Iterators and Generators

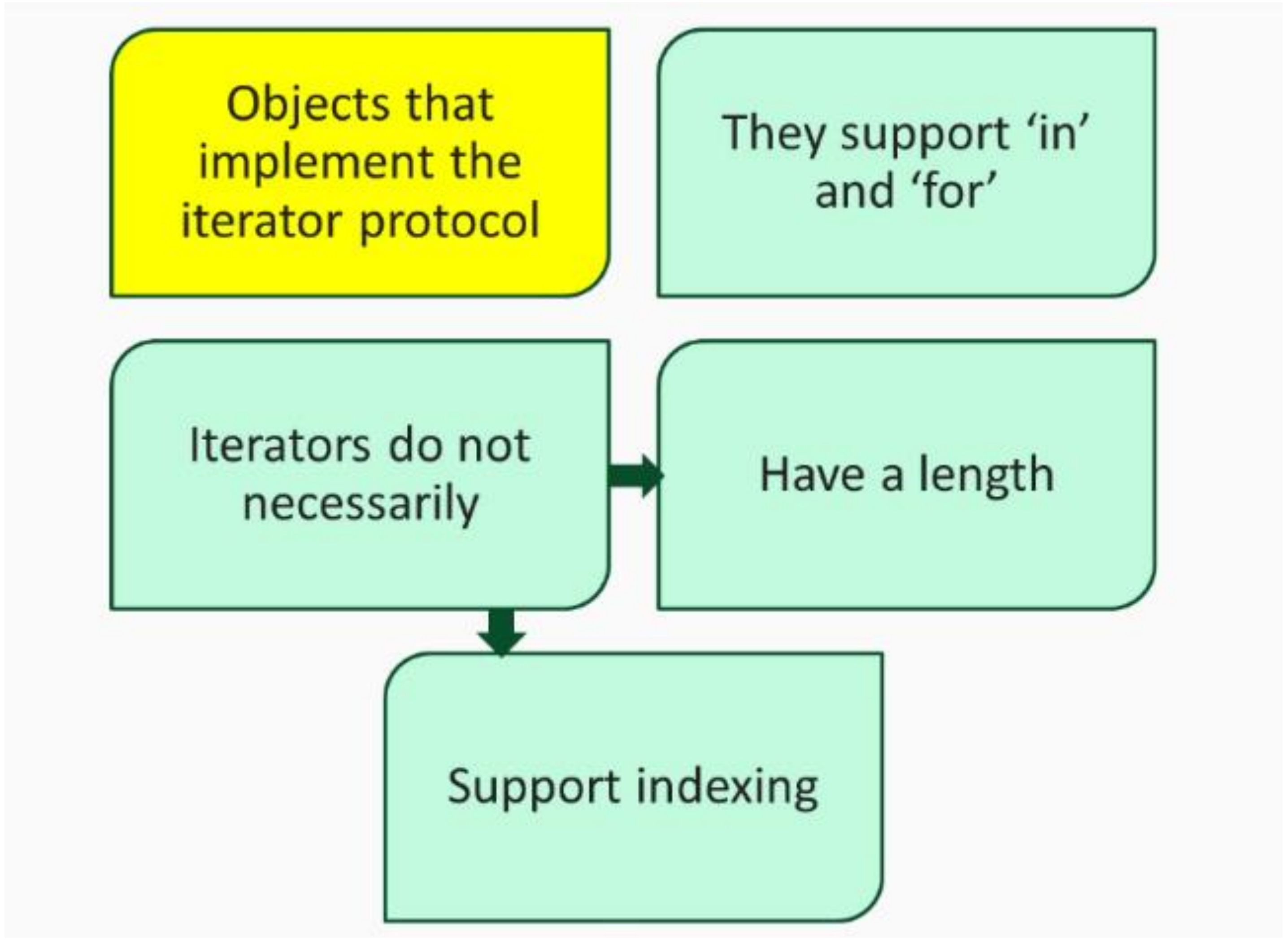
- Iterators
- Generators
- Coroutines
- Collections

What Is an Iterator?

- The iterator protocol
- How you can use an iterator in real code

Iterators Are Objects That You Can Use in a ‘for’ Loop

Iterator – Simplest Collection



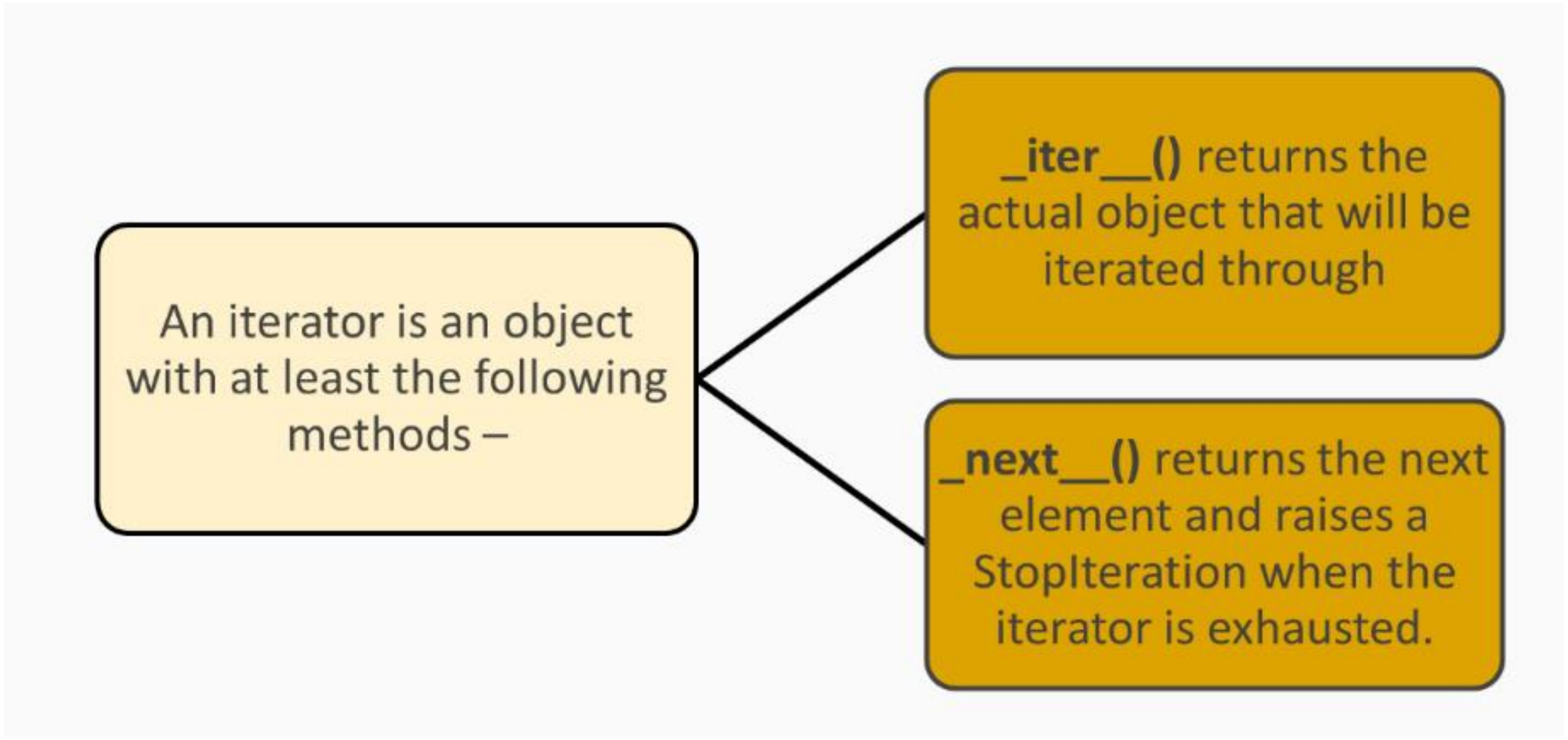
Objects that implement the iterator protocol
They support 'in' and 'for'
Iterators do not necessarily
Have a length
Support indexing

