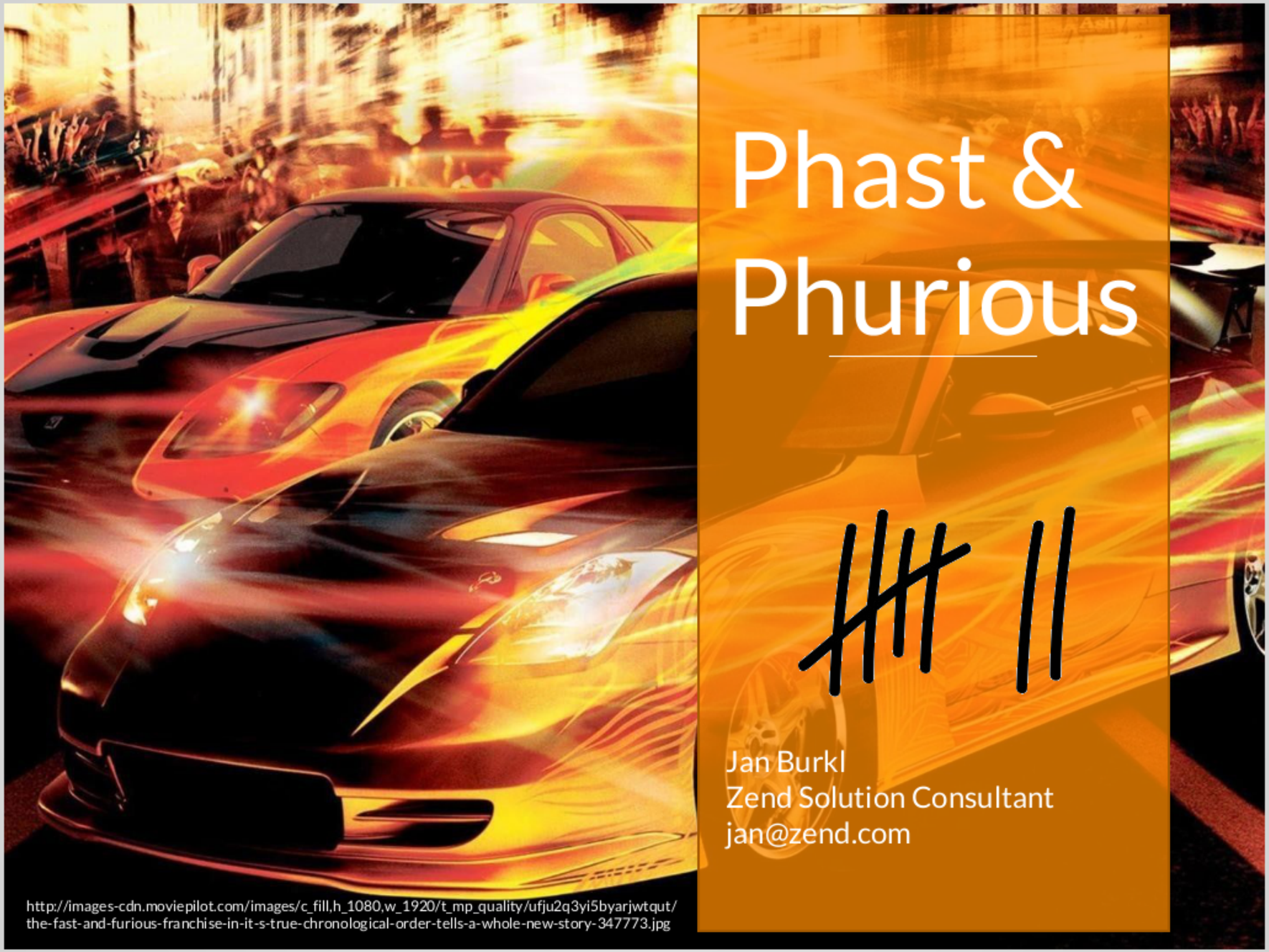


# Phast & Phurious

Jan Burkl
Zend Solution Consultant
jan@zend.com

http://images-cdn.moviepilot.com/images/c_fill,h_1080,w_1920/t_mp_quality/ufju2q3yi5byarjwtqut/the-fast-and-furious-franchise-in-it-s-true-chronological-order-tells-a-whole-new-story-347773.jpg

# EVOLUTION


# PHP 3.0

- Released: June 1998
- New Features:
  - Full fledged language
  - Extensions
- Performance Features:
  - Token Cache
  - Memory Manager


# PHP 4.0

- Released: May 2000

# PHP 4.0

- Released: May 2000
- New Features:
  - Modularity
  - Sessions
  - Downwards compatibility (!)
- Performance Features:
  - Zend Engine


