

PHP 5.3

Johannes Schlüter



PHP Roadmap



PHP 4



Photo: Patricia Hecht

PHP 5.2

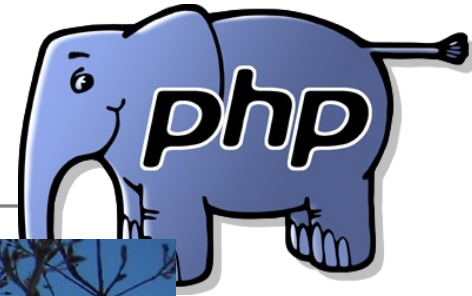


Photo: Gabriele Kantel

PHP 5.3

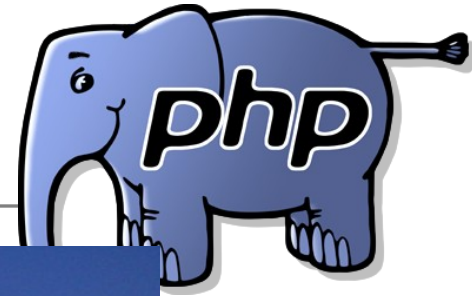


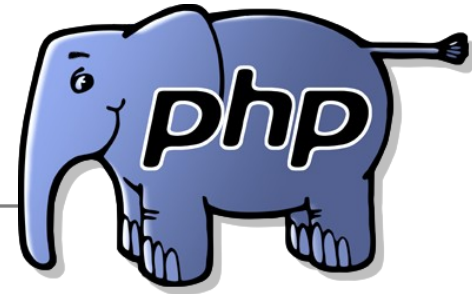
Photo: Jamie Sanford

PHP 6

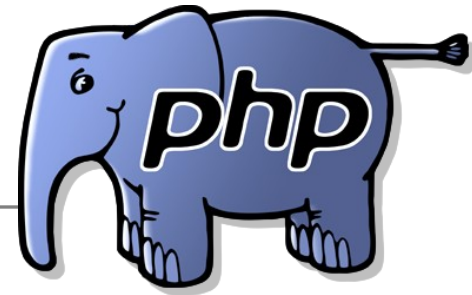


Photo: G4Glenno (flickr)

New language (syntax) features



- | Namespaces
- | Closures
- | Compile time constants
- | late static binding
- | New operators
 - | ?:
 - | goto
 - | NOWDOC syntax, HEREDOC with quotes
- | dynamic static calls



New functionality

| New extensions

- | SQLite 3, fileinfo, intl, enchant, phar

| Added “Unix” functions to Windows

- | fnmatch, symlink, readlink, time_sleep_until, stream_socket_pair, ...

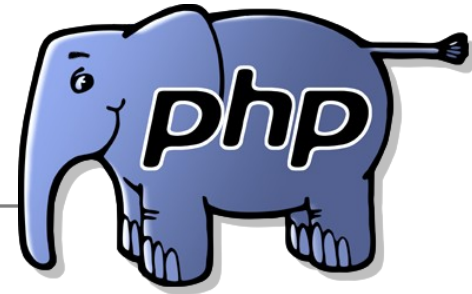
| Improved SPL functionality

| Improved php.ini handling

| E_DEPRECATED error level

| ...

Infrastructure improvements

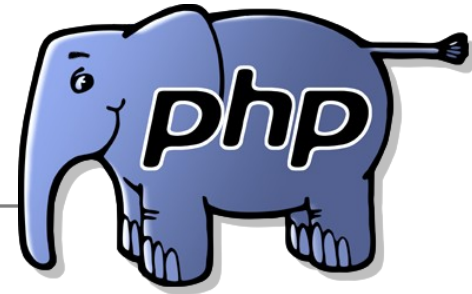


- | Improved php.ini handling
- | FastCGI always available with CGI SAPI
- | New support for litespeed http server
- | mysqlnd as PHP-specific replacement for libmysql
- | Improved performance

Namespaces



Namespaces – The Reasons



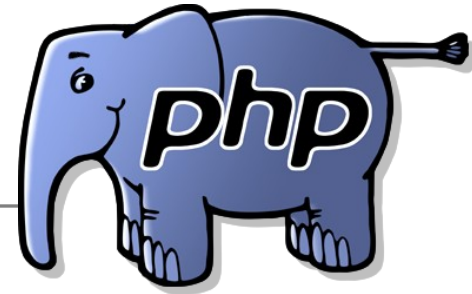
- | Class names have to be unique per running script
- | PHP runtime developers tend to add class with generic names
 - | “Date”
- Class names tend to be long
 - | MyFramework_Category_Helper_FooBar

Namespaces – Design Ideas



- PHP's namespace implementation is resolving names (mostly) at compile-time
 - They should have no (measurable) impact on the runtime performance
 - new \$string; won't know anything about namespaces
 - neither do callbacks or anything else which takes class names as string
- When using namespaces the *namespace* declaration has to be in at the beginning of the file
- There can be multiple namespaces per file

Namespace-able elements

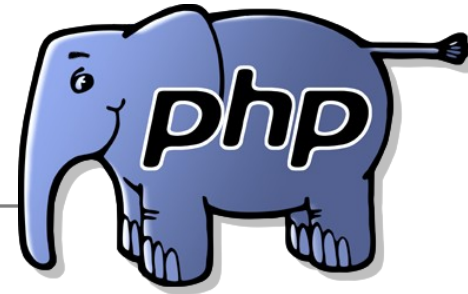


Namespaces can contain classes, functions and constants

```
<?php
namespace Foo;
const ANSWER = 42;
class C { /* ... */ }
function f() { }
?>
```

```
<?php
echo Foo\ANSWER;
new Foo\C();
Foo\f();
?>
```

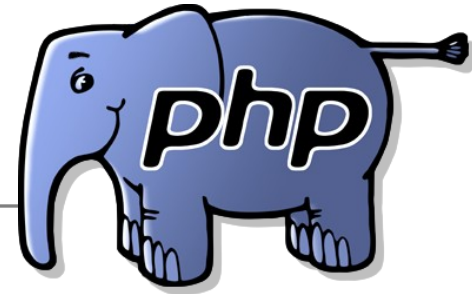
Namespace syntax



You can use curly braces to define multiple namespaces:

```
<?php
namespace Foo {
    class C { /* ... */ }
}
namespace Bar {
    class C { /* ... */ }
}
?>
```

