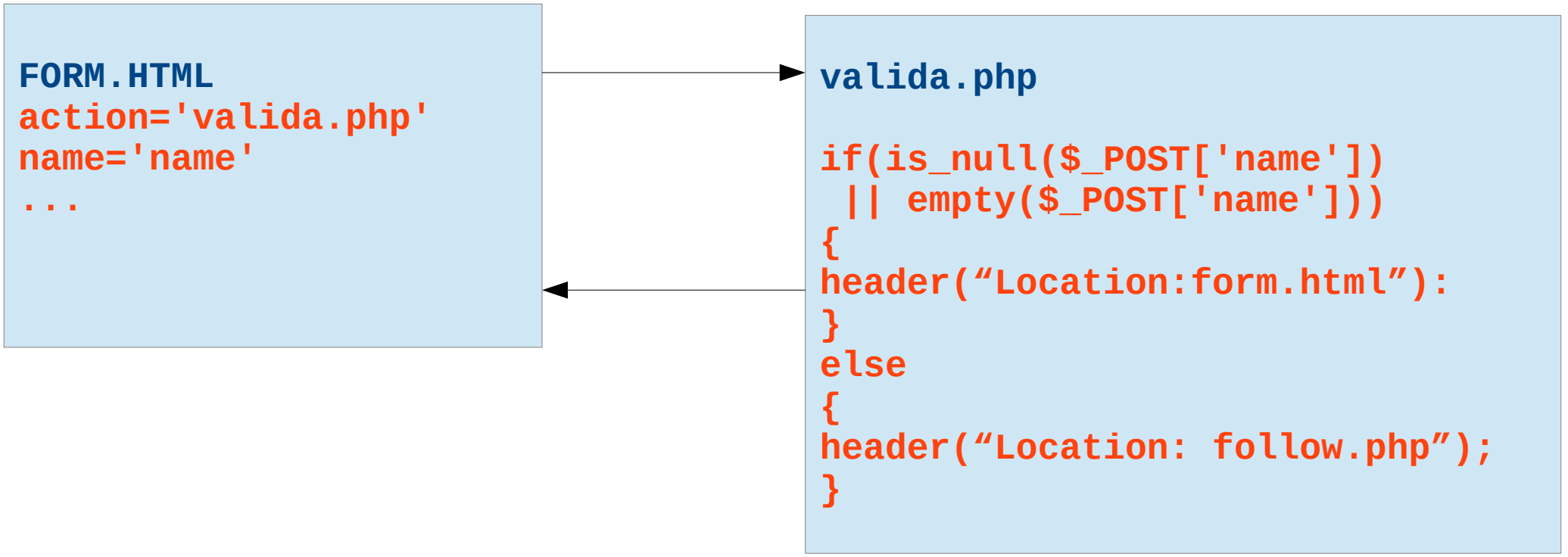
FORM VALIDATION FIRST WAY

```
FORM
action='valida.php'
name='name'
...
```

```
valida.php

if(is_null($_POST['name'])
 || empty($_POST['name']))
{
header("Location:error.php"):
}
else
{
header("Location: follow.php");
}
```

