OS Lab
Session 4: Process Programming & Threads

AUT – CEIT

Instructor: @MerajNouredini

How to create Process?

Do you remember the parent-child relations…?

Create Process

- system
- exec
- fork
- wait

Lets see it in action!

Inter-Process Communication

IPC Mechanisms

- Signals
- Pipes
- Sockets
- Shared Memory
- Semaphores

We’ll see them in the next section

Threads

Threads

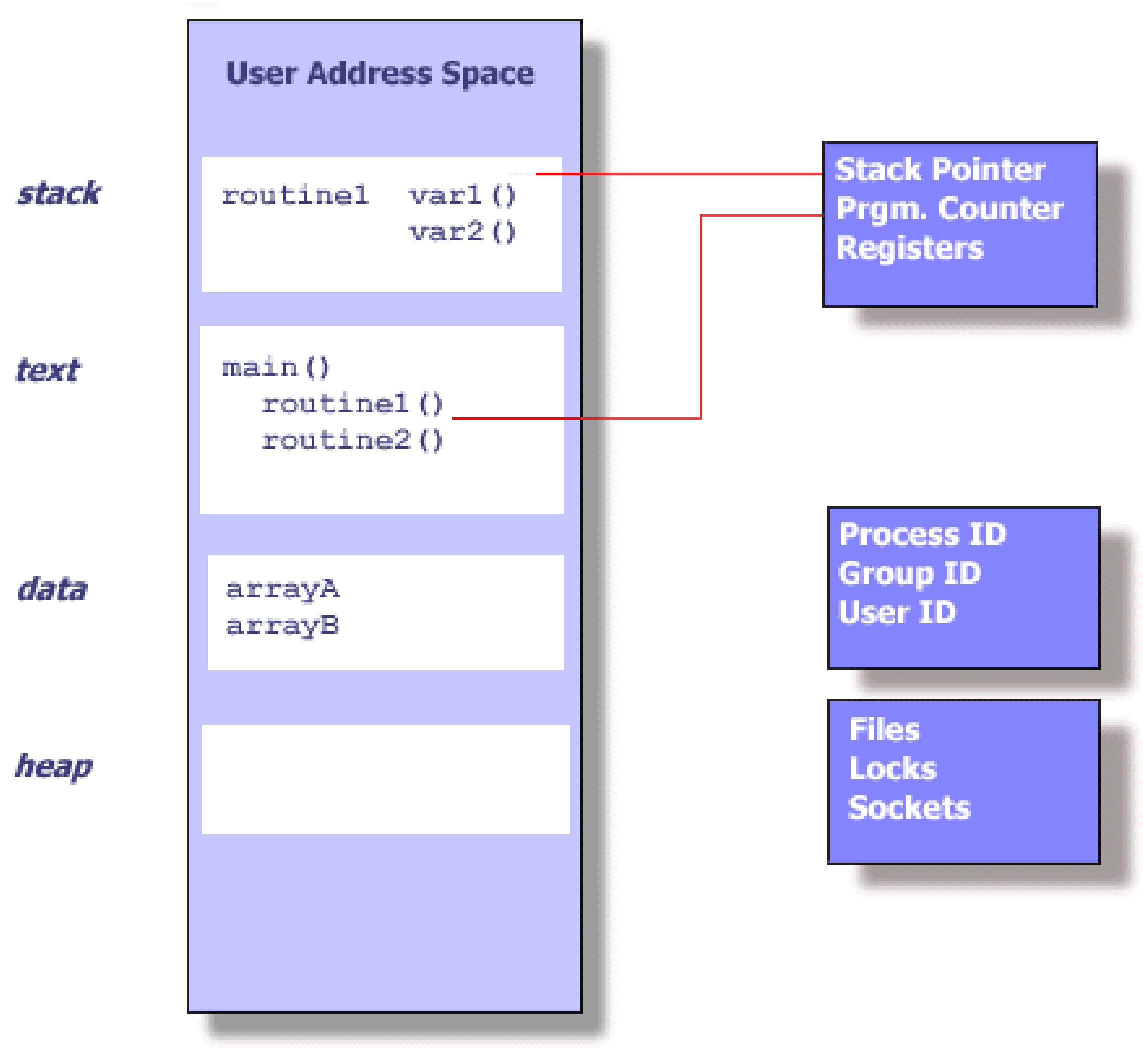
- Technically, a thread is defined as **an independent stream of instructions** that can be scheduled to run as such by the operating system. But what does this mean?
- To the software developer, the concept of a "procedure" that runs independently from its main program may best describe a thread.

