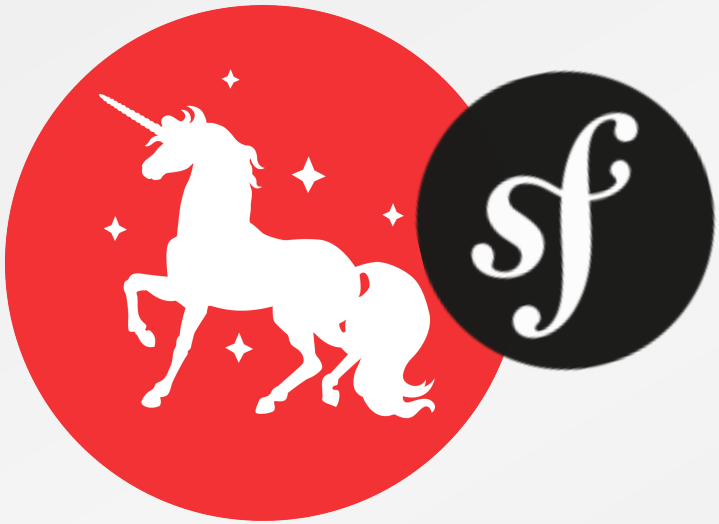


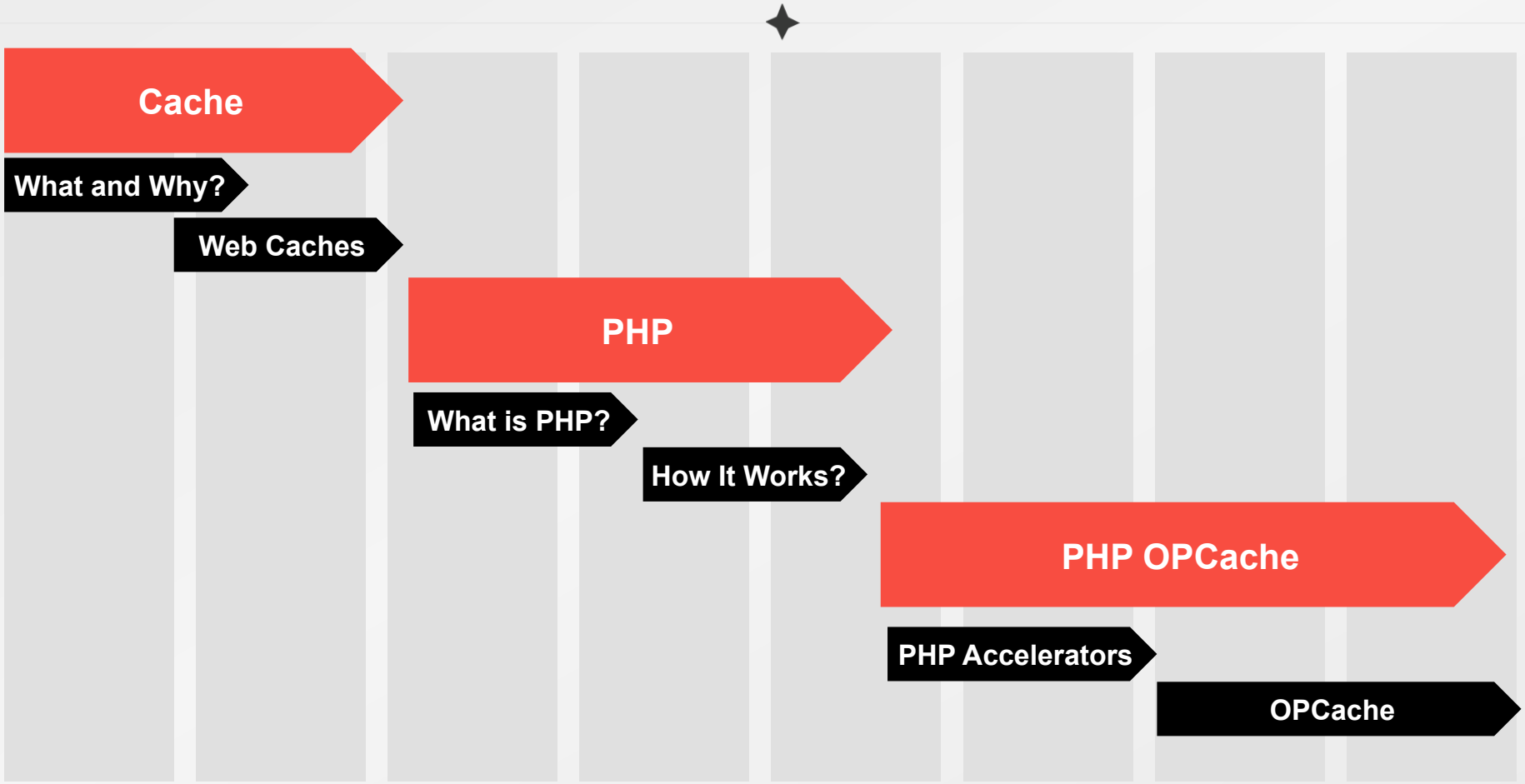
DigitasLBi

Cache - PHP - OPCache

05/23/2014

Stéphane EL MANOUNI

Planning


Cache
What and Why?
Web Caches
PHP
What is PHP?
How It Works?
PHP OPCache
PHP Accelerators
OPCache

Cache?

“

T’as bien vidé ton cache?

”

- Joe la prod digitale

Cache computing?

