Introduction to Async PHP with Swoole

Juan Carlos Morales | Dev-Night

08 August 2020

https://dev-night.io/

About me:

Name: Juan Carlos Morales

Origin: Argentina

Work at: Software Developer at Tradebyte GmbH - Germany

Contact:

Email: jcmargentina@gmail.com

LinkedIn: https://www.linkedin.com/in/jcmcvm-en/

Agenda for today

- Concurrency
- Coroutines
    - Introduction
    - Coroutines in bare-bone PHP
- Swoole
    - Introduction and more herbs
    - Code samples
- Live experiment (anything can happen here)

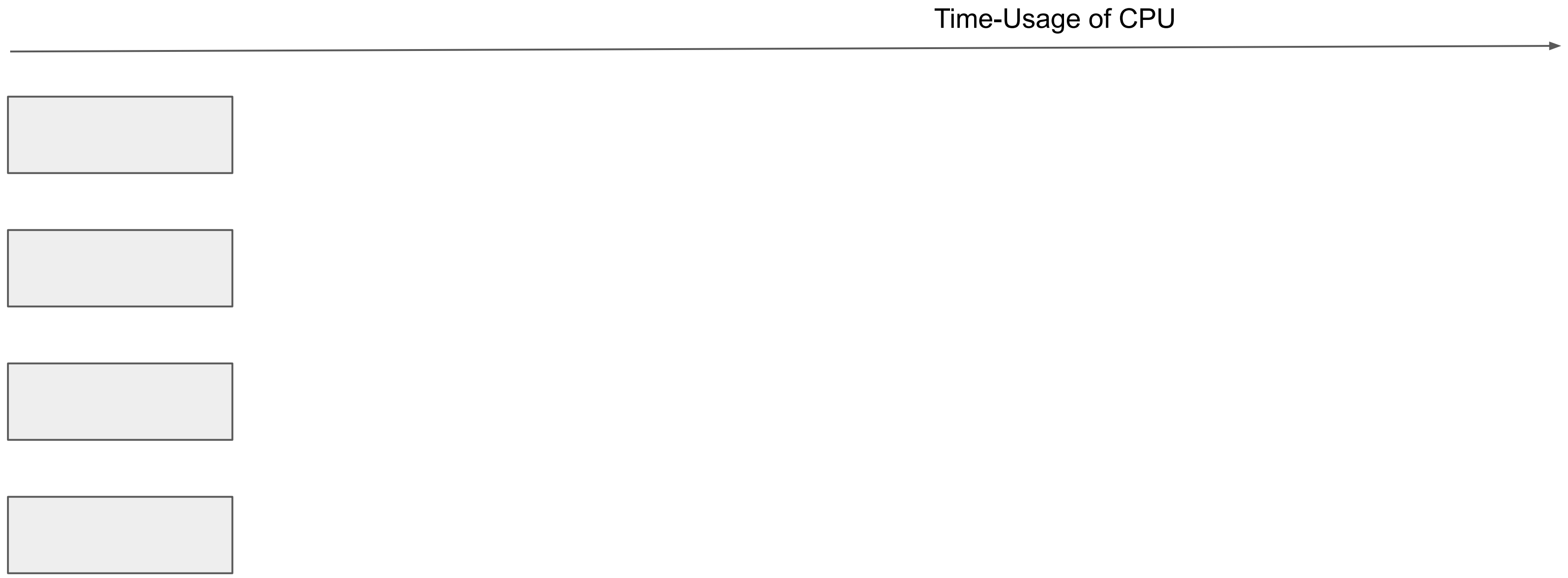
Concurrency

In short terms … and simple human language:

Concurrency is the ability of an Operating System to share the CPU time of use, to a group of processes/threads, executing themselves by “parts”, without affecting the final result of their execution. In a single core computer, gives the illusion of parallelism.

