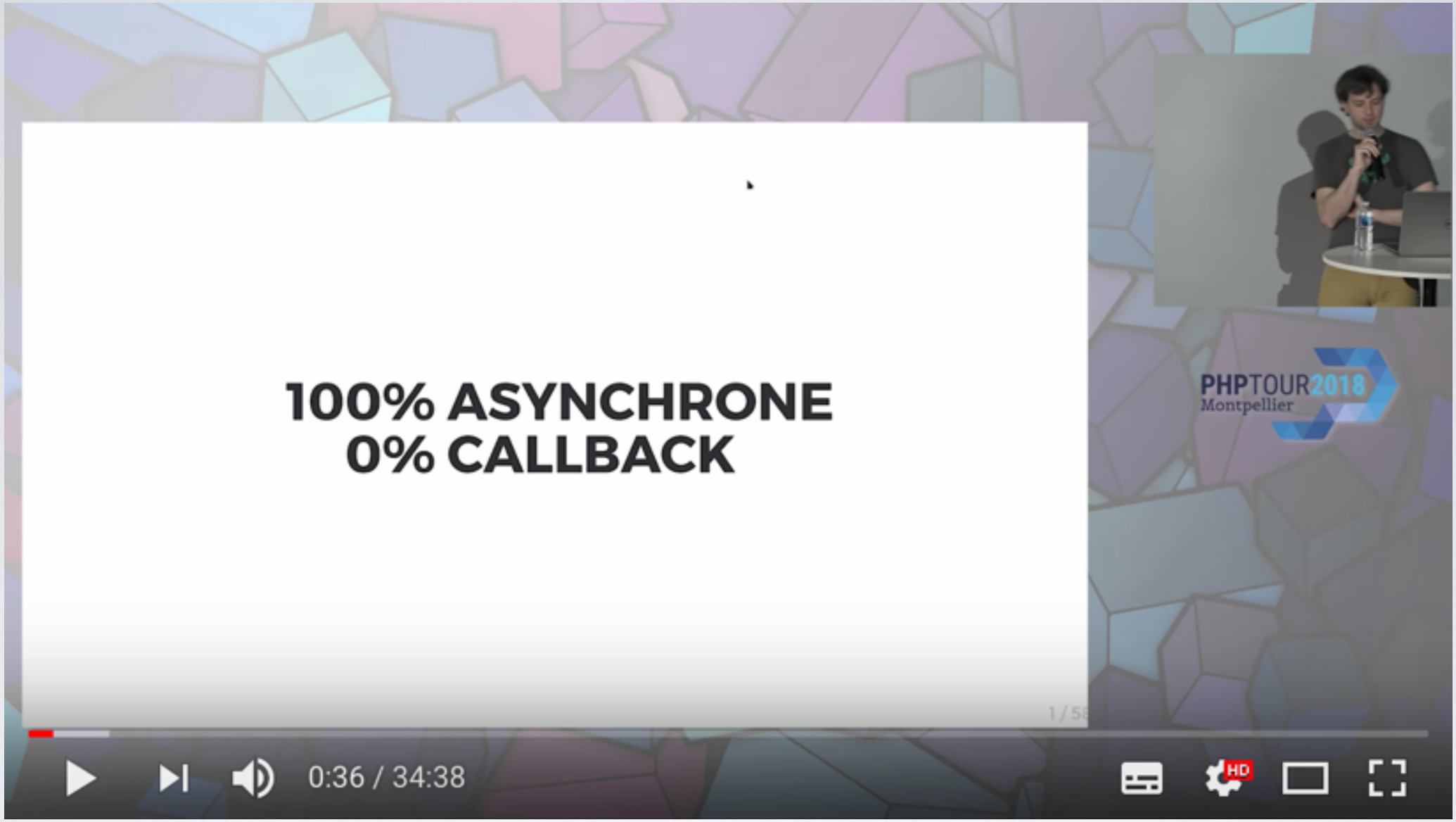
Generators & Asynchronous Computation

Benoit Viguier
@b_viguier

AFUP
12/06/2018

Trending Topic



Joël Wurtz

PhpTour 2018

Julien Bianchi

PhpTour 2016

Cooperative multitasking using coroutines (in Php!)

