What's new in PHP 8.0?

Nikita Popov

SymfonyWorld
ONLINE 2020
DECEMBER 3-4

PHP 8.0

- Just-In-Time (JIT) compiler
- Many new language features
- SemVer major version: Backwards incompatible changes

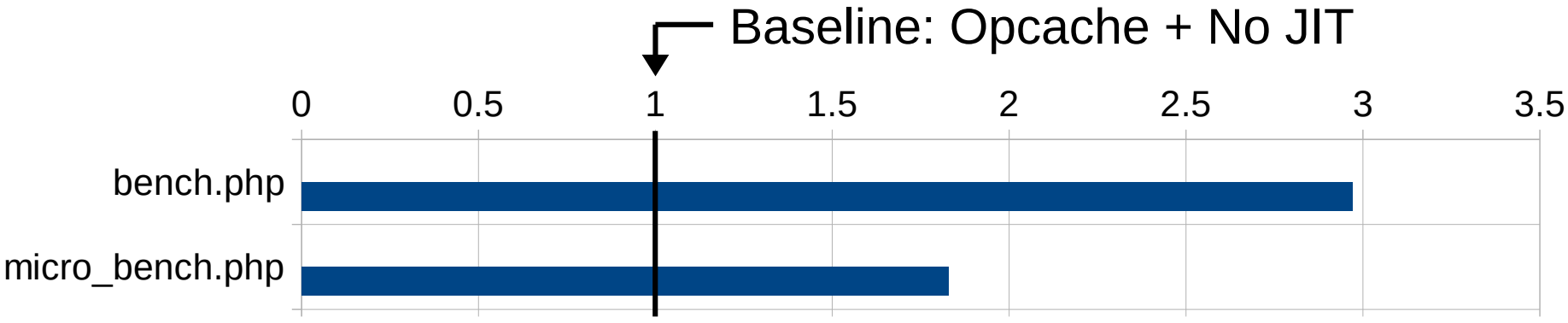
Just-In-Time (JIT) Compiler

- Compiles PHP code to x86 machine code to improve performance

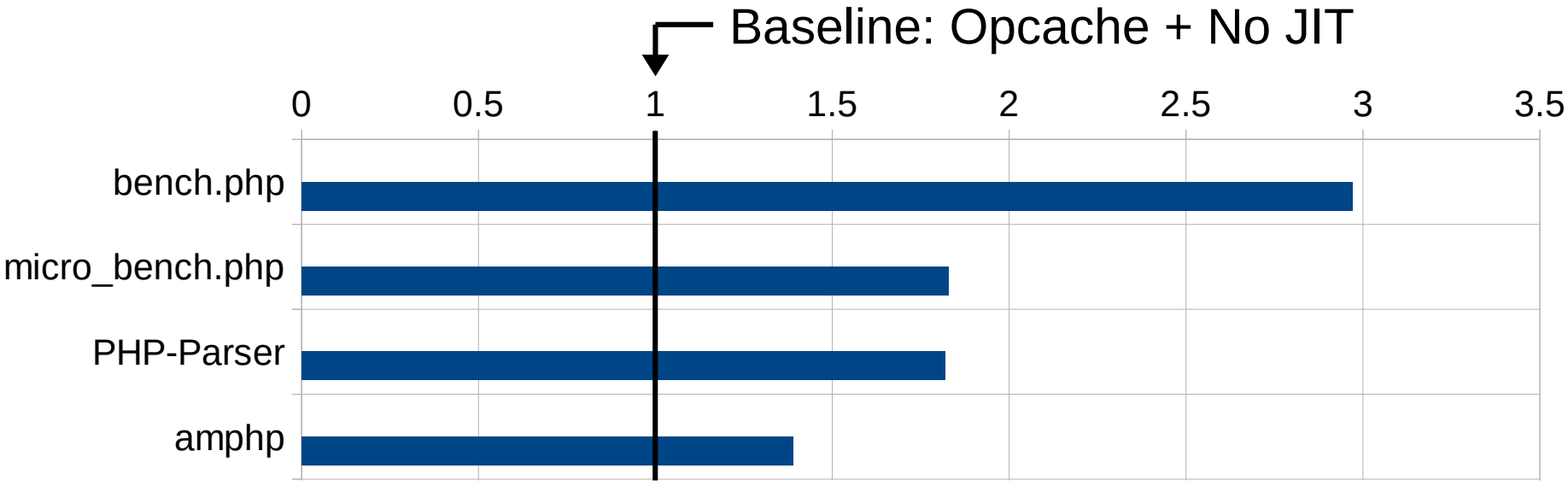
Just-In-Time (JIT) Compiler

- Compiles PHP code to x86 machine code to improve performance
- Basic usage:
  - Enable opcache
  - opcache.jit_buffer_size=128M

