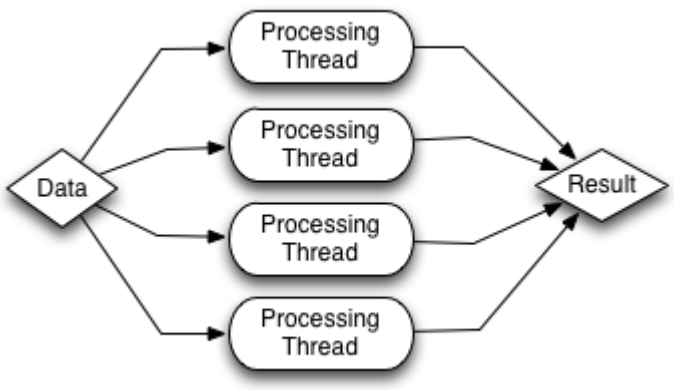
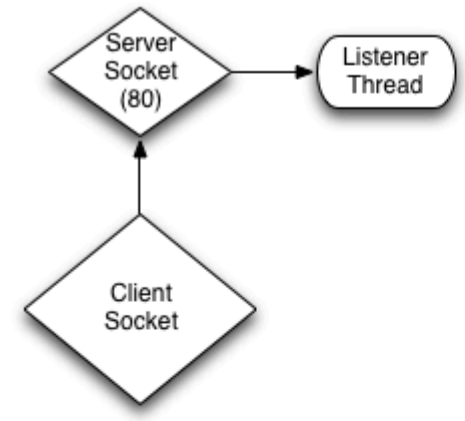
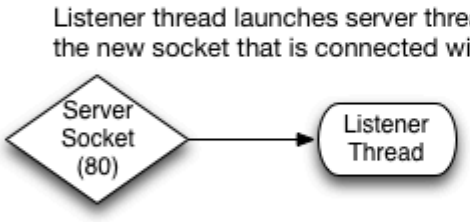
Threads-and-locks in general
threads - rust - main
- Concurrent vs parallel execution
- Languages - concurrent programming, threading and multiprocessing
- Thread Termination - in general
- Bypassing the GIL for Parallel Processing in Python
- Thread pool - general - task queue
- threads - general - thread safety
- Multithreading - java

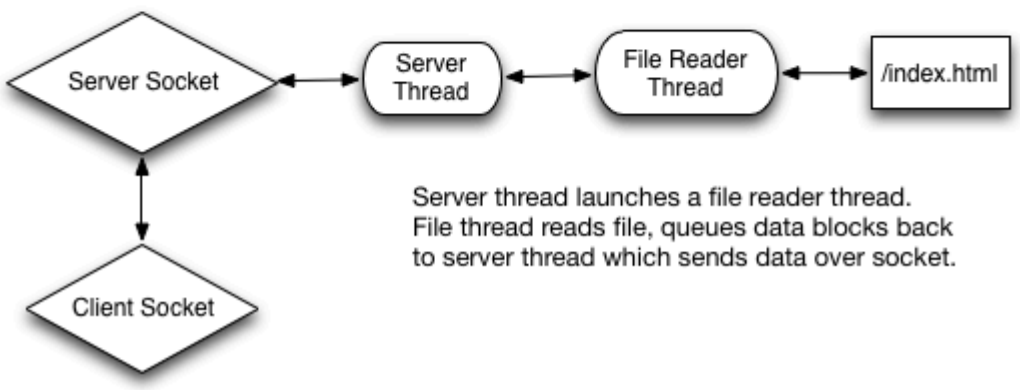
rust:
- Concurrency rust main
- threads - rust - main

videos:
- Great video: Functional Programming in Python: Parallel Processing with "concurrent.futures"




## Threads-and-locks

Threads-and-locks programming is like a Ford Model T. It will get you from point A to point B, but it is primitive, difficult to drive, and both unreliable and dangerous compared to newer technology.
Despite their well-known problems, threads and locks are still the default choice for writing much concurrent software, and they underpin many of the other technologies we'll be covering. Even if you don't plan to use them directly, you should understand how they work.

## Threads of Execution

A thread of execution—or thread, for short—represents the execution of a series of instructions. Programs often start with a single thread and terminate when this thread is finished.
More precisely, user code is executed within a single thread of execution.
Typically, a JVM uses additional threads for garbage collection,
just-in-time compilation, and other operations.