



# Arhitecturi Paralele Complexitate + Măsurare Timp

Lect. Dr. Ing. Cristian Chilipirea – [cristian.chilipirea@mta.ro](mailto:cristian.chilipirea@mta.ro)







# Măsurarea timpului de execuție

time ./executabil p a r a m e t r i



# Măsurare timp – Linia de comandă

***time sleep 5***

real 0m5.001s  
user 0m0.000s  
sys 0m0.001s

***time sleep 5***

sleep 5 0.00s user 0.00s system 0% cpu 5.002 total

***/usr/bin/time sleep 5***

0.00user 0.00system 0:05.00elapsed 0%CPU (0avgtext+0avgdata 2076maxresident)k  
0inputs+0outputs (0major+73minor)pagefaults 0swaps



# Măsurare timp – Linia de comandă

***time sleep 5***

real 0m5.001s  
user 0m0.000s  
sys 0m0.001s

Wall clock time – Timpul trecut de la pornirea programului – Pe acesta îl folosim

***time sleep 5***

sleep 5 0.00s user 0.00s system 0% cpu 5.002 total

***/usr/bin/time sleep 5***

0.00user 0.00system 0:05.00elapsed 0%CPU (0avgtext+0avgdata 2076maxresident)k  
0inputs+0outputs (0major+73minor)pagefaults 0swaps



# Măsurare timp – Linia de comandă

```
time sleep 2
```

```
real  0m2.021s
user  0m0.000s
sys   0m0.000s
```

```
time sleep 2
```

```
real  0m2.018s
user  0m0.000s
sys   0m0.016s
```

```
time sleep 2
```

```
real  0m2.016s
user  0m0.000s
sys   0m0.000s
```

```
time sleep 2
```

```
real  0m2.015s
user  0m0.000s
sys   0m0.000s
```

Timpii mășurați nu sunt exacti.  
Pentru a măsura corect trebuie  
să facem medie a timpilor după  
mai multe rulări sau să  
considerăm doar timpi mari –  
peste o secundă.



# Măsurare timp – Linia de comandă

***time sleep 5***

real 0m5.001s  
user 0m0.000s  
sys 0m0.001s

**Suma** timpului petrecut  
în user space pe fiecare  
core.

***time sleep 5***

sleep 5 0.00s user 0.00s system 0% cpu 5.002 total

***/usr/bin/time sleep 5***

0.00user 0.00system 0:05.00elapsed 0%CPU (0avgtext+0avgdata 2076maxresident)k  
0inputs+0outputs (0major+73minor)pagefaults 0swaps



# Măsurare timp – Linia de comandă

**Suma** timpului petrecut  
în user space pe fiecare  
core.

```
time timeout 5 ./useAllCPU 12
```

```
real    0m4.075s  
user    0m47.797s  
sys     0m0.031s
```





# Măsurare timp – Linia de comandă

***time sleep 5***

real 0m5.001s  
user 0m0.000s  
sys 0m0.001s

**Suma timpului petrecut  
în kernel pe fiecare core.**

***time sleep 5***

sleep 5 0.00s user 0.00s system 0% cpu 5.002 total

***/usr/bin/time sleep 5***

0.00user 0.00system 0:05.00elapsed 0%CPU (0avgtext+0avgdata 2076maxresident)k  
0inputs+0outputs (0major+73minor)pagefaults 0swaps



# Măsurare timp – Linia de comandă

Orice I/O este făcut de Kernel

```
time dd if=/dev/zero of=file.txt count=1024 bs=1 048576  
1024+0 records in  
1024+0 records out  
1073741824 bytes (1.1 GB) copied, 9.4847 s, 113 MB/s
```

```
real 0m9.490s  
user 0m0.000s  
sys 0m0.992s
```



# Măsurare timp – Din program

```
clock_t t;  
t = clock();  
WORK();  
t = clock() - t;  
double time_taken = ((double)t)/CLOCKS_PER_SEC; // in seconds
```



# Măsurare timp – Din program

**DO NOT USE**

```
clock_t t;  
t = clock();  
WORK();  
t = clock() - t;  
double time_taken = ((double)t)/CLOCKS_PER_SEC; // in seconds
```



# Măsurare timp – Din program

```
clock_t t;  
t = clock();  
WORK();  
t = clock() - t;  
double time_taken = ((double)t)/CLOCKS_PER_SEC; // in seconds
```

From man: the value returned by **clock()** also includes the times of any children.



# Măsurare timp – Din program

```
struct timespec start, finish;  
double elapsed;  
clock_gettime(CLOCK_MONOTONIC, &start);  
WORK();  
clock_gettime(CLOCK_MONOTONIC, &finish);  
elapsed = (finish.tv_sec - start.tv_sec);  
elapsed += (finish.tv_nsec - start.tv_nsec) / 1000000000.0;
```



# Măsurare timp cu sau fără I/O?



# Performanța

- Timp de execuție
- Memorie ocupată
- Număr de procese (thread-uri)
- Scalabilitate
- Toleranță la defecte
- Cost





# Măsuri

- T - Timpul total necesar execuției programului paralel
- P - Numărul de procesoare utilizate
- S – Speedup
  - $S = \frac{G}{T}$
- G – Timp execuție cel mai rapid algoritm secvențial



- Costul  $C = T * P$

- Eficiența  $E = \frac{G}{C} = \frac{G}{TP} = \frac{S}{P}$



# Complexitate

9 6 9 4 2 7 6 5 6 ... 1

\* 3

27 18 27 12 6 21 18 15 18 ... 3

Complexitate secvențială?



# Complexitate



\* 3

27

Complexitate secvențială?



# Complexitate



\* 3



Complexitate secvențială?



# Complexitate



\* 3



Complexitate secvențială?



# Complexitate



\* 3



Complexitate secvențială?



# Complexitate

9 6 9 4 2 7 6 5 6 ... 1

\* 3

27 18 27 12 6 21 18 15 18 ... 3

Complexitate secvențială?  $O(N)$





# Complexitate



\* 3

Complexitate paralelă?



# Complexitate

9 6 9 4 2 7 6 5 6 ... 1

\* 3

27 18 27 12 6 21 18 15 18 ... 3

Complexitate paralelă? **O(1)** ?



# Complexitate

9 6 9 4 2 7 6 5 6 ... 1

\* 3

27 18 27 12 6 21 18 15 18 ... 3

Complexitate paralelă? **O(1)** ? **P = N**



# Complexitate

9	6	9	4	2	7	6	5	6	...	1
---	---	---	---	---	---	---	---	---	-----	---

\* 3

27	18	27	12	6	21	18	15	18	...	3
----	----	----	----	---	----	----	----	----	-----	---

Complexitate paralelă?  $O(\frac{N}{P})$



# Complexitate



Speedup?



# Complexitate



Speedup?

$$T = O\left(\frac{N}{P}\right)$$



# Complexitate



Speedup?

$$T = O\left(\frac{N}{P}\right)$$

$$G = O(N)$$



# Complexitate



Speedup?

$$T = O\left(\frac{N}{P}\right)$$
$$G = O(N)$$

$$S = \frac{O(N)}{O\left(\frac{N}{P}\right)}$$





# Complexitate



Speedup?

$$T = O\left(\frac{N}{P}\right)$$
$$G = O(N)$$

$$S = \frac{O(N)}{O\left(\frac{N}{P}\right)} = O(P)$$



# Complexitate



$$S = \frac{O(N)}{O(\frac{N}{P})} = O(P)$$

Eficiența?

$$T = O(\frac{N}{P})$$

$$G = O(N)$$



# Complexitate



$$S = \frac{O(N)}{O(\frac{N}{P})} = O(P)$$

Eficiența?

$$T = O(\frac{N}{P})$$
$$G = O(N)$$

$$E = \frac{S}{P} = \frac{O(P)}{P} = 1$$



# Timpi adunare a doi vectori $C = A + B$

Sequential		Pthread (1 Thread)		Pthread(2 Thread)		Pthread(4 Thread)	
real	0m6.151s	real	0m7.777s	real	0m3.954s	real	0m2.011s
user	0m6.141s	user	0m7.766s	user	0m7.828s	user	0m7.875s
sys	0m0.000s	sys	0m0.000s	sys	0m0.000s	sys	0m0.031s



# Timpi adunare a doi vectori $C = A + B$

Sequential		Pthread (1 Thread)		Pthread(2 Thread)		Pthread(4 Thread)	
real	0m6.151s	real	0m7.777s	real	0m3.954s	real	0m2.011s
user	0m6.141s	user	0m7.766s	user	0m7.828s	user	0m7.875s
sys	0m0.000s	sys	0m0.000s	sys	0m0.000s	sys	0m0.031s
S = 1							

$$S = \frac{6.14}{6.14}$$



# Timpi adunare a doi vectori $C = A + B$

Sequential		Pthread (1 Thread)		Pthread(2 Thread)		Pthread(4 Thread)	
real	0m6.151s	real	0m7.777s	real	0m3.954s	real	0m2.011s
user	0m6.141s	user	0m7.766s	user	0m7.828s	user	0m7.875s
sys	0m0.000s	sys	0m0.000s	sys	0m0.000s	sys	0m0.031s
S = 1		S = 0.8					

$$S = \frac{6.14}{7.76}$$



# Timpi adunare a doi vectori $C = A + B$

Sequential		Pthread (1 Thread)		Pthread(2 Thread)		Pthread(4 Thread)	
real	0m6.151s	real	0m7.777s	real	0m3.954s	real	0m2.011s
user	0m6.141s	user	0m7.766s	user	0m7.828s	user	0m7.875s
sys	0m0.000s	sys	0m0.000s	sys	0m0.000s	sys	0m0.031s
S = 1		S = 0.8		S = 0.78			

$$S = \frac{6.14}{7.78}$$



# Timpi adunare a doi vectori $C = A + B$

Sequential		Pthread (1 Thread)		Pthread(2 Thread)		Pthread(4 Thread)	
real	0m6.151s	real	0m7.777s	real	0m3.954s	real	0m2.011s
user	0m6.141s	user	0m7.766s	user	0m7.828s	user	0m7.875s
sys	0m0.000s	sys	0m0.000s	sys	0m0.000s	sys	0m0.031s
S = 1		S = 0.8		S = 0.78		S = 0.78	

$$S = \frac{6.14}{7.78}$$





# Timpi adunare a doi vectori $C = A + B$

Sequential		Pthread (1 Thread)		Pthread(2 Thread)		Pthread(4 Thread)	
real	0m6.151s	real	0m7.777s	real	0m3.954s	real	0m2.011s
user	0m6.141s	user	0m7.766s	user	0m7.828s	user	0m7.875s
sys	0m0.000s	sys	0m0.000s	sys	0m0.000s	sys	0m0.031s
S = 1		S = 0.8		S = 0.78		S = 0.78	

