11 Lezione

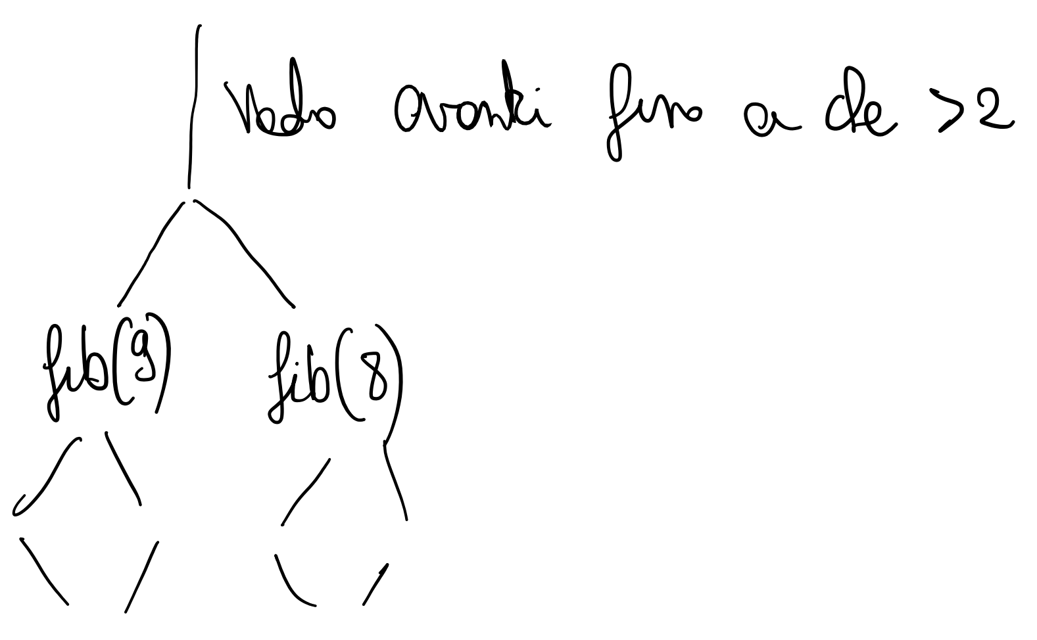
martedì 12 marzo 2019 14:14

```
#pragma omp task depend (in: x)
   ⋮
#pragma omp task depend (out: x)
```

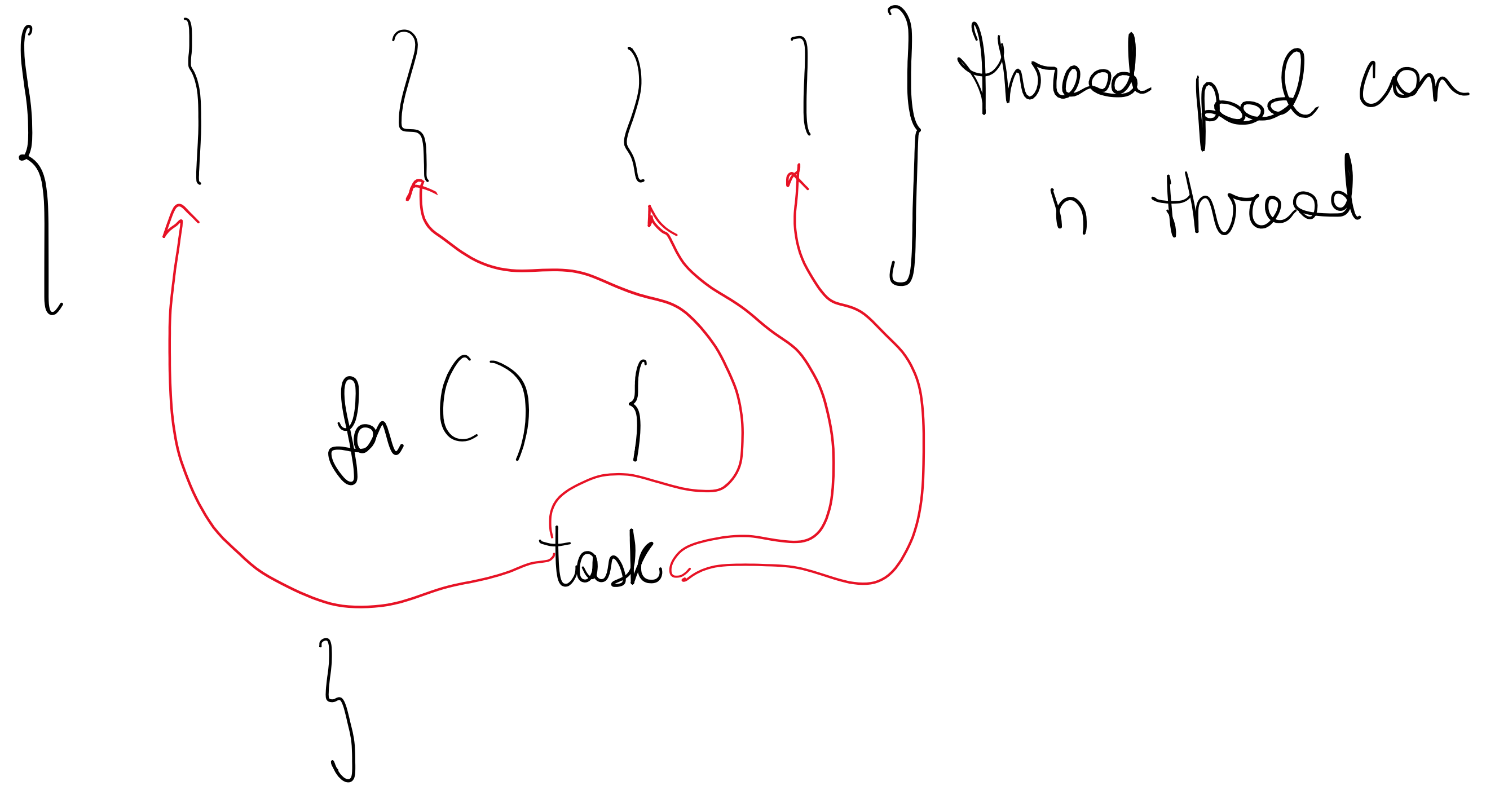
↑ il primo task non parte fino a che il secondo non termina e x è disponibile

