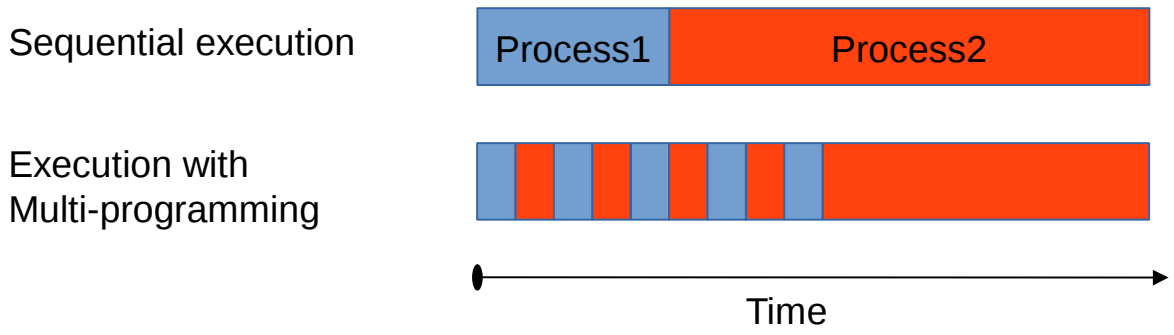
MINISTÉRIO DA CIÊNCIA E TECNOLOGIA
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

Programming with Threads

Emiliano F. Castejon
INPE – Instituto Nacional de Pesquisas Espaciais
DPI – Divisão de Processamento de Imagens

Multiprogramming

Multiprogramming is a basic form of “parallel” processing in which several programs are run at the same time on a uniprocessor (shared time execution).