Lists, Dicts, Tuples Are Sequences That Extend the Iterator Protocol

Creating Your Own Iterator

- __iter__() method
- __next__() method
- How to implement a custom iterator in real code

Iterator – on the inside

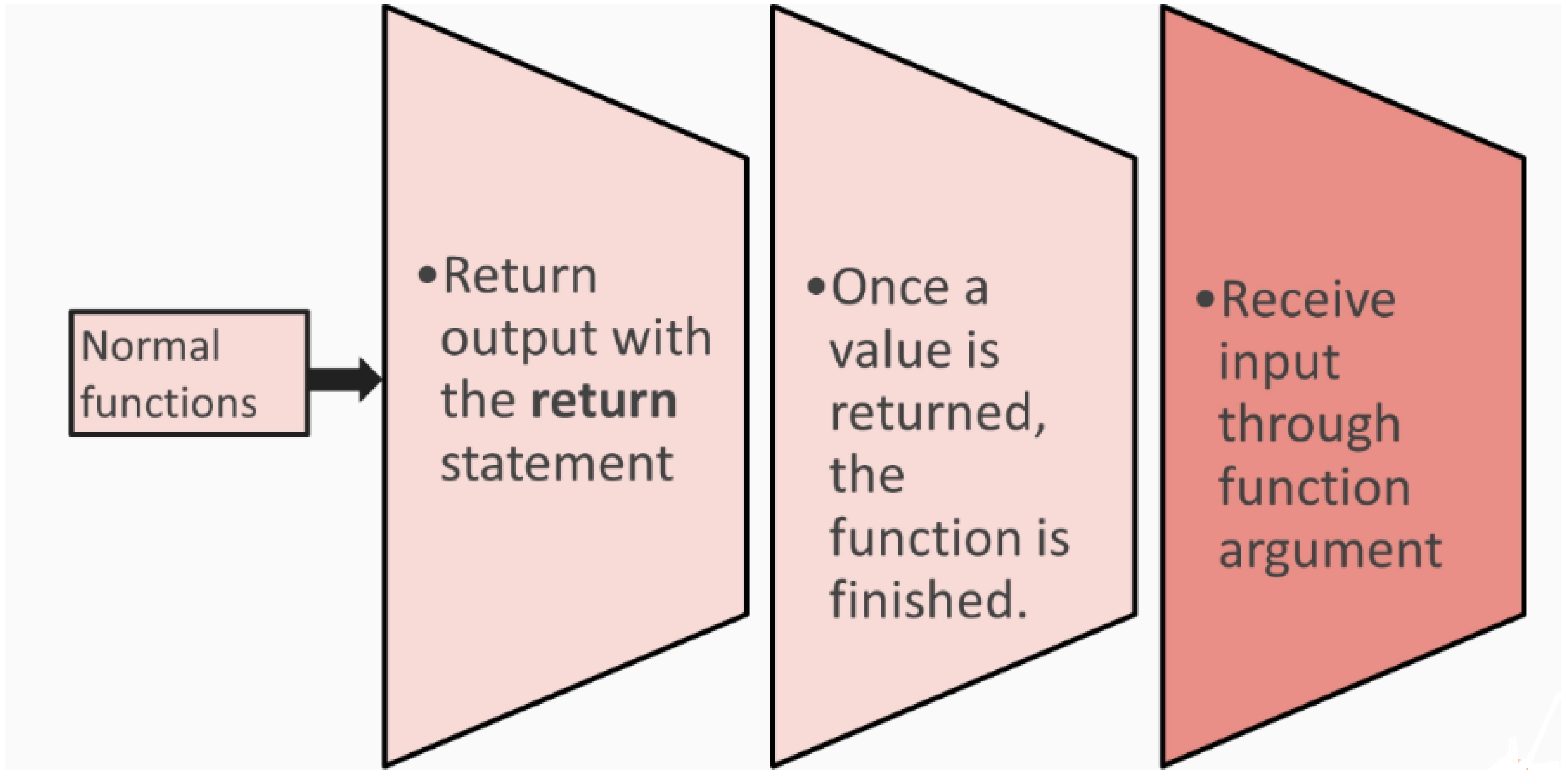

An iterator is an object with at least the following methods –
_iter__() returns the actual object that will be iterated through
_next__() returns the next element and raises a StopIteration when the iterator is exhausted.

