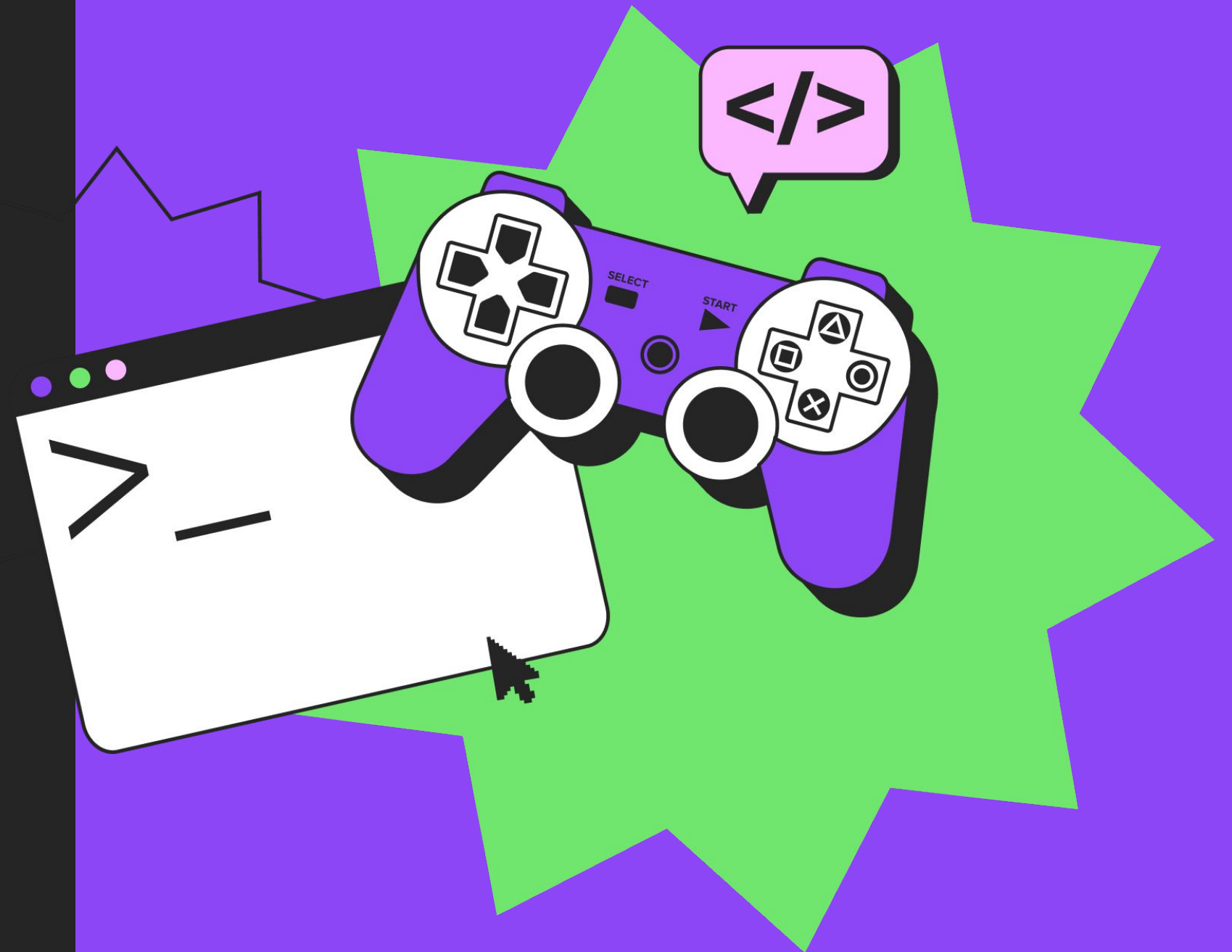


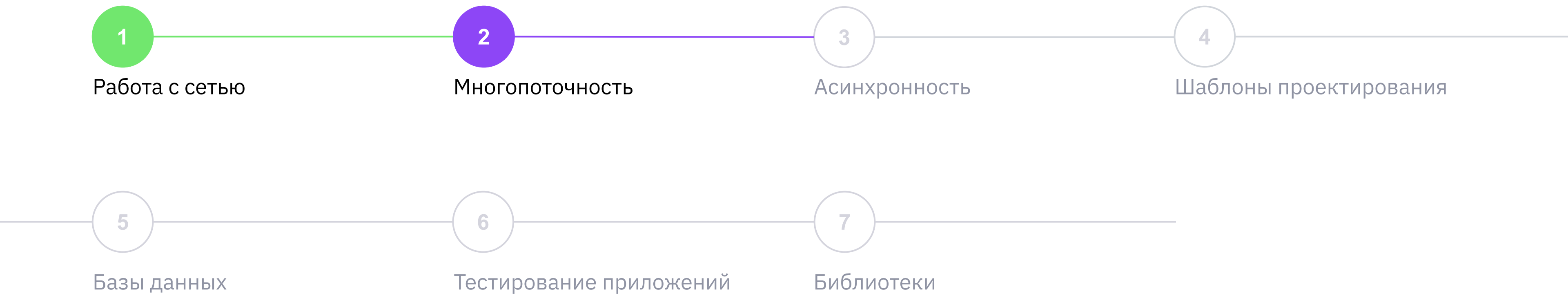
Разработка сетевого приложения на C#

Урок 2
Многопоточность





План курса





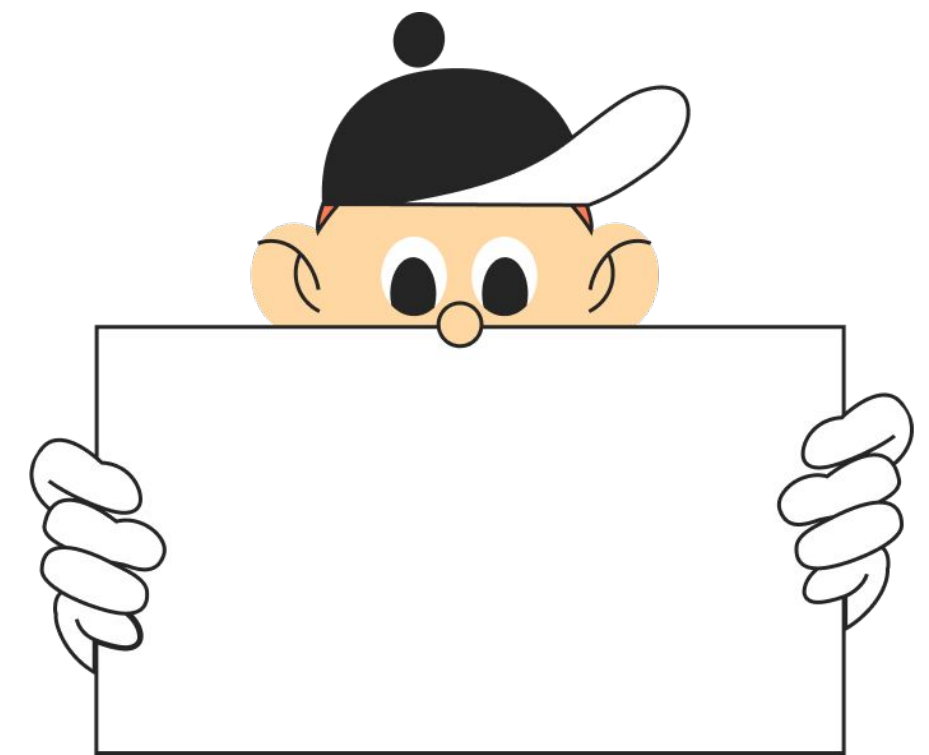
Содержание урока

- Многопоточность
- System.Threading
- Thread
- ThreadPool
- Синхронизация потоков
- Примитивы синхронизации
- Concurrent collections



Многопоточность

Многопоточность – это способность приложения (или среды) выполнять несколько задач (участков кода) одновременно (параллельно).





