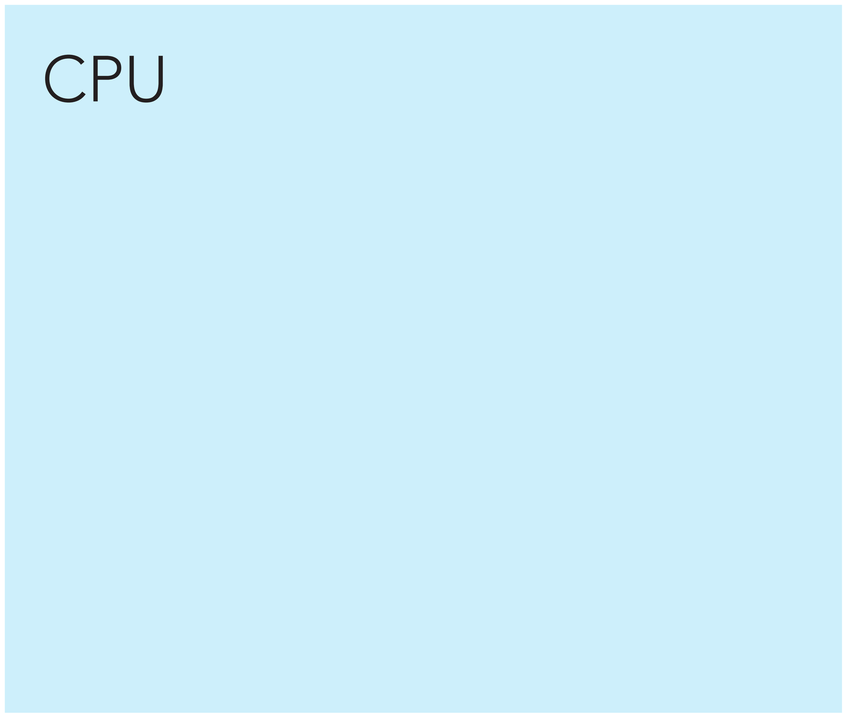
# GPU Programming

# Multi-threading

Christian Lessig

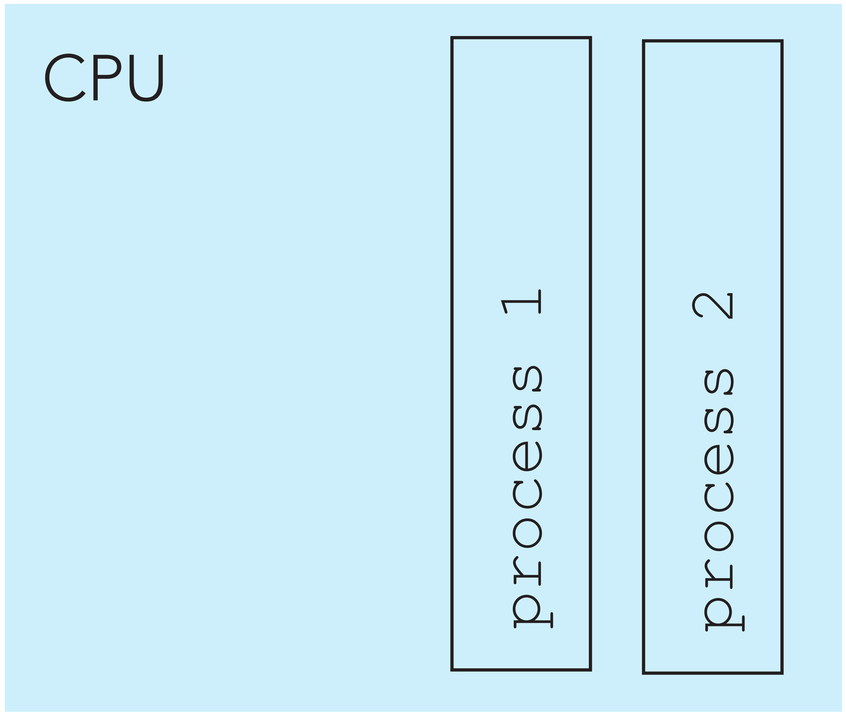
# Multiple Processes

- How to execute two tasks A and B on a MIMD?