Exploring Generators

- What a generator is
- How you can implement a generator in real code

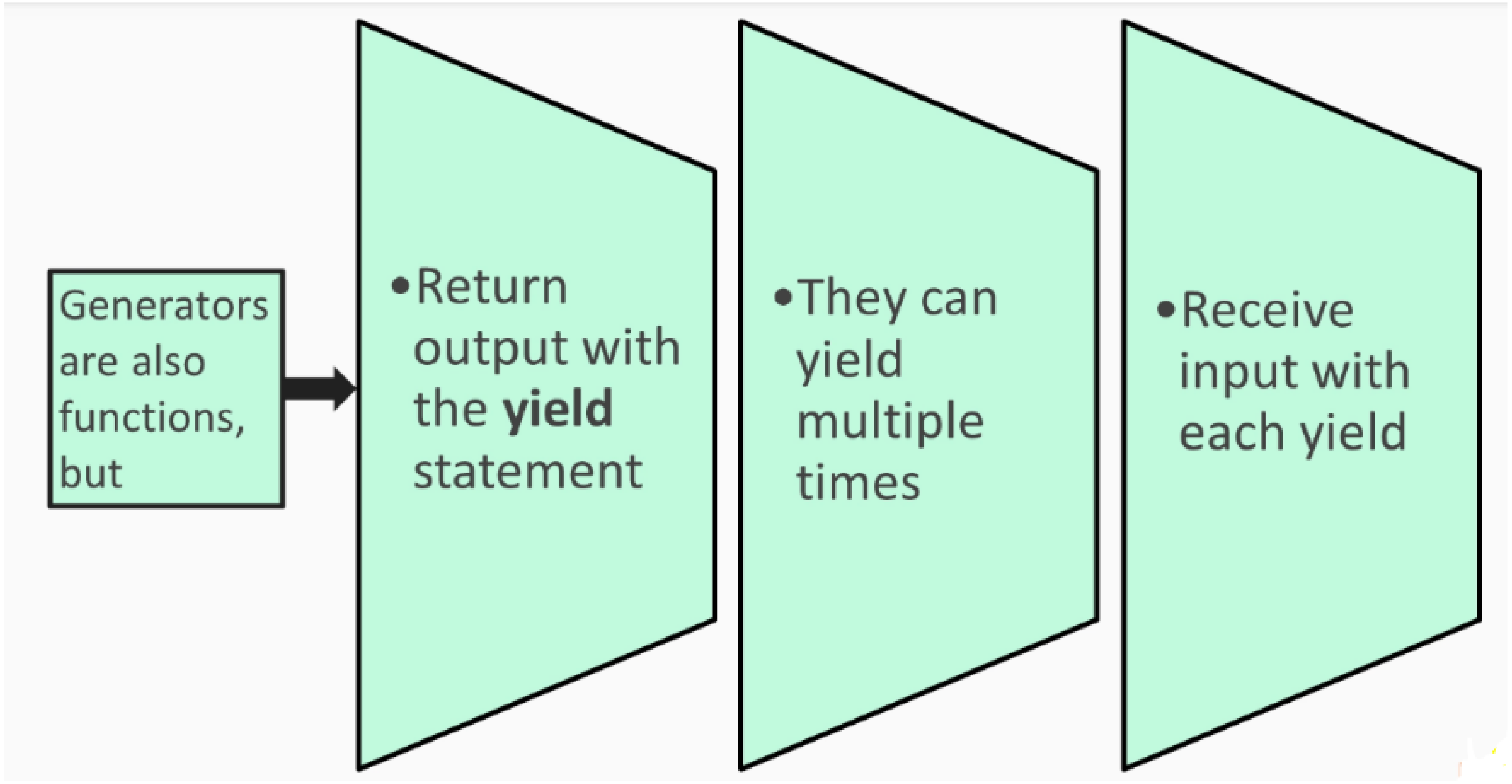
A Generator Is a Function with a Yield Statement

Functions – functions that return



- Return output with the **return** statement
- Once a value is returned, the function is finished.
- Receive input through function argument

Generators – functions that yields


Generators are also functions, but
• Return output with the **yield** statement
• They can yield multiple times
• Receive input with each yield

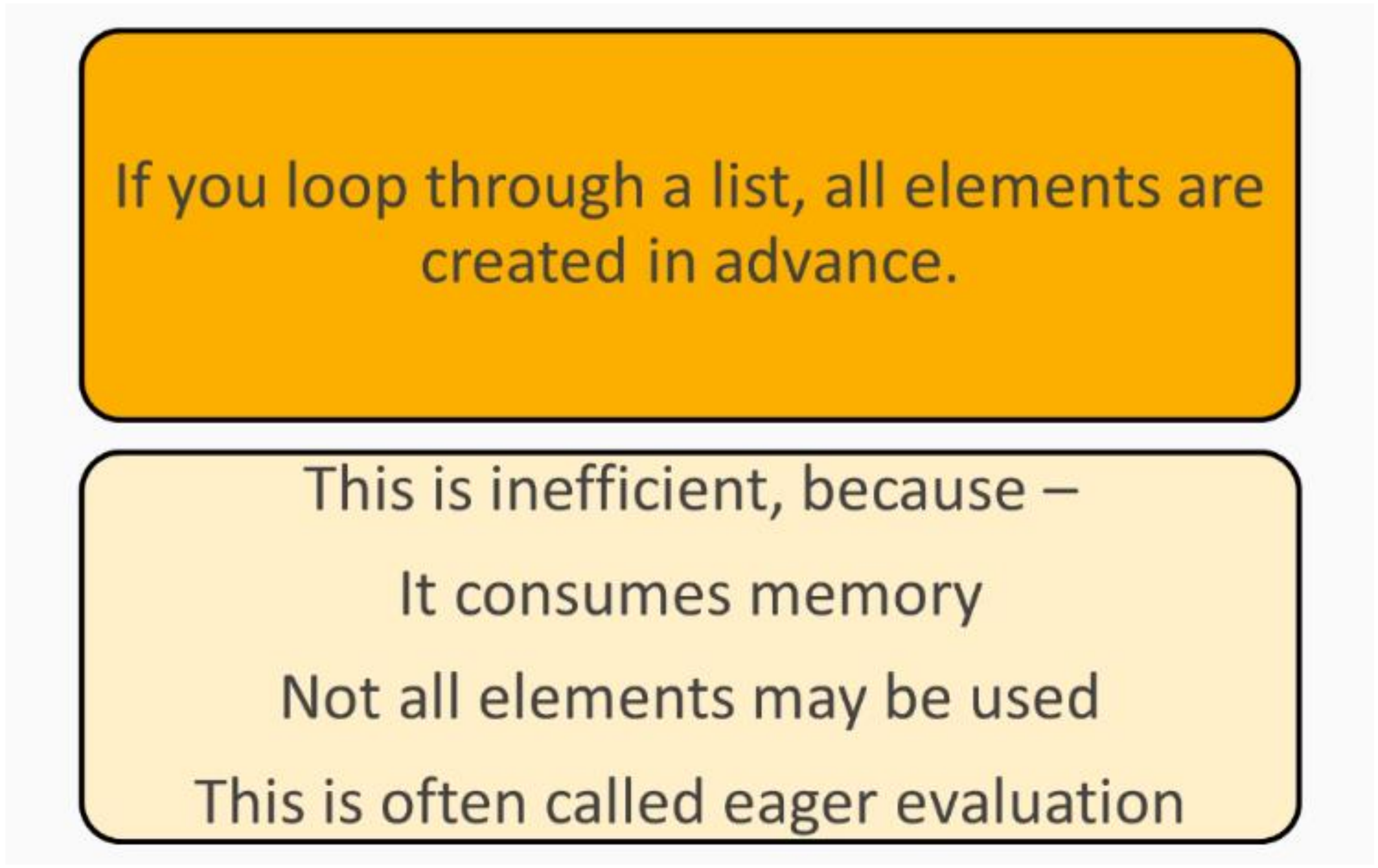
A Generator Is a Type of Iterator

Lazy Evaluation

- What lazy evaluation is
- How you can implement this through iterators and generators

Lazy Evaluation == Evaluate What You Need, When You Need It.

