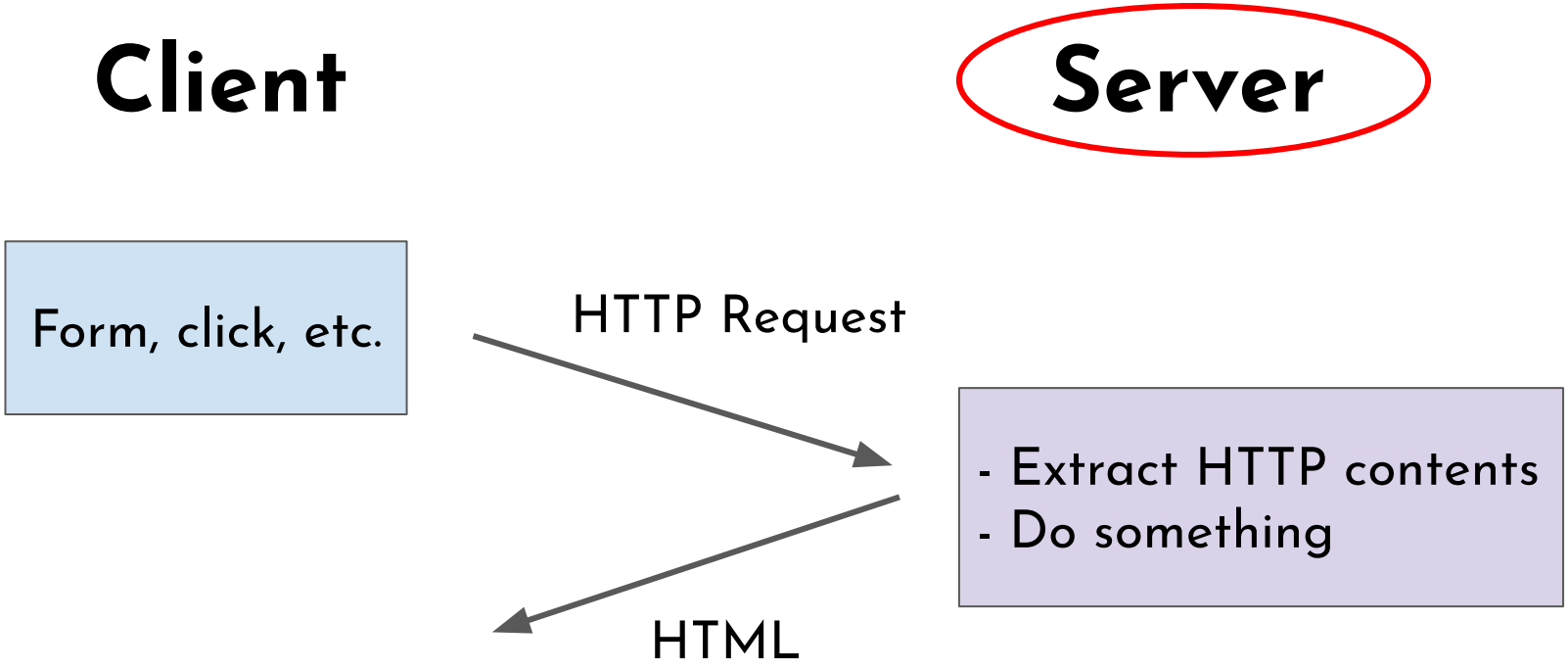
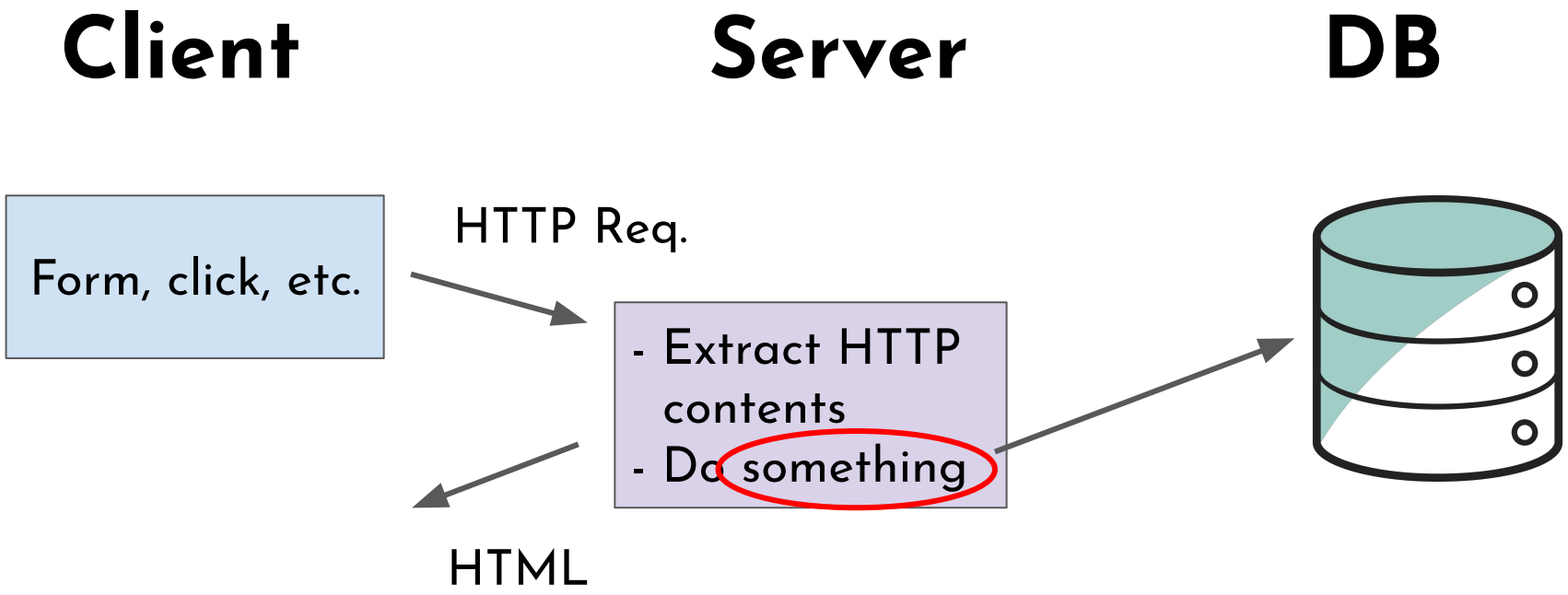
You made a mistake today

