# One year of asynchronous Php in production

BEDROCK

Benoit Viguier

@b_viguier

ConFoo.CA

DEVELOPER CONFERENCE

# One year of asynchronous Php in production


Benoit Viguier
@b_viguier

*Previously…*

# Episode 0

# Why and how to use generators for asynchronous programming

It was yesterday…

ConFoo.ca

DEVELOPER CONFERENCE

# ASYNCHRONOUS PHP

# IS POSSIBLE


PARENTAL
ADVISORY
EXPLICIT PHP

# *Once Upon a Time...*

## Context


NETHERLANDS
IRELAND
Dublin
Amsterdam
Berlin
Warsaw
BELAR
London
GERMANY
POLAND
Celtic Sea
English Channel
Brussels
BELGIUM
LUX.
Prague
CZECH REP.
Paris
Luxembourg
SLOVAKIA
Bratislava
LIECH.
Vienna
Budapest
AUSTRIA
HUNGARY
FRANCE
Bern
SWITZERLAND
SLOVENIA
Chisin
ROMANIA
Ljubljana
Zagreb
CROATIA
Bay of Biscay
ITALY
Bucharest
BOS. & HER.
Belgrade
Sarajevo
SERBIA
SAN MARINO
BULGARIA
Pristina
MONT.
Sofia
KOS.
MONACO
Rome
Podgorica
Skopje
MACEDONIA
Tirana
ALB.
ANDORRA
VATICAN CITY
Madrid
PORTUGAL
SPAIN
Lisbon
Mediterranean Sea
GREECE
Athens

# Lyon, France


BR
BEDROCK


BR
BEDROCK
EXIT
BACK/ RETURN
MEDIA
GUIDE
TIMER
TEXT
Panasonic
TV


GROUPE
RTL
GROUP

## PRODUCTION ET ACQUISITIONS DE CONTENUS

### TÉLÉVISION

- STUDIO 89 PRODUCTIONS
- C-PRODUCTIONS

### DIGITAL

- GOLDEN NETWORK

### CINÉMA

- SND GROUPE M6
- STUDIO
- FILMS
- SNC GROUPE M6

## MÉDIAS

### TÉLÉVISION EN CLAIR

- 6
- W9
- 6ter
- gulli

### TÉLÉVISION PAYANTE

- PARIS PREMIERE
- téva
- serieclub
- Music
- Tiji
- canal J
- MCM TOP
- MCM
- RFM TV

### TÉLÉVISION INTERNATIONALE

- gulli GIRL
- gulli بالعربي
- gulli AFRICA
- Tiji RUSSIA

### DIGITAL

- gulli Replay
- 6play
- gulli MAX

### RADIO

- RTL
- RTL2 LE SON POP-ROCK
- fun radio LE SON DANCEFLOOR

## DIVERSIFICATIONS

- INTERACTIONS
- EDITIONS
- EVENEMENTS
- MUSIC LABEL
- gulli Parc
- Ventadis — Vente à distance - Groupe M6
- M6 boutique
- BEST of TV
- Leonis
- JOJIKA
- CREATIONS
- DIGITAL SERVICES
- Radins.com
- iGraal
- odiso
- mindbaz — your emailing service
- MINDTARGET
- TURBO
- M6météo
- passeport santé
- FOURCHETTE & BIKINI
- cuisineaz
- CROQ'KILOS
- BOTANIQUE
- déco.fr
- Panorabanques.com

## RÉGIE

- CTZAR {za:(r)}
- PUBLICITÉ
- UNLIMITED

## PRODUCTION ET ACQUISITIONS DE CONTENUS

### TÉLÉVISION

STUDIO 89 PRODUCTIONS

C-PRODUCTIONS

### DIGITAL

GOLDEN NETWORK

### CINÉMA

SND GROUPE M6

M6 STUDIO

M6 FILMS

SNC GROUPE M6

## MÉDIAS

### TÉLÉVISION EN CLAIR

M6 · W9 · 6ter · gulli

### TÉLÉVISION PAYANTE

Paris Première · téva · serieclub · Music · Tiji · Canal J · MCM Top · MCM · RFM TV

### TÉLÉVISION INTERNATIONALE

gulli girl · gulli بالعربي · gulli Africa · Tiji Russia

### DIGITAL

gulli Replay · 6play · gulli MAX

### RADIO

RTL · RTL2 LE SON POP-ROCK · fun radio LE SON DANCEFLOOR

## DIVERSIFICATIONS

M6 INTERACTIONS · M6 EDITIONS · M6 EVENEMENTS · M6 MUSIC LABEL · gulli Parc

Ventadis Vente à distance - Groupe M6 · M6 Boutique · Best of TV · Leonis Corner · Jojika · M6 CREATIONS

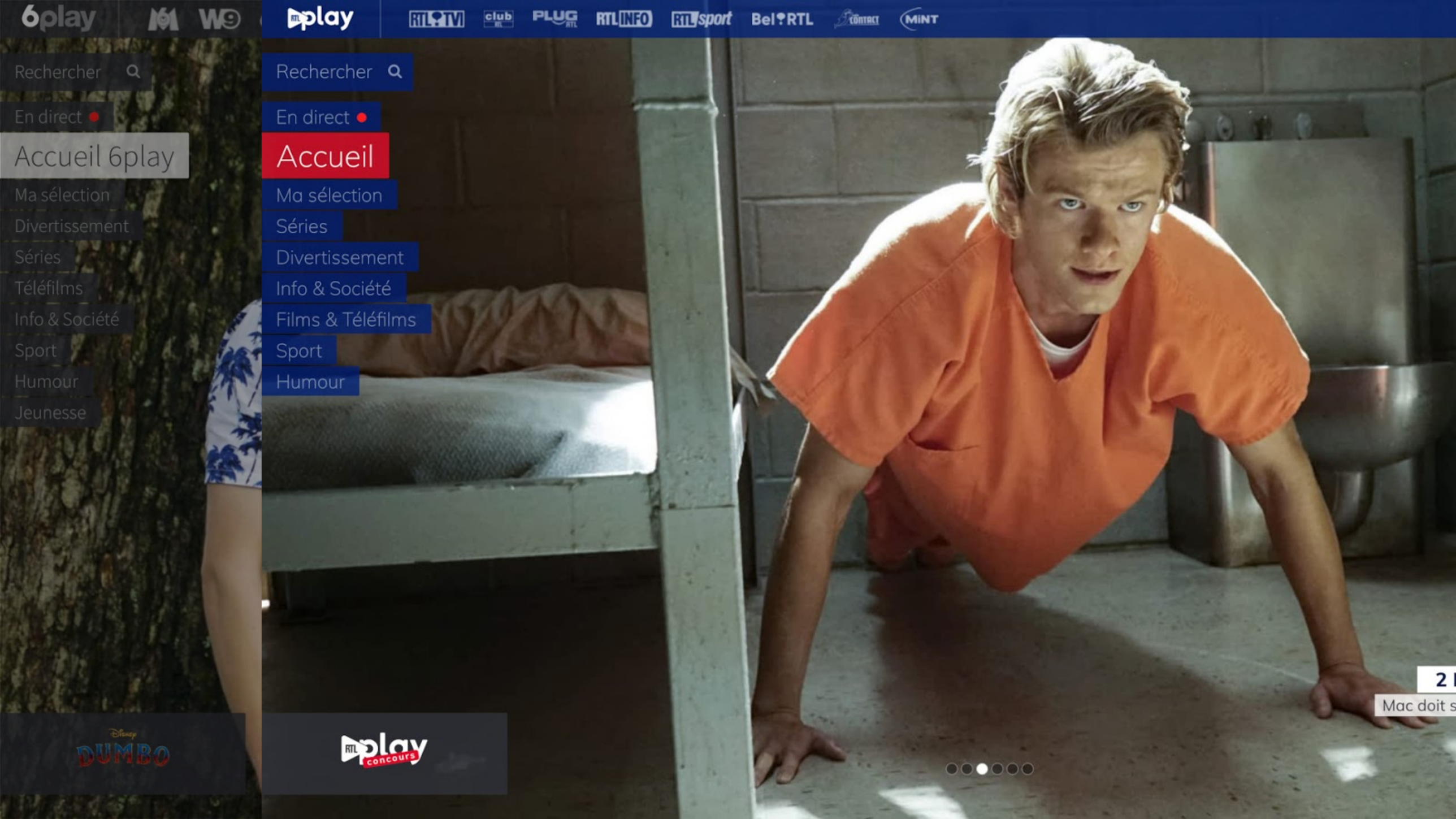
M6 DIGITAL SERVICES

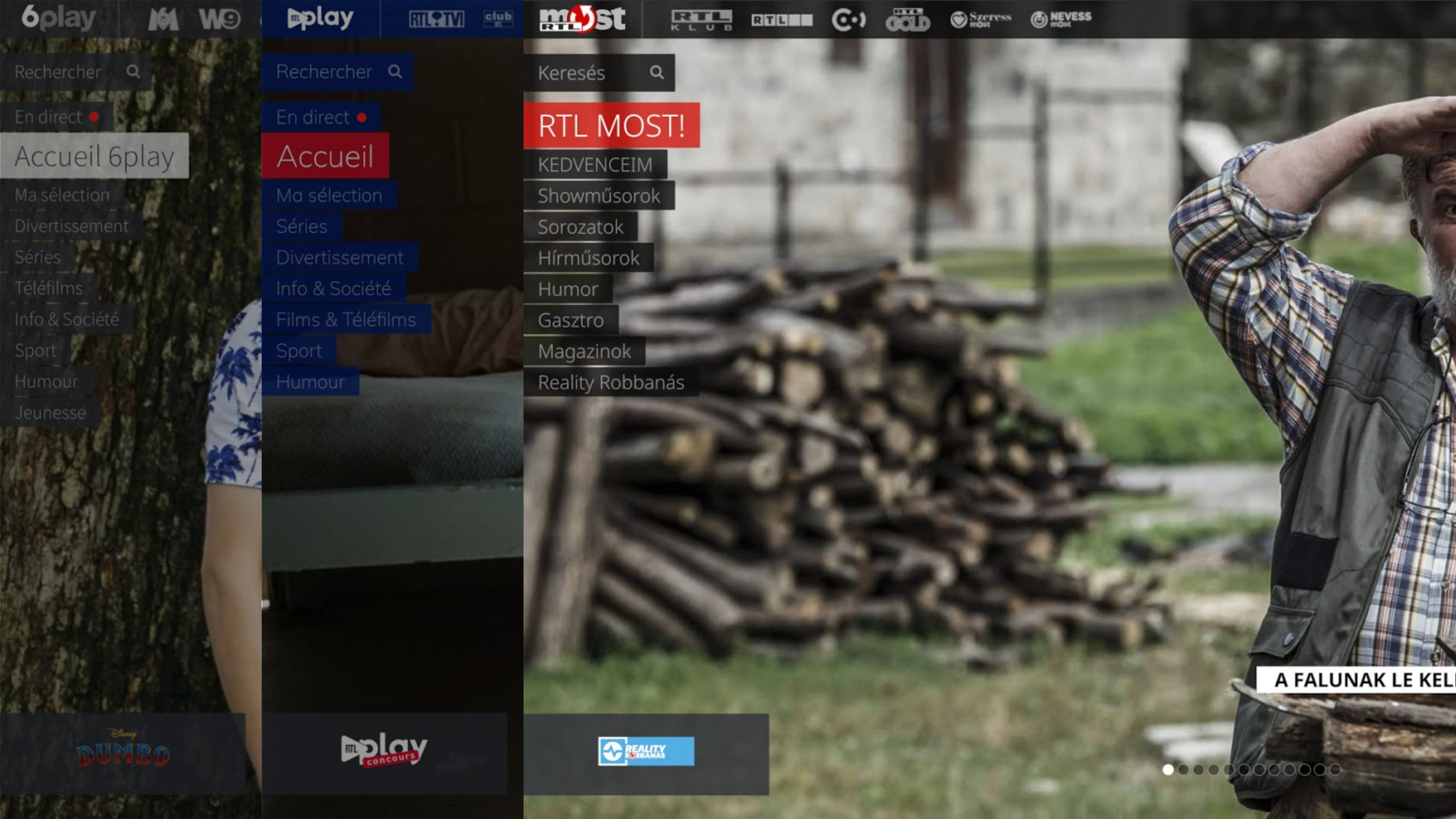
Radins.com · iGraal · odiso · mindbaz your emailing service · MINDTARGET · TURBO · M6météo · passeport santé · Fourchette & Bikini · cuisineaz · Croq'Kilos · Botanique · déco.fr · Panorabanques.com

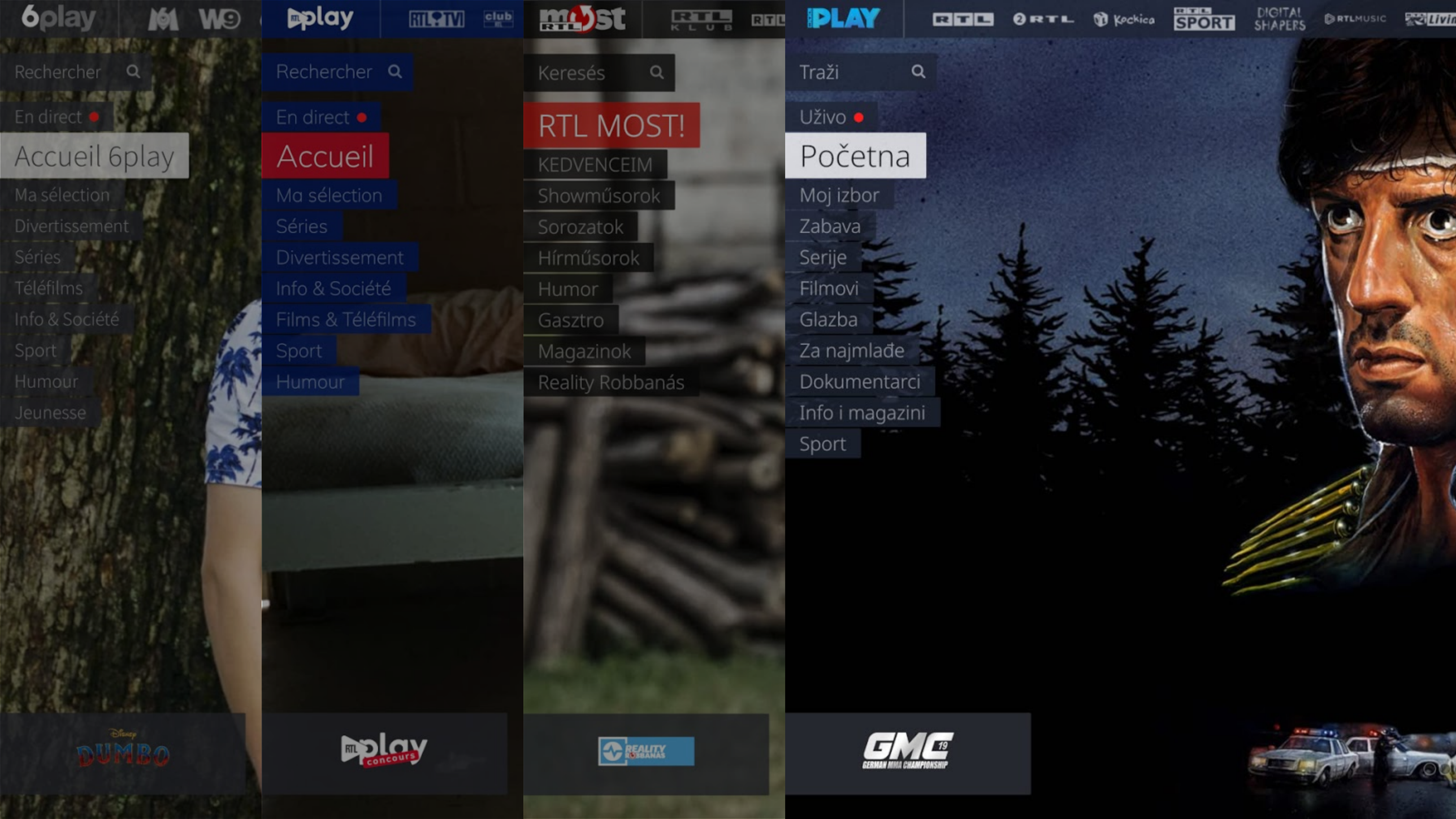
## RÉGIE

CTZAR {za:(r)} · M6 PUBLICITÉ · UNLIMITED

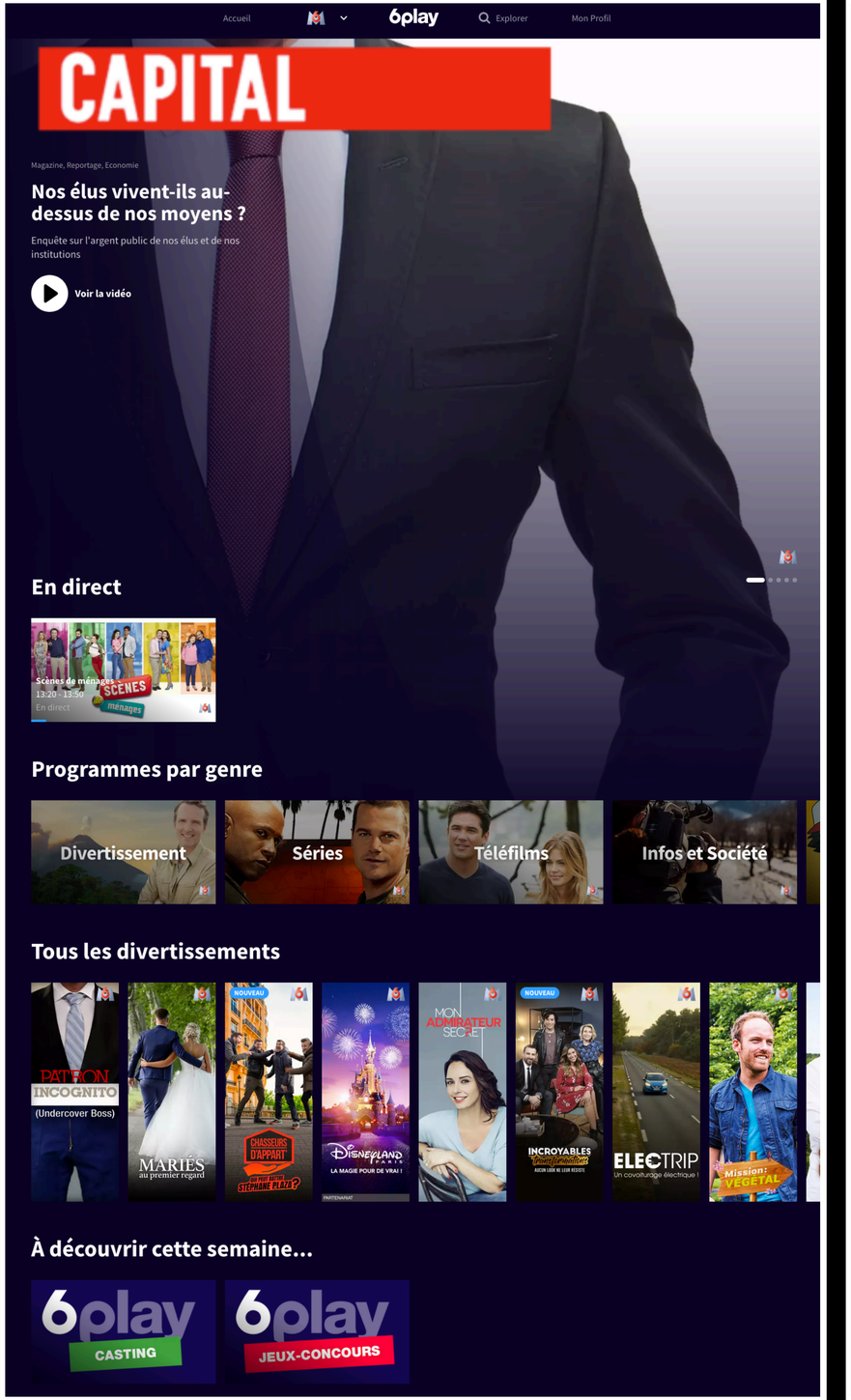

6play
Se connecter
Rechercher
En direct
Accueil 6play
Ma sélection
Divertissement
Séries
Téléfilms
Info & Société
Sport
Humour
Jeunesse
MACGYVER
SAISON 3 : ÉPISODE 3
Retour à la fac
VOIR LA VIDÉO
DUMBO


6play
Rechercher
En direct
Accueil 6play
Ma sélection
Divertissement
Séries
Téléfilms
Info & Société
Sport
Humour
Jeunesse
DUMBO
RTL play
RTL TVI
club RTL
PLUG RTL
RTL INFO
RTL sport
Bel RTL
CONTACT
MINT
Rechercher
En direct
Accueil
Ma sélection
Séries
Divertissement
Info & Société
Films & Téléfilms
Sport
Humour
RTL play concours
2
Mac doit s


6play
Rechercher
En direct
Accueil 6play
Ma sélection
Divertissement
Séries
Téléfilms
Info & Société
Sport
Humour
Jeunesse
Disney DUMBO
play
RTL TV
club
Rechercher
En direct
Accueil
Ma sélection
Séries
Divertissement
Info & Société
Films & Téléfilms
Sport
Humour
play concours
most RTL
RTL KLUB
RTL
Szeress most
NEVESS most
Keresés
RTL MOST!
KEDVENCEIM
Showműsorok
Sorozatok
Hírműsorok
Humor
Gasztro
Magazinok
Reality Robbanás
REALITY ROBBANÁS
A FALUNAK LE KEL


6play
Rechercher
En direct
Accueil 6play
Ma sélection
Divertissement
Séries
Téléfilms
Info & Société
Sport
Humour
Jeunesse
DUMBO
play
RTL TV
club
Rechercher
En direct
Accueil
Ma sélection
Séries
Divertissement
Info & Société
Films & Téléfilms
Sport
Humour
play concours
most
RTL KLUB
Keresés
RTL MOST!
KEDVENCEIM
Showműsorok
Sorozatok
Hírműsorok
Humor
Gasztro
Magazinok
Reality Robbanás
REALITY BANAS
PLAY
RTL
2 RTL
Kockica
RTL SPORT
DIGITAL SHAPERS
RTL MUSIC
Traži
Uživo
Početna
Moj izbor
Zabava
Serije
Filmovi
Glazba
Za najmlađe
Dokumentarci
Info i magazini
Sport
GMC
GERMAN MMA CHAMPIONSHIP


Accueil
Chaînes
6play
Explorer
Mon Profil
En direct sur nos chaînes
Météo
13:15 - 13:20
En direct
N.C.I.S
12:40 - 13:30
En direct
La petite maison dans la prairie
12:50 - 13:45
En direct
Le 12-16 : Le Son DANCEFLOOR
12:00 - 16:00
En direct
Vos lectures en cours
Mon admirateur secret
Emission du 06 janvier
Il y a une semaine • 49:53
Les Reines du Shopping
Spéciale duel : Originale en noir / Jour…
Il y a 3 semaines • 50:30
Capital
Vêtements, aliments, produits high-t…
1:43:23
Equipe de France
Didier Deschamps : "Un match un peu …
2:58
Programmes par genre
Divertissement
Séries
Téléfilms
Infos et Société
Programmes recommandés pour vous
Docteur Quinn, femme médecin
6 épisodes
Bruno dans la radio LA RADIO
433 vidéos
RTL2 Pop-Rock Station
184 émissions
N.C.I.S : Los Angeles
3 épisodes
2020, une année riche en divertissements
PATRON INCOGNITO
(Undercover Boss)
Princes et princesses de l'amour
MARIÉS au premier regard
CHASSEURS D'APPART
Disneyland Paris
MON ADMIRATEUR SECRET
MAISON À VENDRE
Mission: VÉGÉTAL
À découvrir cette semaine…
CONCERT
KIDS UNITED
Nouvelle Génération
Disneyland Paris
6play CASTING
6play JEUX-CONCOURS
6play
Informations
Aide et contact
Mentions légales
Confidentialité
Les programmes
Tous les programmes
Grille des programmes
Participation
Jeux concours
Castings
M6
Groupe M6
Nos réseaux sociaux


Accueil
6play
Explorer
Mon Profil
CAPITAL
Magazine, Reportage, Economie
Nos élus vivent-ils au-dessus de nos moyens ?
Enquête sur l'argent public de nos élus et de nos institutions
Voir la vidéo
En direct
Scènes de ménages
13:20 - 13:50
En direct
Programmes par genre
Divertissement
Séries
Téléfilms
Infos et Société
Tous les divertissements
PATRON INCOGNITO
(Undercover Boss)
MARIÉS au premier regard
CHASSEURS D'APPART
Disneyland Paris
MON ADMIRATEUR SECRET
INCROYABLES TRANSFORMATIONS
ELECTRIP
Un covoiturage électrique !
Mission: VÉGÉTAL
À découvrir cette semaine…
6play CASTING
6play JEUX-CONCOURS


Accueil
Chaînes
6play
Explorer
Mon Profil
Windeck
Voir la vidéo
Episode 1
Episodes 1 à 10
Episode 1
46:54
Episode 2
45:50
Episode 3
44:53
Episode 4
45:09
Episodes 11 à 20
Episode 11
44:34
Episode 12
45:17
Episode 13
43:41
Episode 14
45:30
Episodes 21 à 30
Episode 21
44:43
Episode 22
40:41
Episode 23
43:05
Episode 24
45:20
Episodes 31 à 40
Episode 31
41:25
Episode 32
43:13
Episode 33
39:55
Episode 34
43:41
Episodes 41 à 50
Episode 41
45:22
Episode 42
43:51
Episode 43
44:28
Episode 44
42:35
Episodes 51 à 60
Episode 51
45:00
Episode 52
43:52
Episode 53
44:05
Episode 54
44:07

