Controlling Robots via Large Language Models

Sanjiban Choudhury

Cornell Bowers CIS
Computer Science

Today is the last day.

Have we learnt anything useful at all?

Let's go back to Episode #1

The Problem: Real world is complex!


Object Detection, Segmentation, NERFs
Lec #19,20!
High-level Task Planner
Perception
State
s_t
Action
a_t
Low-level Policies
RLHF
Lec #21!
Model-based RL (Dreamer)
Lec #17,18!
Offline IL / RL
Lec #23!
Sim2Real
Lec #24!
Foundational Algorithms
Lec #1-16!


Object Detection, Segmentation, NERFs
Lec #19,20!
High-level Task Planner
TODAY!!
Perception
State
s_t
Action
a_t
Low-level Policies
RLHF
Lec #21!
Model-based RL (Dreamer)
Lec #17,18!
Offline IL / RL
Lec #23!
Sim2Real
Lec #24!
Foundational Algorithms
Lec #1-16!

Poll!

When poll is active respond at **PollEv.com/sc2582**

Send **sc2582** to **22333**

The Problem:

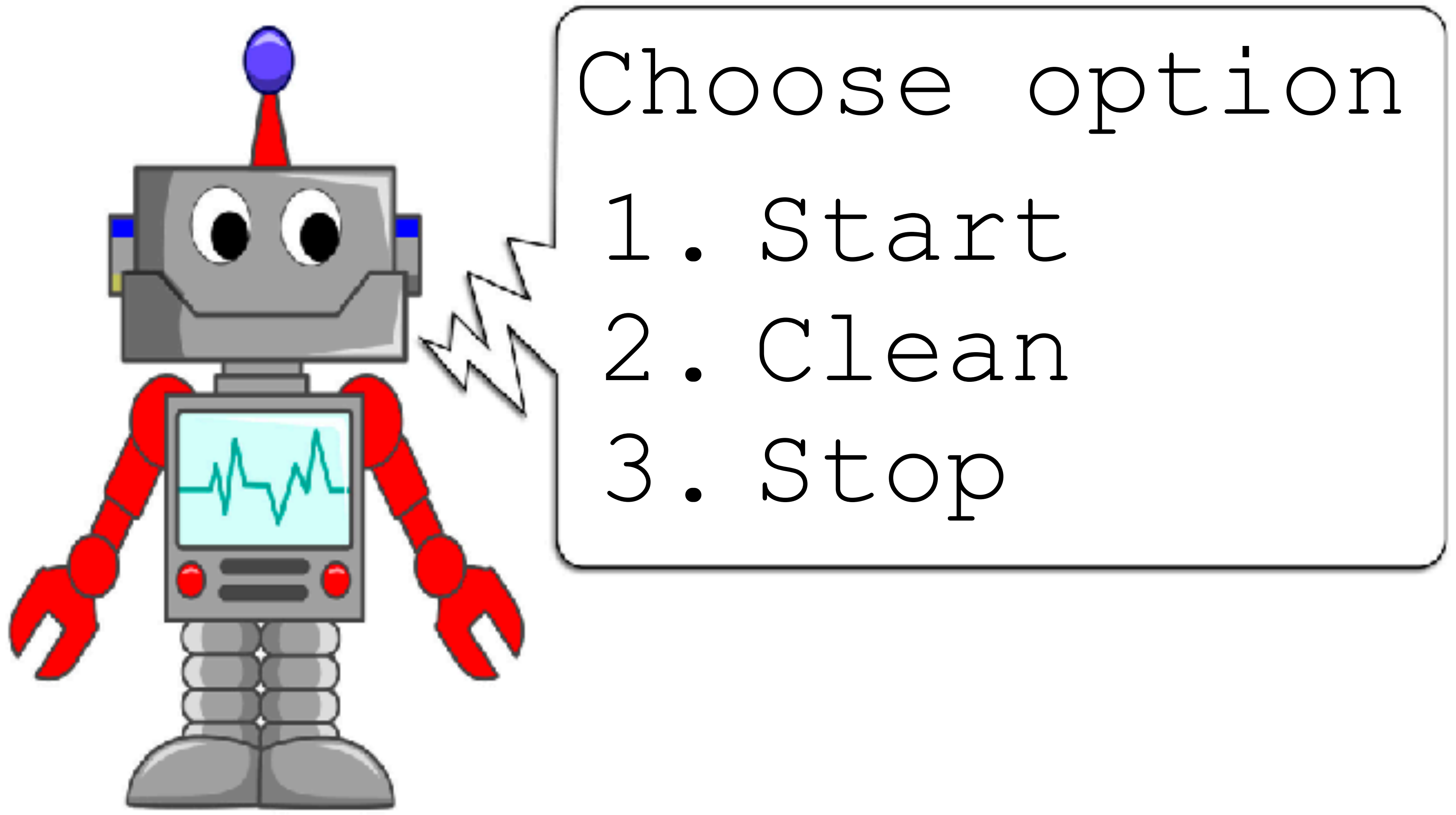
Many tasks are *personal*

Cooking is personal

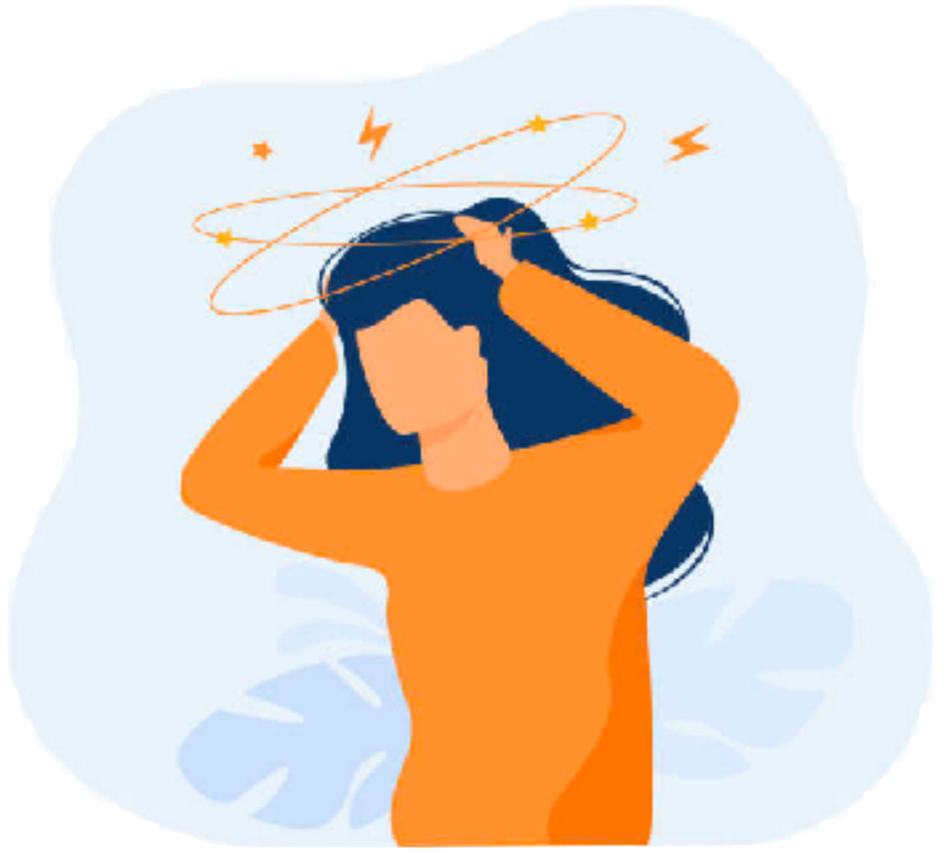
Home organization is personal

Robots today are NOT personal

Engineers program behaviors



Ship robot

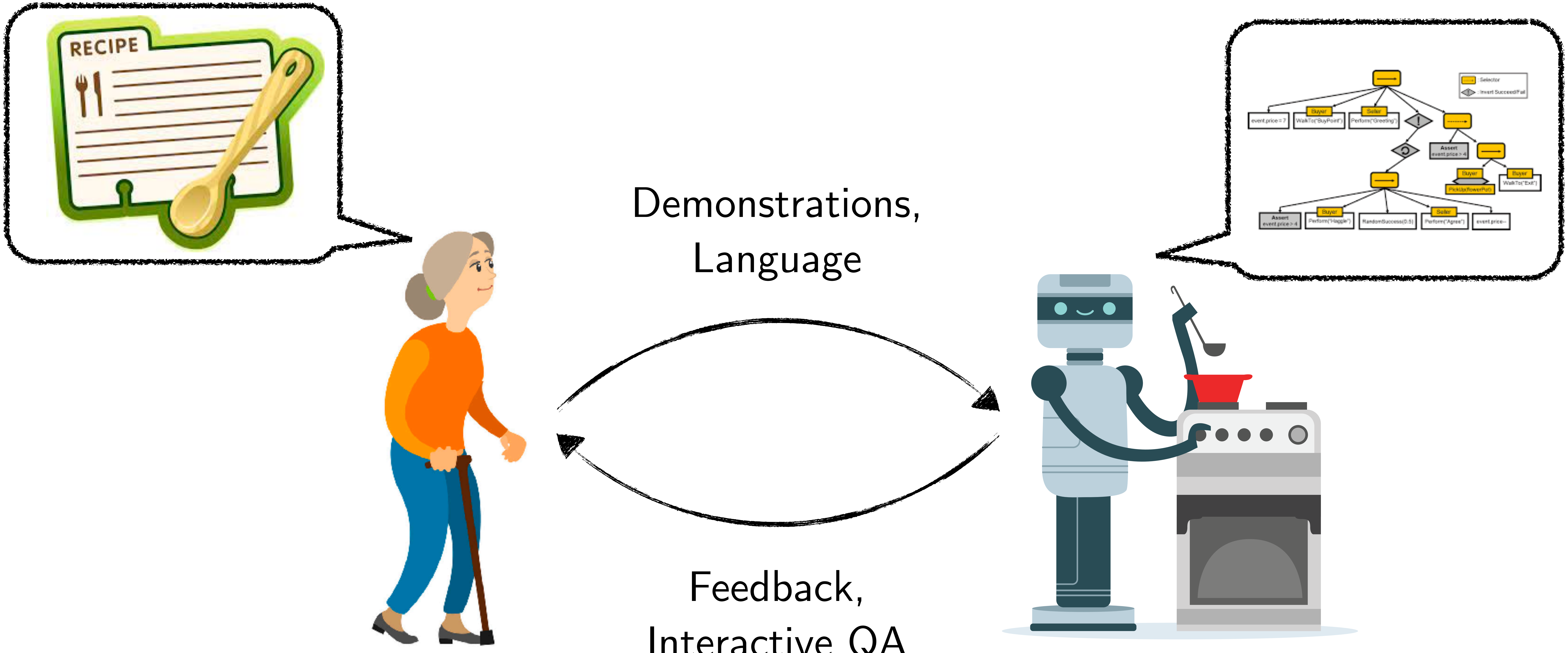


Frustrate users!

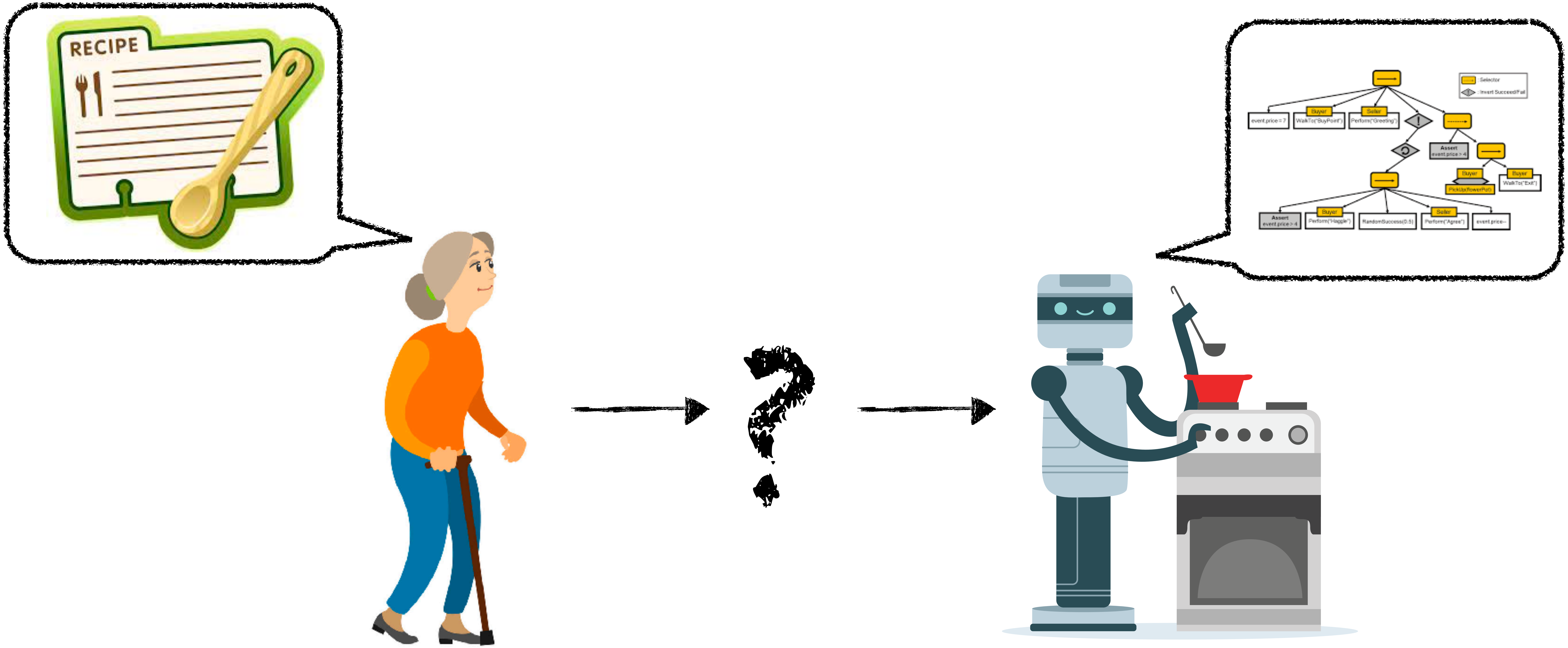
Cannot be flexibly re-programmed by everyday users

Can we implicitly program robots via natural interactions?

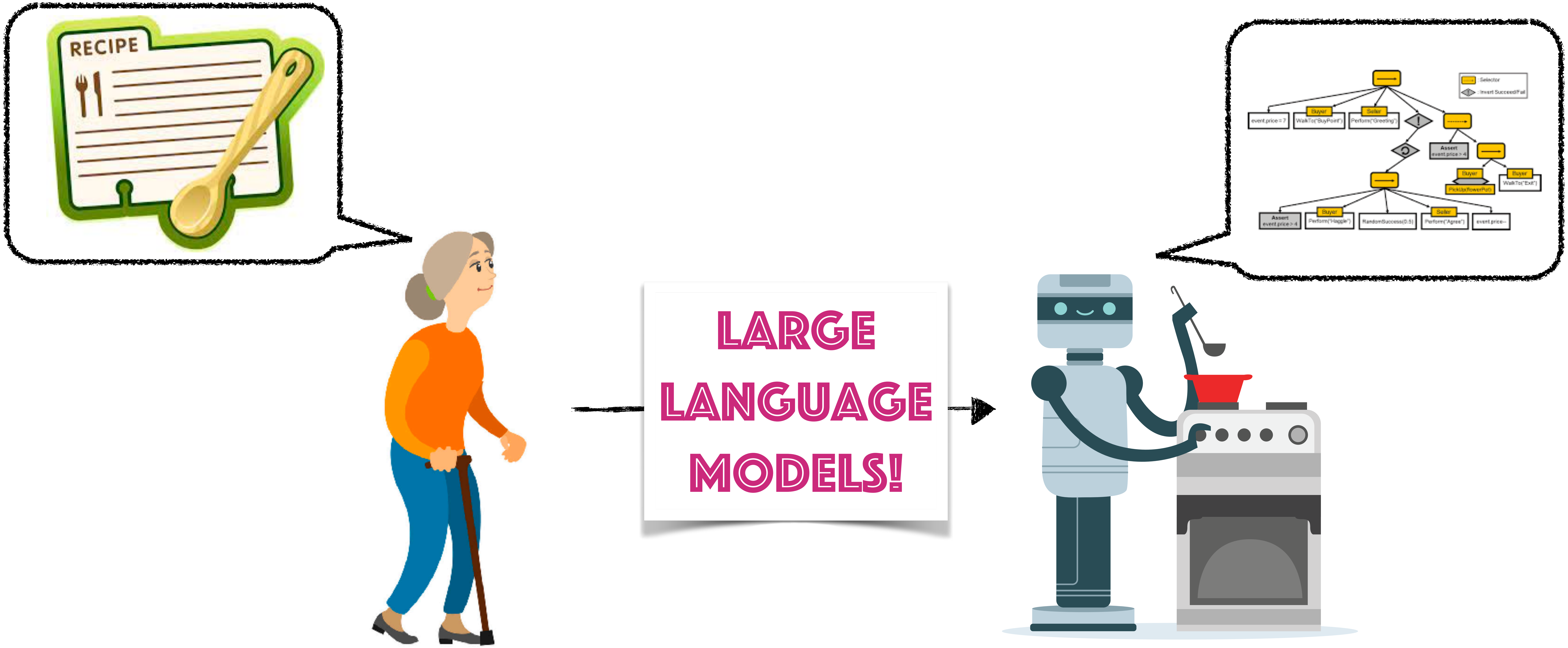
Programming via natural interactions



Question: How do we translate between humans and robots?



Large Language Models to the rescue!



An Example

MOSAIC

A Modular System for Assistive and Interactive Cooking

MOSAIC

A Modular System for Assistive and Interactive Cooking


Hey MOSAIC! It's so cold today! Can you suggest some soup recipe?
TP
Chicken tomato soup would be a great choice for a cold day. Let's start cooking!
No, I don't want chicken in it. How about mixed vegetable soup? I got a bunch of vegetables.
TP
If you are looking to use a bunch of vegetables and want a delicious soup, the mixed vegetable soup from our recipe list would be perfect! Let's get started on that!
TP
Shall Alice pour salt into the pot and then stir it for you? Shall Bob go get the soup base for you?

Activity!

Think-Pair-Share!

Think (30 sec): What are some of the challenges to convert natural language conversation to code that a robot can execute?