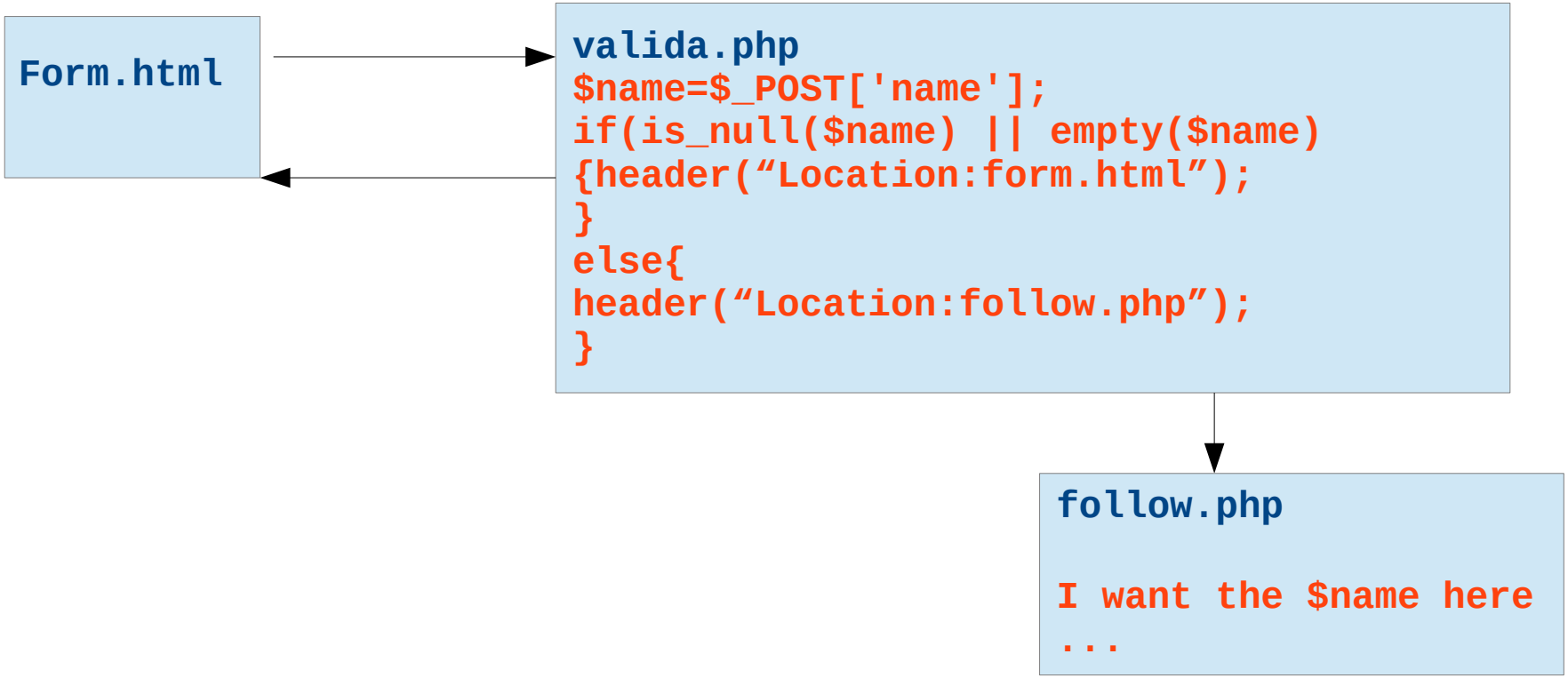
FORM VALIDATION FIRST WAY



Many times after a validation error we go back to the initial form to fill in the values again

