Async… and Generators …chronous

Benoit Viguier

LFT M6Web
27/04/2018

Php & Js Friendly

❤️

Asynchronous programming

≠ Parallel programming

- **Asynchrony**, in computer programming, refers to the occurrence of events independent of the main program flow and ways to deal with such events.

- **Parallel** computing is a type of computation in which many calculations or the execution of processes are carried out concurrently.

- **Concurrent** computing is a form of computing in which several computations are executed during overlapping time periods—concurrently—instead of sequentially (one completing before the next starts).

en.wikipedia.org

Parallel

Asynchronous


ASYNCHRONOUS PROGRAMMING
LIGHTYEAR
EVERYWHERE !

How to take/release « ownership » ?

No blocking wait

EventLoop

The master process

Events listener

Mainly for external events

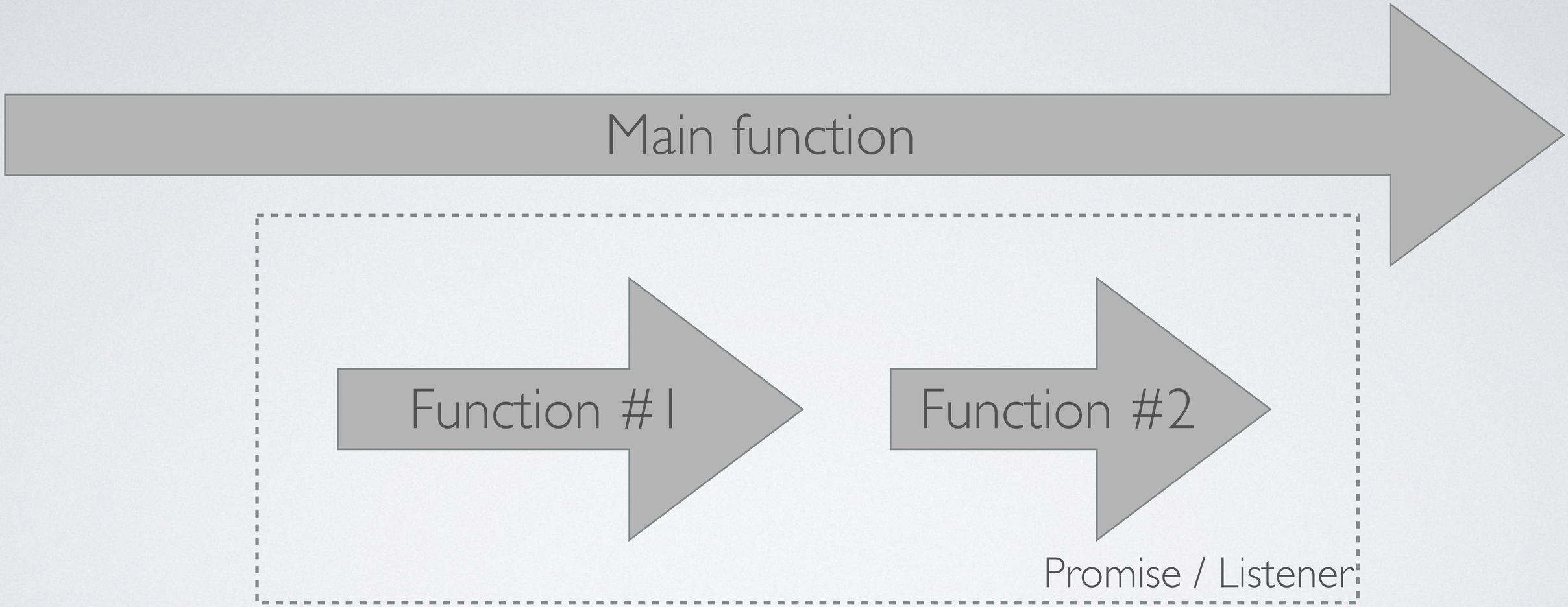
```
el.addEventListener(
  "click",
  function() {},
  false
);
```