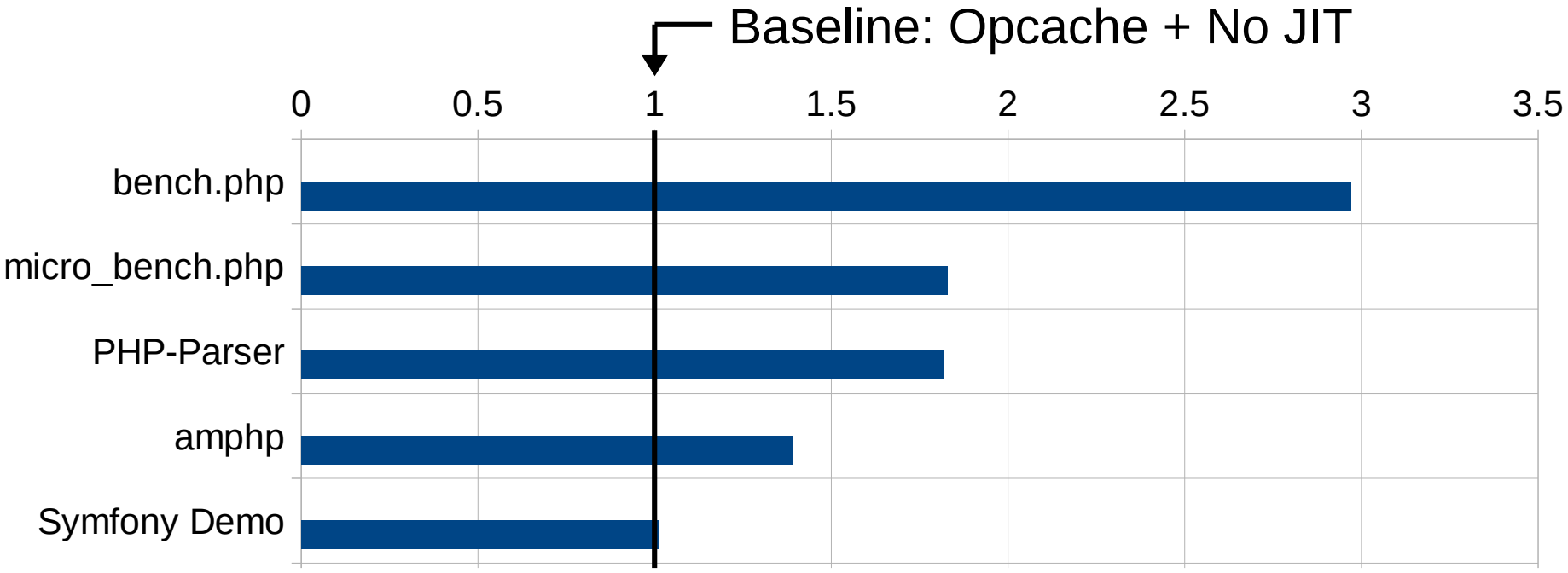
Just-In-Time (JIT) Compiler



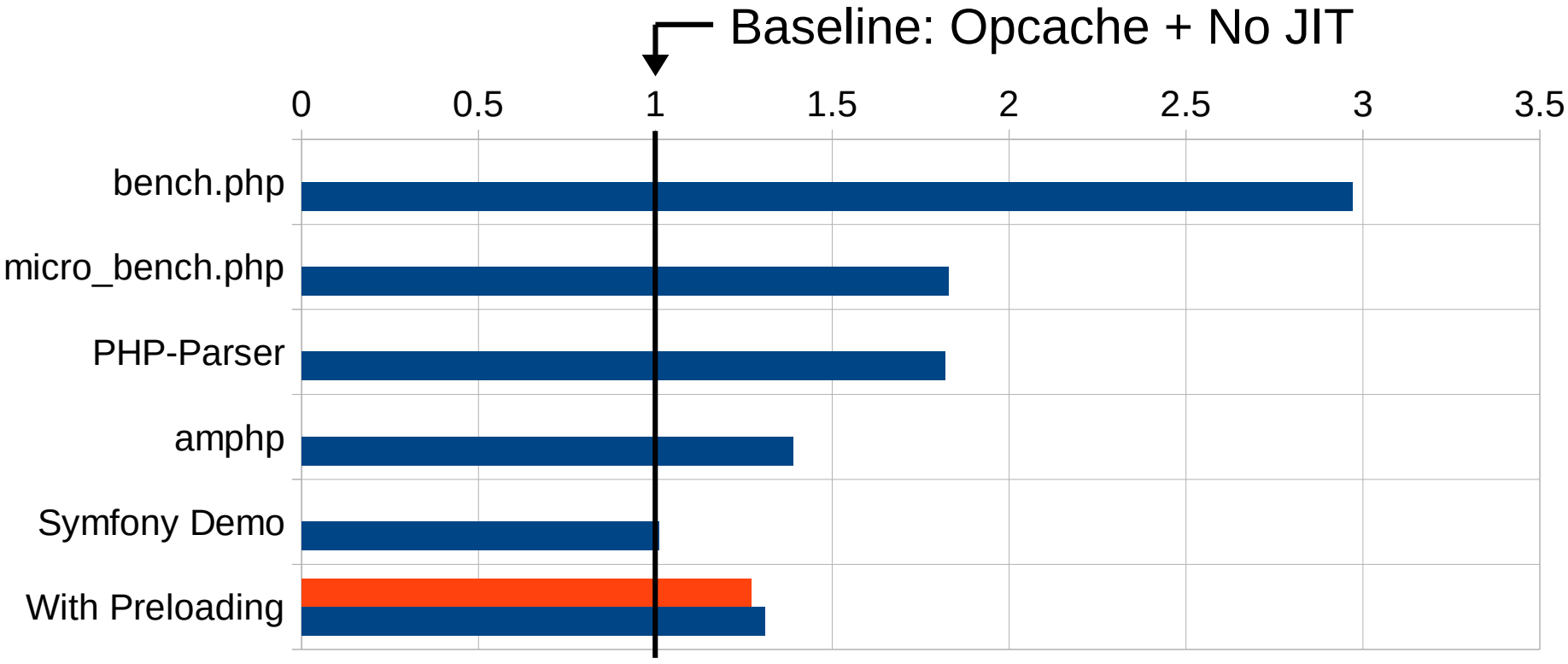
Just-In-Time (JIT) Compiler



Just-In-Time (JIT) Compiler



Just-In-Time (JIT) Compiler



Attributes

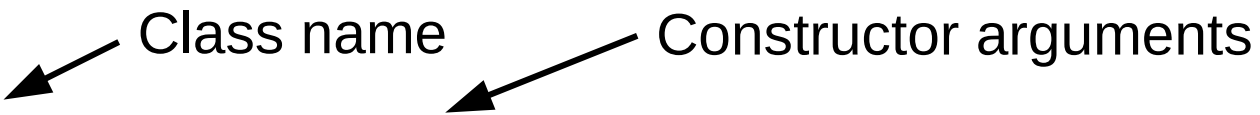
```php
<?php
use Symfony\Component\Routing\Annotation\Route;

class SomeController {
    /**
     * @Route("/path", name="action")
     */
    public function someAction() {
    }
}
```