# PHP 5

- 5.0: July 2004

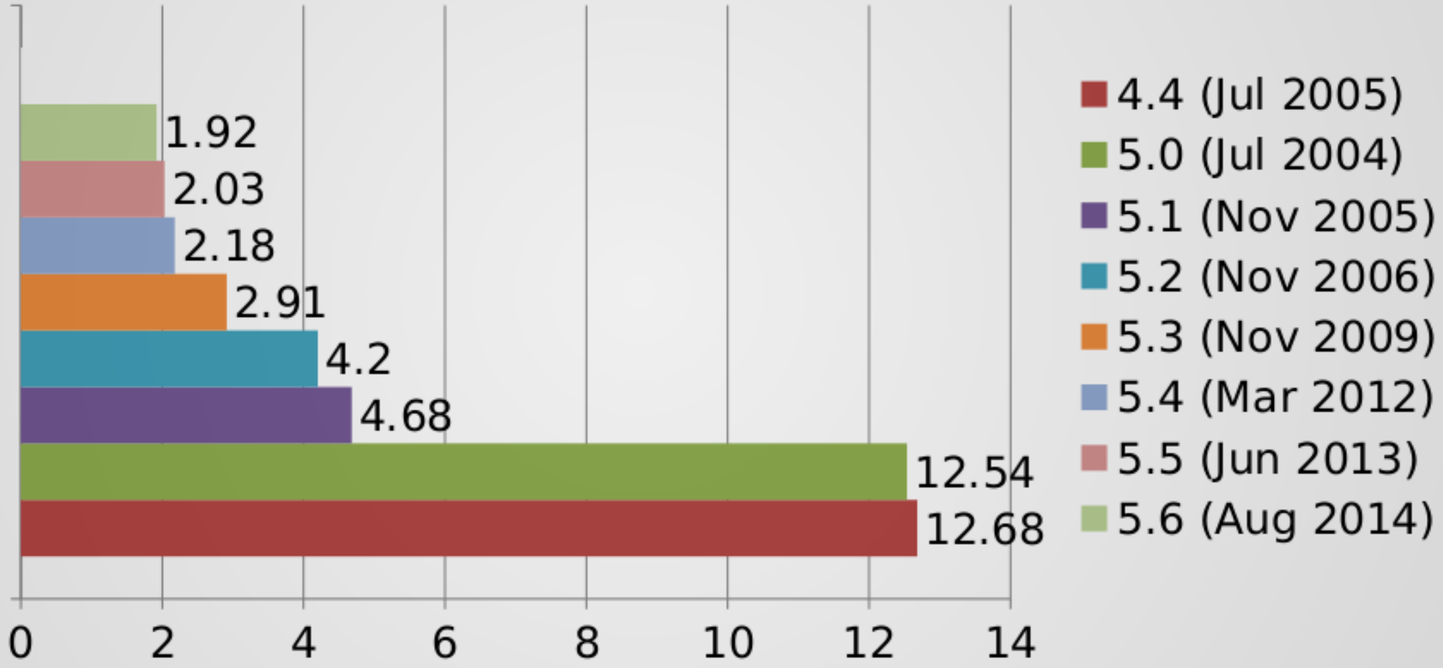
# PHP 5

- 5.0: July 2004
  - New Features:
    - New “true“ object model
    - Destructors
    - Exception handling
- Performance features
  - Be thankful it‘s not slower!
  - Seeds were sown...

# PHP 6

# PHP PERFORMANCE EVOLUTION

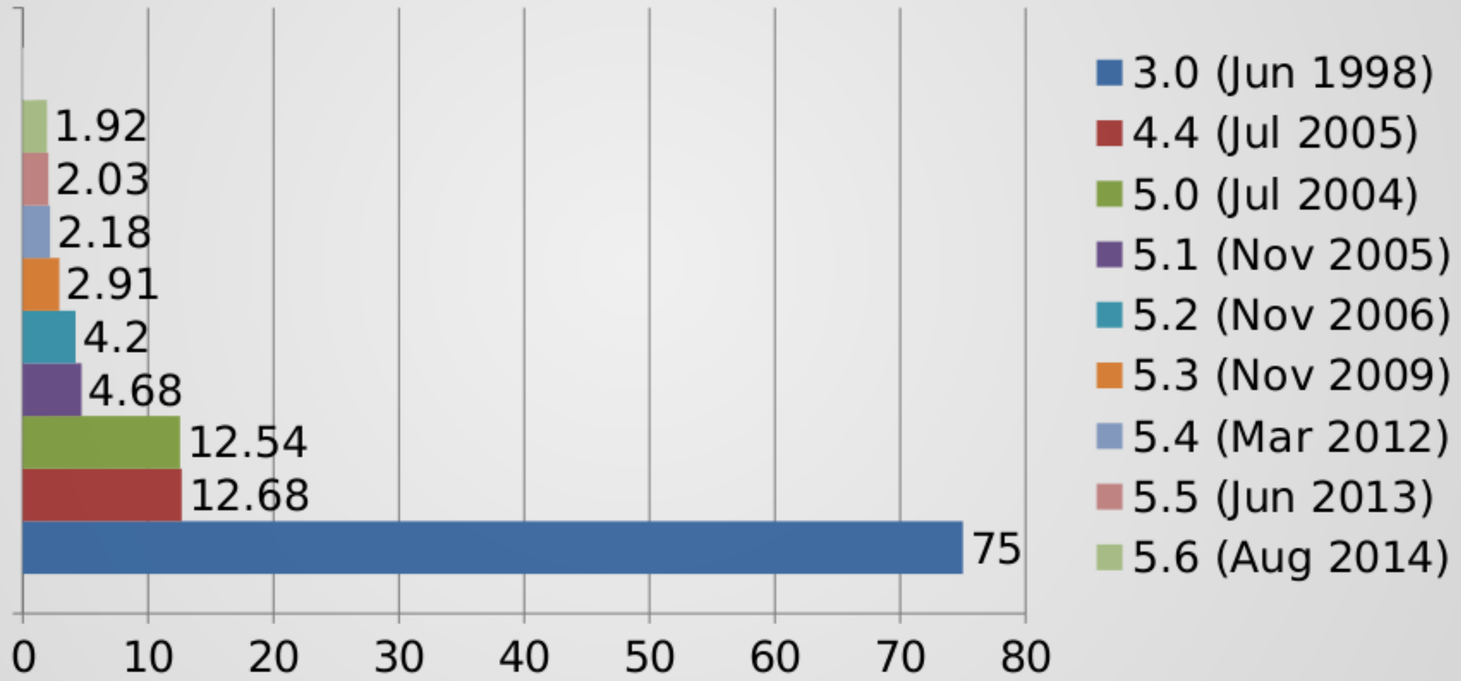
## BENCH.PHP (LOWER IS BETTER)




