CS343: Operating System

Threading and Synchronization

Lect16 : 04th Sept 2023

Dr. A. Sahu

Dept of Comp. Sc. & Engg.

Indian Institute of Technology Guwahati

Outline

- Threading
- Threading Examples
- Thread mappings
  - Pthread/Uthread, Kthread, Hthread
- Synchronization

Multithreading

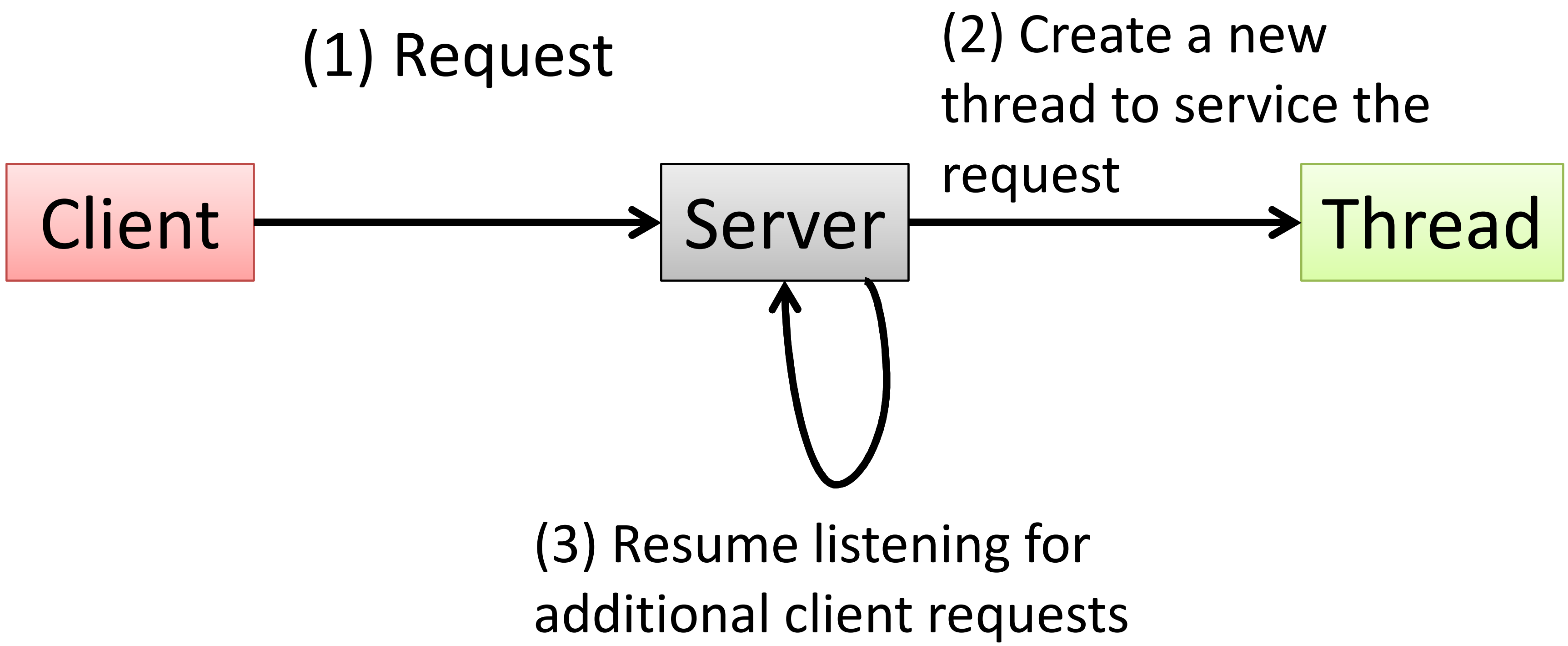
Thread: Motivation

- Most modern applications are multithreaded
- Threads run within application
- Multiple tasks with the application can be implemented by separate threads
  - Update display, Fetch data, Spell checking, Answer a network request

Thread: Motivation

- Process creation is heavy-weight while thread creation is light-weight
  - Thread as Light Weight Process
- Can simplify code, increase efficiency
- **Kernels are generally multithreaded**