Attributes

```php
<?php
use Symfony\Component\Routing\Annotation\Route;

class SomeController {


    #[Route("/path", name: "action")]
    public function someAction() {
    }
}
```

Attributes

```php
<?php
use Symfony\Component\Routing\Annotation\Route;

class SomeController {

    #[Route("/path", name: "action")]
    public function someAction() {
    }
}
```



Attributes

```php
<?php
namespace Symfony\Component\Routing\Annotation;
use Attribute;

#[Attribute(Attribute::TARGET_METHOD)]
class Route {
    public function __construct(string $path, ...) {
        // ...
    }
}
```

Attributes

```php
<?php
$rm = new ReflectionMethod(
    SomeController::class, "someAction");
foreach ($rm->getAttributes() as $attr) {
    var_dump($attr->getName());
    // => "Symfony\Component\Routing\Annotation\Route"

    var_dump($attr->getArguments());
    // => ["/path", "name" => "action"]

    var_dump($attr->newInstance());
    // object(Symfony\Component\Routing\Annotation\Route)
}
```

Attributes

```php
<?php
$rm = new ReflectionMethod(
    SomeController::class, "someAction");
foreach ($rm->getAttributes() as $attr) {
    var_dump($attr->getName());
    // => "Symfony\Component\Routing\Annotation\Route"

    var_dump($attr->getArguments());
    // => ["/path", "name" => "action"]

    var_dump($attr->newInstance());
    // object(Symfony\Component\Routing\Annotation\Route)
}
```


Attribute validation happens HERE.

Constructor Promotion

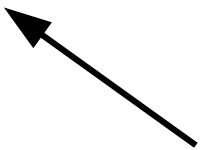
```php
<?php
class Point {
    public float $x;
    public float $y;
    public float $z;

    public function __construct(
        float $x = 0.0,
        float $y = 0.0,
        float $z = 0.0,
    ) {
        $this->x = $x;
        $this->y = $y;
        $this->z = $z;
    }
}
```

Constructor Promotion

```php
<?php
class Point {
    public function __construct(
        public float $x = 0.0,
        public float $y = 0.0,
        public float $z = 0.0,
    ) {}
}
```

Constructor Promotion

```php
<?php
class Point {
    public function __construct(
        public float $x = 0.0,
        public float $y = 0.0,
        public float $z = 0.0,
    ) {}
}
```



Trailing comma now allowed

Named Arguments

```php
<?php

// Using positional arguments:
array_fill(0, 100, 50);
```

Named Arguments

```php
<?php

// Using positional arguments:
array_fill(0, 100, 50);

// Using named arguments:
array_fill(start_index: 0, count: 100, value: 50);
```

Named Arguments

```php
<?php

// Using positional arguments:
array_fill(0, 100, 50);

// Using named arguments:
array_fill(start_index: 0, count: 100, value: 50);

// Order does not matter!
array_fill(value: 50, count: 100, start_index: 0);
```

Named Arguments

```php
<?php

// Using positional arguments:
htmlspecialchars(
    $string, ENT_COMPAT | ENT_HTML401, 'UTF-8', false);
```

Named Arguments

```php
<?php

// Using positional arguments:
htmlspecialchars(
    $string, ENT_COMPAT | ENT_HTML401, 'UTF-8', false);

// Using named arguments:
htmlspecialchars($string, double_encode: false);
```

Named Arguments

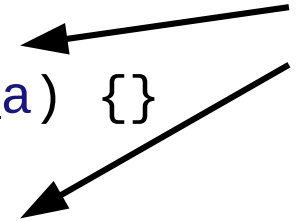
```php
<?php
use Symfony\Component\Routing\Annotation\Route;

class SomeController {
    #[Route("/path", name: "action")]
    public function someAction() {
        // ...
    }
}
```

Named Arguments

```php
<?php

class Point {
    public function __construct(
        public float $x,
        public float $y,
        public float $z,
    ) {}
}

new Point(x: 2.0, y: 3.1, z: 4.2);
```

