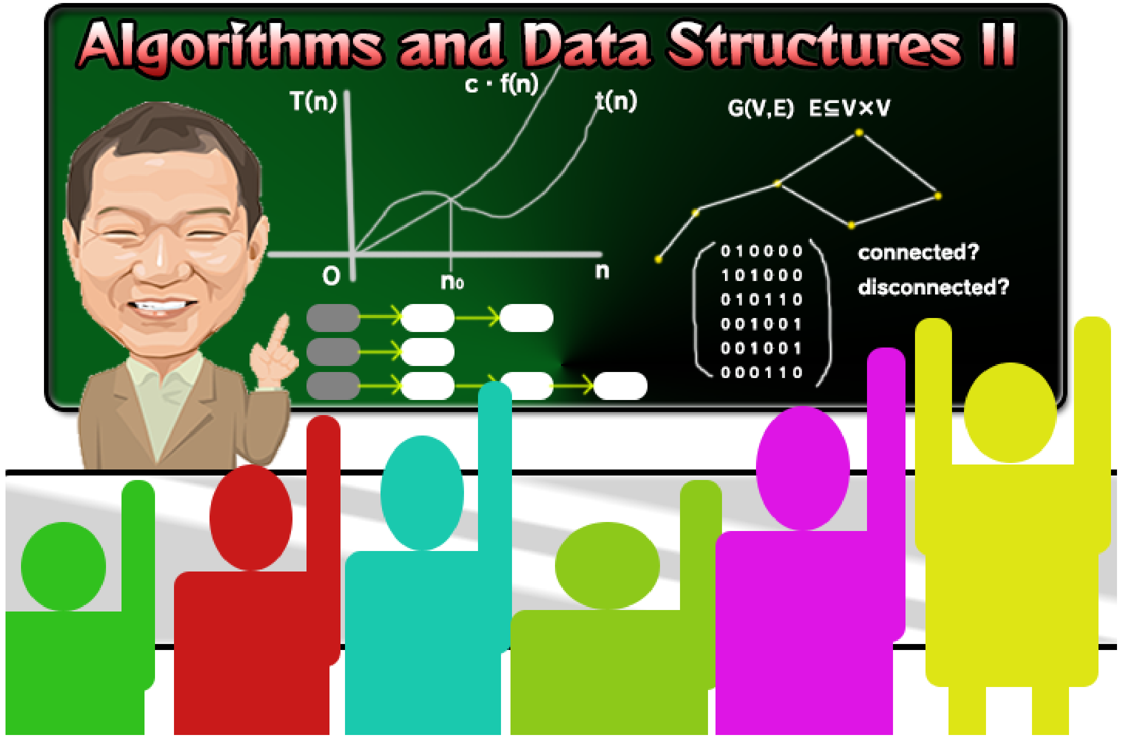


アルゴリズムとデータ構造 II
Algorithms and Data Structures II

Exercise 09. Algorithm Design Techniques II

Exercise 08-11のために

アルゴリズム設計のためのストラテジー
（問題を解くための、広い意味での解法の考え方）

- Greedy Algorithm (Ex08)
- **Divide and Conquer (Ex09)**
- Dynamic Programming (Ex10)
- Backtracking (Ex11)

「どのようにプログラムを書いたか？」までが重要です。
プログラムのソースコードの提出も必須です。

[Ex09] Divide and Conquer (分割統治法)

・与えられた問題を小さく（そして全く同じ問題に）分割して、小さな部分問題ごとに解く

⇒ 部分問題の解を集めてまとめる

問題サイズが小さくなっただけの「全く同じ問題を解く」という性質上、再帰実装をする機会も多い

具体例

- Merge sort
- Strassen’s Algorithm (Ex09)

