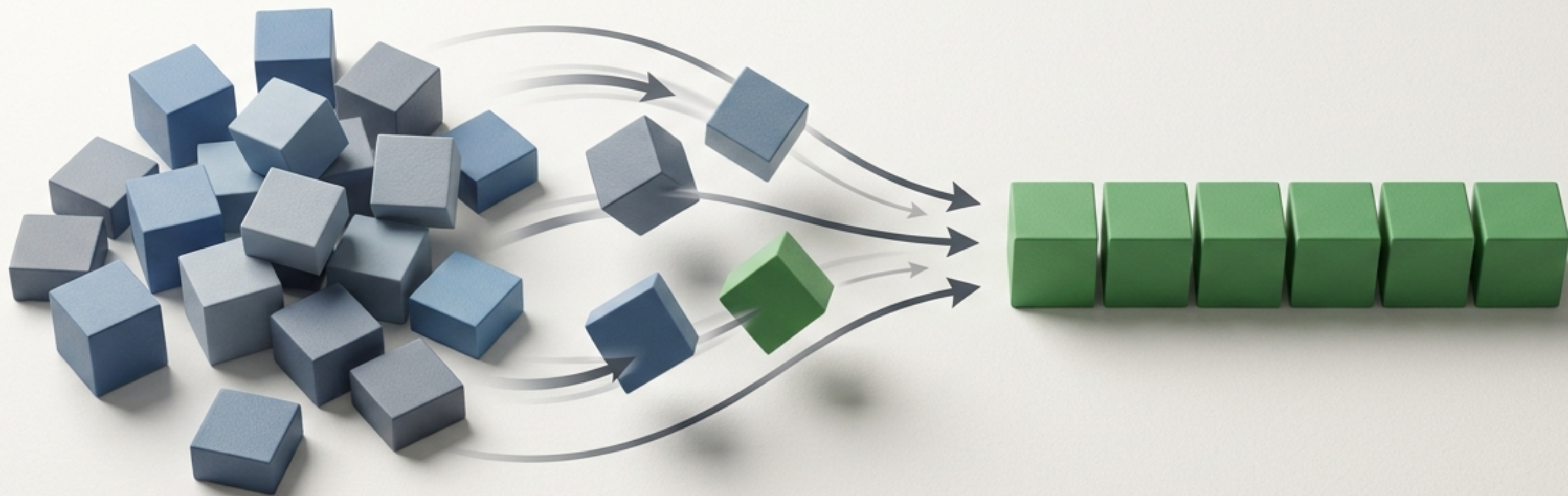


図解でわかる！クイックソート入門

データ構造とアルゴリズムを直感的に理解する



クイックソートの重要性



誕生は1960年

アントニー・ホーア考案。
半世紀以上使われる名作
アルゴリズム



圧倒的なスピード

非常に高速で効率的な並
べ替えアルゴリズム



世界の標準

多くのプログラミング言
語の標準ライブラリで採用

コアとなる哲学：分割統治法 (Divide and Conquer)



**大きな問題は、
そのまま解かない。**

データを小さく分けてから、それぞれを解決していく戦略です。
クイックソートの根幹はこの「分割」にあります。

アルゴリズムの3つの黄金ルール

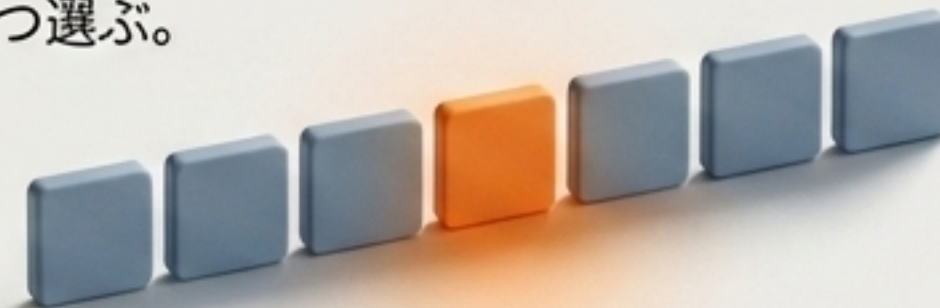
3. 再帰 (リピート)

