DATA PARALLEL

$f(x_i) \,||\, f(x_j)$

MAP $\langle x_1 \dots x_n \rangle \; f \to \langle f(x_1) \dots f(x_n) \rangle$

- embarrassingly parallel
- result isomorphic to the input

$x_i : \alpha \qquad f : \alpha \to \beta \qquad y_i : \beta$

"in place" not "in place"

$x \to x^2$

REDUCE $\langle x_1 \dots x_n \rangle \; \oplus \; x_1 \oplus x_2 \oplus \dots \oplus x_n$

associative
commutative

$x_i : \alpha \qquad \oplus : \alpha \times \alpha \to \alpha \qquad y : \alpha$

foldr : α list $\times \alpha \to \alpha$ $\qquad [x_1 :: [x_2 :: [\dots]]$

foldl

$x_2 \oplus (x_2 \oplus \dots$

Inner Product

$\langle x_1 \dots x_n \rangle \times \langle y_1 \dots y_n \rangle$

$\sum_{i=1}^{n} x_i * y_i$

↓

(reduce (+)) (map (*))

map zip $\langle x_1 \dots x_n \rangle \langle y_1 \dots y_n \rangle \; * \; =$

$\langle x_1 * y_1 \; , \; x_2 * y_2 \; \dots \; x_n * y_n \rangle$

Google Map Reduce f g $\langle x_n \dots x_2 \rangle$

$f : x_i \rightarrow \text{pair}(v_i, k_i)$

$g : \langle \text{pair}(v_1, k_x) \dots \text{pair}(v_m, k_x) \rangle$

$\rightarrow v_1 (g) v_2 (g) v_3 \dots (g) v_m$

count words $\equiv$ Google MapReduce(word $\rightarrow$ (1, word),
$(x, w) \times (y, w) \rightarrow (x+y, w)$)

for(w : words)
if(w $\in$ HashMap) hashmap(w) ++
else insert(w, 1) $\rightarrow$ hashmap

Stencil pattern

$\phi : \alpha \times \alpha \times \ldots \alpha \rightarrow \alpha \quad < \quad x_i \quad >$

κ

$\forall i$ x_i

$i-1$ i $i+1$

3-ary neighbourhood

κ items

ϕ

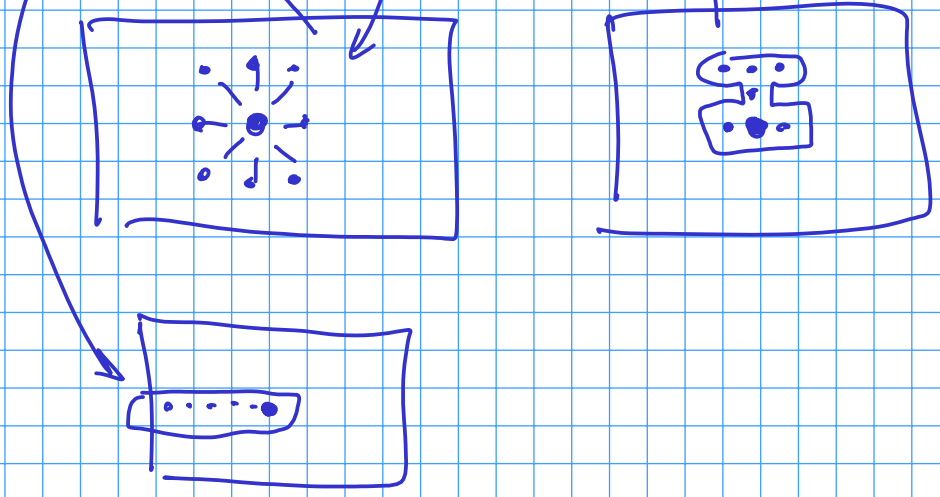
y_i

x_i

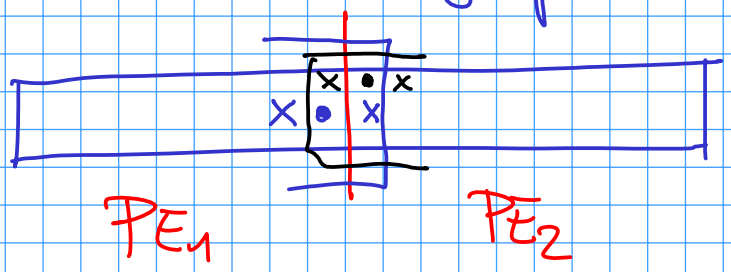
$f(\square\square\square\bigcirc\square)$

neighbourhood

fixed	regular
variable	irregular

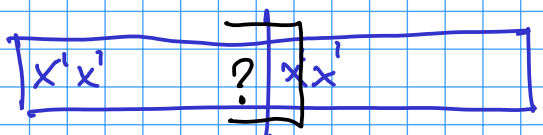


1) it is not embarrassingly parallel



$avg(x_{i-1}, x_i, x_{i+1})$

2) in place not in place (iterative stencils)
makes difference

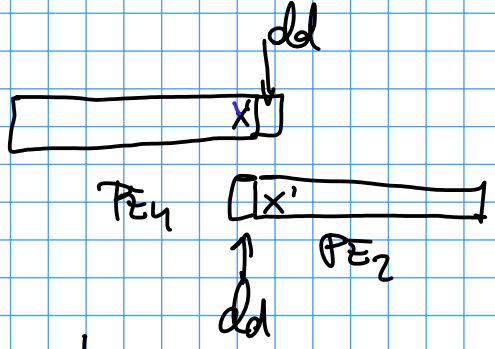


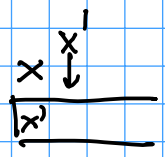
for(i=0, i<N/2-1

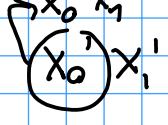
for(i=N/2, i<N

x[i] = f(x_i, x_{i-1}, x_{i+1})