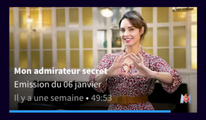
Accueil Chaînes 6play Explorer Mon Profil

## En direct sur nos chaînes

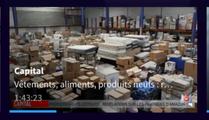




## Vos lectures en cours

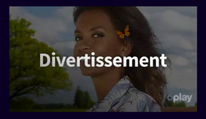








## Programmes par genre





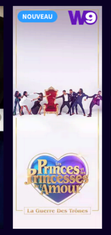


## Programmes recommandés pour vous





## 2020, une année riche en divertissements





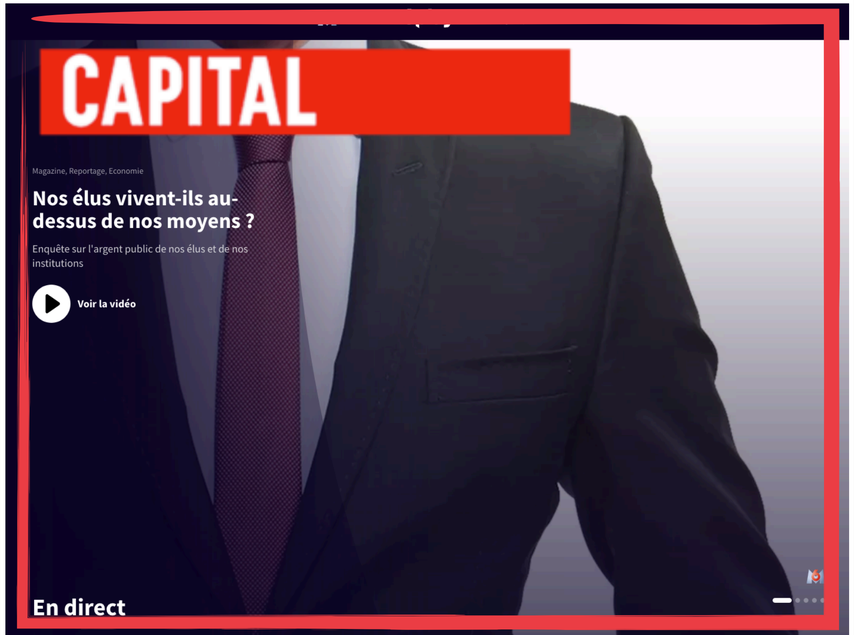



## À découvrir cette semaine…



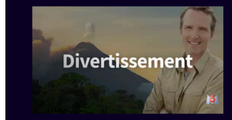

6play

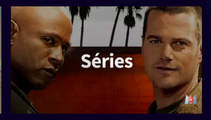
| Informations | Les programmes | Participation | M6 | Nos réseaux sociaux |
|---|---|---|---|---|
| Aide et contact | Tous les programmes | Jeux concours | Groupe M6 | |
| Mentions légales | Grille des programmes | Castings | | |
| Confidentialité | | | | |





## Programmes par genre



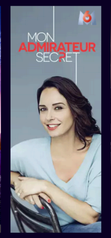





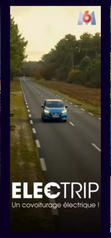

## Tous les divertissements















## À découvrir cette semaine…







Accueil Chaînes 6play Explorer Mon Profil

Windeck




## Episodes 1 à 10













## Episodes 11 à 20













## Episodes 21 à 30













## Episodes 31 à 40













## Episodes 41 à 50













## Episodes 51 à 60

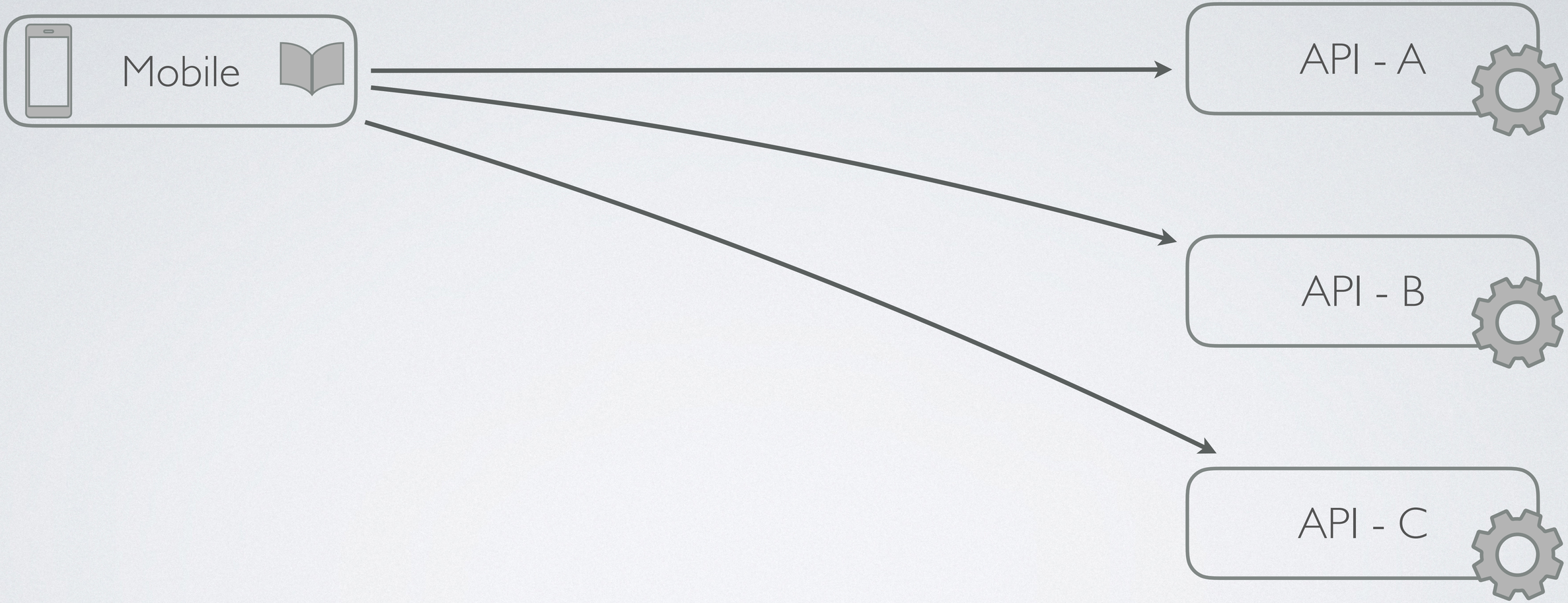
Benoit Viguier
Backend
Lead Developer

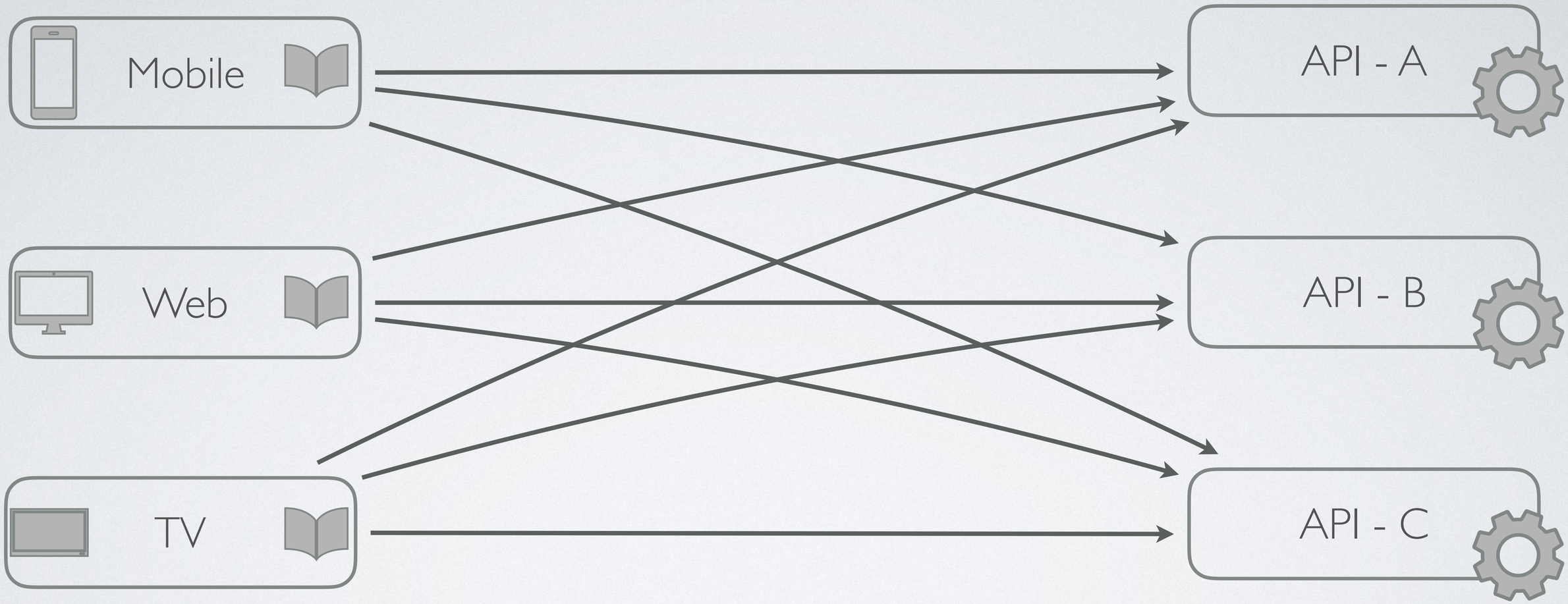
API - A

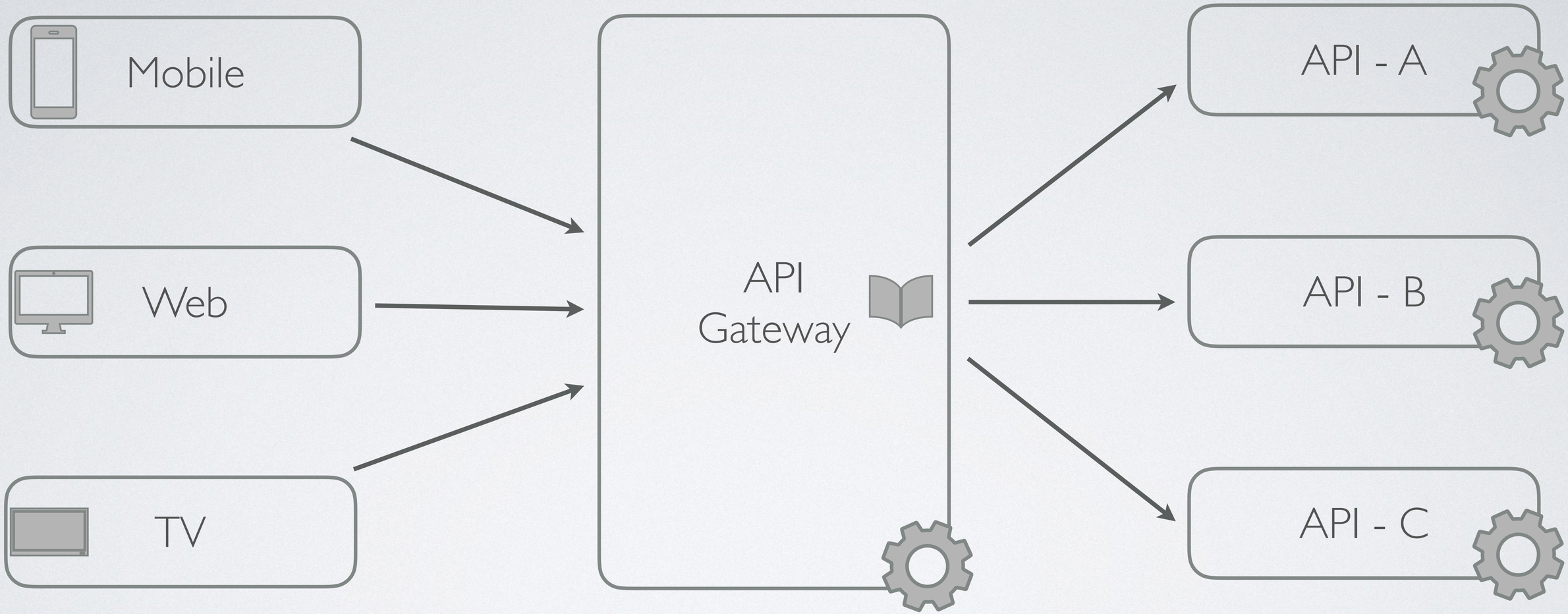
API - B

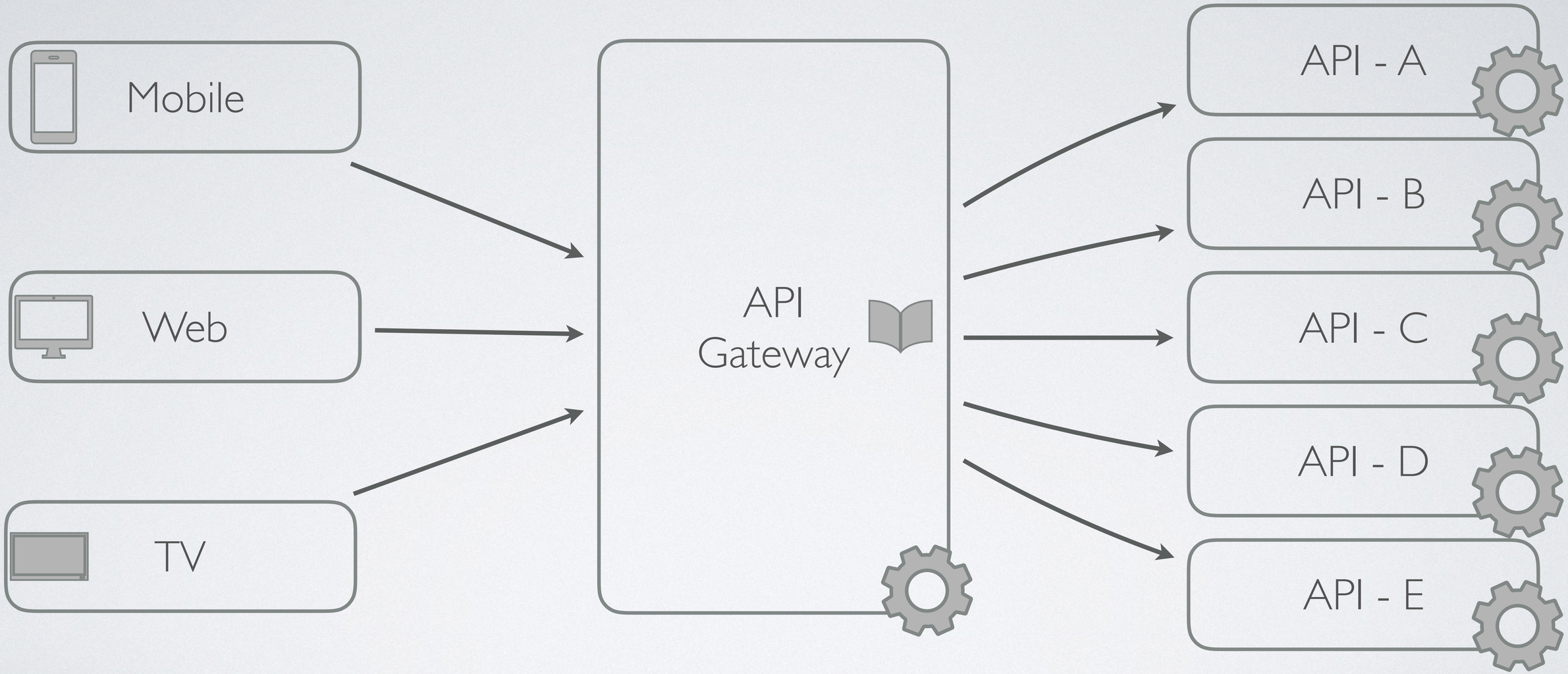
API - C

PHP Inside !


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

```php
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);
    }
}
```

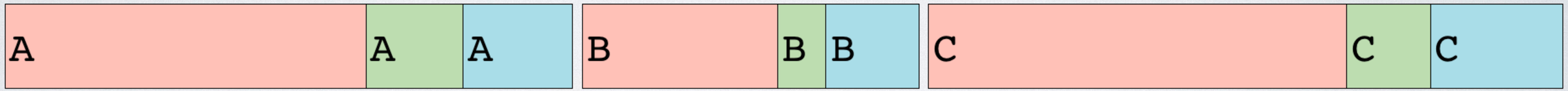
# Synchronous

```php
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);
    }
}
```