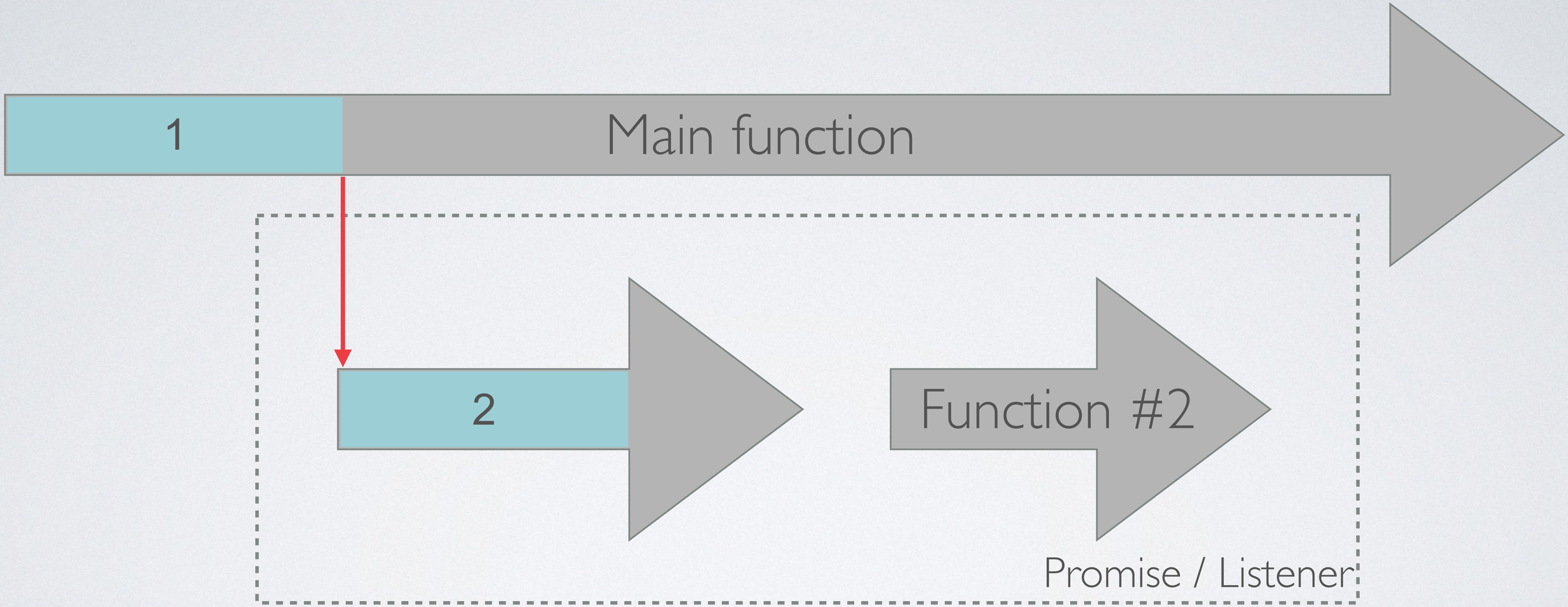
Promises

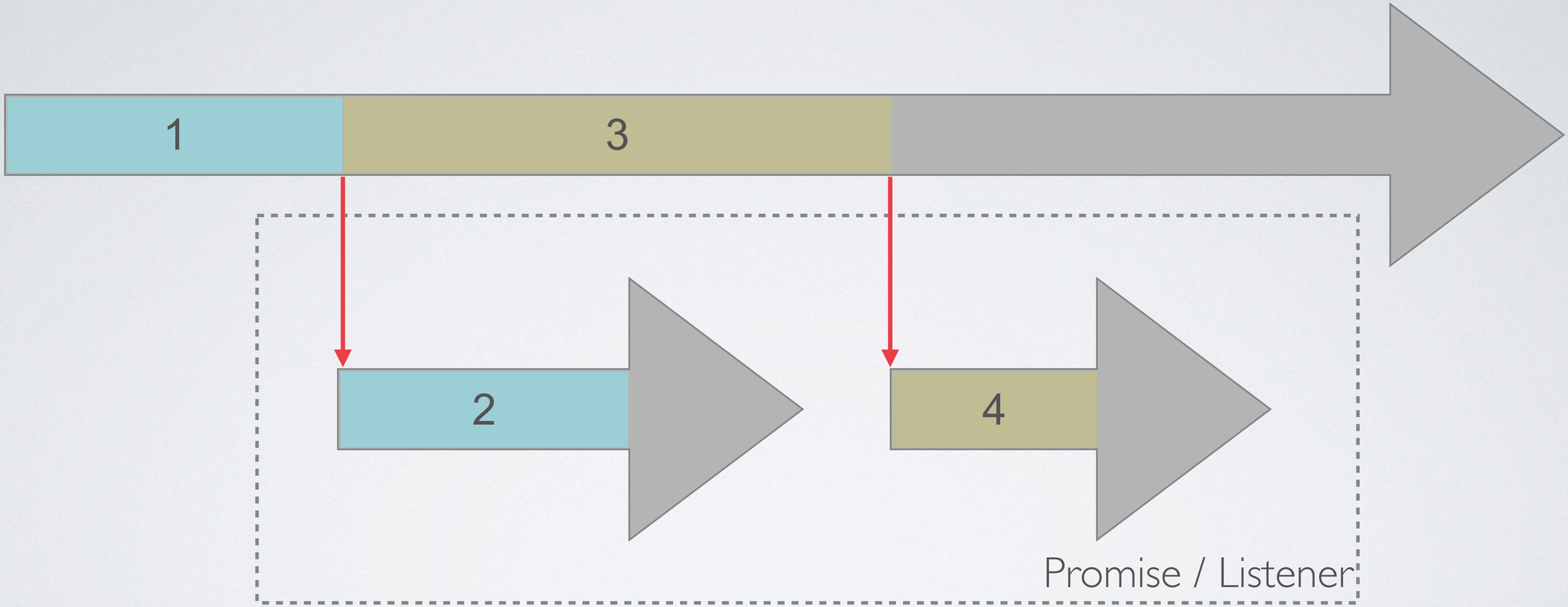
Promises/A+

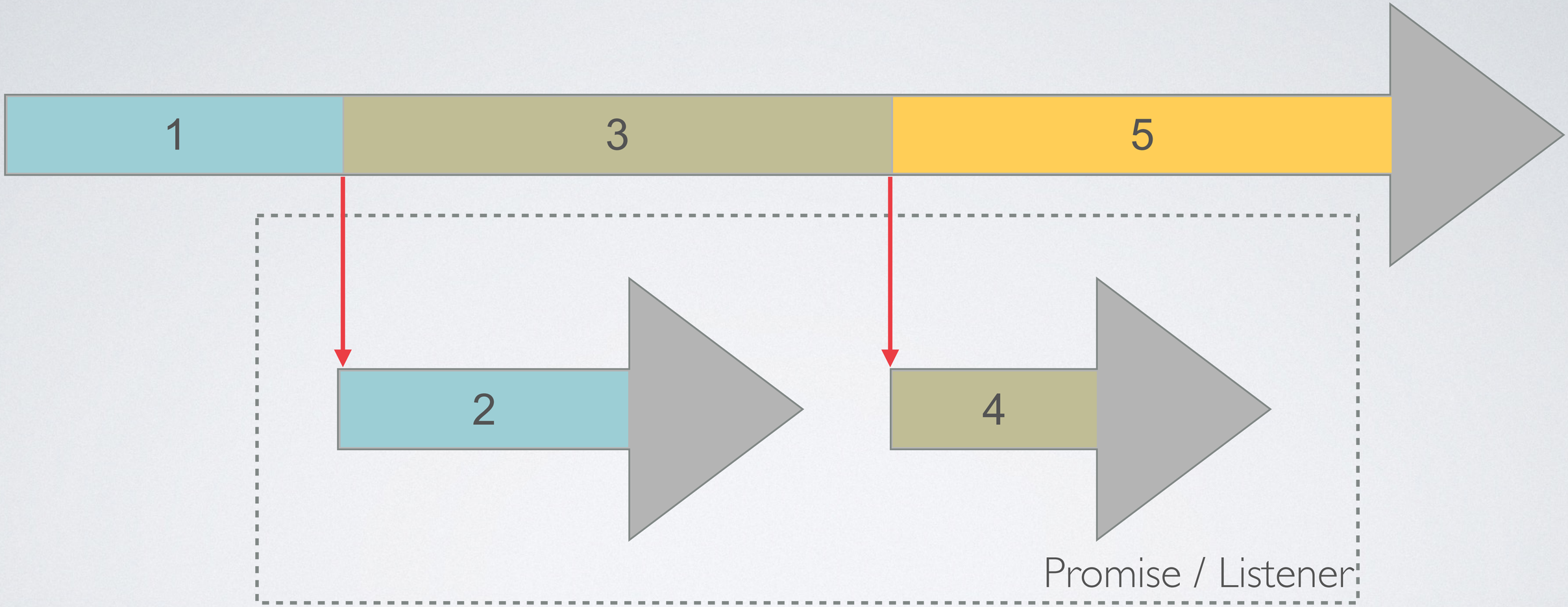
```
const promise = doSomething();
const promise2 = promise.then(
  successCallback,
  failureCallback
);
```

```
doSomething().then(function(result) {
  return doSomethingElse(result);
})
.then(function(newResult) {
  return doThirdThing(newResult);
})
.then(function(finalResult) {
  console.log(‘Result: '+finalResult);
});
```

