Large Language Models and Task Planning

Sanjiban Choudhury

Cornell Bowers CIS
Computer Science

The Problem

What do we want from Personal Robots?

[Nvidia, 2018]

[Toyota, 2020]

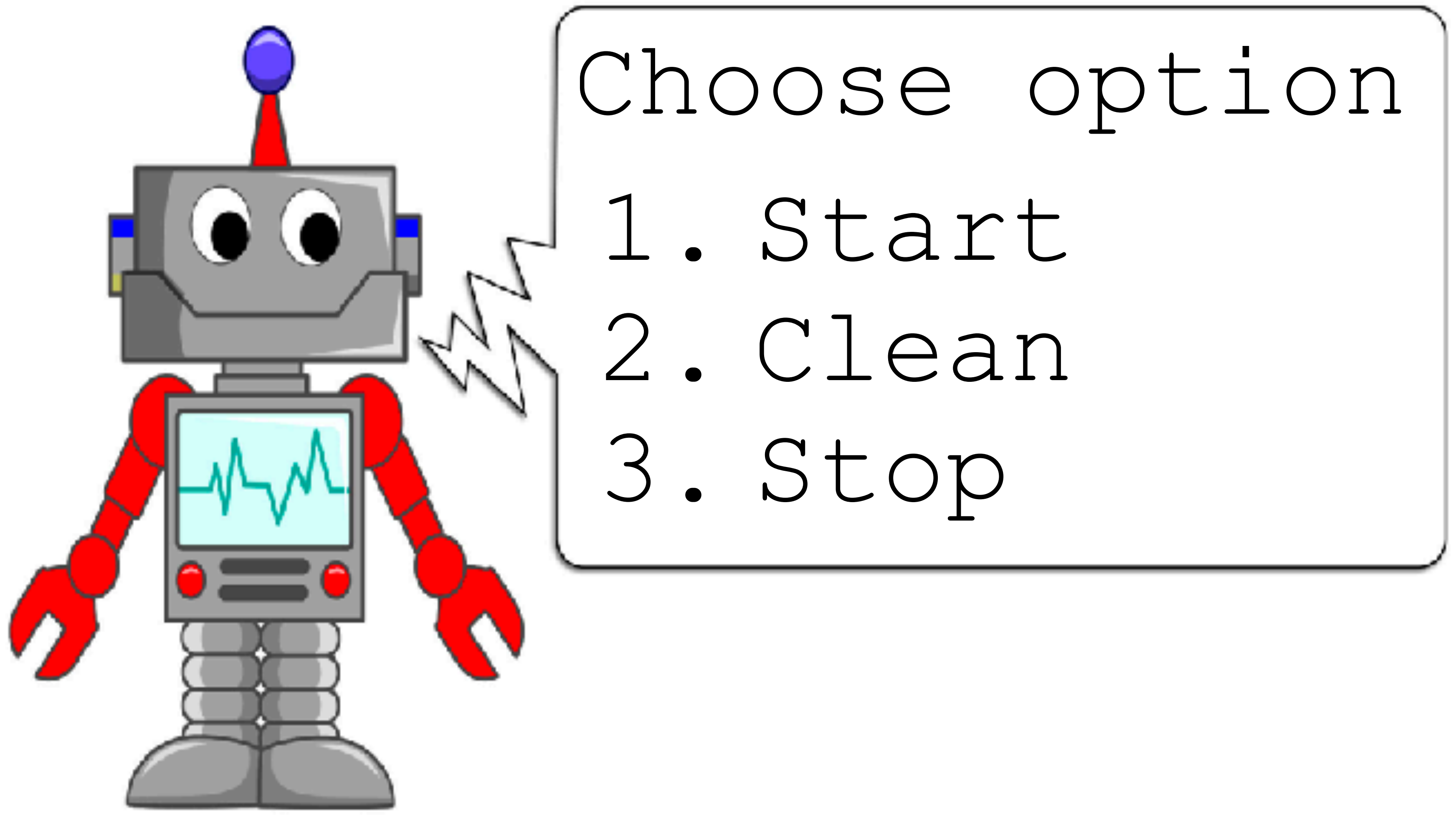
[Google, 2022]

[Tesla, 2023]

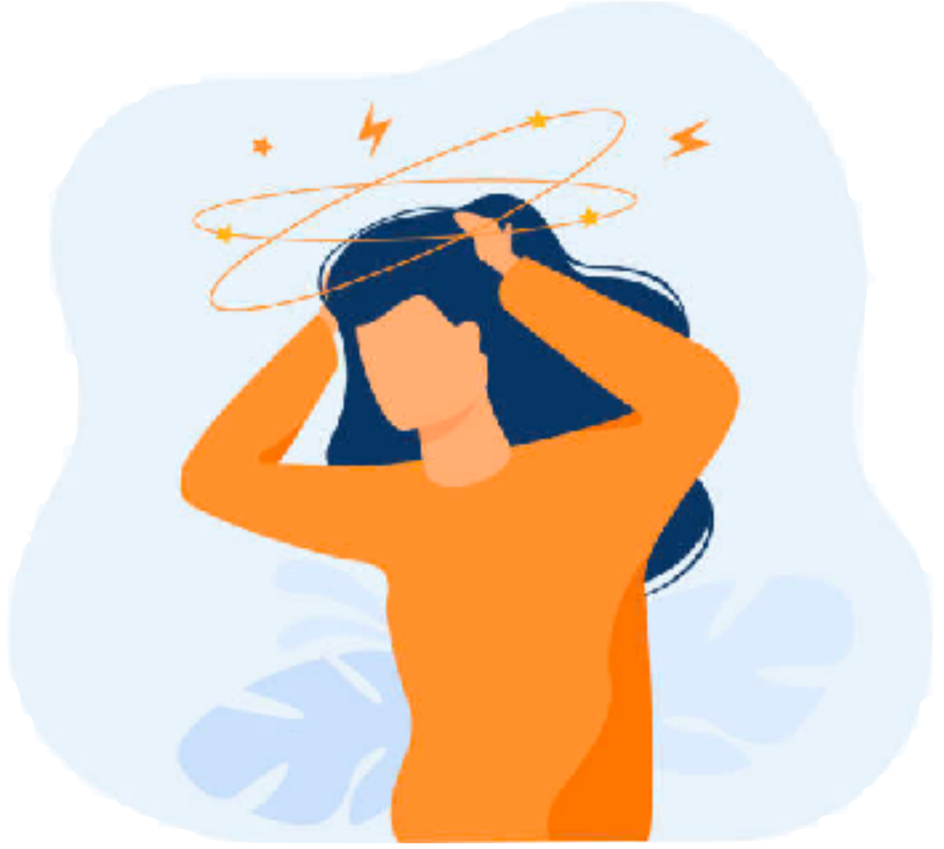
Every home is different

The way we program robots today is ... rigid!

Engineers hand-craft behaviors



Ship robot



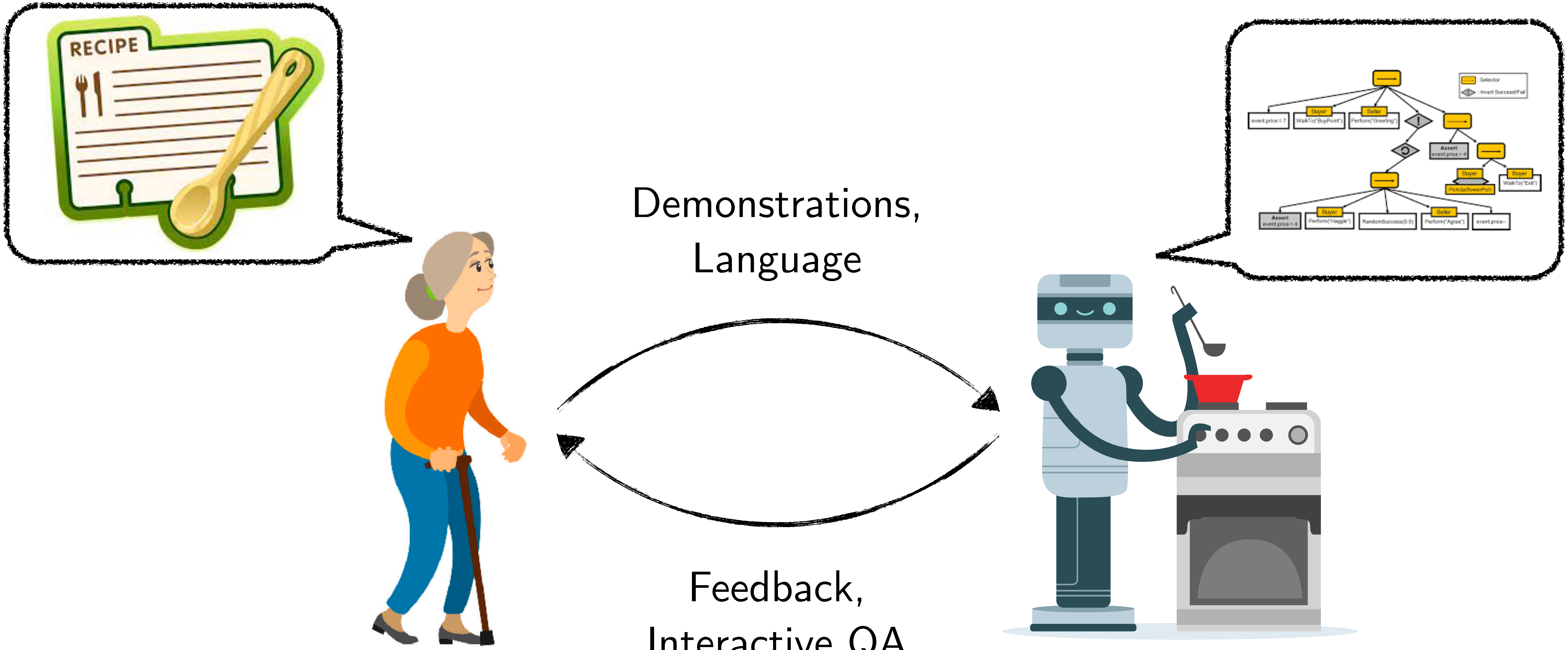
Frustrate users!

Cannot be flexibly re-programmed by everyday users

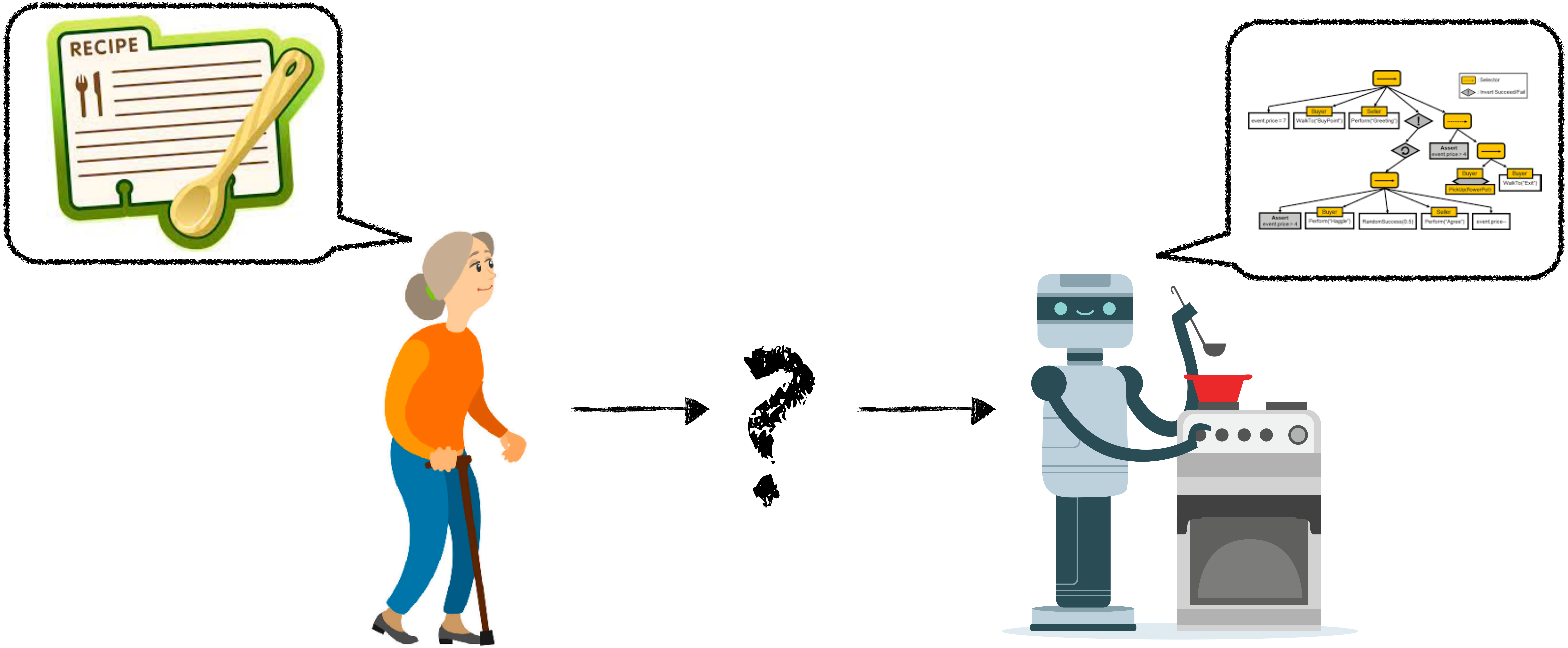
Instead of explicitly engineering behaviors

Can we implicitly program robots via natural interactions?

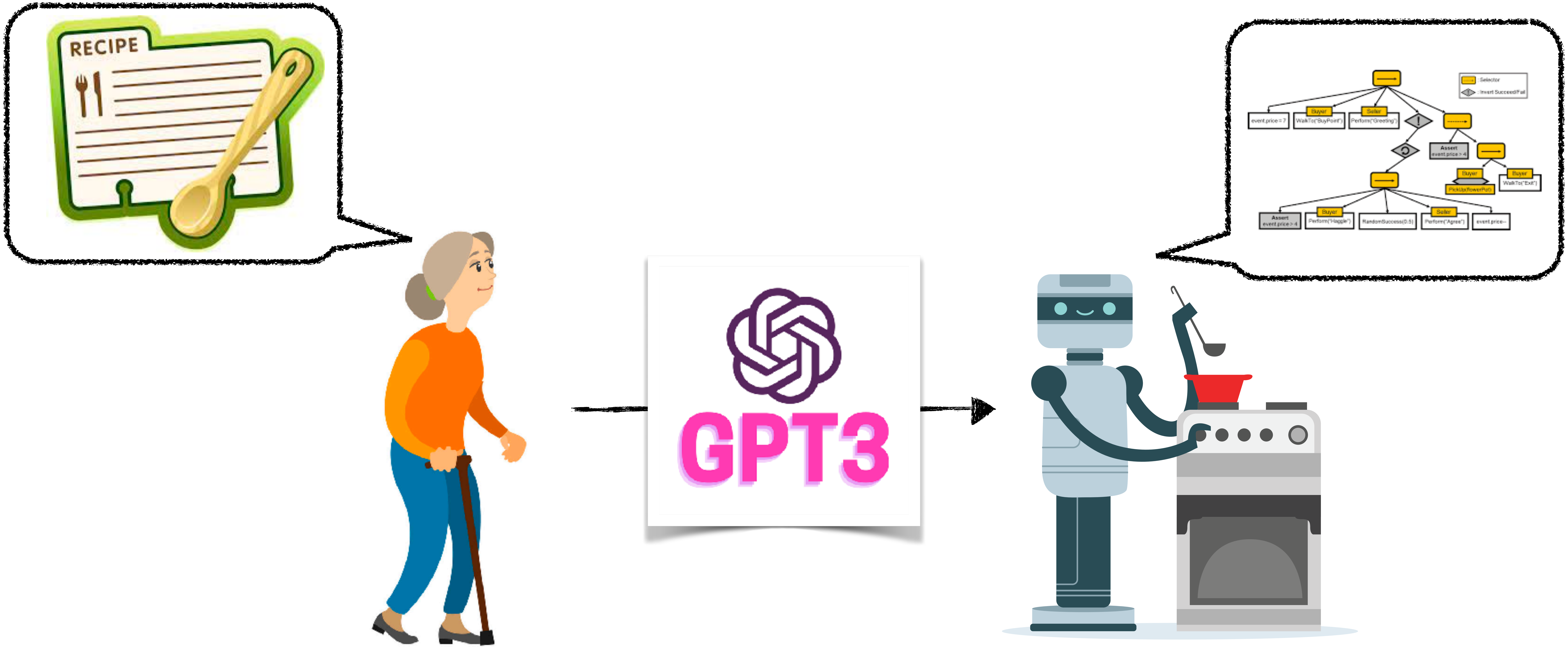
Programming via natural interactions



Question: How do we translate between humans and robots?



Large Language Models to the rescue!



An Example

HAL

Helping Out In the Kitchen

(**H**ome **A**pprentice **L**earner)

PORTAL

CORNELL UNIVERSITY FOUNDED A.D. 1865

Activity!

Think-Pair-Share!

Think (30 sec): Think of all the steps to go from what the human said to the code the robot has to execute.

Pair: Find a partner

Share (45 sec): Partners exchange ideas

Human: "Help me make vegetable soup"

Robot:

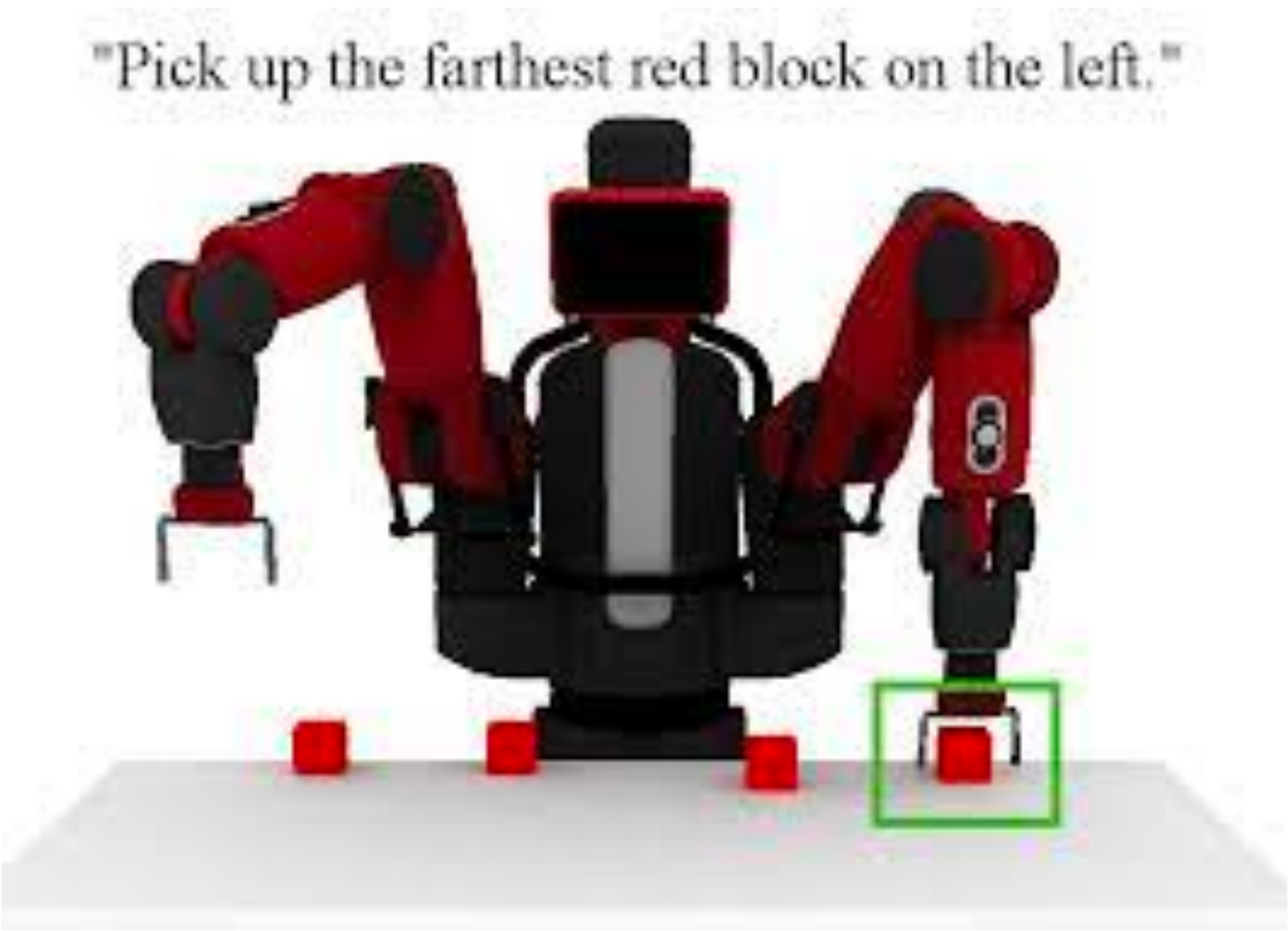
```
go_to(SALT)
pick_up_item(SALT)
go_to(TABLE)
place_item_at(TABLE)
```

