One year of asynchronous Php in production

Benoit Viguier

Afup Lyon
10/10/2019

One year of asynchronous Php in production

Benoit Viguier

Afup Lyon
10/10/2019

Previously…

Episode 1

Generators for Asynchronous Programming

User Manual

Benoit Viguier
@b_viguier

Générateurs et Programmation Asynchrone: Mode d’emploi - Benoit Viguier - Forum PHP 2018

716 vues

AFUP PHP
Ajoutée le 13 nov. 2018

Plus d'informations sur cette conférence : https://afup.org/talks/2752-generateu...

Cette vidéo vous a plu ? Adhérez à l'AFUP pour soutenir son activité :

PLUS

2 commentaires TRIER PAR

Seb7876557 il y a 8 mois

J'ai pas compris "le truc" technique, sûrement parce que je manque de connaissance sur les générateurs :/

RÉPONDRE

SOLenG il y a 9 mois

vendre du synchrone comme étant de l'asynchrone, c'est pas parce qu'il parle de 'tick' que ça rend le paradigme différent xD

1 RÉPONDRE

SOLenG il y a 9 mois

vendre du synchrone comme étant de l'asynchrone, c'est pas parce qu'il parle de 'tick' que ça rend le paradigme différent xD

RÉPONDRE

ASYNCHRONOUS PHP

IS POSSIBLE

I WANT TO BELIEVE

Once Upon a Time…

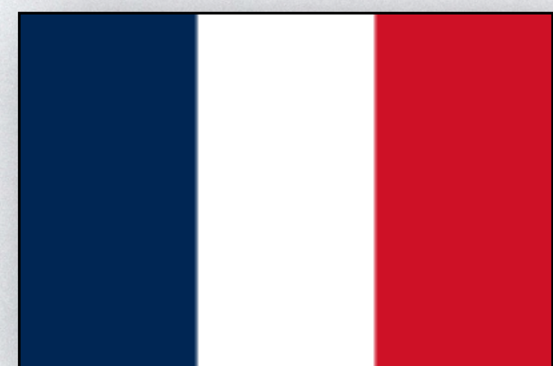
Context

@b_viguier

6play

W9 M6 6ter

6play Family



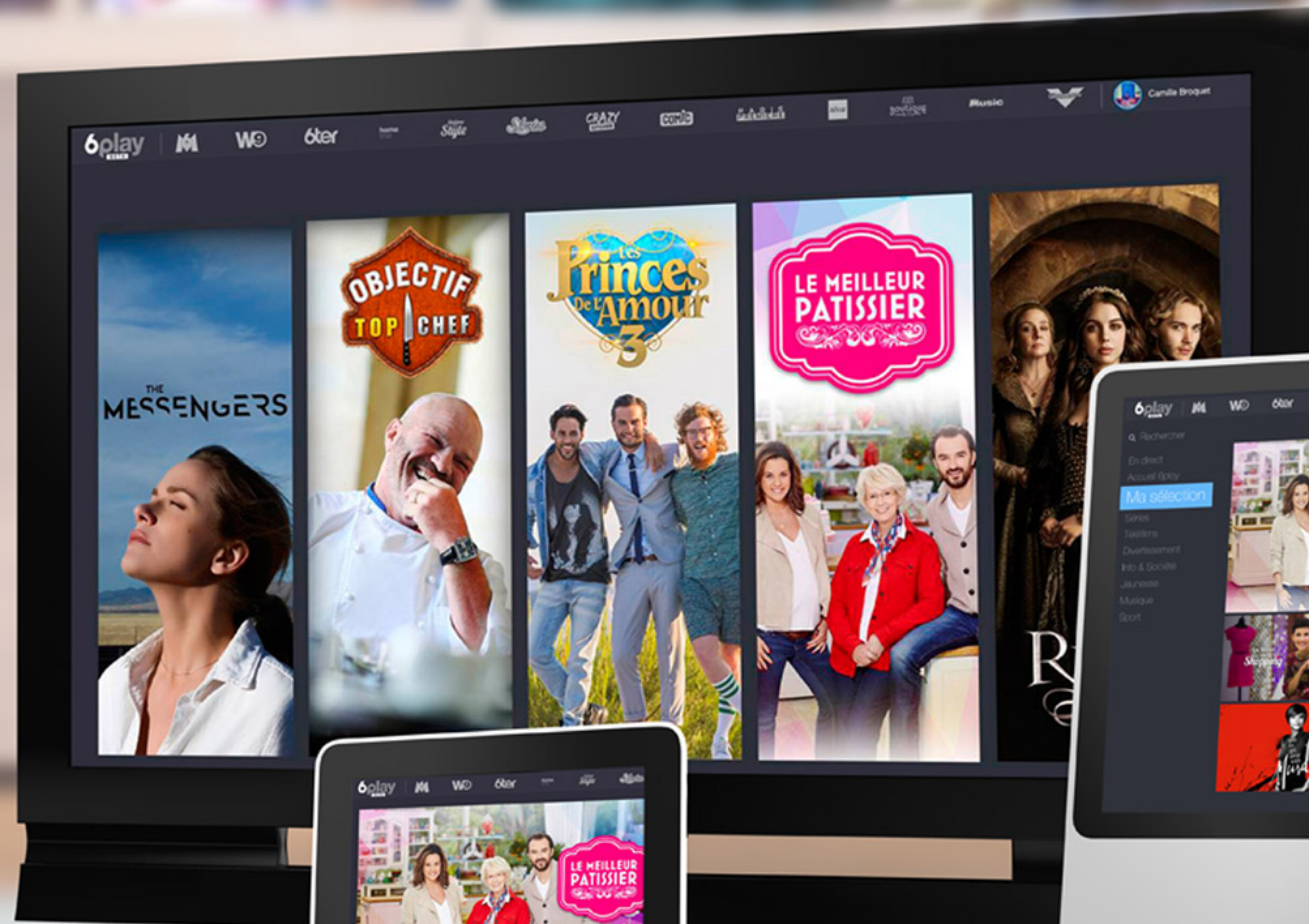
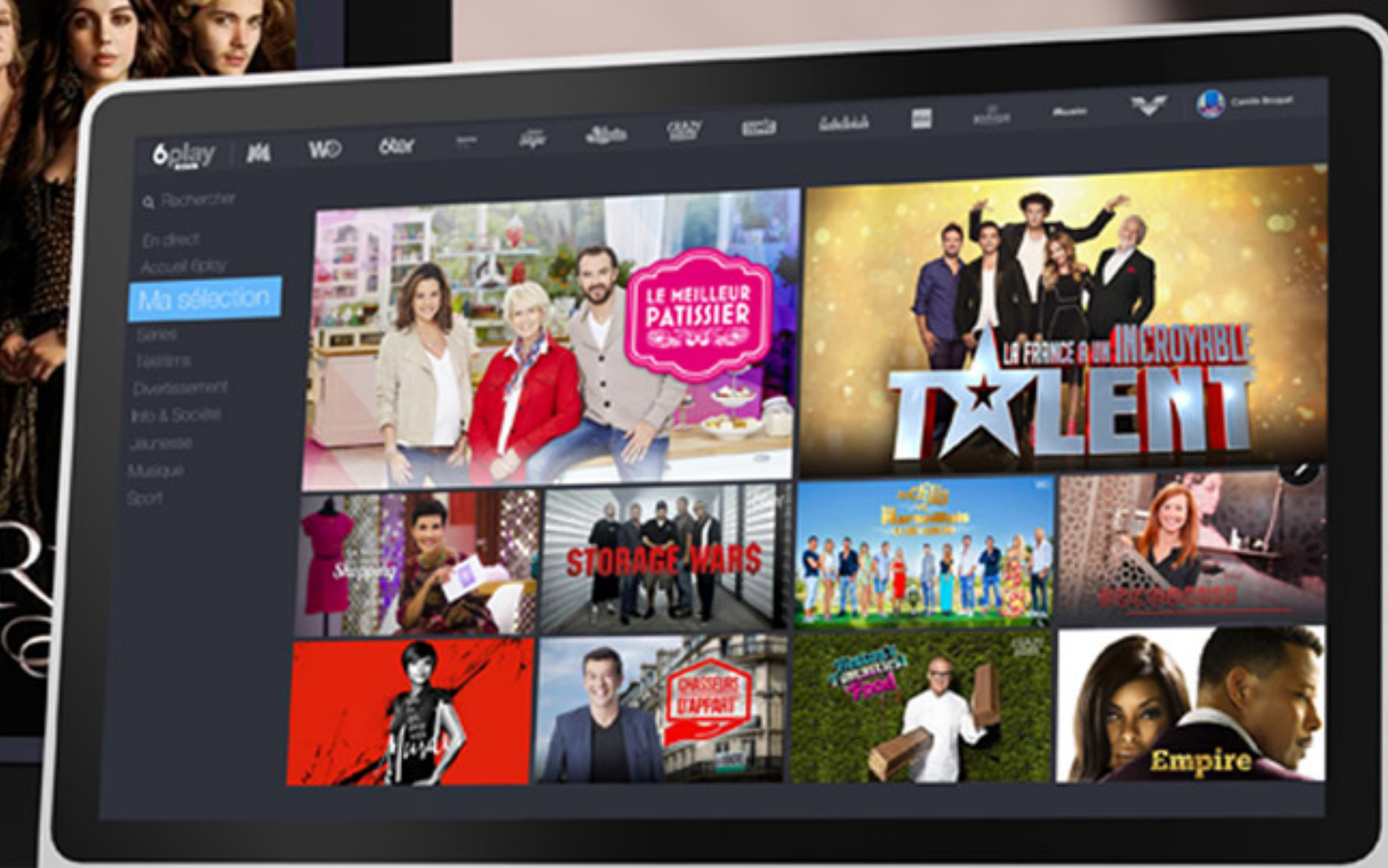
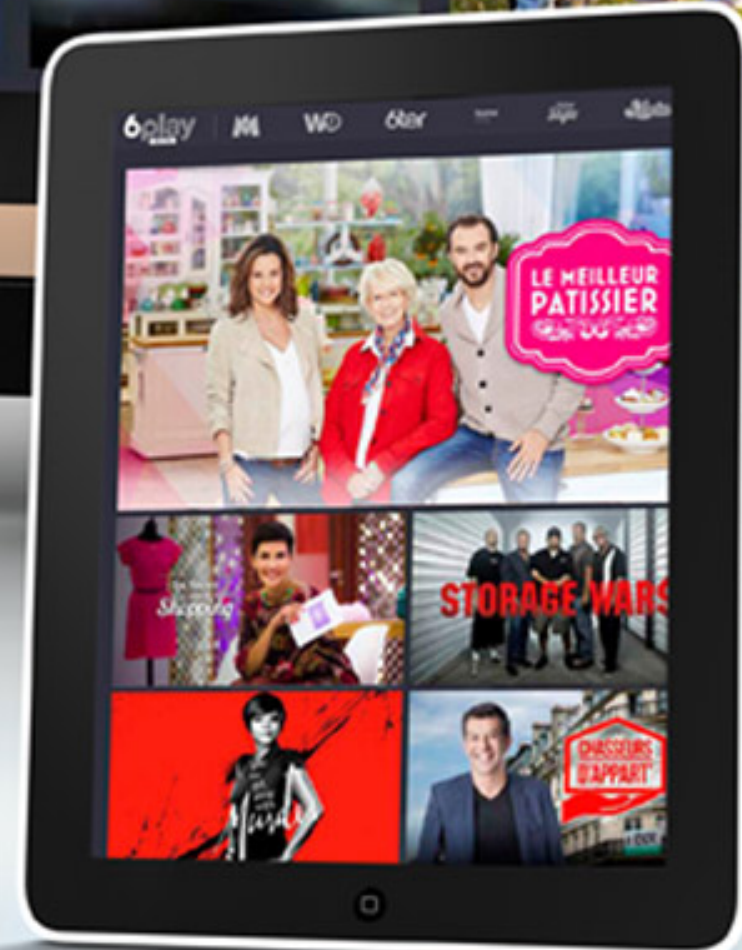
6play Family

DE 6PLAY A UNE PLATEFORME INTERNATIONALE

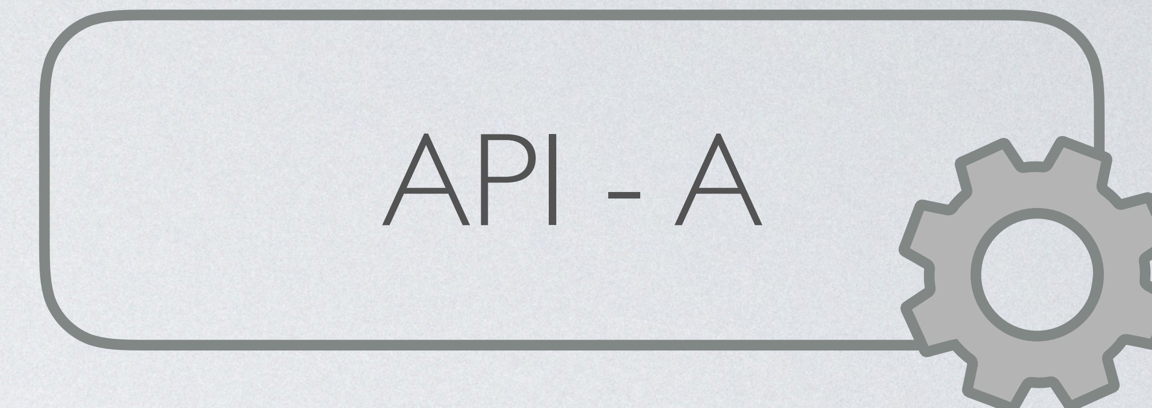
Retour d'expérience

6play

FORUMPHP PARIS2018


6play
W9
6ter
THE MESSENGERS
OBJECTIF TOP CHEF
Les Princes de l'Amour 3
LE MEILLEUR PATISSIER

LE MEILLEUR PATISSIER
LA FRANCE A UN INCROYABLE TALENT
STORAGE WARS
Empire

LE MEILLEUR PATISSIER
STORAGE WARS

Benoit Viguier
Backend
Lead Developer

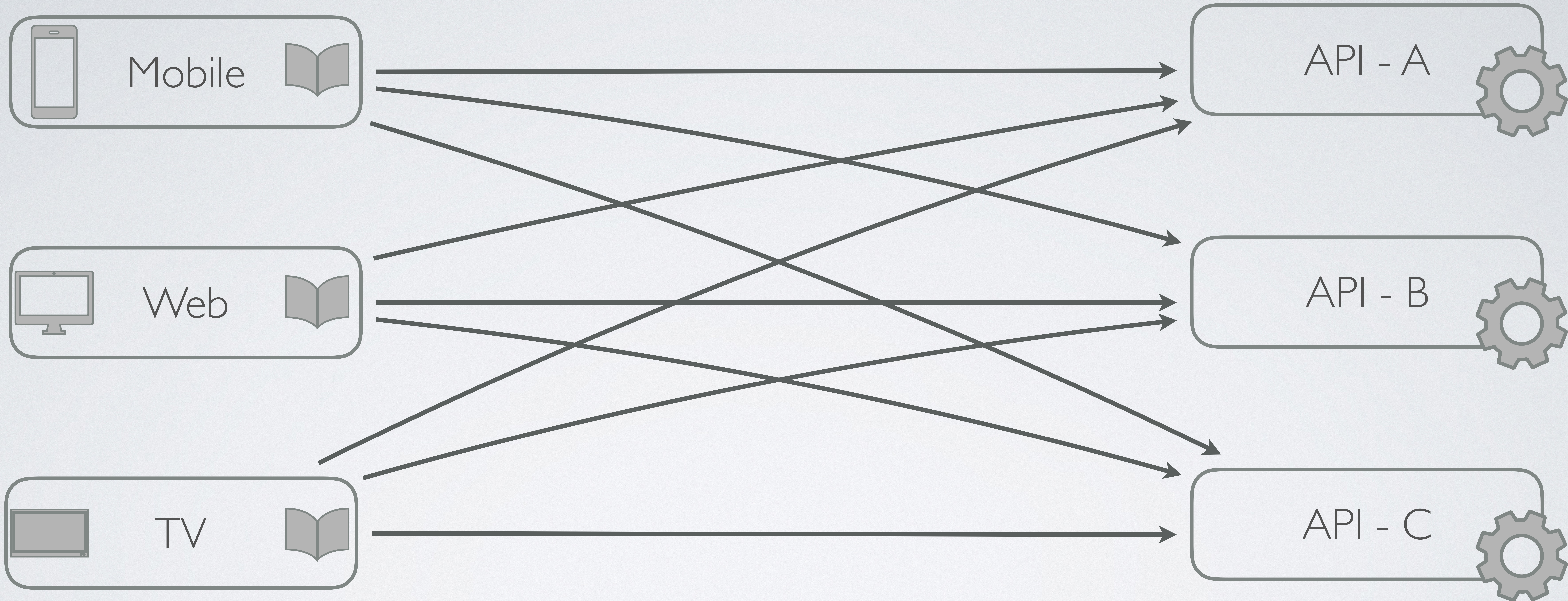


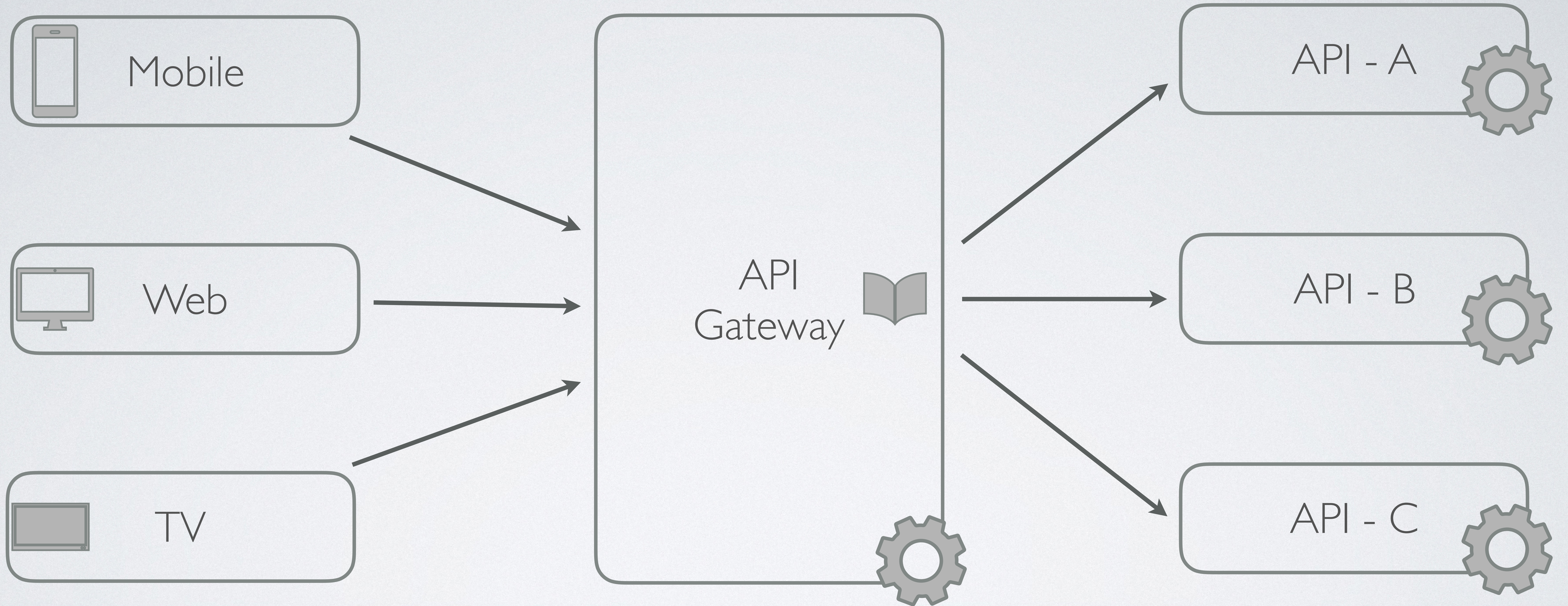
Mobile

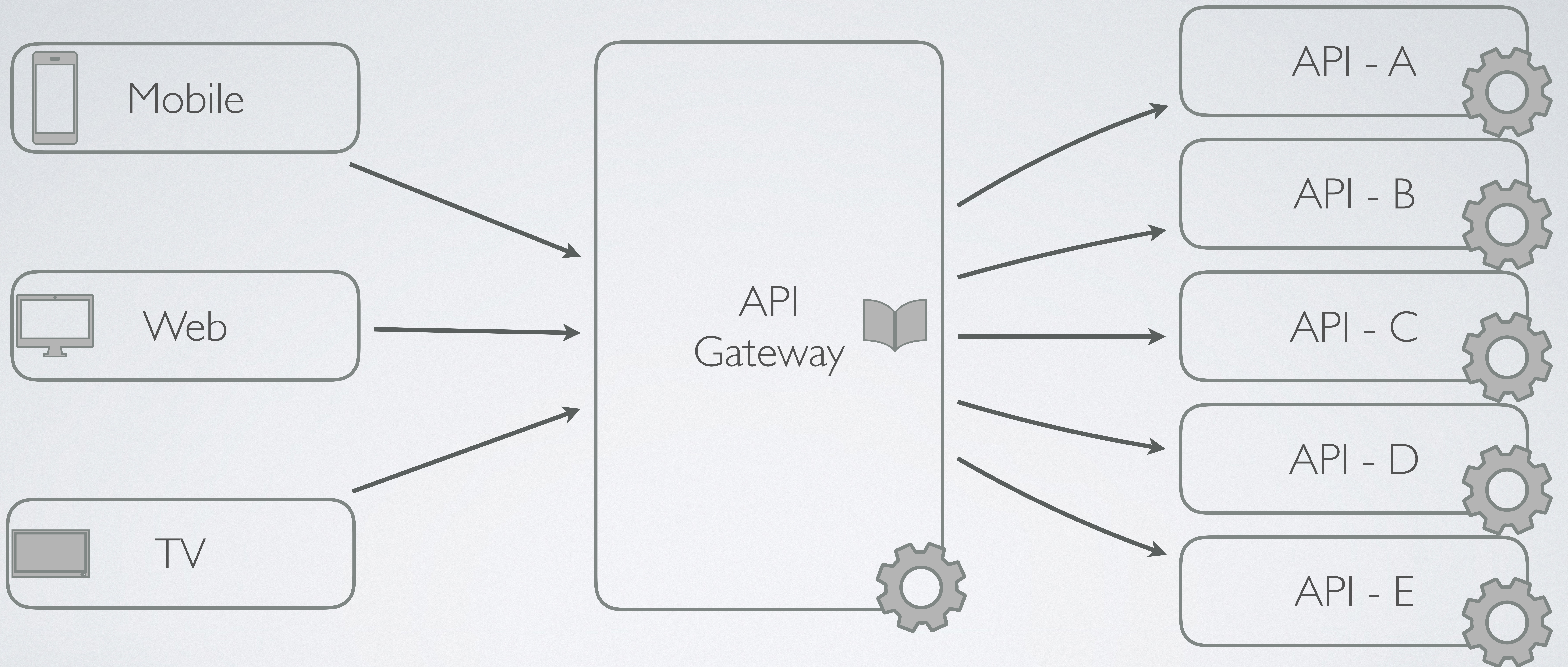
API - A

API - B

API - C


Mobile
Web
TV
API - A
API - B
API - C


Mobile
Web
TV
API
Gateway
API - A
API - B
API - C



Mobile
Web
TV
API
Gateway
API - A
API - B
API - C
API - D
API - E

Chapter 1

Technical overview

Let’s send *some* HTTP requests!

