Et si le futur de la programmation concurrentielle avait déjà 50 ans ?

→Baptiste Langlade

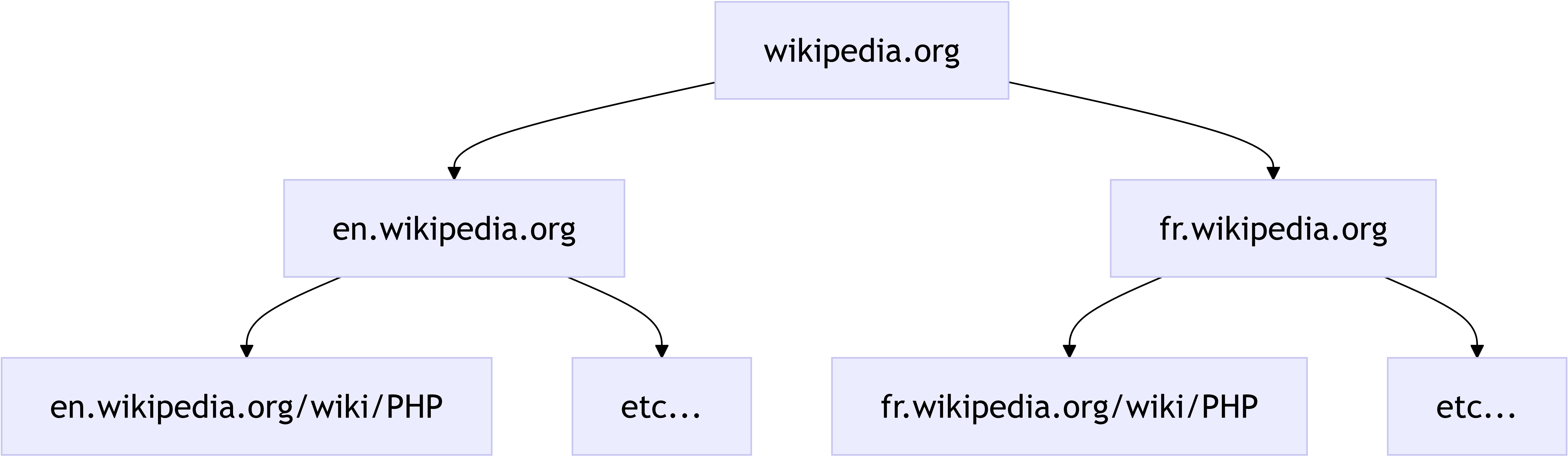
→Architecte chez Efalia

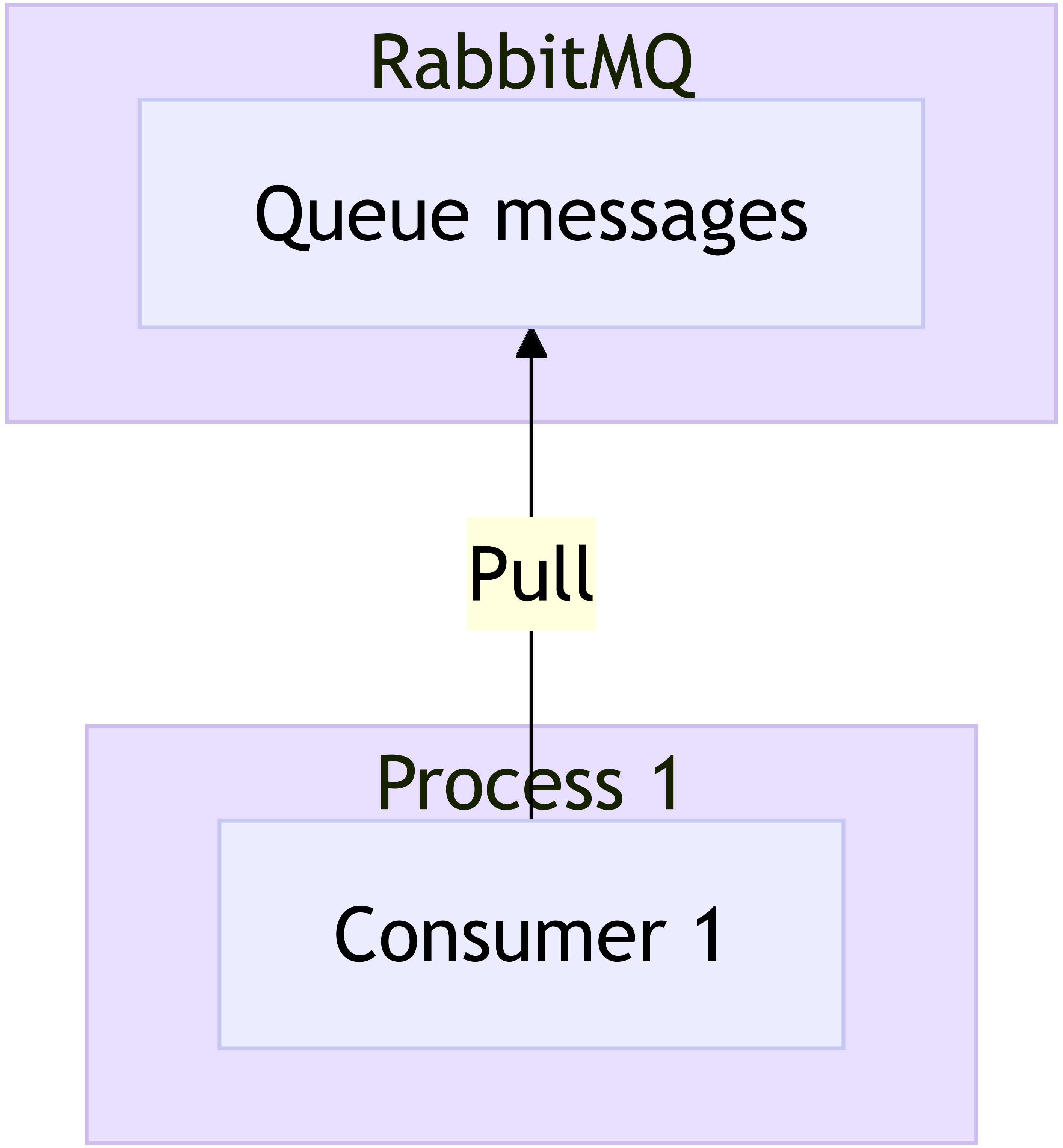
→Lyon

→10+ ans XP

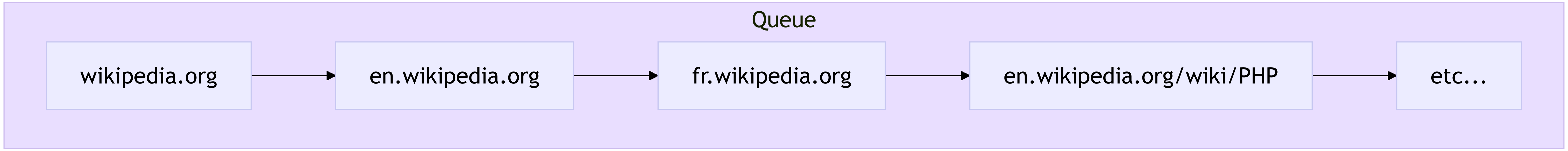
→~100 packages Open Source

Crawler


wikipedia.org
en.wikipedia.org
fr.wikipedia.org
en.wikipedia.org/wiki/PHP
etc...
fr.wikipedia.org/wiki/PHP
etc...


