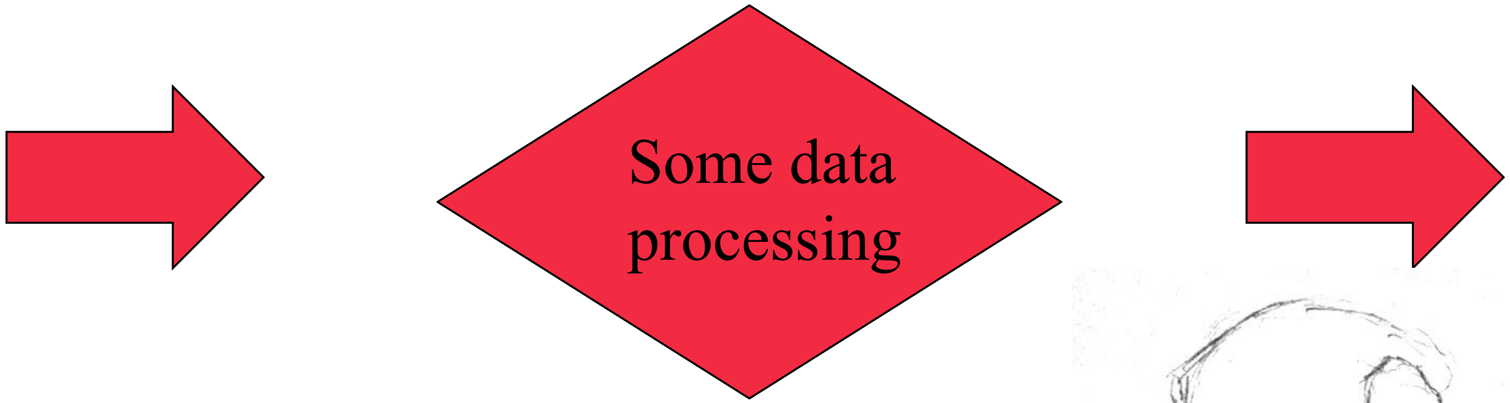
Datum: 02.05.2012

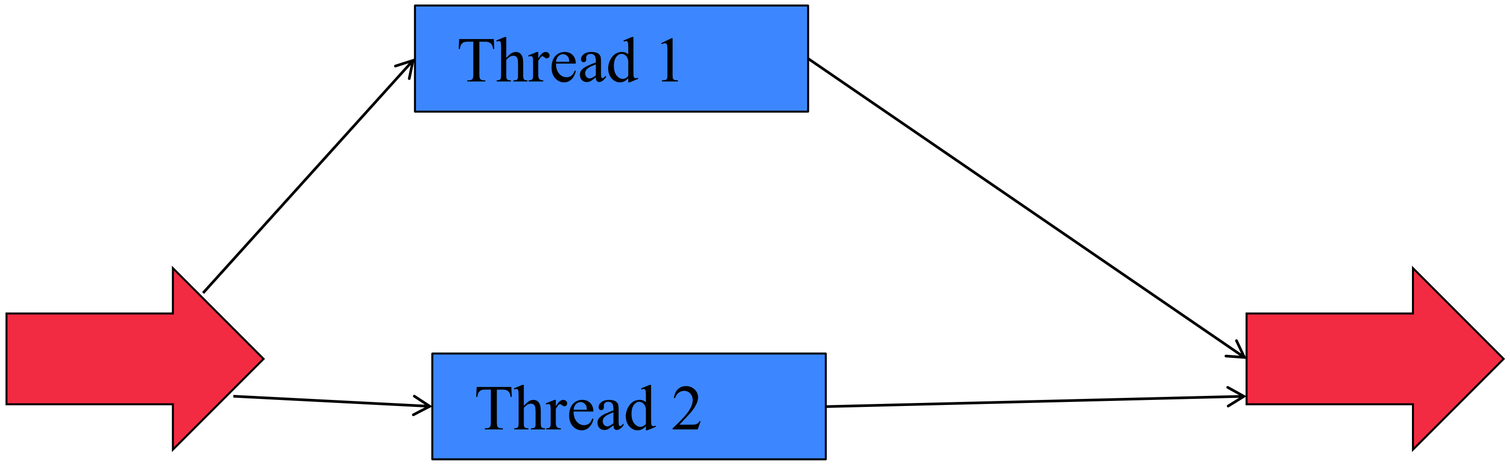
Bug - Squashing - Seminar
Threading with Barriers

