Friday, 22 March 2024

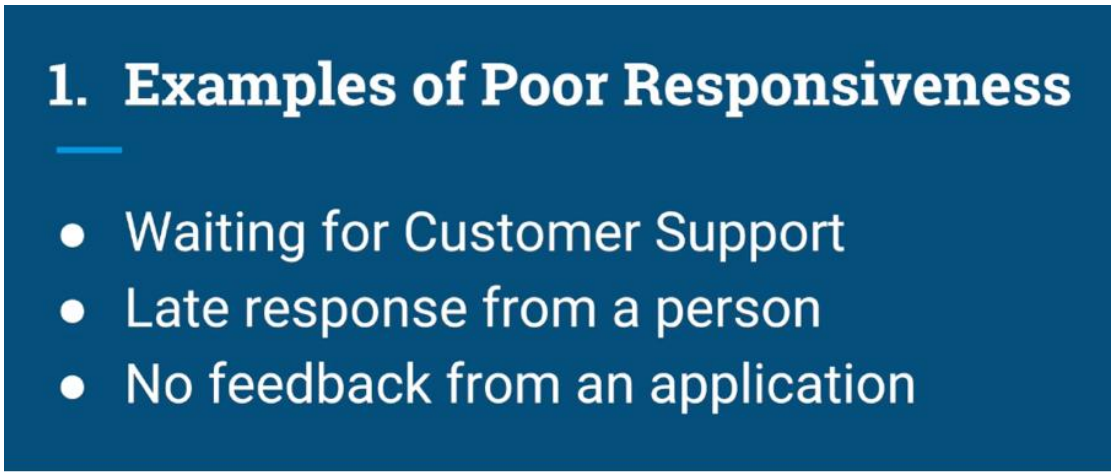
Multithreading, Concurrency & Parallel programming

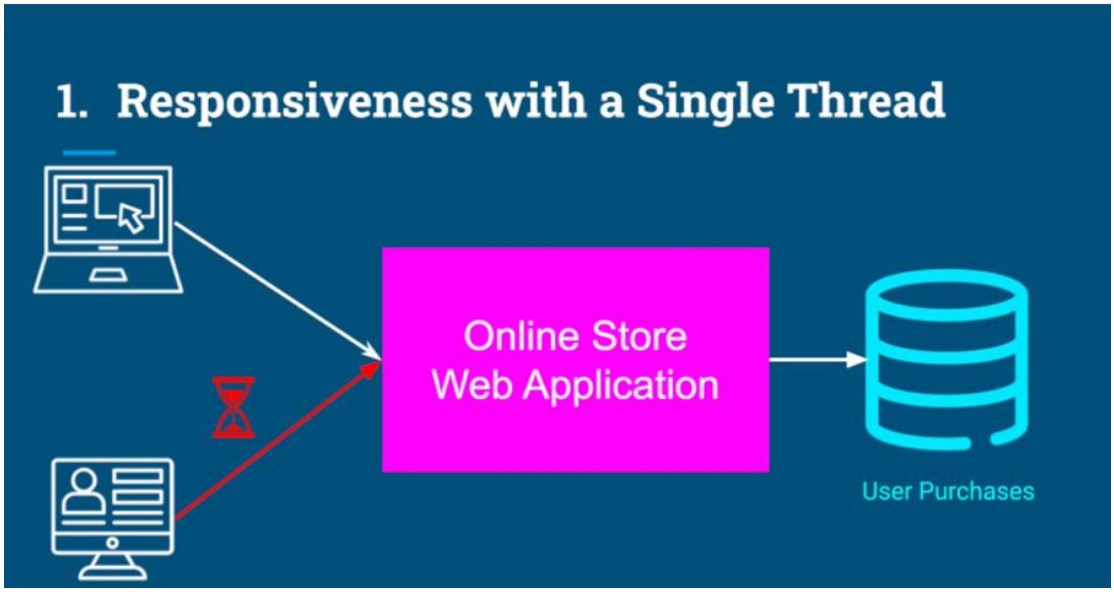
Why we even need multiple, both threads.

