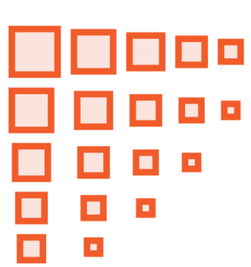
# Building on Ordered and Partially Ordered Lists

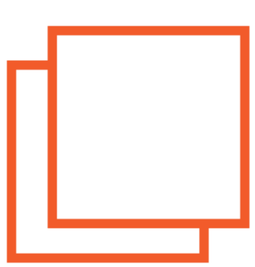
**Zoran Horvat**
CEO at Coding Helmet

@zoranh75 https://codinghelmet.com

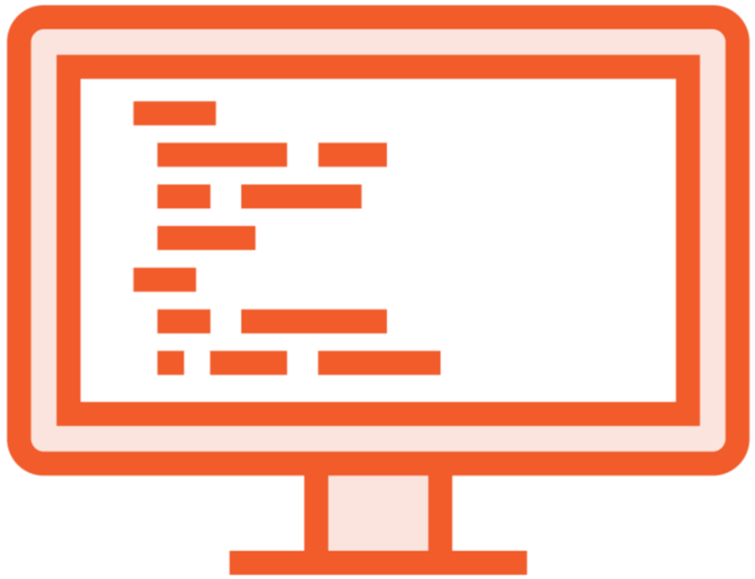
# Defining Requirements



We will add sorting requirements to linear collections



Implement a class which paginates a ***sorted*** sequence of objects



**Given**

- An unordered sequence
- Page size
- A custom sorting criterion

**Required**

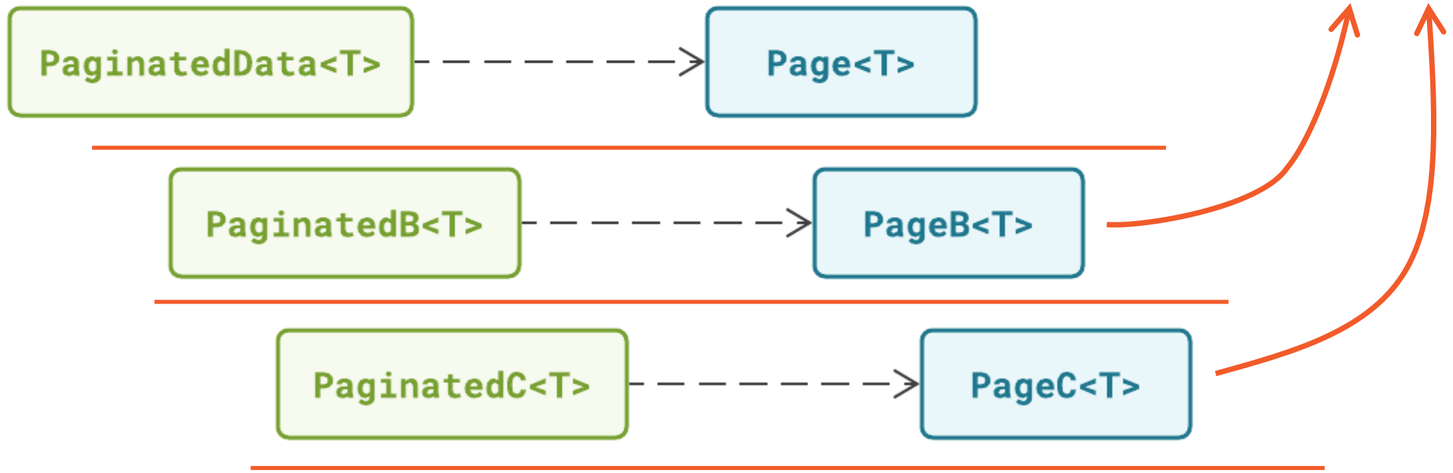
- Sort the sequence
- Divide it into pages in the sorting order

Models > Common > IPaginated.cs > { } Models.Common > Models.Common.IPaginated<T>

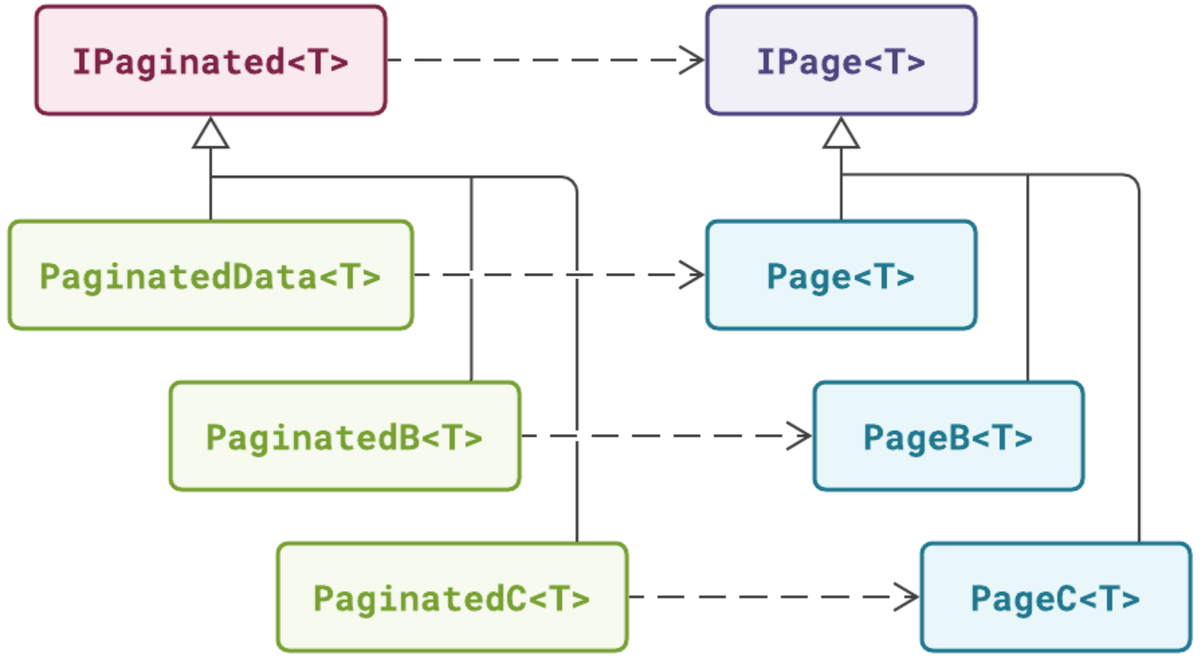


```
1 namespace Models.Common;
2
3 public interface IPaginated<T>
4 {
5
6 }
```

