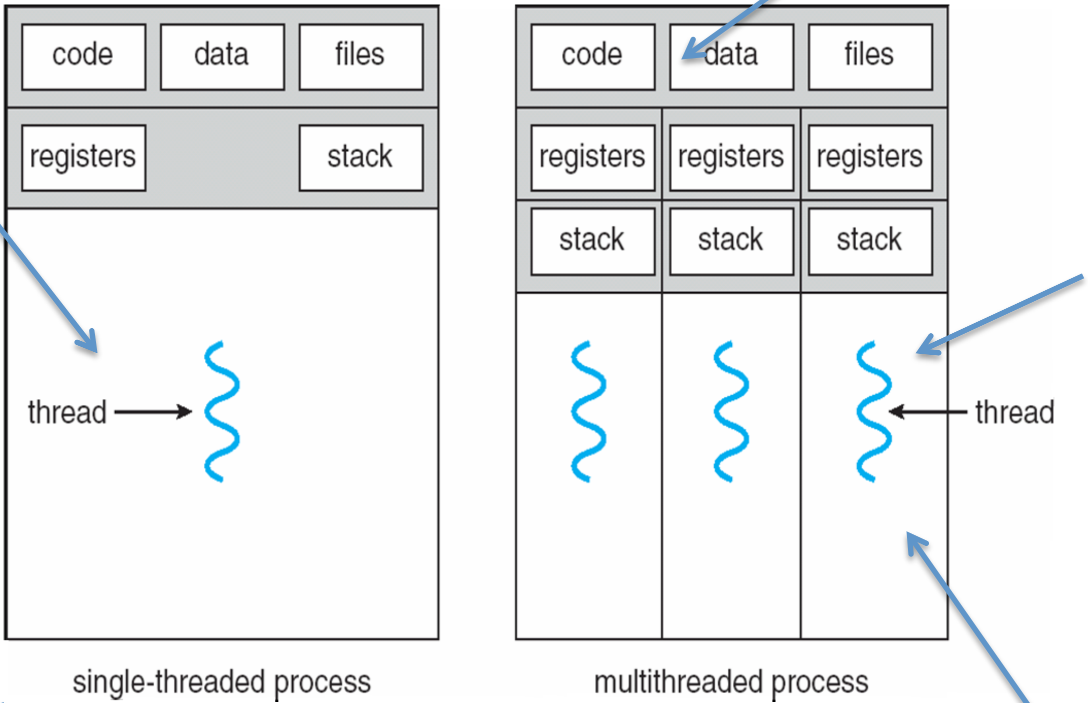
Threads and Multithreaded Programming

Davis (FALL 2015)

Processes and Threads

A **thread** is just a sequence of instructions to execute

Threads share the same memory space as other **threads** in the same application



Idea is that these **threads** run approximately in parallel to speed up execution.

A **process** is an instance of a computer program

Threads also automatically share data and variables.

Example 4 Threads 1 CPU Core

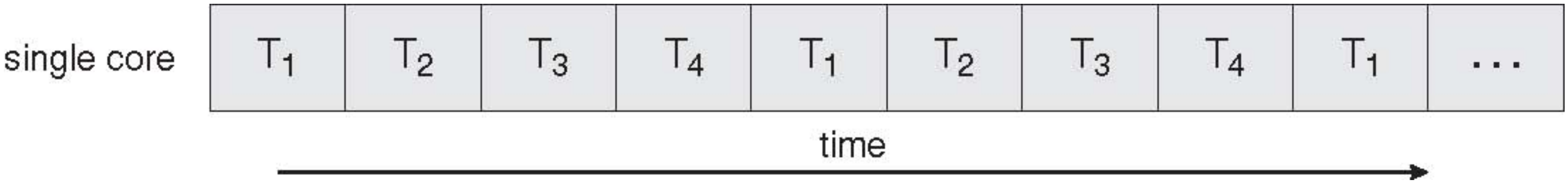
T1 T2 T3 T4

T1	T2	T3	T4
Maybe this is a function that calculates the Nth number in the Fibonacci sequence	*Maybe this is a function that calculates the Nth factorial*	*Maybe this is a function that calculates pi to the Nth decimal point*	*Maybe this calculates the golden ratio as the ratio of the Nth term to (N-1)th term in Fibonacci Sequence*

