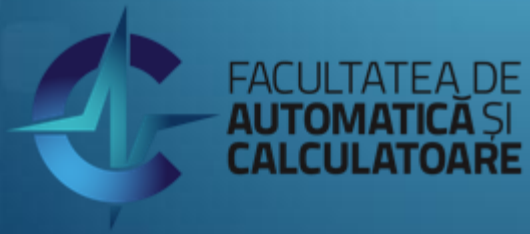
Arhitecturi Paralele
Abordări probleme paralele

Prof. Florin Pop
As. Drd. Ing. Cristian Chilipirea
cristian.chilipirea@cs.pub.ro

Elemente preluate din cursul Prof. Ciprian Dobre




POLITEHNICA
UNIVERSITATEA
BUCUREŞTI
UPB
1818

Merge algorithm

2	3	4	5	7

1	2	4	4	6

Take two **sorted** lists
merge them
Into one **sorted** list

1	2	2	3	4	4	4	5	6	7

Merge algorithm

2	3	4	5	7

1	2	4	4	6

Solution:

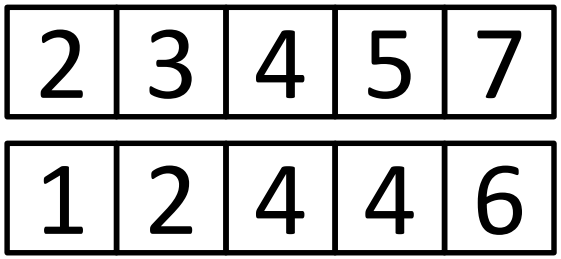
Always extract the smallest element from the lists

(guaranteed to be first in one of them)

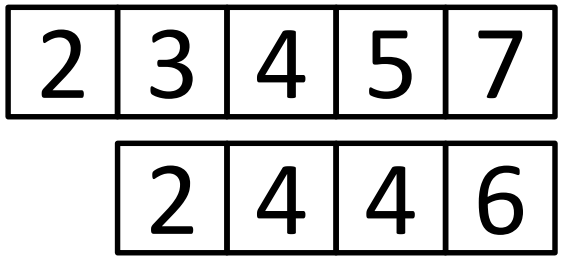
O(n) complexity

1	2	2	3	4	4	4	5	6	7

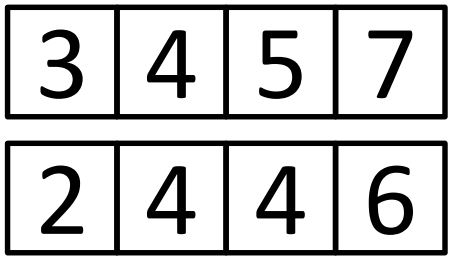
Merge algorithm

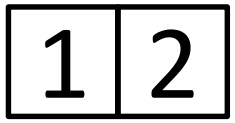


Merge algorithm

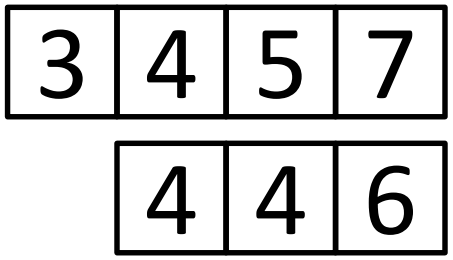


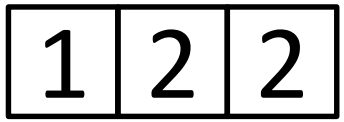
Merge algorithm



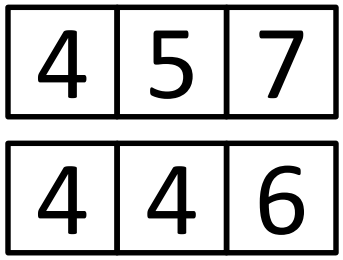


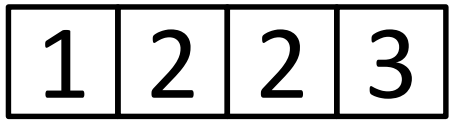
Merge algorithm



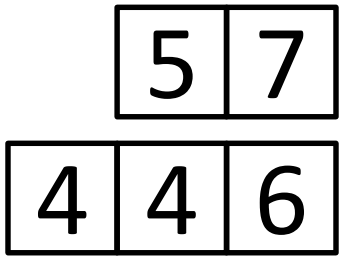


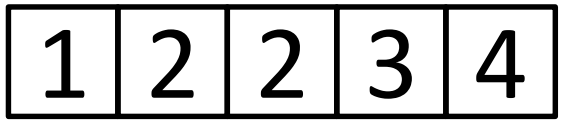
Merge algorithm



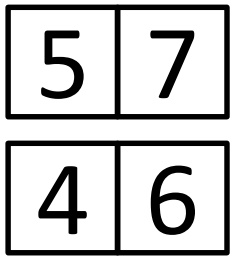


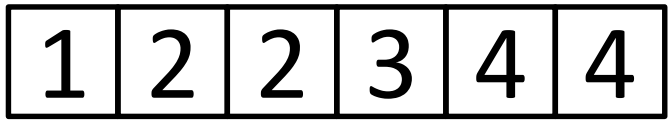
Merge algorithm



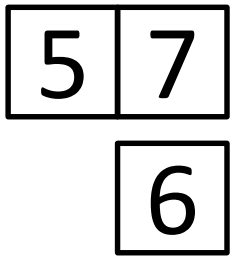


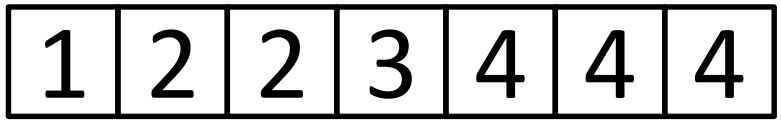
Merge algorithm



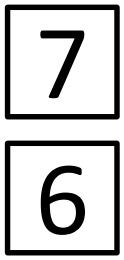


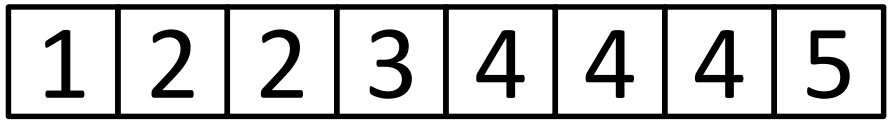
Merge algorithm



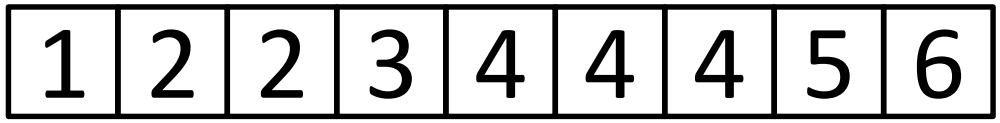


Merge algorithm

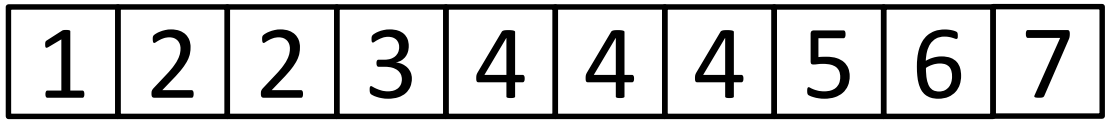




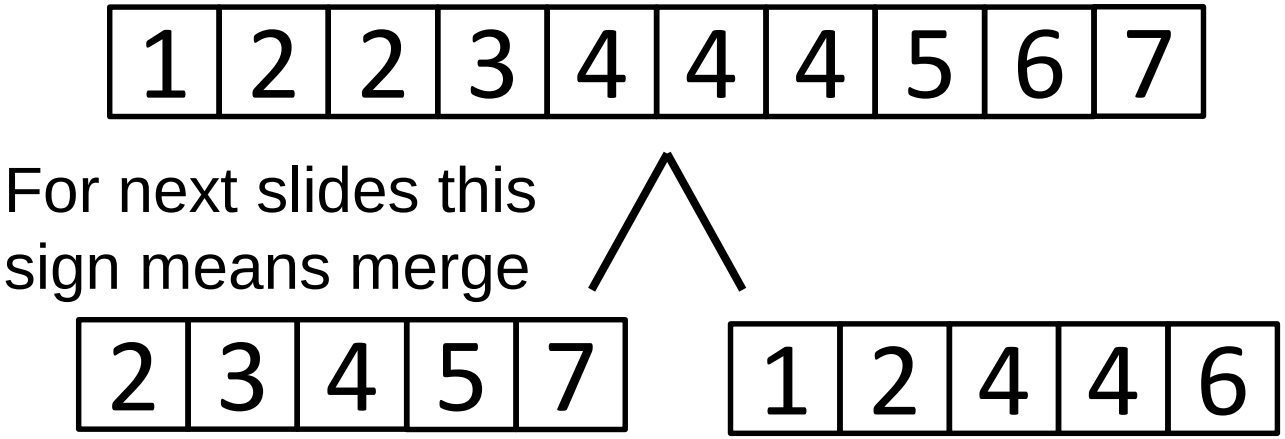
Merge algorithm



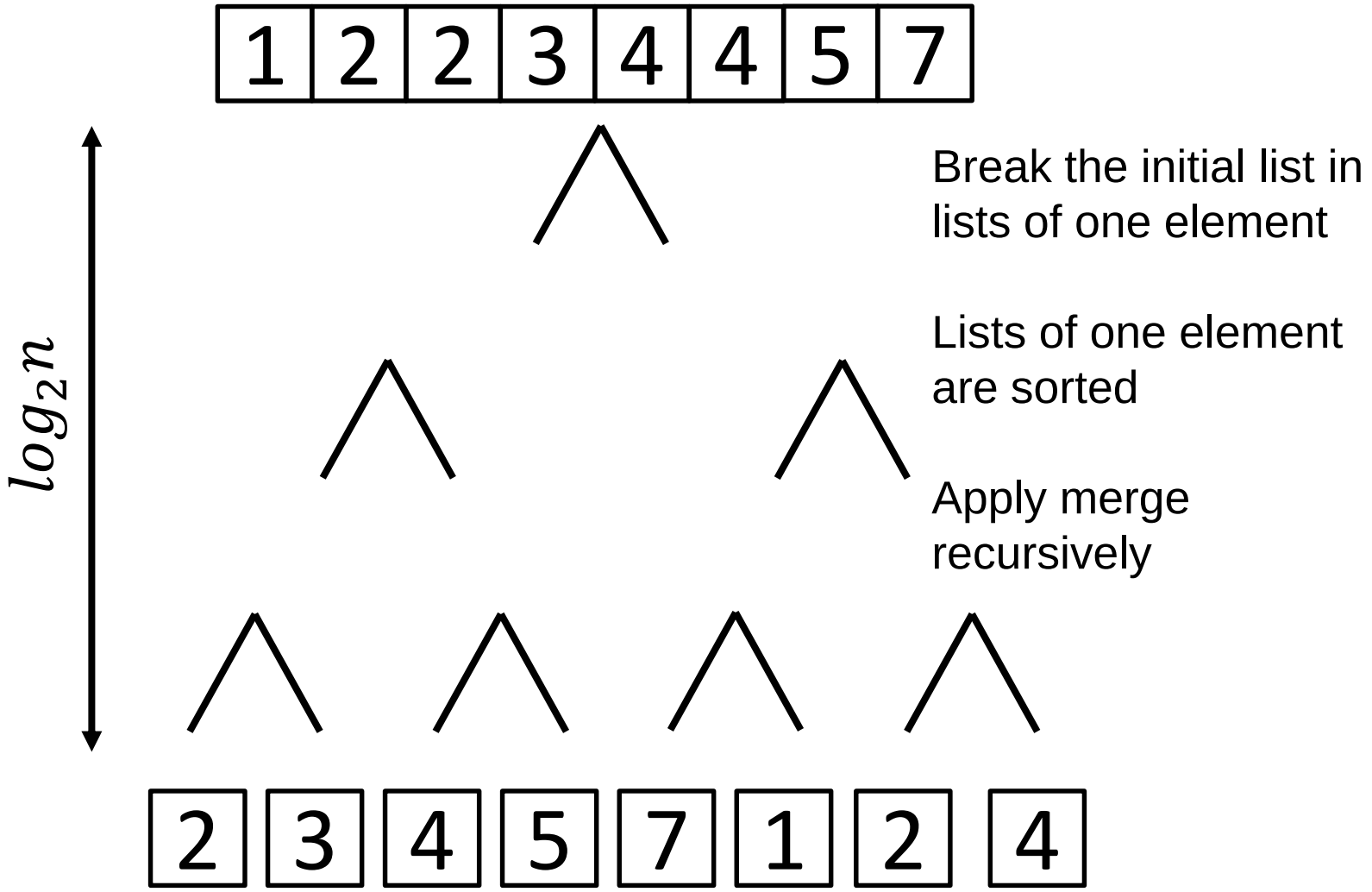
Merge algorithm



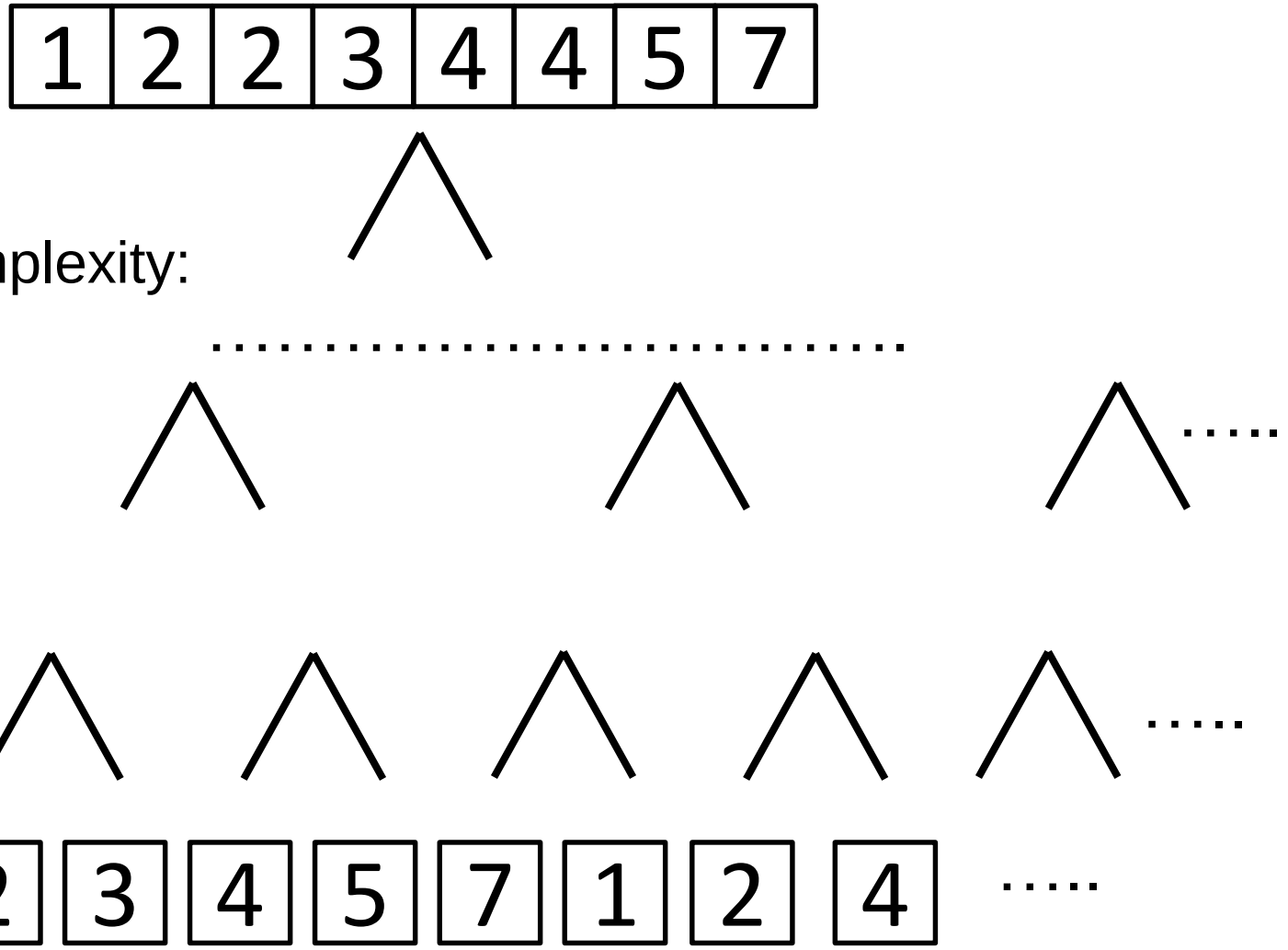
Merge algorithm



Merge sort


1 2 2 3 4 4 5 7
$log_2 n$
Break the initial list in lists of one element
Lists of one element are sorted
Apply merge recursively
2 3 4 5 7 1 2 4

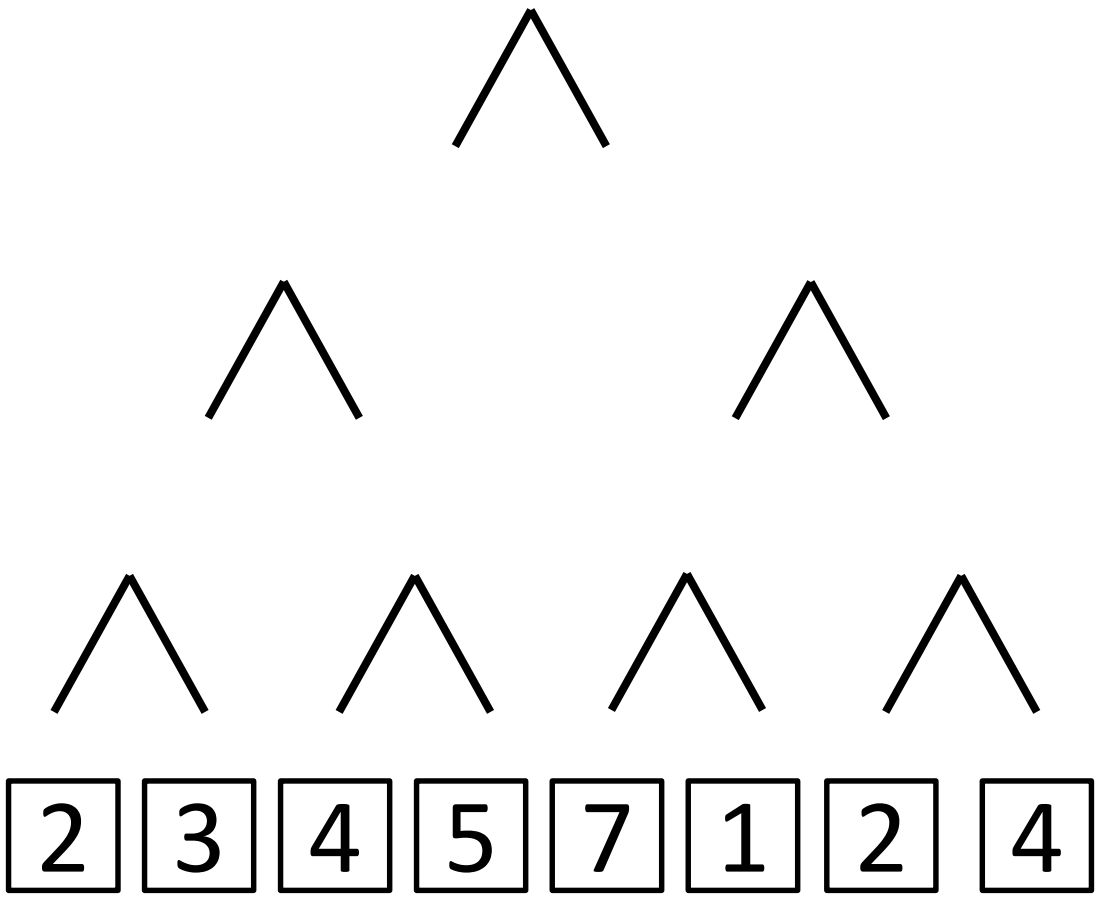
Merge sort



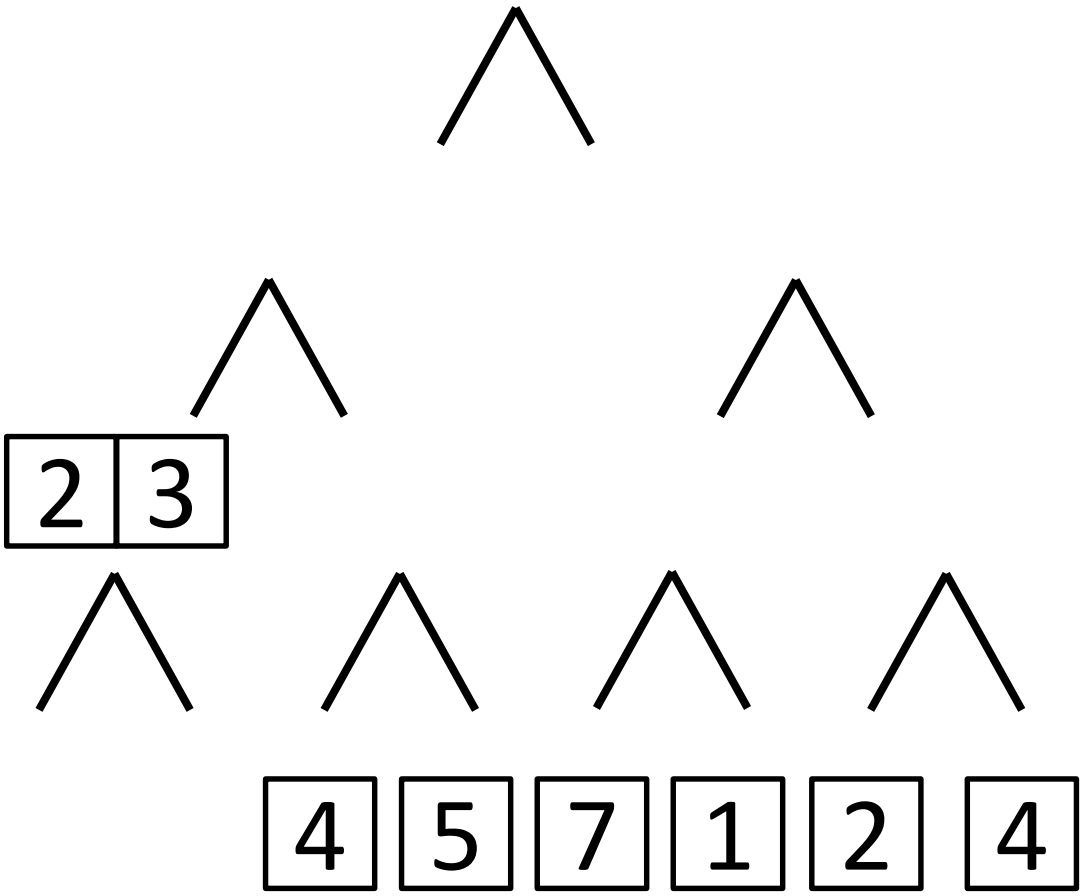
Sequential complexity:

