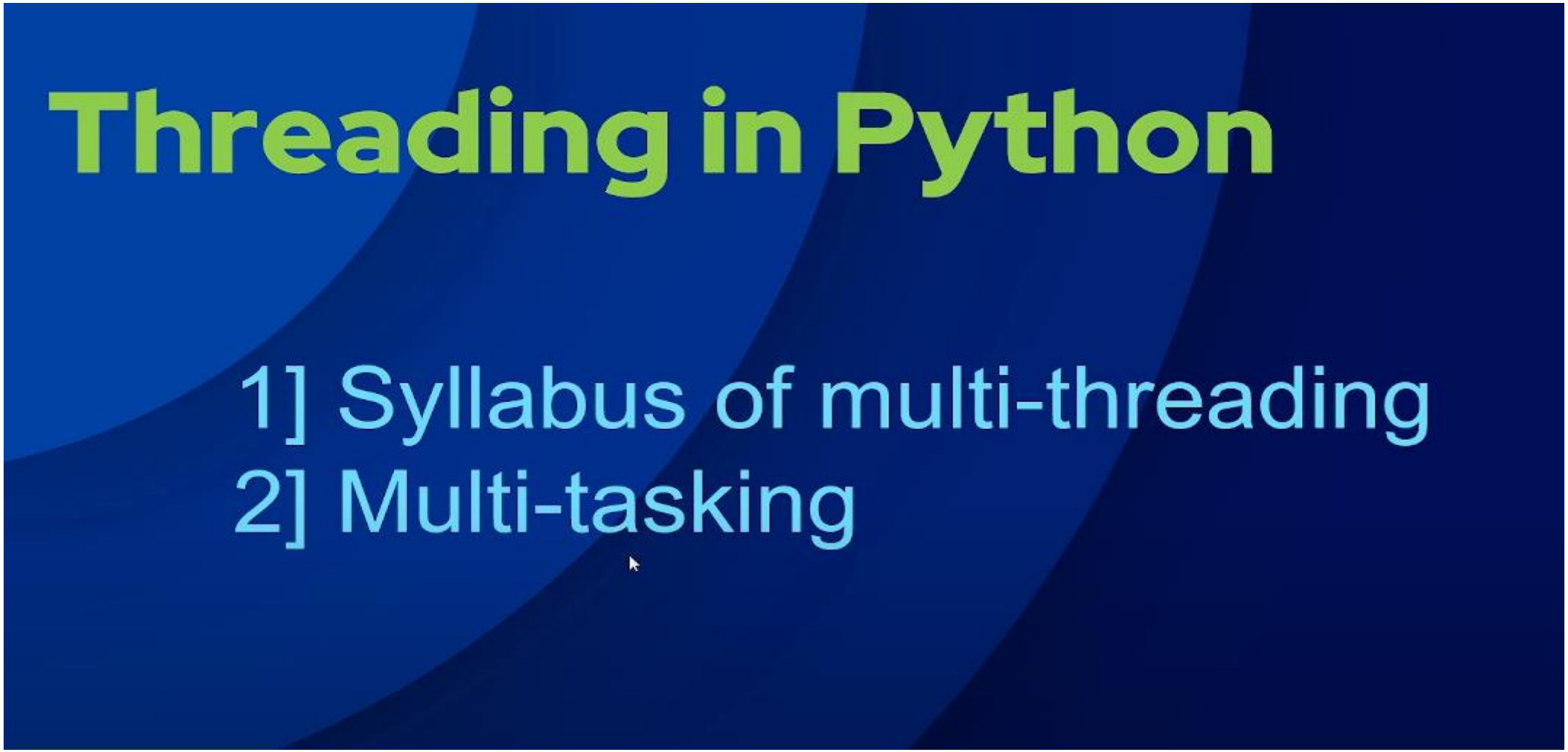
PYTHON- MULTITHREADING

Dr Sivabalan,

Technical Training Advisor

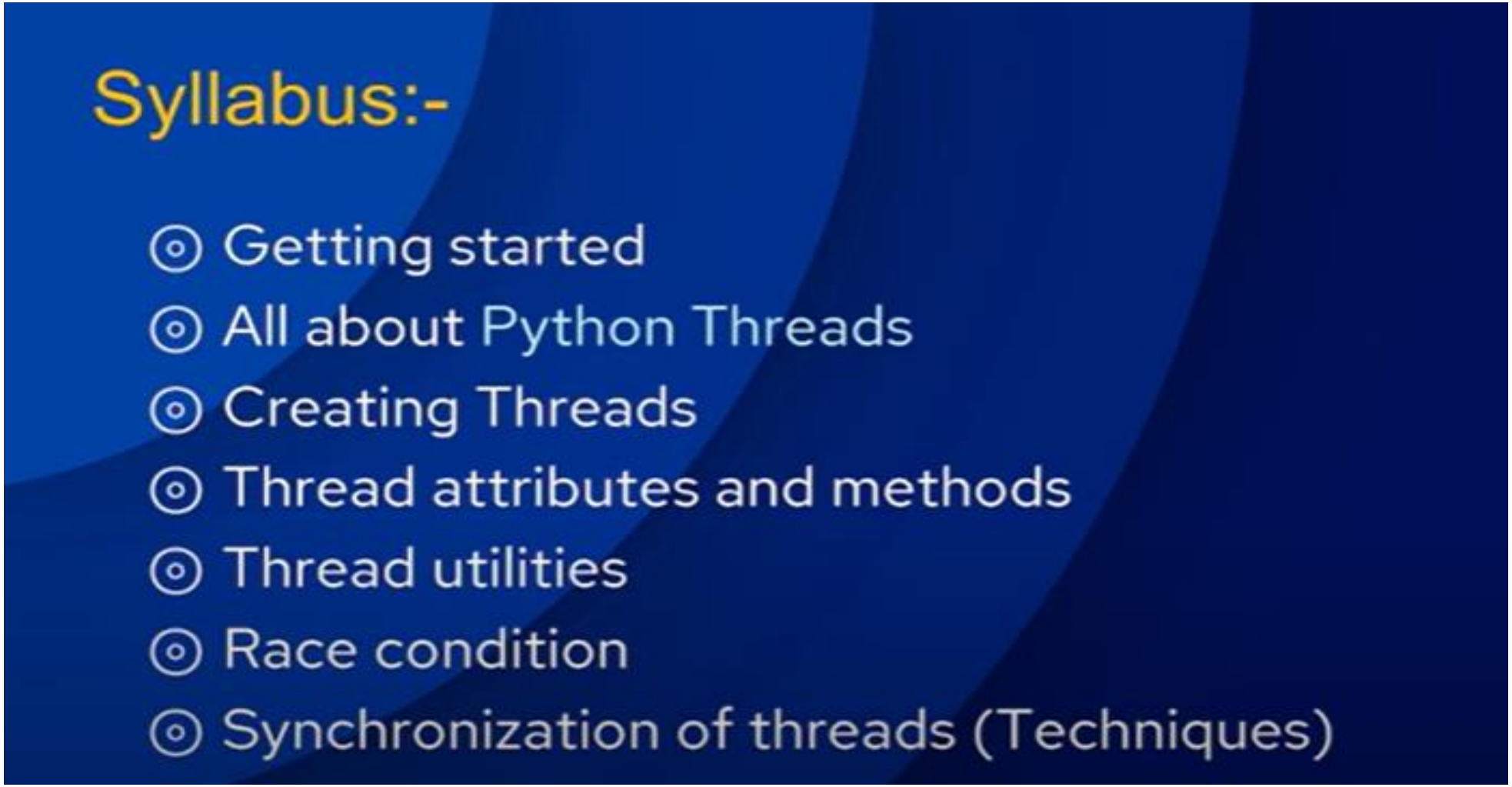
Sivabalan.n@nttdata.com

Multithreading in Python



Agenda

.