Namespaces – an example



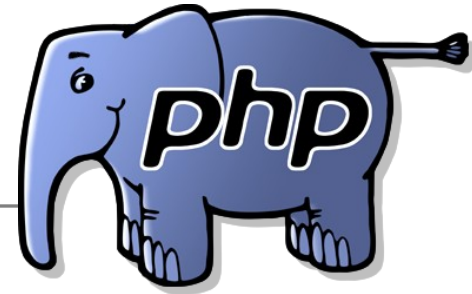
```
foo.php
<?php
namespace MyFramework\someModule;
class Foo { /* ... */ }
?>
```

The compiler translates this to
MyFramework\someModule\Foo

```
bar.php
<?php
class Bar extends MyFramework\someModule\Foo { /* ...
*/ }
?>
```

We can use the full name from within another file

Namespaces – an example



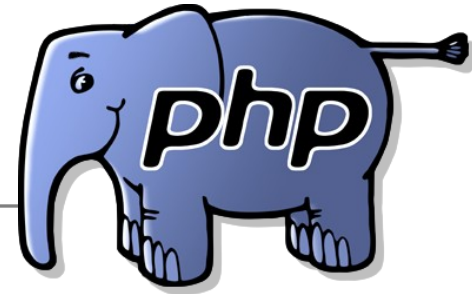
```
foo.php
<?php
namespace MyFramework\someModule;
class Foo { /* ... */ }
?>
```

This will be prefixed with the namespace

```
bar.php
<?php
namespace MyFramework\someModule;
class Bar extends Foo { /* ... */ }
?>
```

As will this, so we are referring to our previously declared class

Accessing the same Namespace



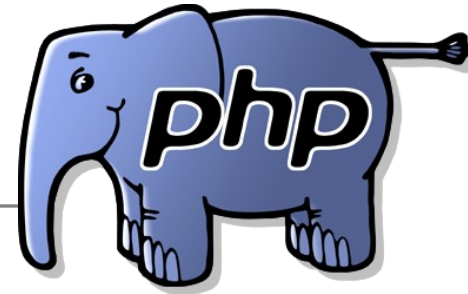
For usage in strings use the magic `__NAMESPACE__` constant.

```
call_user_func(
    array(__NAMESPACE__.'\some_class',
        'method'),
    $param1, $param2, $param3);
```

For accessing elements of the same namespace you may use “namespace”

```
return new namespace\some_class();
```

Namespaces and globals

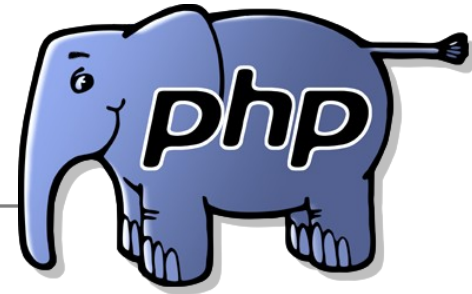


```
<?php
namespace MyFramework\someModule;
function strlen($param) {
    return 0;
}

echo strlen("hello world");

echo \strlen("hello world");
echo some\other\space\strlen("hello world");
?>
```

Namespaces – *Using* classes

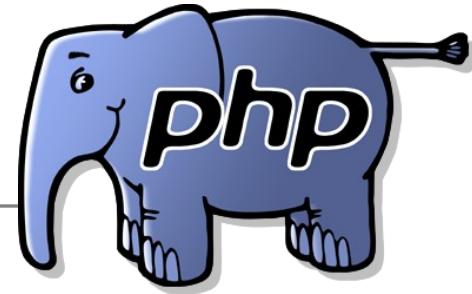


- | Often you want to use classes from other namespaces
 - | Typing the fully qualified name is long

- | “use” creates an alias which can be used in the given file

- | **No** `use foo*;`

Use example



- use `Foo\Bar`;

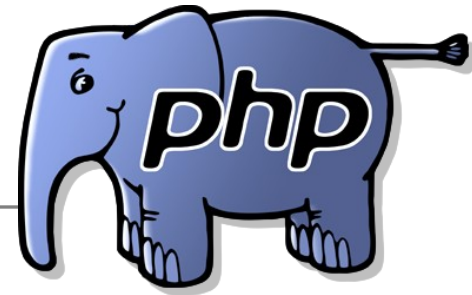
Use class *Bar* from *Foo* and make *Bar* the alias

- use `Foo\Bar as Baz`;

As above but make *Baz* the alias

- use `RecursiveIteratorIterator as RII`;

It's just an alias so we can create an alias to global classes, too



Use namespace

```
| namespace foo\bar;  
class class1 { // foo\bar\class1  
class class2 { // foo\bar\class2
```

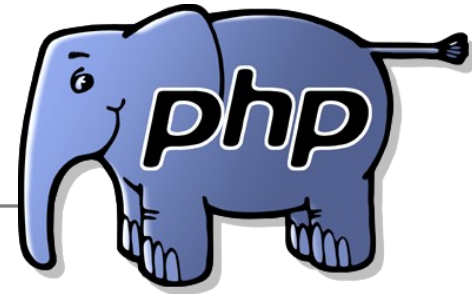
```
| use foo\bar;  
new bar\class1();  
new bar\class2();
```

```
| No "as" allowed!
```

Closures



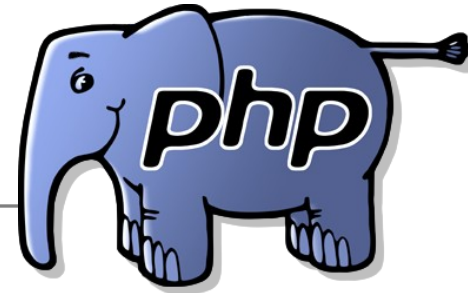
Callbacks



```
| $data = array(  
    array('sort' => 2, 'foo' => 'some value'),  
    array('sort' => 1, 'foo' => 'other value'),  
    array('sort' => 3, 'foo' => 'one more'),  
    /* ... */  
);
```

| Task: Sort the array `$data` using the `sort` field of the array

Callbacks



```
bool usort (array &$array, callback $cmp_function)
```

This function will sort an array by its values using a user-supplied comparison function.

→ Problem: Where to put the callback function?