There will be alternate solutions later



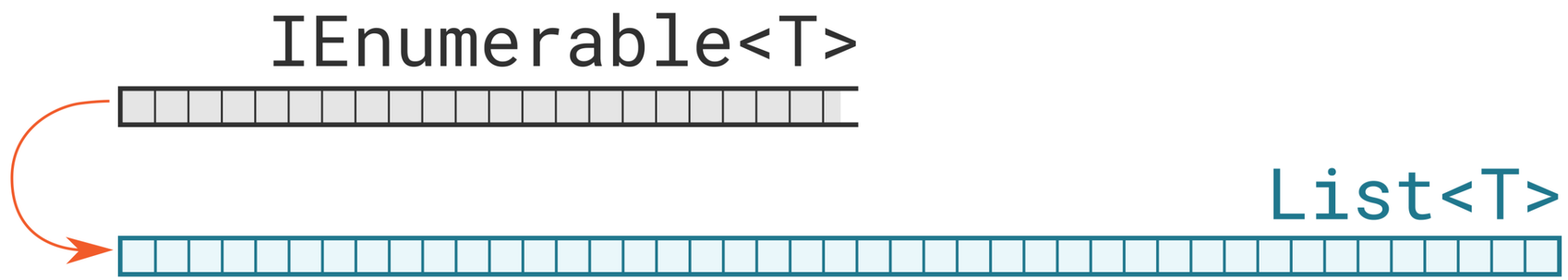
```
1 namespace Models.Common;
2
3 public interface IPaginated<T>
4 {
5
6 }
```



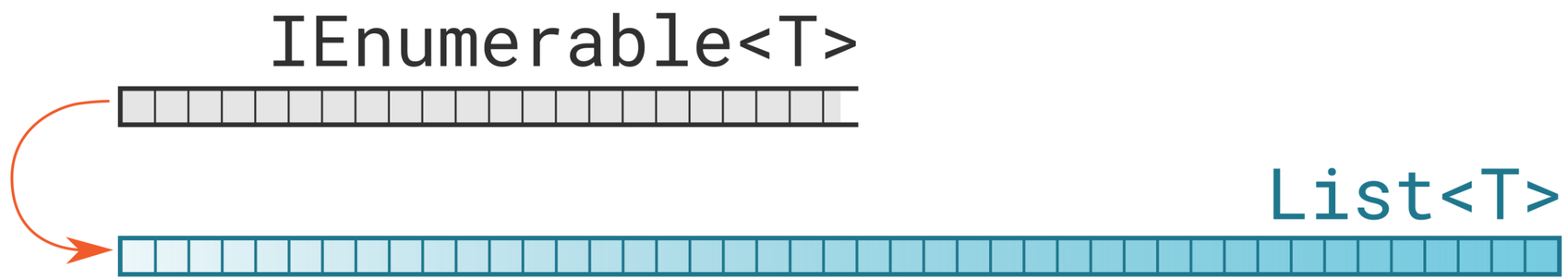
Models > Common > IPaginated.cs > { } Models.Common > Models.Common.IPaginated<T>



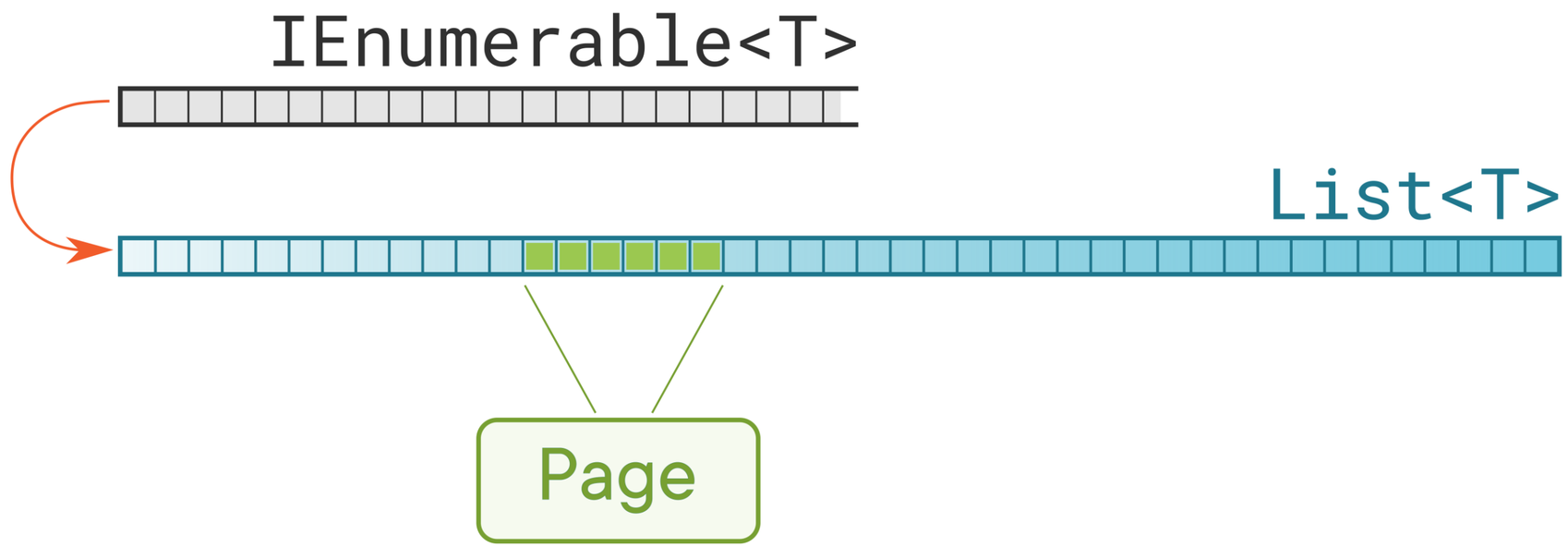
```
1 namespace Models.Common;
2
3 public interface IPaginated<T> : IEnumerable<IPage<T>>
4 {
5     int PagesCount { get; }
6     IPage<T> this[int offset] { get; }
7 }
```



```
1 namespace Models.Common;
2
3 public interface IPaginated<T> : IEnumerable<IPage<T>>
4 {
5     int PagesCount { get; }
6     IPage<T> this[int offset] { get; }
7 }
```



```
1 namespace Models.Common;
2
3 public interface IPaginated<T> : IEnumerable<IPage<T>>
4 {
5     int PagesCount { get; }
6     IPage<T> this[int offset] { get; }
7 }
```