Synchronous

```
function foo1(ClientInterface $client, RequestInterface ...$requests)
{
    $entities = [];
    foreach ($requests as $request) {
        $response = $client->sendRequest($request);

        $jsonArray = json_decode(
            (string) $response->getBody(), true
        );

        $entities[] = Entity::fromArray($jsonArray);
    }
}
```

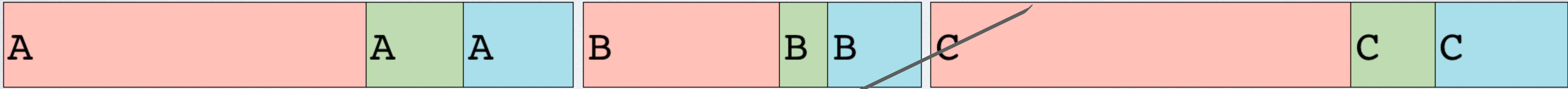
Synchronous

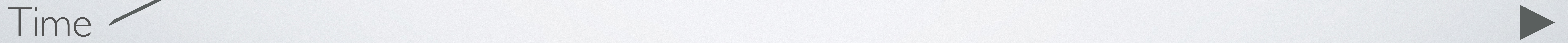
```
function foo1(ClientInterface $client, RequestInterface ...$requests)
{
    $entities = [];
    foreach ($requests as $request) {
        $response = $client->sendRequest($request);

        $jsonArray = json_decode(
            (string) $response->getBody(), true
        );

        $entities[] = Entity::fromArray($jsonArray);
    }
}
```

HTTP request

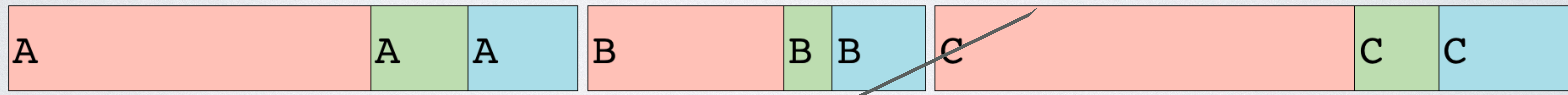
Parsing

Business logic

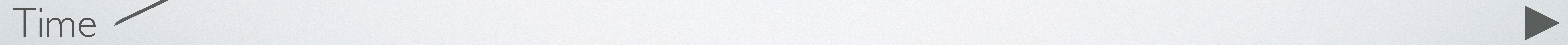

A
A
A
B
B
B
C
C
C

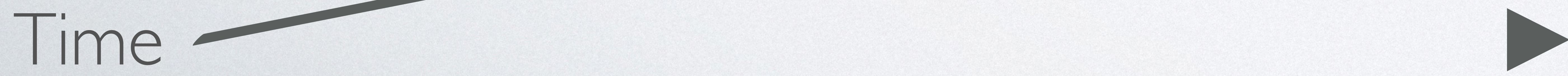

Time

Sending
+
Waiting...

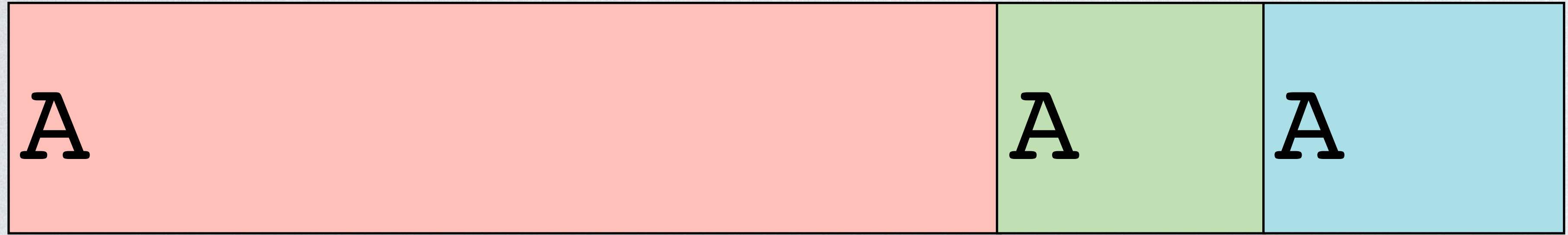
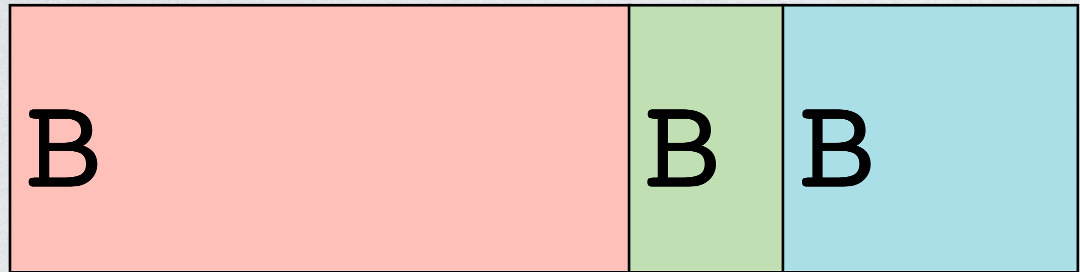
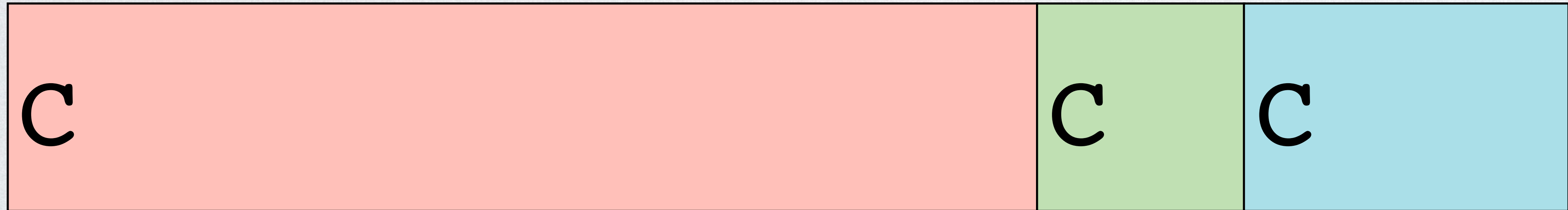


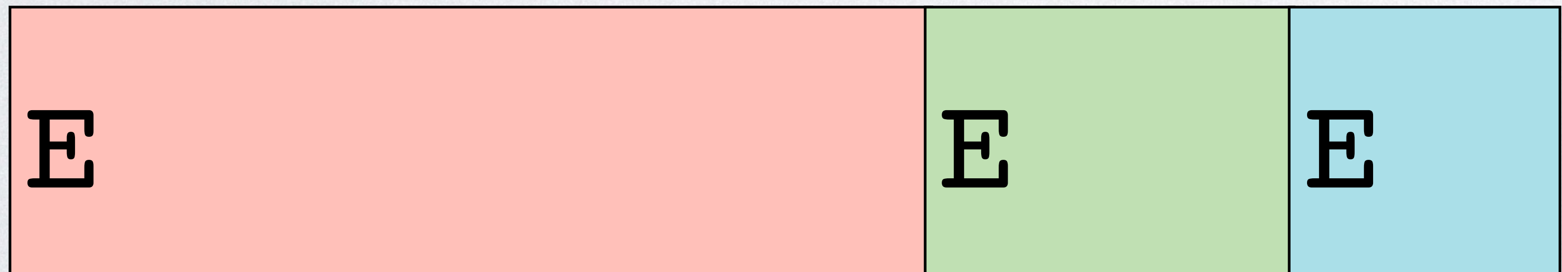
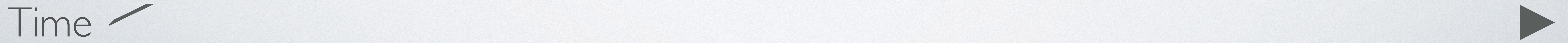
Working

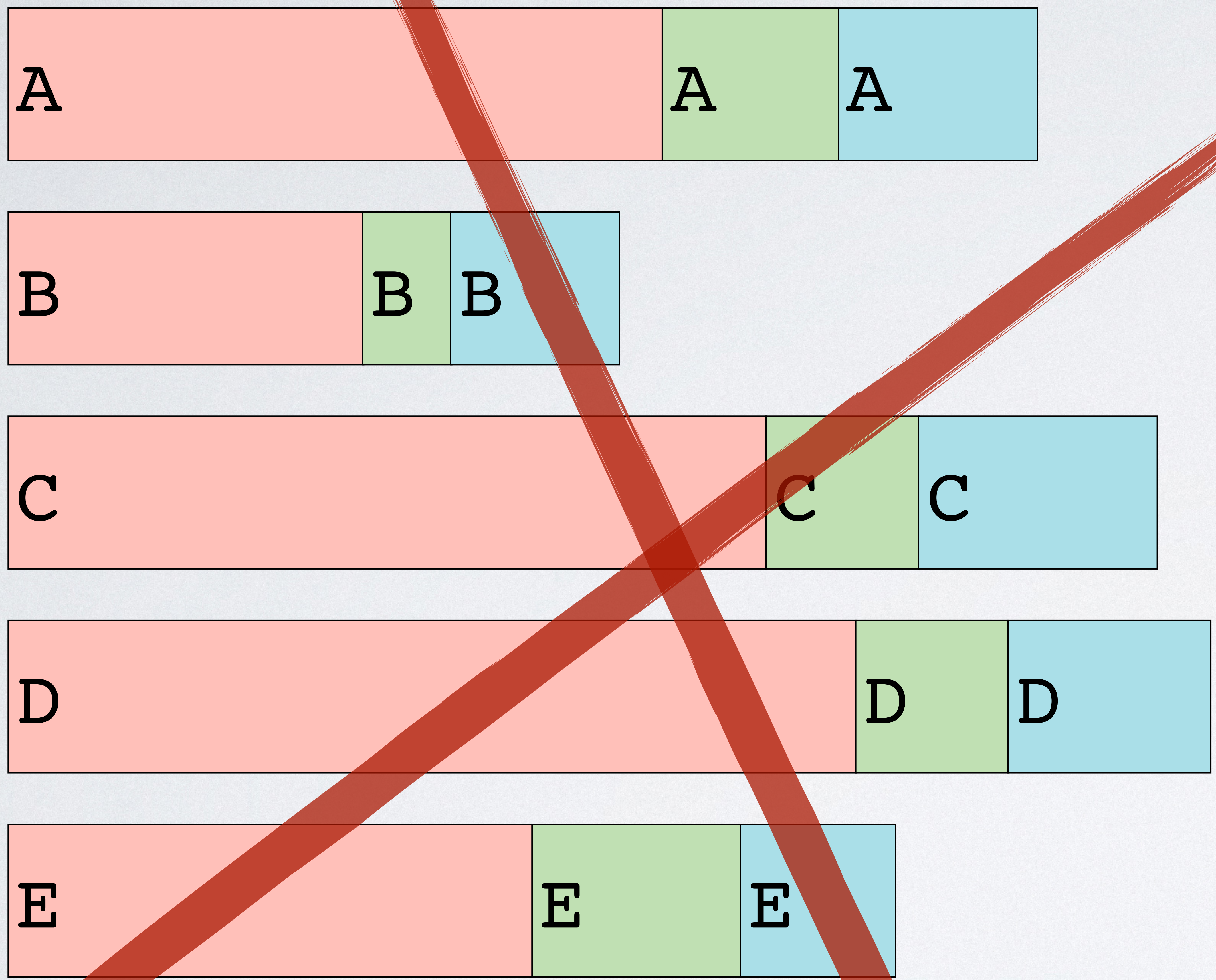


Time 

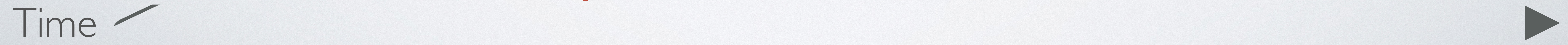
What about concurrency ?


A
A
A

B
B
B

C
C
C

D
D
D

E
E
E

Time


A
A
A
B
B
B
C
C
C
D
D
D
E
E
E

Php is Single-Threaded


Time

Asynchronous HTTP calls

Thanks to Guzzle

Asynchronous way

```
function foo2(\GuzzleHttp\Client $client, RequestInterface ...$requests)
{
    $promises = [];
    foreach ($requests as $request) {
        $promises[] = $client->sendAsync($request);
    }

    $entities = [];
    foreach (\GuzzleHttp\Promise\unwrap($promises) as $response) {
        $jsonArray = json_decode(
            (string) $response->getBody(), true
        );

        $entities[] = Entity::fromArray($jsonArray);
    }
}
```

Asynchronous way

```
function foo2(\GuzzleHttp\Client $client, RequestInterface ...$requests)
{
    $promises = [];
    foreach ($requests as $request) {
        $promises[] = $client->sendAsync($request);
    }

    $entities = [];
    foreach (\GuzzleHttp\Promise\unwrap($promises) as $response) {
        $jsonArray = json_decode(
            (string) $response->getBody(), true
        );

        $entities[] = Entity::fromArray($jsonArray);
    }
}
```

Sending

Asynchronous way

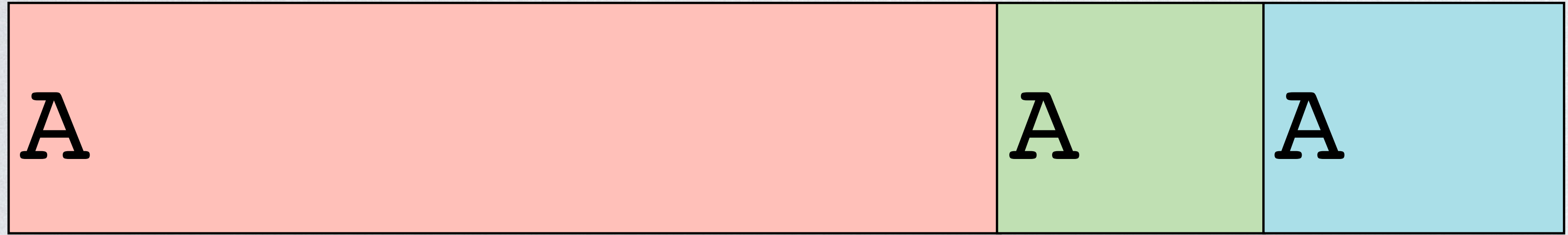
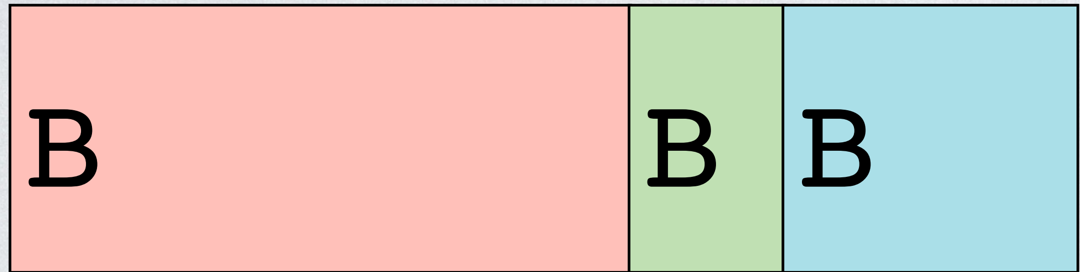
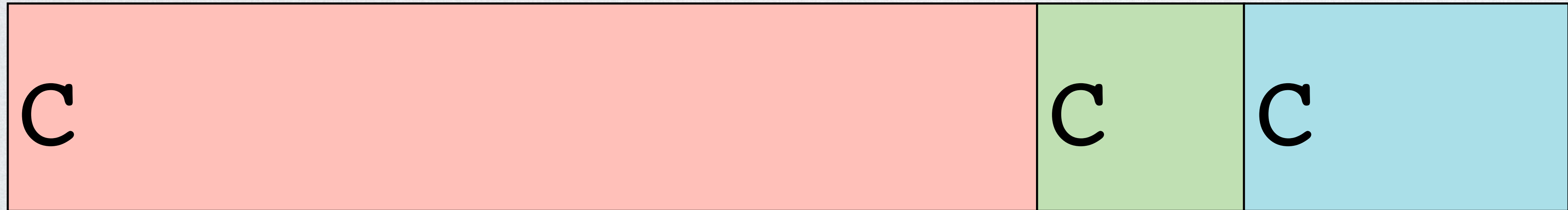
```
function foo2(\GuzzleHttp\Client $client, RequestInterface ...$requests)
{
    $promises = [];
    foreach ($requests as $request) {
        $promises[] = $client->sendAsync($request);
    }

    $entities = [];
    foreach (\GuzzleHttp\Promise\unwrap($promises) as $response) {
        $jsonArray = json_decode(
            (string) $response->getBody(), true
        );

        $entities[] = Entity::fromArray($jsonArray);
    }
}
```

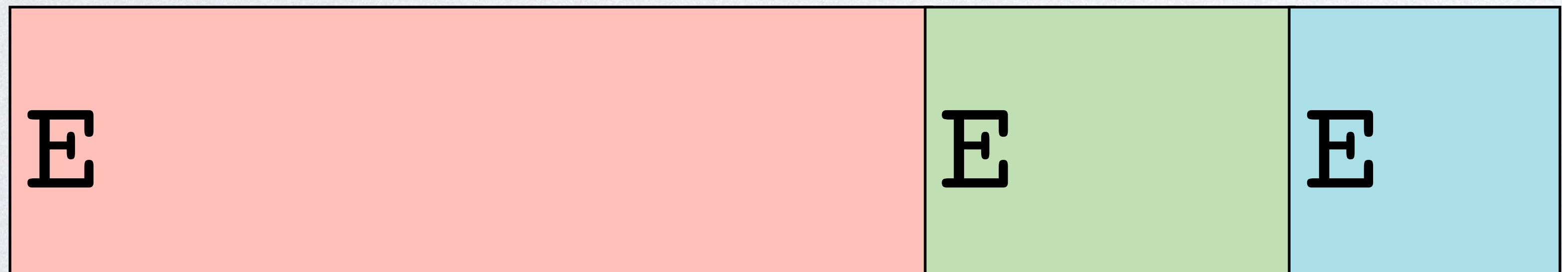
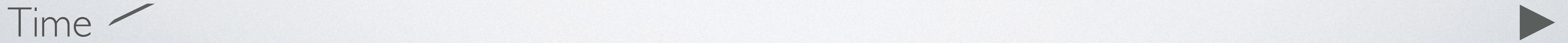
Waiting...

