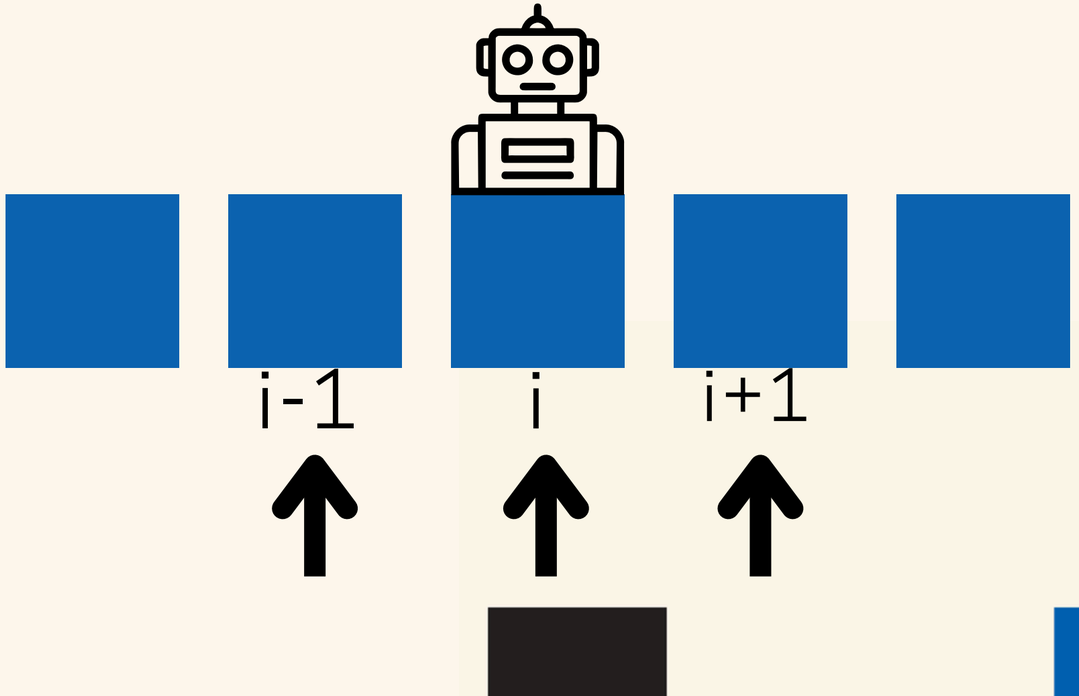
AUTOMATA

STATEMENT

- Cells: 0 ... N-1
- Cells have heights: P[0], ..., P[N-1]
- P is a permutation

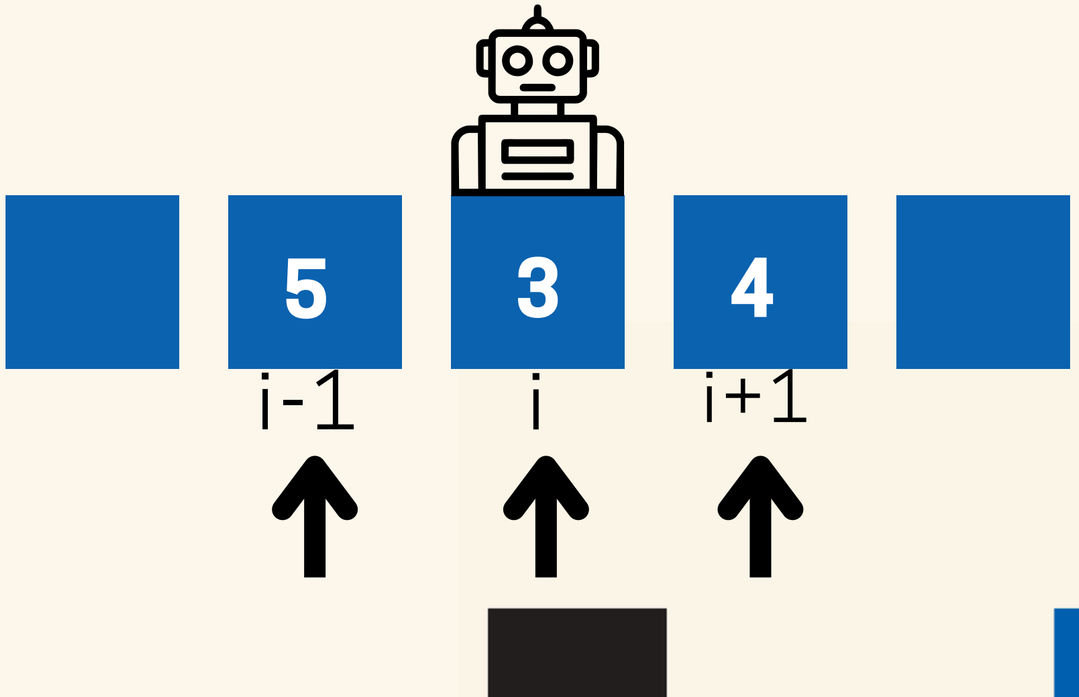
STATEMENT

- Cells: 0 ... N-1
- Cells have heights: P[0], ..., P[N-1]
- P is a permutation



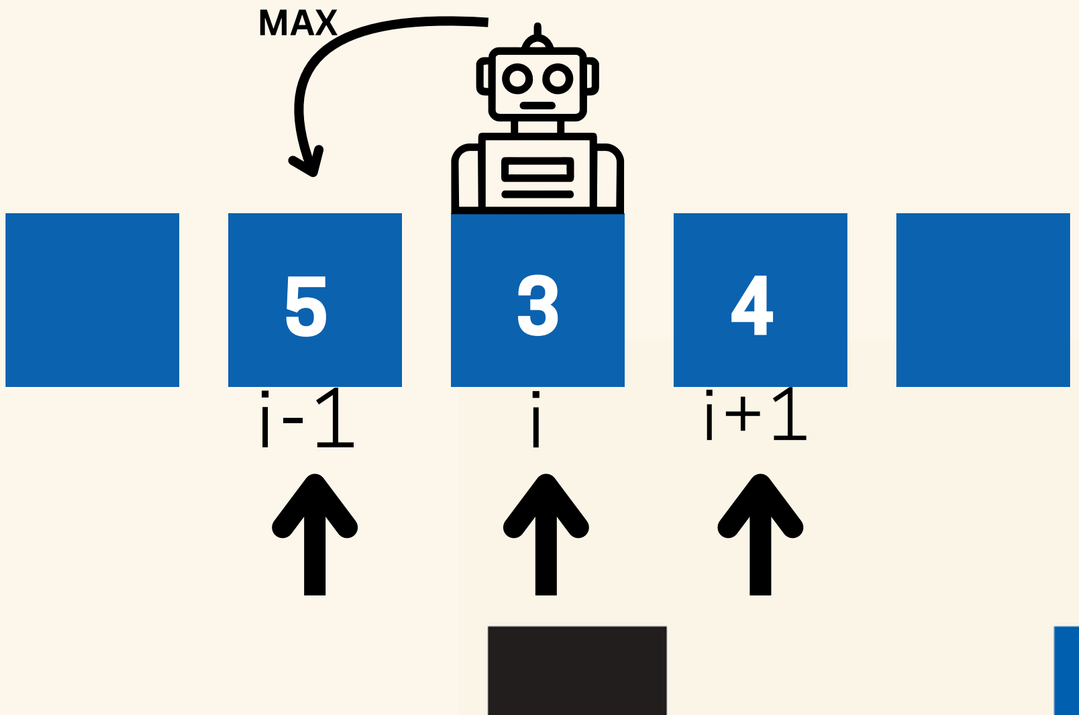
STATEMENT

- Cells: 0 ... N-1
- Cells have heights: P[0], ..., P[N-1]
- P is a permutation



