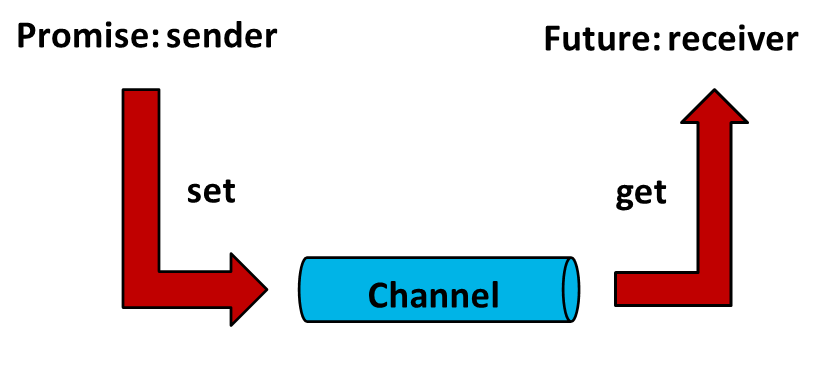
Tasks as data channels



Tasks behave like data channels. On one side, the sender sets a value. On the other side, the receiver picks up the value. The sender is called promise, the receiver - future. Or to say it in different words, the sender promises to provide a value, which the receiver can pick up in the future.

A few more details. The sender can provide the value for more than one future. Beside a value, the sender can also provide a notification or an exception. The get call of the future blocks. It means, in case the future calls wait, it must wait until the promise puts the value into the channel.

Tasks are available in three variations. As asynchronous function call with std::async, as simple wrapper for a callable with std::packaged\_task and as the explicit pair std::promise and std::future.

The best way to get the differences between threads and tasks is to compare them.

## Threads versus Tasks

This small code example illustrates the difference:

int res;

std::thread t([&]{res= 3+4;});

t.join();

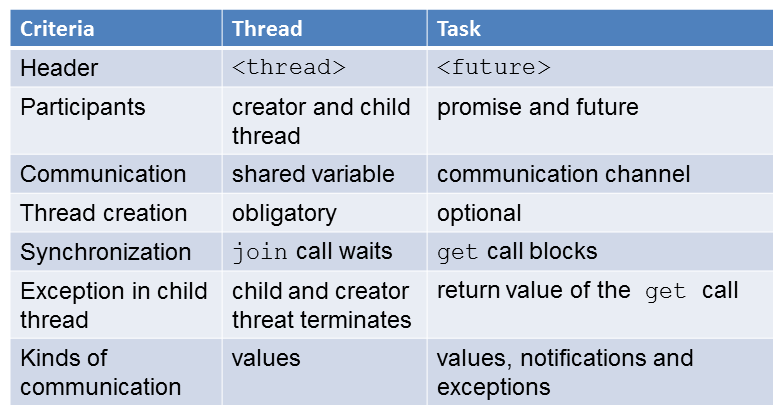
std::cout << res << std::endl;

auto fut=std::async([]{return 3+4;});

std::cout << fut.get() << std::endl;

Both the child thread and the promise calculate the sum of 3+4 and return the result. The std::async call generates a data channel with both endpoints fut and std::async. fut is a future, std::async  is a promise. The future gets the value with the call fut.get(). This value is provided by the promise. The future can act at a later point in time.

What are the differences?



The thread needs the <thread> header, the task  needs the <future> header. The participants of the threads are the creator thread and the child thread, the participant of the task are the promise and the future. The shared variable res is the child's way to transfer the calculation result to the creator. In opposite, promise and future use a common data channel,  std::async creates the data channel.Using fut.get the future gets the result. Using  threads you have to protect the shared variable with a lock. But there is implicitly no possibility of a [race condition](http://modernescpp.com/index.php/component/content/article?id=157:threads-sharing-%20%20data&catid=35:c&Itemid=239#RaceCondition)for the promise and the future. The creator of the threads waits with its t.join call, until its child is done. On the other side, the fut.get call blocks. In case there is an exception in the child thread, the child thread and the creator thread terminate. So at the end the whole program terminates. The promise can deliver an exception to the future. The future has to handle the exception. While the child thread is only able to provide values for the creator thread, the promise is able to send values, exceptions and notifications to the associated future.

The key difference between threads and tasks is the higher abstraction level of tasks. A task will not automatically generate a thread. To be precise, the C++ runtime decides  if a thread should be created. Reasons for the decision are: How heavy is the payload? How many cores are available? How high is the systemload?