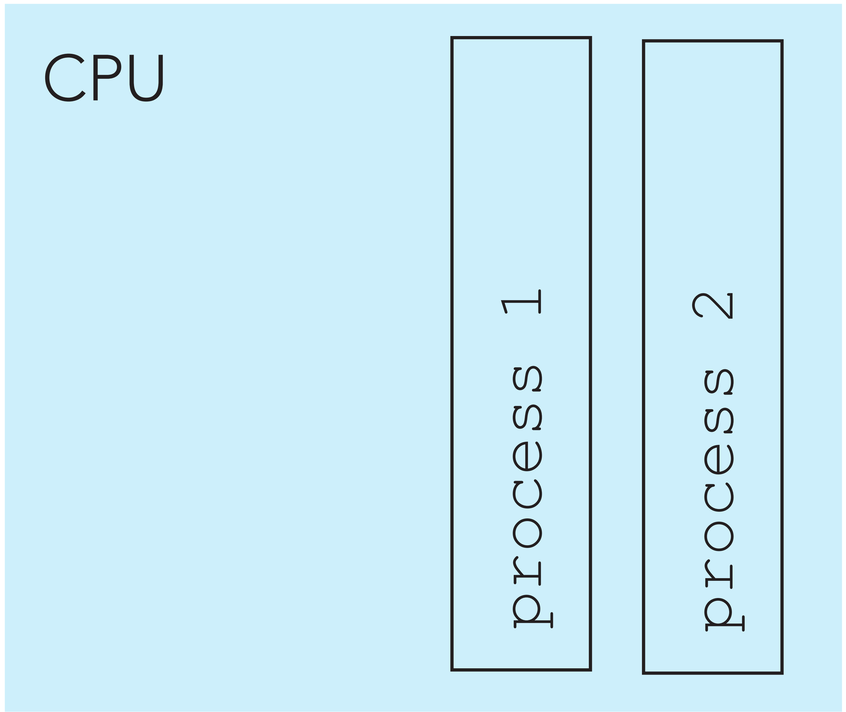
# Multiple Processes

- How to execute two tasks A and B on a MIMD?



# Multiple Processes

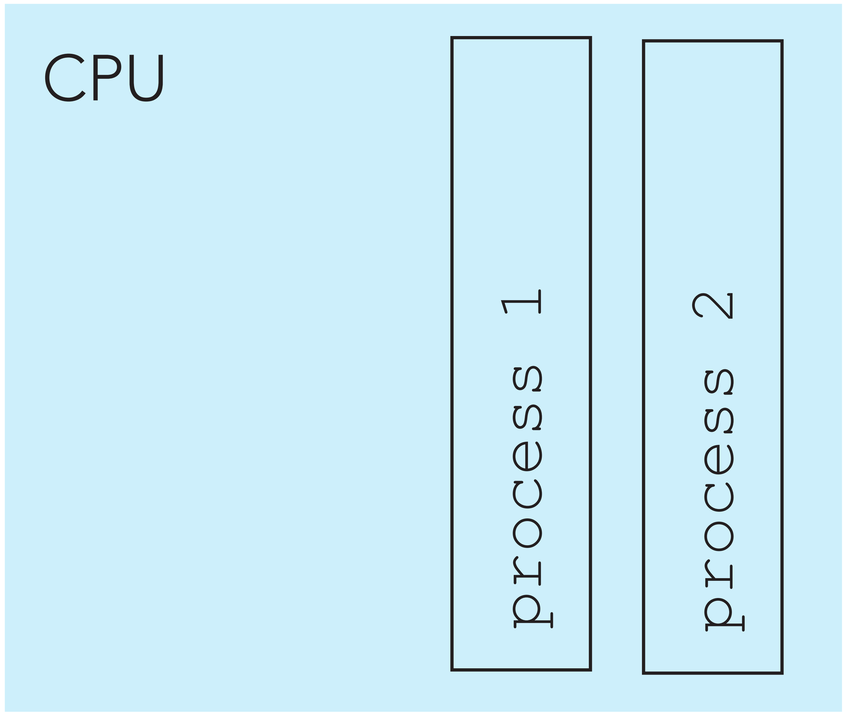
- How to execute two tasks A and B on a MIMD?