How things worked pre-LLM

Two Fundamental Challenges

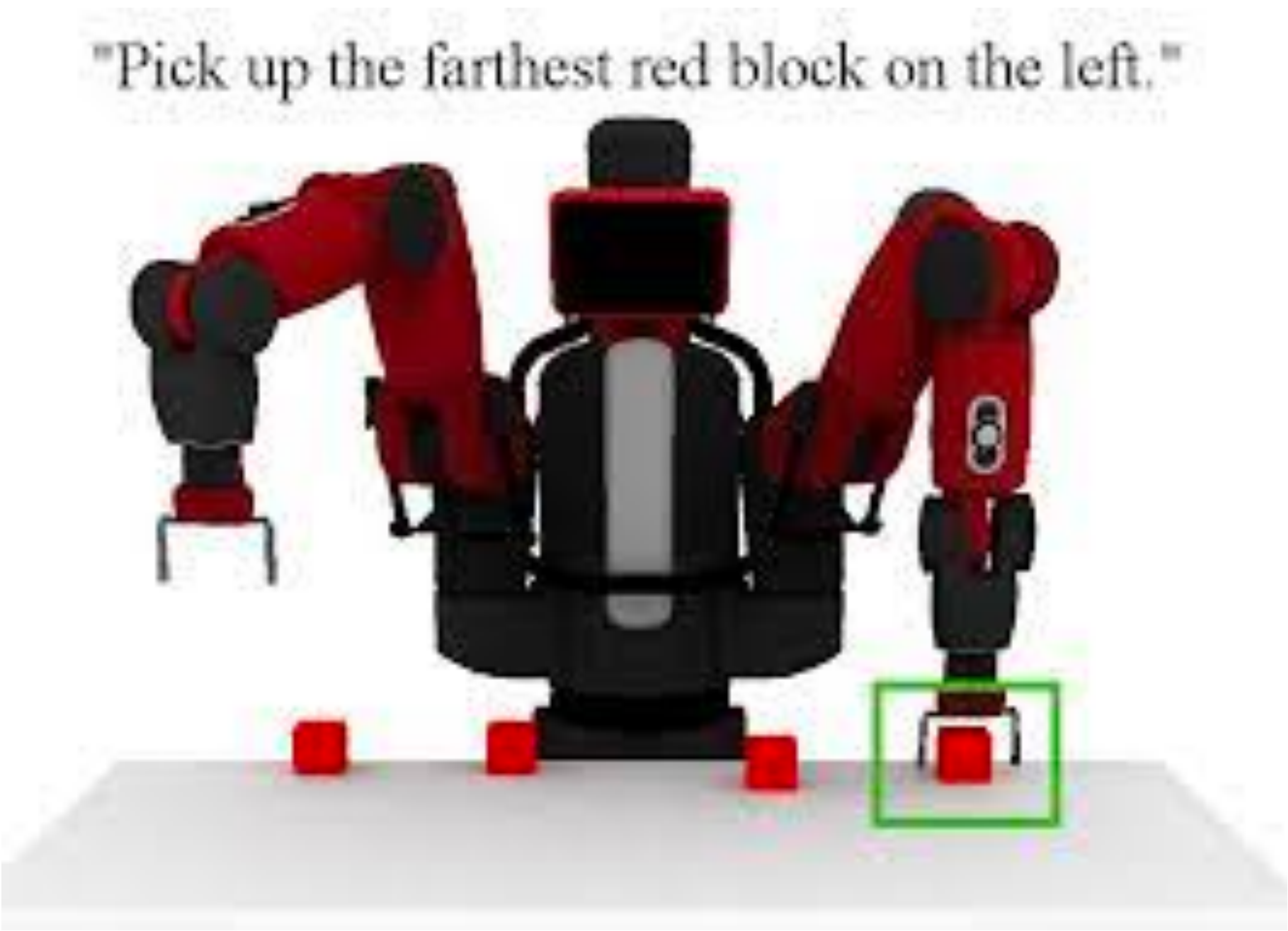
Two Fundamental Challenges

Challenge 1:
Ground natural language
in robot state

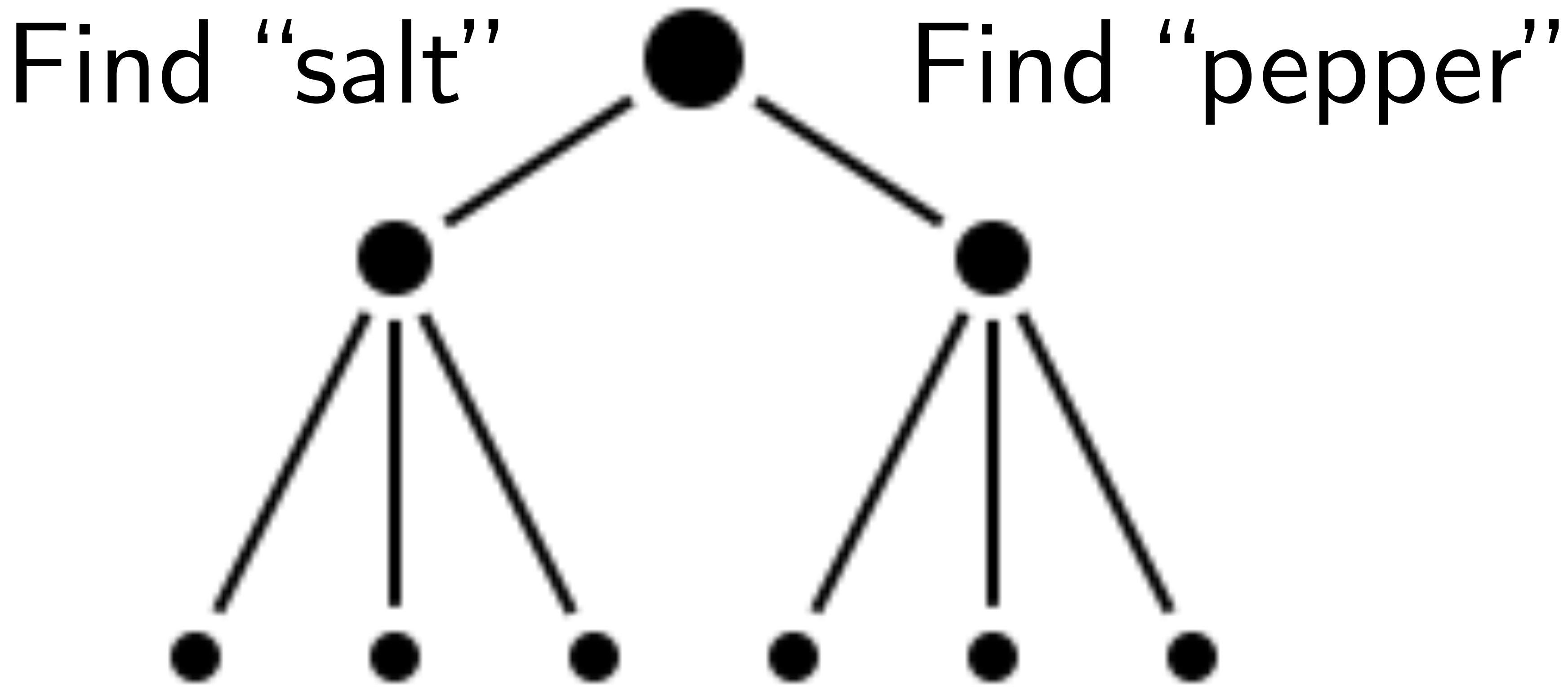


Two Fundamental Challenges

Challenge 1:
Ground natural language in robot state

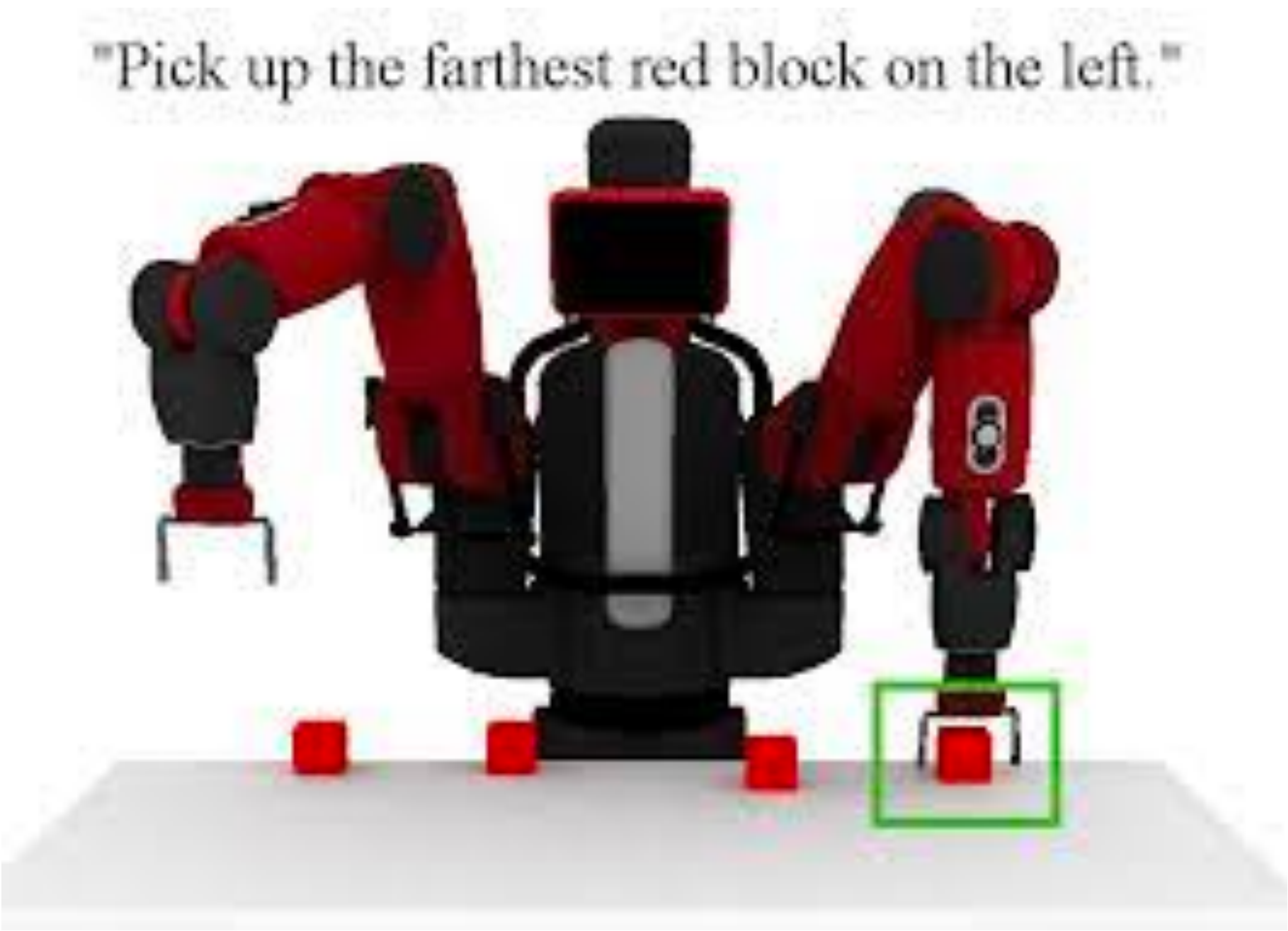


Challenge 2:
Planning actions to solve a task

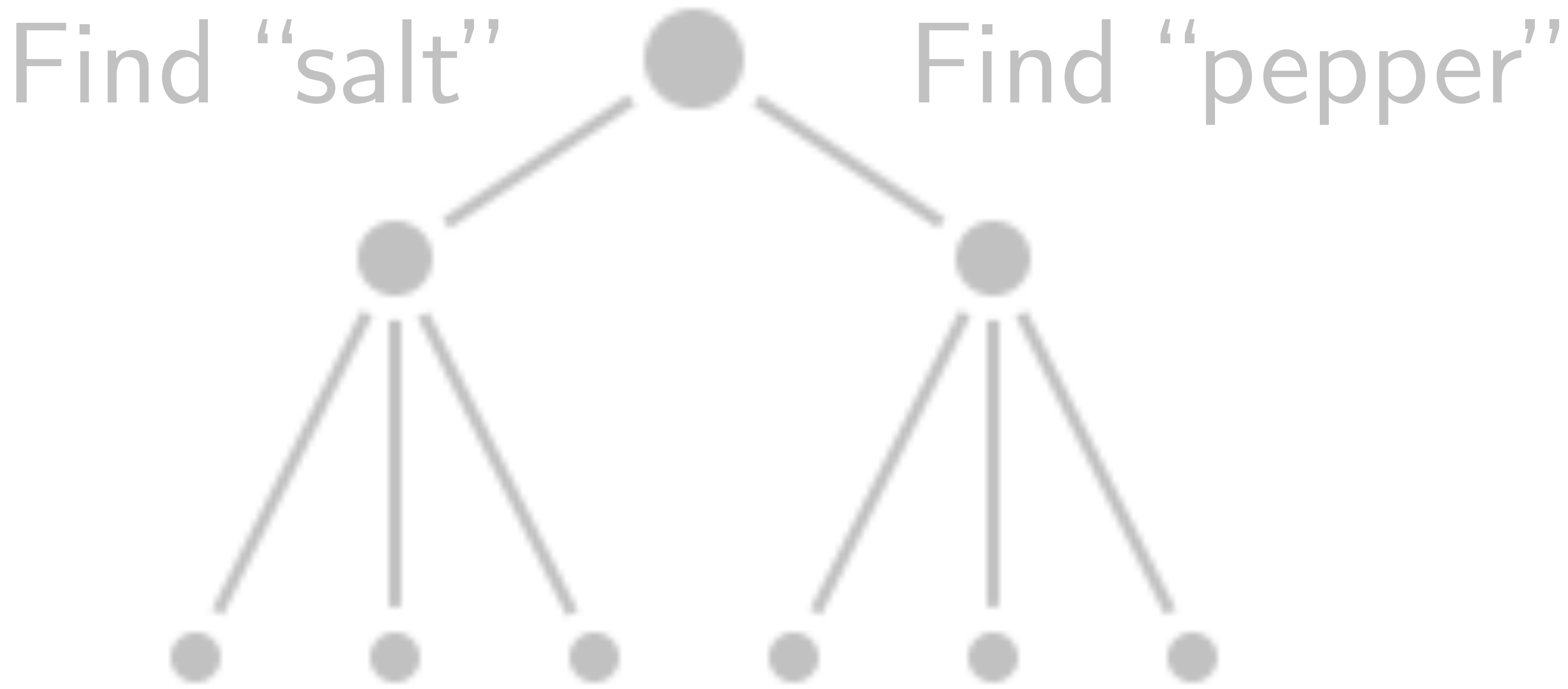


Two Fundamental Challenges

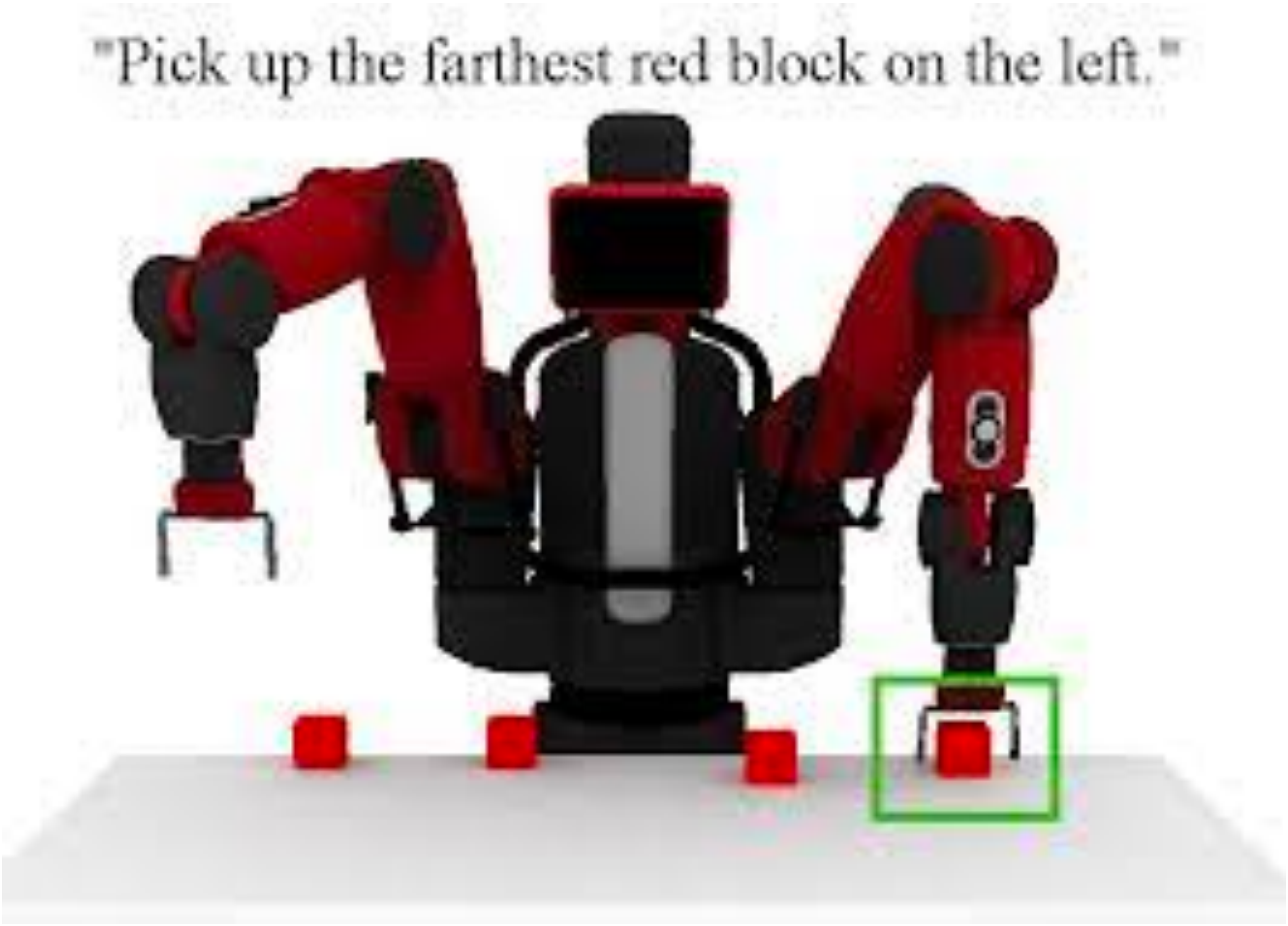
Challenge 1:
Ground natural language in robot state



Challenge 2:
Planning actions to solve a task



What is grounding? Why is it hard?

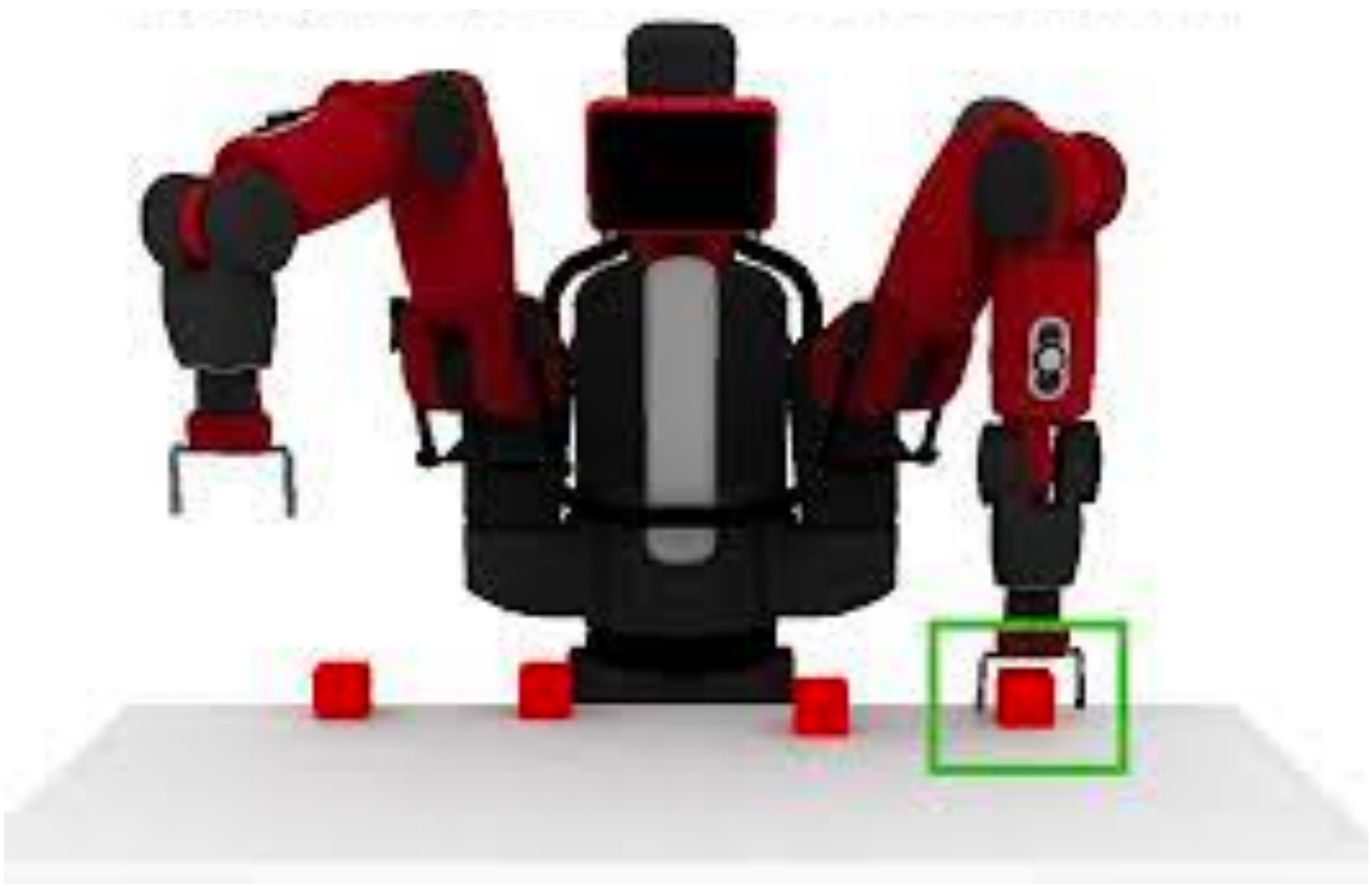


Grounding: Mapping language to robot's internal state

Natural Language → **MDP**

"Pick up the farthest red block"

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

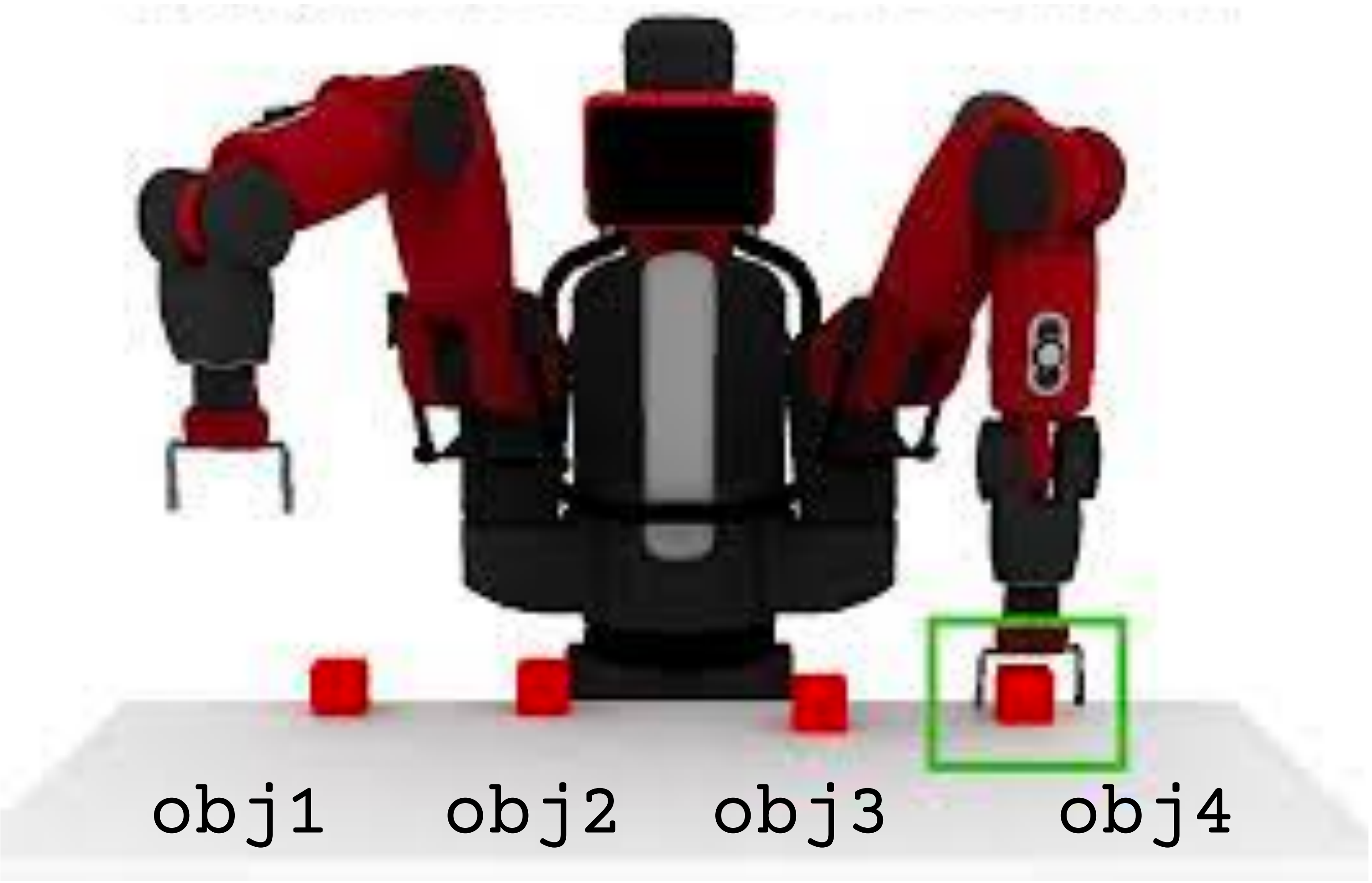


Grounding: Mapping language to robot's internal state

Natural Language $\longrightarrow$ **MDP**

"Pick up the farthest red block"

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



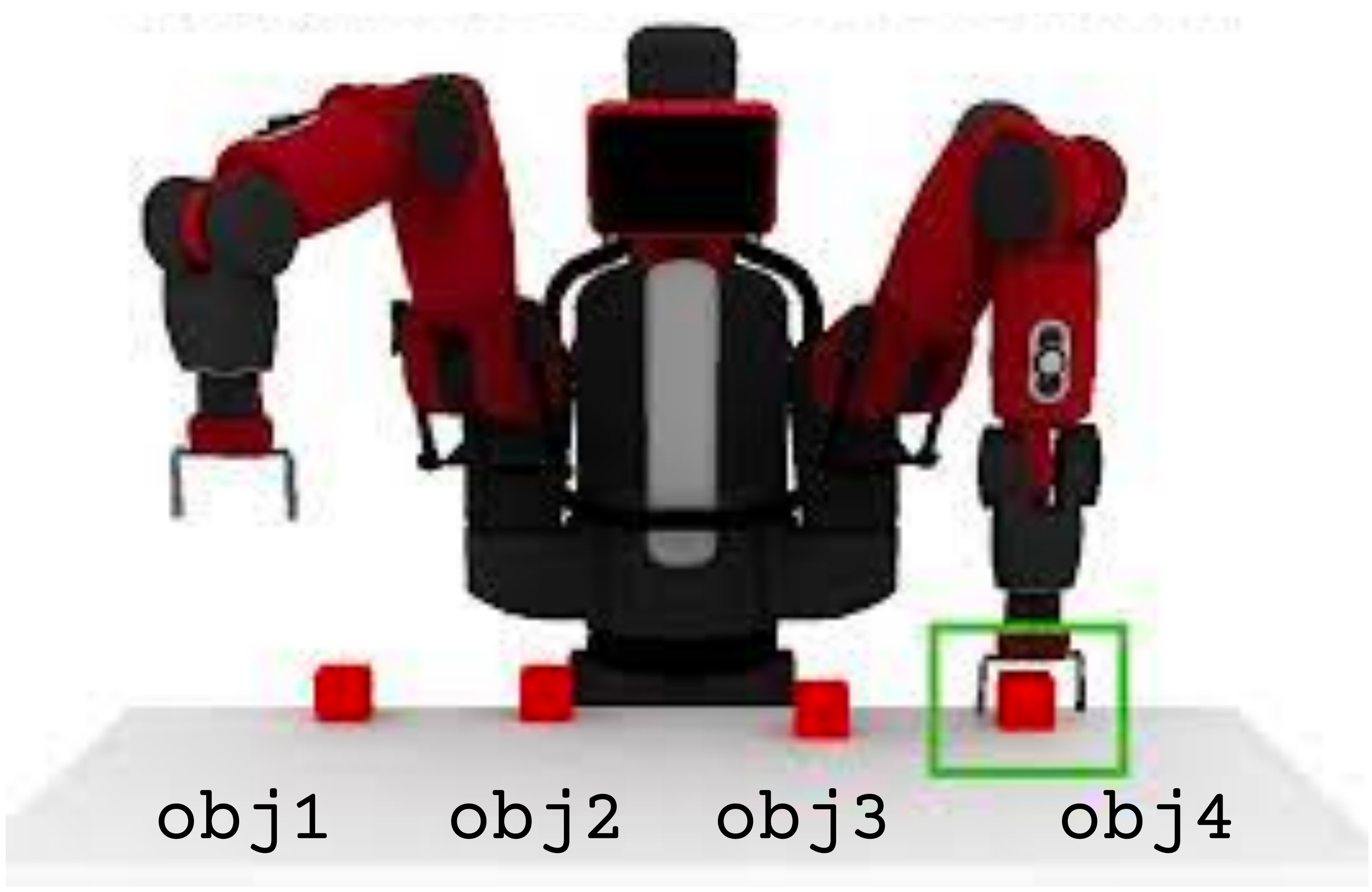

```
on('obj1','table')
on('obj2','table')
on('obj3','table')
on('obj4','table')
left('obj2','obj1')
left('obj3','obj2')
left('obj4','obj3')
...
```

Grounding: Mapping language to robot's internal state

Natural Language ⟶ **MDP**

"Pick up the farthest
red block"

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

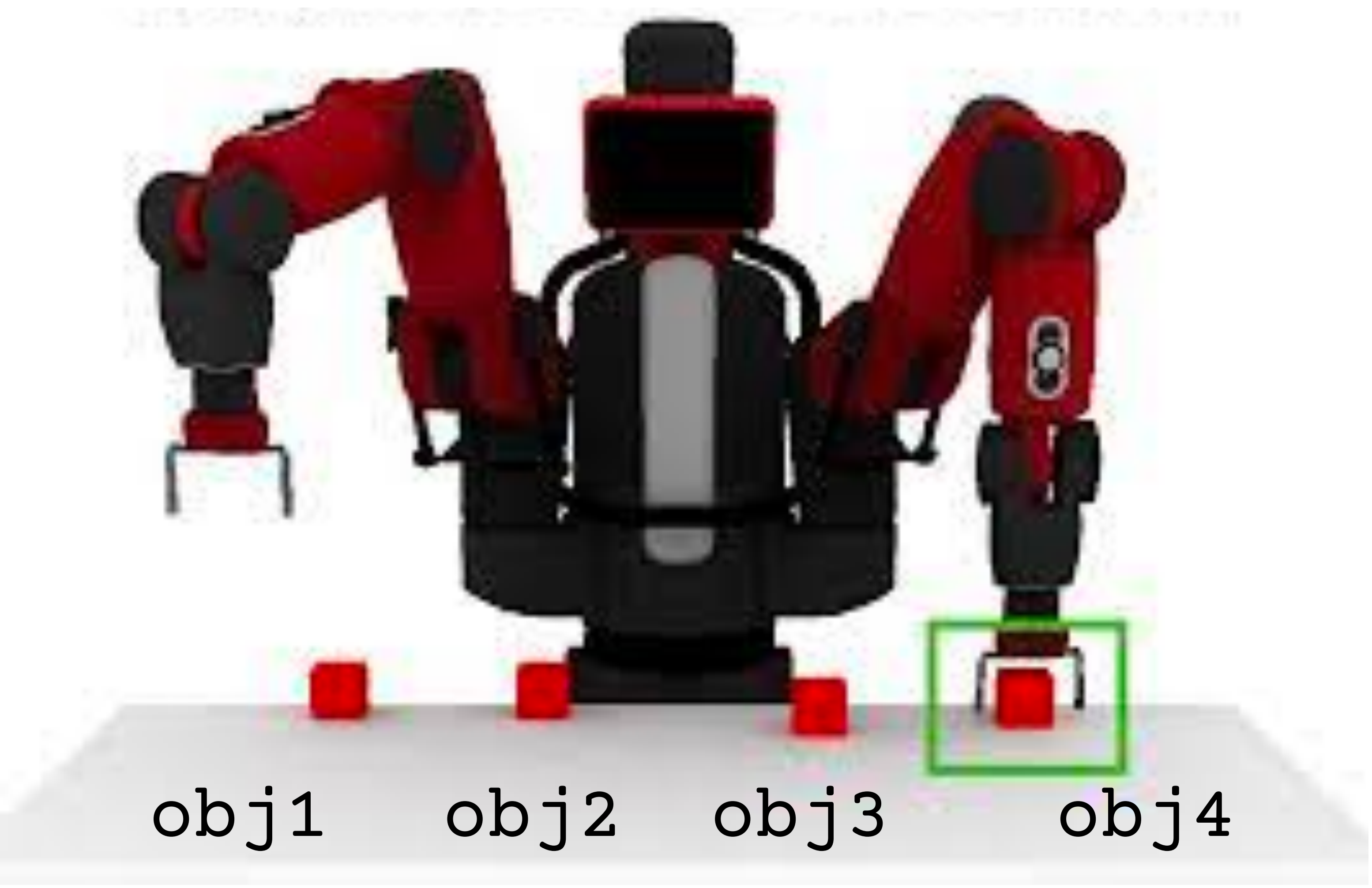


$\mathrm{R}(\texttt{in(obj4, hand)}) = +1$

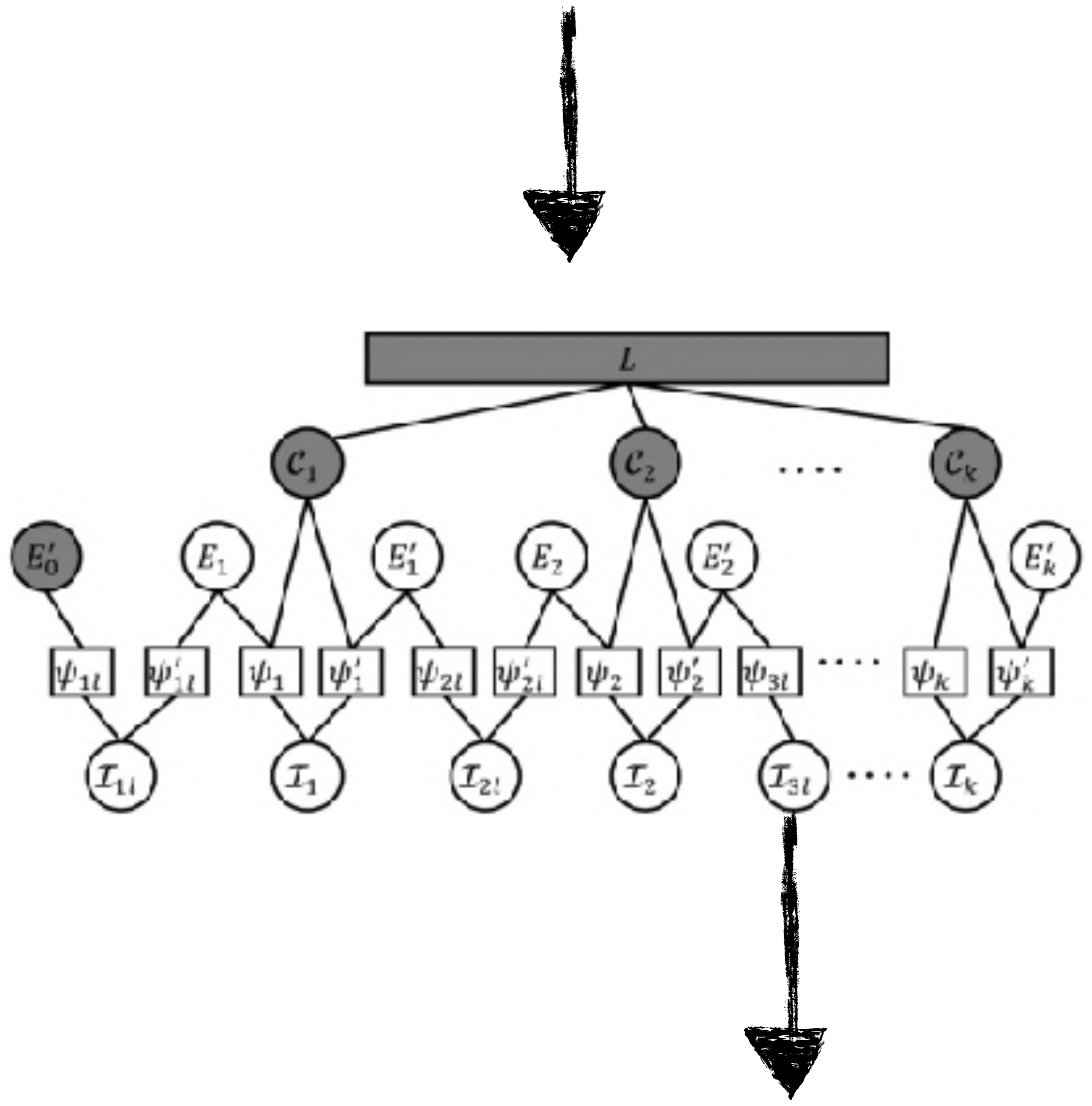
How did we solve grounding?