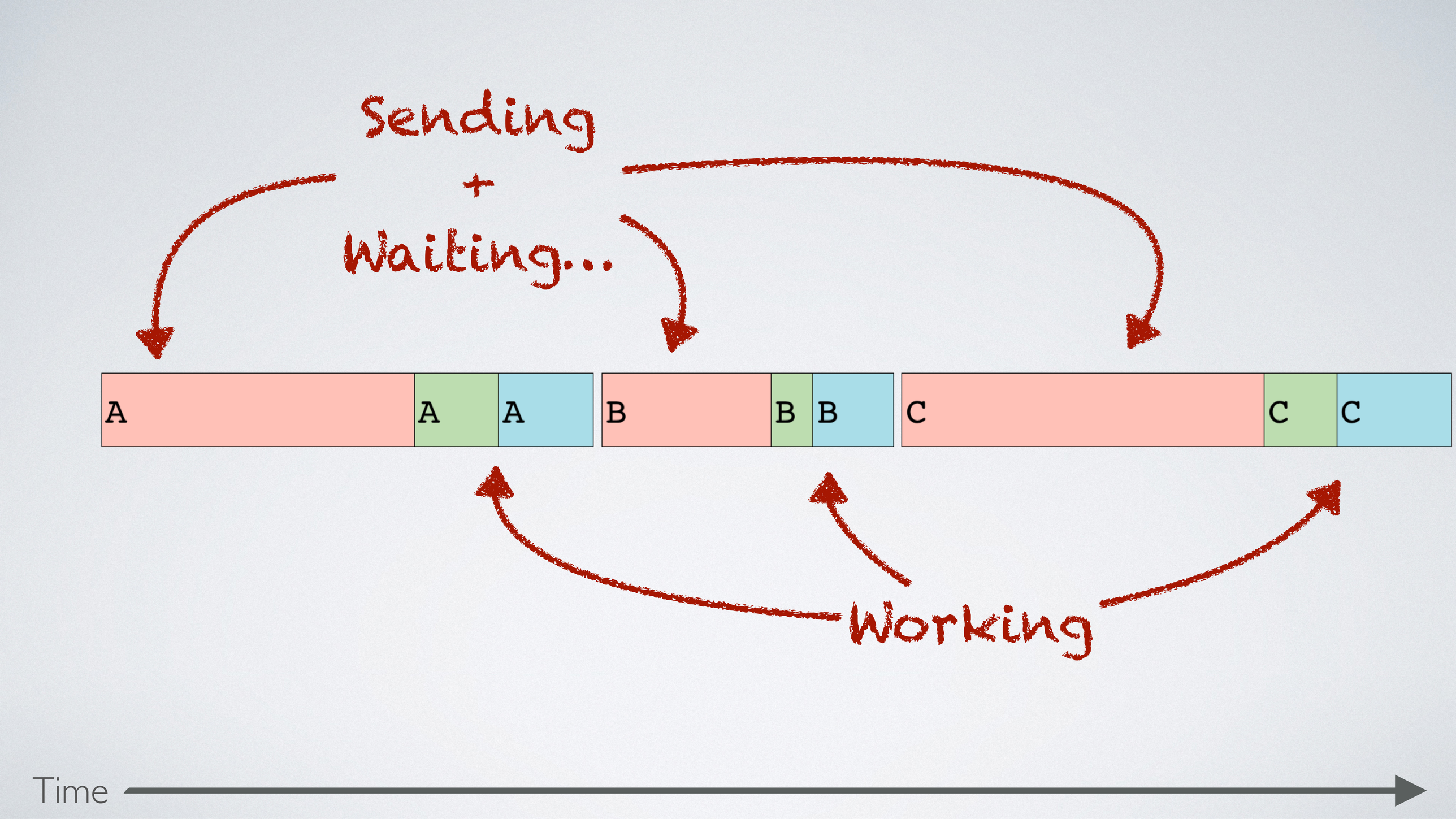
HTTP request

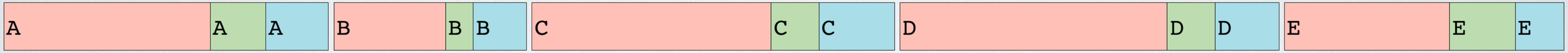
Parsing

Business logic



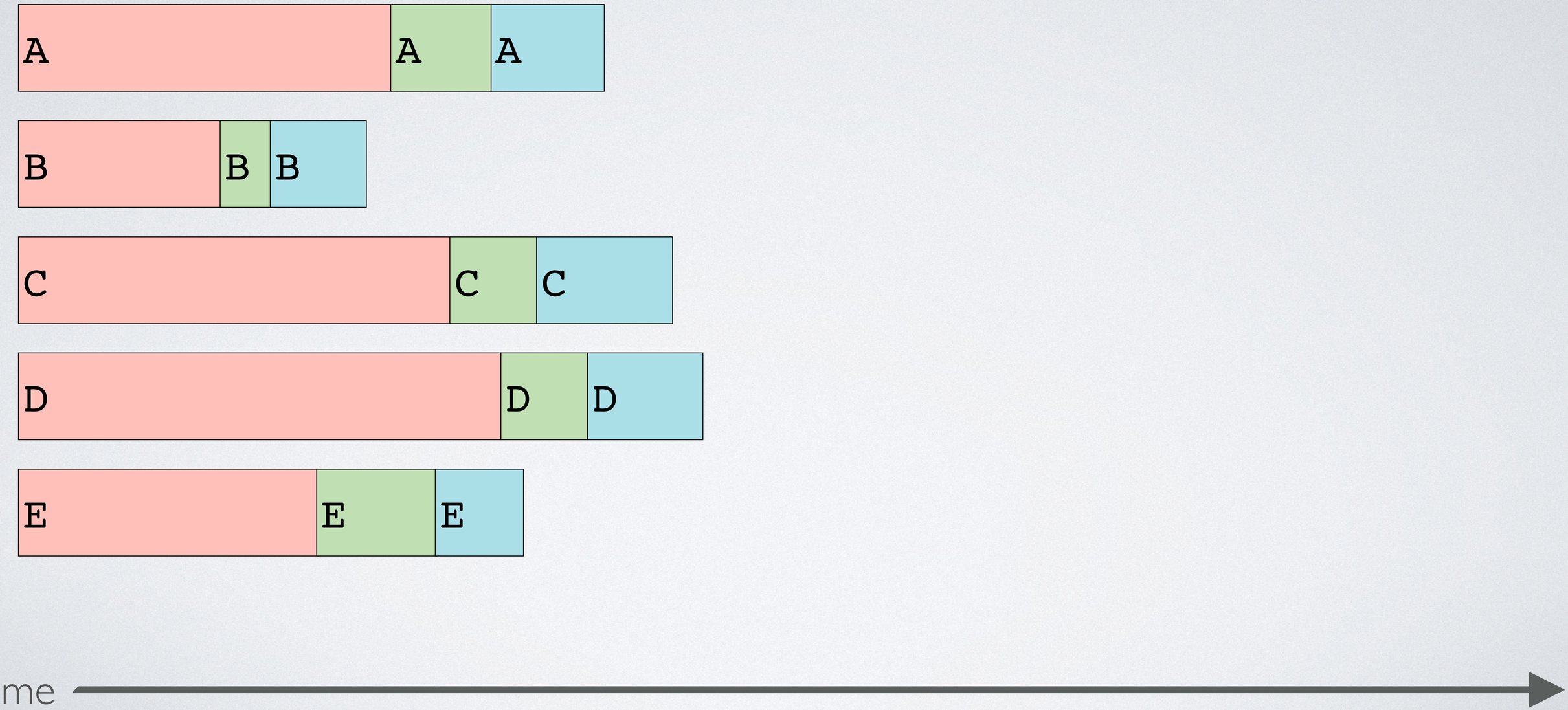

Time

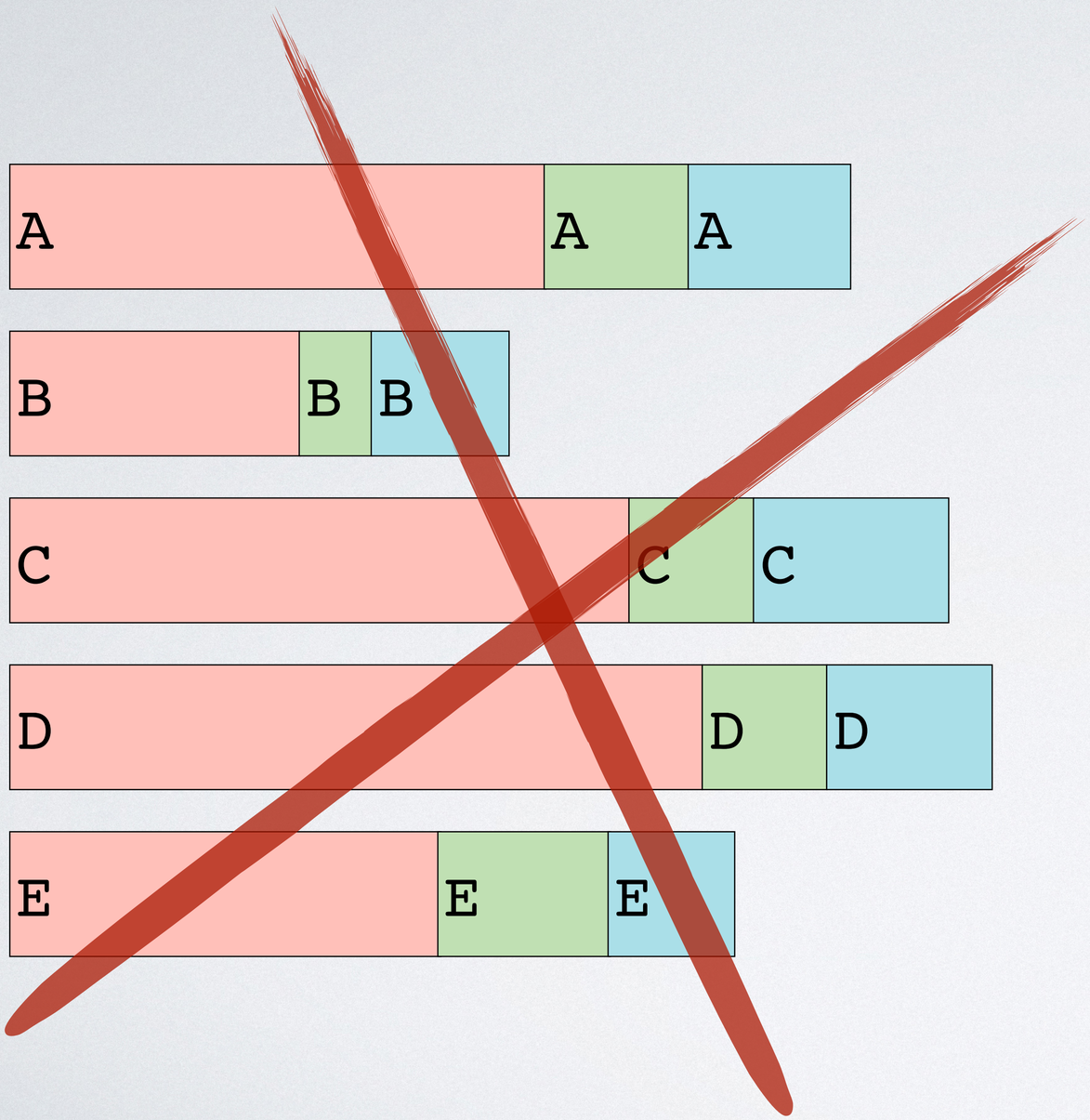

Sending
+
Waiting...
A
A
A
B
B
B
C
C
C
Working
Time




Time

# What about concurrency ?


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




# 🚫 Php is Single-Threaded

# Asynchronous HTTP calls

## Thanks to Guzzle

# Asynchronous way

```php
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

```php
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

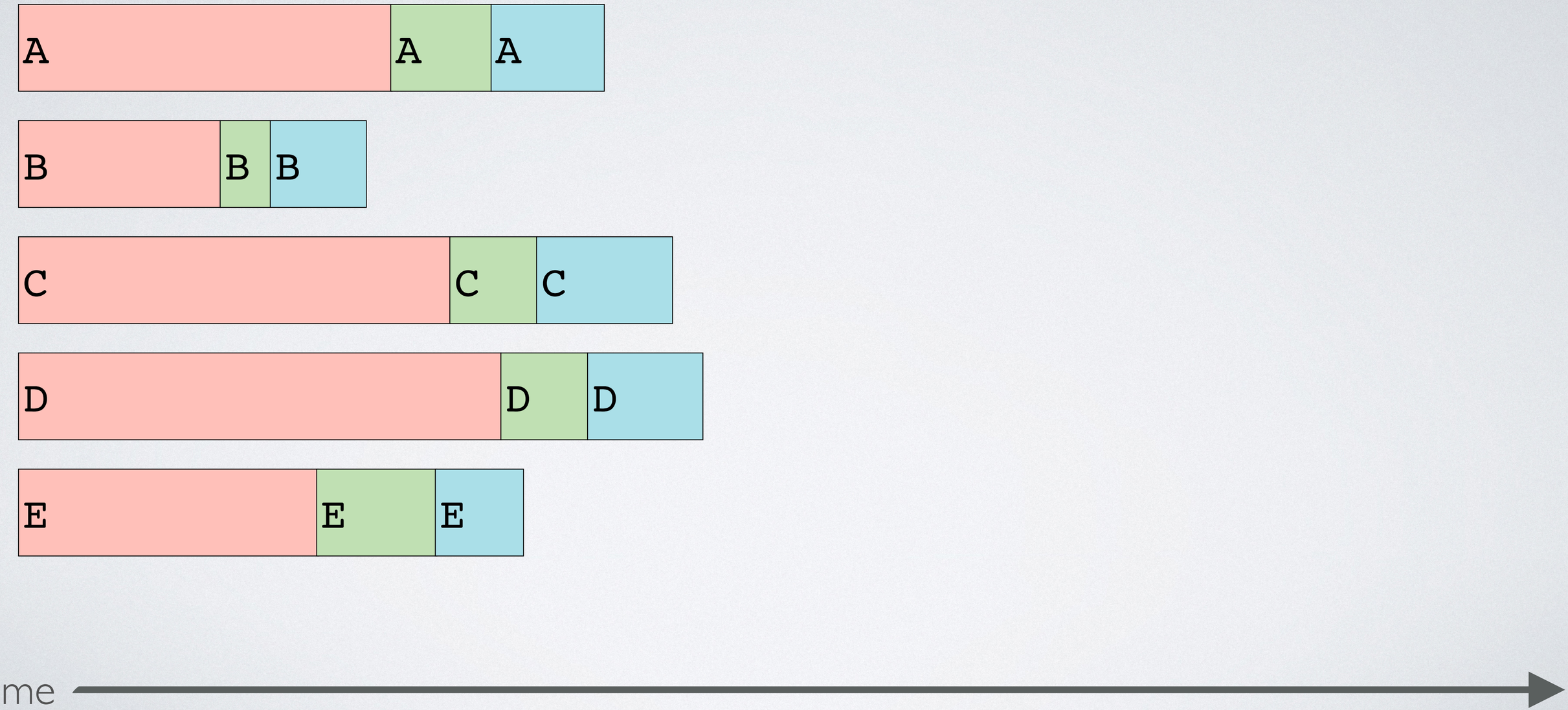
```php
function foo2(\GuzzleHttp\Client $client, RequestInterface ...$requests)
{
    $promises = [];
    foreach ($requests as $request) {
        $promises[] = $client->sendAsync($request);
    }

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

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

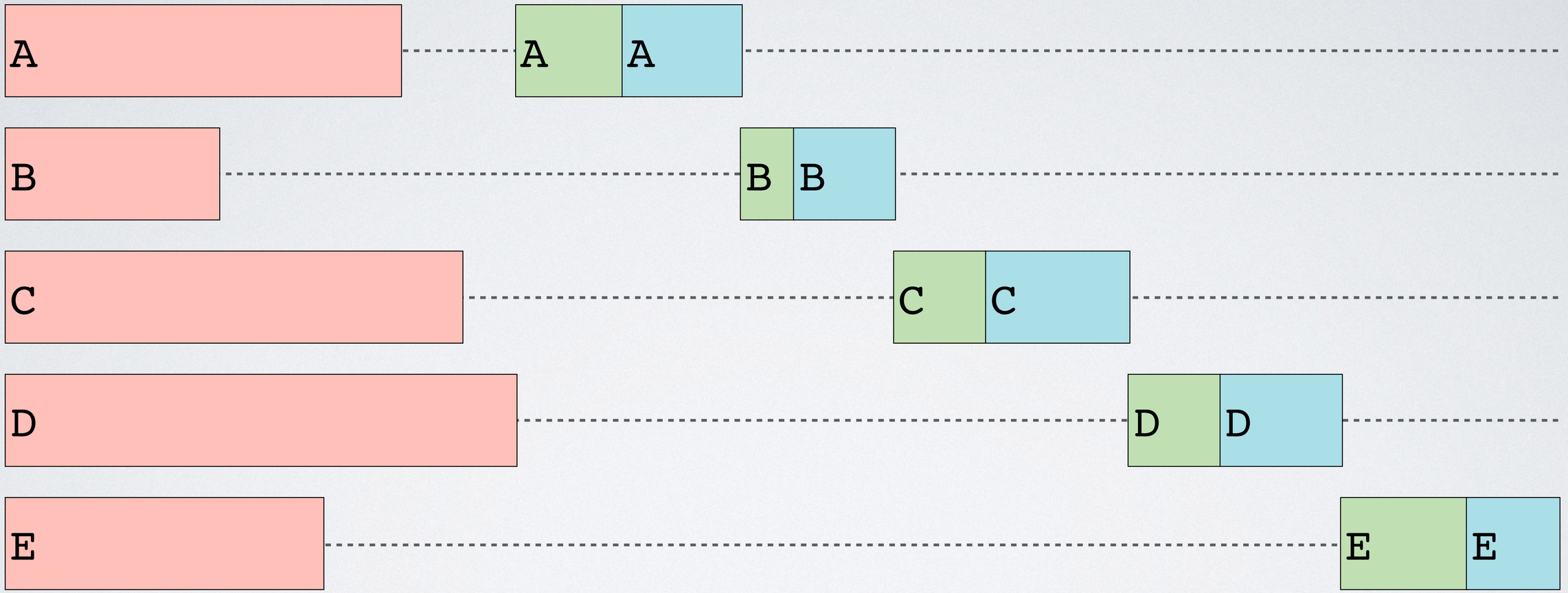
Waiting...

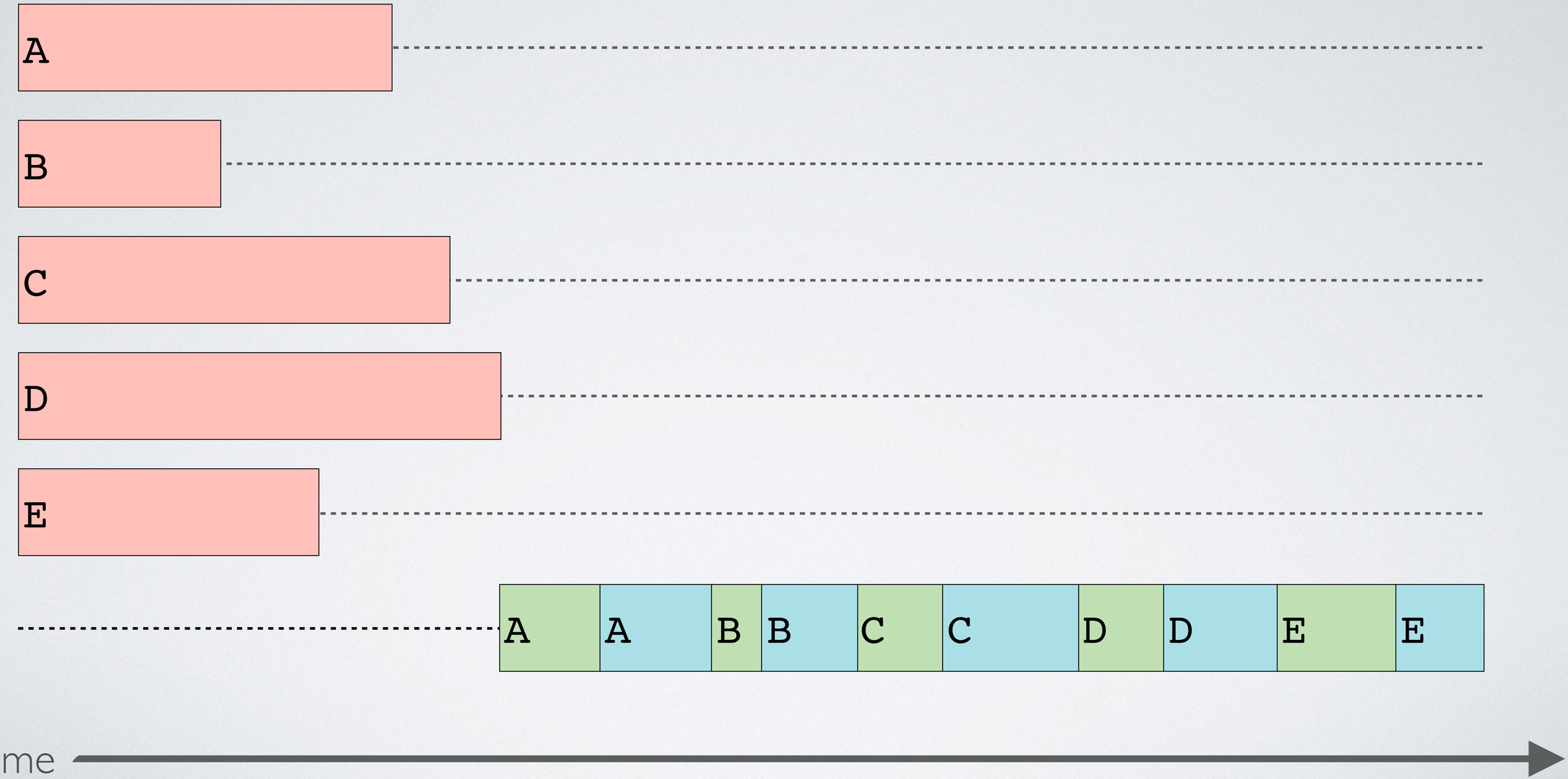
# Asynchronous way

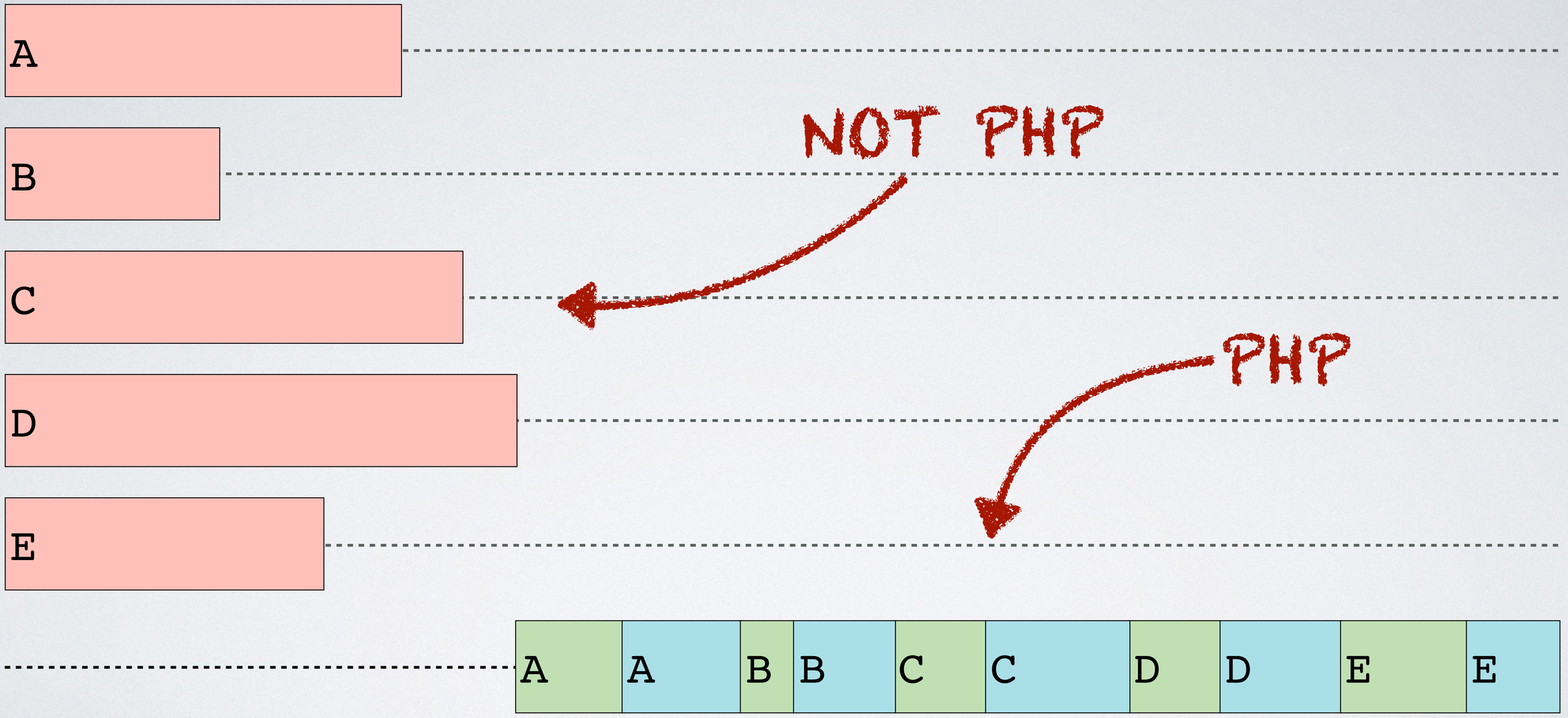
```php
function foo2(\GuzzleHttp\Client $client, RequestInterface ...$requests)
{
    $promises = [];
    foreach ($requests as $request) {
        $promises[] = $client->sendAsync($request);
    }

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

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

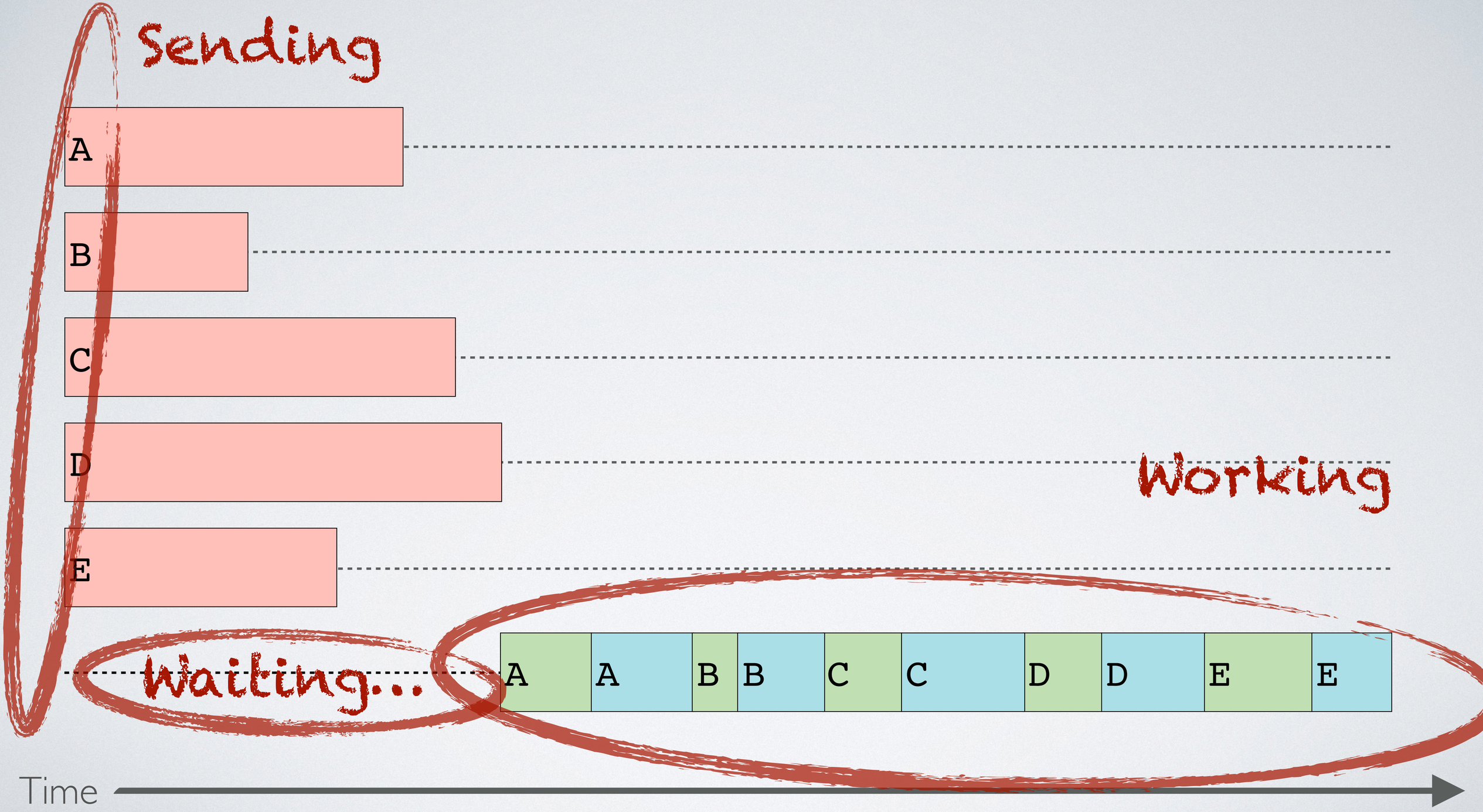
Working


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
Time


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


A
B
C
D
E
NOT PHP
PHP
A
A
B
B
C
C
D
D
E
E
Time


Sending
A
B
C
D
E
Working
Waiting...
A
A
B
B
C
C
D
D
E
E
Time

# We can do even better!

# 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);
```

# 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

```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
1 Request...