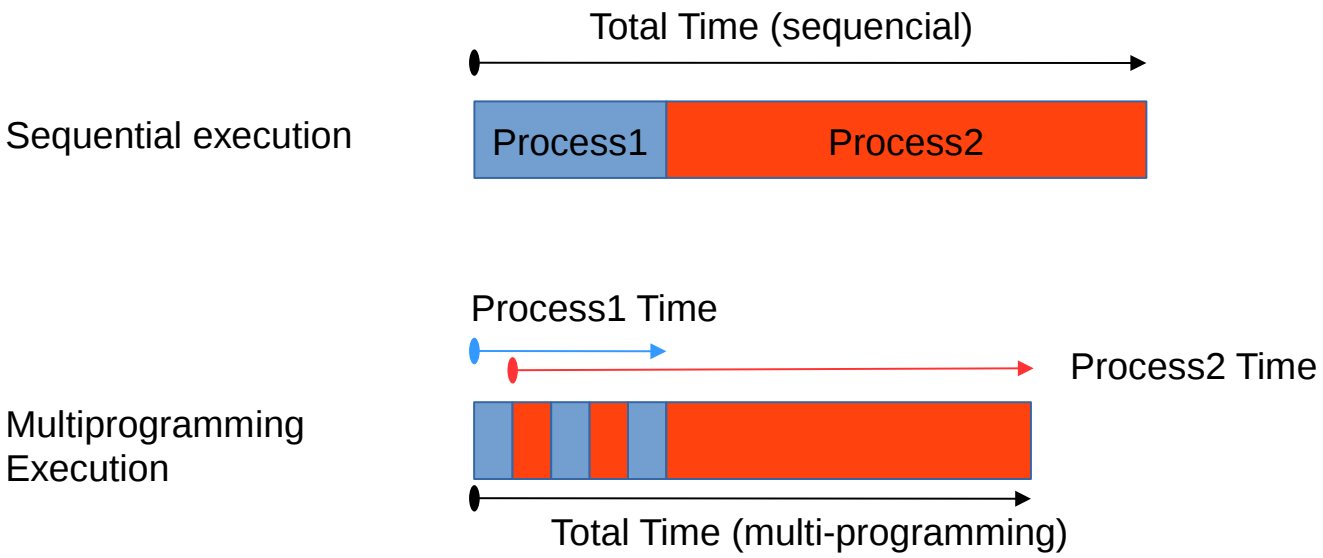


Multiprogramming

Example

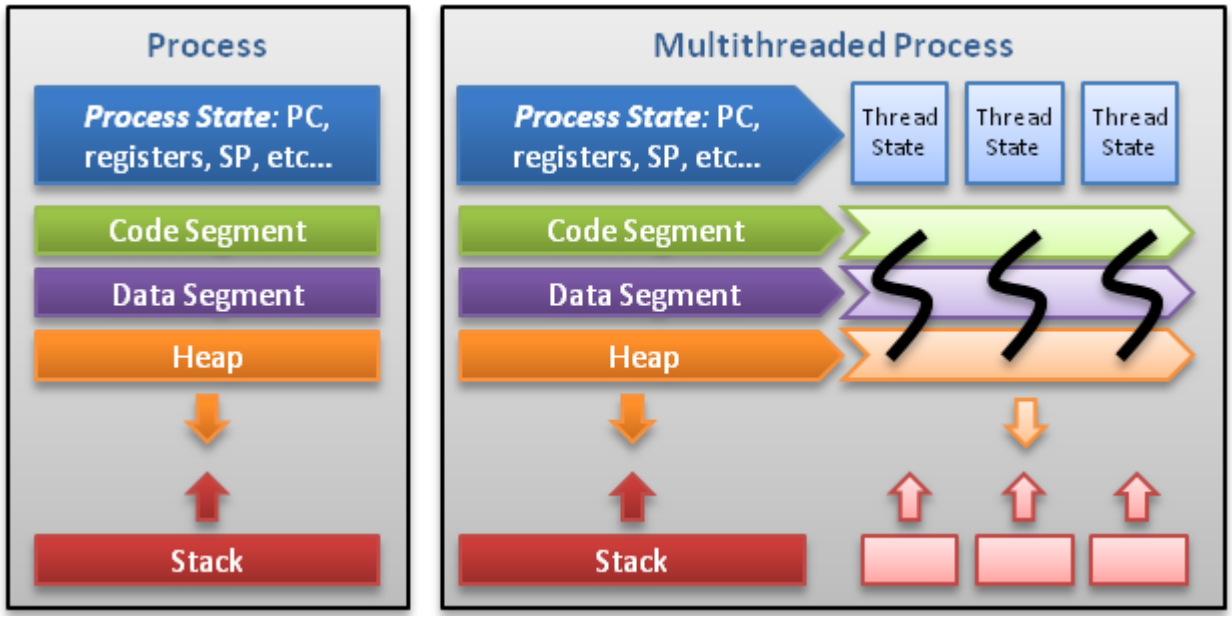
Process 1 – I/O (receiving network data)

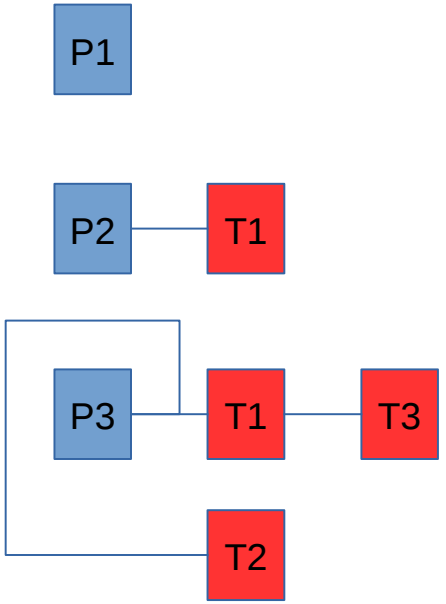
Process 2 – Intensive computation (CPU)



Multi-thread programming

A thread (lightweight process) is a sequence of such instructions within a program that can be executed independently of other code.





