Dive into PHP7

PHPSW 11 May 2016

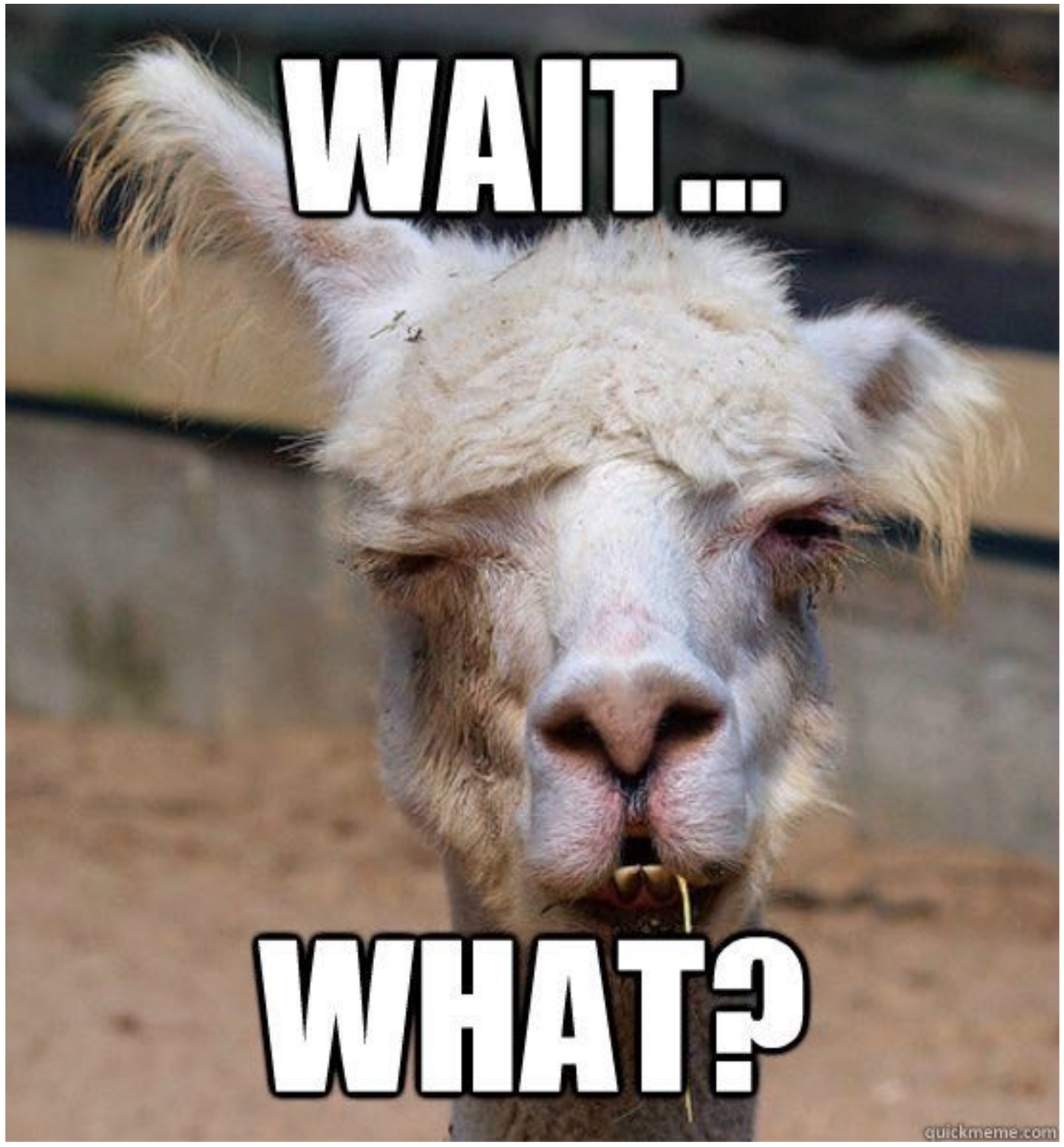
About me

- Kat
- Software developer, PHP and Go
- Currently at Brightpearl, previously at Buffer and Brightpearl

@kasiazien

PHP 7

- 5.0.0 released 13 July 2004 - 11 years ago (!)
- 7.0.0 released on 3 Dec 2015
- currently at 7.0.6 (released 28 Apr 2016)


WAIT...
WHAT?
quickmeme.com

PHP 6 no worky

TLDR; PHP 5 = PHP 6 - Unicode

- Launched in 2005 to bring native Unicode support to PHP
- Officially abandoned in 2010: shortage of developers with the necessary knowledge and choice of UTF-16 (as opposed to UTF-8) quoted as main reasons. Post mortem
- PHP 5.3 created in 2009 with many non-Unicode features back-ported from PHP 6 (e.g. namespaces)
- PHP 5.4 release prepared containing most remaining non-Unicode features from PHP 6 (e.g. traits) after PHP 6 declared dead
- *Huge* debate on what to call the version after that - see the official RFC for pros & cons & vote results (surprise surprise… 7 won)

