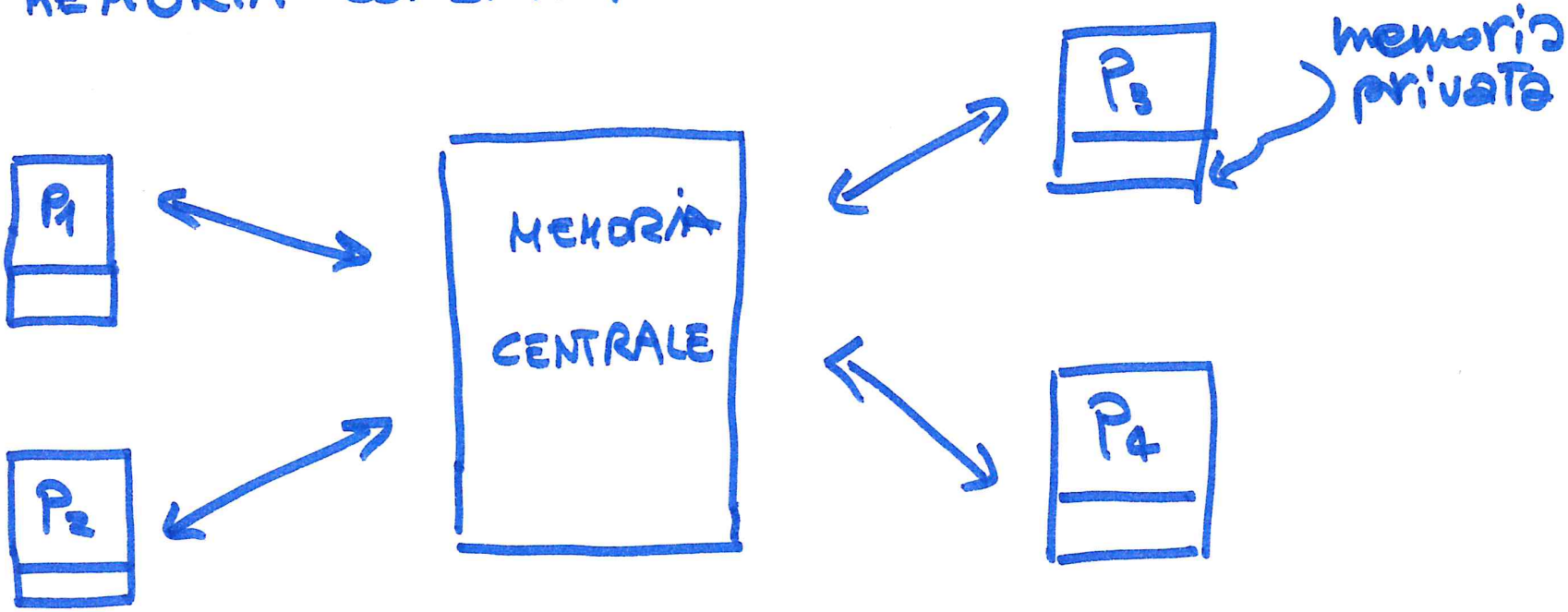
TIPI DI ARCHITETTURE PARALLELE

1 A MEMORIA CONDIVISA

2 A MEMORIA DISTRIBUITA

---

1 A MEMORIA CONDIVISA



Proprietà non visibile:

Abbiamo un unico clock centrale

Proprietà sulla comunicazione:

Comunicazione costante in Tempo Tra i processori

Cioè:

se P_j vuole comunic. una dato x a P_i:

- P_j scrive x in mem. centrale
- P_i legge x dalla mem. centrale

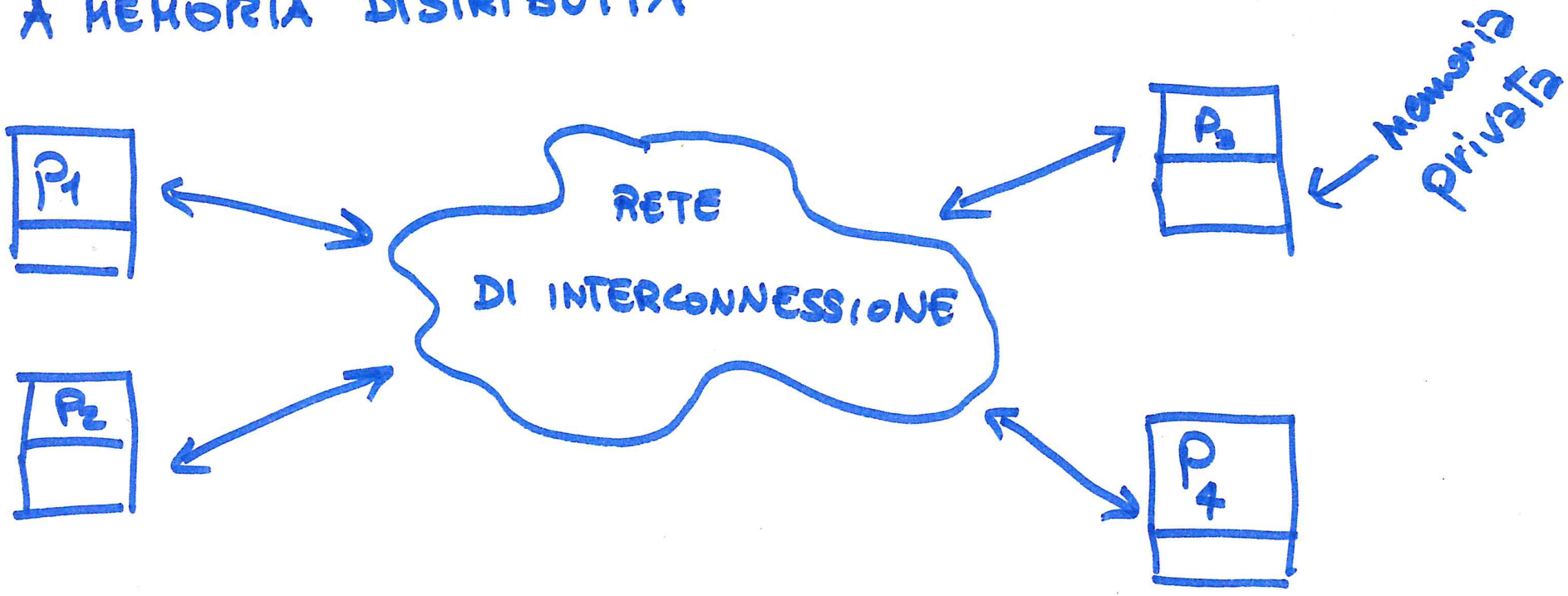
es: multicore attuali

Domanda:

Questa architettura permette una forte parallelizzazione?

Perchè?

2 A MEMORIA DISTRIBUITA



"DA NON CONFONDERE CON LE ARCHITETTURE DISTRIBUITE"

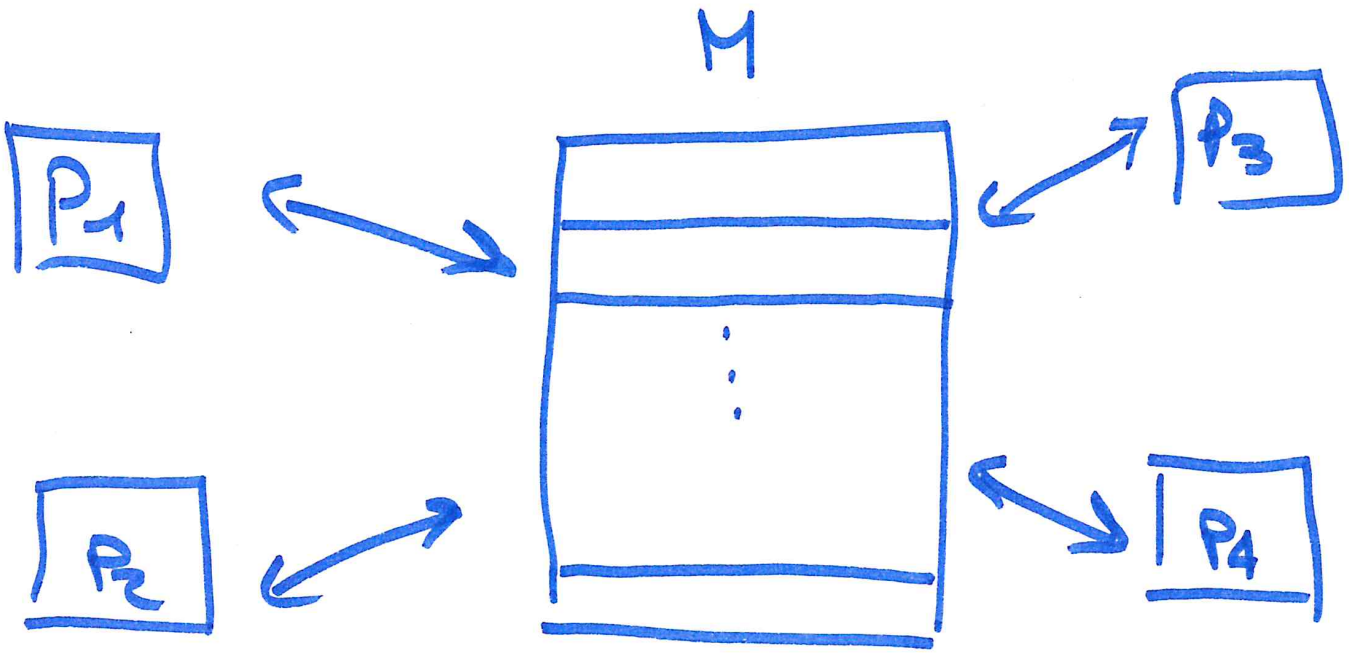
Proprietà non visibile:
Abbiamo un unico clock centrale