1)

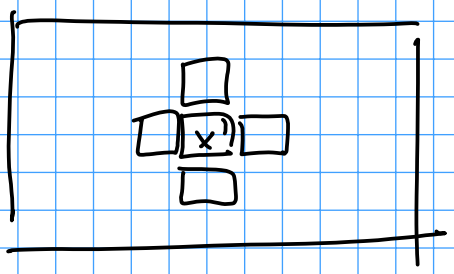


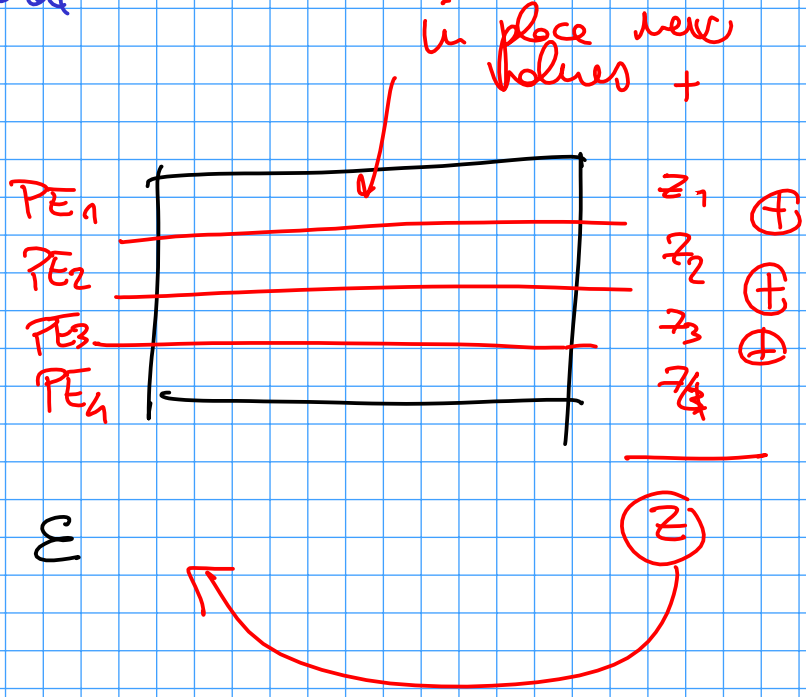




2) need a buffer
hosting the new
values computed
up to the point
the corresponding positions
are no more
needed "as old values"
to compute "new"
values

stencil - reduce pattern





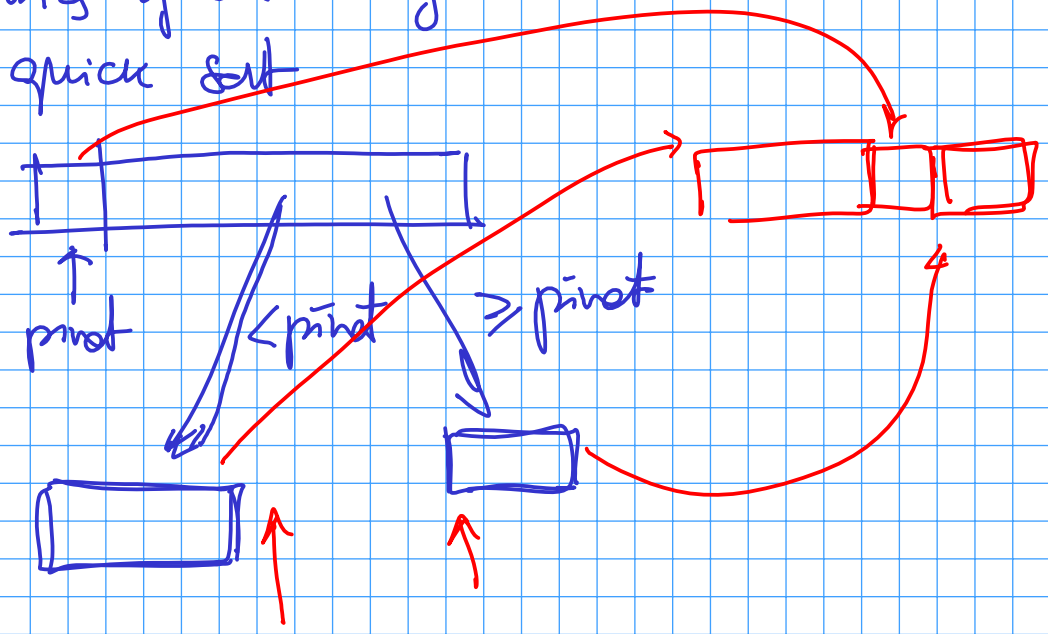
$\bigoplus_{i=\dots} |x^i - x^{i+1}| < \varepsilon$