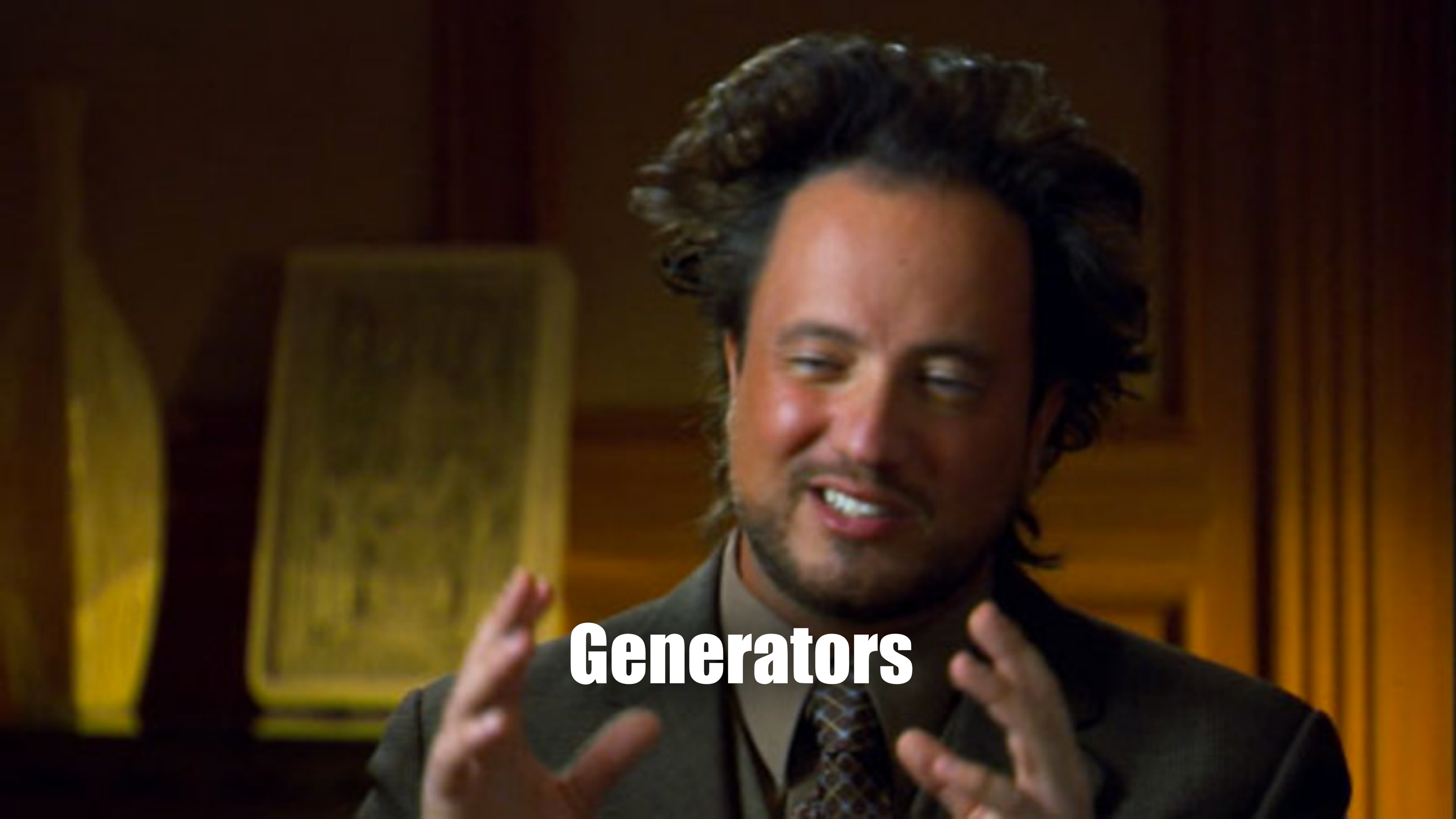

Main function
Function #1
Function #2
Promise / Listener


1
Main function
2
Function #2
Promise / Listener


1
3
2
4
Promise / Listener


1
3
5
2
4
Promise / Listener

Not so bad…


Generators

Generators

and Coroutines

```
function* idMaker() {
  var index = 0;
  while(true)
    yield index++;
}

var gen = idMaker();
console.log(gen.next().value); // 0
console.log(gen.next().value); // 1
console.log(gen.next().value); // 2
// ...
```

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

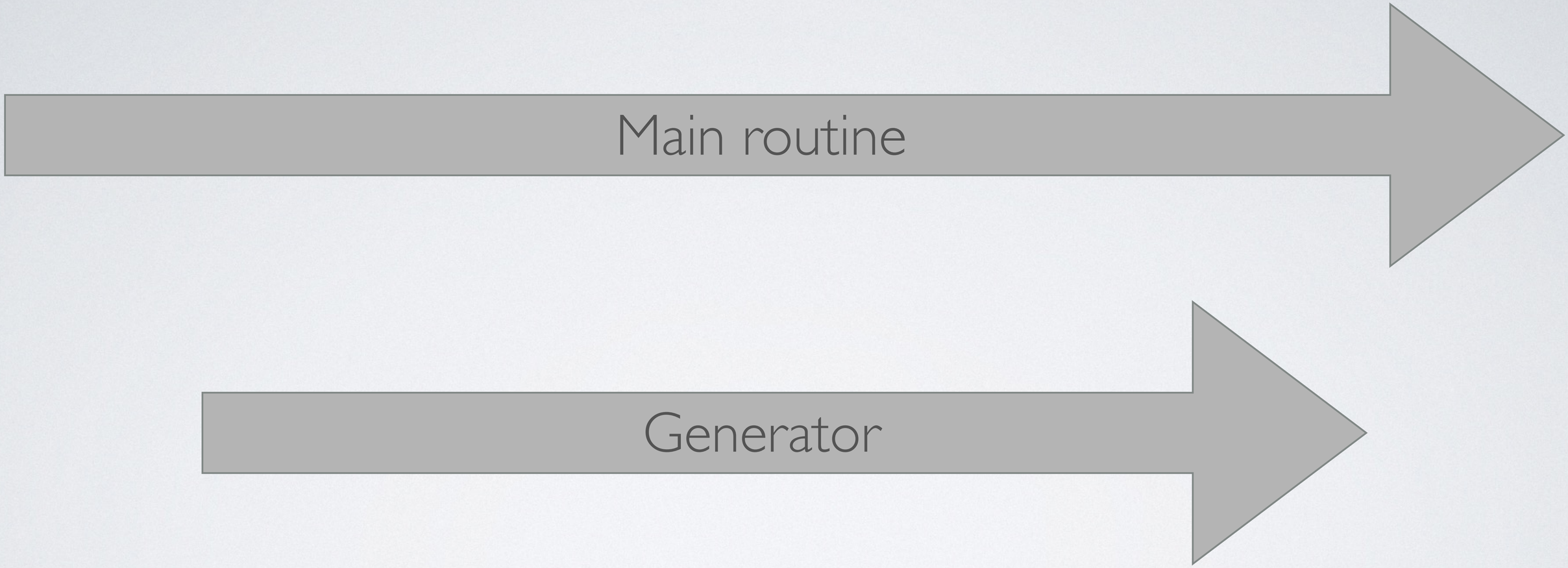
« Generators in JavaScript -- especially when combined with Promises -- are a very powerful tool for asynchronous programming as they mitigate -- if not entirely eliminate -- the problems with callbacks, such as Callback Hell and Inversion of Control.

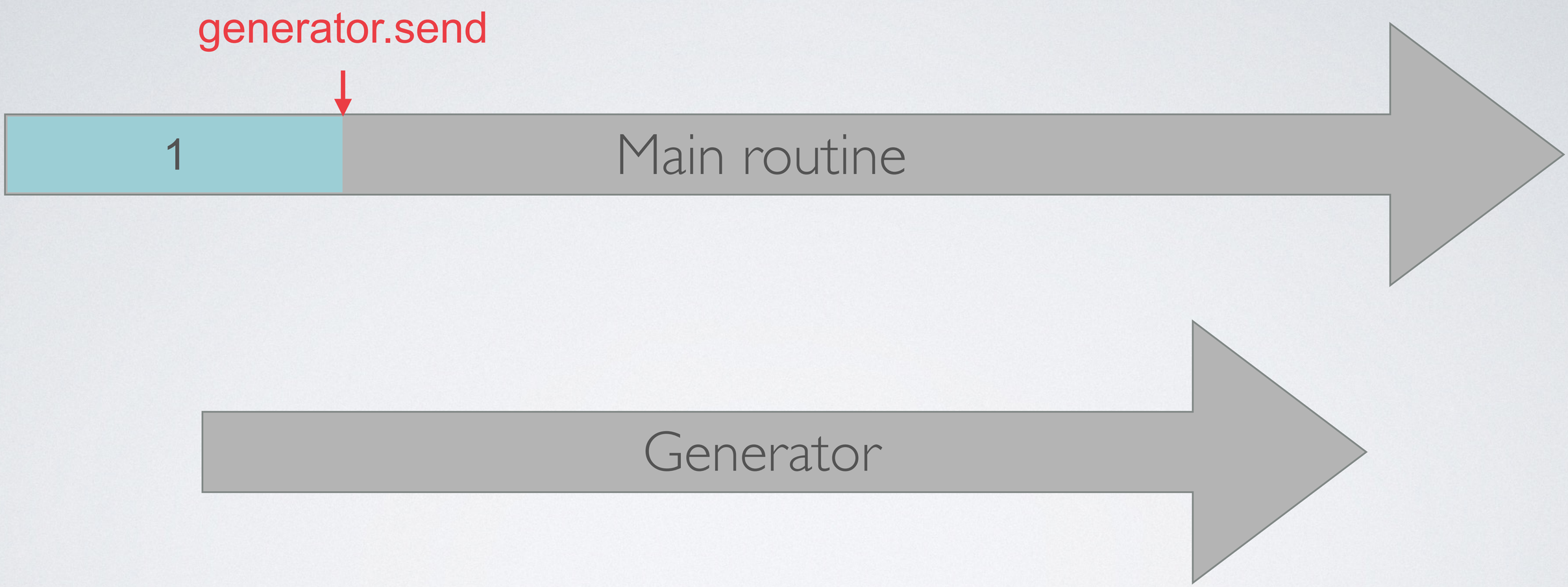
This pattern is what async functions are built on top of. *»*