Pair: Find a partner

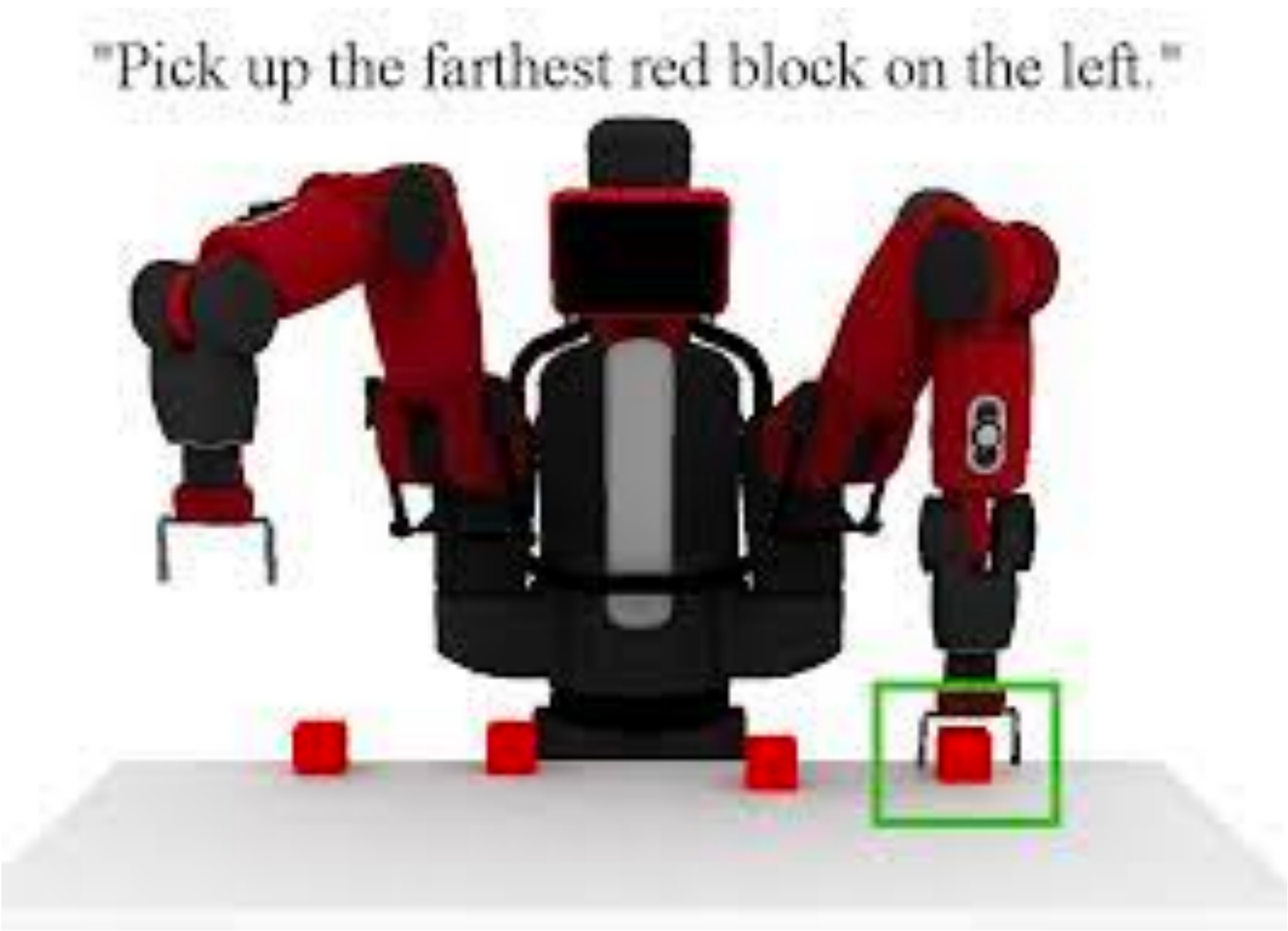
Share (45 sec): Partners exchange ideas

How things worked pre-LLM

Two Fundamental Challenges

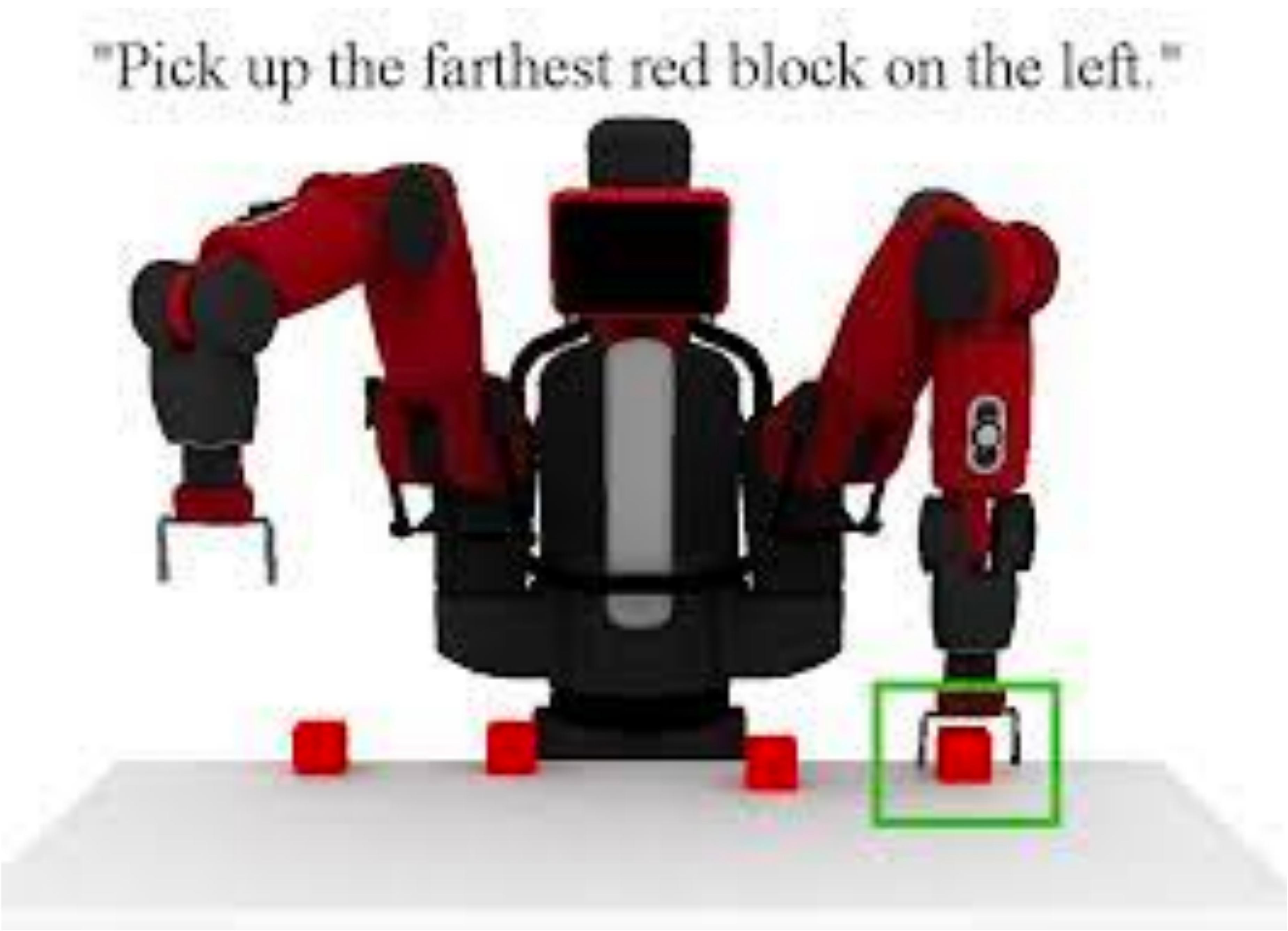
Two Fundamental Challenges

Challenge 1:
Ground natural language
in robot state

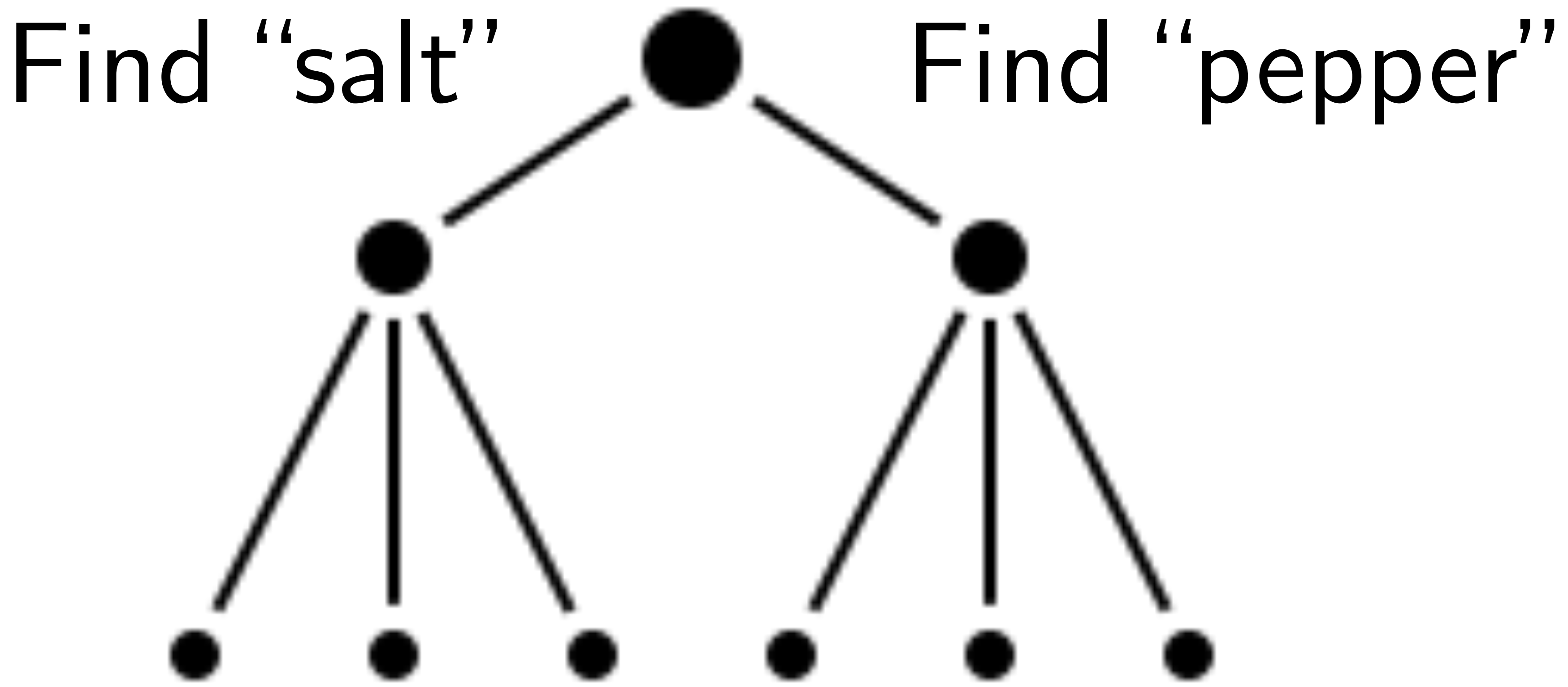


Two Fundamental Challenges

Challenge 1:
Ground natural language
in robot state

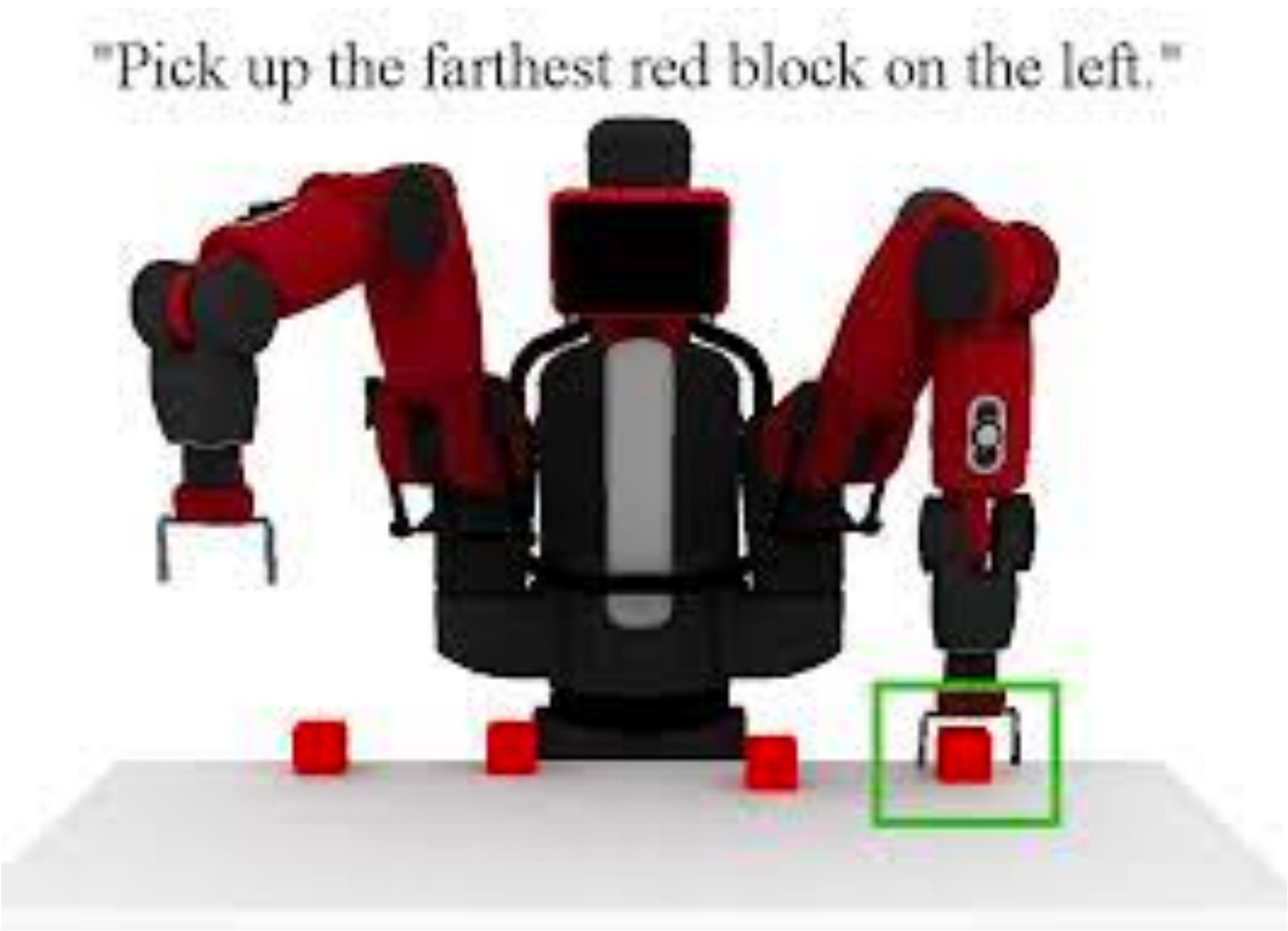


Challenge 2:
Planning actions to
solve a task

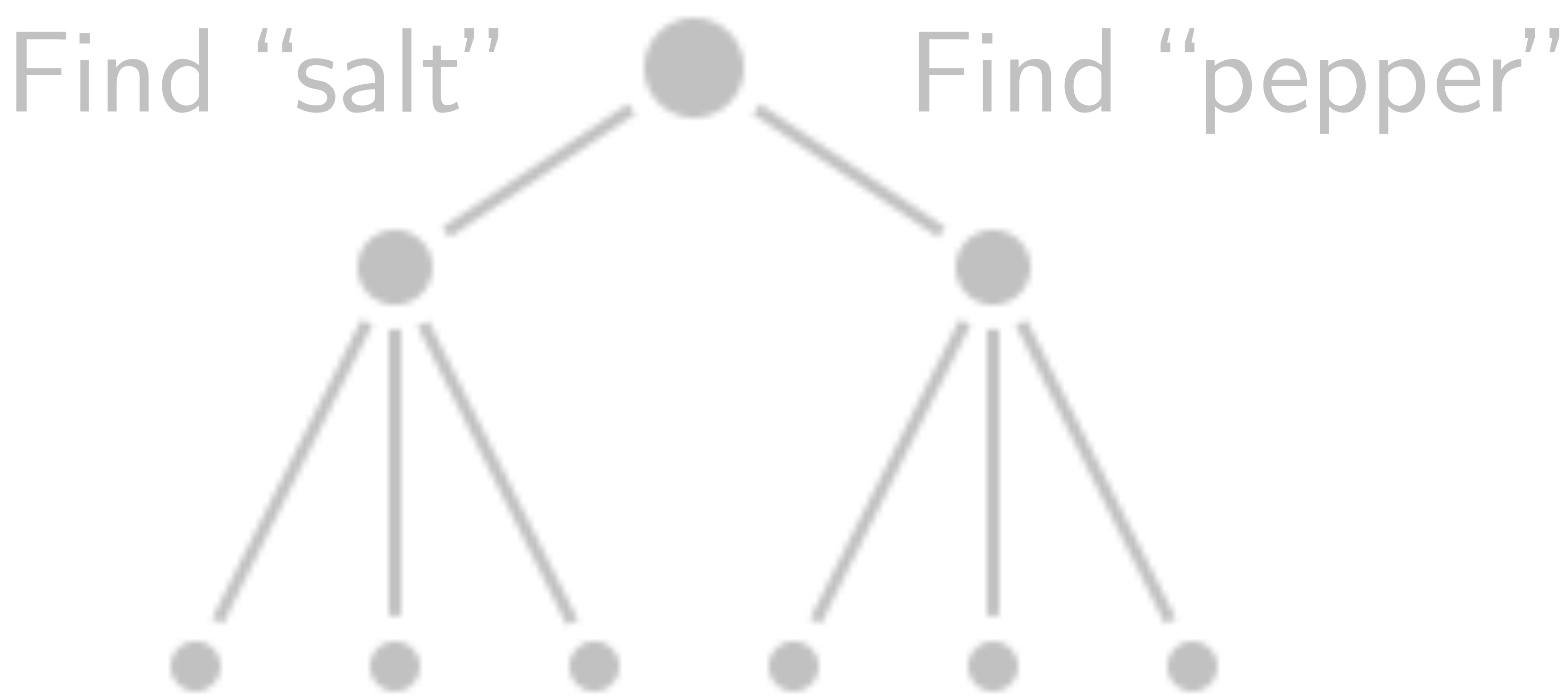


Two Fundamental Challenges

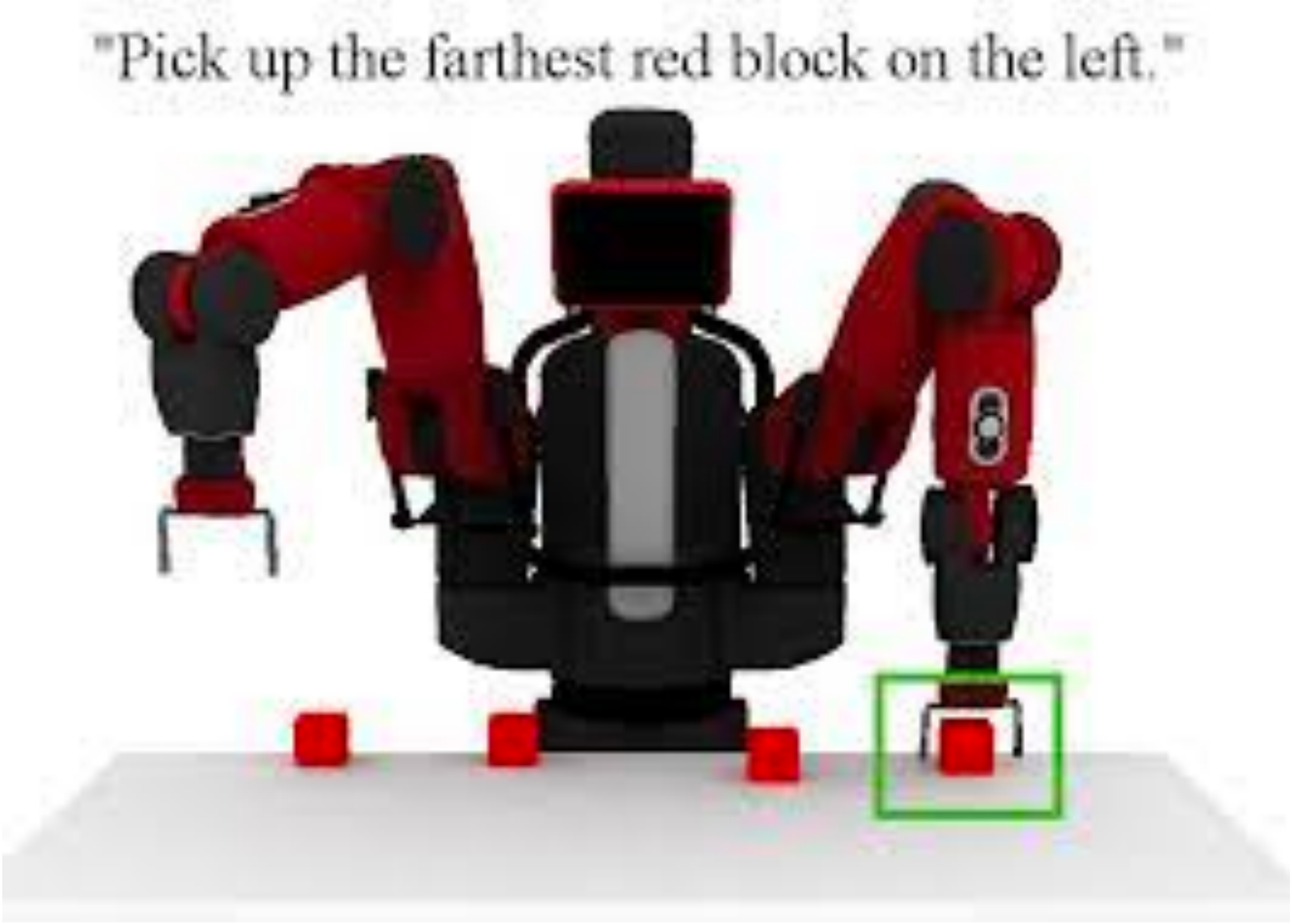
Challenge 1:
Ground natural language
in robot state



Challenge 2:
Planning actions to
solve a task



What is grounding? Why is it hard?

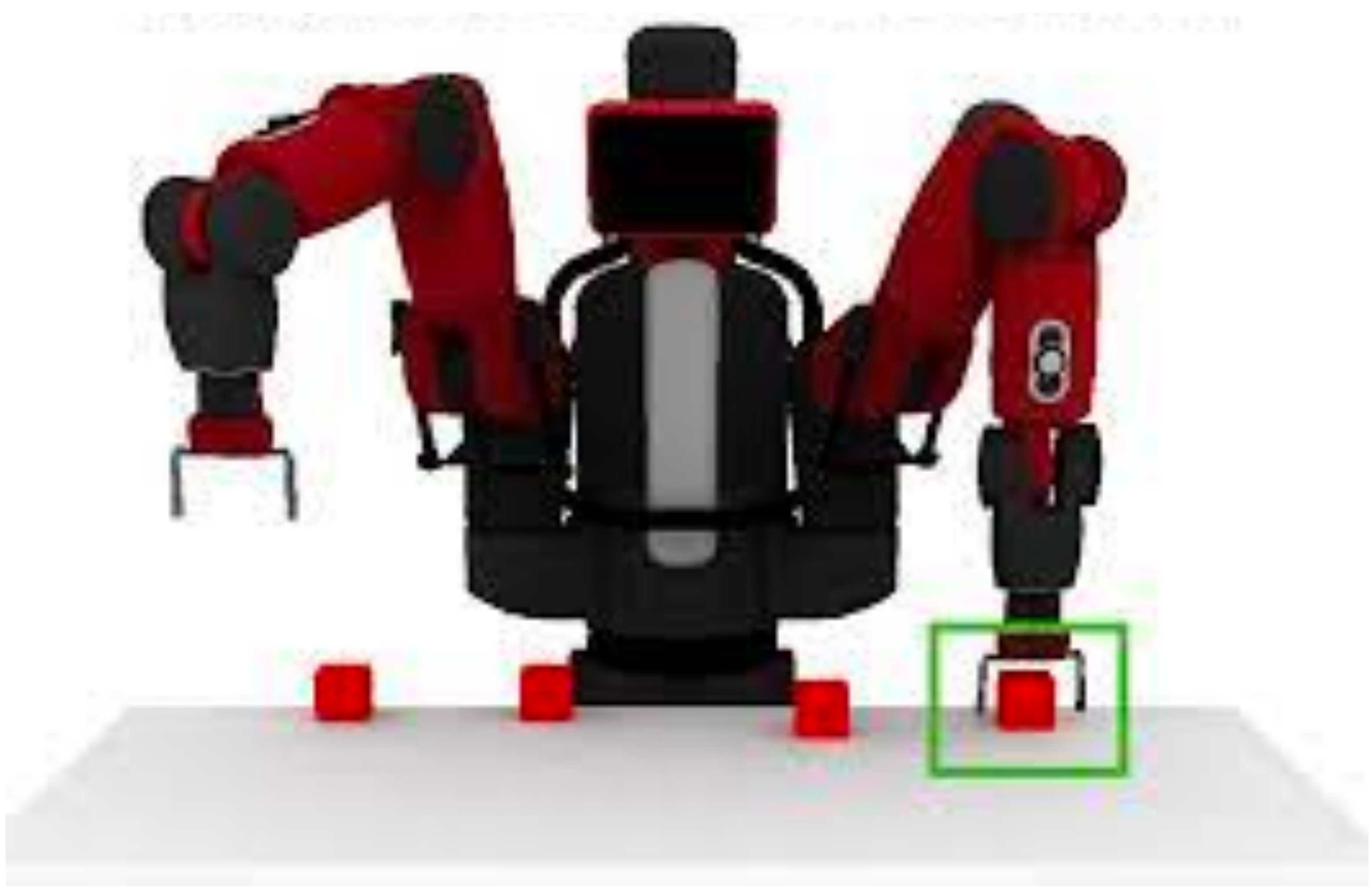


Grounding: Mapping language to robot's internal state

Natural Language → **MDP**

"Pick up the farthest red block"

$$< S \,,\, A \,,\, R \,,\, \mathcal{T} >$$

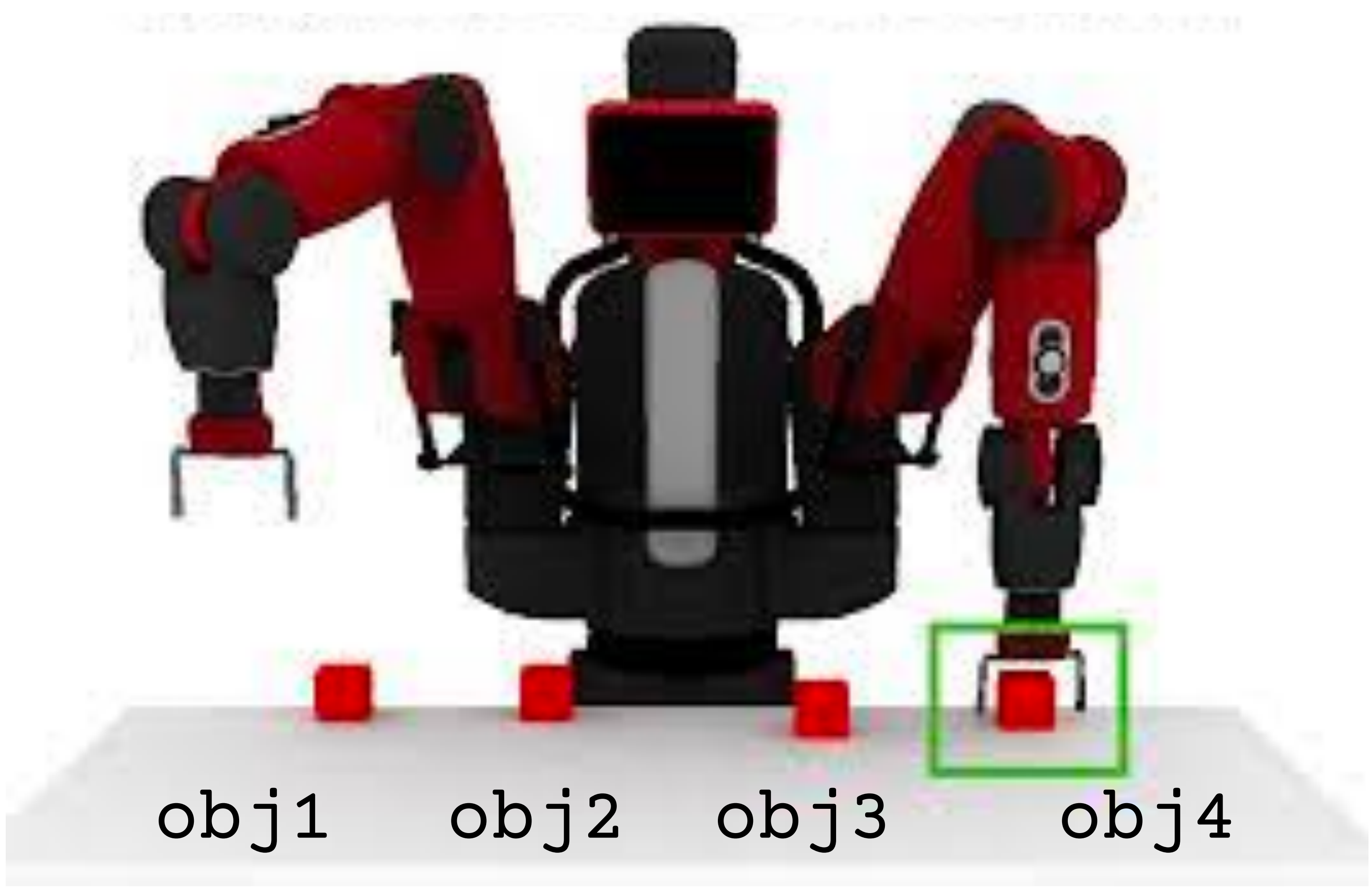


Grounding: Mapping language to robot's internal state

Natural Language → **MDP**

"Pick up the farthest red block"

