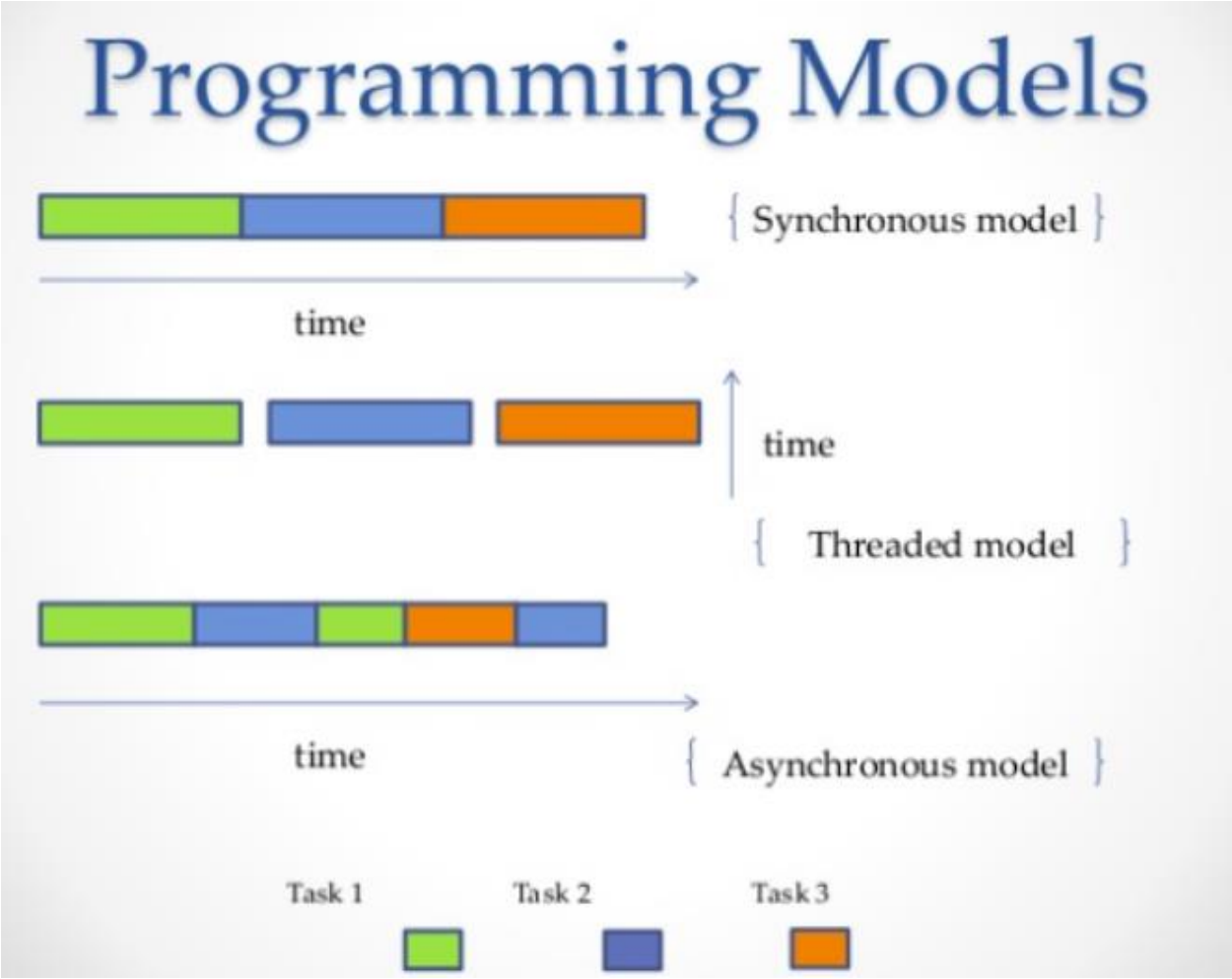
Асинхронное программирование – это вид параллельного программирования, в котором какая-либо единица работы может выполняться отдельно от основного потока выполнения приложения. Когда работа завершается, основной поток получает уведомление о завершении рабочего потока или произошедшей ошибке. У такого подхода есть множество преимуществ, таких как повышение производительности приложений и повышение скорости отклика.

Несмотря на то, что этот вид программирования может быть сложнее традиционного последовательного выполнения, он гораздо более эффективен.

Например, вместо того, что ждать завершения HTTP-запроса перед продолжением выполнения, вы можете отправить запрос и выполнить другую работу, которая ждет своей очереди, с помощью асинхронных корутин в Python.



Как Python умудряется делать несколько вещей одновременно?

1. **Множественные процессы**

Самый очевидный способ – это использование нескольких процессов. Из терминала вы можете запустить свой скрипт два, три, четыре, десять раз, и все скрипты будут выполняться независимо и одновременно. Операционная система сама позаботится о распределении ресурсов процессора между всеми экземплярами. В качестве альтернативы вы можете воспользоваться библиотекой multiprocessing, которая умеет порождать несколько процессов

2. **Множественные потоки**

Еще один способ запустить несколько работ параллельно – это использовать потоки. Поток – это очередь выполнения, которая очень похожа на процесс, однако в одном процессе вы можете иметь несколько потоков, и у всех них будет общий доступ к ресурсам. Однако из-за этого написать код потока будет сложно. Аналогично, все тяжелую работу по выделению памяти процессора сделает операционная система, но глобальная блокировка интерпретатора (GIL) позволит только одному потоку Python запускаться в одну единицу времени, даже если у вас есть многопоточный код. Так GIL на CPython предотвращает многоядерную конкурентность. То есть вы насильно можете запуститься только на одном ядре, даже если у вас их два, четыре или больше.

3. **Корутины и yield**

Корутины – это обобщение подпрограмм. Они используются для кооперативной многозадачности, когда процесс добровольно отдает контроль (yield) с какой-то периодичностью или в периоды ожидания, чтобы позволить нескольким приложениям работать одновременно. Корутины похожи на генераторы, но с дополнительными методами и небольшими изменениями в

том, как мы используем оператор yield. Генераторы производят данные для итерации, в то время как корутины могут еще и потреблять данные.

4. **Асинхронное программирование**

Четвертый способ – это асинхронное программирование, в котором не участвует операционная система. Со стороны операционной системы у вас останется один процесс, в котором будет всего один поток, но вы все еще сможете выполнять одновременно несколько задач.

Asyncio – модуль асинхронного программирования, который был представлен в Python 3.4. Он предназначен для использования корутин и future для упрощения написания асинхронного кода и делает его почти таким же читаемым, как синхронный код, из-за отсутствия callback-ов.

С помощью asyncio вы можете структурировать свой код так, чтобы подзадачи определялись как корутины и позволяли планировать их запуск так, как вам заблагорассудится, в том числе и одновременно. Корутины содержат точки yield, в которых мы определяем возможные точки переключения контекста. В случае, если в очереди ожидания есть задачи, то контекст будет переключен, в противном случае – нет.