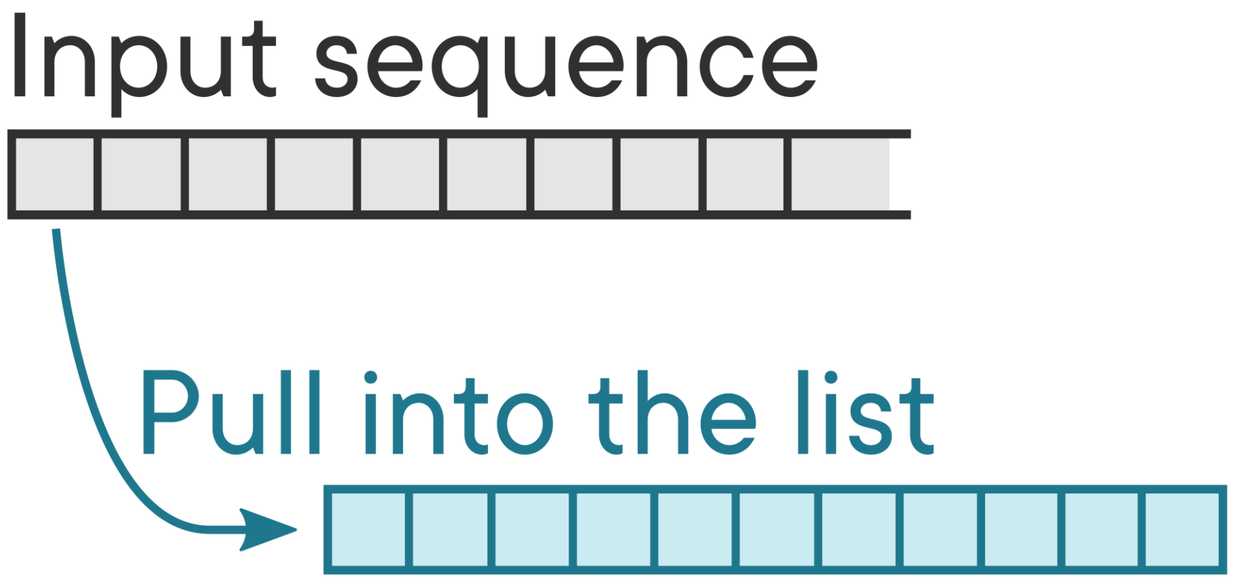
- .vscode
- ConsoleDemo
  - ConsoleDemo.csproj
  - Operators.cs
  - Program.cs
- Models
  - Common
    - Formatting
    - Pagination
      - SortedListPaginator.cs
    - Shuffling
    - ArgumentExtensions.cs
    - IPage.cs
    - IPaginated.cs
    - Operators.cs
    - SinglePassSequence.cs
  - Currency.cs
  - FinanceExtensions.cs
  - Models.csproj
  - Money.cs
  - PayRate.cs
  - Worker.cs
- Models.Tests
  - Data
    - Currencies.cs
    - ReplicatingOperators.cs
    - Workers.cs
  - Models.Tests.csproj
- ContractorsCo.sln


```
1 namespace Models.Common.Pagination;
2
3 internal class SortedListPaginator<T>
4 {
5
6 }
```

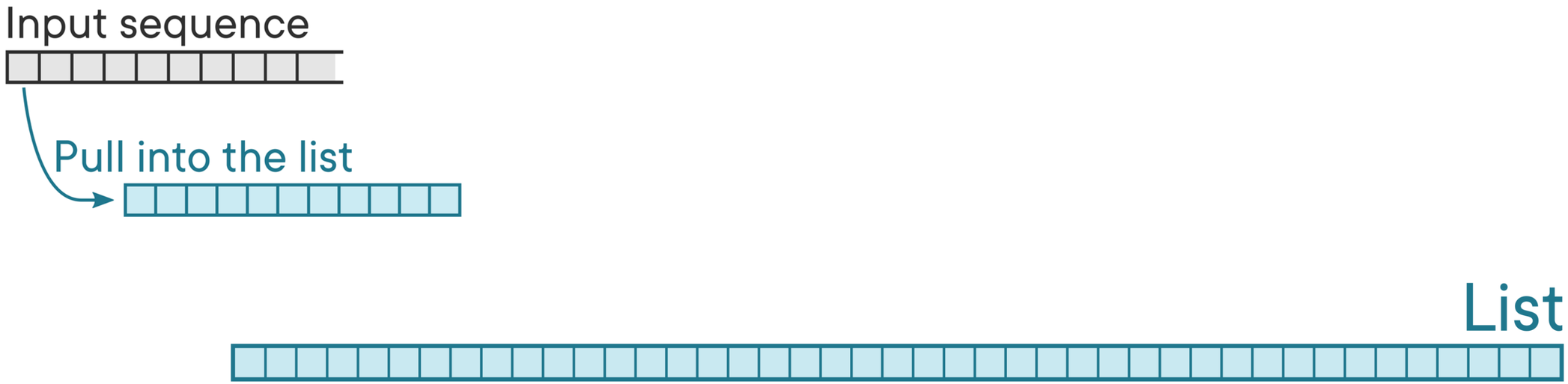
| Collection | Add items | Sort | Fully populate |
|---|---|---|---|
| `SortedList<TKey, TValue>*` | $n \times O(n)$ | | $O(n^2)$ |
| `List<T>` | $n \times O(1)$ | $O(n \log n)$ | $O(n \log n)$ |

