MACRO DATA FLOW IMPLEMENTATION

DATA FLOW (VS. CONTROL FLOW)
↳ VON NEUMANN ARCHITECTURE

↓

DATA FLOW ARCHITECTURE

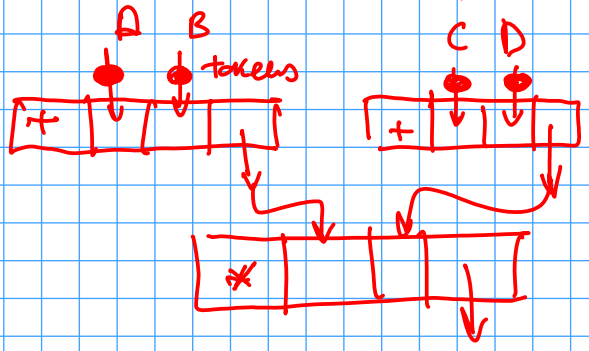
$(a+b)*(c+d)$

CONTROL FLOW WAY

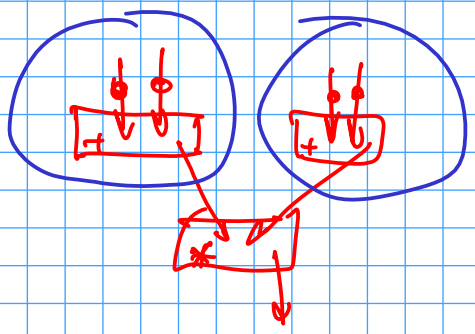
```
LDC
LDA
LDB
LDD
ADD A,B
ADD C,D
MUL
```

```
PC
LD A
LD B
SUM → TEMP
LD C
LD D
SUM → TEMP2
MUL TEMP,TEMP2
ST
```

DATA FLOW WAY



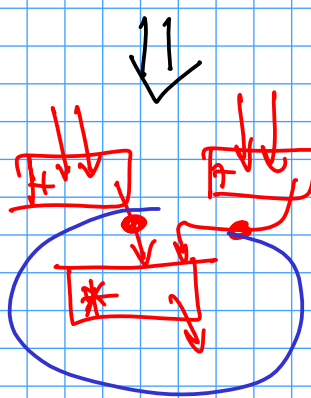
DF graph of DF INSTRUCTIONS is computed by repeatedly
- selecting fireable DF inst (those with all input tokens available)
- computing them
- directing results to the proper places



fireable

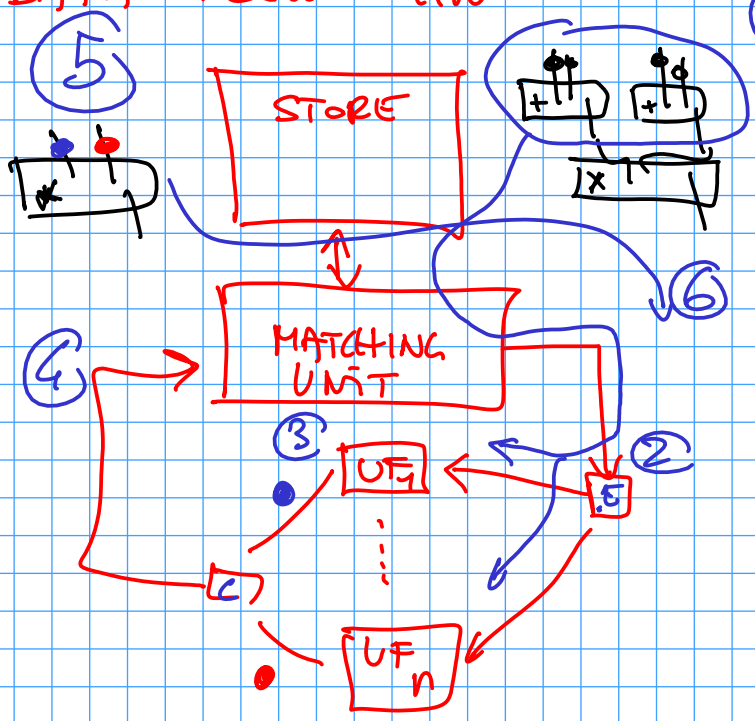
both can be computed "in parallel"

