Debugging Our Way Through PHP

International PHP Conference

Derick Rethans

derick@xdebug.org — @derickr@phpc.social

https://derickrethans.nl/talks/xdebug-ipc23

https://xdebug.org

About Me

Derick Rethans

- I'm European, living in London
- ~~PHP 7.4 **Release Manager**~~
- **PHP Foundation**
- **Xdebug** & PHP's Date/Time support
- I ♥🌍 maps, I ♥🍺 beer, I ♥🥃 whisky
- mastodon: @derickr@phpc.social
- twitter: @derickr

Introducing Modes

Off

Loaded, but miniscule overhead

Introducing Modes

Develop

Every day debugging aides

Introducing Modes

Debugger

Debugging applications with an IDE

Previously: remote debugger

Introducing Modes

Tracing

Tracing the execution of a script to a file

Introducing Modes

Profiler

Profiling application performance

Introducing Modes

Coverage

Which code is run during tests

Introducing Modes

Time Traveller

Introducing Modes

Configuration Settings

```
xdebug.cli_color
xdebug.cloud_id
xdebug.collect_assignments
xdebug.collect_return
xdebug.dump.COOKIE
xdebug.dump.ENV
xdebug.dump.FILES
xdebug.dump.GET
xdebug.dump_globals
xdebug.dump_once
xdebug.dump.POST
xdebug.dump.REQUEST
xdebug.dump.SERVER
xdebug.dump.SESSION
xdebug.dump_undefined
xdebug.file_link_format
xdebug.filename_format
xdebug.force_display_errors
xdebug.force_error_reporting
xdebug.gc_stats_output_name
xdebug.halt_level
xdebug.idekey
xdebug.log
xdebug.log_level
xdebug.max_nesting_level
```

```
xdebug.max_stack_frames
xdebug.mode
xdebug.output_dir
xdebug.profiler_append
xdebug.profiler_output_name
xdebug.client_discovery_header
xdebug.client_host
xdebug.client_port
xdebug.discover_client_host
xdebug.connect_timeout_ms
xdebug.scream
xdebug.show_error_trace
xdebug.show_exception_trace
xdebug.show_local_vars
xdebug.start_upon_error
xdebug.start_with_request
xdebug.trace_format
xdebug.trace_options
xdebug.trace_output_name
xdebug.trigger_value
xdebug.var_display_max_children
xdebug.var_display_max_data
xdebug.var_display_max_depth
```

