&lt;/&gt;

FIBERS

Nová éra asynchronního programovaní v PHP?

PeoplePath

27. 4. 2022

18:00

U Salzmannů

Salonek, 1. patro

PHP 8.1

New exciting features:

- Enums,
- readonly properties and
- **fibers.**

What are fibers?

In computer science, a fiber is a particularly lightweight thread of execution. Like threads, fibers share address space. However, fibers use cooperative multitasking while threads use preemptive multitasking.

[source: https://en.wikipedia.org/wiki/Fiber_(computer_science)]

What problems does fibers solve?

The problem this RFC seeks to address is a difficult one to explain, but can be referred to as the "What color is your function?" problem.

[source: https://wiki.php.net/rfc/fibers]

What problems does fibers solve?

This RFC (Fibers RFC) seeks to eliminate the distinction between synchronous and asynchronous functions by allowing functions to be interruptible without polluting the entire call stack.

[source: https://wiki.php.net/rfc/fibers]

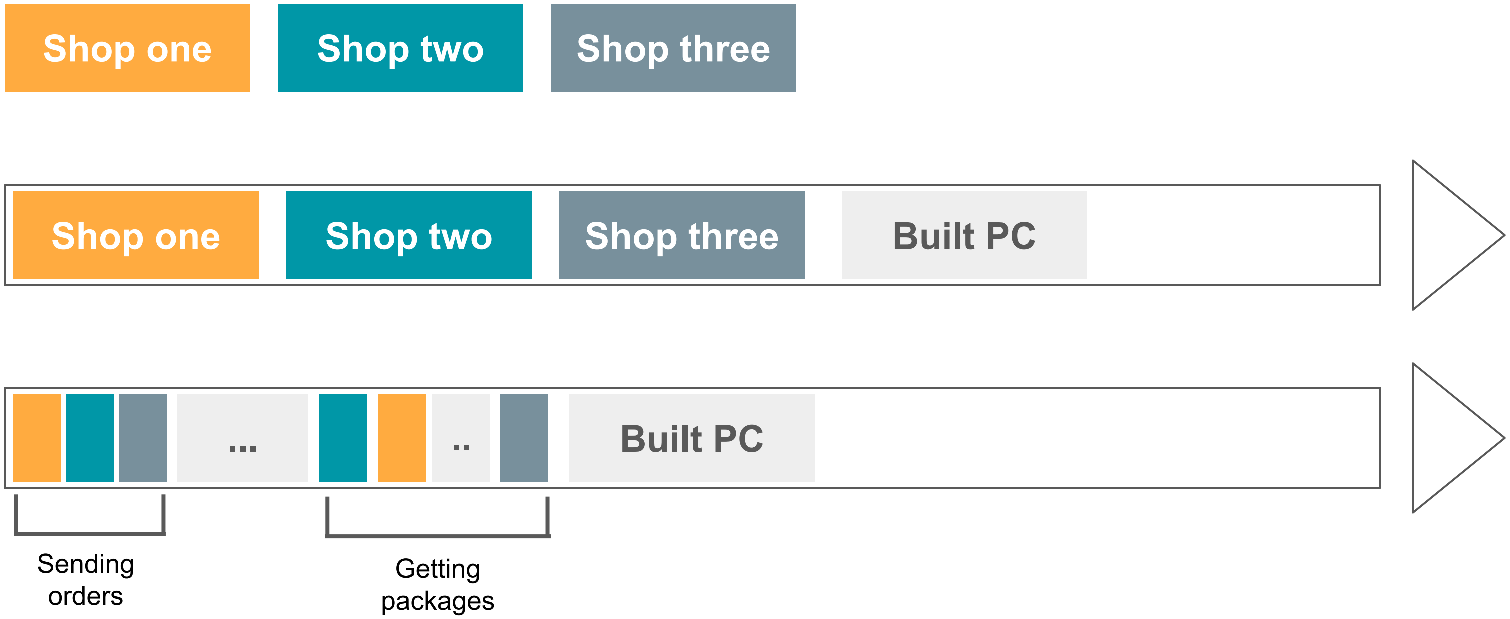
What problems should fibers solve?

- It's a problem that is not so common in PHP world...
- ...but it will be probably more prevalent in future.

Together with other language construction they make asynchronous code more readable and less “hackish”.

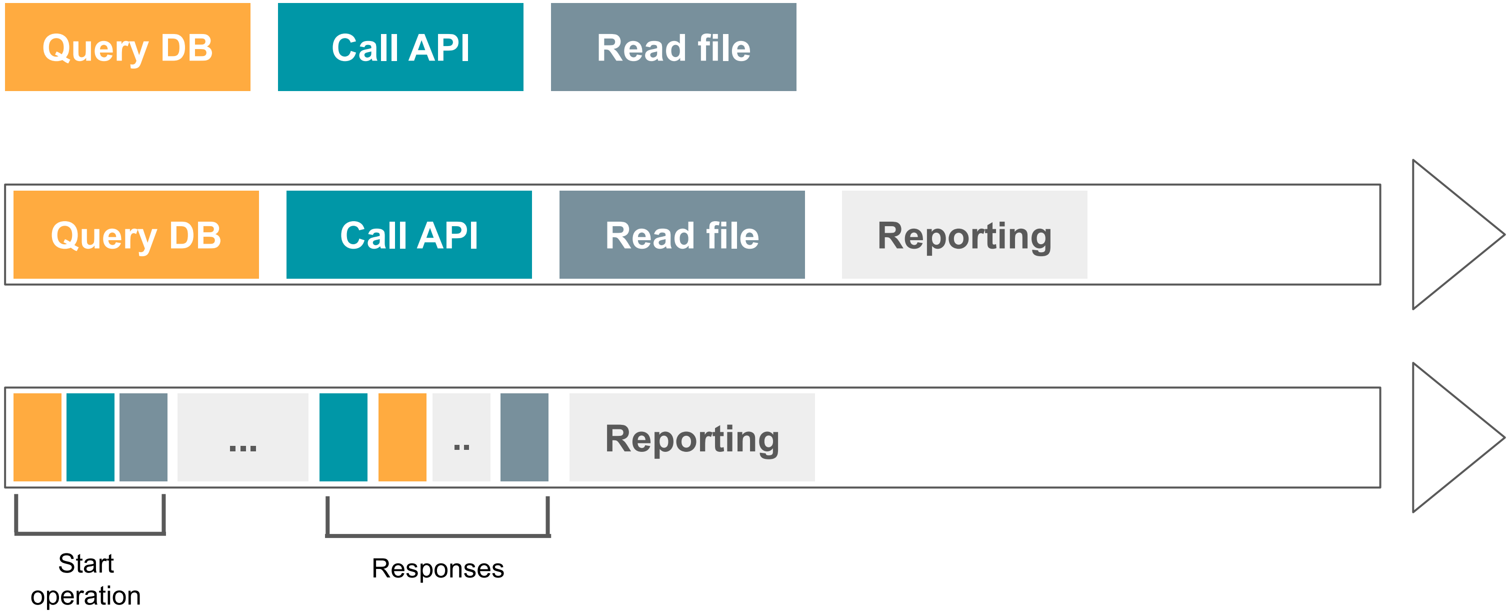
World around us is asynchronous

- ordering PC components from three shops at once.



Asynchronous programing

- It’s IO which usually block the program/request flow



Query DB

Call API

Read file

Query DB

Call API

Read file

Reporting

...

..