分けられたグループ
に対し、データが1つ
(または0) になるまで
手順1~2を繰り返す。



1. 軸 (ピボット) の選択

データの中から基準となる値を
1つ選ぶ。

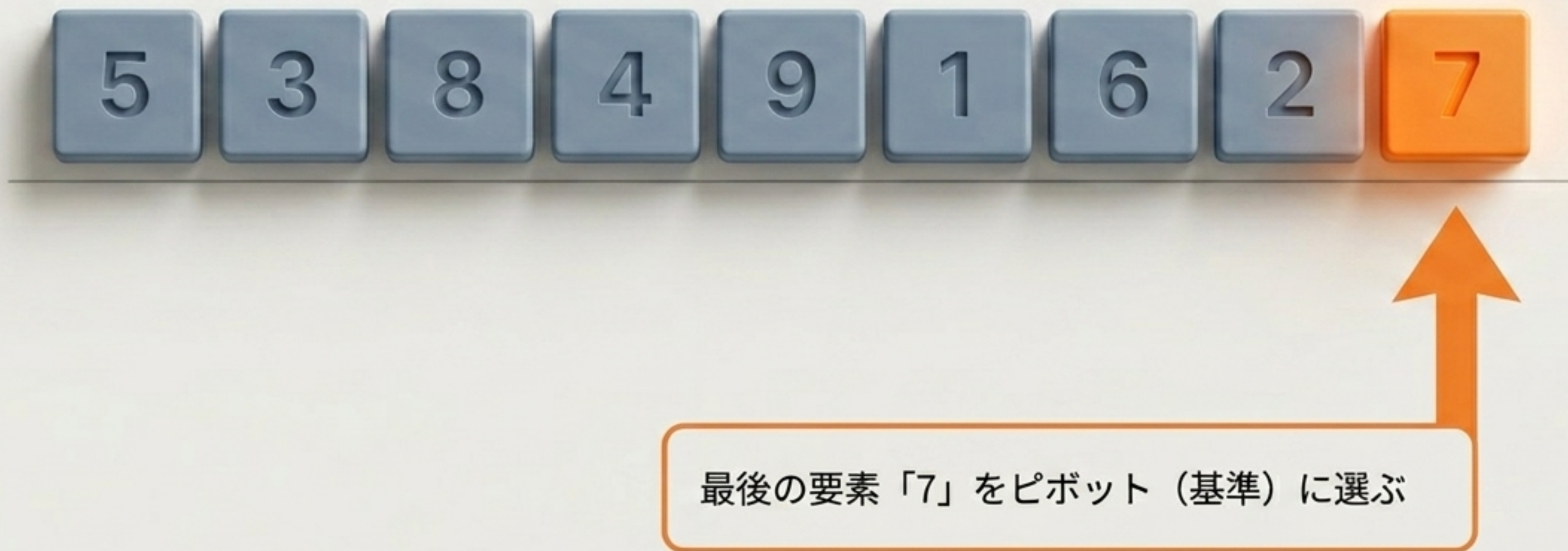


2. 分割 (パーティション)

