知能システム学
第5回　問題分割法

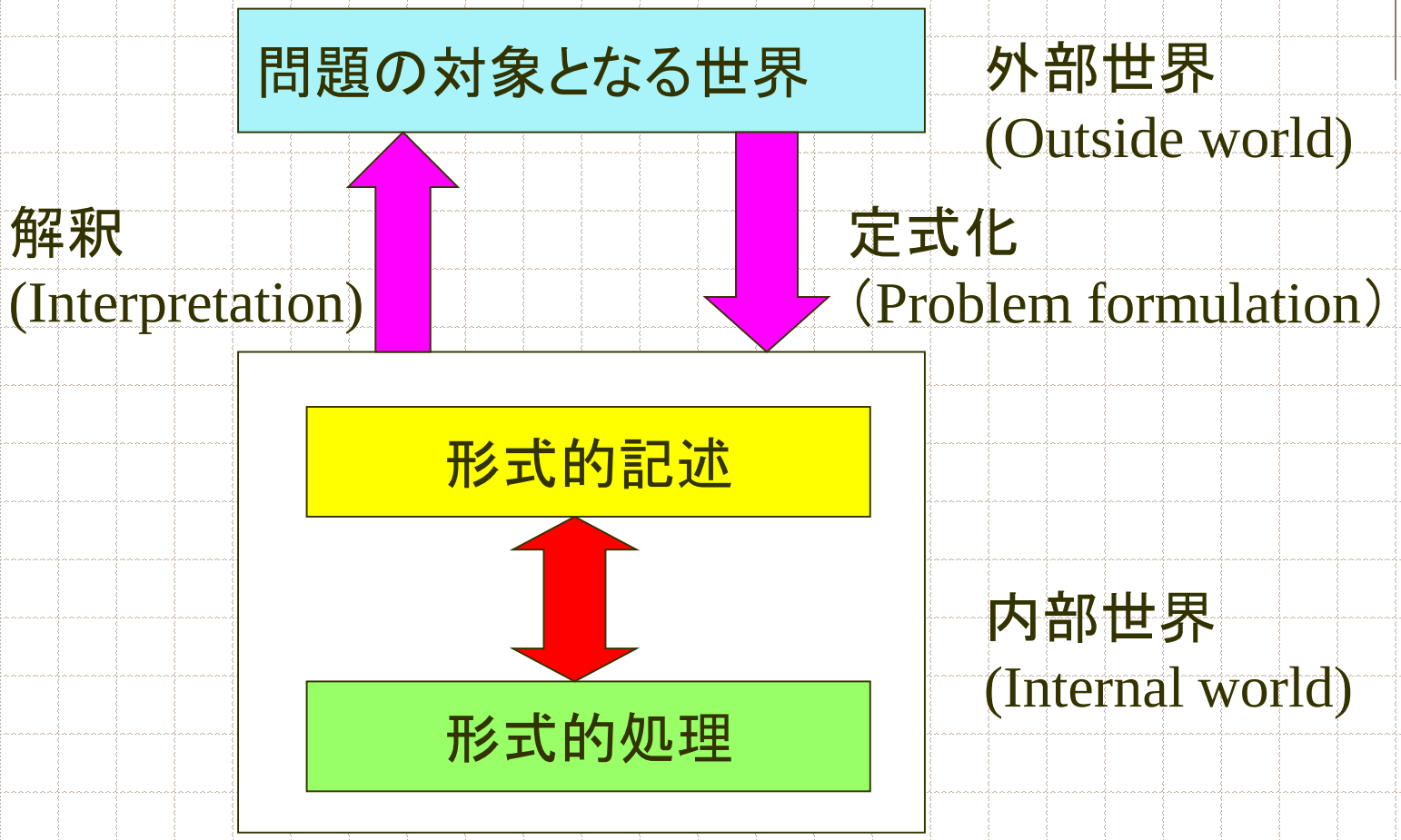
ソフトウェア情報学部
David Ramamonjisoa

目次

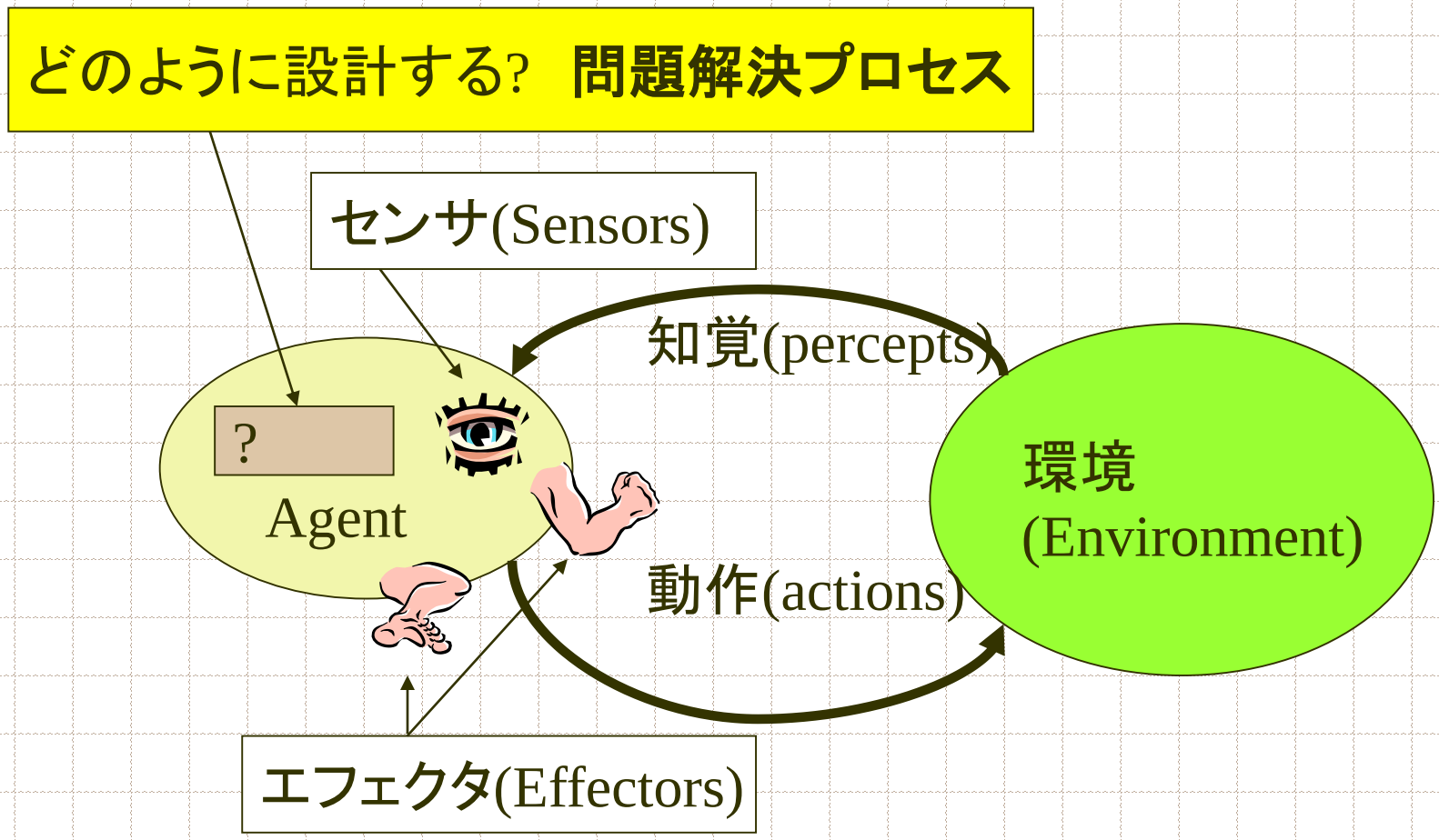
- 前回の内容：問題解決の状態空間法
- 問題分割法：問題を定式化する
- 例題

- まとめ

問題解決プロセス（The Problem Solving Process）



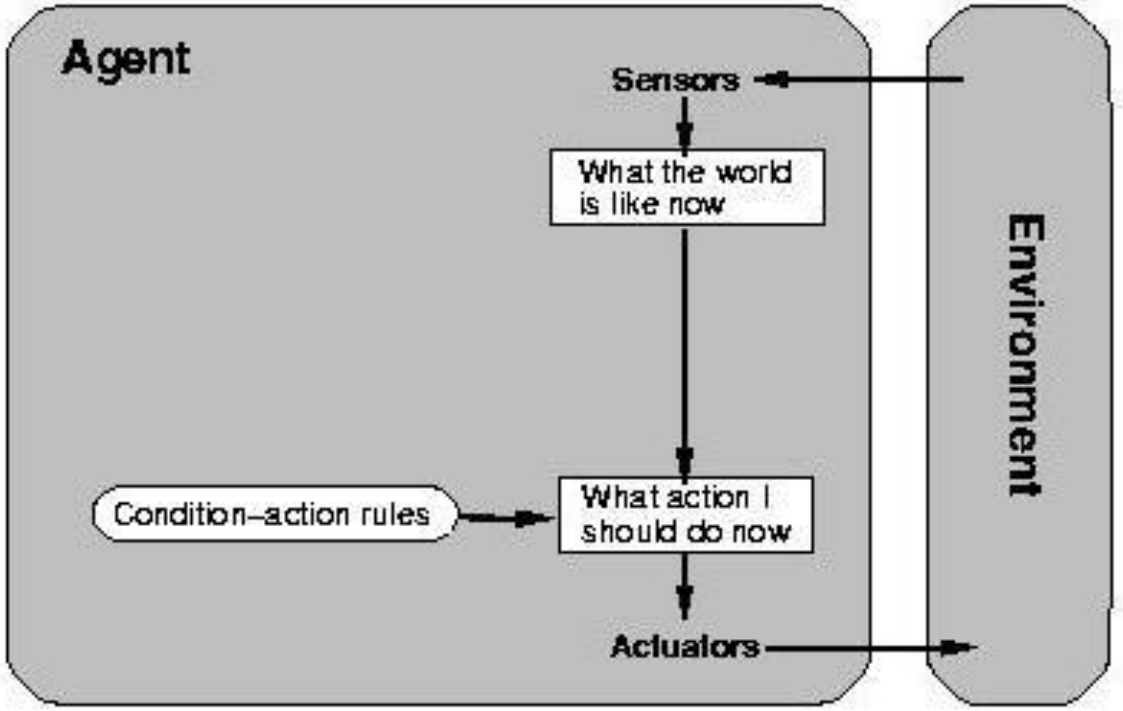
エージェント（agent）



問題解決エージェント(Problem Solving Agents (1))