divide & conquer

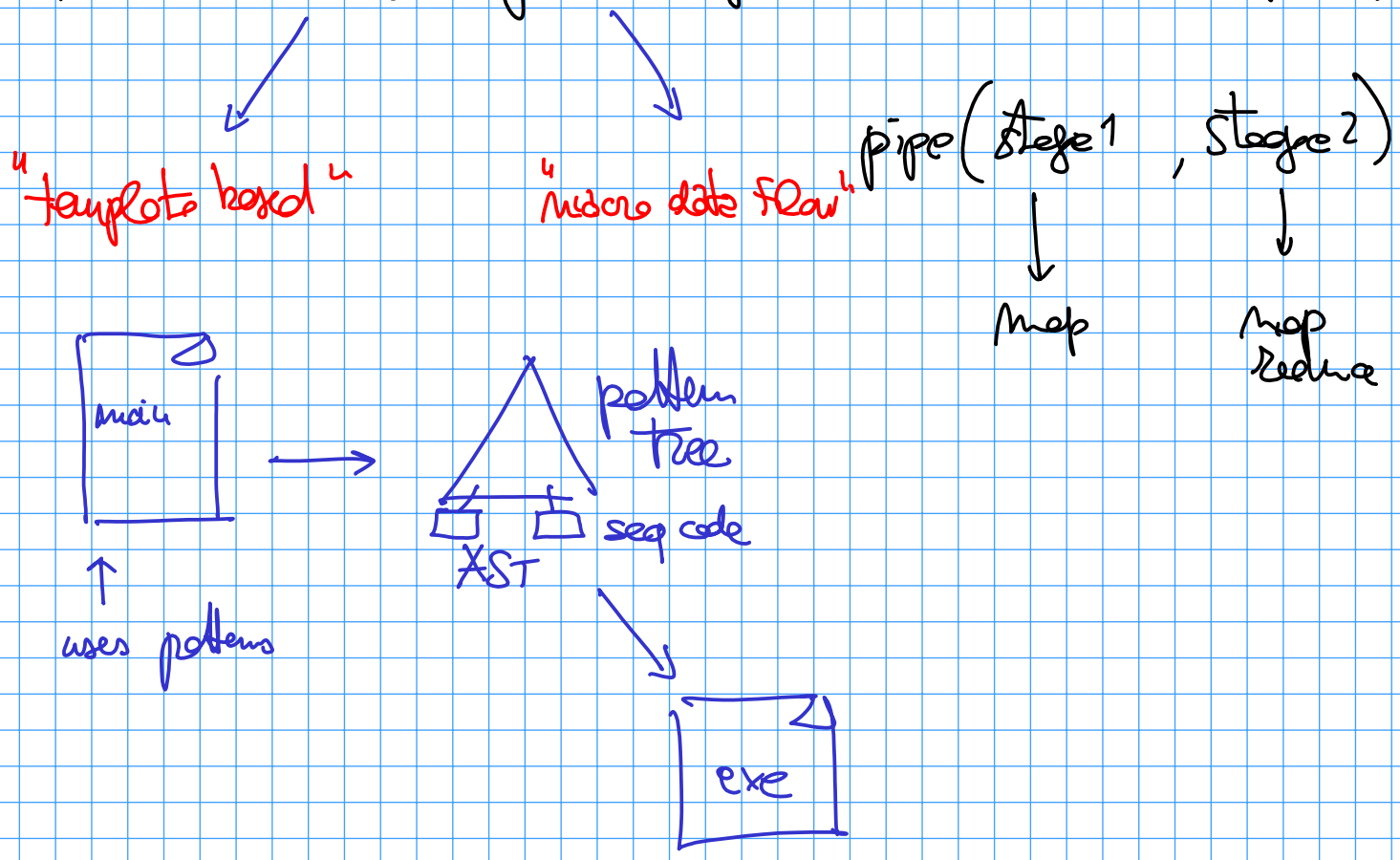
↓

Sorting of an array

quick sort



Implementation of "pattern frameworks" with composition



①

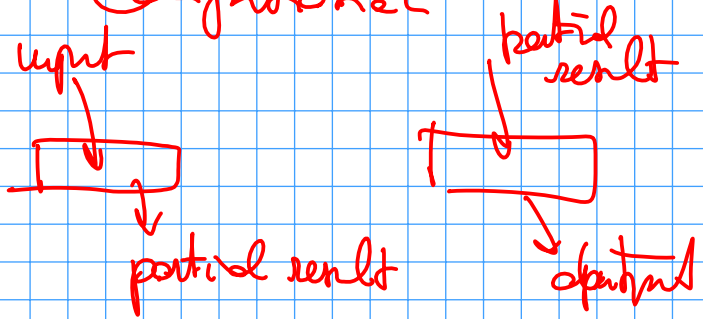
∀ patterns I know some "template" implementation