Strassen's Algorithm

・行列積を（高速に？）計算するアルゴリズム

通常のｎ次の行列乗算（代数的な定義）

$$A'_{ij} = \sum_{k=1}^{n} (A_{ik} \cdot A_{kj})$$

プログラムで書こうとすると…
（APPの単純な演算定義）

$$A'_{ij} = A'_{ij} + A_{ik} \times A_{kj}$$

オーダーは O(n^3)

Strassen's Algorithm はこれを少し改善する

乗算１回のコストは
加算１回よりも重い。
⇒　できるだけ乗算を減らしたい…

Strassen's Algorithm

・まず、通常の行列積

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \cdot \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$C_{11} = A_{11} \cdot B_{11} + A_{12} \cdot B_{21}$$

$$C_{12} = A_{11} \cdot B_{12} + A_{12} \cdot B_{22}$$

$$C_{21} = A_{21} \cdot B_{11} + A_{22} \cdot B_{21}$$

$$C_{22} = A_{21} \cdot B_{12} + A_{22} \cdot B_{22}$$

Strassen's Algorithm

計算方法を変える

・ Pを7種類計算

・7つのPをまとめる

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \cdot \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$P_1 = (A_{12} - A_{22})(B_{21} + B_{22})$$

$$P_2 = (A_{11} + A_{22})(B_{11} + B_{22})$$

$$P_3 = (A_{11} - A_{21})(B_{11} + B_{12})$$

$$P_4 = (A_{11} + A_{12})B_{22}$$

$$P_5 = A_{11}(B_{12} - B_{22})$$

$$P_6 = A_{22}(B_{21} - B_{11})$$

$$P_7 = (A_{21} + A_{22})B_{11}$$

$$C_{11} = P_1 + P_2 - P_4 + P_6$$

$$C_{12} = P_4 + P_5$$

$$C_{21} = P_6 + P_7$$

$$C_{22} = P_2 - P_3 + P_5 - P_7$$

Strassen's Algorithm

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \cdot \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

これは2x2の正方行列の場合　⇒ 一般の場合は？

⇒ 問題を分割する　(Divide and Conquerの登場)

行列全体を４分割して、各ブロック行列ごとに解く

⇒ 2x2になるまで分割すれば、そのまま適用可能

$$\begin{aligned}
P_1 &= (A_{12} - A_{22})(B_{21} + B_{22}) \\
P_2 &= (A_{11} + A_{22})(B_{11} + B_{22}) \\
P_3 &= (A_{11} - A_{21})(B_{11} + B_{12}) \\
P_4 &= (A_{11} + A_{12})B_{22} \\
P_5 &= A_{11}(B_{12} - B_{22}) \\
P_6 &= A_{22}(B_{21} - B_{11}) \\
P_7 &= (A_{21} + A_{22})B_{11}
\end{aligned}$$

$$\begin{aligned}
C_{11} &= P_1 + P_2 - P_4 + P_6 \\
C_{12} &= P_4 + P_5 \\
C_{21} &= P_6 + P_7 \\
C_{22} &= P_2 - P_3 + P_5 - P_7
\end{aligned}$$

