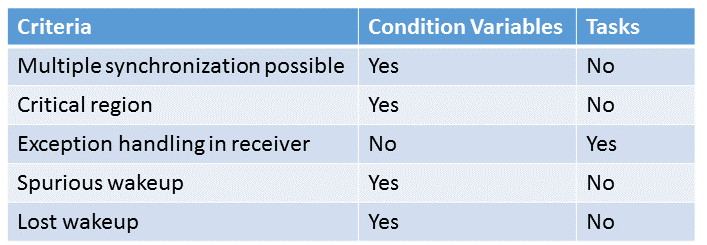
In case you are using  promise and future to synchronize threads, they have a lot in common with [condition variables](http://modernescpp.com/index.php/tag/condition-variable). But most of the time, task are the better choice.

Synchronization of threads

To get the big picture, get the facts. The table compares condition variables with tasks (promise and future).



The benefit of a condition variable to a promise and future is, that you can use condition variables to synchronize threads multiple times. In opposite to that, a promise can send its notification only once. So you have to use more promise and future pairs to get the functionality of a condition variable. But in case you use the condition variable only for one synchronization, the condition variable is a lot more difficult to use in right way. So a promise and future pair needs no shared variable and therefore no lock, they are not prone to [spurious wakeups](http://modernescpp.com/index.php/condition-variables) or [lost wakeups](http://modernescpp.com/index.php/condition-variables). In addition to that, they can handle exceptions. **So there are lot of reasons to prefer tasks to condition variables.**

How can I use tasks to synchronize threads?

// promiseFutureSynchronize.cpp

#include <future>

#include <iostream>

#include <utility>

void doTheWork(){

std::cout << "Processing shared data." << std::endl;

}

void waitingForWork(std::future<void>&& fut){

std::cout << "Worker: Waiting for work." << std::endl;

fut.wait();

doTheWork();

std::cout << "Work done." << std::endl;

}

void setDataReady(std::promise<void>&& prom){

std::cout << "Sender: Data is ready." << std::endl;

prom.set\_value();

}

int main(){

std::cout << std::endl;

std::promise<void> sendReady;

auto fut= sendReady.get\_future();

std::thread t1(waitingForWork,std::move(fut));

std::thread t2(setDataReady,std::move(sendReady));

t1.join();

t2.join();

std::cout << std::endl;

}

Quite easy.

With the help of the promise sendReady (line32) I get the future fut (line34). The promise signalises in this case by his return value void (std::promise<void> sendReady), that it is only capable of sending notifications. Both communications endpoints are moved into the thread t1 respectively t2 (line 35 and 36). The future is waiting with its call fut.wait() (line 15) for the notification of the promise: prom.set\_value() (line 24).

As well the structure as the output of the program matches the program in den post about the [condition variable](http://modernescpp.com/index.php/tag/condition-variable).