Reporting

Start
operation

Responses

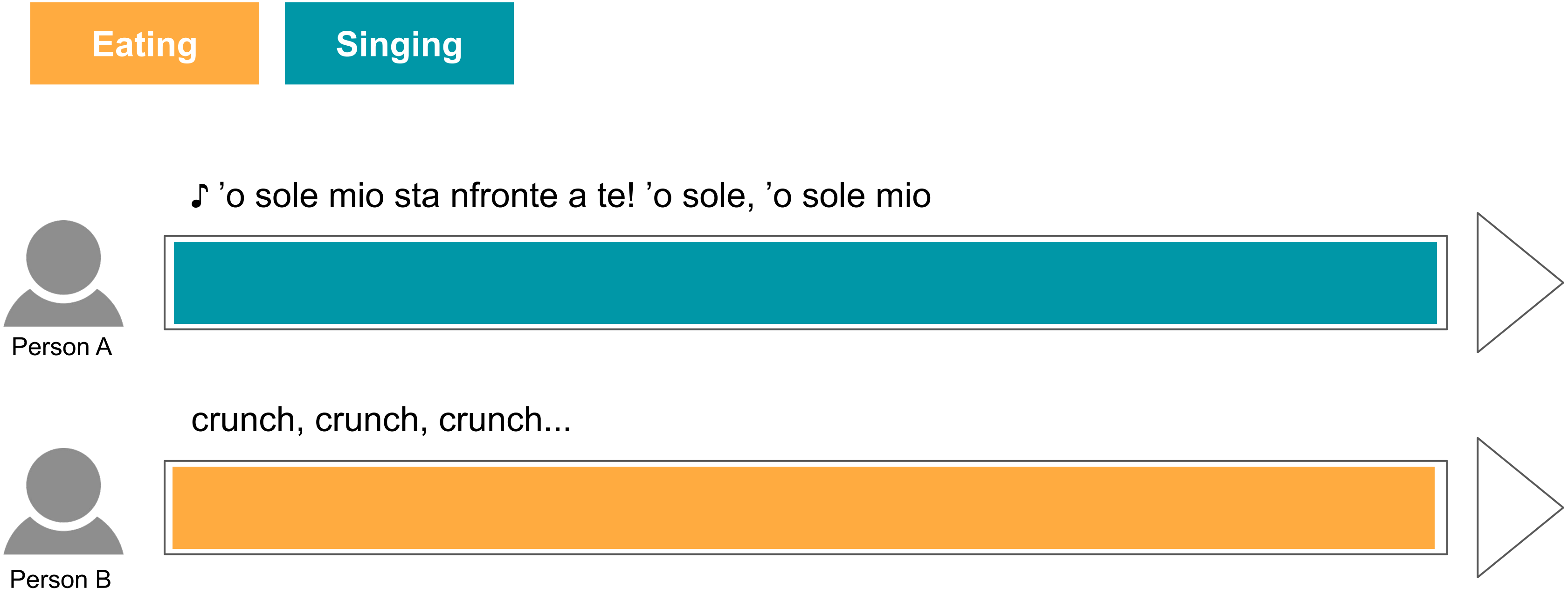
Asynchronous programing

- Asynchronous programming helps us to achieve concurrency.
- Motivation:
  - Faster execution of IO heavy code.
  - Ensure reactivity of an application.

Concurrency

- Concurrency means executing multiple tasks at the same time **but not necessarily simultaneously**.
- Two approaches:
    - parallelism,
    - task switching.

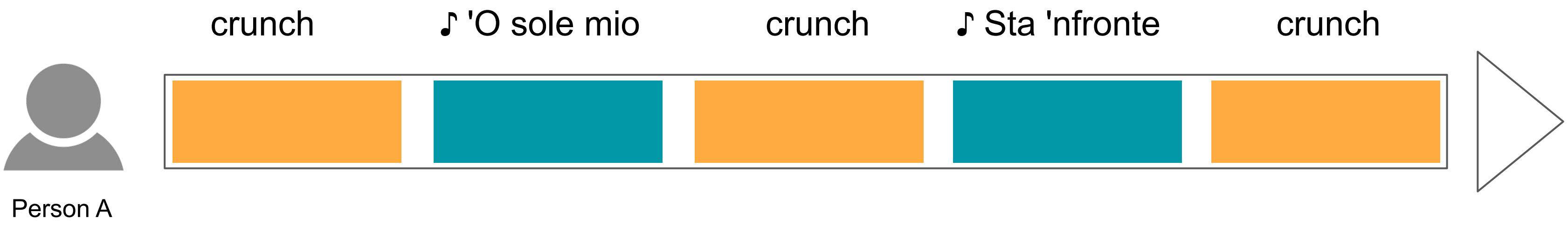
Concurrency - parallelism


Eating
Singing
♪ ’o sole mio sta nfronte a te! ’o sole, ’o sole mio
Person A
crunch, crunch, crunch...
Person B

