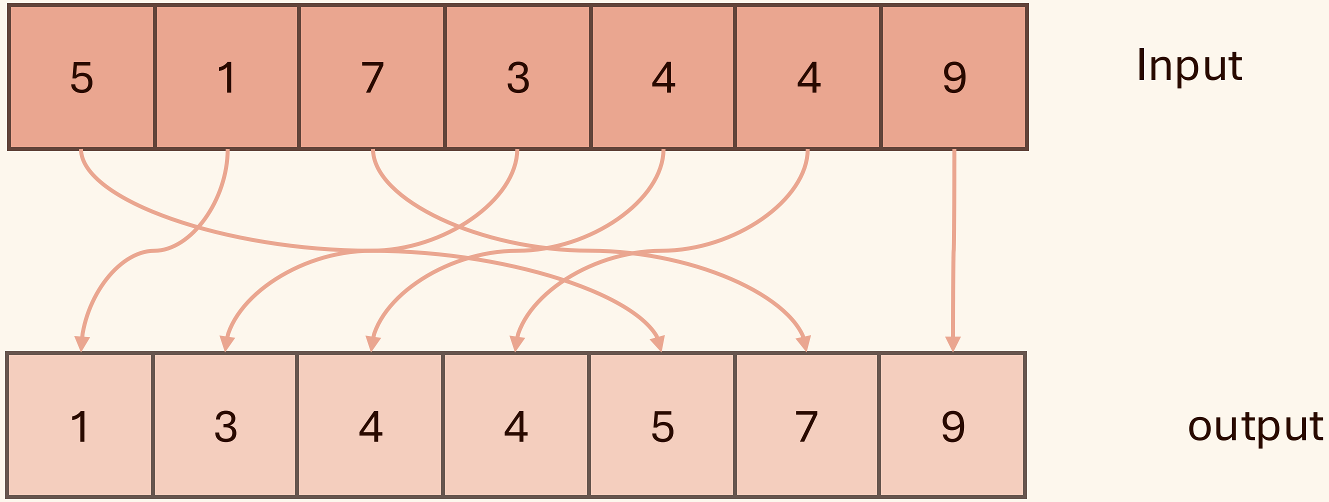
O(1) a beautiful sorting

A constant runtime sorting algorithm

Sorting



Enumeration Sort

- Calculate 'rank' of each element

	5	1	7	3	4	4	9
5		5	**7**	5	5	5	**9**
1	**5**		**7**	**3**	**4**	**4**	**9**
7	7	7		7	7	7	**9**
3	**5**	3	**7**		**4**	**4**	**9**
4	**5**	4	**7**	4		4	**9**
4	**5**	4	**7**	4	4		**9**
9	9	9	9	9	9	9	
Rank	4	0	5	1	2	2	6

Need a tie-break

How long will this take?