# WRONG

$$S = \frac{6.14}{7.78}$$



# Timpi adunare a doi vectori $C = A + B$

Sequential		Pthread (1 Thread)		Pthread(2 Thread)		Pthread(4 Thread)	
real	0m6.151s	real	0m7.777s	real	0m3.954s	real	0m2.011s
user	0m6.141s	user	0m7.766s	user	0m7.828s	user	0m7.875s
sys	0m0.000s	sys	0m0.000s	sys	0m0.000s	sys	0m0.031s
S = 1							

$$S = \frac{6.15}{6.15}$$



# Timpi adunare a doi vectori $C = A + B$

Sequential		Pthread (1 Thread)		Pthread(2 Thread)		Pthread(4 Thread)	
real	0m6.151s	real	0m7.777s	real	0m3.954s	real	0m2.011s
user	0m6.141s	user	0m7.766s	user	0m7.828s	user	0m7.875s
sys	0m0.000s	sys	0m0.000s	sys	0m0.000s	sys	0m0.031s
S = 1		S = 0.8					

$$S = \frac{6.15}{7.77}$$



# Timpi adunare a doi vectori $C = A + B$

Sequential		Pthread (1 Thread)		Pthread(2 Thread)		Pthread(4 Thread)	
real	0m6.151s	real	0m7.777s	real	0m3.954s	real	0m2.011s
user	0m6.141s	user	0m7.766s	user	0m7.828s	user	0m7.875s
sys	0m0.000s	sys	0m0.000s	sys	0m0.000s	sys	0m0.031s
S = 1		S = 0.8		S = 1.55			

$$S = \frac{6.15}{3.95}$$



# Timpi adunare a doi vectori $C = A + B$

Sequential		Pthread (1 Thread)		Pthread(2 Thread)		Pthread(4 Thread)	
real	0m6.151s	real	0m7.777s	real	0m3.954s	real	0m2.011s
user	0m6.141s	user	0m7.766s	user	0m7.828s	user	0m7.875s
sys	0m0.000s	sys	0m0.000s	sys	0m0.000s	sys	0m0.031s
S = 1		S = 0.8		S = 1.55		S = 3.05	

$$S = \frac{6.15}{2.01}$$



# Timpi adunare a doi vectori $C = A + B$

Sequential		Pthread (1 Thread)		Pthread(2 Thread)		Pthread(4 Thread)	
real	0m6.151s	real	0m7.777s	real	0m3.954s	real	0m2.011s
user	0m6.141s	user	0m7.766s	user	0m7.828s	user	0m7.875s
sys	0m0.000s	sys	0m0.000s	sys	0m0.000s	sys	0m0.031s
S = 1		S = 0.8		S = 1.55		S = 3.05	

De ce nu **S = P** ?



# Timpi adunare a doi vectori $C = A + B$

**Nu se ține cont de timpul de citire/scriere RAM.**

Sequential		Pthread (1 Thread)		Pthread(2 Thread)		Pthread(4 Thread)	
real	0m6.151s	real	0m7.777s	real	0m3.954s	real	0m2.011s
user	0m6.141s	user	0m7.766s	user	0m7.828s	user	0m7.875s
sys	0m0.000s	sys	0m0.000s	sys	0m0.000s	sys	0m0.031s
S = 1		S = 0.8		S = 1.55		S = 3.05	

De ce nu  **$S = P$**  ?  **$S = O(P)$**  este **ideal**.



# Timpi adunare a doi vectori $C = A + B$

$$E = \frac{1}{1}$$

Sequential		Pthread (1 Thread)		Pthread(2 Thread)		Pthread(4 Thread)	
real	0m6.151s	real	0m7.777s	real	0m3.954s	real	0m2.011s
user	0m6.141s	user	0m7.766s	user	0m7.828s	user	0m7.875s
sys	0m0.000s	sys	0m0.000s	sys	0m0.000s	sys	0m0.031s
S = 1		S = 0.8		S = 1.55		S = 3.05	
E = 1							





# Timpi adunare a doi vectori $C = A + B$

$$E = \frac{0.8}{1}$$

Sequential		Pthread (1 Thread)		Pthread(2 Thread)		Pthread(4 Thread)	
real	0m6.151s	real	0m7.777s	real	0m3.954s	real	0m2.011s
user	0m6.141s	user	0m7.766s	user	0m7.828s	user	0m7.875s
sys	0m0.000s	sys	0m0.000s	sys	0m0.000s	sys	0m0.031s
S = 1		S = 0.8		S = 1.55		S = 3.05	
E = 1		E = 0.8					



# Timpi adunare a doi vectori $C = A + B$

$$E = \frac{1.55}{2}$$

Sequential		Pthread (1 Thread)		Pthread(2 Thread)		Pthread(4 Thread)	
real	0m6.151s	real	0m7.777s	real	0m3.954s	real	0m2.011s
user	0m6.141s	user	0m7.766s	user	0m7.828s	user	0m7.875s
sys	0m0.000s	sys	0m0.000s	sys	0m0.000s	sys	0m0.031s
S = 1		S = 0.8		S = 1.55		S = 3.05	
E = 1		E = 0.8		E = 0.77			



# Timpi adunare a doi vectori $C = A + B$

$$E = \frac{3.05}{4}$$

Sequential		Pthread (1 Thread)		Pthread(2 Thread)		Pthread(4 Thread)	
real	0m6.151s	real	0m7.777s	real	0m3.954s	real	0m2.011s
user	0m6.141s	user	0m7.766s	user	0m7.828s	user	0m7.875s
sys	0m0.000s	sys	0m0.000s	sys	0m0.000s	sys	0m0.031s
S = 1		S = 0.8		S = 1.55		S = 3.05	
E = 1		E = 0.8		E = 0.77		E = 0.76	



# Amdahl's law

## Validity of the single processor approach to achieving large scale computing capabilities<sup>1</sup>

Gene M. Amdahl

IBM Sunnyvale, California

### 1 INTRODUCTION

For over a decade prophets have voiced the contention that the old approach has reached its limits and that truly significant advances can be made only by the multiplicity of computers in such a manner as to permit cooperative action. One direction has been pointed out as general purpose computers with large amounts of memories, or as specialized computers with geometrically related functions controlled by one or more instruction streams.





# Amdahl's law - prerequisites

$f$  - Procent de operații din algoritmul secvențial care se pot executa paralel.

$(1 - f)$  – Procentul de operații din algoritmul secvențial ce **NU** se pot executa paralel

$$T = (1 - f)G + f \frac{G}{P}$$

$$S = \frac{G}{T}$$



# Amdahl's law - prerequisites

$f$  - Procent de operații din algoritmul secvențial care se pot executa paralel.

$(1 - f)$  – Procentul de operații din algoritmul secvențial ce **NU** se pot executa paralel

$$T = (1 - f)G + f \frac{G}{P}$$

$$S = \frac{G}{T} = \frac{G}{(1 - f)G + \frac{fG}{P}}$$



# Amdahl's law

$$S = \frac{1}{(1 - f) + \frac{f}{P}}$$



# Amdahl's law

$$S = \frac{1}{(1 - f) + \frac{f}{P}}$$

Ce se întâmplă dacă P e foarte mare (chiar mai mare decât N)?





# Amdahl's law

$$S = \frac{1}{(1 - f) + \frac{f}{P}}$$

Ce se întâmplă dacă  $P$  e foarte mare (chiar mai mare decât  $N$ )?

$$S \leq \frac{1}{1 - f}$$



# Amdahl's law - prerequisites

$f$  - Procent de operații din algoritmul secvențial care se pot executa paralel.

$(1 - f)$  – Procentul de operații din algoritmul secvențial ce **NU** se pot executa paralel

$$T = (1 - f)G + f \frac{G}{P}$$

Ce se întâmplă dacă  $P$  e foarte mare (chiar mai mare decât  $N$ )?



# Amdahl's law - prerequisites

$f$  - Procent de operații din algoritmul secvențial care se pot executa paralel.

$(1 - f)$  – Procentul de operații din algoritmul secvențial ce **NU** se pot executa paralel

$$T = (1 - f)G + f \frac{G}{P}$$

Ce se întâmplă dacă  $P$  e foarte mare (chiar mai mare decât  $N$ )?

$$\mathbf{T \geq (1 - f)G}$$





## Pauză

```
int a = 0
co [Tid=1 to 2]
{
    for(int i = 0; i < 10000; i++)
        a = a + 2
}
print(a)
```

**Care este valoarea minimă ce poate fi printată?**





# Amdahl's law

Secvență Ne-paralelizabilă

Secvență Paralelizabilă



Secvență Ne-paralelizabilă

Secvență Paralelizată



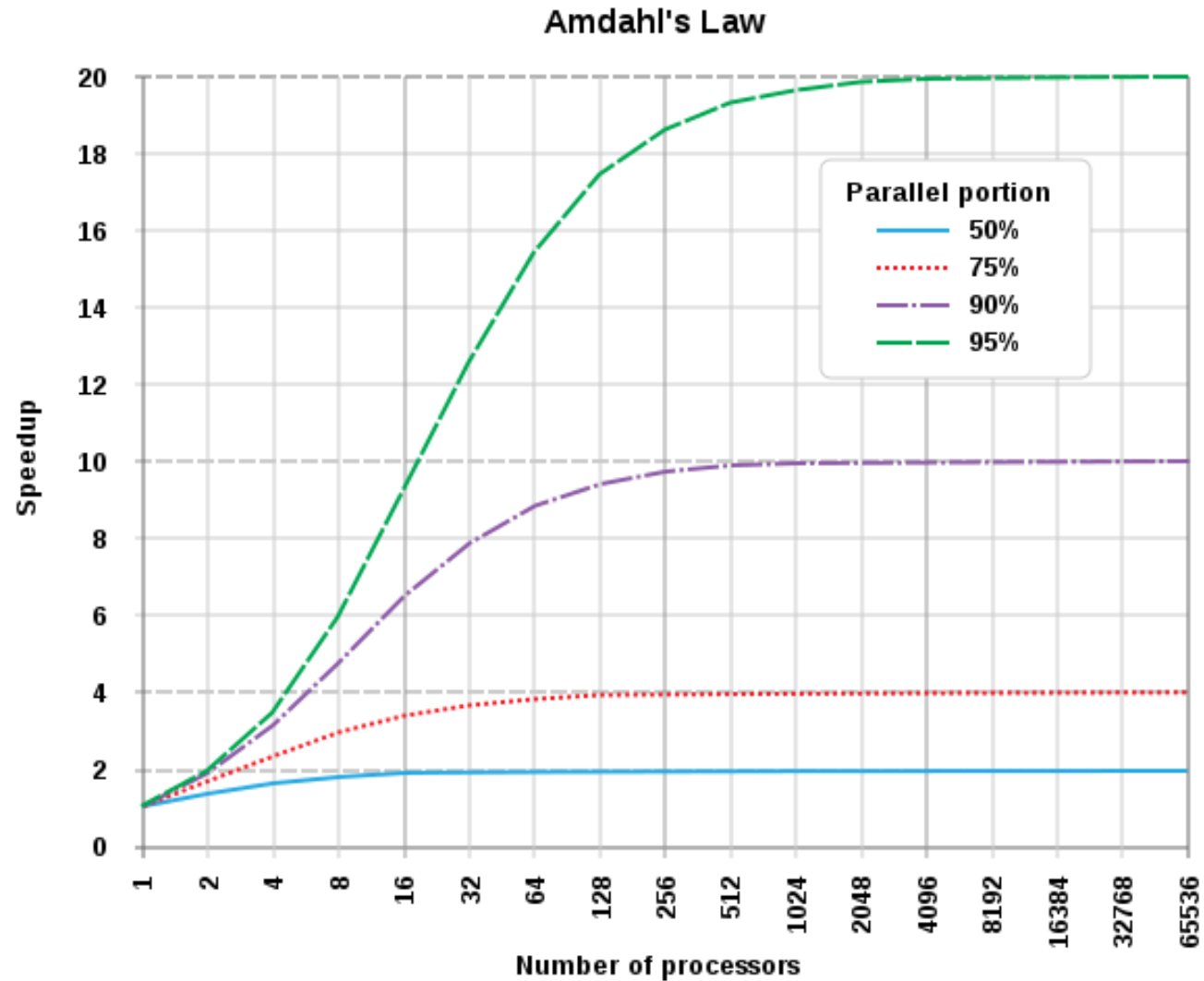


# Măsurare timp cu sau fără I/O?





# Amdahl's law





# Amdahl's law

$$S = \frac{1}{(1 - f) + \frac{f}{P}}$$





## Reminder

- Ce este un semafor?