RabbitMQ
Queue messages
Pull
Process 1
Consumer 1

```
$rabbitmq
    ->with(Consume::of('queue')->handle(
        static function(Message $message) use ($rabbitmq) {
            $url = decodeUrl($message);
            $urls = crawl($url);
            $rabbitmq
                ->with(Publish::many($urls)->to('queue'))
                ->run();
        },
    ))
    ->run();
```



Queue

wikipedia.org

en.wikipedia.org

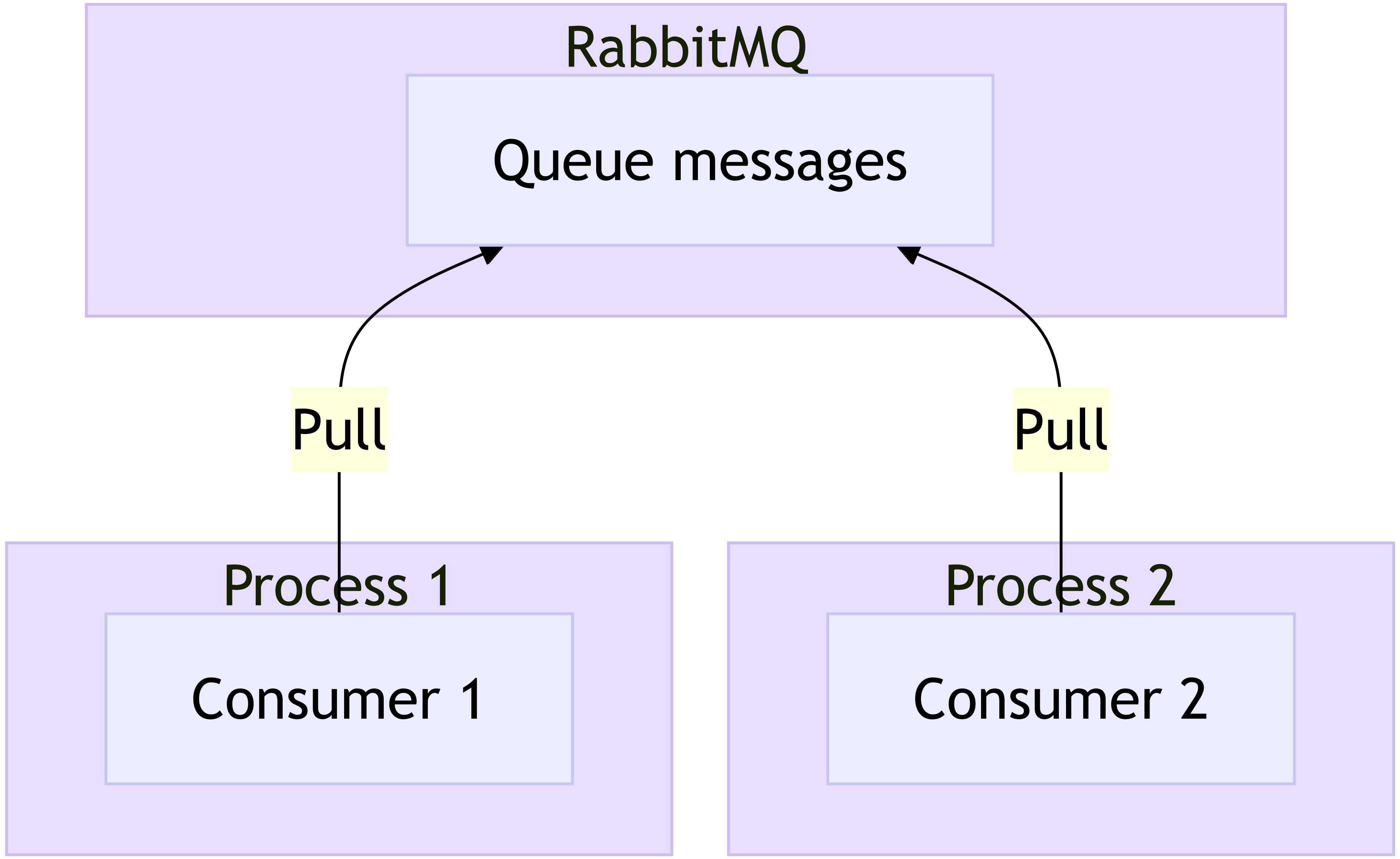
fr.wikipedia.org

en.wikipedia.org/wiki/PHP

etc...

Simple mais inefficace

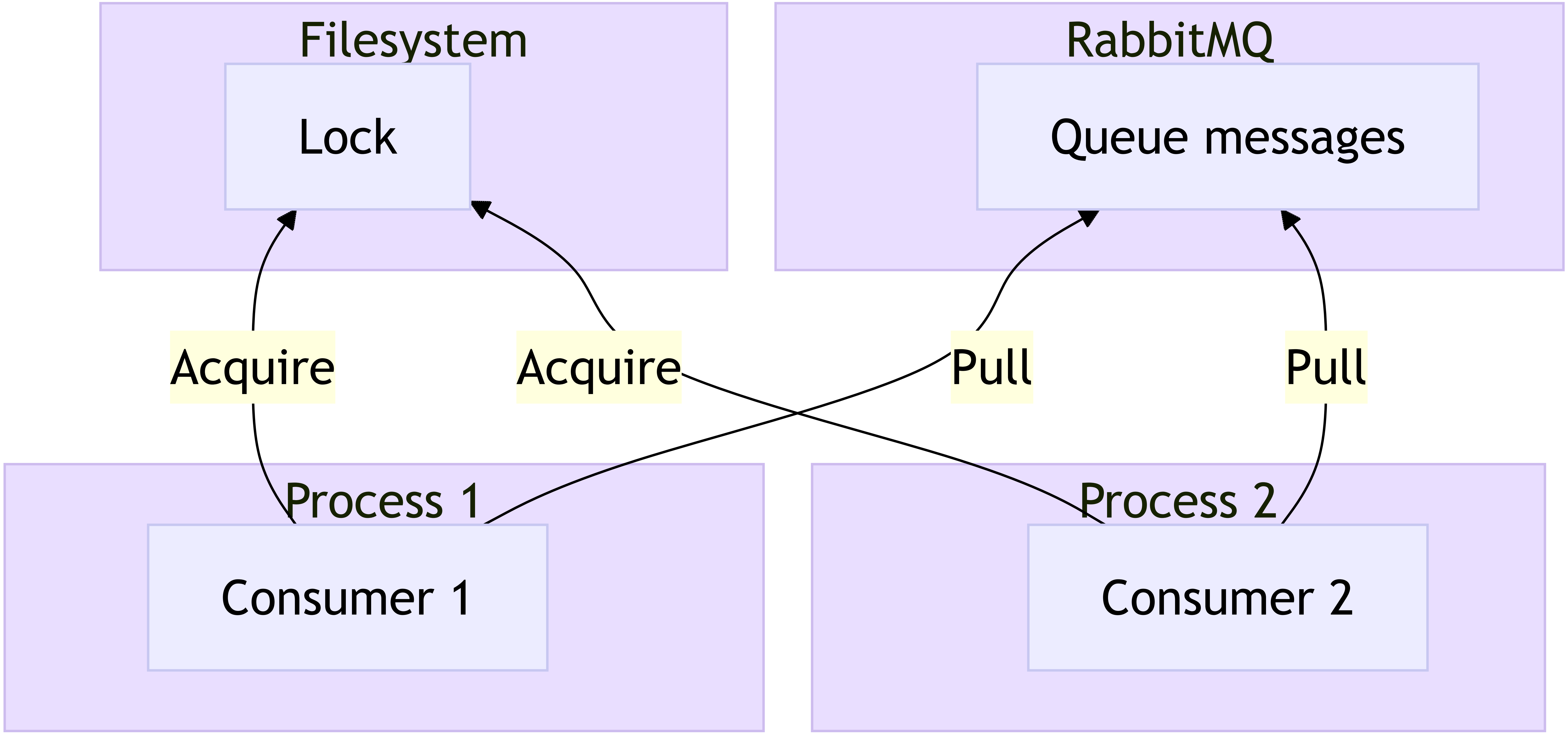
Parallélisation



```
php consumer.php & php consumer.php &
```