`xdebug_info()`

Like `phpinfo()`, but then for Xdebug

DEMO

`xdebug_info( string $category )`

Obtain information about Xdebug programatically

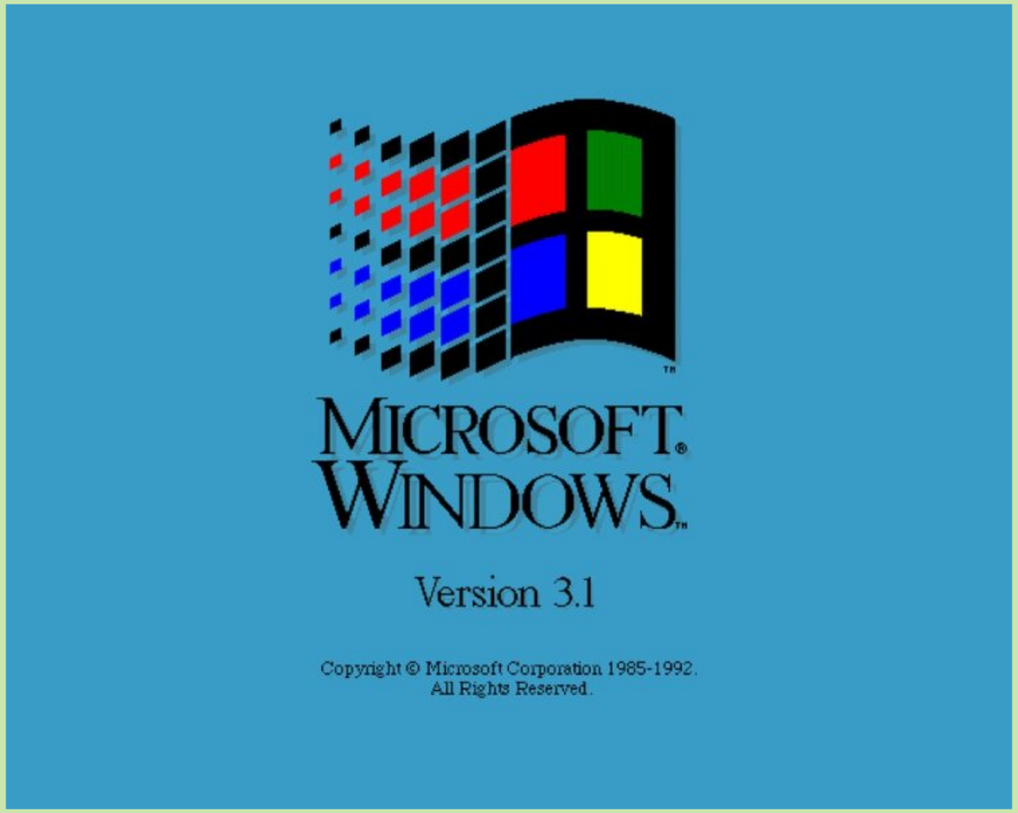
DEMO

`xdebug_notify( mixed $data )`

Send data over debugging protocol

DEMO

Co-operative Multi-Tasking



Show and Tell: `Fibers`

Everyone:

- Open `fibers.php`

VS Code:

- Click green triangle / F5 to start listening

PhpStorm

- Click: `Run`, `Start Listening for PHP Debug Connections`

Everyone:

- Open `http://localhost:8123/fibers.php` in browser

Breakpoint Resolving

Xdebug now finds the closest breakpoint line

DEMO

Breakpoint Resolving

Xdebug now finds the closest breakpoint line

DEMO

How does this work?

Breakpoint Resolving

Xdebug now finds the closest breakpoint line

DEMO

How does this work?

Let's check the log…

DEMO

Return Values?

DEMO

Breakpoint Resolving — Revised

The IDE can set breakpoints on non-existing files...

DEMO

`xdebug_connect_to_client()`

Prods Xdebug to try to make a debug connection again

DEMO

Episode #4: Short Arrow Functions

Nikita Popov

Arrow functions have the following basic form:

```
fn(parameter_list) => expr
```

Traditional Closure:

```
$arrow = function ($x) use ($y) {
    return $x + $y;
};
```

https://wiki.php.net/rfc/arrow_functions_v2

Episode #4: Short Arrow Functions

Nikita Popov

Arrow functions have the following basic form:

```
fn(parameter_list) => expr
```

Traditional Closure:

```
$arrow = function ($x) use ($y) {
    return $x + $y;
};
```

New Short Closure:

```
$arrow = fn($x) => $x + $y;
```

https://wiki.php.net/rfc/arrow_functions_v2

First Class Callable Syntax

```php
<?php
class Test {
    public function getPrivateMethod() {
        return [$this, 'privateMethod'];
        return Closure::fromCallable([$this, 'privateMethod']);
        return $this->privateMethod(...);
    }

    private function privateMethod() {
        echo __METHOD__, "\n";
    }
}

$test = new Test;
$privateMethod = $test->getPrivateMethod();
$privateMethod();
?>
```

First Class Callable Syntax

```php
<?php
class Test {
    public function getPrivateMethod() {
        return [$this, 'privateMethod'];
        return Closure::fromCallable([$this, 'privateMethod']);
        return $this->privateMethod(...);
    }

    private function privateMethod() {
        echo __METHOD__, "\n";
    }
}

$test = new Test;
$privateMethod = $test->getPrivateMethod();
$privateMethod();
?>
```

First Class Callable Syntax

```php
<?php
class Test {
    public function getPrivateMethod() {
        return [$this, 'privateMethod'];*
        return Closure::fromCallable([$this, 'privateMethod']);
        return $this->privateMethod(...); // new in PHP 8.1
    }

    private function privateMethod() {
        echo __METHOD__, "\n";
    }
}

$test = new Test;
$privateMethod = $test->getPrivateMethod();
$privateMethod();
?>
```

First Class Callable Syntax

Syntactic Sugar

```
$fn = Closure::fromCallable('strlen');
$fn = strlen(...);

$fn = Closure::fromCallable([$this, 'method']);
$fn = $this->method(...)

$fn = Closure::fromCallable([Foo::class, 'method']);
$fn = Foo::method(...);
```