Concurrency - task switching

Eating

Singing



crunch

♪ 'O sole mio

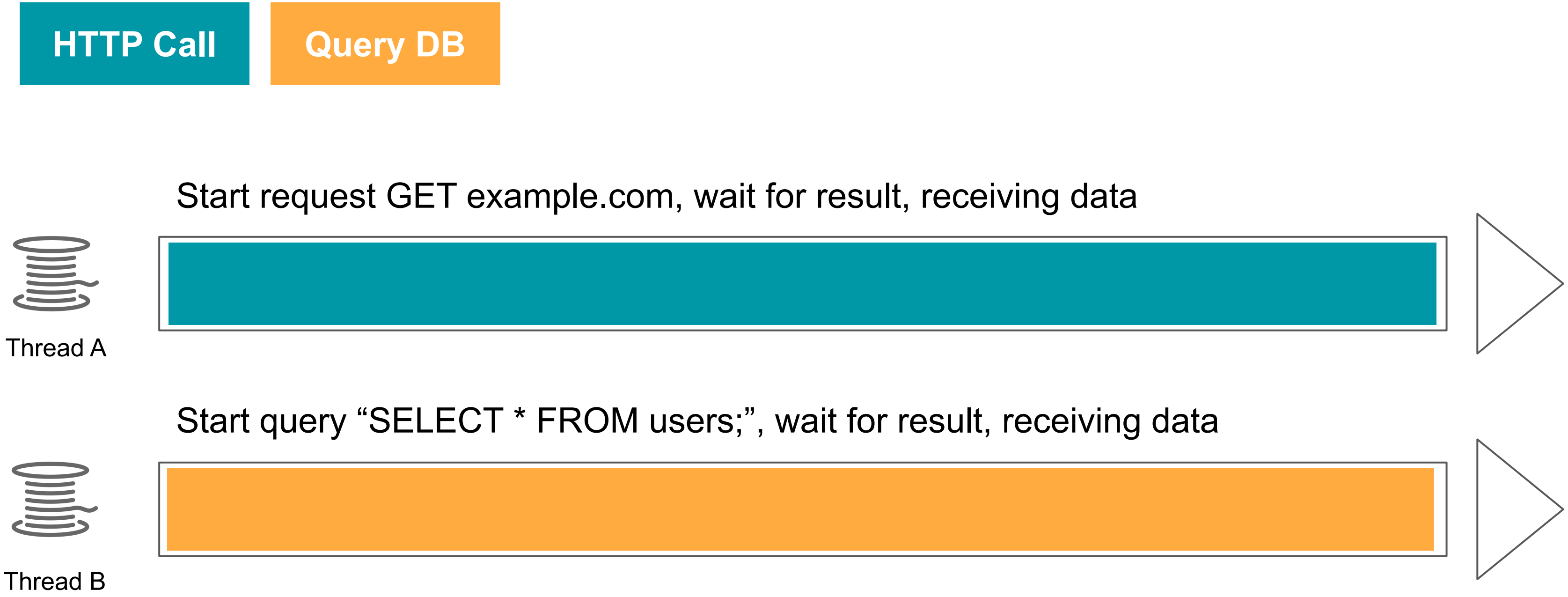
crunch

♪ Sta 'nfronte

crunch

Person A

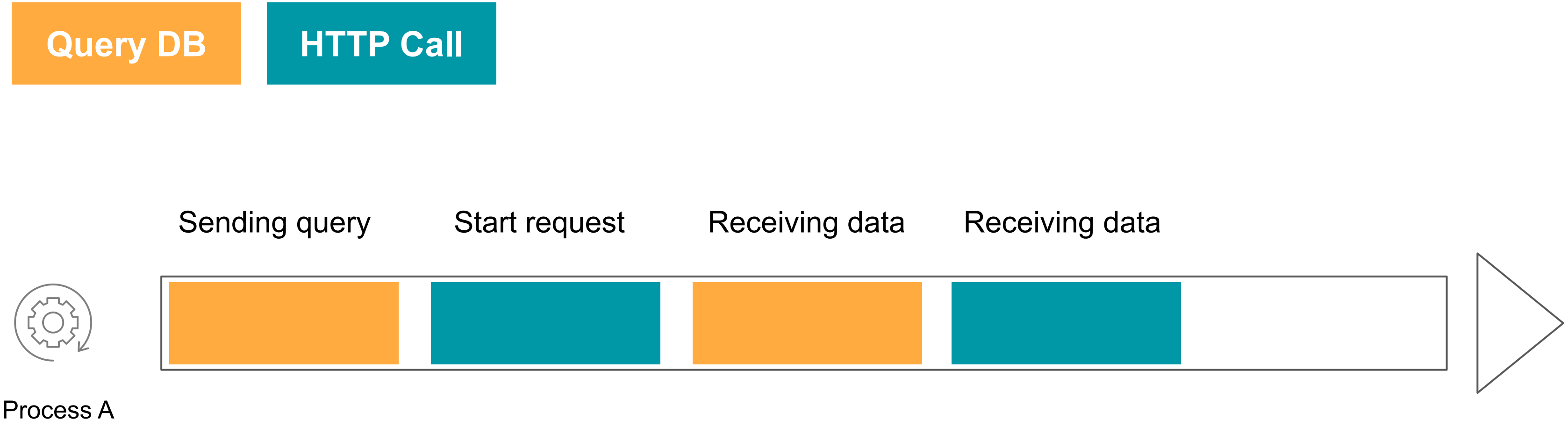
Concurrency - parallelism



Start request GET example.com, wait for result, receiving data

Start query “SELECT * FROM users;”, wait for result, receiving data

Concurrency - task switching



Query DB

HTTP Call

Sending query

Start request

Receiving data

Receiving data

Process A

Concurrency - implementation

- Parallelism:
    - threads, processes
- Task switching:
    - event loops (coroutines + non blocking IO)

Concurrency - implementation in PHP

- Parallelism:
    - Threads:
        - ~~pthreads~~ dead (PHP 7.0-7.2)
        - parallel (from PHP 7.2)
    - Process:
        - fork
        - php fpm

Concurrency - implementation in PHP

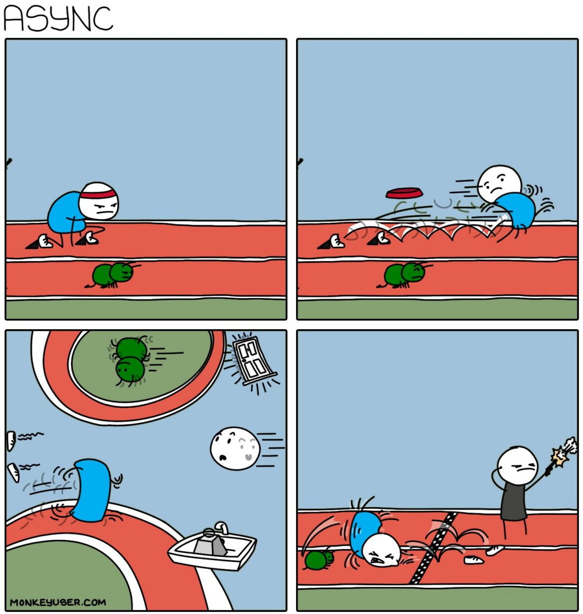
- Task switching:
    - ReactPHP
    - AMP
    - Swoole
    - <your favourite async framework>

Concurrency - threads vs event loops

Threads:

- CPU parallelism (often just illusion).
- Threads can be expensive - context switch.
- Multi-threading is easy. Correct synchronization is hard.

Correct synchronization is hard



Non atomic bank

```php
class Bank {
   private int $balance;
   public function deposit(int $val) {
       $temp = $this->balance;
       $temp = $temp + $val;
       $this->balance = $temp;
   }
}
```

Who Wants to Be a Millionaire?

Two concurrent threads want to add 5$ and 10$ to 500$ balance at same time using same object instance.

What will be the end balance?

Who Wants to Be a Millionaire?

Two concurrent threads want to add 5$ and 10$ to 500$ balance at same time using same object instance.

What will be the balance?

1. End balance will be 515$
2. End balance will be 505$
3. End balance will be 510$

Non atomic bank - one of possible scenarios

Thread 1

```
class Bank {
  private int $balance;        500$
  public function deposit($val)    val = 5$
  {
     $temp = $this->balance;    temp = 500$
     $temp = $temp + $val;      temp = 505$
     $this->balance = $temp;    $this->balance = 505$
  }
}
```

Thread 2

```
class Bank {
  private int $balance;        500$
  public function deposit($val)    val = 5$
  {
     $temp = $this->balance;    temp = 500$
     $temp = $temp + $val;      temp = 510$
     $this->balance = $temp;    $this->balance = 510$
  }
}
```

Non atomic bank

Not solution!!!

- addition assignment operator is not atomic

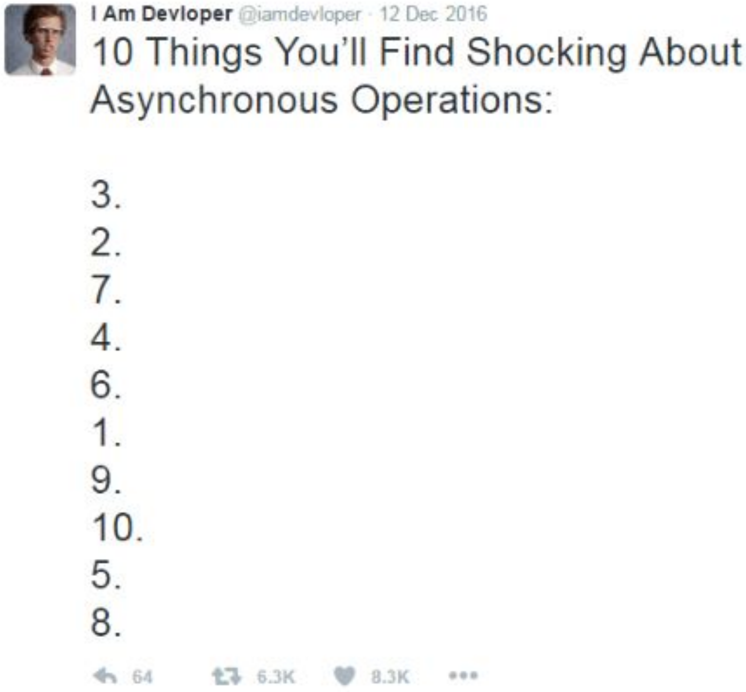
```php
<?php

class Bank {
    private int $balance;
    public function deposit(int $val) {
        $this->balance += $val;
    }
}
```

Non atomic bank

- One of the possible solution is to use locking:

```
class Bank {
   private int $balance;
   public function deposit(int $val) {
       DepositLock::obtain();
       $temp = $this->balance;
       $temp = $temp + $val;
       $this->balance = $temp;
       DepositLock::release();
   }
}
```

Correct synchronization is hard



I Am Devloper @iamdevloper · 12 Dec 2016

10 Things You'll Find Shocking About Asynchronous Operations:

3.
2.
7.
4.
6.
1.
9.
10.
5.
8.