Train this on small, custom robot datasets!

“Pick up the farthest red block”

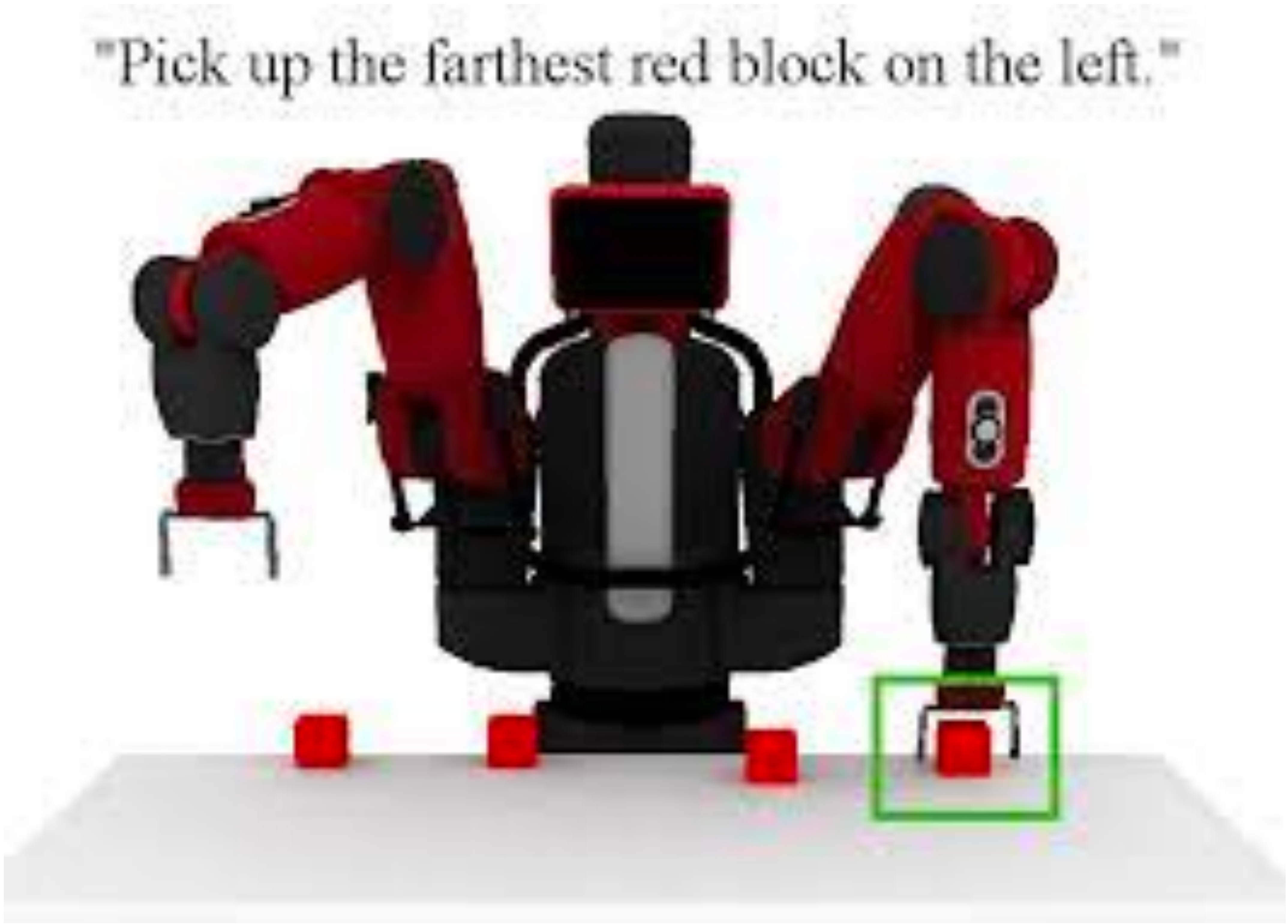


Complex graphical models!



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

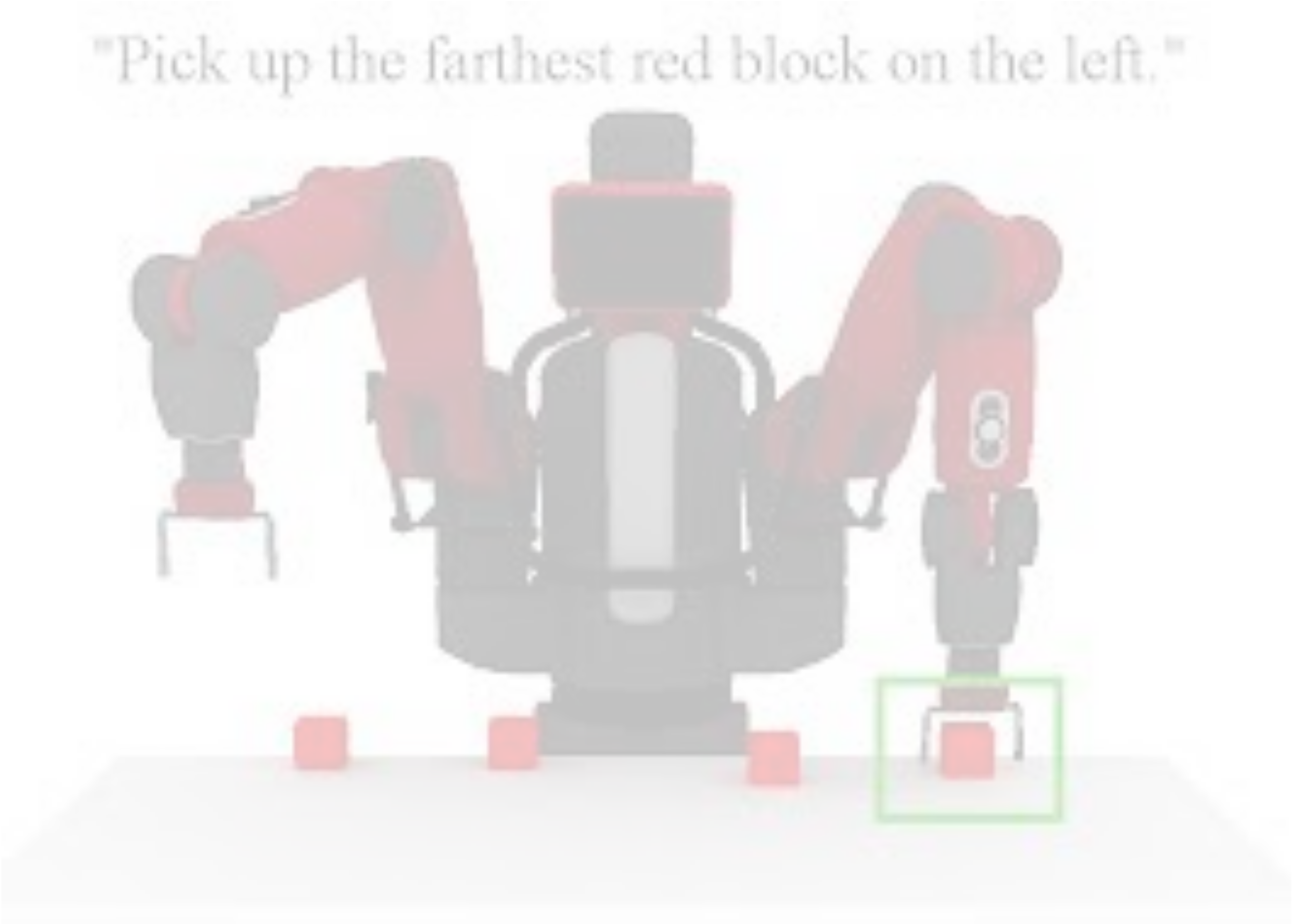
Why did this not scale?



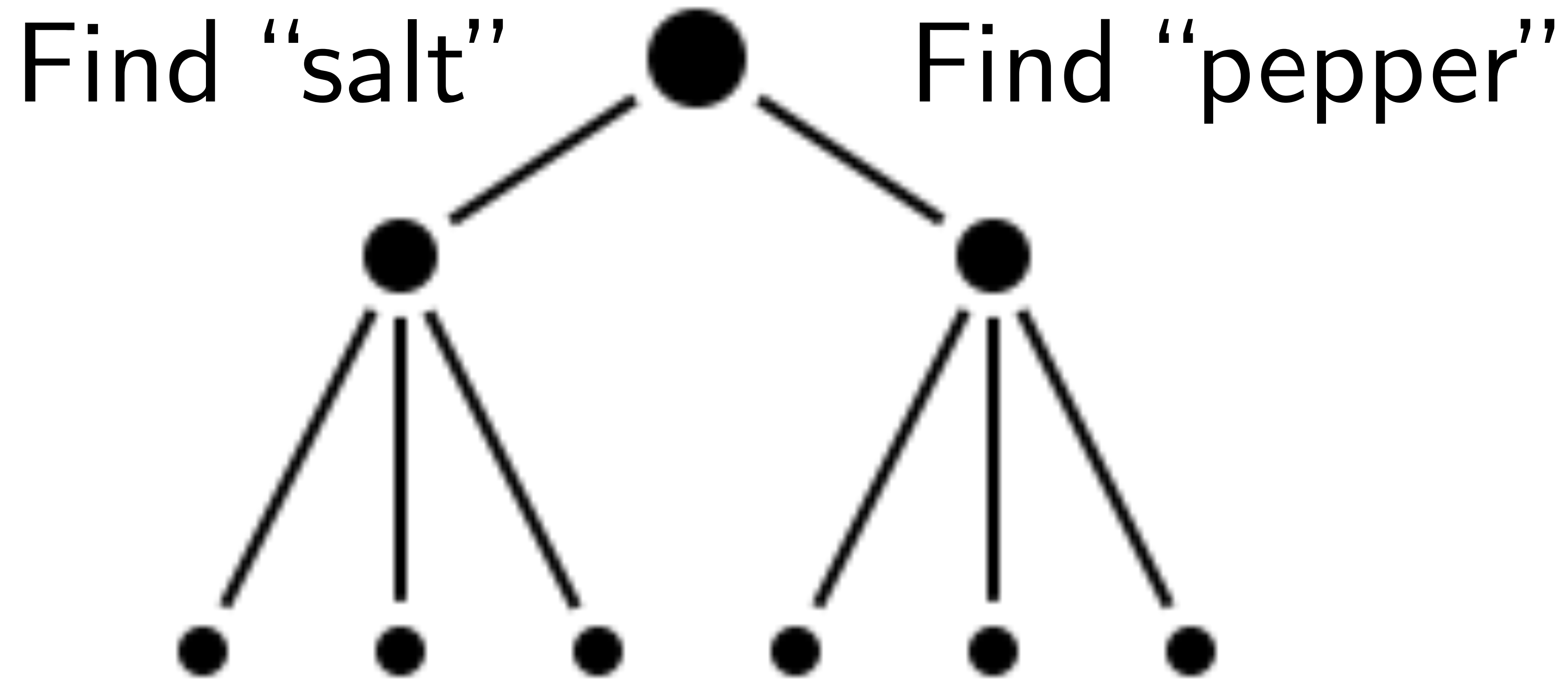
1. Failure to generalize to different human utterances
2. Failure to capture common sense
3. Failure to capture complex instructions (while loops)

Two Fundamental Challenges

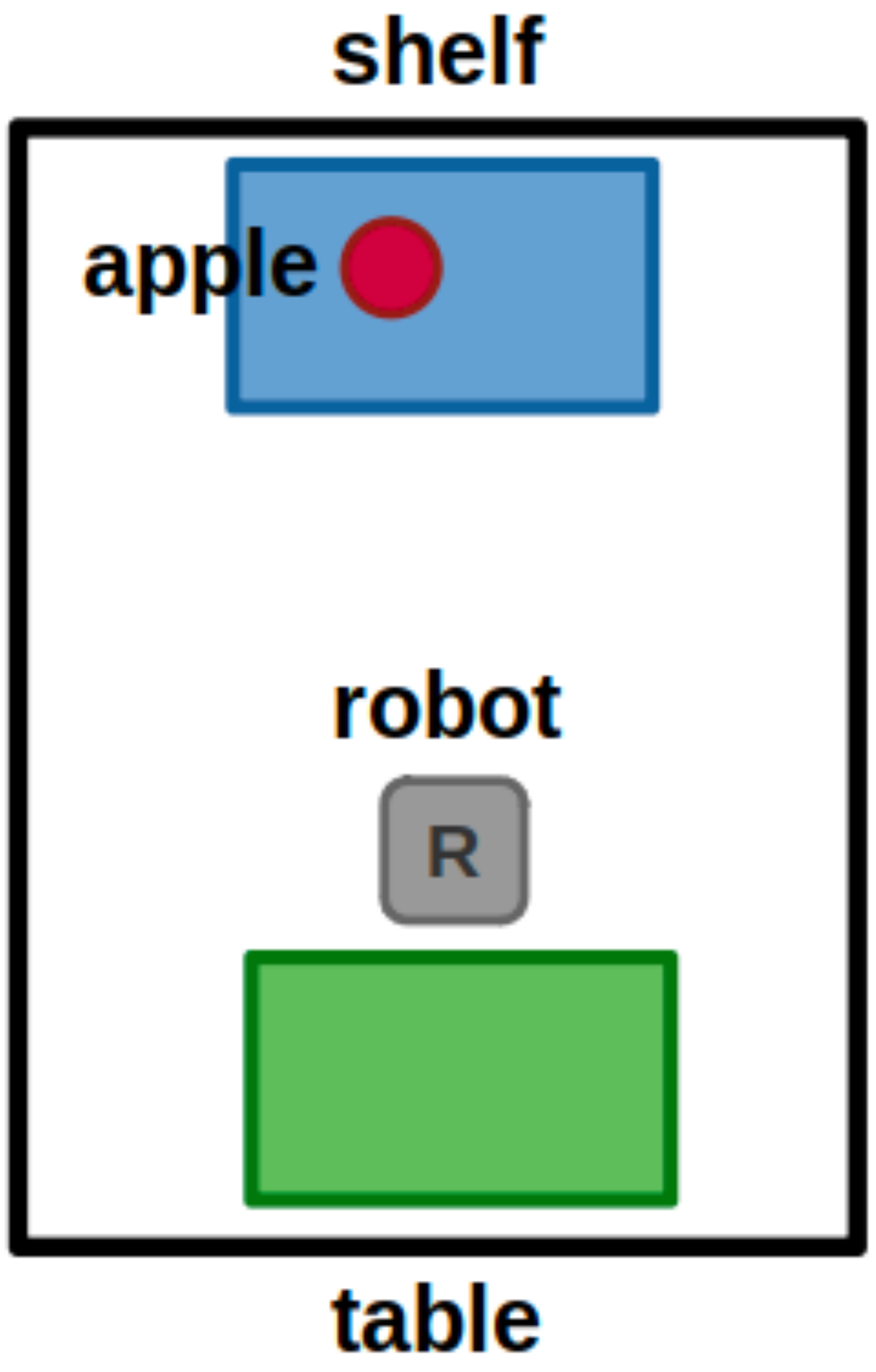
Challenge 1:
Ground natural language
in robot state



Challenge 2:
Planning actions to
solve a task

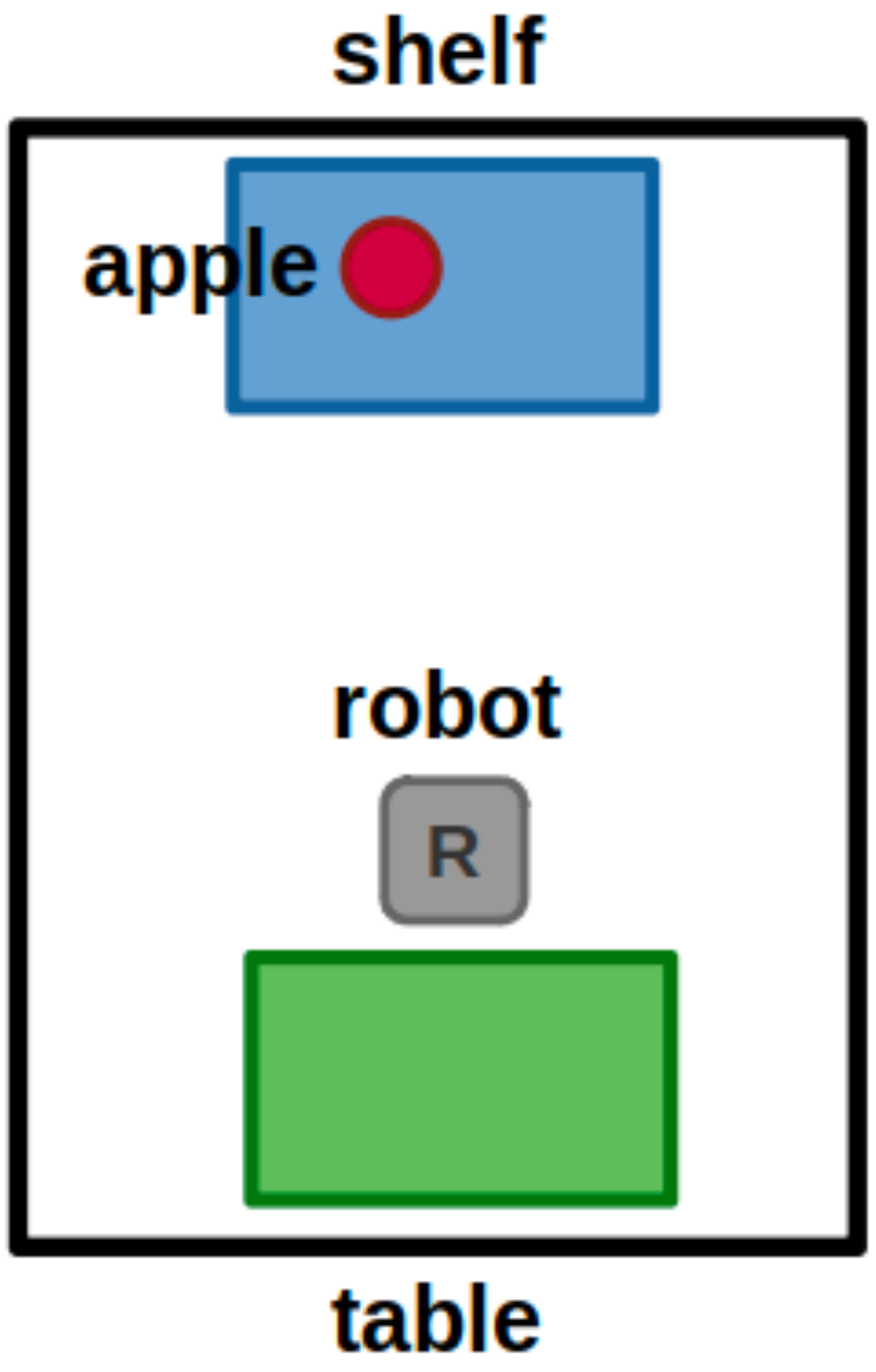


What is task planning? Why is it hard?



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

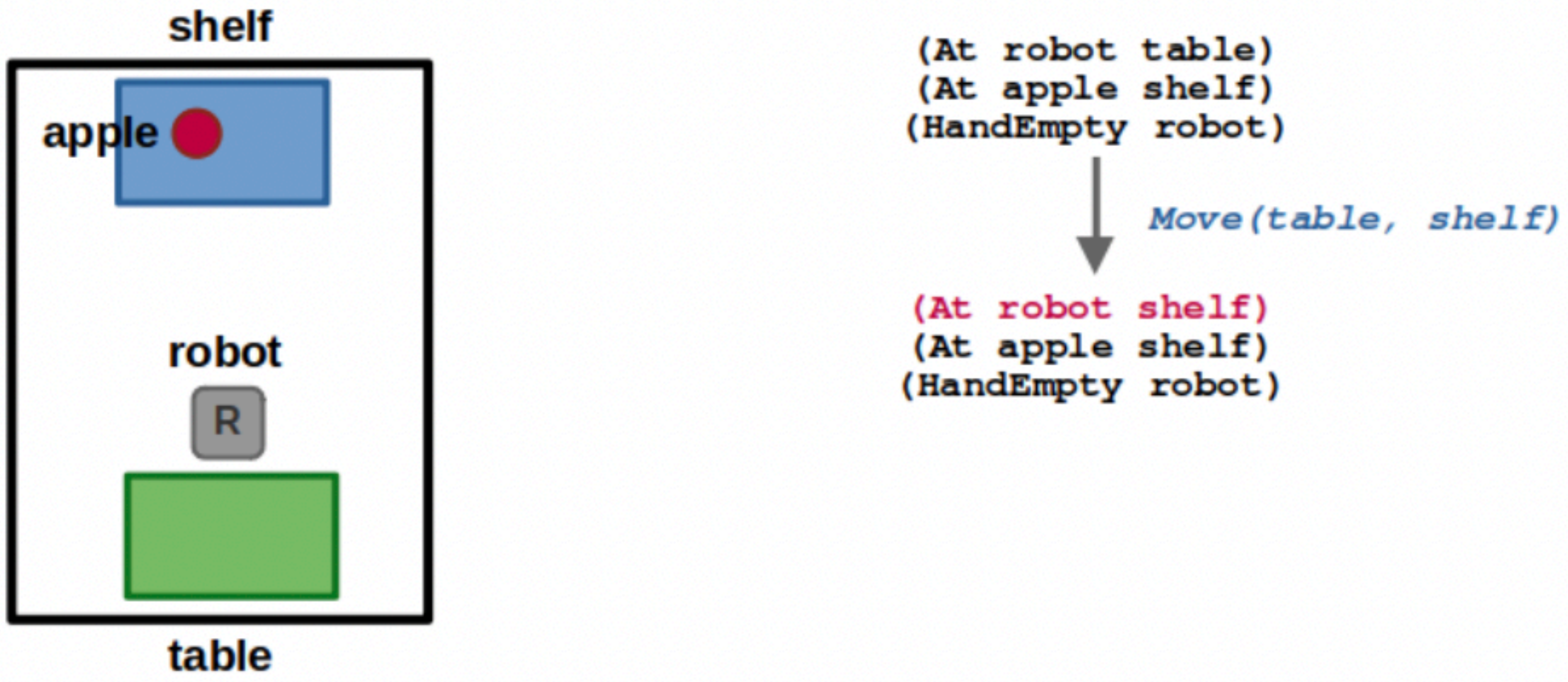
What is task planning? Why is it hard?



