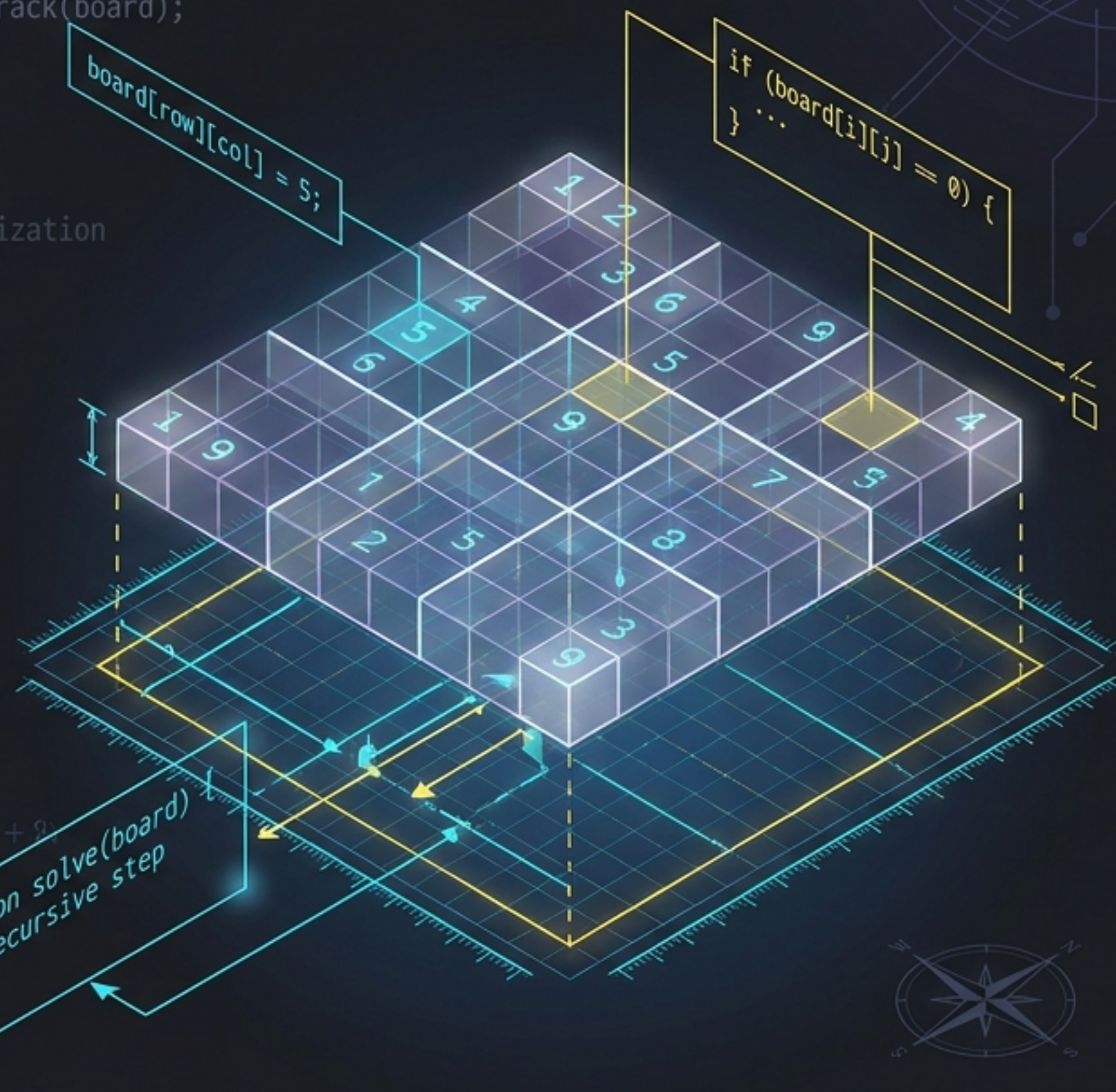


応用情報 令和4年秋
午後問3 プログラミング

設問1 攻略ガイド： アルゴリズムの 「設計図」を読み解く

```
function sudokuSolver(board) {  
  for (int i = 0; i < 9; i++) {  
    for (col = col, num) {  
      if (isValid(board, row, col, num)) {  
        return backtrack(board);  
      }  
    }  
  }  
}
```