Поток

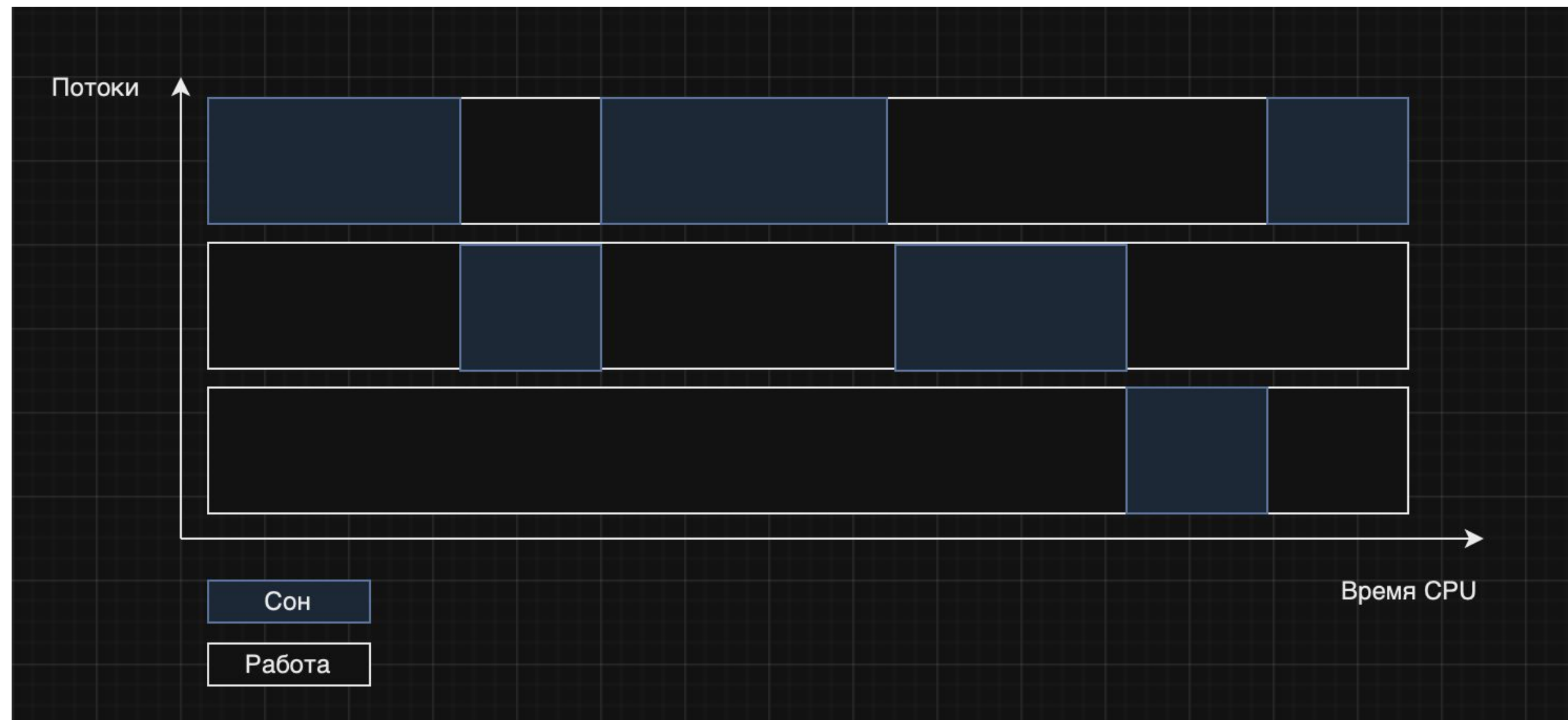
Поток – это последовательность инструкций, выполняемых одновременно с другой последовательностью инструкций. В каждом приложении есть как минимум один поток, который называется главный поток приложения. Код метода Main консольных приложений, например, выполняется в этом потоке.



Потоки позволяют использовать всю мощь современных процессоров, повышая эффективность приложений.