Named Arguments

```php
<?php

class A {
    public function method($name_a) {}
}
class B extends A {
    public function method($name_b) {}
}

// Error: Unknown named parameter $name_a
(new B)->method(name_a: 42);
```



Names not the same

Union Types

```php
<?php
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
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

```php
<?php
function strpos(
    string $haystack, string $needle, int $offset = 0
): int|false {}
```

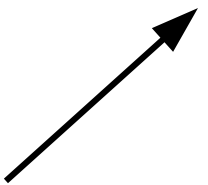
Union Types

```php
<?php
function strpos(
    string $haystack, string $needle, int $offset = 0
): int|false {}
```

Union Types

```php
<?php
function strpos(
    string $haystack, string $needle, int $offset = 0
): int|false {}

function array_key_first(array $arg): int|string|null {}
```

?Type is a shorthand for Type|null now



Union Types

```php
<?php declare(strict_types=0);

function test(int|float|bool $arg) {
    var_dump($arg);
}

test(45);      // int(45)
test(45.8);    // float(45.8)
test("45");    // int(45)
test("45.8"); // float(45.8)
test("");      // bool(false)
test("X");     // bool(true)
test([]);      // TypeError
```

Union Types

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

function test(int|float|bool $arg) {
    var_dump($arg);
}

test(45);      // int(45)
test(45.8);    // float(45.8)
test("45");    // TypeError
test("45.8"); // TypeError
test("");      // TypeError
test("X");     // TypeError
test([]);      // TypeError
```

Mixed Type

- Distinguishes between:
  - Type is missing because I didn't add one yet
  - This function really does accept any value

Mixed Type

```php
<?php

function var_dump(mixed $value, mixed ...$value): void {}

function serialize(mixed $value): string {}
```

Mixed Type

```php
<?php

function var_dump(mixed $value, mixed ...$value): void {}

function serialize(mixed $value): string {}

// "mixed" is a common approximation for generic functions:
function array_reduce(
    array $arg, callable $callback,
    mixed $initial = null
): mixed {}
```

Mixed Type

```php
<?php

// For argument types:
// No type same as mixed type

class A {
    public function method(mixed $arg) {}
}

class B extends A {
    public function method($arg) {}
}
```

Mixed Type

```php
<?php

// For return types:
// No type effectively means mixed|void

class A {
    public function method(): mixed {}
}

class B extends A {
    public function method() {}
}
```

Mixed Type

```php
<?php

// For return types:
// No type effectively means mixed|void

class A {
    public function method(): mixed {}
}

class B extends A {
    public function method() {}
}
```

Forbidden: Widening return type

Static Return Type

```php
<?php

// Named constructor:
class TestParent {
    public function createFromWhatever($whatever): static {
        return new static($whatever);
    }
}
```

Static Return Type

```php
<?php

// Named constructor:
class TestParent {
    public function createFromWhatever($whatever): static {
        return new static($whatever);
    }
}

class TestChild extends TestParent {}

// TestChild::createFromWhatever(...)
// must return TestChild, not TestParent!
```

Static Return Type

```php
<?php

// Fluent methods:
class Test {
    public function doWhatever(): static {
        // Do whatever.
        return $this;
    }
}
```

Static Return Type

```php
<?php

// Wither pattern:
class Test {
    public function withWhatever($whatever): static {
        $clone = clone $this;
        $clone->whatever = $whatever;
        return $clone;
    }
}
```

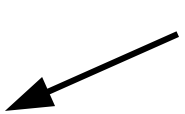
Match Expression

```php
<?php
switch ($operator) {
case '+':
    $result = $a + $b;
    break;
case '-':
    $result = $a - $b;
    break;
case '*':
    $result = $a * $b;
    break;
default:
    throw new UnsupportedOperator($operator);
}
```

Match Expression

```php
<?php

$result = match ($operator) {
    '+' => $a + $b,
    '-' => $a - $b,
    '*' => $a * $b,
    default => throw new UnsupportedOperator($operator);
};
```