“

*In computing, a cache is a component that **transparently stores data** so that **future requests** for that data **can be served faster**.*

”

- Wikipedia(http://en.wikipedia.org/wiki/Cache_(computing))

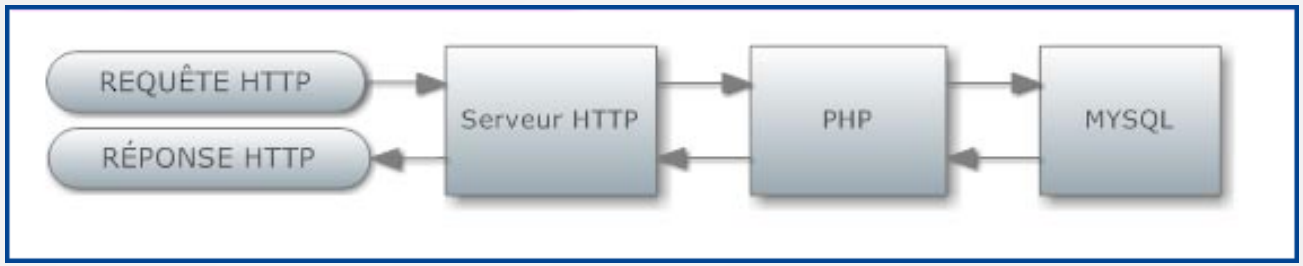
Why web caching?

For one HTTP Request, we need:

- Speed data processing
- Reduce the amount of information transmitted across Network
- Least solicit web server
- Good referencing
- Better User Experience

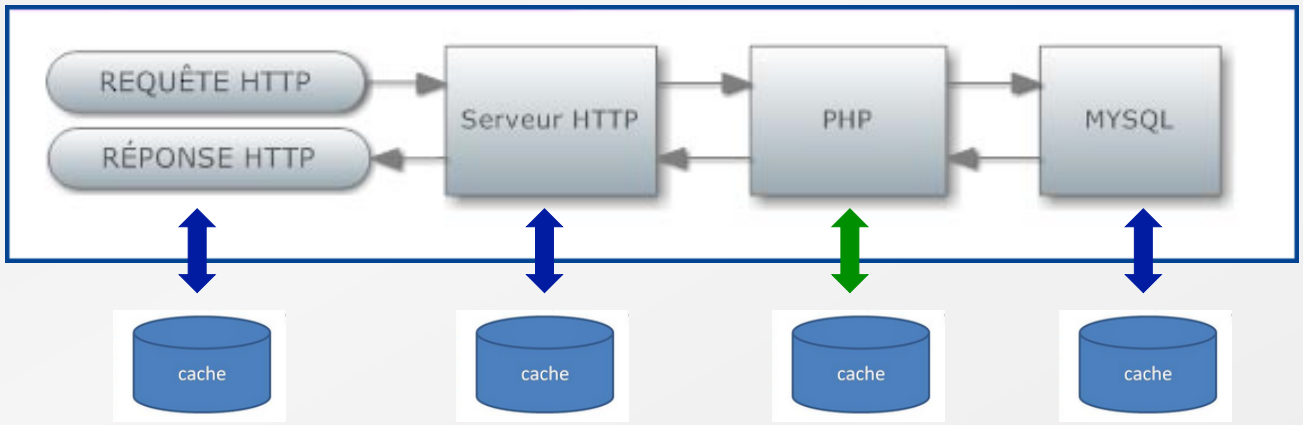
Web caches

HTTP Request on PHP website



Web caches

Request on PHP website

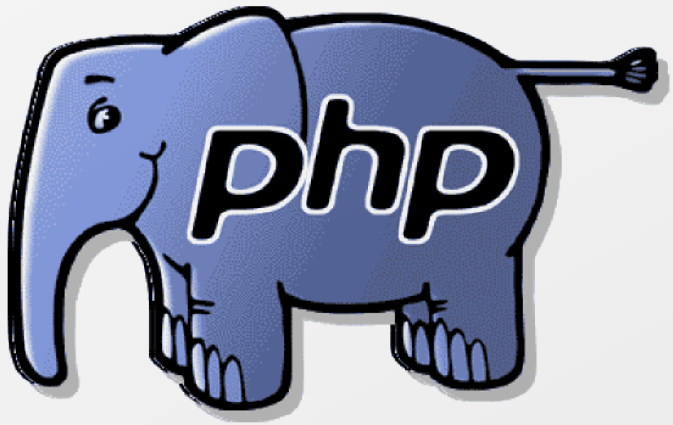


PHP?

What is PHP?