or any scheduling may be chosen to compute them serially

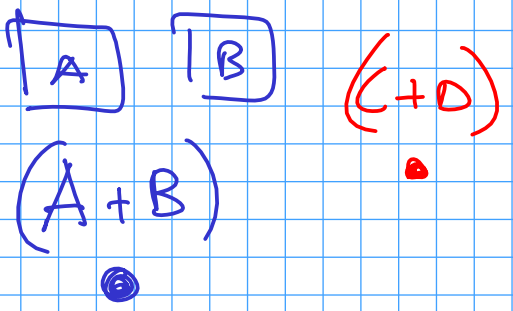


fireable

DATA FLOW HW



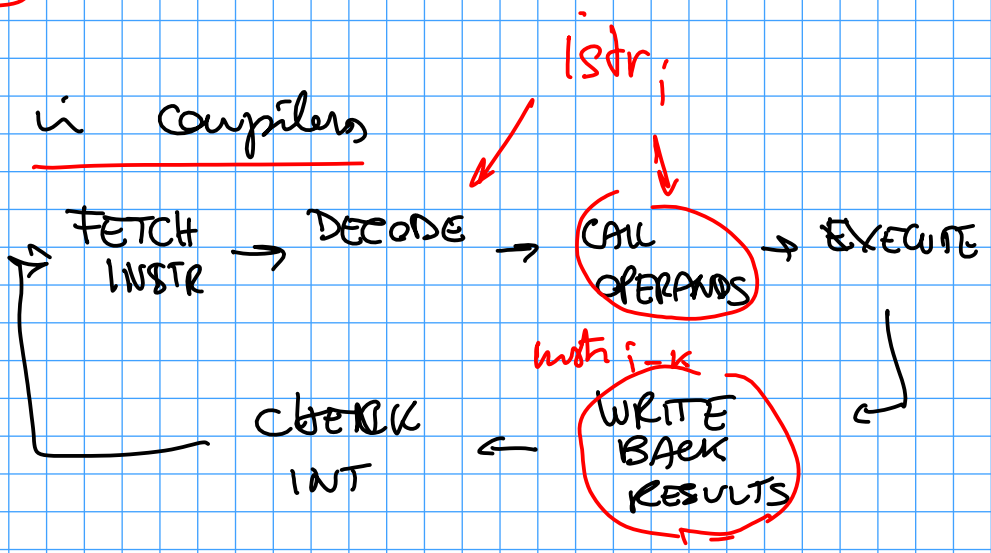
$a_{ij} + b_{ij} \rightarrow (A+B)_{ij}$



