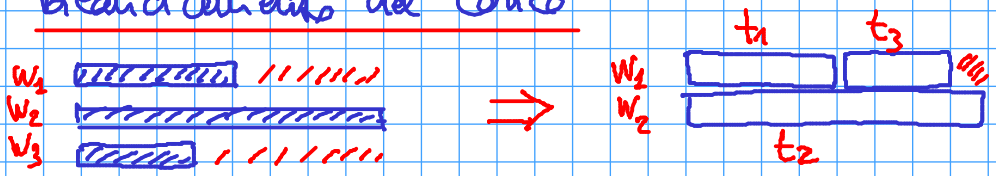
Bilanciamento del carico



map (coll, f)

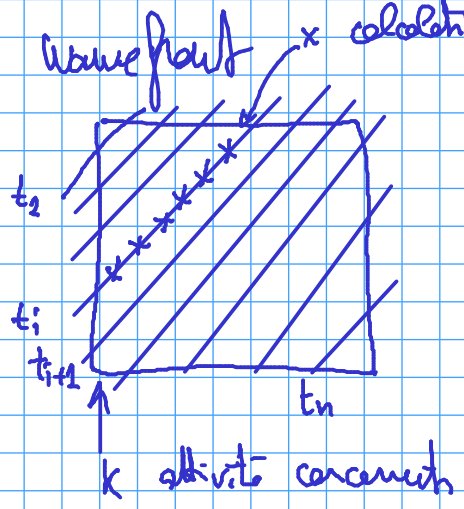
"teorica" $\forall x_i \in coll \mid f(x_i)$ "in parallelo"

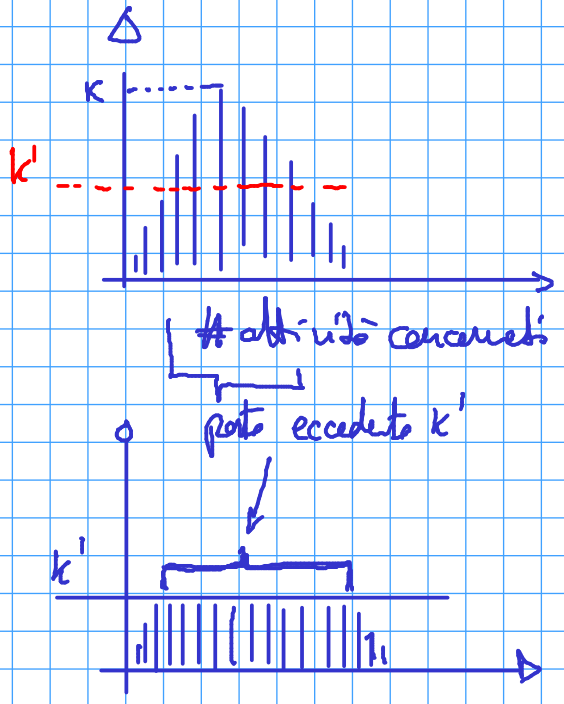
"in pratica" coll ↔ partizione$_1$ … partizione$_n$

in fondo c'è sempre uno "barriera"

stream parallelism

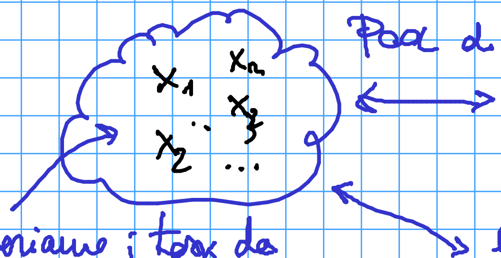
worker W_1 W_2 W_3 : worker di 1 farm





Tecniche di implementazione (di map e di Farm) con bilanciamento di carico

Pool di TASK



esecutore$_1$ … esecutore$_{nw}$ { ciclo: while(∃task;){
calcola il task;
"deposita" il risultato;
}

inseriamo i task da calcolare

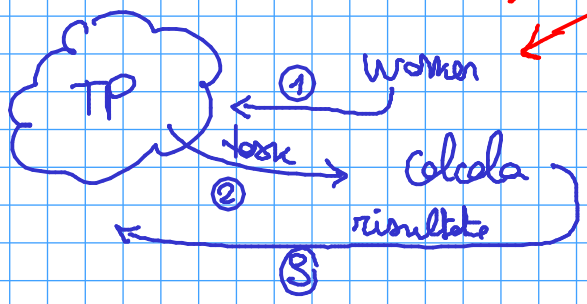
task: rappresentazione dei dati necessari x calcolare un'attività concorrente

sorgente dello stream (stream par.)
partizionatore della collezione (data par.) } inseriscono task nel task pool

