Есть несколько разных понятий, связанных с областью параллельных вычислений.

* Конкурентное исполнение (concurrency)
* Параллельное исполнение (parallel execution)
* Многопоточное исполнение (multithreading)
* Асинхронное исполнение (asynchrony)

Каждый из этих терминов строго определен и имеет четкое значение.

**Конкурентность (concurrency)**

Конкурентность (\*) (concurrency) - это наиболее общий термин, который говорит, что одновременно выполняется более одной задачи. Например, вы можете одновременно смотреть телевизор и комментить фоточки в фейсбуке. Винда, даже 95-я могла (\*\*) одновременно играть музыку и показывать фотки.

(\*) К сожалению, вменяемого русскоязычного термина я не знаю. Википедия говорит, что concurrent computing - это параллельные вычисления, но как тогда будет parallel computing по русски?

(\*\*) Да, вспоминается анекдот про Билла Гейтса и многозадачность винды, но, *теоретически* винда могла делать несколько дел одновременно. Хотя и не любых.

Конкурентное исполнение - это самый общий термин, который не говорит о том, каким образом эта конкурентность будет получена: путем приостановки некоторых вычислительных элементов и их переключение на другую задачу, путем действительно одновременного исполнения, путем делегации работы другим устройствам или еще как-то. Это не важно.

*Конкурентное исполнение* говорит о том, что за определенный промежуток времени будет решена более, чем одна задача. Точка.

**Параллельное исполнение**

Параллельное исполнение (parallel computing) подразумевает наличие более одного вычислительного устройства (например, процессора), которые будут *одновременно* выполнять несколько задач.

Параллельное исполнение - это строгое подмножество конкурентного исполнения. Это значит, что на компьютере с одним процессором параллельное программирование - невозможно;)

**Многопоточность**

Многопоточность - это один из способов реализации конкурентного исполнения путем выделения абстракции "рабочего потока" (worker thread).

Потоки "абстрагируют" от пользователя низкоуровневые детали и позволяют выполнять более чем одну работу "параллельно". Операционная система, среда исполнения или библиотека прячет подробности того, будет многопоточное исполнение конкурентным (когда потоков больше чем физических процессоров), или параллельным (когда число потоков меньше или равно числу процессоров и несколько задач физически выполняются одновременно).

**Асинхронное исполнение**

Асинхронность (asynchrony) подразумевает, что операция может быть выполнена кем-то на стороне: удаленным веб-узлом, сервером или другим устройством за пределами текущего вычислительного устройства.

Основное свойство таких операций в том, что начало такой операции требует значительно меньшего времени, чем основная работа. Что позволяет выполнять множество асинхронных операций одновременно даже на устройстве с небольшим числом вычислительных устройств.

**CPU-bound и IO-Bound операции**

Еще один важный момент, с точки зрения разработчика - разница между CPU-bound и IO-bound операциями. CPU-Bound операции нагружают вычислительные мощности текущего устройства, а IO-Bound позволяют выполнить задачу вне текущей железки.

Разница важна тем, что число одновременных операций зависит от того, к какой категории они относятся. Вполне нормально запустить параллельно сотни IO-Bound операций, и надеяться, что хватит ресурсов обработать все результаты. Запускать же параллельно слишком большое число CPU-bound операций (больше, чем число вычислительных устройств) бессмысленно.

Возвращаясь к исходному вопросу: нет смысла выполнять в 1000 потоков метод Calc, если он является CPU-Intensive (нагружает центральный процессор), поскольку это приведет к падению общей эффективности вычислений. ОС-ке придется переключать несколько доступных ядер для обслуживания сотен потоков. А этот процесс не является дешевым.

Самым простым и эффективным способом решения CPU-Intensive задачи, заключается в использовании идиомы Fork-Join: задачу (например, входные данные) нужно разбить на определенное число подзадач, которые можно выполнить параллельно. Каждая подзадача должна быть независимой и не обращаться к разделяемым переменным/памяти. Затем, нужно собрать промежуточные результаты и объединить их.

**PYTHON**

Прежде всего стоит отметить, что **многопоточность** - это всего лишь один из вариатов (способов, методов) получения **асинхронности** (асинхронного кода).

***Асинхронность - это техника, которая позволяет вашей программе запустить потенциально длительную задачу и по-прежнему иметь возможность реагировать на другие события во время выполнения этой задачи, вместо того, чтобы ждать, пока эта задача завершится***. <https://it4each.com/blog/threading-multiprocessing-i-asyncio-v-python/>

В Python есть 3 наиболее популярных способа сделать код асинхронным: пакеты **threading**, **multiprocessing** и **asyncio**. (Очень подробно об этом рассказано здесь: <https://youtu.be/0E50FVd5PGQ>).