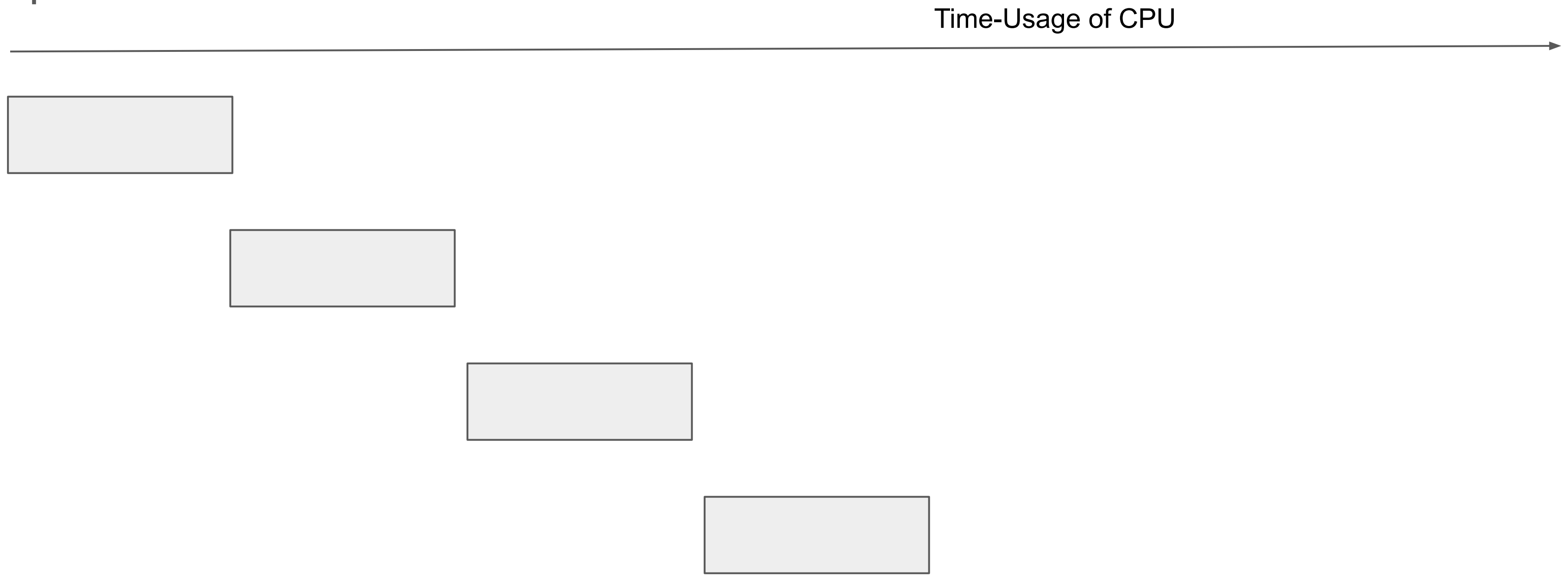
Concurrency

Parallel execution



Concurrency

Sequential execution

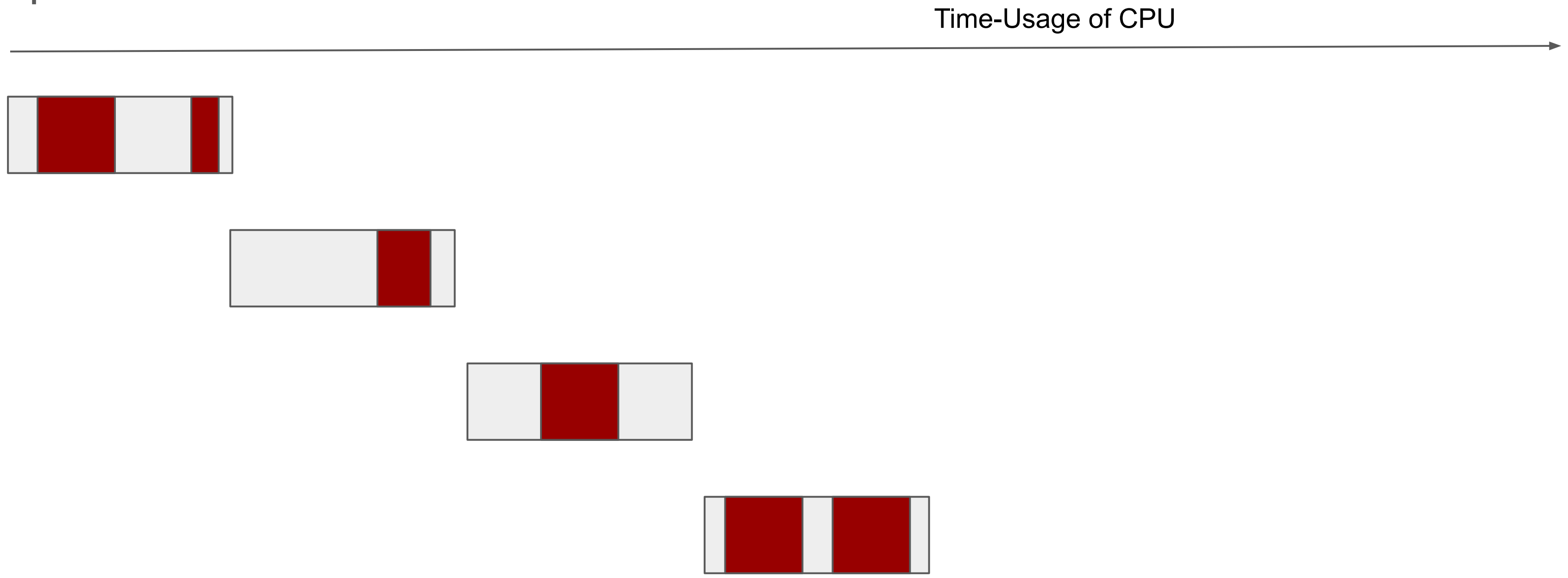


Concurrency

🟥 Idle Time

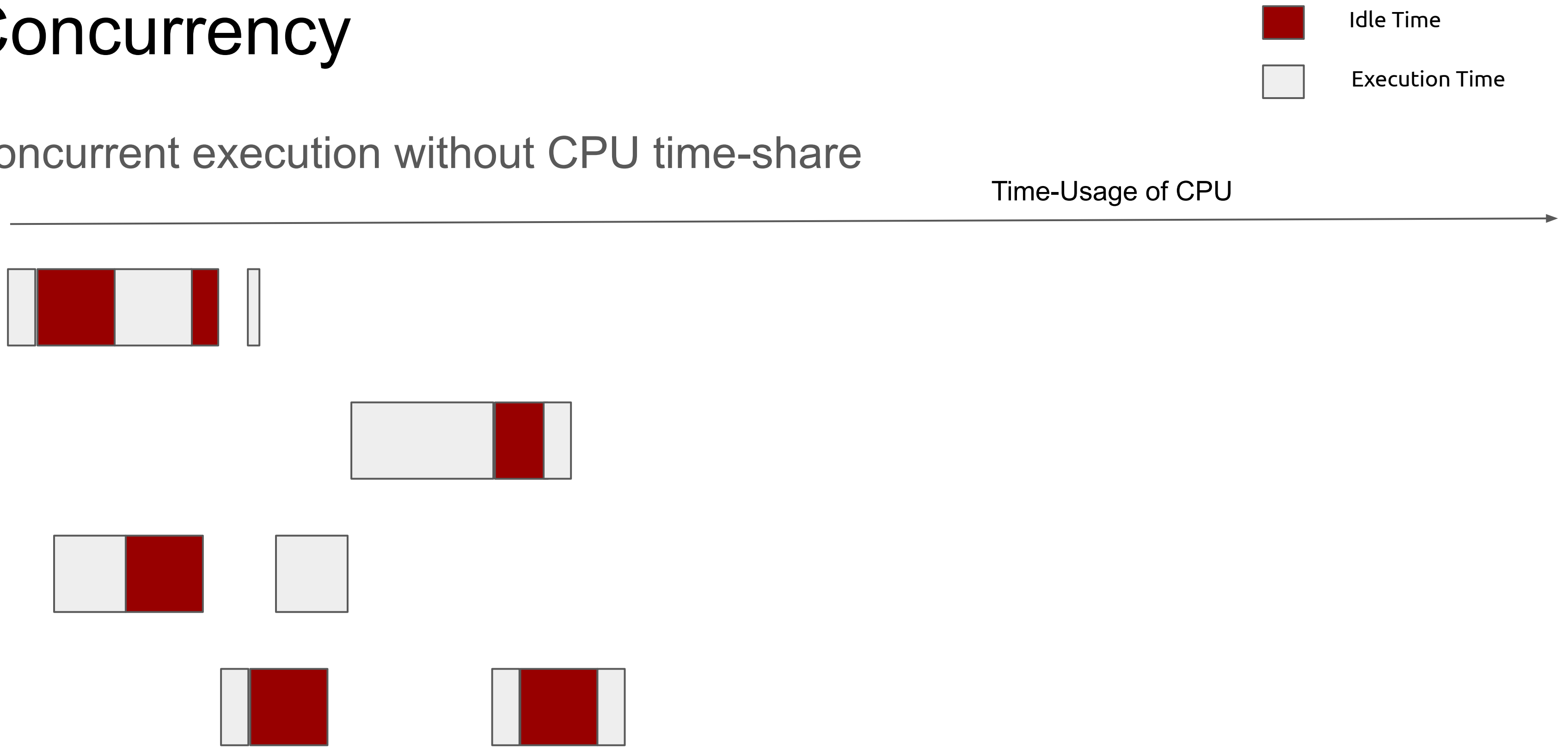
⬜ Execution Time

Sequential execution



Concurrency

Concurrent execution without CPU time-share

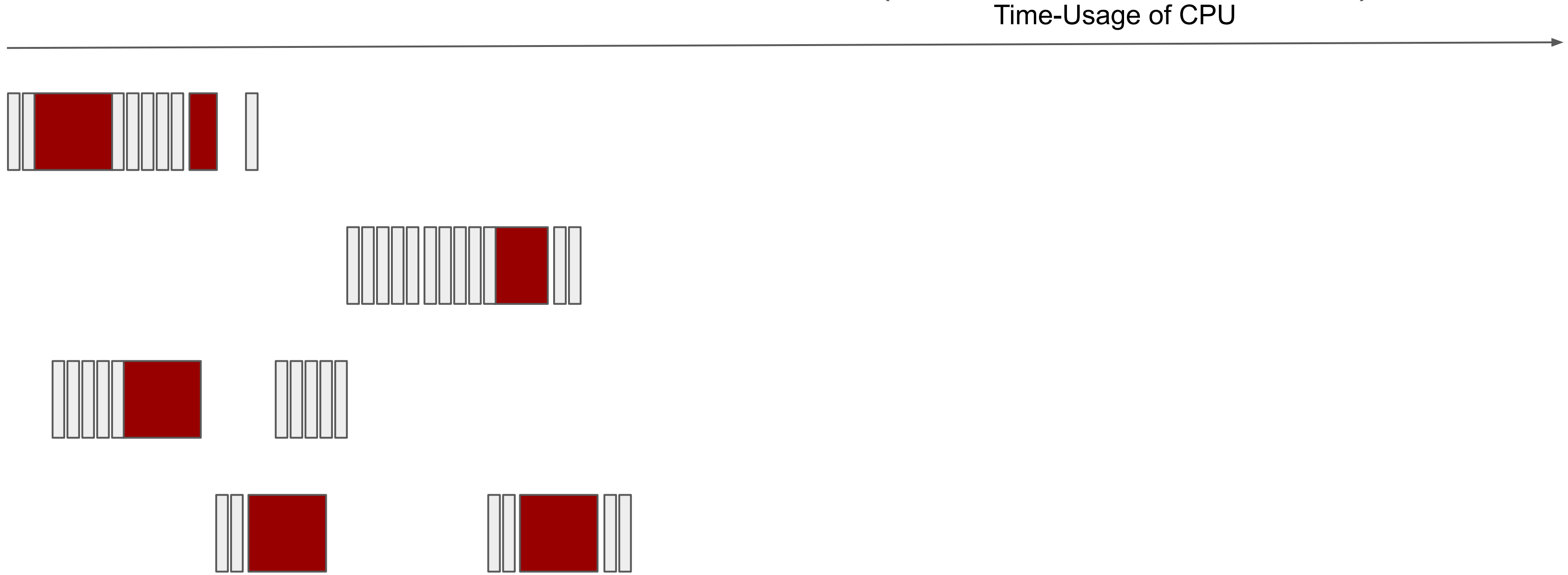


Concurrency

🟥 Idle Time

⬜ Execution Time

Concurrent execution without CPU time-share (Divide in frames of time)

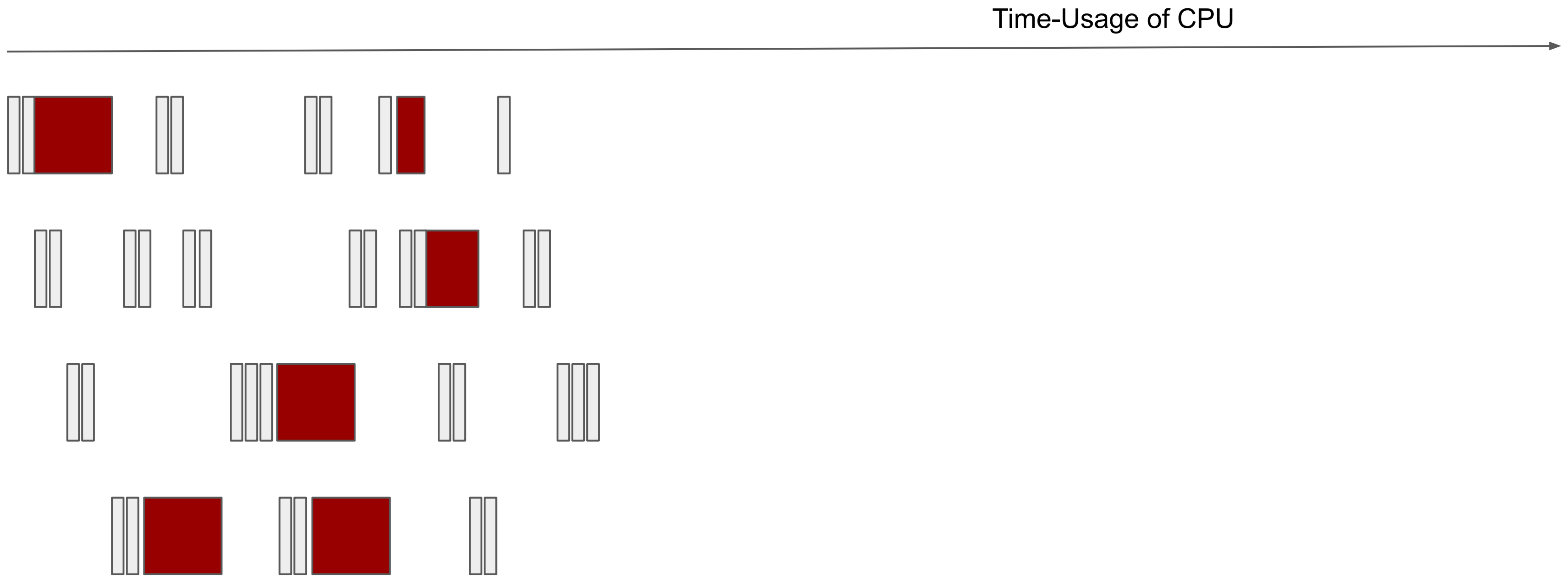


Concurrency

🟥 Idle Time

⬜ Execution Time

Concurrent execution with CPU time-share



Concurrency

Concurrency is achieved by time-shares from the OS.

A thread executes its code in its assigned time frames on the CPU core.

So it can be preempted by OS. It may also yield control to the OS.

But ...

Concurrency

But keep this in mind …

1 CPU cycle takes 0,3 ns

Linux OS context switch takes 1.000,00 ns

Linux thread lunch takes 5.000,00 ns

Linux process lunch takes 20.000,00 ns

Linux thread stack uses 8 MB (ulimit -s)

What other options do we have?