- Ce este un mutex?

- Ce este o barieră?



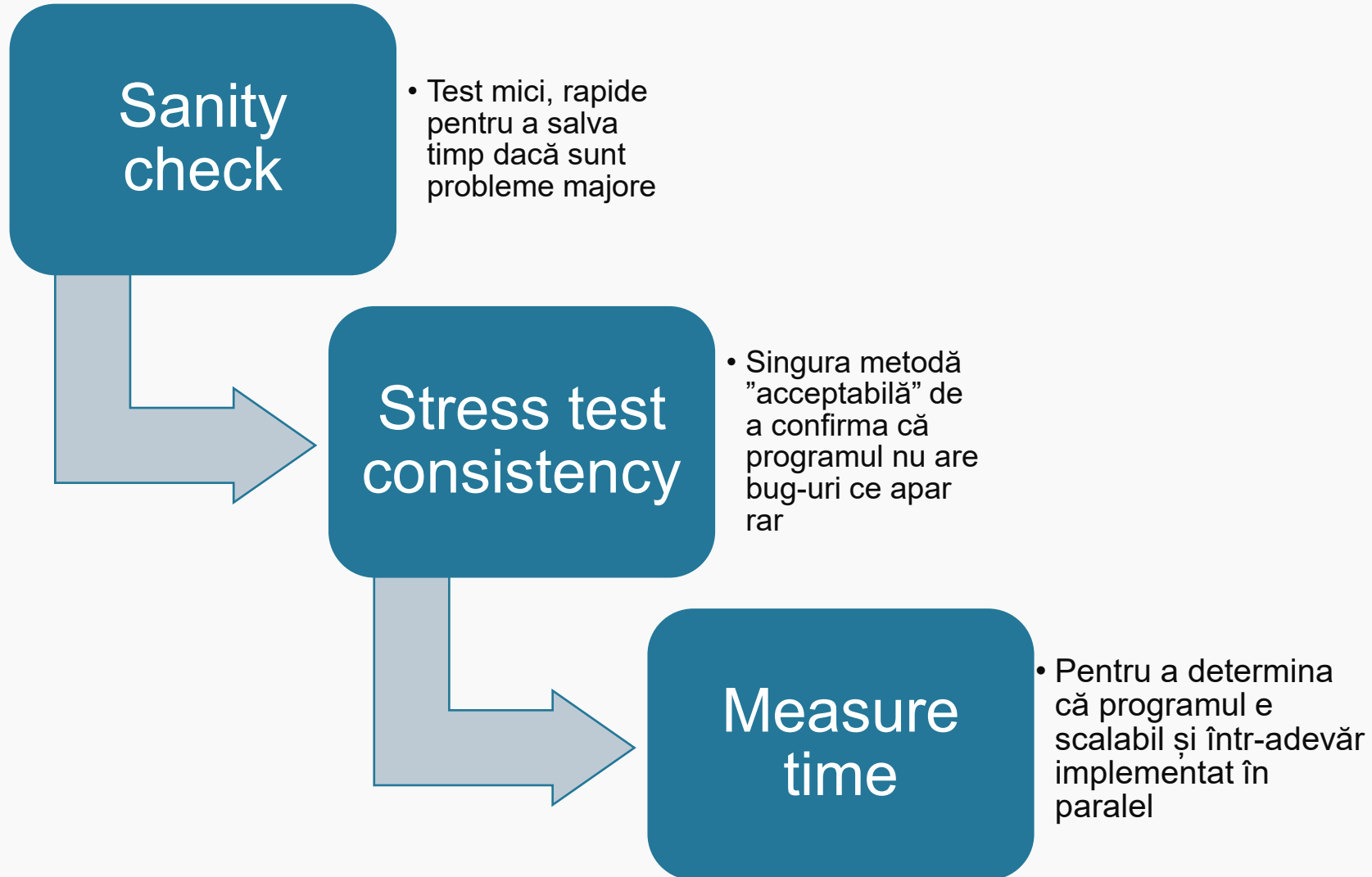


# Consistența?

- Demonstrație formală
- Stres test
  - ▣ mereu comparați cu rezultatul algoritmului secvențial sau cu un program de care sunteți siguri că oferă rezultat corect