* `SortedList` requires unique keys

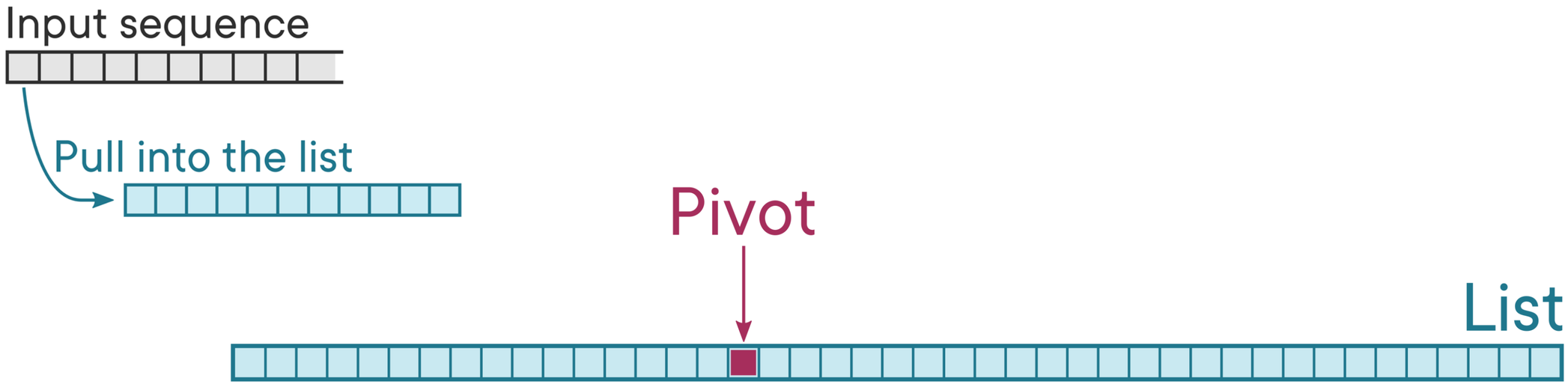
# Understanding Full Sorting Inefficiency



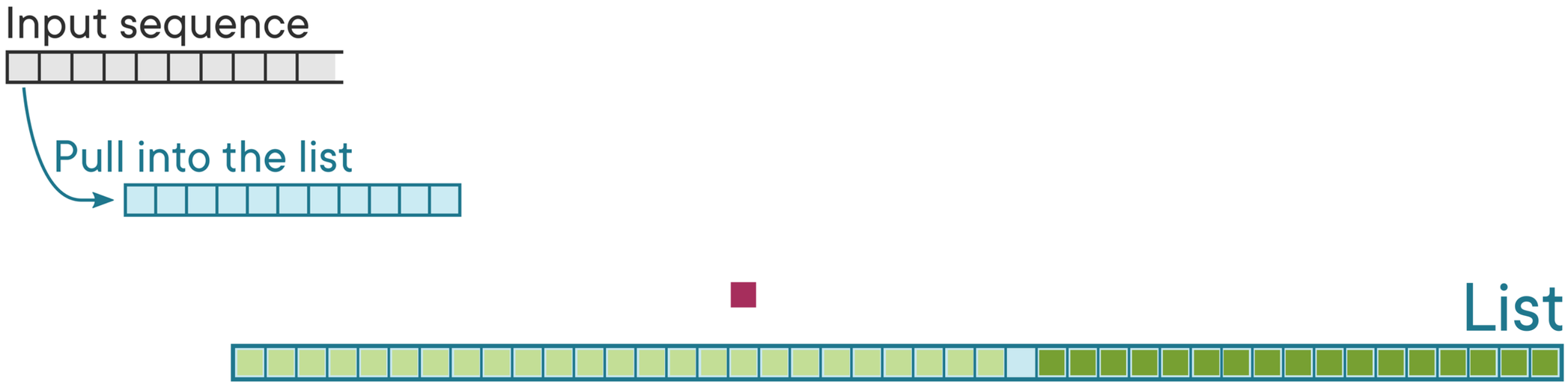
# Understanding Full Sorting Inefficiency



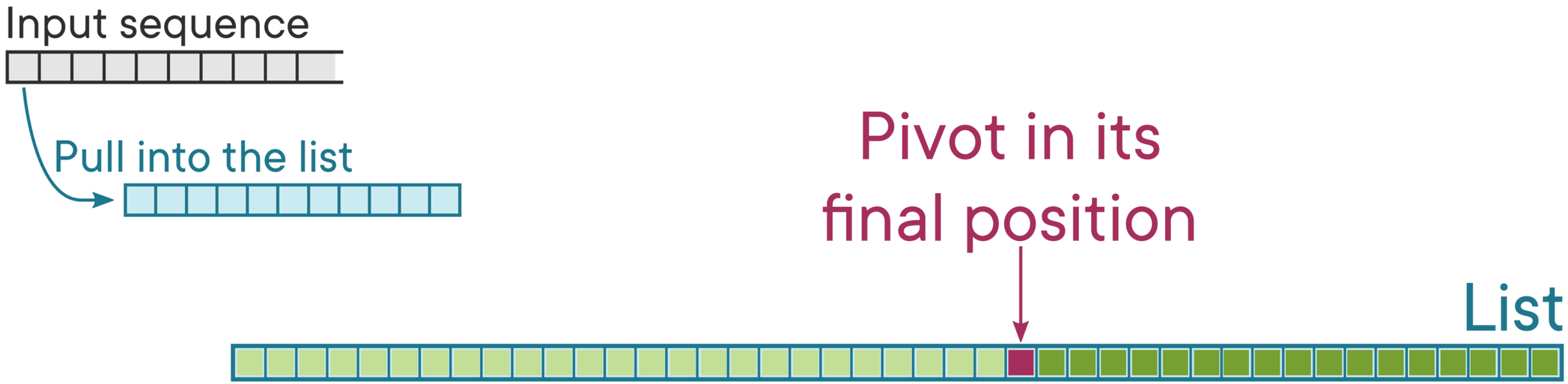
# Understanding Full Sorting Inefficiency



