CORK INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

Big Data Processing

L03-05: Distributed Programming

Dr. Ignacio Castineiras
Department of Computer Science

Outline

1. Sequential vs. Concurrent Execution.
2. List Inversions: A Real-world Example.
3. Scalability: Computational Complexity Barrier.
4. Concurrency: Infrastructure-based Approach.
5. Can We Do Better? An Algorithm-based Approach.

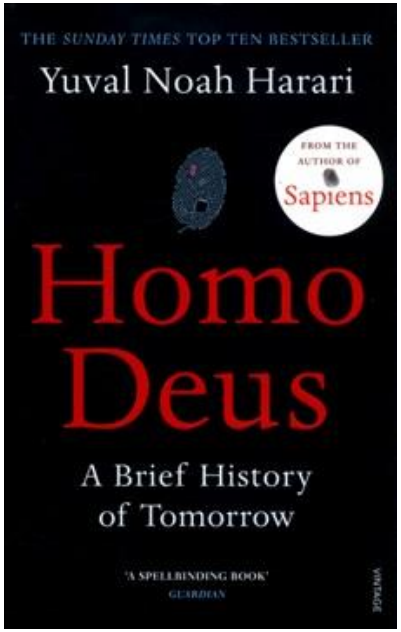
Outline

1. Sequential vs. Concurrent Execution.
2. List Inversions: A Real-world Example.
3. Scalability: Computational Complexity Barrier.
4. Concurrency: Infrastructure-based Approach.
5. Can We Do Better? An Algorithm-based Approach.

Sequential vs. Concurrent Execution

- In the last slide of L02 we quoted a paragraph from the book **Homo Deus, *Yuval Noah Harari*, Harvill Secker London, 2015**

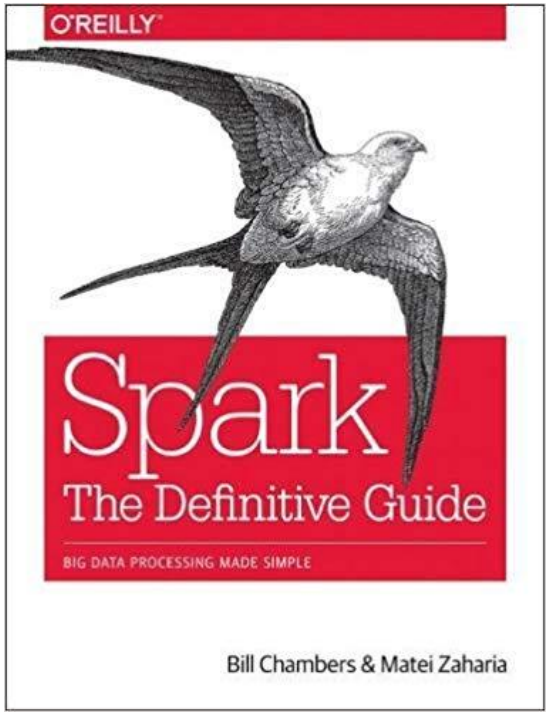
  as a good way to reflect about the future that Artificial Intelligence and Biotechnology will bring us.



Sequential vs. Concurrent Execution

- Now we start this lecture by quoting some paragraphs from the book:
  Spark - The Definite Guide, *Bill Chambers and Matei Zaharia*, O'Reilly, 2018.

  as a good way to reflect on the use of distributed programming in the context of big data.

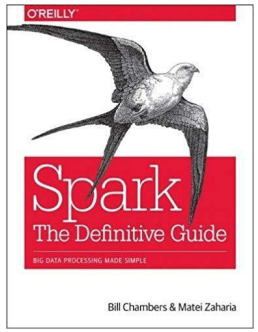


Sequential vs. Concurrent Execution

- Matei Zaharia, assistant professor in Computer Science at Stanford and Chief Technologist Officer at Databricks. He started the Spark project at UC Berkeley in 2009.
- Bill Chambers, product manager at Databricks.

➢ ***Context: The Big Data Problem***

➢ *Why do we need a new engine and programming model for data analytics in the first place? As with many trends in computing, this is due to changes in the economic factors that underlie computer applications and hardware.*

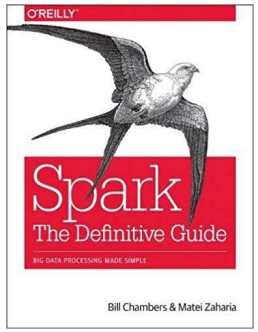


Sequential vs. Concurrent Execution

- Matei Zaharia, assistant professor in Computer Science at Stanford and Chief Technologist Officer at Databricks. He started the Spark project at UC Berkeley in 2009.
- Bill Chambers, product manager at Databricks.

➢ ***Context: The Big Data Problem***

➢ *For most of their history, computers became faster every year through processor speed increases: the new processors each year could run more instructions per second than the previous year's. As a result, applications also automatically became faster every year, without any changes needed to their code. This trend led to a large and established ecosystem of applications building up over time, most of which were designed to run only on a single processor. These applications rode the trend of improved processor speeds to scale up to larger computations and larger volumes of data over time.*

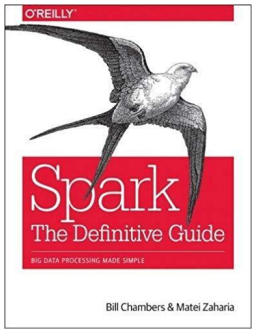


Sequential vs. Concurrent Execution

- Matei Zaharia, assistant professor in Computer Science at Stanford and Chief Technologist Officer at Databricks. He started the Spark project at UC Berkeley in 2009.
- Bill Chambers, product manager at Databricks.

➢ ***Context: The Big Data Problem***

➢ *Unfortunately, this trend in hardware stopped around 2005: due to hard limits in heat dissipation, hardware developers stopped making individual processors faster, and switched toward adding more parallel CPU cores all running at the same speed. This change meant that suddenly applications needed to be modified to add parallelism in order to run faster, which set the stage for new programming models such as Apache Spark.*

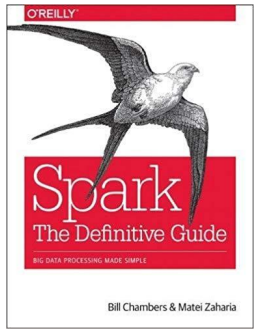


Sequential vs. Concurrent Execution

- Matei Zaharia, assistant professor in Computer Science at Stanford and Chief Technologist Officer at Databricks. He started the Spark project at UC Berkeley in 2009.
- Bill Chambers, product manager at Databricks.

➢ ***Context: The Big Data Problem***

➢ *On top of that, the technologies for storing and collecting data did not slow down appreciably in 2005, when processor speeds did. The cost to store 1 TB of data continues to drop by roughly two times every 14 months, meaning that it is very inexpensive for organizations of all sizes to store large amounts of data. Moreover, many of the technologies for collecting data (sensors, cameras, public datasets, etc.) continue to drop in cost and improve in resolution.*

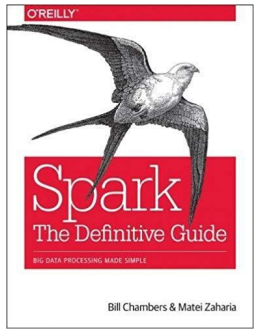


Sequential vs. Concurrent Execution

- Matei Zaharia, assistant professor in Computer Science at Stanford and Chief Technologist Officer at Databricks. He started the Spark project at UC Berkeley in 2009.
- Bill Chambers, product manager at Databricks.

➢ ***Context: The Big Data Problem***

➢ *The end result is a world in which collecting data is extremely inexpensive—many organizations today even consider it negligent not to log data of possible relevance to the business—but processing it requires large, parallel computations, often on clusters of machines. Moreover, in this new world, the software developed in the past 50 years cannot automatically scale up, and neither can the traditional programming models for data processing applications, creating the need for new programming models.*
It is this world that Apache Spark was built for.



Outline

1. Sequential vs. Concurrent Execution.
2. List Inversions: A Real-world Example.
3. Scalability: Computational Complexity Barrier.
4. Concurrency: Infrastructure-based Approach.
5. Can We Do Better? An Algorithm-based Approach.

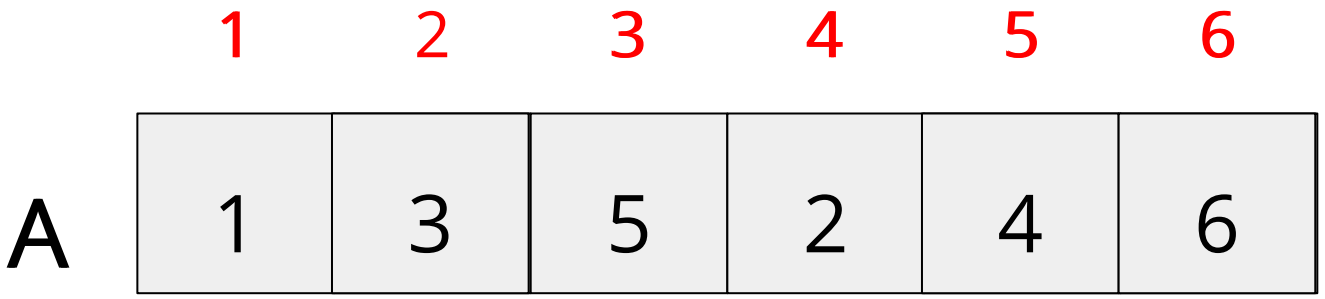
List Inversions: A Real-world Example

Problem Definition:

Given an array A of size *n* containing a permutation of the numbers 1..n,
we define the number of inversions in A as the amount of pairs (i,j) such that
i < j and A[i] > A[j]

List Inversions: A Real-world Example

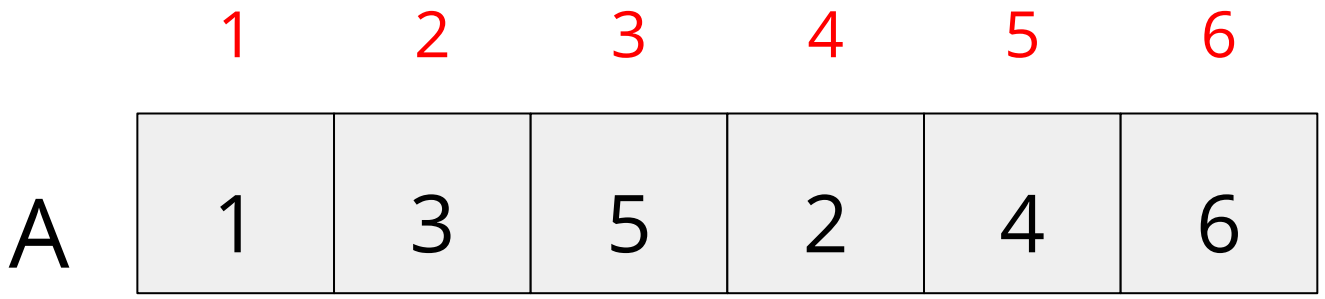
Example:



- The array A is of size 6, and thus contains a permutation of the numbers 1..6

List Inversions: A Real-world Example

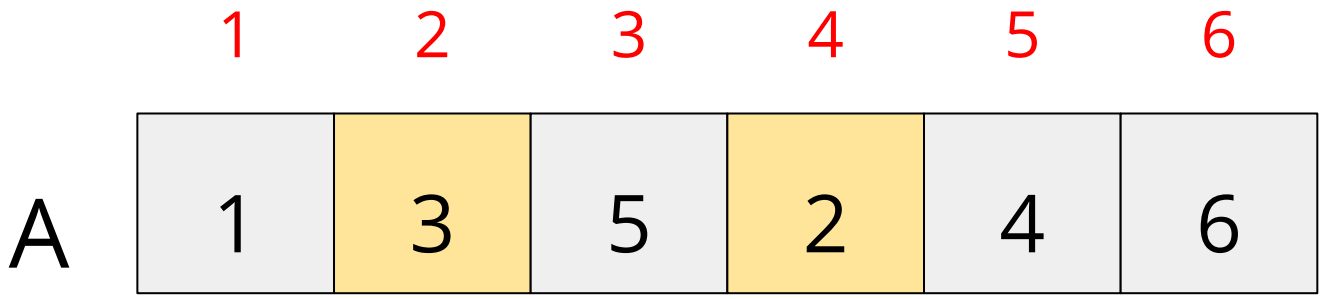
Example:



- The array A is of size 6, and thus contains a permutation of the numbers 1..6
- The number of inversions of this array is **3**

List Inversions: A Real-world Example

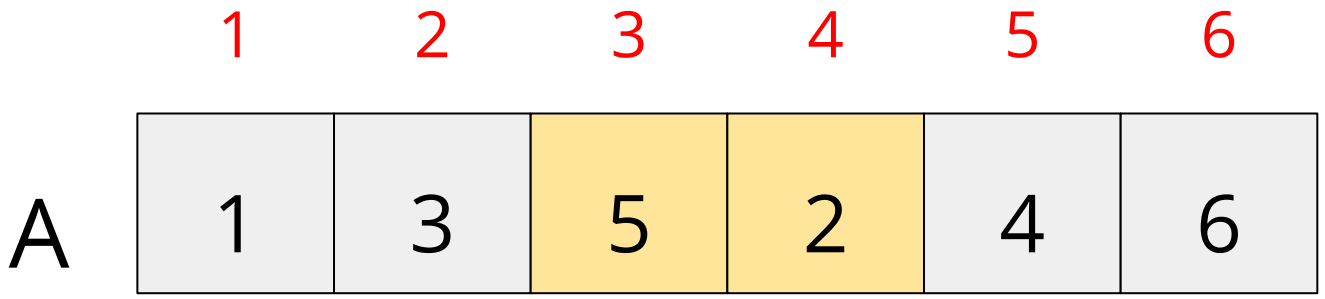
Example:



- The array A is of size 6, and thus contains a permutation of the numbers 1..6
- The number of inversions of this array is **3**
  - i = 2, j = 4. A[i] = 3, A[j] = 2. Thus i < j but A[i] > A[j]

List Inversions: A Real-world Example

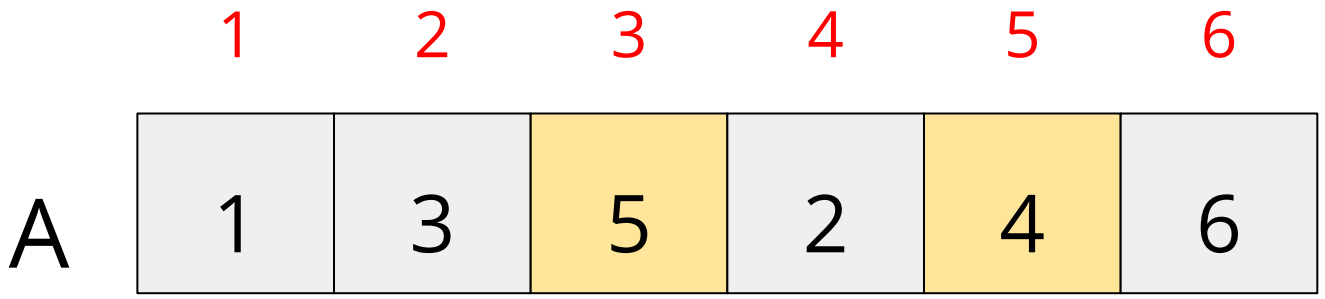
Example:



- The array A is of size 6, and thus contains a permutation of the numbers 1..6
- The number of inversions of this array is **3**
  - i = 2, j = 4. A[i] = 3, A[j] = 2. Thus i < j but A[i] > A[j]
  - i = 3, j = 4. A[i] = 5, A[j] = 2.

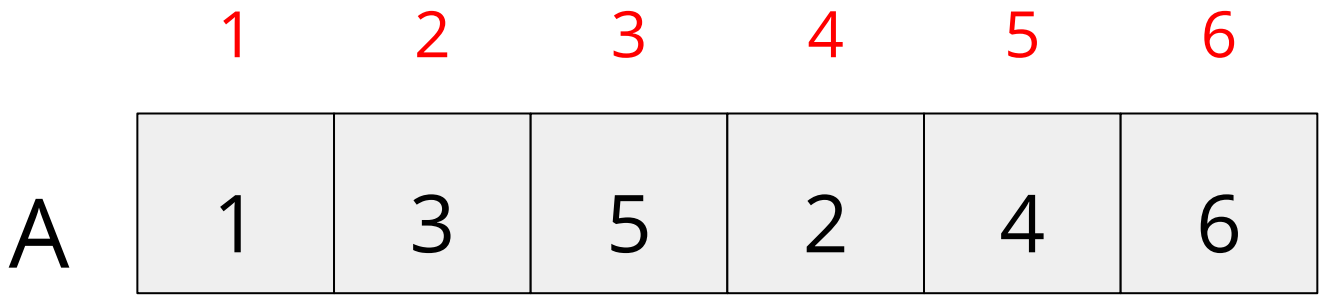
List Inversions: A Real-world Example

Example:



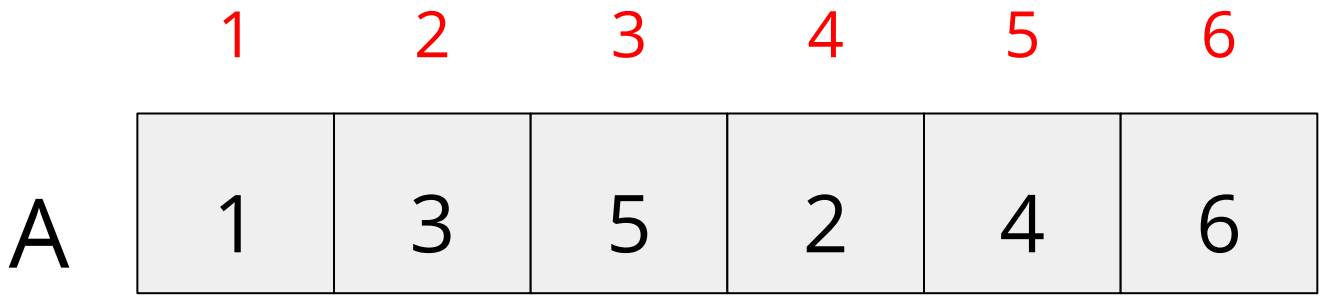
- The array A is of size 6, and thus contains a permutation of the numbers 1..6
- The number of inversions of this array is **3**
  - i = 2, j = 4. A[i] = 3, A[j] = 2. Thus i < j but A[i] > A[j]
  - i = 3, j = 4. A[i] = 5, A[j] = 2.
  - i = 3, j = 5. A[i] = 5, A[j] = 4

List Inversions: A Real-world Example



- This simple problem is be very interesting in the context of Artificial Intelligence, more specifically in the context of Recommender Systems.

List Inversions: A Real-world Example

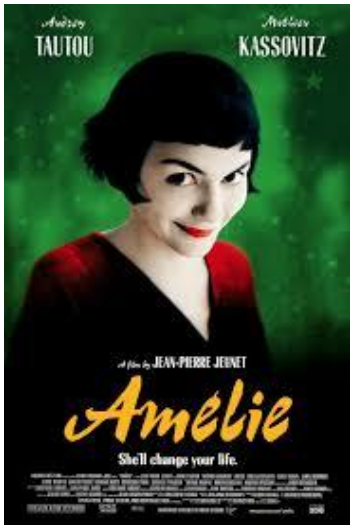


- This simple problem is be very interesting in the context of Artificial Intelligence, more specifically in the context of Recommender Systems.
- It provides a way of measuring the difference (in terms of their preferences) between two people.

List Inversions: A Real-world Example

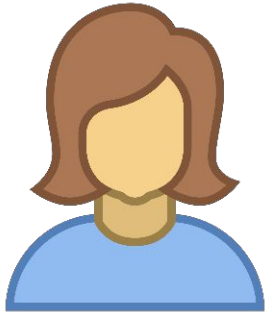
Imagine we are asked to rank
a number of movies.

List Inversions: A Real-world Example



List Inversions: A Real-world Example

- **Person P1 ranked them as follows:**

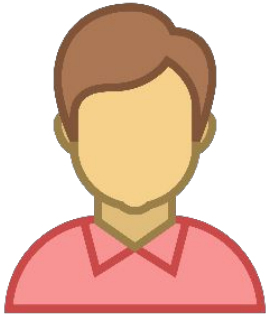


1 2 3 4 5 6

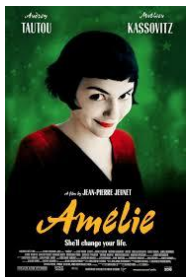
- *My prefered movie is Amelie.*
- *Then Almost Famous.*
- *Followed by Silver Lining Playbooks.*
- *Wonder would be next.*
- *Inception probably the next one.*
- *And Batman last.*

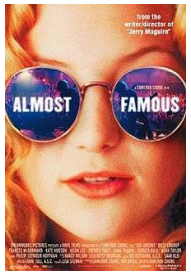
List Inversions: A Real-world Example

- **Person P2 ranked them as follows:**



1 2 3 4 5 6



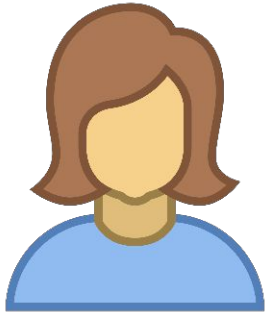


- *My prefered movie is Amelie.*
- *Then Wonder.*
- *Followed by Almost Famous.*
- *Inception would be next.*
- *Silver Lining Playbook probably the next one.*
- *And Batman last.*

List Inversions: A Real-world Example

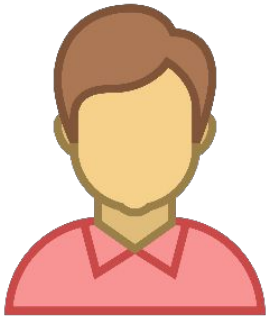
In this context we can use the array A of list inversions to compare the preferences of P1 and P2.

P1



1 2 3 4 5 6

P2



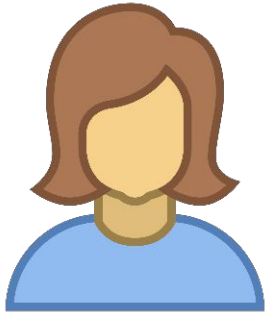
1 2 3 4 5 6

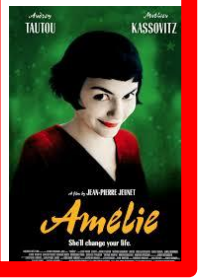
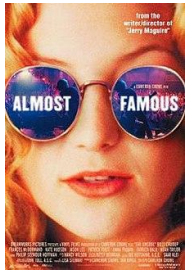
List Inversions: A Real-world Example

The movie P1 ranks 1st is Amelie.

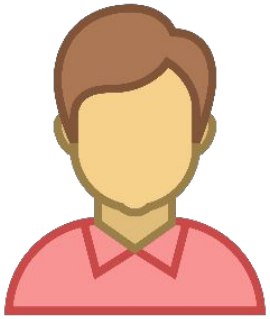
P2 ranks this movie in position 1 → **A[1] = 1**

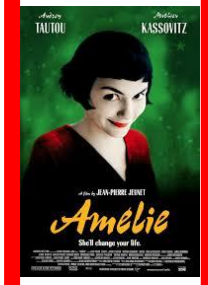
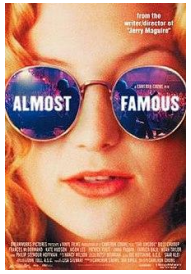
P1



1	2	3	4	5	6
					

P2

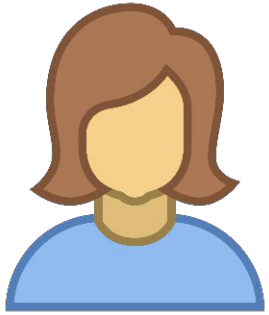


1	2	3	4	5	6
					

List Inversions: A Real-world Example

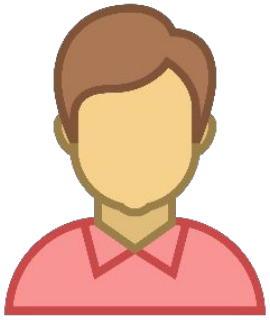
The movie P1 ranks 2nd is Almost Famous.
P2 ranks this movie in position 3 → **A[2] = 3**

P1



1 2 3 4 5 6

P2

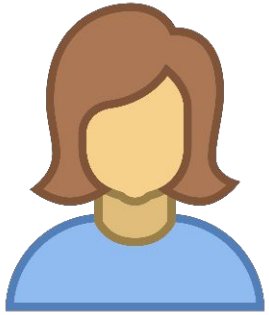


1 2 3 4 5 6

List Inversions: A Real-world Example

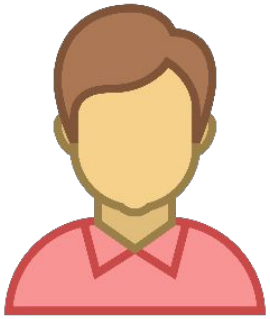
The movie P1 ranks 3rd is Silver Lining Playbook.
P2 ranks this movie in position 5 → **A[3] = 5**

P1



1 2 3 4 5 6

P2



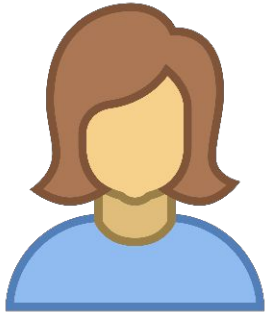
1 2 3 4 5 6

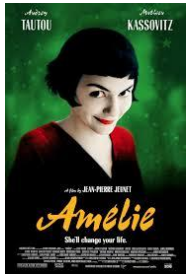
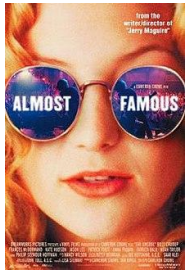
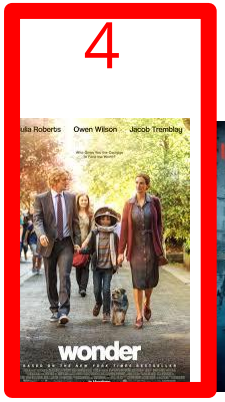
List Inversions: A Real-world Example

The movie P1 ranks 4th is Wonder.

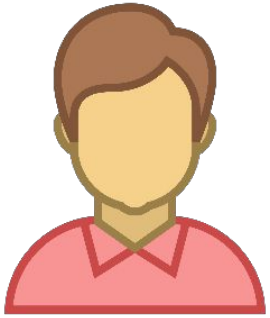
P2 ranks this movie in position 2 → **A[4] = 2**

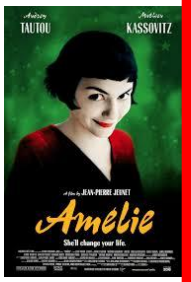
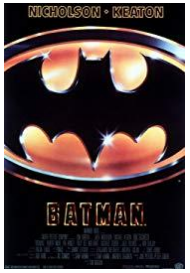
P1



1	2	3	4	5	6
					

P2



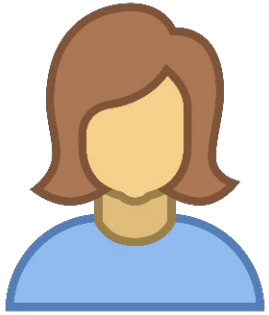
1	2	3	4	5	6
					

List Inversions: A Real-world Example

The movie P1 ranks 5th is Inception.

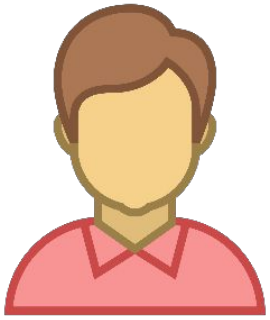
P2 ranks this movie in position 4 → **A[5] = 4**

P1



1 2 3 4 5 6

P2



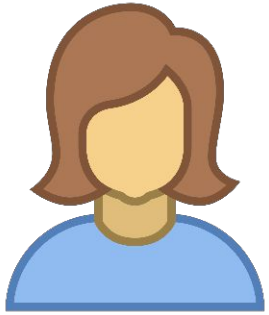
1 2 3 4 5 6

List Inversions: A Real-world Example

The movie P1 ranks 6th is Batman.

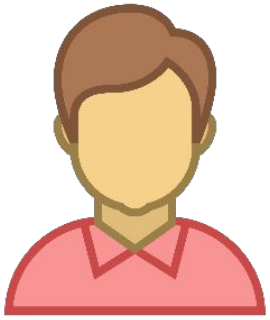
P2 ranks this movie in position 6 → **A[6] = 6**

P1



1 2 3 4 5 6

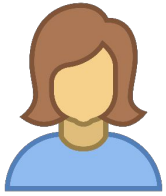
P2



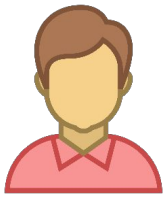
1 2 3 4 5 6

List Inversions: A Real-world Example

P1



P2



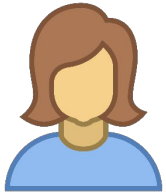
A

1	2	3	4	5	6
1	3	5	2	4	6

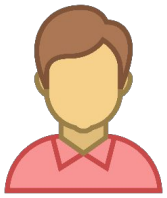
- As you can see, the array A represents the ranks of P2 vs the ranks of P1.

List Inversions: A Real-world Example

P1



P2



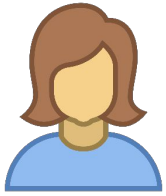
A

1	2	3	4	5	6
1	3	5	2	4	6

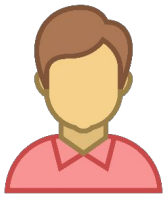
- And computing the list inversions of A is a way of measuring the differences of P2 and P1 in ranking the movies.

List Inversions: A Real-world Example

P1



P2



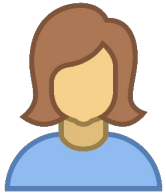
A

1	2	3	4	5	6
1	3	5	2	4	6

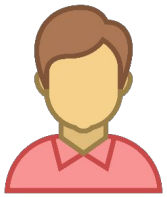
- Interestingly, list inversions does not measure the differences of P1 and P2 based on the concrete ranking they assigned to each single movie.

List Inversions: A Real-world Example

P1



P2



A

1	2	3	4	5	6
1	3	5	2	4	6

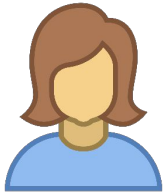
- Interestingly, list inversions does not measure the differences of P1 and P2 based on the concrete ranking they assigned to each single movie.

Batman: Both P1 and P2 rank it equal, in number 6.

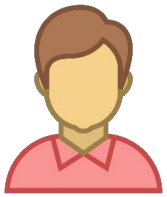
Thus, P1 and P2 are similar.

List Inversions: A Real-world Example

P1



P2



A

1	2	3	4	5	6
1	3	5	2	4	6

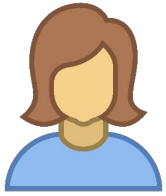
- Interestingly, list inversions does not measure the differences of P1 and P2 based on the concrete ranking they assigned to each single movie.

Almost Famous: P1 ranks it 2nd and P2 ranks it 3rd.

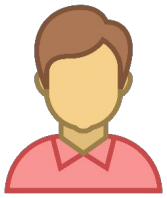
Thus, P1 and P2 are different.

List Inversions: A Real-world Example

P1



P2



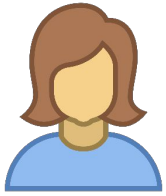
A

1	2	3	4	5	6
1	3	5	2	4	6

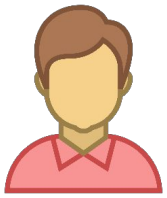
- No, list inversions does not work this way.
  Again, it does not measure the differences of P1 and P2 based on the concrete ranking they assigned to each single movie.

List Inversions: A Real-world Example

P1



P2

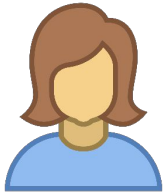


A

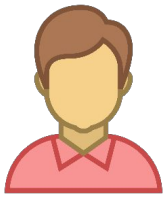
1	2	3	4	5	6
1	3	5	2	4	6

- Instead, list inversions focuses on pairs of movies!

List Inversions: A Real-world Example



P1



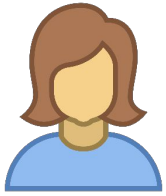
P2

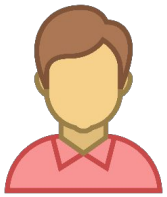
A

1	2	3	4	5	6
1	3	5	2	4	6

- Instead, list inversions focuses on pairs of movies!
  Given the pair of movies Amelie and Almost Famous:
  - P1 prefers Amelie (rank 1) to Almost Famous (rank 2).
    Does P2 also prefer Amelie to Almost Famous?

List Inversions: A Real-world Example





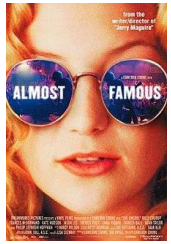
A

1	2	3	4	5	6
1	3	5	2	4	6

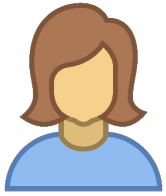
- Instead, list inversions focuses on pairs of movies!
  Given the pair of movies Amelie and Almost Famous:
  - ➢ P1 prefers Amelie (rank 1) to Almost Famous (rank 2).
    Does P2 also prefer Amelie to Almost Famous?

    Yes, P2 also prefers Amelie (rank 1) to Almost Famous (r3).
    Perfect. They are similar on this.

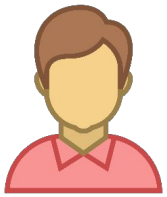
List Inversions: A Real-world Example



P1



P2

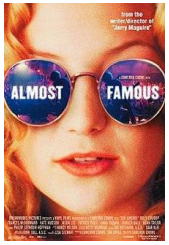


A

1	2	3	4	5	6
1	3	5	2	4	6

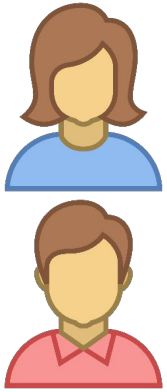
- Instead, list inversions focuses on pairs of movies!
  Given the pair of movies Almost Famous and Wonder:
  - P1 prefers Almost Famous (rank 2) to Wonder (rank 4).
    Does P2 also prefer Almost Famous to Wonder?

List Inversions: A Real-world Example



P1

P2



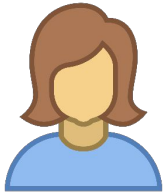
	1	2	3	4	5	6
A	1	3	5	2	4	6

- Instead, list inversions focuses on pairs of movies!
  Given the pair of movies Almost Famous and Wonder:
  - ➢ P1 prefers Almost Famous (rank 2) to Wonder (rank 4).
    Does P2 also prefer Almost Famous to Wonder?

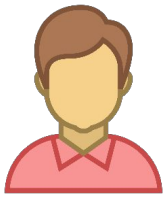
No, in this case P2 prefers Wonder (rank 2) to Almost Famous (rank 3).
So there is a difference in their taste. We have found one inversion!

List Inversions: A Real-world Example

P1



P2



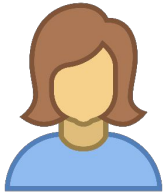
A

1	2	3	4	5	6
1	3	5	2	4	6

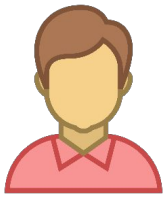
- And we can define computing the List Inversions of array A as:
  - Computing the number of inversions when comparing all movies by pairs.

0

List Inversions: A Real-world Example



P1



P2

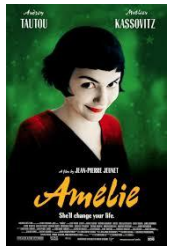
A

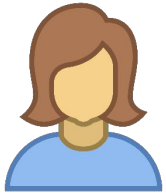
1	2	3	4	5	6
1	3	5	2	4	6

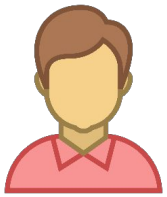
- And we can define computing the List Inversions of array A as:
  - Computing the number of inversions when comparing all movies by pairs.

0

List Inversions: A Real-world Example







P1

P2

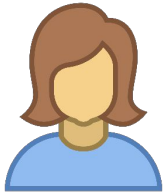
A

1	2	3	4	5	6
1	3	5	2	4	6

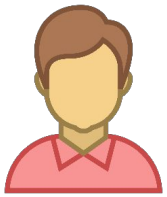
- And we can define computing the List Inversions of array A as:
  - Computing the number of inversions when comparing all movies by pairs.

0

List Inversions: A Real-world Example



P1



P2

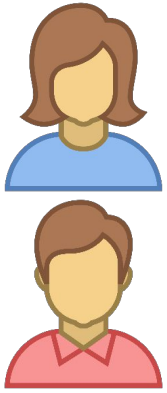
A

1	3	5	2	4	6
1	2	3	4	5	6

- And we can define computing the List Inversions of array A as:
  - Computing the number of inversions when comparing all movies by pairs.

0

List Inversions: A Real-world Example



P1

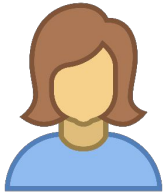
P2

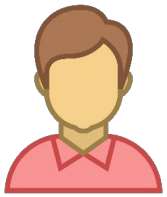
	1	2	3	4	5	6
A	1	3	5	2	4	6

- And we can define computing the List Inversions of array A as:
  - Computing the number of inversions when comparing all movies by pairs.

0

List Inversions: A Real-world Example





P1

P2

A

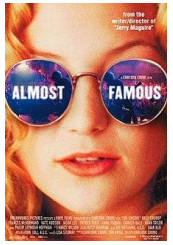
1	3	5	2	4	6

1 2 3 4 5 6

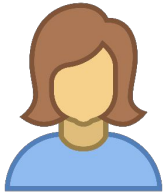
- And we can define computing the List Inversions of array A as:
  - Computing the number of inversions when comparing all movies by pairs.

0

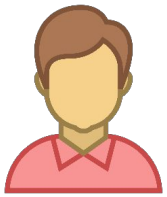
List Inversions: A Real-world Example



P1



P2



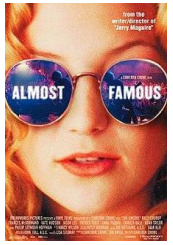
A

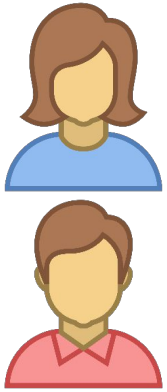
1	2	3	4	5	6
1	3	5	2	4	6

- And we can define computing the List Inversions of array A as:
  - Computing the number of inversions when comparing all movies by pairs.

0

List Inversions: A Real-world Example





P1

P2

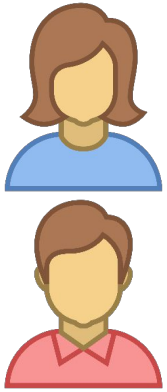
A

1	2	3	4	5	6
1	3	5	2	4	6

- And we can define computing the List Inversions of array A as:
  - Computing the number of inversions when comparing all movies by pairs.

1

List Inversions: A Real-world Example



P1

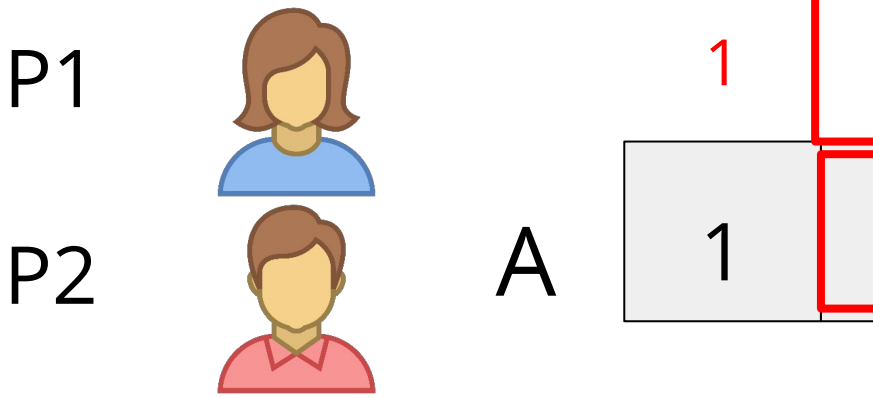
P2

A

1	2	3	4	5	6
1	3	5	2	4	6

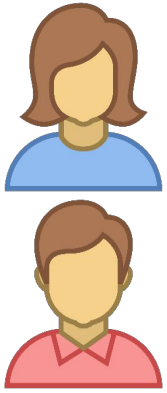
- And we can define computing the List Inversions of array A as:
  - Computing the number of inversions when comparing all movies by pairs.

List Inversions: A Real-world Example



- And we can define computing the List Inversions of array A as:
  - Computing the number of inversions when comparing all movies by pairs.

List Inversions: A Real-world Example



A

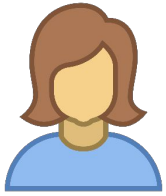
1	2	3	4	5	6
1	3	5	2	4	6

- And we can define computing the List Inversions of array A as:
  - Computing the number of inversions when comparing all movies by pairs.

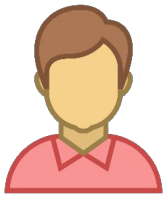
2

List Inversions: A Real-world Example

P1



P2



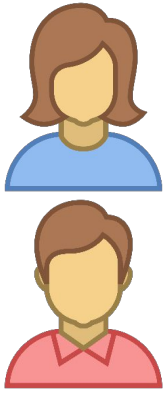
A

1	2	3	4	5	6
1	3	5	2	4	6

- And we can define computing the List Inversions of array A as:
  - Computing the number of inversions when comparing all movies by pairs.

3

List Inversions: A Real-world Example



P1

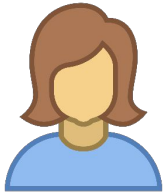
P2

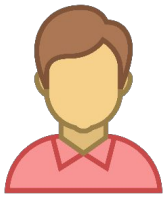
A

1	2	3	4	5	6
1	3	5	2	4	6

- And we can define computing the List Inversions of array A as:
  - Computing the number of inversions when comparing all movies by pairs.

List Inversions: A Real-world Example





P1

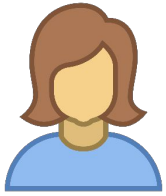
P2

A

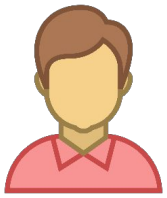
1	2	3	4	5	6
1	3	5	2	4	6

- And we can define computing the List Inversions of array A as:
  - Computing the number of inversions when comparing all movies by pairs.

List Inversions: A Real-world Example



P1



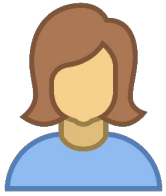
P2

A

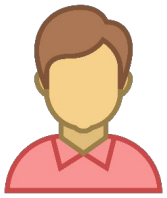
1	2	3	4	5	6
1	3	5	2	4	6

- And we can define computing the List Inversions of array A as:
  - Computing the number of inversions when comparing all movies by pairs.

List Inversions: A Real-world Example



P1



P2

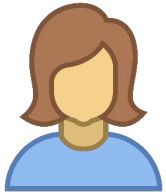
A

1	2	3	4	5	6
1	3	5	2	4	6

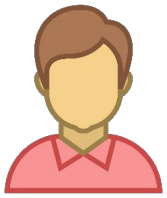
- And we can define computing the List Inversions of array A as:
  - Computing the number of inversions when comparing all movies by pairs.

List Inversions: A Real-world Example

P1



P2



A

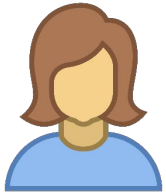
1	2	3	4	5	6
1	3	5	2	4	6

- And we can define computing the List Inversions of array A as:
  - Computing the number of inversions when comparing all movies by pairs.

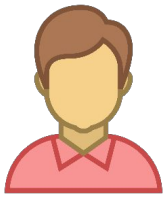
The total amount of list inversions for array A is 3

List Inversions: A Real-world Example

P1



P2



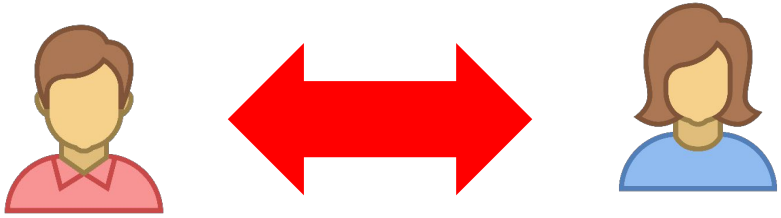
A

1	2	3	4	5	6
1	3	5	2	4	6

So, all in all, we can measure the differences between P1 and P2 as value 3.

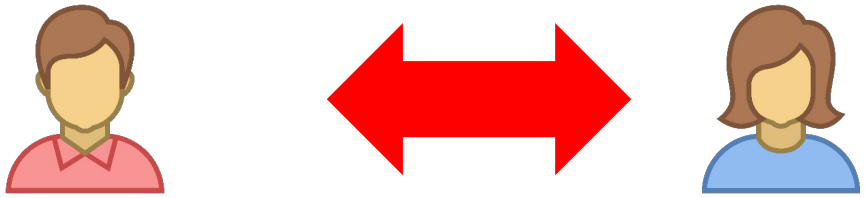
List Inversions: A Real-world Example

But, why to compare P vs P1-only?

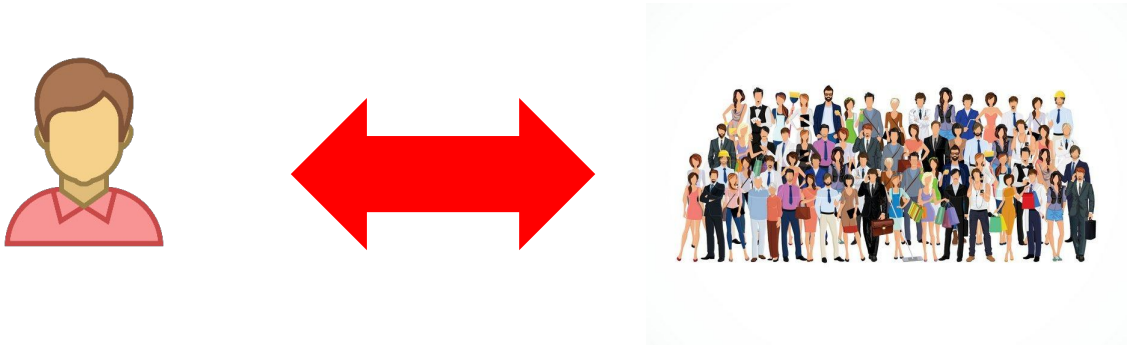


List Inversions: A Real-world Example

But, why to compare P vs P1-only?



Why not to compare P vs an entire population [P1, P2, P3, ..., Pk]?



List Inversions: A Real-world Example

- We can simply extend this very same concept of running list inversions to an entire benchmark (set of instances of the problem).

Person P1  A1

1	2	3	4	5	6
1	3	5	2	4	6

 P vs P1 → 3 inversions.

Person P2  A2

1	2	3	4	5	6
1	6	4	3	5	2

 P vs P2 → 8 inversions.

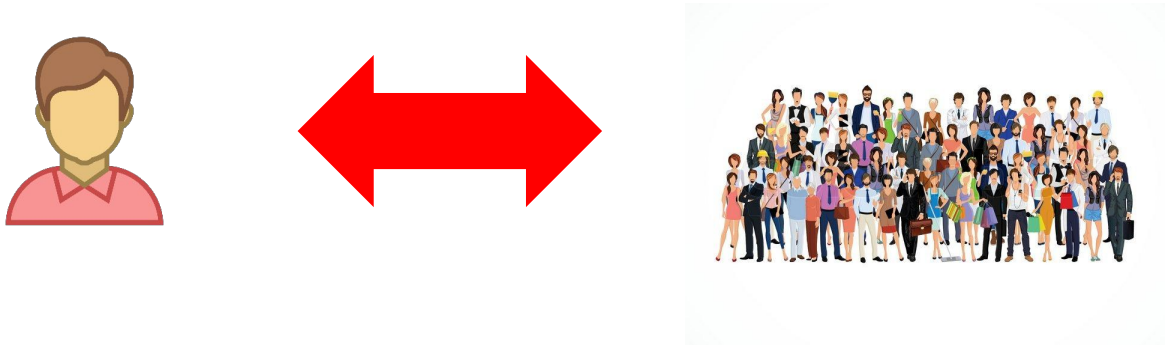
...

Person Pn  An

1	2	3	4	5	6
3	2	5	4	1	6

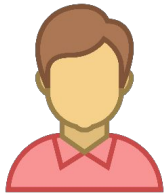
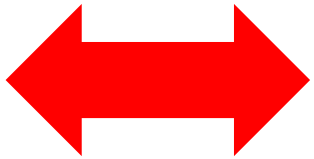
 P vs Pn → 6 inversions.

List Inversions: A Real-world Example



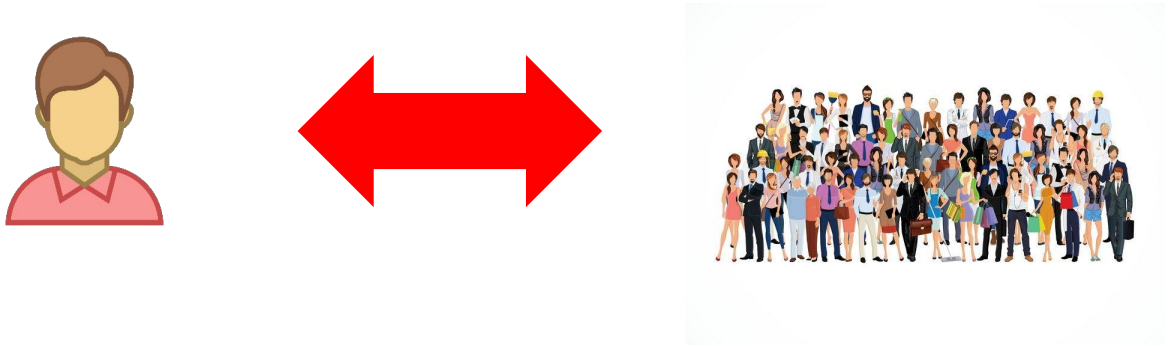
- This new context becomes much more interesting from an Artificial Intelligence point of view.
- In particular, in the context of a Recommender System, where list inversions can be used to measure the distance from P to each person of an entire population.

List Inversions: A Real-world Example


- This new context becomes much more interesting from an Artificial Intelligence point of view.
- In particular, in the context of a Recommender System, where list inversions can be used to measure the distance from P to each person of an entire population.
  - We can run the regression K-Nearest Neighbour algorithm to find the closest person to P in the population.
    - *If person P' is the closest to P, and person P' liked this movies/books/holidays, perhaps P would like them as well.*

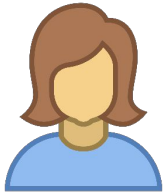
List Inversions: A Real-world Example



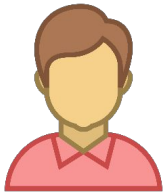
- This new context becomes much more interesting from an Artificial Intelligence point of view.
- In particular, in the context of a Recommender System, where list inversions can be used to measure the distance from P to each person of an entire population.
  - Given a clustering algorithm categorising the population, we can find the cluster P fells under.
    - *If person P belongs to the cluster of "people liking cheesy comedies", maybe we can recommend him these movies/books/holidays as well.*

List Inversions: A Real-world Example

P1

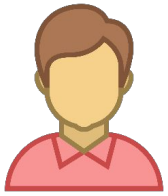


P2



A

1	2	3	4	5	6
1	3	5	2	4	6



All in all, we have seen that list inversions is a very simple but interesting problem.

List Inversions: A Real-world Example

Let's use this problem to guide the 3 goals of today's lecture:

1. Why does scalability matter?
   - The whole idea of big data came into place because a big enough problem becomes non-tractable when following the traditional sequential-based approach.

List Inversions: A Real-world Example

Let's use this problem to guide the 3 goals of today's lecture:

2. Scalability can be tackled via a distributed programming infrastructure.

List Inversions: A Real-world Example

Let's use this problem to guide the 3 goals of today's lecture:

2. Scalability can be tackled via a distributed programming infrastructure.
   - We will show that the same problem can be tackled by distributing the workload over a set of cores/nodes.

List Inversions: A Real-world Example

Let's use this problem to guide the 3 goals of today's lecture:

2. Scalability can be tackled via a distributed programming infrastructure.
   - We will show that the same problem can be tackled by distributing the workload over a set of cores/nodes.
   - On it, the heavy workload will be accomplished in a cooperative way, with each core doing part of the job.

List Inversions: A Real-world Example

Let's use this problem to guide the 3 goals of today's lecture:

2. Scalability can be tackled via a distributed programming infrastructure.
   - We will show that the same problem can be tackled by distributing the workload over a set of cores/nodes.
   - On it, the heavy workload will be accomplished in a cooperative way, with each core doing part of the job.
   - For doing so, we will need a new meta-algorithm responsible for coordinated the distributed execution.

List Inversions: A Real-world Example

Let's use this problem to guide the 3 goals of today's lecture:

3. Scalability can be tackled via a
   more efficient algorithm being used on the first place.

List Inversions: A Real-world Example

Let's use this problem to guide the 3 goals of today's lecture:

3. Scalability can be tackled via a more efficient algorithm being used on the first place.
   - Distributed programming (and the benefits it provides us with) should not make us avoid challenging our algorithms.

List Inversions: A Real-world Example

Let's use this problem to guide the 3 goals of today's lecture:

3. Scalability can be tackled via a more efficient algorithm being used on the first place.
   - Distributed programming (and the benefits it provides us with) should not make us avoid challenging our algorithms.
   - As computer scientists, whatever algorithm we design has to pass the rigorous question: can we do better?

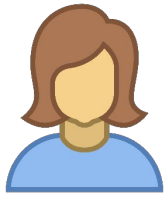
Outline

1. Sequential vs. Concurrent Execution.
2. List Inversions: A Real-world Example.
3. Scalability: Computational Complexity Barrier.
4. Concurrency: Infrastructure-based Approach.
5. Can We Do Better? An Algorithm-based Approach.

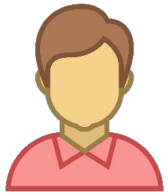
Scalability: Computational Complexity Barrier

Let's think together of an algorithm for computing the list inversions of an array A

P1



P2

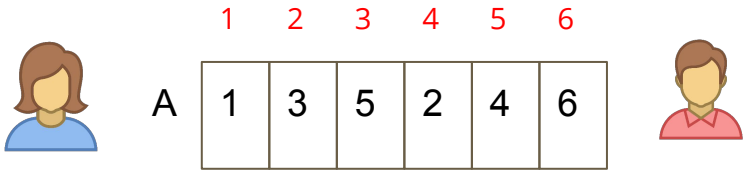


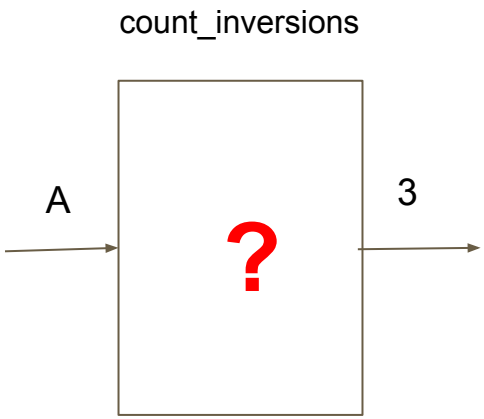
A

1	2	3	4	5	6
1	3	5	2	4	6

Scalability: Computational Complexity Barrier

How would this algorithm look like?





Scalability: Computational Complexity Barrier

How would this algorithm look like?

```
def count_inversions(A):
    # 1. We create the output variable
    res = 0

    # 2. We compute the length of the list
    size = len(A)

    # 3. We iterate to compute the number of inversions
    for i in range(size):
        for j in range(i+1, size):
            if A[i] > A[j]:
                res = res + 1

    # 4. We return res
    return res
```

Scalability: Computational Complexity Barrier

Advantage of this algorithm?

```
def count_inversions(A):
    # 1. We create the output variable
    res = 0

    # 2. We compute the length of the list
    size = len(A)

    # 3. We iterate to compute the number of inversions
    for i in range(size):
        for j in range(i+1, size):
            if A[i] > A[j]:
                res = res + 1

    # 4. We return res
    return res
```

Scalability: Computational Complexity Barrier

Advantage of this algorithm?
It is very simple!

```
def count_inversions(A):
    # 1. We create the output variable
    res = 0

    # 2. We compute the length of the list
    size = len(A)

    # 3. We iterate to compute the number of inversions
    for i in range(size):
        for j in range(i+1, size):
            if A[i] > A[j]:
                res = res + 1

    # 4. We return res
    return res
```

Scalability: Computational Complexity Barrier

Disadvantage of this algorithm?

```
def count_inversions(A):
    # 1. We create the output variable
    res = 0

    # 2. We compute the length of the list
    size = len(A)

    # 3. We iterate to compute the number of inversions
    for i in range(size):
        for j in range(i+1, size):
            if A[i] > A[j]:
                res = res + 1

    # 4. We return res
    return res
```

Scalability: Computational Complexity Barrier

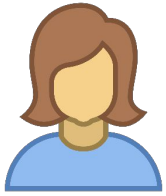
Disadvantage of this algorithm?

It has computational complexity $O(n^2)$!

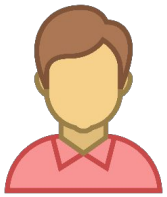
```
def count_inversions(A):
    # 1. We create the output variable
    res = 0

    # 2. We compute the length of the list
    size = len(A)

    # 3. We iterate to compute the number of inversions
    for i in range(size):
        for j in range(i+1, size):
            if A[i] > A[j]:
                res = res + 1

    # 4. We return res
    return res
```

Scalability: Computational Complexity Barrier

P1



P2



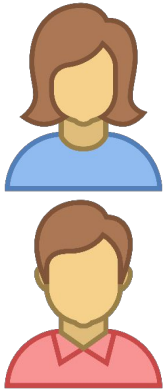
A

1	2	3	4	5	6
1	3	5	2	4	6

- We compare all pair of movies.

0

Scalability: Computational Complexity Barrier



P1

P2

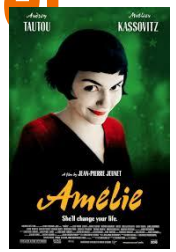
A

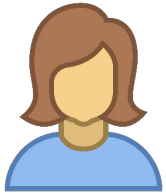
1	2	3	4	5	6
1	3	5	2	4	6

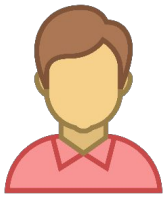
- We compare all pair of movies.

0

Scalability: Computational Complexity Barrier







P1

P2

A

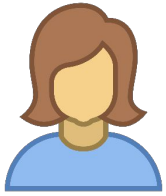
1	3	5	2	4	6

1 2 3 4 5 6

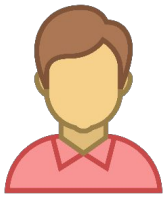
- We compare all pair of movies.

0

Scalability: Computational Complexity Barrier



P1



P2

A

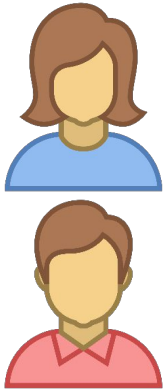
1	3	5	2	4	6

1 2 3 4 5 6

- We compare all pair of movies.

0

Scalability: Computational Complexity Barrier



P1

P2

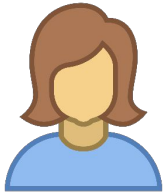
A

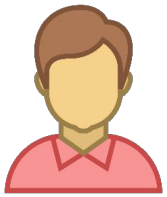
1	2	3	4	5	6
1	3	5	2	4	6

- We compare all pair of movies.

0

Scalability: Computational Complexity Barrier





P1

P2

A

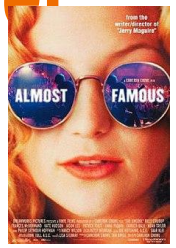
1	3	5	2	4	6

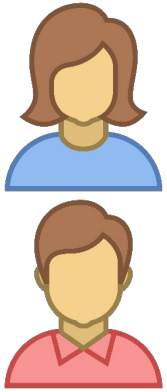
1 2 3 4 5 6

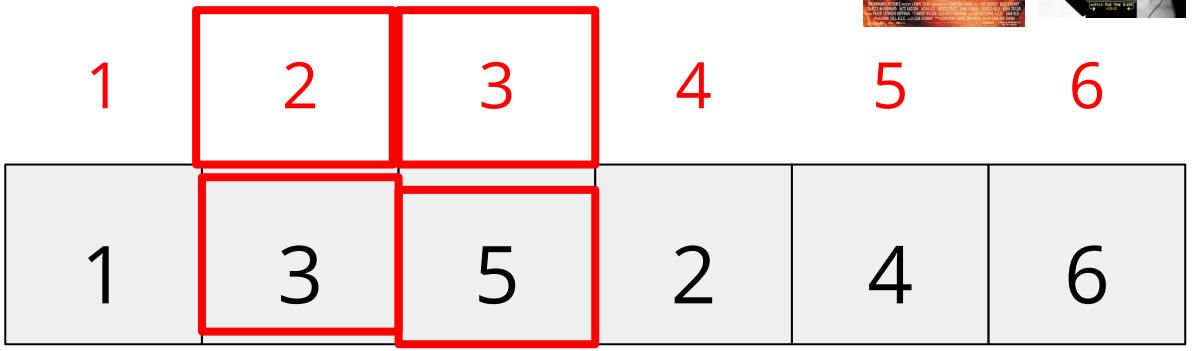
- We compare all pair of movies.

0

Scalability: Computational Complexity Barrier



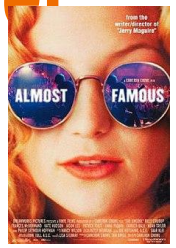




- We compare all pair of movies.

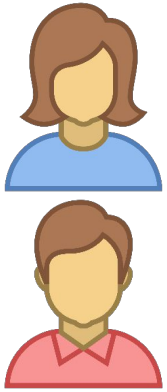
0

Scalability: Computational Complexity Barrier



P1

P2



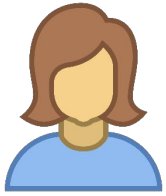
A

1	2	3	4	5	6
1	3	5	2	4	6

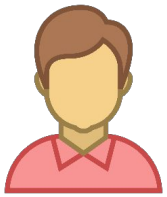
- We compare all pair of movies.

1

Scalability: Computational Complexity Barrier



P1



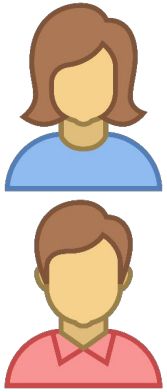
P2

A

1	2	3	4	5	6
1	3	5	2	4	6

- We compare all pair of movies.

Scalability: Computational Complexity Barrier



P1

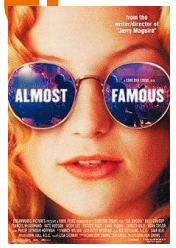
P2

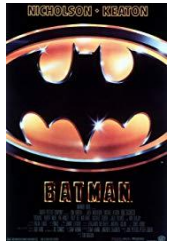
A

1	2	3	4	5	6
1	3	5	2	4	6

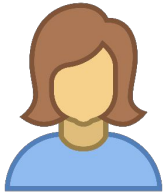
- We compare all pair of movies.

Scalability: Computational Complexity Barrier

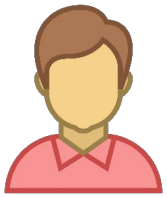




P1



P2



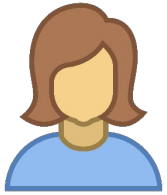
A

1	2	3	4	5	6
1	3	5	2	4	6

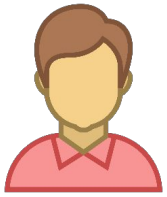
- And so on...

