# Generators for Asynchronous Programming

User Manual

Benoit Viguier
@b_viguier

# Generators for Asynchronous Programming

User Manual

Benoit Viguier
@b_viguier

6M WEB

FORUMPHP PARIS2018

# Generators for Asynchronous Programming

User Manual

Benoit Viguier
@b_viguier

6M WEB

FORUMPHP PARIS2018

# Generators for Asynchronous Programming

User Manual

Benoit Viguier
@b_viguier

6M WEB

FORUMPHP PARIS2018

# Generators for Asynchronous Programming

User Manual

Benoit Viguier
@b_viguier

6M WEB

FORUMPHP PARIS2018

# Generators for Asynchronous Programming

## User Manual

Benoit Viguier
@b_viguier

6 WEB

FORUMPHP PARIS2018

# Generators for Asynchronous Programming

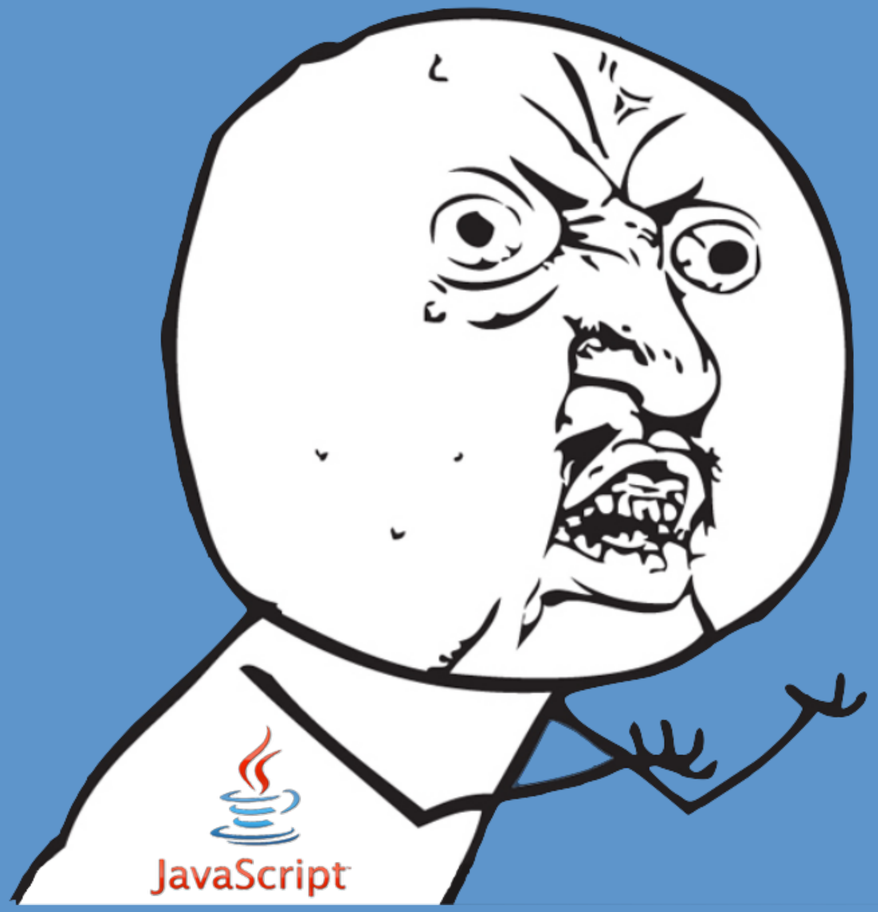
User Manual

Benoit Viguier
@b_viguier

6M WEB

FORUMPHP PARIS2018

# Asynchronous Programming

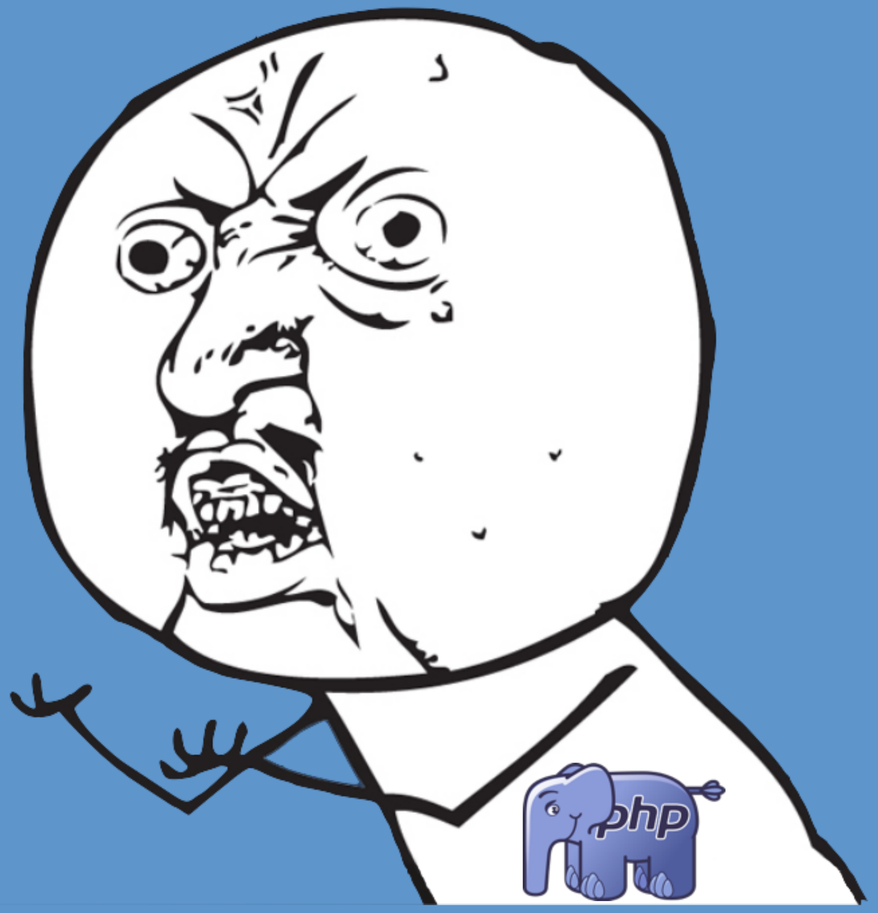



# PHP DEV

# Y U NO USE JAVASCRIPT

# PHP DEV

# Y U NO USE PHP



# Tea for Two

An illustrated example

# Tasks

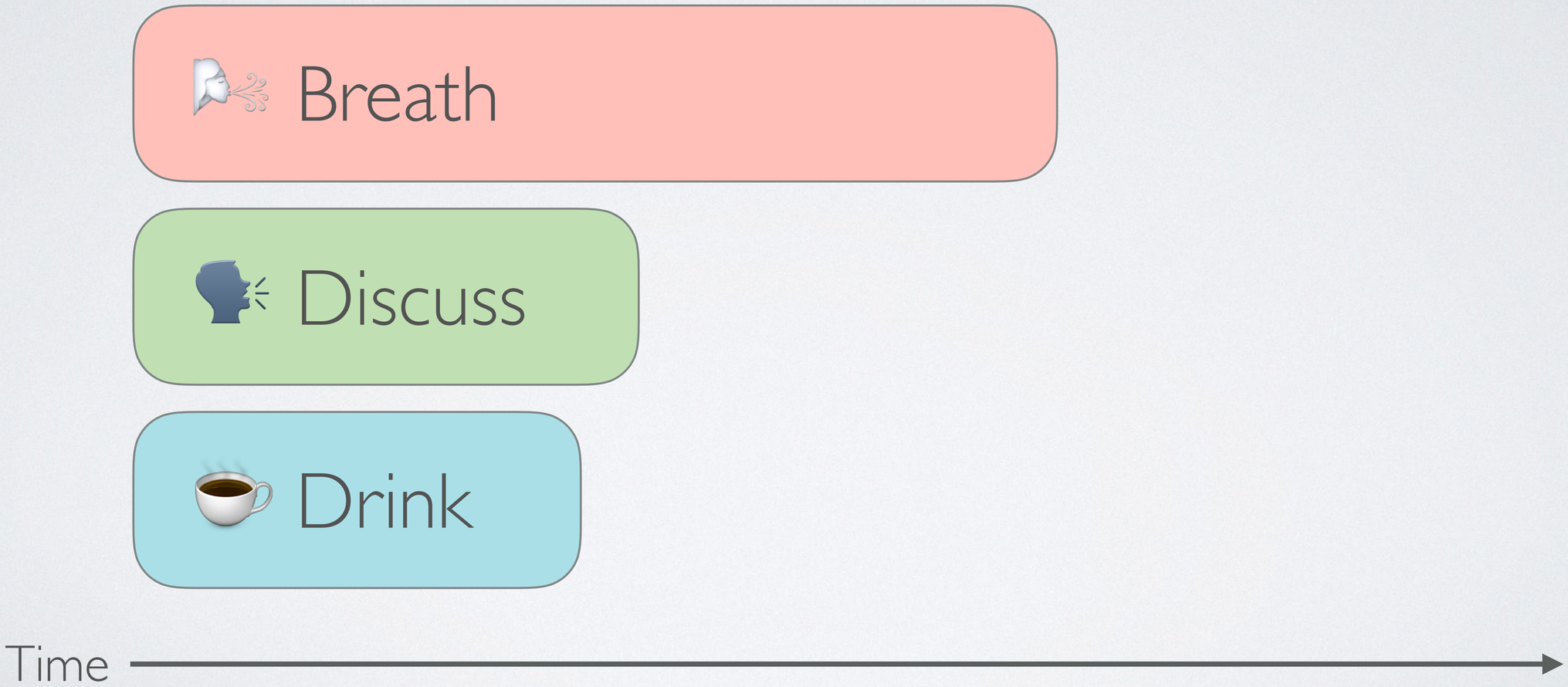
- ☕ **Drinking** Tea
- 🗣 **Discussing** with a friend
- 🌬 **Breathing** just… to live

# Synchronous

# Concurrency

Starting next task before that current one ends

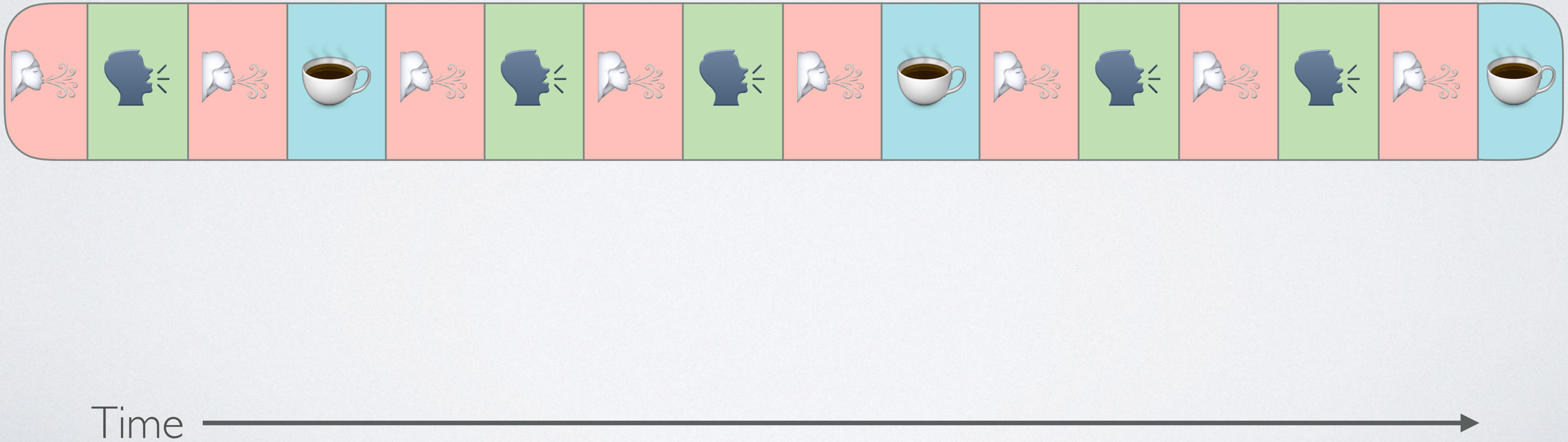
# Parallel



# ⚠️ Php is single threaded ⚠️

… and it’s a good thing

# Asynchronous



# But… 🤔

Time →

# Analyzing tasks

*Internal/External* operations

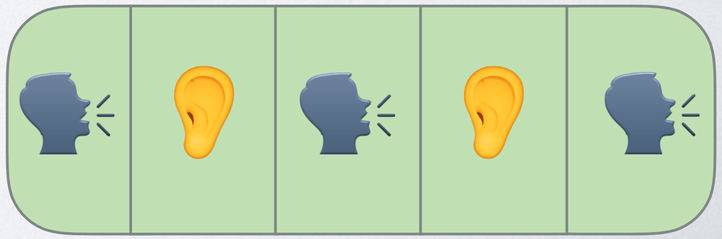
# Drinking Tea

- Waiting good temperature
- Drinking (actually)
- Enjoying

# **Discussing** with a friend

Speaking, asking something

Listening

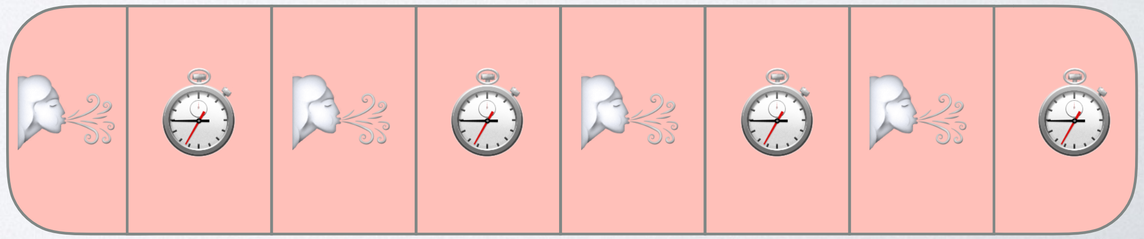


# Breathing

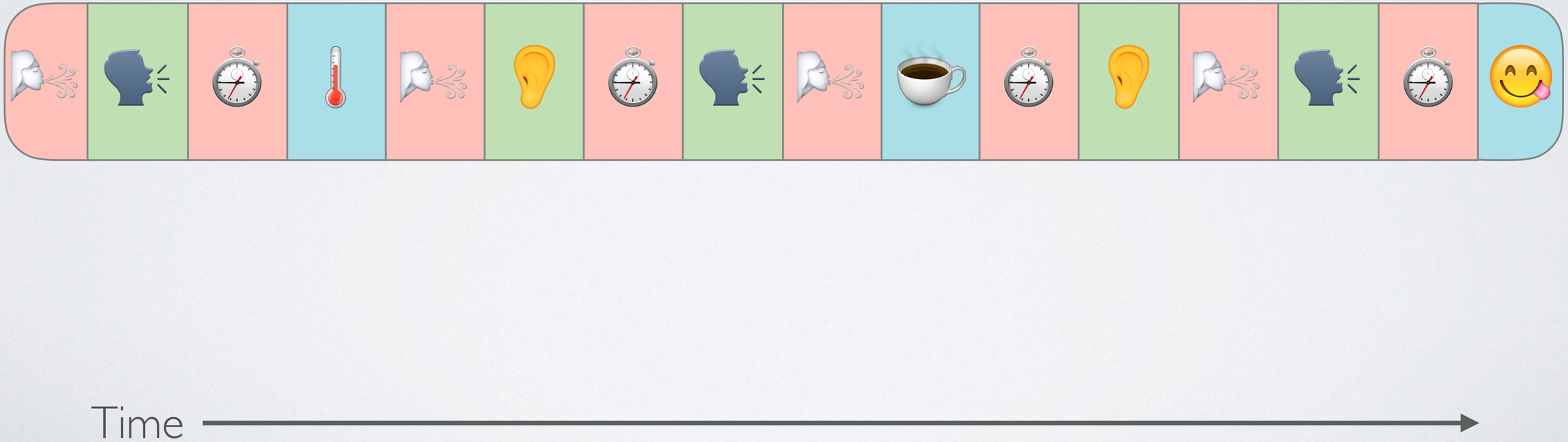
Breathing…

Waiting (a little)