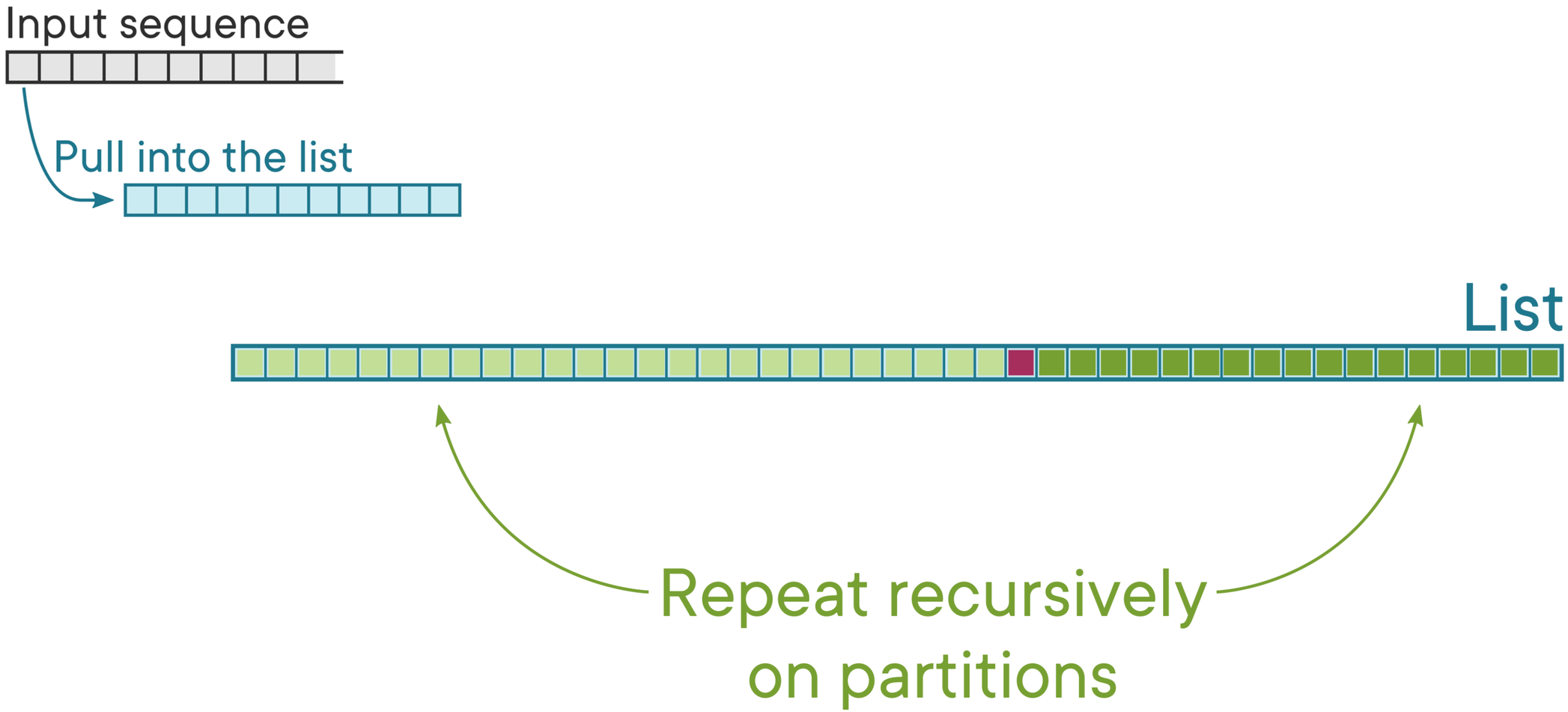
# Understanding Full Sorting Inefficiency



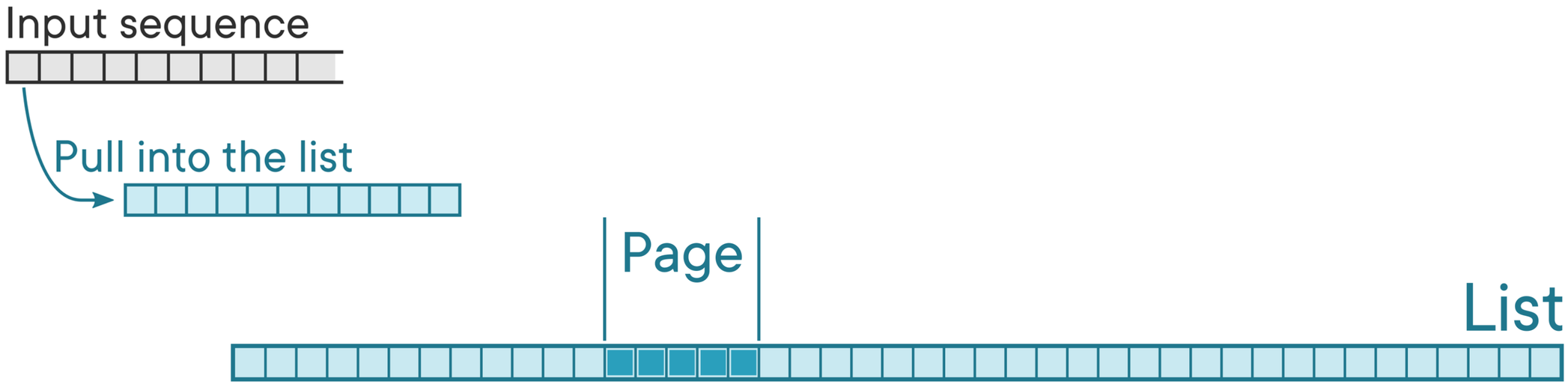
# Understanding Full Sorting Inefficiency