The two main reasons we need threads for -

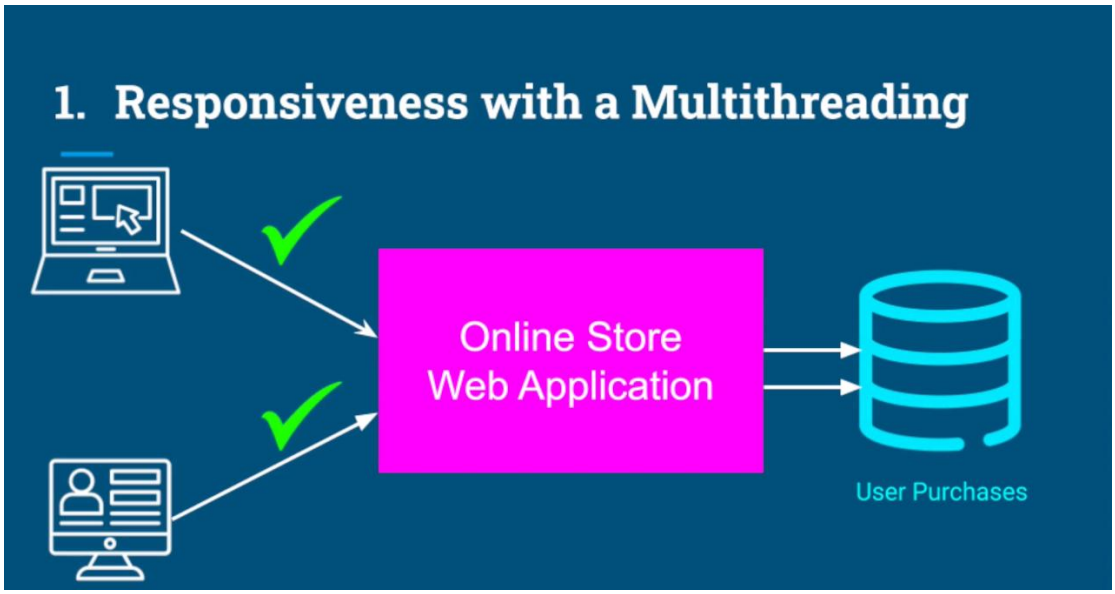
- Responsiveness
- Performance

Responsivness

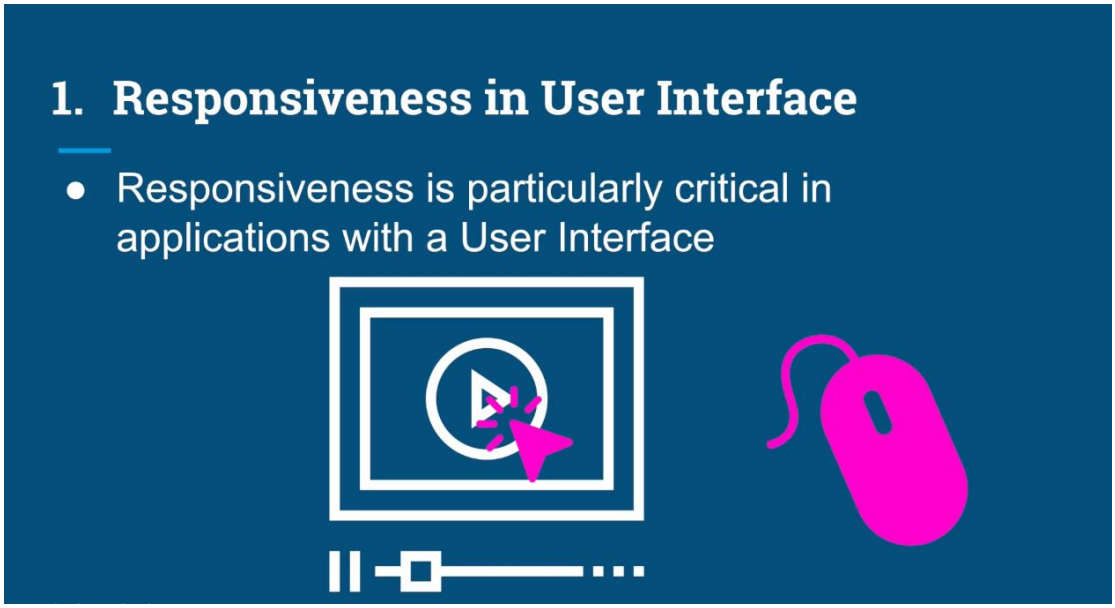




With multithreading we could actually serve multiple users simultaneously, but serving each request on a different thread.



