# Why and how to use generators for asynchronous programming

BEDROCK

Benoit Viguier

@b_viguier

ConFoo.CA
DEVELOPER CONFERENCE

# Why and how to use generators for asynchronous programming

BEDROCK

Benoit Viguier

@b_viguier

ConFoo.CA
DEVELOPER CONFERENCE

# Why and how to use generators for asynchronous programming

Misconception

Viguier
_viguier

ConFoo.CA
DEVELOPER CONFERENCE

# Why and how to use generators for asynchronous programming

BEDROCK

Benoit Viguier

@b_viguier

ConFoo.CA
DEVELOPER CONFERENCE

# Why and how to use generators for asynchronous programming

BEDROCK

Benoit Viguier

@b_viguier

ConFoo.CA
DEVELOPER CONFERENCE

# Why and how to use generators for asynchronous programming

BEDROCK

Benoit Viguier

@b_viguier

ConFoo.CA
DEVELOPER CONFERENCE

# Why and how to use generators for asynchronous programming

Benoit Viguier
@b_viguier

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

Lyon, France

BEDROCK

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

M6
GROUPE
RTL
GROUP

## 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
- M6 Music
- Tiji
- canal J
- MCM TOP
- MCM
- RFM TV

### TÉLÉVISION INTERNATIONALE

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

### RADIO

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

### DIGITAL

- gulli Replay
- 6play
- gulli MAX

## DIVERSIFICATIONS

- M6 INTERACTIONS
- M6 ÉDITIONS
- M6 ÉVÉNEMENTS
- M6 MUSIC LABEL
- gulli Parc
- Ventadis – Vente à distance - Groupe M6
- M6 boutique
- BEST of TV
- Leonis Corner
- JOIKKA
- M6 CRÉATIONS
- M6 DIGITAL SERVICES
- Radins.com
- iGraal
- odiso
- mindbaz – your emailing service
- MINDTARGET
- TURBO
- M6météo
- passeport santé
- Fourchette & Bikini
- cuisineaz
- CROQ'KILOS
- Comptoir Botanique
- déco.fr
- Panorabanques.com

## RÉGIE

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

## 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

- M6
- W9
- 6ter
- gulli

### TÉLÉVISION PAYANTE

- PARIS PREMIÈRE
- téva
- serieclub
- M6 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 ÉDITIONS
- M6 ÉVÉNEMENTS
- M6 MUSIC LABEL
- gulli Parc
- Ventadis – Vente à distance - Groupe M6
- M6 boutique
- BEST of TV
- Leonis Corner
- JOJIKA
- M6 CRÉATIONS
- M6 DIGITAL SERVICES
- Radins.com
- iGraal
- odiso
- mindbaz – your emailing service
- MINDTARGET
- TURBO
- M6météo
- passeport santé
- Fourchette & Bikini
- cuisineaz
- CROQ'KILOS
- Comptoir Botanique
- déco.fr
- Panorabanques.com

## RÉGIE

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

6play › M6 W9 6ter funradio etc.

RTL play À VOUS DE VOIR › RTL TVI club RTL PLUG RTL etc.

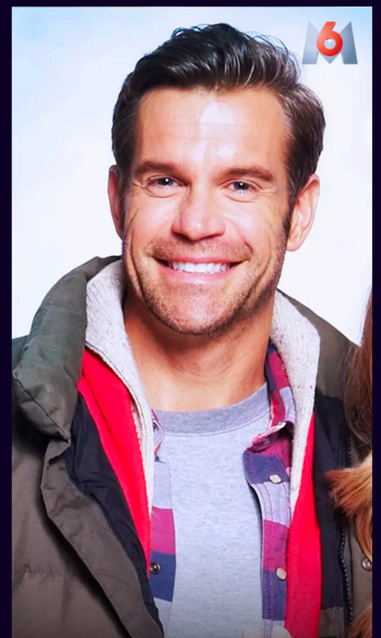
RTL most › RTL KLUB RTL etc.

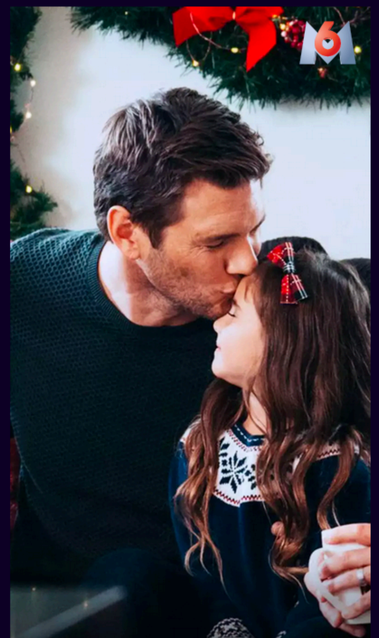
RTL PLAY › RTL 2 RTL Kockica DIGITAL CHAPTERS etc.

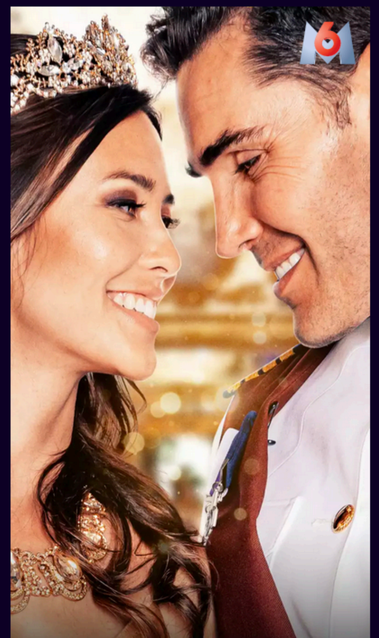
10 Million unique
users per month

# Programmes par genre

# Le Groupe M6 vous souhaite une bonne année !







# One year of asynchronous PHP in production

ConFoo.CA
DEVELOPER CONFERENCE

**Tomorrow**, 13:00

# Asynchronous Programming

BEDROCK

Benoit Viguier

@b_viguier

ConFoo.CA
DEVELOPER CONFERENCE

# Tea for Two

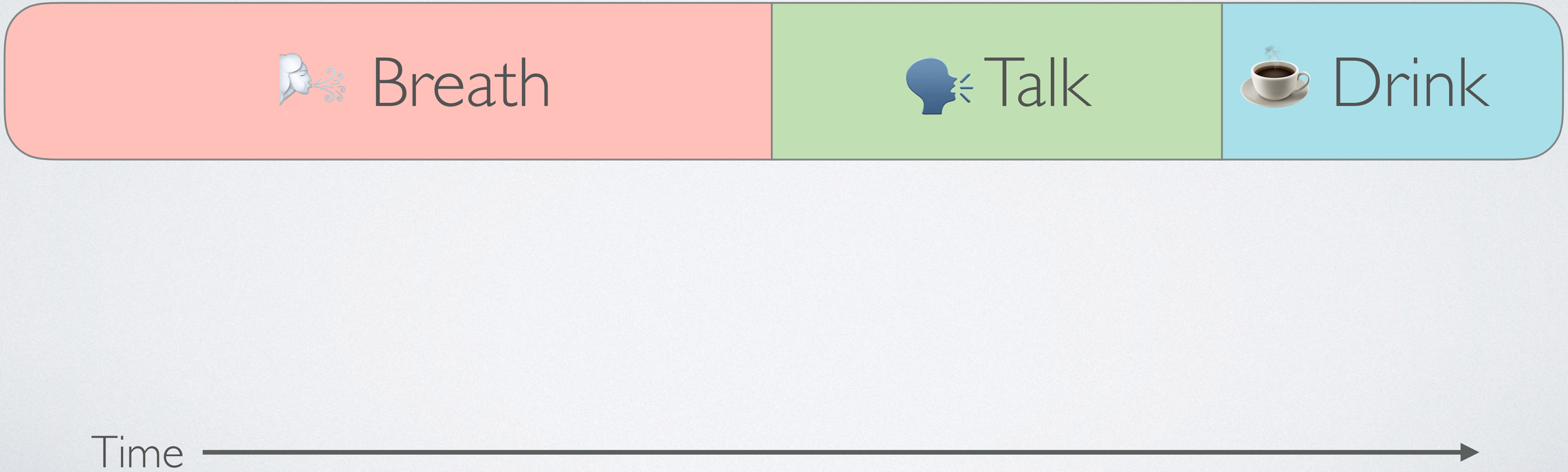
An illustrated example

# Tasks

- ☕ **Drink** Tea
- 🗣️ **Talk** with a friend
- 🌬️ **Breath** just… to live

Tea parlor ? Cafe
Talk/Catch up

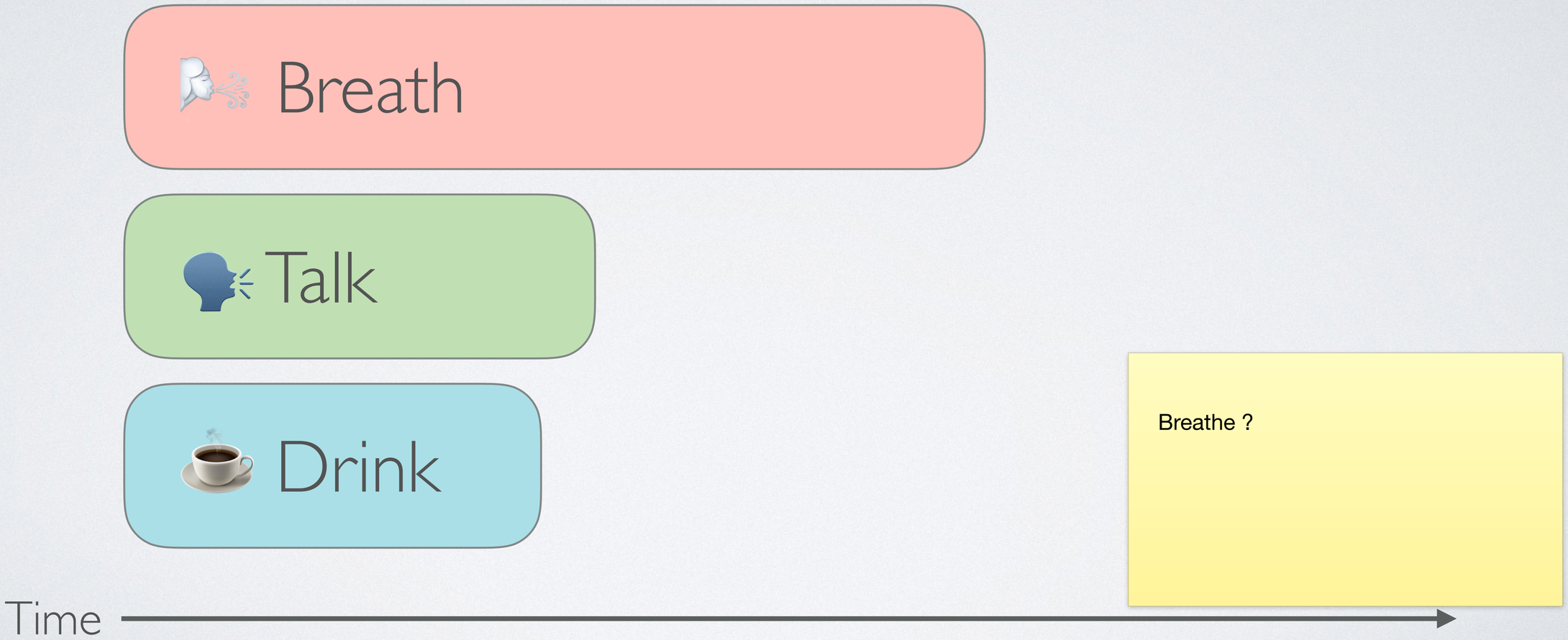
# Synchronous



# Concurrency

Starting next task before the current one ends

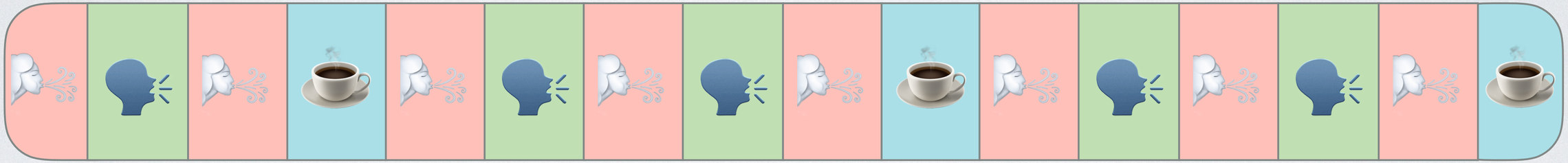
# Parallel



# Most languages are single threaded

… by default

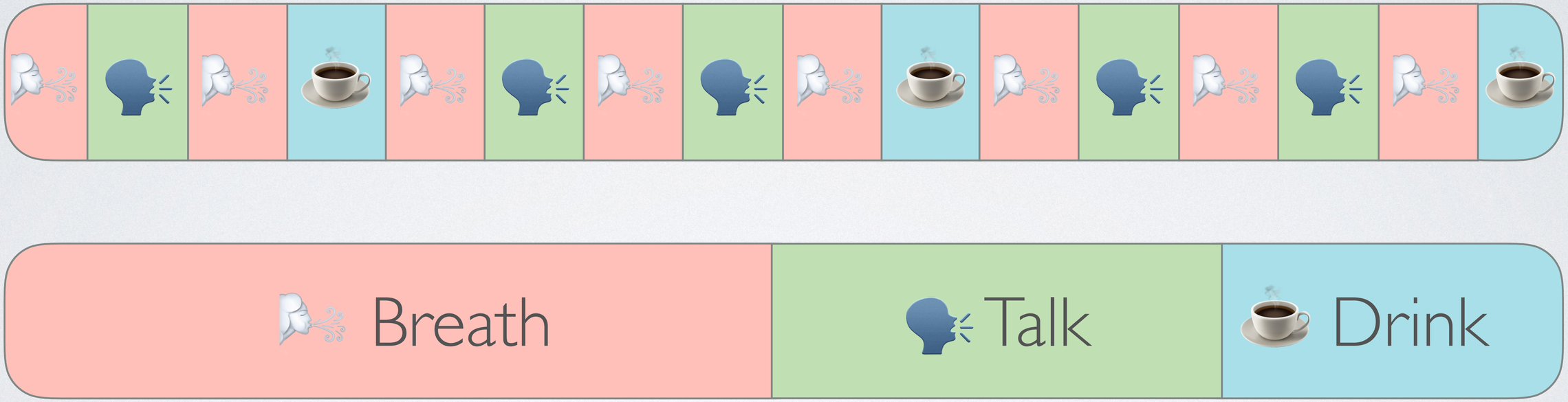
# Asynchronous



Time

Serials tasks (vs parallel)

# But… 🤔




Time →

Discuss / Talk

# Analyzing tasks

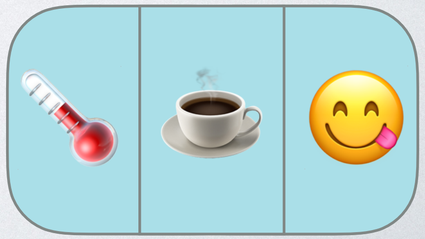
*Internal/External* operations

# **Drinking** Tea

Waiting for the right temperature

Drinking (actually)

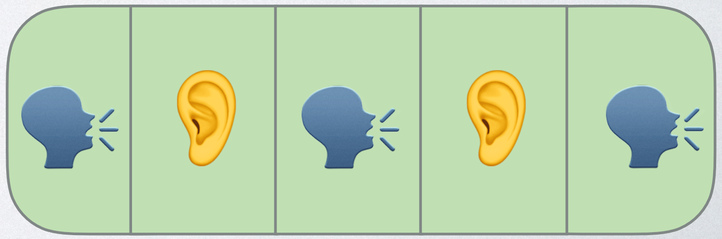
Enjoying



# **Talking** with a friend

Speaking, asking something

Listening

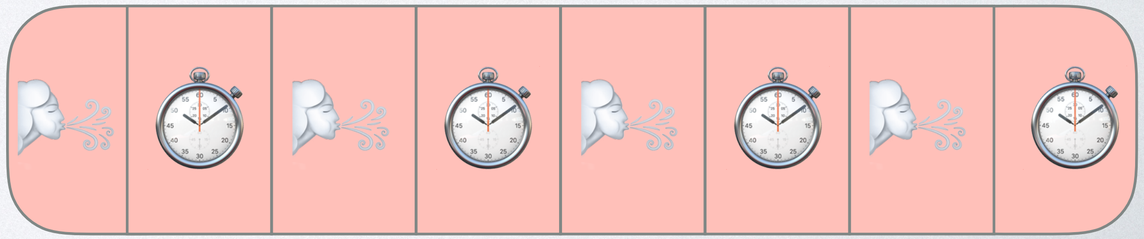


# Breathing

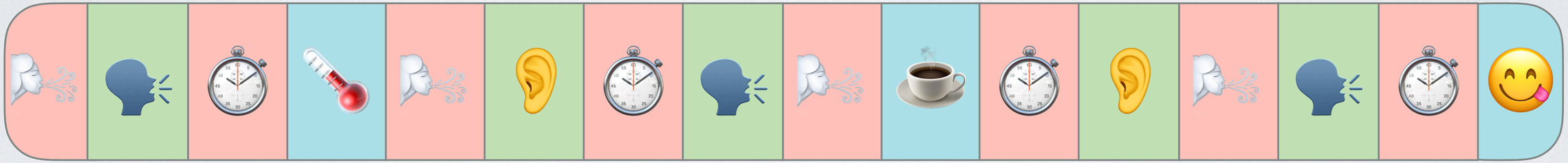
Breathing…

Waiting (a little)