Nikita Popov
Blog - 2012

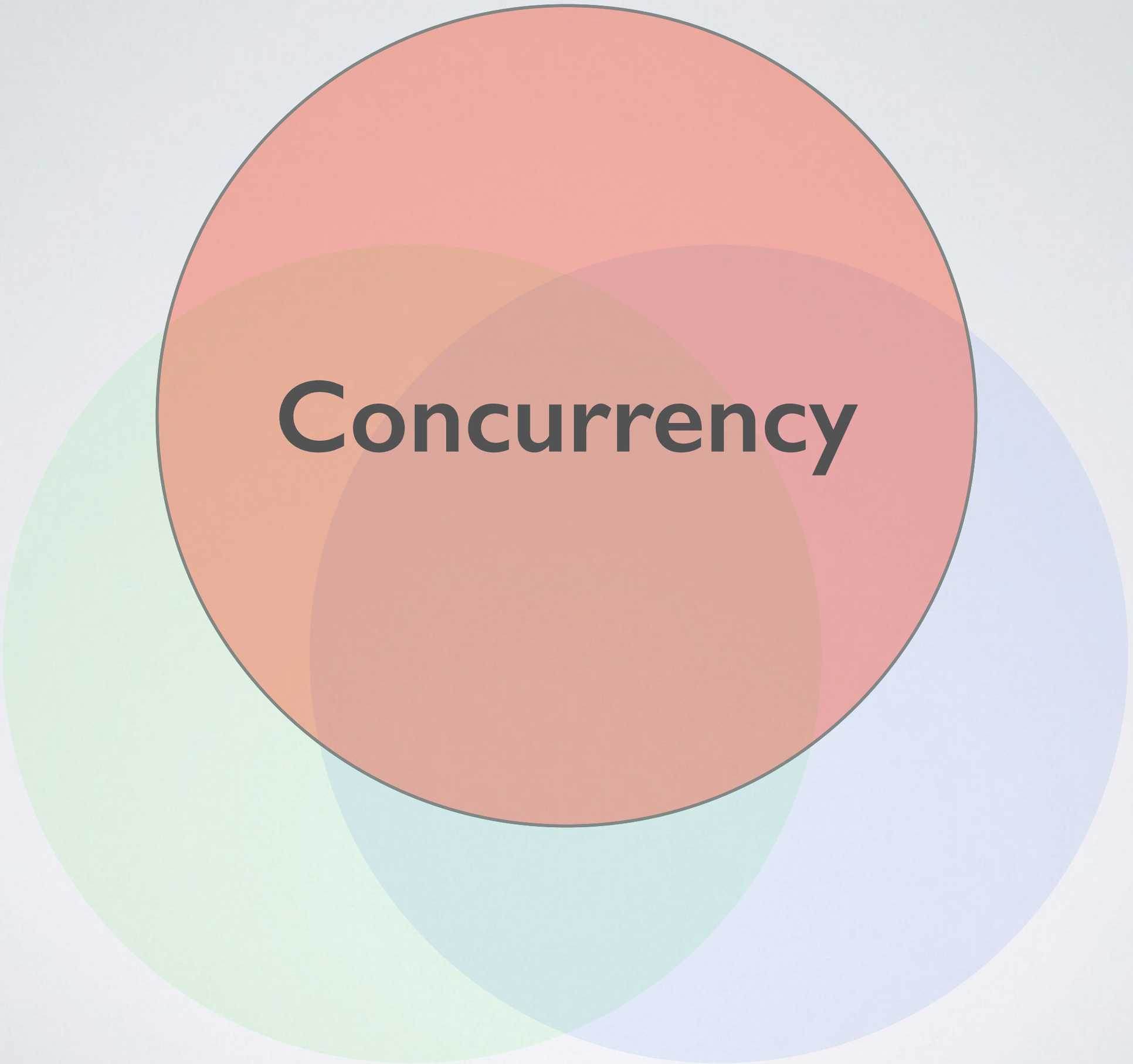
Asynchrony

≠

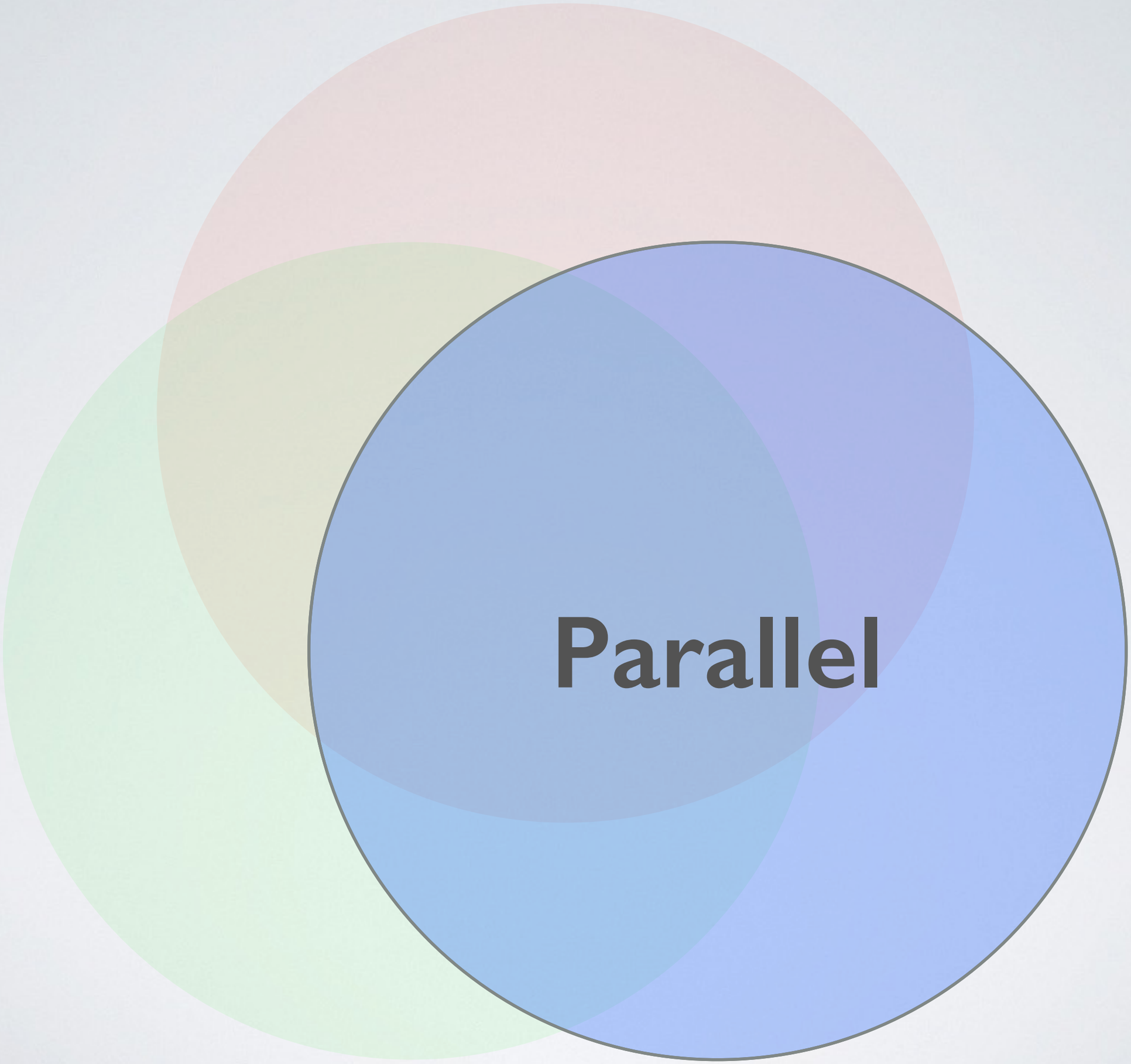
Parallel

≠

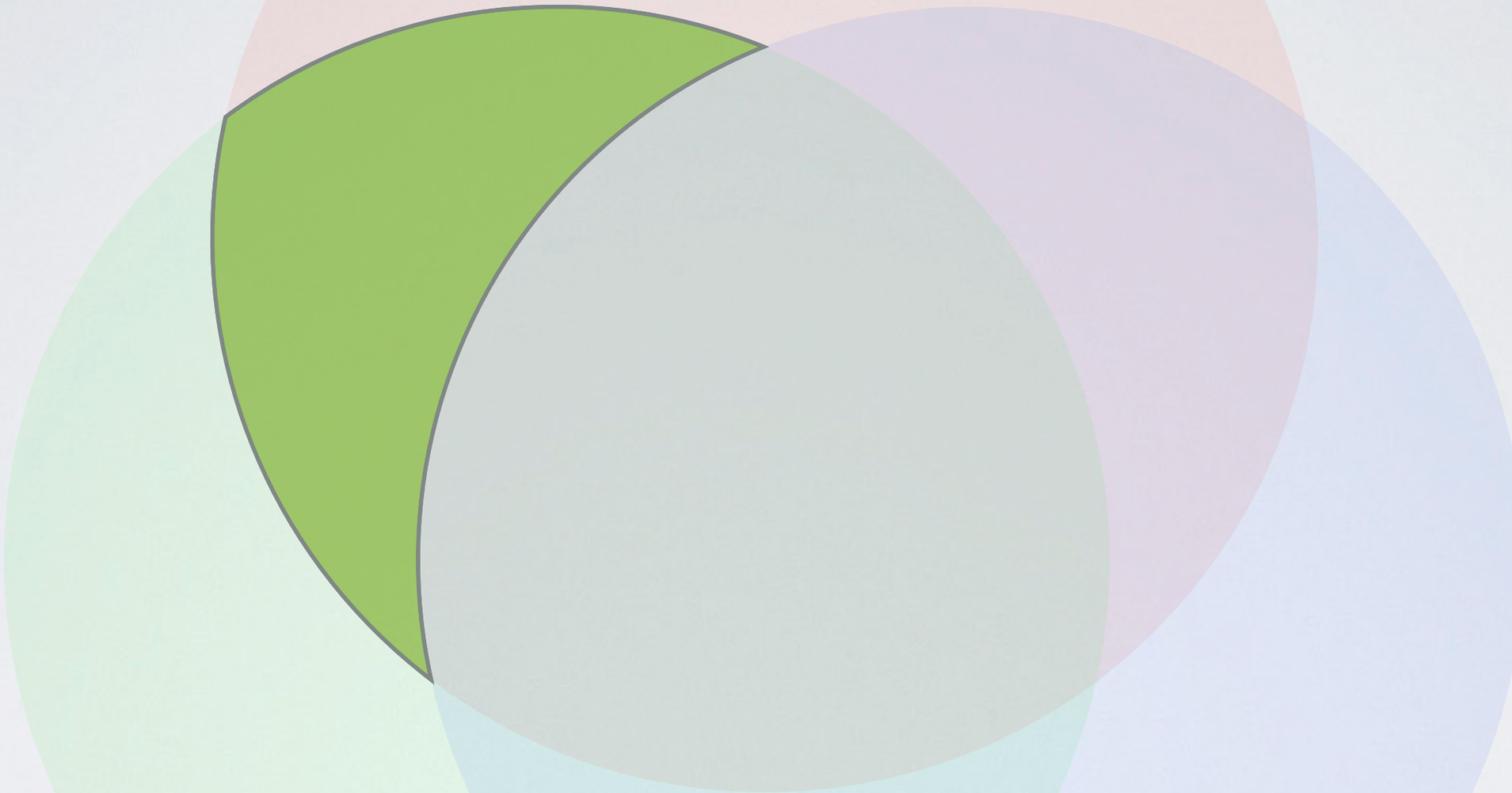
Concurrency


Concurrency

Asynchrony

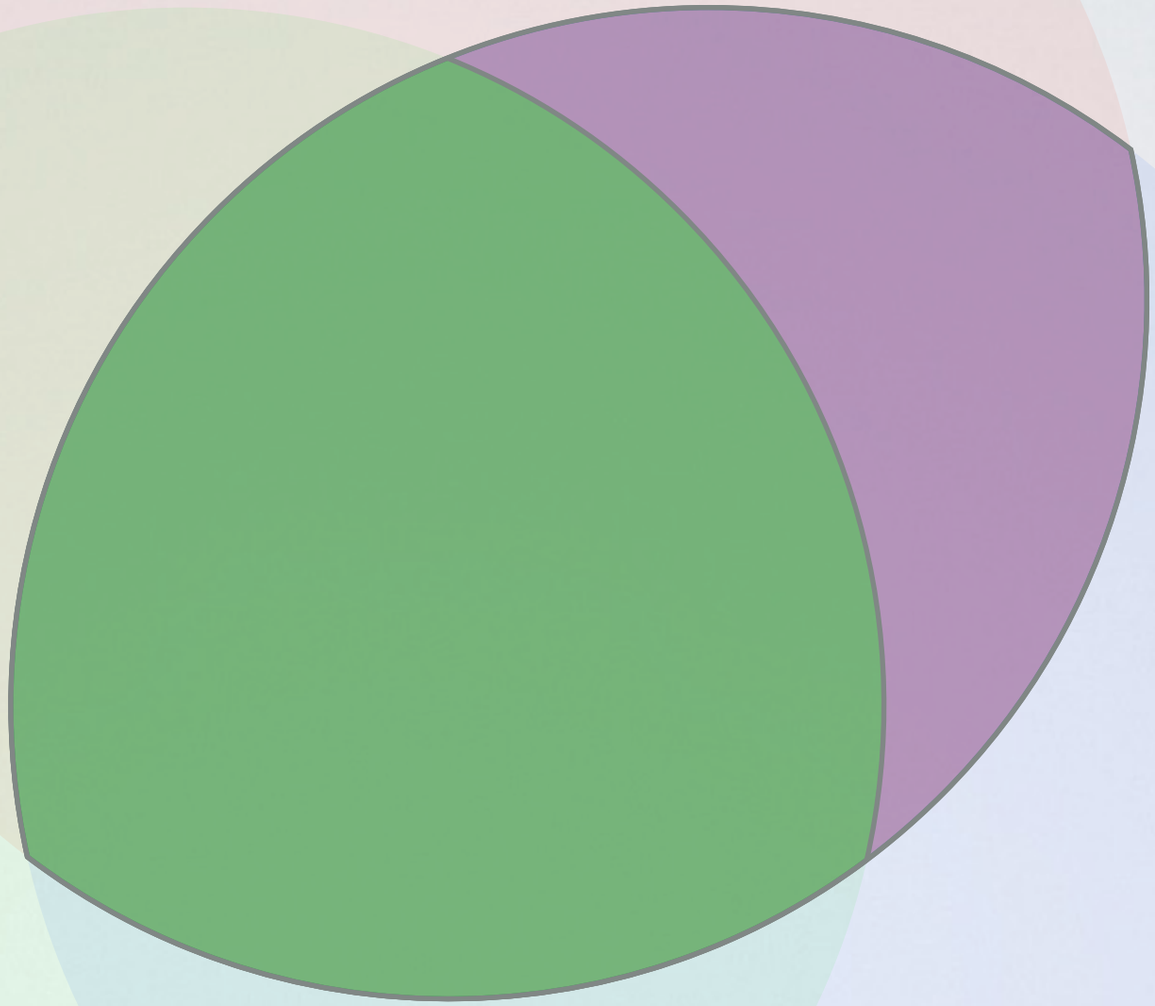

Parallel

Asynchronous Concurrency

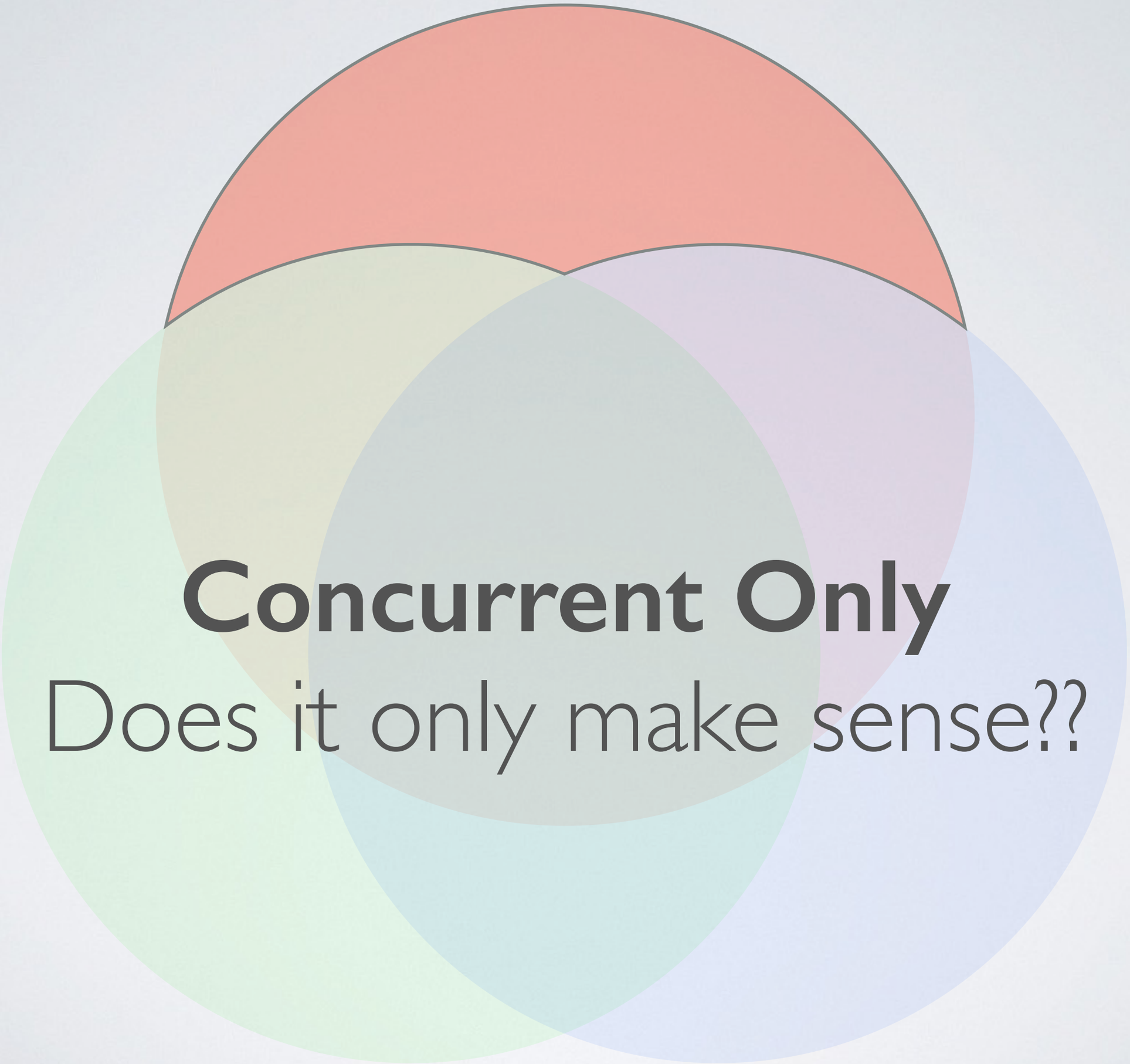


Exactly what we will talk about!

Parallel Concurrency



Not today 😉

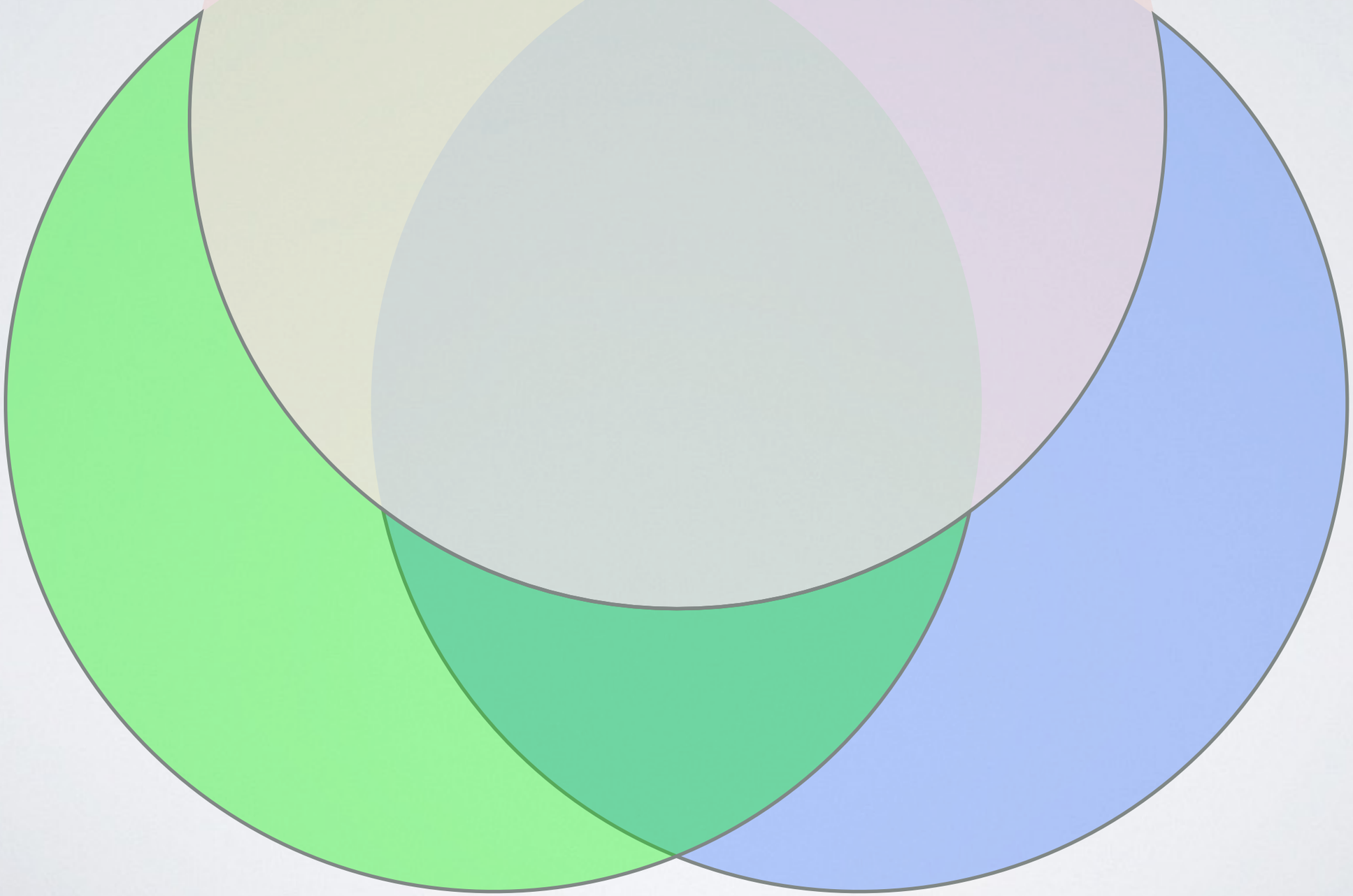


Concurrent Only

Does it only make sense??

No Concurrency

Ok, but… why would you do that?



ASYNCHRONOUS PROGRAMMING

EVERYWHERE !

What you see

What happens

How to take/release « ownership » ?

No blocking wait

EventLoop

The master *process*

```php
function run()
{
    while (!$eventLoop->isFinished()) {
        $task = $eventLoop->popTask();
        $task->run();
    }
}
```

EventLoop

≠

Http Server

Events listener

Mainly for external events

PARENTAL ADVISORY

EXPLICIT JAVASCRIPT

```
$element.addEventListener ("mousedown" , onMouseDown , false);


function onMouseDown () {
    // Init and track motion
    // Code here…
    document.addEventListener ("mousemove" , onMouseMove , false);
}


function onMouseMove (event) {
    // Move logic here
    // Code here…
    document.addEventListener ("mouseup" , onMouseUp , false);
}


function onMouseUp () {
    // Finish motion tracking
    // Code here…
    document.removeEventListener ("mousemove" , onMouseMove , false);
    document.removeEventListener ("mouseup" , onMouseUp , false);
}
```

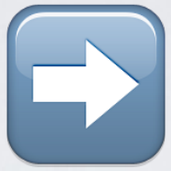
Promises

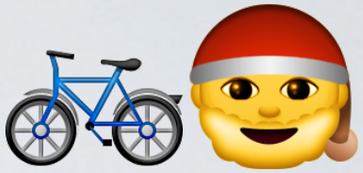