// Algorithm initialization



```
function sudokuSolver(board) {  
  for (int i = 0; i < 9; i++) {  
    if (isValid(board, row, col, num)) {  
      return backtrack(board);  
    }  
  }  
}  
// Algorithm initialization
```

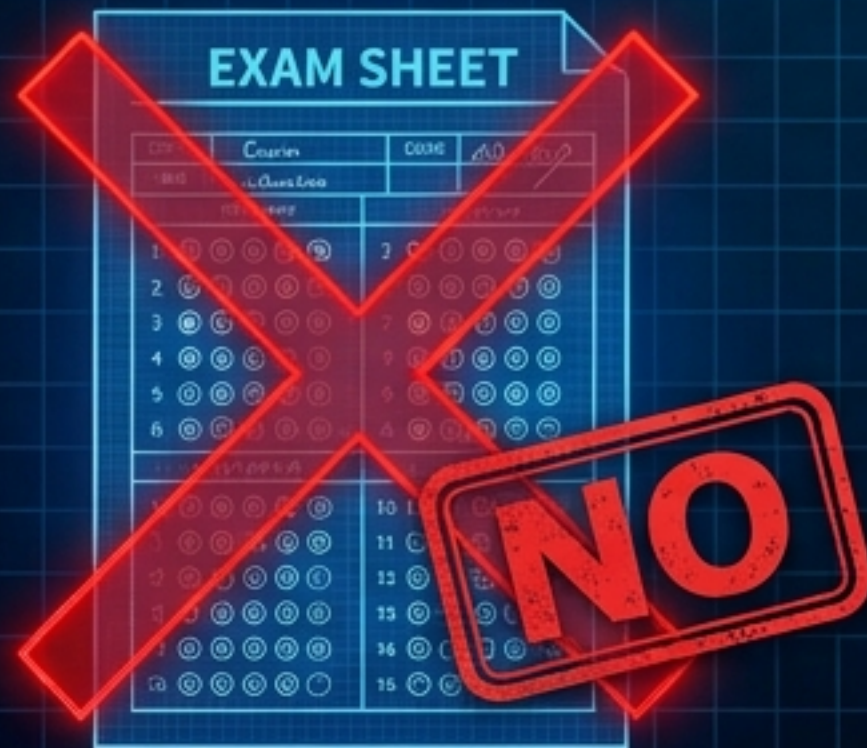
$$\arg a = \frac{(n+e)}{t+1} + k \cdot e^2 = 1$$

```
function solve(board) {  
  // recursive step  
}
```

