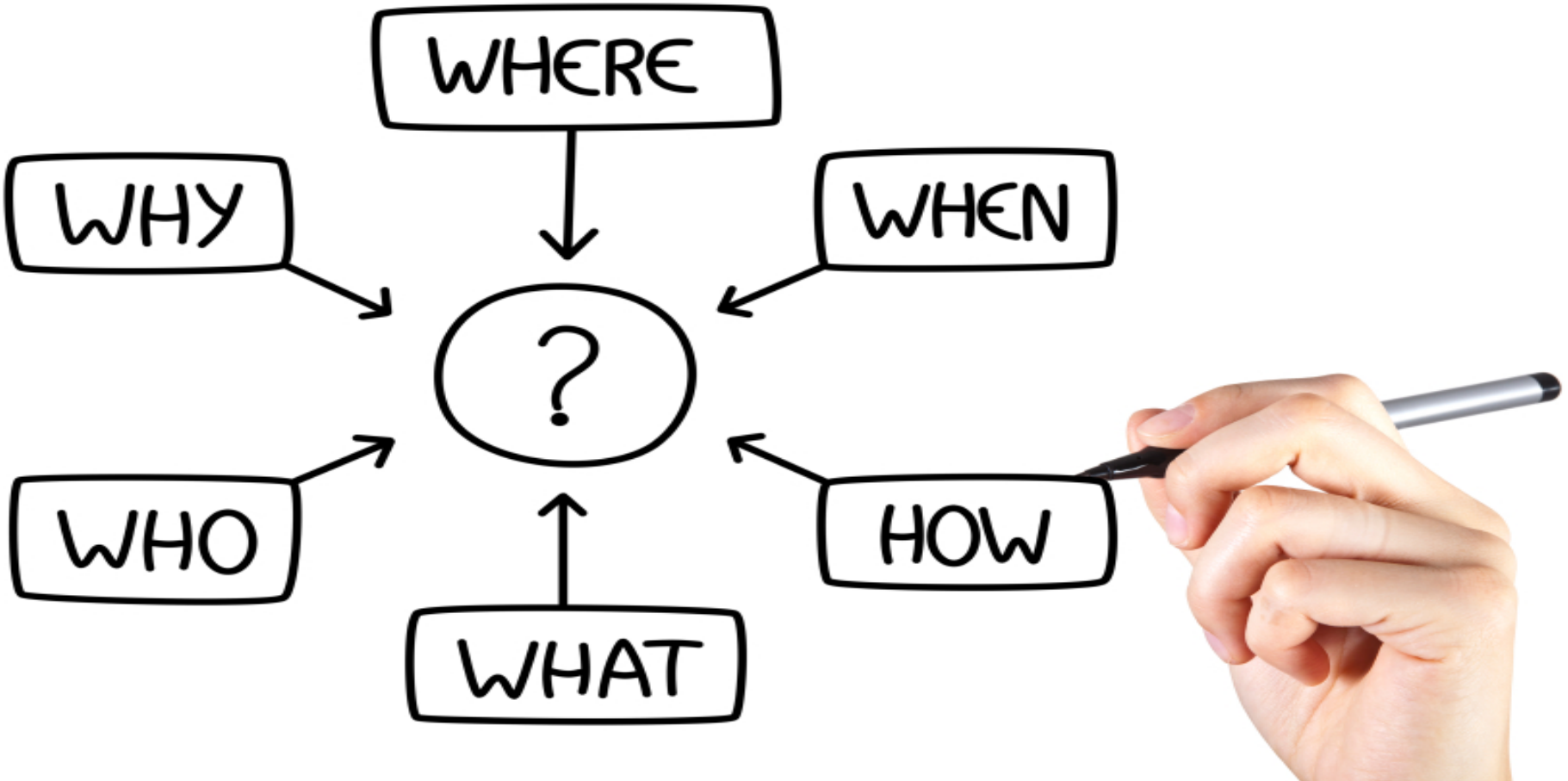