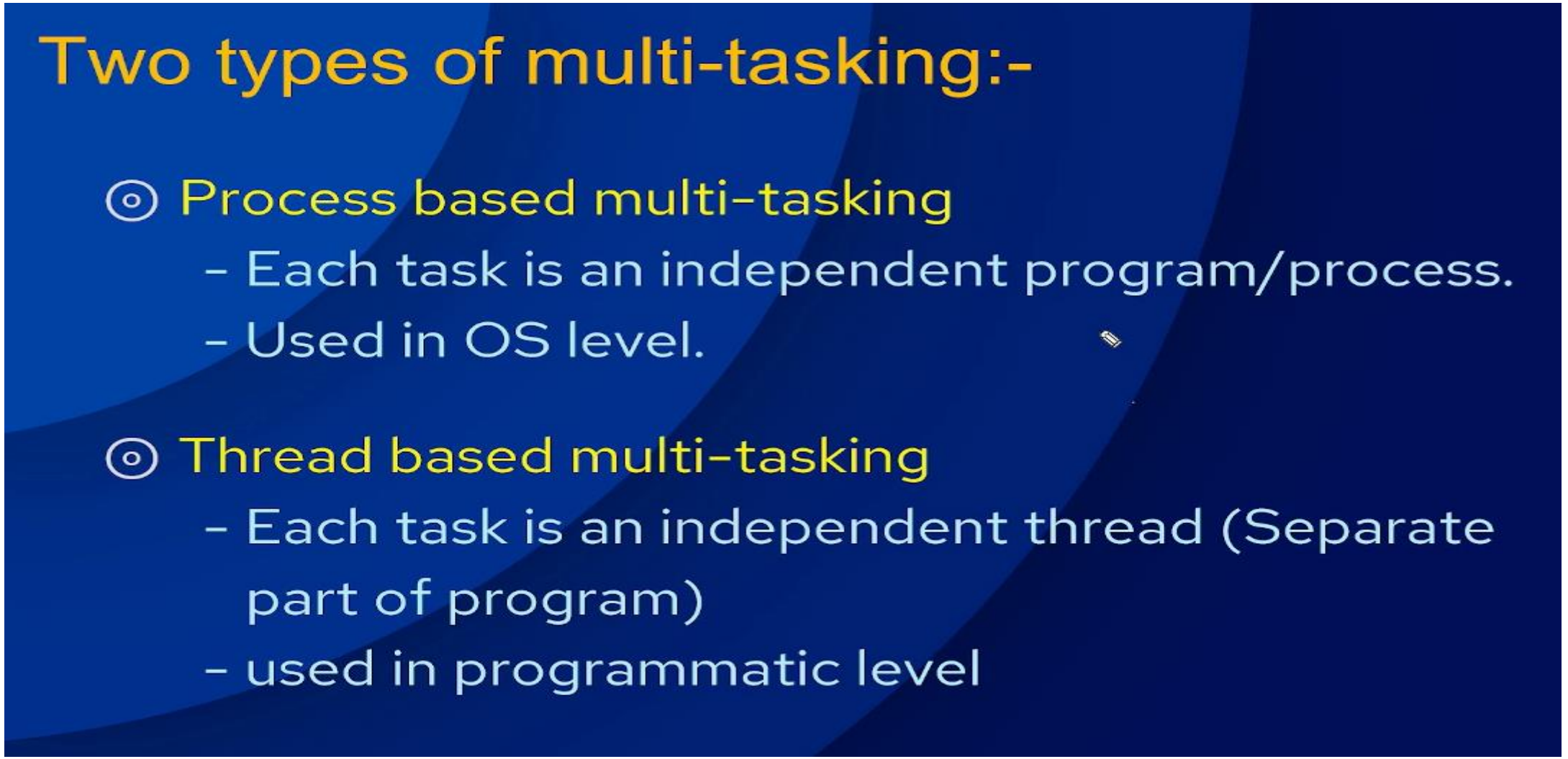


Multithreading in Python

Multi-tasking in python:-

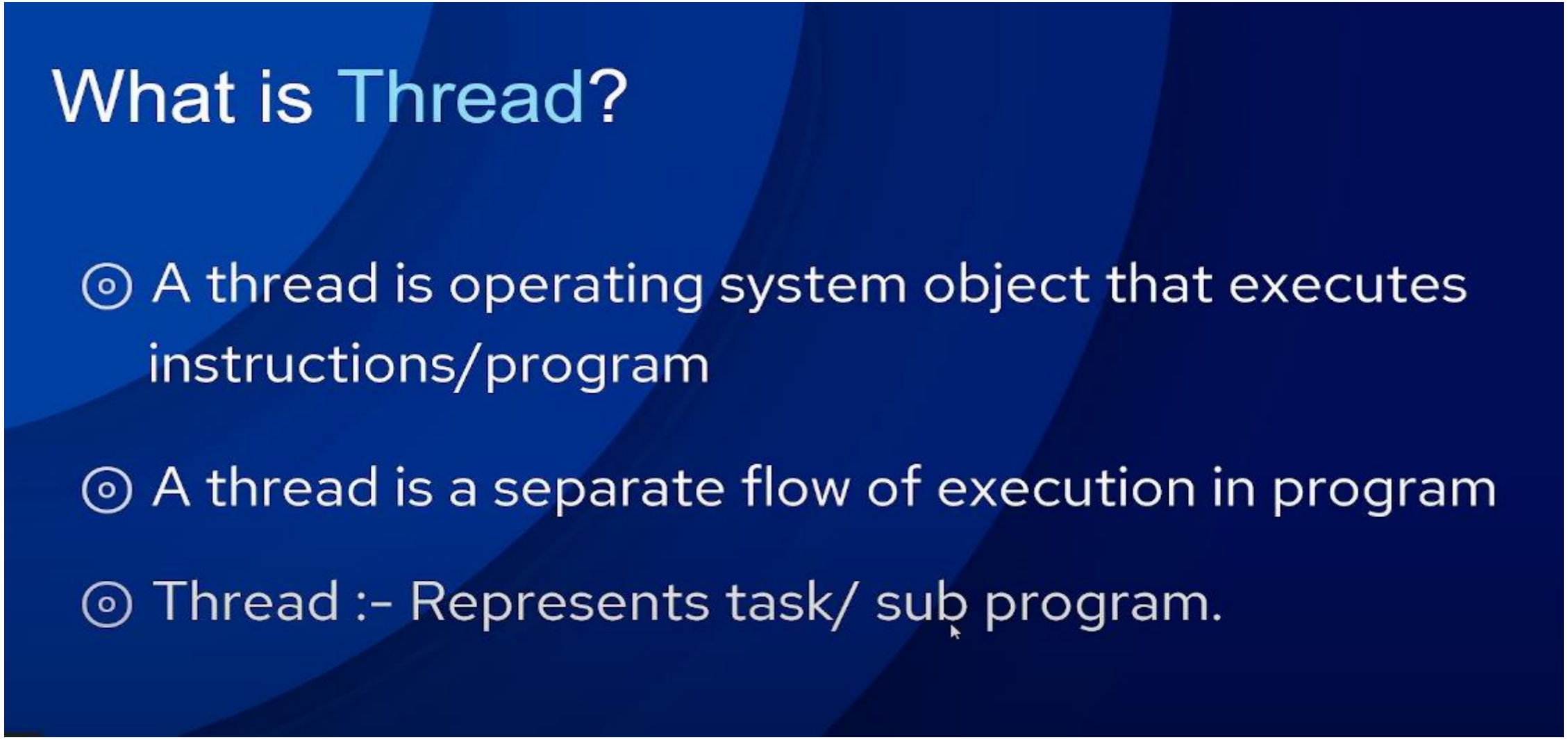
- Executing multiple tasks at the same time.