zend

PHP 2015 .RUHR

# PUTTING THINGS IN PERSPECTIVE




zend

PHP 2015 .RUHR

# PHP 6

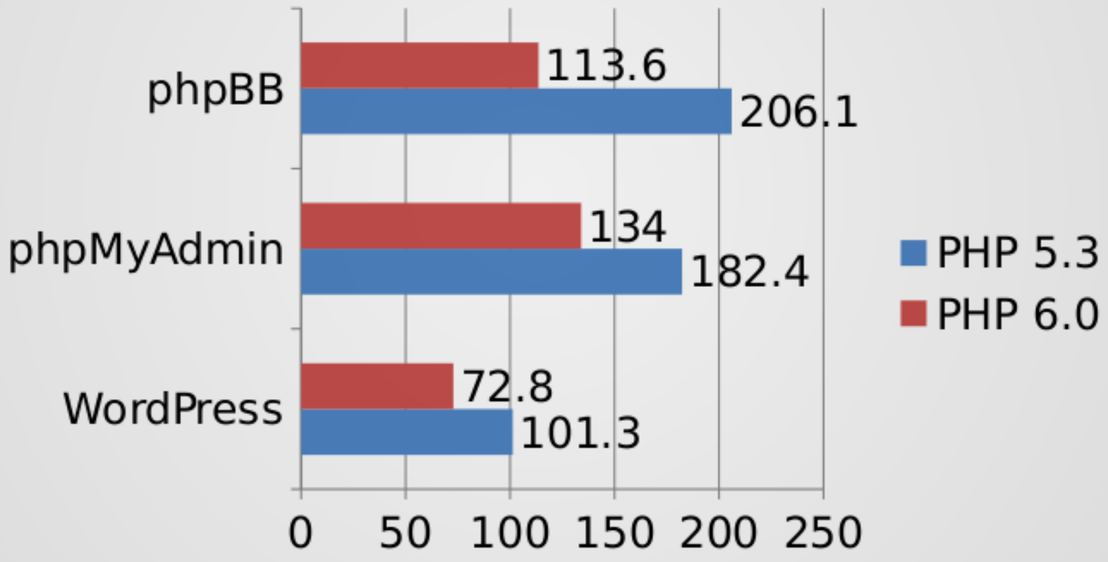
## PHP 6 = PHP 5 + UNICODE


## WHAT ENDED UP HAPPENING


BONUS: ROUGHLY DOUBLED MEMORY FOOTPRINT

# WHAT ENDED UP HAPPENING

## BONUS: ROUGHLY DOUBLED MEMORY FOOTPRINT.




zend

PHP 2015 .RUHR

THE END

# THE END

# TIME OF DEATH

## MARCH 11, 2010

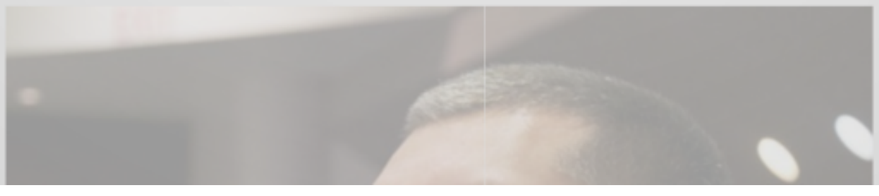
http://bit.ly/php6whathappened


# PHP 7

## LEADING UP TO PHP 7

2012: Research PHP+JIT begins, led by Dmitry Stogov

2014: No performance gains realized for real-world workloads



# LEADING UP TO PHP 7

2012: Research PHP+JIT begins, led by Dmitry Stogov

2014: No performance gains realized for real-world workloads

zend

PHP 2015 .RUHR

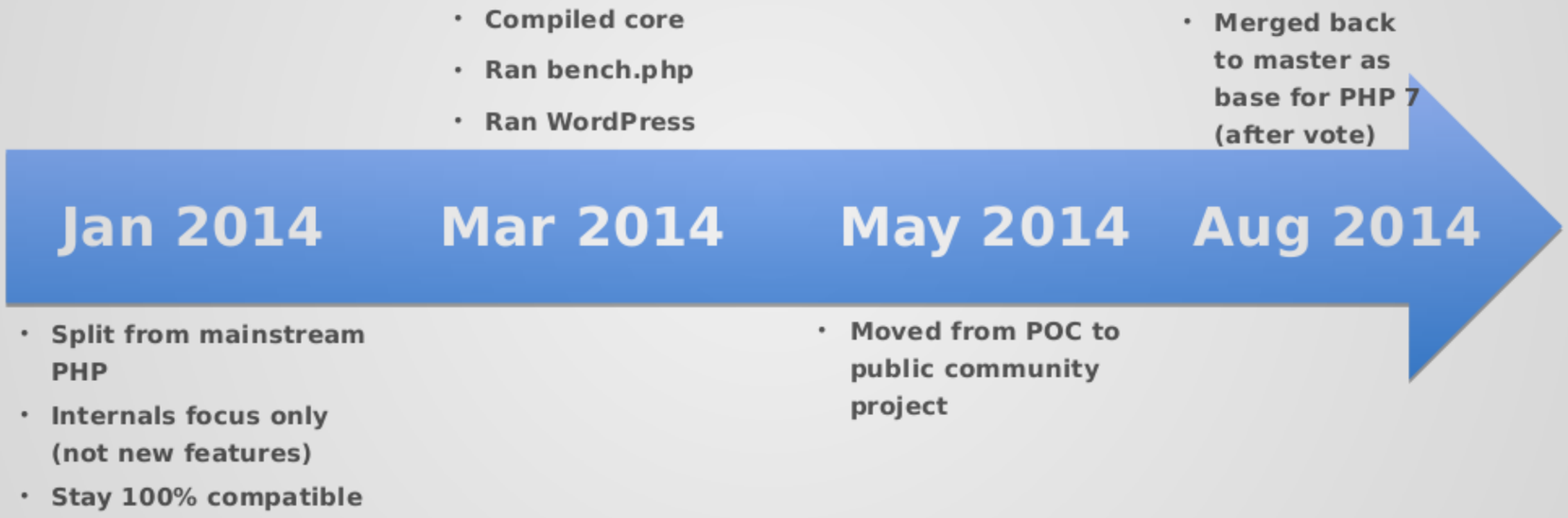
2014: No performance gains realized for real-world workloads