$O(n * log_2 n)$

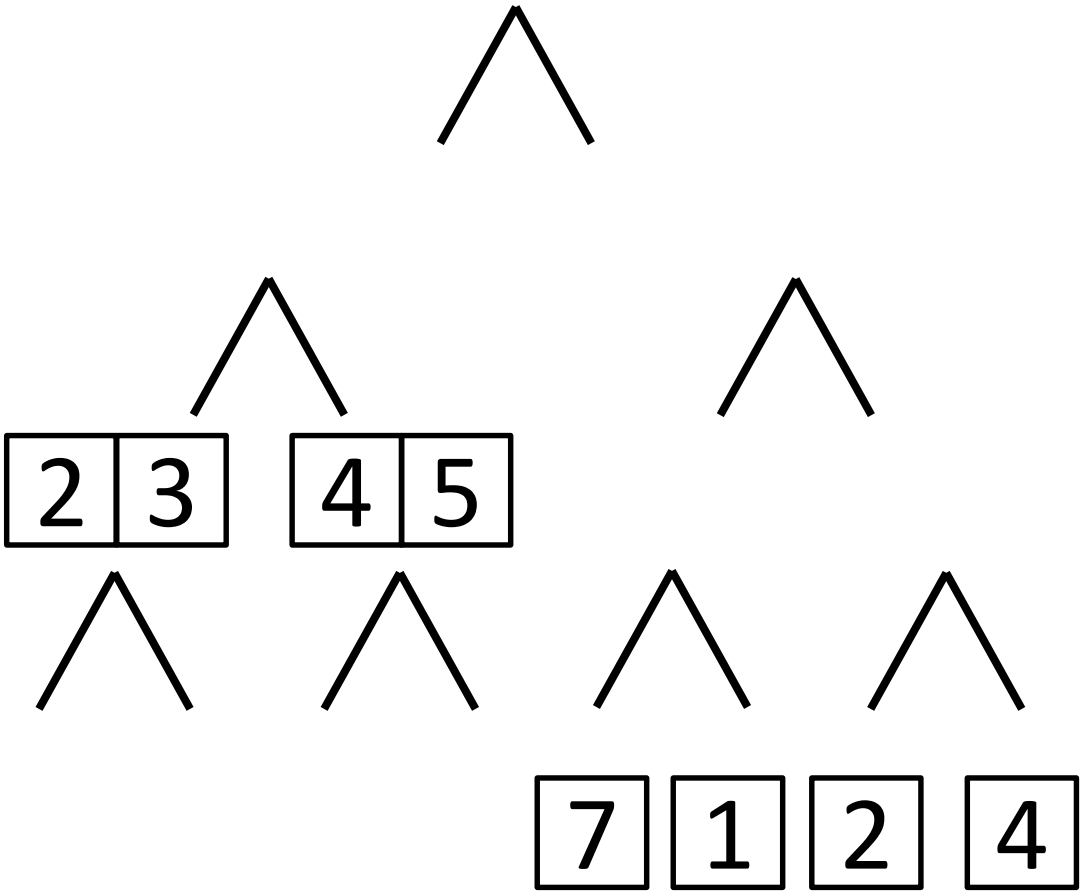
Merge sort



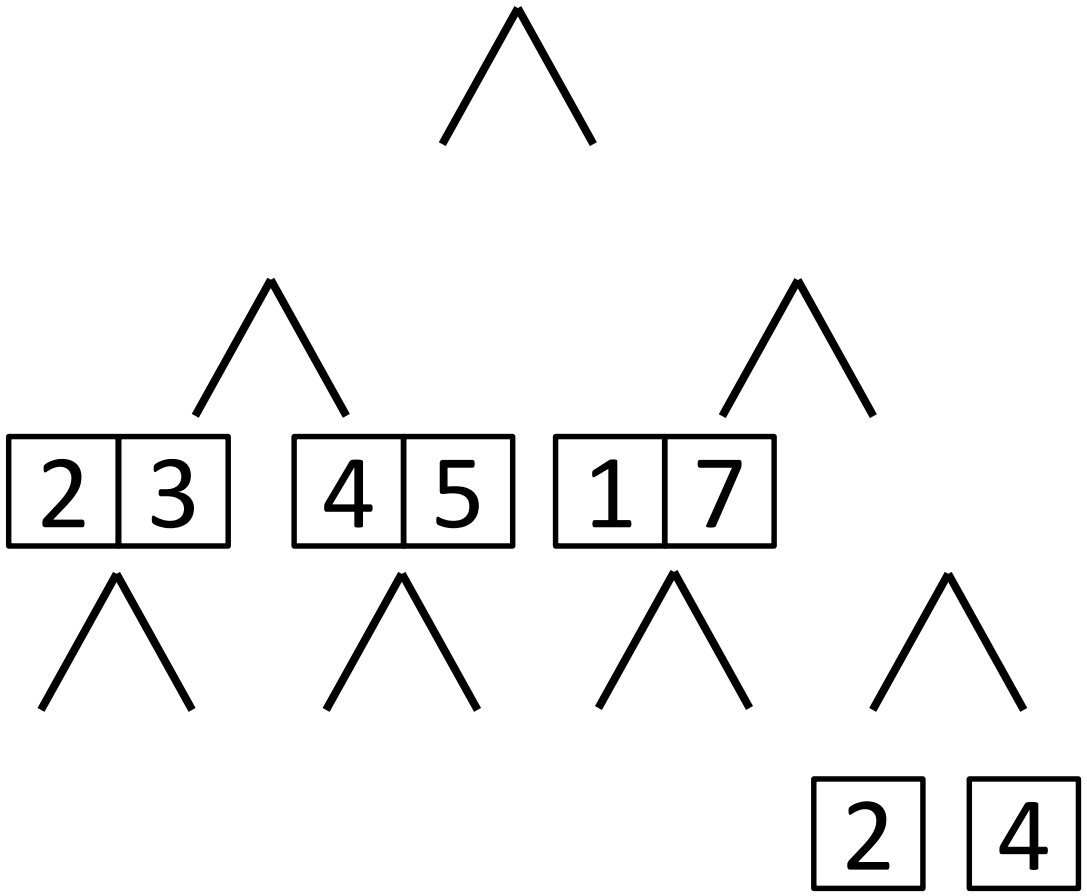
Merge sort



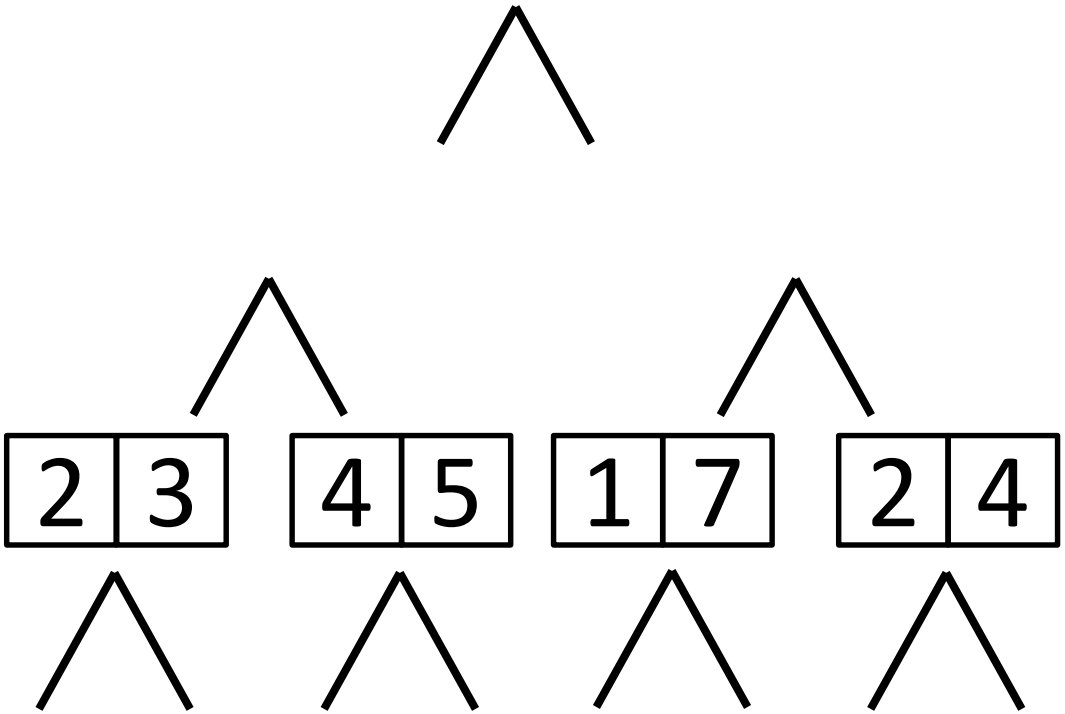
Merge sort



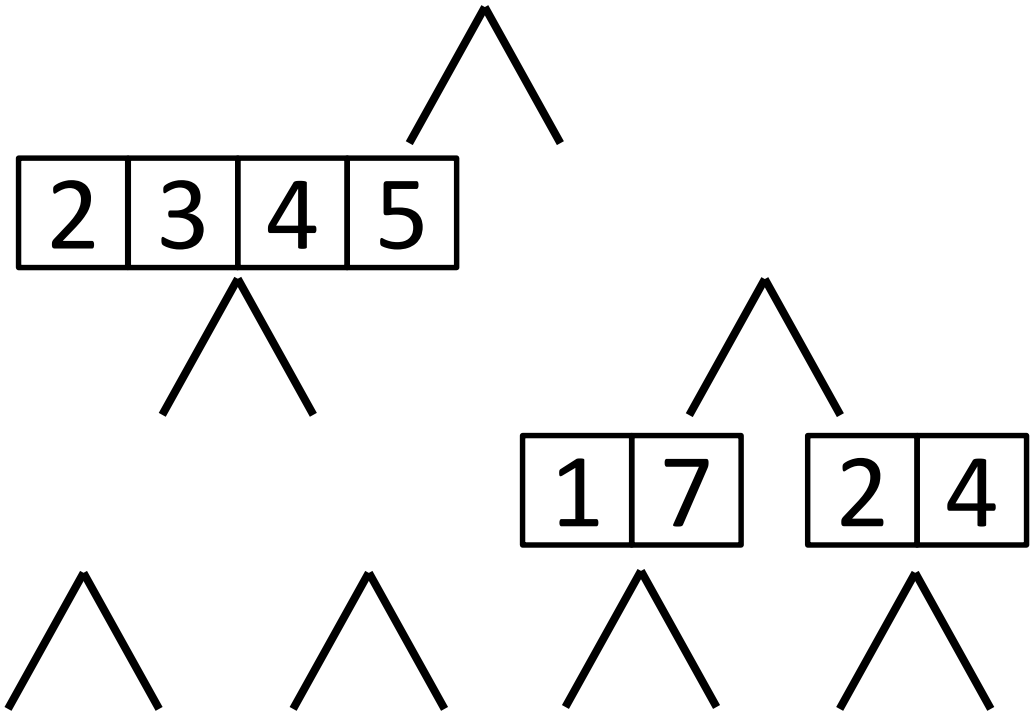
Merge sort



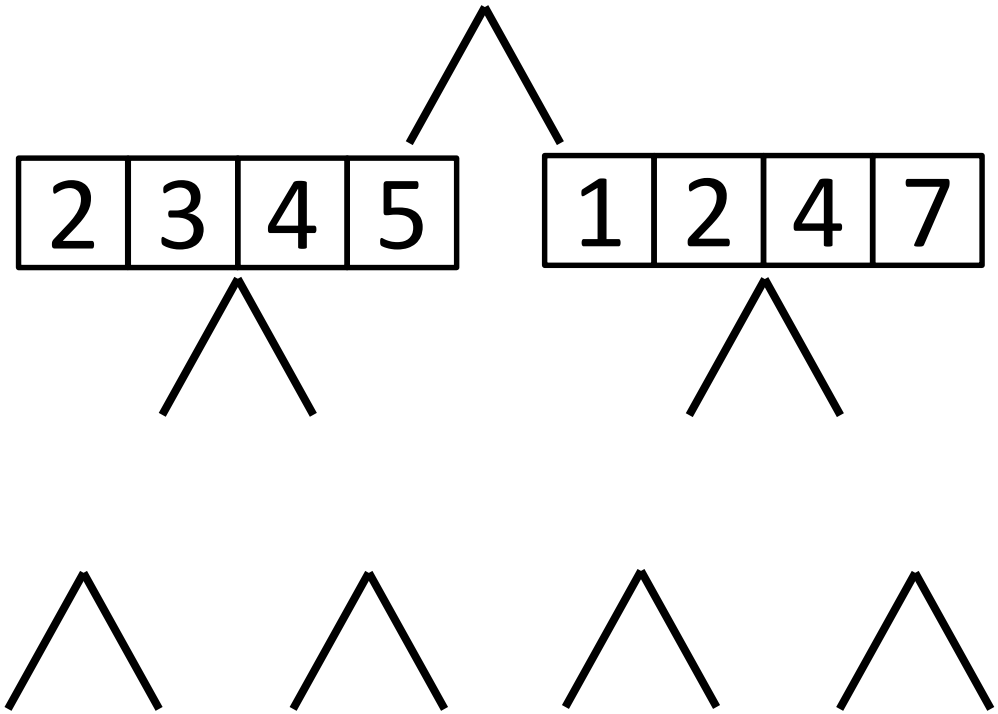
Merge sort



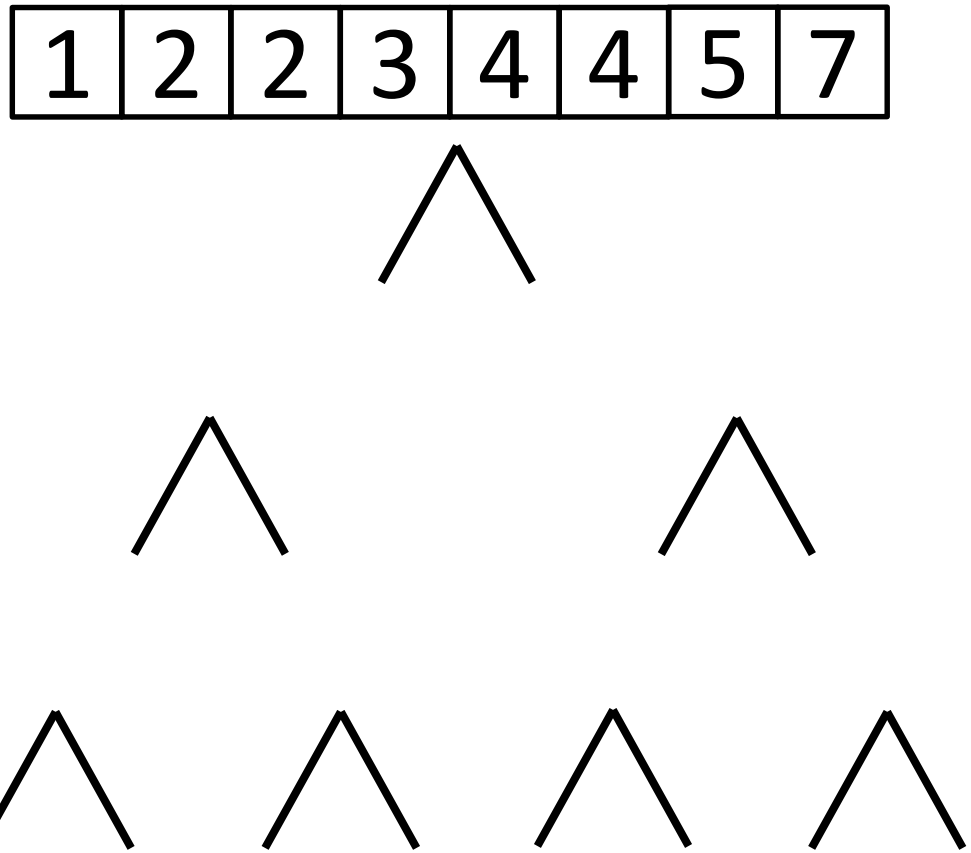
Merge sort



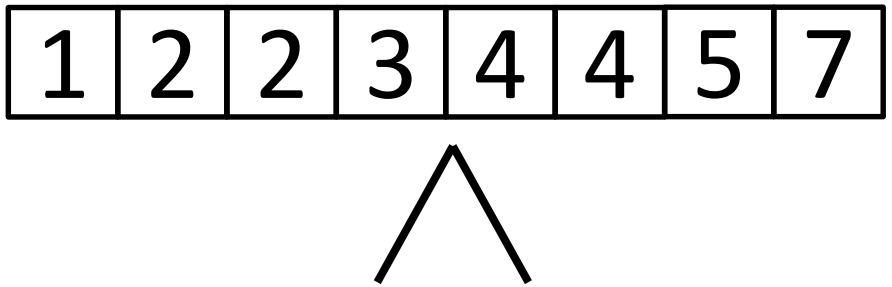
Merge sort



Merge sort

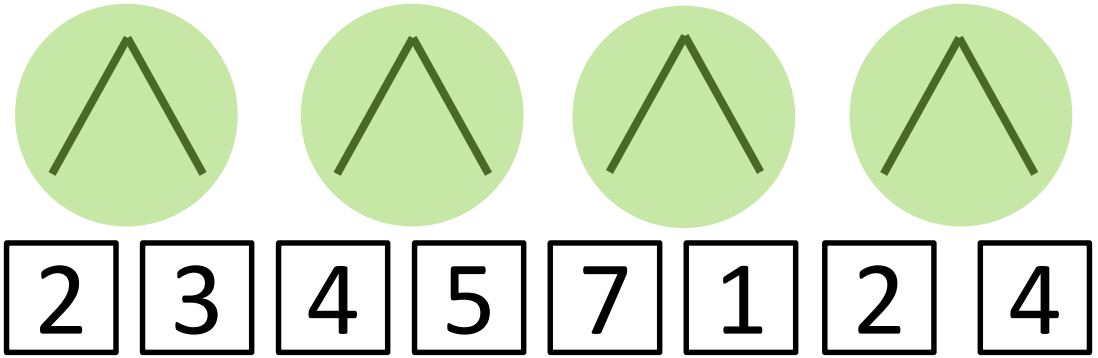


Parallel merge sort

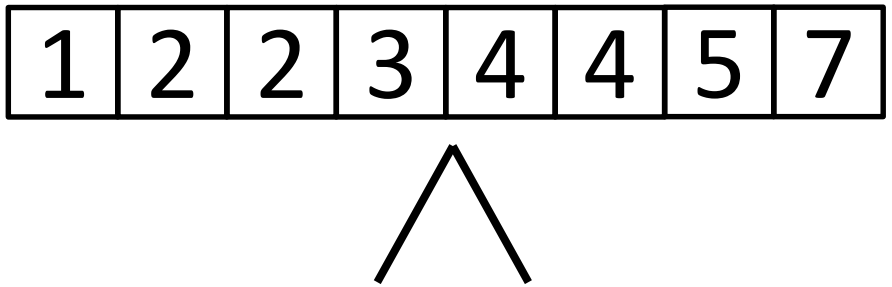


Can be executed in parallel →

Can be executed in parallel →



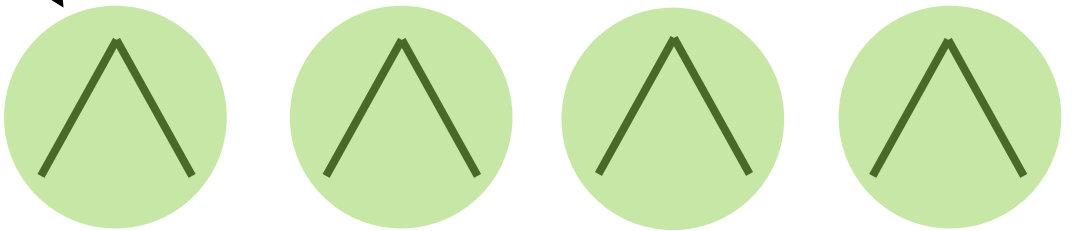
Parallel merge sort

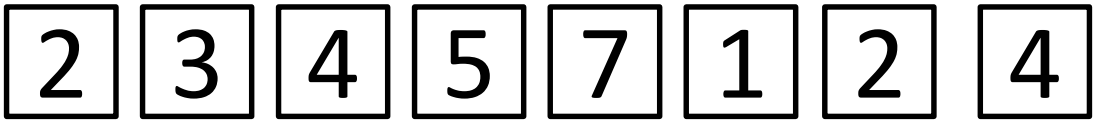


Result of

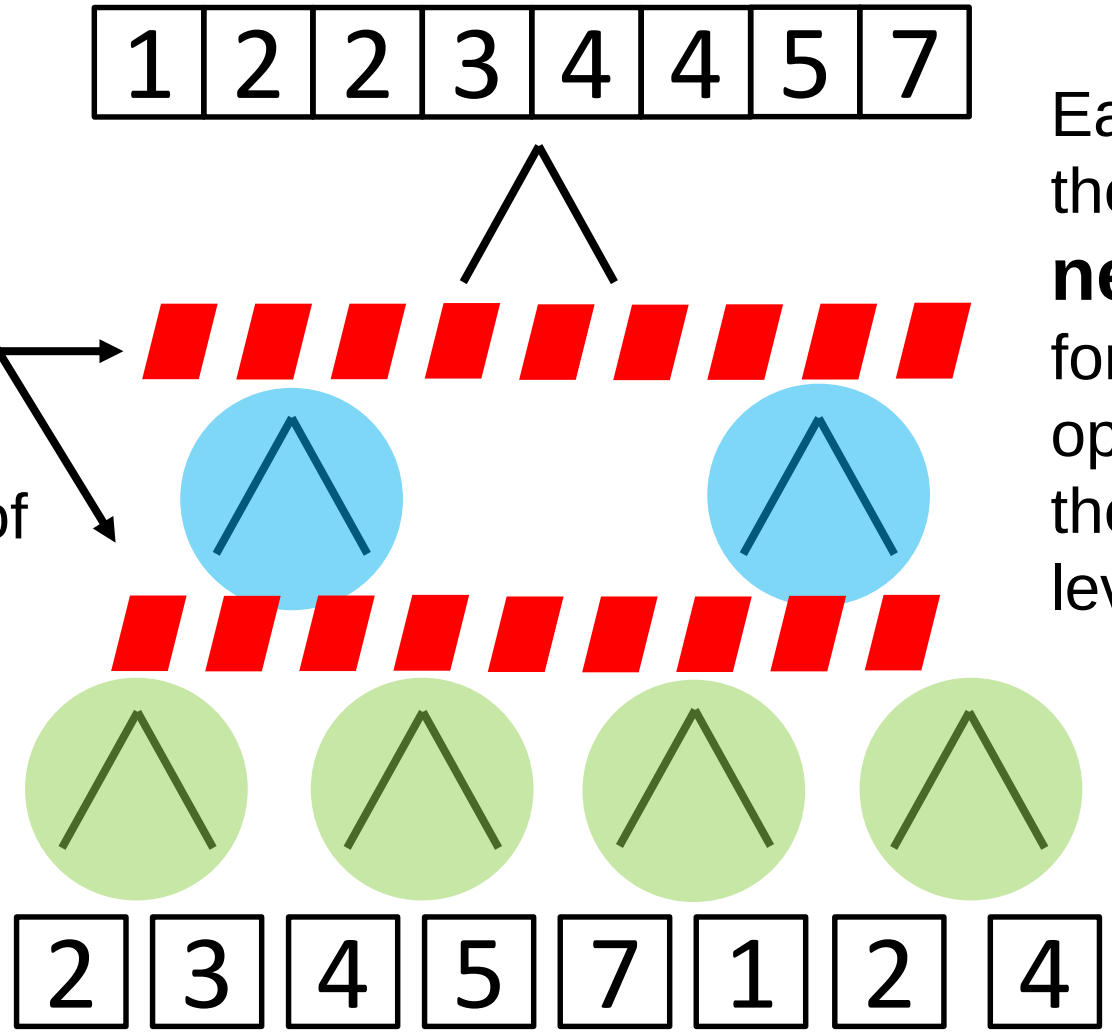
depends on result of

Can **NOT** be executed in parallel





Parallel merge sort



Solution:
Barrier
between
each level of
the tree

Each level of
the tree
needs to wait
for the
operations on
the previous
level to finish