Coroutines !!!

(is one of them)

Coroutines

Definition:

“Coroutines are computer components that generalise subroutines for non-preemptive multitasking, by allowing execution to be suspended and resumed.”

- Wikipedia

Coroutines

Definition … in reasonable language:

A type of function that enables concurrency via

cooperative multitasking.

Coroutines

Definition … in reasonable language:

Coroutines are functions that essentially break up their execution into multiple parts.

Each time you call the coroutine, the next part of the task is performed.

They essentially can break in the middle of the execution and return to where they left off the next time they are called.

Coroutines

Features:

- It is a purely user-mode thread. Compared with thread or process, all the operations in coroutines are happening in user mode, so the cost to create or switch coroutines is cheaper.

  SWOOLE Coroutine lunch takes 190 ns.

- Coroutines = Cooperative - Functions
- Concept similar to ErlangVM light-processes

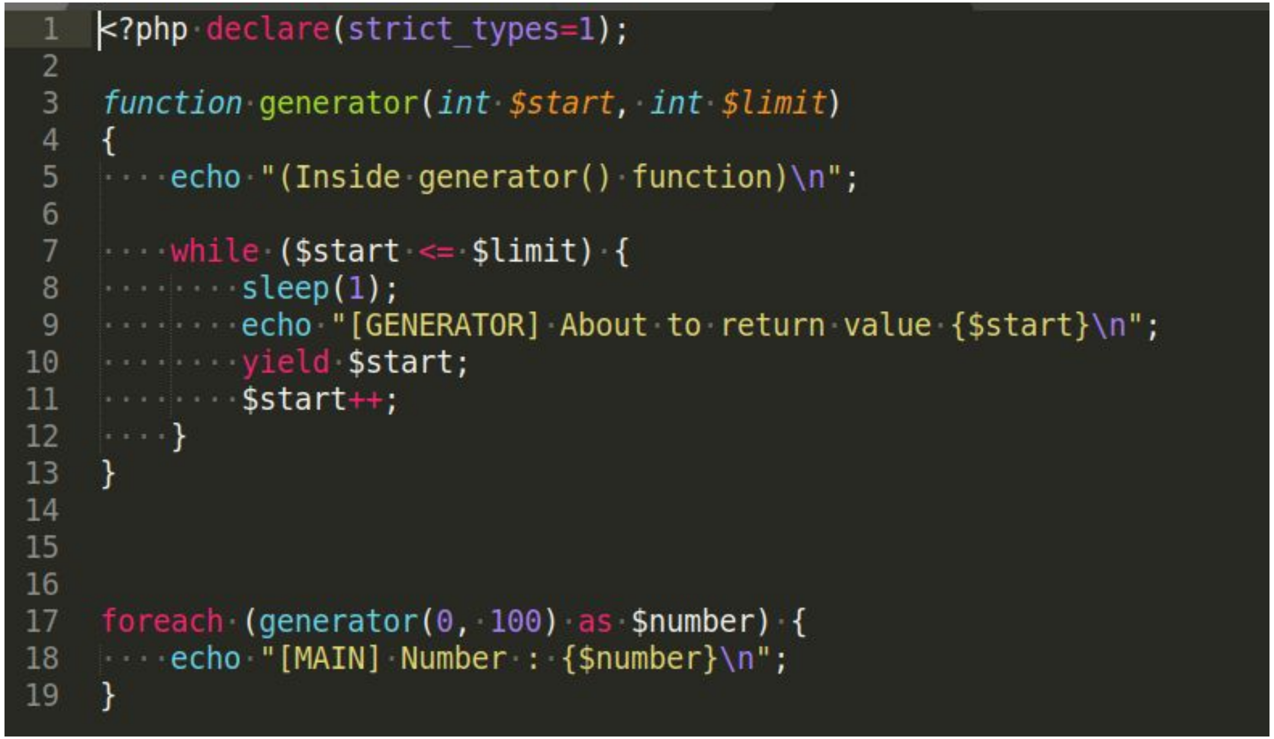
Coroutines in PHP

- PHP will generate coroutines using the Generators concept.
- Execution is divided using the “yield” instruction.
- Each time the Generator is called, the routine (function) picks up where it left off just after the “yield” keyword/instruction invocation.
- Generators can be used to consume values from another function. When used in such a way they are often referred to as enhanced generators, reverse generators or **coroutines**.

Coroutines in PHP

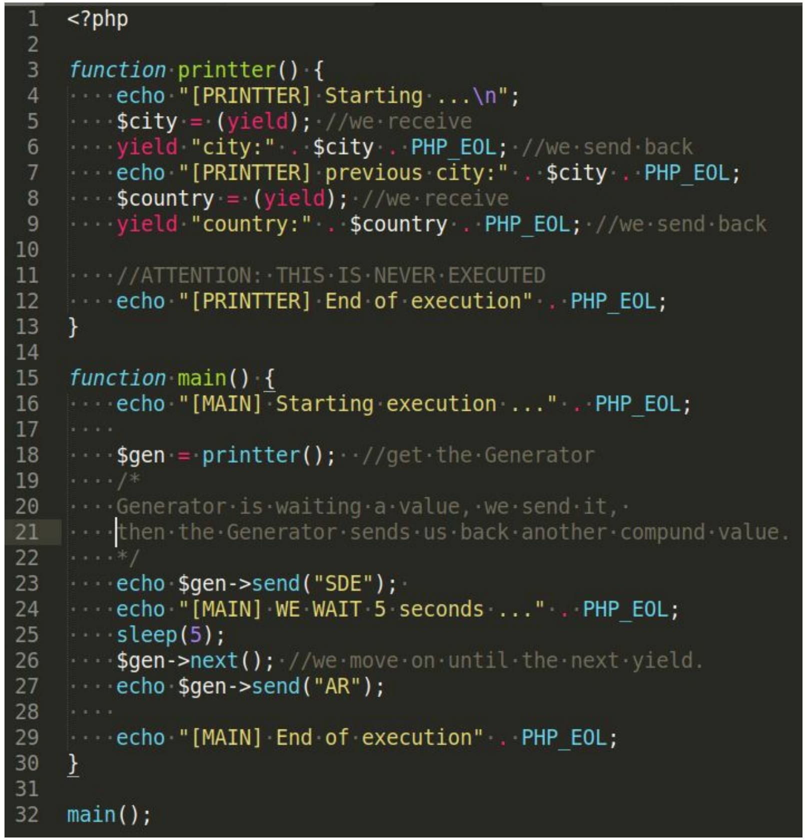
Code Sample

Coroutines in PHP - Simple Generator

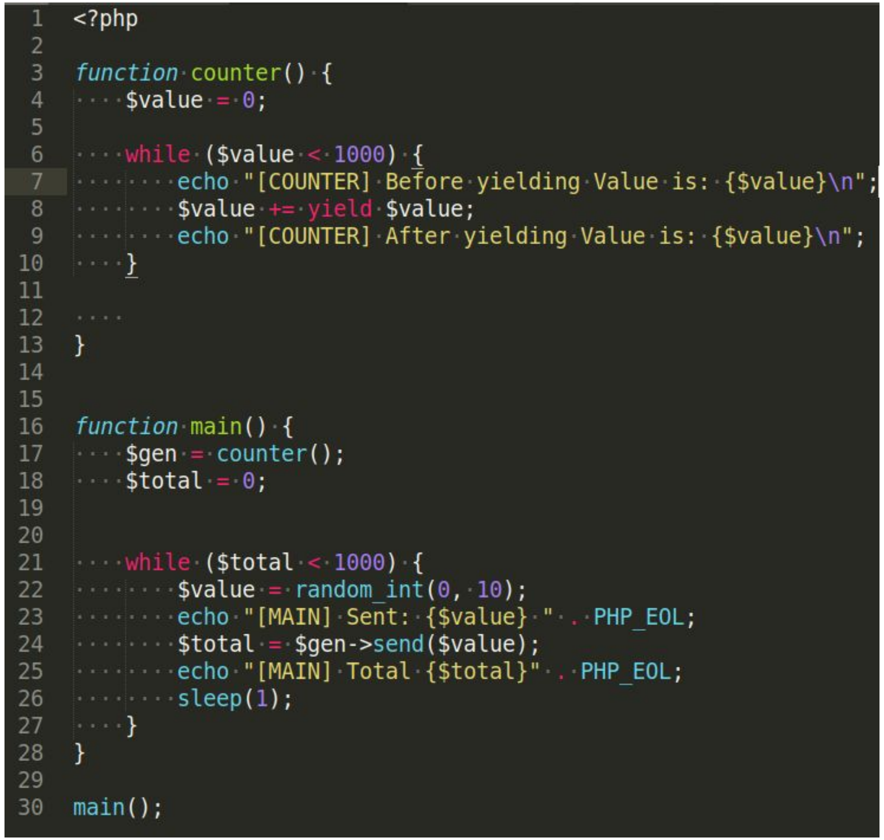


```php
<?php declare(strict_types=1);

function generator(int $start, int $limit)
{
    echo "(Inside generator() function)\n";

    while ($start <= $limit) {
        sleep(1);
        echo "[GENERATOR] About to return value {$start}\n";
        yield $start;
        $start++;
    }
}



foreach (generator(0, 100) as $number) {
    echo "[MAIN] Number : {$number}\n";
}
```

Coroutines in PHP - Inverse Generator



```php
<?php

function printter() {
    echo "[PRINTTER] Starting ...\n";
    $city = (yield); //we receive
    yield "city:" . $city . PHP_EOL; //we send back
    echo "[PRINTTER] previous city:" . $city . PHP_EOL;
    $country = (yield); //we receive
    yield "country:" . $country . PHP_EOL; //we send back

    //ATTENTION: THIS IS NEVER EXECUTED
    echo "[PRINTTER] End of execution" . PHP_EOL;
}

function main() {
    echo "[MAIN] Starting execution ..." . PHP_EOL;
    
    $gen = printter();  //get the Generator
    /*
    Generator is waiting a value, we send it, 
    then the Generator sends us back another compund value.
    */
    echo $gen->send("SDE"); 
    echo "[MAIN] WE WAIT 5 seconds ..." . PHP_EOL;
    sleep(5);
    $gen->next(); //we move on until the next yield.
    echo $gen->send("AR");
    
    echo "[MAIN] End of execution" . PHP_EOL;
}

main();
```



```php
<?php

function counter() {
    $value = 0;

    while ($value < 1000) {
        echo "[COUNTER] Before yielding Value is: {$value}\n";
        $value += yield $value;
        echo "[COUNTER] After yielding Value is: {$value}\n";
    }

    
}


function main() {
    $gen = counter();
    $total = 0;


    while ($total < 1000) {
        $value = random_int(0, 10);
        echo "[MAIN] Sent: {$value} " . PHP_EOL;
        $total = $gen->send($value);
        echo "[MAIN] Total {$total}" . PHP_EOL;
        sleep(1);
    }
}

main();
```

Coroutines in PHP

DIY is not scalable !!!