$$< S , A , R , \mathcal{T} >$$



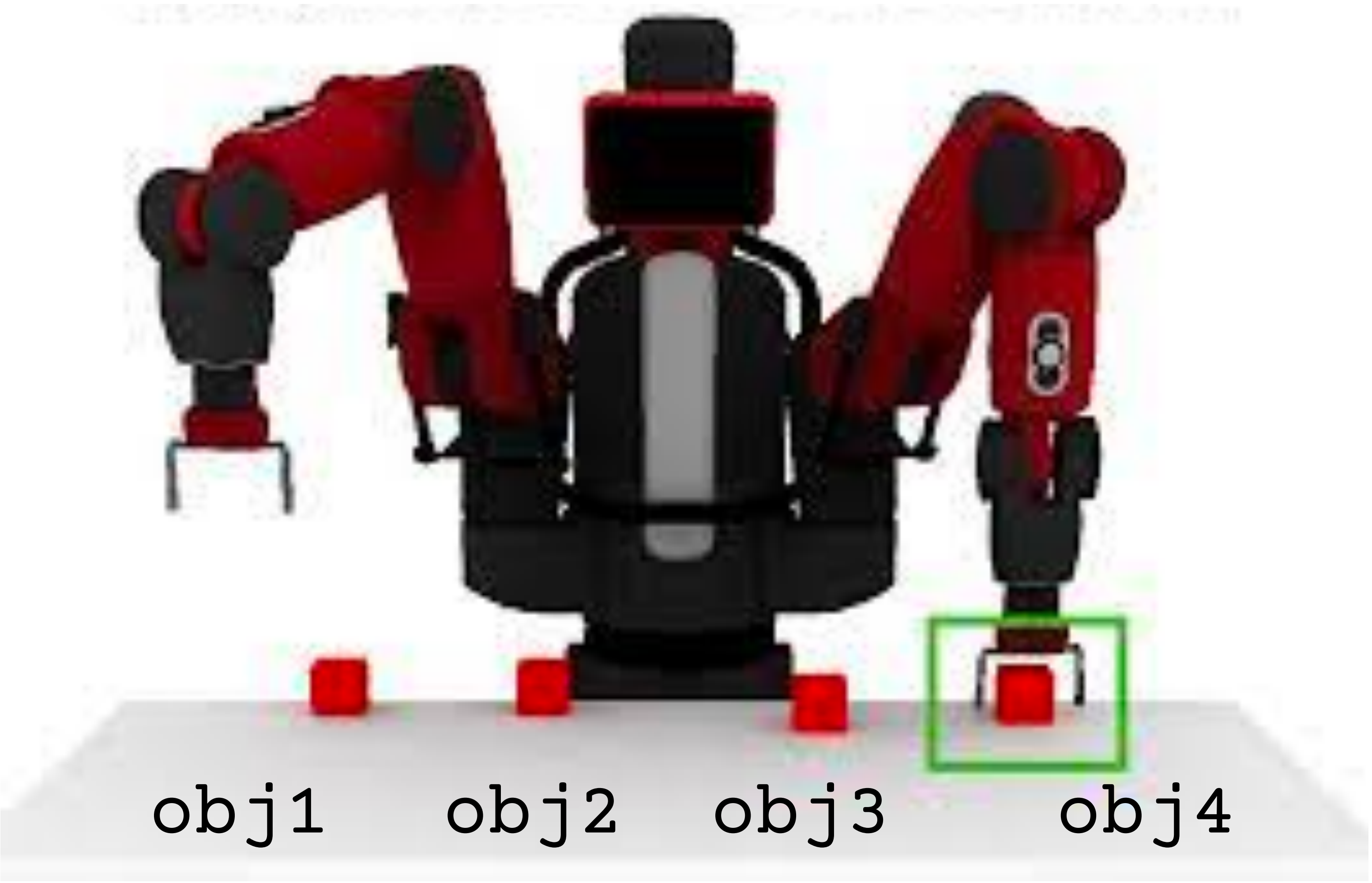

```
on('obj1','table')
on('obj2','table')
on('obj3','table')
on('obj4','table')
left('obj2','obj1')
left('obj3','obj2')
left('obj4','obj3')
...
```

Grounding: Mapping language to robot's internal state

Natural Language → **MDP**

"Pick up the farthest red block"

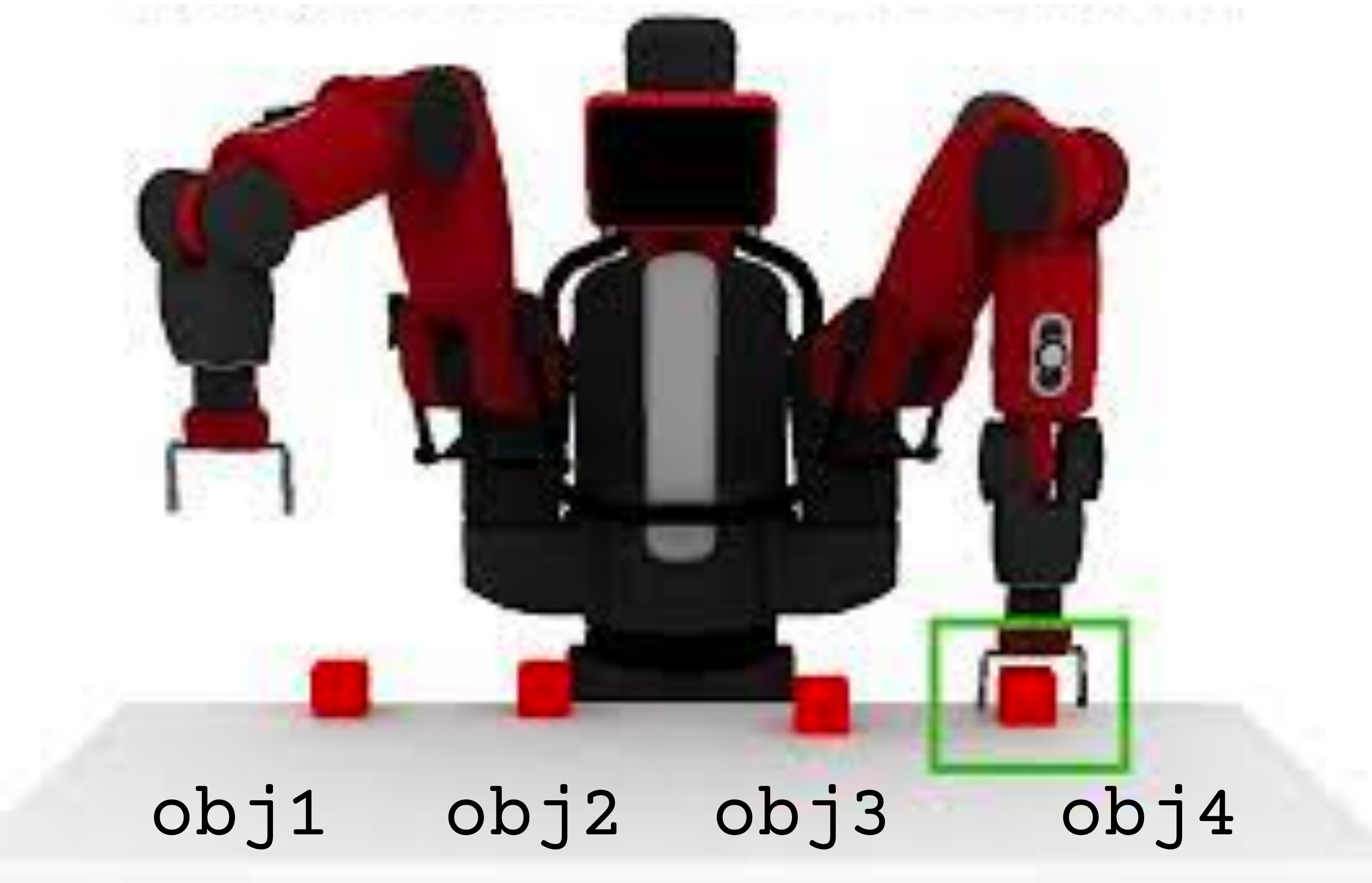
$$< S , A , R , \mathcal{T} >$$



R(`in(obj4, hand)`) $= +1$

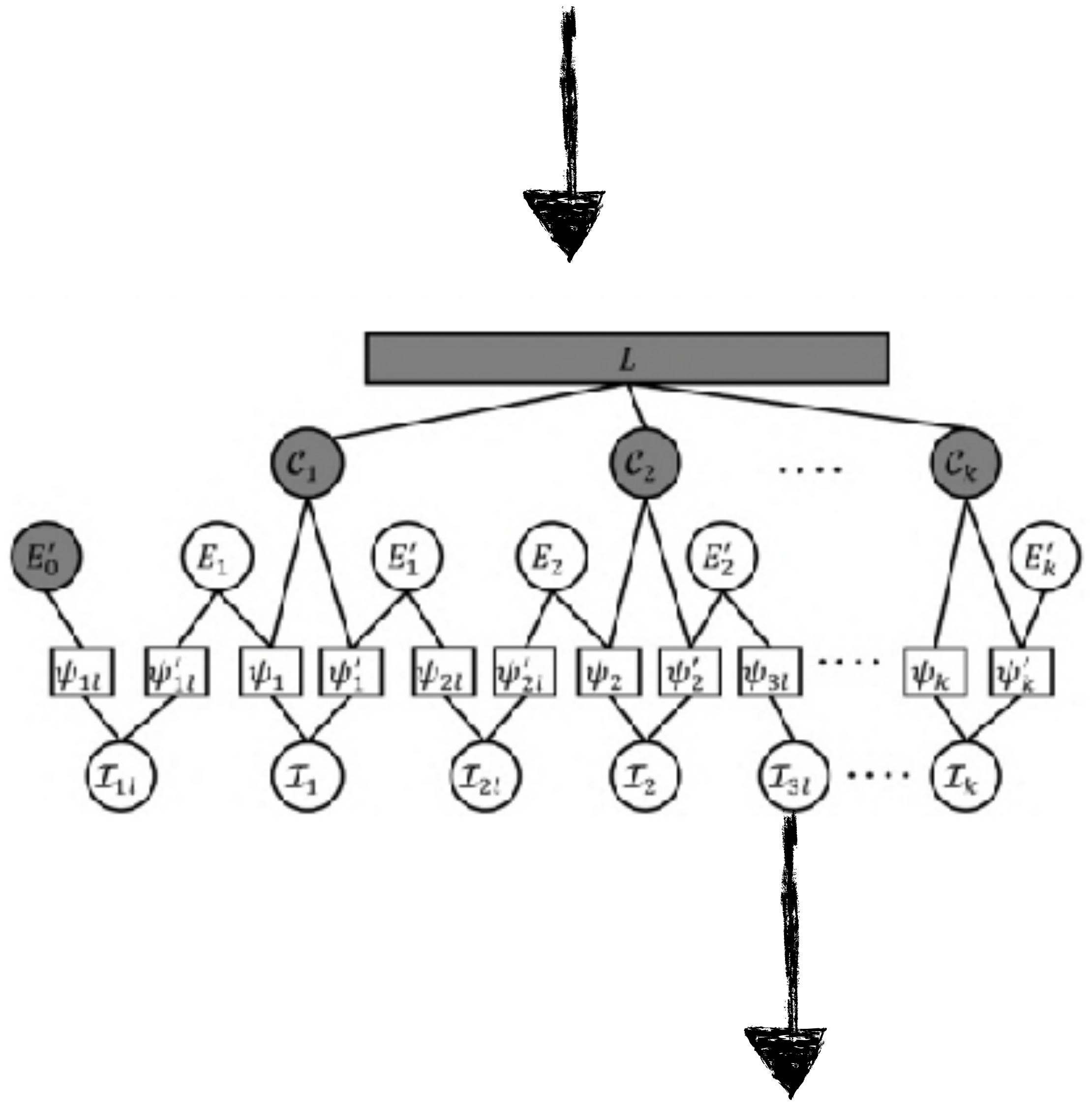
How did we solve grounding?

Train this on small, custom robot datasets!



"Pick up the farthest red block"

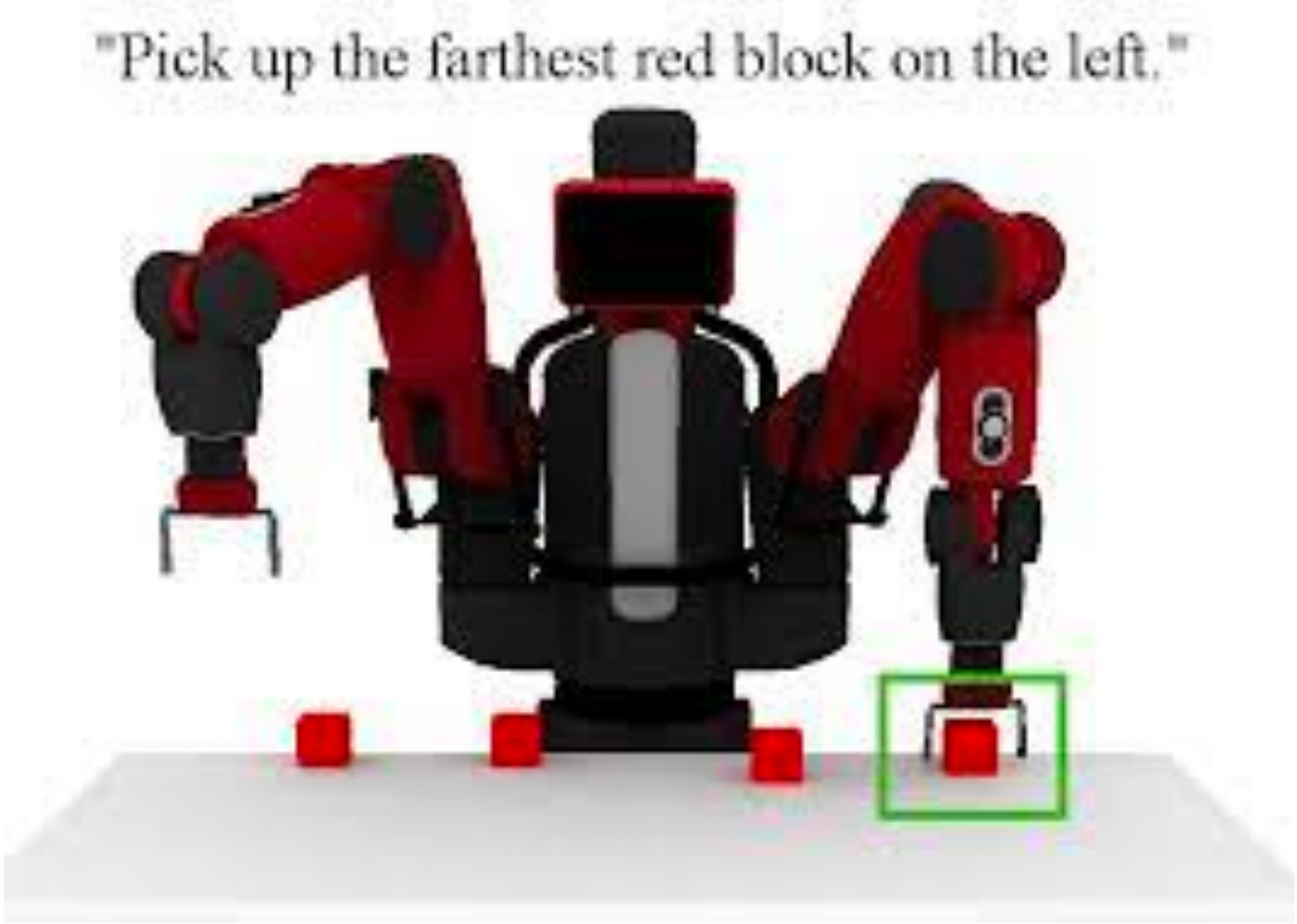
Complex graphical models!



R(`in(obj4, hand)`) $= +1$

Misra et al. Tell me Dave: Context-sensitive grounding of natural language to manipulation instructions

Why did this not scale?



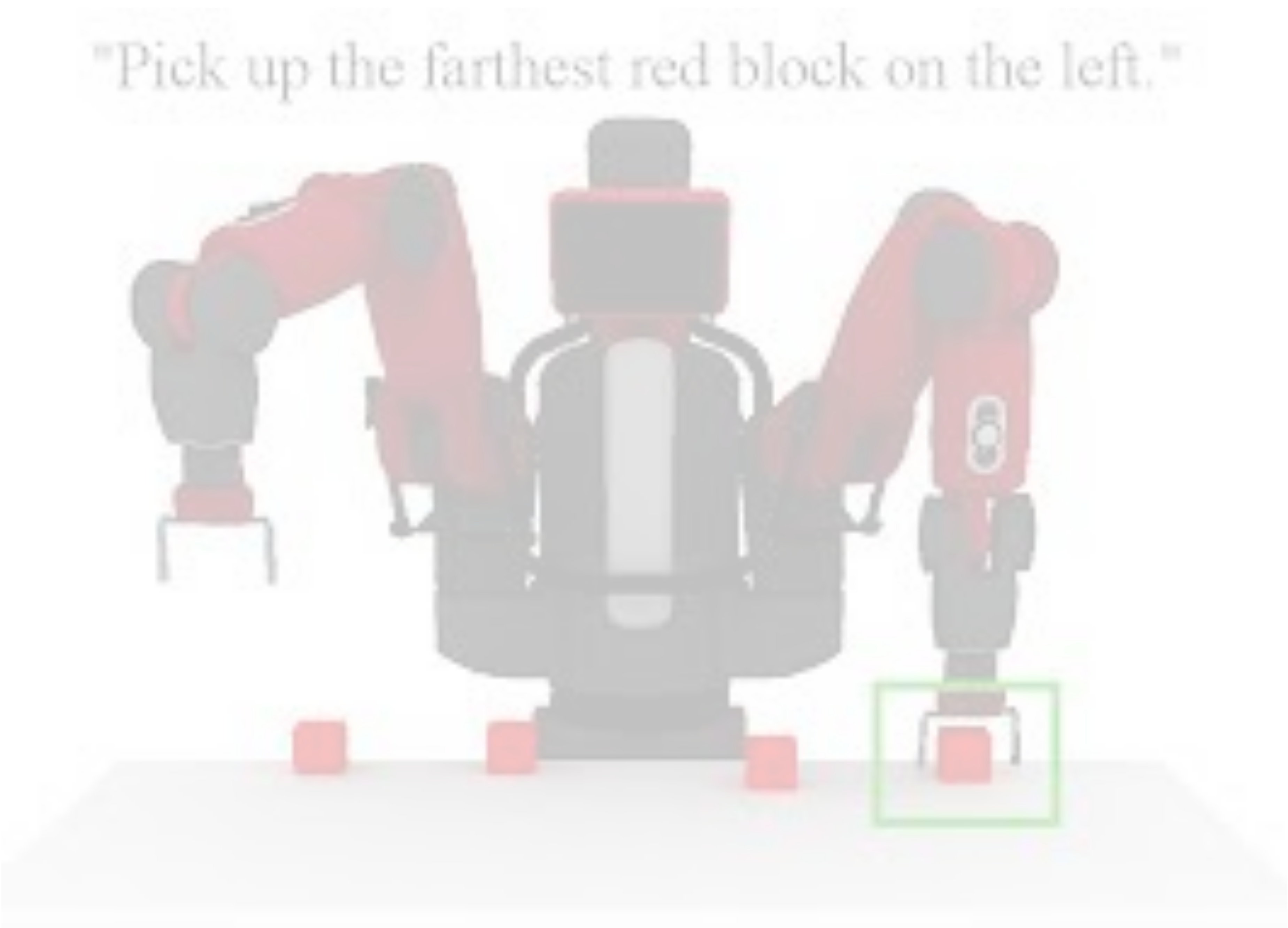
1. Failure to generalize to different human utterances

2. Failure to capture common sense

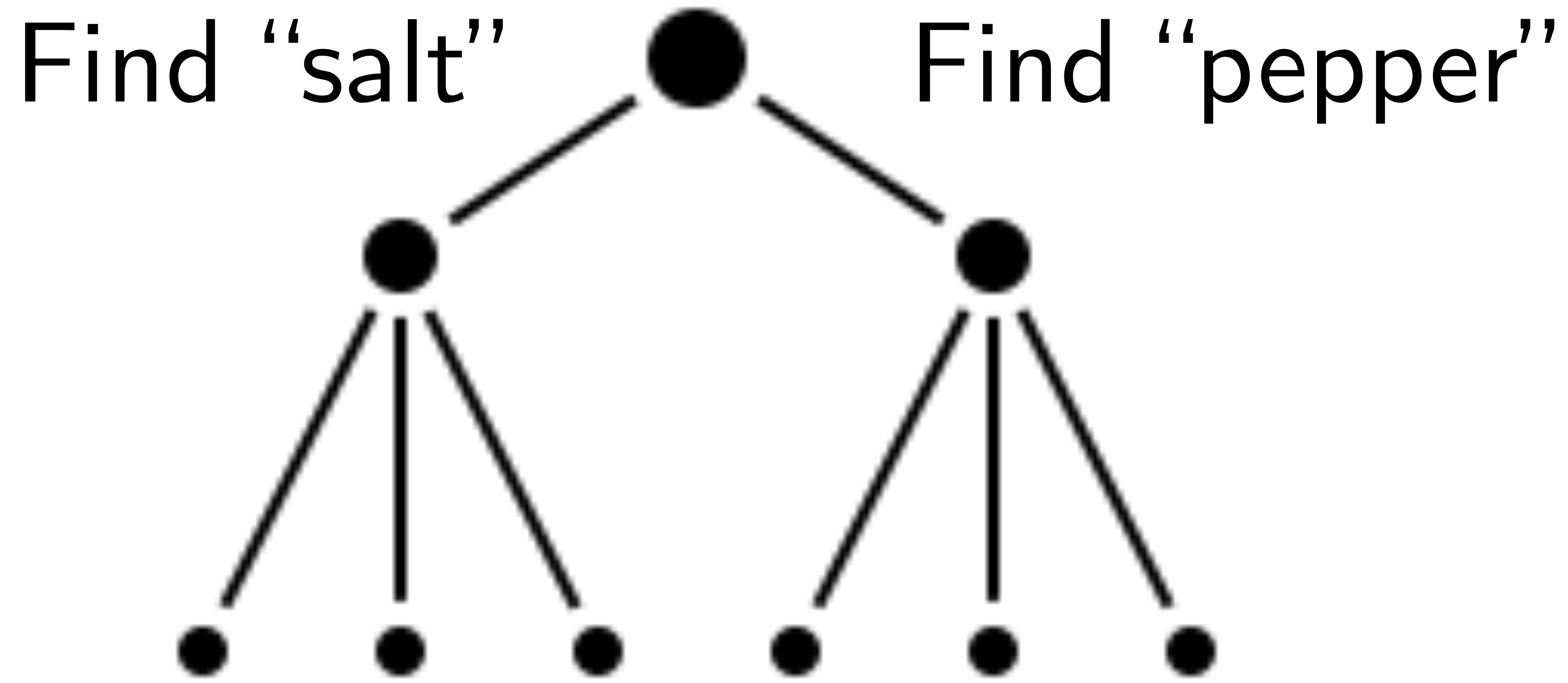
3. Failure to capture complex instructions (while loops)