Multi-thread programming

Thread (spiritual exotic definition):

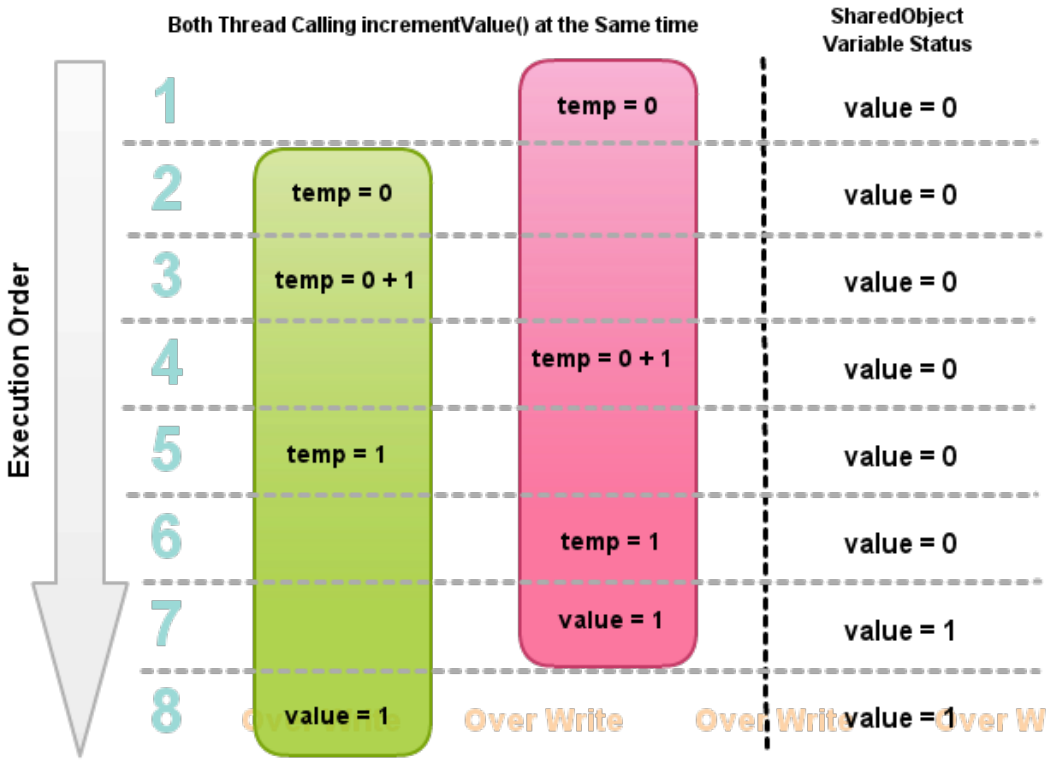
Multi-thread programming

Shared Resources

Threads may operate on disparate data, but often threads may have to touch the same data. It is unsafe to allow concurrent access to such data

```
Global integer value = 0;

incrementValue()
{
  integer temp = value;
  temp = temp + 1 ;
  value = temp;
}
```