... so what?

SWOOLE

Coroutine based Async PHP programming framework

What is Swoole?

- Async PHP programming framework
- Build high-performance, scalable, concurrent TCP, UDP, Unix Socket, HTTP, WebSocket services with PHP and fluent Coroutine API.
- Swoole is designed for building large scale concurrency systems. It is written in C/C++ and installed as a PHP extension.
- Developers can use sync or async, coroutine API to write the applications or create thousands of light weight coroutines within one PHP process.
- The Swoole framework is released as a PHP extension (PECL) and runs as a PHP CLI application.

Features (some of them, there is more … believe me)

- Coroutine based concurrent asynchronous IO programming.
- Built-in Coroutine Async TCP/UDP/MQTT Server/HTTP/WebSocket/HTTP2 clients and servers.
- TCP/UDP Server provides the API to write TCP, UDP (DTLS), Unix Socket asynchronous servers. It supports IPv4, IPv6, one Way, two Way SSL and TLS Encryption. Developers do not have to know the internal implementations, only have to write the logics of the server in the callback functions.
- Coroutine Async MySQL, Redis, DNS, CURL, PostgresSQL, Task client and connection pool.
- Milliseconds scheduler.
- Coroutine Async File I/O API.
- Golang style channels.
- Capacity to interact with the low level EventLoop system of the framework with an API.
- Timer capabilities with coroutines enabled.

How does the code look like?

Requirements

- Operation system: Linux, FreeBSD or MacOS
- Linux kernel version >= 2.3.32
- PHP version >= 7.0.0
- GCC version >= 4.8

Installation

Linux users

```
#!/bin/bash
pecl install swoole
```

Mac users

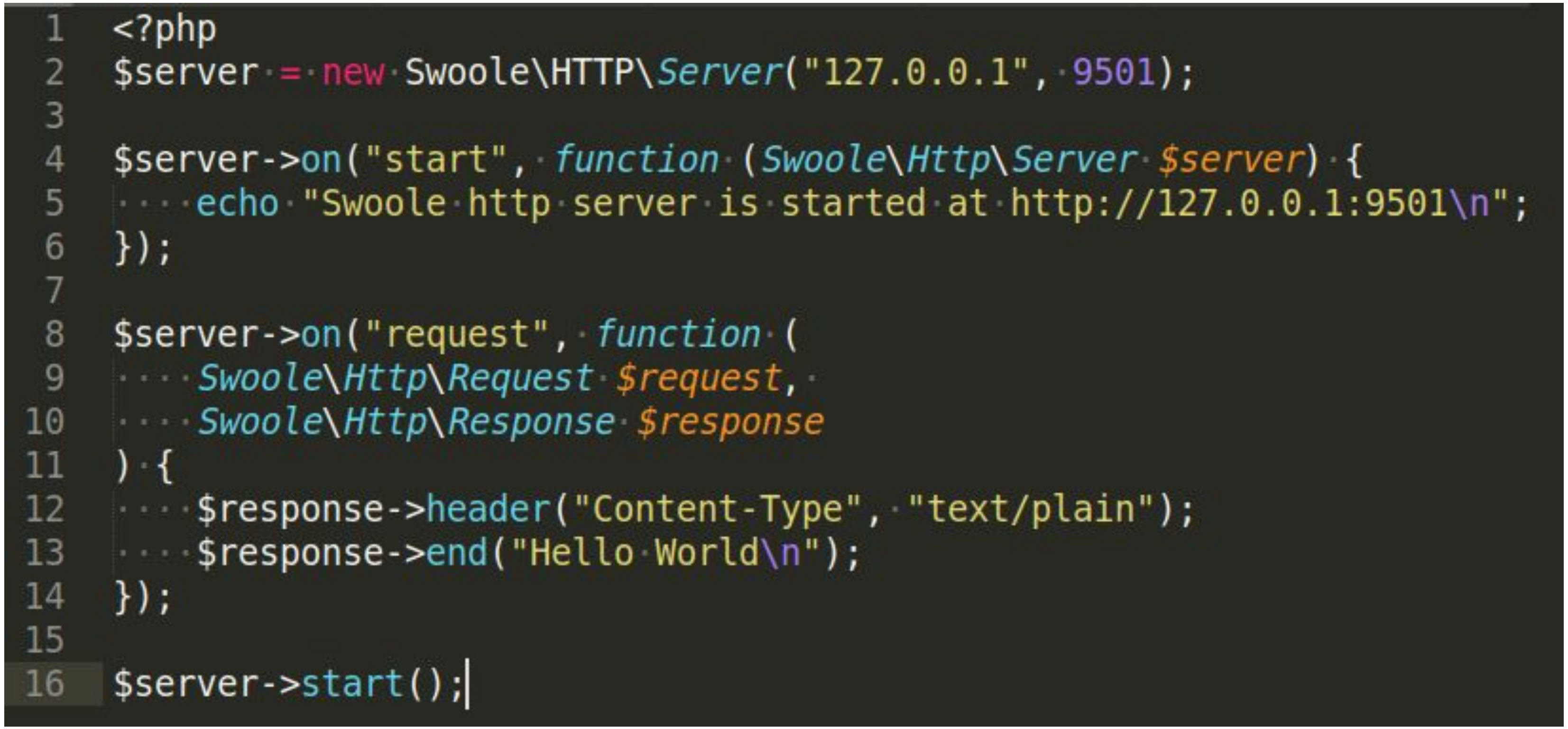
```
brew install php
```

```
#!/bin/bash
pecl install swoole
```

Disable this extensions

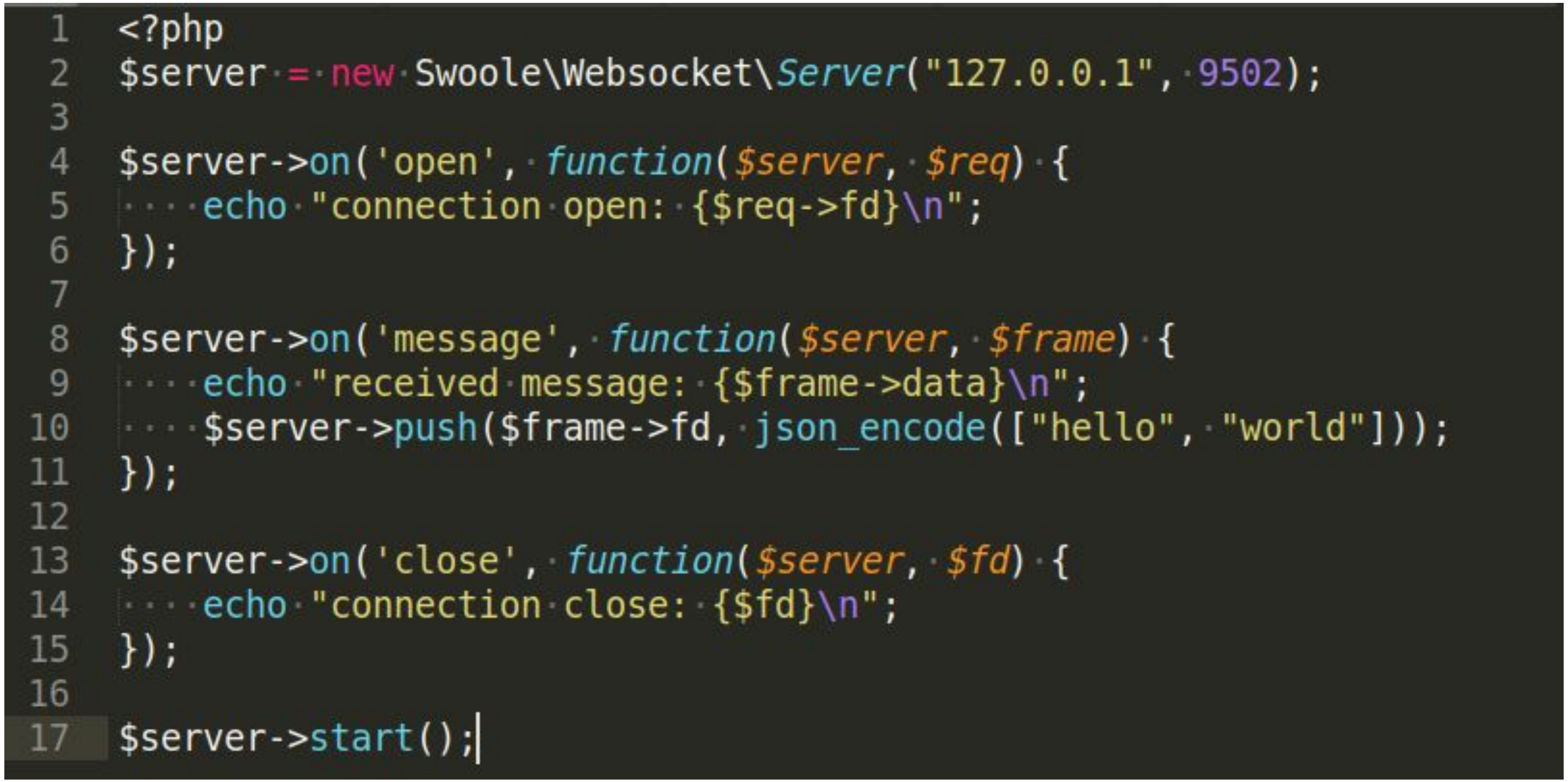
- xdebug
- phptrace
- aop
- molten
- Xhprof

