Resumo

é o opsto de multitarefa preemptiva. aqui o p´roprio precesso ceder espaço de CPU a outro enquanto, em sua execussão, entra em estado de IO. ou seja. quando ela espera o output do usuário ela cede espaço a outra operação ser feita. é uma multitarefa cooperativa

São rotinas **cooperativas**, ou seja, são rotinas (funções, métodos, procedimentos) que concordam em parar sua execução permitindo que outra rotina possa ser executada naquele momento esperando que essa rotina secundária devolva a execução para ela em algum momento, portanto uma coopera com a outra.

Isso permite a execução em partes. Um dos grandes benefícios é manter algum estado entre os momentos de execução, então sua principal função é facilitar a execução segmentada, possivelmente criando alguma abstração na execução.

O principal mecanismo em linguagens para alcançar isto é o [yield](https://pt.stackoverflow.com/q/44293/101). Também pode usar uma [máquina de estados](https://pt.stackoverflow.com/q/208328/101) manualmente ou por biblioteca.

Então elas mantém o estado da rotina em algum lugar, incluindo aí em que instrução ela parou para poder continuar dali quando voltar.

A partir dessa estrutura é possível criar uma máquina de estados específica para permitir que a execução crie assincronicidade de execução que pode substituir *threads* que eram usadas para permitir a execução paralela e concorrente, mas era o mecanismo errado nestes casos.

Não é que *threads* não sejam úteis, elas são, mas apenas uma para cada processador, assim permite que várias execuções ocorram de forma efetivamente simultâneas. Mas a execução das operações diversas concorrentes não necessariamente simultâneas, não se beneficiam delas. Em cada processador só pode executar uma coisa de cada vez, mesmo que tenha a ilusão de estar executando várias. É mais vantajoso deixar as rotinas dizerem quando não precisam mais executar do que deixar o sistema operacional ficar trocando isso muitas vezes sem necessidade.

Então uma rotina executa sua parte e nos momentos que ela fará IO ela pode entregar a execução para outra rotina. O IO será feito pelo sistema operacional ou pelo serviço que está sendo usado por conta dele, nada tem a ver com sua aplicação, então não tem porque sua aplicação ficar esperando sem fazer nada. Com a corrotina ela coopera dizendo que outra rotina pode ser executada nesse momento. Quando a parte de IO terminar volta para a rotina anterior terminando de executar com o resultado do IO em mãos.

Não tem tanta diferença assim, só que a suspensão de execução é determinada pela própria rotina e não pelo sistema operacional, por isso ela é chamada de cooperativa, o sistema operacional faz de forma **preemptiva**.

Algumas linguagens abstraem tudo isso, outras exigem que você cuide de todo o mecanismo de troca de execuções (a máquina de estado), o que na prática faz "ninguém" usar.

*Thread* é útil quando precisa criar linhas de processamento e a maior vantagem se dá quando se tem vários processadores disponíveis. Claro que a *thread*, que é um recurso do sistema operacional, pode ser usada como máquina de estados de execução, mas tem um custo mais alto. [Veja mais](https://pt.stackoverflow.com/q/1946/101). Corrotinas neste contexto é mais útil quando tem entrada e saída ou quando precisa linhas de processamento que vão além dos processadores disponíveis.

Há muito desperdício no uso de *thread* porque tem o custo de gerenciamento do sistema operacional, tem a troca de contexto que é cara, tem a alta probabilidade de uma *thread* ser acionada e não ter nada para fazer. Corrotinas são bem leves, gerenciadas de forma simples pela própria aplicação e são acionadas conforme a necessidade. O "sob demanda" é o segredo aqui.

Claro que se a rotina que deve cooperar for mal escrita, pode ser que enrosque por muito tempo atrapalhando tudo. A preempção força a troca de execução da rotina de uma *thread* para outra, mesmo que a aplicação não ajude.

Houve uma revolução de performance em servidores HTTP, que o Node levou a fama sem ter o mérito, justamente por usar corrotinas no lugar de *threads*.

A **corrotina é ortogonal à *thread***, então uma não depende da outra, mas pode ser que uma corrotina seja colocada em *thread* separada, dependendo da necessidade.

Nos comentários o AP fala sobre escrever código sequencial que acaba sendo distribuído concorrentemente, e é isso mesmo, desde que a linguagem ou até mesmo a biblioteca ajude.

Por falta de um mecanismo melhor nos anos 90 se disseminou o uso de *threads* para algo que não precisava delas, inclusive porque quase nenhum computador tinha mais que um processador.