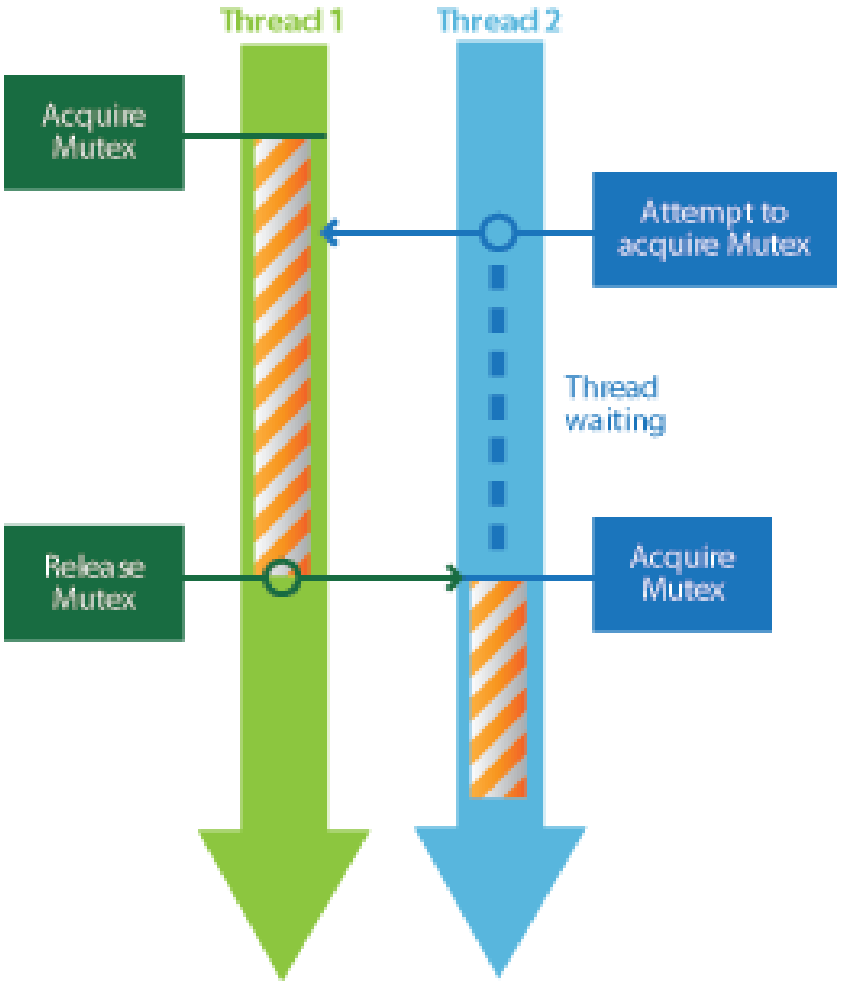
Multi-thread programming

Thread synchronization mechanisms: Mutex, Semaphore, Condition Variables, Barries, others.

Mutex (mutual exclusion): Only one thread can lock (or own) a mutex variable at any given time. Thus, even if several threads try to lock a mutex only one thread will be successful. No other thread can own that mutex until the owning thread unlocks that mutex.

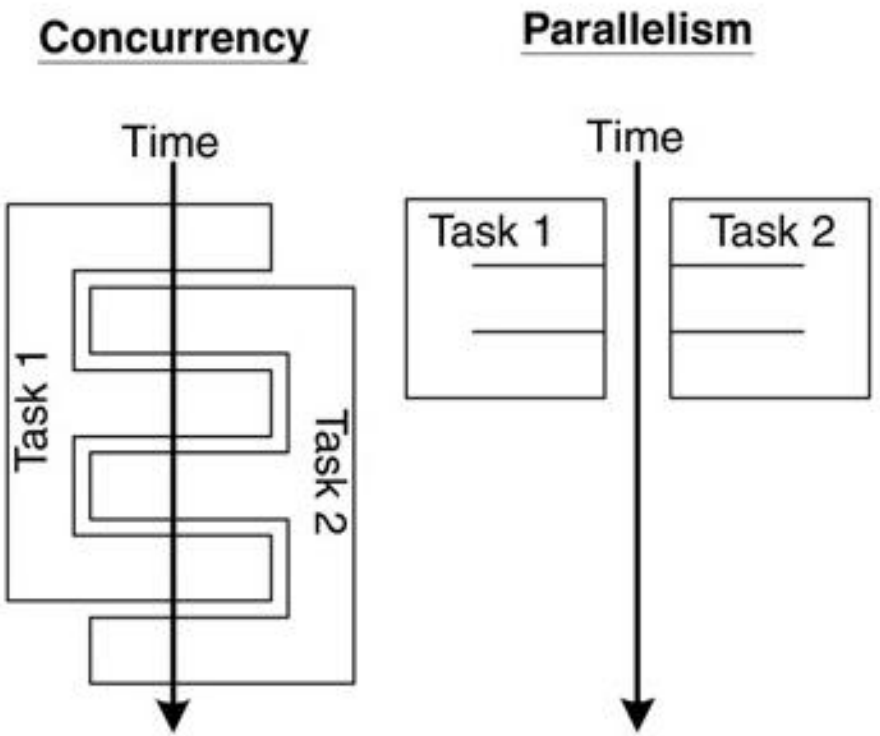
Multi-thread programming

Mutex:



Multi-thread programming

Thread Parallelism with multi-processors ou multi-core systems.