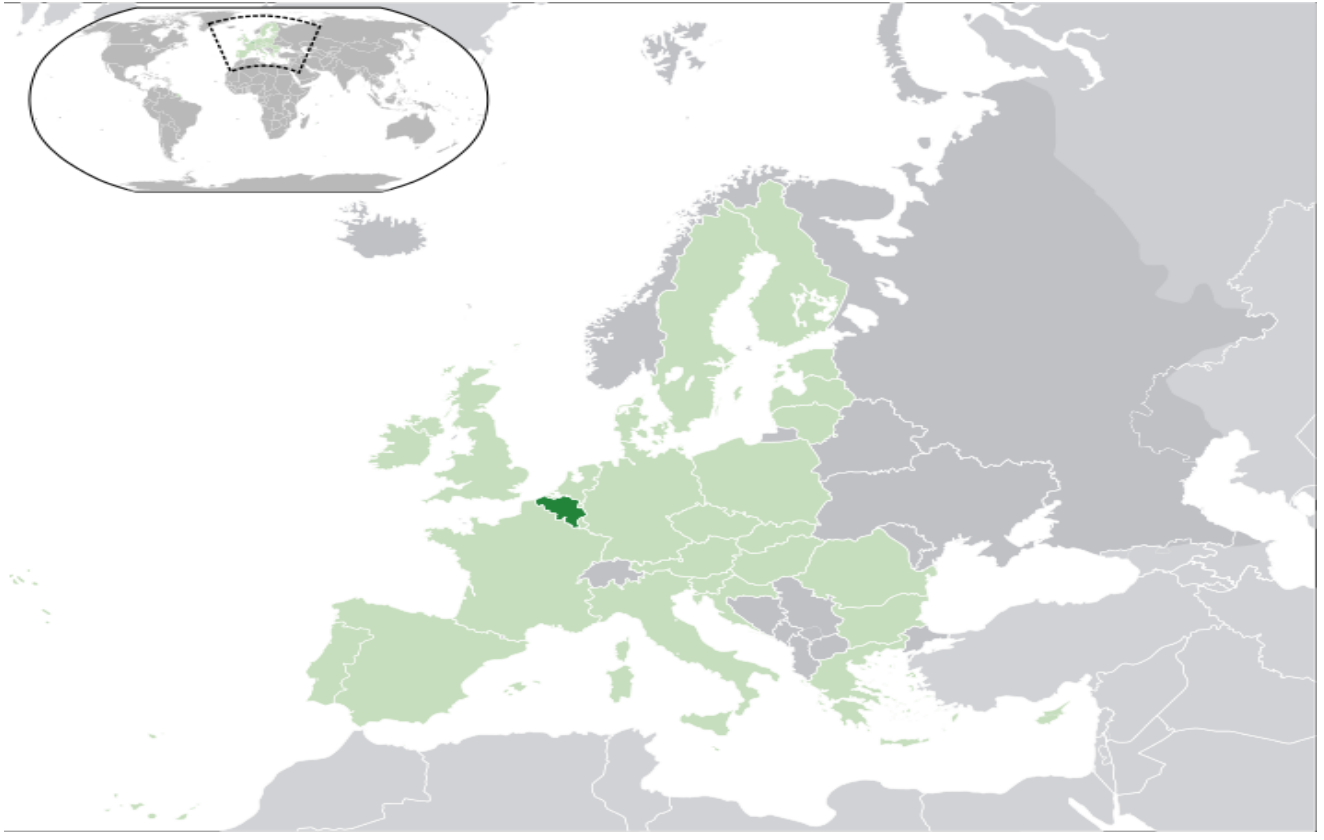
The Why and How of moving to PHP 7.x



