PHP Memory leaks

and how to find them

Bohuslav Šimek

Today’s agenda

- What is memory leak/space leak?
- How PHP handle memory?
- How to debug memory usage in PHP?
- How to solve common problems?

What is memory leak?

…a memory leak is a particular type of unintentional memory consumption by a computer program where the program fails to release memory when that memory is no longer needed.

[source: https://en.wikipedia.org/wiki/Memory_leak]

Layman's terms

Program memory contains data that are no longer need it and developer don’t properly react to this.

- Talking about memory leaks in context of PHP can be confusing as…

PHP manage memory for us…

- Automatic memory management.
- Share-nothing architecture.
- Memory safe.

Should we care about memory usage?

Yes we should, because…

- Inefficient memory management.
- Long running processes.
- Wrong usage of FFI.
- Bugs in PHP or extensions.

Yes you should, because…

- **Inefficient memory management.**
- Long running processes.
- Not so common:
  - Wrong usage of FFI.
  - Bugs in PHP or extensions.

Did you ever encounter following fatal error?

```
Fatal error: Allowed memory size of x bytes
exhausted (tried to allocate x bytes)
```

- Common problem - 500+ question on stack overflow

Memory leak vs space leak

- Not every issue with memory implies memory leak.
- Some programs an ineffective thanks to wrong design.
- Seemingly good looking implementation can have an issue.

Some of the programs can cause “space leak“.

Space leak

- A space leak occurs when a computer program uses more memory than necessary.
- In contrast to memory leaks:
  - The memory consumed by program is released, but later than expected.

Space leak

- For example: you buy a 26-volume printed encyclopedia with intention to just read article about memory leaks.
- Border between memory leaks and space leaks is blurry.

X

- Space leaks are usually more prevalent.

How PHP handle memory

How does PHP handle memory?

- Automatic memory management:
  - reference counting and garbage collection.
- Individual memory optimization.
- Memory limitation built into language.

Reference counting

- Keep a counter of references to every object/array/string.
  - reference does not means “PHP reference” **&**
- Basic rules:
  - Add 1 when copying the reference (eg. pass to fce).
  - Subtract 1 when clearing a reference (eg. unset).
  - If counter drop to 0, remove the object from memory.

Reference counting - few facts

- Is predictable.
- Frees memory as soon as possible.
- No pause for cycle collection.

X

- Fails to address circular references.

Code

```
1  function bar() {
2    $array  = [];
3    $parent = new StdClass();
4    $child  = new StdClass();
5    $parent->child = $child;
6    $child->parent = $parent;
7  }
8
9  bar();
10 echo 'end';
```

Memory

main scope:

Code

```
1  function bar() {
2    $array  = [];
3    $parent = new StdClass();
4    $child  = new StdClass();
5    $parent->child = $child;
6    $child->parent = $parent;
7  }
8
9  bar();
10 echo 'end';
```

Memory

function bar scope:

$array, ref count: 1

Code

```
1  function bar() {
2    $array  = [];
3    $parent = new StdClass();
4    $child  = new StdClass();
5    $parent->child = $child;
6    $child->parent = $parent;
7  }
8
9  bar();
10 echo 'end';
```

Memory

function bar scope:

$array, ref count: 1

$parent, ref count: 1

Code

```php
function bar() {
  $array  = [];
  $parent = new StdClass();
  $child  = new StdClass();
  $parent->child = $child;
  $child->parent = $parent;
}

bar();
echo 'end';
```

Memory

function bar scope:

$array, ref count: 1

$parent, ref count: 1

$child, ref count: 1

Code

```php
function bar() {
  $array  = [];
  $parent = new StdClass();
  $child  = new StdClass();
  $parent->child = $child;
  $child->parent = $parent;
}

bar();
echo 'end';
```

Memory

function bar scope:

$array, ref count: 1

$parent, ref count: 1

$child, ref count: 2 (+1)

Code

```php
function bar() {
  $array  = [];
  $parent = new StdClass();
  $child  = new StdClass();
  $parent->child = $child;
  $child->parent = $parent;
}

bar();
echo 'end';
```

Memory

function bar scope:

$array, ref count: 1

$parent, ref count: 2 (+1)

$child, ref count: 2 (+1)

Code

```php
function bar() {
  $array  = [];
  $parent = new StdClass();
  $child  = new StdClass();
  $parent->child = $child;
  $child->parent = $parent;
}

bar();
echo 'end';
```

Memory

function bar scope:

$array, ref count: 0 (-1) -> removed

$parent, ref count: 2

$child, ref count: 2

Ref count must must be lower by one for any variable that won't leave function and its **not referenced by other variable**.

