Analogy – Team manager and Developers

A process in the computer system can be closely related to the Team manager in a tech company. As one process can have many threads to complete the task assigned to process the same way a manager can divide work among the team members in his team to complete the task that was assigned to him. Like thread has their unique Id developers have their unique name (most of the time). As thread shares code section, data sections to other threads; developers in a team also pull code from the same repo (codebase) they work with the same database.

As we know that in Computer System a process can spawn multiple threads which in our case is the limited number of members in a team. A thread can create another thread sharing its memory space the same can a developer in a team do. In our case, in analogy, a developer does not add another developer instead if he needs to do something that was not expected in the beginning then he stops his original task(if the original task is dependent on this new task) for a while and finishes the new task. It is just like parent threat creating child thread. As the new task gets completes developer can work on his original task; it is just like parent thread waiting for child thread.

Taking about the advantages of threads: In a team, if one person is stuck in a part, he is working on this won’t affect the performance of others in a team. It is just like an independent thread in a process. All the members in a team share the same codebase and database as threads share memory. Creating different processes and different memory spaces for that thread can be expensive which is also true in a team. Having a new team with manager architect and developer can be expensive instead if we can increase team size by a small percentage so overall, we become more productive with more team members. This can also be thought of as having many developers work under a manager is more productive than having one developer or only manager in a team which can be related a lot towards a single threaded process.

There can be a lot of challenges while working in a team. The same is true with threads within a process. Like threads identifying tasks can be difficult; while working in a team we don’t want one team member waiting for others to complete so he can continue his assigned task. We want every team member to be independent and not affected by others. Same as thread we want to make sure every developer has an equal workload. And dividing work among developers is a very difficult task. In the case of a thread, there is data splitting, but in our case, it is the same data with different copies. But this doesn’t minimize our problem. While different developers are working in different instances of the same data, they can by mistake change the data structure, which is a big problem in the team. Like Data Dependency in thread, we must make sure if one developer needs a component that another developer is working on than the component must be other developer priority. These components are hard to find at the beginning of the project. Testing and debugging are also harder when the project is divided among team members. One member changing one component can break the entire system. There is always a huge problem in “git” branching and merging. Sometimes people end up working in the same branch causing merge conflict. They also change the same file and can cause the problem related to identifying task like threads.

In the threads system, we have this asynchronous cancellation, deferred cancellation, and complete. The same can happen in a team too. Let’s think of the situation when developers are periodically asking the database guy to design the database so they can frame their code accordingly. This will end the pending work of the designing frame(a pipeline between database and code) in the developers side. This is more like our deferred cancellation. Let’s think of another situation: Our developers and architect are working on designing the application and the next day manager comes with this new design from somewhere that will stop everyone's current work and they will start new work. That is our asynchronous cancellation. And complete is the one I explained above: when a developer finishes his current assigned task he will move to a new task. Or a developer completes his sub-task and moves to the main task.

Process – Managers  
Threads – Developer and team members  
Thread Id – Developers name  
Code section – codebase  
data section – database  
Multiple developers with one manager – Many to one  
One developer in a team – One to one