#pragma omp taskwait → aspetta tutti i task dichiarati dentro il pragma

Used to do wait across different calls,



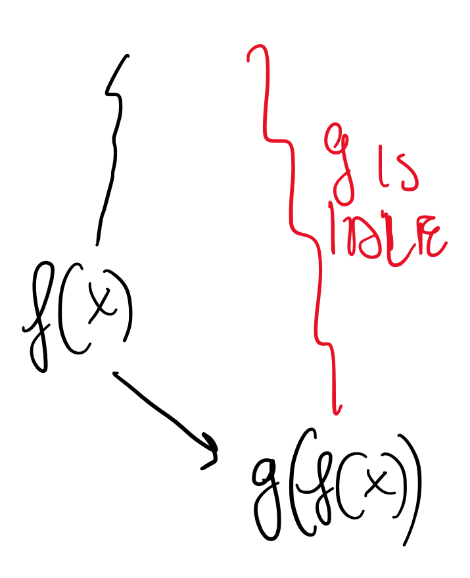
taskwait attende che entrambi siano terminati e poi somma i risultati parziali $(i_2 + i_1)$

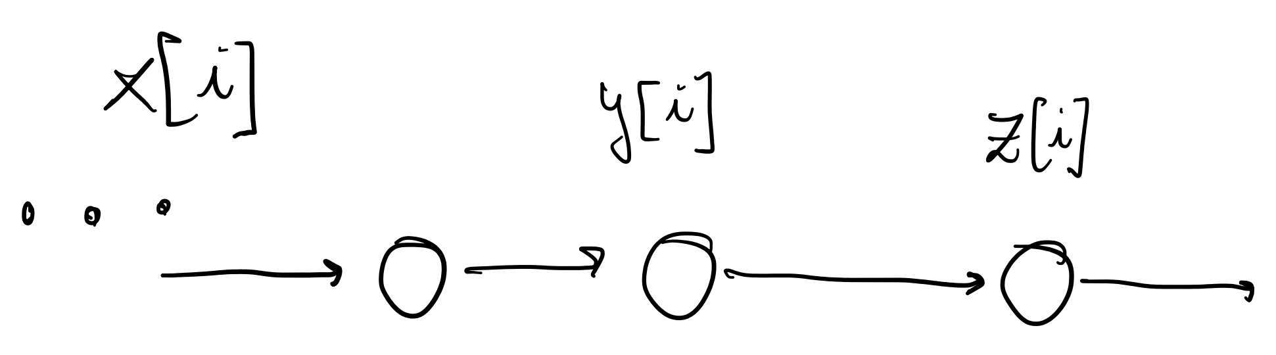


pipe fg

int x g(f(x))