HTTP Server

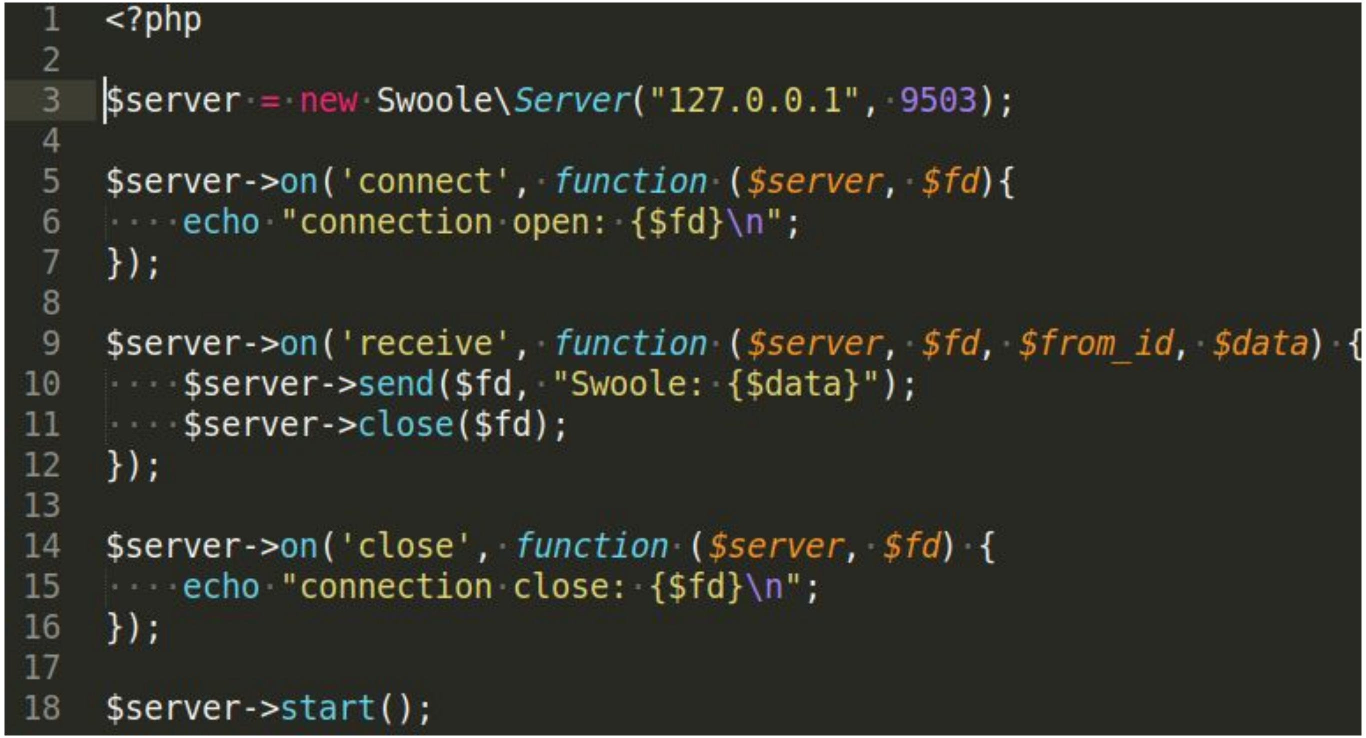


```php
<?php
$server = new Swoole\HTTP\Server("127.0.0.1", 9501);

$server->on("start", function (Swoole\Http\Server $server) {
    echo "Swoole http server is started at http://127.0.0.1:9501\n";
});

$server->on("request", function (
    Swoole\Http\Request $request, 
    Swoole\Http\Response $response
) {
    $response->header("Content-Type", "text/plain");
    $response->end("Hello World\n");
});

$server->start();
```

Websocket Server

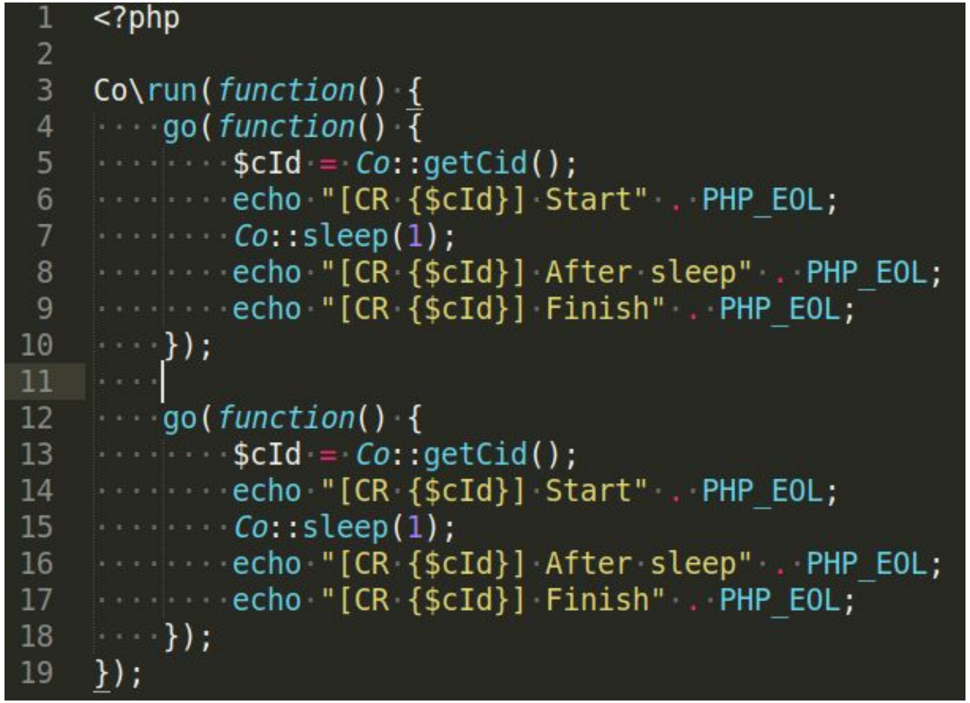

```php
<?php
$server = new Swoole\Websocket\Server("127.0.0.1", 9502);

$server->on('open', function($server, $req) {
    echo "connection open: {$req->fd}\n";
});

$server->on('message', function($server, $frame) {
    echo "received message: {$frame->data}\n";
    $server->push($frame->fd, json_encode(["hello", "world"]));
});

$server->on('close', function($server, $fd) {
    echo "connection close: {$fd}\n";
});

$server->start();
```

TCP Server



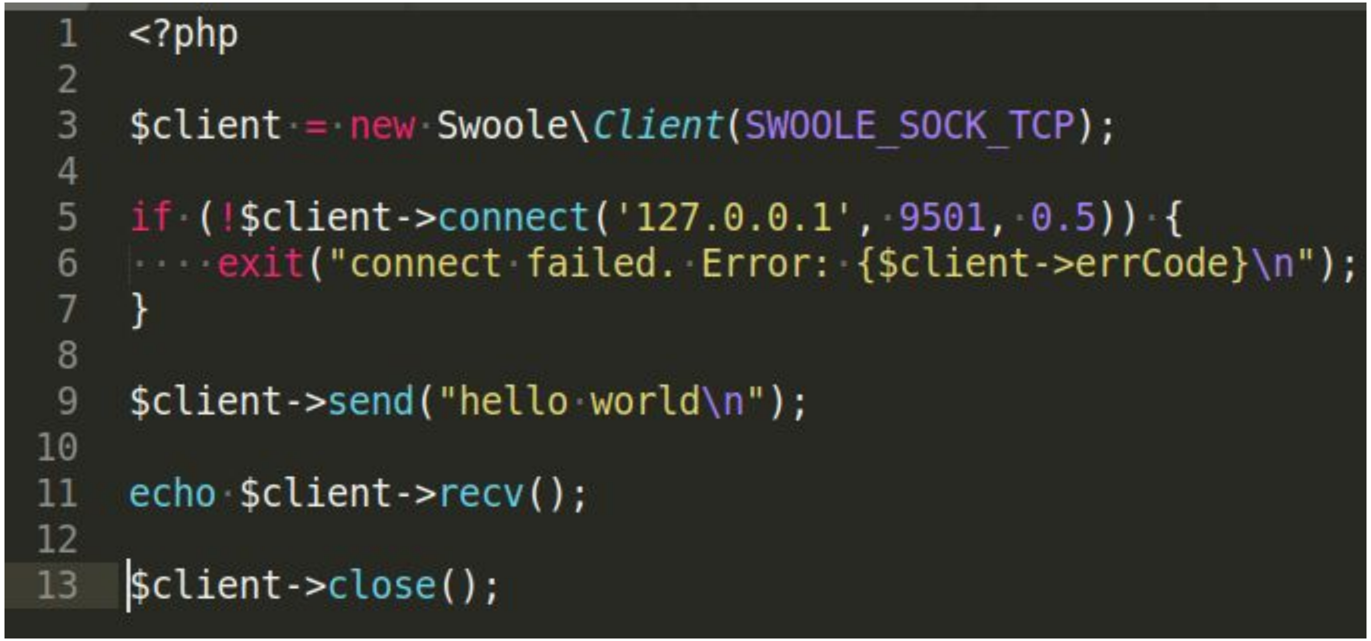
```php
<?php

$server = new Swoole\Server("127.0.0.1", 9503);

$server->on('connect', function ($server, $fd){
    echo "connection open: {$fd}\n";
});

$server->on('receive', function ($server, $fd, $from_id, $data) {
    $server->send($fd, "Swoole: {$data}");
    $server->close($fd);
});

$server->on('close', function ($server, $fd) {
    echo "connection close: {$fd}\n";
});

$server->start();
```