Multithreaded Server Architecture



Benefits of multithreading

- **Responsiveness –** may allow continued execution if part of process is blocked, especially important for user interfaces
- **Resource Sharing –** threads share resources of process, easier than **shared memory or message passing (to be discussed IPC/send/pipe)**
- **Economy –** cheaper than process creation, thread switching lower overhead than context switching
- **Scalability –** process can take advantage of multiprocessor architectures

Multicore Programming

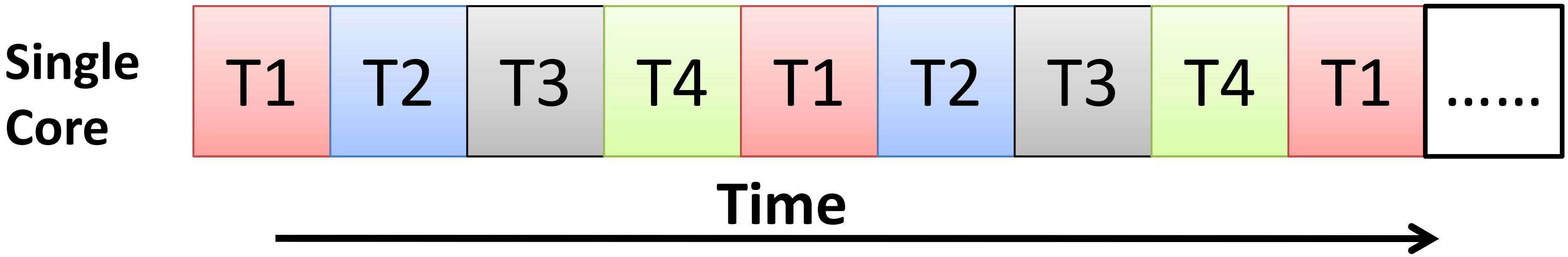
- **Multicore or multiprocessor** systems putting pressure on programmers, challenges include:
  - **Dividing activities, Balance**
  - **Data splitting, Data dependency**
  - **Testing and debugging**
- ***Parallelism*** implies a system can perform more than one task simultaneously
- ***Concurrency*** supports more than one task making progress
  - Single processor / core, scheduler providing concurrency

Multicore Programming (Cont.)

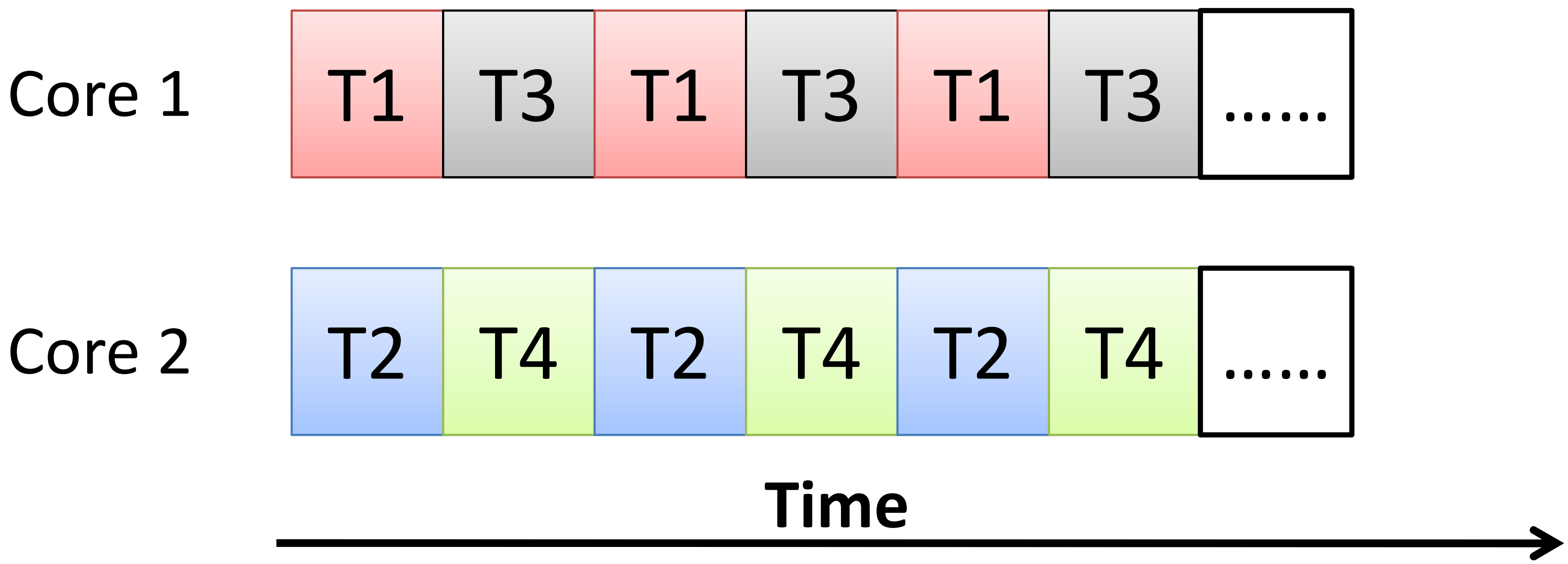
- Types of parallelism
  - **Data parallelism** – distributes subsets of the same data across multiple cores, same operation on each
  - **Task parallelism** – distributing threads across cores, each thread performing unique operation
- As # of threads grows, so does architectural support for threading
  - CPUs have cores as well as ***hardware threads***
  - **AMD thread ripper** with 128 cores, and 2 hardware threads per core
  - **Intel Core i7**: 8 cores, 2 thread per core