PHP

And from now on, you will see me in your nightmares...


MISERY
GUILTY
PLEASRE


Client
Server
Form, click, etc.
HTTP Request
- Extract HTTP contents
- Do something
HTML


Client
Server
DB
Form, click, etc.
HTTP Req.
- Extract HTTP contents
- Do something
HTML

Let me tell you a story...

Escaping MySQL Strings

mysql_escape_string

(PHP 4 >= 4.0.3, PHP 5)

mysql_escape_string — Escapes a string for use in a mysql_query

```
mysql_escape_string(string $unescaped_string): string
```

mysql_real_escape_string

```
mysql_real_escape_string(string $unescaped_string, resource $link_identifier = NULL): string
```

mysql_real_escape_string

```
mysql_real_escape_string(string $unescaped_string, resource $link_identifier = NULL): string
```

mysql_real_escape_string

```
mysql_real_escape_string(string $unescaped_string, resource $link_identifier = NULL): string
```

link_identifier

The MySQL connection. If the link identifier is not specified, the last link opened by mysql_connect() is assumed. If no such link is found, it will try to create one as if mysql_connect() had been called with no arguments. If no connection is found or established, an **E_WARNING** level error is generated.

Description

Object-oriented style

```
public mysqli::real_escape_string(string $string): string
```

Procedural style

```
mysqli_real_escape_string(mysqli $mysql, string $string): string
```

```
"SELECT * FROM users WHERE user='" . $_POST[username] . "' AND
password='" . $_POST[password] . "'";
```

```
"SELECT * FROM users WHERE user='" .
mysql_real_escape_string($_POST[username]) . "' AND password='" .
mysql_real_escape_string($_POST[password]) . "'";
```

PHP: A fractal of bad design

Decoding JSON

- `json_decode` returns null for invalid input, even though null is also a perfectly valid object for JSON to decode to—this function is *completely unreliable* unless you also call `json_last_error` every time you use it.

Substring Search

```
strpos(string $haystack, string $needle, int $offset = 0): int|false
```

Find the numeric position of the first occurrence of **needle** in the **haystack** string.