php7

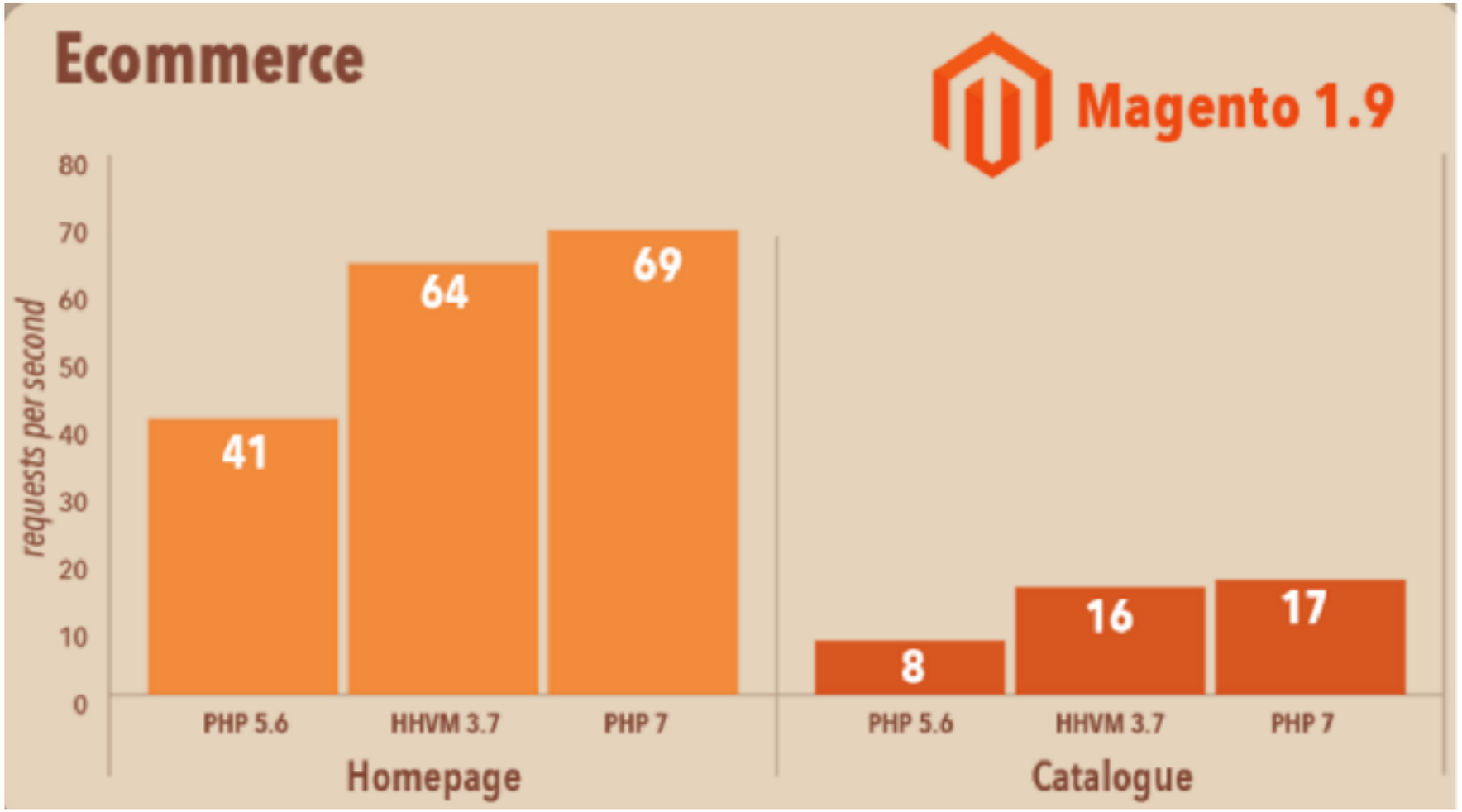
What’s new?

What’s going to break?

How do I get it?

SPEED!