Parallel merge sort

Parallel complexity:

p = processor cores

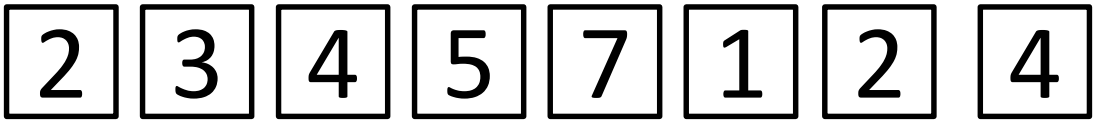
for p=n

$O(\sum_{i=1}^{log_2 n} 2^i) = O(n)$

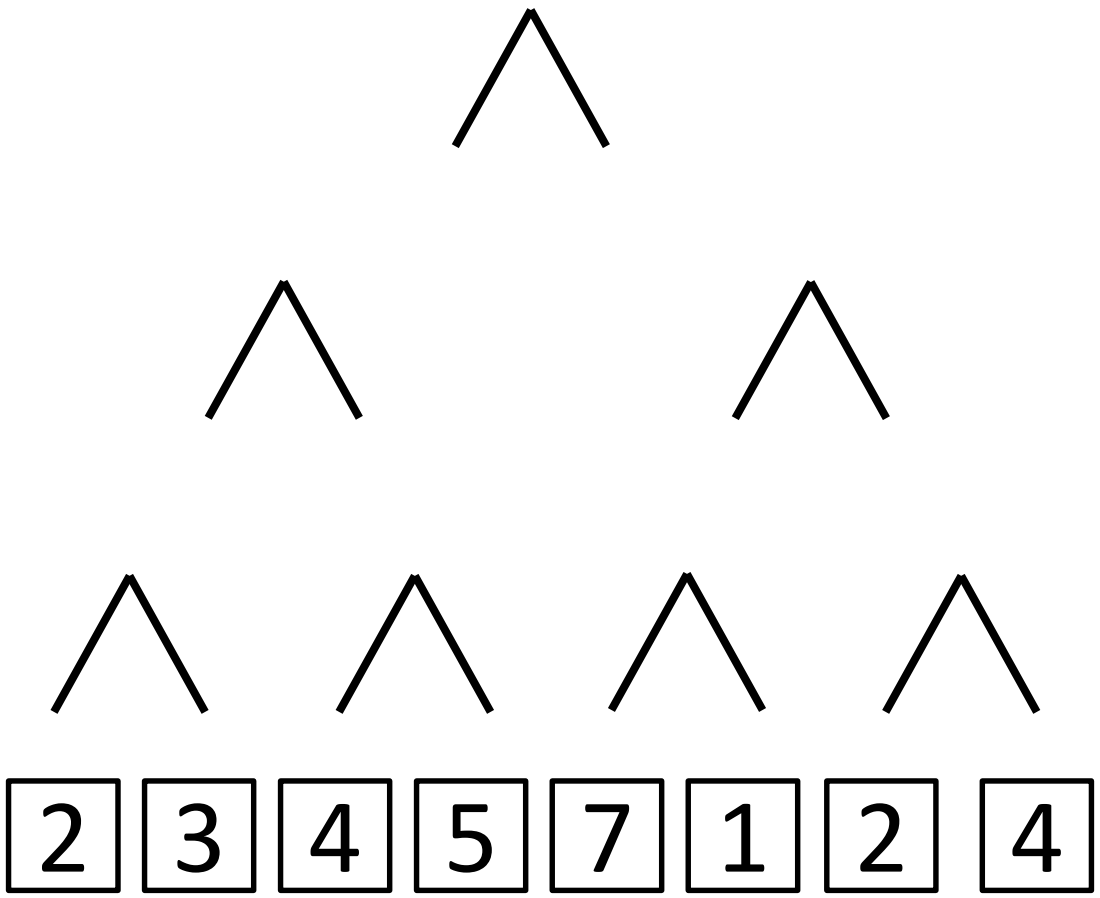
Best implementation: $O(log_2 n)$

(with O(1) merge)

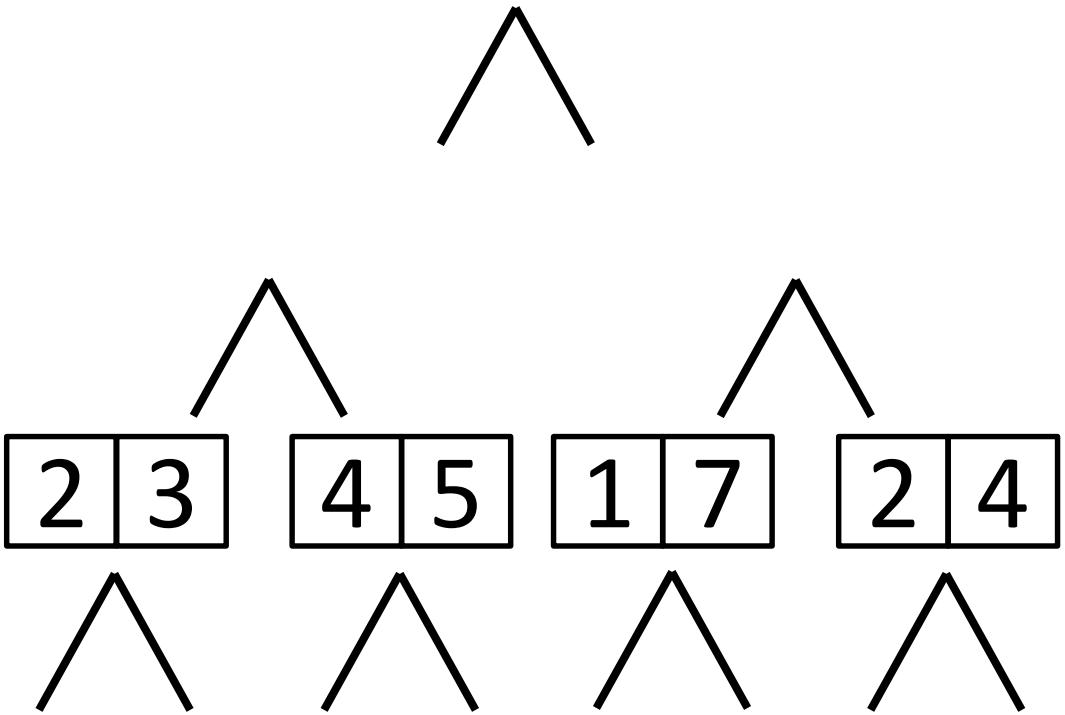
http://goo.gl/okU3fM



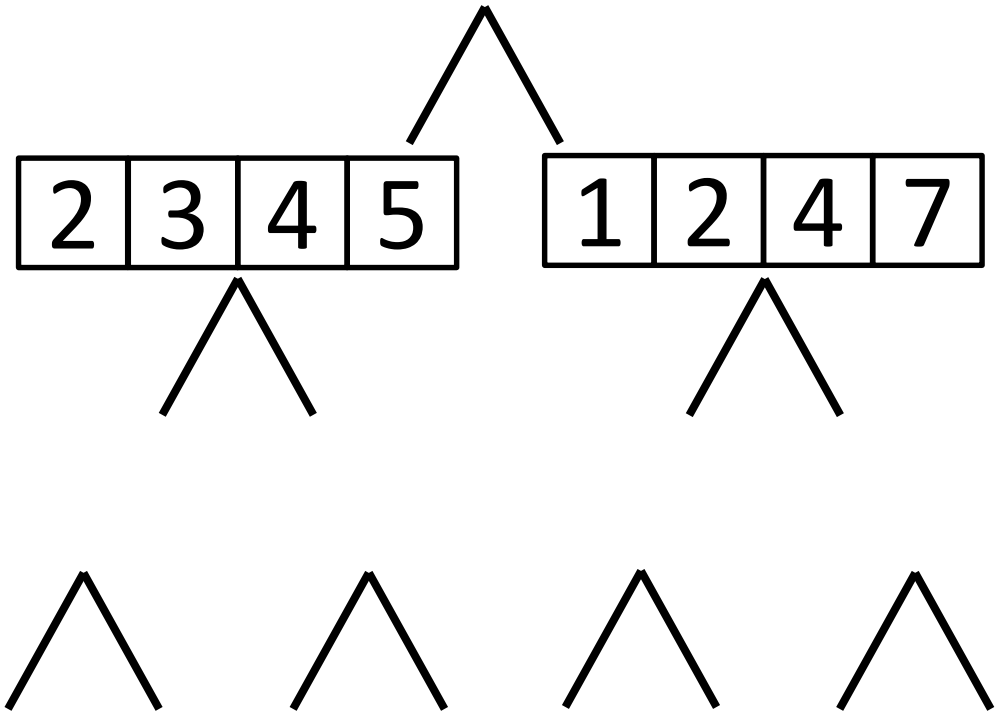
Parallel merge sort



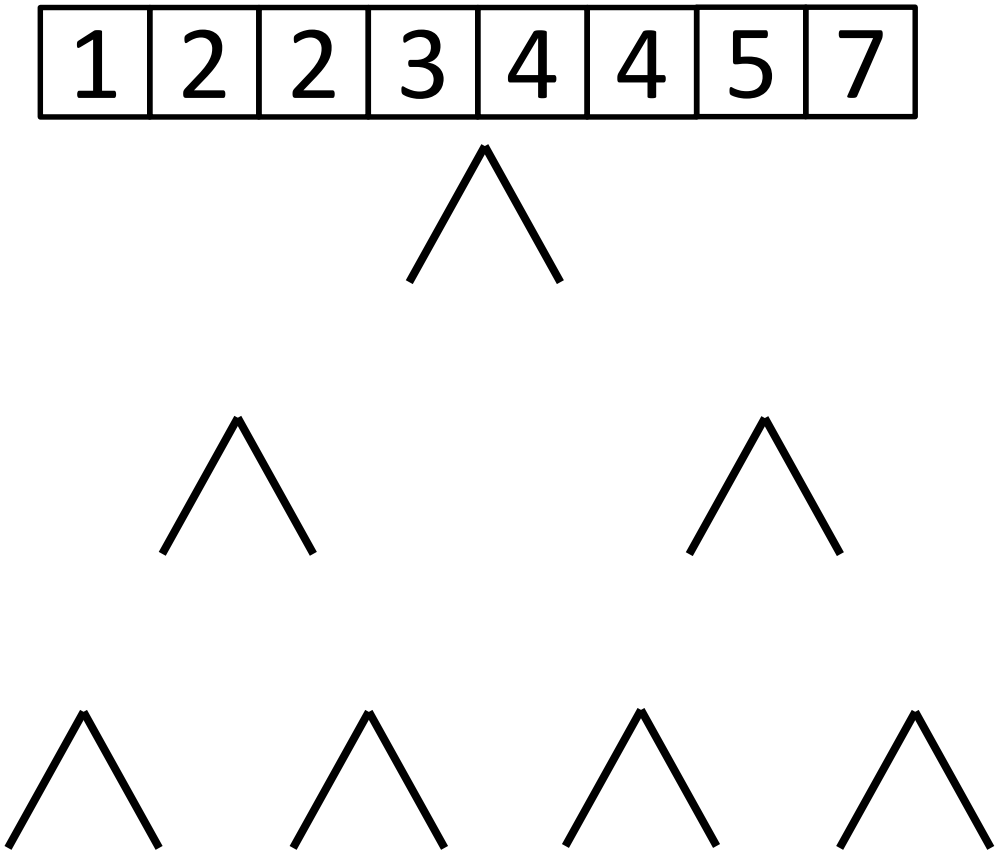
Parallel merge sort



Parallel merge sort



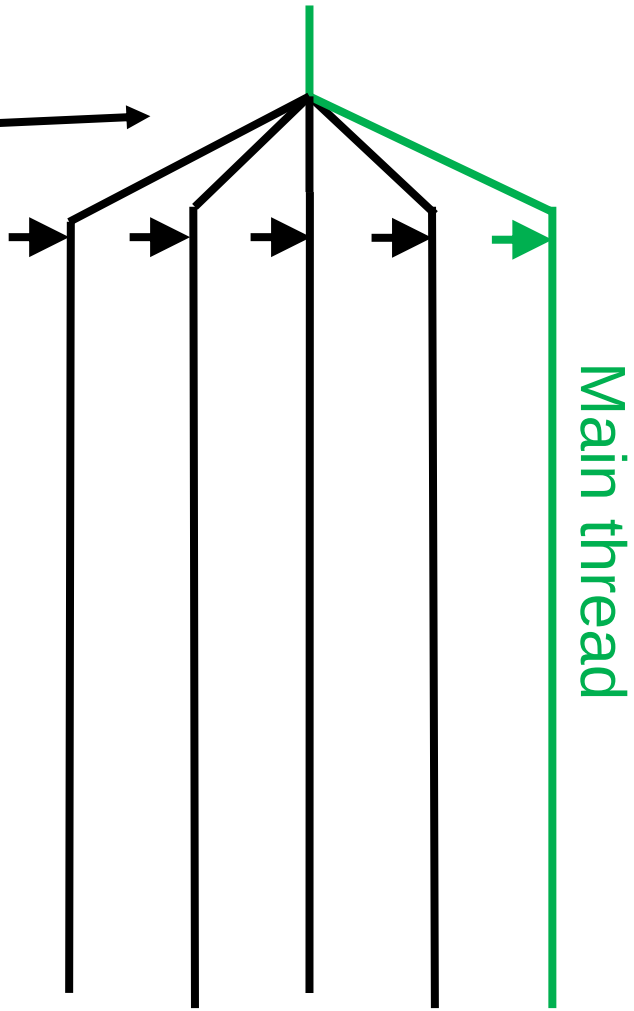
Parallel merge sort



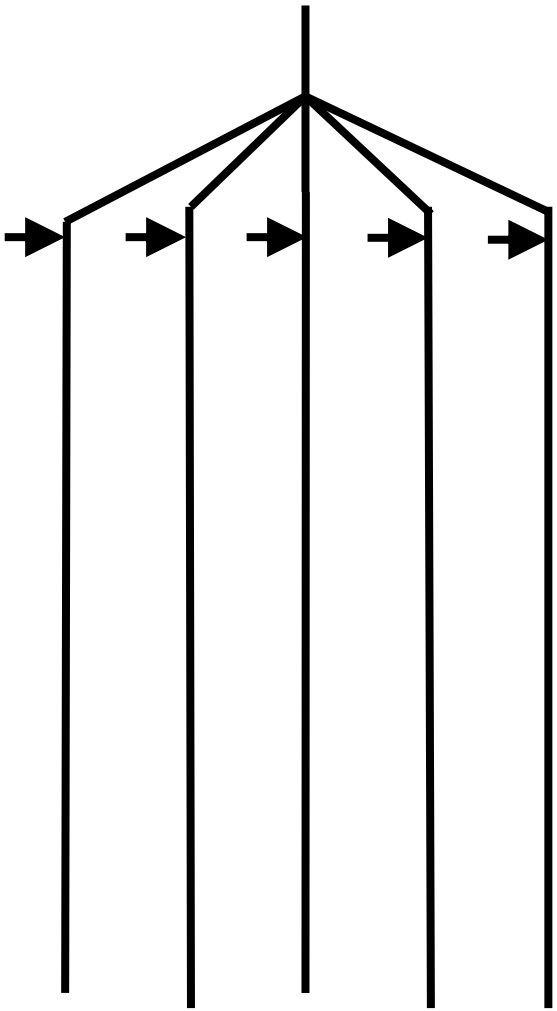

POLITEHNICA
UNIVERSITATEA
BUCUREŞTI
UPB
1818

Executor Service

```
ExecutorService tpe =
Executors.newFixedThreadPool(4);
```



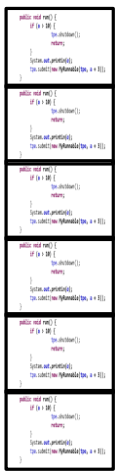
Executor Service

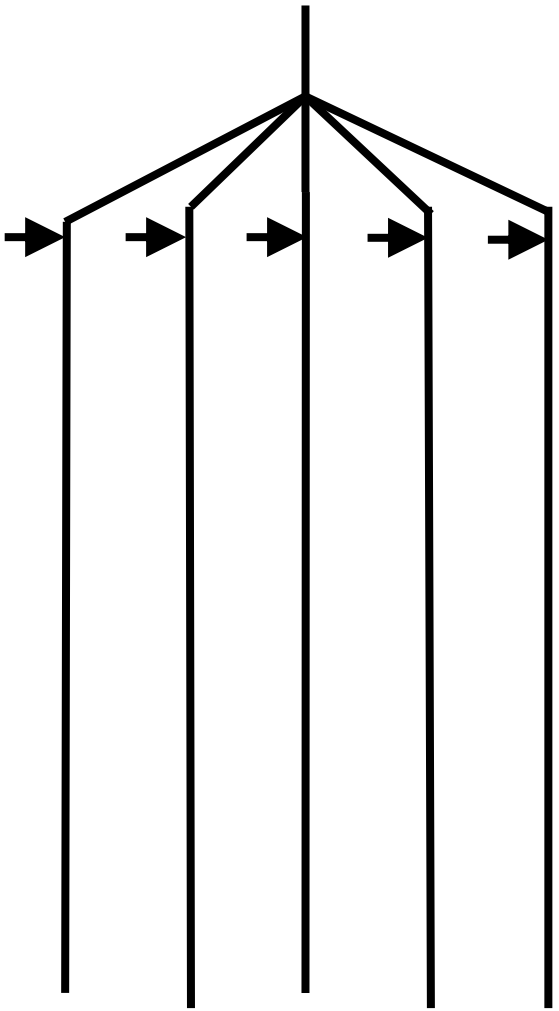


```
public class MyRunnable implements Runnable {
	int a;
	ExecutorService tpe;

	public MyRunnable(ExecutorService tpe, int a) {
		this.a = a;
		this.tpe = tpe;
	}

	@Override
	public void run() {
		if (a > 10) {
				tpe.shutdown();
				return;
		}
		System.out.println(a);
		tpe.submit(new MyRunnable(tpe, a + 3));
	}
}
```

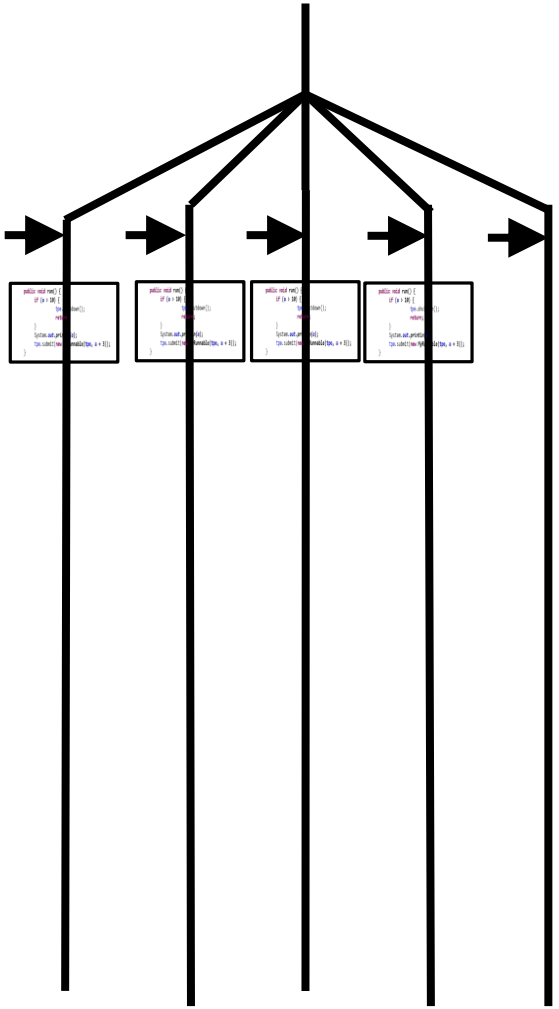
Executor Service

```
tpe.submit(new MyRunnable(tpe, 0));
tpe.submit(new MyRunnable(tpe, 1));
tpe.submit(new MyRunnable(tpe, 2));
……
```





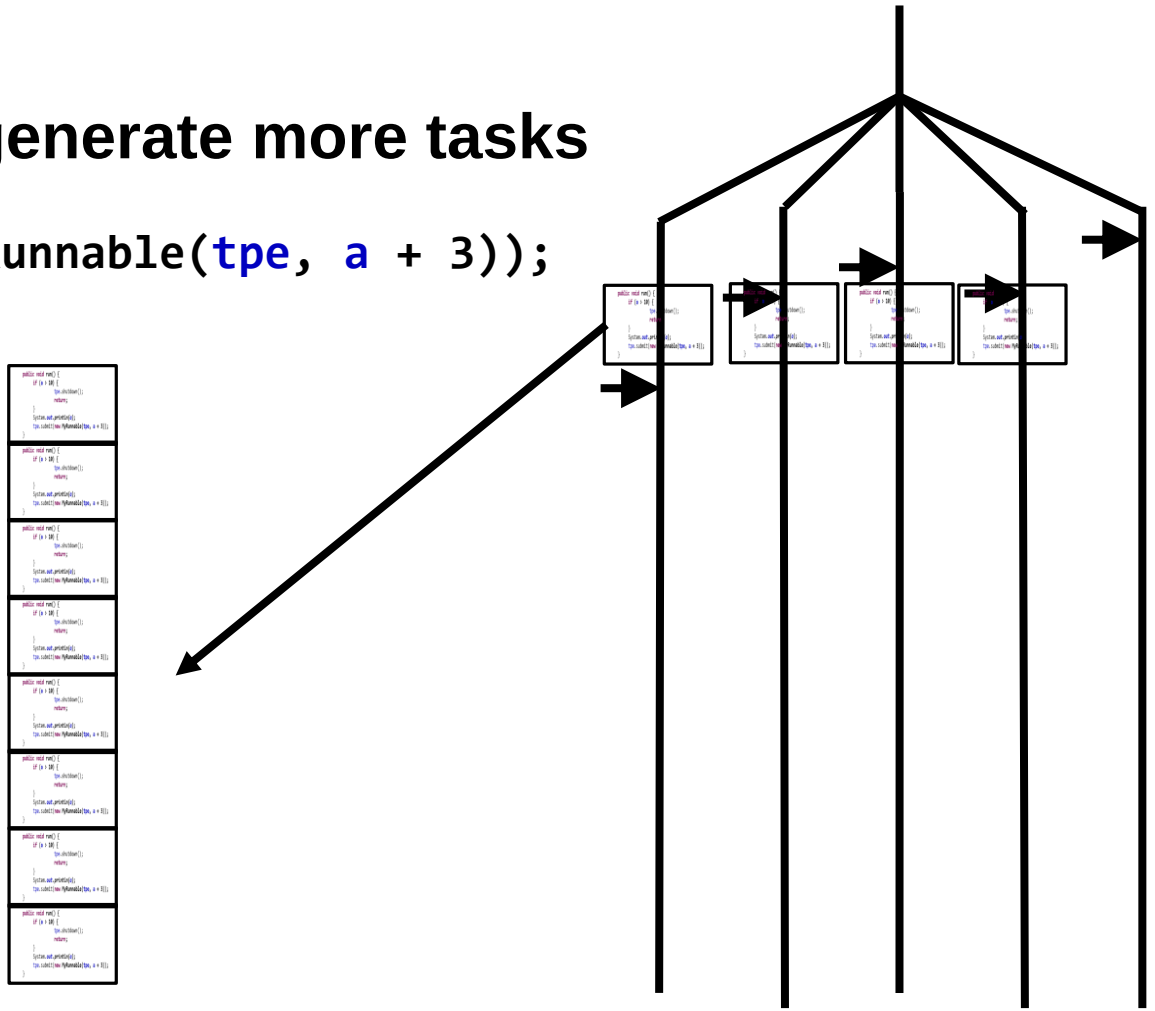
Executor Service



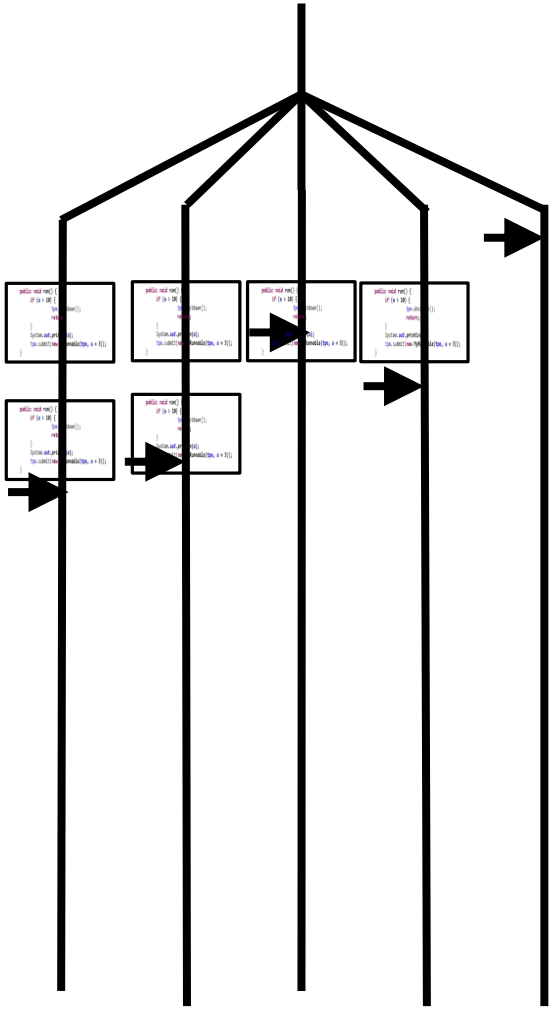
Executor Service

Tasks can generate more tasks

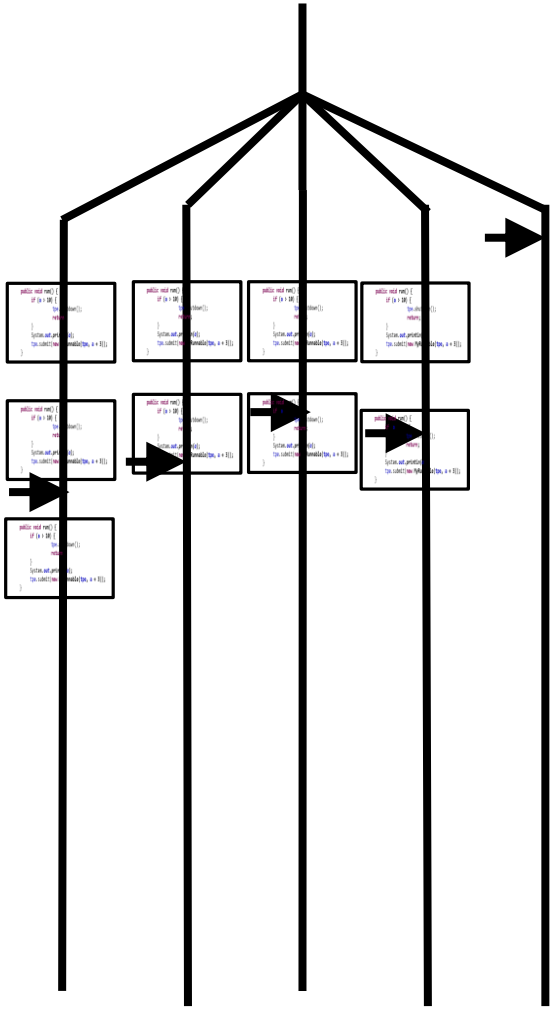
```
tpe.submit(new MyRunnable(tpe, a + 3));
```