Strassen’s Algorithm

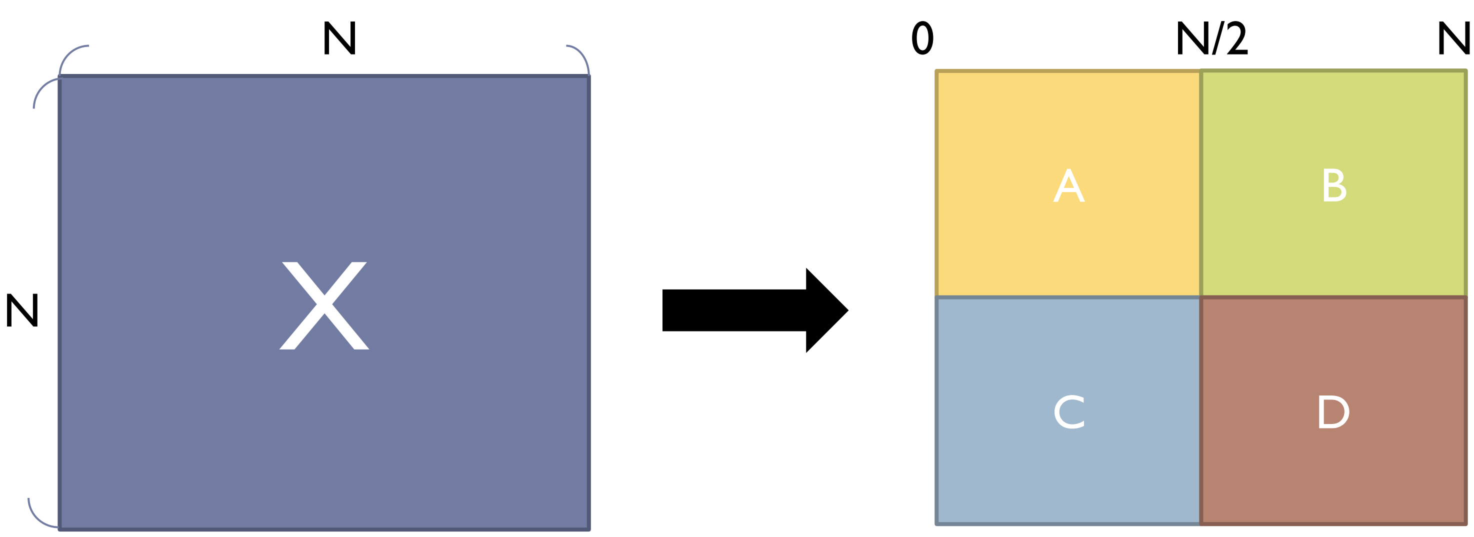
実装の場合、

どこで分割したかを記録 (変数 mid をうまく使う)