Concurrency - threads vs event loops

Event loops:

- No CPU parallelism.
- ASYNC IO is it’s secret sauce.
- Loop + other abstractions helps handle nasty concurrency issues (synchronization).

Concurrency - threads vs event loops

- Events loops provide reasonable tradeoffs for most common problems in web development.

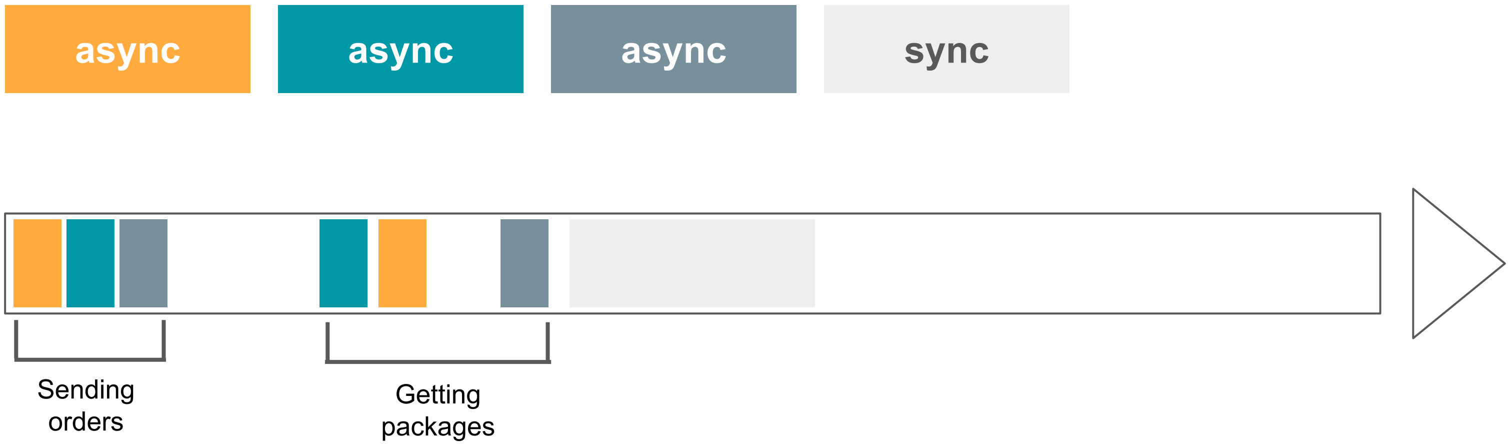
X

- Unless your code is CPU intensive.

But even with event loops there is one issue.

Combine sync and async statements

- You have to combine sync and async statements.



How to handle async statements?

- You cannot obtain their result immediately after execution.
- Callbacks use to be a first choice.
- This lead into infamous callback hell/pyramid of doom.

How to handle async statements?

```php
<?php

shopOrder('motherboard', function($result){
   shopOrder('case', function($result){
       shopOrder('cpu', function($result){
           buildPc();
       });
   });
});
```

Promises

- Same as a promise in real life - it means we are going to do something in the future.
- Helpful abstraction that should reduce callback hell.
- Object that represents callback and allows to chain it with other promises.

Promises

```php
<?php

shopOrder('motherboard')
    ->then(function($result){return shopOrder('case');})
    ->then(function($result){return shopOrder('cpu');})
    ->then(function($result){return buildPc();})
    ->wait();
```

Issue with promises

- You can still cause pyramid of doom with them.

```
shopOrder('motherboard')
   ->then(
       function($result){
           shopOrder('case')->then(
               function($result){
                   shopOrder('cpu')->then(
                       function($result){
                           return buildPc();
                       }
                   );
               }
           );
       }
   )
   ->wait();
```

Async and Await

- Async and Await keywords allows to work with promises in synchronous way.
- No support in PHP.

```
async function shopOrder($part) {}

$motherboard = await shopOrder('motherboard');
$case        = await shopOrder('case');
$cpu         = await shopOrder('cpu');

buildPc($motherboard, $case, $cpu);
```

Callbacks, Promises & async await

- It can be messy...
- Won’t synchronous calling suffice?
- Javascript:
  - reactivity of UI (browsers),
  - fast IO handling (Node.js)

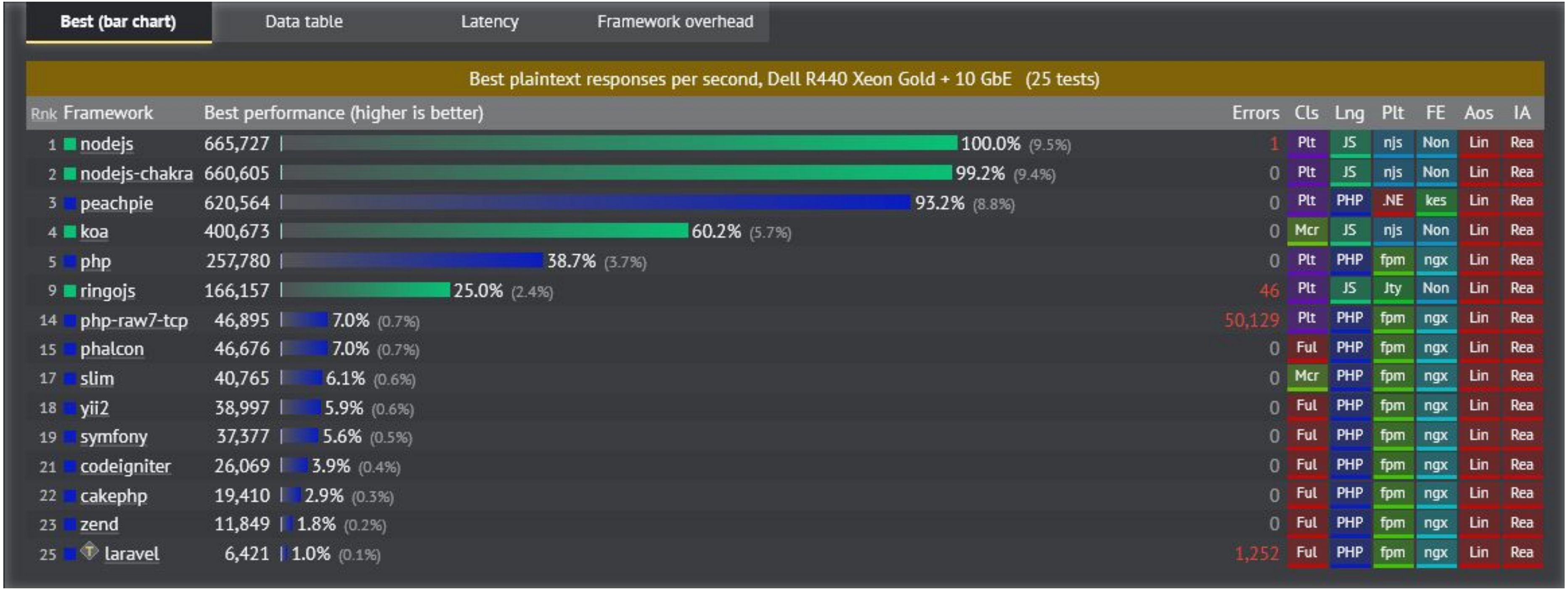
Callbacks, Promises & async await

- In PHP
    - No need for UI reactivity.

X

- Can speed up concurrent request.
- Allows to create asynchronous event driven servers.

Allows to create asynchronous event driven servers?