or

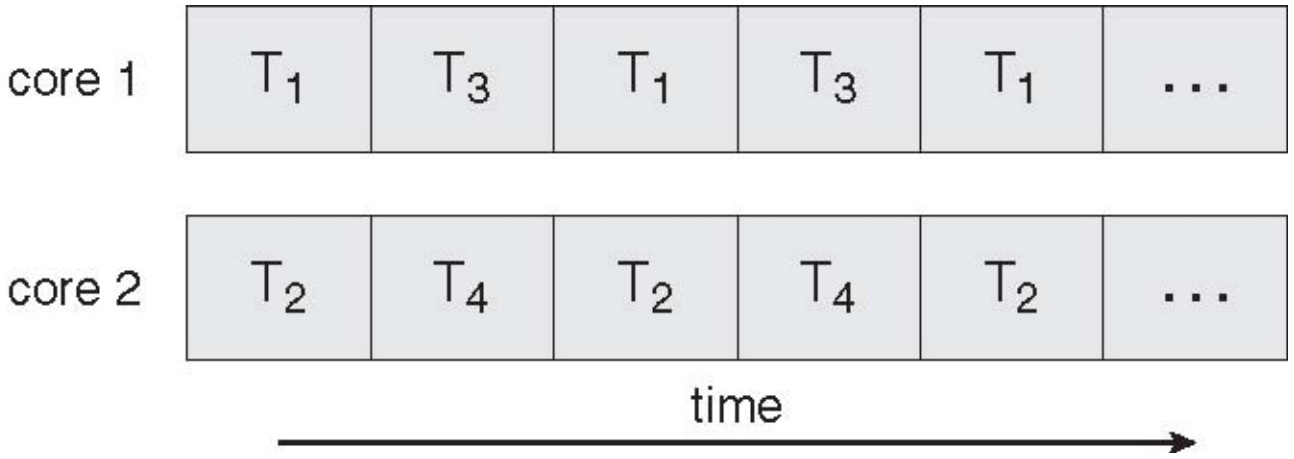
T1	T2	T3	T4
Monitors the keyboard input and stores each word in a vector array of strings	*This monitors the time and prints out the new time every minute*	*Maybe this looks at each string inputted by the user and checks to see if "exit" string appears*	*Maybe this calculates the histogram of words times and puts this is in a data structure*

Concurrent Execution on a Single-Core System



OS can time slice between the four Threads T1…T4

Parallel Execution on a Multi-core System



core 1	T_1	T_3	T_1	T_3	T_1	. . .
core 2	T_2	T_4	T_2	T_4	T_2	. . .

OS can time slice the four Threads T1…T4 on two processor cores. Two threads can run in parallel on different cores. Application could run up to twice as fast.

How can we specify threads in c++?

STL threads (c++11 Version)

The 2011 standard for C++ includes a new standard template library for multithreaded programming.

pThreads

A POSIX standard (IEEE 1003.1c) API for thread creation and synchronization, which is popular on UNIX systems

gThreads

This is a wrapper around pThreads that is specific to Georgia Tech and will be used in your last lab

Let’s start with an example in STL threads library

single threaded process

```
#include <iostream>
#include <thread>
using namespace std;

int main()
{
  cout << “Hello single threaded process” ;
}
```

2-threaded process (trivial)

```
#include <iostream>
#include <thread>
using namespace std;

void hello()
{
   cout <<"Hello 2-threaded process \n”;
}

int main()
{
  thread t(hello);
  t.join();
}
```

Let’s start with an example in STL threads library

single threaded process

```
#include <iostream>
#include <thread>
using namespace std;

void hello()
{
cout <<"Hello Concurrent World" << endl;
}

int main()
{
  cout << “Hello single threaded process” ;
}
```