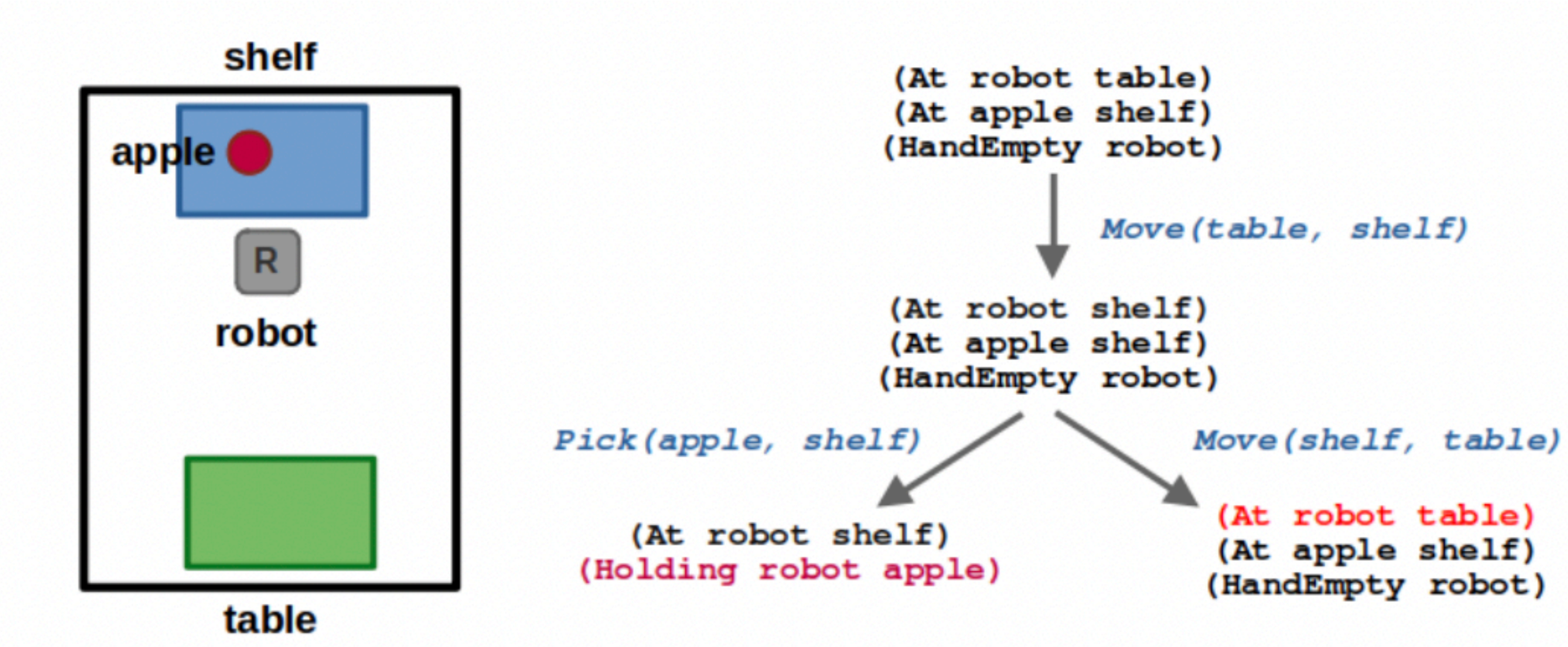
Take the apple from the shelf and put it on the table

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

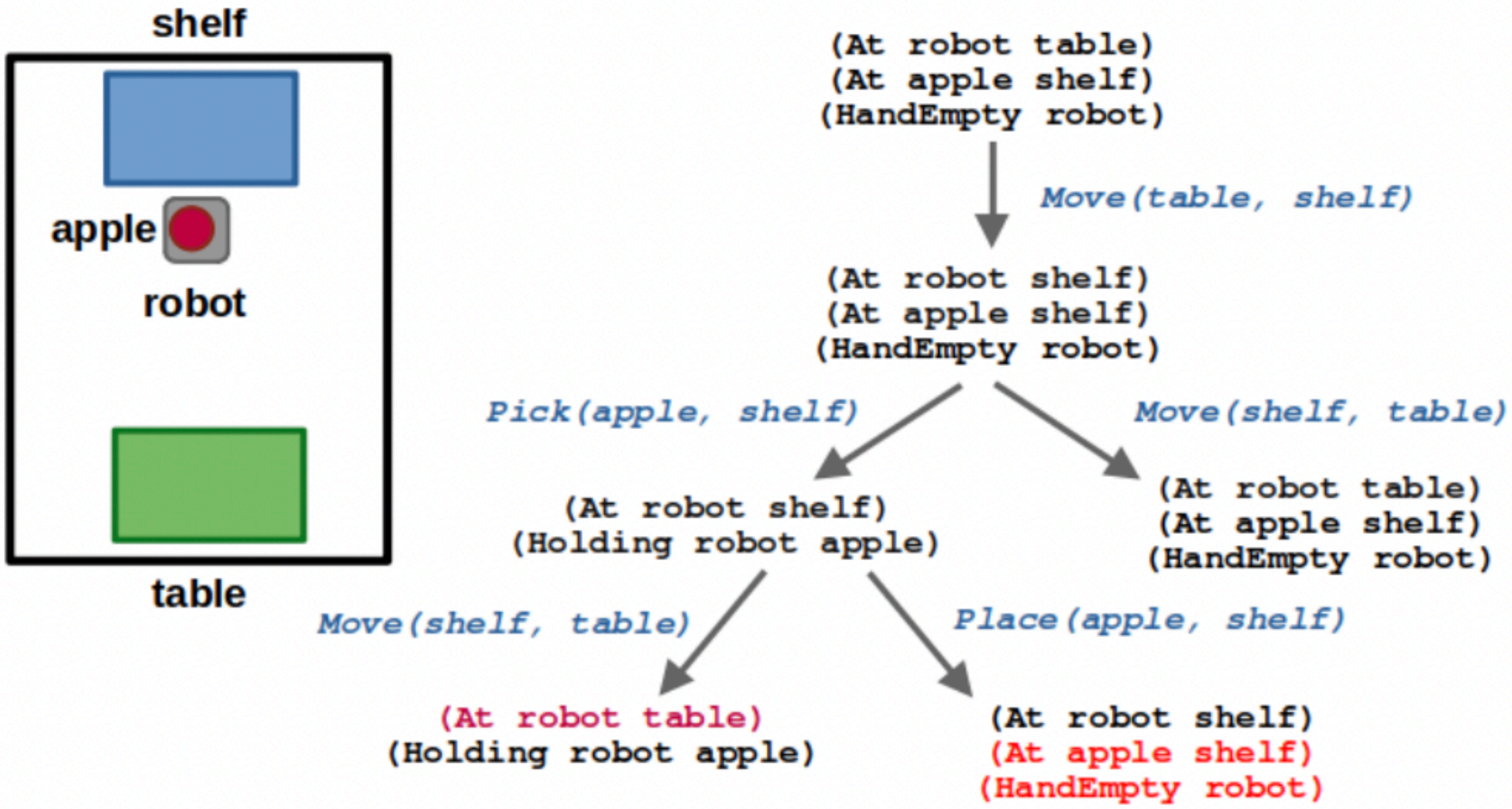
What is task planning? Why is it hard?



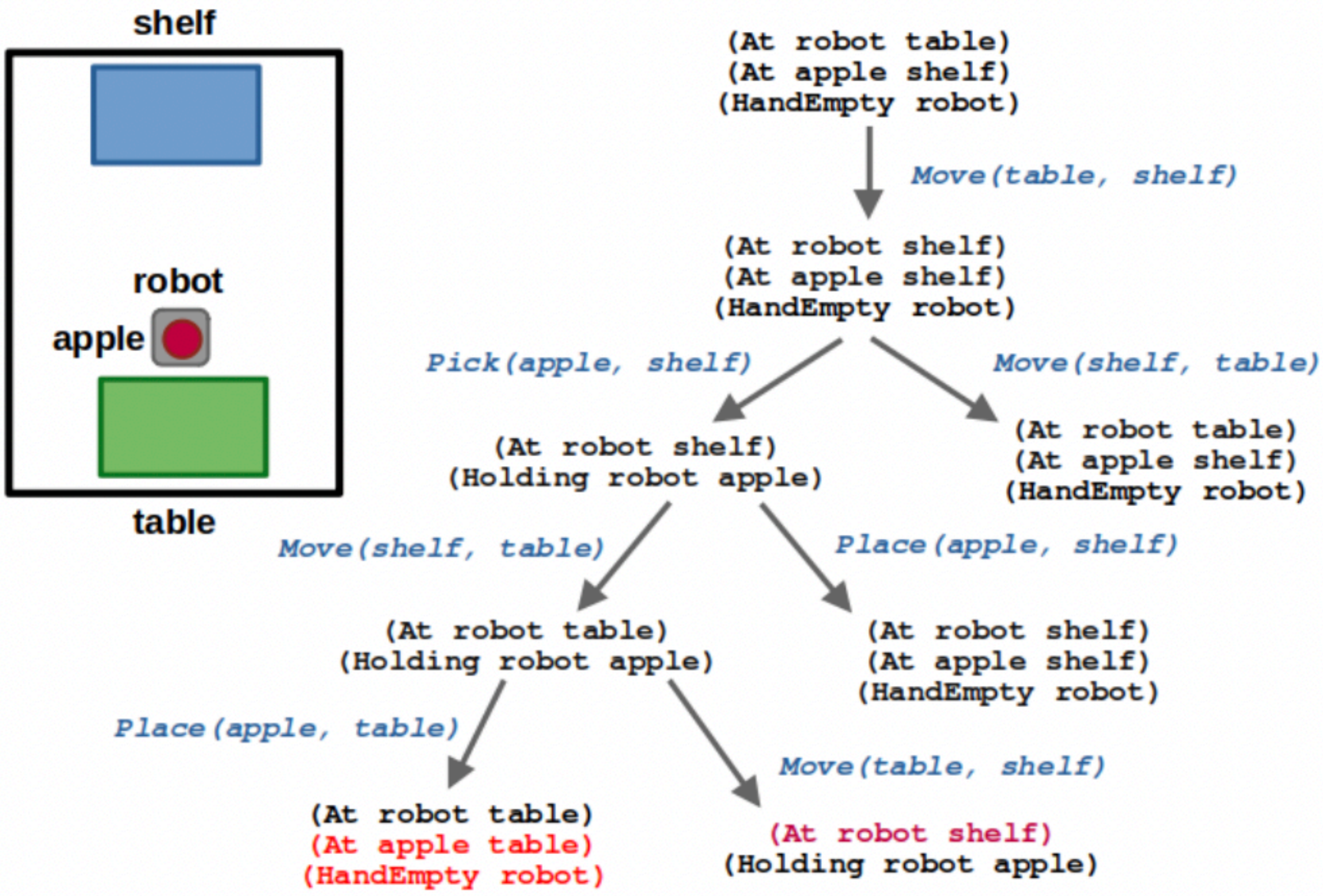
What is task planning? Why is it hard?



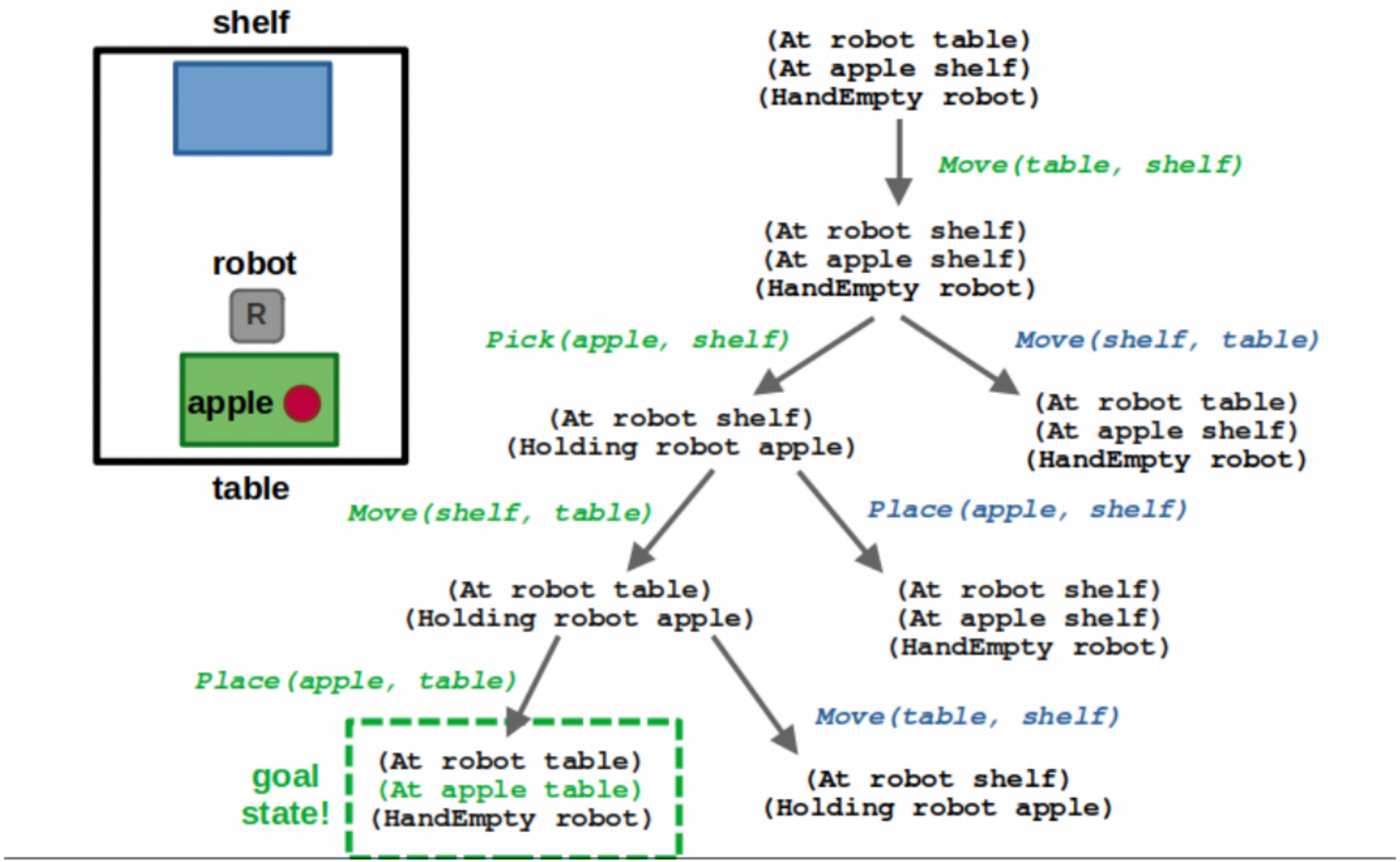
What is task planning? Why is it hard?



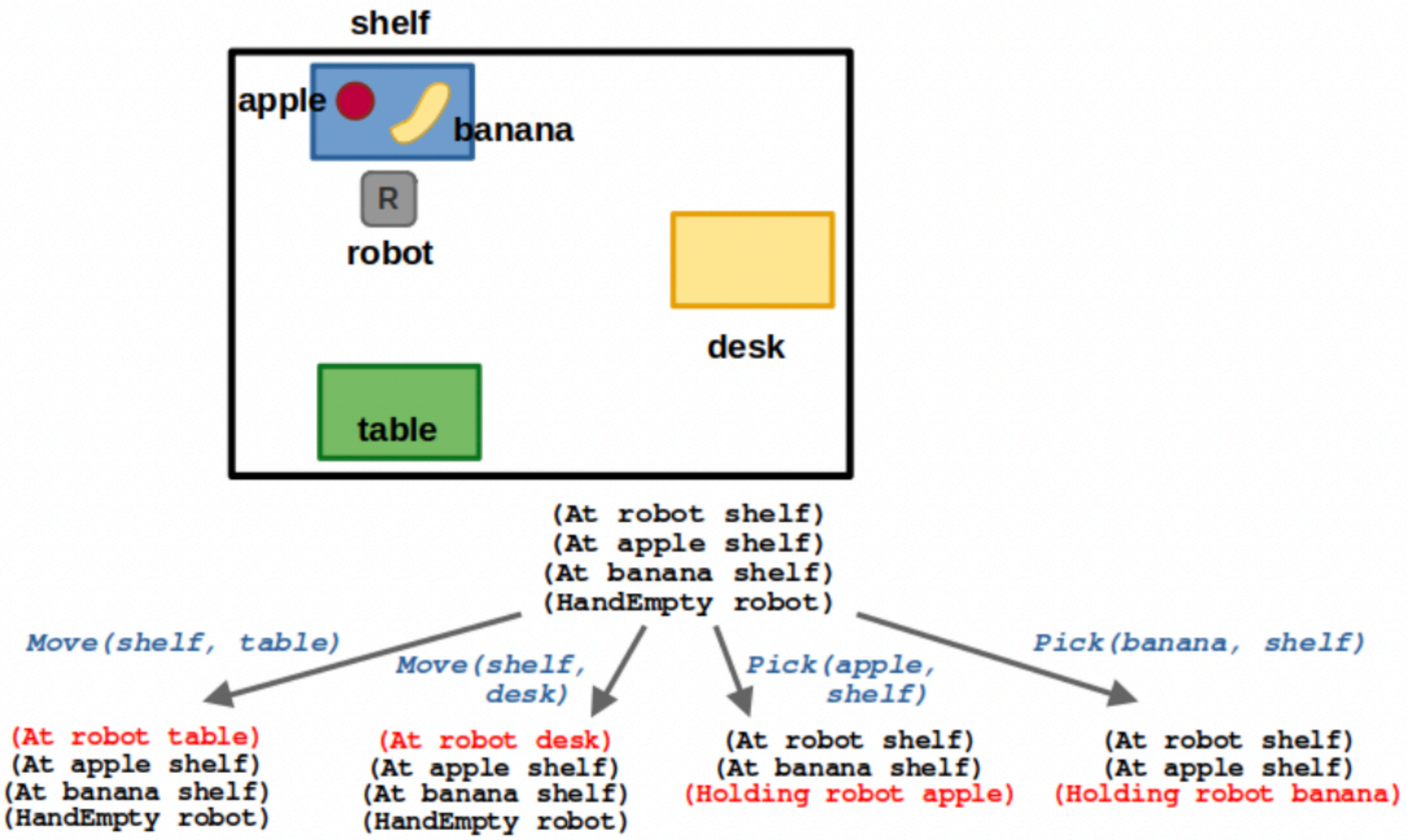
What is task planning? Why is it hard?



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

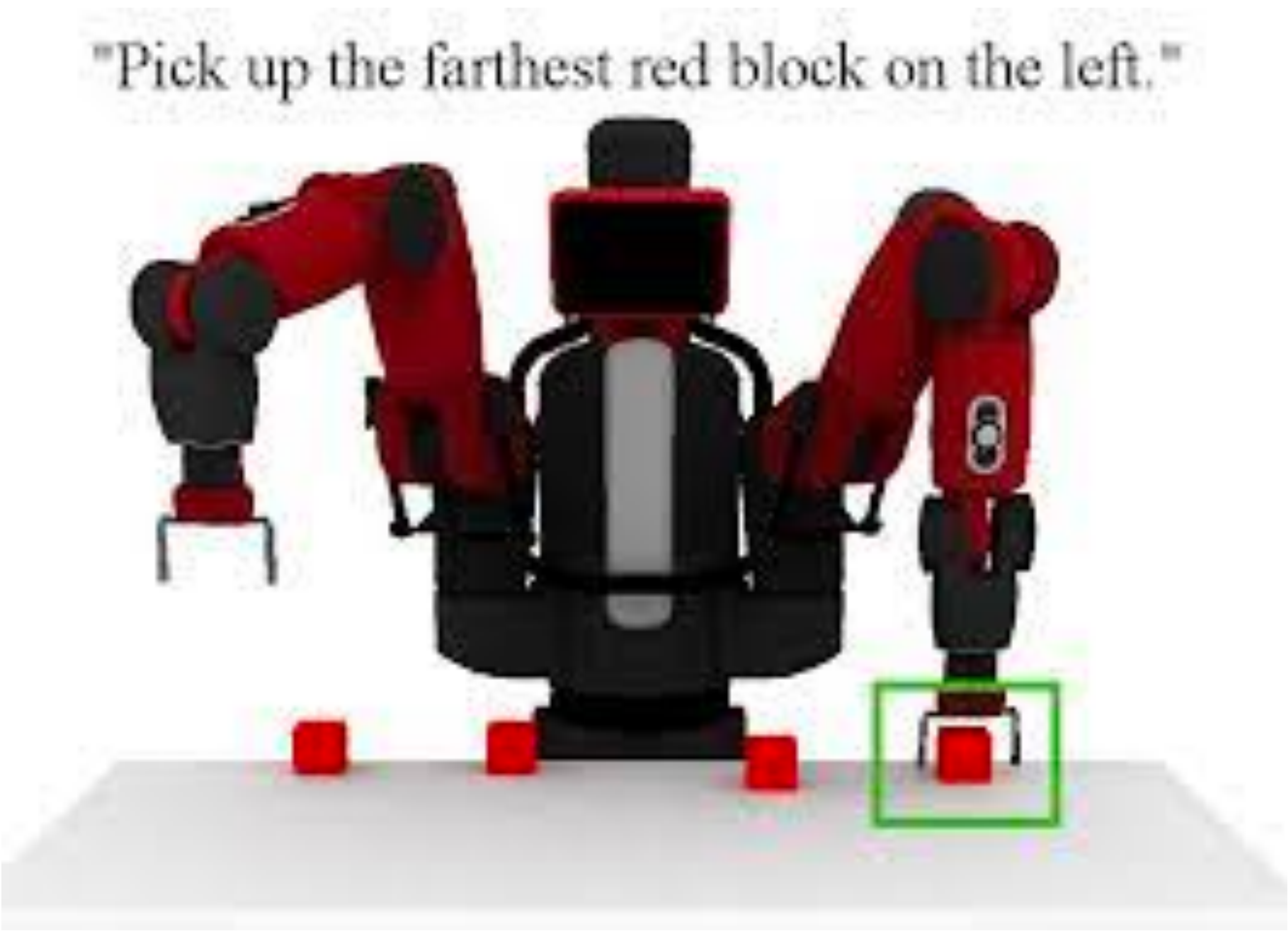
Why did it not scale?