Multithreading in Python

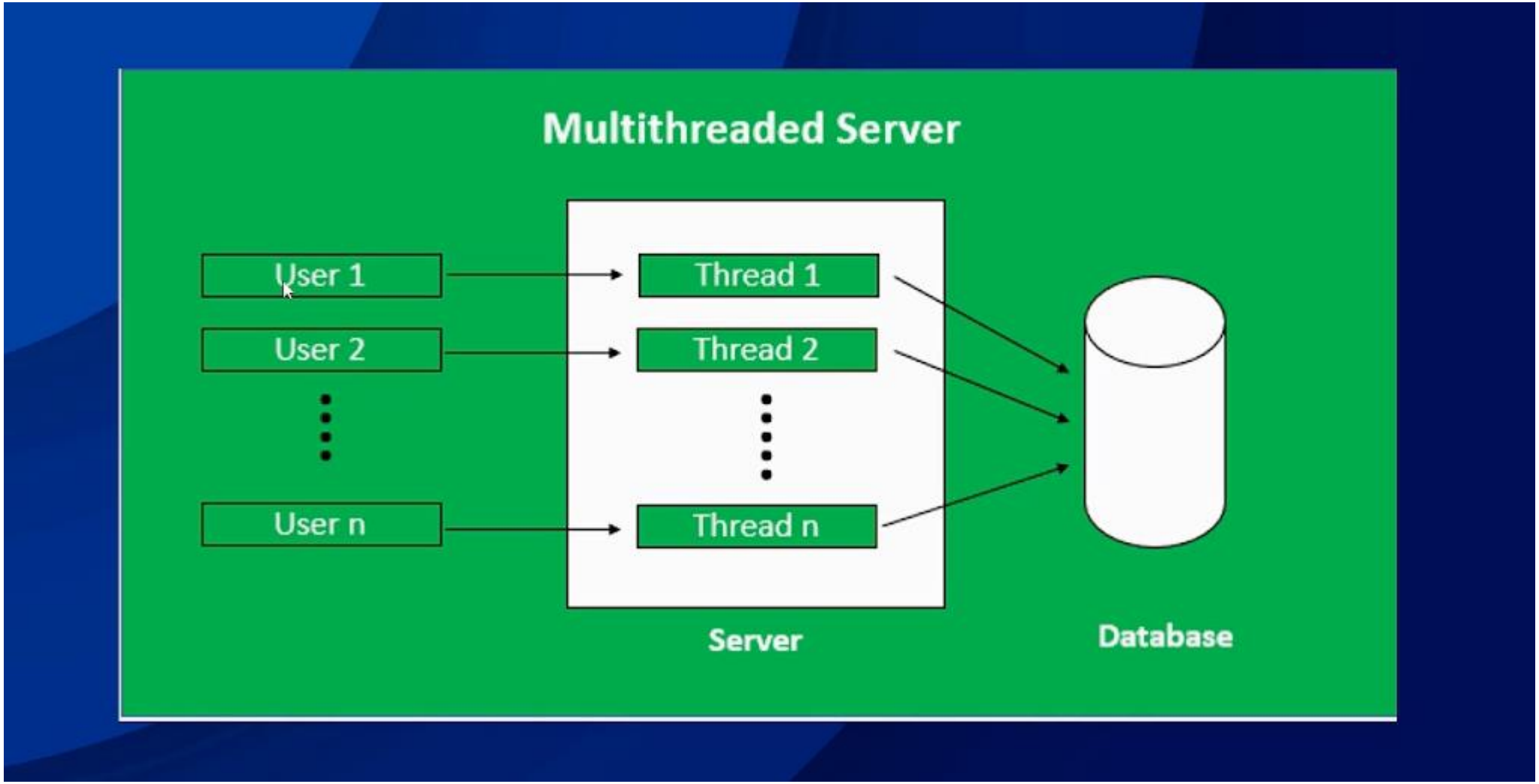


Multithreading in Python

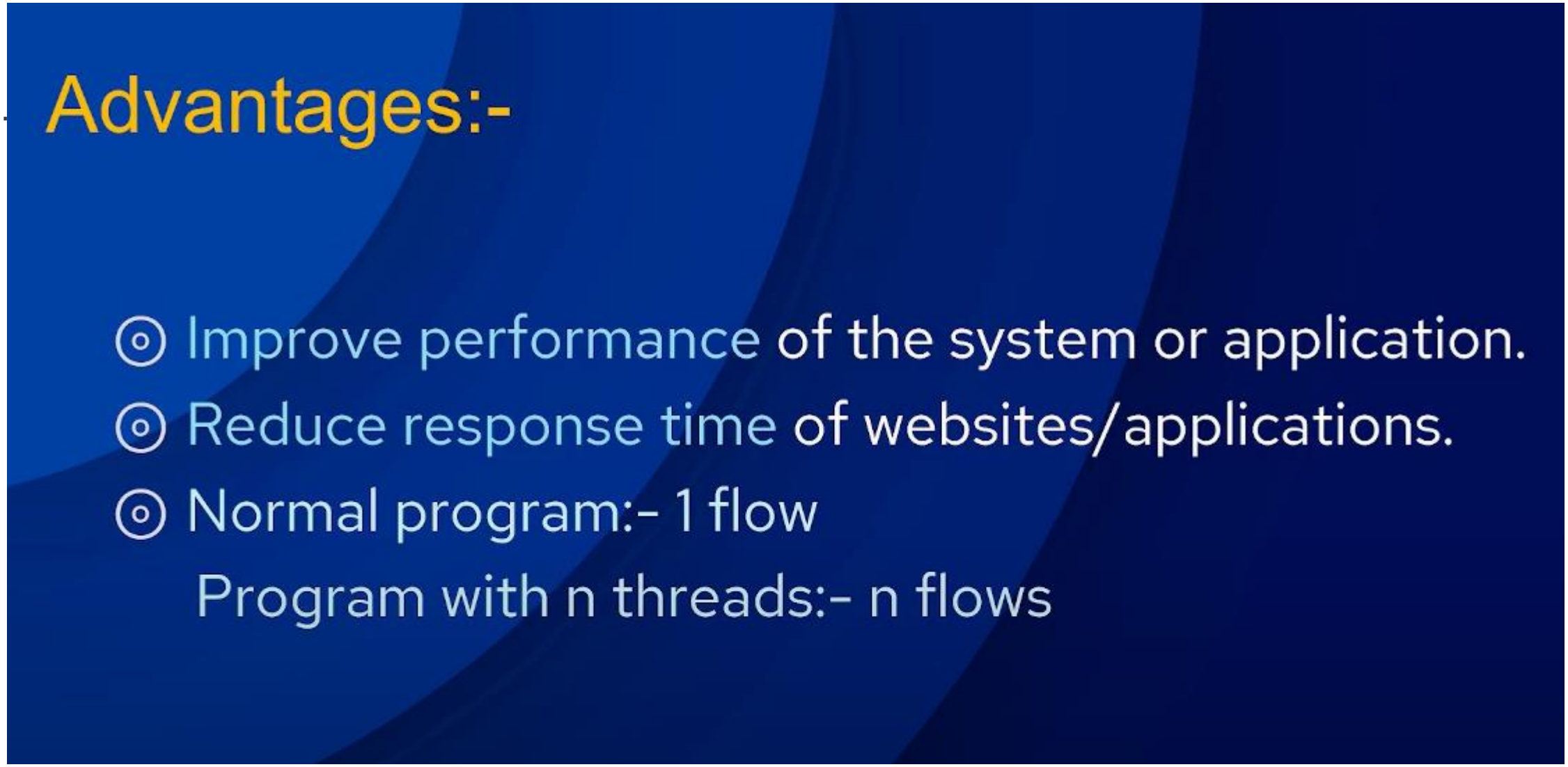
Multithreading in Python



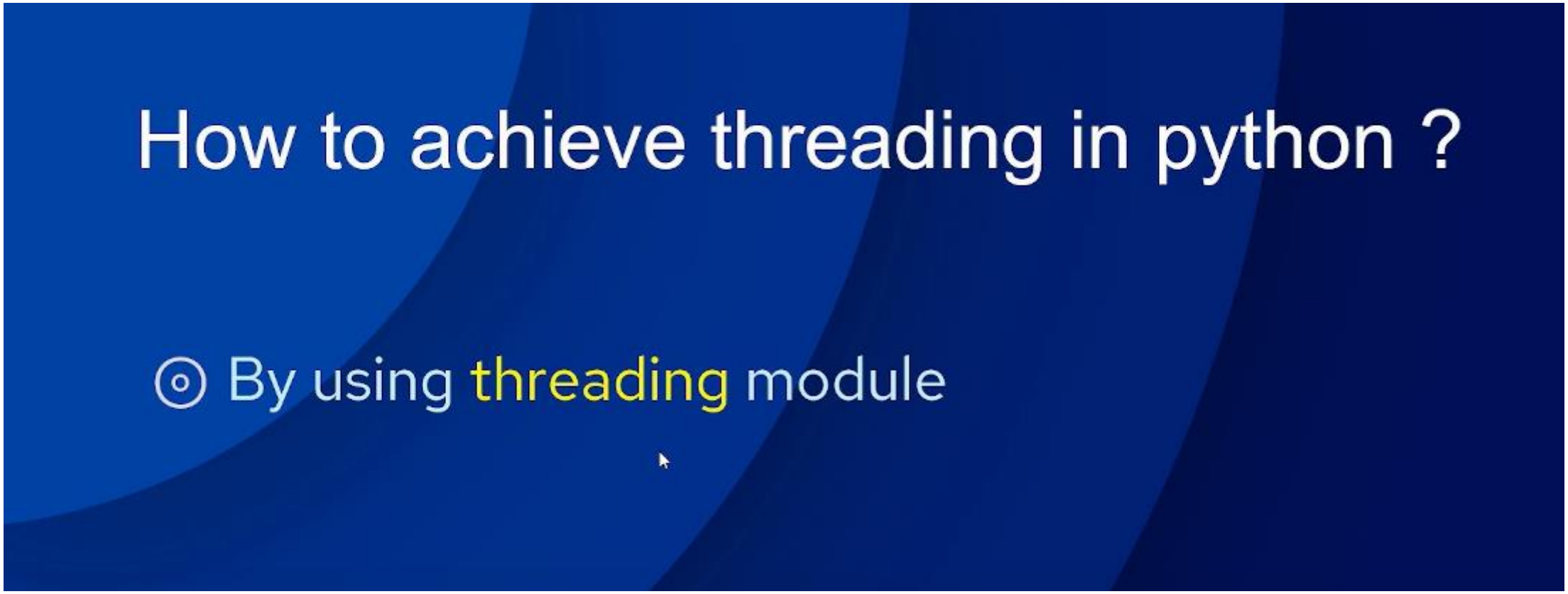
Multithreading in Python



Multithreading in Python



Multithreading in Python



Multithreading in Python

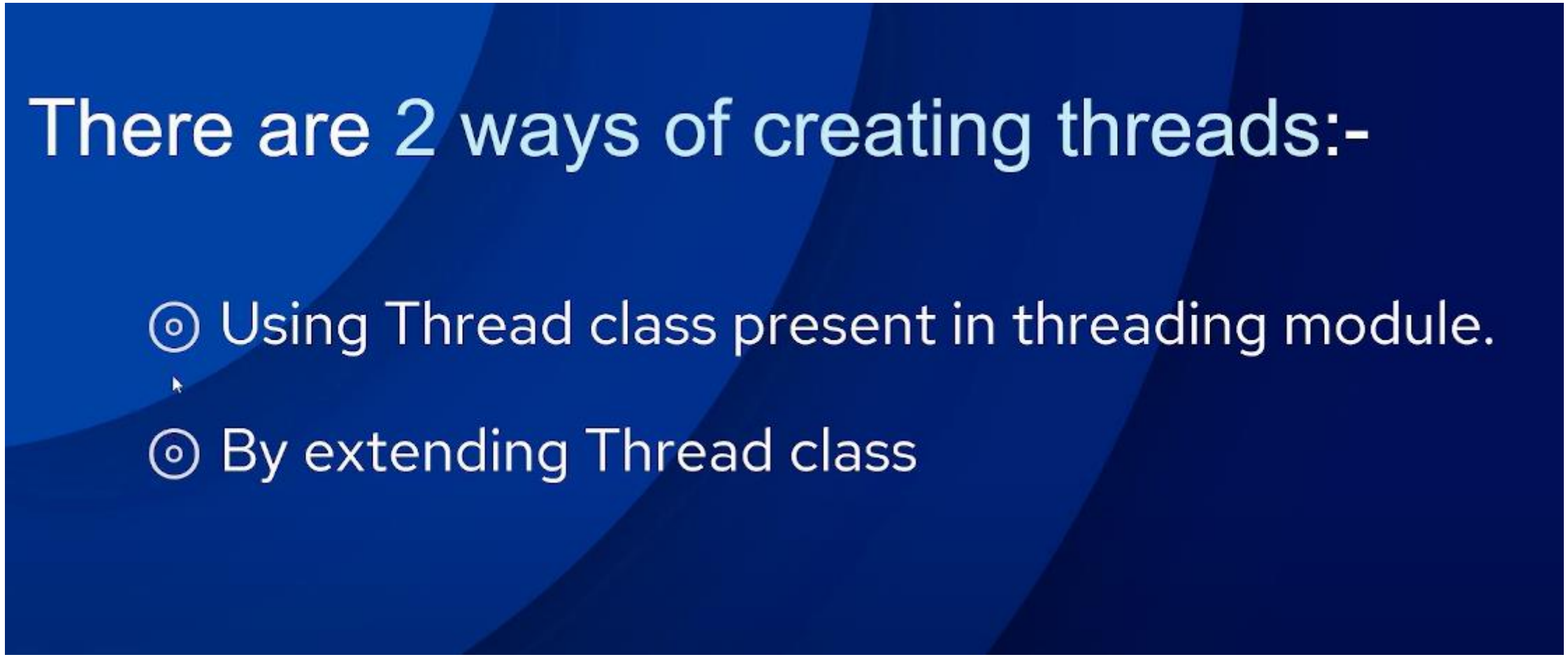
Multithreading in Python

What is Main Thread?

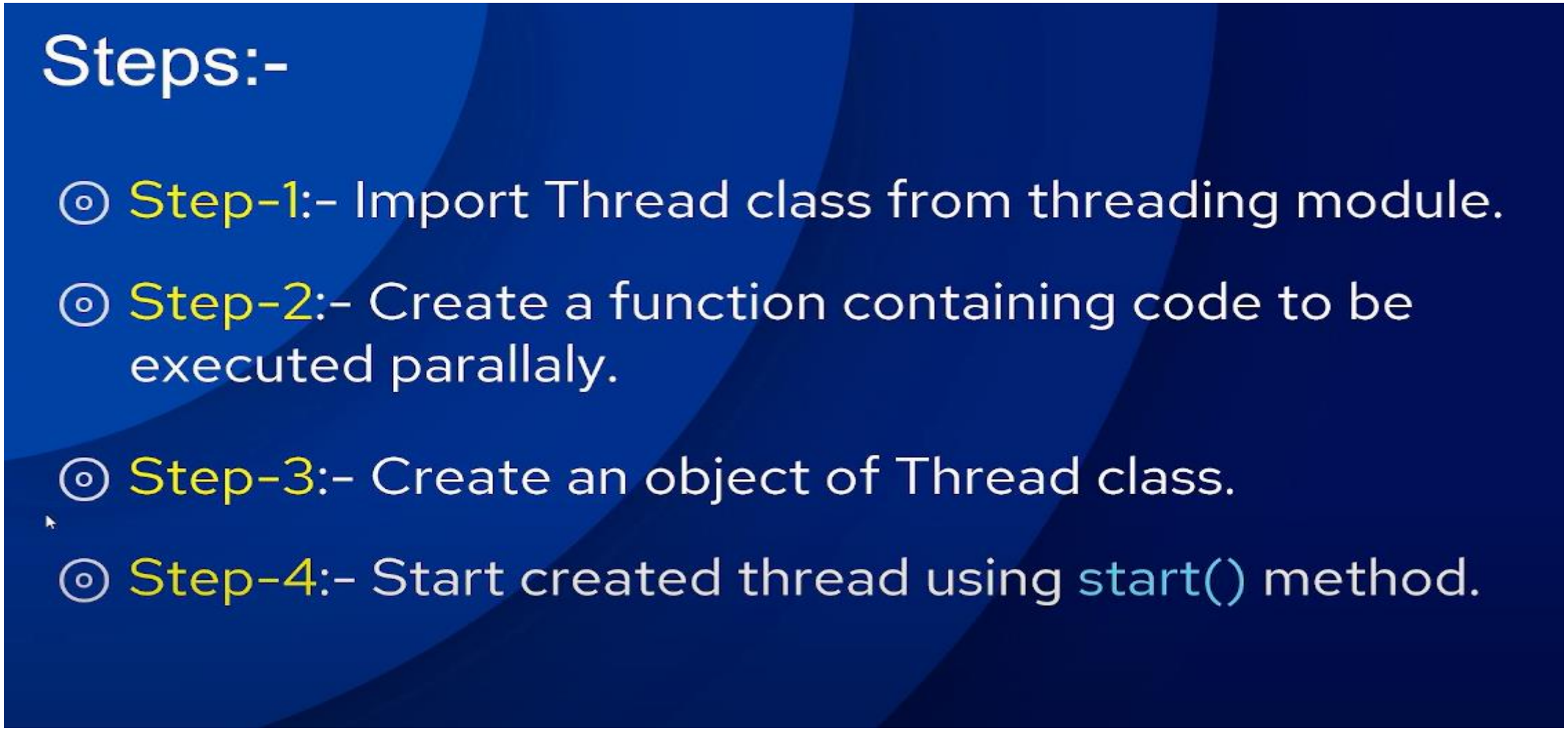
- Run – Python Interpreter starts
  - Python Interpreter request OS for creating one Thread called as Main Thread.