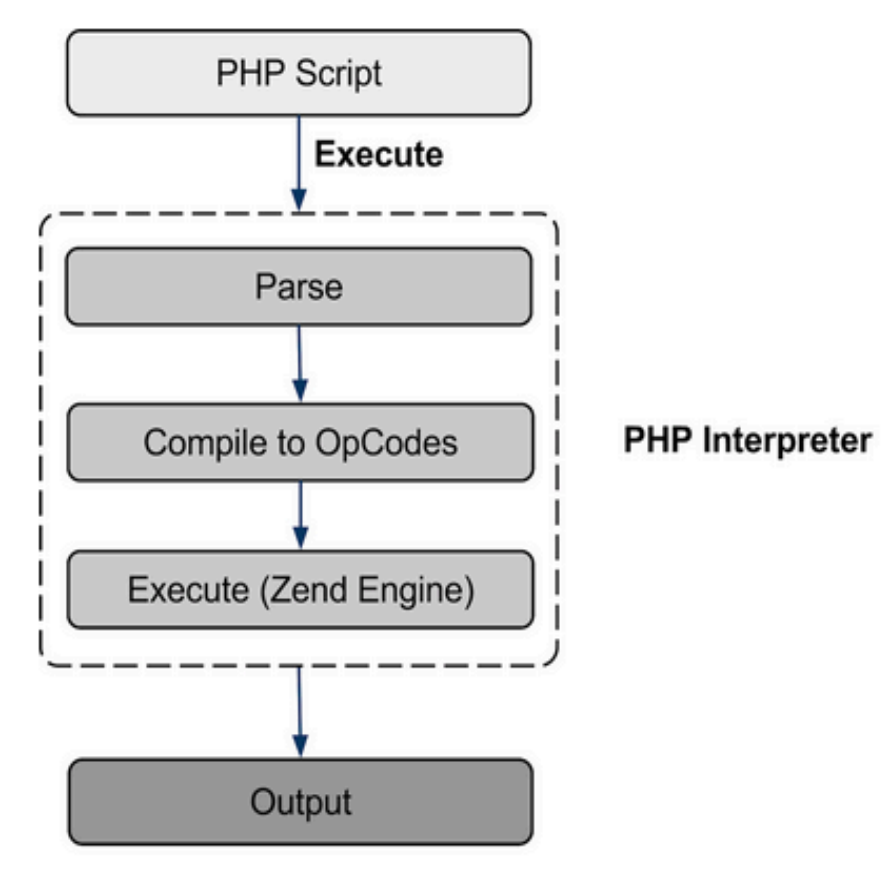
PHP: Hypertext Preprocessor

- Programming Language
- No « manual » compilation needed
- Fire and Forget
- Automatic memory management
- No strong typing
- OOP features
- Highly dynamic, extensible