Breathing…

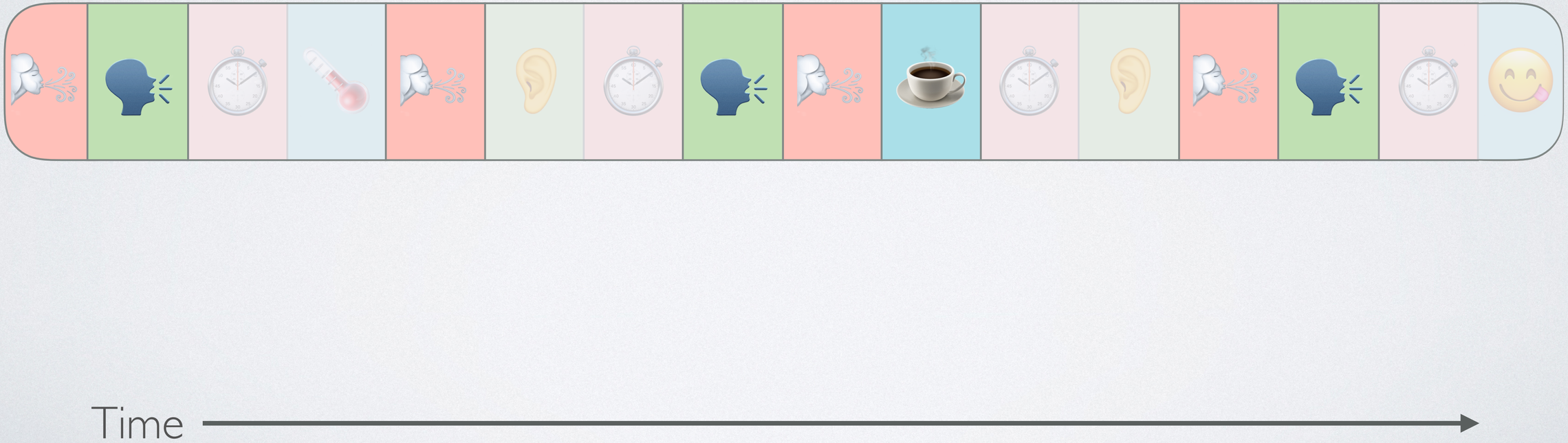


# Asynchronous

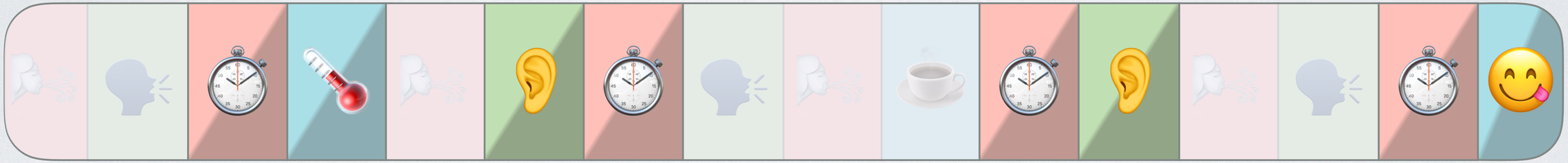


Time

# *Internal* operations



# *External* operations



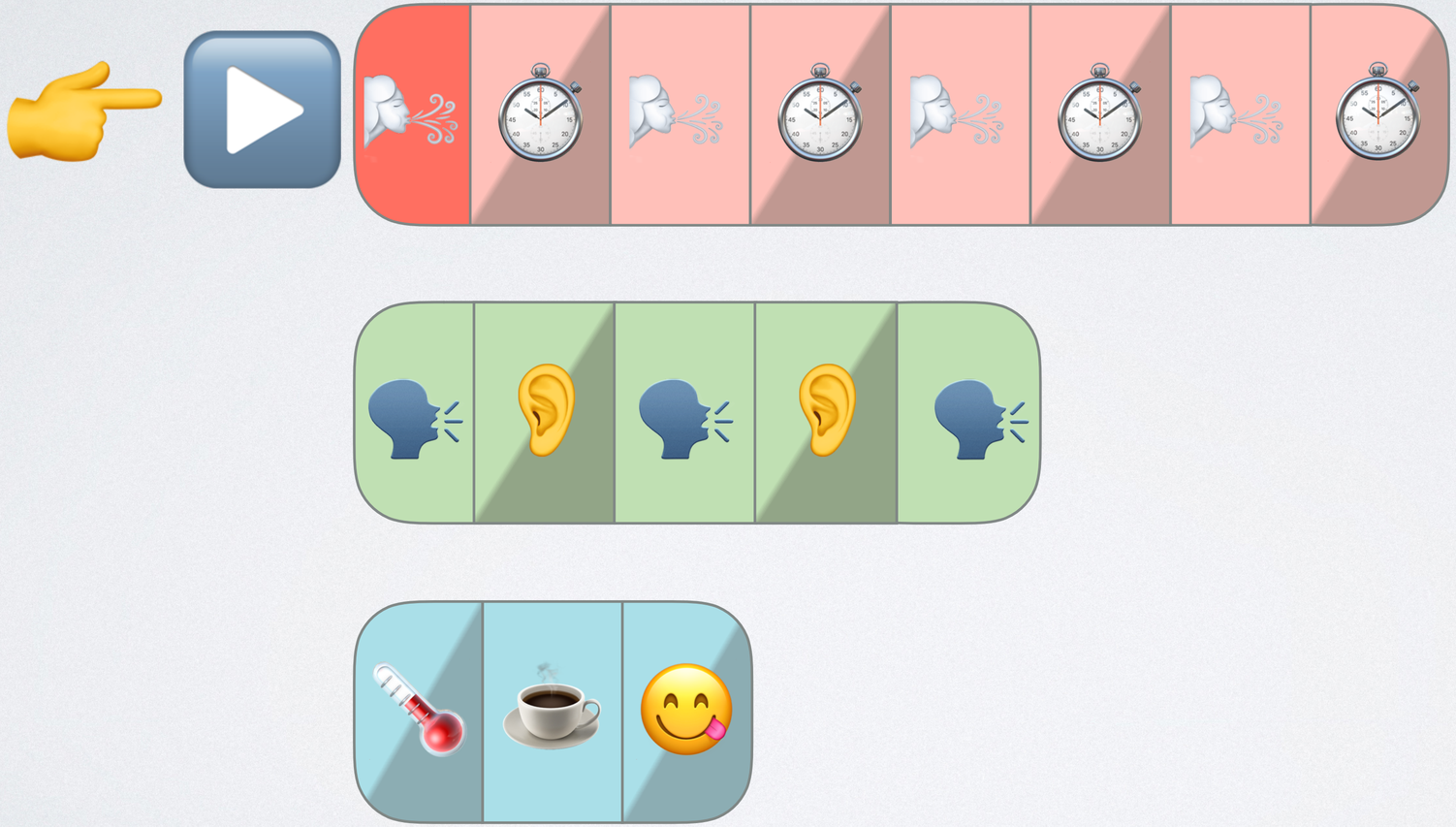
Time

# Dispatching execution

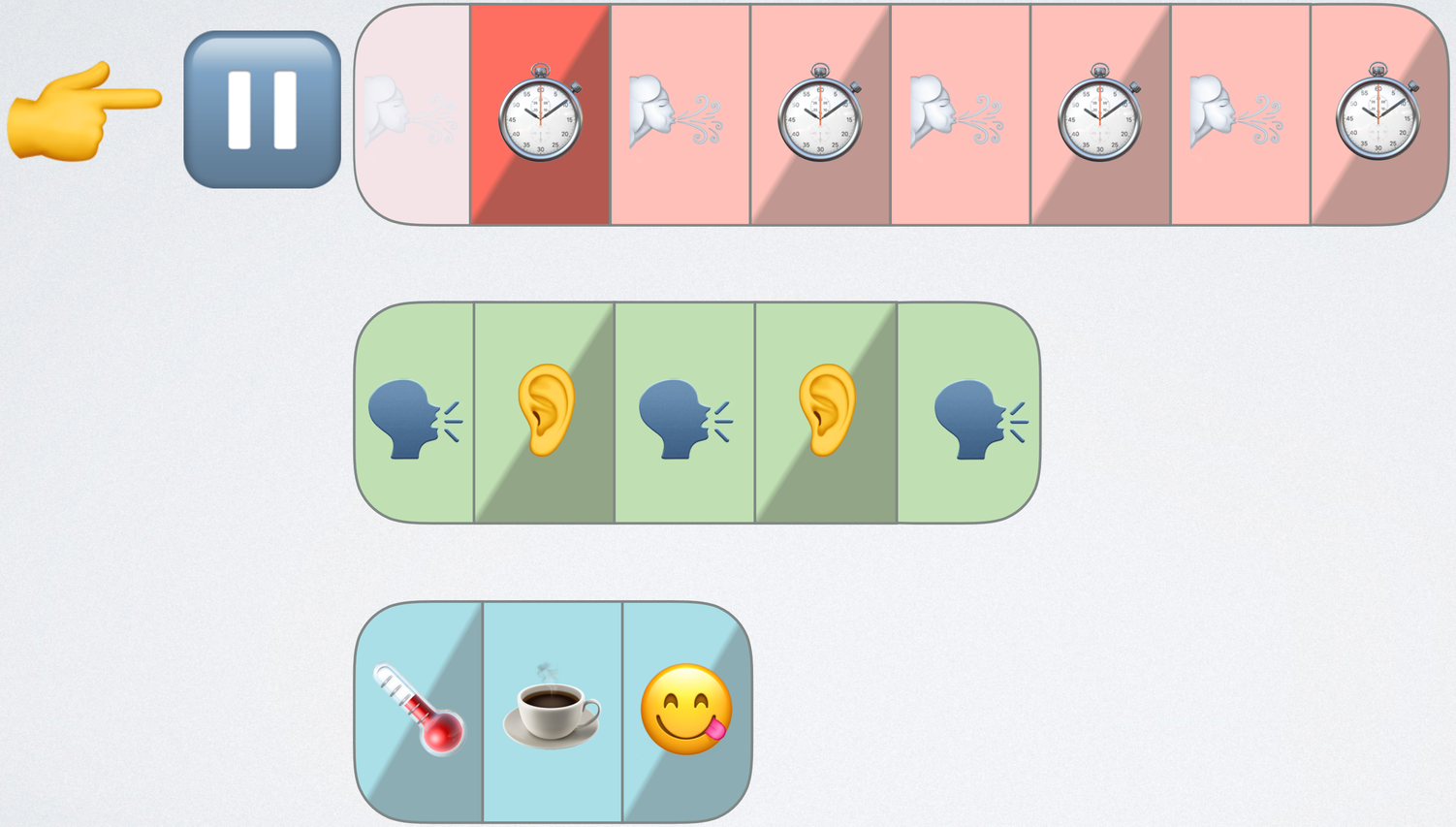
Event Loop

# Event Loop

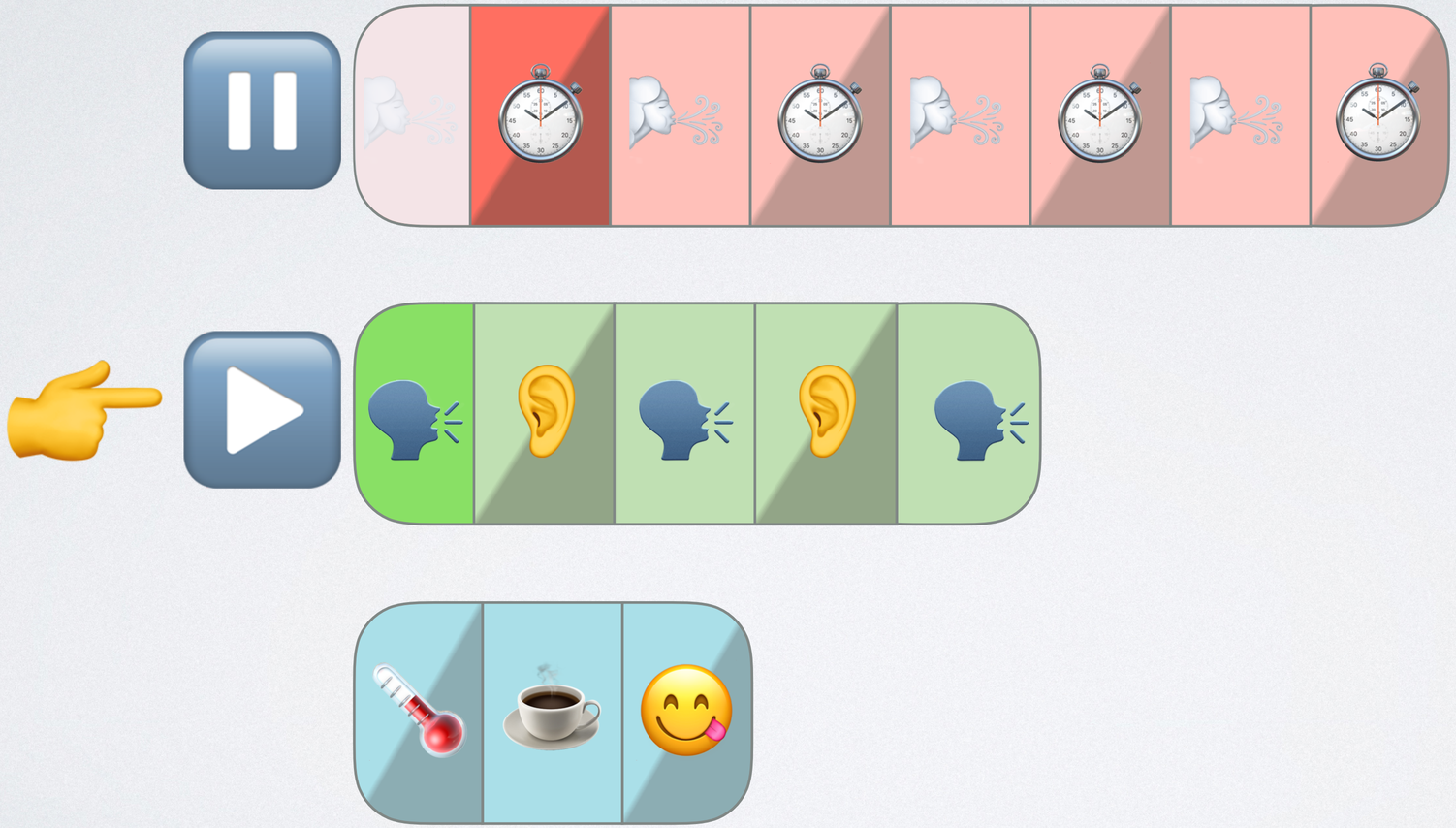
# Event Loop



# Event Loop



# Event Loop

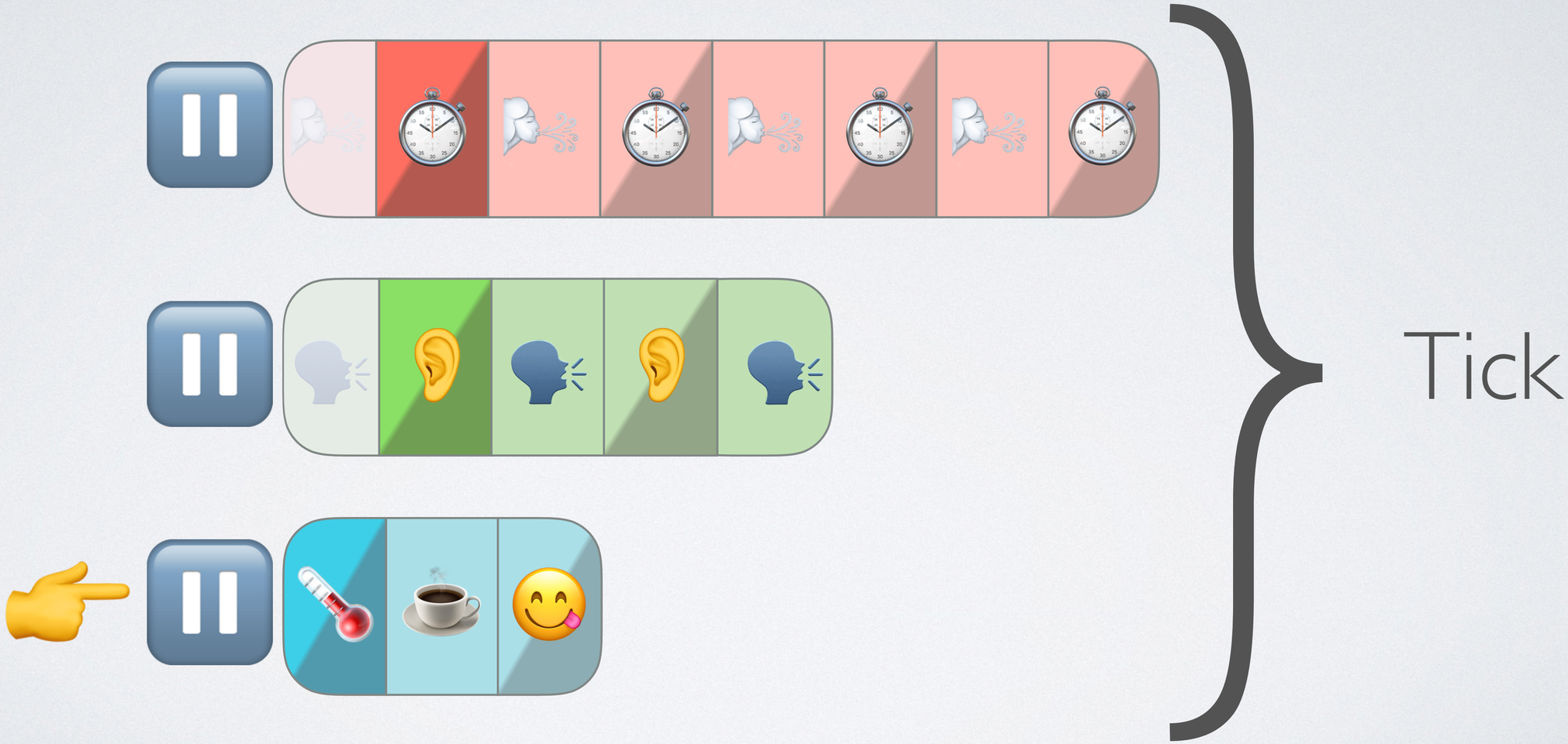


# Event Loop

# Event Loop

# Event Loop

# Event Loop



# Event **Loop**

# Event Loop

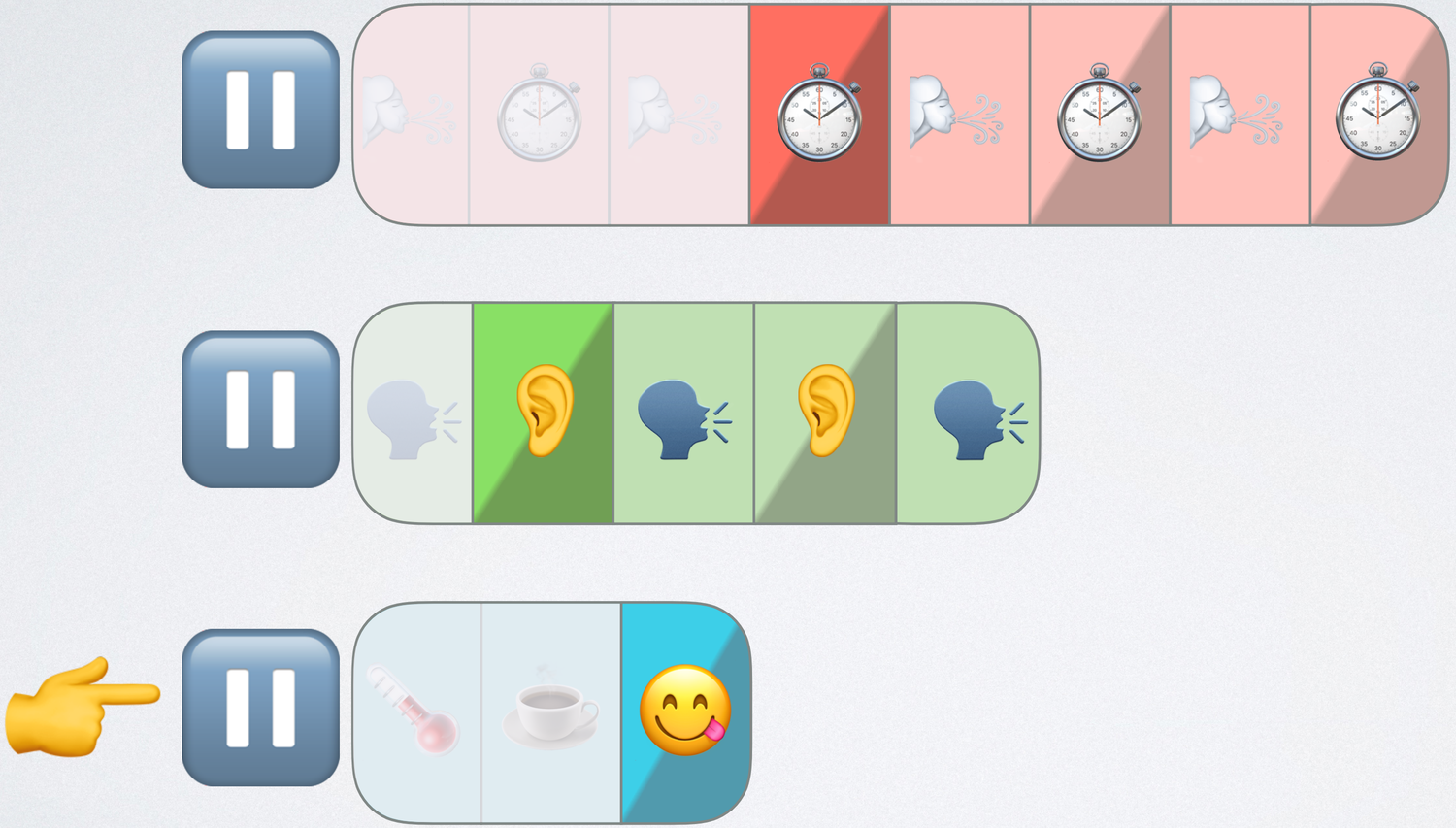
# Event Loop

# Event Loop

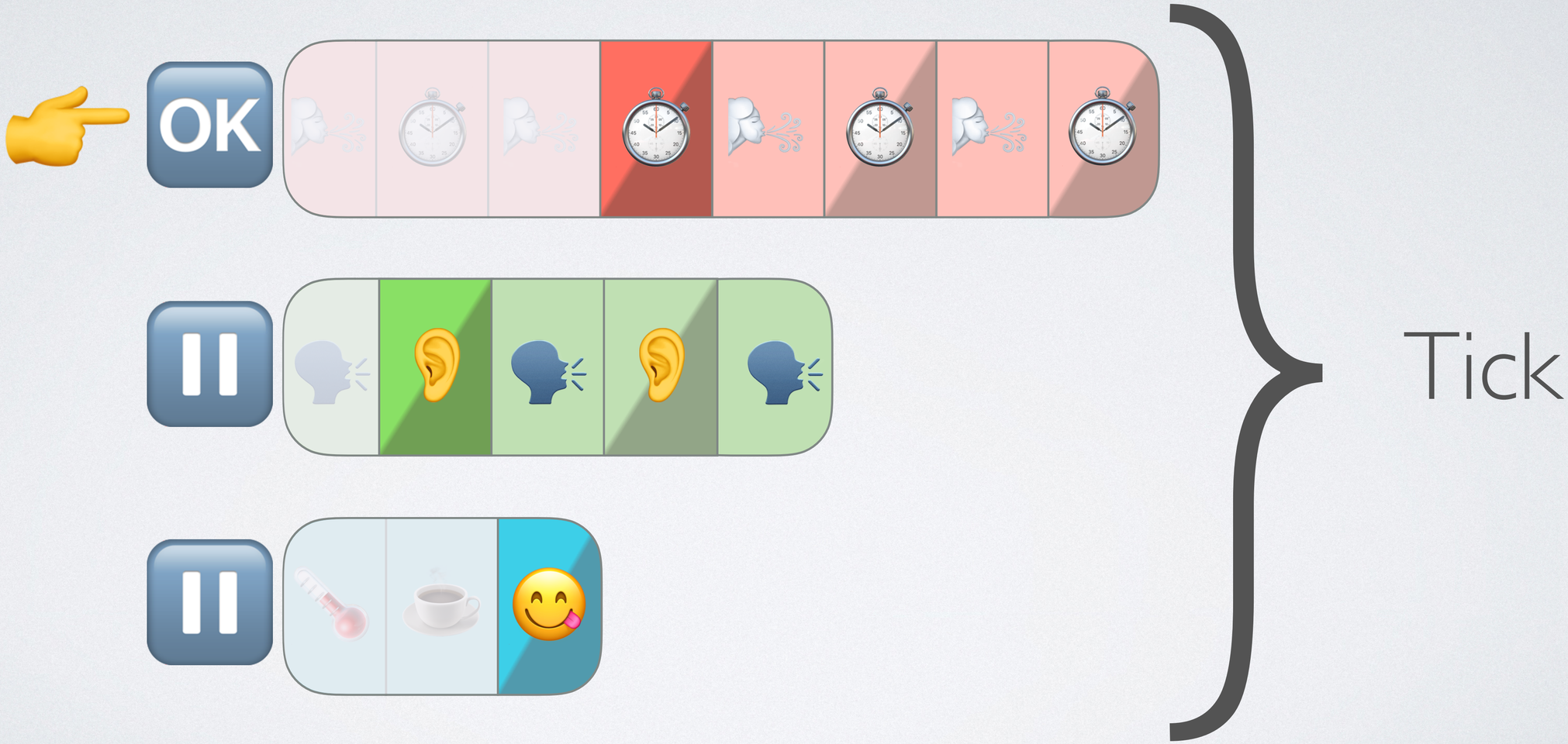