- Any process have at least one default thread called as main thread.
- Main Thread is created by PVM (Interpreter)

Multithreading in Python

Multithreading in Python



Multithreading in Python



Multithreading in Python

Multithreading in Python

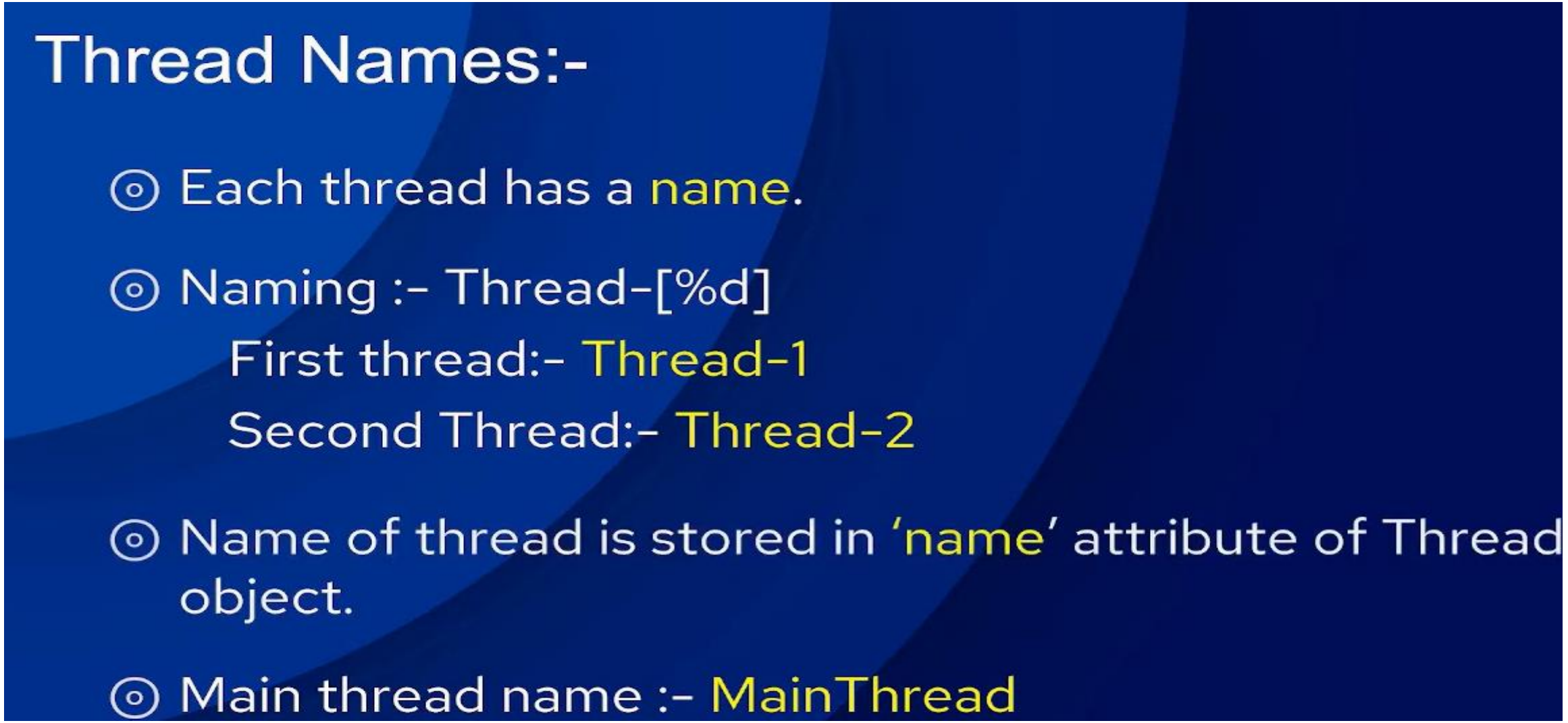
```
from threading import Thread

class MyThread(Thread):
    def run(self):
        #code

t1=MyThread()
t1.start()
```

Threading module:-

```
class Thread:
    def run(self):
        display()
```

Multithreading in Python

Multithreading in Python



Multithreading in Python

.

