DUŠAN KASAN

FORGET ABOUT LOOPS

github.com/DusanKasan

@DusanKasan

me@dusankasan.com

What it'll be about

- The problem
- Functional programming
- Collection pipelines
- Collection pipelines in PHP

What's the problem?

```
$output .= "function {$properties['unifunc']} (Smarty_Internal_Template \$_smarty_tpl) {\n";
// include code for plugins
if (!$cache) {
    if (!empty($_template->compiled->required_plugins[ 'compiled' ])) {
        foreach ($_template->compiled->required_plugins[ 'compiled' ] as $tmp) {
            foreach ($tmp as $data) {
                $file = addslashes($data[ 'file' ]);
                if (is_array($data[ 'function' ])) {
                    $output .= "if (!is_callable(array('{$data['function'][0]}','{$data['function'][1]}'))) require_once '{$file}';\n";
                } else {
                    $output .= "if (!is_callable('{$data['function']}')) require_once '{$file}';\n";
                }
            }
        }
    }
    if ($_template->caching && !empty($_template->compiled->required_plugins[ 'nocache' ])) {
        $_template->compiled->has_nocache_code = TRUE;
        $output .= "echo '/*%%SmartyNocache:{$_template->compiled->nocache_hash}%%*/<?php \$_smarty = \$_smarty_tpl->smarty; ";
        foreach ($_template->compiled->required_plugins[ 'nocache' ] as $tmp) {
            foreach ($tmp as $data) {
                $file = addslashes($data[ 'file' ]);
                if (is_array($data[ 'function' ])) {
                    $output .= addslashes("if (!is_callable(array('{$data['function'][0]}','{$data['function'][1]}'))) require_once '{$file
                } else {
                    $output .= addslashes("if (!is_callable('{$data['function']}')) require_once '{$file}';\n");
                }
            }
        }
        $output .= "?>/*/%%SmartyNocache:{$_template->compiled->nocache_hash}%%*/';\n";
    }
}
```

smarty_internal_runtime_codeframe.php

Let's boil it down

```
[
    {
        "id": 1,
        "username": "john",
        "verified": true,
        "posts": [
            {
                "body": "Hi I'm John.",
                "likes": [2, 3, 5]
            },
            {
                "body": "I hate this site.",
                "likes": []
            }
        ]
    },
    {
        "id": 2,
        "username": "tom",
        "verified": false,
        "posts": []
    },
    {
        "id": 3,
        "username": "jane",
        "verified": true,
        "posts": [
            {
                "body": "Hi I'm Jane.",
                "likes": [4, 5]
            }
        ]
    }
]
```

=>

```
{
    "1": [
        {
            "body": "Hi I'm John.",
            "likes": [2, 3, 5]
        }
    ],
    "3": [
        {
            "body": "Hi I'm jane.",
            "likes": [4, 5]
        }
    ]
}
```

Let's boil it down

```
$likedPosts = [];

foreach ($users as $user) {
    if ($user['verified']) {
        $userId = $user['id'];

        foreach ($user['posts'] as $post) {
            if (count($post['likes'])) {
                $likedPosts[$userId][] = $post;
            }
        }
    }
}
```

4 levels of nesting

Harder to follow / cognitive load

Too much memorization

Refactoring candidate

Refactoring – extract more functions

```php
$likedPosts = [];

foreach ($users as $user) {
    if ($user['verified']) {
        $userId = $user['id'];

        foreach ($user['posts'] as $post) {
            if (count($post['likes'])) {
                $likedPosts[$userId][] = $post;
            }
        }
    }
}
```

```php
private function getLikedPosts(array $posts)
{
    $likedPosts = [];
    foreach ($posts as $post) {
        if (count($post['likes'])) {
            $likedPosts[] = $post;
        }
    }

    return $likedPosts;
}
```

Refactoring – extract more functions

```
$likedPosts = [];


foreach ($users as $user) {
    $likedPosts = $this->getLikedPosts($user['posts']);

    if ($user['verified'] && count($likedPosts)) {
        $likedPosts[$user['id']] = $likedPosts;
    }
}
```

- Pretty good solution
- The new function name carries the meaning of the logic
- Enough for quick understanding
- For real understanding, we'll hop between functions

Refactoring – built-in array functions

```
$indexedUsers = [];
foreach ($users as $user) {
    $indexedUsers[$user['id']] = $user;
}

$verifiedUsers = array_filter(
    $indexedUsers,
    function ($user) {
        return $user['verified'];
    }
);

$likedPosts = array_map(
    function($verifiedUser) {
        return array_filter(
            $verifiedUser['posts'],
            function($post) {return count($post['likes']);}
        );
    },
    $verifiedUsers
);
```