Asynchronous way

```
function foo2(\GuzzleHttp\Client $client, RequestInterface ...$requests)
{
    $promises = [];
    foreach ($requests as $request) {
        $promises[] = $client->sendAsync($request);
    }

    $entities = [];
    foreach (\GuzzleHttp\Promise\unwrap($promises) as $response) {
        $jsonArray = json_decode(
            (string) $response->getBody(), true
        );

        $entities[] = Entity::fromArray($jsonArray);
    }
}
```

Working


A
A
A

B
B
B

C
C
C

D
D
D

E
E
E

Time

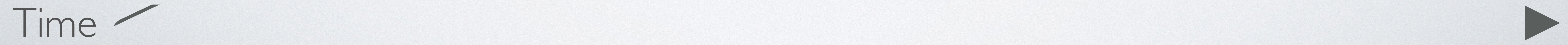
A A A

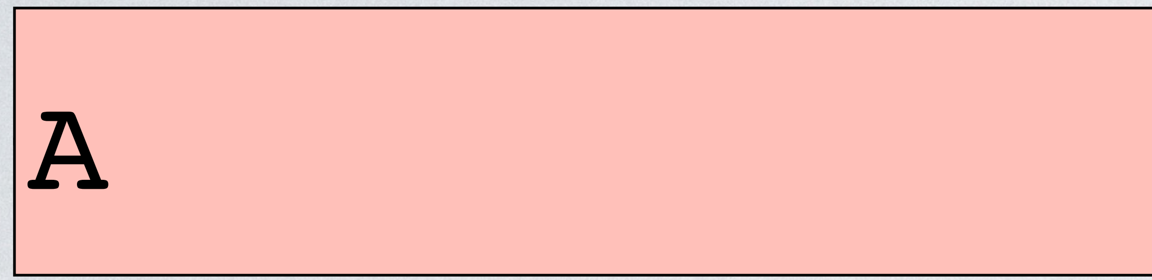
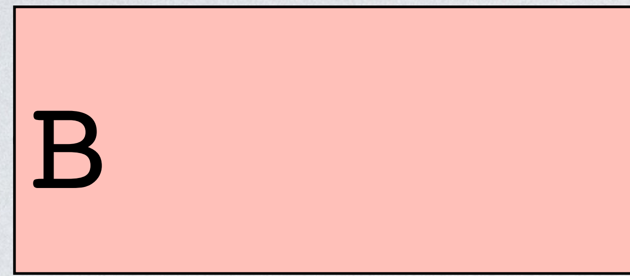
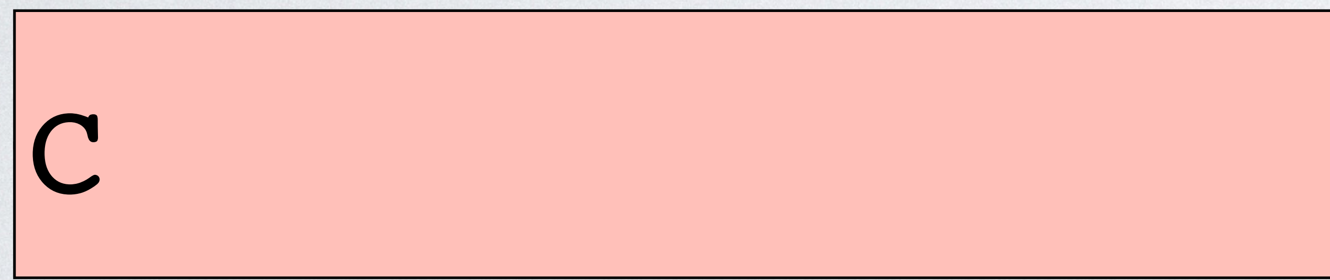
B B B

C C C

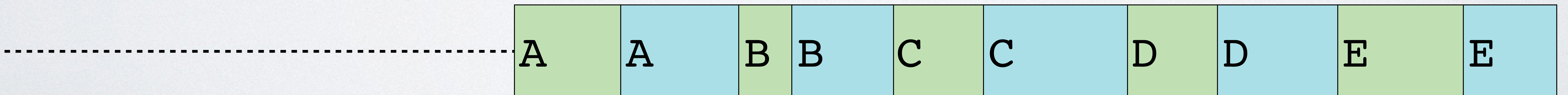
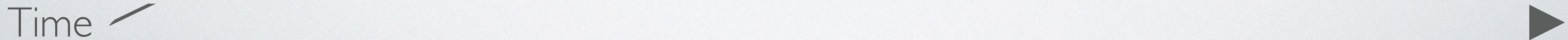
D D D

E E E




A

B

C

D

E

A
A
B
B
C
C
D
D
E
E

Time

A

B

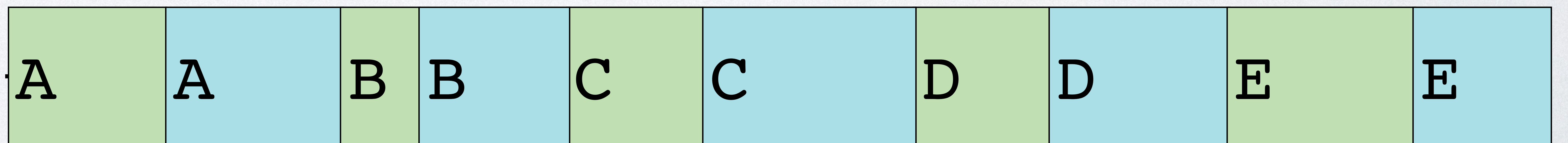
C

D

E

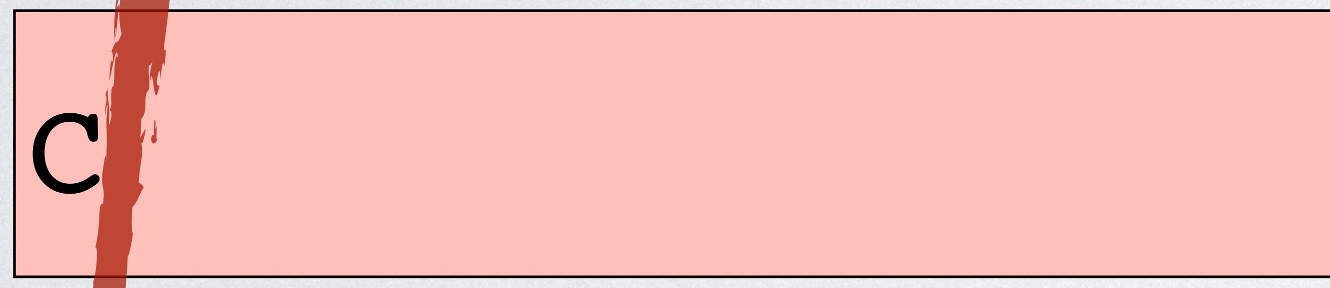
NOT PHP

PHP



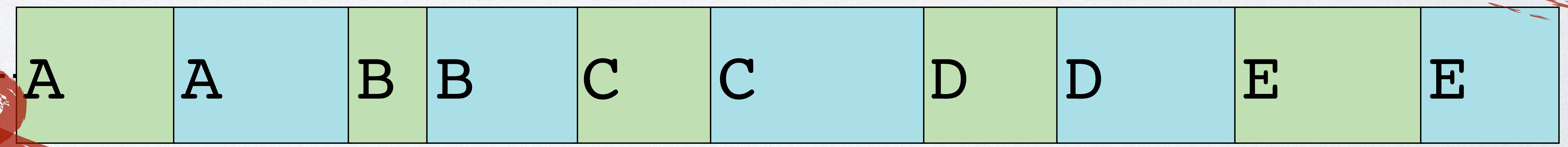
Time

Sending



Working

Waiting...



Time

We can do even better!

Promises

```php
function foo3(\GuzzleHttp\Client $client, RequestInterface ...$requests)
{
    $promises = [];
    foreach ($requests as $request) {
        $promises[] = $client->sendAsync($request)->then(
            function (ResponseInterface $response) {
                $jsonArray = json_decode(
                    (string) $response->getBody(), true
                );

                return Entity::fromArray($jsonArray);
            }
        );
    }

    $entities = \GuzzleHttp\Promise\unwrap($promises);
```

Promises

```
function foo3(\GuzzleHttp\Client $client, RequestInterface ...$requests)
{
    $promises = [];
    foreach ($requests as $request) {
        $promises[] = $client->sendAsync($request)->then(
            function (ResponseInterface $response) {
                $jsonArray = json_decode(
                    (string) $response->getBody(), true
                );

                return Entity::fromArray($jsonArray);
            }
        );
    }

    $entities = \GuzzleHttp\Promise\unwrap($promises);
```

Sending

Promises

```
function foo3(\GuzzleHttp\Client $client, RequestInterface ...$requests)
{
    $promises = [];
    foreach ($requests as $request) {
        $promises[] = $client->sendAsync($request)->then(
            function (ResponseInterface $response) {
                $jsonArray = json_decode(
                    (string) $response->getBody(), true
                );

                return Entity::fromArray($jsonArray);
            }
        );
    }

    $entities = \GuzzleHttp\Promise\unwrap($promises);
```

Waiting
1 Request...

Promises

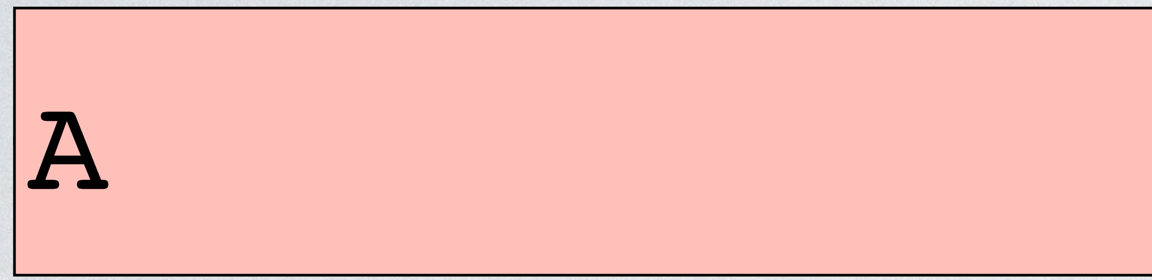
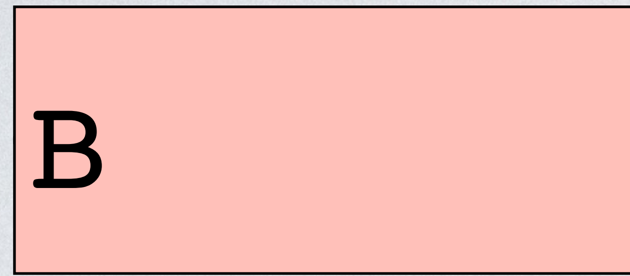
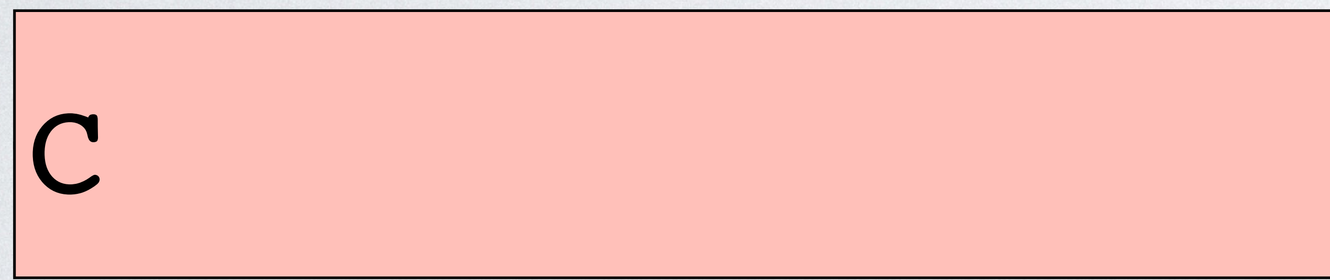
```
function foo3(\GuzzleHttp\Client $client, RequestInterface ...$requests)
{
    $promises = [];
    foreach ($requests as $request) {
        $promises[] = $client->sendAsync($request)->then(
            function (ResponseInterface $response) {
                $jsonArray = json_decode(
                    (string) $response->getBody(), true
                );

                return Entity::fromArray($jsonArray);
            }
        );
    }

    $entities = \GuzzleHttp\Promise\unwrap($promises);
```

Working

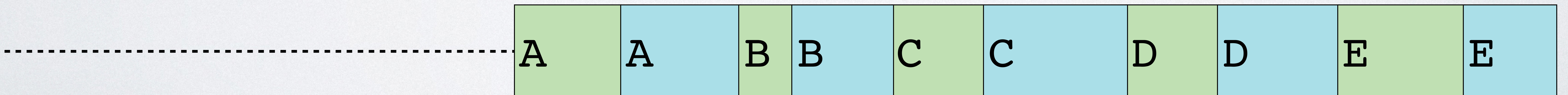
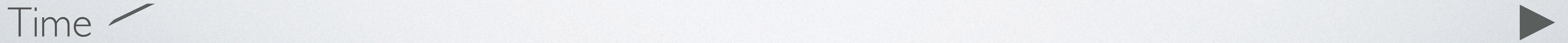
Promises

```php
function foo3(\GuzzleHttp\Client $client, RequestInterface ...$requests)
{
    $promises = [];
    foreach ($requests as $request) {
        $promises[] = $client->sendAsync($request)->then(
            function (ResponseInterface $response) {
                $jsonArray = json_decode(
                    (string) $response->getBody(), true
                );

                return Entity::fromArray($jsonArray);
            }
        );
    }

    $entities = \GuzzleHttp\Promise\unwrap($promises);
```

Waiting all the jobs...