・分割前の境界線を適切に管理して、問題を分けることが重要

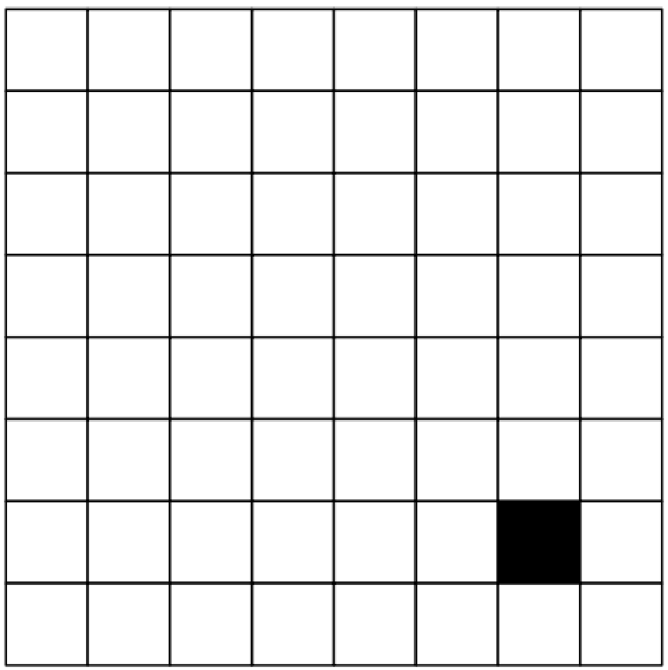
・分割の途中、以下のAやB, C, m はスカラーではなく行列

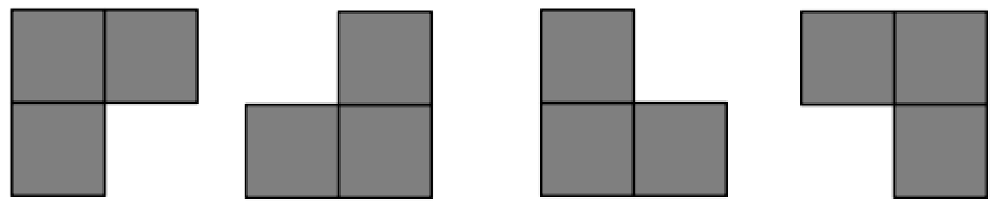
⇒ mid をうまく使わないとデータがめちゃくちゃになるので注意



（類題）ブロック敷き詰めパズル

・正方格子の**１区画だけ黒いブロックが埋まっている**