Scalability: Computational Complexity Barrier

P1



P2



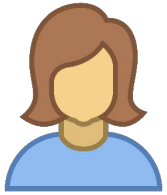
A

1	2	3	4	5	6
1	3	5	2	4	6

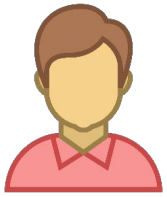
- So in total we compare:
  - Movie 1 with 5 movies [2,3,4,5,6]
  - Movie 2 with 4 movies [3,4,5,6]
  - Movie 3 with 3 movies [4,5,6]
  - Movie 4 with 2 movies [5,6]
  - Movie 5 with 1 movie [6]

Scalability: Computational Complexity Barrier

P1



P2

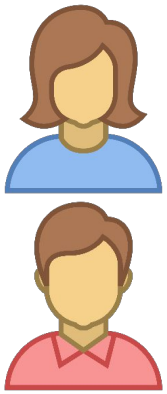


A

1	2	3	4	5	6
1	3	5	2	4	6

- So in total we compare:
  - Movie 1 with 5 movies [2,3,4,5,6]
  - Movie 2 with 4 movies [3,4,5,6]
  - Movie 3 with 3 movies [4,5,6]
  - Movie 4 with 2 movies [5,6]
  - Movie 5 with 1 movie [6]
- Given an array of size 6, the total amount of operations can be computed as the sum of 1 + 2 + 3 + 4 + 5

Scalability: Computational Complexity Barrier

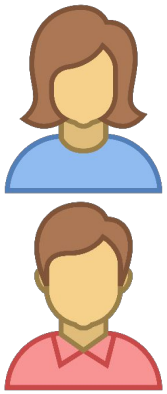


A

1	2	3	4	...	n
1	3	5	2		6

- In general, given an array of size *n*, the total amount of operations can be computed as the sum of the first n-1 natural numbers [1, 2 ..., n-1]

Scalability: Computational Complexity Barrier

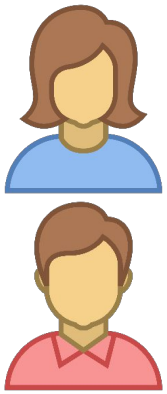


A

1	2	3	4	...	n
1	3	5	2		6

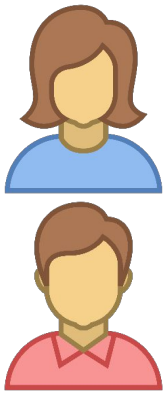
- In general, given an array of size *n*, the total amount of operations can be computed as the sum of the first n-1 natural numbers [1, 2 ..., n-1]
- Given the formula provided by Gauss this sum is known to be ((n-1) * n) / 2, or once reduced => **$(n^2 - 2) / 2$**

Scalability: Computational Complexity Barrier



- In general, given an array of size *n*, the total amount of operations can be computed as the sum of the first n-1 natural numbers [1, 2 ..., n-1]
- Given the formula provided by Gauss this sum is known to be ((n-1) * n) / 2, , or once reduced => **$(n^2 - 2) / 2$**
- And in complexity theory the expression $(n^2 - 2) / 2$ is approximated to **n^2**.

Scalability: Computational Complexity Barrier



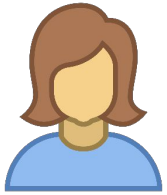
A

1	2	3	4	...	n
1	3	5	2		6

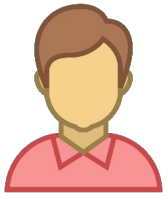
- This means that the number of operations required to solve the problem will increase quadratically as the size of the problem n grows linearly.

Scalability: Computational Complexity Barrier

P1

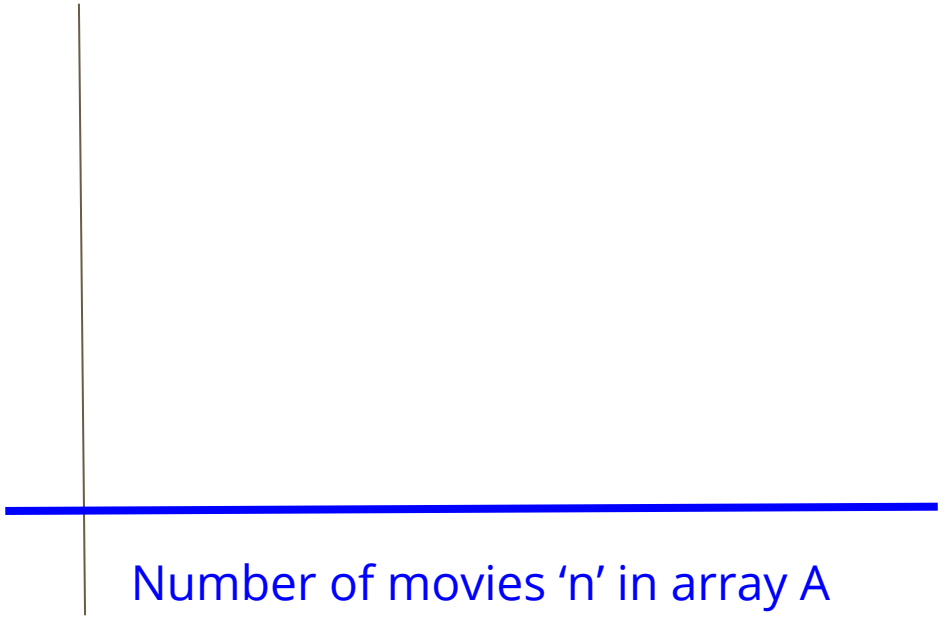


P2



A

1	2	3	4	...	n
1	3	5	2		6

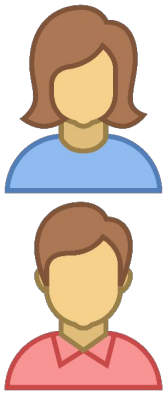


Number of movies ‘n’ in array A
(Grows linearly)

Scalability: Computational Complexity Barrier

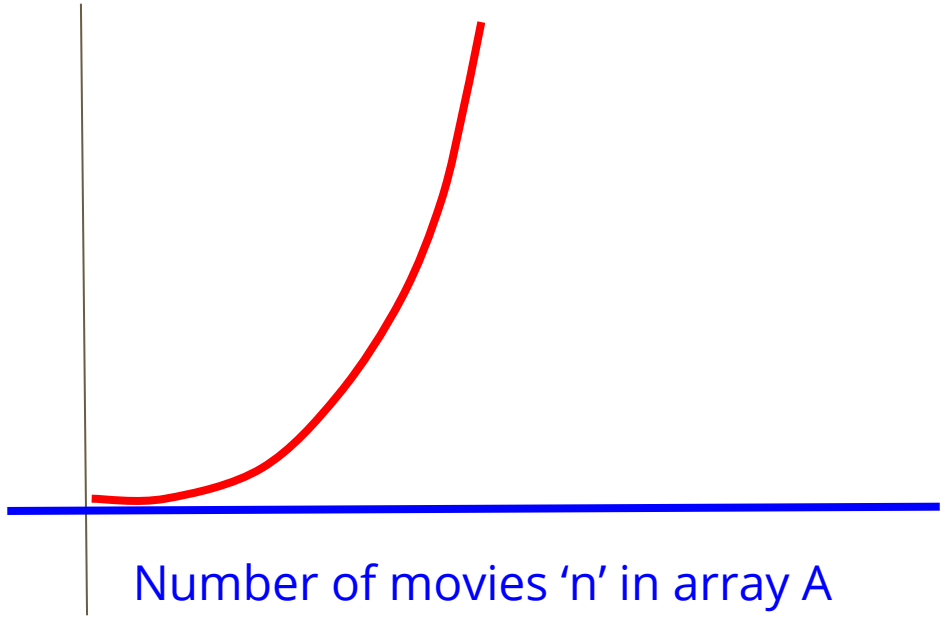
P1

P2

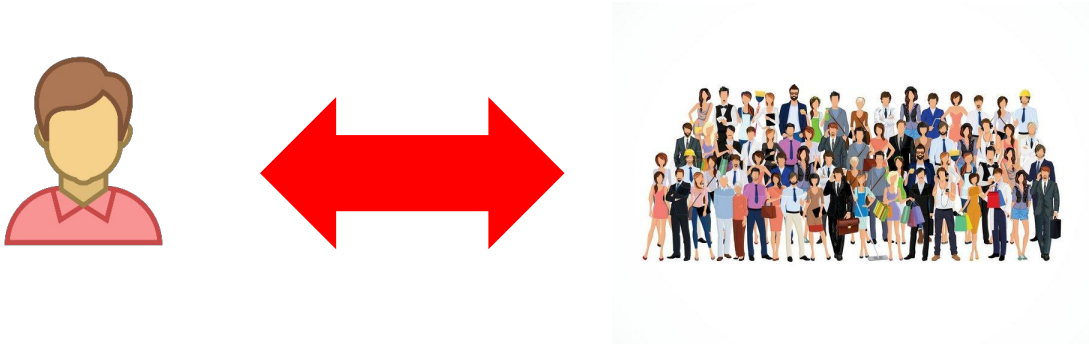


A

1	2	3	4	...	n
1	3	5	2		6

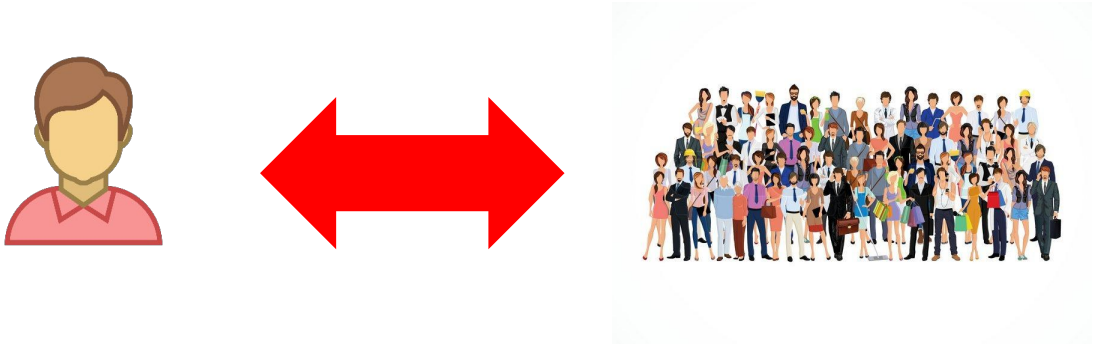


Scalability: Computational Complexity Barrier



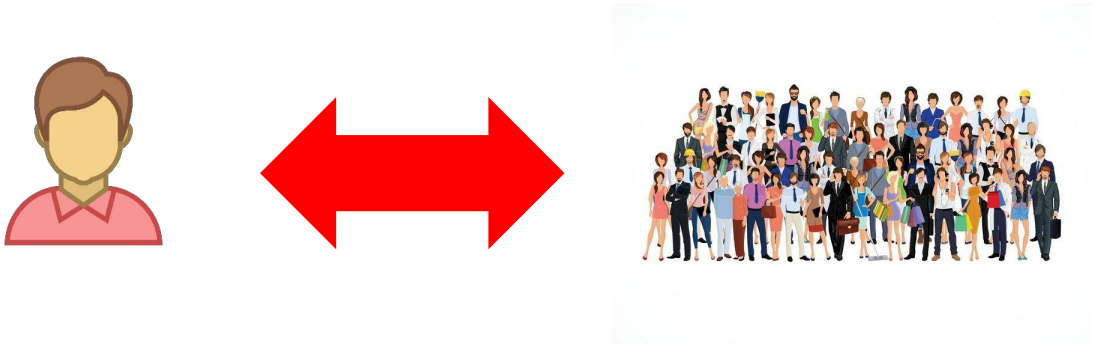
- And if we want to extend the problem to the case in which we compare P against an entire population of **k** people, then the total amount of operations will be:
  K * ((n^2 - 2) / 2)

Scalability: Computational Complexity Barrier



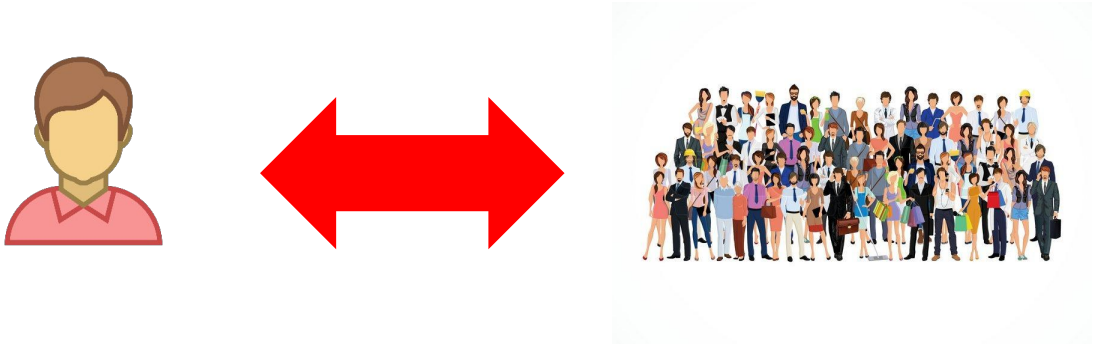
- And if we want to extend the problem to the case in which we compare P against an entire population of **k** people, then the total amount of operations will be:
  K * ((n^2 - 2) / 2)
  - ➢ Where **((n^2 - 2) / 2)** are the operations required to solve a single instance of the problem (e.g., comparing P and Pi).

Scalability: Computational Complexity Barrier



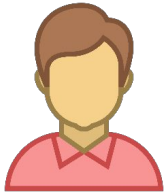
- And if we want to extend the problem to the case in which we compare P against an entire population of **k** people, then the total amount of operations will be:
  K * ((n^2 - 2) / 2)
  - ➢ Where **((n^2 - 2) / 2)** are the operations required to solve a single instance of the problem (e.g., comparing P and Pi).
  - ➢ Where **k** are the number of people in the population [P1, P2, ..., Pk]

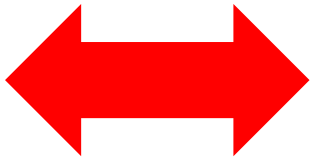
Scalability: Computational Complexity Barrier



- And if we want to extend the problem to the case in which we compare P against an entire population of **k** people, then the total amount of operations will be:
  K * ((n^2 - 2) / 2)
  - ➢ Where **((n^2 - 2) / 2)** are the operations required to solve a single instance of the problem (e.g., comparing P and Pi).
  - ➢ Where **k** are the number of people in the population [P1, P2, ..., Pk]

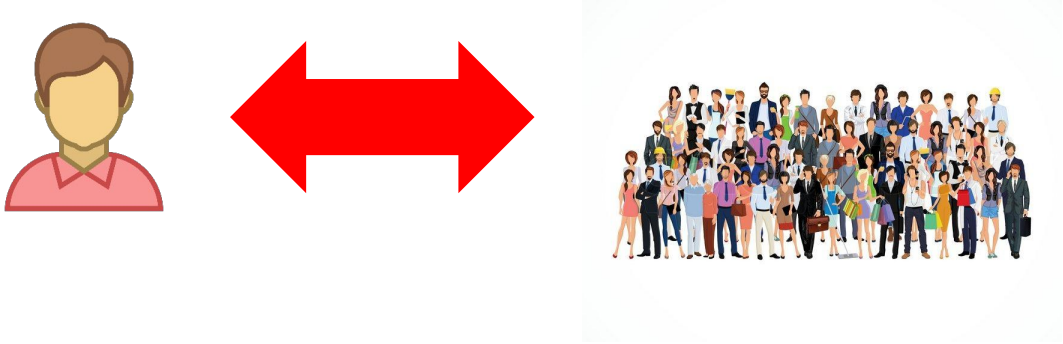
Scalability: Computational Complexity Barrier





- The code examples associated to this lecture provide us with a benchmark generator & solver for running the problem of computing list inversions of person P vs an entire population.

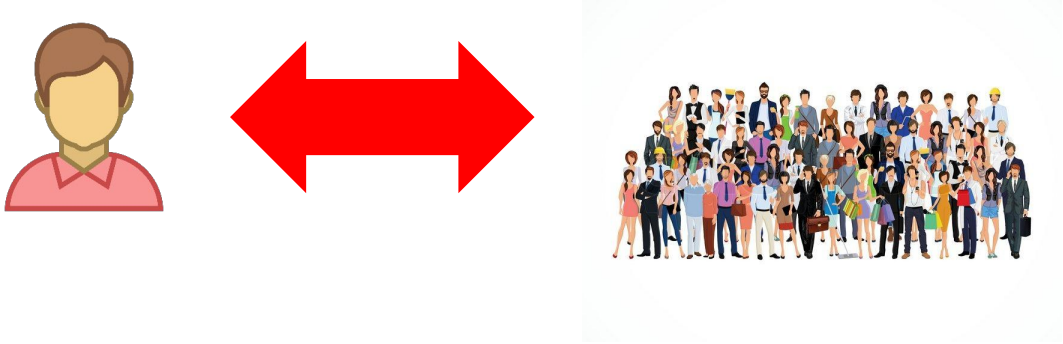
Scalability: Computational Complexity Barrier



- The code examples associated to this lecture provide us with a benchmark generator & solver for running the problem of computing list inversions of person P vs an entire population.

- We are going to use this tool now to test the scalability of:
  - Compute List Inversions of P vs the population using:

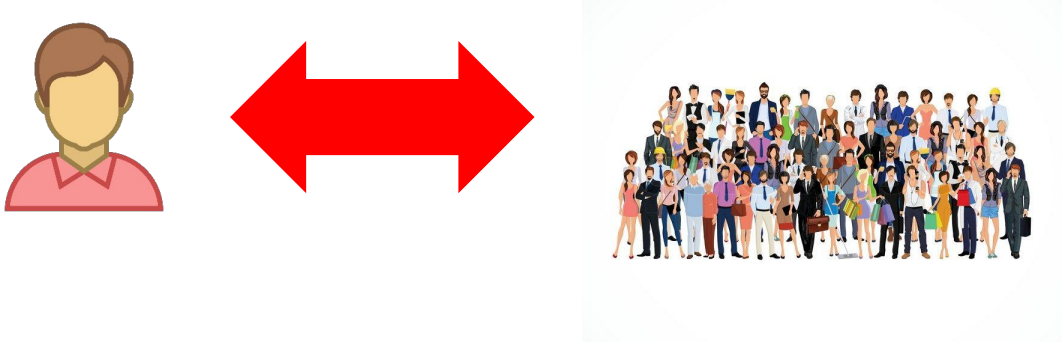
Scalability: Computational Complexity Barrier



- The code examples associated to this lecture provide us with a benchmark generator & solver for running the problem of computing list inversions of person P vs an entire population.

- We are going to use this tool now to test the scalability of:
  - Compute List Inversions of P vs the population using:
    - The algorithm count_inversions of $O(n^2)$.

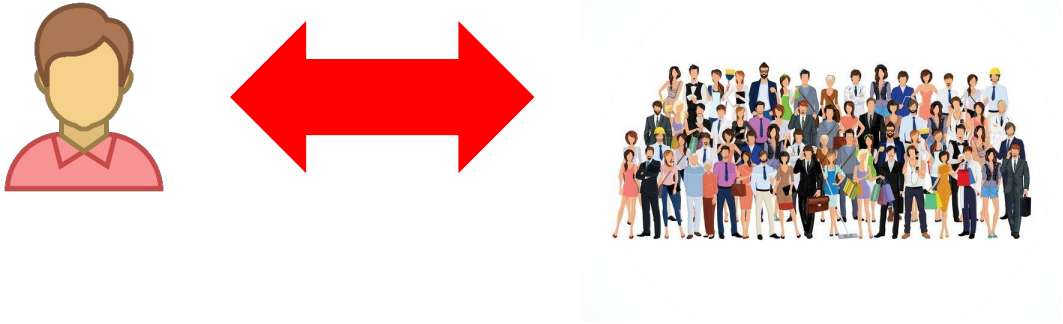
Scalability: Computational Complexity Barrier



- The code examples associated to this lecture provide us with a benchmark generator & solver for running the problem of computing list inversions of person P vs an entire population.

- We are going to use this tool now to test the scalability of:
  - Compute List Inversions of P vs the population using:
    - The algorithm count_inversions of $O(n^2)$.
    - A sequential-based approach.

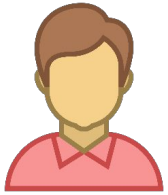
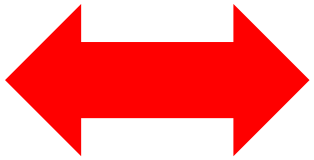
Scalability: Computational Complexity Barrier



- The code examples associated to this lecture provide us with a benchmark generator & solver for running the problem of computing list inversions of person P vs an entire population.

- We are going to use this tool now to test the scalability of:
  - Compute List Inversions of P vs the population using:
    - The algorithm count_inversions of $O(n^2)$.
    - A sequential-based approach.
    - A fixed population size k = 12.

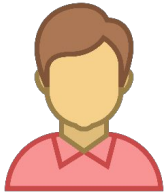
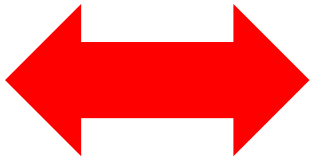
Scalability: Computational Complexity Barrier


- The code examples associated to this lecture provide us with a benchmark generator & solver for running the problem of computing list inversions of person P vs an entire population.

- We are going to use this tool now to test the scalability of:
  - Compute List Inversions of P vs the population using:
    - The algorithm count_inversions of $O(n^2)$.
    - A sequential-based approach.
    - A fixed population size k = 12.
    - A number of movies ranging from 1,000 to 32,000.

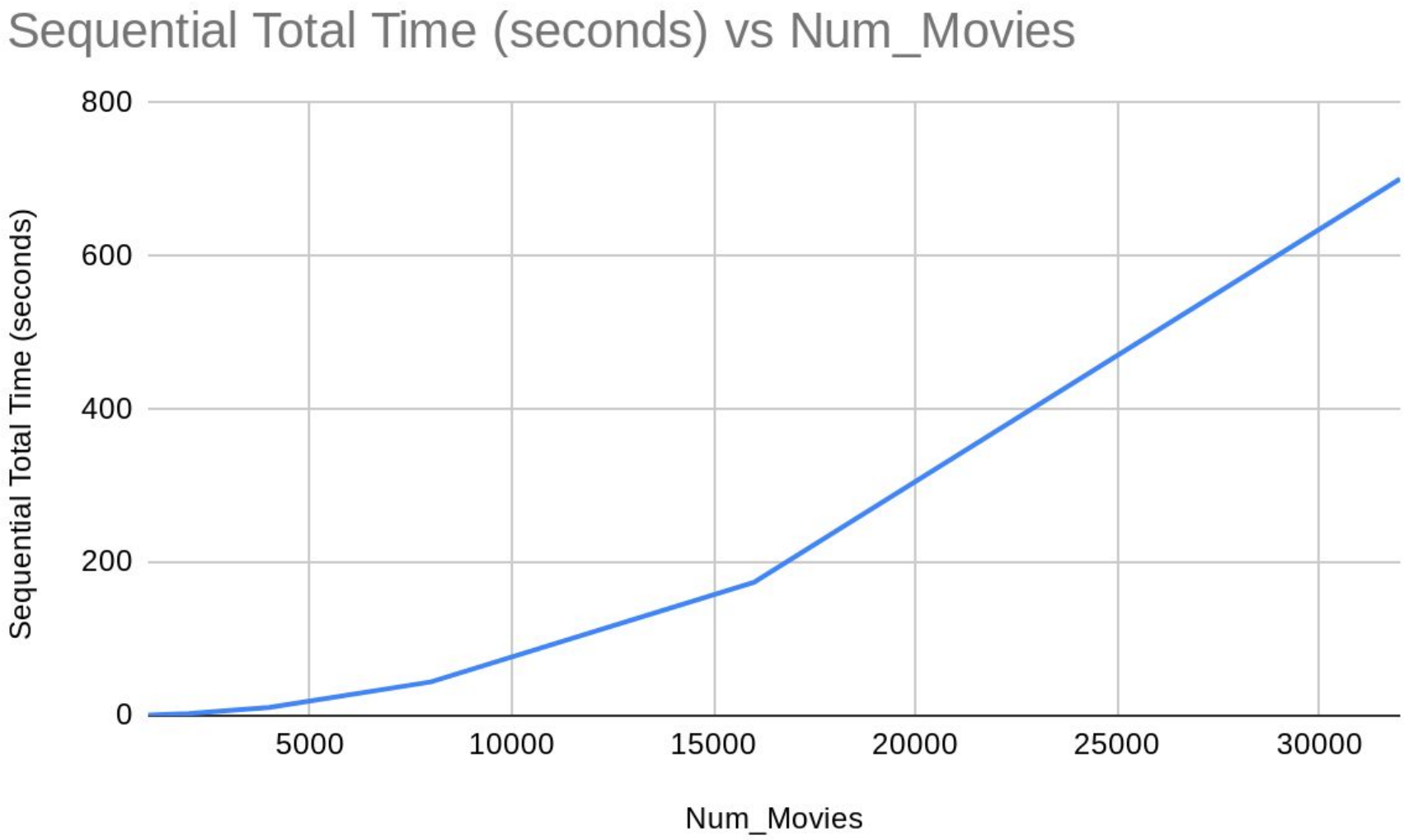
Scalability: Computational Complexity Barrier

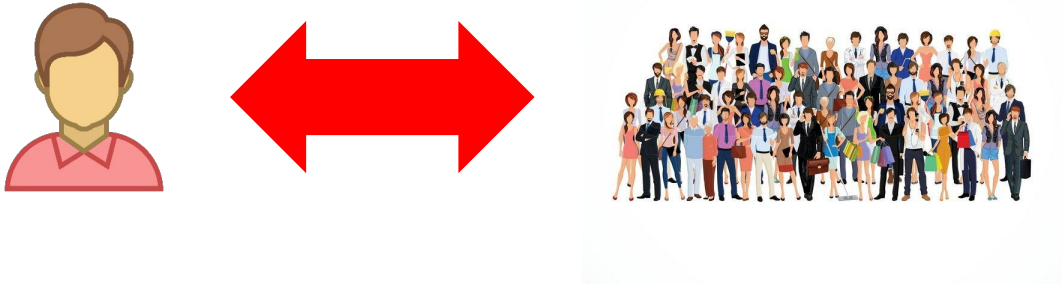

- The results are presented below:

Num_Movies	1000	2000	4000	8000	16000	32000
Sequential Total Time (seconds)	0.7	2.75	10.93	44.03	174.04	700.9

Scalability: Computational Complexity Barrier

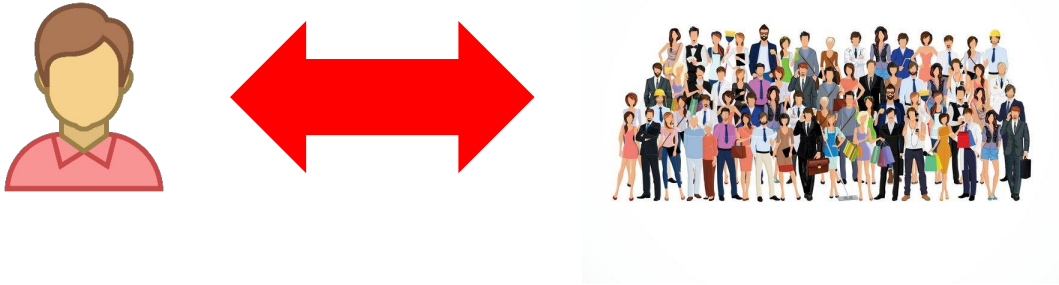


Scalability: Computational Complexity Barrier



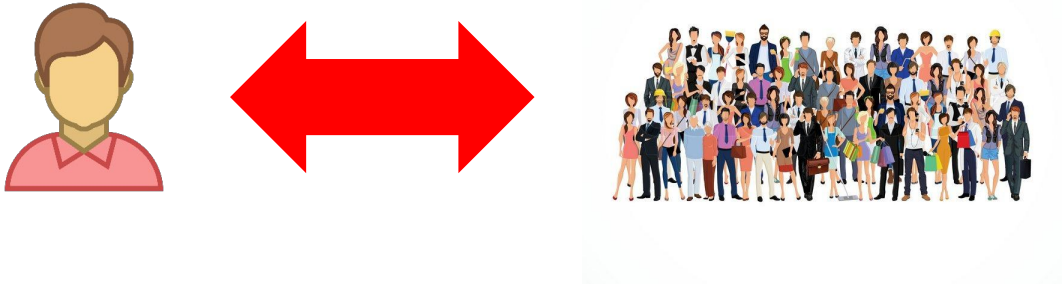
- As we can see, the number of operations increase according to our expression **K * ((n^2 - 2) / 2).**

Scalability: Computational Complexity Barrier



- As we can see, the number of operations increase according to our expression **K * ((n^2 - 2) / 2).**
- Even if we fix k to a small value (e.g, 12) the expression is still dominated by the n^2 number of operations required by our algorithm for solving each single instance of the problem.

Scalability: Computational Complexity Barrier

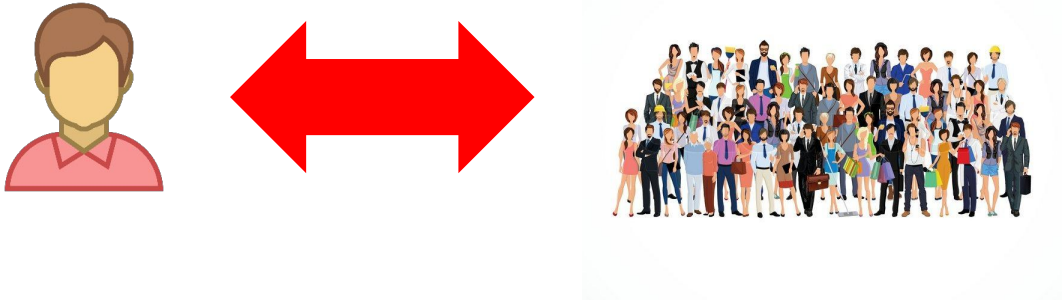


- As we can see, the number of operations increase according to our expression **K * ((n^2 - 2) / 2).**
- Even if we fix k to a small value (e.g, 12) the expression is still dominated by the n^2 number of operations required by our algorithm for solving each single instance of the problem.
- This makes unaffordable to deal with cases where there are hundreds of thousands or millions of movies **n**.

Outline

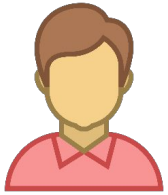
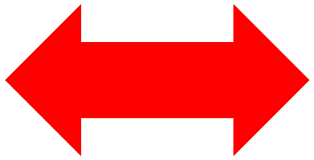
1. Sequential vs. Concurrent Execution.
2. List Inversions: A Real-world Example.
3. Scalability: Computational Complexity Barrier.
4. Concurrency: Infrastructure-based Approach.
5. Can We Do Better? An Algorithm-based Approach.

Concurrency: Infrastructure-Based Approach



- To alleviate the workload of **K * ((n^2 - 2) / 2)** operations we can just balance it among multiple cores.

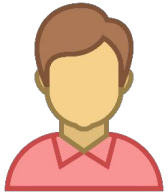
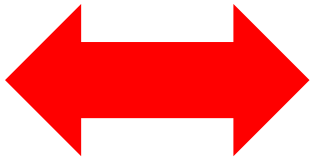
Concurrency: Infrastructure-Based Approach


- The rationale is extremely simple:
  - If we have **c** number of cores, then we can split our workload of **K * ((n^2 - 2) / 2)** among them.

...

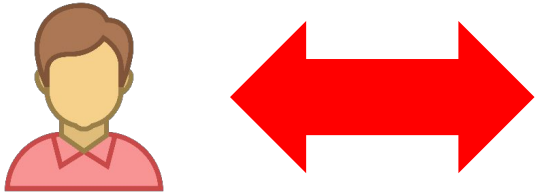

Concurrency: Infrastructure-Based Approach


- The rationale is extremely simple:
  - If we have **c** number of cores, then we can split our workload of **K * ((n^2 - 2) / 2)** among them.
  - Each core will do **(K * ((n^2 - 2) / 2)) / c** operations.

...

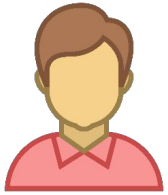
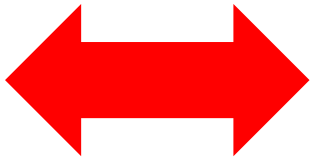

Concurrency: Infrastructure-Based Approach



- The rationale is extremely simple:
  - If we have **c** number of cores, then we can split our workload of **K * ((n^2 - 2) / 2)** among them.
  - Yes, the whole set of cores will still do the original **K * ((n^2 - 2) / 2)** operations we described in the analysis of our algorithm...

...

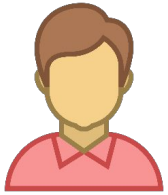
Concurrency: Infrastructure-Based Approach

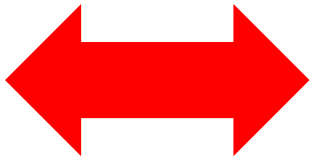




- The rationale is extremely simple:
  - If we have **c** number of cores, then we can split our workload of **K * ((n^2 - 2) / 2)** among them.
  - ...however, they will accomplish these **K * ((n^2 - 2) / 2)** by working in parallel!

...


Concurrency: Infrastructure-Based Approach

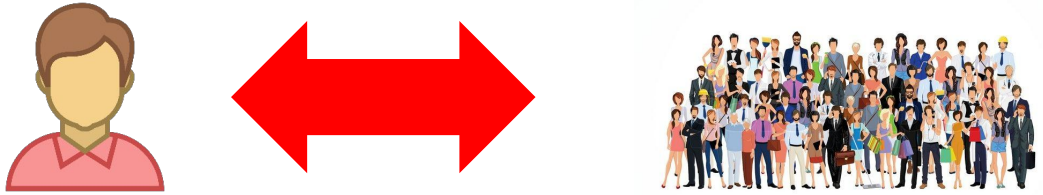




- The rationale is extremely simple:
  - If we have **c** number of cores, then we can split our workload of **K * ((n^2 - 2) / 2)** among them.
  - ...however, they will accomplish these **K * ((n^2 - 2) / 2)** by working in parallel!
    - Each core doing its **(K * ((n^2 - 2) / 2)) / c** at the same time as the other cores work on their own workload.

...

Concurrency: Infrastructure-Based Approach

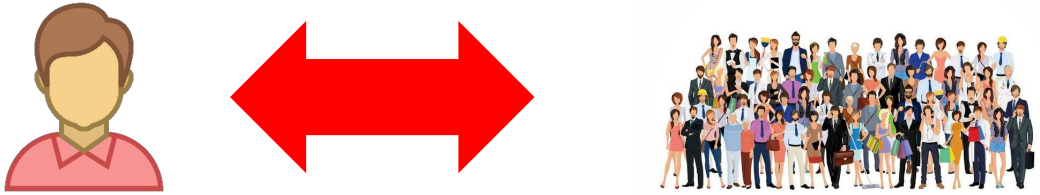


- In this context, the total elapsed time we will observe for doing the total **K * ((n^2 - 2) / 2)** operations will be:
  - The one of the slowest core c' on doing its **(K * ((n^2 - 2) / 2)) / c** subset of operations.

Concurrency: Infrastructure-Based Approach

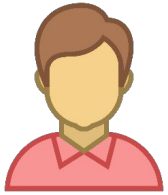
Let’s do an analogy/metaphor for this via an athletics race.

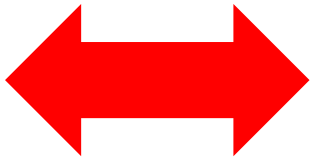
Concurrency: Infrastructure-Based Approach



- Shaunae Miller won the 400m gold medal in the Olympic Games Rio 2016, with a time of **49.44** seconds.

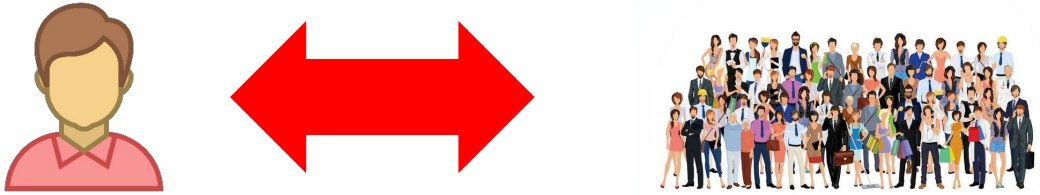
Concurrency: Infrastructure-Based Approach





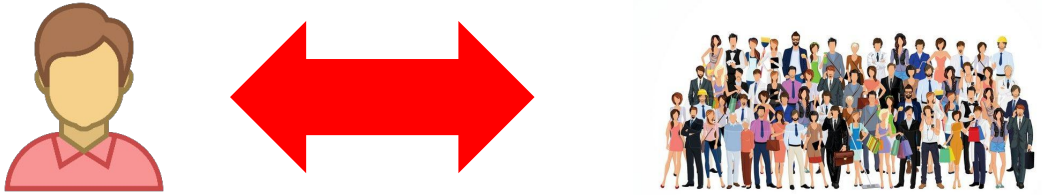
- Shaunae Miller won the 400m gold medal in the Olympic Games Rio 2016, with a time of 49.44 seconds.
  - In our metaphor Shaunae is 1 core.

Concurrency: Infrastructure-Based Approach

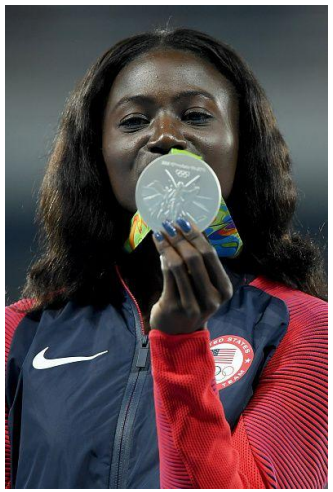


- Shaunae Miller won the 400m gold medal in the Olympic Games Rio 2016, with a time of 49.44 seconds.
  - In our metaphor Shaunae is 1 core.
  - When we watched the 400m race, we were stick to the TV screen for **49.44** seconds, as this was the time Shaunae took to run the whole 400m (in our metaphor the whole set of operations).

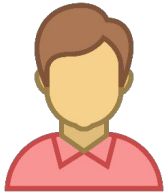
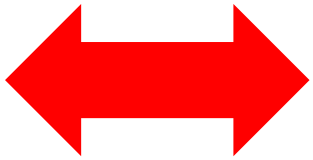
Concurrency: Infrastructure-Based Approach



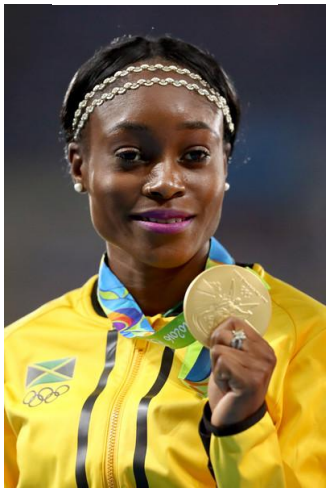
- The first four runners in the 100m race in the Olympic Games Rio 2016 were:
  - Elaine Thompson, **10.71** seconds
  - Tori Bowie, **10.83** seconds
  - Shelly-Ann Fraser-Pryce, **10.86** seconds
  - Marie-Josee Ta Lou, **10.86** seconds

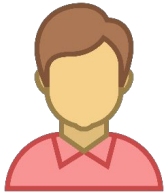
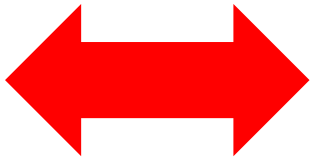

Concurrency: Infrastructure-Based Approach


- In our metaphor they are 4 cores.

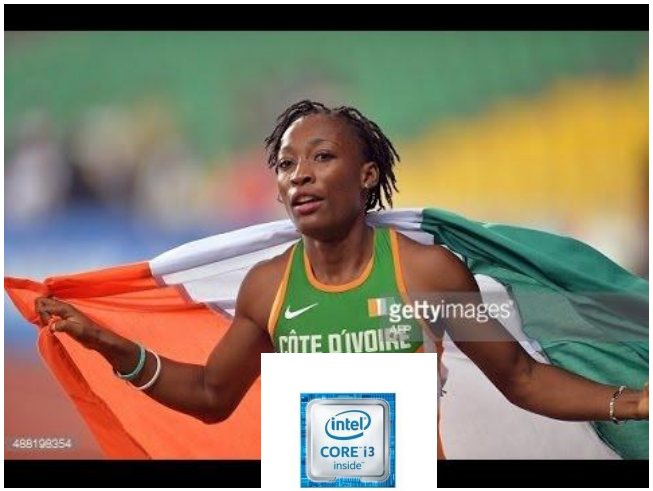


Concurrency: Infrastructure-Based Approach

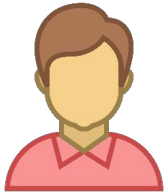
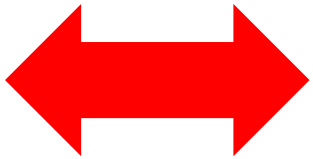




- Sure, the 4 of them together ran... 400m.
  - This is the same amount of meters Shaunae did on her own in the 400m race.



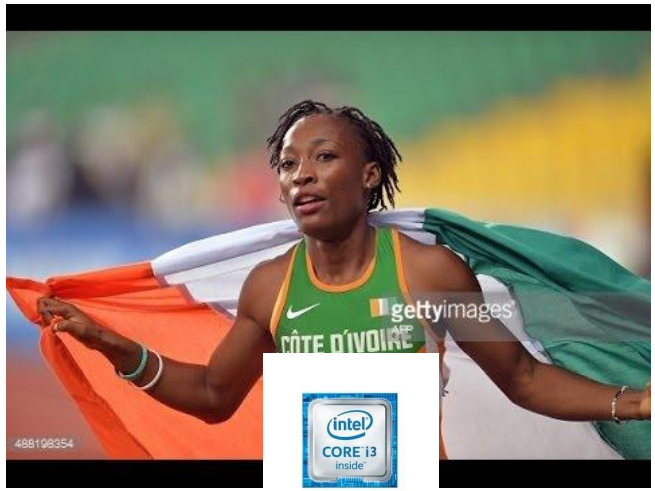



Concurrency: Infrastructure-Based Approach

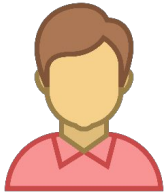
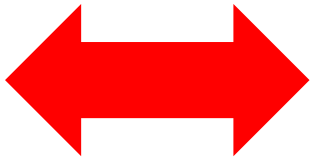




- Sure, the 4 of them together ran… 400m.
  - And yes, on doing all together these 400 meters each of them ran its 100m slices for **10.71**, **10.83**, **10.86** and **10.86** seconds, respectively.



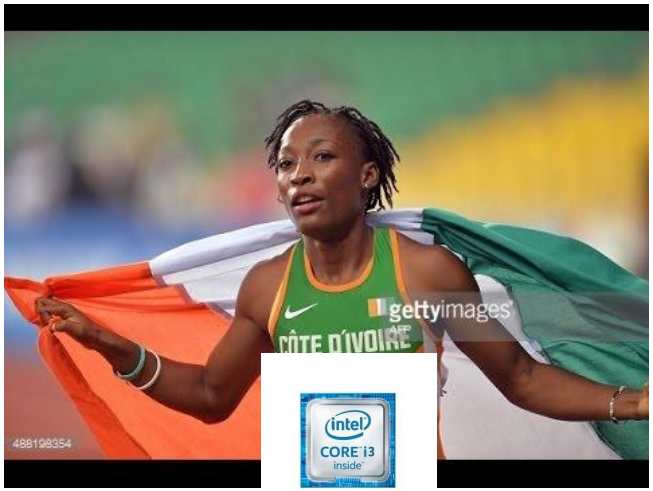



Concurrency: Infrastructure-Based Approach

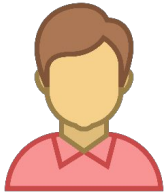
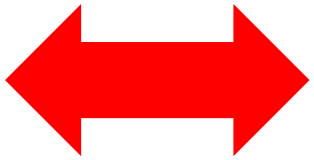




- Sure, the 4 of them together ran… 400m.
  - Collectively they ran for 10.71 + 10.83 + 10.86 + 10.86 = **43.26** seconds.



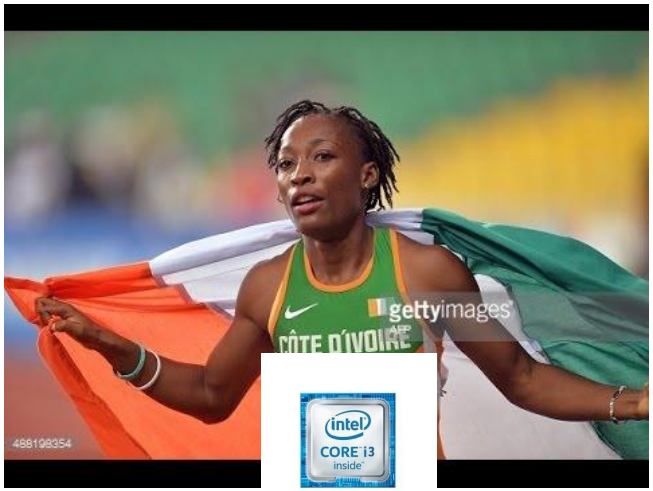



Concurrency: Infrastructure-Based Approach

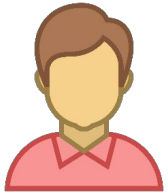
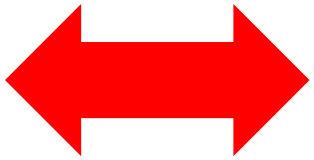




- Sure, the 4 of them together ran... 400m.
  - However the time we stick to the TV to see the 4 of them complete their race was only **10.86** seconds, the time of the slowest runner doing its 100m slice.



Concurrency: Infrastructure-Based Approach


- So when we compare both approaches we can do it at two levels:

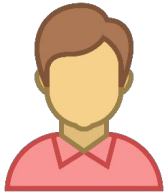
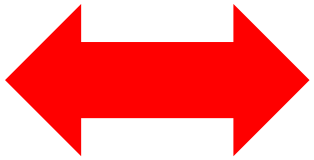
1. **TOTAL COMPUTATION TIME**
   - Sequential Mode (Shaunae) total time = **49.44** seconds.
   - Distributed Mode (Elaine, Tori, Shelly, Marie) total time = **43.26** seconds.

Sequential Mode (single core)

vs.

Distributed Mode (multiple cores)

Concurrency: Infrastructure-Based Approach


- So when we compare both approaches we can do it at two levels:

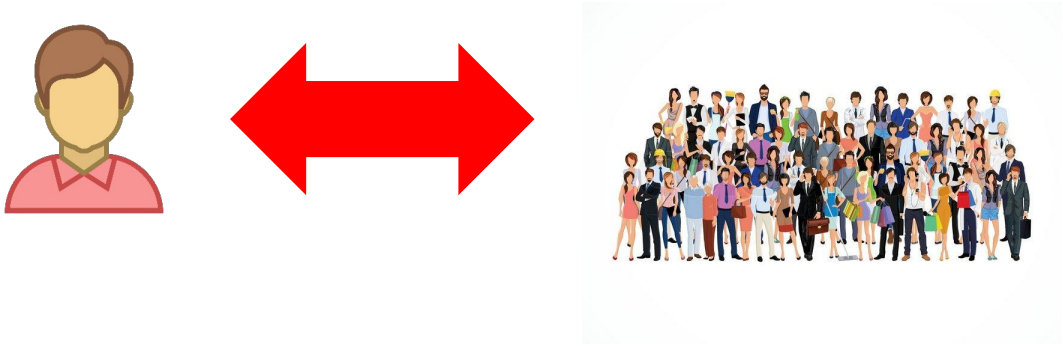
2. **TOTAL ELAPSED TIME**
   - Sequential Mode (Shaunae) total time = **49.44** seconds.
   - Distributed Mode (Elaine, Tori, Shelly, Marie) total time = **10.86** seconds.

Sequential Mode (single core)

vs.

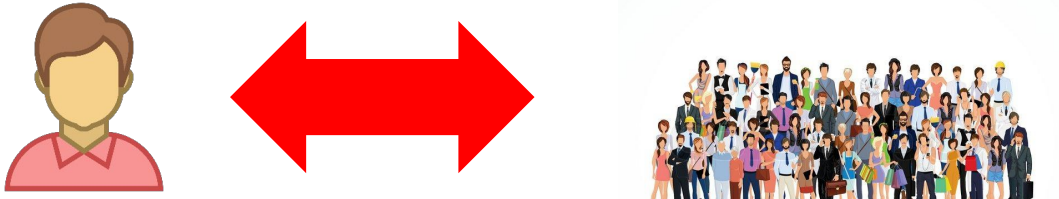
Distributed Mode (multiple cores)

Concurrency: Infrastructure Based Approach



- The code examples associated to this lecture provide us with a benchmark generator & solver for running the problem of computing list inversions of person P vs an entire population.

Concurrency: Infrastructure Based Approach



- The code examples associated to this lecture provide us with a benchmark generator & solver for running the problem of computing list inversions of person P vs an entire population.

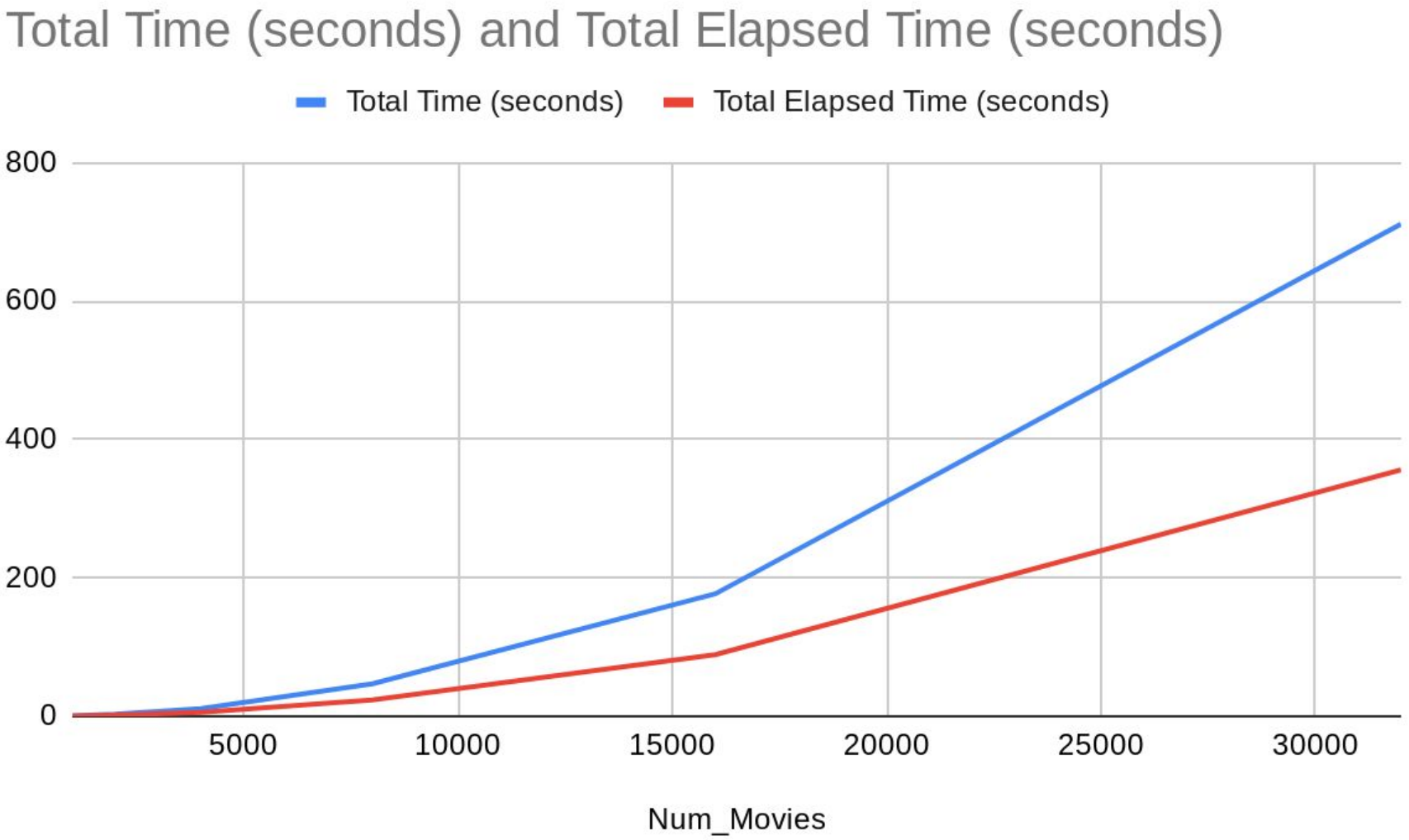
- We are going to use this tool now to test the scalability of:
  - Compute List Inversions of P vs the population using:
    - The algorithm count_inversions of $O(n^2)$.
    - A parallel-based approach with 2 cores.
    - A fixed population size k = 12.
    - A number of movies ranging from 1,000 to 32,000.

Concurrency: Infrastructure Based Approach

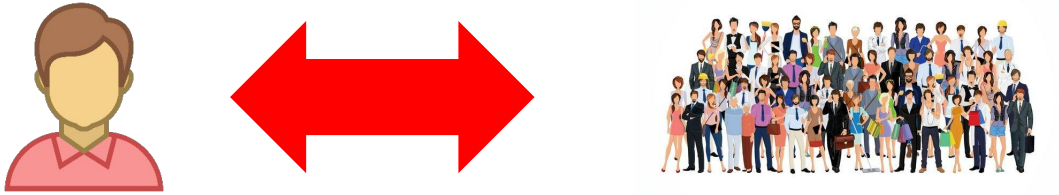
- The results are presented below:

Num_Movies	1000	2000	4000	8000	16000	32000
Parallel 2-cores Total Time	0.86	2.85	11.19	46.91	177.02	711.43
Parallel 2-cores Elapsed Time	0.49	1.43	5.59	23.48	89.1	356.32

Concurrency: Infrastructure Based Approach



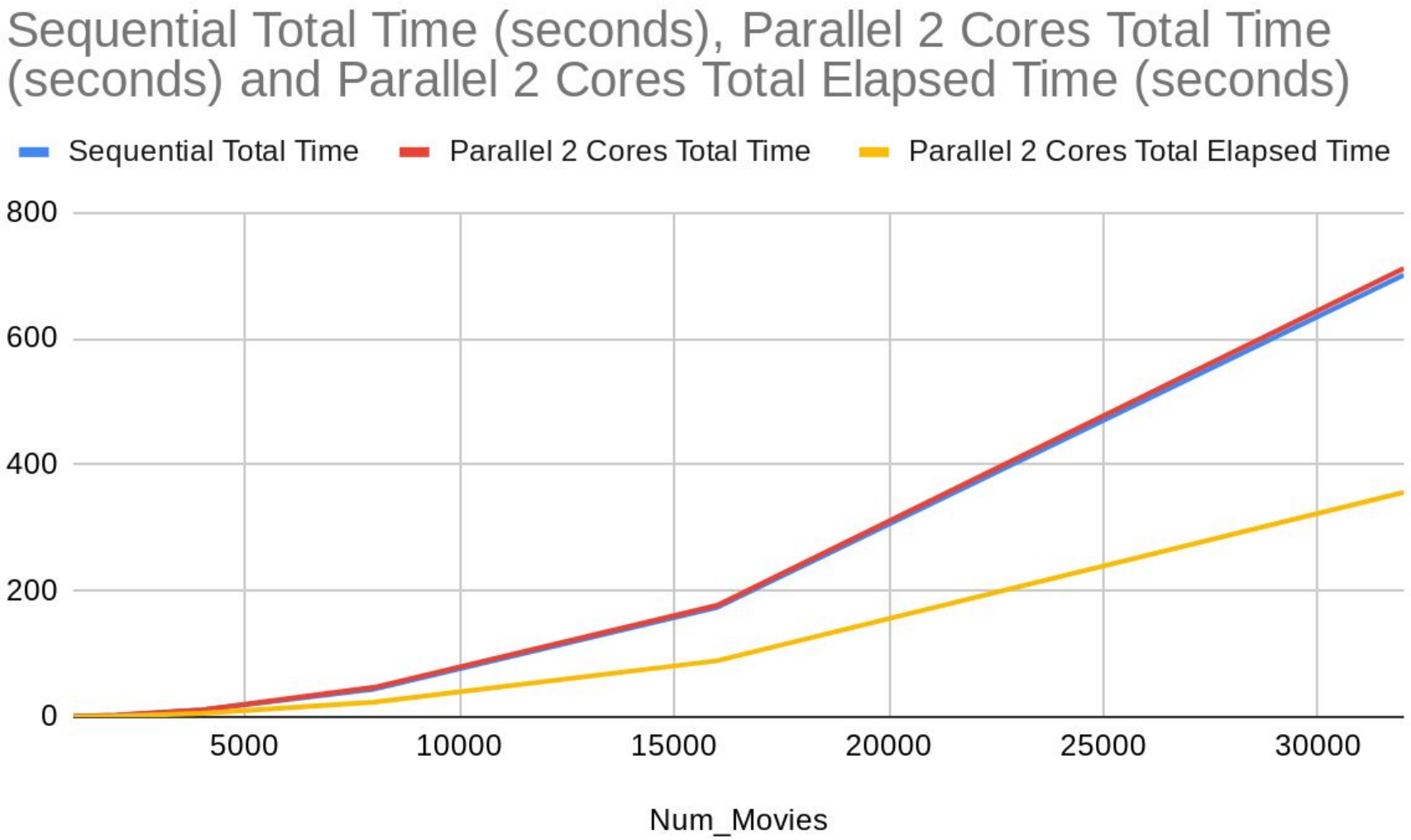
Concurrency: Infrastructure Based Approach



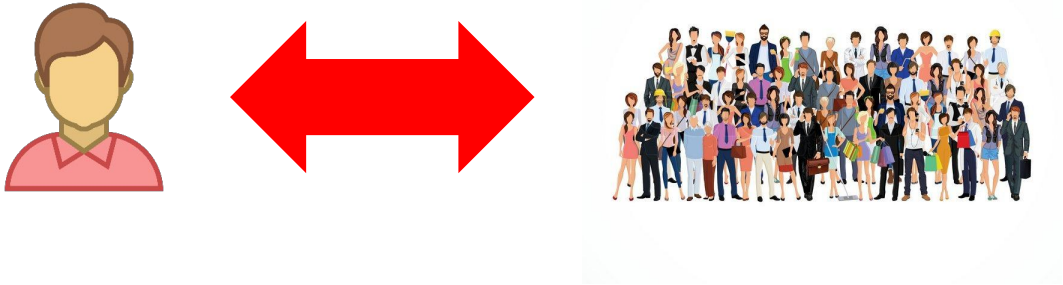
- And when we compare sequential and distributed (2 cores):

Num_Movies	1000	2000	4000	8000	16000	32000
Sequential Total Time	0.7	2.75	10.93	44.03	174.04	700.9
Parallel 2-Cores Total Time	0.86	2.85	11.19	46.91	177.02	711.43
Parallel 2-Cores Elapsed Time	0.49	1.43	5.59	23.48	89.1	356.32

Concurrency: Infrastructure Based Approach

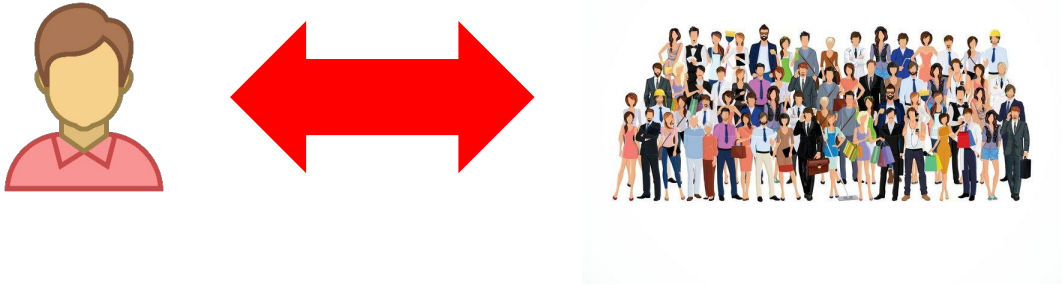


Concurrency: Infrastructure Based Approach



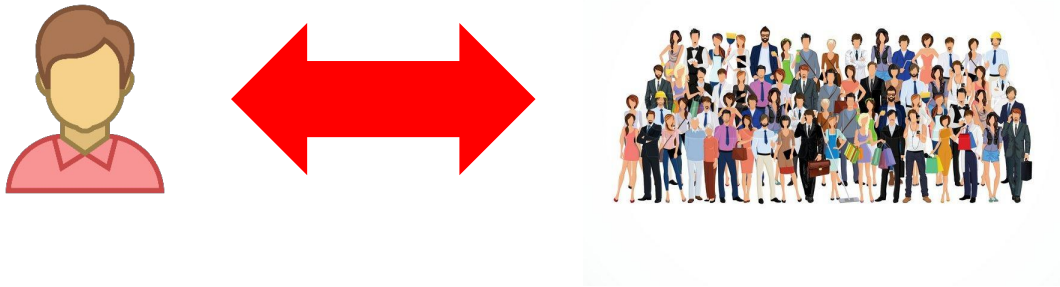
- As we can see, the number of operations still increase according to our expression **(K * ((n^2 - 2) / 2))**

Concurrency: Infrastructure Based Approach



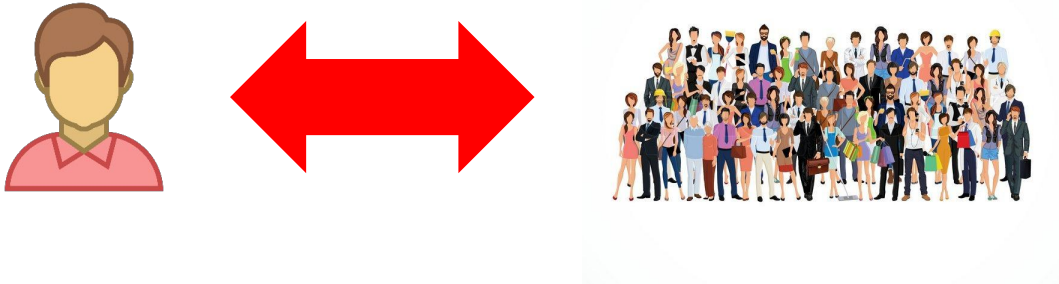
- As we can see, the number of operations still increase according to our expression **(K * ((n^2 - 2) / 2))**
- Thus, whether we use sequential or parallel 2-cores mode the total amount of computation time is nearly the same (see blue and red lines in the chart).

Concurrency: Infrastructure Based Approach



- As we can see, the number of operations still increase according to our expression **(K * ((n^2 - 2) / 2))**
- Thus, whether we use sequential or parallel 2-cores mode the total amount of computation time is nearly the same (see blue and red lines in the chart).
- However, by using parallel 2-cores we reduce the elapsed time w.r.t. to sequential mode to nearly the half (see blue and yellow lines in the chart).

Concurrency: Infrastructure Based Approach



- As we can see, the number of operations still increase according to our expression **(K * ((n^2 - 2) / 2))**
- Thus, whether we use sequential or parallel 2-cores mode the total amount of computation time is nearly the same (see blue and red lines in the chart).
- However, by using parallel 2-cores we reduce the elapsed time w.r.t. to sequential mode to nearly the half (see blue and yellow lines in the chart).
- This is because our **2 cores** balance the workload, and they perform their **(K * ((n^2 - 2) / 2)) / 2** operations in parallel.

Concurrency: Infrastructure Based Approach

We have seen the benefits
of using distributed programming.

Concurrency: Infrastructure Based Approach

We have seen the benefits
of using distributed programming.

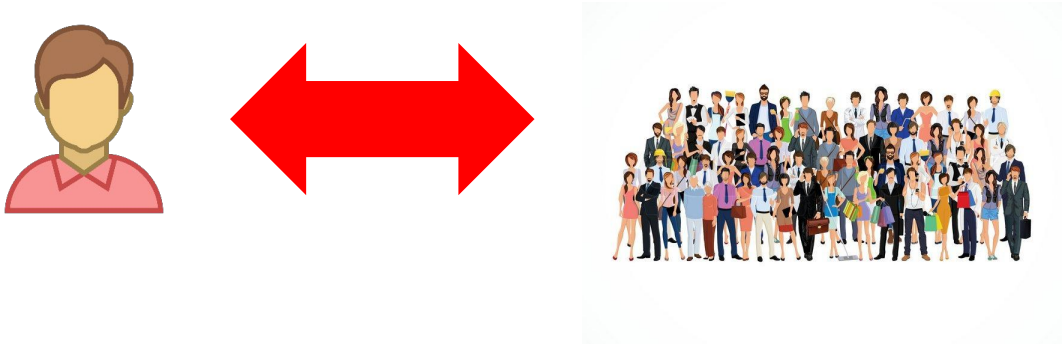
Let’s also see the drawbacks.

