Deadlock

   A deadlock is a state, in which at least two threads are blocked, because each thread is waiting for release of some resource with which other thread works, before it releases its own resource.

The result of a deadlock is total standstill. The Thread and usually the whole program is **blocked forever**. It is easy to produce a deadlock. Curious?

### Exceptions and unknown code

std::mutex m;

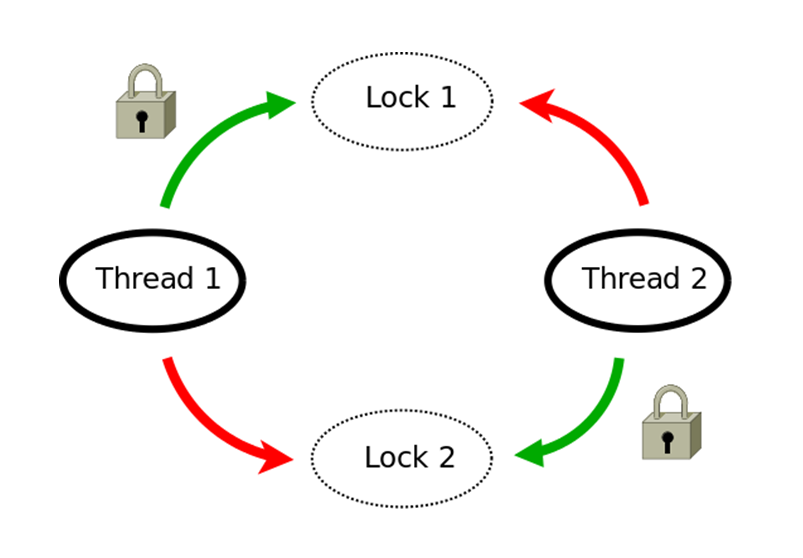
m.lock();

sharedVariable= getVar();

m.unlock();

In case the unknown code in the function getVar() throws an exception, m.unlock() will not be called. Every attempt to to ask for the mutex m will fail and the program will block. Forever. But that is not the only issue with that piece of code. It calls some (unknown to us) function get.Var(), while m.lock() is active. What will happen if the function getVar() tries to get the same lock? Of course, you know it. A deadlock.

### Lock mutexes in different order



Thread 1 and Thread 2 need access to two resources in order to finish their work. Unfortunately, they ask for the resources which are protected by two mutexes in different order. In this case the thread executions will interleave in such a way that thread 1 gets mutex 1, then thread 2 gets mutex 2, and we have a standstill. Each thread wants to get the other's mutex. For this, thread has to wait for the release of the resource.

// deadlock.cpp

#include <iostream>

#include <chrono>

#include <mutex>

#include <thread>

struct CriticalData{

std::mutex mut;

};

void deadLock(CriticalData& a, CriticalData& b){

a.mut.lock();

std::cout << "get the first mutex" << std::endl;

std::this\_thread::sleep\_for(std::chrono::milliseconds(1));

b.mut.lock();

std::cout << "get the second mutex" << std::endl;

// do something with a and b

a.mut.unlock();

b.mut.unlock();

}

int main(){

CriticalData c1;

CriticalData c2;

std::thread t1([&]{deadLock(c1,c2);});

std::thread t2([&]{deadLock(c2,c1);});

t1.join();

t2.join();

}