Promises/A+

Promises

Concept

Synchronous





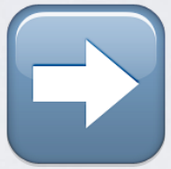


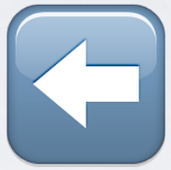


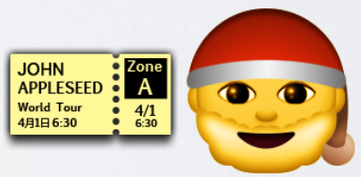




Asynchronous Promise







Promises

Example

```php
// https://reactphp.org/promise/#how-to-use-deferred

getAwesomeResultPromise()
    ->then(
        function ($value) {
  // Deferred resolved, do something with $value
        },
        function ($reason) {
// Deferred rejected, do something with $reason
        },
        function ($update) {
// Progress notification triggered, do something
// with $update
        }
    );
```

```php
// https://reactphp.org/promise/#mixed-resolution-and-rejection-forwarding

getAwesomeResultPromise()
    ->then(function ($x) {
        return $x + 1;
    })
    ->then(function ($x) {
        throw new \Exception($x + 1);
    })
    ->otherwise(function (\Exception $x) {
        // Handle the rejection, and don't propagate.
        // This is like catch without a rethrow
        return $x->getMessage() + 1;
    })
    ->then(function ($x) {
        echo 'Mixed '.$x; // 4
    });
```

```php
// https://reactphp.org/http-client/#example

$loop = React\EventLoop\Factory::create();
$client = new React\HttpClient\Client($loop);

$request = $client->request('GET', 'https://github.com/');
$request->on('response', function ($response) {
    $response->on('data', function ($chunk) {
        echo $chunk;
    });
    $response->on('end', function () {
        echo 'DONE';
    });
});
$request->on('error', function (\Exception $e) {
    echo $e;
});
$request->end();
$loop->run();
```

