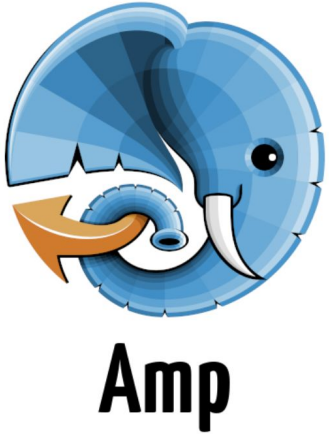
Code **Asynchrone**
dans une application **Symfony Synchrone**

PRISMA MEDIA

Jérôme TAMARELLE
@GromNaN

SymfonyLive
PARIS 2022
7-8 AVRIL



Les systèmes PHP Async

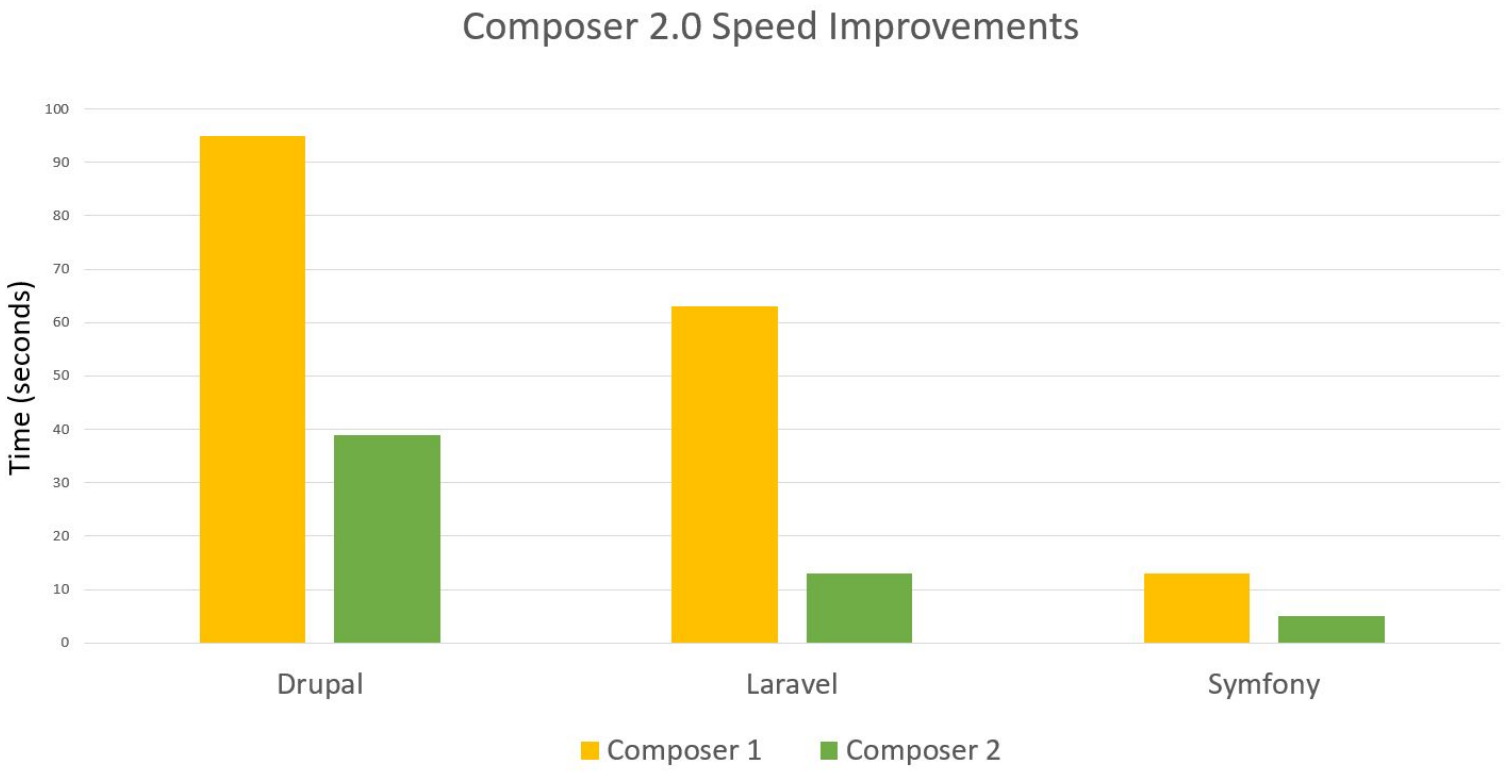
Qui ?

Utilise du PHP Asynchrone régulièrement ?

Qui ?

Utilise Composer ?

Téléchargement et extraction des packages en parallèle



Source: https://blog.packagist.com/composer-2-0-is-now-available/

Qui ?

A déjà écrit du code Asynchrone en PHP ?

PHP Async, qu'est-ce que c'est ?

1. I/O non-bloquants
2. Délégation de traitement à des processus parallèles
3. Branches de code concurrentes dans un même thread

1. I/O Non-bloquants

Capacité à envoyer et recevoir des flux
sans bloquer le processus courant.