# BIRTH OF PHPNG

zend

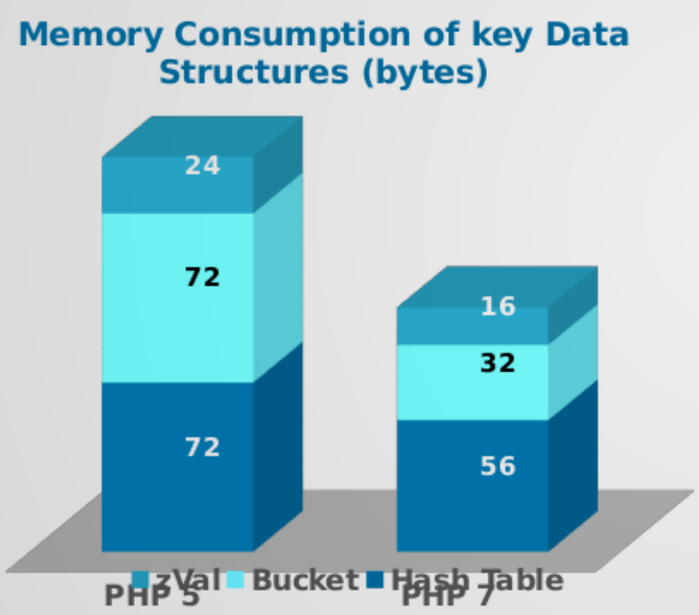
PHP 2015 .RUHR

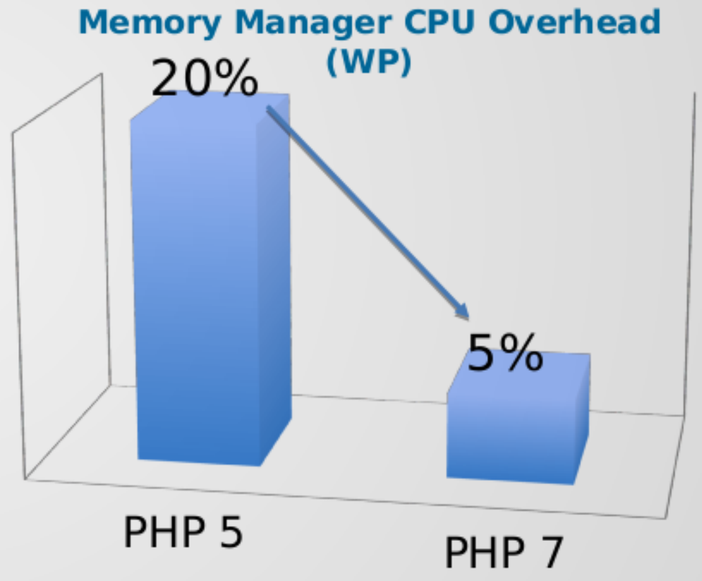
# BIRTH OF PHPNG





# WHAT MADE THE DIFFERENCE?



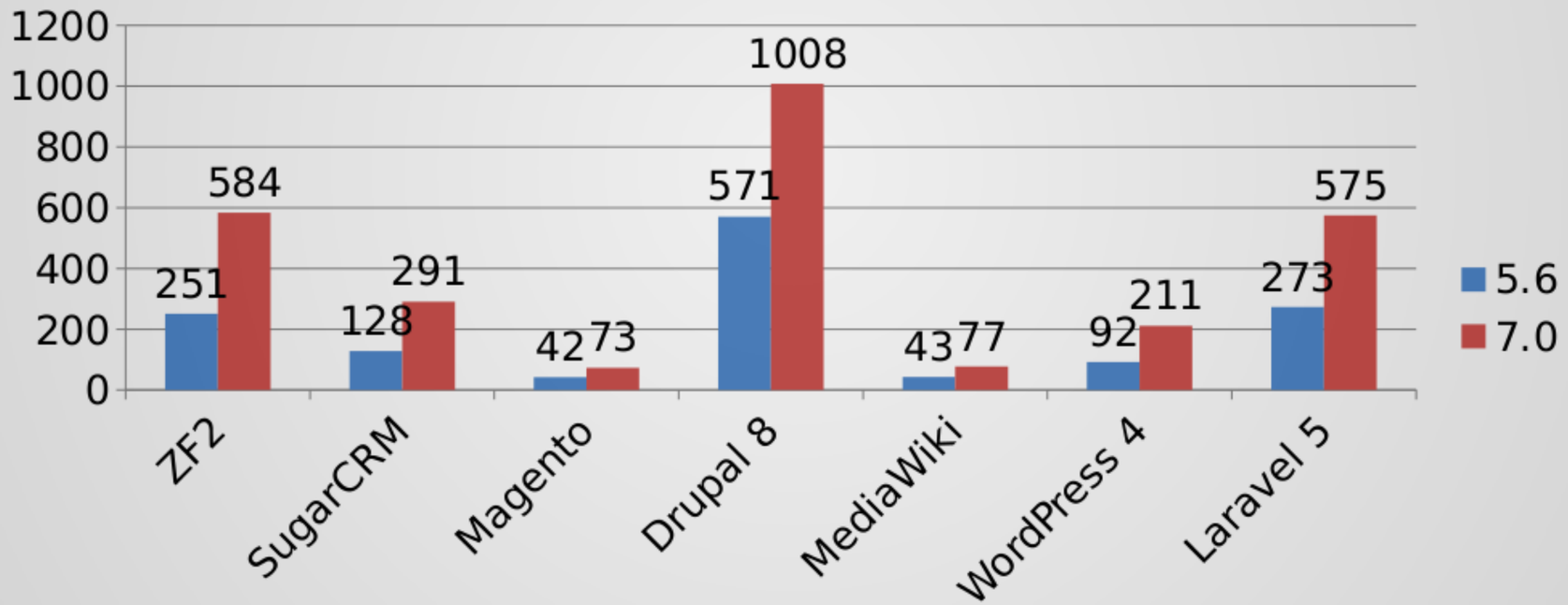




zend

PHP 2015 .RUHR

# PHP 7
# WHAT CAN WE EXPECT?


# 5.6 VS. 7.0

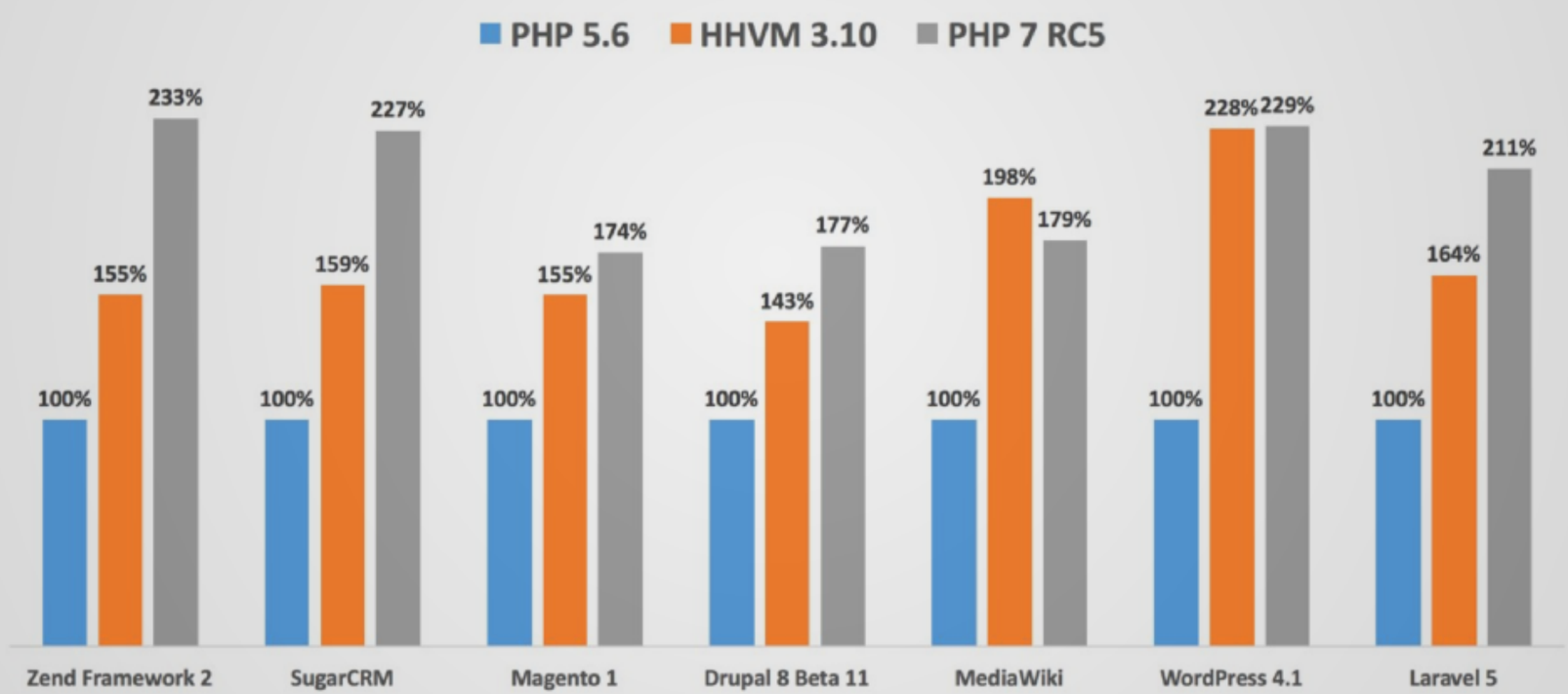



zend

PHP 2015 .RUHR

# REAL WORLD APPLICATION BENCHMARKS

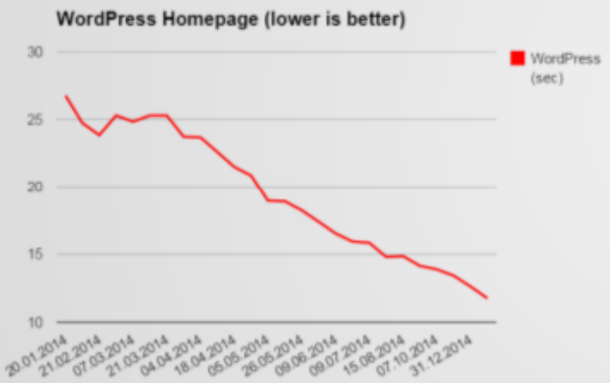
(% improvement vs. PHP 5.6)

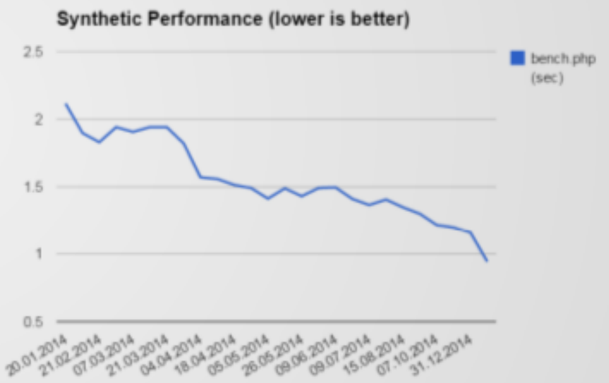



zend

PHP 2015 .RUHR

# IT'S GOTTEN CONSISTENTLY FASTER WITH MORE TO COME!




zend

PHP 2015 .RUHR

# MORE PERFORMANCE INFO

- http://bit.ly/php7perf
- http://bit.ly/php7fast
- http://bit.ly/php7mandel


# What's new?

http://images4.fanpop.com/image/photos/15000000/Seven-seven-of-nine-15093083-2560-1600.jpg

# UNIFORM VARIABLE SYNTAX

```
$foo->bar()();
```

- $foo is an object
- bar() is a method that returns a callable
- The callable is automatically executed

# UNIFORM VARIABLE SYNTAX

```
$foo->bar()();
```

- `$foo` is an object
- `bar()` is a method that returns a callable
- The callable is automatically executed

```