Substring Search

```
strpos(string $haystack, string $needle, int $offset = 0): int|false
```

Find the numeric position of the first occurrence of **needle** in the **haystack** string.

Substring Search

```
strpos(string $haystack, string $needle, int $offset = 0): int|false
```

Find the numeric position of the first occurrence of **needle** in the **haystack** string.

Warning This function may return Boolean **false**, but may also return a non-Boolean value which evaluates to **false**. Please read the section on Booleans for more information. Use the === operator for testing the return value of this function.

Substring Search

```
strpos(string $haystack, string $needle, int $offset = 0): int|false
```

Find the numeric position of the first occurrence of **needle** in the **haystack** string.

"If you use `false` as an index, or do much of anything with it except compare with `===`, PHP will silently convert it to `0` for you. Your program will not blow up; it will, instead, do the *wrong thing* with *no warning*, unless you remember to include the right boilerplate around every place you use `strpos`"

Equality and Comparison

`===` compares values and type... except with objects, where `===` is only true if both operands are actually the same object! For objects, `==` compares both value (of every attribute) and type, which is what `===` does for every *other* type.

Equality and Comparison

`===` compares values and type... except with objects, where `===` is only true if both operands are actually the same object! For objects, `==` compares both value (of every attribute) and type, which is what `===` does for every *other* type.

For a more type-safe `==`, we have `===`. For a more type-safe `<`, we have... nothing. `"123" < "0124"`, always, no matter what you do. Casting doesn't help either

Equality and Comparison

`===` compares values and type... except with objects, where `===` is only true if both operands are actually the same object! For objects, `==` compares both value (of every attribute) and type, which is what `===` does for every *other* type.

For a more type-safe `==`, we have `===`. For a more type-safe `<`, we have... nothing. `"123" < "0124"`, always, no matter what you do. Casting doesn't help either

---

`NULL < -1`, *and* `NULL == 0`

Indexing

`[ ]` can be used on any variable, not just strings and arrays. It returns null and issues no warning.

Indexing

`[ ]` can be used on any variable, not just strings and arrays. It returns null and issues no warning.

You can also use `{ }`

Modules (jk)

`include()` and friends are basically C's `#include`: they dump another source file into yours. There is no module system, even for PHP code.

Error Handling

- Trying to access a non-existent object property, i.e., $foo->x, is a **warning**.
- But trying to access a non-existent class construct, i.e., $foo::x, is a **fatal error**.

Error Handling

At least a dozen functions for getting the last error from a particular subsystem (see below), even though PHP has had **exceptions** for eight years.

Error Handling

Error Handling

Btw, PHP doesn't have stack traces. So, good luck if you're trying to figure out the source of an error.

A Quick Joke

`(int)` is a **single token.**

Constants

PHP has the equivalent
of #define!

Constants

PHP has the equivalent of #define!

```php
<?php
define('MAXNUM', 1000);

for ($i = 2; $i < MAXNUM; ++$i) {
...
}
```

If you report an undefined behavior bug, a common reaction from software developers is “So what? Our code works just fine.” As a random example, here is a discussion I had with Rasmus Lerdorf about five years ago about some UBs in the PHP interpreter. One might point out that it wasn’t a very mature exchange but I wasn’t even 40 yet at the time. (Earlier I had an example here from OpenSSL but this one is more suitable.)

– John Regehr

Easter Egg: Printing

PHP provides... a bunch
of ways to print.

```
echo "Test<br/>";
```

Easter Egg: Printing

PHP provides... a bunch of ways to print.

```
echo "Test<br/>";
print "Test <br/>";
```

Easter Egg: Printing

PHP provides... a bunch of ways to print.

```
echo "Test<br/>";
print "Test <br/>";
print("Test<br/>");
```

Easter Egg: Printing

PHP provides... a bunch of ways to print.

```php
echo "Test<br/>";
print "Test <br/>";
print("Test<br/>");
$city = "Chicago";
printf("The city: %s<br/>", $city);
```

Easter Egg: Printing

PHP provides... a bunch of ways to print.

```php
echo "Test<br/>";
print "Test <br/>";
print("Test<br/>");
$city = "Chicago";
printf("The city: %s<br/>", $city);
$foo['a'] = "blah";
$foo['b'] = "bar";
print_r($foo);
```