Usiamo dependency per eseguire in modo concorrente i due thread



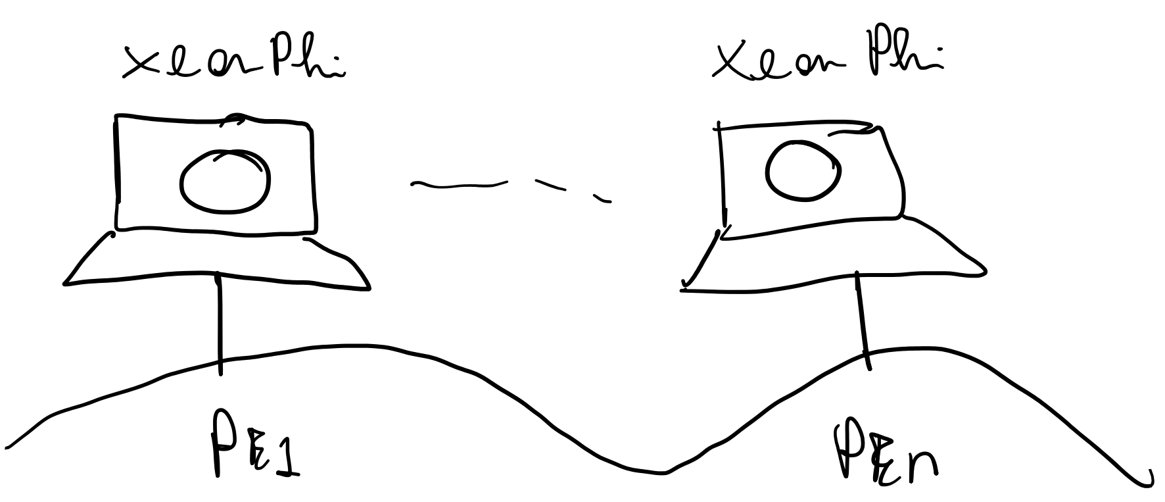


14:45

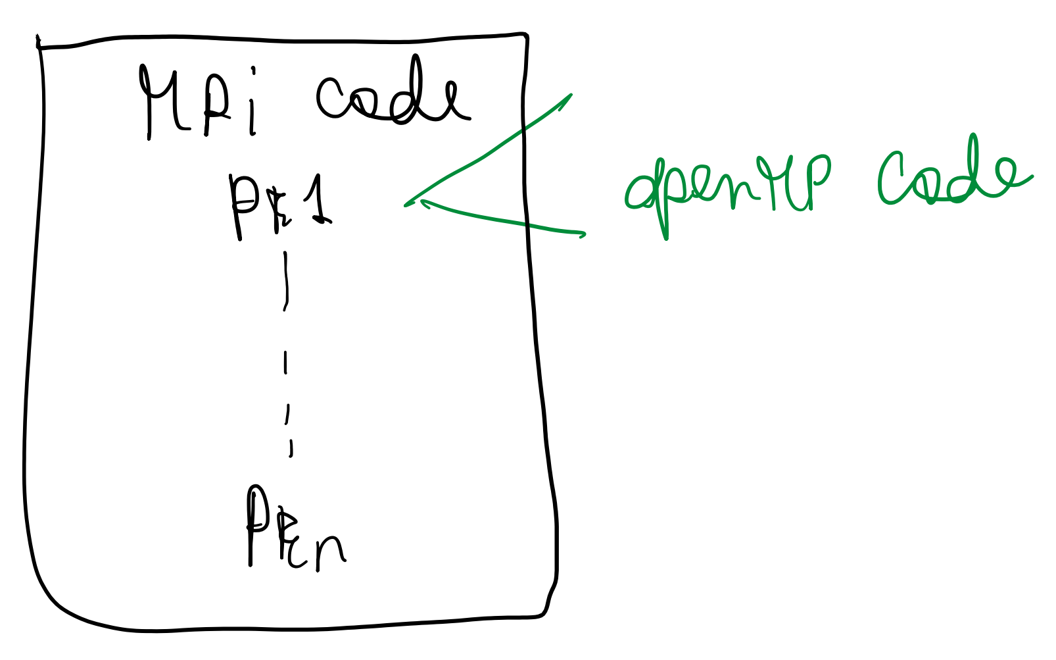
usare depend per implementare una pipeline non è il modo più intelligente per implementare una pipeline.