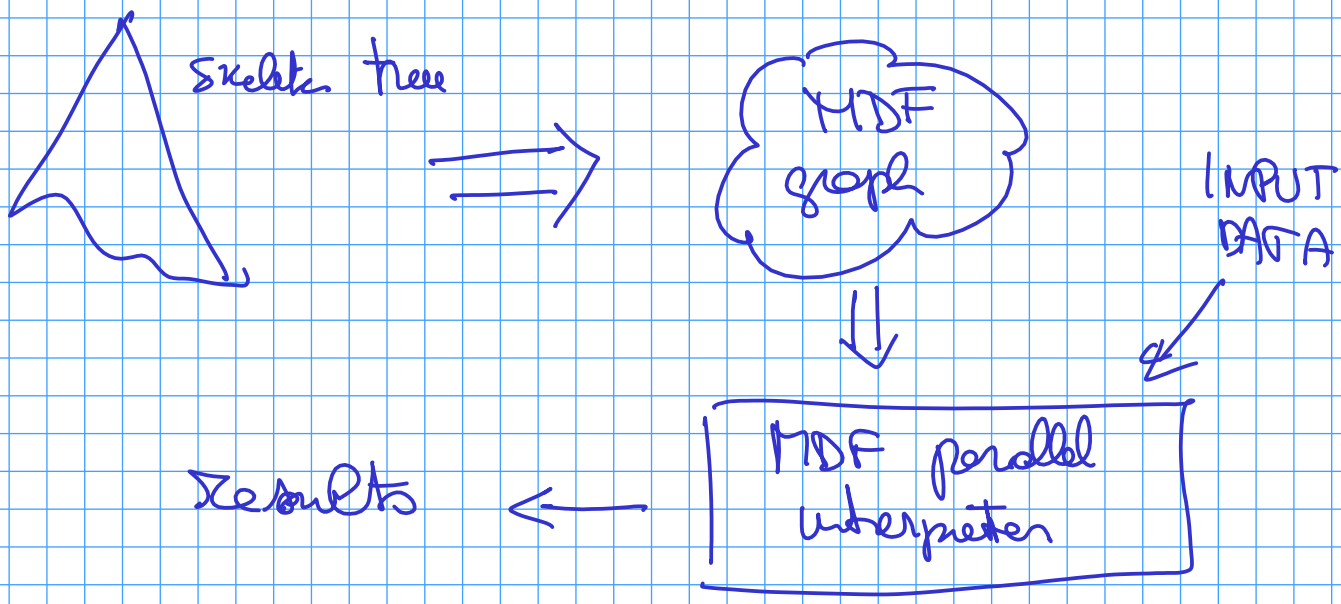
MANCHESTER DF ARCHITECTURE

DATA FLOW in compilers

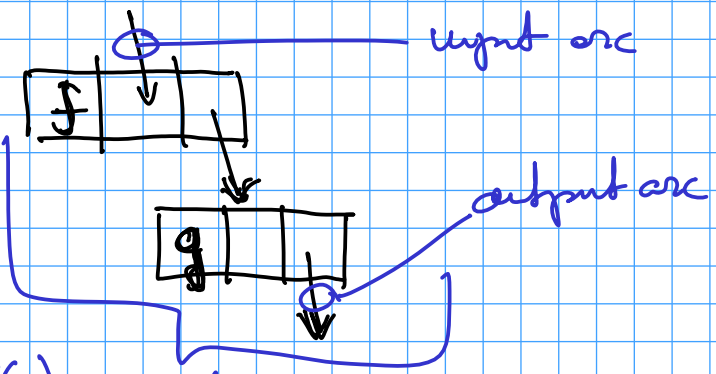
Processor



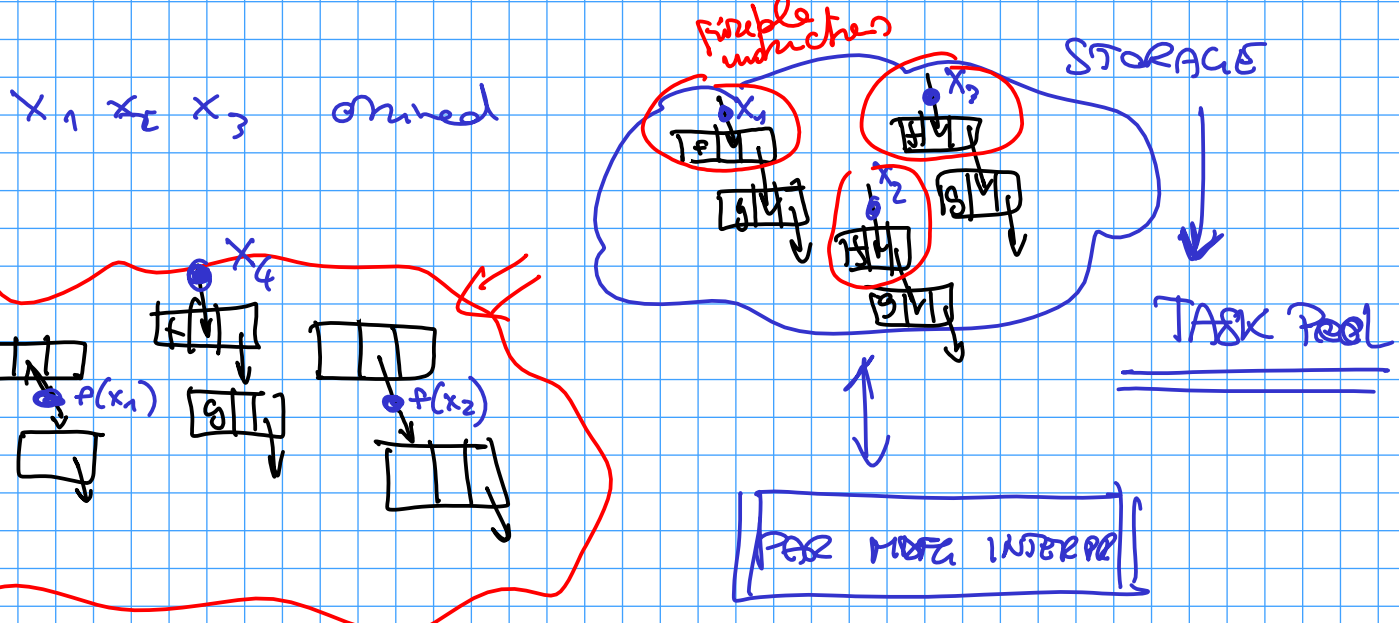
Macro DATA FLOW implementation of structured parallel programming frameworks