Multi-thread programming

Some thread programming libraries:

POSIX Threads (C):

FreeBSD, NetBSD, OpenBSD, Linux,

Mac OS X and Solaris

http://computing.llnl.gov/tutorials/pthreads

Windows API (C/C++/C#)– Windows X

https://msdn.microsoft.com

Boost (C++): Portable (all platforms)

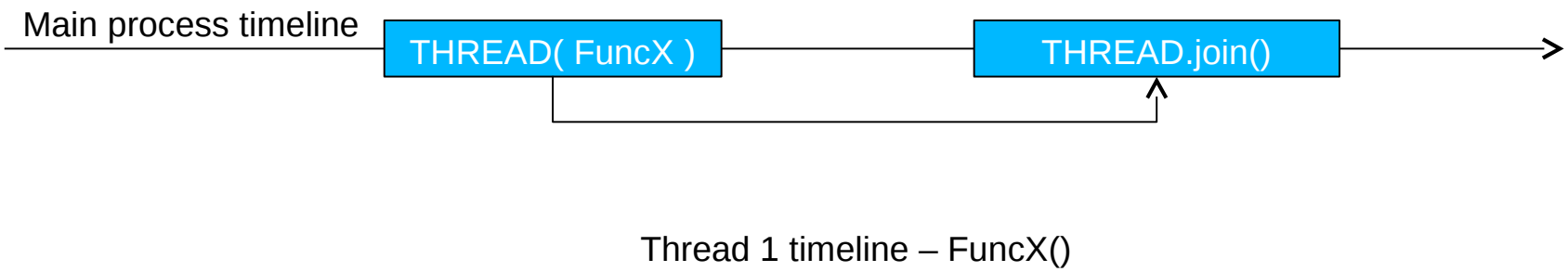
http://www.boost.org

Boost Multi-thread programming

Classes and functions for managing threads and synchronizing data between then.

The thread class: Represents a thread under the calling process or another thread context.

Create/Start, interrupt, join



Boost Multi-thread programming

The thread class

```
void main()
{
  int number = 0;

  boost::thread t1( threadFunction, &number );
  boost::thread t2( threadFunction, &number );

  [...] do some stuff [...]

  t1.join();
  t2.join();
}
```

```
void threadFunction( int* number
)
{
  int myNumber = *number;
  myNumber = myNumber + 1;
  *number = myNumber;
}
```

Boost Multi-thread programming

Better use of Threads:

- Code that can be organized discrete, independent tasks which can execute concurrently (problem partition)
- Work that can be executed, or data that can be operated on, by multiple tasks simultaneously (data partition)
- Block for potentially long I/O waits (disk, network read/write)
- Use many CPU cycles in some places but not others (CPU balance)
- Must respond to asynchronous events (user interfaces)
- Some work is more important than other work (threads priority)