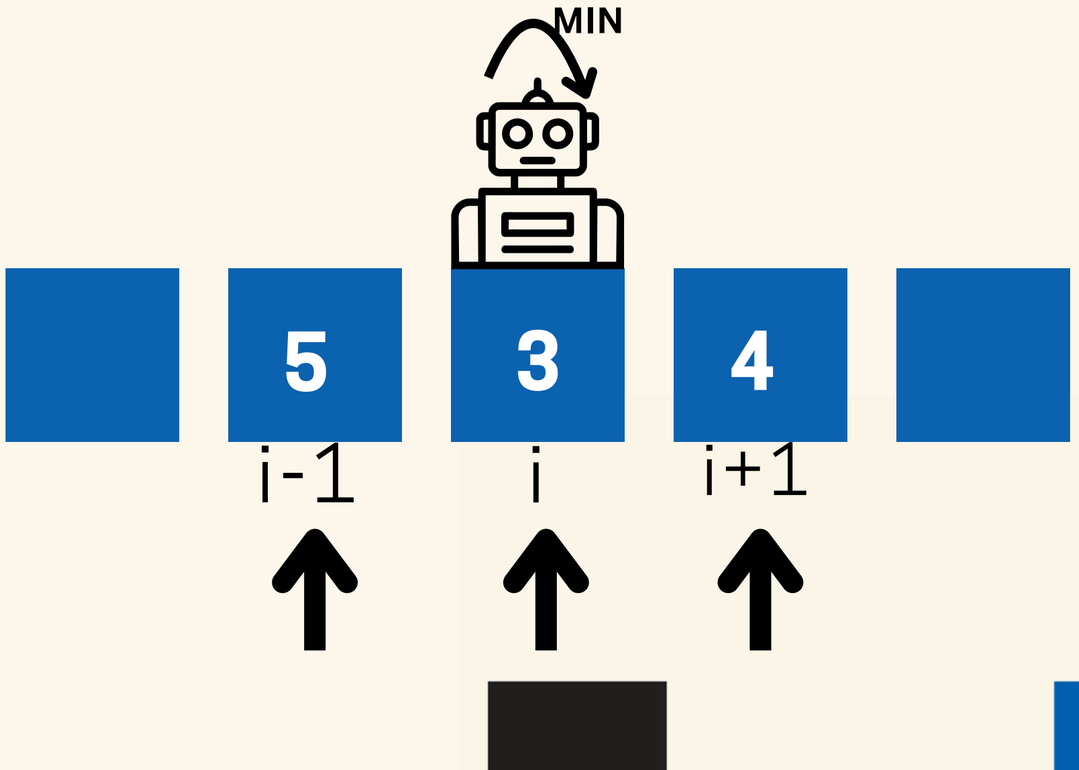
STATEMENT

- Cells: 0 ... N-1
- Cells have heights: P[0], ..., P[N-1]
- P is a permutation



STATEMENT

- Cells: 0 ... N-1
- Cells have heights: P[0], ..., P[N-1]
- P is a permutation



STATEMENT

TASK:

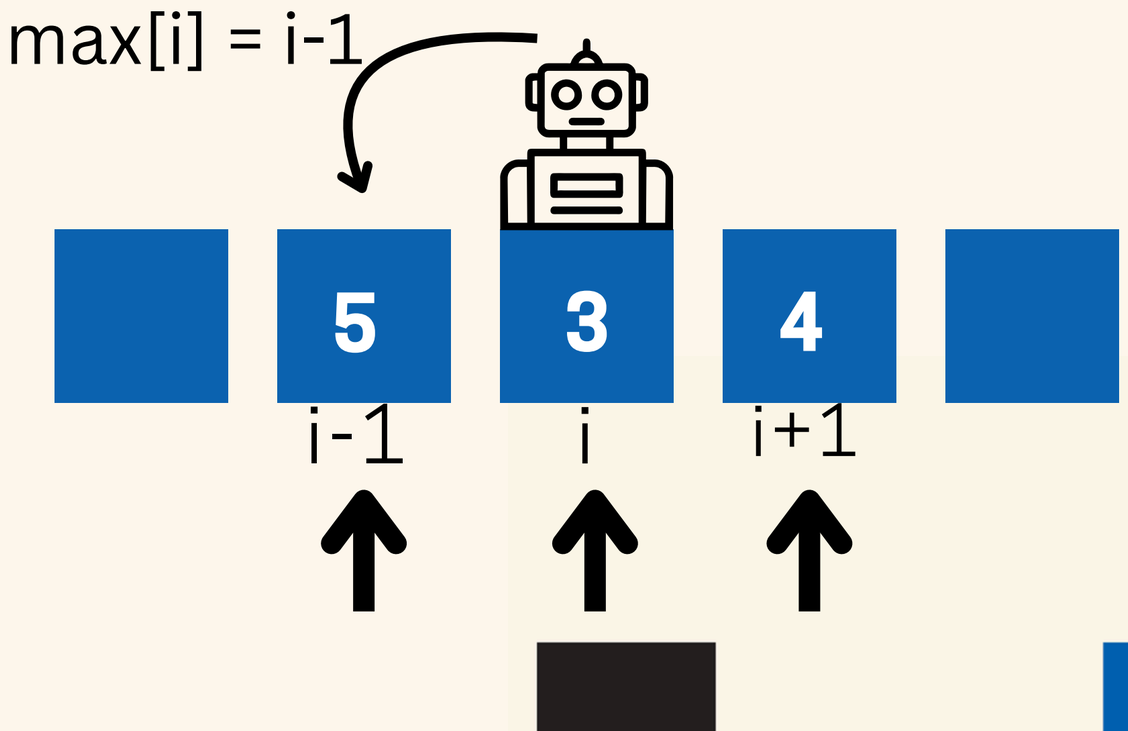
- Given Q queries
- Each query gives positions X[0], ..., X[K-1]
- If we place robots there, *does some program exist,* s.t. the robots finish at the same cell

