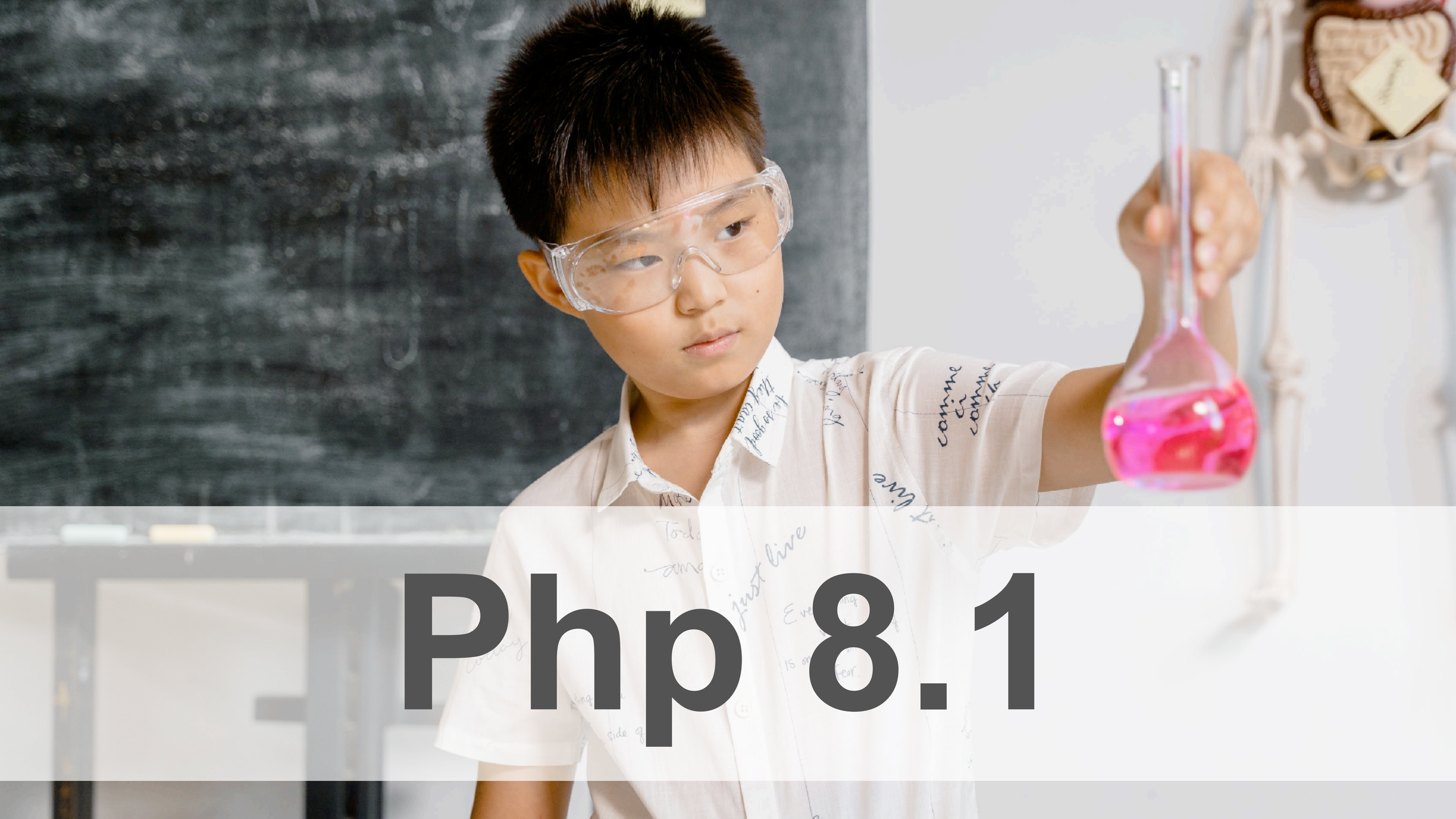
Fiber

The open door to asynchronous

Benoit Viguier
@b_viguier

FORUMPHP
PARIS2021



Php 8.1

Benoit Viguier

Backend Principal Engineer

@b_viguier

BEDROCK

Creating Streaming Champions

www.bedrockstreaming.com

We are hiring!