# Event Loop

# Event Loop

# Event Loop

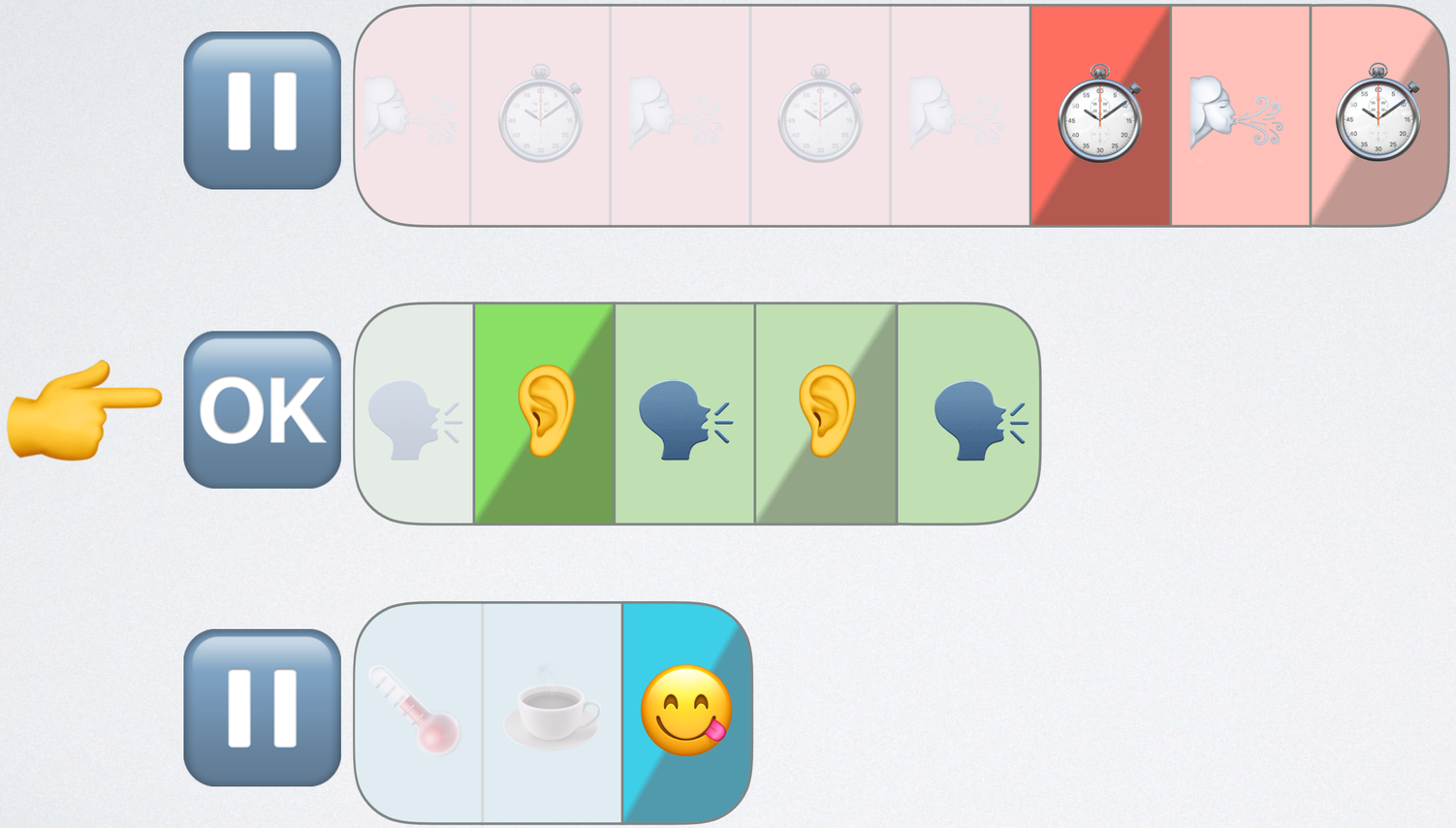


# Event Loop



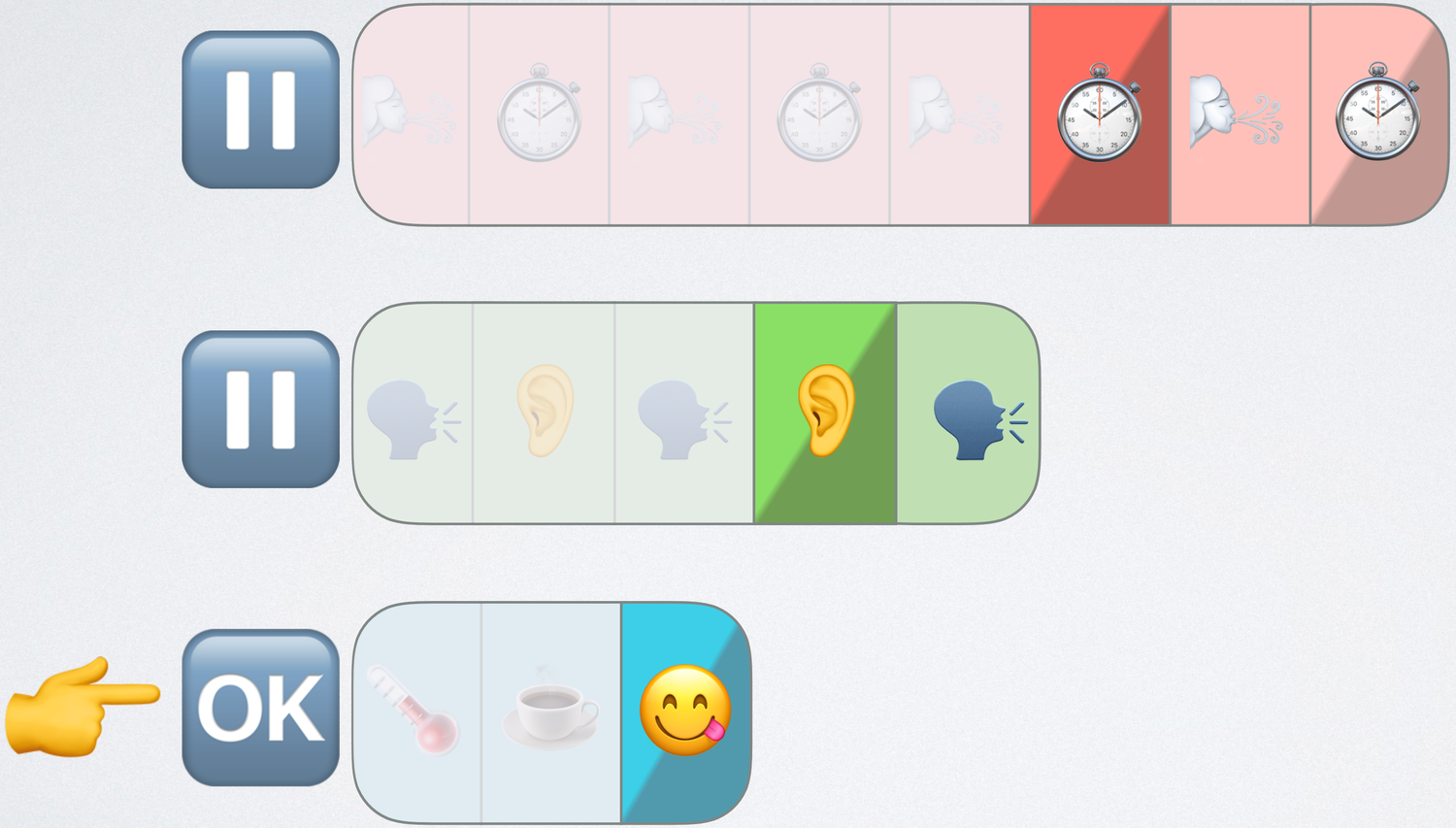
# Event Loop

# Event Loop



# Event Loop

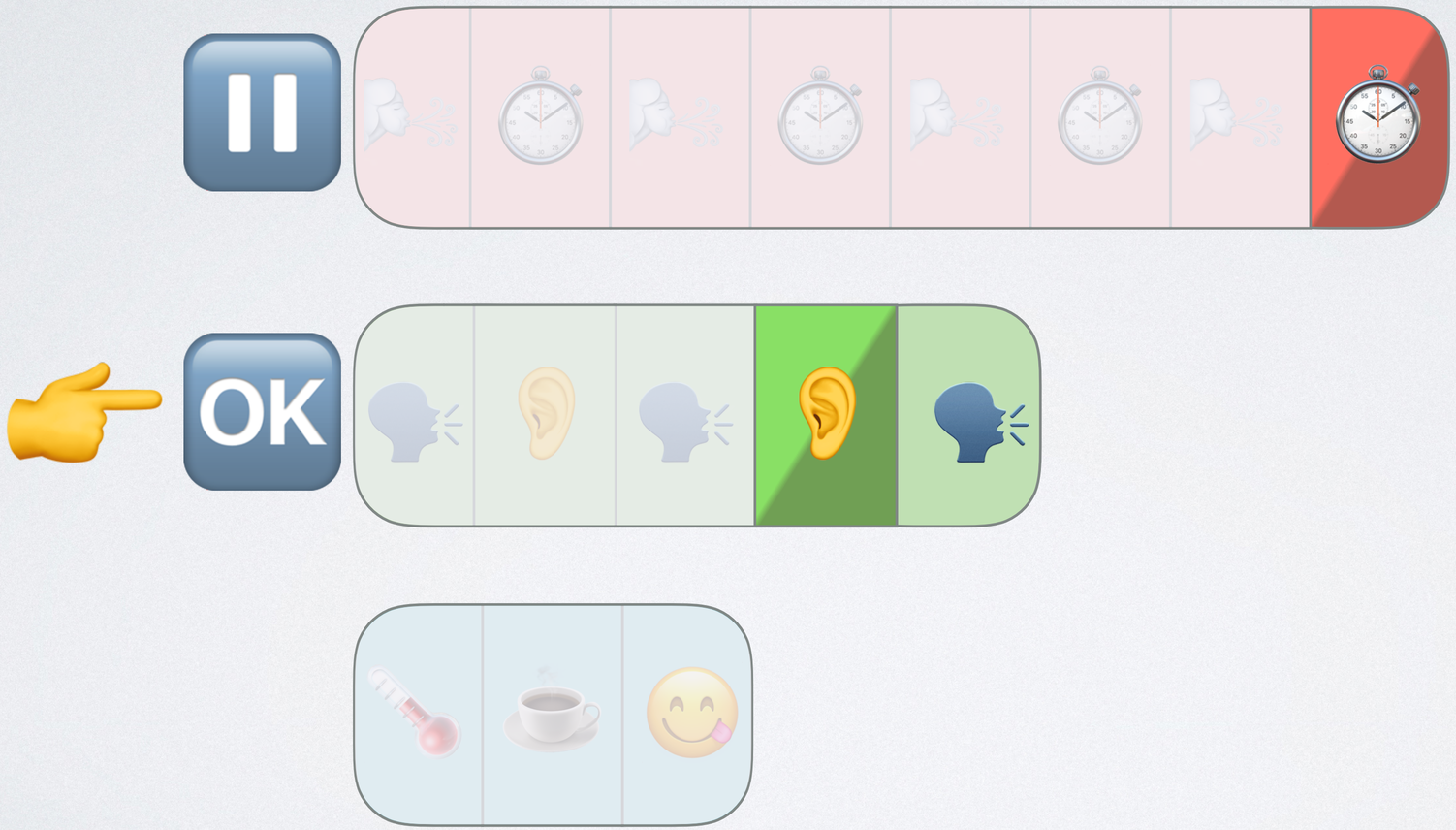
# Event Loop



# Event Loop

# Event Loop

# Event Loop

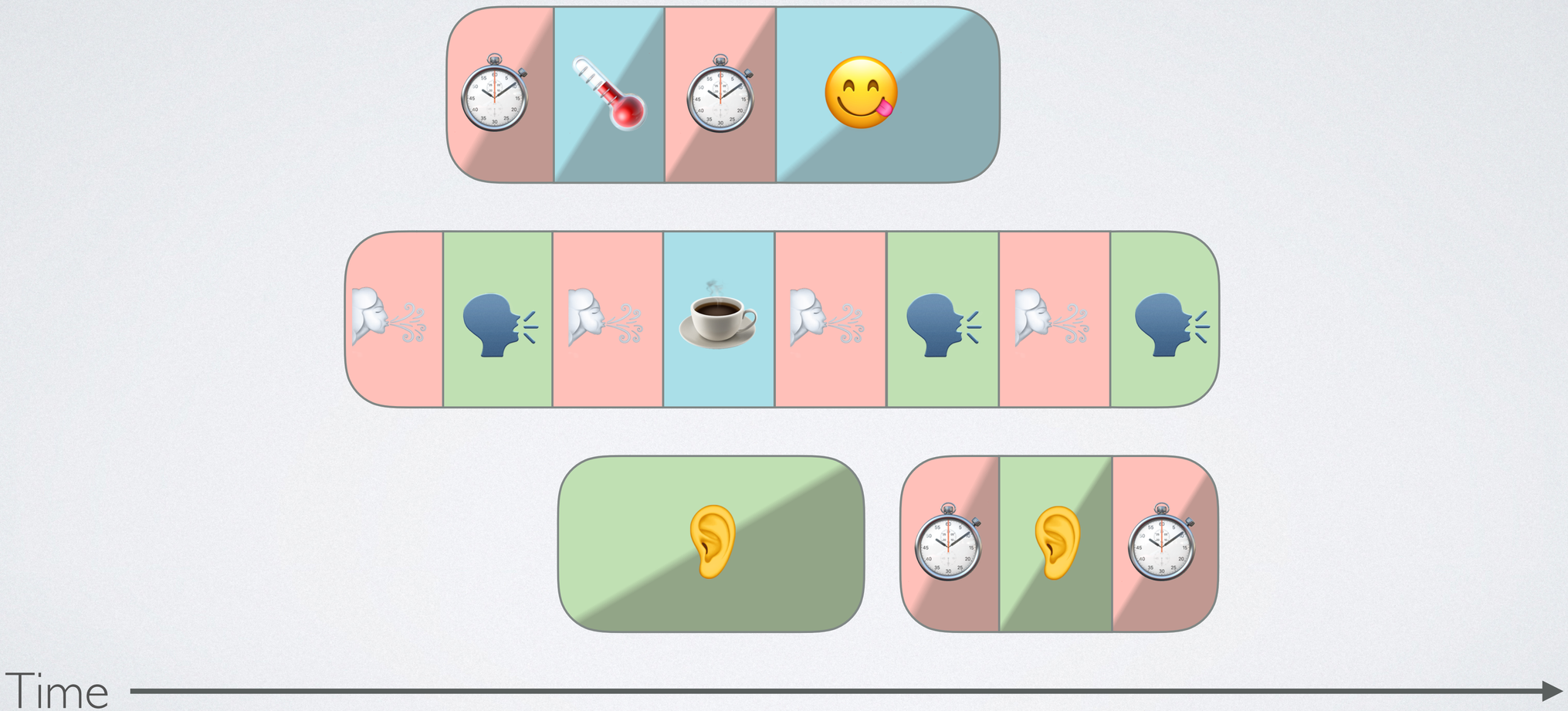


# Event Loop

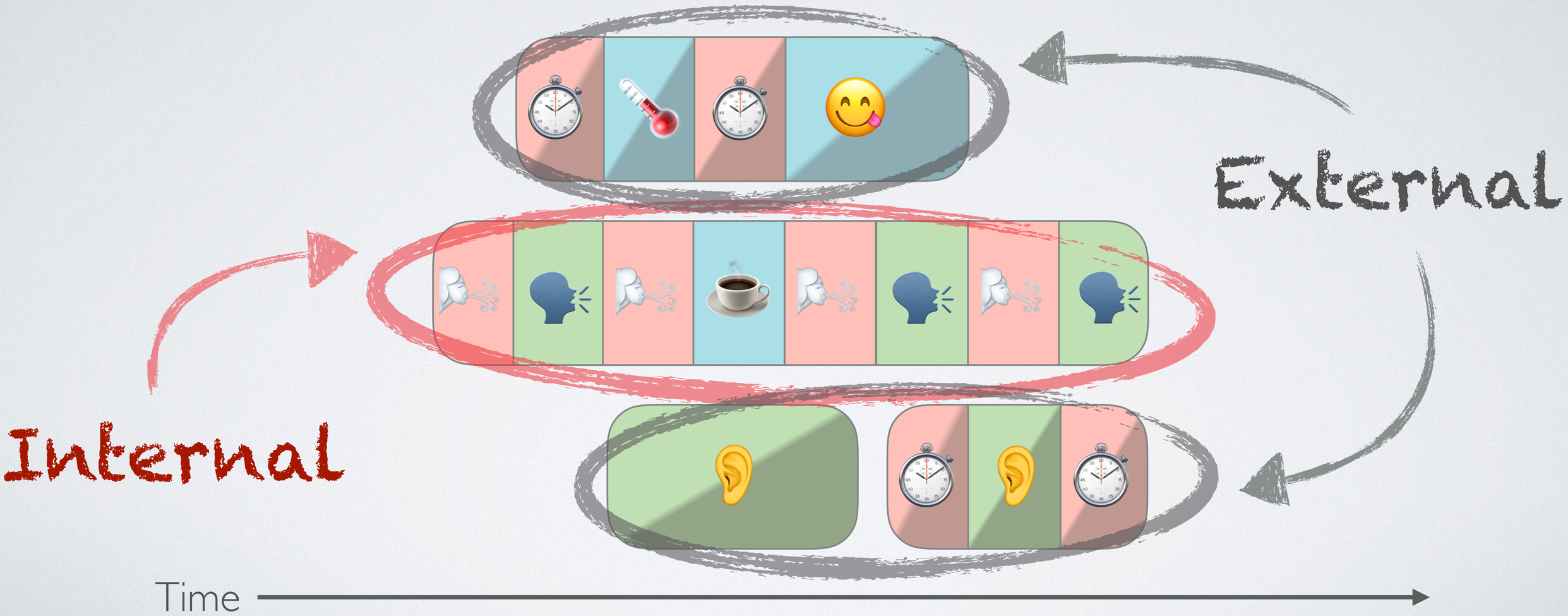
# Event Loop

# Event Loop

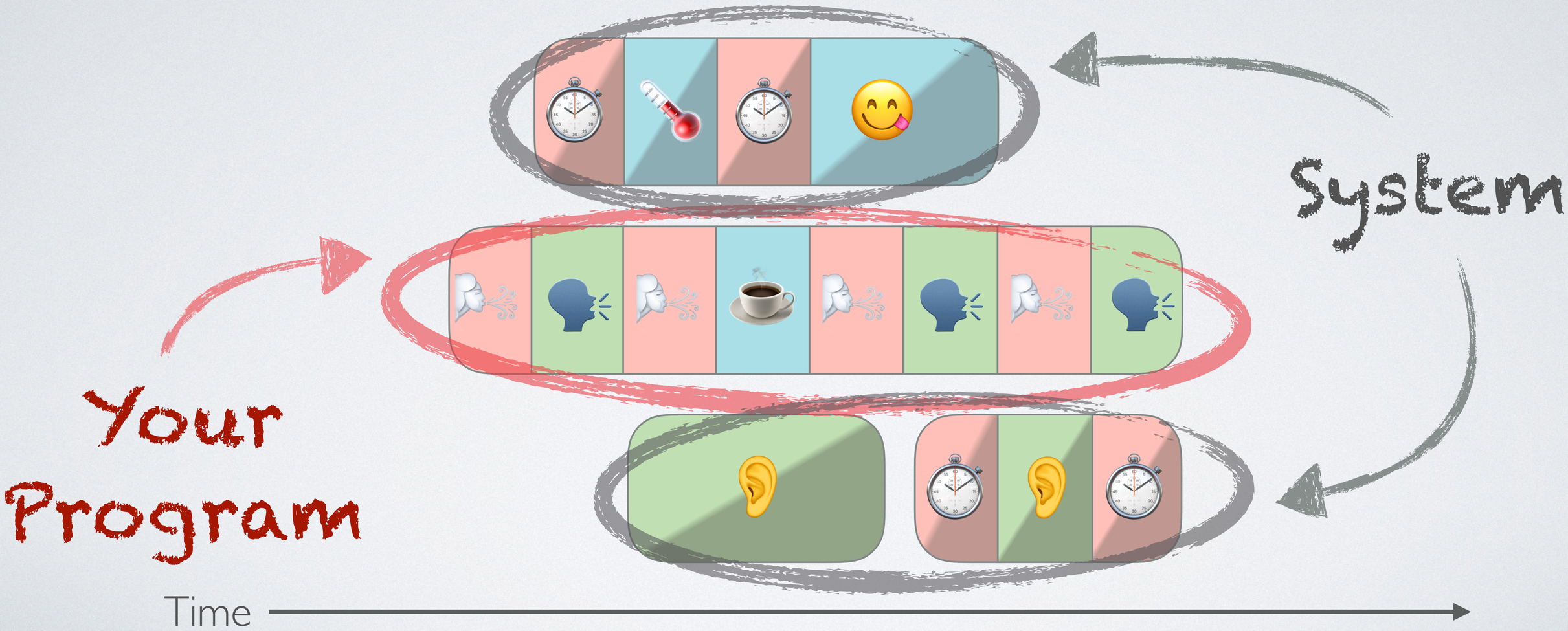
# Asynchronous



# Asynchronous



# Asynchronous



# How to dispatch code execution efficiently?

# Generators

Subset of coroutines

BEDROCK

Benoit Viguier

@b_viguier

ConFoo.CA
DEVELOPER CONFERENCE

PARENTAL

ADVISORY

EXPLICIT PHP

# Example

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

$generator = gen_one_two_three();
foreach ($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

```
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

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

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

Inside

Outside

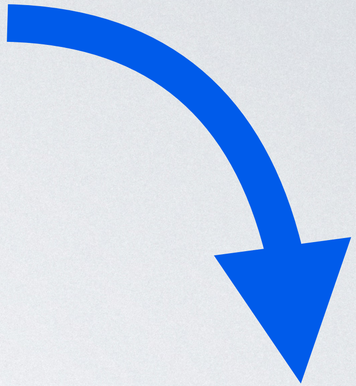
```
$value = $generator->current();
$key = $generator->key();
```

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

Inside

Outside

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

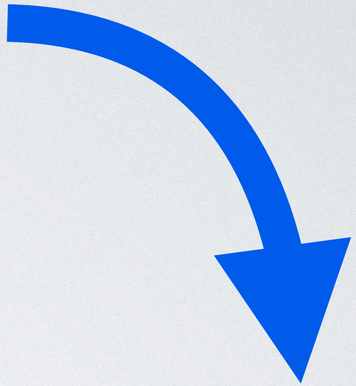