Sven Mersmann

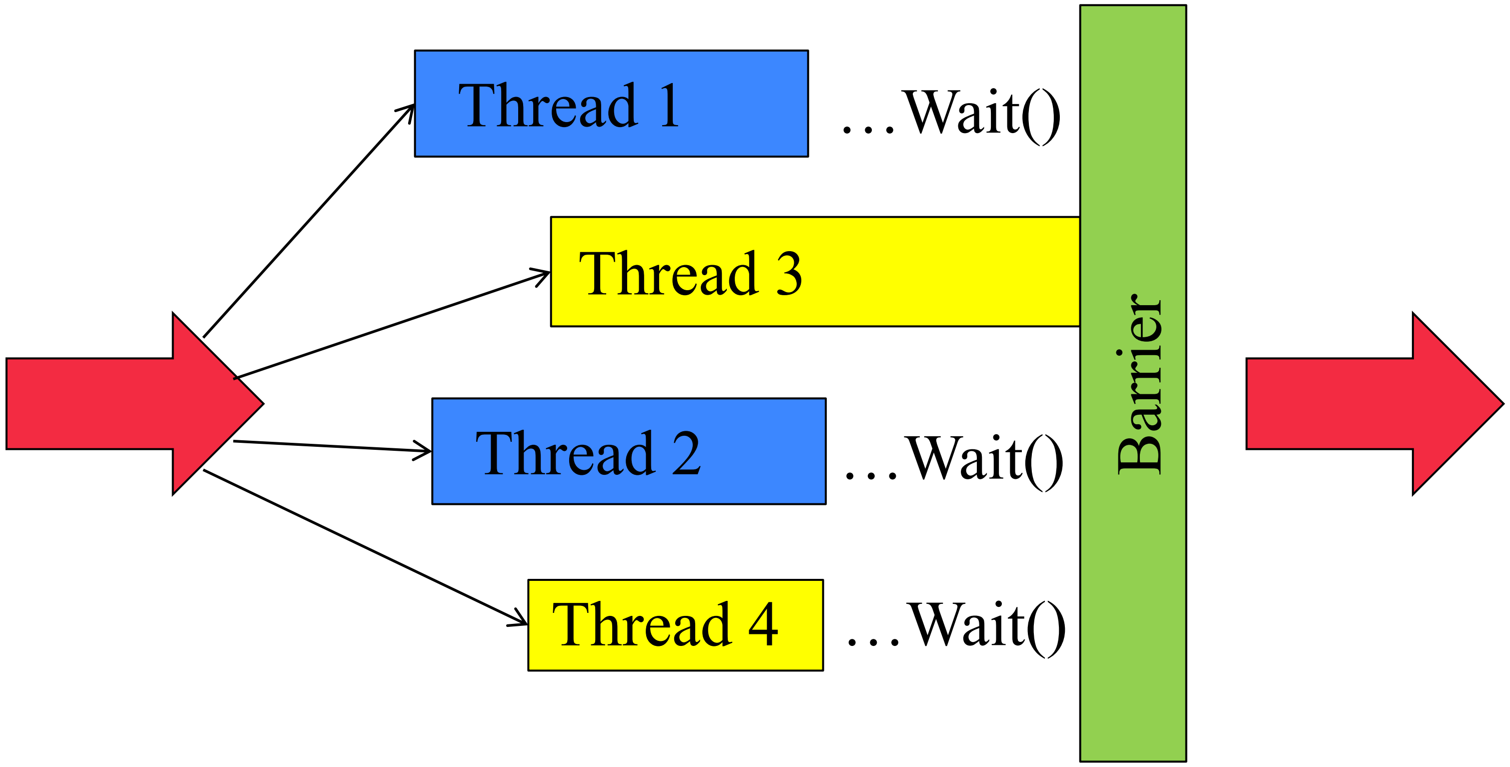
Why should I use a Barrier?