Concurrency vs. Parallelism

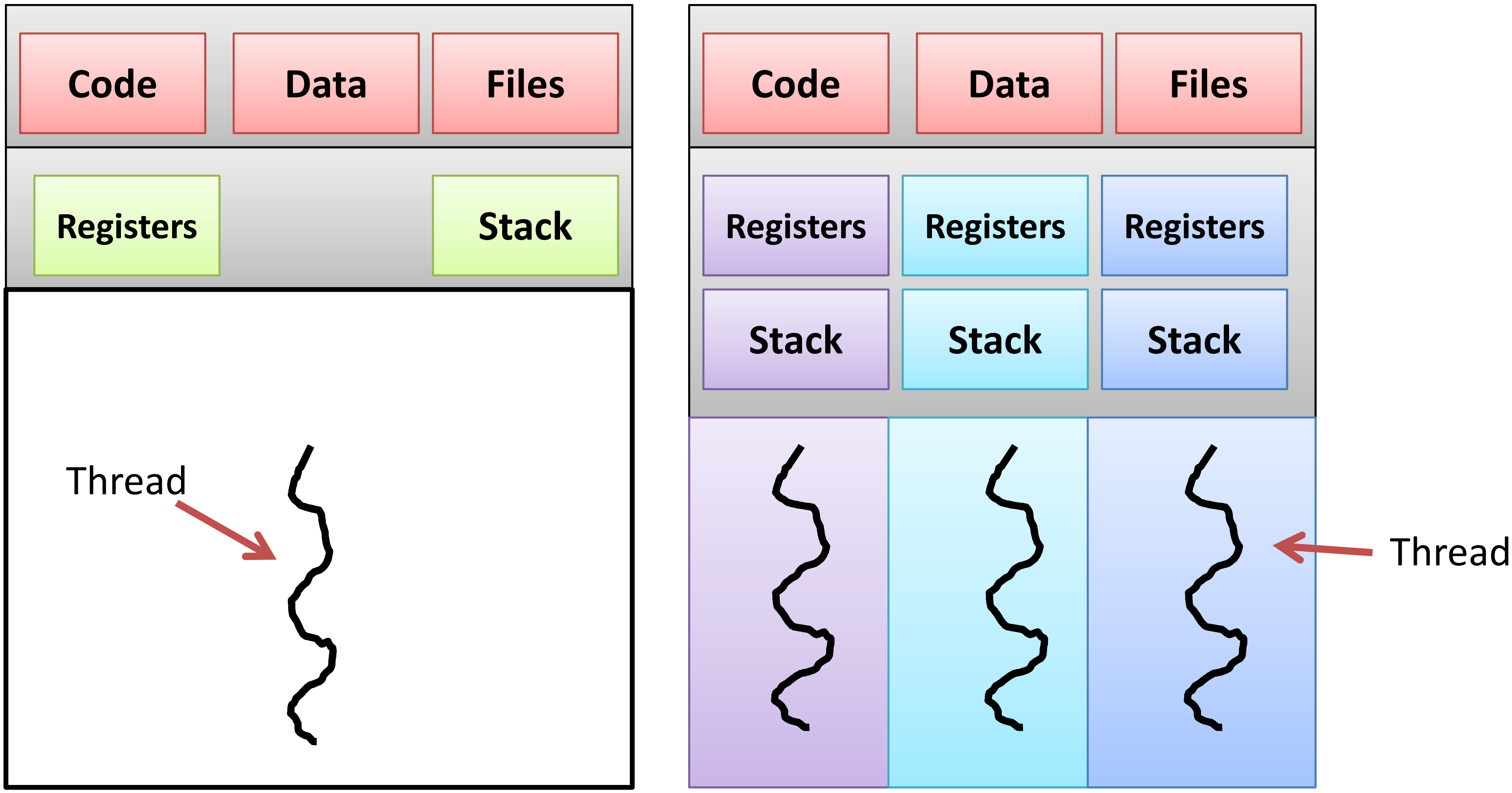
- Concurrent execution on single-core



- Parallelism on a multi-core system:



Single and Multithreaded Processes



Single-threaded process **Multi-threaded process**

Multiprocess Vs Mutithreaded

- **Mutithreaded Apps**
  - Code, Data and Files are shared
  - Easy to create, terminate and schedule/manage in user level, Faster as compared to process
  - Data shared: Lock
  - OS treat all thread of one process as one process. If one Thread block by I/O all the thread get blocked
- **Multiprocess Apps**
  - Code, Data and Files are not shared. So creation and termination is costly
  - Data/Information shared through communication
    - IPC : Pipe, Socket, send/recieve

Multi-process Vs Multi-threaded Apps

- **Adv thread: Thread have less overhead to start/terminate than process**
  - Very little memory copying is required
  - Threads are faster to start than processes.
  - To start a process, the whole process area must be duplicated for the new process copy to start.

Multi-process Vs Multi-threaded Apps

- **Adv of Thread:** Faster task-switching
  - Faster switching between threads
  - The CPU caches and program context can be maintained between threads in a process But reloaded in case of switching to other process.
- Adv of Thread: Data sharing
  - For tasks that require sharing large amounts of data
  - All threads share a process's memory pool

Multi-process Vs Multi-threaded Apps

- Disadvantage of thread over process
- DisAdv Thread: Synchronization overhead of shared data
  - Shared data that is modified requires special handling in the form of locks, mutexes
  - To ensure that data is not being read while written, nor written by multiple threads at the same time.

Multi-process Vs Multi-threaded Apps

- **DisAdv Thread:** Shared process memory space
  - All threads in a process share the same memory space.