A

B

C

D

E

A
A
B
B
C
C
D
D
E
E

Time

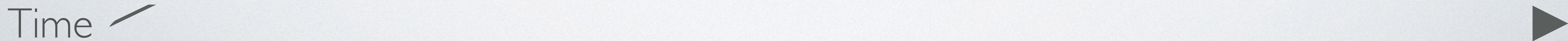
A A A

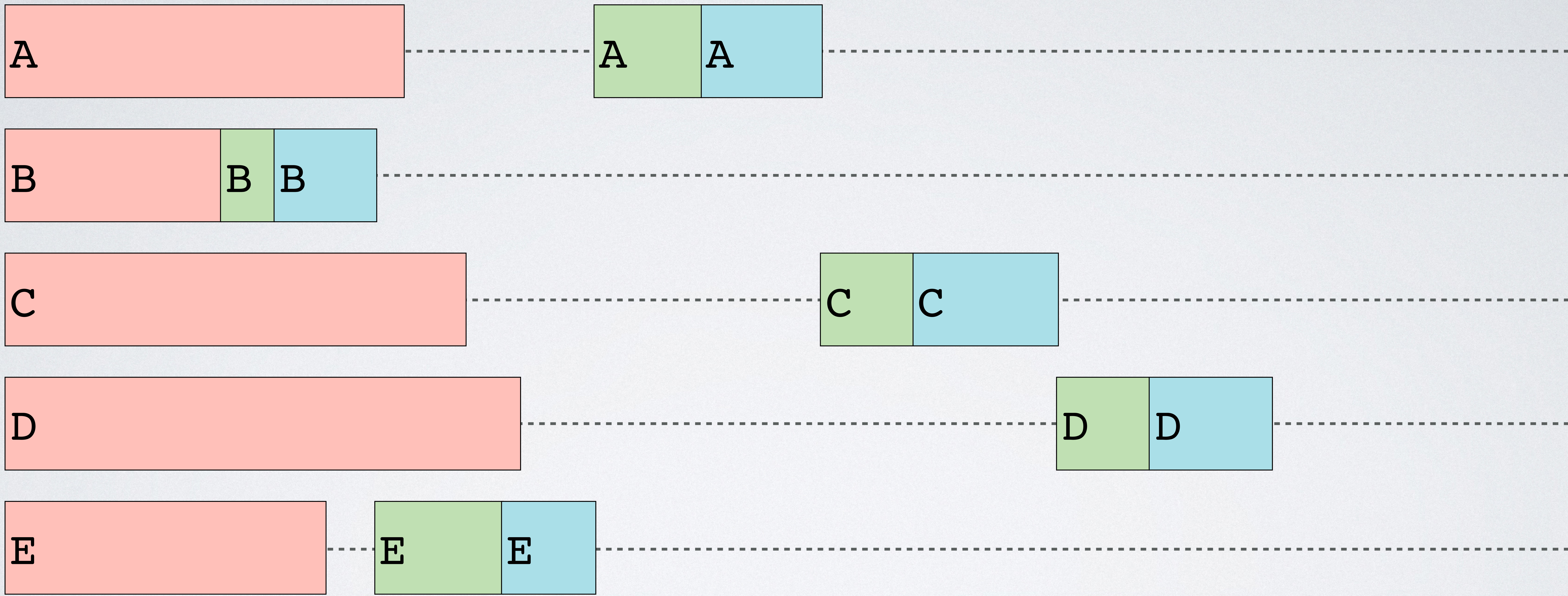
B B B

C C C

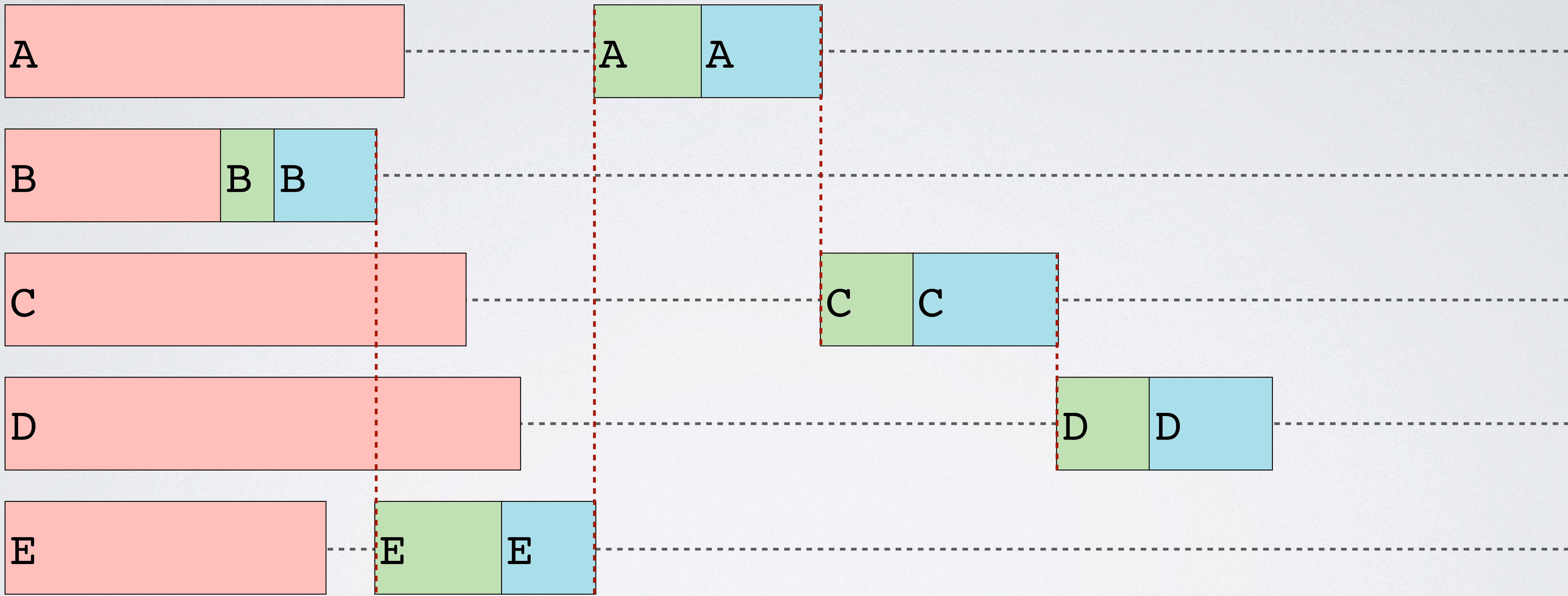
D D D

E E E





Time



Time

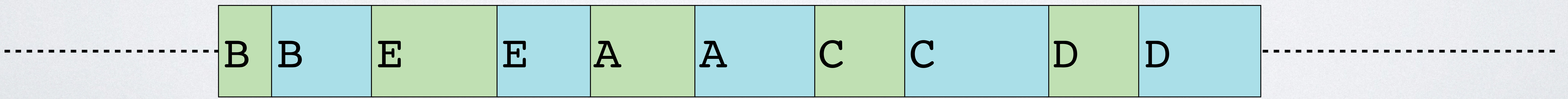
A

B

C

D

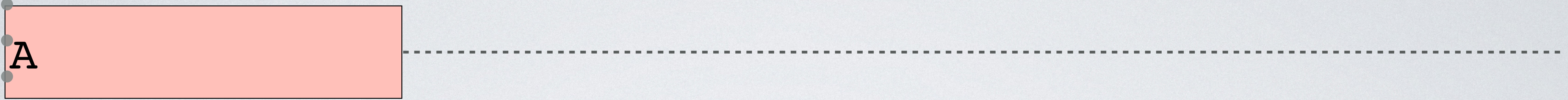
E

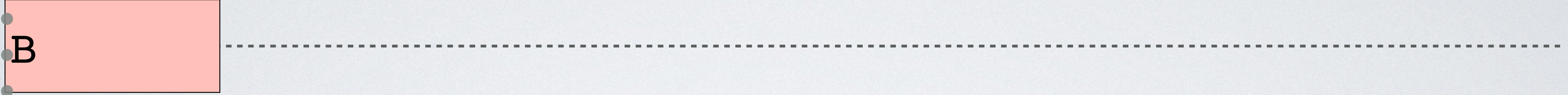


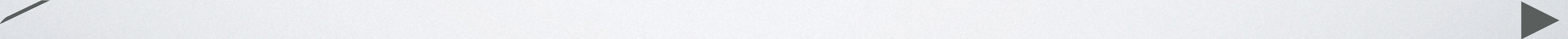
Time

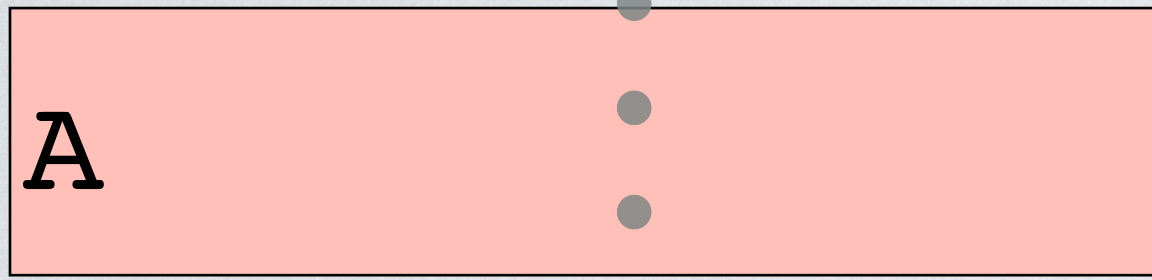
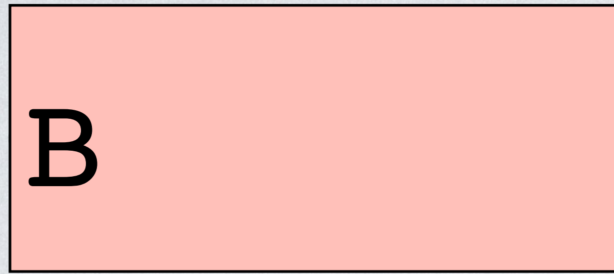
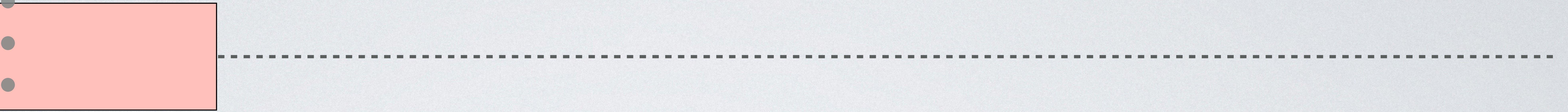
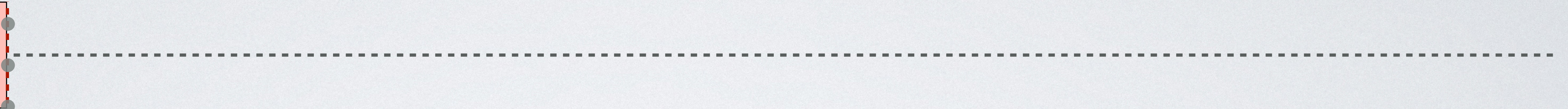
ASYNCHRONOUS PROGRAMMING