14:51 ↓ spiego perché non va usato.

OpenMP is the standard for the shared memory multicore.

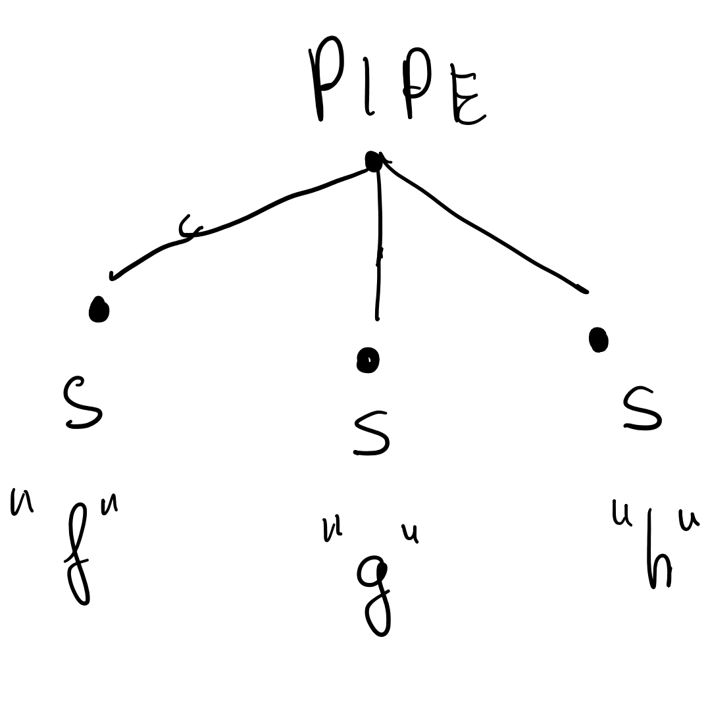


parallel code file:



Problem → program { pipeline / farms

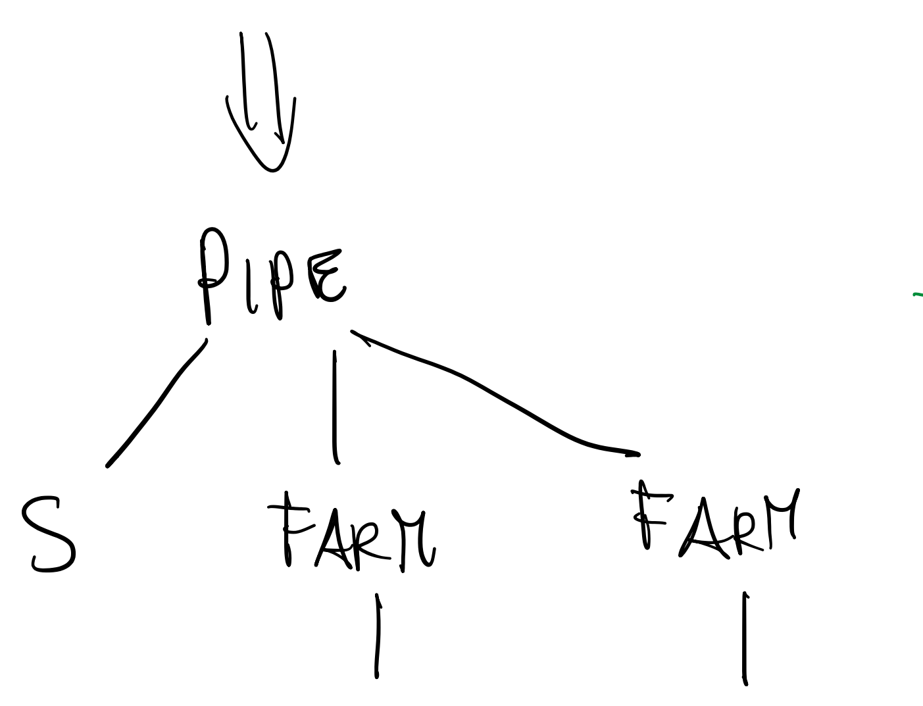
refactoring rules: refactor the tree in modo da partire da una soluzione e applicando le regole ottenere un altro tree che calcola la stessa cosa ma gestisce meglio le ricerche.



t_f $t_g = 10 t_f$ $t_h = 9 t_f$

I can introduce farm in the place where I have a bottleneck:

⇓



this computation has the same semantic as the first tree.