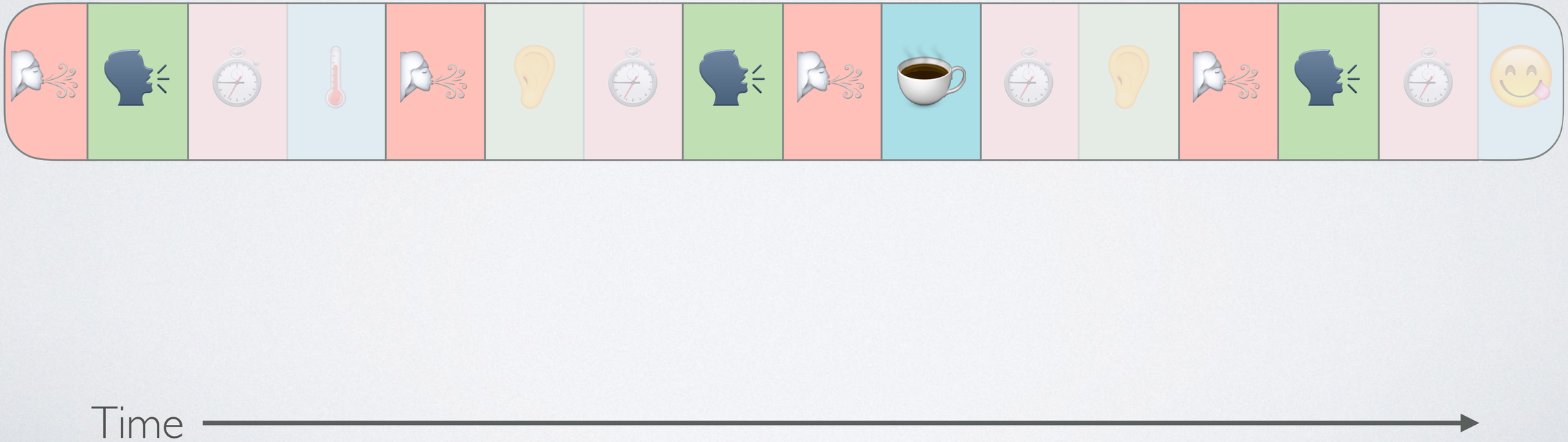
Breathing…



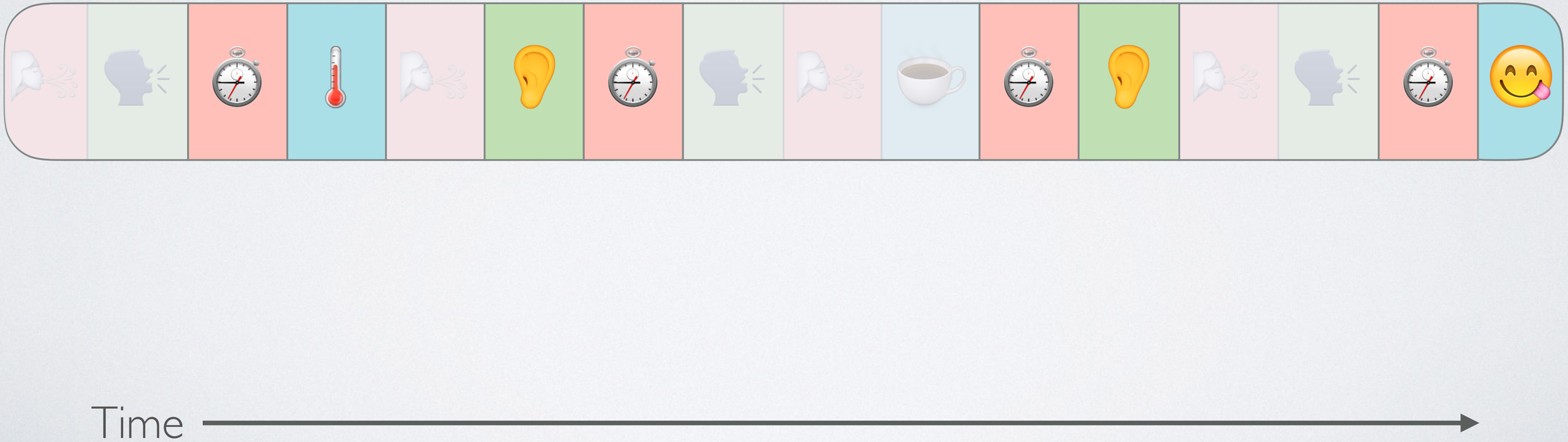
# Asynchronous



# *Internal* operations



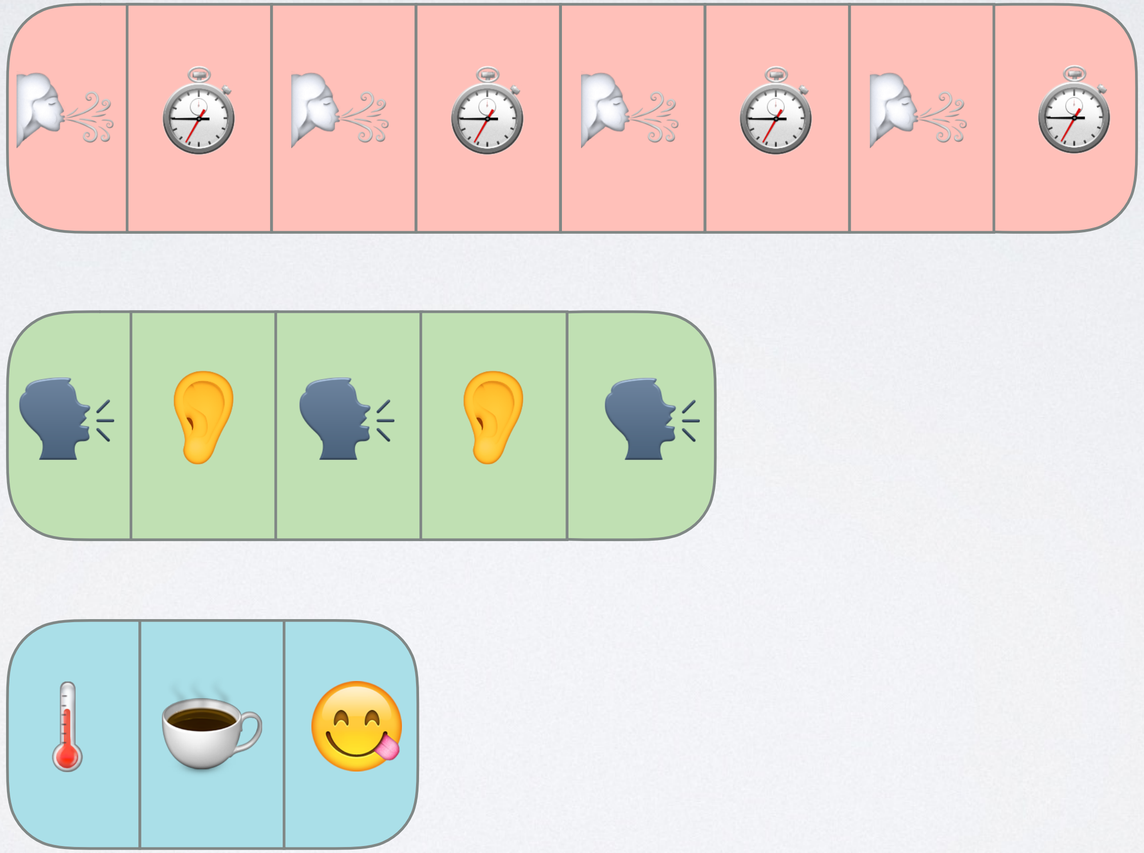
# *External* operations



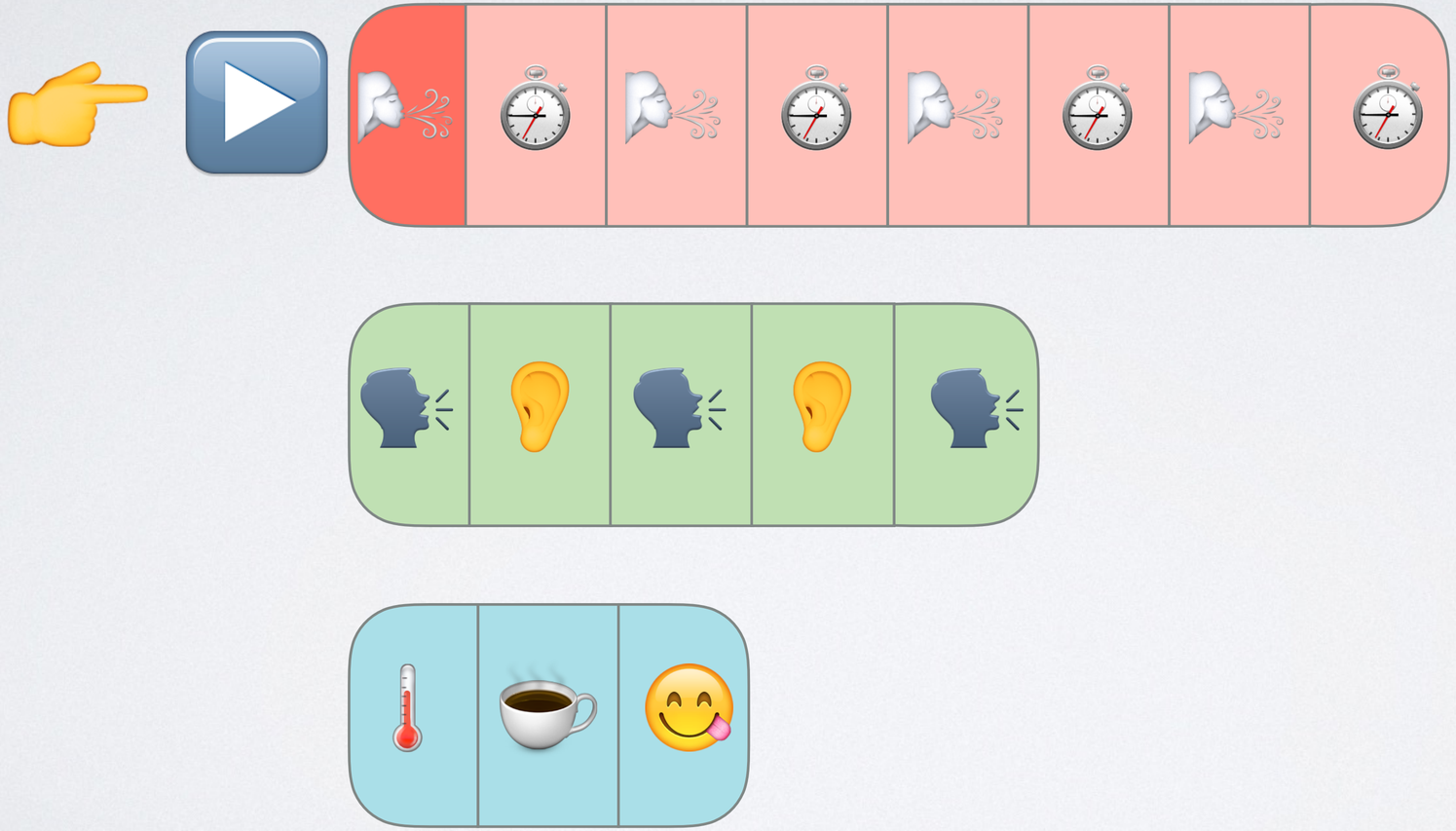
# Dispatching execution

Event Loop

# Event Loop



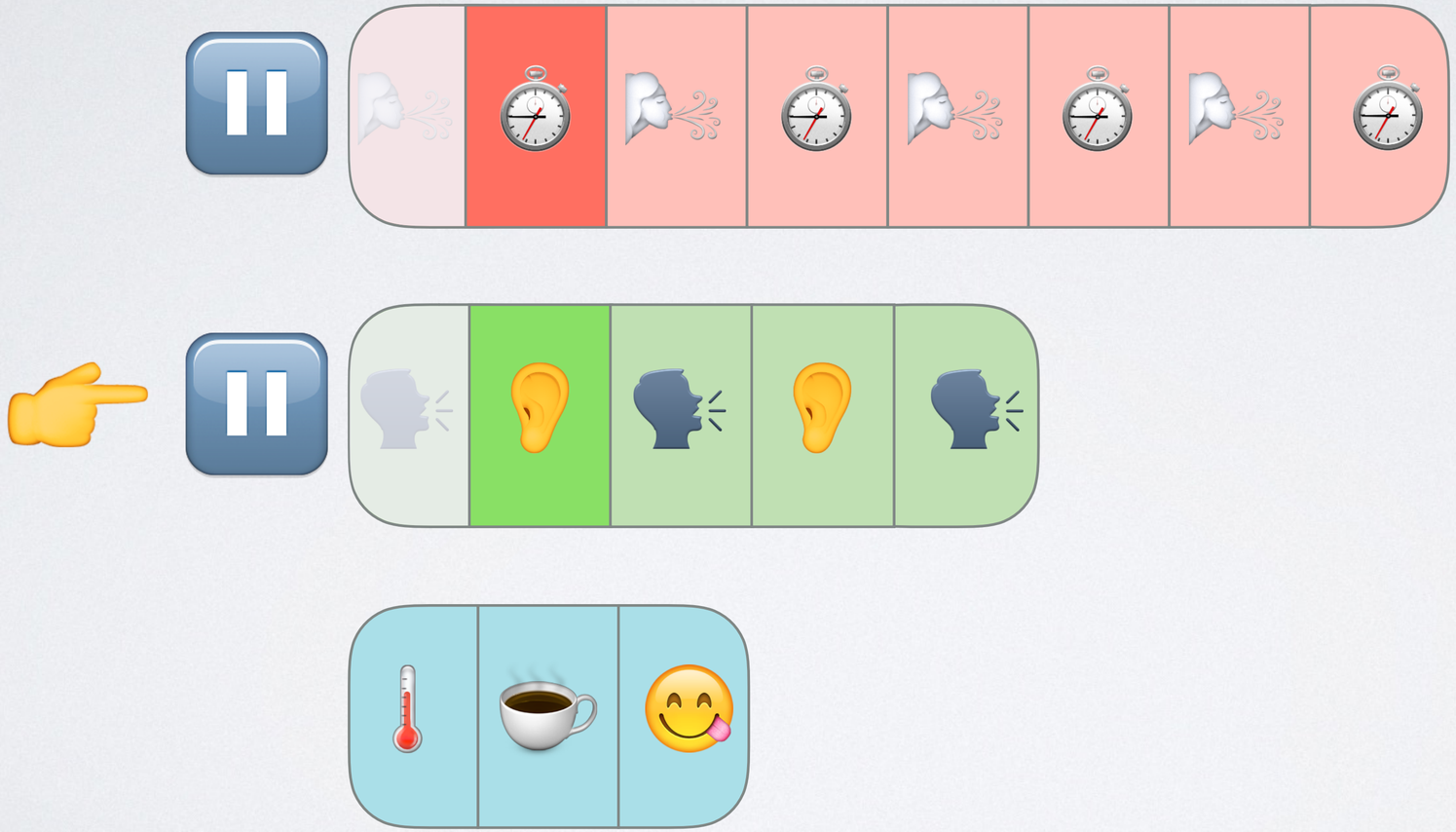
# Event Loop



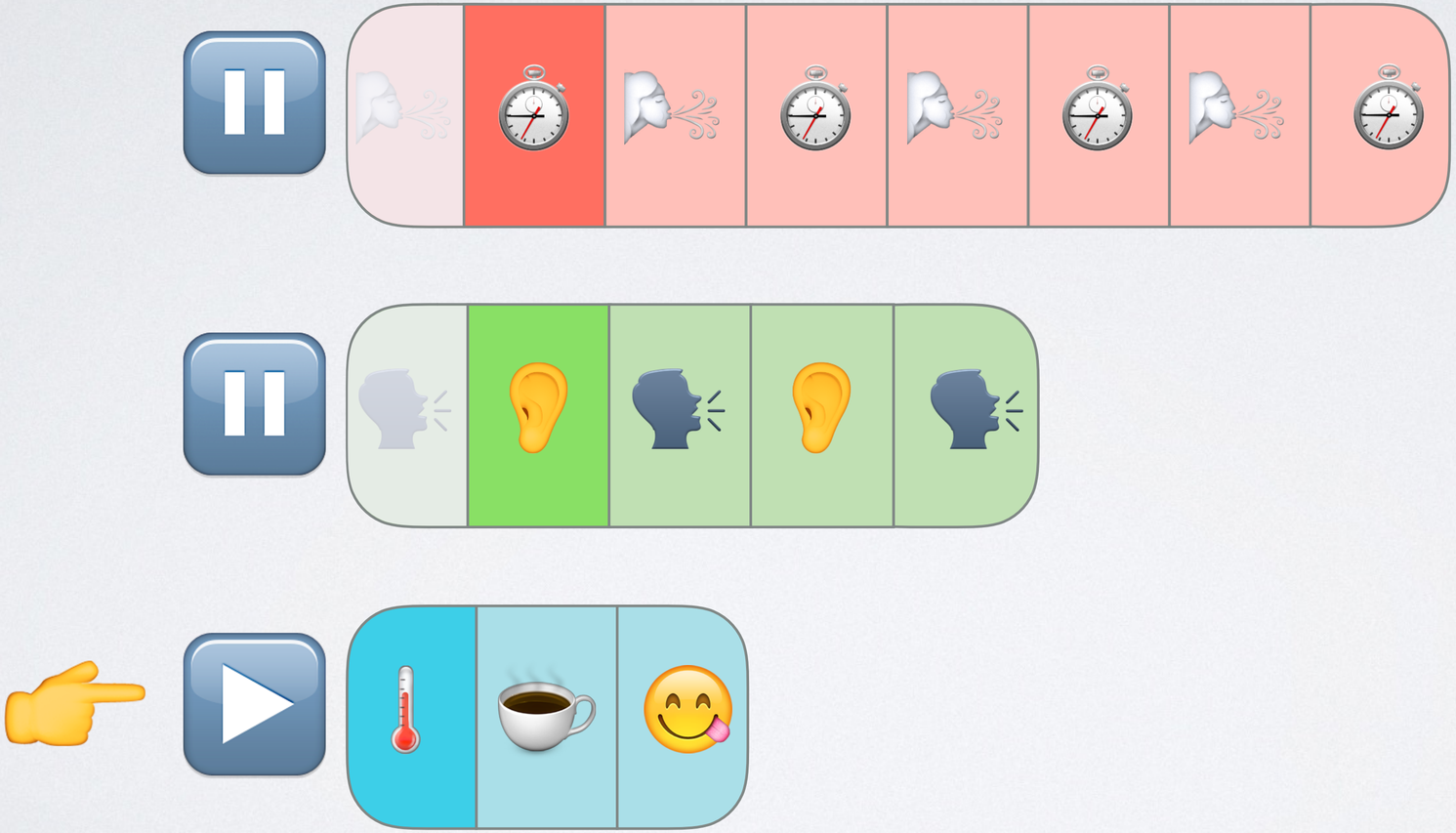
# Event Loop

# Event Loop

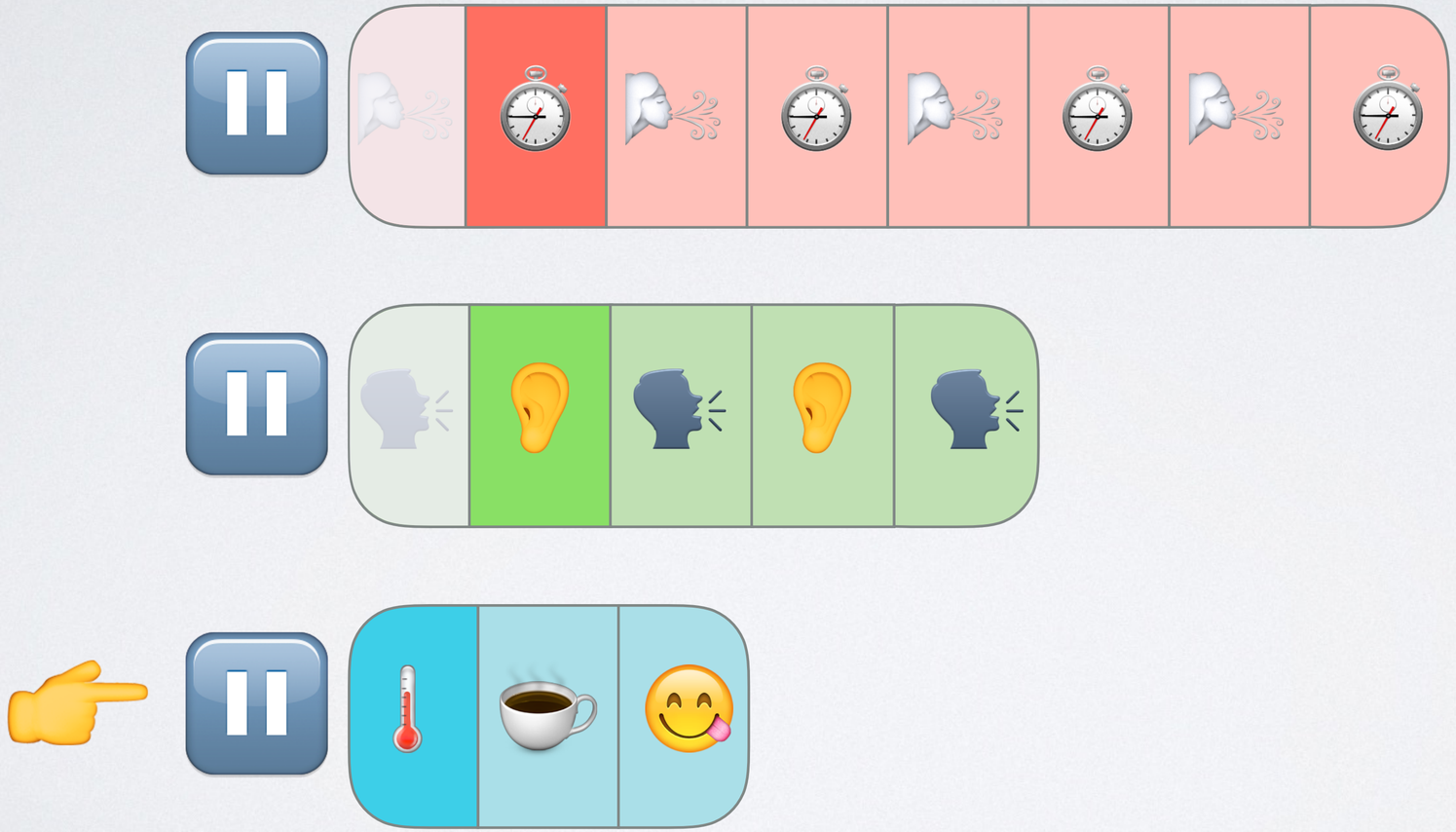
# Event Loop



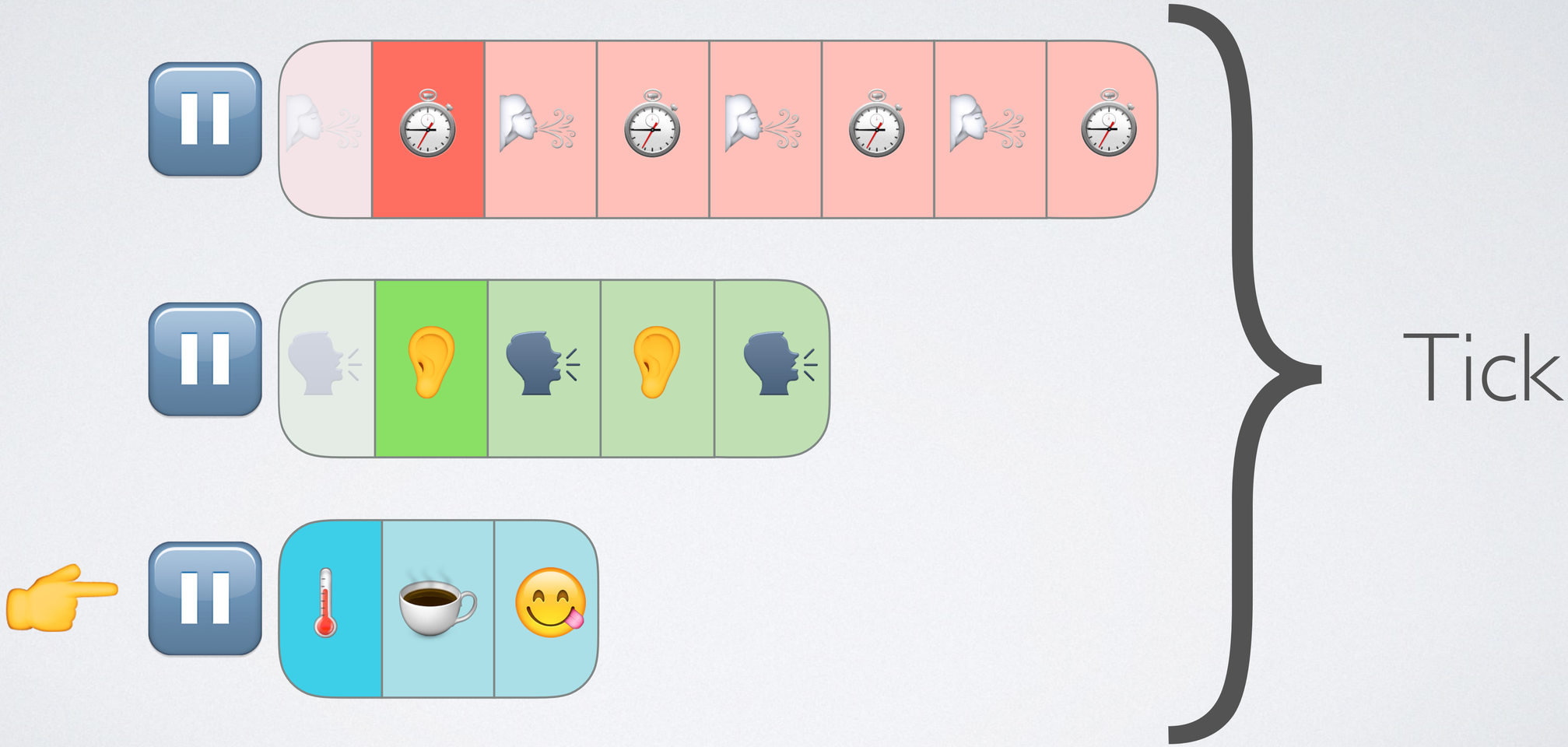