SymfonyLive
PARIS 2022
7-8 AVRIL

Fonctions natives PHP pour les flux non-bloquants

`stream_select`

- ReactPHP, AMPHP
- Symfony AmpHttpClient
- Permet d'implémenter *tous* les protocoles

`curl_multi_select`

- ext-curl
- Symfony CurlHttpClient
- Guzzle (Promises)

@GromNaN — Prisma Media

SymfonyLive PARIS 2022 7-8 AVRIL

Symfony HttpClient est non-bloquant …

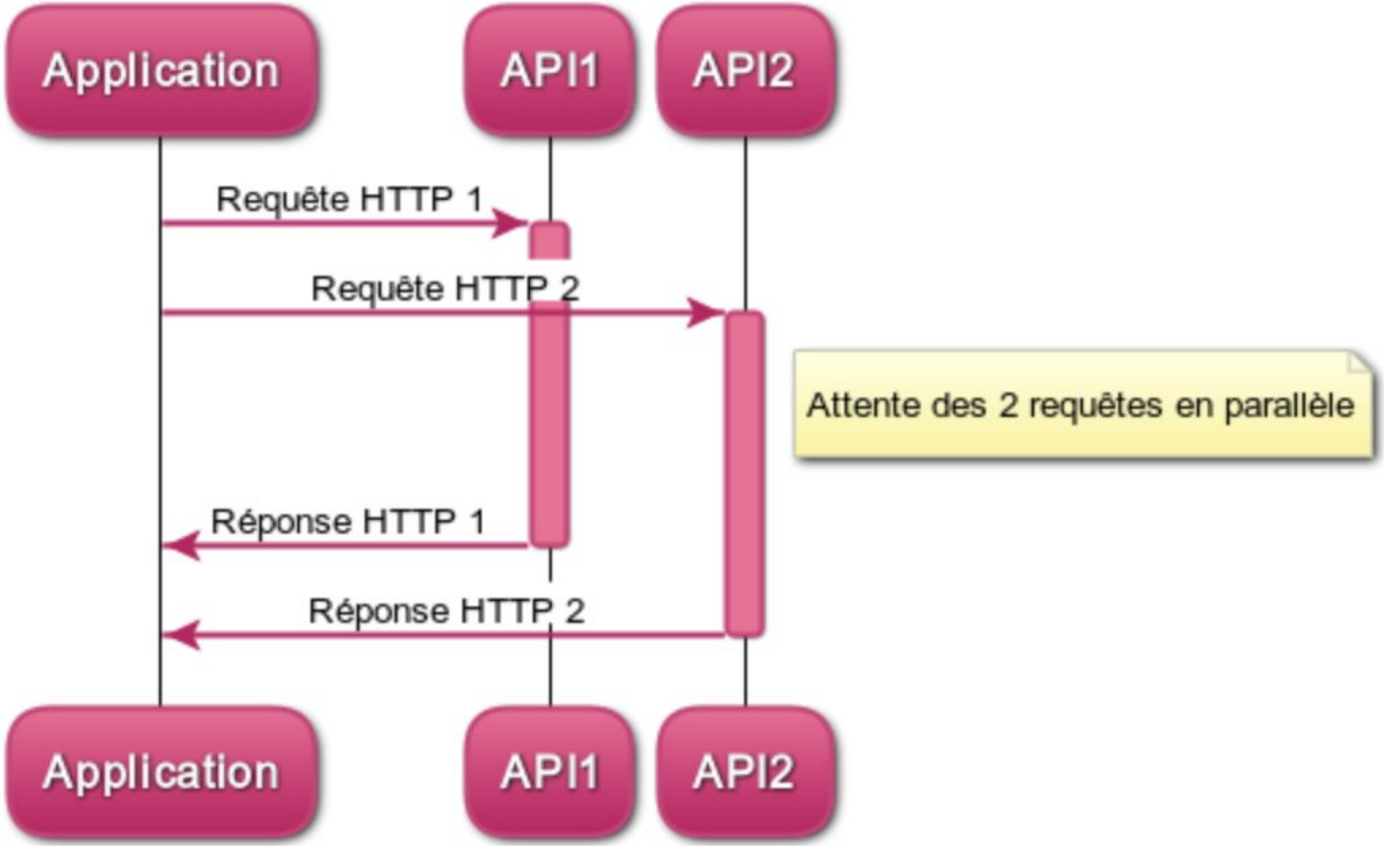
```php
$httpClient = HttpClient::create();

// Requests are non-blocking
$response1 = $httpClient->request('GET', 'https://httpstat.us/200?sleep=500');
$response2 = $httpClient->request('GET', 'https://httpstat.us/200?sleep=200');

// Do anything while responses wait in parallel

// Reading responses is blocking
$response1->getContent();
$response2->getContent();
```

… mais la lecture de la réponse est bloquante

```php
$httpClient = HttpClient::create();

// Requests are non-blocking
$response1 = $httpClient->request('GET', 'https://httpstat.us/200?sleep=500');
$response2 = $httpClient->request('GET', 'https://httpstat.us/200?sleep=200');

// Do anything while responses wait in parallel

// Reading responses is blocking
$response2->getContent();
$response1->getContent();
```

Sauf en lisant les paquets lorsqu'ils arrivent

```php
// Read responses
foreach ($httpClient->stream([$response1, $response2]) as $response => $chunk) {
    if ($chunk->isLast()) {
        $response->getContent();
    }
}
```