Code

```php
function bar() {
  $array  = [];
  $parent = new StdClass();
  $child  = new StdClass();
  $parent->child = $child;
  $child->parent = $parent;
}

bar();
echo 'end';
```

Memory

function bar scope:

$parent, ref count: 2

$child, ref count: 2

Parent and child remains in memory.

Code

```php
function bar() {
  $array  = [];
  $parent = new StdClass();
  $child  = new StdClass();
  $parent->child = $child;
  $child->parent = $parent;
}

bar();
echo 'end';
```

Memory

main scope:

function bar scope:

$parent, ref count: 2

$child, ref count: 2

Parent and child remains in memory.

Garbage collector to rescue!

- Reference counting fails to address circular references.
- Only garbage collector can solve this.
- Since 5.3.0, PHP has a garbage collector.
- Before 5.3.0 - regular memory leaks.

Garbage collector to rescue!

- Garbage collector handle circular references.

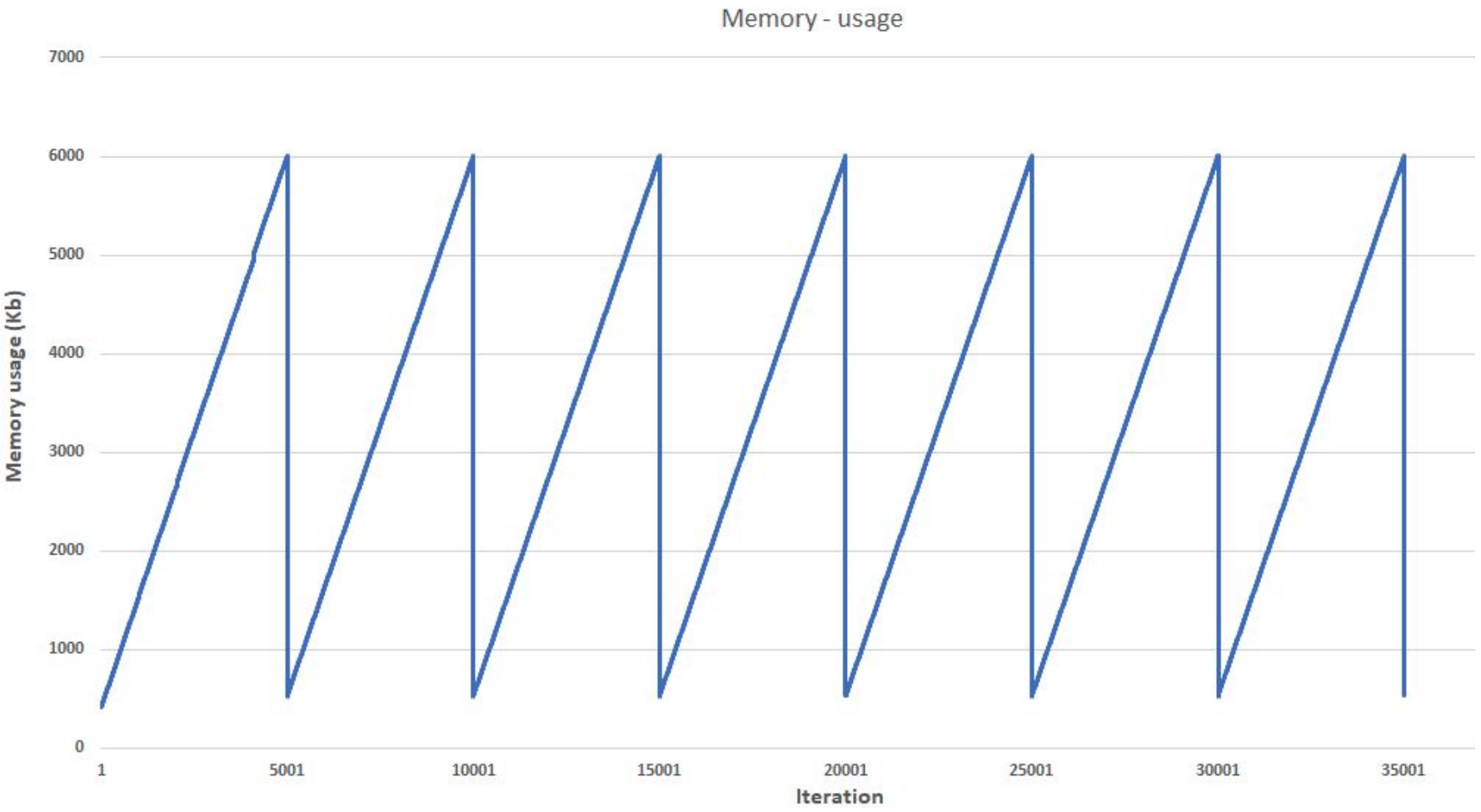
X

- Collection cycle is not done after every function run.
- Fixed threshold - 10 000 objects with cycle references.
- Sawtooth effect on memory usage.

Sawtooth effect - wait is this a memory leak?

```php
function bar() {
    $parent = new StdClass();
    $child  = new StdClass();
    $parent->child = $child;
    $child->parent = $parent;
}

for ($i = 1; $i <= 40000; $i++) {
   bar();
   echo $i.';'.(memory_get_usage()/1024)."\n";
}
```


Memory - usage
Memory usage (Kb)
7000
6000
5000
4000
3000
2000
1000
0
1
5001
10001
15001
20001
25001
30001
35001
Iteration

Memory optimizations - copy on write

A

```php
function array_test(array $test) {
   echo $test[0]."\n";
}

$array = range(1, 10000000);

array_test($array);
```

Memory usage: **512.40MB**

B

```php
function array_test(array $test) {
   $test[] = 10000001;
   echo $test[0]."\n";
}

$array = range(1, 10000000);

array_test($array);
```

Memory usage: **1024.41MB**

Whats happens - copy on write

```
1  function array_test(array $test) {
2      $test[] = 10000001;  // duplicate the array
3      echo $test[0]."\n";
4  }
5
6  $array = range(1, 10000000);
7
8  array_test($array);
```

Whats happens?

Pass array by value into function