- Independent processes
- Communication via inter-process communication

# Multiple Processes

- How to execute two tasks A and B on a MIMD?

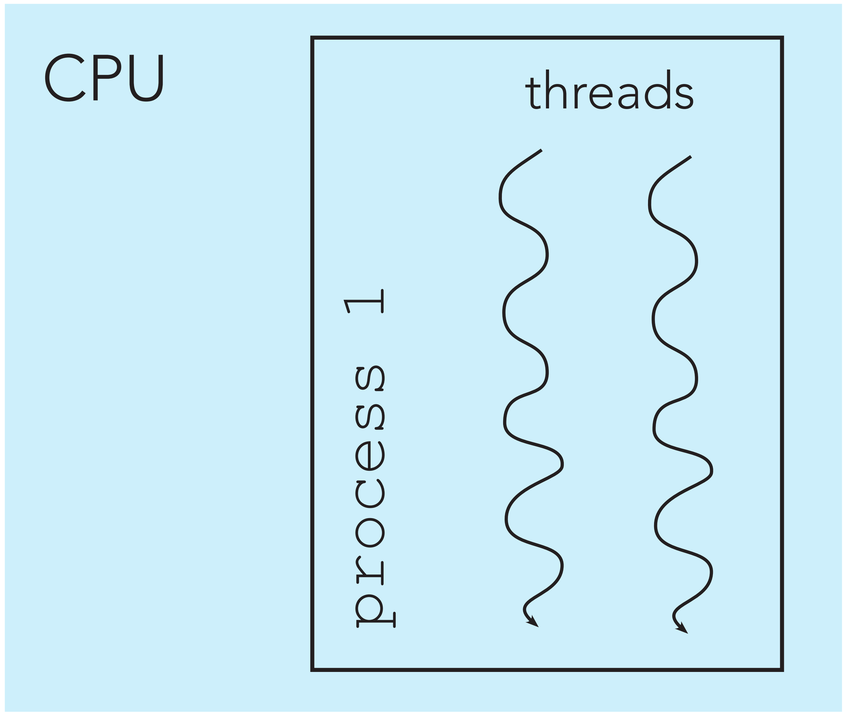



- Independent processes
- Communication via inter-process communication
- Extends directly to a cluster of computers (via MPI)

Communication is expensive and cumbersome

# Multi-threading

- How to execute two tasks A and B on a MIMD?