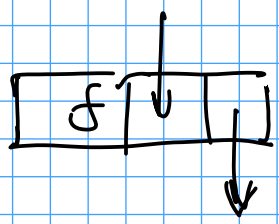
pipeline (f, g)



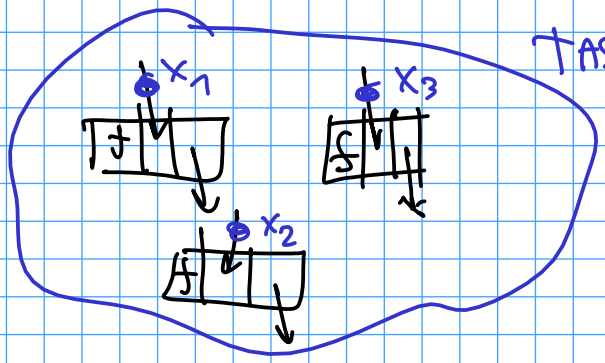
$\forall x_i$ on input → instance of $MDFG_i(pipe(f,g))$ with x_i as the input token



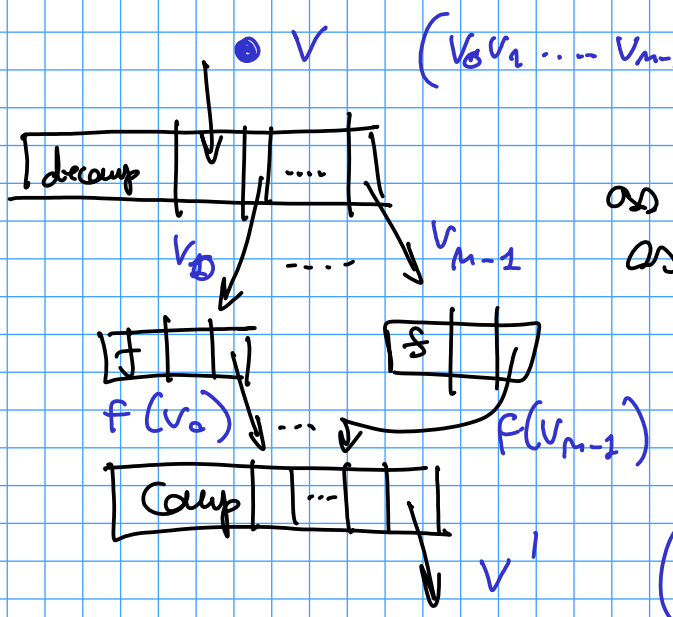
farm (f)



x_3 x_2 x_1



map (f)