Who am I ?

- Wim Godden (@wimgtr)

Where I'm from



Where I'm from

Where I'm from

Where I'm from

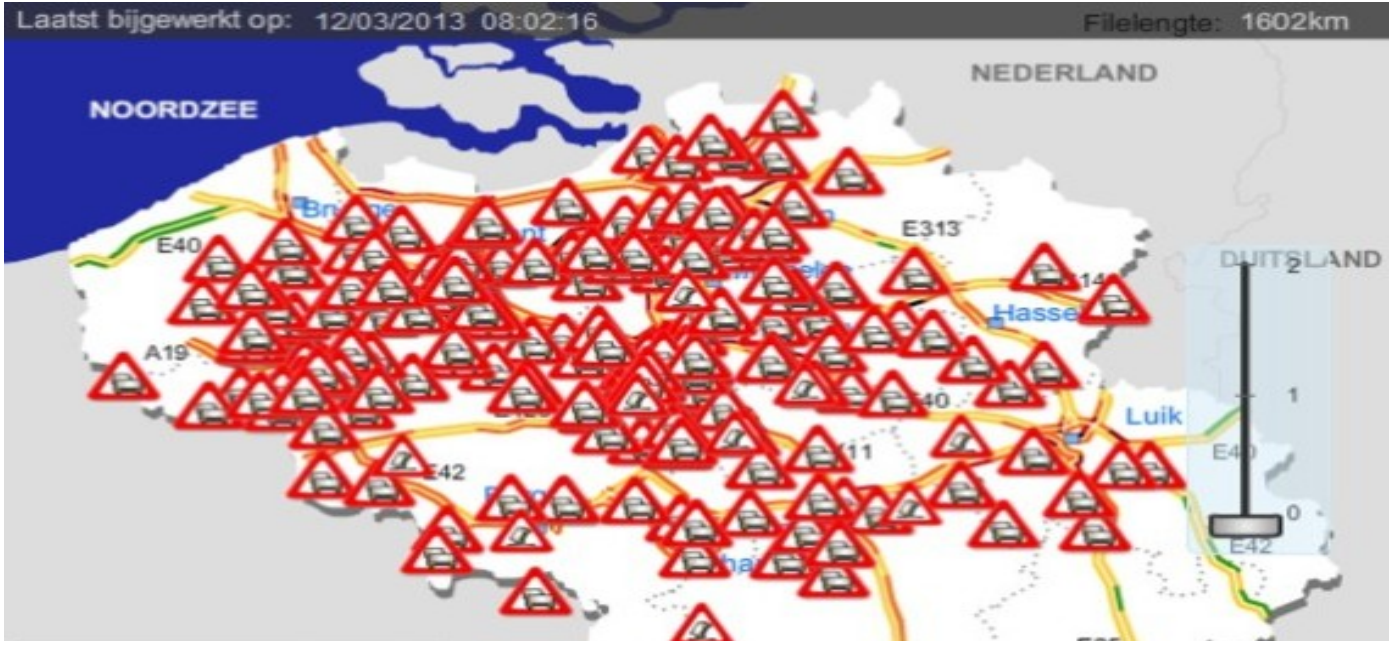
Where I'm from

Where I'm from

My town

My town

Belgium – the traffic



Who am I ?

- Wim Godden (@wimgtr)
- Founder of Cu.be Solutions (http://cu.be)
- Open Source developer since 1997
- Developer of PHPCompatibility, OpenX, ...
- Speaker at Open Source conferences

Why vs How

Part 1 : why upgrade ?

Bad reasons :

- It's cool to have the latest version
- Annoy sysadmins
- Oh cool, a new toy !

Part 2 : how to upgrade ?

The nightmare of compatibility

The joy of automation

No miracles here !

Show of hands

- 3 / 4
- 5.0
- 5.1
- 5.2
- 5.3
- 5.4
- 5.5
- 5.6
- 6.0
- 7.0
- 7.1
- 7.2
- 7.3
- 7.4
- 8.0

The numbers

W3Techs (http://w3techs.com/technologies/details/pl-php/all/all)

	Now	Oct 2018	Jan 2015
PHP 4 :	0.4%	0.8%	1.8%
PHP 5 :	55.2%	87.2%	98.2%
5.0 :	< 0.1%	< 0.1%	0.1%
5.1 :	0.4%	0.5%	1.2%
5.2 :	6.3%	7.8%	19.2%
5.3 :	15.1%	19.7%	45.5%
5.4 :	17.6%	21.2%	26.9%
5.5 :	10.9%	15.4%	6.3%
5.6 :	49.8%	35.3%	0.5%
PHP 7 :	44.3%	12.0%	
7.0 :	23.5%	66.8%	
7.1 :	21.1%	31.2%	
7.2 :	36.3%	2.1%	
7.3 :	19.0%		
7.4 :	0.2%		

5.3 – 5.6 quick recap

- Namespaces (\)
- Late static binding
- Closures
- Better garbage collection
- Goto
- Mysqlnd
- Performance gain
- Short array syntax
- Traits
- Built-in webserver
- Binary notation
- No more register_globals, magic_quotes_gpc and safe_mode
- Generators (yield keyword)
- password_hash() function
- Built-in opcache

PHP 7.x – what's changed ?