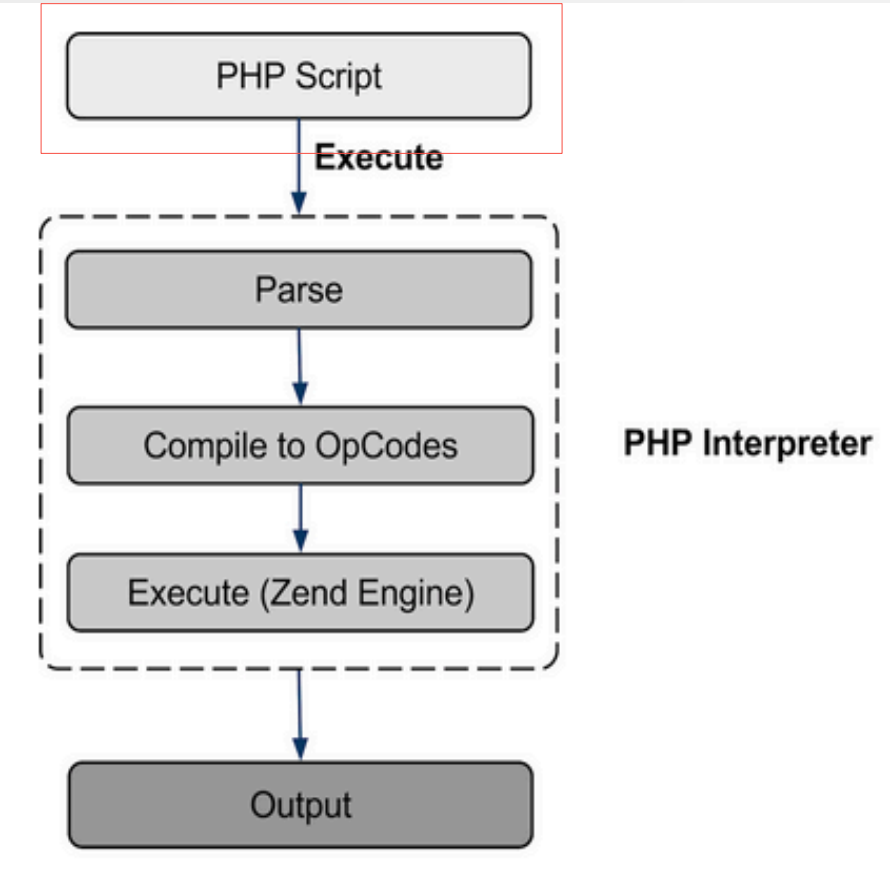


The PHP execution Life-cycle

The PHP execution Life-cycle

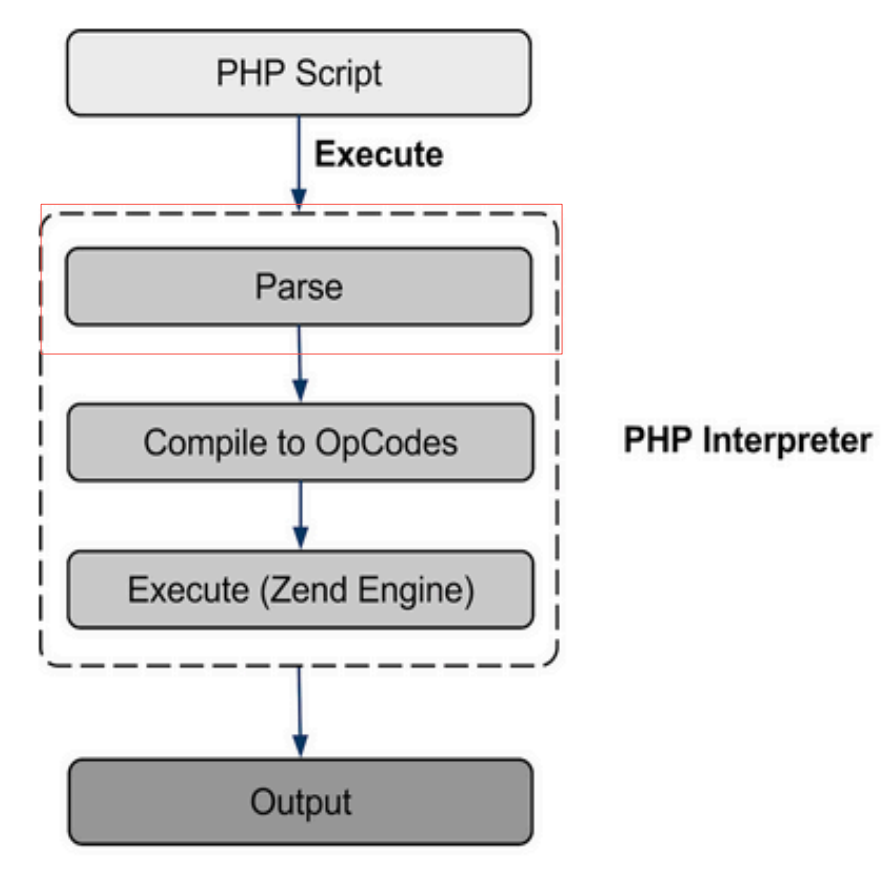


The PHP execution Life-cycle



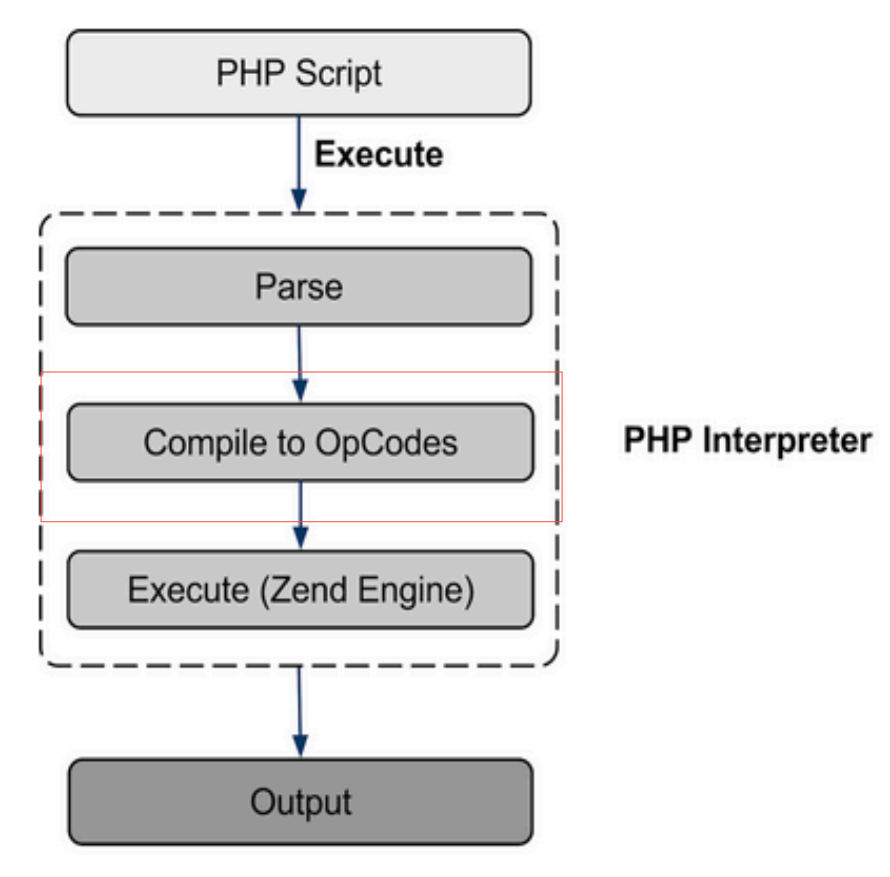

```php
<?php
class Greeting {
    public function sayHello($to)
    {
        echo "Hello $to";
    }
}

$greeter = new Greeting();
$greeter->sayHello("World");
?>
```

The PHP execution Life-cycle



Token Name	Value
T_OPEN_TAG	<?php
T_CLASS	class
T_WHITESPACE	
T_STRING	Greeting
T_WHITESPACE	
	{
T_WHITESPACE	
T_PUBLIC	public
T_WHITESPACE	
T_FUNCTION	function
T_WHITESPACE	
T_STRING	sayHello
	(
T_VARIABLE	$to
	)
T_WHITESPACE	
	{
T_WHITESPACE	
T_ECHO	echo
T_WHITESPACE	
	"
T_ENCAPSED_AND_WHITESPACE	Hello
T_VARIABLE	$to
	"

The PHP execution Life-cycle



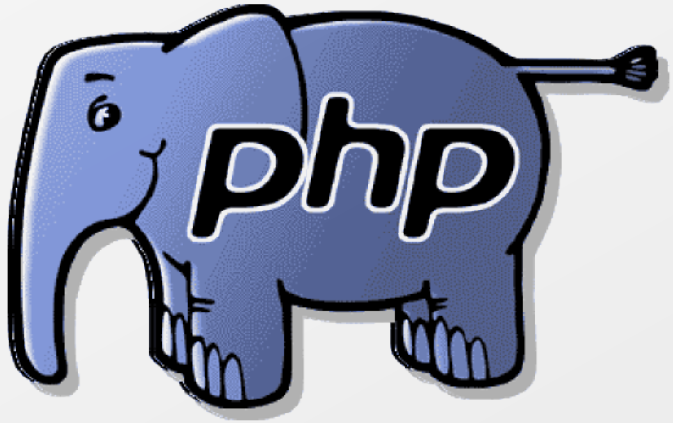

```
Class Greeting:
Function sayhello:
number of ops:  8
compiled vars:  !0 = $to

line     # *   op                    fetch          ext     return     operands
---------------------------------------------------------------------------------
   3     0  >   EXT_NOP
         1      RECV                                          !0
   5     2      EXT_STMT
         3      ADD_STRING                                    ~0      'Hello+'
         4      ADD_VAR                                       ~0      ~0, !0
         5      ECHO                                                  ~0
   6     6      EXT_STMT
         7    > RETURN                                                null
```