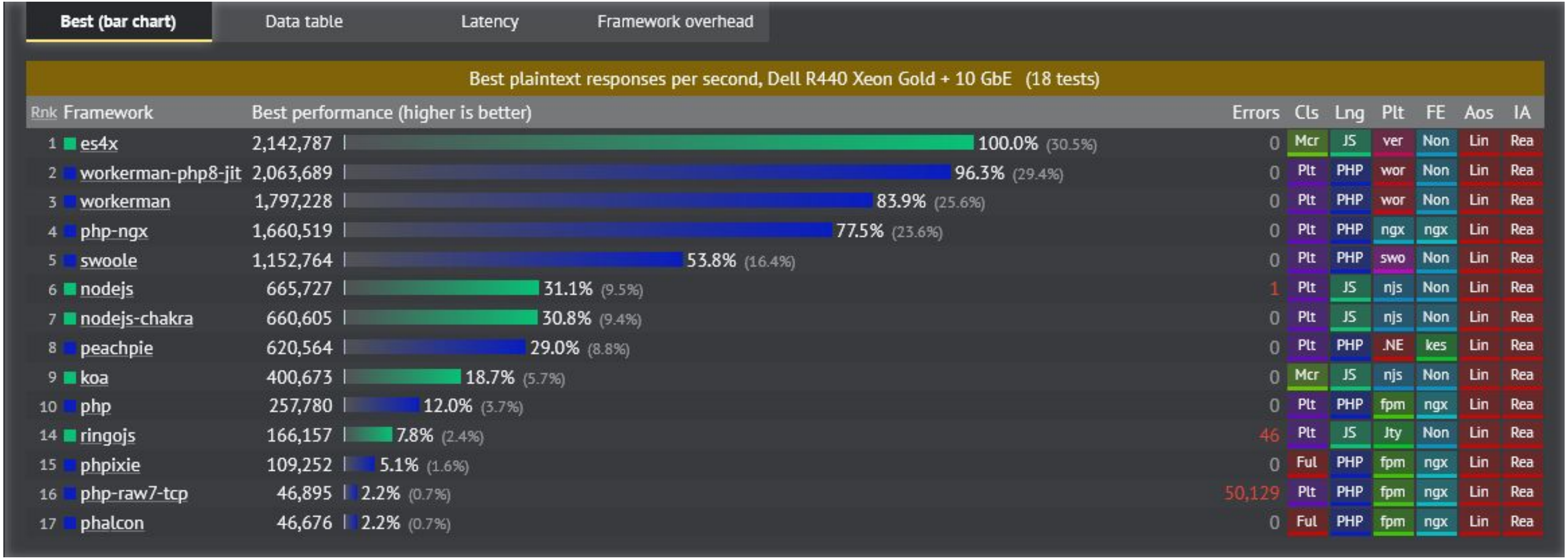


Best plaintext responses per second, Dell R440 Xeon Gold + 10 GbE (25 tests)

Rnk	Framework	Best performance (higher is better)	Errors	Cls	Lng	Plt	FE	Aos	IA
1	🟩 nodejs	665,727 100.0% (9.5%)	1	Plt	JS	njs	Non	Lin	Rea
2	🟩 nodejs-chakra	660,605 99.2% (9.4%)	0	Plt	JS	njs	Non	Lin	Rea
3	🟦 peachpie	620,564 93.2% (8.8%)	0	Plt	PHP	.NE	kes	Lin	Rea
4	🟩 koa	400,673 60.2% (5.7%)	0	Mcr	JS	njs	Non	Lin	Rea
5	🟦 php	257,780 38.7% (3.7%)	0	Plt	PHP	fpm	ngx	Lin	Rea
9	🟩 ringojs	166,157 25.0% (2.4%)	46	Plt	JS	Jty	Non	Lin	Rea
14	🟦 php-raw7-tcp	46,895 7.0% (0.7%)	50,129	Plt	PHP	fpm	ngx	Lin	Rea
15	🟦 phalcon	46,676 7.0% (0.7%)	0	Ful	PHP	fpm	ngx	Lin	Rea
17	🟦 slim	40,765 6.1% (0.6%)	0	Mcr	PHP	fpm	ngx	Lin	Rea
18	🟦 yii2	38,997 5.9% (0.6%)	0	Ful	PHP	fpm	ngx	Lin	Rea
19	🟦 symfony	37,377 5.6% (0.5%)	0	Ful	PHP	fpm	ngx	Lin	Rea
21	🟦 codeigniter	26,069 3.9% (0.4%)	0	Ful	PHP	fpm	ngx	Lin	Rea
22	🟦 cakephp	19,410 2.9% (0.3%)	0	Ful	PHP	fpm	ngx	Lin	Rea
23	🟦 zend	11,849 1.8% (0.2%)	0	Ful	PHP	fpm	ngx	Lin	Rea
25	🟦 laravel	6,421 1.0% (0.1%)	1,252	Ful	PHP	fpm	ngx	Lin	Rea

[source: https://www.techempower.com/benchmarks/]

Allows to create asynchronous event driven servers!

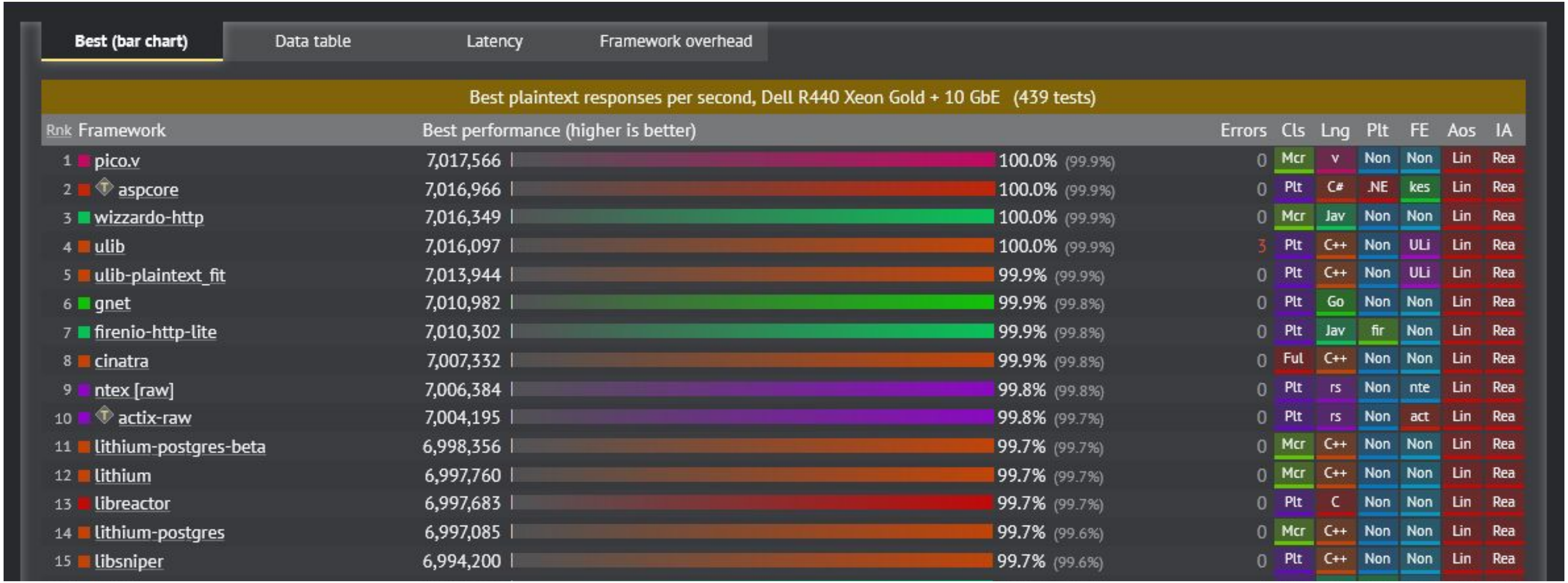


Best plaintext responses per second, Dell R440 Xeon Gold + 10 GbE (18 tests)