・３ブロック分の大きさのＬ字ブロックを使って
残りの区画を全てきれい埋める





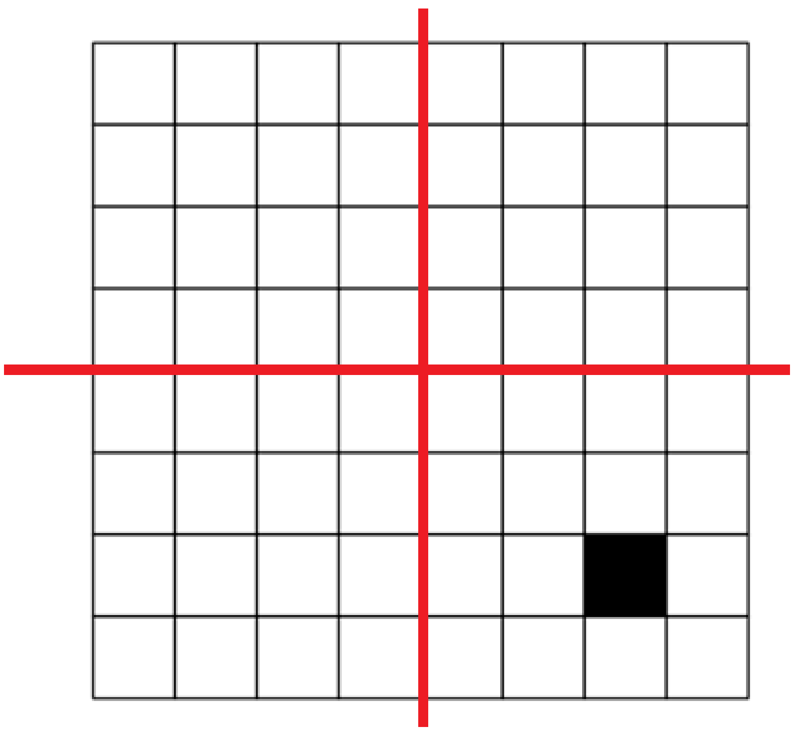
⇒　分割統治法で解く

（類題）ブロック敷き詰めパズル

・４分割してみる

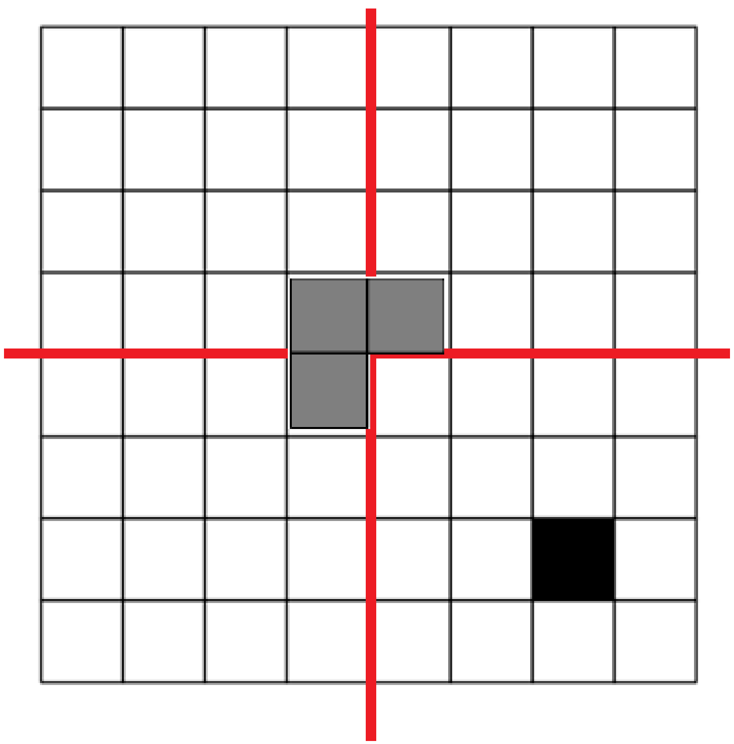
⇒ ４分割のうち、１区画だけブロックが１個既にある

⇒ 「各分割で同じ問題を解いている」とは言えない



（類題）ブロック敷き詰めパズル

