11a. Overview and Goals

INTRODUCTION

Programming languages typically execute code sequentially, following a single path of execution that utilizes one core at a time. This linear approach simplifies reasoning about program behavior, as each operation completes before the next begins. However, hardware these days is commonly equipped with multiple processor cores. Consequently, a sequential execution does all the work on one core, while the others sit idle. This leaves substantial computational power untapped.

Multithreading addresses this limitation by running different segments of our program simultaneously across multiple cores. While this capability opens up significant opportunities for performance improvement, it also introduces new challenges that developers need to navigate carefully. In fact, simple operations that work flawlessly in single-threaded programs may yield incorrect results in a multithreaded setting. Furthermore, writing multithreaded code requires a fundamental shift in the user's mindset regarding program execution. All this makes multithreaded code inherently more difficult to write, test, and debug than its single-threaded counterpart.

Despite these challenges, the potential performance benefits of multithreading make it an essential tool in modern programming. This is particularly true for applications that are computationally intensive or demand that the same code be applied to multiple objects.