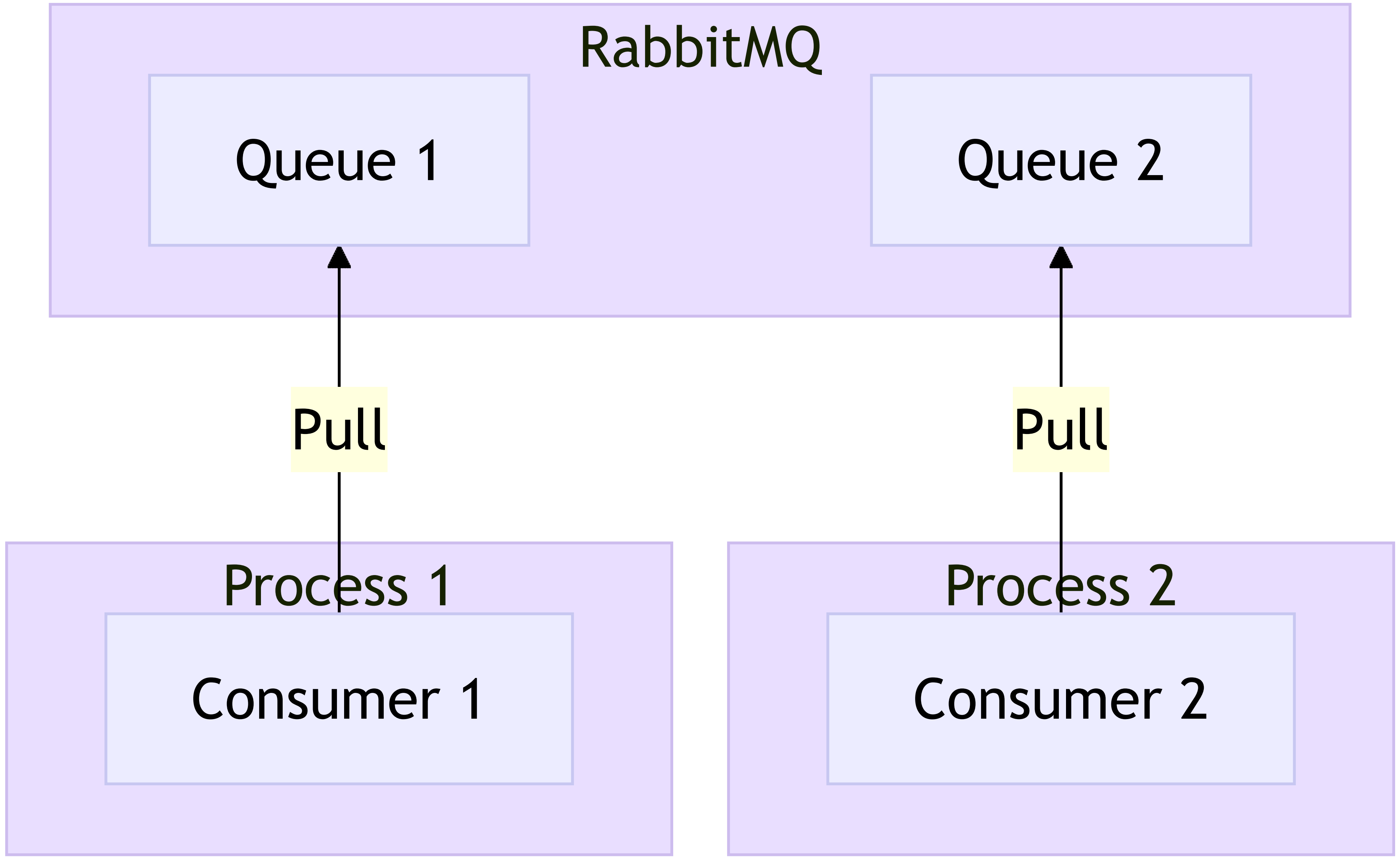
robots.txt

```
User-agent : Googlebot
Allow : /foo
Disallow : /bar
Crawl-delay : 10
```

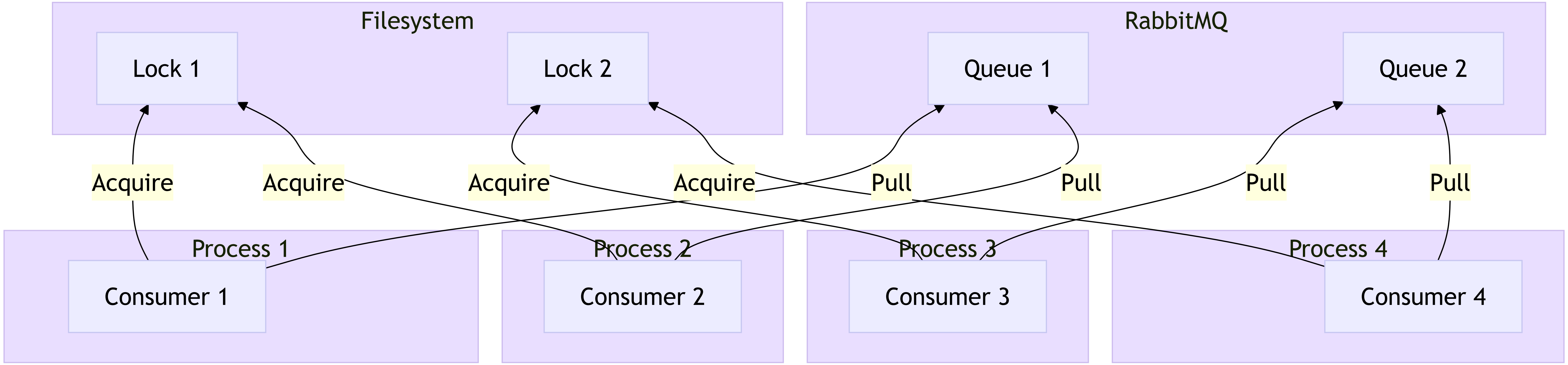

Filesystem
Lock
RabbitMQ
Queue messages
Acquire
Acquire
Pull
Pull
Process 1
Consumer 1
Process 2
Consumer 2

```php
$rabbitmq
    ->with(Consume::of('queue')->handle(
        static function(Message $message) use ($rabbitmq) {
            $url = decodeUrl($message);

            lock($url); // appel bloquant

            $urls = crawl($url);
            $rabbitmq
                ->with(Publish::many($urls)->to('queue'))
                ->run();
        },
    ))
    ->run();
```

Partitionnement / Sharding



```
$rabbitmq
    ->with(Consume::of('queue')->handle(
        static function(Message $message) use ($rabbitmq) {
            $url = decodeUrl($message);
            $urls = crawl($url);
            $fr = $urls->filter(isDotFr(...));
            $org = $urls->filter(isDotOrg(...));
            $rabbitmq
                ->with(Publish::many($fr)->to('queue1'))
                ->with(Publish::many($org)->to('queue2'))
                ->run();
        },
    ))
    ->run();
```


Filesystem
Lock 1
Lock 2
RabbitMQ
Queue 1
Queue 2
Acquire
Acquire
Acquire
Acquire
Pull
Pull
Pull
Pull
Process 1
Consumer 1
Process 2
Consumer 2
Process 3
Consumer 3
Process 4
Consumer 4

Complexité exponentielle

Problème insoluble ?