If I put enough worker I could have a match with the time of the first stage

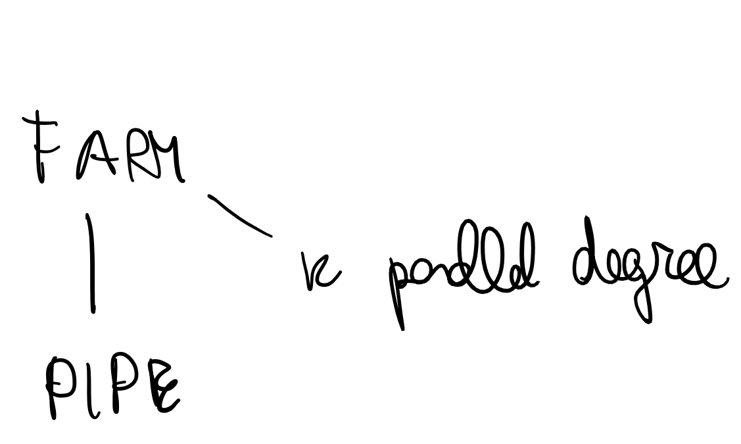
I would like to do ~~this~~ this automatically.

Il numero dei worker come lo scelgo? Lo sceglie chi scrive il programma? No, deve essere scelto durante l'esecuzione.

Possiamo fare varie cose:

→ Far prendere la decisione alla libreria,

Potrei anche avere un albero diverso:



Posso fare una astrazione del programma e cercare di avere un pattern tree che sia 15:33

Programmare queste cose può essere un problema

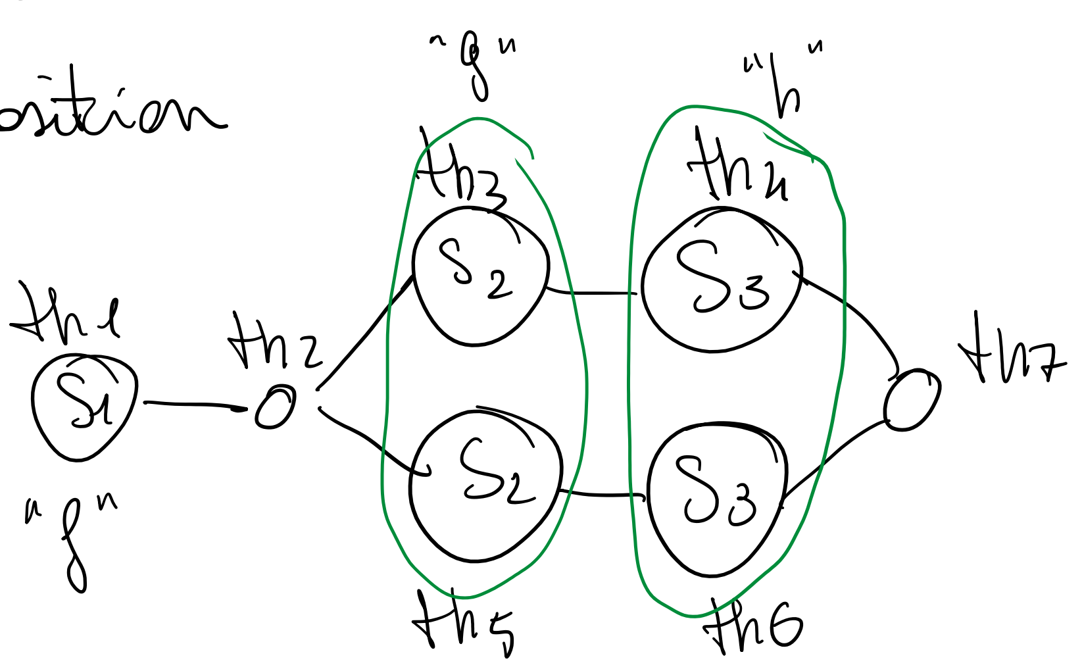
Tool che permette di esplorare lo spazio dei vari possibili pattern transformation.

Come trasformo i pattern durante l'esecuzione?

↳ 15:36

RPLSH

- → Pipeline
- → Farm
- → Sequential
- → Composition



Come trasformo l'albero in un programma?

Ogni parte deve dire il codice che esegue la funzione g, f e h