Other variants

```
strlen(...);
$closure(...);
$obj->method(...);
$obj->$methodStr(...);
($obj->property)(...);
Foo::method(...);
$classStr::$methodStr(...);
self::{$complex . $expression}(...);
[$obj, 'method'](...);
[Foo::class, 'method'](...);
```

Union Types

```
<?php
declare(strict_types=1);

class Number {



    private $number;



    public function setNumber($number) {
        $this->number = $number;
    }



    public function getNumber() {
        return $this->number;
    }
}
```

Union Types

```php
<?php
declare(strict_types=1);

class Number {
    /**
     * @var int|float $number
     */
    private $number;

    /**
     * @param int|float $number
     */
    public function setNumber($number) {
        $this->number = $number;
    }

    /**
     * @return int|float
     */
    public function getNumber() {
        return $this->number;
    }
}
```

Union Types

```php
<?php
declare(strict_types=1);

class Number {



    private int|float $number;




    public function setNumber(int|float $number) {
        $this->number = $number;
    }



    public function getNumber() : int|float {
        return $this->number;
    }
}
```

Union Types — Examples

```
function microtime(bool $get_as_float = false): string|float {}

function gettimeofday(bool $return_float = false): array|float {}

function base64_decode(string $str, bool $strict = false): string|false {}

function implode(string|array $glue, array $pieces): string {}

function substr_replace(
    string|array $str,
    string|array $replace,
    $start,
    $length
) : string|array|false {}
```

Union Types — Rules

All support types can be part of a union, with the following caveats:

- **`void`** can't be part of a union type
- **`?Class`** is equivalent to **`Class|null`**
- **`false`** is an addition, as it is used in the PHP standard library
  e.g.: **`strpos() : int|false`**
- No redundant types, such as: **`bool|false`**, or: **`object|Whisky`**

Variance:

- Union types follow standard variance rules
- Order is irrelevant: **`int|string`** and **`string|int`**
- **`array|Traversable`** is the same as **`iterable`**

Pure Intersection Types

To enforce a value is both `Traversable` and `Countable`:

```
<?php
class Test {
    private ?Traversable $traversable = null;
    private ?Countable $countable = null;
    /** @var Traversable&Countable */
    private $both = null;

    public function __construct($countableIterator) {
        $this->traversable =& $this->both;
        $this->countable =& $this->both;
        $this->both = $countableIterator;
    }
}
```

Pure Intersection Types

In PHP 8.1:

```php
<?php
class Test {
    private Traversable&Countable $countableIterator;

    public function setIterator(Traversable&Countable $countableIterator): void {
        $this->countableIterator = $countableIterator;
    }

    public function getIterator(): Traversable&Countable {
        return $this->countableIterator;
    }
}
```

Pure Intersection Types

In PHP 8.1:

```php
<?php
class Test {
    private Traversable&Countable $countableIterator;

    public function setIterator(Traversable&Countable $countableIterator): void {
        $this->countableIterator = $countableIterator;
    }

    public function getIterator(): Traversable&Countable {
        return $this->countableIterator;
    }
}
```

- Only for class and interface names
- Variance rules are respected, but they are complicated

PHP 8.2: DNF Types

Union Types: **X|Y**

```
class Number {
    private int|float $number;

    public function setNumber(int|float $number): void {
        $this->number = $number;
    }

    public function getNumber(): int|float {
        return $this->number;
    }
}
```

(Pure) Intersection Types: **X&Y**

Disjunctive Normal Form Types: **(X&A)|(Y&B)**

PHP 8.2: DNF Types

Union Types: **X|Y**

(Pure) Intersection Types: **X&Y**

```
class Iteraty {
    private Traversable&Countable $countableIterator;

    public function setIterator(Traversable&Countable $countableIterator): void {
        $this->countableIterator = $countableIterator;
    }

    public function getIterator(): Traversable&Countable {
        return $this->countableIterator;
    }
}
```

Disjunctive Normal Form Types: **(X&A)|(Y&B)**

PHP 8.2: DNF Types

Union Types: **X|Y**

(Pure) Intersection Types: **X&Y**

Disjunctive Normal Form Types: **(X&A)|(Y&B)**

```
class Foreachy {
    private array|(Traversable&Countable) $foreachableData;

    public function setIterator(array|(Traversable&Countable) $foreachable): void
        $this->foreachableData = $foreachable;
    }

    public function getIterator(): array|(Traversable&Countable) {
        return $this->foreachableData;
    }
}
```