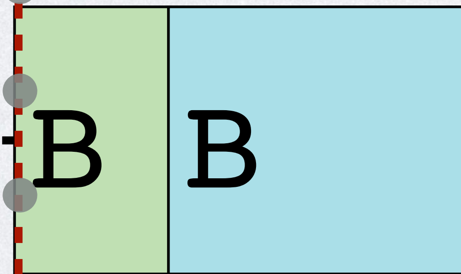
PHP

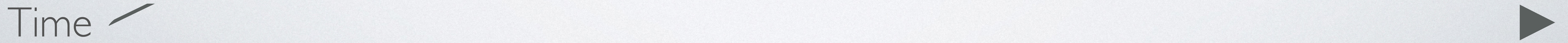




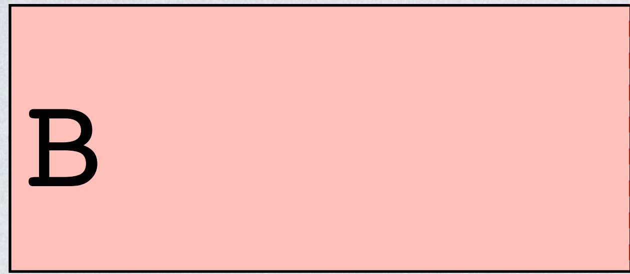


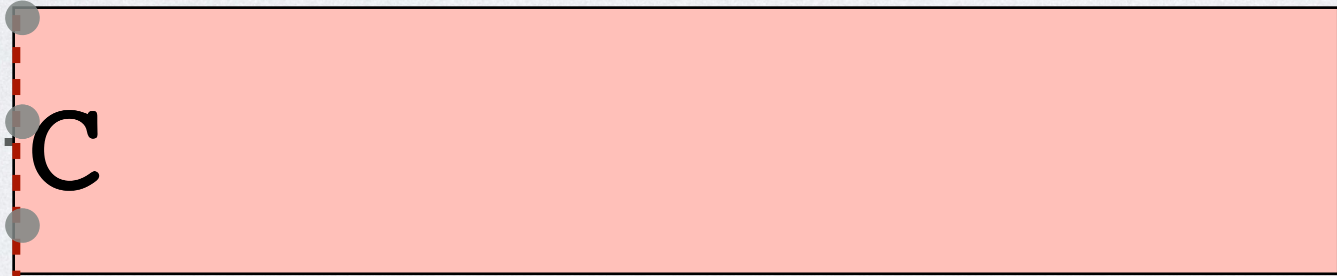

A

B



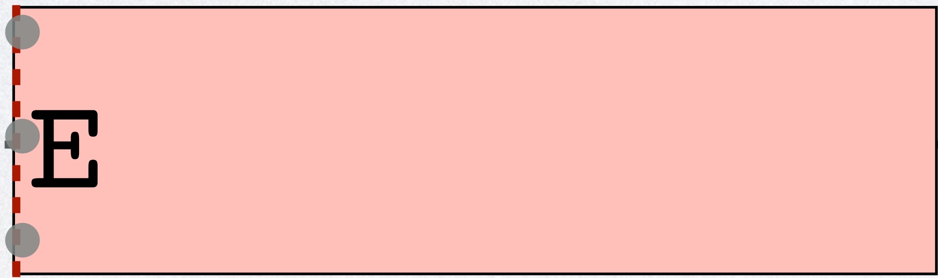
PHP

B
B

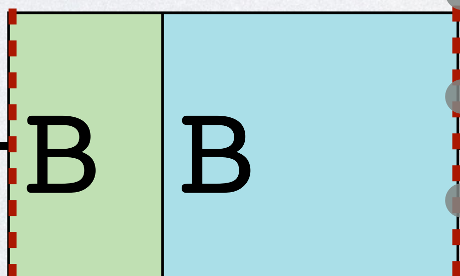

Time

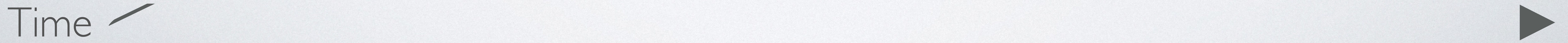
PHP



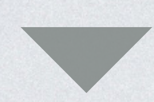








PHP



A

B

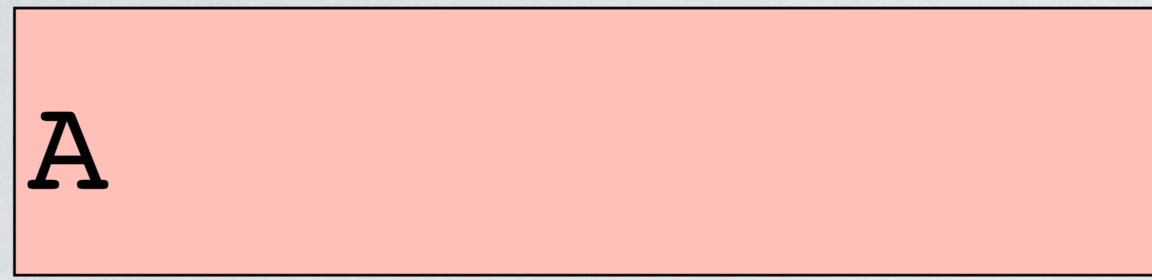
C

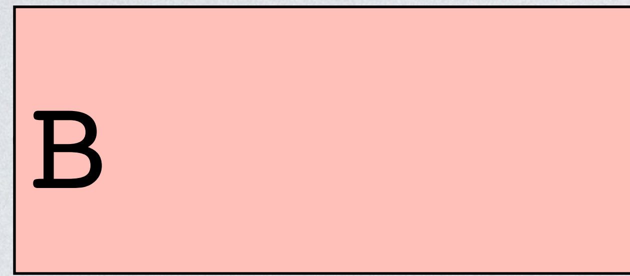
E

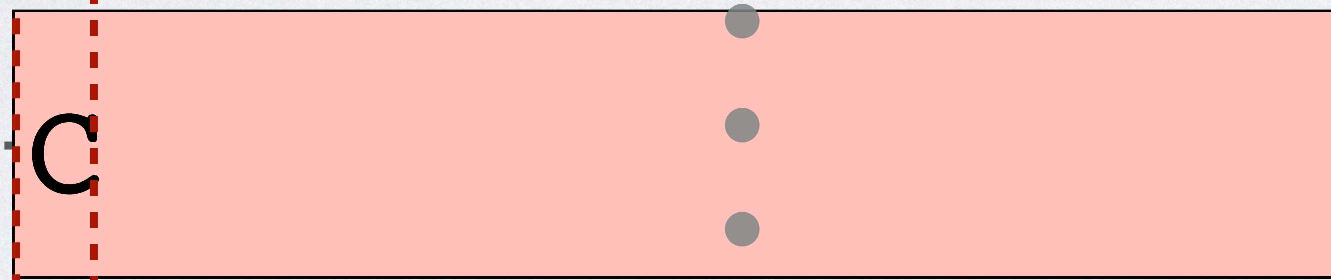
B B A A

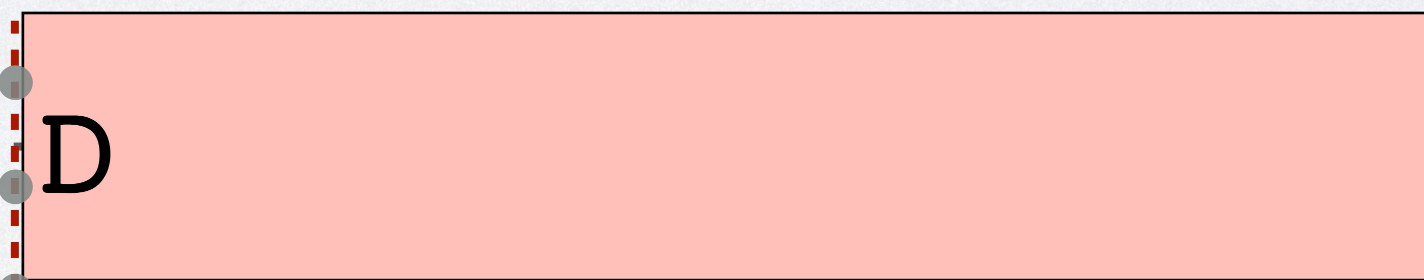
Time

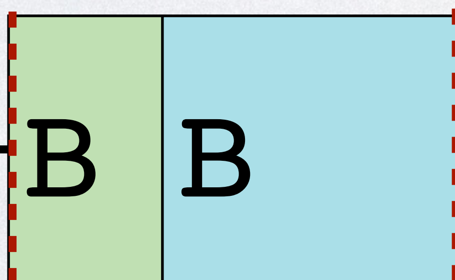
PHP

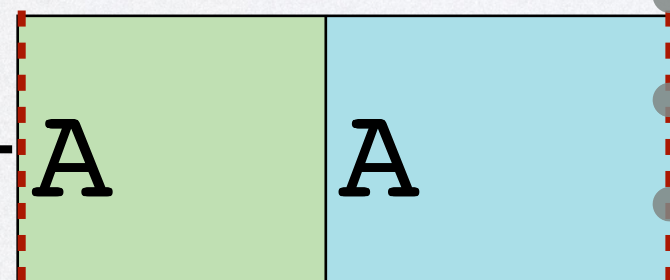


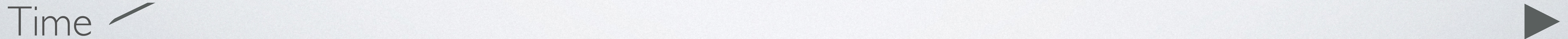






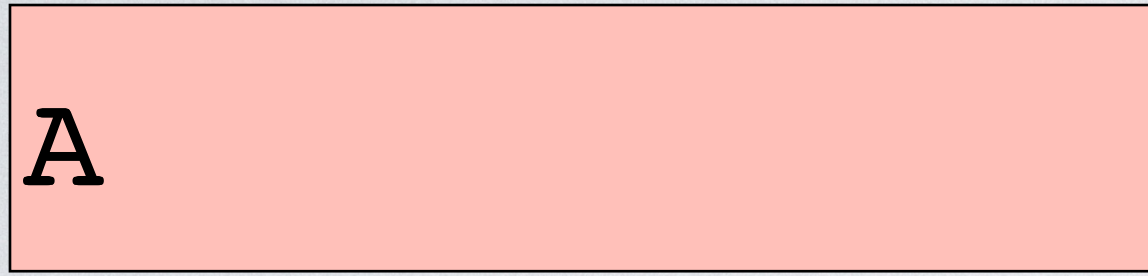


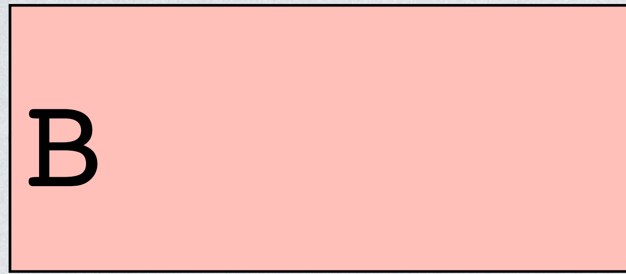


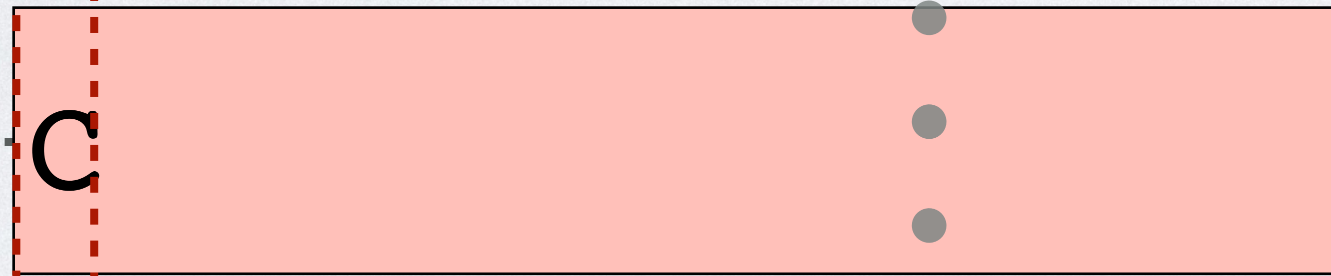


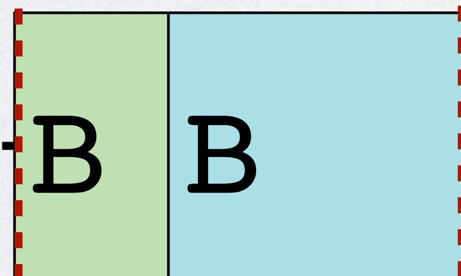
Time

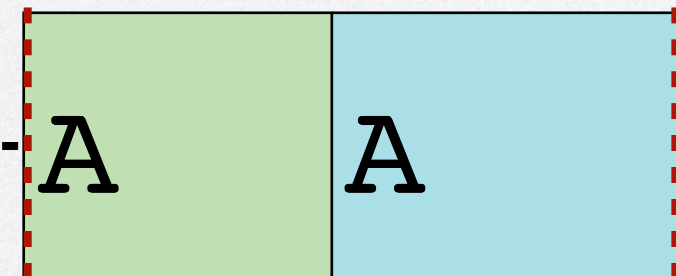
PHP

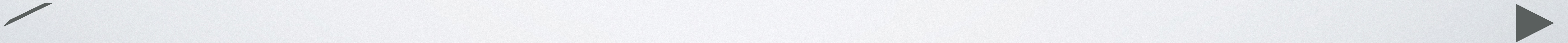






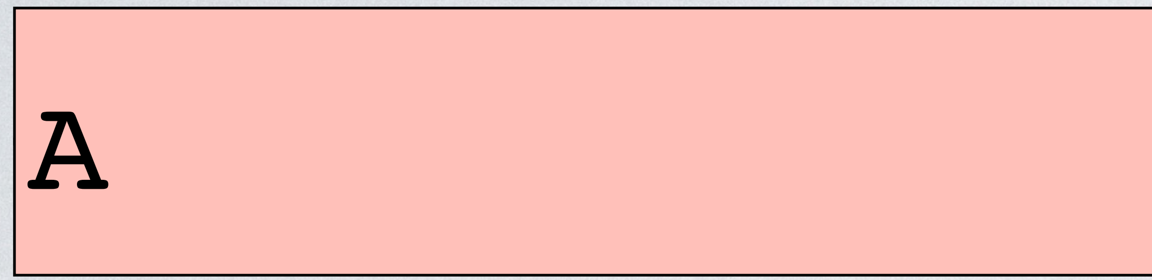


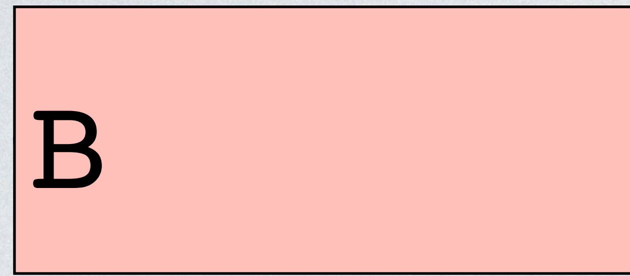


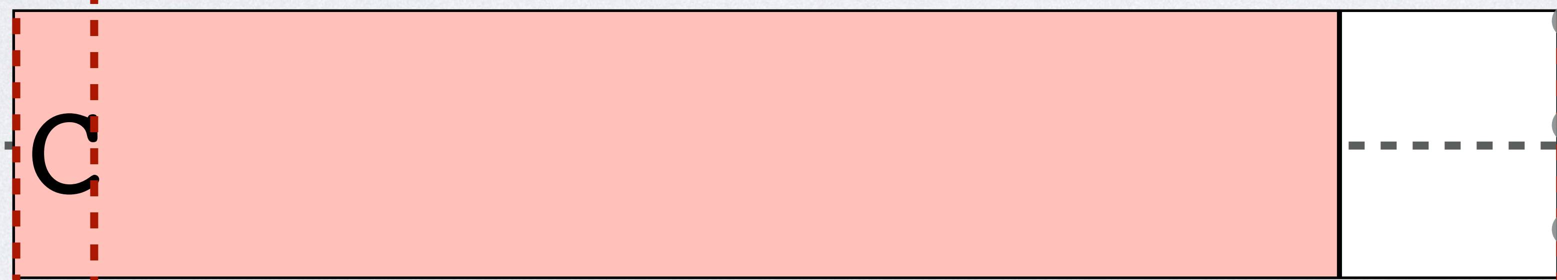


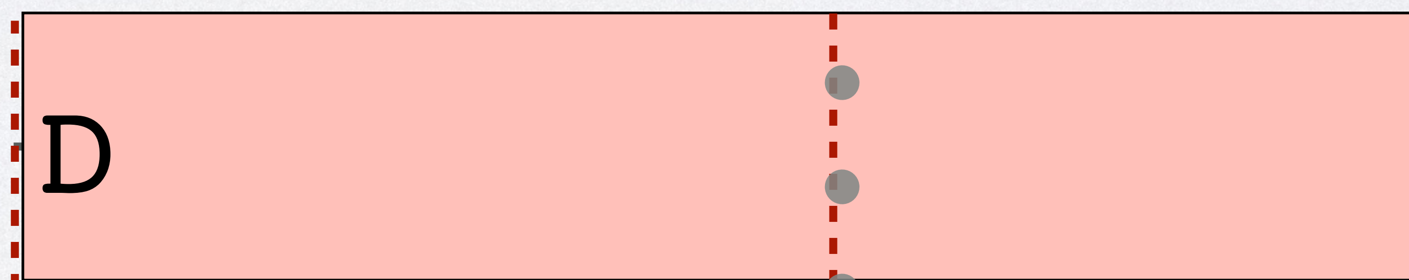
Time

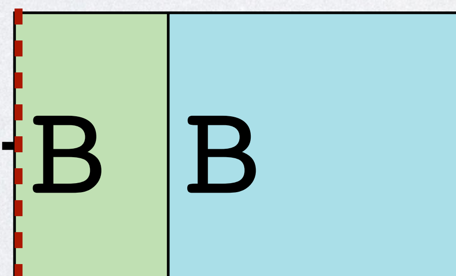
PHP

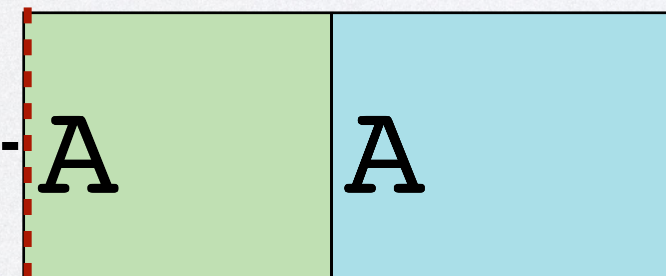


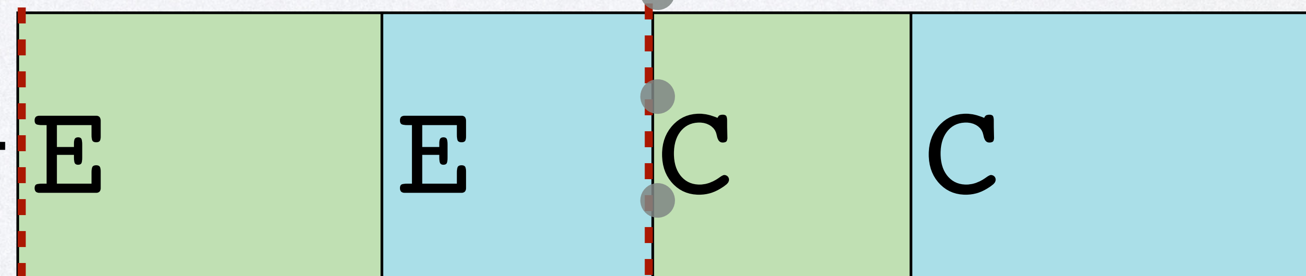


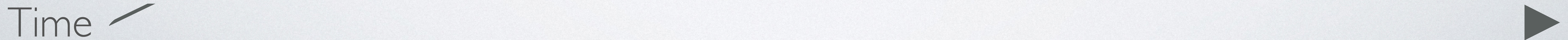




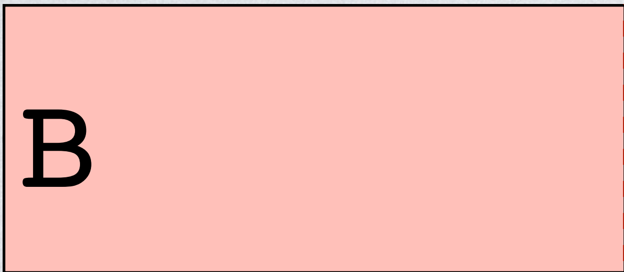


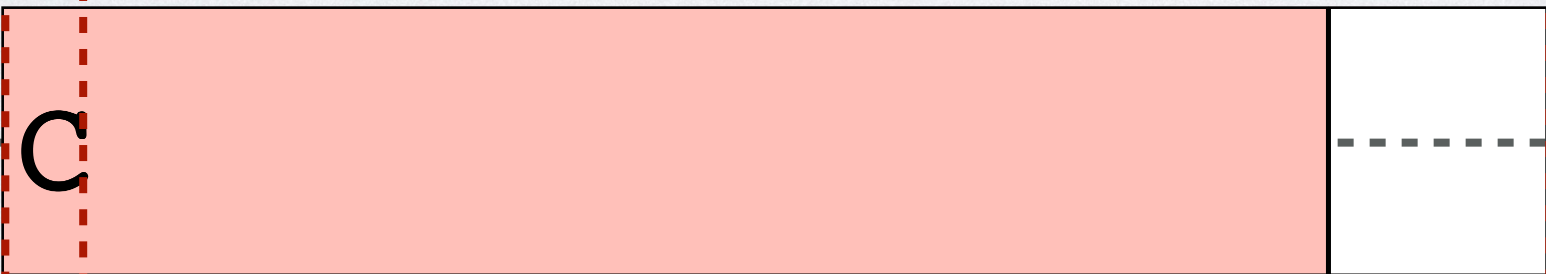


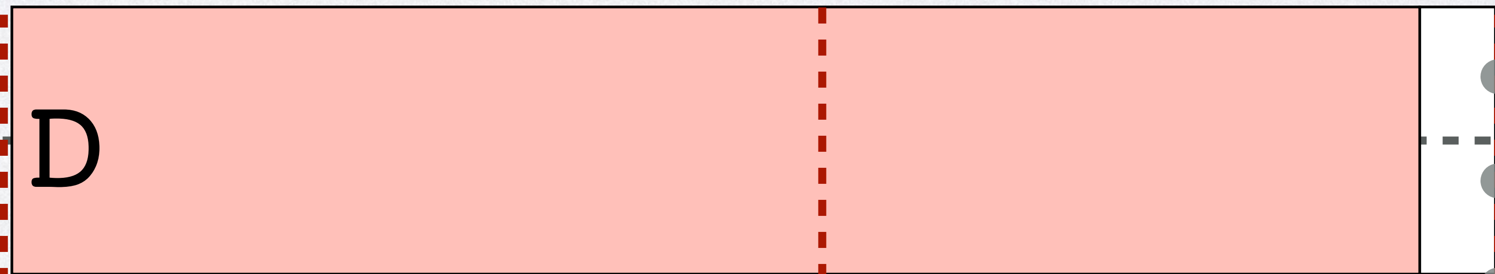


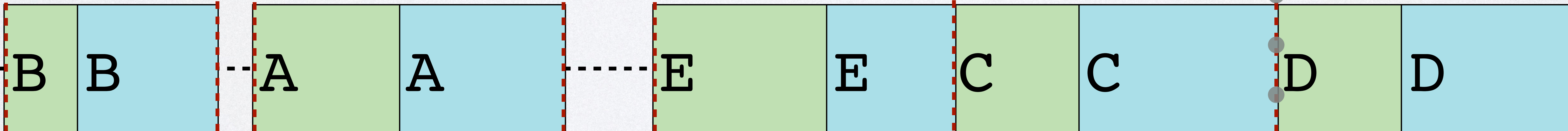


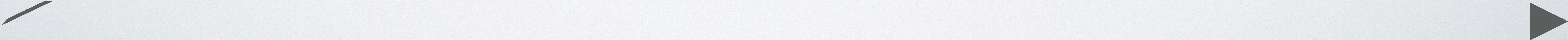
PHP







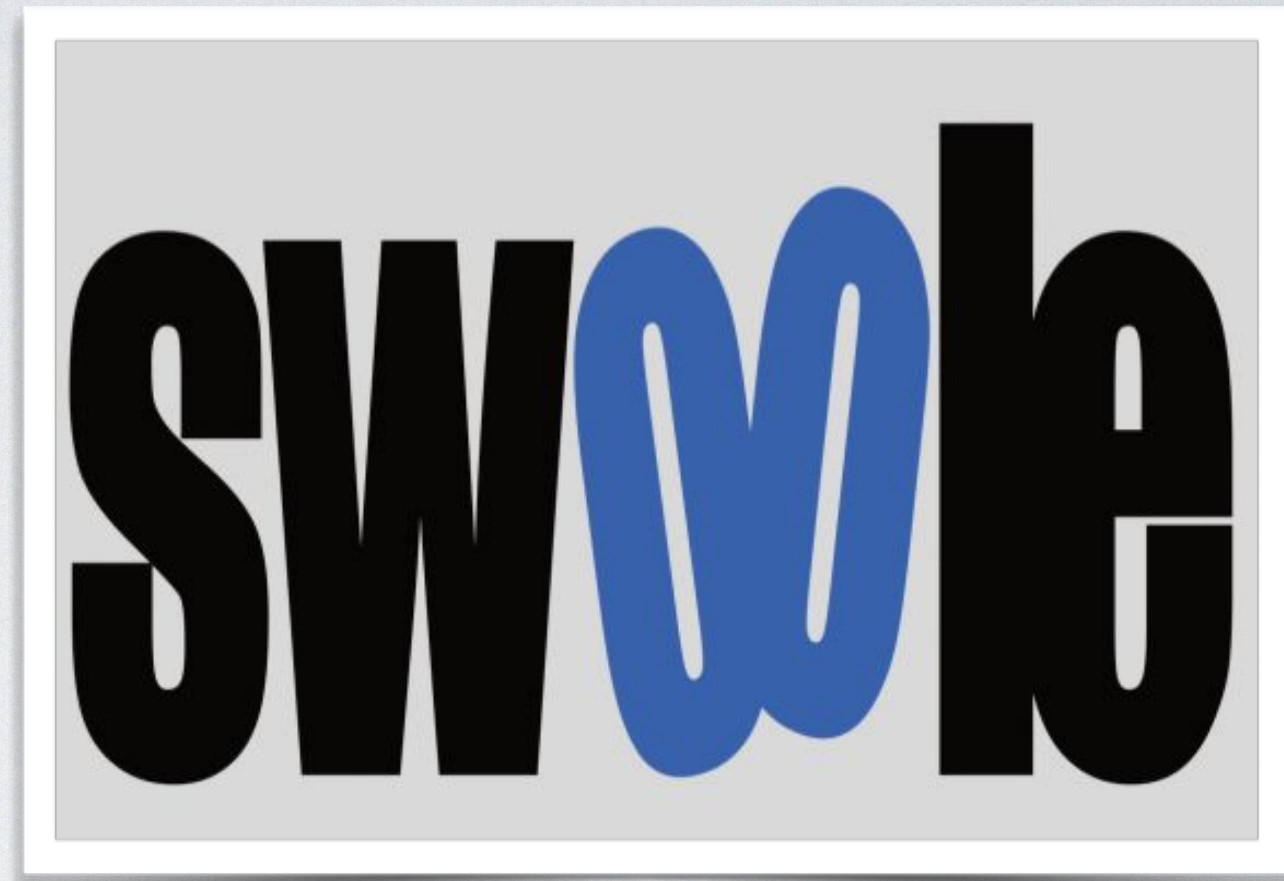


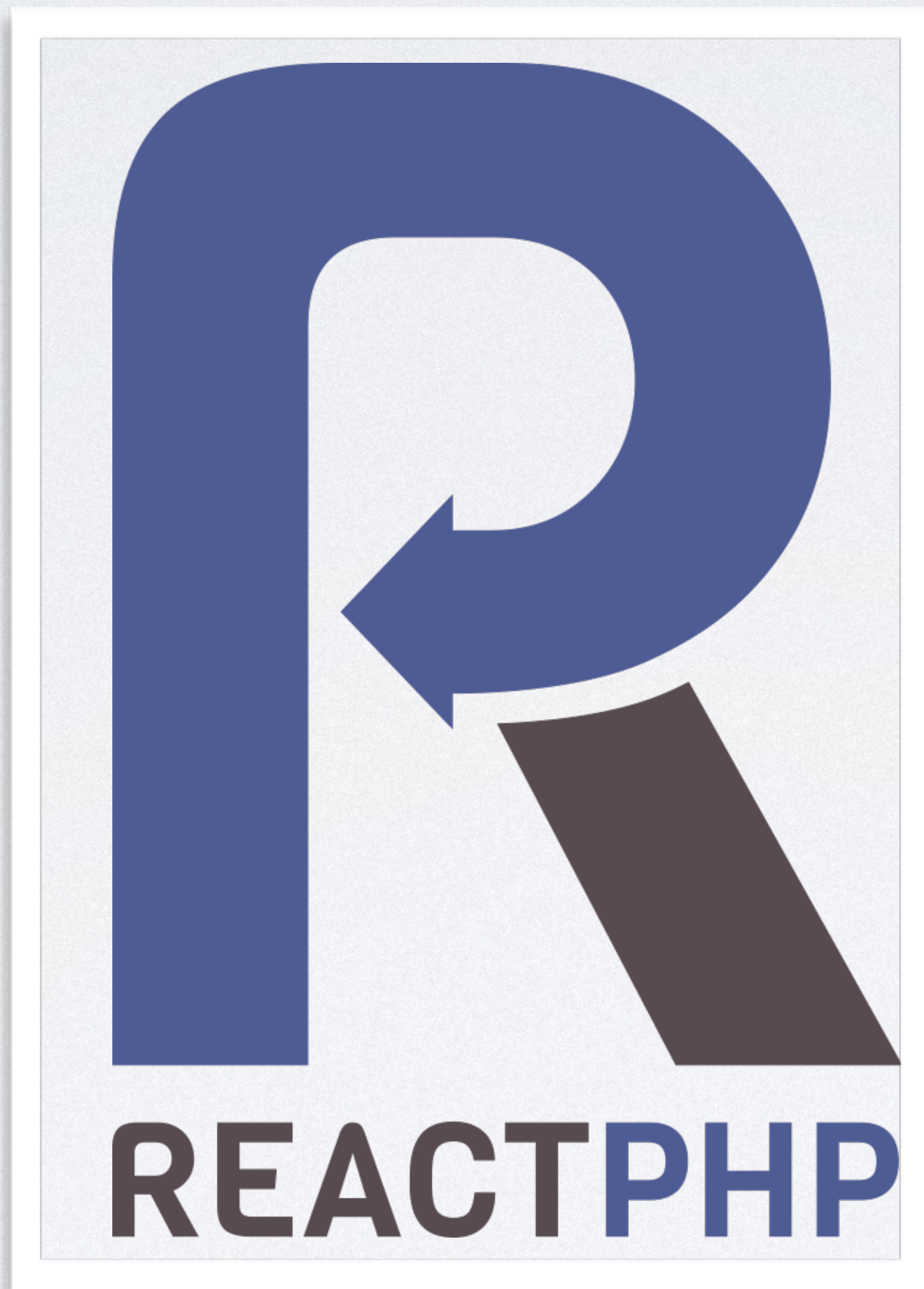


Asynchronous programming goal is to reduce **useless wait**.

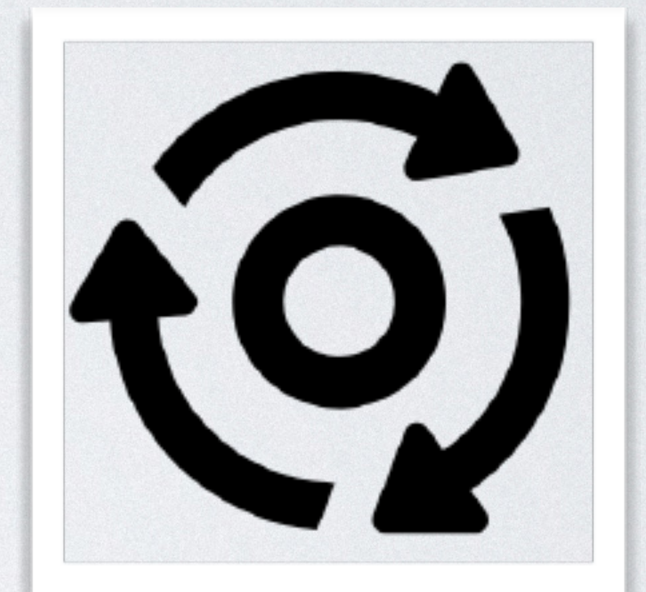
Which framework can I use?