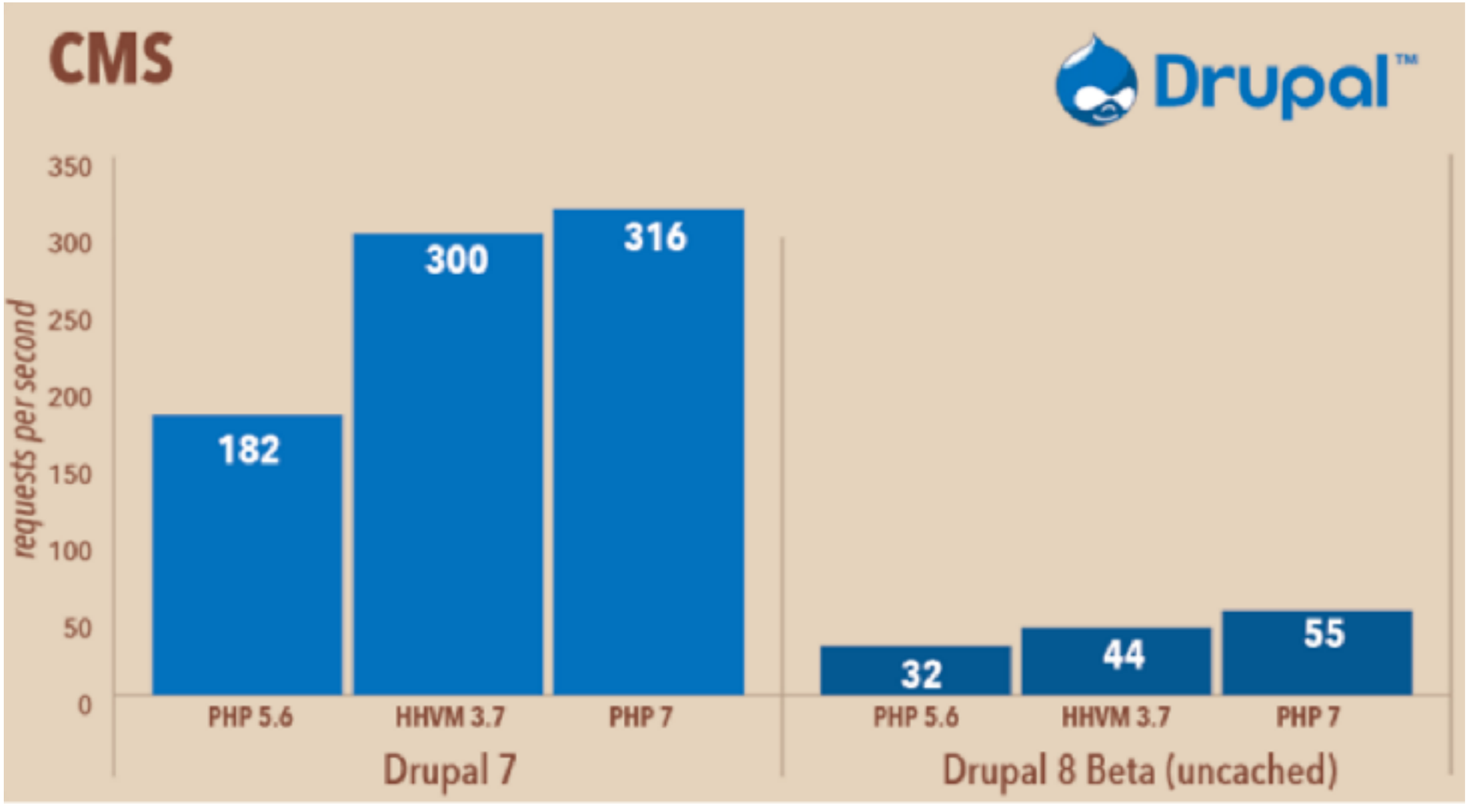
Magento on PHP 7 over 2x faster and uses 30% less memory compared to PHP 5.6.



source: http://www.zend.com/en/resources/php7_infographic

SPEED!

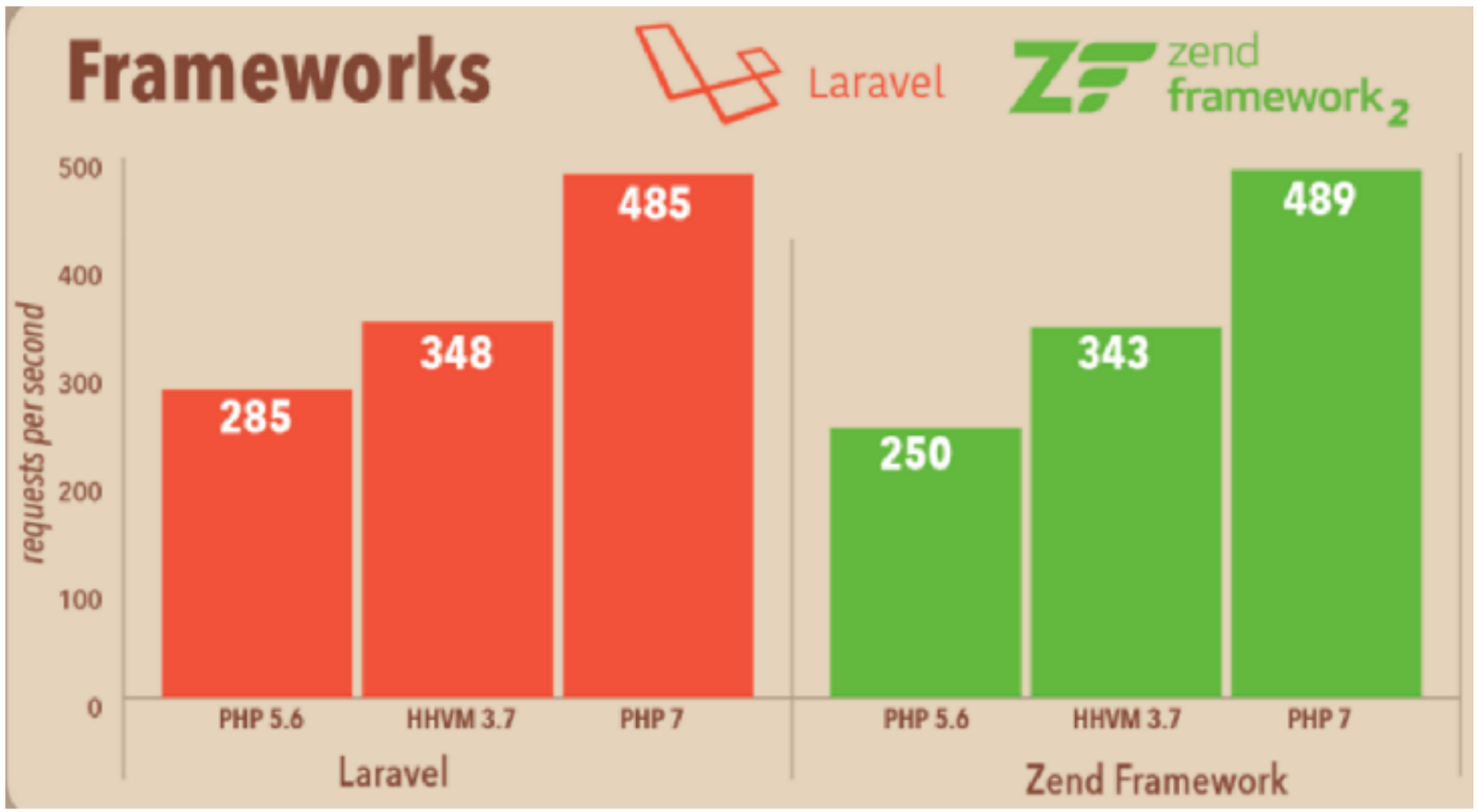
Drupal 8 72% faster on PHP 7 compared to PHP 5.6



source: http://www.zend.com/en/resources/php7_infographic

SPEED!