Application
API1
API2
Requête HTTP 1
Requête HTTP 2
Attente des 2 requêtes en parallèle
Réponse HTTP 1
Réponse HTTP 2
Application
API1
API2

```php
$client = new React\Http\Browser();
$requestHandler = function ($url) use ($client): PromiseInterface {
    return $client->get($url)
        ->then(function (ResponseInterface $response): string {
            return (string) $response->getBody();
        });
};
React\Promise\all([
    $requestHandler('https://httpstat.us/200?sleep=500'),
    $requestHandler('https://httpstat.us/200?sleep=200'),
])->then(function (array $results) use (&$bodies) { $bodies = $results });

React\EventLoop\Loop::run();
```

Source : https://github.com/reactphp/http/tree/1.x/examples

Avec React : Promises

```php
$client = Amp\Http\Client\HttpClientBuilder::buildDefault();
Loop::run(static function () use ($client, &$bodies): \Generator {
    $requestHandler = static function (string $uri) use ($client): \Generator {
        /** @var Amp\Http\Client\Response $response */
        $response = yield $client->request(new Request($uri));
        return yield $response->getBody()->buffer();
    };

    $bodies = yield [
        Amp\call($requestHandler, 'https://httpstat.us/200?sleep=500'),
        Amp\call($requestHandler, 'https://httpstat.us/200?sleep=200')
    ];
});
```

Source : https://github.com/amphp/http-client/tree/master/examples

Avec AMP : Coroutines

```php
$client = Amp\Http\Client\HttpClientBuilder::buildDefault();
$requestHandler = static function (string $uri) use ($client): string {
    return $client->request(new Request($uri))->getBody()->buffer();
};

$futures = [
    Amp\async(fn () => $requestHandler('https://httpstat.us/200?sleep=500')),
    Amp\async(fn () => $requestHandler('https://httpstat.us/200?sleep=200')),
];

$bodies = Amp\Future\await($futures);
```

Source : https://github.com/amphp/http-client/tree/v5/examples/concurrency

Avec AMP v5 : Fibers (PHP 8.1)

Pareil pour tous les flux d’entrée/sortie

- HTTP Requête entrante / réponse sortante
- HTTP Requête sortante / réponse entrante
- Filesystem
- MySQL
- Postgres
- Redis
- AMQP publication / consommation
- …

2. Processus parallèles

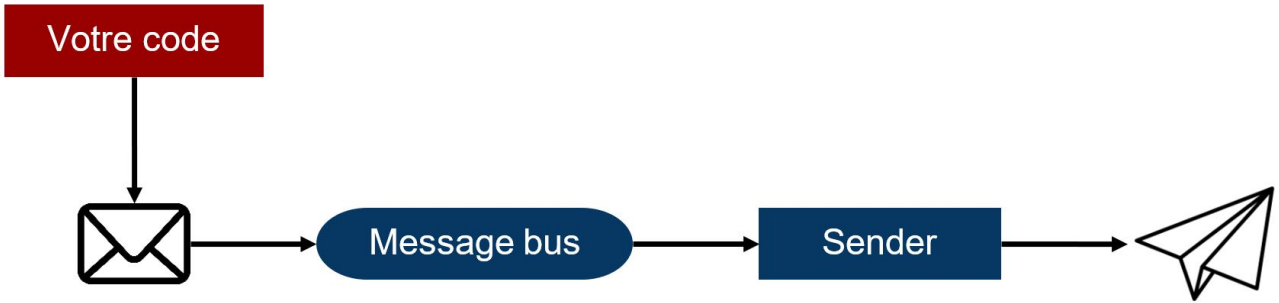
Capacité à effectuer des **traitements en parallèle**,
sans bloquer le processus courant.