Promises

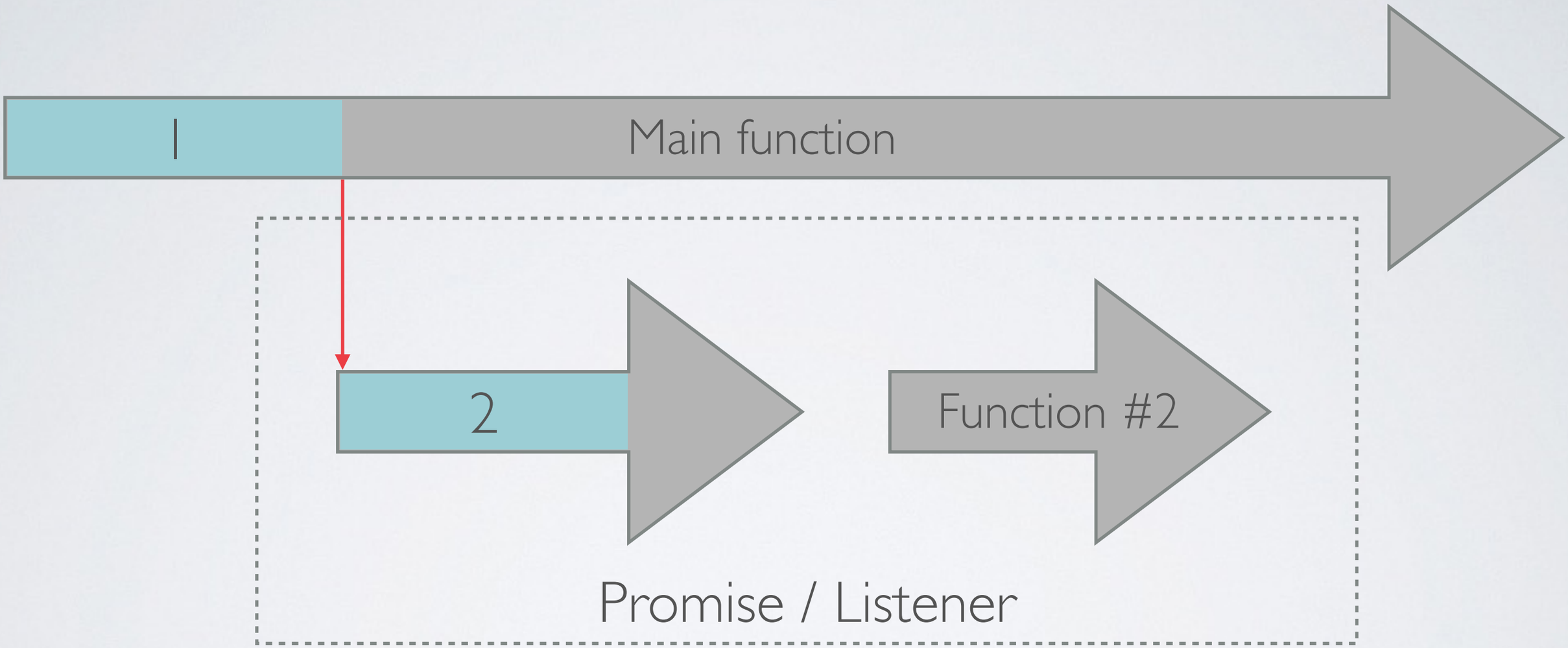
Workflow

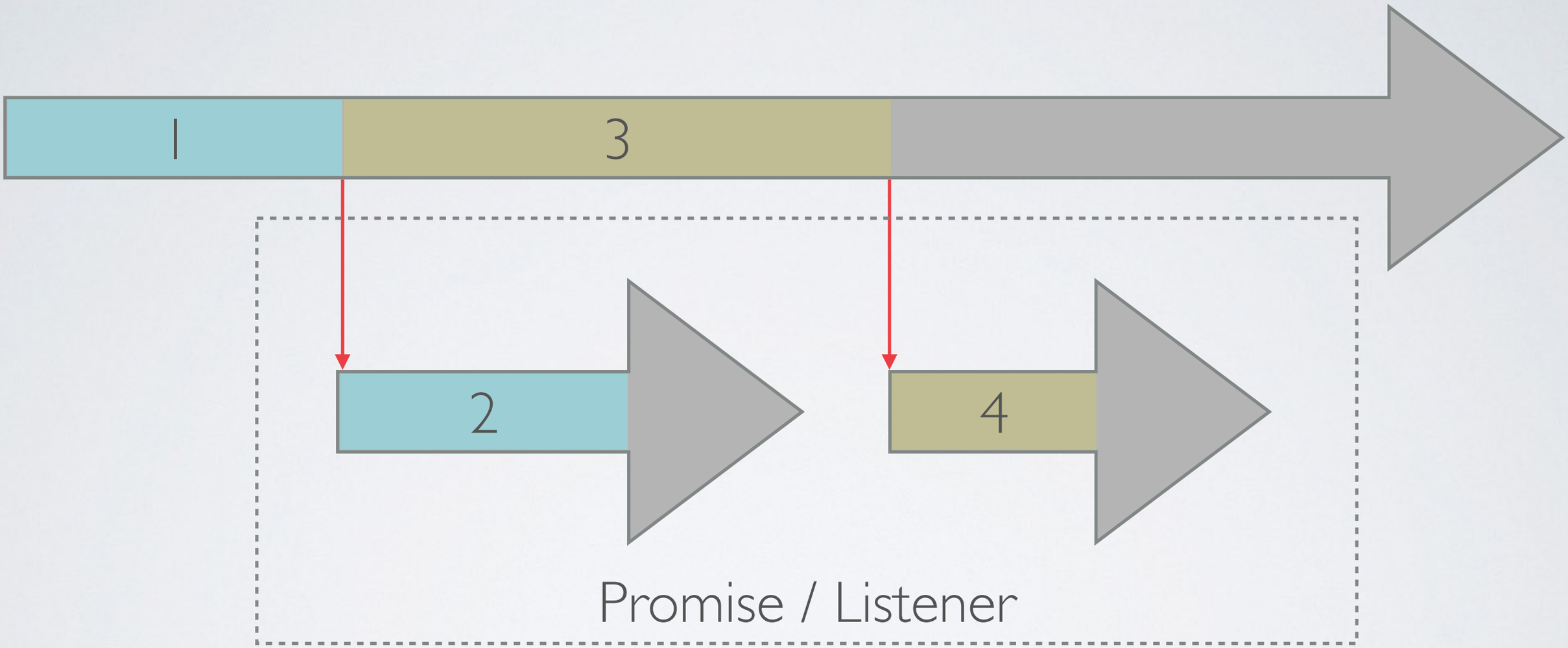
Main function

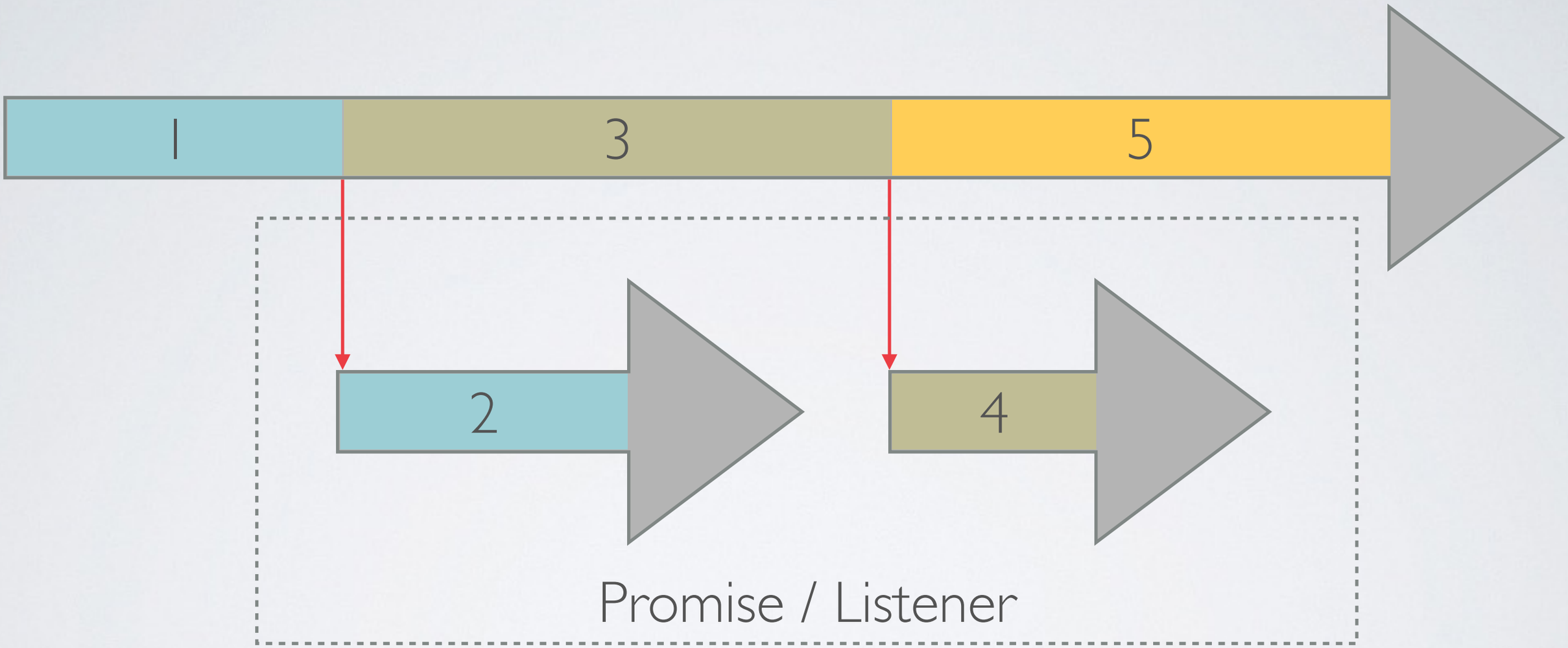
Function #1

Function #2

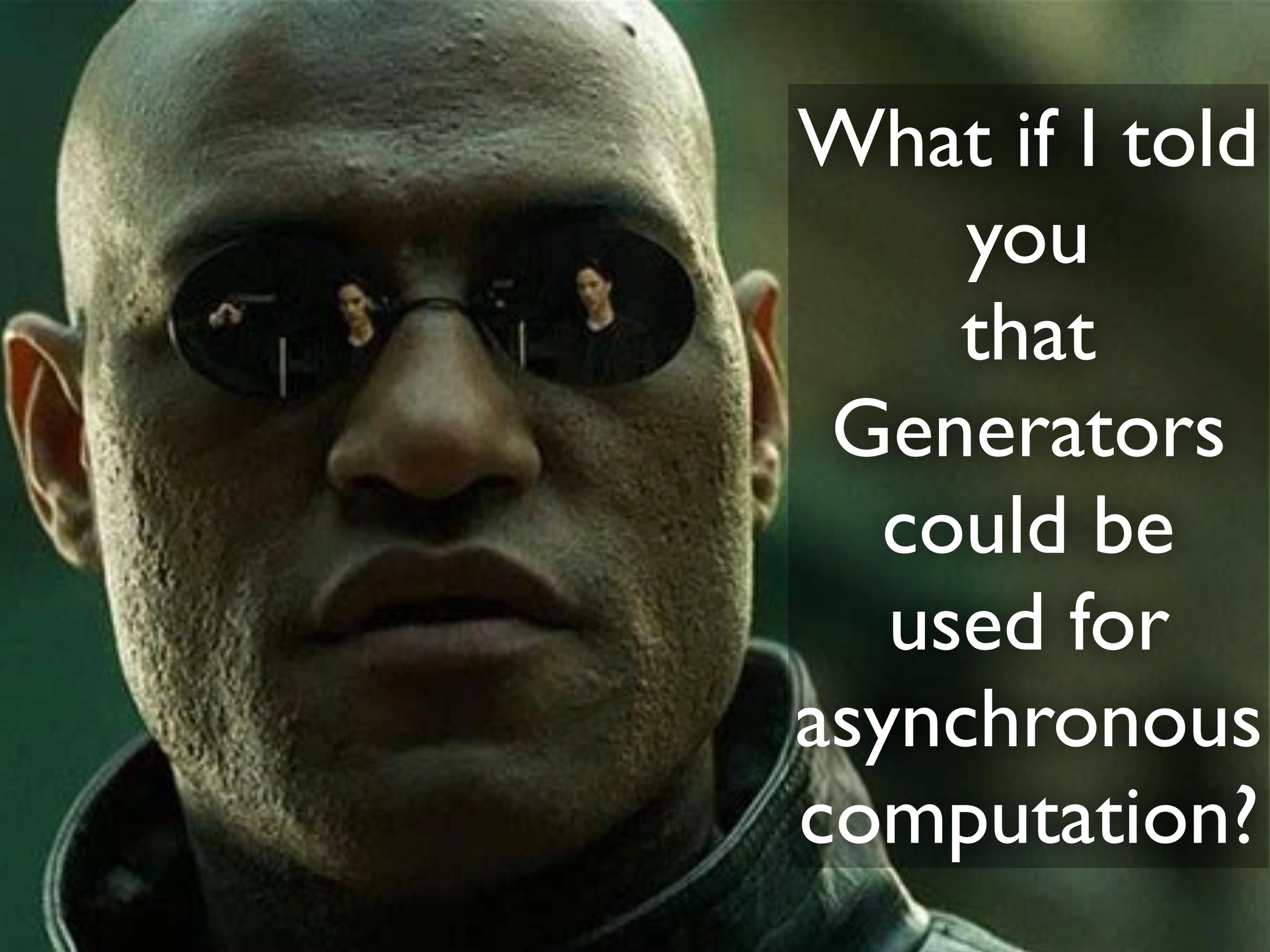
Promise / Listener


1
Main function
2
Function #2
Promise / Listener


1
3
2
4
Promise / Listener


1
3
5
2
4
Promise / Listener

Not so bad…


What if I told you that Generators could be used for asynchronous computation?

« Generators in JavaScript -- especially when combined with Promises -- are a very powerful tool for asynchronous programming as they mitigate -- if not entirely eliminate -- the problems with callbacks, such as ***Callback Hell*** *and Inversion of Control.*

This pattern is what async functions are built on top of. *»*

-MDN Web docs

Generators and Coroutines

```php
// https://secure.php.net/manual/en/language.generators.syntax.php

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";
}
```

***Generators**, also known as semicoroutines, are also a generalisation of subroutines, but are **more limited than coroutines**.*

***Coroutines** are computer-program components that generalize subroutines for **non-preemptive multitasking**, by allowing multiple entry points for suspending and resuming execution at certain locations.*

Wikipedia

Generators

Available operations

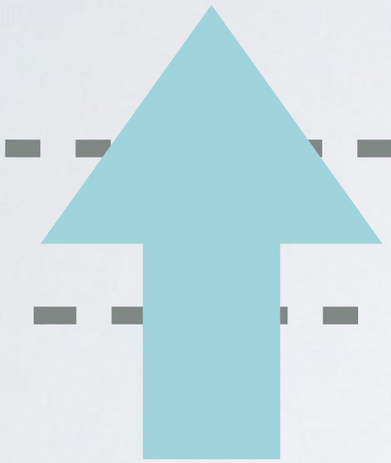
```php
final class Generator implements Iterator {

    function rewind() {}

    function valid(): bool {}

    function current() {}

    function key() {}

    function next() {}

    function send($value) {}

    function throw(Throwable $exception) {}

    function getReturn() {}
}
```