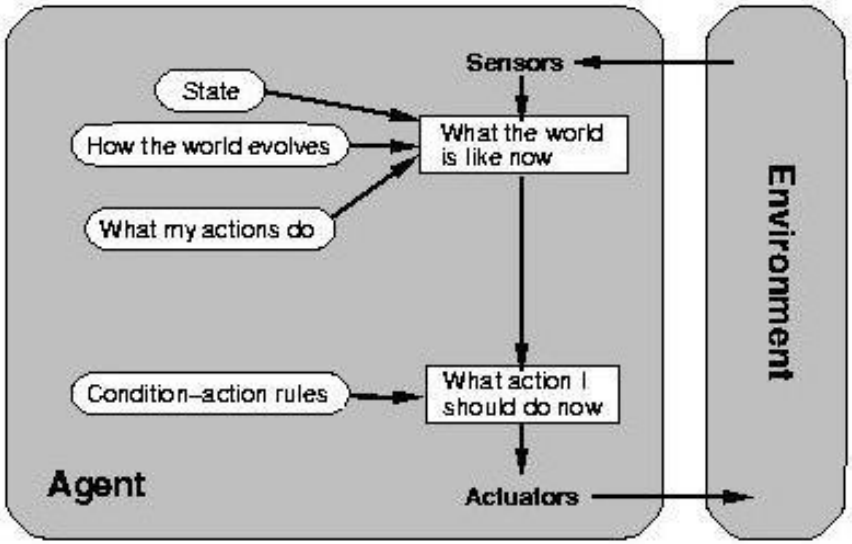
- 自動化する
- ゴールを達成する

反射エージェント

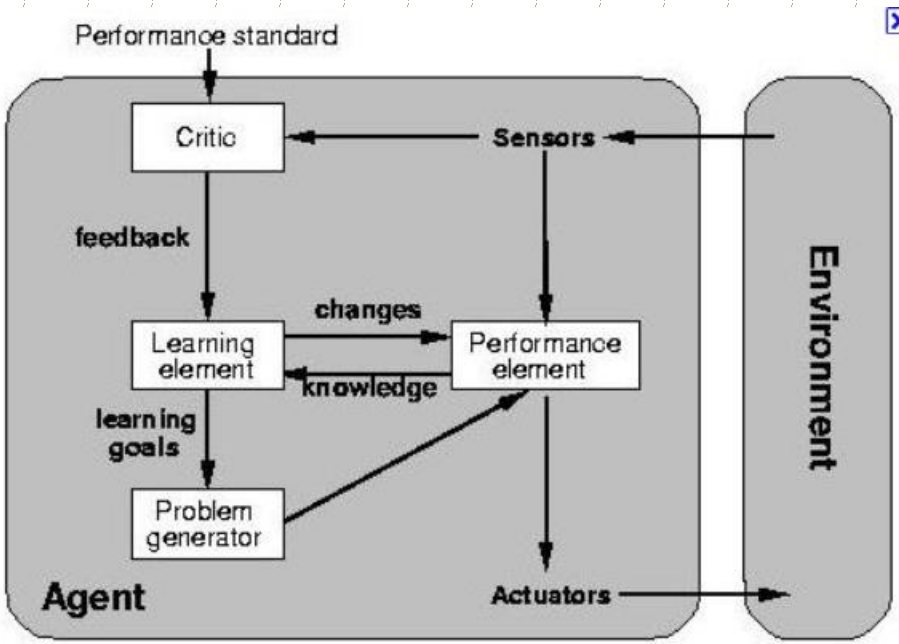


問題解決エージェントの内部構成

- 知的エージェント
  - 知的反射エージェント



- 知的エージェント
  - 学習エージェント

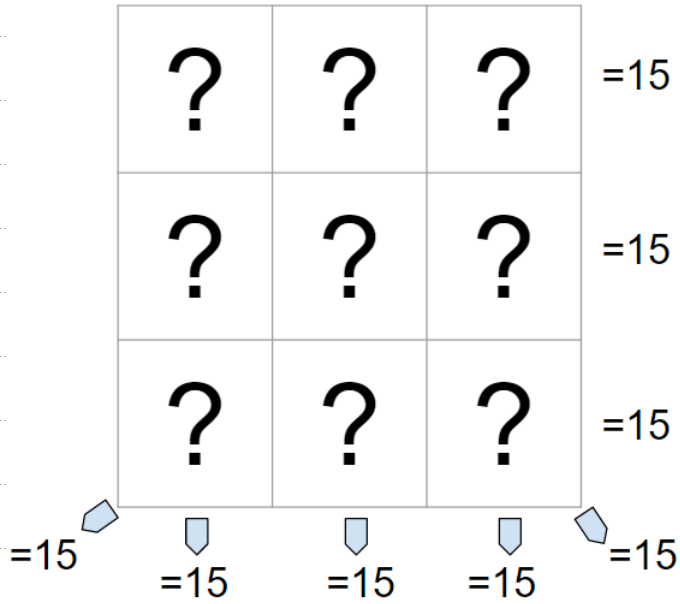


全数探索(exhaustive search)

- 問題： 3x3の表を1から9の9つの数字で理めよ。ただし、各行、各列、対角線上のマスの和は等しくなければならない。

問題:魔方陣

?	?	?
?	?	?
?	?	?



定式化：

- 初期状態：空表
- ゴール状態：すべての数の各行、各列、対角線の和を等しくなる
- 行為：一つの数をマスに置く
- ゴール検査：条件を満たしているかどうか
- コスト：1行為は1コスト
- この問題は9!=362,880通りの状態空間になる
- 全ての配置を生成し、それぞれについてゴール検査し、探索を行う。

戦略：ヒューリスティック

- 定和(magic sum)を調べる
  (1+2+3+…+9)/3=15
- 中央のマスにはどのような数が入らなければならない.
- $a, b, c; d, e, f; g, h, i$;第1行、第2行、第3行に入る数を置くと、$e = 5$を求められる.
- 4つの組(1,9),(2,8),(3,7),(4,6)をその周りに配置することで解決できる。
- 組(1,9)は必ず5と同じ行または同じ列に置く

魔方陣(magic square)

a	b	c
d	e	f
g	h	i

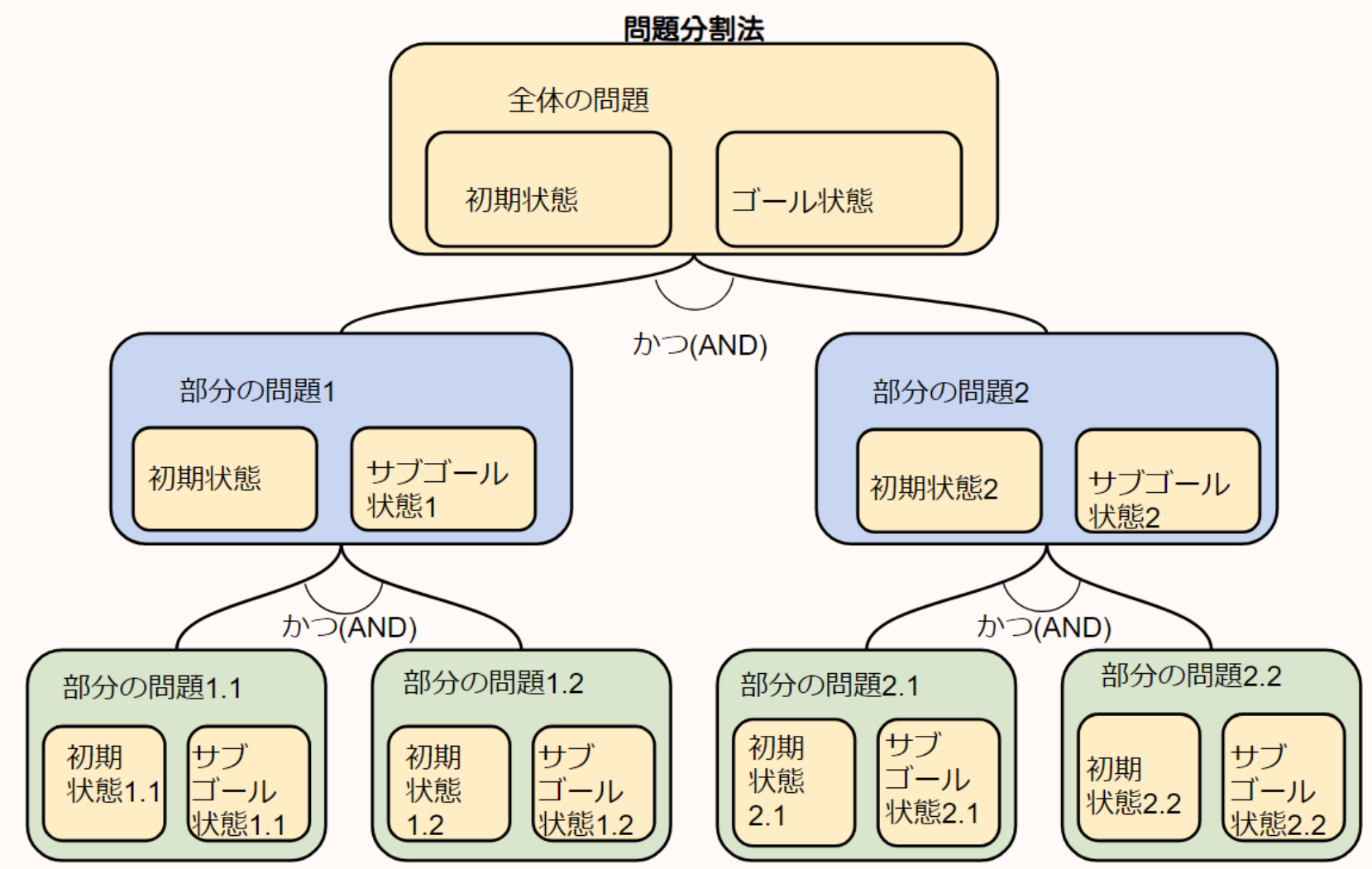
6	1	8	=15
7	5	3	=15
2	9	4	=15
=15	=15	=15	=15