"In computing, an opcode (operation code) is the portion of a machine language instruction that specifies the operation to be performed." - Wikipedia

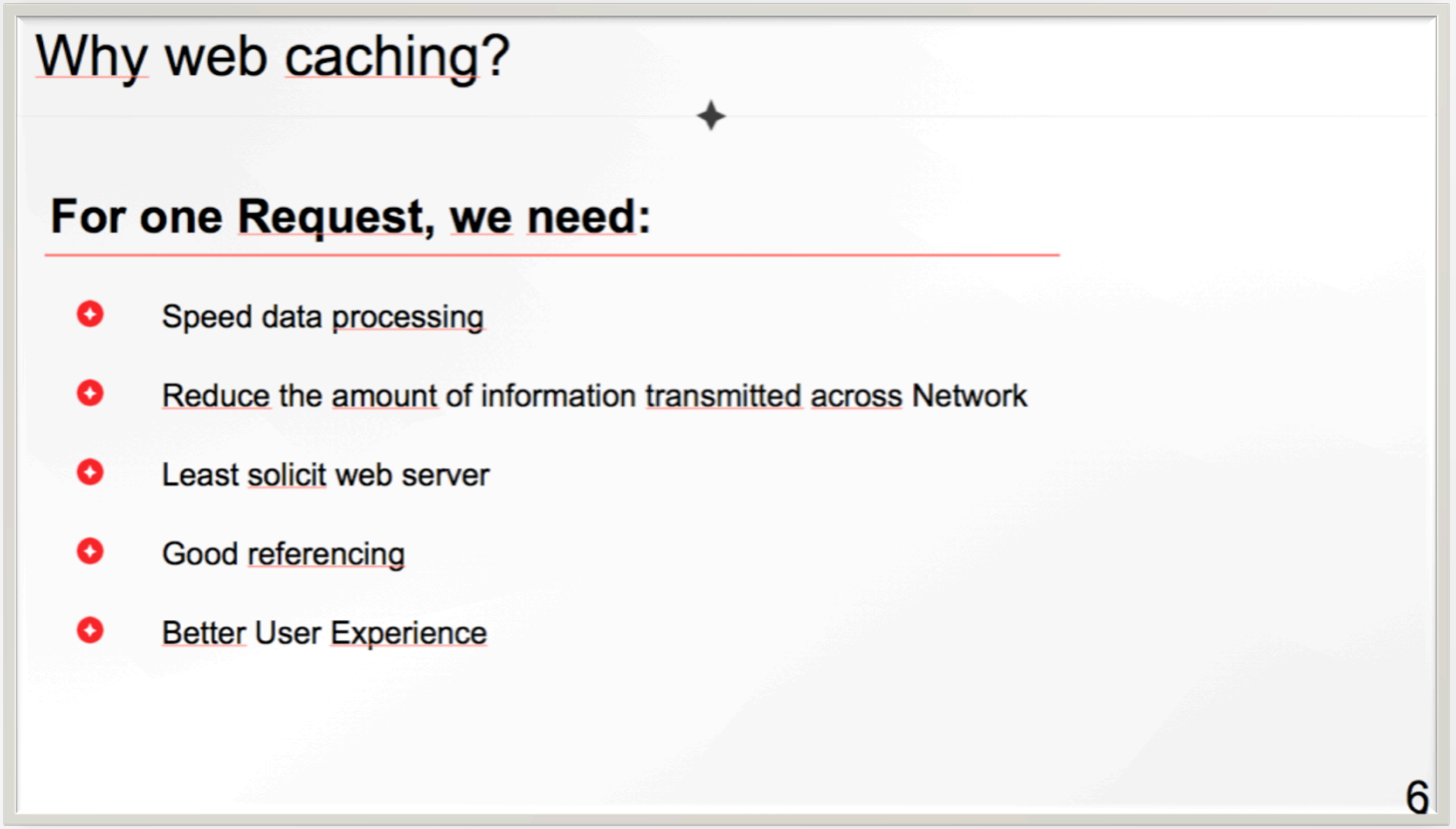
The execution Life-cycle

An elephant without memory

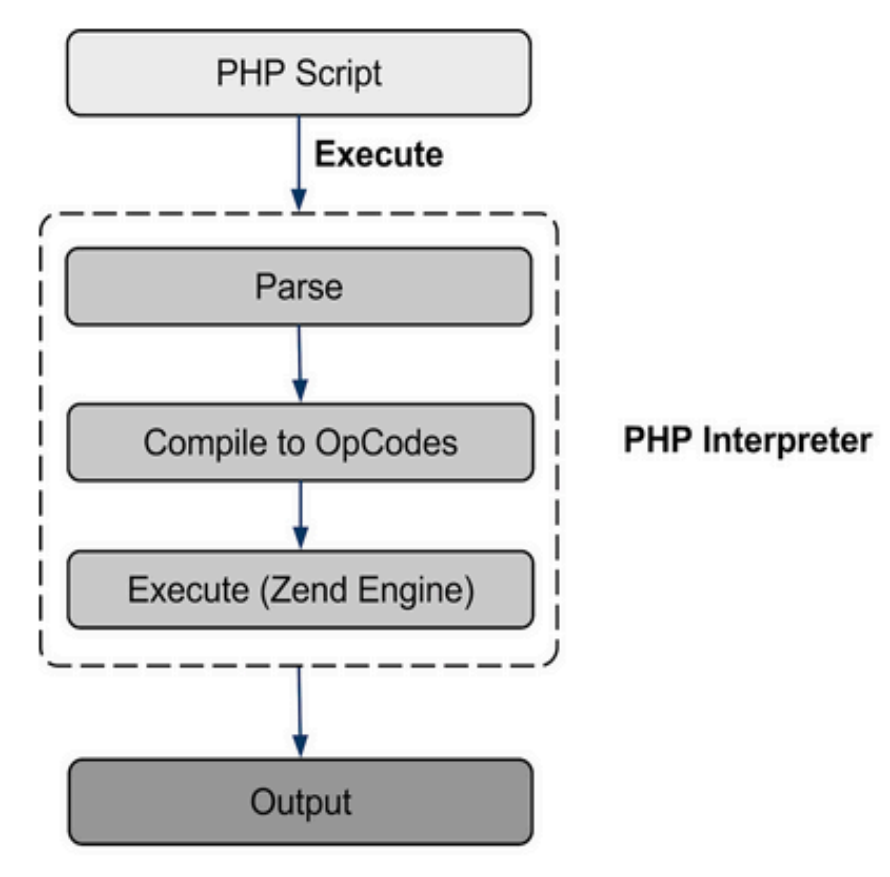
- Compile, execute, forget - Compile, execute, forget - Compile, execute, forget - Compile, execute, forget - Compile, execute, forget - Compile, execute, forget …
- By default, PHP discards all the code it just executed
- Request n+1 knows nothing about request n