single main thread

Let’s start with an example in STL threads library

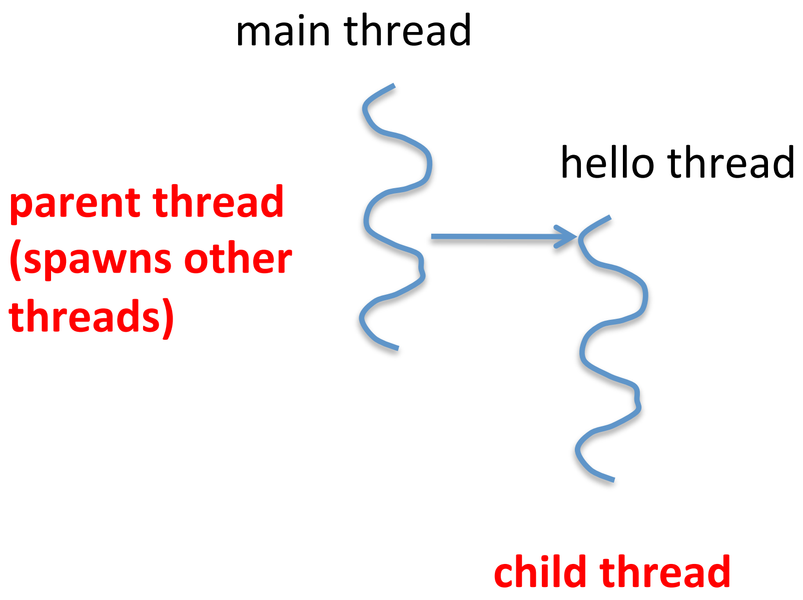
2-threaded process (trivial)

```
#include <iostream>
#include <thread>
using namespace std;

void hello()
{
   cout <<"Hello 2-threaded process \n”;
}

int main()
{
  thread t(hello);
  t.join();
}
```

This creates a thread object!

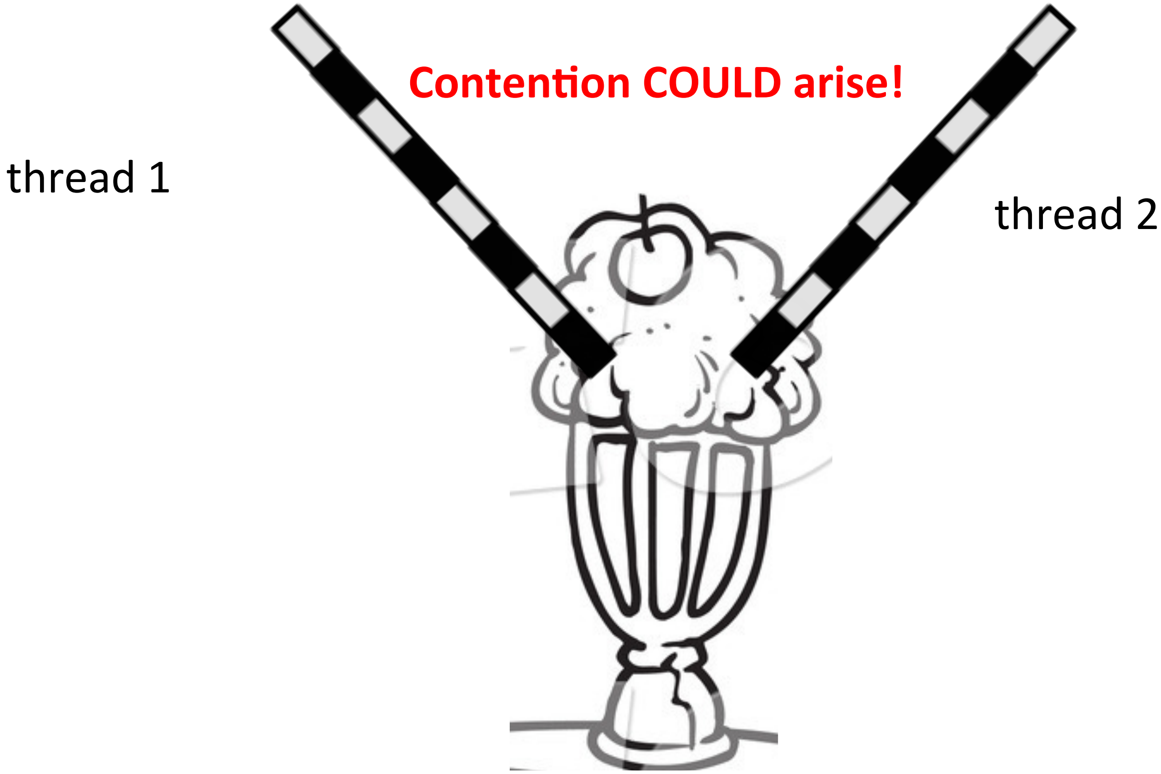
try to take out?

t.join() waits until thread ‘t’ is done!