Combinatorially large search tree

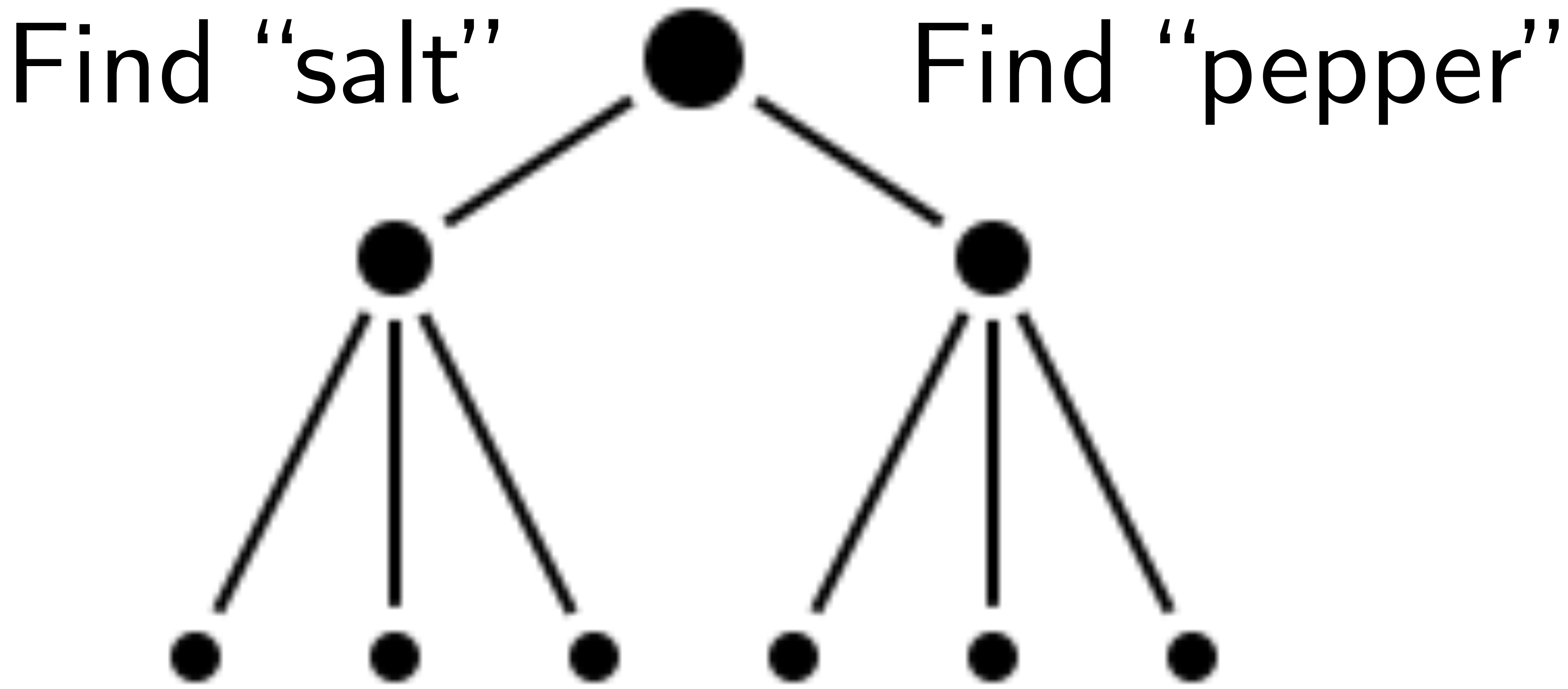
Had no notion of common sense

Two Fundamental Challenges

Challenge 1:
Ground natural language in robot state



Challenge 2:
Planning actions to solve a task



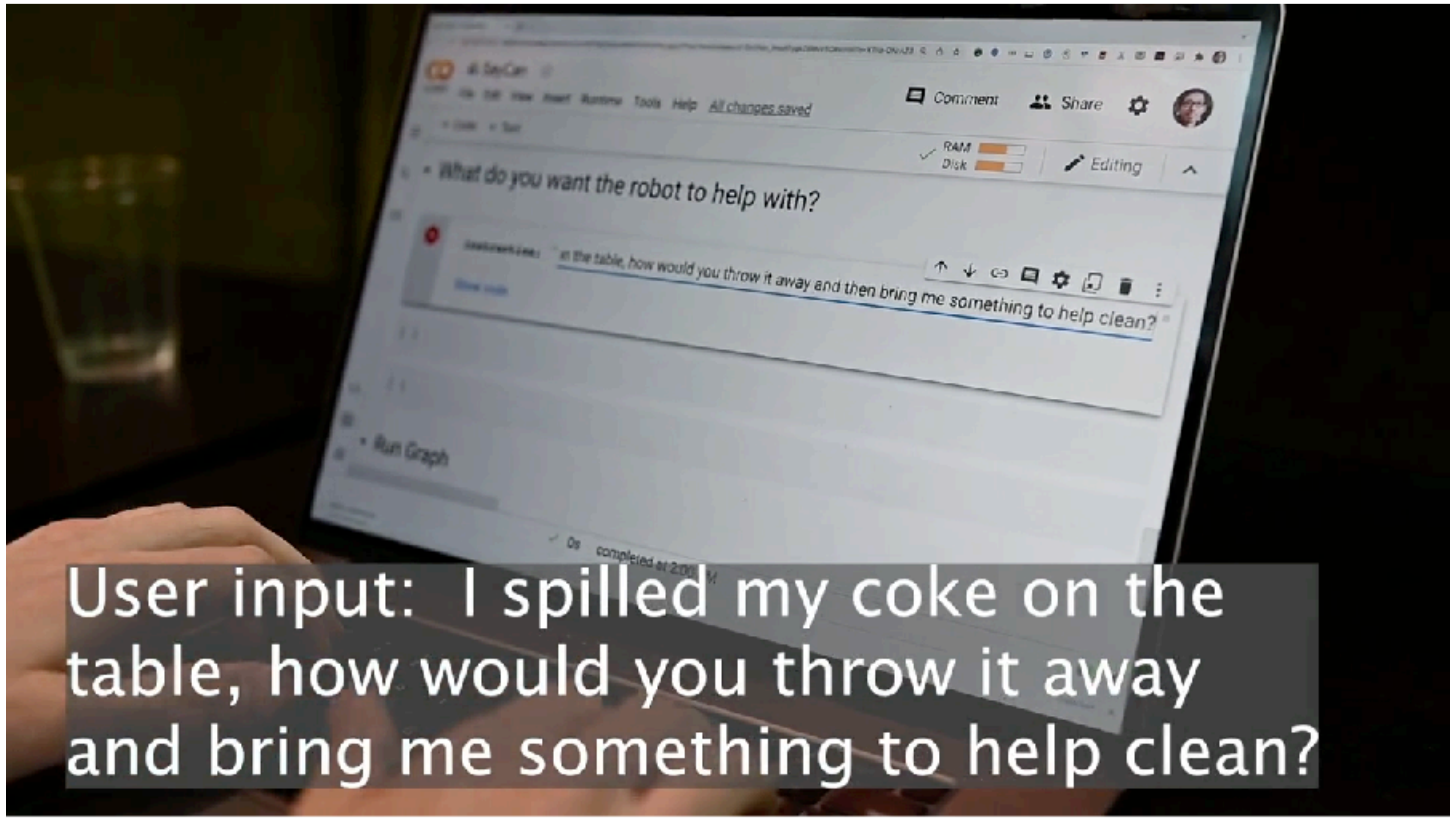
LARGE LANGUAGE MODELS

Episode IV

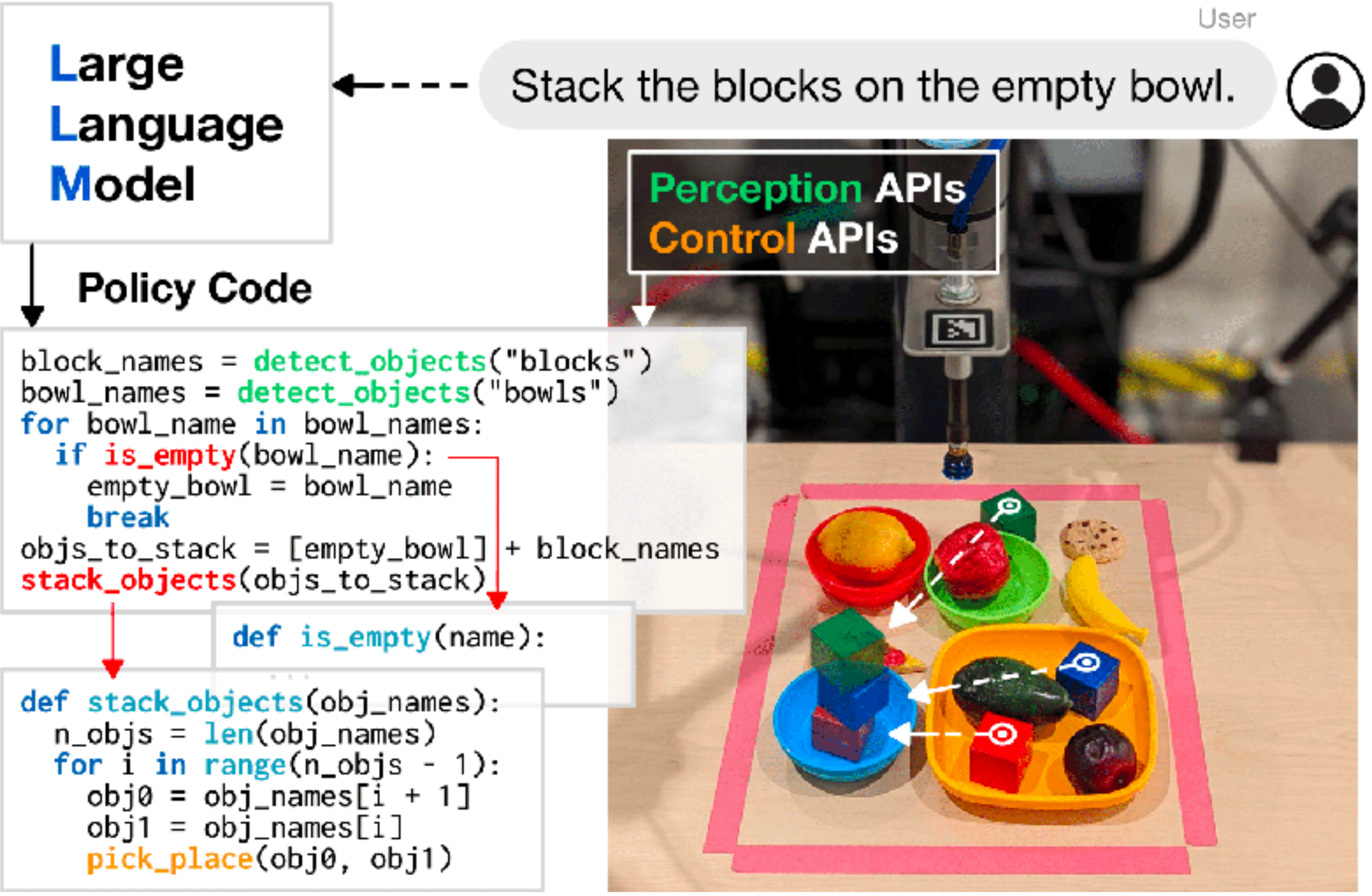
A NEW HOPE

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

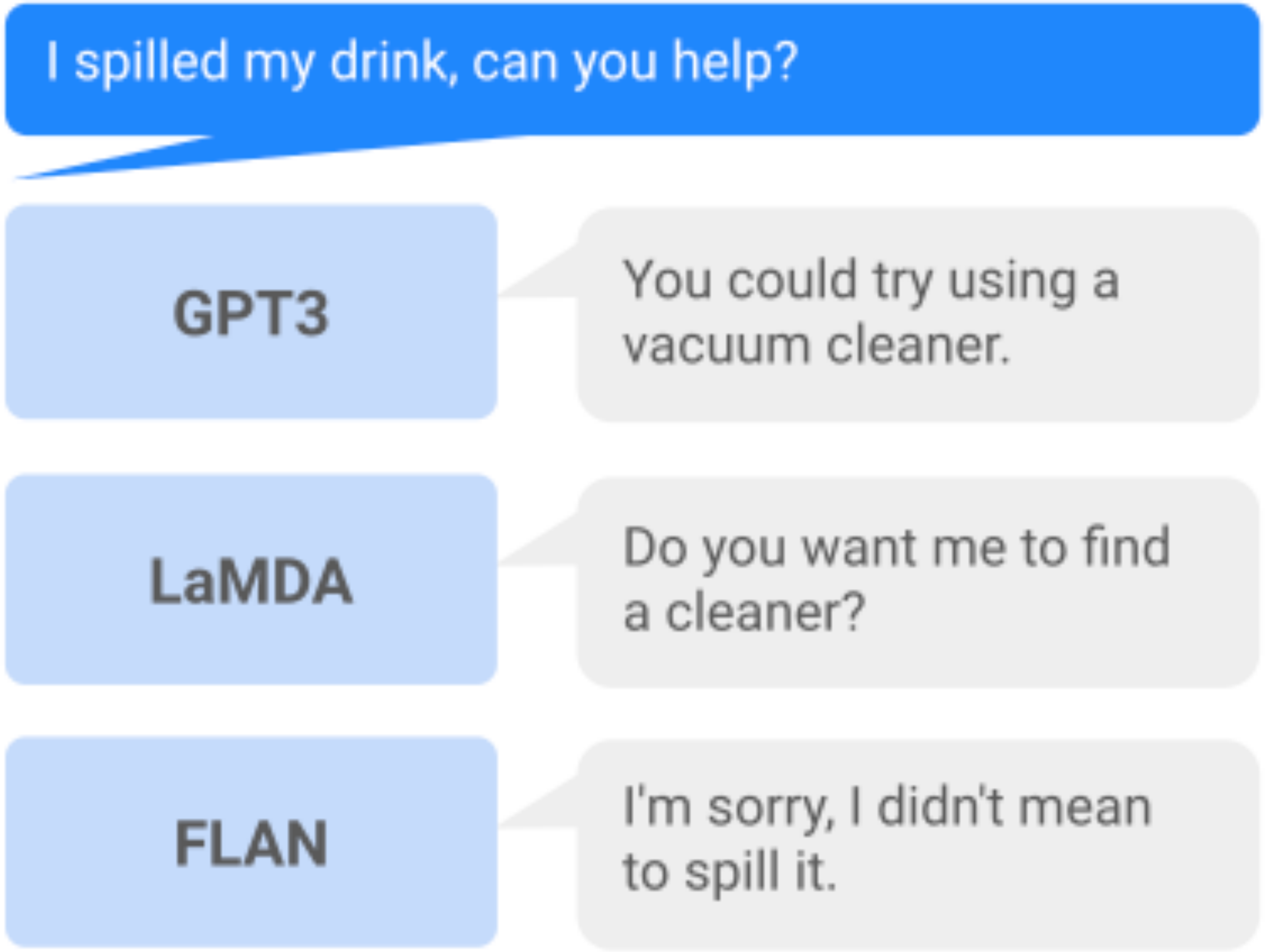
Michael Ahn* Anthony Brohan* Noah Brown* Yevgen Chebotar* Omar Cortes* Byron David* Chelsea Finn*
Chuyuan Fu* Keerthana Gopalakrishnan* Karol Hausman* Alex Herzog* Daniel Ho* Jasmine Hsu* Julian Ibarz*
Brian Ichter* Alex Irpan* Eric Jang* Rosario Jauregui Ruano* Kyle Jeffrey* Sally Jesmonth* Nikhil Joshi*
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


CANS & BOTTLES

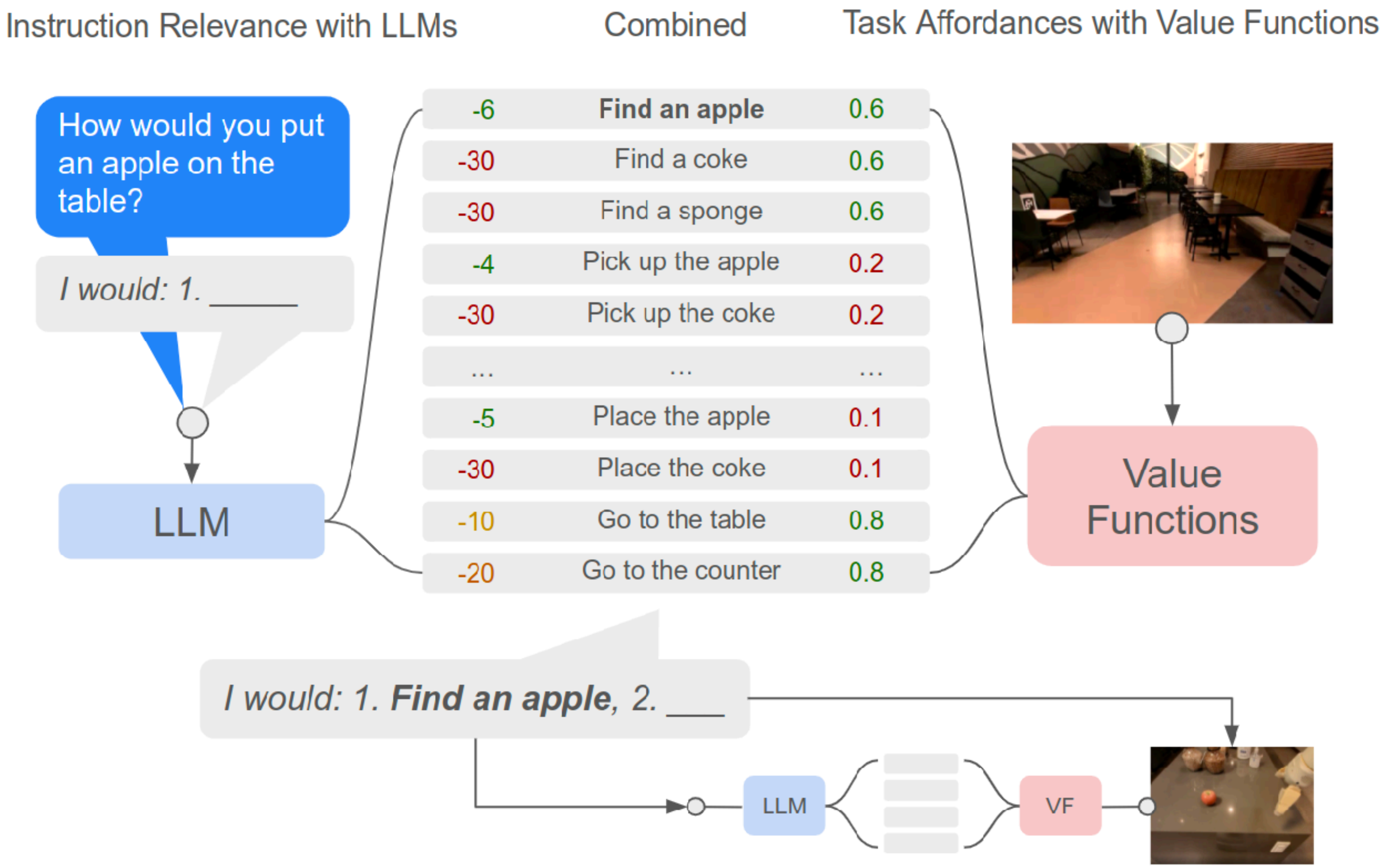
So … we just ask an LLM to tell us what to do?

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

Learn Robot Policies

Use LLMs to plan

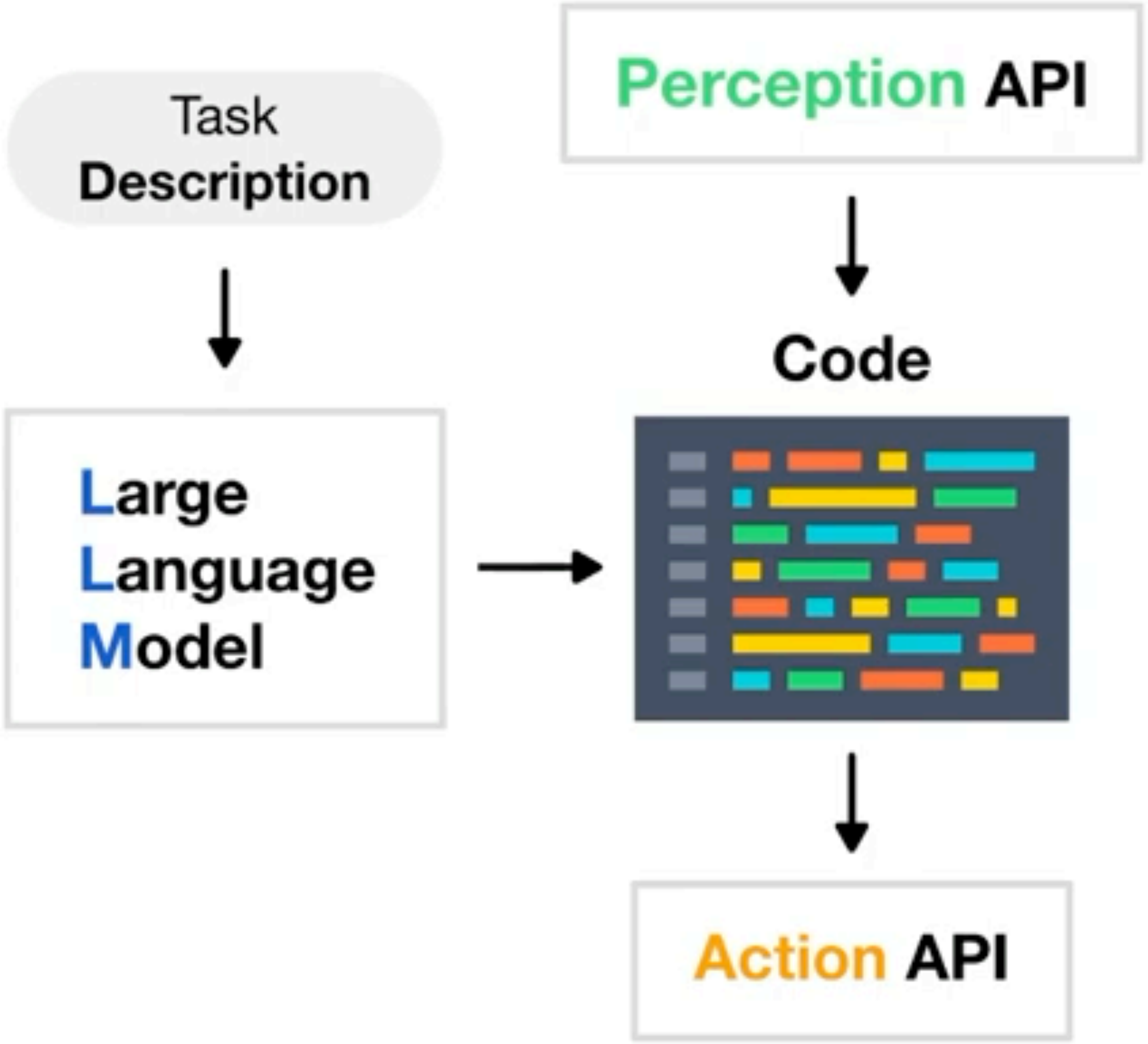
Ours: Use LLMs to write robot code

Why choose code as a representation?

Interpretable

Verifiable

Composable



Ours: Use LLMs to write robot code

Large Language Model

User

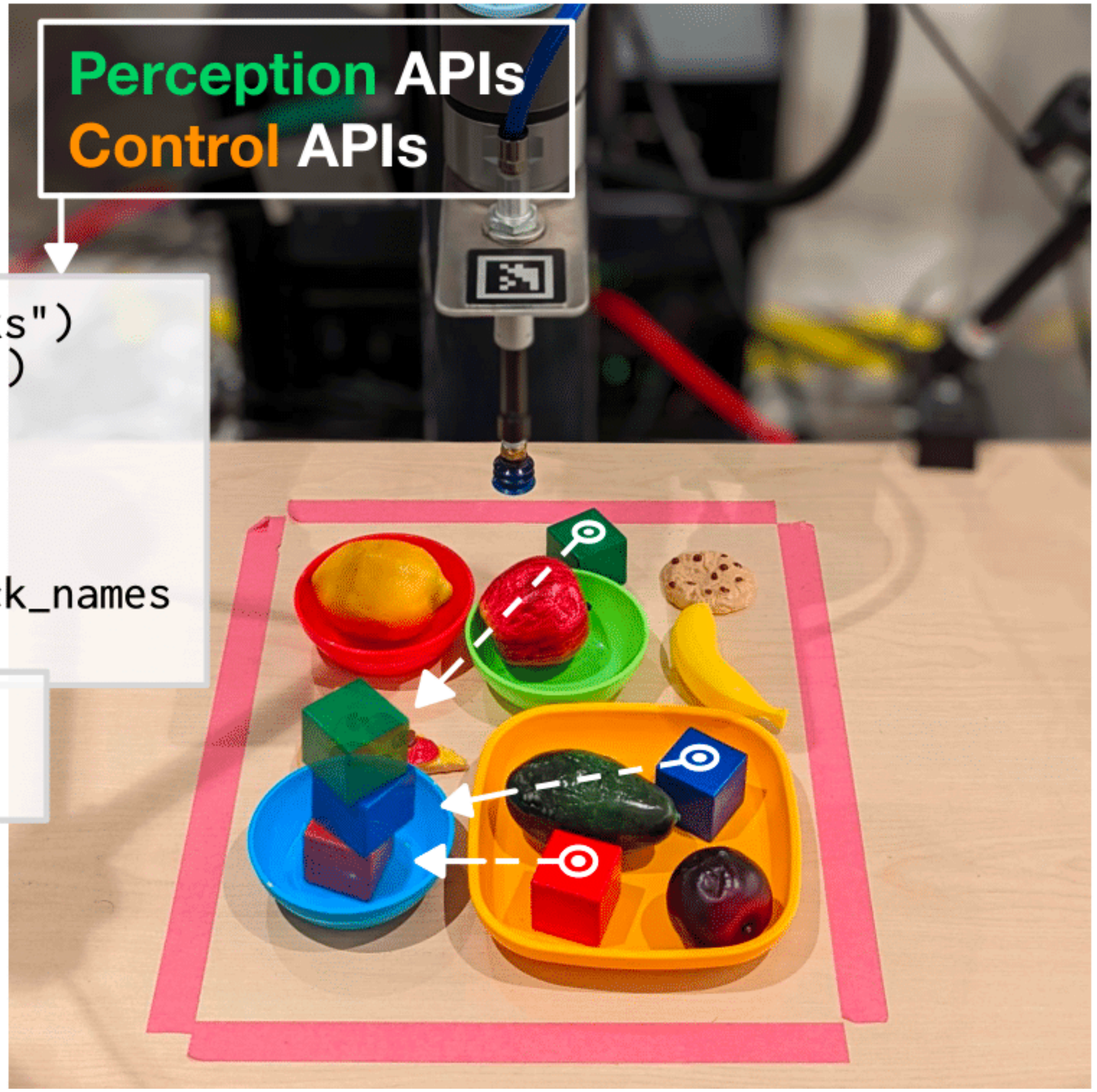
Stack the blocks on the empty bowl.

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

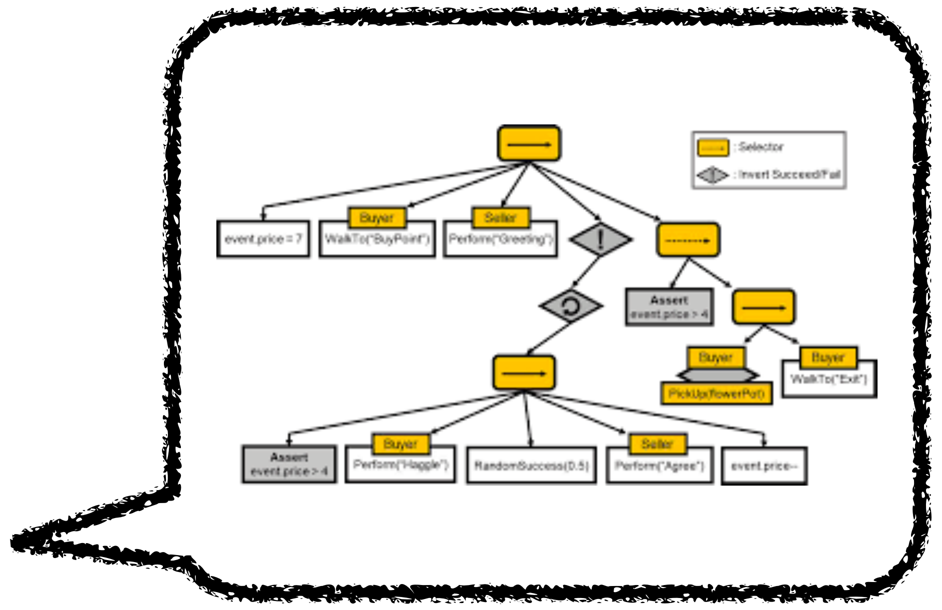
Huaxiaoyue Wang, Gonzalo Gonzalez-Pumariega, Yash Sharma, Sanjiban Choudhury
Cornell University

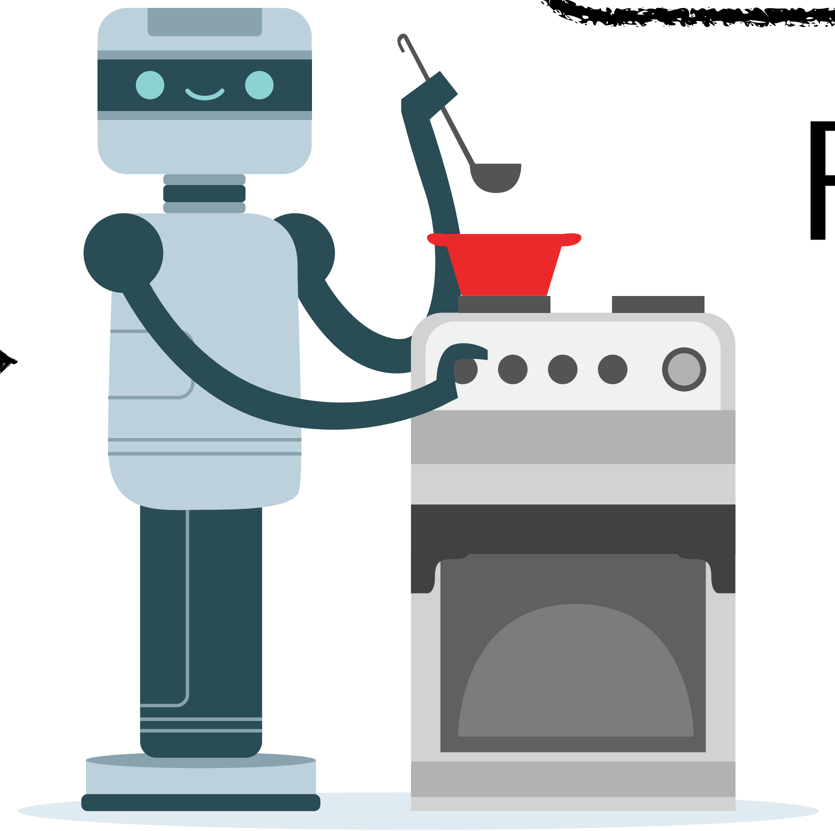
User Story: Helping Grandma in the kitchen

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."





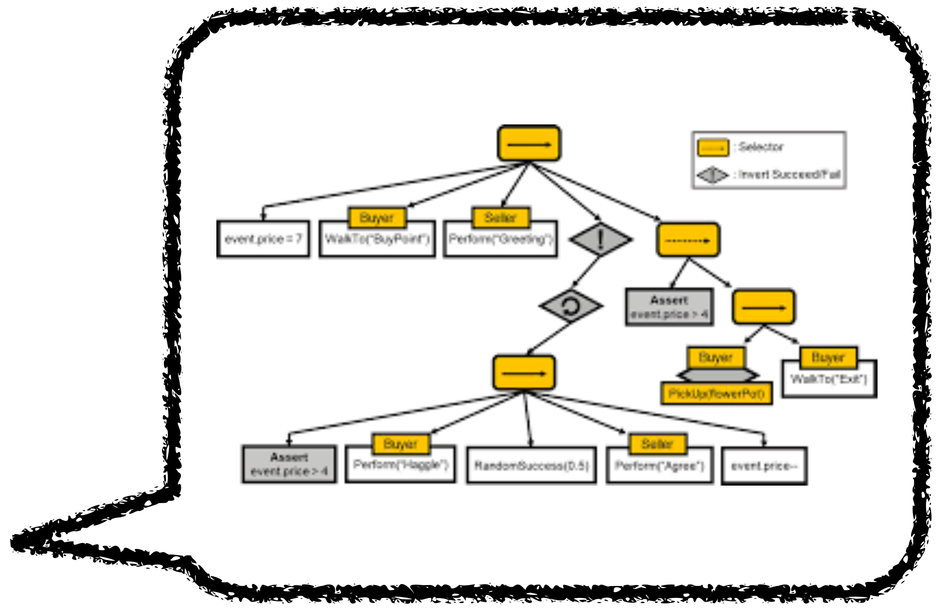
Robot Code

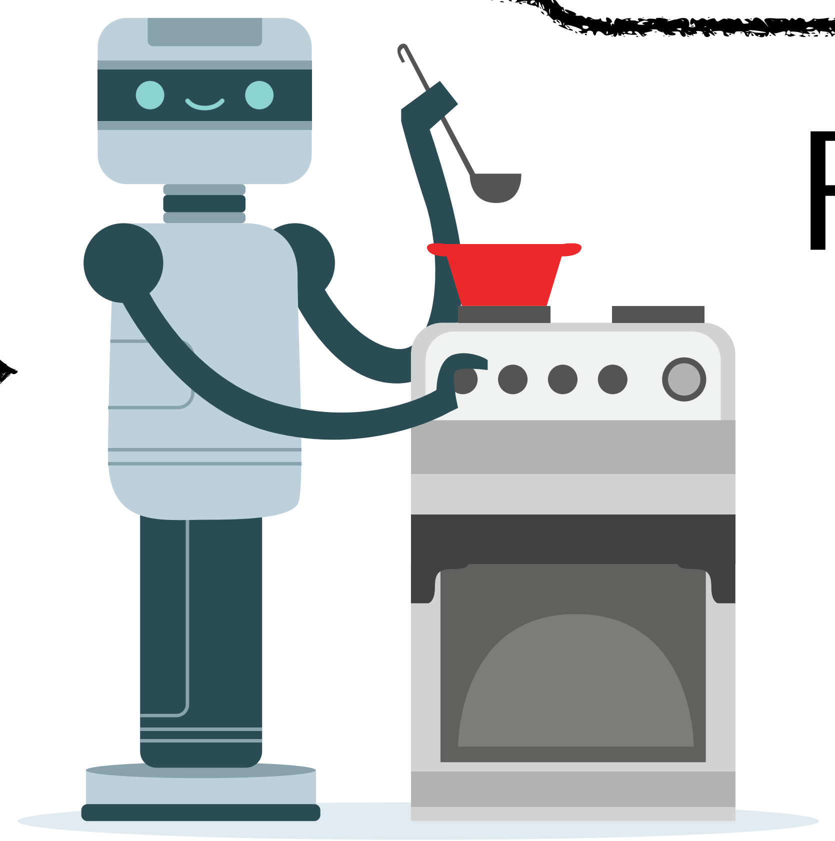
User Story: Helping Grandma in the kitchen

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

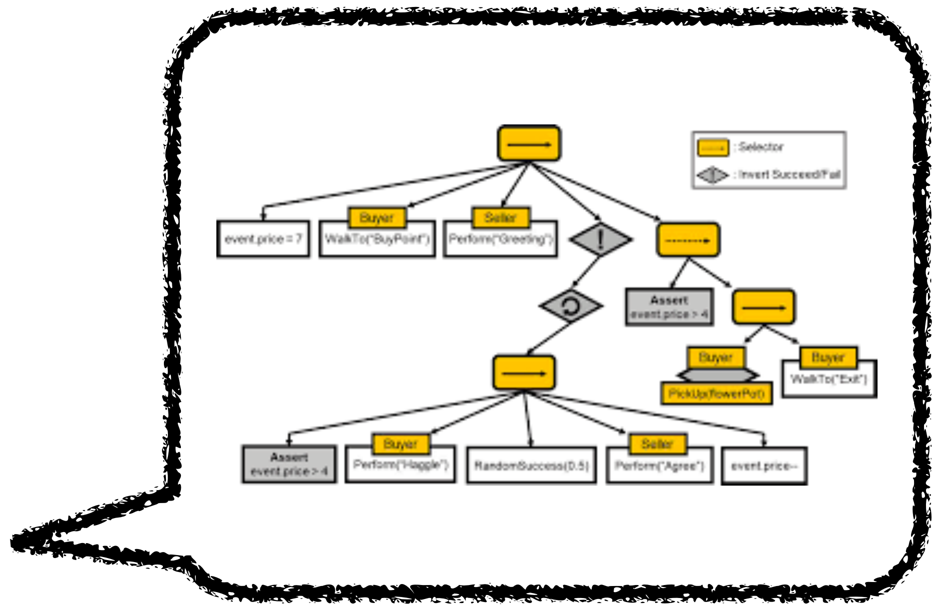
- ✗ Lacks specificity (e.g. Heat up water how? Pour rice where?)
- ✗ Leaves out implicit preferences (e.g. Personal style of stirring?)