as much arcs (output) as the items in the collection

LD R_{base}, R_{index}, R_{dest}

MEM[R_{base} + R_{index}] → R_{dest}

$W(LD) = \{R_{dest}\}$

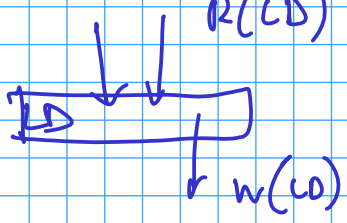
$R(LD) = \{R_{base}, R_{index}\}$

I_1 I_2

$W(I_1) \cap R(I_2) = \emptyset$

$R(I_1) \cap W(I_2) = \emptyset$

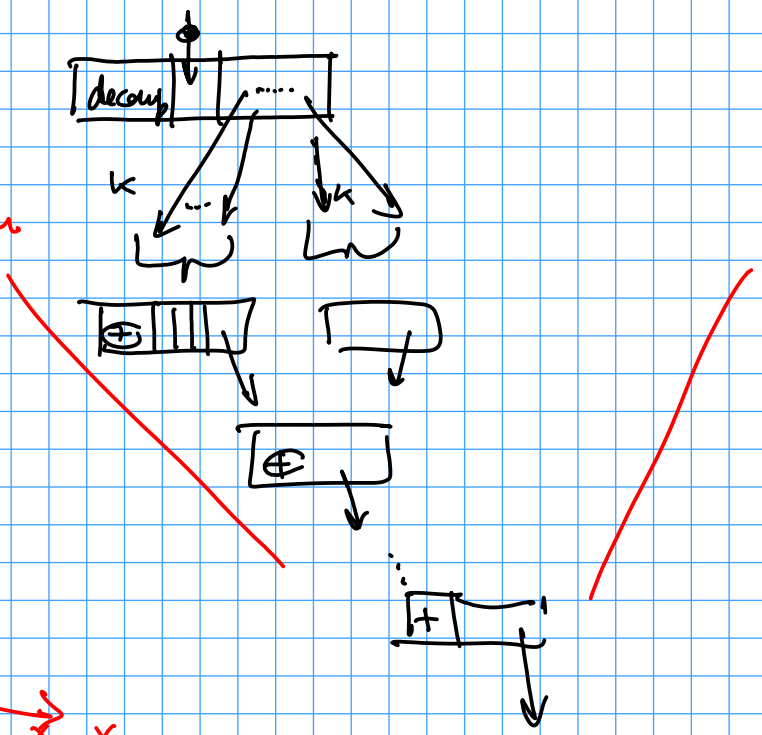
$W(I_1) \cap W(I_2) = \emptyset$



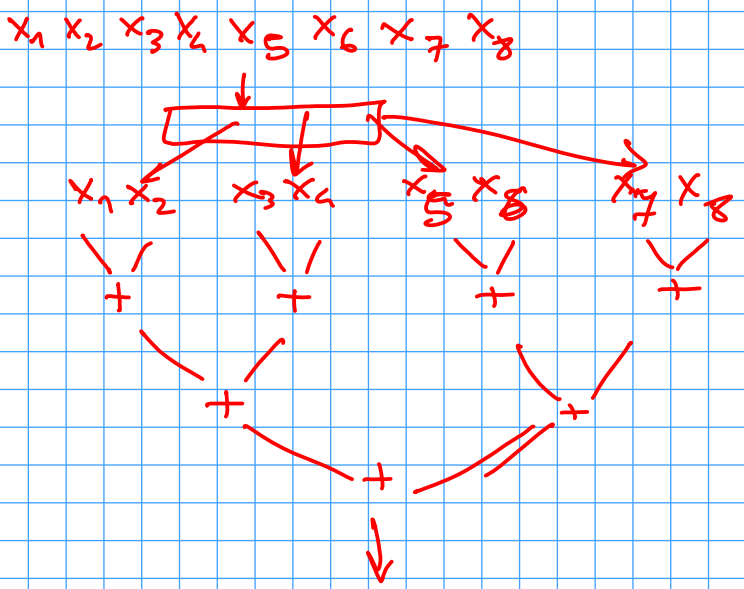
SCOREBOARDING

reduce $(\oplus)$

$\oplus$: k items → item

