

# PHP7.1 New Features & Performance

@Larurence



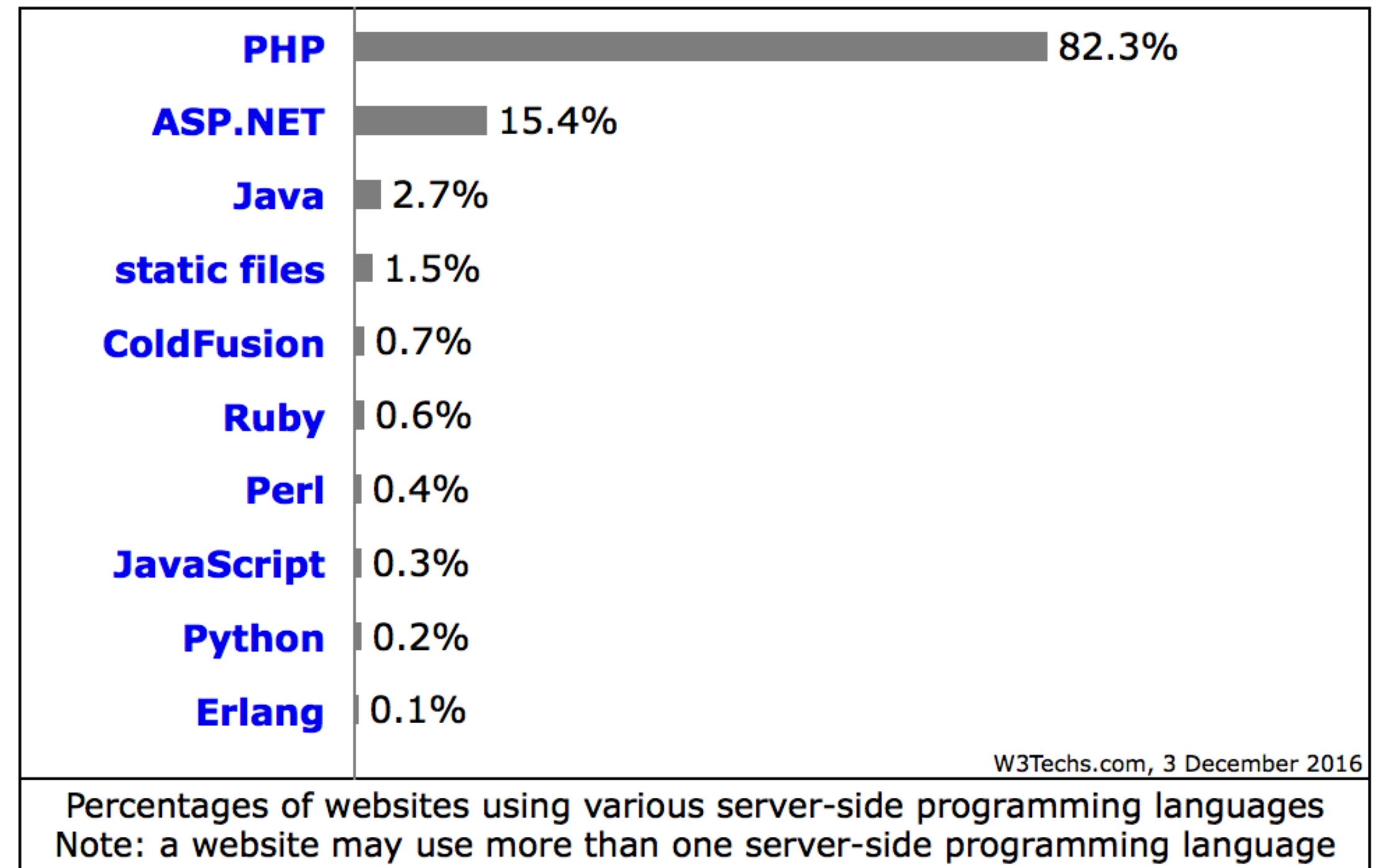
# About Me

---

- Author of Yaf, Yar, Yac, Yaconf, Taint, Lua, etc
- Maintainer of APC, Zend Opcache, Msgpack
- PHP core developer since 2011
- Zend consultant since 2013
- Core author of PHP7
- Chief Architect at Lianjia

# PHP

- 20 years history
- Most popular Web service program language
- Over 82% sites are use PHP as server side program language
- Latest version is PHP-7.1



# PHP71

---

- Nullable types
- Void return type
- Iterable pseudo-type
- Class constant visibility modifiers
- Specify keys in list()
- Square bracket syntax for list()
- Catching multiple exception types
- Missing arguments Exception
- Warn about invalid strings in arithmetic
- Generalize support of negative string offsets
- ....