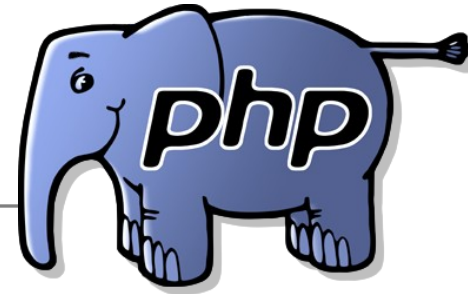
eval is evil, so is create_function



| `create_function($a, $b)` equals to `eval("function lambda($a) { $b }");`

- It is probably insecure, won't work nicely with OpCode caches, editing the code as string leads to mistakes (no proper highlighting in an editor), ...

Anonymous functions

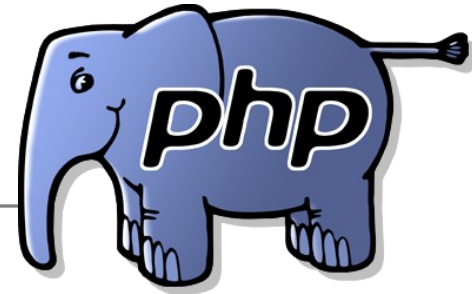


```
$callback = function($a, $b)
{
    if ($a['sort'] == $b['sort']) {
        return 0;
    }

    return ($a['sort'] < $b['sort']) ? -1 : 1;
};

usort($data, $callback);
```

A closer look



```
| var_dump($callback);
```

```
object(Closure)#1 (0) {  
}
```

- Anonymous Functions/Closures are implemented as Objects of the type “Closure”
- Any object with an `__invoke()` method can be used as closure



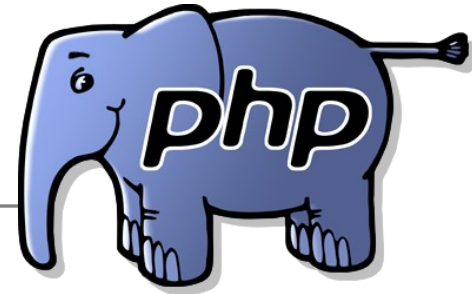
Closures

```
function fancy_count($arr) {  
    $count = 0;  
  
    $callback = function($dat) use (&$count)  
    {  
        $count++;  
    };  
  
    array_walk($arr, $callback);  
    return $count;  
}  
echo fancy_count(array(0,1,2,3,4)); // 5
```

And More!



?: Operator



- | Discussed for a long time under the name “issetor”
- | shortcut for `$foo ? $foo : $bar;`
- | Main usage setting default values for request parameters
 - | `$id = $_GET['id'] ?: 0;`
- | Problem: Might emit an `E_STRICT` error unlike the “traditional” way
 - | `$id = isset($_GET['id']) ? $_GET['id'] : 0;`



Dynamic Static Calls

- As of PHP 5.3 the class name for calling a static class element can be a variable

- ```
$name = 'Classname';
$name::method();
```

- If an object is passed its class is used

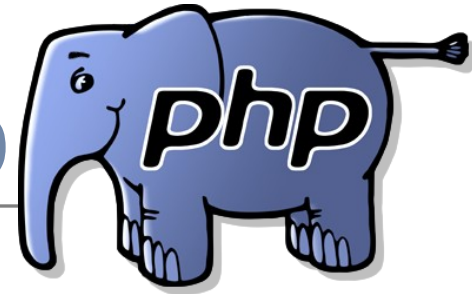
- ```
$o = new Class();  
$o::method();
```

- Use the fully qualified class name in Strings

- ```
namespace Foo;
$name = __NAMESPACE__.'\'.$name;
$name::method();
```

# New error level: E\_DEPRECATED

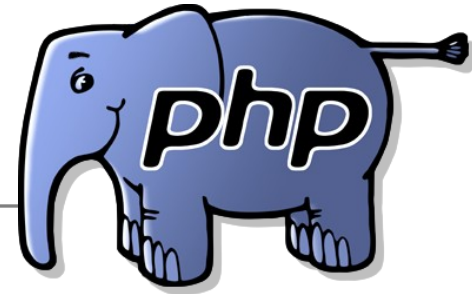
---



- Used by PHP to mark functionality which might (or will) be removed with later releases
- E\_STRICT (should) only include errors related to “bad” coding practices
- E\_DEPRECATED is part of E\_ALL
  - Continue to develop using `error_reporting = E_ALL!`
  - Fix errors to ease migration to future releases

# Improved date support

---



## | Date arithmetics

- | `DateInterval` represents the difference between two `Dates`
- | `DateTime::add()`, `DateTime::sub()` can be used to apply an interval to a date
- | `DatePeriod` represents a period of time and allows iteration



## Dateperiod

---

```
$begin = new DateTime('2007-12-31');
$end = new DateTime('2009-12-31 23:59:59');

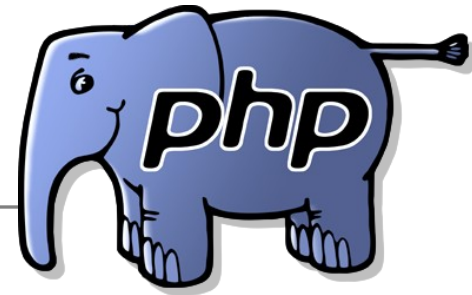
$interval = DateInterval::createFromDateString(
 'last thursday of next month');

$period = new DatePeriod($begin, $interval, $end,
 DatePeriod::EXCLUDE_START_DATE);

foreach ($period as $dt) {
 echo $dt->format("l Y-m-d H:i:s\n");
}
```

# Improved SPL support

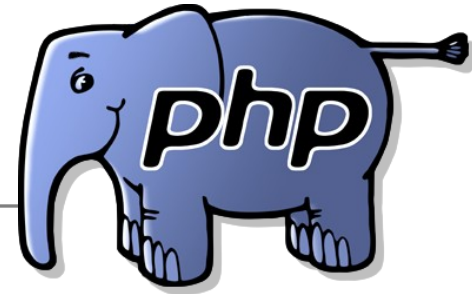
---



- | SPL is the “Standard PHP Library”
- | Until 5.3 it mainly focused on iterators
- | PHP 5.3 introduces
  - | SplDoublyLinkedList
  - | SplStack
  - | SplQueue, SplPriorityQueue
  - | SplHeap, SplMinHeap, SplMaxHeap
  - | SplFixedArray