Eager list – Eager evaluation



If you loop through a list, all elements are created in advance.

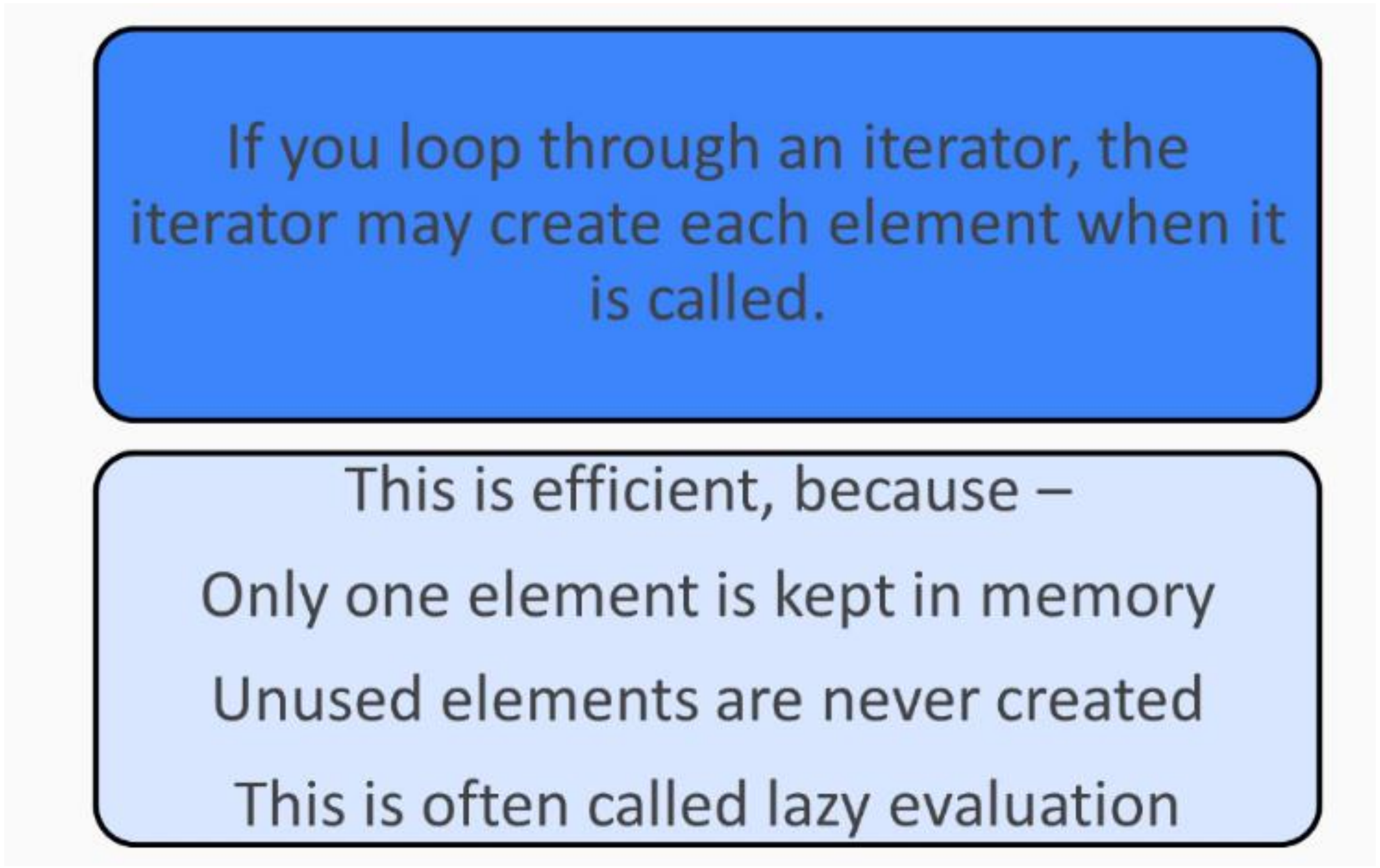
This is inefficient, because –

It consumes memory

Not all elements may be used

This is often called eager evaluation

Lazy Iterator – Lazy Evaluation



If you loop through an iterator, the iterator may create each element when it is called.

This is efficient, because –

Only one element is kept in memory

Unused elements are never created

This is often called lazy evaluation

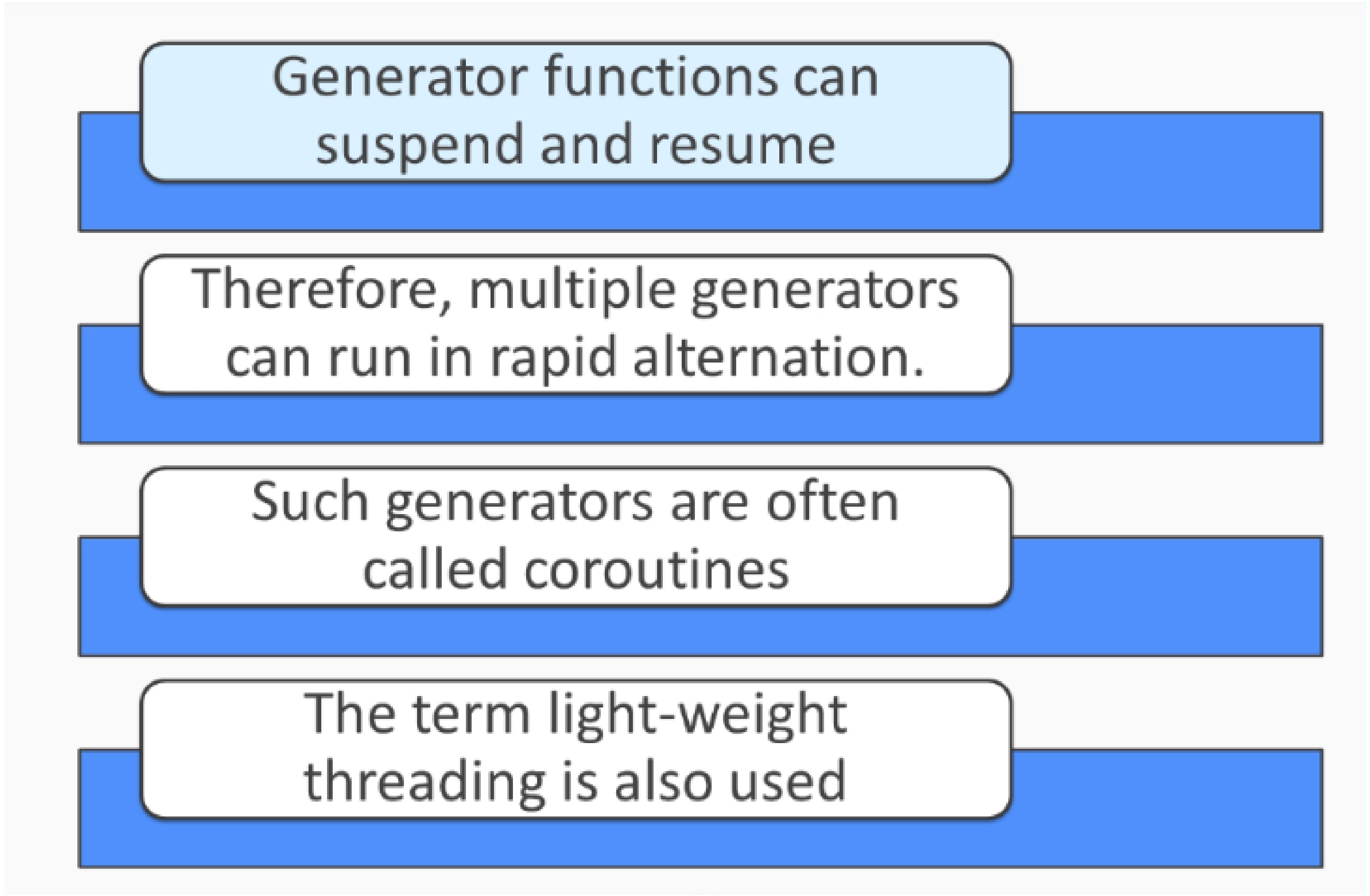
Lazy Evaluation Is Common in Functional Programming