PHP 8.2: DNF Types

Union Types: **X|Y**

(Pure) Intersection Types: **X&Y**

Disjunctive Normal Form Types: **(X&A)|(Y&B)**

Sensitive Parameters

```php
<?php
function logIn($userName, #[\SensitiveParameter] $password) {
    throw new \Exception('Error');
}

logIn( 'derick', 'secret-elephpant' );
```

Without Xdebug:

```
Fatal error: Uncaught Exception: Error in Standard input code:4
Stack trace:
#0 Standard input code(7): logIn('derick', Object(SensitiveParameterValue))
#1 {main}
  thrown in Standard input code on line 4
```

Sensitive Parameters

```php
<?php
function logIn($userName, #[\SensitiveParameter] $password) {
    throw new \Exception('Error');
}

logIn( 'derick', 'secret-elephpant' );
```

With Xdebug:

```
Fatal error: Uncaught Exception: Error in Standard input code on line 4

Exception: Error in Standard input code on line 4

Call Stack:
  4.8626  399576  1. {main}() Standard input code:0
  4.8626  399576  2. logIn($userName = 'derick', $password = '[Sensitive Parameter
```

PHP 8.3: New #[Override] Attribute

```
<?php
final class MyApp\Tests\MyTest extends PHPUnit\Framework\TestCase
{
    protected $logFile;

    protected function setUp(): void
    {
        $this->logFile = fopen('/tmp/logfile', 'w');
    }


    protected function taerDown(): void
    {
        fclose($this->logFile);
        unlink('/tmp/logfile');
    }
}
?>
```

PHP 8.3: New #[Override] Attribute

```php
<?php
final class MyApp\Tests\MyTest extends PHPUnit\Framework\TestCase
{
    protected $logFile;

    protected function setUp(): void
    {
        $this->logFile = fopen('/tmp/logfile', 'w');
    }

    #[\Override]
    protected function tearDown(): void
    {
        fclose($this->logFile);
        unlink('/tmp/logfile');
    }
}
?>
```

If this attribute is added to a method, PHP validates that a method with the same name exists in a parent class or any of the implemented interfaces.

PHP 8.3: Typed Class Constants

```php
<?php
interface I {
    const TEST = "Test";
}

class Foo implements I {
    const TEST = [];
}

class Bar extends Foo {
    const TEST = null;
}
?>
```

PHP 8.3: Typed Class Constants

```php
<?php
interface I {
    const string TEST = "Test";
}

class Foo implements I {
    const TEST = [];
}

class Bar extends Foo {
    const float TEST = M_PI;
}
?>
```

PHP 8.3: Typed Class Constants

```php
<?php
interface I {
    const string TEST = "Test";
}

class Foo implements I {
    const TEST = [];
}

class Bar extends Foo {
    const float TEST = M_PI;
}
?>
```

```
Fatal error: Type of Foo:TEST must be compatible with I::TEST of type string
```

Dynamic Class Constant Fetch

Looking up members' names:

- Variables: `$$foo`
- Properties: `$foo->$bar`
- Static properties: `Foo::${$bar}`
- Methods: `$foo->{$bar}()`
- Static methods: `Foo::{$bar}()`
- Classes for static properties: `$foo::$bar`
- Classes for static methods: `$foo::bar()`
- ~~Class constants: `$foo::{$bar}`~~

Dynamic Class Constant Fetch

Looking up members' names:

- …
- Class constants: `$foo::{$bar}`

```
<?php
class Foo {
    const BAR = 'bar';
}
$bar = 'BAR';

echo Foo::{$bar}; // bar
?>
```

Also works for magic `class` constant:

```
<?php
namespace Foo;
class Bar {}

$class = 'class';
echo Bar::{$class}; // Foo\Bar
?>
```

xdebug_get_function_stack() - local variables

```php
<?php
class Elephpant
{
    function __construct(private string $title, private float $PIE) {}
}

class Error_Class
{
    public static function getBT()
    {
        $tmp = xdebug_get_function_stack( ['local_vars' => true ] );
        var_dump(array_reverse($tmp));
    }

    public static function newError($errno = false)
    {
        $elephpant = new Elephpant("Bluey", M_PI);
        $randoVar = 42;
        return self::getBT();
    }
}

$e = Error_Class::newError(42);
?>
```