Proprietà sulla comunicazione:
La comunicazione dipende dalla "distanza" tra i processori: se P_j vuole comunicare con P_i ci dobbiamo chiedere quanti processori collegano P_j a P_i.

Proprietà
La rete di interconnessione è fissa:
- array lineare
- mesh
- ipercubo

Modello PRAM (Parallel RAM)

Teorico anche se oggi è attuale



celle

M[0]
M[1]
⋮
M[n]

Ogni P_i è una RAM sequenziale

memoria privata

Tipo di istruzioni dei P_i :

- operazioni aritmetico/logiche
- istruzioni da/per la memoria centrale :
  - STORE R[k] M[h]
  - LOAD R[k'] M[h']

N.B. operiamo solo sui dati nella memoria privata

Comunicazione

Esempio: P_j vuole comunicare con P_i il contenuto di R[t]

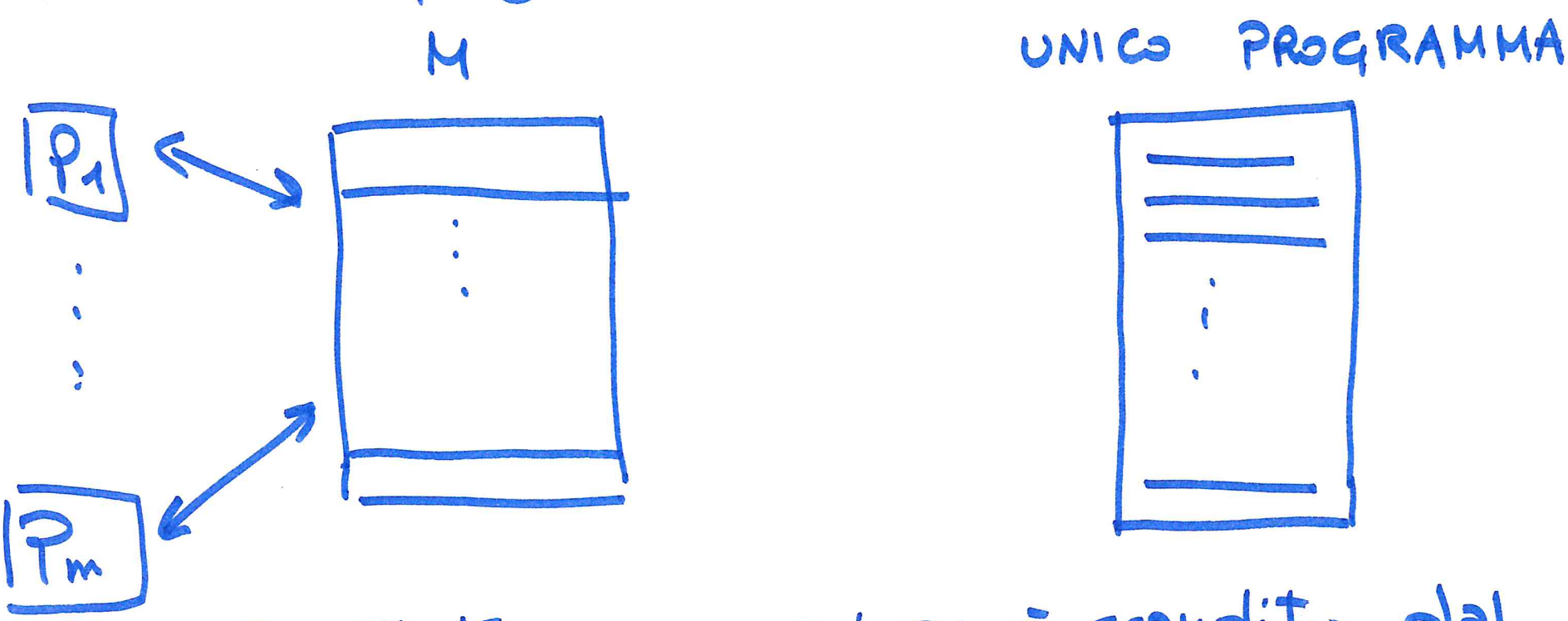
Istruzioni :

P_j : STORE R[t] M[x]

P_i : LOAD R[s] M[x]

La comunicazione avviene in Tempo $O(1)$

Come si programma una PRAM ?