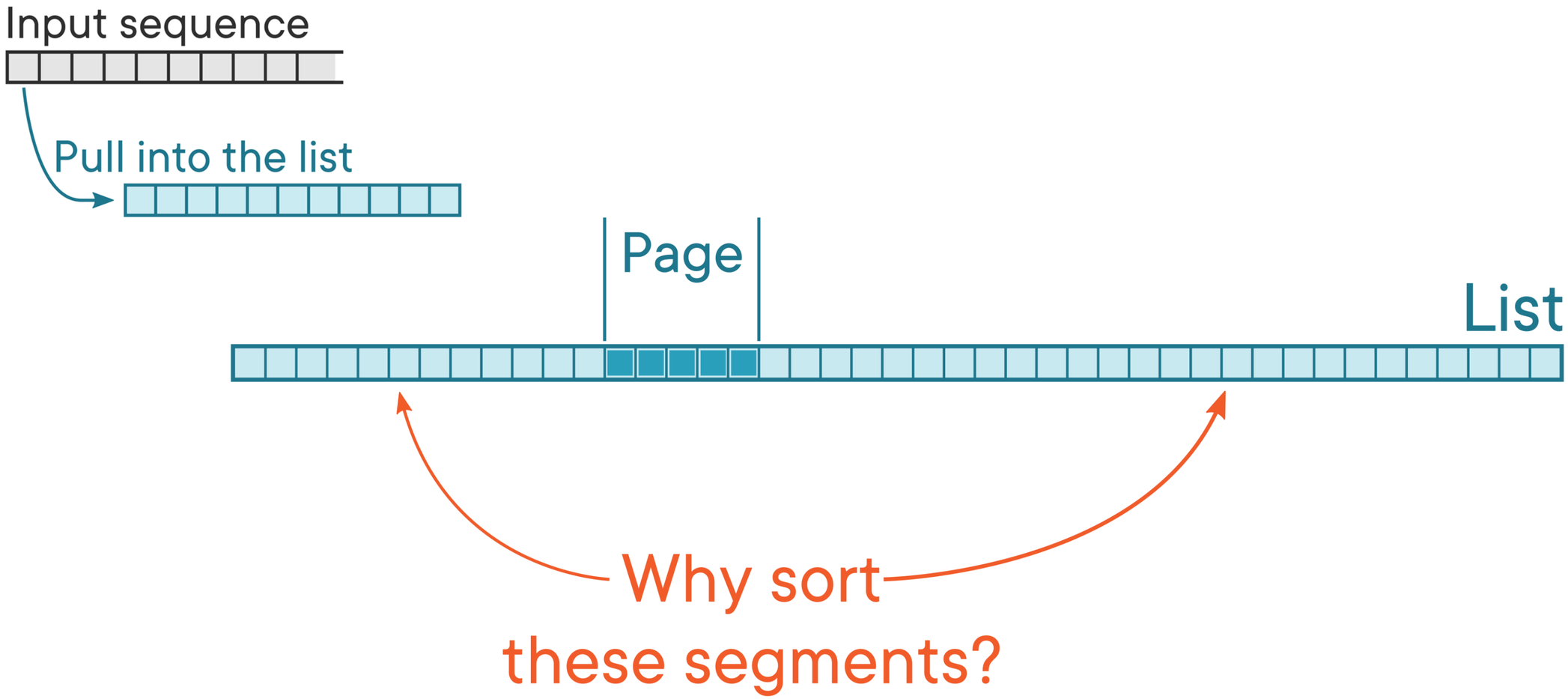
# Understanding Full Sorting Inefficiency



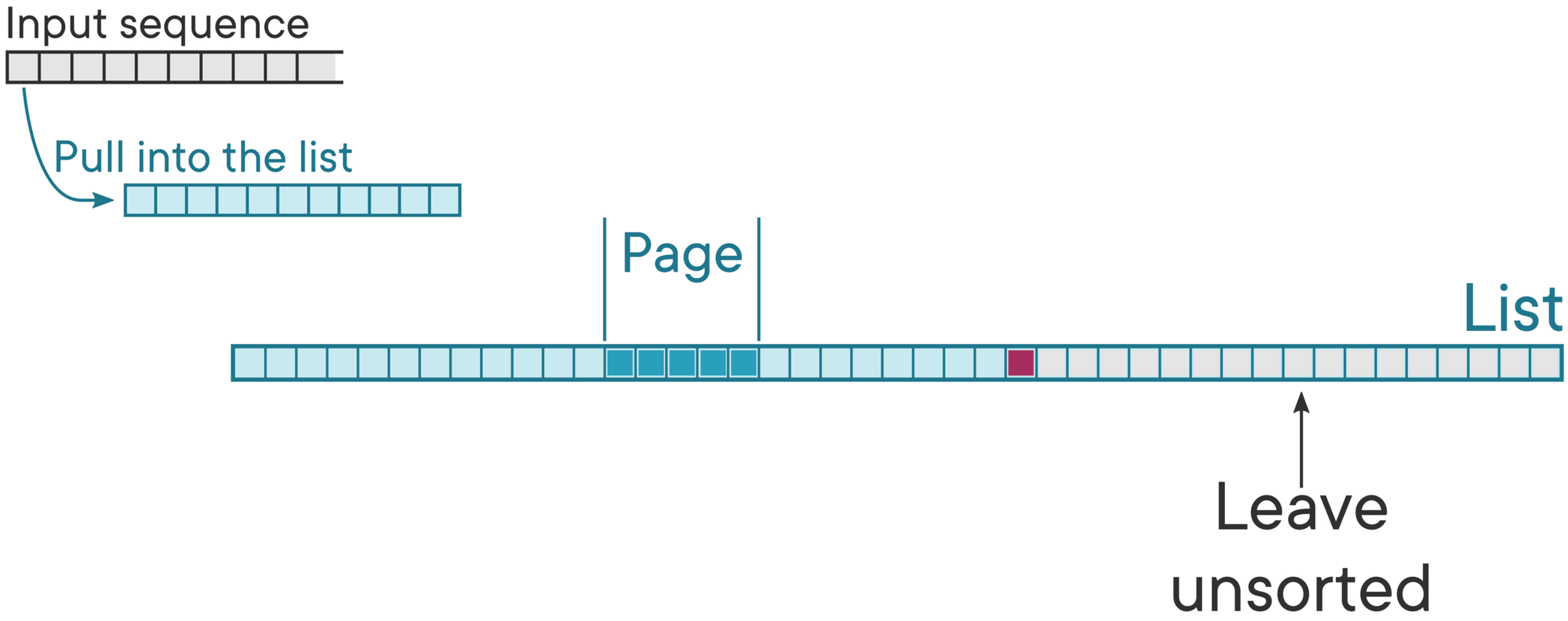
# Understanding Full Sorting Inefficiency