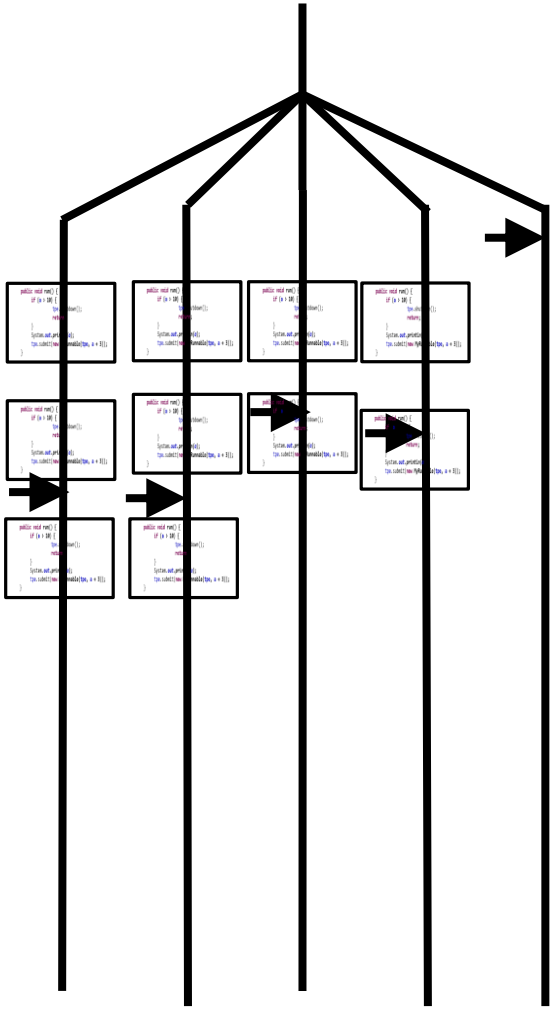
Executor Service



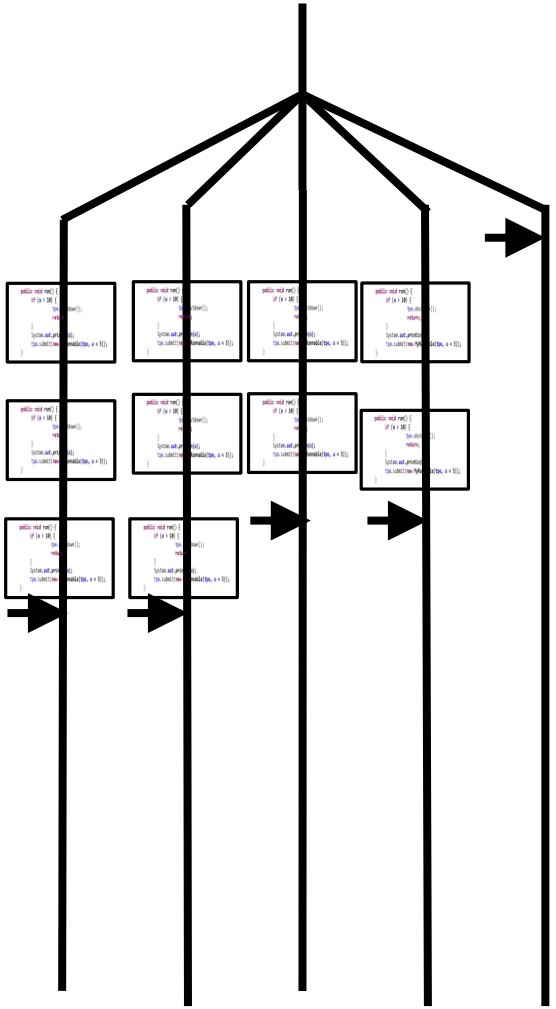
Executor Service



Executor Service

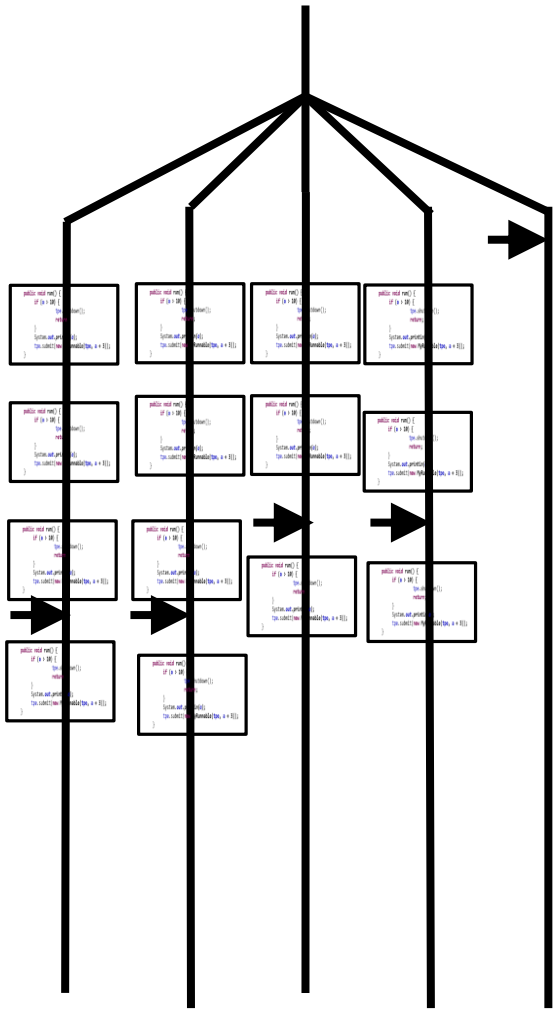


Executor Service



Executor Service

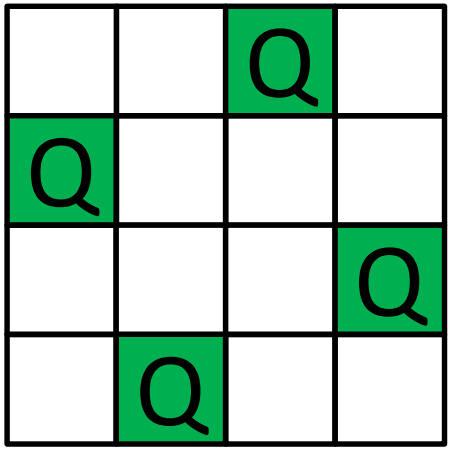
When do you stop the threads?
Depends on the problem.
Sometimes one solution is enough.
But some problems might not have solutsions.



```
tpe.shutdown();
```

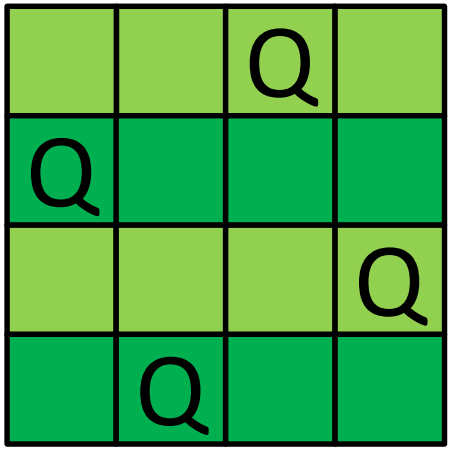

POLITEHNICA
UNIVERSITATEA
BUCUREŞTI
UPB
1818

N Queens Problem



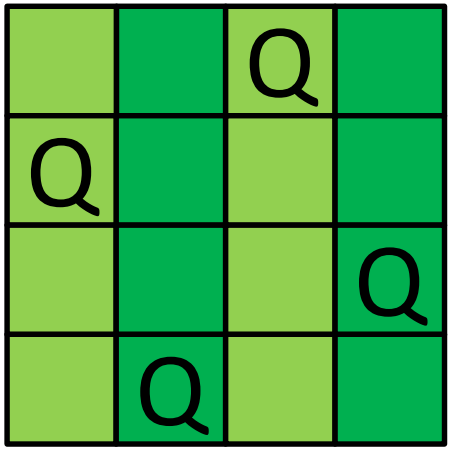
N Queens Problem

No more then one queen per line



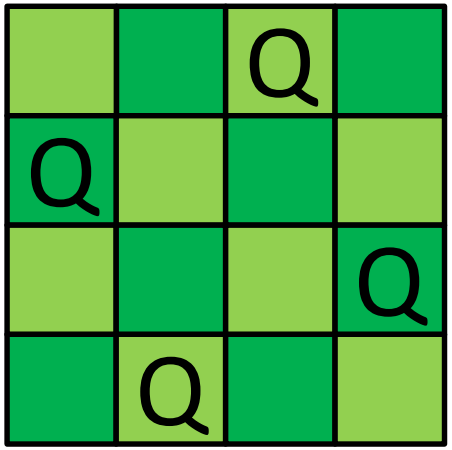
N Queens Problem

No more then one queen per column

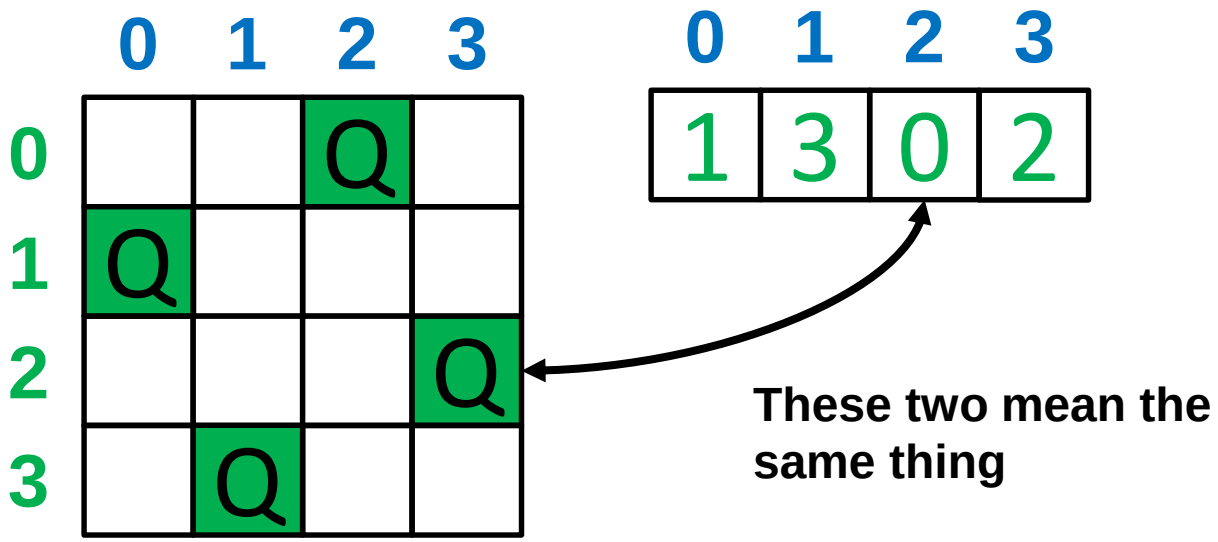


N Queens Problem

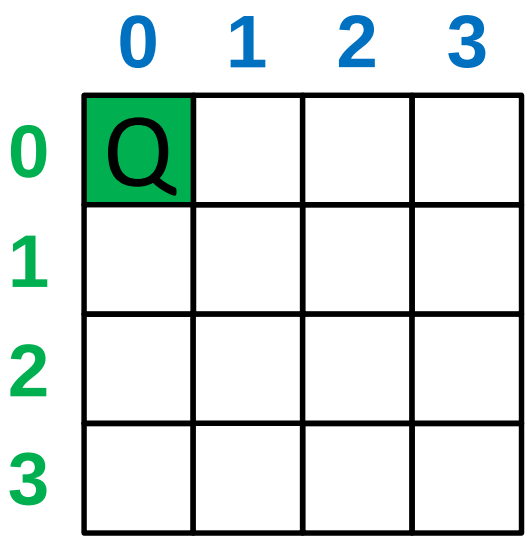
No more then one queen per diagonals

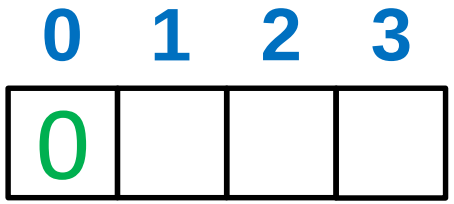


N Queens Problem

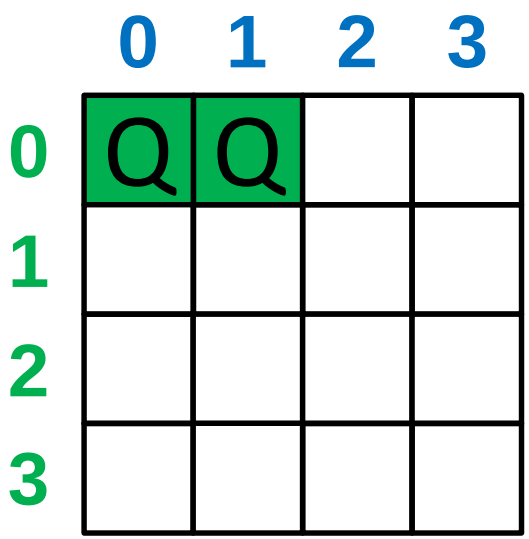


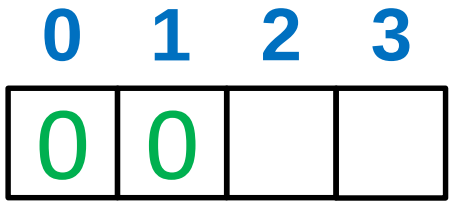
N Queens Problem - Solution





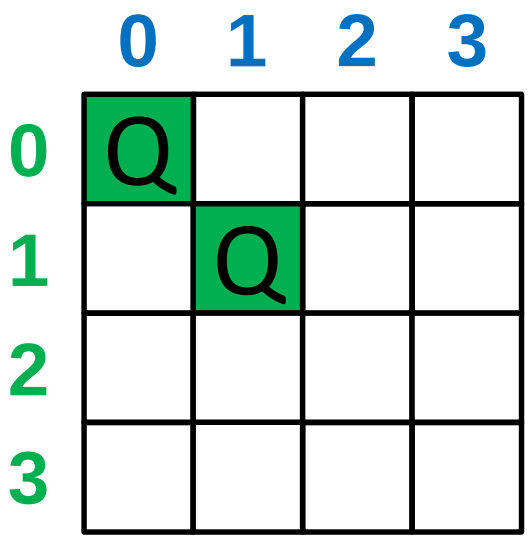
N Queens Problem - Solution

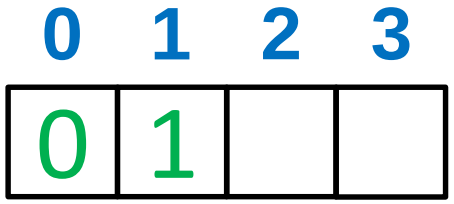




Line conflict

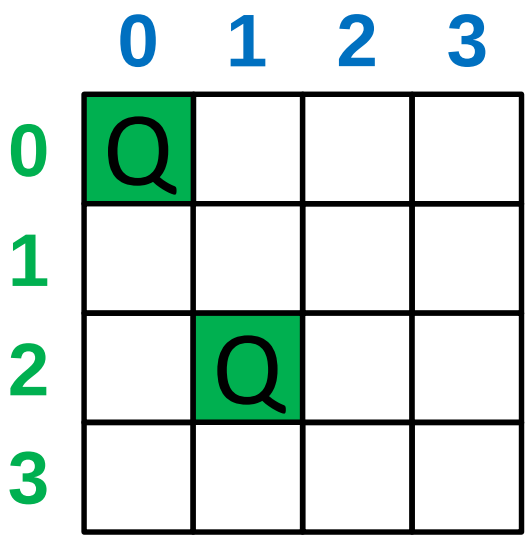
N Queens Problem - Solution





Diagonal conflict

N Queens Problem - Solution

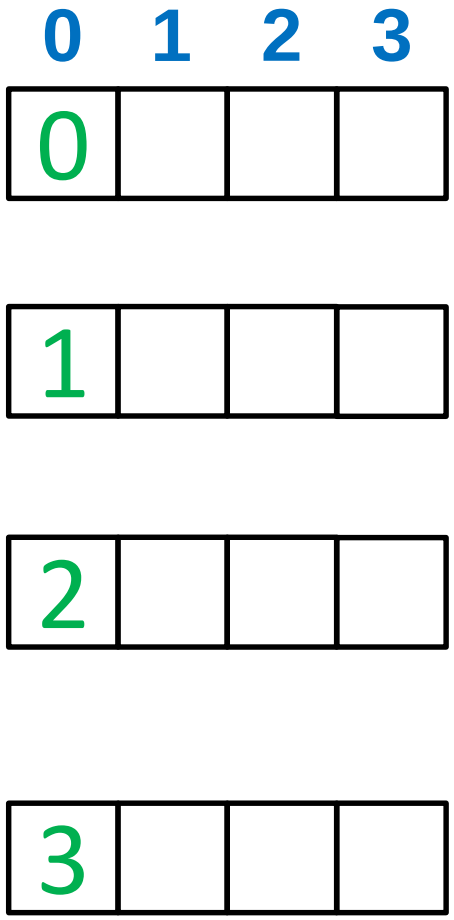


0	1	2	3
0	2		

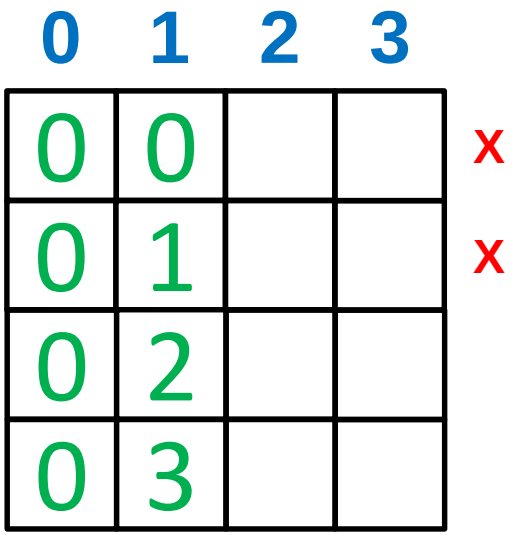
OK.

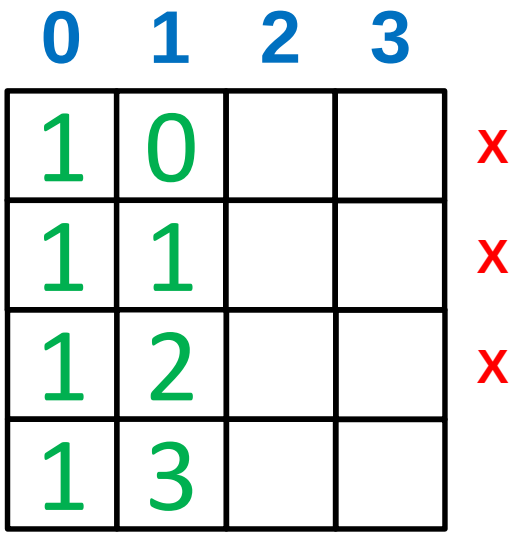
And so on…

N Queens Problem – Parallel Solution

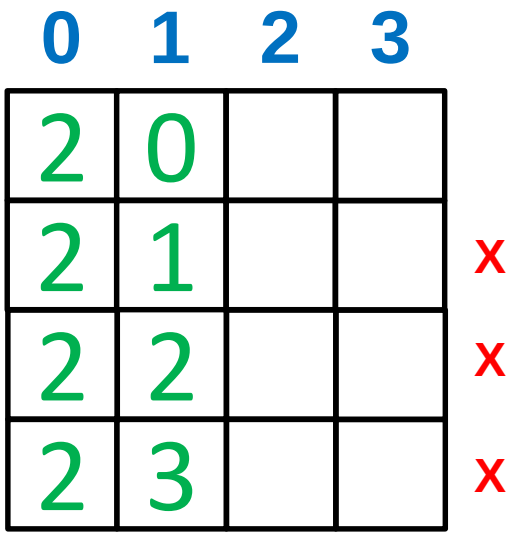


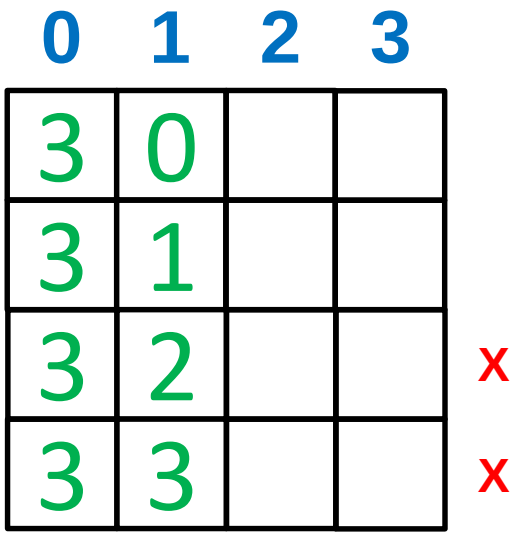
N Queens Problem – Parallel Solution





And so on…





Abordarea log sau arbore

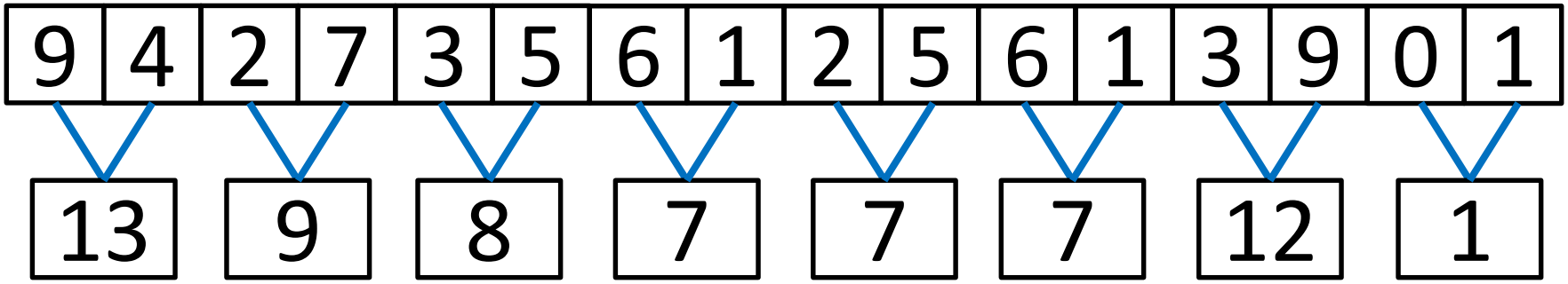
Reduce

Apply the same operation between all the elements of a vector.

