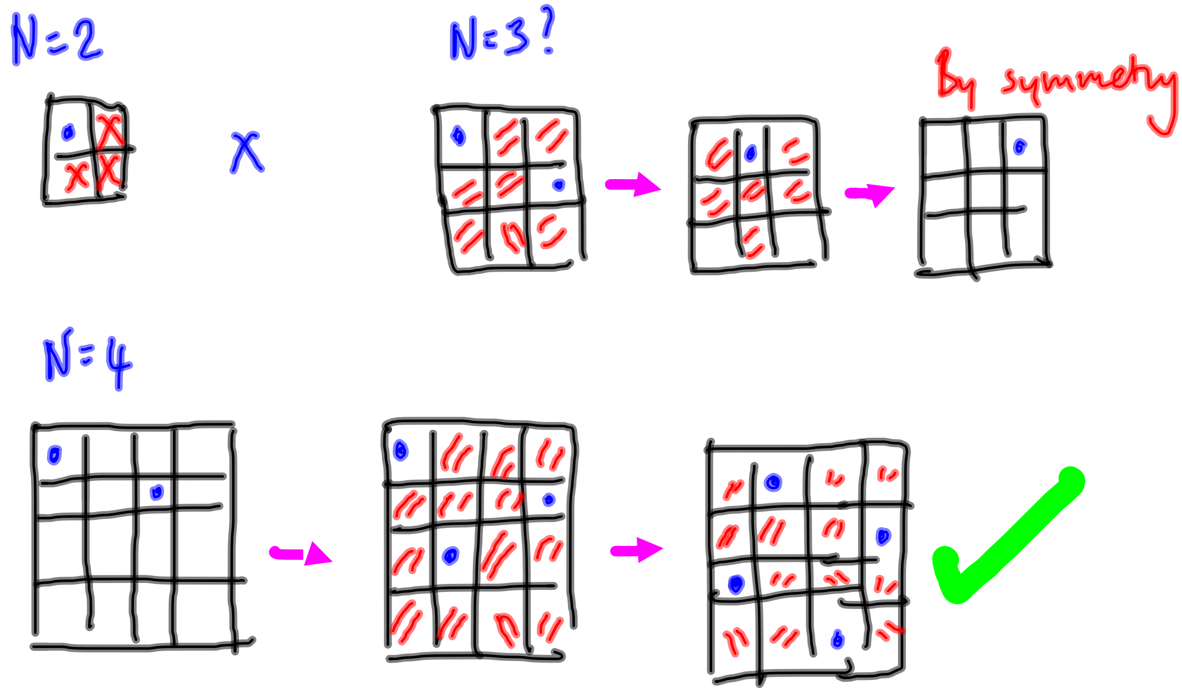
Sudoku

Backtracking

N queens on an $N \times N$ chessboard so that no pair of queens attack each other