# Nullable types

- An enhancement for typehints

```
function answer(): ?int {  
    return null; //ok  
}  
  
function answer(): ?int {  
    return 42; // ok  
}  
  
function answer(): ?int {  
    return new stdClass(); // error  
}
```

return type

```
function say(?string $msg) {  
    if ($msg) {  
        echo $msg;  
    }  
}  
  
say('hello'); // ok -- prints hello  
say(null); // ok -- does not print  
say(); // error -- missing parameter  
say(new stdClass); //error -- bad type
```

parameters

# Void return types

---

- An enhancement for return type hint

```
function lacks_return(): void {  
    // valid  
}  
  
function returns_nothing(): void {  
    return; // valid  
}  
  
function returns_null(): void {  
    return null; // Fatal error: A void function must not return a value  
}  
  
function returns_one(): void {  
    return 1; // Fatal error: A void function must not return a value  
}
```

# Iterable Pseudo type

---

- Iterable accepts any array or object implementing Traversable
- Iterable can also be used in return type

```
function foo(iterable $iterable) {  
    foreach ($iterable as $value) {  
        // ...  
    }  
}
```

```
function bar(): iterable {  
    return [1, 2, 3];  
}
```



# Class constant visibility modifiers

- Support class constant visibility

```
class Token {  
    // Constants default to public  
    const PUBLIC_CONST = 0;  
  
    // Constants then also can have a defined visibility  
    private const PRIVATE_CONST = 0;  
    protected const PROTECTED_CONST = 0;  
    public const PUBLIC_CONST_TWO = 0;  
  
    //Constants can only have one visibility declaration list  
    private const FOO = 1, BAR = 2;  
}
```



# Specify keys in list()

- An enhancement for list()
- Also works in foreach

```
$powersOfTwo = [1 => 2, 2 => 4, 3 => 8];  
list(1 => $oneBit, 2 => $twoBit, 3 => $threeBit) = $powersOfTwo;  
  
list(  
    CURLOPT_GET => $isGet,  
    CURLOPT_POST => $isPost,  
    CURLOPT_URL => $url  
) = $curlOptions;
```

specify keys in list

```
$points = [  
    ["x" => 1, "y" => 2],  
    ["x" => 2, "y" => 1]  
];  
  
foreach ($points as list("x" => $x, "y" => $y)) {  
    echo "Point at ($x, $y)", PHP_EOL;  
}
```

specify keys in list with foreach

# Square backtrack for list()

---

- Continuation syntax for short array syntax introduced in 5.4
- Also works in foreach too

```
list($a, $b, $c) = array(1, 2, 3);  
[$a, $b, $c] = [1, 2, 3];
```

```
list("a" => $a, "b" => $b, "c" => $c) = array("a" => 1, "b" => 2, "c" => 3);  
["a" => $a, "b" => $b, "c" => $c] = ["a" => 1, "b" => 2, "c" => 3];
```

```
list($a, $b) = array($b, $a);  
[$a, $b] = [$b, $a];
```

# Catching multiply exception types

- Allow catching multiply exception types in single catch

```
try {  
    // Some code...  
} catch (ExceptionType1 $e) {  
    // Code to handle the exception  
}  
catch (ExceptionType2 $e) {  
    // Same code to handle the exception  
}  
catch (Exception $e) {  
    // ...  
}
```

normal way

```
try {  
    // Some code...  
} catch (ExceptionType1 | ExceptionType2 $e) {  
    // Code to handle the exception  
}  
catch (\Exception $e) {  
    // ...  
}
```