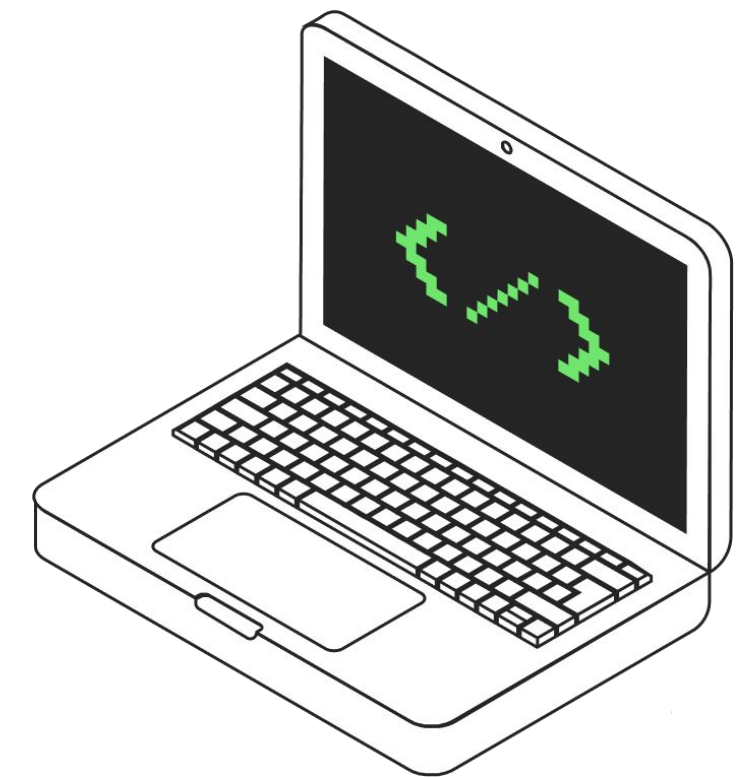
Поток





Пул потоков

Пул потоков позволяет нивелировать все издержки, связанные с созданием и уничтожением потоков, путем предоставления уже готового потока для выполнения произвольного кода.





Синхронизация потоков

Примитивы синхронизации – это специальные классы, объекты которых помогают организовать совместный доступ к чему-либо из разных потоков.





Monitor/lock

Инструкция позволяет блокировать доступ к блоку кода, следующим за инструкцией, для всех потоков, кроме того, который первым выполнит инструкцию.

В общем виде использование инструкции выглядит следующим образом:

```
lock(obj)

{

}
```

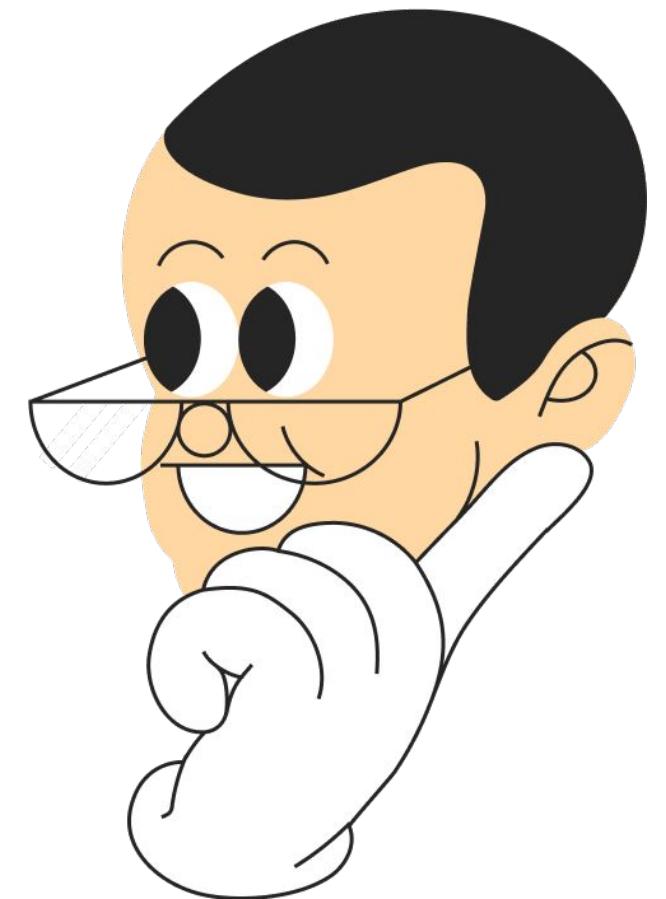
Здесь в качестве объекта выступает любой объект reference-типа.