# Promises

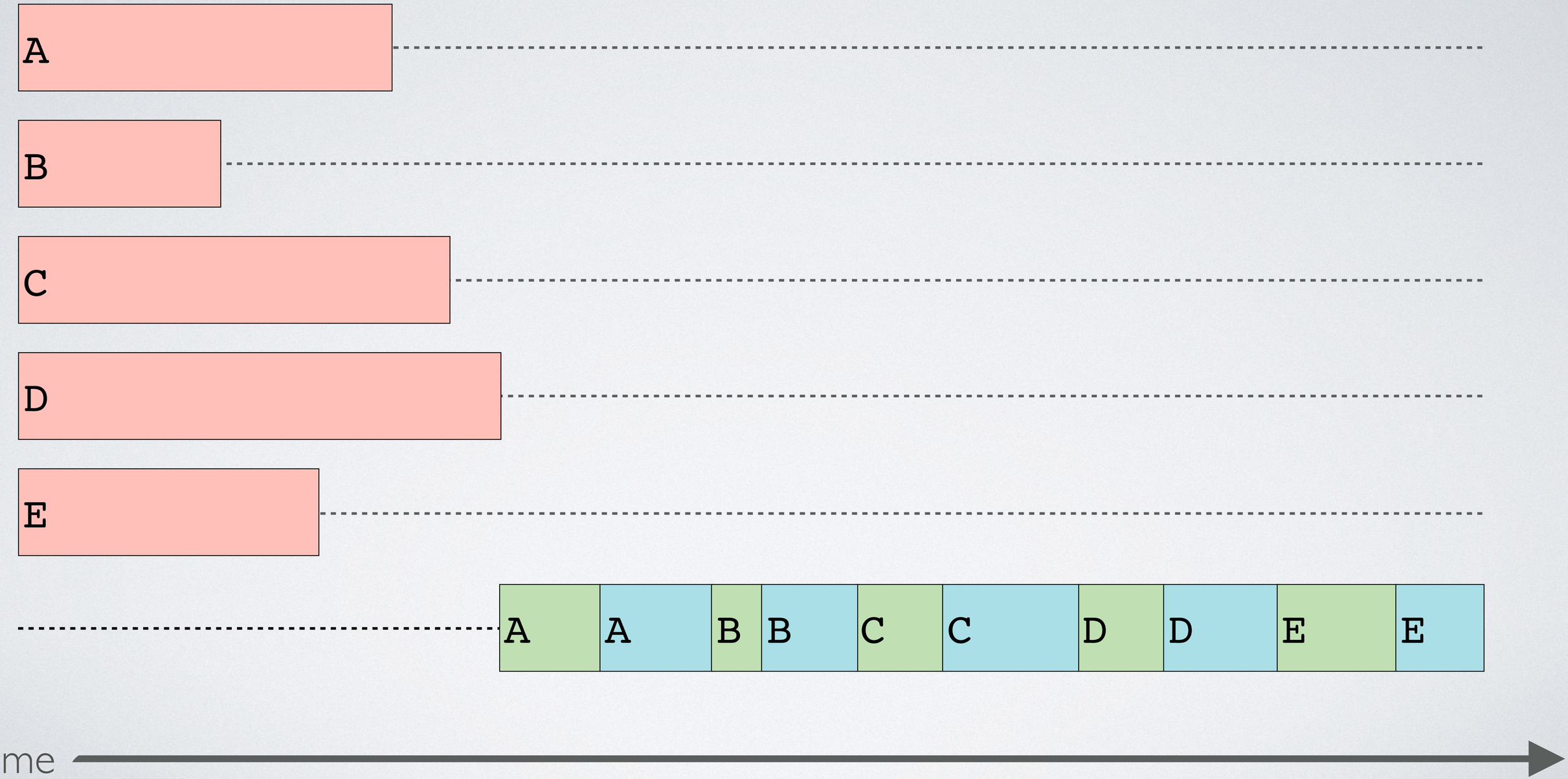
```
function foo3(\GuzzleHttp\Client $client, RequestInterface ...$requests)
{
    $promises = [];
    foreach ($requests as $request) {
        $promises[] = $client->sendAsync($request)->then(
            function (ResponseInterface $response) {
                $jsonArray = json_decode(
          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

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

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

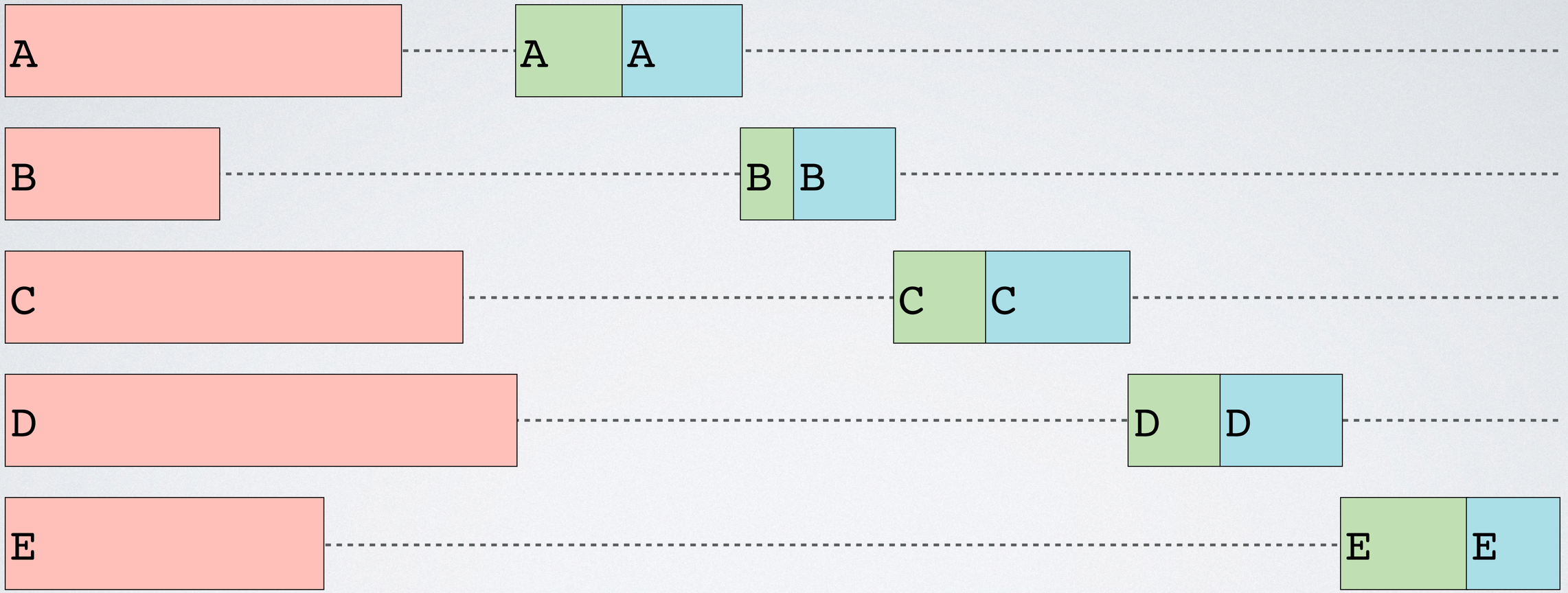
Working

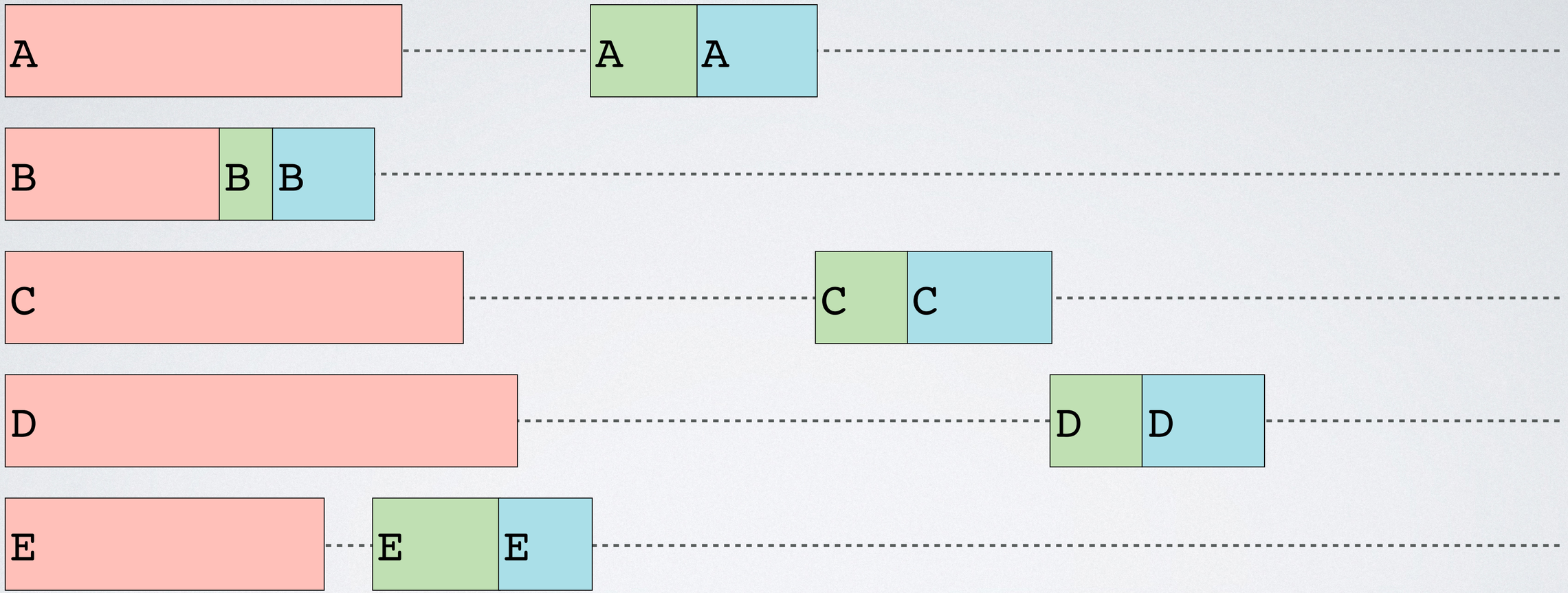
# Promises

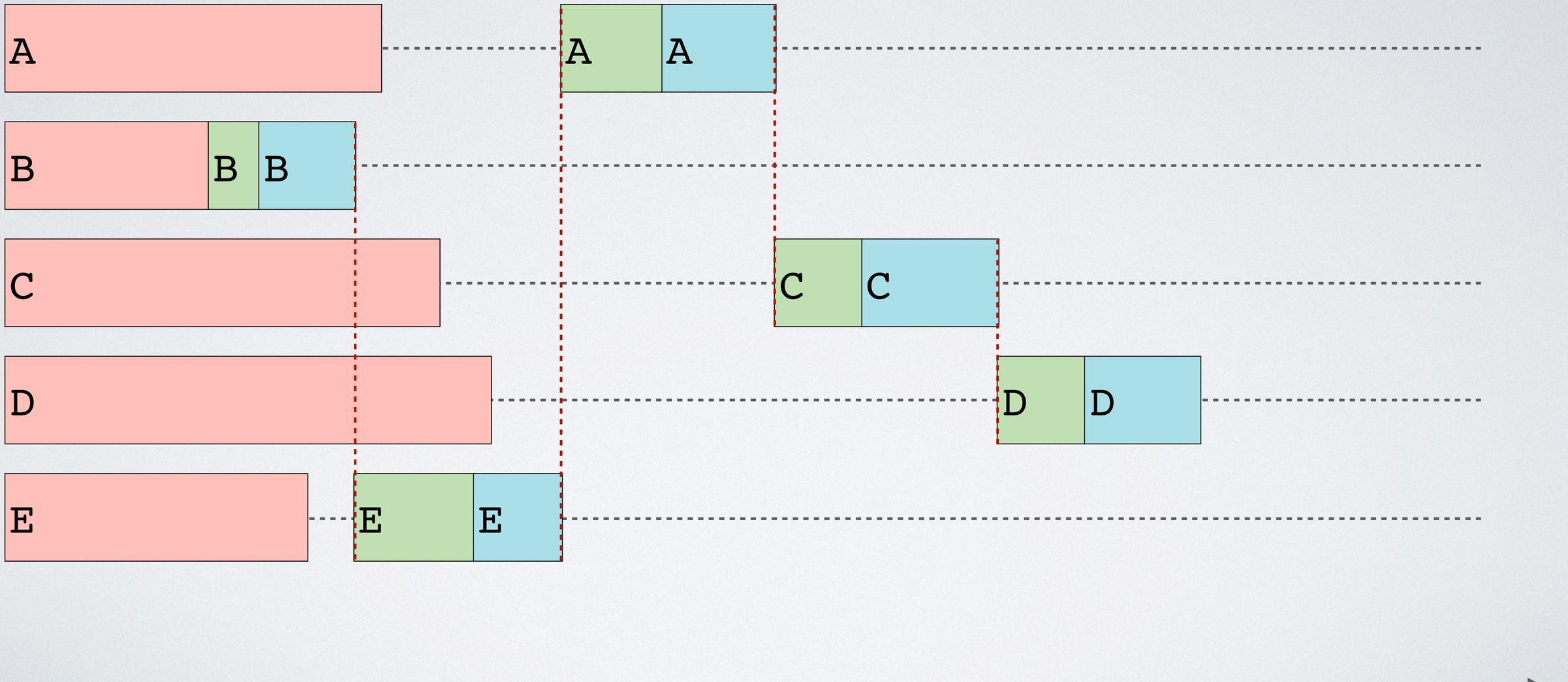
```
function foo3(\GuzzleHttp\Client $client, RequestInterface ...$requests)
{
    $promises = [];
    foreach ($requests as $request) {
        $promises[] = $client->sendAsync($request)->then(
            function (ResponseInterface $response) {
                $jsonArray = json_decode(
          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