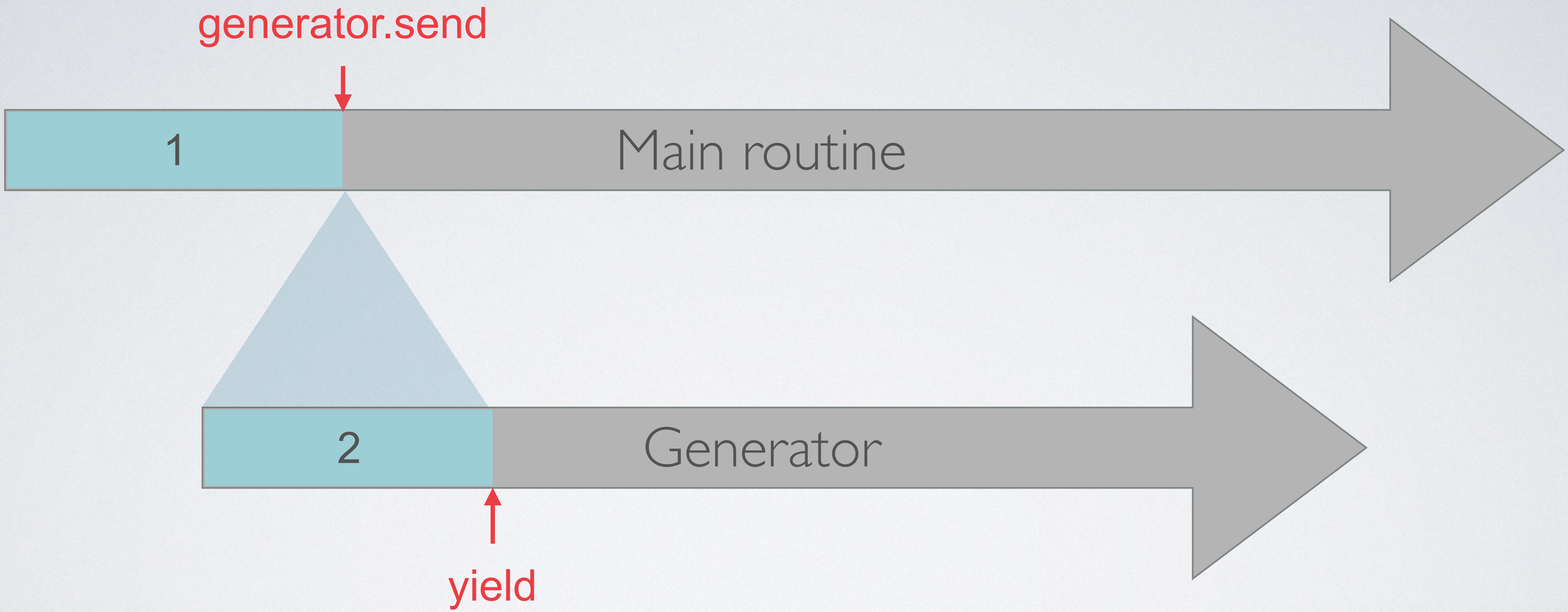
-MDN Web docs

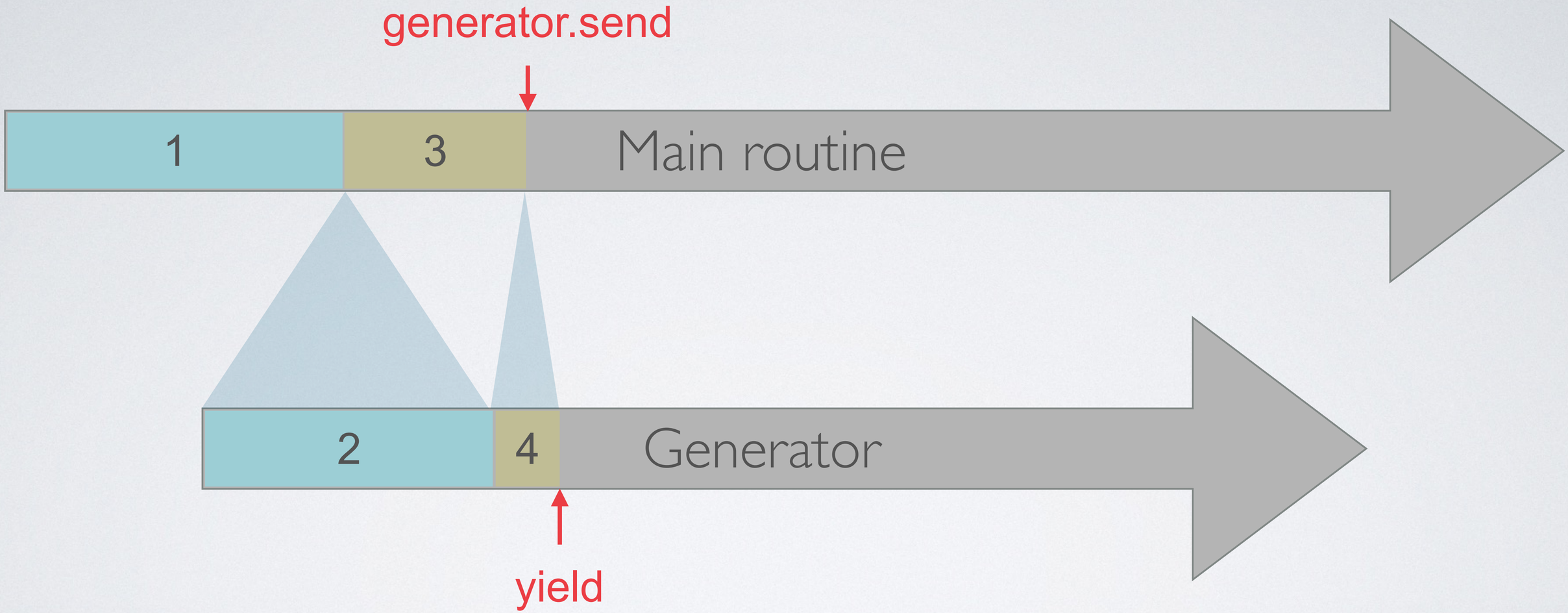
Example #1

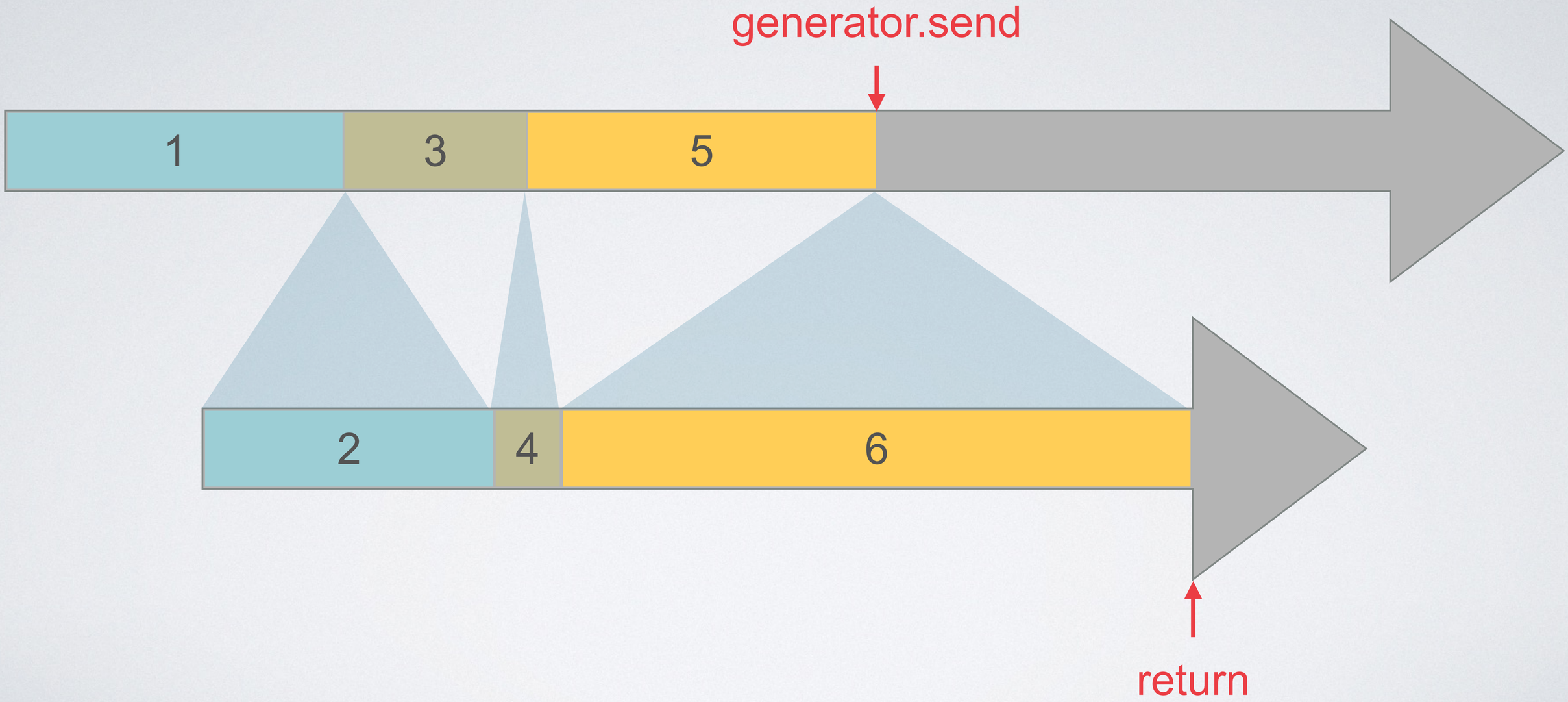
Simple workflow

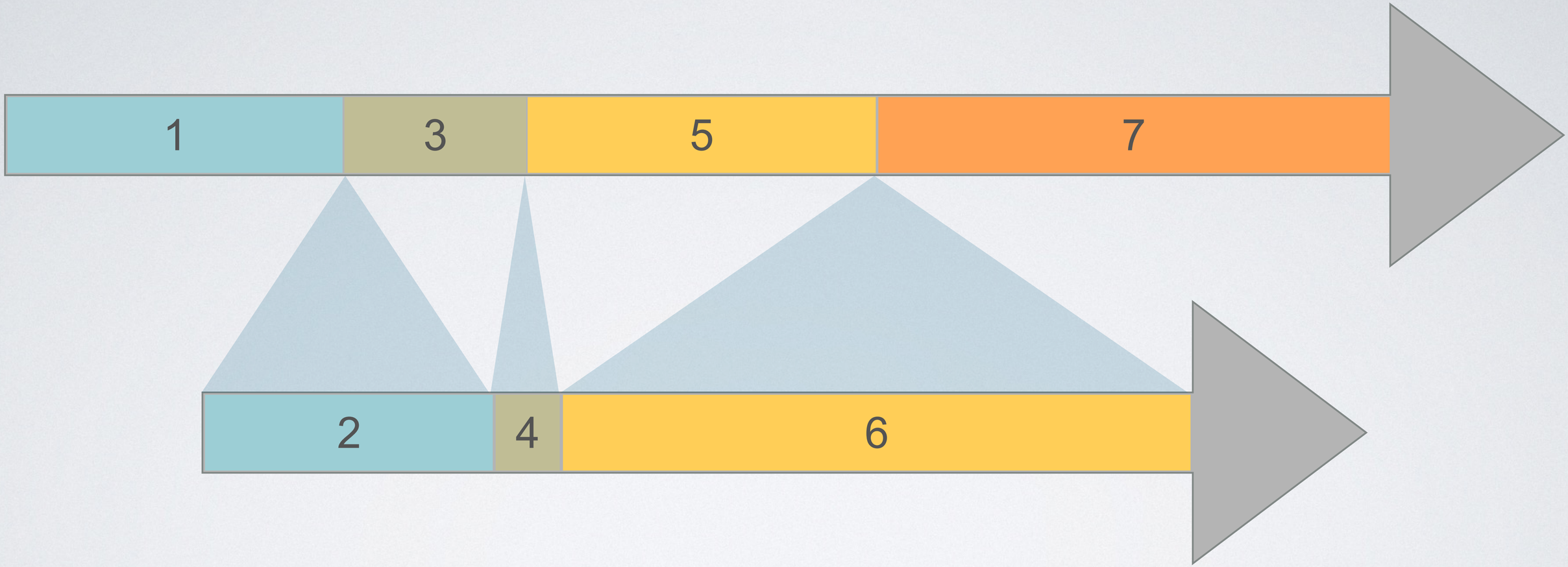

Main routine
Generator


generator.send
1
Main routine
Generator


generator.send
1
Main routine
2
Generator
yield

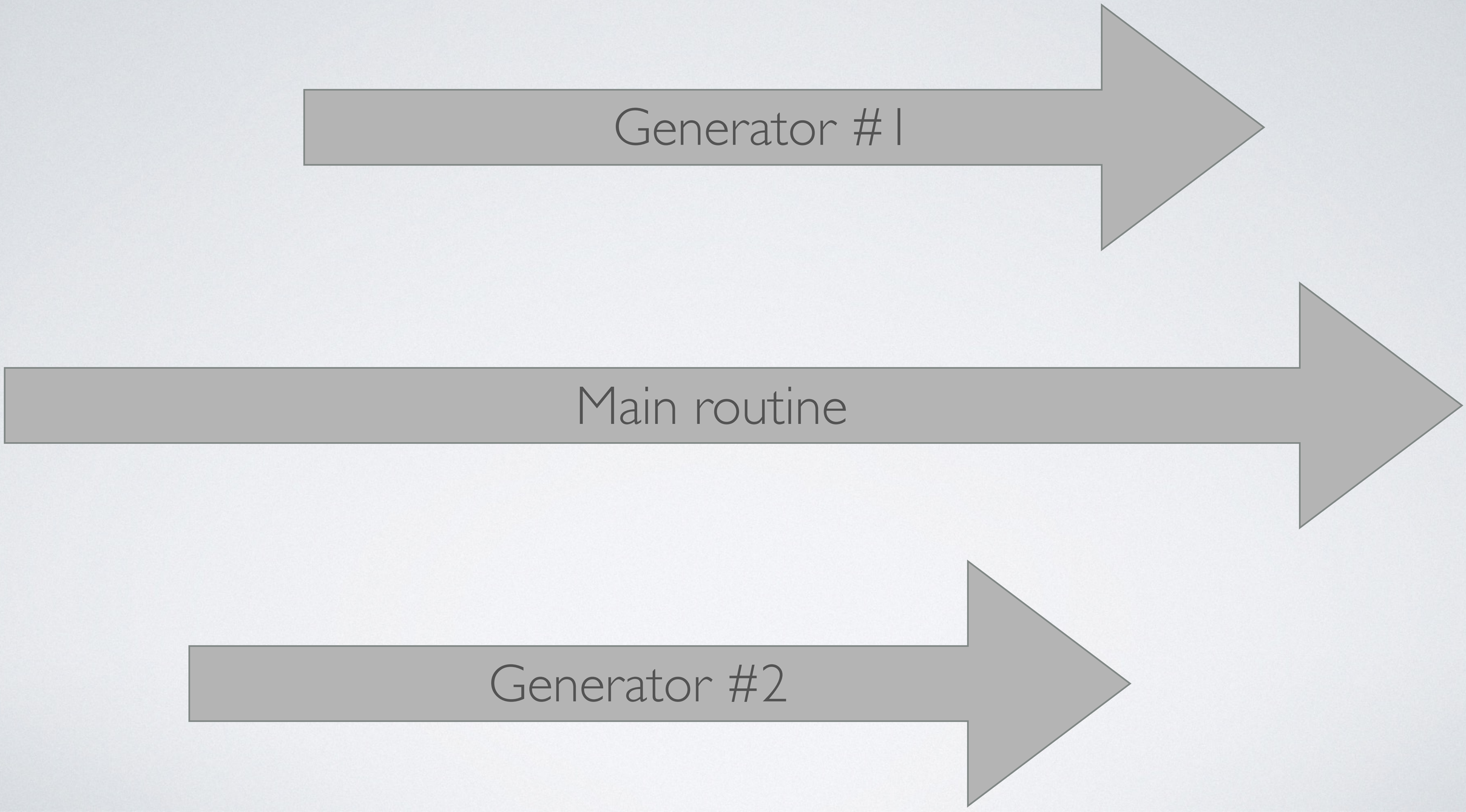

generator.send
1
3
Main routine
2
4
Generator
yield

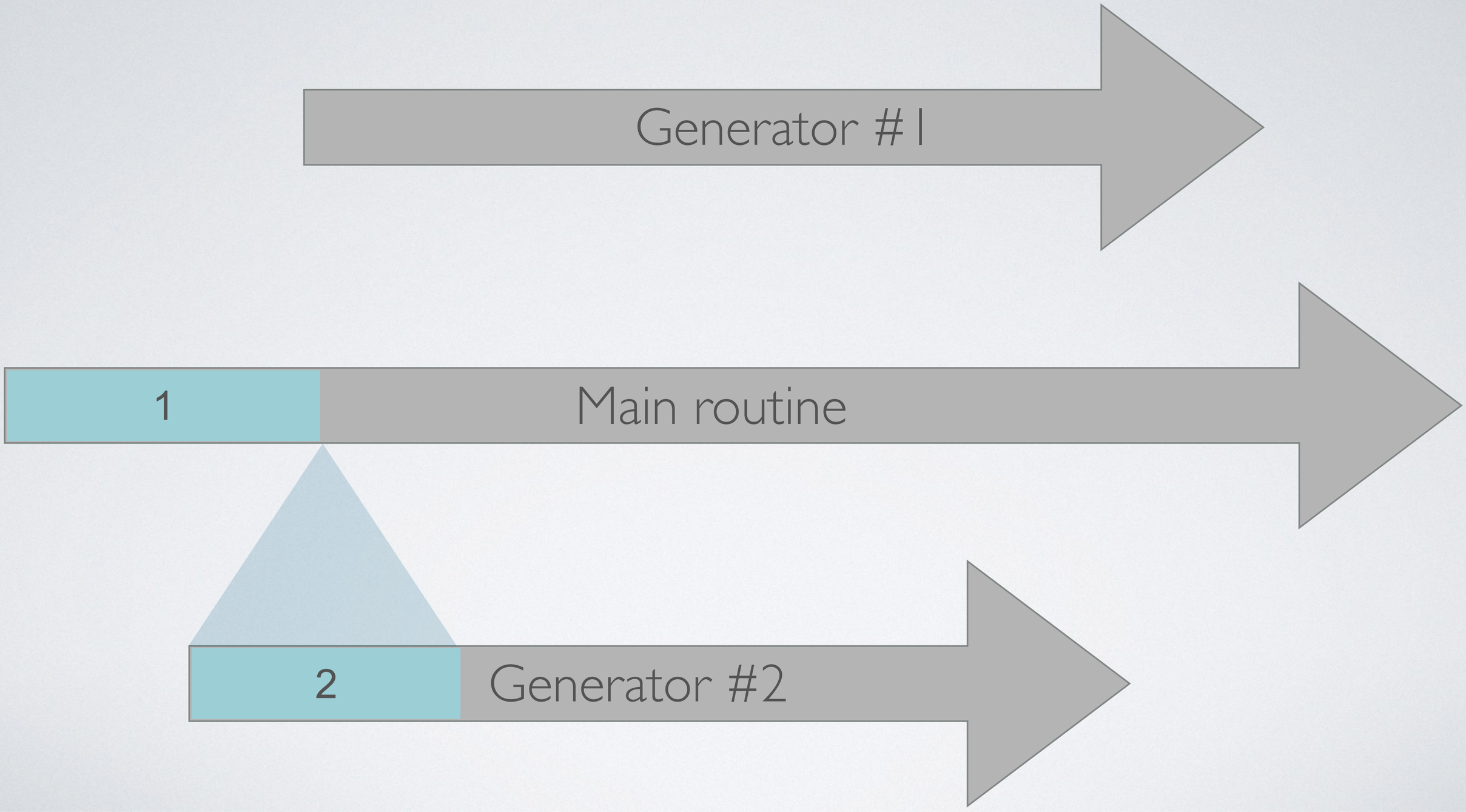

generator.send
1
3
5
2
4
6
return

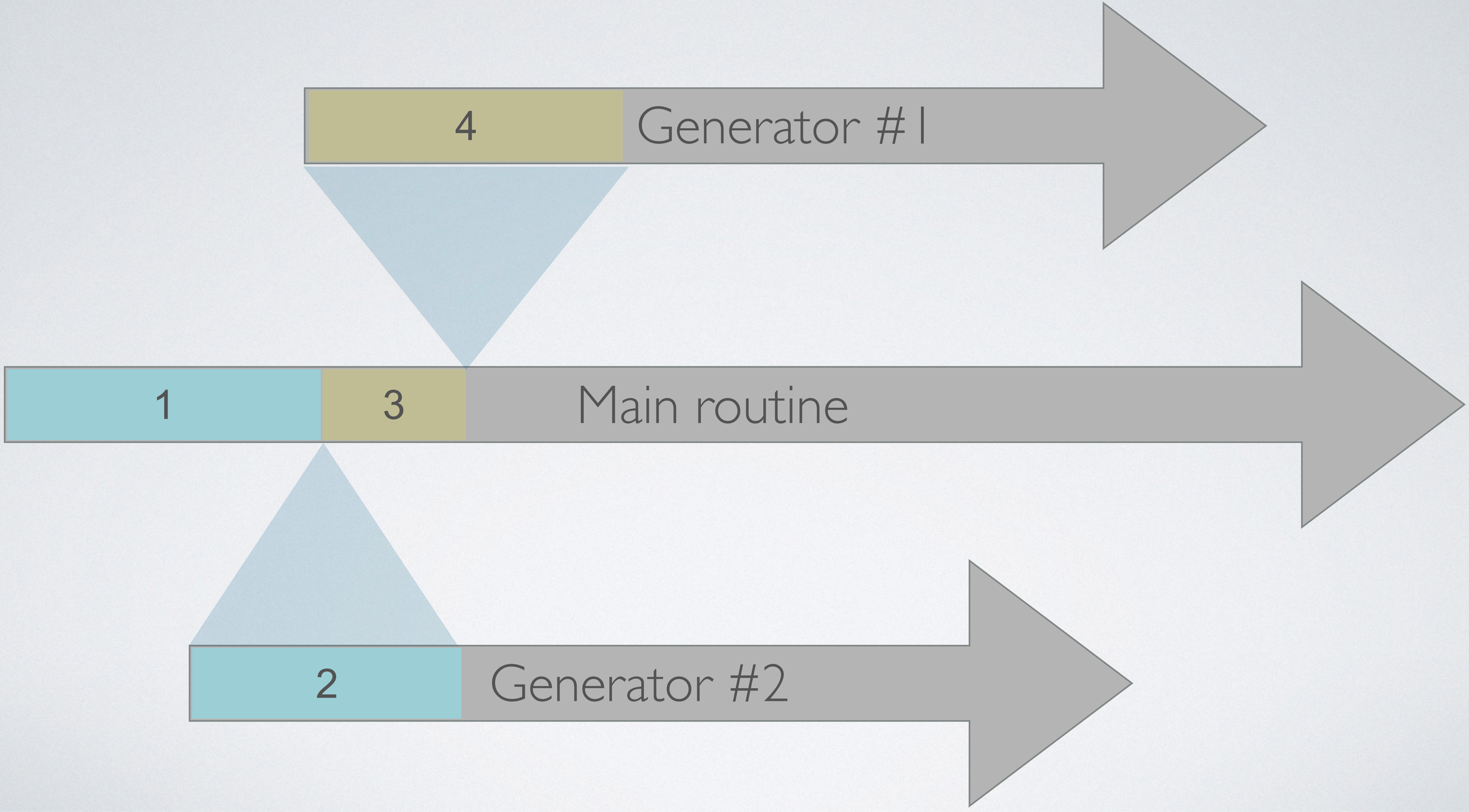

1
3
5
7
2
4
6

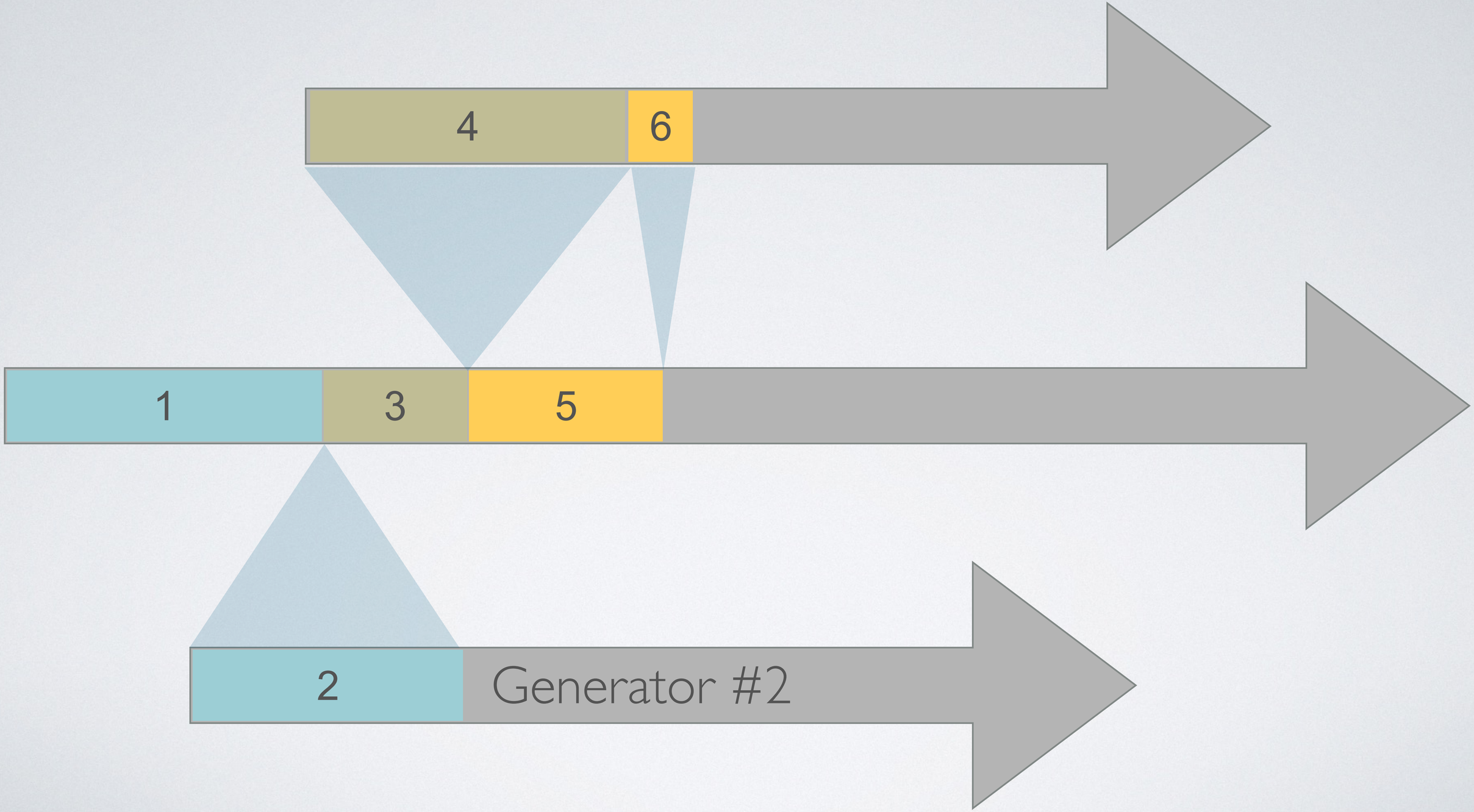