The PHP execution Life-cycle



The PHP execution Life-cycle

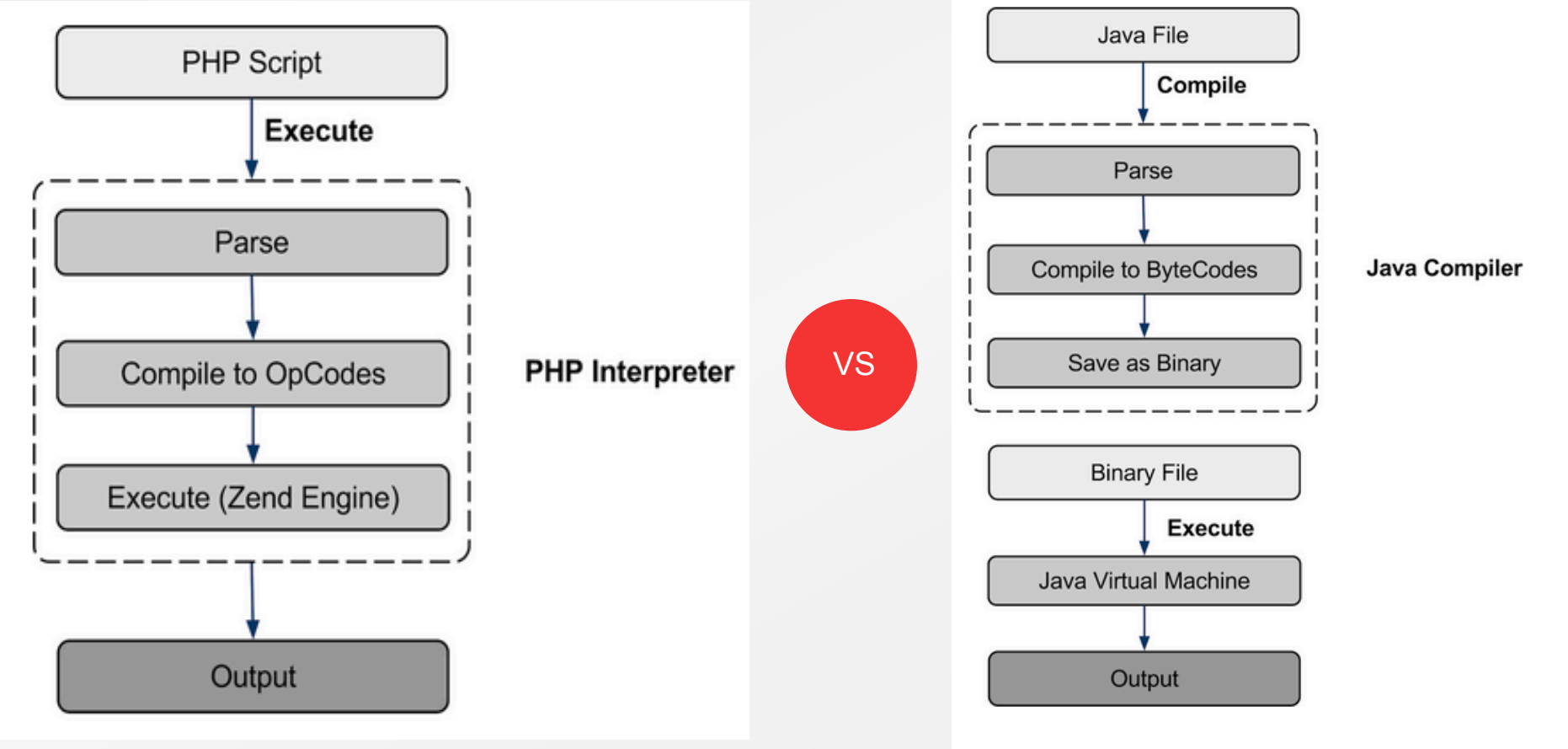


How to implement cache in this cycle ?