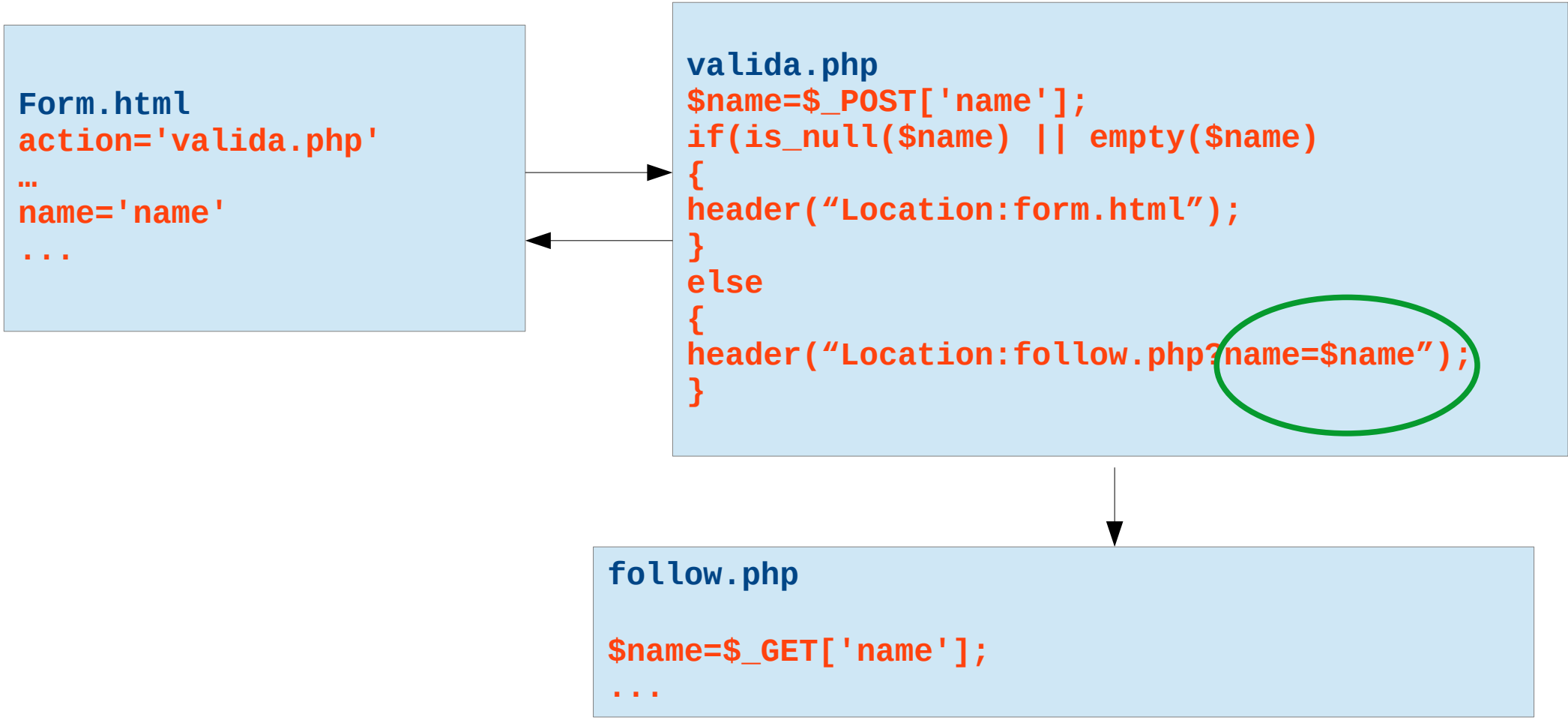
FORM VALIDATION FIRST WAY

The problem comes when we want the input parameters we have passed from the form to the php validation can be transferred to a third php file (e.g. the follow.php)
Because $_POST does not exist in follow.php



FORM VALIDATION FIRST WAY: Parameters with header (GET)

We can pass the parameters in the header attached to the URL of the page, as in a GET type form



FORM VALIDATION FIRST WAY: Parameters with header (GET)

The problem of passing parameters by header is that they must be done as GET parameters, and therefore are visible in the URL that shows the browser window

There is another SOLUTION 2: using SESSION variables

FORM VALIDATION FIRST WAY: Using SESSION VARIABLES

A session provides a way to make variables accessible across multiple web pages.

To begin a session, the session_start function is used. This function must appear before any output is sent to the web page.

```
<?php session_start(); ?>
```

With the session started, the $_SESSION array is used to store session data as well as retrieve it.

FOR EXAMPLE, to create a session variable called “password” with the content of the $pw variable

```
$_SESSION[“password”]=$pw;
```

FORM VALIDATION FIRST WAY: Using SESSION VARIABLES

TO READ THE SESSION VARIABLE WE USE THE ARRAY $_SESSION

```
if($_SESSION[“password”]==”1234”)
```

For removing all session variables, there is the session_destroy function.

```
session_destroy();
```

KEEPING VALUES ON FORMS

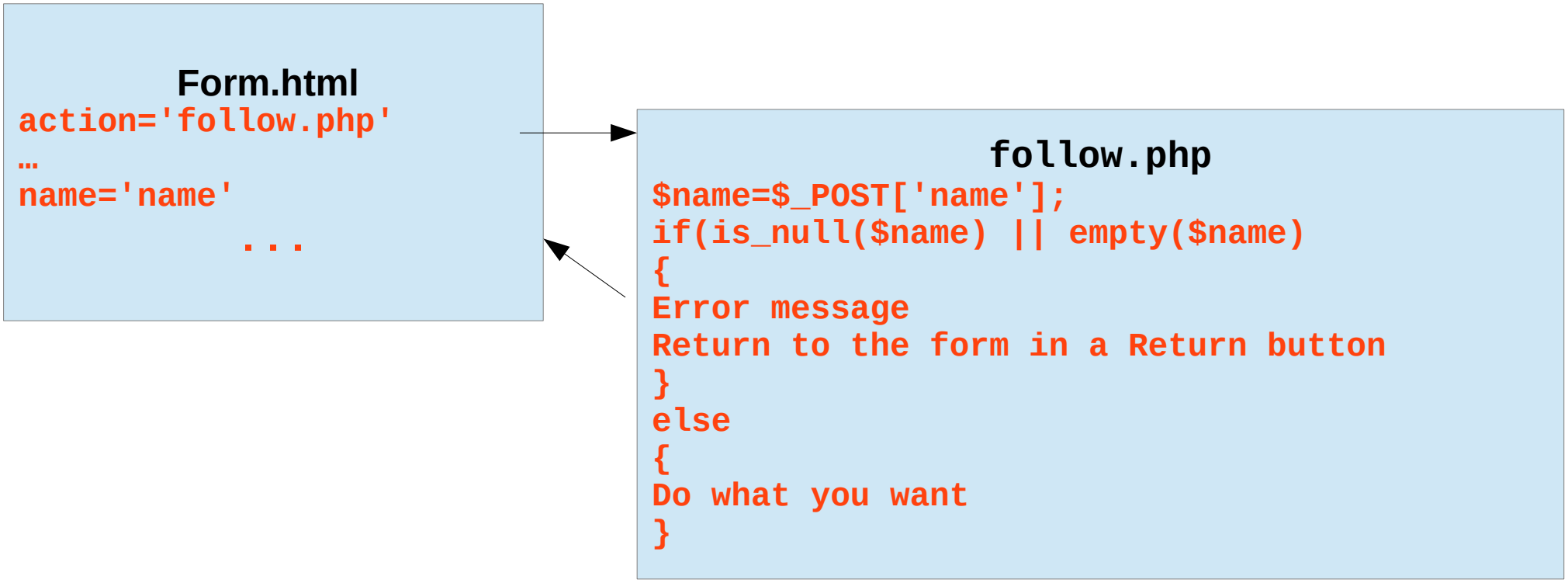
With session variables, when we enter an incorrect value, we return to the form with the values previously entered in the inputs.

