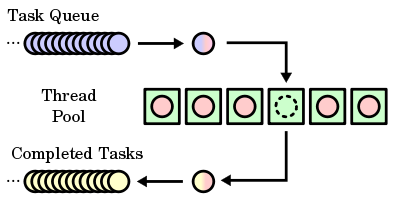
Thread pool pattern

[](http://en.wikipedia.org/wiki/File:Thread_pool.svg)

sample thread pool (green boxes) with waiting tasks (blue) and completed tasks (yellow)

In [computer programming](http://en.wikipedia.org/wiki/Computer_programming), the **thread pool**[**pattern**](http://en.wikipedia.org/wiki/Design_pattern_(computer_science)) is where **a number of**[**threads**](http://en.wikipedia.org/wiki/Thread_(computer_science))**are created to perform a number of tasks,** **which are usually organized in a**[**queue**](http://en.wikipedia.org/wiki/Queue_(data_structure))**.** Typically, there are many more tasks than threads. As soon as a thread completes its task, it will request the next task from the queue until all tasks have been completed. The thread can then terminate, or sleep until there are new tasks available.

The number of threads used is a parameter that can be tuned to provide the best performance. Additionally, the number of threads can be dynamic based on the number of waiting tasks. For example, a [web server](http://en.wikipedia.org/wiki/Web_server) can add threads if numerous [web page](http://en.wikipedia.org/wiki/Web_page) requests come in and can remove threads when those requests taper down. The cost of having a larger thread pool is increased resource usage. The algorithm used to determine when to create or destroy threads will have an impact on the overall performance:

* create too many threads, and resources are wasted and time also wasted creating any unused threads
* destroy too many threads and more time will be spent later creating them again
* creating threads too slowly might result in poor client performance (long wait times)
* destroying threads too slowly may starve other processes of resources

The algorithm chosen will depend on the problem and the expected usage patterns.

If the number of tasks is very large, then creating a thread for each one may be impractical.

Another advantage of using a thread pool over creating a new thread for each [task](http://en.wikipedia.org/wiki/Task_(computers)) is thread creation and destruction overhead is negated, which may result in better[performance](http://en.wikipedia.org/wiki/Performance_tuning) and better system [stability](http://en.wikipedia.org/wiki/Stability). Creating and destroying a thread and its associated resources is an expensive process in terms of time. An excessive number of threads will also waste memory, and context-switching between the runnable threads also damages performance. For example, a socket connection to another machine—which might take thousands (or even millions) of cycles to drop and re-establish—can be avoided by associating it with a thread which lives over the course of more than one transaction.

**When implementing this pattern, the programmer should ensure**[**thread-safety**](http://en.wikipedia.org/wiki/Thread-safe)**of the queue.**

Typically, a thread pool executes on a single computer. However, thread pools are conceptually related to [server farms](http://en.wikipedia.org/wiki/Server_farm) in which a master process distributes tasks to worker processes on different computers, in order to increase the overall throughput. [Embarrassingly parallel](http://en.wikipedia.org/wiki/Embarrassingly_parallel) problems are highly amenable to this approach.