SymfonyLive
PARIS 2022
7-8 AVRIL

Symfony Process

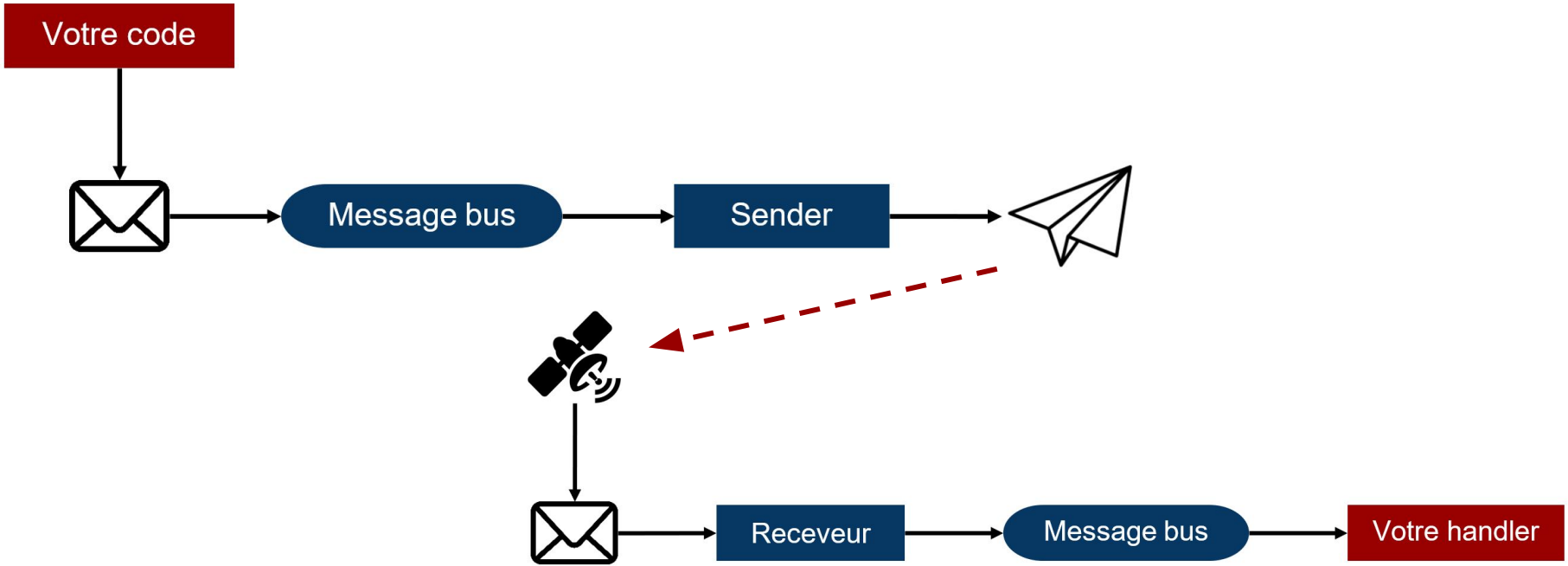

```php
$process1 = new Process(['sleep', '5']);
$process2 = new Process(['sleep', '2']);

$process1->start();
$process2->start();

$process1->wait();
$process2->wait();
```

Symfony Messenger — Synchrone



Source : https://afsy.fr/avent/2019/13-plongee-au-coeur-du-composant-messenger

Symfony Messenger — Asynchrone



Source : https://afsy.fr/avent/2019/13-plongee-au-coeur-du-composant-messenger

Symfony Messenger — Concurrence scalable

Supervisord

```
[program:messenger-consume]
command=/usr/bin/php bin/console messenger:consume async
directory=/var/www/app
process_name=%(program_name)s_%(process_num)02d
autostart=true
autorestart=true
numprocs=10
stopasgroup=true
stopsignal=KILL
```

Kubernetes Helm

```
spec:
   minReplicas: 1
   maxReplicas: 10
   metrics:
     - type: Resource
       resource:
          name: cpu
          targetAverageUtilization: 70
```

Un traitement à faire sur toutes les entités de la base de données ?

Publier des lots d'ids dans des messages

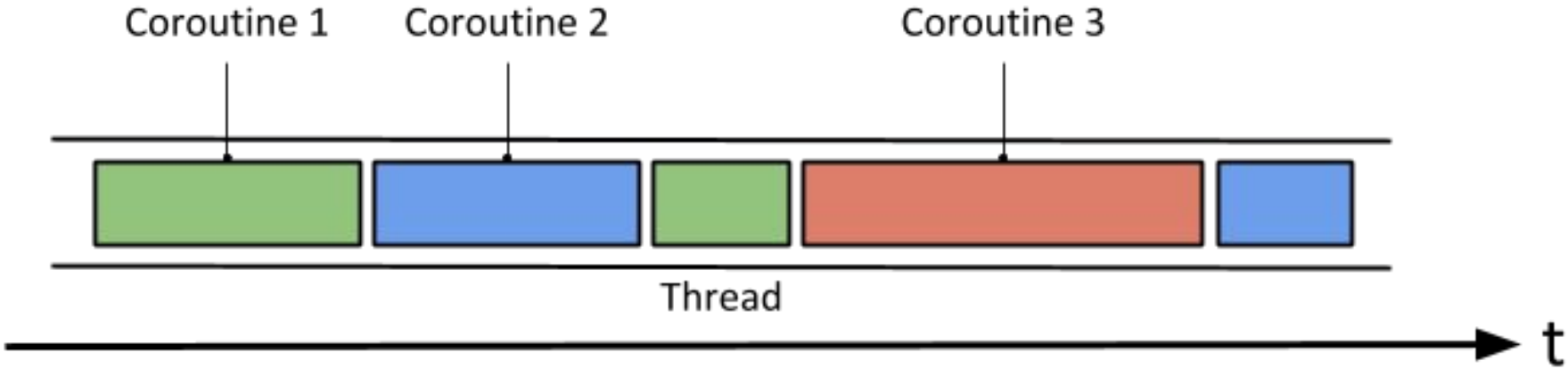
```php
$ids = $this->entityManager
    ->createQueryBuilder()
    ->select('id')
    ->from(Entity::class)
    ->getQuery()
    ->setHydrationMode(Query::HYDRATE_SINGLE_SCALAR)
    ->execute();

foreach (array_chunk($ids, 100) as $chunk) {
    $bus->dispatch(new IndexNotification($chunk));
}
```

3. Promesses, Coroutines et Fibers

Exécution de plusieurs branches de code en **concurrence**,
sur un **même thread**