$foo()['bar']();
```

- `$foo` is a method that returns an array.
- `['bar']` is an element in the array returned by `$foo()`

```
$foo()['bar']();
```

- `$foo` is a method that returns an array.
- `['bar']` is an element in the array returned by `$foo()`
- `['bar']` contains a callable as it's value
- `()` calls the callable contained in `['bar']`

```
function getStr()
{
```

```php
function getStr()
{
    return 'Jan Burkl';
}
echo getStr(){4}; // 'B'
```

Double $ in referencing globals are no longer supported.

Double $ in referencing globals are no longer supported.

```
global $$foo->bar;
//No longer supported
```

There are no longer any restrictions on nesting of dereferencing operations. All of these are now supported:

There are no longer any restrictions on nesting of dereferencing operations. All of these are now supported:

```php
$obj1 = new StdClass();
$obj1->name = 'Barbara';
$obj2 = new StdClass();
$obj2->name = 'Jan';
$returnVal = [$obj1, $obj2][0]->name;
```

- Reference a property statically from a class reference

```php
$foo['bar']::$baz
```

- Reference a property statically from a class reference

```
$foo['bar']::$baz
```

- Statically access a nested referenced property

```
$foo::$bar::$baz
```

- Call a referenced method statically from a class reference returned by a method.

```
$foo->bar()::baz()
```

| | Old Meaning | New Meaning |
|---|---|---|
| `$$foo['bar']['baz']` | `${$foo['bar']['baz']}` | `($$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']()` |



zend

PHP 2015 .RUHR

# SPACESHIP
## COMBINED COMPARISON

- PHP's first trinary operator

```
echo 1<=>2; // -1
echo 1<=>1; // 0
echo 2<=>1; // 1
```

zend

PHP 2015 .RUHR

# SPACESHIP
## COMBINED COMPARISON

- PHP's first trinary operator

```
echo 1<=>2; // -1
echo 1<=>1; // 0
echo 2<=>1; // 1
```

# ARRAYS

```
echo [] <=> [];                    //  0
```

# ARRAYS

```
echo [] <=> [];                   //  0
echo [1, 2, 3] <=> [1, 2, 3]; //  0
echo [1, 2, 3] <=> [];            //  1
echo [1, 2, 3] <=> [1, 2, 1]; //  1
echo [1, 2, 3] <=> [1, 2, 4]; // -1
```

# OBJECTS

# OBJECTS

```
$a = (object) ["a" => "b"];
$b = (object) ["a" => "c"];
echo $a <=> $b; // -1

$a = (object) ["a" => "b"];
$b = (object) ["a" => "b"];
echo $a <=> $b; // 0
```