=15

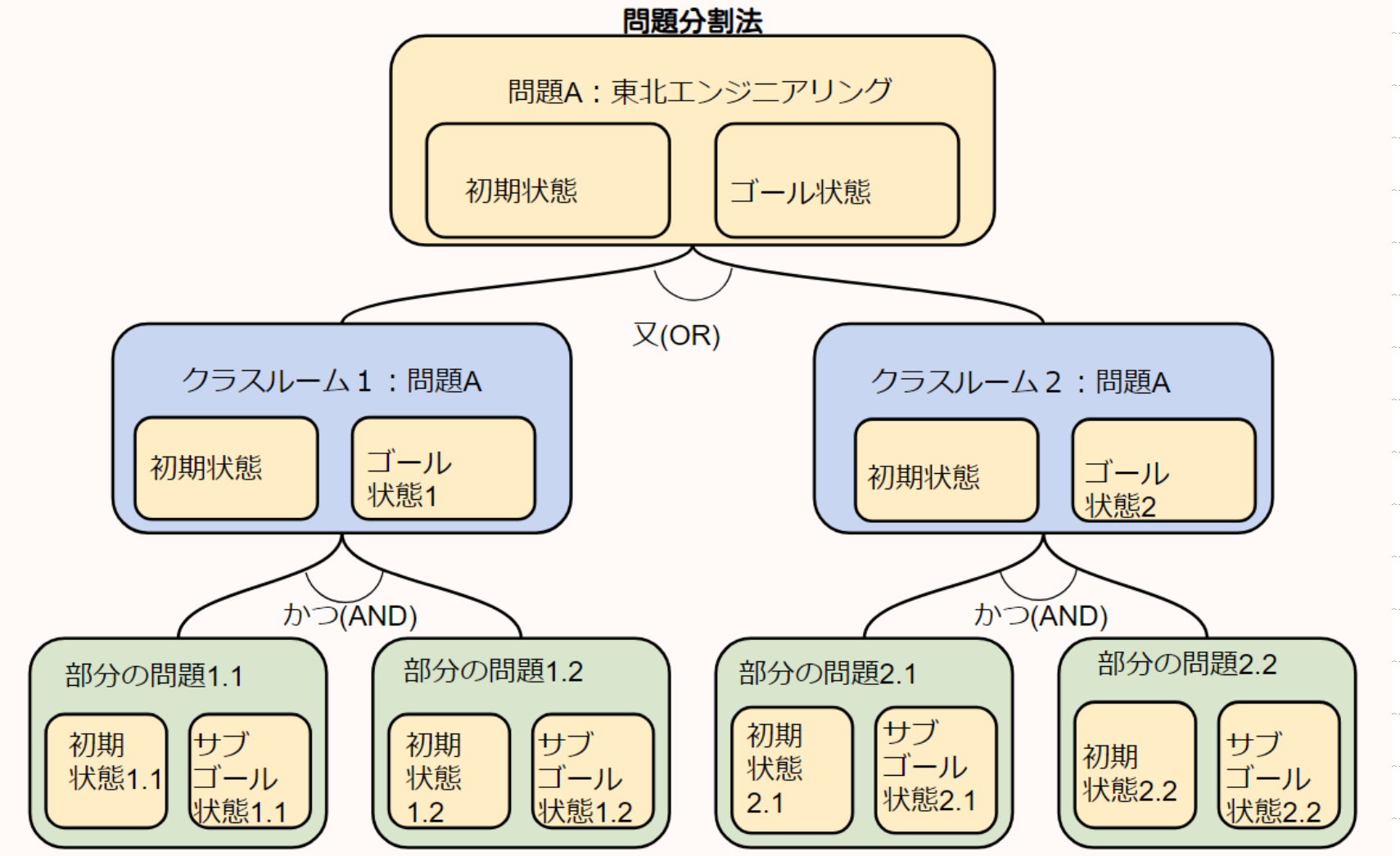
問題分割法

- 複雑な問題：別の問題に置き換える
- いくつかの単純な部分問題に分割して解く
- 形式的にAND/OR木で表現する
- ある節点がAND節点を持つときは、そのすべてのAND節点が解決されていれば、その節点は解決されている
- ある節点がOR節点を持つときは、OR節点の1つが解決されていれば、その節点は解決されている
- 終端節点は解決されている節点である。

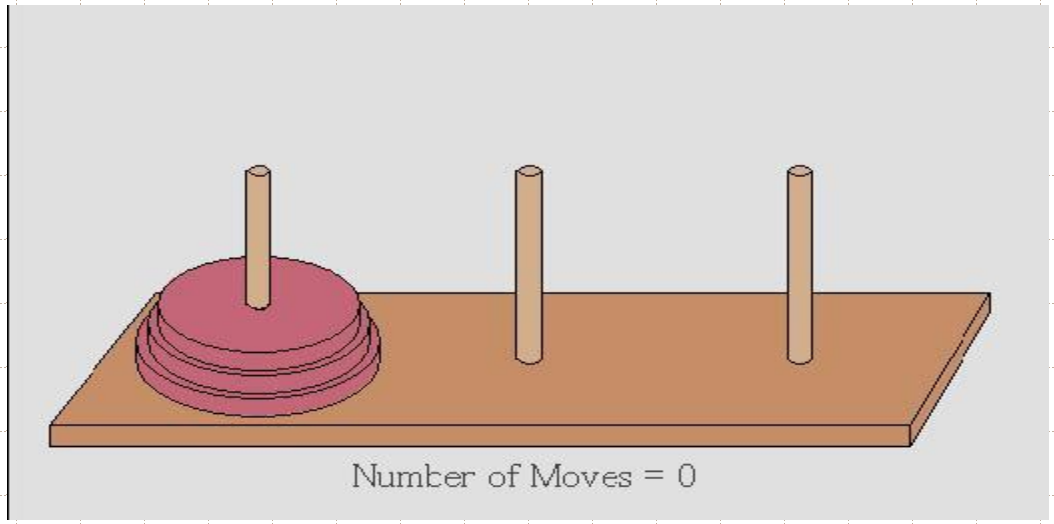
問題分割法：AND木


問題分割法
全体の問題
初期状態
ゴール状態
かつ(AND)
部分の問題1
初期状態
サブゴール状態1
部分の問題2
初期状態2
サブゴール状態2
かつ(AND)
かつ(AND)
部分の問題1.1
初期状態1.1
サブゴール状態1.1
部分の問題1.2
初期状態1.2
サブゴール状態1.2
部分の問題2.1
初期状態2.1
サブゴール状態2.1
部分の問題2.2
初期状態2.2
サブゴール状態2.2