$$n = \frac{(n-1) \pm \sqrt{(n^2 - 3n)}}{2 + 1}$$

本資料のミッション

「答え」ではなく、「答えを導き出すための視点」をインストールする。



単なる解答の丸暗記



自力で解くための分析力



1. データ構造の解読

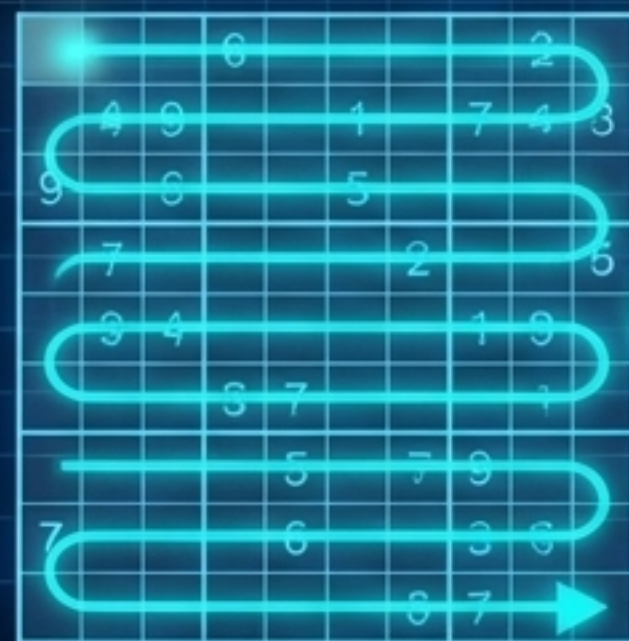


2. コメントの翻訳