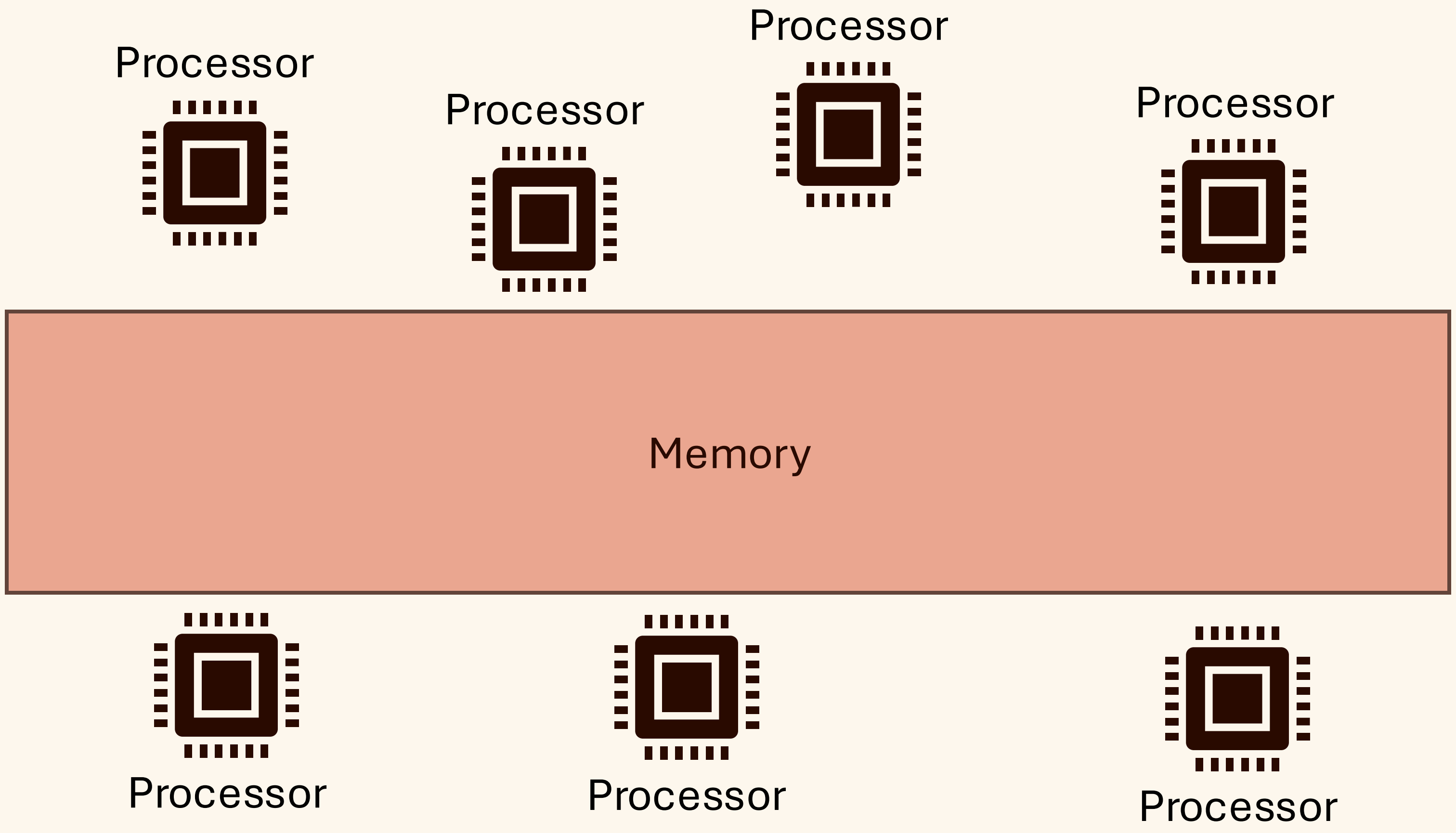
- *Big O Notation*
  - Gives you a rough idea of how different algorithms compare
- For each element, compare to each element
  - N^2 comparisons means $O(N^2)$ runtime

Can we do better?

- Don’t compare each element to **every** other element
- A few different options:
  - Merge sort
  - Quicksort
  - Heapsort
- For each of N elements, log$N$ comparisons
  - O(N log$N$)
  - 18 comparisons compared to 42