Coroutine



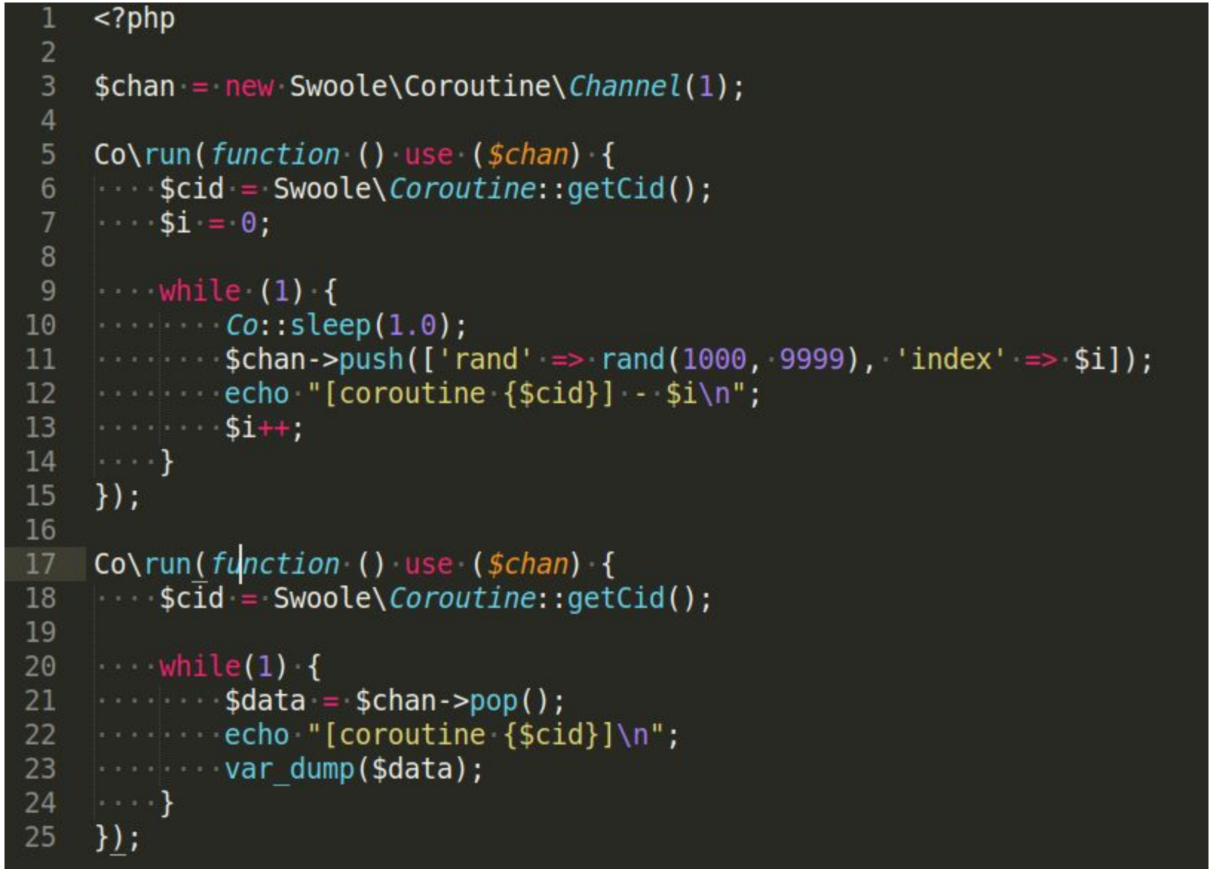
```php
<?php

Co\run(function() {
    go(function() {
        $cId = Co::getCid();
        echo "[CR {$cId}] Start" . PHP_EOL;
        Co::sleep(1);
        echo "[CR {$cId}] After sleep" . PHP_EOL;
        echo "[CR {$cId}] Finish" . PHP_EOL;
    });
    
    go(function() {
        $cId = Co::getCid();
        echo "[CR {$cId}] Start" . PHP_EOL;
        Co::sleep(1);
        echo "[CR {$cId}] After sleep" . PHP_EOL;
        echo "[CR {$cId}] Finish" . PHP_EOL;
    });
});
```

TCP Client



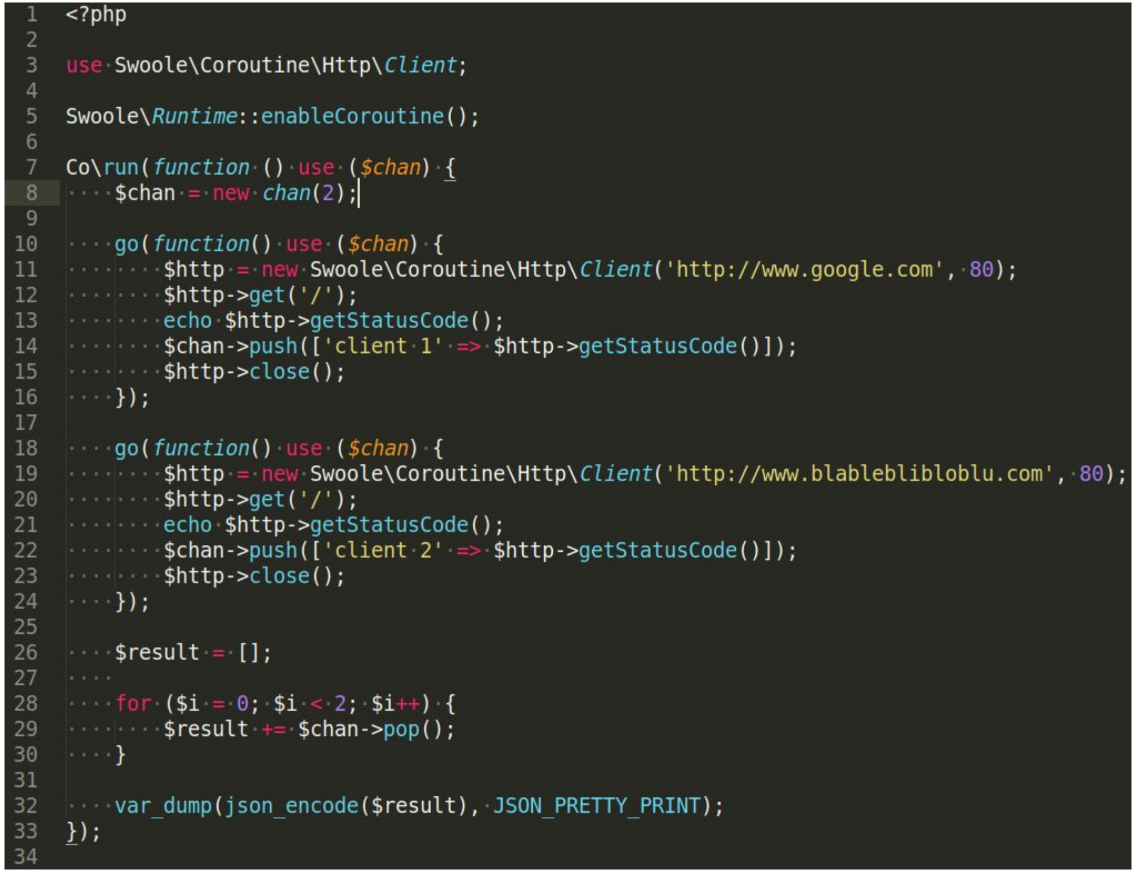
```php
<?php

$client = new Swoole\Client(SWOOLE_SOCK_TCP);

if (!$client->connect('127.0.0.1', 9501, 0.5)) {
    exit("connect failed. Error: {$client->errCode}\n");
}

$client->send("hello world\n");

echo $client->recv();

$client->close();
```

Channels



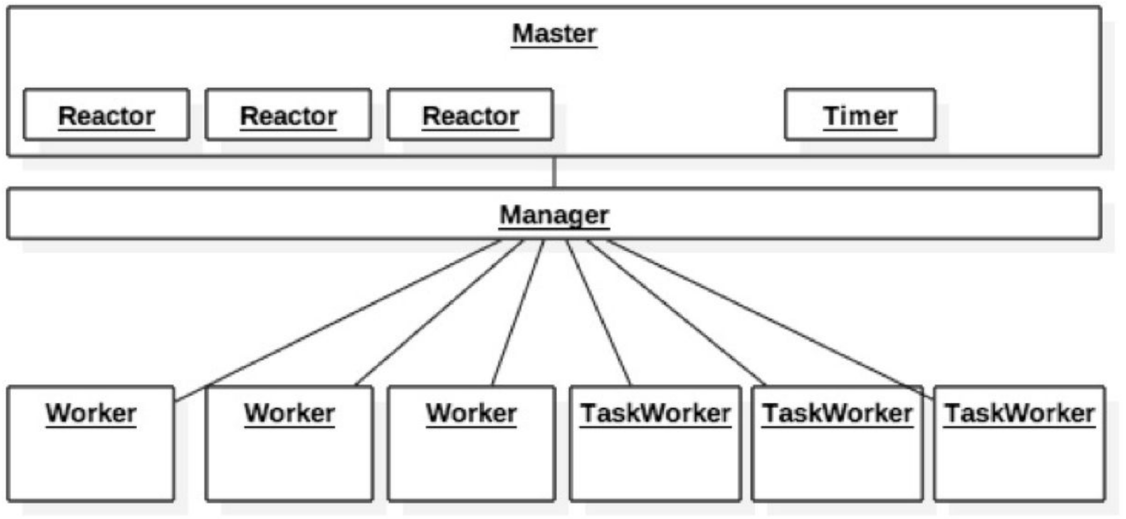
```php
<?php

$chan = new Swoole\Coroutine\Channel(1);

Co\run(function () use ($chan) {
    $cid = Swoole\Coroutine::getCid();
    $i = 0;

    while (1) {
        Co::sleep(1.0);
        $chan->push(['rand' => rand(1000, 9999), 'index' => $i]);
        echo "[coroutine {$cid}] - $i\n";
        $i++;
    }
});

Co\run(function () use ($chan) {
    $cid = Swoole\Coroutine::getCid();

    while(1) {
        $data = $chan->pop();
        echo "[coroutine {$cid}]\n";
        var_dump($data);
    }
});
```