Threads

- To go one step further, imagine a main program (a.out) that contains a number of procedures. Then imagine all of these procedures being able to be scheduled to run simultaneously and/or independently by the operating system. That would describe a "multi-threaded" program.

Threads

We Know What is **Process**

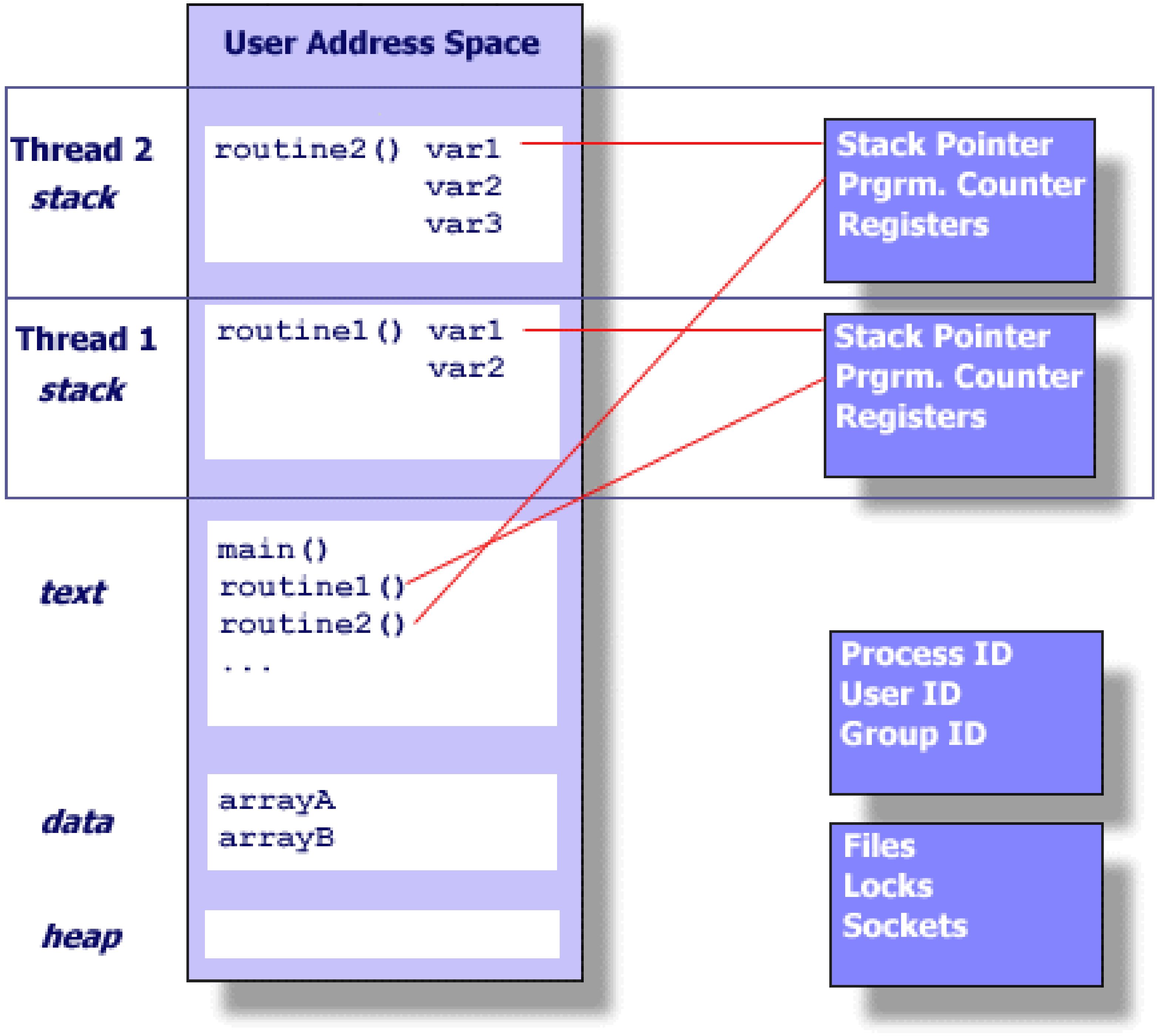


Threads

- Threads use and exist within these process resources, yet are able to be **scheduled** by the operating system and r**un as independent entities** largely because **they duplicate only the bare essential resources** that enable them to exist as executable code.

Threads

- This independent flow of control is accomplished because a thread maintains its own:
  - Stack pointer
  - Registers
  - Scheduling properties (such as policy or priority)
  - Set of pending and blocked signals
  - Thread specific data.