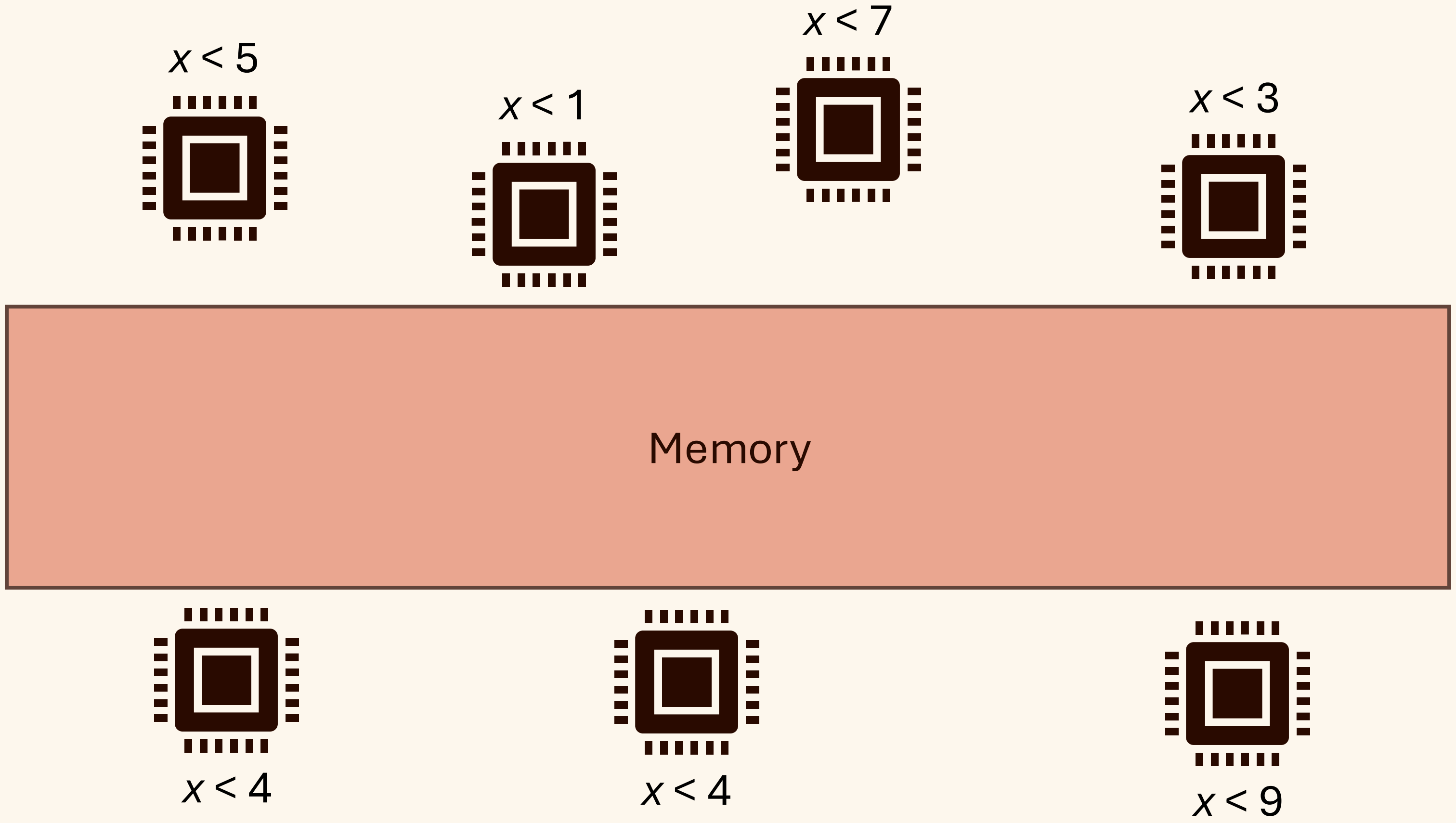
Parallelism

- Consider simplest parallel computer



Parallel Enumeration

- Each processor computes one rank



Parallel Enumeration

- Each processor computes one rank

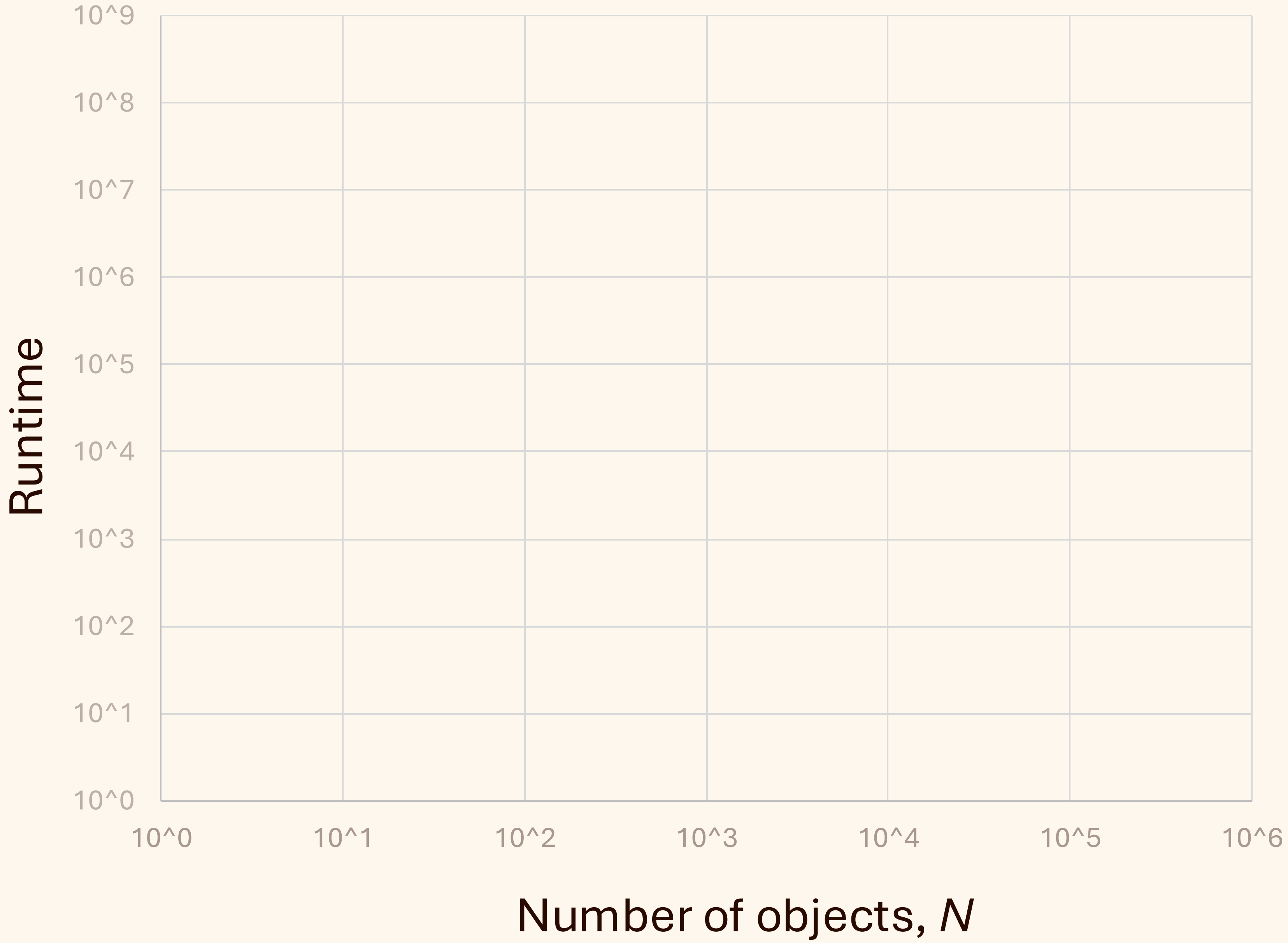
	5	1	7	3	4	4	9
5		5	**7**	5	5	5	**9**
1	**5**		**7**	**3**	**4**	**4**	**9**
7	7	7		7	7	7	**9**
3	**5**	3	**7**		**4**	**4**	**9**
4	**5**	4	**7**	4		4	**9**
4	**5**	4	**7**	4	4		**9**
9	9	9	9	9	9	9	
Rank	4	0	5	1	2	2	6