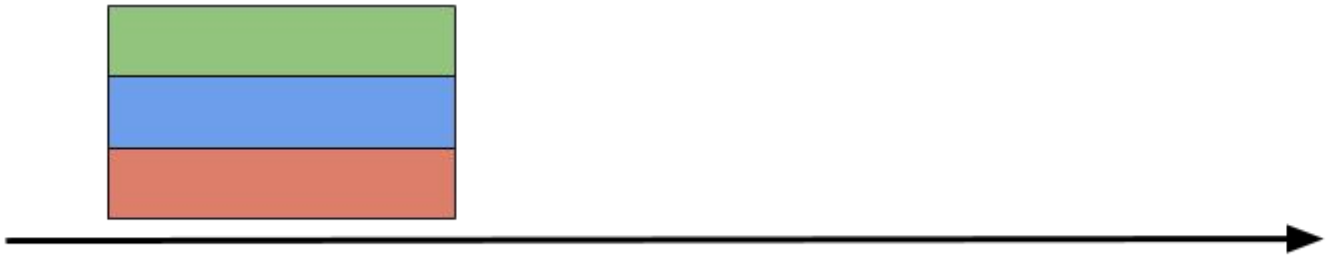
SymfonyLive
PARIS 2022
7-8 AVRIL

@GromNaN — Prisma Media



Source: https://blog.ippon.fr/2021/01/08/des-flux-asynchrones-avec-kotlin-flow/

Modèle synchrone



Modèle avec threads

Modèle avec coroutines

Source: https://blog.ippon.fr/2021/01/08/des-flux-asynchrones-avec-kotlin-flow/

Pré-requis : la boucle

```php
class TaskQueue
{
    private array $queue = [];

    public function add(callable $task)
    {
        $this->queue[] = $task;
    }

    public function run()
    {
        while ($task = array_shift($this->queue)) {
            /** @var callable $task */
            $task();
        }
    }
}
```

Exemples : guzzle/promises TaskQueue ou webonyx/graphql-php SyncPromise

	Promise	Generator	Fiber

PHP	`query()`
	`  ->then(`
	`  callable $success,`
	`  callable $failure`
	`  )`
	`  ->then(`
	`  callable $success,`
	`  callable $failure`
	`  );`
		`call(function (): \Generator {`
		`  $result = yield query();`
		
		`  return yield do($result);`
		`})`
			`async(function() {`
			`  $result = query();`
			
			`  return do($result);`
			`}`

JS	`query()`
	`  ->then(`
	`  callable success,`
	`  callable failure`
	`  )`
	`  ->then(`
	`  callable success,`
	`  callable failure`
	`  )`
		`async function () {`
		`  let result = await query()`
		
		`  return await do(result)`
		`}`
			n/a

Promise ou Generator	Fiber
`all(array $promises): Promise` `→ [$arrayOfValues]`	`await(array $futures): array` `→ [$arrayOfValues]`
`some(array $promises, int $count): Promise` `→ [$arrayOfErrors, $arrayOfValues]`	`awaitAnyN(int $count, array $futures): array` `→ [$arrayOfValues]`
`any(array $promises): Promise` `→ [$arrayOfErrors, $arrayOfValues]`	`awaitAny(array $futures): array` `→ [$arrayOfValues]`
`first(array $promises): Promise` `→ $firstValue`	`awaitFirst(array $futures): mixed` `→ $firstValue`

What color is this function ?

Une fonction synchrone

- peut appeler une fonction synchrone
- **ne peut pas** appeler une fonction **a**synchrone

Une fonction **a**synchrone

- peut appeler une fonction synchrone
- peut appeler une fonction **a**synchrone

https://journal.stuffwithstuff.com/2015/02/01/what-color-is-your-function/

Generator
=
Promise

```php
$callback = function ($url) use ($httpClient) {
    $response = yield $httpClient->request('GET', $url);
    return $response->toArray();
}
$responses = map($callback, $urls);

function map(): Promise {
    // React or Amp implementation
}
```

Fiber
=
Synchrone

```php
$callback = async(function ($url) use ($httpClient) {
    return $httpClient->request('GET', $url)->toArray();
}
$responses = array_map($callback, $urls);
```

Attention avec le scope global

Twig < 6.2 utilise le buffer de sortie unique (`echo`)
N'utilisez pas de Fibers dans vos extensions.

Il peut être utilisé pour rendre un template sans interruption.

(ReactJS 18 vient seulement de supporter ça !)

Mon cas d'usage

PRISMA MEDIA

SymfonyLive PARIS 2022 7-8 AVRIL

Programme-TV.net est un site de presse de divertissement grand public. Il est composé de pages riches en contenus variés. C'est l'exemple que je vais utiliser pour cette présentation.

- **125M** visites par mois
- **10** développeurs
- **2** déploiements en production par jour
- **400** routes Symfony
- **1430** services
- **1500** templates Twig
- **80k** lignes de code PHP (**+ 65k** pour les tests)
- **1** base de données Elasticsearch
- **60 ms** temps de réponse moyen
- **3** refontes graphiques depuis 2016

Design System

PRISMA MEDIA

SymfonyLive PARIS 2022 7-8 AVRIL

Bibliothèque de composants réutilisables.