for the patterns in the AST

→ use these templates

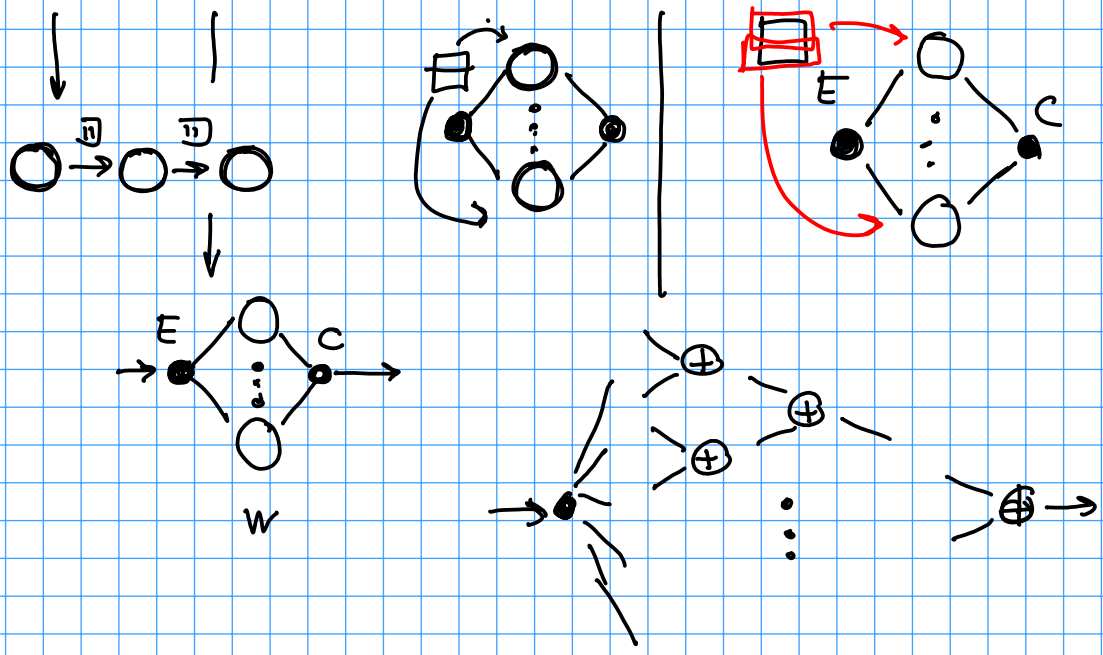
②

turn each pattern into code that computes "parallel" chunk of the overall computation



provide a generic interpreter executing these tasks (ASAP)

PIPE FARM MAP REDUCE STENCIL



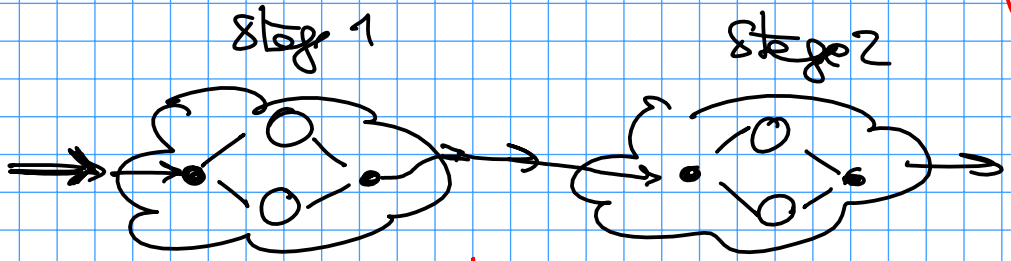
Parameters

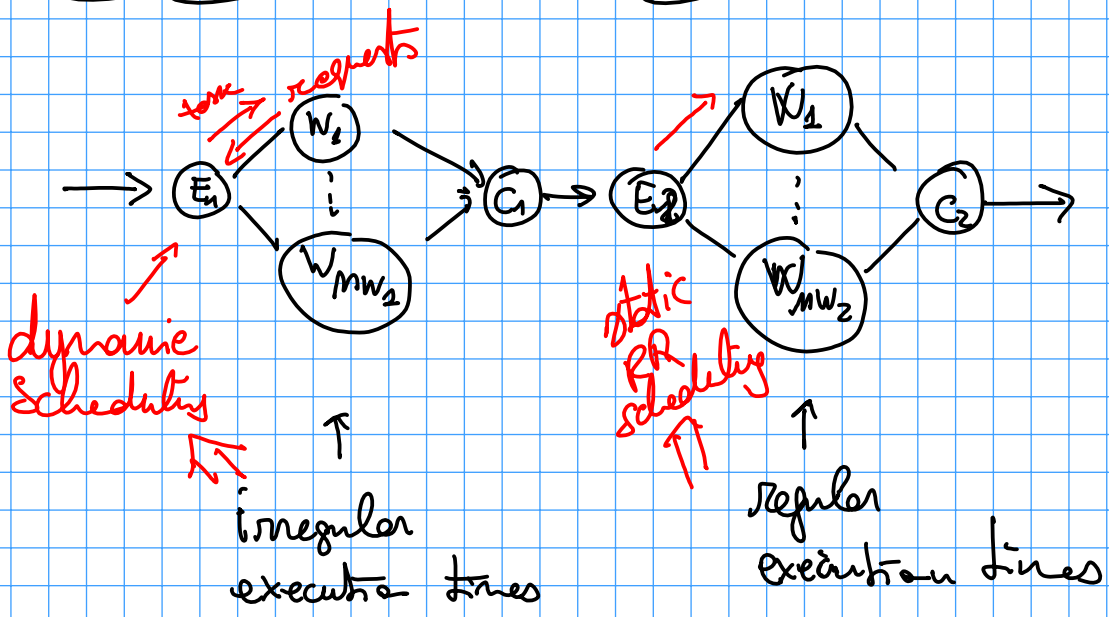
- per degree
- scheduling
- load balancing policies