Why should I use a Barrier?

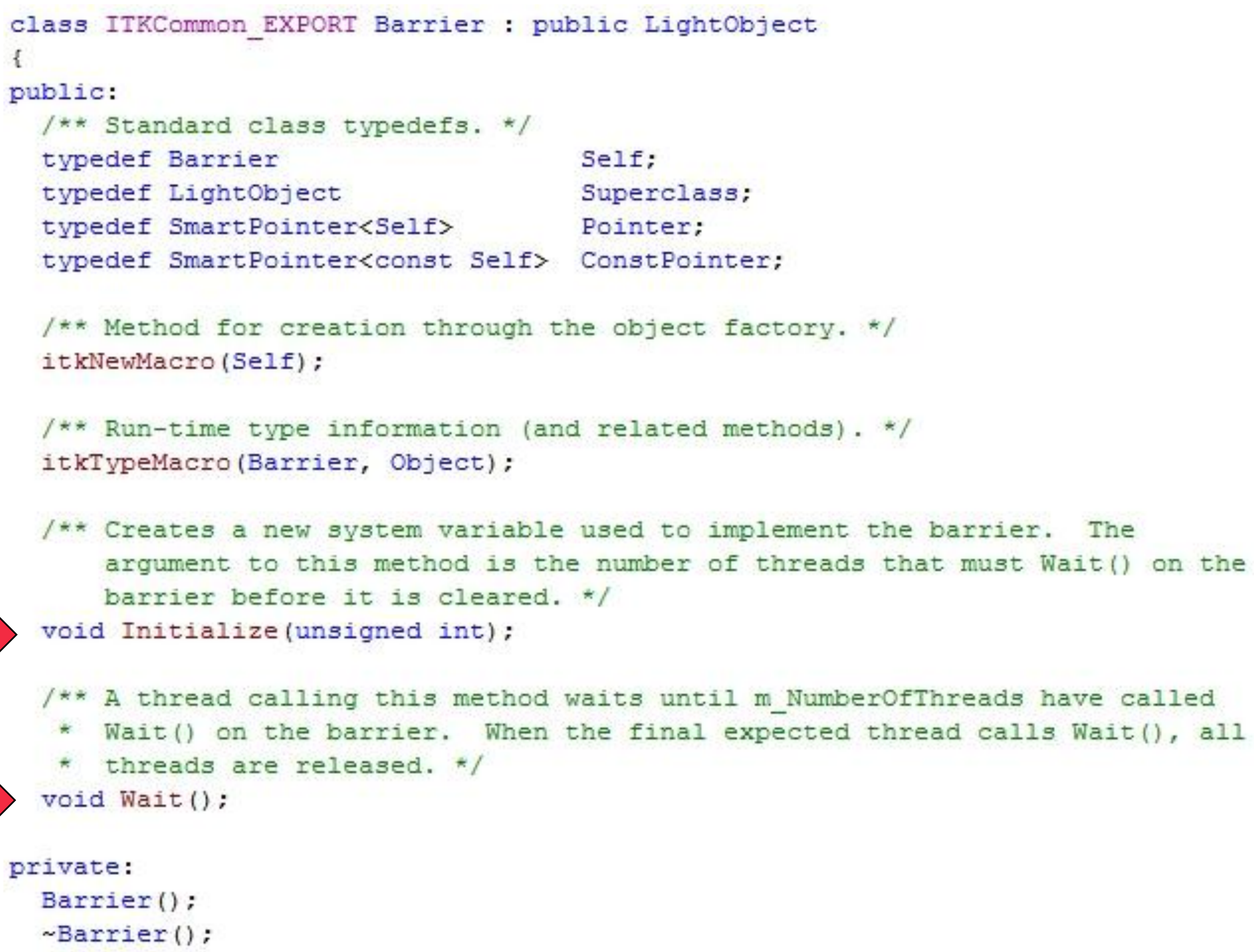


Why should I use a Barrier?