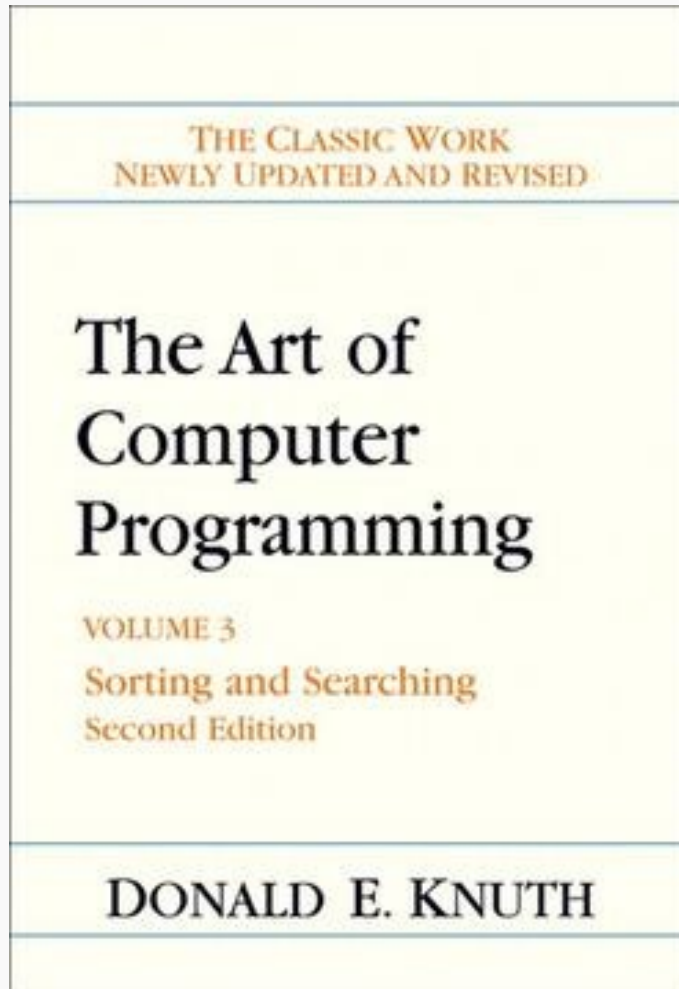
# Workflow - Testarea programelor







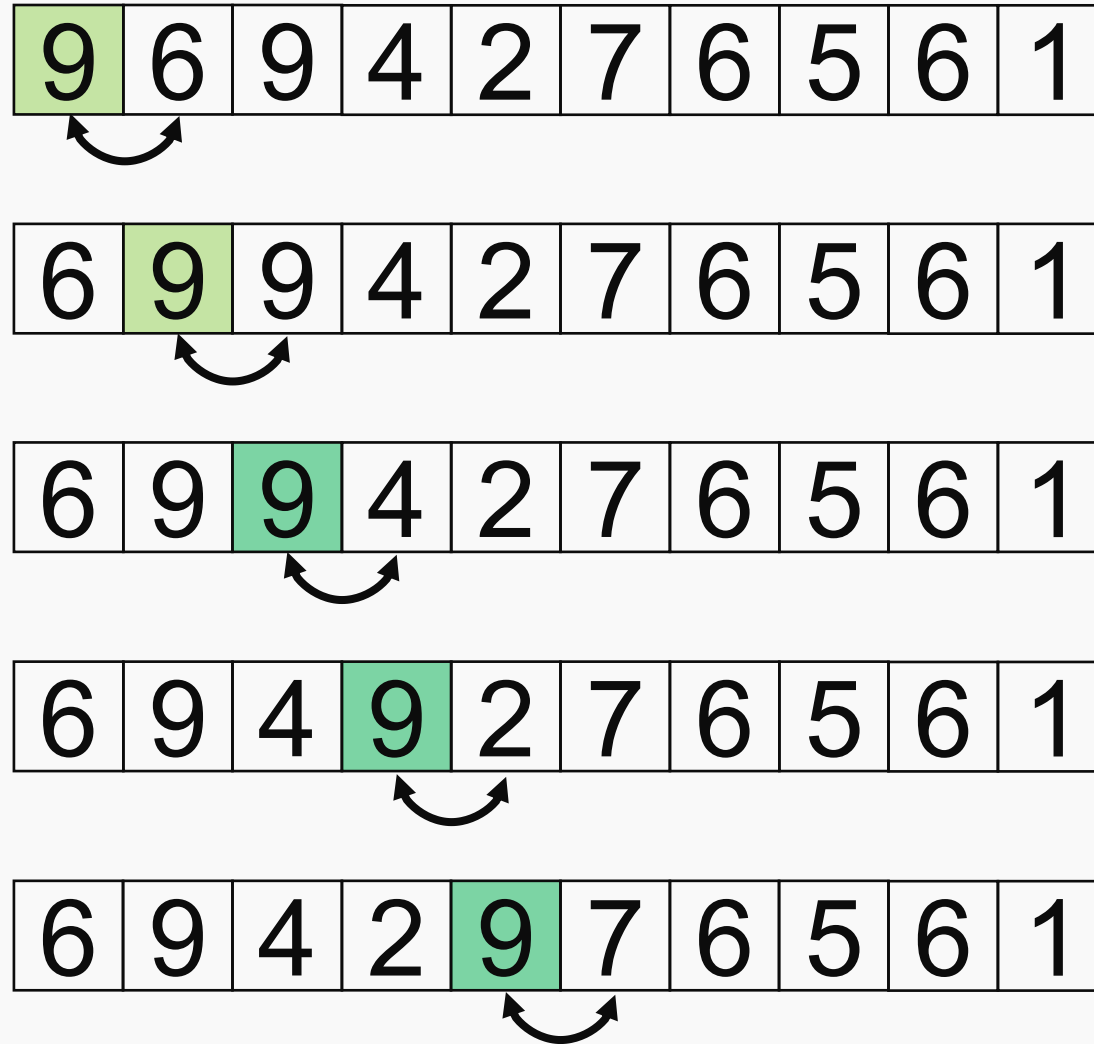
# Let's parallelize some algorithms





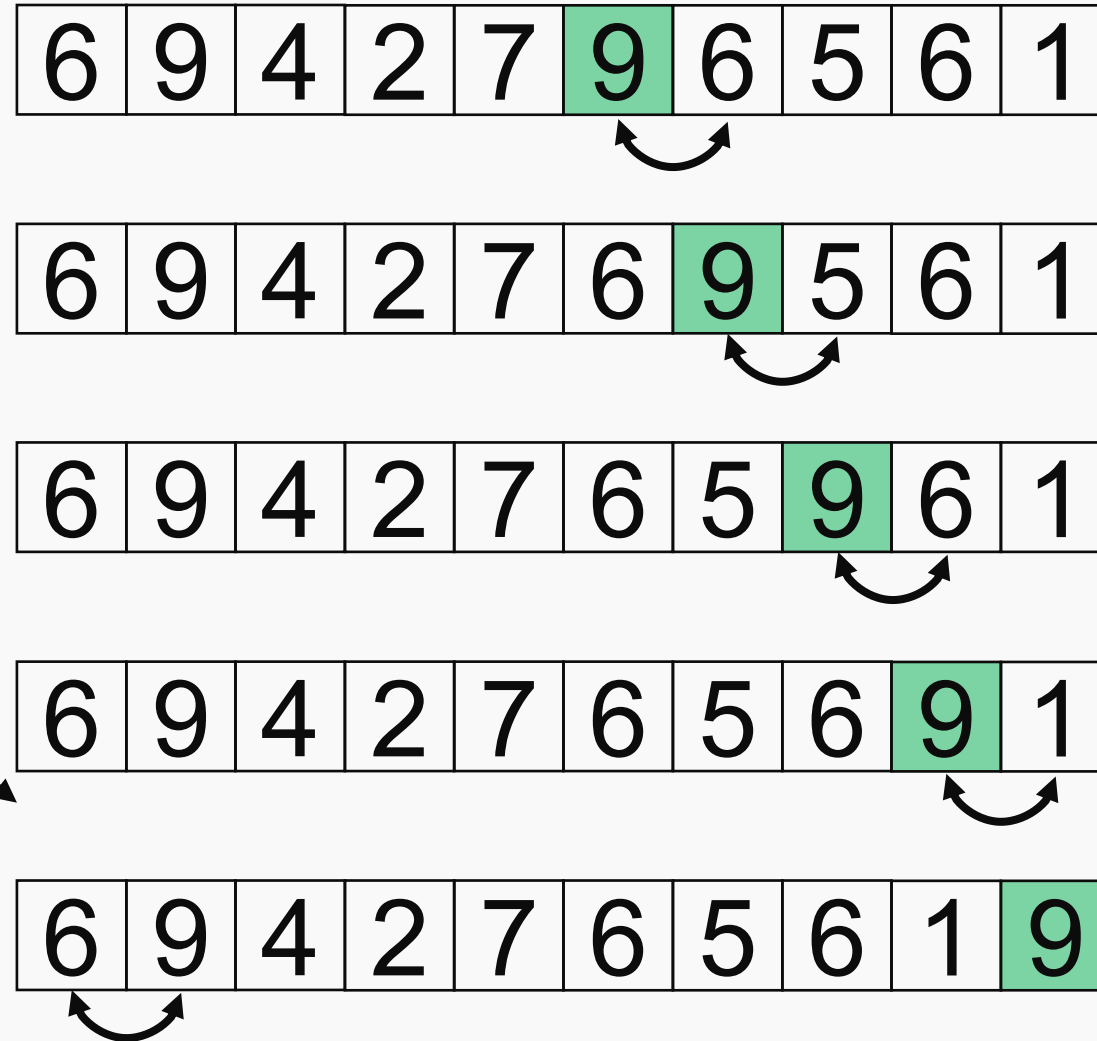


# Bubble sort





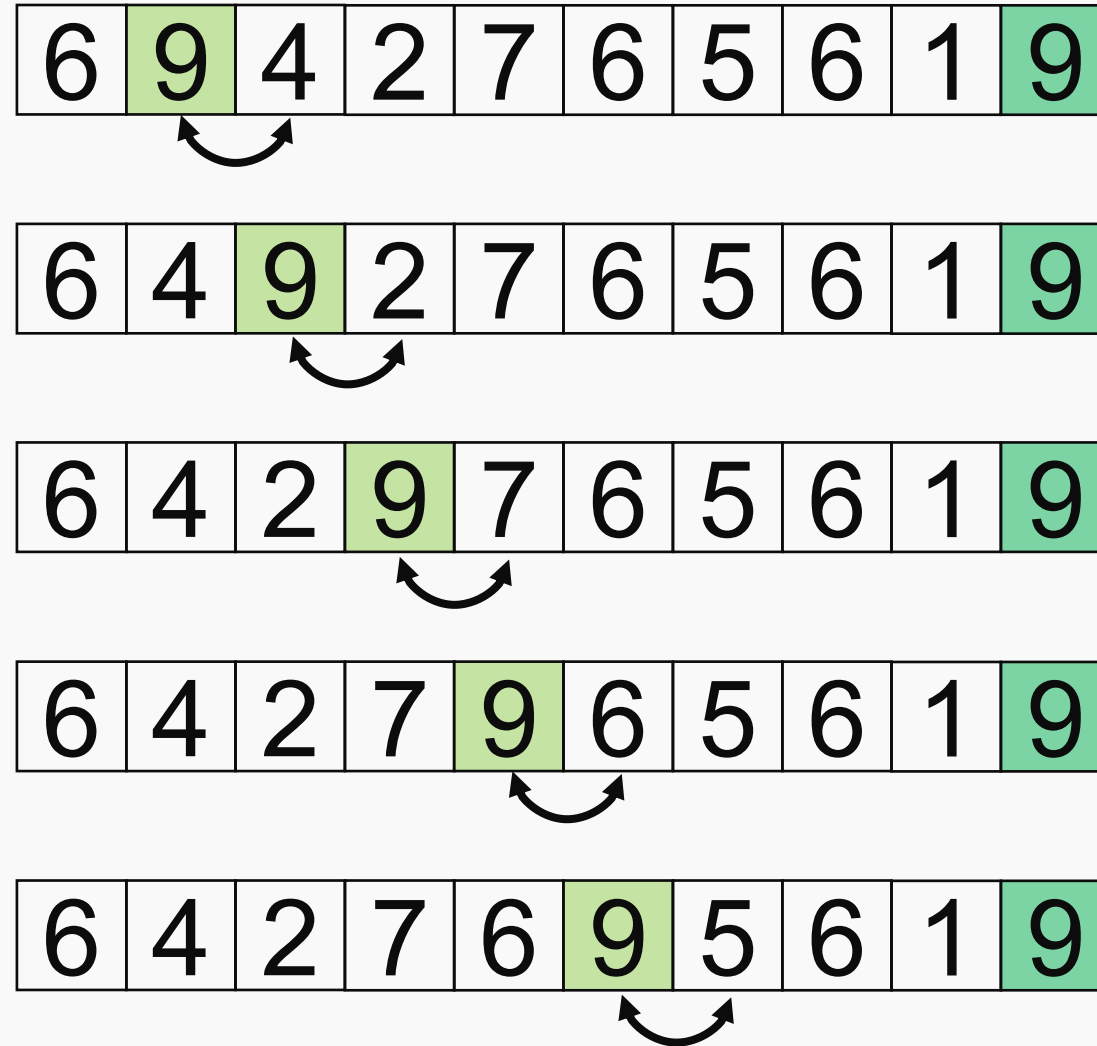
# Bubble sort



Repeat  
until  
sorted



# Bubble sort





# Bubble sort

6	4	2	7	6	5	9	6	1	9
---	---	---	---	---	---	---	---	---	---



6	4	2	7	6	5	6	9	1	9
---	---	---	---	---	---	---	---	---	---



6	4	2	7	6	5	6	1	9	9
---	---	---	---	---	---	---	---	---	---



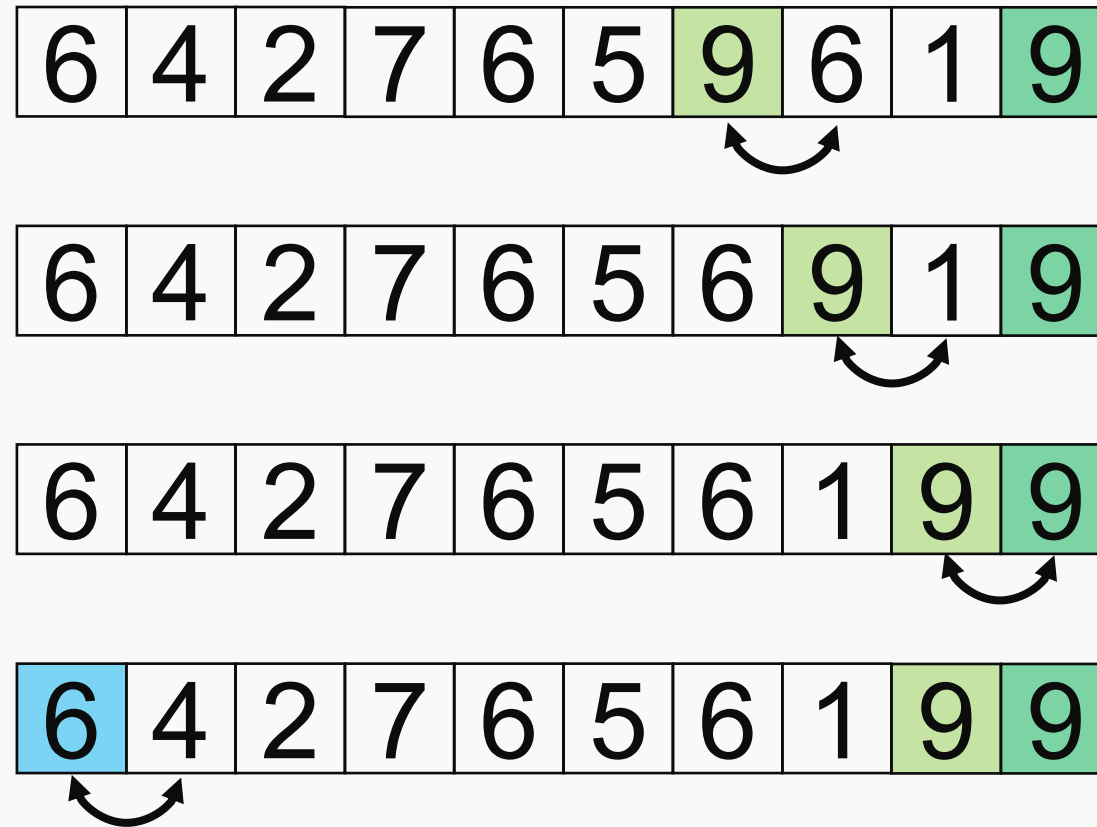
6	4	2	7	6	5	6	1	9	9
---	---	---	---	---	---	---	---	---	---



.....



## Bubble sort



Complexitate?



# Bubble sort

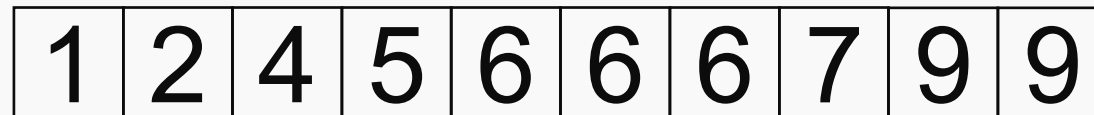


.....

Complexitate:  $O(N^2)$

Un pas trece prin toate elementele

Garantat să termine după n repetiții

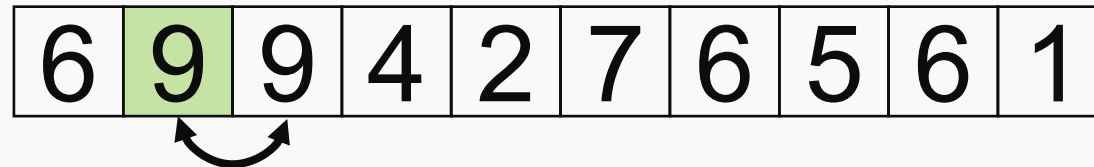




# Parallel bubble sort



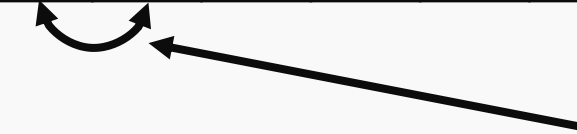
Cum paralelizăm?