Can be executed in log(n) time using a tree form.

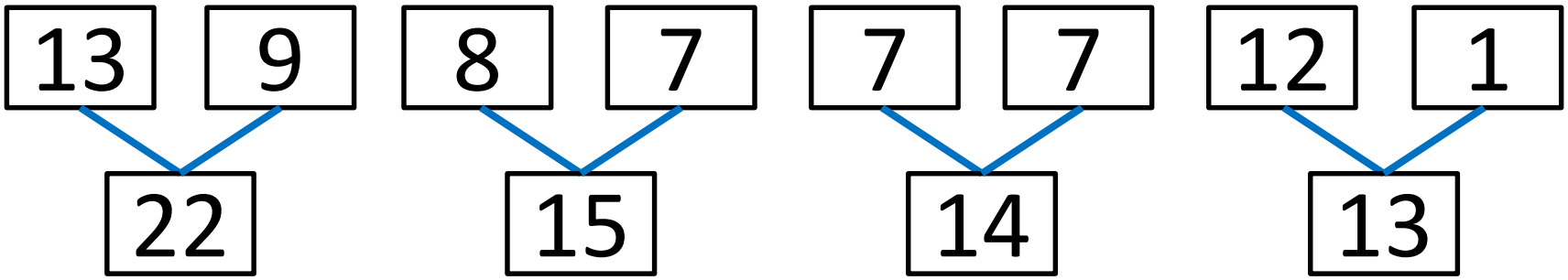
The operation can take many forms (+, *, min, max, and, etc.)

Reduce - sum



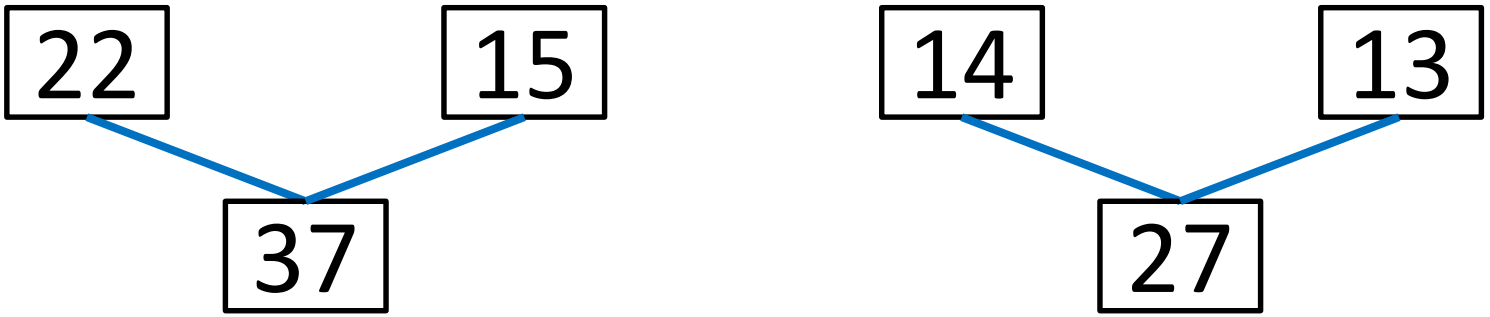
Can all be executed in parallel

Reduce - sum



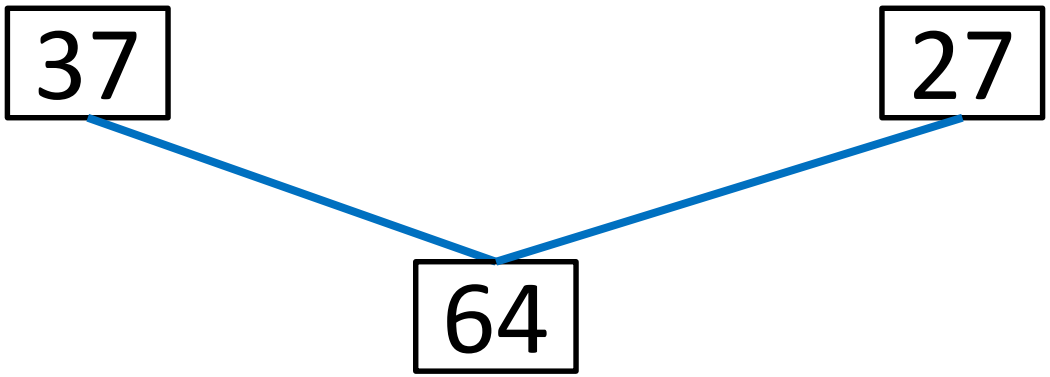
Can all be executed in parallel

Reduce - sum

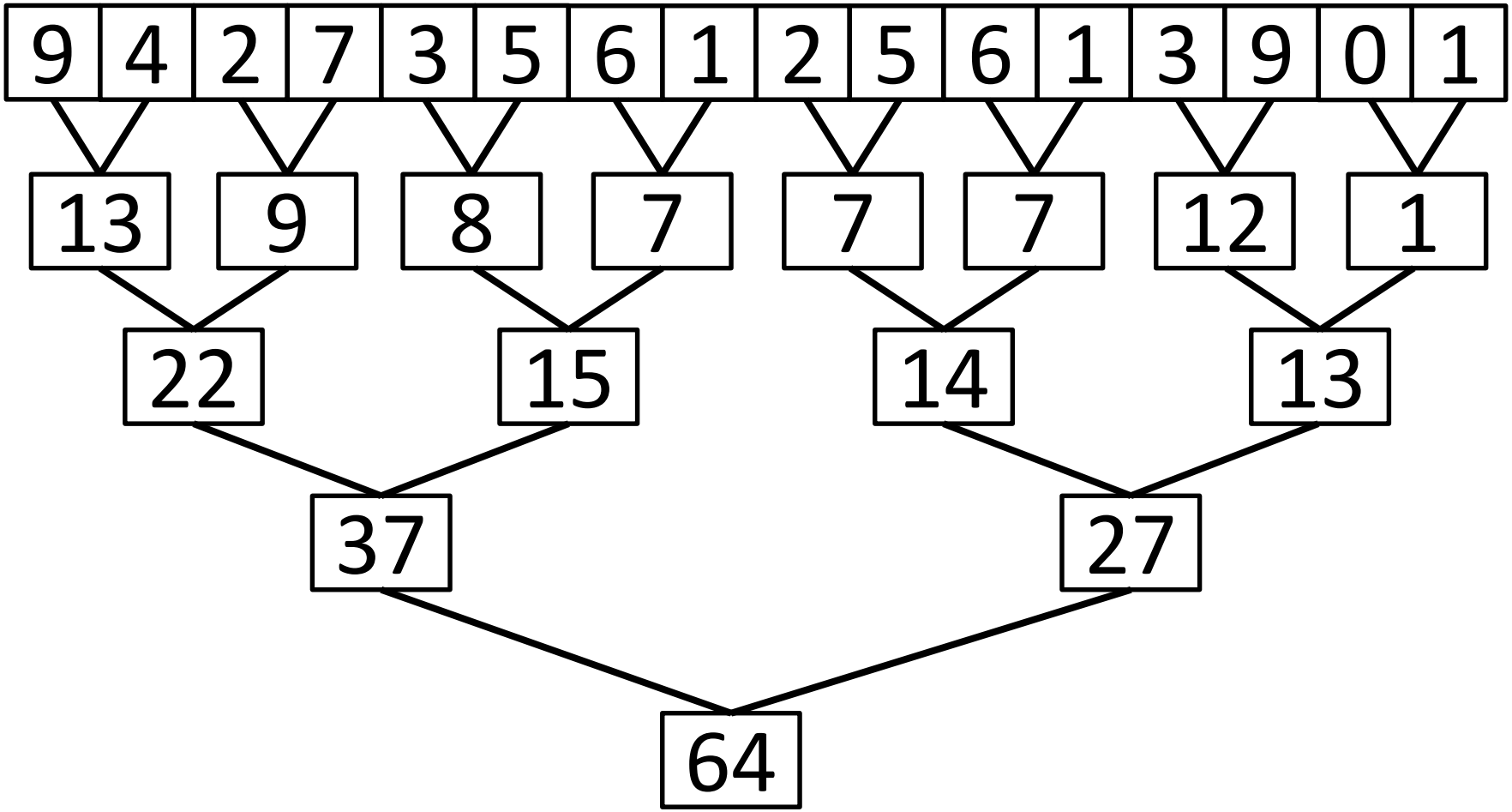


Can all be executed in parallel

Reduce - sum



Reduce - sum




POLITEHNICA
UNIVERSITATEA
BUCUREŞTI
UPB
1818

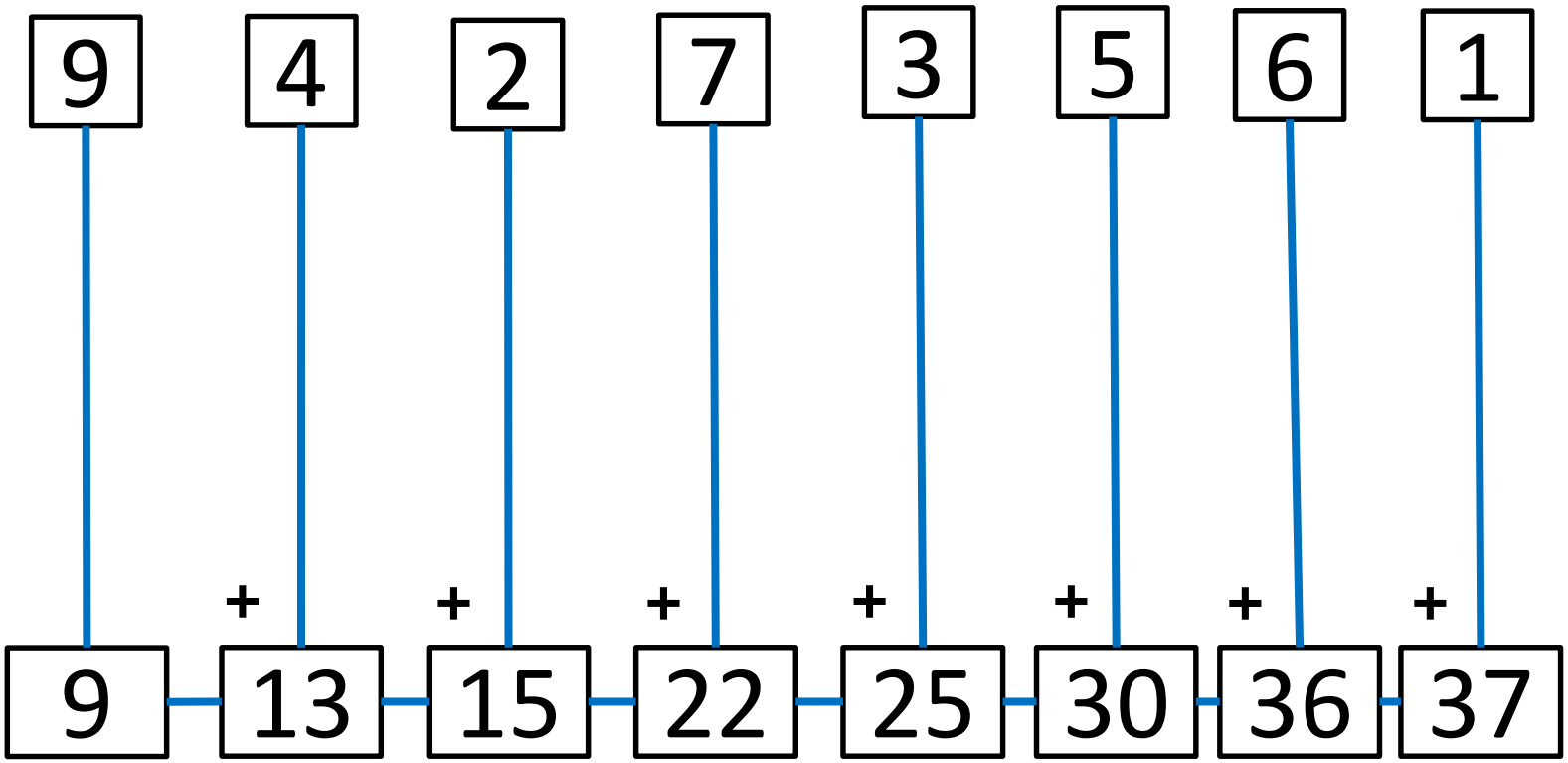
Scan

Apply the same operation between all the elements of a vector.
Obtains all partial results.

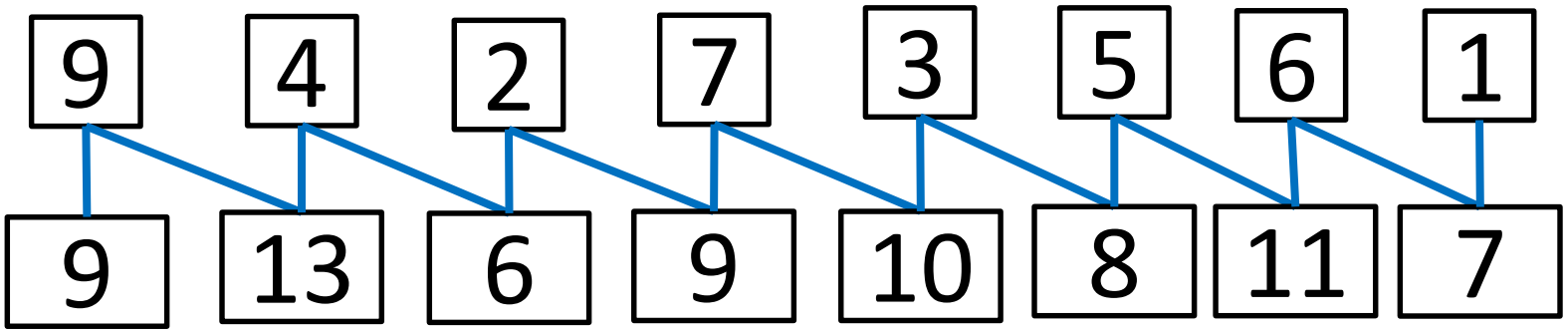
Can be executed in log(n) time using a special tree form.

The operation can take many forms (+, *, min, max, and, etc.)