La mise en place de cet outil répond à plusieurs objectifs :

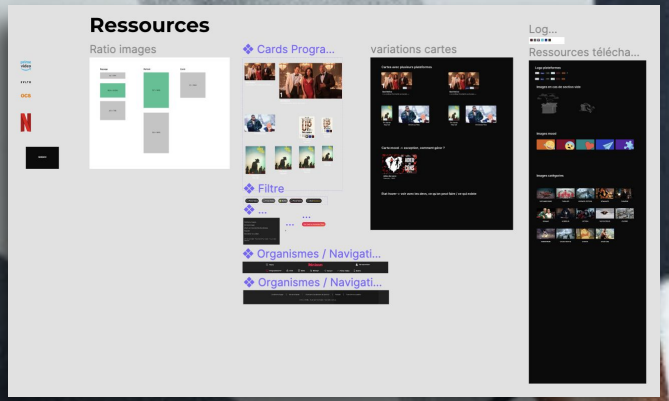
- Vocabulaire commun entre designers et développeurs
- Expérience de navigation cohérente pour les utilisateurs
- Accélérer les temps de développement
- Centraliser les éléments graphiques et les guides de style utiles à la communication du Produit.

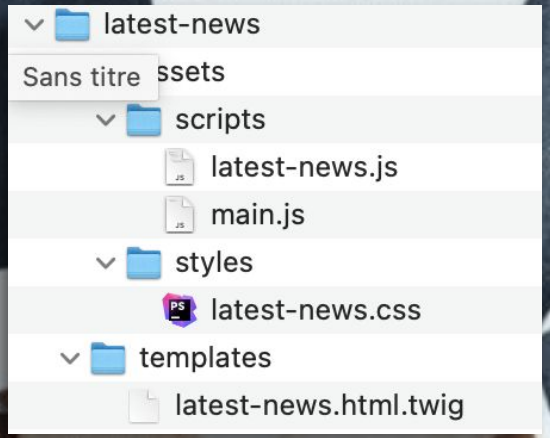
⇒ Faciliter le travail des équipes et de réduire la dette design et technique.
⇒ Ecosystème cohérent pour une meilleure expérience pour les utilisateurs.

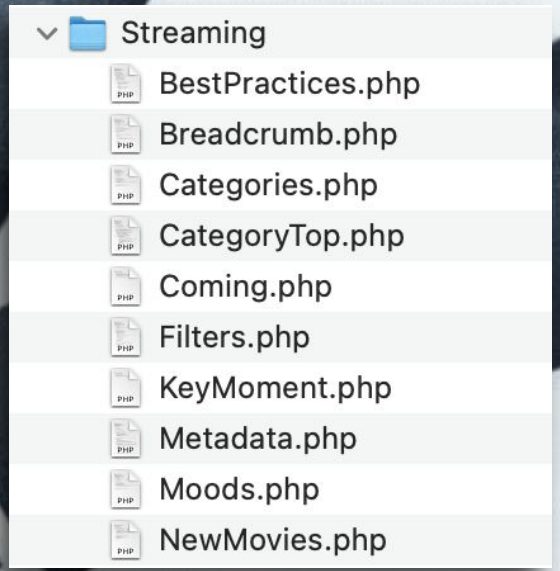
Design

Frontend

Backend







Composition d’une page

@GromNaN — Prisma Media

Netflix : cette hilarante série de la créatrice de Friends touche bientôt à sa fin

Le 23/03/2022 à 18:30 par Adélaïde Fournier

Voir toutes les photos de Jane Fonda

…nda et Lily Tomlin seront de retour le 29 avril sur Netflix pour reprendre une dernière fois leurs rôles de divorcées dans *Grace et Frankie* et boucler la saison 7, dernière salve de cette série diffusée sur la plateforme de streaming depuis 2015.

Ecouter cet article Netflix : cette hilarante série 00:00

Cités dans cet article

Jane Fonda

+ Suivre

Lily Tomlin

+ Suivre

La série américaine ***Grace et Frankie***, disponible sur Netflix depuis 2015 et portée par **Jane Fonda** et **Lily Tomlin**, touche à sa fin. **La saison 7 sera bel et bien la dernière**. Si la plateforme de streaming avait mis en ligne avec un peu d'avance **les quatre premiers épisodes de cette ultime saison** en août dernier, c'est bientôt le moment de découvrir les douze autres ! En effet, créée par Marta Kauffman (**l'une des créatrices de *Friends***) et Howard J. Morris, la série diffusée depuis le 8 mai 2015, a été renouvelée le 4 septembre 2019, pour une septième et dernière saison de seize épisodes. Interrompu à cause du Covid, le tournage a repris le 21 juin 2021, mais dès le 13 août 2021, Netflix avait fait une belle surprise aux fans, en sortant les quatre premiers épisodes de la dernière saison, *Les Colocs, Le Tribunal, Le Lapin* et *La Circoncision*. Les douze suivants seront donc diffusés **le 29 avril prochain.**

L'article parle de...

Grace And Frankie

Série humoristique • Série TV

Saison 1 - Épisode 1

Visionner sur Netflix >>

#Grace et Frankie #Netflix #SVOD

Soyez le premier à réagir !

Les internautes n'ont pas encore fait de commentaire sur cette actualité.