# Parallel bubble sort

9	6	9	4	2	7	6	5	6	1
---	---	---	---	---	---	---	---	---	---



Aceste două operații (și toate perechile similare)  
**NU** pot fi executate în același timp

6	9	9	4	2	7	6	5	6	1
---	---	---	---	---	---	---	---	---	---

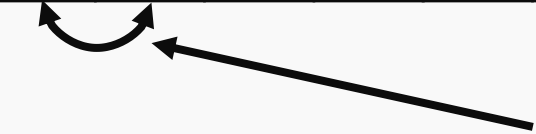






# Parallel bubble sort

9	6	9	4	2	7	6	5	6	1
---	---	---	---	---	---	---	---	---	---



Aceste două operații (și toate perechile similare)  
**NU** pot fi executate în același timp

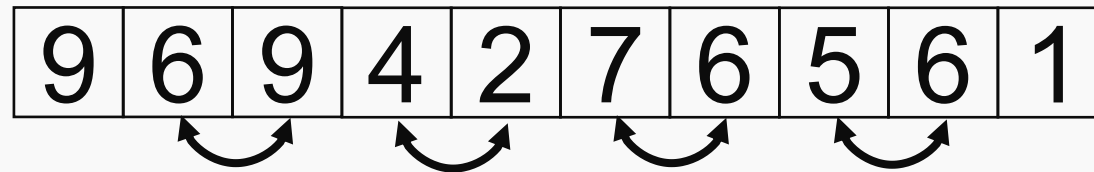
6	9	9	4	2	7	6	5	6	1
---	---	---	---	---	---	---	---	---	---



Hint: Nu este necesar ca operațiile să fie  
executate în această ordine



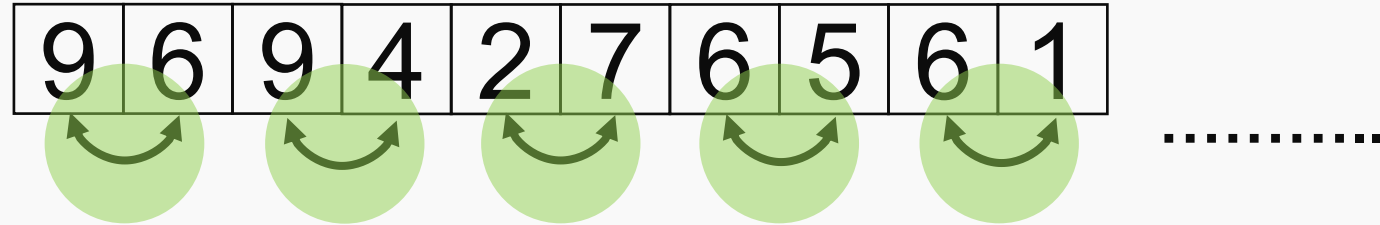
# OETS: Odd-Even Transposition Sort



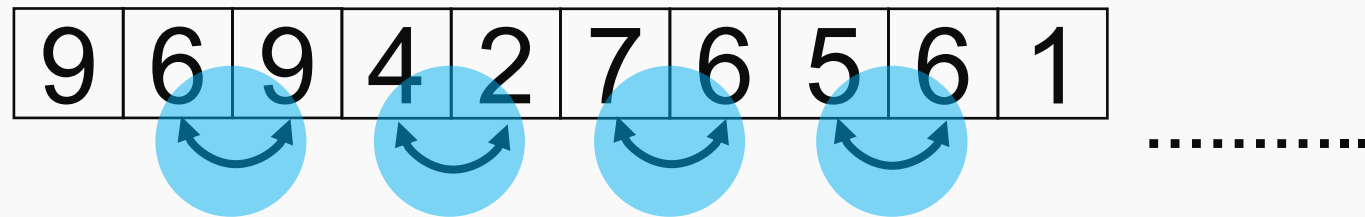


# OETS: Odd-Even Transposition Sort

Pot fi  
executate  
în paralel →



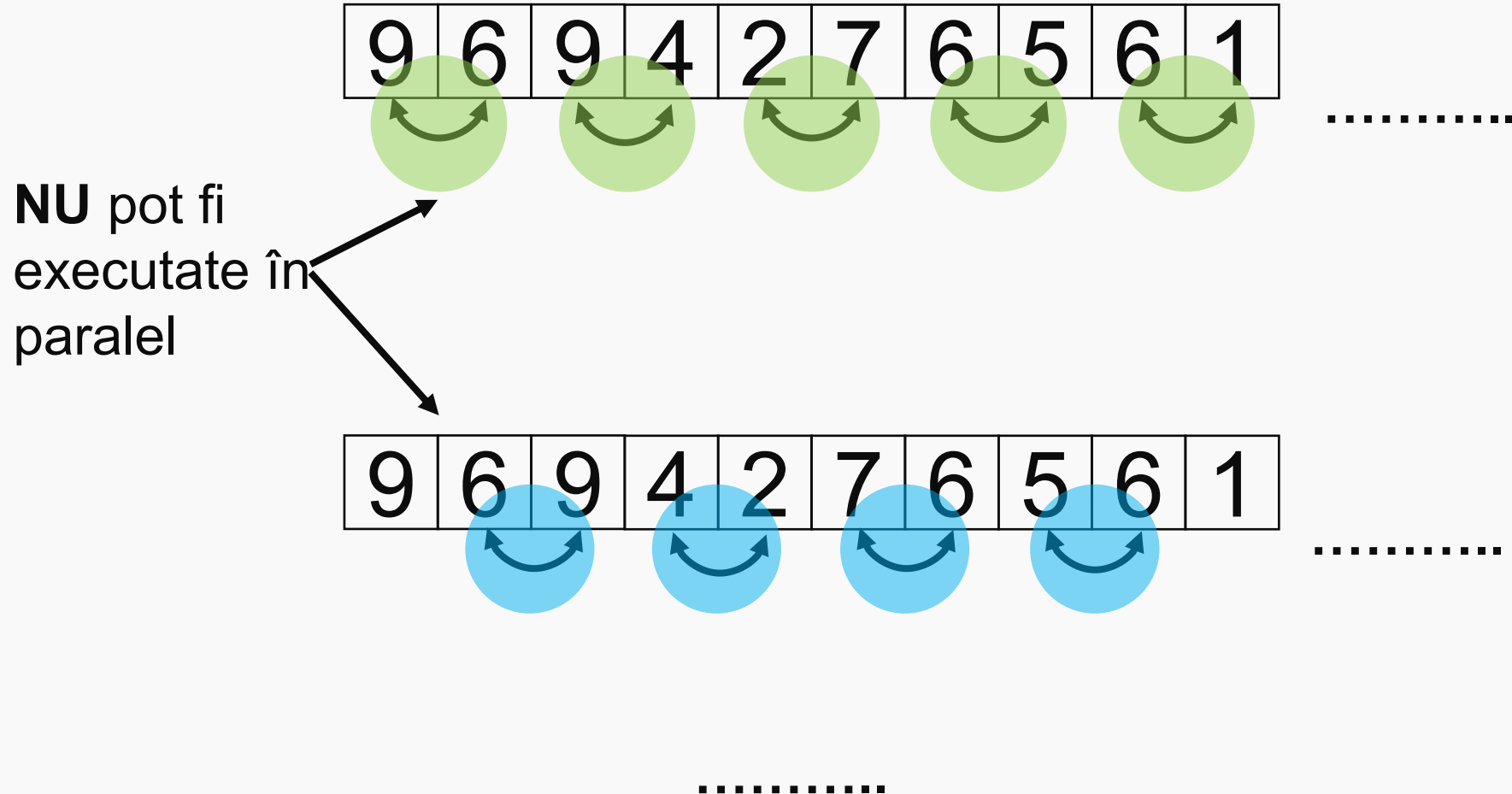
Pot fi  
executate  
în paralel →



.....

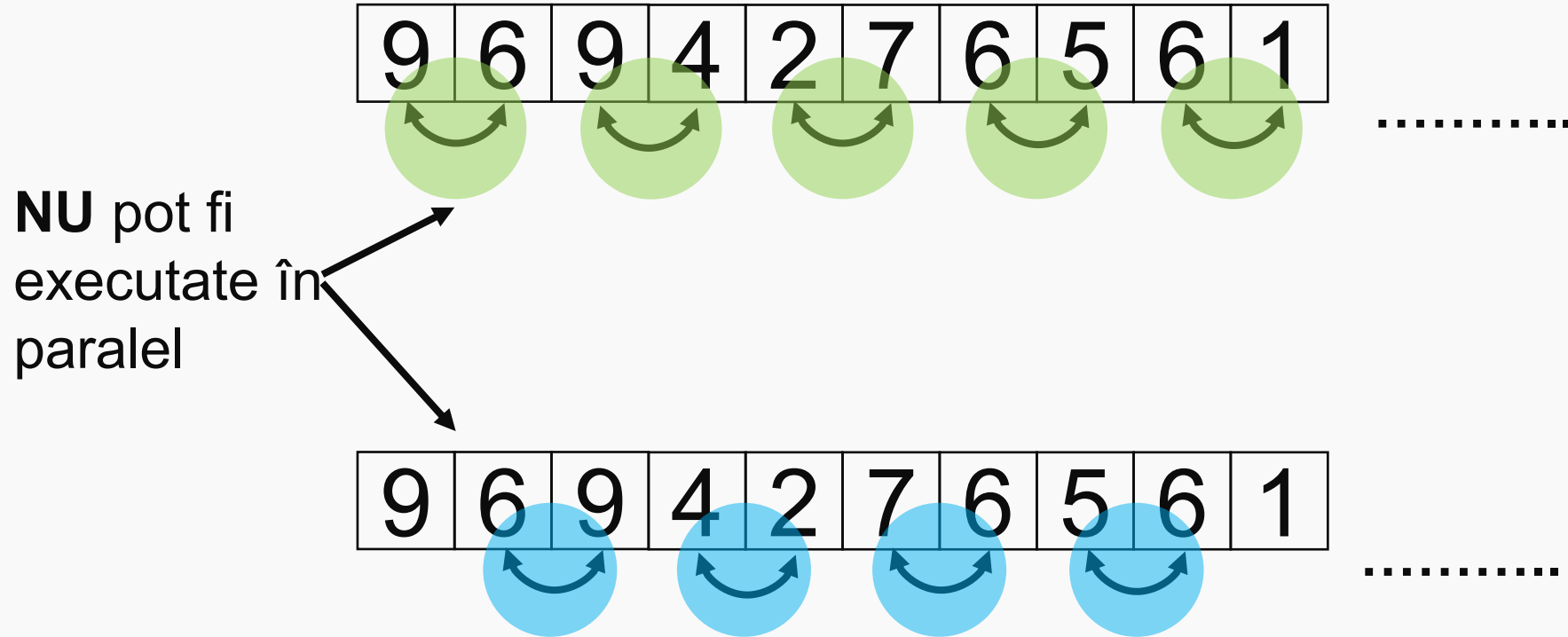


# OETS: Odd-Even Transposition Sort





# OETS: Odd-Even Transposition Sort

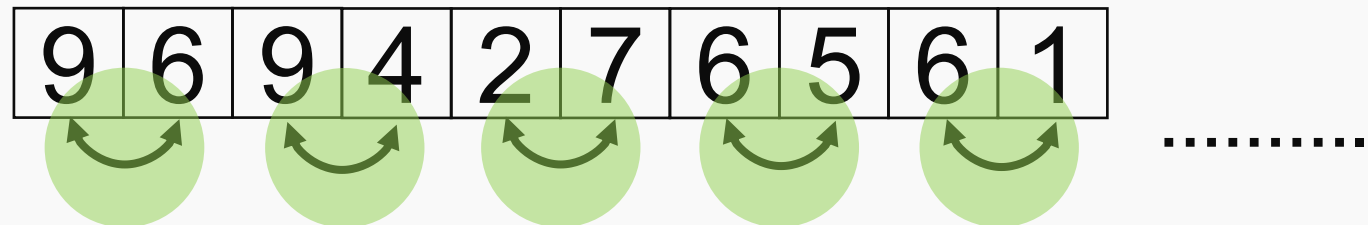


Ce facem? Ce folosim?