Match Expression




```php
<?php

$result = match ($operator) {
    '+' => $a + $b,
    '-' => $a - $b,
    '*' => $a * $b,
    default => throw new UnsupportedOperator($operator);
};
```

Match Expression

```php
<?php

$result = match ($operator) {
    '+' => $a + $b,
    '-' => $a - $b,
    '*' => $a * $b,
    default => throw new UnsupportedOperator($operator),
};
```

Expression with a return value

Each match clause is an expression
("throw" is an expression now)

Match Expression

```php
<?php

function evalOp($operator, $a, $b) {
    return match ($operator) {
        '+' => $a + $b,
        '-' => $a - $b,
        '*' => $a * $b,
    };
}

// Match is exhaustive:
evalOp('/', 10, 2); // UnhandledMatchError
```

Match Expression

```php
<?php

function evalOp($operator, $a, $b) {
    return match ($operator) {
        '+' => $a + $b,
        '-' => $a - $b,
        '*' => $a * $b,
    };
}

// Match compares using ===, not ==.
evalOp(true, 10, 2); // UnhandledMatchError
```

Nullsafe Operator

```php
<?php

$name = $session !== null
            ? $session->getUser()->name
            : null;

// Same as:
$name = $session?->getUser()->name;
```

Nullsafe Operator

```php
<?php

$name = $session?->getUser()?->name;

// Approximately same as:
$name = null;
if ($session !== null) {
    $user = $session->getUser();
    if ($user !== null) {
        $name = $user->name;
    }
}
```

Other Features

- catch (Exception) without variable
- $object::class
- str_contains(), str_starts_with(), str_ends_with()
- get_debug_type()
- Stable sorting
- WeakMap

Backwards Compatibility Breaks

- Functionality deprecated before PHP 8.0 has been removed!
- Full list: https://github.com/php/php-src/blob/PHP-8.0/UPGRADING

Number to String Comparison

```php
<?php

$validValues = ["foo", "bar", "baz"];
$value = 0;
var_dump(in_array($value, $validValues));
// bool(true)
// ???
```

Number to String Comparison

```php
<?php

0 == "foo";
// Before:
0 == (int)"foo";
// After:
(string)0 == "foo";
```

Number to String Comparison

Comparison	Before	After
0 == "0"	true	true
0 == "0.0"	true	true
0 == "foo"	true	false
0 == ""	true	false
42 == " 42"	true	true
42 == "42foo"	true	false

Resource To Object Migration

- Long term goal: Convert all resources to objects
- Objects are type-safe and have much better internal support

Resource To Object Migration

- Long term goal: Convert all resources to objects
- Objects are type-safe and have much better internal support
- Using "opaque objects"
  - Actual object-oriented APIs may be added later

Resource To Object Migration

- CurlHandle, CurlMultiHandle, CurlShareHandle
- EnchantBroker, EnchantDictionary
- GdImage
- InflateContext, DeflateContext
- OpenSSLCertificate, OpenSSLCertificateSigningRequest, OpenSSLAsymmetricKey
- Shmop
- Socket, AddressInfo
- SysvMessageQueue, SysvSemaphore, SysvSharedMemory
- XmlParser
- XmlWriter (already had an OO API)

Resource To Object Migration

```php
<?php

$image = imagecreatefrompng($path);
if (!is_resource($image)) {
    throw new MalformedImageException;
}
```

Resource To Object Migration

```php
<?php

$image = imagecreatefrompng($path);
if (!is_resource($image)) {
    throw new MalformedImageException;
}
```

Now a GdImage object on success

Will always throw...