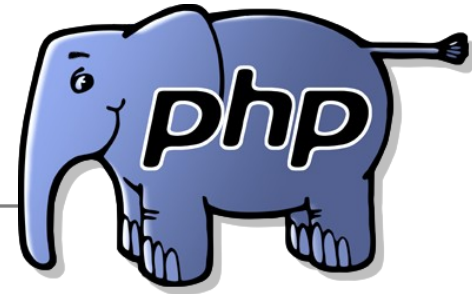
# PHAR – PHP Archive

---



- | Similar to Java's JAR
- | Possibly the future default way for distributing applications
- | PHAR files can use a custom file format or be based on tar or zip archives
- | PHAR includes a flexible front controller system to do the mapping from request to a file inside the phar

# Creating phar archives



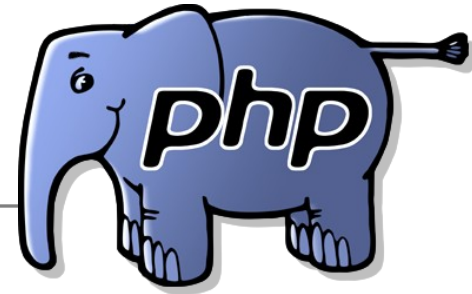
```
try {
 $phar = new Phar('myapp.phar');
 $phar['index.php'] =
 '<?php echo "Welcome to the index!"; ?>';

 $phar['page2.php'] =
 '<?php echo "This is page 2."; ?>';

} catch (Exception $e) {
 echo $e;
}
```

## ... or from command line

---



```
| $ phar pack -f myapp.phar index.php directory/
```

```
| $ phar list -f myapp.phar
```

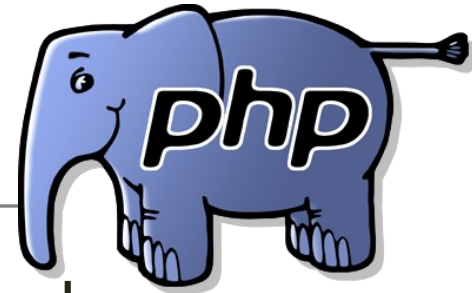
```
|-phar:///index.php
```

```
|-phar:///directory/file.php
```

```
|-phar:///directory/image.png
```

# Stubs

---

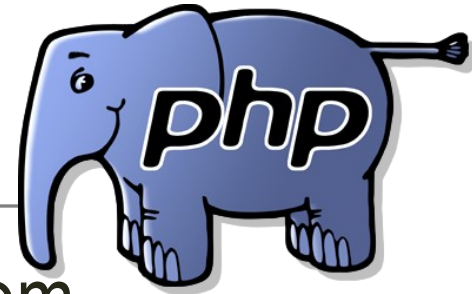


■ A “stub” is put on top of the file and executed when called

```
■ $phar->setStub(
 '<?php
 echo "Hello World!";
 __HALT_COMPILER();
 ?>'
);
```

■ <http://example.com/myapp.phar>

# webPhar



Phar has a front-controller for 1:1 mapping from URLs to files in the phar

```
$phar->setStub(
 '<?php
 Phar::webPhar();
 __HALT_COMPILER();
 ?>'
);
```

<http://example.com/myapp.phar/index.php>

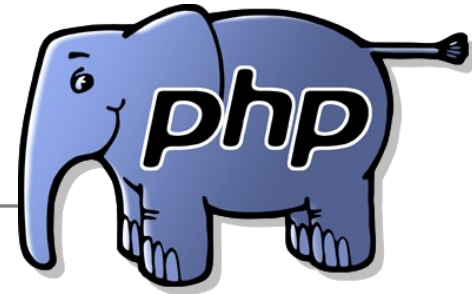
<http://example.com/myapp.phar/page2.php>

<http://example.com/myapp.phar/directory/image.jpg>

# Garbage Collection



# Reference counting



- PHP's memory handling is based on reference counting.
  - A PHP variable consists of a label (\$label) and the variable container (zval structure)
  - PHP is counting the number of labels pointing to the same variable container

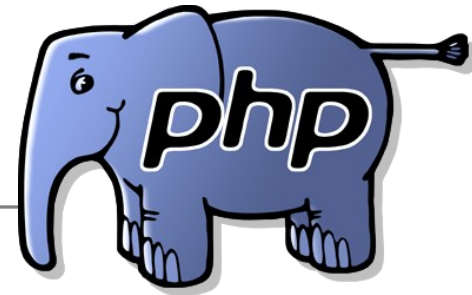
```
<?php
$a = new stdClass();
$b = $a;
unset($a);
unset($b);
```

reference count = 1

reference count = 2

reference count = 1

reference count = 0



# Cyclic references

---

```
| $a = new stdClass();
 $b = new stdClass();
```

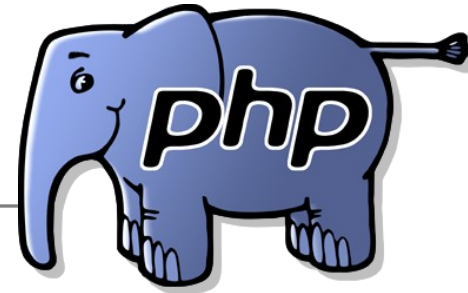
```
$a->b = $b;
$b->a = $a;
```

```
unset($a); unset($b);
```

- | Using reference counting PHP can't see that the objects aren't referenced from somewhere else
  - | Using refcount PHP can't free the memory till the end of the script run

# New garbage collector

---



- | Now PHP can search for cyclic references from time to time
- | To en-/disable GC use
  - | `zend.enable_gc` php.ini setting
  - | `gc_enable()`, `gc_disable()`
- | If enabled the GC is trigger automatically or by
  - | `gc_collect_cycles()`
- | For complex applications this can reduce memory usage by the cost of CPU time
  - | Unit-Tests!