Fiber RFC

Construction

```
final class Fiber
{

 public function __construct(callable $callback) {}

 public function start(mixed ...$args): mixed {}

 public function getReturn(): mixed {}

 //…
}
```

Hello World

```php
function foo($arg): int
{
    echo "$arg\n";
    return 0;
}

$fiber = new \Fiber('foo');
$fiber->start('Hello World');    // Hello World
echo $fiber->getReturn();        // 0
```

Hello World

```php
function foo($arg): int
{
    echo "$arg\n";
    return 0;
}

$fiber = new \Fiber('foo');
$fiber->start('Hello World');    // Hello World
echo $fiber->getReturn();        // 0
```

Hello World

```php
function foo($arg): int
{
    echo "$arg\n";
    return 0;
}

$fiber = new \Fiber('foo');
$fiber->start('Hello World');    // Hello World
echo $fiber->getReturn();        // 0
```

Hello World

```php
function foo($arg): int
{
    echo "$arg\n";
    return 0;
}

$fiber = new \Fiber('foo');
$fiber->start('Hello World');    // Hello World
echo $fiber->getReturn();        // 0
```

Hello World

```php
function foo($arg): int
{
    echo "$arg\n";
    return 0;
}

$fiber = new \Fiber('foo');
$fiber->start('Hello World');    // Hello World
echo $fiber->getReturn();        // 0
```

Hello World

```php
function foo($arg): int
{
    echo "$arg\n";
    return 0;
}

$fiber = new \Fiber('foo');
$fiber->start('Hello World');    // Hello World
echo $fiber->getReturn();        // 0
```

Status

```
final class Fiber
{
 public function isStarted(): bool {}

 public function isRunning(): bool {}

 public function isTerminated(): bool {}

 //…
}
```

Show status

```php
function status(\Fiber $fiber): void
{
    echo "Status: ";
    if ($fiber->isStarted()) echo "Started ";
    if ($fiber->isRunning()) echo "Running ";
    if ($fiber->isTerminated()) echo "Terminated";
    echo PHP_EOL;
}

$fiber = new \Fiber(status(...));
status($fiber);             // Status:
$fiber->start($fiber);      // Status: Started Running
status($fiber);             // Status: Started Terminated
```

Show status

```php
function status(\Fiber $fiber): void
{
    echo "Status: ";
    if ($fiber->isStarted()) echo "Started ";
    if ($fiber->isRunning()) echo "Running ";
    if ($fiber->isTerminated()) echo "Terminated";
    echo PHP_EOL;
}

$fiber = new \Fiber(status(...));
status($fiber);             // Status:
$fiber->start($fiber);      // Status: Started Running
status($fiber);             // Status: Started Terminated
```

Show status

```php
function status(\Fiber $fiber): void
{
    echo "Status: ";
    if ($fiber->isStarted()) echo "Started ";
    if ($fiber->isRunning()) echo "Running ";
    if ($fiber->isTerminated()) echo "Terminated";
    echo PHP_EOL;
}

$fiber = new \Fiber(status(...));
status($fiber);             // Status:
$fiber->start($fiber);      // Status: Started Running
status($fiber);             // Status: Started Terminated
```

Show status

```php
function status(\Fiber $fiber): void
{
    echo "Status: ";
    if ($fiber->isStarted()) echo "Started ";
    if ($fiber->isRunning()) echo "Running ";
    if ($fiber->isTerminated()) echo "Terminated";
    echo PHP_EOL;
}
```

New! 😍

```php
$fiber = new \Fiber(status(...));
status($fiber);              // Status:
$fiber->start($fiber);       // Status: Started Running
status($fiber);              // Status: Started Terminated
```

Show status

```php
function status(\Fiber $fiber): void
{
    echo "Status: ";
    if ($fiber->isStarted()) echo "Started ";
    if ($fiber->isRunning()) echo "Running ";
    if ($fiber->isTerminated()) echo "Terminated";
    echo PHP_EOL;
}

$fiber = new \Fiber(status(...));
status($fiber);                 // Status:
$fiber->start($fiber);          // Status: Started Running
status($fiber);                 // Status: Started Terminated
```

Show status

```php
function status(\Fiber $fiber): void
{
    echo "Status: ";
    if ($fiber->isStarted()) echo "Started ";
    if ($fiber->isRunning()) echo "Running ";
    if ($fiber->isTerminated()) echo "Terminated";
    echo PHP_EOL;
}

$fiber = new \Fiber(status(...));
status($fiber);                 // Status:
$fiber->start($fiber);          // Status: Started Running
status($fiber);                 // Status: Started Terminated
```