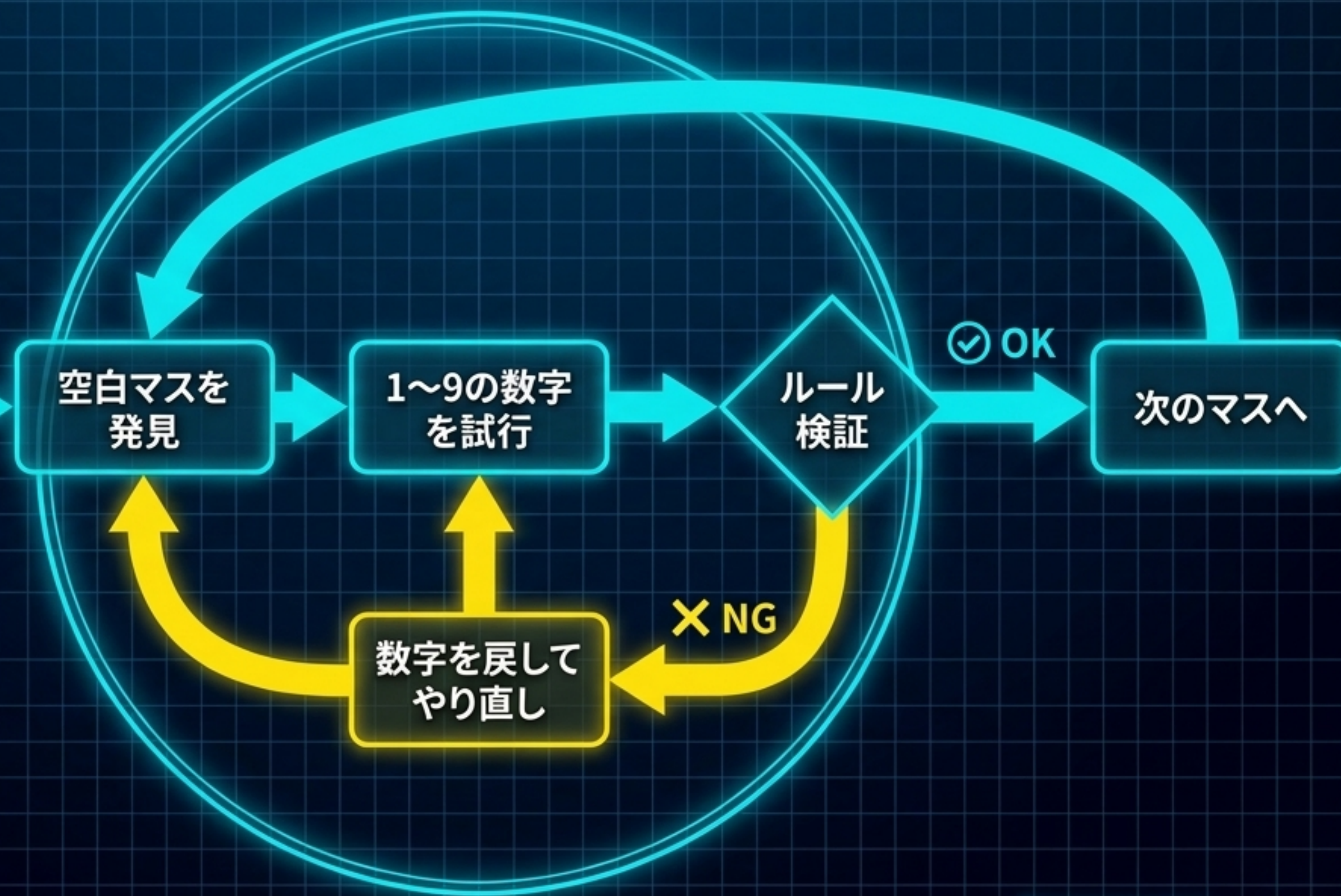


3. アーキテクチャの俯瞰

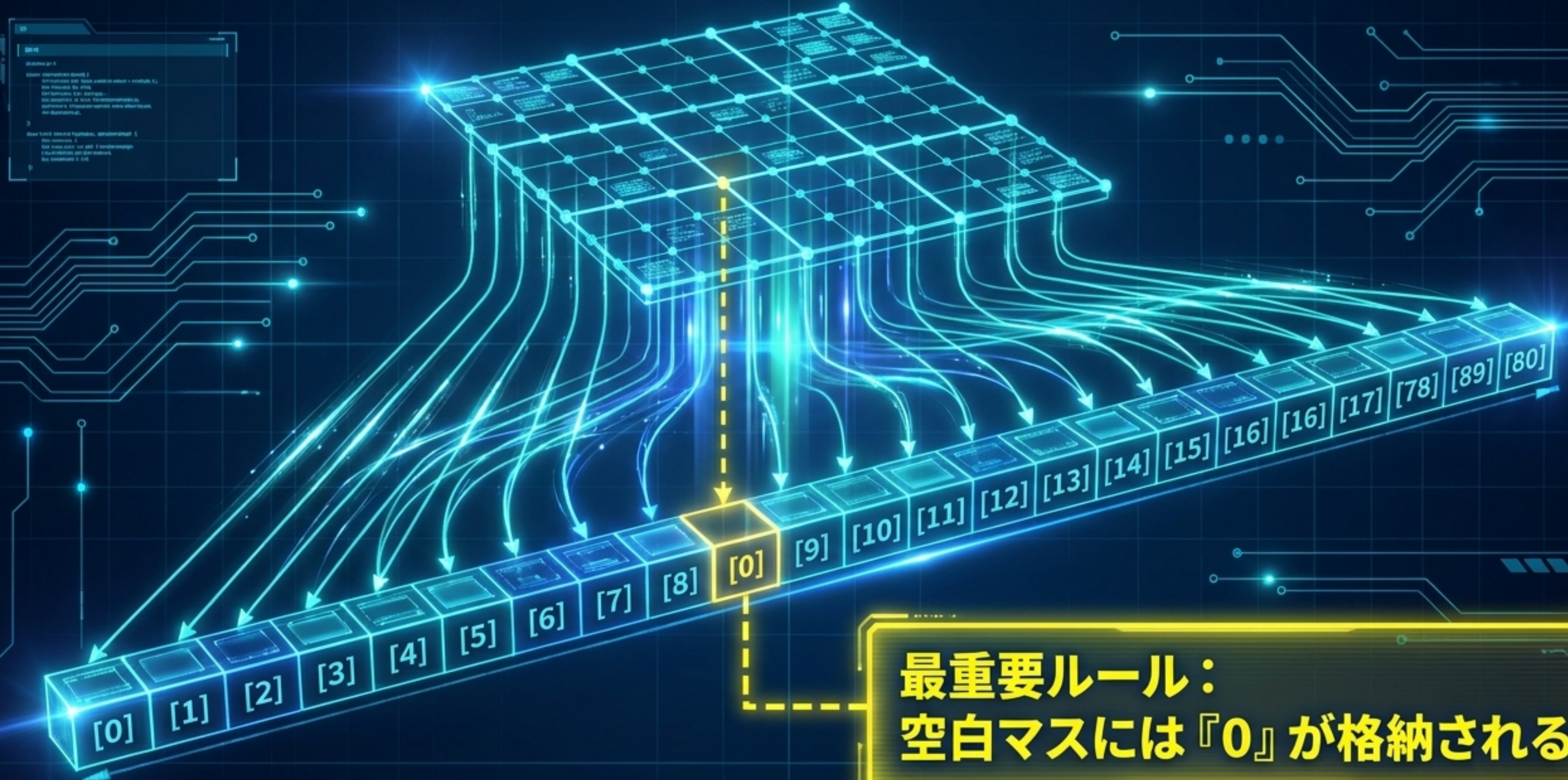
ターゲットの解剖：数独ソルバーの基本方針



		6				2		
4	9		1		7	4		3
9		6		5				
7			2				5	
3	4				1	9		
		5	7				1	
		5		7	9			
7		6			3	6		
				6	7			