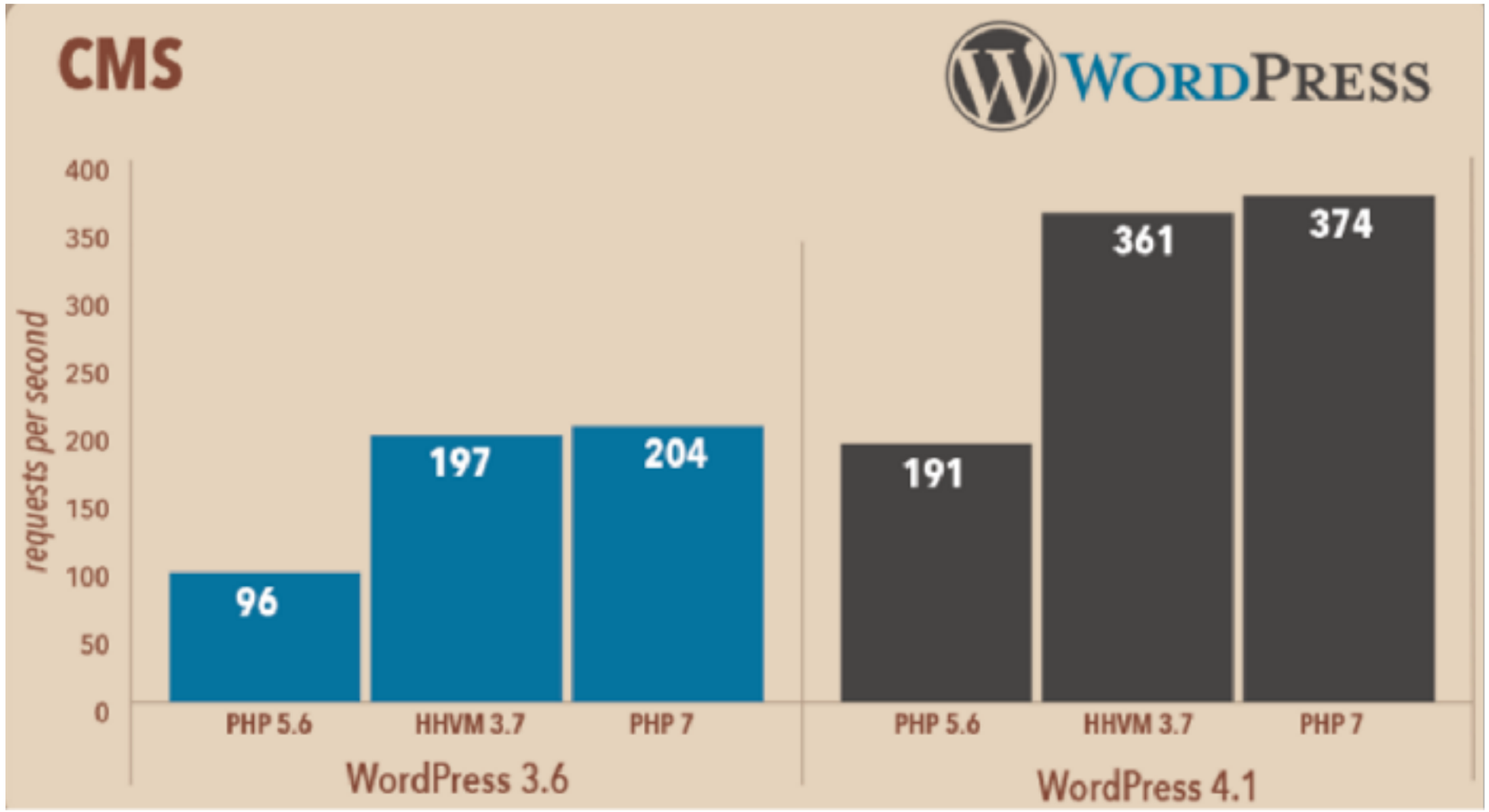
Laravel almost 60% faster and ZF2 over 95% faster on PHP 7 compared to PHP 5.6



