POSIX THREADS PROGRAMMING

Emmanuel S. Pilli

What is a Thread?

- Thread is defined as an independent stream of instructions that can be scheduled to run as such by the operating system
- Its a "procedure" that runs independently from its main program
- Multithreaded Program
  - Imagine a main program that contains a number of procedures
  - Imagine all of these procedures being able to be scheduled to run simultaneously or independently

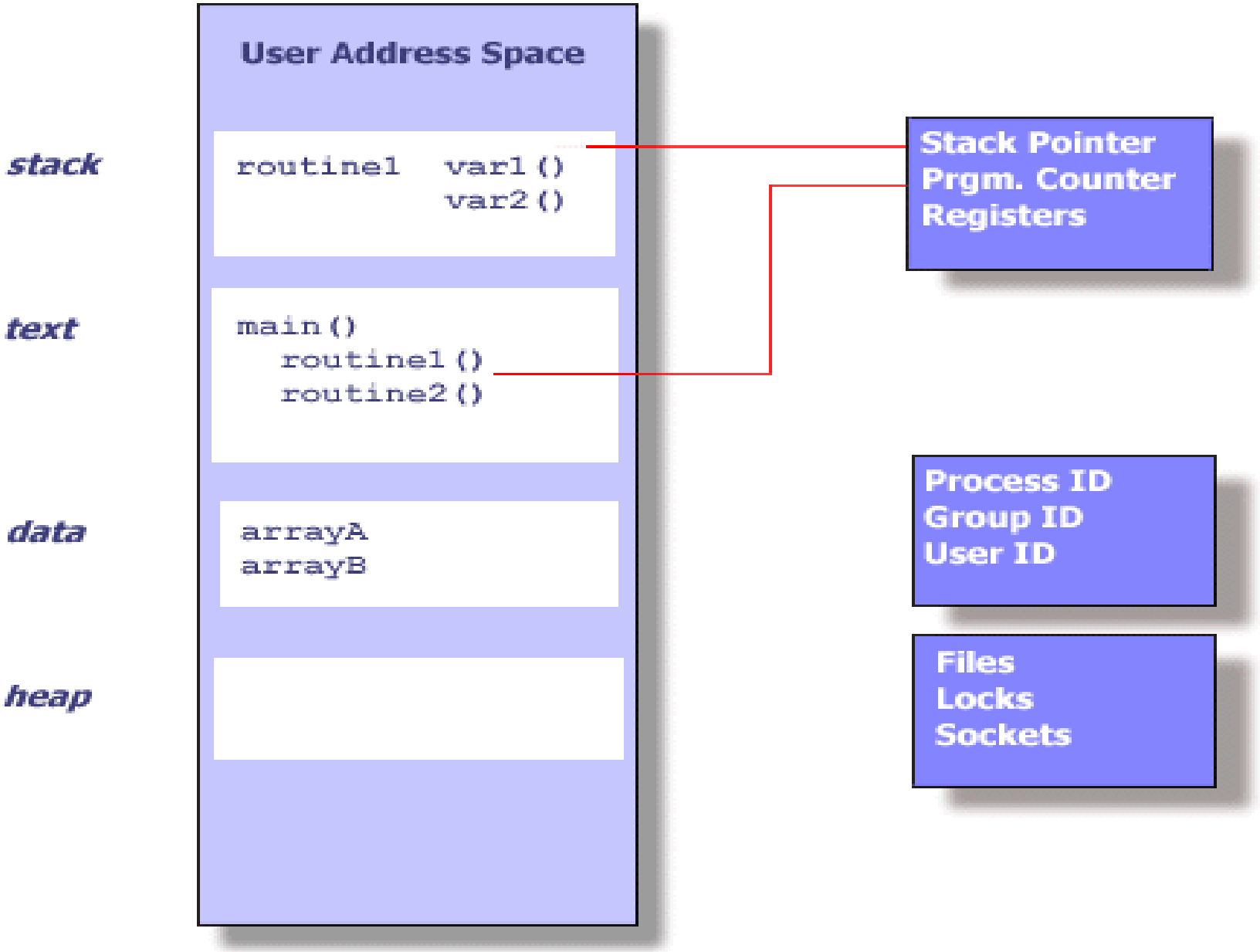
Process

- A process is created by the operating system, and requires a fair amount of "overhead".
- Processes contain information about program resources and program execution state
- Process Information:
  - Process ID, process group ID, user ID, and group ID
  - Environment
  - Working directory
  - Program instructions