Coroutines – Implementing Concurrency through Generators

- What coroutines are
- How you can implement coroutines through Generators

Rapid Alternation ≈ Parallel

Coroutines – Light Weight Threading



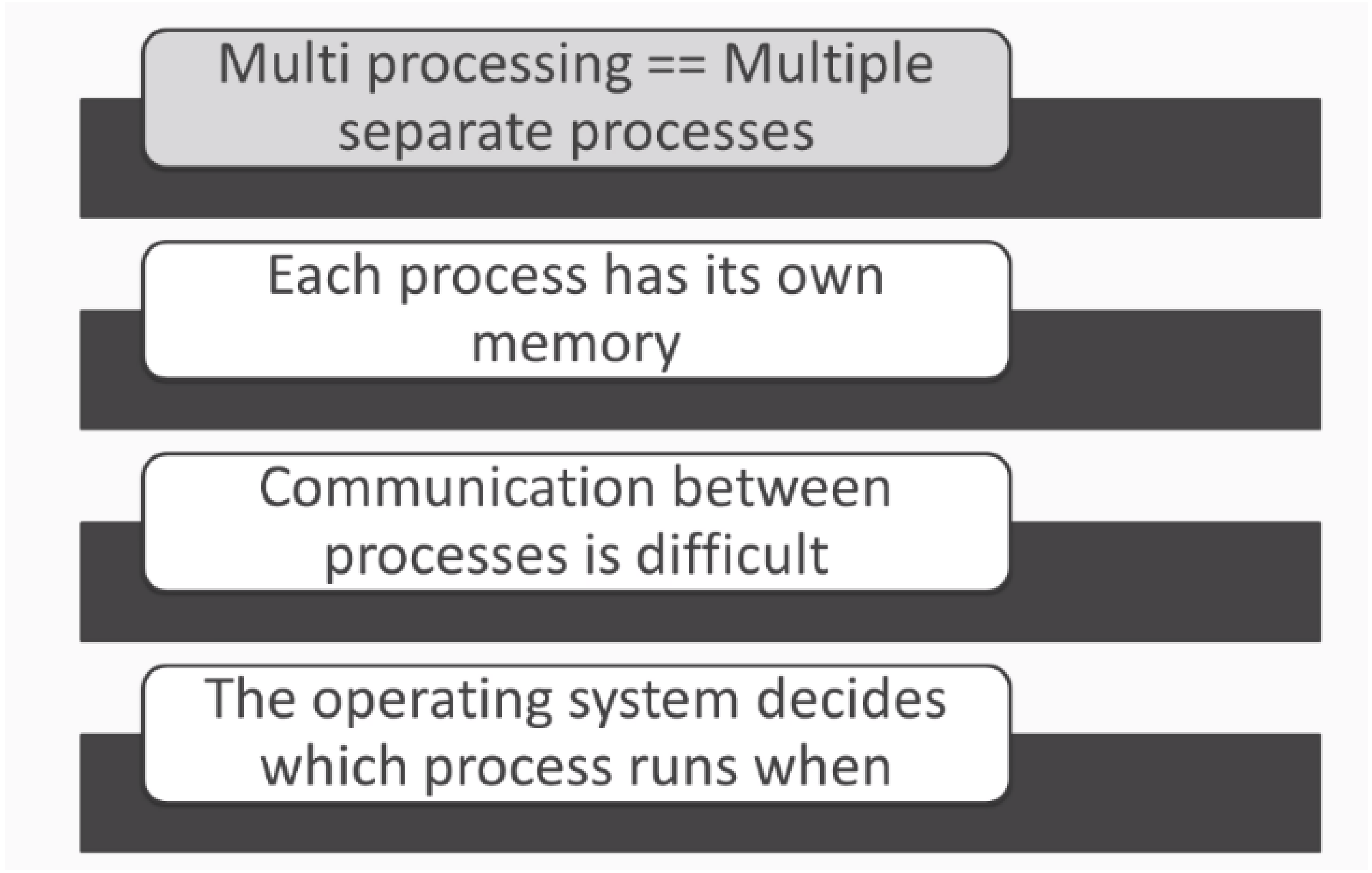
Generator functions can suspend and resume

Therefore, multiple generators can run in rapid alternation.