CONSTRAINTS

- N <= 2*10^5
- Q <= 5*10^5
- The sum of K over all queries does not exceed 10^6
- The queries must be answered online (the moment they are given)

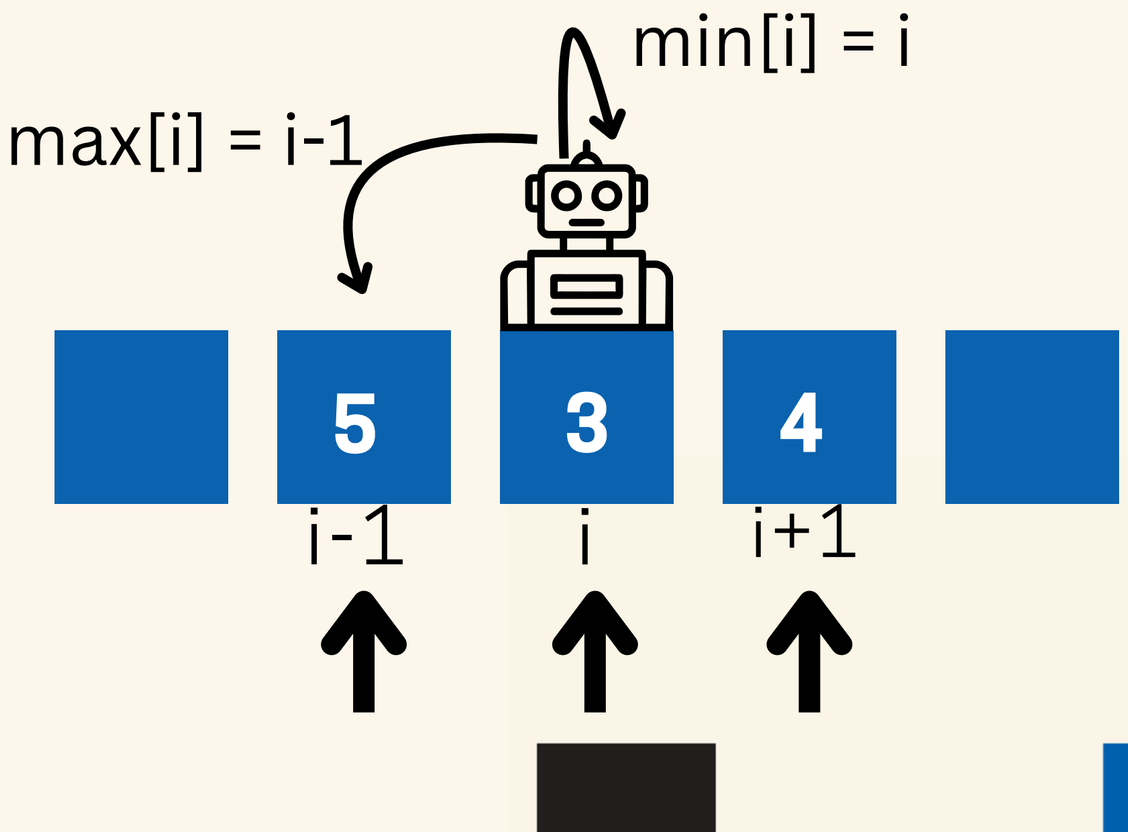
NOTATION

- Two helper arrays: max[i], min[i]



NOTATION

- Two helper arrays: max[i], min[i]



NOTATION

- Two helper arrays: max[i], min[i]

OBSERVE:

- **after getting MAX command, you go i->max[i]**
- **similarly for MIN, you go i->min[i]**

SUBTASK 1

CONSTRAINTS:

- **K=2**
- **if a program exists, it is exactly one command**

SUBTASK 1

CONSTRAINTS:

- **K=2**
- **if a program exists, it is exactly one command**

Two options:

- [MIN]
- [MAX]

SUBTASK 1

CONSTRAINTS:

- **K=2**
- **if a program exists, it is exactly one command**

Two options:

- [MIN] → works if min[x0] == min[x1]
- [MAX] → works if max[x0] == max[x1]

SUBTASK 1

CONSTRAINTS:

- **K=2**
- **if a program exists, it is exactly one command**

Two options:

- [MIN] → works if min[x0] == min[x1]
- [MAX] → works if max[x0] == max[x1]

SUBTASK 2

CONSTRAINTS:

- **K=2**
- **if a program exists, it is at most 5 commands**

Idea: try all possible 5-command combinations

SUBTASK 1-5

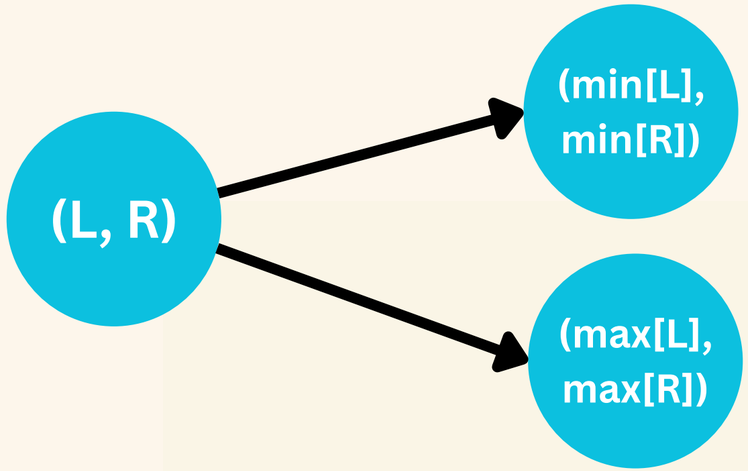
MAIN CONSTRAINT: K=2

- IDEA: Think about the given query with two positions L and R as graph problem.
- Nodes are marked as (L, R)