PHP71 way

# Missing arguments exception

- Disable calling “user” functions with insufficient actual parameters

```
function foo($a) {  
    var_dump($a);    // NULL + Warning: Undefined variable: a  
    var_dump($a);    // NULL + Warning: Undefined variable: a  
}  
foo();               // Warning: Missing argument 1 for foo()
```

before 7.1

```
function foo($a) {  
    var_dump($a);    // not executed  
    var_dump($a);    // not executed  
}  
foo();               // throw Error("Too few arguments to function foo(), 0 passed in %s on line
```

after 7.1

# Warn about invalid strings in arithmetic

---

- Produce E\_NOTICE or E\_WARNING when using invalid numeric strings with arithmetic operators

```
$numberOfApples = "10 apples" + "5 pears";
```

```
Notice: A non well formed numeric string encountered in example.php on line 3
```

```
$numberOfPears = 5 * "orange";
```

```
Warning: A non-numeric string encountered in example.php on line 3
```



# Generalize support of negative string offsets

- Support of negative string offsets when it make sense

```
$str='abcdef';  
var_dump($str[-2]); // => string(1) "e"  
  
$str{-3}='.';  
var_dump($str);           // => string(6) "abc.ef"  
  
var_dump(isset($str{-4})); // => bool(true)  
  
var_dump(isset($str{-10})); // => bool(false)
```

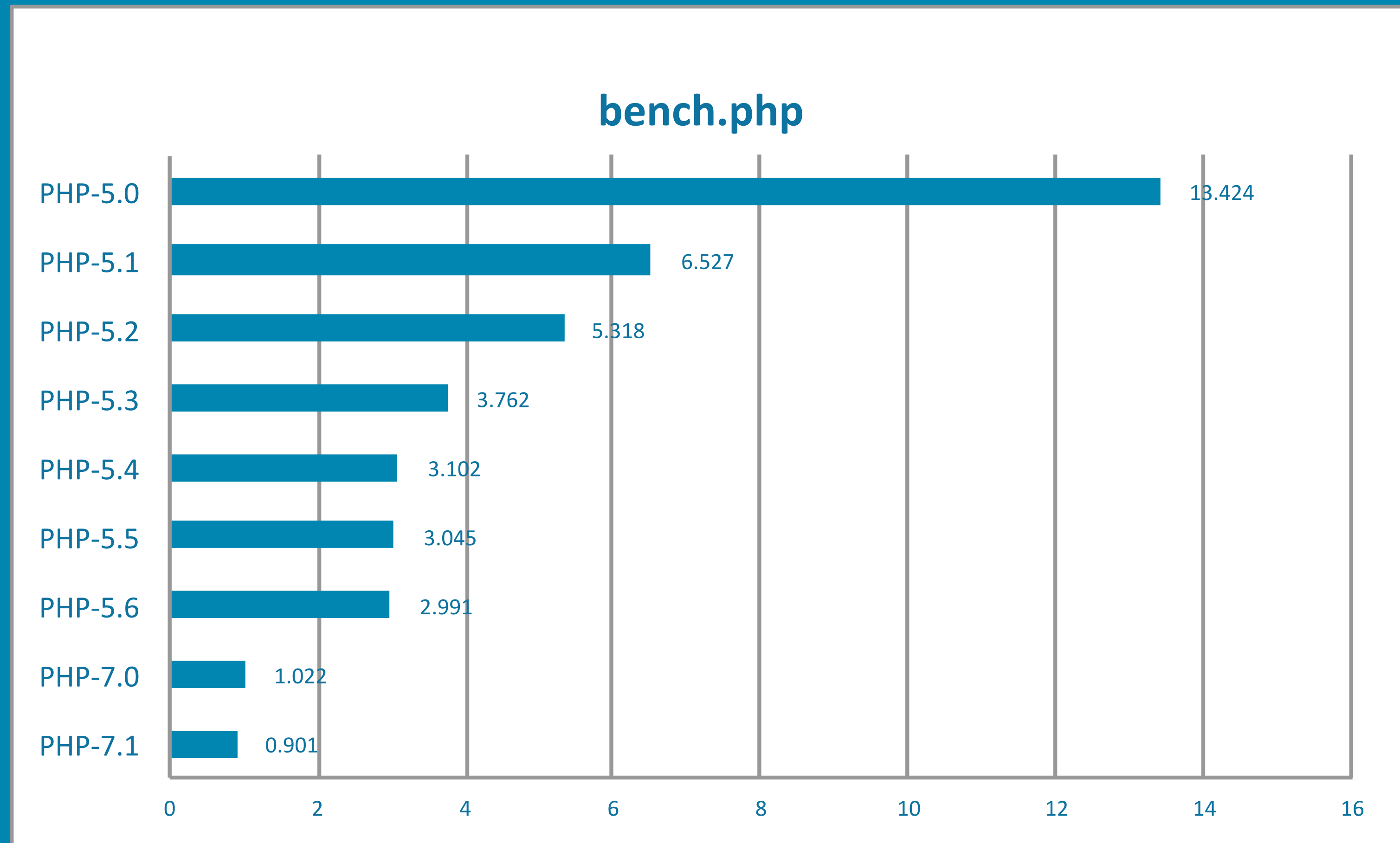
strpos
stripos
substr_count
grapheme_strpos
grapheme_stripos
grapheme_extract
iconv_strpos
file_get_contents
mb_strimwidth
mb_ereg_search_setpos
mb_strpos
mb_stripos