# Event Loop



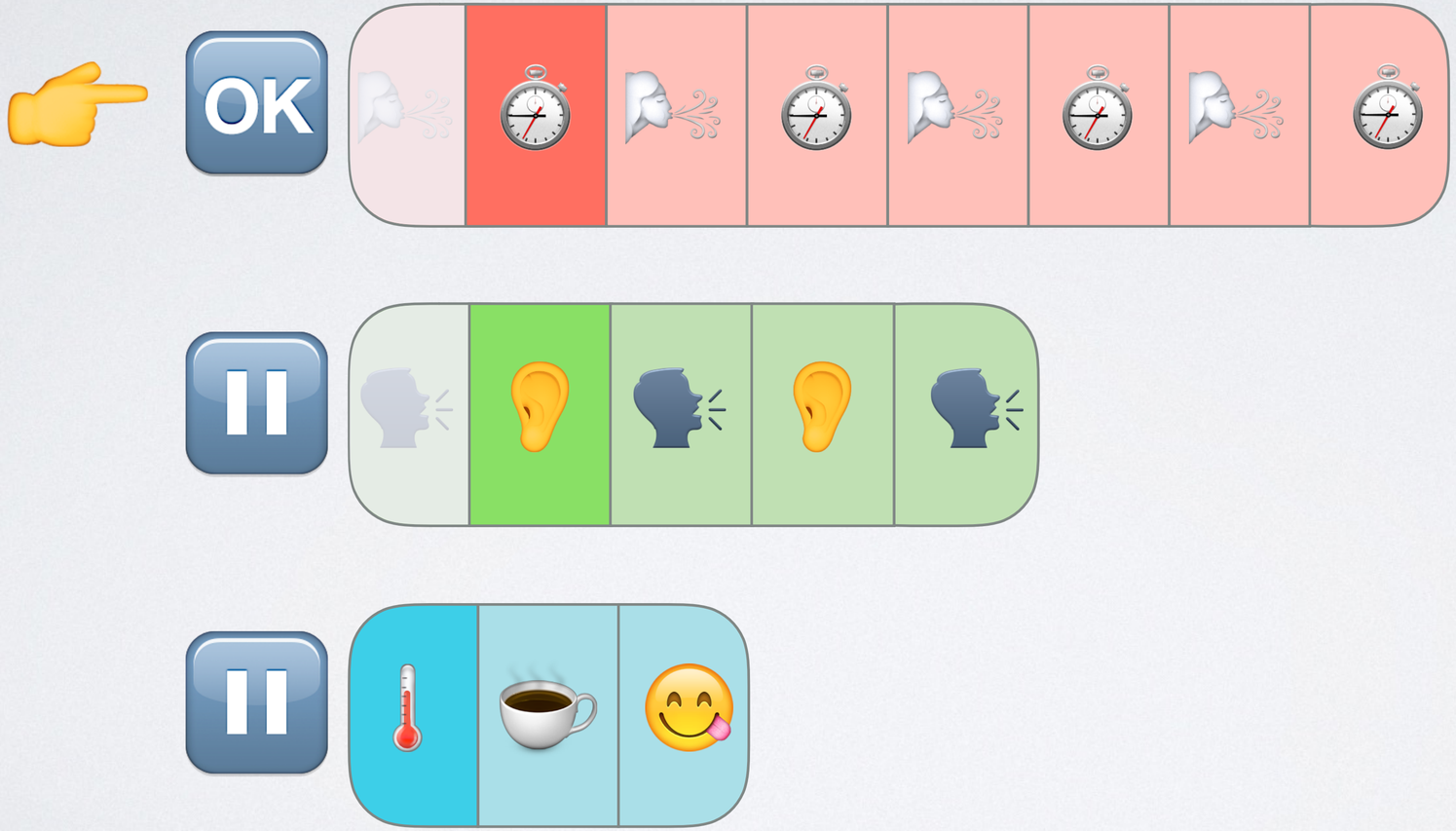
# Event Loop



# Event Loop



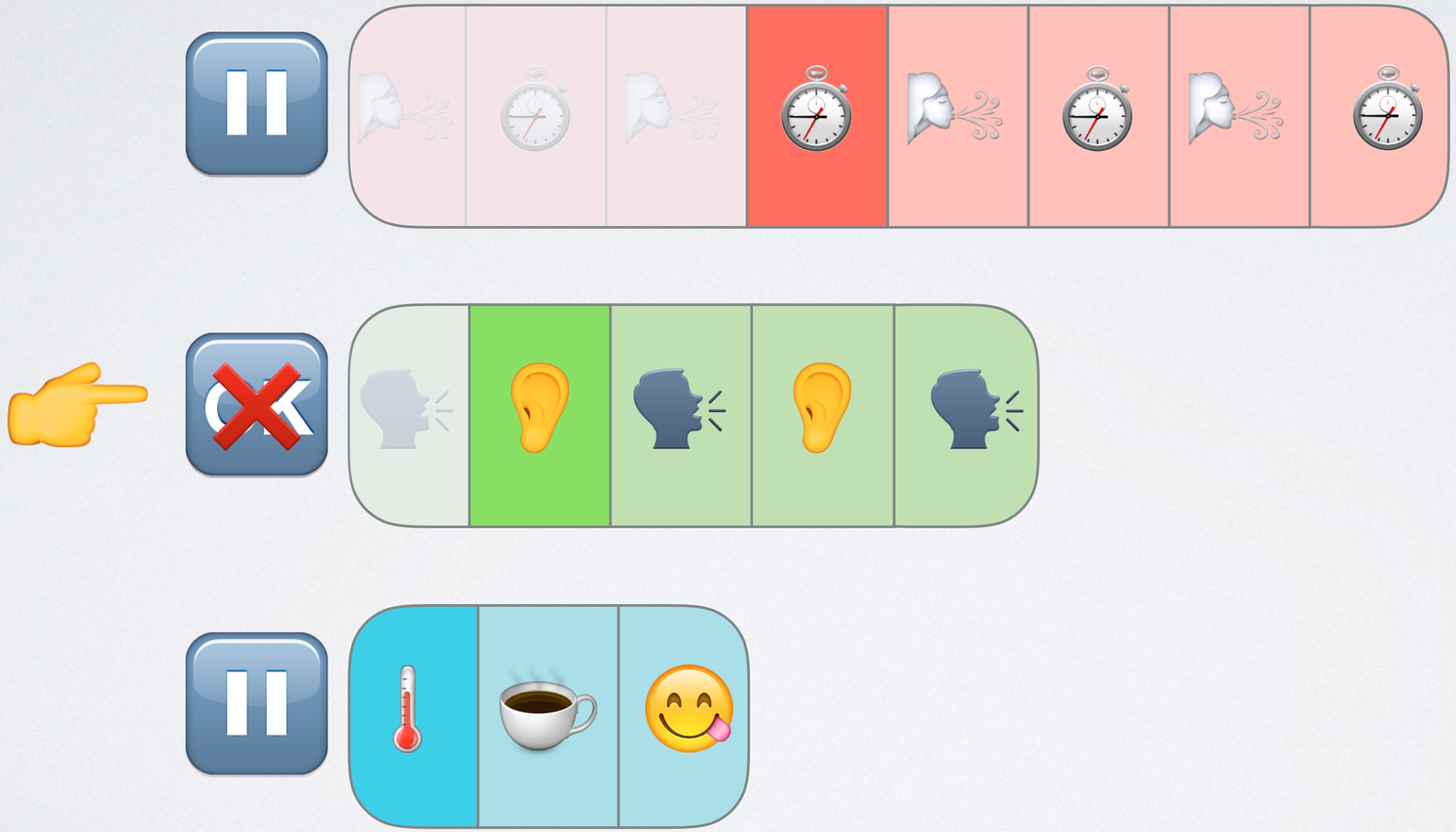
# Event **Loop**



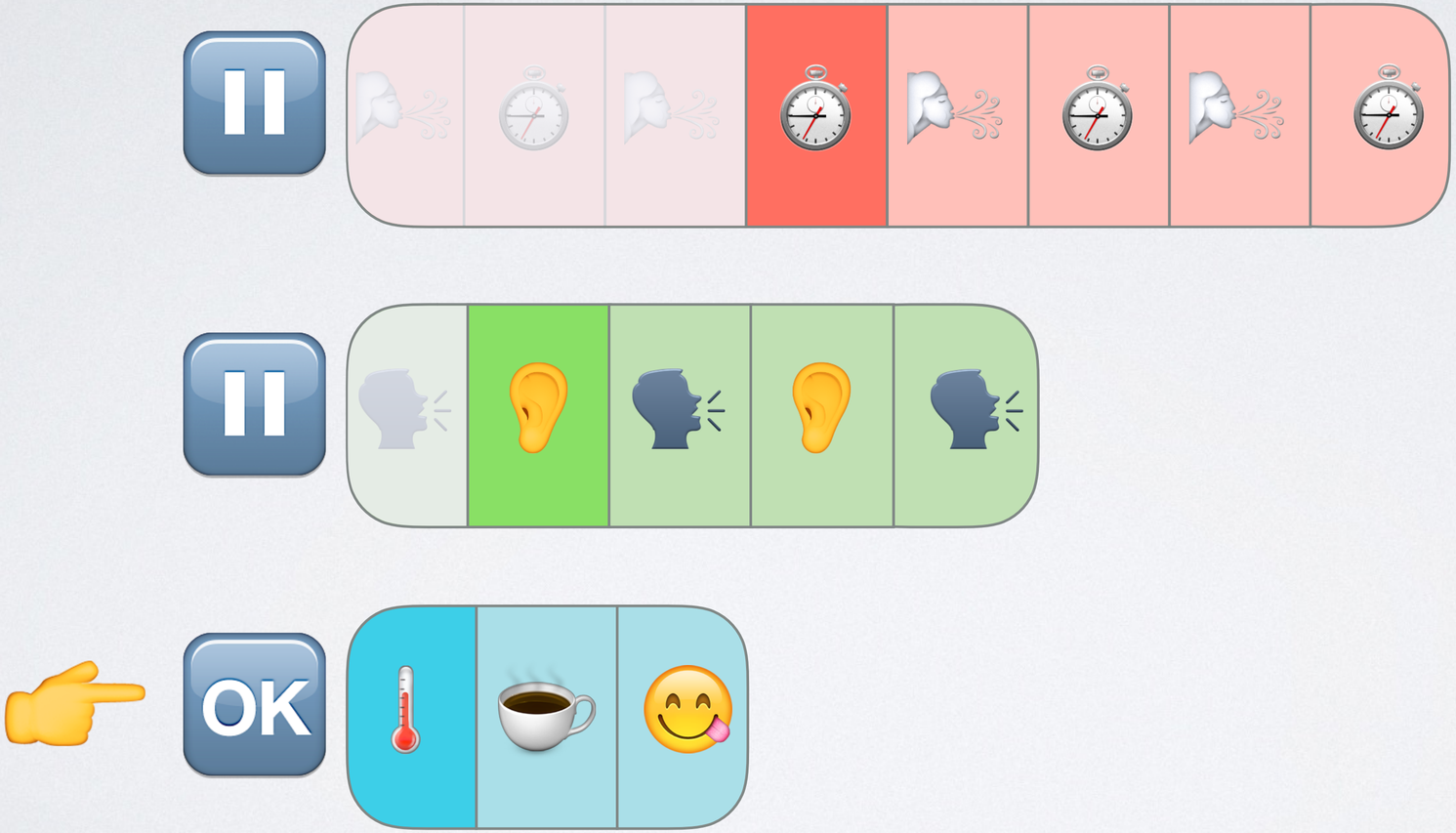
# Event Loop

# Event Loop

# Event Loop



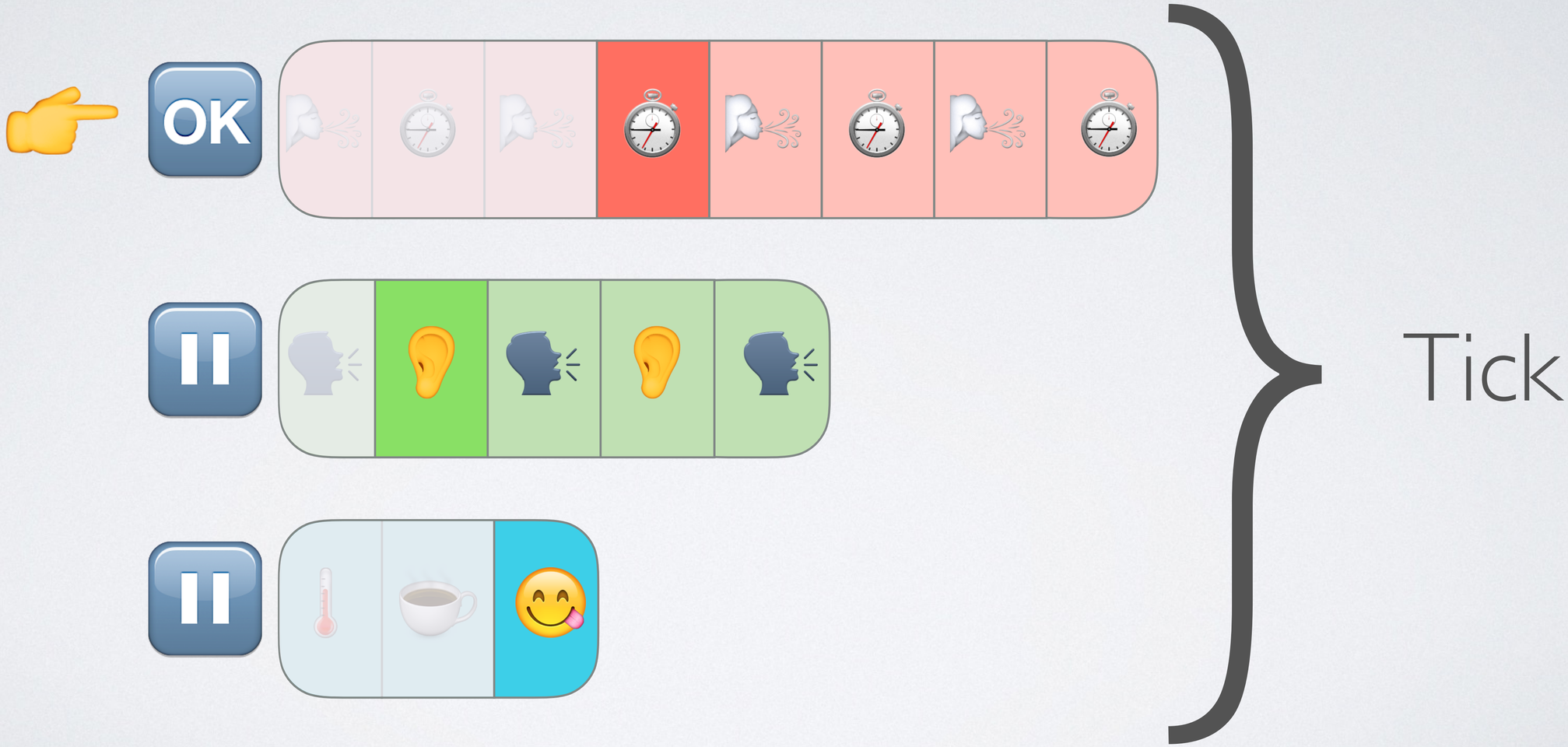
# Event Loop



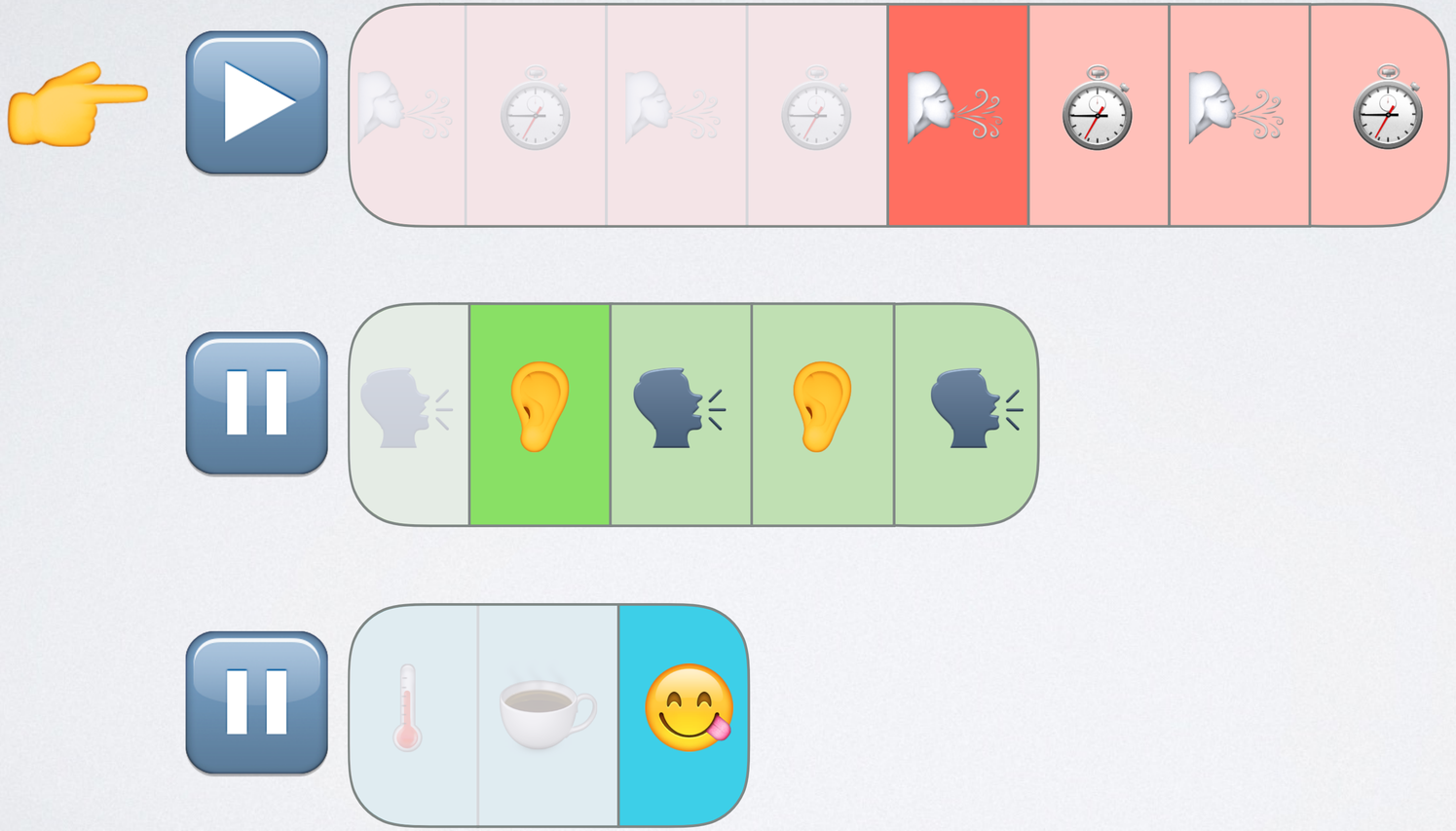
# Event Loop

# Event Loop

# Event Loop

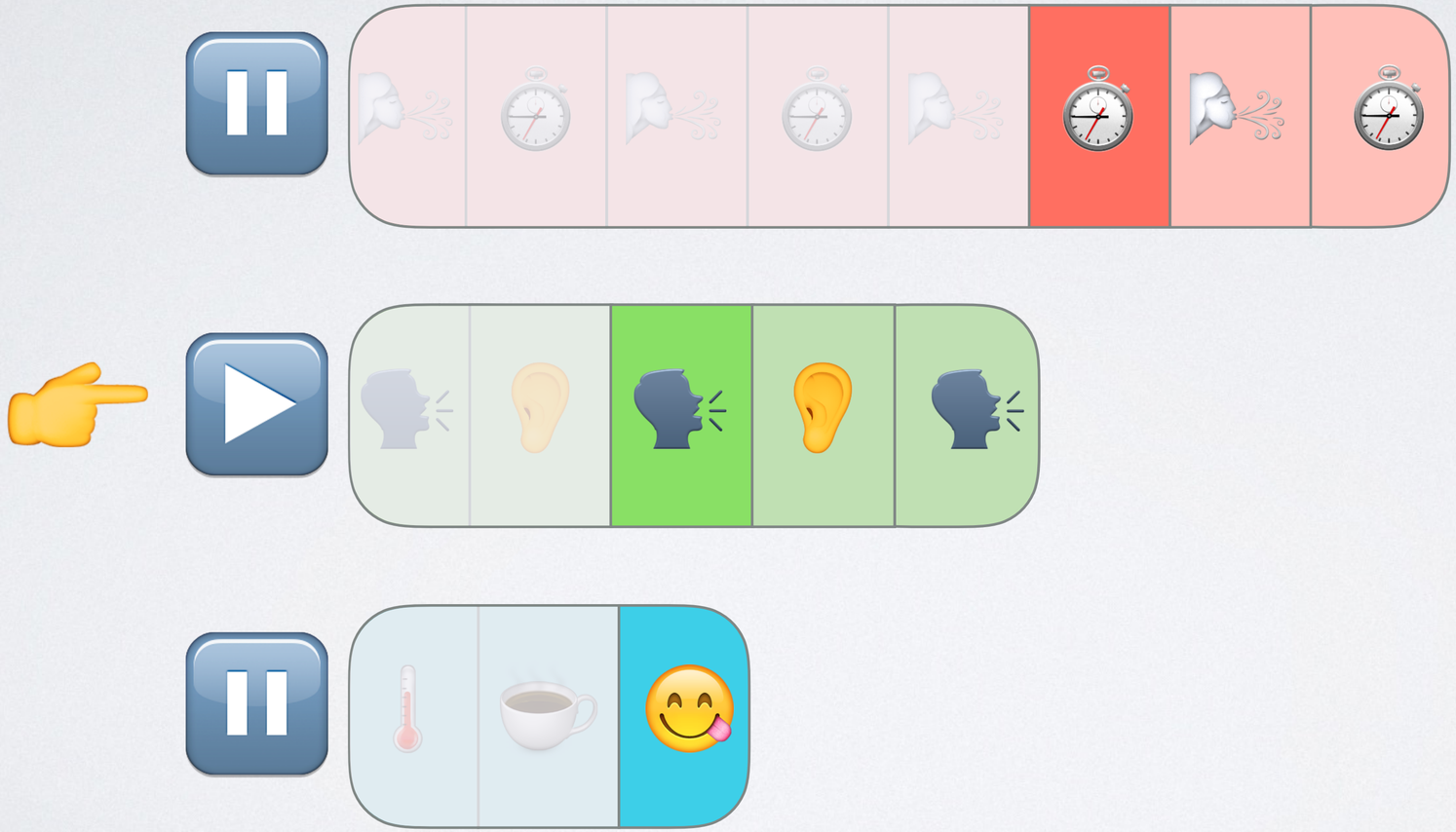


# Event Loop

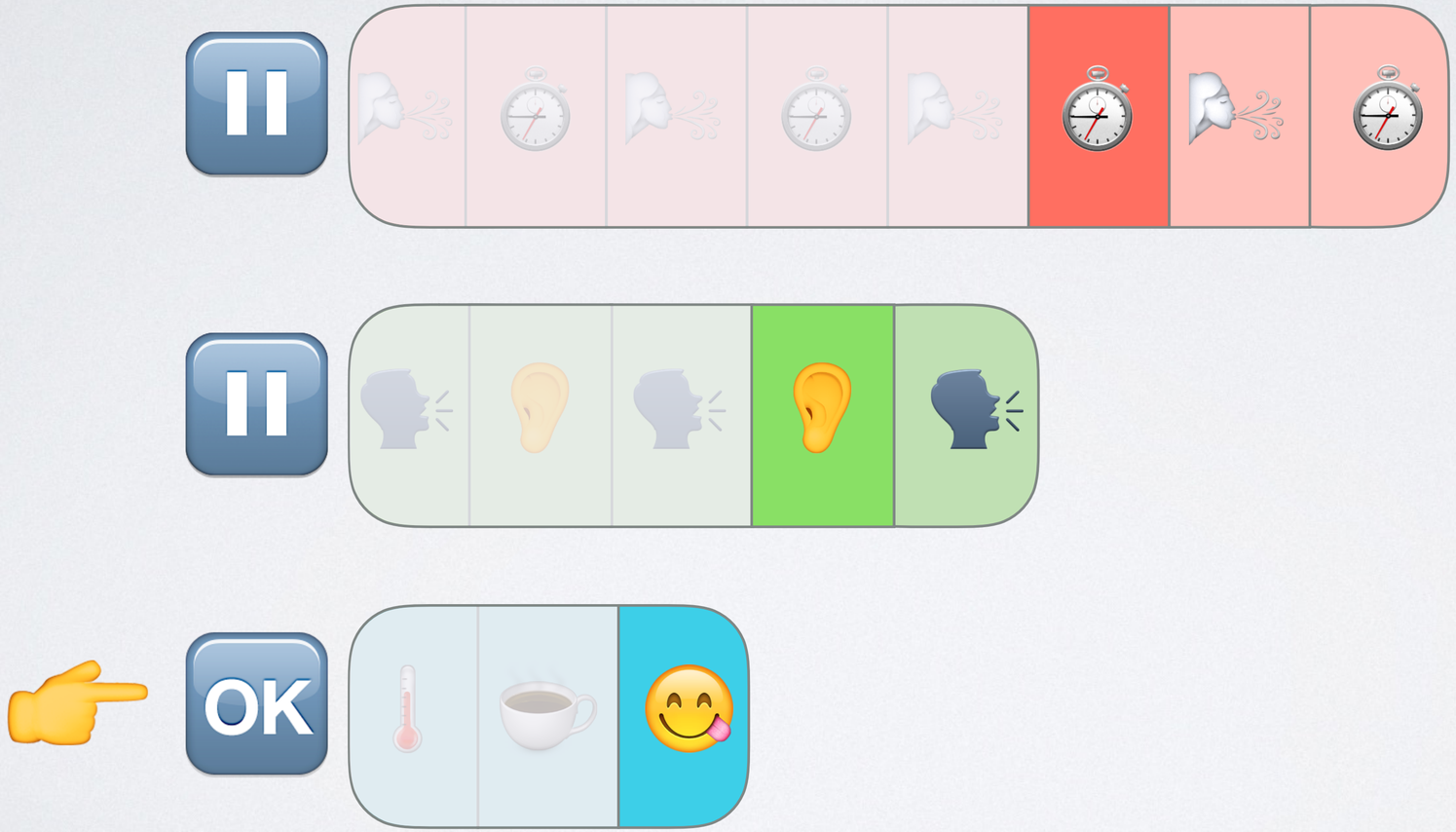


# Event Loop

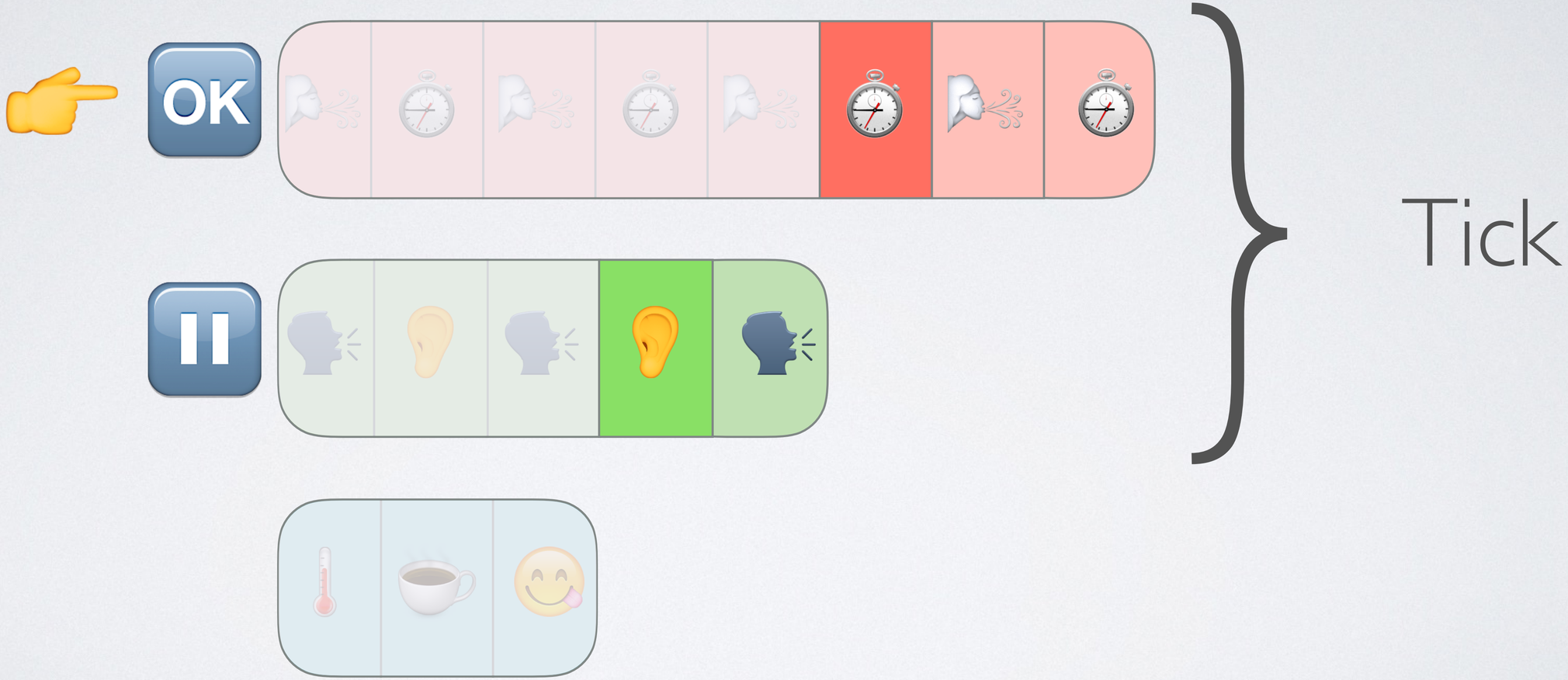