  - If something goes wrong in one thread and causes data corruption or an access violation, then this affects and corrupts all the threads in that process
- **DisAdv Thread:** Program debugging
  - multi-threaded programs present difficulties in finding and resolving bugs
  - Synchronization issues, non-deterministic timing and accidental data corruption all conspire to make debugging more difficult

Inter-process Communication (IPC)

- Many application run multiple process to do a collaborative work
  - Example: Chrome Browser, IE9, Adobe PDF,
- They need to share some information to do the collaboration
- Information sharing
  - Message passing read/write or send/recive
  - Pipe
  - Socket

Multiprocessor Programming using Theading

Threading Language and Support

- Initially threading used for
  - Multiprocessing on single core : earlier days
  - Feeling/simulation of doing multiple work simultaneously even if on one processor
  - Used TDM, time slicing, Interleaving
- Now a day threading mostly used for
  - To take benefit of multicore
  - Performance and energy efficiency
  - We can use both TDM and SDM

Thread Libraries

- **Thread library** provides programmer
  - API for creating and managing threads
- Two primary ways of implementing
  - Library entirely in user space
  - Kernel-level library supported by the OS

Pthreads

- May be provided either as user-level or kernel-level
- A POSIX standard (IEEE 1003.1c) API for thread creation and synchronization
  - Posix (Portable OS Interface)
- API specifies behavior of the thread library, implementation is up to development of the library
- Common in UNIX operating systems
  - Solaris, Linux, Mac OS X

Threading Language and Support

- Pthread: POSIX thread
  - Popular, Initial and Basic one
- Improved Constructs for threading
  - c++ thread : available in c++11, c++14
  - **Java thread : very good memory model**
    - **Atomic function, Mutex**
- Thread Pooling and higher level management
  - **OpenMP (loop based)**
  - **Cilk (dynamic DAG based)**