```
$value = yield;
$nullValue = yield;
```

Inside

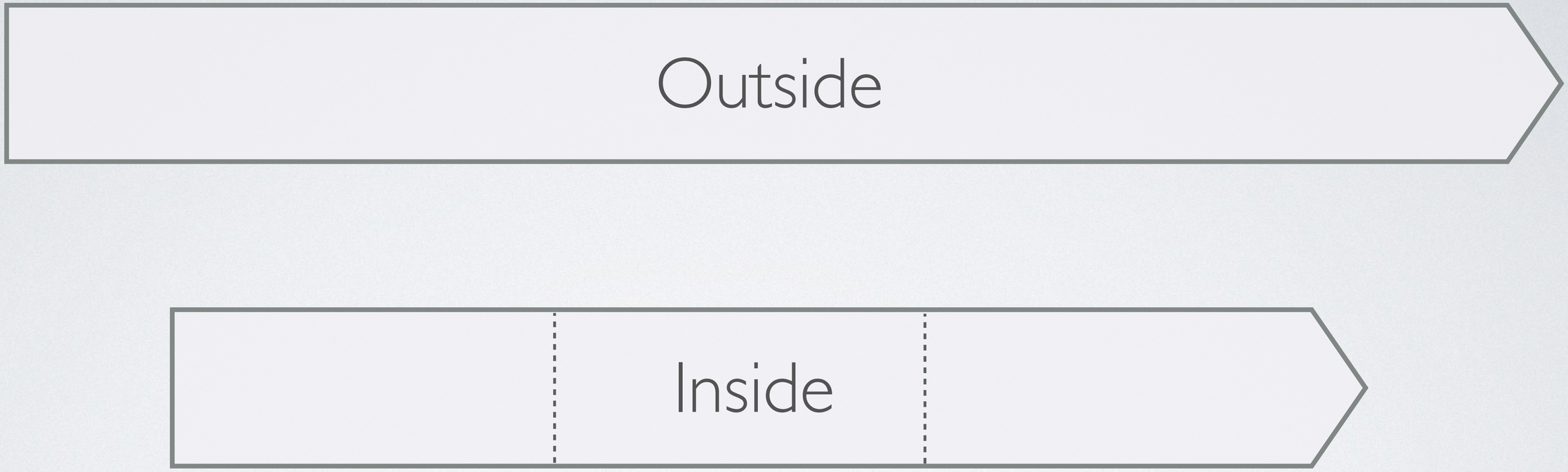
Outside

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

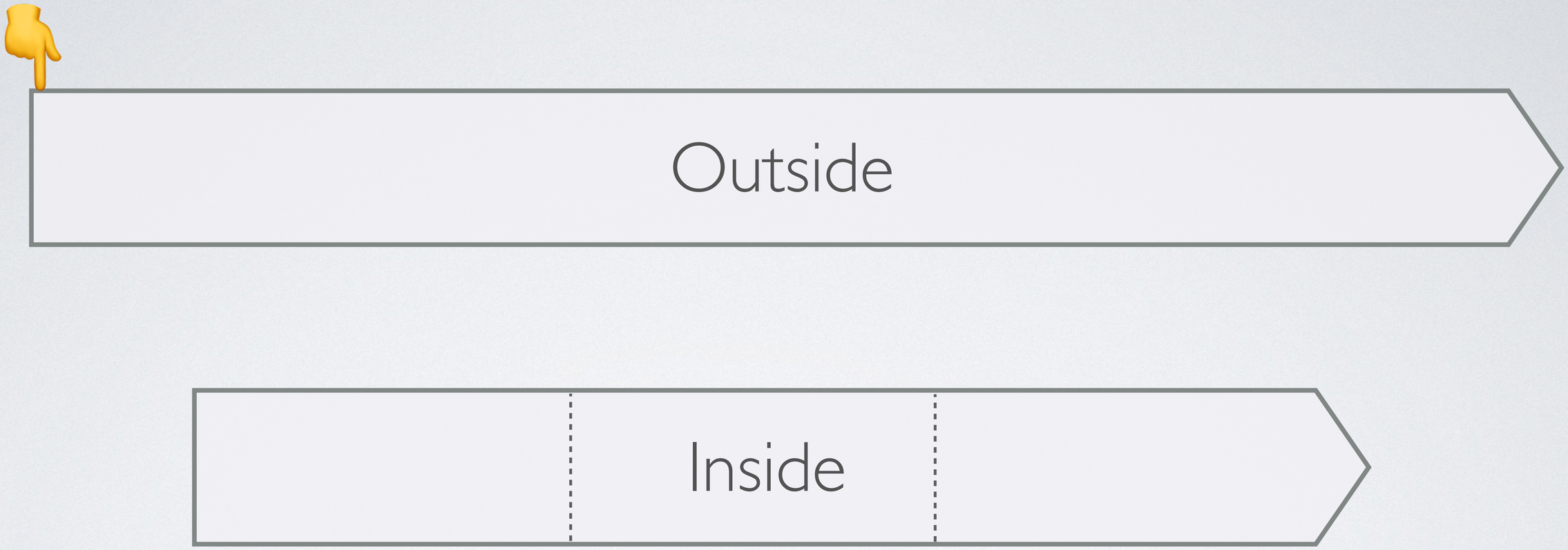


```
$v2 = yield $v1;
yield $v3;
```

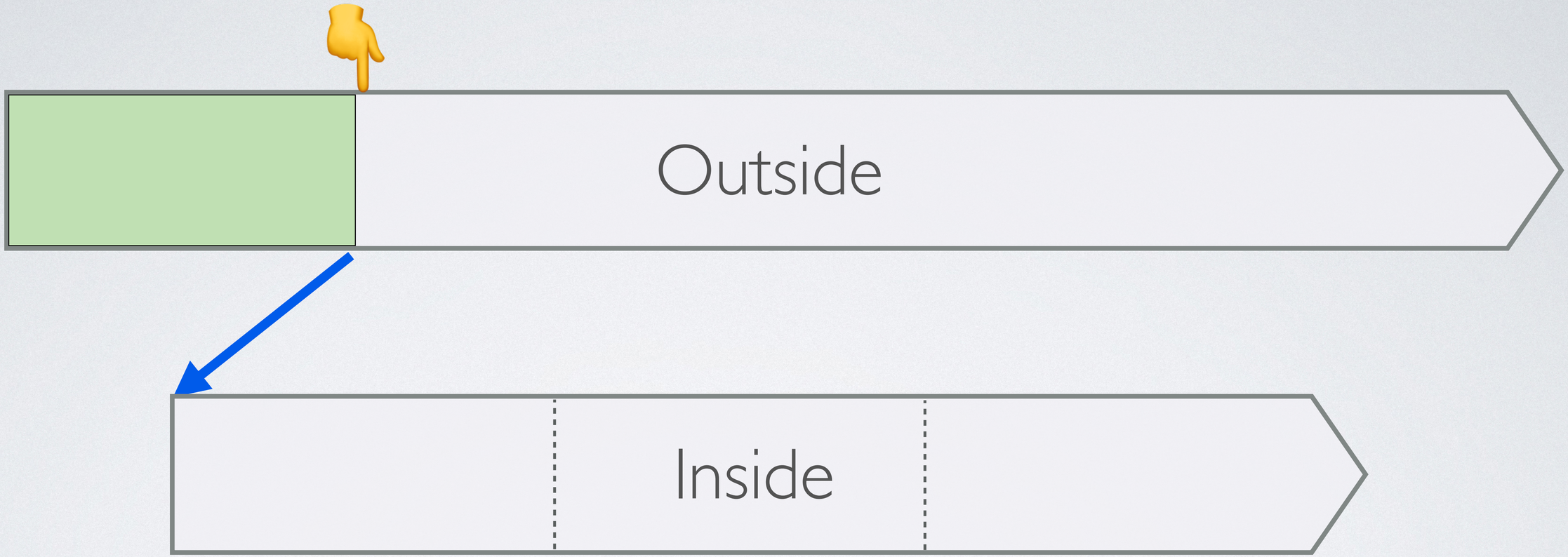
Inside


Outside
Inside

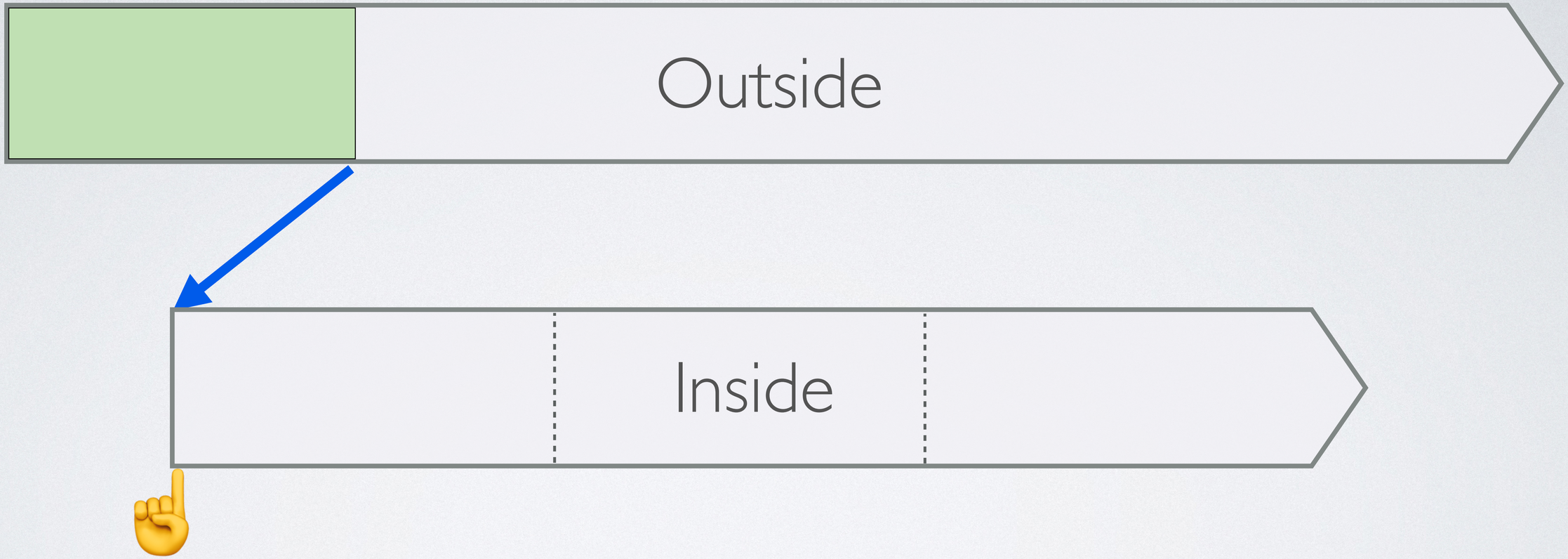
```
$generator = createMyGenerator()
```



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

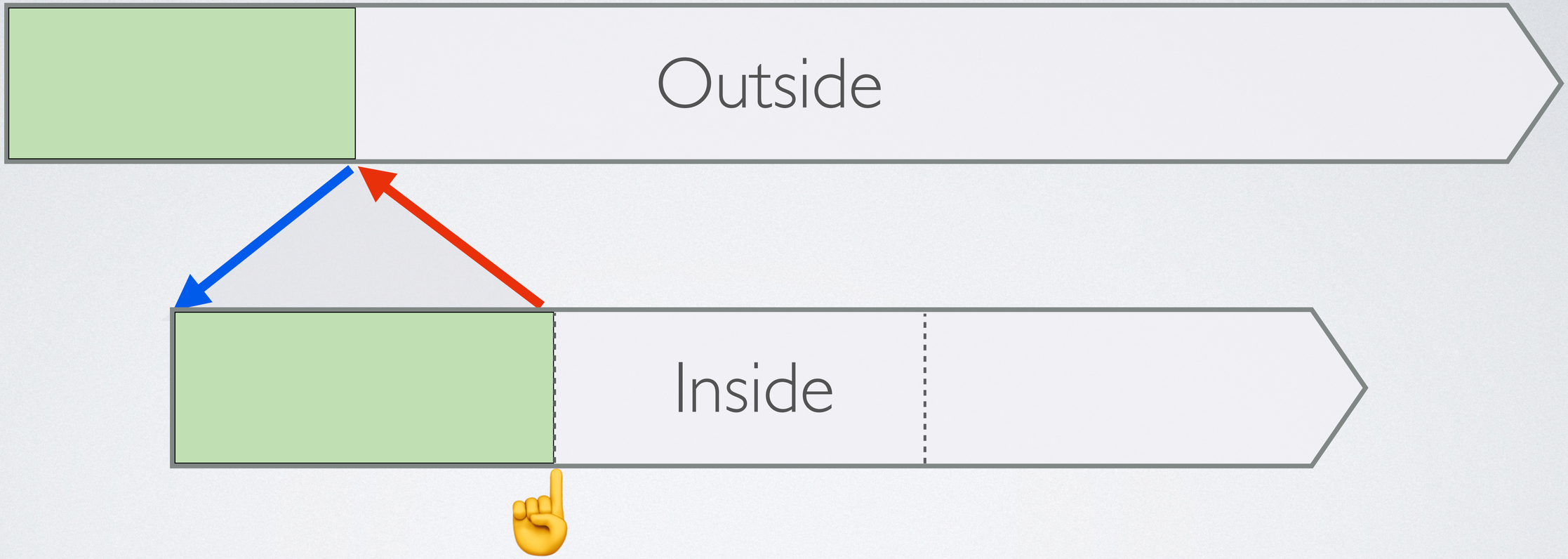


```
$v1 = $generator->current()
```



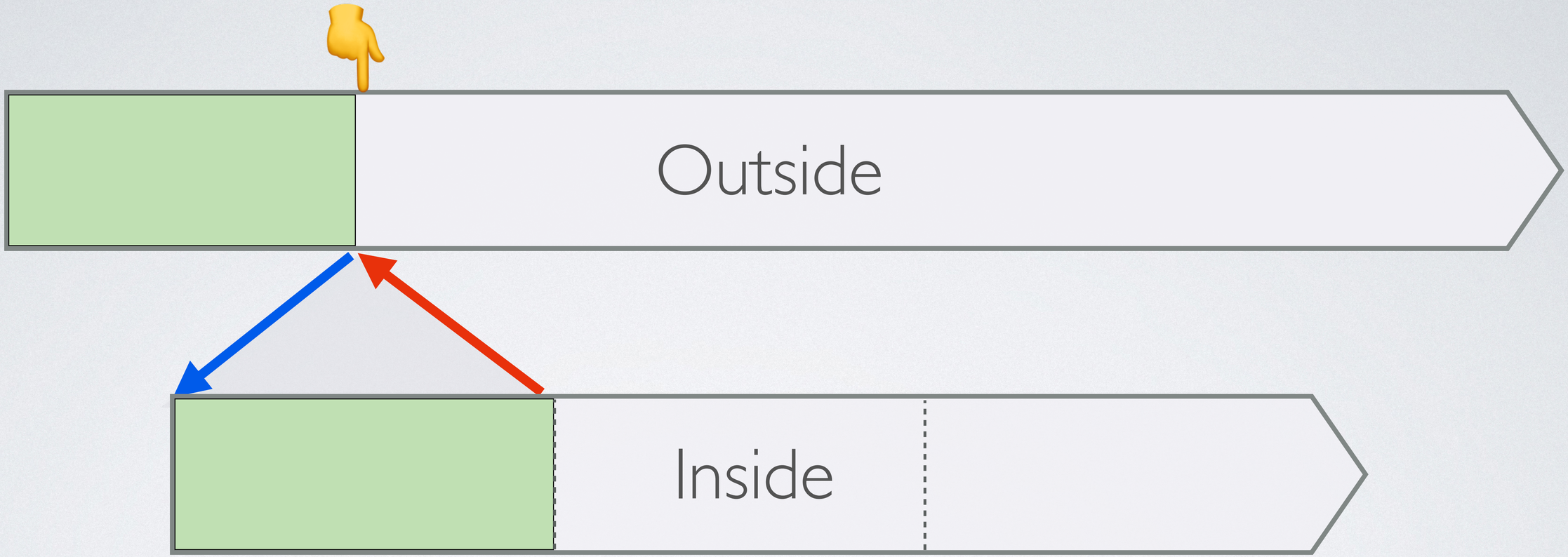

To handoff to

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



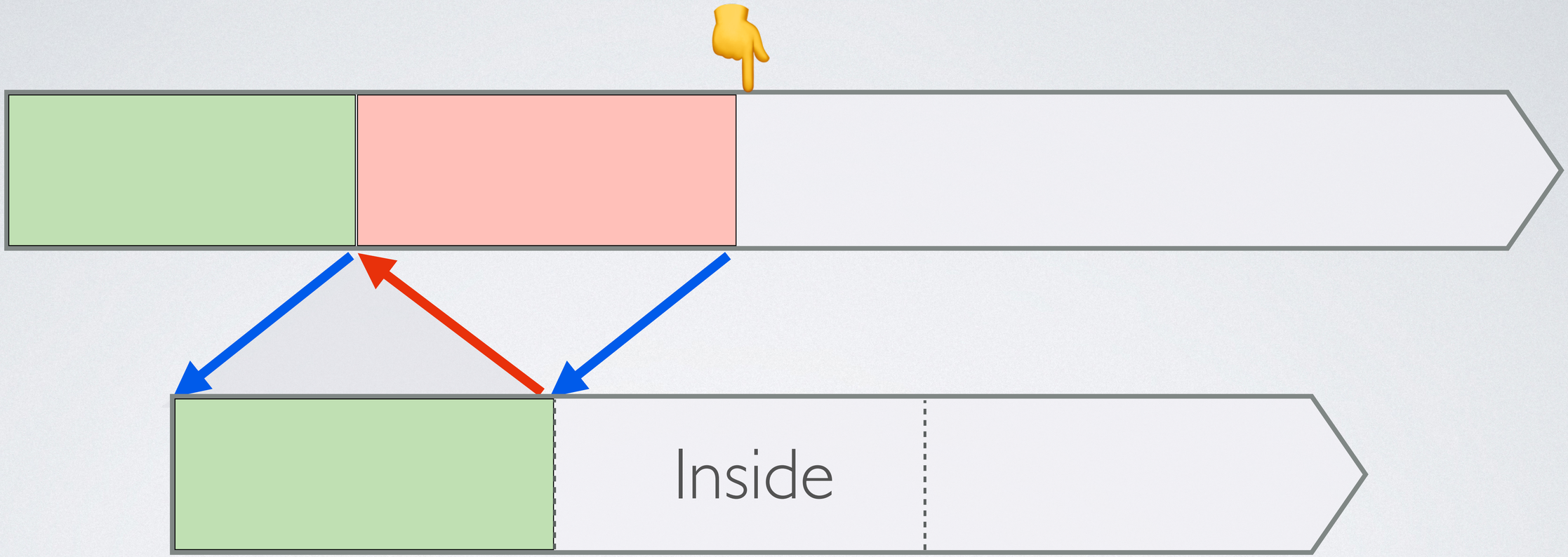