Two Fundamental Challenges

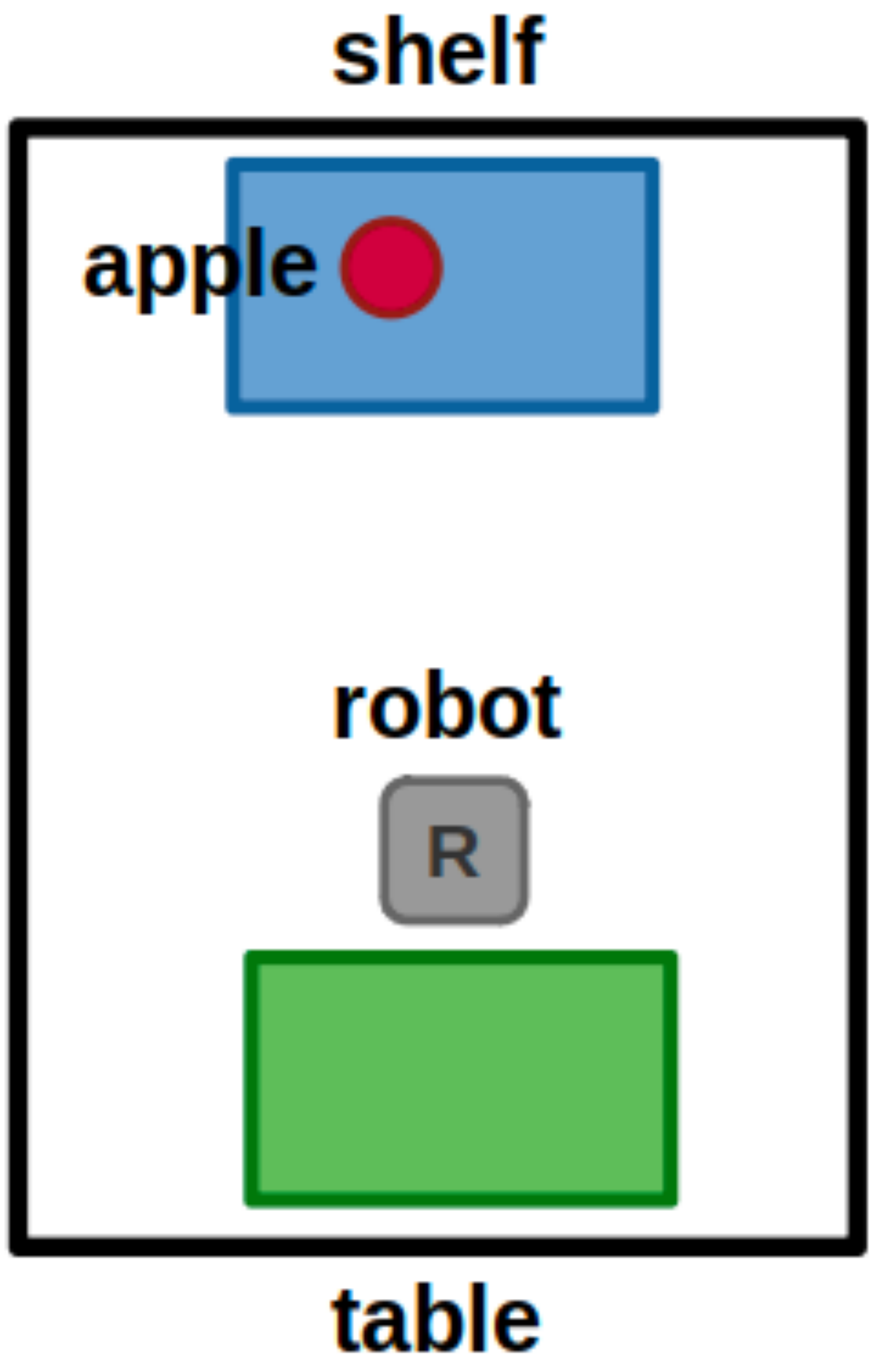
Challenge 1:
Ground natural language
in robot state



Challenge 2:
Planning actions to
solve a task

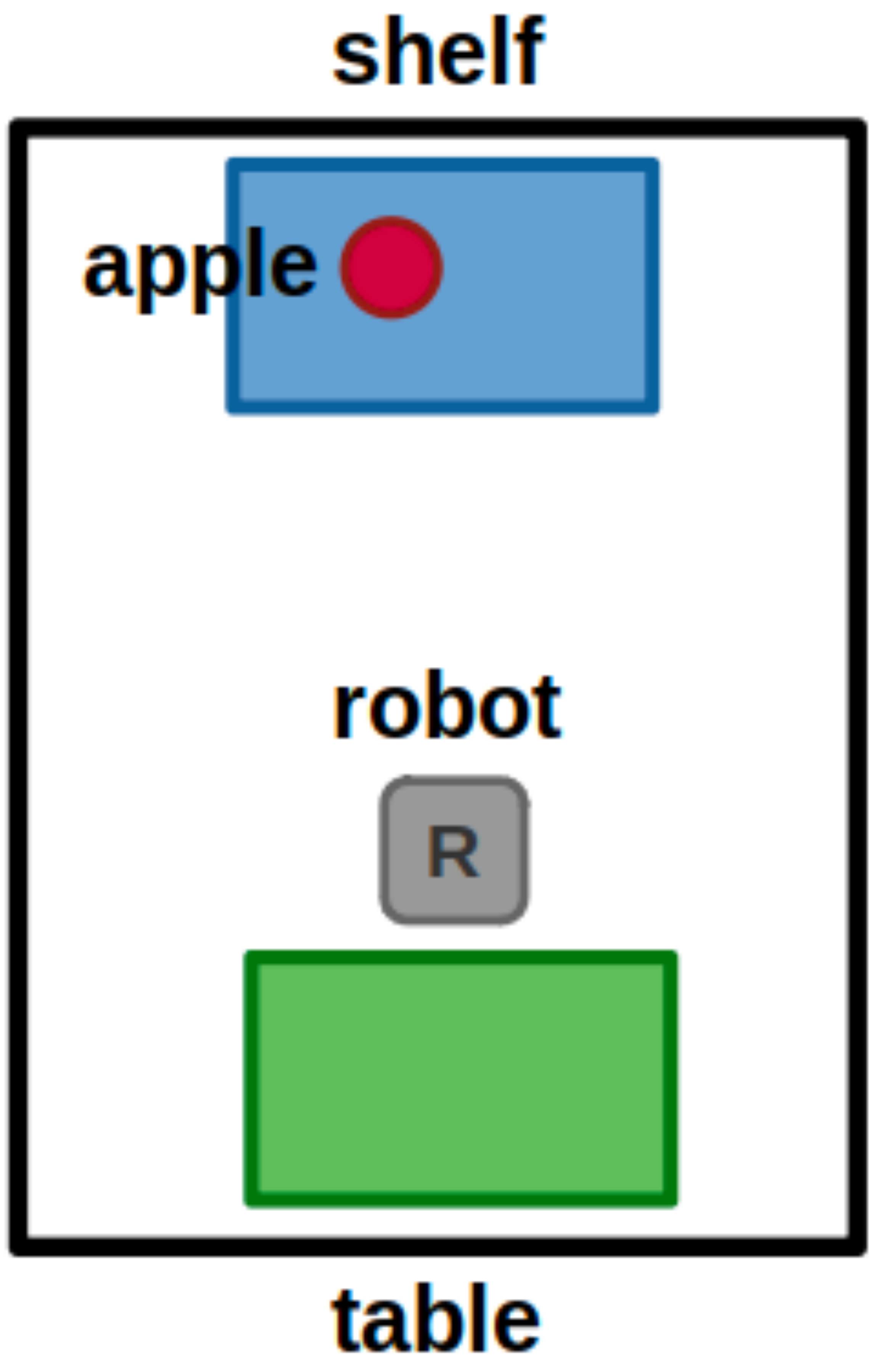


What is task planning? Why is it hard?



Take the apple from the shelf and put it on the table

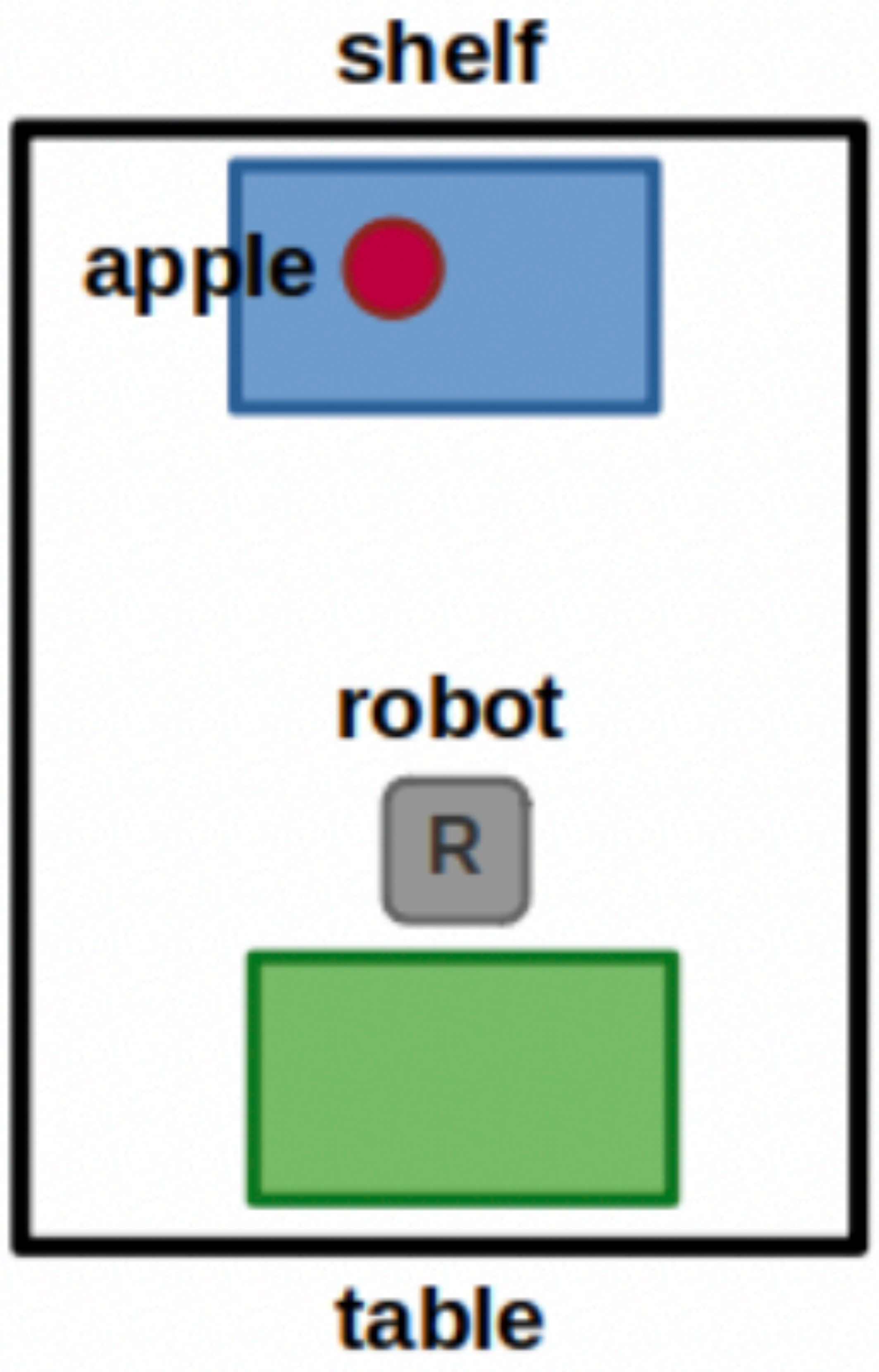
What is task planning? Why is it hard?



Take the apple from the shelf and put it on the table

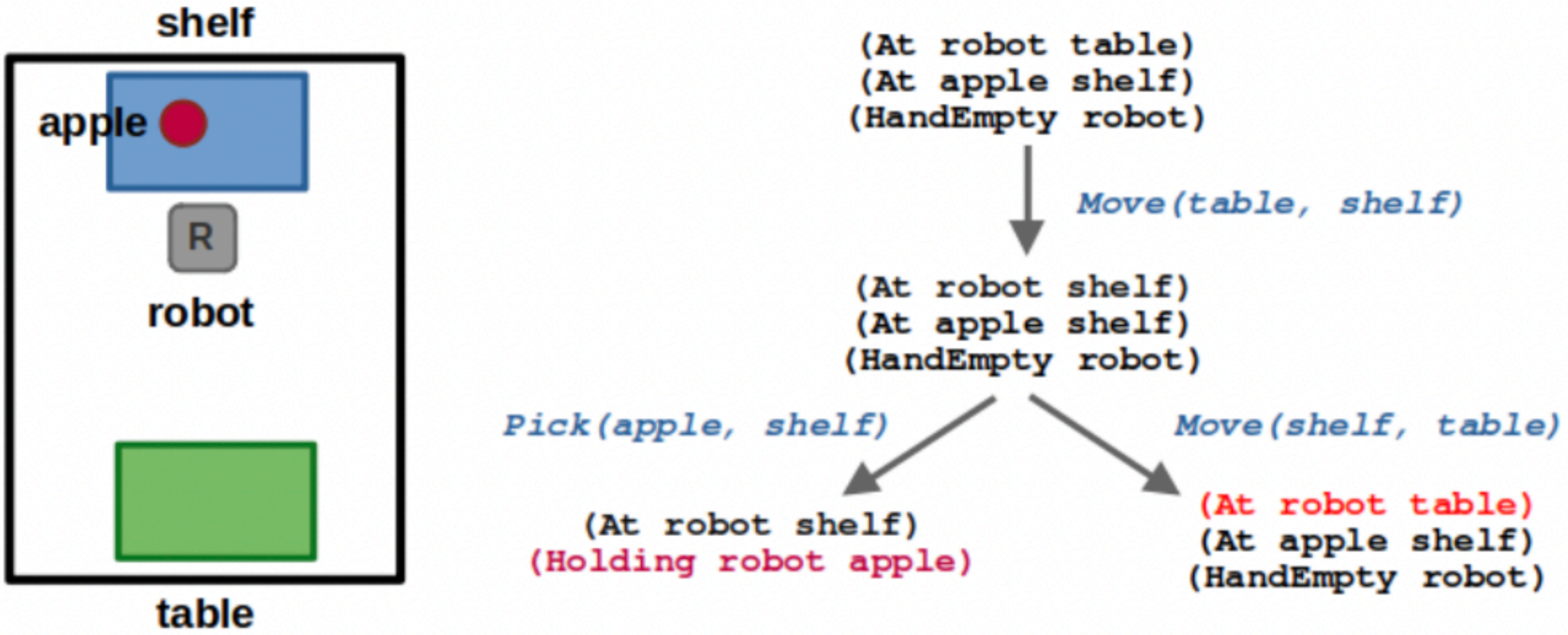
1. Move to the shelf
2. Pick up the apple
3. Move back to the table
4. Place the apple

What is task planning? Why is it hard?

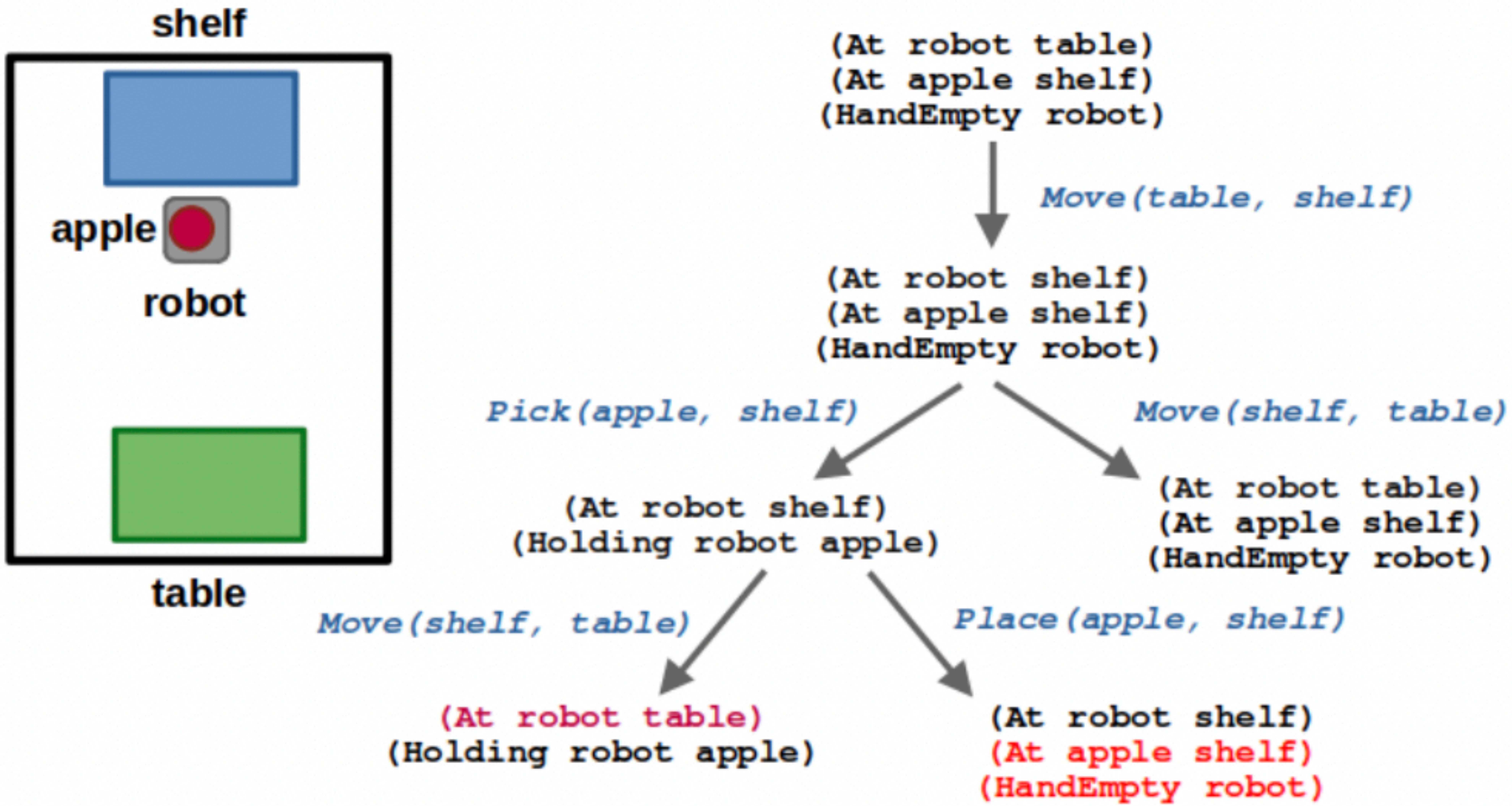



```
(At robot table)
(At apple shelf)
(HandEmpty robot)
        |
        |   Move(table, shelf)
        v
(At robot shelf)
(At apple shelf)
(HandEmpty robot)
```

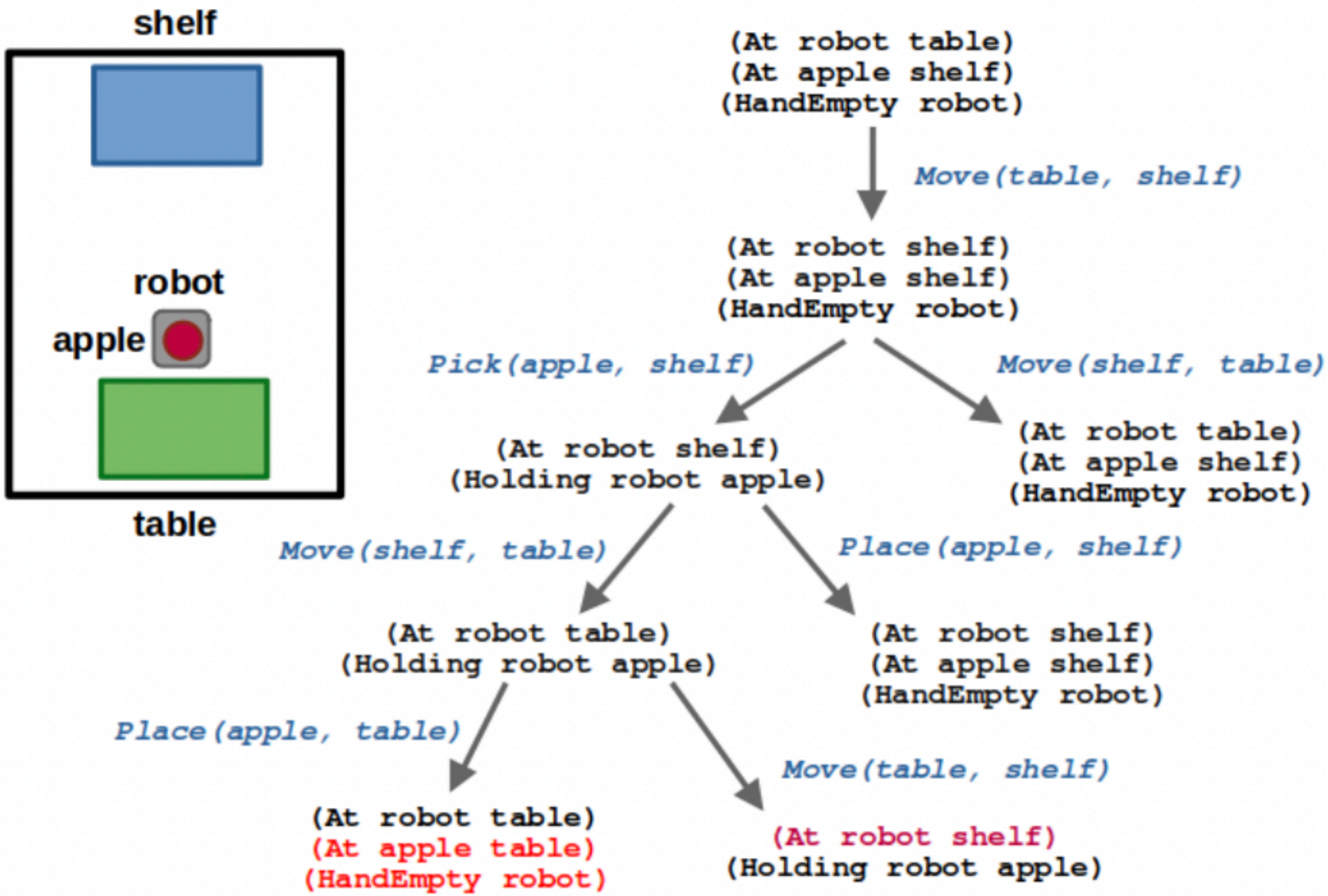
What is task planning? Why is it hard?