Example #2

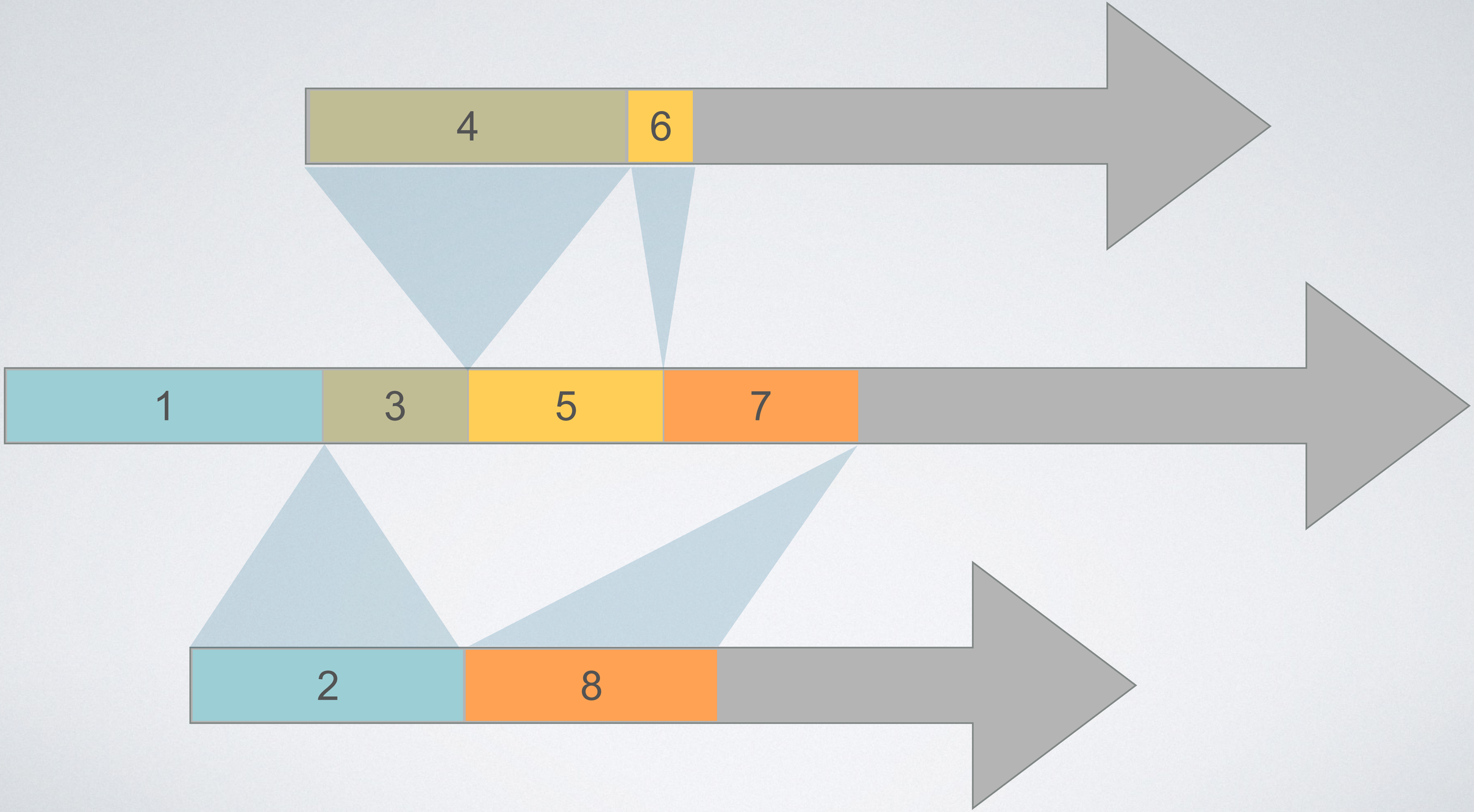
More generators

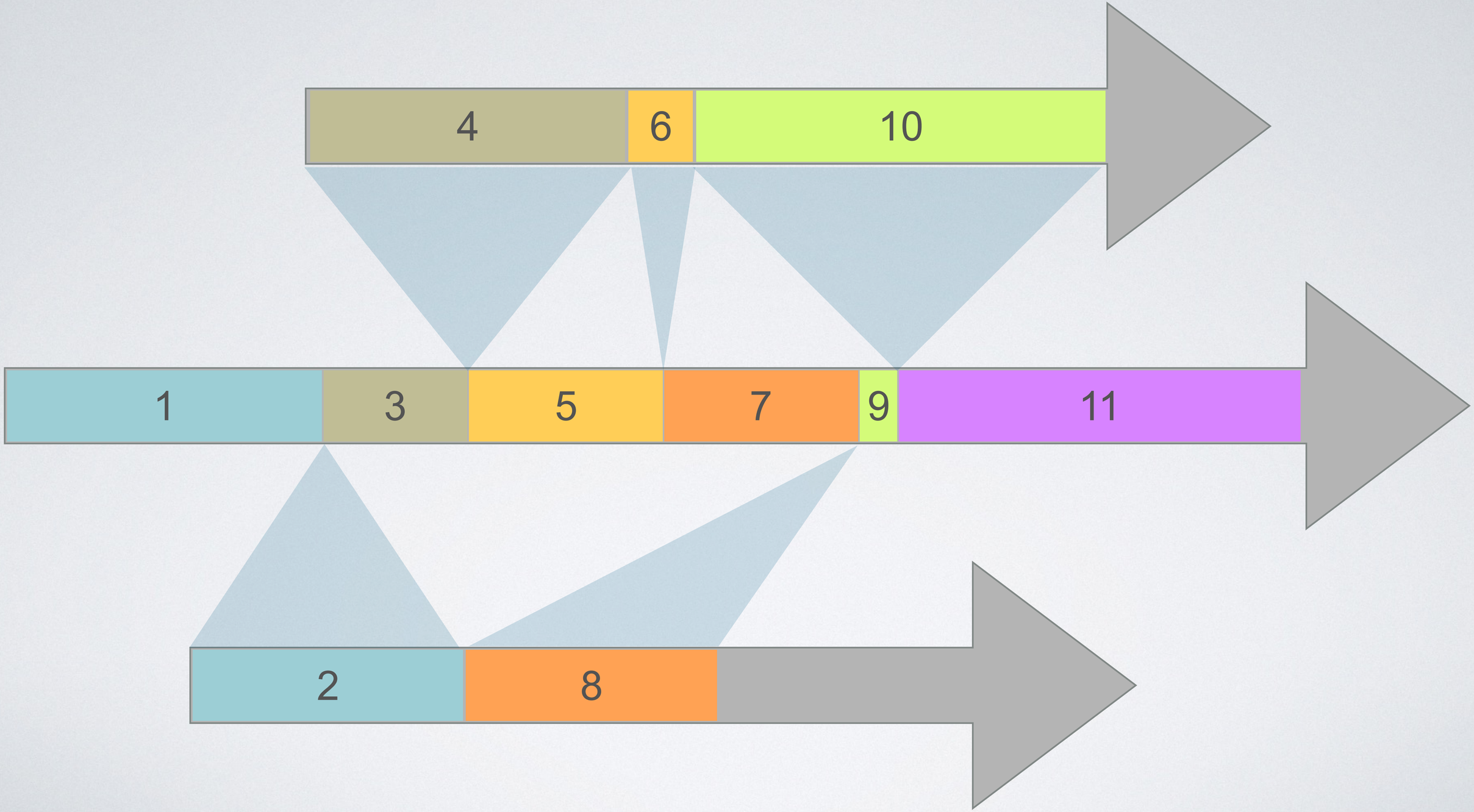

Generator #1
Main routine
Generator #2


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

Available operations

Generator implements Iterator

```
public mixed current ( void )
public mixed key ( void )
```

```
public mixed getReturn ( void )
public bool valid ( void )
```

```
public void next ( void )
public mixed send ( mixed $value )
public mixed throw ( Throwable $e )
```

```php
function myGen(int $a): \Generator {

    $result = yield $key => $value;