# Event Loop



# Event Loop



# Event Loop



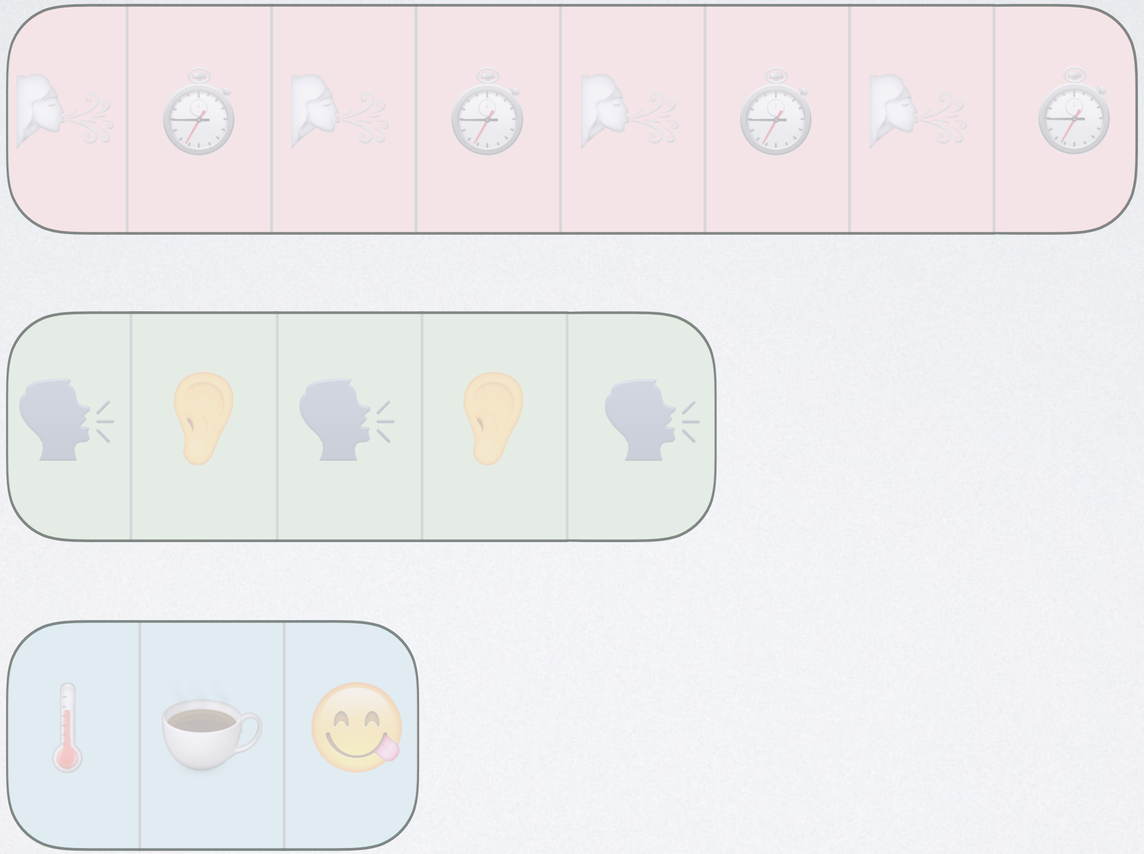
# Event Loop

# Event Loop

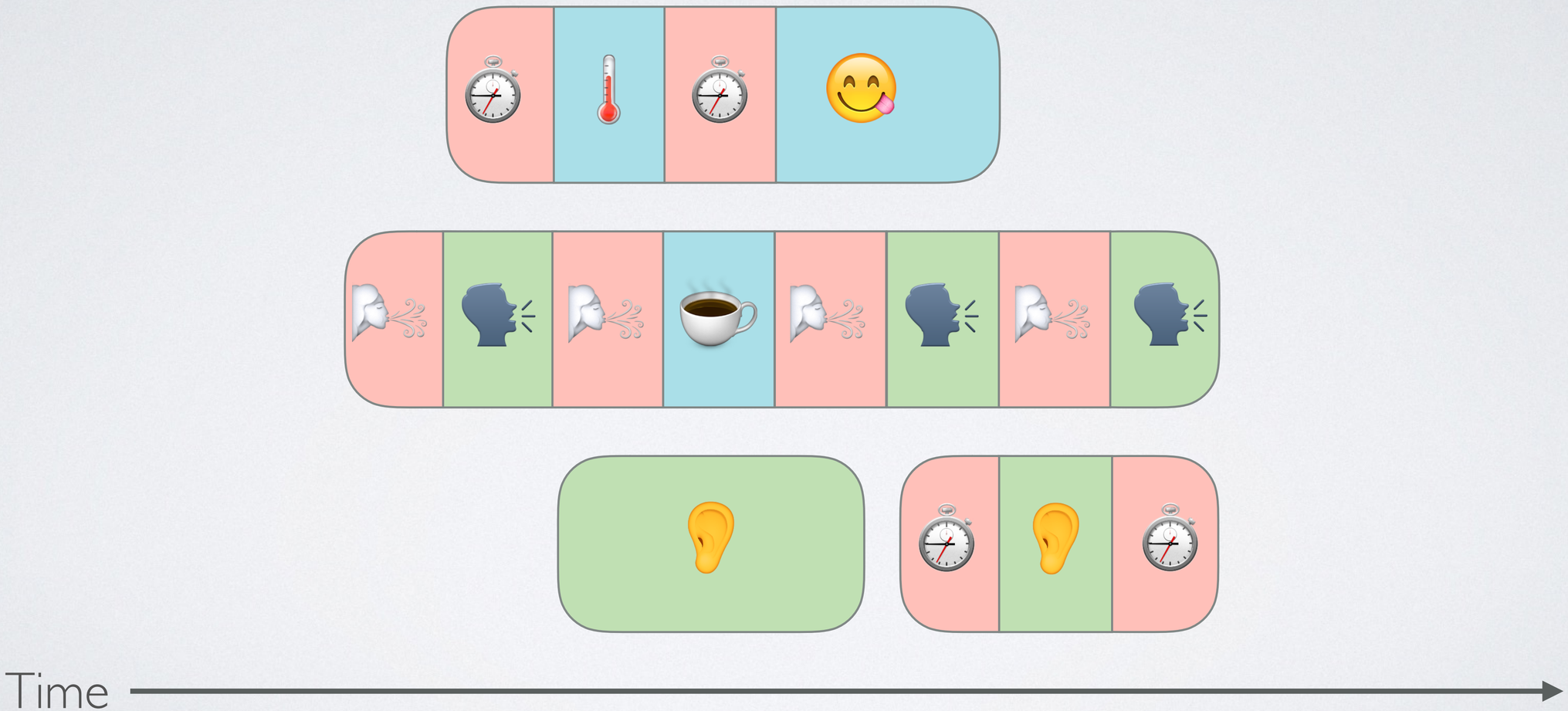
# Event Loop

# Event Loop

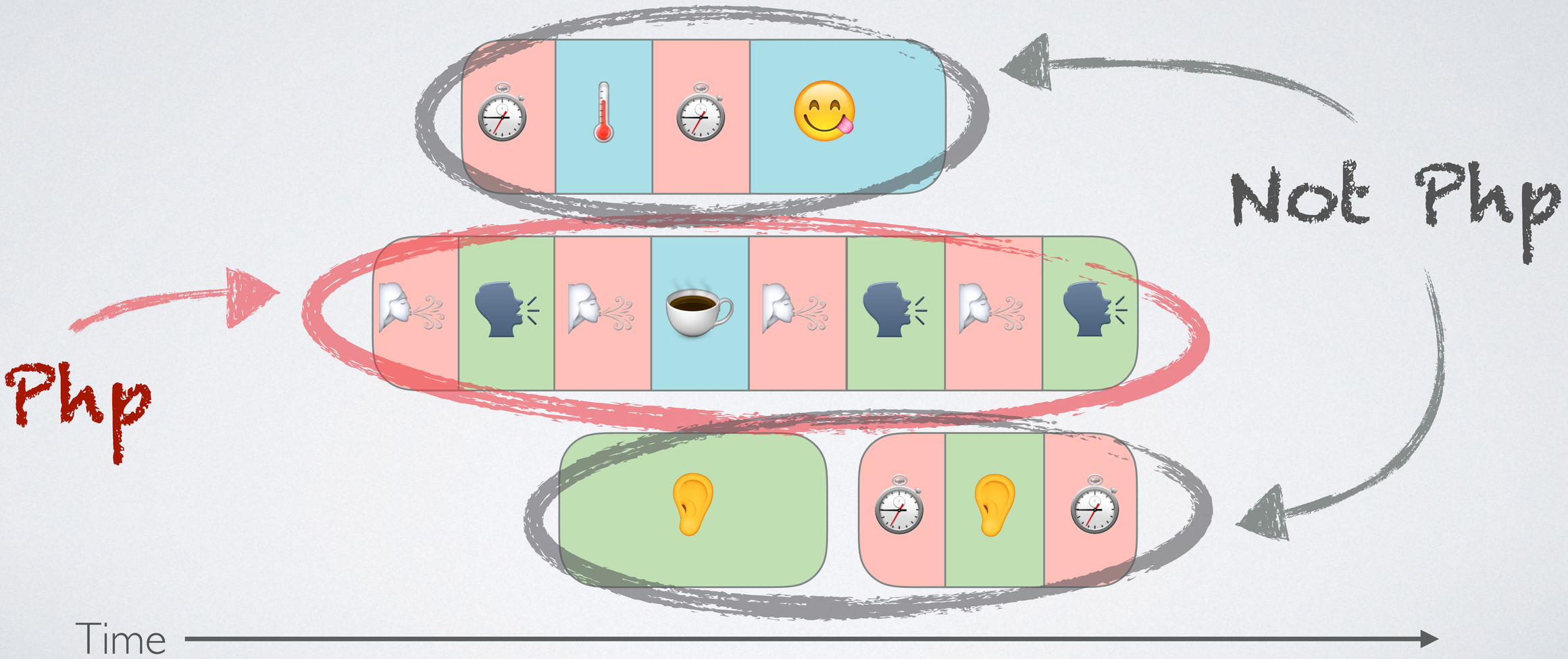
# Event Loop



# Asynchronous



# Asynchronous



# How to dispatch code execution efficiently?

# Generators

Subset of coroutines

# Example

```
function gen_one_to_three() {
    for ($i = 1; $i <= 3; $i++) {
        yield $i;
    }
}

$generator = gen_one_to_three();
foreach ($generator as $value) {
    echo "$value\n";
}
```