Давайте покажем, как работает `lock` на практике. Пусть у нас есть несколько потоков, каждый из которых последовательно выводит в цикле свое имя и числа от 0 до 9.



Mutex

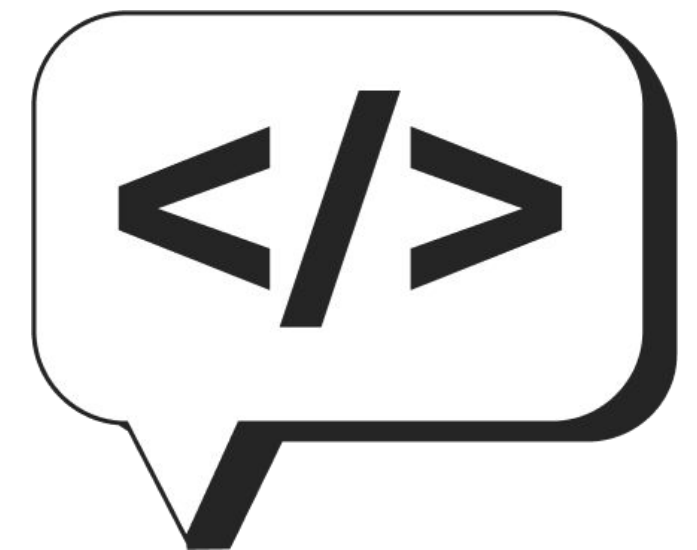
Примитив синхронизации, во многом похожий на `lock`, но, в отличие от последнего, может быть применен для синхронизации доступа независимых процессов/приложений.





AutoResetEvent

С помощью этого примитива синхронизации поток может сигнализировать о наступлении события, ожидаемого в другом потоке (переход в сигнальное состояние), после чего автоматически сбрасывать объект в состояние ожидания сигнала.





ManualResetEvent

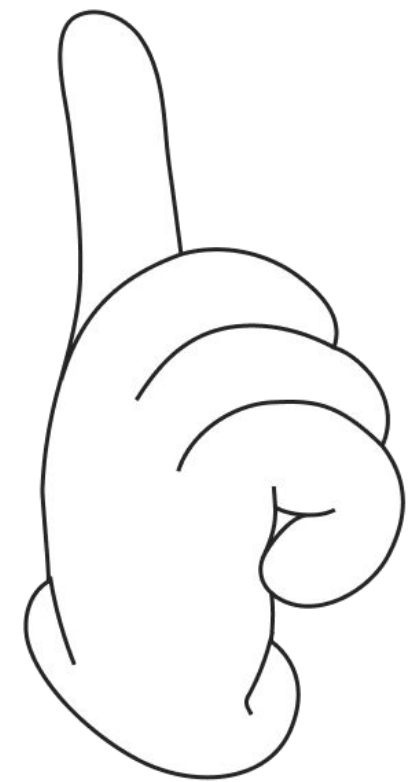
Не переходит в несигнальное состояние автоматически по завершению `WaitOne`. Для перехода в несигнальное состояние используется метод `Reset`.





EventWaitHandle

Примитив синхронизации, наследник абстрактного `WaitHandle`, реализующий синхронизацию потоков в стиле `Manual & AutoResetEvent`, родителем которых он является.





WaitHandle

Абстрактный примитив синхронизации, унаследованный такими классами, как EventWaitHandle, Mutex, Semaphore.