・分割する直前に、中央にＬ字ブロックを１個置いてみる
・これで４分割すれば、全ての部分問題は
「１個のブロックが置かれた状態」になる　⇒　再帰可能



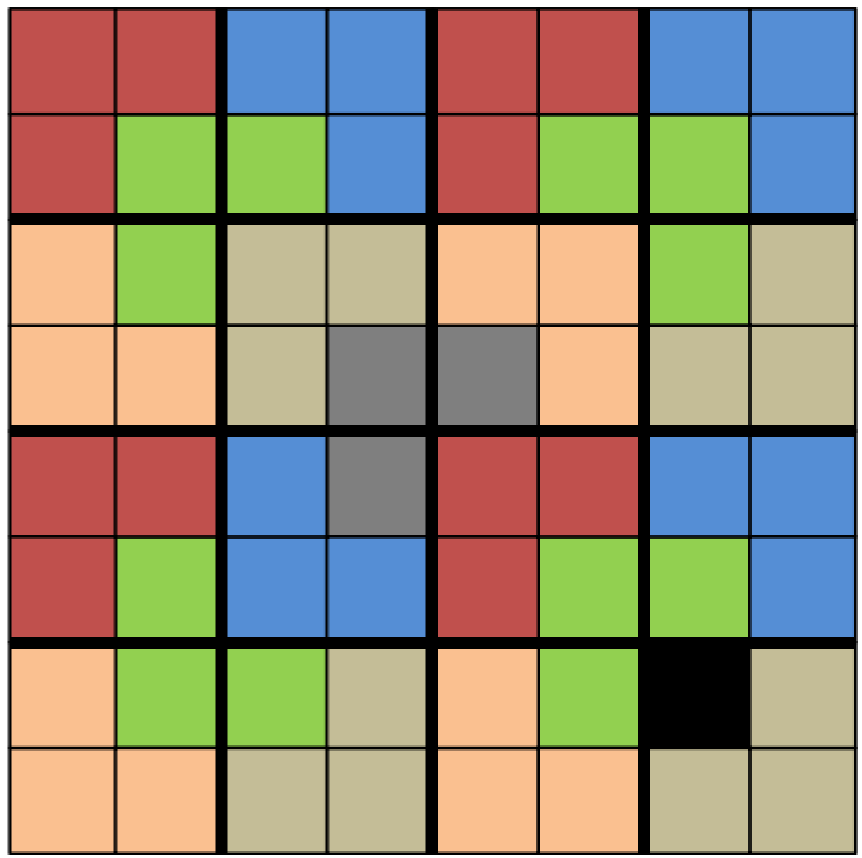
４分割して、それぞれ再帰（４回）

$$T(n) = 4T(\tfrac{n}{2}) + \Theta(f(n))$$

フルコーディングなら、Strassen
よりもとっつきやすいはず

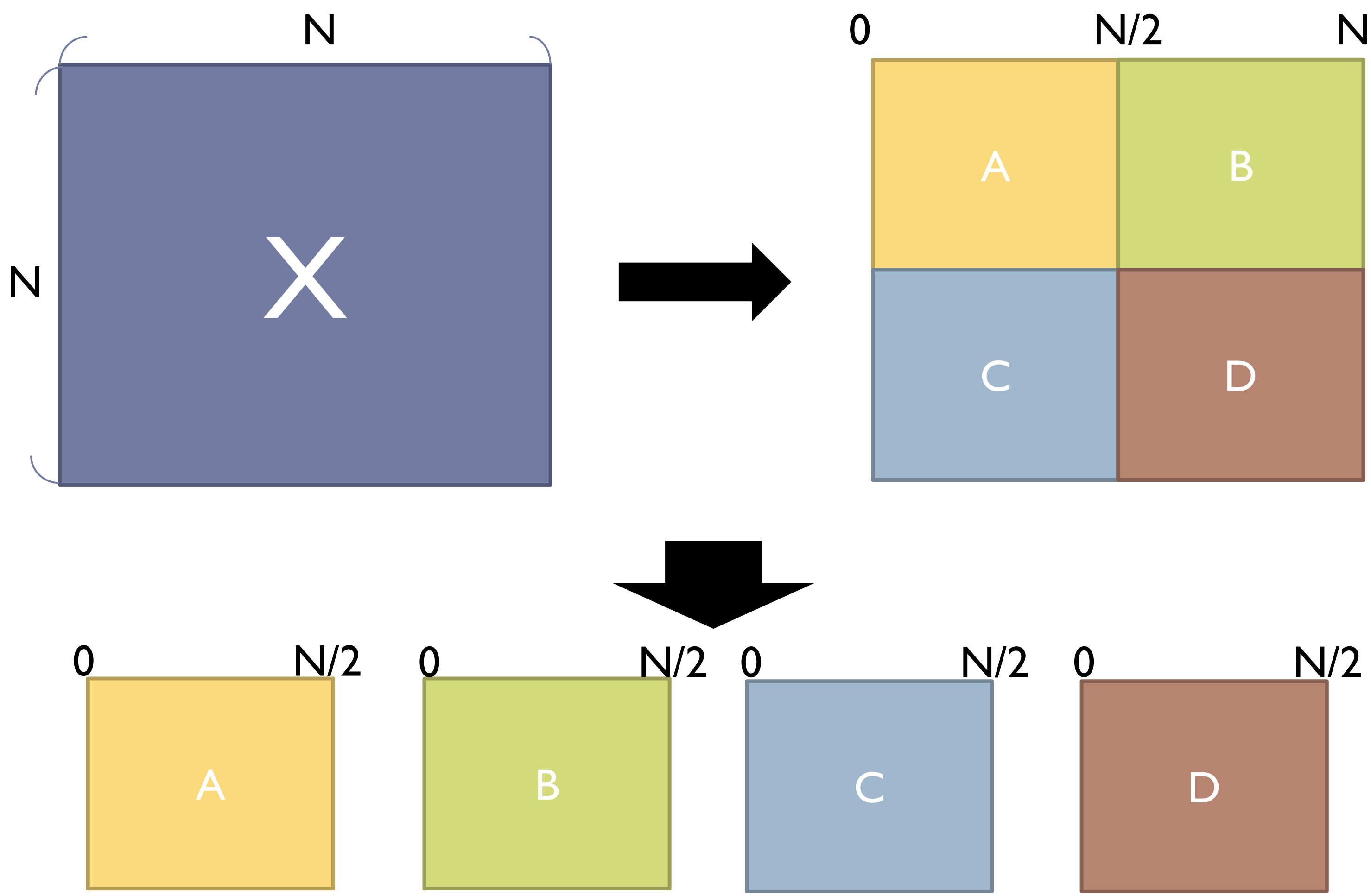