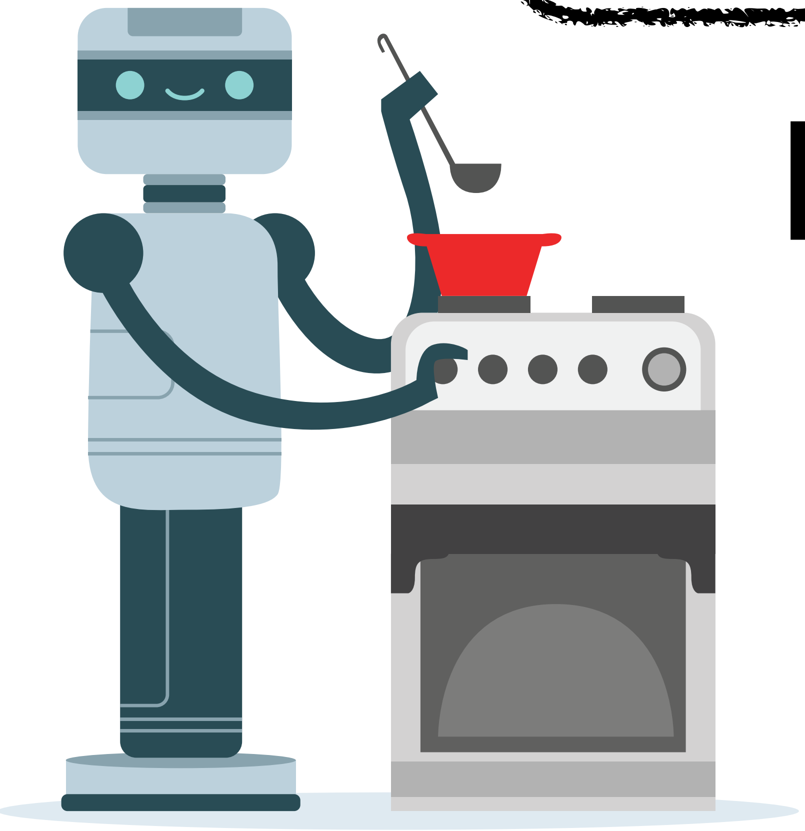
User Story: Helping Grandma in the kitchen

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 Horizon Demonstrations

Long-horizon tasks can have >=hundreds of states

[Damen et al '18]

Multiple such demonstrations

state_1

state_2

⋮

state_T

state_1

state_2

⋮

state_T

……

Naively concatenating demonstrations will easily exhaust context length!

Challenge 2: Complex Task Code

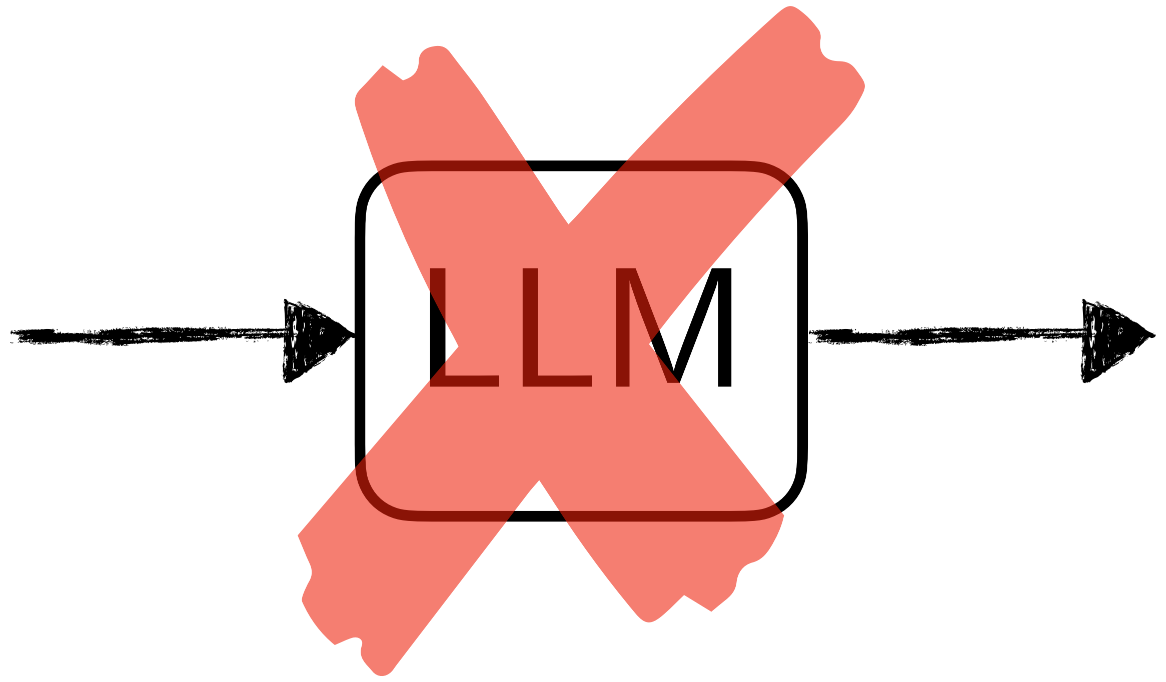
Loops, checks, and calls to custom robot libraries ..

```python
def main():
    # 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 tomatoes in the kitchen.
    tomatoes = get_all_obj_names_that_match_type('tomato')
    # 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 top buns in the kitchen.
    top_buns = get_all_obj_names_that_match_type('top bun')
    # Get a list of all the stoves in the kitchen.
    stoves = get_all_location_names_that_match_type('stove')
    # Get a list of all the cutting boards in the kitchen.
    cutting_boards = get_all_location_names_that_match_type('cutting_boar

    # Decide a stove to use.
    stove_to_cook_at = stoves[0]
    # Cook a patty at that stove. Decide a patty to cook.
    patty_to_cook = patties[0]
    cook_object_at_location(obj=patty_to_cook, location=stove_to_cook_at)

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

    # Decide a tomato to use.
    tomato_to_use = tomatoes[0]
    # Cut that tomato at the cutting board.
    cut_object_at_location(obj=tomato_to_use, location=cutting_boards[0])
    # Stack the tomato on top of the patty. obj1 should be the tomato, ob
    # should be the patty on top of the bottom bun.
    stack_obj1_on_obj2(obj1=tomato_to_use, obj2=patty_to_cook)

    # Decide a lettuce to use.
    lettuce_to_use = lettuces[0]
    # Cut that lettuce at the cutting board.
    cut_object_at_location(obj=lettuce_to_use, location=cutting_boards[0])
    # Stack the lettuce on top of the tomato. obj1 should be the lettuce,
    # should be the tomato on top of the patty on top of the bottom bun.
    stack_obj1_on_obj2(obj1=lettuce_to_use, obj2=tomato_to_use)

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

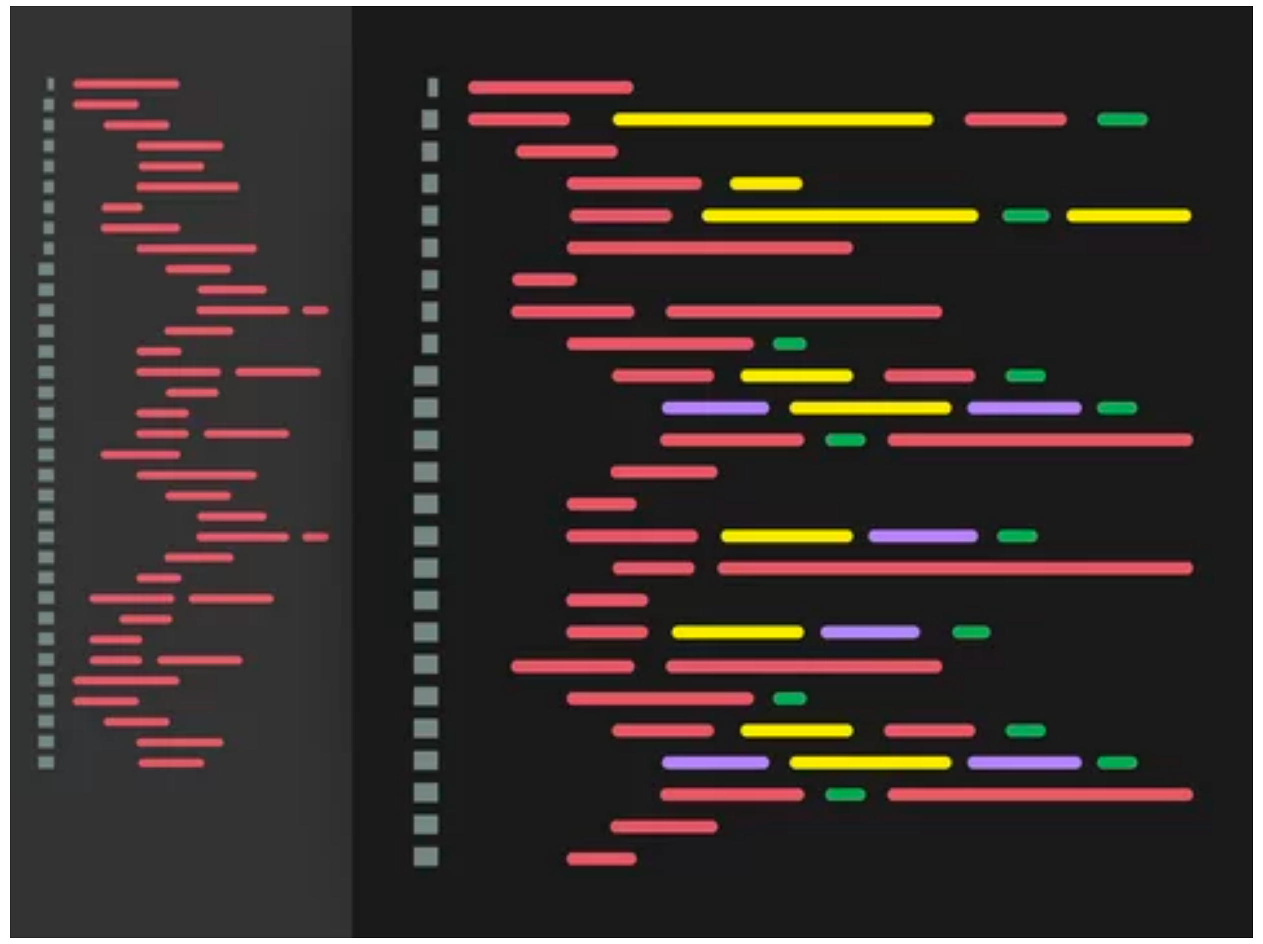
```python
def cook_object_at_location(obj, location):
    # To cook an object, the robot first needs to be holding obj
    if not is_holding(obj):
        # If the robot is not holding obj, there are 2 scenarios:
        #   (1) if obj is in a stack ,unstack obj
        #   (2) else, pick up obj.
        if is_in_a_stack(obj):
            # Because obj is in a stack, robot need to move then unstack the
            obj from the obj_at_bottom first
            obj_at_bottom = get_obj_that_is_underneath(obj_at_top=obj)
            move_then_unstack(obj_to_unstack=obj, obj_at_bottom=obj_at_botto
                unstack_location=get_obj_location(obj_at_bottom))
        else:
            # Since obj is not in a stack, robot can just move then pick it
            move_then_pick(obj=obj)
    # place the object at the location to cook at
    move_then_place(obj=obj, place_location=location)
    # cook the object
    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 th
            obj from the obj_at_bottom first
            obj_at_bottom = get_obj_that_is_underneath(obj_at_top=obj1)
            move_then_unstack(obj_to_unstack=obj1, obj_at_bottom=obj_at_bott
                unstack_location=get_obj_location(obj_at_bottom))
        else:
            # Since obj1 is not in a stack, robot can just move then pick it
            move_then_pick(obj=obj1)
    # determine the location of obj2 to stack on
    obj2_location = get_obj_location(obj2)
    # move to obj2's location then 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
        #   (2) else, pick up obj.
        if is_in_a_stack(obj):
            # Because obj is in a stack, robot need to move then unstack the
            obj from the obj_at_bottom first
            obj_at_bottom = get_obj_that_is_underneath(obj_at_top=obj)
```

```python
def move_then_unstack(obj_to_unstack, obj_at_bottom, unstack_loca
    # For unstacking, we need to move to the location of the bot
    if get_curr_location() != get_obj_location(obj_at_bottom):
        move(get_curr_location(), get_obj_location(obj_at_bottom)
    unstack(obj_to_unstack, obj_at_bottom)
    # After unstacking, we need to move to the unstack_location
    if get_curr_location() != unstack_location:
        move(get_curr_location(), unstack_location)

def move_then_pick(obj):
    obj_location = get_obj_location(obj)
    if get_curr_location() != obj_location:
        move(get_curr_location(), obj_location)
    pick_up(obj, obj_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)
```

Challenge 1:
Long Horizon Demonstrations



Challenge 2:
Complex Task Code



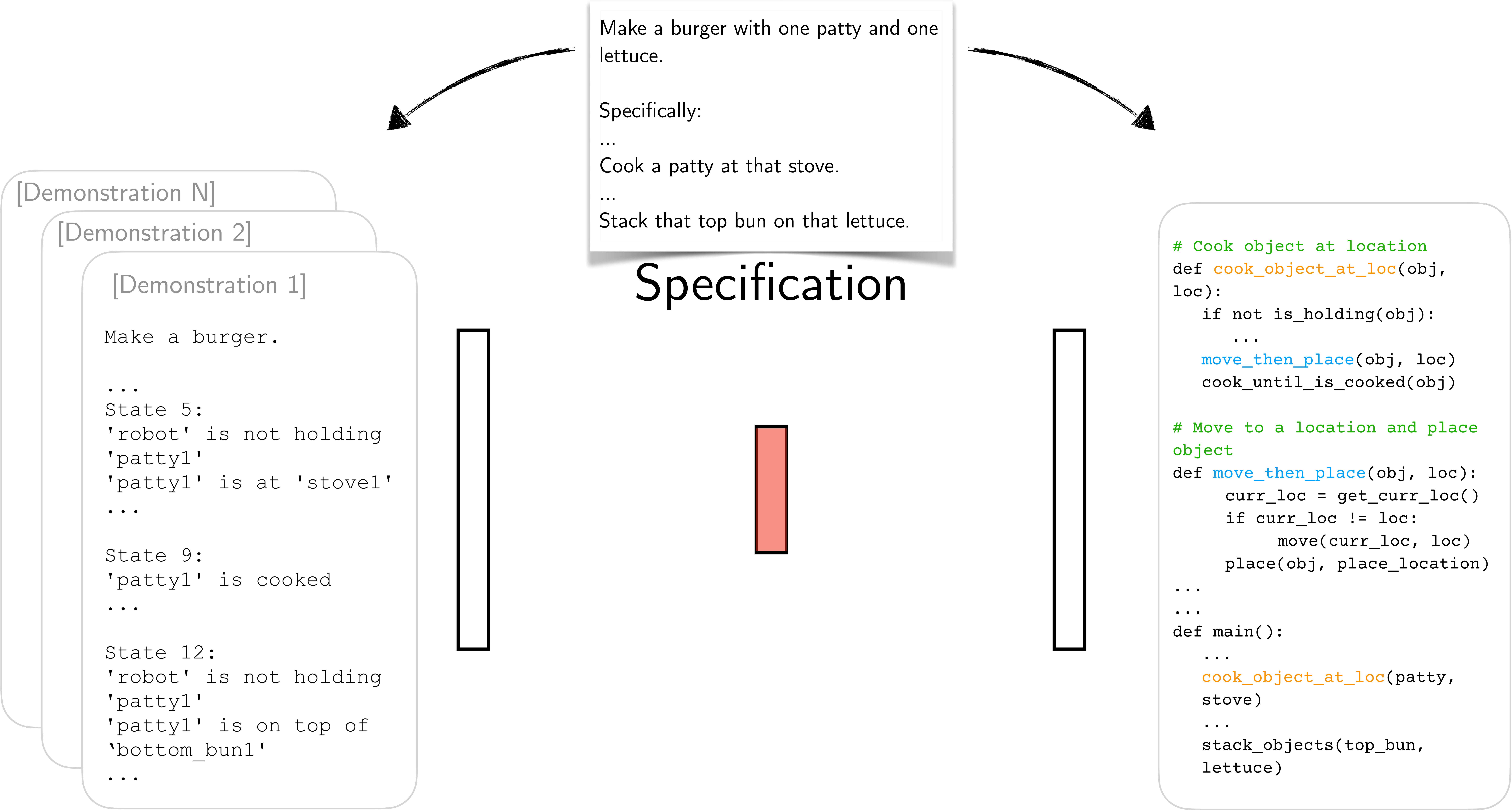
Directly generating code from demonstrations is intractable!

Both

demonstration and code

share a *latent*, *compact*,

specification



Make a burger with one patty and one lettuce.
Specifically:
...
Cook a patty at that stove.
...
Stack that top bun on that lettuce.
Specification
[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)

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)
```

Key Insight: Extended chain-of-thought

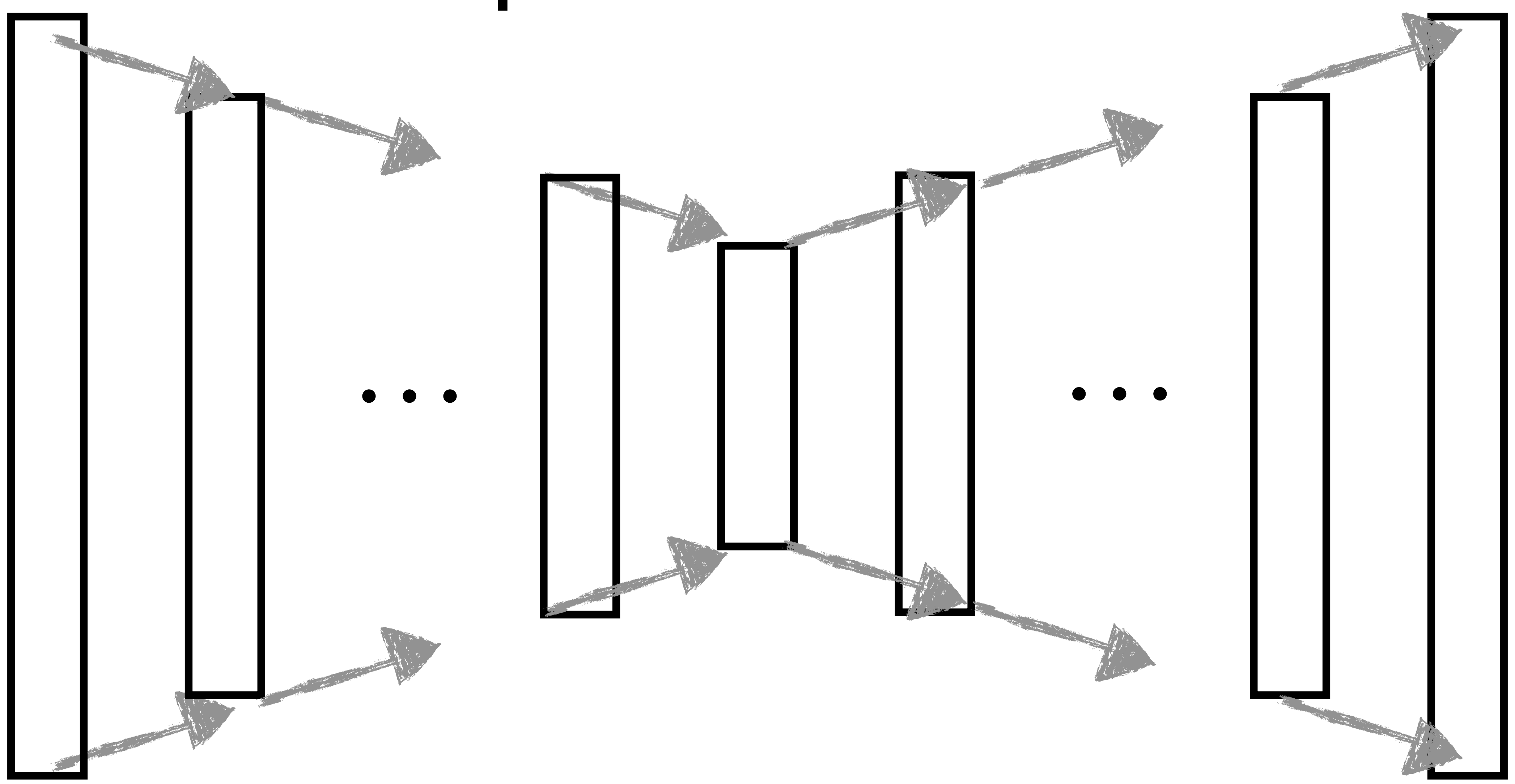
[Demonstration N]

[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.

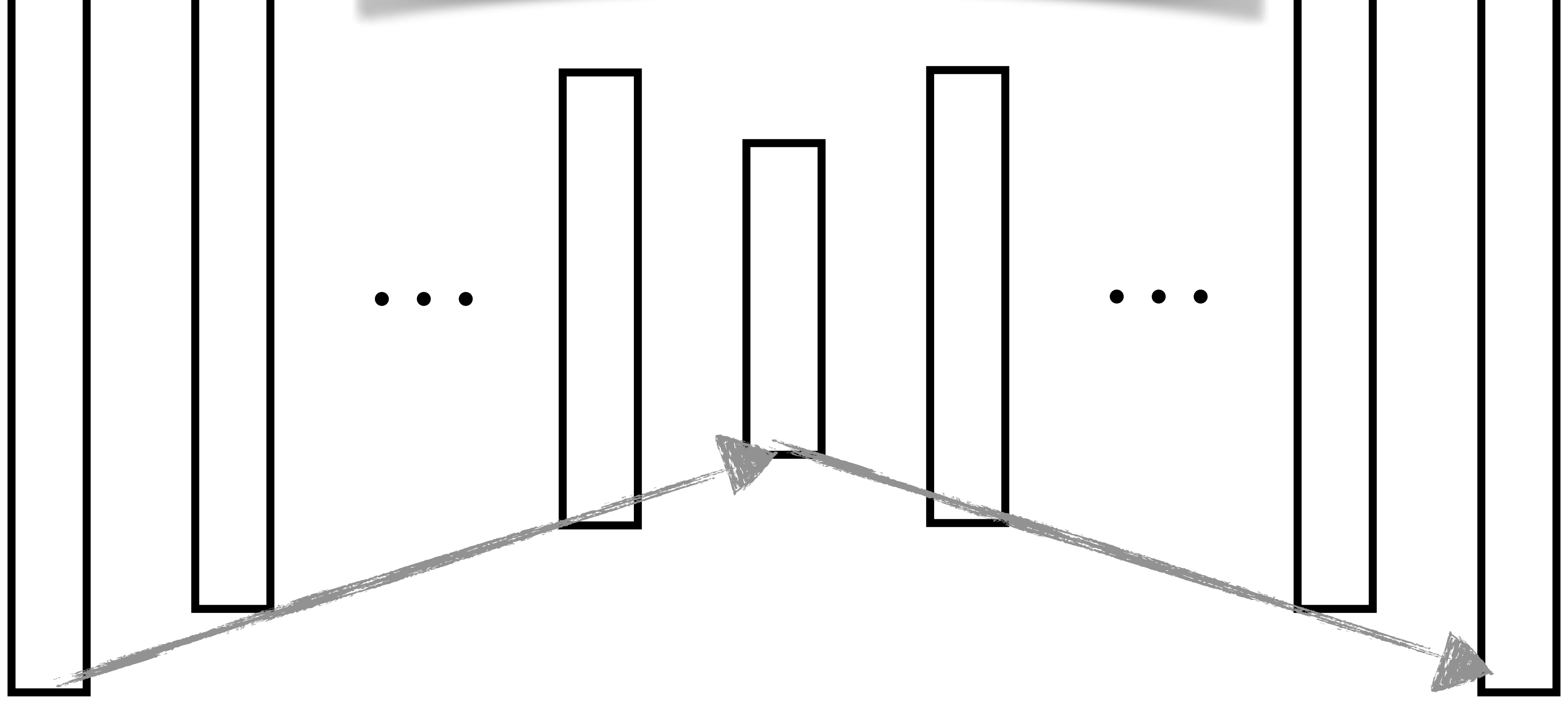
Specifically:

...

Cook a patty at that stove.

...

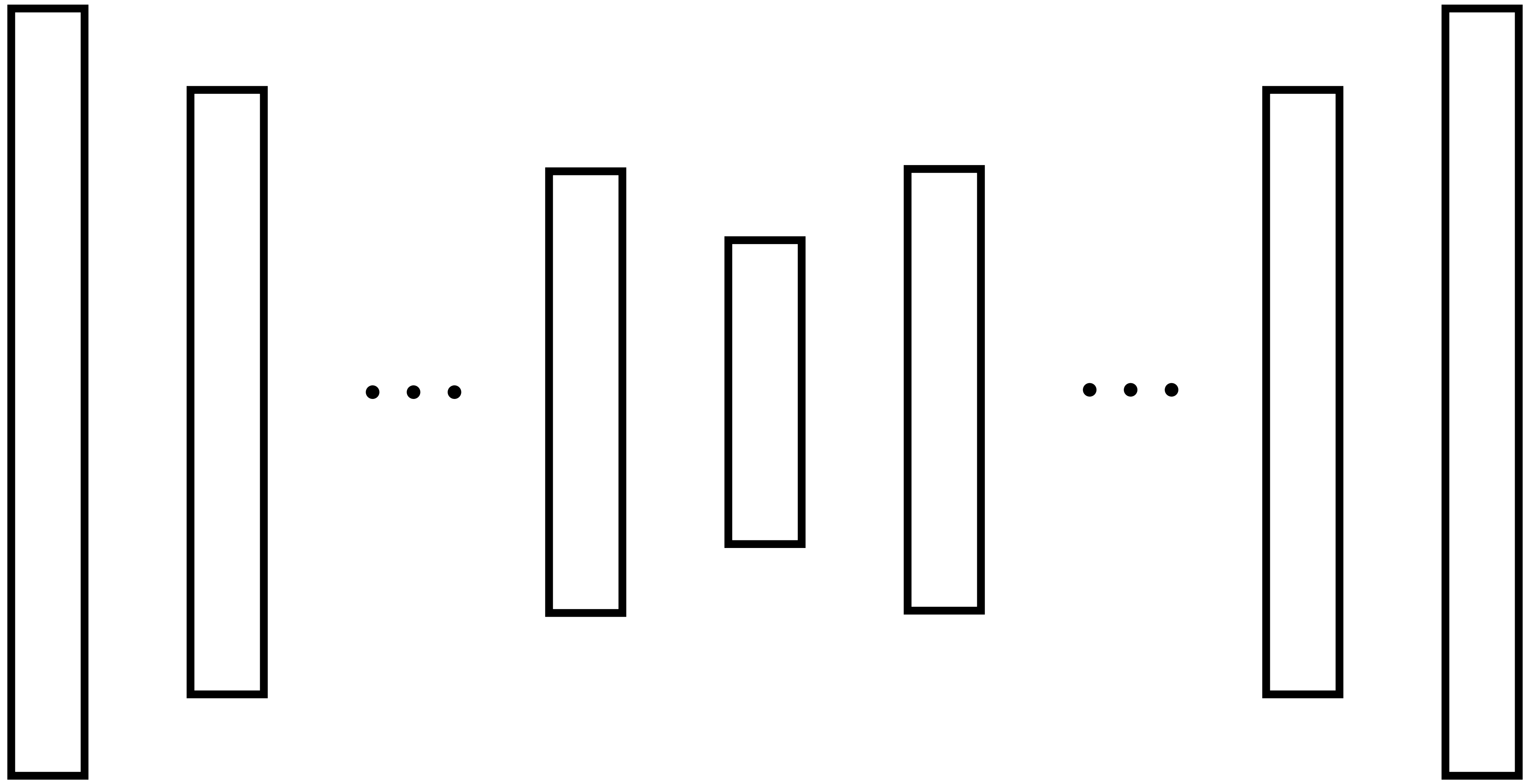
Stack that top bun on that lettuce.