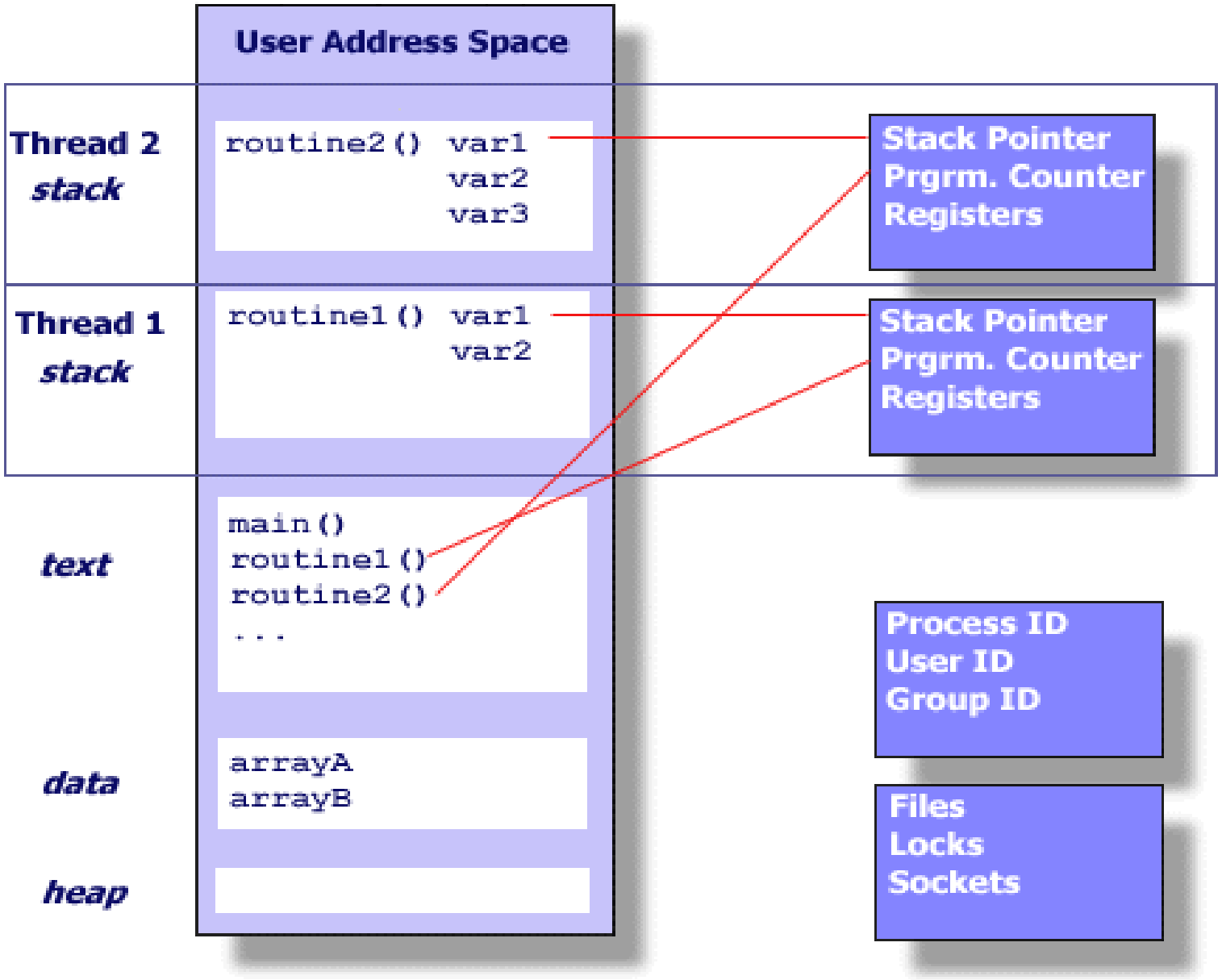
Process

- Process Information:
  - Registers, Stack, Heap
  - File descriptors
  - Signal actions
  - Shared libraries
  - Inter-process communication tools
    - message queues, pipes,
    - semaphores, or shared memory

Unix Process



Threads in a Unix Process



Thread Control

- Threads use and exist within the process resources
- Threads are able to be scheduled by the OS and run as independent entities
- Threads duplicate only the bare essential resources that enable them to exist as executable code
- Thread maintains its own Stack pointer, Registers, Scheduling properties, Set of pending and blocked signals and Thread specific data.