Multithreading in Python

.

Multithreading in Python

Thread Identifier(ident):

- Assigned by the Python interpreter.
- It's a unique integer used to identify a thread within the Python runtime.
- It has no direct relationship to the underlying operating system's thread identifier.
- Primarily used for internal Python threading management and is accessible using threading.get_ident() or threading.current_thread().ident.

Multithreading in Python

Native Identifier(native_id):

- Assigned by the operating system's kernel.
- It's an integer that uniquely identifies a thread at the system level.
- It can be used to identify the thread even outside of the Python process.
- In Python, you can access this using threading.current_thread().native_id (available since Python 3.8).

Multithreading in Python

Key difference between Thread & Native identifier:

- **Scope**: Thread identifiers are Python-specific, whereas native identifiers are system-wide.
- **Assignment:** Python interpreter assigns thread identifiers, while the operating system's kernel assigns native identifiers.
- **Purpose**: Thread identifiers are mainly used internally by Python's threading module for management. Native identifiers are used to uniquely identify threads at the operating system level.
- **Portability**: Thread identifiers might not be consistent across different Python implementations, while native identifiers are generally consistent within the same operating system.

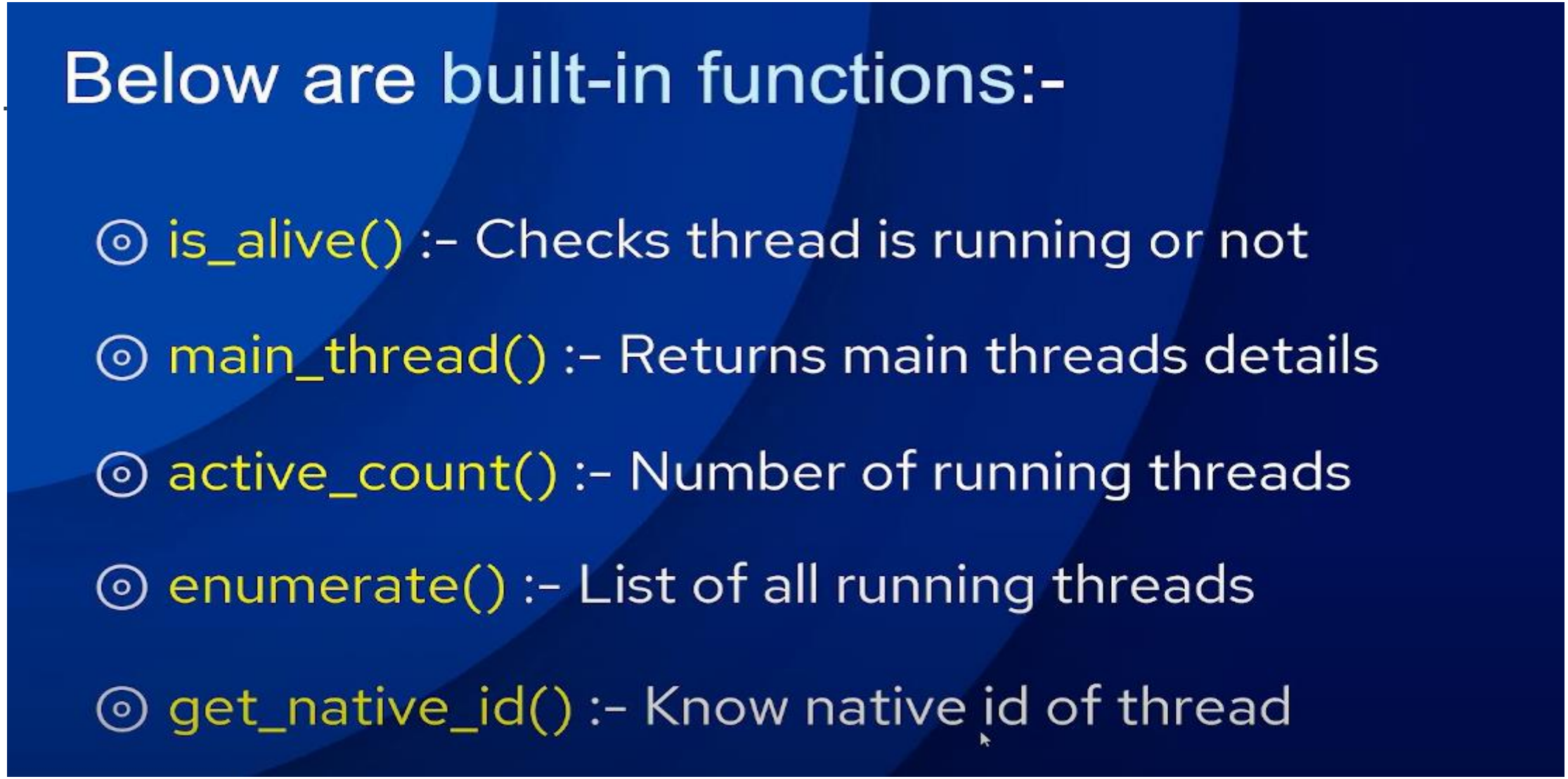
Multithreading in Python

Thread & Native identifier in simple terms:

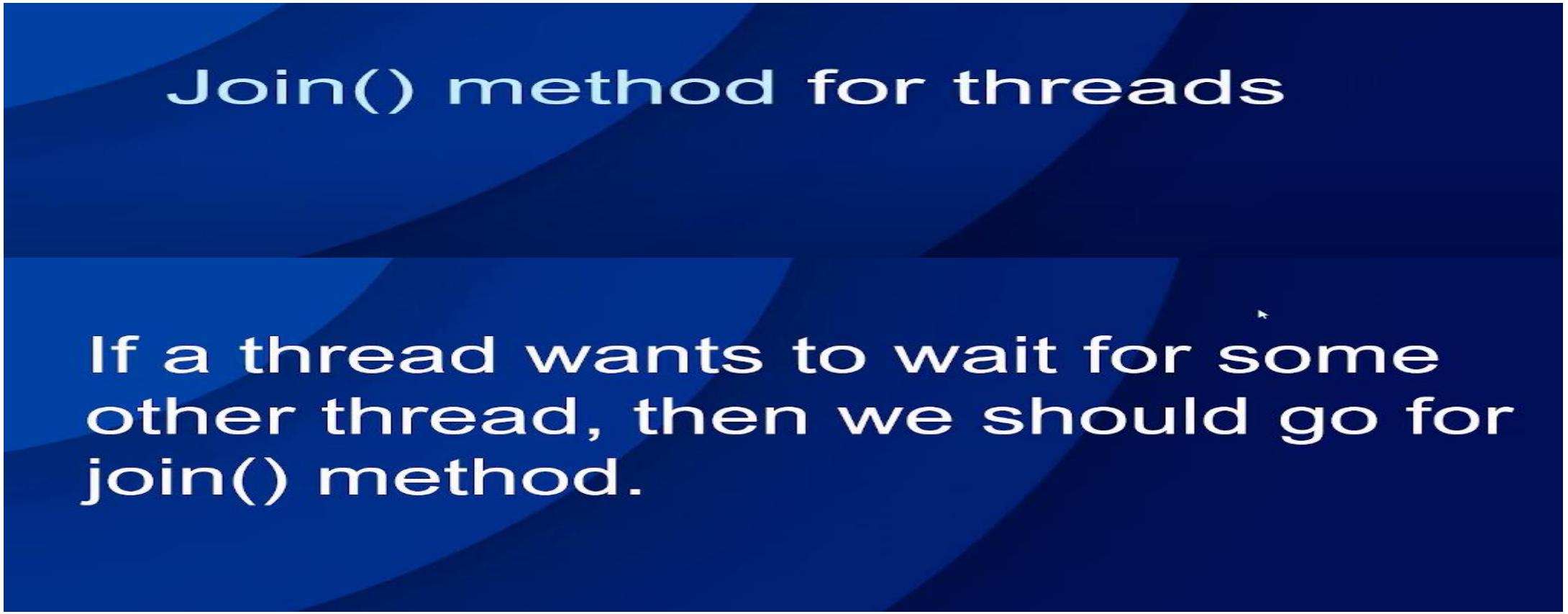
- Imagine you have employees within a company (Python process).
- Each employee has an employee ID (thread identifier) that's unique within the company.
- However, they also have a social security number (native identifier) that's unique across the entire country.