- Synchronization of threads
- Blocks thread process at a distinct position

Itk::Barrier Implementation



```
class ITKCommon_EXPORT Barrier : public LightObject
{
public:
  /** Standard class typedefs. */
  typedef Barrier                  Self;
  typedef LightObject              Superclass;
  typedef SmartPointer<Self>       Pointer;
  typedef SmartPointer<const Self> ConstPointer;

  /** Method for creation through the object factory. */
  itkNewMacro(Self);

  /** Run-time type information (and related methods). */
  itkTypeMacro(Barrier, Object);

  /** Creates a new system variable used to implement the barrier.  The
      argument to this method is the number of threads that must Wait() on the
      barrier before it is cleared. */
  void Initialize(unsigned int);

  /** A thread calling this method waits until m_NumberOfThreads have called
   *  Wait() on the barrier.  When the final expected thread calls Wait(), all
   *  threads are released. */
  void Wait();

private:
  Barrier();
  ~Barrier();
```

Itk::Barrier Implementation

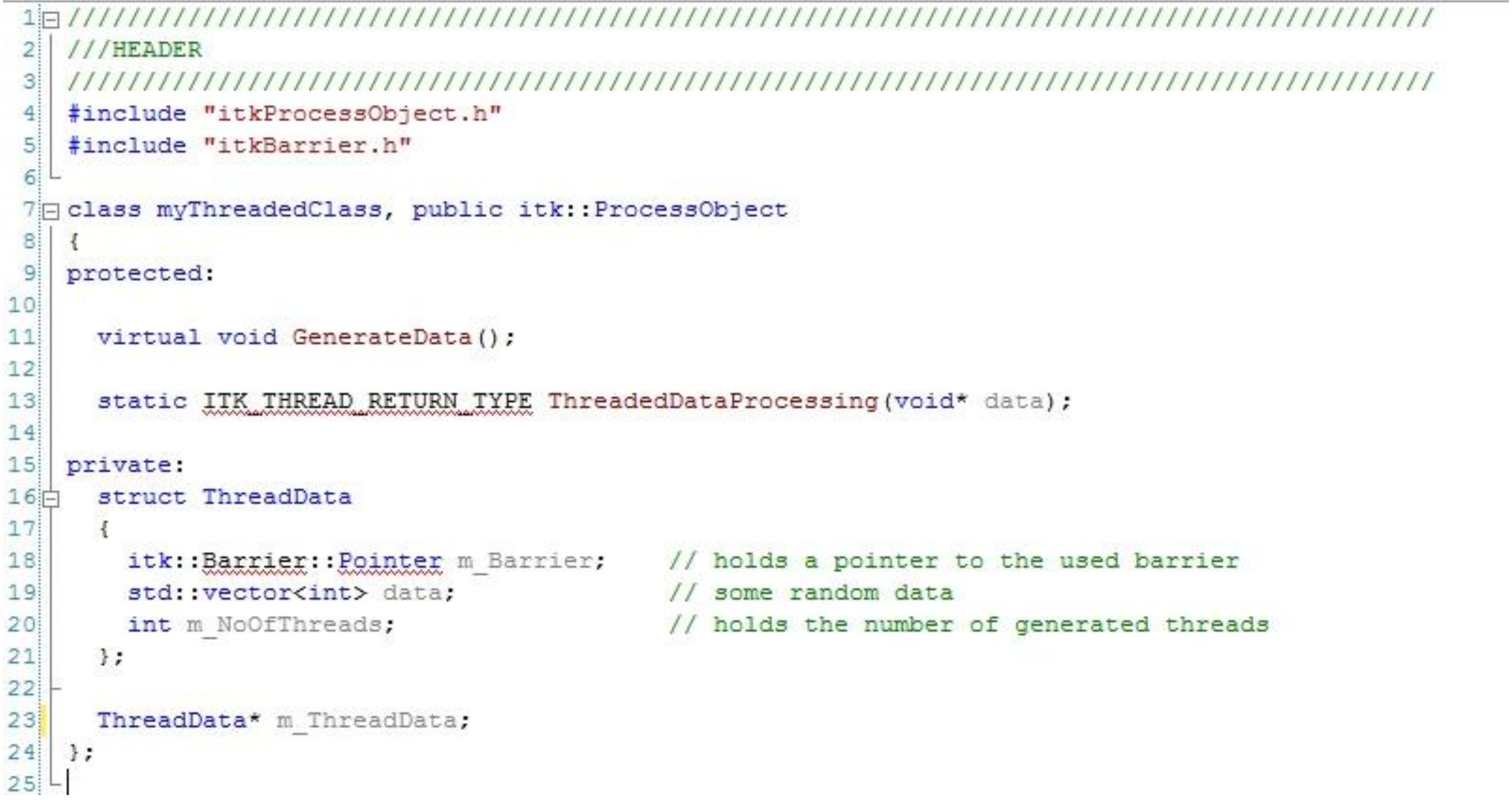
Barrier::Initialize(unsigned int)