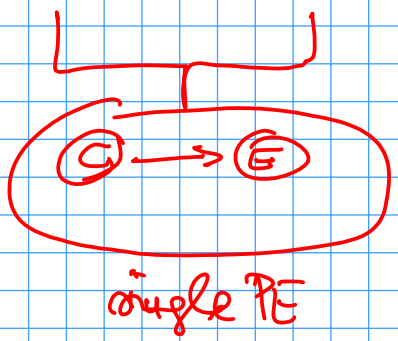
- business logic code e.g. function computed by workers by stages

1 entry point (here I get the input)
1 exit point (here I put the result)

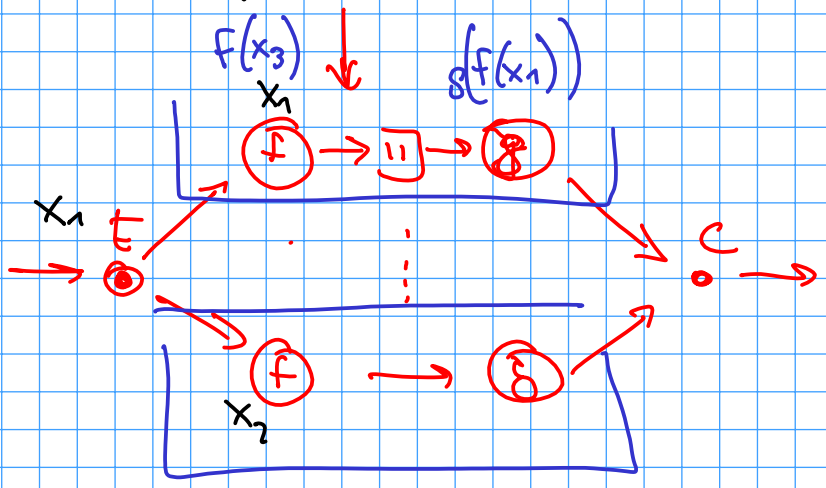
COMPOSE (under 1 entry/exit point assumption)





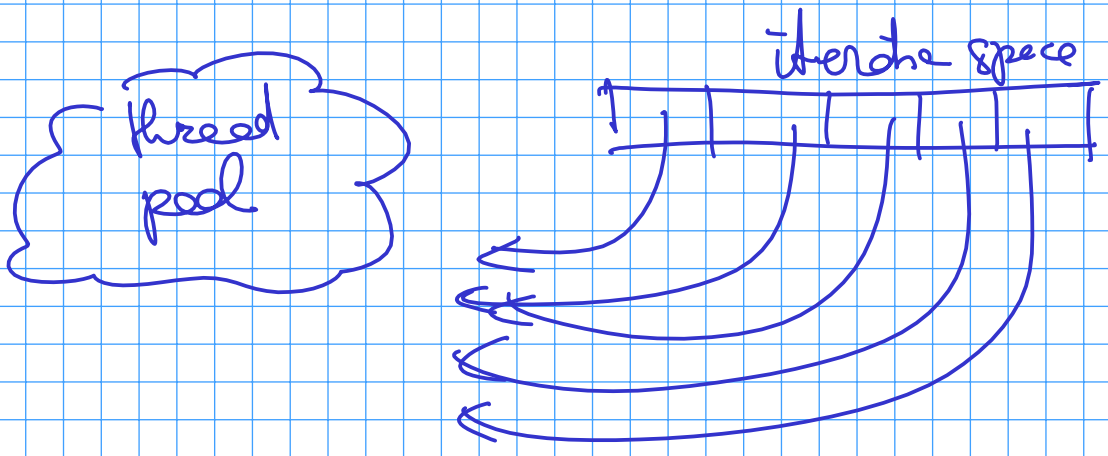


farm (pipe (f, g))