Stage 1
Recursive summarize demo to specification

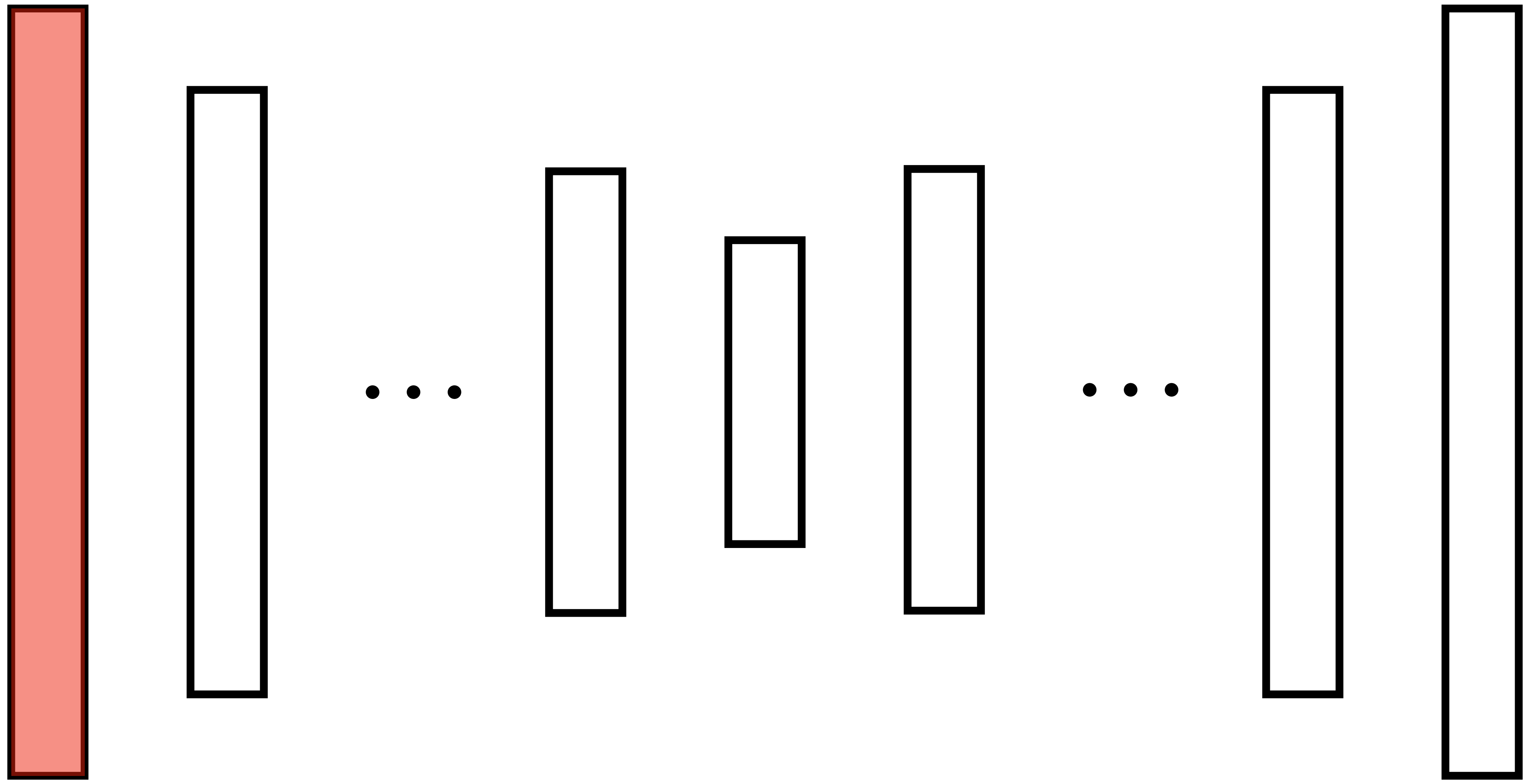
Stage 2
Recursive expand specification to task code

```
# 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)
```



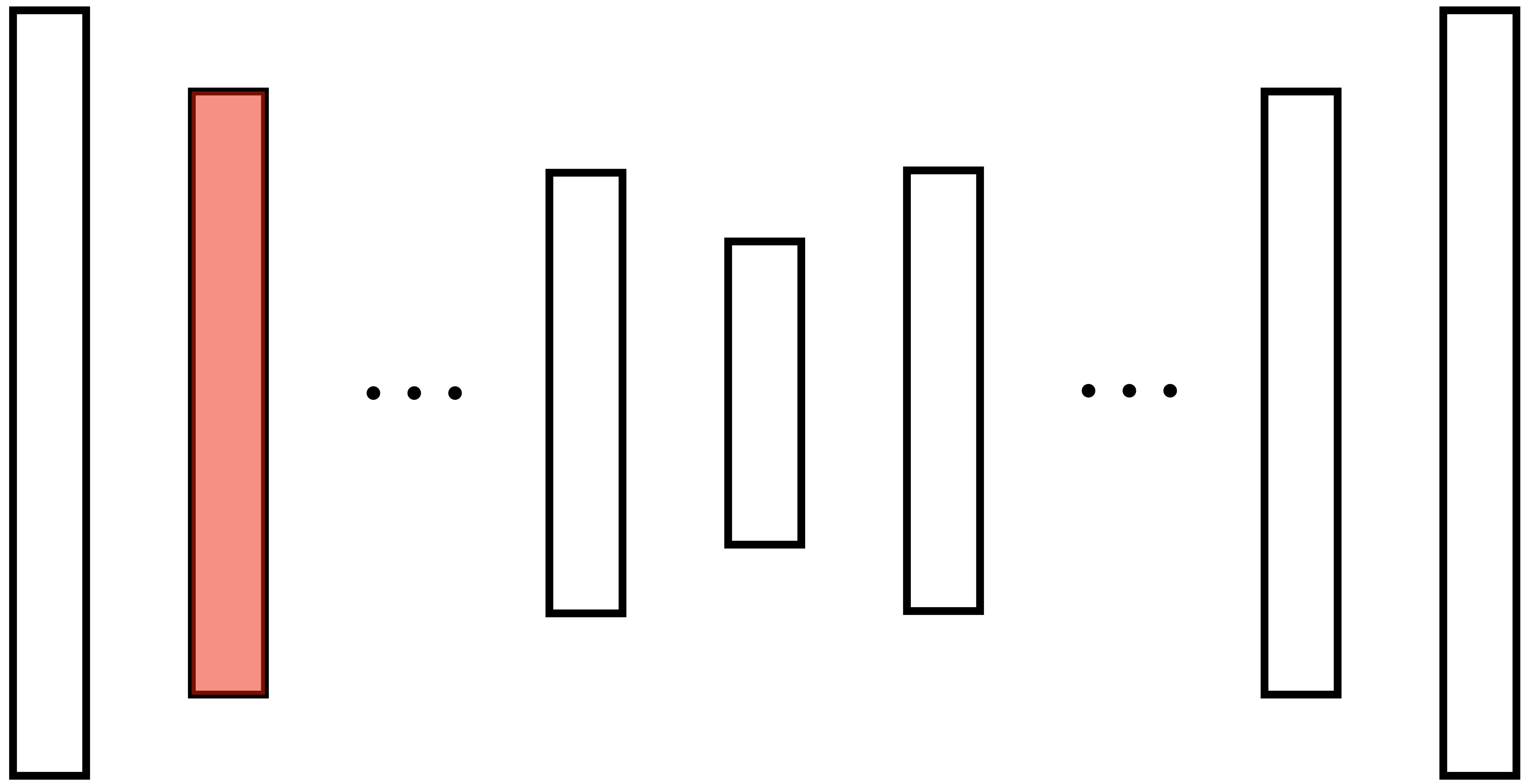
Stage 1: Recursive Summarization

```
[Scenario 1]
Make a burger.

State 2:
'patty1' is not at 'table1'
'robot1' is holding 'patty1'
State 3:
'robot1' is at 'stove2'
'robot1' is not at 'table1'
State 4:
'patty1' is at 'stove2'
'robot1' is not holding 'patty1'
State 5:
State 6:
State 7:
State 8:
'patty1' is cooked
State 9:
'patty1' is not at 'stove2'
'robot1' is holding 'patty1'
State 10:
'robot1' is not at 'stove2'
'robot1' is at 'table3'
State 11:
'patty1' is at 'table3'
'patty1' is on top of 'bottom_bun1'
'robot1' is not holding 'patty1'
State 12:
'robot1' is not at 'table3'
...
...
State 35:
'top_bun3' is at 'table5'
'top_bun3' is on top of 'lettuce3'
'robot1' is not holding 'top_bun3'
```



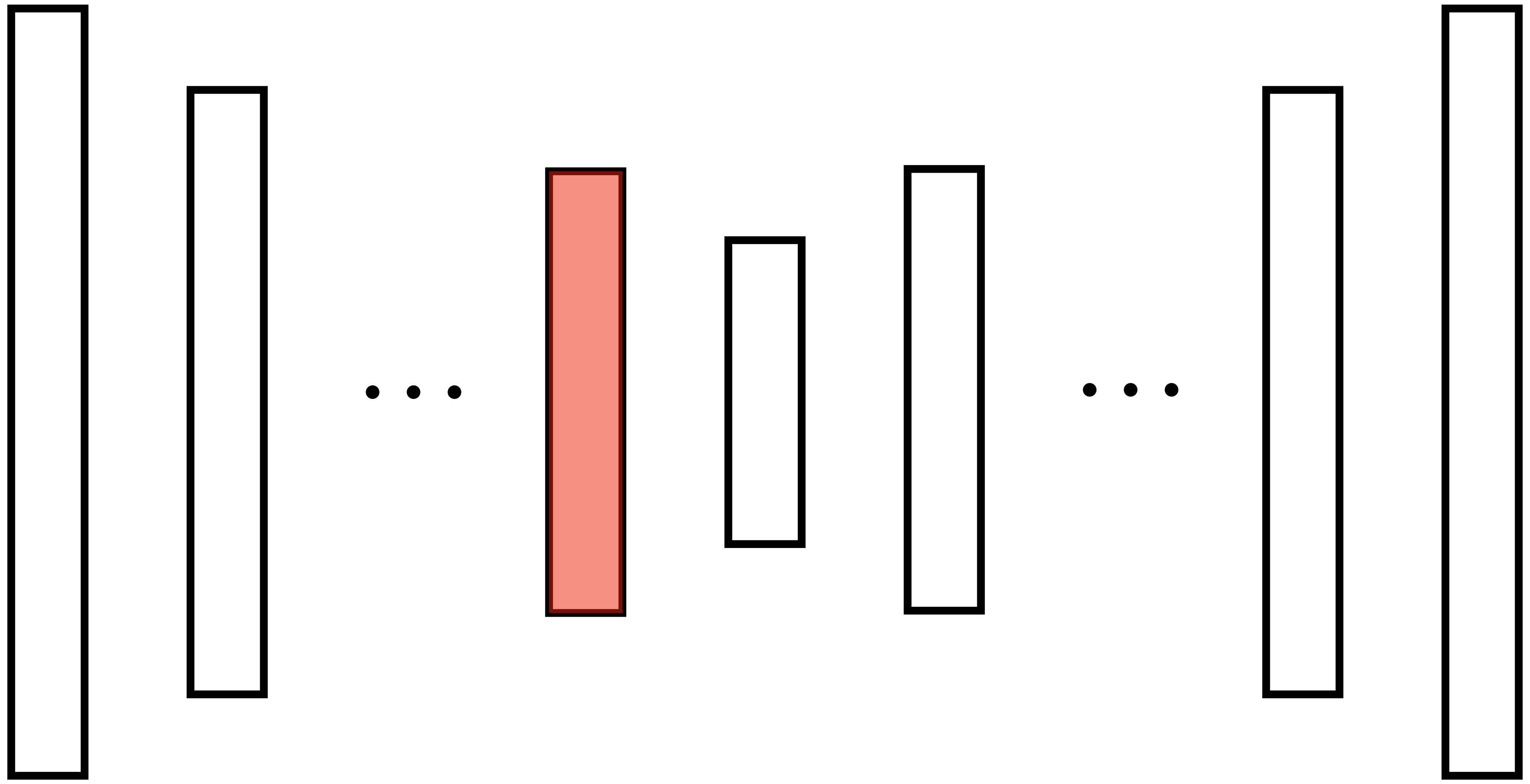
Stage 1: Recursive Summarization

```
[Scenario 2]
Make a burger.

State 2:
'patty3' is not at 'table6'
'robot1' is holding 'patty3'
State 3:
'robot1' is at 'stove3'
'robot1' is not at 'table6'
State 4:
'patty3' is at 'stove3'
'robot1' is not holding 'patty3'
State 5:
State 6:
...
...
State 35:
'top_bun3' is at 'table5'
'top_bun3' is on top of 'lettuce3'
'robot1' is not holding 'top_bun3'
```



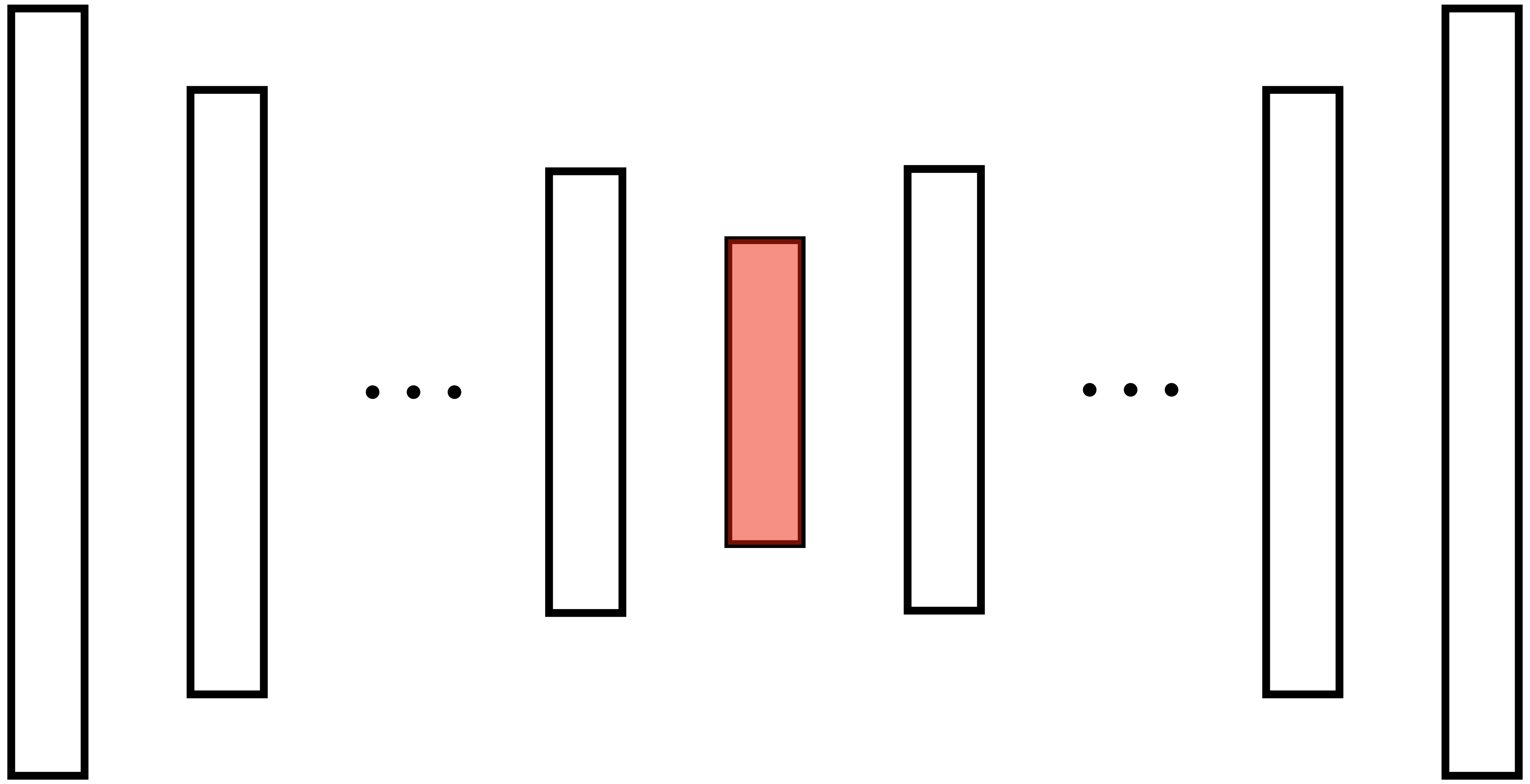
Stage 1: Recursive Summarization

```
* In [Scenario 1], at state 2, the robot picked up 'patty1'.
* At state 3, the robot moved to 'stove2'.
* At state 4, the robot placed 'patty1' on 'stove2'.
* At state 5-7, the robot has cooked 'patty1'.
* At state 8, the robot has finished cooking 'patty1'.
* At state 9, the robot picked up ‘patty1'.
* At state 10, the robot moved to 'table3'.
* At state 11, the robot placed 'patty1' on top of 'bottom_bun1'.
* At state 12, the robot moved to 'table6'.
* At state 13, the robot picked up 'tomato1'.
* At state 14, the robot moved to 'cutting_board1'.
...
* At state 33, the robot picked up 'top_bun1'.
* At state 34, the robot moved to 'table3'.
* At state 35, the robot placed 'top_bun1' on top of 'lettuce1'.
<* In [Scenario 2], at state 2, the robot picked up 'patty3'.
* At state 3, the robot moved to 'stove3'.
* At state 4, the robot placed 'patty3' at location 'stove3'.
...
* At state 35, the robot stacked 'top_bun3' on top of 'lettuce3'.
```



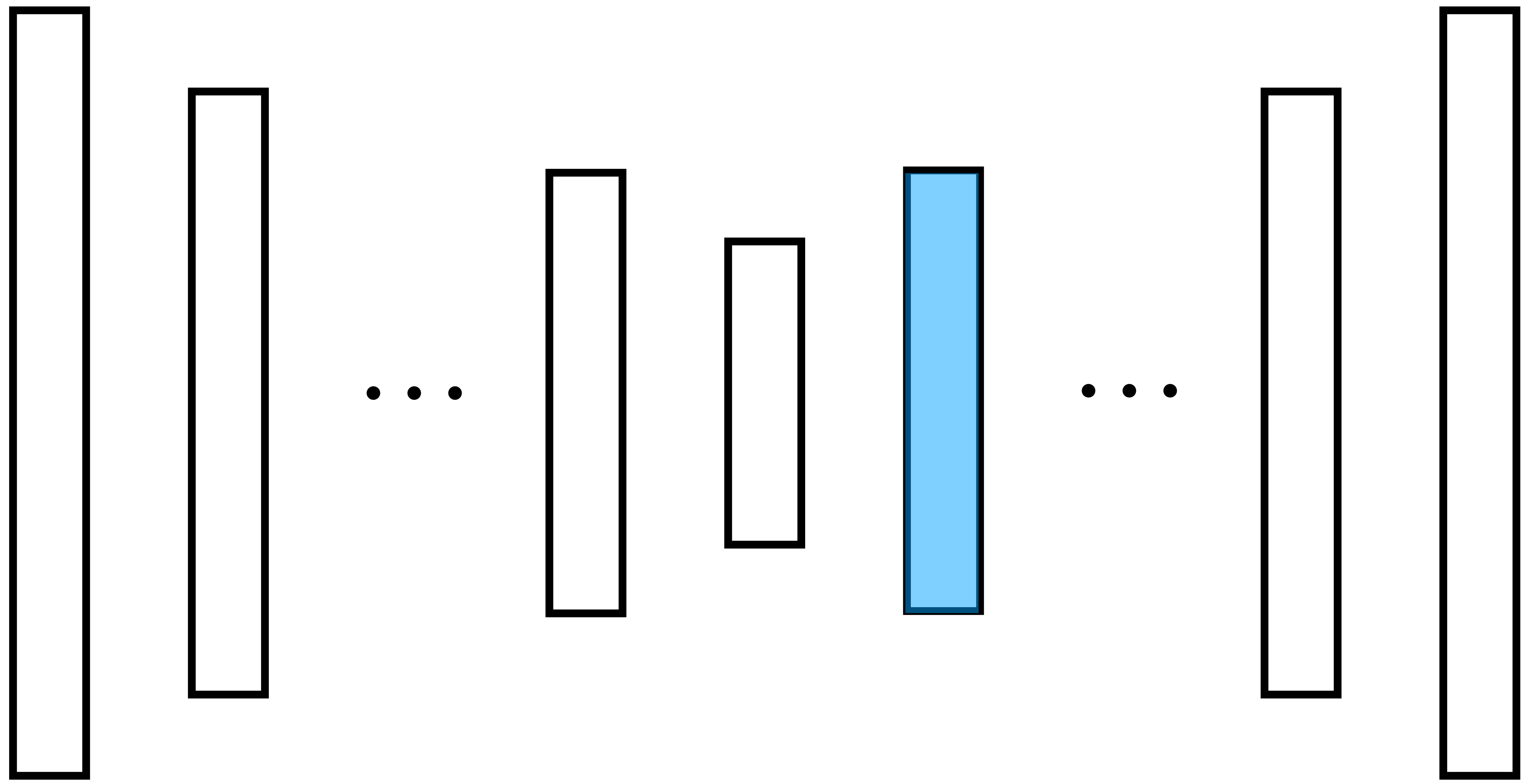
Stage 1: Recursive Summarization

```
* In [Scenario 1], at state 2-8, the subtask is "cook", because: At state 5-7,
the robot has cooked 'patty1'. The robot cooked a patty at a stove, where the
patty is 'patty1', and the stove is 'stove2'.
* At state 9-21, the subtask is "stack", because: At state 11, the robot placed
'patty1' on top of 'bottom_bun1'. ...
* At state 23-28, the subtask is "cut", because: ...
* At state 29-35, the subtask is "stack", because: ...
* In [Scenario 2], at state 2-8, the subtask is "cook", because: ...
* At state 9-11, the subtask is "stack", because: ...
* At state 13-18, the subtask is "cut", because: ...
* At state 19-21, the subtask is "stack", because: ...
* At state 23-28, the subtask is "cut", because: ...
* At state 29-31, the subtask is "stack", because: ...
* At state 33-35, the subtask is "stack", because: ...
```



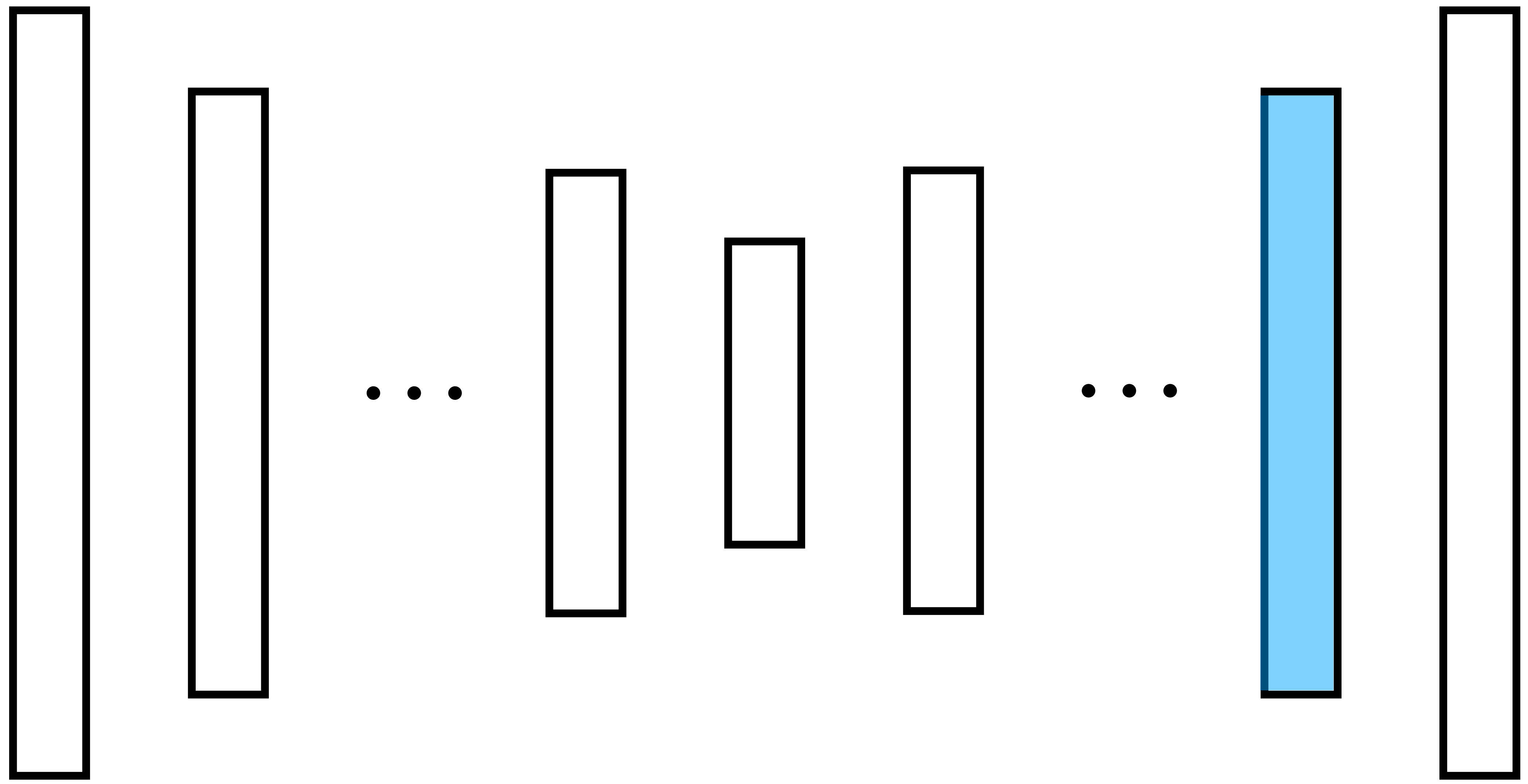
Stage 1: Recursive Summarization

```
* The order of high level actions is: ['cook', 'stack', 'cut', 'stack',
'cut', 'stack']
* In [Scenario 1], 'stove2' is always used for cooking. In [Scenario 2],
'stove3' is always used for cooking. We assume that we just need to decide
a random stove to use in the beginning. Then, we can keep using the same
stove.
* In both scenarios, 'cutting_board1' is used for cutting the lettuce and
tomato. We assume that we just need to use 'cutting_board1' for cutting.

Thus:
Make a burger.

Specifically:
# Get a list of all the bottom buns in the kitchen.
# Get a list of all the patties in the kitchen.
...
# Decide a stove to use.
# Cook a patty at that stove.
...
```



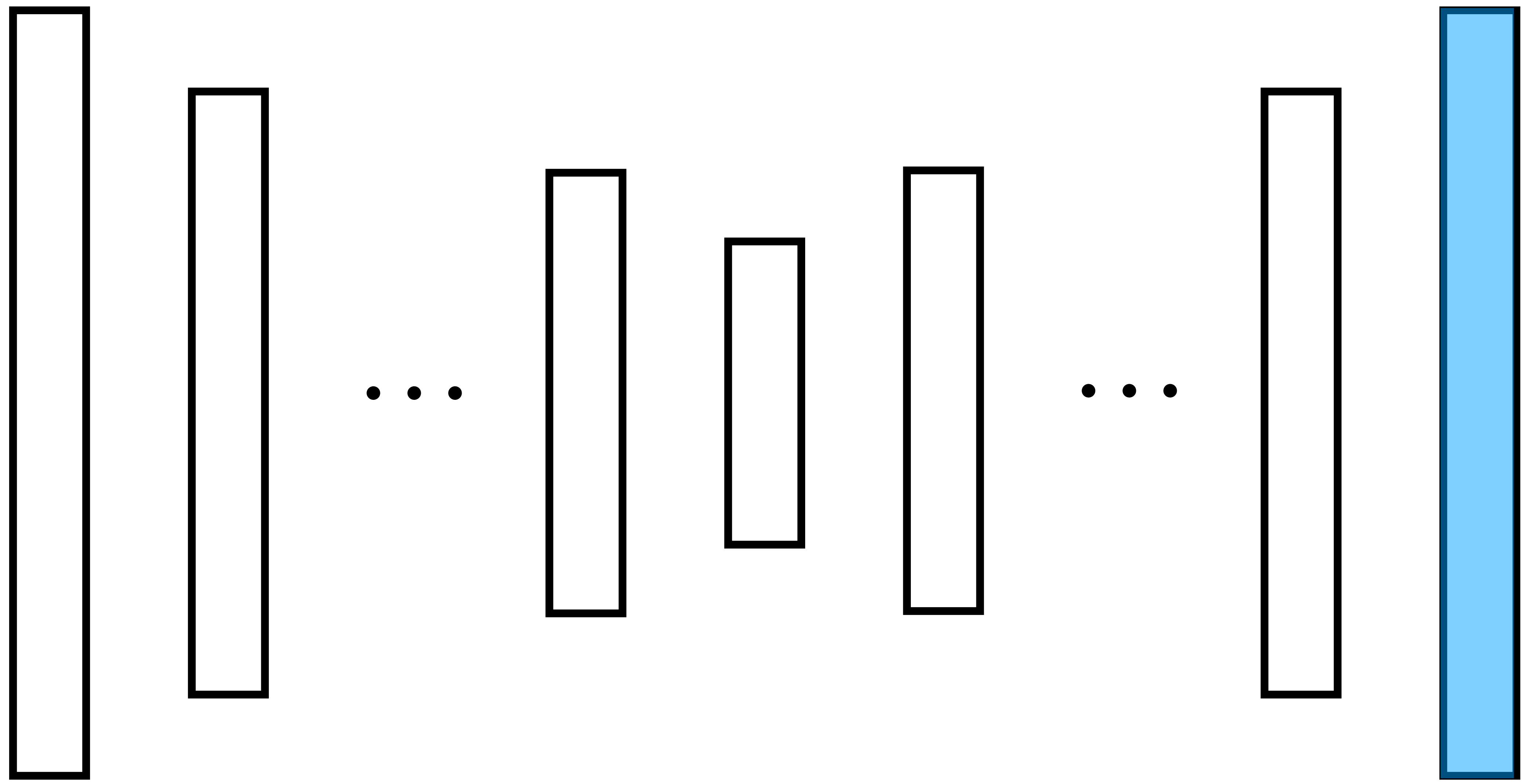
Stage 2: Recursive Expansion

```
from perception_utils import get_all_obj_names_that_match_type, ...

...

# 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')

# Decide a stove to use.
stove_to_cook_at = stoves[0]
# Cook a patty at that stove.
# Decide a patty to cook.
patty_to_cook = patties[0]
cook_object_at_location(obj=patty_to_cook, location=stove_to_cook_at)

...
```