① Il tempo per ogni P_i è scandito dal clock centrale

② Ogni P_i esegue la "stessa istruzione"

Forma dell'istruzione:

for all $i \in I$ par do:

istruzione$_i$

NOTA BENE:

- I processori con indice in I eseguono istruzione$_i$
- I processori con indice non in I eseguono l'istruzione nulla

istruzione$_i$ = ?

Risposta: ci sono due architetture diverse:

- SIMD : single instruction multiple data ←
- MIMD : multiple " " "

{ istruzione$_i$ / istruzione$_j$ } è quindi per $i \neq j$ la stessa istruzione ma su dati diversi che in genere dipendono dagli indici i e j

Risposta:
In funzione della capacità di accedere alla memoria M abbiamo anche qui diverse architetture:

EREW, CREW, CRCW dove

E = exclusive C = concurrent R = read W = write

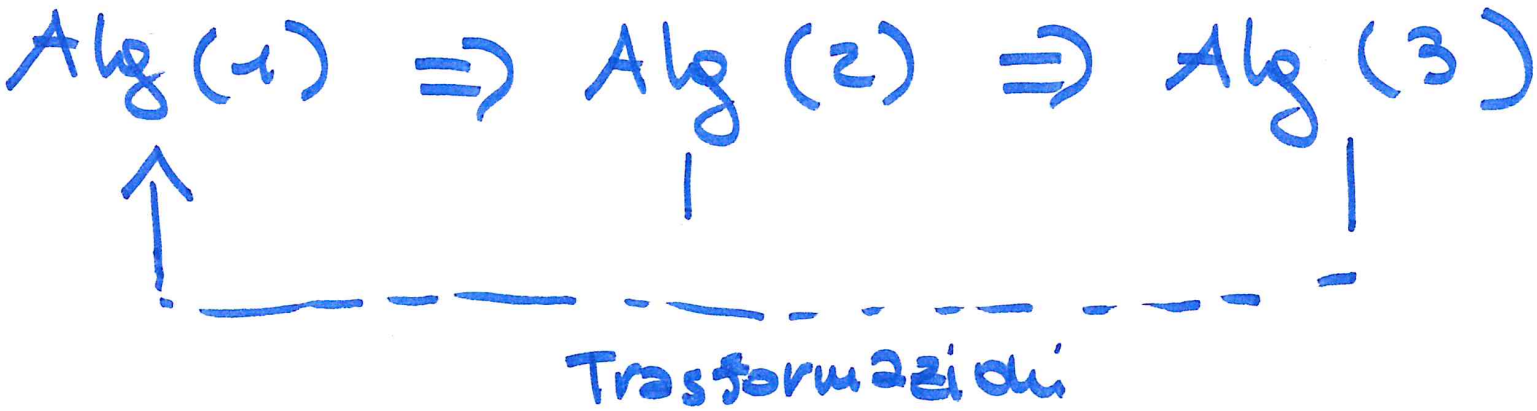
(1) EREW: no scrittura/lettura stessa cella di M

(2) CREW: lettura simultanea sì
scrittura simultanea no

(3) CRCW: sì scrittura/lettura simultanea
Per la scrittura simultanea abbiamo diverse politiche:

- common: i processori possono scrivere solo lo stesso dato pena arresto del sistema
- random: si sceglie un Pi a caso
- max/min: vince il Pi con il dato max/min
- priority: vince il Pi con priorità max

Osservazione:



L'architettura più ragionevole e più semplice è EREW

Risorse di calcolo

- sequenziale: $t(n)$, $s(n)$
- parallelo: $p(n)$, $T(n, p(n))$

Esempio di algoritmo su P-RAM di tipo CREW

- # di processori = n
- assumiamo l'input array A con valori tutti distinti

```
Cerca ( A, n, x)
{ indice = -1
   for  i = 0  to  n-1  parolo
P_i:        if  A[i] = x  then  indice = i
   return  indice
}
```

- Tempo parallelo = costante
- se A può contenere elementi ripetuti ⇒ CRCW

Definizioni informali

$P(n)$ = numero dei processori richiesti su input di lunghezza n (caso peggiore)