- New features
- Performance and memory usage
- Improved consistency
- Lots of things removed or deprecated

New things – Scalar type + return type declarations (7.0)

```php
function someFunction(int $i, string $s) : bool {
}
```

- New scalar types : int, float, bool and string
- Return type can be specified as well
  (and can be scalar type as well)

Weak and strong typing

- Weak :
  - Default
  - Parameters will be coerced (PHP 5 style)
- Strong/strict :
  - At top of file :

```
declare(strict types=1);
```

  - Strict typing is file-specific, so must be enabled in each file !
  - If wrong type is given, a TypeError is thrown :

```
Fatal error: Uncaught TypeError: Return value of testFunction() must be
of the type integer, string returned
```

  - Returning null is also invalid → if you want an int, you will get an int or an error

Null coalescing operator (??)

- PHP 5 :

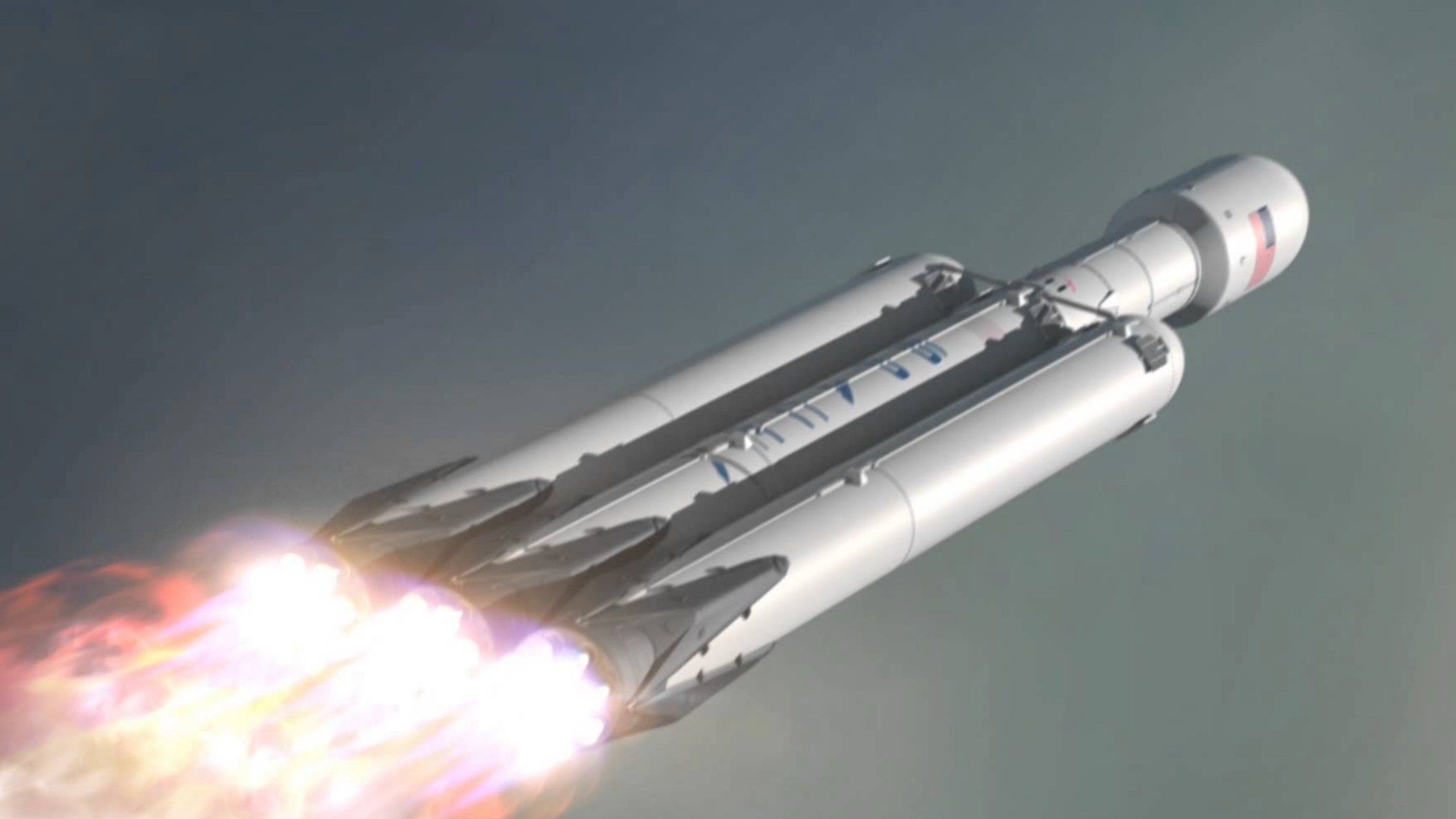
```php
$name = isset($_GET['name']) ? $_GET['name'] : 'anonymous';
```