# One More thing



# PHP71 Performance

- Over 10% Performance improvement



# Type specific opcode handlers

`$a = $b + $c; //what if both $b and $c are int`

```
ZEND_VM_HANDLER(1, ZEND_ADD, CONST|TMPVAR|CV, CONST|TMPVAR|CV)
{
    USE_OPLINE
    zend_free_op free_op1, free_op2;
    zval *op1, *op2, *result;

    op1 = GET_OP1_ZVAL_PTR_UNDEF(BP_VAR_R);
    op2 = GET_OP2_ZVAL_PTR_UNDEF(BP_VAR_R);
    if (EXPECTED(Z_TYPE_INFO_P(op1) == IS_LONG)) {
        if (EXPECTED(Z_TYPE_INFO_P(op2) == IS_LONG)) {
            result = EX_VAR(opline->result.var);
            fast_long_add_function(result, op1, op2);
            ZEND_VM_NEXT_OPCODE();
        } else if (EXPECTED(Z_TYPE_INFO_P(op2) == IS_DOUBLE)) {
            result = EX_VAR(opline->result.var);
            ZVAL_DOUBLE(result, ((double)Z_LVAL_P(op1)) + Z_DVAL_P(op2));
            ZEND_VM_NEXT_OPCODE();
        }
    } else if (EXPECTED(Z_TYPE_INFO_P(op1) == IS_DOUBLE)) {
        if (EXPECTED(Z_TYPE_INFO_P(op2) == IS_DOUBLE)) {
            result = EX_VAR(opline->result.var);
            ZVAL_DOUBLE(result, Z_DVAL_P(op1) + Z_DVAL_P(op2));
            ZEND_VM_NEXT_OPCODE();
        } else if (EXPECTED(Z_TYPE_INFO_P(op2) == IS_LONG)) {
            result = EX_VAR(opline->result.var);
            ZVAL_DOUBLE(result, Z_DVAL_P(op1) + ((double)Z_LVAL_P(op2)));
            ZEND_VM_NEXT_OPCODE();
        }
    }

    SAVE_OPLINE();
    if (OP1_TYPE == IS_CV && UNEXPECTED(Z_TYPE_INFO_P(op1) == IS_UNDEF)) {
        op1 = GET_OP1_UNDEF_CV(op1, BP_VAR_R);
    }
    if (OP2_TYPE == IS_CV && UNEXPECTED(Z_TYPE_INFO_P(op2) == IS_UNDEF)) {
        op2 = GET_OP2_UNDEF_CV(op2, BP_VAR_R);
    }
    add_function(EX_VAR(opline->result.var), op1, op2);
    FREE_OP1();
    FREE_OP2();
    ZEND_VM_NEXT_OPCODE_CHECK_EXCEPTION();
}
```

ZEND\_ADD

```
ZEND_VM_TYPE_SPEC_HANDLER(ZEND_ADD, (res_info == MAY_BE_LONG &&
{
    USE_OPLINE
    zval *op1, *op2, *result;

    op1 = GET_OP1_ZVAL_PTR_UNDEF(BP_VAR_R);
    op2 = GET_OP2_ZVAL_PTR_UNDEF(BP_VAR_R);
    result = EX_VAR(opline->result.var);
    ZVAL_LONG(result, Z_LVAL_P(op1) + Z_LVAL_P(op2));
    ZEND_VM_NEXT_OPCODE();
}
```