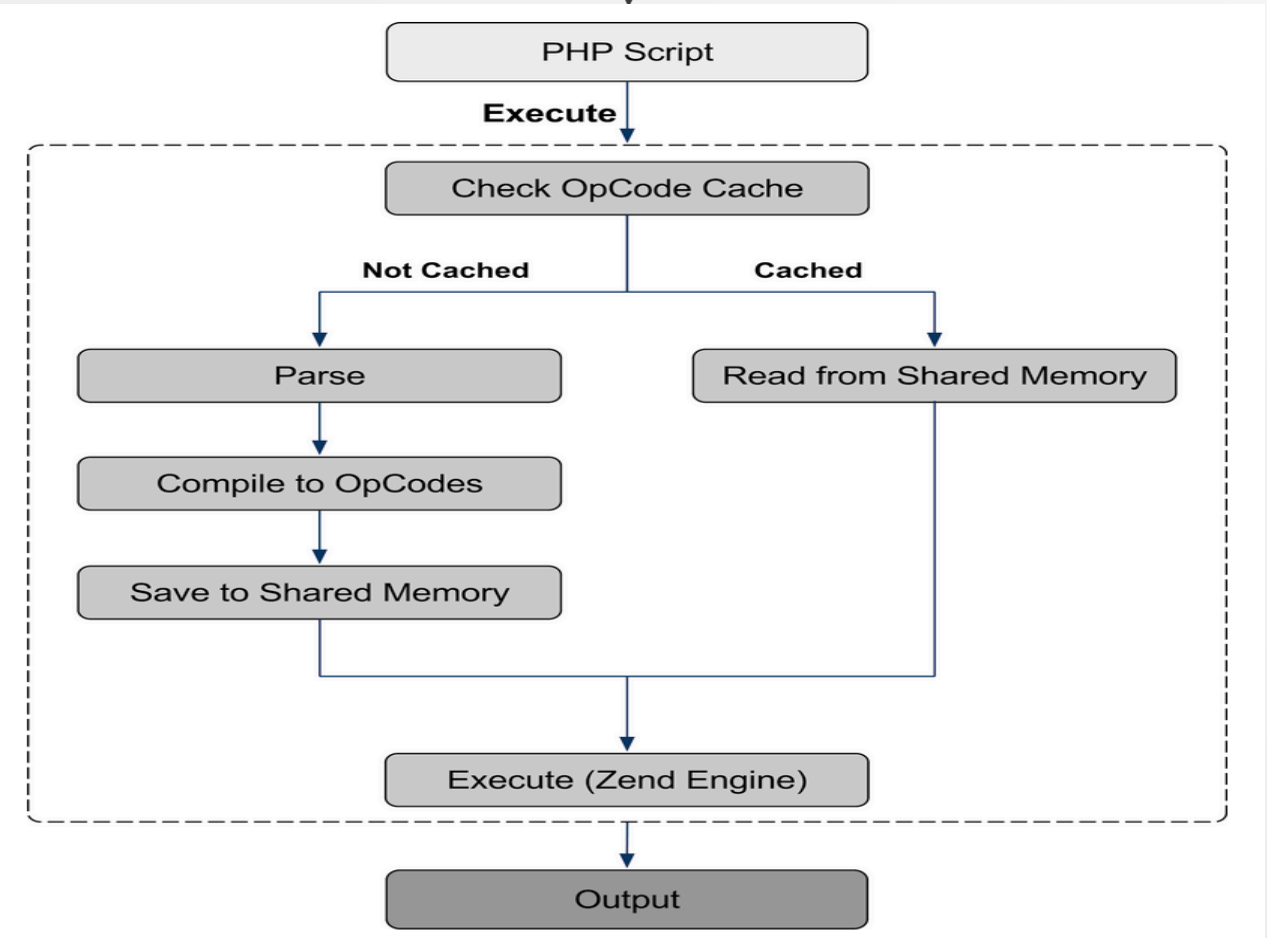
OPCache?