- Set number of generated threads
- Member m_NumberExpected

Barrier::Wait()

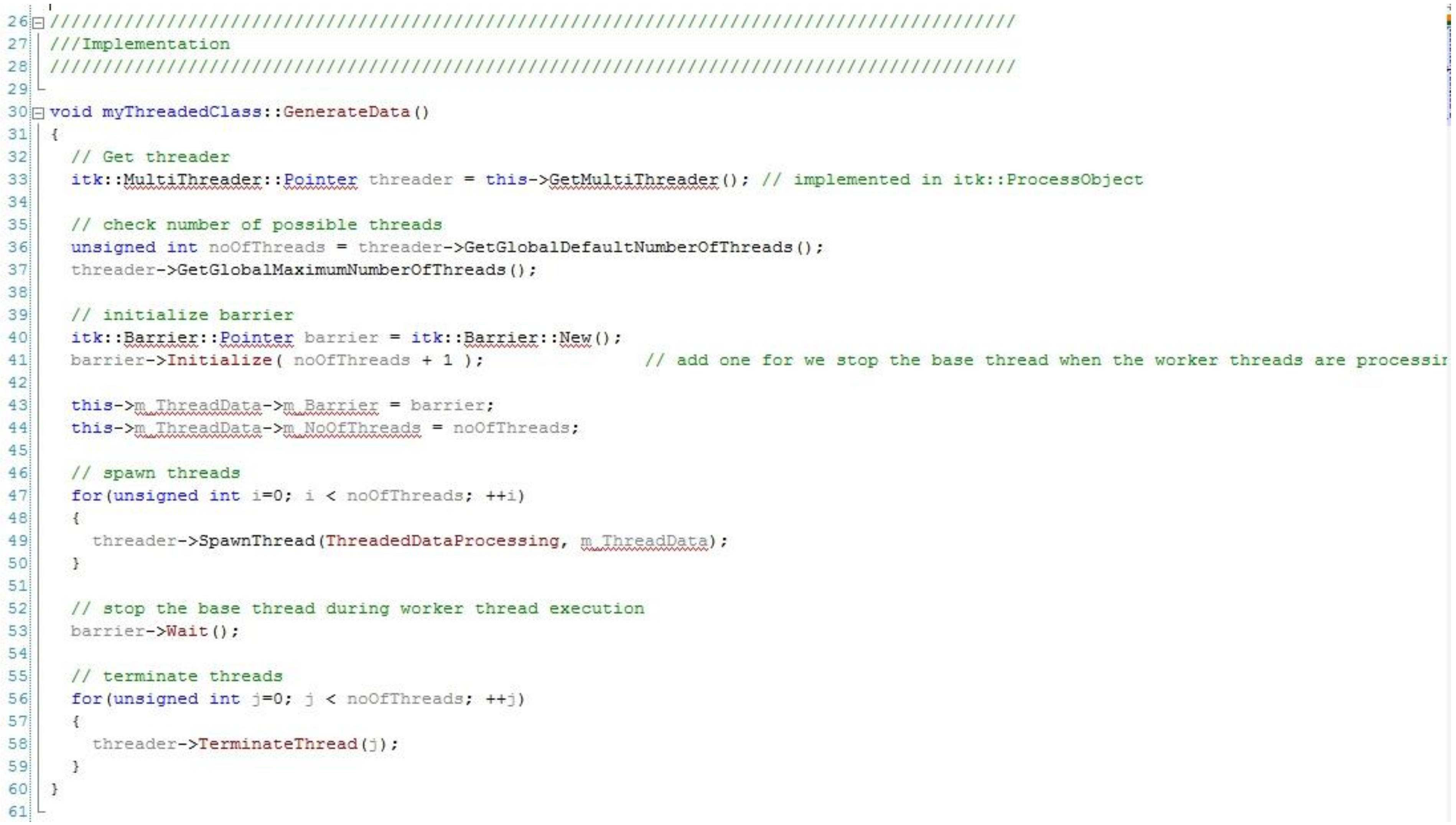
- Called when thread processing has finished
- Increase member m_NumberArrived