Master Coroutine

Child Coroutine

Master Coroutine

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



Child Coroutine

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

Master Coroutine

```
$g->send($a);
$g->next();
```

Child Coroutine

```
$a = yield;
```

Master Coroutine

```
$g->throw($e);
```

Child Coroutine

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

Master Coroutine

```
$g->valid();
$r = $g->getReturn();
```

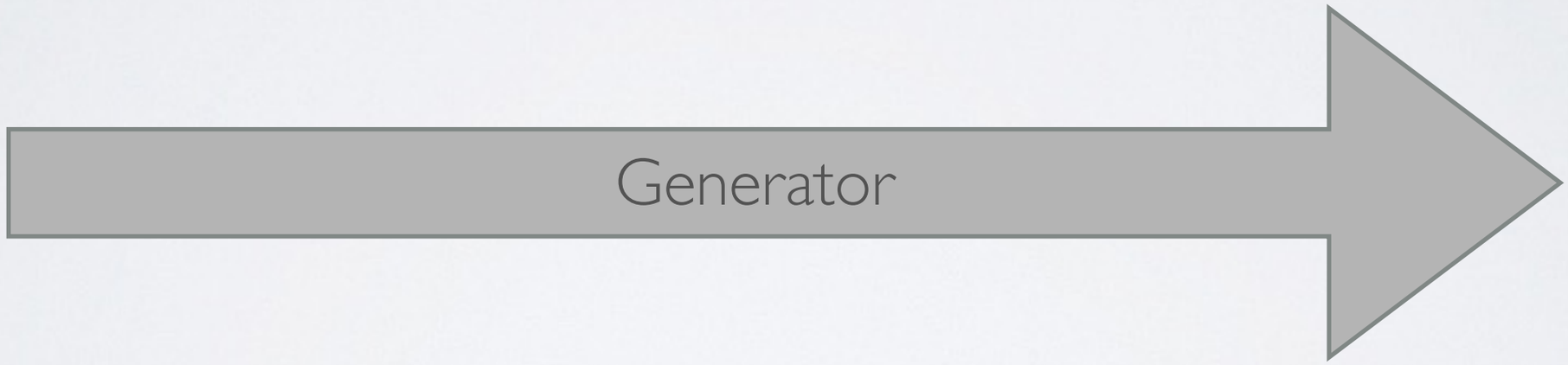
Child Coroutine

```
return $r;
```

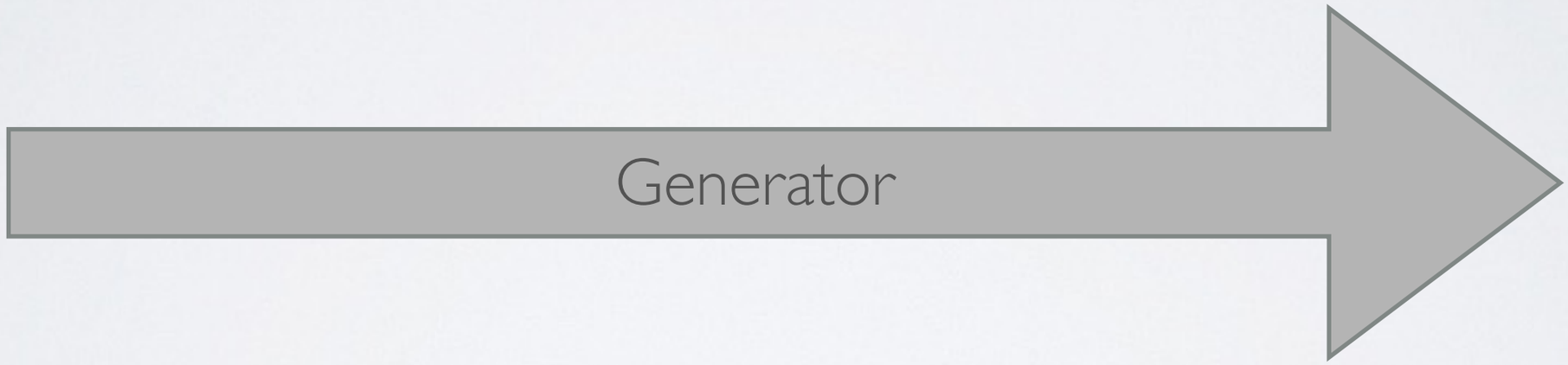
Example #1

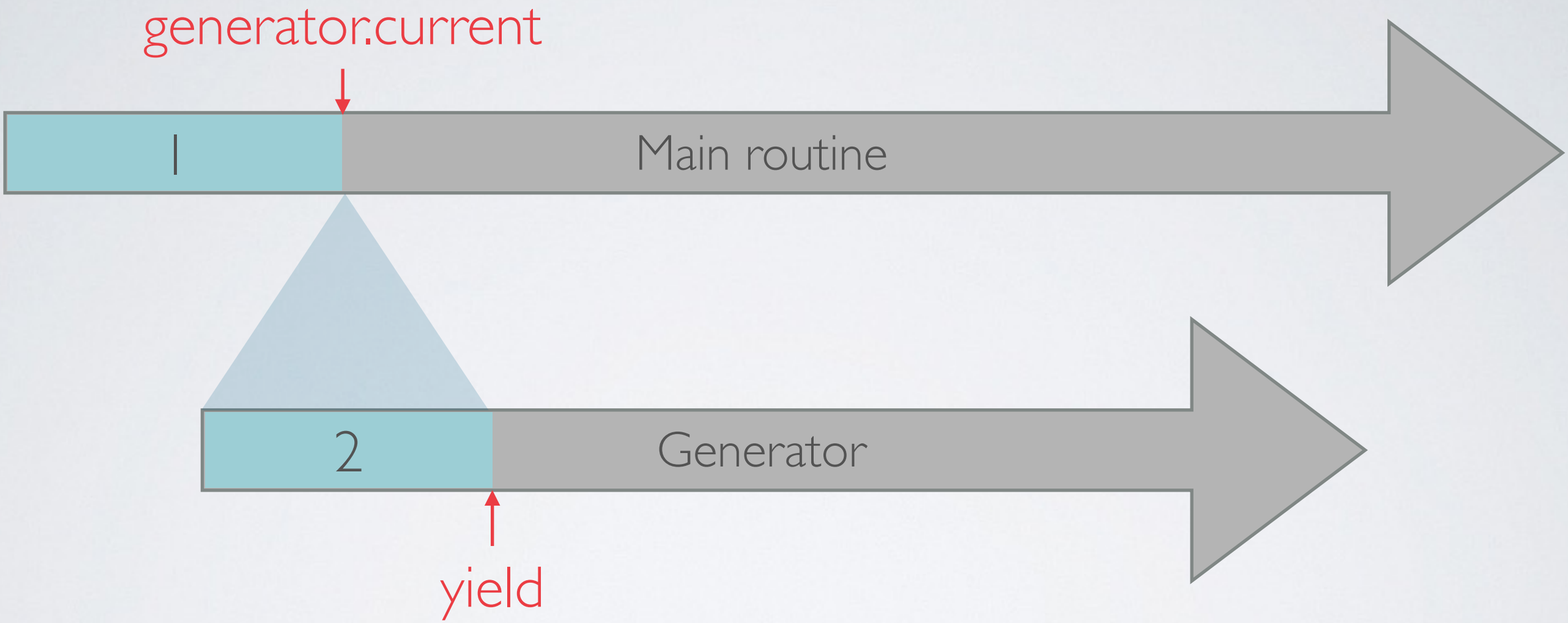
Simple workflow

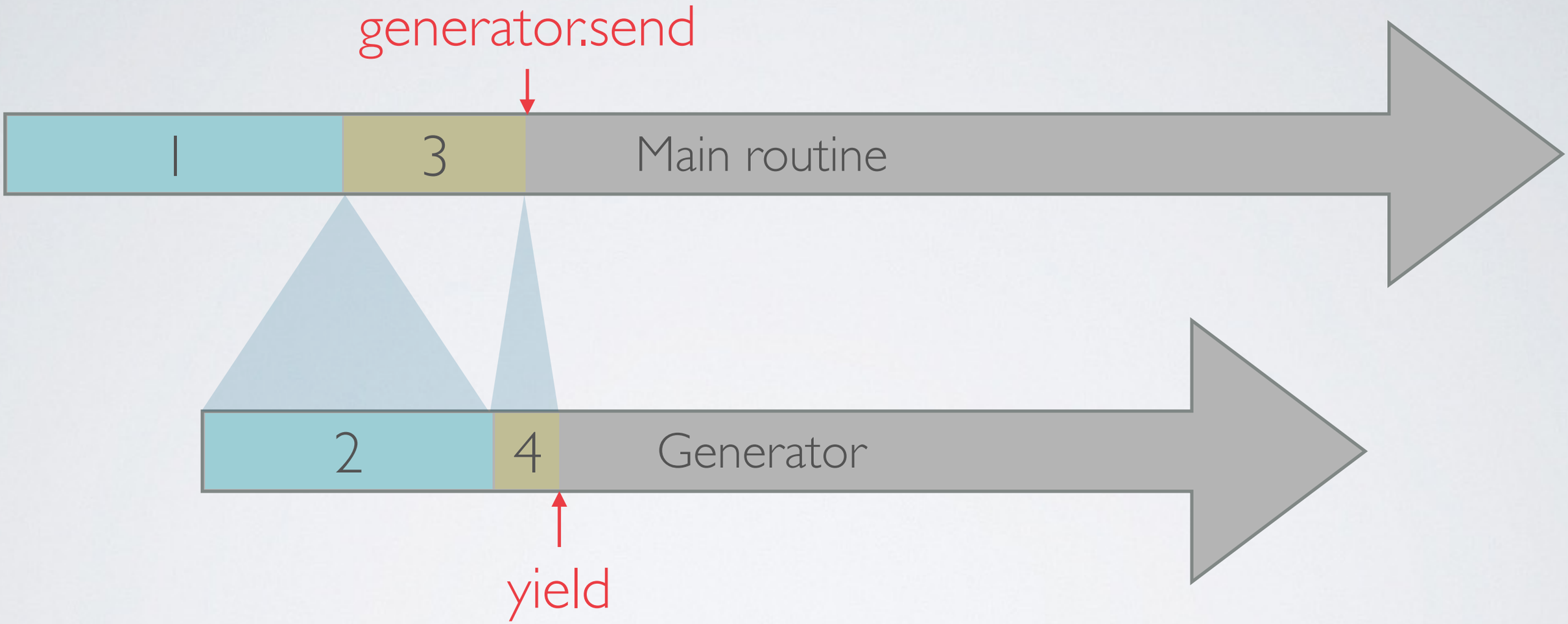

Main routine

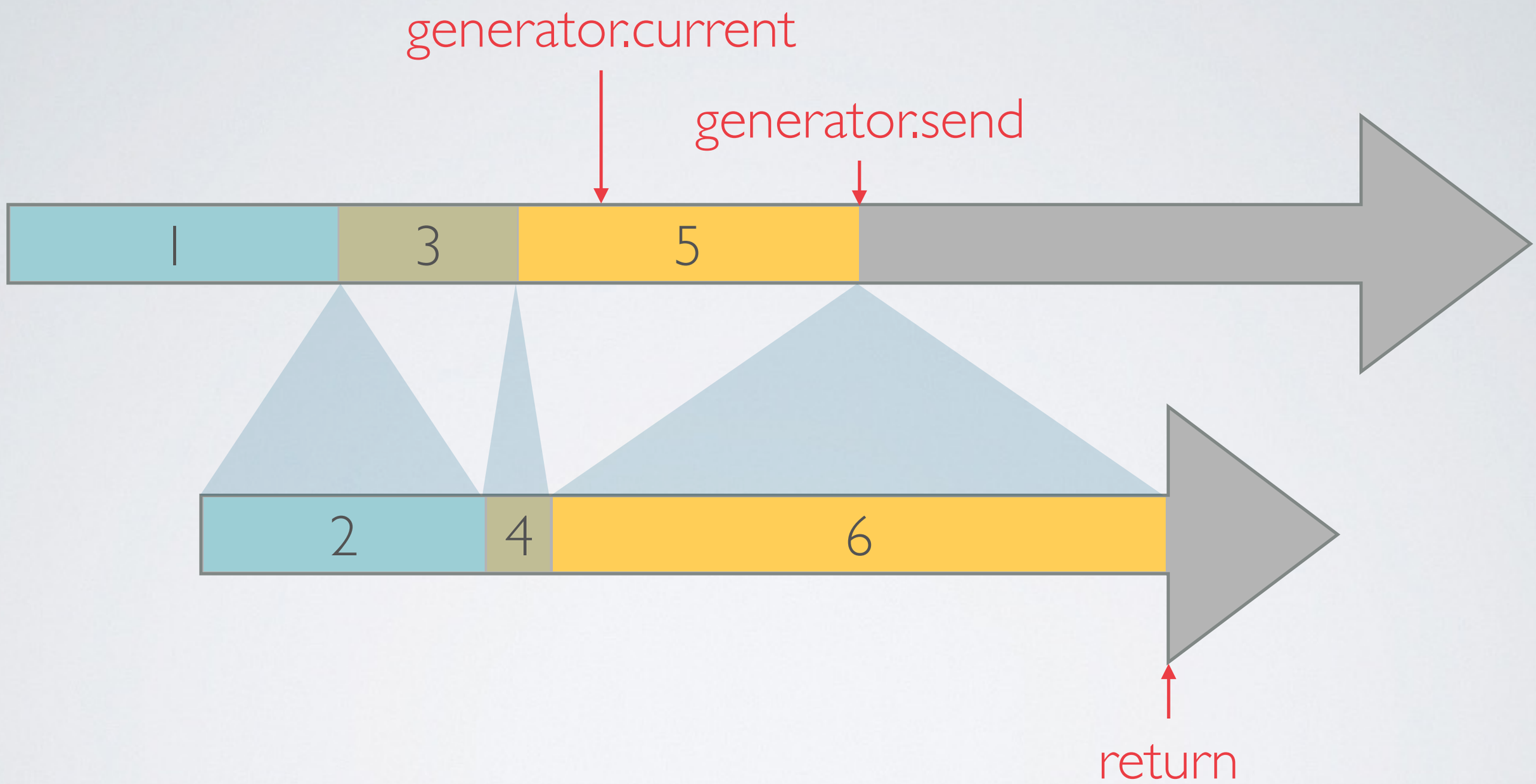

Generator

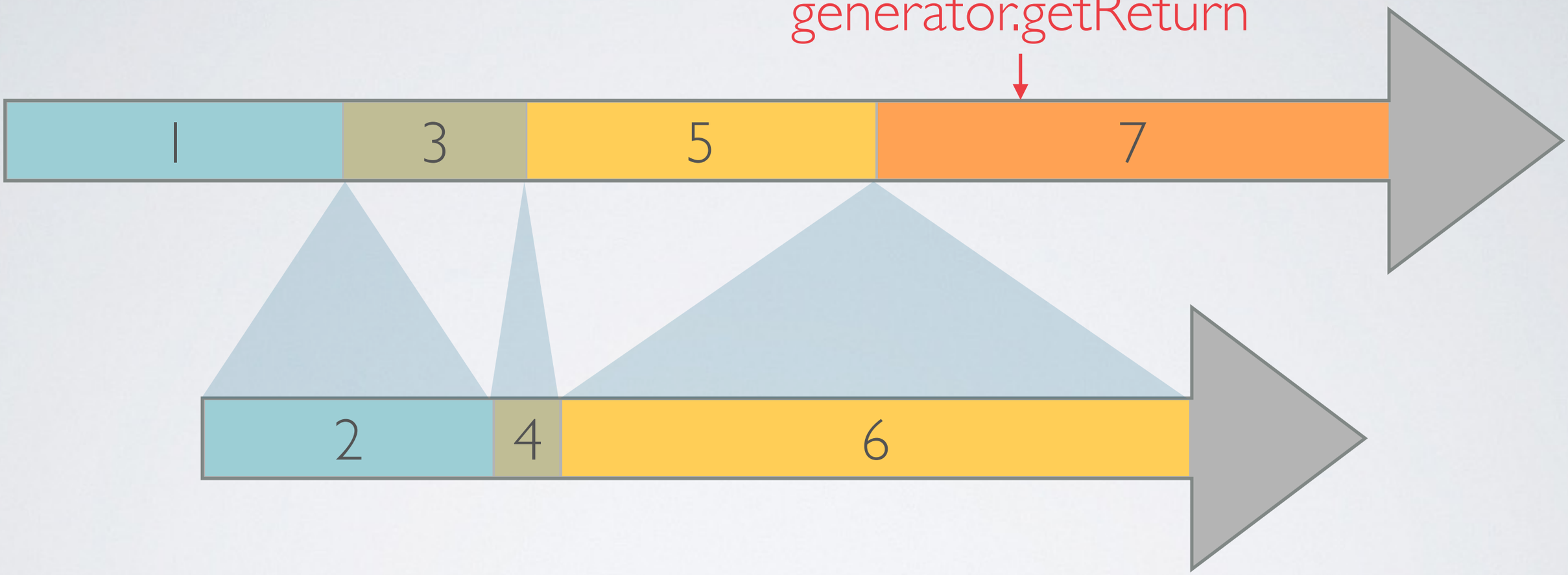

generator.current
|
Main routine


Generator


generator.current
1
Main routine
2
Generator
yield


generator.send
1
3
Main routine
2
4
Generator
yield


generator.current
generator.send
1
3
5
2
4
6
return


generator.getReturn
1
3
5
7
2
4
6

Example #2

More generators

Generator #1

Main routine