Rnk	Framework	Best performance (higher is better)	Errors	Cls	Lng	Plt	FE	Aos	IA
1	🟩 es4x	2,142,787 100.0% (30.5%)	0	Mcr	JS	ver	Non	Lin	Rea
2	🟦 workerman-php8-jit	2,063,689 96.3% (29.4%)	0	Plt	PHP	wor	Non	Lin	Rea
3	🟦 workerman	1,797,228 83.9% (25.6%)	0	Plt	PHP	wor	Non	Lin	Rea
4	🟦 php-ngx	1,660,519 77.5% (23.6%)	0	Plt	PHP	ngx	ngx	Lin	Rea
5	🟦 swoole	1,152,764 53.8% (16.4%)	0	Plt	PHP	swo	Non	Lin	Rea
6	🟩 nodejs	665,727 31.1% (9.5%)	1	Plt	JS	njs	Non	Lin	Rea
7	🟩 nodejs-chakra	660,605 30.8% (9.4%)	0	Plt	JS	njs	Non	Lin	Rea
8	🟦 peachpie	620,564 29.0% (8.8%)	0	Plt	PHP	.NE	kes	Lin	Rea
9	🟩 koa	400,673 18.7% (5.7%)	0	Mcr	JS	njs	Non	Lin	Rea
10	🟦 php	257,780 12.0% (3.7%)	0	Plt	PHP	fpm	ngx	Lin	Rea
14	🟩 ringojs	166,157 7.8% (2.4%)	46	Plt	JS	Jty	Non	Lin	Rea
15	🟦 phpixie	109,252 5.1% (1.6%)	0	Ful	PHP	fpm	ngx	Lin	Rea
16	🟦 php-raw7-tcp	46,895 2.2% (0.7%)	50,129	Plt	PHP	fpm	ngx	Lin	Rea
17	🟦 phalcon	46,676 2.2% (0.7%)	0	Ful	PHP	fpm	ngx	Lin	Rea

[source: https://www.techempower.com/benchmarks/]

Allows to create asynchronous event driven servers?



Best plaintext responses per second, Dell R440 Xeon Gold + 10 GbE (439 tests)

Rnk	Framework	Best performance (higher is better)			Errors	Cls	Lng	Plt	FE	Aos	IA
1	pico.v	7,017,566	100.0%	(99.9%)	0	Mcr	v	Non	Non	Lin	Rea
2	aspcore	7,016,966	100.0%	(99.9%)	0	Plt	C#	.NE	kes	Lin	Rea
3	wizzardo-http	7,016,349	100.0%	(99.9%)	0	Mcr	Jav	Non	Non	Lin	Rea
4	ulib	7,016,097	100.0%	(99.9%)	3	Plt	C++	Non	ULi	Lin	Rea
5	ulib-plaintext_fit	7,013,944	99.9%	(99.9%)	0	Plt	C++	Non	ULi	Lin	Rea
6	gnet	7,010,982	99.9%	(99.8%)	0	Plt	Go	Non	Non	Lin	Rea
7	firenio-http-lite	7,010,302	99.9%	(99.8%)	0	Plt	Jav	fir	Non	Lin	Rea
8	cinatra	7,007,332	99.9%	(99.8%)	0	Ful	C++	Non	Non	Lin	Rea
9	ntex [raw]	7,006,384	99.8%	(99.8%)	0	Plt	rs	Non	nte	Lin	Rea
10	actix-raw	7,004,195	99.8%	(99.7%)	0	Plt	rs	Non	act	Lin	Rea
11	lithium-postgres-beta	6,998,356	99.7%	(99.7%)	0	Mcr	C++	Non	Non	Lin	Rea
12	lithium	6,997,760	99.7%	(99.7%)	0	Mcr	C++	Non	Non	Lin	Rea
13	libreactor	6,997,683	99.7%	(99.7%)	0	Plt	C	Non	Non	Lin	Rea
14	lithium-postgres	6,997,085	99.7%	(99.6%)	0	Mcr	C++	Non	Non	Lin	Rea
15	libsniper	6,994,200	99.7%	(99.6%)	0	Plt	C++	Non	Non	Lin	Rea

[source: https://www.techempower.com/benchmarks/]

What’s role of fibers?

What problems does fibers solve?

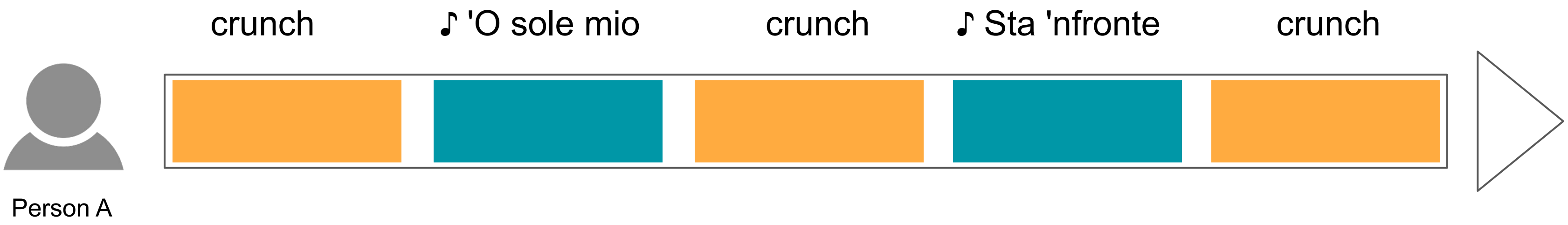
This RFC (Fibers RFC) seeks to eliminate the distinction between synchronous and asynchronous functions by allowing functions to be interruptible without polluting the entire call stack.

[source: https://wiki.php.net/rfc/fibers]

Concurrency - task switching

Eating

Singing



How?

What problems does fibers solve?

- They are like play/stop buttons on video player but for your PHP methods.
- Fibers are created, started, suspended, and terminated by the PHP script itself.
- They are loosely similar to generators.

Generators vs fibers

- Generators allow to “pause” function and return value from function by using keyword yield.

```
function gen_one_to_three() {
    for ($i = 1; $i <= 3; $i++) {
        // Note that $i is preserved between yields.
        yield $i;
    }
}

$generator = gen_one_to_three();
foreach ($generator as $value) {
    echo "$value\n";
}
```

Generators vs fibers

- Generators cannot be easily resumed from yield keyword.

X

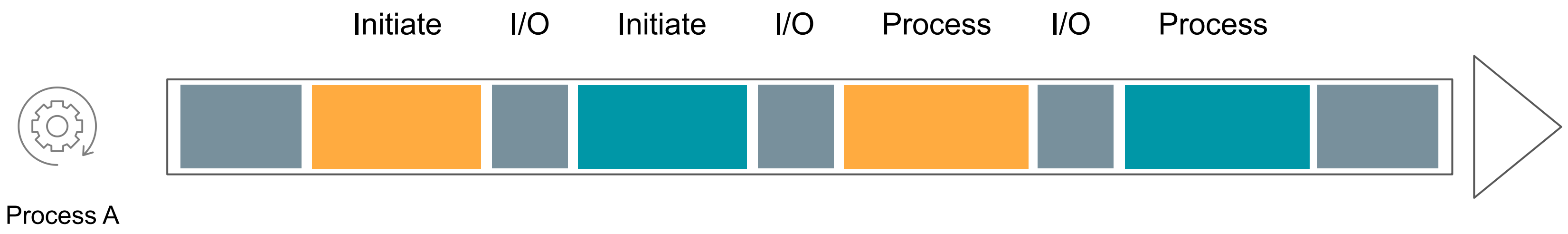
- Fibers can be, but there are few rules:
  - Fiber cannot be resumed by itself.
  - Main flow cannot suspend fiber.