Pthread, C++ Thread, Cilk and OpenMP

```
pthread_t tid1, tid2;
pthread_create(&tid1,NULL,Fun1, NULL);
pthread_create(&tid2,NULL,Fun2, NULL);
pthread_join(tid1, NULL);
pthread_join(tid2, NULL);
```

Pthread

```
thread t1(Fun1);
thread t1(Fun2, 0, 1, 2);
  // 0, 1,2 param to Fun2
t1.join();
t2.join();
```

C++ thread

Pthread, C++ Thread, Cilk and OpenMP

OpenMP

```
#pragma omp parallel for
for(i=0;i<N;i++)
     A[i]=B[i]*C[i];
//Auto convert serial code to threaded code
// $gcc -fopenmp test.c;
 $export OMP_NUM_THREADS=10
 $ ./a.out
```

Cilk

```
cilk fib (int n){
 //Cilk dynamic parallism, DAG recursive code
     if (n<2) return n;
     int x=spawn fib(n-1); //spawn new thread
     int y=spawn fib(n-2); //spawn new thread
     sync;
     return x+y;
}
```

Programming with Threads

- Threads
- Shared variables
- The need for synchronization
- Synchronizing with semaphores
- Thread safety and reentrancy
- Races and deadlocks

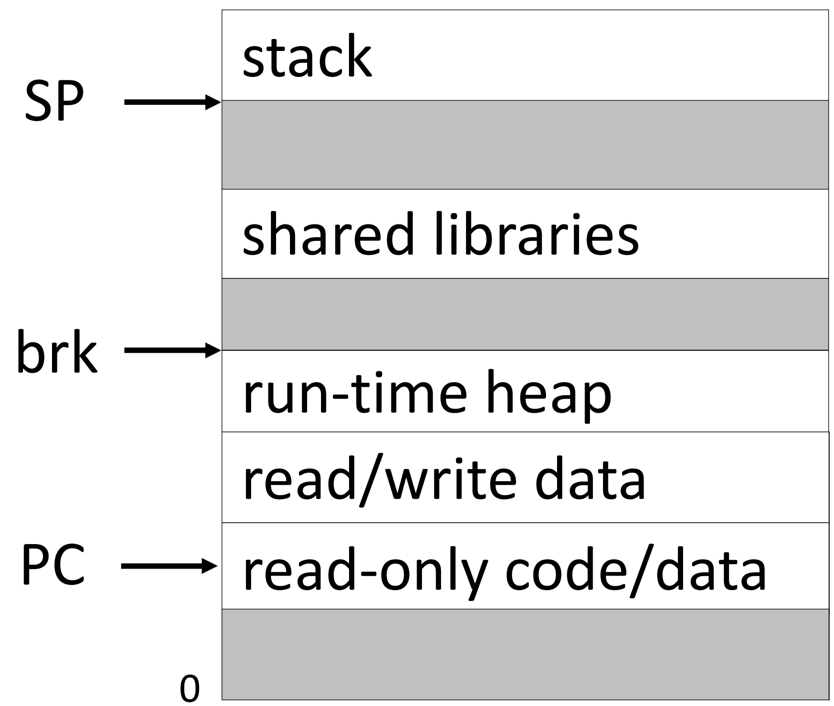
Traditional View of a Process

- Process = process context + code, data, and stack

Process context

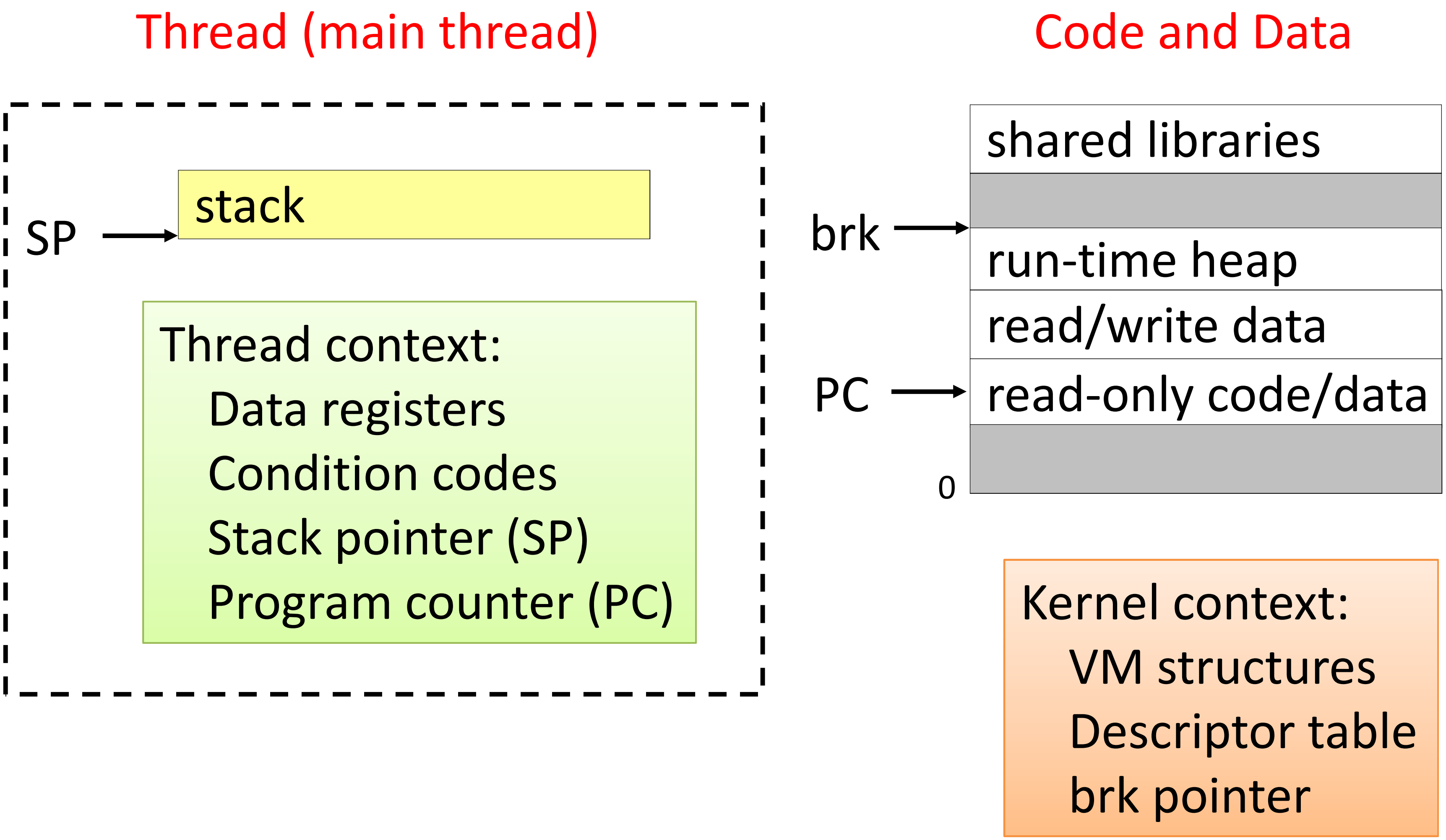
Program context:
Data registers
Condition codes
Stack pointer (SP)
Program counter (PC)
Kernel context:
VM structures (VMem)
Descriptor table
brk pointer

Code, data, and stack



Alternate View of a Process

- Process = thread+ code, data & kernel context



A Process With Multiple Threads

- Multiple threads can be associated with a process
  - Each thread has its *own* logical control flow (sequence of PC values)
  - Each thread *shares* the same code, data, and kernel context
  - Each thread has its own thread id (TID)

A Process With Multiple Threads

Thread 1 (main thread)