Multithreading in Python

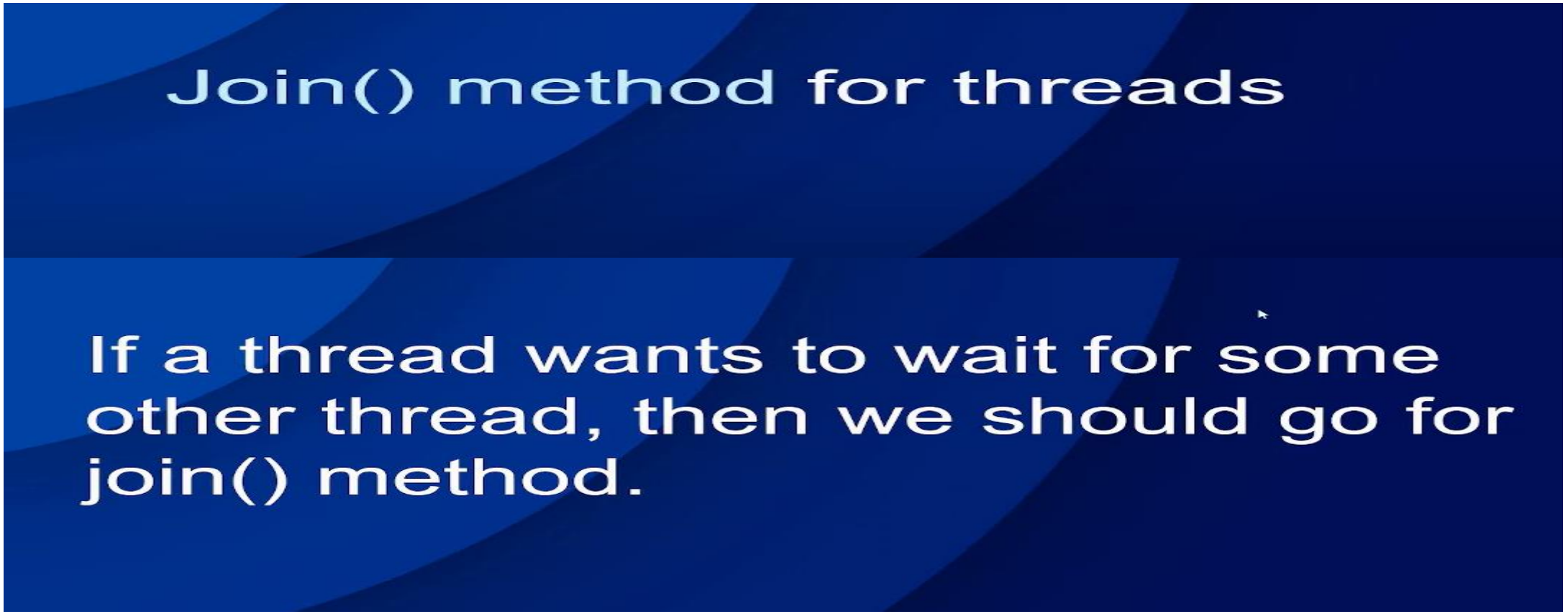
Multithreading in Python



Multithreading in Python



Multithreading in Python



Multithreading in Python

Join() Method:

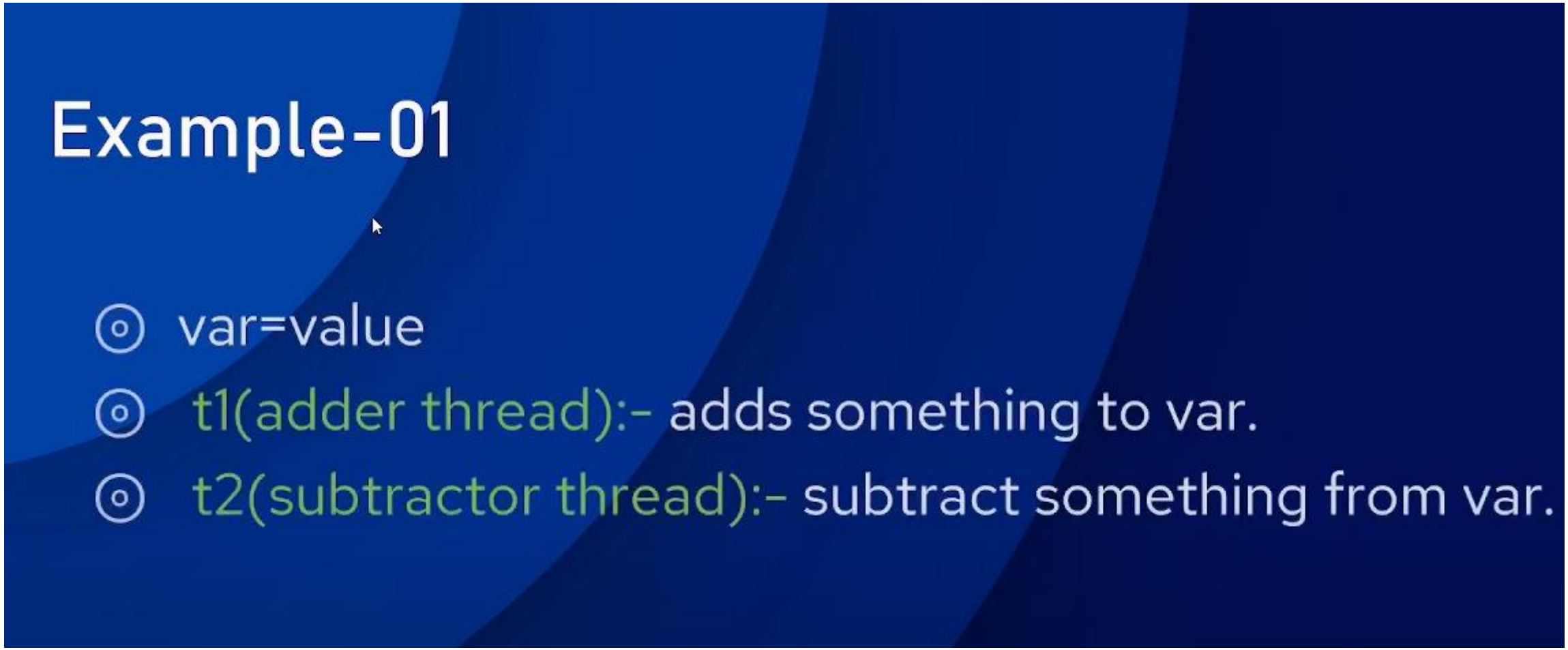
- The join() method is used to wait for a thread to complete its execution before proceeding further in the main thread.
- It essentially blocks the calling thread (usually the main thread) until the thread on which join() is called has finished.

Multithreading in Python

1. **Join() Method Steps:**
    1. **Import modules**: threading for threads and time for pausing execution.
    2. **Define a function**: my_function represents the thread's task.
    3. **Create and start a thread**: A thread is created and started, executing my_function.
    4. **Call join():** The join() method is called on the thread object. This blocks the main thread until the created thread completes its execution.
    5. **Main thread continues**: After the created thread finishes, the main thread resumes and prints a message..