i.e. no pair is on the same row, column, diagonal



Claim: For all $N \geq 4$, there are solutions

Program the 8x8 case

One queen per row & one queen per column

Place Q1 in the first free position on row 1

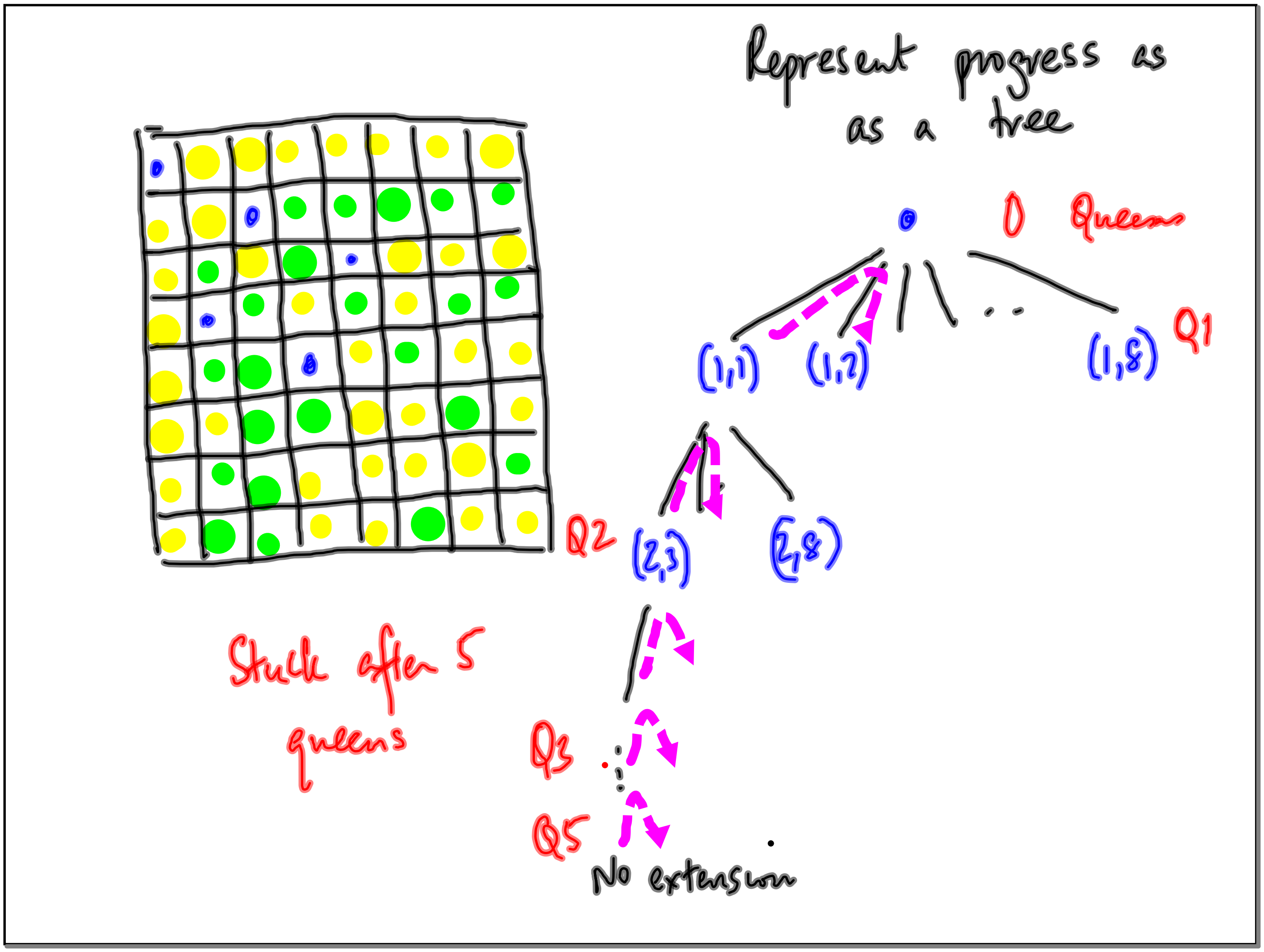
Place Q2 in " " " " " row 2

Cannot place Qj

go back and try next free position for Q_{j-1}

⋮

Place Q8 ✓


Represent progress as as a tree
0 Queens
Q1
(1,1)
(1,2)
(1,8)
Q2
(2,3)
(2,8)
Stuck after 5 queens
Q3
Q5
No extension

Need

Data representation for the "state" of the board

Need an efficient way to update this representation

- Adding a queen
- Remove a queen (backtracking)

Assuming we have a "good" representation

How does backtracking work?

```
def placequeen(i):
    for each free position (i,c)
        place Qi at (i,c) & update representation
        if i==8:
            return(True)
        else:
            try = placequeen(i+1)
            if try:
                return(True)
```

```
def placequeen(i):
    for each free (i,c)
        update representation
        if i==8:
            return(True)
        else:
            try = placequeen(i+1)
            if try:
                return(True)
        remove (i,c) from representation
    return(False)
```

Representations

Naive: 8x8 (9x9) array of {0,1}

board[i][j] == 1 iff Qi is at (i,j)

Expensive to calculate if (r,c) is free

Add information about attacked squares 8x8 array

attack[i][j] = k if Qk was first queen to attack (i,j) attack[i][j] $\in \{0,1\}$

Easy to check free(r,c)

Easy to update after placing Q_i at (i,c)

What happens if we backtrack?

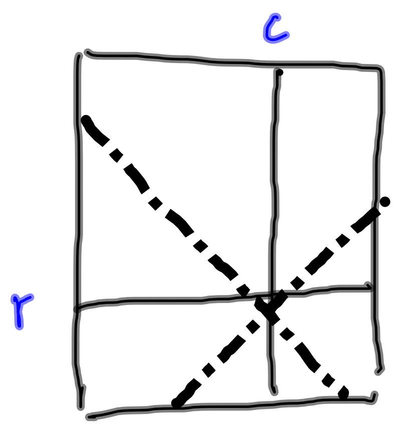
If attack[i][j] = k $\Leftrightarrow$ Q_k is earliest queen to attack

Remove Q_i $\rightarrow$ Restore all squares labelled i to 0

Diagonal for (r,c)

$(i+j) = (r+c)$

$(i-j) = (r-c)$



Can compress board : $8 \times 8 \to \{0,1\}$

to board : $8 \to \{1,2,\ldots,8\}$

board[i] = j $\to$ Q_i is in row i
col j

Can we compress attack from $O(N^2)$ space

to $O(N)$ space ?

Hint: Record attack information in larger units than single squares

Record for each row, column, diagonal

"Is row r attacked?"

⋮

8 rows

8 columns

$2 \cdot 8 - 1$ ⤡ diags

$2 \cdot 8 - 1$ ⤢ diags

In general $N \times N$ board

N rows
N cols
$2N-1$ ↖↘ diags
$2N-1$ ↙↗ diags

$O(N)$ objects

Identify these objects:

rows: 1..8
cols: 1..8
diags: According to $i+j$, $i-j$

rows : 1..8

cols : 1..8

lrdiags : -7..+7

rldiags 2..16

Add Q_i at (i, c) :

board[i] = c

row[i] = 1 # Attacked

col[c] = 1

lrdiag[i-c] = 1

rldiag[i+c] = 1

Remove Q_i from (i,c)

board[i] = 0

row[i] = 0

col[c] = 0

lrdiag[i-c] = 0

rldiag[i+c] = 0

When is (r,c) attacked?

row[r] == 1 or col[c] == 1 or lrdiag[r-c] == 1

or rldiag[r+c] == 1

In our function placequeen(i)

Use the explicit update code where we write "Update representation"

and also insert code to check free(r,c)

```
for c in range(1,9):
    if free(i,c):
        ⋮
```

Initialize

row[i]=0 $\forall i$

col[j]=0 $\forall j$

board[i]=0 $\forall i$

lrdiag[d]=0 $\forall d$

rldiag[d]=0 $\forall d$

~~placequeen(1)~~ try = placequeen(1)

if try:
    print out board[1..8]

board[], row[] etc are updated both inside and outside placequeen()

Must declare these "global"

This gives one solution.

How do we enumerate all solutions?