User Address Space
Thread 2
stack
routine2() var1
var2
var3
Stack Pointer
Prgrm. Counter
Registers
Thread 1
stack
routine1() var1
var2
Stack Pointer
Prgrm. Counter
Registers
text
main()
routine1()
routine2()
. . .
data
arrayA
arrayB
heap
Process ID
User ID
Group ID
Files
Locks
Sockets

Threads

- So, in summary, in the UNIX environment a thread:
  - Exists within a process and uses the process resources
  - Has its own independent flow of control as long as its parent process exists and the OS supports it
  - Duplicates only the essential resources it needs to be independently schedulable
  - May share the process resources with other threads that act equally independently (and dependently)
  - Dies if the parent process dies - or something similar
  - Is "lightweight" because most of the overhead has already been accomplished through the creation of its process.

How to use threads?

- Historically, hardware vendors have implemented their own proprietary versions of threads. These implementations differed substantially from each other making it difficult for programmers to develop portable threaded applications.
- In order to take full advantage of the capabilities provided by threads, a standardized programming interface was required.

POSIX threads

PThreads

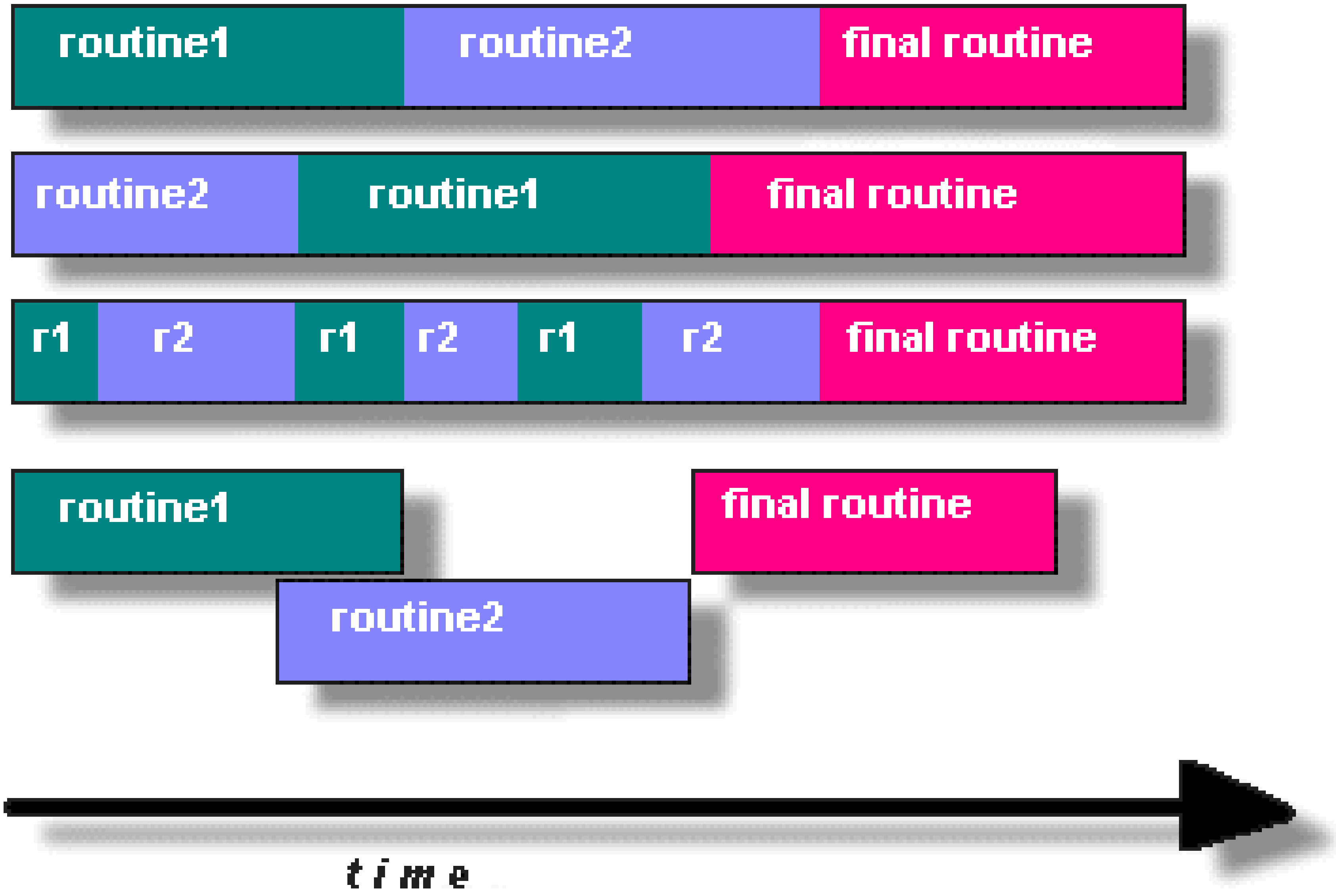
- Why pthreads?
  - When compared to **the cost of creating** and **managing** a process, a thread can be created with much less operating system overhead. Managing threads requires fewer system resources than managing processes