IF PARENT IS TERMINATED THEN ALL CHILDREN ARE AUTOMATICALLY KILLED!

Big Issue: What if two threads try to use the same resource at the same time!



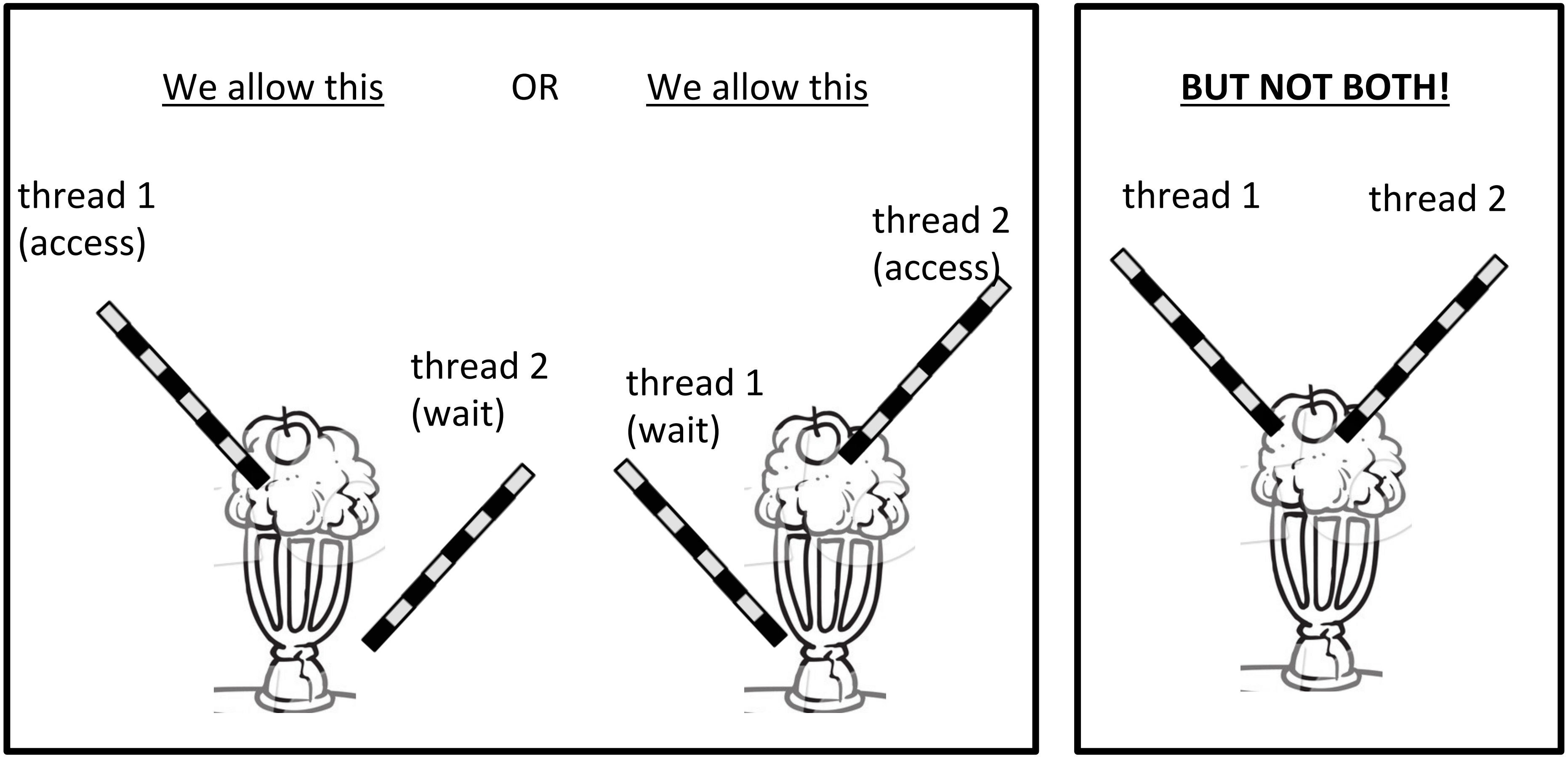
Contention COULD arise!

thread 1

thread 2

resource (perhaps shared memory)

We need a way to specify that two actions are MUTUAL EXCLUSIVE (MutEx)!