Boost Multi-thread programming

A real world example: Image segmentation

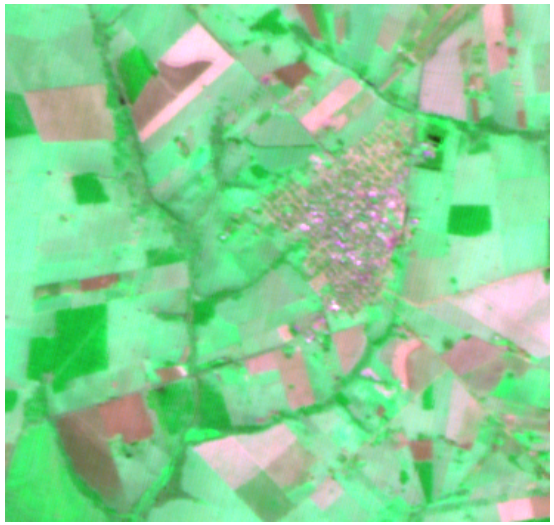
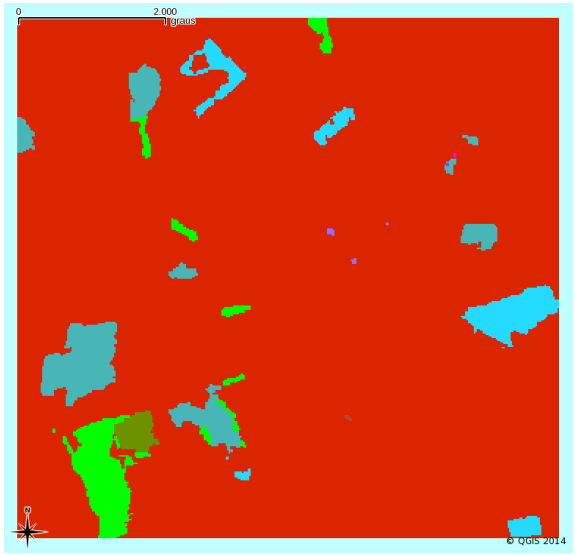


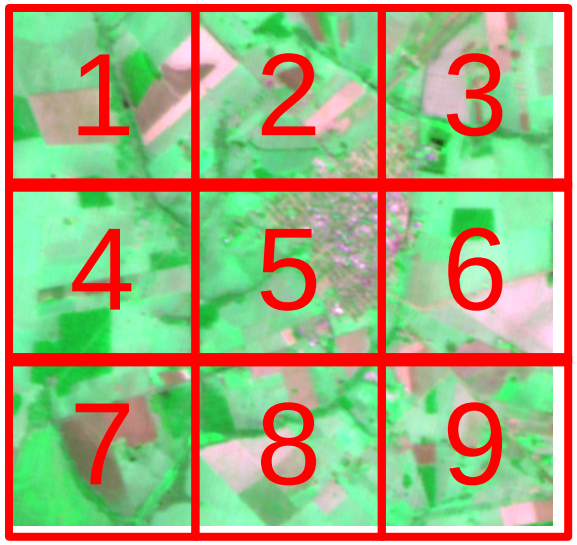
Image Segmentation



Segmentation of a typical CBERS scene: 6000 x 6000 x 5 bands floating point pixels → ~1.3GBytes

Boost Multi-thread programming

A real world example: Image segmentation



Can all blocks be processed simultaneously ? (Number of processing units and available memory).

Boost Multi-thread programming

A real world example: Image segmentation

```
void main()
{
  Image inputI;
  Image outpuI;
  int processorsNumber = 4;
  std::vector< bool > blocksStatus( 9, false );
  boost::mutex mutex;

  boost::thread_group threads;

  for( int threadIndex = 0 ; threadIndex < processorsNumber ; +
+threadIndex )
  {
    threads.add_thread( new boost::thread( segmenterThread, &inputI,
                &outputI, &blocksStatus, &mutex );
  }

  threads.join_all();
}
```

Boost Multi-thread programming

A real world example: Image segmentation

```
void segmenterThread( Image* inputI, Image* outputI,
    std::vector< bool >* blocksStatus, boost::mutex* mutex );
{
    for( int blockIndex = 0 ; blockIndex < 9 ; ++blockIndex )
    {
        mutex->lock();

        if( *blocksStatus[ blockIndex ] == false )
        {
            *blocksStatus[ blockIndex ] == true;
            ImageBlock block = inputI->loadBlock( blockIndex );
            mutex->unlock();

            [...] process block [...]

            mutex->lock();
            outputI->saveBlock( block );
            mutex->unlock();
        }
        else
        {
            mutex->unlock();
        }
    }
}
```

Questions ???