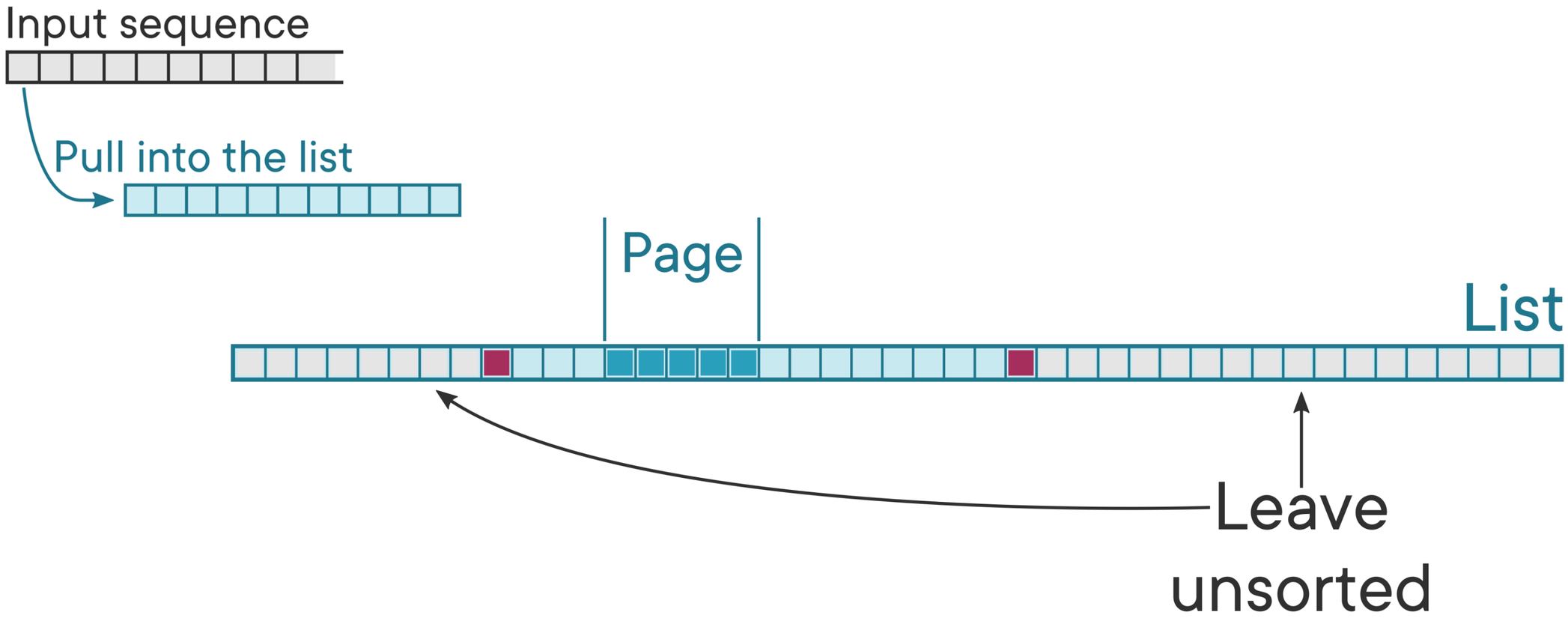
# Understanding Full Sorting Inefficiency



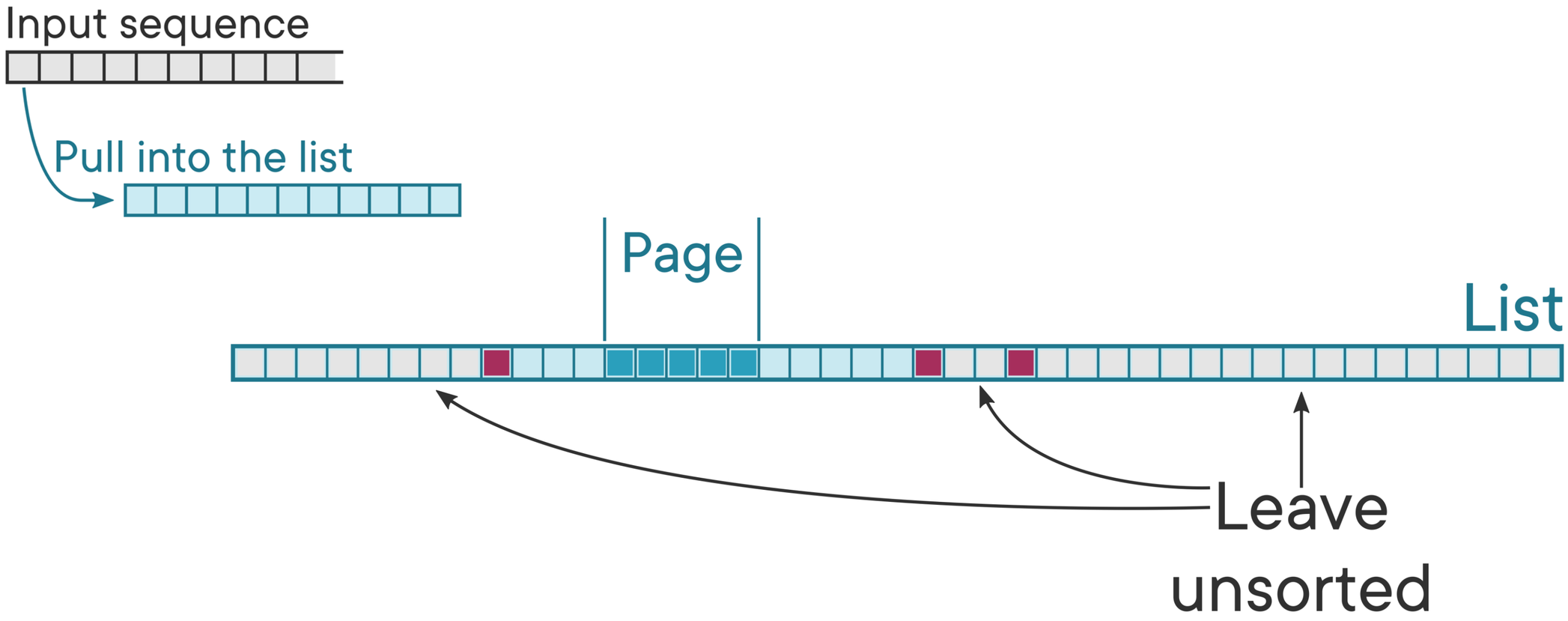
# Understanding Full Sorting Inefficiency