Semaphore

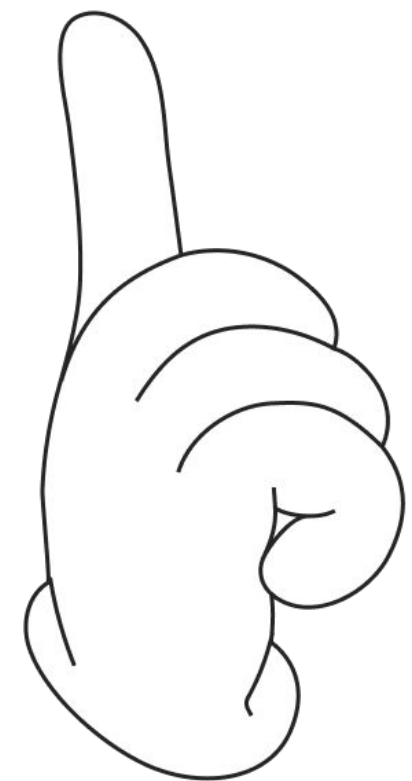
Примитив синхронизации, позволяющий одновременную работу заранее определенного числа потоков, тогда как остальные ждут своей очереди.





Interlocked

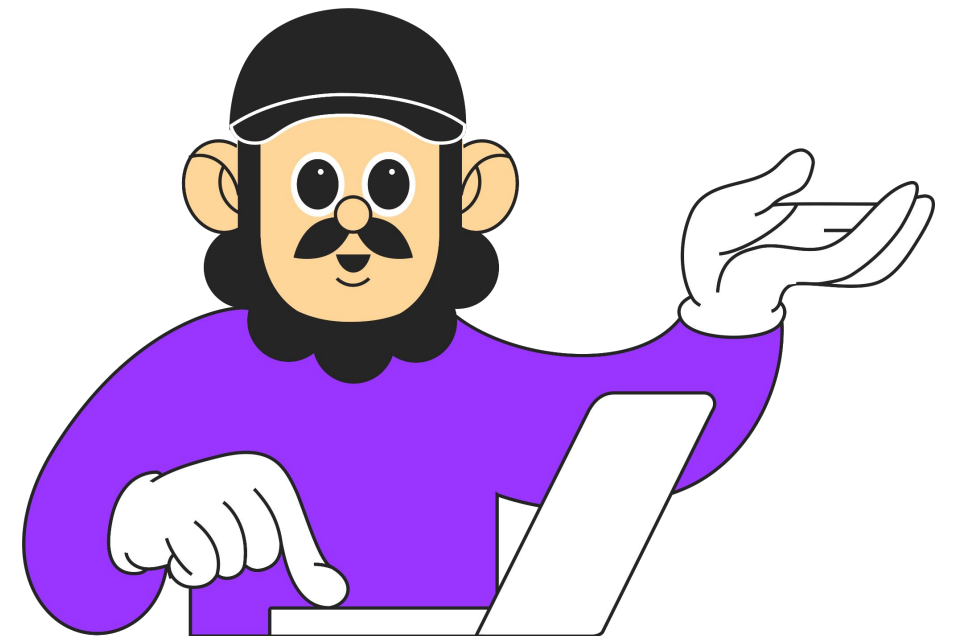
Атомарная операция – это такая операция, которая производится как одно целое и не может быть выполнена наполовину или быть прервана в процессе из другого потока. В силу архитектуры ПК и также современных языков программирования большинство операций не атомарны.





volatile

```
volatile тип имя;
```





Concurrent collections

Пространство имен `System.Collections.Generic` предоставляет ряд потокобезопасных коллекций, работать с которыми можно безопасно из нескольких потоков одновременно. При этом блокировка такая, как, например `lock`, не требуется – коллекции умеют блокировать потоки только при доступе к определенным элементам или операциям, выполняемым в данный момент из другого потока. Примерами таких коллекций являются:


- **`ConcurrentDictionary<TKey,TValue>`** – словарь, позволяющий безопасно работать со своими элементами из разных потоков.
- **`BlockingCollection<T>`** – коллекция позволяет добавлять и читать элементы из разных потоков.
- **`ConcurrentQueue<T>`, `ConcurrentStack<T>`** – многопоточные версии очереди и стека.



Подведение итогов

На этой лекции вы:

- Разобрались с тем, как работает многопоточность
- Узнали про различные примитивы синхронизации
- Узнали про конкурентные коллекции

Спасибо 
за внимание