Parallel Enumeration

- We are doing $N \times N$ comparisons, but N at the same time
- $O(N)$

- Drawback: needs N cores

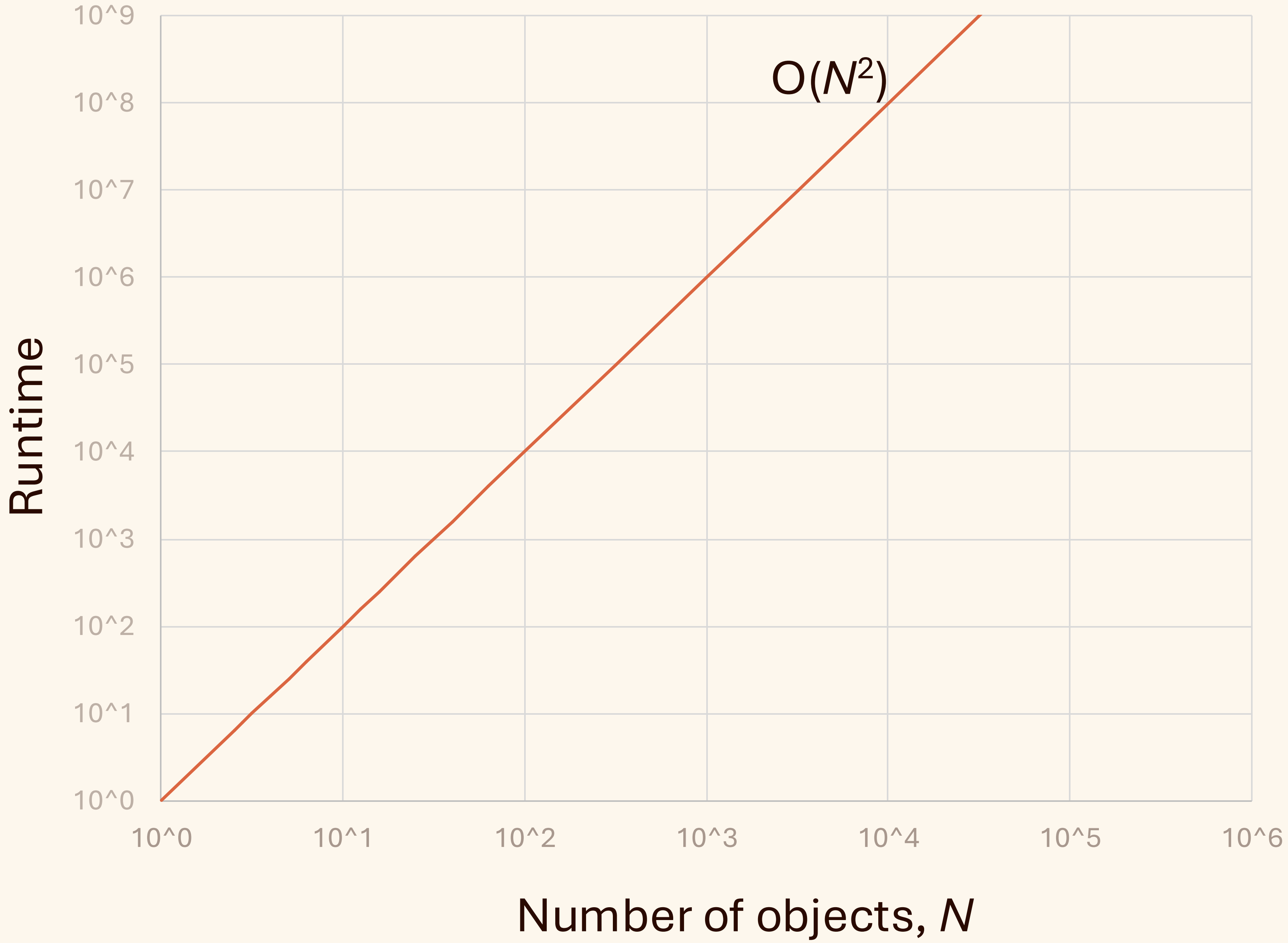
So far



Runtime
10^9
10^8
10^7
10^6
10^5
10^4
10^3
10^2
10^1
10^0
10^0
10^1
10^2
10^3
10^4
10^5
10^6
Number of objects, *N*

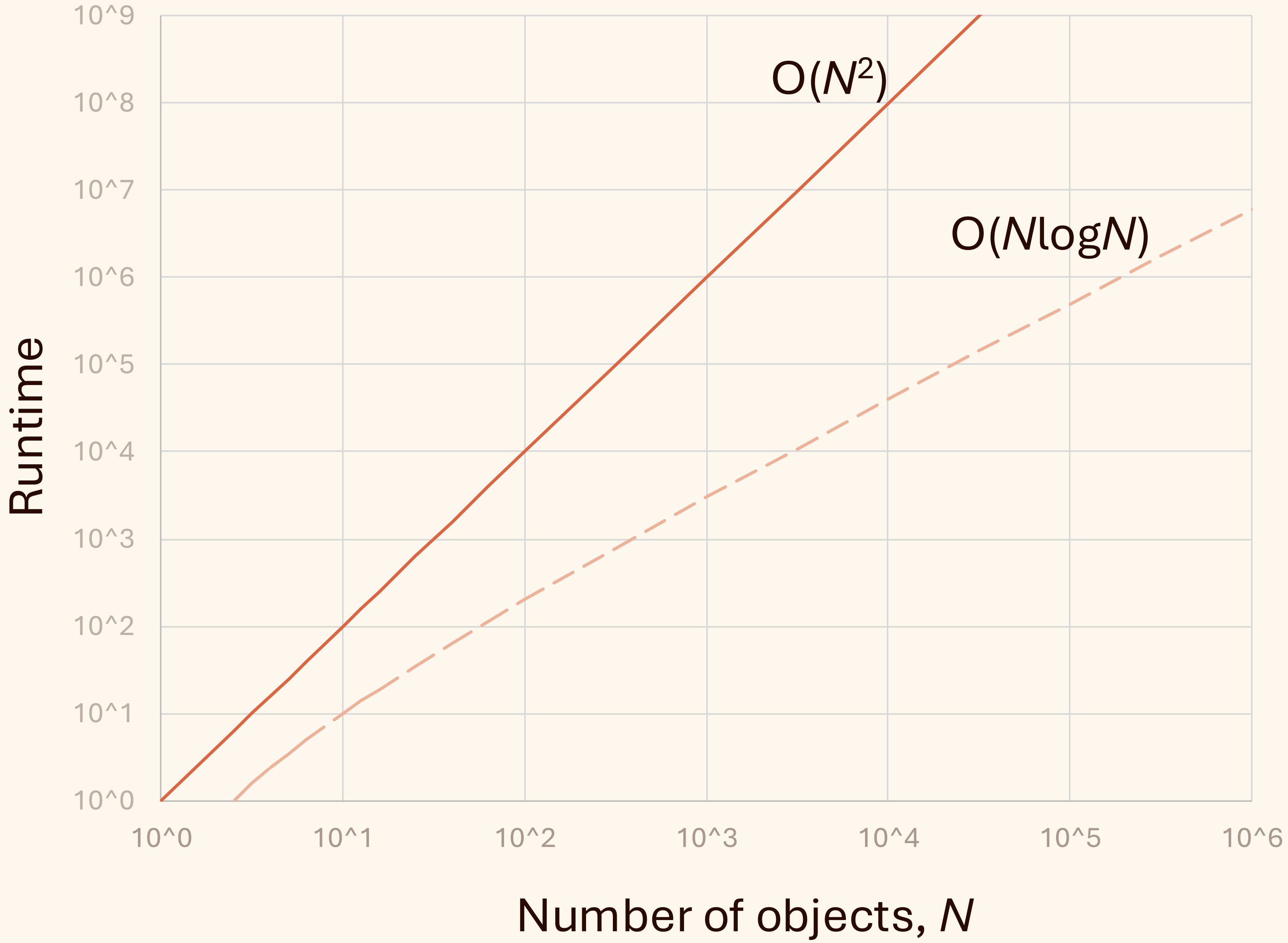
So far

- O(N^2)