Refactoring – built-in array functions

(ノ °□°)ノ ︵ ┻━┻

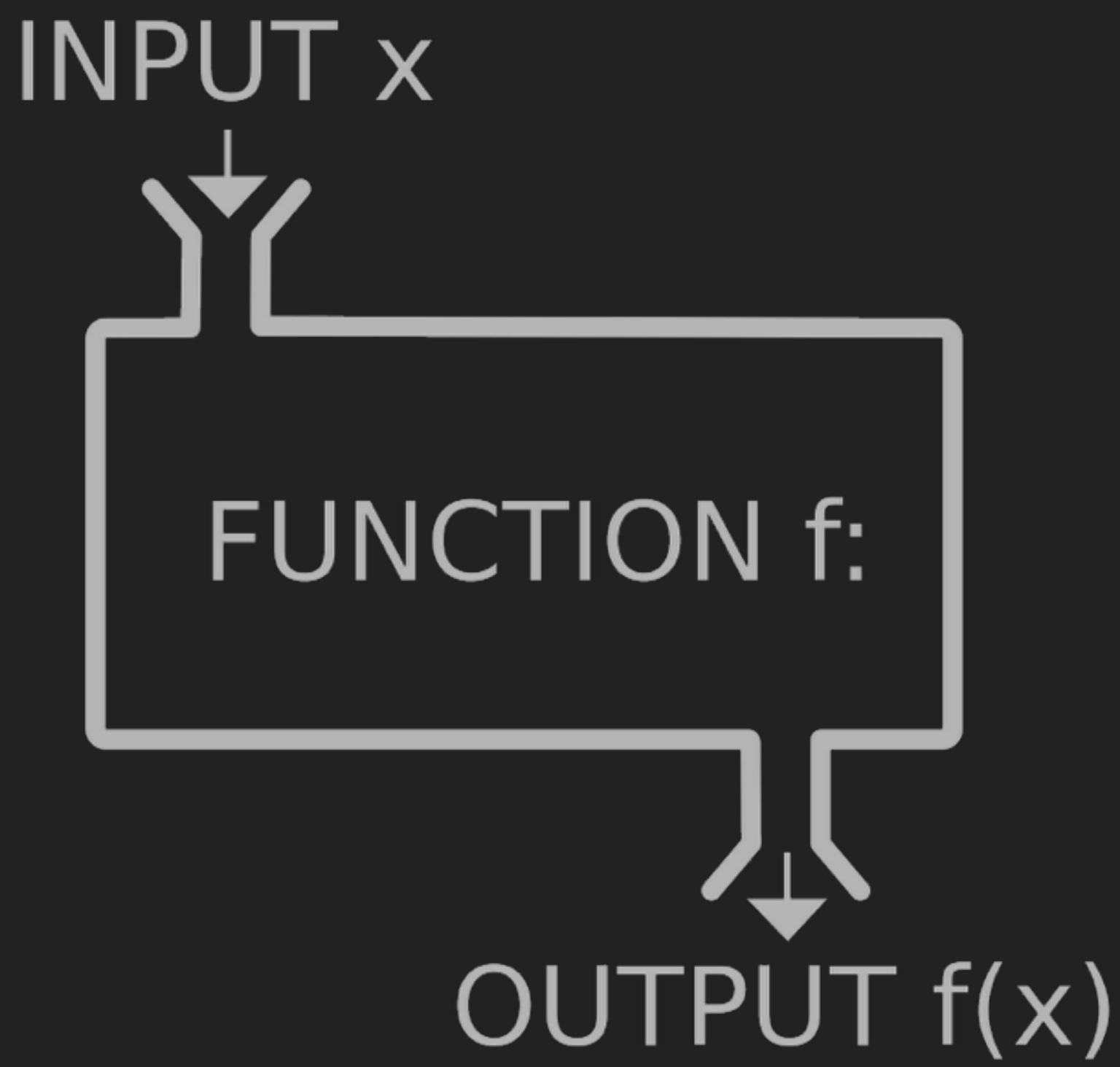
Refactoring – built-in array functions

```
$indexedUsers = [];
foreach ($users as $user) {
    $indexedUsers[$user['id']] = $user;
}

$verifiedUsers = array_filter(
    $indexedUsers,
    function ($user) {
        return $user['verified'];
    }
);

$likedPosts = array_map(
    function($verifiedUser) {
        return array_filter(
            $verifiedUser['posts'],
            function($post) {return count($post['likes']);}
        );
    },
    $verifiedUsers
);
```

Functional programming



Functional programming buzzwords

Immutability

Higher-order functions

No side effects

Parallelism

Concurrency

Functional programming buzzwords

Immutability

```
class User
{
    ...

    public function withName($name)
    {
        return new self($this->getId(), $name, ...);
    }
}
```

```
function withName(User $user, $name) {
    $result = User::from($user);
    $result->setName($name);

    return $result;
}

$tim = new User(1, 'tim', ....);
$tom = withName($tim, 'tom');

echo $tim->getName(); //'tim'
echo $tom->getName(); //'tom'
```

Functional programming buzzwords

Immutability

Higher-order functions

No side effects

Parallelism

Concurrency

Functional programming buzzwords

```
$totallyNoSideEffectsHere = function () use ($randomObscureThing) {
    return $randomObscureThing . $_REQUEST['absolutely_safe'] . DateTime::ATOM;
};

$a = $totallyNoSideEffectsHere();
```