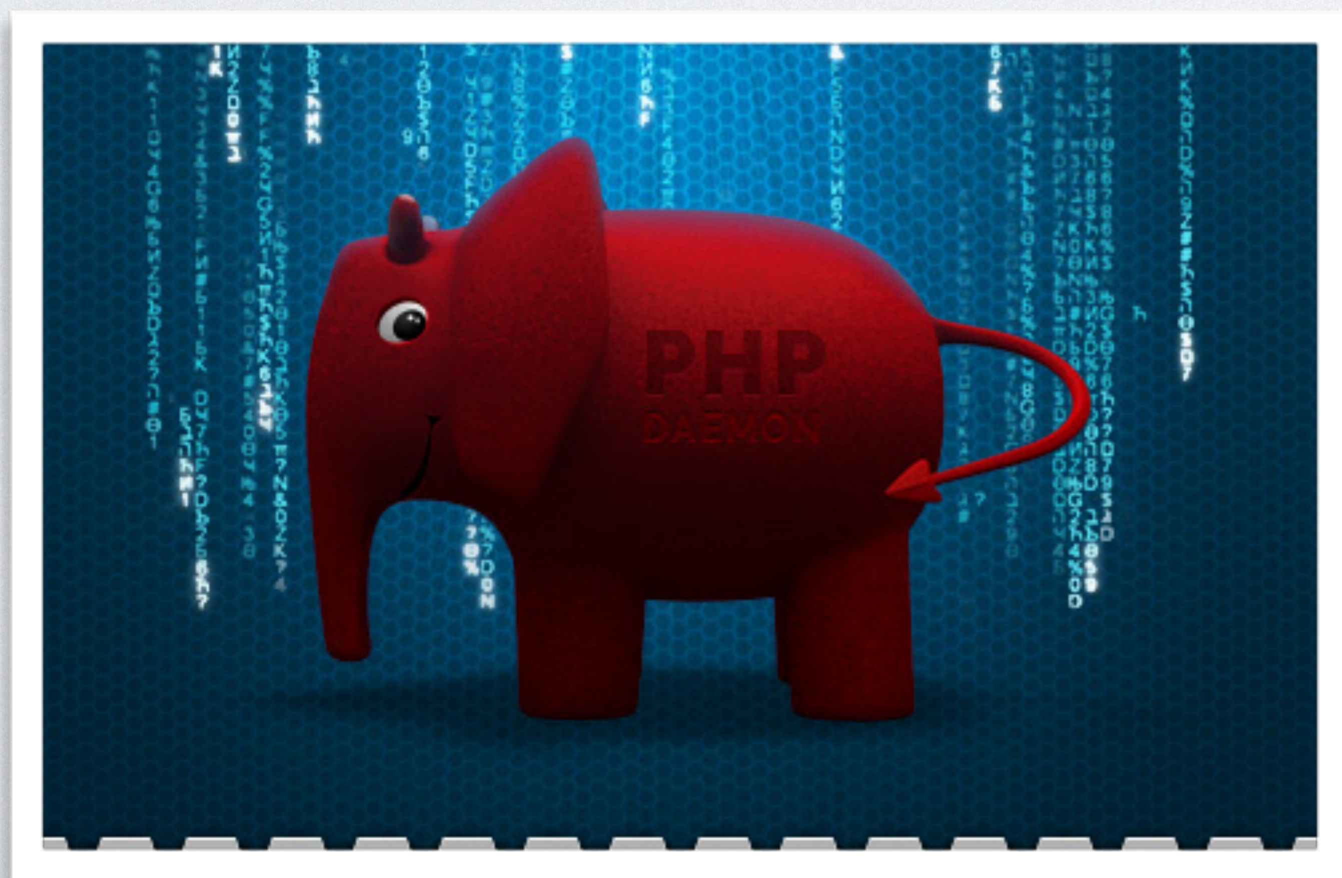
Kraken



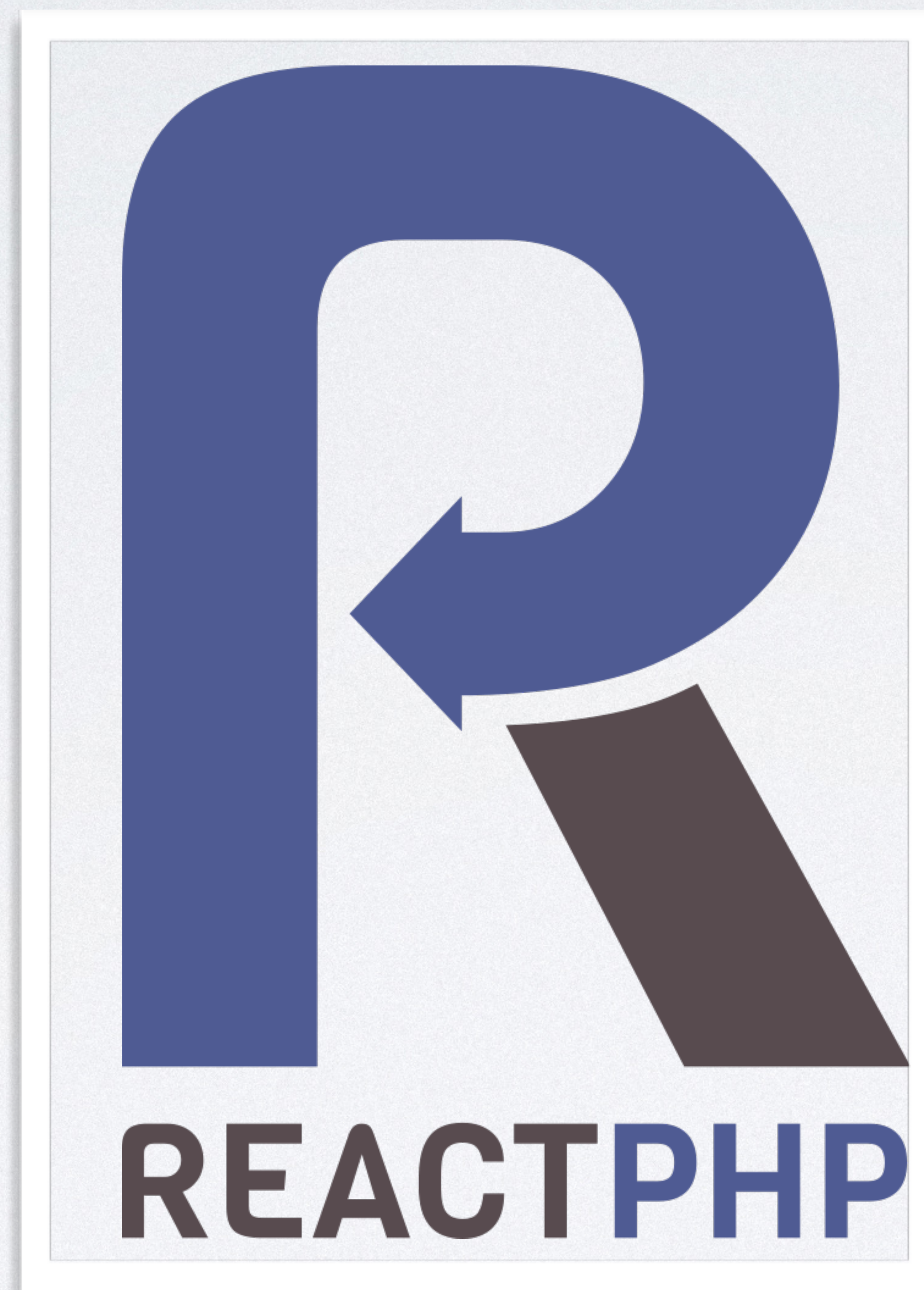


Amp


Recoil



PhpDaemon



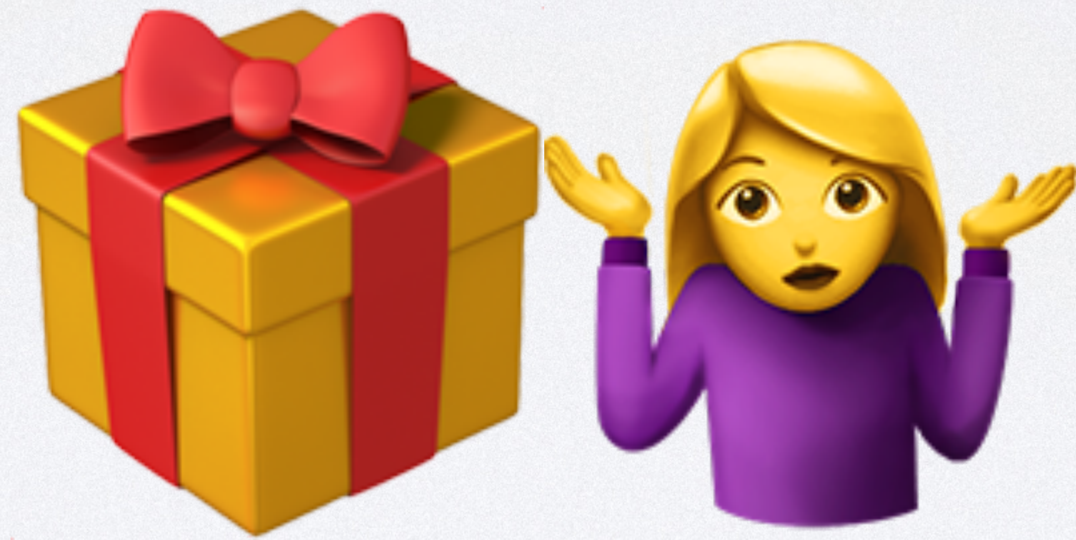
Amp

Extensions

- ext-event
- ext-ev
- ext-uv
- ext-libevent
- ext-libev

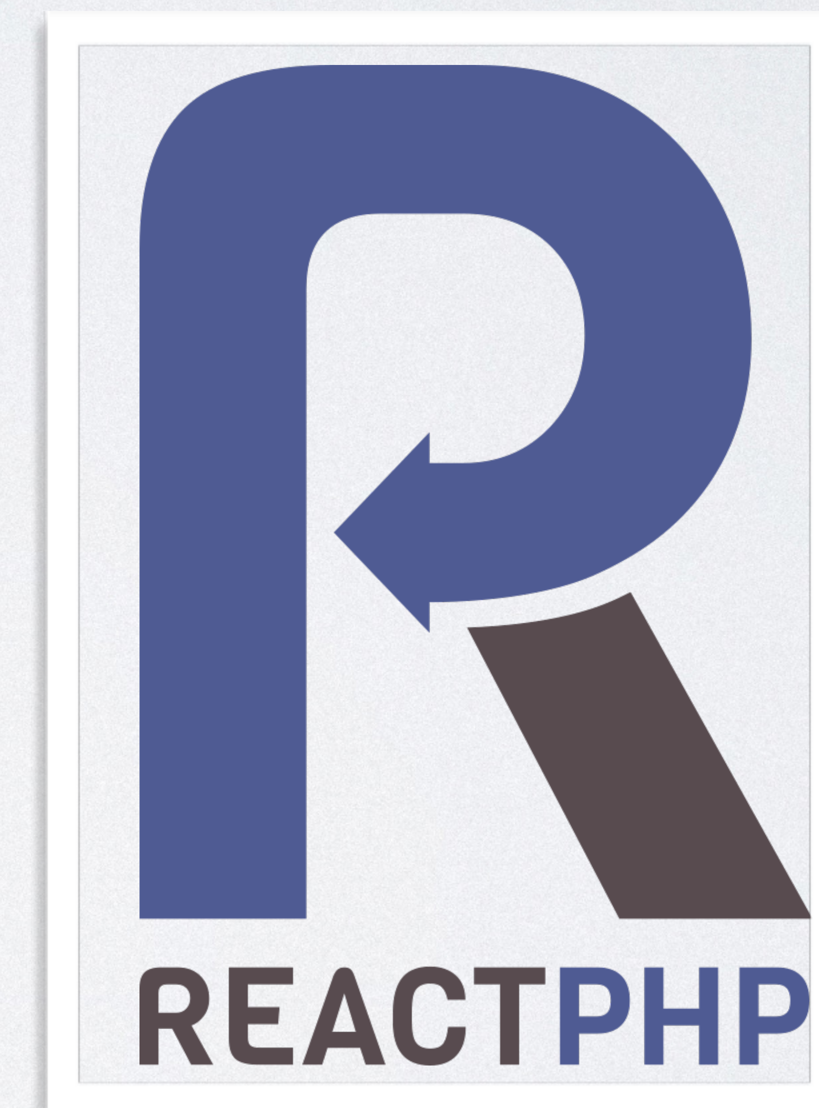
SO MANY CHOICES

Promises



```php
echo 'A';
$promise = asyncFunction();

$promise->then(function () {
    echo 'B';

    return asyncFunction();

})->then(function () {
    echo 'C';
});
```

Thenable



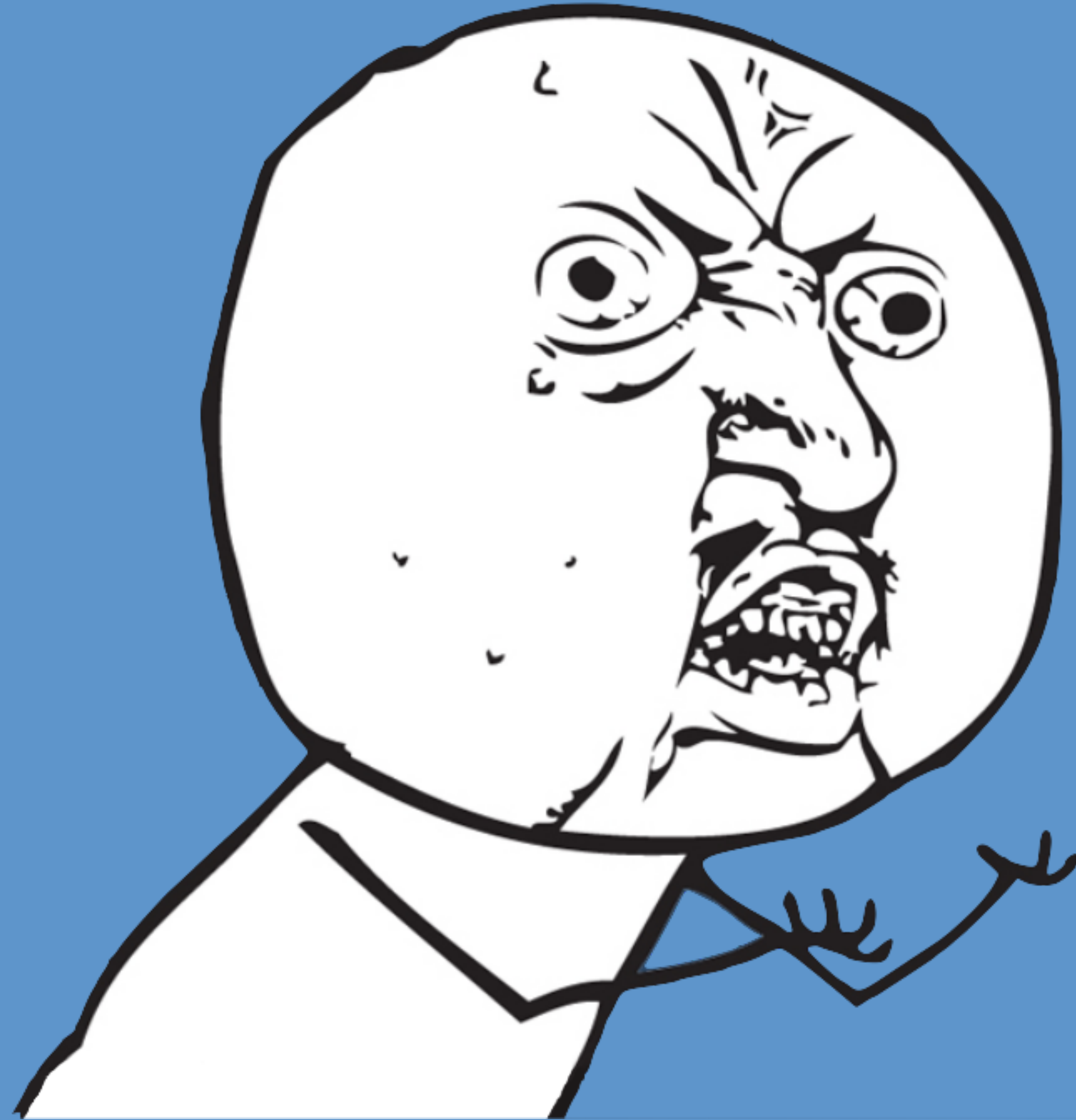
Yieldable

```php
echo 'A';
$promise = asyncFunction();
yield $promise;

echo 'B';

yield asyncFunction();

echo 'C';
```

```php
echo 'A';
$promise = asyncFunction();

$promise->then(function () {
    echo 'B';

    return asyncFunction();

})->then(function () {
    echo 'C';
});
```

```php
echo 'A';
$promise = asyncFunction();
yield $promise;

echo 'B';

yield asyncFunction();

echo 'C';
```

To asynchronously *wait* a **promise**'s value, `yield` it.



ASYNCHRONOUS PROGRAMMING SHOULD BE A DETAIL OF IMPLEMENTATION

A synchronous program is **different** from an asynchronous one

What about a standard?

IT’S COMPLICATED

HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC.)

SITUATION:
THERE ARE
14 COMPETING
STANDARDS.

14?! RIDICULOUS!
WE NEED TO DEVELOP
ONE UNIVERSAL STANDARD
THAT COVERS EVERYONE'S
USE CASES.

YEAH!

SOON:

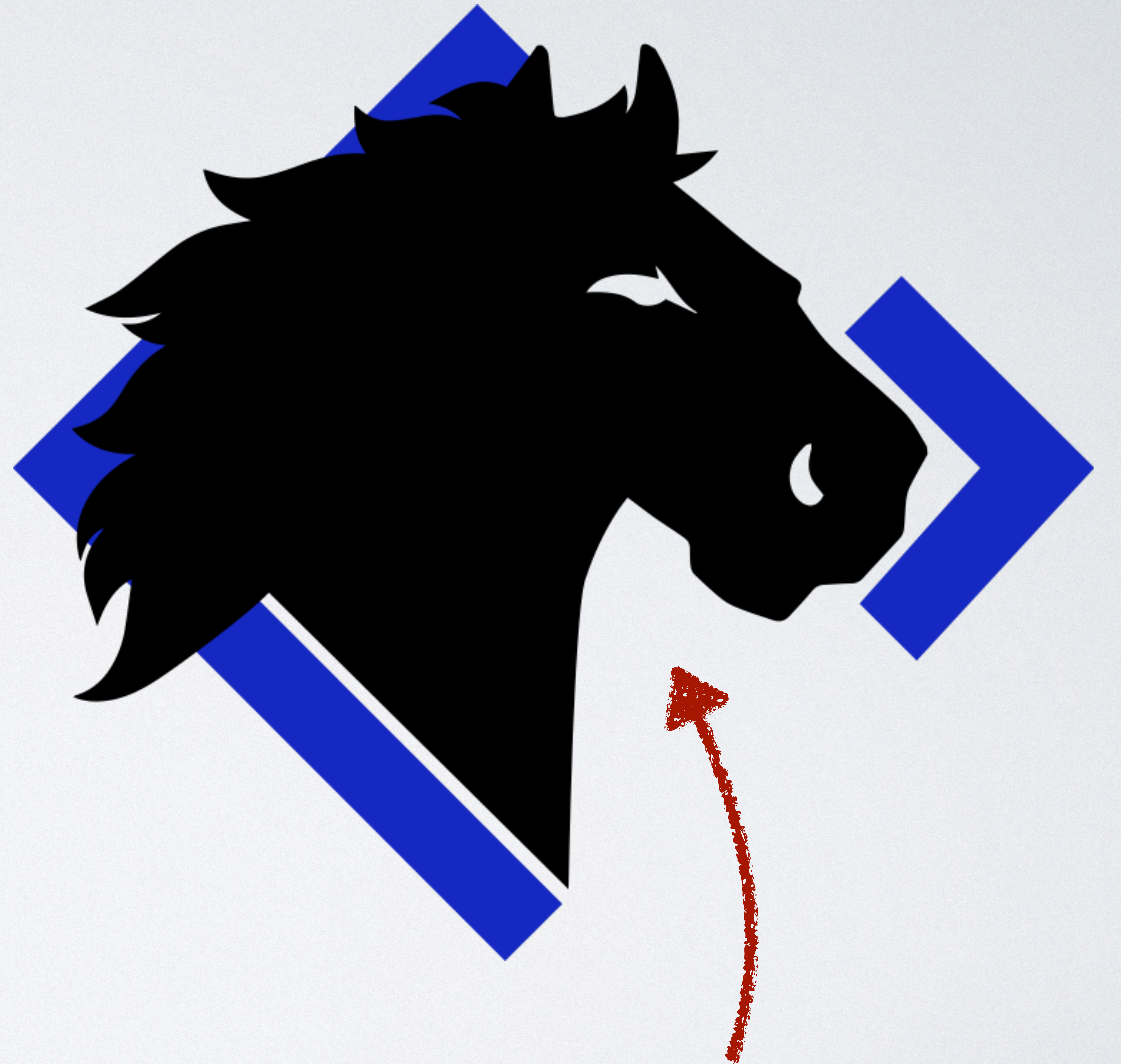
SITUATION:
THERE ARE
15 COMPETING
STANDARDS.

https://xkcd.com/927/

Tornado

https://github.com/M6Web/Tornado

```
composer require m6web/tornado
```



Generators Inside

Interfaces

Weak coupling

Must we run the HTTP Server in PHP?

ReactPhp Example

```php
$loop = React\EventLoop\Factory::create();

$server = new React\Http\Server(function (ServerRequestInterface $request) {
    return new React\Http\Response(
        200,
        array('Content-Type' => 'text/plain'),
        "Hello World!\n"
    );
});

$socket = new React\Socket\Server(8080, $loop);
$server->listen($socket);

echo "Server running at http://127.0.0.1:8080\n";

$loop->run();
```

Long Running Process

Asynchronous programing does **not** require to run a Php **Http Server**

« Local » Event Loop

```
function myController(RequestInterface $request)
{
    // Choose your adapter.
    $eventLoop = new Adapter\Amp\EventLoop();
    // $eventLoop = new Adapter\ReactPhp\EventLoop(
    //     new React\EventLoop\StreamSelectLoop()
    // );
    // $eventLoop = new Adapter\Tornado\EventLoop();
    // $eventLoop = new Adapter\Tornado\SynchronousEventLoop();

    $response = $eventLoop->wait(
        myAsynchronousFunction($request)
    );
}
```

Technical Decisions

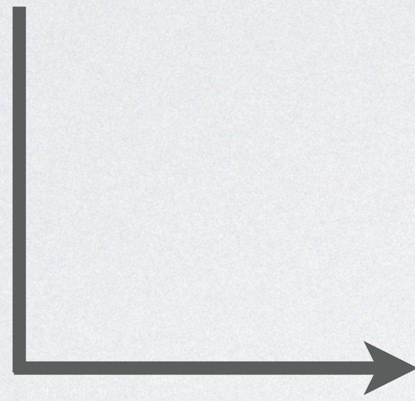
- ✓ Asynchronous programing
- ✓ Generators
- ✓ Tornado interfaces
- ✓ Local event loop

Chapter 2

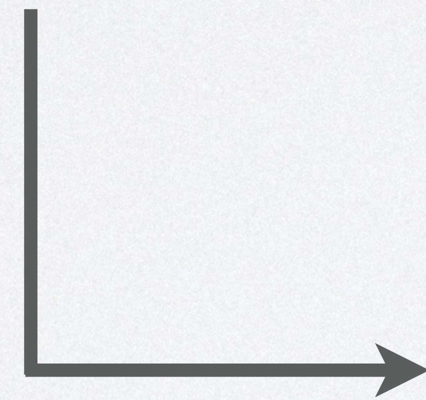
Let’s go!

