Conceito

Uma **condição de corrida** ocorre quando o resultado de um programa depende de como o sistema operacional escalona as threads. Ou seja, seu resultado não é determinístico.

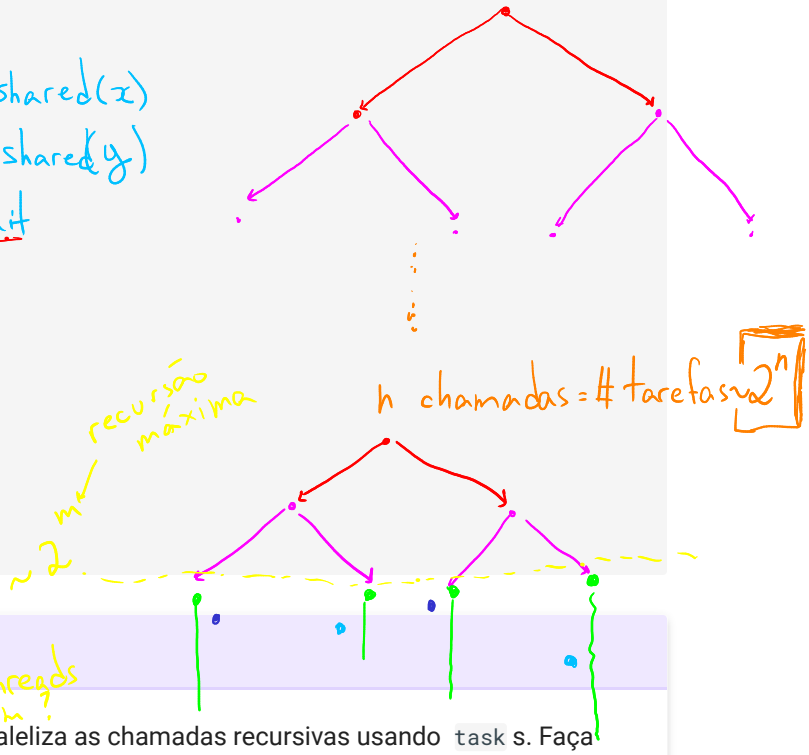
Parte 1 - tarefas e funções recursivas

O código abaixo calcula a sequência de Fibonacci [https://pt.wikipedia.org/wiki/Sequ%C3%AAncia_de_Fibonacci] usando um algoritmo recursivo.

```
#include <iostream>

int fib(int n) {
    int x,y;
    if(n<2) return n;
    x=fib(n-1);
    y=fib(n-2);
    return(x+y);
}

int main() {
    int NW=45;
    int f=fib(NW);
    std::cout << f << std::endl;
}
```



Tarefa 2

Crie uma nova função `int fib_par1(int n);` e paraleliza as chamadas recursivas usando `task` s. Faça com que cada chamada recursiva seja executada como uma tarefa. **Dica**: é preciso esperar as tarefas acabarem antes do `return` ?

Tarefa 3

Compare com o código original. Houve melhora? Por que?

O exercício acima exemplifica o custo de criar e escalonar tarefas. Vamos melhorar esta paralelização agora limitando o número de tarefas criadas.

Tarefa 4

Crie uma nova função `int fib_par2(int n)` e paraleliza as chamadas recursivas. Faça com que sejam criadas no máximo `max_threads` tarefas. **Dica**: passar como argumento o nível da recursão pode ajudar.

Parte 2 - estratégias de paralelização