# Starting with Generators

… by the weird part…

# Creation

```
// ❌ Does NOT work
$generator = new \Generator;

// PHP Fatal error:  Uncaught Error:
// The "Generator" class is reserved
// for internal use
// and cannot be manually instantiated
```

# Creation

```
function create(): \Generator
{
    yield;
}

$generator1 = create();
$generator2 = create();
```

# Only `yield` matters

```
function emptyGenerator(): \Generator
{
    return 1;
    yield; // Never reached…
}

$generator = emptyGenerator();
```

# Execution start

```
function dyingGenerator(): \Generator
{
    die('hard');
    yield;
}
// The function is not executed…
$generator = dyingGenerator();
// Will die ‘hard’
$generator->valid();
```

```php
final class Generator implements Iterator {

    function rewind() {}
    function valid(): bool {}
    function current() {}
    function key() {}
    function next() {}

    function send($value) {}
    function throw(Throwable $exception) {}
    function getReturn() {}
}
```

Outside

```php
$generator = createMyGenerator();
// Here, outside content
```

```php
function createMyGenerator():
\Generator
{
    // Here, inside content
}
```

Inside

Outside

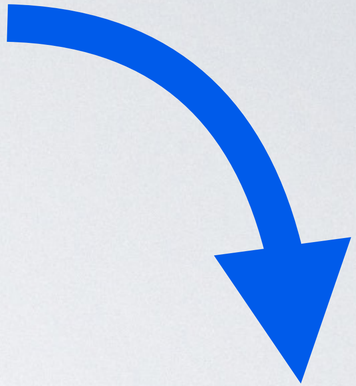
```
$value = $generator->current();
$key = $generator->key();
```

```
yield $key => $value;
```

Inside

Outside

```
$generator->send($value);
$generator->next();
```

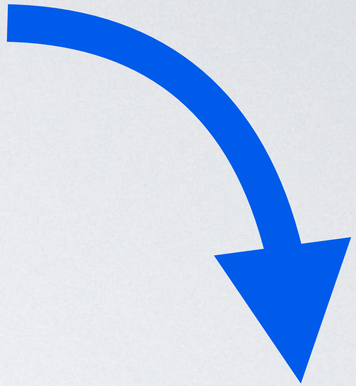


```
$value = yield;
$nullValue = yield;
```

Inside

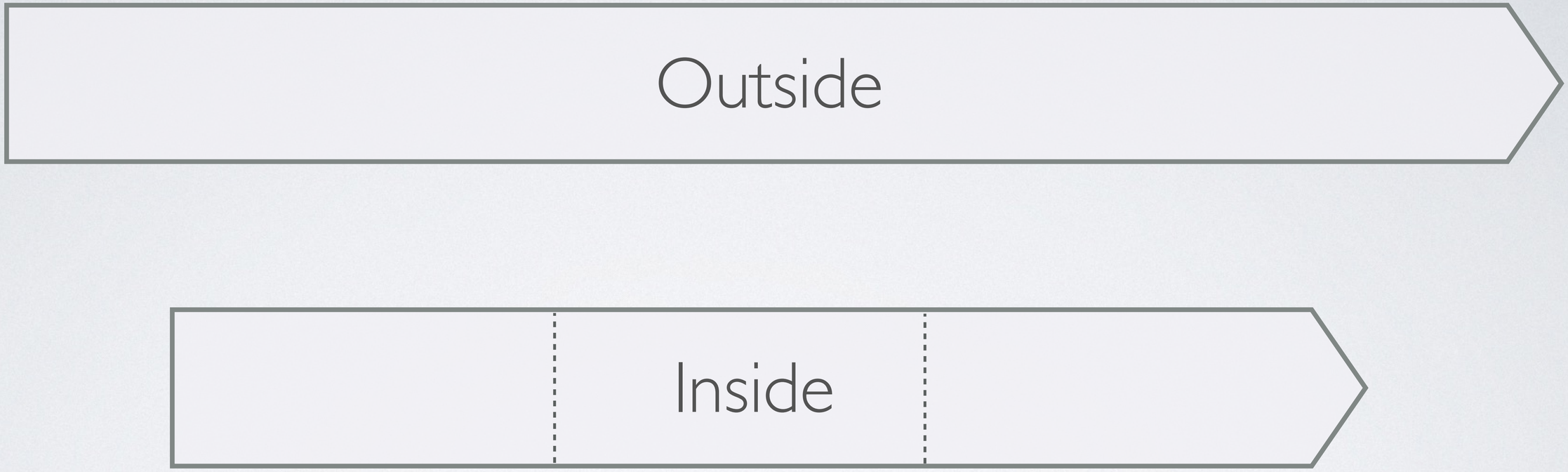
Outside

```
$v1 = $generator->current();
$v3 = $generator->send($v2);
```

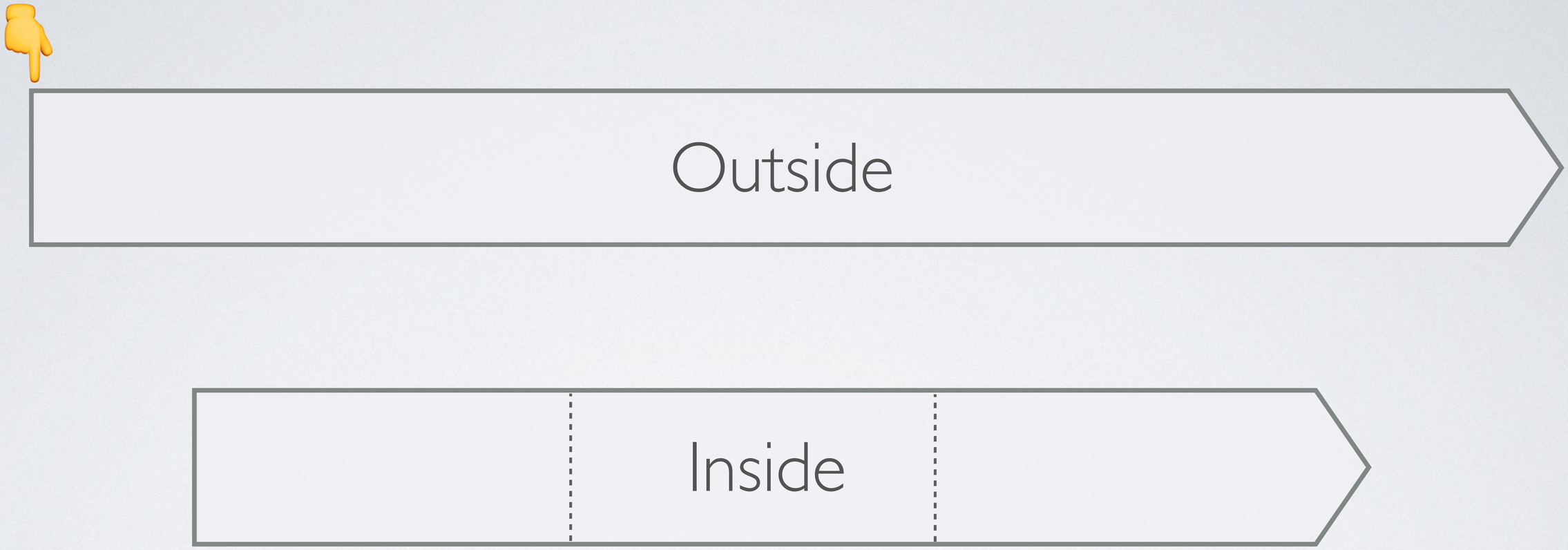


```
$v2 = yield $v1;
yield $v3;
```

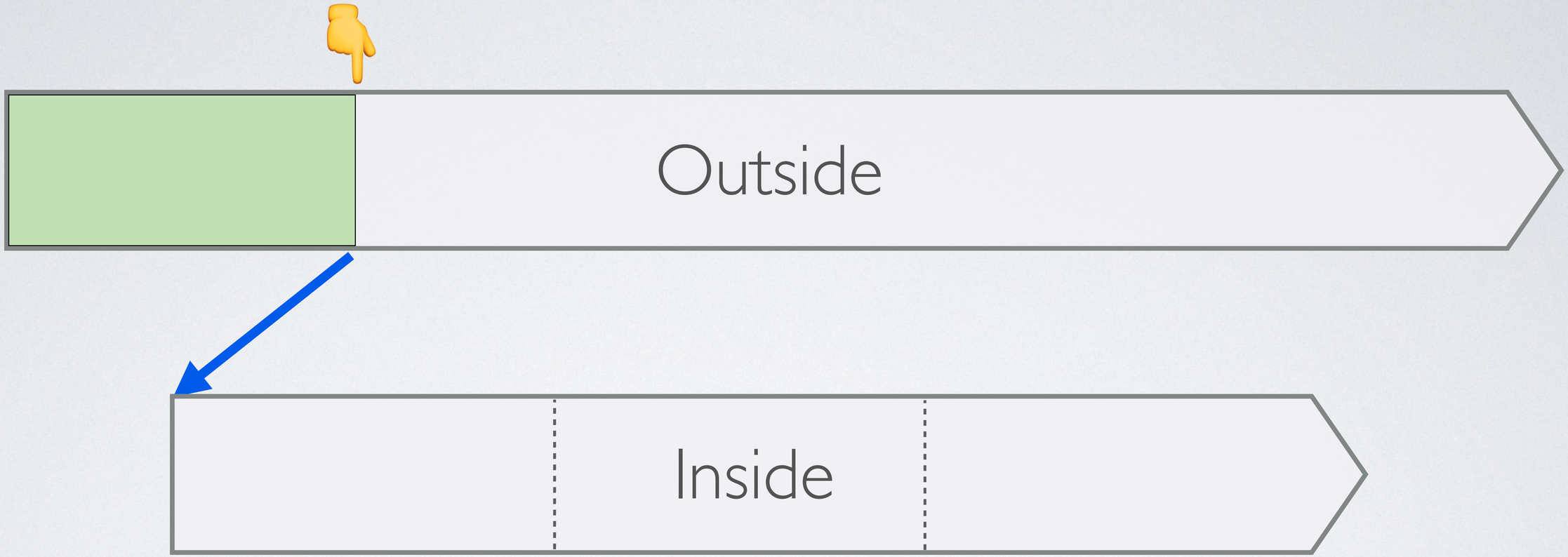
Inside


Outside
Inside

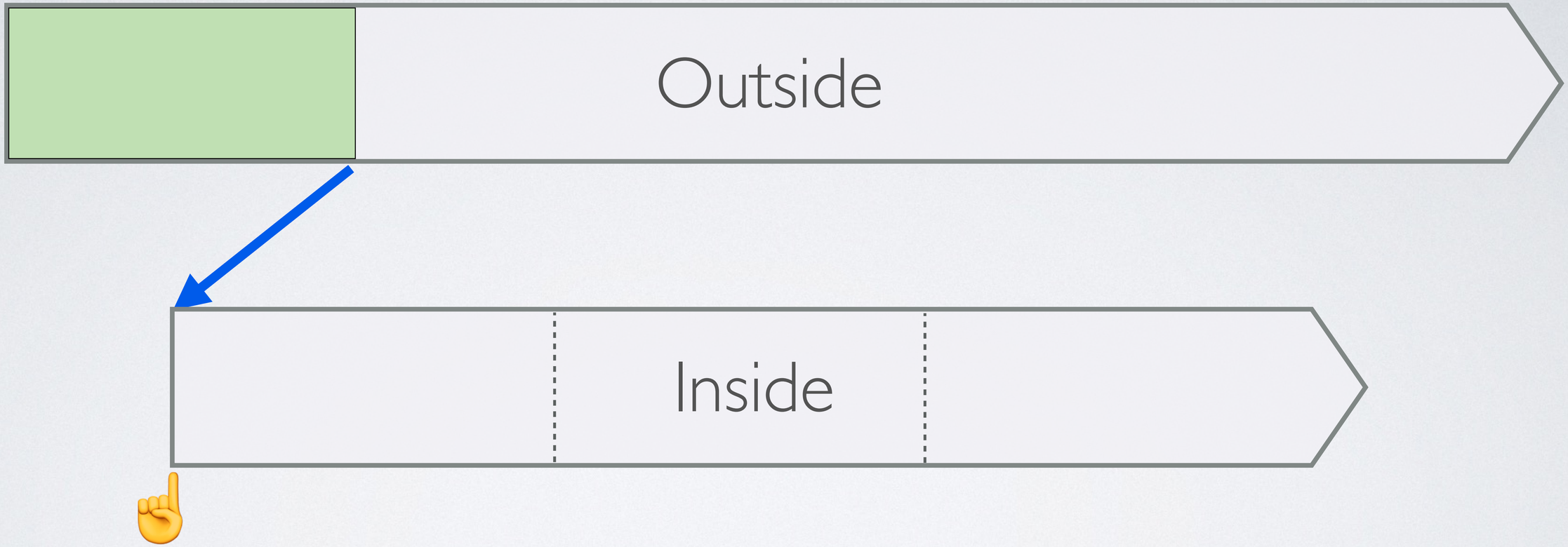
```
$generator = createMyGenerator()
```



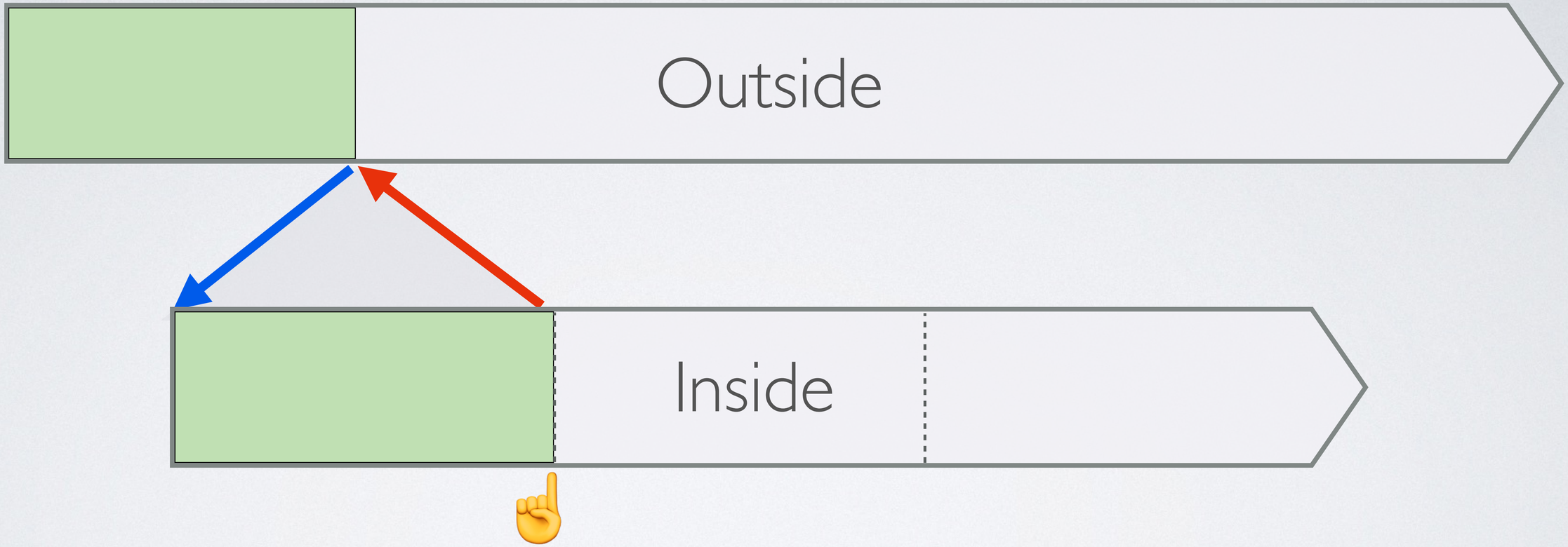
```
$v1 = $generator->current()
```



```
$v1 = $generator->current()
```