$\neq$

Immediately duplication of variable

Copy on write

- What about writing into passed array?
- You can use references, but it's better to use objects.
- It has also other advantages:
  - More expressiveness.
  - Encapsulation of business.

How to debug memory usage?

Tools

- Builtin functions,
- debuggers (Xdebug) and
- various single purpose extensions (php-meminfo).

Builtin functions

- Always available.
- A “**`var_dump`**” debug strategy - trial and error.
- Three functions:
  - **`memory_get_usage`**
  - **`memory_get_peak_usage`**
  - **`memory_reset_peak_usage`** (from PHP 8.2)

```php
function get_data(string $file_path) : array {
    $file = fopen($file_path, 'r');
    $lines = [];
    while (($line = fgets($file)) !== false) {
        $lines[] = trim($line);
    }

    return $lines;
}

$lines = get_data('heap.txt');
foreach ($lines as $line_number => $line) {
    if ($line === 'needle') {
        echo 'found at line: '.$line_number."\n";
        break;
    }
}
```

Example

```
Fatal error: Allowed memory size of 31457280 bytes
exhausted (tried to allocate 8388616 bytes) in
/usr/src/myapp/readDataFromFile.php on line 5
```

```php
function get_data(string $file_path) : array {
    $file = fopen($file_path, 'r');
    $lines = [];
    while (($line = fgets($file)) !== false) {
        $lines[] = trim($line); // Allowed memory size of x bytes exhausted
    }

    return $lines;
}

$lines = get_data('heap.txt');
foreach ($lines as $line_number => $line) {
    if ($line === 'needle') {
        echo 'found at line: '.$line_number."\n";
        break;
    }
}
```

```php
function get_data(string $file_path) : array {
    echo sprintf("Fce enter: %dKB\n", memory_get_usage()/1024);
    $file = fopen($file_path, 'r');
    $lines = [];
    while (($line = fgets($file)) !== false) {
        echo sprintf("Read line: %dKB\n", memory_get_usage()/1024);
        $lines[] = trim($line);
    }

    return $lines;
}

$lines = get_data('heap.txt');
foreach ($lines as $line_number => $line) {
    if ($line === 'needle') {
        echo 'found at line: '.$line_number."\n";
        break;
    }
}
```