ZEND\_ADD\_LONG\_LONG

# Type Inference system

- Part work of our JIT in 2013

```
function type() {  
    $a = 123;  
  
    //no change to $a  
  
    .....  
  
    return $a + 3;  
}
```

normal

```
function type($flag) {  
    $a = 123;  
  
    if ($flag) {  
        $a = "string";  
    }  
  
    return $a + 3;  
}
```

branch

```
function type($flag) {  
    $a = 123;  
start:  
    $b = $a + 3;  
    if ($flag) {  
        $a = "string";  
        goto start;  
    }  
  
    return $b;  
}
```

loop

# Static Single-Assignment Form

- An intermediate representation, variable is assigned exactly once
- Explicit use-def chain

```
function type() {  
  
    $a = 123;  
    $b = $a + 3;  
    $a = (string)$b;  
    $c = $a + 2;  
  
    return c;  
}
```

```
function type() {  
  
    $a1 = 123;  
    $b1 = $a1 + 3;  
    $a2 = (string)$b1;  
    $c1 = $a2 + 2;  
  
    return $c1;  
}
```

```
function type($flag) {  
    $a = 123;  
  
    if ($flag) {  
        $a = "string";  
    }  
  
    return $a + 3;  
}
```

```
function type($flag) {  
    $a1 = 123;  
  
    if ($flag) {  
        $a2 = "string";  
    }  
  
    return $a(?) + 3;  
}
```

# Type Inference

- What's ? 's type at ? point
- Computed in compiling time

```
function type() {  
  
    $a1 = 123;  
    $b1 = $a1 + 3;  
    $a2 = (string)$b1;  
    $c1 = $a2 + 2;  
    return $c1;  
  
    // $a1 MAY_BE_LONG  
    // $b1 MAY_BE_LONG  
    // $a2 MAY_BE_STRING  
    // $c1 MAY_BE_LONG  
}
```

```
function type($flag) {  
    $a1 = 123;  
  
    if ($flag) {  
        $a2 = "string";  
    }  
  
    return $a3 + 3;  
  
    // $a1 MAY_BE_LONG  
    // $a2 MAY_BE_STRING  
    // $a3 = PI($a1, $a2) MAY_BE_LONG|MAY_BE_STRING  
}
```



# Type Specific opcode handler

- Use type specific opcode handler if possible

```
function type() {  
    $a = 123;  
    $b = $a + 3;           //ZEND_ADD_LONG_NO_OVERFLOW_SPEC_CONST_TMPVARCV_HANDLER  
    $a = (string)$b;  
    $c = $a + 2;           //ZEND_ADD  
  
    return $c;  
}
```

# Type Inference system

---

- A tedious work
- And we can only get ~30% type-infos in WP

```
$a1 = (string)$dummy; // $a1 MAY_BE_STRING
$a2 = ++(int)$dummy; // $a2 MAY_BE_LONG/MAY_BE_DOUBLE
$a3 = strlen($dummy); // $a3 MAY_BE_LONG/MAY_BE_NULL
$a4 = trim($dummy); // $a4 MAY_BE_STRING
$a5 = explode(",", $dummy); // $a5 MAY_BE_NULL/MAY_BE_FALSE /MAY_BE_ARRAY
$a6 = array_pop($dummy); // $a6 UNKNOWN
```



# Type Specific opcode handlers

---

- ZEND\_ADD(SUB|MUL)
- ZEND\_PRE\_INC(DEC)
- ZEND\_POST\_INC(DEC)
- ZEND\_IS\_(NOT\_)EQUAL
- ZEND\_IS\_SMALLER(\_OR\_EQUAL)
- ZEND\_QM\_ASSIGN
- ZEND\_SEND\_VAR(\_EX)
- ZEND\_FETCH\_DIM\_R

# And dozens of minor improvements

---

- More packed array constructions
- Constant propagation based on DFA
- Return type checks eliding
- dozens of minor improvements even I can not recall

Q&A