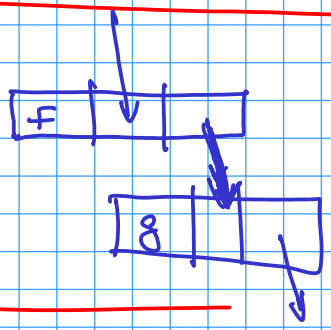
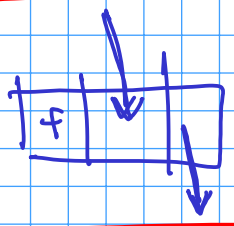
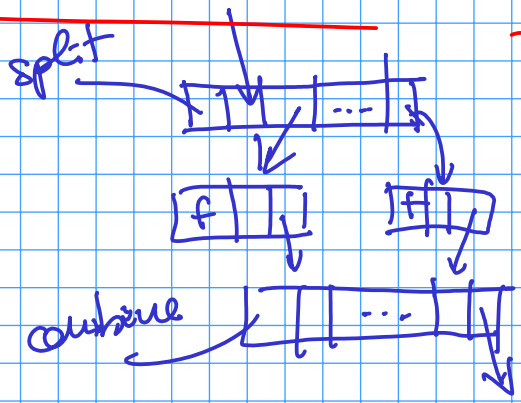


Open MP

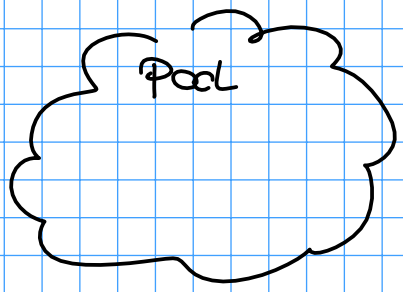
#pragma omp parallel for ... chunksize(n)

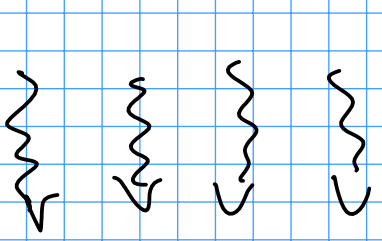


MACRO DATA FLOW BASED

PATTERN	GRAPH of // COMPUTATIONS
PIPE (f, g)	
FARM (f)	
MAP (f)	

PARALLEL INTERPRETER

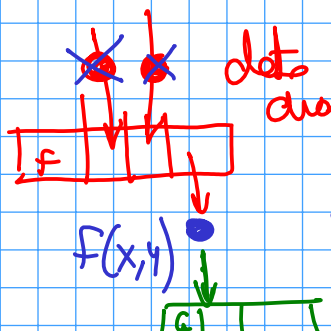




forever {
- get a fireable instr from pool
- compute
- put results in the pool

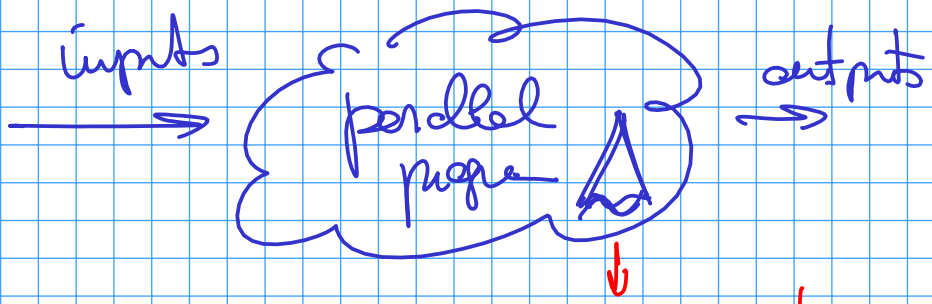
}

FIREABLE INSTR



COMPUTE

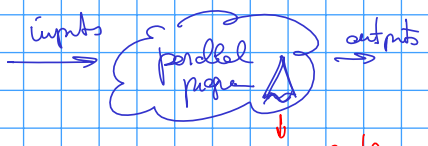
put tokens into pool: if it is the last one → FIREABLE INSTRUCTION



① compile the skeleton tree into a graph

② ∀ input item
→ create 1 instance of the graph with the input as input token & put it into the pool

③ activate the thread pool (interpreters)