Responsiveness is particularly critical when it comes to applications with a user interface. A good example for that can be a movie player application. The application is showing us images, blamed the audio.
And in the same time, we expect that if we move the mouse or click a button, we would get an instant feedback for our actions on the screen.

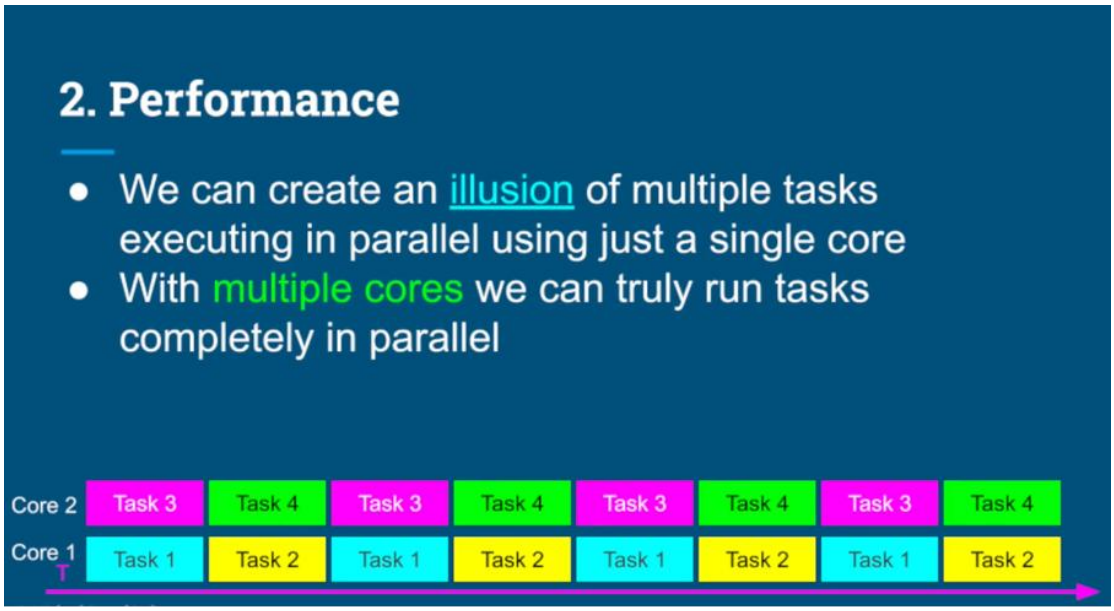


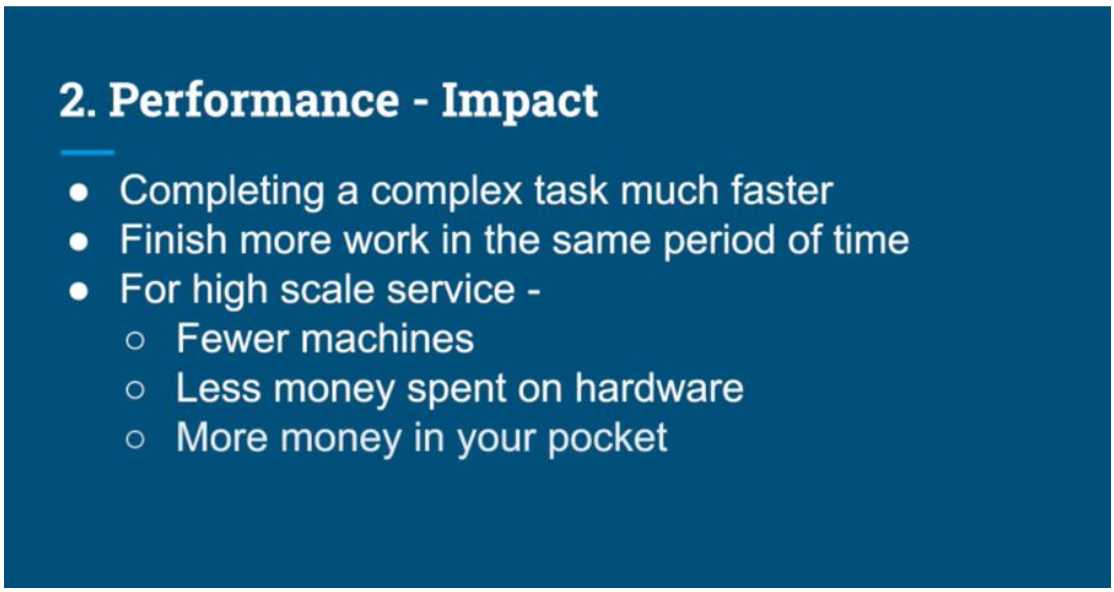
This kind of responsiveness can be achieved by using multiple threads, each thread for a different task.
By multitasking quickly between different threads, Our computer can create an illusion that all those tasks are actually happening in the same time.
The term we use for this kind of multitasking is concurrency.

we don't even need multiple cores to achieve concurrency. Even with one core, we can create responsive applications by using multiple threads.

Performance

As mentioned before using concurrency, we can create an illusion of multiple tasks running in parallel just with single core. With multiple core we can truly run multiple tasks completely in parallel.