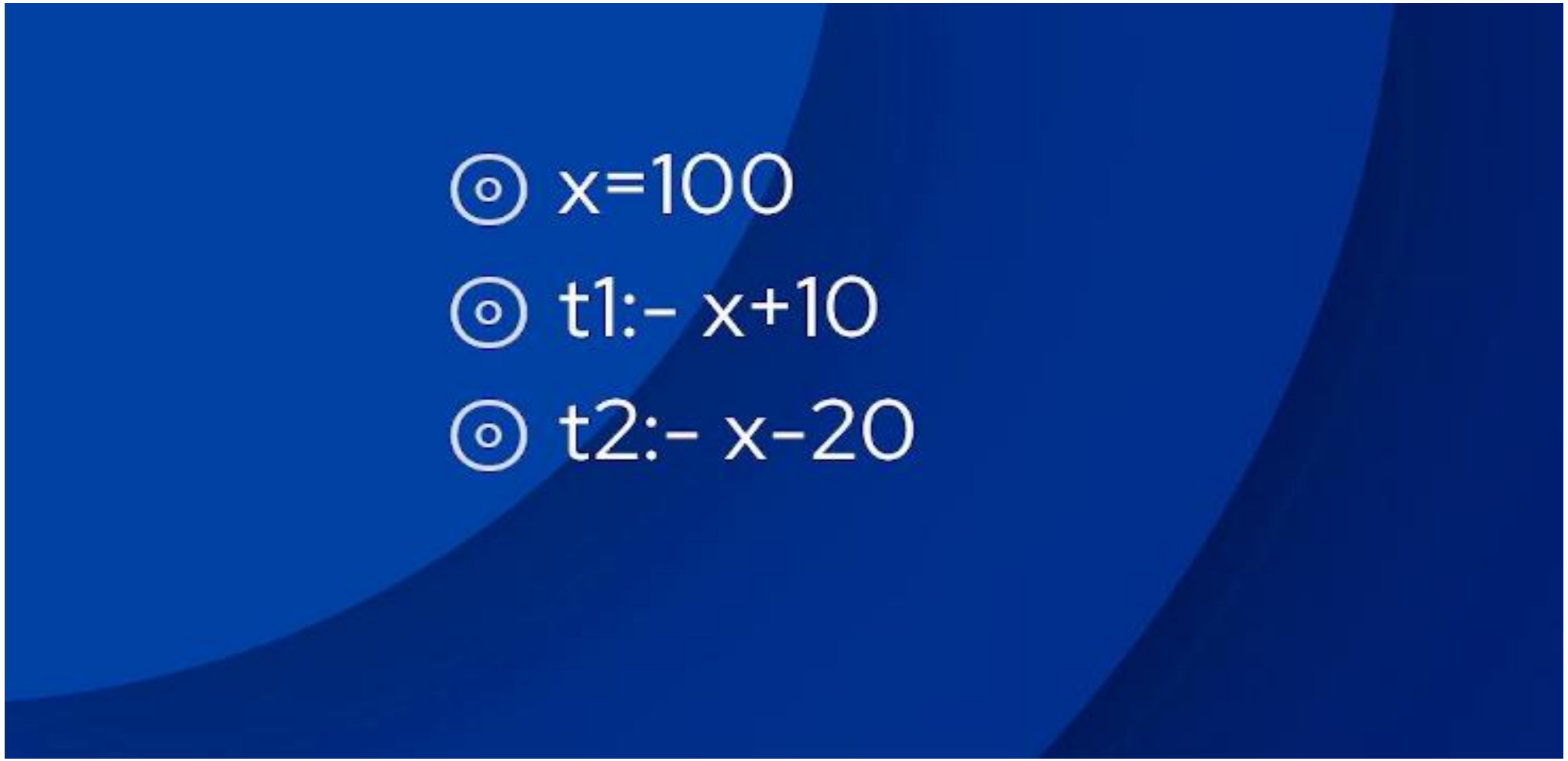
Multithreading in Python

Multithreading in Python

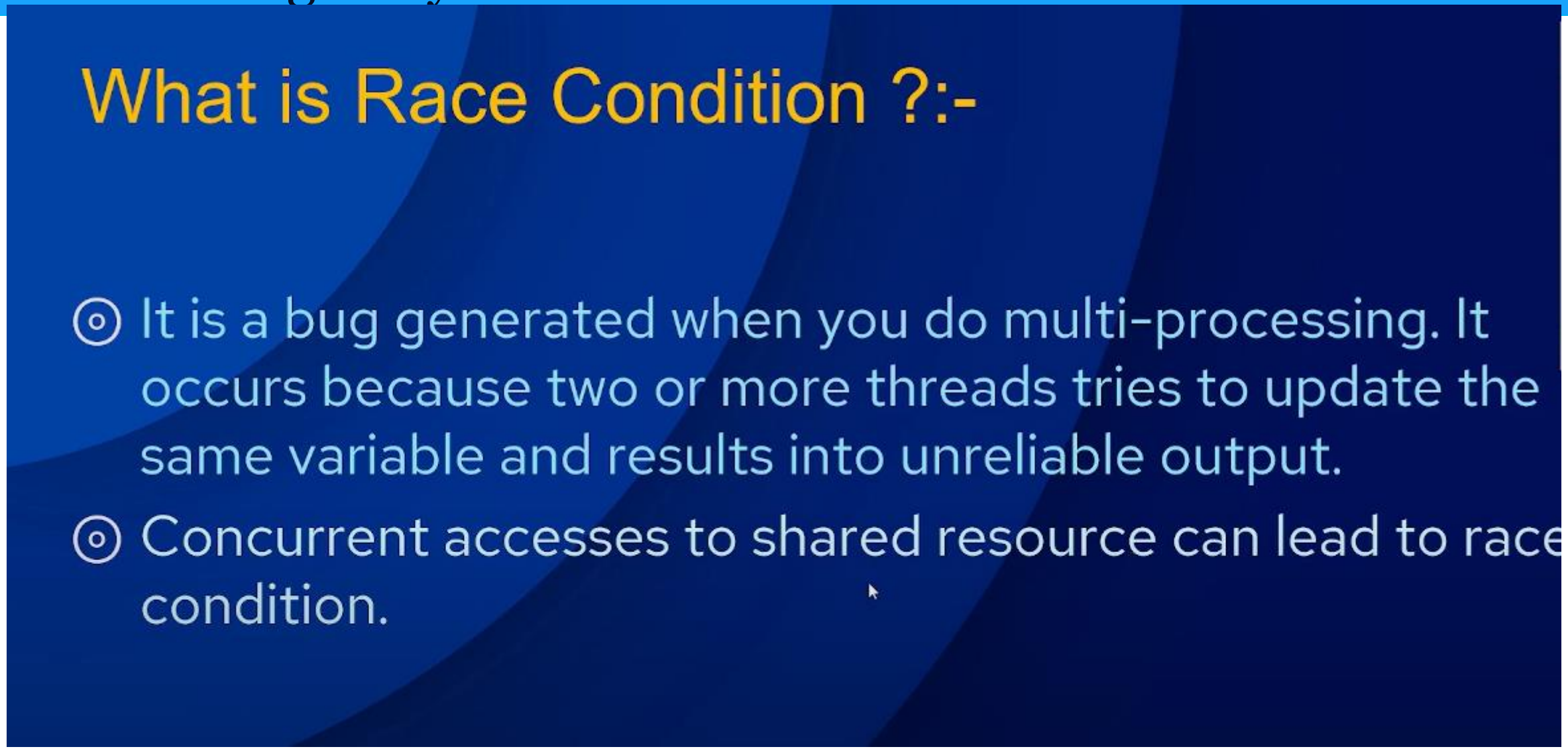


Multithreading in Python

Multithreading in Python



Multithreading in Python



Multithreading in Python

Thread synchronization Technique :-

- Thread synchronization is defined as a mechanism which ensures that two or more concurrent threads do not simultaneously execute some particular program segment known as critical section.

- We have following thread synchronization techniques.
  - i) Using Locks.
  - ii) Using R-Lock
  - iii) Using Semaphores.

Multithreading in Python

Locks in Python:-

- threading module provides a Lock class to deal with the race conditions.

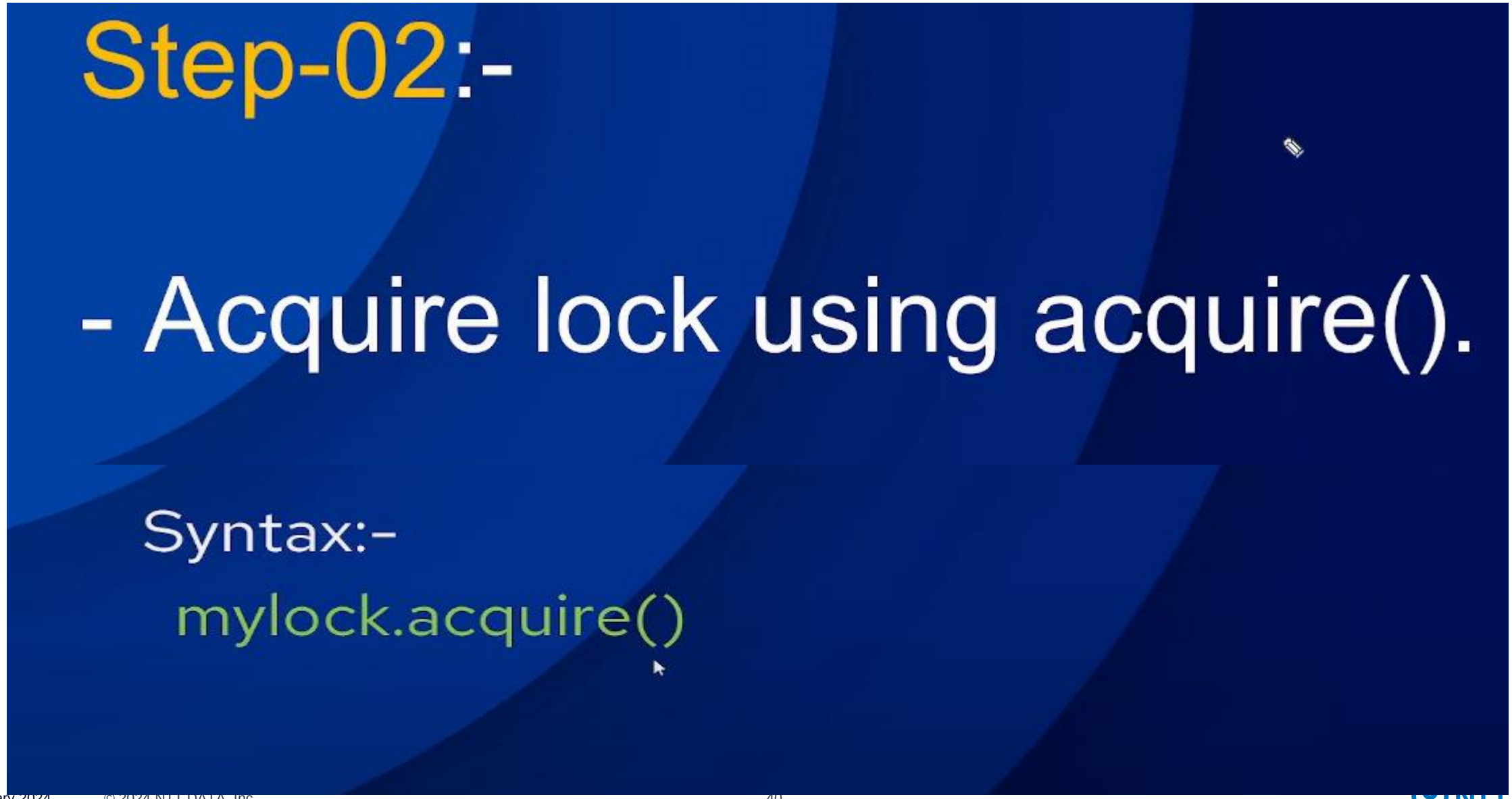
Lock has two states:-

1) Locked :- The lock has been acquired by one thread and any thread that makes an attempt to acquire it must wait until it is released.