Threading Example using Barriers 1/3



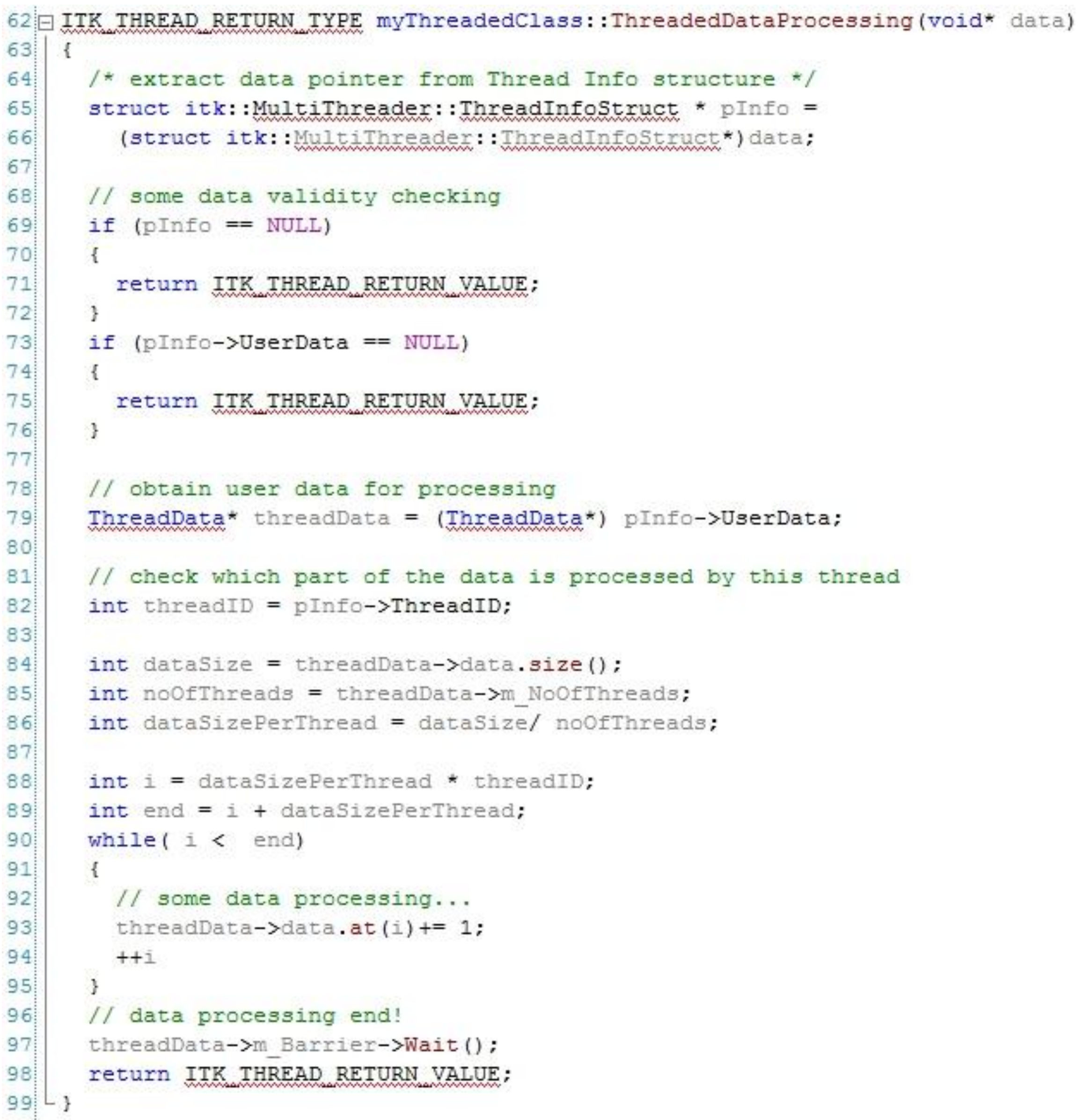
```
///////////////////////////////////////////////////////////////////////////////////////////
///HEADER
///////////////////////////////////////////////////////////////////////////////////////////
#include "itkProcessObject.h"
#include "itkBarrier.h"

class myThreadedClass, public itk::ProcessObject
{
protected:

  virtual void GenerateData();

  static ITK_THREAD_RETURN_TYPE ThreadedDataProcessing(void* data);

private:
  struct ThreadData
  {
    itk::Barrier::Pointer m_Barrier;    // holds a pointer to the used barrier
    std::vector<int> data;              // some random data
    int m_NoOfThreads;                  // holds the number of generated threads
  };

  ThreadData* m_ThreadData;
};
```