Show status

```php
function status(\Fiber $fiber): void
{
    echo "Status: ";
    if ($fiber->isStarted()) echo "Started ";
    if ($fiber->isRunning()) echo "Running ";
    if ($fiber->isTerminated()) echo "Terminated";
    echo PHP_EOL;
}

$fiber = new \Fiber(status(...));
status($fiber);              // Status:
$fiber->start($fiber);       // Status: Started Running
status($fiber);              // Status: Started Terminated
```

Show status

```php
function status(\Fiber $fiber): void
{
    echo "Status: ";
    if ($fiber->isStarted()) echo "Started ";
    if ($fiber->isRunning()) echo "Running ";
    if ($fiber->isTerminated()) echo "Terminated";
    echo PHP_EOL;
}

$fiber = new \Fiber(status(...));
status($fiber);             // Status:
$fiber->start($fiber);      // Status: Started Running
status($fiber);             // Status: Started Terminated
```

Interruptions

```
final class Fiber
{
 public static function suspend(mixed $value = null): mixed {}

 public function isSuspended(): bool {}

 public function resume(mixed $value = null): mixed {}

 //…
}
```

Updated status function

```php
function status(\Fiber $fiber): void
{
    echo "Status: ";
    if ($fiber->isStarted()) echo "Started ";
    if ($fiber->isRunning()) echo "Running ";
    if ($fiber->isSuspended()) echo "Suspended ";
    if ($fiber->isTerminated()) echo "Terminated";
    echo PHP_EOL;
}
```

Updated status function

```php
function status(\Fiber $fiber): void
{
    echo "Status: ";
    if ($fiber->isStarted()) echo "Started ";
    if ($fiber->isRunning()) echo "Running ";
    if ($fiber->isSuspended()) echo "Suspended ";
    if ($fiber->isTerminated()) echo "Terminated";
    echo PHP_EOL;
}
```

Interrupted

```php
$fiber = new \Fiber(function (): void
{
    echo "before\n";
    echo \Fiber::suspend('hello');
    echo "after\n";
});

status($fiber);                 // Status:
$value = $fiber->start();       // before
status($fiber);                 // Status: Started Suspended
$fiber->resume(
    "$value world\n"            // hello world
);                              // after
status($fiber);                 // Status: Started Terminated
```

Interrupted

```php
$fiber = new \Fiber(function (): void
{
    echo "before\n";
    echo \Fiber::suspend('hello');
    echo "after\n";
});

status($fiber);                 // Status:
$value = $fiber->start();       // before
status($fiber);                 // Status: Started Suspended
$fiber->resume(
    "$value world\n"            // hello world
);                              // after
status($fiber);                 // Status: Started Terminated
```

Interrupted

```php
$fiber = new \Fiber(function (): void
{
    echo "before\n";
    echo \Fiber::suspend('hello');
    echo "after\n";
});

status($fiber);                 // Status:
$value = $fiber->start();       // before
status($fiber);                 // Status: Started Suspended
$fiber->resume(
    "$value world\n"            // hello world
);                              // after
status($fiber);                 // Status: Started Terminated
```

Interrupted

```php
$fiber = new \Fiber(function (): void
{
    echo "before\n";
    echo \Fiber::suspend('hello');
    echo "after\n";
});

status($fiber);                 // Status:
$value = $fiber->start();       // before
status($fiber);                 // Status: Started Suspended
$fiber->resume(
    "$value world\n"            // hello world
);                              // after
status($fiber);                 // Status: Started Terminated
```

Interrupted

```php
$fiber = new \Fiber(function (): void
{
    echo "before\n";
    echo \Fiber::suspend('hello');
    echo "after\n";
});

status($fiber);                 // Status:
$value = $fiber->start();       // before
status($fiber);                 // Status: Started Suspended
$fiber->resume(
    "$value world\n"            // hello world
);                              // after
status($fiber);                 // Status: Started Terminated
```

Interrupted

```php
$fiber = new \Fiber(function (): void
{
    echo "before\n";
    echo \Fiber::suspend('hello');
    echo "after\n";
});

status($fiber);                 // Status:
$value = $fiber->start();       // before
status($fiber);                 // Status: Started Suspended
$fiber->resume(
    "$value world\n"            // hello world
);                              // after
status($fiber);                 // Status: Started Terminated
```

Interrupted

```php
$fiber = new \Fiber(function (): void
{
    echo "before\n";
    echo \Fiber::suspend('hello');
    echo "after\n";
});

status($fiber);                 // Status:
$value = $fiber->start();       // before
status($fiber);                 // Status: Started Suspended
$fiber->resume(
    "$value world\n"            // hello world
);                              // after
status($fiber);                 // Status: Started Terminated
```