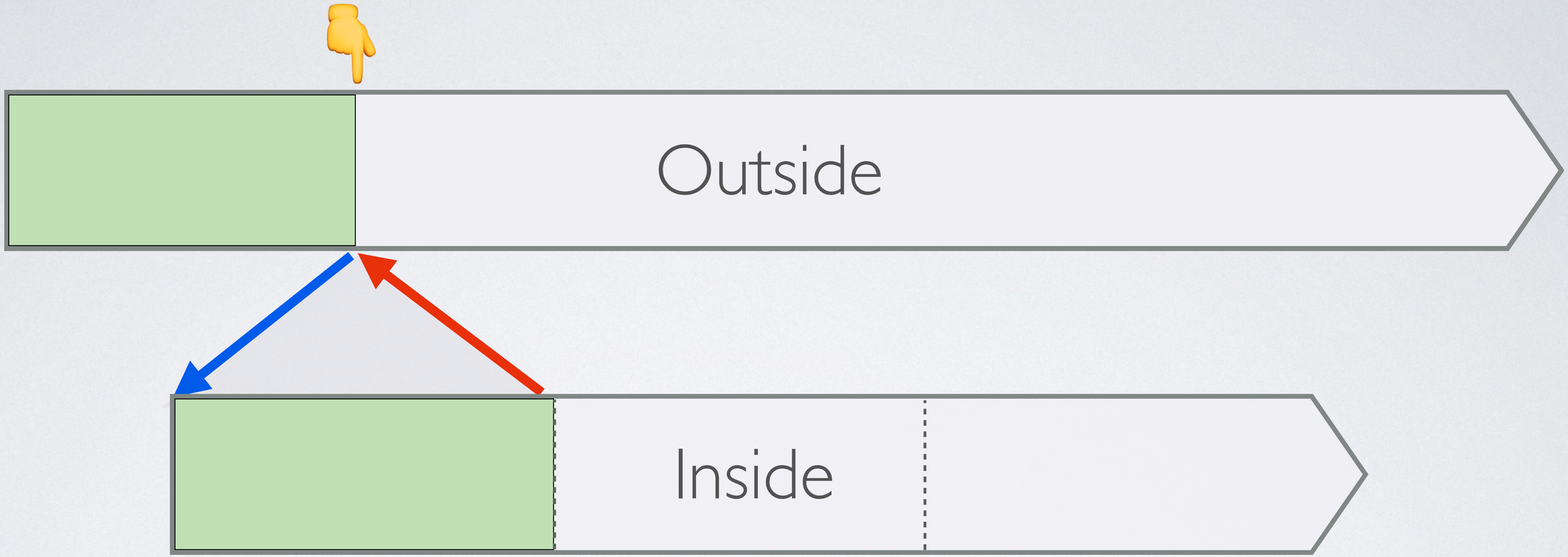


```
$v1 = $generator->current()
```



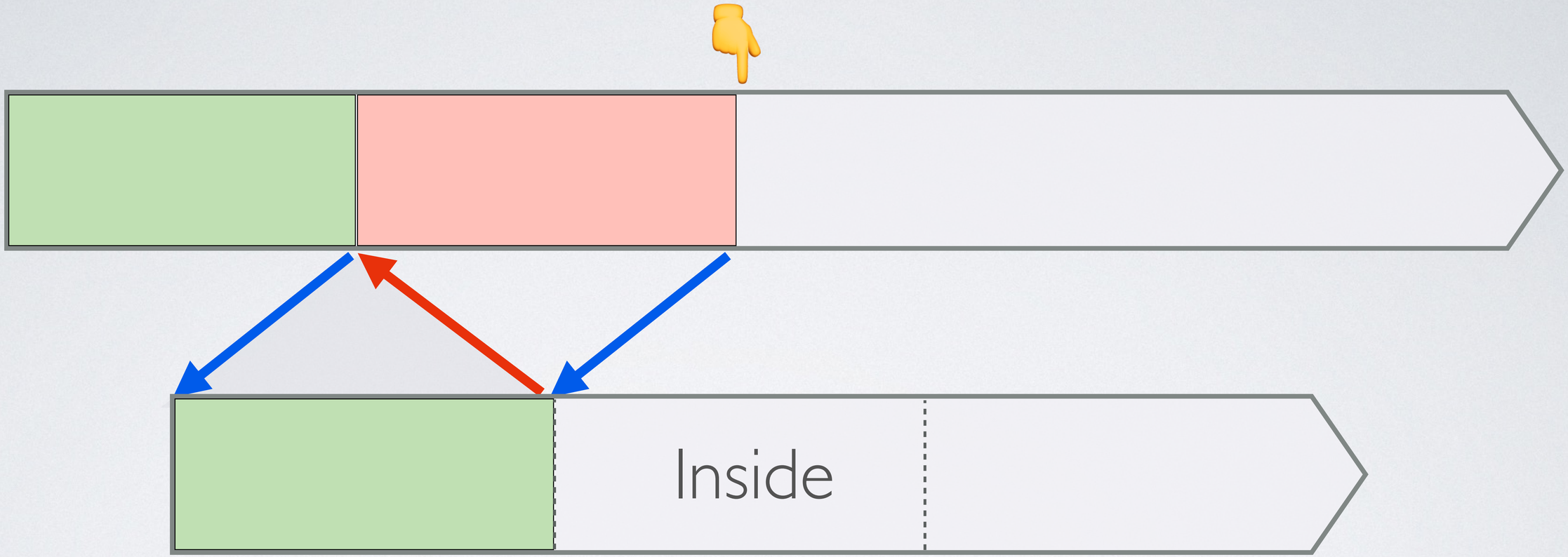

```
$v2 = yield $v1
```

```
$v1 = $generator->current()
```



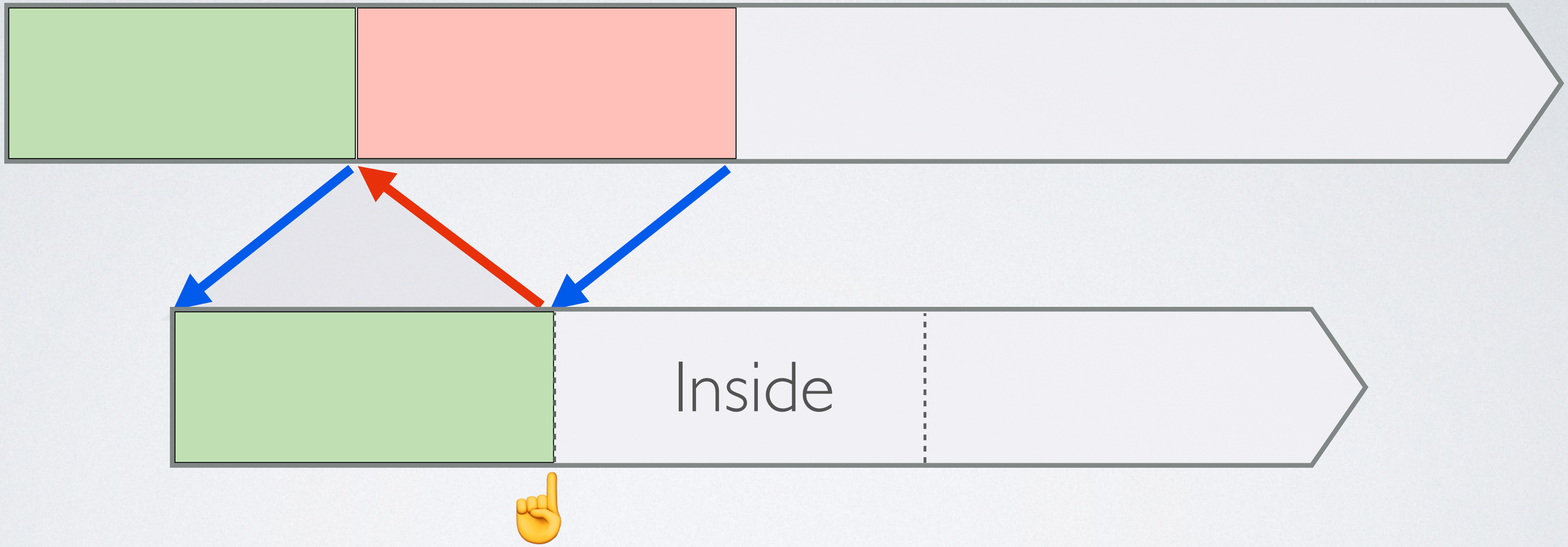

```
$v2 = yield $v1
```

```
$v3 = $generator->send($v2)
```



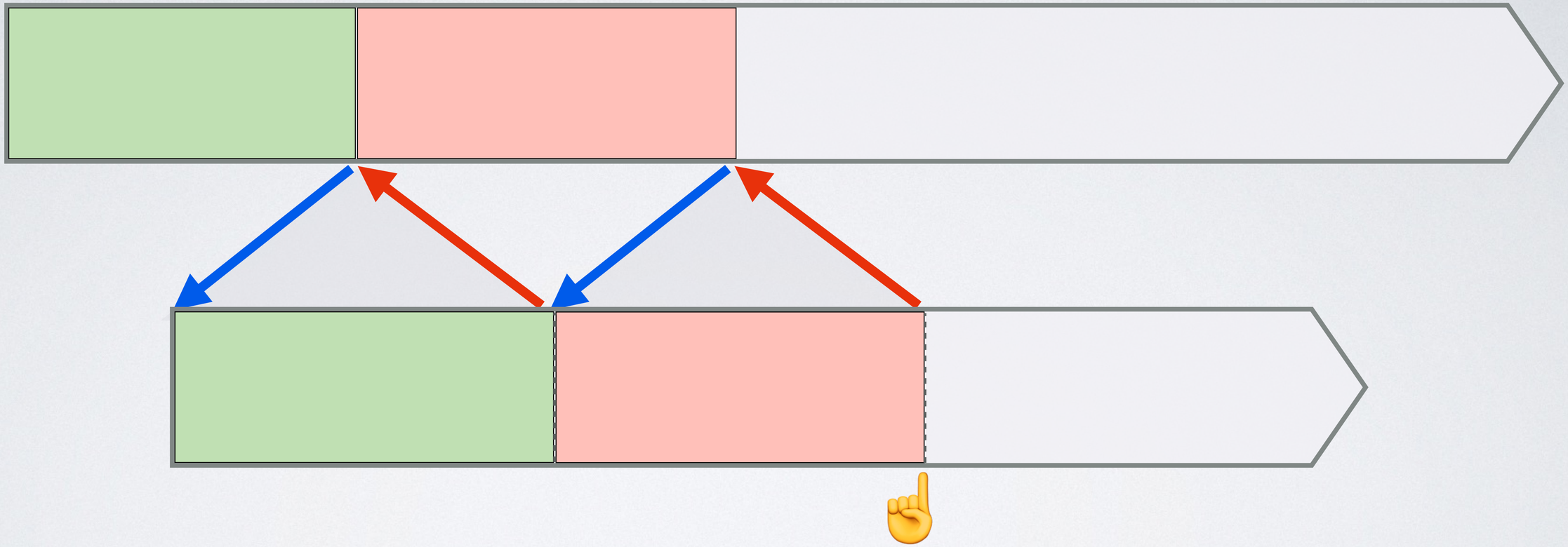

```
$v2 = yield $v1
```

```
$v3 = $generator->send($v2)
```



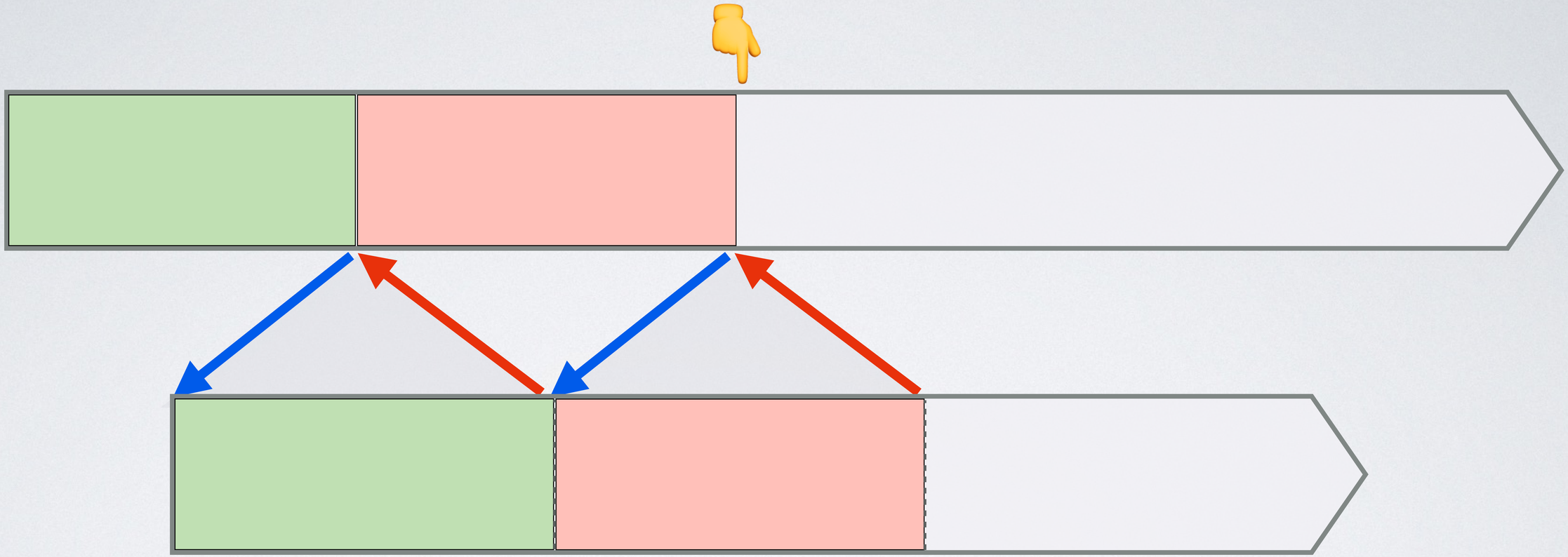

```
$v2 = yield $v1
```

```
$v3 = $generator->send($v2)
```



```
yield $v3
```

```
$v3 = $generator->send($v2)
```



```
yield $v3
```

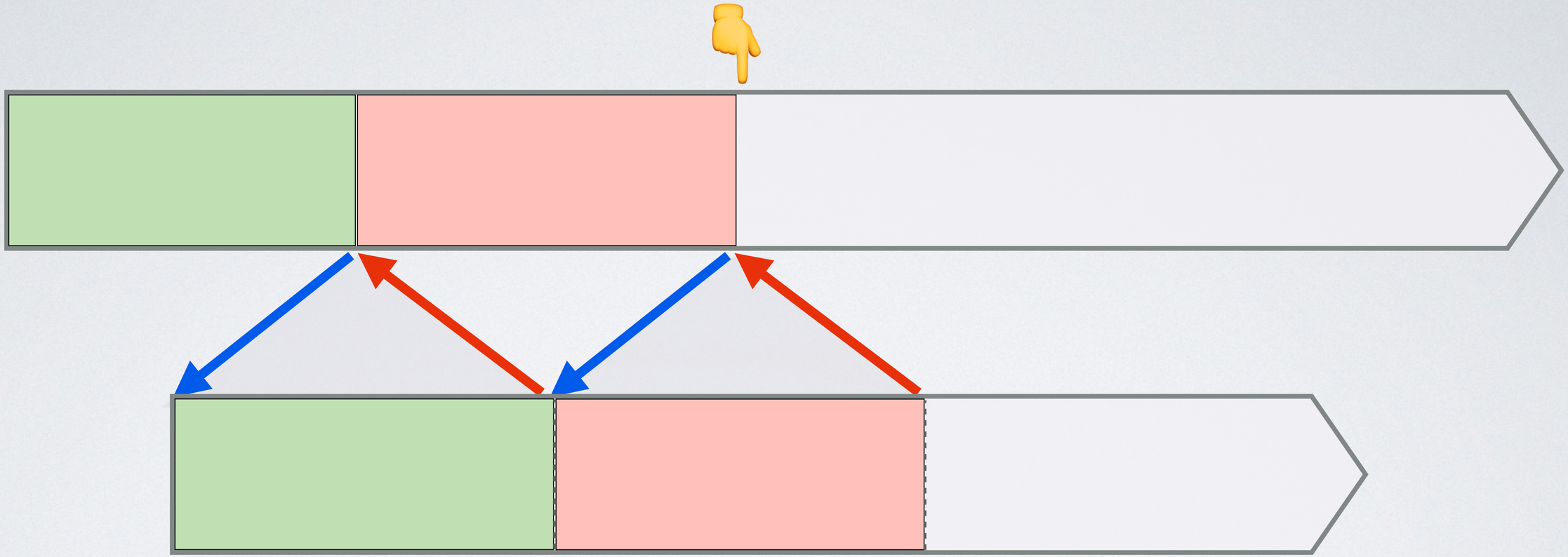
Outside

```
if (!$generator->valid()) {
  $value = $generator->getReturn();
}
```

```
return $value;
```

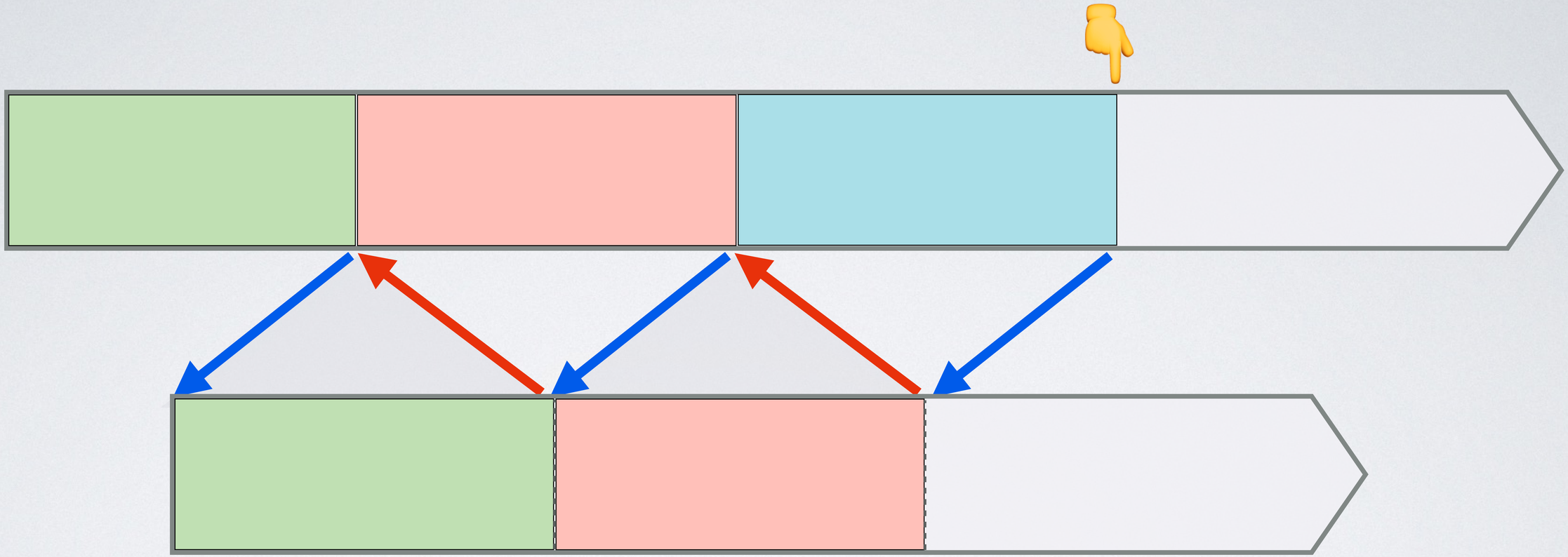
Inside

```
$v3 = $generator->send($v2)
```



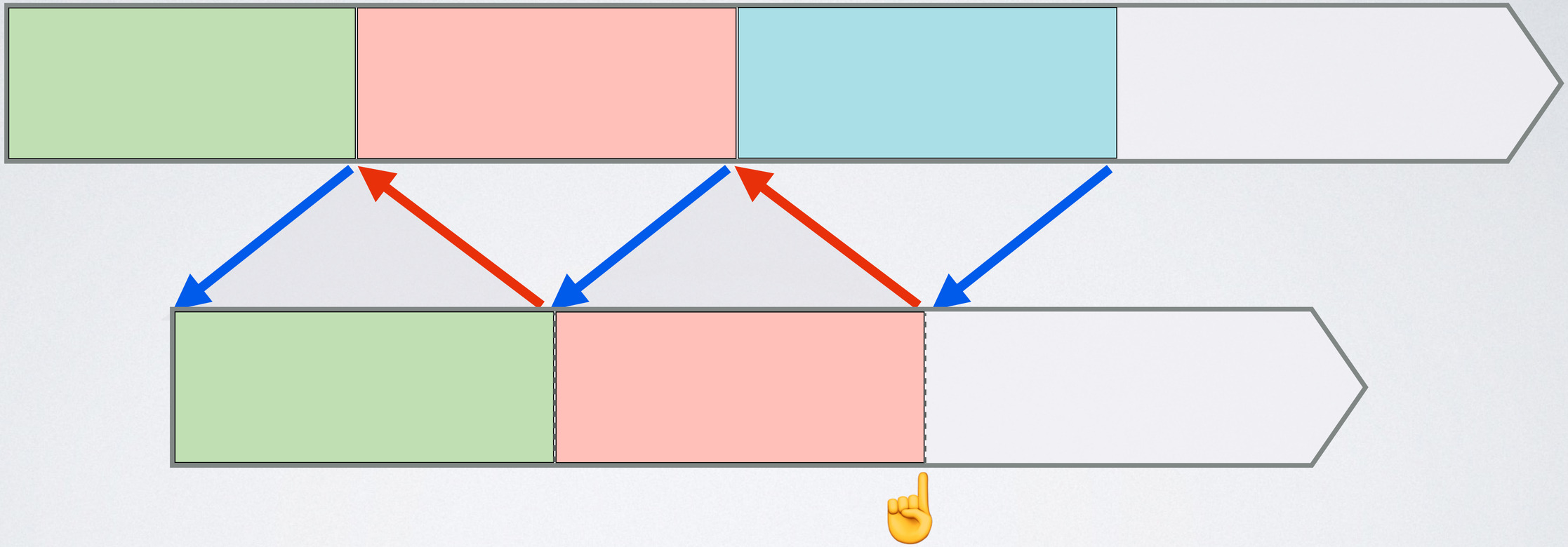
```
yield $v3
```

```
$generator->next()
```