問題分割法：OR木

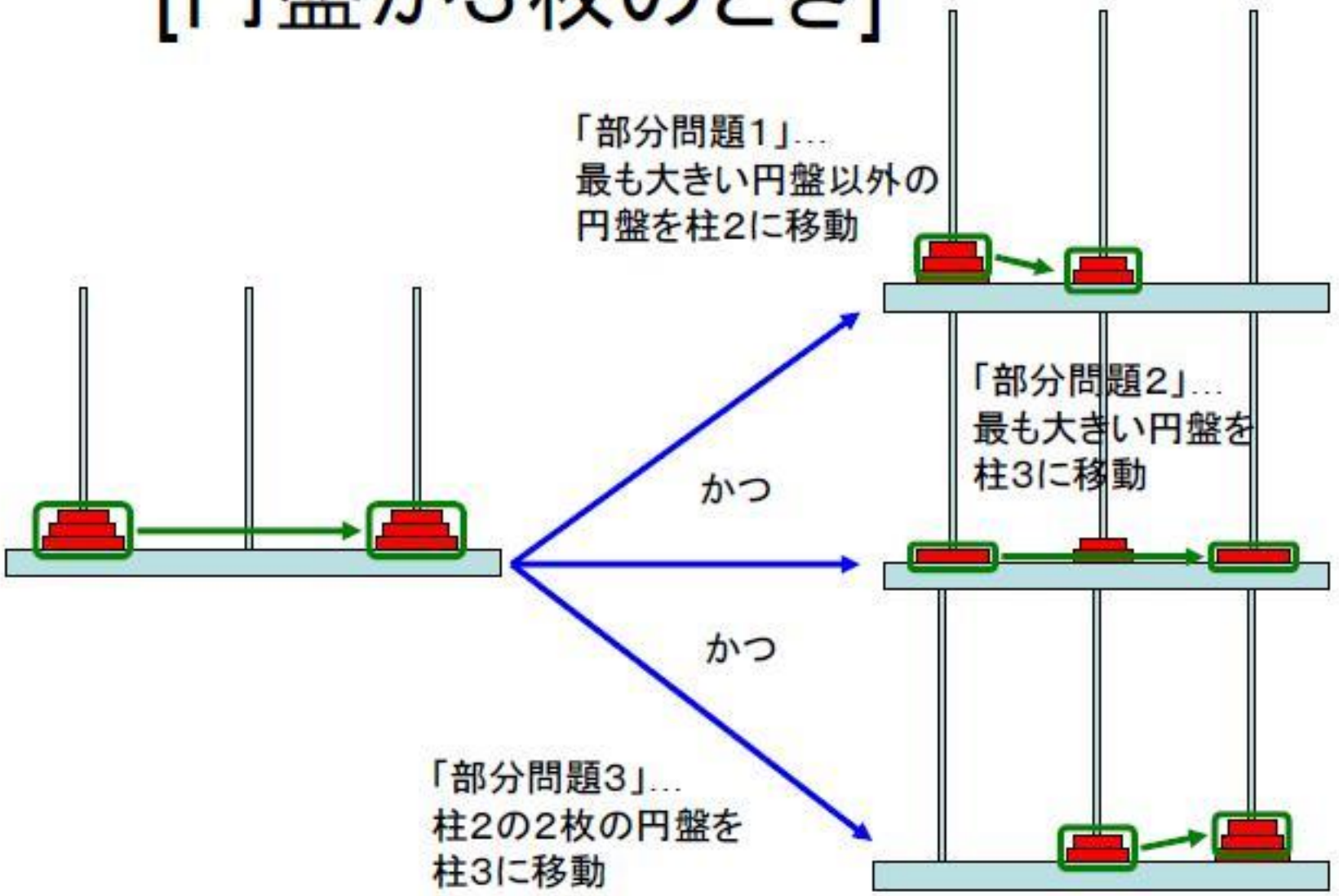

問題分割法
問題A：東北エンジニアリング
初期状態
ゴール状態
又(OR)
クラスルーム１：問題A
初期状態
ゴール
状態1
クラスルーム２：問題A
初期状態
ゴール
状態2
かつ(AND)
かつ(AND)
部分の問題1.1
初期
状態1.1
サブ
ゴール
状態1.1
部分の問題1.2
初期
状態
1.2
サブ
ゴール
状態1.2
部分の問題2.1
初期
状態
2.1
サブ
ゴール
状態2.1
部分の問題2.2
初期
状態2.2
サブ
ゴール
状態2.2

問題分割法：ハノイの塔

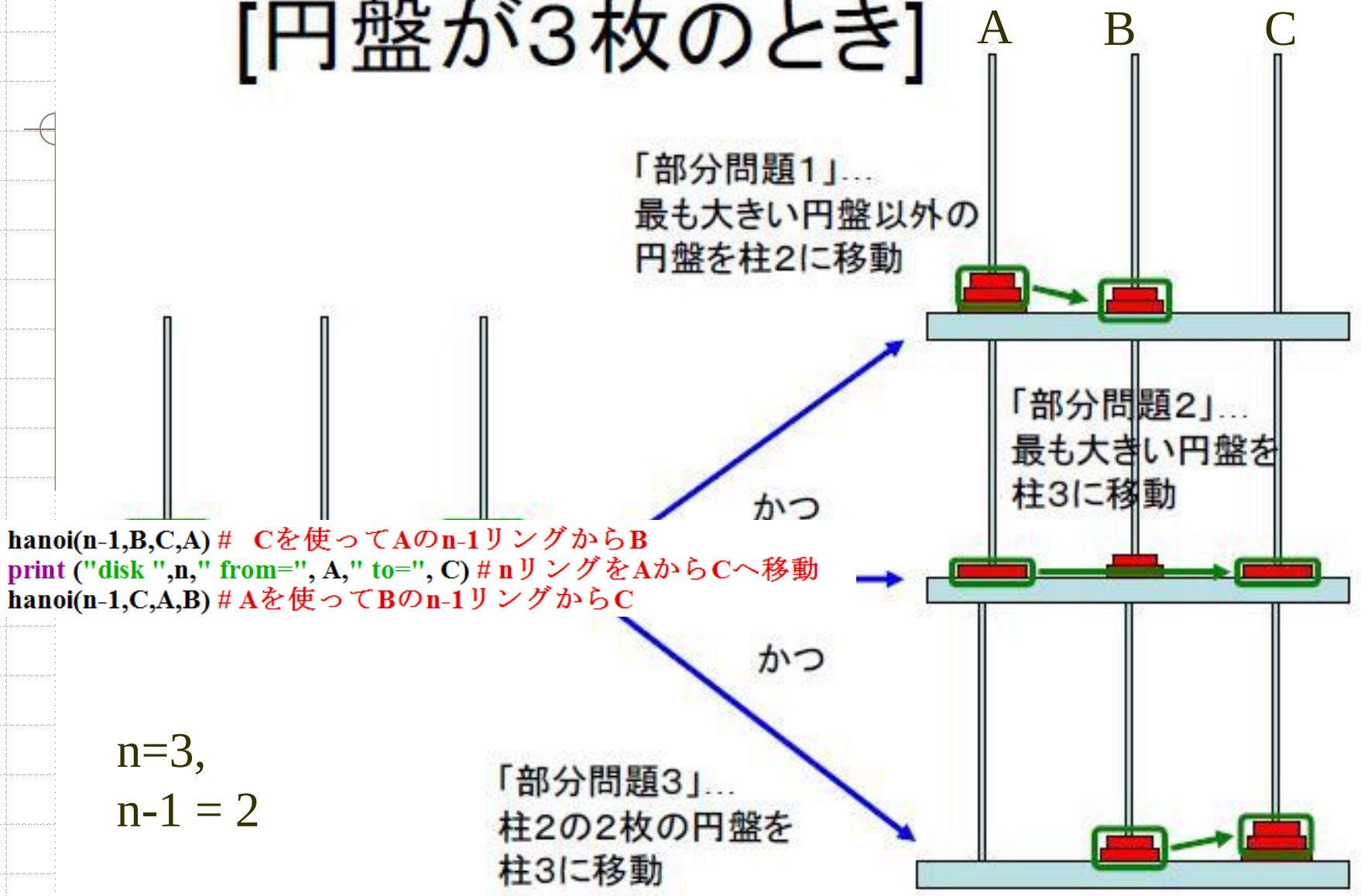


- 初期状態は上図の形
- 1回につき、1枚の円盤を移動する
- 小さい円盤の上に大きい円盤は重ねられない
- 目標状態は、すべての円盤を柱3に下から大きい順に重ねる

[円盤が3枚のとき]


「部分問題1」…
最も大きい円盤以外の
円盤を柱2に移動
「部分問題2」…
最も大きい円盤を
柱3に移動
かつ
かつ
「部分問題3」…
柱2の2枚の円盤を
柱3に移動

[円盤が3枚のとき]