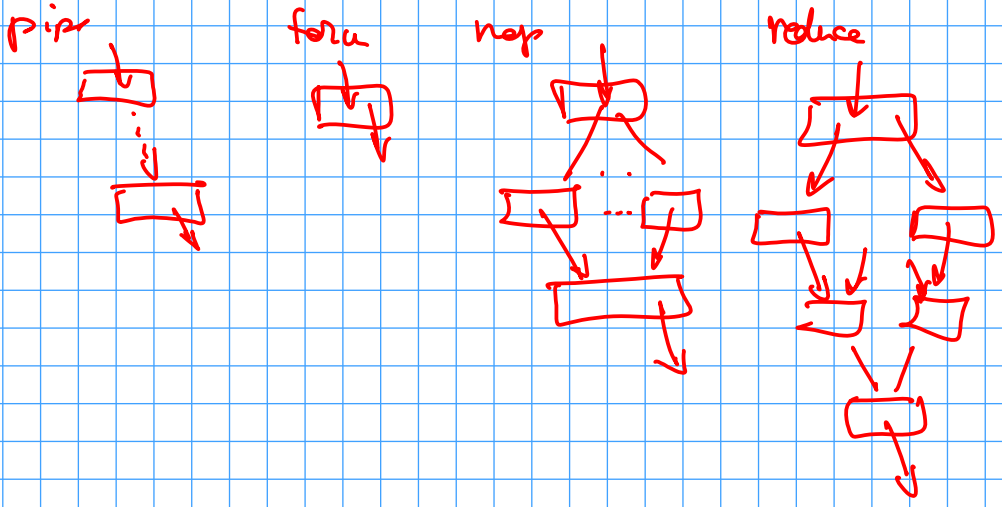


reduce $(+)$: vecteurs



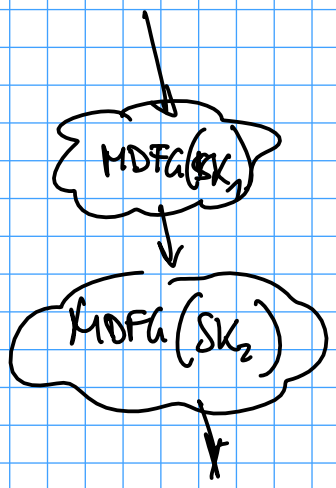
$\log_2 n$ niveaux

Composition



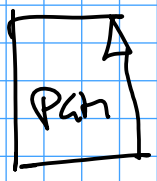
PIPE (SK_1 , SK_2)

FARM (PIPE (SK_1 , SK_2))

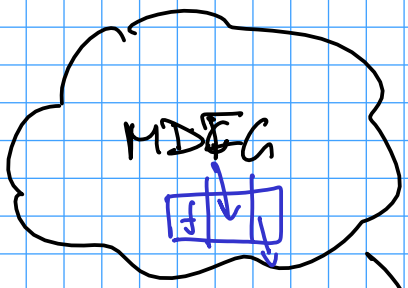


the same !

farm only introduces replication of the worker graph × input token/task

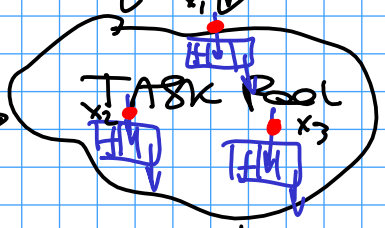


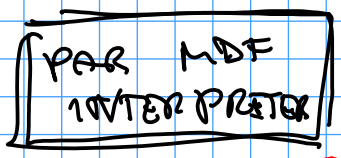
farm (P)