Such generators are often called coroutines

The term light-weight threading is also used

Different ways to do things in parallel



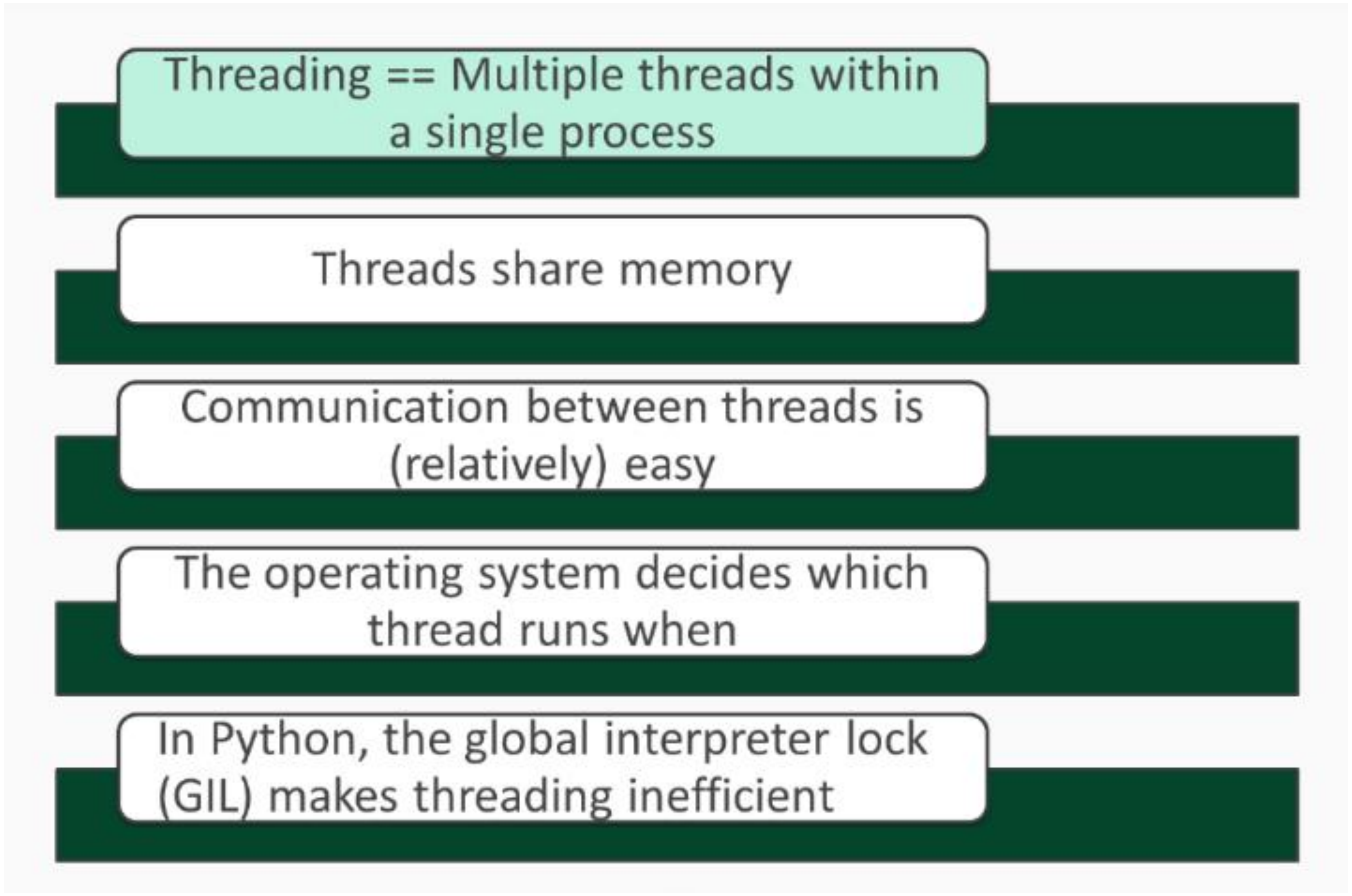
Multi processing == Multiple separate processes