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

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

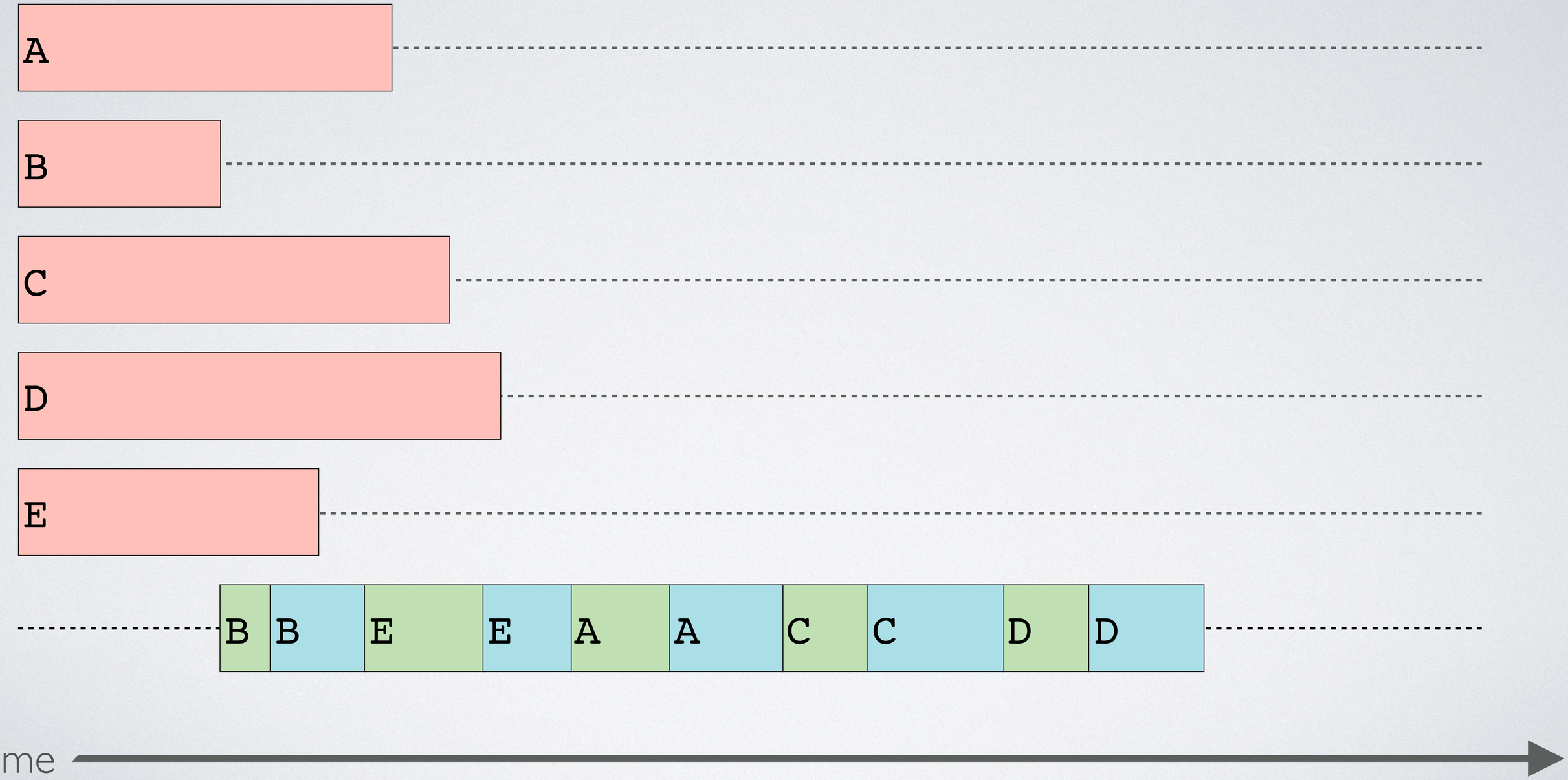
Waiting for all the jobs to finish


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

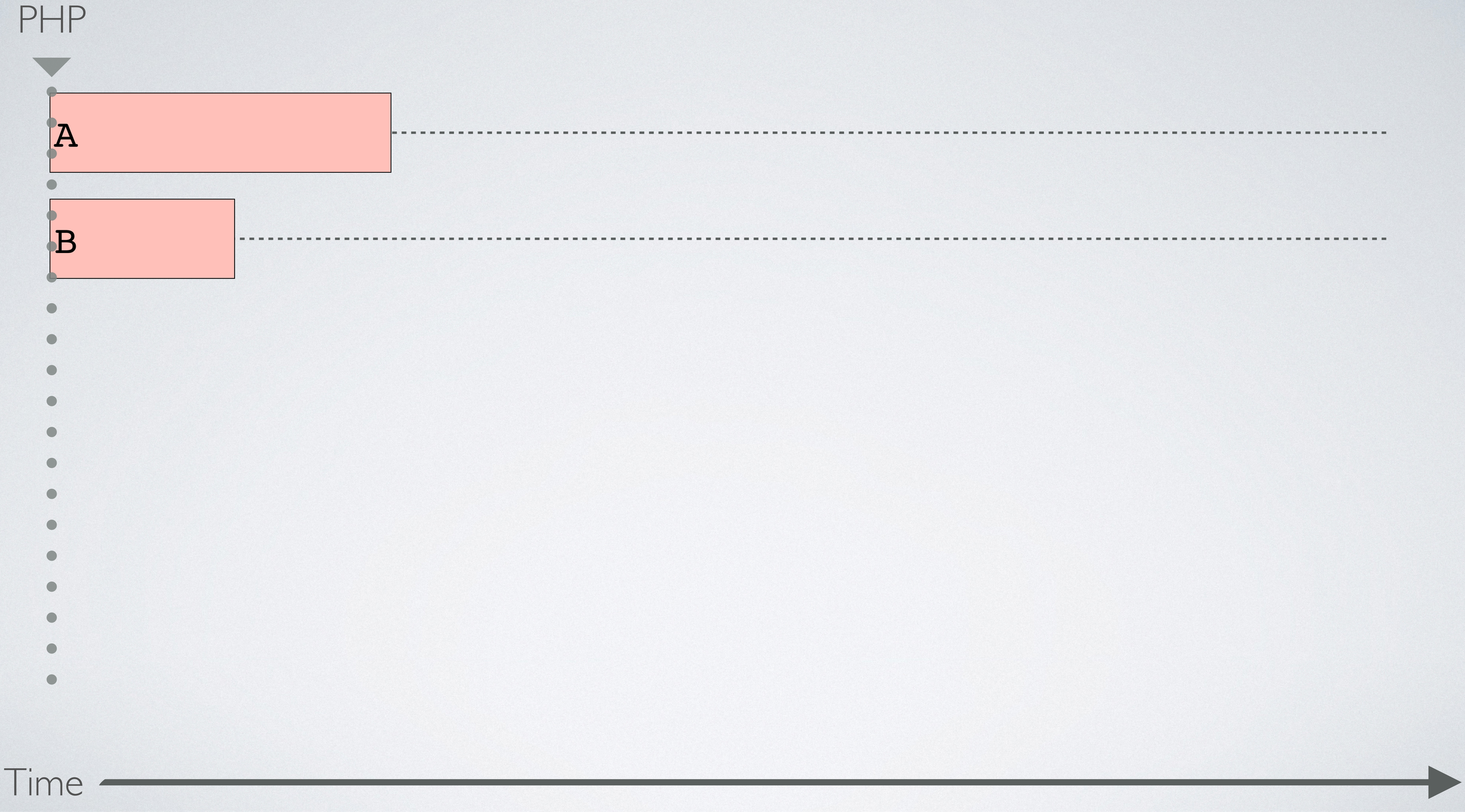

A
A
A
B
B
B
C
C
C
D
D
D
E
E
E
Time

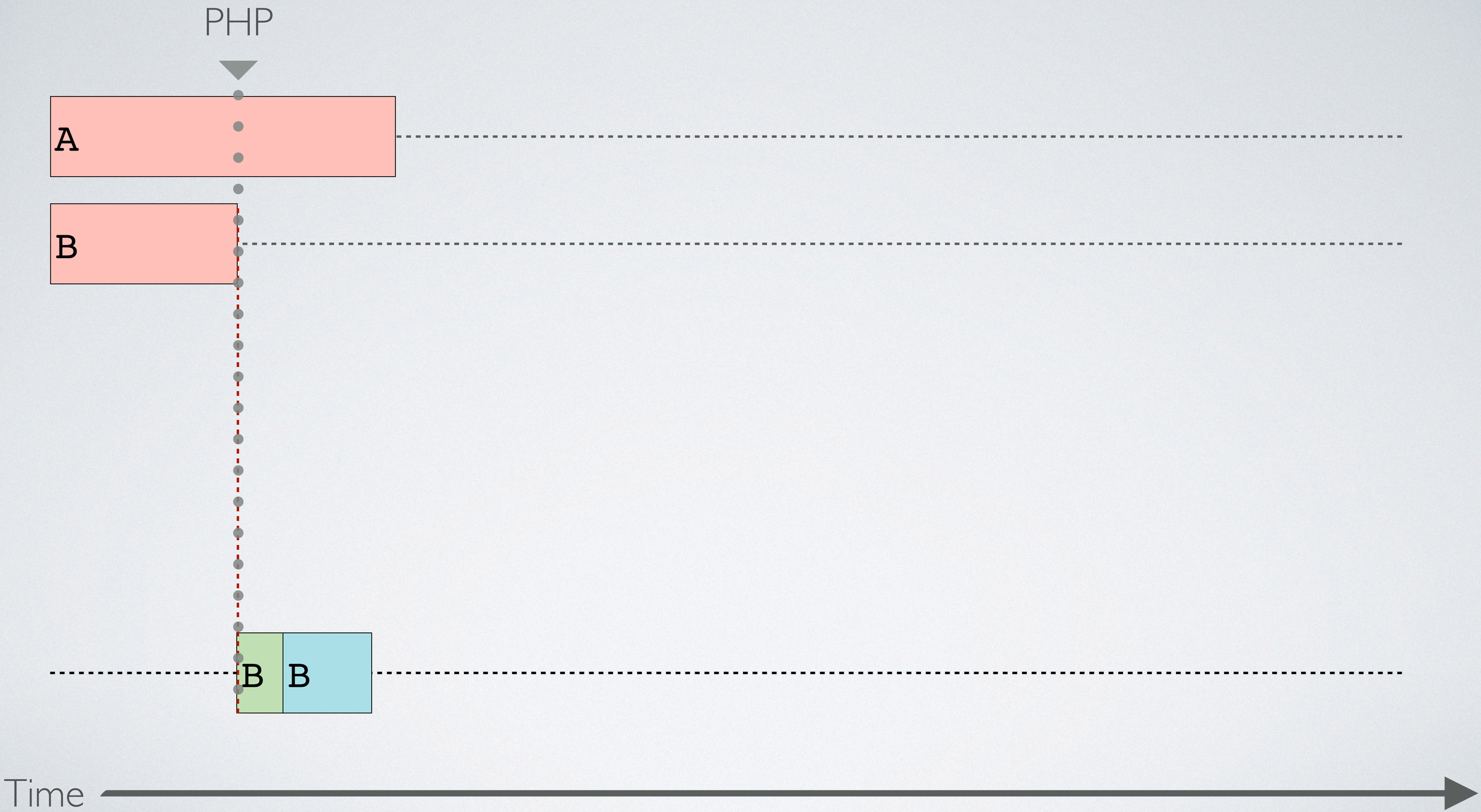

A
A
A
B
B
B
C
C
C
D
D
D
E
E
E
Time

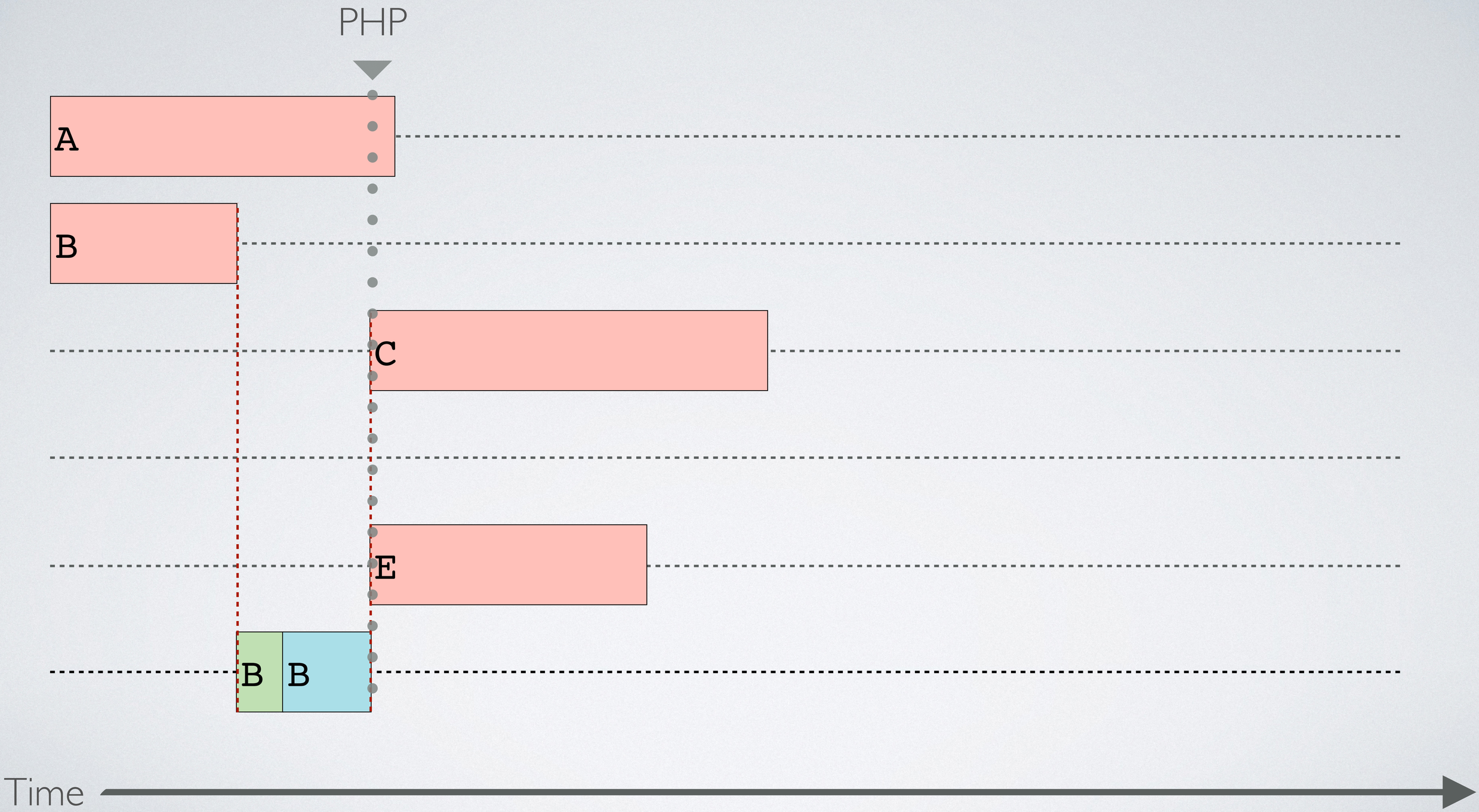

A
A
A
B
B
B
C
C
C
D
D
D
E
E
E
Time

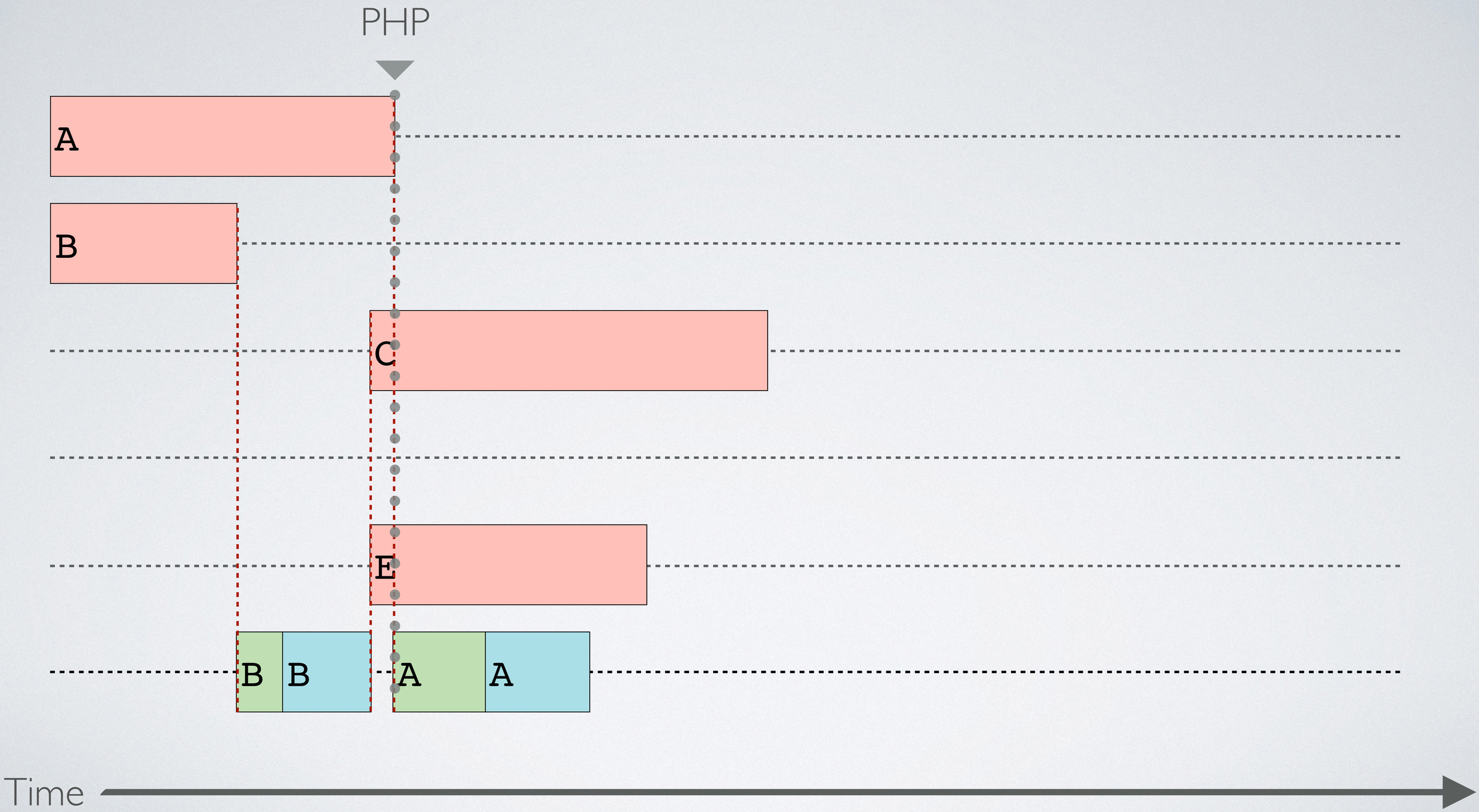

A
B
C
D
E
B B E E A A C C D D
Time

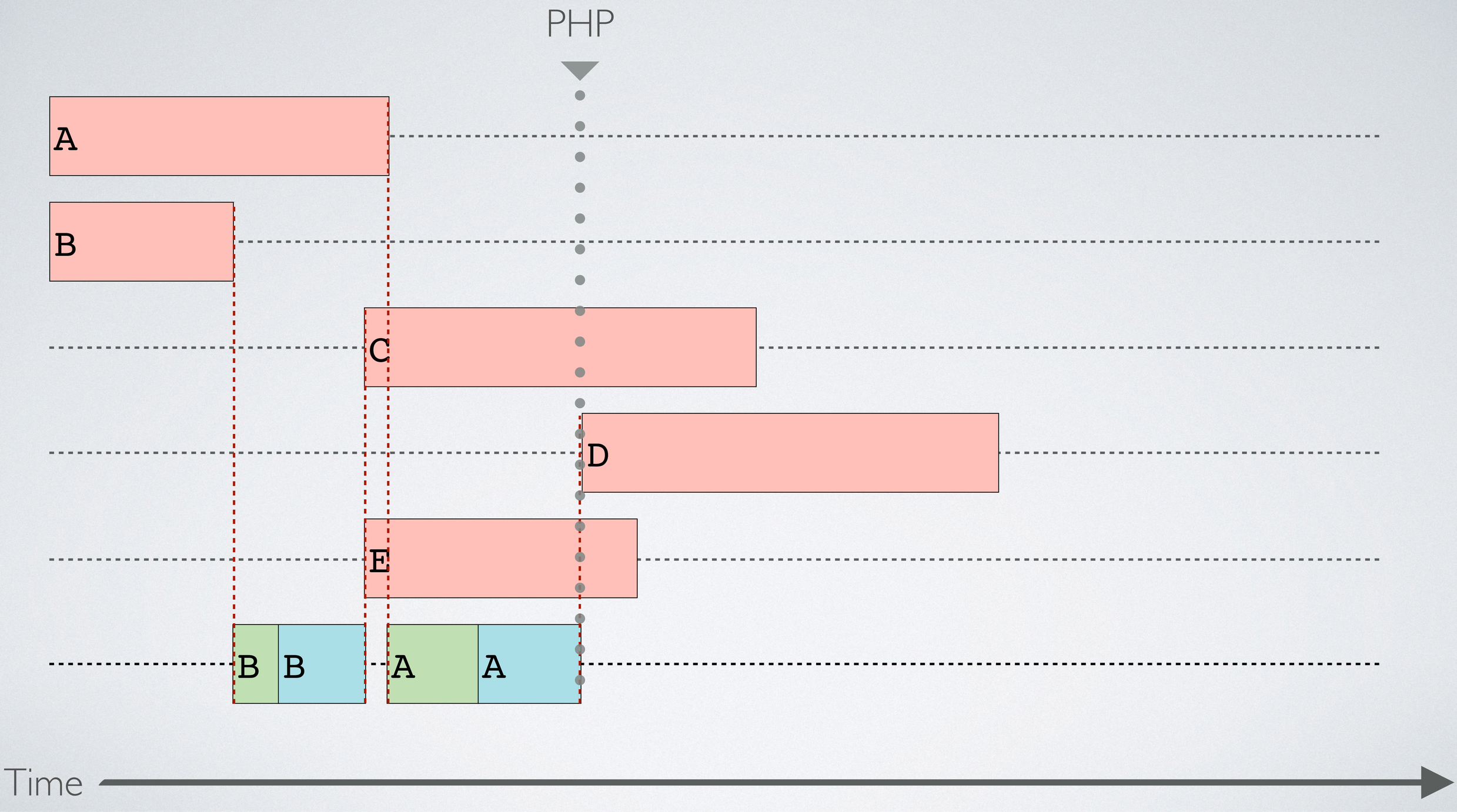

ASYNCHRONOUS PROGRAMMING
SPACE RANGER
LIGHTYEAR
EVERYWHERE

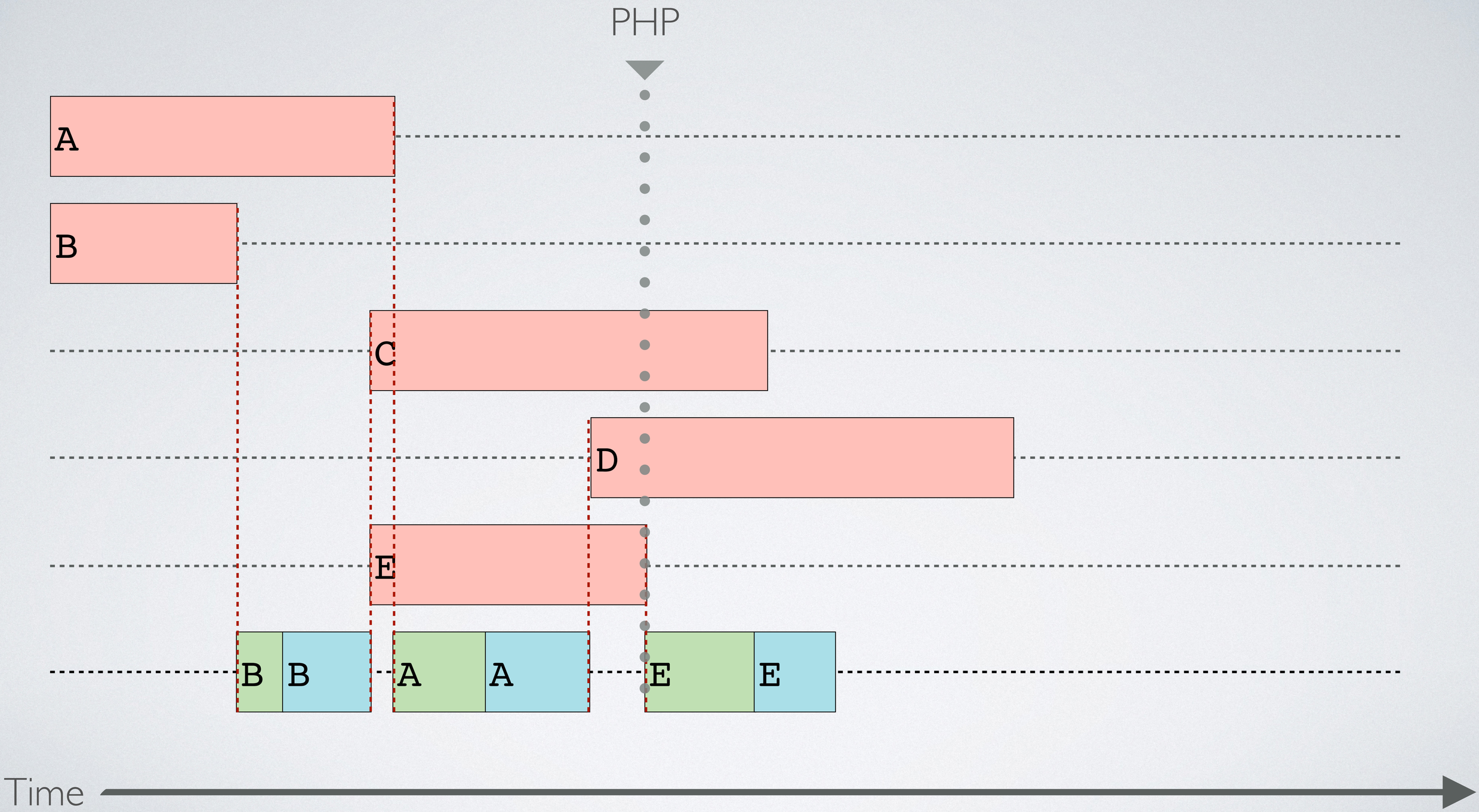

PHP
A
B
Time

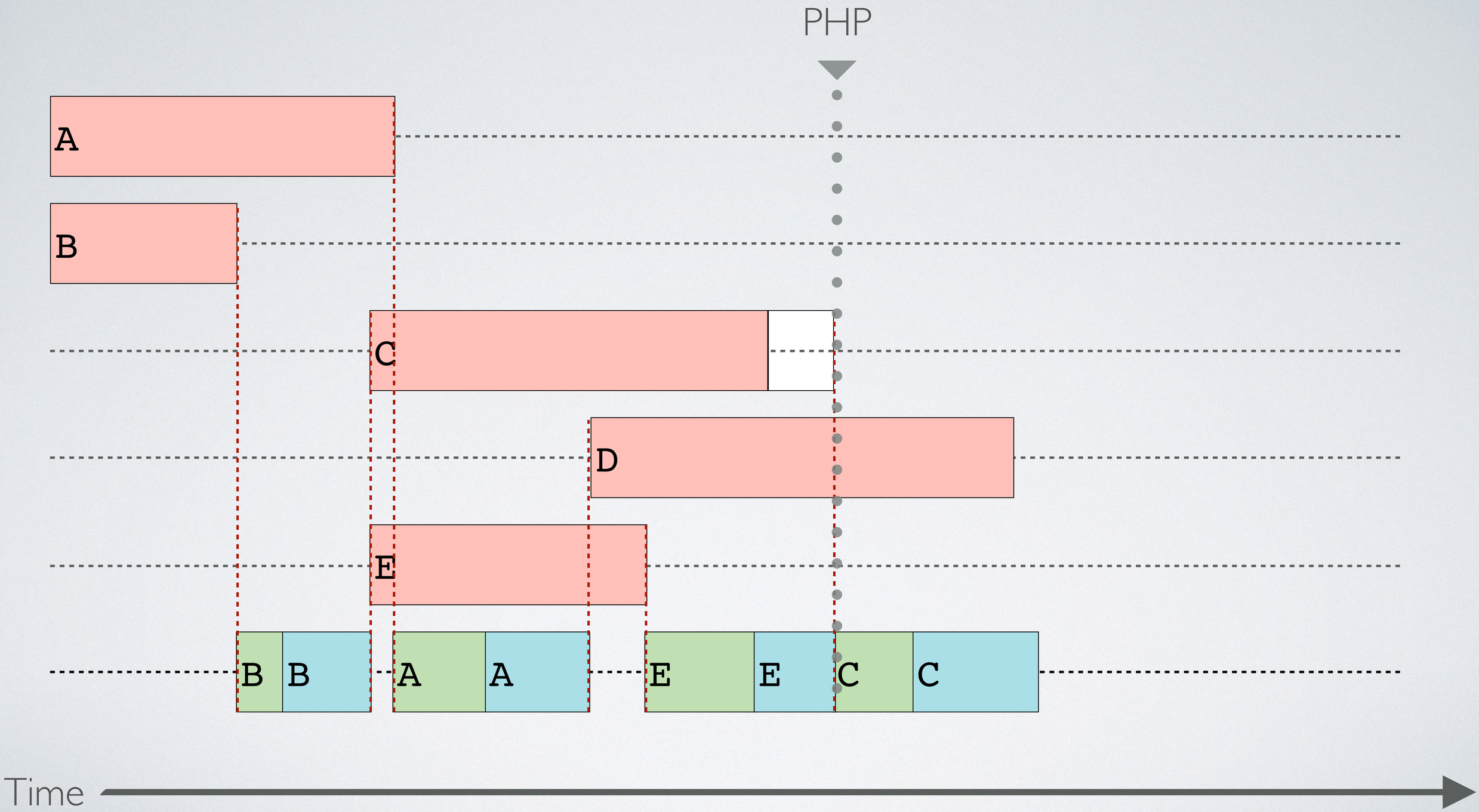

PHP
A
B
B
B
Time

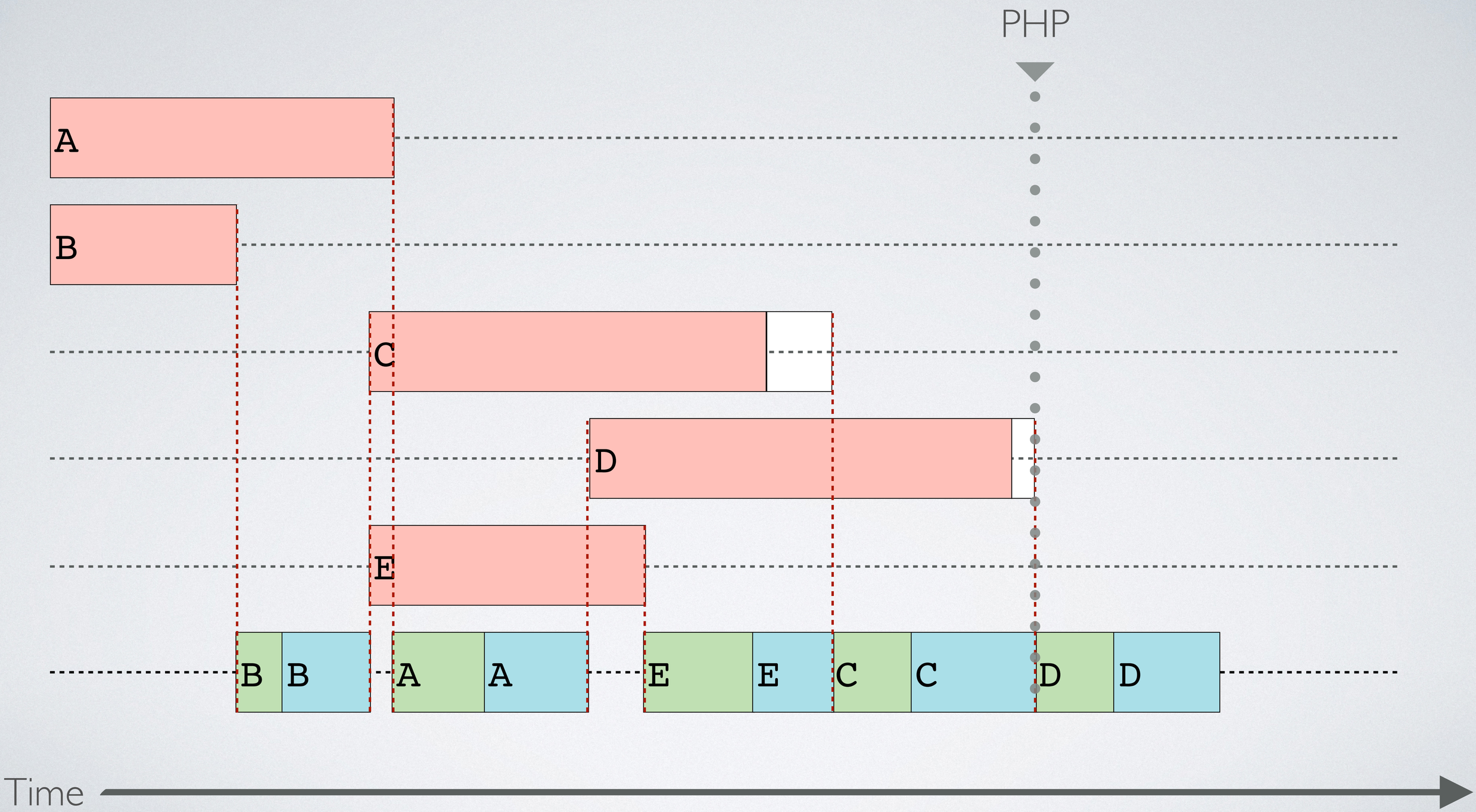

PHP
A
B
C
E
B
B
Time


PHP
A
B
C
E
B
B
A
A
Time


PHP
A
B
C
D
E
B
B
A
A
Time


PHP
A
B
C
D
E
B
B
A
A
E
E
Time


PHP
A
B
C
D
E
B
B
A
A
E
E
C
C
Time


PHP
A
B
C
D
E
B B A A E E C C D D
Time

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

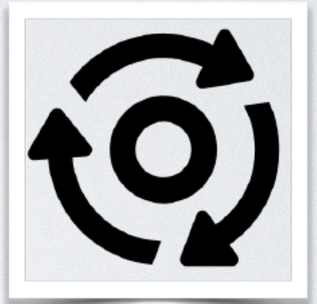
# Which framework can I use?