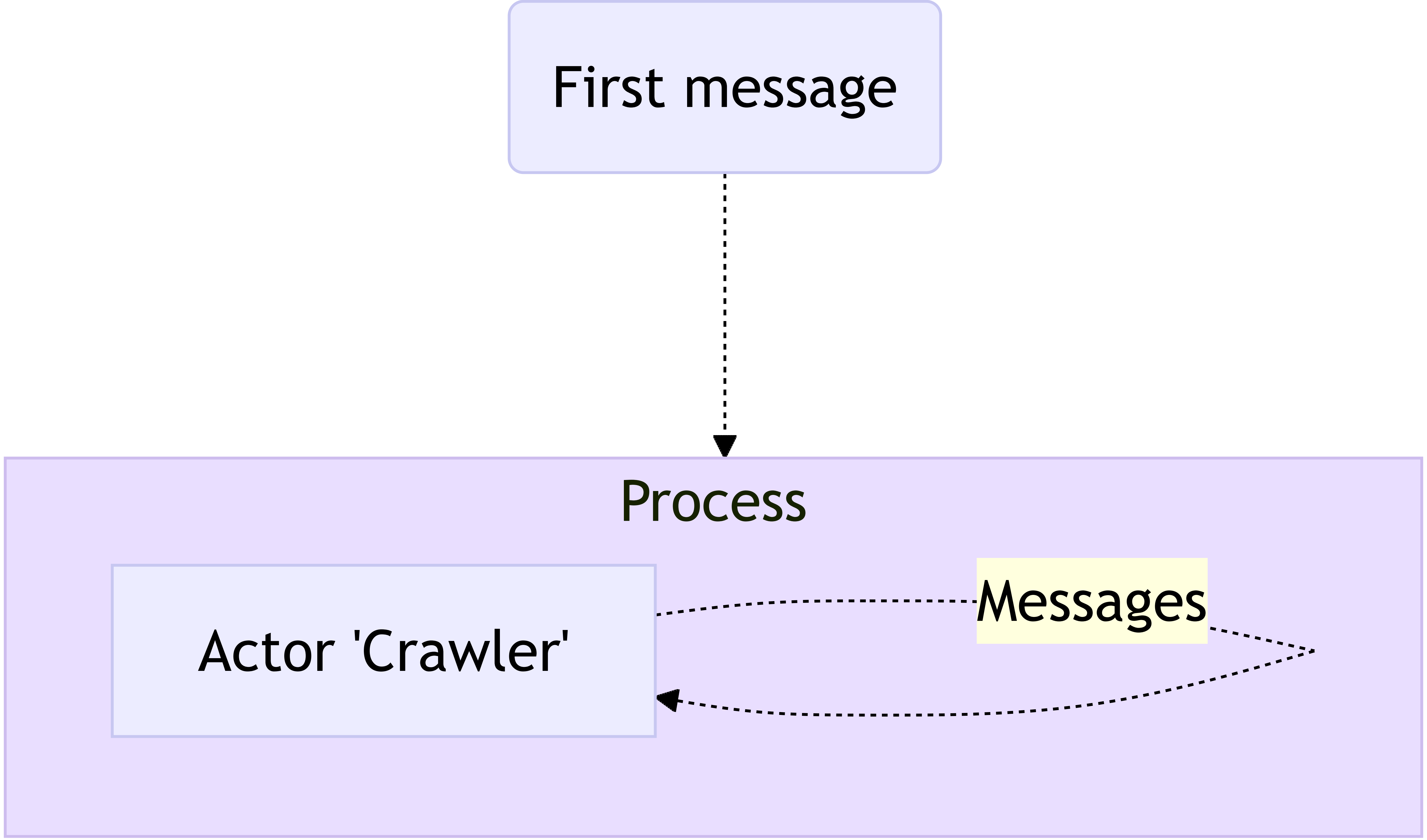
Actor Model

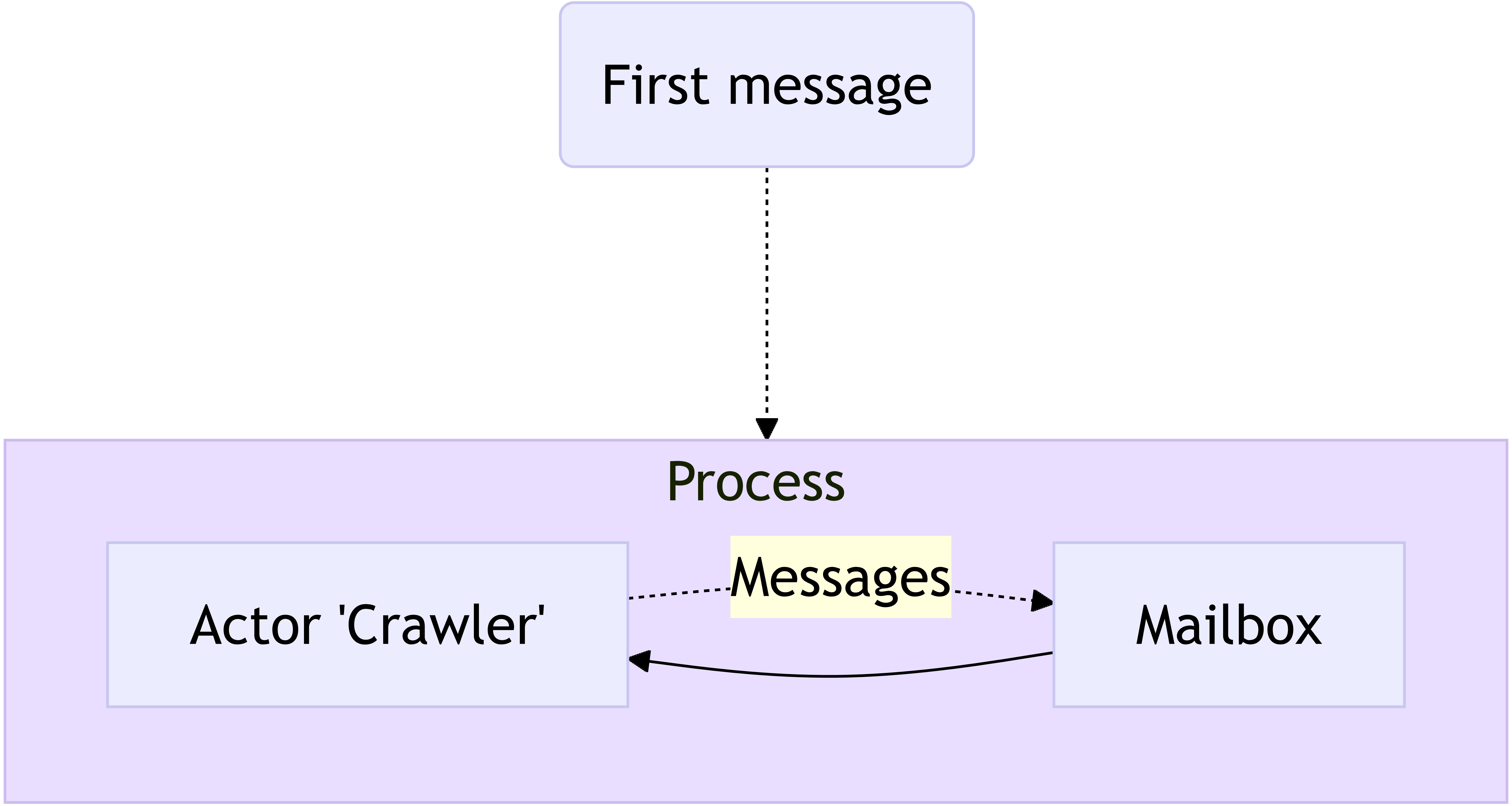
Actor

→Traite une file de messages

→Peut créer d'autres acteurs

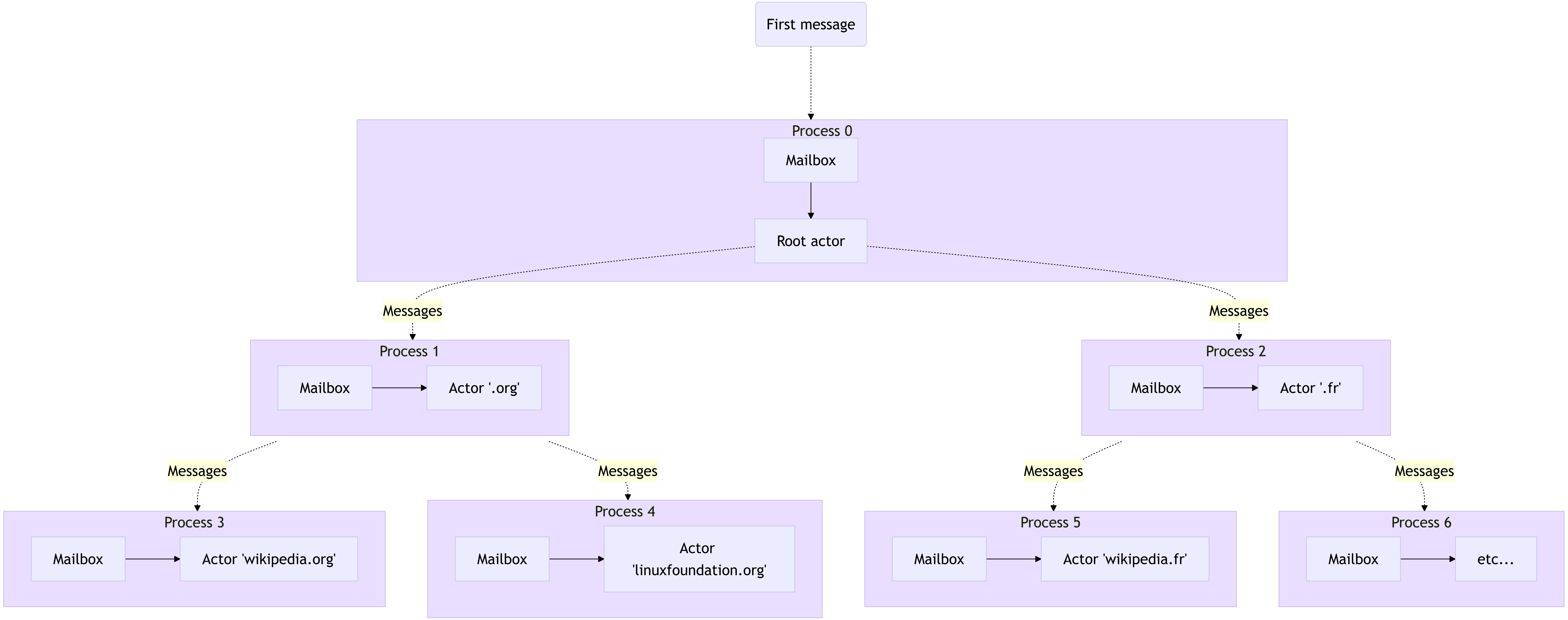
→Peut envoyer des messages aux autres acteurs


First message
Process
Actor 'Crawler'
Messages


First message
Process
Actor 'Crawler'
Messages
Mailbox

```
final class Crawler implements Actor
{
    public function __invoke(Receive $receive): Receive
    {
        return $receive->on(
            Url::class,
            function(
                Url $url,
                Address $sender,
                Continuation $continuation,
            ) {
                $urls = crawl($url);

                return $continuation->continue($urls);
            },
        );
    }
}
```

```
System::of()
    ->actor(
        Crawler::class,
        static fn() => new Crawler,
    )
    ->run(
        Crawler::class,
        Url::of('https://wikipedia.org'),
    );
```


First message
Process 0
Mailbox
Root actor
Messages
Messages
Process 1
Mailbox
Actor '.org'
Process 2
Mailbox
Actor '.fr'
Messages
Messages
Messages
Messages
Process 3
Mailbox
Actor 'wikipedia.org'
Process 4
Mailbox
Actor 'linuxfoundation.org'
Process 5
Mailbox
Actor 'wikipedia.fr'
Process 6
Mailbox
etc...

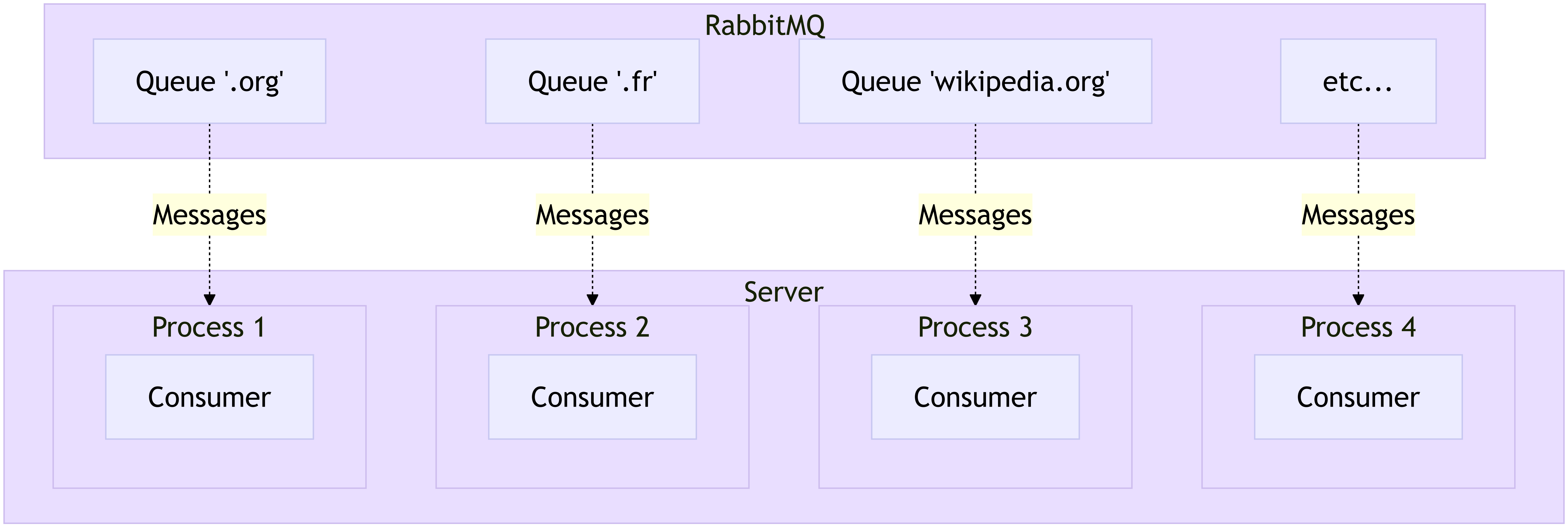
```php
final class Crawler implements Actor
{
    /** @var array<string, Address> */
    private array $tlds;

    public function __construct(private Spawn $spawn)
    {
    }

    public function __invoke(Receive $receive): Receive
    {
        return $receive->on(
            Url::class,
            function(Url $url, Address $sender, Continuation $continuation) {
                $tld = $url->tld();
                $child = $this->tlds[$tld] ??= ($this->spawn)(
                    ChildCrawler::class,
                );

                $child($url);

                return $continuation->continue();
            },
        );
    }
}
```

```
final class ChildCrawler implements Actor
{
    public function __invoke(Receive $receive): Receive
    {
        return $receive->on(
            Url::class,
            function(Url $url, Address $sender, Continuation $continuation) {
                $urls = crawl($url);
                $urls->foreach(static fn($url) => $sender($url));

                return $continuation->continue();
            },
        );
    }
}
```

```
System::of()
    ->actor(
        Crawler::class,
        static fn($_, $__, Spawn $spawn) => new Crawler($spawn),
    )
    ->actor(
        ChildCrawler::class,
        static fn() => new ChildCrawler,
    )
    ->run(
        Crawler::class,
        Url::of('https://wikipedia.org'),
    );
```