- PHP 7 :

```php
$name = $_GET['name'] ?? 'anonymous';
```

- Can be chained :

```php
$name = $ GET['name'] ?? $ POST['name'] ?? 'anonymous';
```



Spaceship operator (<=>)

- Compares expressions
- Returns -1, 0 or 1
- Examples :

```php
echo 1 <=> 1; // 0
echo 1 <=> 3; // -1
echo 5 <=> 2; // 1

echo "a" <=> "a"; // 0
echo "a" <=> "z"; // -1
echo "z" <=> "a"; // 1
```

Unicode codepoint escape syntax

- Converts hexadecimal Unicode codepoint to UTF8 double quoted string
- ```
  echo "\u{2615}";
  ```

  will output :

  
  ```

Filterable unserialize()

- Defines which classes can be unserialized
- New 2nd parameter
- Examples :

```
$data = unserialize($foo, ["allowed_classes" => false]);
```

returns __PHP_Incomplete_Class objects

```
$data = unserialize($foo, ["allowed_classes" => ["Article", "User"]]);
```

returns objects of type Article, User, or __PHP_Incomplete_Class

CSPRNG functions

- Cross platform functions to generate random data
- Cryptographically secure
- 2 functions :
  - random_bytes($length)
  - random_int($min, $max)

Deprecated in PHP 7.0

- PHP 4 style constructors

```php
class foo() {
    function foo() {
        // do something
    }
}
```

- Static calls to non-static methods

```php
class foo {
    function bar() {
        echo 'I am not static!';
    }
}

foo::bar();
```

Error handling in PHP 7

- Most fatal errors in PHP 5 → Exceptions in PHP 7
- New class : *Error*
  - If your PHP 5 code has a class called *Error*, you will need to rename it
- All *Error* and *Exception* classes now implement *Throwable*
- Error Flow :
  - *Error* is thrown
  - Bubbles up through called functions/methods
  - At first matching *catch* block, code is run
  - No matching *catch* → default exception handler
  - No default exception handler → Fatal error

- Error
  - ArithmeticError
    - DivisionByZeroError
  - AssertionError
  - ParseError
  - TypeError
    - ArgumentCountError
- Exception
  - ClosedGeneratorException
  - DOMException
  - ErrorException
  - IntlException
  - LogicException
    - BadFunctionCallException
      - BadMethodCallException
    - DomainException
    - InvalidArgumentException
    - LengthException
    - OutOfRangeException
  - PharException
  - ReflectionException
  - RuntimeException
    - OutOfBoundsException
    - OverflowException
    - PDOException
    - RangeException
    - UnderflowException
    - UnexpectedValueException
  - SodiumException

Variable handling

- PHP 7 uses abstract syntax tree (AST)
- Old and new evaluation of indirect expressions

Expression	PHP 5 interpretation	PHP 7 interpretation
`$$foo['bar']['baz']`	`${$foo['bar']['baz']}`	`($$foo)['bar']['baz']`
`$foo->$bar['baz']`	`$foo->{$bar['baz']}`	`($foo->$bar)['baz']`
`$foo->$bar['baz']()`	`$foo->{$bar['baz']}()`	`($foo->$bar)['baz']()`
`Foo::$bar['baz']()`	`Foo::{$bar['baz']}()`	`(Foo::$bar)['baz']()`

- Can be detected automatically in some cases
- Requires manual fixing and testing

Removed extensions

- ereg
- mssql
- mysql
- sybase_ct
- mcrypt (PHP 7.1)

Other changes (1/2)

- Invalid octals now throw a ParseError

```
$octal = 0921;
```

```
PHP Parse error:  Invalid numeric literal in octal.php
```

- Negative bitshifts throw an ArithmeticError

```
echo (5 >> -2);
```

```
Fatal error: Uncaught ArithmeticError: Bit shift by negative number
```

- Division by zero throws DivisionByZeroError
- Hexadecimal strings are no longer numeric

```
var_dump(is_numeric("0xa3"));
```

In PHP 5 : `bool(true)`

In PHP 7 : `bool(false)`

Other changes (2/2)

- New reserved keywords : bool, float, int, null, string, true, false
- Reserved for future use : mixed, number, object, resource, void (7.1), iterable (7.1)
- However, keyword usage inside classes is less restrictive. This is now allowed :