Each process has its own memory

Communication between processes is difficult

The operating system decides which process runs when

Different ways to do things in parallel



Threading == Multiple threads within a single process

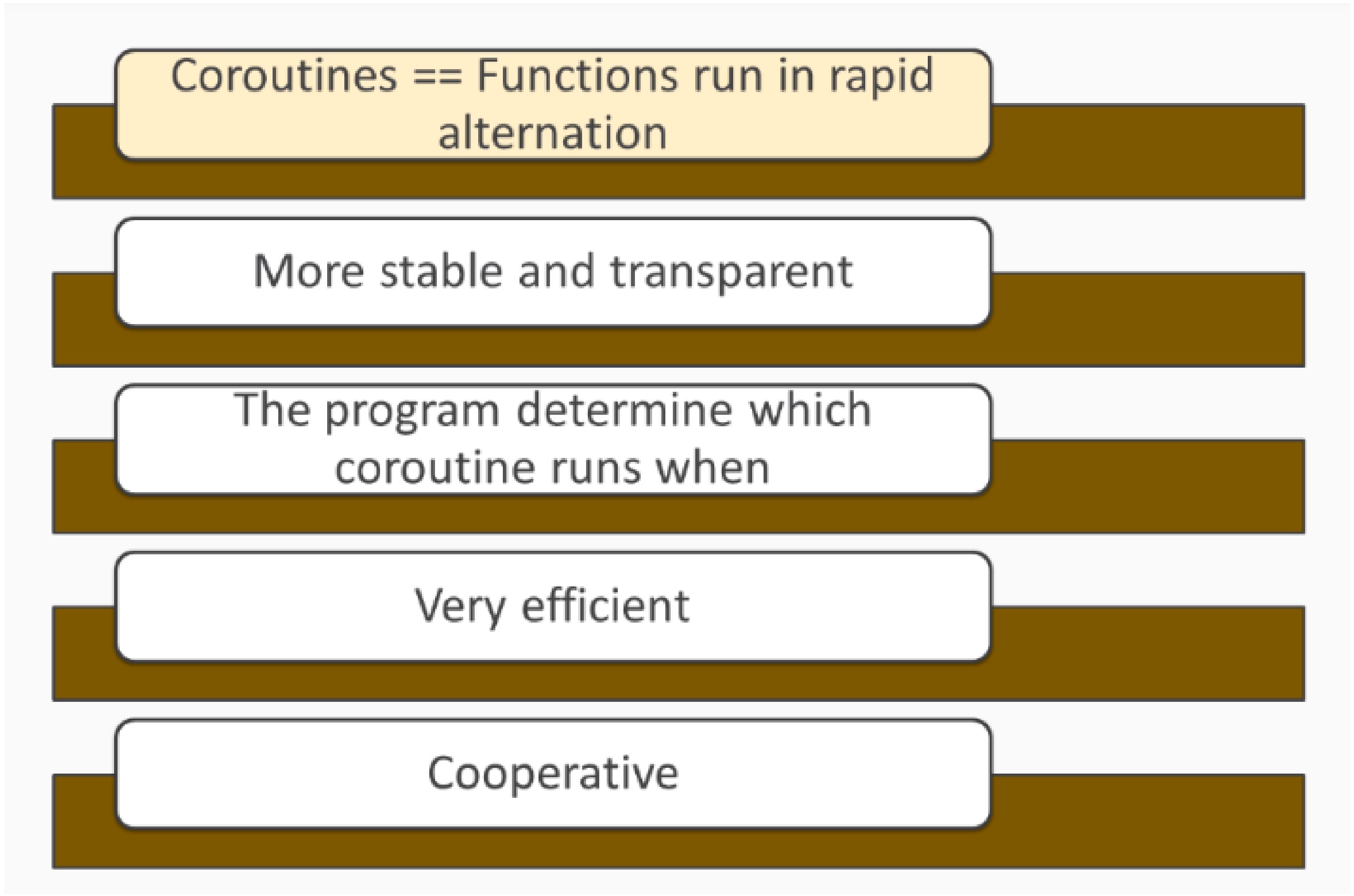
Threads share memory

Communication between threads is (relatively) easy

The operating system decides which thread runs when

In Python, the global interpreter lock (GIL) makes threading inefficient

Different ways to do things in parallel



Coroutines == Functions run in rapid alternation

More stable and transparent

The program determine which coroutine runs when

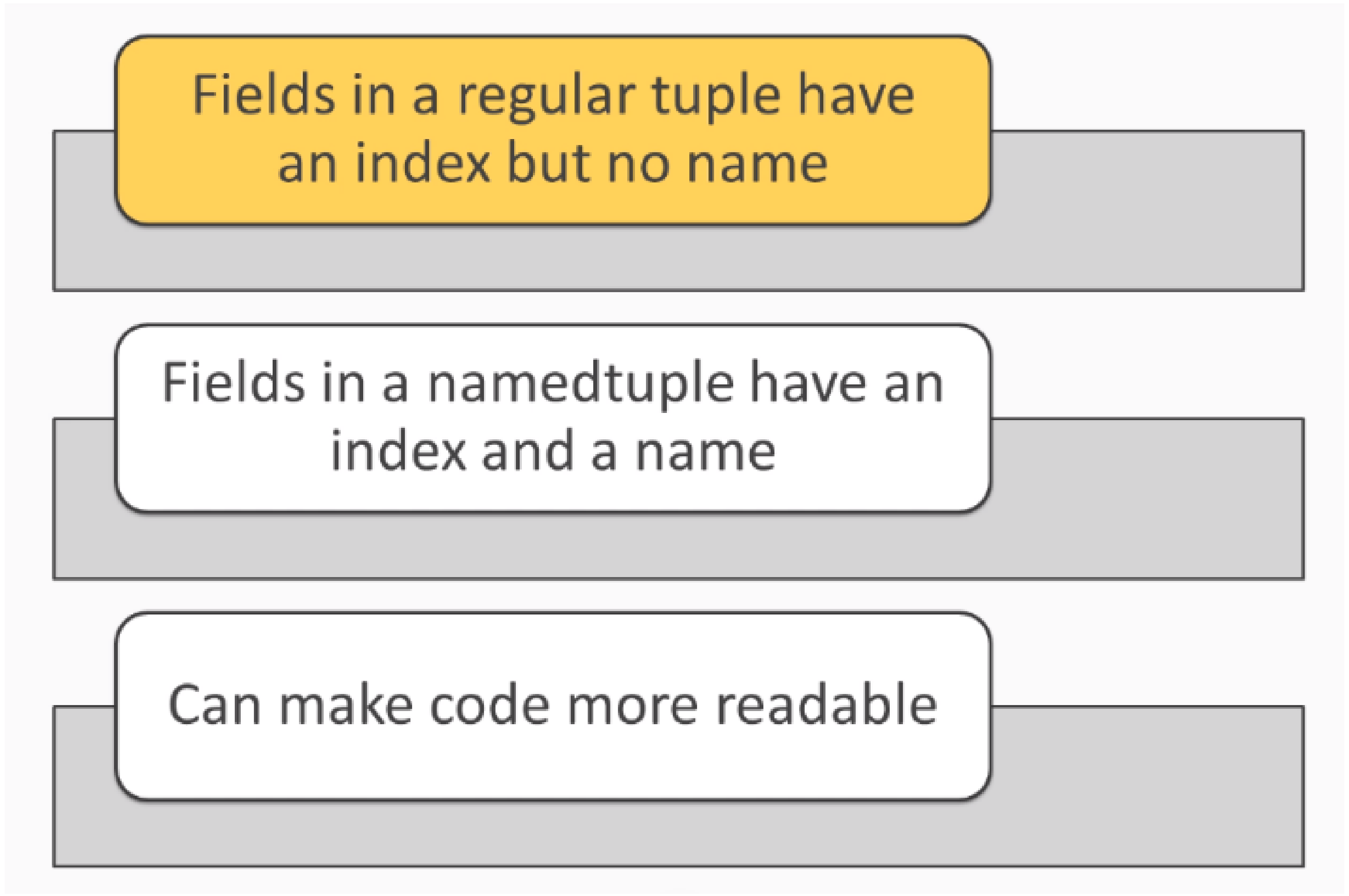
Very efficient

Cooperative

Convenience Iterators – The Collections Module

- The collections module
- Three convenient iterators from this module –
  - namedtuple
  - OrderedDict
  - defaultdict