Concurrency: Infrastructure Based Approach

Drawback 1:

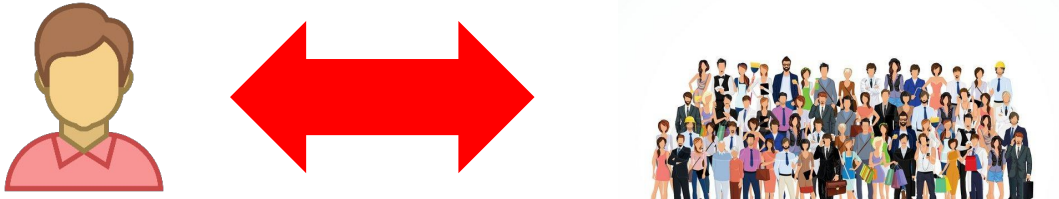
The speed-up gained with distributed programming is limited by the number of cores in our cluster.

Concurrency: Infrastructure Based Approach



- The code examples associated to this lecture provide us with a benchmark generator & solver for running the problem of computing list inversions of person P vs an entire population.

Concurrency: Infrastructure Based Approach



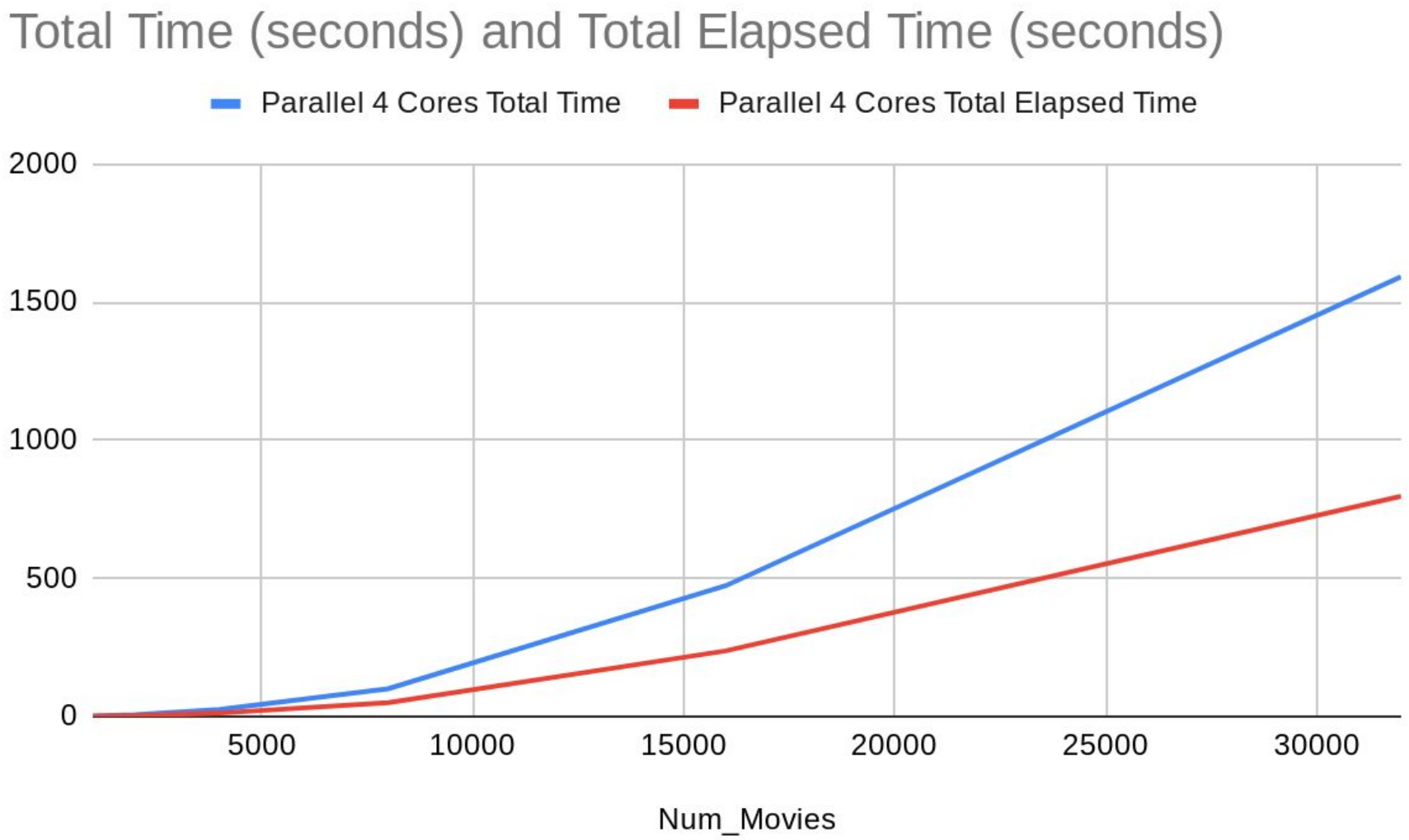
- The code examples associated to this lecture provide us with a benchmark generator & solver for running the problem of computing list inversions of person P vs an entire population.

- We are going to use this tool now to test the scalability of:
  - Compute List Inversions of P vs the population using:
    - The algorithm count_inversions of $O(n^2)$.
    - A parallel-based approach with 4 cores.
    - A fixed population size k = 12.
    - A number of movies ranging from 1,000 to 32,000.

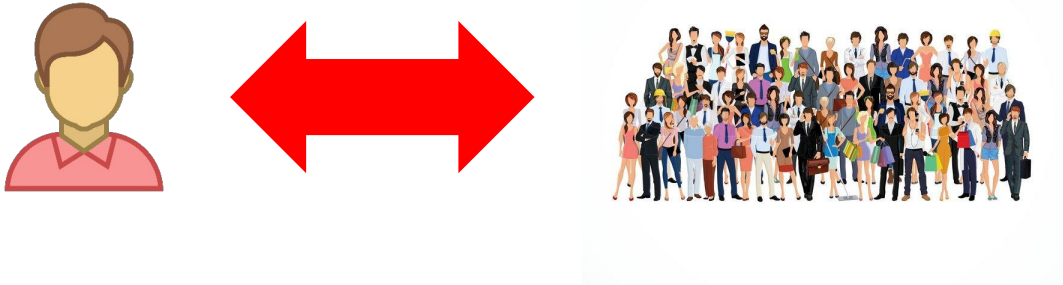
Concurrency: Infrastructure Based Approach

- The results are presented below:

Num_Movies	1000	2000	4000	8000	16000	32000
Parallel 4-cores Total Time	1.52	6.13	24.86	100.76	473.61	1591.8
Parallel 4-cores Elapsed Time	0.73	3.07	12.46	50.3	237.56	797.77

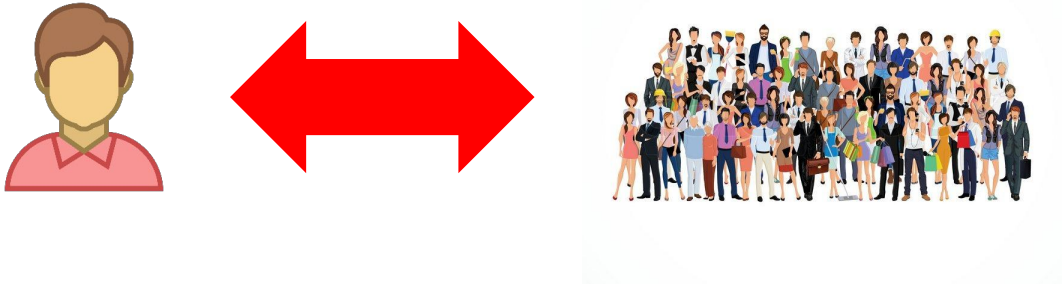

Total Time (seconds) and Total Elapsed Time (seconds)
Parallel 4 Cores Total Time
Parallel 4 Cores Total Elapsed Time
2000
1500
1000
500
0
5000
10000
15000
20000
25000
30000
Num_Movies

Concurrency: Infrastructure Based Approach



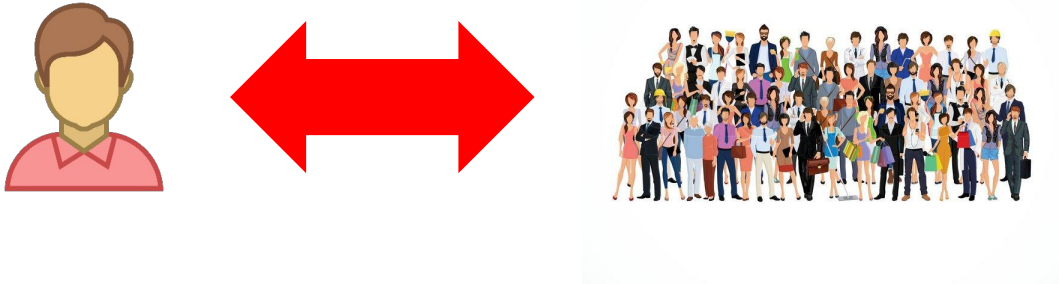
- As we can see, the number of operations still increase according to our expression **(K * ((n^2 - 2) / 2))**

Concurrency: Infrastructure Based Approach



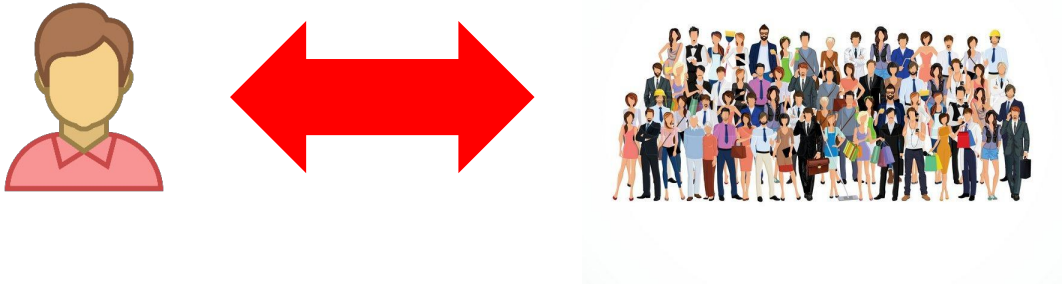
- As we can see, the number of operations still increase according to our expression **(K * ((n^2 - 2) / 2))**
- The tendency of elapsed time being reduced w.r.t. total time by using parallel processes obviously still happens, as the workload is balanced among the different cores.

Concurrency: Infrastructure Based Approach



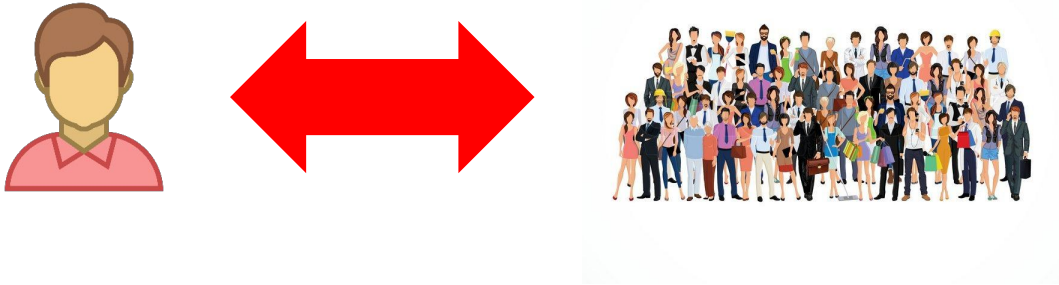
- As we can see, the number of operations still increase according to our expression **(K * ((n^2 - 2) / 2))**
- The tendency of elapsed time being reduced w.r.t. total time by using parallel processes obviously still happens, as the workload is balanced among the different cores.
- However, we might had expected the elapsed time to be **¼** of the total time, as we are balancing the load among 4 cores. But, as we can see, the elapsed time is only reduced by ½.

Concurrency: Infrastructure Based Approach



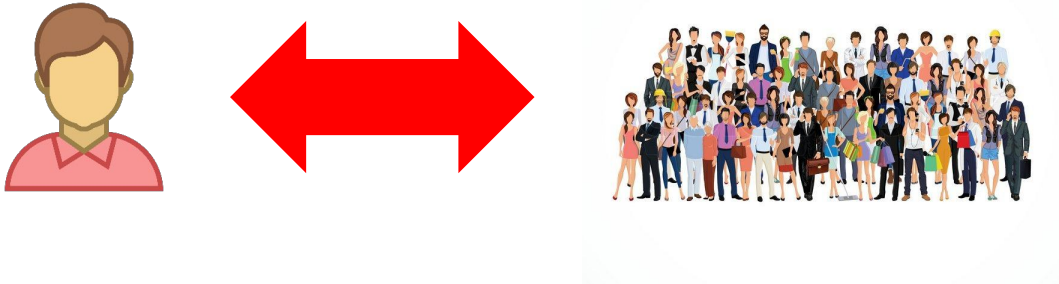
- The reason for this is simple: my computer only has 2 cores!

Concurrency: Infrastructure Based Approach



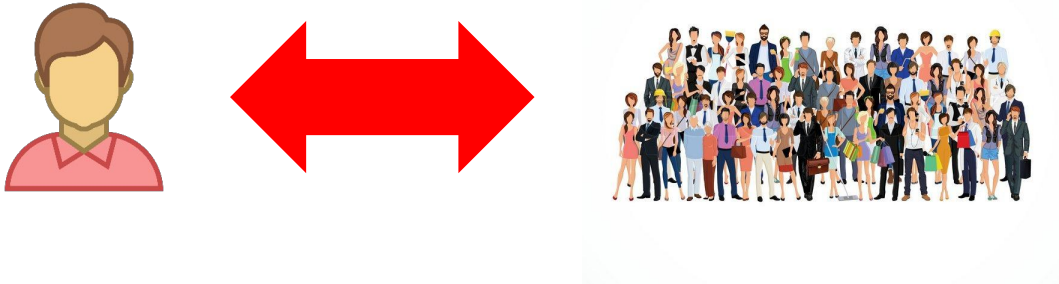
- The reason for this is simple: my computer only has 2 cores!
- Thus, if we split the workload over 4 processes, 2 processes are indeed assigned to each core. And these 2 processes start competing for the CPU time!

Concurrency: Infrastructure Based Approach



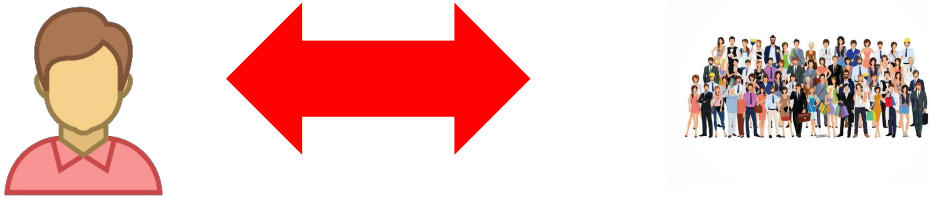
- The reason for this is simple: my computer only has 2 cores!
- Thus, if we split the workload over 4 processes, 2 processes are indeed assigned to each core. And these 2 processes start competing for the CPU time!
- As a matter of being fair (and polite) the OS multiplex the CPU time between the 2 processes.

Concurrency: Infrastructure Based Approach



- The reason for this is simple: my computer only has 2 cores!
- Thus, if we split the workload over 4 processes, 2 processes are indeed assigned to each core. And these 2 processes start competing for the CPU time!
- As a matter of being fair (and polite) the OS multiplex the CPU time between the 2 processes.
- But this only makes the things worse, and a big overhead time is needed to swap from process and process, ending up in taking way too longer to solve the subproblems.

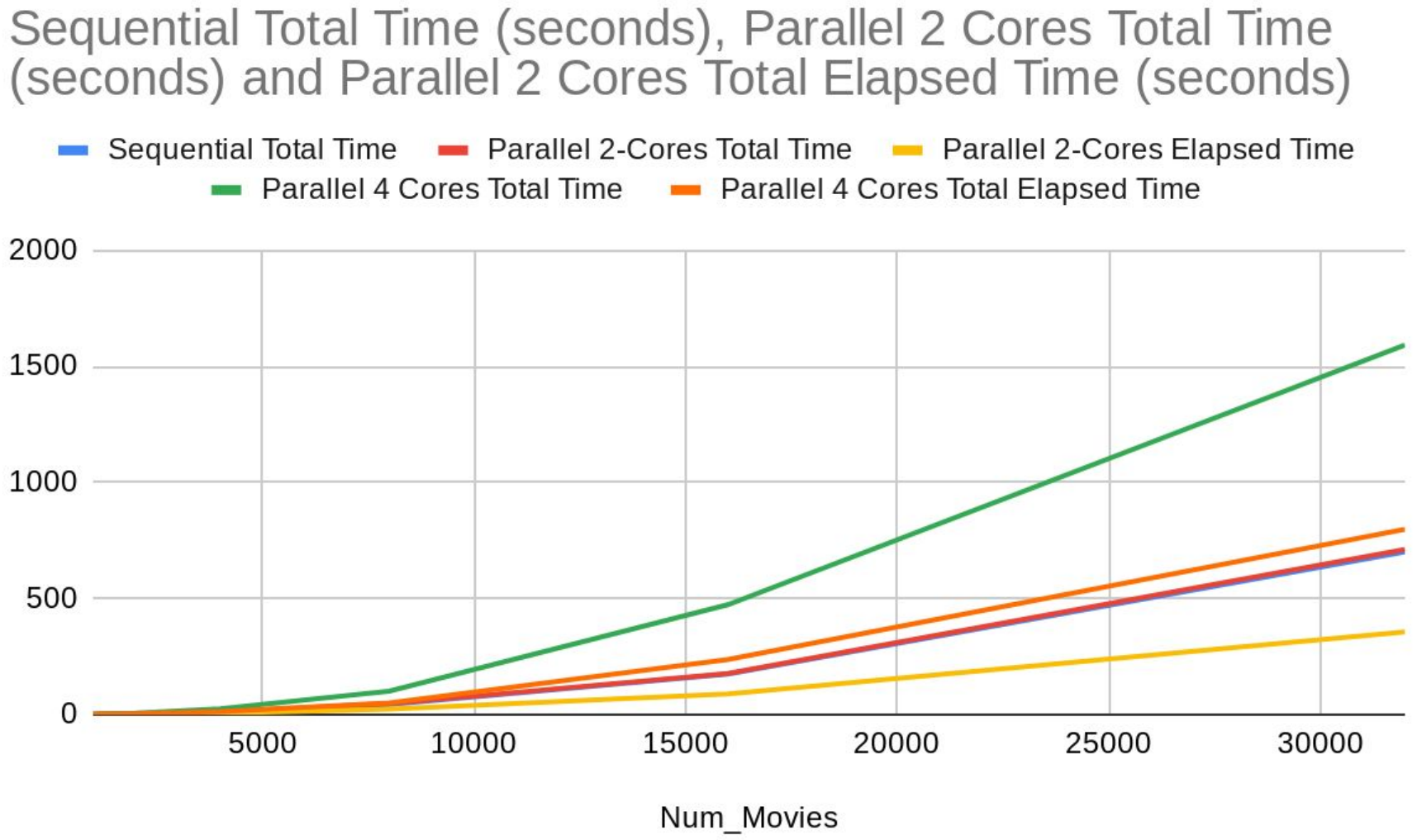
Concurrency: Infrastructure Based Approach



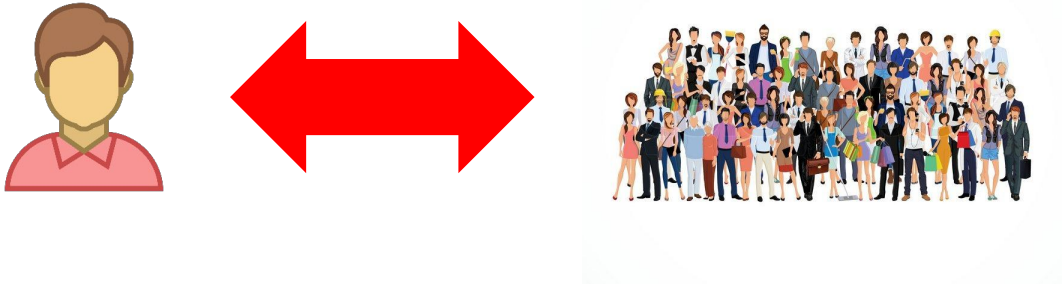
- When we compare sequential, distributed (2 cores) and distributed (4 cores) we see all of this more clear:

Num_Movies	1000	2000	4000	8000	16000	32000
Sequential Total Time	0.7	2.75	10.93	44.03	174.04	700.9
Parallel 2-Cores Total Time	0.86	2.85	11.19	46.91	177.02	711.43
Parallel 2-Cores Elapsed Time	0.49	1.43	5.59	23.48	89.1	356.32
Parallel 4-Cores Total Time	1.52	6.13	24.86	100.76	473.61	1591.8
Parallel 4-Cores Elapsed Time	0.73	3.07	12.46	50.3	237.56	797.77

Concurrency: Infrastructure Based Approach

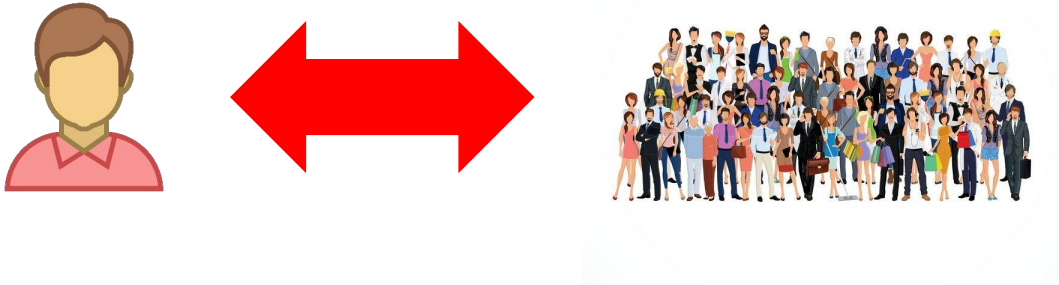


Concurrency: Infrastructure Based Approach



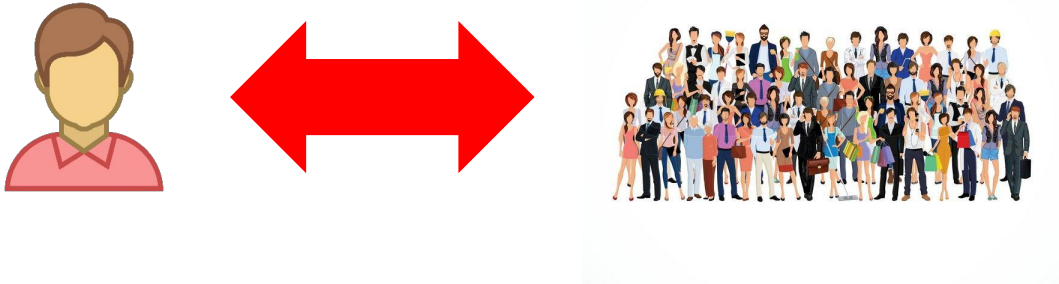
- As we can see, the total time with parallel 4 cores is longer than the sequential total time (see lines green and blue).

Concurrency: Infrastructure Based Approach



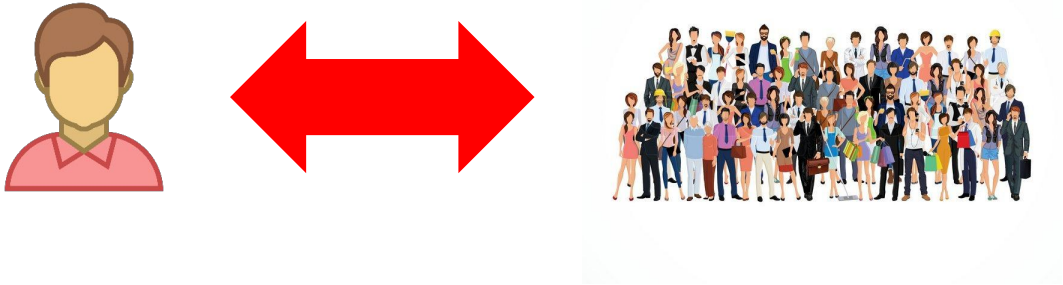
- As we can see, the total time with parallel 4 cores is longer than the sequential total time (see lines green and blue).
- This is not that terrible, as it also happened with the total time with parallel 2 cores vs. sequential mode (see lines red and blue).

Concurrency: Infrastructure Based Approach



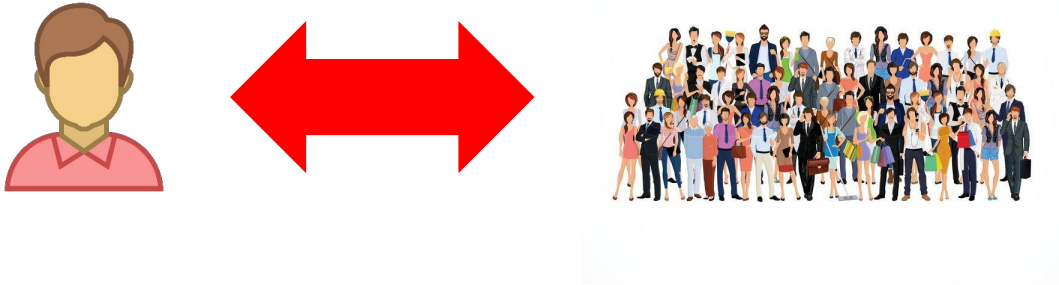
- As we can see, the total time with parallel 4 cores is longer than the sequential total time (see lines green and blue).
- This is not that terrible, as it also happened with the total time with parallel 2 cores vs. sequential mode (see lines red and blue).
- That being said, the tendency gets worse. Definitely, total time of parallel 4 cores is much worse that total time of parallel 2 nodes (see lines green and red).

Concurrency: Infrastructure Based Approach



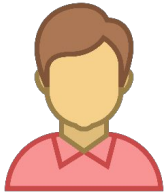
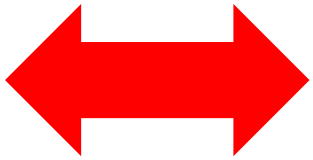
- But the things are getting even worse!
  Parallel 4-cores elapsed time is even worse than sequential total time (see lines orange and blue).

Concurrency: Infrastructure Based Approach



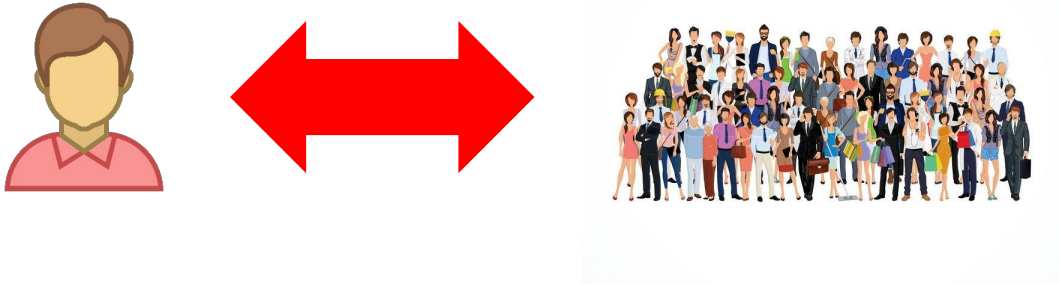
- But the things are getting even worse!
  Parallel 4-cores elapsed time is even worse than sequential total time (see lines orange and blue).
- And this definitely did not happen with parallel 2-cores elapsed time and sequential total time (see lines yellow and blue).

Concurrency: Infrastructure Based Approach

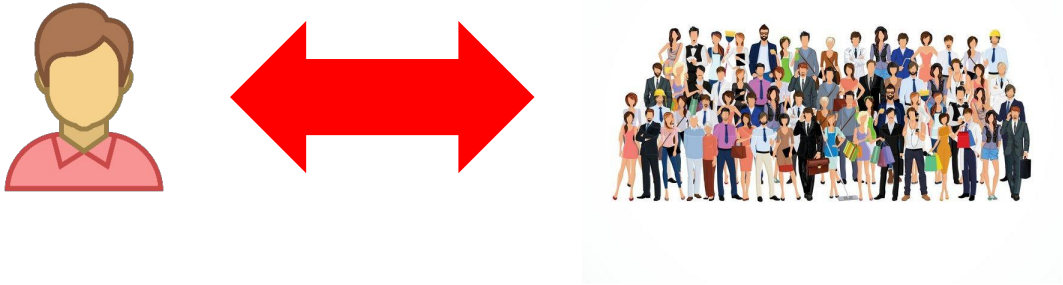

- So, all in all, we can see a tendency here…
- If my cluster has c cores:

Concurrency: Infrastructure Based Approach



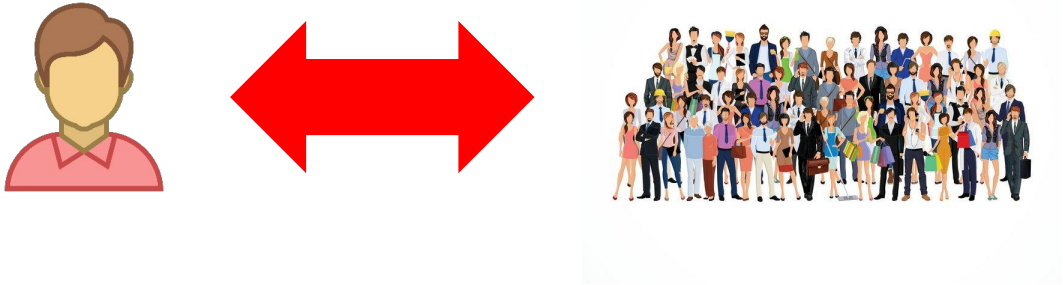
- So, all in all, we can see a tendency here...
- If my cluster has c cores:
  - Then distributing the workload among c cores will improve the sequential mode.

Concurrency: Infrastructure Based Approach



- So, all in all, we can see a tendency here...
- If my cluster has c cores:
  - Then distributing the workload among c cores will improve the sequential mode.
  - After that, distributing the workload among more than c cores will only make performance go down, due to multiplexing of CPU among different processes.

Concurrency: Infrastructure Based Approach



All in all, once again...

The benefits of distributed programming is limited by the number of cores in your cluster!

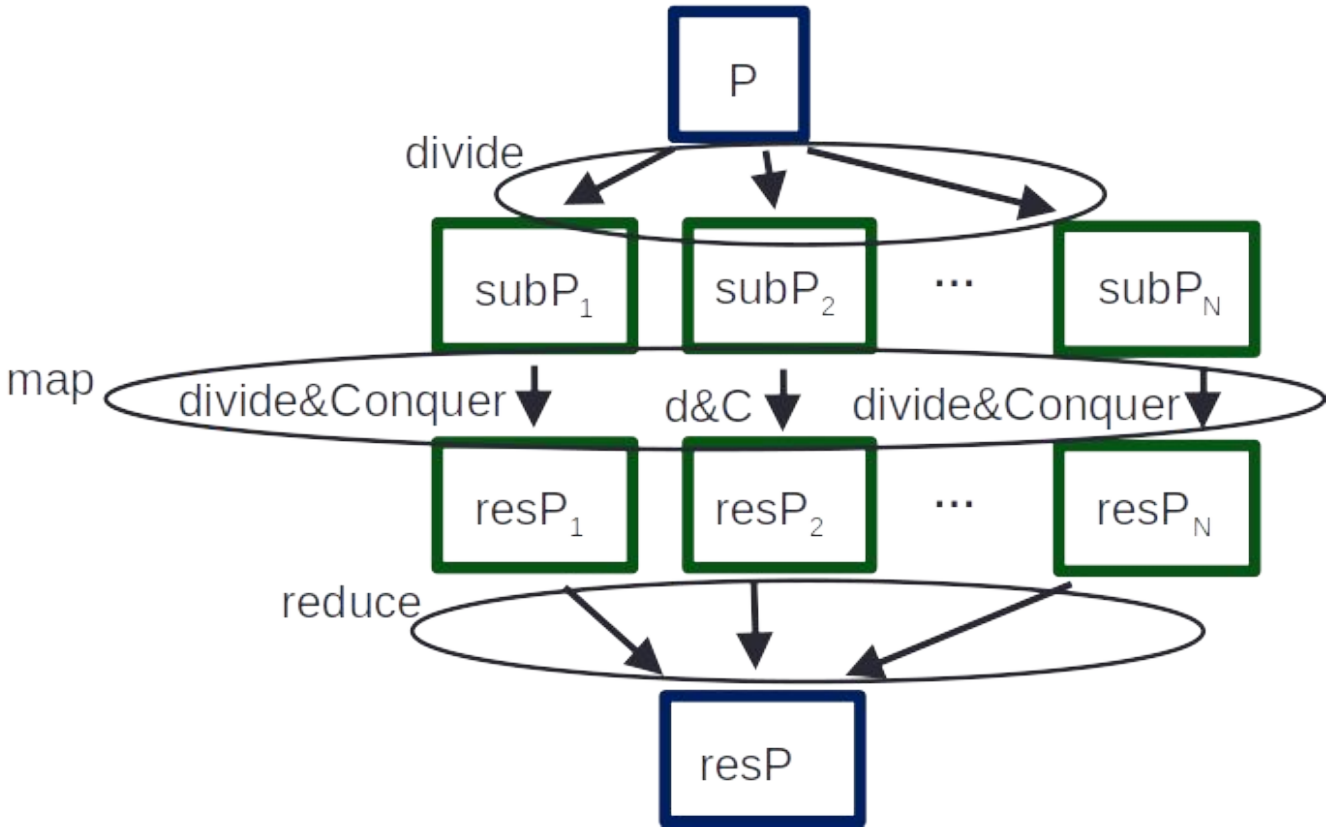
Concurrency: Infrastructure Based Approach

Drawback 2:

Passing from sequential to distributed mode
requires the use of a meta-algorithm,
in charge of controlling the distributed execution.

Concurrency: Infrastructure-Based Approach

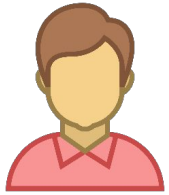
This meta-algorithm follows the **divide-map-reduce** approach of the classical Divide and Conquer programming paradigm!

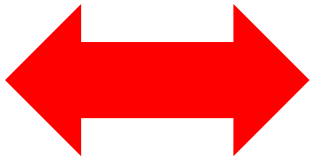


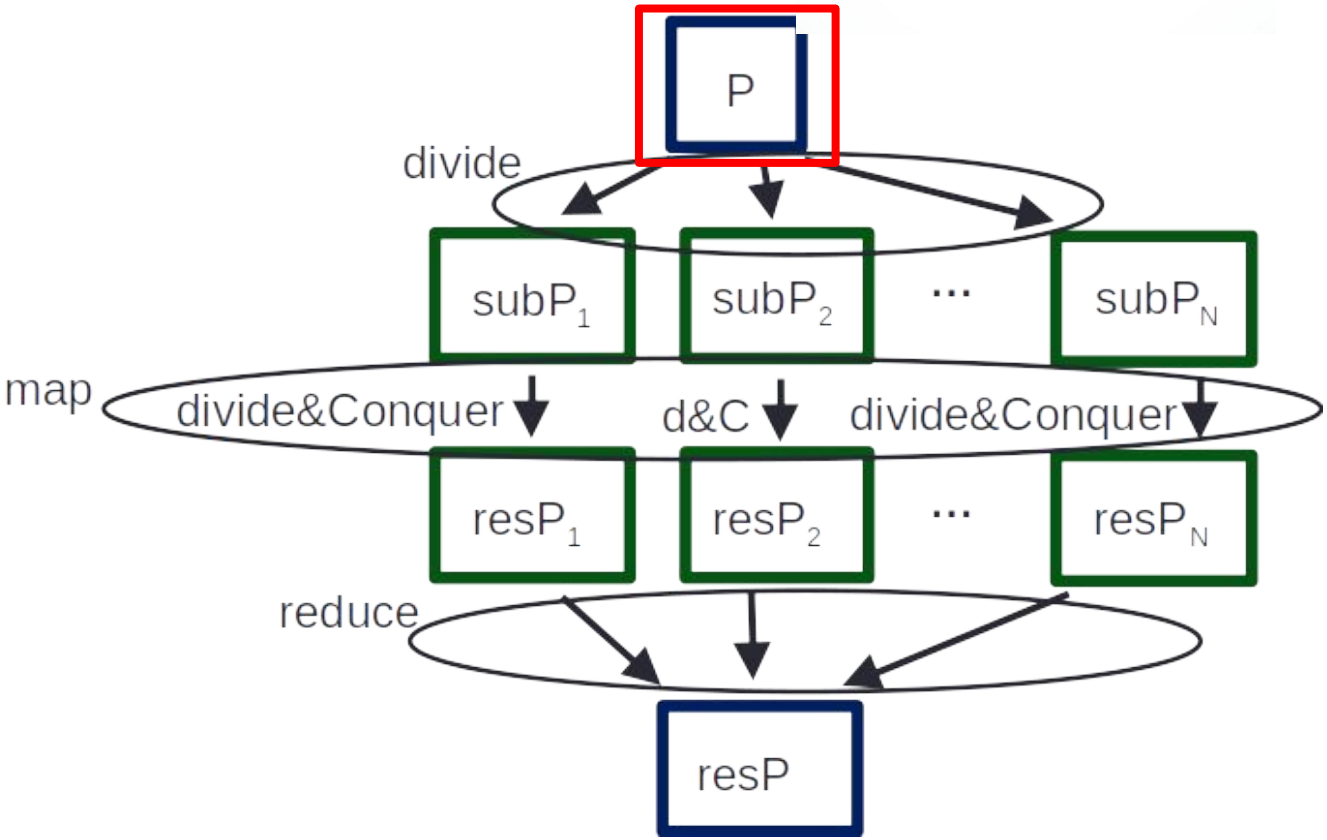
Concurrency: Infrastructure-Based Approach

Given an original problem P...

(in our case running list inversions of P against an entire population...)

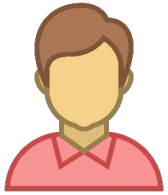


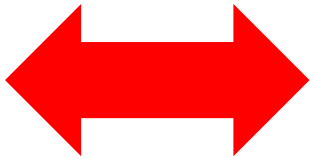


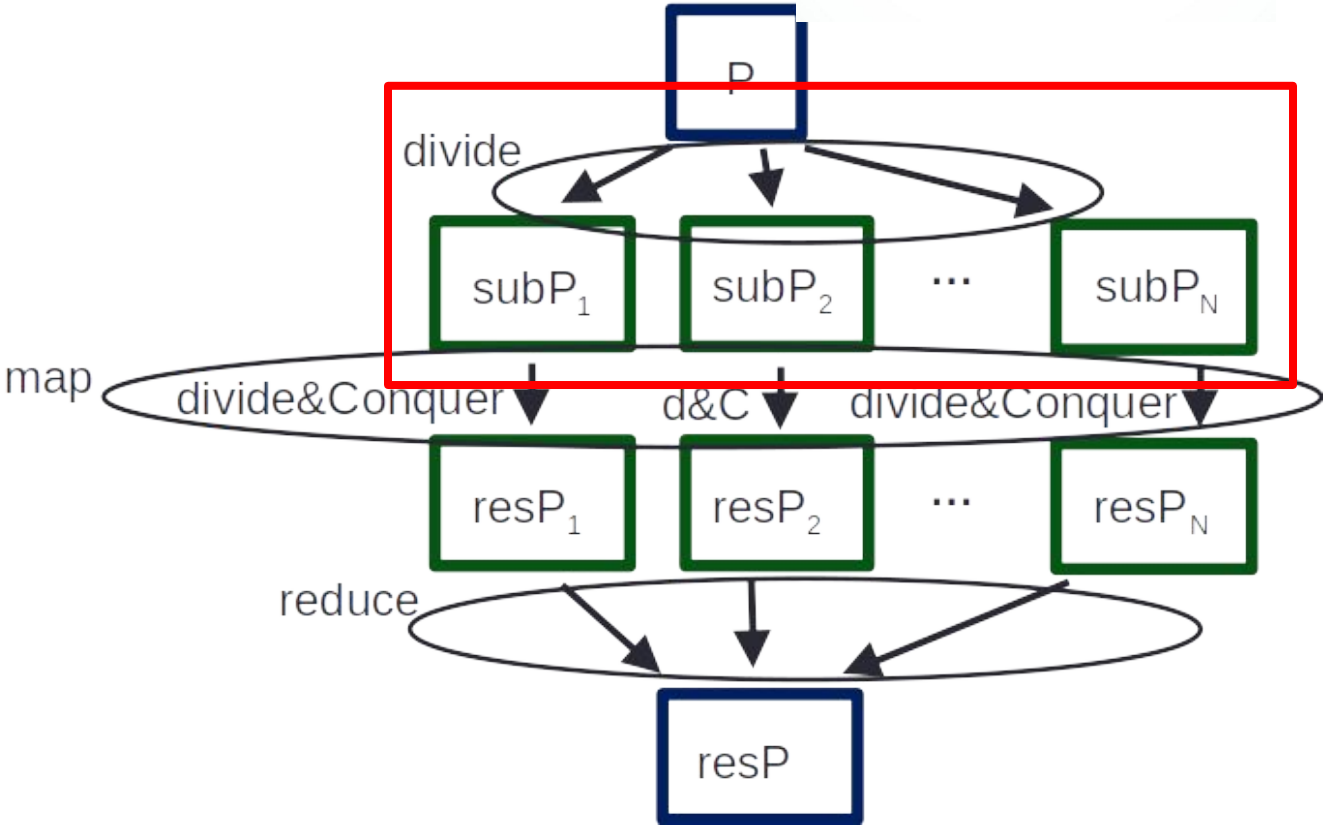


Concurrency: Infrastructure-Based Approach

1. **DIVIDE STAGE.**

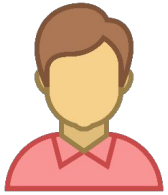


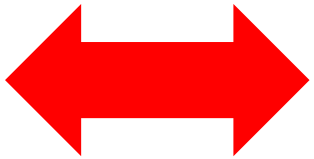




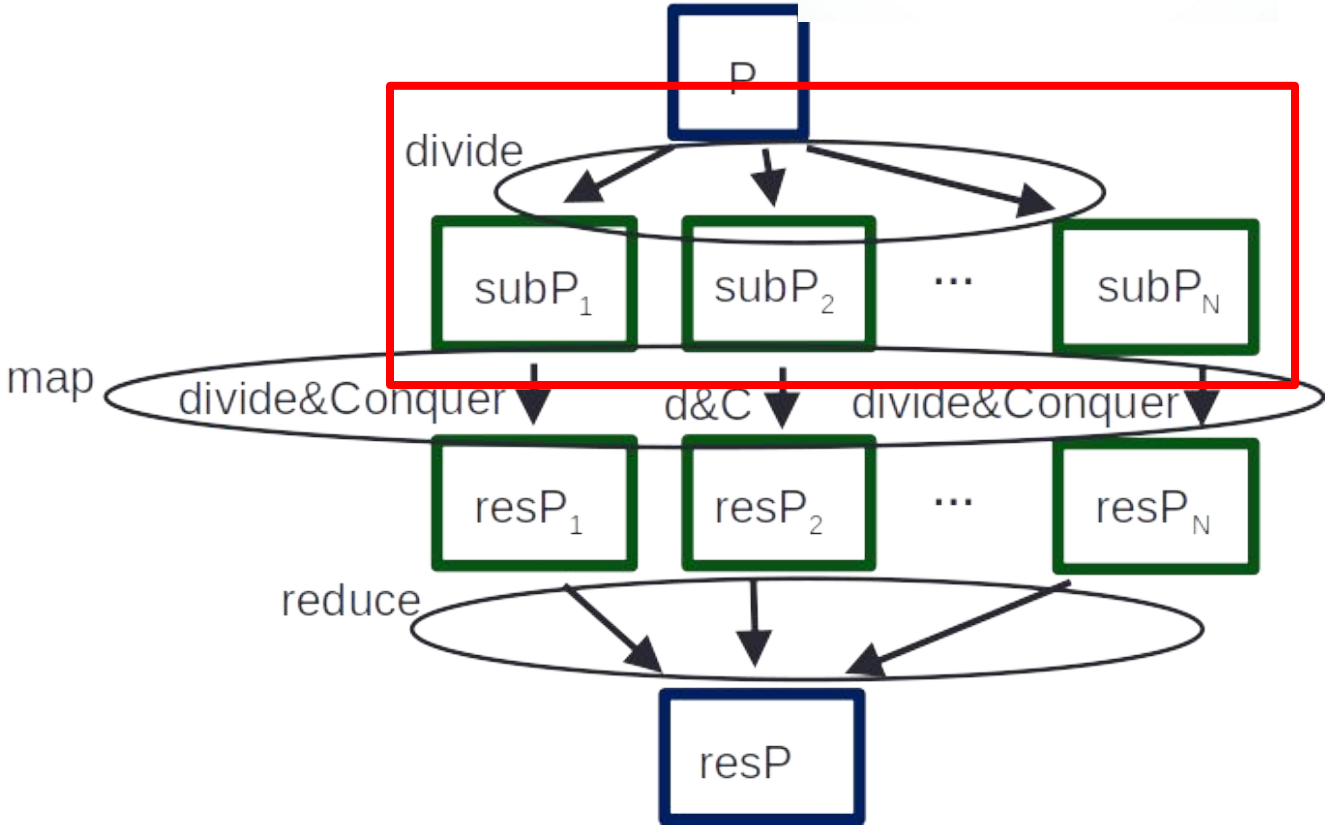
Concurrency: Infrastructure-Based Approach

1. **DIVIDE STAGE.**

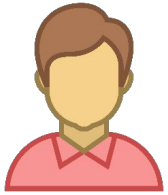




We divide the original problem by splitting it into *n* sub-parts.



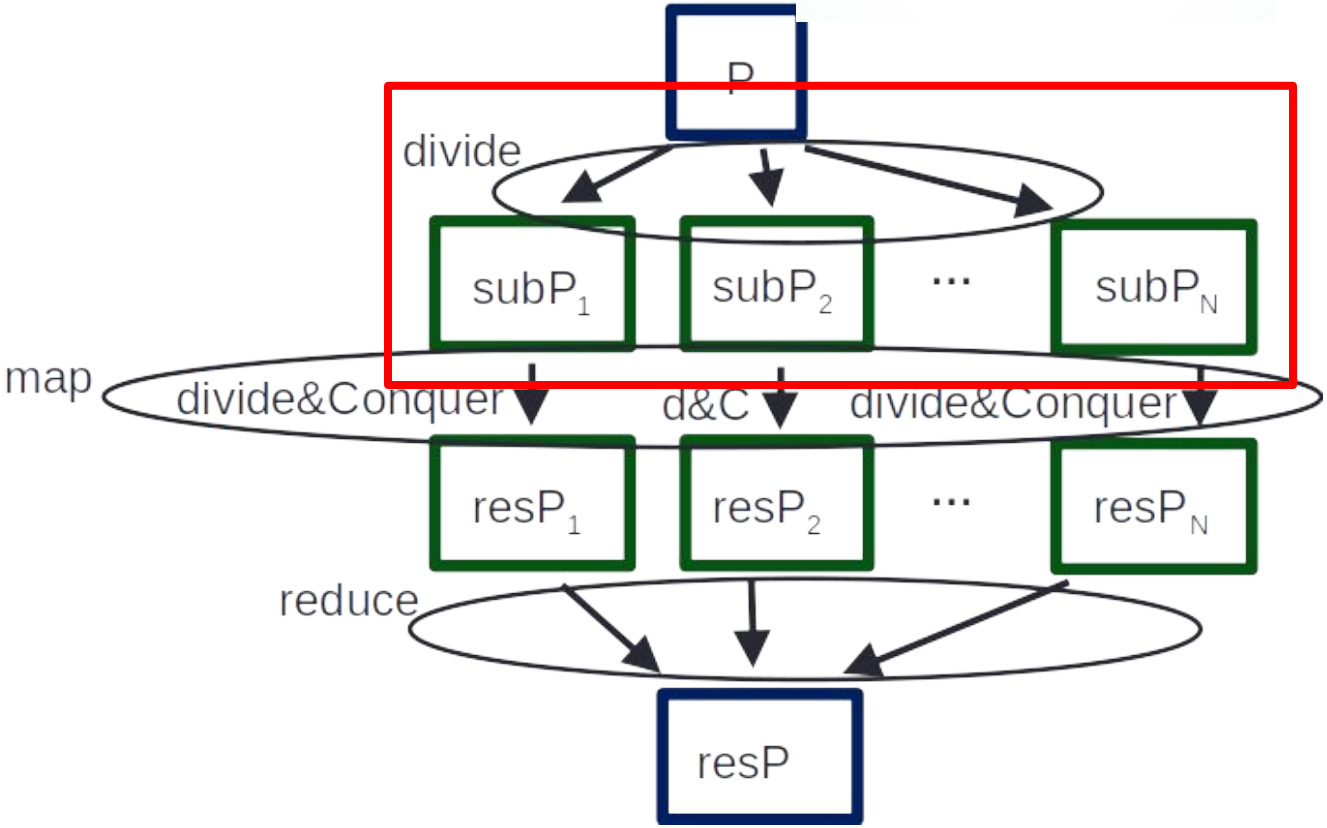
Concurrency: Infrastructure-Based Approach



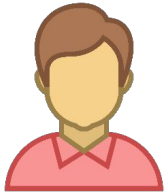
1. **DIVIDE STAGE.**

We divide the original problem by splitting it into *n* sub-parts.

The n sub-parts together represent the same problem as our original problem P.



Concurrency: Infrastructure-Based Approach

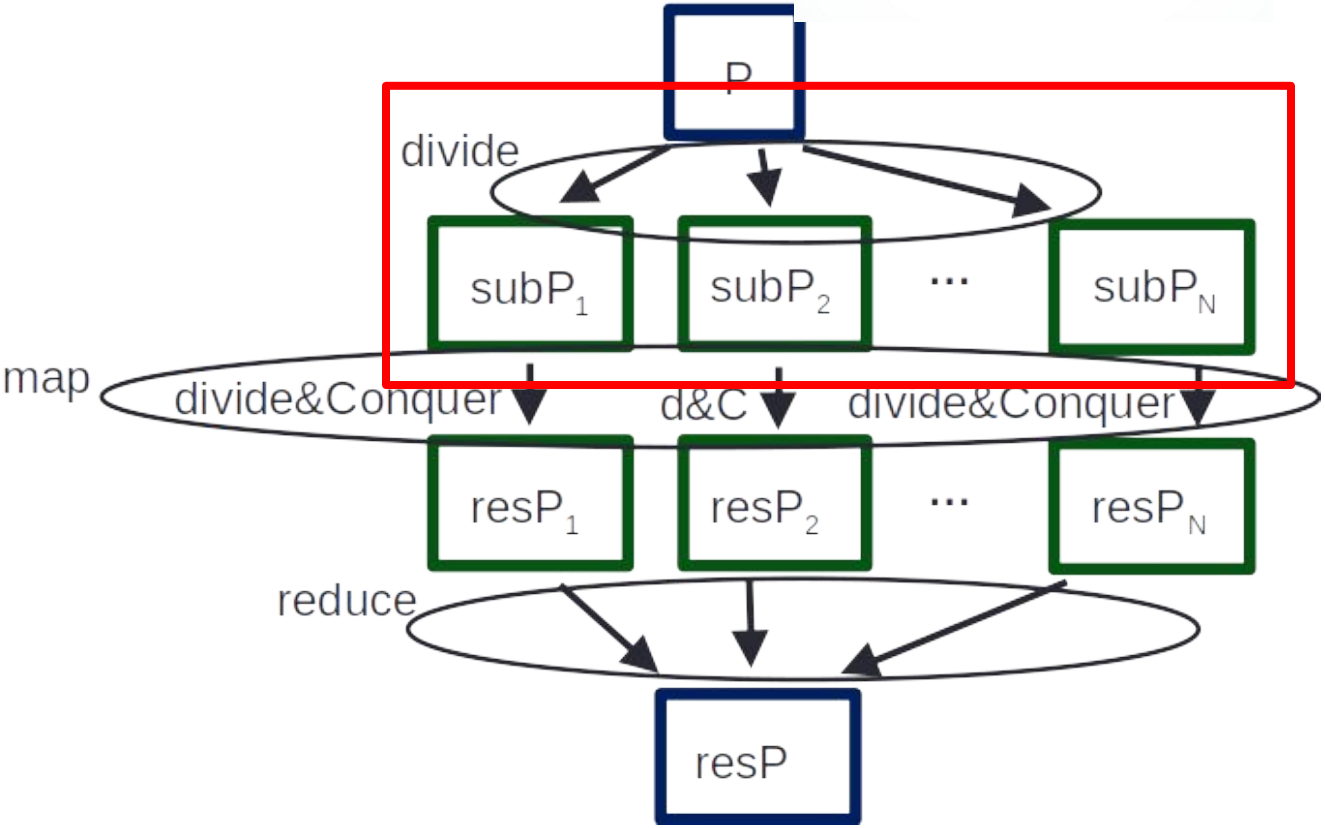


1. **DIVIDE STAGE.**

We divide the original problem by splitting it into *n* sub-parts.

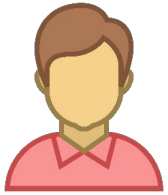
The n sub-parts together represent the same problem as our original problem P.

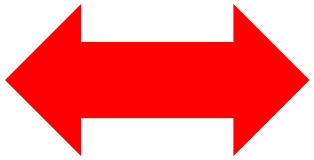
However, each sub-part Pi is strictly smaller than P.



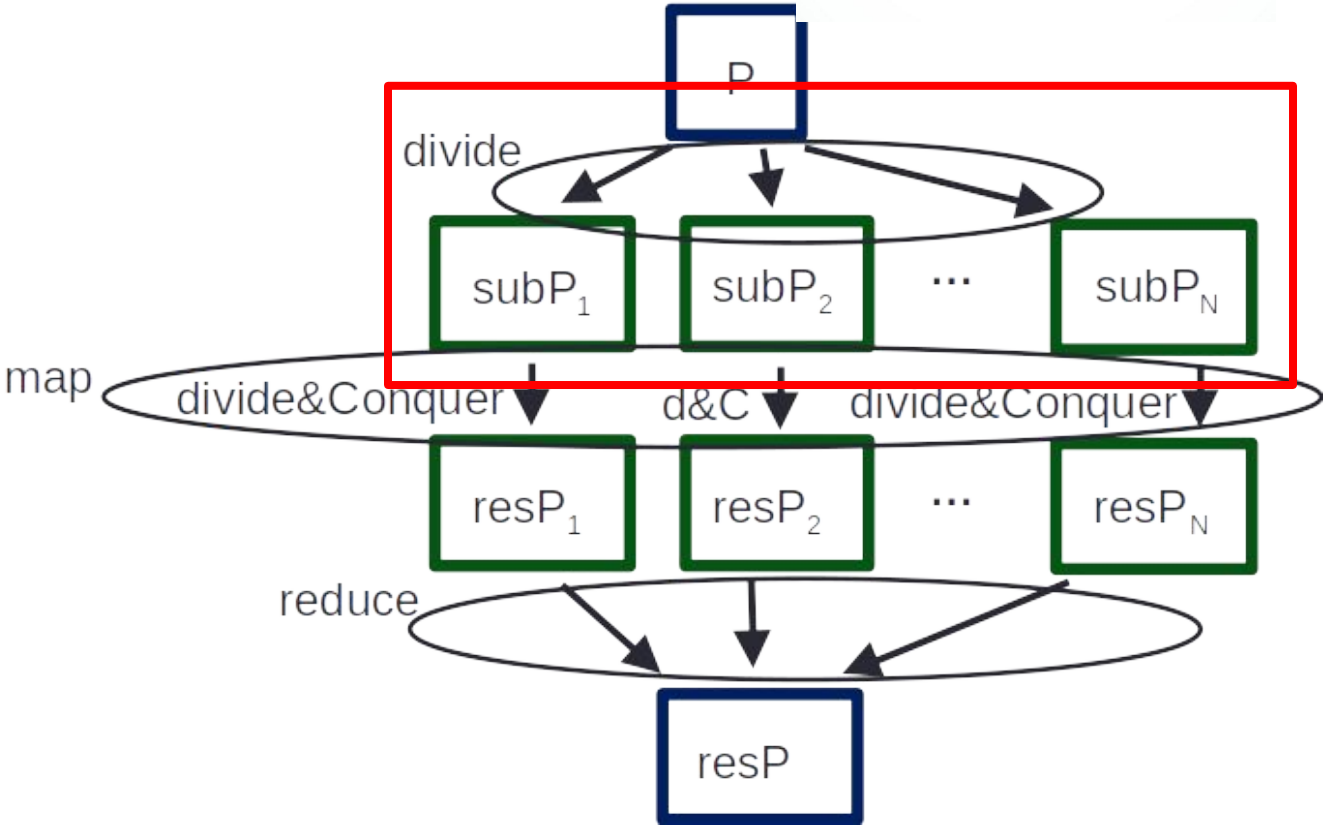
Concurrency: Infrastructure-Based Approach

1. **DIVIDE STAGE.**



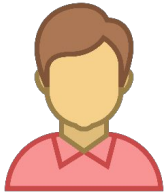


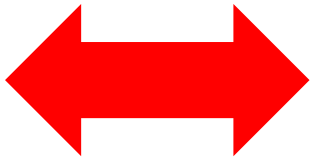
...in our case for this lecture we will need our problem-specific **divide** stage algorithm.



Concurrency: Infrastructure-Based Approach

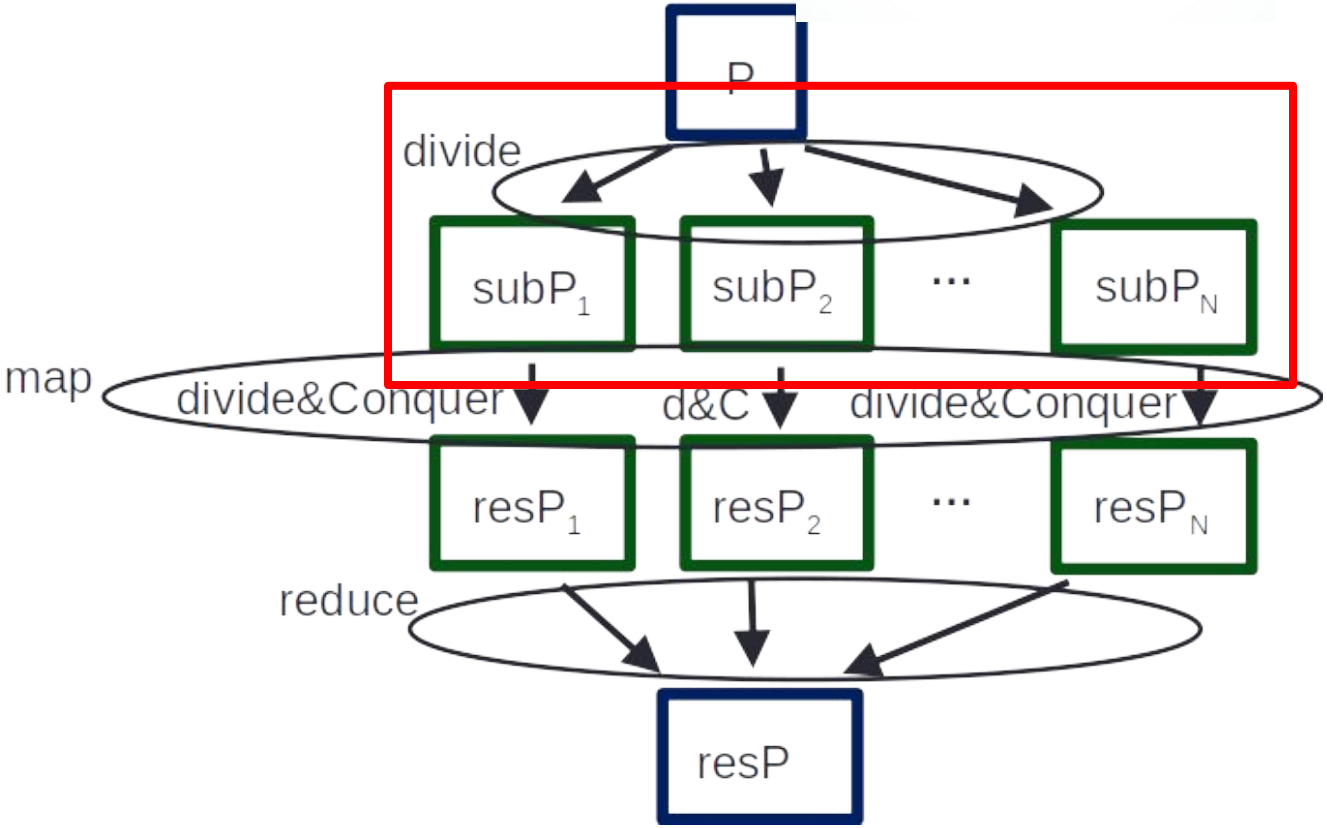
1. **DIVIDE STAGE.**





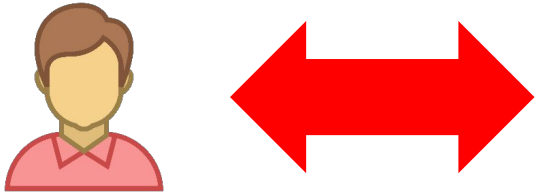
...in our case for this lecture we will need our problem-specific **divide** stage algorithm.

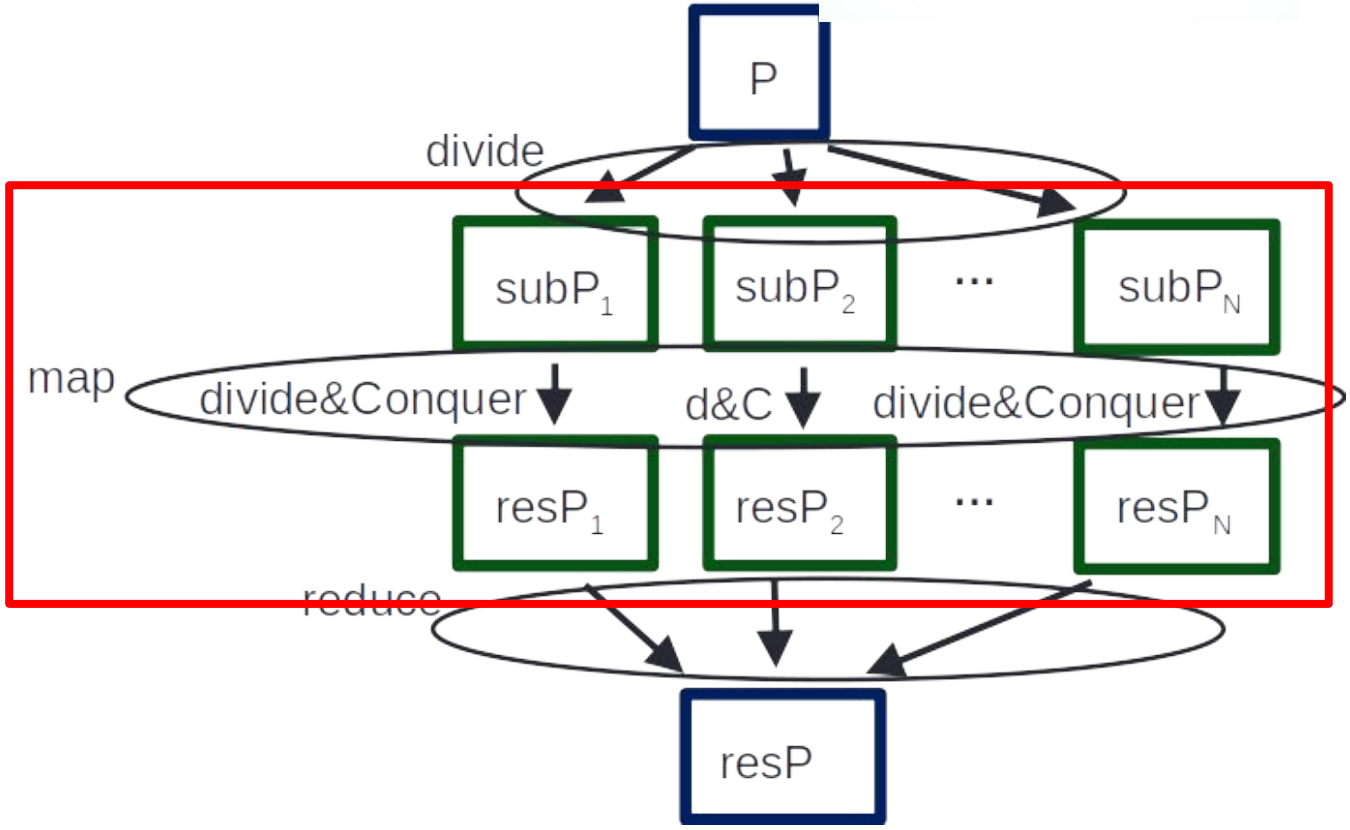
This algorithm just splits the population into equally sized people-subsets.