```
class bla {
    function yield() {
    }
}
```

Performance and memory usage from 5.6 to 7.0

- Performance : 200 – 300% increase
  - How ?
    - Core optimizations
    - More direct approach
    - Fewer memory allocations
    - Smaller data structures across all data types
    - …
- Reduced memory usage : up to 50% !
  - Big impact on large frameworks
  - Even bigger impact on codebases such as Wordpress, Drupal, ...

PHP 7.0 → 7.1 (1/2)

- Nullable types

```php
function testMe(int $i) : ?int
{
    if ($i > 5) {
        return $i;
    } else {
        return null;
    }
}
```

- Exception on passing too few function arguments

```php
function test($param){}
test();
```

```
Fatal error: Uncaught ArgumentCountError: Too few arguments to
function test(), 0 passed in %s on line %d and exactly 1
expected in %s:%d
```

PHP 7.0 → 7.1 (2/2)

- Empty index operator on a string throws a fatal error

```
$str[] = $something;
```

- DateTime constructor now uses microseconds
- SSLv2 stream support has been dropped

PHP 7.1 → 7.2

- *object* type is available call parameter type and return type of any objects

```php
function test(object $obj) : object
{
    return new SplQueue();
}

test(new StdClass());
```

- Sodium extension added : modern cryptographic library
- TLS version used is now 1.0, 1.1 or 1.2 (instead of 1.0 only)
- *create_function()* is deprecated
- *__autoload()* is deprecated
- *each()* is deprecated

PHP 7.2 → 7.3 (1/2)

- Flexible heredoc and nowdoc

```
class test
{
    public $something = <<<EOT
        some content
    EOT;
}
```

- Trailing commas are allowed in function calls

```
someFunction(
    5,
    $foo,
    8,
);
```

- JSON_THROW_ON_ERROR option
  → json_decode() will throw exception if invalid JSON is encountered

PHP 7.2 → 7.3 (2/2)

- array_key_first() and array_key_last()
- Argon2 hashing algorithm
- continue in switch statement generates a warning

```php
foreach ($someArray as $item) {
    switch ($item) {
        case 1:
            continue;
    }
}
```

- Mysqli and PDO_Mysql now return fractional seconds for datetime, time and timestamp (!)
- Constants can no longer be defined as case-insensitive
- is_countable()

```php
if (is_array($foo) || $foo instanceof Countable) {
    // $foo is countable
}
```

```php
if (is_countable($foo)) {
    // $foo is countable
}
```

PHP 7.3 → 7.4 (1/2)

- Typed properties

```php
class User {
    public int $id;
    public string $name;
    public ?Company $company;
}
```

→ Will be checked on read/write

- Null Coalesce Assignment operator :

```php
$something ??= 'If something is null, this one is used';
```

- FFI (Foreign Function Interface) :
  - Allows PHP to talk directly to C libraries
  - Experimental
- Preloading
  - Like Opcache on steroids
  - Will preload all files and opcache them
  - Requires restart of web server / PHP-FPM to reload any changed files

PHP 7.3 → 7.4 (2/2)

- Operator precedence deprecation :

```php
$a = 5; $b = 3;
echo "Sum is : " . $a + $b;
```

Output : 3 !!!

In PHP 7.4 :

Deprecated: The behavior of unparenthesized expressions containing both '.' and '+'/'-' will change in PHP 8: '+'/'-' will take a higher precedence in test.php on line 4

- Deprecated curly brace syntax for accessing array elements :