Result:

xdebug_get_function_stack() - parameters as values

```php
<?php
class Elephpant
{
    function __construct(private string $title, private float $PIE) {}
}

class Error_Class
{
    public static function getBT()
    {
        $tmp = xdebug_get_function_stack( ['params_as_values' => true ] );
        var_dump(array_reverse($tmp));
    }

    public static function newError($errno = false)
    {
        $elephpant = new Elephpant("Bluey", M_PI);
        $randoVar = 42;
        return self::getBT();
    }
}

$e = Error_Class::newError(42);
?>
```

Result:

xdebug_get_function_stack() - exceptions

```php
<?php
function exceptionHandler($exception)
{
    print_r(array_reverse(xdebug_get_function_stack()));
}

class Elephpant
{
    function __construct(private string $title, private float $PIE) {}
}

class Error_Class
{
    public static function newError($errno = false)
    {
        $elephpant = new Elephpant("Bluey", M_PI);
        $randoVar = 42;
        throw new Exception("My Exception");
    }
}

set_exception_handler('exceptionHandler');

$e = Error_Class::newError(42);
```

```
Array
```

xdebug_get_function_stack() - exceptions

```php
<?php
function exceptionHandler($exception)
{
    print_r(array_reverse(xdebug_get_function_stack( ['from_exception' => $excepti
}

class Elephpant
{
    function __construct(private string $title, private float $PIE) {}
}

class Error_Class
{
    public static function newError($errno = false)
    {
        $elephpant = new Elephpant("Bluey", M_PI);
        $randoVar = 42;
        throw new Exception("My Exception");
    }
}

set_exception_handler('exceptionHandler');

$e = Error_Class::newError(42);
```

Flamegraphs

```
xdebug.mode=trace
xdebug.start_with_request=no
xdebug.trace_format=3 // time=3, memory=4
```

Flamegraphs

Outputs to a file:

```
{main};require;require_once;create_initial_taxonomies;__ 2574
{main};require;require_once;create_initial_taxonomies;register_taxonomy;wp_parse_args 1803
{main};require;require_once;create_initial_taxonomies;register_taxonomy;WP_Taxonomy->__construct;WP_Taxonomy->set_props;wp_parse_args 17
{main};require;require_once;create_initial_taxonomies;register_taxonomy;WP_Taxonomy->__construct;WP_Taxonomy->set_props;apply_filters 23
{main};require;require_once;create_initial_taxonomies;register_taxonomy;WP_Taxonomy->__construct;WP_Taxonomy->set_props;apply_filters 24
{main};require;require_once;create_initial_taxonomies;register_taxonomy;WP_Taxonomy->__construct;WP_Taxonomy->set_props;array_merge 2876
{main};require;require_once;create_initial_taxonomies;register_taxonomy;WP_Taxonomy->__construct;WP_Taxonomy->set_props;array_merge 1723
{main};require;require_once;create_initial_taxonomies;register_taxonomy;WP_Taxonomy->__construct;WP_Taxonomy->set_props;array_unique 140
{main};require;require_once;create_initial_taxonomies;register_taxonomy;WP_Taxonomy->__construct;WP_Taxonomy->set_props;get_taxonomy_lab
{main};require;require_once;create_initial_taxonomies;register_taxonomy;WP_Taxonomy->__construct;WP_Taxonomy->set_props;get_taxonomy_lab