Backwards  
Compatibility



# We try to give our best

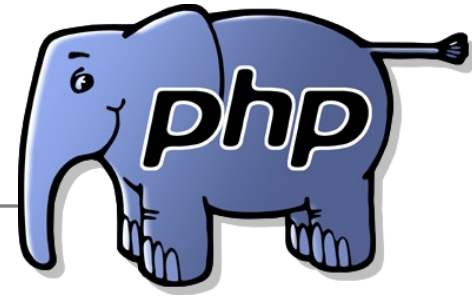
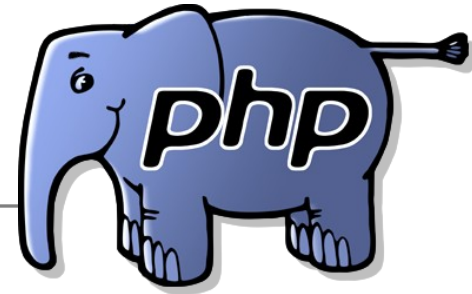


Photo: fireflythegreat (flickr)

- | Sometimes we have to break things
- | Sometimes we make mistakes
- | We need real life tests!

# Things known to break

---



- | New Keywords

- | *namespace, closure*

- | New global identifiers

- | New extensions

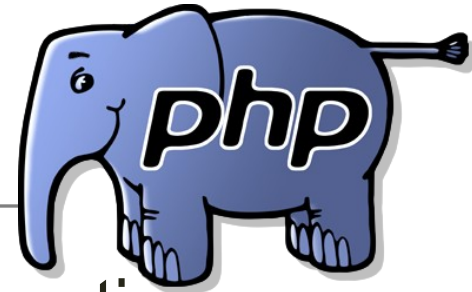
- | New *Closure* class

- | Extensions dropped

- | ncurses, fpdf, dbase, fbsql, ming, msql

- | Dropped zend.ze1\_compatibility\_mode

# Function parameters



## Changes to parameter parsing by internal functions

### Functions interpreted Objects as Arrays

*current, next, key, each, reset, end, natsort, natcasesort, usort, uasort, uksort, array\_flip, array\_unique*

### *call\_user\_func\_array* stricter about references

```
function foo(&$param) {}
```

```
call_user_func_array("foo", array(23));
```

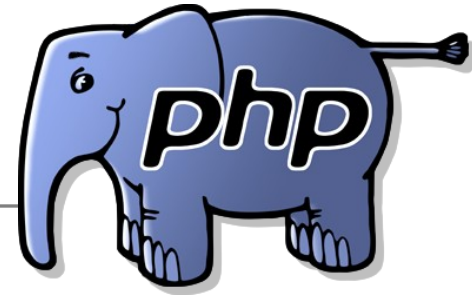
```
call_user_func_array("foo", array($variable));
```

Summary



# Much new stuff but still faster!?

---



- | Yes!
- | New scanner (based on re2c instead of flex)
- | Improved internal stack usage
- | Improved access to internal data structures
- | VisualStudio 9 builds for Windows
- | ...

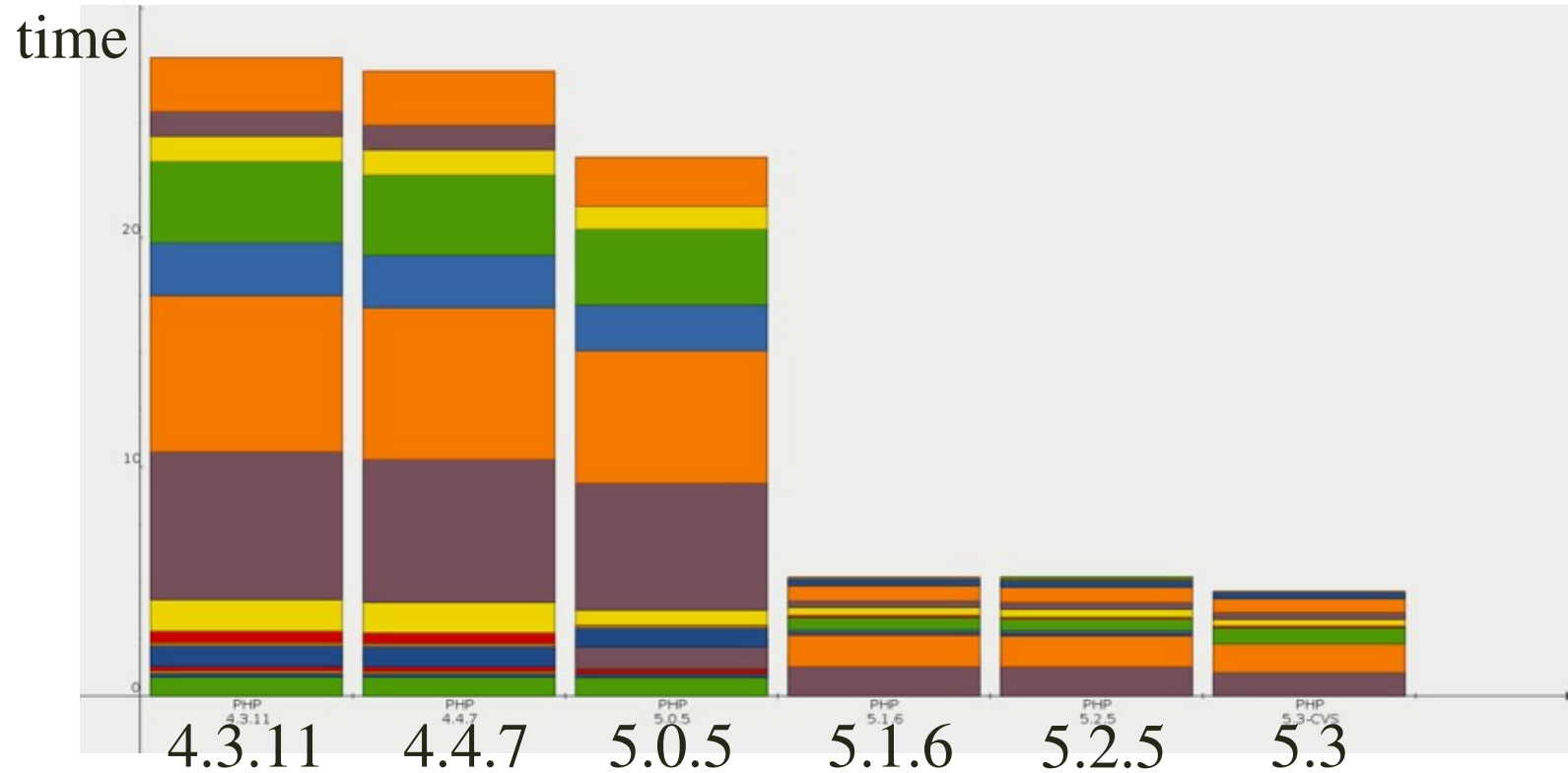
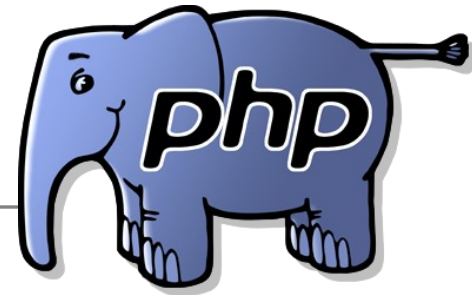