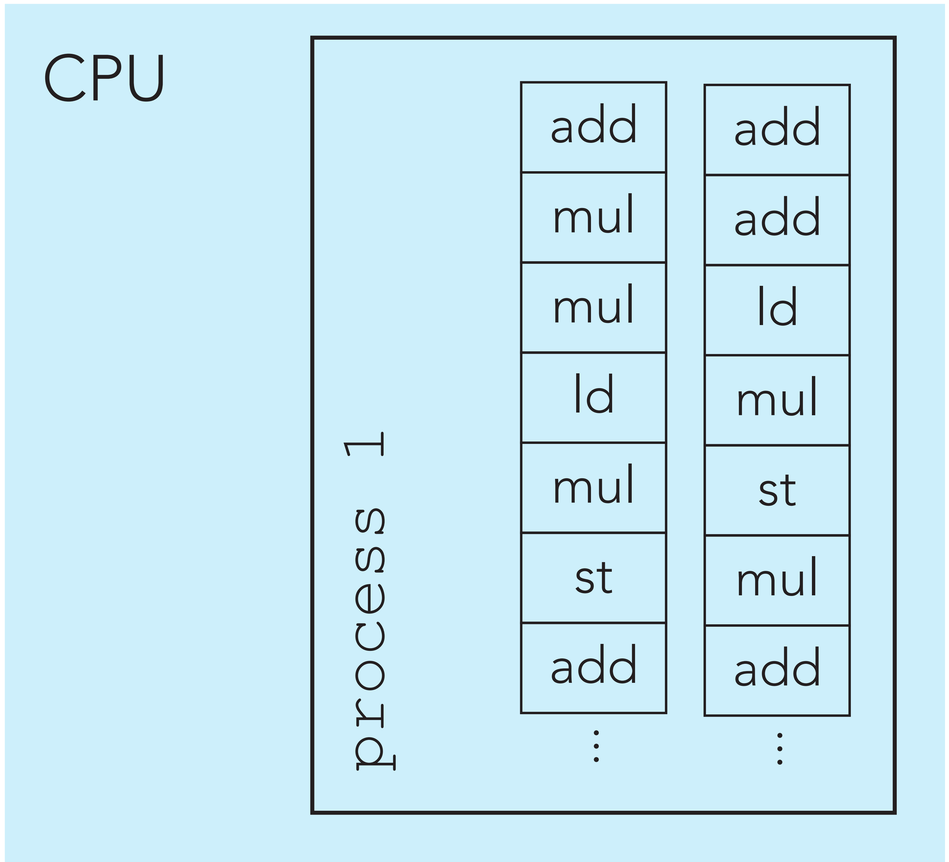

- One process with two threads
- Threads share memory and other process resources
- Per thread program counter (and hence scheduling)

# Multi-threading

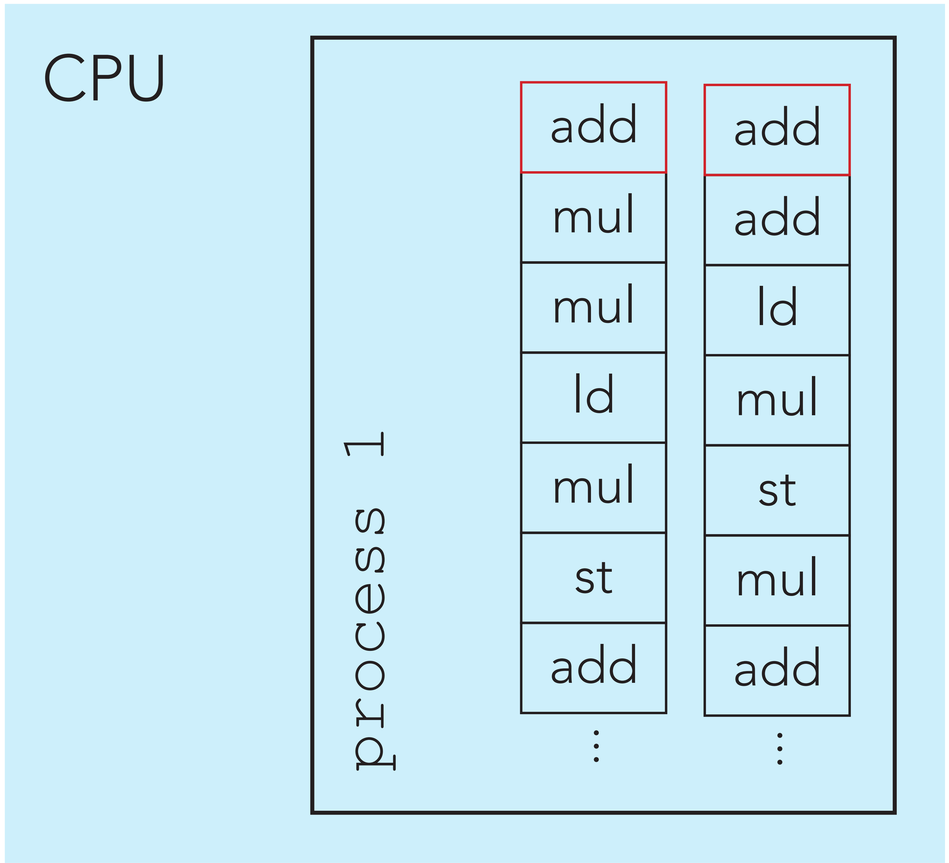
"Although threads seem to be a small step from sequential computation, in fact, they represent a huge step. They discard the most essential and appealing properties of sequential computation: understandability, predictability, and determinism. Threads, as a model of computation, are wildly non-deterministic, and the job of the programmer becomes one of pruning that nondeterminism."