{main};require;require_once;create_initial_taxonomies;register_taxonomy;WP_Taxonomy->__construct;WP_Taxonomy->set_props;get_taxonomy_lab
{main};require;require_once;create_initial_taxonomies;register_taxonomy;WP_Taxonomy->__construct;WP_Taxonomy->set_props;get_taxonomy_lab
{main};require;require_once;create_initial_taxonomies;register_taxonomy;WP_Taxonomy->__construct;WP_Taxonomy->set_props;get_taxonomy_lab
{main};require;require_once;create_initial_taxonomies;register_taxonomy;WP_Taxonomy->__construct;WP_Taxonomy->set_props;get_taxonomy_lab
{main};require;require_once;create_initial_taxonomies;register_taxonomy;WP_Taxonomy->__construct;WP_Taxonomy->set_props 30859
{main};require;require_once;create_initial_taxonomies;register_taxonomy;WP_Taxonomy->__construct 3997
{main};require;require_once;create_initial_taxonomies;register_taxonomy;WP_Taxonomy->add_rewrite_rules 1743
{main};require;require_once;create_initial_taxonomies;register_taxonomy;WP_Taxonomy->add_hooks;add_filter;WP_Hook->add_filter;_wp_filter
{main};require;require_once;create_initial_taxonomies;register_taxonomy;WP_Taxonomy->add_hooks;add_filter;WP_Hook->add_filter 5380
{main};require;require_once;create_initial_taxonomies;register_taxonomy;WP_Taxonomy->add_hooks;add_filter 4588
{main};require;require_once;create_initial_taxonomies;register_taxonomy;WP_Taxonomy->add_hooks 3257
{main};require;require_once;create_initial_taxonomies;register_taxonomy;do_action 2475
{main};require;require_once;create_initial_taxonomies;register_taxonomy;do_action 2524
{main};require;require_once;create_initial_taxonomies;register_taxonomy 16131
{main};require;require_once;create_initial_taxonomies;_x;translate_with_gettext_context;get_translations_for_domain 1624
{main};require;require_once;create_initial_taxonomies;_x;translate_with_gettext_context;NOOP_Translations->translate 1513
{main};require;require_once;create_initial_taxonomies;_x;translate_with_gettext_context;apply_filters 2174
{main};require;require_once;create_initial_taxonomies;_x;translate_with_gettext_context;apply_filters 2204
{main};require;require_once;create_initial_taxonomies;_x;translate_with_gettext_context 8876
{main};require;require_once;create_initial_taxonomies;_x 2925
{main};require;require_once;create_initial_taxonomies;_x;translate_with_gettext_context;get_translations_for_domain 1583
{main};require;require_once;create_initial_taxonomies;_x;translate_with_gettext_context;NOOP_Translations->translate 1483
{main};require;require_once;create_initial_taxonomies;_x;translate_with_gettext_context;apply_filters 2214
{main};require;require_once;create_initial_taxonomies;_x;translate_with_gettext_context;apply_filters 2214
{main};require;require_once;create_initial_taxonomies;_x;translate_with_gettext_context 8777
{main};require;require_once;create_initial_taxonomies;_x 2785
{main};require;require_once;create_initial_taxonomies;current_theme_supports 2114
{main};require;require_once;create_initial_taxonomies;register_taxonomy;wp_parse_args 1844
```