O(N2)
Runtime
10^9
10^8
10^7
10^6
10^5
10^4
10^3
10^2
10^1
10^0
10^0
10^1
10^2
10^3
10^4
10^5
10^6
Number of objects, N

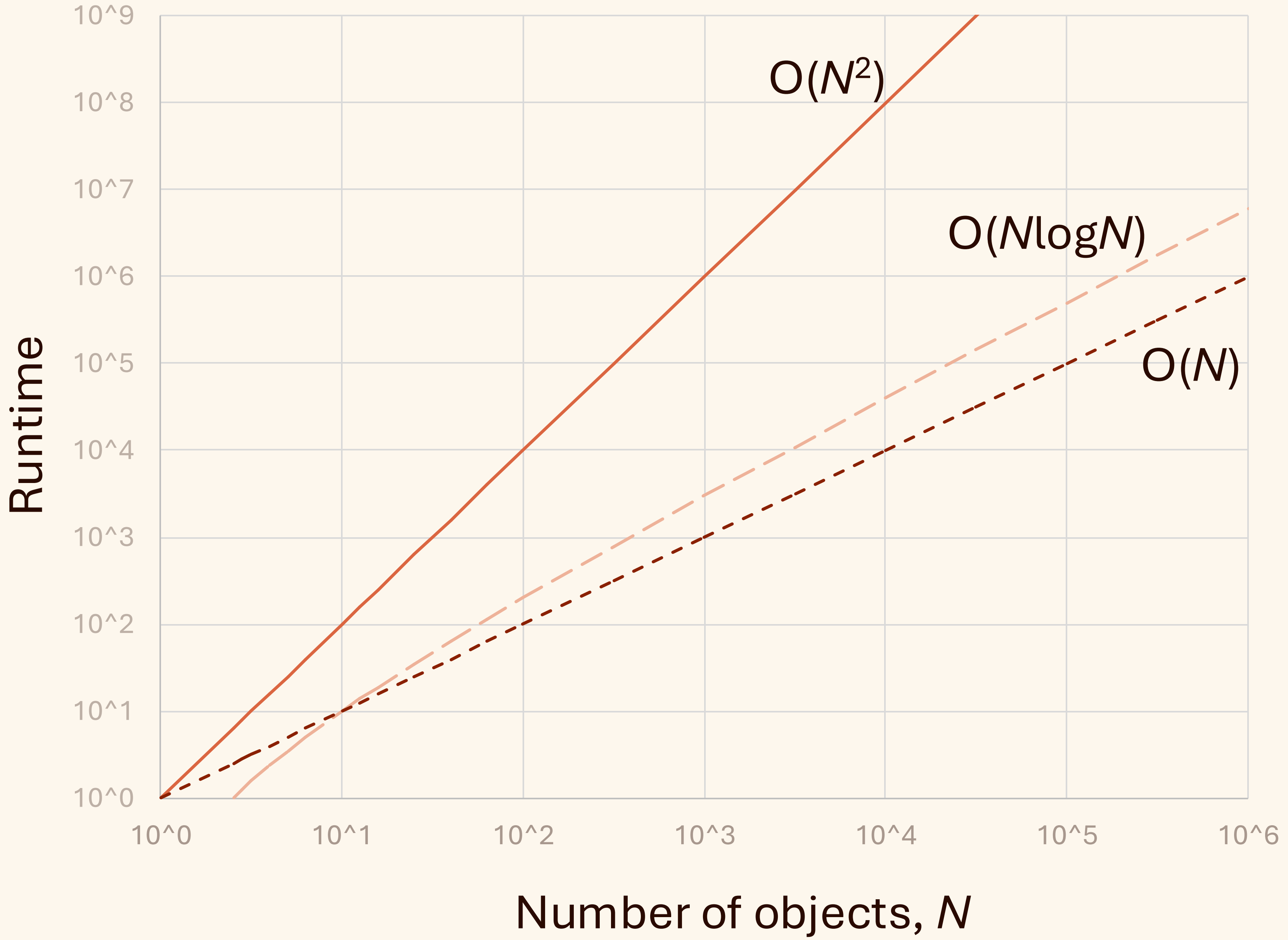
So far

- O(N^2)
- O(Nlog$N$)