Form.php

```
<?php session_start();
$month=''; $year='';
if(isset($_SESSION['month'])&&isset($_SESSION['year']))
{
    $month=$_SESSION['month'];
    $year=$_SESSION['year'];
}
…
<form action="valida.php" method="POST">
 Month<input type="text" name="month" value="<?echo $month;?>"/>
 Year<input type="text" name="year" value="<?echo $year;?>"/>
```

FORM VALIDATION SECOND WAY

In this case we will work without the validation page (valida.php), so we will go directly to follow.php. Now it's not necessary to use $_SESSION or GET in header to pass the variables to a third file



SECOND WAY: KEEPING VALUES ON FORMS

In this case, if the data are not correct and we want to return to the initial form keeping values, we can use SESSION variables as in slide 7 or we can use a form with $_POST variables BUT HIDDEN, TO PASS THE CORRECT VALUES(we usually do the second way when there is more than one variable)

Form.php

```
<form action="follow.php" method="POST">
   Month<input type="text" name="month" value="<?php
if(isset($_POST["month"])) echo $_POST["month"];?>"/>
   Year<input type="text" name="year" value="<?php
if(isset($_POST["year"])) echo $_POST["year"];?>"/>
</form>
```

follow.php: If the month is incorrect, we pass year as hidden

```
echo 'incorrect month';
  <form action="form.php" method="post">
  <?php
 echo "<INPUT TYPE='hidden' NAME='year' VALUE='".$_POST["year"]."'>";
  ?>
  <INPUT TYPE='submit' VALUE='Return'>
  </form>
```

SECOND WAY: KEEPING VALUES ON FORMS

follow.php

```
<?php
$name=$_POST['name'];
$surname=$_POST['surname'];
if (is_null($name) || empty($name)||is_null($surname) || empty($surname))
{
?>
<form action="form.php" method="post">
   You didn't write anything
   <?php
 echo "<INPUT TYPE='hidden' NAME='name' VALUE='$name'>";
 echo "<INPUT TYPE='hidden' NAME='surname' VALUE='$surname'>";
?>
<INPUT TYPE='submit' VALUE='Return'>
</form>
<?php
}
else
{
echo "Hello, $name";
?>
<form action="form.php">
  <INPUT TYPE='submit' VALUE='Return'>
</form>
 <?php
}
?>
```

Keep values with hidden and return to the form

If all goes well, and you don't want to keep the values, we go back to the form but without passing the hidden variables.

FORM VALIDATION THIRD WAY

The form sends the data to itself, and we check if the data exist before deciding what to do

follow.php

```
<?php
if(isset($_POST['variable']) {
// we do something with the data
}
// THE FORM MIGHT BE INSIDE THE ELSE SECTION, OR OUTSIDE THE IF...ELSE
?>
<form action="<?php echo $_SERVER['PHP_SELF'];?>"
METHOD="POST">
…
</form>
```

COOKIES INTRODUCTION

The HTTP protocol is a stateless protocol.
In a web application it is almost always necessary to maintain the state of the session

What is a Cookie?

A cookie is often used to identify a user. A cookie is a small file that the server embeds on the user's computer. Each time the same computer requests a page with a browser, it will send the cookie too. With PHP, you can both create and retrieve cookie values.

Create Cookies With PHP

A cookie is created with the setcookie() function.This function must be called before any output is sent to the browser.

```
setcookie(name, value, expire, path, domain, secure, httponly);
```

Only the name parameter is required. All other parameters are optional. The value of the cookie is stored on the clients computer; Assuming the name is 'lastvisit', this value is retrieved through $_COOKIE['lastvisit']

```
setcookie("lastvisit", date("H:i:s"), time() + 60*60);
```

In this example, the cookie expires after one hour (in seconds), and the value is the current moment.

WORK WITH COOKIES

Retrieve a Cookie

It can then be accessed through the $_COOKIE array.

```
if (isset($_COOKIE['lastvisit']))
echo "Last visit: " . $_COOKIE['lastvisit'];
```

Modify a Cookie Value

To modify a cookie, just set (again) the cookie using the setcookie() function

Deleting Cookies

To delete a cookie, use the setcookie() function with an expiration date in the past:

```
setcookie("lastvisit", 0, 0);
```