1) "auto scheduling"

(controllo è dei worker)

worker: entità attive



+

entità attiva (super) che regola l'attivazione e la terminazione dei worker

task pool è entità passiva

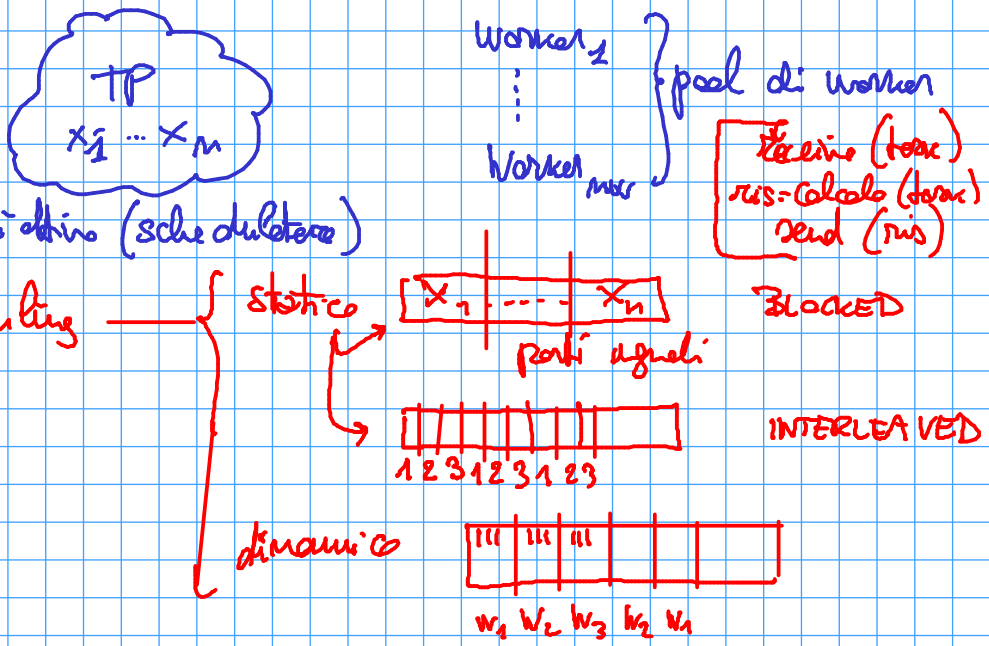
(struttura dati: pila, coda, lista ...)

↳ "concurrent self" (rientrante)

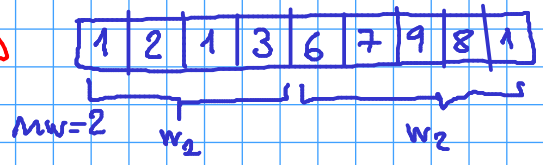
il task pool costituisce la risorsa critica

può costituire il collo di bottiglia

2) Scheduling statico/dinamico
(controllo sul Task Pool)