Kraken

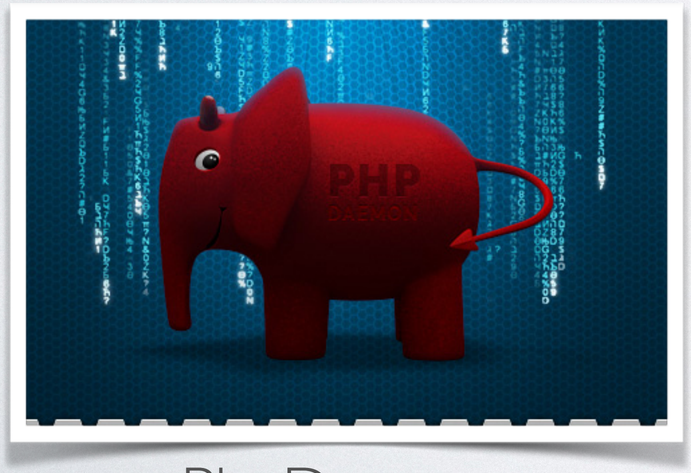
icicle.io

swoole


Amp



Recoil




PhpDaemon

Guzzle


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*

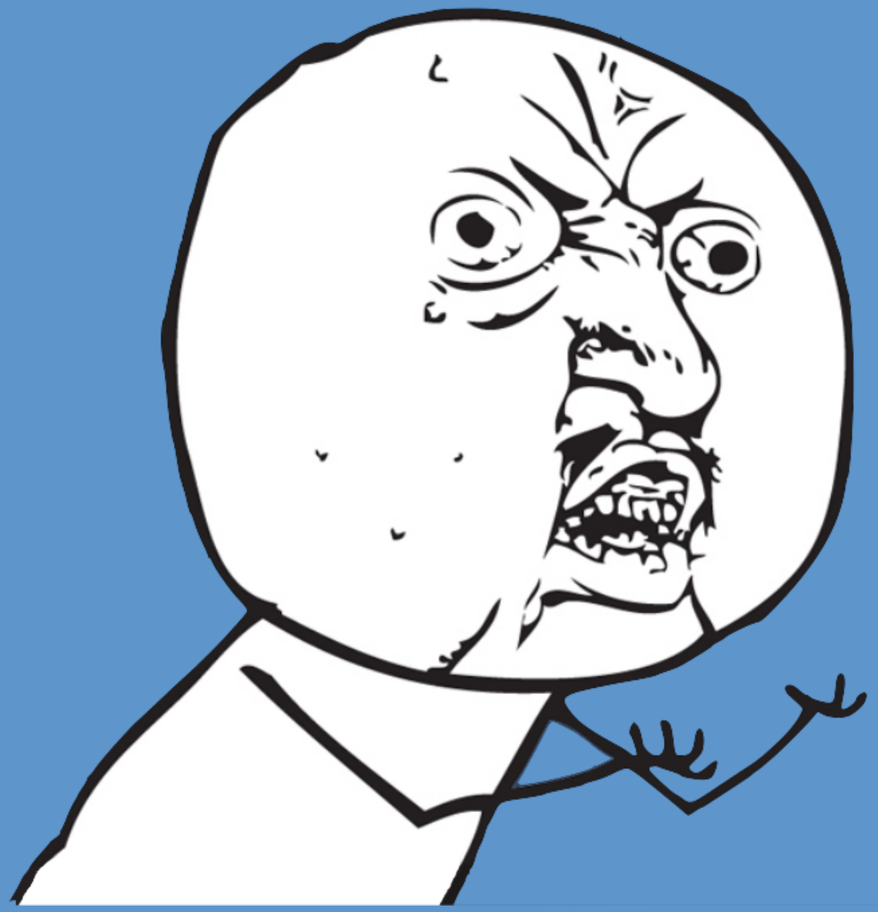
# *Yieldable*

```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.

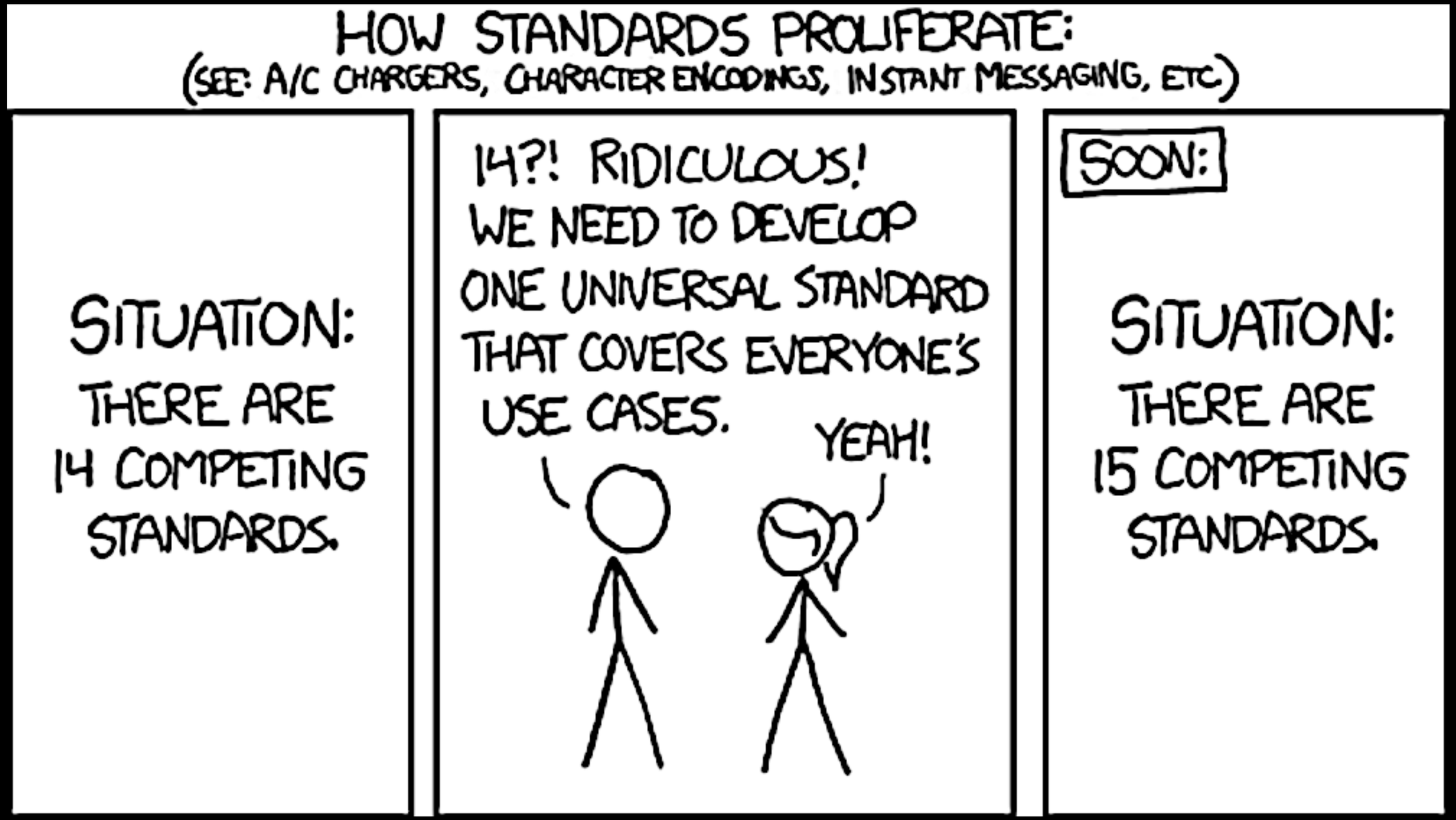


# ASYNCHRONOUS PROGRAMMING SHOULD BE A DETAIL OF IMPLEMENTATION

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

# What about a standard?


IT’S COMPLICATED




https://xkcd.com/927/

# Tornado

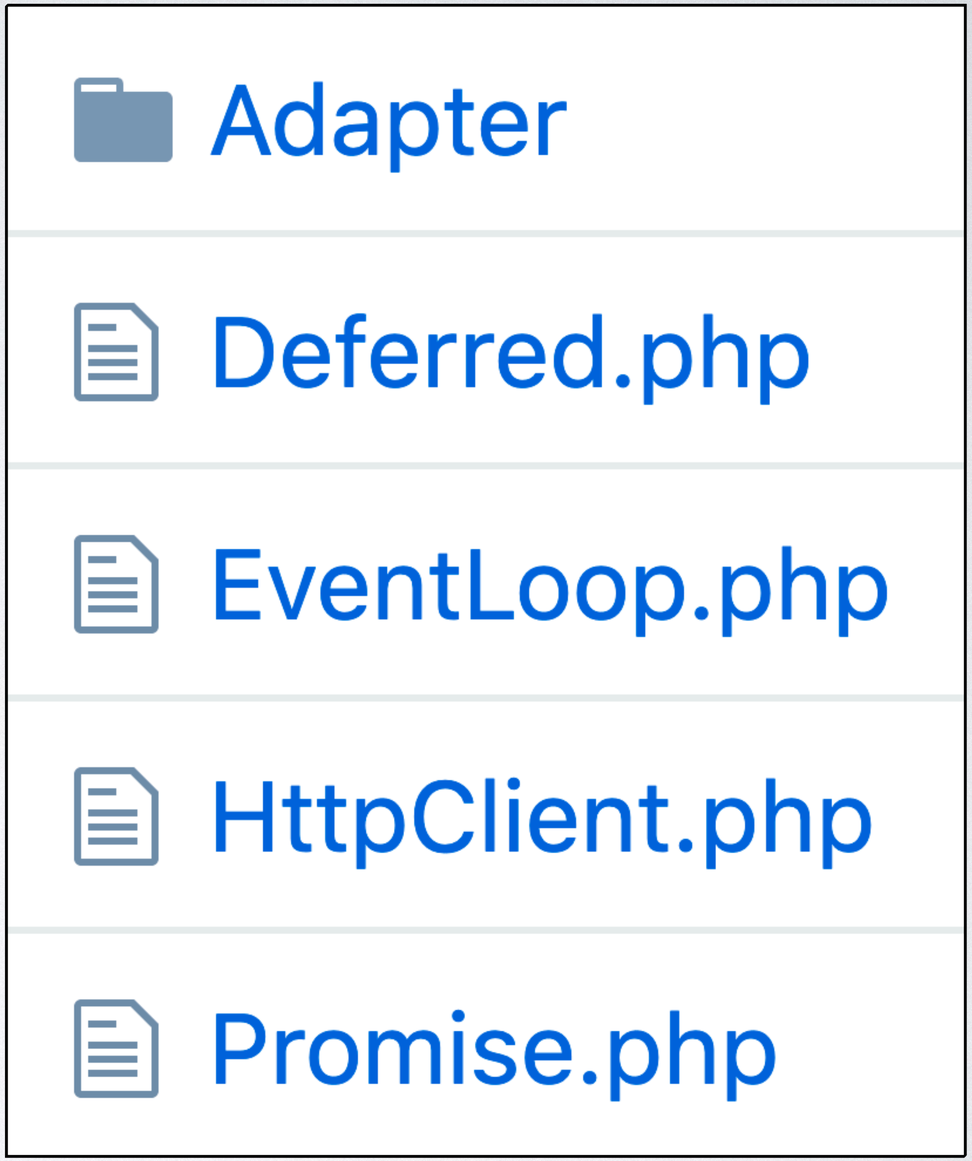
https://github.com/M6Web/Tornado

```
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

⚠️ Memory

⚠️ What if it crashes?

⚠️ Is your stack ready for that?

Asynchronous programing does **not** require running a Php **Http Server**

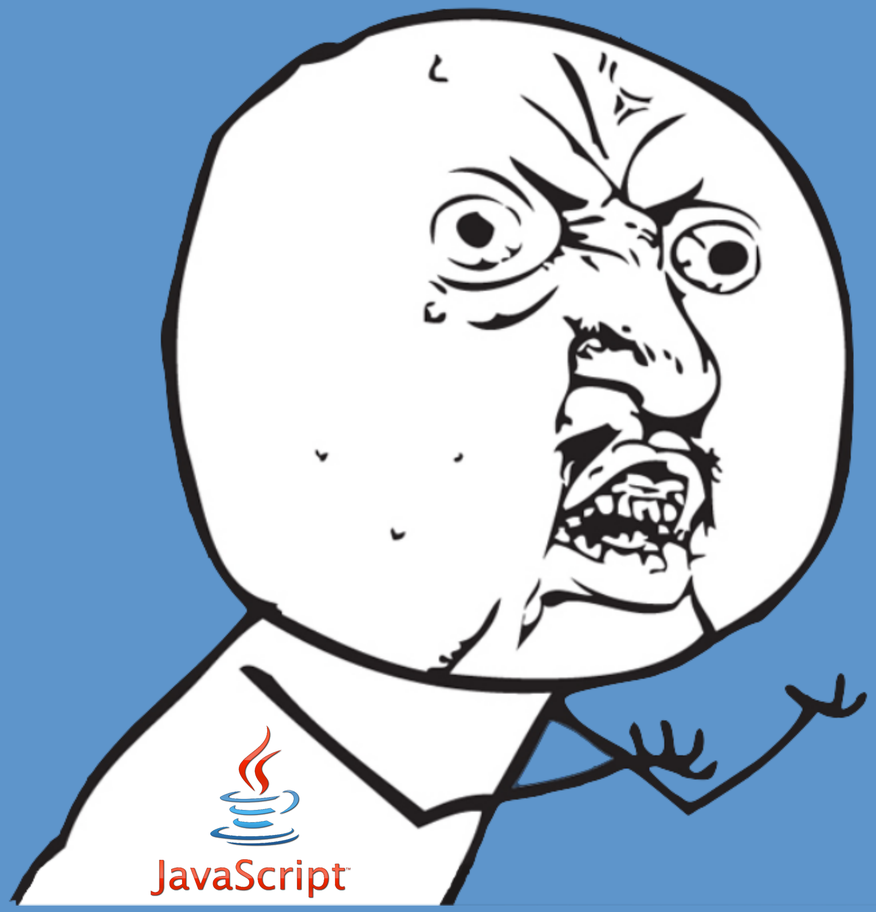
# « Local » Event Loop

```php
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

# The « *J* » syndrom

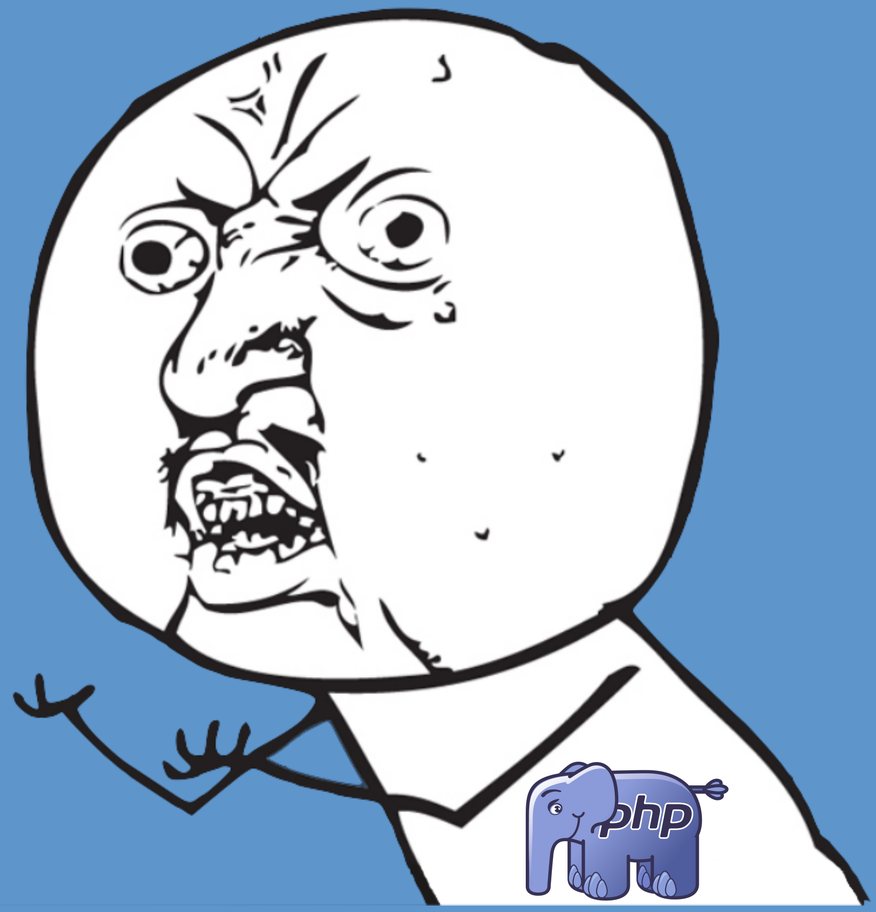



# PHP DEV

# Y U NO USE JAVASCRIPT

# PHP DEV

# Y U NO USE PHP



# *Chapter 2*

## Let it go!

# HttpClient

```php
interface HttpClient
{
    /**
     * Sends a http request and returns a promise that will be resolved with a
     * Psr\Http\Message\ResponseInterface
     */
    public function sendRequest(RequestInterface $request): Promise;
}
```

# Promise

```
/**
 * To resolve the value of a promise, you have to yield it from a generator registered in
 * the event loop.
 */
interface Promise
{
}
```

# EventLoop

```
interface EventLoop {
    public function wait(Promise $promise);

    public function async(\Generator $generator): Promise;

    public function promiseFulfilled($value): Promise;
    public function promiseRejected(\Throwable $throwable): Promise;

    public function promiseAll(Promise ...$promises): Promise;
    public function idle(): Promise;
    public function delay(int $milliseconds): Promise;

    public function deferred(): Deferred;
}
```

# Deferred

```
interface Deferred
{

    public function getPromise(): Promise;

    public function resolve($value): void;