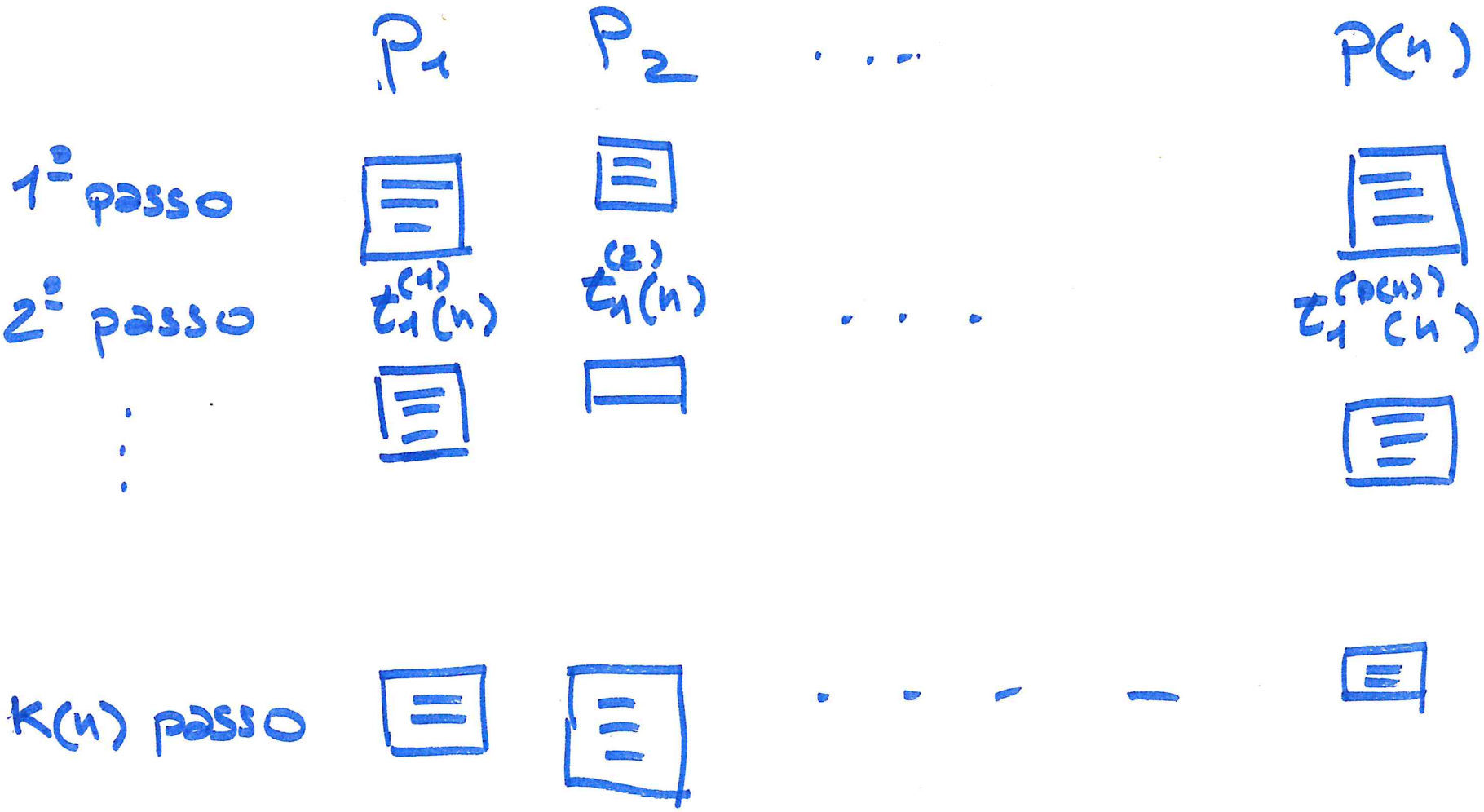
$T(n, P(n))$ = Tempo richiesto da un input di lunghezza n e $P(n)$ processori (caso peggiore)

Osservazione

$T(n, 1)$ = Tempo sequenziale

Valutazione precisa del Tempo $T(n, p(n))$

Programma parallelo:



NOTAZIONE

$$t_i(n) = \max \left\{ t_i^{(j)}(n) \;\middle|\; 1 \leq j \leq p(n) \right\}$$

Pertanto

$$T(n, p(n)) = \sum_{i=1}^{k(n)} t_i(n)$$

- T dipende da $k(n)$
- T dipende anche dalla dimensione dell'input
  [costo logaritmico / costo uniforme)
- T dipende da $p(n)$