Scan - sum

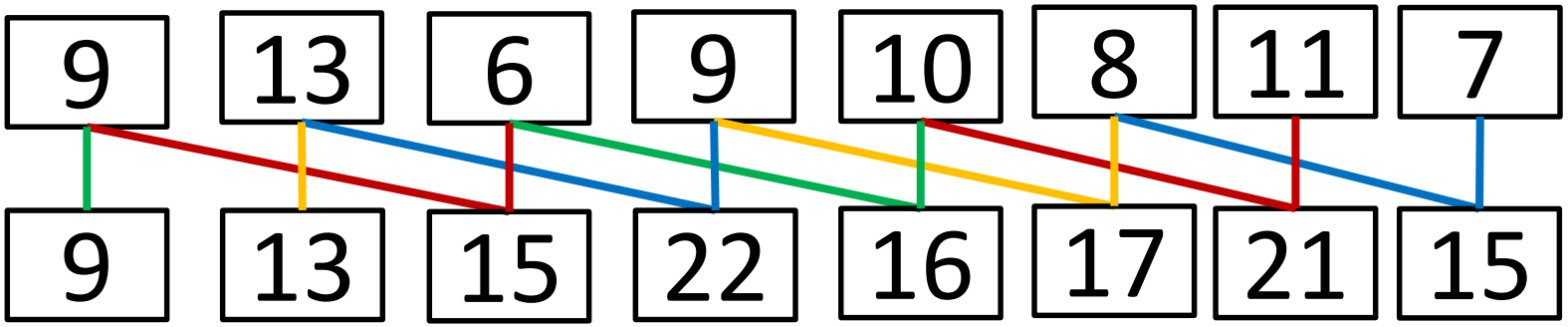


Scan - sum



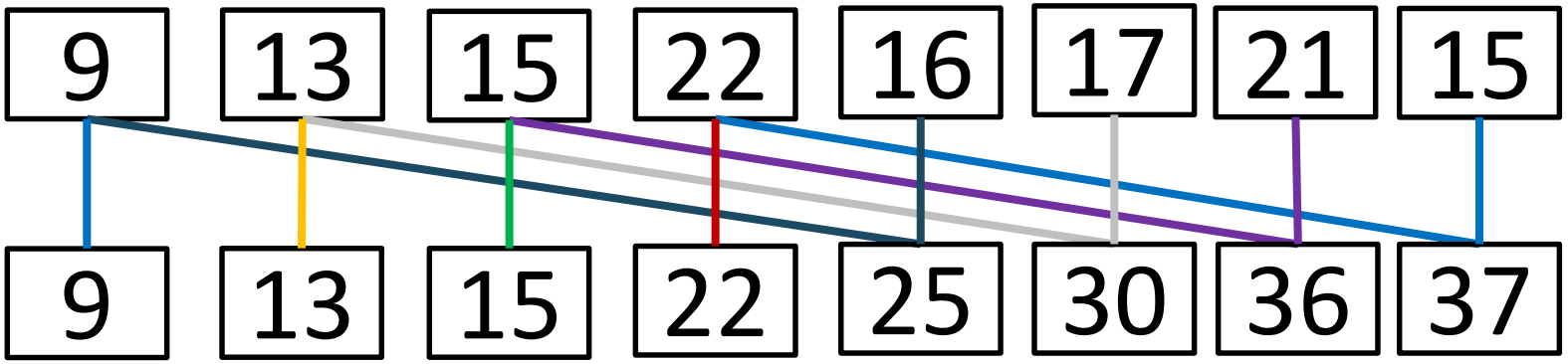
Can all be executed in parallel

Scan - sum



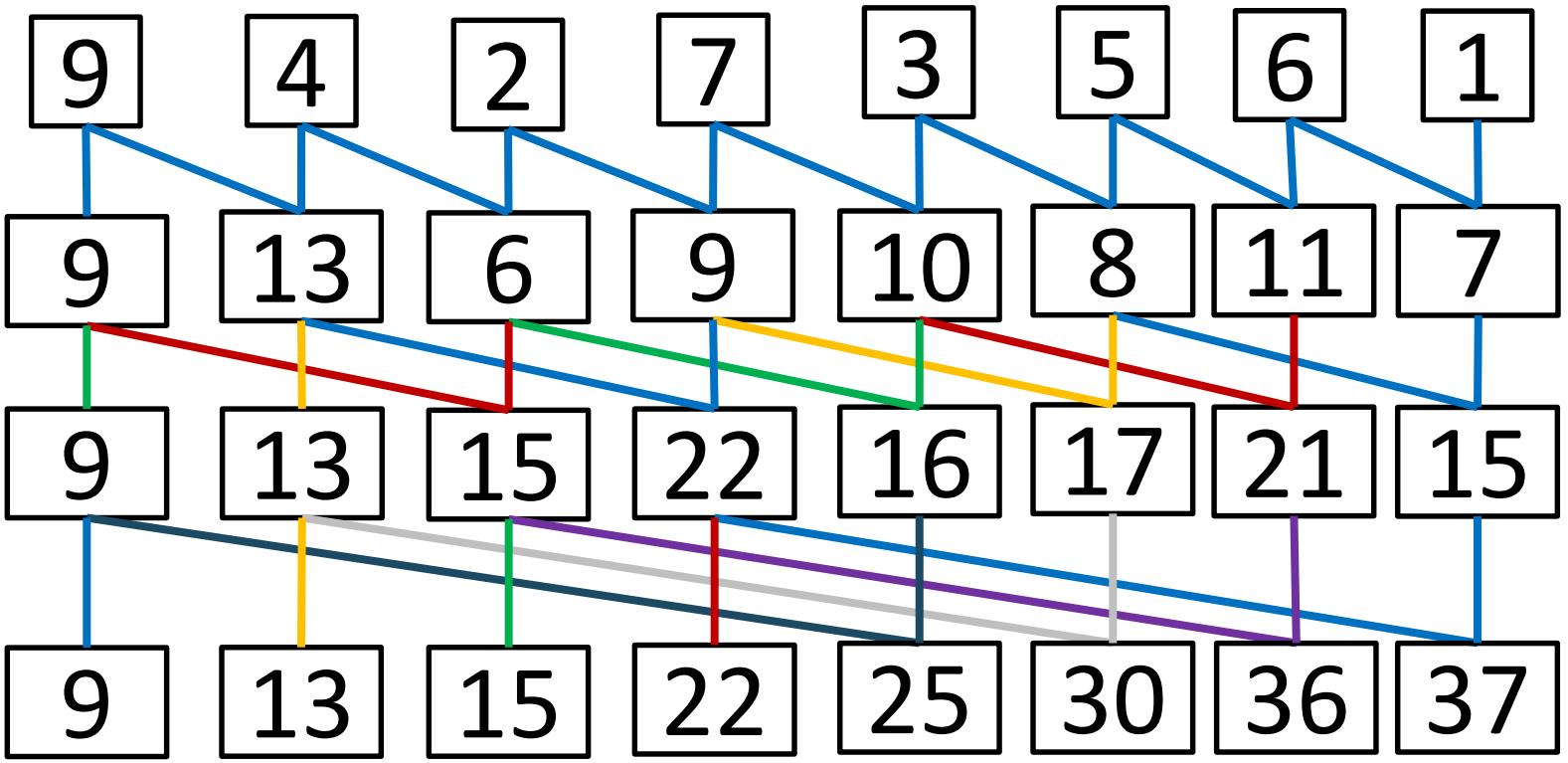
Can all be executed in parallel

Scan - sum

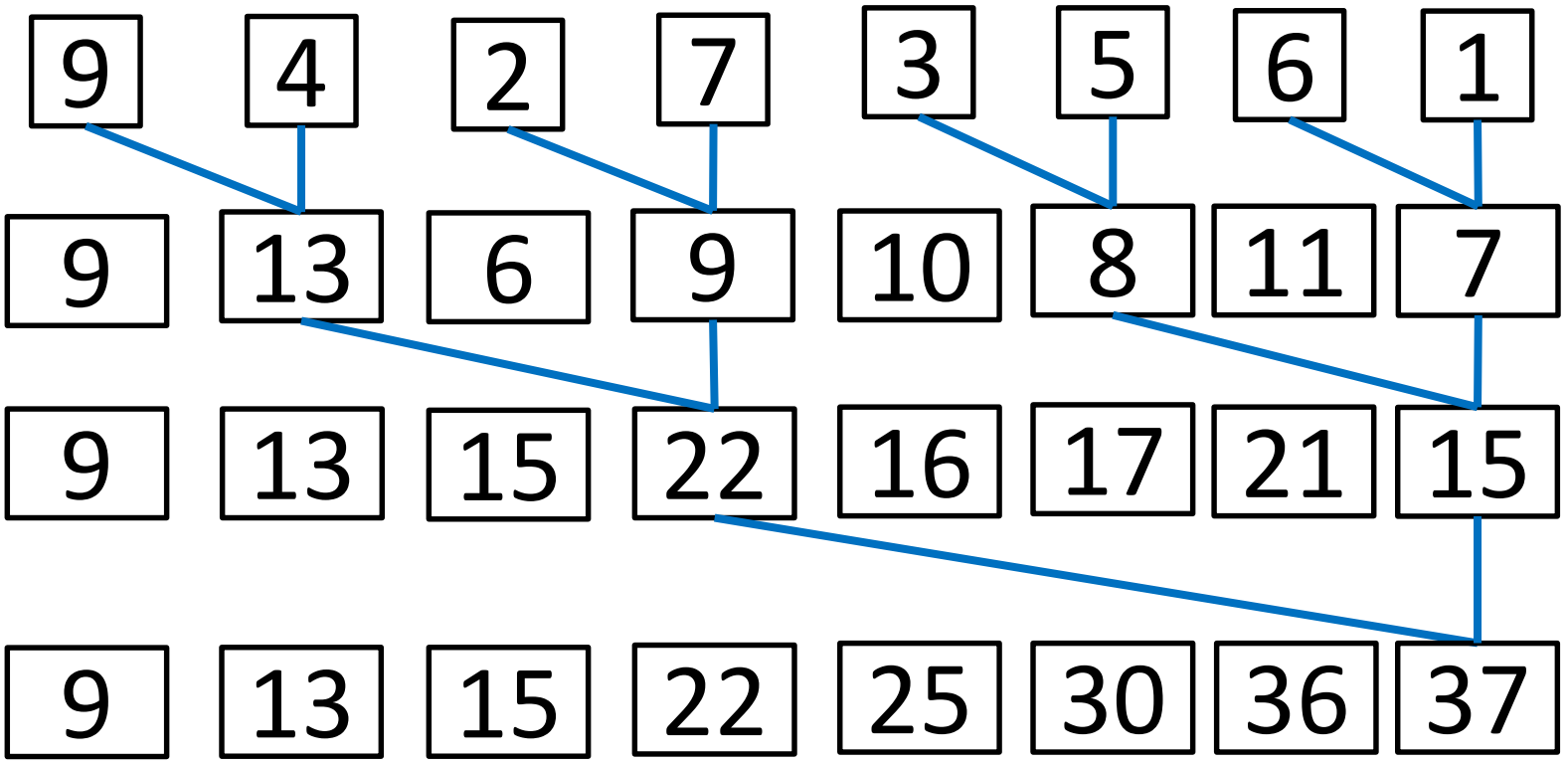


Can all be executed in parallel

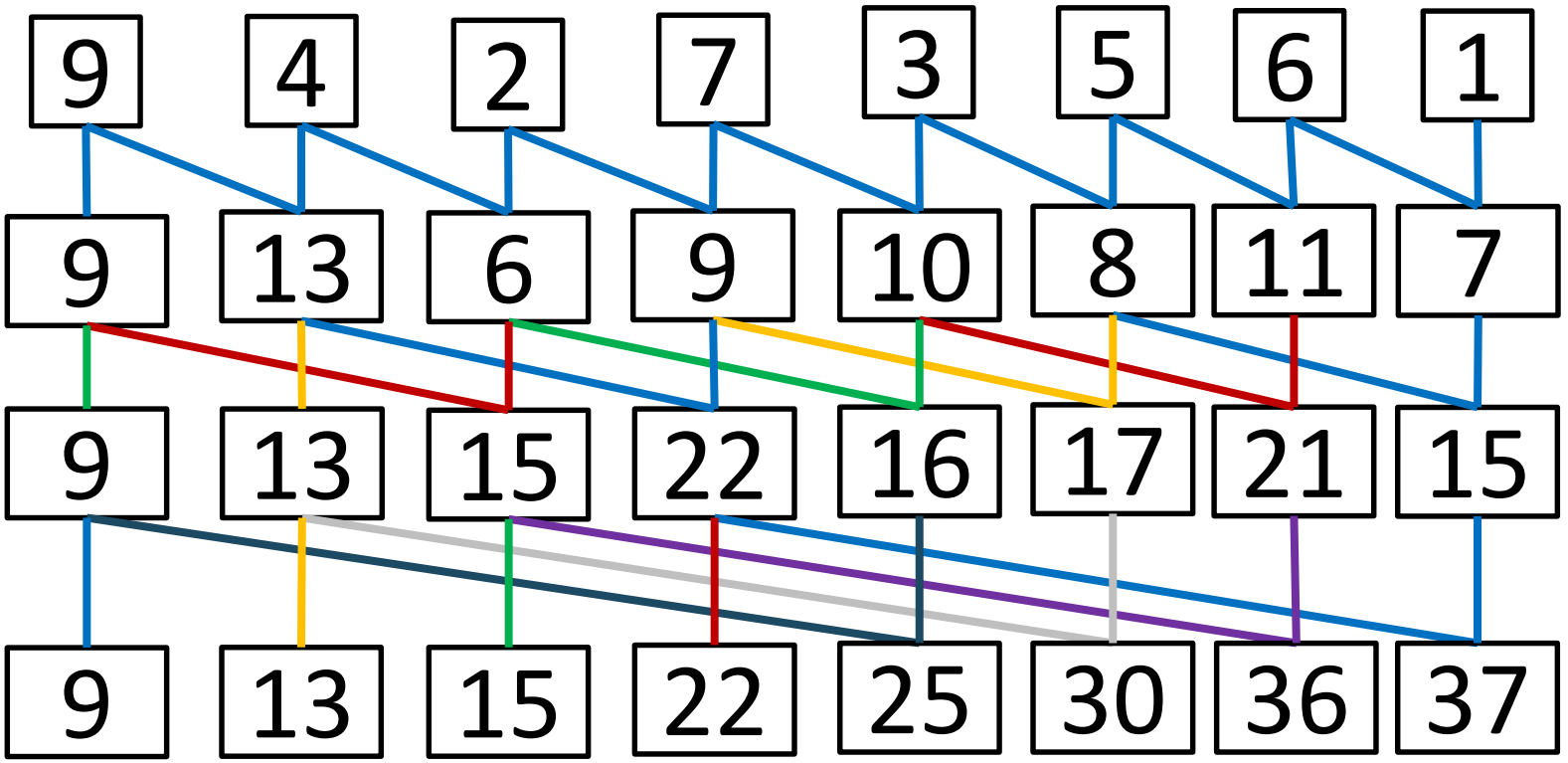
Scan – How it works?



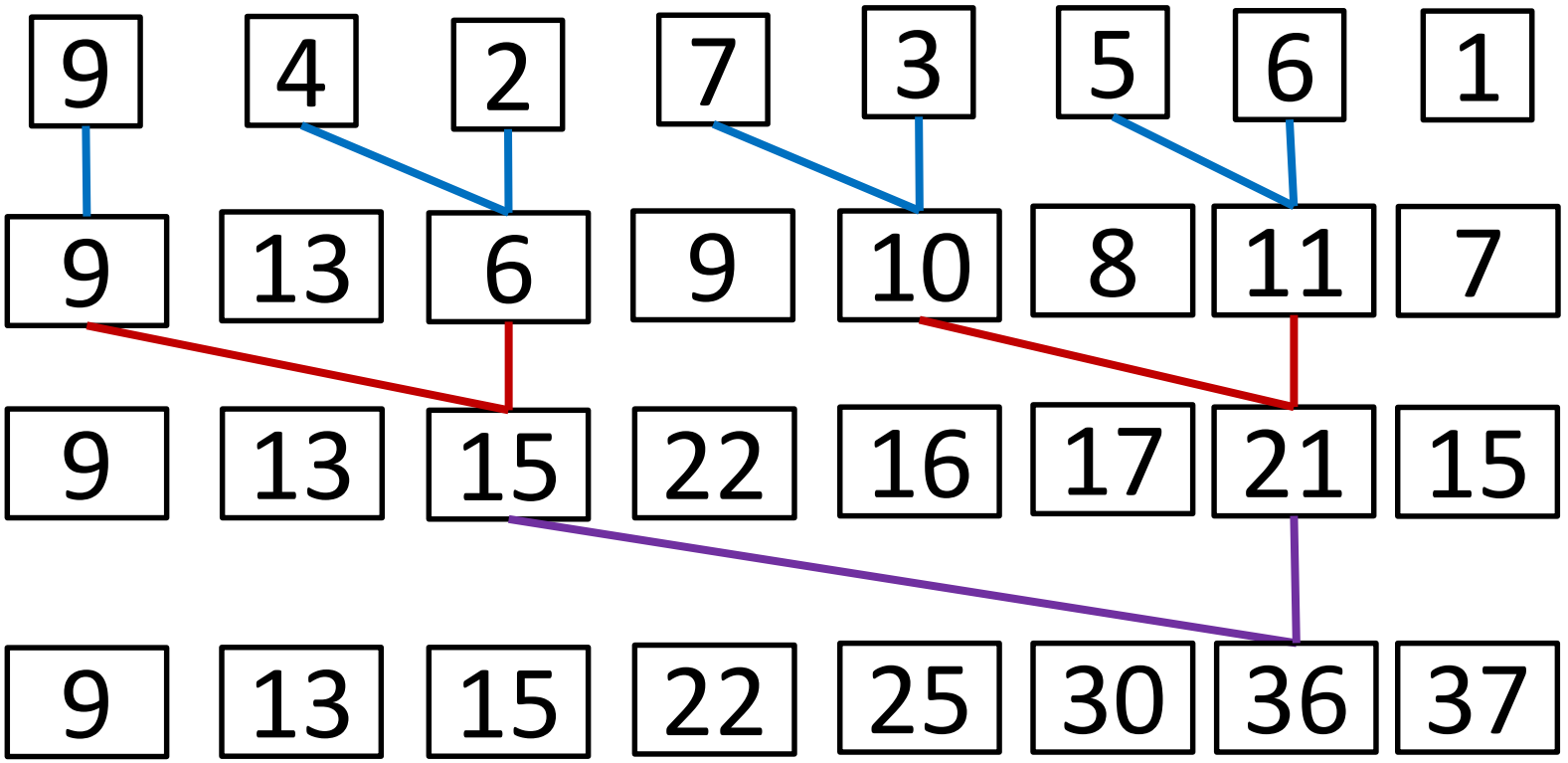
Scan – How it works?



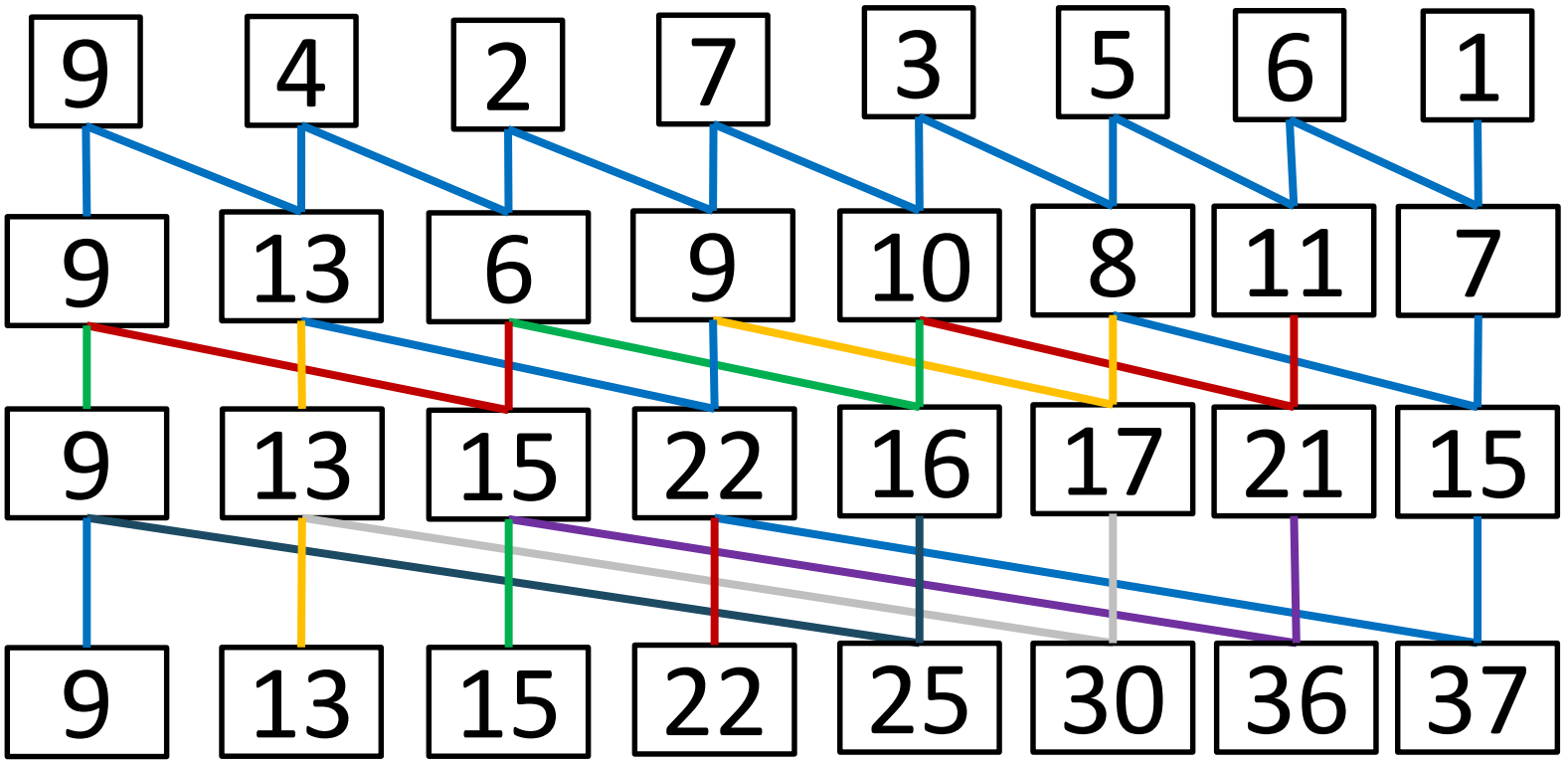
Scan – How it works?



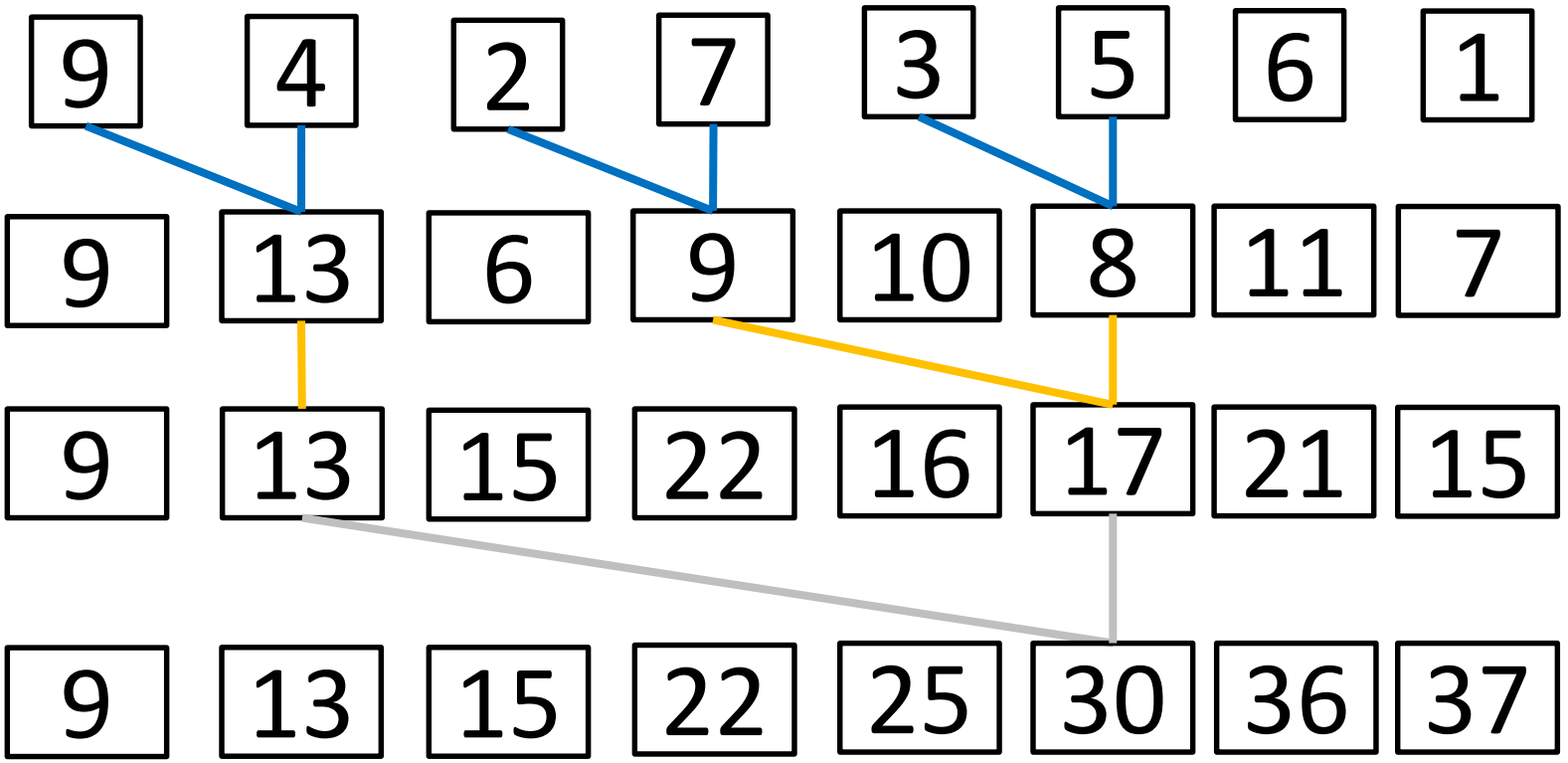
Scan – How it works?



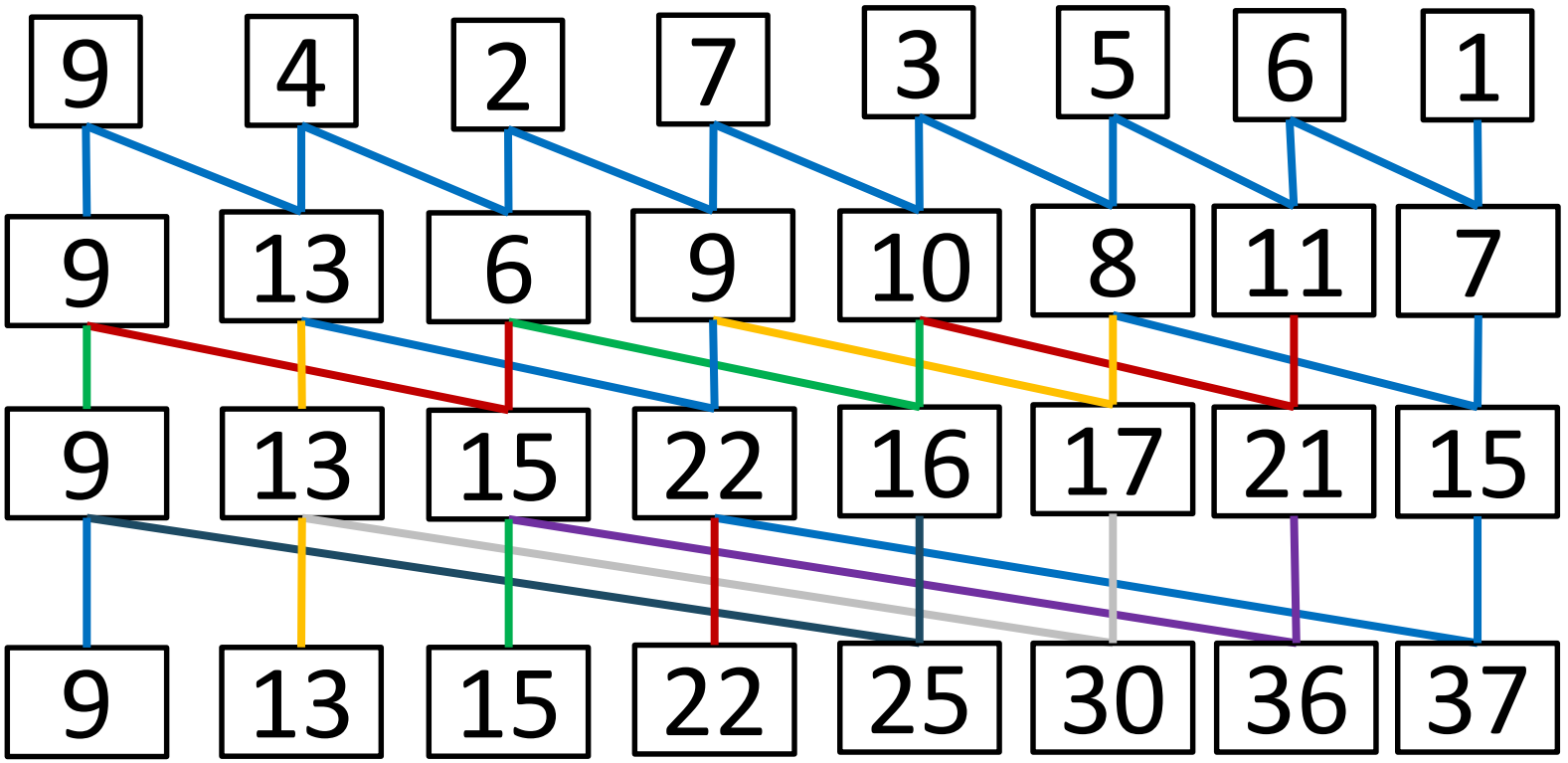
Scan – How it works?