.....



# OETS: Odd-Even Transposition Sort

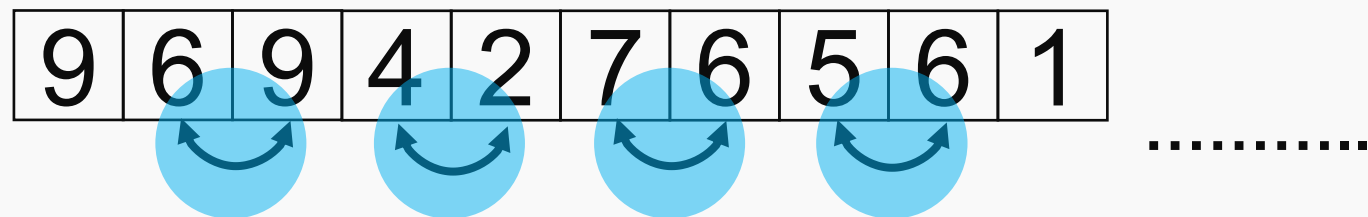
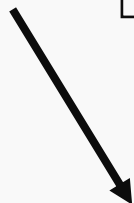


Soluția:

**Barrier**



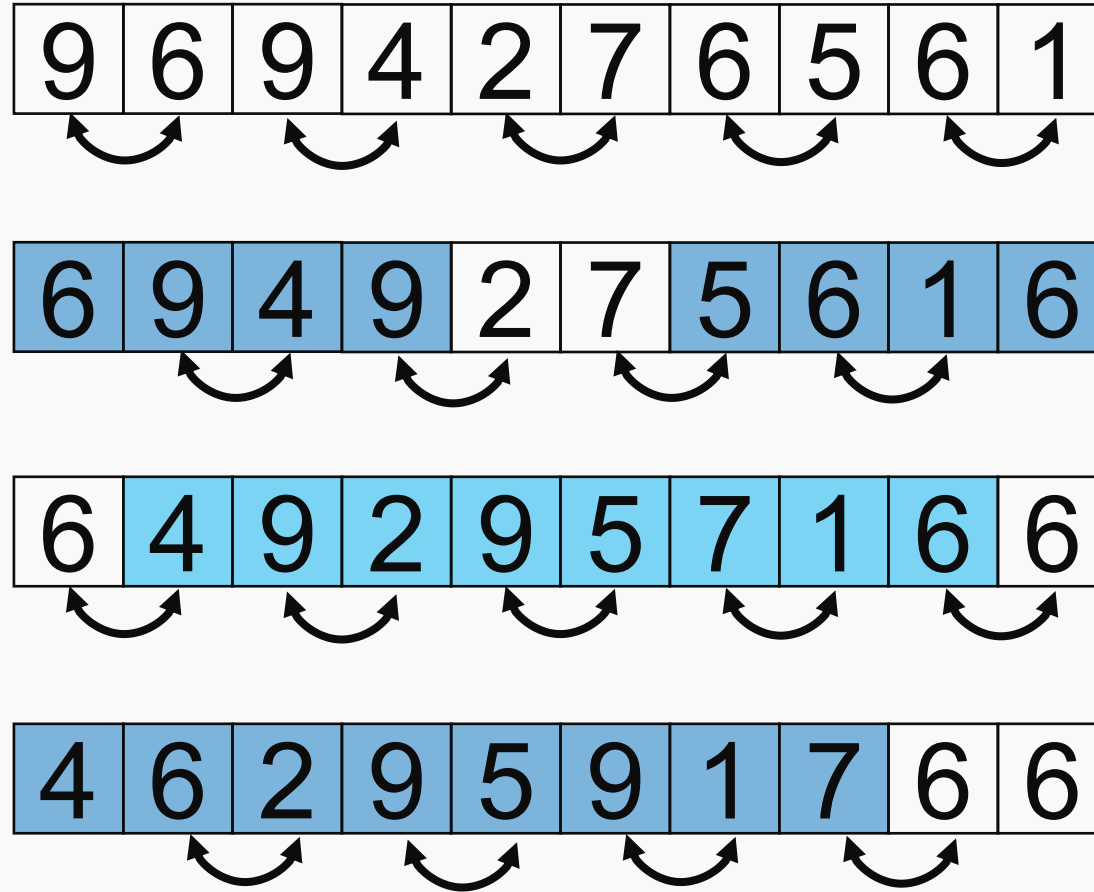
Între  
fiecare  
repetiție



.....