Threads in UNIX

- 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.

What are Pthreads?

- Hardware vendors have implemented their own proprietary versions of threads.
- These implementations differed substantially from each other making portable threaded applications difficult to develop.
- Standardized programming interface was required to take full advantage of the capabilities provided by threads
- IEEE POSIX 1003.1c standard (1995) - Portable Operating System Interface (POSIX)

Pthreads

- Implementations adhering to this standard are referred to as POSIX threads, or Pthreads.
- Most hardware vendors now offer Pthreads in addition to their proprietary API's.
- Pthreads are defined as a set of C language programming types and procedure calls, implemented with a **pthread.h** header / include file and a thread library
- This library may be part of another library, such as **libc**, in some implementations.

Lightweight

- 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.
- For example, the following table compares timing results for the **fork()** subroutine and the **pthread_create()** subroutine.
- Timings reflect 50,000 process/thread creations, were performed with the **time** utility, and units are in seconds, no optimization flags.

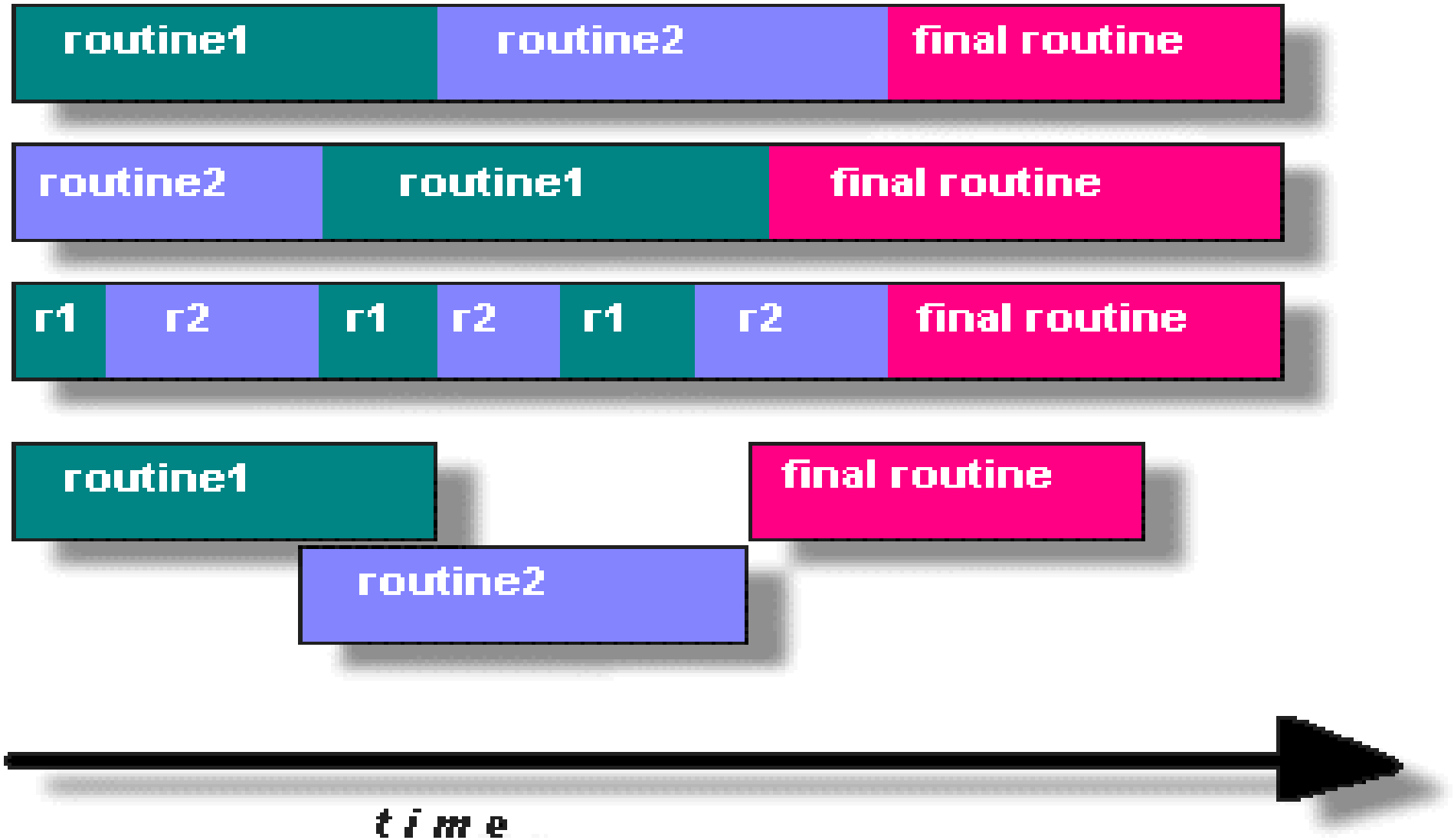
Lightweight

Platform	fork()			pthread_create()		
	real	user	sys	real	user	sys
Intel 2.6 GHz Xeon E5-2670 (16 cores/node)	8.1	0.1	2.9	0.9	0.2	0.3
Intel 2.8 GHz Xeon 5660 (12 cores/node)	4.4	0.4	4.3	0.7	0.2	0.5
AMD 2.3 GHz Opteron (16 cores/node)	12.5	1.0	12.5	1.2	0.2	1.3
AMD 2.4 GHz Opteron (8 cores/node)	17.6	2.2	15.7	1.4	0.3	1.3
IBM 4.0 GHz POWER6 (8 cpus/node)	9.5	0.6	8.8	1.6	0.1	0.4
IBM 1.9 GHz POWER5 p5-575 (8 cpus/node)	64.2	30.7	27.6	1.7	0.6	1.1
IBM 1.5 GHz POWER4 (8 cpus/node)	104.5	48.6	47.2	2.1	1.0	1.5
INTEL 2.4 GHz Xeon (2 cpus/node)	54.9	1.5	20.8	1.6	0.7	0.9
INTEL 1.4 GHz Itanium2 (4 cpus/node)	54.5	1.1	22.2	2.0	1.2	0.6

Concurrent and Parallel Programming

- Problem partitioning
- Load balancing
- Communications
- Data dependencies
- Synchronization and race conditions
- Memory issues
- I/O issues
- Program complexity
- Programmer effort/costs/time