```
$v2 = yield $v1
```

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



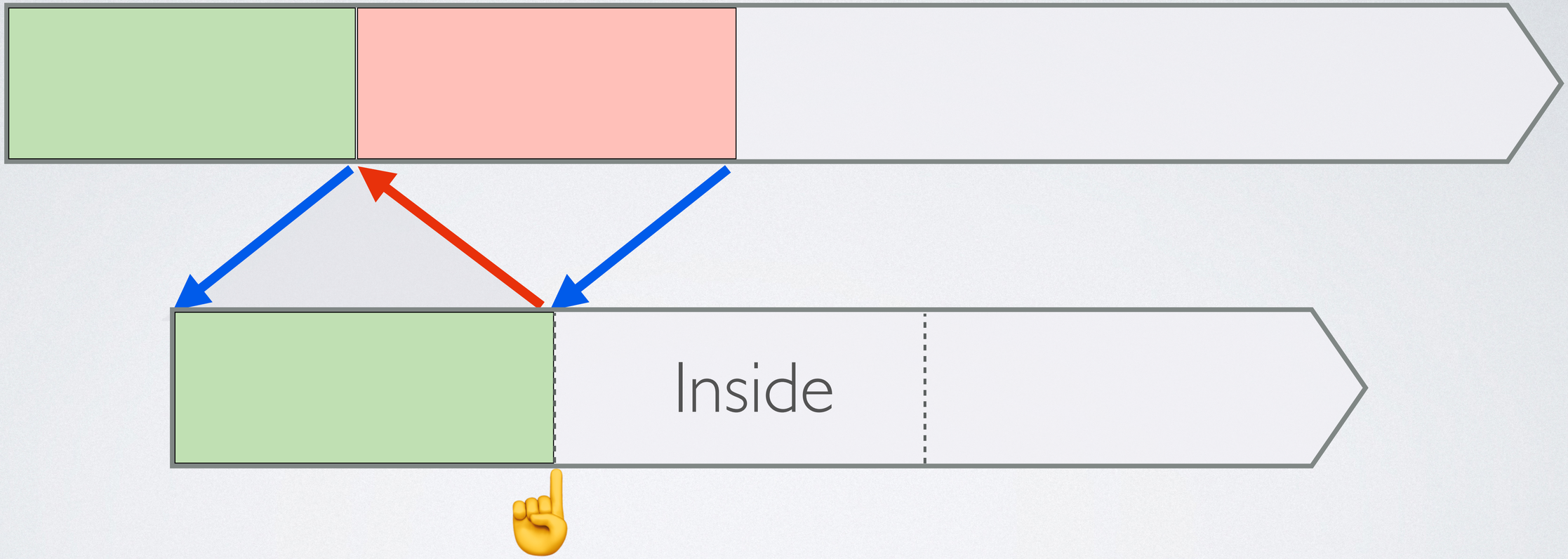

```
$v2 = yield $v1
```

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



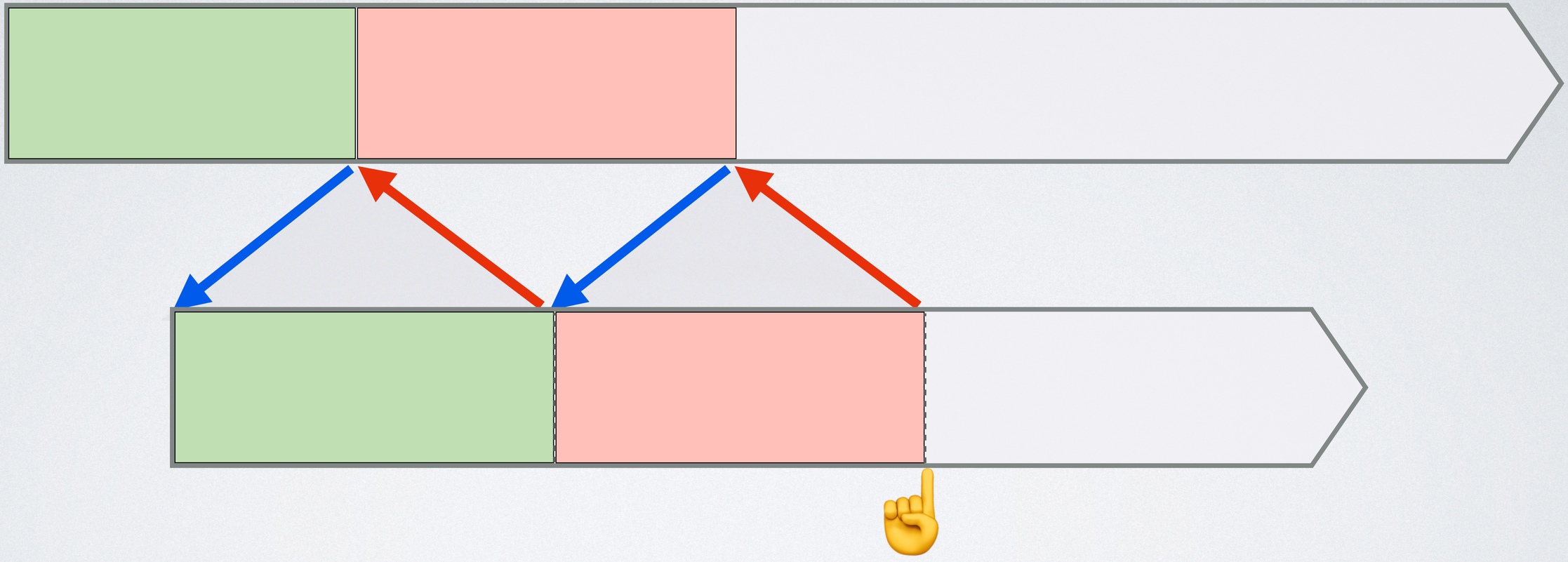

```
$v2 = yield $v1
```

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



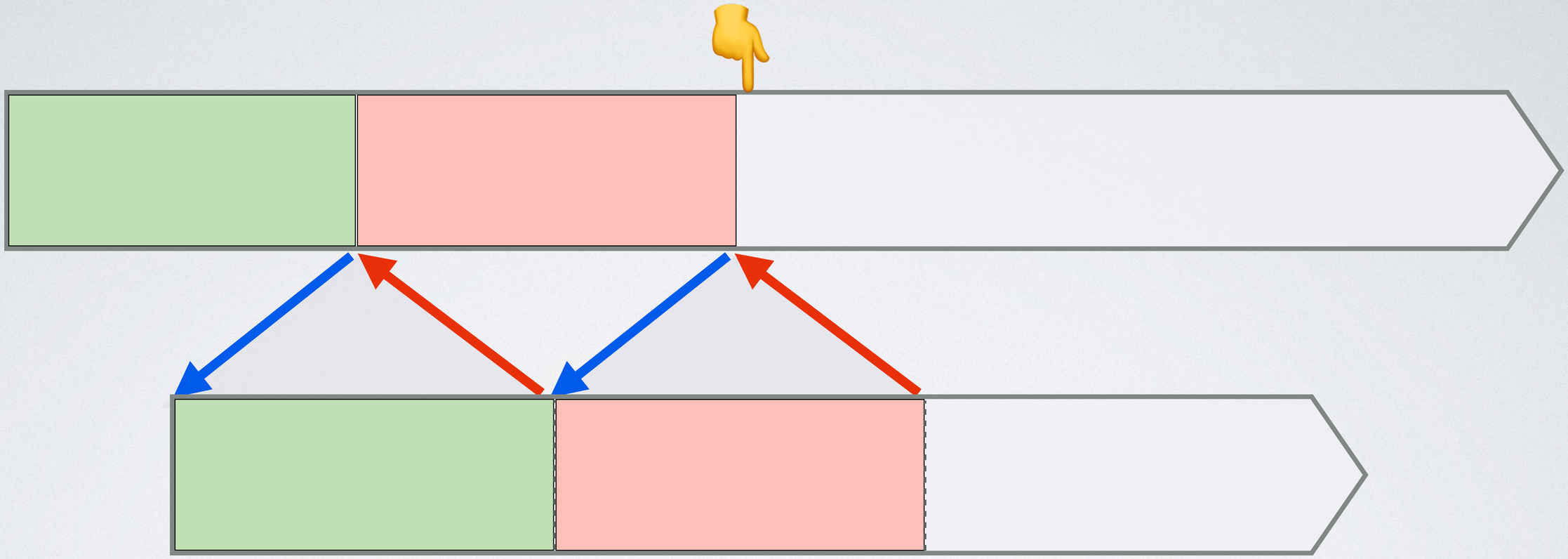

```
$v2 = yield $v1
```

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



```
yield $v3
```

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



```
yield $v3
```

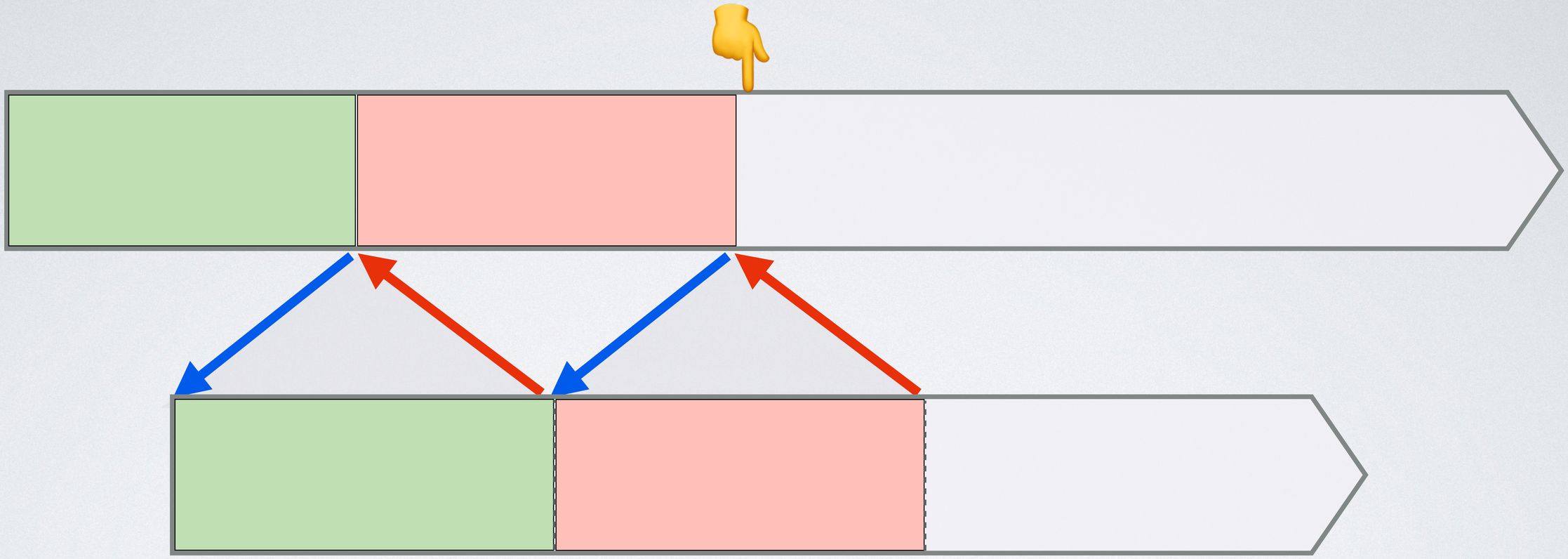
Outside

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