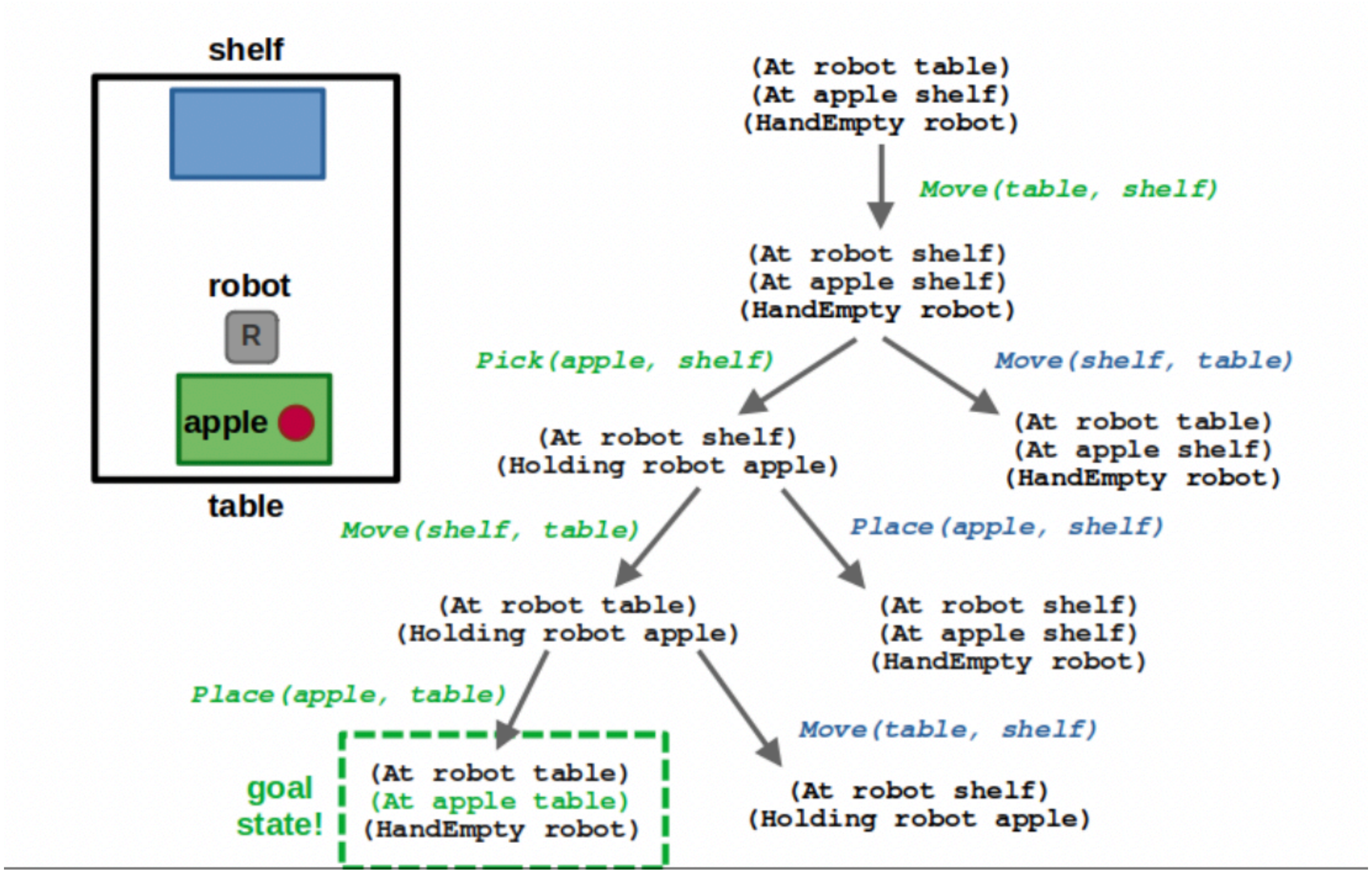
What is task planning? Why is it hard?



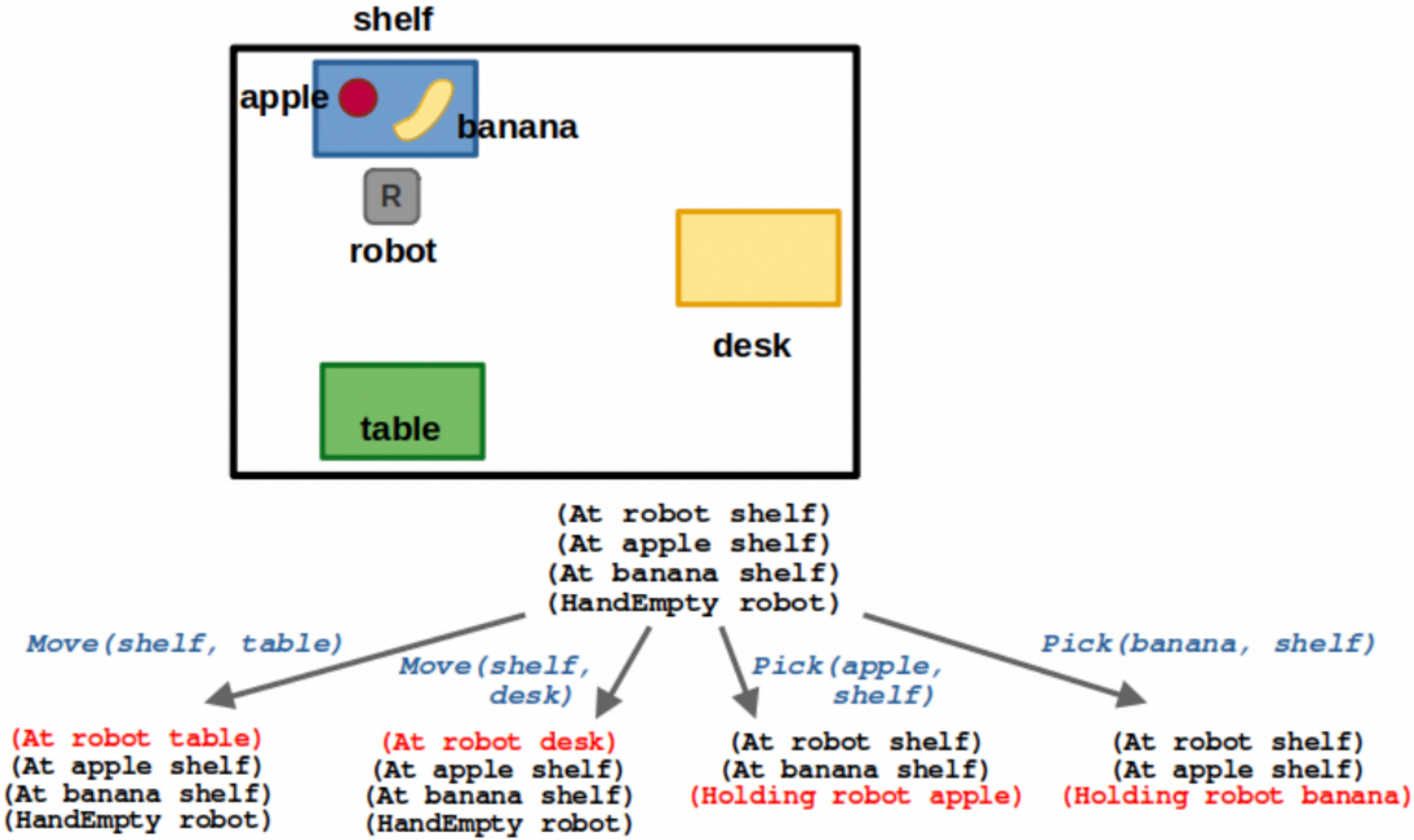
What is task planning? Why is it hard?



What is task planning? Why is it hard?



What is task planning? Why is it hard?



How did we solve it?

Good old fashioned search!

Lots of heuristics to make it real time

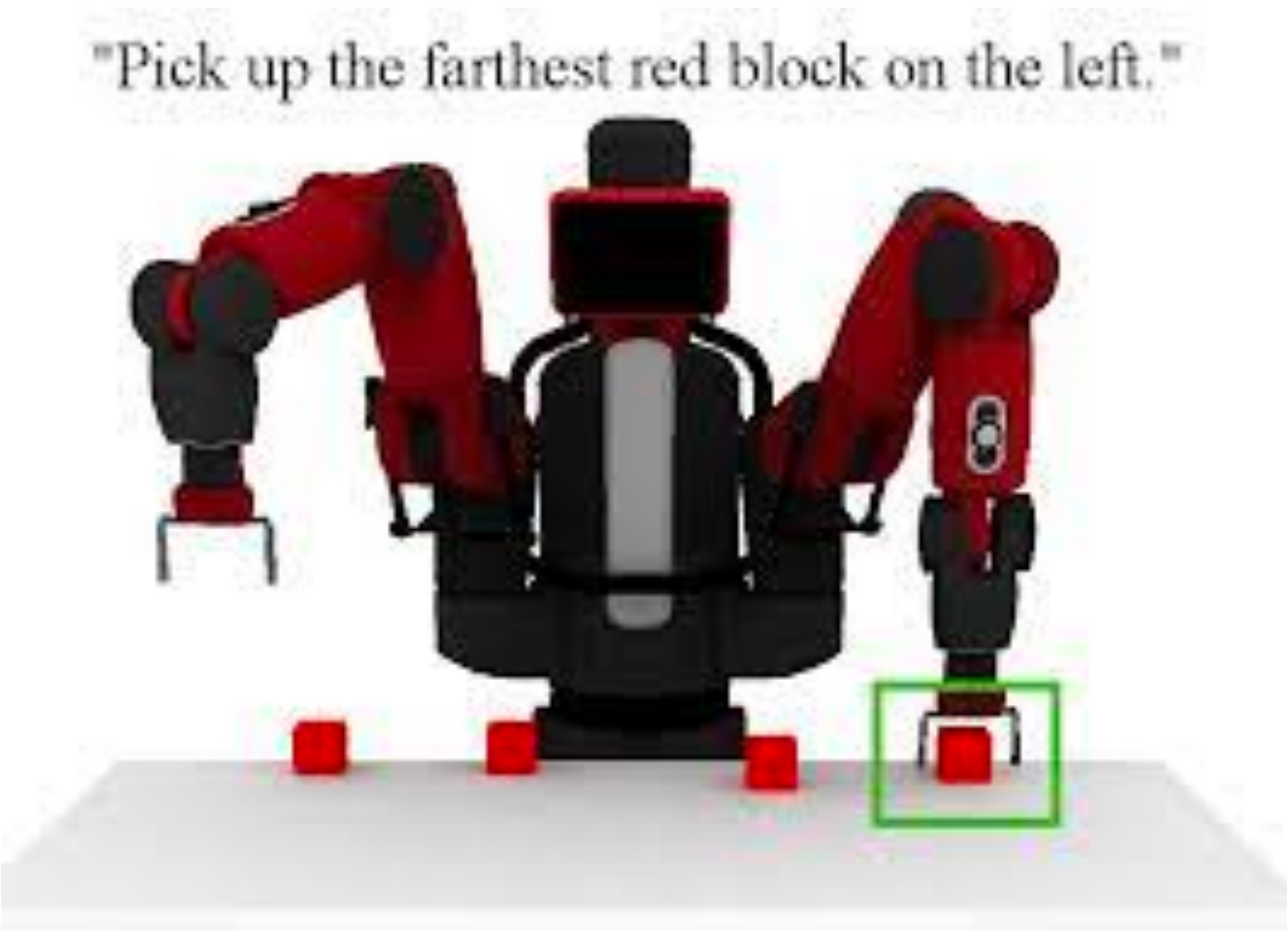
Why did it not scale?

Combinatorially large search tree

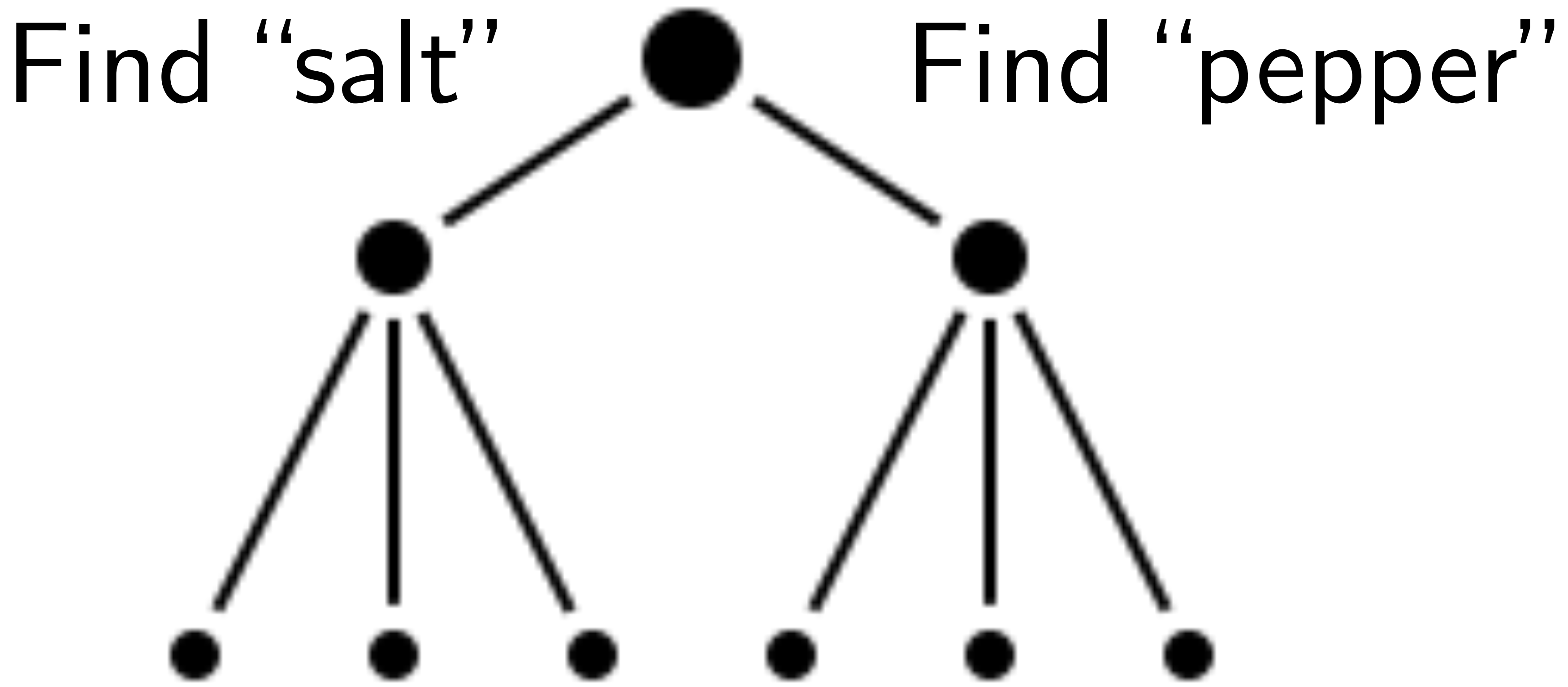
Had no notion of common sense

Two Fundamental Challenges

Challenge 1:
Ground natural language in robot state



Challenge 2:
Planning actions to solve a task



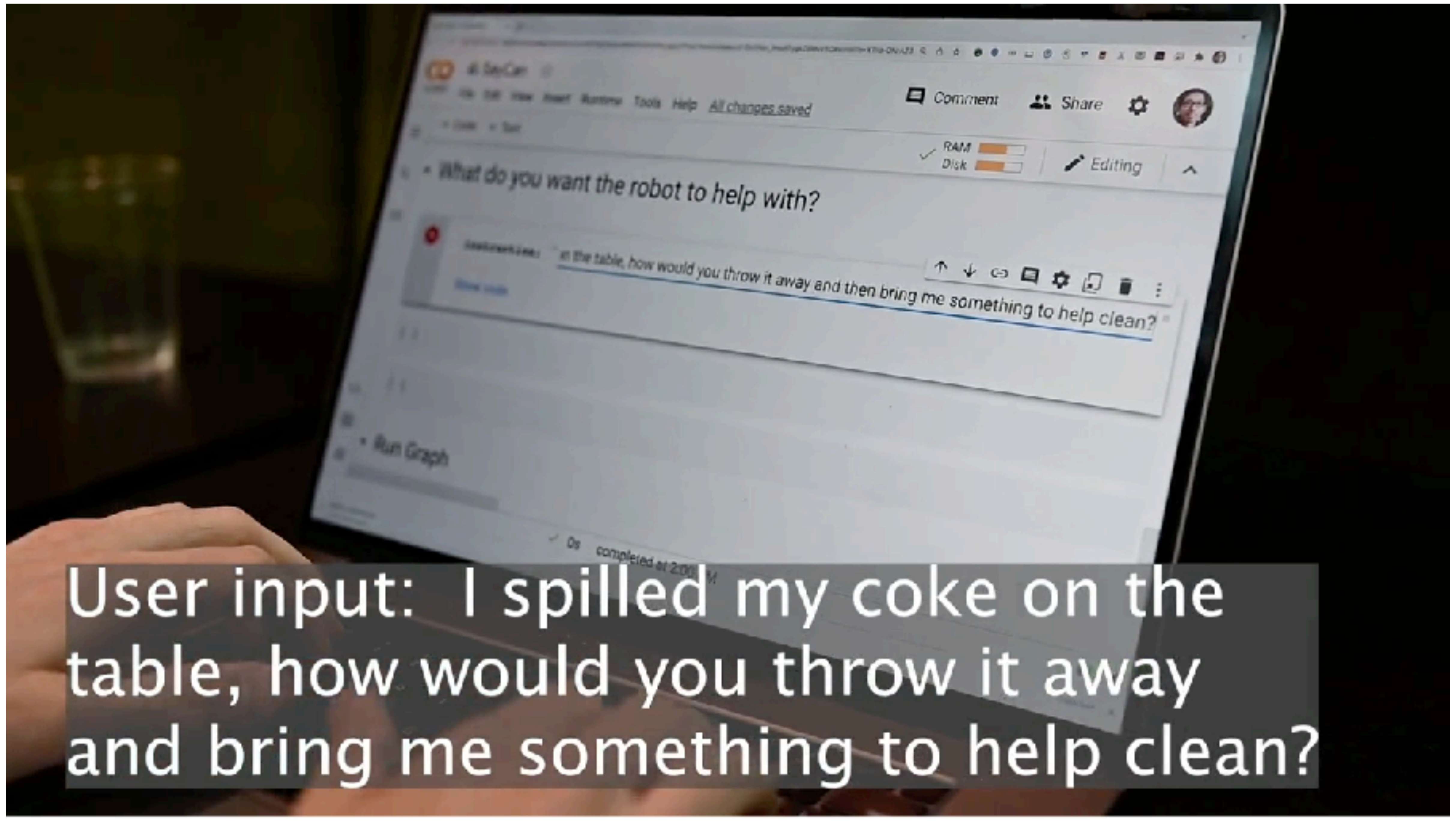
LARGE LANGUAGE MODELS

Episode IV

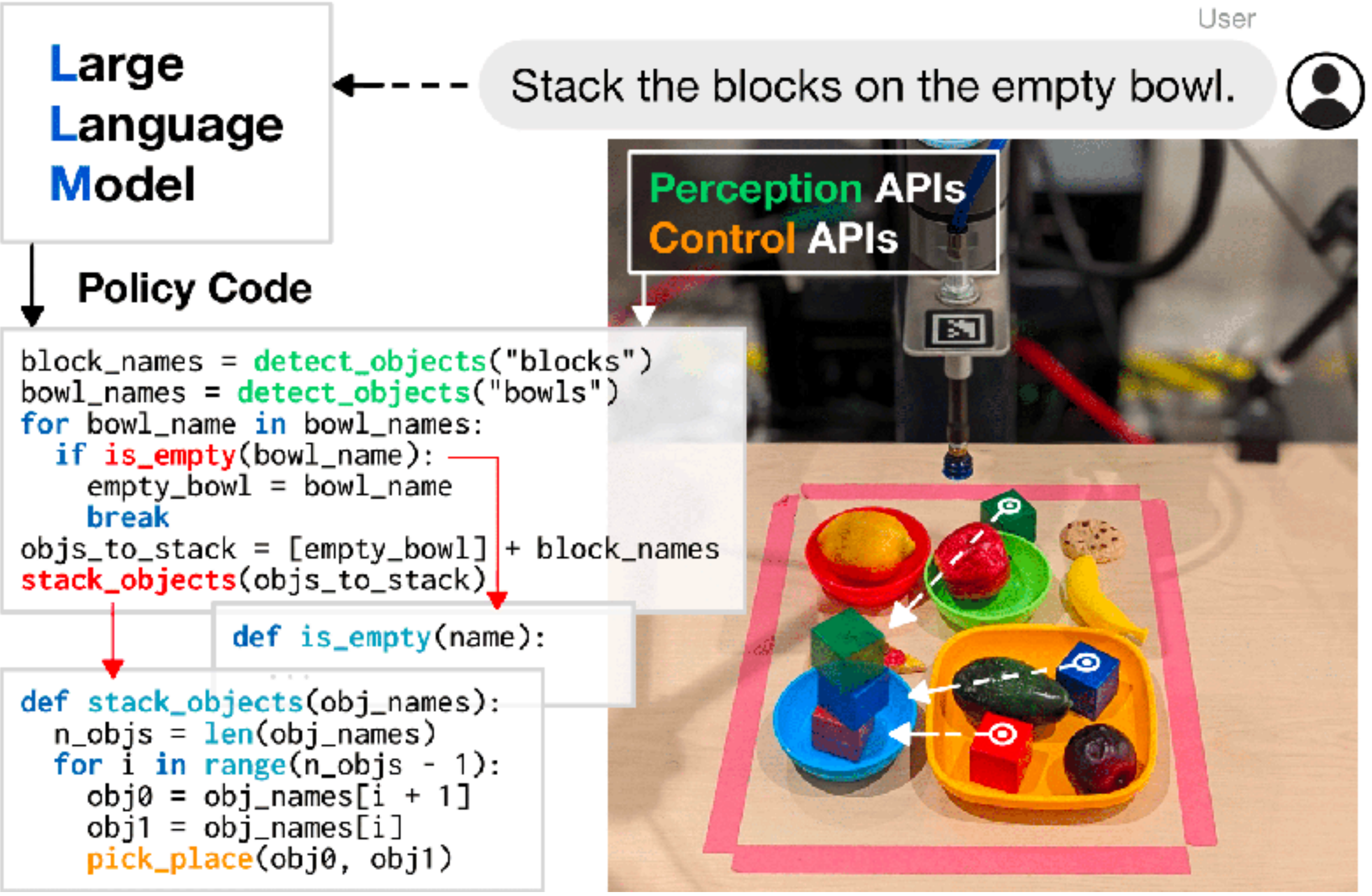
A NEW HOPE

Many recent papers on LLM+Task Planning

SayCan [Ichter et al.'22]



Code-As-Policies [Liang et al.'22]



Also ProgPrompt [Singh et al. '22], InnerMonologue [Huang et al.'22], Socratic [Zeng et al.'22], TidyBot [Wu et al'23], CLARIFY [Skreta et al.'23], Text2Motion [Lin et al. '23], ...