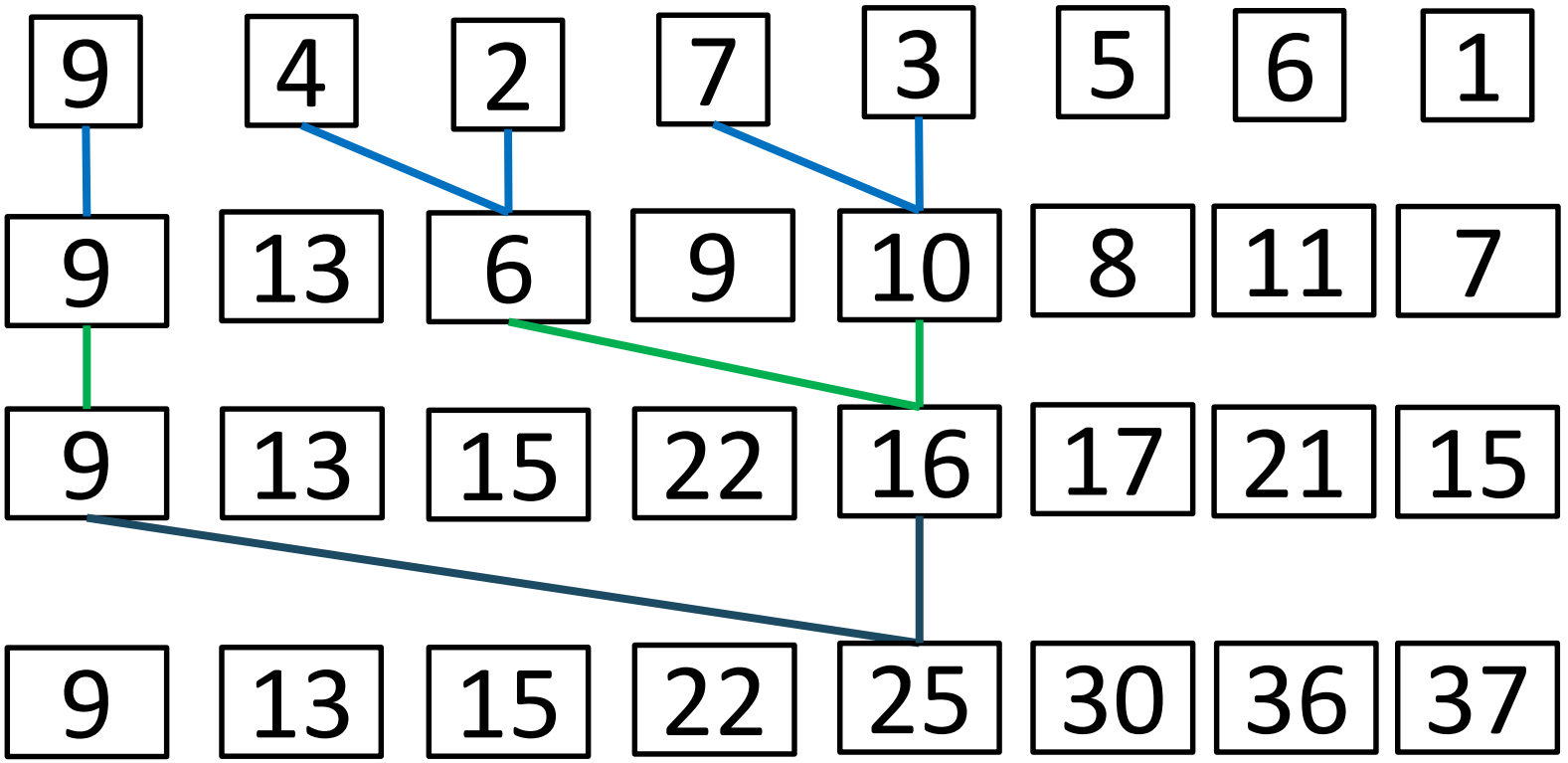
Scan – How it works?



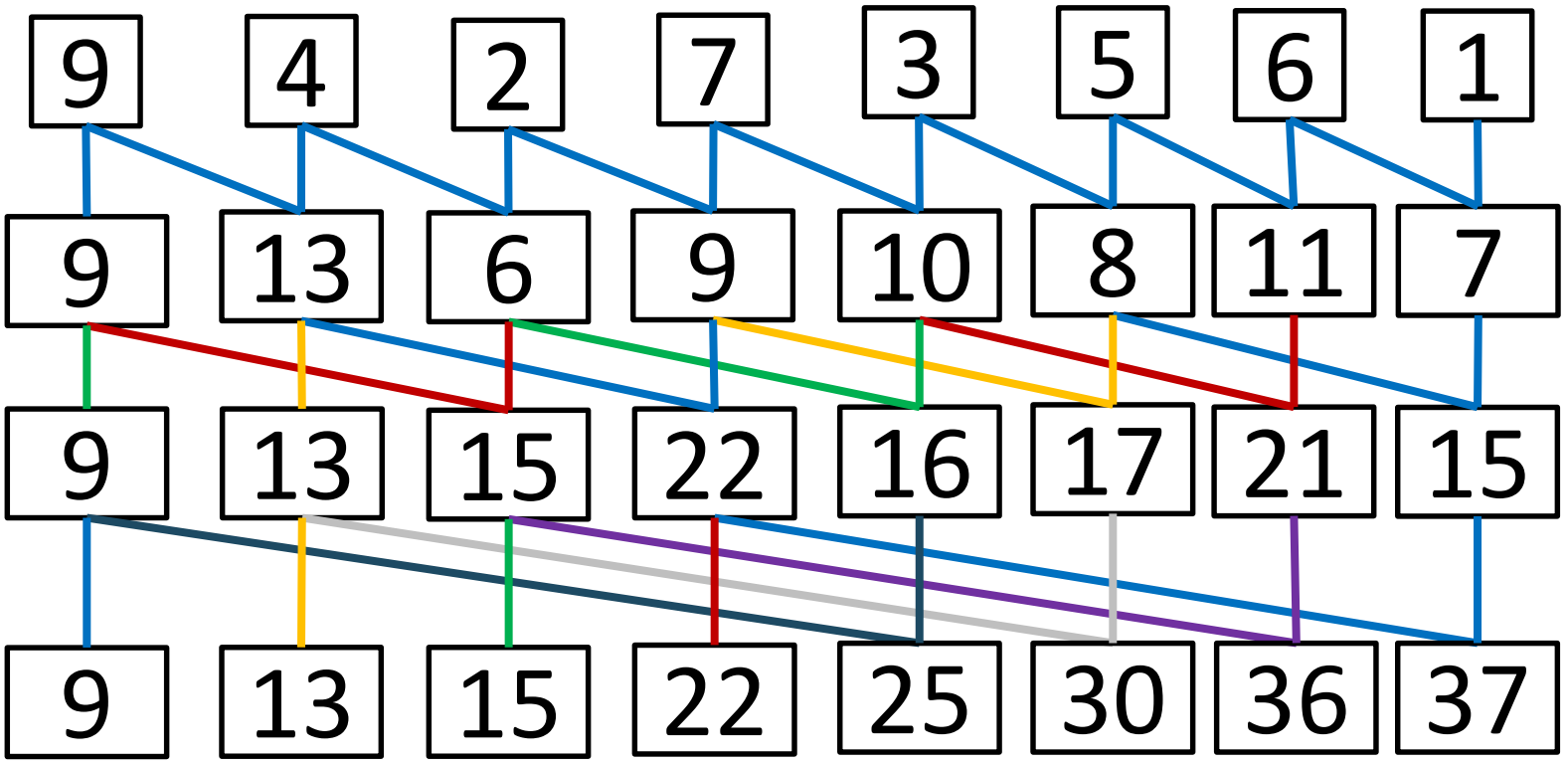
Scan – How it works?



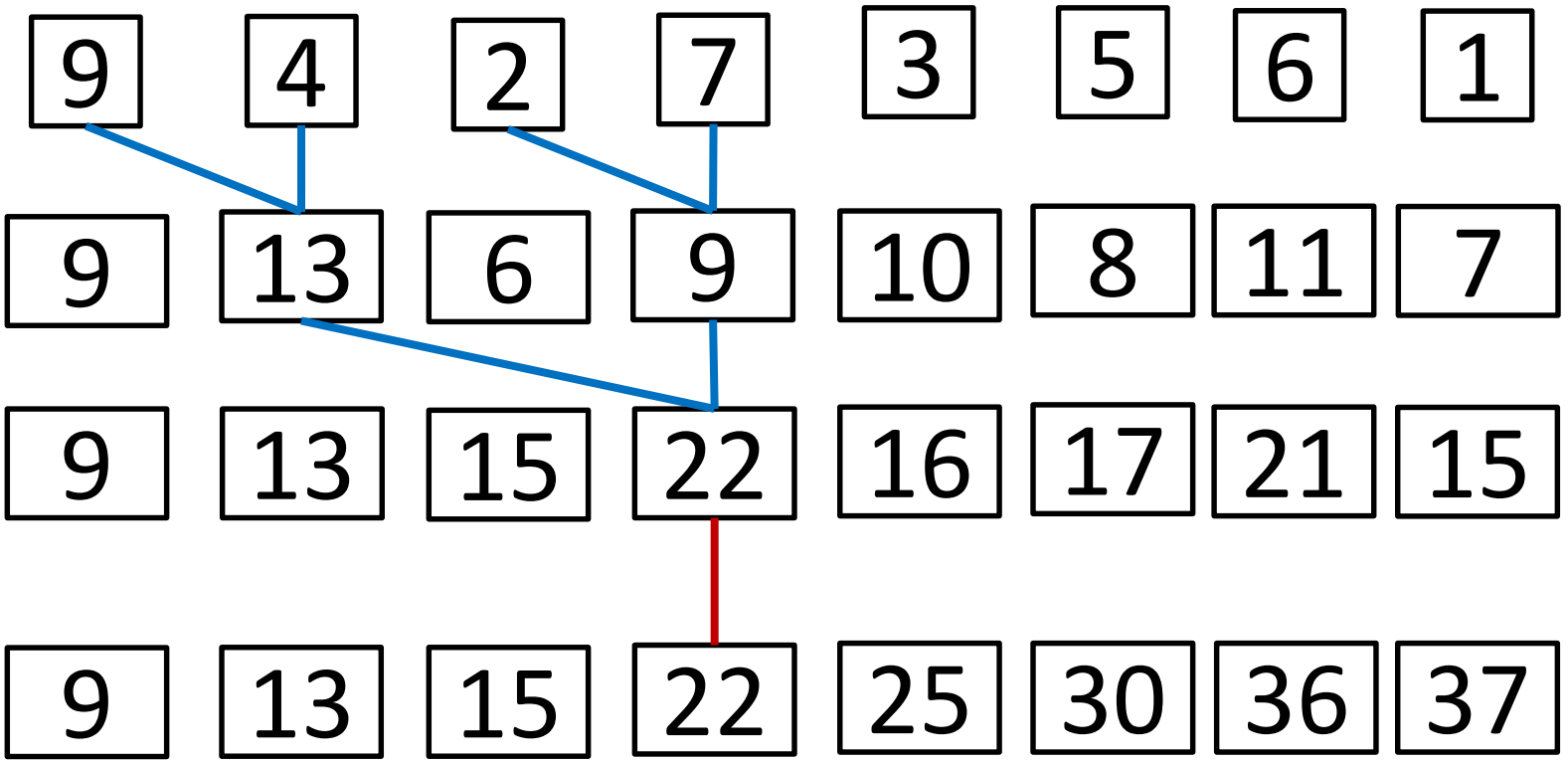
Scan – How it works?



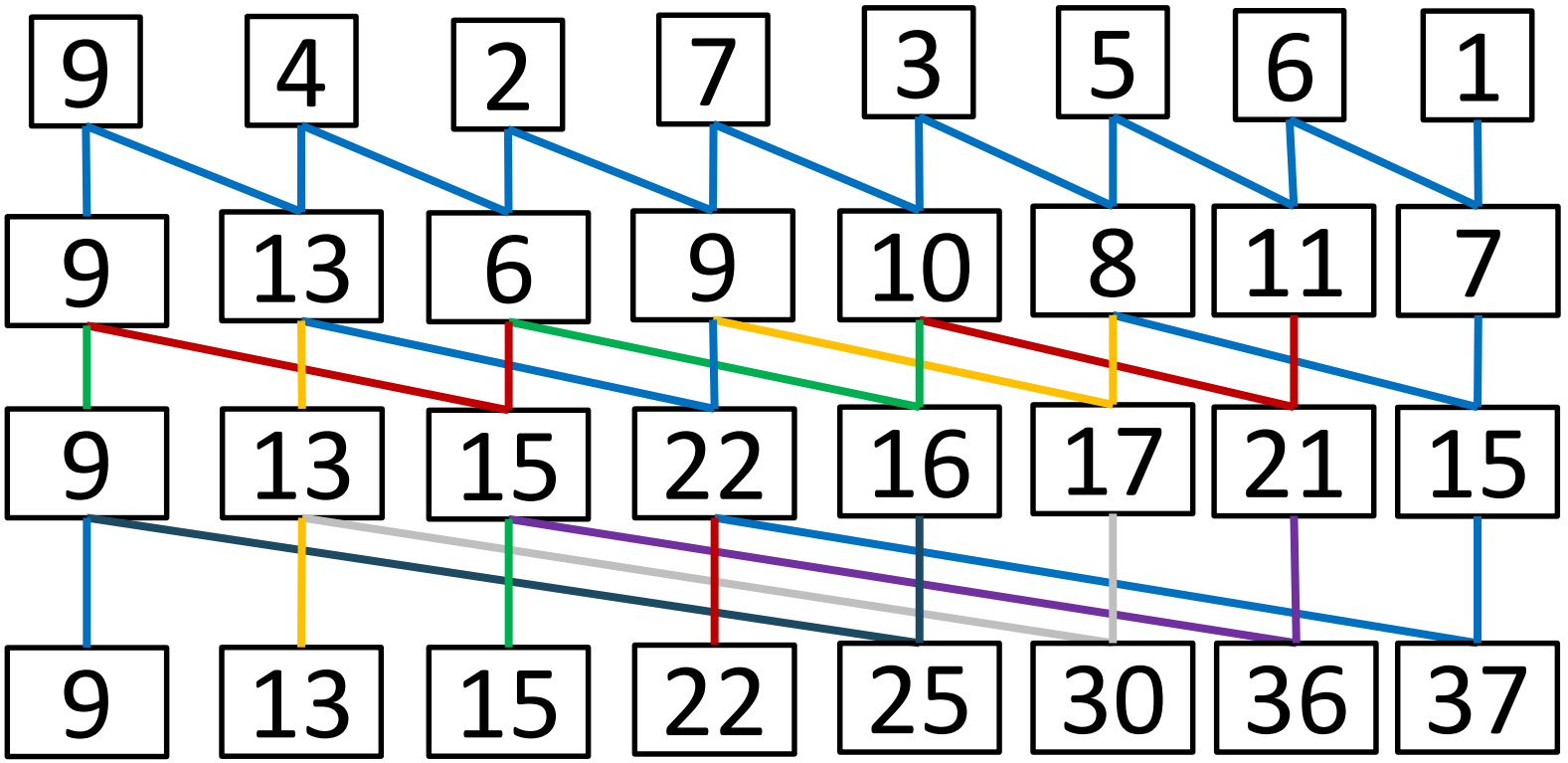
Scan – How it works?



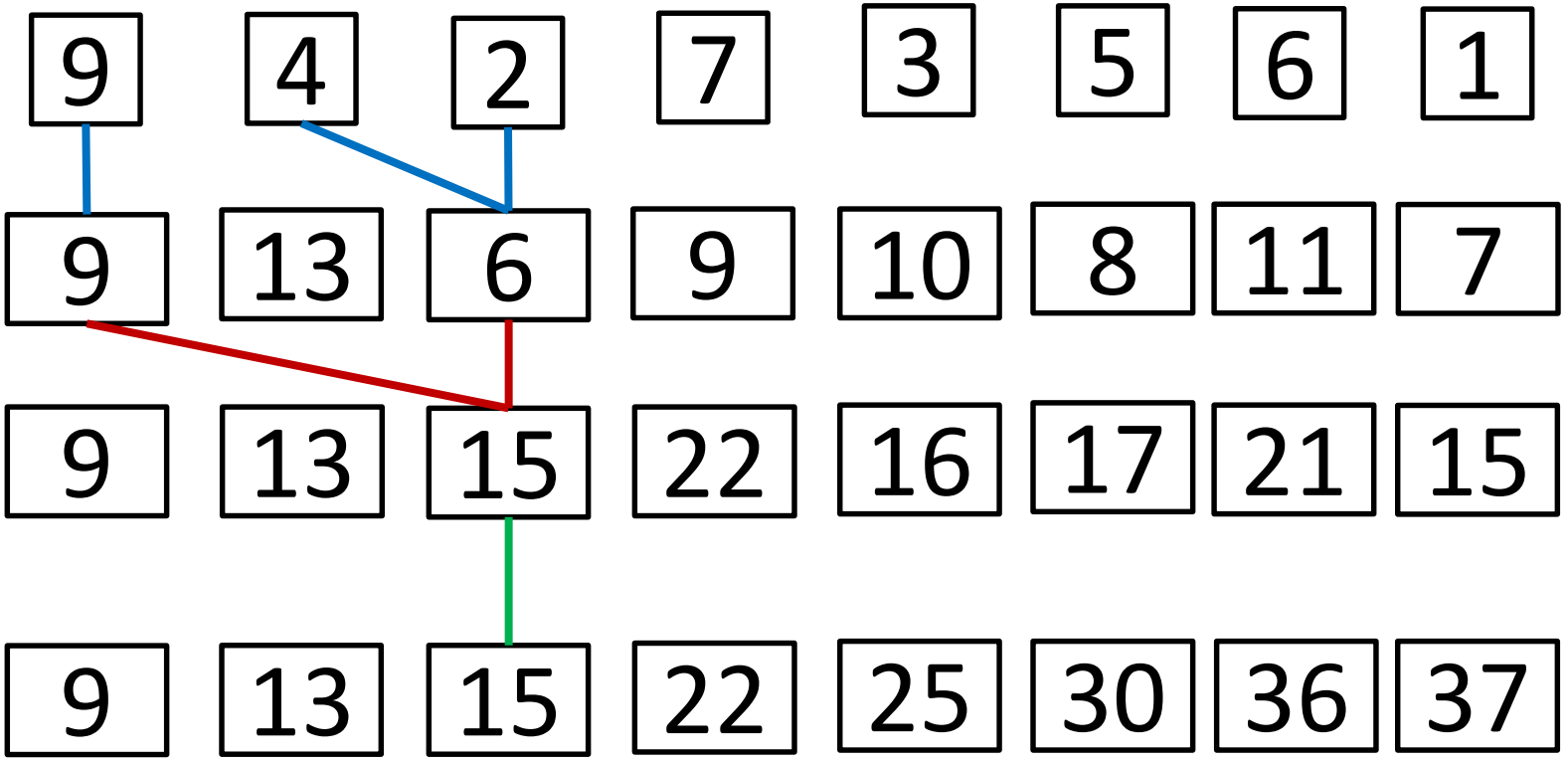
Scan – How it works?



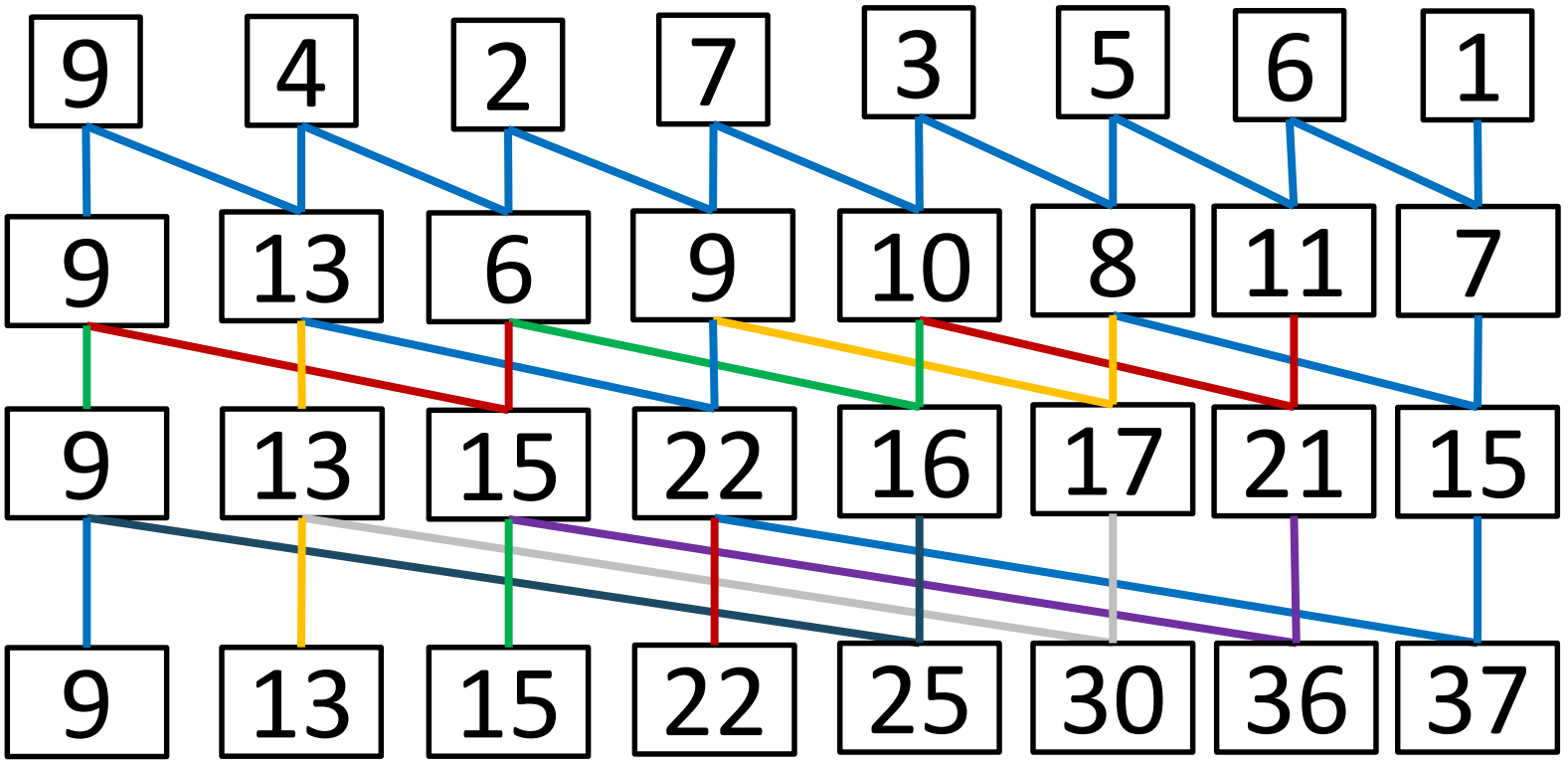
Scan – How it works?



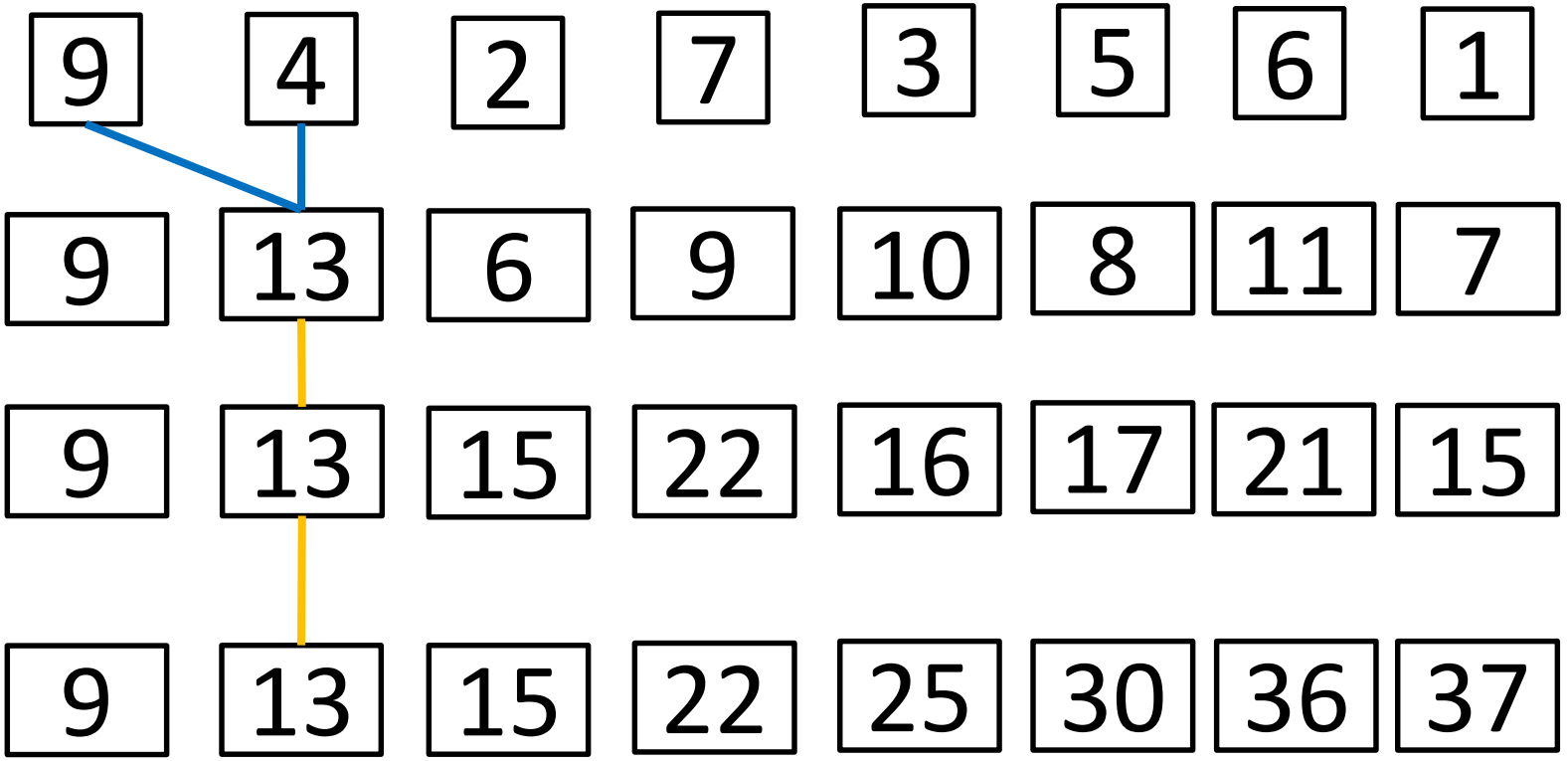
Scan – How it works?