Diagram: Sebastian Bergmann

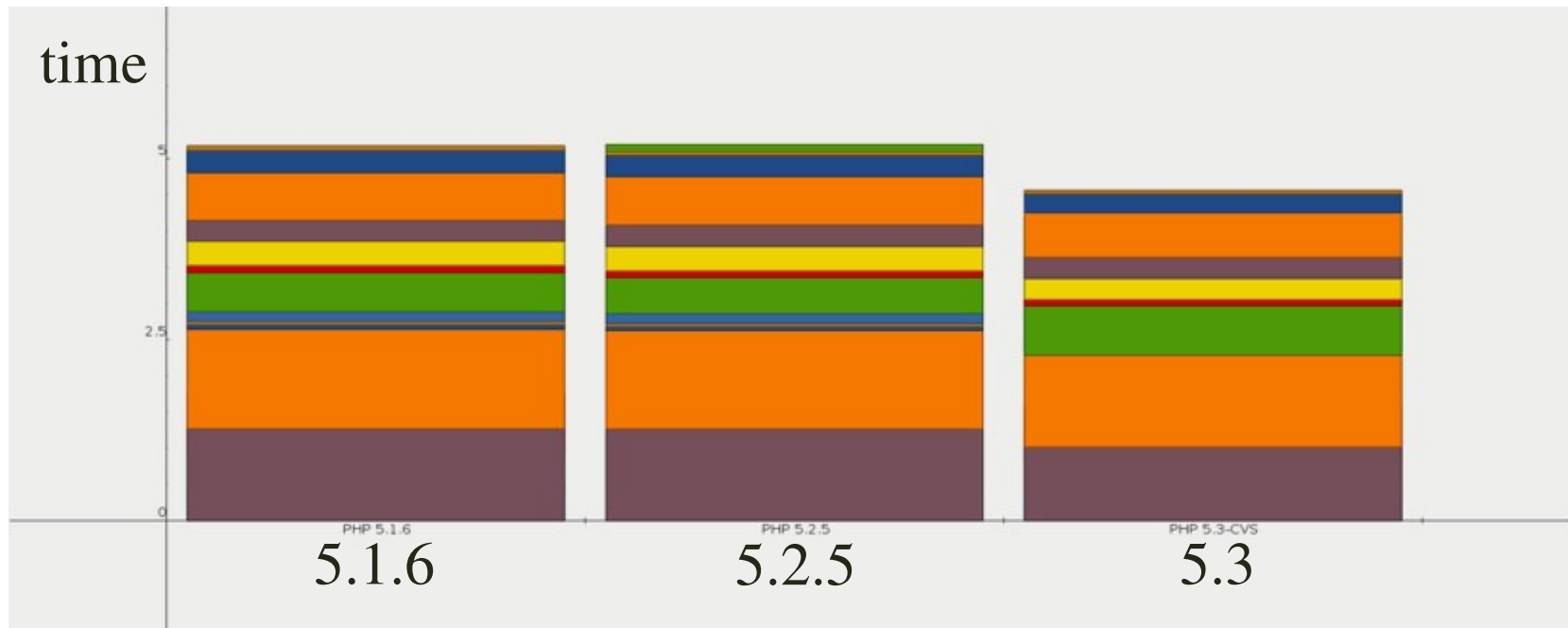
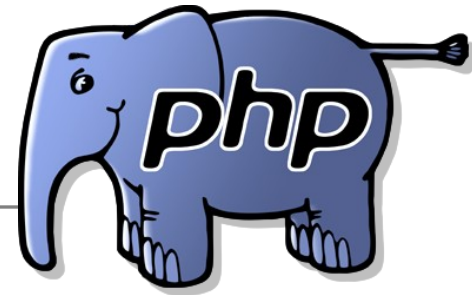


Diagram: Sebastian Bergmann

# Links

---



Downloads:

<http://downloads.php.net/johannes/> (Source)

<http://windows.php.net/> (Windows Binaries)

<http://snaps.php.net/> (Latest snapshots)

Documentation:

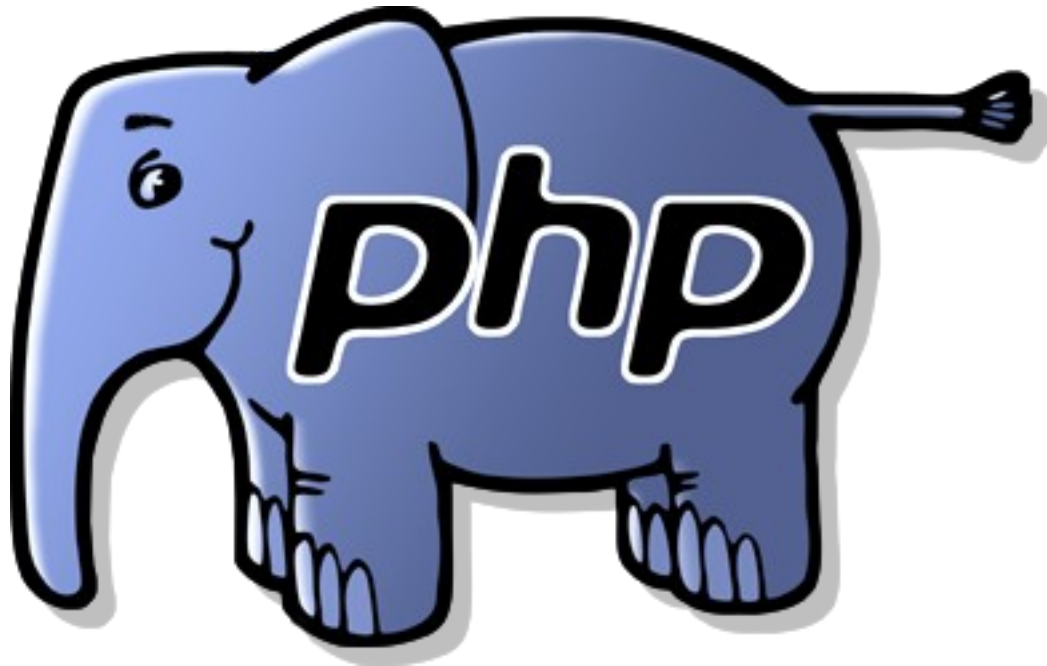
php-src/UPGRADING

<http://php.net/manual/>

<http://wiki.php.net/doc/scratchpad/upgrade/53>

????