We allow this OR We allow this

thread 1 (access)

thread 2 (wait)

thread 1 (wait)

thread 2 (access)

BUT NOT BOTH!

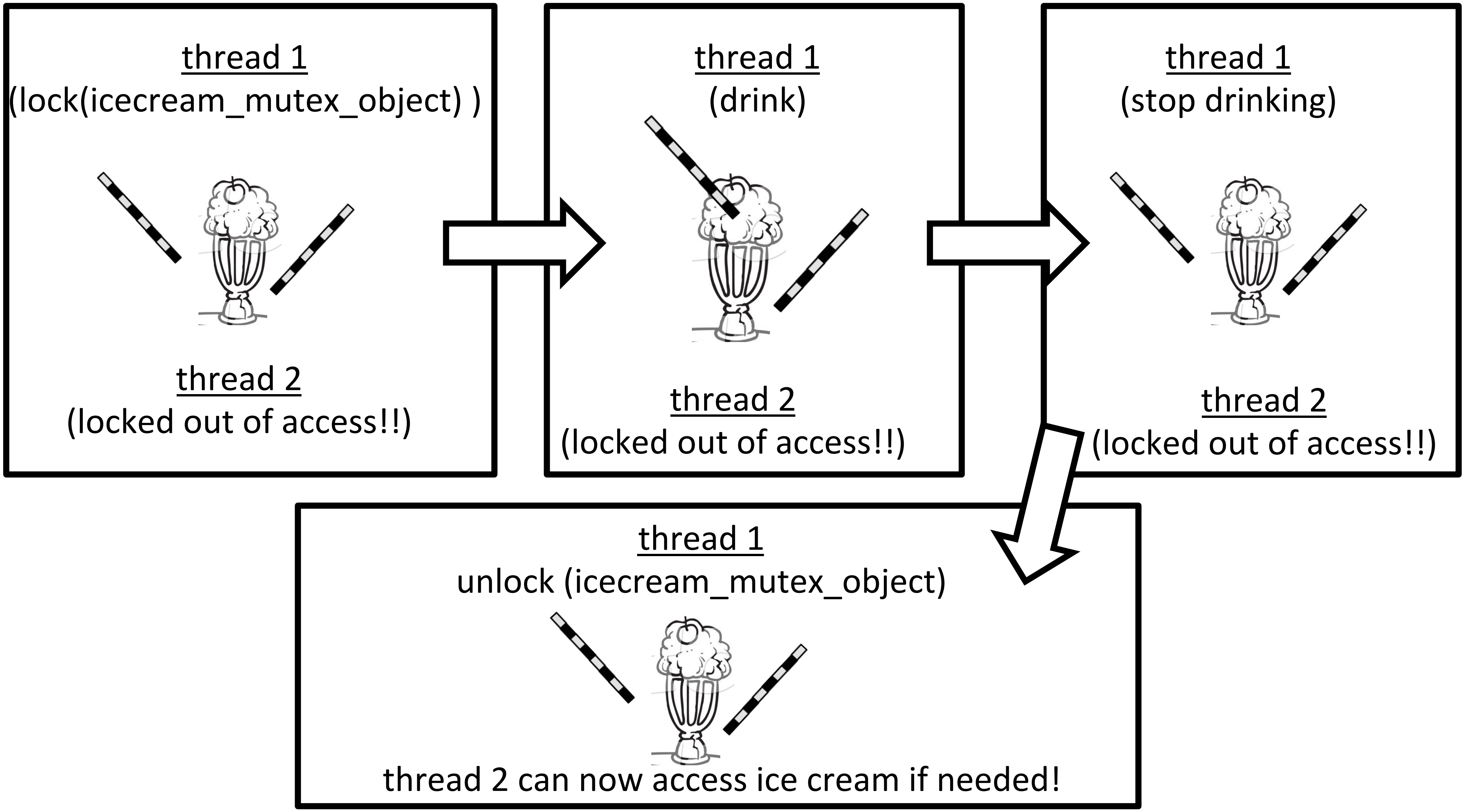
thread 1

thread 2

We would define these two accesses as mutual exclusive!