Interrupted

```php
$fiber = new \Fiber(function (): void
{
    echo "before\n";
    echo \Fiber::suspend('hello');
    echo "after\n";
});

status($fiber);                 // Status:
$value = $fiber->start();       // before
status($fiber);                 // Status: Started Suspended
$fiber->resume(
    "$value world\n"            // hello world
);                              // after
status($fiber);                 // Status: Started Terminated
```

Interrupted

```php
$fiber = new \Fiber(function (): void
{
    echo "before\n";
    echo \Fiber::suspend('hello');
    echo "after\n";
});

status($fiber);                 // Status:
$value = $fiber->start();       // before
status($fiber);                 // Status: Started Suspended
$fiber->resume(
    "$value world\n"            // hello world
);                              // after
status($fiber);                 // Status: Started Terminated
```

Interrupted

```php
$fiber = new \Fiber(function (): void
{
    echo "before\n";
    echo \Fiber::suspend('hello');
    echo "after\n";
});

status($fiber);                 // Status:
$value = $fiber->start();       // before
status($fiber);                 // Status: Started Suspended
$fiber->resume(
    "$value world\n"            // hello world
);                              // after
status($fiber);                 // Status: Started Terminated
```

Current Fiber

```
final class Fiber
{
 public static function getCurrent(): ?self {}

 //…
}
```

Here we are

```php
function foo(): void
{
    if(\Fiber::getCurrent() !== null) {
        echo "Fiber\n";
    } else {
        echo "Main\n";
    }
}

(new \Fiber(foo(...)))->start();     // Fiber
foo();                               // Main
```

Here we are

```php
function foo(): void
{
    if(\Fiber::getCurrent() !== null) {
        echo "Fiber\n";
    } else {
        echo "Main\n";
    }
}

(new \Fiber(foo(...)))->start();      // Fiber
foo();                                // Main
```

Here we are

```php
function foo(): void
{
    if(\Fiber::getCurrent() !== null) {
        echo "Fiber\n";
    } else {
        echo "Main\n";
    }
}

(new \Fiber(foo(...)))->start();      // Fiber
foo();                                // Main
```

Here we are

```php
function foo(): void
{
    if(\Fiber::getCurrent() !== null) {
        echo "Fiber\n";
    } else {
        echo "Main\n";
    }
}

(new \Fiber(foo(...)))->start();     // Fiber
foo();                               // Main
```

Reflection, Errors, Throw

Single threaded

Single threaded

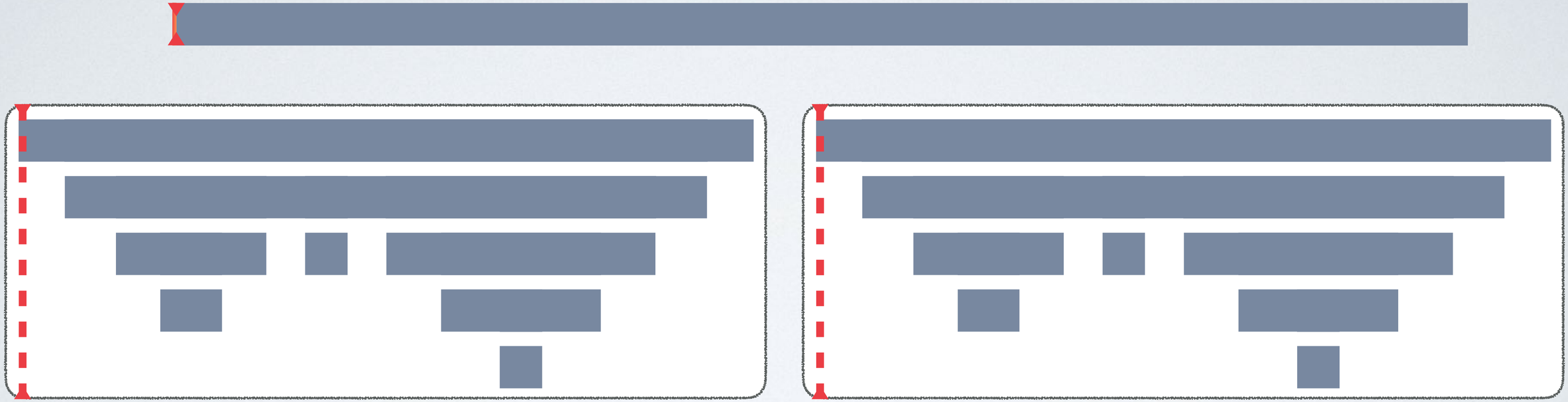
Self interrupted

What for?