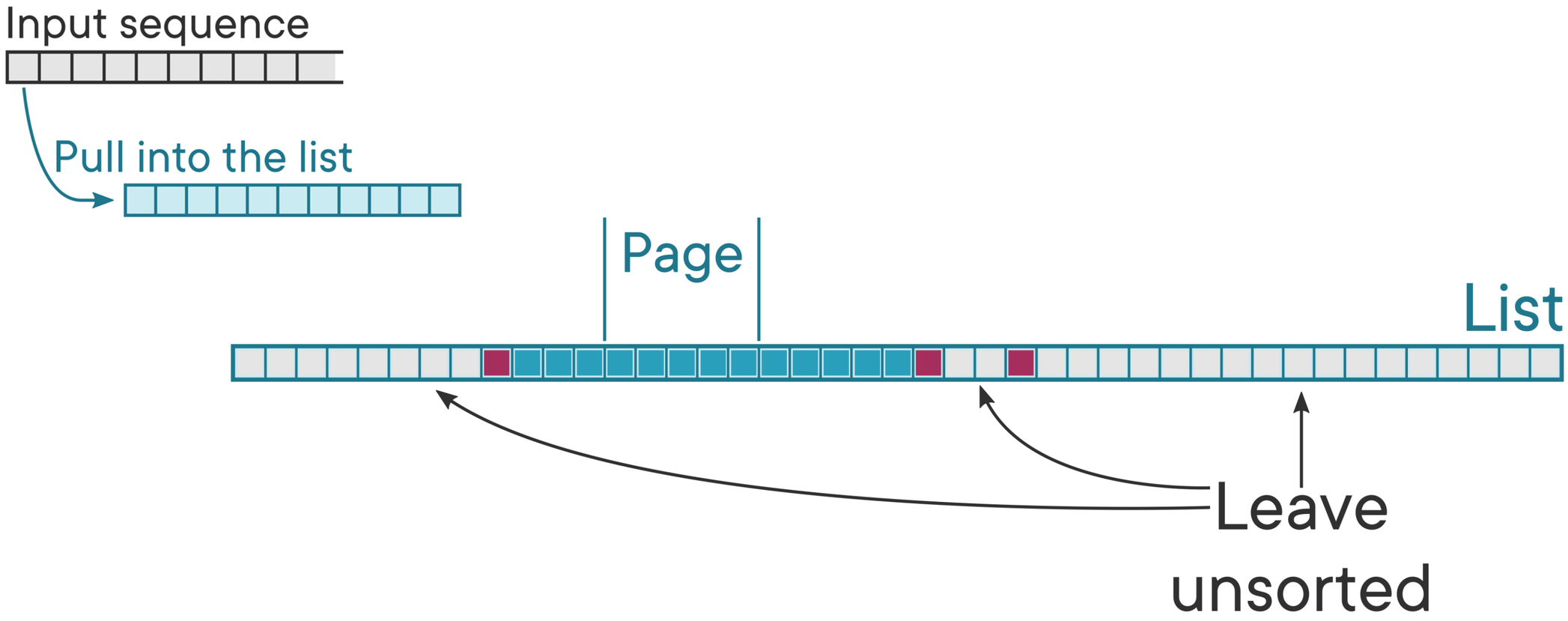
# Understanding Full Sorting Inefficiency



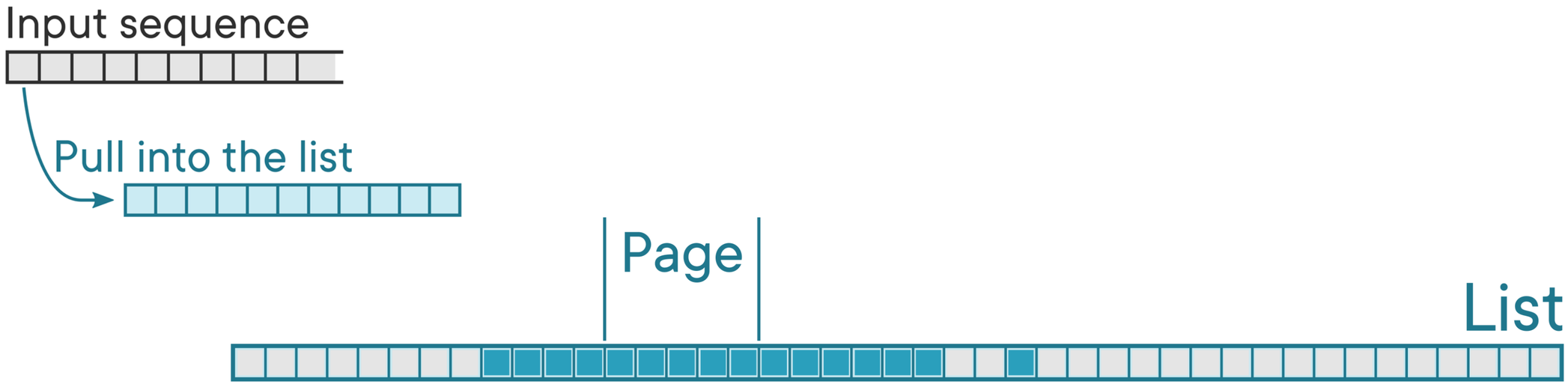
# Understanding Full Sorting Inefficiency



# Understanding Full Sorting Inefficiency



# Understanding Full Sorting Inefficiency



# Summary

**Working with a sorted list**

- Sorting brings great benefits
- But sorting takes time
- It requires access to all items

**The problem of partitioning a sorted list**

- Addressable in several ways
- Solutions can target specific use cases

# Summary

**Applying object-oriented design**

- Abstract types to specify promises clearly
- Admit concrete implementations later
- All functional promises must be satisfied
- Nonfunctional promises vary by concrete implementation

# Summary

**OO design applied to pagination**

- Using amortized complexity assumption
- Repeated calls amortize high initial cost
- Implementation underperforms if caller behaves differently
- Alternatively use a partially sorted list

# Up Next: Optimizing Performance