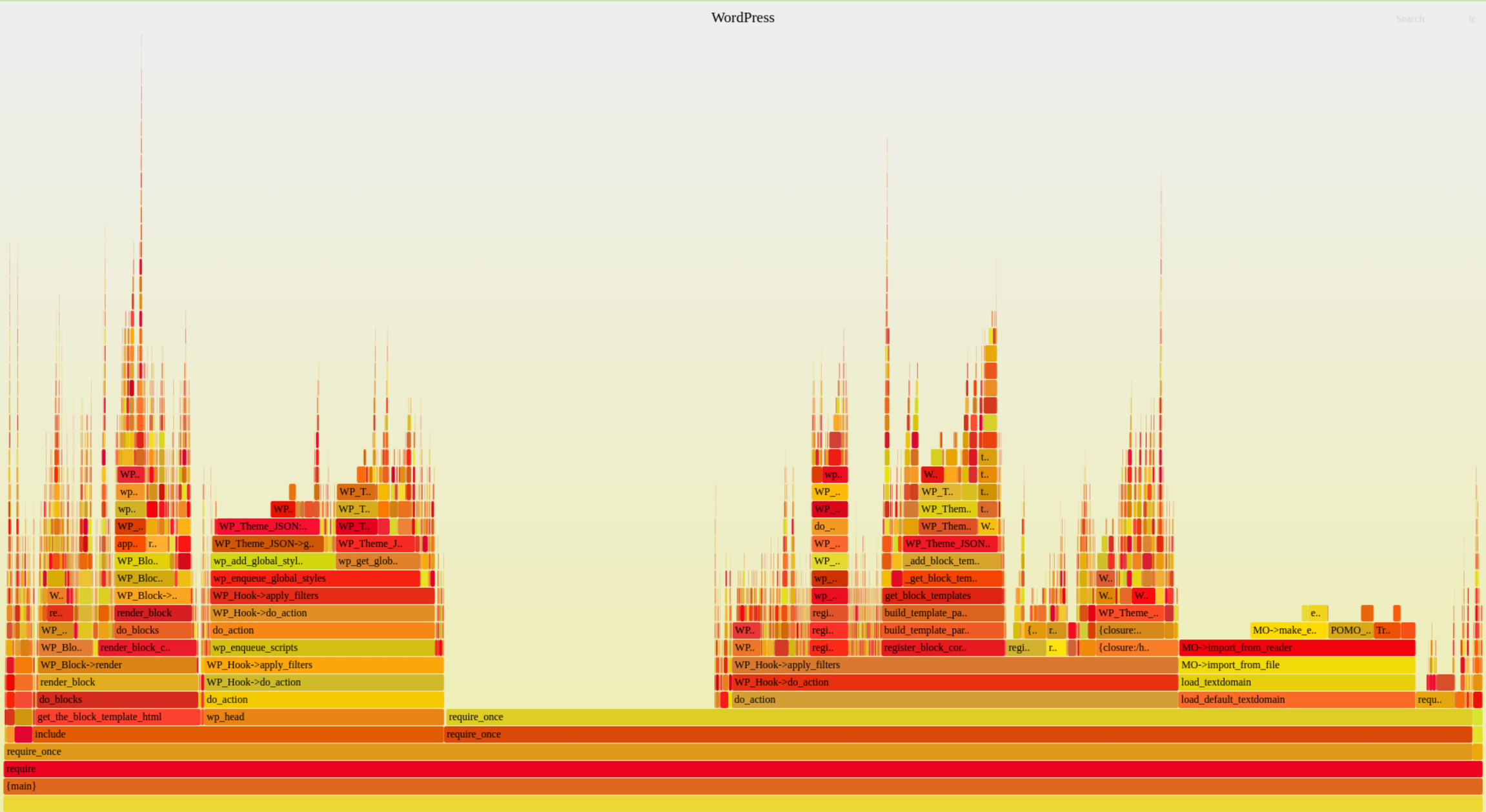
Flamegraphs

Convert to flamegraph:

```
zcat /tmp/trace._.xt.gz
    | /tmp/FlameGraph/flamegraph.pl --width=1728 --height=20 --title="WordPress"
    > /tmp/derick/flamegraph.svg
```

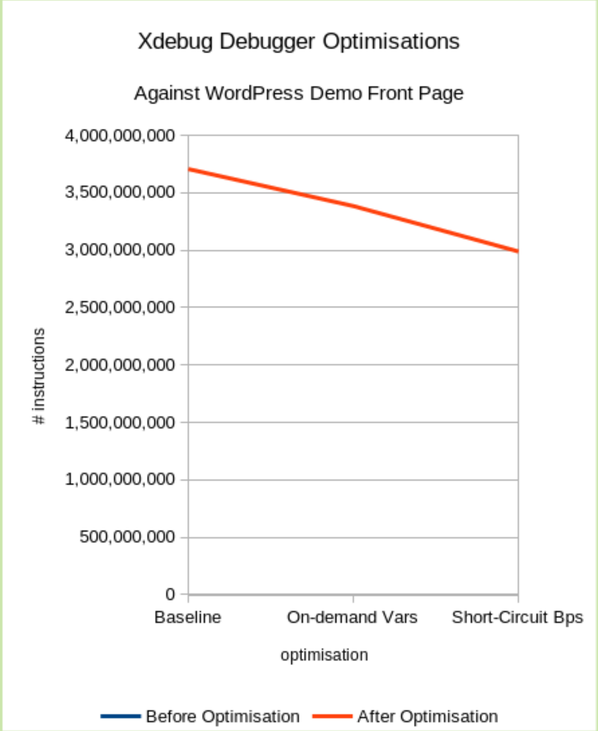
https://github.com/brendangregg/FlameGraph