2) Unlocked:- The lock has not been acquired and can be acquired by the next thread that makes an attempt.

Multithreading in Python

Multithreading in Python



Step-03:-

- Release lock using release().

Syntax:-

```
mylock.release()
```

Multithreading in Python

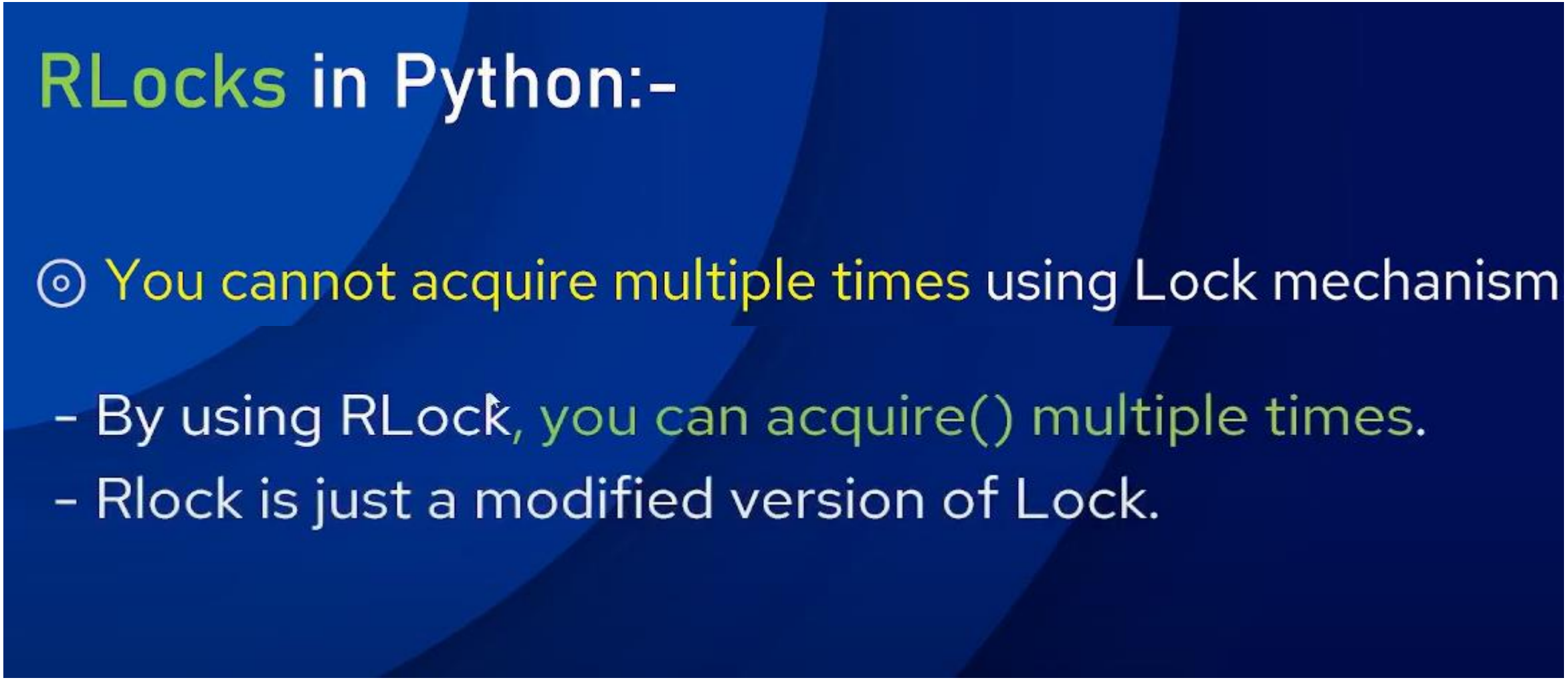
acquire() method

- change the state of code to locked.
- other threads have to wait until lock is released by curren working thread.

syntax:-

```
lock_object.acquire([blocking=True],timeout=-1)
```

Multithreading in Python



Multithreading in Python

Necessity of Rlock:

In Python's threading module, both Lock and RLock are used for synchronization between threads. However, they differ in their behavior and usage.

Lock: A Lock can be acquired only once by a thread. If the same thread tries to acquire the lock again before releasing it, it will result in a deadlock. This is because the thread will be blocked waiting for itself to release the lock.

RLock: An RLock is a type of lock that allows a thread to acquire the lock multiple times before releasing it. This is useful in scenarios where a function needs to call itself recursively or where a thread needs to acquire the lock multiple times within a block of code.

Multithreading in Python

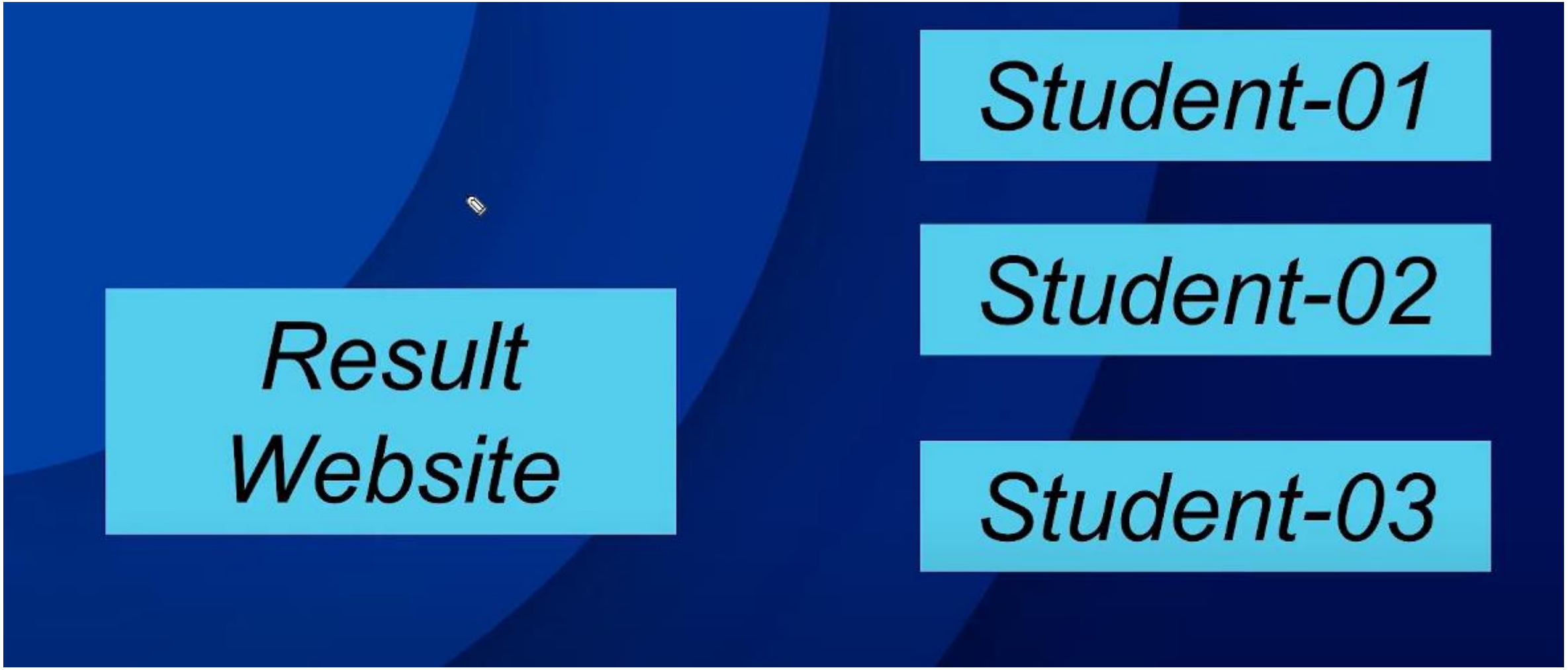
Multithreading in Python

Multithreading in Python

Multithreading in Python

- In Lock and RLock, at a time only one Thread is allowed to execute.
- but sometimes our requirement is to execute a particular number of Threads at a time.

Multithreading in Python



Multithreading in Python

.

Multithreading in Python

Multithreading in Python

Multithreading in Python