source: http://www.zend.com/en/resources/php7_infographic

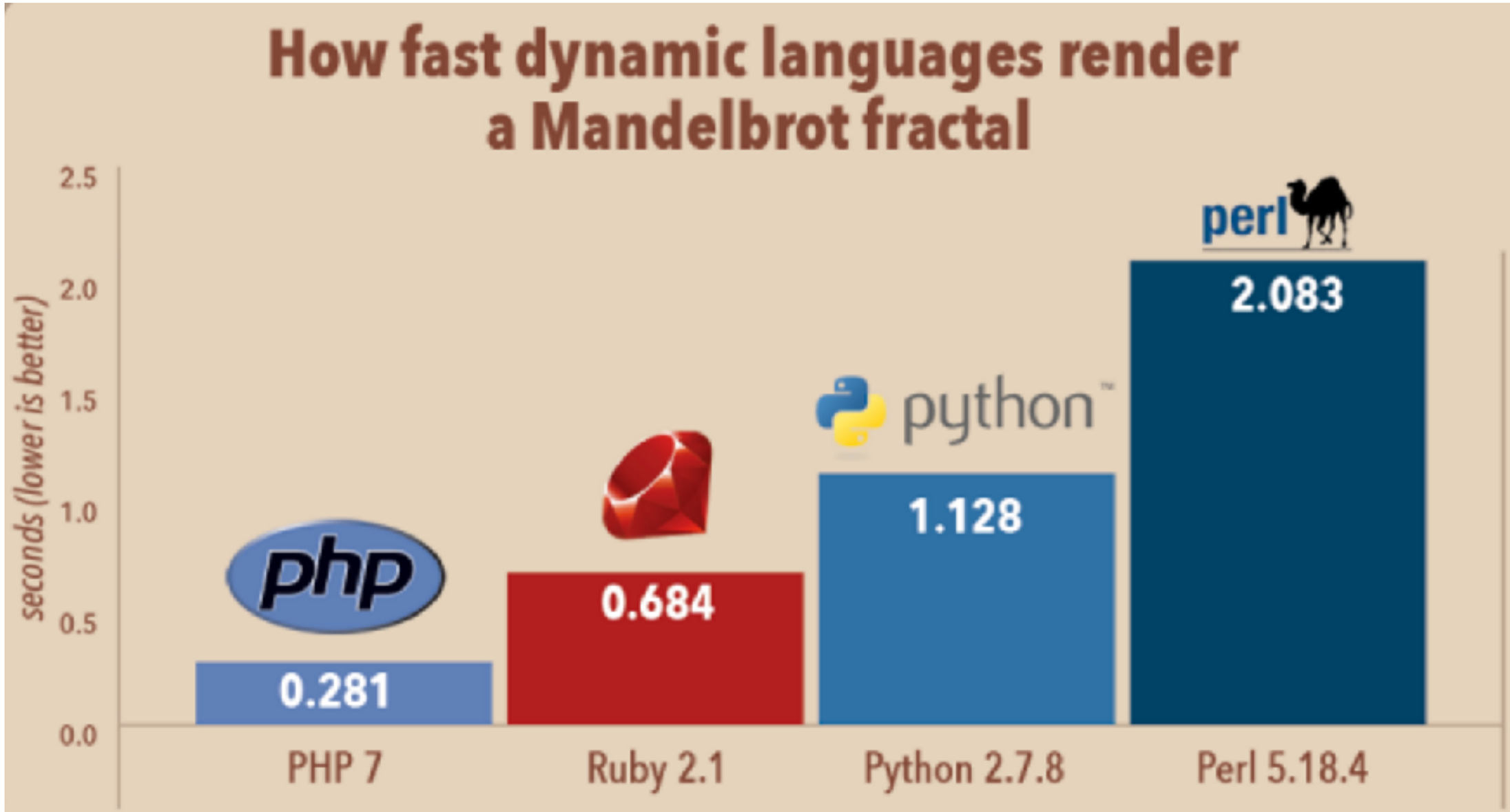
SPEED!

One WordPress request on PHP 5.6 executes just under 100 billion CPU instructions, PHP 7 does only 25 billion to do the same job! 196% - 213% faster!



source: http://www.zend.com/en/resources/php7_infographic

SPEED!



source: http://www.zend.com/en/resources/php7_infographic

SPEED!
Where does it come from?

- The foundation of PHP 7 is a PHP branch that was originally dubbed PHP next generation (**phpng**). It was aimed to optimize PHP performance by refactoring the Zend Engine while retaining near-complete language compatibility. Because of the significant changes, the reworked Zend Engine is called **Zend Engine 3**, succeeding Zend Engine 2 used in PHP 5.

PS. Intel's performance team collaborates with Zend on PHP optimization efforts (https://software.intel.com/en-us/blogs/2015/10/27/high-performance-php-7)

\Error

- PHP 7 introduced exceptions as a replacement for fatal or recoverable fatal errors. These exceptions do not extend `\Exception`, but instead extend a new class `\Error` to prevent the new exceptions from getting accidentally caught by legacy catch-all statements like `catch (\Exception $e) { }`

- Both `\Error` and `\Exception` implement a new `\Throwable` interface

- Just like before, an uncaught exception is still a regular fatal error

```
interface Throwable
    |- Exception implements Throwable
        |- ...
    |- Error implements Throwable
        |- TypeError extends Error
        |- ParseError extends Error
        |- ArithmeticError extends Error
            |- DivisionByZeroError extends ArithmeticError
        |- AssertionError extends Error
```

\Error

- `TypeError` is thrown when a function argument or return value does not match a type declaration
- `ParseError` is thrown when an included/required file or `eval()`'d code contains a syntax error
- `ArithmeticError` is thrown from negative bit shifts
- `AssertionError` will be thrown when the condition set by `assert()` is not met

- `Throwable`, `Error`, `TypeError`, and `ParseError` are now built-in interfaces/classes and so it's no longer possible to create classes with those names. It's possible for those names to be used within a non-global namespace though.