We can define a MutEx Object that we lock and unlock!

first instantiate an icecream_mutex_object!



thread 1
(lock(icecream_mutex_object))

thread 2
(locked out of access!!)

thread 1
(drink)

thread 2
(locked out of access!!)

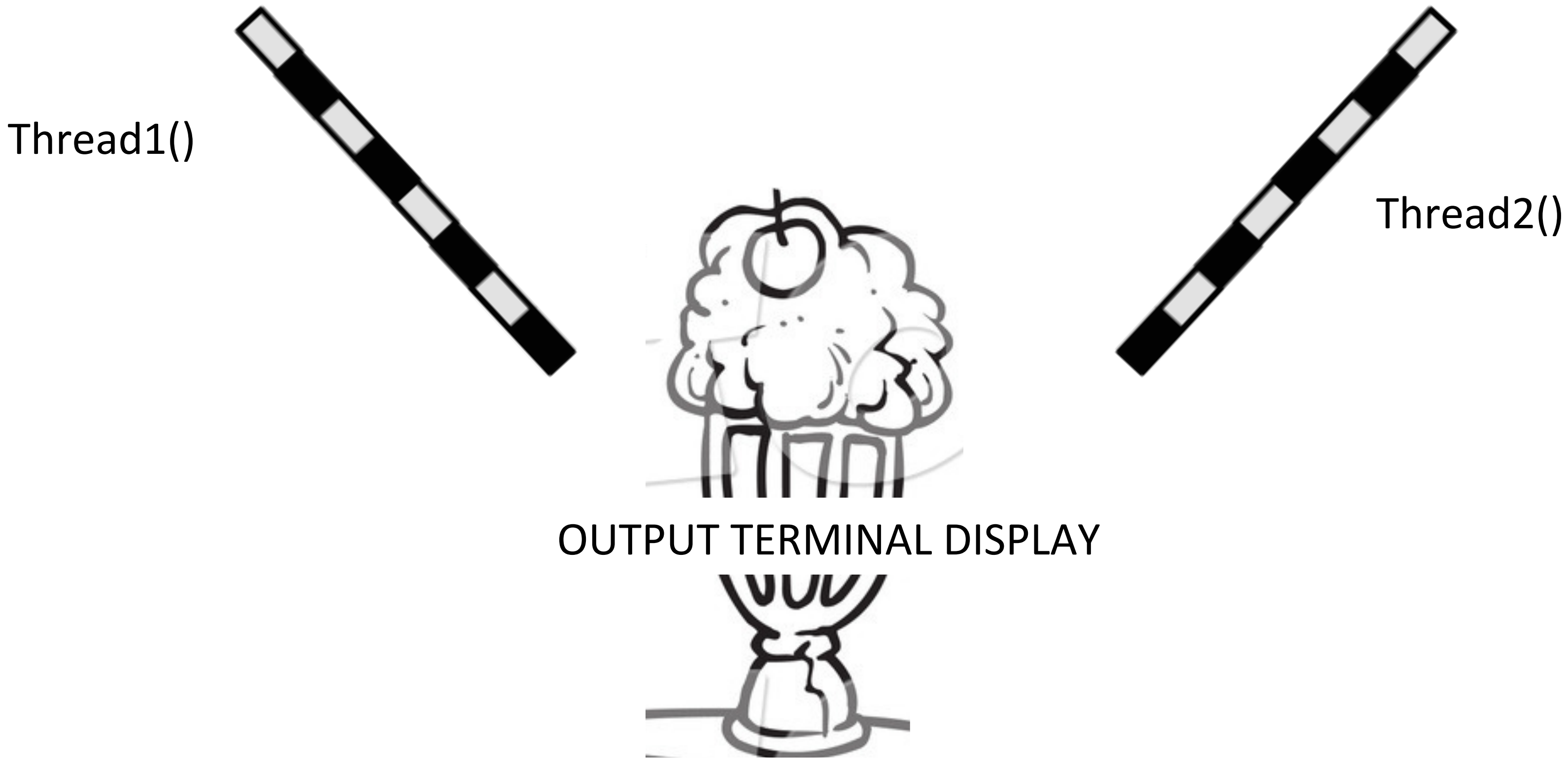
thread 1
(stop drinking)

thread 2
(locked out of access!!)

thread 1
unlock (icecream_mutex_object)

thread 2 can now access ice cream if needed!