E. A. Lee, "The Problem with Threads," Computer, vol. 39, no. 5, pp. 33–42, May 2006.

# Multi-threading

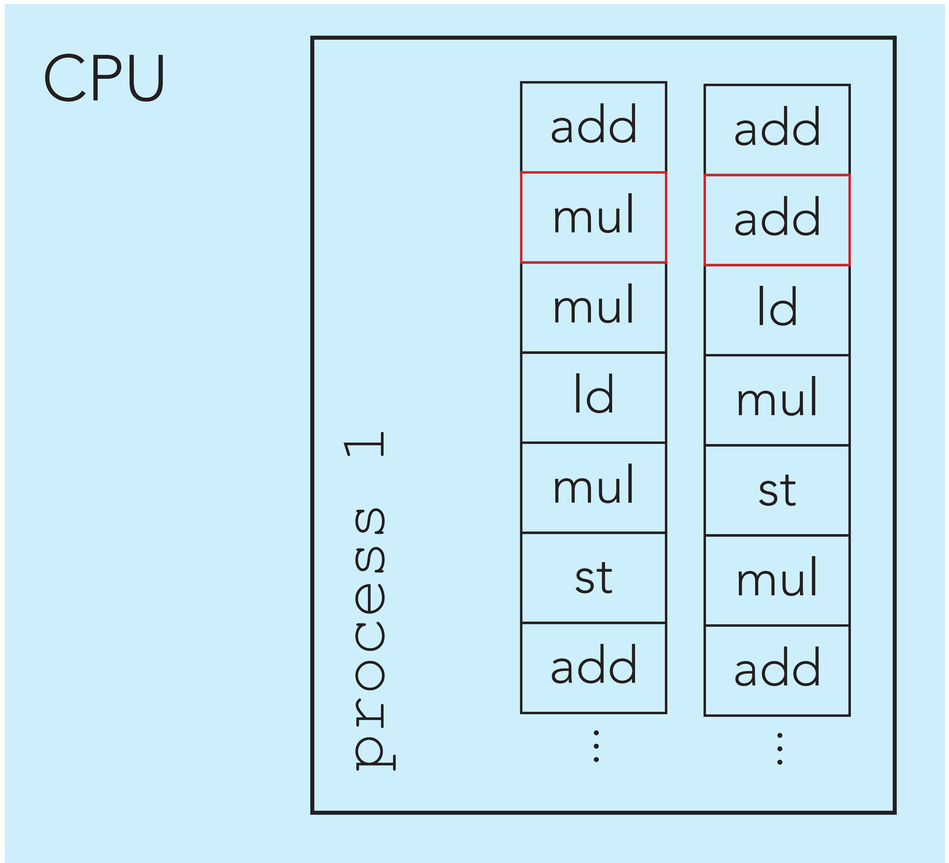


# Multi-threading



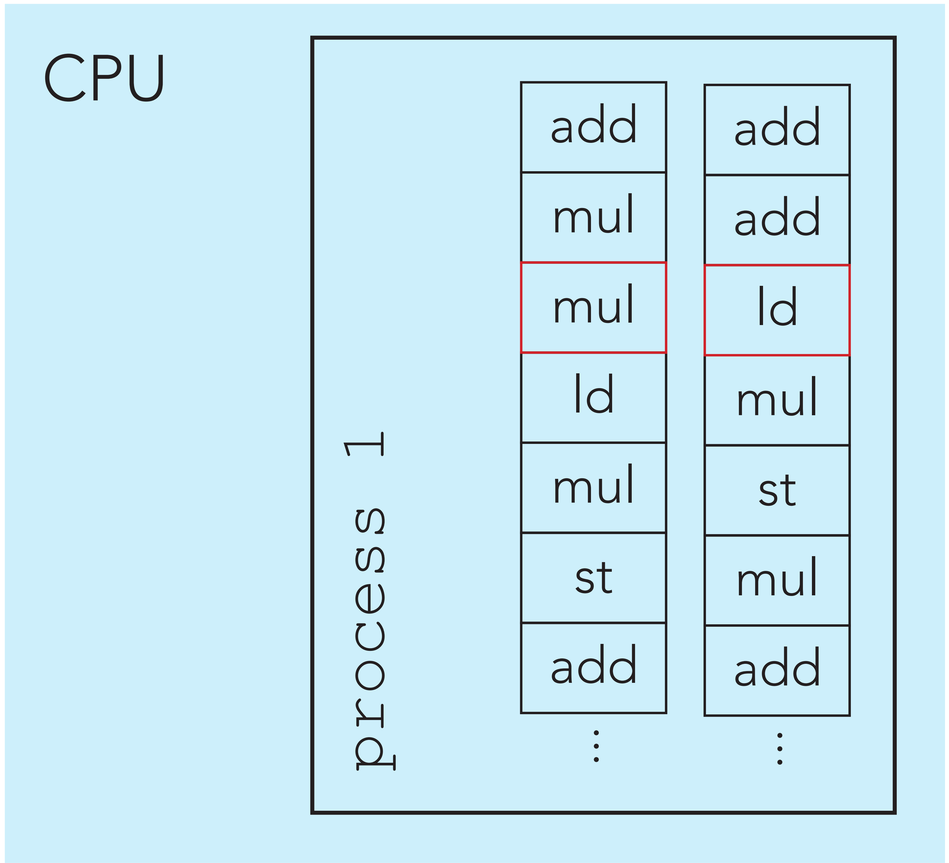

t = 0

# Multi-threading



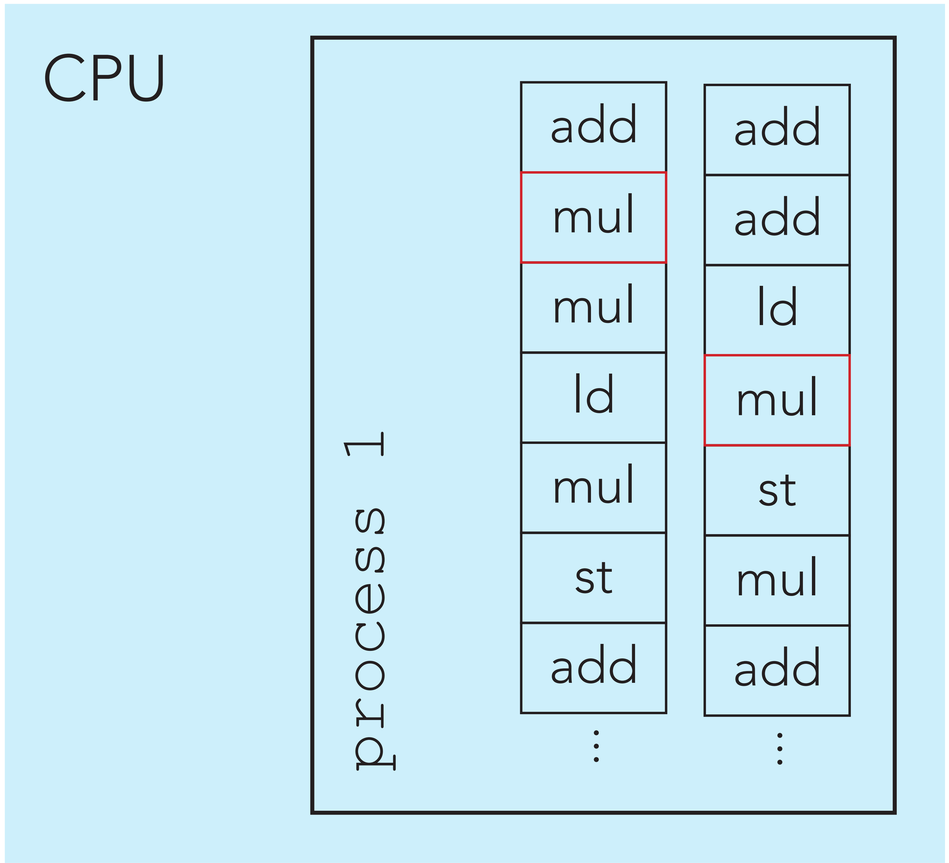

t = 1

# Multi-threading



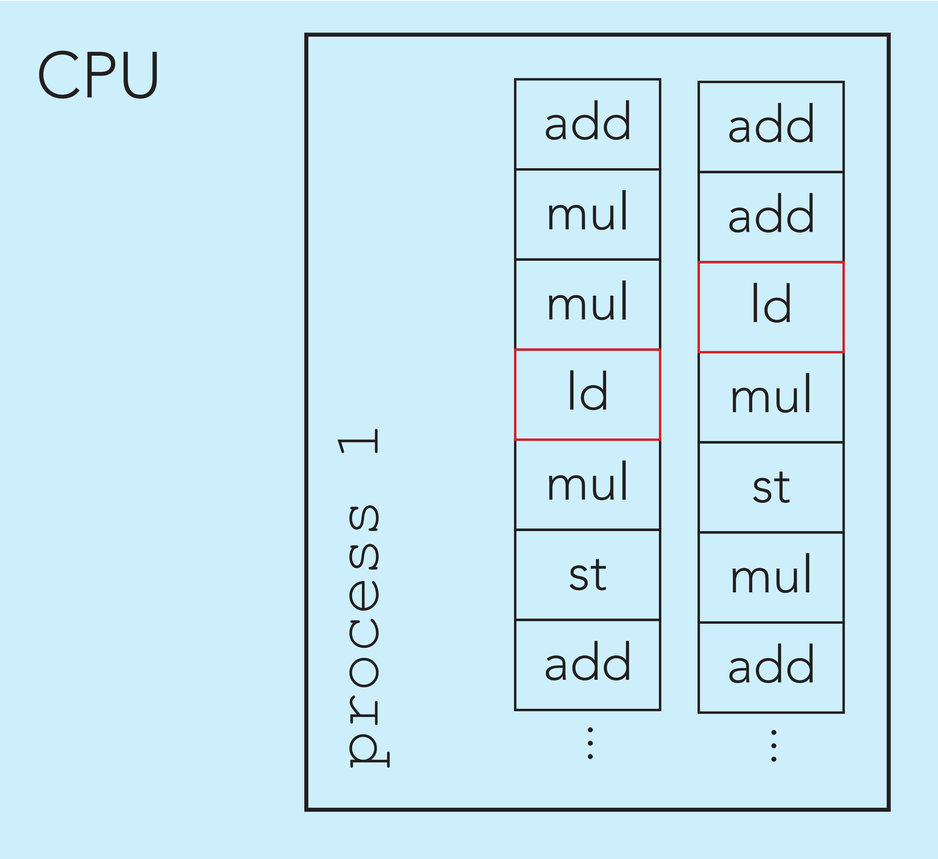

t = 2

# Multi-threading




t = 2