Resource To Object Migration

```php
<?php

$image = imagecreatefrompng($path);
if (false === $image) {
    throw new MalformedImageException;
}
```

Warning → Error exception

- Many warnings converted to Error exceptions
  - TypeError
  - ValueError

Warning → Error exception

- Only allowed for error conditions that imply programmer error
- It makes no sense to "handle" the error, code needs to be fixed instead

Warning → Error exception

```php
<?php

var_dump(strlen([]));
// Warning: strlen() expects parameter 1 to be string,
// array given
// NULL

function strlen(string $str): int|null {}
```

Warning → Error exception

```php
<?php

var_dump(strlen([]));
// Uncaught TypeError: strlen(): Argument #1 ($str)
// must be of type string, array given


function strlen(string $str): int {}
```

Warning → Error exception

```php
<?php

var_dump(array_fill(0, -100, "foobar"));
// Warning: array_fill(): Number of elements can't
// be negative
// bool(false)

function array_fill(
    int $start_index, int $num, mixed $value
): array|false {}
```

Warning → Error exception

```php
<?php

var_dump(array_fill(0, -100, "foobar"));
// Uncaught ValueError: array_fill(): Argument #2 ($count)
// must be greater than or equal to 0


function array_fill(
    int $start_index, int $count, mixed $value
): array {}
```

Warning → Error exception

```php
<?php

var_dump(fopen("does_not_exist.txt", "r"));
// Warning: fopen(does_not_exist.txt):
// Failed to open stream: No such file or directory
// bool(false)
```

Warning → Error exception

```php
<?php

var_dump(fopen("does_not_exist.txt", "r"));
// Warning: fopen(does_not_exist.txt):
// Failed to open stream: No such file or directory
// bool(false)
```

NOT going to change!

fopen() failure is an environment failure condition,
it does not imply programmer error!

PHP Stubs

- PHP stub files specify function signatures for internal functions/methods
- Used to generate C code for function registration

PHP Stubs

```php
<?php

function array_search(
    mixed $needle, array $haystack, bool $strict = false
): int|string|false {}
```

PHP Stubs

```php
<?php

function array_search(
    mixed $needle, array $haystack, bool $strict = false
): int|string|false {}

ZEND_BEGIN_ARG_WITH_RETURN_TYPE_MASK_EX(
        arginfo_array_search, 0, 2,
        MAY_BE_LONG|MAY_BE_STRING|MAY_BE_FALSE)
    ZEND_ARG_TYPE_INFO(0, needle, IS_MIXED, 0)
    ZEND_ARG_TYPE_INFO(0, haystack, IS_ARRAY, 0)
    ZEND_ARG_TYPE_INFO_WITH_DEFAULT_VALUE(
            0, strict, _IS_BOOL, 0, "false")
ZEND_END_ARG_INFO()
```

PHP Stubs

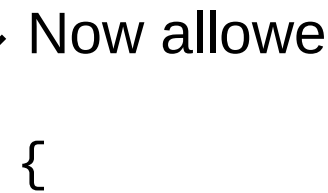
- Data available through Reflection:
  - ReflectionFunction::getReturnType()
  - ReflectionParameter::getType()
  - ReflectionParameter::getDefaultValue()

PHP Stubs

```php
<?php

// Stub
class DateTime implements DateTimeInterface {
    /** @return DateTime */
    public function add(DateInterval $interval) {}
}

// Your code
class MyDateTime extends DateTime {
    public function add(DateInterval $interval) {
        // Do something
    }
}
```

PHP Stubs

```php
<?php

// Stub
class DateTime implements DateTimeInterface {
    /** @return DateTime */
    public function add(DateInterval $interval) {}
}

// Your code
class MyDateTime extends DateTime {
    public function add(DateInterval $interval) {
        // Do something
    }
}
```



PHP Stubs

```php
<?php

// Stub
class DateTime implements DateTimeInterface {
    /** @return DateTime */
    public function add(DateInterval $interval) {}
}

// Your code
class MyDateTime extends DateTime {
    public function add(DateInterval $interval) {
        // Do something
    }
}
```

A real return type would force all extending classes to specify it.

Now allowed!

PHP Stubs

- Ties in with many other efforts:
  - Union types critical for standard library signatures
  - Warning -> Error exception promotions avoid unnecessary return types
  - Resource to object migration allows use as types
  - Named arguments based on stubs default values

Thank You!