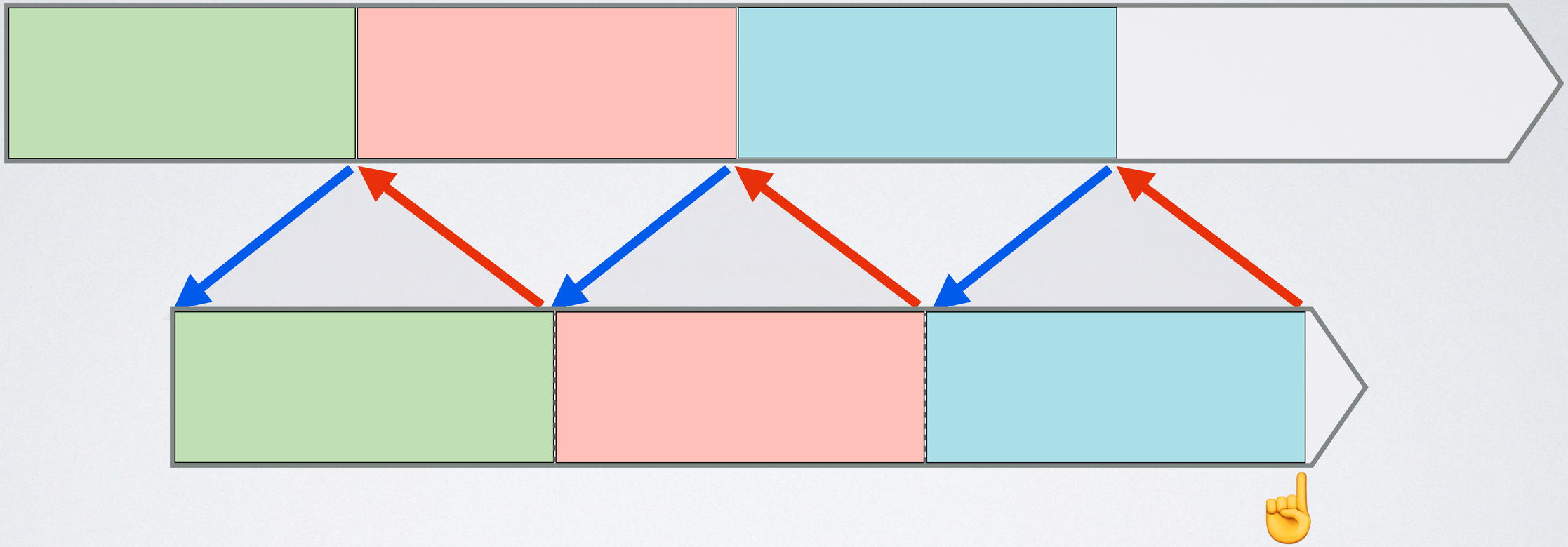
```
yield $v3
```

```
$generator->next()
```



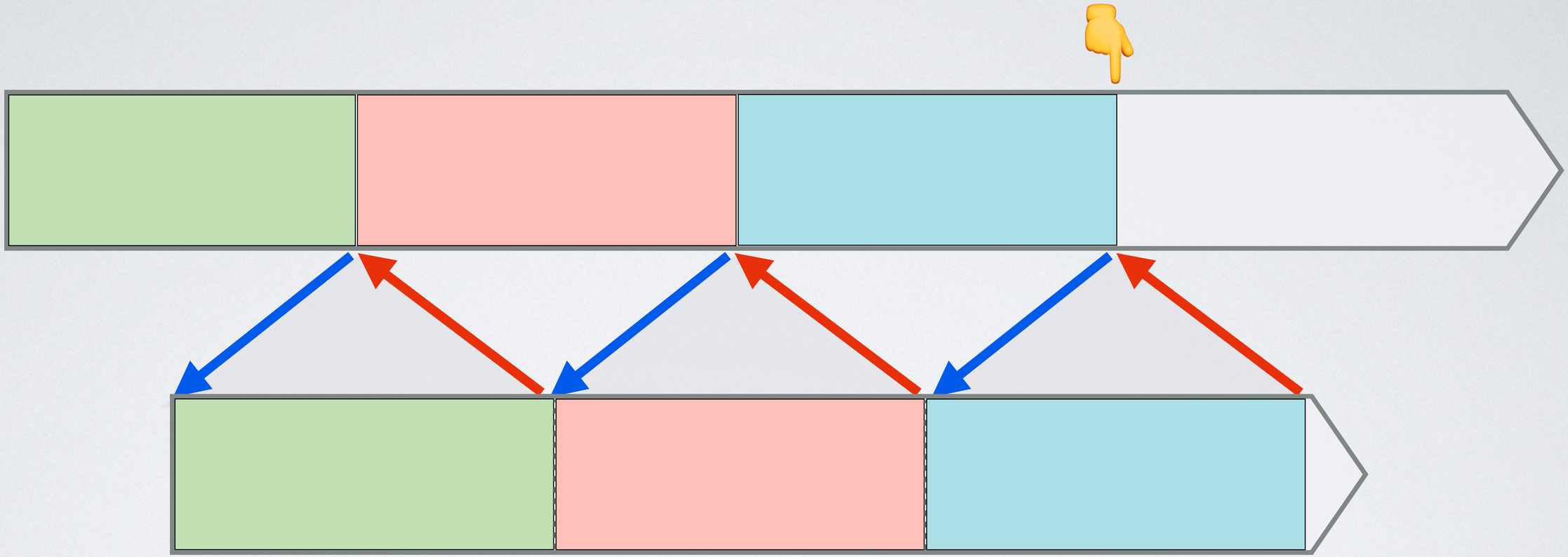
```
yield $v3
```

```
$generator->next()
```



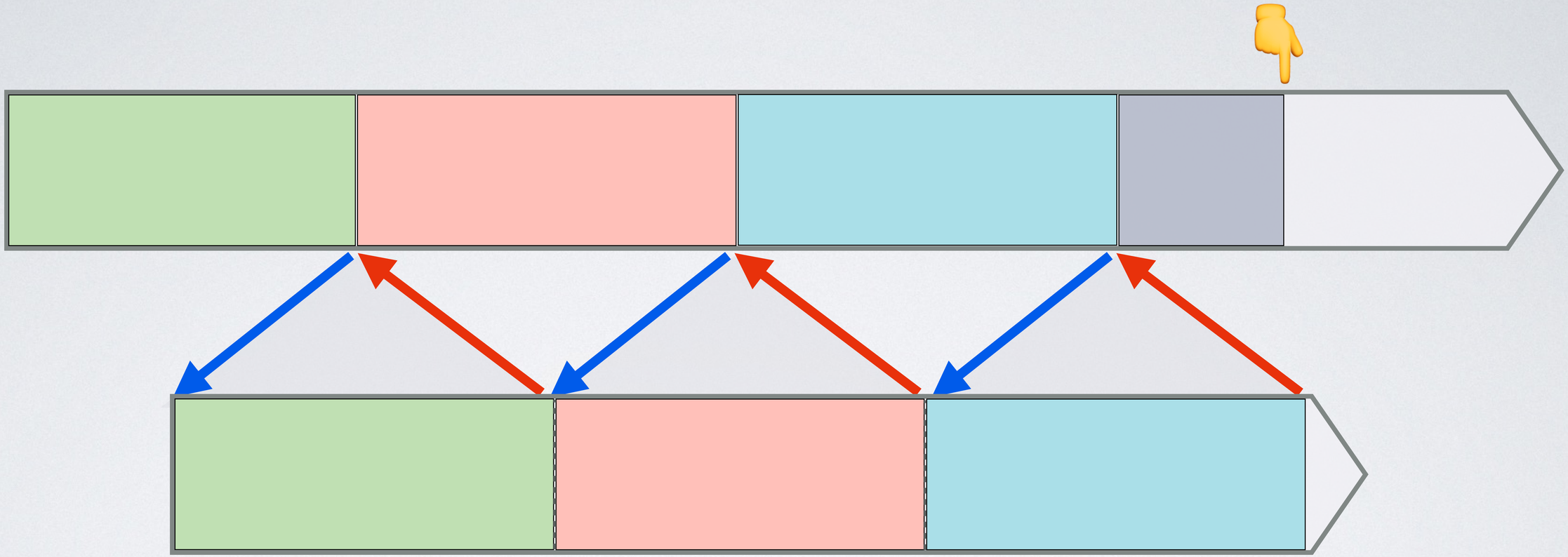
```
return $value;
```

```
$generator->next()
```



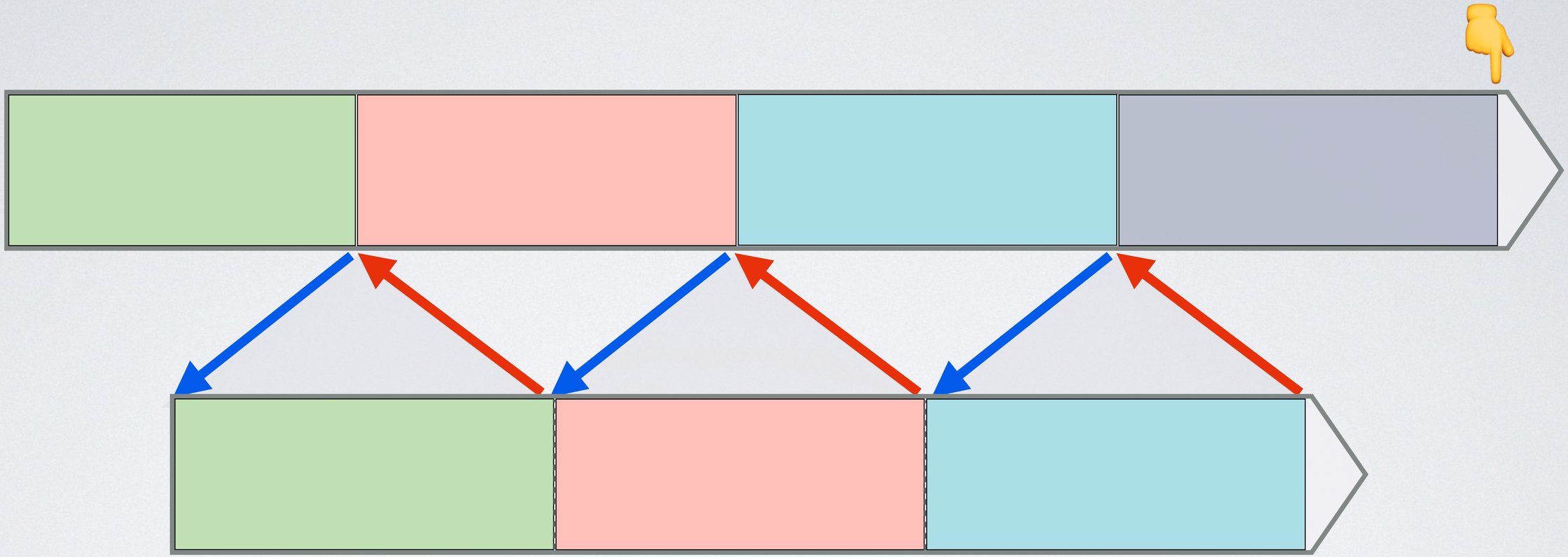
```
return $value;
```

```
!$generator->valid()
```



```
return $value;
```

```
$value = $generator->getReturn()
```



```
return $value;
```

Outside

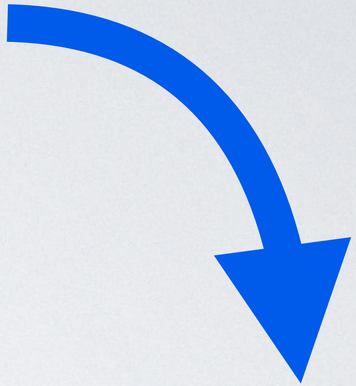
```
try {
    $generator->next();
} catch (\Exception $exception)
{}
```

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

Inside

Outside

```
$generator->throw($exception);
```

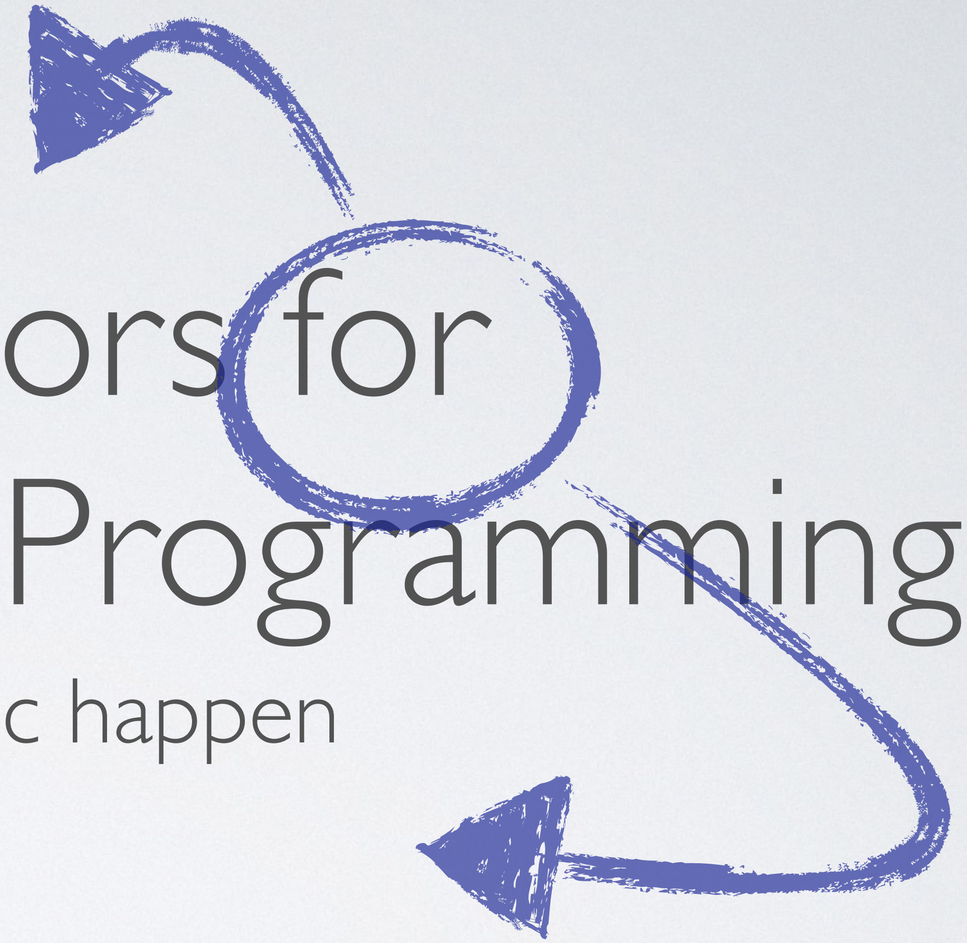


```
try {
    yield;
} catch (\Exception $exception)
{}
```

Inside

# Generators for Asynchronous Programming

Let the magic happen



# Promises

# Synchronous

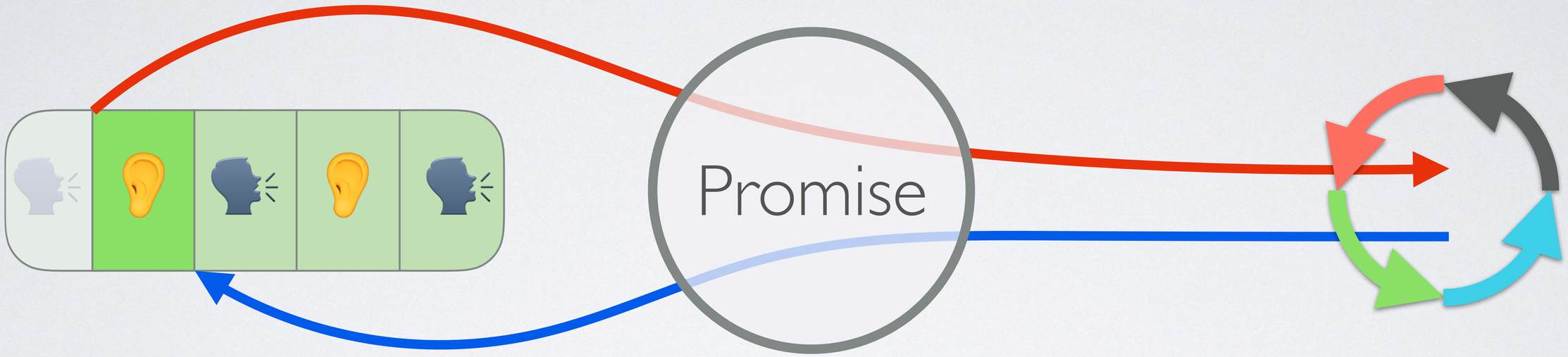
```
function discuss(string $question1, string $question2)
{
    /** @var string $response */
    $response = askQuestion($question1);
    echo $response;

    /** @var string $response */
    $response = askQuestion($question2);
    echo $response;
}
```

# Synchronous

```
function askQuestion(string $question): string
{
    // …
}
```

Waiting for a result…




Here your result!

# Asynchronous

```
function askQuestion(string $question): Promise
{
    // …
}
```

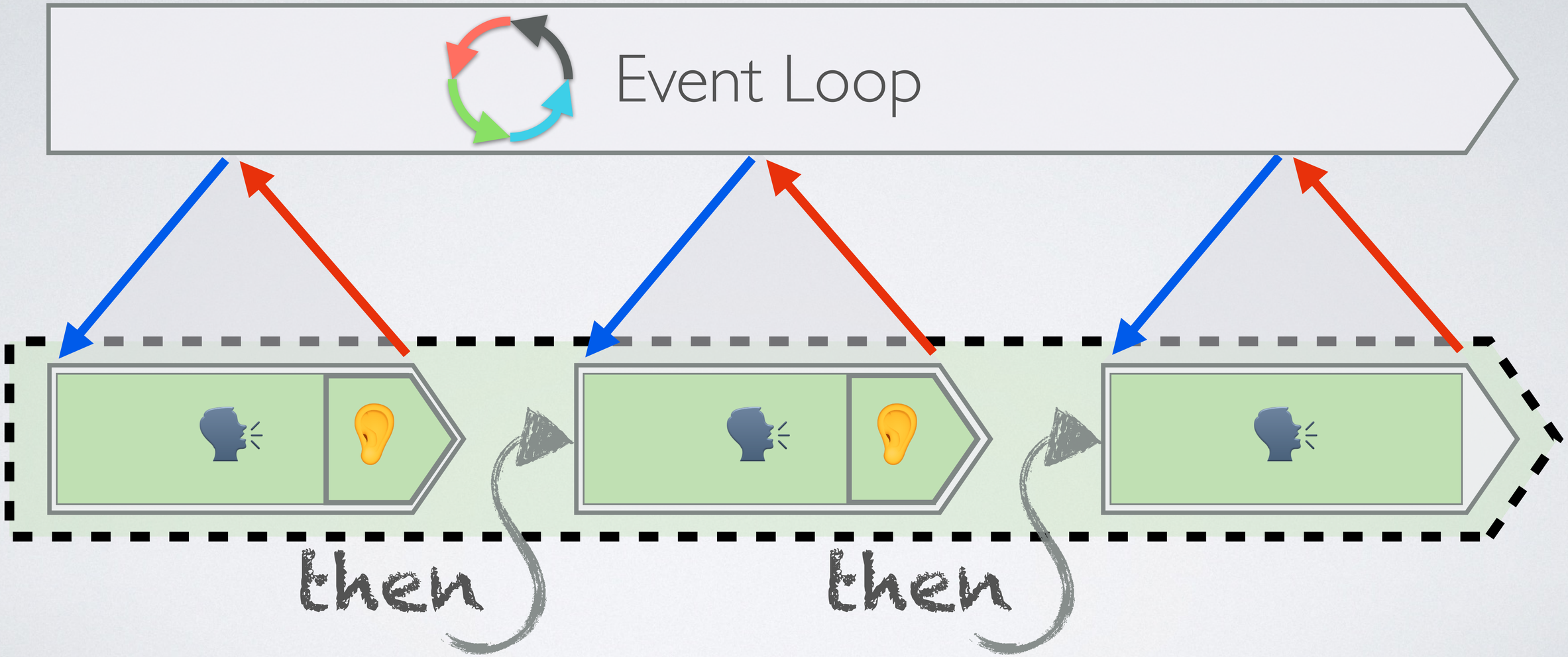
# Asynchronous

```
function discuss(string $question1, string $question2)
{
    /** @var Promise $promise */
    $promise = askQuestion($question1);
    // !? echo $response;

    /** @var Promise $promise */
    $promise = askQuestion($question2);
    // !? echo $response;
}
```