Flamegraphs



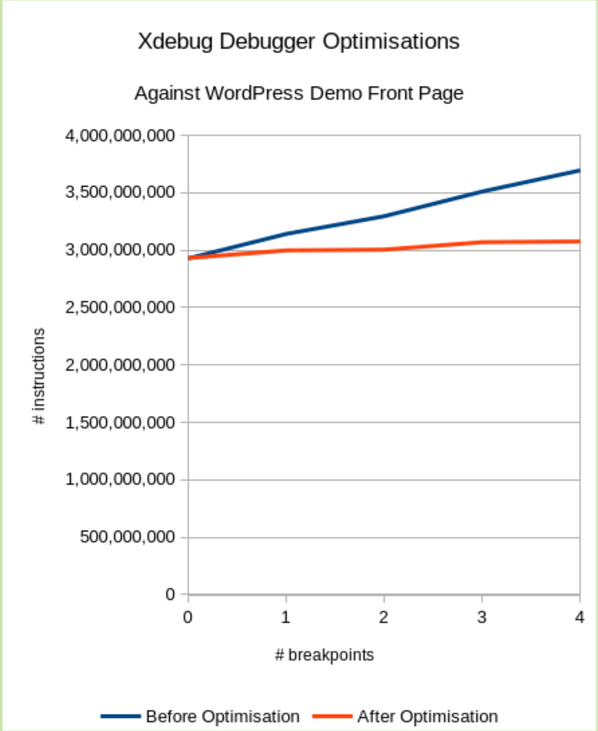
Xdebug 3.3 Step Debugger Optimisations

Do less work:



Xdebug 3.3 Step Debugger Optimisations

Do smarter work:



Supporting Xdebug

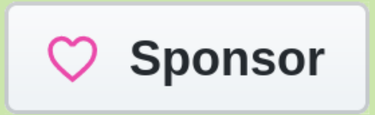
It's Open Source and free (as in "free beer")

I would like to work more on Xdebug, to help **you**

Features and Bugs

`https://bugs.xdebug.org`

`https://www.patreon.com/derickr`



`https://github.com/sponsors/derickr`

Business Support

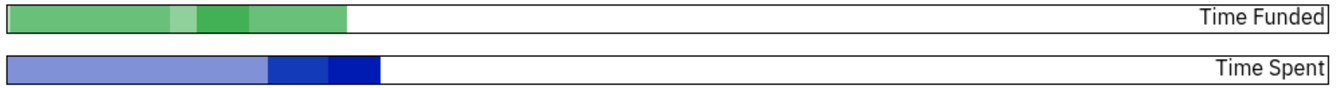
Community	Pro	Business
	1 developer	2-20 developers
Choose amount on Patreon or GitHub	£ 200/year *	£ 1000/year *
Tech Support: Stackoverflow Issue Priority: Low ✔ Monthly News Post	✔ Tech Support: E-mail ✔ Issue Priority: Medium ✔ Access to Commercial Add-ons (soon) ✔ Monthly E-mail Newsletter (opt-in) ✔ Invoice †	✔ Tech Support: E-mail ✔ Issue Priority: High ✔ An hour long presentation or workshop ✔ Access to Commercial Add-ons (soon) ✔ Monthly E-mail Newsletter (opt-in) ✔ Invoice † ✔ "Thank You" with link on site
Support Xdebug via Patreon or GitHub	Sign Up	Sign Up

https://xdebug.org/support

TYPO3 · Tideways · Xdebug Cloud · Seiden Group · Offers.com · Private Packagist · VEMA eG · Bubble Shooter · Find My Electric · 11com7 · IO Belgium · b13 · REWE Digital GmbH

Transparency

September 2023



Day	Type	Description	Hours
3	bug	Reinvestigating comments to #1374	0.50
5	xdebug3	Preparing releases etc for PHP 8.3 and a 3.3.0alpha1 release	4.00
6	xdebug3	Finalising 3.3.0alpha1 release	0.75
6	xdebug3	Making 3.3.0alpha2 release as PECL web was broken	0.75
8	xdebug3	Issue #2203: Change default max_nesting_level to 512	0.25
14	bug	Triaging issues	2.50
18	xdebug3	Issue #2077: Reimplementing collect_params, after a few exploratory implementations	3.00
19	xdebug3	Issue #1732: Finalise and merge flamegraph tracing output	2.00
25	bug	Investigate read after free with generators	1.00
25	docs	Make flamegraph video	1.50
25	docs	Make dbgpProxy video	3.00
28	xdebug3	Switching from intercepting execute_ex to Observer API	4.00
29	xdebug3	Switching from intercepting execute_ex to Observer API	5.00

For additional information, please see the **monthly** report.

Any Queries?

Resources

Slides

`https://derickrethans.nl/talks/xdebug-ipc23`

`https://xdebug.org`

`https://xdebug.cloud`

Derick Rethans — @derickr@phpc.social — derick@xdebug.org