Concurrency: Infrastructure-Based Approach

2. MAP STAGE.

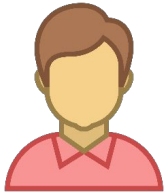


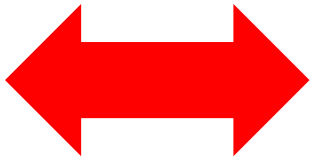


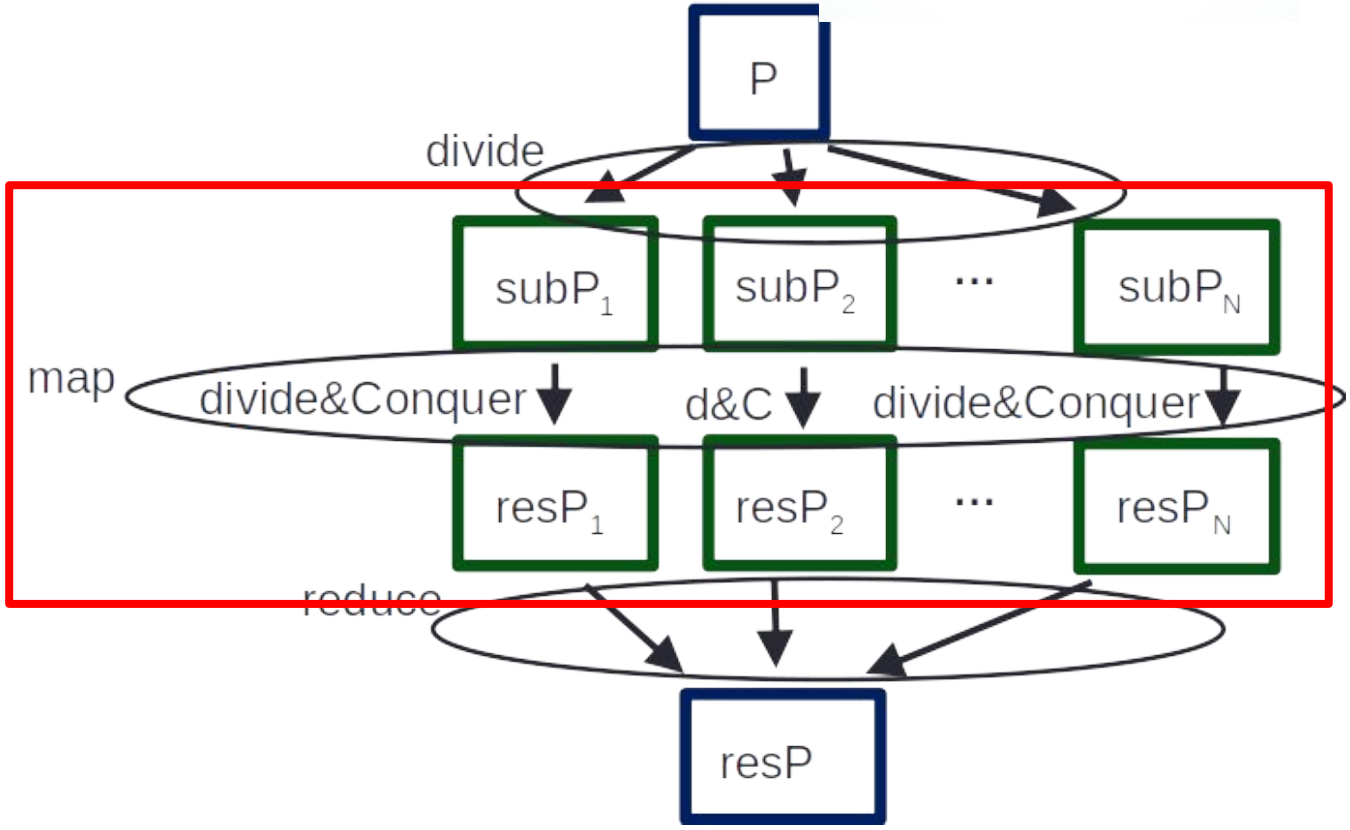
Concurrency: Infrastructure-Based Approach

2. MAP STAGE.

We solve each sub-part separately.

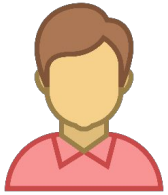


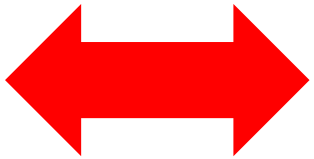




Concurrency: Infrastructure-Based Approach

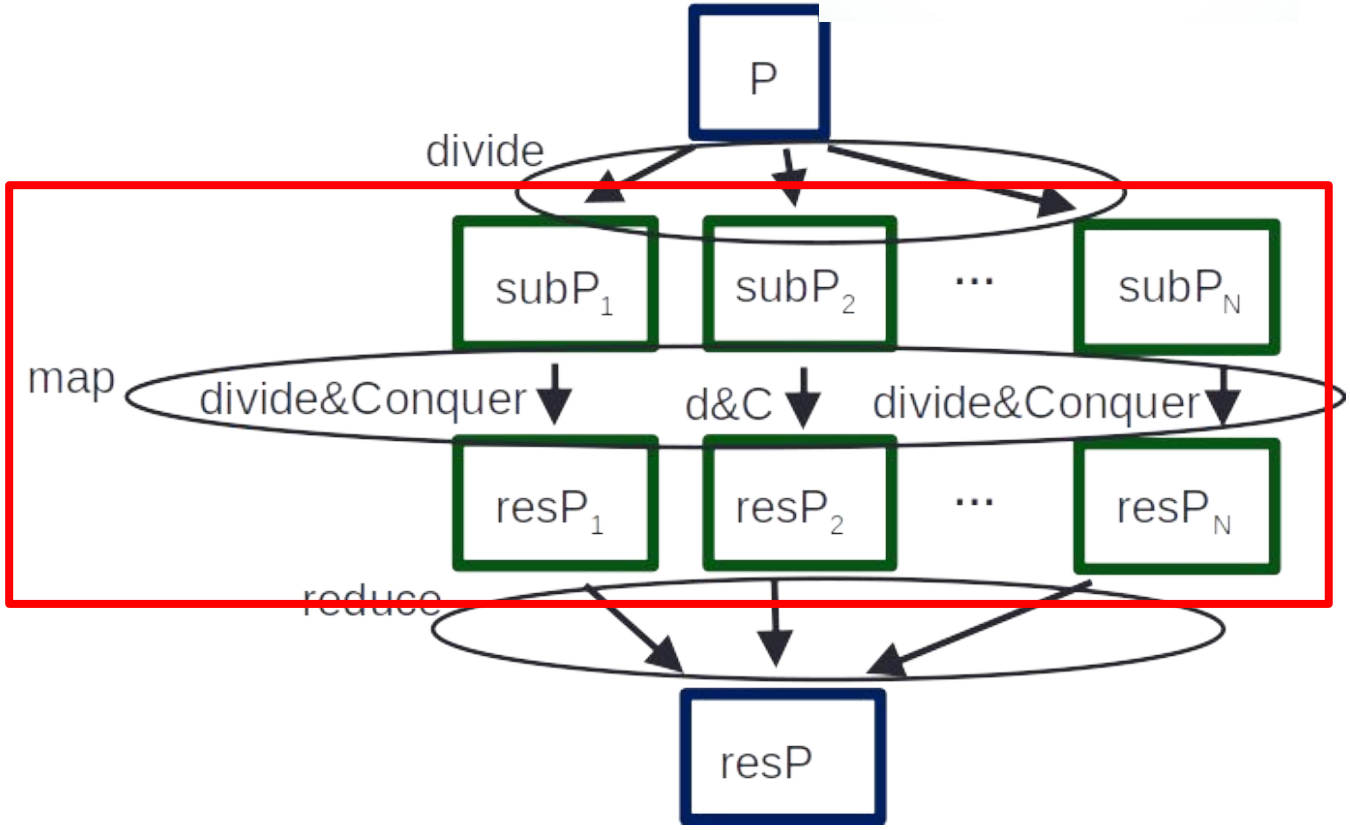
2. MAP STAGE.





We solve each sub-part separately.

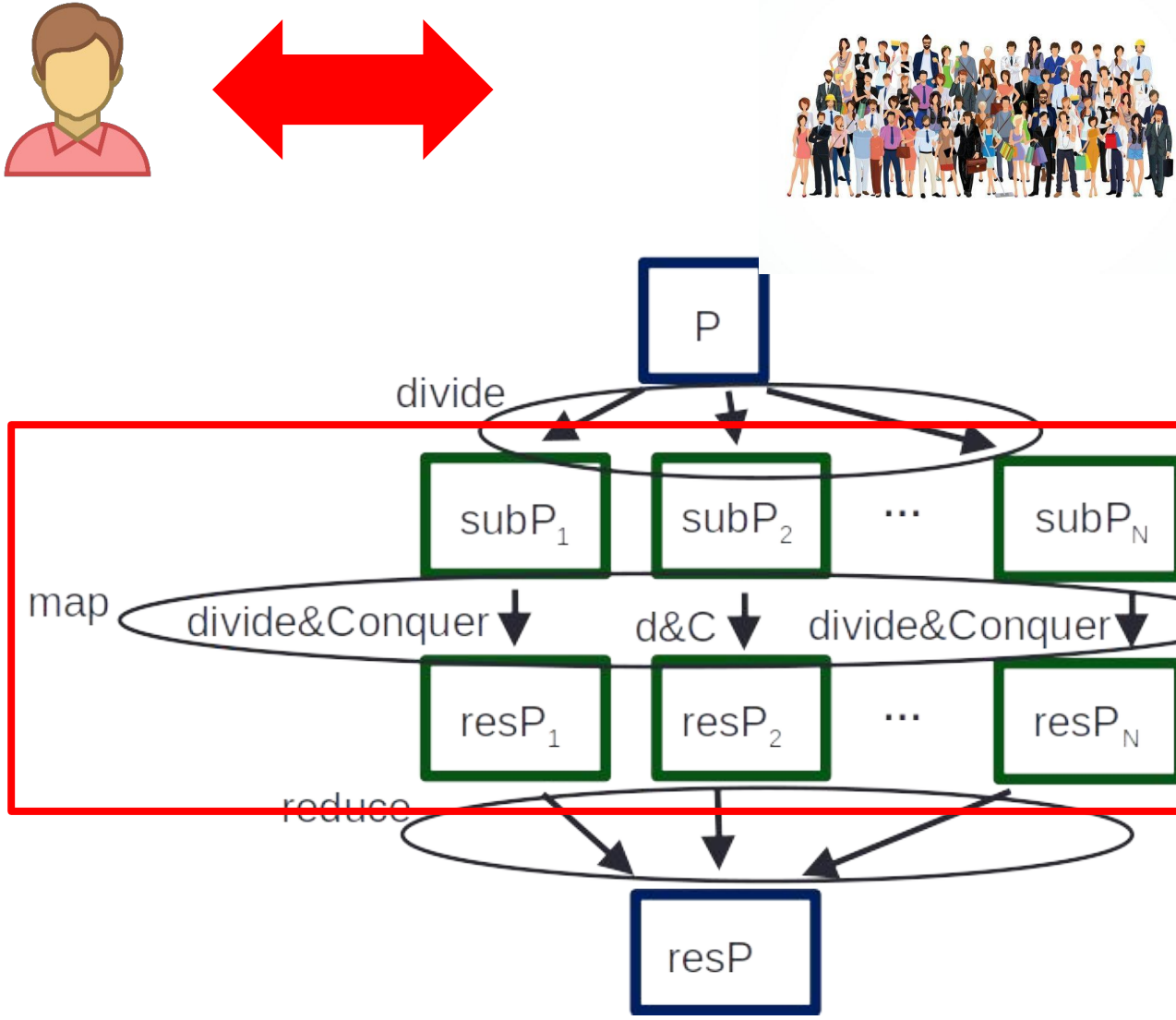
This ends up creating n processes, each of them assigned to a core and responsible for solving the sub-part.



Concurrency: Infrastructure-Based Approach

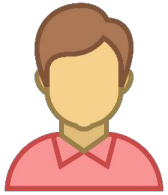
2. MAP STAGE.

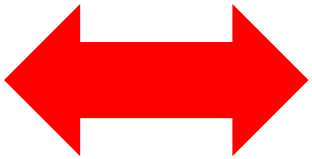
The meta-algorithm must supervise the execution of the processes.



Concurrency: Infrastructure-Based Approach

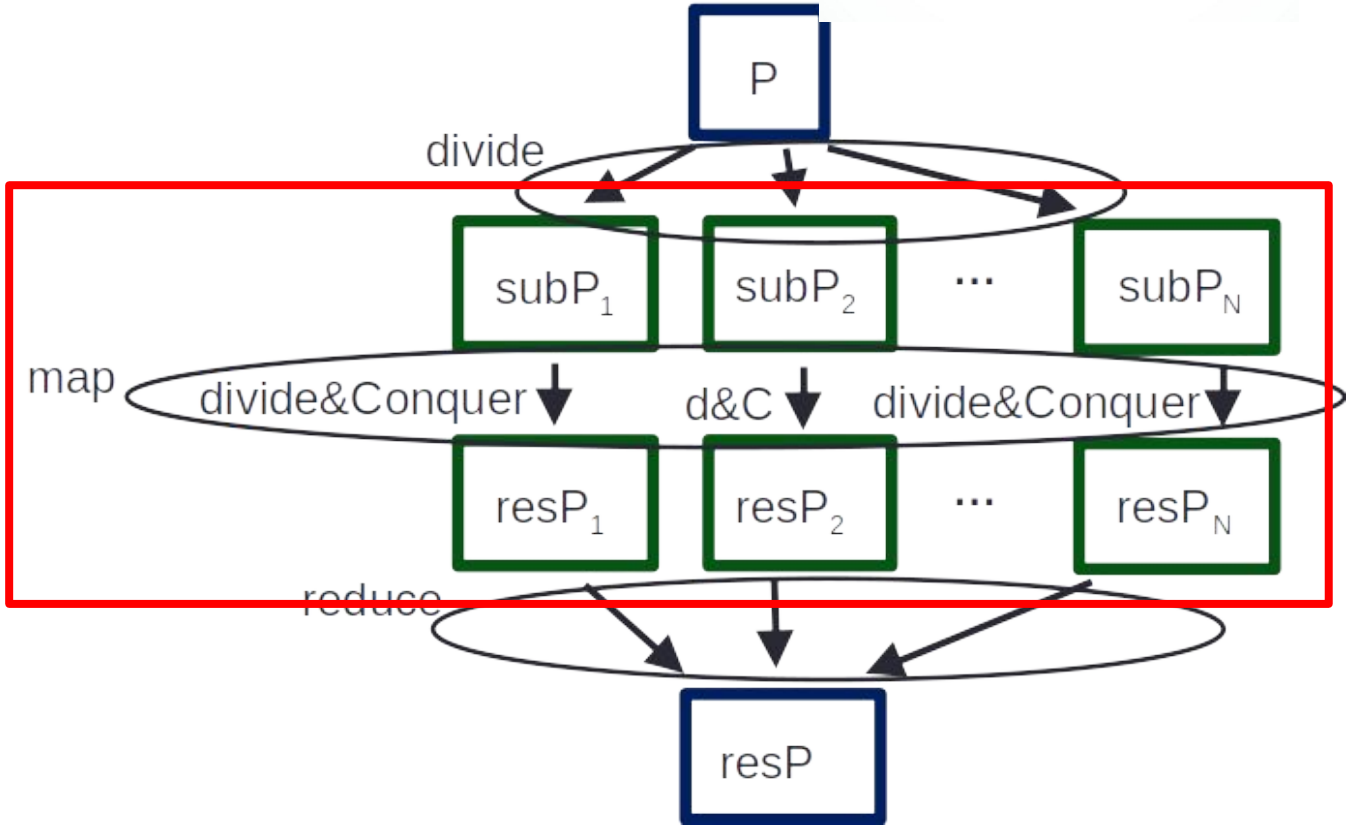
2. MAP STAGE.





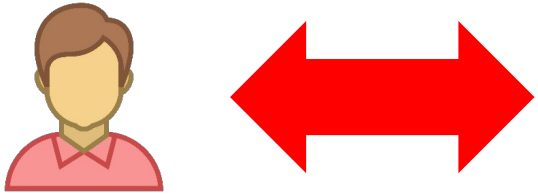
The meta-algorithm must supervise the execution of the processes.

In particular:



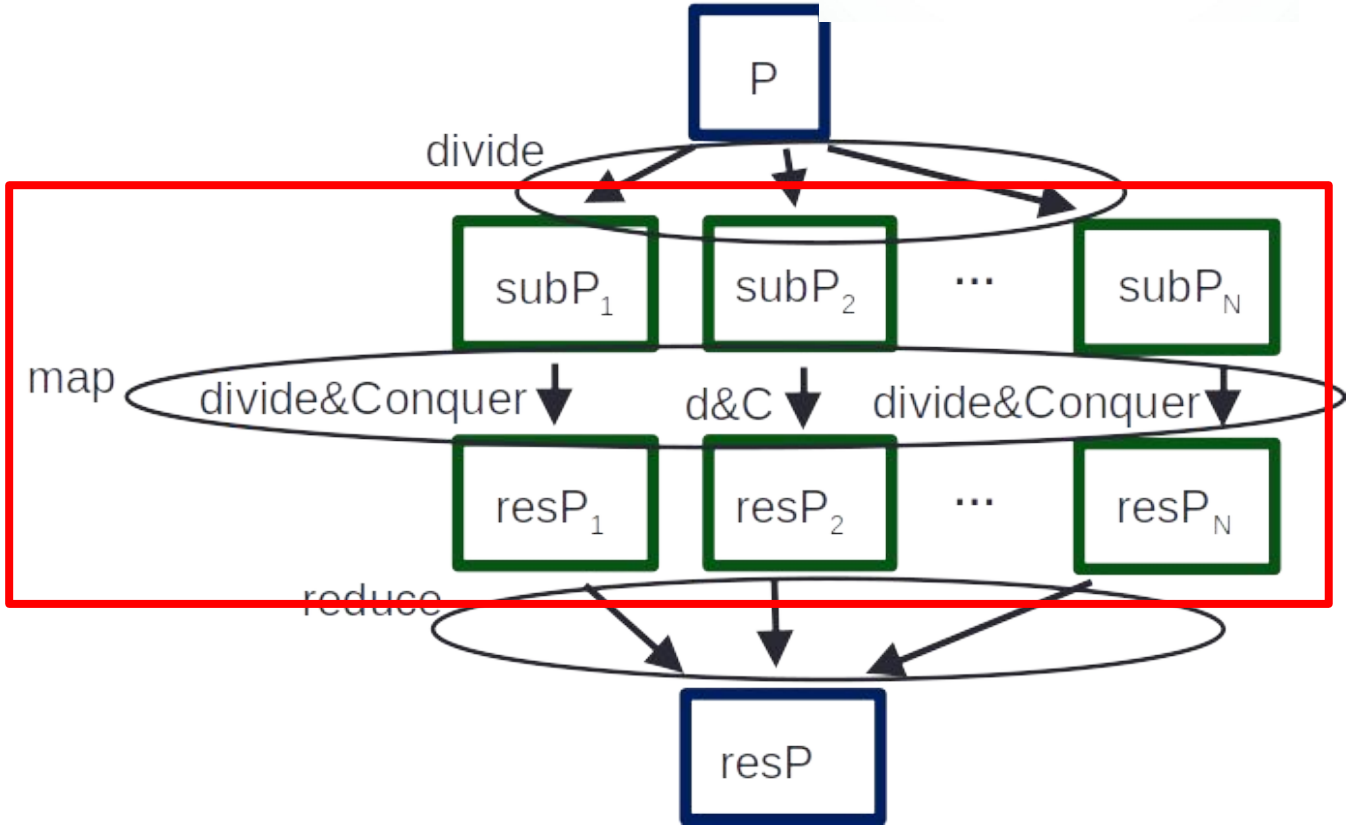
Concurrency: Infrastructure-Based Approach

2. MAP STAGE.

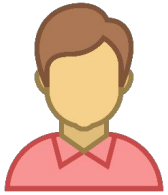


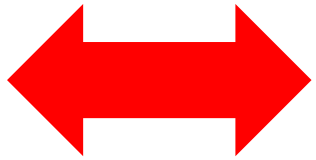
The meta-algorithm must supervise the execution of the processes.

In particular:
- It provides a scheduler, to decide which core each process is assigned to.



Concurrency: Infrastructure-Based Approach

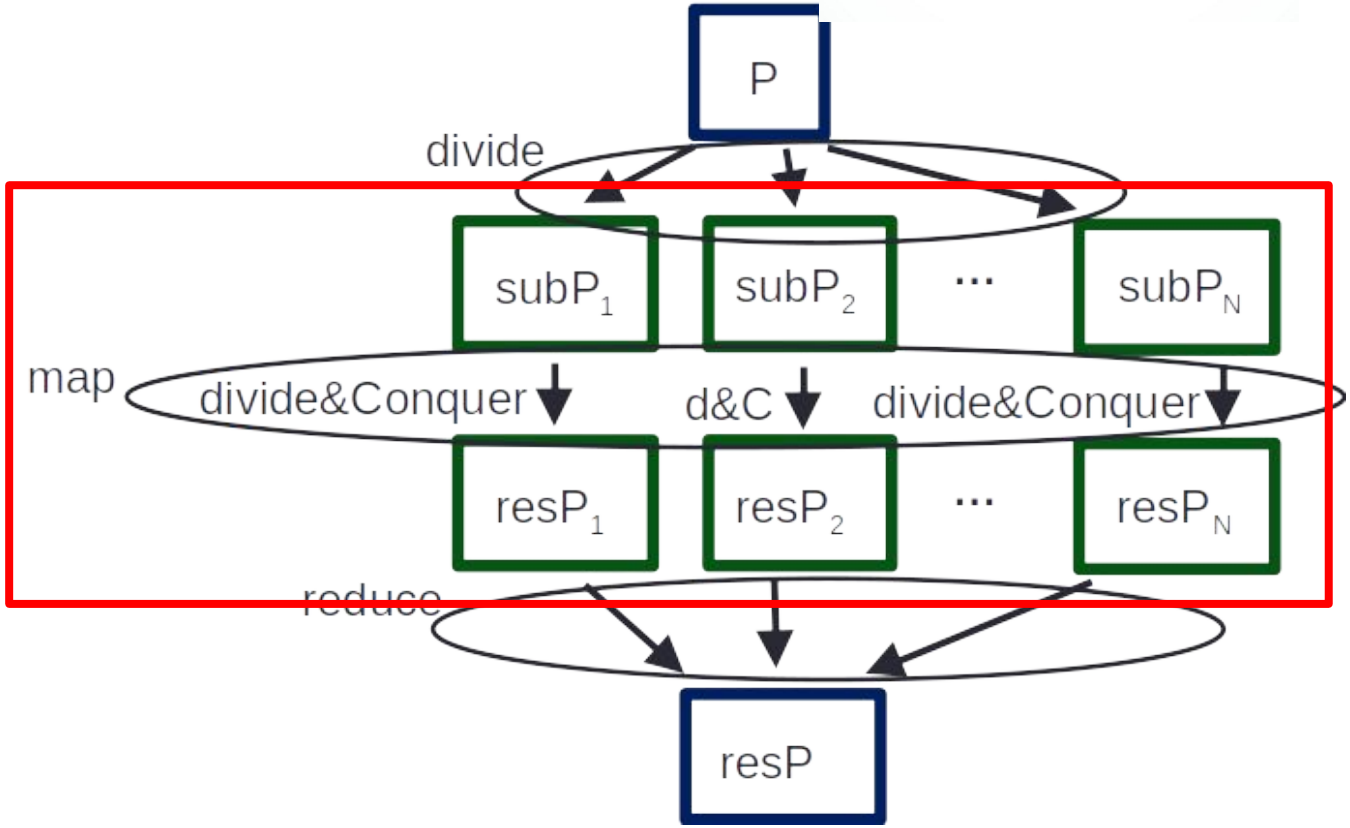




2. MAP STAGE.

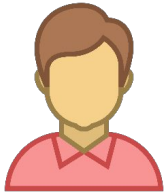
The meta-algorithm must supervise the execution of the processes.

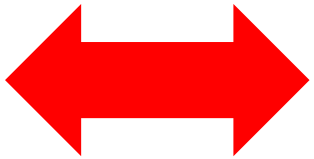
In particular:
- It provides fault tolerance capabilities, to reallocate processes among cores in case of failure, idle or slow down runtime.



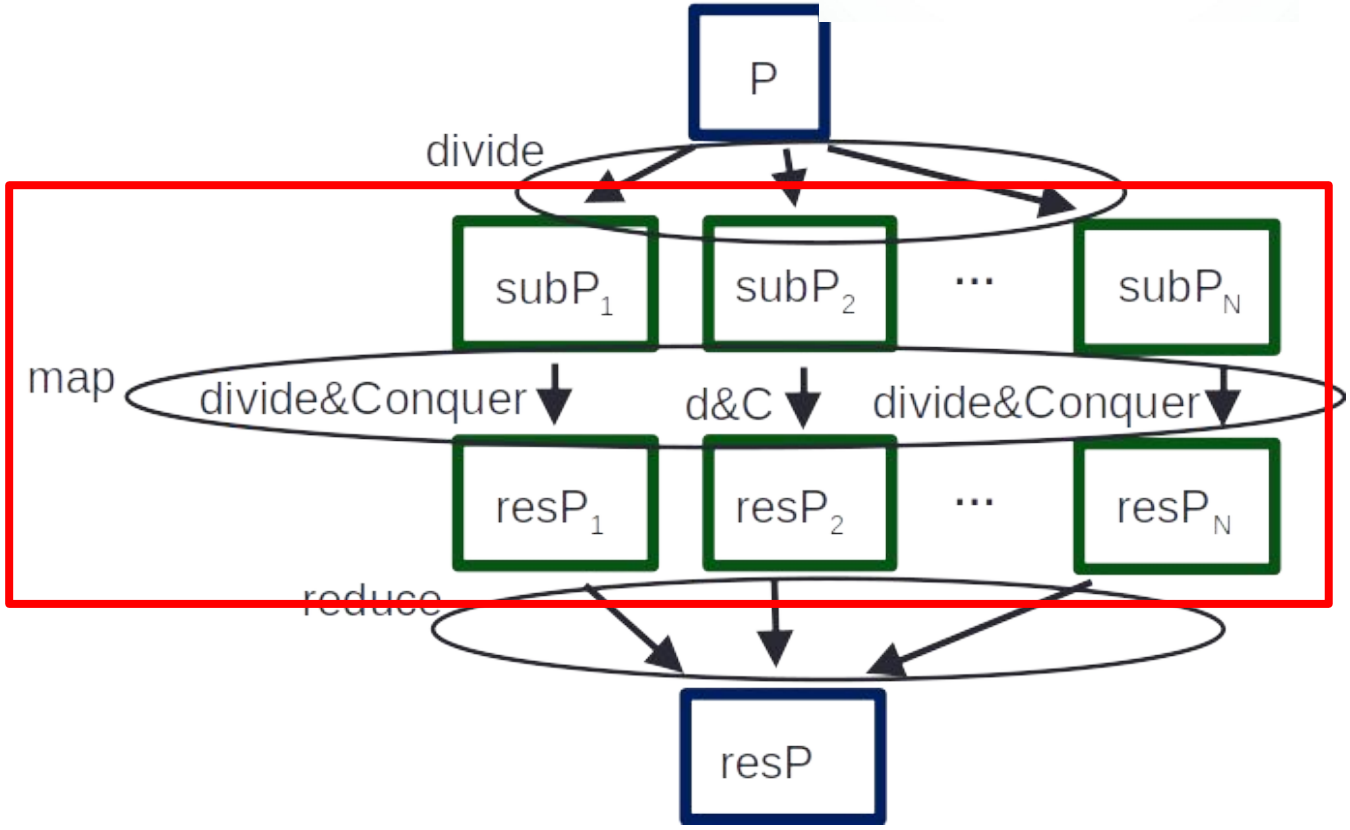
Concurrency: Infrastructure-Based Approach

2. MAP STAGE.



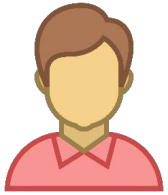


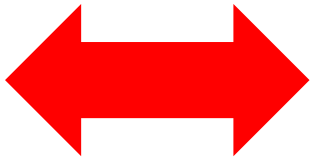
...in our case for this lecture we will need our problem-specific **map** stage algorithm.



Concurrency: Infrastructure-Based Approach

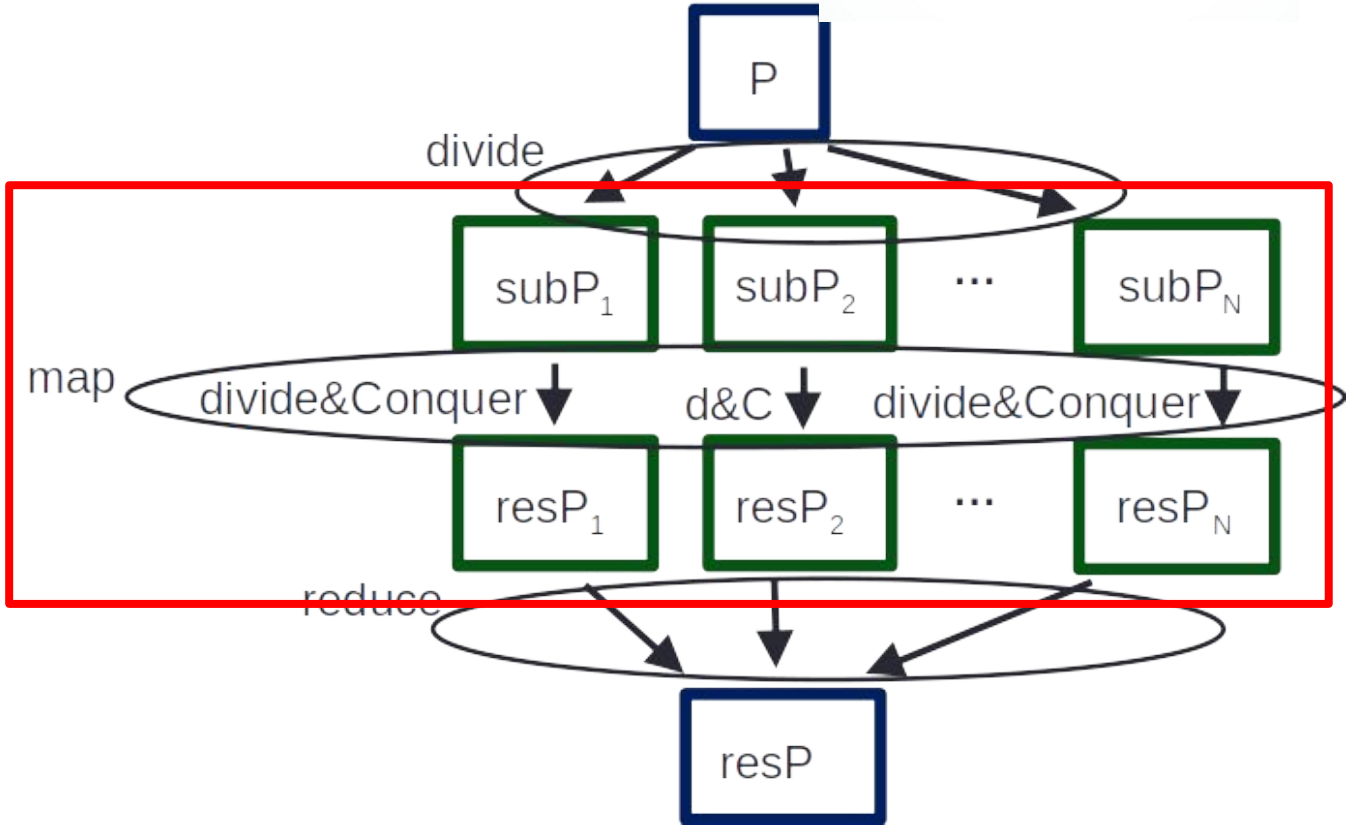
2. MAP STAGE.





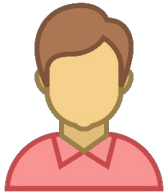
...in our case for this lecture we will need our problem-specific map stage algorithm.

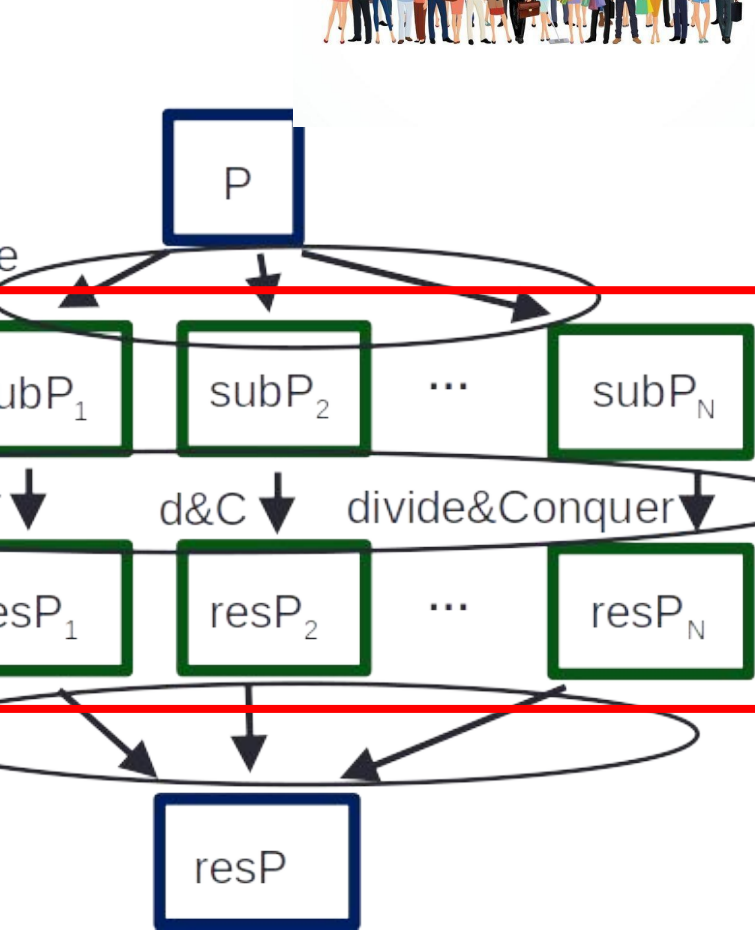
This algorithm just uses multi-threading to trigger the solving of each sub-part via separate threads.



Concurrency: Infrastructure-Based Approach

2. MAP STAGE.



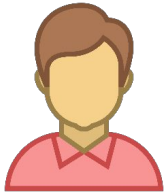


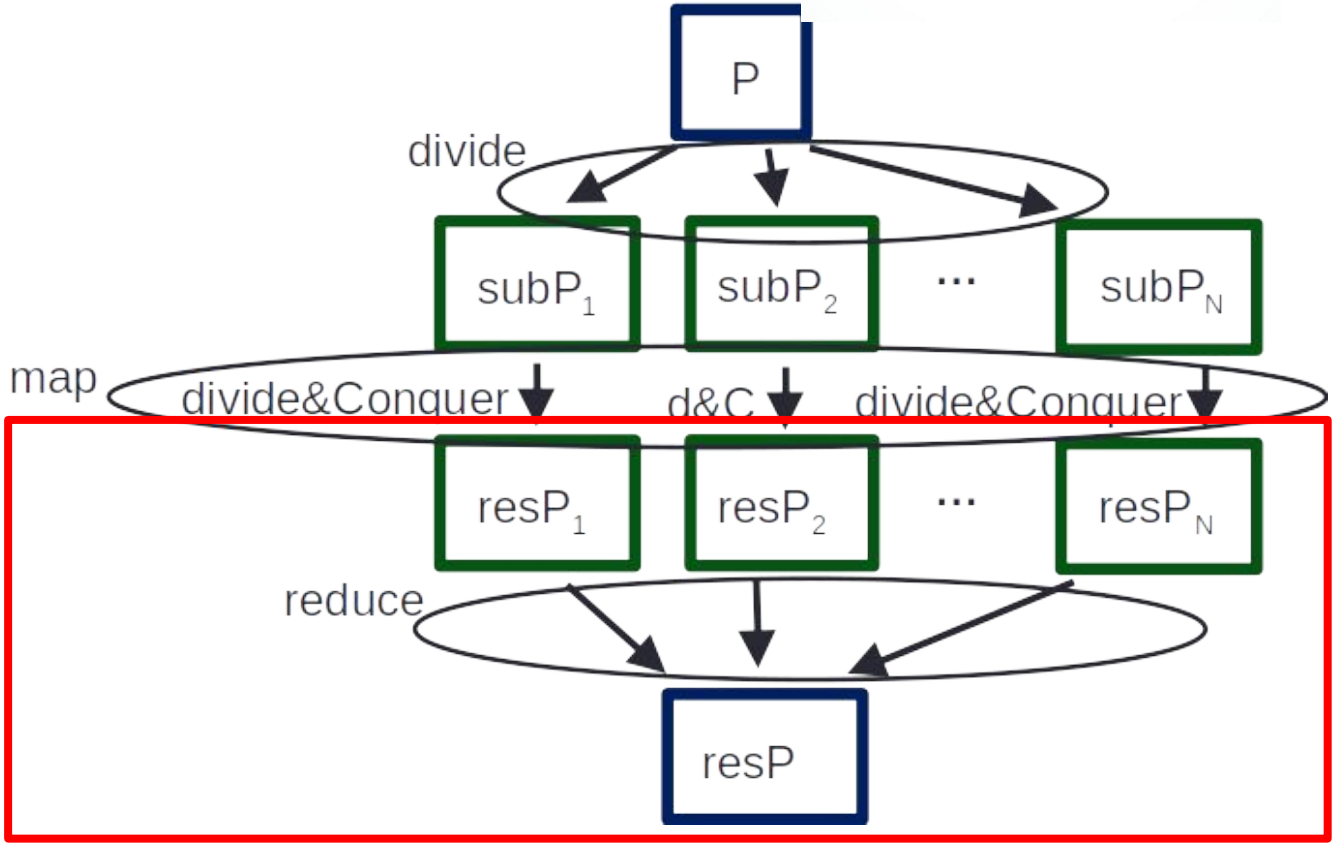
...in our case for this lecture we will need our problem-specific map stage algorithm.

Our algorithm is modest enough. It does not schedule the processes to cores (relies on the OS for doing so) nor provide fault tolerance capabilities.

Concurrency: Infrastructure-Based Approach

3. REDUCE STAGE.

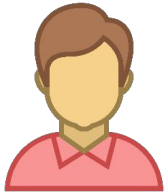


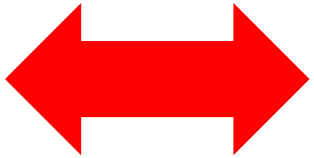


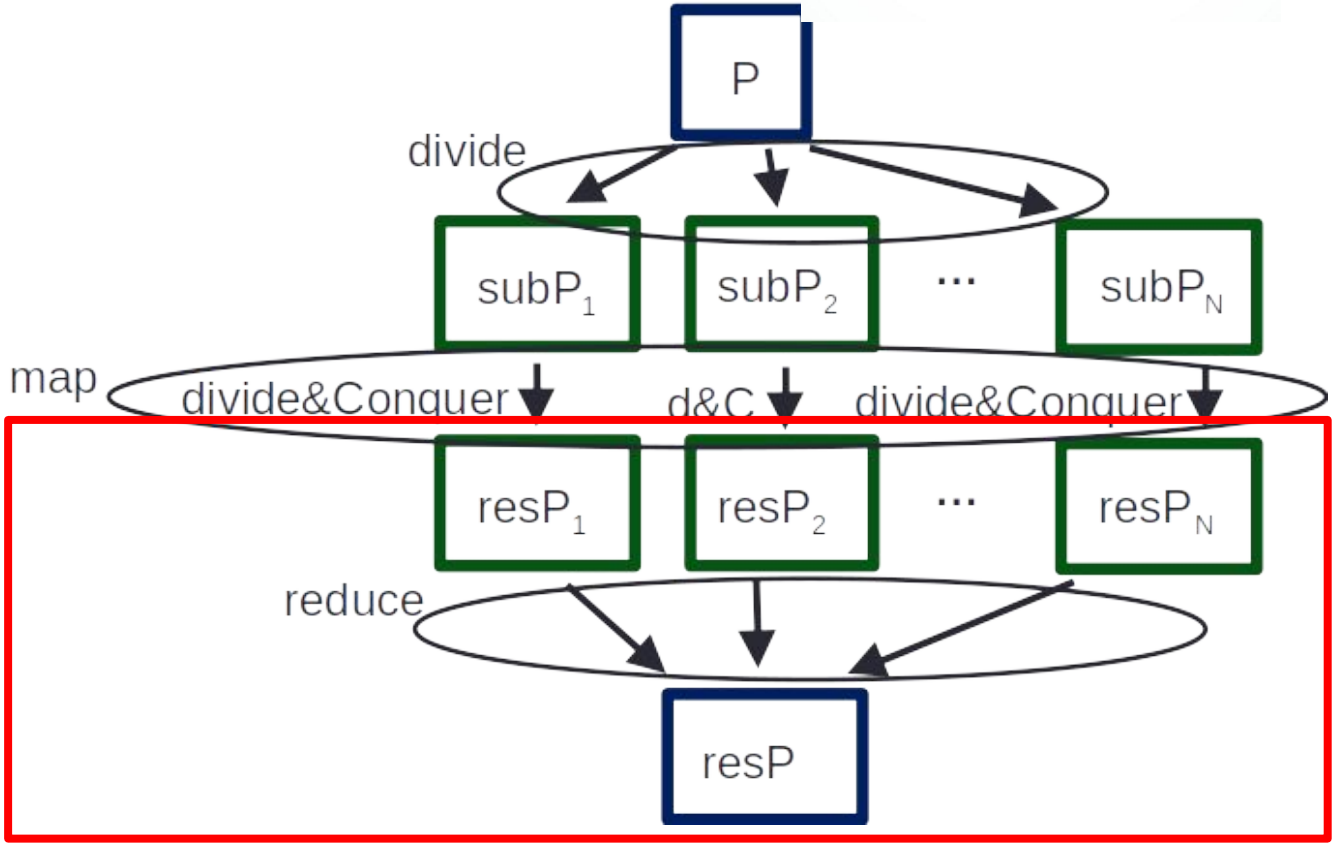
Concurrency: Infrastructure-Based Approach

3. REDUCE STAGE.

We combine the sub-part solutions to get the solution to the original problem.

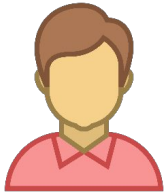


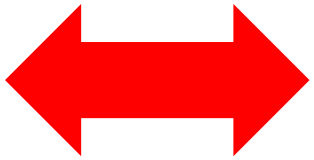




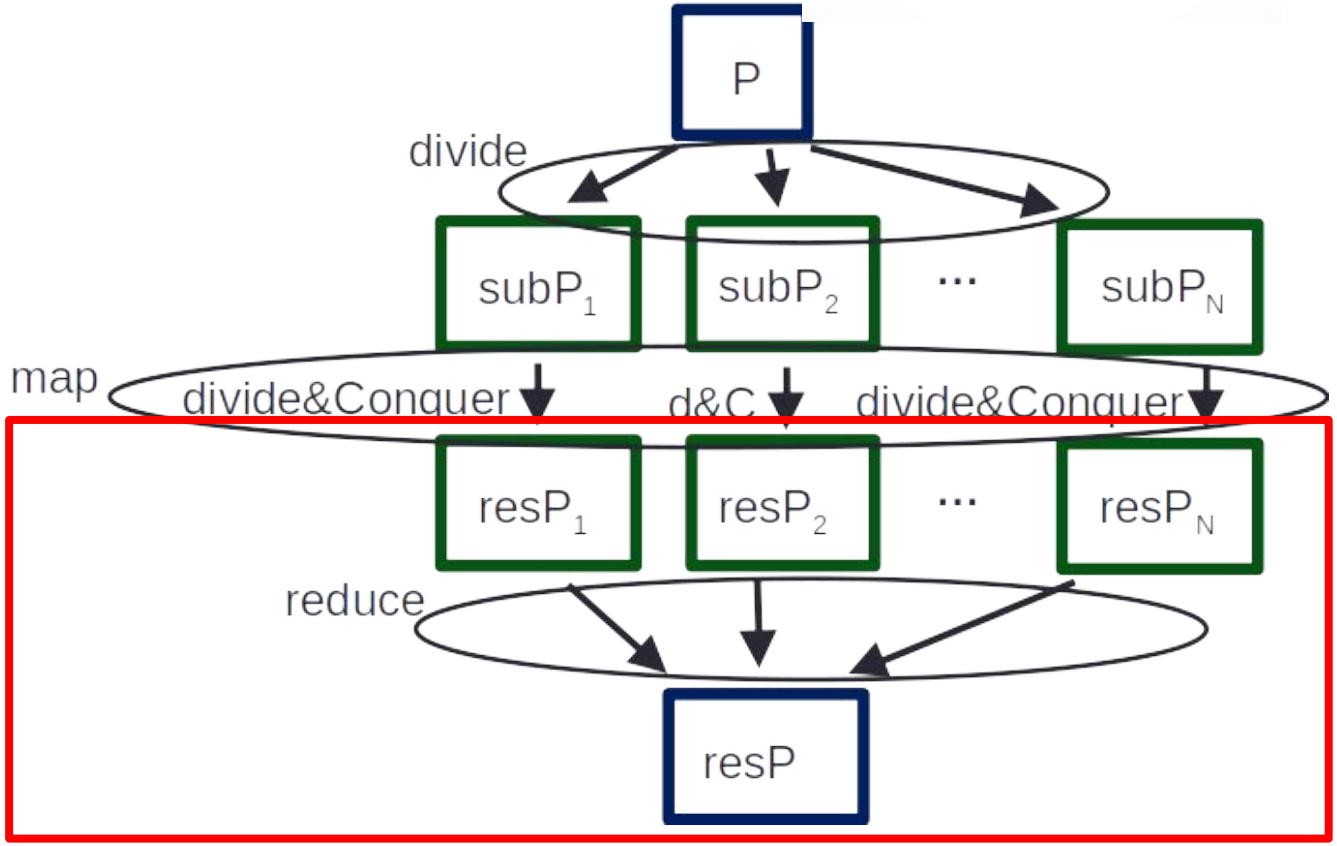
Concurrency: Infrastructure-Based Approach

3. REDUCE STAGE.



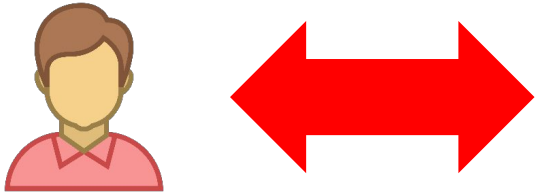


...in our case for this lecture we will need our problem-specific **reduce** stage algorithm.



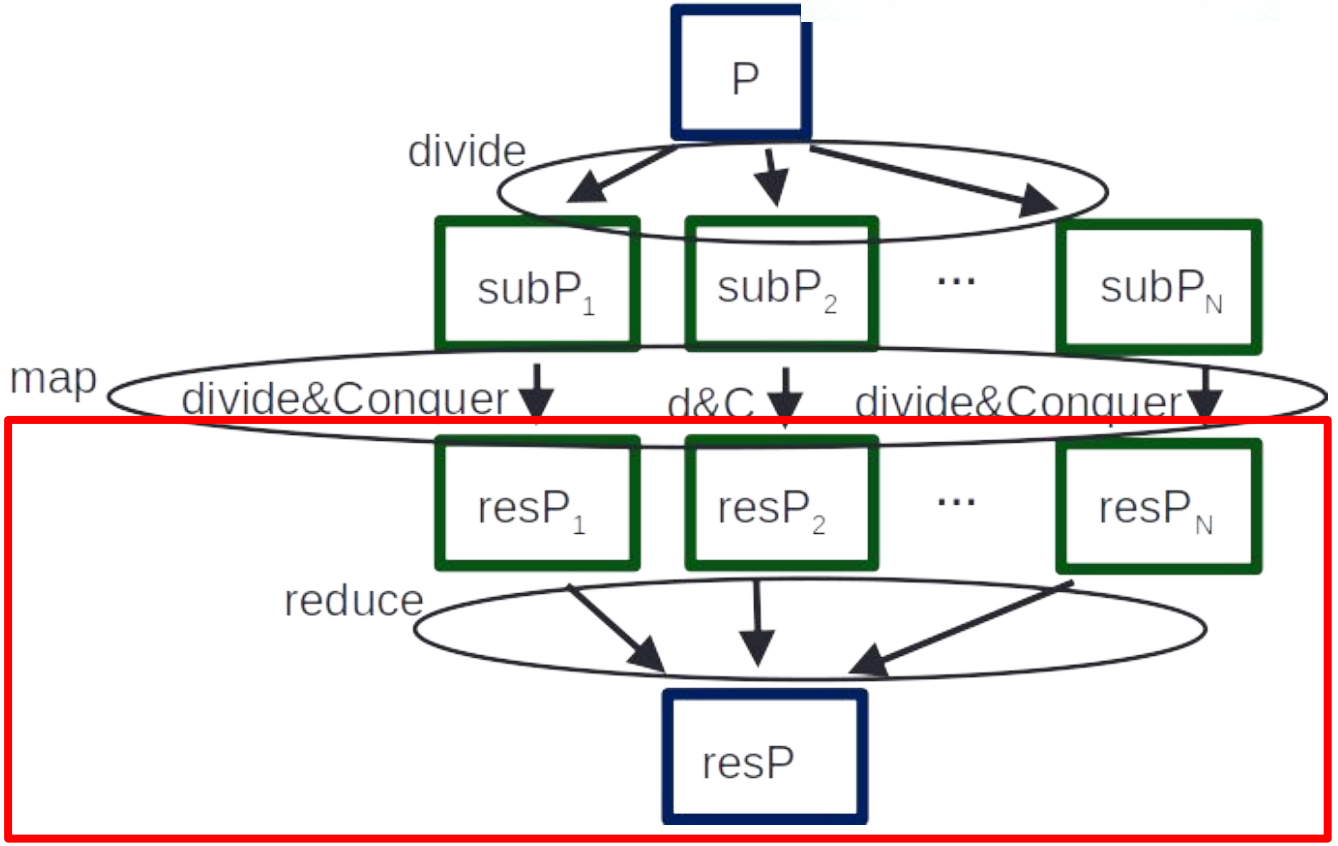
Concurrency: Infrastructure-Based Approach

3. REDUCE STAGE.



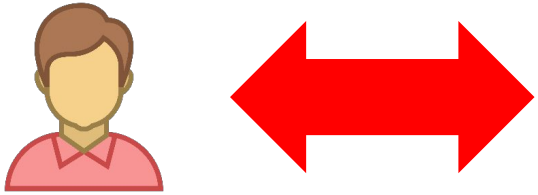
...in our case for this lecture we will need our problem-specific **reduce** stage algorithm.

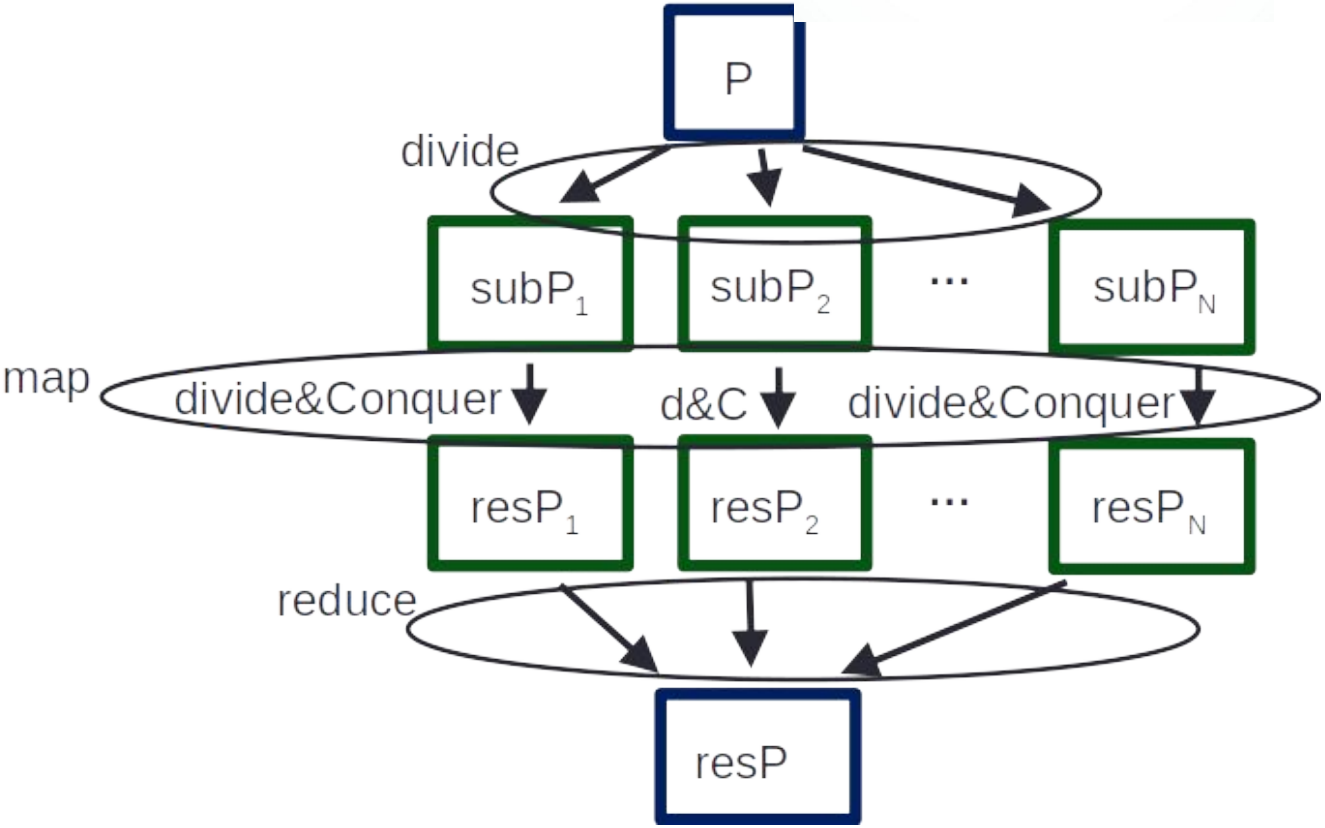
This algorithm just puts together the solution lists of the sub-parts and computes the total and elapsed times.



Concurrency: Infrastructure-Based Approach

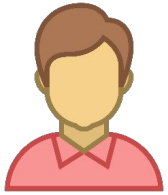
The code examples of the lecture allow us to explore our **problem-specific divide**, **map** and **reduce** algorithms for solving the list inversions of P against an entire population.

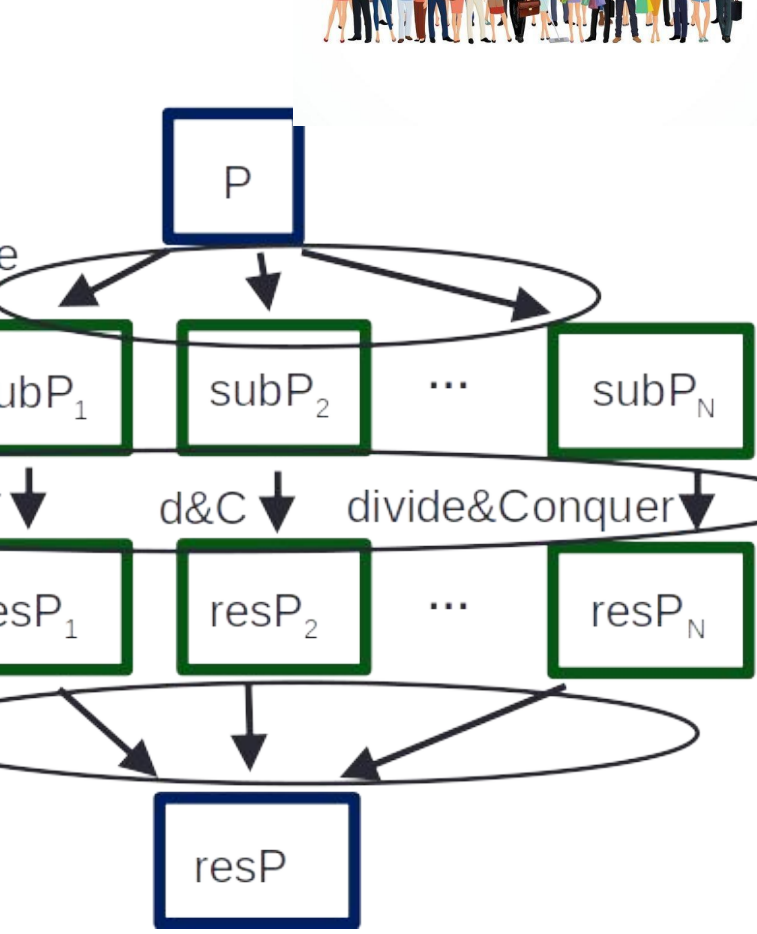




Concurrency: Infrastructure-Based Approach

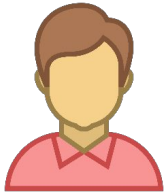
State-of-the-art frameworks like Spark will outperform our implementation in 2 ways:



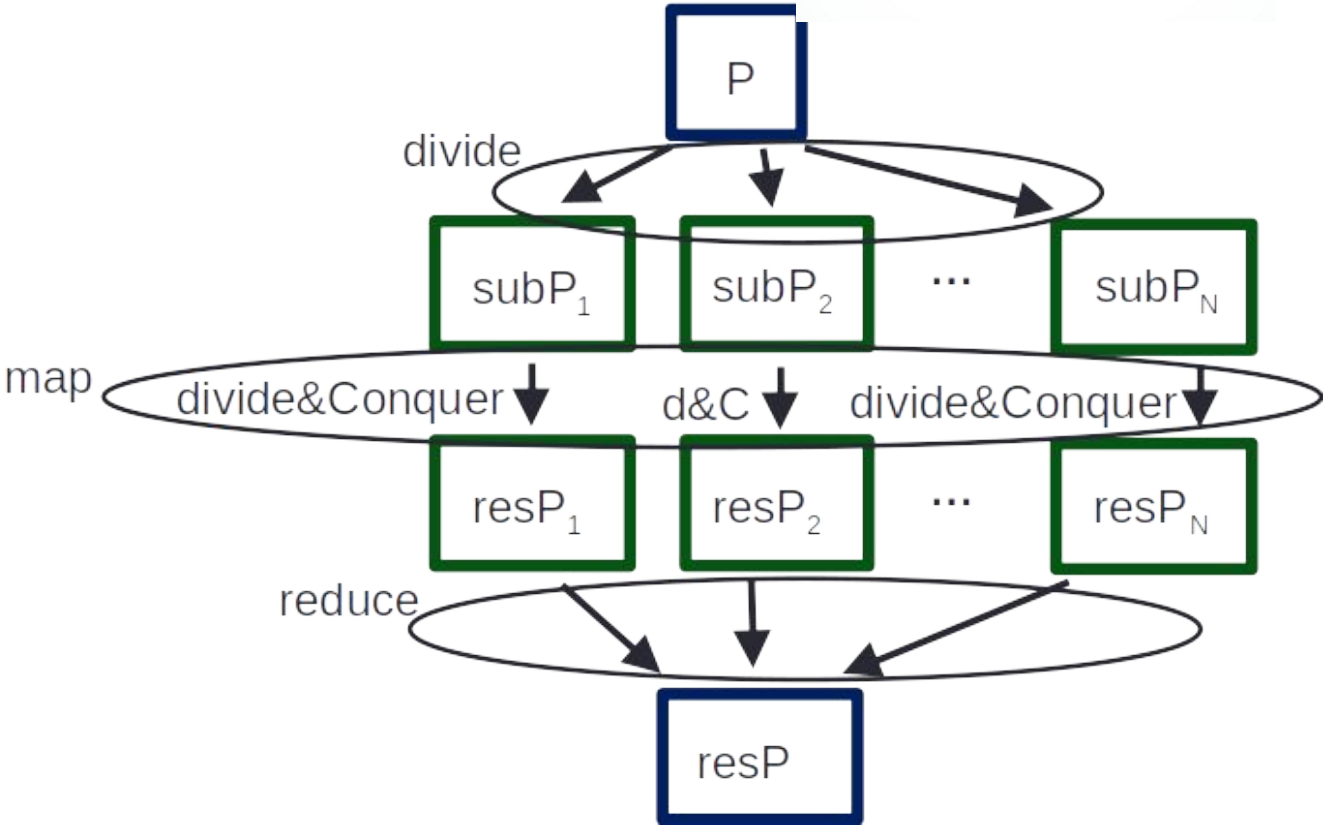


Concurrency: Infrastructure-Based Approach

State-of-the-art frameworks like Spark will outperform our implementation in 2 ways:

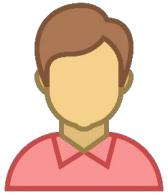


1. **It abstracts the creation of the meta-algorithm.**

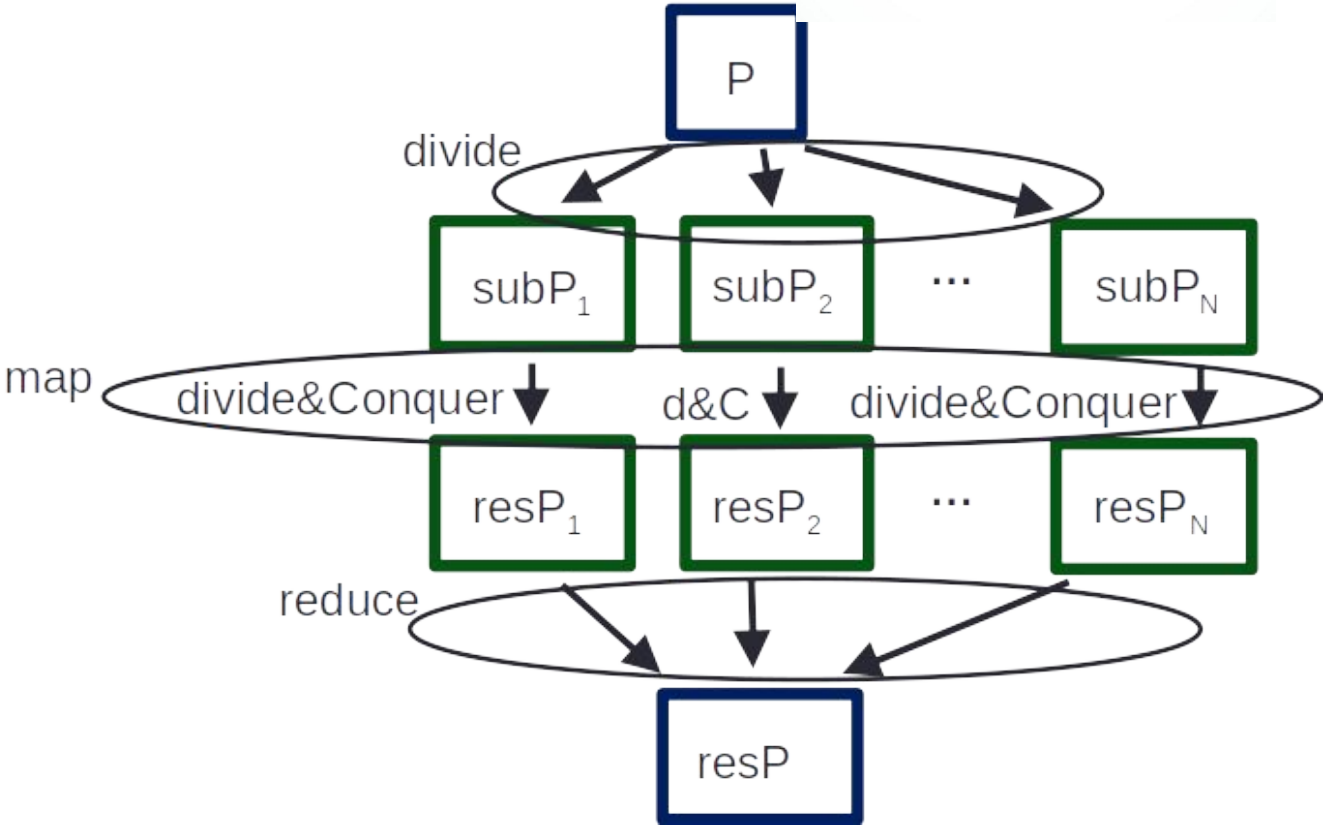


Concurrency: Infrastructure-Based Approach

1. **It abstracts the creation of the meta-algorithm.**

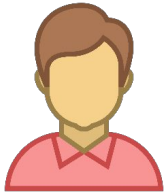


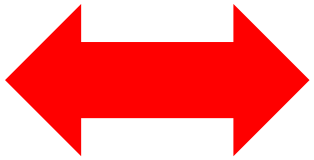
- The user can focus on its problem, programming it as if it was going to be solved sequentially. Its distributed execution will be scheduled in a transparent way to the user.