# Promises A+

*Thenable* Promises


Event Loop
then
then

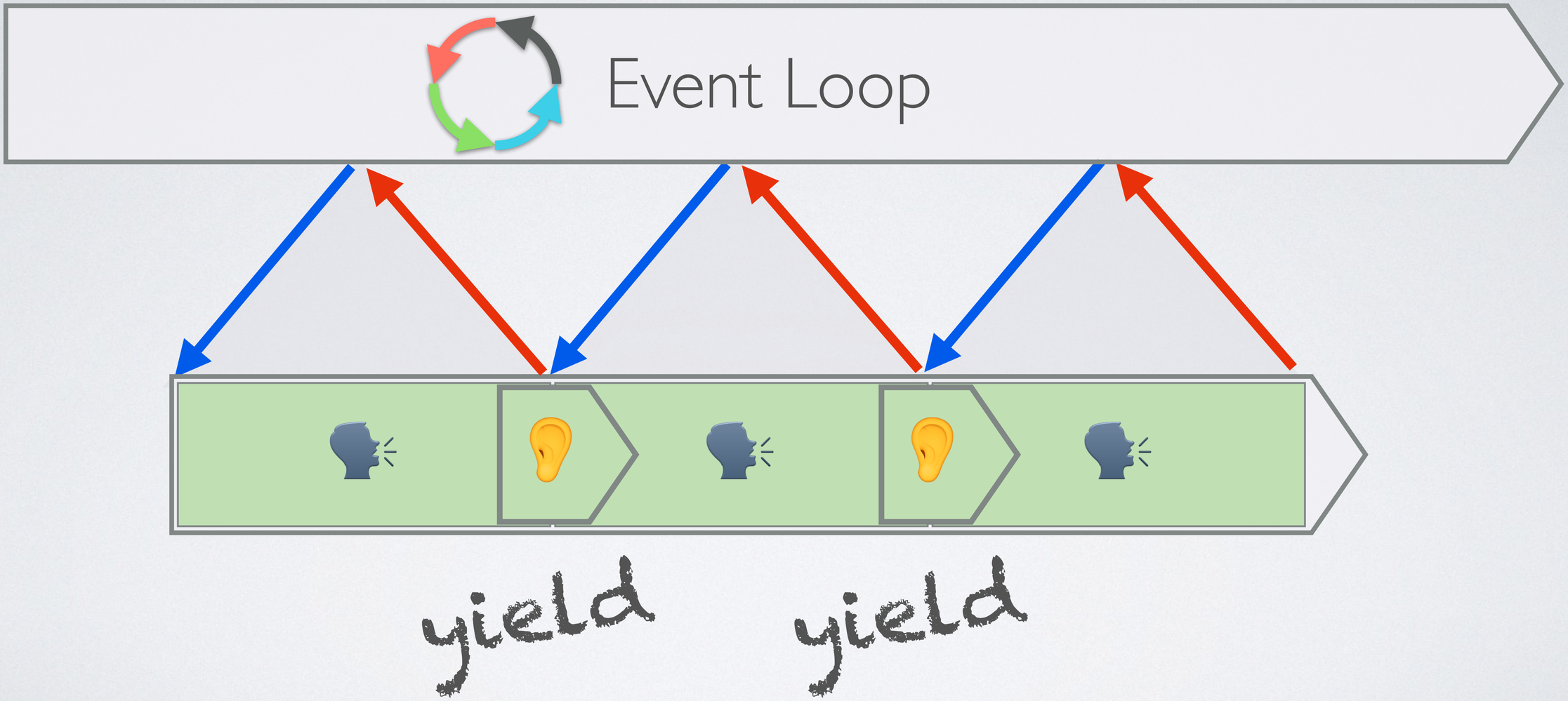
```php
function discuss(string $question1, string $question2)
{
    $promise = askQuestion($question1);
    $promise->then(
        function(string $response) use($question2) {
            echo $response;

            return askQuestion($question2);
        }
    )->then(
        function(string $response) {
            echo $response;
        }
    );
}
```

# 🔥Callback Hell🔥

```
$promise
    ->then(function ($x) {
        return $x + 1;
    })
    ->then(function ($x) {
        throw new \Exception($x + 1);
    })
    ->otherwise(function (\Exception $x) {
        return $x->getMessage() + 1;
    })
    ->then(function ($x) {
        echo 'Mixed '.$x;
    });
```

# *Generator* Promises


Event Loop
yield
yield

# Synchronous

```
function discuss(string $question1, string $question2)
{

    /** @var string $response */
    $response = askQuestion($question1);
    echo $response;


    /** @var string $response */
    $response = askQuestion($question2);
    echo $response;
}
```

# Asynchronous

```
function discuss(string $question1, string $question2)
{

    /** @var string $response */
    $response = yield askQuestion($question1);
    echo $response;


    /** @var string $response */
    $response = yield askQuestion($question2);
    echo $response;
}
```

# Asynchronous

```
function discuss(string $question1, string $question2)
{
    $promise = askQuestion($question1);
    /** @var string $response */
    $response = yield $promise;
    echo $response;

    $promise = askQuestion($question2);
    /** @var string $response */
    $response = yield $promise;
    echo $response;
}
```

To asynchronously *wait* the value of a **promise**, use **`yield`**.

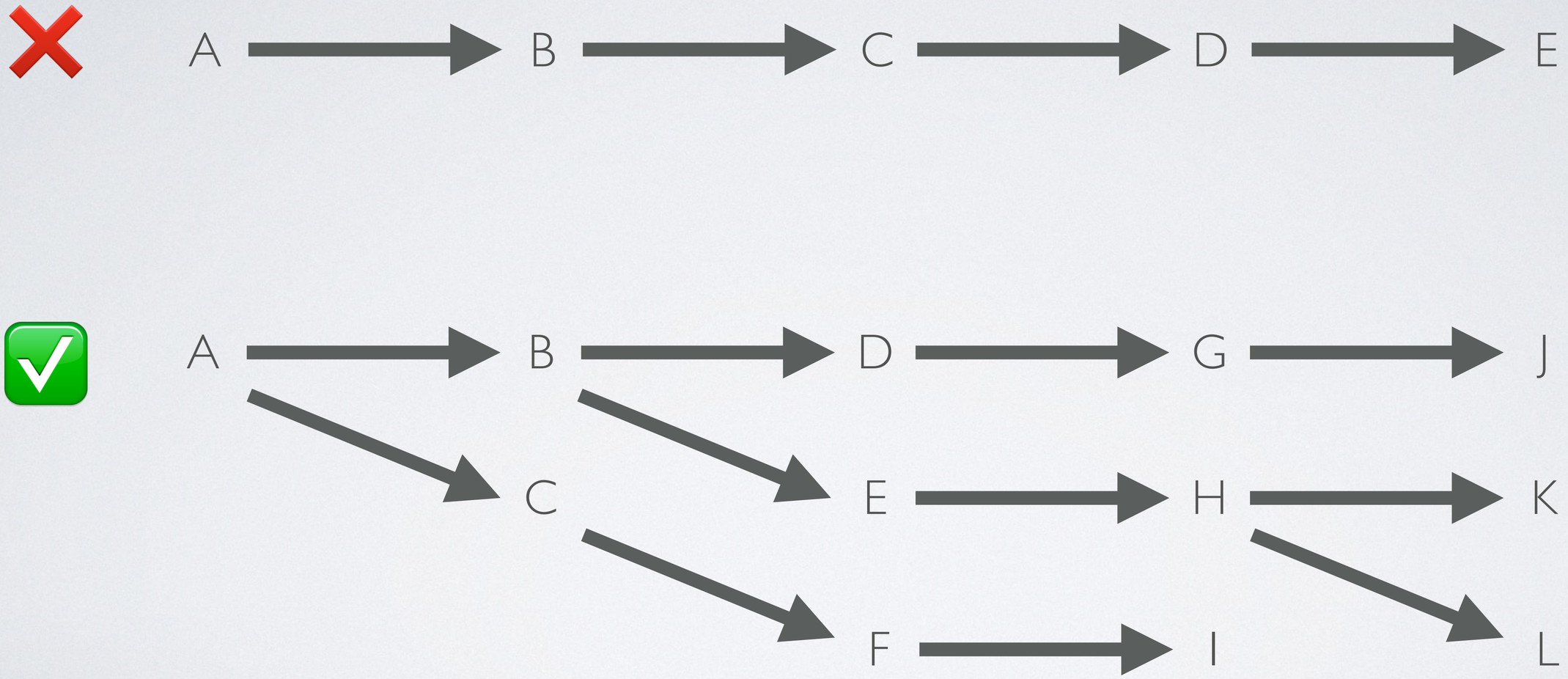
**Asynchronous Function:**

a **generator**

yielding only *promises*.

# Getting started!

# Finding the good use case


A
B
C
D
E
A
B
D
G
J
C
E
H
K
F
I
L

# Choose your Event Loop

# reactphp/react

👍 Community

😩 *Thenable* Promises

# amphp/amp

👍 Generators

🤔 Coupling ?

# m6web/tornado

- 👍 Generators
- 😘 Interfaces / Adapters
- 🤔 M6Web

# Tea for Two

Let’s code it!

# Discussing with a friend

```
function discuss(string $question1, string $question2)
: \Generator
{
    /** @var string $response */
    $response = yield askQuestion($question1);
    echo $response;

    /** @var string $response */
    $response = yield askQuestion($question2);
    echo $response;
}
```

# Breathing

```
function breath(EventLoop $eventLoop, int $count)
: \Generator
{
    while (--$count >= 0) {
        echo "Breathing\n";
        yield $eventLoop->idle();
    }
}
```

# Drinking Tea

```
function tMax(EventLoop $eventLoop, int $maximum)
: \Generator
{
    do {
        yield $eventLoop->idle();
        $current = rand(20, 40);
    } while ($current > $maximum);

    return $current;
}
```

# Drinking Tea

```
function drink(EventLoop $eventLoop): \Generator
{
    $generator = tMax($eventLoop, 35);
    $promise = $eventLoop->async($generator);
    $temperature = yield $promise;

    echo "Drinking [$temperature °]\n";

    yield $eventLoop->delay(1000/*ms*/);
}
```

# Drinking Tea

```
function drink(EventLoop $eventLoop): \Generator
{
    $generator = tMax($eventLoop, 35);
    $promise = $eventLoop->async($generator);
    $temperature = yield $promise;

    echo "Drinking [$temperature °]\n";

    yield $eventLoop->delay(1000/*ms*/);
}
```

# Drinking Tea

```
function drink(EventLoop $eventLoop): \Generator
{
    $generator = tMax($eventLoop, 35);
    $promise = $eventLoop->async($generator);
    $temperature = yield $promise;

    echo "Drinking [$temperature °]\n";

    yield $eventLoop->delay(1000/*ms*/);
}
```

# Drinking Tea

```
function drink(EventLoop $eventLoop): \Generator
{
    $generator = tMax($eventLoop, 35);
    $promise = $eventLoop->async($generator);
    $temperature = yield $promise;

    echo "Drinking [$temperature °]\n";

    yield $eventLoop->delay(1000/*ms*/);
}
```

# Wait for it…

```
$concurrentPromise = $eventLoop->promiseAll(

    $eventLoop->async( breath($eventLoop, 10)      ),
    $eventLoop->async( discuss('Where?', ‘What?') ),
    $eventLoop->async( drink($eventLoop)           )

);

$eventLoop->wait($concurrentPromise);
```

# Wait for it…

```
$concurrentPromise = $eventLoop->promiseAll(

    $eventLoop->async( breath($eventLoop, 10)      ),
    $eventLoop->async( discuss('Where?', ‘What?') ),
    $eventLoop->async( drink($eventLoop)           )

);

$eventLoop->wait($concurrentPromise);
```

# Wait for it…

```
$concurrentPromise = $eventLoop->promiseAll(

    $eventLoop->async( breath($eventLoop, 10)      ),
    $eventLoop->async( discuss('Where?', ‘What?') ),
    $eventLoop->async( drink($eventLoop)           )

);

$eventLoop->wait($concurrentPromise);
```

# So…

Asynchronous Programming

**doesn't** require…

# ~~Http Server~~

# ~~Threads~~

JavaScript™

# Asynchronous Programming

**requires**…

# Event Loop

# Interruptible Tasks

# **Generators**:
# Powerful tool for Asynchronous Programming

# PRACTICE

# Generators for Asynchronous Programming

User Manual

Benoit Viguier
@b_viguier

6M WEB

FORUMPHP PARIS2018

# Thanks !

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

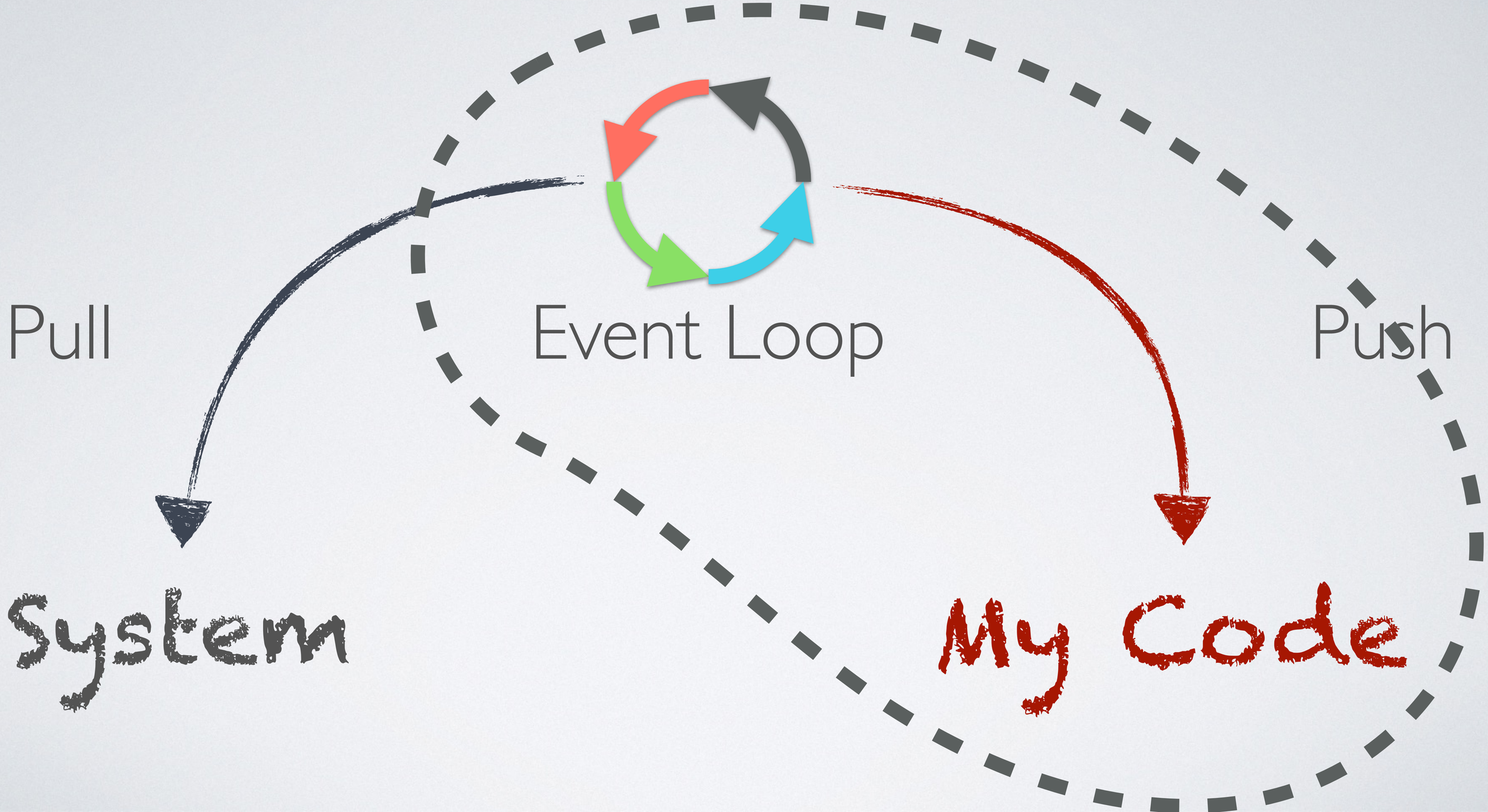
Benoit Viguier
@b_viguier

6M WEB
FORUMPHP PARIS2018

# Hidden Slides

Not enough time…

Pull
Event Loop
Push
System
My Code


Pull
Event Loop
Push
System
My Code

# Interruptible tasks

# Anonymous function trap

```
// Do not forget to execute
// your anonymous function!
$generator = (function (): \Generator
{
    yield;
})();
```

# Type hinting 😭

```
function before(): int {
    return 10;
}

function after(): \Generator {
    $mixed = yield $mixed;
    return 10;
}
```

# Example

```
function fibonacci(array $buffer, int $count) {
    while($count--) {
        $buffer[] = yield array_sum($buffer);
        array_shift($buffer);
    }
}

$fibonacci = fibonacci([0, 1], 10);

while($fibonacci->valid()) {
    echo $fibonacci->current() . PHP_EOL;
    $fibonacci->send($fibonacci->current());
}
```

Outside

```
$v1 = $generator->current();
$generator->next();
$v2 = $generator->current();
```

```
yield from [$v1, $v2];
```

Inside

Asynchronous Php

$\neq$

Http Server written in Php

# Http Call**s**

# An Event Loop in your Controller

```
class MyController
{
    //...

    public function myAction($httpInput)
    {
        $eventLoop = new EventLoop();
        $promise = $this->myAsynchronousService->handleAction(
            $eventLoop,
            $httpInput
        );
        $result = $eventLoop->wait($promise);

        return $result;
    }
}
```

# Stats

# Use Cases

# Asynchronous Function

# Managing Errors

# Monitor Promise

# Producing Promises

# Synchronous

```
interface  SynchronousHttpClient
{
    /**
     * Sends a PSR-7 request and returns a PSR-7 response.
     *
     *
     * @return ResponseInterface
     */
    public function sendRequest(
        RequestInterface $request
    ): ResponseInterface;
}
```

# Asynchronous

```
interface AsynchronousHttpClient
{
    /**
     * Sends a PSR-7 request and returns a Promise
     * that will be resolved into a PSR-7 response.
     *
     * @return Promise
     */
    public function sendRequest(
        RequestInterface $request
    ): Promise;
}
```

# Synchronous

```
function myAction(SynchronousHttpClient $client)
{
    /** @var RequestInterface $request */
    $request = createRequest();

    echo "before";

    /** @var ResponseInterface $response */
    $response = $client->sendRequest($request);
    echo (string)$response->getBody();

    echo "after";
}
```

# Asynchronous

```
function myAction(AsynchronousHttpClient $client)
{
    /** @var RequestInterface $request */
    $request = createRequest();

    echo "before";

    /** @var Promise $promise */
    $promise = $client->sendRequest($request);
    //?? echo (string)$response->getBody();

    echo "after";
}
```

# *Thenable* Promise

```
echo "before";

/** @var Promise $response */
$promise = $client->sendRequest($request);

$promise->then(function(ResponseInterface $response) {
    echo (string)$response->getBody();
});

echo "after";
```

# Synchronous

```
function myAction(SynchronousHttpClient $client)
{
    /** @var RequestInterface $request */
    $request = createRequest();

    echo "before";

    /** @var ResponseInterface $response */
    $response = $client->sendRequest($request);
    echo (string)$response->getBody();

    echo "after";
}
```

# Asynchronous

```
function myAction(AsynchronousHttpClient $client)
{
    /** @var RequestInterface $request */
    $request = createRequest();

    echo "before";

    /** @var ResponseInterface $response */
    $response = yield $client->sendRequest($request);
    echo (string)$response->getBody();

    echo "after";
}
```