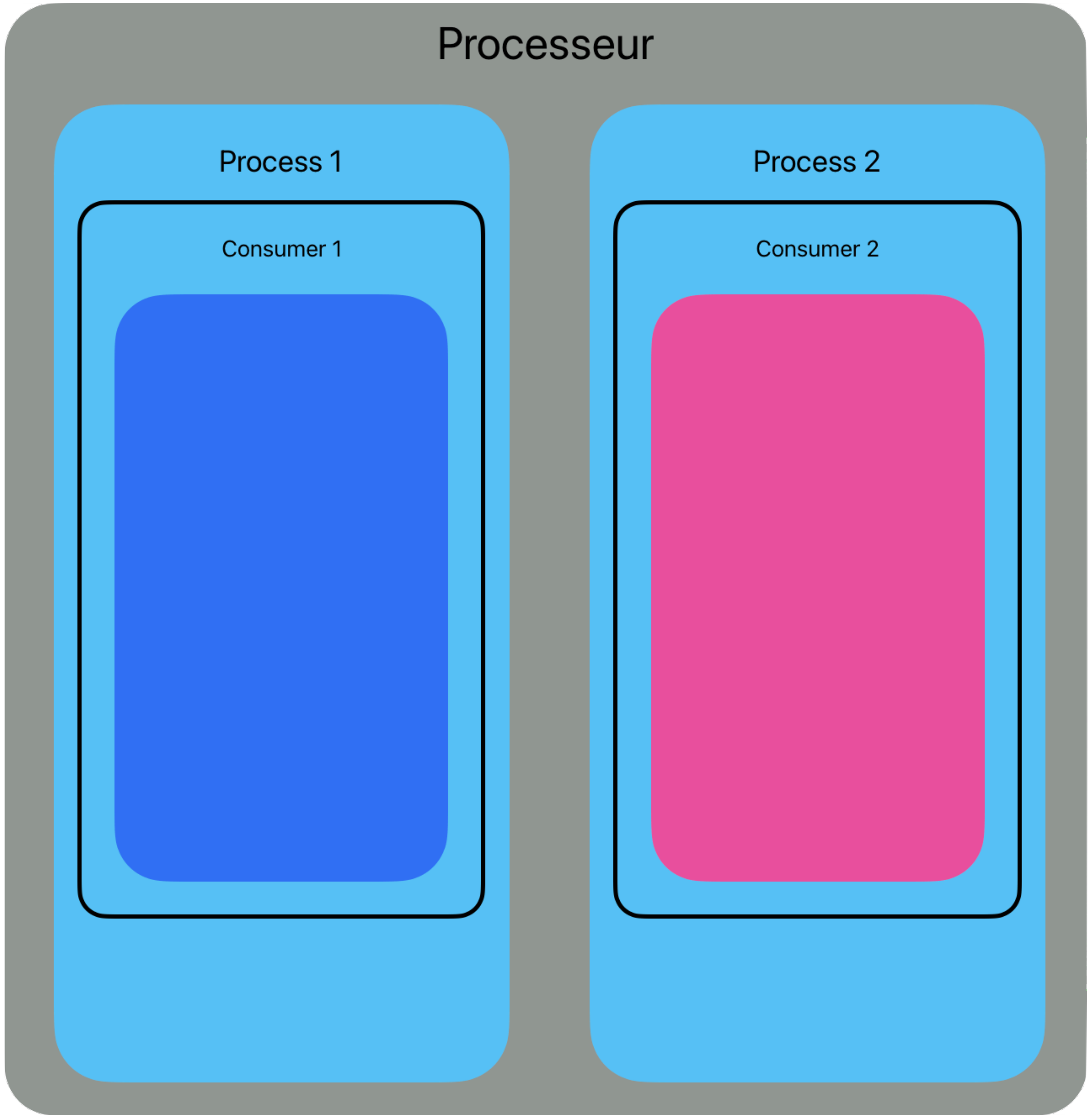
En pratique ça donne quoi ?

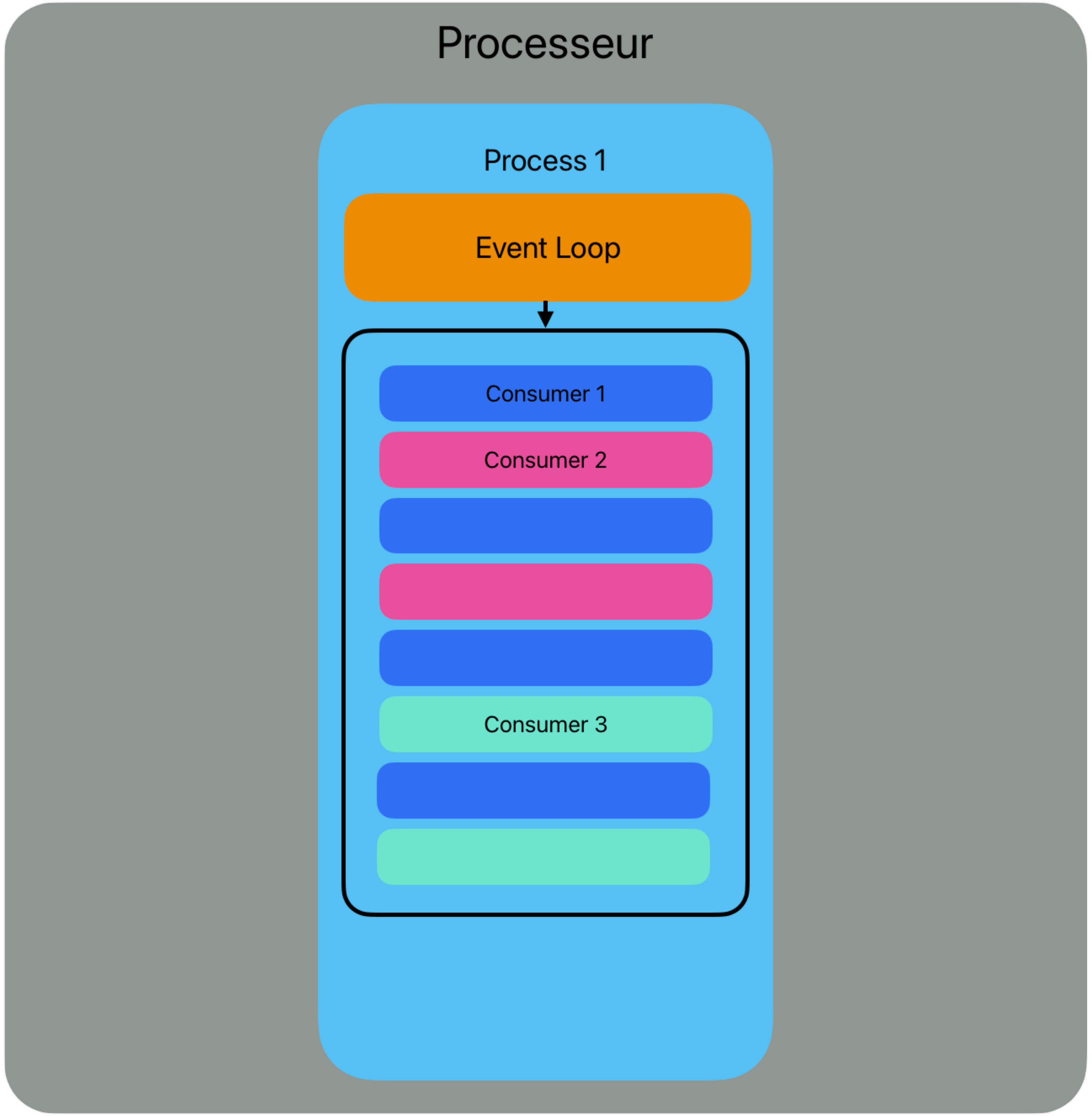
Actor Model	RabbitMQ
Mailbox	Queue
Actor	Consumer