```php
$a = [1, 2, 3, 4];
echo $a[2]; // Still valid
echo $a{2}; // Deprecated
```

So...

Should you upgrade today ?

Postponing upgrades - End-Of-Life

- In the past : we'll see
- Now : 2 years after initial release
- Critical security patches : 1 extra year (but no regular bugfixes)
- In practice
  - 7.1 was released Dec 2016 → already EOL (Nov 30 2019)
  - 7.2 was released Nov 2017 → already EOL (Nov 2019)
  - 7.3 was released Dec 2018 → EOL Dec 2020
  - 7.4 was released Nov 28 → EOL Nov 2021
- If you’re on PHP 7.0 - 7.2 → start upgrading !

Postponing upgrades

- Security
- Performance
- Framework support
  - Symfony 5 : PHP 7.2.5+
  - Zend Framework 3 : PHP 5.6+
  - Laravel 6 : PHP 7.2+
- Developer motivation

Upgrade paths

- 1 upgrade every 5 years
  - Knowledge of upgrade steps will be forgotten
  - Documentation is not very useful (for example : SysvInit → Systemd)
  - Massive task to upgrade all apps, all environments, all servers
- Upgrade every release
  - Upgrade steps can be automated
  - Can be integrated with continuous integration and continuous deployment
  - Documentation is in the automation flow

So you want to upgrade...

- Option 1 : run your unit tests
- Option 2 : visit each page (good luck !) + check error_log
  - Or : record visits, then replay log on test environment
  - Or : proxy requests to 2 environments
- Option 3 : automated static analysis

Back in 2010...

- PHP Architect @ Belgian Railways 
- 8 years of legacy code (4.x and 5.x)
- 40+ different developers
- 40+ projects

Challenge :
migrate all projects from
PHP 5.1.x (on Solaris)
to
PHP 5.3.x (on Linux)

The idea

- Automate it
- How ? → Use the CI environment
- Which tool ? → PHP_CodeSniffer

PHP_CodeSniffer

- Originally PEAR package (*pear install PHP_CodeSniffer*)
- Also on Composer now
- Detects coding standard violations
- Supports multiple standards
- Static analysis tool
  - → Runs without executing code
  - → Splits code in tokens
    - Ex. : T_OPEN_CURLY_BRACKET
      - T_FALSE
      - T_SEMICOLON
  - → Parses each file separately

PHP_CodeSniffer

Let's see what it looks like

PHP_CodeSniffer options

-i Show available standards
-p Show progress
-s Show real error/warning sniff names
-n Ignore warnings
-v Verbose
--parallel=*x* (since PHP_CodeSniffer 3)

PHPCompatibility

- PHP_CodeSniffer standard
- Only purpose : find compatibility issues
- Detects :
  - Deprecated functions
  - Deprecated extensions
  - Deprecated php.ini settings and ini_set() calls
  - Prohibited function names, class names, …
  - …
- Works for PHP ~~5.0~~ 5.3 and above (5.4 for PHP_CodeSniffer 3 support)

PHPCompatibility – making it work - Composer

- In require-dev : phpcompatibility/php-compatibility
- If PHPCompatibility is the only PHP CodeSniffer standard :

```
"scripts": {
    "post-install-cmd": "\"vendor/bin/phpcs\" --config-set installed_paths vendor/wimg/php-compatibility/PHPCompatibility",
    "post-update-cmd" : "\"vendor/bin/phpcs\" --config-set installed_paths vendor/wimg/php-compatibility/PHPCompatibility"
}
```

- Otherwise use one of these :
  - DealerDirect/phpcodesniffer-composer-installer
  - higidi/composer-phpcodesniffer-standards-plugin

PHPCompatibility – making it work - Github

- Download PHP CodeSniffer
- Download from http://github.com/phpcompatibility/PHPCompatibility
- Run *phpcs --config-set installed_paths /path/to/PHPCompatibility*

PHPCompatibility – making it work – testing and running

- Check if coding standard is available :

  phpcs -i

  Should output something similar to :