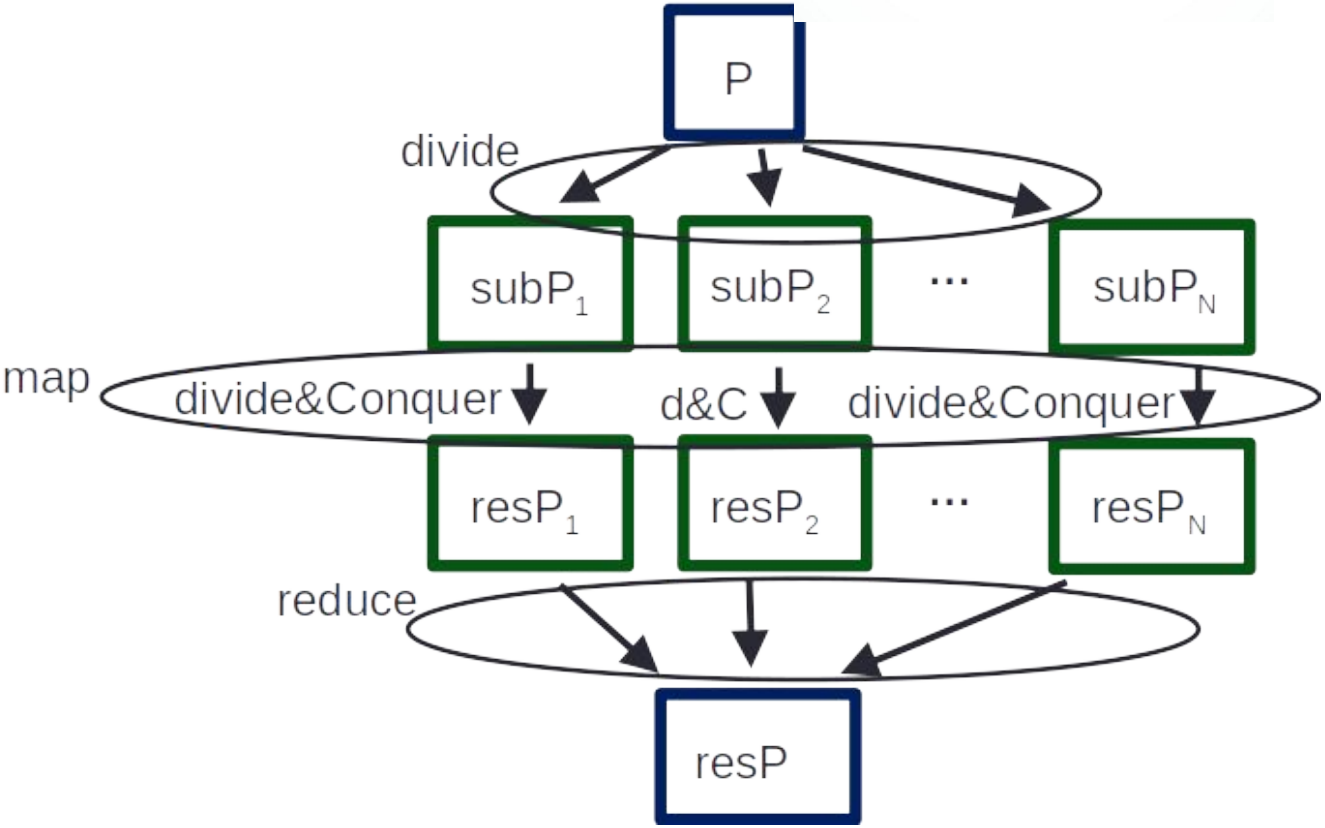
Concurrency: Infrastructure-Based Approach

State-of-the-art frameworks like Spark will outperform our implementation in 2 ways:



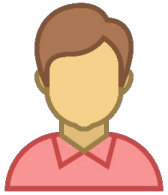


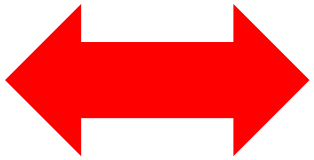
2. **It comes up with general, <u>problem independent</u> divide, map and reduce subroutines.**



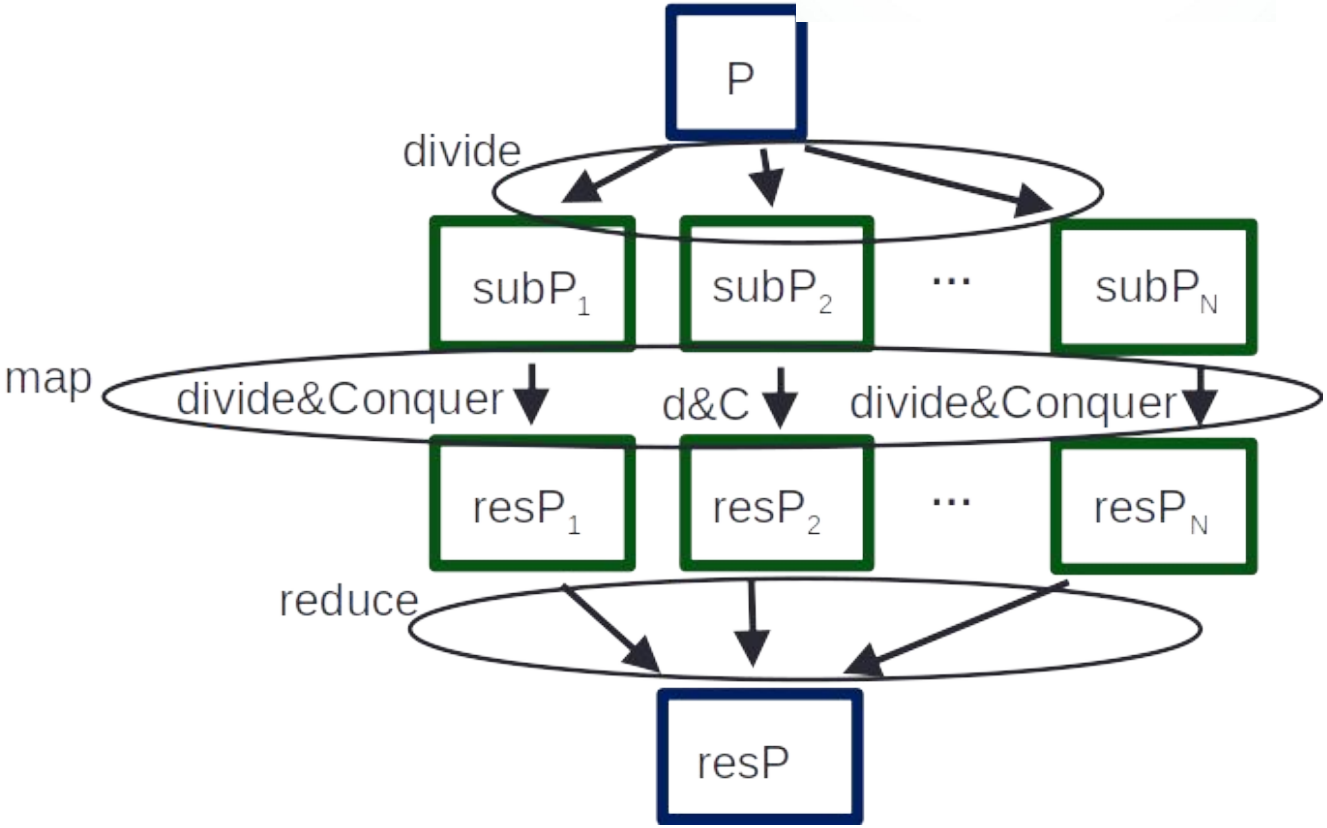
Concurrency: Infrastructure-Based Approach

2. **It comes up with general, <u>problem independent</u> divide, map and reduce subroutines.**





- In our code for the lecture, the problem-specific divide, map and reduce algorithms cannot be reused for another problem.



Concurrency: Infrastructure Based Approach

So, all in all, we conclude that distributed programming is a wonderful approach to tackle scalability problems.

Concurrency: Infrastructure Based Approach

So, all in all, we conclude that distributed programming is a wonderful approach to tackle scalability problems.

However, it has its own limitations, based on:

- the dependency of the speed-up on the number of cores of the cluster.
- the meta-algorithm requirement to schedule the distributed execution.

Outline

1. Sequential vs. Concurrent Execution.
2. List Inversions: A Real-world Example.
3. Scalability: Computational Complexity Barrier.
4. Concurrency: Infrastructure-based Approach.
5. Can We Do Better? An Algorithm-based Approach.

Can We Do Better? An Algorithm Based Approach

Distributed Programming is the approach we are going to follow for Big Data Processing.

Can We Do Better? An Algorithm Based Approach

Distributed Programming is the approach we are going to follow for Big Data Processing.

Full stop.

Can We Do Better? An Algorithm Based Approach

Now, as computer scientists,
we simply cannot finish this lecture here.

Can We Do Better? An Algorithm Based Approach

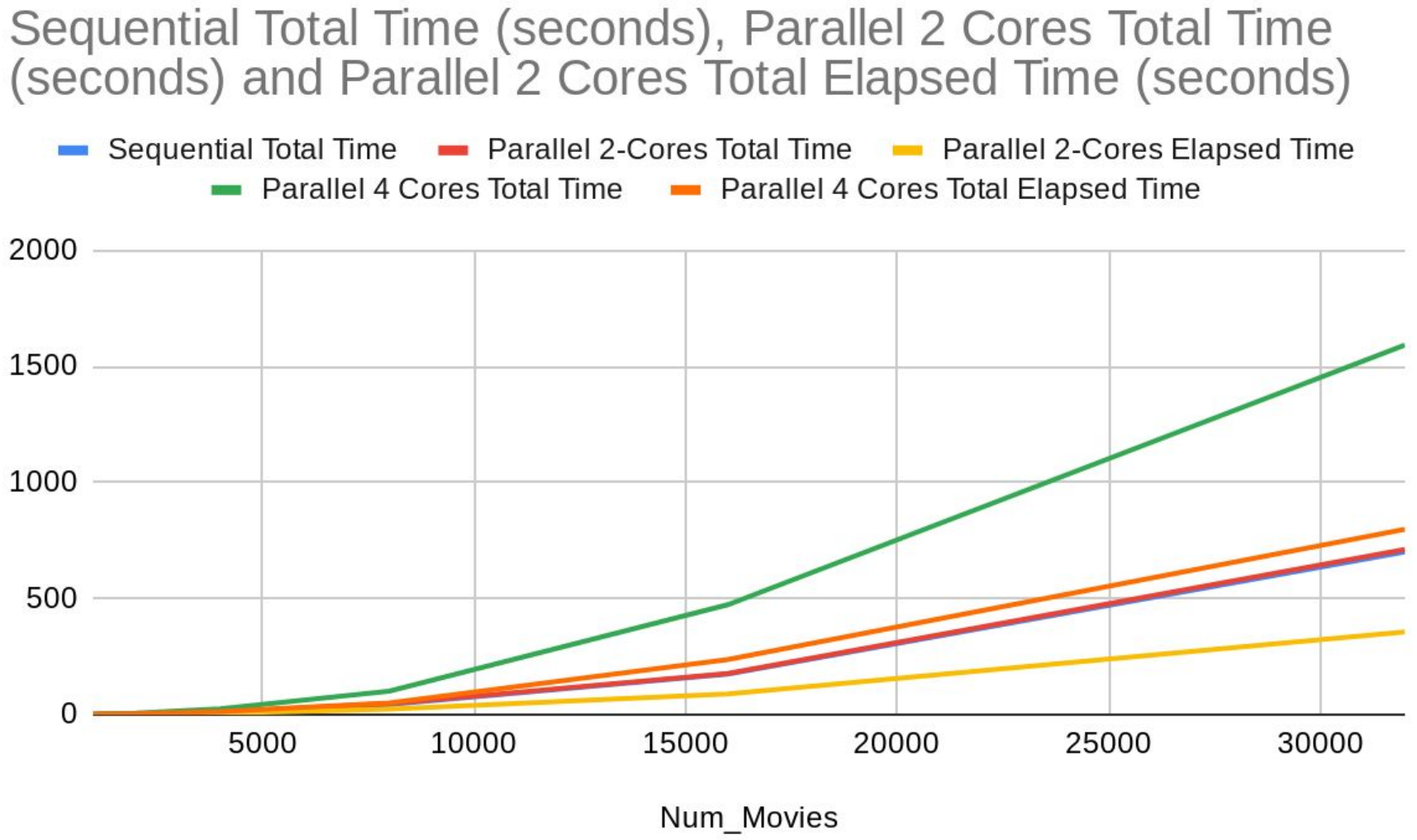
Now, as computer scientists,
we simply cannot finish this lecture here.

On top of using Distributed Programming to tackle scalability, as computer scientists we always have to challenge our algorithms.

Can We Do Better? An Algorithm Based Approach

Let's look back to our time analysis again...

Can We Do Better? An Algorithm Based Approach



Can We Do Better? An Algorithm Based Approach

Let's look back to our time analysis again...

...but now let's look at it by focusing on the algorithm count inversions $O(n^2)$.

Can We Do Better? An Algorithm Based Approach

Let's look back to our time analysis again...

...but now let's look at it by focusing on the algorithm count inversions $O(n^2)$.

Can we make these times go down by using a more efficient algorithm?

Can We Do Better? An Algorithm Based Approach

Let's look back to our time analysis again...

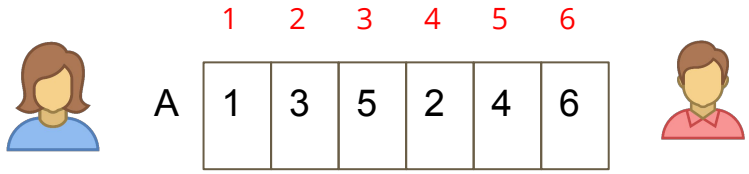
...but now let's look at it by focusing on the algorithm count inversions $O(n^2)$.

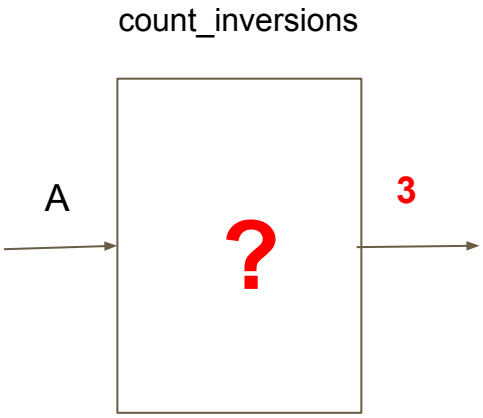
Can we make these times go down by using a more efficient algorithm?

Yes, we can.

Can We Do Better? An Algorithm Based Approach

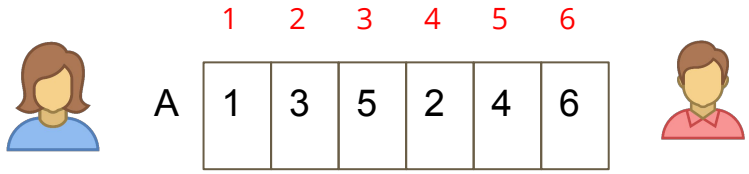
- Our problem of
  - Computing list inversions of an array A.

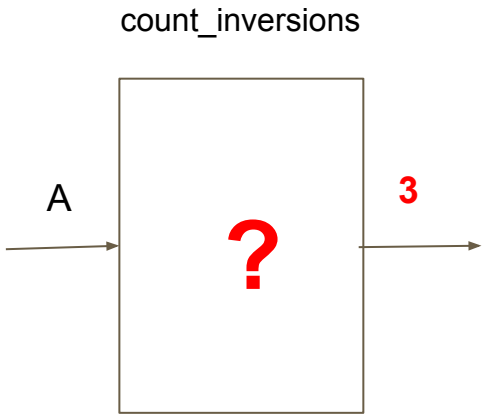




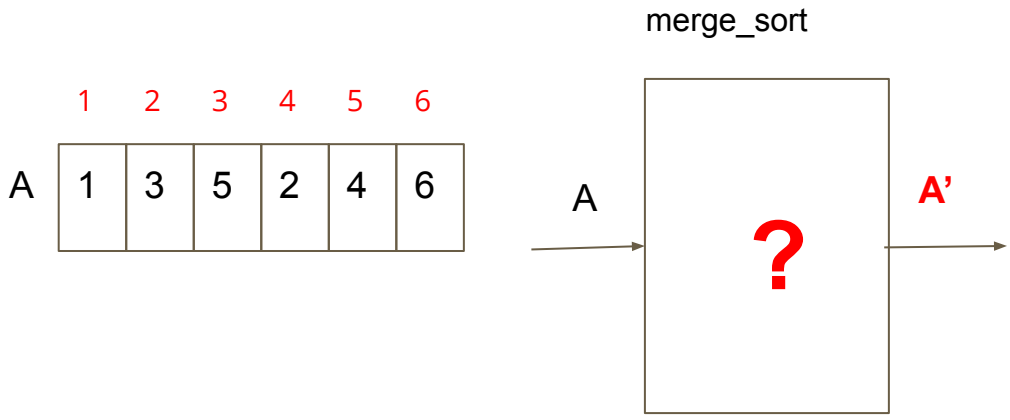
Can We Do Better? An Algorithm Based Approach

- Our problem of
  - Computing list inversions of an array A.





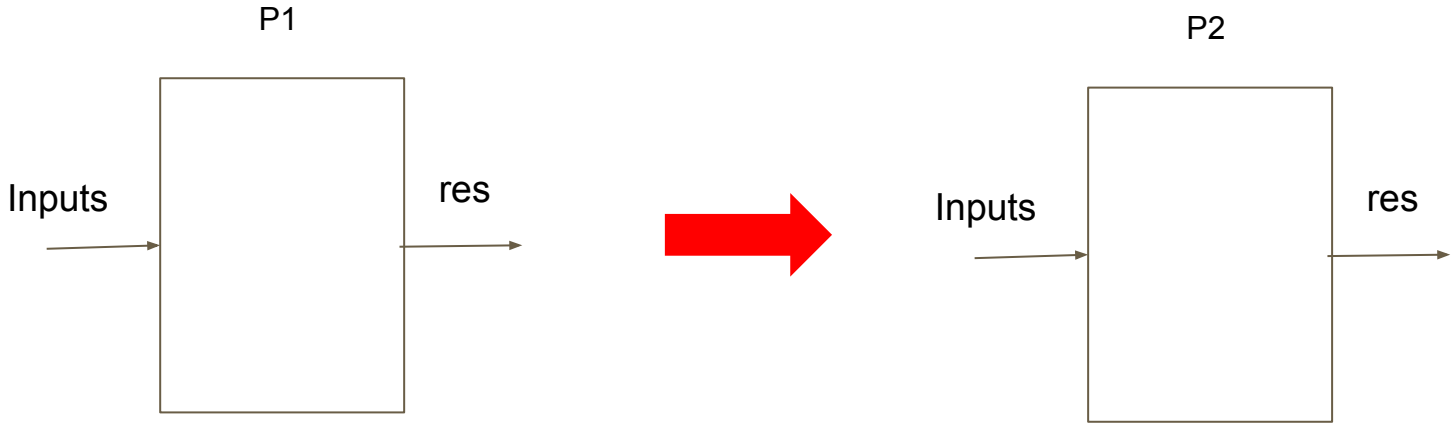
- **reduces** to the problem of
  - Sorting the array A via mergesort.



Can We Do Better? An Algorithm Based Approach

Let's remember the formal definition of **reduction** in Computer Science.

A Problem P1 reduces to P2 if,
given a polynomial-time algorithm for solving P2,
we can use the algorithm in charge of P2
to build a new polynomial-time algorithm solving P1.



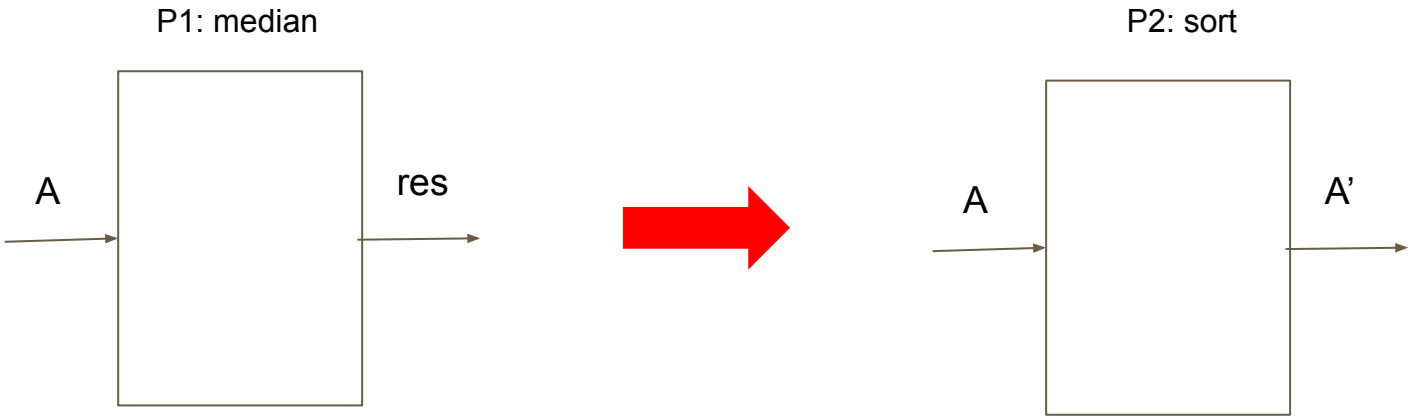
Can We Do Better? An Algorithm Based Approach

Example:

P1: Compute the median number of array A

reduces to

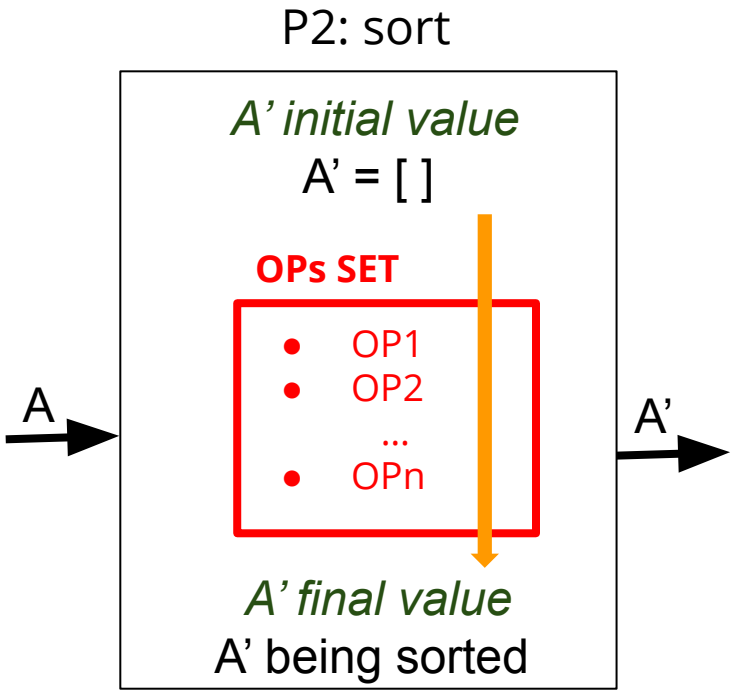
P2: Sort the array A.



Can We Do Better? An Algorithm Based Approach

Proof:

Given the polynomial-time algorithm for solving P2: Sort an Array...

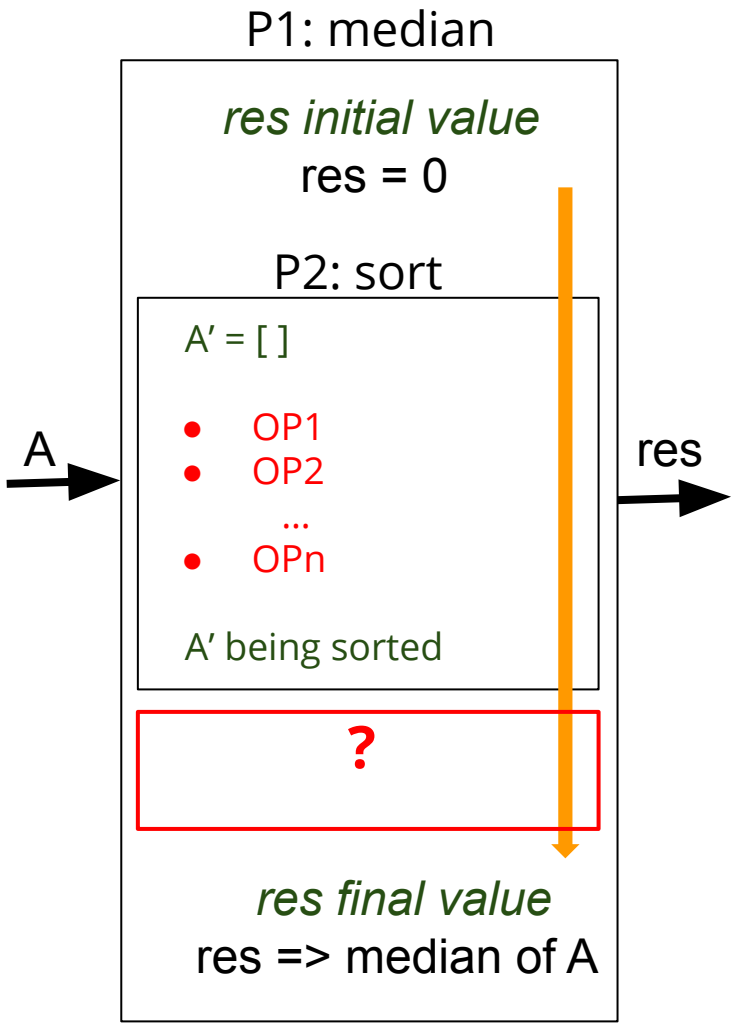


Can We Do Better? An Algorithm Based Approach

Proof:

Given the polynomial-time algorithm for solving P2: Sort an Array...

...we can use this algorithm to build a polynomial time solution for P1: Median of A.



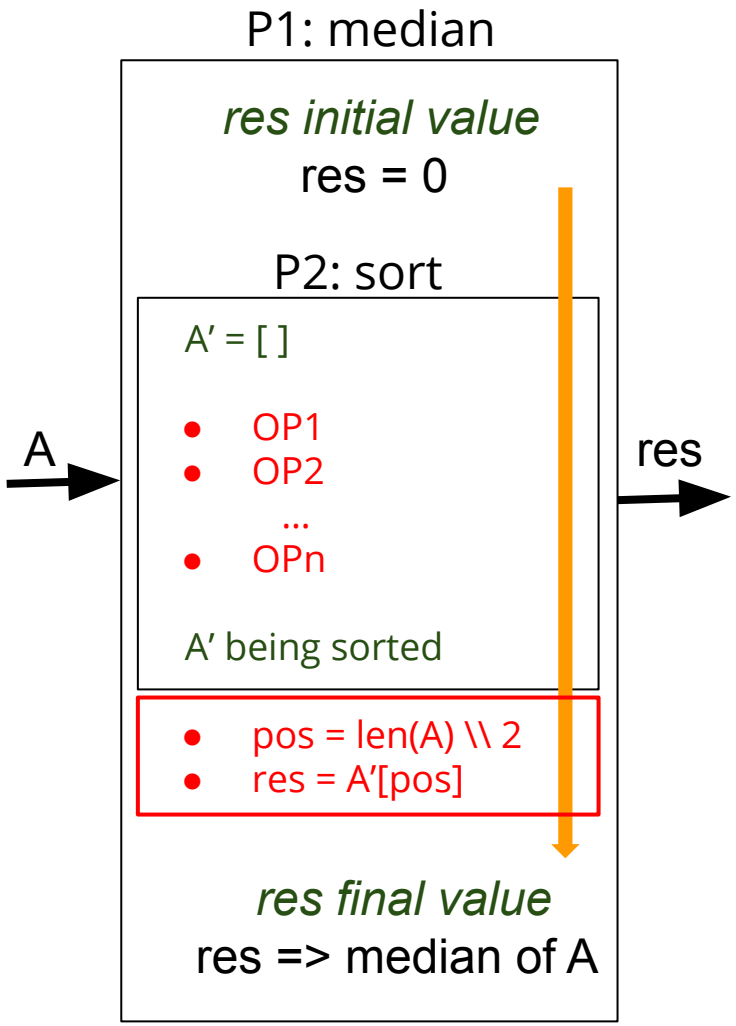
Can We Do Better? An Algorithm Based Approach

Proof:

Given the polynomial-time algorithm for solving P2: Sort an Array...

...we can use this algorithm to build a polynomial time solution for P1: Median of A.

It suffices with looking at the middle position of the sorted array A'.



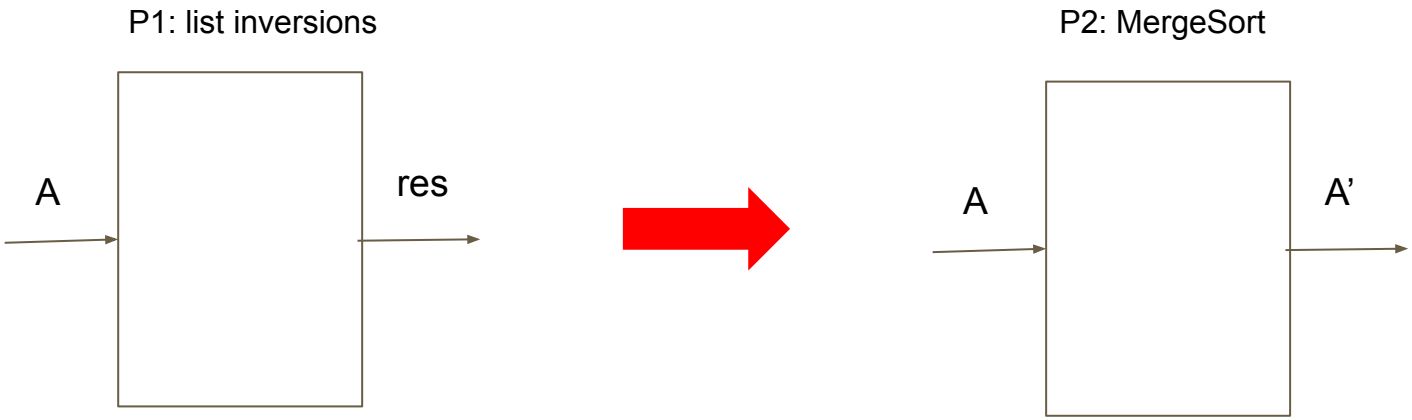
Can We Do Better? An Algorithm Based Approach

Let's now understand the process for our list inversions problem:

P1: Compute the list inversions of array A

reduces to

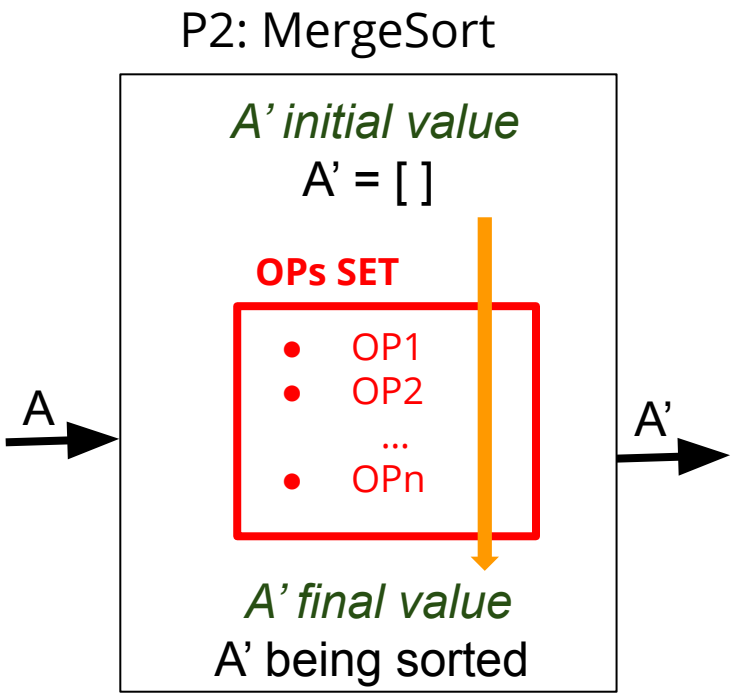
P2: Merge Sort the array A.



Can We Do Better? An Algorithm Based Approach

Proof:

Given the polynomial-time algorithm for solving P2: MergeSort an Array...

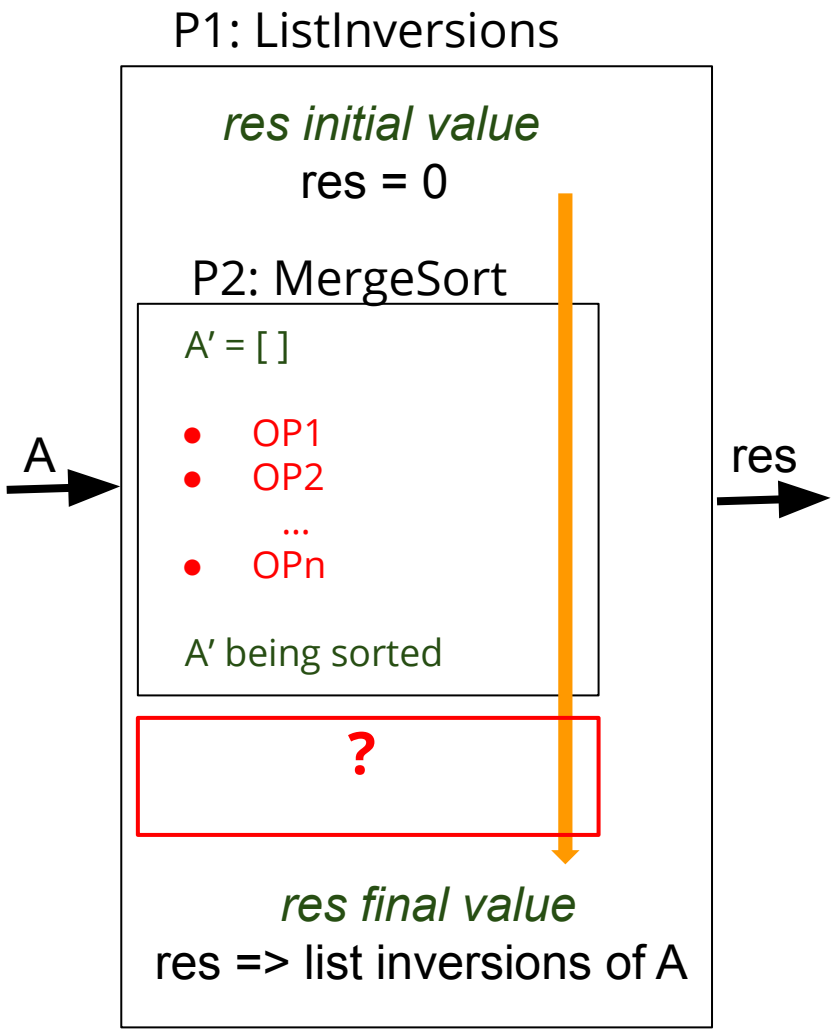


Can We Do Better? An Algorithm Based Approach

Proof:

Given the polynomial-time algorithm for solving P2: MergeSort an Array...

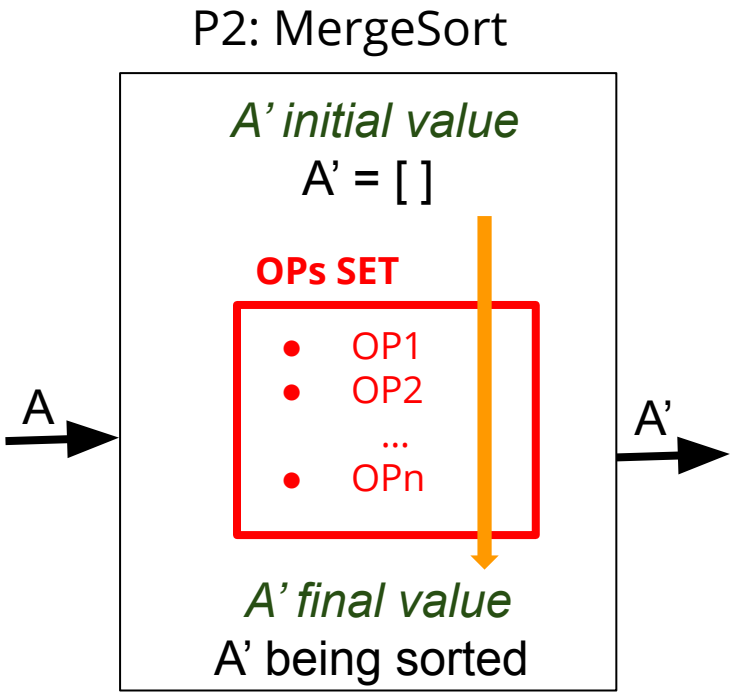
...we can use this algorithm to build a polynomial time solution for P1: List Inversions of A.



Can We Do Better? An Algorithm Based Approach

Let's first focus on how MergeSort works,
and analyse its time complexity
O(n log n)...

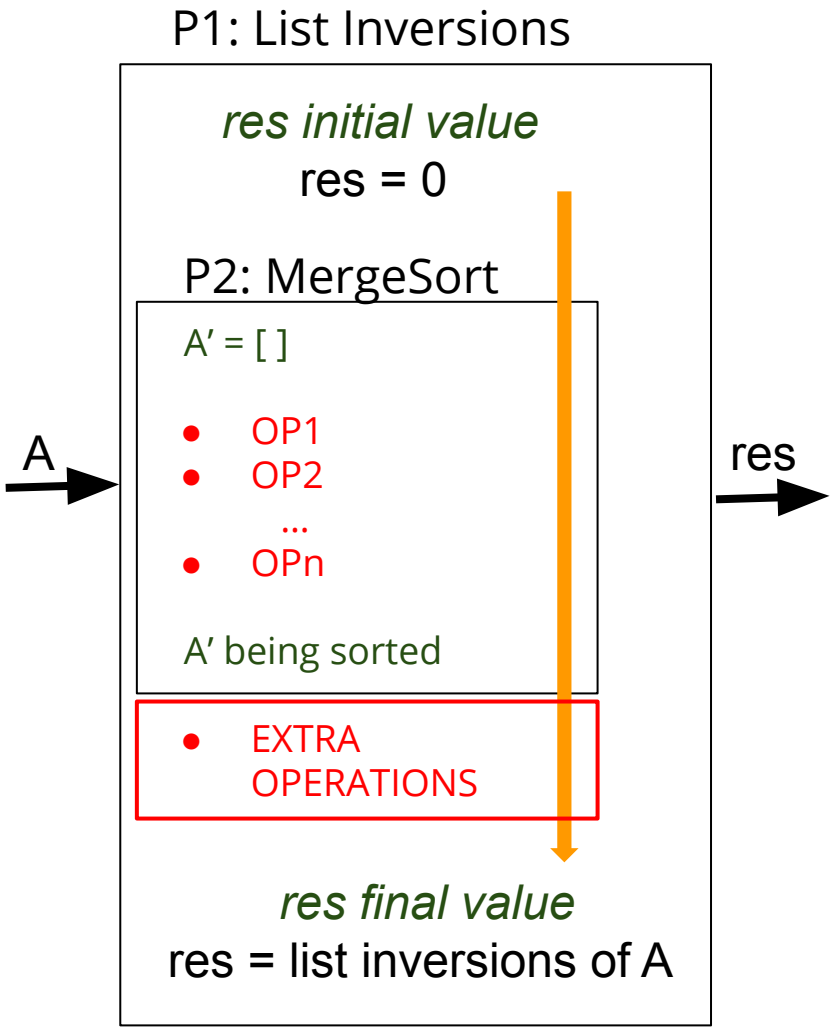
Can We Do Better? An Algorithm Based Approach



Can We Do Better? An Algorithm Based Approach

...then, we will focus on how list inversions reduces to MergeSort, and how its time complexity is still O(n log n).

Can We Do Better? An Algorithm Based Approach

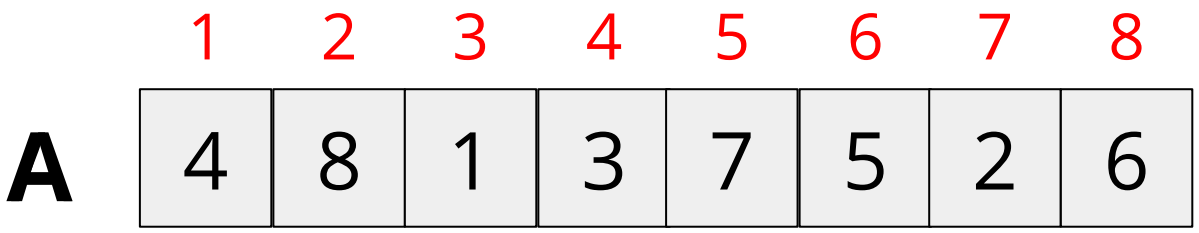


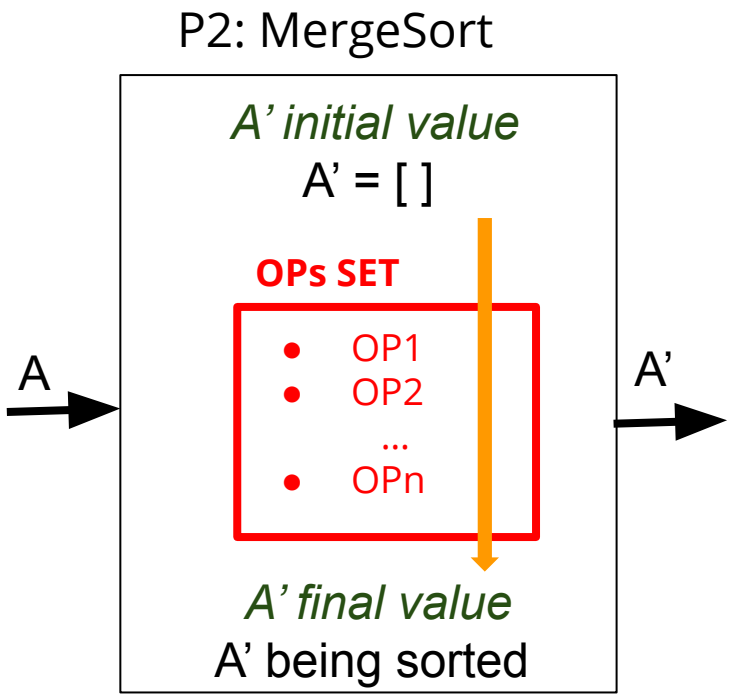
Can We Do Better? An Algorithm Based Approach

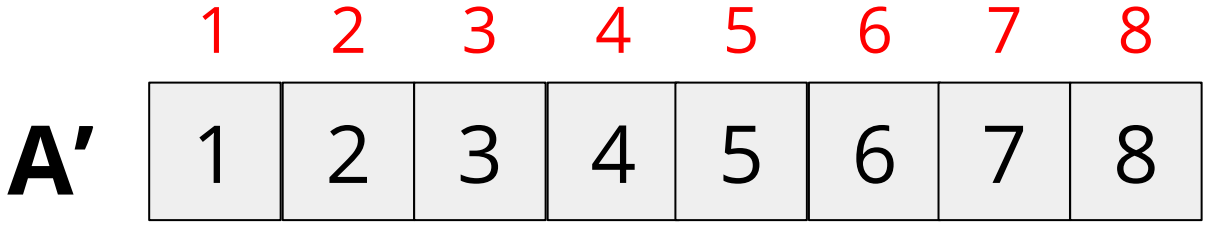
Sounds like a good plan?

Let's first focus on how MergeSort works,
and analyse its time complexity
O(n log n)...

- MergeSort
  Sorts Array A by...



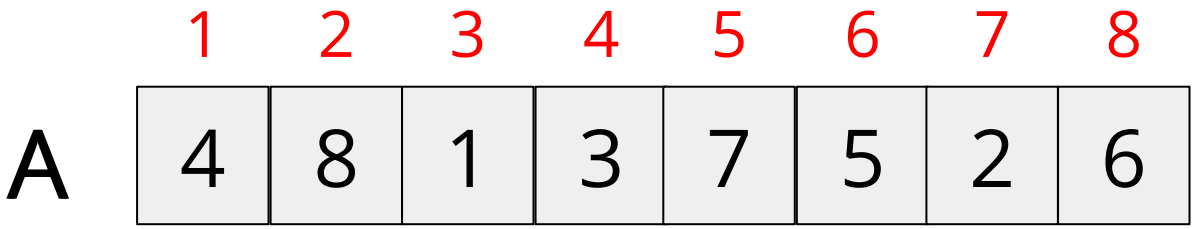




Can We Do Better? An Algorithm Based Approach

MergeSort

Sorts Array A by...

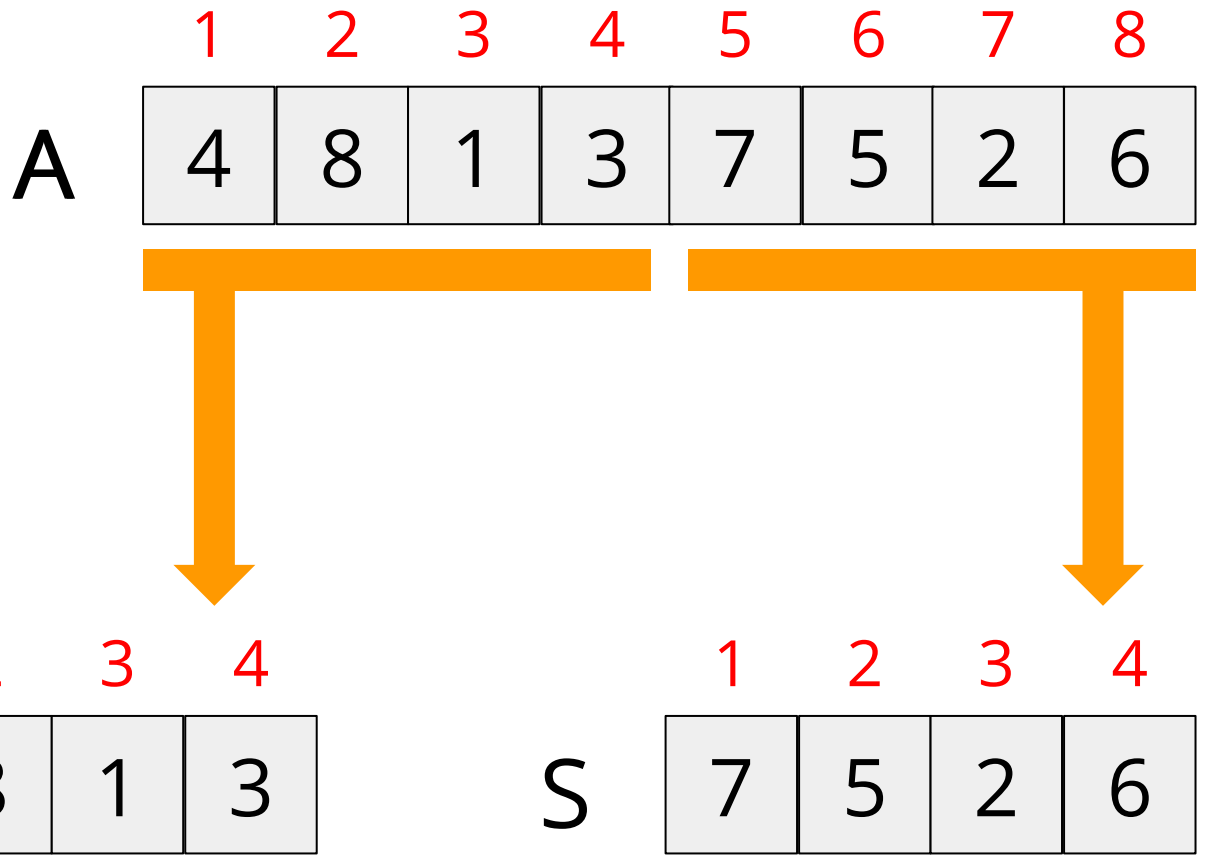


Can We Do Better? An Algorithm Based Approach

MergeSort

Sorts Array A by...

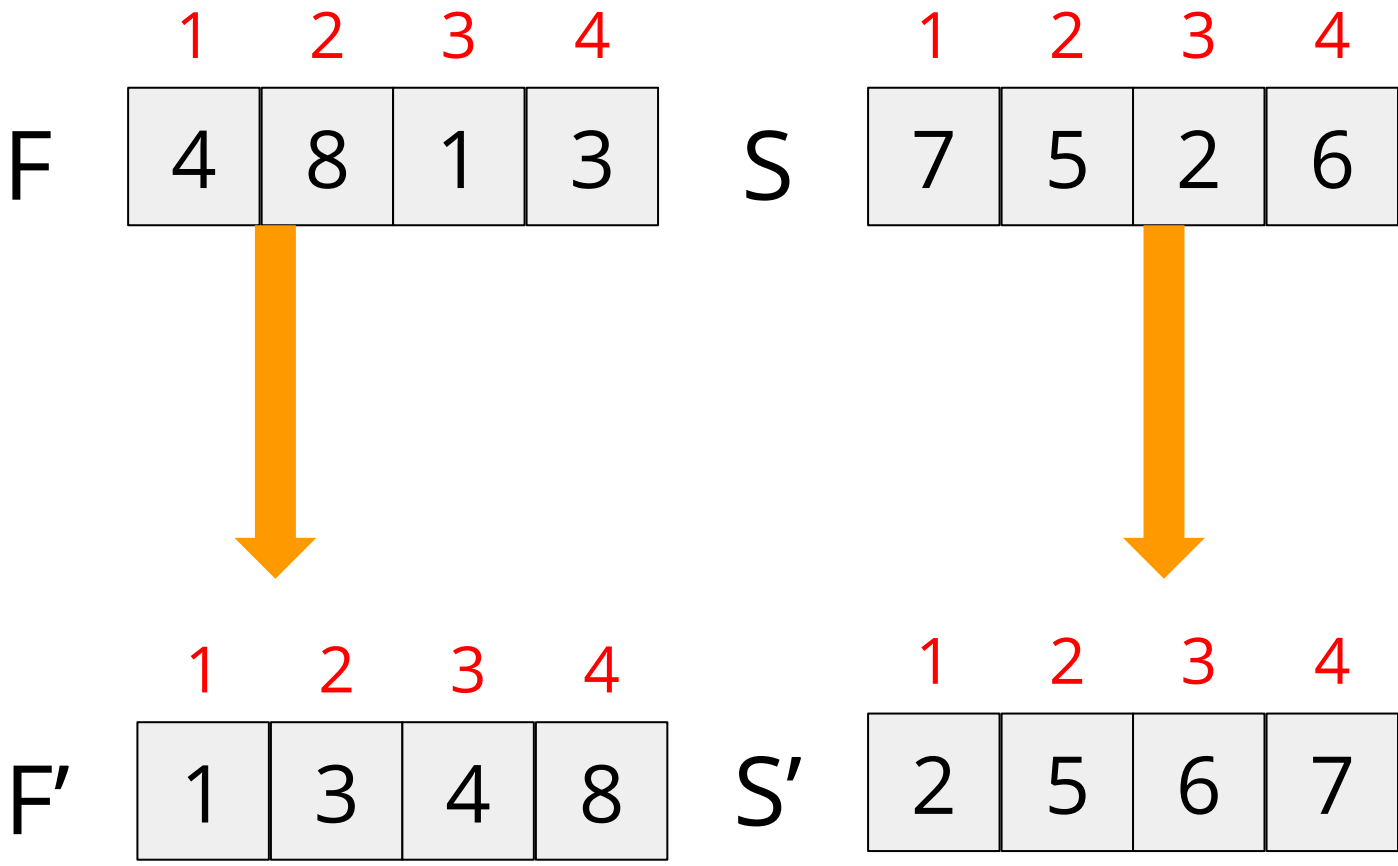
1. **Divide** A in halves



Can We Do Better? An Algorithm Based Approach

MergeSort
Sorts Array A by...

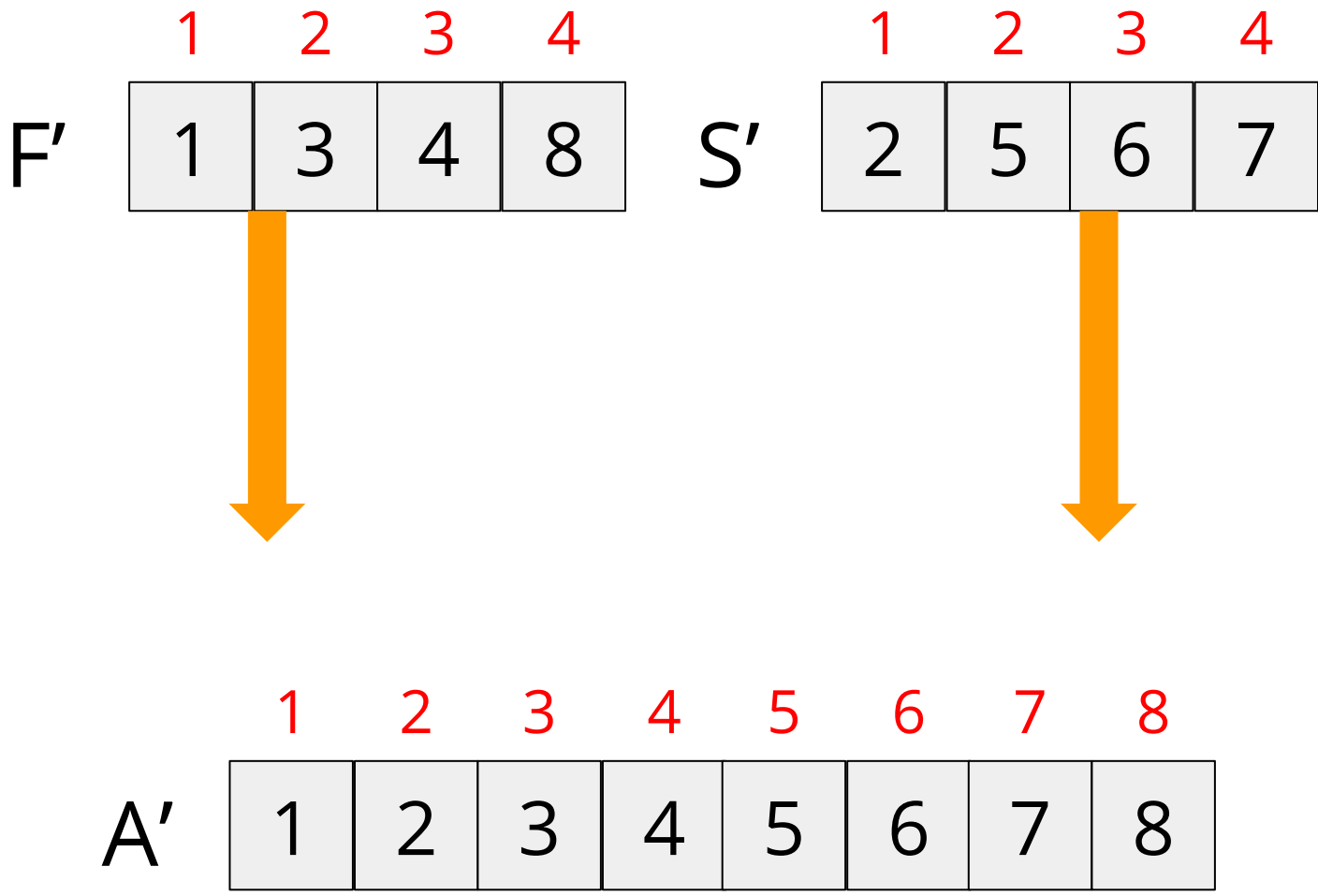
2. **Map** the solving of each sub-problem recursively.



Can We Do Better? An Algorithm Based Approach

MergeSort
Sorts Array A by...

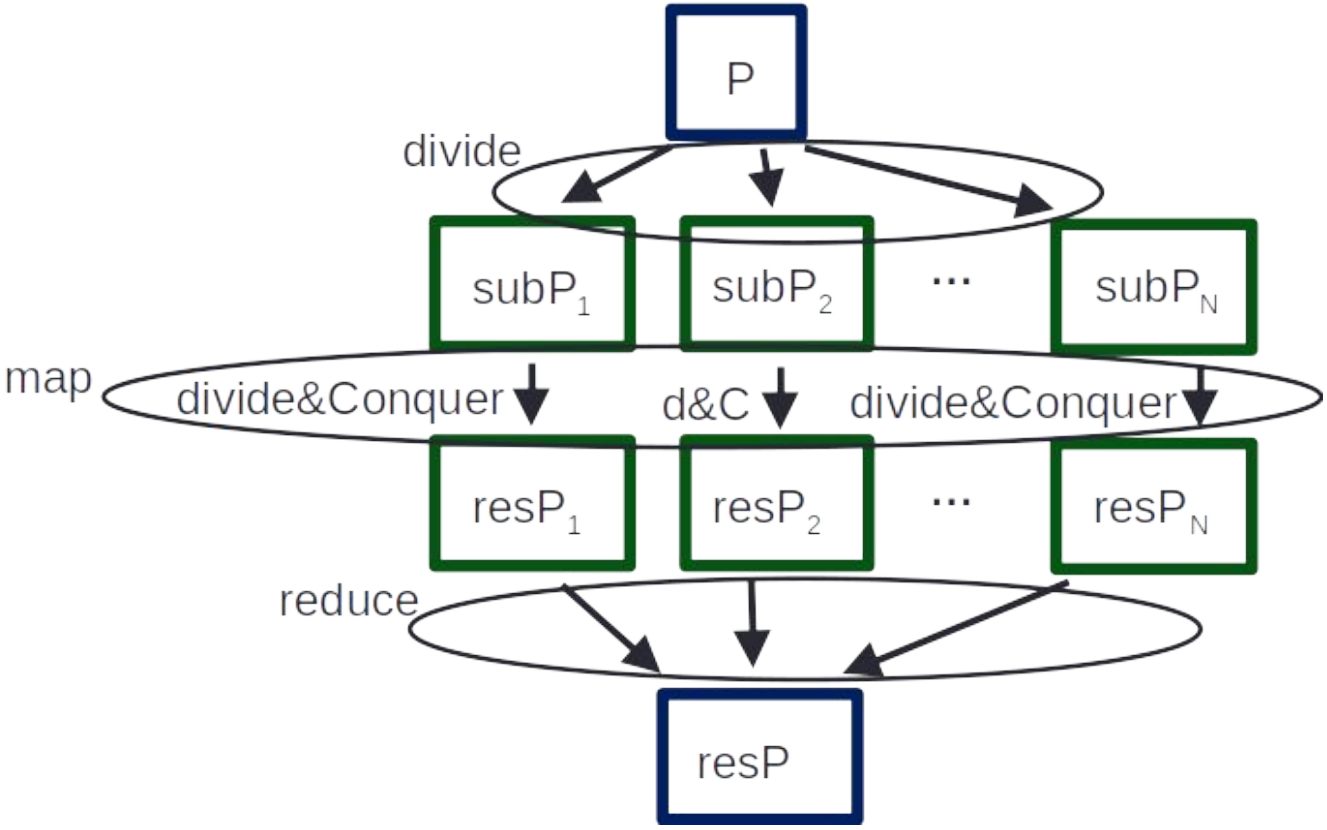
3. **Reduce** the solutions of the sub-problems to get the solution to the original problem.



Can We Do Better? An Algorithm Based Approach

MergeSort

As we can see, MergeSort is a divide and conquer-based algorithm!

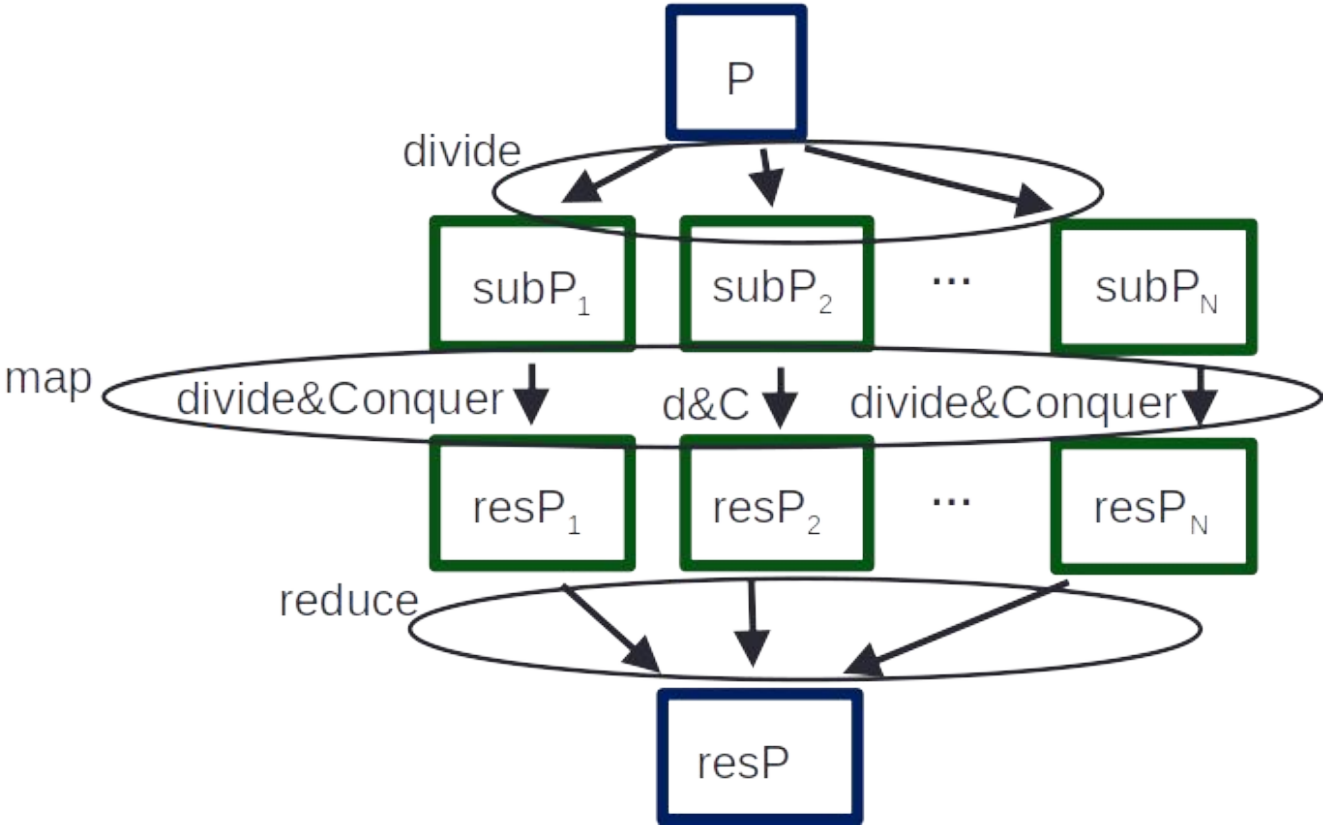


Can We Do Better? An Algorithm Based Approach

MergeSort

As we can see, MergeSort is a divide and conquer-based algorithm!