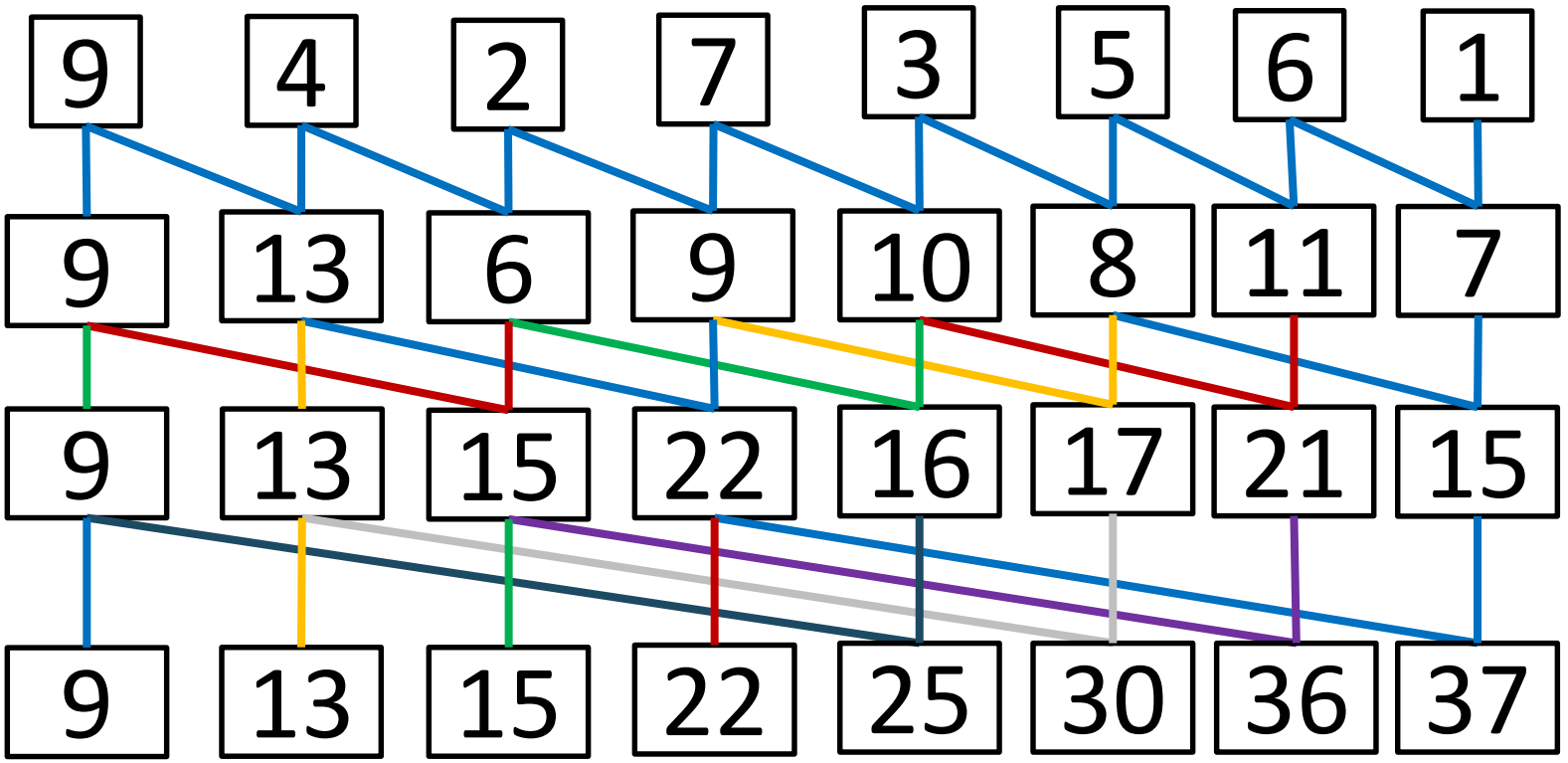
Scan – How it works?



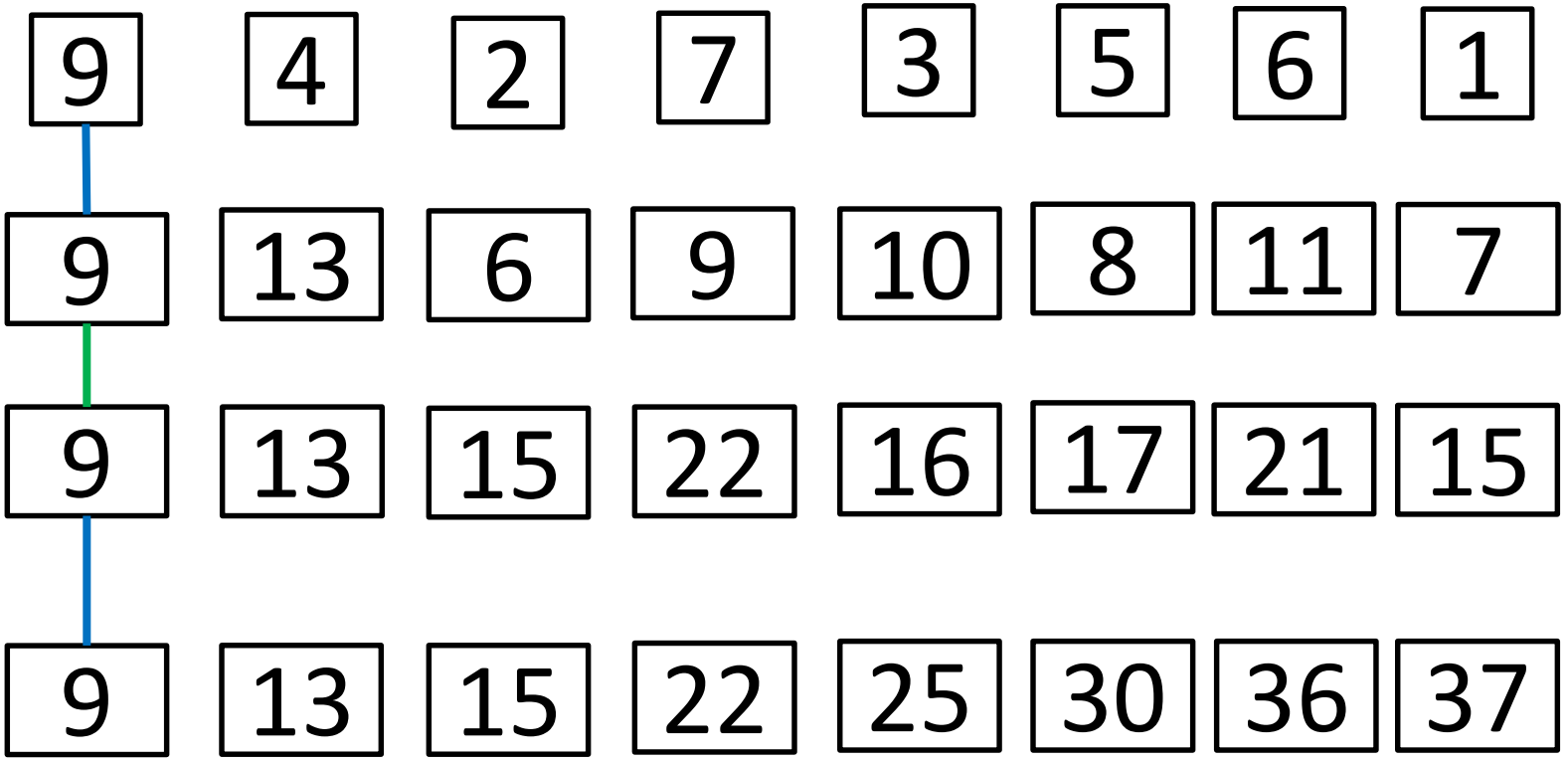
Scan – How it works?



Scan – How it works?



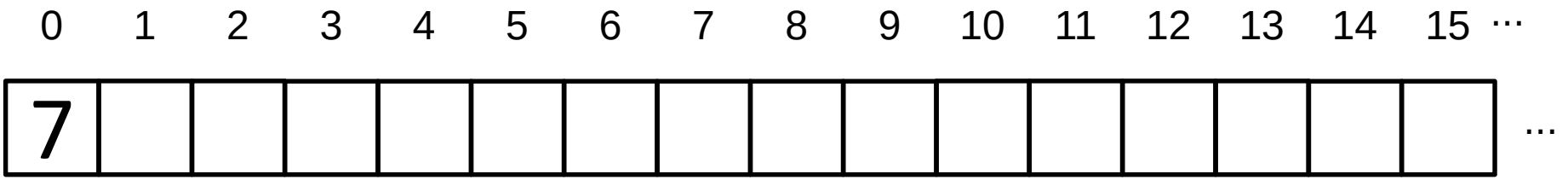
Scan – How it works?




POLITEHNICA
UNIVERSITATEA
BUCUREŞTI
UPB
1818

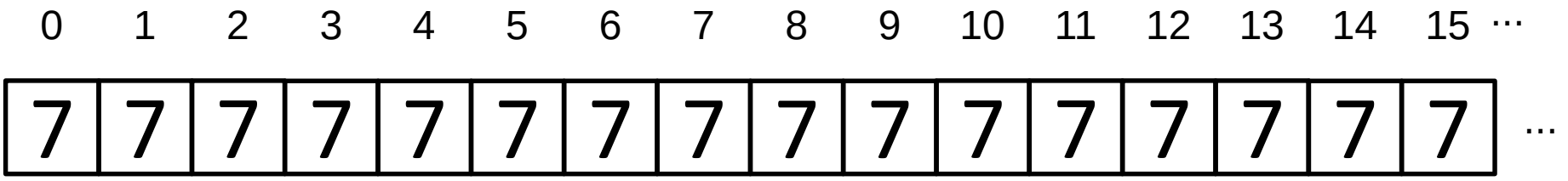
Value Broadcast

Start



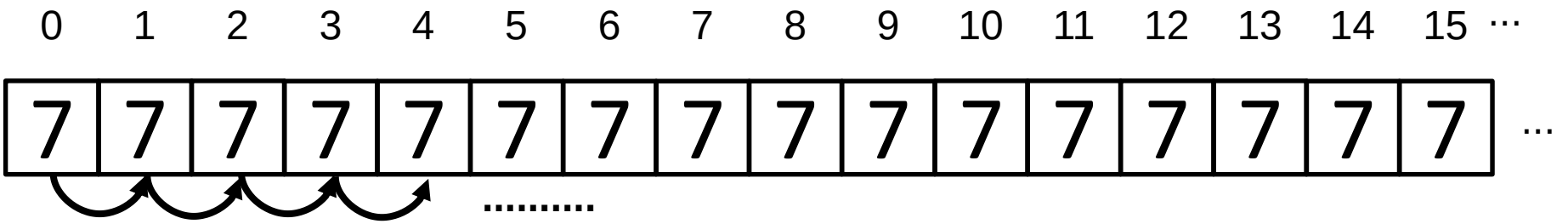
Value Broadcast

End



Inefficient Value Broadcast

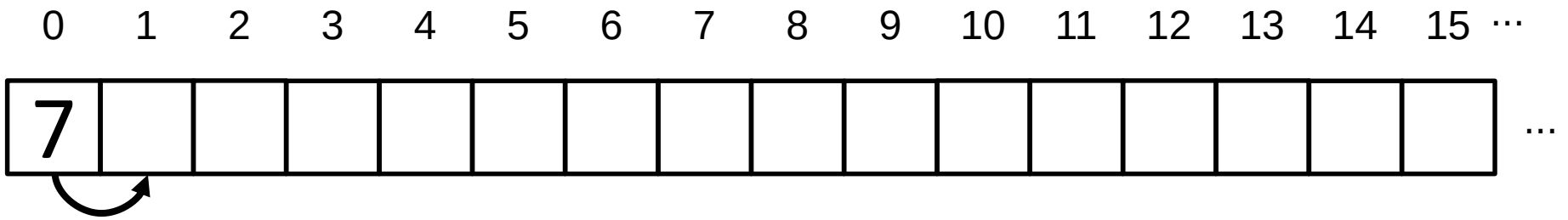
O(n) time



Efficient Value Broadcast

Every element that has the value
copies it to its current position + i

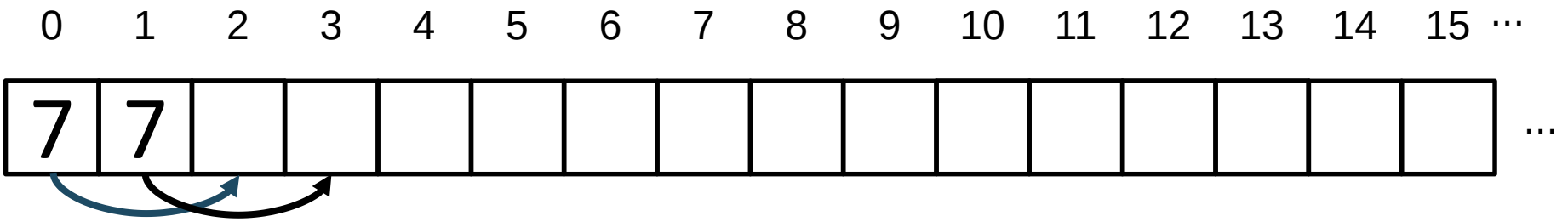
i = 1



Efficient Value Broadcast

Every element that has the value
copies it to its current position + i

i = 2

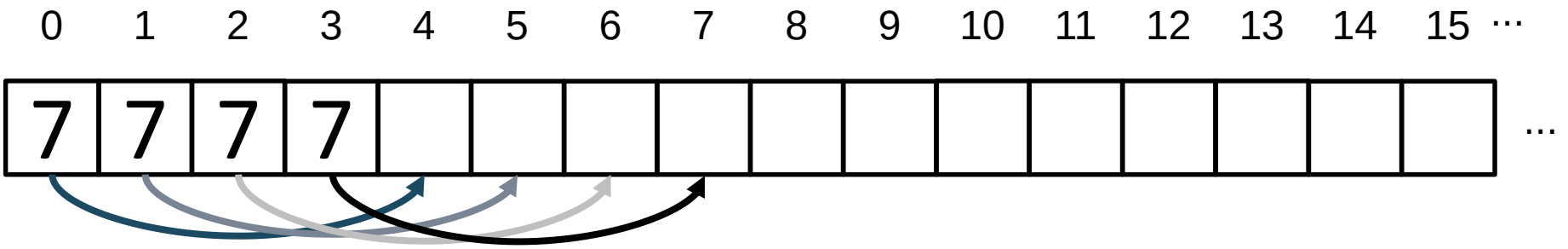


These operations can be executed in parallel

Efficient Value Broadcast

Every element that has the value
copies it to its current position + i

i = 4

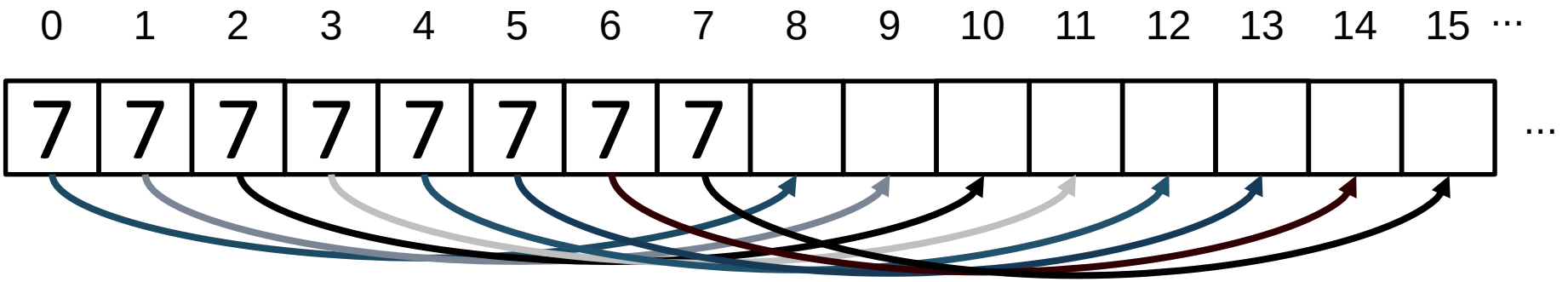


These operations can be executed in parallel

Efficient Value Broadcast

Every element that has the value
copies it to its current position + i

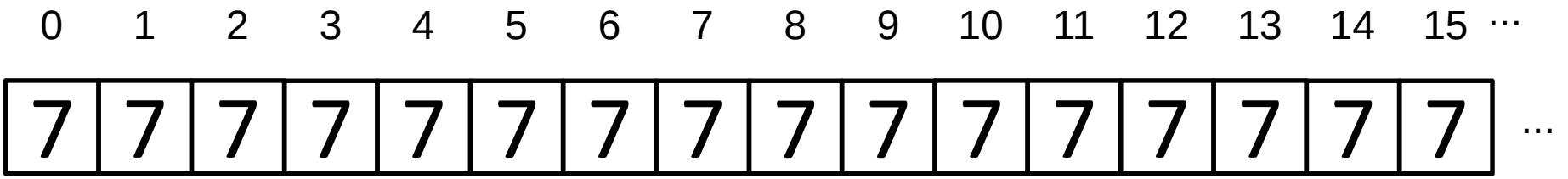
i = 8



These operations can be executed in parallel

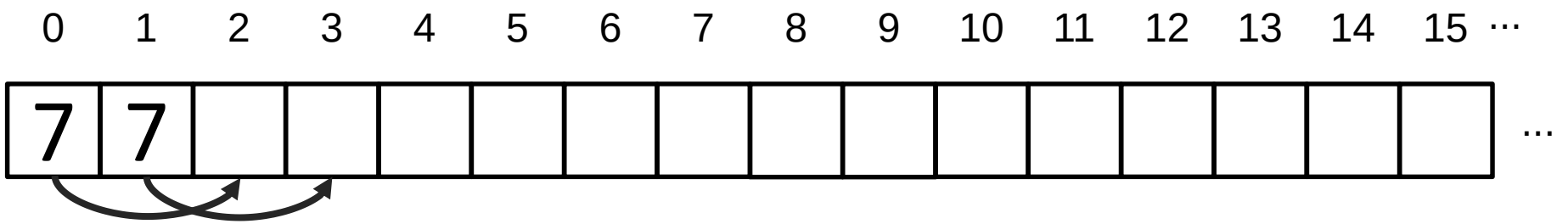
Efficient Value Broadcast

O(log2(n)) time with p = n/2 processors



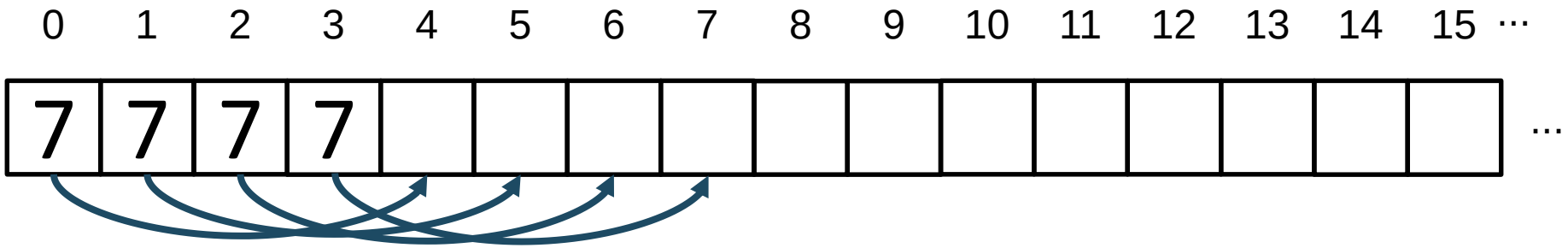
Efficient Value Broadcast

i = 2



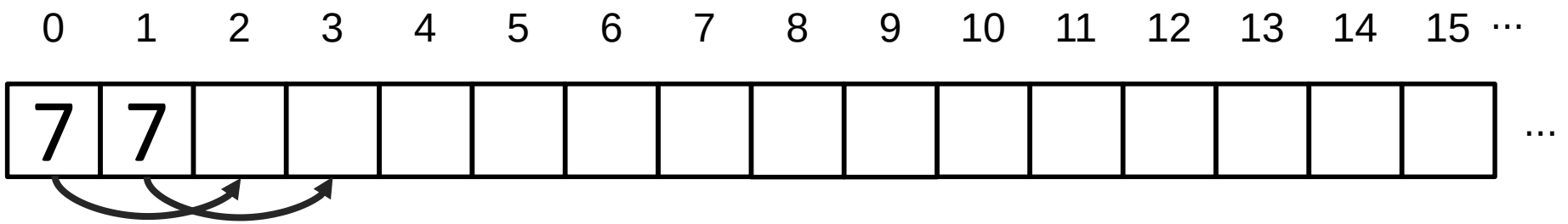
These operations can**NOT** be executed in parallel

i = 4

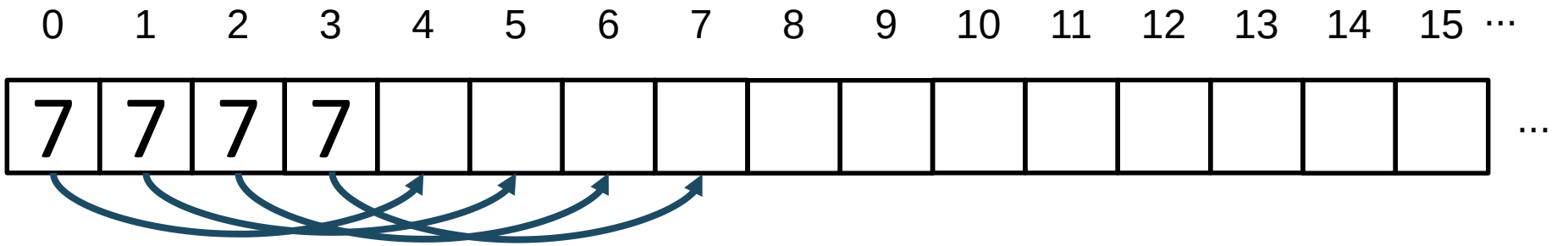


Efficient Value Broadcast

i = 2



i = 4




POLITEHNICA
UNIVERSITATEA
BUCUREŞTI
UPB
1818