input stream

tokens directed to MDFG input arcs



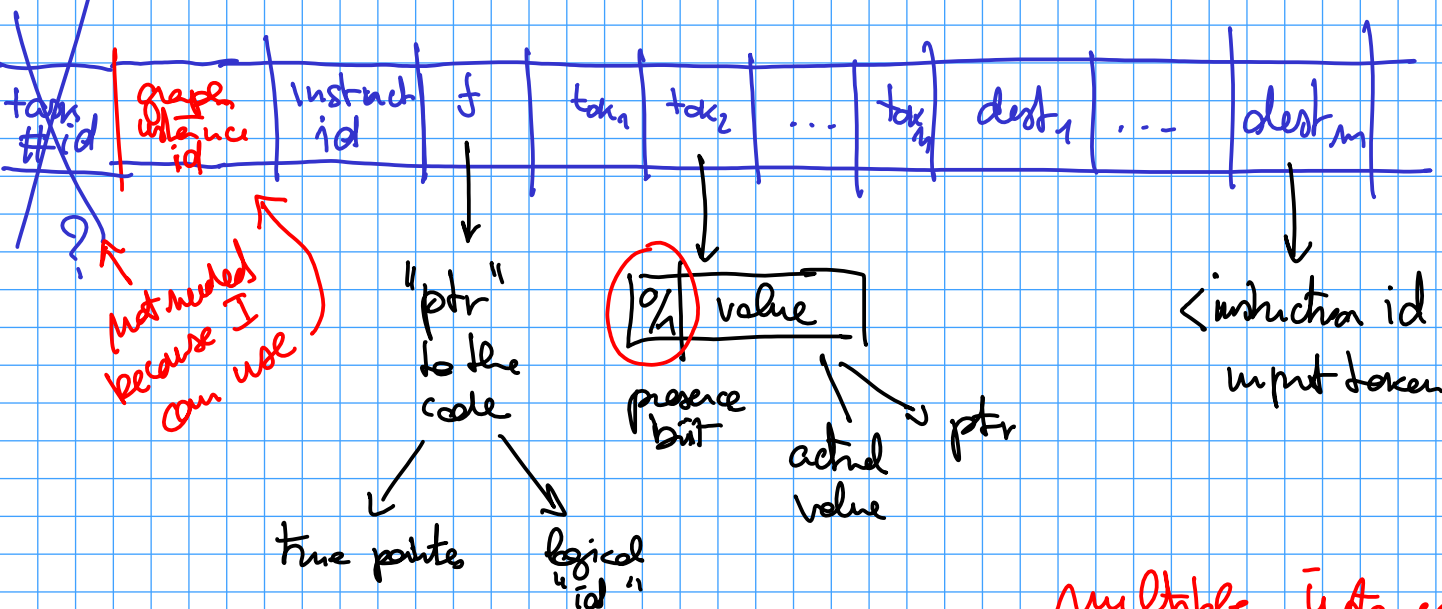


INPUT - OUTPUT ORDER IS IMPORTANT

output stream

tokens from MDFG output arcs

MDF instruction



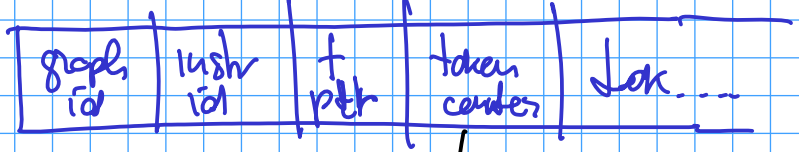
multiple instances of MDFG ⇒ instruction

write presence bit the arrival of a token ⇒

① → storing the value
② → updating the presence bit
③ → clear all presence bits to see if instruction is becomes fireable

OPTIMIZATION

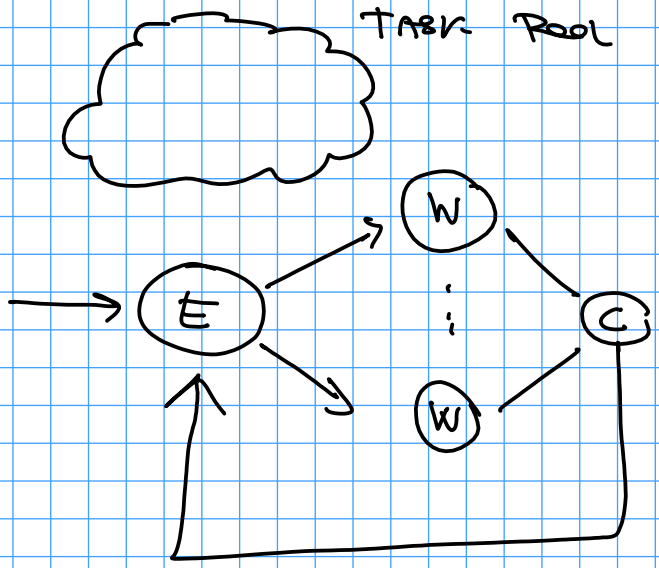
but we load a token counter to the instruction



init: # input tokens

decreased at each token arrival

if ∅ → fireable



E: ① accepts input tokens and creates instances of the MDFG in the task pool

② accepts results from the workers and stores into proper MDF instruction in the task pool

③ ∀ ① send the initial fizzable instruction to 1 worker

∀ ② leading to a fizzable instruction do the same

W: receives a fizzable instruction and computes it delivering output tokens to the feedback