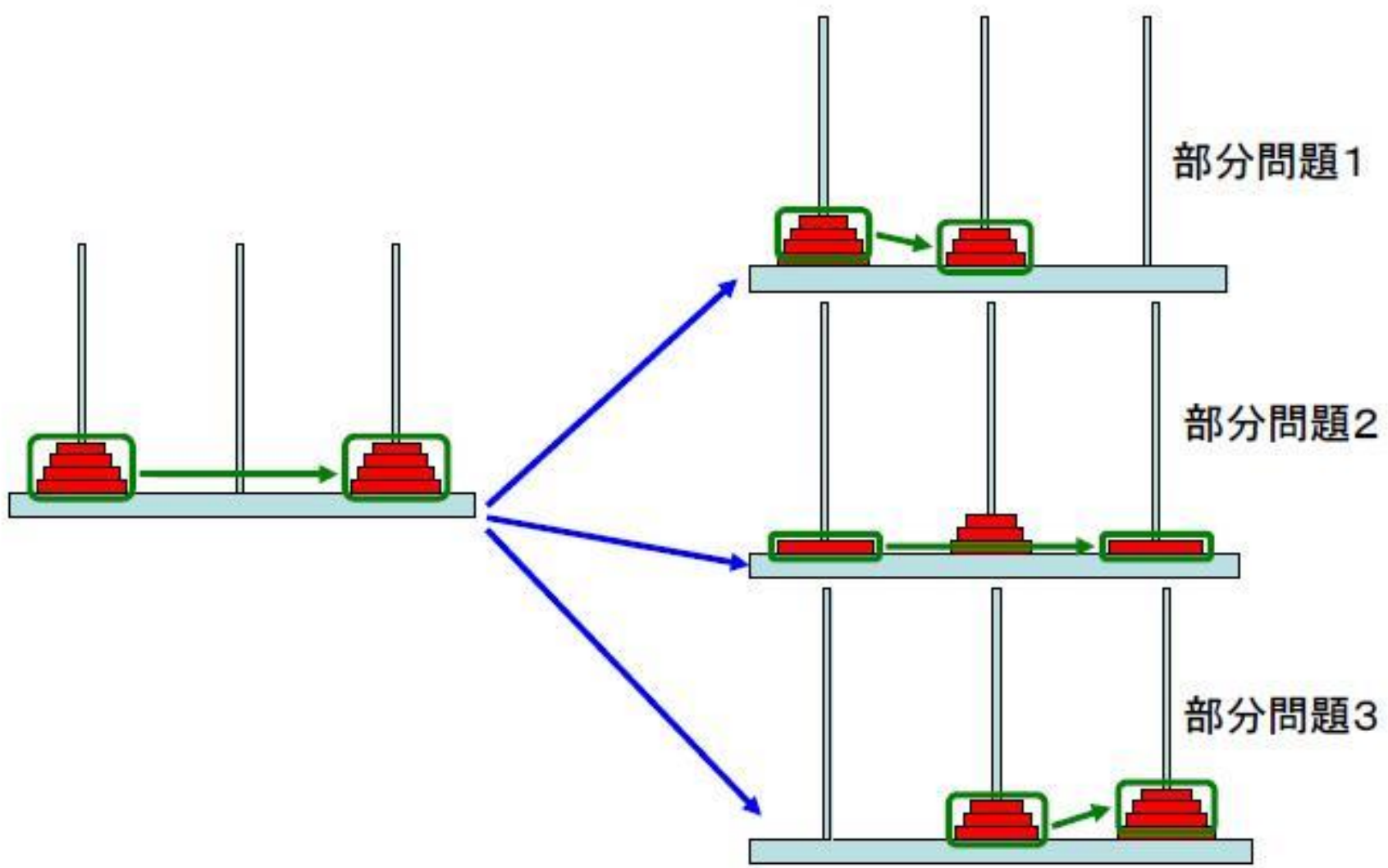



```
hanoi(n-1,B,C,A) #  Cを使ってAのn-1リングからB
print ("disk ",n," from=", A," to=", C) # nリングをAからCへ移動
hanoi(n-1,C,A,B) # Aを使ってBのn-1リングからC
```

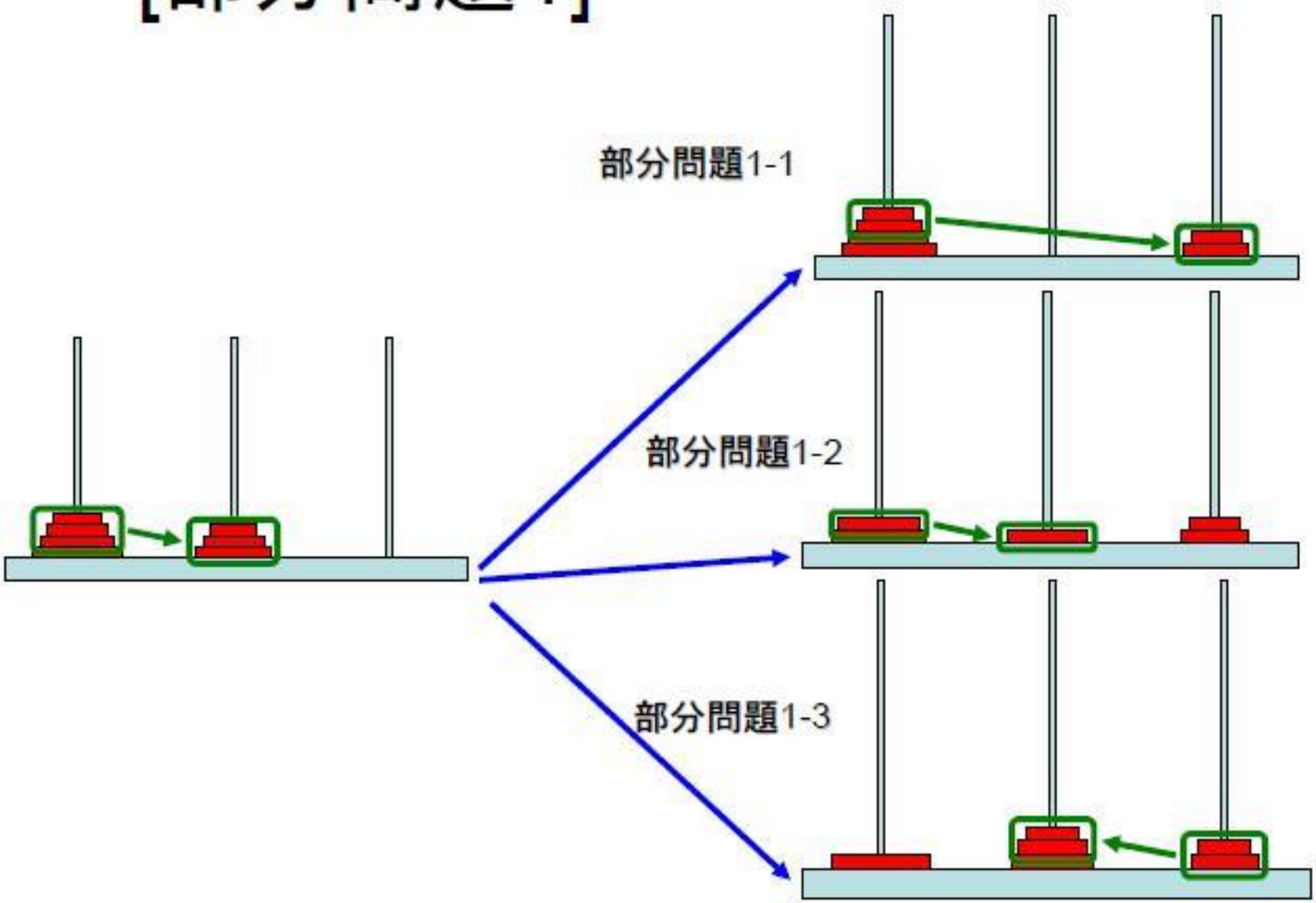
n=3,
n-1 = 2

```
# ハノイの塔：再帰呼出関数を用いて解く

def hanoi(n,C,B,A):
    if n> 0:
        hanoi(n-1,B,C,A) #  Cを使ってAのn-1リングからB
        print ("disk ",n," from=", A," to=", C) # nリングをAからCへ移動
        hanoi(n-1,C,A,B) # Aを使ってBのn-1リングからC
    elif n==0:
        return None

# hanoi(リングの数, 目的柱,仮柱,スタート柱)
>>> hanoi(3,"C","B","A")
disk  1  from= A  to= C
disk  2  from= A  to= B
disk  1  from= C  to= B
disk  3  from= A  to= C
disk  1  from= B  to= A
disk  2  from= B  to= C
disk  1  from= A  to= C
```

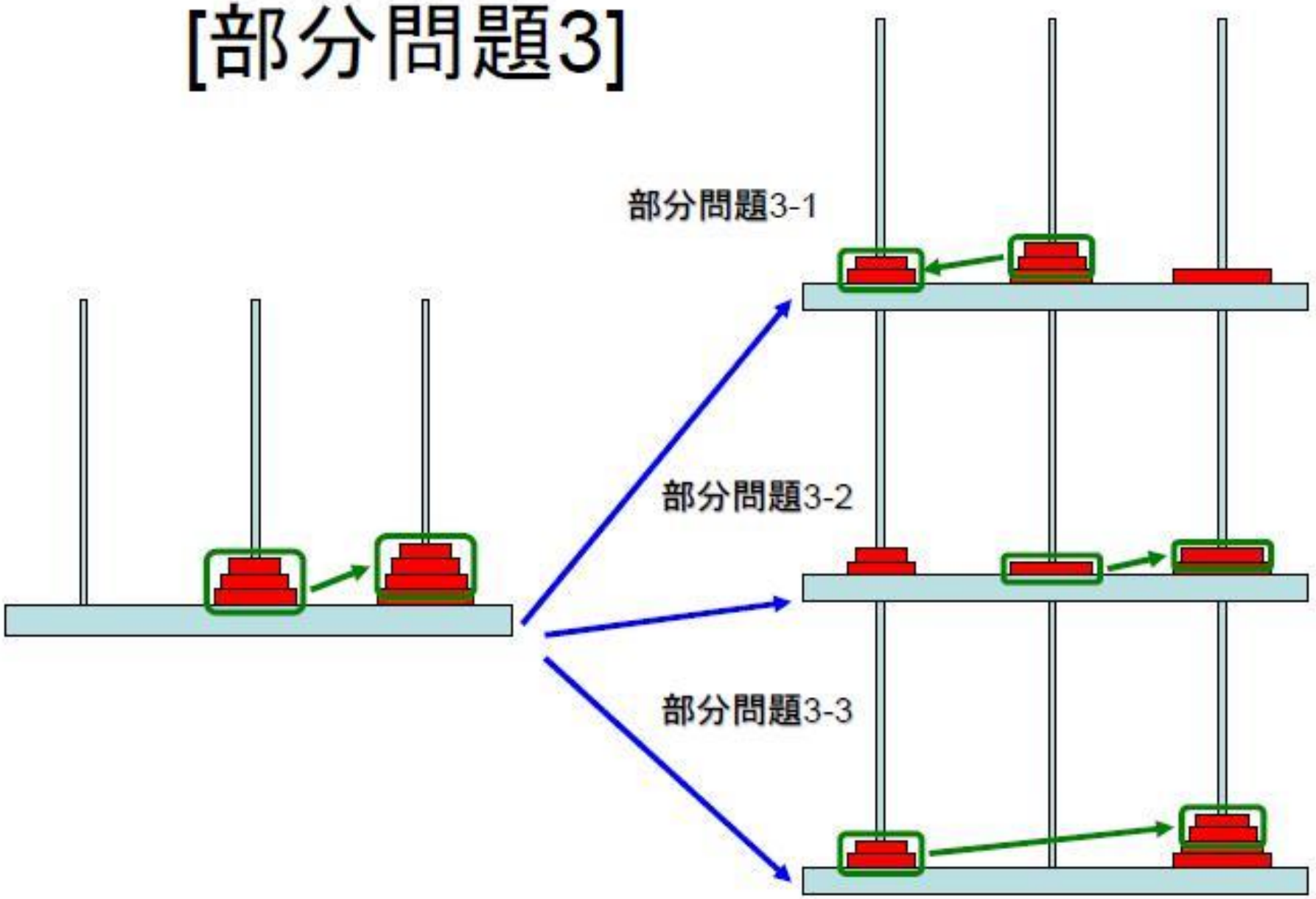
[円盤が4枚のとき]