ピボットより「小さい値」と「大
きい値」のグループに分ける。



ステップ1：スタートラインとピボットの決定



ステップ2：ピボットを基準にデータを分割（パーティション）

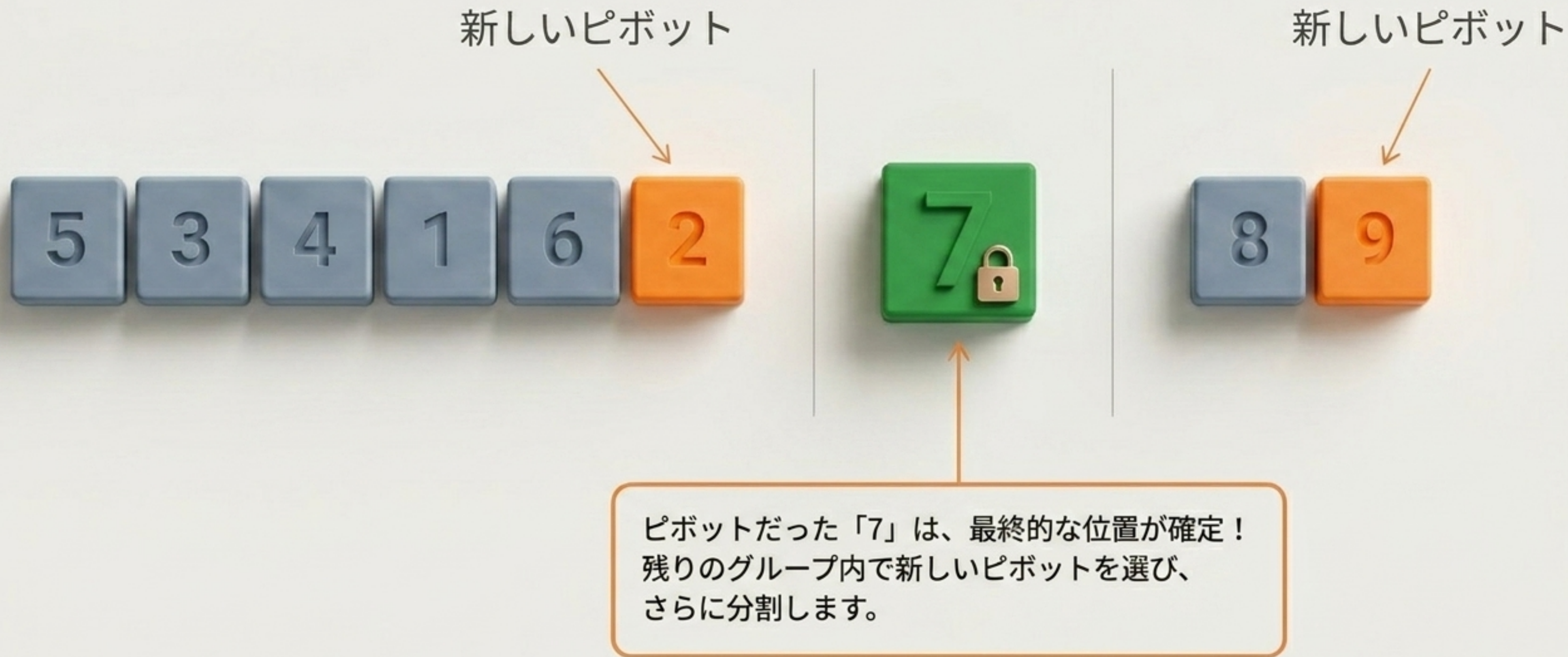
7より小さいグループ



7より大きいグループ

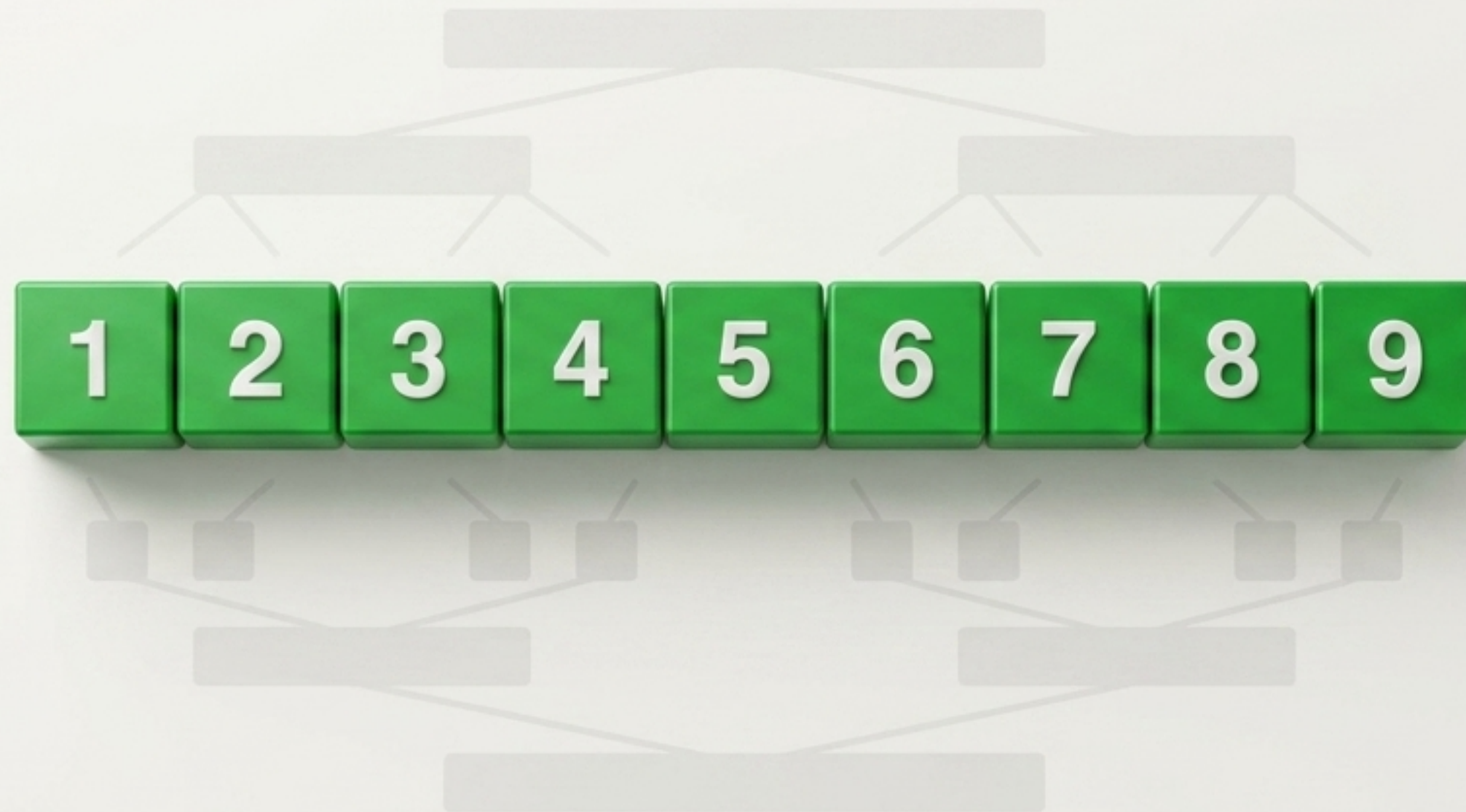


ステップ3：左右のグループで同じ操作を繰り返す（再帰）



最終ステップ：すべての要素が結合し、ソート完了

各グループの要素が1つになるまで分割を繰り返すと、最終的にすべてのデータが正しい順序で結合されます。





圧倒的な平均速度 ($O(n \log n)$)

平均的なケースでは非常に高速。

メモリの節約

追加のメモリ空間をほとんど必要としない。



最悪のケース ($O(n^2)$)