# USORT()

```
$array = ['oranges', 'apples',
          'bananas', 'grapes'];
```

# USORT()

```php
$array = ['oranges', 'apples',
          'bananas', 'grapes'];
usort($array,
  function ($left,$right) {
    return $left<=>$right;
  }
);
```

```
Array (
    [0] => apples
    [1] => bananas
    [2] => grapes
    [3] => oranges
)
```

# ANONYMOUS CLASSES

```
$obj = new class($i) {
    public function __construct($i) {
        $this->i = $i;
    }
};
```

ANONYMOUS

```php
$obj = new class($i) {
    public function __construct($i) {
        $this->i = $i;
    };
}
```

```php
(new class extends ConsoleProgram {
    public function main() {
```

```php
(new class extends ConsoleProgram {
    public function main() {
        /* ... */
    }
})->bootstrap();
```

```php
return new class($controller)
    implements Page {
```

```php
return new class($controller)
    implements Page {

    public function
        __construct($controller) {
        /* ... */
    }
    /* ... */
};
```


```php
$pusher->setLogger(new class {
    public function log($msg) {
```

```php
$pusher->setLogger(new class {
    public function log($msg) {
        print_r($msg . "\n");
    }
});
```

# CAVEATS, WARNING, AND COMPATIBILITY

# CAVEATS, WARNING, AND COMPATIBILITY

- Inheritance works as expected
- Traits work as expected
- Reflection now has ReflectionClass::isAnonymous() method
- Serialization will not work, just like with anonymous functions


# GENERATOR RETURN EXPRESSIONS

```
function foo() {
    $returnValue = 1;
    yield 1;
    $returnValue = 2;
    yield 2;
    return $returnValue;
}

$bar = foo();
```

# GENERATOR RETURN

```php
function foo() {
    $returnValue = 1;
    yield 1;
    $returnValue = 2;
    yield 2;
    return $returnValue;
}

$bar = foo();
```

```php
foreach ($bar as $element) {
    echo $element, "\n";
}
```

```
    return $returnValue;
}

$bar = foo();
```

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

var_dump($bar->getReturn());

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

# GENERATOR DELEGATION

```
function foo() {
    yield 1;
    yield 2;
}
function both() {
    yield from foo();
    yield from [2, 3, 4];
    yield "done"
}
```

# GENERATOR

```
function foo() {
    yield 1;
    yield 2;
}
function both() {
    yield from foo();
    yield from [2, 3, 4];
    yield "done"
}
```

```
$a = both();

foreach ($a as $value) {
```

```
    yield from [2, 3, 4];
    yield "done"
}
```

```
$a = both();

foreach ($a as $value) {
    echo $value . "\n";
}
```

# FILTERED UNSERIALIZE()

```php
// this will unserialize everything
// as before
$data = unserialize($foo);
```

```php
// this will convert all objects into
// __PHP_Incomplete_Class object
$data = unserialize(
    $foo,
    ["allowed_classes" => false]
);
```

```php
// this will unserialize everything
// as before
$data = unserialize($foo);
```

```php
// this will convert all objects into
// __PHP_Incomplete_Class object
$data = unserialize(
    $foo,
    ["allowed_classes" => false]
);
```

```
// this will unserialize everything
// as before
$data = unserialize($foo);
```

```
// this will convert all objects into
// __PHP_Incomplete_Class object
$data = unserialize(
    $foo,
    ["allowed_classes" => false]
);
```

```
// converts all objects except MyClass
// into __PHP_Incomplete_Class object
$data = unserialize(
    $foo,
    ["allowed_classes" => [
        "MyClass"
```


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

```
// converts all objects except MyClass
// into __PHP_Incomplete_Class object
$data = unserialize(
    $foo,
    ["allowed_classes" => [
        "MyClass"
    ]);
```

```
//accept all classes as in default
$data = unserialize(
    $foo,
    ["allowed_classes" => true]
);
```


```
    ["allowed_classes" => [
        "MyClass"
    ]);
```

```
//accept all classes as in default
$data = unserialize(
    $foo,
    ["allowed_classes" => true]
);
```


# THROWABLE INTERFACE

## EXCEPTIONS IN THE ENGINE

```
try {
    do_something(null); // oops!
} catch (\Error $e) {
    echo "Err: {$e->getMessage()}\n";
}

// Error: Call to a member function
// method() on a non-object
```

# THROWABLE INTERFACE

```
try {
    do_something(null); // oops!
} catch (\Error $e) {
    echo "Err: {$e->getMessage()}\n";
}

// Error: Call to a member function
// method() on a non-object
```


- interface Throwable
  - Exception implements Throwable

- interface Throwable
  - Exception implements Throwable
  - Error implements Throwable
    - TypeError extends Error
    - ParseError extends Error

```
function add(int $left, int $right) {
    return $left + $right;
```

```php
function add(int $left, int $right) {
    return $left + $right;
}
```

```php
try {
    echo add('left', 'right');
} catch (\TypeError $e) {
    // Log error and end gracefully
} catch (\Exception $e) {
    // Handle any exceptions
} catch (\Throwable $e) {
    // Handle everything else
}
```


```
}
```

```
try {
    echo add('left', 'right');
} catch (\TypeError $e) {
    // Log error and end gracefully
} catch (\Exception $e) {
    // Handle any exceptions
} catch (\Throwable $e) {
    // Handle everything else
}
```

# PARSEERROR

- Thrown when you have a parse error in your code

# PARSEERROR

- Thrown when you have a parse error in your code
- Will not allow bad code to run at compile time.
- Will be thrown if you have a parse error in an eval()
- Will be thrown if you have a parse error in a file that is included during execution.

```
$code = 'var_dup($admin);';

try {
    $result = eval($code);
} catch (\ParseError $error) {
    // Handle $error
}
```

```
if ($admin) {
    try {
```

```
}
```

```
if ($admin) {
    try {
        include "./issue_code.php";
    } catch (\ParseError $error) {
        // Handle $error
    }
}
```

# BENEFITS

- finally gets called
- __destruct() gets called.