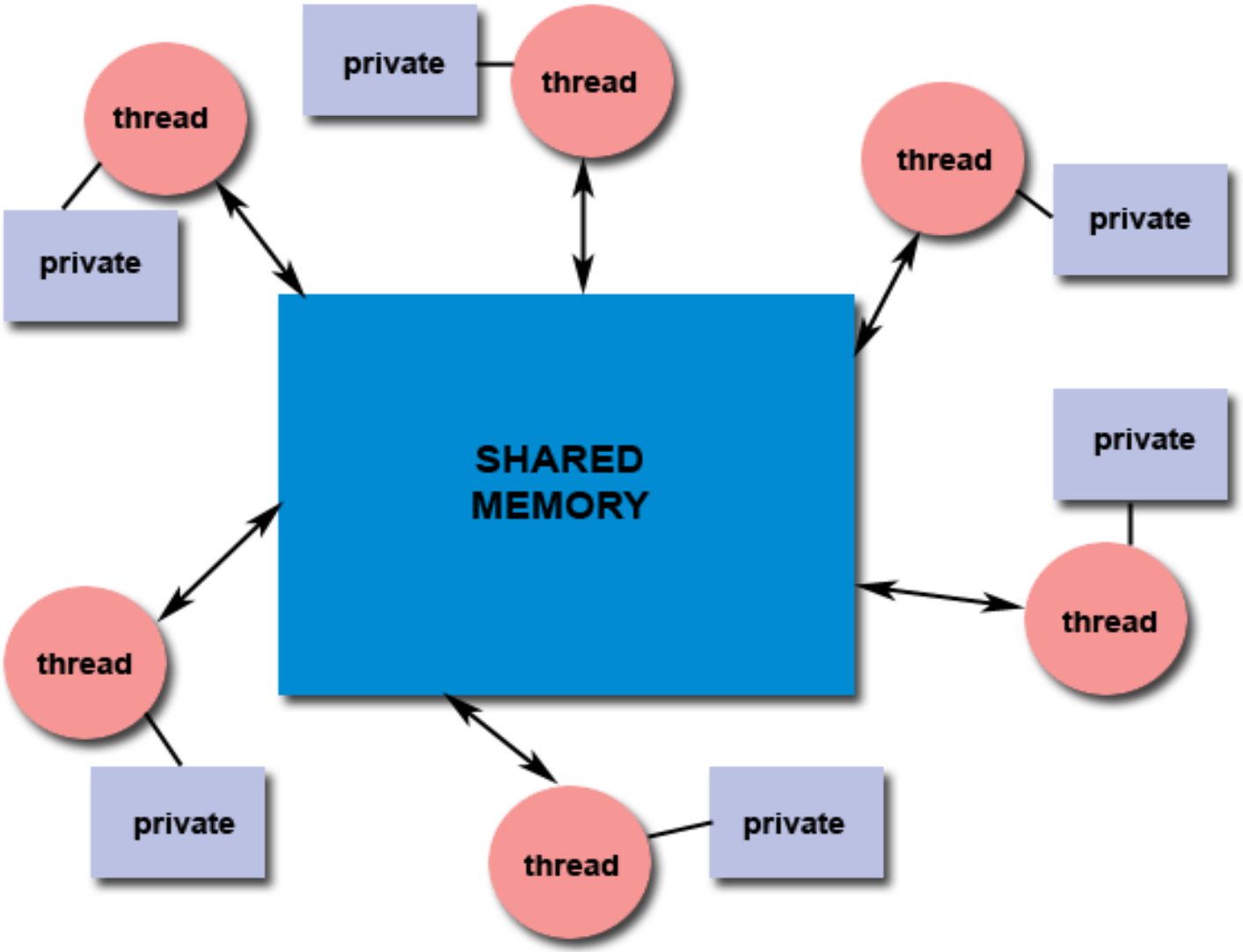
Concurrent (Parallel) Execution



Programs well suited for pthreads

- Work that can be executed, or data that can be operated on, by multiple tasks simultaneously:
- Block for potentially long I/O waits
- Use many CPU cycles in some places but not others
- Must respond to asynchronous events
- Some work is more important than other work (priority interrupts)