（類題）ブロック敷き詰めパズル

・ちゃんと解ければこんな感じ　（８ｘ８の場合）



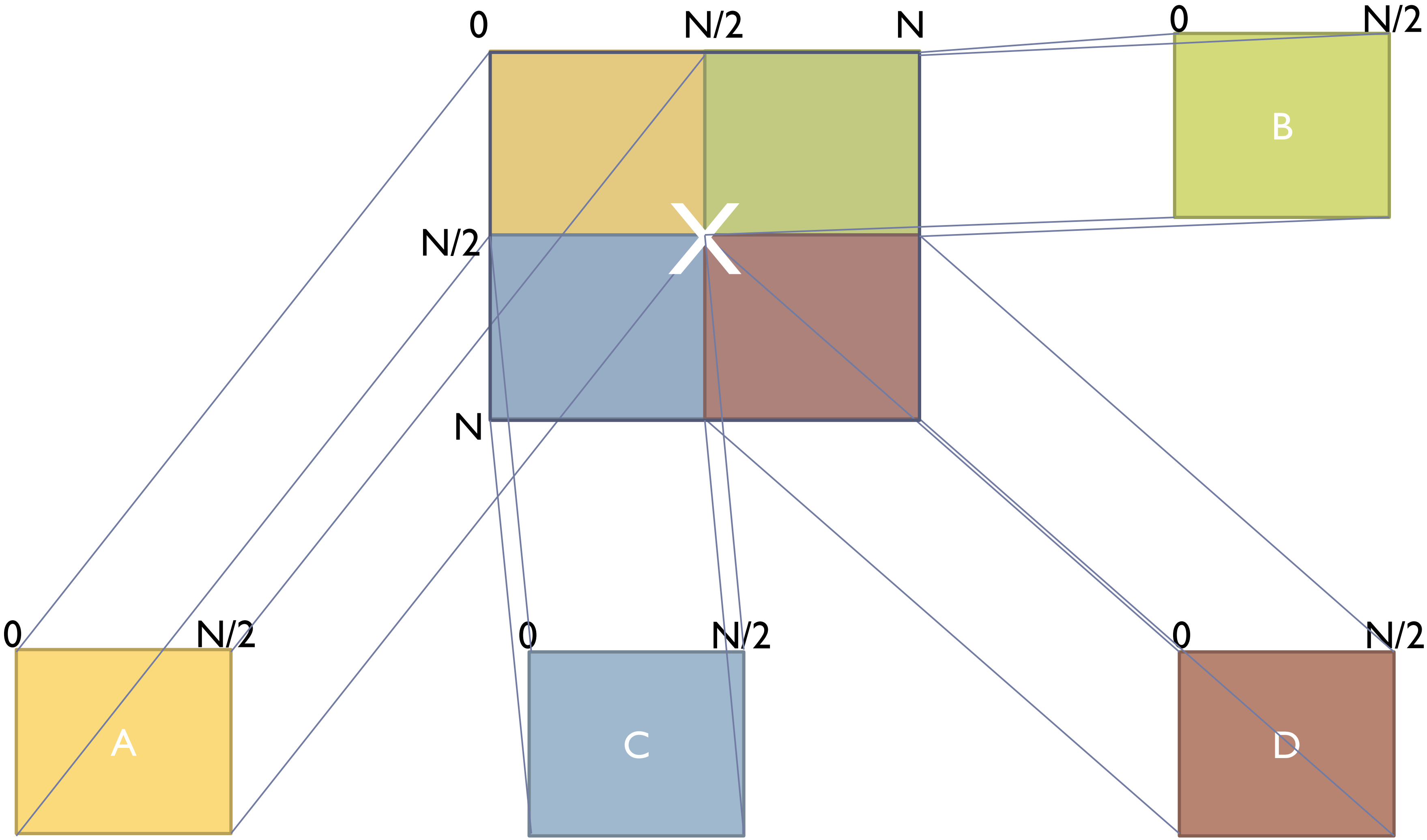
領域の分割

2次元配列 X での添字と
1次元配列A,B,C,Dでの添字の対応は…



領域の分割

要素の対応を意識して実装しないと、
Segmentation Fault の嵐に！(デバッグが大変)



領域の分割

バラバラに解いたら、解を合体