Threading Example using Barriers 2/3



```
////////////////////////////////////////////////////////////////////////////
///Implementation
////////////////////////////////////////////////////////////////////////////

void myThreadedClass::GenerateData()
{
  // Get threader
  itk::MultiThreader::Pointer threader = this->GetMultiThreader(); // implemented in itk::ProcessObject

  // check number of possible threads
  unsigned int noOfThreads = threader->GetGlobalDefaultNumberOfThreads();
  threader->GetGlobalMaximumNumberOfThreads();

  // initialize barrier
  itk::Barrier::Pointer barrier = itk::Barrier::New();
  barrier->Initialize( noOfThreads + 1 );             // add one for we stop the base thread when the worker threads are processi

  this->m_ThreadData->m_Barrier = barrier;
  this->m_ThreadData->m_NoOfThreads = noOfThreads;

  // spawn threads
  for(unsigned int i=0; i < noOfThreads; ++i)
  {
    threader->SpawnThread(ThreadedDataProcessing, m_ThreadData);
  }

  // stop the base thread during worker thread execution
  barrier->Wait();

  // terminate threads
  for(unsigned int j=0; j < noOfThreads; ++j)
  {
    threader->TerminateThread(j);
  }
}
```

Threading Example using Barriers 3/3



```
ITK_THREAD_RETURN_TYPE myThreadedClass::ThreadedDataProcessing(void* data)
{
  /* extract data pointer from Thread Info structure */
  struct itk::MultiThreader::ThreadInfoStruct * pInfo =
    (struct itk::MultiThreader::ThreadInfoStruct*)data;

  // some data validity checking
  if (pInfo == NULL)
  {
    return ITK_THREAD_RETURN_VALUE;
  }
  if (pInfo->UserData == NULL)
  {
    return ITK_THREAD_RETURN_VALUE;
  }

  // obtain user data for processing
  ThreadData* threadData = (ThreadData*) pInfo->UserData;

  // check which part of the data is processed by this thread
  int threadID = pInfo->ThreadID;

  int dataSize = threadData->data.size();
  int noOfThreads = threadData->m_NoOfThreads;
  int dataSizePerThread = dataSize/ noOfThreads;

  int i = dataSizePerThread * threadID;
  int end = i + dataSizePerThread;
  while( i <  end)
  {
    // some data processing...
    threadData->data.at(i)+= 1;
    ++i
  }
  // data processing end!
  threadData->m_Barrier->Wait();
  return ITK_THREAD_RETURN_VALUE;
}
```