SUBTASK 1-5

MAIN CONSTRAINT: K=2

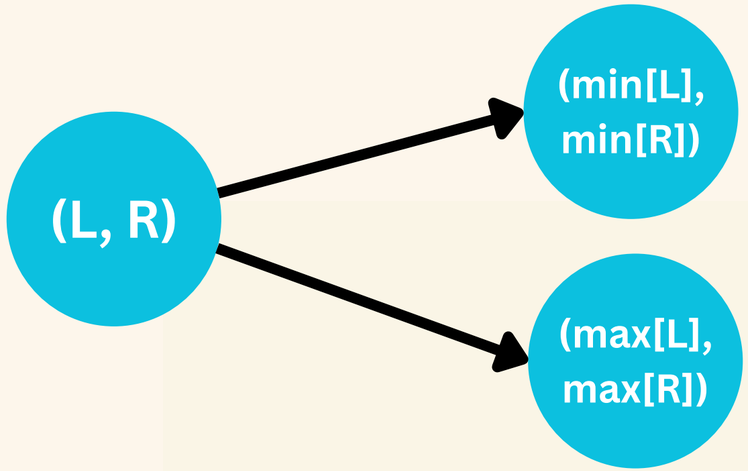
- IDEA: Think about the given query with two positions L and R as graph problem.
- Nodes are marked as (L, R)
- You can traverse (L, R) to (min[L], min[R]) or to (max[L], max[R])



SUBTASK 1-5

MAIN CONSTRAINT: K=2

- IDEA: Think about the given query with two positions L and R as graph problem.
- Nodes are marked as (L, R)
- You can traverse (L, R) to (min[L], min[R]) or to (max[L], max[R])

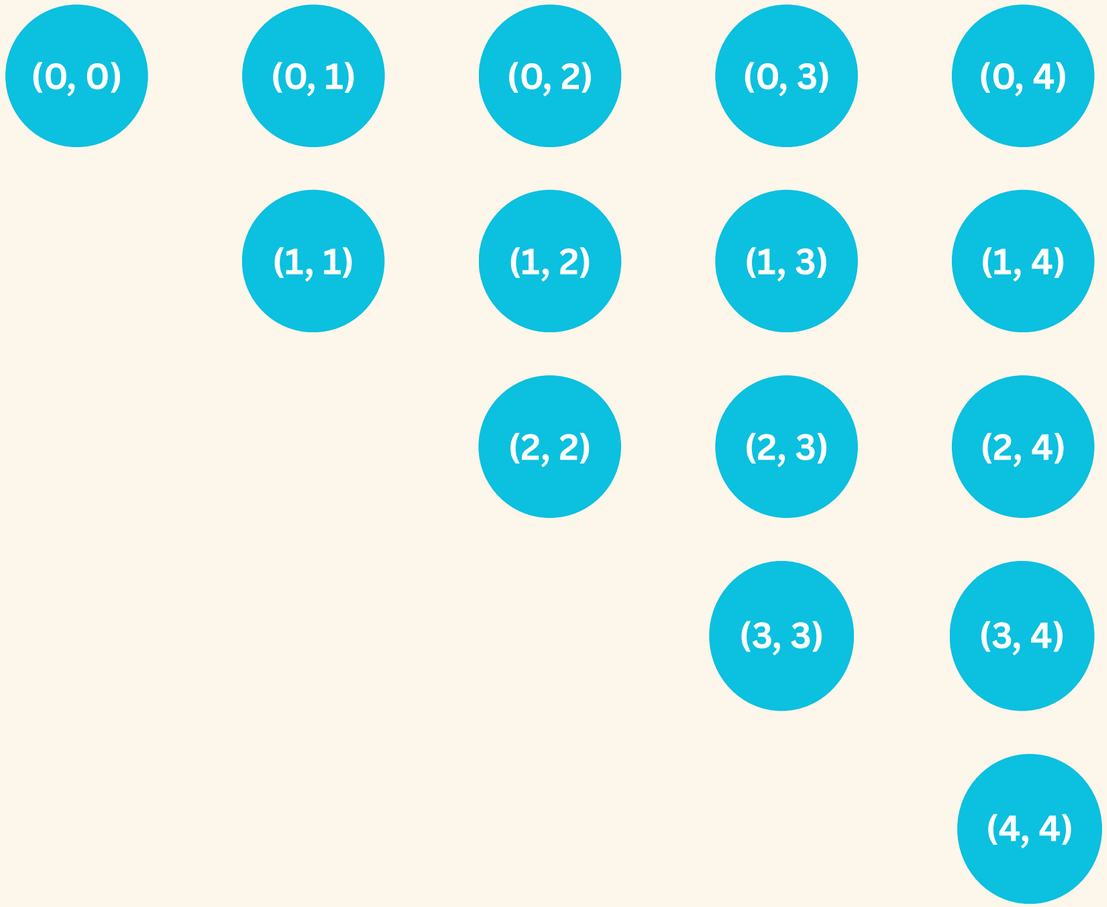


GOAL: From (L, R) move to some position (i, i)

EXAMPLE

P = [3, 2, 1, 0, 4]