  ```
  The installed coding standards are MySource, PEAR,
  PHPCompatibility, PHPCS, PSR1, PSR2, Squiz and Zend
  ```

- To run :

  phpcs --standard=PHPCompatibility /path/of/your/code

Important notes

- Large directories → can be slow !
- Use --extensions=php,phtml,...
  No point scanning .js files
- Test PHP x.x compatibility → needs PHP x.x on the system
- Static analysis
  - Doesn't actually run the code
  - Can not detect every single incompatibility → some things only happen on runtime
  - Provides filename and line number

PHPCompatibility

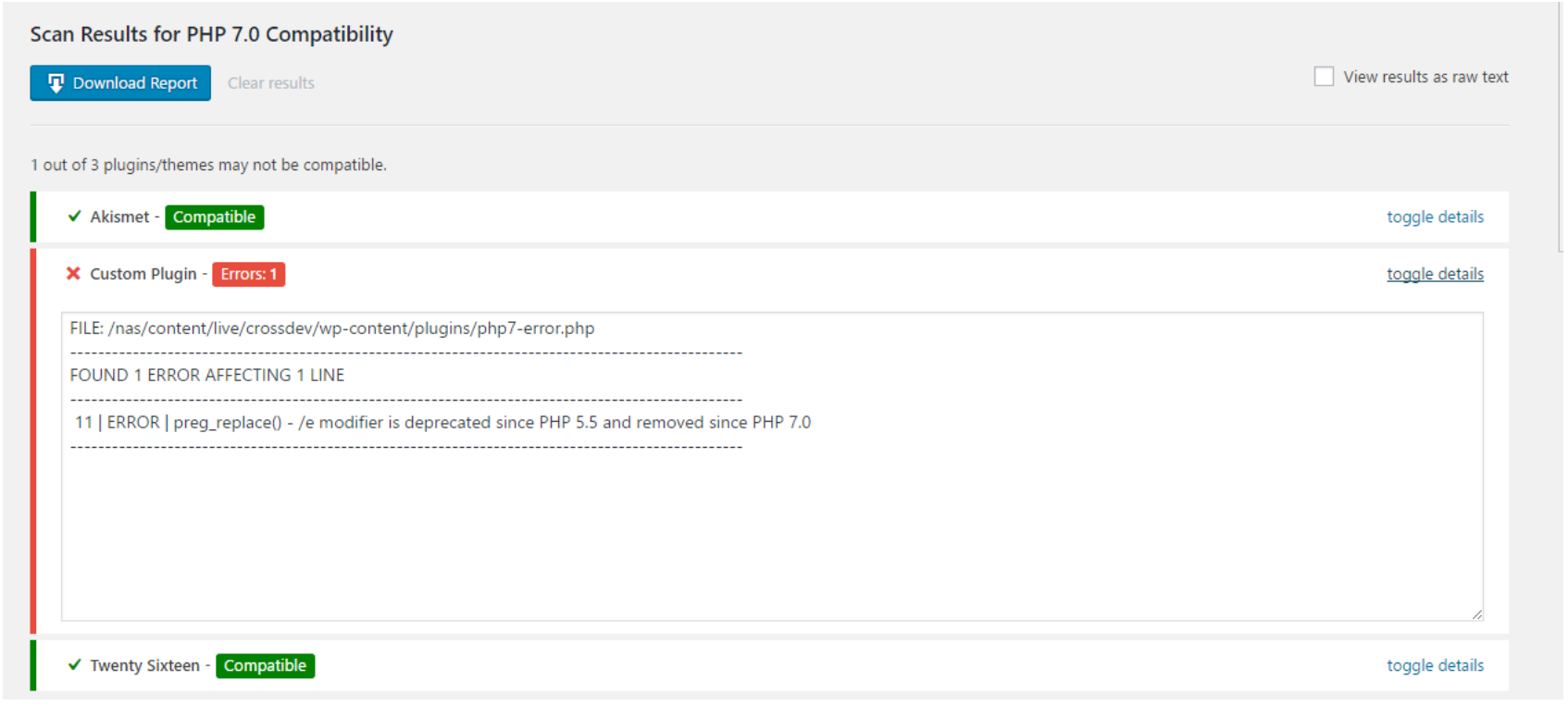
Let's see what it looks like

Checking for specific versions

- Default : latest PHP version
- Check for single version :
  phpcs --standard=PHPCompatibility --runtime-set testVersion 7.0 srcdir
- Check for multiple specific versions :
  phpcs --standard=PHPCompatibility --runtime-set testVersion 7.0-7.1 srcdir
- Check for minimum version :
  phpcs --standard=PHPCompatibility --runtime-set testVersion 7.0- srcdir
- Checking for older version :
  phpcs --standard=PHPCompatibility --runtime-set testVersion 5.0 srcdir

Other tools

- For Wordpress : PHP Compatibility Checker (uses PHPCompatibility)



Other tools

- For Wordpress : PHP Compatibility Checker (uses PHPCompatibility)
- PhpStorm 10+ : PHP 7 Compatibility Inspection
- sstalle/php7cc : similar functionality, slightly less up-to-date
- phan/phan : general static analyzer, compatibility checks are mostly useful for type checking
- adamculp/php-compatibility-check : docker image that uses PHPCompatibility, php7cc and phan

Conclusion

- No 100% detection
- But : 95% automation = lots of time saved !

- First : PHPCompatibility on local machine
- Then : use your CI environment

Start upgrading !

Questions ?

Questions ?

Big thanks to...

- Juliette Reinders Folmer
- Has been the main contributor (95%+ of all commits) in last 3 years
- PHP_CodeSniffer wizard
- Speaking at ConFoo

Thanks !