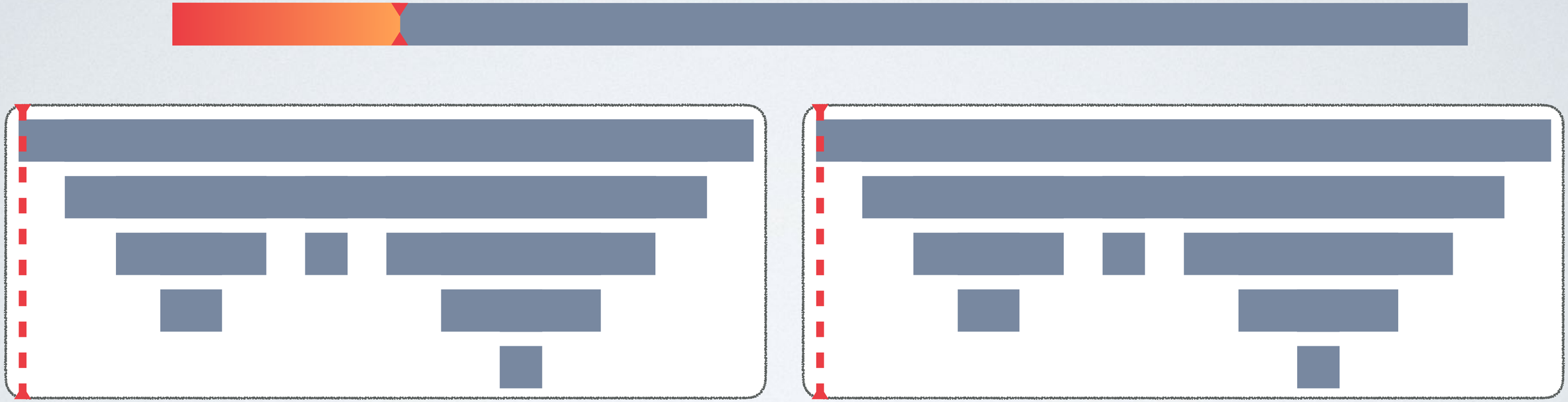
Asynchronous

The perfect match

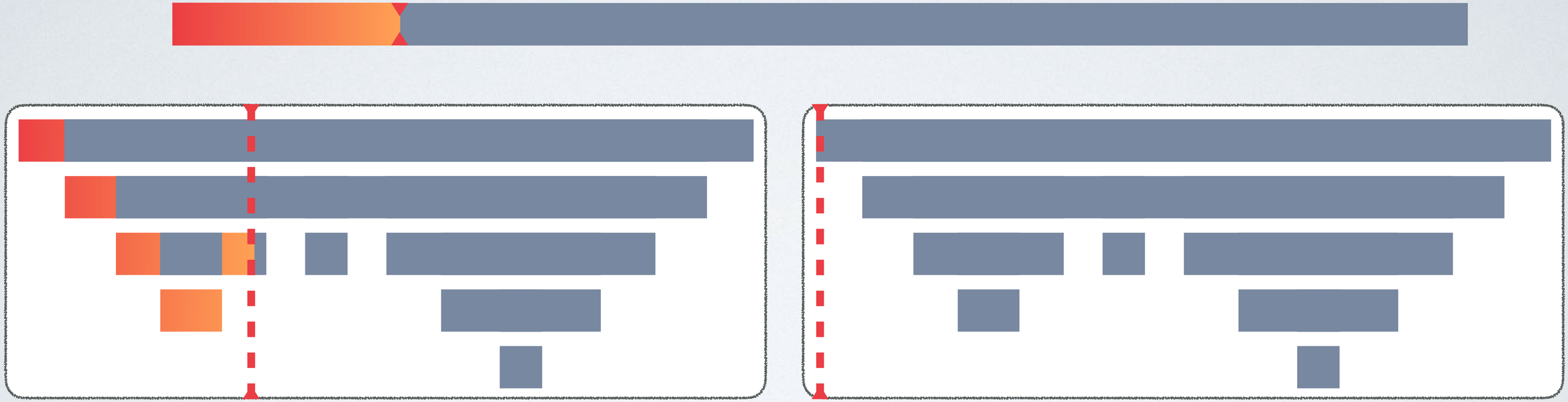
Interruptible call stack



Interruptible call stack



Interruptible call stack



Interruptible call stack

Interruptible call stack

Interruptible call stack

Interruptible call stack

Interruptible call stack

Interruptible call stack

Interruptible call stack

Interruptible call stack

Example

Nano framework for asynchronous `sleep`

Slip

Nano framework for asynchronous `sleep`

https://github.com/b-viguier/Slip

Slip Framework

```php
namespace bviguier\Slip;

function sleep(int $seconds): void {}

class Slipper
{
  public function __construct(callable $callable, mixed ...$args)
  {}


  static public function run(Slipper ...$slips): array {}
}
```

Slip Framework

```php
namespace bviguier\Slip;

function sleep(int $seconds): void {}

class Slipper
{
    public function __construct(callable $callable, mixed ...$args)
    {}


    static public function run(Slipper ...$slips): array {}
}
```

Slip\sleep

```
function sleep(int $seconds): void
{
    if (\Fiber::getCurrent() !== null) {
        \Fiber::suspend($seconds);
    } else {
        \sleep($seconds);
    }
}
```

Slip\sleep

```
function sleep(int $seconds): void
{
    if (\Fiber::getCurrent() !== null) {
        \Fiber::suspend($seconds);
    } else {
        \sleep($seconds);
    }
}
```

Slip\sleep

```
function sleep(int $seconds): void
{
    if (\Fiber::getCurrent() !== null) {
        \Fiber::suspend($seconds);
    } else {
        \sleep($seconds);
    }
}
```

Slip\Slipper

```
class Slipper
{
 private \Fiber $fiber;
 private int $timeout;

 public function __construct(callable $callable, mixed ...$args)
 {
     $this->fiber = new \Fiber($callable);
     $this->timeout = $this->fiber->start(...$args) ?? 0;
 }

 // …
}
```

Slip\Slipper

```
class Slipper
{
 private \Fiber $fiber;
 private int $timeout;

 public function __construct(callable $callable, mixed ...$args)
 {
      $this->fiber = new \Fiber($callable);