Если же совсем коротко, то все 3 способа позволяют позволяют "запараллелить" выполнение кода. Но слово "параллелизм" в английском языке имеет 2 значения: **parallelism** (реальный параллелизм) и **concurrency** (квази-парраллелизм, часто ошибочно переводимый как "конкурентность").

### **multiprocessing (parallelism) - это несколько процессов**

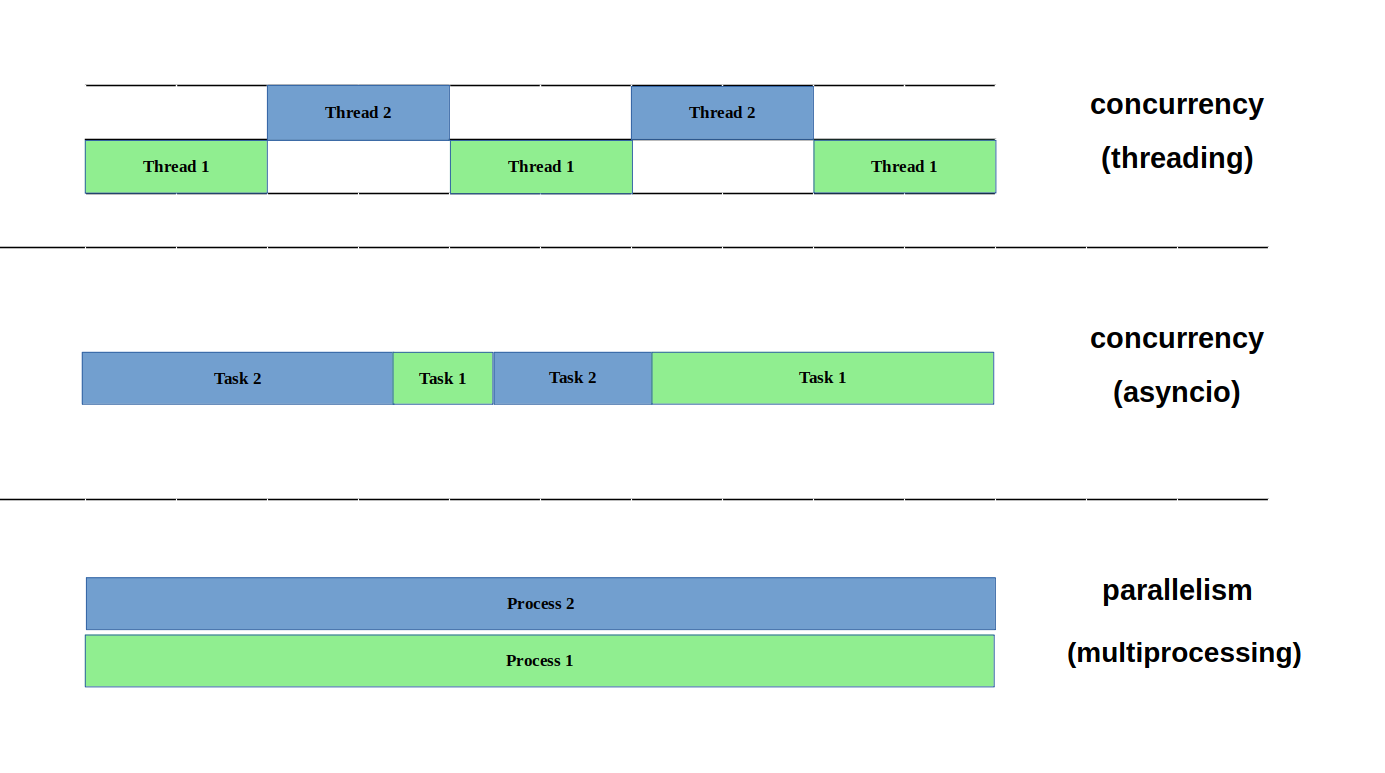
Из трёх перечисленных споособов асинхронности только **multiprocessing** физически запускает несколько самостоятельных процессов на разных ядрах процессора, что делает выполнение программы действительно параллельным (**parallelism**).

Два других (**threading** и **asyncio**) - это квази-параллельное (**concurrency**) выполнение кода, когда идёт только один процесс, но переключение между разными участками кода происходит настолько быстро, что стороннему наблюдателю кажется, что эти части кода выполняются параллельно.

### **threading (concurrency) - это один процесс и несколько потоков**

Разница только в том, что в **threading** (потоки) преключения происходят через равные промежутки времени и делает это операционная система, поэтому исходный код практически не меняется.

### **asyncio (concurrency) - это один процесс и один поток**

В **asyncio**, в отличии от **threading**, точки переключения указывает программист, поэтому в результате итоговый код довольно-таки значительно отличается от исходного. [](https://i.stack.imgur.com/LR34N.png)