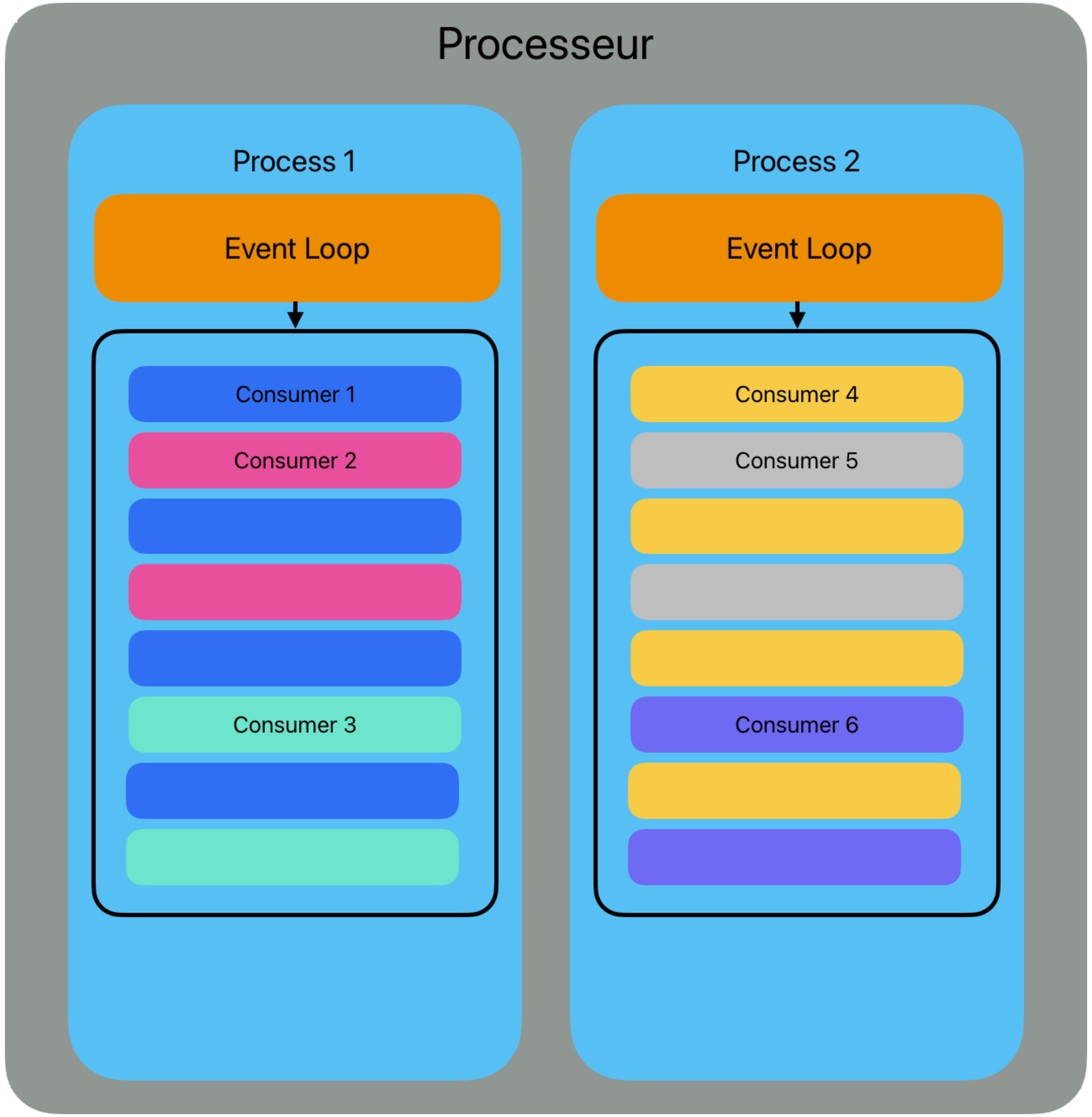

RabbitMQ
Queue '.org'
Queue '.fr'
Queue 'wikipedia.org'
etc...
Messages
Messages
Messages
Messages
Server
Process 1
Consumer
Process 2
Consumer
Process 3
Consumer
Process 4
Consumer

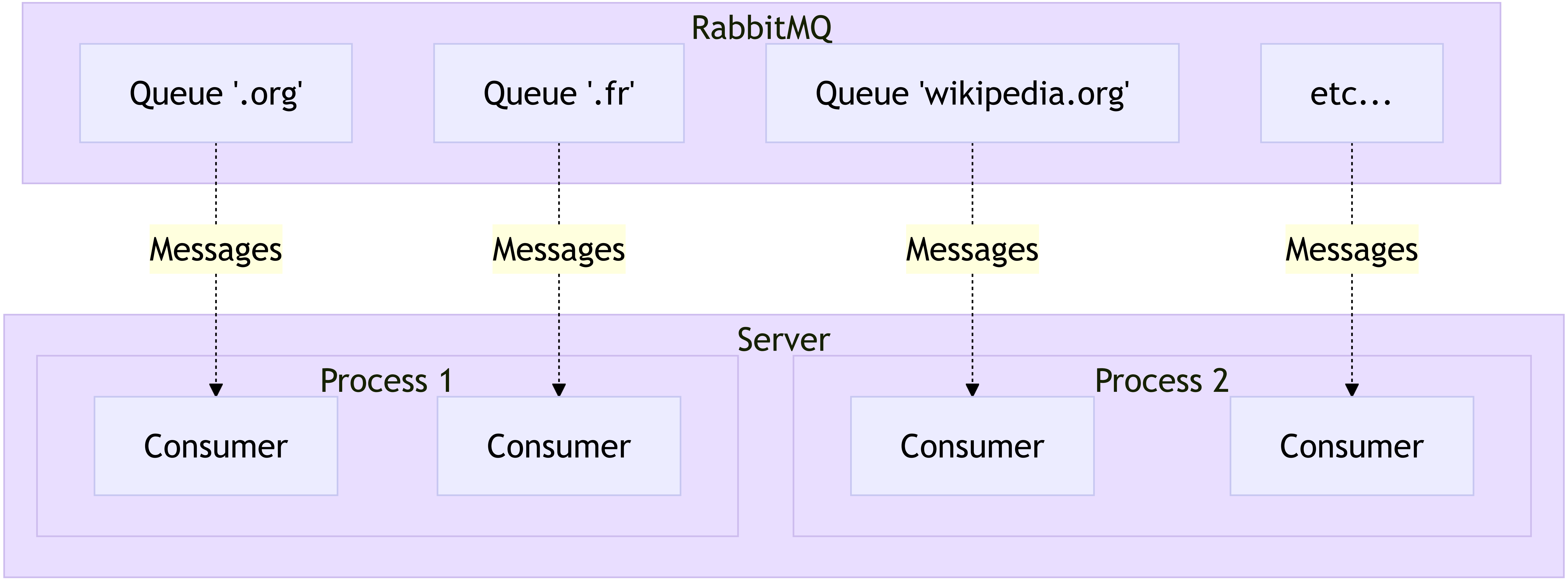
Problème de ressources ?