      $this->timeout = $this->fiber->start(...$args) ?? 0;
 }

 // …
}
```

Slip\Slipper

```
class Slipper
{
 private \Fiber $fiber;
 private int $timeout;

 public function __construct(callable $callable, mixed ...$args)
 {
      $this->fiber = new \Fiber($callable);
      $this->timeout = $this->fiber->start(...$args) ?? 0;
 }

 // …
}
```

Slip\Slipper::run

```php
static public function run(Slipper ...$slippers): array
{
    // Init
    while ($slippers) {
        // 1/
        // 2/
        // 3/
        // 4/
        // 5/
    }

    return $results;
}
```

Slip\Slipper::run

```
static public function run(Slipper ...$slippers): array
{
    // Init
    while ($slippers) {
        // 1/
        // 2/
        // 3/
        // 4/
        // 5/
    }

    return $results;
}
```

Slip\Slipper::run

```php
static public function run(Slipper ...$slippers): array
{
    // Init
    while ($slippers) {
        // 1/ Sorts Sleepers according to their timeout.
        // 2/
        // 3/
        // 4/
        // 5/
    }

    return $results;
}
```

Slip\Slipper::run

```php
static public function run(Slipper ...$slippers): array
{
    // Init
    while ($slippers) {
        // 1/ Sorts Sleepers according to their timeout.
        // 2/ Pops the first element.
        // 3/
        // 4/
        // 5/
    }

    return $results;
}
```

Slip\Slipper::run

```php
static public function run(Slipper ...$slippers): array
{
    // Init
    while ($slippers) {
        // 1/ Sorts Sleepers according to their timeout.
        // 2/ Pops the first element.
        // 3/ Is Fiber terminated? ->getReturn() & continue
        // 4/
        // 5/
    }

    return $results;
}
```

Slip\Slipper::run

```php
static public function run(Slipper ...$slippers): array
{
    // Init
    while ($slippers) {
        // 1/ Sorts Sleepers according to their timeout.
        // 2/ Pops the first element.
        // 3/ Is Fiber terminated? ->getReturn() & continue
        // 4/ Else, sleeps until next timeout.
        // 5/
    }

    return $results;
}
```