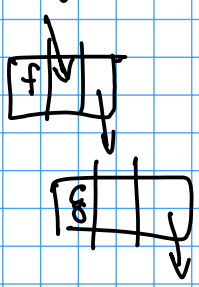


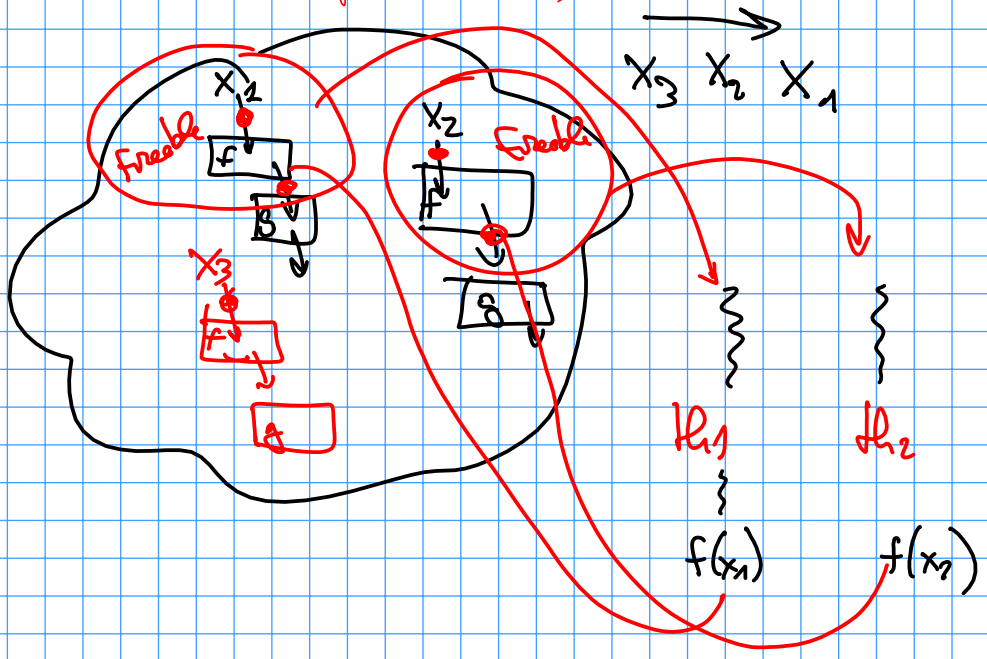
① compile the nodes tree into a graph

② ∀ input item
→ create 1 instance of the graph with the input as input token & put it into the pool

③ activate the thread pool (interpreters)

pipe (farm (f), 8)

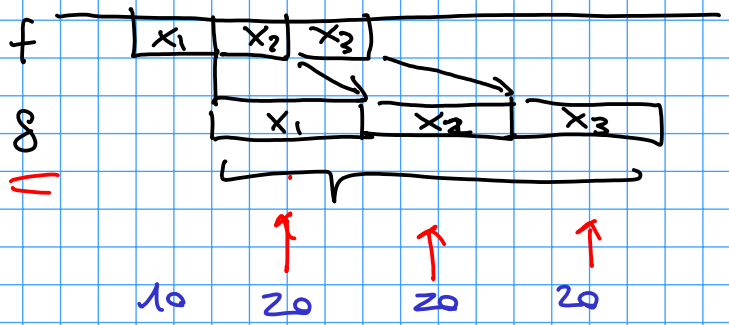




pipe(f , g)

$t_f = 10t$ $t_g = 20t$

$T_S = 20t$



x1

↓

freeable f(x1)

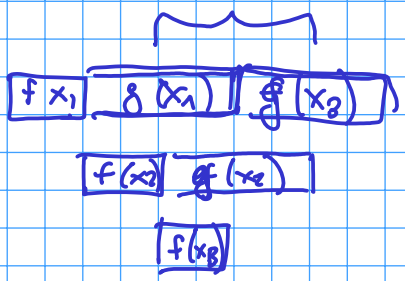
freeable g(x1)

freeable g(x1)

x1

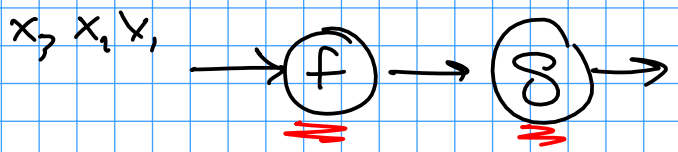
f

f x2



10 20 20

$t_f = 10t$ $t_g = 20t$



$T_S = 20t$

pipe (f, g)