No side effects

```
$anotherTry = function ($a, $b, $c) {
    return $a . $b . $c;
};

$a = $anotherTry(
    $randomObscureThing,
    $_REQUEST['absolutely_safe'],
    DateTime::ATOM
);
```

Functional programming buzzwords

Immutability

Higher-order functions

No side effects

Parallelism

Concurrency

Functional programming buzzwords

```php
function show(callable $c) {
    echo $c();
}

function style(callable $c) : callable {
    return function () use ($c) {
        return '**' . $c() . '**';
    };
}

$nameProvider = function () {
    return 'John';
};

show($nameProvider); //John
show(style($nameProvider)); //**John**
```

Higher-order functions

Functional programming buzzwords

Immutability

Higher-order functions

No side effects

Parallelism

Concurrency

Functional programming buzzwords

```
getOrderDetails(
    getOrderById($id),
    getPaymentByOrderId($id)
);
```

Parallelism

Concurrency

Collection Pipelines

┬──┬ ノ(゜ -゜ ノ)

Collection Pipelines

- What are collections?
  - Group of values or key, value pairs
  - Think arrays or Traversables
- Collection pipeline
  - Design pattern
  - Sequential operations on collections
  - Operations e.g.: filter, map or reduce
  - Used by both FP and OOP languages
  - Immutable, lazy, concurrent

Simple explanation

map([🌽, 🐮, 🐔], cook) => [🍿, 🍔, 🍳]

filter([🍿, 🍔, 🍳], isVegetarian) => [🍿, 🍳]

reduce([🍿, 🍳], eat) => 💩

From a tweet by @steveluscher

Simple explanation

[🌽, 🐮, 🐓]

.map(cook) // [🍿, 🍔, 🍳]

.filter(isVegetarian) // [🍿, 🍳]

.reduce(eat) // 💩

Collection pipeline operations

append, combine, concat, contains, countBy, cycle, diff, distinct, drop, dropLast, dropWhile, each, every, except, extract, filter, find, first, flatten, flip, frequencies, get, getOrDefault, groupBy, groupByKey, has, indexBy, interleave, interpose, intersect, isEmpty, isNotEmpty, keys, last, map, mapcat, only, partition, partitionBy, prepend, realize, reduce, reduceRight, reductions, reject, replace, reverse, second, shuffle, size, slice, some, sort, splitAt, splitWith, take, takeNth, takeWhile, transform, toArray, zip,

The original problem

```
[
    {
        "id": 1,
        "username": "john",
        "verified": true,
        "posts": [
            {
                "body": "Hi I'm John.",
                "likes": [2, 3, 5]
            },
            {
                "body": "I hate this site.",
                "likes": []
            }
        ]
    },
    {
        "id": 2,
        "username": "tom",
        "verified": false,
        "posts": []
    },
    {
        "id": 3,
        "username": "jane",
        "verified": true,
        "posts": [
            {
                "body": "Hi I'm Jane.",
                "likes": [4, 5]
            }
        ]
    }
]
```

=>

```
{
    "1": [
        {
            "body": "Hi I'm John.",
            "likes": [2, 3, 5]
        }
    ],
    "3": [
        {
            "body": "Hi I'm jane.",
            "likes": [4, 5]
        }
    ]
}
```

Refactoring – Collections

```php
$likedPosts = [];

foreach ($users as $user) {
    if ($user['verified']) {
        $userId = $user['id'];

        foreach ($user['posts'] as $post) {
            if (count($post['likes'])) {
                $likedPosts[$userId][] = $post;
            }
        }
    }
}
```

```php
$likedPosts = Collection::from($users)
    ->filter(function ($user) {return $user['verified'];})
    ->indexBy(function($user) {return $user['id'];})
    ->map(function($user) {
        return Collection::from($user['posts'])
            ->filter(function($post) {return count($post['likes']);});
    })
    ->reject(function($posts) {return isEmpty($posts);});
```

When to use collection pipelines

- Where you work with collections?
  - API
  - Import CSV
- Any nontrivial collection based work
- Refactoring of current code base to for comprehension

Drawbacks?

- Function call overhead
- You have to understand it :)

Collection pipelines in PHP

- CakePHP Collection (github.com/cakephp/collection)
- Laravel Collections (github.com/illuminate/support)
- PHP-Collection (github.com/emonkak/php-collection)
- Knapsack (github.com/DusanKasan/Knapsack)

PHP's shortcomings

```
$collection->map(function($item) {
    return $item->field;
});
```

```
$collection->map(function($item) {return $item->field;});
```

```
$collection->map($item ~> $item->field);
```

```
$collection->map(function($item) => $item->field);
```

RFC targeting PHP 7.1

Interested?

- Martin Fowler (martinfowler.com)
- Adam Wathan (adamwathan.me/refactoring-to-collections)
- Documentation and tests to the mentioned libraries

DUŠAN KASAN

FORGET ABOUT LOOPS