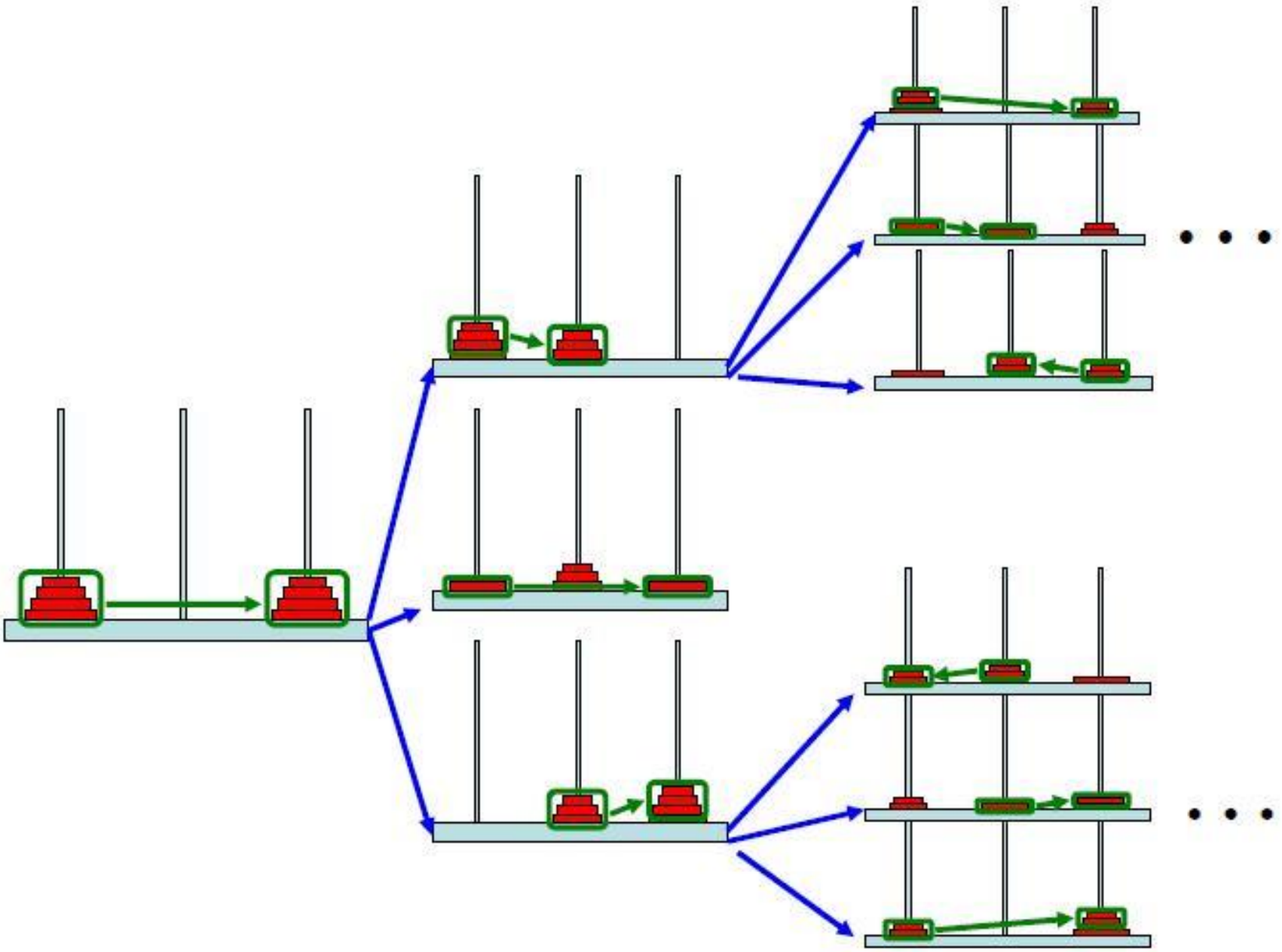


[部分問題1]

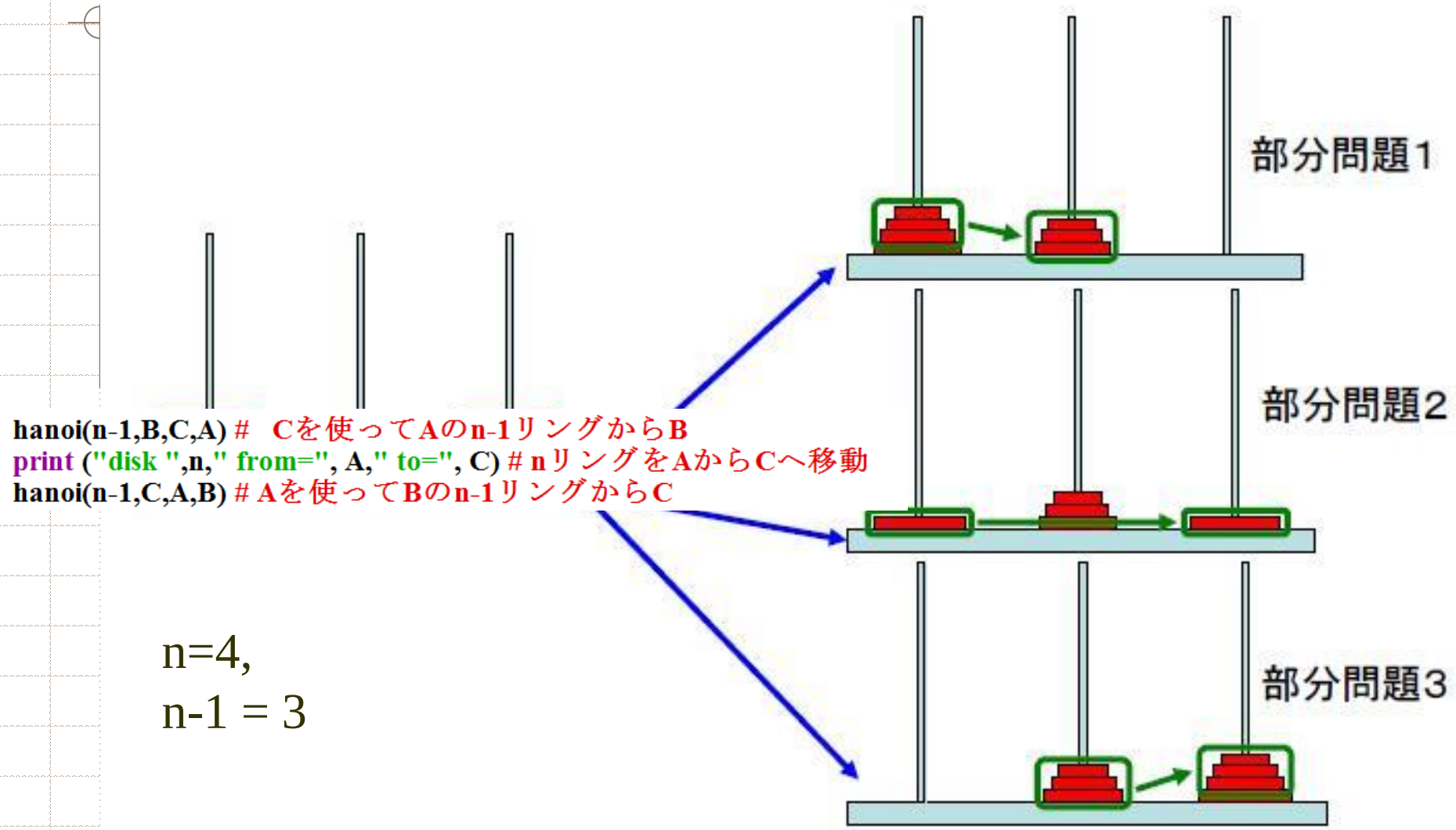


[部分問題3]





[円盤が4枚のとき]