```
return $value;
```

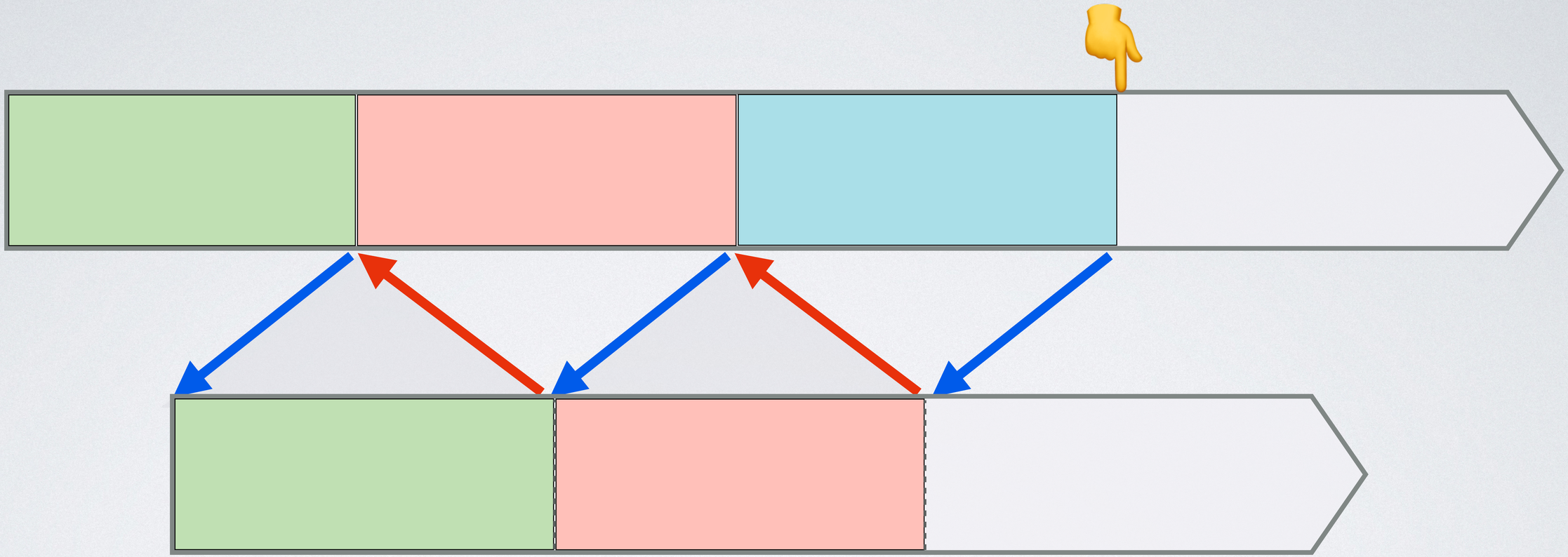
Inside

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



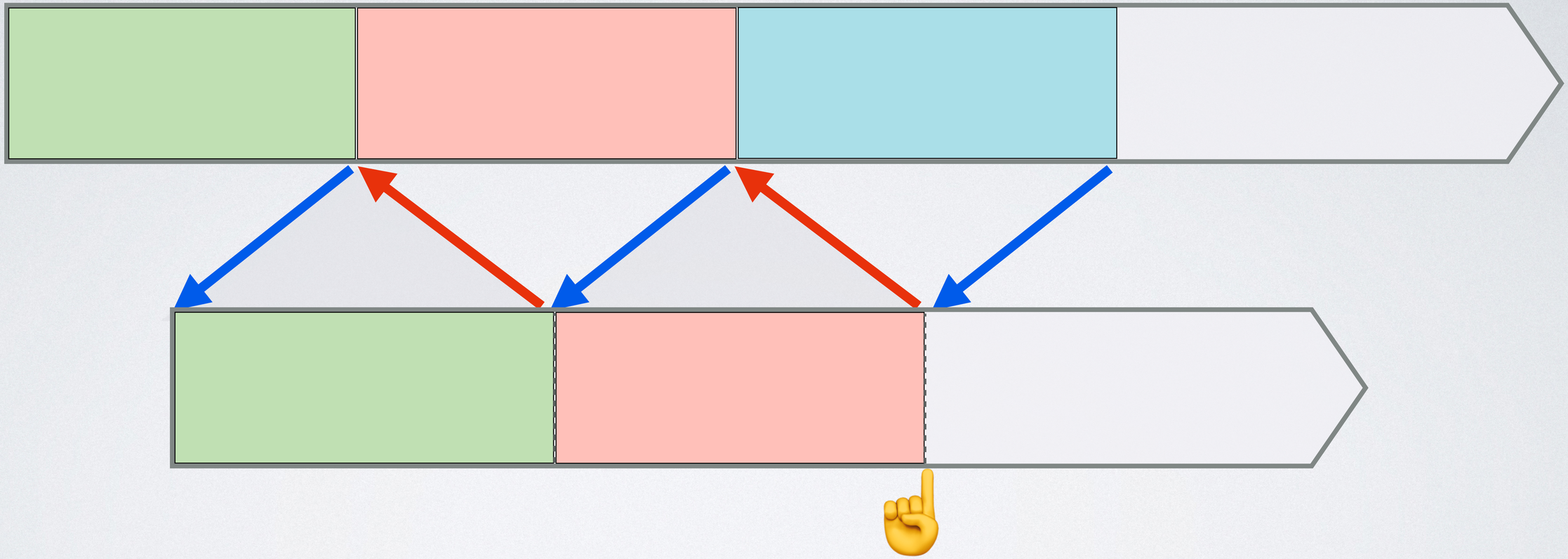
```
yield $v3
```

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



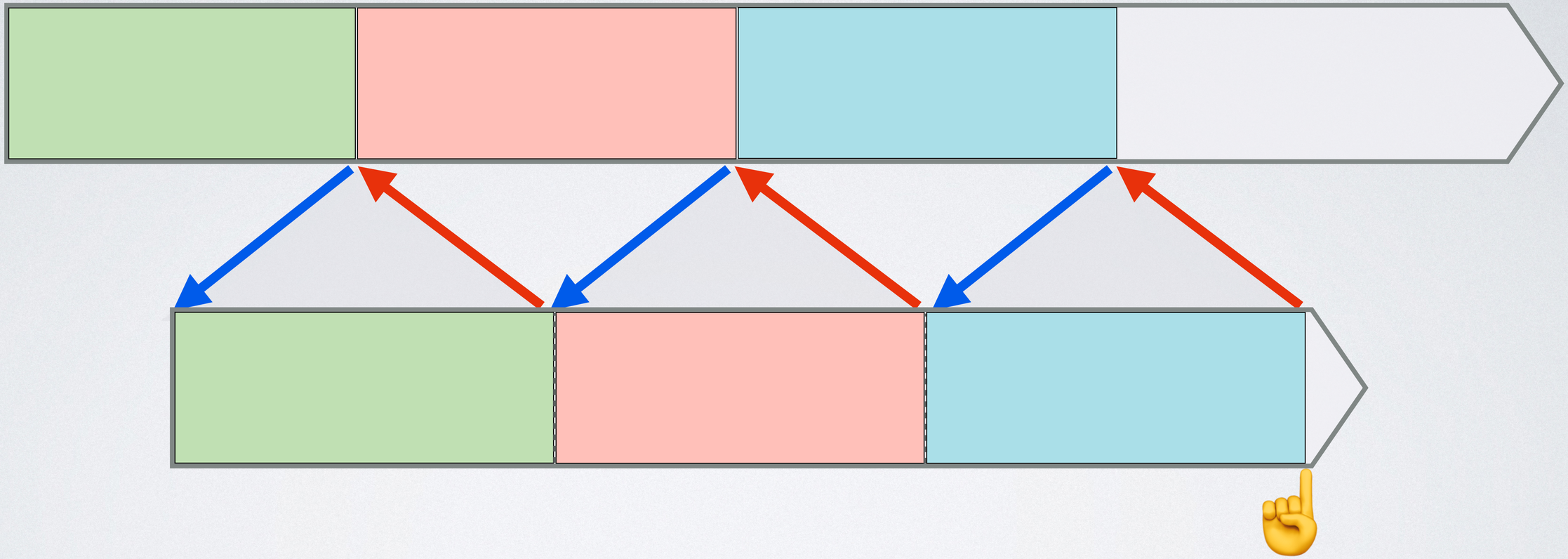
```
yield $v3
```

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



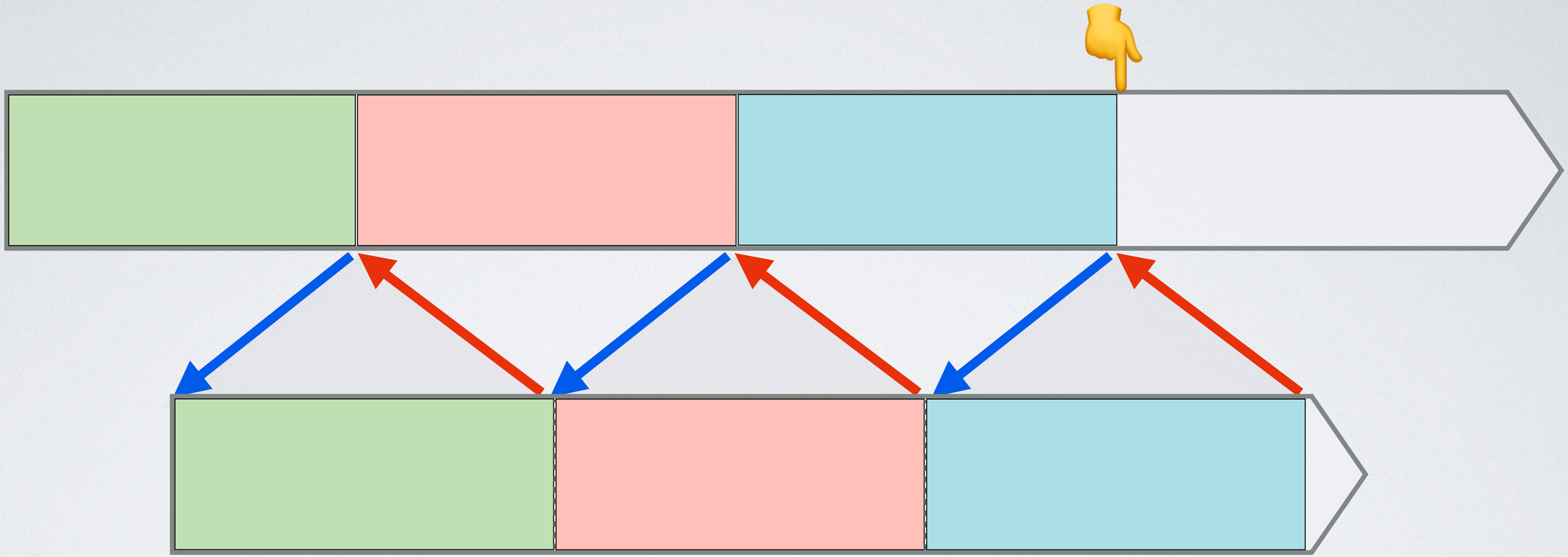
```
yield $v3
```

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



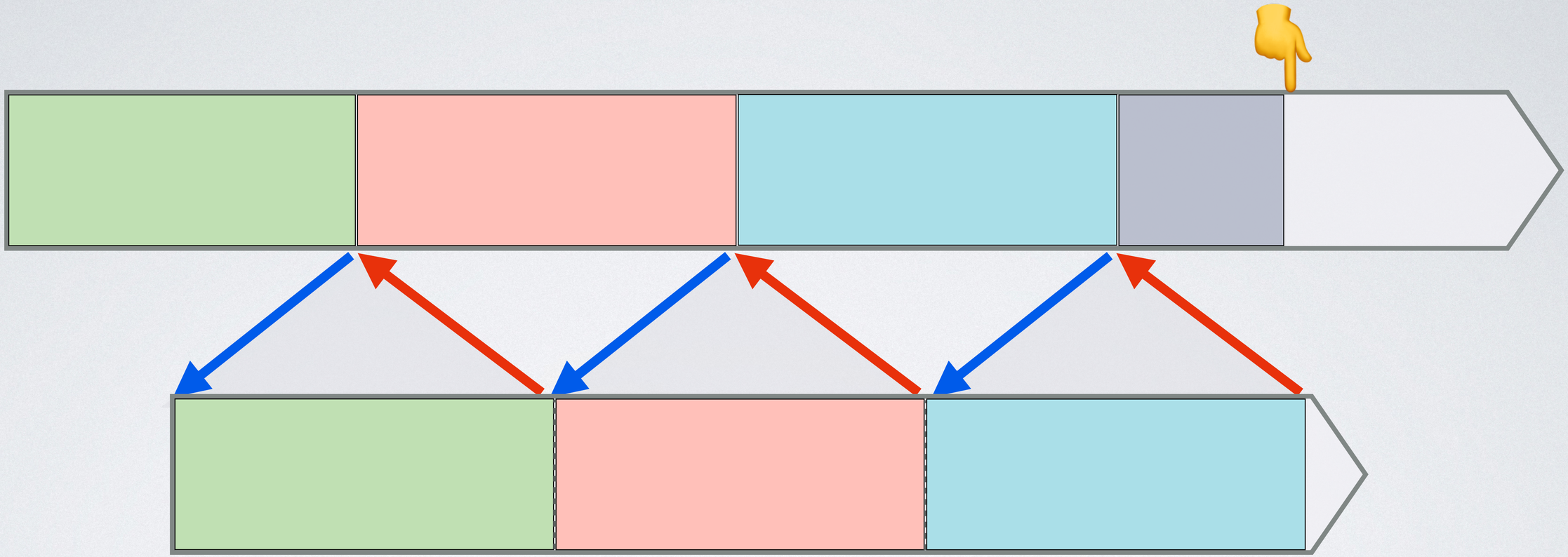
```
return $value;
```

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



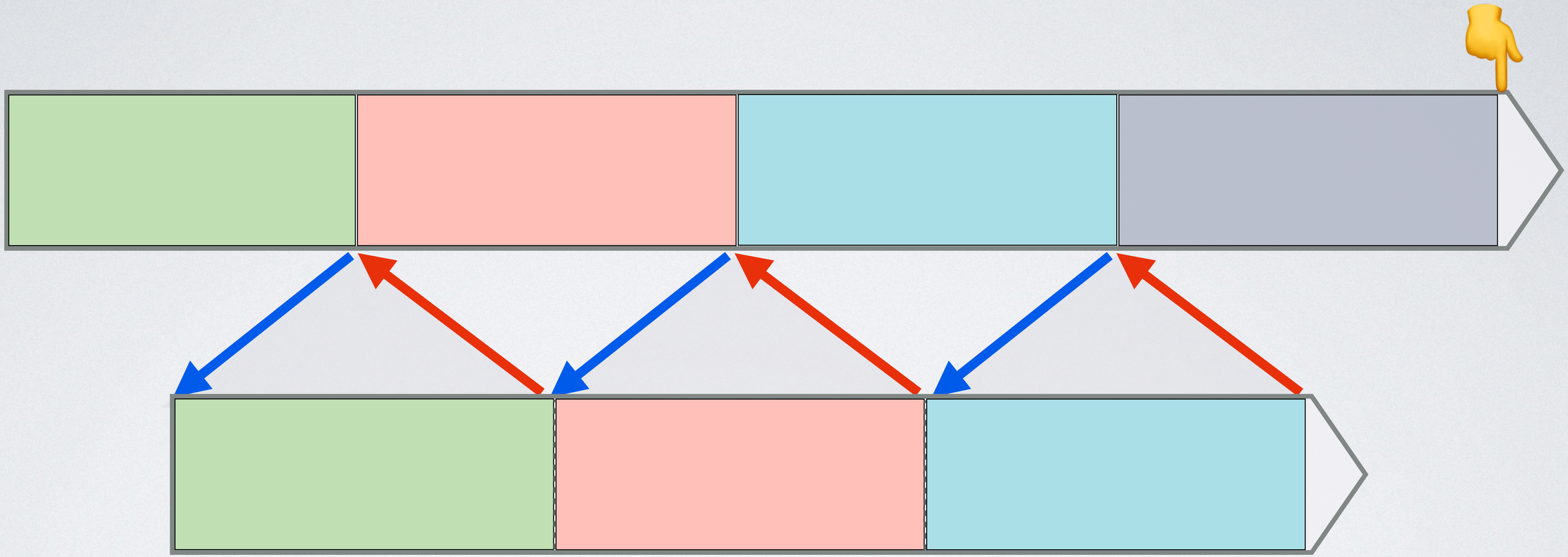
```
return $value;
```

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



```
return $value;
```

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



```
return $value;
```

Outside

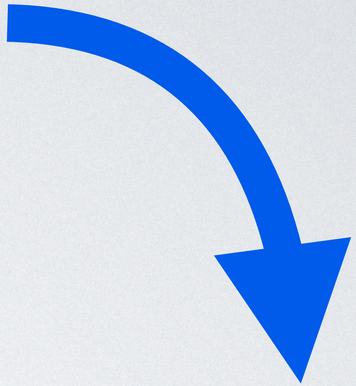
```
try {
    $generator->next();
} catch (\Exception $exception)
{}
```

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

Inside

Outside

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



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

Inside

# Why using Generators for Asynchronous Programming

Let the magic happen

BEDROCK

Benoit Viguier

@b_viguier

ConFoo.CA
DEVELOPER CONFERENCE

# Promises

# Synchronous

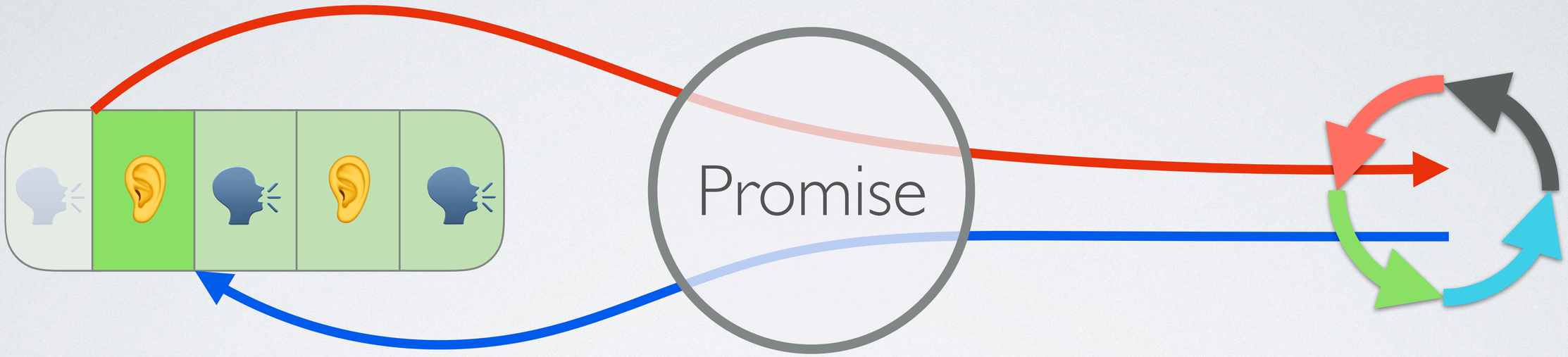
```
function talk(string $question1, string $question2)
{
    /** @var string $response */
    $response = askQuestion($question1);
    echo $response;