**Merci pour votre attention!**

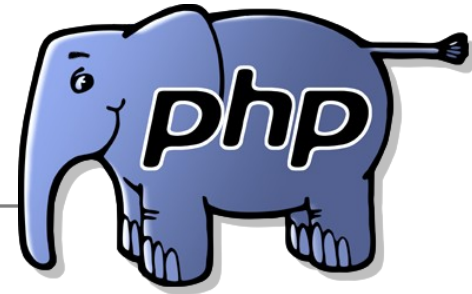
Johannes Schlüter  
johannes@php.net

intl



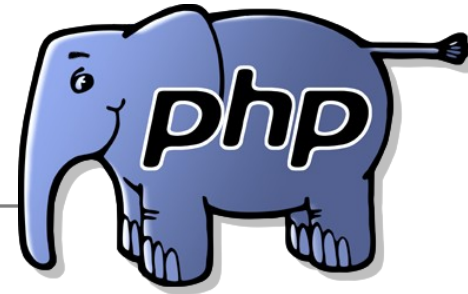
# What is intl?

---



- | Internationalization parts of ICU
- | ICU is the base library for PHP 6's Unicode implementation
  
- | Collations
- | Output Formatting
- | Normalization

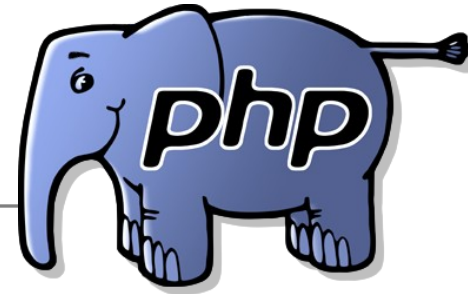
# MessageFormatter



```
| $fmt = new MessageFormatter("en_US",
 "{0,number,integer} monkeys on {1,number,integer} trees
make {2,number} monkeys per tree");
echo $fmt->format(array(4560, 123, 4560/123));
```

```
| 4,560 monkeys on 123 trees make 37.073 monkeys per tree
```

# MessageFormatter

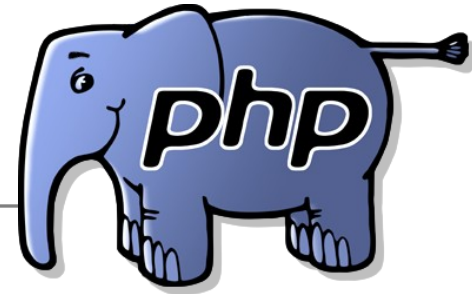


```
| $fmt = new MessageFormatter("de",
 "{0,number,integer} Affen auf {1,number,integer} Bäumen
 ergeben {2,number} Affen pro Baum");
echo $fmt->format(array(4560, 123, 4560/123));
```

| 4.560 Affen auf 123 Bäumen ergeben 37,073 Affen pro Baum

# NOWDOC

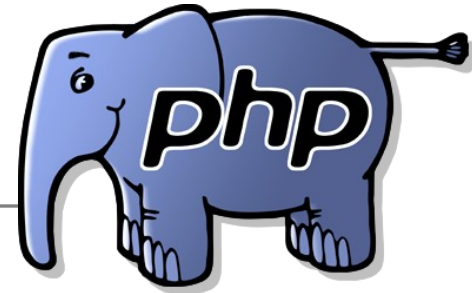
---



- | Similar to HEREDOC but does no expanding of variables
- | 

```
<?php
echo <<<'EOT'
This item costs $US 23.42
EOT;
?>
```
- | This item costs \$US 23.42

# HEREDOC with quotes



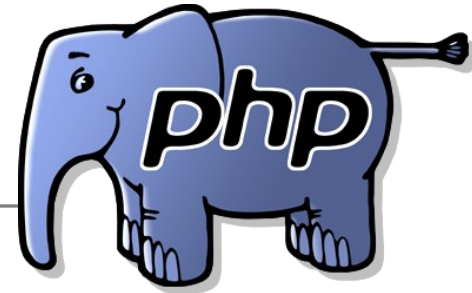
| For being consistent with NOWDOCs HEREDOCs allow passing the name escaped by quotes

```
| <?php
 echo <<<"EOT"
 This item costs $US 23.42
 EOT;
 ?>
```

| Notice: Undefined variable: US in ...  
This item costs 23.42

# Stream-Wrapper Support

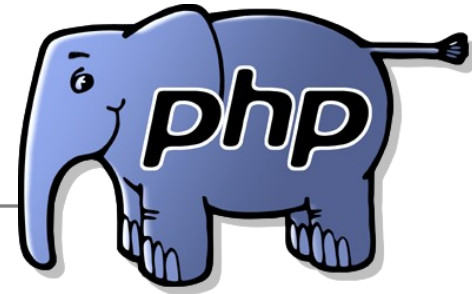
---



- | The `include_path` ini-Setting can include paths provided by stream wrappers
  - | `include_path = http://example.com/files:/local:.`
- | Mainly thought for phar streams

# mysqlnd

---



- | PHP-specific replacement for the MySQL Client library (libmysql)
- | Developed by Sun/MySQL
- | Deeply bound to PHP
  - | Using PHP memory management
  - | Using PHP Streams
  - | No external dependencies
- | Not yet another MySQL extension but an internal library sitting below other extension
  - | Powers ext/mysql, mysqli and pdo\_mysql