既に整列されたデータで端をピボットにするなど、選び方が悪いと速度が急激に低下。

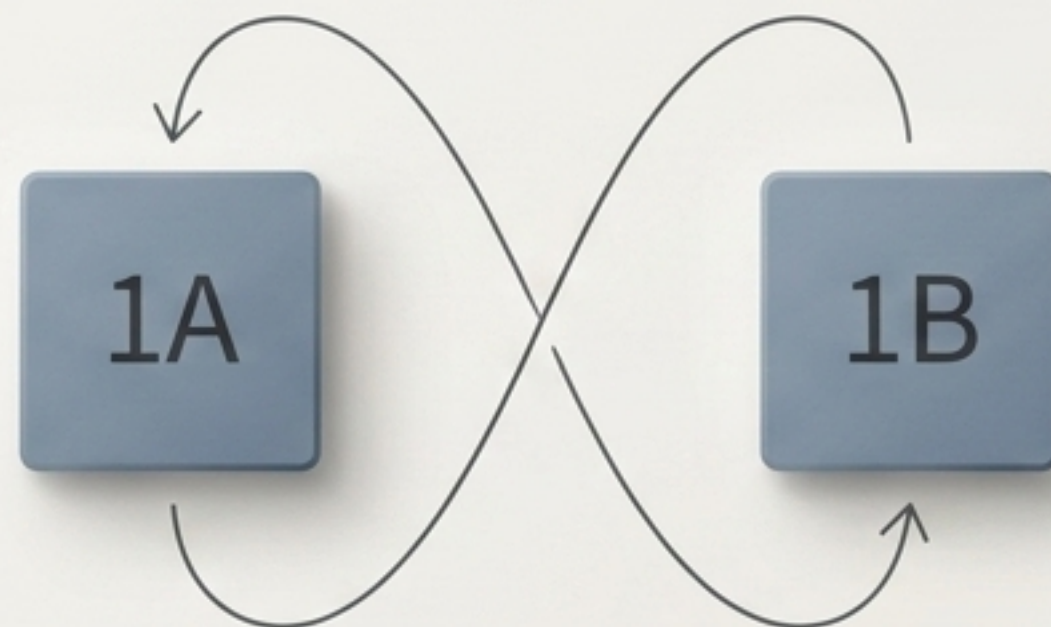
インプレース / In-place



元の配列内で入れ替え

別の巨大な箱（メモリ）を用意せず、今ある場所の中でデータを並べ替えるため、省メモリ。

不安定なソート / Unstable

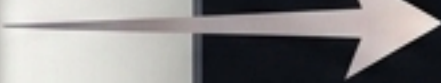


同じ値の順序が入れ替わる

「安定ソート」ではないため、同じ値を持つ要素の元の前後関係が保たれないことがあります。

再帰の終了条件

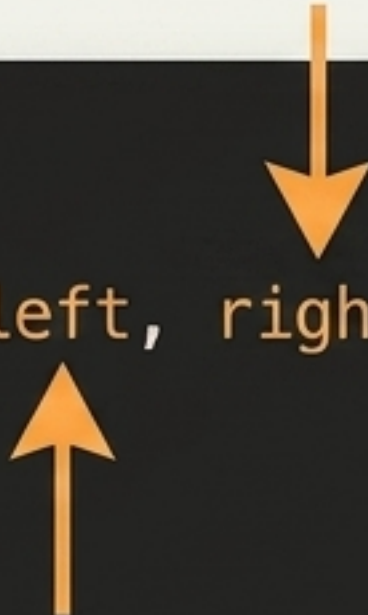
データが1つ以下になれば、
それ以上分割する必要はない
のでストップ。



```
def quick_sort(arr):  
    if len(arr) <= 1:  
        return arr
```

使用例

```
data = [3, 6, 8, 10, 1, 2, 1]  
print(f"ソート後: {quick_sort(data)}")
```



```
def quick_sort_inplace(arr, left, right):  
    if left >= right:  
        return
```

実行例

```
data = [5, 3, 8, 4, 9, 1, 6, 2, 7]  
quick_sort_inplace(data, 0, len(data) - 1)
```

インプレース版の工夫

新しい配列を作らず、left（左端）とright（右端）のインデックス（位置情報）だけを動かしてメモリを節約します。

まとめ：クイックソートの3つの要点



分割統治法で解く

基準（ピボット）を決めて、大小に分割する作業を繰り返す。



速いが、最悪ケースに注意

平均的には超高速だが、ピボットの選び方によっては速度が落ちる ($O(n^2)$)。



省メモリ（インプレース）

元の配列のまま並べ替えが可能。ただし、順序が入れ替わる可能性（不安定）がある。