# BENEFITS

- finally gets called
- __destruct() gets called.
- Fully backwardly compatible

# NULL COALESCE OPERATOR

```php
$name = $firstName ?? "Jan";
 $name  .=  " ";
  $name  .= $lastName ?? "Burkl";
```

## NULL COALESCE

```php
$name = $firstName ?? "Jan";
$name .=  " ";
$name .= $lastName ?? "Burkl";
```

```php
$this->maxCount = is_null(
    $input->getOption('count'))
```

```php
$this->maxCount = is_null(
    $input->getOption('count'))
    ? -1 : $input->getOption('count');
```

```php
$this->maxCount =
    $input->getOption('count') ?? -1;
```

```php
$x = NULL;
```

```php
$x = NULL;
$y = NULL;
$z = 3;

var_dump($x ?? $y ?? $z); // int(3)
```

# SCALAR TYPE HINTS

- int
- float
- string
- bool

# SCALAR TYPE HINTS

- int
- float
- string
- bool

```
function add(float $a, float $b) {
    return $a + $b;
```

```php
function add(float $a, float $b) {
    return $a + $b;
}

$returnValue = add(1.5, 2.5);
// int(4)
// Works
```

```php
$returnValue = add("1 foo", "2");
// PHP 5.6 and below gives a Notice
// PHP7 TypeError

$returnValue = add(1, 2); // int(3)
// Widening
```


```php
$returnValue = add("1 foo", "2");
// PHP 5.6 and below gives a Notice
// PHP7 TypeError

$returnValue = add(1, 2); // int(3)
// Widening
```

# WIDENING

- The only type casting done in strict mode
- Integers can be “widened” into floats.

# WIDENING

- The only type casting done in strict mode
- Integers can be “widened” into floats.

```
declare(strict_types=1);

function add(float $a, float $b) {
    return $a + $b;
}
var_dump(add(1, 2)); // float(3)
```


# RETURN TYPE DECLARATIONS

```
function foo(): array {
    return [];
}
```

# RETURN TYPE DECLARATIONS

```
function foo(): array {
    return [];
}
```

NOT ALLOWED

# NOT ALLOWED

- You cannot change the return type of a subclassed method.

```
class MyClass{
    public function foo(): array {
        return [];
    }
}
class MyOtherClass extends MyClass {
    public function foo(): MyClass {
        return new MyClass();
    }
}
```

class MyClass {
    function make(): MyClass {

NOT ALLOWED

- You cannot change the return type of a subclassed method.

```
class MyClass{
    public function foo(): array {
        return [];
    }
}
class MyOtherClass extends MyClass {
    public function foo(): MyClass {
        return new MyClass();
    }
}
```

ALLOWED

```
class MyClass {
    function make(): MyClass {
        return new MyClass();
    }
}
class MyOtherClass extends MyClass {
    function make(): MyOtherClass {
        return new MyOtherClass();
    }
}
```


# GROUP USE DECLARATIONS

BEFORE

```
namespace MyProj\Command;

use MyProj\Traits\WritelineTrait;
use MyProj\Twitter;

use Symfony\Component\Console\Command\C
use Symfony\Component\Console\Input\Inp
use Symfony\Component\Console\Output\Ou
```

GROUP USE DECLARATIONS

# BEFORE

```
namespace MyProj\Command;

use MyProj\Traits\WritelineTrait;
use MyProj\Twitter;

use Symfony\Component\Console\Command\C
use Symfony\Component\Console\Input\Inp
use Symfony\Component\Console\Output\Ou
```

namespace MyProj\Command;

# GROUP USE DECLARATIONS

BEFORE

```
namespace MyProj\Command;

use MyProj\Traits\WritelineTrait;
use MyProj\Twitter;

use Symfony\Component\Console\Command\C
use Symfony\Component\Console\Input\Inp
use Symfony\Component\Console\Output\Ou
```

AFTER

```
namespace MyProj\Command;
use MyProj\ {
    Traits\WritelineTrait,
    Twitter
};
use Symfony\Component\Console\ {
    Command\Command,
    Input\InputOption,
    Output\OutputInterface
};
```


# NEW FEATURES THAT DO NOT BREAK THINGS

- Integer Semantics
- Unicode Codepoint Escape Syntax
- Array Constants
- Expectations

# NEW FEATURES THAT DO NOT BREAK THINGS

- Integer Semantics
- Unicode Codepoint Escape Syntax
- Array Constants
- Expectations


# The Big 3

## PHP NG // Engine Exceptions // Scalar Type Hints

http://img15.deviantart.net/d7ab/i/2011/126/c/e/avengers_big_3_by_bigbmh-d3fomu1.png

zend

PHP 2015 .RUHR

# DANKE SCHÖN!

JAN@ZEND.COM


zend

PHP 2015 .RUHR