Example

```
php readDataFromFileDebug.php
```

```
Fce enter: 385KB
Read line: 394KB
... // abbreviated for sake of clarity
Read line: 937KB
... // abbreviated for sake of clarity
Read line: 1847KB

Fatal error: Allowed memory size of 2097152 bytes exhausted (tried to
allocate 4096 bytes) in readDataFromFile.php on line 8
```

Example - temporary disable memory limit

```
php -d memory_limit=-1 readDataFromFileDebug.php
```

```
Fce enter: 385KB
Read line: 394KB
Read line: 394KB
... // abbreviated for sake of clarity
Read line: 260957KB
Read line: 260957KB
Read line: 260958KB
Read line: 260958KB
found at line: 497096
```

```php
function get_data(string $file_path) : array {
    $file = fopen($file_path, 'r');
    $lines = [];
    while (($line = fgets($file)) !== false) {
        $lines[] = trim($line);
    }

    return $lines;
}

$lines = get_data('heap.txt');
foreach ($lines as $line_number => $line) {
    if ($line === 'needle') {
        echo 'found at line: '.$line_number."\n";
        break;
    }
}
```

(one of possible)

Solution

Solution, eg.: by using generators

```php
function get_data(string $file_path) : iterable {
    $file = fopen($file_path, 'r');
    while (($line = fgets($file)) !== false) {
        yield trim($line);
    }
}

$lines = get_data('heap.txt');
foreach ($lines as $line_number => $line) {
    if ($line === 'needle') {
        echo 'found at line: '.$line_number."\n";
        break;
    }
}
```

Debuggers - xdebug

- Profiling function contains also information about memory.
- If you have xdebug already installed, you can turn it by environmental variables:

```
XDEBUG_MODE=profile
XDEBUG_CONFIG=output_dir=/usr/src/myapp/tmp/
```

Debuggers - Xdebug

- Execute problematic code.
- Generate profile snapshot and open it in PHPStorm.
- Works on function level.
- Memory usage is aggregate per method.

Xdebug

fullimplemenation.out.1 | solution.out.1

Server: ... Time: ms Refresh

Execution Statistics | Call Tree

Callable	Time	Own Time	Memory (B)	Own Memory (B)	Calls
/usr/src/myapp/readDataFromFile.php	269,905 **100.0%**	65,407 24.2%	266,817,848	32	1 0.0%
get_data	204,610 **75.8%**	154,819 **57.3%**	400,761,632	0	1 0.0%
php::fgets	32,495 12.0%	32,495 12.0%	201,056,392	201,056,392	1,129,554 50.0%
php::trim	17,290 6.4%	17,290 6.4%	199,704,656	199,704,656	1,129,553 50.0%
php::fopen	4 0.0%	4 0.0%	584	584	1 0.0%

Callees | Callers

Callable	Time	Calls
<All scripts>	269,905 **100.0%**	2,259,110 **100.0%**
/usr/src/myapp/readDataFromFile.php	269,905 **100.0%**	1 0.0%

php-meminfo

- https://github.com/BitOne/php-meminfo
- Native PHP extension, PHP 7-8
- Two components:
  - extension itself,
  - analyzers written in PHP.

php-meminfo - how to use it

- Extension provide just one method (`meminfo_dump`).
- Call the method at end of request.

```
meminfo_dump(fopen('memory.json', 'w'));
```