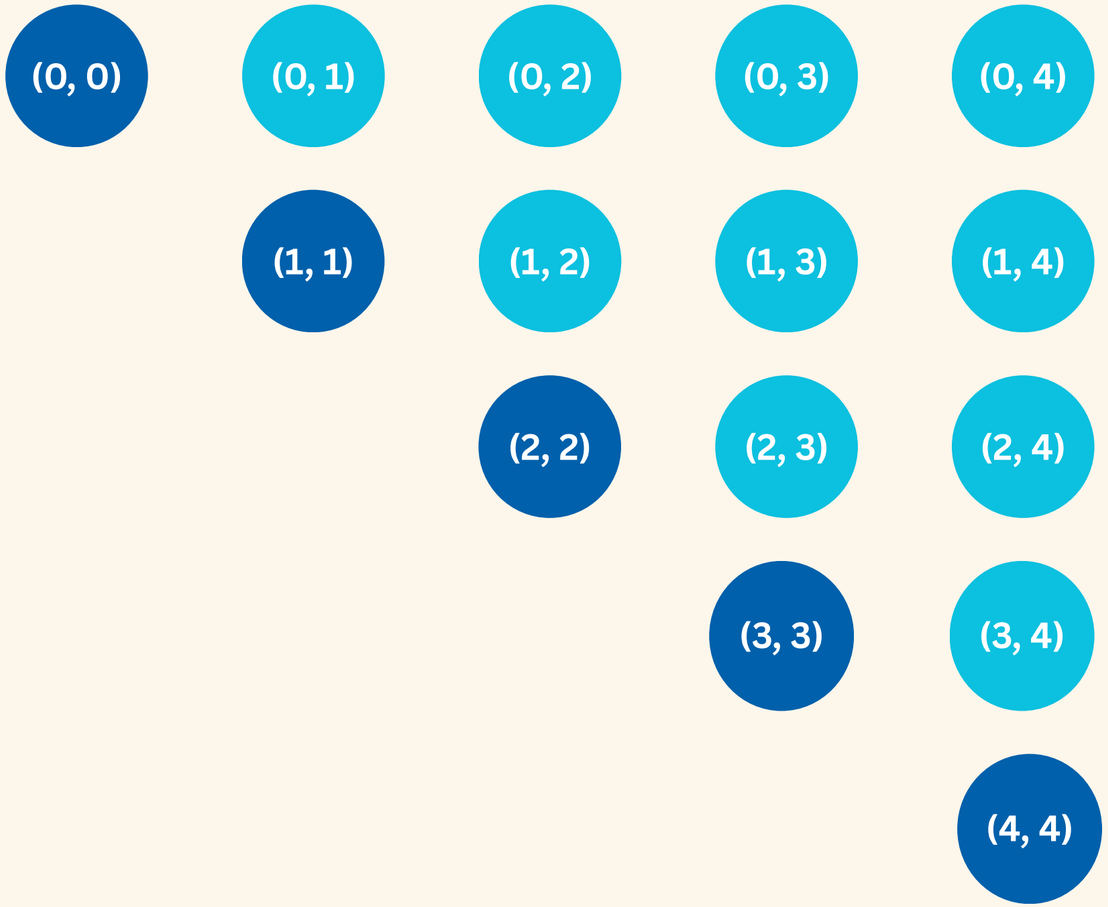
1. **STATES**



EXAMPLE

P = [3, 2, 1, 0, 4]