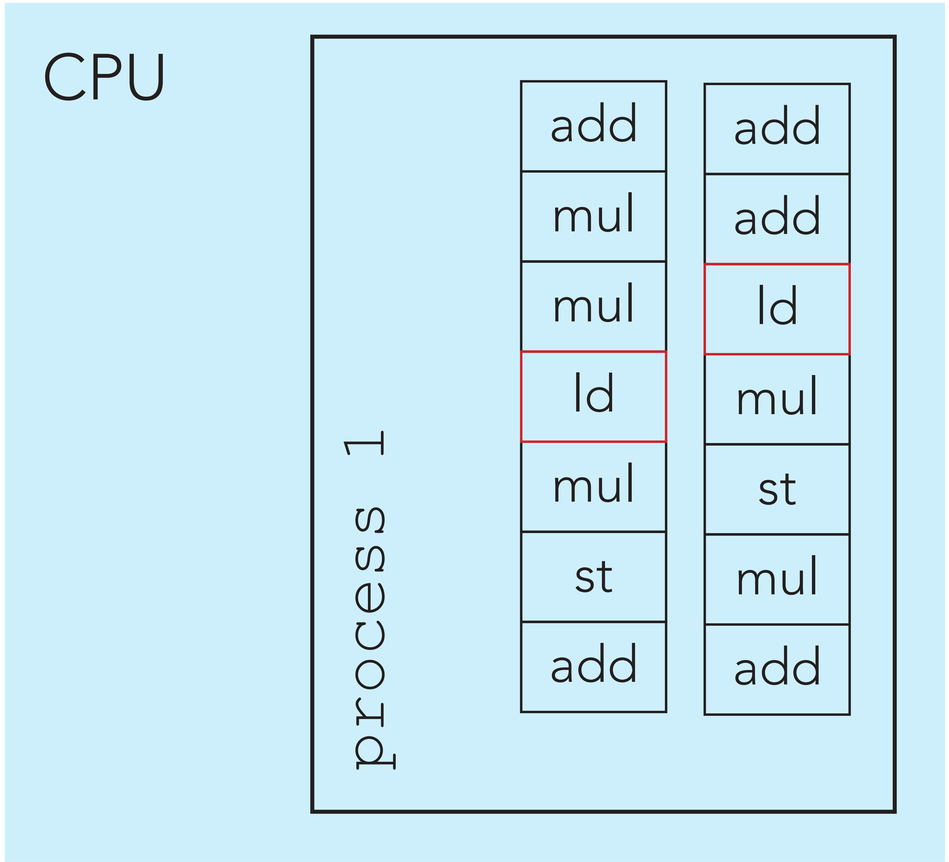
# Multi-threading




t = 2

Interleaving of threads is effectively not deterministic

# Multi-threading




t = 2

Interleaving of threads is effectively not deterministic

Consistent results require special precaution, e.g. mutual exclusion

# Multi-threading

- Standard programming model for shared memory, MIMD architectures.
- In C++ (since C++ 11) implemented via `std::thread`
- Other programming languages (such as Java) have analogous libraries

# Fibers

- Similar to threads but scheduled cooperatively, i.e. a fiber yields explicitly to another fiber and is not scheduled preemptively
  - Switching has less overhead than for threads
  - Often used for asynchronous I/O
- Typically implemented on user level

# Further reading


- T. Rauber and G. Rünger, *Parallel Programming*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2013.
- N. Matloff, Programming on Parallel Machines. 2016.
- P. E. McKenney, Is Parallel Programming Hard, And, If So, What Can You Do About It? 2016.
- M. Raynal, Concurrent Programming: Algorithms, Principles, and Foundations. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013.