    return $data;
}
```

```
function foo(): \Generator {

  yield $timer->wait(100/*ms*/);

  $r = yield $client->get("http://…");

  return $data;
}
```

Event Loop

The tricky part

```
public function async(\Generator $gen)
: PromiseInterface {

    $genPromise = new Promise()
    $this->tasks[] = [$gen,$genPromise];

    return $genPromise;
}
```

```
while ($notFinished) {
    // This is called a tick
    foreach(
        $this->tasks as list($gen, $genPromise)
    ) {
        // Treat tasks…
    }
}
```

```php
$p = $gen->current();

if (!$p instanceof PromiseInterface) {
    throw new \Exception();
}
```

```
switch ($p->state()) {
    case PromiseState::SUCCESS:
        $gen->send($p->getValue());
        break;
    case PromiseState::FAILURE:
        $gen->throw($p->getException());
        break;
}
```

```php
if (!$gen->valid()) {
    $genPromise->resolve($g->getReturn());
    unset($gen);
}
```


But how are resolved blocking promises?

```
function foo(): \Generator {

  yield $timer->wait(100/*ms*/);

  $r = yield $client->get("http://…");

  return $data;
}
```

```
function tickListener() {
    // Polling
    if( $jobIsFinished ) {
        $promise->resolve($value);
    }
}
```

Asynchronous Events in Php

`stream_select`, (`stream_*`)

Threads

System call

Interruptions

extensions

...

Generators

the weird part…

```php
function myGen(): \Generator {
    die("hard");
    yield;
}

$g = myGen(); // Don’t die at all…
$g->current(); // Die hard
```

```php
function myGen(): \Generator {

    // No type hinting :(
    $var = yield "string";
    $var = yield new MyObject();

    // No more possible here :(((
    return 42;
}
```


Photos by Ralph DeHaan
SOON
IN YOUR PROJECT !!

Thanks!