Can LLMs directly predict robot action?

Do As I Can, Not As I Say:

Grounding Language in Robotic Affordances

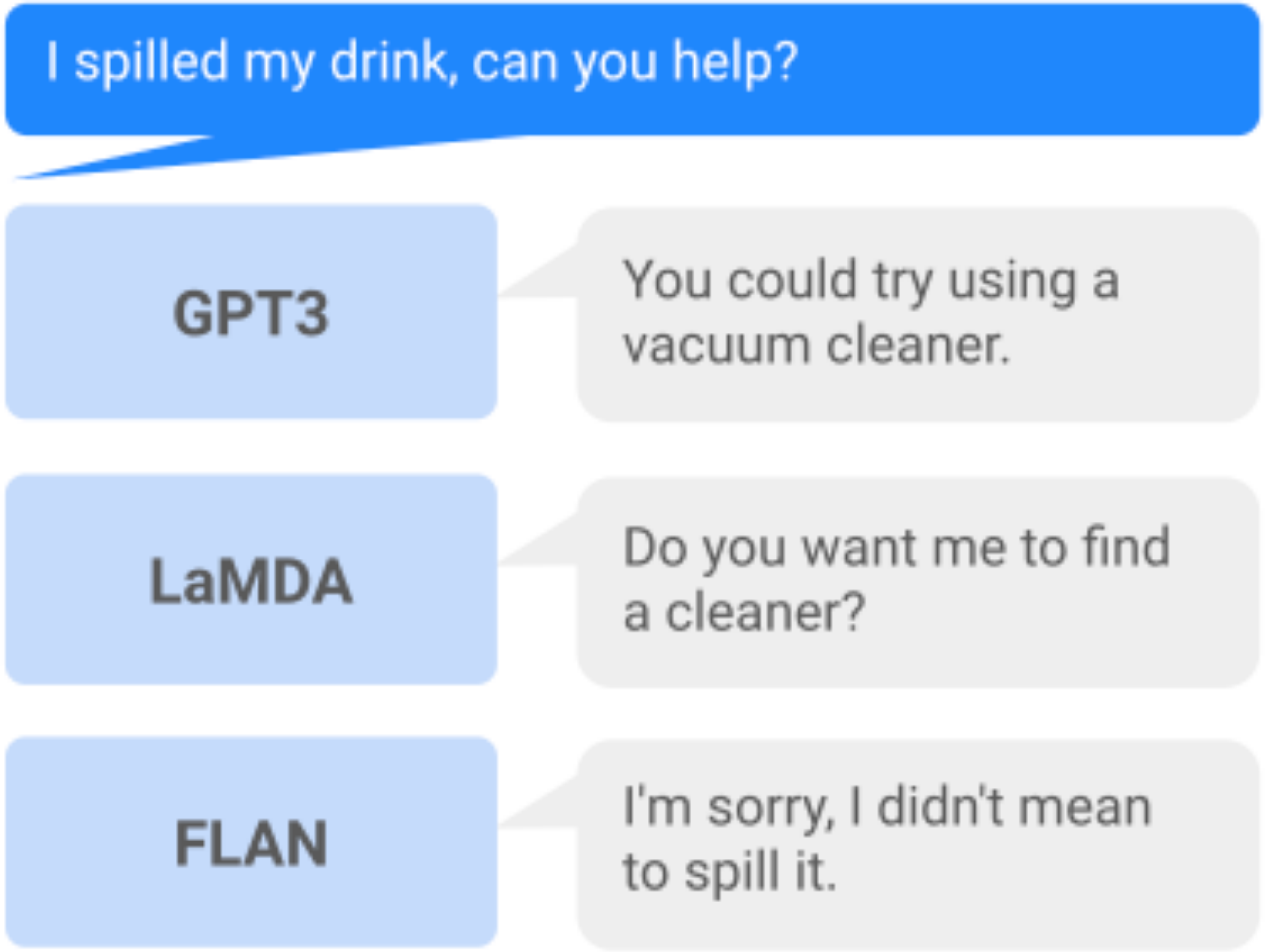
Michael Ahn* Anthony Brohan* Noah Brown* Yevgen Chebotar* Omar Cortes* Byron David* Chelsea Finn*
Chuyuan Fu* Keerthana Gopalakrishnan* Karol Hausman* Alex Herzog* Daniel Ho* Jasmine Hsu* Julian Ibarz*
Brian Ichter* Alex Irpan* Eric Jang* Rosario Jauregui Ruano* Kyle Jeffrey* Sally Jesmonth* Nikhil Joshi*
Ryan Julian* Dmitry Kalashnikov* Yuheng Kuang* Kuang-Huei Lee* Sergey Levine* Yao Lu* Linda Luu* Carolina Parada*
Peter Pastor* Jornell Quiambao* Kanishka Rao* Jarek Rettinghouse* Diego Reyes* Pierre Sermanet* Nicolas Sievers*
Clayton Tan* Alexander Toshev* Vincent Vanhoucke* Fei Xia* Ted Xiao* Peng Xu* Sichun Xu* Mengyuan Yan* Andy Zeng*

Robotics at Google Everyday Robots

I spilled my coke on the table, how would you
throw it away and bring me something to help clean?
Robot: I would: 1. find a coke can, 2. ___

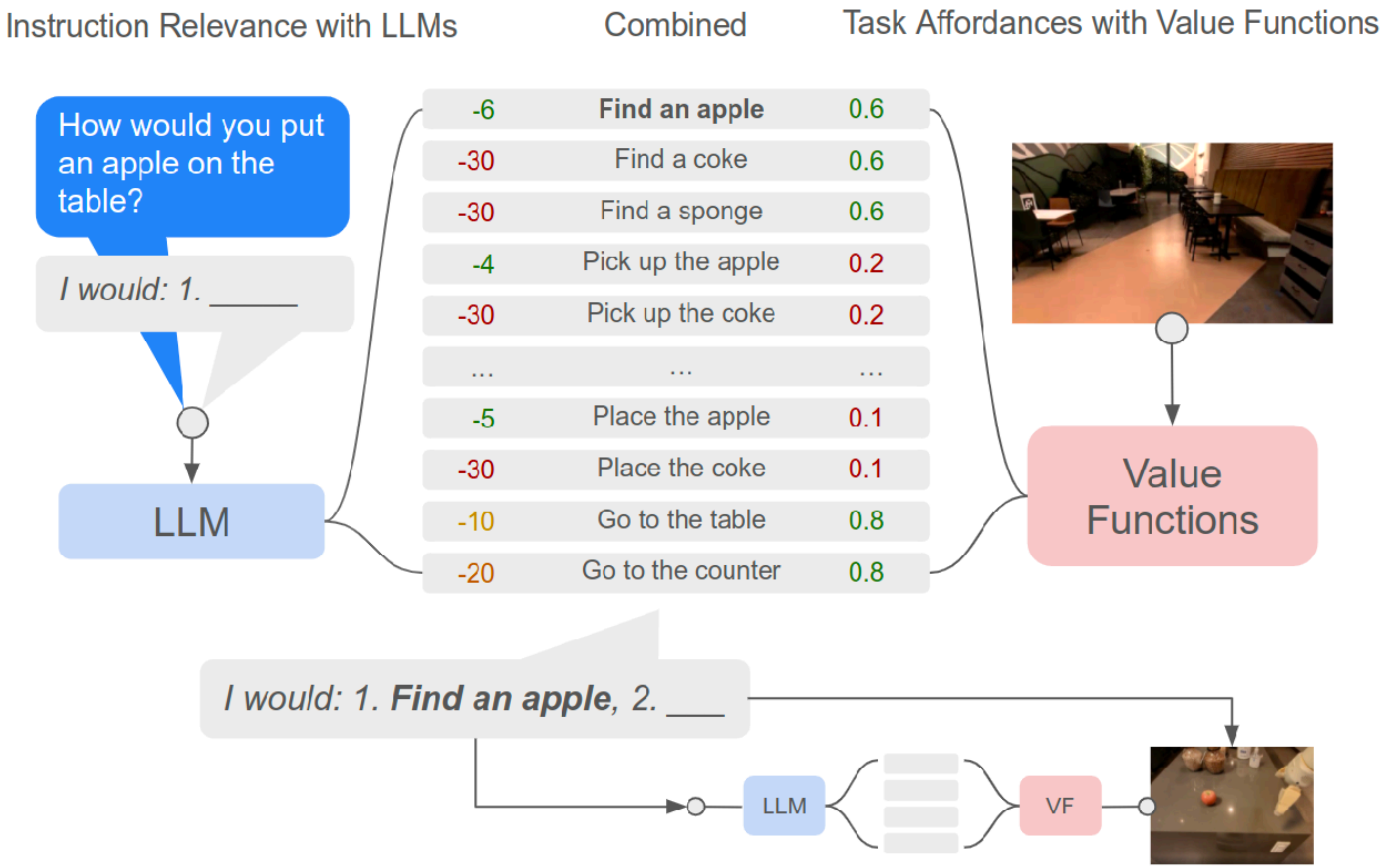
So ... we just ask an LLM to tell us what to do?

No! LLMs can say *anything* ..



Idea: Constrain LLM by what the robot can do (affordance)

The “SayCan” Approach




10x speed
COMPOSTABLES
CANS & BOTTLES


6x speed
48
Language
× Affordance
Combined Score
find a grapefruit soda
1.00
find a lime soda
0.01
find a redbull
0.00
find a 7up
0.00
go to the table
0.00
User input: Bring me a fruit flavoured drink without caffeine.
Robot: 1.

Can LLMs predict robot code?

Code as Policies:

Language Model Programs for Embodied Control

Jacky Liang Wenlong Huang Fei Xia Peng Xu Karol Hausman Brian Ichter Pete Florence Andy Zeng

Robotics at Google

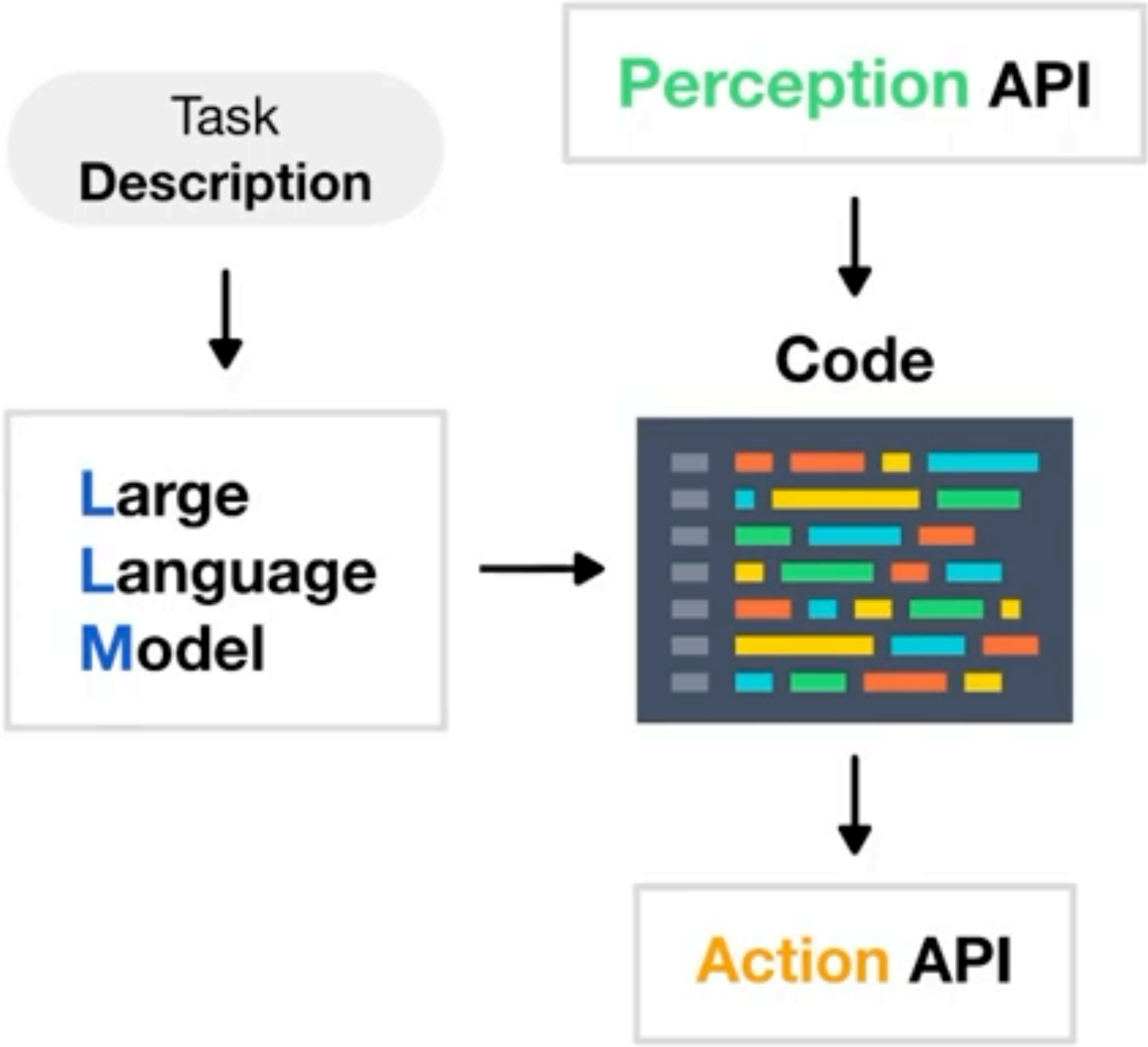
Different policy representations

Why choose code as a representation?

Interpretable

Verifiable

Composable



Ours: Use LLMs to write robot code

Large Language Model

User

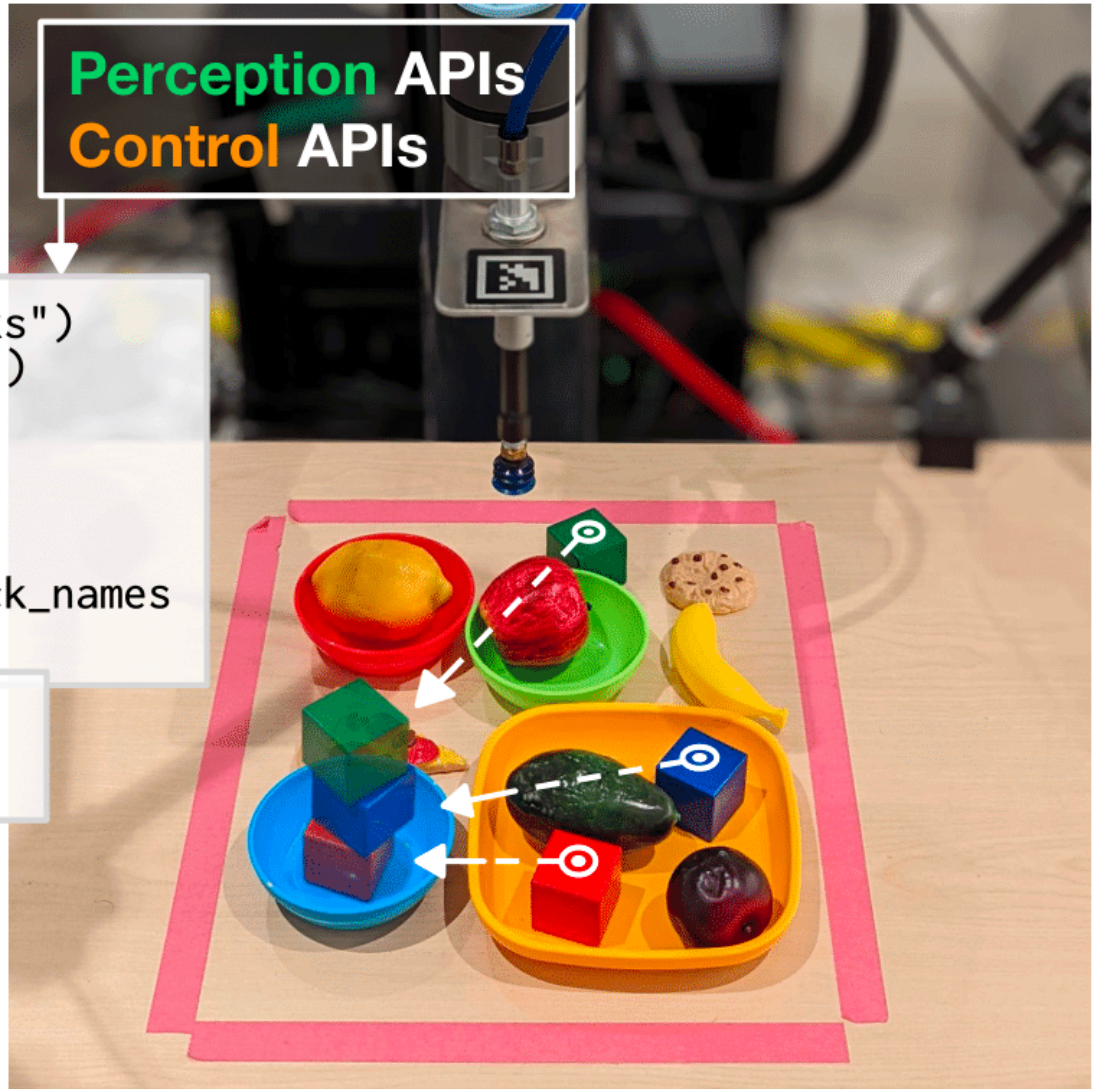
Stack the blocks on the empty bowl.

Policy Code

```
block_names = detect_objects("blocks")
bowl_names = detect_objects("bowls")
for bowl_name in bowl_names:
  if is_empty(bowl_name):
    empty_bowl = bowl_name
    break
objs_to_stack = [empty_bowl] + block_names
stack_objects(objs_to_stack)
```

```
def is_empty(name):
  ...
```

```
def stack_objects(obj_names):
  n_objs = len(obj_names)
  for i in range(n_objs - 1):
    obj0 = obj_names[i + 1]
    obj1 = obj_names[i]
    pick_place(obj0, obj1)
```



Simple code generation examples

```
# if you see an orange, move backwards.
if detect_object("orange"):
    robot.set_velocity(x=-0.1, y=0, z=0)
# move rightwards until you see the apple.
while not detect_object("apple"):
    robot.set_velocity(x=0, y=0.1, z=0)
```

```
# do it again but faster, to the left, and with a banana.
while not detect_object("banana"):
    robot.set_velocity(x=0, y=-0.2, z=0)
```

How do we prompt LLMs to generate robot code?

1. Instructions

   You are an AI assistant writing robot code given natural language instructions. Please refer to the following API guidelines ...

2. Import Hints

```
import numpy as np
from utils import get_obj_names, put_first_on_second
```

3. Few-shot Examples

Example: Using imported functions

```
from utils import get_pos, put_first_on_second
...
# move the purple bowl toward the left.
target_pos = get_pos('purple bowl') + [-0.3, 0]
put_first_on_second('purple bowl', target_pos)
objs = ['blue bowl', 'red block', 'red bowl', 'blue block']
# move the red block a bit to the right.
target_pos = get_pos('red block') + [0.1, 0]
put_first_on_second('red block', target_pos)
# put the blue block on the bowl with the same color.
put_first_on_second('blue block', 'blue bowl')
```

Example: Using control flows

```
# while the red block is to the left of the blue bowl, move it to the
right 5cm at a time.
while get_pos('red block')[0] < get_pos('blue bowl')[0]:
    target_pos = get_pos('red block') + [0.05, 0]
    put_first_on_second('red block', target_pos)
```

Example: Hierarchical Code Generation

```
import numpy as np
from utils import get_obj_bbox_xyxy
# define function: total = get_total(xs).
def get_total(xs):
    return np.sum(xs)
# define function: get_objs_bigger_than_area_th(obj_names, bbox_area_th).
def get_objs_bigger_than_area_th(obj_names, bbox_area_th):
    return [name for name in obj_names
            if get_obj_bbox_area(name) > bbox_area_th]
```

Have the LLM recursively define functions!

```
# define function: get_obj_bbox_area(obj_name).
def get_obj_bbox_area(obj_name):
    x1, y1, x2, y2 = get_obj_bbox_xyxy(obj_name)
    return (x2 - x1) * (y2 - y1)
```

Verifiably solve a number of tasks!

Can LLMs convert demonstrations (non-language) to code?

Demo2Code: From Summarizing Demonstrations to Synthesizing Code via Extended Chain-of-Thought

NeurIPS 2023

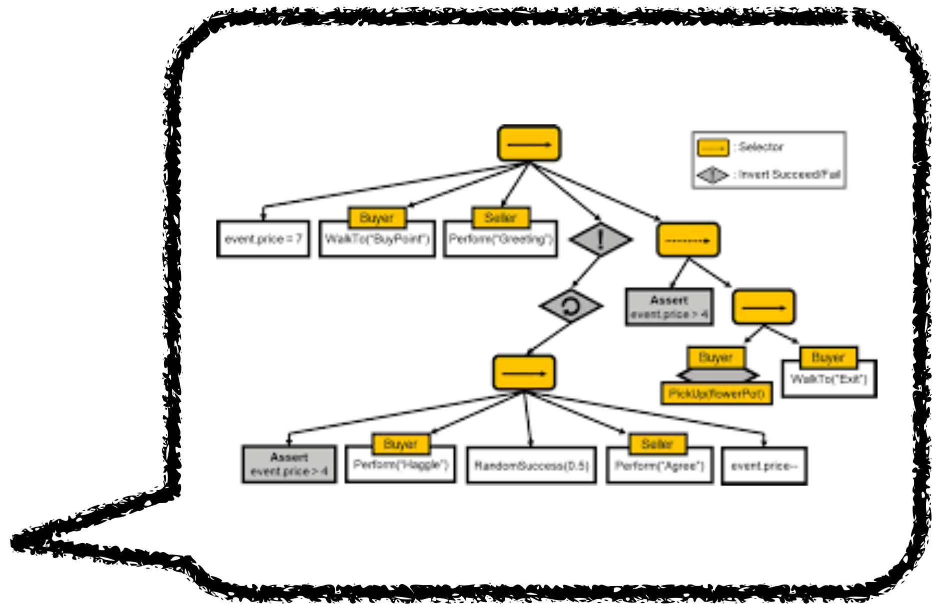
Huaxiaoyue Wang, Gonzalo Gonzalez-Pumariega, Yash Sharma, Sanjiban Choudhury
Cornell University

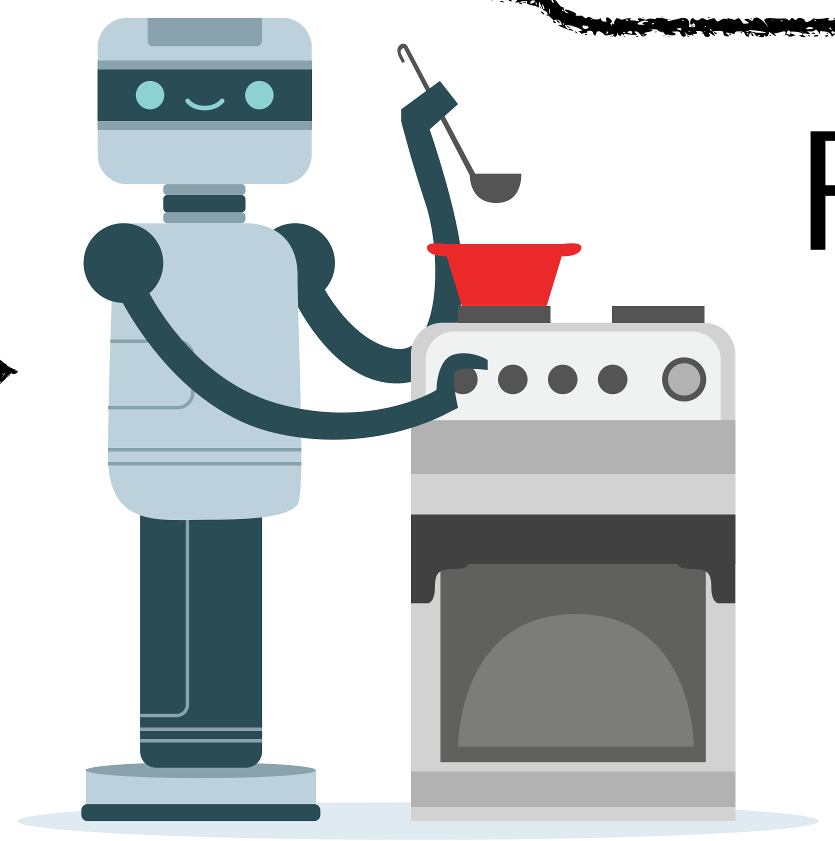
How can we teach robots *personalized* tasks?