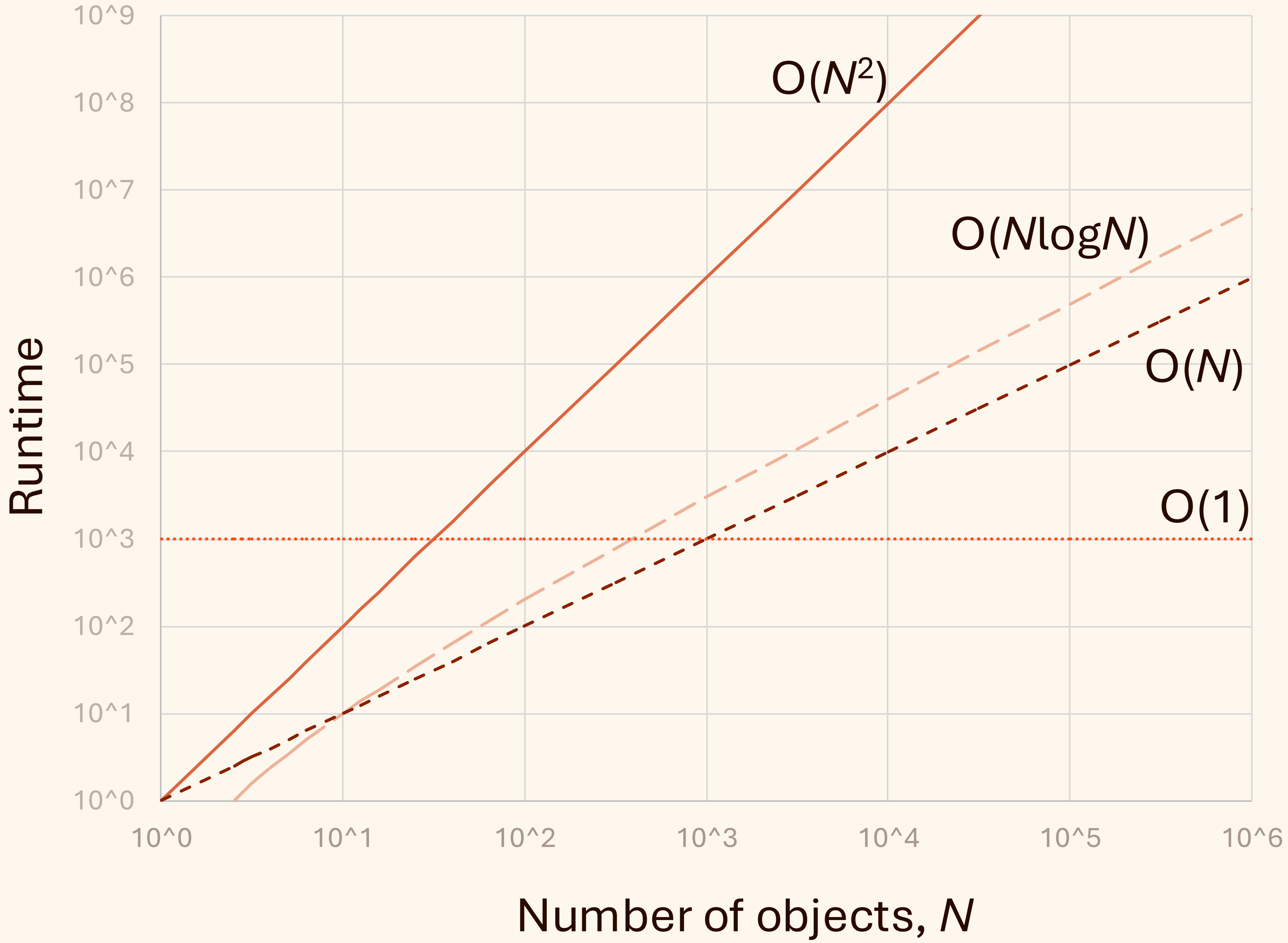
So far

- O(N^2)
- O(Nlog$N$)
- O(N)



So far

- O(N^2)
- O($N\log N$)
- O(N)
- O(1)?



Constant runtime?

- A processor for each rank

	5	1	7	3	4	4	9
5		5	**7**	5	5	5	**9**
1	**5**		**7**	**3**	**4**	**4**	**9**
7	7	7		7	7	7	**9**
3	**5**	3	**7**		**4**	**4**	**9**
4	**5**	4	**7**	4		4	**9**
4	**5**	4	**7**	4	4		**9**
9	9	9	9	9	9	9	
Rank	4	0	5	1	2	2	6