データ構造の視覚化：盤面はいかにして記憶されるか



関数 solve(x) のブループリント

```
function solve(x)
```

```
  if (x が MAX_BOARD-1 より大きい)
```

```
    print_board()
```

```
    exit()
```

```
  else
```

```
    if ( [ア] )
```

```
      solve ( [イ] )
```

```
    else
```

```
      for (n を 1 から 9 まで 1 ずつ増やす) //1~9 の数字を順にマスに入れる
```

```
        if ( [ウ] )
```

```
          board[x] <- n
```

```
          solve ( [イ] )
```

```
          board[x] <- [エ]
```

```
        endif
```

```
      endfor
```

```
    endif
```

```
  endif
```

```
endfunction
```

//解答を出力する

//メインプログラムの処理を終了する

終了判定

ゴール到達

//対象のマスが空白でない場合

//次の探索

前進・スキップ

既に数字がある
マスを飛ばす

//次の探索

//再帰から戻った場合のマスの初期化

探索と後退 (DFSコア)

試行 → 検証 →
バックトラック

空欄[ア]へのアプローチ：スキップの条件

🔍 手がり

if ([ア])

//対象のマスが空白でない場合

🧠 推理

コメント：
「対象のマスが空白でない」

絶対ルール：
「空白マス = 0」

「対象マスの値 board[x] が
0 ではない」を、
コードでどう表現する？

空欄 [イ] へのアプローチ：次へのステップ

```
solve ( [イ] ) //次の探索
```



現在のマスが『x』のとき、
右隣のマスへ進むための計算式は？

空欄 [ウ] へのアプローチ：妥当性の検証

```
if ( [ウ] )
```

数字 n を入れる前のルール確認

タテ・ヨコ・枠のすべてのルール違反を
『一撃で』判定できる究極の関数はどれか？
その関数を呼び出すために必要な引数は？

Toolbox

row_ok(n, x)

横1行の重複チェック

column_ok(n, x)

縦1列の重複チェック

frame_ok(n, x)

3x3枠内の重複チェック

check_ok(n, x)

全ての重複チェックを実行

空欄 [エ] へのアプローチ：バックトラックスの真髄



```
board[x] ← [エ] //再帰から戻った場合のマス初期化
```

行き止まりから戻ってきたということは、試した数字『 n 』は失敗だったということ。マスを未決定の『真っ白な状態』にリセットするには、何を代入すべきか？ (Slide 4のルールを参照)

アルゴリズムの役割マトリクス

<u>空欄</u>	<u>コード上のアクション</u>	<u>深さ優先探索 (DFS) における役割</u>
[ア]	条件判定 (<code>board[x] != 0</code>)	探索のスキップ (Skip) 
[イ]	引数のインクリメント (<code>x + 1</code>)	次の階層への前進 (Dive) 
[ウ]	制約関数の呼び出し (<code>check_ok</code>)	無効な分岐の刈り込み (Prune) 
[エ]	初期値の代入 (<code>0</code>)	状態の巻き戻し・復元 (Restore) 

攻略の鉄則：未知のコードを読むための3つの視点

01

データ構造の
定義を
絶対視せよ

「空白マス=0」のような
前提ルールこそが、
アルゴリズムの挙動を
決定づける。

02

コメントは
アルゴリズムの
直訳である

コード右側のコメントは
単なるメモではない。
実装すべきロジックの
「答え」そのものである。

03

再帰は
「戻る時」の処
理に魂が宿る

深さ優先探索 (DFS) で
は、再帰呼び出しの直
後に必ず「状態の復元
(バックトラック)」が
潜んでいる。