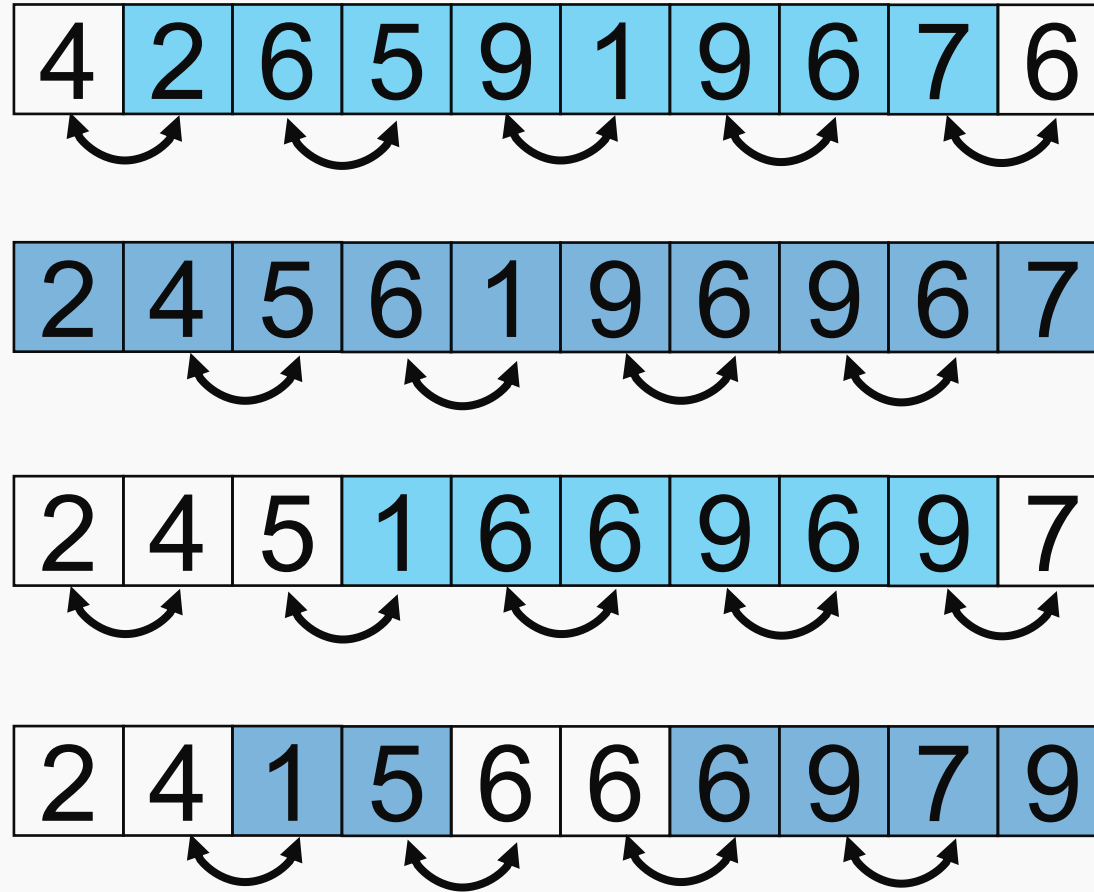


# OETS: Odd-Even Transposition Sort





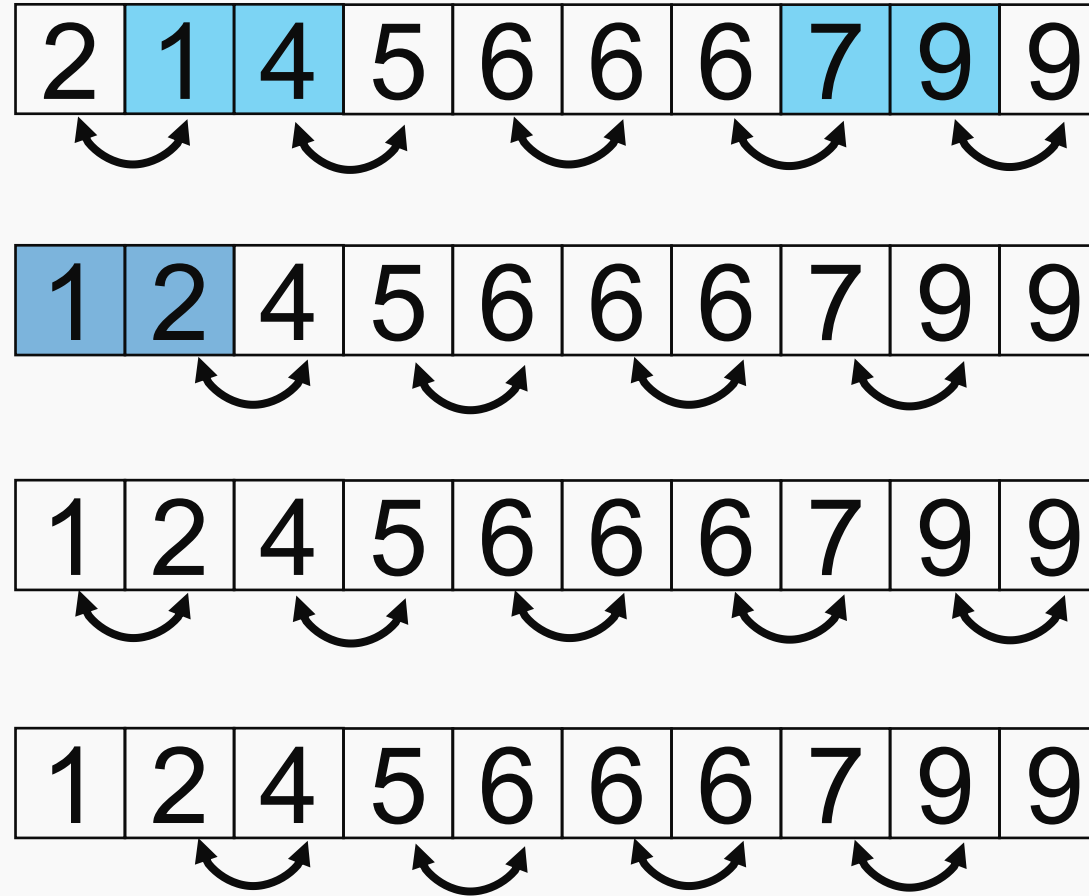
# OETS: Odd-Even Transposition Sort





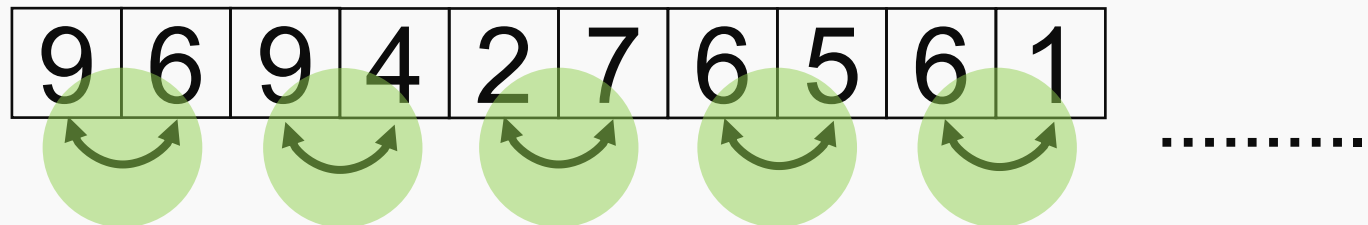


# OETS: Odd-Even Transposition Sort

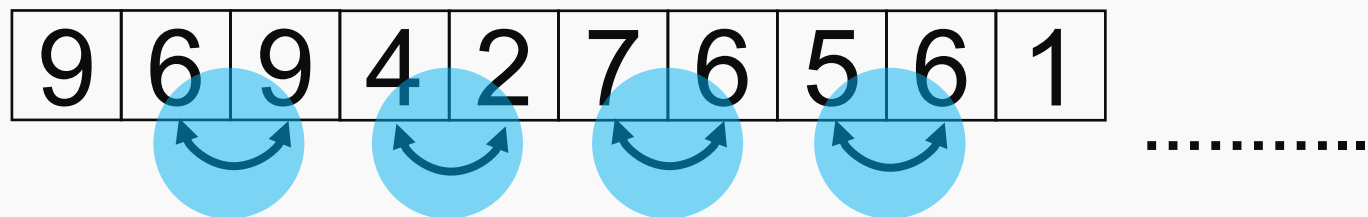




# OETS: Odd-Even Transposition Sort



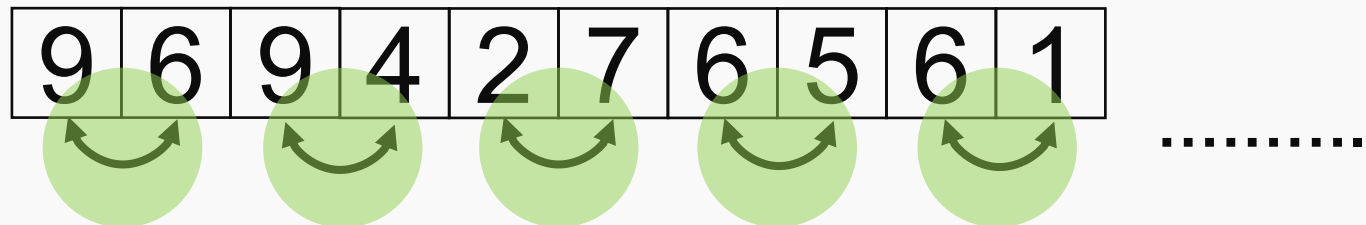
Complexitate a soluției paralele?



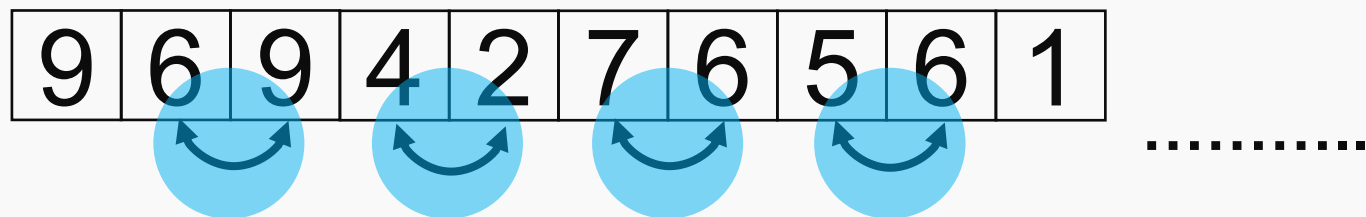
.....



# OETS: Odd-Even Transposition Sort



Complexitate a soluției paralele?

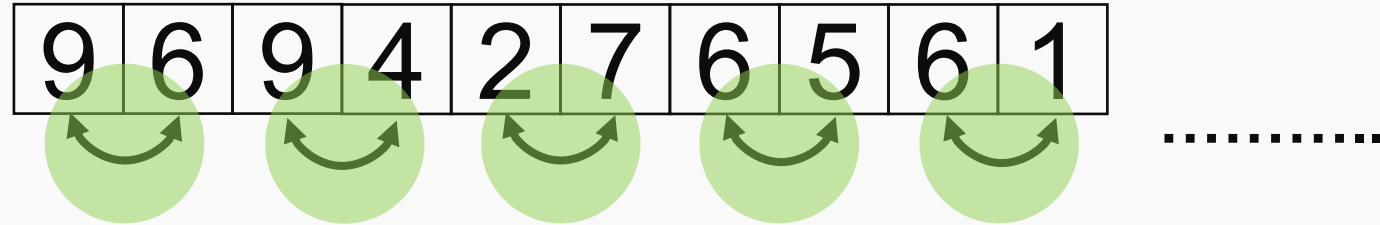


$$T = O\left(\frac{N}{P} * N\right) (= O(N) \text{ pentru } P = N)$$

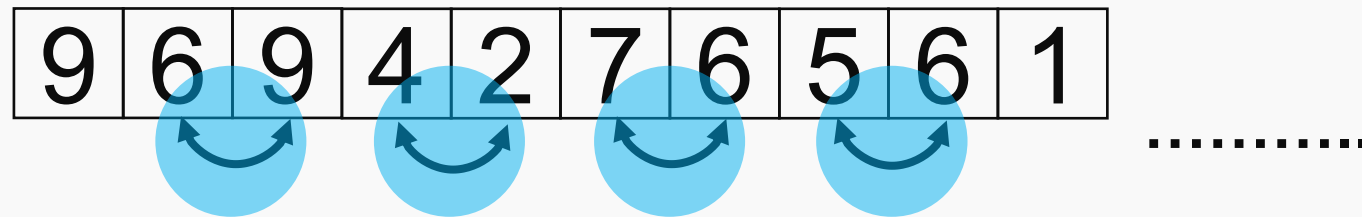
.....



# OETS: Odd-Even Transposition Sort



Speedup?

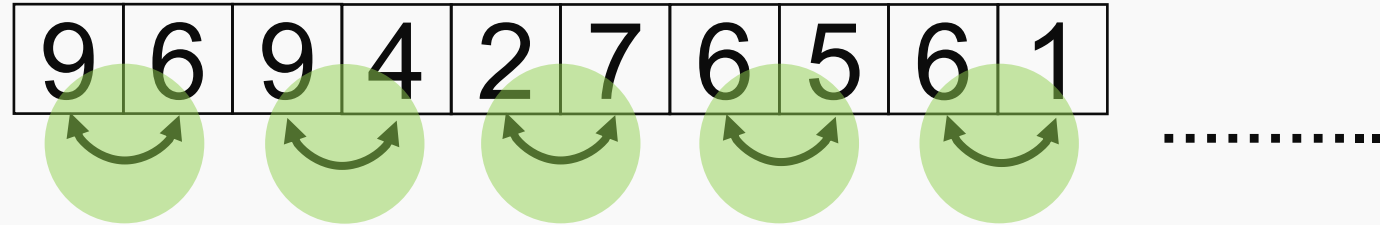


$$T = O\left(\frac{N}{P} * N\right) (= O(N) \text{ pentru } P = N)$$

.....



# OETS: Odd-Even Transposition Sort



Speedup?

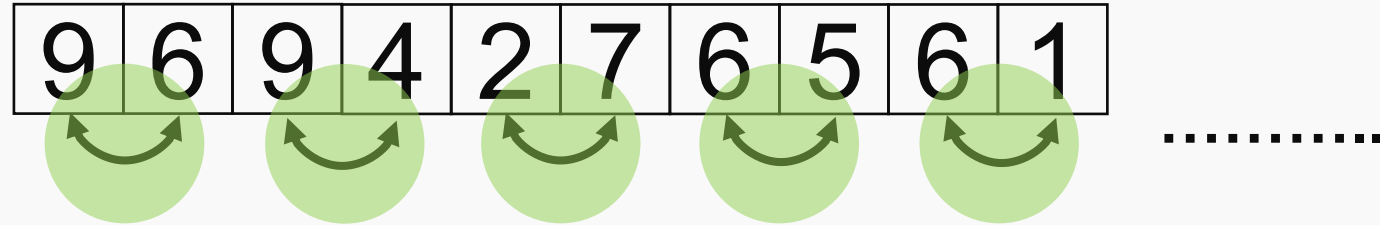


$$S = \frac{N^2}{\frac{N^2}{P}} = P$$

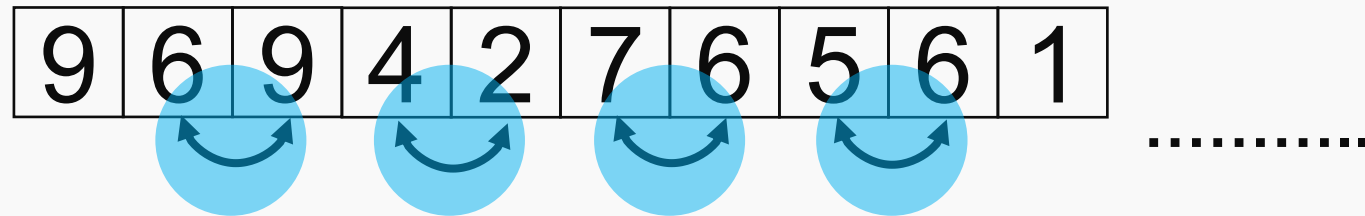
.....



# OETS: Odd-Even Transposition Sort



Speedup? **But is it really?**

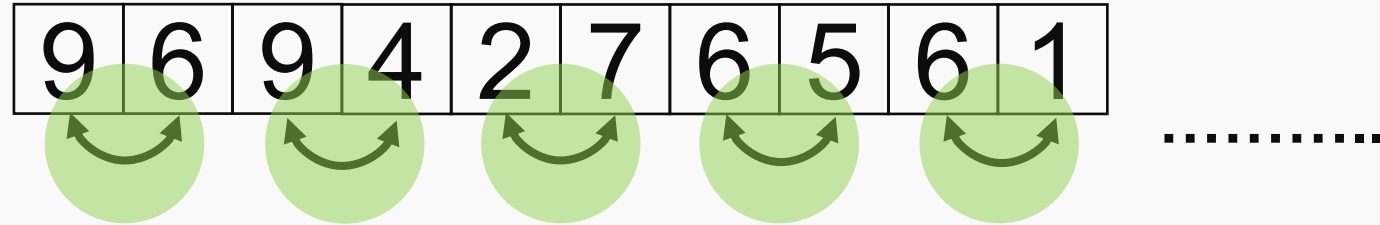


$$S = \frac{N^2}{\frac{N^2}{P}} = P$$

.....



# OETS: Odd-Even Transposition Sort



Speedup?