    /** @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
{
    // …
}
```

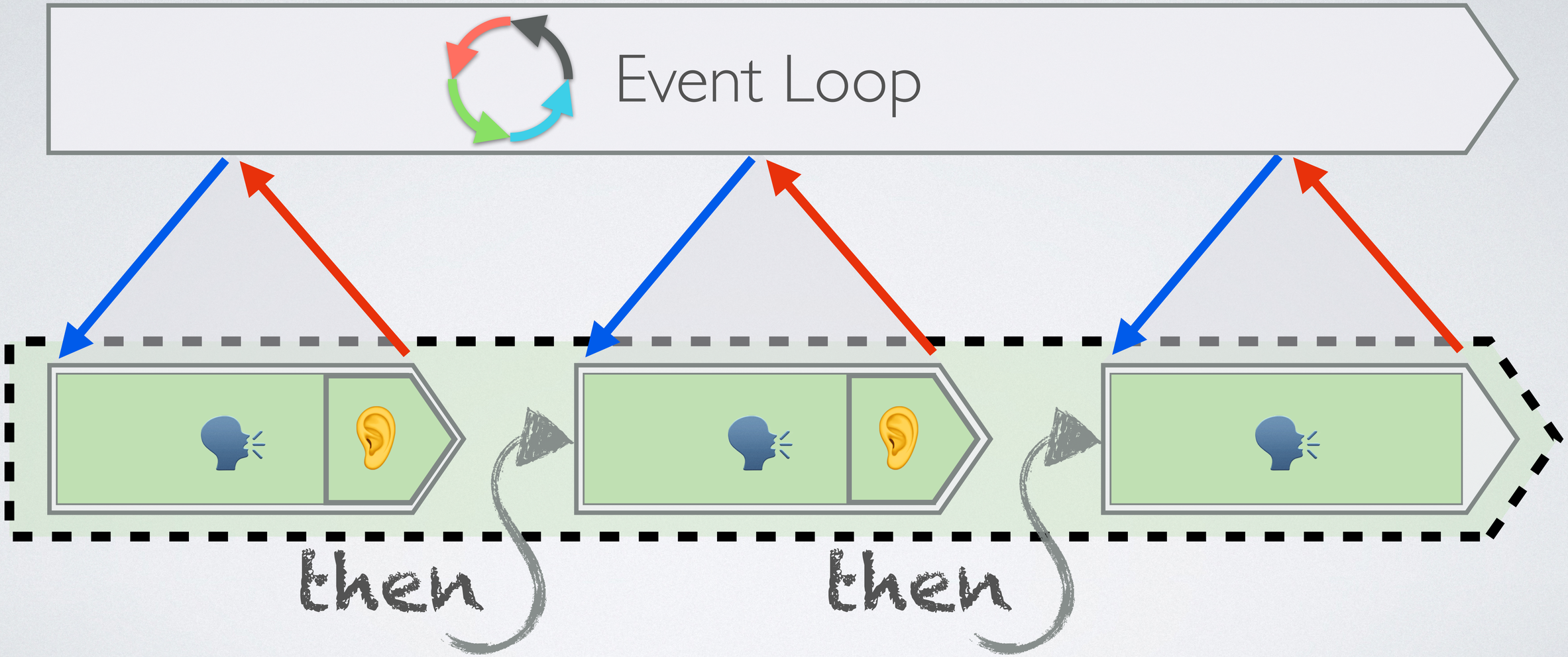
# Asynchronous

```
function talk(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

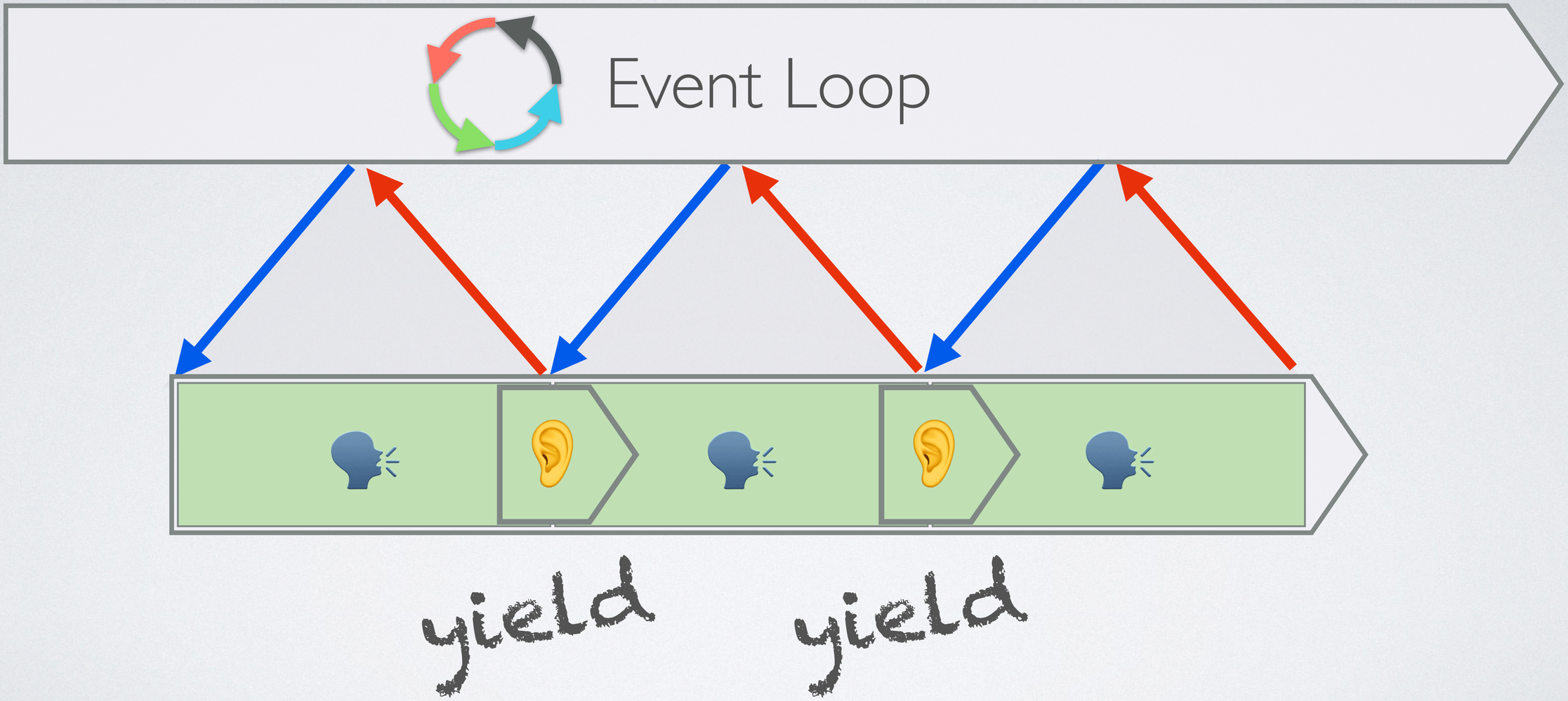
```php
function talk(string $question1, string $question2)
{
    $promise = askQuestion($question1);
    $promise->then(
        function(string $response) use($question2) {
            echo $response;

            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 talk(string $question1, string $question2)
{

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


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

# Asynchronous

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

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


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

# Asynchronous

```
function talk(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*.

BEDROCK

Benoit Viguier

@b_viguier

ConFoo.CA
DEVELOPER CONFERENCE

# 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 talk(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

# One year of asynchronous PHP in production

ConFoo.CA
DEVELOPER CONFERENCE

**Tomorrow**, 13:00

# Why and how to use generators for asynchronous programming

BEDROCK

Benoit Viguier

@b_viguier

ConFoo.CA
DEVELOPER CONFERENCE

# Thanks !

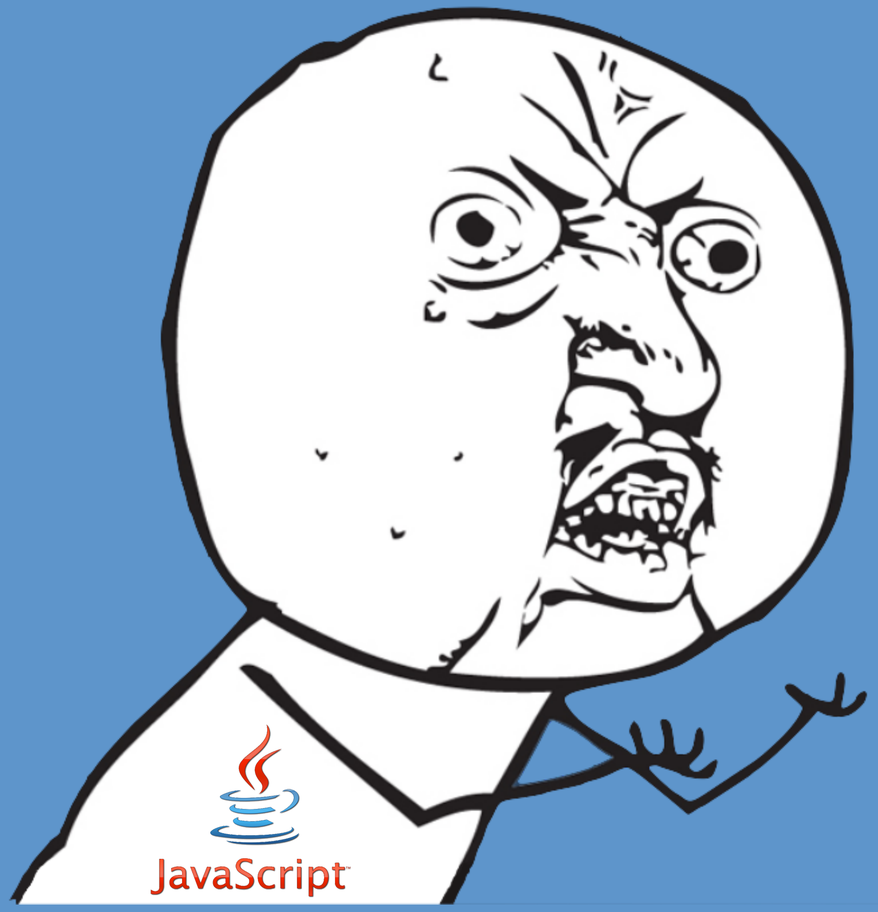
```
$answer = yield $you->ask($me);
```

BEDROCK

Benoit Viguier

@b_viguier

ConFoo.CA
DEVELOPER CONFERENCE

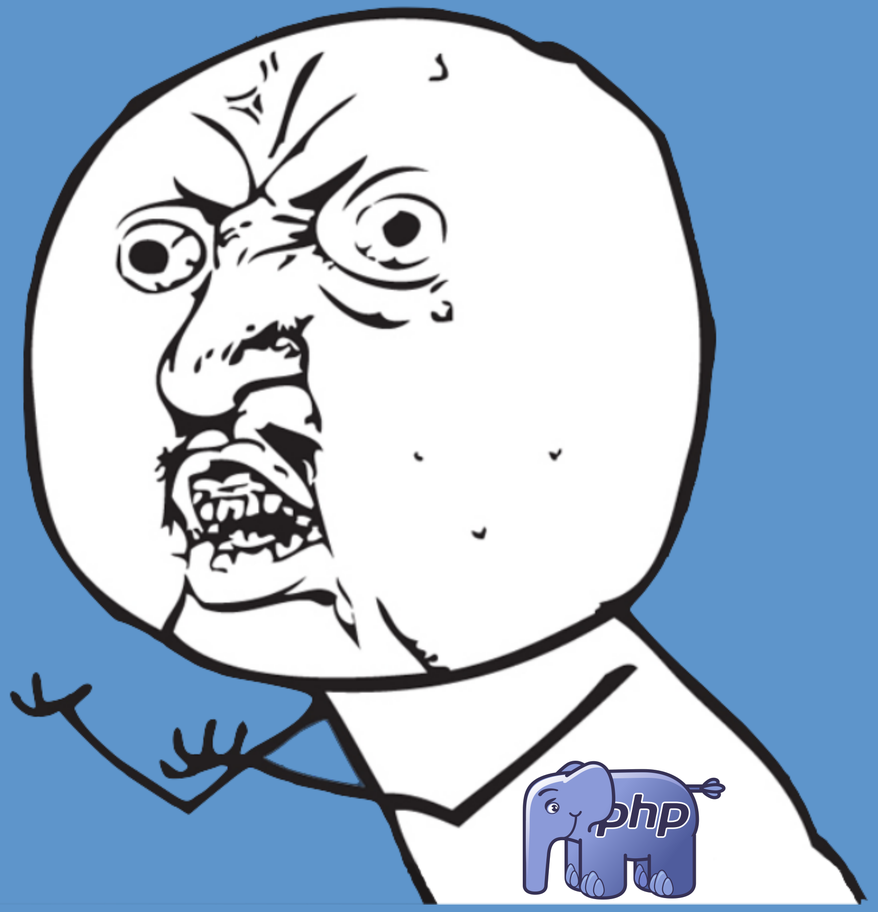



# PHP DEV

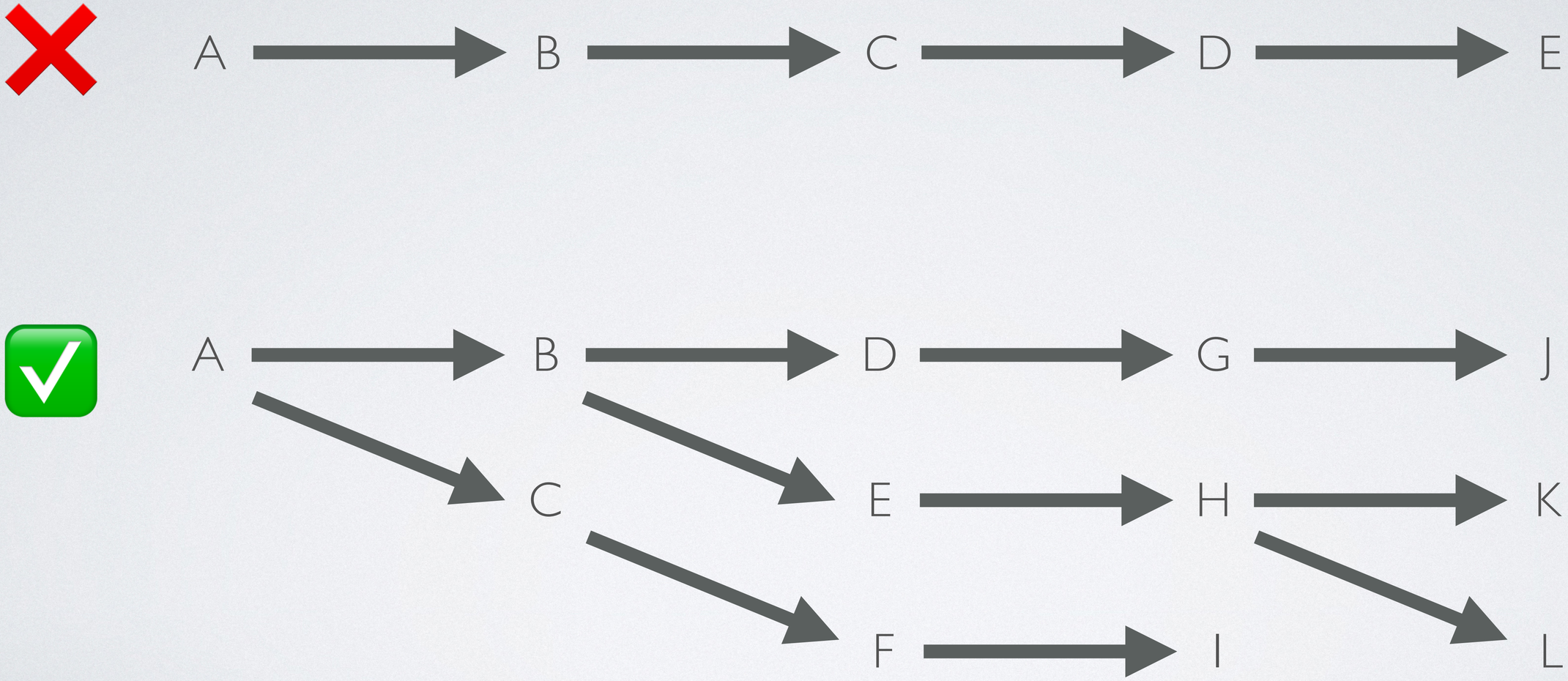
# Y U NO USE JAVASCRIPT

# PHP DEV

# Y U NO USE PHP



# Finding the good use case


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