Clients with coroutines

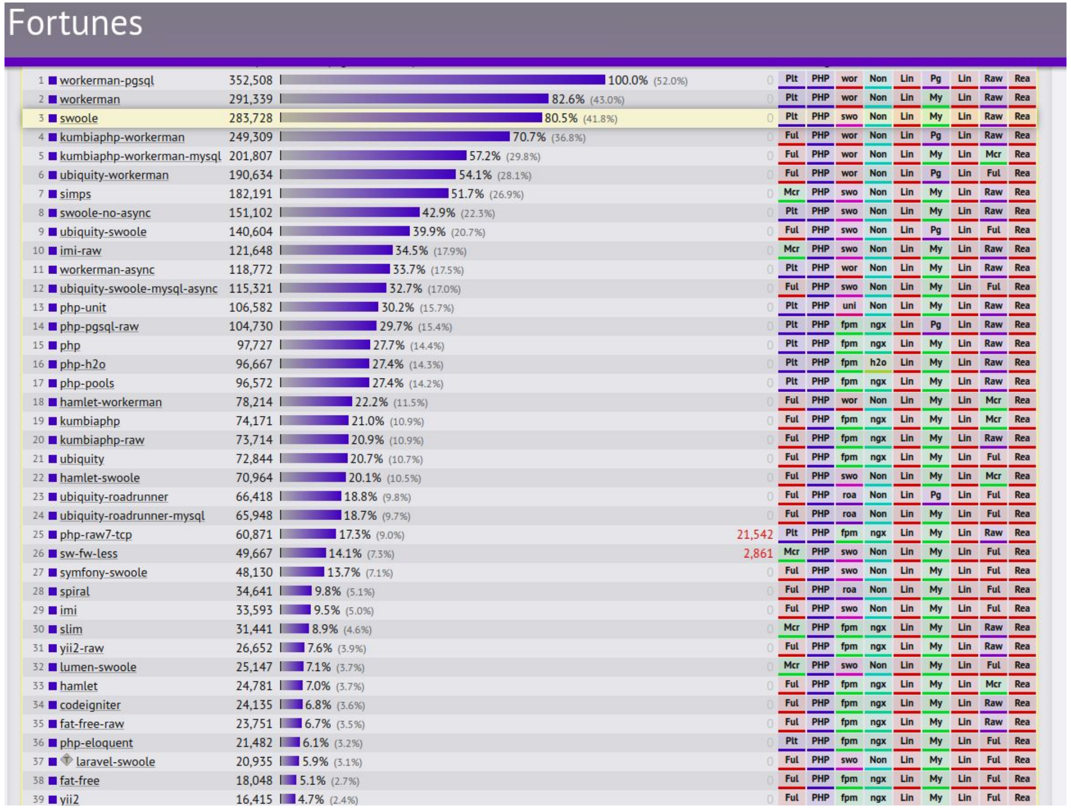


```php
<?php

use Swoole\Coroutine\Http\Client;

Swoole\Runtime::enableCoroutine();

Co\run(function () use ($chan) {
    $chan = new chan(2);

    go(function() use ($chan) {
        $http = new Swoole\Coroutine\Http\Client('http://www.google.com', 80);
        $http->get('/');
        echo $http->getStatusCode();
        $chan->push(['client 1' => $http->getStatusCode()]);
        $http->close();
    });

    go(function() use ($chan) {
        $http = new Swoole\Coroutine\Http\Client('http://www.blablebliblobl u.com', 80);
        $http->get('/');
        echo $http->getStatusCode();
        $chan->push(['client 2' => $http->getStatusCode()]);
        $http->close();
    });

    $result = [];
    
    for ($i = 0; $i < 2; $i++) {
        $result += $chan->pop();
    }

    var_dump(json_encode($result), JSON_PRETTY_PRINT);
});

```

Internals - Process Architecture



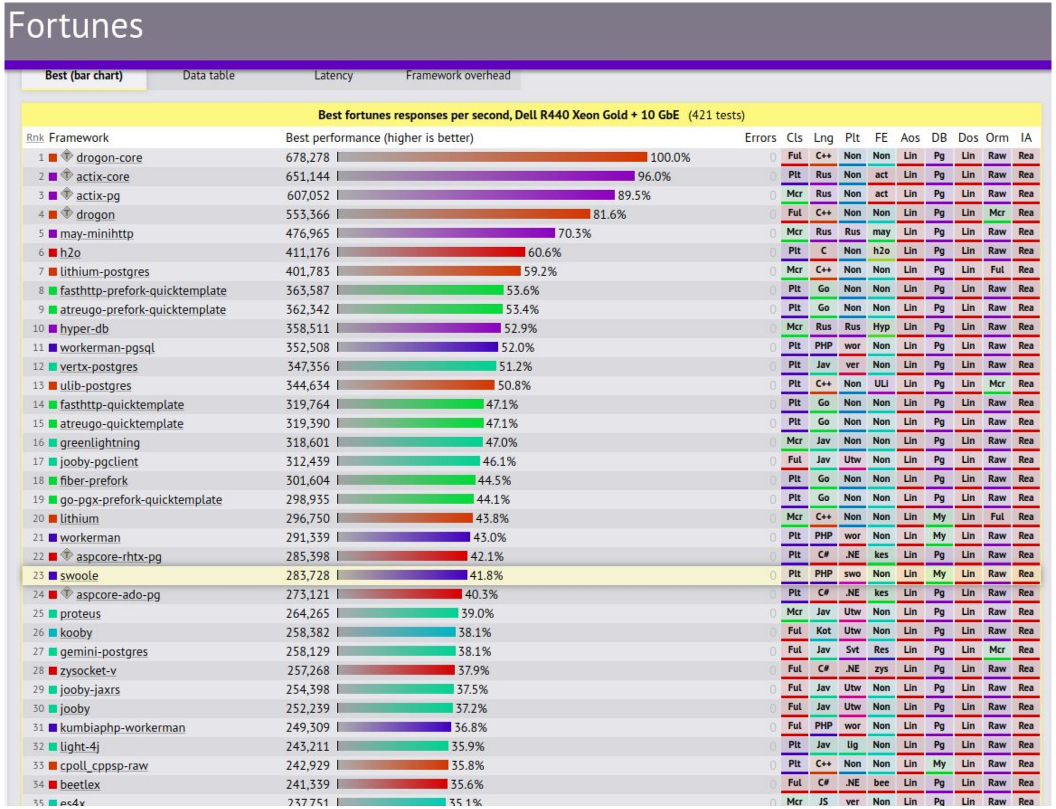
Benchmarks



Fortunes

1	workerman-pgsql	352,508	100.0% (52.0%)	0	Plt	PHP	wor	Non	Lin	Pg	Lin	Raw	Rea
2	workerman	291,339	82.6% (43.0%)	0	Plt	PHP	wor	Non	Lin	My	Lin	Raw	Rea
3	swoole	283,728	80.5% (41.8%)	0	Plt	PHP	swo	Non	Lin	My	Lin	Raw	Rea
4	kumbiaphp-workerman	249,309	70.7% (36.8%)	0	Ful	PHP	wor	Non	Lin	Pg	Lin	Raw	Rea
5	kumbiaphp-workerman-mysql	201,807	57.2% (29.8%)	0	Ful	PHP	wor	Non	Lin	My	Lin	Mcr	Rea
6	ubiquity-workerman	190,634	54.1% (28.1%)	0	Ful	PHP	wor	Non	Lin	Pg	Lin	Ful	Rea
7	simps	182,191	51.7% (26.9%)	0	Mcr	PHP	swo	Non	Lin	My	Lin	Raw	Rea
8	swoole-no-async	151,102	42.9% (22.3%)	0	Plt	PHP	swo	Non	Lin	My	Lin	Raw	Rea
9	ubiquity-swoole	140,604	39.9% (20.7%)	0	Ful	PHP	swo	Non	Lin	Pg	Lin	Ful	Rea
10	imi-raw	121,648	34.5% (17.9%)	0	Mcr	PHP	swo	Non	Lin	My	Lin	Raw	Rea
11	workerman-async	118,772	33.7% (17.5%)	0	Plt	PHP	wor	Non	Lin	My	Lin	Raw	Rea
12	ubiquity-swoole-mysql-async	115,321	32.7% (17.0%)	0	Ful	PHP	swo	Non	Lin	My	Lin	Ful	Rea
13	php-unit	106,582	30.2% (15.7%)	0	Plt	PHP	uni	Non	Lin	My	Lin	Raw	Rea
14	php-pgsql-raw	104,730	29.7% (15.4%)	0	Plt	PHP	fpm	ngx	Lin	Pg	Lin	Raw	Rea
15	php	97,727	27.7% (14.4%)	0	Plt	PHP	fpm	ngx	Lin	My	Lin	Raw	Rea
16	php-h2o	96,667	27.4% (14.3%)	0	Plt	PHP	fpm	h2o	Lin	My	Lin	Raw	Rea
17	php-pools	96,572	27.4% (14.2%)	0	Plt	PHP	fpm	ngx	Lin	My	Lin	Raw	Rea
18	hamlet-workerman	78,214	22.2% (11.5%)	0	Ful	PHP	wor	Non	Lin	My	Lin	Mcr	Rea
19	kumbiaphp	74,171	21.0% (10.9%)	0	Ful	PHP	fpm	ngx	Lin	My	Lin	Mcr	Rea
20	kumbiaphp-raw	73,714	20.9% (10.9%)	0	Ful	PHP	fpm	ngx	Lin	My	Lin	Raw	Rea
21	ubiquity	72,844	20.7% (10.7%)	0	Ful	PHP	fpm	ngx	Lin	My	Lin	Ful	Rea
22	hamlet-swoole	70,964	20.1% (10.5%)	0	Ful	PHP	swo	Non	Lin	My	Lin	Mcr	Rea
23	ubiquity-roadrunner	66,418	18.8% (9.8%)	0	Ful	PHP	roa	Non	Lin	Pg	Lin	Ful	Rea
24	ubiquity-roadrunner-mysql	65,948	18.7% (9.7%)	0	Ful	PHP	roa	Non	Lin	My	Lin	Ful	Rea
25	php-raw7-tcp	60,871	17.3% (9.0%)	21,542	Plt	PHP	fpm	ngx	Lin	My	Lin	Raw	Rea
26	sw-fw-less	49,667	14.1% (7.3%)	2,861	Mcr	PHP	swo	Non	Lin	My	Lin	Ful	Rea
27	symfony-swoole	48,130	13.7% (7.1%)	0	Ful	PHP	swo	Non	Lin	My	Lin	Ful	Rea
28	spiral	34,641	9.8% (5.1%)	0	Ful	PHP	roa	Non	Lin	My	Lin	Ful	Rea
29	imi	33,593	9.5% (5.0%)	0	Ful	PHP	swo	Non	Lin	My	Lin	Ful	Rea
30	slim	31,441	8.9% (4.6%)	0	Mcr	PHP	fpm	ngx	Lin	My	Lin	Raw	Rea
31	yii2-raw	26,652	7.6% (3.9%)	0	Ful	PHP	fpm	ngx	Lin	My	Lin	Raw	Rea
32	lumen-swoole	25,147	7.1% (3.7%)	0	Mcr	PHP	swo	Non	Lin	My	Lin	Ful	Rea
33	hamlet	24,781	7.0% (3.7%)	0	Ful	PHP	fpm	ngx	Lin	My	Lin	Mcr	Rea
34	codeigniter	24,135	6.8% (3.6%)	0	Ful	PHP	fpm	ngx	Lin	My	Lin	Raw	Rea
35	fat-free-raw	23,751	6.7% (3.5%)	0	Ful	PHP	fpm	ngx	Lin	My	Lin	Raw	Rea
36	php-eloquent	21,482	6.1% (3.2%)	0	Plt	PHP	fpm	ngx	Lin	My	Lin	Ful	Rea
37	laravel-swoole	20,935	5.9% (3.1%)	0	Ful	PHP	swo	Non	Lin	My	Lin	Ful	Rea
38	fat-free	18,048	5.1% (2.7%)	0	Ful	PHP	fpm	ngx	Lin	My	Lin	Ful	Rea
39	yii2	16,415	4.7% (2.4%)	0	Ful	PHP	fpm	ngx	Lin	My	Lin	Ful	Rea