stack 1

Thread 1 context:
- Data registers
- Condition codes
- SP1
- PC1

Shared code and data

shared libraries

run-time heap

read/write data

read-only code/data

0

Kernel context:
- VM structures
- Descriptor table
- brk pointer

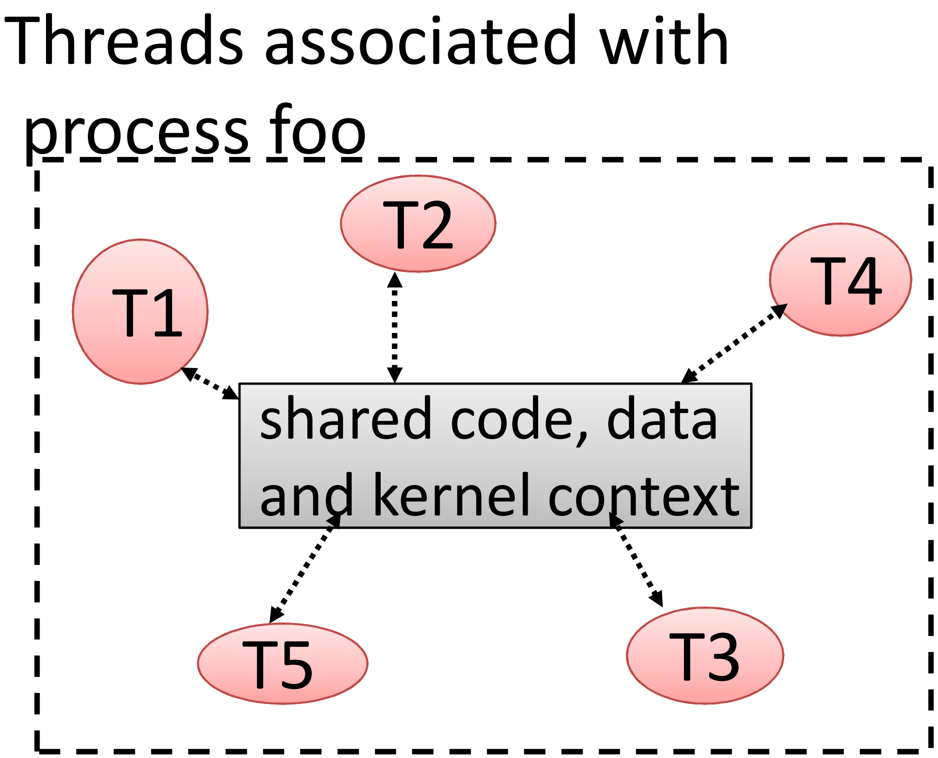
Thread 2 (peer thread)

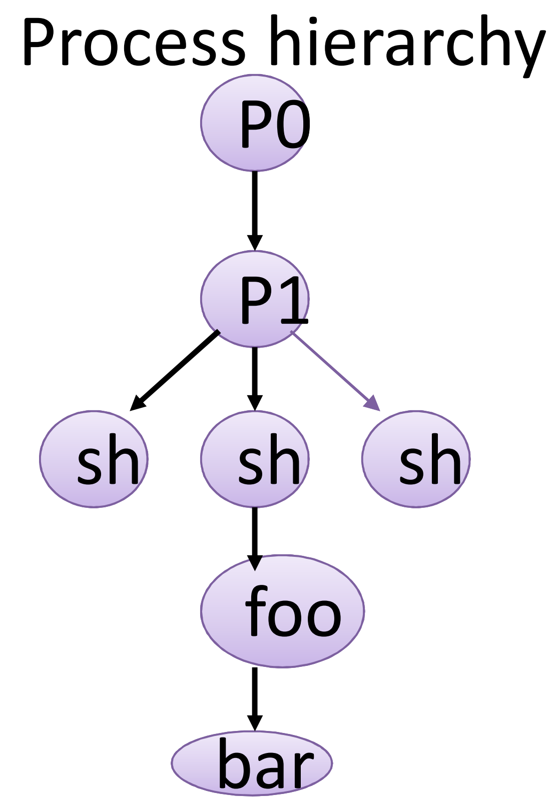
stack 2

Thread 2 context:
- Data registers
- Condition codes
- SP2
- PC2

Logical View of Threads

- Threads associated with a process form a pool of peers
  - Unlike processes, which form a tree hierarchy





Posix Threads (Pthreads) Interface

- **Creating and reaping threads**
  - `pthread_create, pthread_join`
- **Determining your thread ID :** `pthread_self`
- **Terminating threads**
  - `pthread_cancel, pthread_exit`
  - `exit` [terminates all threads] , `return` [terminates current thread]
- **Synchronizing access to shared variables**
  - `pthread_mutex_init, pthread_mutex_[un]lock`
  - `pthread_cond_init, pthread_cond_[timed]wait`