Where do I start?

```
function foo1(): void {
    //…
    foo2();
    //…
}
```

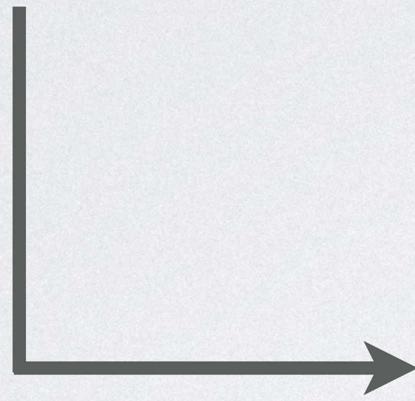


```
function foo2(): void {
    //…
    $bar = foo3();
    //…
}
```

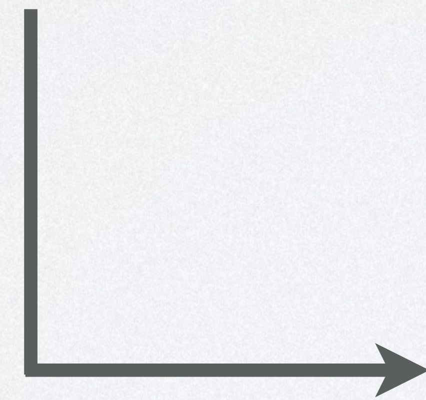


```
function foo3(): int {
    //…
    $client->send(…);
    //…
}
```

```php
function foo1(): void {
    //…
    foo2();
    //…
}
```

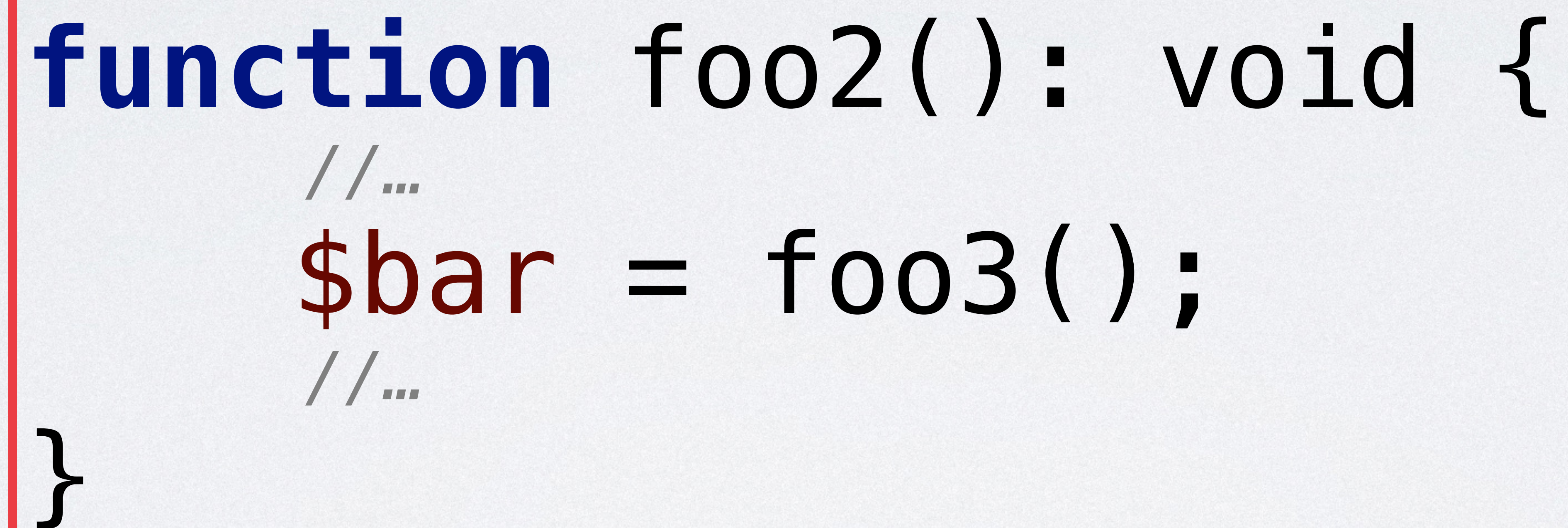


```php
function foo2(): void {
    //…
    $bar = foo3();
    //…
}
```



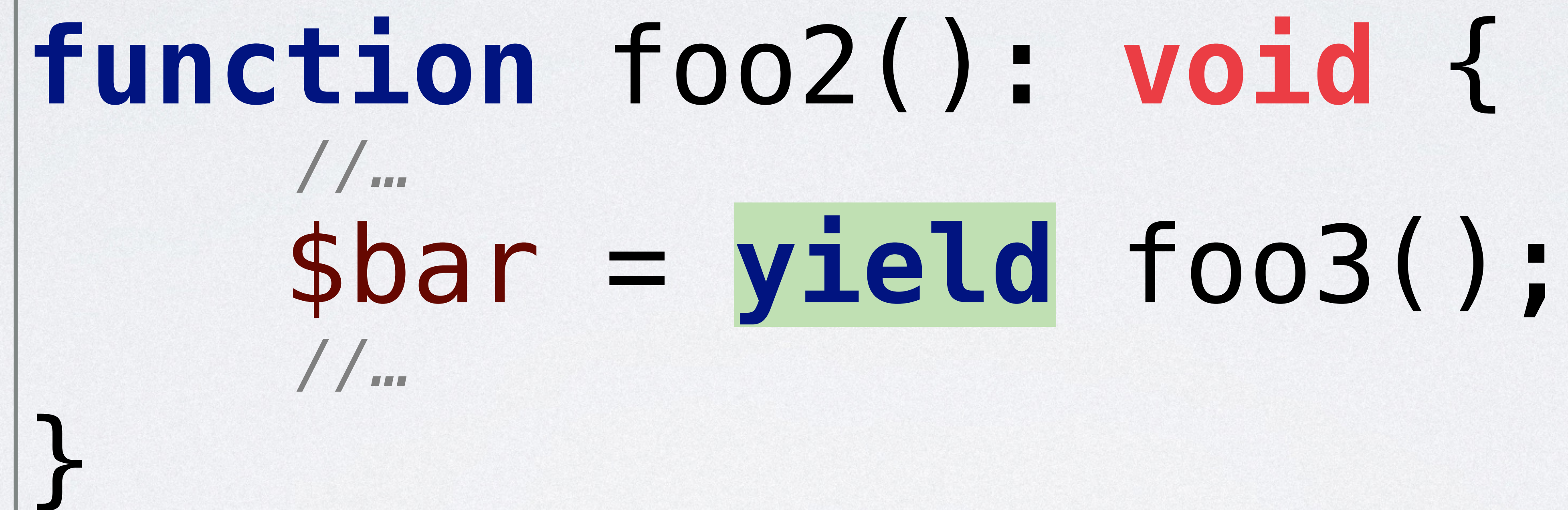
```php
function foo3(): int {
    //…
    yield $client->send(…);
    //…
}
```

```
function foo1(): void {
    //…
    foo2();
    //…
}
```



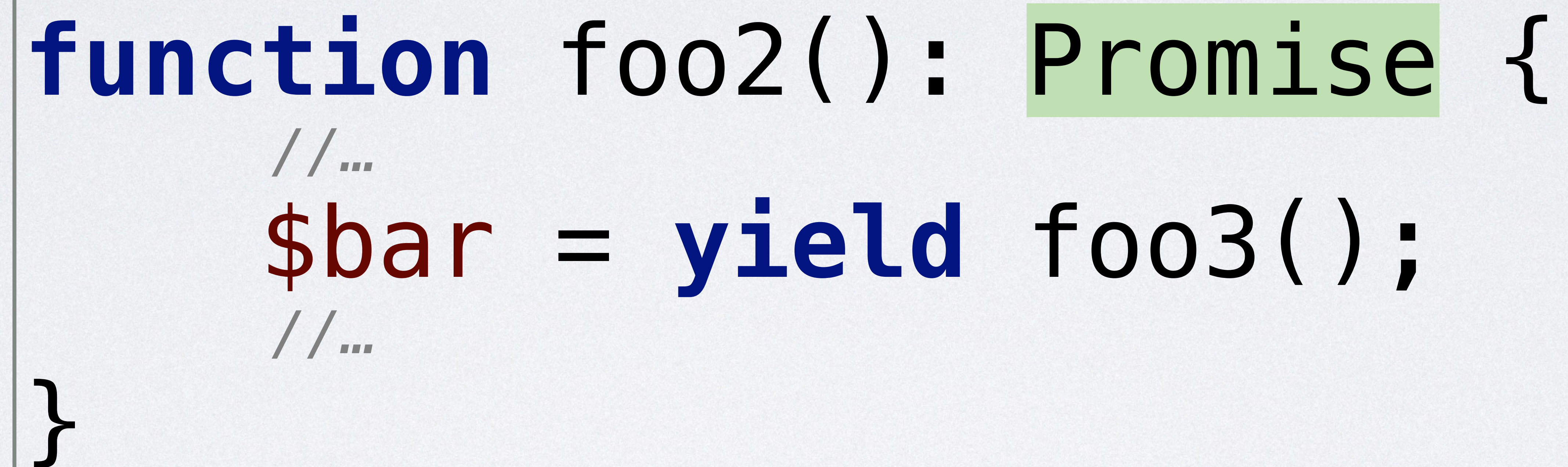

```
function foo3(): Promise {
    //…
    yield $client->send(…);
    //…
}
```

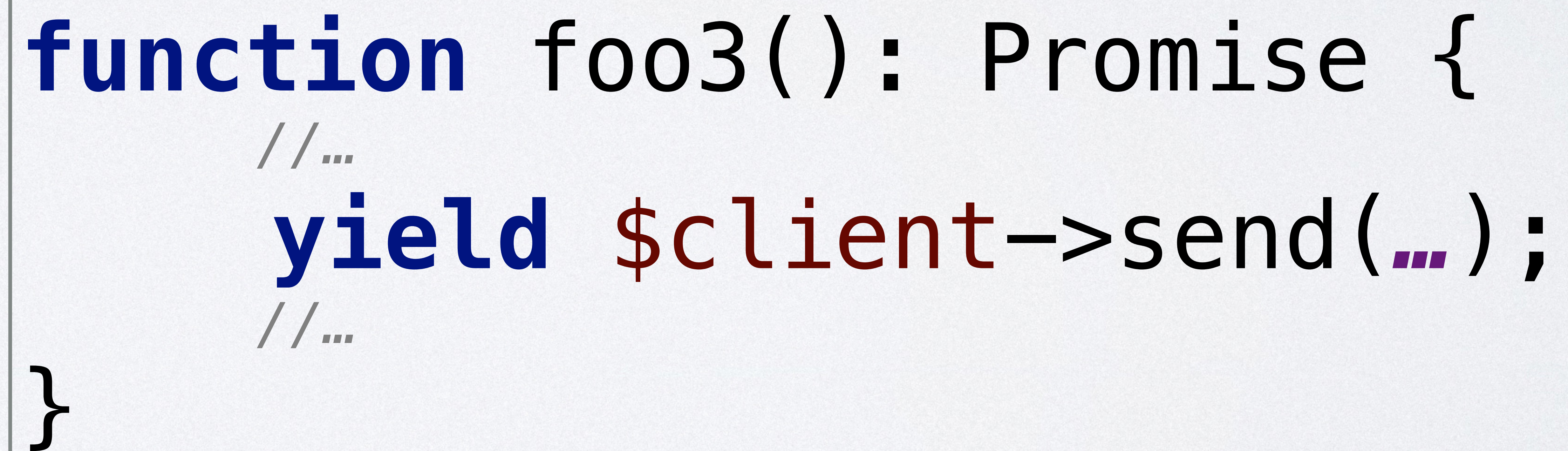
```
function foo1(): void {
    //…
    foo2();
    //…
}
```



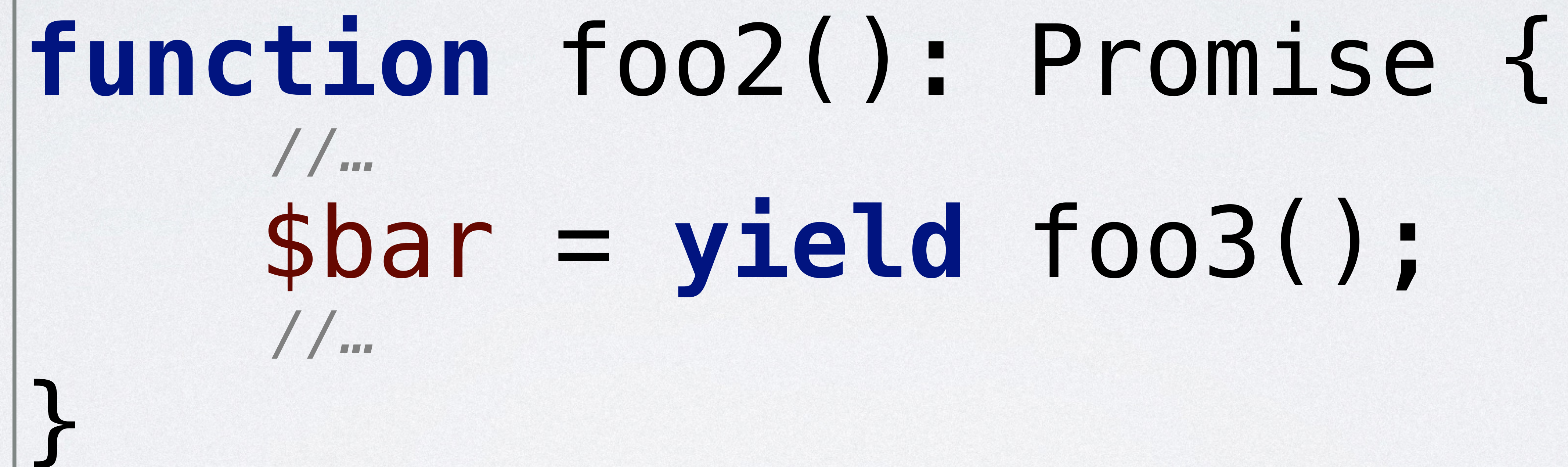

```
function foo3(): Promise {
    //…
    yield $client->send(…);
    //…
}
```

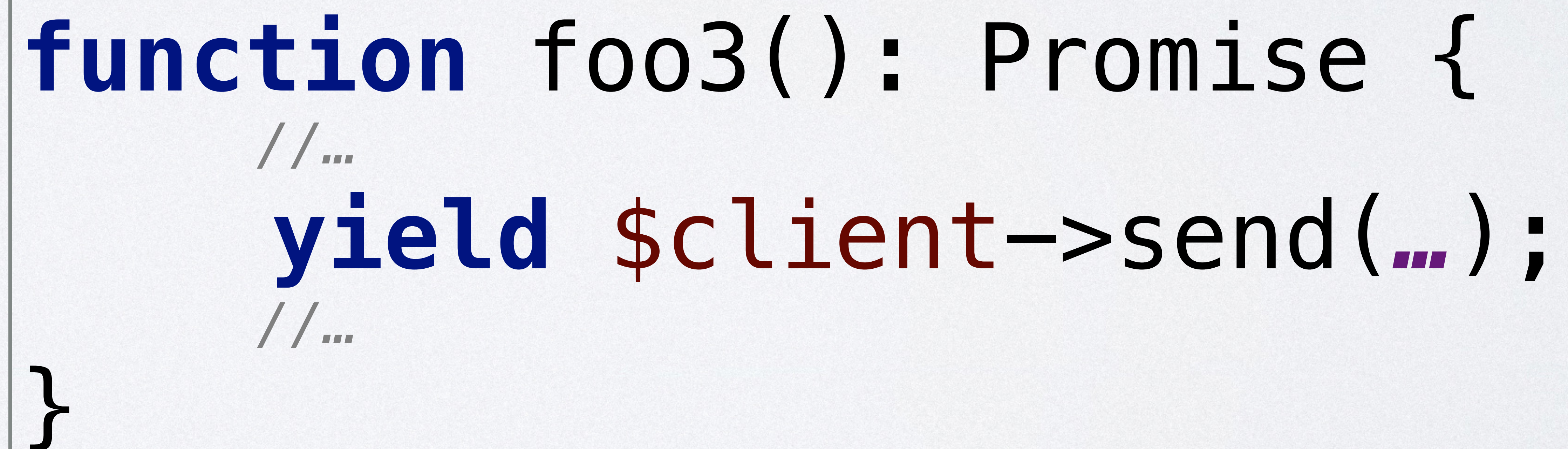
```
function foo1(): void {
    //…
    foo2();
    //…
}
```





```
function foo1(): void {
    //…
    $loop->wait(foo2());
    //…
}
```





```php
function foo1(): void {
    //…
    foo2();
    //…
}

function foo2(): void {
    //…
    $bar = foo3();
    //…
}

function foo3(): int {
    //…
    $client->send(…);
    //…
}
```

```php
function foo1(): void {
    //…
    $loop->wait(foo2());
    //…
}

function foo2(): Promise {
    //…
    $bar = yield foo3();
    //…
}

function foo3(): Promise {
    //…
    yield $client->send(…);
    //…
}
```

A function calling an **asynchronous** function is also **asynchronous**.

How to enable concurrency?

Asynchronous programming is **useless** without **concurrency**.

```php
public function foo()
{
    $user = yield getUserDetails();
    $content = yield getContentDetails();

    return createPage($user, $content);
}
```

getUserDetails()

getContentDetails()

createPage()

```php
public function foo(): \Generator
{
    [$user, $content] = yield $this->eventLoop->promiseAll(
        getUserDetails(),
        getContentDetails()
    );

    return createPage($user, $content);
}
```

```php
public function foo(): \Generator
{
    [$user, $content] = yield $this->eventLoop->promiseAll(
        getUserDetails(),
        getContentDetails()
    );
    $premium = yield getPremiumContent($user);

    return createPage($user, $content, $premium);
}
```

getUserDetails()

getContentDetails()

getPremiumContent() createPage()

Consecutive `yield` instructions means **dependency** between **promises**.

```php
// Intermediate function
private function userAndPremium(): \Generator
{
    $user = yield getUserDetails();
    $premium = yield getPremiumContent($user);

    return [$user, $premium];
}
```

getUserDetails() getPremiumContent()

```php
// Intermediate function
private function userAndPremium(): \Generator
{
    $user = yield getUserDetails();
    $premium = yield getPremiumContent($user);

    return [$user, $premium];
}
```

getUserDetails() getPremiumContent()

```php
public function foo(): \Generator
{
    [[$user, $premium], $content] = yield $this->eventLoop->promiseAll(
        $this->eventLoop->async($this->userAndPremium()),
        getContentDetails()
    );

     return createPage($user, $content, $premium);
}
```

getUserDetails() getPremiumContent()

getContentDetails()

createPage()

Create an **asynchronous** function per **goal**.

```php
public function allPremiumContent(array $users): \Generator
{
    $allPremium = [];
    foreach ($users as $user) {
        $allPremium[] = yield getPremiumContent($user);
    }

    return $allPremium;
}
```

getPremiumContent()

getPremiumContent()

getPremiumContent()

IT'S A TRAP!

```php
public function allPremiumContent(array $users): \Generator
{
    $promises = [];
    foreach ($users as $user) {
        $promises[] = getPremiumContent($user);
    }
    $allPremium = yield $this->eventLoop->promiseAll(...$promises);

    return $allPremium;
}
```

getPremiumContent()

getPremiumContent()

getPremiumContent()

```php
public function allPremiumContent(array $users): \Generator
{
    $promises = [];
    foreach ($users as $user) {
        $promises[] = getPremiumContent($user);
    }
    $allPremium = yield $this->eventLoop->promiseAll(...$promises);

    return $allPremium;
}
```

```php
public function allPremiumContent(array $users): \Generator
{
    $allPremiumPromise =
        $this->eventLoop->promiseForeach($users, function ($user) {
        return yield getPremiumContent($user);
    });

    return yield $allPremiumPromise;
}
```

Identical

Should I return
a Promise
or a Generator?

Generator is only one of the possible ways to produce a **promise**.

Constant promises

```
public function foo(): Promise
{
    //...
    if (isset($this->cache[$key])) {
        return $this->eventLoop->promiseFulfilled(
            $this->cache[$key]
        );
    }

    return $this->cache[$key] = asyncFunction();
}
```

Asynchronous function

```
public function foo(): Promise
{
    return $this->eventLoop->async((function () {
        $success = yield something();
        if ($success) {
            return 'My Value';
        }

        throw new \Exception();
    })());
}
```

Return a **Promise** from **public** asynchronous functions.