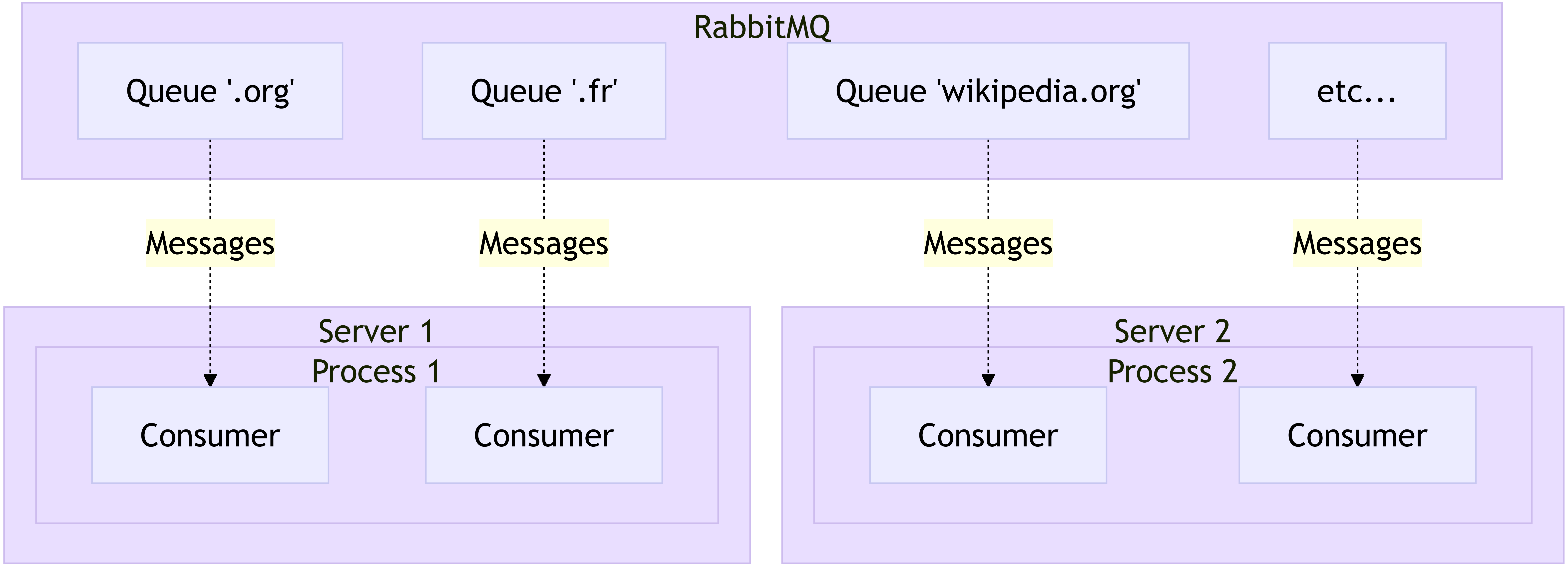
Parallélisation + Asynchrone


Processeur
Process 1
Consumer 1
Process 2
Consumer 2


Processeur
Process 1
Event Loop
Consumer 1
Consumer 2
Consumer 3


Processeur
Process 1
Event Loop
Consumer 1
Consumer 2
Consumer 3
Process 2
Event Loop
Consumer 4
Consumer 5
Consumer 6


RabbitMQ
Queue '.org'
Queue '.fr'
Queue 'wikipedia.org'
etc...
Messages
Messages
Messages
Messages
Server
Process 1
Process 2
Consumer
Consumer
Consumer
Consumer


RabbitMQ
Queue '.org'
Queue '.fr'
Queue 'wikipedia.org'
etc...
Messages
Messages
Messages
Messages
Server 1
Process 1
Consumer
Consumer
Server 2
Process 2
Consumer
Consumer

Avantages

Scalabilité infinie

Résilience

Déploiement progressif

Unifier des paradigmes différents

https://innmind.org

Innmind

Search

Home Philosophy Getting started Use cases Testing Tools Other Packages Blog

Welcome to Innmind

Innmind bridges Object Oriented Programming and Functional Programming in a coherent ecosystem to bring high level abstraction to life.

This documentation will show you how to move from simple scripts all the way to distributed systems (and all the steps in-between) by using a single way to code.

If you've seen modern Java, C#, Rust, Swift a

Sneak peek

The code below shows how the declarative nature of

```
$os
    ->filesystem()
    ->mount(Path::of('somewhere/data/'))
    ->get(Name::of('avatars'))
    ->keep(Instance::of(Directory::class))
    ->map(
        static fn(Directory $directory) => $
            'users.csv',
            Content::ofLines(
                $orm
                    ->repository(User::class
                    ->all()
                    ->map(static fn(User $us
                    ->map(static fn(array $u
                    ->map(Str::of(...))
                    ->map(Line::of(...)),
            ),
        )),
    )
    ->map(Tar::encode($os->clock()))
    ->map(Gzip::encode())
    ->match(
        static fn(File $tar) => Response::of
            StatusCode::ok,
            ProtocolVersion::v11,
            null,
            $tar->content(),
        ),
        static fn() => Response::of(
            StatusCode::noContent,
            ProtocolVersion::v11,
```

Monades

09 & 10 Octobre 2025
Hotel New-York - TAOM
Disneyland Paris

FORUM PHP
PARIS 2025

event.afup.org

ARCHIVER EFFICACEMENT DE GRANDS VOLUMES DE DONNÉES GRÂCE AUX MONADES

Baptiste LANGLADE

afup

Tests

FORUMPHP
PARIS2023

afup

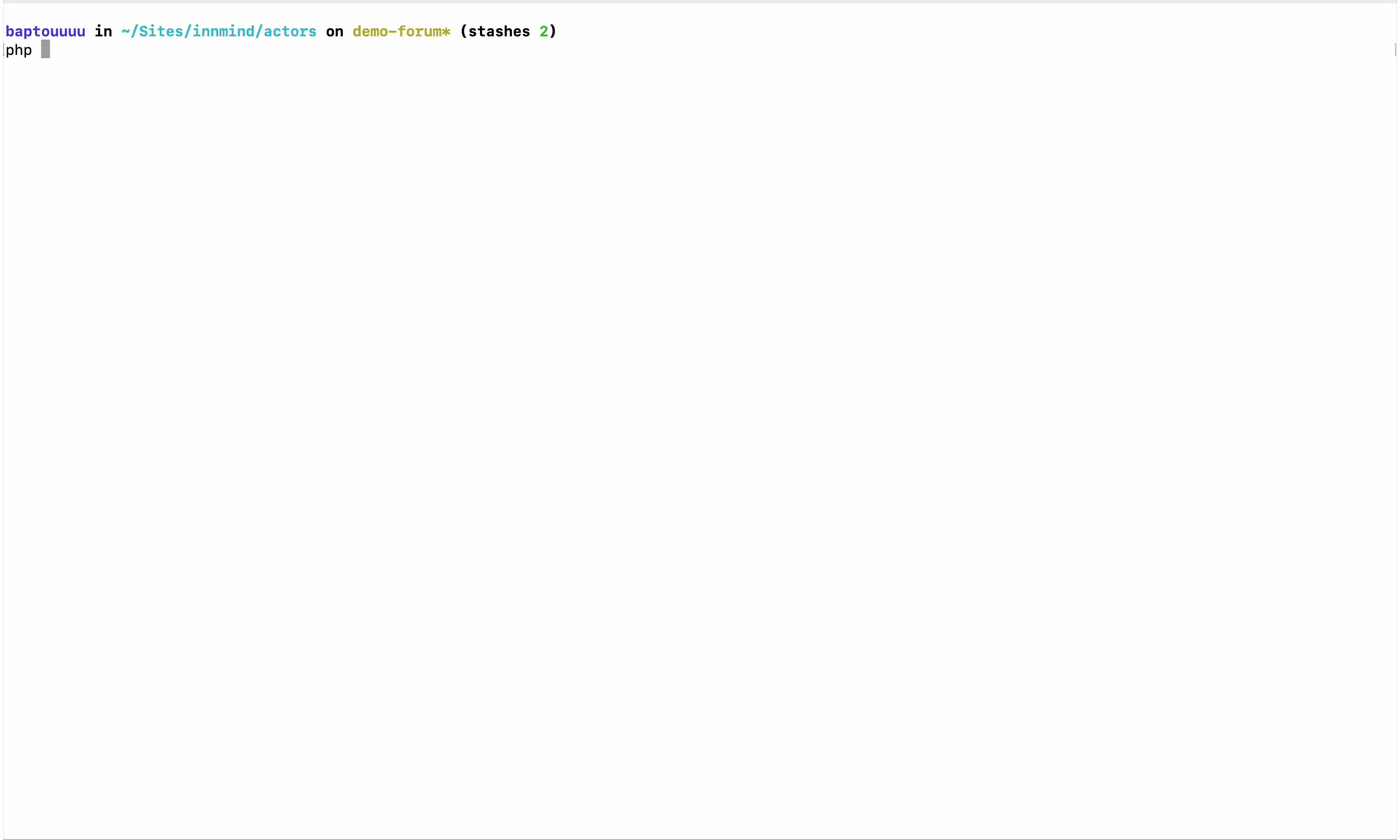
Baptiste LANGLADE

Augmentez votre couverture : supprimez des tests !

12 et 13 octobre 2023
HOTEL NEW YORK - TAOM
DISNEYLAND PARIS

event.afup.org

Demo



```
baptouuuu in ~/Sites/innmind/actors on demo-forum* (stashes 2)
php 
```

🚧 **https://github.com/innmind/actors** 🚧

Questions

baptiste_forum20.25

X/Bluesky/Mastodon @Baptouuuu

https://baptouuuu.github.io/conferences/