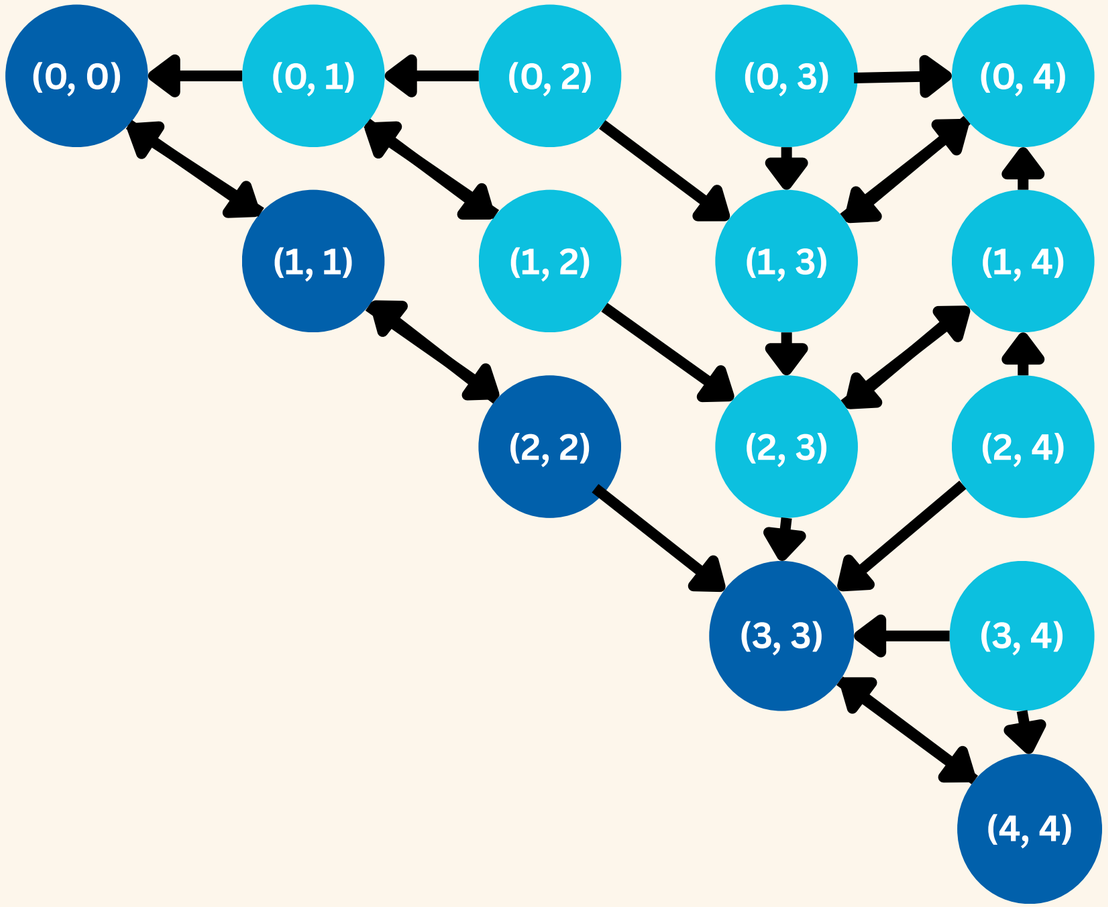
1. **STATES**
2. **SPECIAL STATES**



EXAMPLE

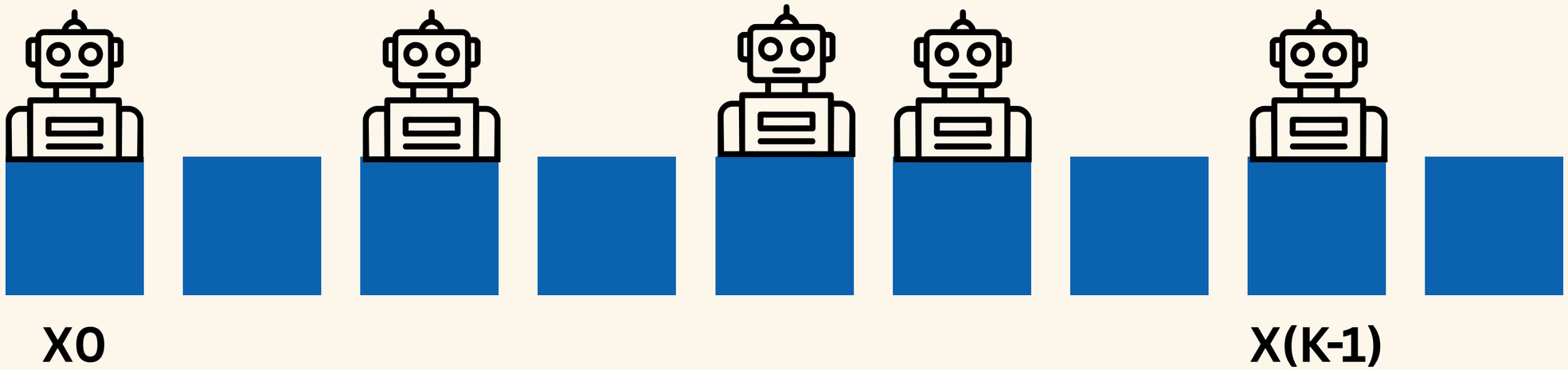
P = [3, 2, 1, 0, 4]

1. **STATES**
2. **SPECIAL STATES**
3. **EDGES**



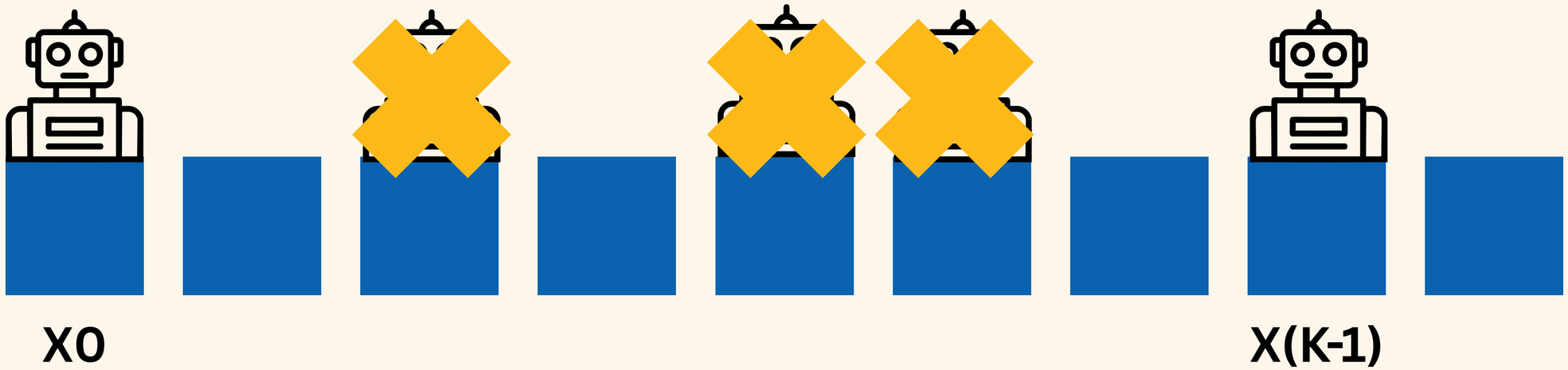
OBSERVATION

We only care about the left-most and the right-most possible positions: at X[0] and X[K-1].



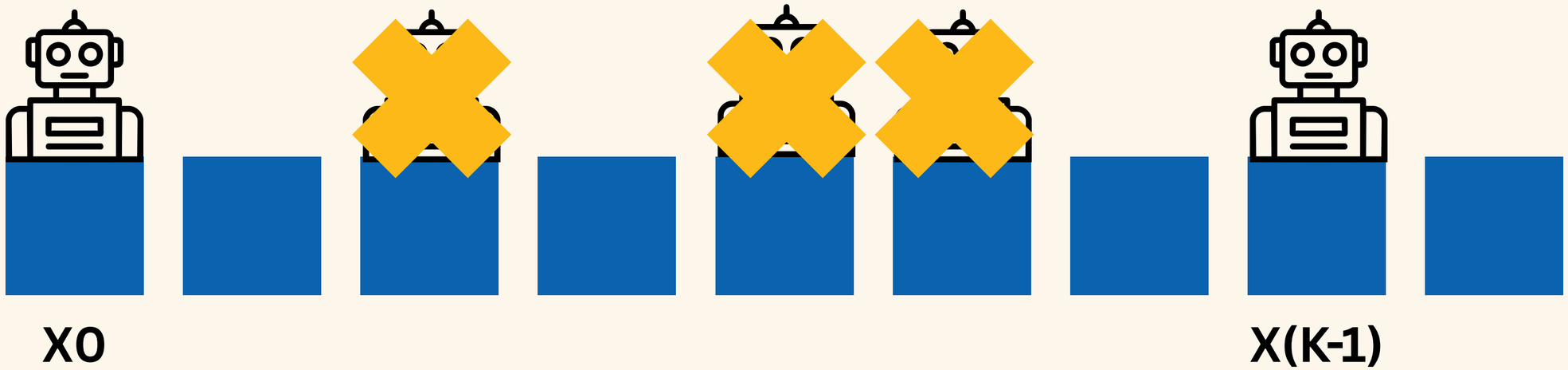
OBSERVATION

We only care about the left-most and the right-most possible positions: at X[0] and X[K-1].