Let’s look at a code example…



Thread1()

Thread2()

OUTPUT TERMINAL DISPLAY

We will spawn 2 threads!

→

We will use a mutex lock to make sure that NO two threads use output terminal at the sames time!

```
#include <iostream>
#include <mutex>
#include <vector>
#include <thread>
#include <stdlib.h>

using namespace std;

mutex coutMutex;

void Thread1()
{
  for (int i=0; i < 10; ++i)
  {
    coutMutex.lock();
    cout << "Hello from thread 1"<<endl;
    coutMutex.unlock();
    for (int j = 0; j < 10; ++j)
      { //consume time to slow this
function down
      }
  }
}
```

```
void Thread2()
{
  for (int i=0; i < 10; ++i)
  {
    coutMutex.lock();
    cout << "Hello from thread 2"<<endl;
    coutMutex.unlock();
    for (int j = 0; j < 10; ++j)
      { //consume time to slow this function
down
      }
  }
}

 int main()
 { thread t1(Thread1);
   thread t2(Thread2);

   t1.join();
   t2.join();

   cout << "Main exiting now" << endl;
 }
```

Run Code with Variations

- Take out the mutex lock and unlock and run program
- Make Thread1 run much slower than Thread2
- Make Thread2 run much slower than Thread1
- Put mutex locking and unlocking back in and run code again.

gthread wrapper

- Disclaimer – gthreads library is not part of a standard library!!!

- Definitions are in gthreads.h
- The gthreads library is a significantly reduced functionality– but sufficient for our next lab

Creating a Thread in GThreads

- CreateThread(function) – This spawns a separate thread of execution
- You can have a function with up to 4 arguments that you can start as a thread
  - These are passed by value to the thread
  - They must match the type in the function

Example on next page

Hello Concurrent World

```
#include <iostream>
#include "gthread.h"

using namespace std;

void hello()
{
   cout << "Hello Concurrent World!" << endl;
   EndThread();
}

int main()
{
CreateThread(hello);
WaitAllThreads();
}//end main
```

```
//Demonstration of gThreads.cc

#include <iostream>
#include "gthread.h"
#include <stdlib.h>

using namespace std;

//Gthread Mutexes MUST be in global memory
gthread_mutex_t coutMutex;

void MyThreadThreeArgs(int myId, int count1, int
count2)
{
  for (int i=0; i < count1; ++i)
  {
    LockMutex(coutMutex);
    cout << "Hello from thread " << myId << " count1
" << i << endl;
    UnlockMutex(coutMutex);
    for (int j = 0; j < count2; ++j)
      {
      }
  }
  EndThread(); // Required by GThreads library
}
```

```
int main(int argc, char **argv)
{

  if (argc < 4)
  {
    cout << "Usage: need nThreads count1 count2"
<< endl;
    exit(1);
  }

  int nThreads = atol(argv[1]);
  int count1 = atol(argv[2]);
  int count2 = atol(argv[3]);

  for (int i = 0; i < nThreads; i++)
  {  //Start each thread
    CreateThread(MyThreadThreeArgs, i, count1,
count2);
  }

  //Now wait for all to complete
  WaitAllThreads();
  cout << "Main exiting now" << endl;
}
```

Ending a Thread in GThread

- EndThread() – This must be be called by each thread (except main thread)
  - This will decrement a counter keeping track of threads
  - If this internal counter is zero then it notifies the parent thread that all its children have finish