Personalized Tasks

Language Narration:

"Here's how to make vegetable fried rice. Heat up some water. While the water boils, keep stirring vegetables. Pour rice."





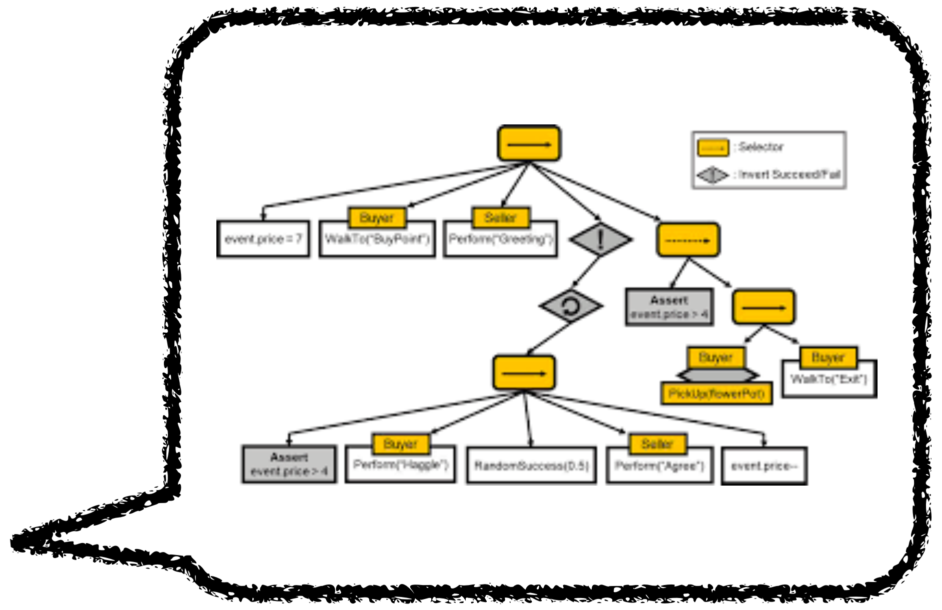
Robot Code

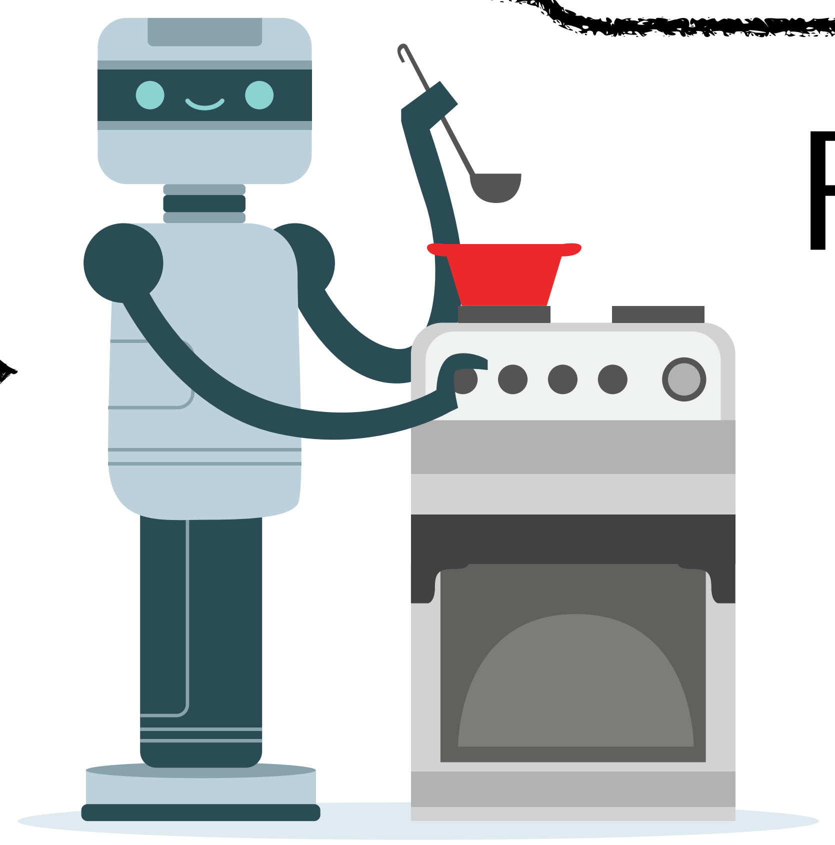
How can we teach robots *personalized* tasks?

Personalized Tasks

Language Narration:

"Here's how to make vegetable fried rice. Heat up some water. While the water boils, keep stirring vegetables. Pour rice."





Robot Code

Language alone is insufficient to communicate the task

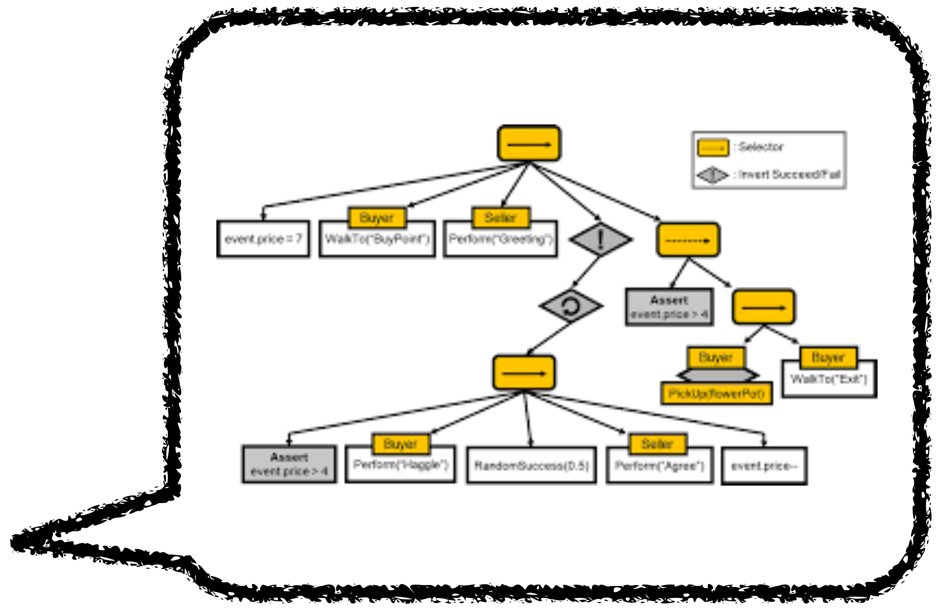
- ✗ Lacks specificity (e.g. Heat up water how? Pour rice where?)
- ✗ Leaves out implicit preferences (e.g. Personal style of stirring?)

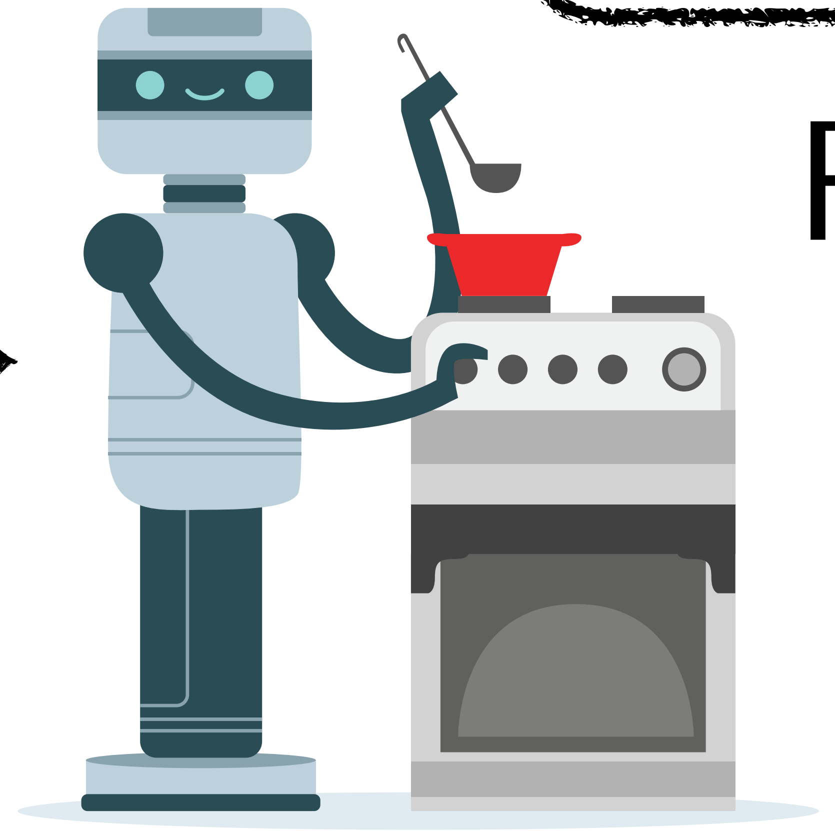
How can we teach robots *personalized* tasks?

Personalized Tasks

Language Narration:

"Here's how to make vegetable fried rice. Heat up some water. While the water boils, keep stirring vegetables. Pour rice."





Robot Code

Demonstrations:

Demonstrations convey dense information on how states change

`over('kettle', 'left_pan')`

`in('spatula', 'hand')`

`over('rice', 'left_pan')`

Language:

"Here's how to make vegetable fried rice. Heat up some water. While the water boils, keep stirring vegetables. Pour rice."

+

Demonstrations

(Sequence of states represented as text)

s_1

`over('kettle', 'left_pan')`

s_2

`in('spatula', 'hand')`

s_3

`over('rice', 'left_pan')`

→ LARGE LANGUAGE MODELS! →

Robot Code

Challenges

Challenge 1: Long-horizons

```
state_1
state_2
⋮
state_T
```

```
state_1
state_2
⋮
state_T
```

……

Each demonstration >= hundreds of states.
Multiple such demonstrations.

Challenge 2: Complex Task Code

```
def main():
    # Get a list of all the lettuces in the kitchen.
    lettuces = get_all_obj_names_that_match_type('lettuce')
    # Get a list of all the bottom buns in the kitchen.
    bottom_buns = get_all_obj_names_that_match_type('bottom_bun')
    # Get a list of all the patties in the kitchen.
    patties = get_all_obj_names_that_match_type('patty')
    # Get a list of all the top buns in the kitchen.
    top_buns = get_all_obj_names_that_match_type('top_bun')

    # Decide a lettuce to use.
    lettuce_to_use = lettuces[0]
    # Get a list of all the available cutting boards in the kitchen.
    cutting_boards = get_all_location_names_that_match_type('cutting
    # Decide a cutting board to go to.
    cutting_board_to_cut_at = cutting_boards[0]
    # Cut that lettuce at that cutting board.
    cut_object_at_location(obj=lettuce_to_use, location=cutting_boar

    # Decide a bottom bun to use.
    bottom_bun_to_use = bottom_buns[0]
    # Stack the lettuce on top of the bottom bun.
    # obj1 should be the lettuce, obj2 should be the bottom bun.
    stack_obj1_on_obj2(obj1=lettuce_to_use, obj2=bottom_bun_to_use)

    # Decide a patty to use.
    patty_to_use = patties[0]
    # Get a list of all the available stoves in the kitchen.
    stoves = get_all_location_names_that_match_type('stove')
    # Decide a stove to go to.
    stove_to_cook_at = stoves[0]
    # Cook that patty at that stove.
    cook_object_at_location(obj=patty_to_use, location=stove_to_cook

    # Stack the patty on top of the lettuce.
    # obj1 should be the patty, obj2 should be the lettuce.
    stack_obj1_on_obj2(obj1=patty_to_use, obj2=lettuce_to_use)

    # Decide a top bun to use.
    top_bun_to_use = top_buns[0]
    # Stack the top bun on top of the patty.
    # obj1 should be the top bun, obj2 should be the patty.
    stack_obj1_on_obj2(obj1=top_bun_to_use, obj2=patty_to_use)
```

```
def cook_object_at_location(obj, location):
    # To cook an object, the robot first needs to be holding obj
    if not is_holding(obj):
        # If the robot is not holding obj, there are 2 scenarios:
        #   (1) if obj is in a stack ,unstack obj
        #   (2) else, pick up obj.
        if is_in_a_stack(obj):
            # Because obj is in a stack, robot need to move then unstack the o
            obj_at_bottom = get_obj_that_is_underneath(obj_at_top=obj)
            # move then unstack: first you move to the obj_at_bottom's locatio
            move_then_unstack(obj_to_unstack=obj, obj_at_bottom=obj_at_bottom,
        else:
            # move_then_pick: first you move to obj's location, then you pick
            move_then_pick(obj=obj, pick_location=get_obj_location(obj))
    # move then place: first you move to the location to cook at, then you pla
    move_then_place(obj=obj, place_location=location)
    # cook the object until it is cooked
    cook_until_is_cooked(obj=obj)

def stack_obj1_on_obj2(obj1, obj2):
    # To stack obj1 on obj2, the robot needs to be holding obj1
    if not is_holding(obj1):
        # If the robot is not holding obj1, there are 2 scenarios:
        #   (1) if obj1 is in a stack ,unstack obj1
        #   (2) else, pick up obj1.
        if is_in_a_stack(obj1):
            # Because obj1 is in a stack, robot need to move then unstack the
            obj_at_bottom = get_obj_that_is_underneath(obj_at_top=obj1)
            # move then unstack: first you move to the obj_at_bottom's location, then you unstack obj from obj_at_bottom
            move_then_unstack(obj_to_unstack=obj1, obj_at_bottom=obj_at_bottom, unstack_location=get_obj_location(obj_at_bottom))
        else:
            # move_then_pick: first you move to obj's location, then you pick obj up
            move_then_pick(obj=obj1, pick_location=get_obj_location(obj1))
    # determine the location of obj2 to stack on
    obj2_location = get_obj_location(obj2)
    # move then stack: first you move to obj2's location, then you stack obj1 on obj2
    move_then_stack(obj_to_stack=obj1, obj_at_bottom=obj2, stack_location=obj2_location)

def cut_object_at_location(obj, location):
    # To cut an object, the robot first needs to be holding obj
    if not is_holding(obj):
        # If the robot is not holding obj, there are 2 scenarios:
        #   (1) if obj is in a stack ,unstack obj
```

```
def move_then_unstack(obj_to_unstack, obj_at_bottom, unstack_location):
    if get_curr_location() != unstack_location:
        move(get_curr_location(), unstack_location)
    unstack(obj_to_unstack, obj_at_bottom)

def move_then_pick(obj, pick_location):
    if get_curr_location() != pick_location:
        move(get_curr_location(), pick_location)
    pick_up(obj, pick_location)

def move_then_place(obj, place_location):
    if get_curr_location() != place_location:
        move(get_curr_location(), place_location)
    place(obj, place_location)

def cook_until_is_cooked(obj):
    start_cooking(obj)
    while not is_cooked(obj):
        noop()

def move_then_stack(obj_to_stack, obj_at_bottom, stack_location):
    if get_curr_location() != stack_location:
        move(get_curr_location(), stack_location)
    stack(obj_to_stack, obj_at_bottom)

def cut_until_is_cut(obj):
    while not is_cut(obj):
        cut(obj)
```

Loops, checks, and calls to custom robot libraries ..

Directly going from demo to code is hard …

[Demonstration N]

[Demonstration 2]

[Demonstration 1]

```
Make a burger.

...
State 5:
'robot' is not holding
'patty1'
'patty1' is at 'stove1'
...

State 9:
'patty1' is cooked
...

State 12:
'robot' is not holding
'patty1'
'patty1' is on top of
'bottom_bun1'
...
```

```
# Cook object at location
def cook_object_at_loc(obj,
loc):
   if not is_holding(obj):
      ...
   move_then_place(obj, loc)
   cook_until_is_cooked(obj)

# Move to a location and place
object
def move_then_place(obj, loc):
     curr_loc = get_curr_loc()
     if curr_loc != loc:
          move(curr_loc, loc)
     place(obj, place_location)
...
...
def main():
   ...
   cook_object_at_loc(patty,
   stove)
   ...
   stack_objects(top_bun,
   lettuce)
```

Demonstrations can be rationalized by a latent, compact specification

(Like Reward Functions in IRL)

Key Insight: Extended chain-of-thought

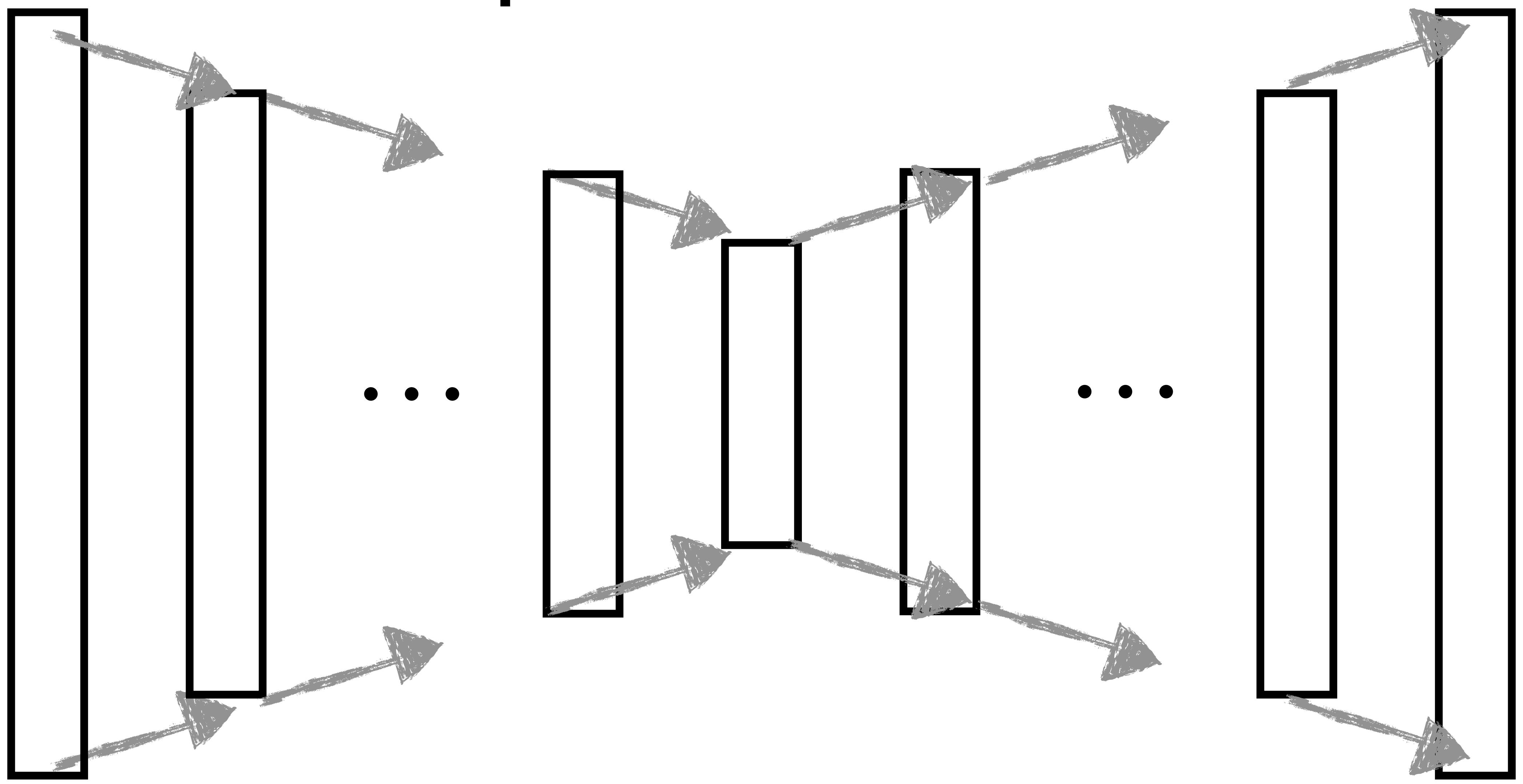
[Demonstration N]

[Demonstration 2]

[Demonstration 1]

```
Make a burger.

...
State 5:
'robot' is not holding
'patty1'
'patty1' is at 'stove1'
...

State 9:
'patty1' is cooked
...

State 12:
'robot' is not holding
'patty1'
'patty1' is on top of
'bottom_bun1'
...
```

Specification



Every step along the chain is small and easy for LLM

```
# Cook object at location
def cook_object_at_loc(obj,
loc):
   if not is_holding(obj):
      ...
   move_then_place(obj, loc)
   cook_until_is_cooked(obj)

# Move to a location and place
object
def move_then_place(obj, loc):
    curr_loc = get_curr_loc()
    if curr_loc != loc:
         move(curr_loc, loc)
    place(obj, place_location)
...
...
def main():
   ...
   cook_object_at_loc(patty,
   stove)
   ...
   stack_objects(top_bun,
   lettuce)
```

Demo2Code

Demo2Code: Recursive Summarization and Expansion

[Demonstration N]

[Demonstration 2]

[Demonstration 1]

```
Make a burger.

...
State 5:
'robot' is not holding
'patty1'
'patty1' is at 'stove1'
...

State 9:
'patty1' is cooked
...

State 12:
'robot' is not holding
'patty1'
'patty1' is on top of
'bottom_bun1'
...
```

Make a burger with one patty and one lettuce.