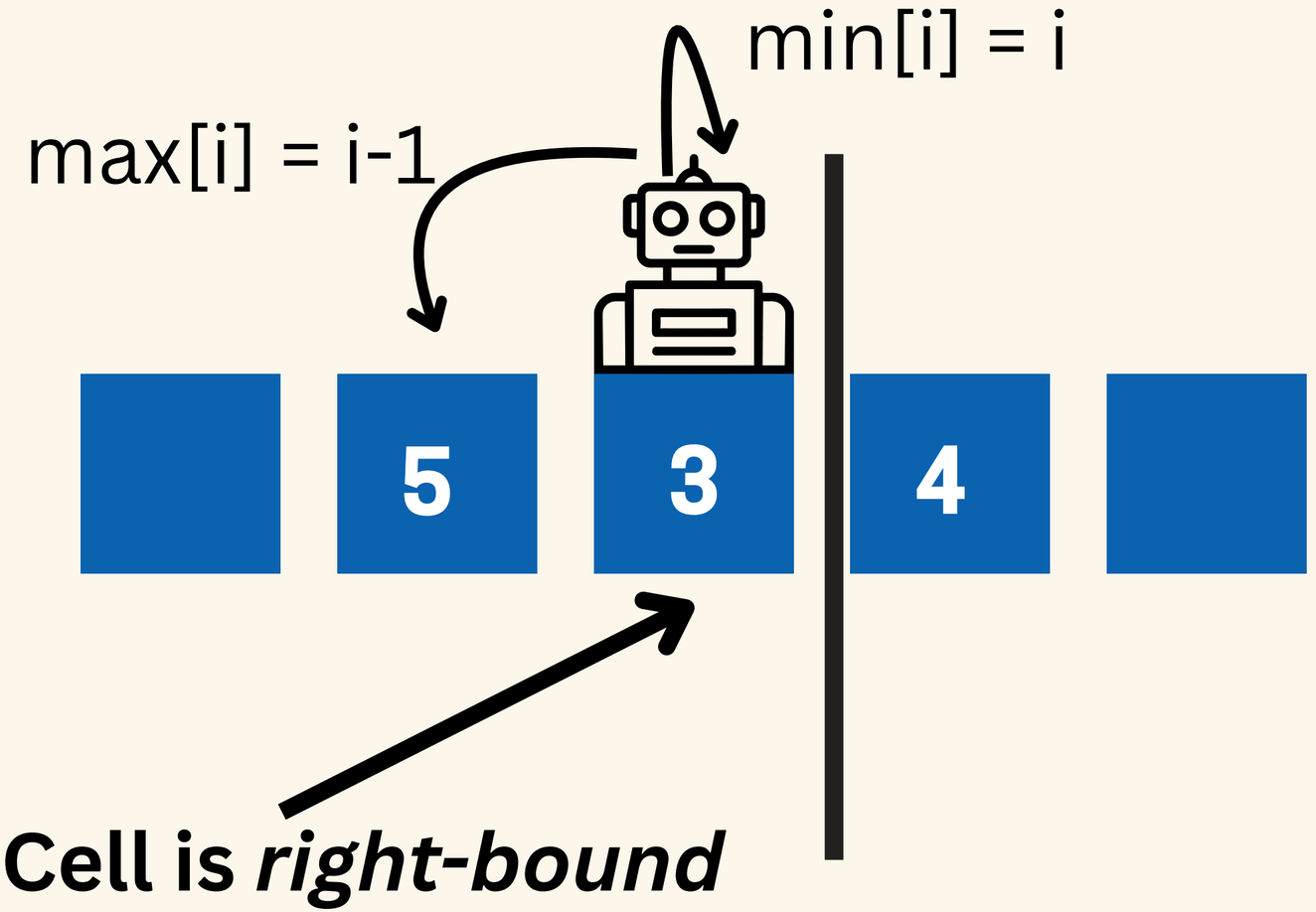


OBSERVATION

We only care about the left-most and the right-most possible positions: at X[0] and X[K-1]. => We can say we have K=2



BOUNDS



BOUNDS

- **i** is *left-bound:* max[i] >= i AND min[i] >= i.
- **i** is *right-bound*: max[i] <= i AND min[i] <= i.

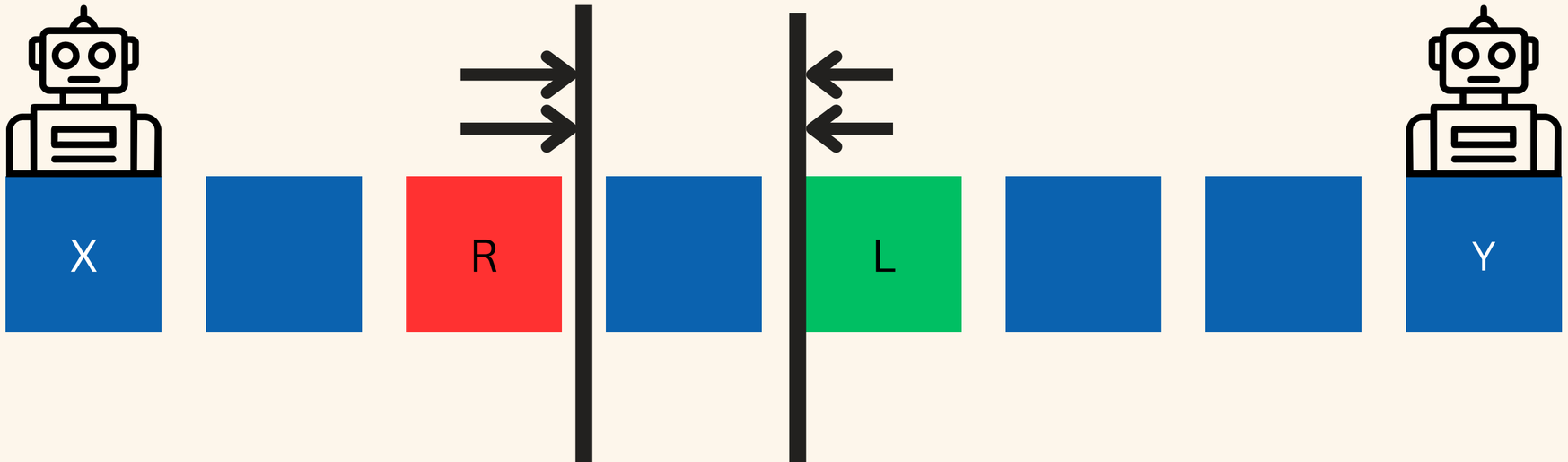
SOLUTION

We need to answer query X, Y (two robot positions), and say if it is *good*.

SOLUTION

We need to answer query X, Y (two robot positions), and say if it is *good*.

