Threads and processes are equivalent from the perspective of the Linux kernel, thus there is no OS-level preference for one or the other. Given the availability of sharing memory between processes, the fundamental question of threads versus processes for parallelism boils down to implicit versus explicit sharing of memory. The private data structures that are typically desired for coarse-granularity parallelism are provided automatically with processes, but can only be achieved by discipline when using threads. Meanwhile, data and synchronization using shared memory has identical performance semantics with threads and processes (modulo TLB sharing between hyperthreads). In this talk, we investigate the nature of essential synchronization with threads vs processes, consequences for programming models, performance gotchas, and debuggability. We present experiments in the context of multilevel PDE solvers with a focus on versatility from the latency-limited strong scaling regime through to large memory-limited cases.