- Execute the code.
- Analyze report in provided PHP tool (analyzer)

```php
function get_data(string $file_path) : array {
    $file = fopen($file_path, 'r');
    $lines = [];
    while (($line = fgets($file)) !== false) {
        $lines[] = trim($line);
    }

    return $lines;
}

$lines = get_data('heap.txt');
foreach ($lines as $line_number => $line) {
    if ($line === 'needle') {
        echo 'found at line: '.$line_number."\n";
        break;
    }
}

meminfo_dump(fopen('memory.json', 'w'));
```

php-meminfo

```
./bin/analyzer summary memory.json

+--------+-----------------+-----------------------------+
| Type   | Instances Count | Cumulated Self Size (bytes) |
+--------+-----------------+-----------------------------+
| string | 1129594         | 171884022                   |
| array  | 10              | 720                         |
| int    | 4               | 64                          |
| float  | 1               | 16                          |
+--------+-----------------+-----------------------------+
```

php-meminfo

```
./bin/analyzer top-children memory.json

+-----+----------------+----------+
| Num | Item ids       | Children |
+-----+----------------+----------+
| 1   | 0x7f2138a14070 | 1129553  |
| 2   | 0x7f2138a59300 | 26       |
| 3   | 0x7f2138a592c0 | 17       |
| 4   | 0x7f2138a59280 | 1        |
| 5   | 0x7f2138a76900 | 1        |
+-----+----------------+----------+
```

php-meminfo

```
./bin/analyzer ref-path  0x7f2138a14070 memory.json

Found 1 paths
Path to 0x7f2138a14070
(<GLOBAL>)$lines["<self>"]
```

More complex example

```php
use \Kambo\MemoryLeaks\{UserId,UserService};

$userService = new UserService();

for ($i = 1; $i <= 1000000; $i++) {
    $result = $userService->getUser(new UserId($i));
    echo $result."\n";
}
```

```
Annabelle Emard
Grady Abshire
Aliza Little

Fatal error: Allowed memory size of 3145728 bytes exhausted (tried to allocate
4096 bytes) in UserService.php on line 31
```

```php
use \Kambo\MemoryLeaks\{UserId,UserService};

$userService = new UserService();

for ($i = 1; $i <= 1000000; $i++) {
    $result = $userService->getUser(new UserId($i));
    echo $result."\n";
}

meminfo_dump(fopen('complex-dump.json', 'w'));
```

php-meminfo

```
./bin/analyzer summary complex-dump.json
```

Type	Instances Count	Cumulated Self Size (bytes)
string	3000044	74046833
Kambo\MemoryLeaks\User	1000000	72000000
array	25	1800
int	6	96
bool	2	32
Kambo\MemoryLeaks\UserService	1	72
SQLite3	1	72
Kambo\MemoryLeaks\ArrayBasedCache	1	72
Composer\Autoload\ClassLoader	1	72
float	1	16
null	1	16

```php
use \Kambo\MemoryLeaks\{UserId,UserService};

$userService = new UserService();

for ($i = 1; $i <= 1000000; $i++) {
    $result = $userService->getUser(new UserId($i));
    echo $result."\n";
}

```

```php
namespace Kambo\MemoryLeaks;

class User
{
   private string $userName;
   private string $name;
   private string $surname;

   public function __construct($userName, $name, $surname) {
       $this->userName = $userName;
       $this->name     = $name;
       $this->surname  = $surname;
   }

   public static function fromArray(array $data) : self {
       return new self($data['username'], $data['name'], $data['surname']);
   }

   public function __toString() : string{
       return $this->name. ' ' . $this->surname;
   }
}
```

php-meminfo

```
./bin/analyzer top-children complex-dump.json

+-----+----------------+----------+
| Num | Item ids       | Children |
+-----+----------------+----------+
| 1   | 0x7ff209458e28 | 1000000  |
| 2   | 0x7ff209459300 | 26       |
| 3   | 0x7ff2094592c0 | 17       |
| 4   | 0x7ff20947c380 | 11       |
| 5   | 0x7ff209456780 | 3        |
+-----+----------------+----------+
```