Please do not get confused with the meta-algorithm of the past section.



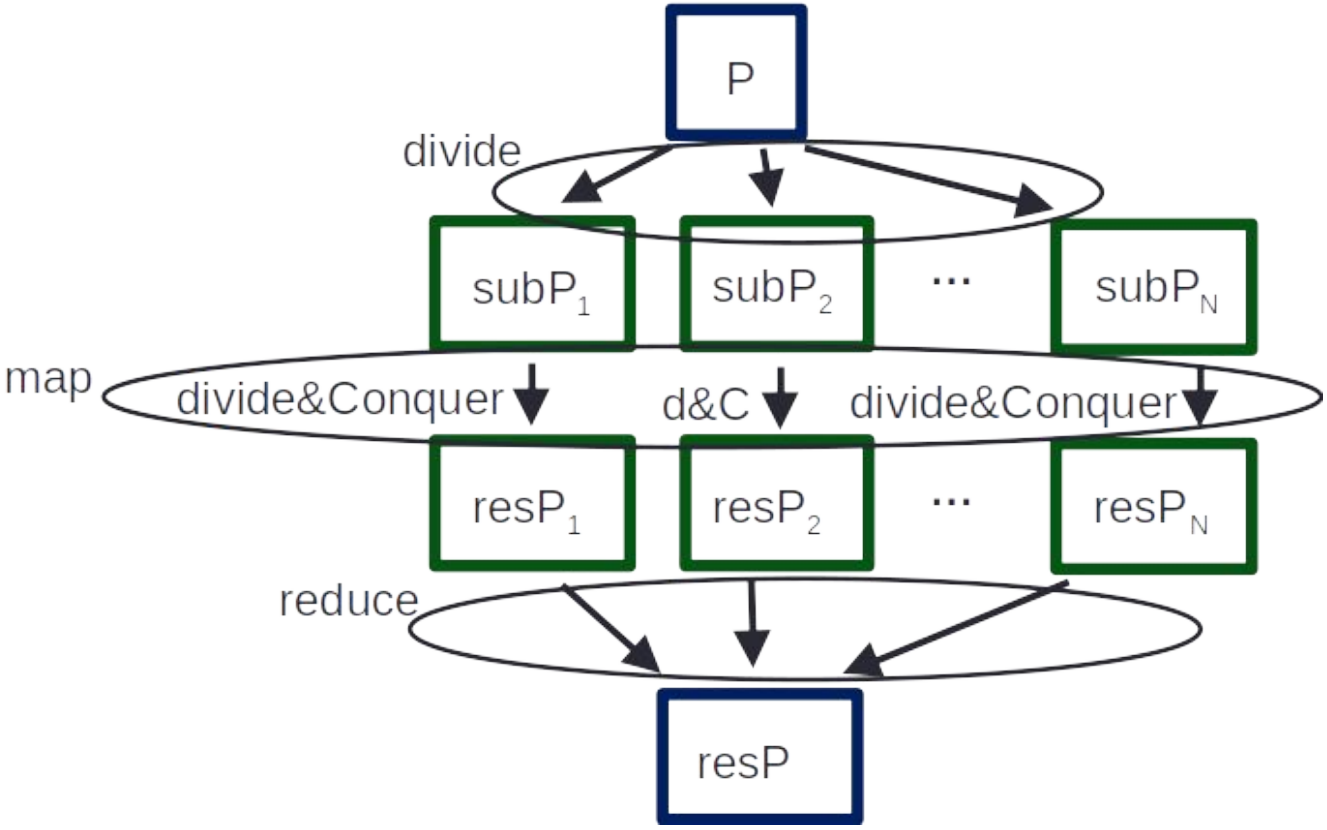
Can We Do Better? An Algorithm Based Approach

MergeSort

As we can see, MergeSort is a divide and conquer-based algorithm!

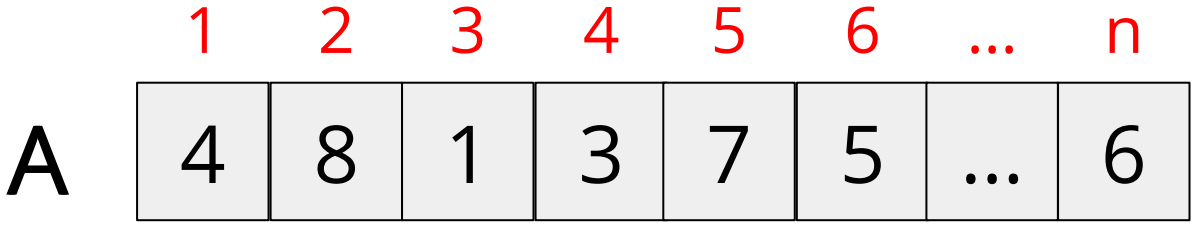
Please do not get confused with the meta-algorithm of the past section.

We are not going to run MergeSort in parallel! (although we could have done it).



Can We Do Better? An Algorithm Based Approach

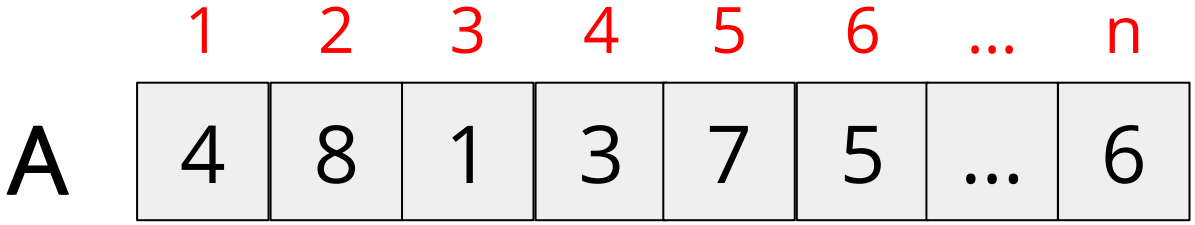
MergeSort



As for any divide and conquer-based algorithm, its time complexity can be determined by the Master Theorem.

Can We Do Better? An Algorithm Based Approach

MergeSort

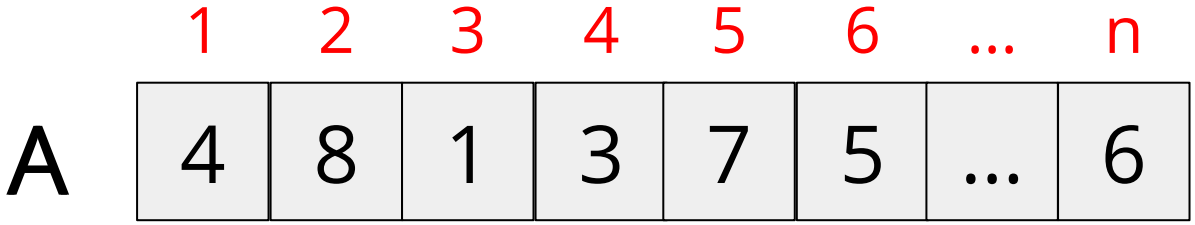


As for any divide and conquer-based algorithm, its time complexity can be determined by the Master Theorem.

To do so, we need to follow the expression **$T(n) = a * T(n/b) + O(n^d)$**, where:

Can We Do Better? An Algorithm Based Approach

MergeSort



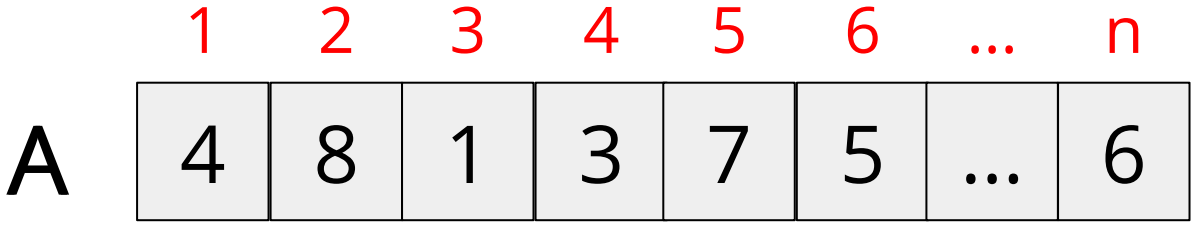
As for any divide and conquer-based algorithm, its time complexity can be determined by the Master Theorem.

To do so, we need to follow the expression **T(n) = a * T(n/b) + O(n^d)**, where:

- a => Number of sub-problems created by each recursive call.

Can We Do Better? An Algorithm Based Approach

MergeSort



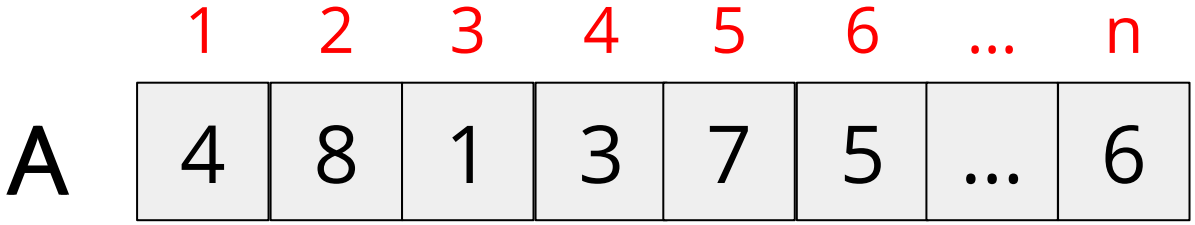
As for any divide and conquer-based algorithm, its time complexity can be determined by the Master Theorem.

To do so, we need to follow the expression **$T(n) = a * T(n/b) + O(n^d)$**, where:

- a => Number of sub-problems created by each recursive call.
- b => How smaller n/b is each sub-problem (w.r.t. the original problem of size n).

Can We Do Better? An Algorithm Based Approach

MergeSort



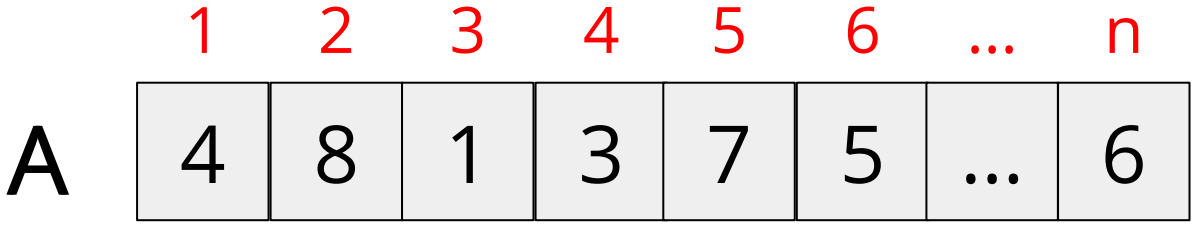
As for any divide and conquer-based algorithm, its time complexity can be determined by the Master Theorem.

To do so, we need to follow the expression **T(n) = a * T(n/b) + O(n^d)**, where:

- a => Number of sub-problems created by each recursive call.
- b => How smaller n/b is each sub-problem (w.r.t. the original problem of size n).
- d => What is the cost of the **reduce** stage in terms of **O(n^d)**.

Can We Do Better? An Algorithm Based Approach

MergeSort



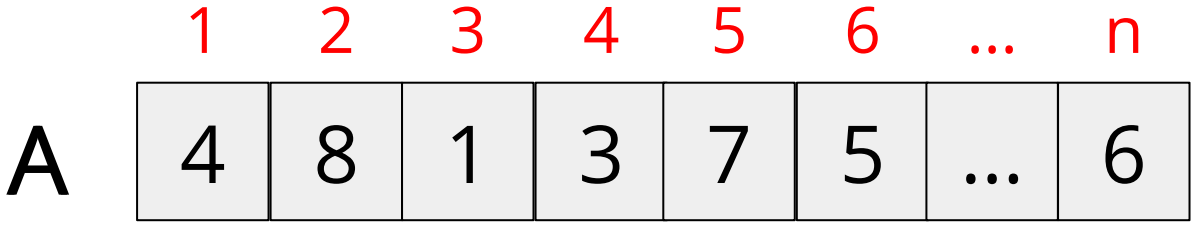
As for any divide and conquer-based algorithm, its time complexity can be determined by the Master Theorem.

Depending on the values of **a**, **b** and **d**, the algorithm complexity will be:

- If ($a == b^d$)
  This means that the same work is being done on each recursive level.

Can We Do Better? An Algorithm Based Approach

MergeSort



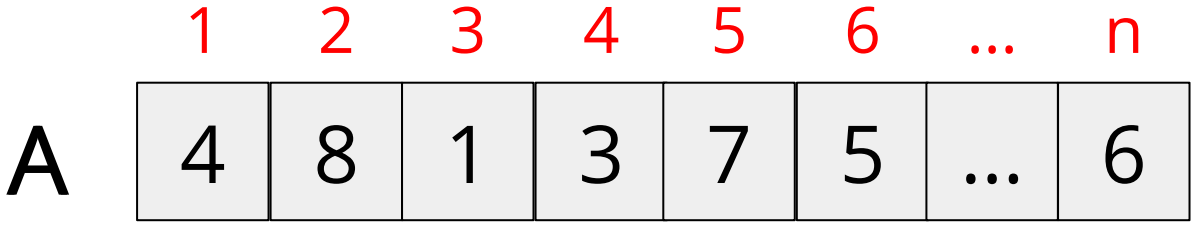
As for any divide and conquer-based algorithm, its time complexity can be determined by the Master Theorem.

Depending on the values of **a**, **b** and **d**, the algorithm complexity will be:

- If ($a == b^d$) then $T(n) = (n^d) * \log n$
  This means that the same work is being done on each recursive level.

Can We Do Better? An Algorithm Based Approach

MergeSort



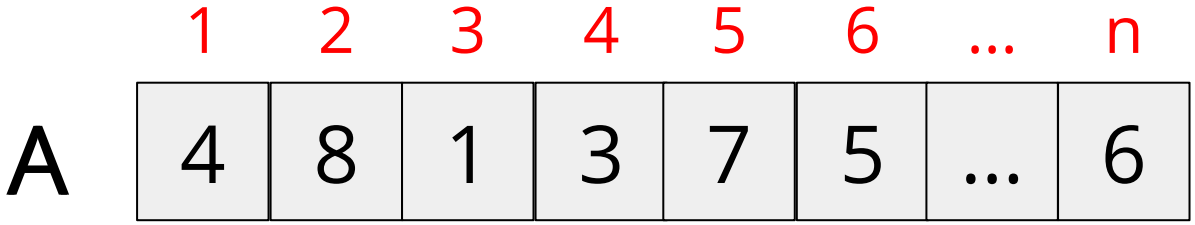
As for any divide and conquer-based algorithm, its time complexity can be determined by the Master Theorem.

Depending on the values of **a**, **b** and **d**, the algorithm complexity will be:

- If ($a == b^d$) then $T(n) = (n^d) * \log n$
  This means that the same work is being done on each recursive level.
- If ($a < b^d$)
  This means that less work is being done as we go down in the recursion tree.

Can We Do Better? An Algorithm Based Approach

MergeSort



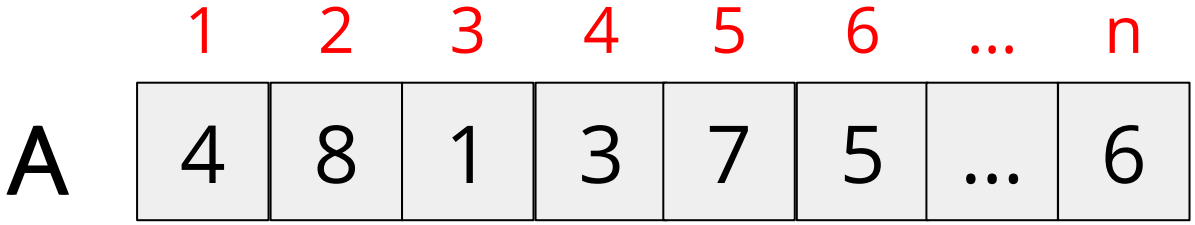
As for any divide and conquer-based algorithm, its time complexity can be determined by the Master Theorem.

Depending on the values of **a**, **b** and **d**, the algorithm complexity will be:

- If ($a == b^d$) then $T(n) = (n^d) * \log n$
  This means that the same work is being done on each recursive level.
- If ($a < b^d$) then $T(n) = (n^d)$
  This means that less work is being done as we go down in the recursion tree.

Can We Do Better? An Algorithm Based Approach

MergeSort



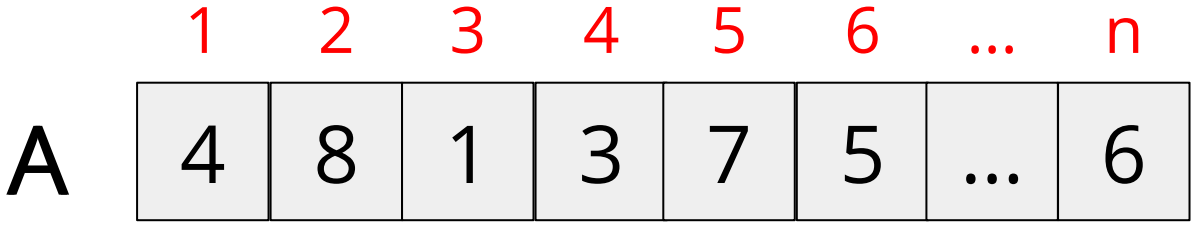
As for any divide and conquer-based algorithm, its time complexity can be determined by the Master Theorem.

Depending on the values of **a**, **b** and **d**, the algorithm complexity will be:

- If ($a == b^d$) then $T(n) = (n^d) * \log n$
  This means the same work is being done on each recursive level.
- If ($a < b^d$) then $T(n) = (n^d)$
  This means less work is being done as we go down in the recursion tree.
- If ($a > b^d$)
  This means more work is being done as we go down in the recursion tree.

Can We Do Better? An Algorithm Based Approach

MergeSort



As for any divide and conquer-based algorithm, its time complexity can be determined by the Master Theorem.

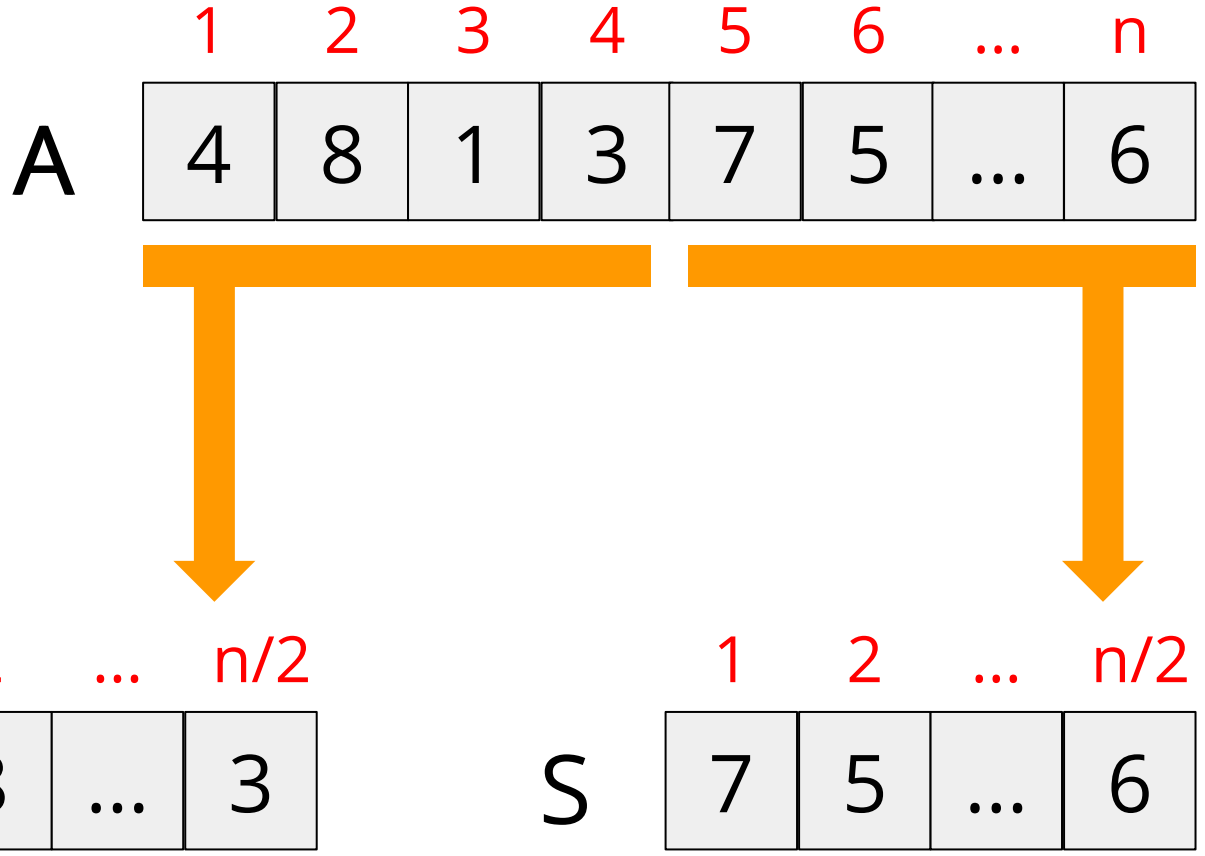
Depending on the values of **a**, **b** and **d**, the algorithm complexity will be:

- If ($a == b^d$) then $T(n) = (n^d) * \log n$
  This means the same work is being done on each recursive level.
- If ($a < b^d$) then $T(n) = (n^d)$
  This means less work is being done as we go down in the recursion tree.
- If ($a > b^d$) then $T(n) = (n^{\log_b a})$
  This means more work is being done as we go down in the recursion tree.

Can We Do Better? An Algorithm Based Approach

MergeSort

By looking at the divide stage it is clear that:



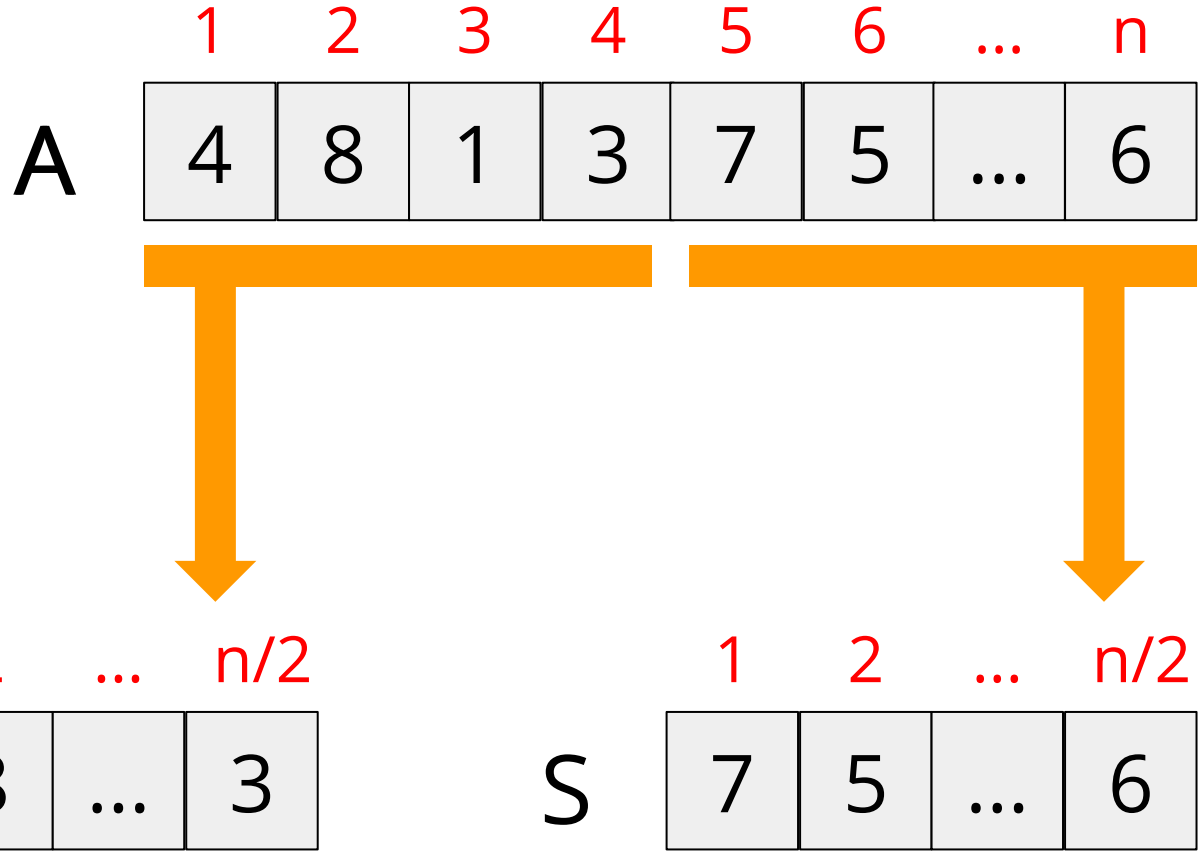
Can We Do Better? An Algorithm Based Approach

MergeSort

By looking at the divide stage it is clear that:

- **a = 2**

Given an original problem, we divide it into two problems.



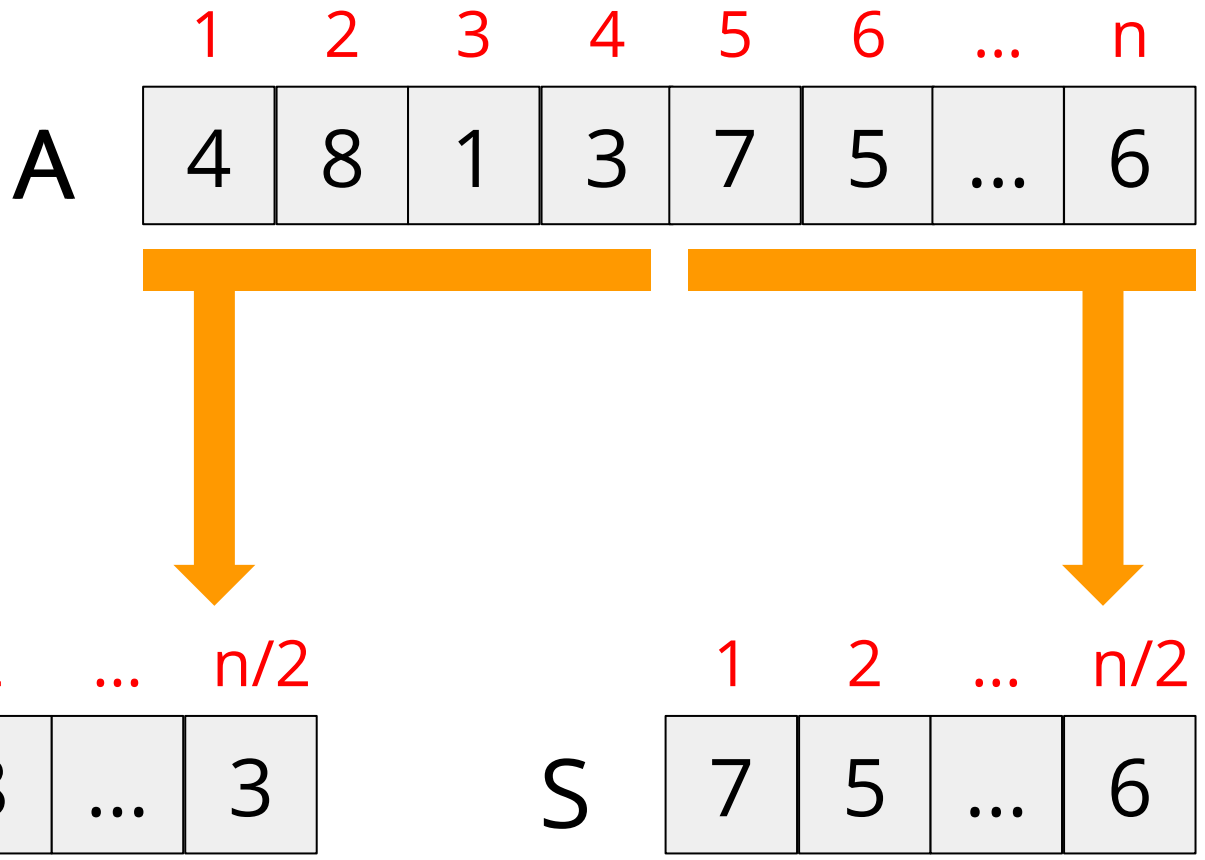
Can We Do Better? An Algorithm Based Approach

MergeSort

By looking at the divide stage it is clear that:

- **b = 2**

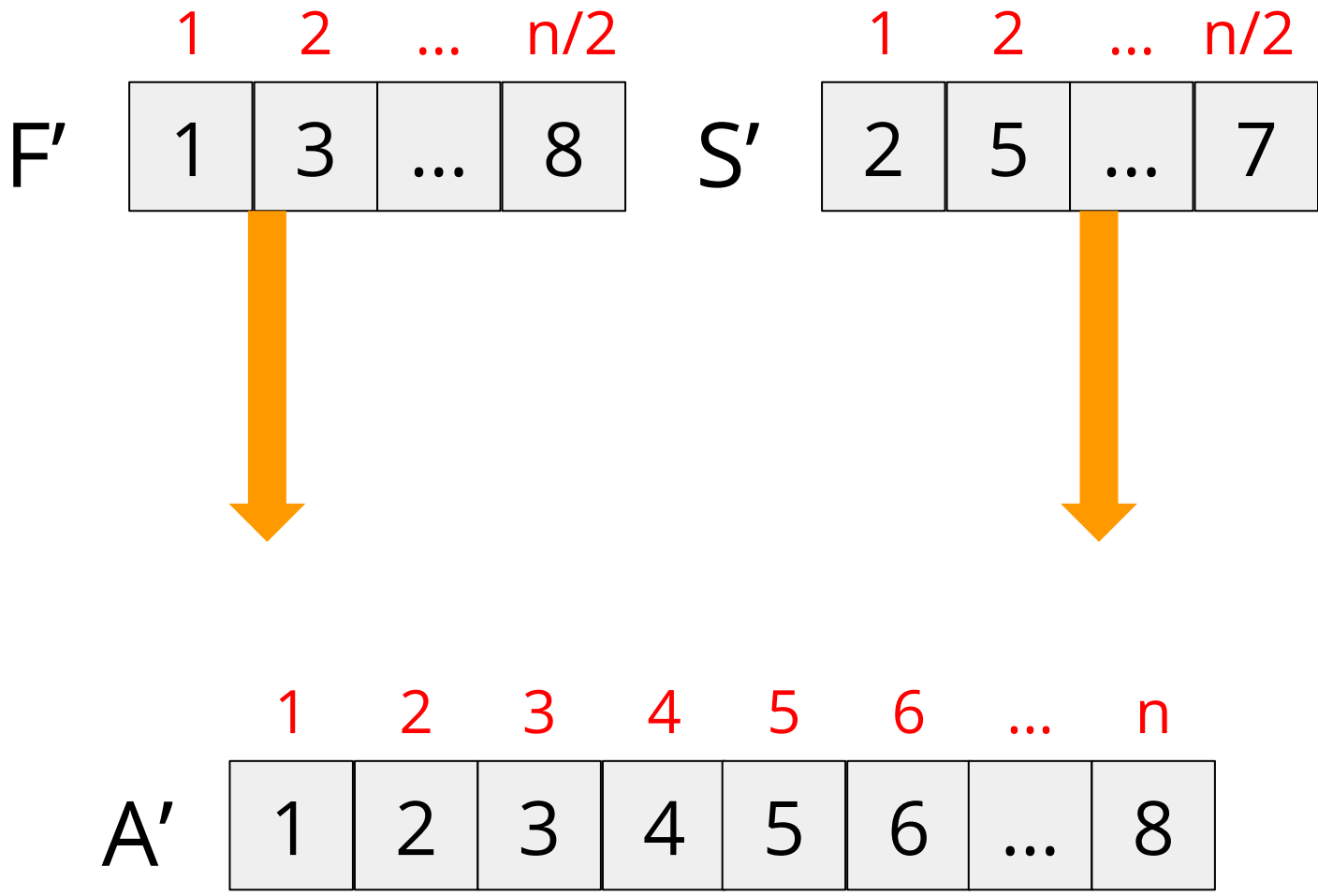
Given an original problem of size n, each sub-problem is of size (n/b). In our case, each sub-problem is half the size of the original problem.



Can We Do Better? An Algorithm Based Approach

MergeSort

To get the value of **d** we need to analyse the cost of our reduce stage $O(n^d)$.

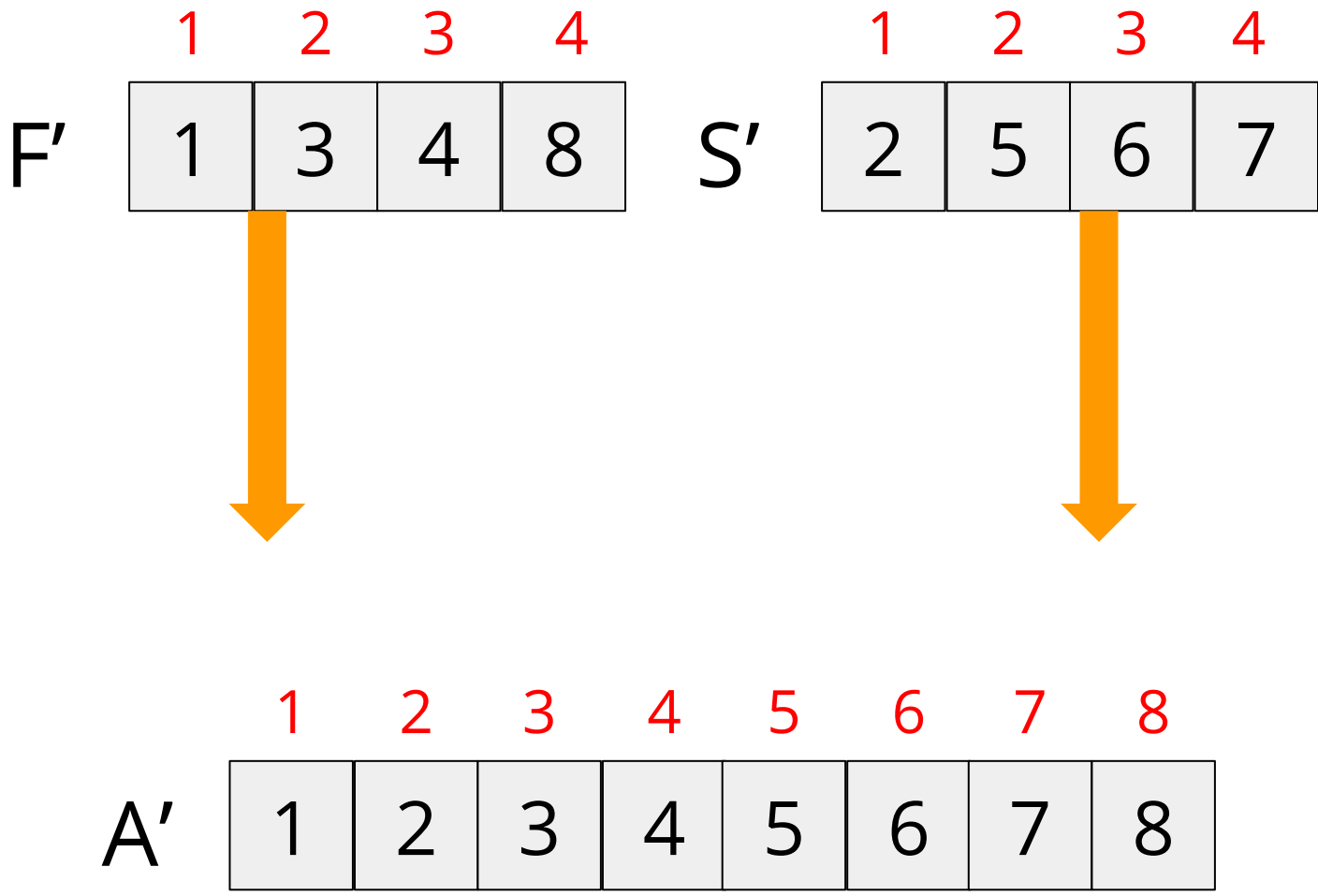


Can We Do Better? An Algorithm Based Approach

MergeSort

To get the value of **d** we need to analyse the cost of our reduce stage $O(n^d)$.

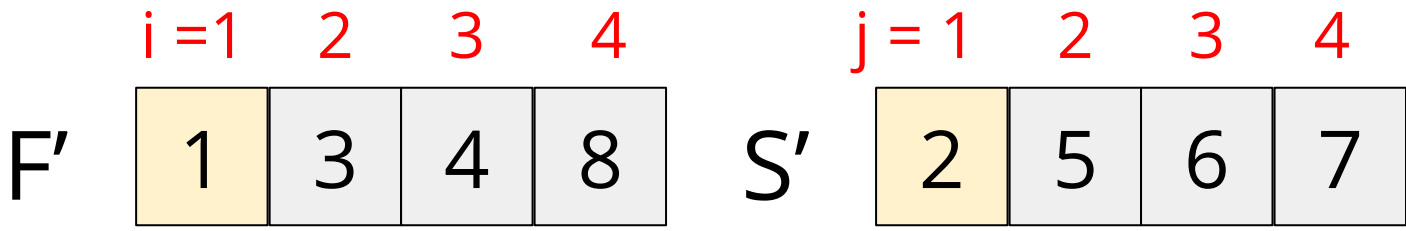
Let's study it for the array A we are using in our example.



Can We Do Better? An Algorithm Based Approach

MergeSort

To get the value of **d** we need to analyse the cost of our reduce stage $O(n^d)$.



We start with:
- An empty array A'.
- An index **i** at the beginning of F'.
- An index **j** at the beginning of S'.

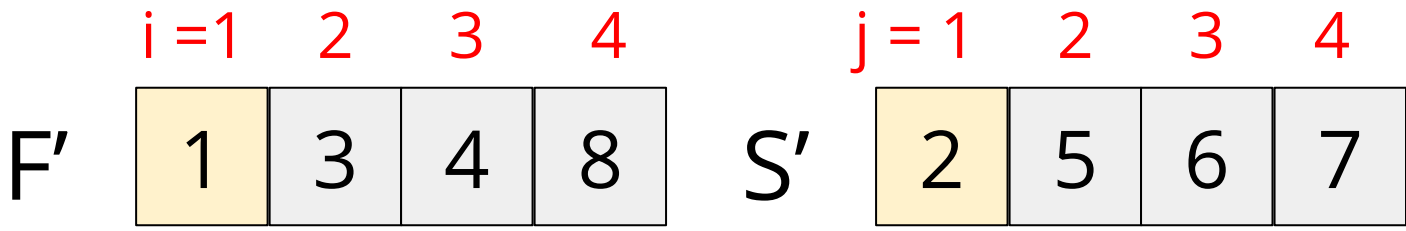
A'

Can We Do Better? An Algorithm Based Approach

MergeSort

To get the value of **d** we need to analyse the cost of our reduce stage $O(n^d)$.

- We compare the positions i and j. As both F' and S' are already sorted, whoever is the smallest of them has also to be the smallest element of F' and S'.



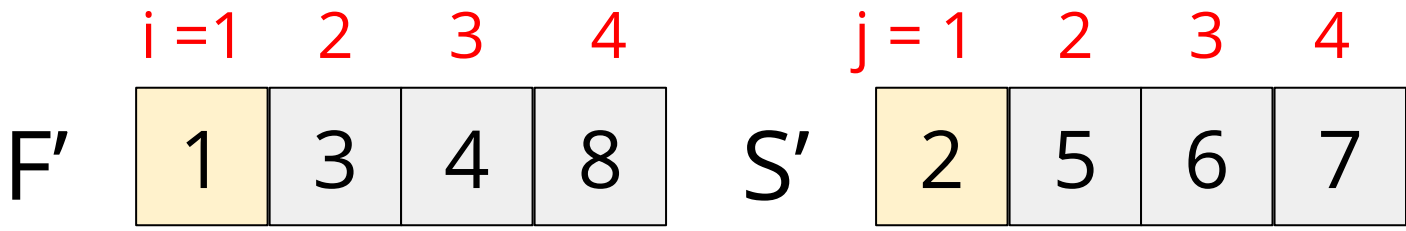
A'

Can We Do Better? An Algorithm Based Approach

MergeSort

To get the value of **d** we need to analyse the cost of our reduce stage $O(n^d)$.

- We compare the positions i and j. As both F' and S' are already sorted, whoever is the smallest of them has also to be the smallest element of F' and S'.



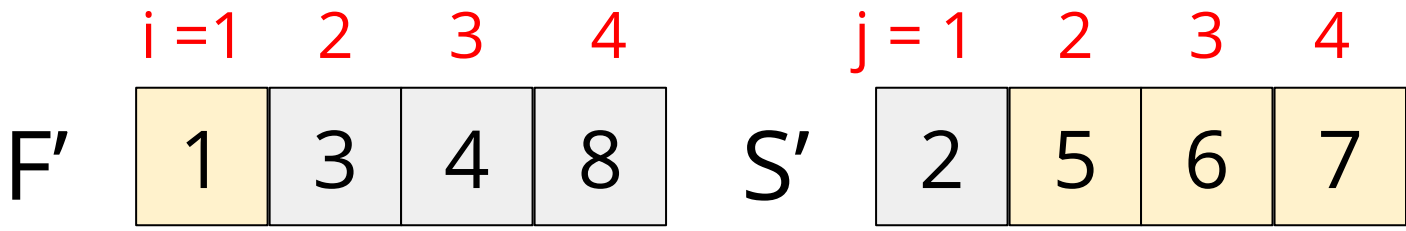
- F[i] < S'[j].

A'

Can We Do Better? An Algorithm Based Approach

MergeSort
To get the value of **d** we need to analyse the cost of our reduce stage $O(n^d)$.

- We compare the positions i and j.
As both F' and S' are already sorted, whoever is the smallest of them has also to be the smallest element of F' and S'.



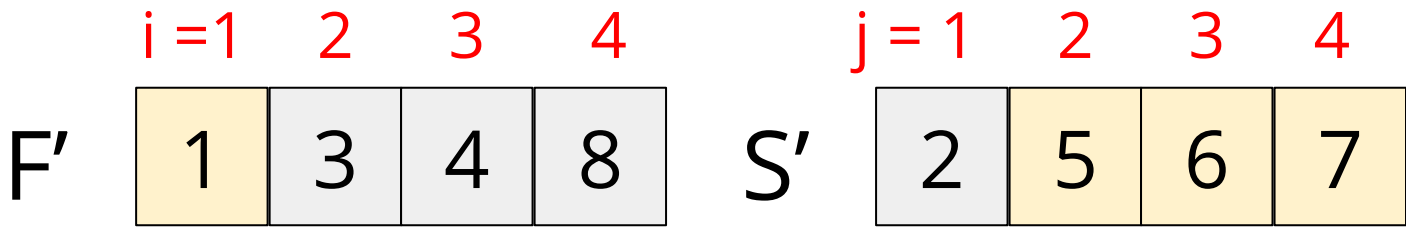
- F[i] < S'[j].
- But, as F' is in order, this also means F'[i] will be smaller than all the other values of F'.

A'

Can We Do Better? An Algorithm Based Approach

MergeSort
To get the value of **d** we need to analyse the cost of our reduce stage $O(n^d)$.

- We compare the positions i and j.
As both F' and S' are already sorted, whoever is the smallest of them has also to be the smallest element of F' and S'.



- F[i] < S'[j].
- But, as F' is in order, this also means F'[i] will be smaller than all the other values of F'.
- But, as S' is in order, this also means F'[i] will be smaller than all the other values of S'.

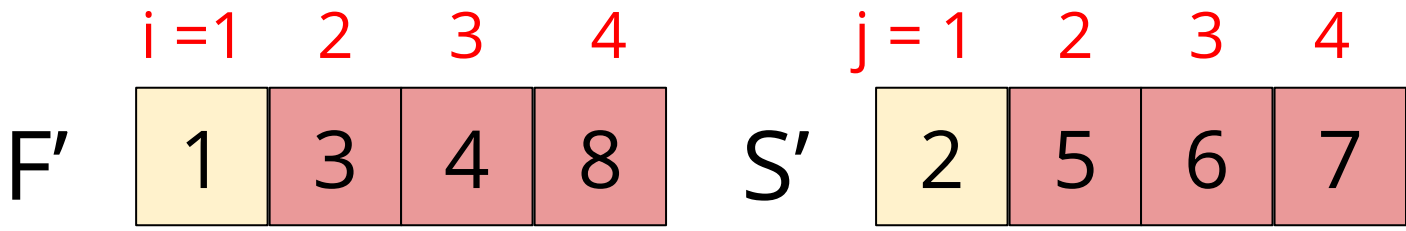
A'

Can We Do Better? An Algorithm Based Approach

MergeSort

To get the value of **d** we need to analyse the cost of our reduce stage $O(n^d)$.

- We compare the positions i and j.
As both F' and S' are already sorted, whoever is the smallest of them has also to be the smallest element of F' and S'.



- F[i] < S'[j].
- But, as F' is in order, this also means F'[i] will be smaller than all the other values of F'.
- But, as S' is in order, this also means F[i] will be smaller than all the other values of S'.
- Interestingly, to reach to this conclusion we don't need to compare F'[i] with all the remaining values of F' nor of S', just with S'[j].

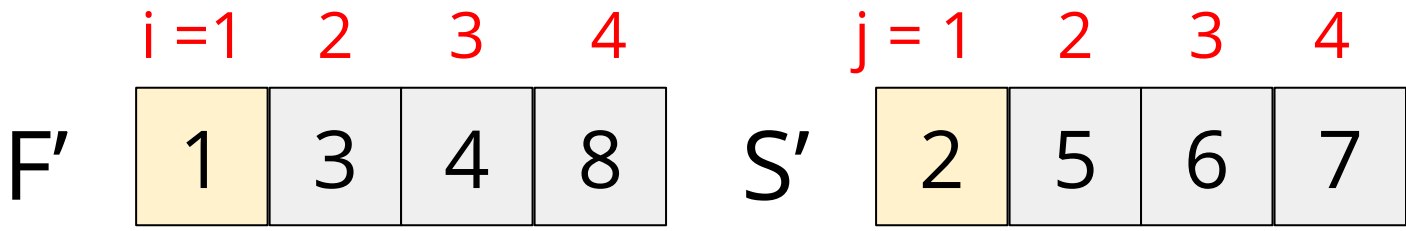
A'

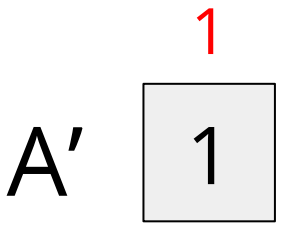
Can We Do Better? An Algorithm Based Approach

MergeSort

To get the value of **d** we need to analyse the cost of our reduce stage $O(n^d)$.

- We add the smallest of F'[i] and S'[j] to A'.



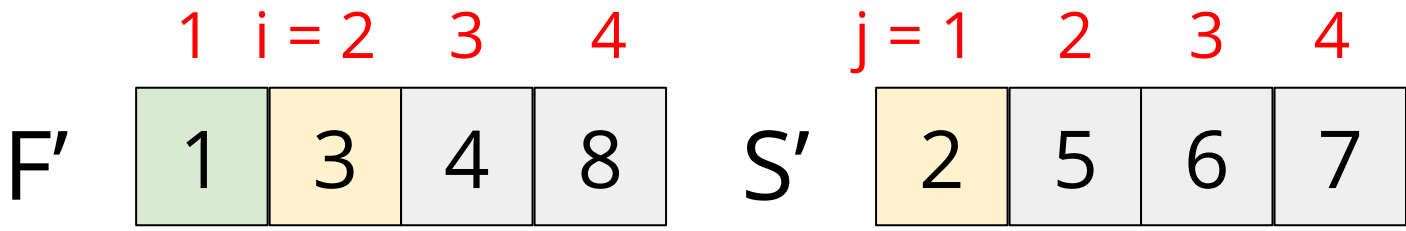


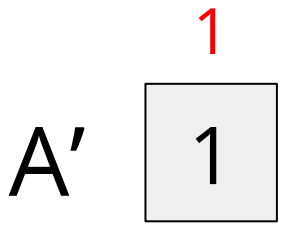
Can We Do Better? An Algorithm Based Approach

MergeSort

To get the value of **d** we need to analyse the cost of our reduce stage $O(n^d)$.

- We increase i or j accordingly.



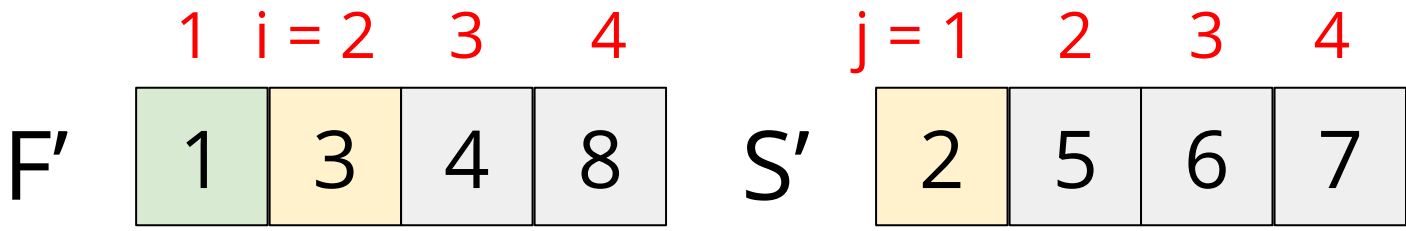


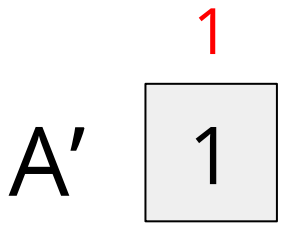
Can We Do Better? An Algorithm Based Approach

MergeSort

To get the value of **d** we need to analyse the cost of our reduce stage $O(n^d)$.

- And we repeat these steps until all elements are added to A'.



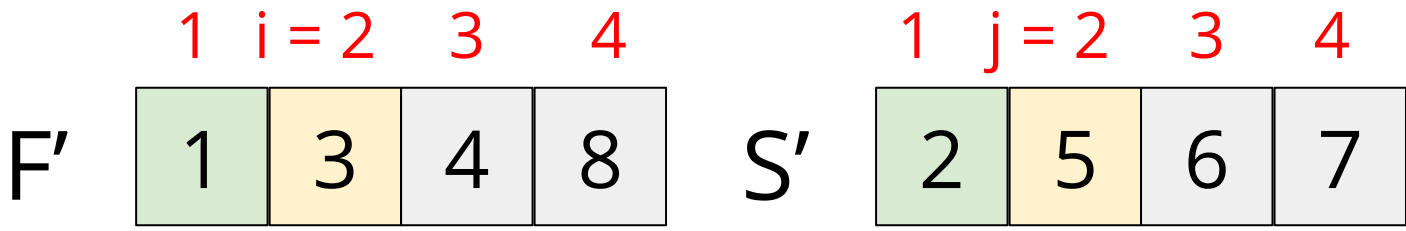


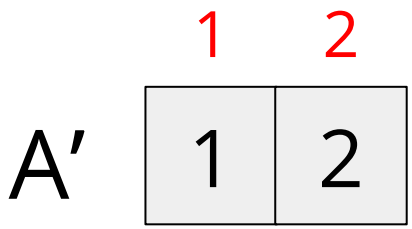
Can We Do Better? An Algorithm Based Approach

MergeSort

To get the value of **d** we need to analyse the cost of our reduce stage $O(n^d)$.

- And we repeat these steps until all elements are added to A'.



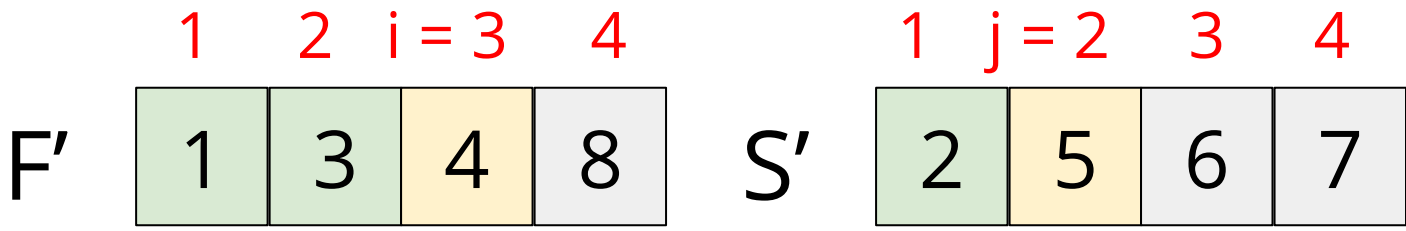


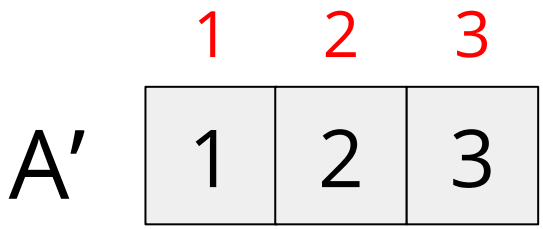
Can We Do Better? An Algorithm Based Approach

MergeSort

To get the value of **d** we need to analyse the cost of our reduce stage $O(n^d)$.

- And we repeat these steps until all elements are added to A'.



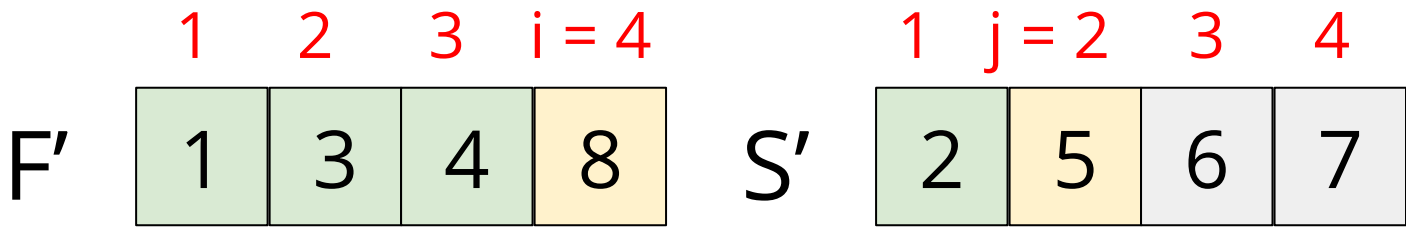


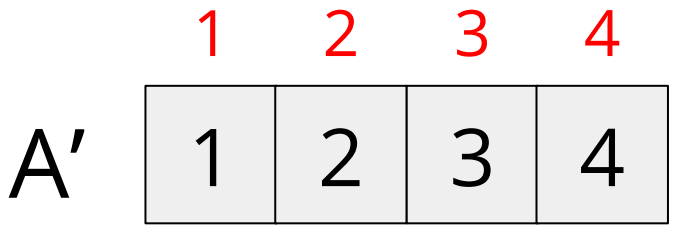
Can We Do Better? An Algorithm Based Approach

MergeSort

To get the value of **d** we need to analyse the cost of our reduce stage $O(n^d)$.

- And we repeat these steps until all elements are added to A'.



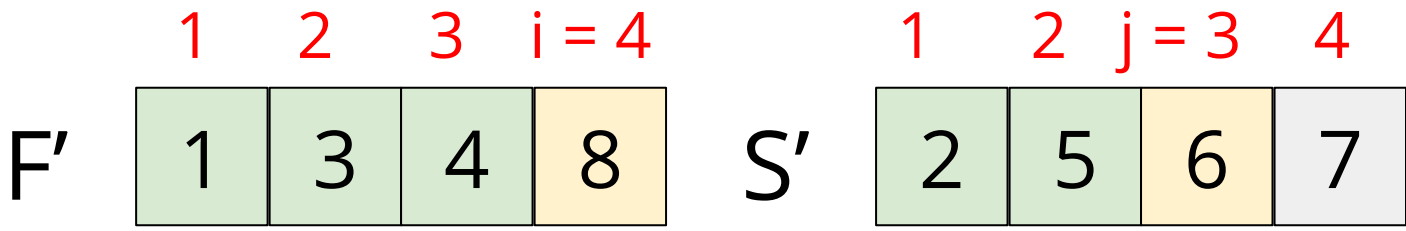


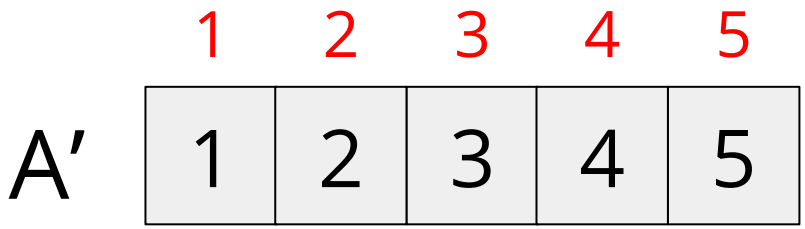
Can We Do Better? An Algorithm Based Approach

MergeSort

To get the value of **d** we need to analyse the cost of our reduce stage $O(n^d)$.

- And we repeat these steps until all elements are added to A'.



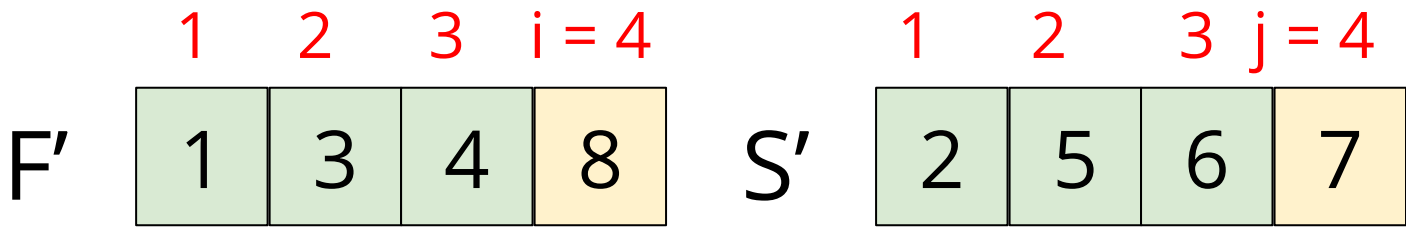


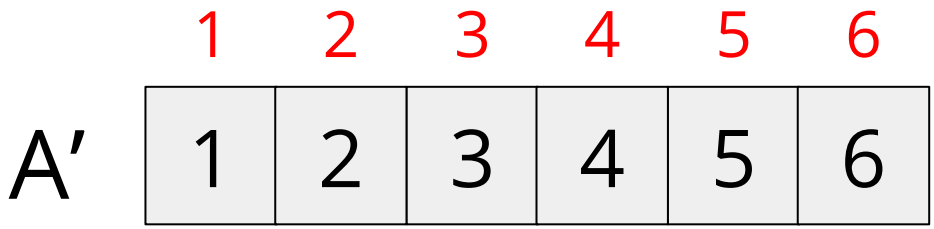
Can We Do Better? An Algorithm Based Approach

MergeSort

To get the value of **d** we need to analyse the cost of our reduce stage $O(n^d)$.

- And we repeat these steps until all elements are added to A'.



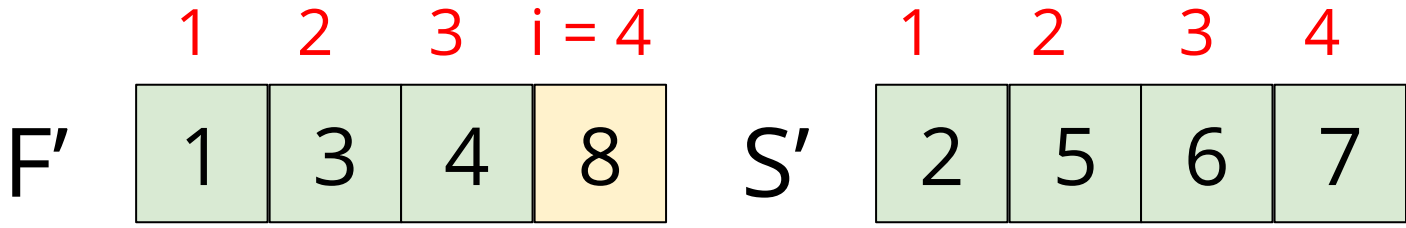


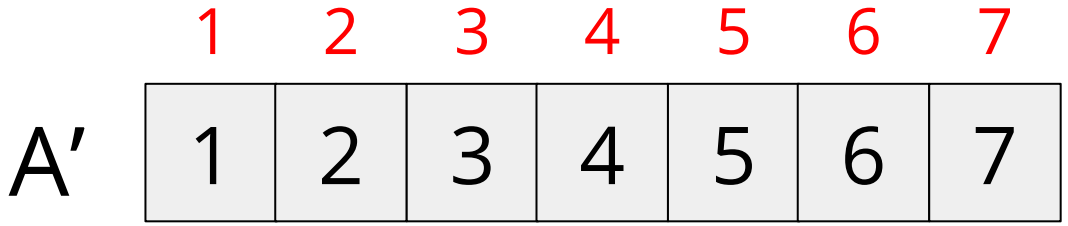
Can We Do Better? An Algorithm Based Approach

MergeSort

To get the value of **d** we need to analyse the cost of our reduce stage $O(n^d)$.

- And we repeat these steps until all elements are added to A'.





Can We Do Better? An Algorithm Based Approach

MergeSort

To get the value of **d** we need to analyse the cost of our reduce stage $O(n^d)$.

- And we repeat these steps until all elements are added to A'.