    public function reject(\Throwable $throwable): void;
}
```

# Adapters

HttpClient

- Guzzle
- Symfony

EventLoop

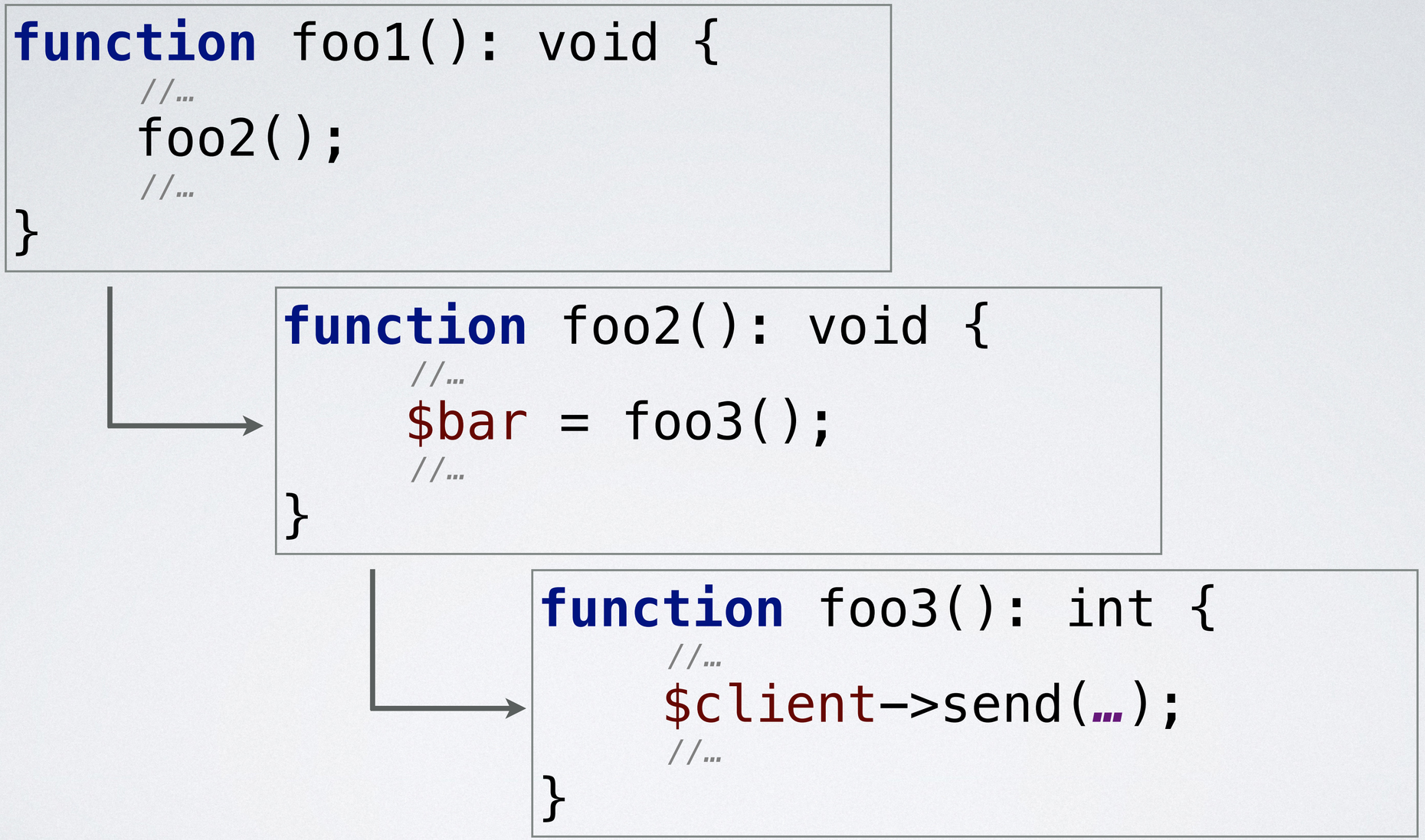
- ReactPhp
- Amp
- Tornado

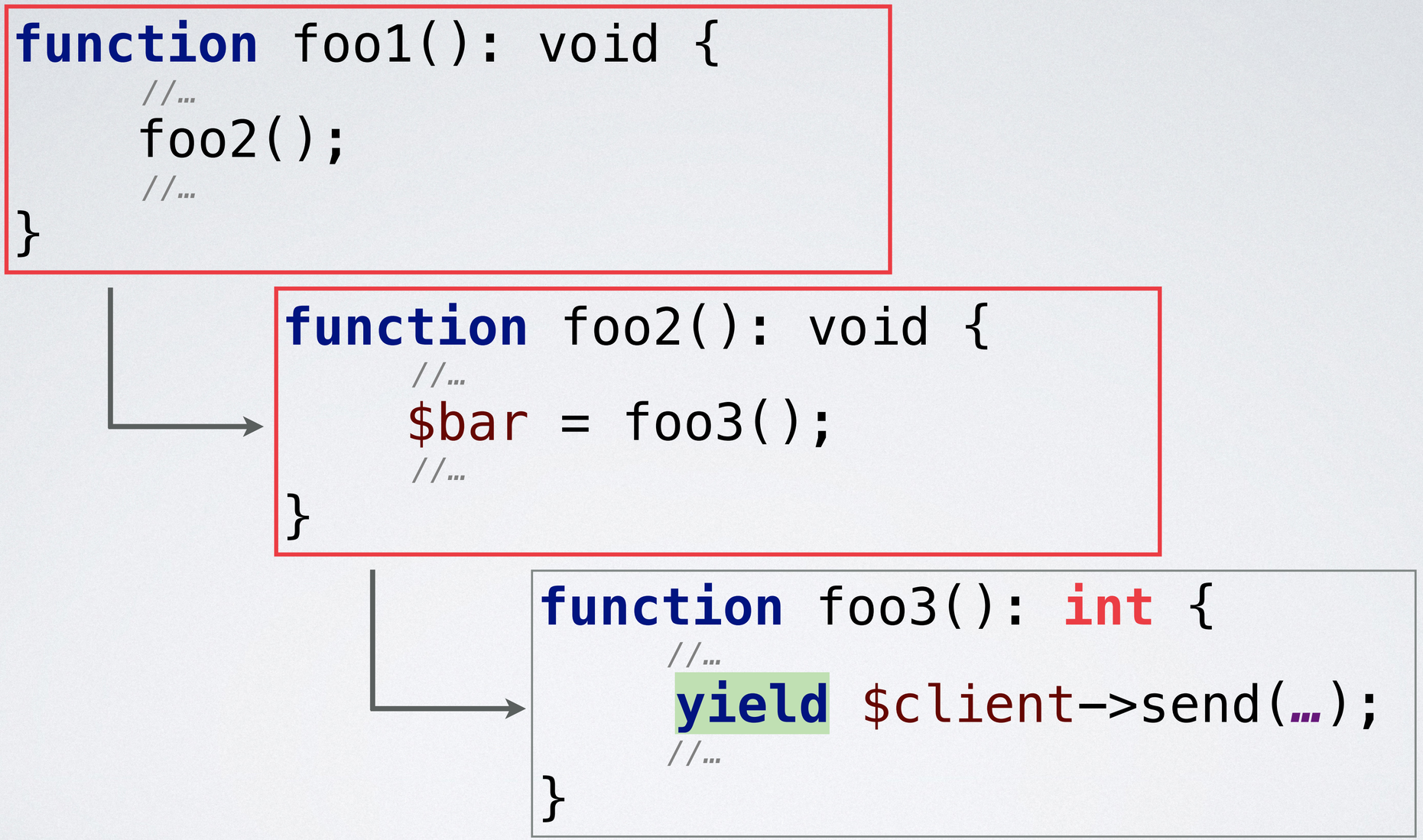
# POC

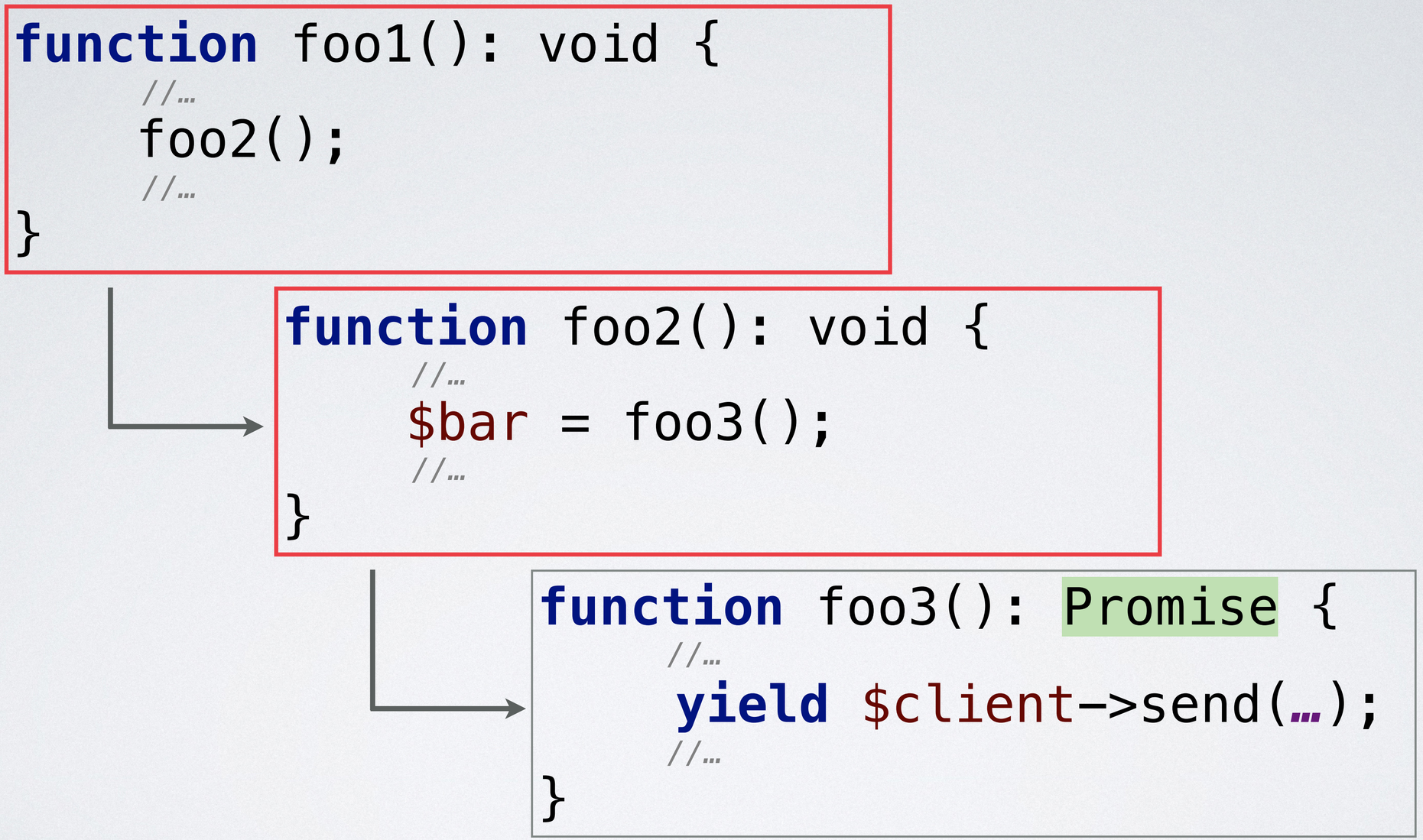
- overblog/GraphQLBundle
  https://github.com/b-viguier/tornado-graphqlbundle-demo/
- AWS SDK
  https://github.com/b-viguier/tornado-workshop

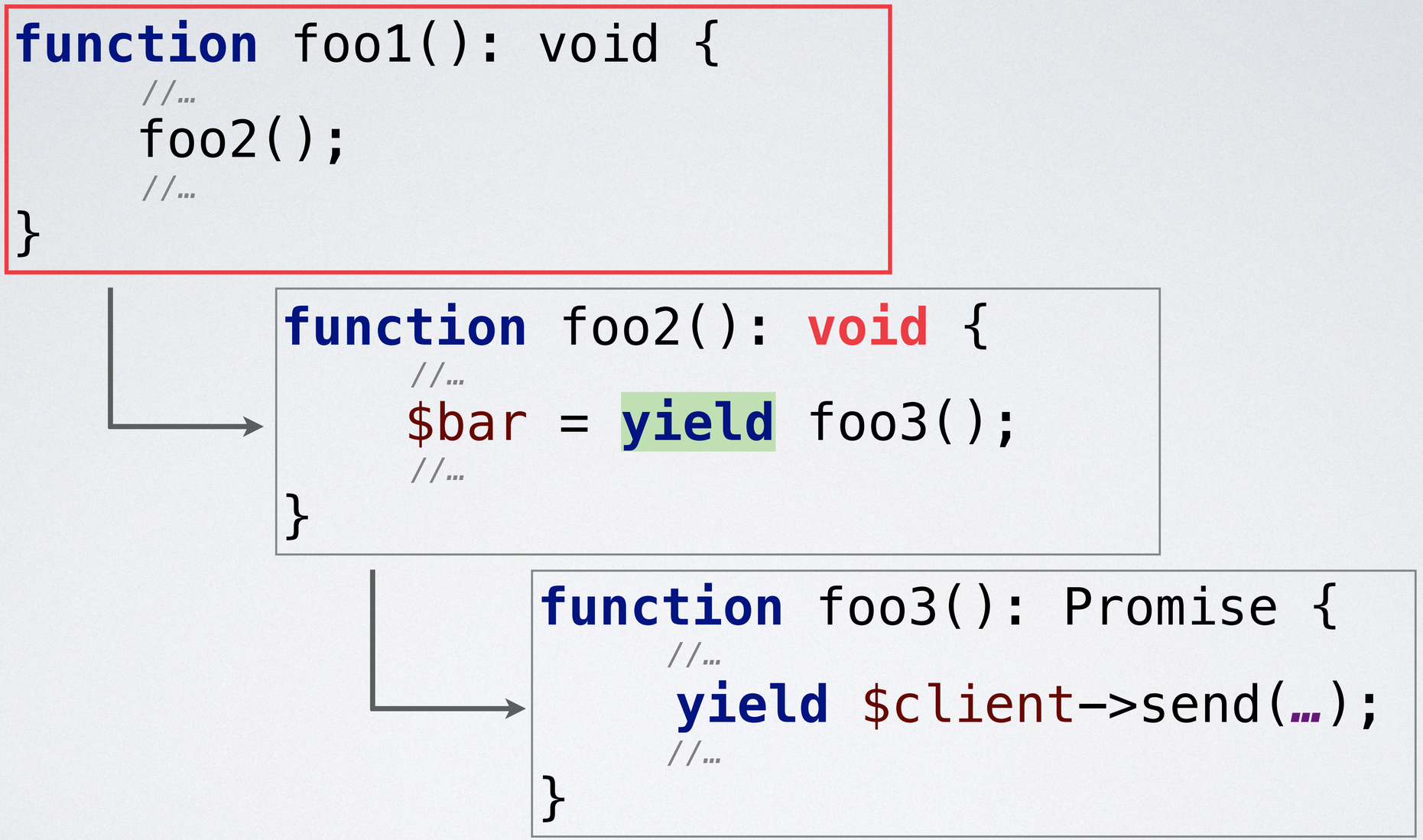
# Where do I start?

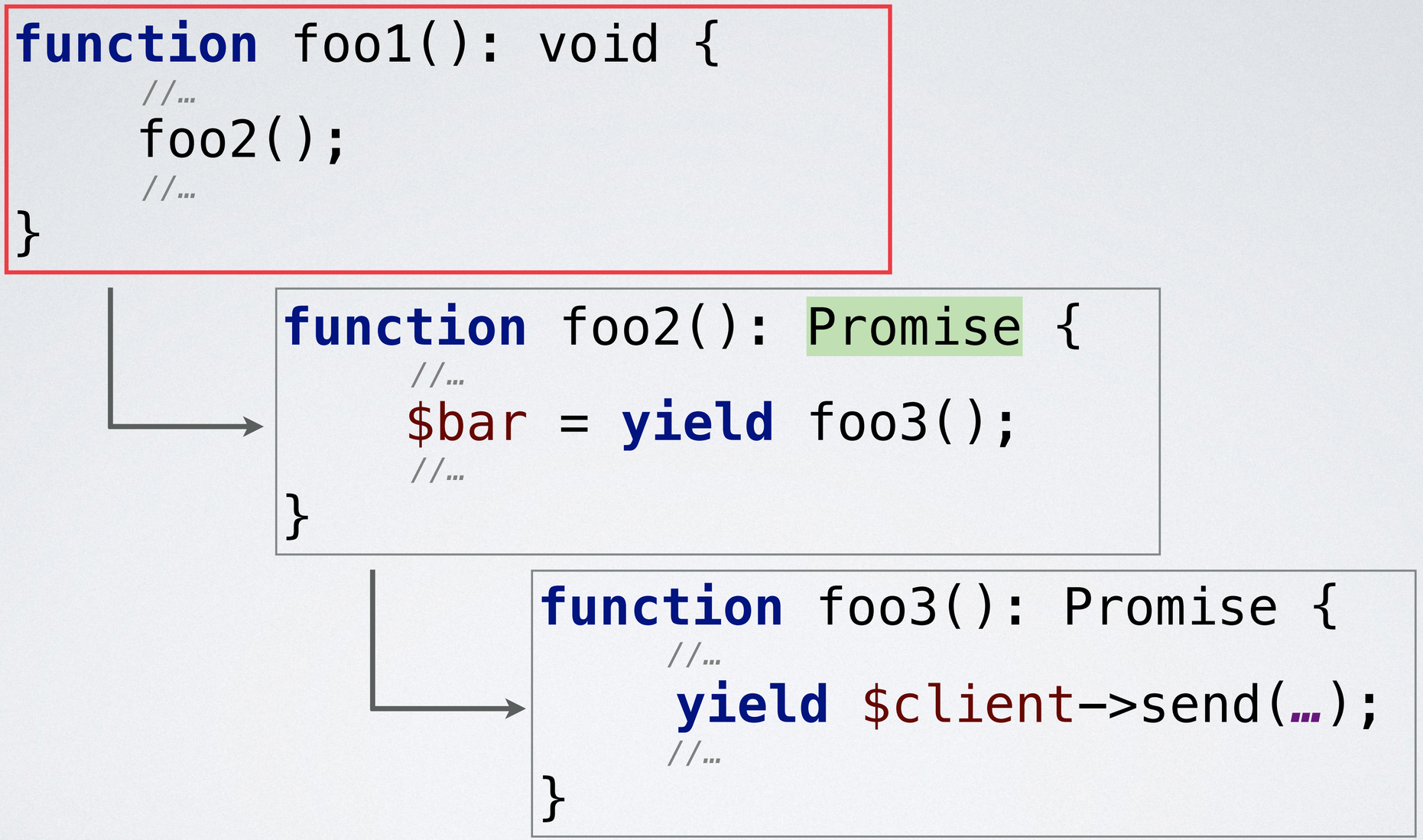
It's **easier** to use asynchronous programming from the **beginning**

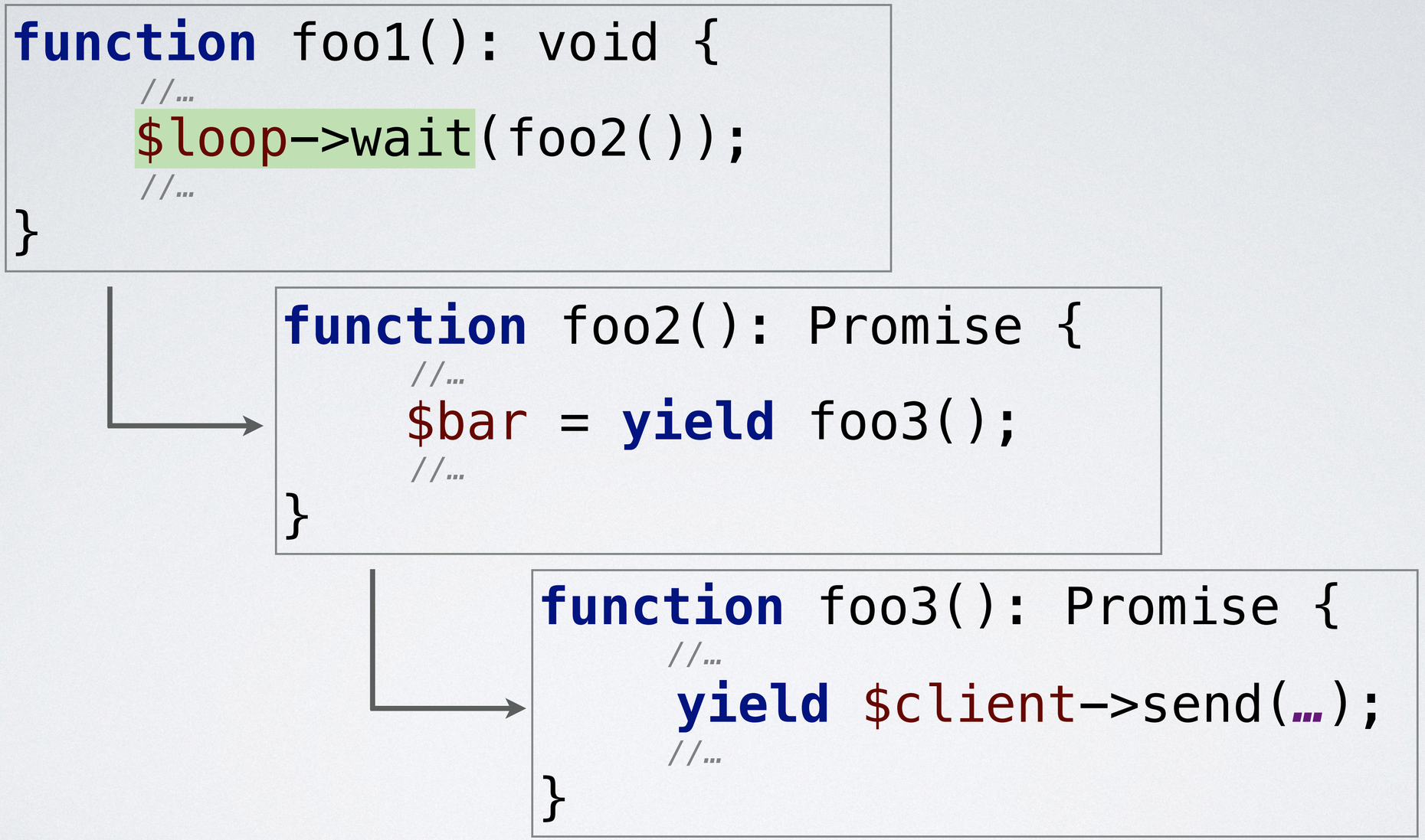

function foo1(): void {
//…
foo2();
//…
}
function foo2(): void {
//…
$bar = foo3();
//…
}
function foo3(): int {
//…
$client->send(…);
//…
}


function foo1(): void {
//…
foo2();
//…
}
function foo2(): void {
//…
$bar = foo3();
//…
}
function foo3(): int {
//…
yield $client->send(…);
//…
}


function foo1(): void {
//…
foo2();
//…
}
function foo2(): void {
//…
$bar = foo3();
//…
}
function foo3(): Promise {
//…
yield $client->send(…);
//…
}


function foo1(): void {
//…
foo2();
//…
}
function foo2(): void {
//…
$bar = yield foo3();
//…
}
function foo3(): Promise {
//…
yield $client->send(…);
//…
}


function foo1(): void {
//…
foo2();
//…
}
function foo2(): Promise {
//…
$bar = yield foo3();
//…
}
function foo3(): Promise {
//…
yield $client->send(…);
//…
}


function foo1(): void {
//…
$loop->wait(foo2());
//…
}
function foo2(): Promise {
//…
$bar = yield foo3();
//…
}
function foo3(): Promise {
//…
yield $client->send(…);
//…
}

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

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

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

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

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

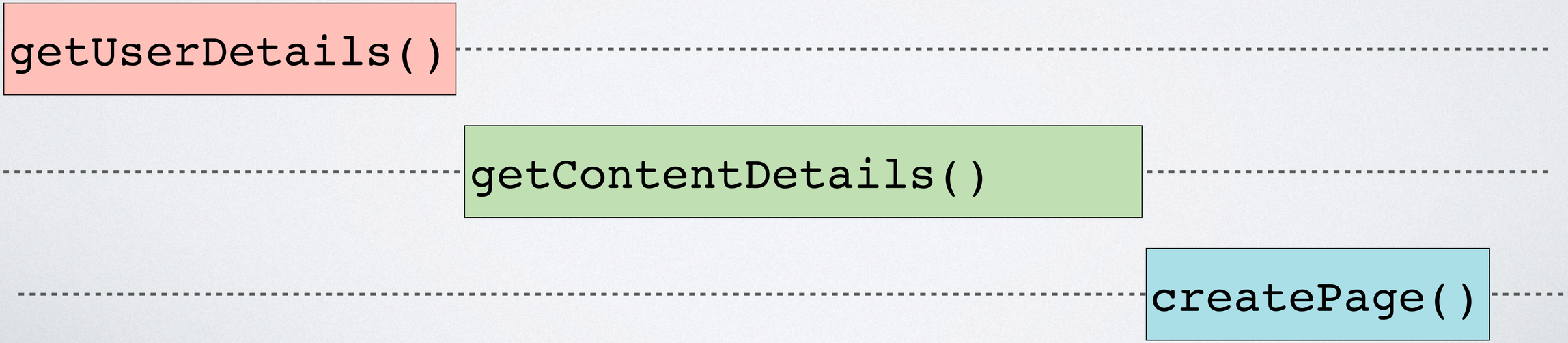
```php
function foo3(): Promise {
    //…
    yield $client->send(…);
    //…
}
```

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

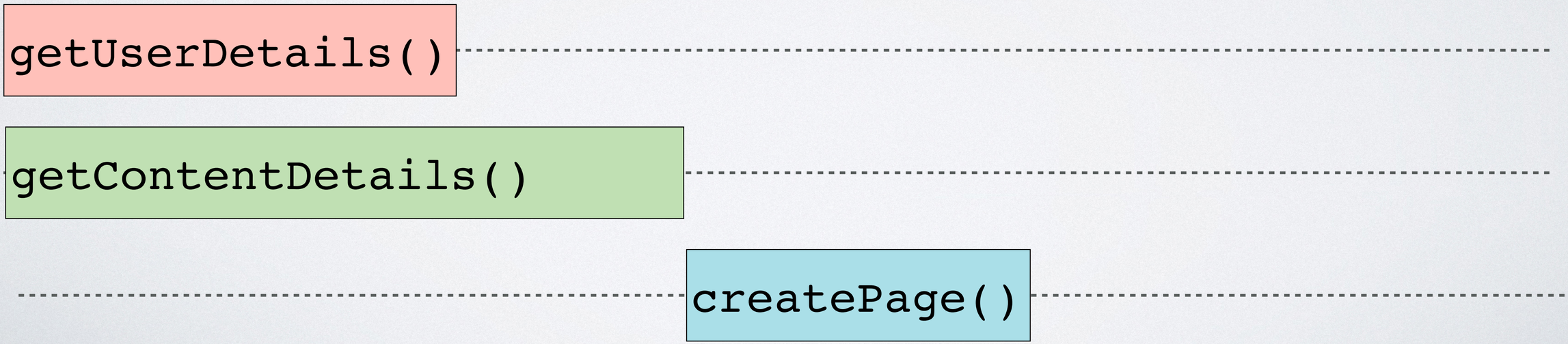
# How to enable concurrency?

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

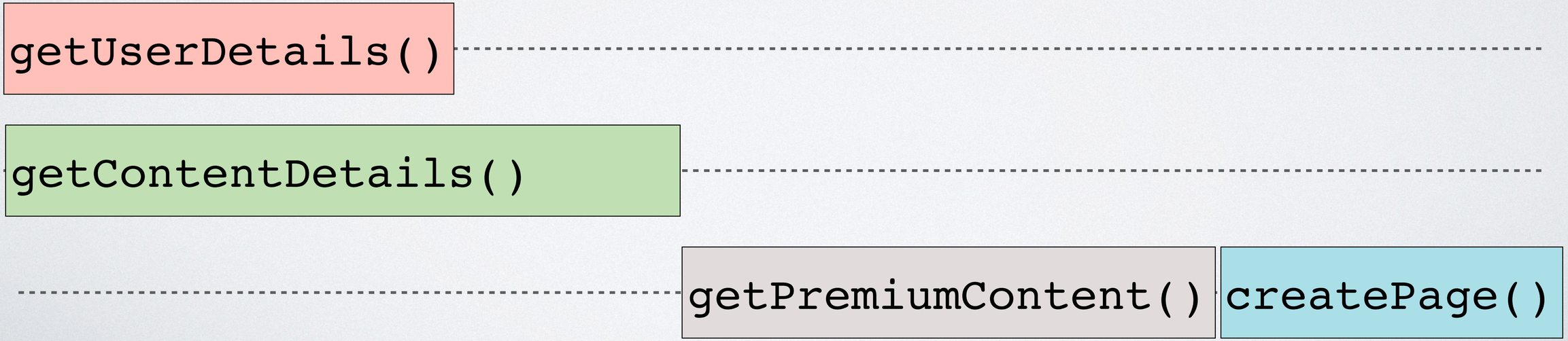
```php
public function foo()
{
    $user = yield getUserDetails();
    $content = yield getContentDetails();

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



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

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

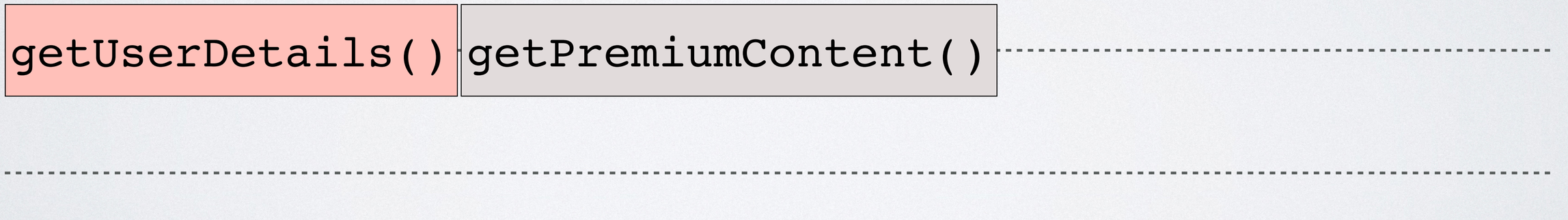


```php
public function foo(): \Generator
{
    [$user, $content] = yield $this->eventLoop->promiseAll(
        getUserDetails(),
        getContentDetails()
    );
    $premium = yield getPremiumContent($user);

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



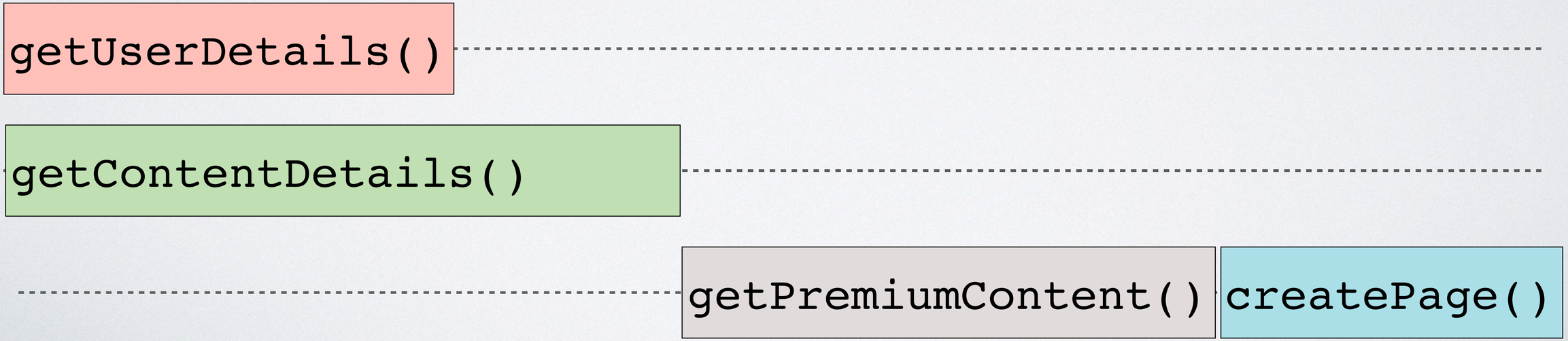
**Consecutive** `yield` instructions means **dependency** between **promises**.

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

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

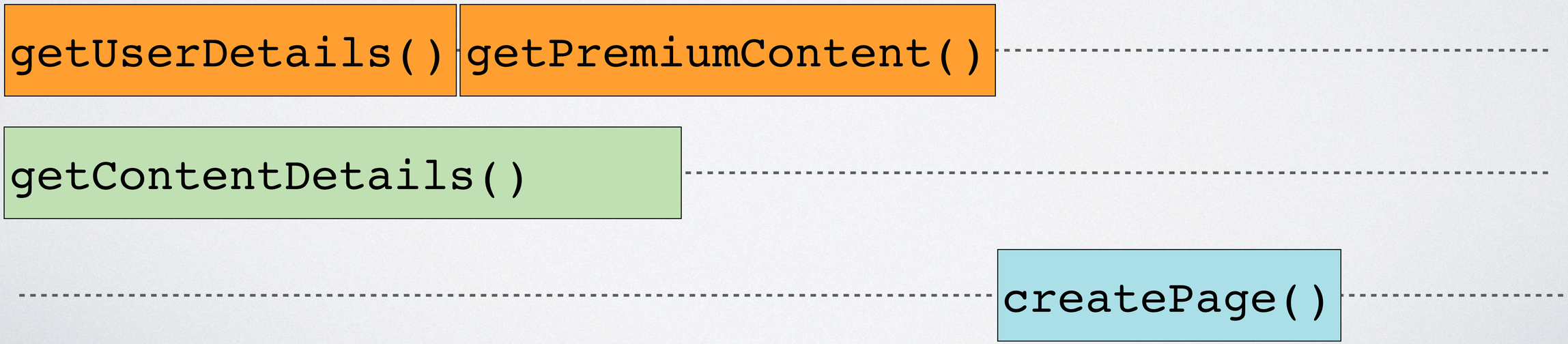


```php
// Intermediate function
private function userAndPremium(): \Generator
{
    $user = yield getUserDetails();
    $premium = yield getPremiumContent($user);

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

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

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



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

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

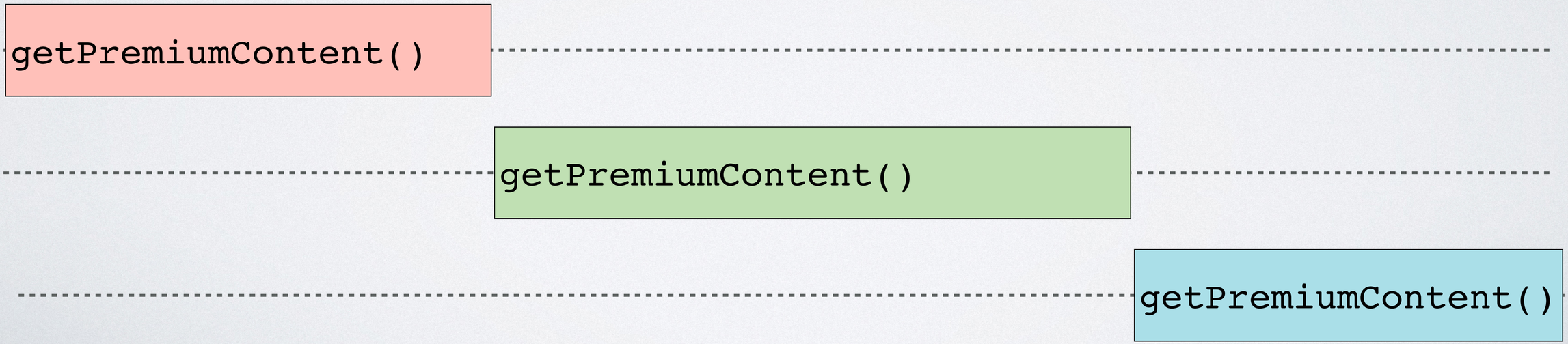


Create an

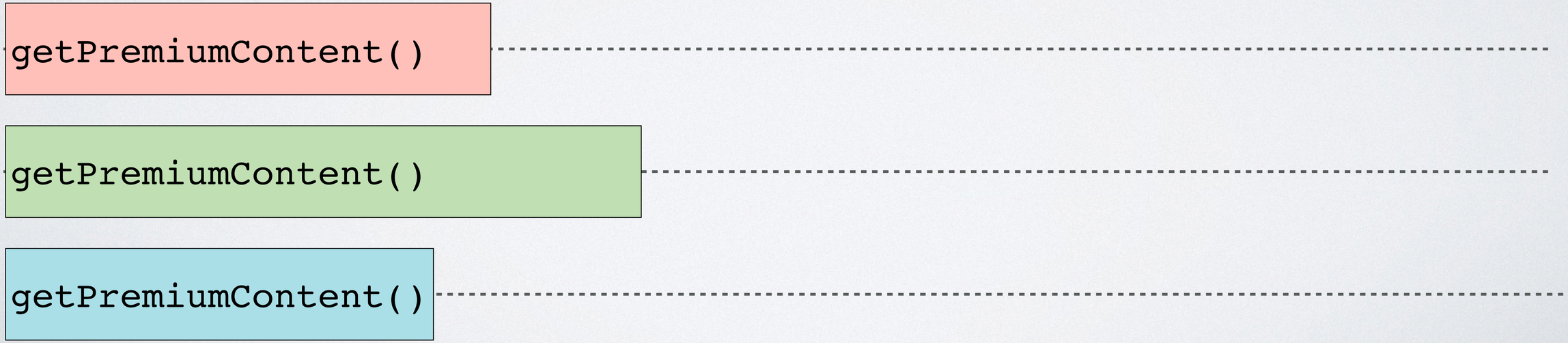
**asynchronous**

function per **goal**.

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

    return $allPremium;
}
```



```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
{
    $promises = [];
    foreach ($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?