- catch-all blocks should now catch `Throwable`

- more on errors: https://trowski.com/2015/06/24/throwable-exceptions-and-errors-in-php7/

Scalar Type Hints

- You can now type-hint with scalar types: int, float, string, and bool
- By default type-hints are non-strict, which means they will cast the original type to the type specified by the type-hint

  e.g. if you pass int(1) into a function that requires a float, it will be come float(1.0). Passing float(1.5) into a function that requires an int, it will be come int(1)

- You can enable strict mode by placing `declare(strict_types=1);` at the top of any file (strict type-checking mode will be used for function calls and return statements in that file)
- If a type-hint mismatch occurs, a Catchable Fatal Error is thrown

Return Type Declarations

- You can add optional return type declaration to function declarations including closures, functions, generators, and methods
- Return type may be omitted unless a method inherits from a parent method that declares a return type
- Code which does not declare a return type will continue to work exactly as it does in PHP 5.x
- Class constructors, destructors and clone methods may not declare return types

```
function sayHello(string $name): string {
 return “Hello ” . $name;
}
```

Space Ships <=>

- aka the Combined Comparison Operator (RFC)
- `(expr) <=> (expr)`, returns 0 if both operands are equal, 1 if the left is greater, and -1 if the right is greater.
- can be used on strings, integers, floats, arrays, etc.

```
// Pre PHP 7
function order_func($a, $b) {
    return ($a < $b) ? -1 : (($a > $b) ? 1 : 0);
}

// Post PHP 7
function order_func($a, $b) { return $a <=> $b; }
```

Null Coalesce Operator

- `??` returns the result of its first operand if it exists and is not NULL, or else its second operand, e.g.
  `$username = $_GET['user'] ?? 'nobody';`
- e.g. `$_GET['mykey'] ?? ""` is completely safe and will not raise an E_NOTICE
- can be chained, e.g.
  ```
  $x = NULL; $y = NULL; $z = 3;
  var_dump($x ?? $y ?? $z); // int(3)
  ```

CSPRNG

- Cryptographically Secure Pseudo-Random Number Generator API (RFC)
- provides an easy and reliable way to generate crypto-strong random integers and bytes for use within cryptographic contexts (e.g. password salts)
- there are random number generators in PHP (e.g. `rand()`), but none of the options in version 5 are very secure. In PHP 7, CSPRNG uses the operating system's random number generator. If that gets hacked we have bigger problems.

Unicode Codepoint Escape Syntax

- new escape character: `\u{CODEPOINT}`
- allows to specify Unicode hex character code points unambiguously inside PHP strings, e.g. `\u{1F60D}` outputs 😍

Inconsistency Fixes

- needle/haystack issues have not been fixed (sad face)

however

- PHP 7 introduces an **Abstract Syntax Tree (AST)** (RFC) - an intermediate representation of the code during compilation (as opposed to emitting opcodes directly from the parser)

  Decoupling the parser and compiler allows us to remove a number of hacks and makes the implementation more maintainable and understandable in general. Furthermore it allows implementing syntax that was not feasible with a single-pass compilation process.

Inconsistency Fixes

- **Uniform Variable Syntax** (RFC) which solves numerous inconsistencies in how expressions are evaluated, e.g.:
  - support operations on arbitrary (`...`) expressions
  - ability to call closures assigned to properties using `($object->closure)()`
  - ability to chain static calls (nested `::`) and nested `()` e.g `$foo->bar()::baz()` or `Foo::bar()()`

Inconsistency Fixes

- indirect variable, property and method references are now interpreted with left-to-right semantics:

```
$$foo['bar']['baz']  interpreted as  ($$foo)['bar']['baz']

$foo->$bar['baz']()  interpreted as  ($foo->$bar)['baz']()

Foo::$bar['baz']()   interpreted as  (Foo::$bar)['baz']()
```

- this is backwards incompatible (with low practical impact) and it is always possible to recreate the old behaviour by explicitly using braces:

```
${$foo['bar']['baz']}

$foo->{$bar[‘baz']}()

Foo::{$bar['baz']}()
```

Bind Closure on Call

- `Closure->call()`, a new function in PHP 7 takes the object as it's first argument, followed by any arguments to pass into the closure:

```
class HelloWorld {
 private $greeting = “Hello ";
}
$closure = function($whom) {
 echo $this->greeting . $whom;
}
$obj = new HelloWorld();
$closure->call($obj, 'World'); // Hello World
```

Group Use Declarations

```
// Original
use Framework\Component\SubComponent\ClassA;
use Framework\Component\SubComponent\ClassB as
ClassC;
use Framework\Component\OtherComponent\ClassD;


// With Group Use
use Framework\Component\{
     SubComponent\ClassA,
     SubComponent\ClassB as ClassC,
     OtherComponent\ClassD
};
```

Generator Return Expressions

- allows you to now return a value upon (successful) completion of a generator
- if the generator has not yet returned, or has thrown an uncaught exception, calling $generator->getReturn() will throw an exception
- if the generator has completed but there was no return, null is returned

Generator Return Expressions

```
function foo() {
    yield 1;
    yield 2;
    return 42;
}

$bar = foo();
foreach ($bar as $element) {
    echo $element, "\n";
}

var_dump($bar->getReturn());

// 1
// 2
// int(42)
```

Generator Delegation

```
yield from <expr>
```

- allows you to return another iterable structure that can itself be traversed - an array, an iterator, or another generator

- iteration of sub-structures is done by the outer-most original loop as if it were a single flat structure rather than a recursive one

Generator Delegation

```
function g() {
  yield 1;
  yield from [2, 3, 4];
  yield 5;
}

$g = g();
foreach ($g as $yielded) {
    var_dump($yielded);
}
```

```
/*
int(1)
int(2)
int(3)
int(4)
int(5)
*/
```

What's going to break?