Chapter 3

Drawbacks

Some noise...

```
public function myFunction(): Promise
{
    $myAsyncFunction = function(): \Generator  {
        // ...
        yield $promise;
        // ...
    };

    return $this->eventLoop->async($myAsyncFunction);
}
```

Promise hide actual type

```php
function createRandomNumber(): int;

/**
 * @return Promise Will be resolved with an integer
 */
function createRandomNumber(): Promise;

/** @var int $number */
$number = yield createRandomNumber();
```

Stacktrace

```
Call Stack:
1. {main}() /workspace/Tornado/examples/01-async-countdown.php:0
2. M6Web\Tornado\Adapter\Amp\EventLoop->wait() /workspace/Tornado/examples/01-async-countdown.p…
3. Amp\Promise\wait() /workspace/Tornado/src/Adapter/Amp/EventLoop.php:18
4. Amp\Loop::run() /workspace/Tornado/vendor/amphp/amp/lib/functions.php:170
5. Amp\Loop\NativeDriver->run() /workspace/Tornado/vendor/amphp/amp/lib/Loop.php:84
6. Amp\Loop\NativeDriver->tick() /workspace/Tornado/vendor/amphp/amp/lib/Loop/Driver.php:72
7. M6Web\Tornado\Adapter\Amp\EventLoop->M6Web\Tornado\Adapter\Amp\{closure:/workspace/Tornado/…
8. Amp\Deferred->resolve() /workspace/Tornado/src/Adapter/Amp/EventLoop.php:188
9. {anonymous-class:/workspace/Tornado/vendor/amphp/amp/lib/Deferred.php:20-25}->resolve() /…
10. Amp\Internal\ResolutionQueue->__invoke() /workspace/Tornado/vendor/amphp/amp/lib/Internal/…
11. Amp\Coroutine->Amp\{closure:/workspace/Tornado/vendor/amphp/amp/lib/Coroutine.php:79-135}() …
12. Generator->send() /workspace/Tornado/vendor/amphp/amp/lib/Coroutine.php:105
13. M6Web\Tornado\Adapter\Amp\EventLoop->M6Web\Tornado\Adapter\Amp\{closure:/workspace/Tornado…
14. Generator->send() /workspace/Tornado/src/Adapter/Amp/EventLoop.php:67
15. asynchronousCountdown() /workspace/Tornado/src/Adapter/Amp/EventLoop.php:67
```

Blackfire

run_init::public/index.php
Kernel::handle Ker
HttpKernel::handle Htt
HttpKernel::handleRaw ker
LayoutController::__invoke Eve
EventLoop::async EventLoop::wait Eve
Coroutine::__construct wait {clo
Success::onResolve Loop::run Sha
{closure}::li…ne.php/58-99 Driver::run run
Generator::send Driver::tick Driv…tick Driv…ick Dri…ck Dri…ick Re:
{closure}::A…p.php/44-77 {closure}::Amp/…oop.php/187-189 {clo…-189 {cl…189 {cl…89 {cl…89 Val
Generator::send@1 Deferred::resolve Defe…lve Def…lve De…e De…ve Val
{closure}::C…php/110-168 class::resolve::l…eferred.php/18-23 clas…-23 clas…-23 cl…23 cla…23 Jsc
C… LayoutR…resolve ResolutionQueue::__invoke Res…oke Res…ke Re…e Re…ke R…
EventL…sync@1 {closure}::lib/Coroutine.php/58-99 {clo…8-99 {clo…-99 {cl…99 {cl…99
Corouti…truct@1 Generator::send Gen…end Gen…nd Ge…d Ge…nd
Succes…lve@1 {closure}::Amp/E…tLoop.php/44-77 {clo…4-77 {clo…-77 {cl…77 {cl…77
{closu…8-99@1 Generator::send@1 Gen…@1 Ge…@1 Ge…1 Ge…1
Gener…end@2 HttpClient::guzzleEventLoop Http…oop Htt…oop Ht…op Htt…op
{closu…4-77@1 CurlMultiClientWrapper::tick Curl…tick Curl…ick Cu…ck Cu…ck
Gener…end@3 CurlMultiClientWrapper::tick Curl…tick Curl…ick Cu…ck Cu…ck
Layout…Layout CurlMultiHandler::tick Curl…tick Curl…ick Cu…ck Cu…ck
EventL…ync@2 TaskQueue::run curl…xec Tas…run Ta…un Ta…un
Corout…ruct@2 {closur…154-158 {clo…158 {cl…58 {… {c…8
Generat…urrent Promise…andler Pro…ler Pr…er P… Pr…r
{closu…4-77@2 {closur…p/39-42 {clo…-42 {cl…42 {… {c…2

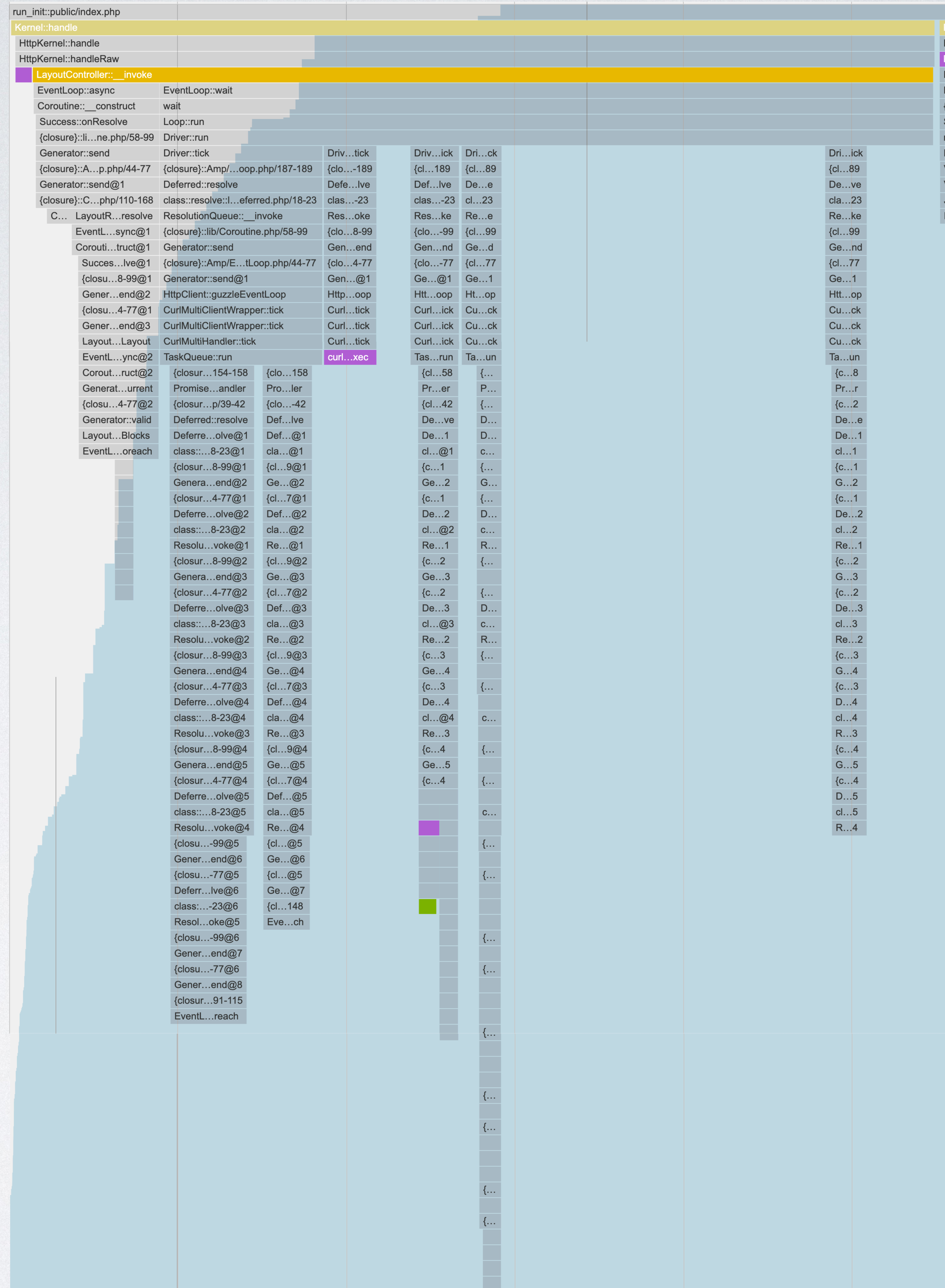
{closur…8-99@4
Genera…end@5
{closur…4-77@4
Deferre…olve@5
class::…8-23@5
Resolu…voke@4
{closu…-99@5
Gener…end@6
{closu…-77@5
Deferr…lve@6
class:…-23@6
Resol…oke@5
{closu…-99@6
Gener…end@7
{closu…-77@6
Gener…end@8
{closur…91-115
EventL…reach

{cl…9@4
Ge…@5
{cl…7@4
Def…@5
cla…@5
Re…@4
{cl…@5
Ge…@6
{cl…@5
Ge…@7
{cl…148
Eve…ch

{c…4
Ge…5
{c…4

{…
{…
c…
{…
{…
{…
{…
{…
{…
{…
{…
{…

{c…4
G…5
{c…4
D…5
cl…5
R…4



Work In progress…

Homemade tricks

Foo1

Foo2

Foo3

Foo4

Foo5

An event loop is always running!

⚠️ Takes care of your CPU ⚠️

It's **really** (too) easy to send **too many** requests.

True story…

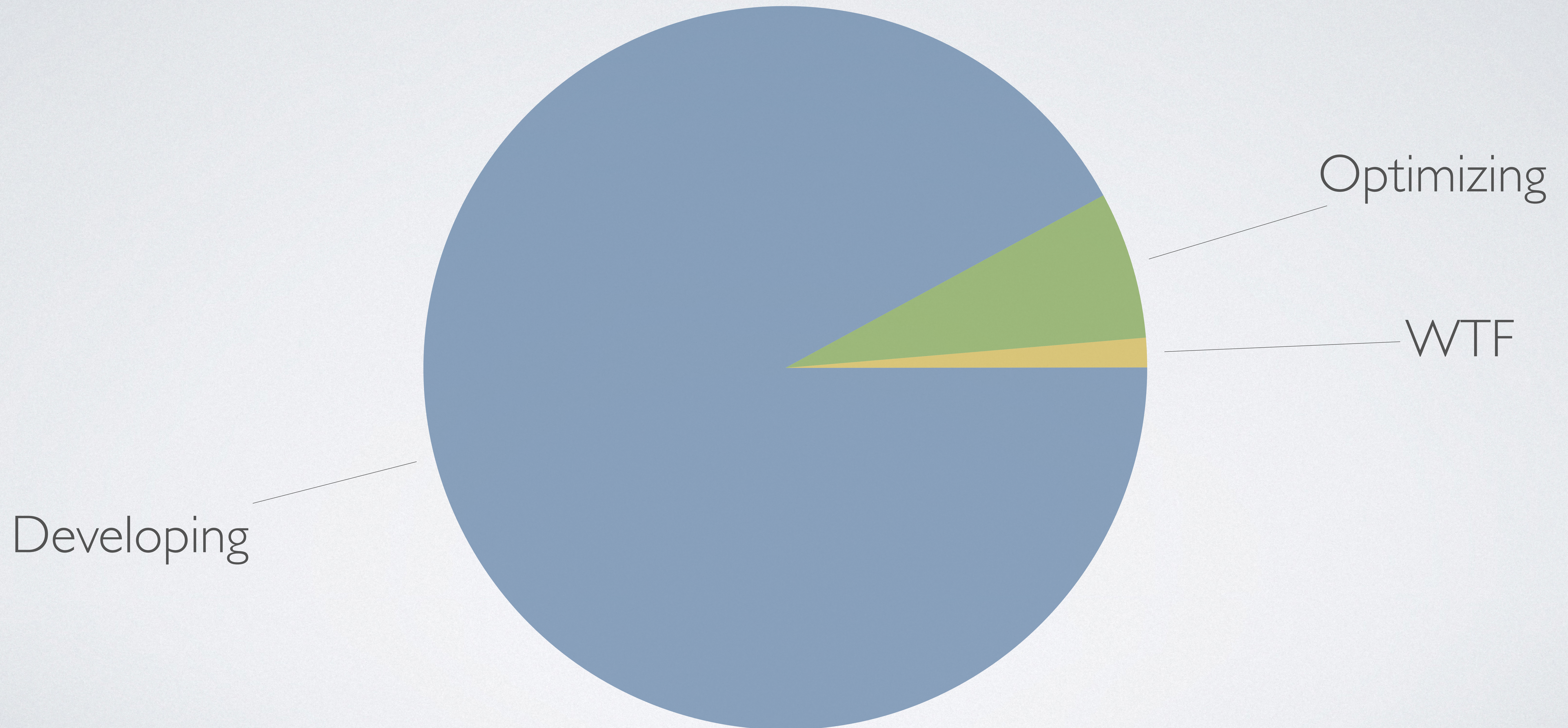
Asynchronously ever After...

Conclusion

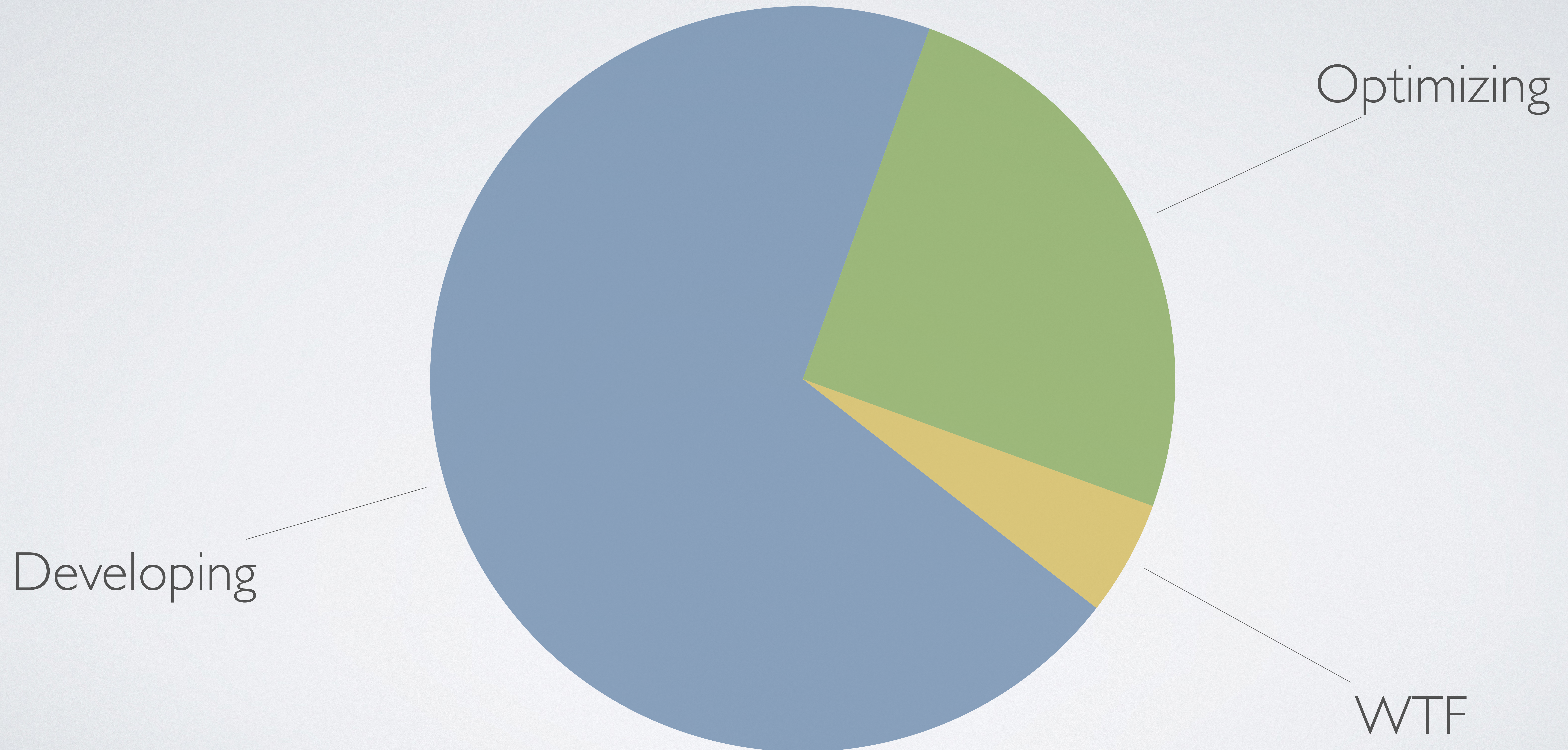
Training?

Using asynchronous programing is **not** a **problem**.

Synchronous



Asynchronous



Metrics?

Depends **a lot** of
your use case…

https://github.com/b-viguier/tornado-workshop

Text

Sentence Sentence Sentence

Word Word Word Word Word Word Word Word Word

Letter Letter Letter Letter Letter Letter Letter Letter Letter Letter Letter Letter Letter Letter Letter

https://github.com/b-viguier/tornado-workshop

Synchronous program: 20 **minutes**
Asynchronous program: 20 **seconds**

Technical Decisions

- ✓ Asynchronous programing
- ✓ Generators
- ✓ Tornado interfaces
- ✓ Local event loop

Tornado

https://github.com/M6Web/Tornado

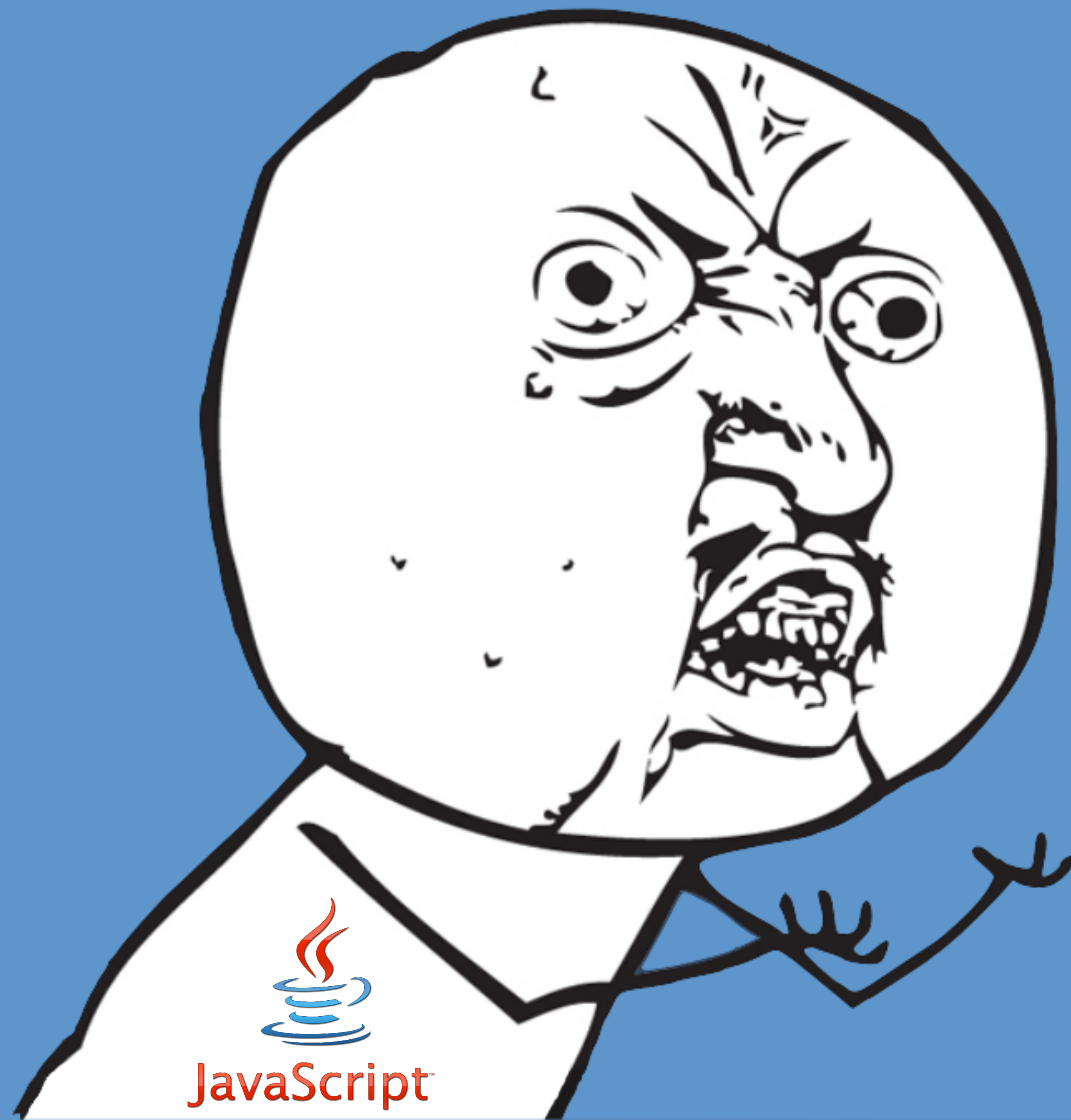
```
composer require m6web/tornado
```

One year of asynchronous Php in production

Benoit Viguier

Afup Lyon
10/10/2019

« J » Syndrom



PHP DEV

Y U NO USE JAVASCRIPT

PHP DEV

Y U NO USE PHP