Benchmarks



Fortunes

Best (bar chart) | Data table | Latency | Framework overhead

Best fortunes responses per second, Dell R440 Xeon Gold + 10 GbE (421 tests)

Rnk	Framework	Best performance (higher is better)		Errors	Cls	Lng	Plt	FE	Aos	DB	Dos	Orm	IA
1	🟥 drogon-core	678,278	100.0%	0	Ful	C++	Non	Non	Lin	Pg	Lin	Raw	Rea
2	🟪 actix-core	651,144	96.0%	0	Plt	Rus	Non	act	Lin	Pg	Lin	Raw	Rea
3	🟪 actix-pg	607,052	89.5%	0	Mcr	Rus	Non	act	Lin	Pg	Lin	Raw	Rea
4	🟥 drogon	553,366	81.6%	0	Ful	C++	Non	Non	Lin	Pg	Lin	Mcr	Rea
5	🟪 may-minihttp	476,965	70.3%	0	Mcr	Rus	Rus	may	Lin	Pg	Lin	Raw	Rea
6	🟥 h2o	411,176	60.6%	0	Plt	C	Non	h2o	Lin	Pg	Lin	Raw	Rea
7	🟥 lithium-postgres	401,783	59.2%	0	Mcr	C++	Non	Non	Lin	Pg	Lin	Ful	Rea
8	🟩 fasthttp-prefork-quicktemplate	363,587	53.6%	0	Plt	Go	Non	Non	Lin	Pg	Lin	Raw	Rea
9	🟩 atreugo-prefork-quicktemplate	362,342	53.4%	0	Plt	Go	Non	Non	Lin	Pg	Lin	Raw	Rea
10	🟪 hyper-db	358,511	52.9%	0	Mcr	Rus	Rus	Hyp	Lin	Pg	Lin	Raw	Rea
11	🟦 workerman-pgsql	352,508	52.0%	0	Plt	PHP	wor	Non	Lin	Pg	Lin	Raw	Rea
12	🟩 vertx-postgres	347,356	51.2%	0	Plt	Jav	ver	Non	Lin	Pg	Lin	Raw	Rea
13	🟥 ulib-postgres	344,634	50.8%	0	Plt	C++	Non	ULi	Lin	Pg	Lin	Mcr	Rea
14	🟩 fasthttp-quicktemplate	319,764	47.1%	0	Plt	Go	Non	Non	Lin	Pg	Lin	Raw	Rea
15	🟩 atreugo-quicktemplate	319,390	47.1%	0	Plt	Go	Non	Non	Lin	Pg	Lin	Raw	Rea
16	🟩 greenlightning	318,601	47.0%	0	Mcr	Jav	Non	Non	Lin	Pg	Lin	Raw	Rea
17	🟩 jooby-pgclient	312,439	46.1%	0	Ful	Jav	Utw	Non	Lin	Pg	Lin	Raw	Rea
18	🟩 fiber-prefork	301,604	44.5%	0	Plt	Go	Non	Non	Lin	Pg	Lin	Raw	Rea
19	🟩 go-pgx-prefork-quicktemplate	298,935	44.1%	0	Plt	Go	Non	Non	Lin	Pg	Lin	Raw	Rea
20	🟥 lithium	296,750	43.8%	0	Mcr	C++	Non	Non	Lin	My	Lin	Ful	Rea
21	🟦 workerman	291,339	43.0%	0	Plt	PHP	wor	Non	Lin	My	Lin	Raw	Rea
22	🟥 aspcore-rhtx-pg	285,398	42.1%	0	Plt	C#	.NE	kes	Lin	Pg	Lin	Raw	Rea
23	🟦 swoole	283,728	41.8%	0	Plt	PHP	swo	Non	Lin	My	Lin	Raw	Rea
24	🟥 aspcore-ado-pg	273,121	40.3%	0	Plt	C#	.NE	kes	Lin	Pg	Lin	Raw	Rea
25	🟩 proteus	264,265	39.0%	0	Mcr	Jav	Utw	Non	Lin	Pg	Lin	Raw	Rea
26	🟦 kooby	258,382	38.1%	0	Ful	Kot	Utw	Non	Lin	Pg	Lin	Raw	Rea
27	🟩 gemini-postgres	258,129	38.1%	0	Ful	Jav	Svt	Res	Lin	Pg	Lin	Mcr	Rea
28	🟥 zysocket-v	257,268	37.9%	0	Ful	C#	.NE	zys	Lin	Pg	Lin	Raw	Rea
29	🟩 jooby-jaxrs	254,398	37.5%	0	Ful	Jav	Utw	Non	Lin	Pg	Lin	Raw	Rea
30	🟩 jooby	252,239	37.2%	0	Ful	Jav	Utw	Non	Lin	Pg	Lin	Raw	Rea
31	🟦 kumbiaphp-workerman	249,309	36.8%	0	Ful	PHP	wor	Non	Lin	Pg	Lin	Raw	Rea
32	🟩 light-4j	243,211	35.9%	0	Plt	Jav	lig	Non	Lin	Pg	Lin	Raw	Rea
33	🟥 cpoll_cppsp-raw	242,929	35.8%	0	Plt	C++	Non	Non	Lin	My	Lin	Raw	Rea
34	🟥 beetlex	241,339	35.6%	0	Ful	C#	.NE	bee	Lin	Pg	Lin	Raw	Rea
35	🟩 es4x	237,751	35.1%	0	Mcr	JS	ver	Non	Lin	Pg	Lin	Raw	Rea

Precautions - Not everything is easy-peasy

- Do NOT use sleep or derivatives, instead use the ones provided in the Coroutine API.
- Avoid usage of exit() and die(), can lead o undesired behaviors.
- Use the register_shutdown_function() to catch fatal errors and do proper clean up.
- If the code provided to a callback function can throw an exception, then use try-catch blocks.
- The set_exception_handler() is not supported.
- Do NOT share MySQL/Redis clint connection between callback functions. Create them dynamically instead.

Precautions - Not everything is easy-peasy

- Always include/require your php files before the server starts, otherwise you can incur in “re declaration” errors.
- Take care about the memory you are using, remember that with this approach PHP becomes a long running process.

```
function test() {
    global $e;
    $a = new Object;
    $b = fopen('/data/t.log', 'r+');
    $c = new swoole_client(SWOOLE_SYNC);
    $d = new swoole_client(SWOOLE_SYNC);
    $e['client'] = $d;
  }
```

Precautions - Not everything is easy-peasy

SWOOLE

- Avoid or pay special attention to global variables and static ones, because they will be released (recycled) when the server is finish.
- Etc.

http://wiki.swoole.com/#/getting_started/notice

Alternatives

Not as powerful, flexible, but really good ones.

https://reactphp.org/

https://amphp.org/

Frameworks made up with Swoole

https://www.hyperf.io/

https://simps.io/

Workerman

https://github.com/walkor/Workerman

https://swoft.org/

Useful links

Github: https://github.com/swoole

Docker image: https://github.com/swoole/docker-swoole

Courses (CH): https://course.swoole-cloud.com/

Official site (EN): https://www.swoole.co.uk

Wiki (CH): http://wiki.swoole.com/

Awesome Swoole: https://github.com/swooletw/awesome-swoole