Deprecated: (http://php.net/manual/en/migration70.deprecated.php)

- PHP 4 style constructors (methods that have the same name as the class they are defined in) now deprecated
- Static calls to methods that are not declared static
- `password_hash()` salt option has been deprecated to prevent developers from generating their own (usually insecure) salts. The function itself generates a cryptographically secure salt when no salt is provided, so custom salt generation should not be needed
- The `capture_session_meta SSL` context option has been deprecated. SSL metadata is now available through the `stream_get_meta_data()` function

What's going to break?

Removed: (http://php.net/manual/en/migration70.incompatible.php, RFC)

- #allthedeprecatedthings (see the RFC for details):
  - ext/ereg (deprecated since PHP 5.3; use ext/pcre instead)
  - ext/mysql (deprecated since PHP 5.5; use ext/mysqli or ext/pdo_mysql instead)
  - a bunch of other extensions and SAPIs (http://php.net/manual/en/migration70.removed-exts-sapis.php)
  - assignment of new by reference (deprecated since PHP 5.3; use normal assignment instead)
  - scoped calls of non-static methods from incompatible `$this` context (deprecated since PHP 5.6) - static calls made to a non-static method with an incompatible context will now result in the called method having an undefined $this variable and a deprecation warning issued
  - deprecated functions: `dl` on fpm-fcgi, `set_magic_quotes_runtime` and `magic_quotes_runtime, set_socket_blocking` (use `stream_set_blocking` instead), `mcrypt_generic_end` (use `mcrypt_generic_deinit` instead), `mcrypt_ecb`, `mcrypt_cbc`, `mcrypt_cfb` and `mcrypt_ofb` (use `mcrypt_encrypt` and `mcrypt_decrypt` instead), `datefmt_set_timezone_id` and `IntlDateFormatter::setTimeZoneID` (use `datefmt_set_timezone` or `IntlDateFormatter::setTimeZone` instead)

What’s going to break?

Removed: (http://php.net/manual/en/migration70.incompatible.php, RFC)

- #allthedeprecatedthings (contd):
  - deprecated ini options: `xsl.security_prefs` (use `XsltProcessor::setSecurityPrefs` instead), `iconv.input_encoding`, `iconv.output_encoding`, `iconv.internal_encoding`, `mbstring.http_input`, `mbstring.http_output` and `mbstring.internal_encoding` (use `php.input_encoding`, `php.internal_encoding` and `php.output_encoding` instead)
  - `$is_dst` parameter of `mktime()` and `gmmktime()` removed
  - # style comments in ini files removed
  - String category names in `setlocale()` removed (use LC_* constants instead)
  - Unsafe curl file uploads (use `CurlFile` instead)
  - `preg_replace()` eval modifier removed (use `preg_replace_callback` instead)
  - `PDO::PGSQL_ATTR_DISABLE_NATIVE_PREPARED_STATEMENT` driver option removed (use `PDO::ATTR_EMULATE_PREPARES` instead)
  - `CN_match` and `SNI_server_name` stream context option removed (use `peer_name` instead)

What's going to break?

Removed: (http://php.net/manual/en/migration70.incompatible.php, RFC)

- Changes to error and exception handling (see the PHP migration manual for details)
- Changes to variable handling:
  - expressions now evaluated left-to-right (`$$foo->bar` —> (`$$foo)->bar`) unless you use curly braces to change that
  - variable variables can no longer be used with the global keyword unless you use curly braces (`global $$foo->bar;` —> `global ${$foo->bar};` or it won't work)
- Changes to `list()` handling, most important is probably that `list()` no longer assigns variables in reverse order when it's used in conjunction with `array[]` (`list($a[], $a[], $a[]) = [1, 2, 3];` will produce `$a[1, 2, 3]` not `$a[3, 2, 1]`)

What's going to break?

Removed: (http://php.net/manual/en/migration70.incompatible.php, RFC)

- Changes to foreach:
  - foreach no longer changes the internal array pointer (had the potential to cause some weird bugs if you were passing by ref)

```
$array = [0, 1, 2];
foreach ($array as &$val) {
    var_dump(current($array));
}
```

outputs `int(1), int(2), bool(false)` in PHP 5 and `int(0), int(0), int(0)` in PHP 7

  - foreach by-value operates on a copy of the array
  - foreach by-reference has improved iteration behaviour, e.g. appending to an array while iterating will now result in the appended values being iterated over as well

What’s going to break?

Removed: (http://php.net/manual/en/migration70.incompatible.php, RFC)

- Changes to integer handling
  - Bitwise shifts by negative numbers will now throw an `ArithmeticError`
  - Division by zero will throw a `DivisionByZeroError` exception (fatal)
- Changes to string handling ¶
  - Hexadecimal strings are no longer considered numeric
  - Strings containing a literal \u{ followed by an invalid sequence will cause a fatal error due to the addition of the new Unicode codepoint escape syntax (escape the leading backslash to handle this)

What's going to break?