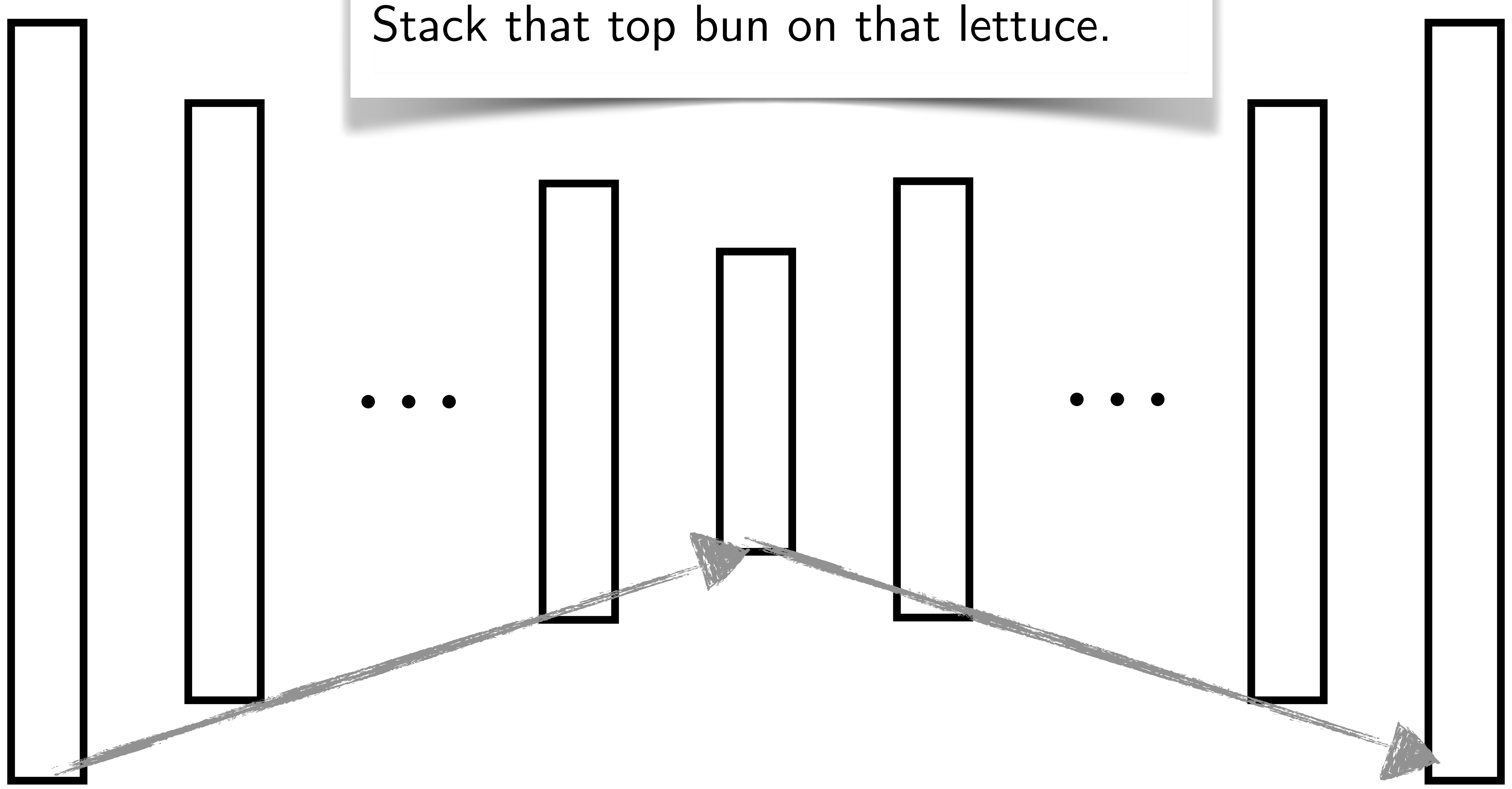
Specifically:

...

Cook a patty at that stove.

...

Stack that top bun on that lettuce.



Stage 1
Recursive summarize demo to specification

Stage 2
Recursive expand specification to task code

```
# Cook object at location
def cook_object_at_loc(obj,
loc):
   if not is_holding(obj):
      ...
   move_then_place(obj, loc)
   cook_until_is_cooked(obj)

# Move to a location and place
object
def move_then_place(obj, loc):
     curr_loc = get_curr_loc()
     if curr_loc != loc:
          move(curr_loc, loc)
     place(obj, place_location)
...
...
def main():
   ...
   cook_object_at_loc(patty,
   stove)
   ...
   stack_objects(top_bun,
   lettuce)
```

Experiments

We made a new game: Robotouille!

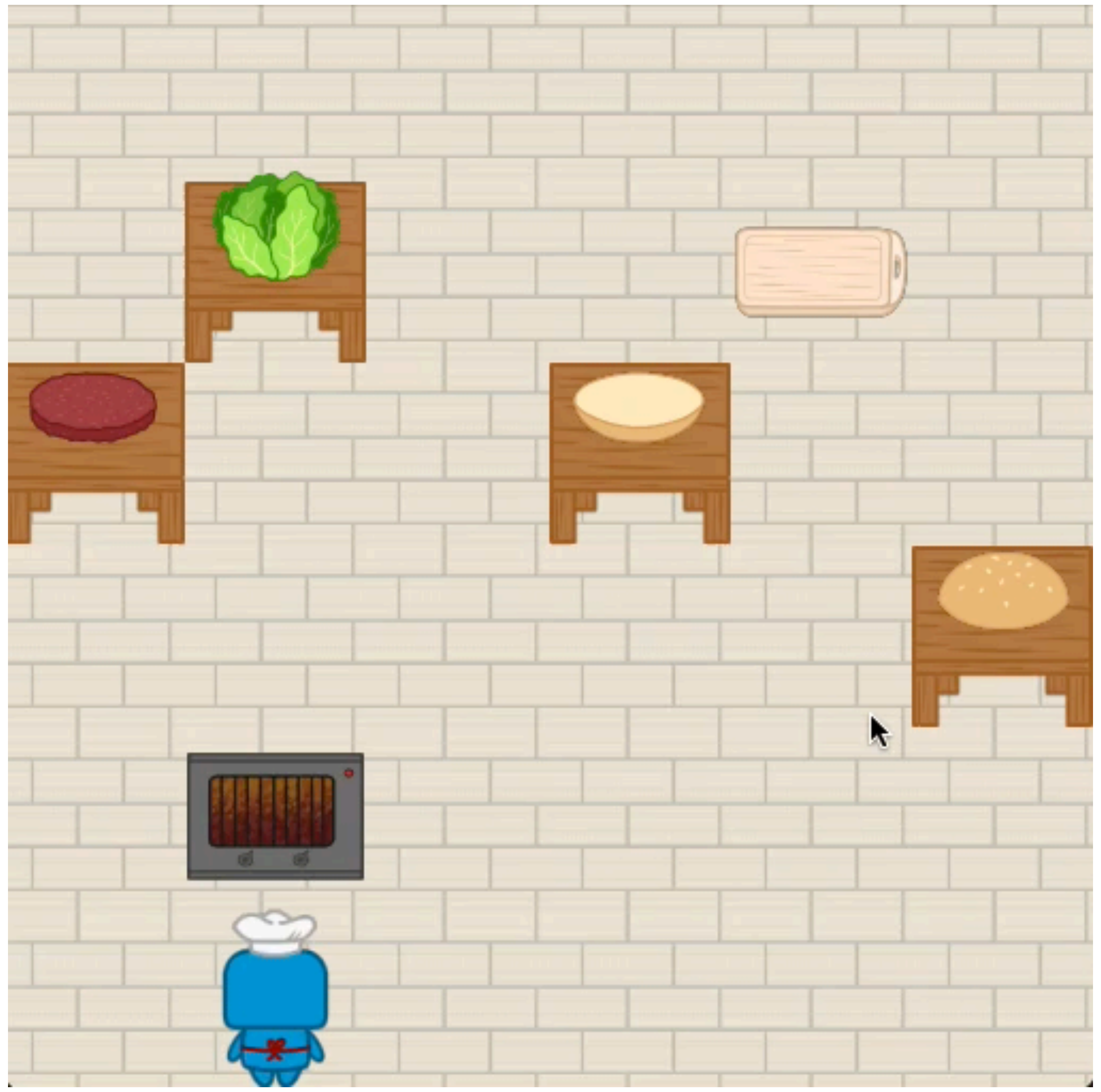
Open-source game coming soon on ioS / Android

Human runs a restaurant with robot sous-chef

Fun way to learn how humans plan and communicate tasks!

Demo2Code can generalize to new, complex environments

Make a burger.



User provides a demonstration in a simple environment

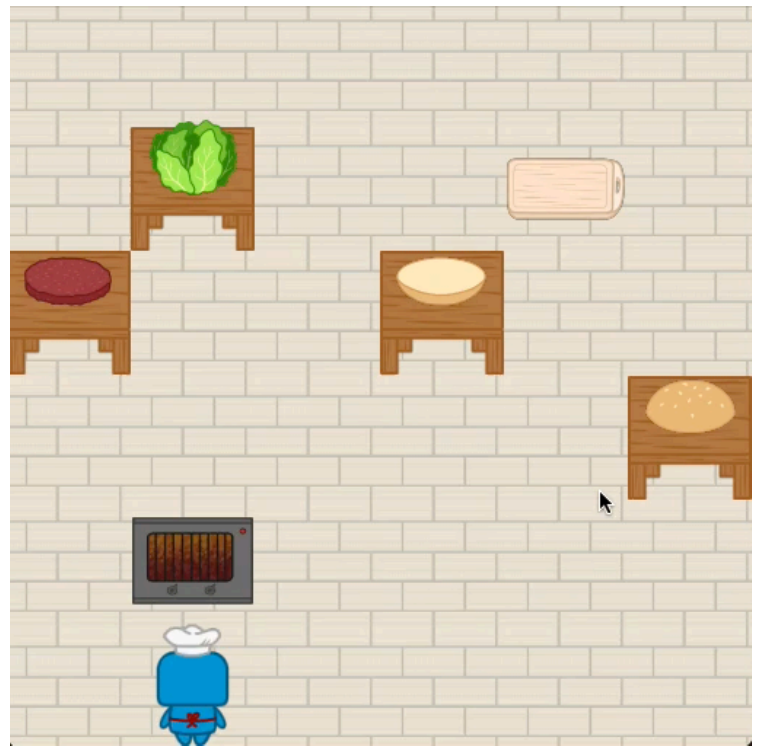
....

Make a burger.
...
Cut that lettuce at that cutting board.
...
Stack the lettuce on top of the bottom bun.
...
Cook that patty at that stove.
...
Stack the patty on top of the lettuce.
...
Stack the top bun on top of the patty.

....

```
def cook_object_at_loc(obj,
loc):
   if not is_holding(obj):
      ...
   move_then_place(obj, loc)
   cook_until_is_cooked(obj)

def move_then_place(obj, loc):
   curr_loc = get_curr_loc()
   if curr_loc != loc:
      move(curr_loc, loc)
   place(obj, place_location)

def main():
   ...
   cut_object_at_loc(lettuce,
   stove)
   ...
   stack_objects(lettuce,
   bottom_bun)
   ...
   cook_object_at_loc(patty,
   stove)
   ...
   stack_objects(top_bun,
   patty)
```

Demo2Code can generalize to new, complex environments



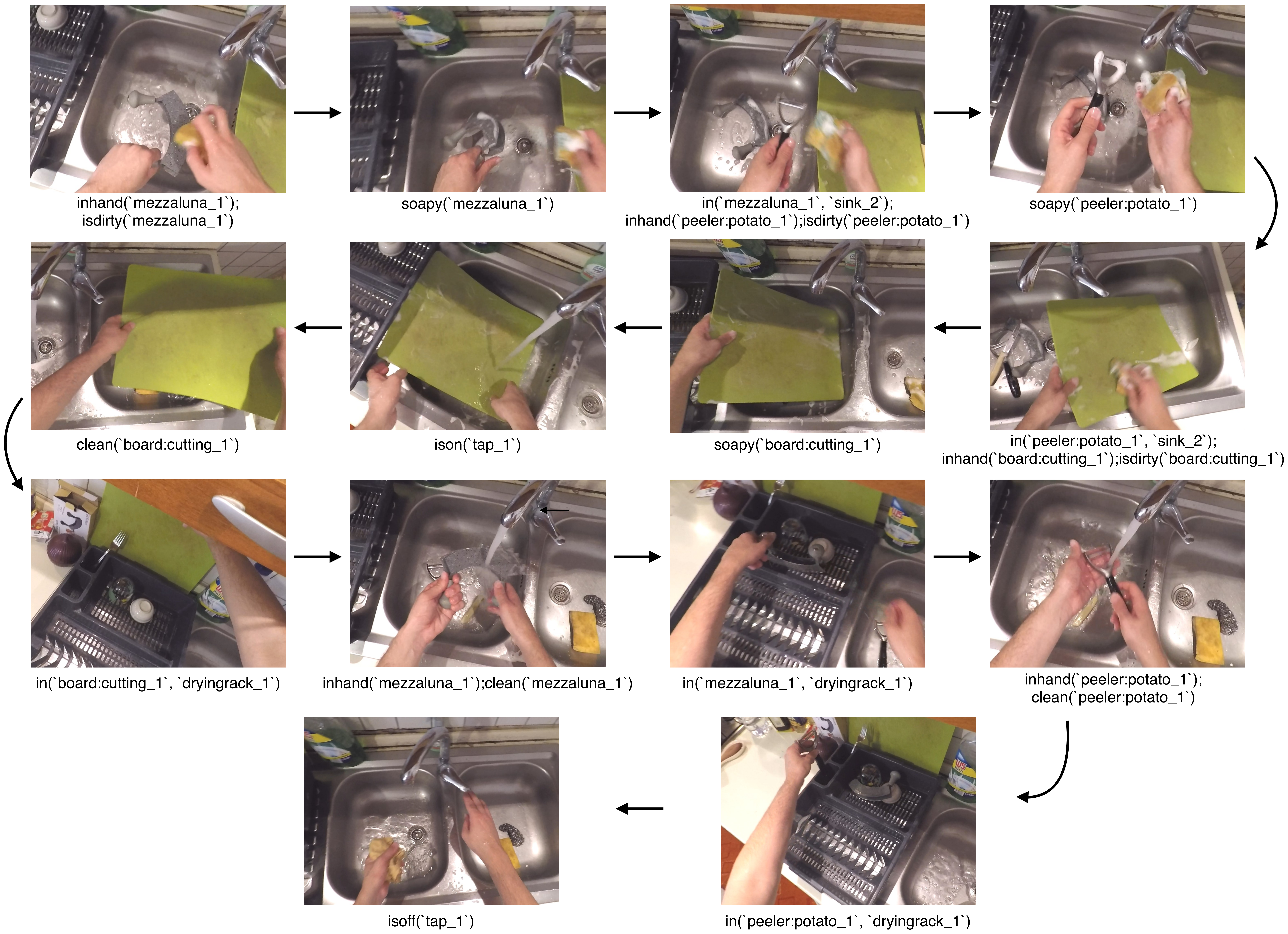
```
def cook_object_at_loc(obj,
loc):
   if not is_holding(obj):
      ...
   move_then_place(obj, loc)
   cook_until_is_cooked(obj)

def move_then_place(obj, loc):
   curr_loc = get_curr_loc()
   if curr_loc != loc:
      move(curr_loc, loc)
   place(obj, place_location)

def main():
   ...
   cut_object_at_loc(lettuce,
   stove)
   ...
   stack_objects(lettuce,
   bottom_bun)
   ...
   cook_object_at_loc(patty,
   stove)
   ...
   stack_objects(top_bun,
   patty)
```

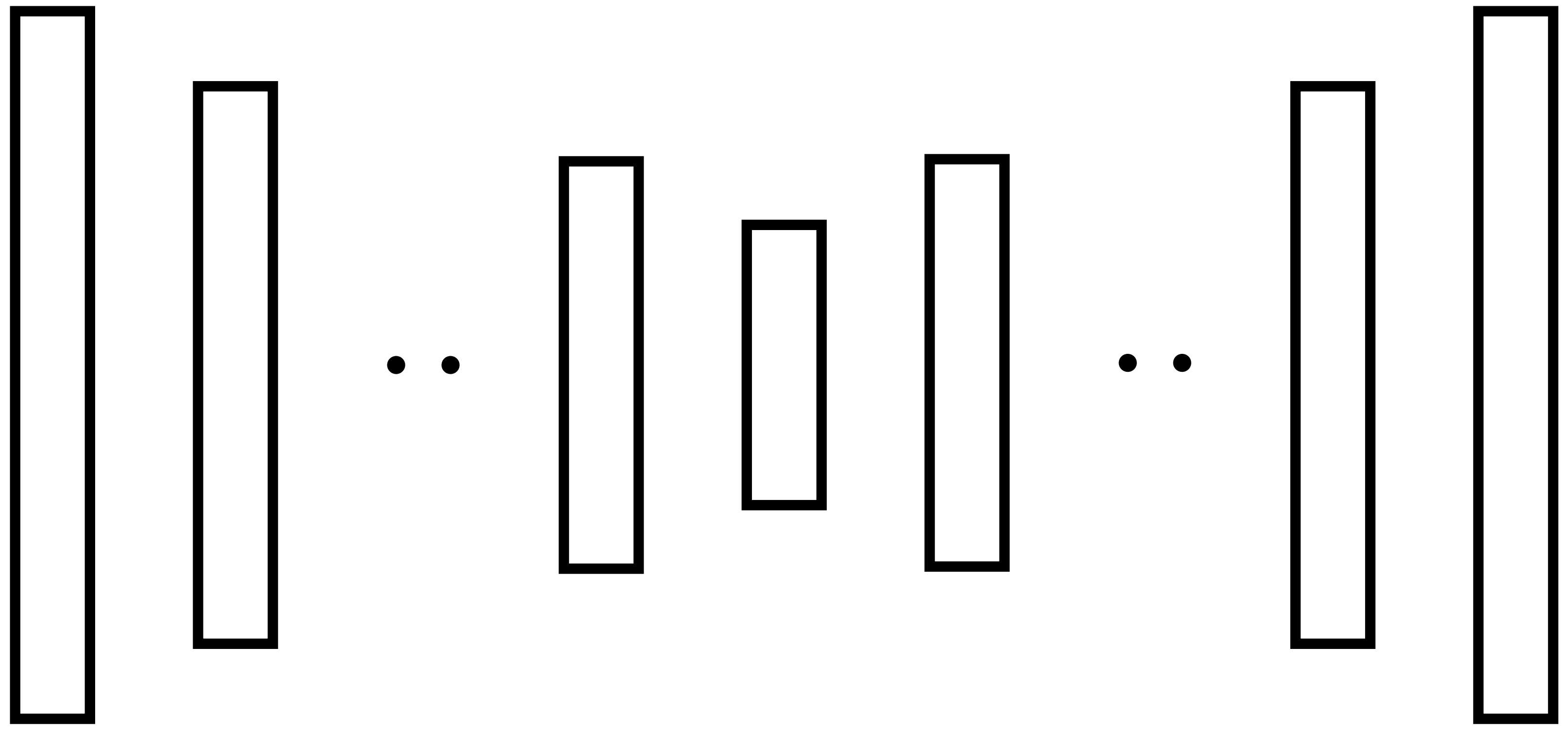
Demo2Code generates correct code that passes unit tests

Task	Lang2Code[33]			DemoNoLang2Code			Demo2Code(*ours*)			Horizon
	Exec.	Pass.	BLEU.	Exec.	Pass.	BLEU.	Exec.	Pass.	BLEU.	Length
Cook a patty	1.00	1.00	0.90	1.00	1.00	0.90	1.00	1.00	0.90	8.0
Cook two patties	0.80	0.80	0.92	0.80	0.80	0.92	0.80	0.80	0.92	16.0
Stack a top bun on top of a cut lettuce on top of a bottom bun	1.00	1.00	0.70	0.00	0.00	0.75	1.00	1.00	0.60	14.0
Cut a lettuce	1.00	1.00	0.87	0.00	0.00	0.76	1.00	1.00	0.87	7.0
Cut two lettuces	0.80	0.80	0.92	0.00	0.00	0.72	0.80	0.80	0.92	14.0
Cook first then cut	1.00	1.00	0.88	1.00	1.00	0.88	1.00	1.00	0.88	14.0
Cut first then cook	1.00	1.00	0.88	0.00	0.00	0.82	1.00	1.00	0.88	15.0
Assemble two burgers one by one	0.00	0.00	0.34	1.00	1.00	0.77	1.00	1.00	0.76	15.0
Assemble two burgers in parallel	0.00	0.00	0.25	1.00	1.00	0.51	0.00	0.00	0.71	15.0
Make a cheese burger	1.00	0.00	0.24	1.00	1.00	0.69	1.00	1.00	0.69	18.0
Make a chicken burger	0.00	0.00	0.57	0.00	0.00	0.64	0.90	0.90	0.69	25.0
Make a burger stacking lettuce atop patty immediately	1.00	0.00	0.74	0.20	0.00	0.71	0.00	0.00	0.71	24.5
Make a burger stacking patty atop lettuce immediately	0.00	0.00	0.74	0.20	0.00	0.71	1.00	1.00	0.74	25.0
Make a burger stacking lettuce atop patty after preparation	1.00	0.00	0.67	0.10	0.00	0.65	0.00	0.00	0.66	26.5
Make a burger stacking patty atop lettuce after preparation	1.00	0.00	0.67	0.00	0.00	0.53	1.00	0.00	0.69	27.0
Make a lettuce tomato burger	0.00	0.00	0.13	1.00	1.00	0.85	1.00	0.00	0.66	34.0
Make two cheese burgers	0.00	0.00	0.63	1.00	1.00	0.68	1.00	1.00	0.68	38.0
Make two chicken burgers	0.00	0.00	0.52	0.00	0.00	0.68	1.00	0.00	0.56	50.0
Make two burgers stacking lettuce atop patty immediately	0.80	0.00	0.66	0.80	1.00	0.69	0.00	0.00	0.66	50.0
Make two burgers stacking patty atop lettuce immediately	0.80	0.00	0.67	1.00	0.00	0.48	1.00	1.00	0.73	50.0
Make two burgers stacking lettuce atop patty after preparation	0.80	0.00	0.66	0.60	0.00	0.66	0.80	0.00	0.67	54.0
Make two burgers stacking patty atop lettuce after preparation	0.80	0.00	0.67	0.50	0.00	0.71	0.80	0.00	0.68	54.0
Make two lettuce tomato burgers	1.00	0.00	0.55	0.00	0.00	0.70	1.00	1.00	0.84	70.0
Overall	0.64	0.29	0.64	0.49	0.38	0.71	0.79	0.59	0.74	28.8

[Damen et al '18]

EPIC-Kitchens Tasks





```
objects = get_all_objects()
for object in objects:
    pick_up(object)
    if check_if_dirty(object):
        while check_if_dirty(object):
            scrub(object)
    place(object, "sink_2")
turn_on("tap_1")
for object in objects:
    pick_up(object)
    rinse(object)
    place(object, "dryingrack_1")
turn_off("tap_1")
```

EPIC-Kitchens Dishwashing Tasks

	P4-101 (7)		P7-04 (17)		P7-10 (6)		P22-05 (28)		P22-07 (30)		P30-07 (11)		P30-08 (16)	
	Pass.	BLEU.	Pass.	BLEU.	Pass.	BLEU.	Pass.	BLEU.	Pass.	BLEU.	Pass.	BLEU.	Pass.	BLEU.
Lang2Code [33]	1.00	0.58	0.00	0.12	0.00	0.84	0.00	0.48	0.00	0.37	1.00	0.84	0.00	0.66
DemoNoLang2Code	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.37	0.00	0.51	1.00	0.57	0.00	0.00
Demo2Code	1.00	0.33	0.00	0.19	1.00	0.63	1.00	0.43	1.00	0.66	1.00	0.58	0.00	0.24

[Damen et al '18]


Exciting coming years for robot learning!


SO LONG AND...
Thanks for all the fish!