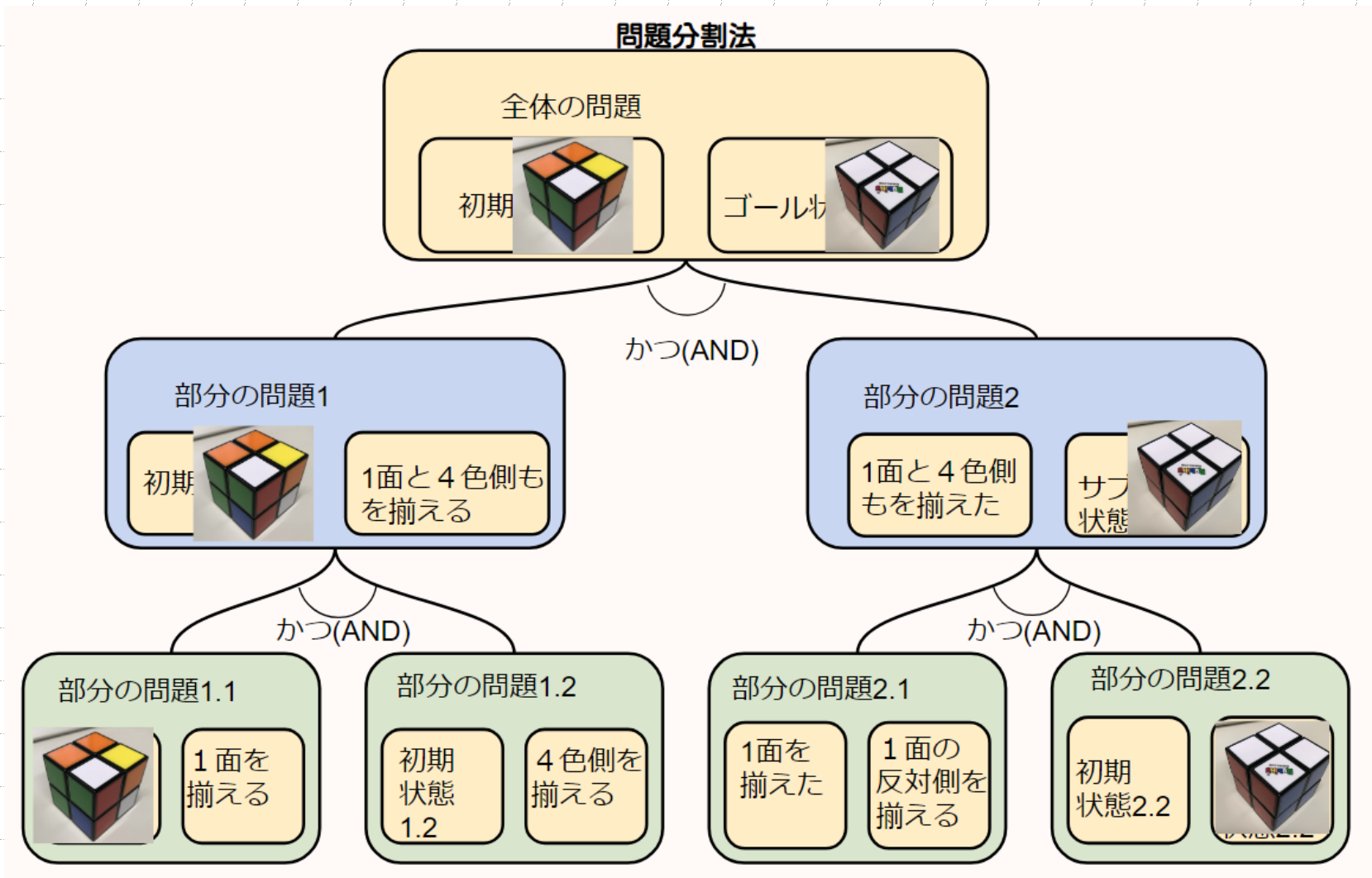

```
hanoi(n-1,B,C,A) #  Cを使ってAのn-1リングからB
print ("disk ",n," from=", A," to=", C) # nリングをAからCへ移動
hanoi(n-1,C,A,B) # Aを使ってBのn-1リングからC
```

n=4,

n-1 = 3

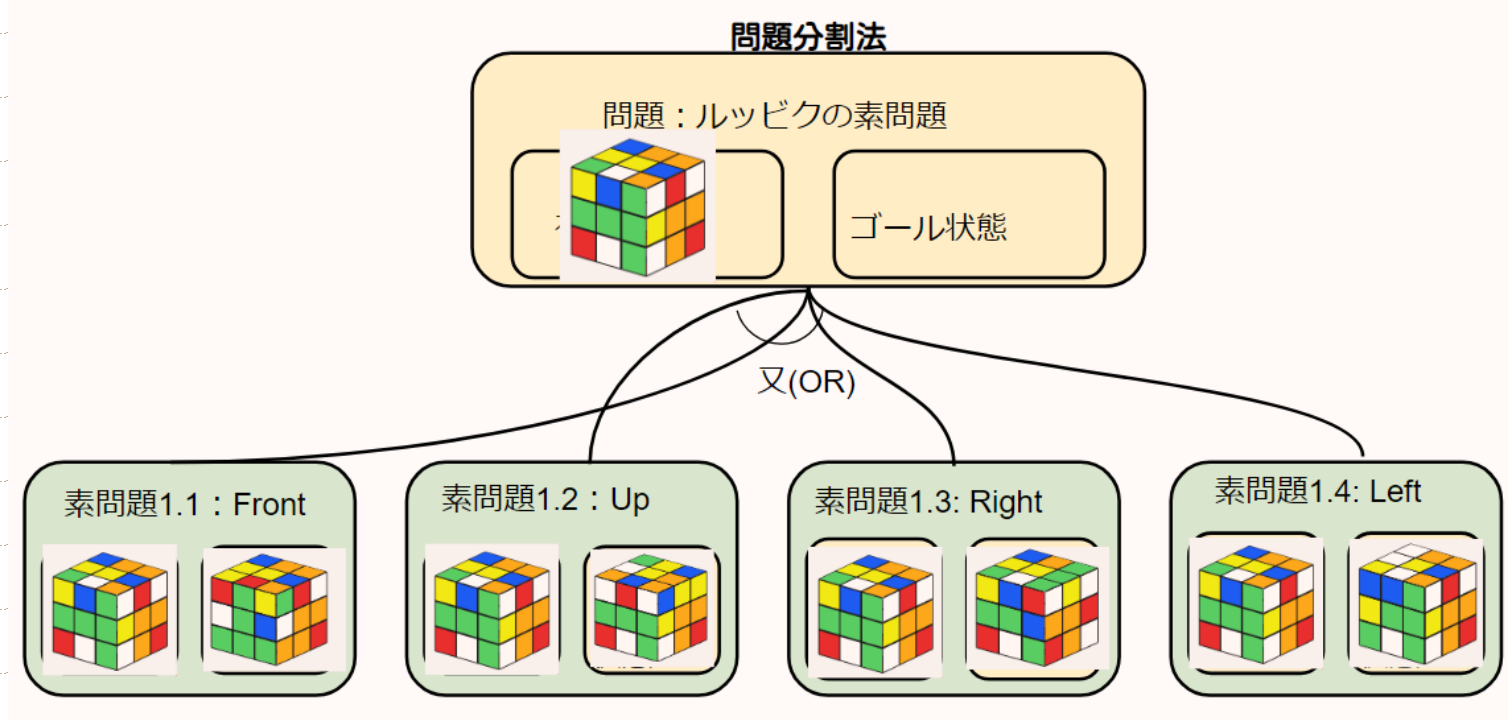
```
# ハノイの塔 : 再帰呼出関数を用いて解く

def hanoi(n,C,B,A):
    if n> 0:
        hanoi(n-1,B,C,A) #  Cを使ってAのn-1リングからB
        print ("disk ",n," from=", A," to=", C) # nリングをAからCへ移動
        hanoi(n-1,C,A,B) # Aを使ってBのn-1リングからC
    elif n==0:
        return None

# hanoi(リングの数、目的柱、仮柱、スタート柱)
>>> hanoi(4,"C","B","A")
disk  1  from= A  to= B
disk  2  from= A  to= C
disk  1  from= B  to= C
disk  3  from= A  to= B
disk  1  from= C  to= A
disk  2  from= C  to= B
disk  1  from= A  to= B
disk  4  from= A  to= C
disk  1  from= B  to= C
disk  2  from= B  to= A
disk  1  from= C  to= A
disk  3  from= B  to= C
disk  1  from= A  to= B
disk  2  from= A  to= C
disk  1  from= B  to= C
```