Removed: (http://php.net/manual/en/migration70.incompatible.php, RFC)

- New objects cannot be assigned by reference (`$f =&newFoo()` will throw a parse error)
- ASP and script PHP tags removed (<% and %>, <script language=php> and </script>)
- Calls made to a non-static method with an incompatible context will now result in the called method having an undefined `$this` variable and a deprecation warning being issued ("`Non-static method A::test() should not be called statically`")
- `yield` is now a right associative operator, no longer needs parentheses (`echo yield -1;` was previously interpreted as `echo (yield) - 1;` and is now interpreted as `echo yield (-1);`)
- functions cannot have multiple parameters with the same name ¶
- `switch` statements cannot have multiple default blocks
- `$HTTP_RAW_POST_DATA` is no longer available (use `php://input` instead)
- new forbidden class names: `bool, int, float, string, NULL, TRUE, FALSE` and also `resource, object, mixed and numeric` are reserved for future use and should be considered deprecated

What’s going to break?

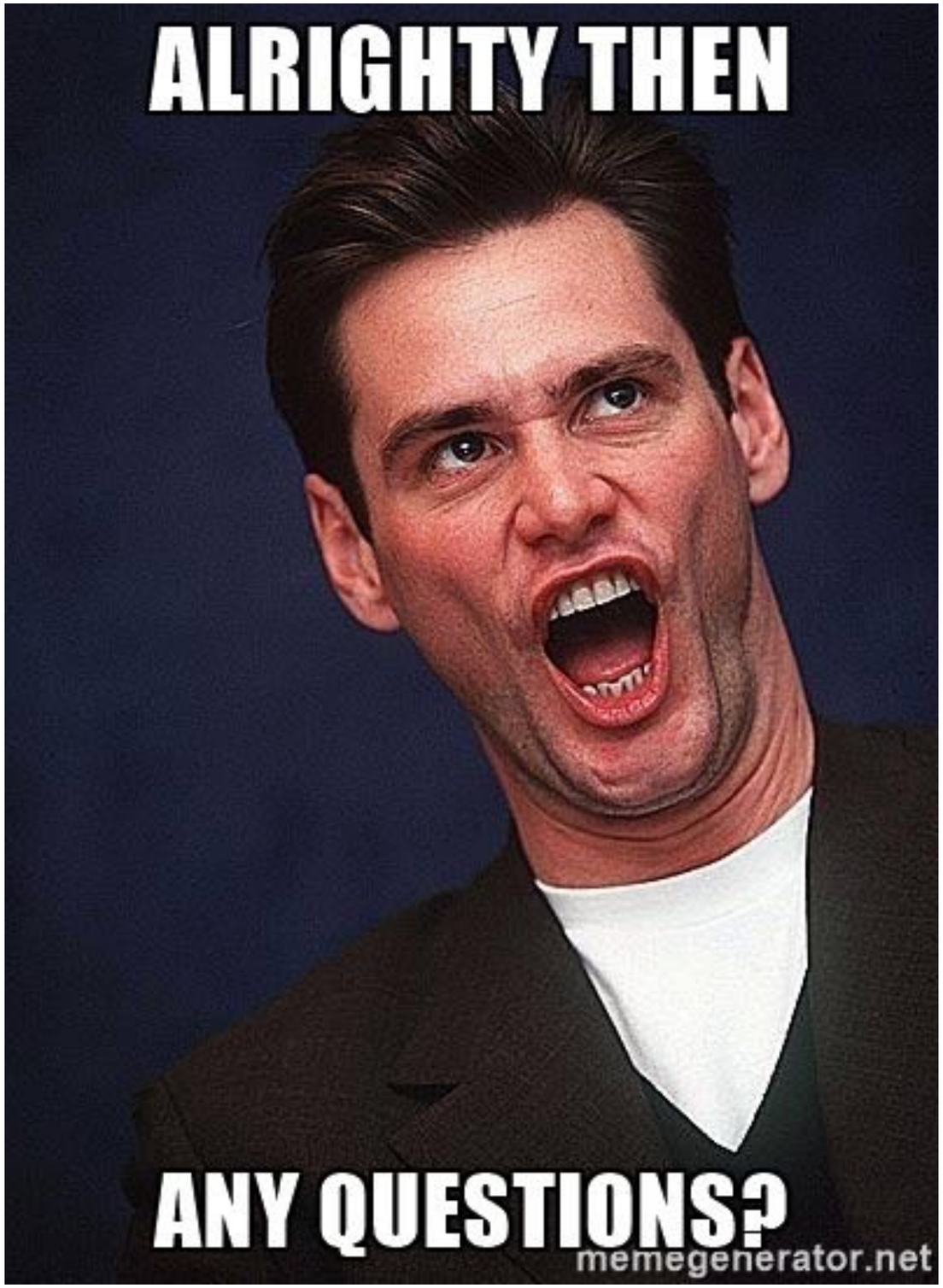
—> usePHPStorm’s built-in PHP 7 compatibility check (2016 edition)

> Code > Run inspection by name… > PHP 7 Compatibility

—> PHP 7 Compatibility Checker(php7cc) on github

Great, how do I get it?

- Grab Rasmus's PHP 7 vagrant box from http://github.com/rlerdorf/php7dev
- Download PHP 7 from http://php.net/downloads.php
- Google around, e.g. How To Upgrade to PHP 7 on Ubuntu 14.04 from Digital Ocean


ALRIGHTY THEN
ANY QUESTIONS?
memegenerator.net