namedtuple – A tuple with named fields

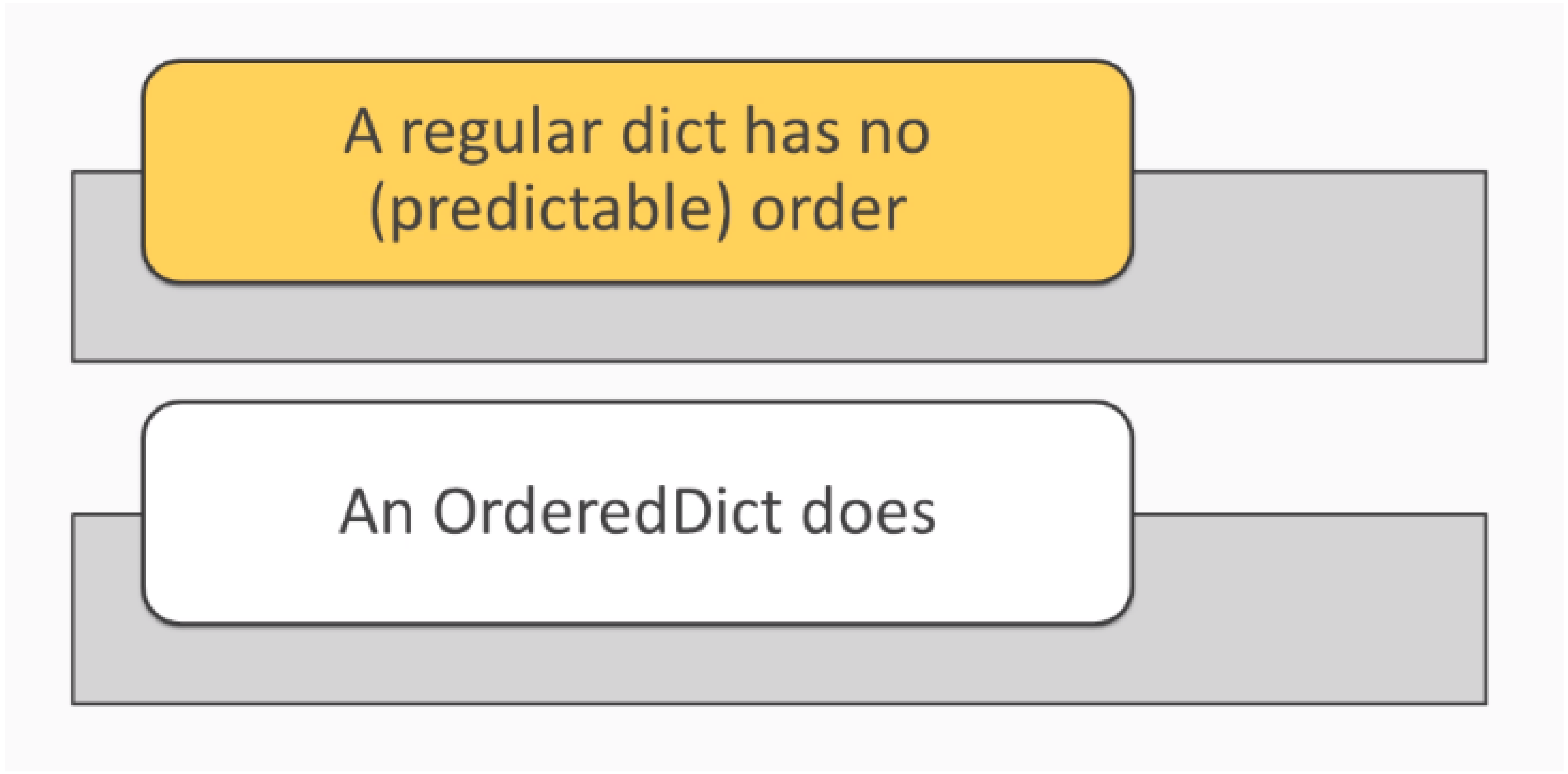


Fields in a regular tuple have an index but no name

Fields in a namedtuple have an index and a name

Can make code more readable

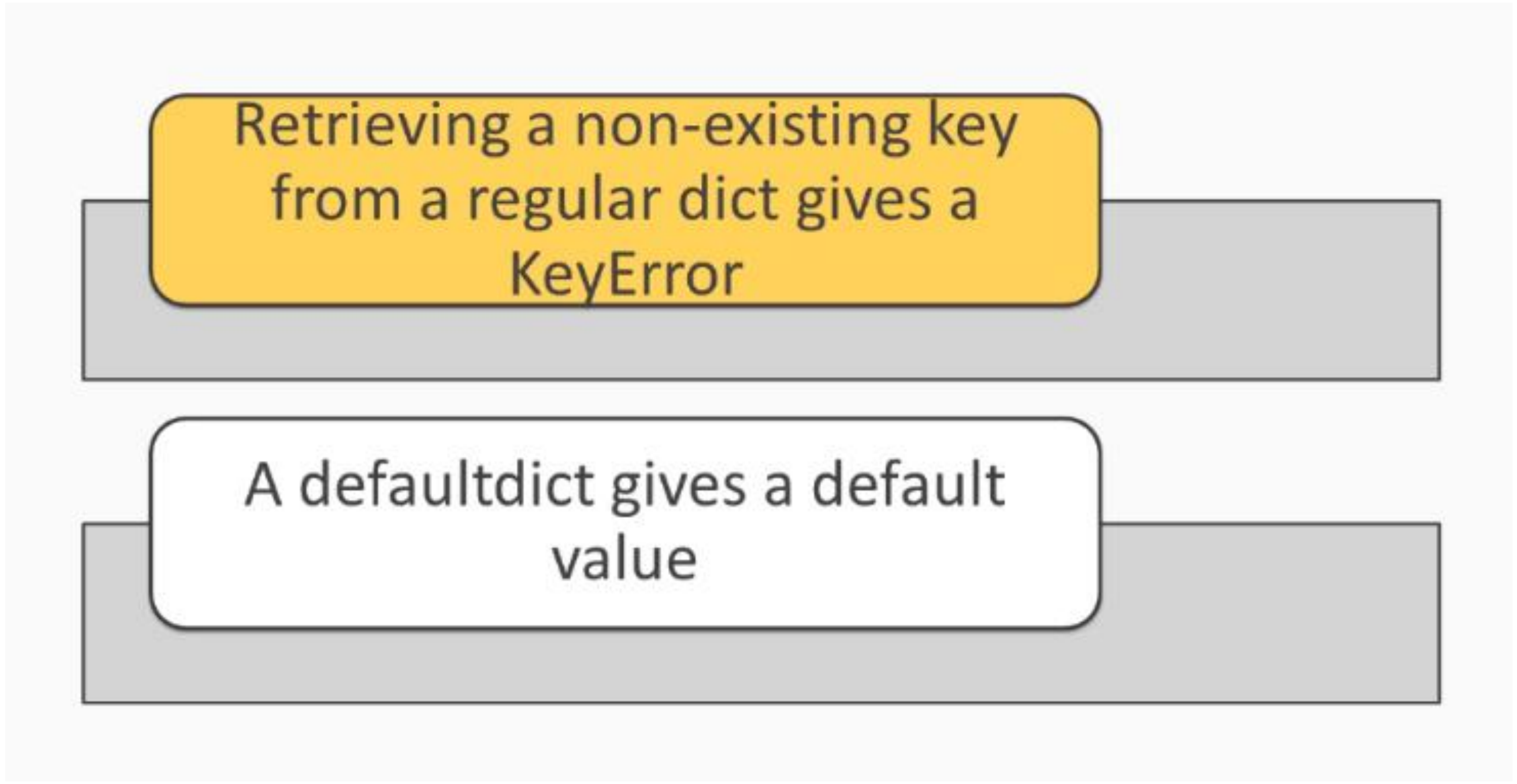
orderDict – A tuple with named fields



A regular dict has no (predictable) order

An OrderedDict does

defaultdict – A Dict with default values


Retrieving a non-existing key from a regular dict gives a KeyError

A defaultdict gives a default value

Summary

- Explained that iterators are objects that contain other objects
- Explored some built-in iterators are such as list, dict, tuple, and set.
- Learned the collections module offers other convenient iterators
- Studied that generators are functions that yield and they are also iterators
- Understood that generators allow for lazy evaluation and coroutines, or light-weight threading.