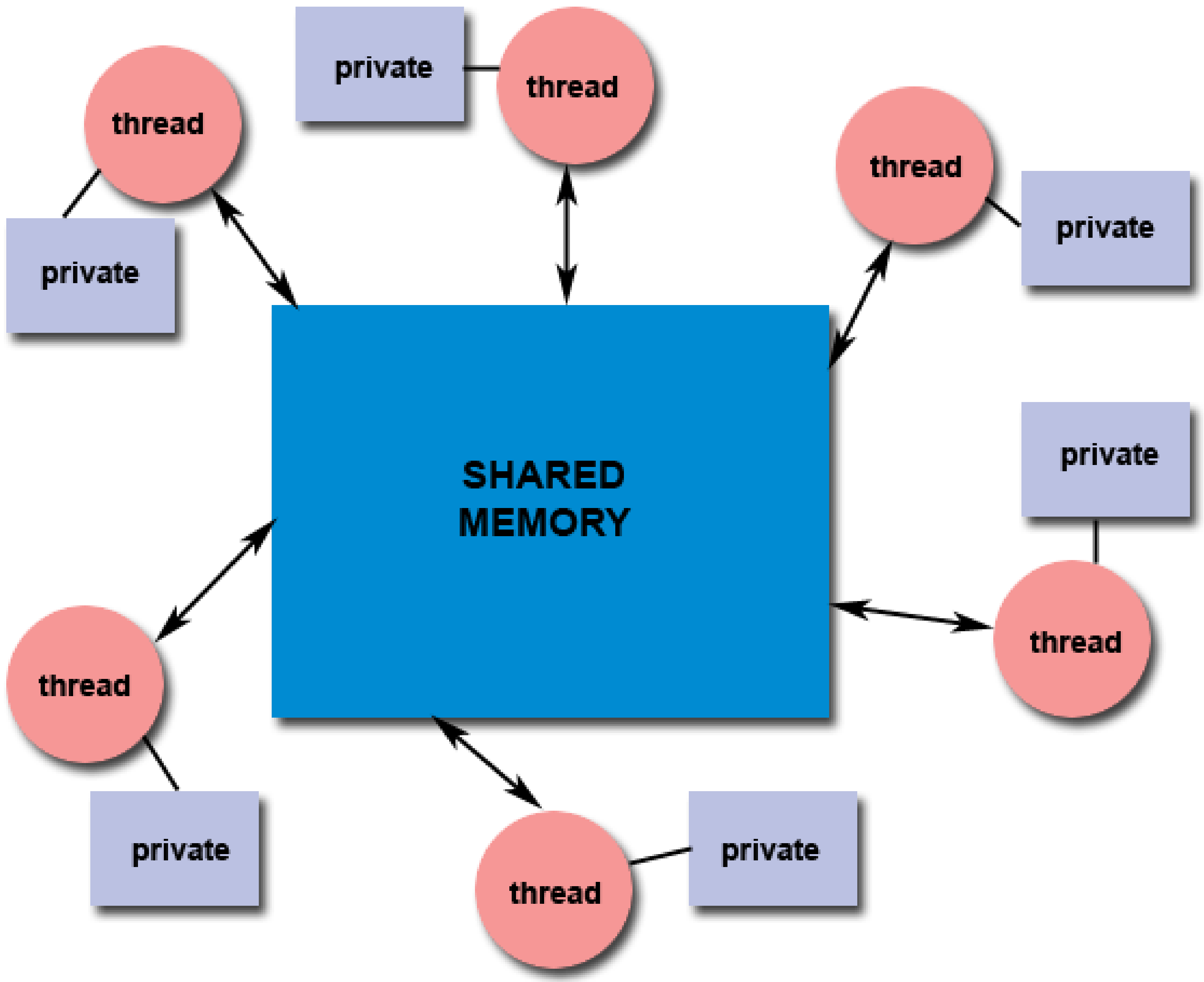
Designing Threaded Programs

Designing Threaded Programs

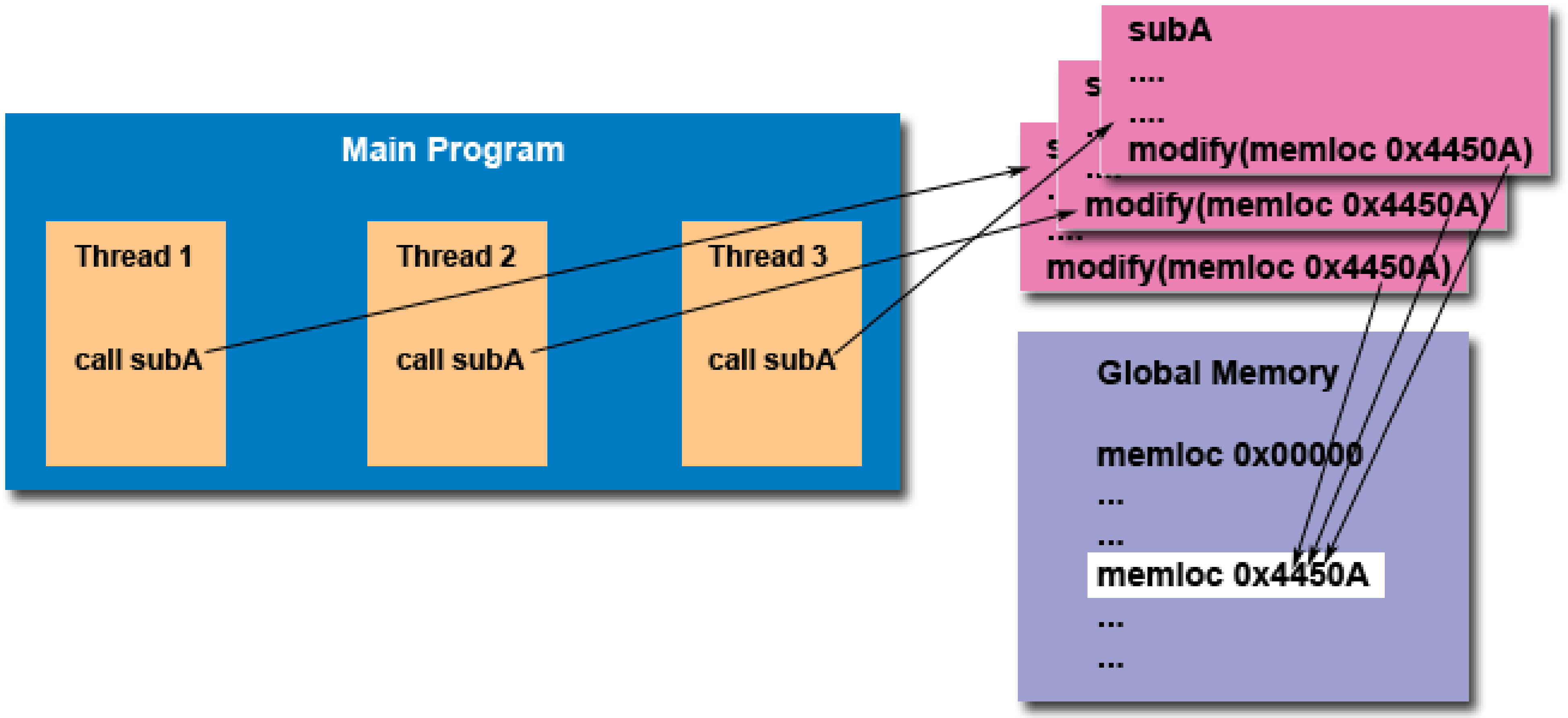
- There are many considerations for designing parallel programs, such as:
  - What type of parallel programming model to use?
  - Problem partitioning
  - Load balancing
  - Communications
  - Data dependencies
  - Synchronization and race conditions
  - Memory issues
  - I/O issues
  - Program complexity
  - Programmer effort/costs/time

They're explained in detail in concurrent programming courses.
https://computing.llnl.gov/tutorials/parallel_comp


routine1
routine2
final routine
routine2
routine1
final routine
r1
r2
r1
r2
r1
r2
final routine
routine1
final routine
routine2
time


private
thread
private
thread
thread
private
SHARED MEMORY
private
thread
thread
private
thread
private

Thread-safeness



How to use pthreads?

The Pthreads API

- Thread management
- Mutexes
- Condition variables
- Synchronization

Lets see it in action!

Questions?