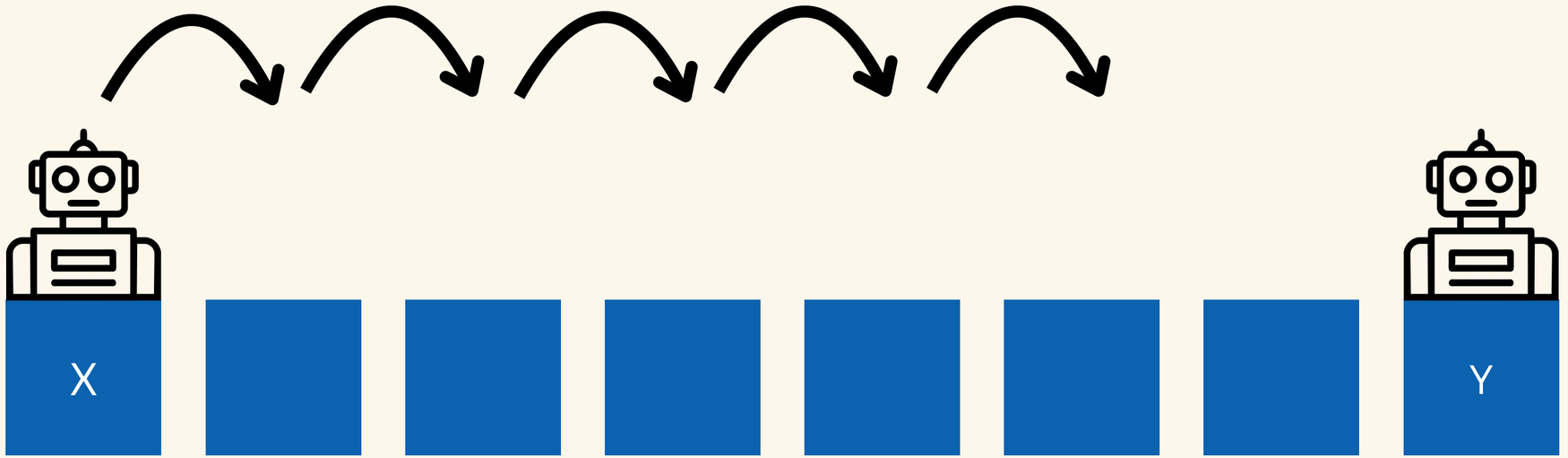
- Case X==Y → good
- Case when there is X <= R < L <= Y → bad



SOLUTION

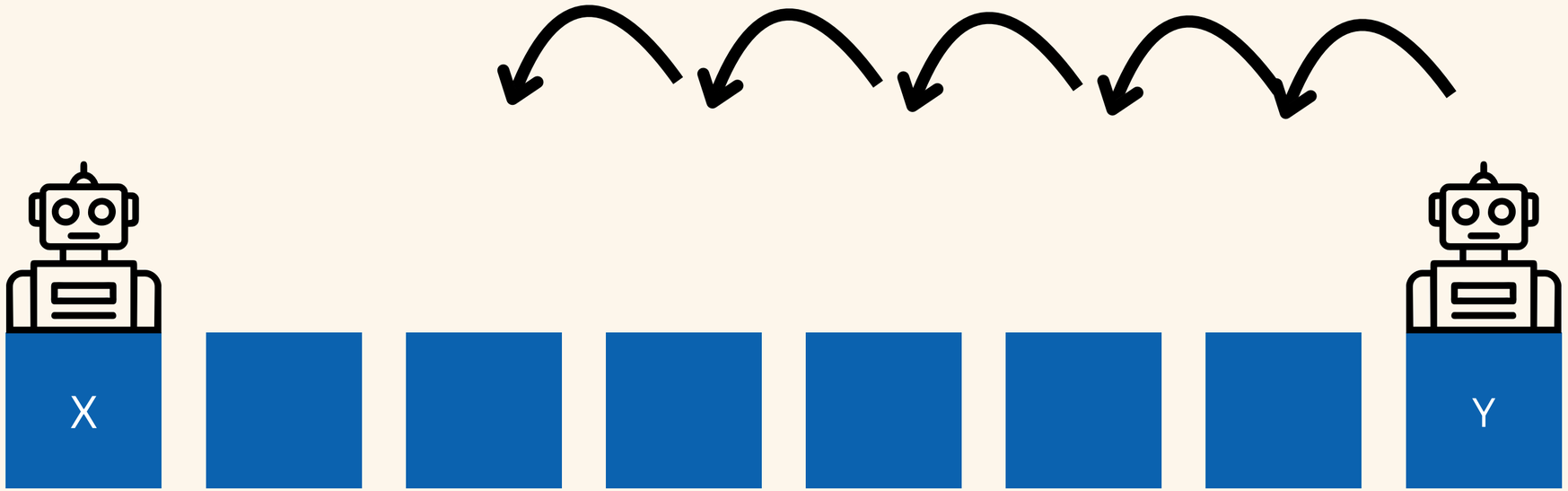
Case: no right-bound in range [X, Y)

Idea: robot from X can be moving to the right until it meets the other one



SOLUTION

Same idea for when there is no left-bound in (X, Y].



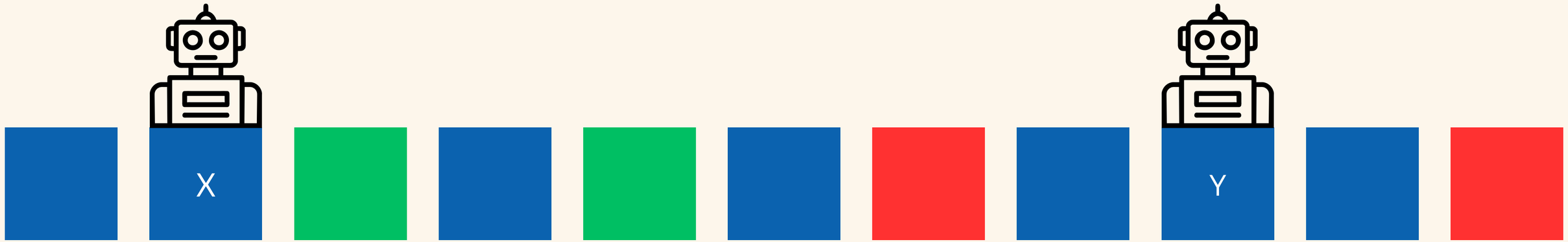
SOLUTION

Hardest case: between X and Y all left-bounds are to the left of all right-bounds.

SOLUTION

Hardest case: between X and Y all left-bounds are to the left of all right-bounds.

If the **first** bound to the right of Y is **right-bound** → good
(or, if first to the left of X is left-bound → good)



SOLUTION

Otherwise, check if X can reach a good bound before Y reaches a bad bound.