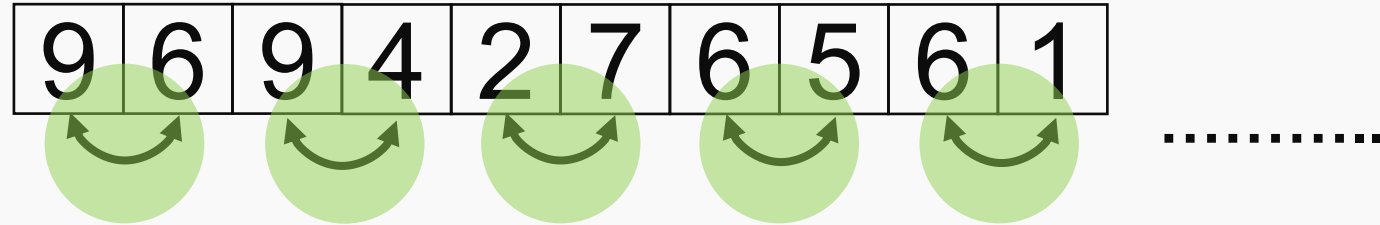


$$S = \frac{N \log_2 N}{\frac{N^2}{P}}$$

.....



# OETS: Odd-Even Transposition Sort



Speedup?



$$S = \frac{P \log_2 N}{N}$$

.....





# OETS: Odd-Even Transposition Sort



Nu uitați așteptatul la barieră poate introduce timpuri mari!



$$S = \frac{P \log_2 N}{N}$$

.....





# Shear sort (Row-column sort) (Snake sort)

9	6	9	4	>
2	7	6	5	<
9	3	6	2	>
5	4	1	5	<

Sortează fiecare linie **pară** în mod **ascendent**  
Sortează fiecare linie **impară** în mod **descendent**



# Shear sort

9	6	9	4
2	7	6	5
9	3	6	2
5	4	1	5

> > > >

Sortează crescător coloanele



# Shear sort

9	6	9	4	>
2	7	6	5	<
9	3	6	2	>
5	4	1	5	<
>	>	>	>	

Repetă tot de  $\log_2 n$  ori



# Shear sort

9	6	9	4	>
2	7	6	5	<
9	3	6	2	>
5	4	1	5	<
>	>	>	>	

De ce liniile pare crescător și celelalte descrescător?



# Shear sort

9	6	9	4	>
2	7	6	5	<
9	3	6	2	>
5	4	1	5	<
>	>	>	>	

De ce liniile pare crescător și celelalte descrescător?  
Dorim compararea celui mai mare element de pe linia  $i$  cu cel mai mic de pe linia  $i+1$



# Shear sort

4	6	9	9	⋈
7	6	5	2	⋈
2	3	6	9	⋈
5	5	4	1	⋈





# Shear sort

2	3	4	1
4	5	5	2
5	6	6	9
7	6	9	9

> > > >



# Shear sort

1	2	3	4	>
5	5	4	2	<
5	6	6	9	>
9	9	7	6	<



# Shear sort

1	2	3	2
5	5	4	4
5	6	6	6
9	9	7	9

> > > >



# Shear sort

1	2	2	3	>
5	5	4	4	<
5	6	6	6	>
9	9	9	7	<



# Shear sort

1	2	2	3
5	5	4	4
5	6	6	6
9	9	9	7

> > > >



# Shear sort

1	2	2	3
5	5	4	4
5	6	6	6
9	9	9	7

Lista finală se obține citind în formă de șerpuită  
(snake sort)



# Shear sort

1	2	2	3
5	5	4	4
5	6	6	6
9	9	9	7

1	2	2	3	4	4	5	5	5	6	6	6	7	9	9	9
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---



# Shear sort

9	6	9	4	>
2	7	6	5	<
9	3	6	2	>
5	4	1	5	<
>	>	>	>	

Complexitate?  $G =$





# Shear sort

9	6	9	4	>
2	7	6	5	<
9	3	6	2	>
5	4	1	5	<
>	>	>	>	

Complexitate?  $G = \log_2 N * \log_2 N$  repetiții



# Shear sort

9	6	9	4	>
2	7	6	5	<
9	3	6	2	>
5	4	1	5	<
>	>	>	>	

Complexitate?  $G = \log_2 N * 2 * \sqrt{N}$   
 $\sqrt{N}$  linii/coloane



# Shear sort

9	6	9	4	>
2	7	6	5	<
9	3	6	2	>
5	4	1	5	<
>	>	>	>	

Complexitate?  $G = \log_2 N * 2 * \sqrt{N} * \sqrt{N} * \log_2 \sqrt{N}$   
 $\sqrt{N} * \log_2 \sqrt{N}$  cel mai bun algoritm secvențial de sortare



# Shear sort

9	6	9	4	>
2	7	6	5	<
9	3	6	2	>
5	4	1	5	<
>	>	>	>	

Complexitate?  $G = \log_2 N * N * \log_2 \sqrt{N}$



# Shear sort

9	6	9	4	>
2	7	6	5	<
9	3	6	2	>
5	4	1	5	<
>	>	>	>	

Complexitate?  $G = N \log_2 N * \log_2 \sqrt{N}$

Cel mai bun algoritm secvențial rămâne  $N \log_2 N$



# Shear sort

9	6	9	4	>
2	7	6	5	<
9	3	6	2	>
5	4	1	5	<
>	>	>	>	

Cum paralelizăm?



# Shear sort

9	6	9	4	>
2	7	6	5	<
9	3	6	2	>
5	4	1	5	<
>	>	>	>	

Cum paralelizăm?

Toate liniile în paralel, barieră, apoi toate coloanele, apoi barieră, și repetăm.



# Shear sort

9	6	9	4	>
2	7	6	5	<
9	3	6	2	>
5	4	1	5	<
>	>	>	>	

Cum paralelizăm?

Toate liniile în paralel, barieră, apoi toate coloanele, apoi barieră, și repetăm.





# Shear sort

9	6	9	4	>
2	7	6	5	<
9	3	6	2	>
5	4	1	5	<
>	>	>	>	

Complexitate versiune paralelă?

$$G = \log_2 N * 2 * \sqrt{N} * \sqrt{N} * \log_2 \sqrt{N}$$



# Shear sort

9	6	9	4	>
2	7	6	5	<
9	3	6	2	>
5	4	1	5	<
>	>	>	>	

Complexitate versiune paralelă?  $N = P$

$$T = \log_2 N * 2 * \sqrt{N} * \log_2 \sqrt{N}$$



# Shear sort

9	6	9	4	>
2	7	6	5	<
9	3	6	2	>
5	4	1	5	<
>	>	>	>	

Complexitate versiune paralelă?  $N = P$

$$T = \sqrt{N} \log_2 \sqrt{N} * \log_2 N$$



# Shear sort

9	6	9	4	>
2	7	6	5	<
9	3	6	2	>
5	4	1	5	<
>	>	>	>	

Complexitate versiune paralelă?

$$T = \log_2 N * 2 * \frac{\sqrt{N}}{P} * \sqrt{N} * \log_2 \sqrt{N}$$



# Shear sort

9	6	9	4	>
2	7	6	5	<
9	3	6	2	>
5	4	1	5	<
>	>	>	>	

Complexitate versiune paralelă?

$$T = \frac{N}{P} \log_2 N * \log_2 \sqrt{N}$$



# Shear sort

9	6	9	4	>
2	7	6	5	<
9	3	6	2	>
5	4	1	5	<
>	>	>	>	

Speedup?

$$T = \frac{N}{P} \log_2 N * \log_2 \sqrt{N}$$

$$G = N \log_2 N$$



# Shear sort

$$S = \frac{N \log_2 N}{\frac{N}{P} \log_2 N * \log_2 \sqrt{N}}$$

9	6	9	4	>
2	7	6	5	<
9	3	6	2	>
5	4	1	5	<
>	>	>	>	

Speedup?

$$T = \frac{N}{P} \log_2 N * \log_2 \sqrt{N}$$

$$G = N \log_2 N$$



# Shear sort

$$S = \frac{P}{\log_2 \sqrt{N}}$$

9	6	9	4	>
2	7	6	5	<
9	3	6	2	>
5	4	1	5	<
>	>	>	>	

Speedup?

$$T = \frac{N}{P} \log_2 N * \log_2 \sqrt{N}$$

$$G = N \log_2 N$$







# Matrix multiply

```
for(i=0; i<N; i++)  
    for(j=0; j<N; j++)  
        for(k=0; k<N; k++)  
            c[i][j] += a[i][k] * b[k][j]
```



# Matrix multiply

```
for(i=0; i<N; i++)  
    for(j=0; j<N; j++)  
        for(k=0; k<N; k++)  
            c[i][j] += a[i][k] * b[k][j]
```

Complexitate?



# Matrix multiply

```
for(i=0; i<N; i++)  
    for(j=0; j<N; j++)  
        for(k=0; k<N; k++)  
            c[i][j] += a[i][k] * b[k][j]
```

Complexitate?

$$G = N^3$$



# Matrix multiply

Atenție avem  $N^2$  elemente

```
for(i=0; i<N; i++)  
    for(j=0; j<N; j++)  
        for(k=0; k<N; k++)  
            c[i][j] += a[i][k] * b[k][j]
```

Complexitate?

$$G = N^3$$



# Matrix multiply

```
co[Tid = 1 to P]
{
    start = Tid * ceil(N/P)
    end = min((Tid+1) * ceil(N/P),N)
    for(i=start; i<end; i++)
        for(j=0; j<N; j++)
            for(k=0; k<N; k++)
                c[i][j] += a[i][k] * b[k][j]
}
```



# Matrix multiply

Complexitate?

```
co[Tid = 1 to P]
{
    start = Tid * ceil(N/P)
    end = min((Tid+1) * ceil(N/P),N)
    for(i=start; i<end; i++)
        for(j=0; j<N; j++)
            for(k=0; k<N; k++)
                c[i][j] += a[i][k] * b[k][j]
}
```



# Matrix multiply

Complexitate?

```
co[Tid = 1 to P]
{
    start = Tid * ceil(N/P)
    end = min((Tid+1) * ceil(N/P),N)
    for(i=start; i<end; i++)
        for(j=0; j<N; j++)
            for(k=0; k<N; k++)
                c[i][j] += a[i][k] * b[k][j]
}
```

$$T = \frac{N^3}{P}$$





# Matrix multiply

Speedup?

```
co[Tid = 1 to P]
{
    start = Tid * ceil(N/P)
    end = min((Tid+1) * ceil(N/P),N)
    for(i=start; i<end; i++)
        for(j=0; j<N; j++)
            for(k=0; k<N; k++)
                c[i][j] += a[i][k] * b[k][j]
}
```

$$T = \frac{N^3}{P}$$



# Matrix multiply

Speedup?

```
co[Tid = 1 to P]
{
    start = Tid * ceil(N/P)
    end = min((Tid+1) * ceil(N/P),N)
    for(i=start; i<end; i++)
        for(j=0; j<N; j++)
            for(k=0; k<N; k++)
                c[i][j] += a[i][k] * b[k][j]
}
```

$$S = \frac{N^3}{\frac{N^3}{P}}$$



# Matrix multiply

Speedup?

```
co[Tid = 1 to P]
{
    start = Tid * ceil(N/P)
    end = min((Tid+1) * ceil(N/P),N)
    for(i=start; i<end; i++)
        for(j=0; j<N; j++)
            for(k=0; k<N; k++)
                c[i][j] += a[i][k] * b[k][j]
}
```

$$S = P$$



# Super-linear speedup?

$S > P?$