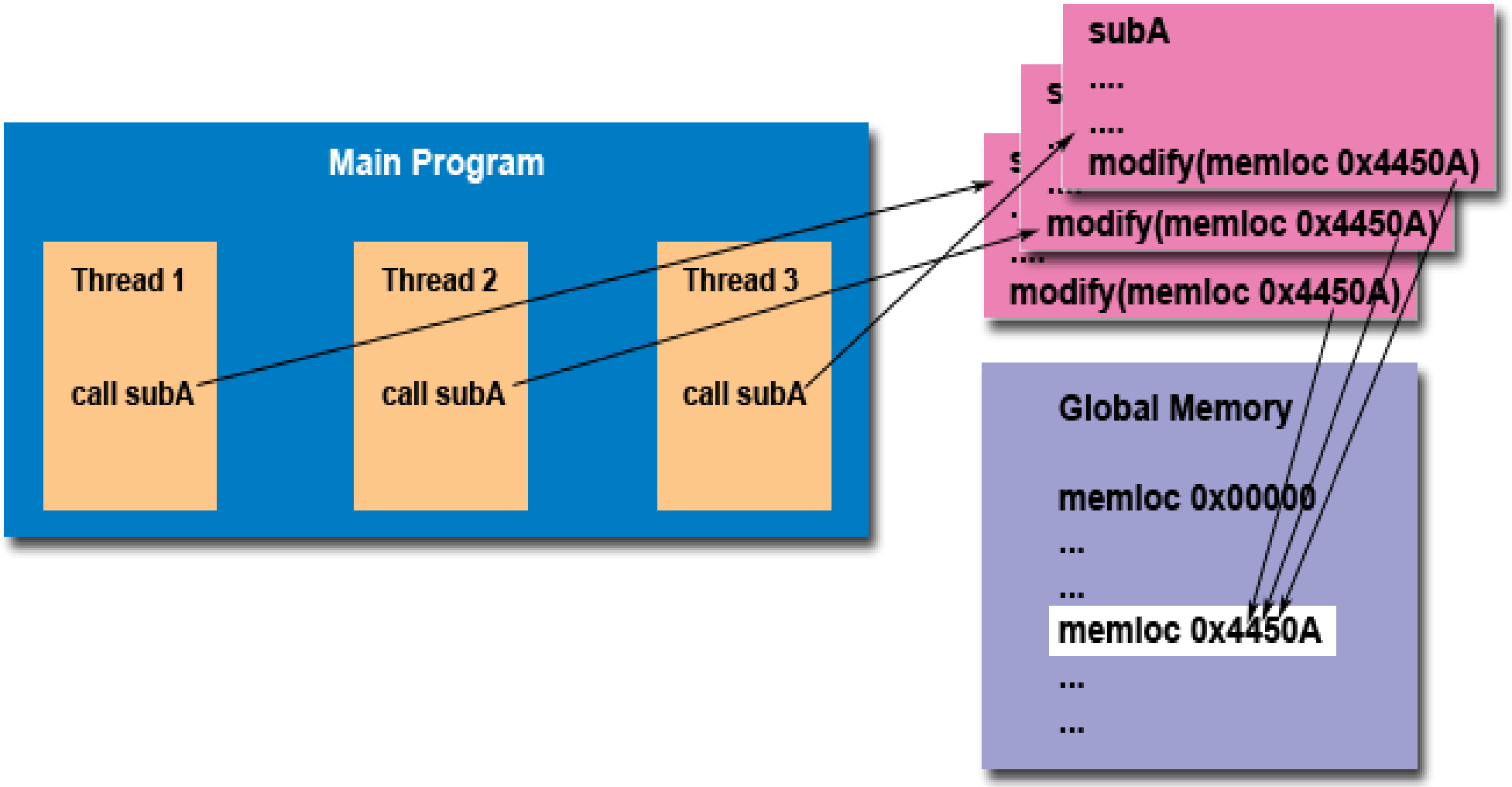
Shared Memory



Shared Memory Model

- All threads have access to the same global, shared memory
- Threads also have their own private data
- Programmers are responsible for synchronizing access (protecting) globally shared data.

Thread Safeness



Thread Safeness

- Thread-safeness refers to an application's ability to execute multiple threads simultaneously without "clobbering" shared data or creating " race" conditions
- If an application creates several threads, each of which makes a call to the same library routine:
  - This library routine modifies a global memory.
  - Each thread may try to modify this global memory location at the same time.
  - If the routine does not employ some sort of synchronization constructs, it is not thread-safe.

Pthreads API

- Pthreads API was defined in the ANSI/IEEE POSIX 1003.1 - 1995
- Four major Categories
  - **Thread management**: Creating, Detaching, Joining
  - **Mutexes** (mutual exclusion) deals with synchronization: Mutex creation, destroying, locking, unlocking
  - **Condition variables**
  - **Synchronization**: Routines that manage read / write locks and barriers

Pthread Library

Routine Prefix	Functional Group
pthread_	Threads themselves and miscellaneous subroutines
pthread_attr_	Thread attributes objects
pthread_mutex_	Mutexes
pthread_mutexattr_	Mutex attributes objects.
pthread_cond_	Condition variables
pthread_condattr_	Condition attributes objects
pthread_key_	Thread-specific data keys
pthread_rwlock_	Read/write locks
pthread_barrier_	Synchronization barriers

Pthread Library

- The Pthreads API contains around 100 subroutines
- For portability, the **pthread.h** header file should be included in each source file using the Pthreads library.
- The current POSIX standard is defined only for the C language.
- Compile Commands - GNU C

  gcc –pthread

Creating and Managing Threads

- pthread_create (thread, attr, start_routine, arg)
- pthread_exit (status)
- pthread_cancel (thread)
- pthread_attr_init (attr)
- pthread_attr_destroy (attr)