Écrivez votre avis

Publier

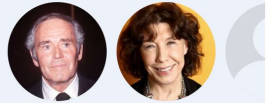
Ça va vous intéresser

Si vous aimez Vikings, cette série bientôt de retour sur Netflix est faite pour vous >>

Cette série Netflix à rattraper d'urgence ce week-end dont la fin va vous scotcher >>

Voir toute l'actu

Autour de Jane Fonda



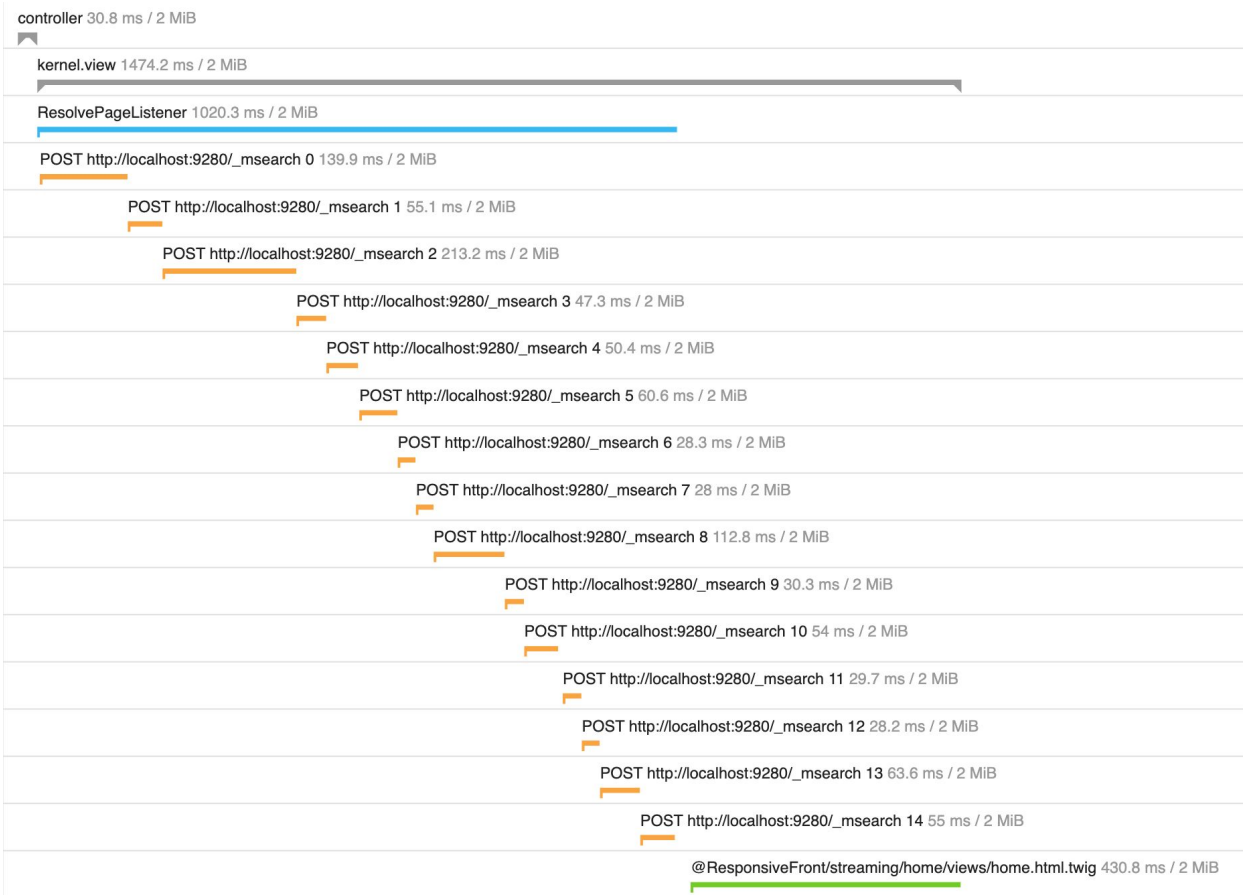
Joe Cocker + Suivre

Henry Fonda + Suivre

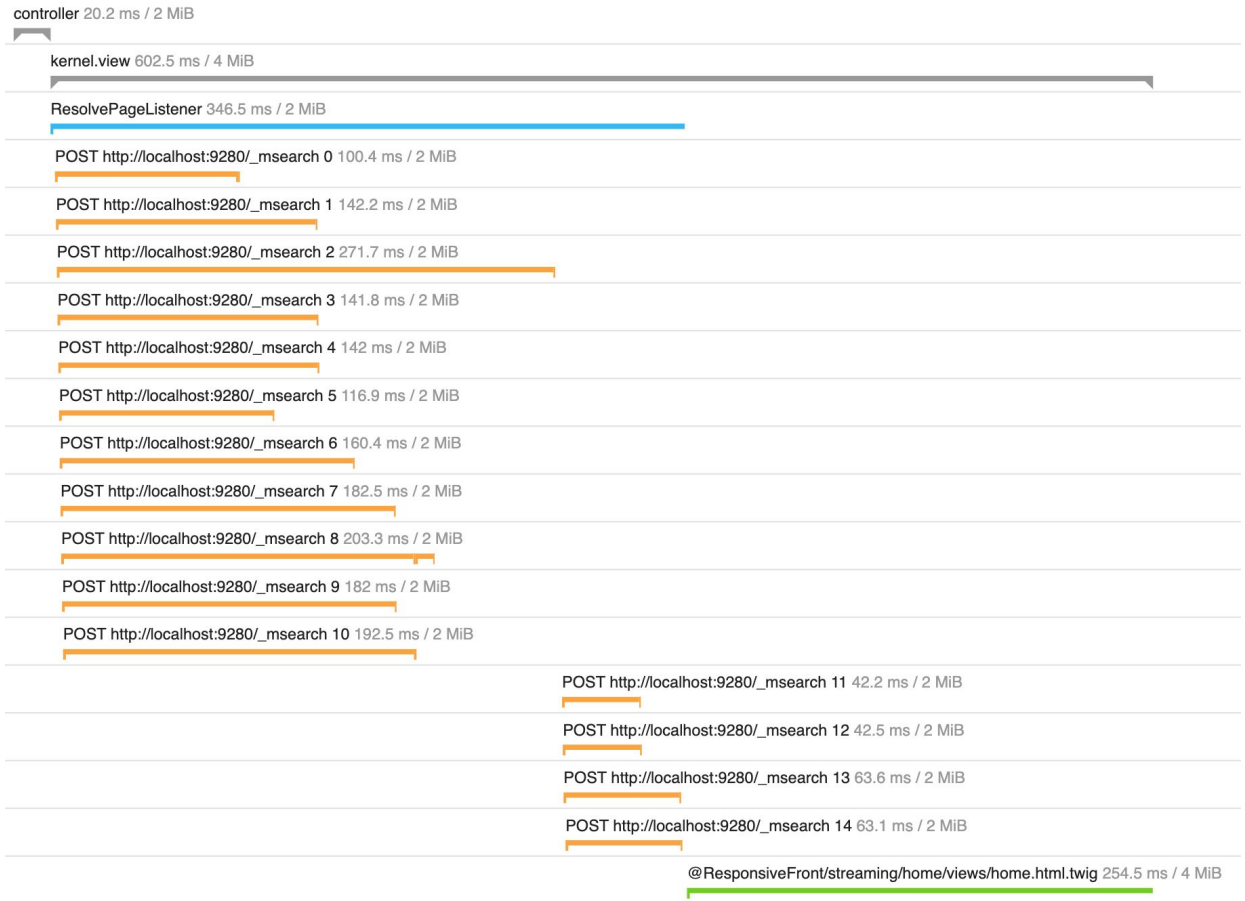
Lily Tomlin + Suivre

```php
public function home(Request $request, array $platforms): Page
{
    $route = $request->attributes->get('_route');

    return $this->pageFactory->create()
        ->add(HeaderNavbar::class, ['active' => HeaderNavbar::ITEM_STREAMING])
        ->add(Streaming\Metadata::class, ['route' => $route, 'platforms' => $platforms])
        ->add(Streaming\Breadcrumb::class, ['isHome' => true, 'platforms' => $platforms])
        ->add(Streaming\Title::class, ['route' => $route, 'platforms' => $platforms])
        ->add(Streaming\Filters::class, ['route' => $route, 'platforms' => $platforms])
        ->add(Streaming\Trending::class, ['platforms' => $platforms])
        ->add(Streaming\Moods::class, ['platforms' => $platforms])
        ->add(Streaming\NewMovies::class, ['platforms' => $platforms, 'size' => 6])
        ->add(Streaming\NewSeries::class, ['platforms' => $platforms, 'size' => 6])
        ->add(Streaming\KeyMoment::class, ['platforms' => $platforms, 'size' => 10])
        ->add(Streaming\Coming::class, ['platforms' => $platforms])