Example on next page

Hello Concurrent World

```
#include <iostream>
#include "gthread.h"

using namespace std;

void hello()
{
   cout << "Hello Concurrent World!" << endl;
   EndThread();
}

int main()
{
CreateThread(hello);
WaitAllThreads();
}//end main
```

```
//Demonstration of gThreads.cc

#include <iostream>
#include "gthread.h"
#include <stdlib.h>

using namespace std;

//Gthread Mutexes MUST be in global memory
gthread_mutex_t coutMutex;

void MyThreadThreeArgs(int myId, int count1, int
count2)
{
  for (int i=0; i < count1; ++i)
  {
    LockMutex(coutMutex);
    cout << "Hello from thread " << myId << " count1
" << i << endl;
    UnlockMutex(coutMutex);
    for (int j = 0; j < count2; ++j)
      {
      }
  }
  EndThread(); // Required by GThreads library
}
```

```
int main(int argc, char **argv)
{

  if (argc < 4)
  {
    cout << "Usage: need nThreads count1 count2"
<< endl;
    exit(1);
  }

  int nThreads = atol(argv[1]);
  int count1 = atol(argv[2]);
  int count2 = atol(argv[3]);

  for (int i = 0; i < nThreads; i++)
  {  //Start each thread
    CreateThread(MyThreadThreeArgs, i, count1,
count2);
  }

  //Now wait for all to complete
  WaitAllThreads();
  cout << "Main exiting now" << endl;
}
```

Synchronizing Threads in GThread

- WaitAllThreads() – This must be be called by each parent thread!
  - The parent thread will wait until all is children are done
  - The CPU is not assigned to the main thread until this is done
- When WaitAllThreads() returns parent thread will continue to run

↓

Example on next page

Hello Concurrent World

```
#include <iostream>
#include "gthread.h"

using namespace std;

void hello()
{
  cout << "Hello Concurrent World!" << endl;
  EndThread();
}

int main()
{
CreateThread(hello);
WaitAllThreads();
}//end main
```

```
//Demonstration of gThreads.cc

#include <iostream>
#include "gthread.h"
#include <stdlib.h>

using namespace std;

//Gthread Mutexes MUST be in global memory
gthread_mutex_t coutMutex;

void MyThreadThreeArgs(int myId, int count1, int
count2)
{
  for (int i=0; i < count1; ++i)
  {
    LockMutex(coutMutex);
    cout << "Hello from thread " << myId << " count1
" << i << endl;
    UnlockMutex(coutMutex);
    for (int j = 0; j < count2; ++j)
      {
      }
  }
  EndThread(); // Required by GThreads library
}
```

```
int main(int argc, char **argv)
{

  if (argc < 4)
  {
    cout << "Usage: need nThreads count1 count2"
<< endl;
    exit(1);
  }

  int nThreads = atol(argv[1]);
  int count1 = atol(argv[2]);
  int count2 = atol(argv[3]);

  for (int i = 0; i < nThreads; i++)
  {  //Start each thread
    CreateThread(MyThreadThreeArgs, i, count1,
count2);
  }

  //Now wait for all to complete
  WaitAllThreads();
  cout << "Main exiting now" << endl;
}
```

Mutex Objects in GThread

- GThread mutex definitions must be global
- LockMutex(mutex_object) is a global function
- UnlockMutex(mutex_object) is a global function

↓

Example on next page

```
//Demonstration of gThreads.cc

#include <iostream>
#include "gthread.h"
#include <stdlib.h>

using namespace std;

//Gthread Mutexes MUST be in global memory
gthread_mutex_t coutMutex;

void MyThreadThreeArgs(int myId, int count1, int
count2)
{
  for (int i=0; i < count1; ++i)
  {
    LockMutex(coutMutex);
    cout << "Hello from thread " << myId << " count1
" << i << endl;
    UnlockMutex(coutMutex);
    for (int j = 0; j < count2; ++j)
      {
      }
  }
  EndThread(); // Required by GThreads library
}
```

```
int main(int argc, char **argv)
{

  if (argc < 4)
  {
    cout << "Usage: need nThreads count1 count2"
<< endl;
    exit(1);
  }

  int nThreads = atol(argv[1]);
  int count1 = atol(argv[2]);
  int count2 = atol(argv[3]);

  for (int i = 0; i < nThreads; i++)
  {  //Start each thread
    CreateThread(MyThreadThreeArgs, i, count1,
count2);
  }

  //Now wait for all to complete
  WaitAllThreads();
  cout << "Main exiting now" << endl;
}
```