Concurrency - task switching

Fiber DB

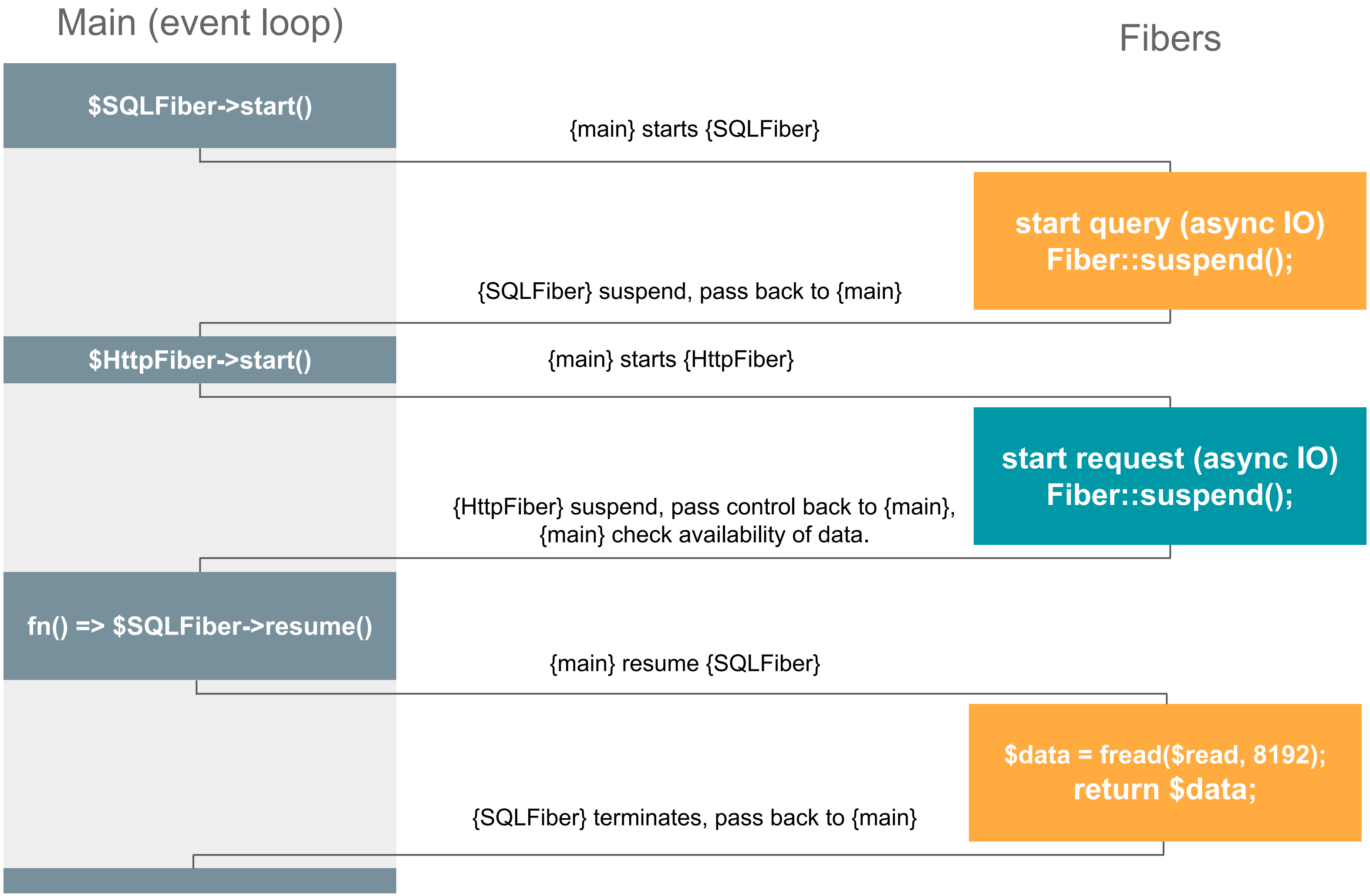
Fiber HTTP

Main flow



Initiate I/O Initiate I/O Process I/O Process

Process A



Main (event loop)

Fibers

$SQLFiber->start()

{main} starts {SQLFiber}

start query (async IO)
Fiber::suspend();

{SQLFiber} suspend, pass back to {main}

$HttpFiber->start()

{main} starts {HttpFiber}

start request (async IO)
Fiber::suspend();

{HttpFiber} suspend, pass control back to {main},
{main} check availability of data.

fn() => $SQLFiber->resume()

{main} resume {SQLFiber}

$data = fread($read, 8192);
return $data;

{SQLFiber} terminates, pass back to {main}

Live example

What problems does Fibers solve?

Does make usage of Fibers code more readable?

- Not alone, they are too low level.
- There is a need for a lot of additional abstraction.

Do they offer async IO out of box?

- You still need support of async IO.

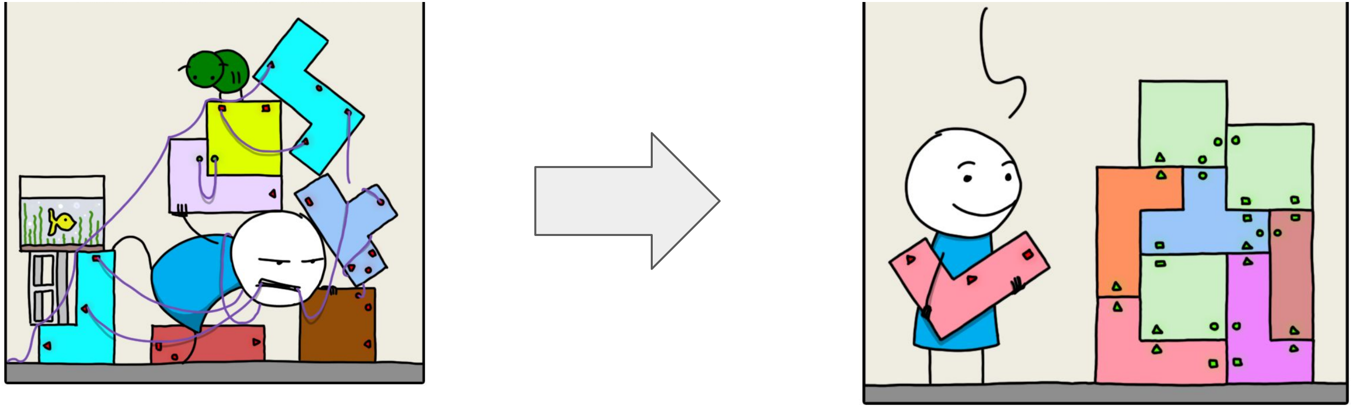
Support across frameworks

Async PHP frameworks are still need it as they provide:

- event loops,
- useful async IO wrappers,
- other abstractions - promises, etc.

Then why we need them?

- They are one of the basic building blocks for proper and maintainable async implementation.



Then why we need them?

Why we need fibers?

- You will probably never directly need them.
- But you will use them indirectly through some of the ASYNC frameworks.