# mysqlnd



| To compile PHP using mysqlnd use

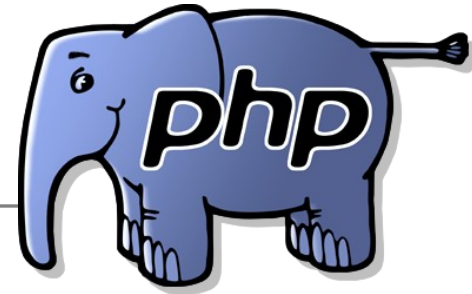
- | --with-mysql=mysqlnd
- | --with-mysqli=mysqlnd
- | --with-pdo-mysql=mysqlnd

| Windows builds use mysqlnd by default

## mysqli

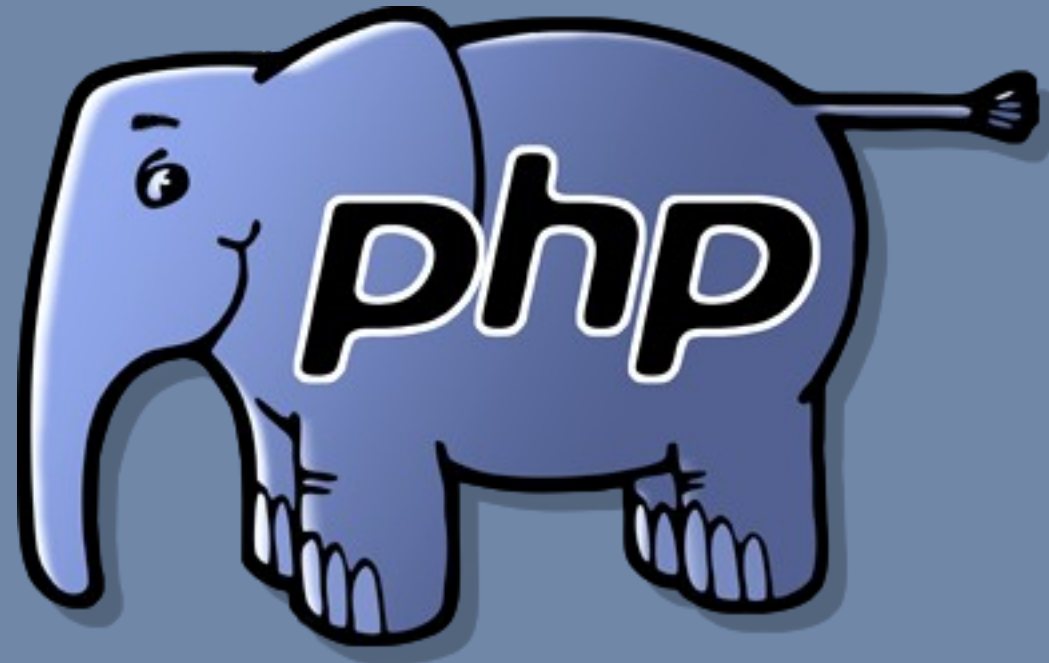
| MySQL Support              | enabled                                   |
|----------------------------|-------------------------------------------|
| Client API library version | mysqlnd 5.0.4-dev - 080501 - \$Revision\$ |
| Active Persistent Links    | 0                                         |
| Inactive Persistent Links  | 0                                         |
| Active Links               | 18                                        |
| Persistent cache           | enabled                                   |

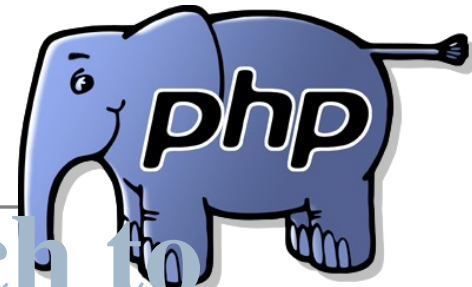
# goto



- | Yes, exactly what the name says
- | But there are few Limitations:
  - | You can only jump within the same code unit (the same function, global space from the same file, technically precise: the same OpArray)
  - | You can jump *out* of blocks but not into blocks
- | `<?php`  
    label: echo 1;  
    goto label;  
    ?>

# Late Static Binding





## Active Record Pattern

---

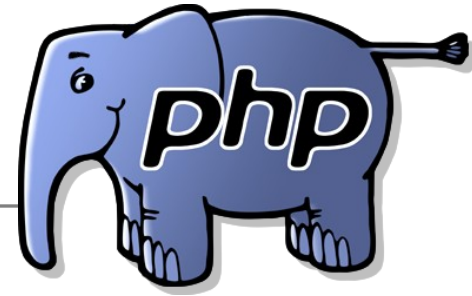
“Active record is an approach to accessing data in a database. A database table or view is wrapped into a class, thus an object instance is tied to a single row in the table.”

**Wikipedia**

[http://en.wikipedia.org/wiki/Active\\_record\\_pattern](http://en.wikipedia.org/wiki/Active_record_pattern)

# Active Record

---

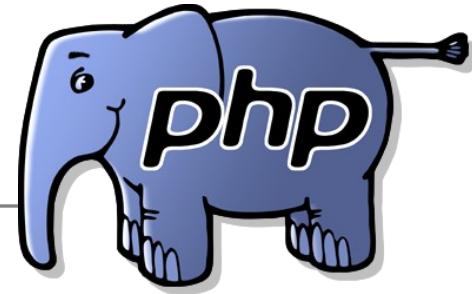


```
| <?php
$user = User::findByID(23);

echo "Hello ".$user->getName()."\n";

$user->setPassword("verysecret");
$user->save();
?>
```

# Late Static Binding

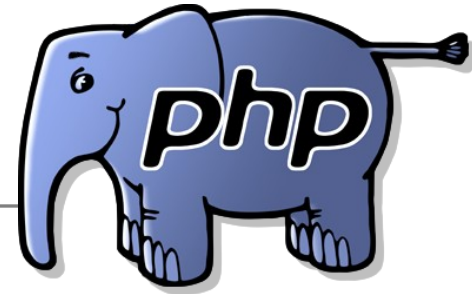


```
<?php
abstract class ActiveRecord {
 static function findById($id) {
 $stable = get_called_class();
 $query = "SELECT * FROM ".$stable
 ." WHERE id=".$id;
 /* */
 }
}

class User extends ActiveRecord {}
class Entry extends ActiveRecord {}

$a = User::findById(2);
$b = Entry::findById(4);
```

# Late Static Binding II



```
class Base {
 public static function m()
 {
 self::printName();
 static::printName();
 }

 static function printName()
 {
 echo __CLASS__;
 }
}
```

Base::m();

**Base**

**Base**

```
class Extended extends Base {
 static function printName()
 {
 echo __CLASS__;
 }
}
```

Extended::m();

**Base**

**Extended**