F'

1	2	3	4
1	3	4	8

S'

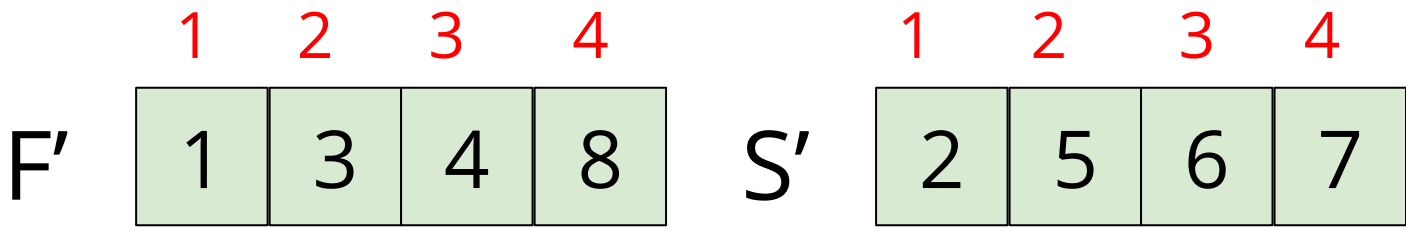
1	2	3	4
2	5	6	7

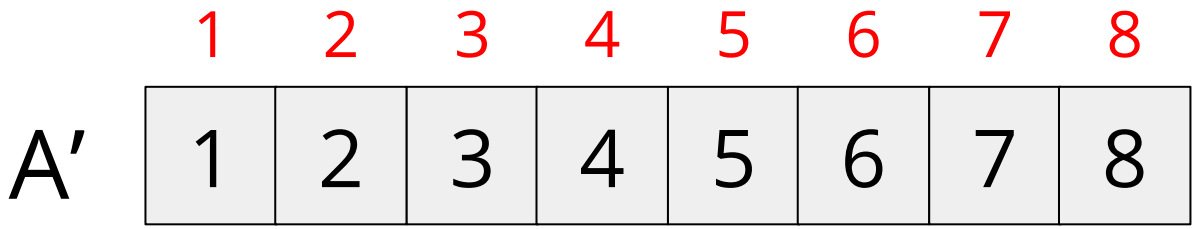
A'

1	2	3	4	5	6	7	8
1	2	3	4	5	6	7	8

Can We Do Better? An Algorithm Based Approach

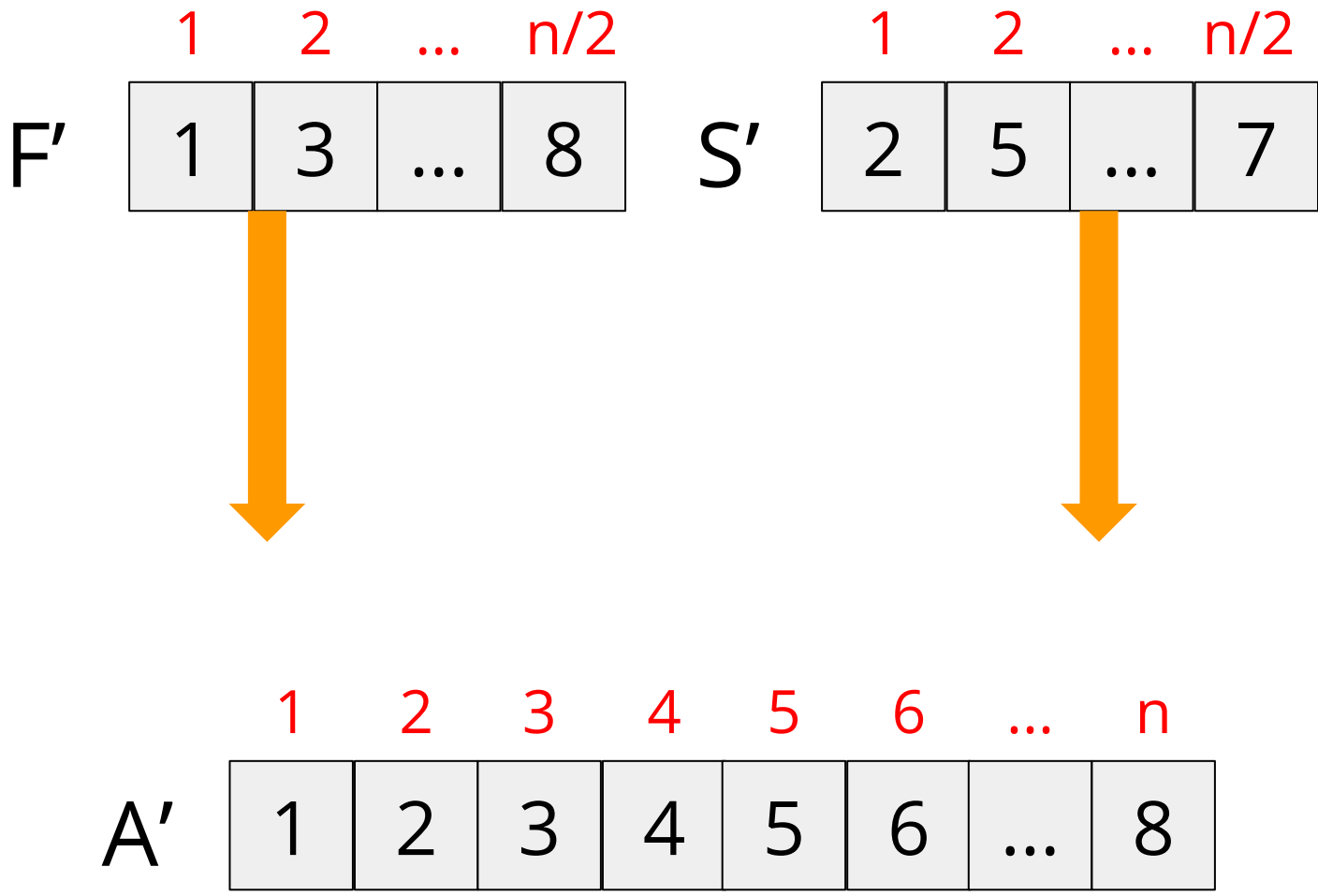
MergeSort
In our case, the 2 sub-problems of size 4 lead to a new array A' of size 8 by making...
8 comparissons!





Can We Do Better? An Algorithm Based Approach

MergeSort
In general the 2 sub-problems of size n/2 lead to a new array A' of size n by making...
n comparissons!

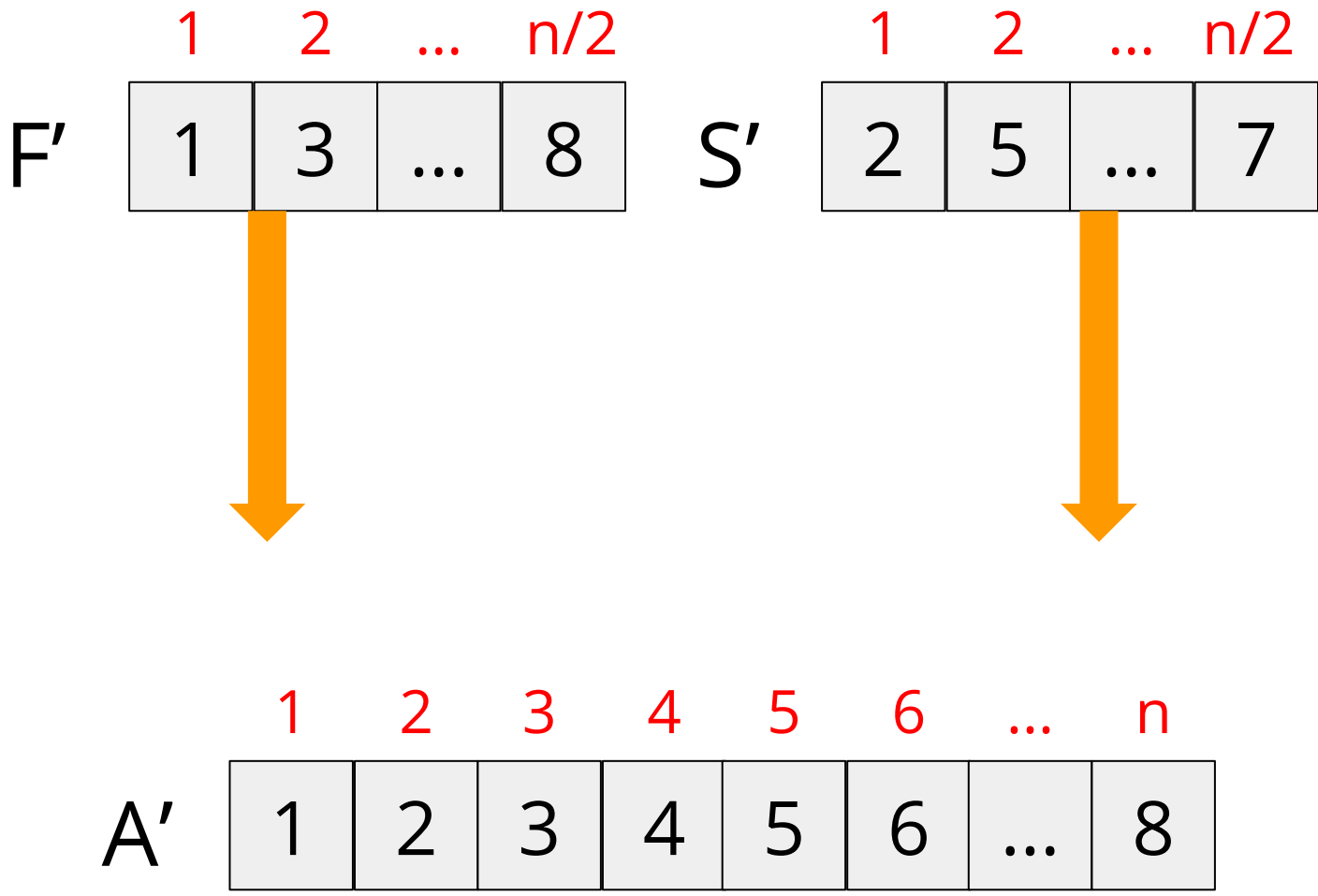


Can We Do Better? An Algorithm Based Approach

MergeSort

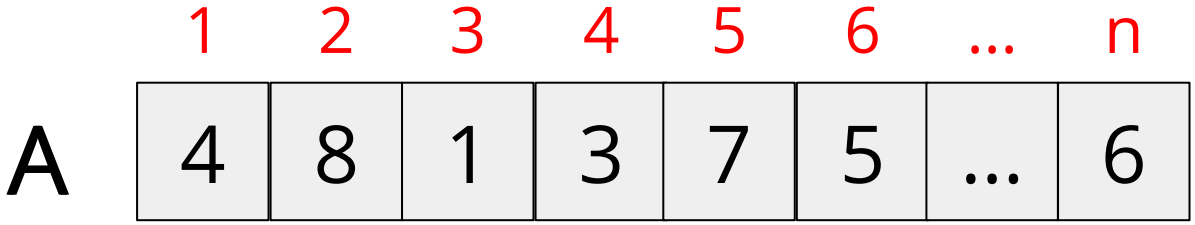
Thus, the cost of our reduce stage is O(n), which according to $O(n^d)$ makes:

- **d = 1**



Can We Do Better? An Algorithm Based Approach

MergeSort



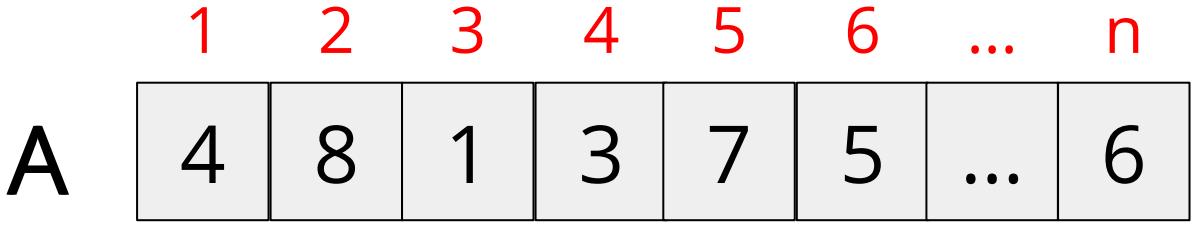
As for any divide and conquer-based algorithm, its time complexity can be determined by the Master Theorem.

Thus, given that **a = 2**, **b = 2** and **d = 1**, we are in this case where ($a == b^d$):

- If ($a == b^d$) then $T(n) = (n^d) * \log n$
  This means the same work is being done on each recursive level.
- If ($a < b^d$) then $T(n) = (n^d)$
  This means less work is being done as we go down in the recursion tree.
- If ($a > b^d$) then $T(n) = (n^{\log_b a})$
  This means more work is being done as we go down in the recursion tree.

Can We Do Better? An Algorithm Based Approach

MergeSort



As for all divide and conquer-based algorithms, its time complexity can be determined by the Master Theorem.

Thus, given that **a = 2**, **b = 2** and **d = 1**, we are in this case where ($a == b^d$):

- If ($a == b^d$) then $T(n) = (n^d) * \log n$
  Which means the same work is being done on each recursive level.

- And, as d = 1 our $T(n) = (n^d) * \log n$ turns into **$T(n) = n * \log n$**

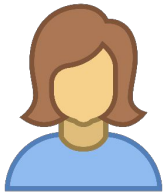
Can We Do Better? An Algorithm Based Approach

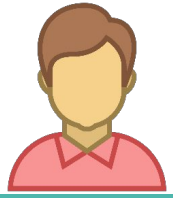
And this is why we say our algorithm MergeSort has time complexity O(n logn)

Can We Do Better? An Algorithm Based Approach

Cool!

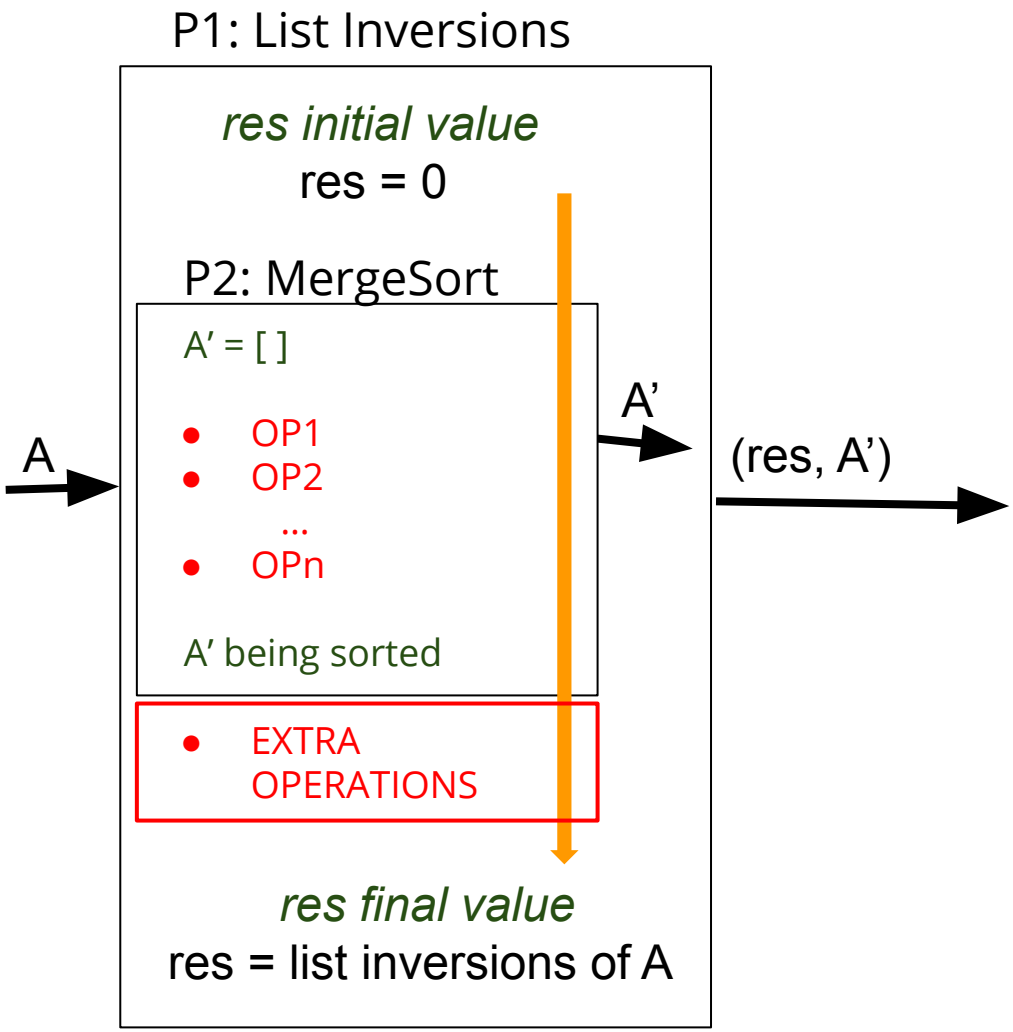
Let's focus now on how
List inversions reduces to MergeSort,
maintaining also time complexity O(n log n).

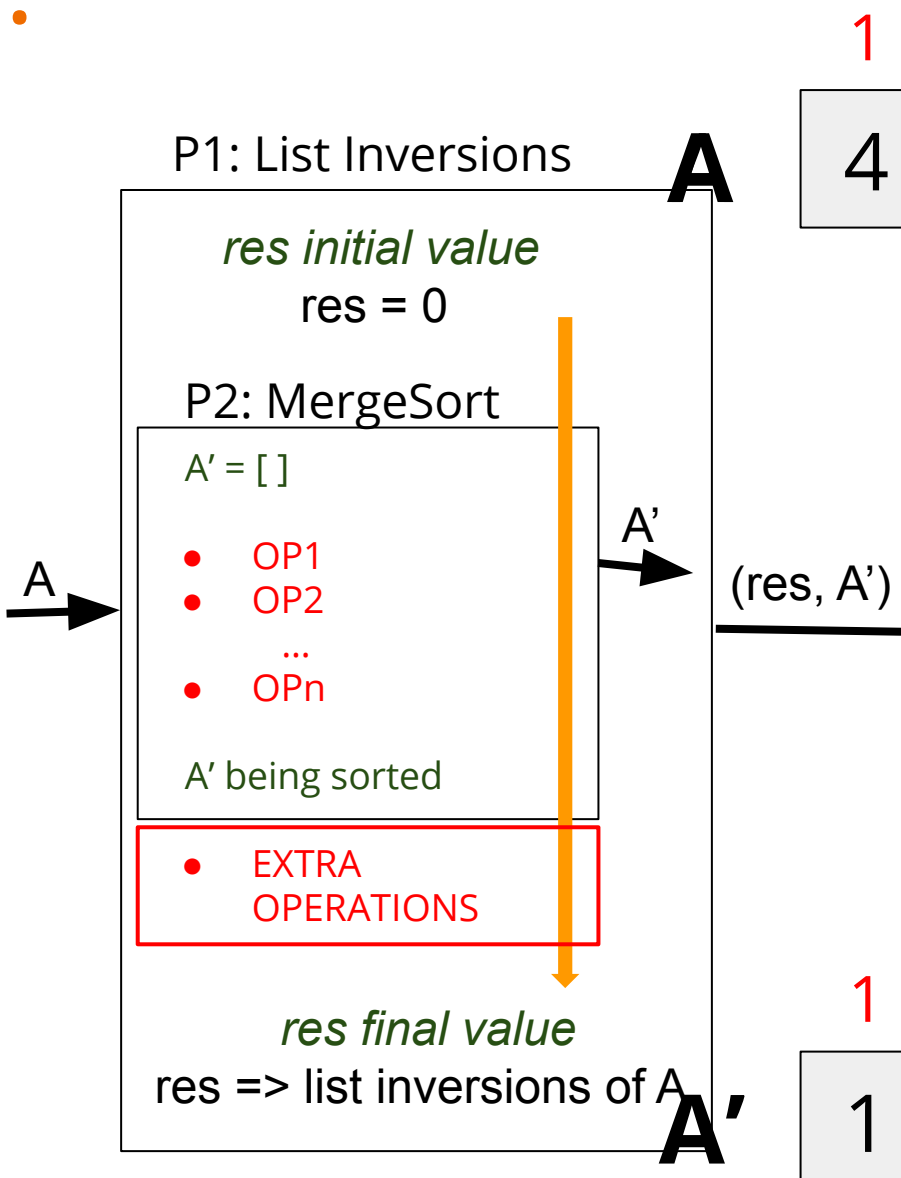




	1	2	3	4	5	6	7	8
A	4	8	1	3	7	5	2	6

Can We Do Better? An Algorithm Based Approach



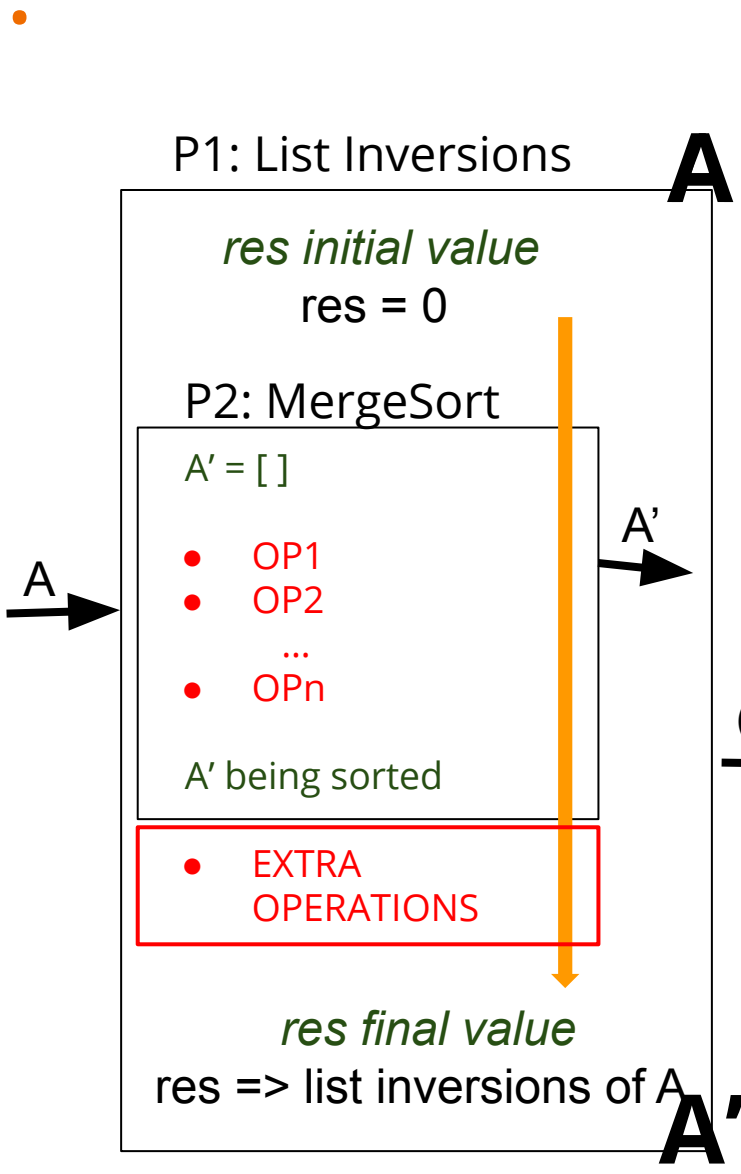


1	2	3	4	5	6	7	8
4	8	1	3	7	5	2	6

List Inversions

As we can see, the fact that MergeSort creates a new array A' with the content of A being sorted is...

1	2	3	4	5	6	7	8
1	2	3	4	5	6	7	8



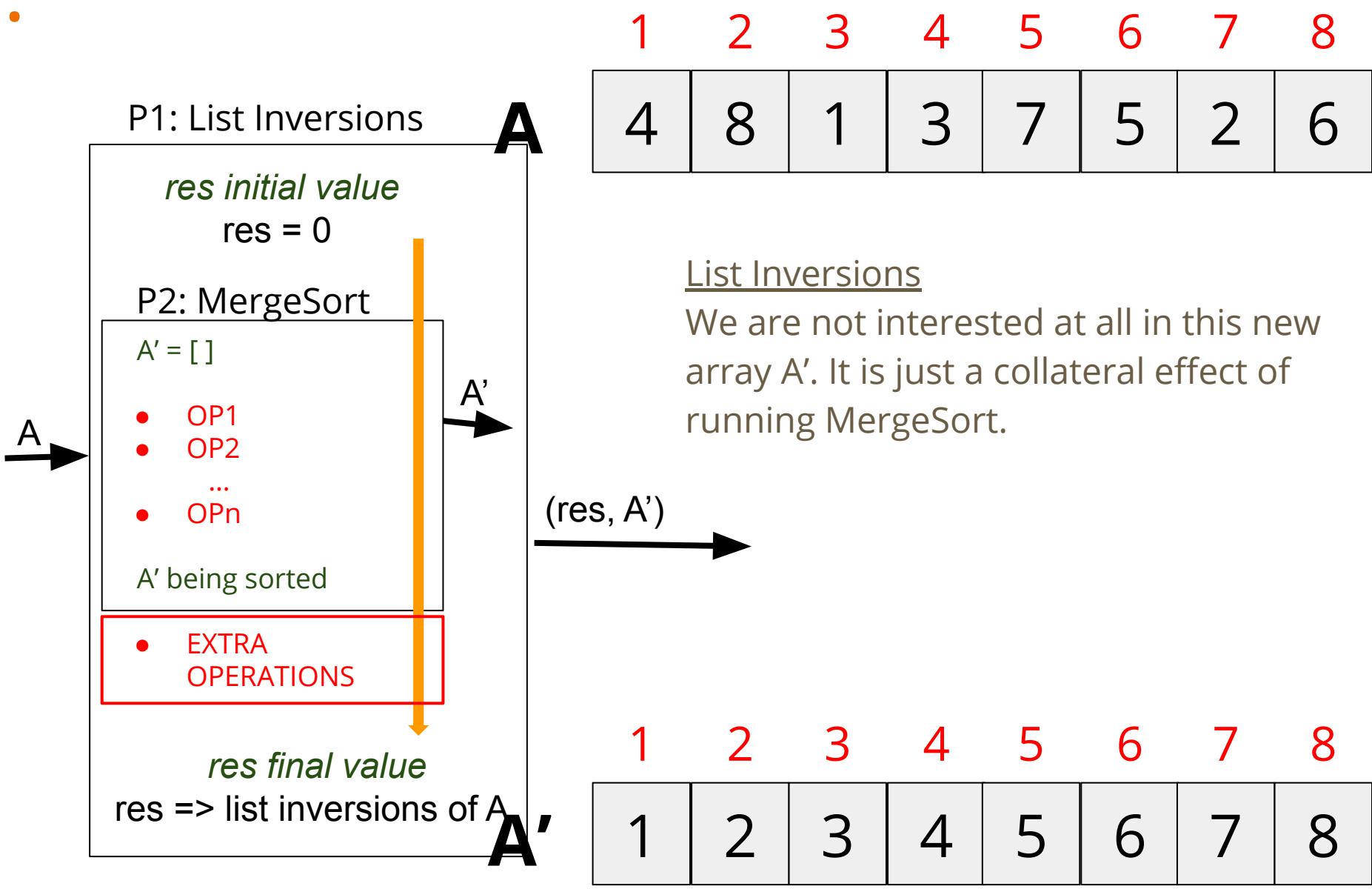
1	2	3	4	5	6	7	8
4	8	1	3	7	5	2	6

List Inversions

As we can see, the fact that MergeSort creates a new array A' with the content of A being sorted is...

...irrelevant to us!

1	2	3	4	5	6	7	8
1	2	3	4	5	6	7	8

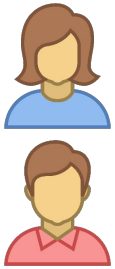


List Inversions

We are not interested at all in this new array A'. It is just a collateral effect of running MergeSort.

Can We Do Better? An Algorithm Based Approach

To understand how to compute res, let's come back to MergeSort.



A

1	2	3	4	5	6	7	8
4	8	1	3	7	5	2	6

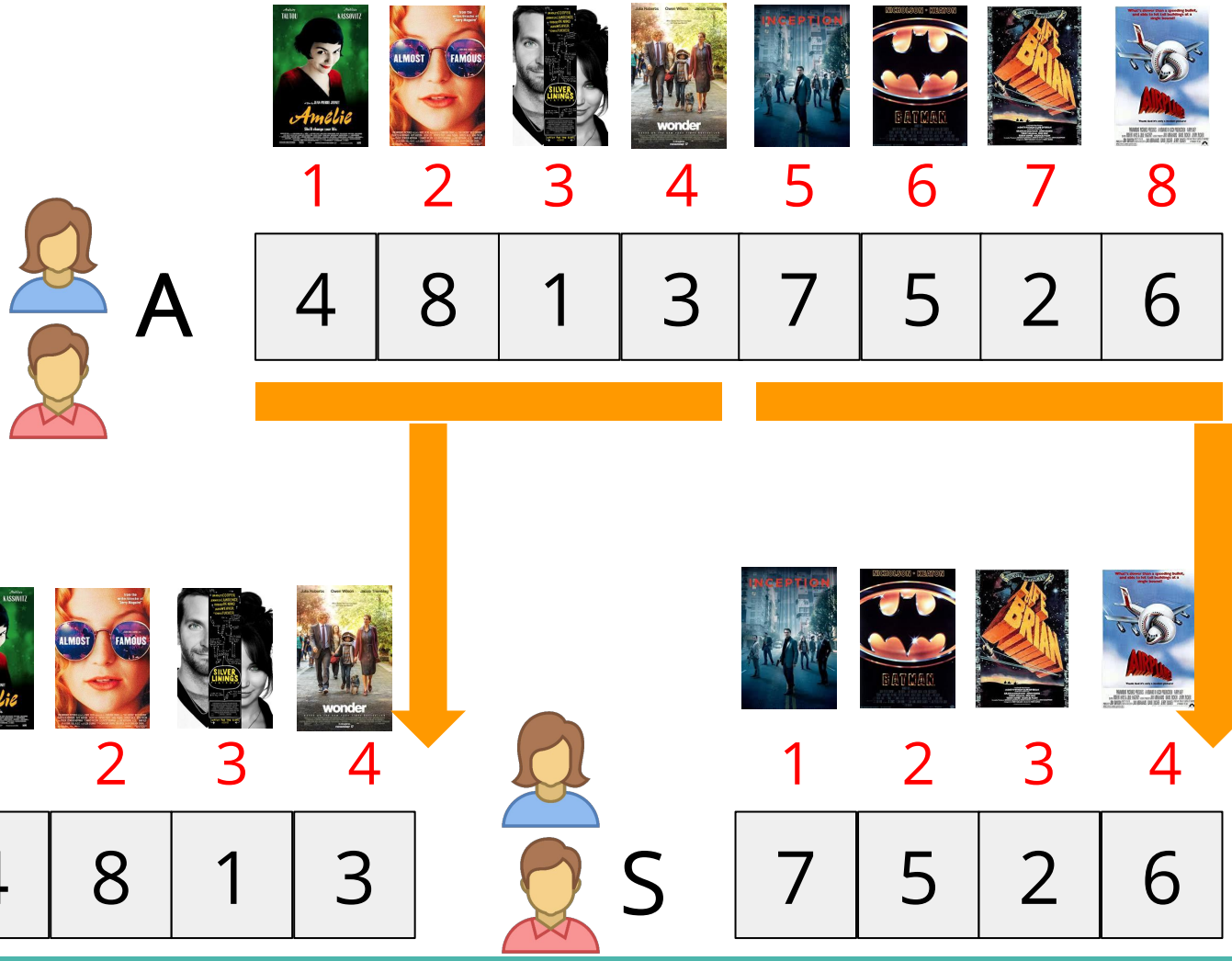
Can We Do Better? An Algorithm Based Approach

The **divide** stage is the same as it was before.

Can We Do Better? An Algorithm Based Approach

To understand how to compute res, let's come back to MergeSort.

1. **Divide** stage.



Can We Do Better? An Algorithm Based Approach

We can see that the divide stage still divides the problem into two equally-sized sub-problems, so:

a = 2, **b =2**.

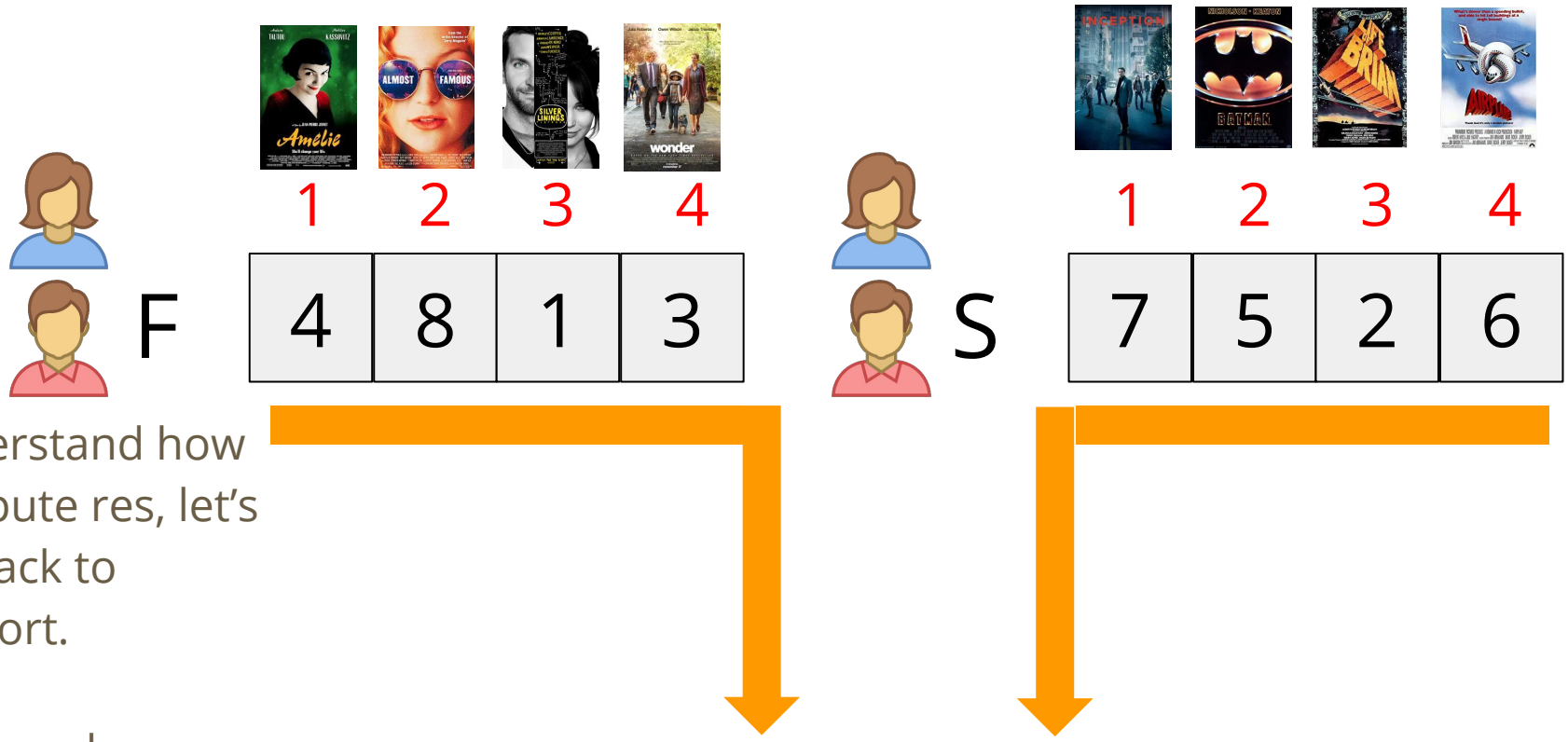
Can We Do Better? An Algorithm Based Approach

The **map** stage solves the sub-problems as it did before...

...the only difference is that now it returns:

- The sorted sub-problem array.
- The number of inversions in the sub-problem.

Can We Do Better? An Algorithm Based Approach



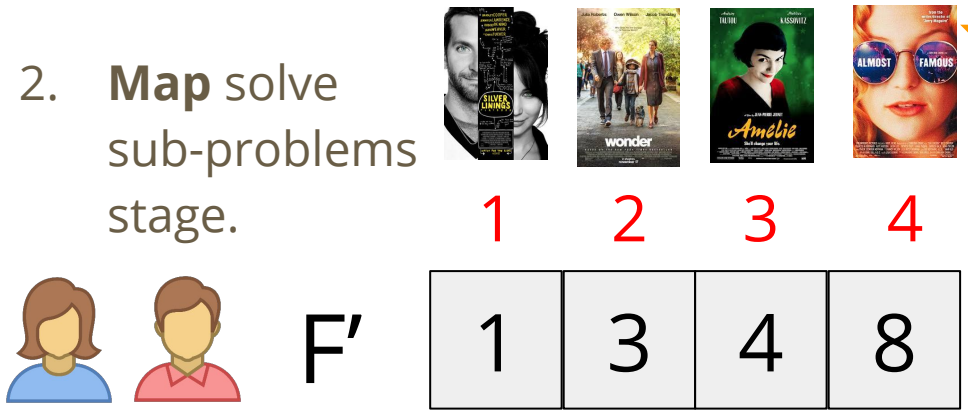
To understand how to compute res, let's come back to MergeSort.

2. **Map** solve sub-problems stage.

Can We Do Better? An Algorithm Based Approach

To understand how to compute res, let's come back to MergeSort.

2. **Map** solve sub-problems stage.



Can We Do Better? An Algorithm Based Approach

F

4	8	1	3

S

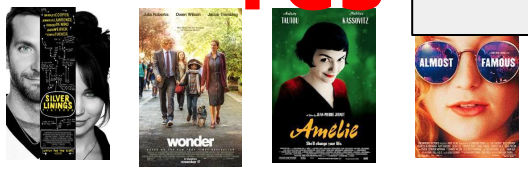
7	5	2	6

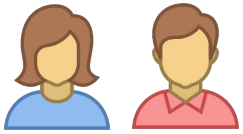
To understand how to compute res, let's come back to MergeSort.

res 0

res 4

2. **Map** solve sub-problems stage.





F'

1	3	4	8

Can We Do Better? An Algorithm Based Approach

F

4	8	1	3

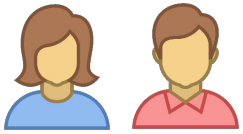
S

7	5	2	6

To understand how to compute res, let's come back to MergeSort.

res 0

2. **Map** solve sub-problems stage.



F'

1	3	4	8

Can We Do Better? An Algorithm Based Approach

F

4	8	1	3

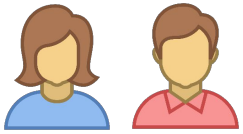
S

7	5	2	6

To understand how to compute res, let's come back to MergeSort.

res | 1 |
|---|

2. **Map** solve sub-problems stage.



F'

1	3	4	8

Can We Do Better? An Algorithm Based Approach

F

4	8	1	3

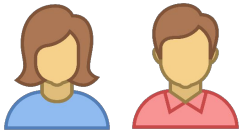
S

7	5	2	6

To understand how to compute res, let's come back to MergeSort.

res 2

2. **Map** solve sub-problems stage.



F'

1	3	4	8

Can We Do Better? An Algorithm Based Approach

F

4	8	1	3

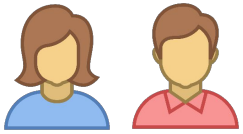
S

7	5	2	6

To understand how to compute res, let's come back to MergeSort.

res 3

2. **Map** solve sub-problems stage.



F'

1	3	4	8

Can We Do Better? An Algorithm Based Approach

F

4	8	1	3

S

7	5	2	6

To understand how to compute res, let's come back to MergeSort.

res 4

2. **Map** solve sub-problems stage.

F'

1	3	4	8

Can We Do Better? An Algorithm Based Approach

F

4	8	1	3

S

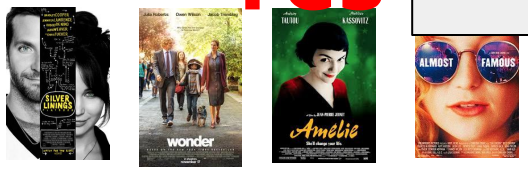
7	5	2	6

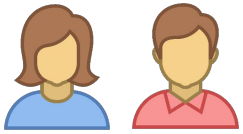
To understand how to compute res, let's come back to MergeSort.

res 0

res 4

2. **Map** solve sub-problems stage.

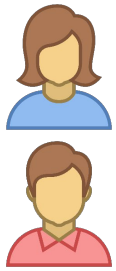




F'

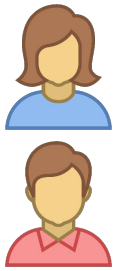
1	3	4	8

Can We Do Better? An Algorithm Based Approach



F

4	8	1	3



S

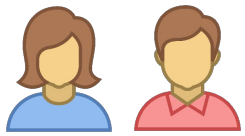
7	5	2	6

To understand how to compute res, let's come back to MergeSort.

res	0
res	**4**

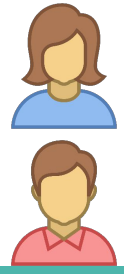
res	0
res	**4**

2. **Map** solve sub-problems stage.



F'

1	3	4	8



S'

2	5	6	7

Can We Do Better? An Algorithm Based Approach

1 2 3 4

F 4 8 1 3

S 7 5 2 6

1 2 3 4

To understand how to compute res, let's come back to MergeSort.

res 0

res 4

res 0

2. **Map** solve sub-problems stage.

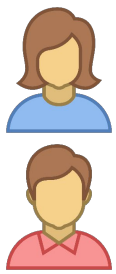
1 2 3 4

F' 1 3 4 8

S' 2 5 6 7

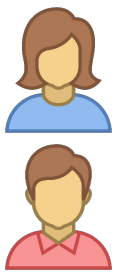
1 2 3 4

Can We Do Better? An Algorithm Based Approach



1 2 3 4

F | 4 | 8 | 1 | 3 |



1 2 3 4

S | 7 | 5 | 2 | 6 |

To understand how to compute res, let's come back to MergeSort.

res 0

res 4

res 1

2. **Map** solve sub-problems stage.

1 2 3 4

F′ | 1 | 3 | 4 | 8 |

S′ | 2 | 5 | 6 | 7 |

Can We Do Better? An Algorithm Based Approach

F

4	8	1	3

S

7	5	2	6

To understand how to compute res, let's come back to MergeSort.

res 0

res 2

res 4

2. **Map** solve sub-problems stage.

F'

1	3	4	8

S'

2	5	6	7

Can We Do Better? An Algorithm Based Approach

To understand how to compute res, let's come back to MergeSort.

res 0

res 4

res 3

2. **Map** solve sub-problems stage.

Can We Do Better? An Algorithm Based Approach

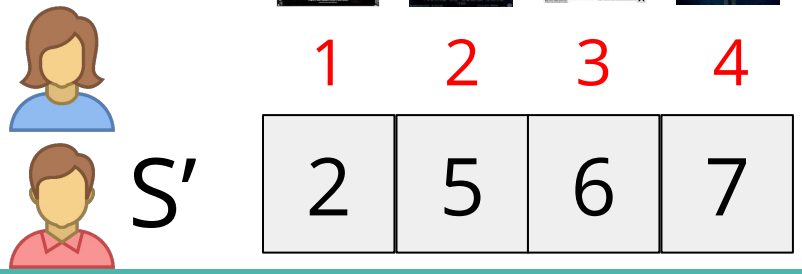
To understand how to compute res, let's come back to MergeSort.

res 0

res 4

res 4

2. **Map** solve sub-problems stage.



Can We Do Better? An Algorithm Based Approach

1 2 3 4

F

4	8	1	3

S

7	5	2	6

To understand how to compute res, let's come back to MergeSort.

res 0

res 4

res 0

res 4

2. **Map** solve sub-problems stage.

1 2 3 4

F′

1	3	4	8

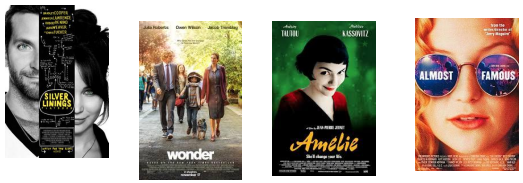
S′

2	5	6	7

Can We Do Better? An Algorithm Based Approach

By solving the 2 sub-problems in the **map** stage we have already found 8 inversions.

Can We Do Better? An Algorithm Based Approach



res	rF′		1	2	3	4	rS′		1	2	3	4
0	4	F′	1	3	4	8	4	S′	2	5	6	7

Can We Do Better? An Algorithm Based Approach


res

4

rF’

4

	1	2	3	4
F’	1	3	4	8

rS’

4

	1	2	3	4
S’	2	5	6	7

Can We Do Better? An Algorithm Based Approach

res

8

rF’

4

F’

1	3	4	8

1 2 3 4

rS’

4

S’

2	5	6	7


1 2 3 4

Can We Do Better? An Algorithm Based Approach

By solving the 2 sub-problems in the **map** stage we have got already found 8 inversions.

...these 8 inversions had one property:
The two movies (MovieA, MovieB) where both within the same sub-problem.

Can We Do Better? An Algorithm Based Approach

But what happens with inversions (MovieA, MovieB) where each movie fell in a different sub-problem?

Can We Do Better? An Algorithm Based Approach

But what happens with inversions (MovieA, MovieB) where each movie fell in a different sub-problem?

The **reduce** stage will find them!

Can We Do Better? An Algorithm Based Approach

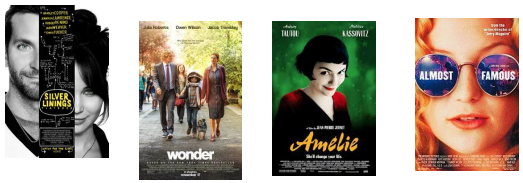
Obviously, the total amount of inversions is:

- The ones found by the **map** stage on solving each sub-problem

+

- The new ones found now by the **reduce** stage.

Can We Do Better? An Algorithm Based Approach



res

8

rF'

4

F'

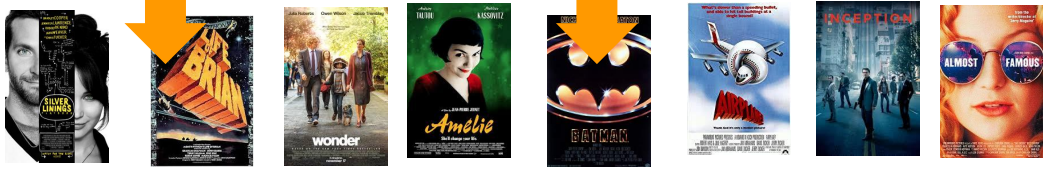
1	3	4	8

rS'

4

S'

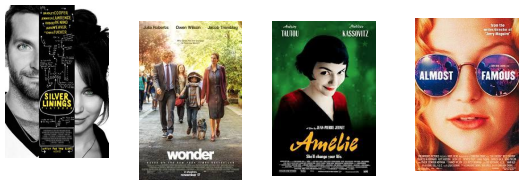
2	5	6	7



A'

1	2	3	4	5	6	7	8

Can We Do Better? An Algorithm Based Approach



res

9

rF’

4

F’

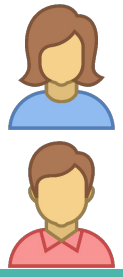
1	3	4	8

rS’

4

S’

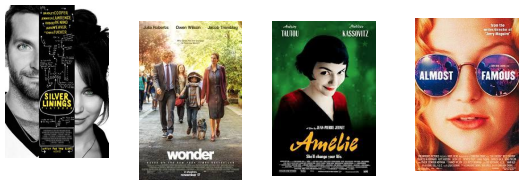
2	5	6	7



A’

1	2	3	4	5	6	7	8

Can We Do Better? An Algorithm Based Approach



res

10

rF’

4

F’

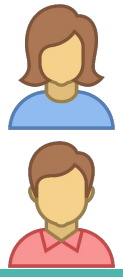
1	3	4	8

rS’

4

S’

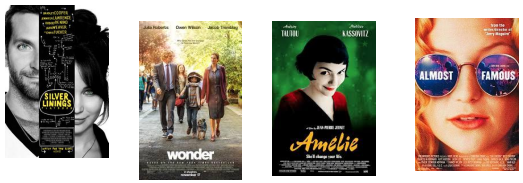
2	5	6	7



A’

1	2	3	4	5	6	7	8

Can We Do Better? An Algorithm Based Approach



res

11

rF′

4

F′

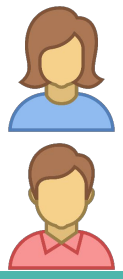
1	3	4	8

rS′

4

S′

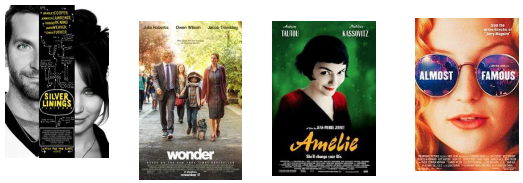
2	5	6	7



A′

1	2	3	4	5	6	7	8

Can We Do Better? An Algorithm Based Approach



res

12

rF’

4

F’

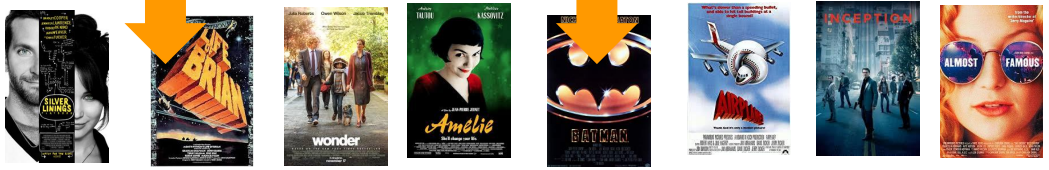
1	2	3	4
1	3	4	8

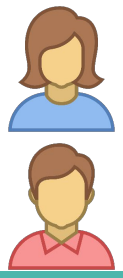
rS’

4

S’

1	2	3	4
2	5	6	7

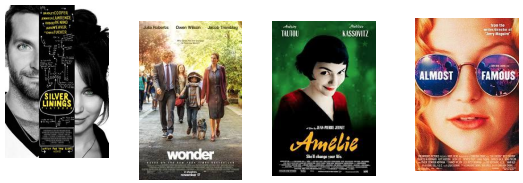




A’

1	2	3	4	5	6	7	8

Can We Do Better? An Algorithm Based Approach



res

13

rF′

4

F′

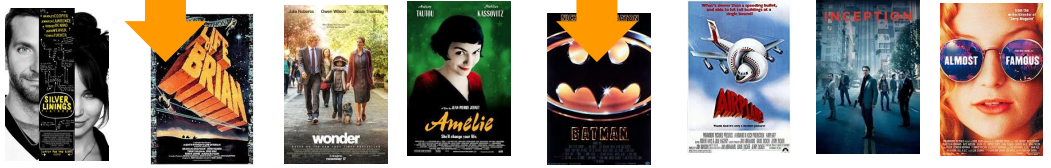
1	3	4	8

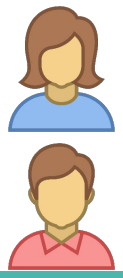
rS′

4

S′

2	5	6	7

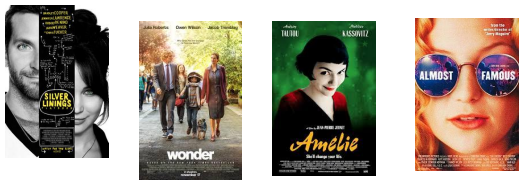




A′

1	2	3	4	5	6	7	8

Can We Do Better? An Algorithm Based Approach



res

14

rF’

4

F’

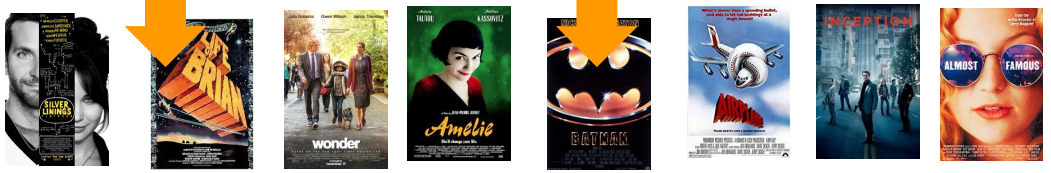
1	2	3	4
1	3	4	8

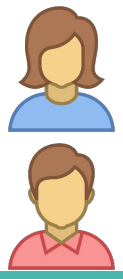
rS’

4

S’

1	2	5	6	7
2	5	6	7	

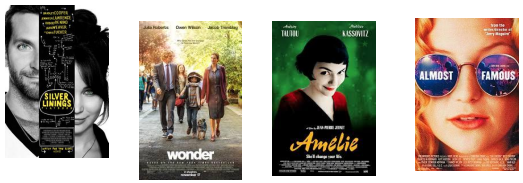




A’

1	2	3	4	5	6	7	8

Can We Do Better? An Algorithm Based Approach



res

8

rF’

4

F’

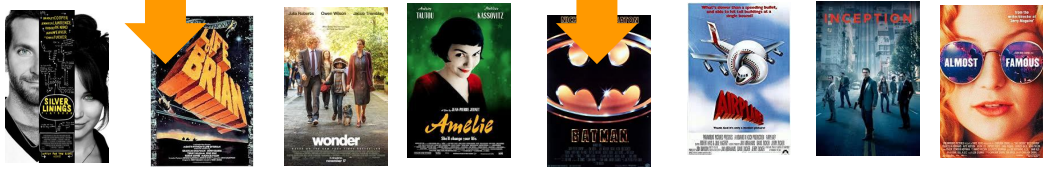
1	2	3	4
1	3	4	8

rS’

4

S’

1	2	3	4
2	5	6	7



A’

1	2	3	4	5	6	7	8

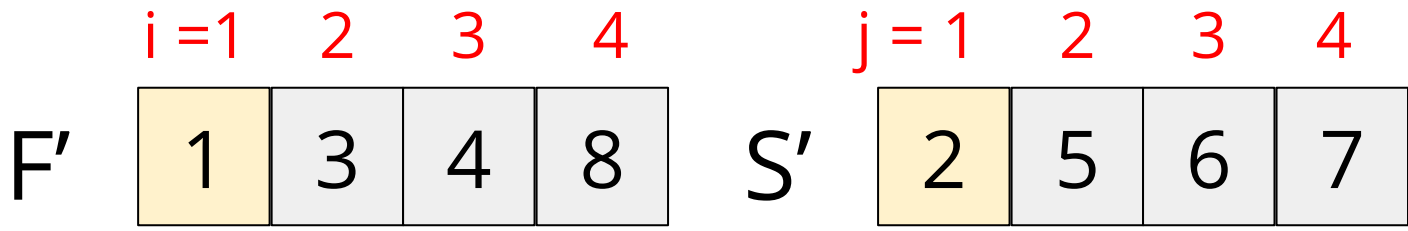
res **14**

Can We Do Better? An Algorithm Based Approach

Let's see the modified **reduce** stage
in charge of finding these
inter sub-problem inversions.

Can We Do Better? An Algorithm Based Approach

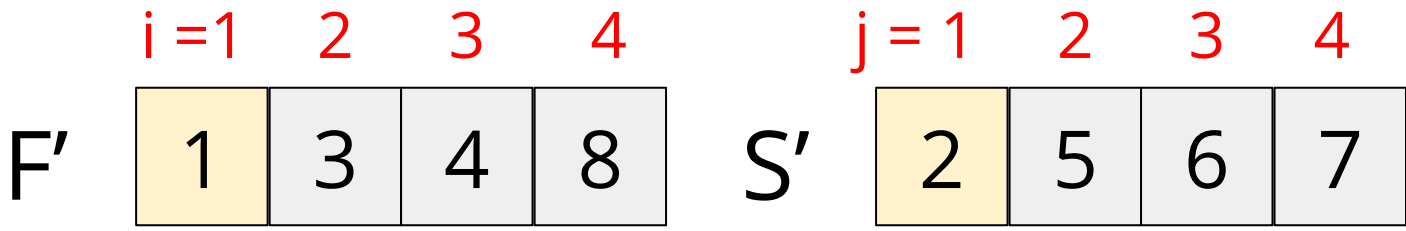
We start with:
- An empty array A'.
- An index **i** at the beginning of F'.
- An index **j** at the beginning of S'.
- res = 0



A'

Can We Do Better? An Algorithm Based Approach

- We compare the positions i and j. As both F' and S' are already sorted, whoever is the smallest of them has also to be the smallest element of F' and S'.

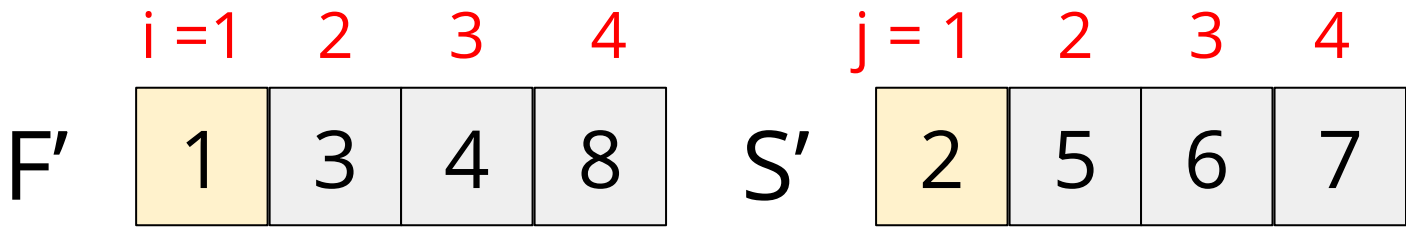


res 0

A'

Can We Do Better? An Algorithm Based Approach

- We compare the positions i and j. As both F' and S' are already sorted, whoever is the smallest of them has also to be the smallest element of F' and S'.

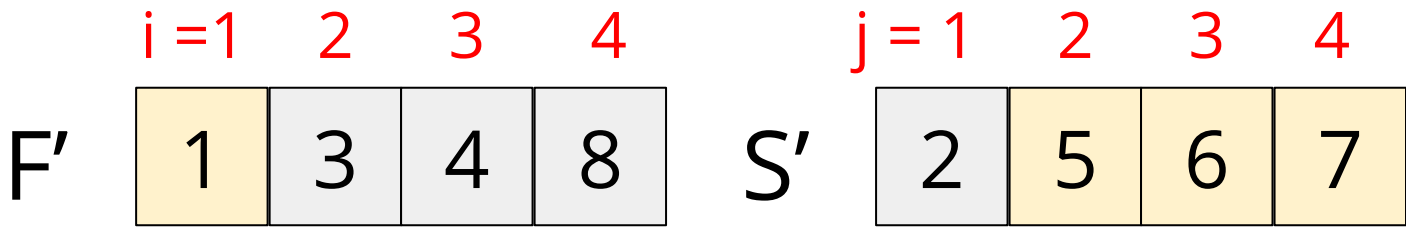


- F[i] < S'[j].

A'

Can We Do Better? An Algorithm Based Approach

- We compare the positions i and j. As both F' and S' are already sorted, whoever is the smallest of them has also to be the smallest element of F' and S'.

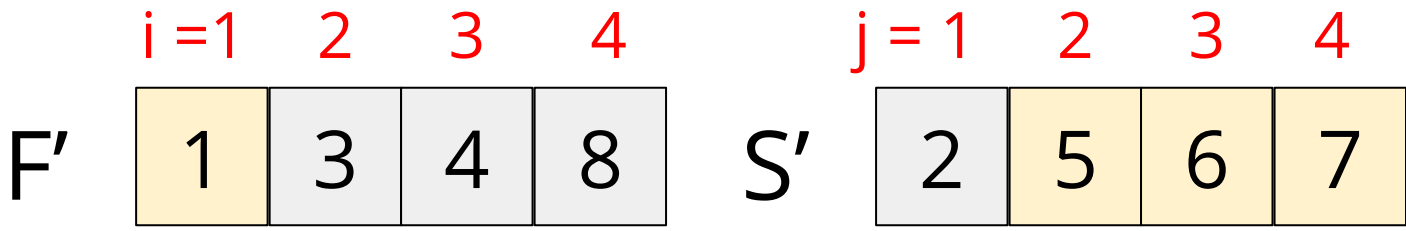


- F[i] < S'[j].
- But, as F' is in order, this also means F'[i] will be smaller than all the other values of F'.

A'

Can We Do Better? An Algorithm Based Approach

- We compare the positions i and j.
As both F' and S' are already sorted, whoever is the smallest of them has also to be the smallest element of F' and S'.

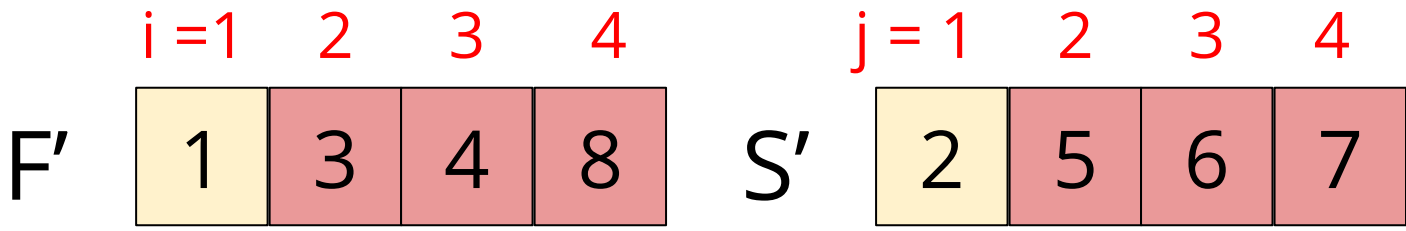


- F[i] < S'[j].
- But, as F' is in order, this also means F'[i] will be smaller than all the other values of F'.
- But, as S' is in order, this also means F'[i] will be smaller than all the other values of S'.

A'

Can We Do Better? An Algorithm Based Approach

- We compare the positions i and j. As both F' and S' are already sorted, whoever is the smallest of them has also to be the smallest element of F' and S'.



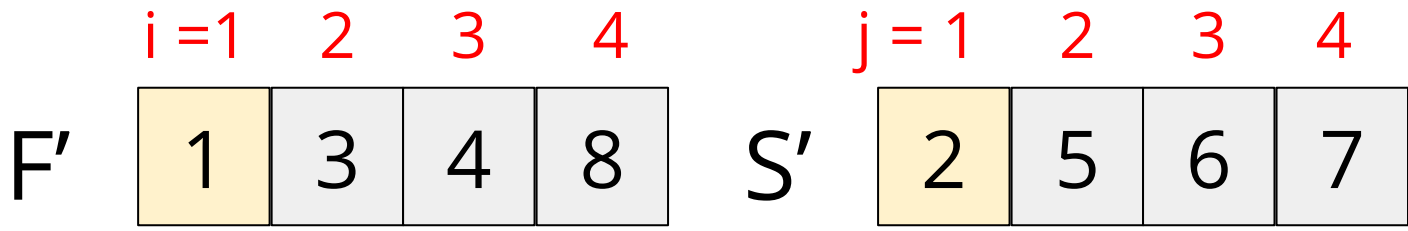
- F[i] < S'[j].
- But, as F' is in order, this also means F'[i] will be smaller than all the other values of F'.
- But, as S' is in order, this also means F[i] will be smaller than all the other values of S'.
- Interestingly, to reach to this conclusion we don't need to compare F'[i] with all the remaining values of F' nor S', just with S'[j].

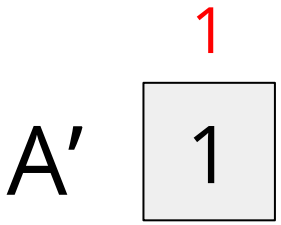
res 0

A'

Can We Do Better? An Algorithm Based Approach

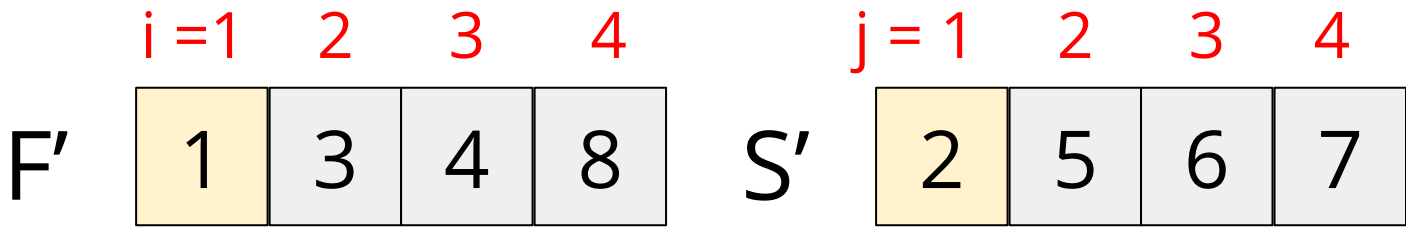
- If F'[i] < S'[j] we do not modify res.



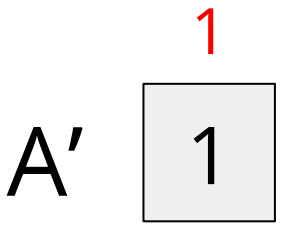


Can We Do Better? An Algorithm Based Approach

- We add the smallest of F'[i] and S'[j] to A'.

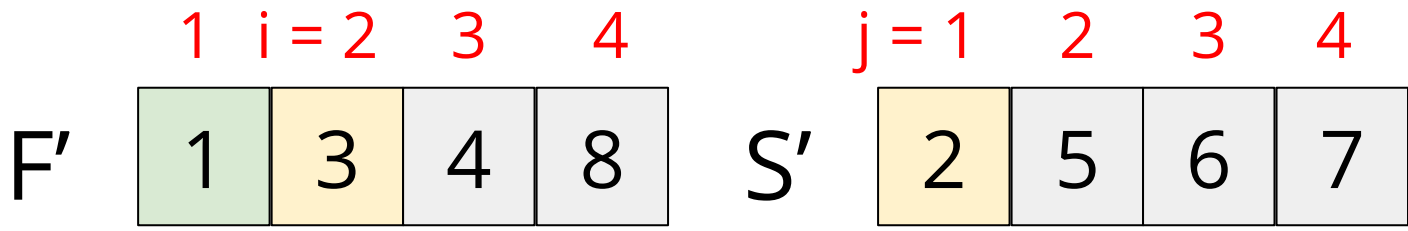


res 0

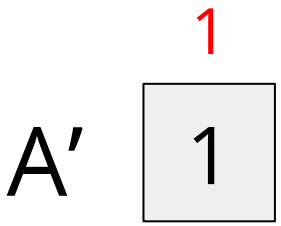


Can We Do Better? An Algorithm Based Approach

- We increase i or j accordingly.