Generator #2

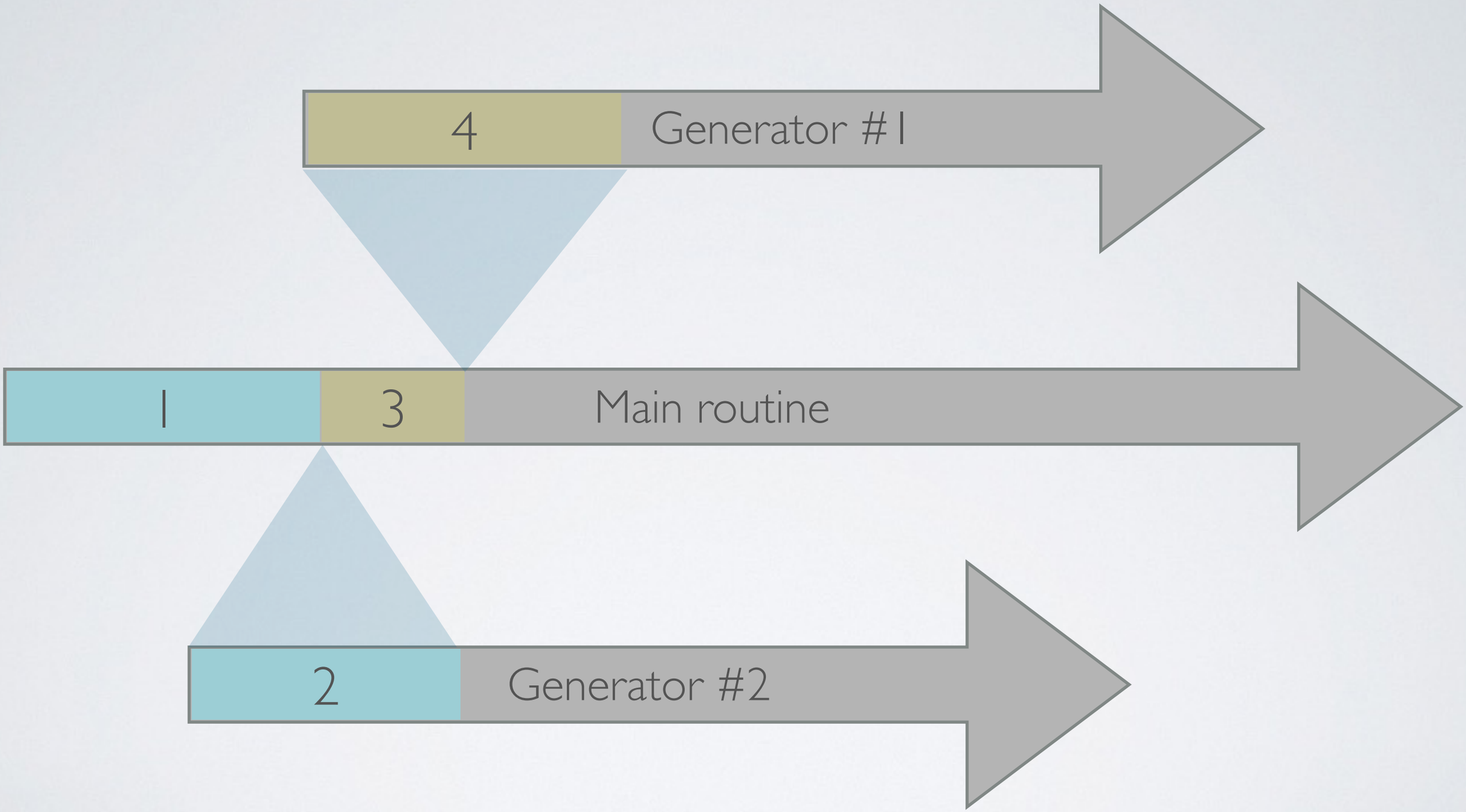
Generator #1

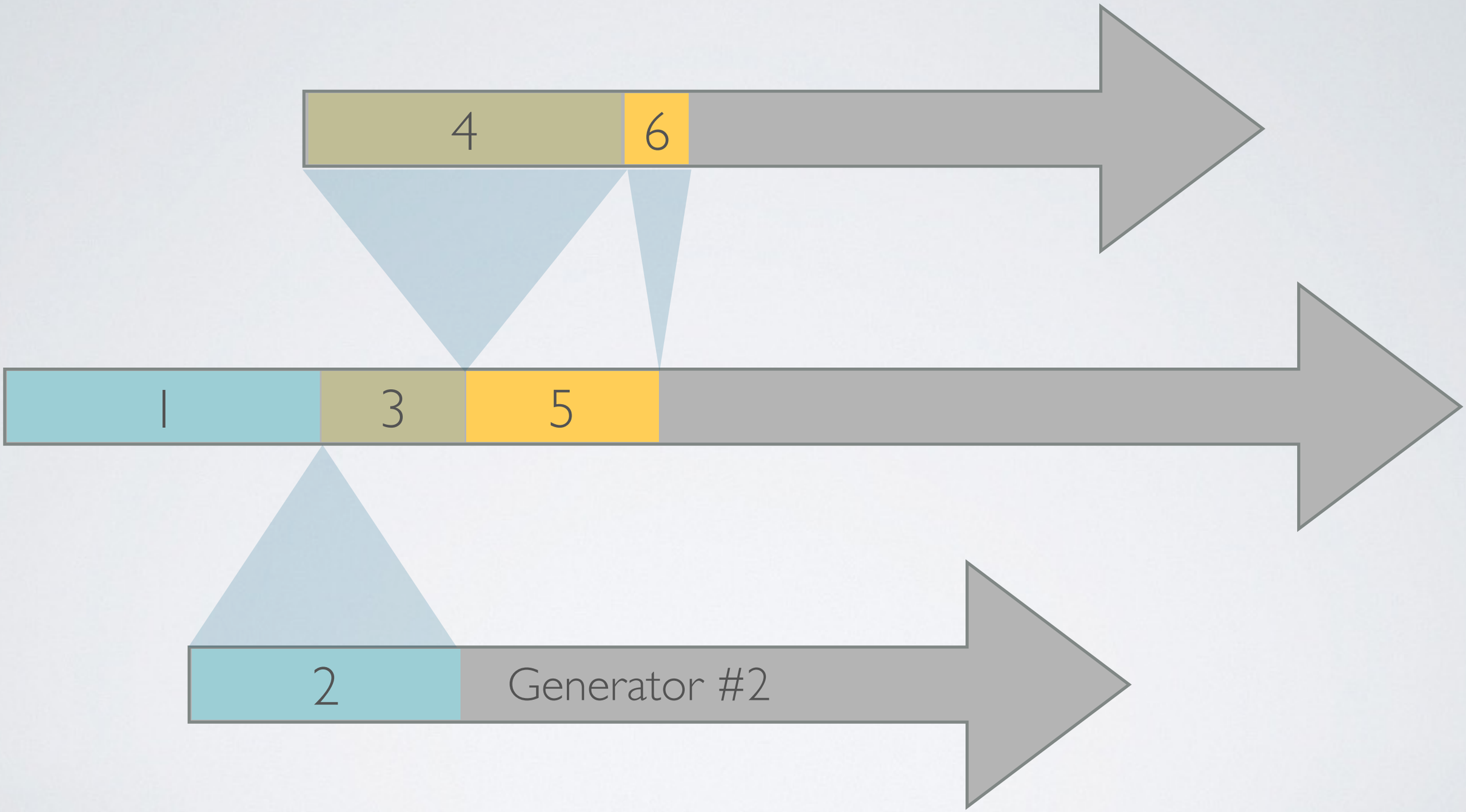
1

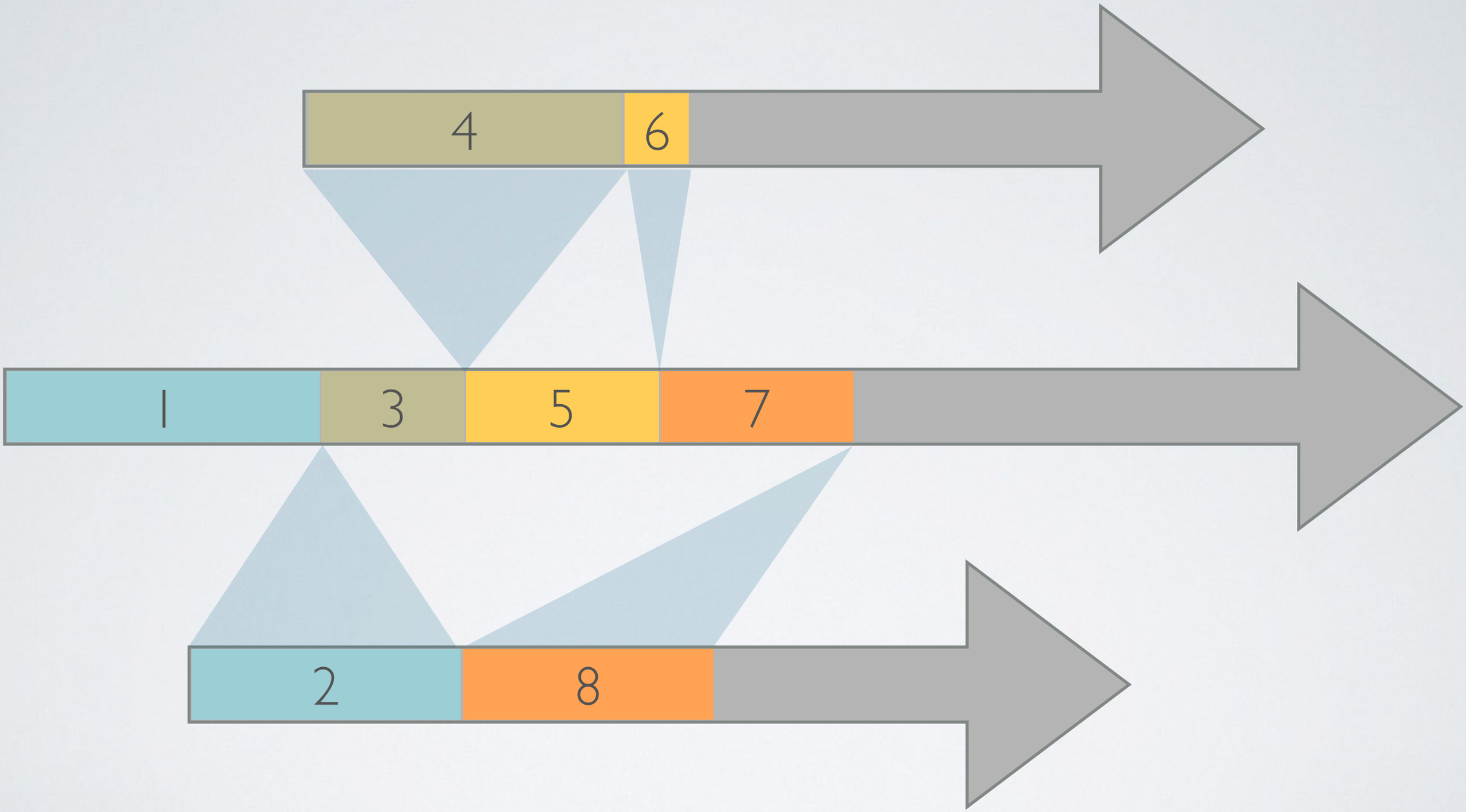
Main routine

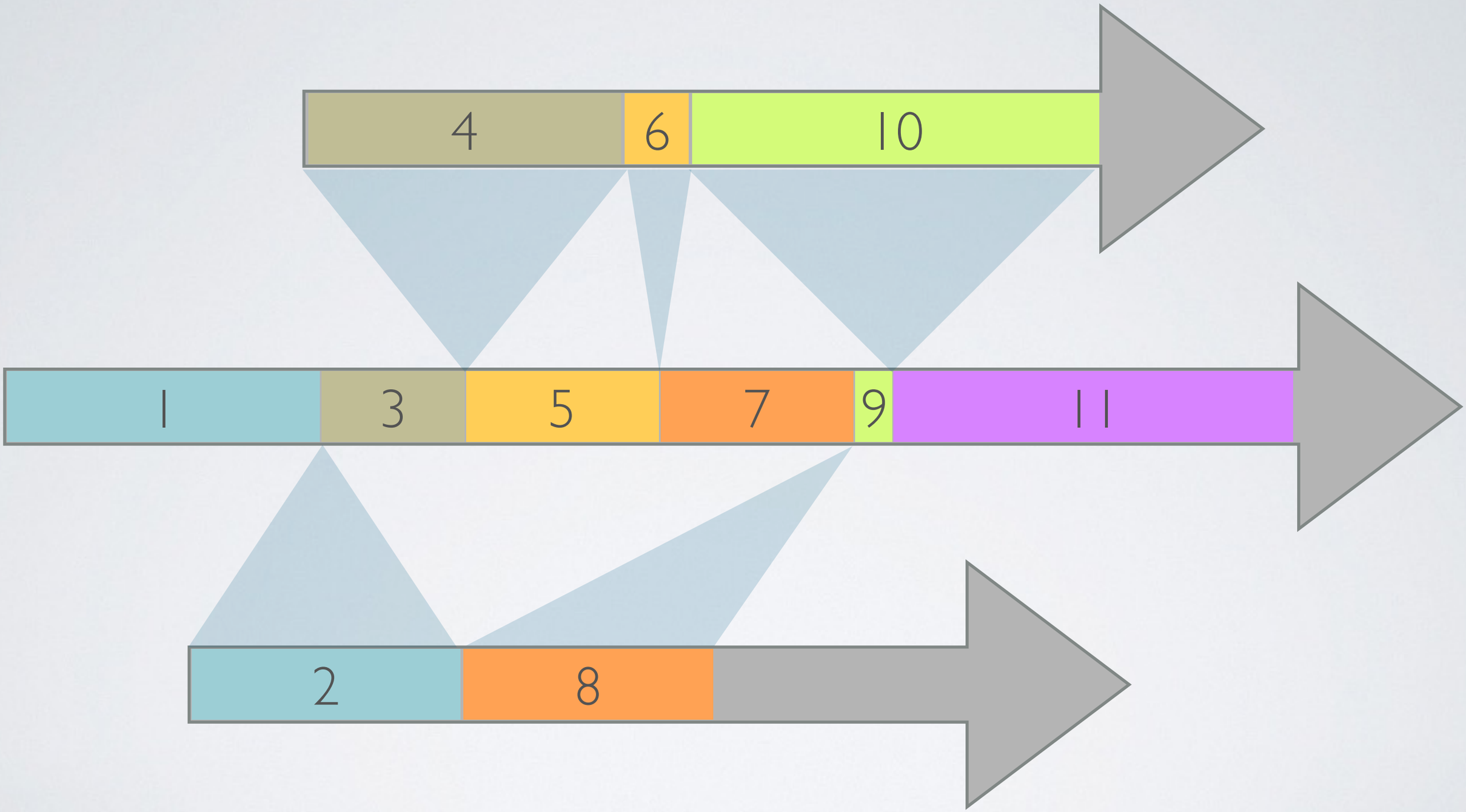
2

Generator #2


4
Generator #1
1
3
Main routine
2
Generator #2


4
6
1
3
5
2
Generator #2


4
6
1
3
5
7
2
8


4
6
10
1
3
5
7
9
11
2
8

Generators

the weird part…

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

// Here the only way to create a
// generator
function create(): \Generator {
    yield;
}

$generator = create();
```

```php
// Only the 'yield' keyword matters
function emptyGenerator(): \Generator {
    return;
    yield;
}

$generator = emptyGenerator();
```

```php
function dyingGenerator(): \Generator {
    die('hard');
    yield;
}

// The 'die' is not reached,
// the function is not executed
$generator = dyingGenerator();

// Now it's time to die hard
$generator->valid();
```

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

```php
// Type hinting ✅
function before(): int {
    return 10;
}

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