Slip\Slipper::run

```php
static public function run(Slipper ...$slippers): array
{
    // Init
    while ($slippers) {
        // 1/ Sorts Sleepers according to their timeout.
        // 2/ Pops the first element.
        // 3/ Is Fiber terminated? ->getReturn() & continue
        // 4/ Else, sleeps until next timeout.
        // 5/ Resumes/Appends fiber, update Slipper's timeout.
    }

    return $results;
}
```

Ping-Pong

Ping

```
function ping(int $count): string
{

    while ($count--) {
        echo "Ping\n";
        Slip\sleep(2);
    }

    return "Finished Ping\n";
}
```

Pong

```
function pong(int $count): string
{
    Slip\sleep(1);
    while ($count--) {
        echo "Pong\n";
        Slip\sleep(2);
    }

    return "Finished Pong\n";
}
```

Synchronous...

```
echo ping(2);
// Ping
// Ping
// Finished Ping
echo pong(2);
// Pong
// Pong
// Finished Pong
```

Asynchronous!

```
echo join(
    Slip\Slipper::run(
        new Slip\Slipper(ping(...), 2),
        new Slip\Slipper(pong(...), 2),
    )
);
// Ping
// Pong
// Ping
// Pong
// Finished Ping
// Finished Pong
```

Ping-Pong

🏓 with Symfony 🏓

Ping Command

```
protected function execute(InputInterface $input, OutputInterface $output): int
{
    $count = $input->getArgument('count');


    while ($count--) {
        $output->writeln("Ping");
        Slip\sleep(2);
    }
    $output->writeln("Finished Ping");

    return Command::SUCCESS;
}
```

Pong Command

```php
protected function execute(InputInterface $input, OutputInterface $output): int
{
    $count = $input->getArgument('count');

    Slip\sleep(1);
    while ($count--) {
        $output->writeln("Pong");
        Slip\sleep(2);
    }
    $output->writeln("Finished Pong");

    return Command::SUCCESS;
}
```

Pong Command

```
## php console.php ping 2
Ping
Ping
Finished Ping

## php console.php pong 2
Pong
Pong
Finished Pong
```

PingPong Console

```
function runCommand(string $className): void
{
    $application = new Application();
    $application->add($command = new $className());
    $application->setDefaultCommand(
        $command->getName(),
        true
    );
    $application->setAutoExit(false);
    $application->run();
}
```

PingPong Console

```
function runCommand(string $className): void
{
    $application = new Application();
    $application->add($command = new $className());
    $application->setDefaultCommand(
        $command->getName(),
        true
    );
    $application->setAutoExit(false);
    $application->run();
}
```

PingPong Console

```
function runCommand(string $className): void
{
    $application = new Application();
    $application->add($command = new $className());
    $application->setDefaultCommand(
        $command->getName(),
        true
    );
    $application->setAutoExit(false);
    $application->run();
}
```

PingPong Console

```
Slip\Slipper::run(
    new bviguier\Slip\Slipper(
        runCommand(...),
        PingCommand::class
    ),
    new bviguier\Slip\Slipper(
        runCommand(...),
        PongCommand::class
    ),
);
```