Support across frameworks

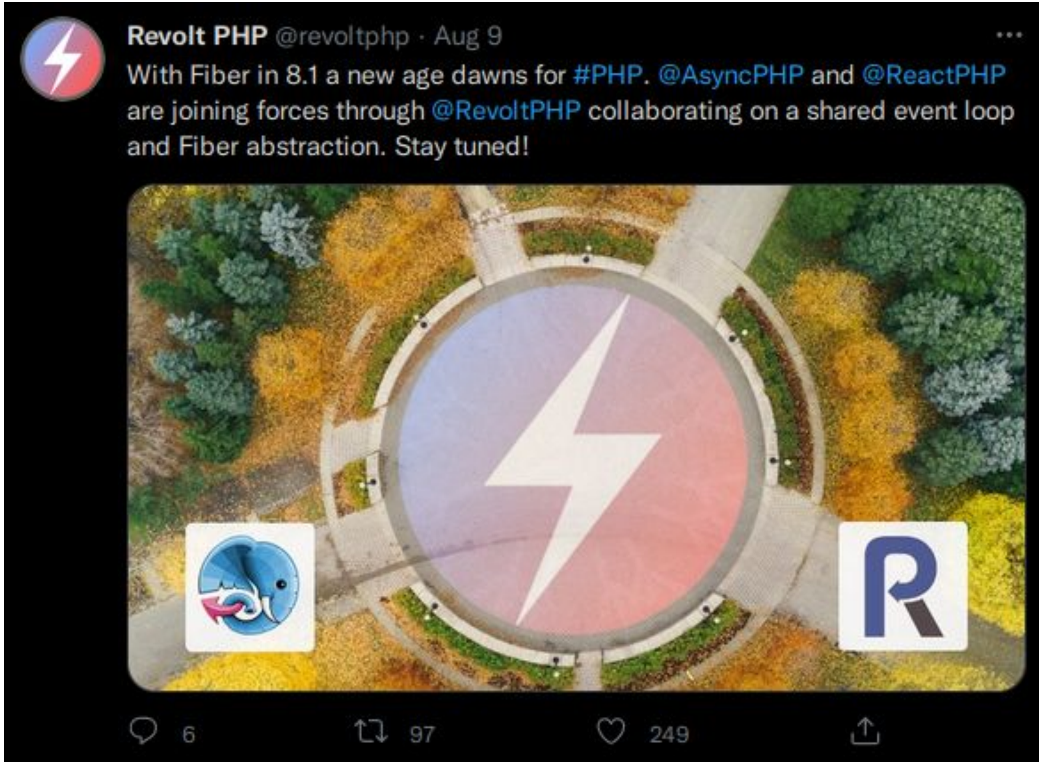
- AMP supports fibers in V3
  - https://github.com/amphp/amp/tree/v3
- React - POC in:
  - https://github.com/trowski/react-fiber
- Swoole - no support.

Live example

Future?

- Event loop in PHP core?
- Async/Await in PHP?
- Revolt PHP?

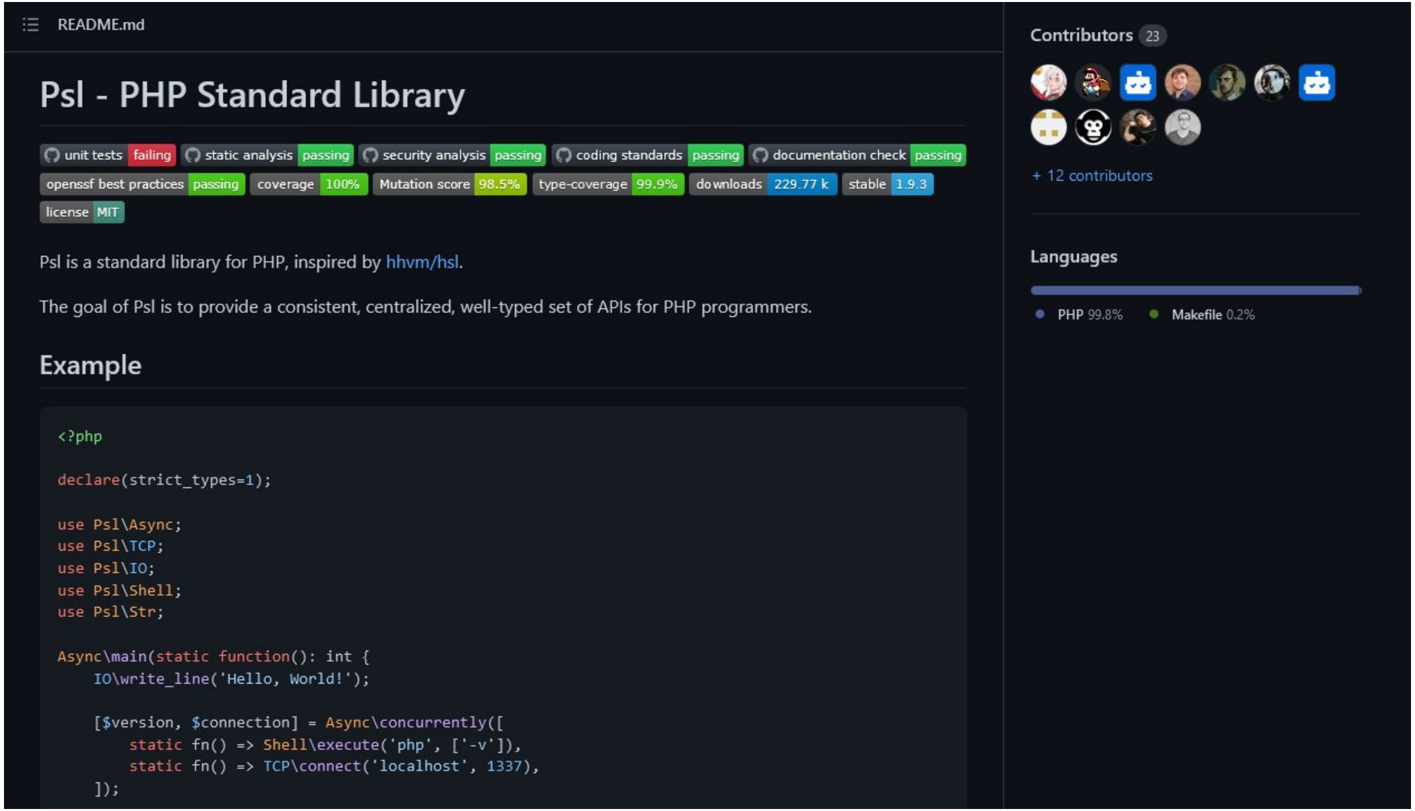
Future?



Revolt PHP @revoltphp · Aug 9

With Fiber in 8.1 a new age dawns for #PHP. @AsyncPHP and @ReactPHP are joining forces through @RevoltPHP collaborating on a shared event loop and Fiber abstraction. Stay tuned!

First foray!



README.md

Psl - PHP Standard Library

unit tests failing | static analysis passing | security analysis passing | coding standards passing | documentation check passing | openssf best practices passing | coverage 100% | Mutation score 98.5% | type-coverage 99.9% | downloads 229.77 k | stable 1.9.3 | license MIT

Psl is a standard library for PHP, inspired by hhvm/hsl.

The goal of Psl is to provide a consistent, centralized, well-typed set of APIs for PHP programmers.

Example

```php
<?php

declare(strict_types=1);

use Psl\Async;
use Psl\TCP;
use Psl\IO;
use Psl\Shell;
use Psl\Str;

Async\main(static function(): int {
    IO\write_line('Hello, World!');

    [$version, $connection] = Async\concurrently([
        static fn() => Shell\execute('php', ['-v']),
        static fn() => TCP\connect('localhost', 1337),
    ]);
```

Contributors 23

+ 12 contributors

Languages

● PHP 99.8% ● Makefile 0.2%

Psl - PHP Standard Library

- Async IO out of box and much more!

```
Async\main(static function(): int {
   IO\write_line('Hello, World!');

   [$version, $connection] = Async\concurrently([
       static fn() => Shell\execute('php', ['-v']),
       static fn() => TCP\connect('localhost', 1337),
   ]);

   $messages = Str\split($version, "\n");
   foreach($messages as $message) {
       $connection->writeAll($message);
   }

   $connection->close();

   return 0;
});
```

Questions?

RTsoft
Softwarová řešení na míru

Informační systémy na míru

Business analytika

Mobilní aplikace

Weby a E-shopy

REST API best practices

{ REST API }

25. května 2022

18:00 hod