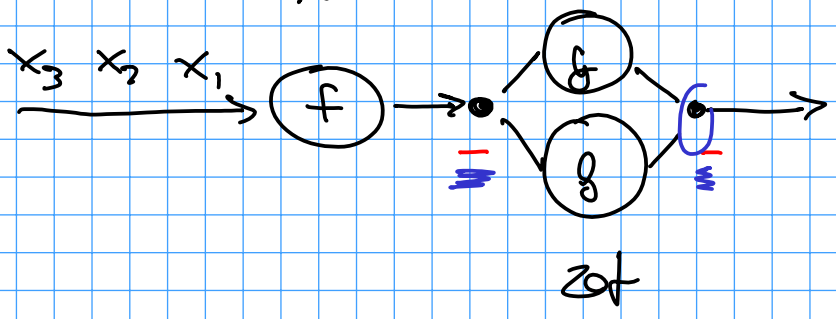
↓

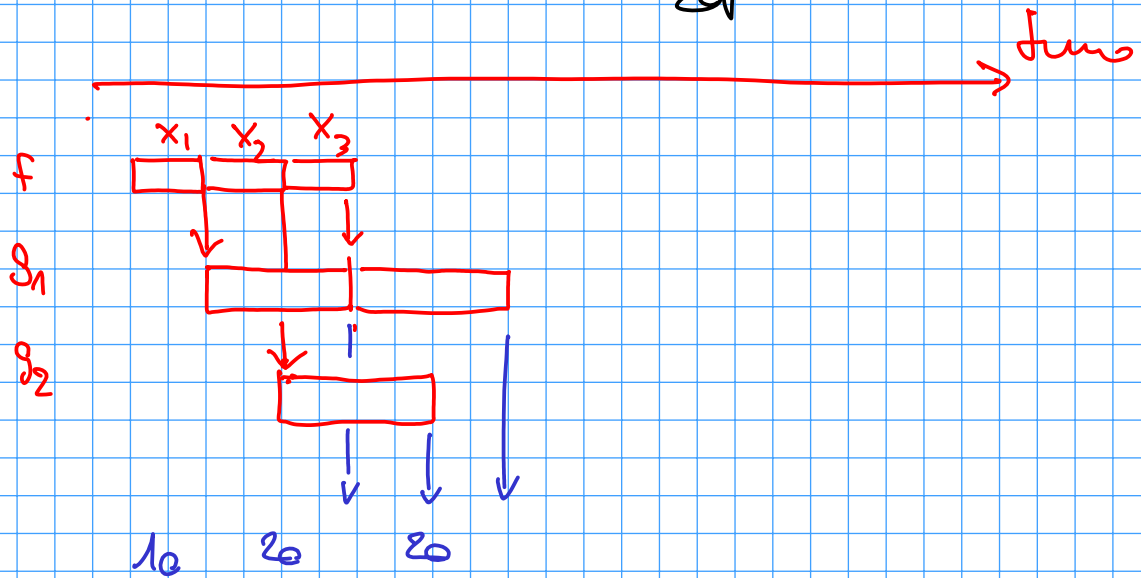
$\Delta \equiv$ farm (Δ)

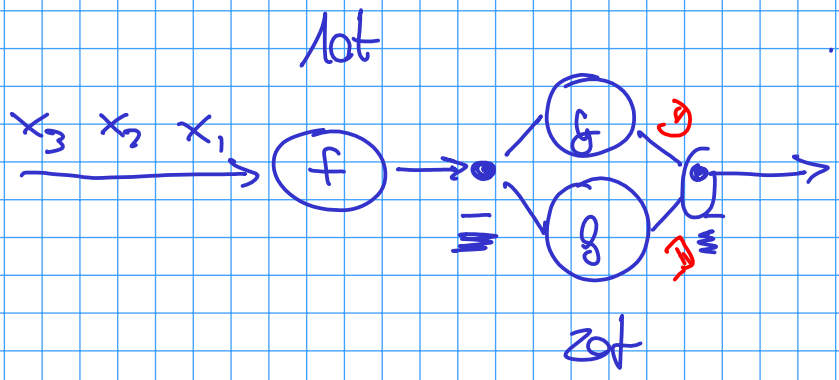
↓

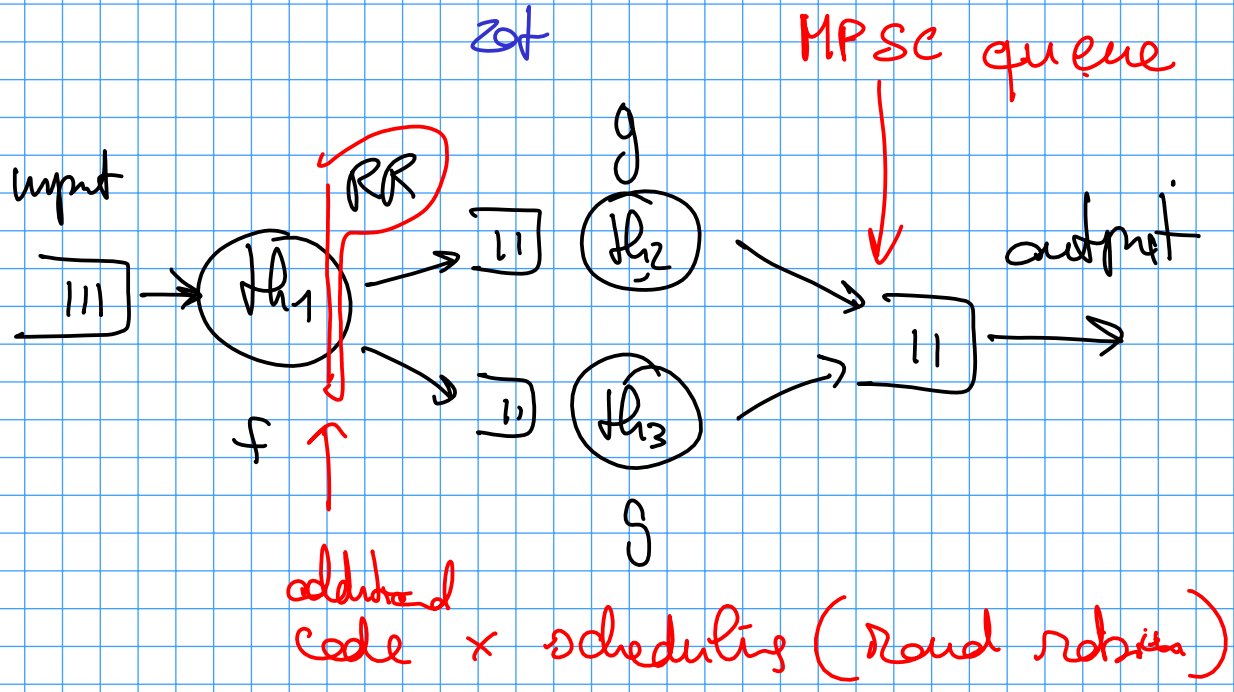
pipe (f, farm (g))

mw = 2









addthread

code × scheduling (round robin)