php-meminfo

```
./bin/analyzer ref-path 0x7ff209458e28 complex-dump.json

Found 1 paths
Path to 0x7ff209475cd0
(<GLOBAL>)$userService
->cache
  ->cache
```

```
./bin/analyzer -v ref-path 0x7ff209458e28 complex-dump.json


Found 1 paths                                        +--------------------------------+
Path from 0x7ff209475cd0                             |                                |
+-------------------------+                          |                                |
| Id: 0x7ff209458e28      |                          |                         +------------------------------------------+
| Type: array             |                          |                         | Id: 0x7ff209458e00                       |
| Size: 72 B              |                          |                         | Type: object                             |
| Is root: No             |                          |                         | Class: Kambo\MemoryLeaks\ArrayBasedCache |
| Children count: 1000000 |                          |                         | Object Handle: 4                         |
+-------------------------+                          |                         | Size: 72 B                               |
        ^                                            |                         | Is root: No                              |
        |                                            |                         | Children count: 1                        |
        cache                                        |                         +------------------------------------------+
        |                                            |                                ^
        |                                            |                                |
        |                                            |                                cache
        |                                            |                                |
        |                                            |                                |
        |                                            |                         +--------------------------------------+
        |                                            |                         | Id: 0x7ff209475cd0                   |
        |                                            |                         | Type: object                         |
        |                                            |                         | Class: Kambo\MemoryLeaks\UserService |
        |                                            |                         | Object Handle: 3                     |
        |                                            |                         | Size: 72 B                           |
        |                                            |                         | Is root: Yes                         |
        |                                            |                         | Execution Frame: <GLOBAL>            |
        |                                            |                         | Symbol Name: userService             |
        |                                            |                         | Children count: 2                    |
        |                                            |                         +--------------------------------------+
        +--------------------------------------------+
```

```php
class UserService
{
    private Cache $cache;

    public function __construct() {
        $this->cache = new ArrayBasedCache();
    }

    public function getUser(UserId $userId) : ?User {
        if ($this->cache->has($userId)) {
            return $this->cache->get($userId);
        }

        // Get $userData from database

        $user = User::fromArray($userData);

        $this->cache->set($userId, $user);

        return $user;
    }
}
```

Multiple solutions

- Possible solutions:
  - Limit cache size
  - TTL
  - WeakMap (from PHP 8.0)

WeakMap to rescue!

- A `WeakMap` is map that accepts objects as keys.
- An object in a key of WeakMap does not contribute toward the object's reference count.
- Offers a partial solution, item will be in cache as long as we have an instance of particular `UserId`.

WeakMap to rescue!

```php
use \Kambo\MemoryLeaks\{UserId,User};

$weakMap = new WeakMap();

$userId = new UserId(1);

$weakMap[$userId] = new User('foo', 'bar', 'baz');

var_dump(count($weakMap)); // int(1)
unset($userId);
var_dump(count($weakMap)); // int(0)
```

```php
namespace Kambo\MemoryLeaks;


class WeakMapBasedCache implements Cache
{
   private \WeakMap $cache;

   public function __construct() {
       $this->cache = new \WeakMap();
   }

   public function set(object $key, mixed $data) : void {
       $this->cache[$key] = $data;
   }

   public function get(object $key) : mixed {
       if ($this->has($key) === false) {
           throw new \LogicException("Value ".$key. " does not exists.");
       }

       return $this->cache[$key];
   }
}
```

Overview

Be aware of…

- Monitor long running scripts.
- Don’t disable memory limit.
- Think about memory usage from beginning => proper architecture from start.
- Generators are not mandatory, but they can help.

Be aware of…

- Objects holding large sets of data, eg.:
  - cache,
  - identity maps (ORM).
- Generally - think outside box (eg.: logged SQLs 😂).

Be aware of…

- Large arrays:
  - Can be unpredictably duplicated.
  - By default they consume more memory then objects.
  - Best way is to wrap them into objects and provide convenient methods.

Be aware of…

- IO handling
  - Handle big files/DB/external data in chunks.
  - Size of development and production data can differ.
  - Use the best method for data parsing. (eg.: pull parsers for large XMLs)

A new challenges!

- FFI,
- Async PHP frameworks and
- alternative PHP process managers.

Questions?

Slides: http://bohuslav.simek.si/ipc-memory-leaks/

Source code: https://github.com/kambo-1st/ipc-memory-leaks-talk

Bonus slides

FFI - Memory leaks

- Valgrind can be used for leaks detection.
- This is same as in other C/C++ program.

```
valgrind --leak-check=full php test.php
```