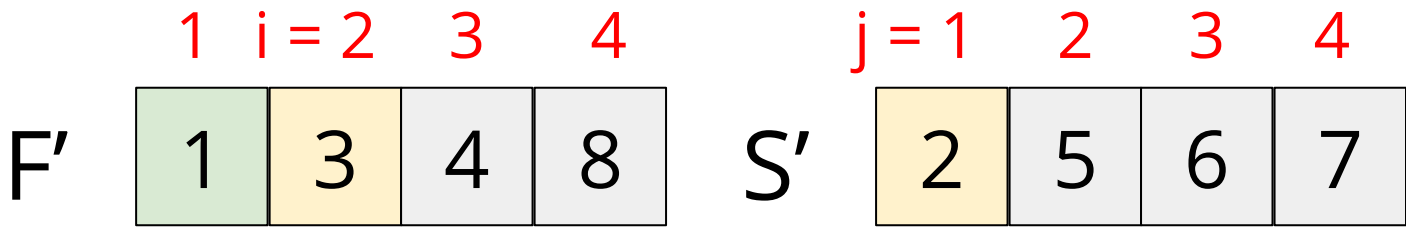


res 0

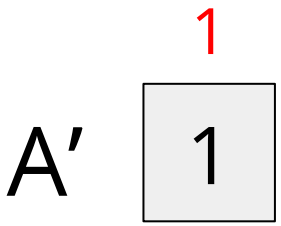


Can We Do Better? An Algorithm Based Approach

- And we repeat these steps until all elements are added to A'.

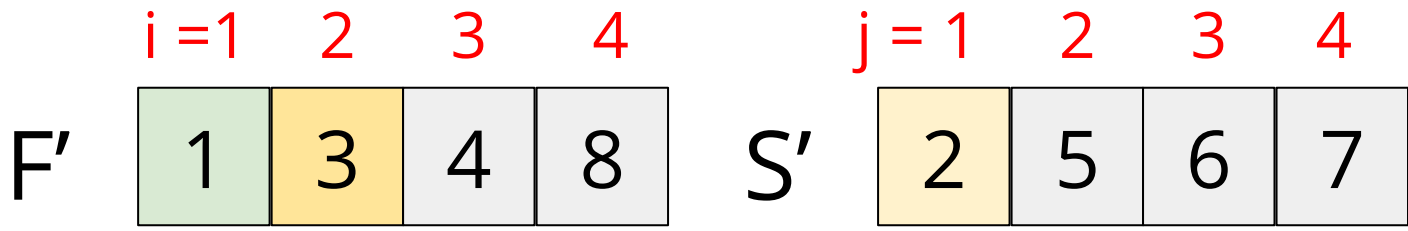


res 0

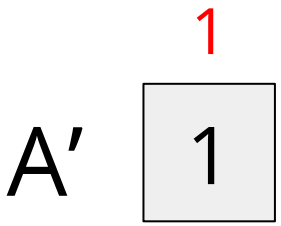


Can We Do Better? An Algorithm Based Approach

- But, if F'[i] > S'[j] then we have found an inversion.

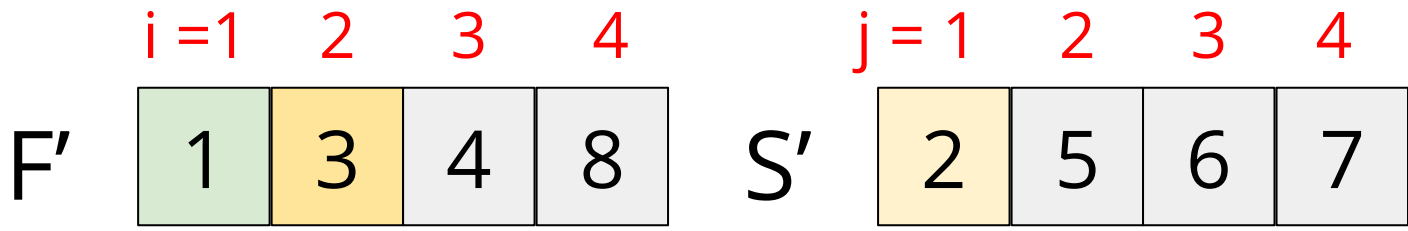


res 0



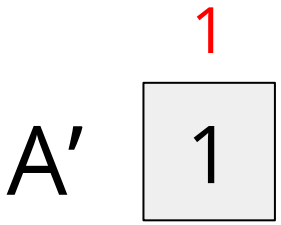
Can We Do Better? An Algorithm Based Approach

- But, if F'[i] > S'[j] then we have found an inversion.



And we need to update res.

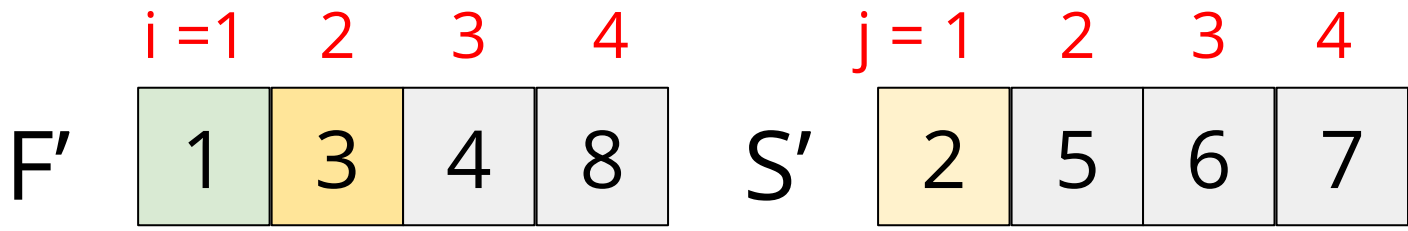
res 0 + 1



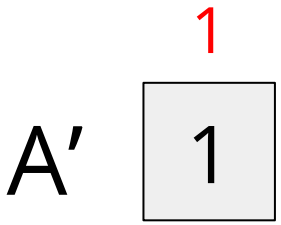
Can We Do Better? An Algorithm Based Approach

- But the same reasoning applies here:

If F'[i] > S'[j]
and F' is sorted...



res 0 + ?

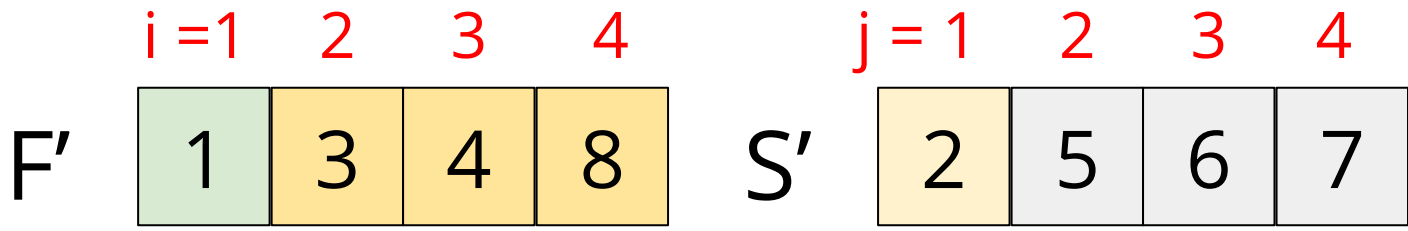


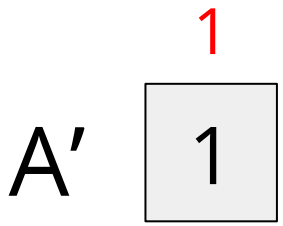
Can We Do Better? An Algorithm Based Approach

- But the same reasoning applies here:

If F'[i] > S'[j]
and F' is sorted...

...then not only F[i] but all the remaining elements of F' will also be > S'[j]

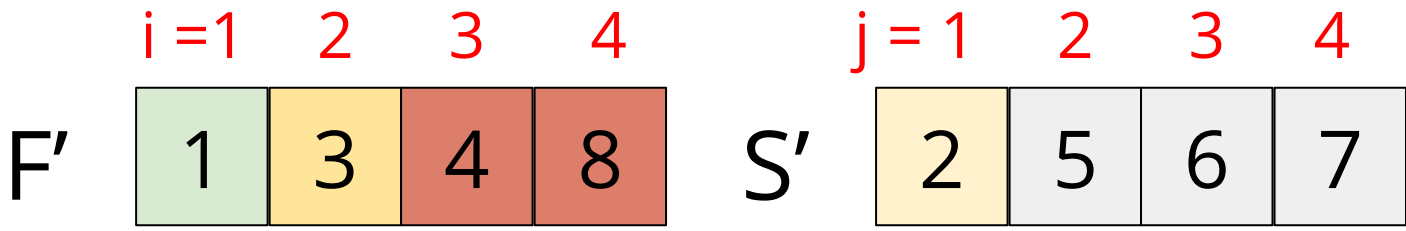




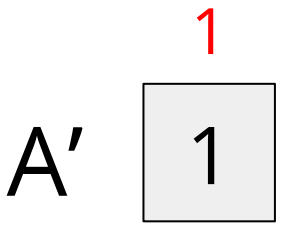
res 0 + 3

Can We Do Better? An Algorithm Based Approach

- Interestingly, to reach to this conclusion we don't need to compare all the remaining values of F' with S'[j]', just F'[i] and S'[j].

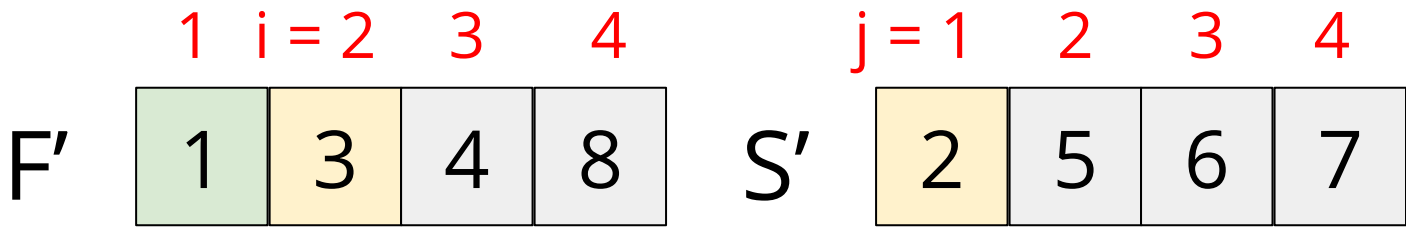


res 0 + 3

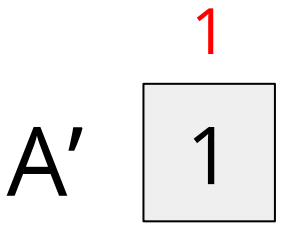


Can We Do Better? An Algorithm Based Approach

- And now yes, we repeat these steps until all elements are added to A'.

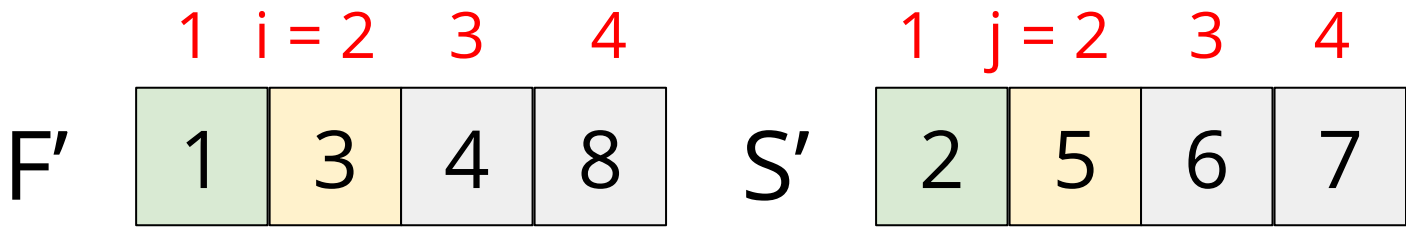


res 3

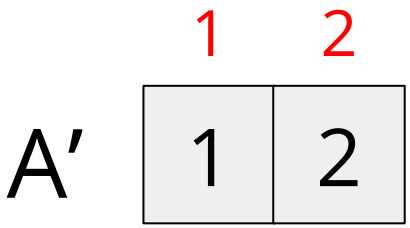


Can We Do Better? An Algorithm Based Approach

- And we repeat these steps until all elements are added to A'.

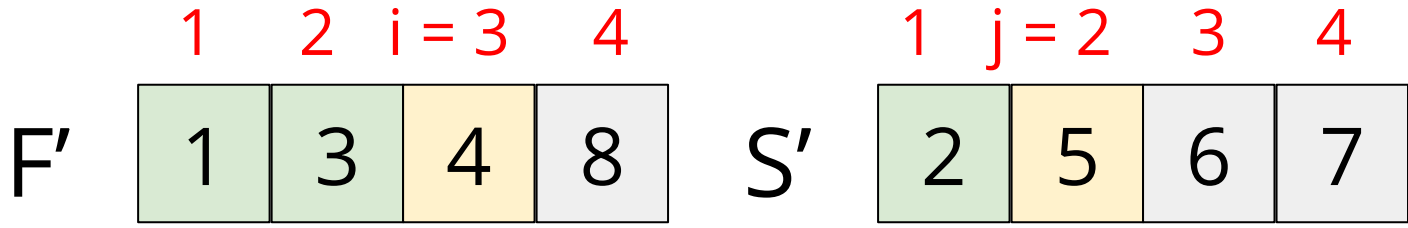


res 3

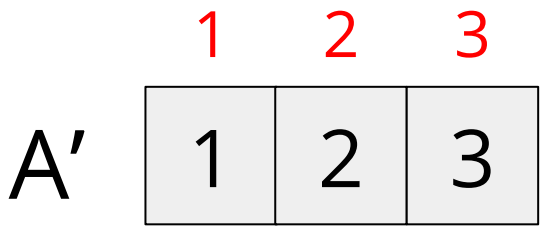


Can We Do Better? An Algorithm Based Approach

- And we repeat these steps until all elements are added to A'.

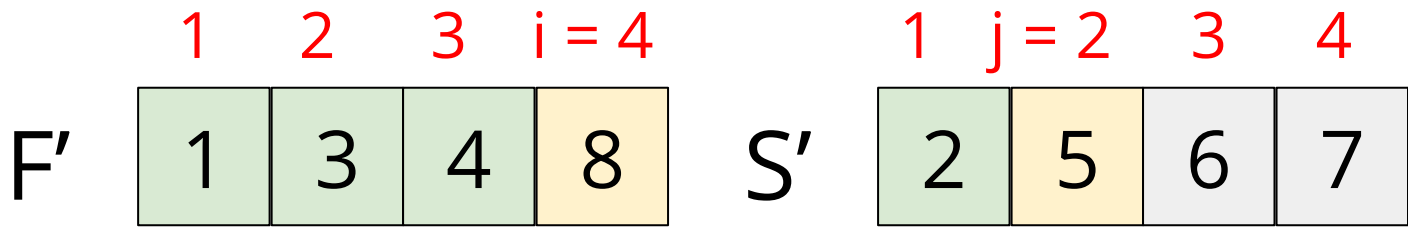


res 3

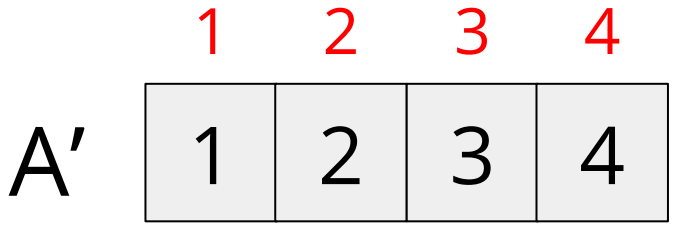


Can We Do Better? An Algorithm Based Approach

- And we repeat these steps until all elements are added to A'.

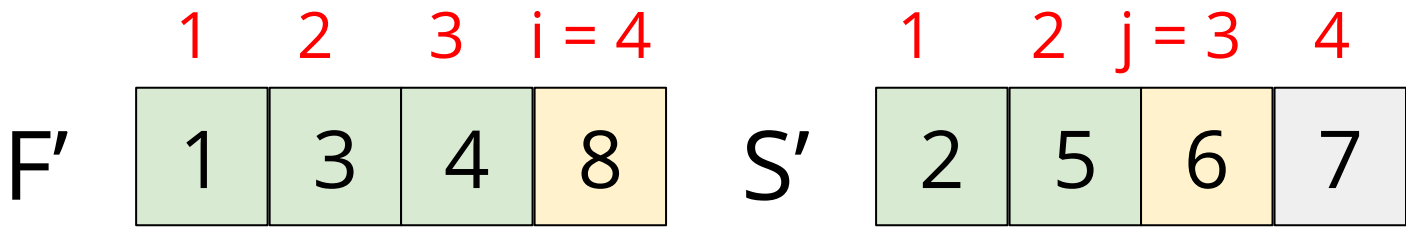


res **4**

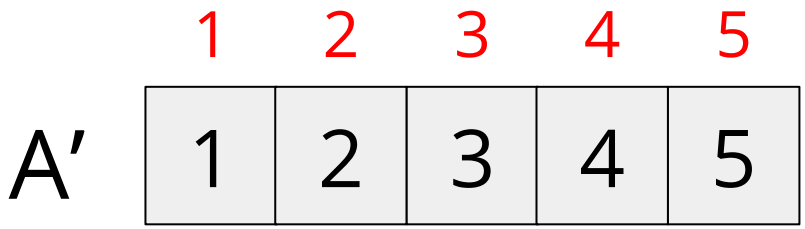


Can We Do Better? An Algorithm Based Approach

- And we repeat these steps until all elements are added to A'.

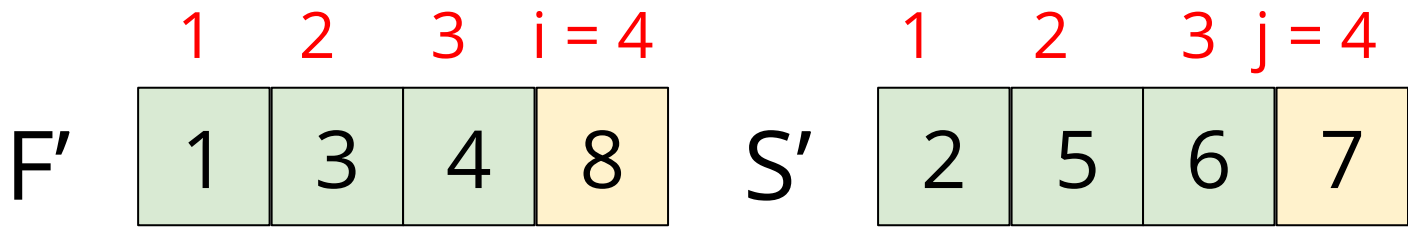


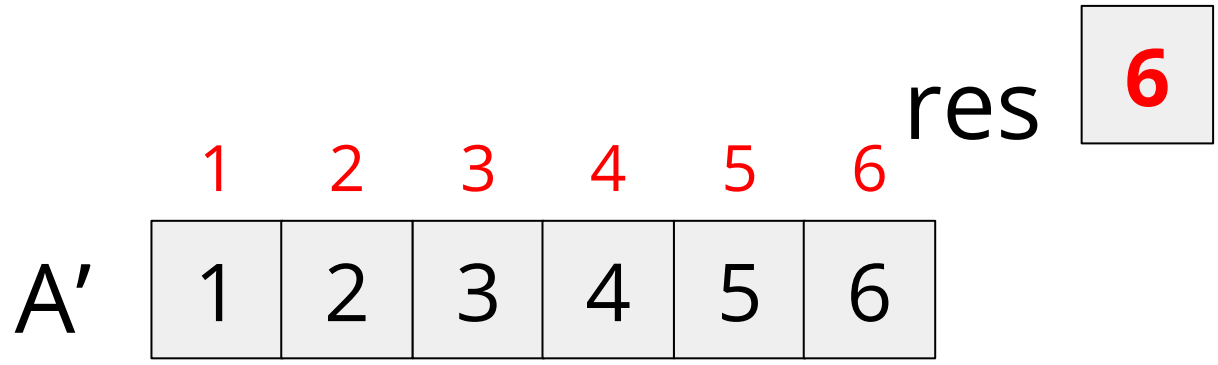
res **5**



Can We Do Better? An Algorithm Based Approach

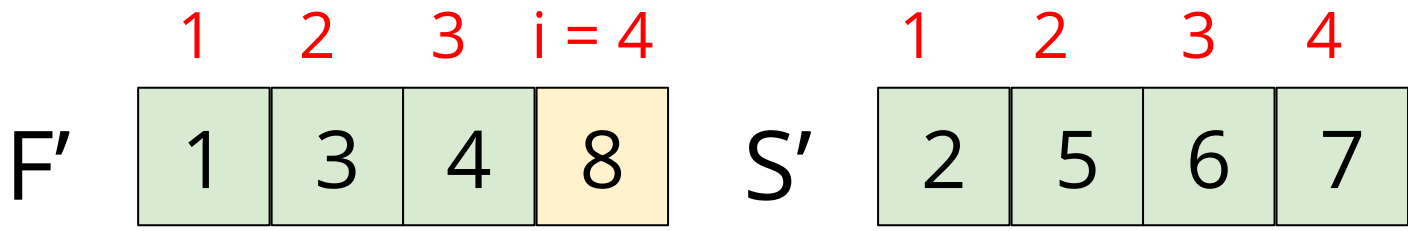
- And we repeat these steps until all elements are added to A'.



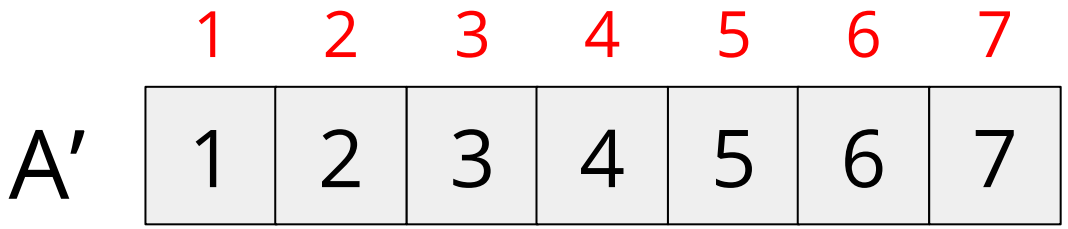


Can We Do Better? An Algorithm Based Approach

- And we repeat these steps until all elements are added to A'.

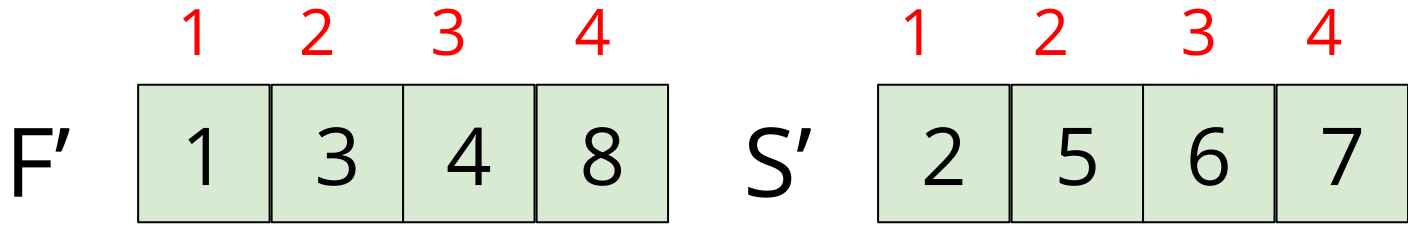


res 6

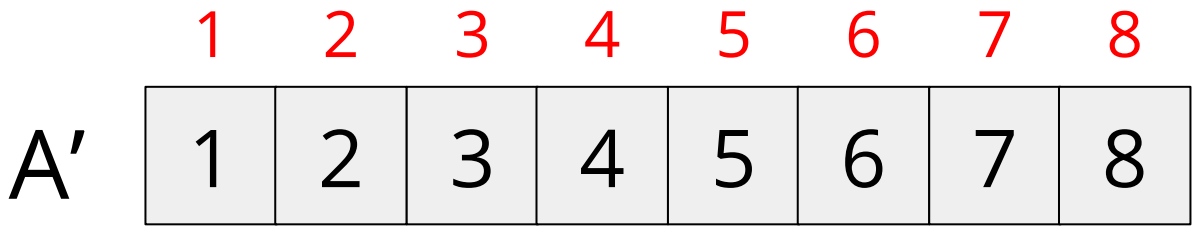


Can We Do Better? An Algorithm Based Approach

- And we repeat these steps until all elements are added to A'.



res 6



Can We Do Better? An Algorithm Based Approach

We can see that the reduce stage still requires $O(n^1)$ operations, so:

d = 1.

Can We Do Better? An Algorithm Based Approach

Thus, given that we still have **a = 2**, **b = 2**, **d = 1**, we are still in the case (a == b^d) for which

$T(n) = (n^d) * \log n$

Can We Do Better? An Algorithm Based Approach

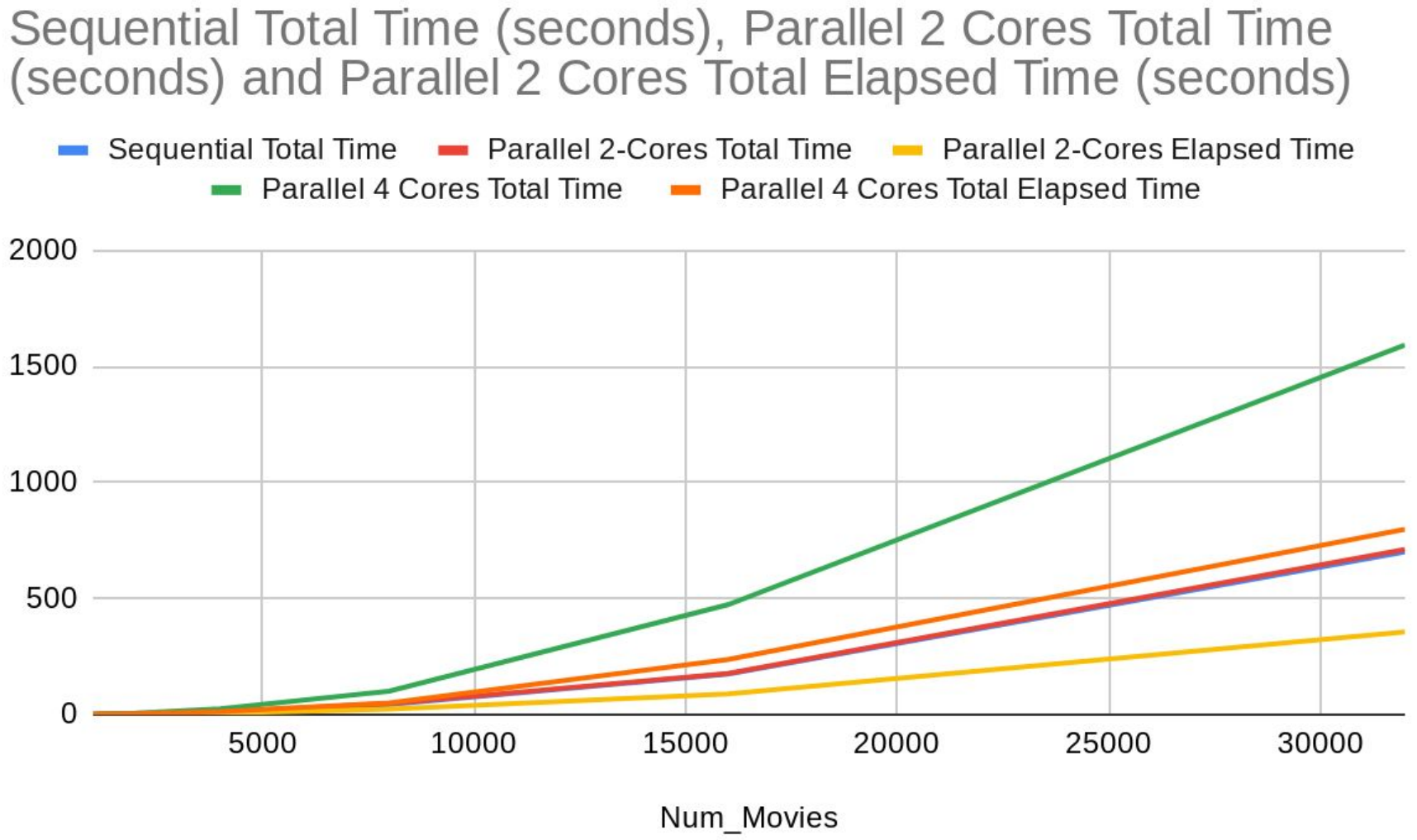
Thus, given that we still have **a = 2**, **b = 2**, **d = 1**,
we are still in the case ($a == b^d$) for which
$T(n) = (n^d) * \log n$

And this is why our
new list inversions algorithm
has time complexity $O(n * \log n)$

Can We Do Better? An Algorithm Based Approach

Now, let's look back to our time analysis again...

Can We Do Better? An Algorithm Based Approach

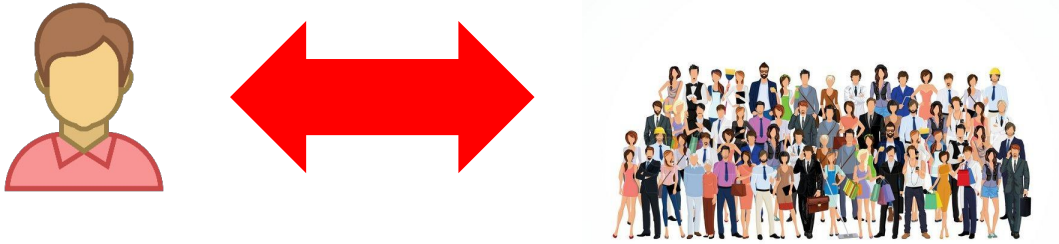


Can We Do Better? An Algorithm Based Approach

Let's look back to our time analysis again...

...but now let's look at it by focusing on the new algorithm count inversions O(n log n).

Can We Do Better? An Algorithm Based Approach



- The code examples associated to this lecture provide us with a benchmark generator & solver for running the problem of computing list inversions of person P vs an entire population.

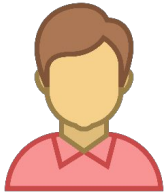
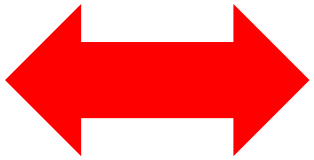
- We are going to use this tool now to test the scalability of:
  - Compute List Inversions of P vs the population using:
    - **The algorithm count_inversions of O(n log n).**
    - **A sequential-based approach.**
    - A fixed population size k = 12.
    - A number of movies ranging from 1,000 to 32,000.

Can We Do Better? An Algorithm Based Approach

- The results are presented below:

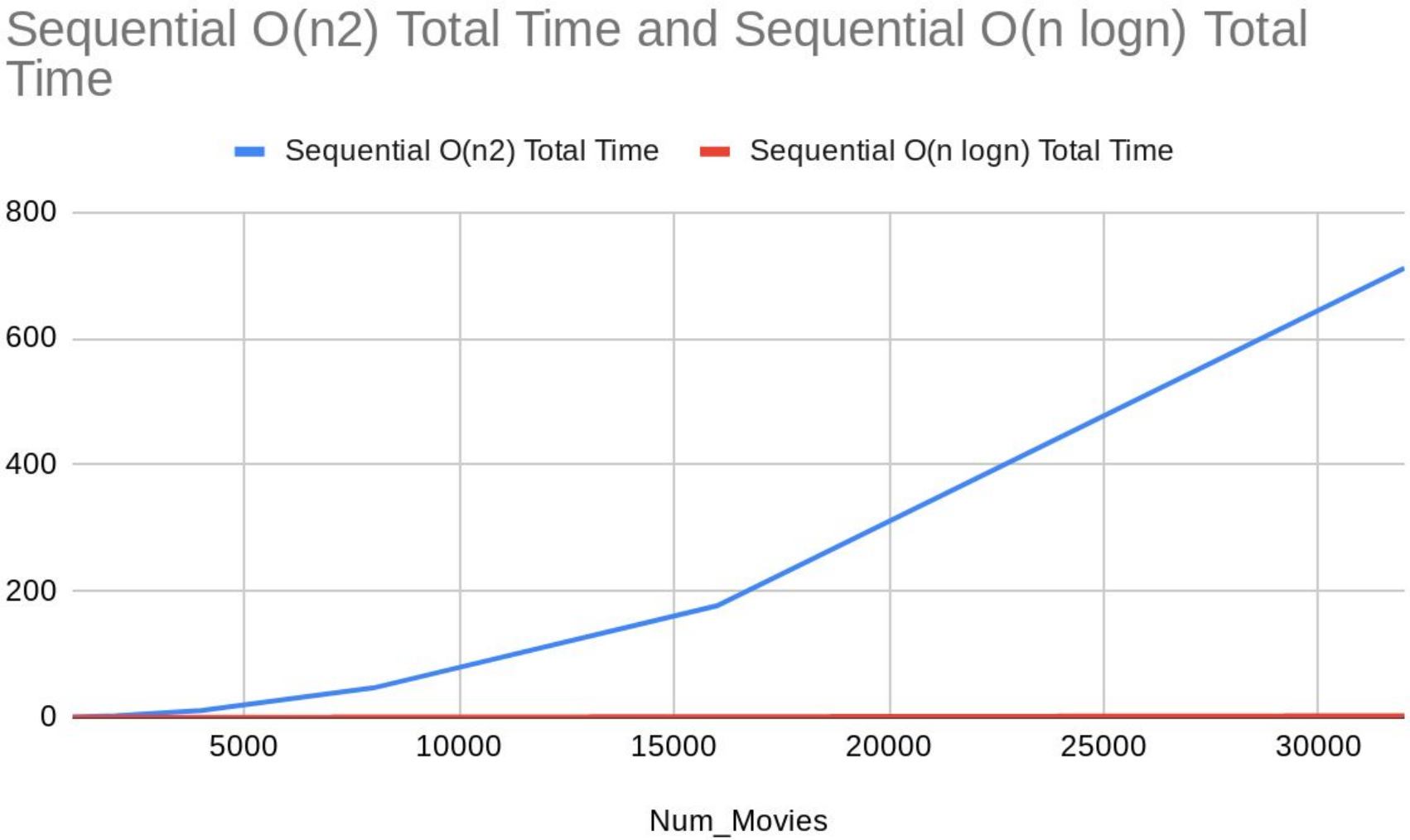
Num_Movies	1000	2000	4000	8000	16000	32000
Sequential O(n logn) Total Time	0.09	0.16	0.30	0.62	1.28	2.77

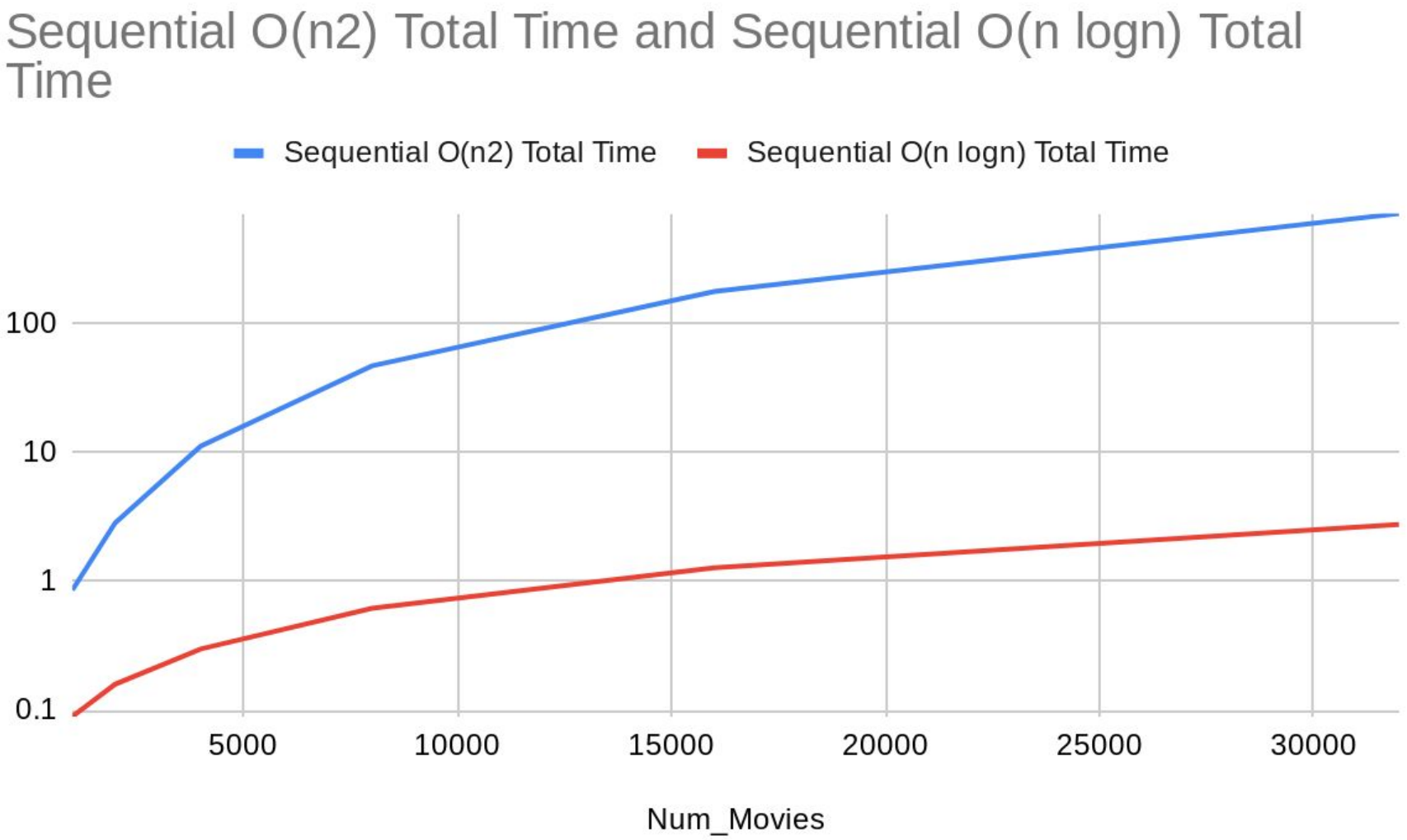
Concurrency: Infrastructure Based Approach


- And when we compare sequential $O(n^2)$ vs sequential O(nlogn):

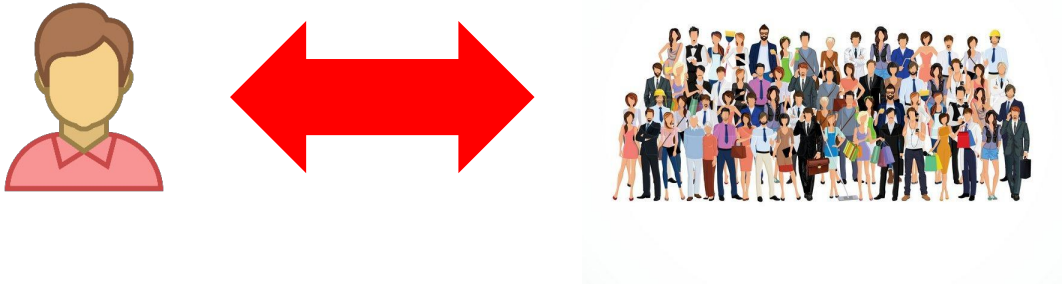
Num_Movies	1000	2000	4000	8000	16000	32000
Sequential $O(n^2)$ Total Time	0.7	2.75	10.93	44.03	174.04	700.9
Sequential O(n logn) Total Time	0.09	0.16	0.30	0.62	1.28	2.77


Sequential O(n2) Total Time and Sequential O(n logn) Total Time
Sequential O(n2) Total Time
Sequential O(n logn) Total Time
800
600
400
200
0
5000
10000
15000
20000
25000
30000
Num_Movies



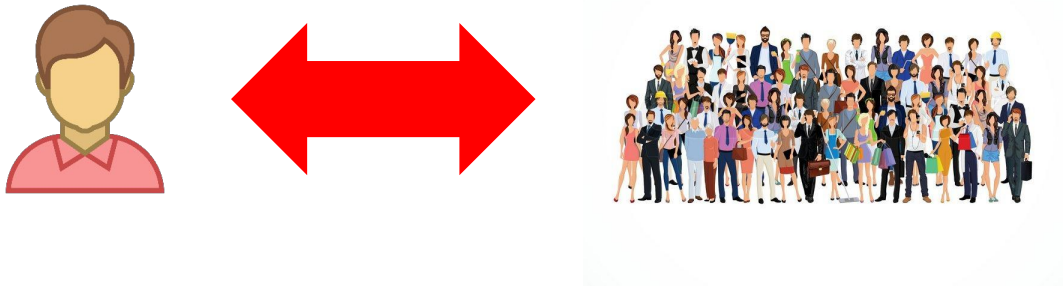
These are the results in logaritmic scale, as it is the only way to see the red line :)

Concurrency: Infrastructure Based Approach



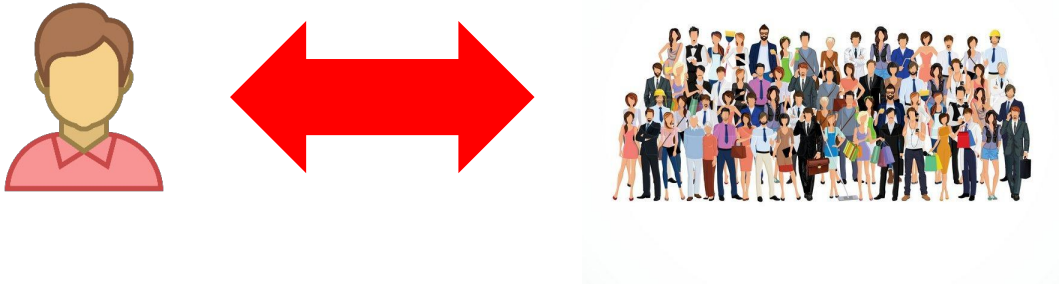
- As we can see, we have reduced the time dramatically! (see blue and red lines).

Concurrency: Infrastructure Based Approach



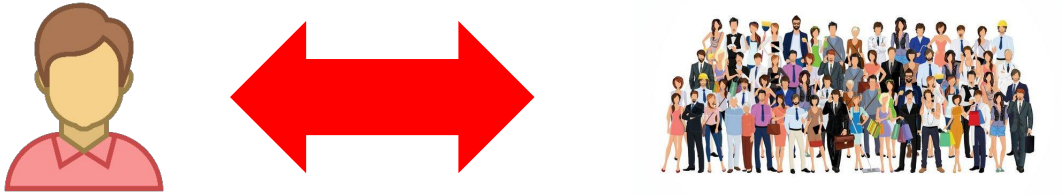
- As we can see, we have reduced the time dramatically! (see blue and red lines).
- In the case of n = 32,000 we are passing from 700.90s with the $O(n^2)$ algorithm to 2.77s with the O(n log n) algorithm.

Concurrency: Infrastructure Based Approach



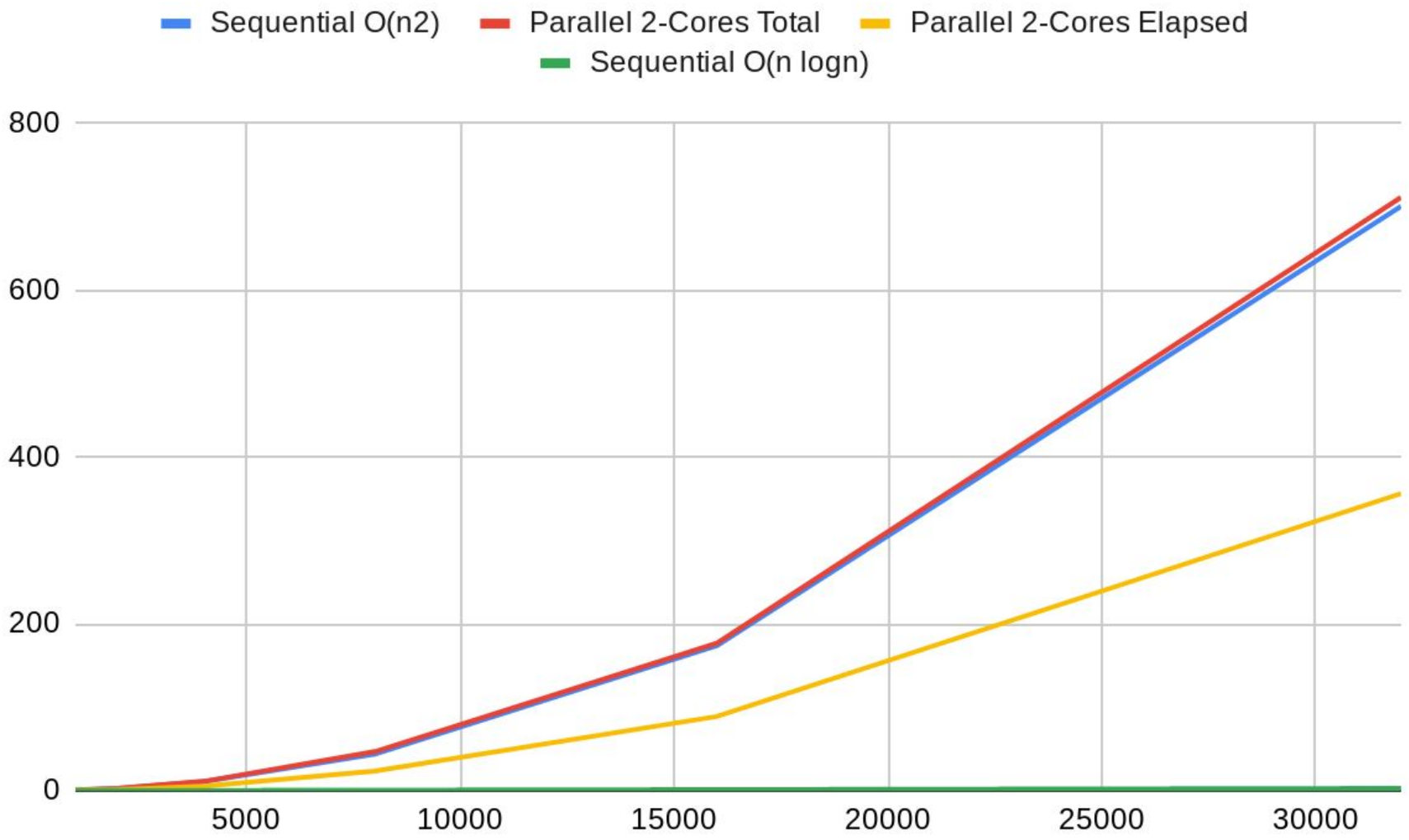
- As we can see, we have reduced the time dramatically! (see blue and red lines).
- In the case of n = 32,000 we are passing from 700.90s with the $O(n^2)$ algorithm to 2.77s with the O(n log n) algorithm.
- This is 253 times faster (2-3 orders of magnitude improvement).

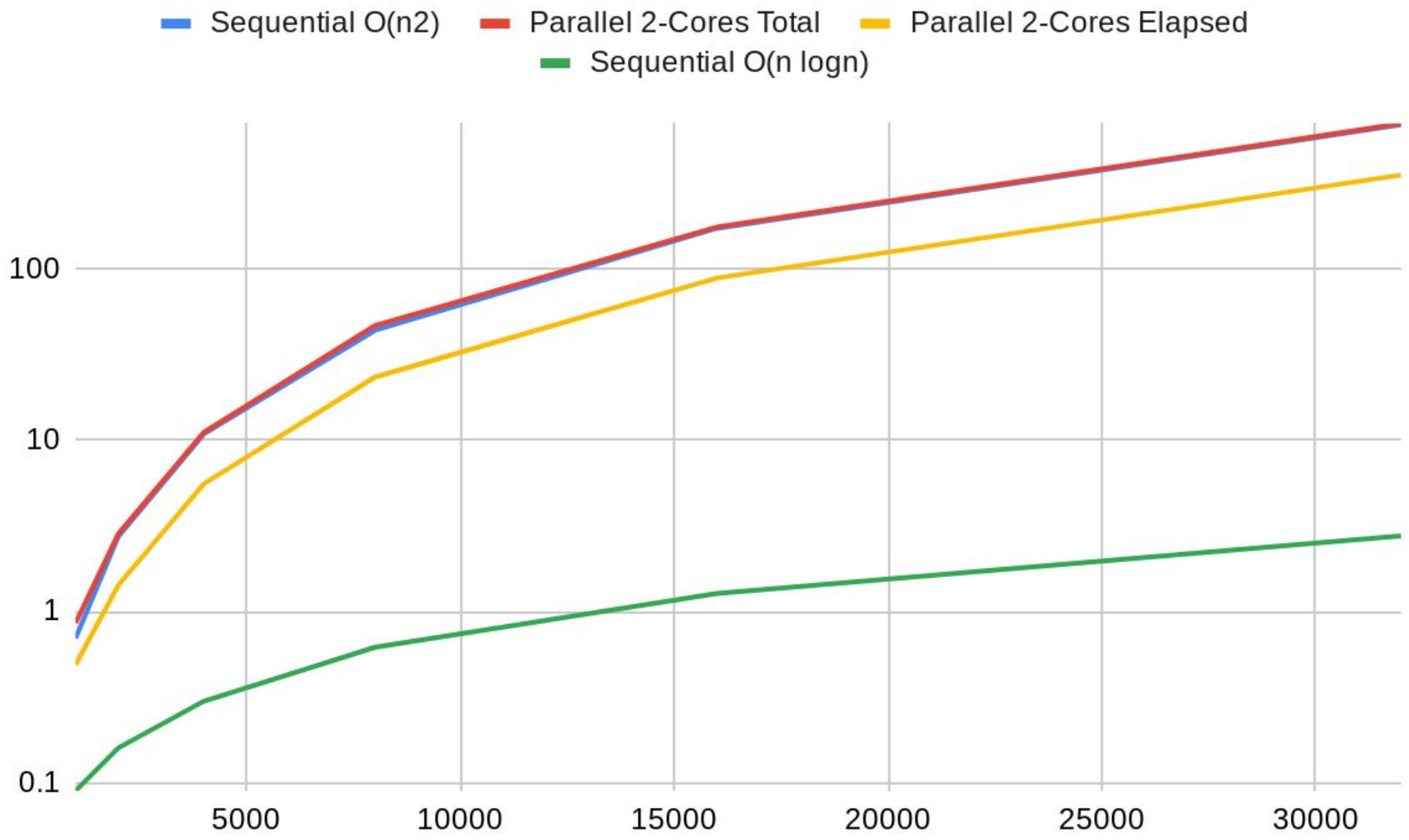
Concurrency: Infrastructure Based Approach



- The results are even more impressive when we compare sequential O(nlogn) vs parallel 2-cores $O(n^2)$:

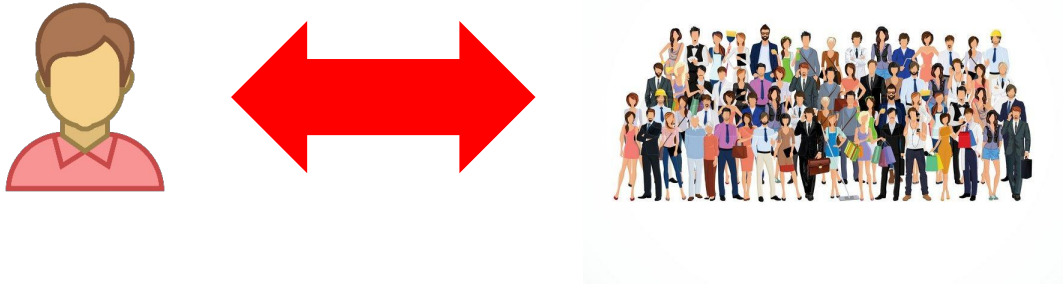
Num_Movies	1000	2000	4000	8000	16000	32000
Sequential $O(n^2)$ Total Time	0.7	2.75	10.93	44.03	174.04	700.9
Parallel 2-Cores Total Time	0.86	2.85	11.19	46.91	177.02	711.43
Parallel 2-Cores Elapsed Time	0.49	1.43	5.59	23.48	89.1	356.32
Sequential O(n logn) Total Time	0.09	0.16	0.30	0.62	1.28	2.77


Sequential O(n2)
Parallel 2-Cores Total
Parallel 2-Cores Elapsed
Sequential O(n logn)
800
600
400
200
0
5000
10000
15000
20000
25000
30000



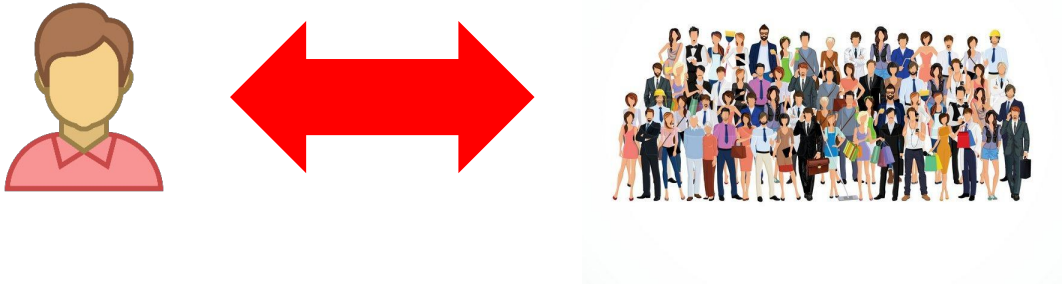
These are the results in logaritmic scale, as it is the only way to see the green line

Concurrency: Infrastructure Based Approach



- As we can see, parallel 2-cores $O(n^2)$ improved sequential $O(n^2)$ by reducing the time by half (i.e., by a factor of 2)!

Concurrency: Infrastructure Based Approach



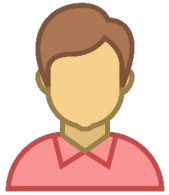
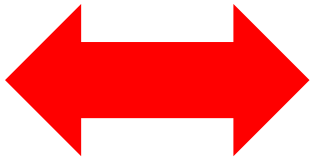
- As we can see, parallel 2-cores $O(n^2)$ improved sequential $O(n^2)$ by reducing the time by half (i.e., by a factor of 2)
- But we have seen now that sequential O(n log n) improves sequential $O(n^2)$ by a factor 253!

Concurrency: Infrastructure Based Approach

Thus...

- if we had kept using the algorithm $O(n^2)$ and he had needed to achieve an improvement factor of 253 by using distributed programming...

Concurrency: Infrastructure Based Approach

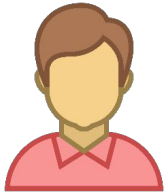
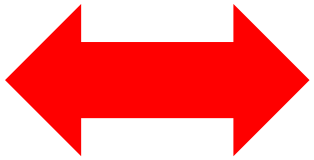

Thus...

- ...then we would have needed to rent online machines for a total of 253 cores!

Concurrency: Infrastructure Based Approach

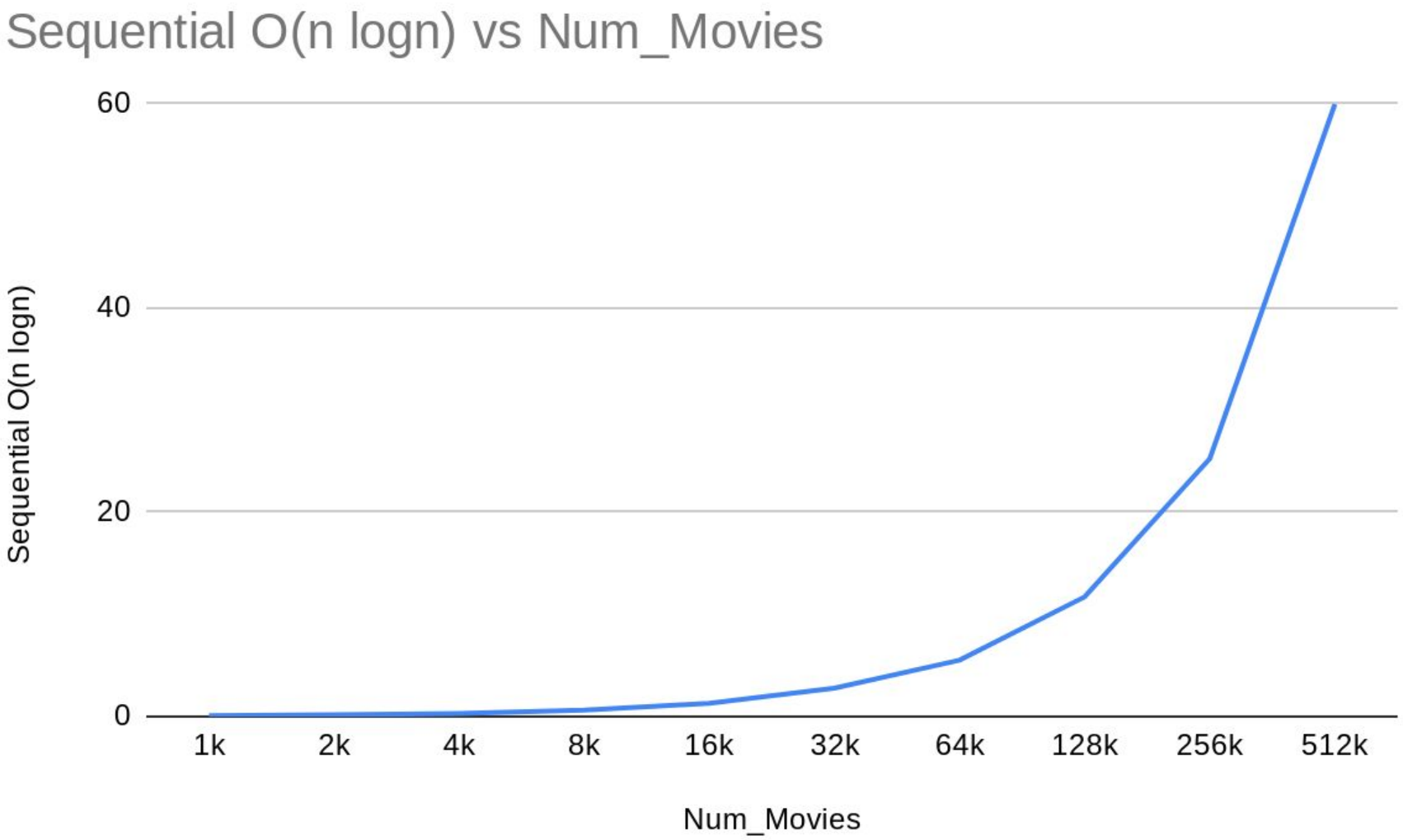
But this is not all...

Concurrency: Infrastructure Based Approach

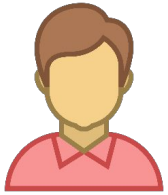
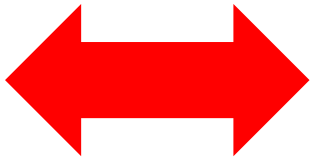




- As sequential O(nlogn) is so fast, we have now been able to solve instances in the order of *n = hundreds of thousands* in a very small amount of time

Num_Movies	1k	2k	4k	8k	16k	32k	64k	128k	256k	512k
Sequential O(n logn)	0.09	0.16	0.30	0.62	1.28	2.77	5.52	11.72	25.26	59.98

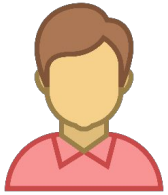
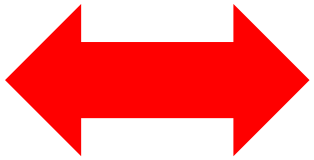

Sequential O(n logn) vs Num_Movies
60
40
20
0
Sequential O(n logn)
1k
2k
4k
8k
16k
32k
64k
128k
256k
512k
Num_Movies

Concurrency: Infrastructure Based Approach

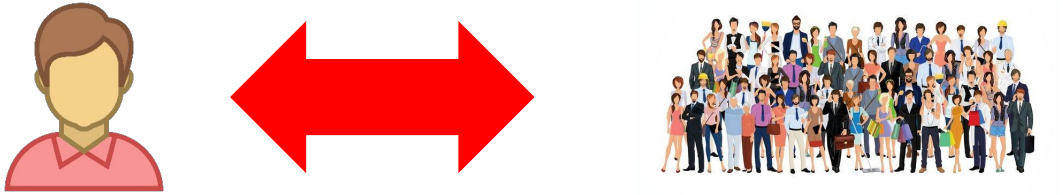

- What would have been the improvement factor of sequential O(n log n) over sequential $O(n^2)$ for an instance of 512,000 movies?
  - Maybe 5,000?
  - Maybe 10,000?

Concurrency: Infrastructure Based Approach

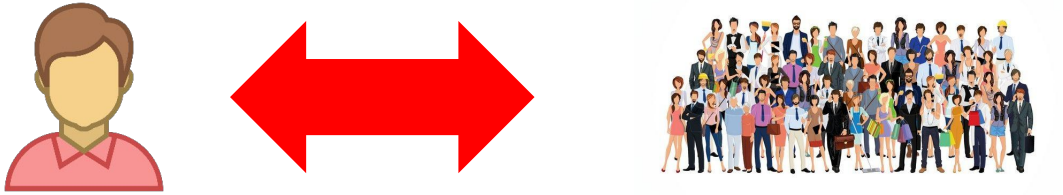

- What would have we done in that case?

Concurrency: Infrastructure Based Approach



- What would have we done in that case?
  - Renting an entire data centre with 10,000 cores?

Concurrency: Infrastructure Based Approach



- What would have we done in that case?
  - Rent an entire data centre with 10,000 cores?
  - And what for? To get the same performance as the one we could have got by using a single core and the more efficient algorithm?

Concurrency: Infrastructure Based Approach

In summary...

Concurrency: Infrastructure Based Approach

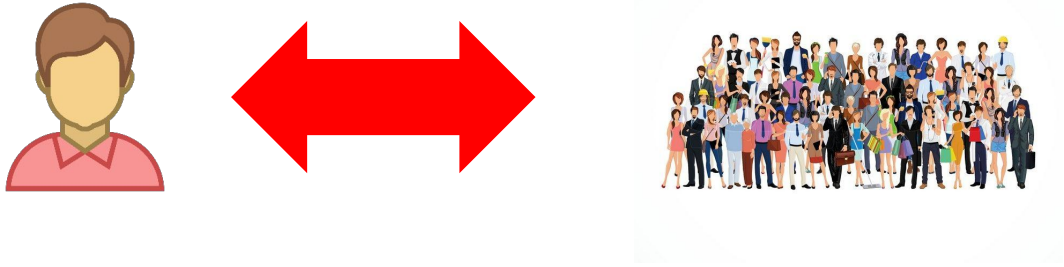
This is not a story of sequential vs distributed environments execution.

Concurrency: Infrastructure Based Approach

This is not a story of sequential vs distributed environments execution.

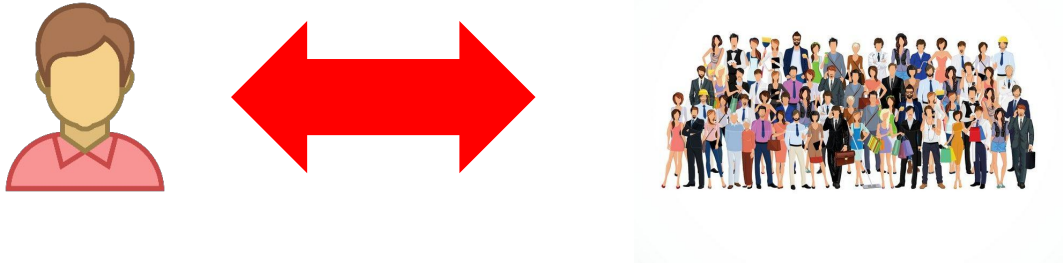
This is a story of challenge our algorithms.
Of always wonder: Can we do better?

Concurrency: Infrastructure Based Approach



Indeed, our algorithm O(n log n) can be used in a distributed environment as well, as we did for the $O(n^2)$ algorithm.

Concurrency: Infrastructure Based Approach



This will reduce our elapsed time in solving the problem in a ***c * f*** factor:

- where **c** is the number of cores in our cluster
- where **f** is the improvement factor of the O(n logn) algorithm over the $O(n^2)$ algorithm.

Outline

1. Sequential vs. Concurrent Execution.
2. List Inversions: A Real-world Example.
3. Scalability: Computational Complexity Barrier.
4. Concurrency: Infrastructure-based Approach.
5. Can We Do Better? An Algorithm-based Approach.

Thank you for your attention!