http://fr.slideshare.net/jpauli/yoopee-cache-op-cache-internals

The execution Life-cycle



The execution Life-cycle with OPCodes Cache



Many PHP Accelerators

PHP5.5
Zend OPcache

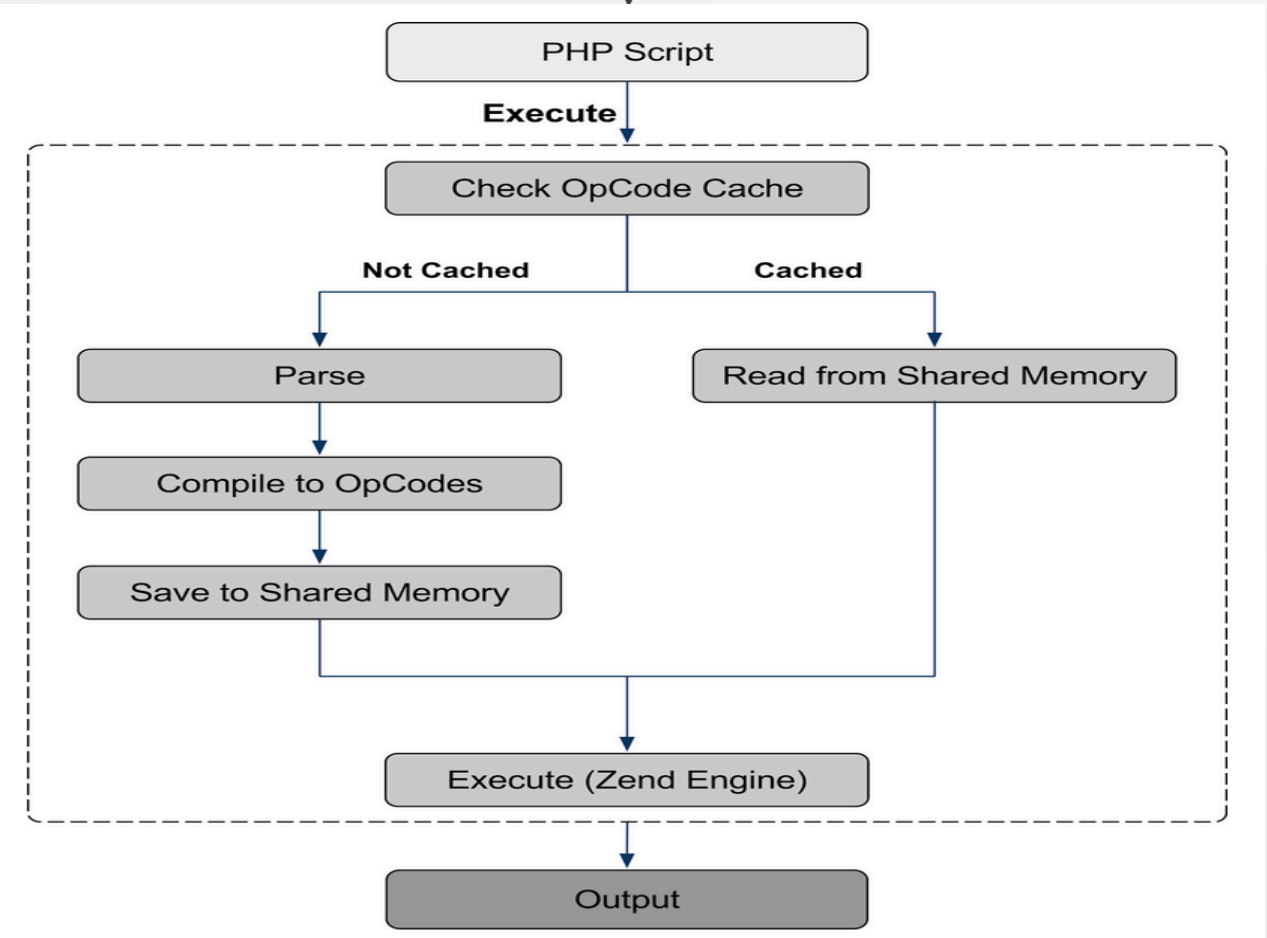
APC

NuSphere

Cache

PHP Accelerator!

The execution Life-cycle with OPCodes Cache



Many PHP Accelerators

Same Problems

- *« The more code to parse, the longer the compilation phase »*
- *« The more OPCodes generated, the longer execution »*

Many PHP Accelerators

Same Answers… well almost

- *« Let's optimize the OPCode array »*

Optimize compilation

- *« The more code to parse, the longer the compilation phase »*
  - OPCode cache compile any script just once

Optimize Execution – The OPCache Way

- *« The more OPCodes generated, the longer execution »*
- Tries to simplify/optimize OPCodes so that there are less of them and they are more efficient
  - *Works on code branch(if, try, switch…)*
  - *Optimize constant expressions*
  - *Look for dead code*
  - *Look for ways to reuse things*

Optimize Execution – The OPCache Way

```
if (false) {
    echo "foo";
} else {
   echo "bar";
}
```

Classic compilation :

```
line     # *  op                                fetch          ext  return  operands
-------------------------------------------------------------------------------
   3     0  > > JMPZ                                                      false, ->3
   4     1  >   ECHO                                                     'foo'
   5     2    > JMP                                                      ->4
   6     3  >   ECHO                                                     'bar'
   8     4  > > RETURN                                                   1
```

Optimized compilation :

```
line     # *  op                                fetch          ext  return  operands
-------------------------------------------------------------------------------
   6     0  >   ECHO                                                     'bar'
   8     1    > RETURN                                                   1
```

Optimize Execution – The OPCache Way

```
$a = 4 + "33";

echo $a;
```

Classic compilation :

```
compiled vars:  !0 = $a
line     # *  op                           fetch          ext  return  operands
---------------------------------------------------------------------------------
   3     0  >   ADD                                            ~0      4, '33'
         1      ASSIGN                                                 !0, ~0
   5     2      ECHO                                                   !0
  19     3    > RETURN                                                 1
```

Optimized compilation :

```
compiled vars:  !0 = $a
line     # *  op                           fetch          ext  return  operands
---------------------------------------------------------------------------------
   3     0  >   ASSIGN                                                 !0, 37
   5     1      ECHO                                                   !0
  19     2    > RETURN                                                 1
```

Optimize Execution – The OPCache Way

```
$i = "foo";
$i = $i + 42;
echo $i;
```

Classic compilation :

```
compiled vars:  !0 = $i
line     # *  op                           fetch          ext  return  operands
---------------------------------------------------------------------------------
   3     0  >   ASSIGN                                                 !0, 'foo'
   5     1      ADD                                            ~1      !0, 42
         2      ASSIGN                                                 !0, ~1
   7     3      ECHO                                                   !0
  22     4    > RETURN                                                 1
```

Optimized compilation :

```
compiled vars:  !0 = $i
line     # *  op                           fetch          ext  return  operands
---------------------------------------------------------------------------------
   3     0  >   ASSIGN                                                 !0, 'foo'
   5     1      ASSIGN_ADD                                     0       !0, 42
   7     2      ECHO                                                   !0
  22     3    > RETURN                                                 1
```

OPCache API – Extension in PHP >= 5.5.0

OPcache

- Introduction
- Installing/Configuring
  - Requirements
  - Installation
  - Runtime Configuration
  - Resource Types

- OPcache Functions
  - opcache_compile_file — Compiles and caches a PHP script without executing it
  - opcache_get_configuration — Get configuration information about the cache
  - opcache_get_status — Get status information about the cache
  - opcache_invalidate — Invalidates a cached script
  - opcache_reset — Resets the contents of the opcode cache

PHP 5.7

The PHP core guys have refactored the Zend Engine (which drives PHP).

```
  *Some benchmarks we ran so far:*

Wordpress 3.6 — 20.0% gain (253 vs 211 req/sec)

Drupal 6.1 — 11.7% gain (1770 vs 1585 req/sec

Qdig — 15.3% gain (555 vs 482 req/sec)

ZF test app — 30.5% gain (217 vs 166 req/sec)
```

http://news.php.net/php.internals/73888