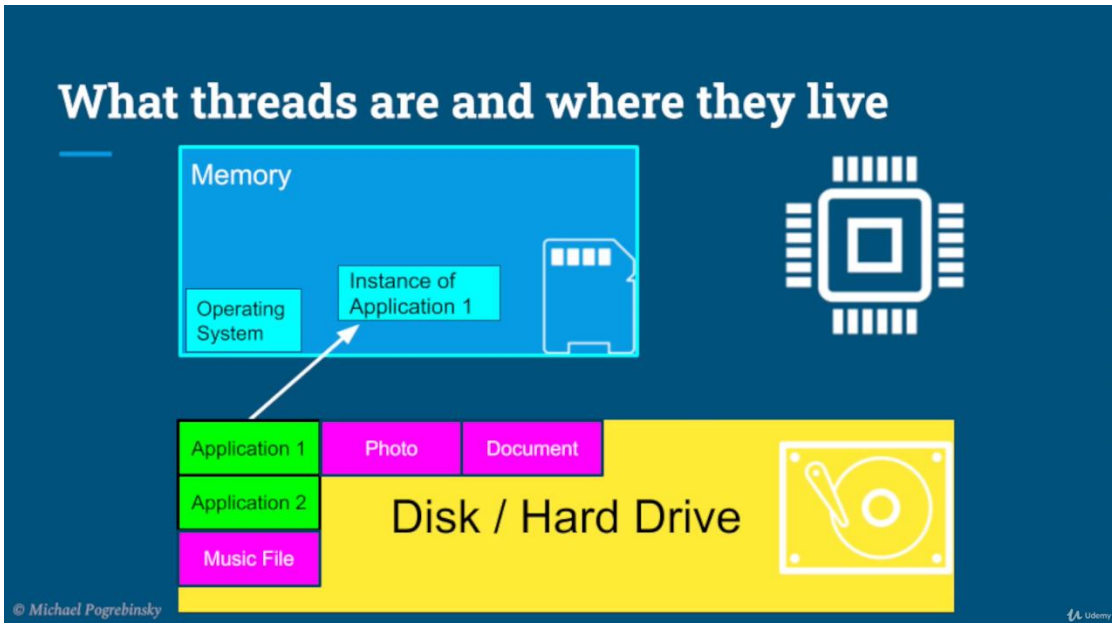




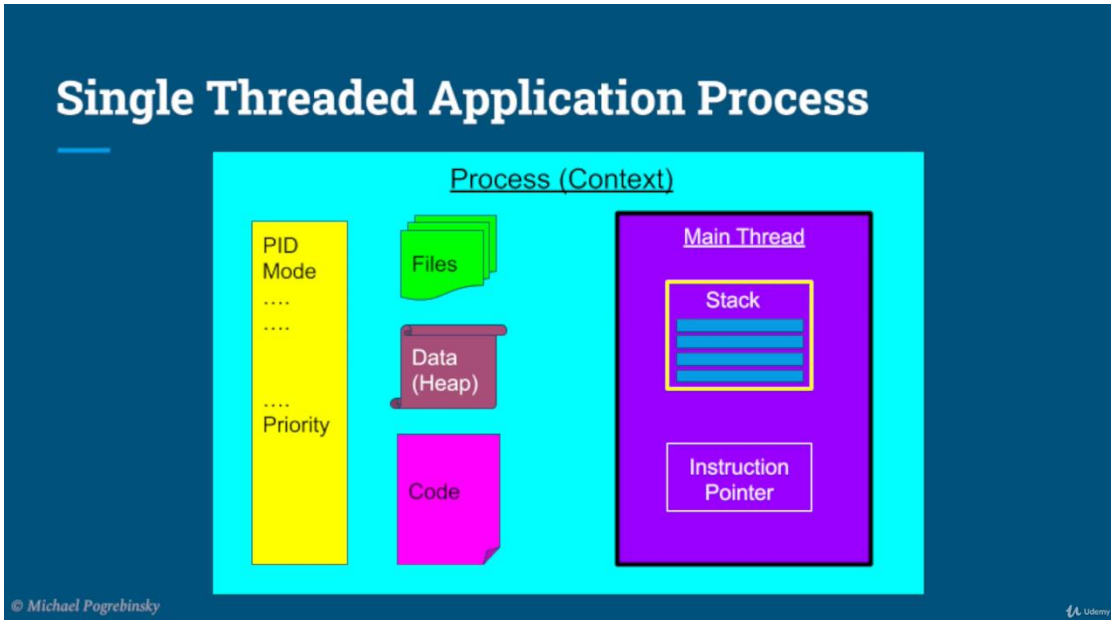
Responsiveness - Concurrency
Performance - Parallelism

What threads are and where they live

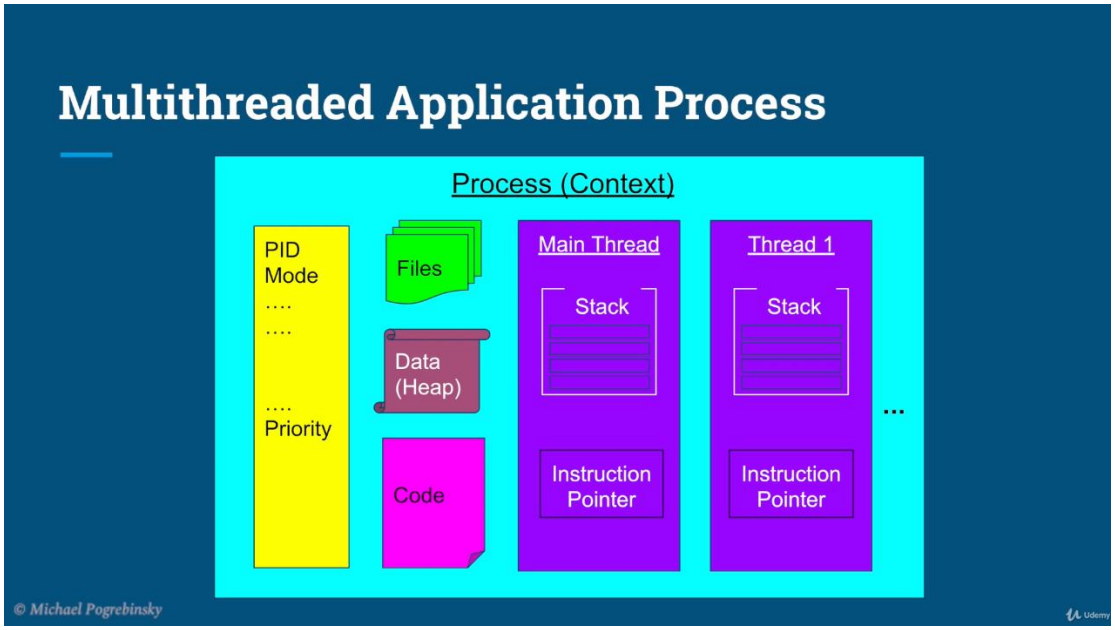
When we turn on our computer a special program called the operating system is loaded from the disc into the memory.
The operating system takes over and provides an obstruction for us.
The application developers, and helps us interact with the hardware and the CPU. So we can focus on developing our apps.
All our applications, such as the text editor, a web browser, or a music player reside on a disc in a form of a file, just like any Other music file image or document when they user runs an application, the operating system takes the program from the disc and creates an instance of that application in the memory.