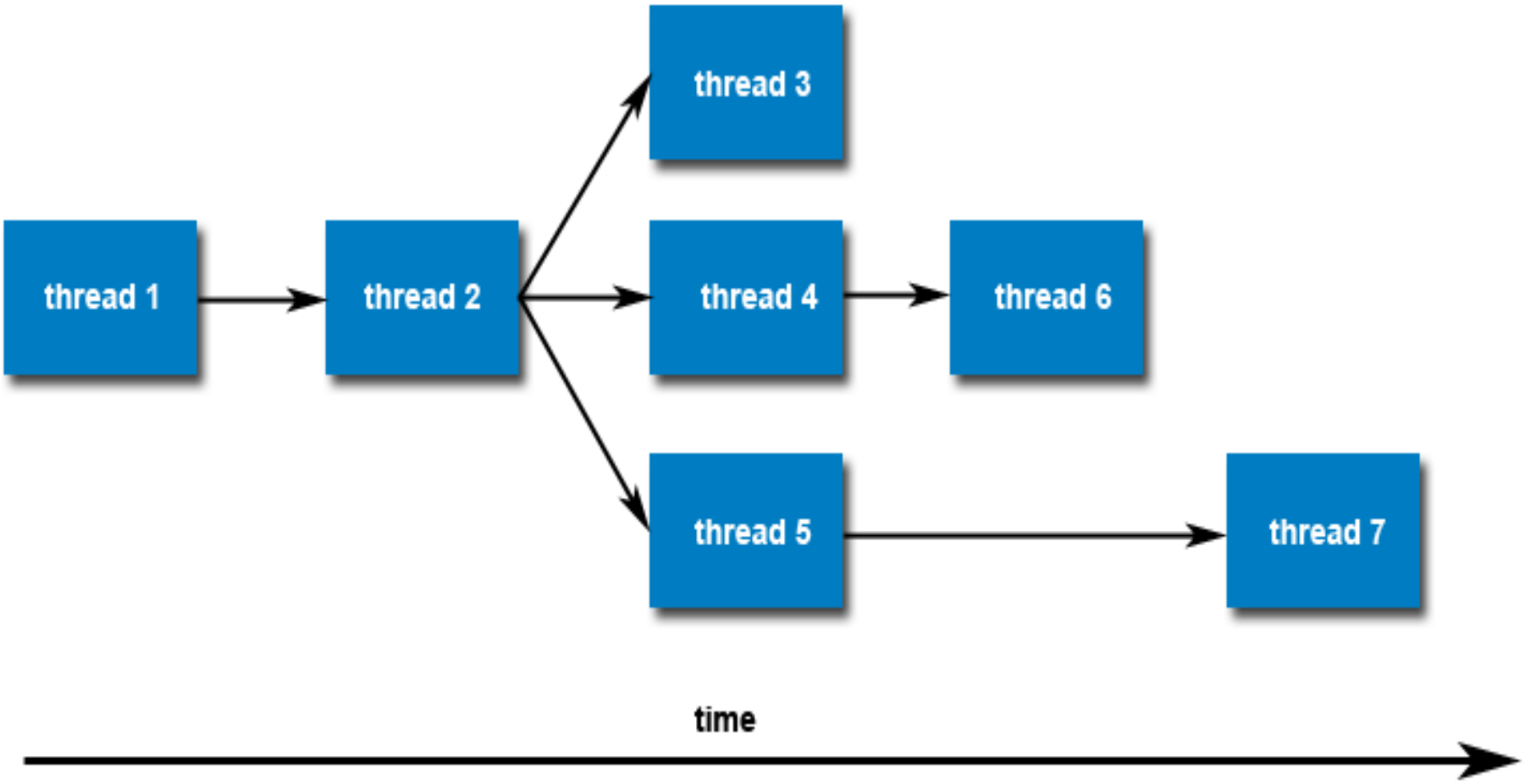
Creating Threads

- main() program comprises a single, default thread
- All other threads must be explicitly created by the programmer
- **pthread_create** creates a new thread and makes it executable
- This routine can be called any number of times from anywhere within your code
- Once created, threads are peers, and may create other threads
- There is no implied hierarchy or dependency between threads.

pthread_create – Arguments

- thread: A unique identifier for the new thread returned by the subroutine.
- attr: An attribute object that may be used to set thread attributes. We can specify a thread attributes object, or NULL for the default values.
- start_routine: C routine that the thread will execute once it is created.
- arg: A single argument that may be passed to *start_routine*. It must be passed by reference as a pointer cast of type void. NULL may be used if no argument is to be passed

Creating Threads



Thread Attributes

- Thread is created with certain default attributes. Some of these attributes can be changed by the programmer via the thread attribute object.
- **pthread_attr_init** & **pthread_attr_destroy** are used to initialize / destroy the thread attribute object.
- Other routines are then used to query / set specific attributes in the thread attribute object
- Attributes include:
  - Detached or joinable state
  - Scheduling inheritance, policy, parameters, contention scope
  - Stack size, address, guard (overflow) size

Terminating Threads

- The thread returns normally from its starting routine. Its work is done.
- The thread makes a call to the **pthread_exit** subroutine - whether its work is done or not.
- The thread is canceled by another thread via the **pthread_cancel** routine.
- The entire process is terminated due to making a call to either the **exec() or exit()**
- If main() finishes first, without calling **pthread_exit** explicitly itself