SCHED STATICO BLOCKED



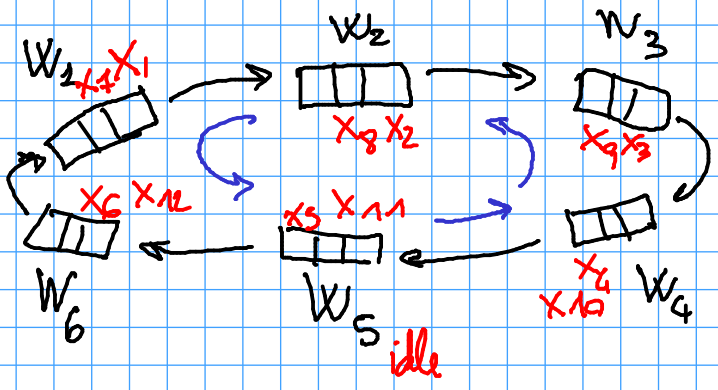
T_t $t \gg T_t$

SCHED STATICO INTERLEAVED

w_1 1 1 6 9 1

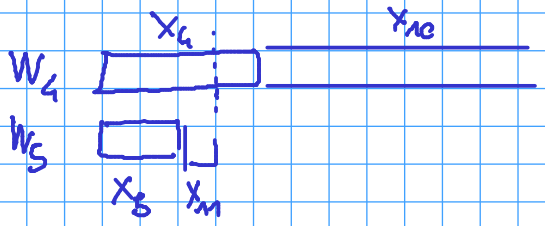
w_2 2 3 7 8

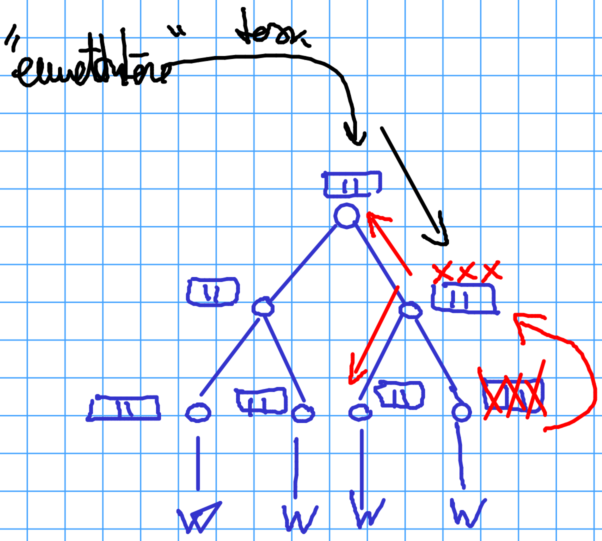
Realizzazione distribuita del task pool



inserzione iniziale

avviene 1 elemento x vettore in RR (Round Robin)



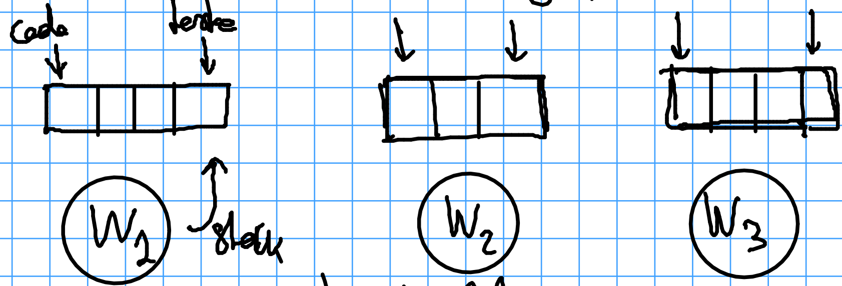


distrib iniziale uniforme

worker esegue task "locali"

se non ce ne sono propaga la richiesta verso antenato + vicino

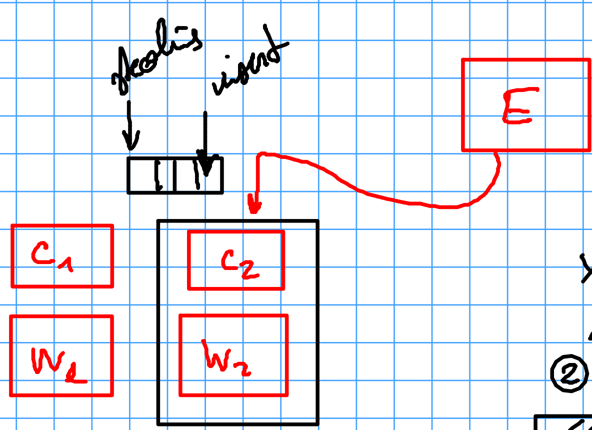
Random Job stealing / Work stealing

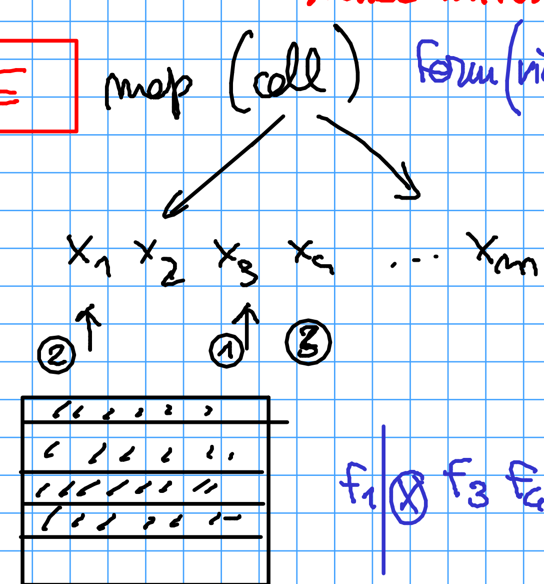


Hp: interconnessione completa !!

pop → task da calcolare
↓
genera in ris che sia un nuovo task
push (task)

pop → errore
⇒ $i = rand()$
$i \in [1, nw]$
e al worker i della coda TP ruba un task





Forum (video)

f_1 | f_2 f_3 f_4 f_5

schedule (guided) (omp)