PingPong Console

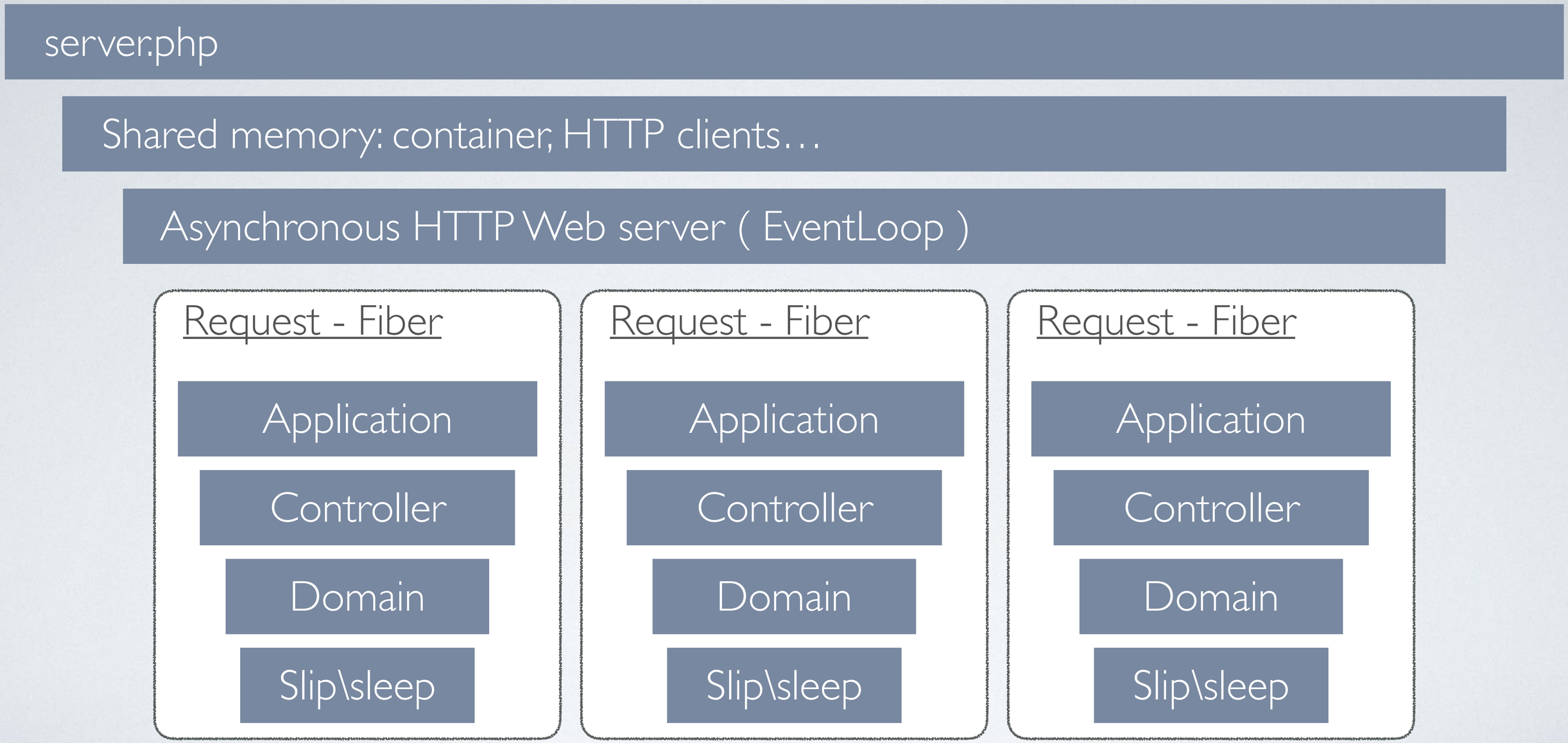
```
Slip\Slipper::run(
    new bviguier\Slip\Slipper(
        runCommand(...),
        PingCommand::class
    ),
    new bviguier\Slip\Slipper(
        runCommand(...),
        PongCommand::class
    ),
);
```

Pong Command

```
## /php ping-pong.php 2
Ping
Pong
Ping
Pong
Finished Ping
Finished Pong
```

Web Server

The other way


server.php
Shared memory: container, HTTP clients…
Asynchronous HTTP Web server (EventLoop)
Request - Fiber
Application
Controller
Domain
Slip\sleep
Request - Fiber
Application
Controller
Domain
Slip\sleep
Request - Fiber
Application
Controller
Domain
Slip\sleep

What's next?

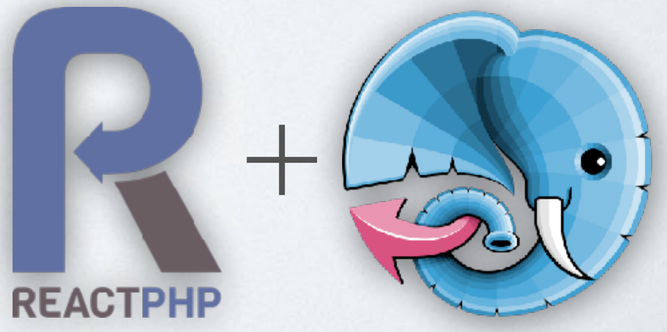
Standard

PSR ? 🤞

RevoltPhp

Shared event loop and Fiber abstraction

revolt.run



Fiber Ready Libraries

Compatibility

Adoption

It's up to you!

? Core feature ?

Non-blocking API

Asynchronous code !== Synchronous code

Leaky Abstraction concept

Fiber

The open door to asynchronous

Benoit Viguier
@b_viguier

FORUMPHP
PARIS2021

Thank you!

The open door to asynchronous

Benoit Viguier
@b_viguier

FORUMPHP
PARIS2021

BEDROCK

Creating Streaming Champions

www.bedrockstreaming.com

We are hiring!

@b_viguier