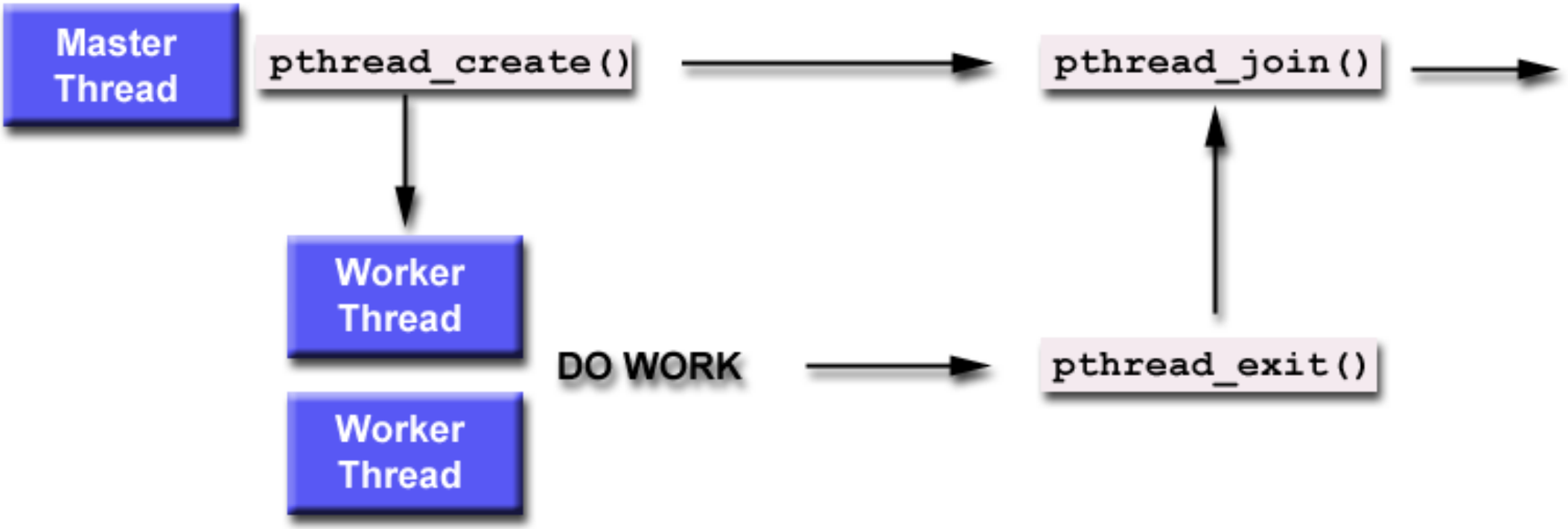
Terminating Threads

- The **pthread_exit()** routine allows the programmer to specify an optional termination *status* parameter
- This optional parameter is typically returned to threads " joining" the terminated thread
- In subroutines that execute to completion normally, calling **pthread_exit()** can often be dispensed with unless the optional status code needs to be passed back
- Cleanup: the **pthread_exit()** routine does not close files; any files opened inside the thread will remain open after the thread is terminated

Joining and Detaching Threads

- pthread_join (threadid, status)
- pthread_detach (threadid)
- pthread_attr_setdetachstate (attr, detachstate)
- pthread_attr_getdetachstate (attr, detachstate)

Joining


Master Thread
pthread_create()
pthread_join()
Worker Thread
Worker Thread
DO WORK
pthread_exit()

Joining

- "Joining" is one way to accomplish synchronization between threads
- The **pthread_join()** subroutine blocks the calling thread until the specified threadid thread terminates.
- The programmer is able to obtain the target thread's termination return status if it was specified in the target thread's call to **pthread_exit()** .
- A joining thread can match one **pthread_join()** call
- It is a logical error to attempt multiple joins on the same thread.

Joinable?

- When a thread is created, one of its attributes defines whether it is joinable or detached
- Only threads that are created as joinable can be joined
- If a thread is created as detached, it can never be joined
- The final draft of the POSIX standard specifies that threads should be created as joinable
- To explicitly create a thread as joinable or detached, the attr argument in the **pthread_create()** routine is used

Joinable?

- Four Step Process:
  - Declare a pthread attribute variable of the pthread_attr_t data type
  - Initialize the attribute variable with **pthread_attr_init()**
  - Set the attribute detached status with **pthread_attr_setdetachstate()**
  - When done, free library resources used by the attribute with **pthread_attr_destroy()**

Detaching

- The **pthread_detach()** routine can be used to explicitly detach a thread even though it was created as joinable
- There is no converse routine