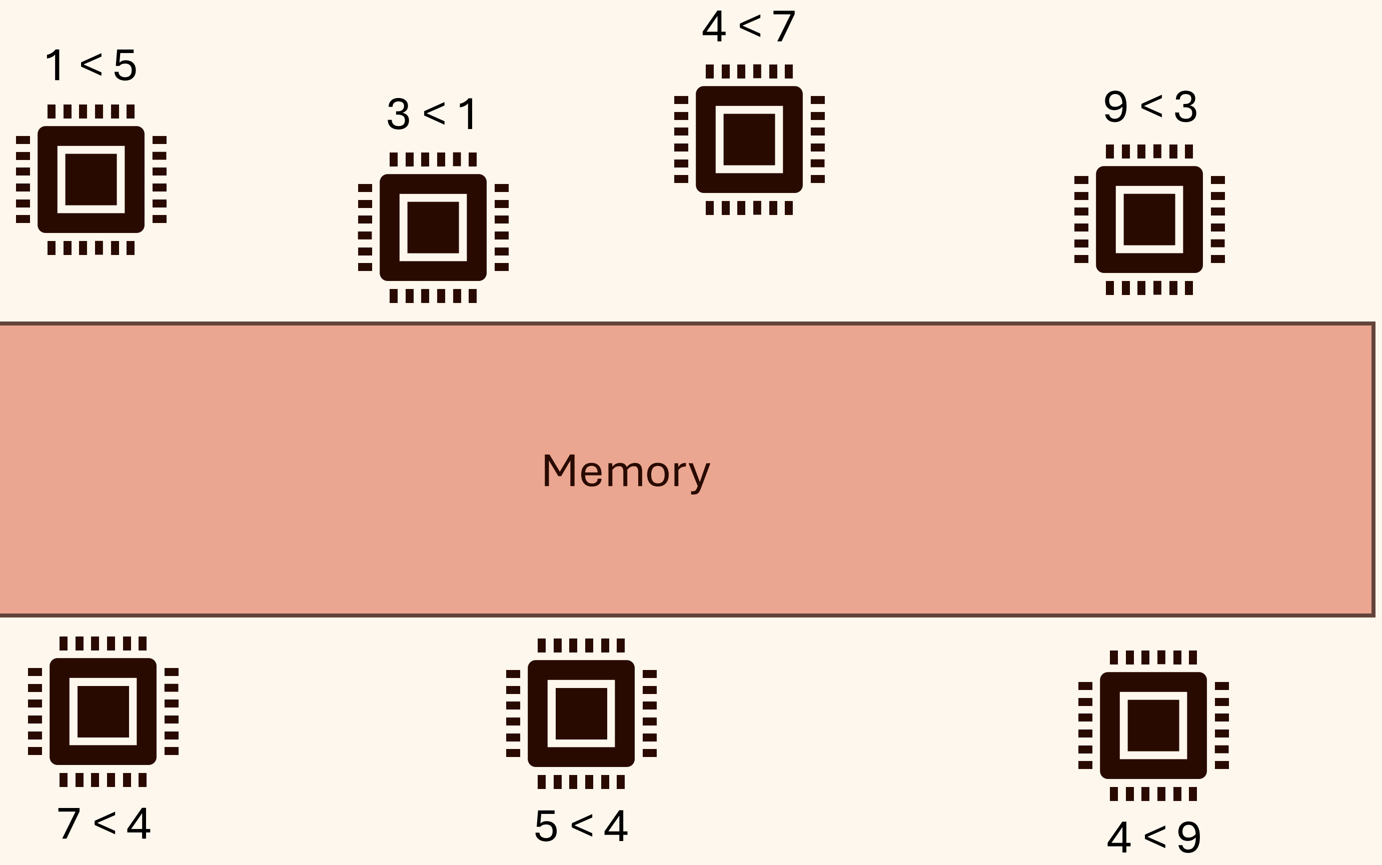
Constant runtime?

- A processor for each ~~rank~~ *comparison*

	5	1	7	3	4	4	9
5		5	**7**	5	5	5	**9**
1	**5**		**7**	**3**	**4**	**4**	**9**
7	7	7		7	7	7	**9**
3	**5**	3	**7**		**4**	**4**	**9**
4	**5**	4	**7**	4		4	**9**
4	**5**	4	**7**	4	4		**9**
9	9	9	9	9	9	9	
Rank	4	0	5	1	2	2	6

Parallel Enumeration

- Each processor computes one comparison



Constant runtime!

- You have $N \times N$ comparisons, but you do N^2 at the same time
- O(1) runtime!
- Caveats:
  - Ignores “message passing costs”
  - Needs N^2 processes

Question Time

Roughly how many numbers could we sort with our parallel algorithm, using every core of cuillin’s worker nodes?

a) 10
b) 100
c) 1000

Question Time

Roughly how many numbers could we sort with our parallel algorithm, using every core of cuillin's worker nodes?

a) **10 – 1692 cores means 41 numbers**

b) 100

c) 1000

Conclusion

- You can make enough assumptions and pick any metric to make something sound better than it is.
- Inherent assumptions: Enough cores, perfect system, …
- If something sounds too good to be true, *it is*.