Event Loop

With generators?

```php
// https://github.com/amphp/artax/blob/master/examples/1-get-request.php

Loop::run(function () use ($argv) {
    try {
        $client = new Amp\Artax\DefaultClient;
        $promise = $client->request(
            $argv[1] ?? 'https://httpbin.org/user-agent'
        );

        $response = yield $promise;
        print $response->getStatus() . "\n";

        $body = yield $response->getBody();
        print $body . "\n";

    } catch (Amp\Artax\HttpException $error) {
        echo $error;
    }
});
```

Event Loop

from backstage

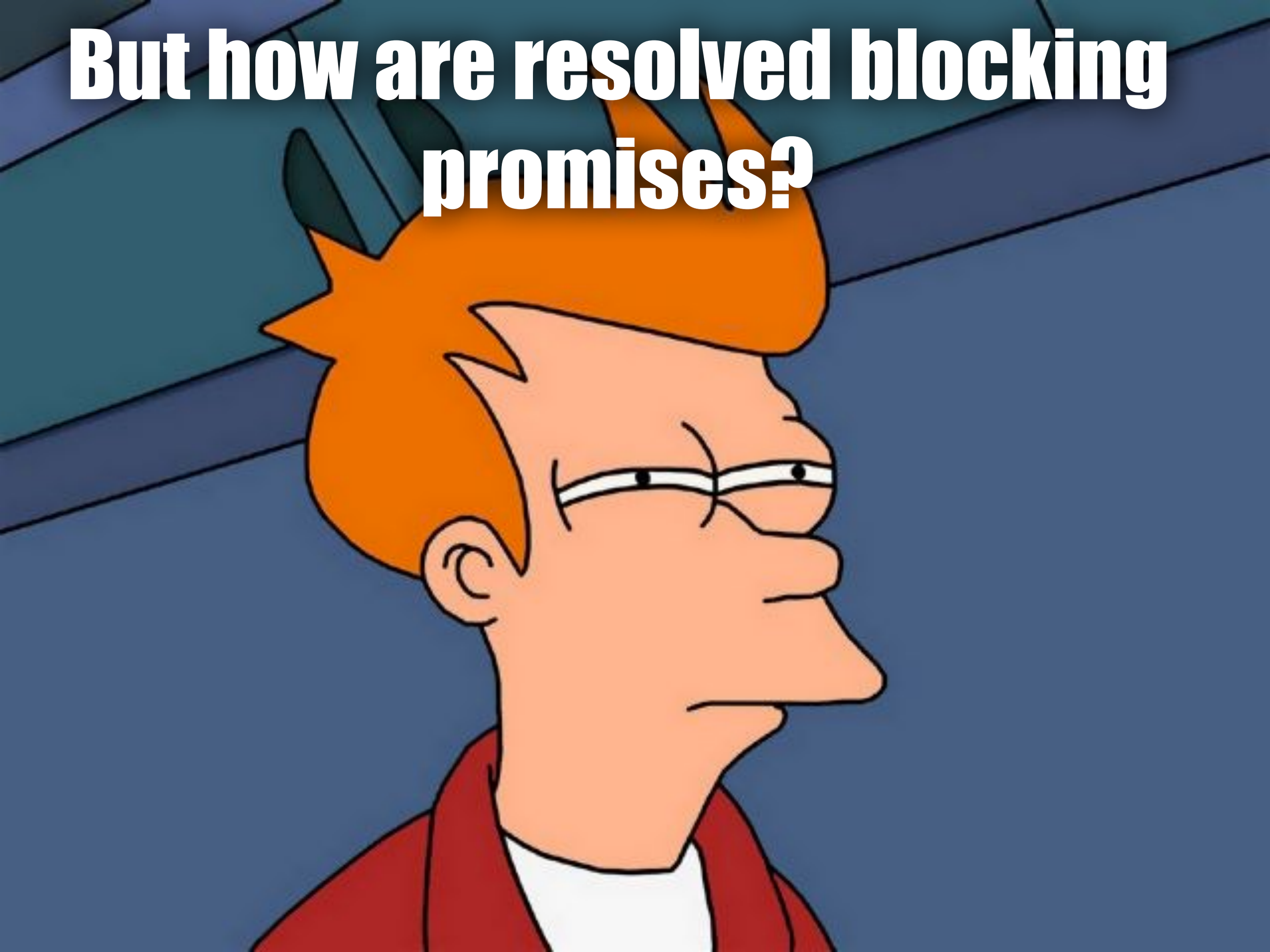
```php
while ($notFinished) {
    // This is called a tick
    foreach ($this->tasks as [$g, $gPromise]) {

        // Is generator already resolved?
        // (wait for it…)


        // Get Promise to resolve
        // (wait for it…)

        // Is promise finished?
        // (wait for it…)

    }
}
```

```php
// Is generator already resolved?
if (!$g->valid()) {
    $gPromise->resolve(
        $g->getReturn()
    );
    $this->remove($g);
    continue;
}
```

```php
// Get Promise to resolve
$p = $g->current();
if (!$p instanceof PromiseInterface) {
    throw new \Exception();
}
```

```php
// Is promise finished?
switch ($p->state()) {
    case PromiseState::SUCCESS:
        $g->send($p->getValue());
        break;
    case PromiseState::FAILURE:
        $g->throw($p->getException());
        break;
}
```

But how are resolved blocking promises?



```php
// #1 Old fashion listener
function tickListener()
{
    $this->tryToFinishTheTask();

    if ($this->isFinished) {
        $this->promise->resolve(
            $this->result
        );
    }
}
```

```php
// #2 Top Hype generator 🤘
function subEventLoop(): \Generator
{
    while (!$this->isFinished) {
        yield $this->eventLoop->waitNextTick();
        $this->tryToFinishTheTask();
    }

    return $this->result;
}
```

Asynchronous Events in Php

`stream_select`

`curl_multi_exec`

Threads

System call

Interruptions

extensions

...


Photos by Ralph Dettmann
SOON
IN YOUR PROJECT !!
Bodybuilding-pics.com

Use cases

Fiber RFC ?

https://wiki.php.net/rfc/fiber

Thanks!

@b_viguier