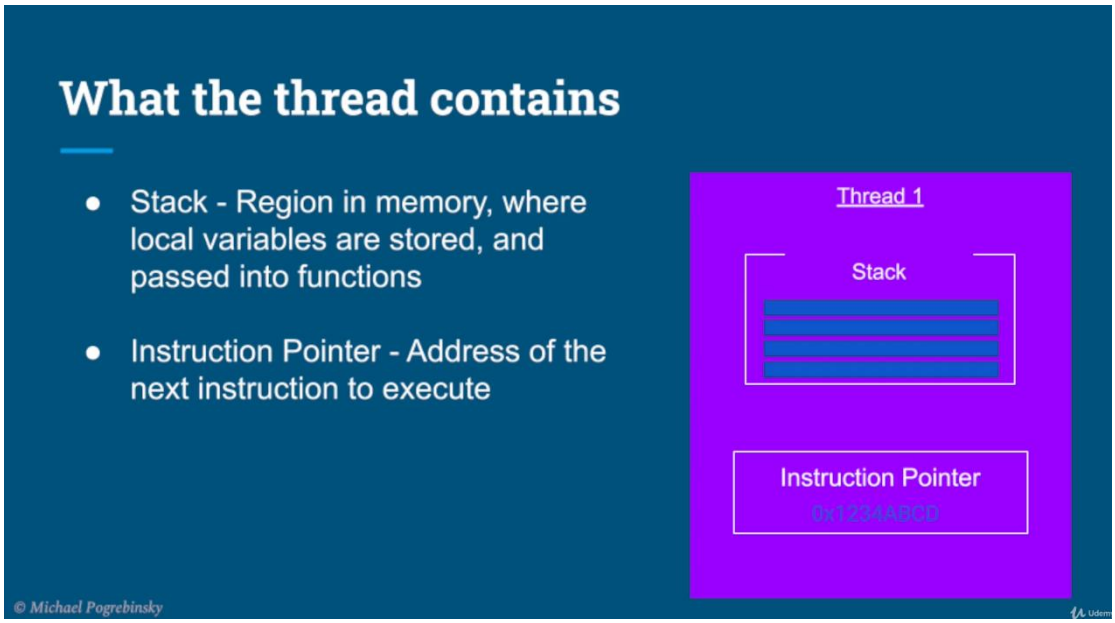
That instance is called a process, and it's also sometimes called
a context of an application.
Each process is completely isolated from any other process that runs on the system.
A few of the things that the process contains are the metadata, like
the process ID, the files that the application opens for reading and
writing the code, which is the program instructions that are going to be executed
on the CPU, the heap, which contains all the data, our application needs.
And finally, at least one thread called the main thread



The thread contains two main things, the stack and the instruction pointer.
In a multithreaded application, each thread comes with its own stack and its own instruction pointer, but all the rest of the components in the process are shared by all threads.



The stack is a region in memory where the local variables are stored and functions are executed.
And the instruction pointer is nothing more than a pointer that points to the address of the next instruction that the thread is going to execute.



Summary

- Motivation for multithreading
  - Responsiveness achieved by *concurrency*
  - Performance achieved by parallelism
- Threads are and what they contain
  - Stack
  - Instruction pointer
- What threads share
  - Files
  - Heap
  - Code