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

# Use case: Cache

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

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

*Chapter 3*

# Drawbacks

# Some noise…

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

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

# Some noise…

```php
public function myFunction(): Promise
{
    return $this->eventLoop->async($this->myFunctionAsync());
}

private function myFunctionAsync(): \Generator
{
    // ...
    yield $promise;
    // ...
}
```

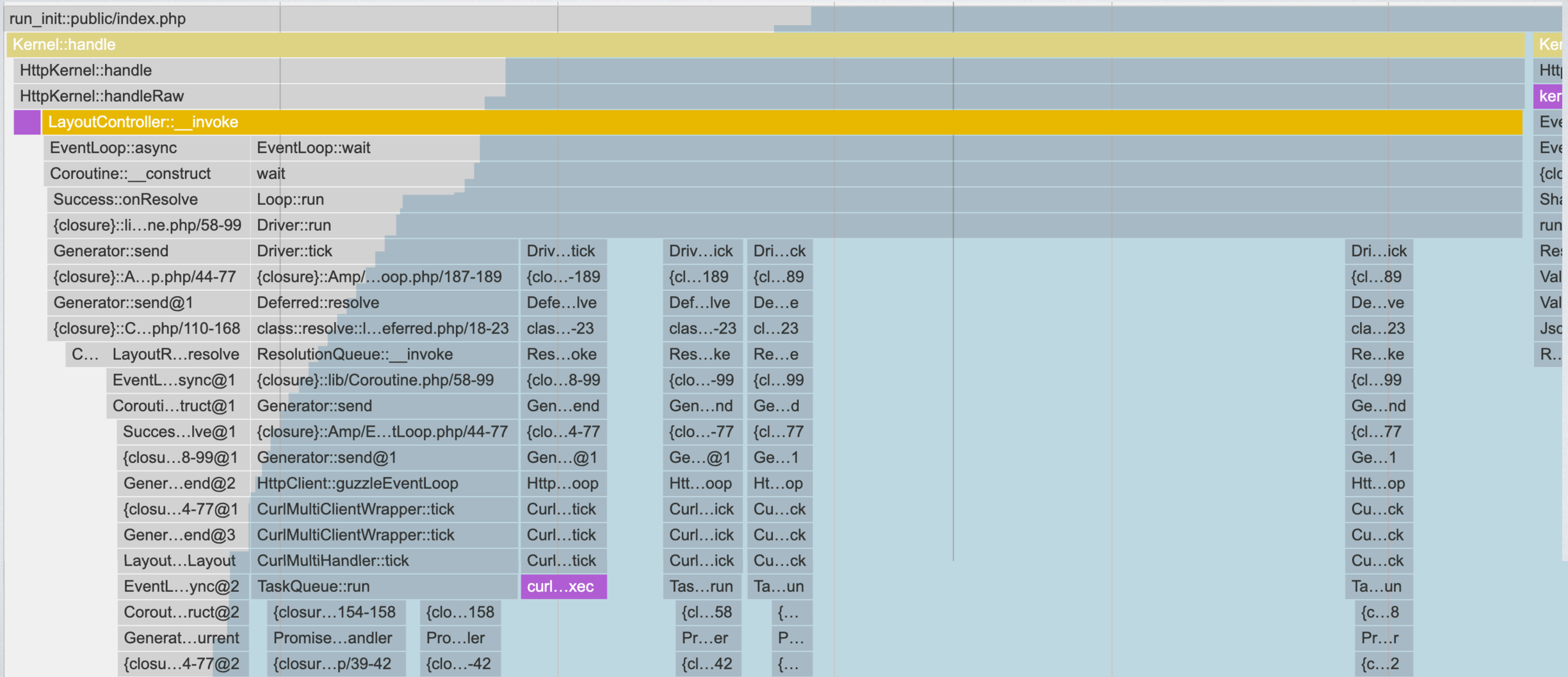
# Promise hides actual type

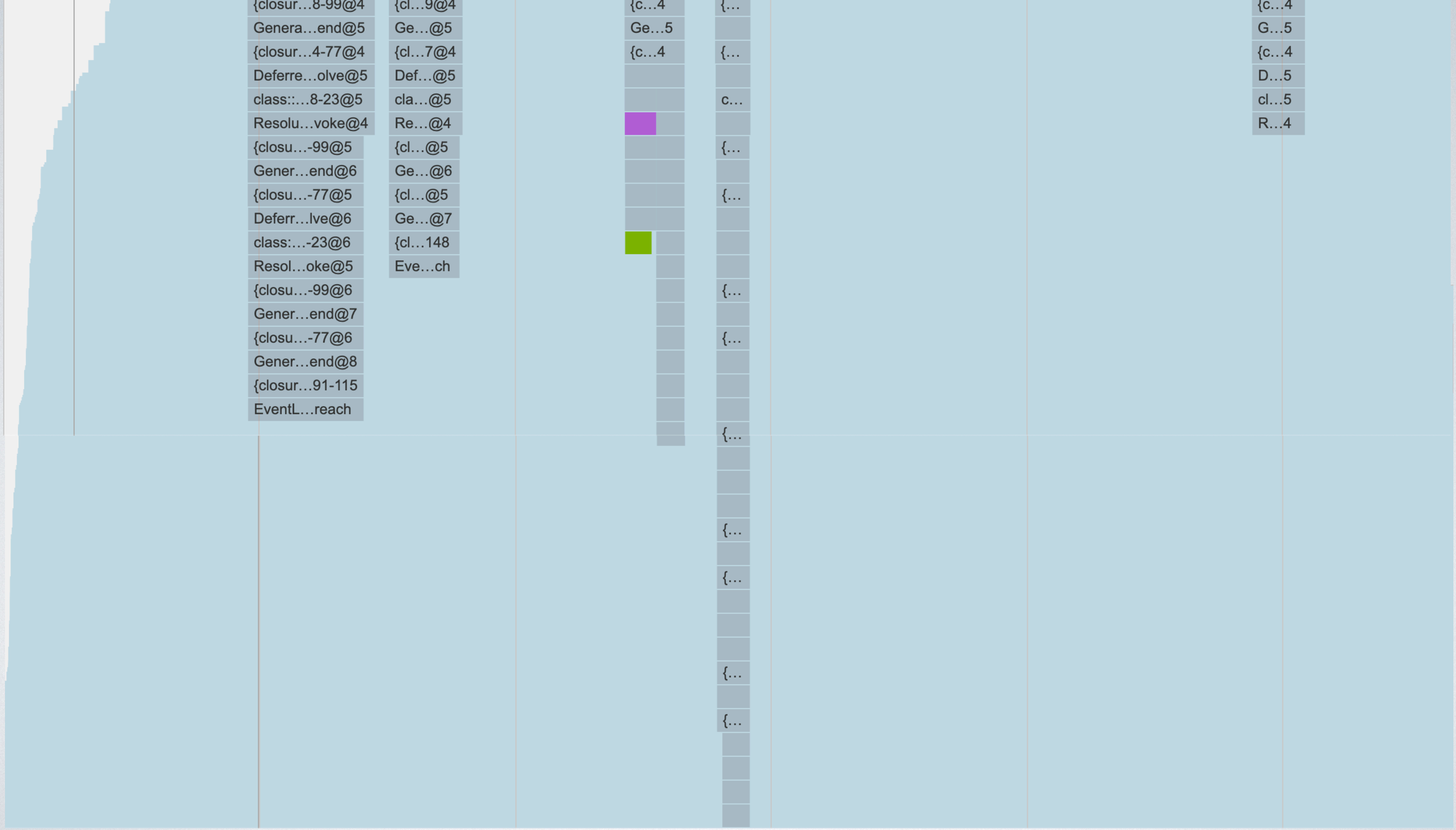
```php
function createRandomNumber(): int;


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


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

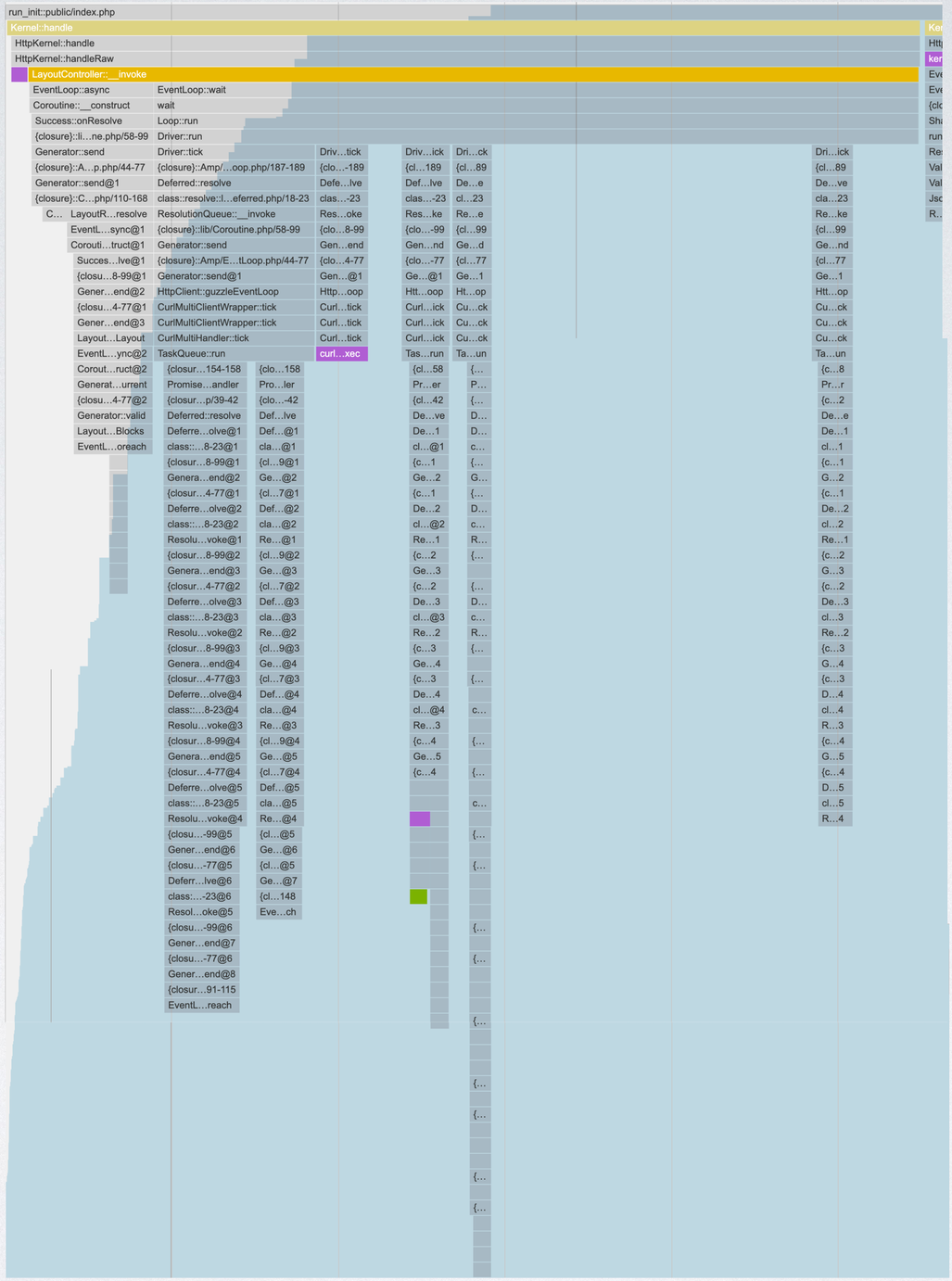
# Stacktrace

```
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

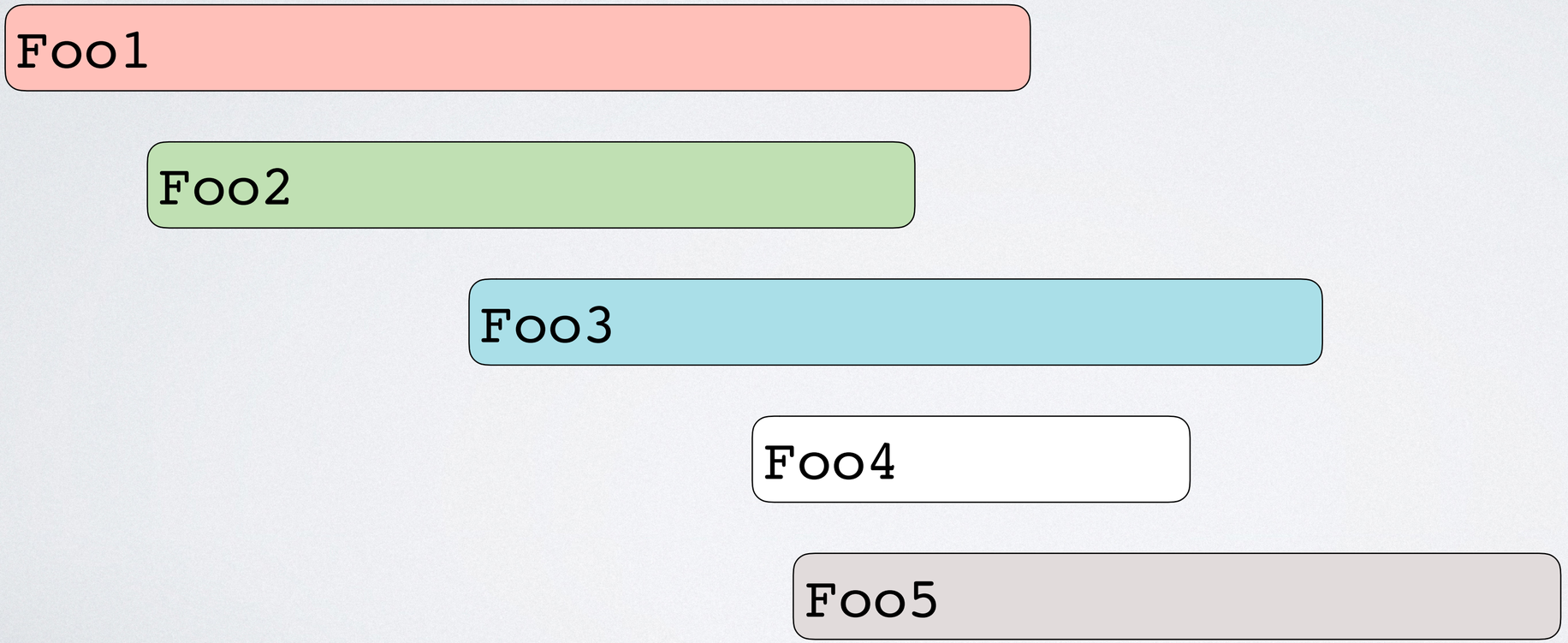

run_init::public/index.php
Kernel::handle
HttpKernel::handle
HttpKernel::handleRaw
LayoutController::__invoke
EventLoop::async
EventLoop::wait
Coroutine::__construct
wait
Success::onResolve
Loop::run
{closure}::li…ne.php/58-99
Driver::run
Generator::send
Driver::tick
{closure}::A…p.php/44-77
{closure}::Amp/…oop.php/187-189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



# An event loop is always running!

⚠️ Takes care of your CPU ⚠️

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

True story…

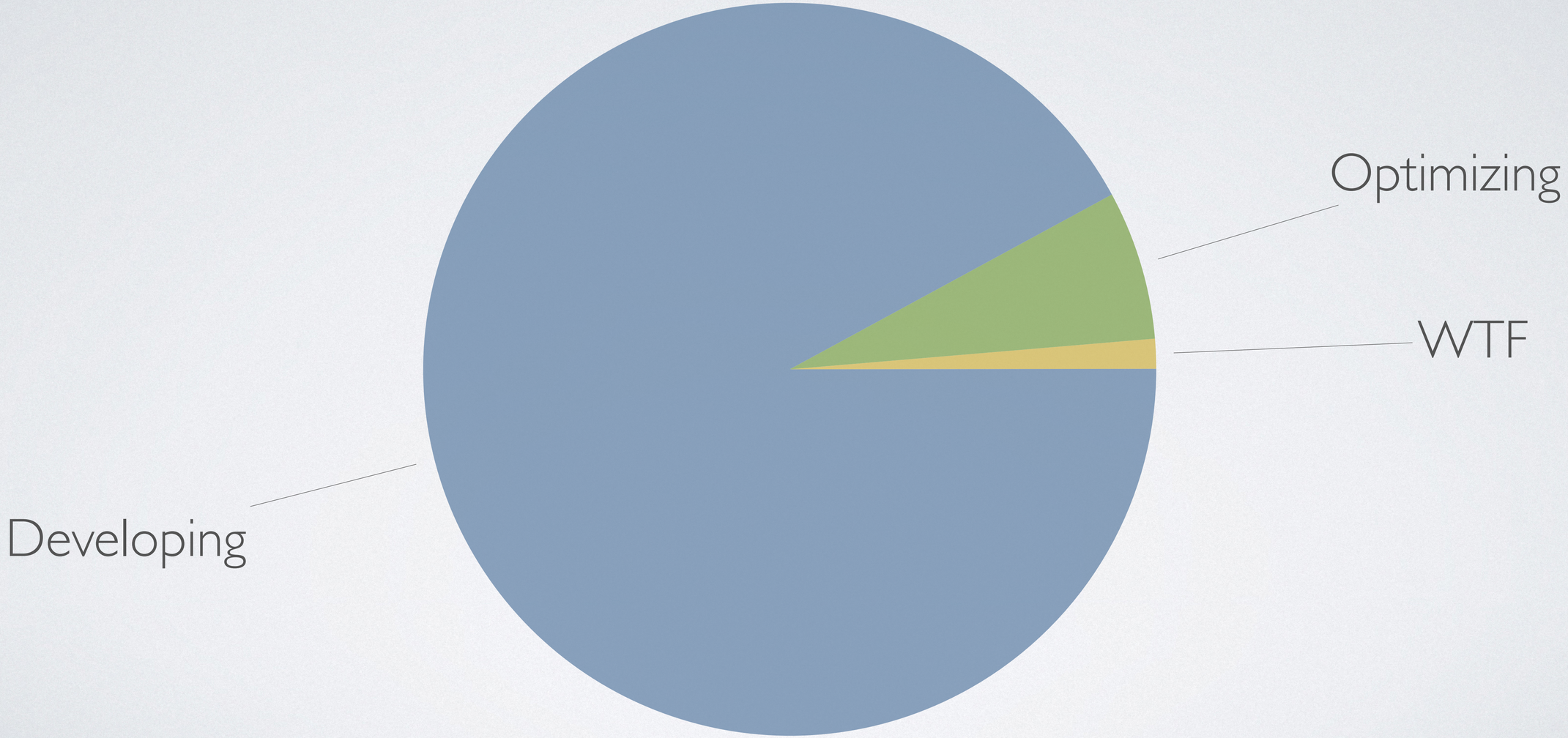
# *Asynchronously ever After...*

## Conclusion

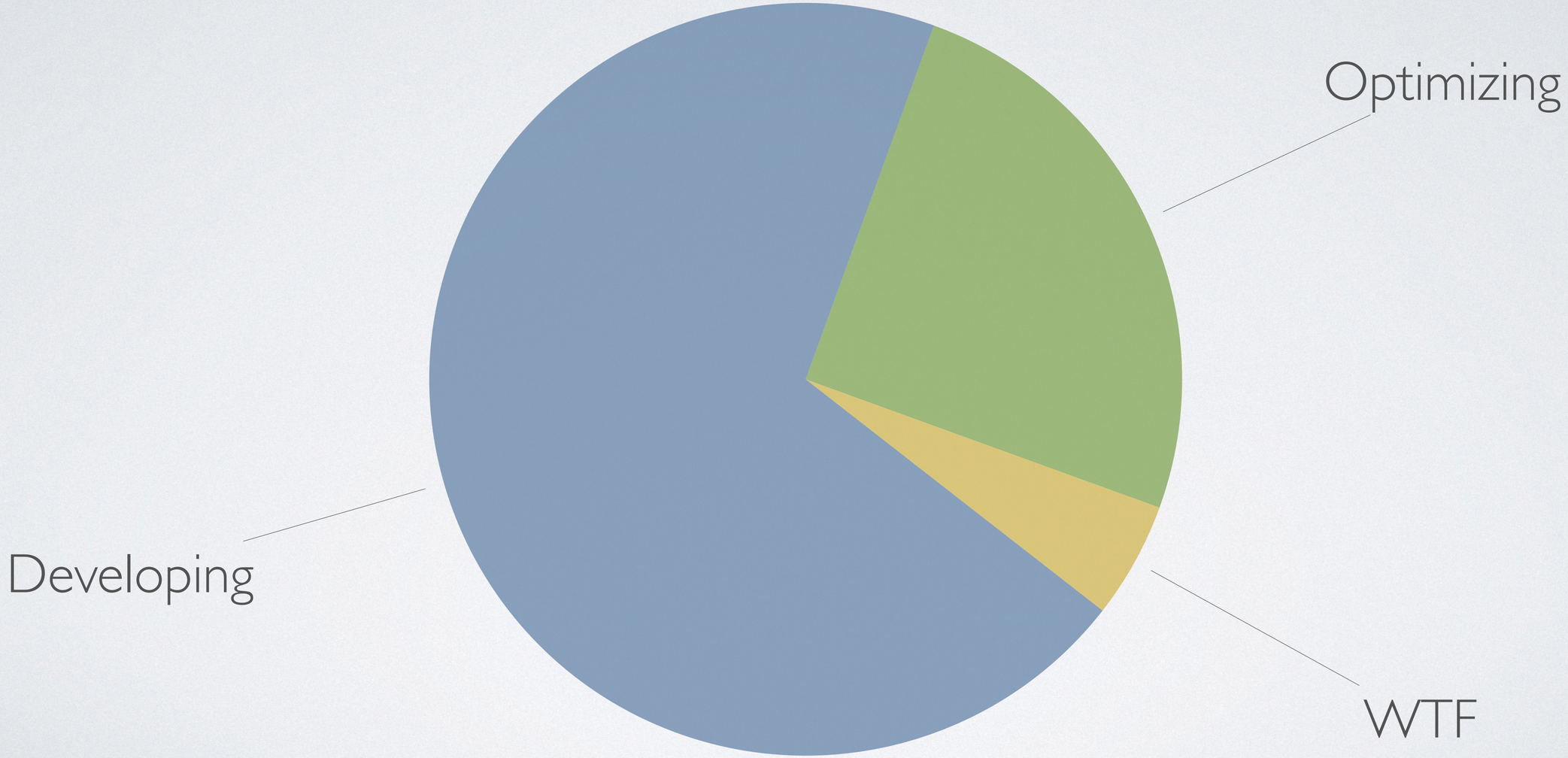
# Training?

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

# Synchronous



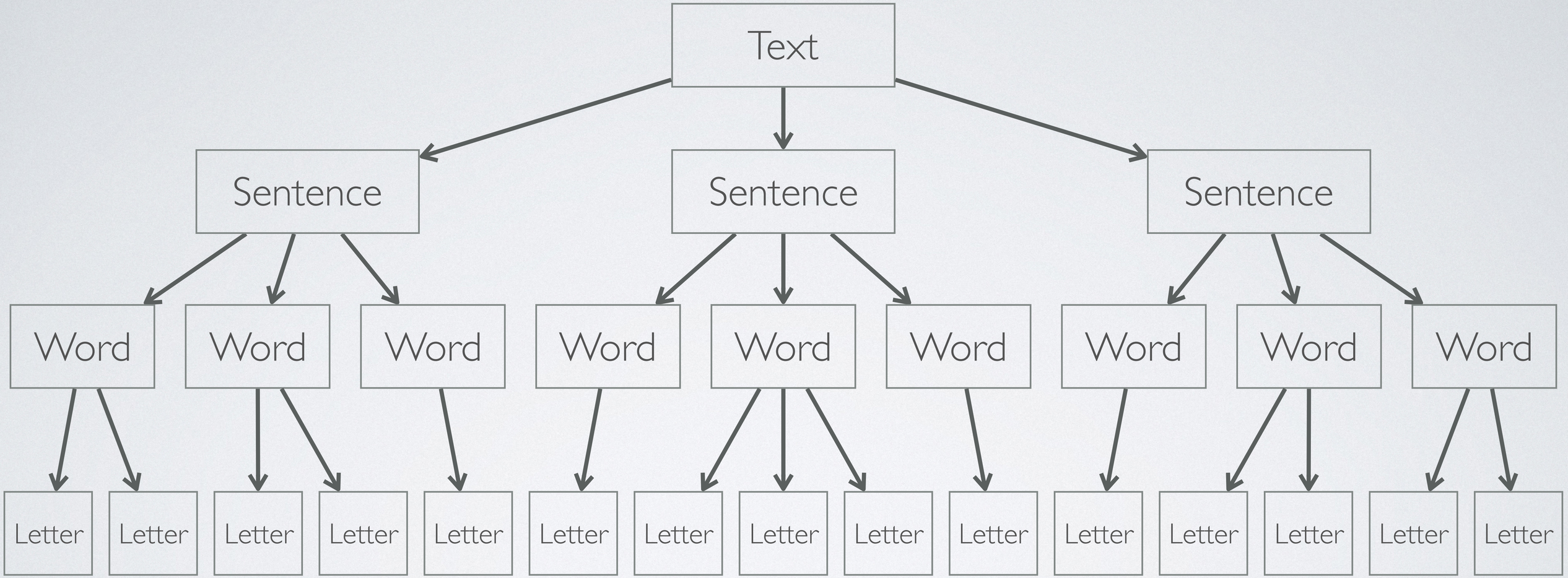
# Asynchronous



# Metrics?

Depends **a lot** on your use case…

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




≈ 8000 requests…

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

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

https://6play.fr

≈ 50 output requests per input request

≈ 200ms per output request

**Synchronous** program: ≈**10s** (*theoretical*)

**Asynchronous** program: ≈**0.5s**

# Technical Decisions

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

# Thanks !

```
$answer = yield $you->ask($me);
```

BEDROCK

Benoit Viguier
@b_viguier

ConFoo.CA
DEVELOPER CONFERENCE