        ->add(Streaming\UserFavorite::class, ['platforms' => $platforms])
        ->add(Streaming\Categories::class, ['platforms' => $platforms])
        ->add(Streaming\News::class, ['platforms' => $platforms])
        ->add(Streaming\BestPractices::class, ['platforms' => $platforms, 'size' => 6])
        ->add(Streaming\PreFooter::class, ['platforms' => $platforms]);
}
```

Chaque composant PHP génère 1 objet pour la vue



Synchrone & requêtes HTTP bloquantes



Asynchrone & Requêtes HTTP en parallèle

Demo

https://github.com/GromNaN/async-page-builder-demo

Conclusions

1. Plusieurs concepts dans l'asynchrone :
   I/O non-bloquants, parallélisation, coroutines
2. Chaque concept est utilisable indépendamment,
   dans un contexte synchrone
3. Les *fibers* permettent de démocratiser les coroutines
   à partir de PHP 8.1, pour structurer le code par domaine

```php
class FiberHttpClient implements Symfony\Contracts\HttpClient\HttpClientInterface
{
    public function __construct(private HttpClientInterface $httpClient) {}

    public function request(string $method, string $url, array $options = []): ResponseInterface
    {
        $response = $this->httpClient->request($method, $url, $options);

        \Amp\delay(0);

        return $response;
    }

    // stream and withOptions are proxies
}
```

Avec gestion d'ordre d'arrivée : https://gist.github.com/GromNaN/a9014a9c69c9a5cc08ac8c30677408f4

Prototype **très incomplet** d'implémentation de Symfony HttpClient avec Fiber