The Pthreads "hello, world" Program

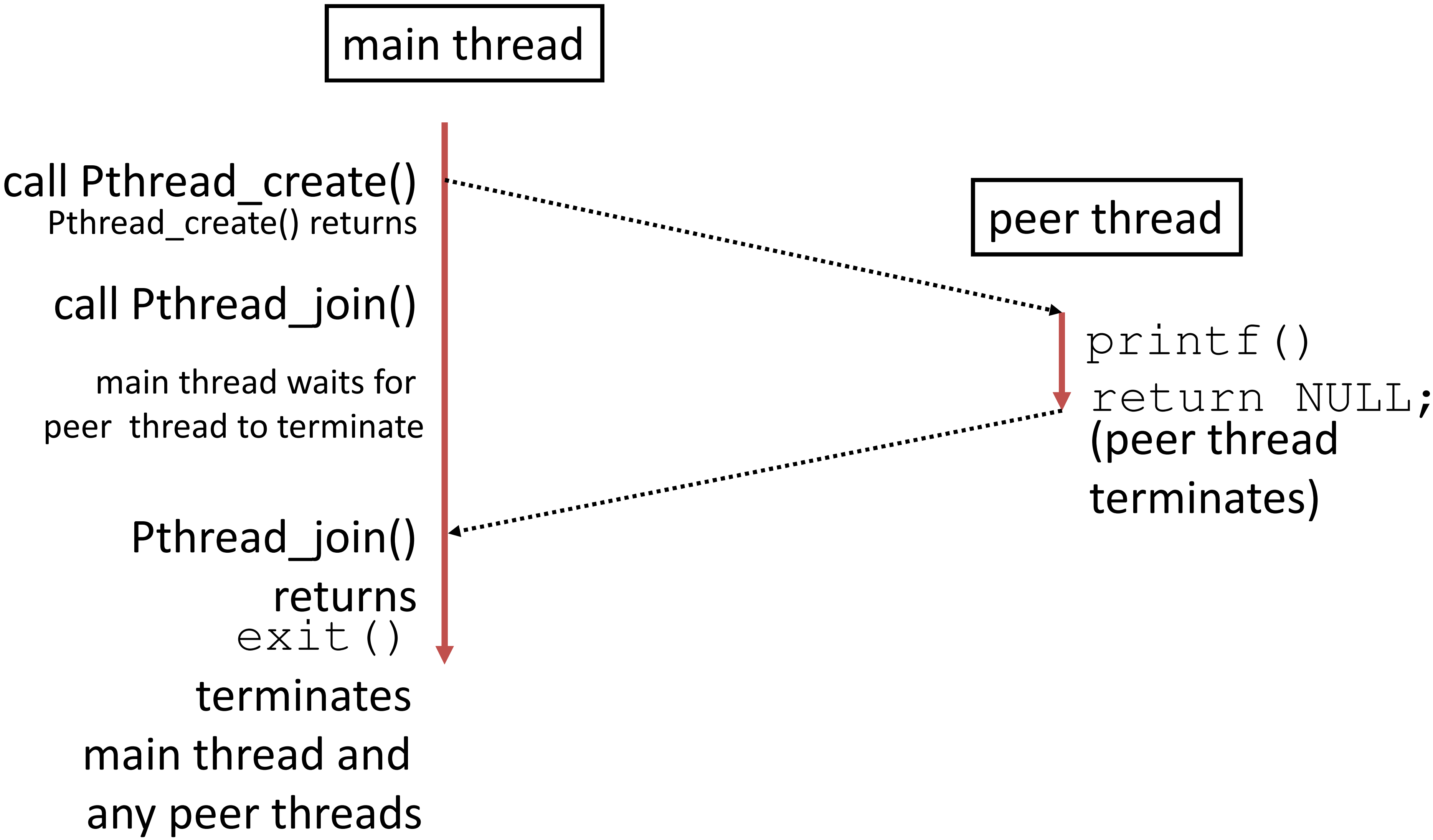
```
/* thread routine */
void *HelloW(void *vargp) {
  printf("Hello, world!\n");
  return NULL;
}

int main() {
  pthread_t tid;
  pthread_create(&tid, NULL, Hellow, NULL);
  pthread_join(tid, NULL);
  return 0;
}
```

Thread attributes (usually NULL)

*Thread arguments (void *p)*

*return value (void **p)*

Execution of Threaded "hello, world"



Pros and Cons: Thread-Based Designs

- **+ Easy to share data structures between threads**
  - E.g., logging information, file cache
- **+ Threads are more efficient than processes**
- **– Unintentional sharing can introduce subtle and hard-to-reproduce errors!**
  - Ease of data sharing is greatest strength of threads
  - Also greatest weakness!

Thanks