ルッビク2x2問題分割法：AND木



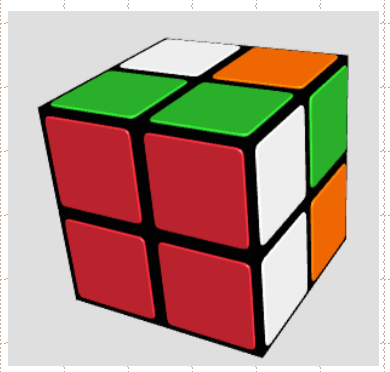
ルッビクの素問題

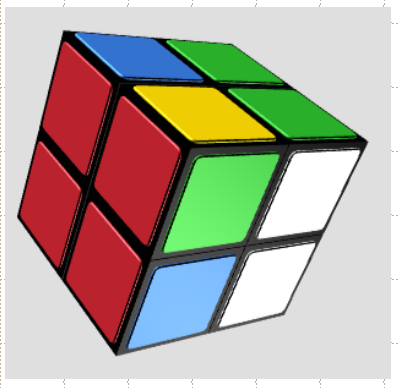
- 分割できない
- 状態Aから状態Bまでは1つのアクションしかない



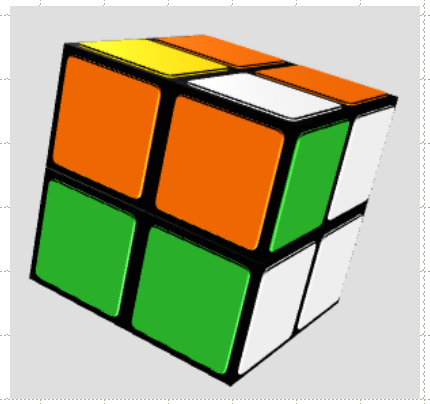
ルッビク2x2の問題分割法

- 分割法:
  - １面の１色のベースレイヤ問題
    - 外側面の4色問題

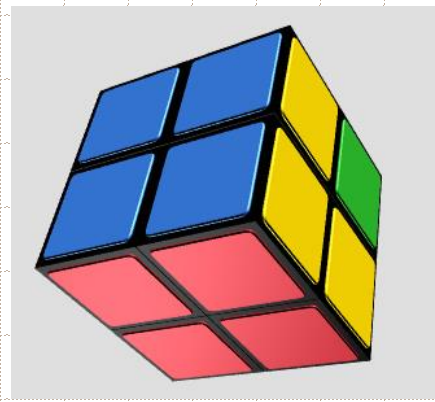




  - １面のトップレイヤ問題
    - コーナーを揃える
    - コーナーをフリップする



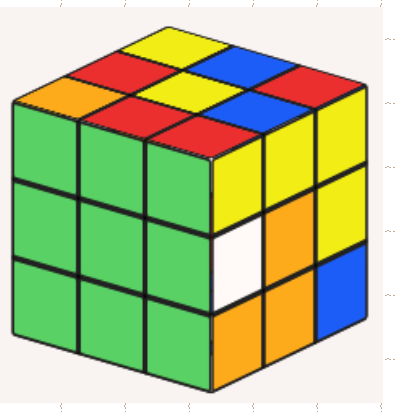
対面



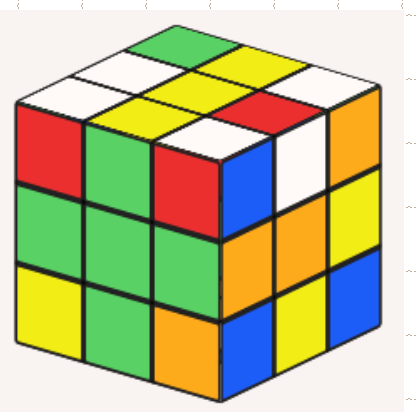
裏面

ルッビク3x3の問題分割法

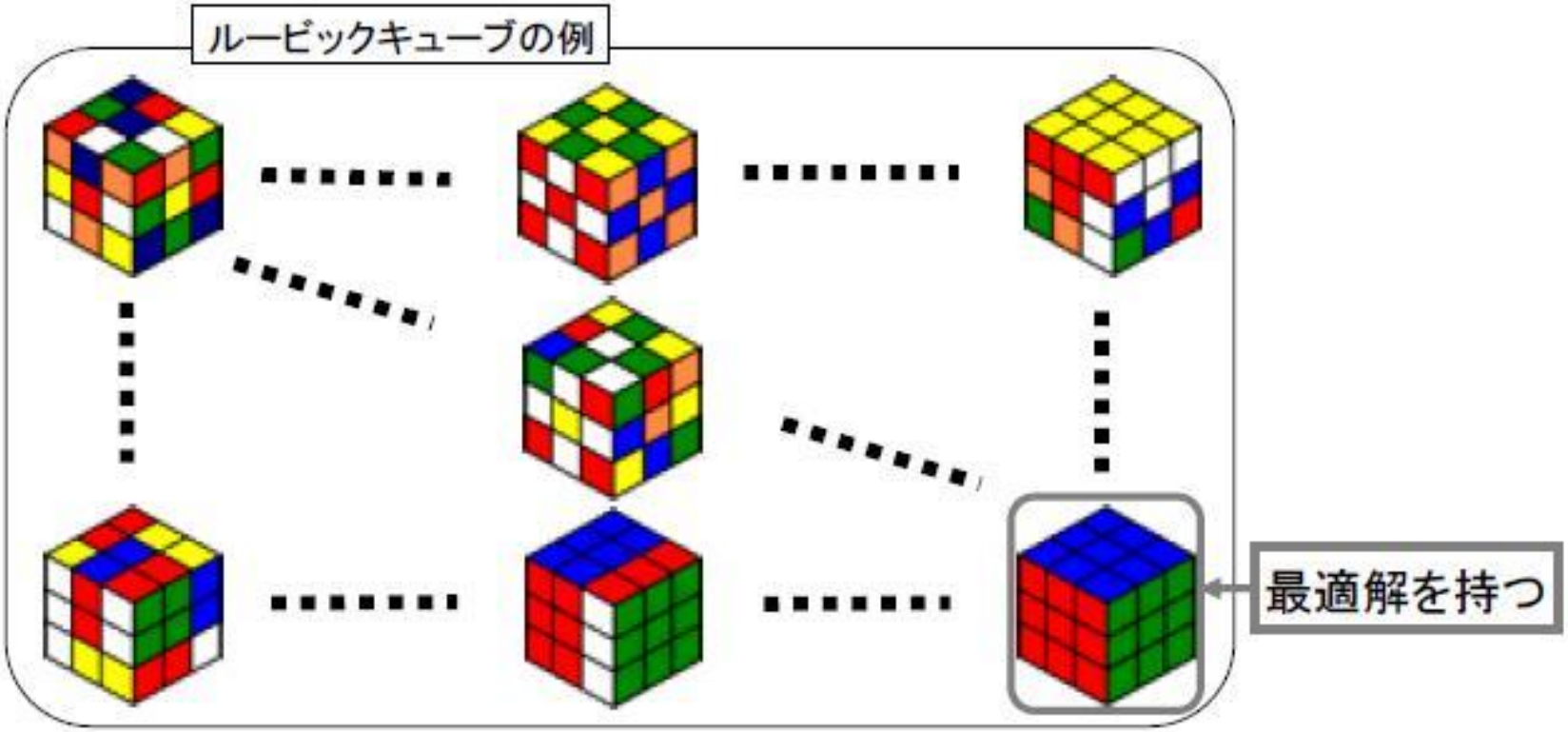
- 分割法1:
  - 1面の1色のベースレイヤ問題
  - 外側面(ミドルレイヤ)の4色問題
  - 1面のトップレイヤ問題



- 分割法2:
  - 1面のクロス問題
  - 外側の4色クロスの問題
  - 1面のトップのクロス問題
  - コーナーを揃える
  - コーナーをフリップする



ルッビク3x3の状態空間と問題分割法



再帰（帰納）(recursive (inductive))

- 階乗
  - $0! = 1 \; and \; 1! \; = 1$
  - $(n+1)! \; = \; (n+1) * n!$
- フィボナッチ数 (*Fibonacci number*)
  - $fib(0) = 0, fib(1) = 1$
  - $fib(x) = fib(x-2) + \; fib(x-1) \;\; x > 1 \; and \; x \in N$
- フィボナッチ数列 (*Fibonacci sequence*)
  - *[0,1,1,2,3,* 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610,…]

再帰（帰納）(recursive (inductive))

- 階乗

```
import functools
import sys

def factorial (n):
    if n == 1:
        return 1
    else :
        return n * factorial(n - 1)

def applicativeFactorial (n) :
    def mult(x,y): return x * y
    return functools.reduce (mult, range(1,n+1))

def iterFactorial (n) :
    retval = 1
    for i in range (n,0,-1) :
        retval = retval * i
    return retval
```

再帰(帰納) (recursive(inductive))

- フィボナッチ数、数列

```
def fib(x):
    global numFibCalls
    numFibCalls += 1
    if x==0:
        return 0
    elif x==1:
        return 1
    else:
        return fib(x-1)+fib(x-2)

def testFib(n):
    seq = []
    for i in range(n+1):
        global numFibCalls
        numFibCalls = 0
        f_i = fib(i)
        print("n=",i,"のフィボナッチ数は",fib(i))
        print("フィボナッチ再帰関数は",numFibCalls,"回に呼び出した.")
        seq.append(f_i)
    return(seq)

fib_seq = testFib(15)
print("¥n数列：",fib_seq)
```

再帰 （帰納） (recursive (inductive))

- 分割統治(divide-and-conquer)
  - 部分問題は元の問題よりも解くのが簡単である
  - 部分問題の回答は、元の問題を解くために統合できる
  - ある文字列は回文かどうか調べること:
    - sを文字列と仮定
    - sが回文ならTrueを返し, それ以外ならFalseを返す

    ただし　可読点、空白、大文字、小文字は無視する

再帰 （帰納）

- 回文(is Palyndrome)

```
def isPalindrome(s):
    sent=s
    print("Start...")
    def toChars(s):
        s=s.lower()
        letters=''
        for c in s:
            if c in 'abcdefghijklmnopqrstuvwxyz':
                letters+=c
        return letters
    def isPal(s):
        print('isPal関数は',s,'文字列で呼出す.')
        if len(s)<=1:
            print(sent,'は回文である.')
            return True
        else:
            print(s,'の回文を求める.')
            answer=s[0]==s[-1] and isPal(s[1:-1])
            return answer
    return isPal(toChars(s))


reply = isPalindrome('aba')
print("回答：",reply)
```

まとめ

- 問題分割法は問題定式化の一つの手法.
- 問題分割法:
  - **問題をいくつかの単純な部分問題に分割**
  - **[初期状態、最終状態]のペア**
  - **AND/OR木で表現**
  - **AND/OR木を探索することに帰着**
  - **素問題は分割できない問題**
- 例題:ハノイの塔、Rubik's Cube
- 分割統治と再帰(帰納)