Stage 2: Recursive Expansion

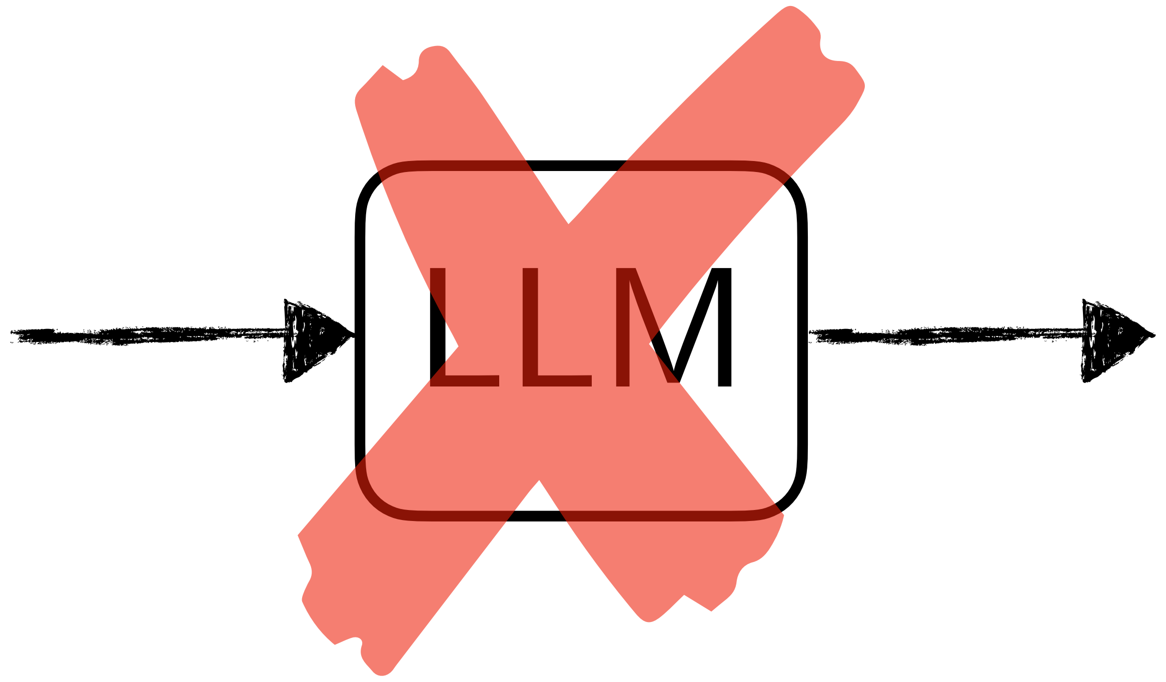
```python
from robot_utils import is_holding, is_in_a_stack, get_obj_that_is_underneath

...
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 obj
from the obj_at_bottom first
            obj_at_bottom = get_obj_that_is_underneath(obj_at_top=obj)
            move_then_unstack(obj_to_unstack=obj, obj_at_bottom=obj_at_bottom,
unstack_location=get_obj_location(obj_at_bottom))
        else:
            # Since obj is not in a stack, robot can just move then pick it up
            move_then_pick(obj=obj)
    # place the object at the location to cook at
    move_then_place(obj=obj, place_location=location)
    # cook the object
    cook_until_is_cooked(obj=obj)
...
```



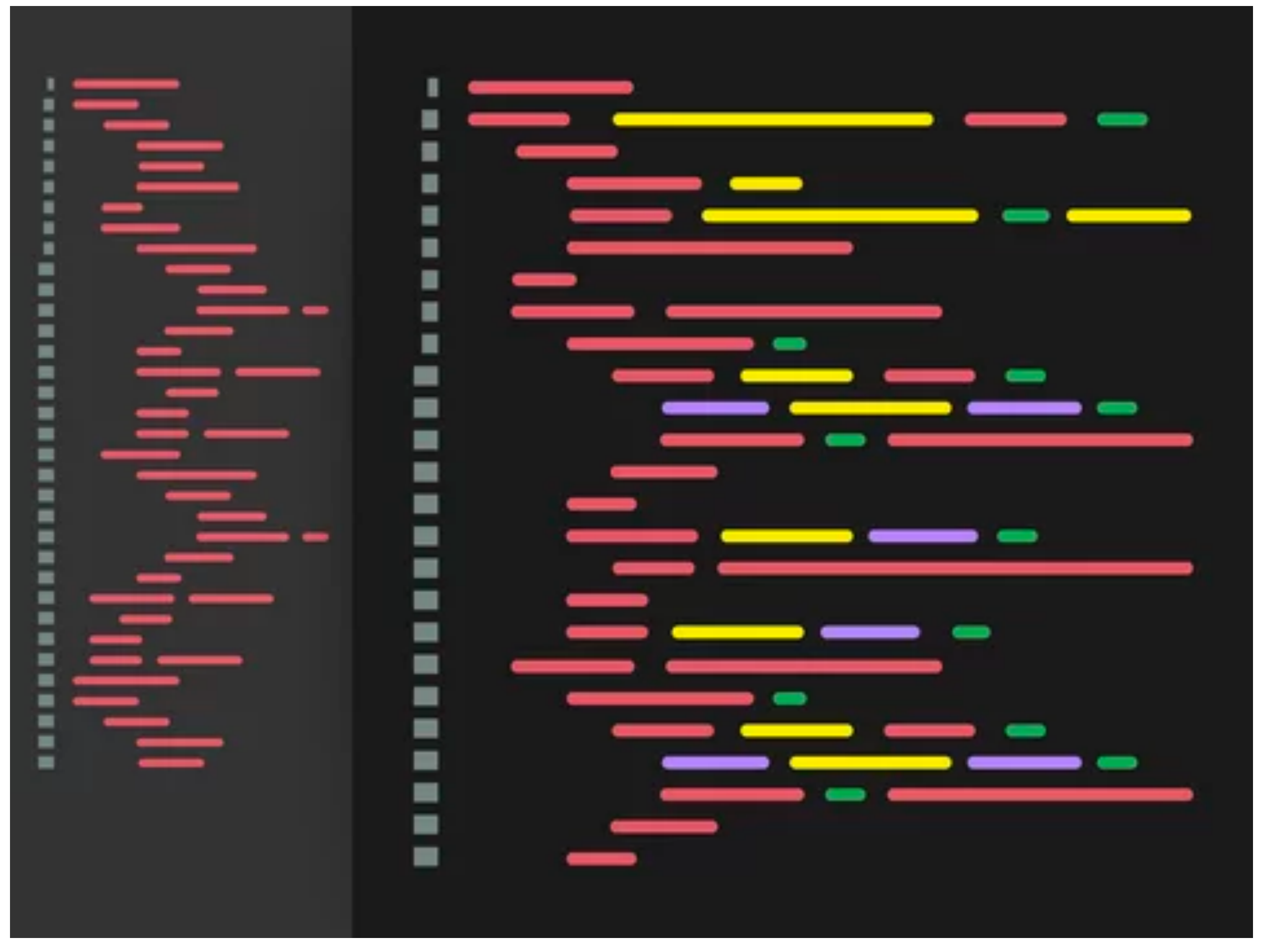
Stage 2: Recursive Expansion

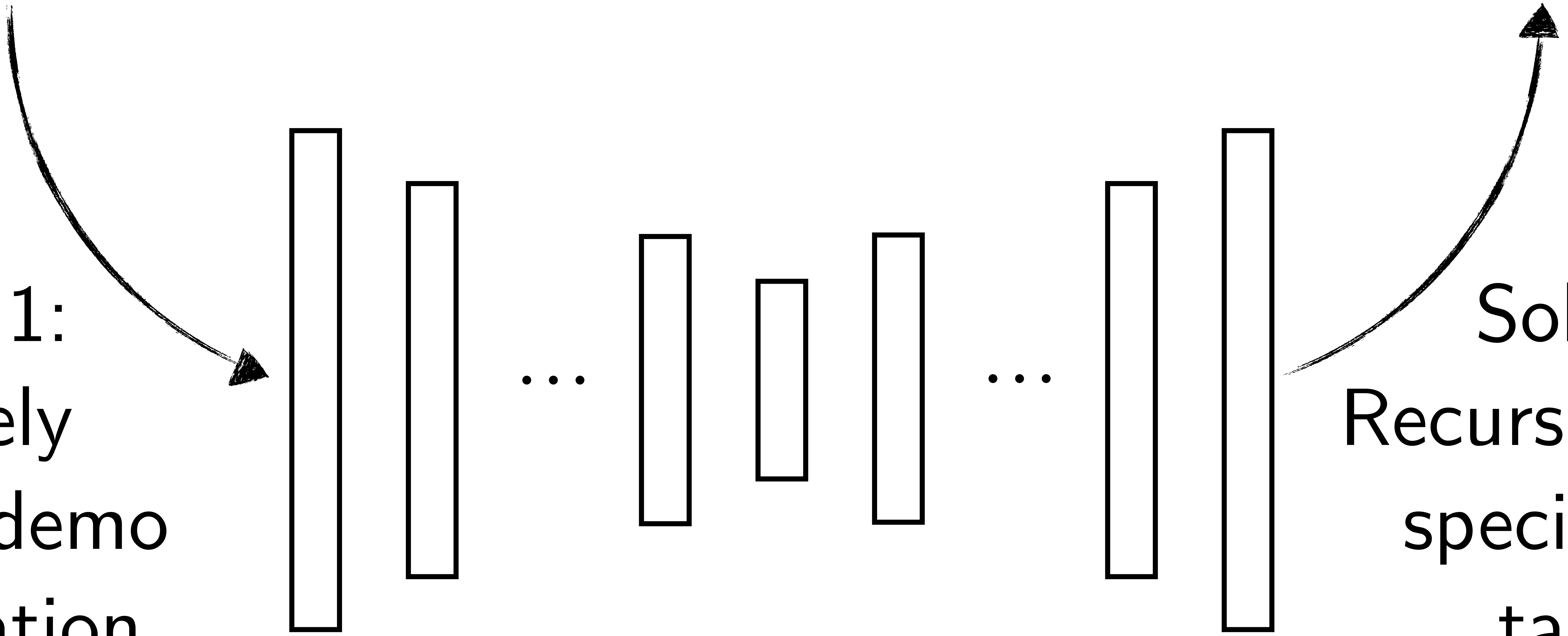
```python
from perception_utils import get_curr_location, get_obj_location, is_cooked
from action_utils import move, unstack, start_cooking, noop

...

def move_then_unstack(obj_to_unstack, obj_at_bottom, unstack_location):
    # For unstacking, we need to move to the location of the bottom object
    if get_curr_location() != get_obj_location(obj_at_bottom):
        move(get_curr_location(), get_obj_location(obj_at_bottom))
    unstack(obj_to_unstack, obj_at_bottom)
    # After unstacking, we need to move to the unstack_location
    if get_curr_location() != unstack_location:
        move(get_curr_location(), unstack_location)


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

...
```

Challenge 1:
Long Horizon Demonstrations



Challenge 2:
Complex Task Code





Solution 1:
Recursively summarize demo to specification

Solution 2:
Recursively expand specification to task code

Experiments

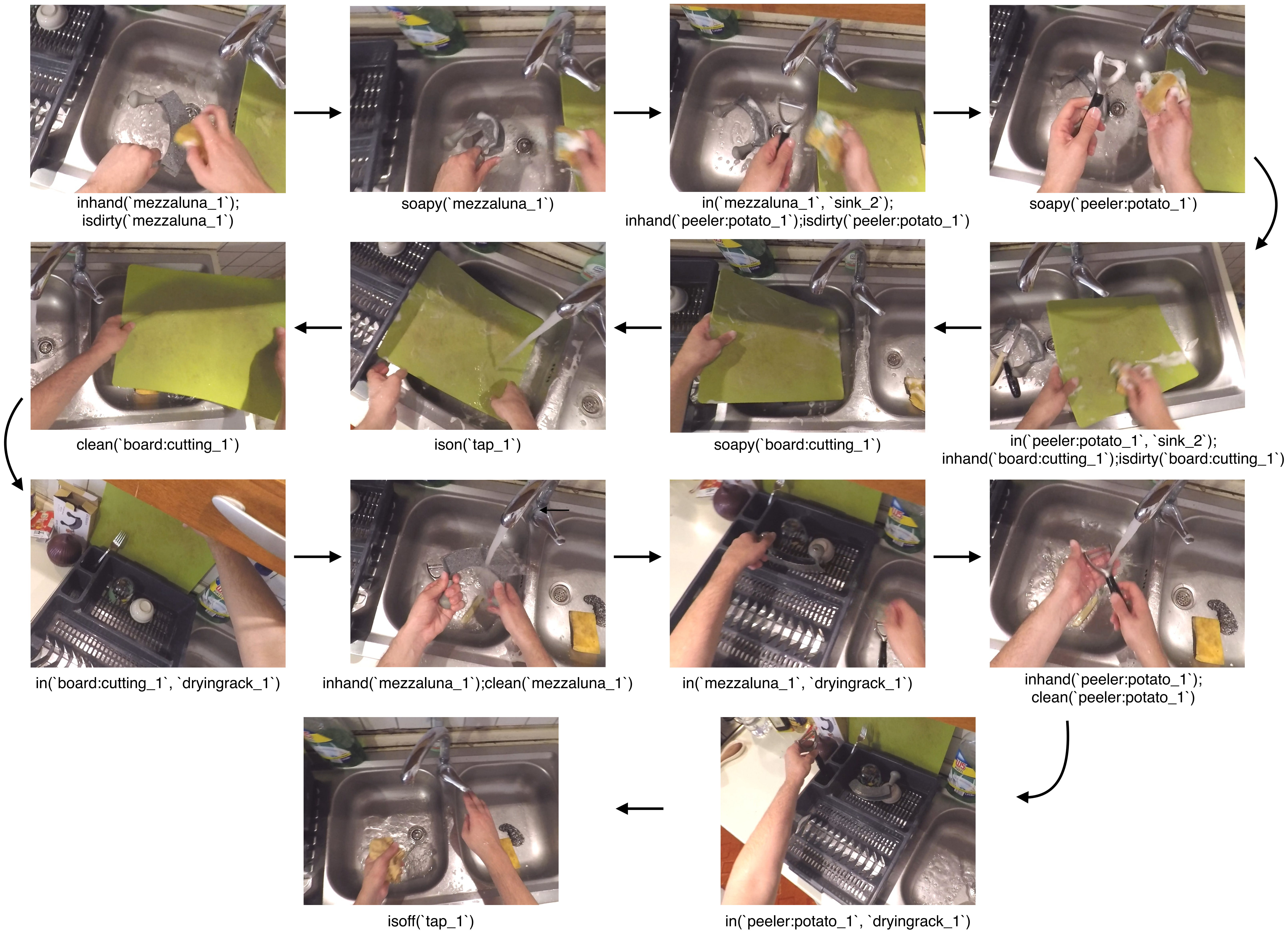
Procedurally generated environment and recipes

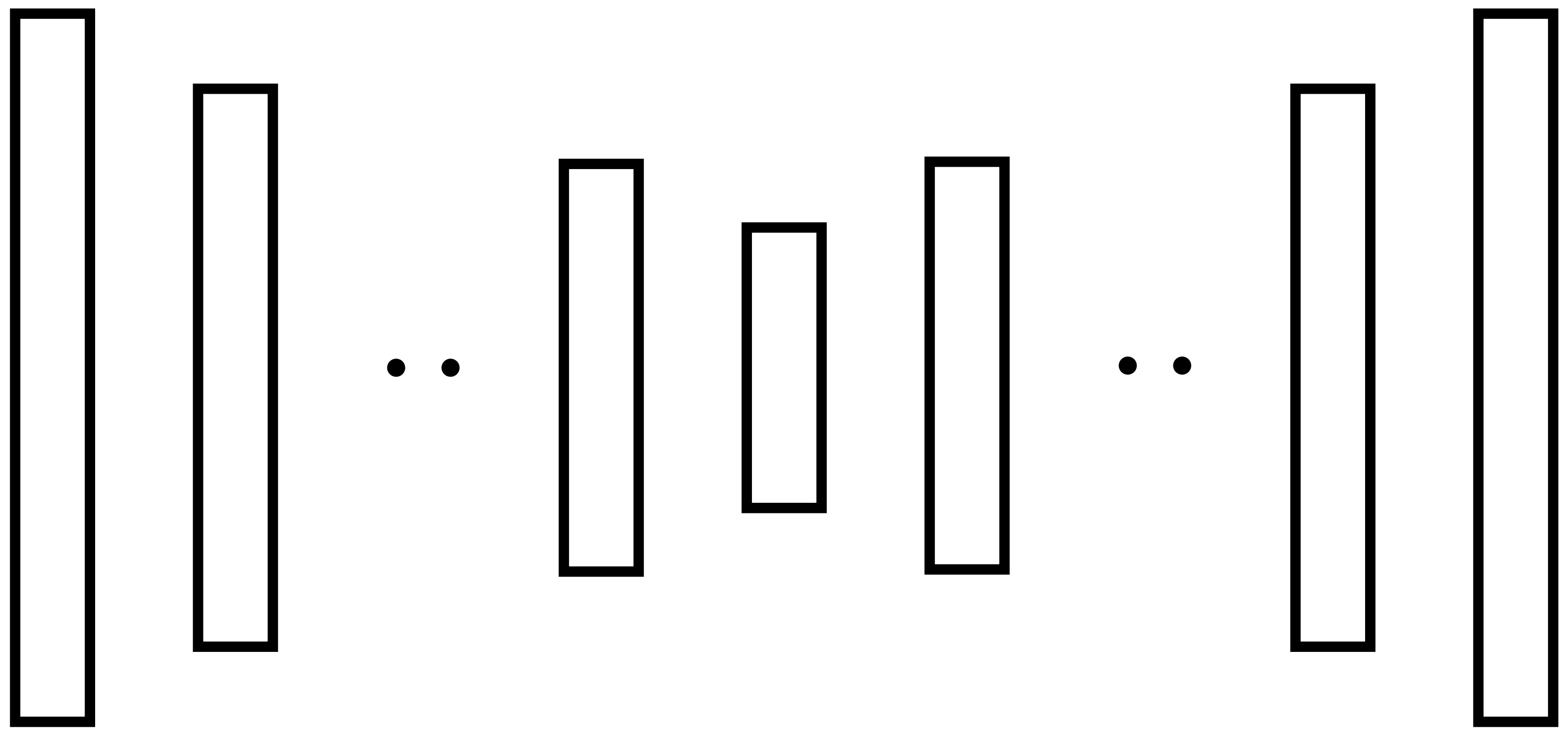
Demo2Code generates correct code that passes unit tests

Task	Lang2Code[30]			DemoNoLang2Code			Demo2Code(*ours*)		
	Exec.	Pass.	Match.	Exec.	Pass.	Match.	Exec.	Pass.	Match
Cook first then cut	1.00	1.00	0.18	0.00	0.00	0.19	1.00	1.00	0.39
Cut first then cook	1.00	1.00	0.11	0.00	1.00	0.10	1.00	1.00	0.34
Cook two patties	1.00	1.00	0.84	0.00	0.00	0.41	1.00	1.00	0.40
Cut two lettuces	1.00	1.00	0.11	0.00	0.00	0.46	1.00	1.00	0.57
Assemble two burgers one by one	0.00	0.00	0.09	0.00	0.60	0.10	0.60	0.60	0.09
Assemble two burgers in parallel	0.00	0.00	0.06	0.00	0.00	0.08	0.00	0.00	0.07
Make a cheese burger	0.00	0.00	0.11	0.50	0.50	0.19	1.00	1.00	0.17
Make a chicken burger	0.00	0.00	0.05	0.00	0.00	0.08	0.50	0.50	0.07
Make a burger stacking lettuce atop patty immediately	0.00	0.00	0.14	1.00	1.00	0.31	0.00	0.00	0.32
Make a burger stacking patty atop lettuce immediately	0.00	0.00	0.14	0.00	0.00	0.27	1.00	1.00	0.08
Make a burger stacking lettuce atop patty after preparation	0.00	0.00	0.14	0.00	0.00	0.29	0.00	0.00	0.16
Make a burger stacking patty atop lettuce after preparation	0.00	0.00	0.13	0.00	0.00	0.15	0.50	0.50	0.25
Make a lettuce tomato burger	1.00	0.00	0.07	0.00	0.00	0.19	1.00	1.00	0.23
Make two cheese burgers	0.00	0.00	0.13	0.00	0.00	0.17	0.00	0.00	0.22
Make two chicken burgers	0.00	0.00	0.06	0.00	0.00	0.07	0.00	0.00	0.07
Make two burgers stacking lettuce atop patty immediately	0.00	0.00	0.20	0.00	0.00	0.20	0.00	0.00	0.28
Make two burgers stacking patty atop lettuce immediately	0.00	0.00	0.20	0.00	0.00	0.26	0.00	0.00	0.09
Make two burgers stacking lettuce atop patty after preparation	0.00	0.00	0.13	0.00	0.00	0.28	0.00	0.00	0.12
Make two burgers stacking patty atop lettuce after preparation	0.00	0.00	0.14	1.00	1.00	0.08	0.00	0.00	0.25
Make two lettuce tomato burgers	1.00	0.00	0.10	1.00	0.00	0.26	0.70	0.70	0.27
Overall	0.27	0.18	0.15	0.20	0.23	0.21	0.42	0.42	0.22

[Damen et al '18]

EPIC Kitchen Tasks





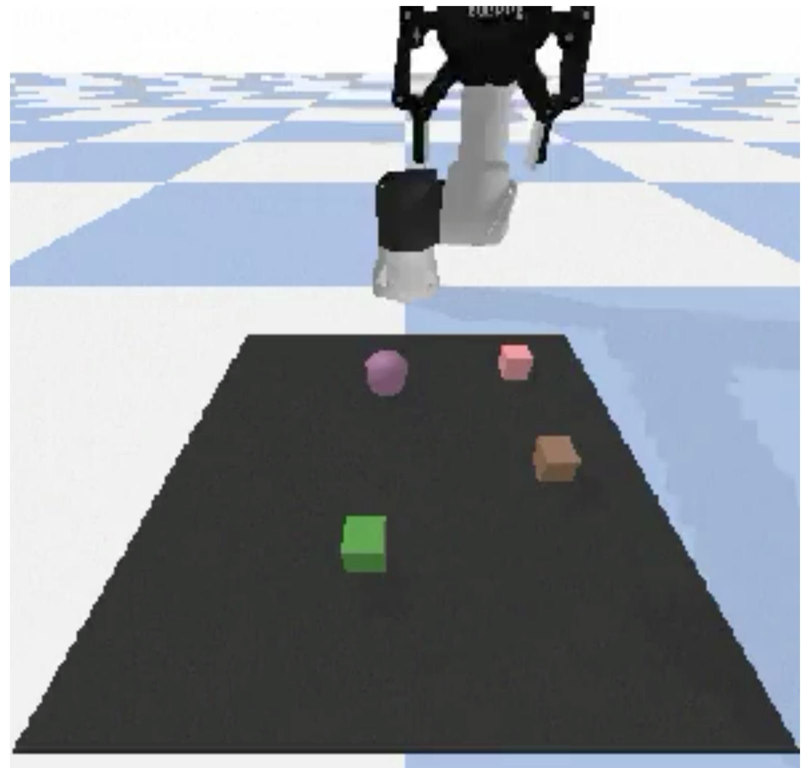
```
objects = get_all_objects()
for object in objects:
    pick_up(object)
    if check_if_dirty(object):
        while check_if_dirty(object):
            scrub(object)
    place(object, "sink_2")
turn_on("tap_1")
for object in objects:
    pick_up(object)
    rinse(object)
    place(object, "dryingrack_1")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

Dishwashing Tasks across Users

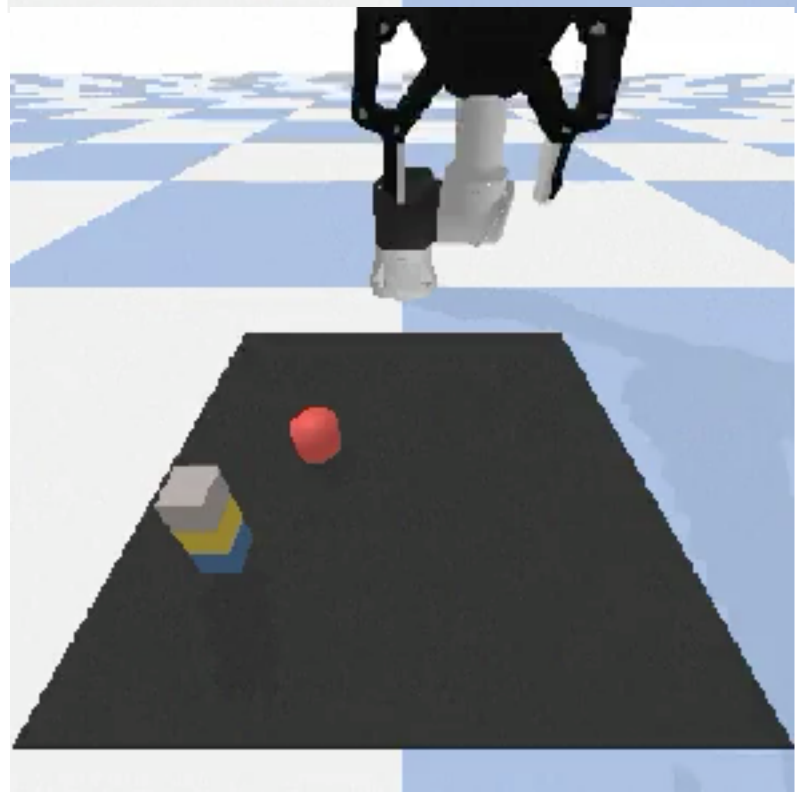
	P4-101 (7)		P7-04 (17)		P7-10 (6)		P22-05 (28)		P22-07 (30)		P30-07 (11)		P30-08 (16)	
	Pass	Match	Pass	Match	Pass	Match	Pass	Match	Pass	Match	Pass	Match	Pass	Match
Lang2Code [30]	1	0.856	0	0.350	0	0.569	0	0.620	0	0.696	1	0.872	0	0.706
DemoNoLang2Code	1	0.233	0	0.522	0	0.695	0	0.537	0	0.233	1	0.966	0	0.671
Demo2Code	1	0.854	1	0.660	1	1.000	0	0.838	1	0.855	1	0.873	0	0.796

Tabletop Manipulation Tasks

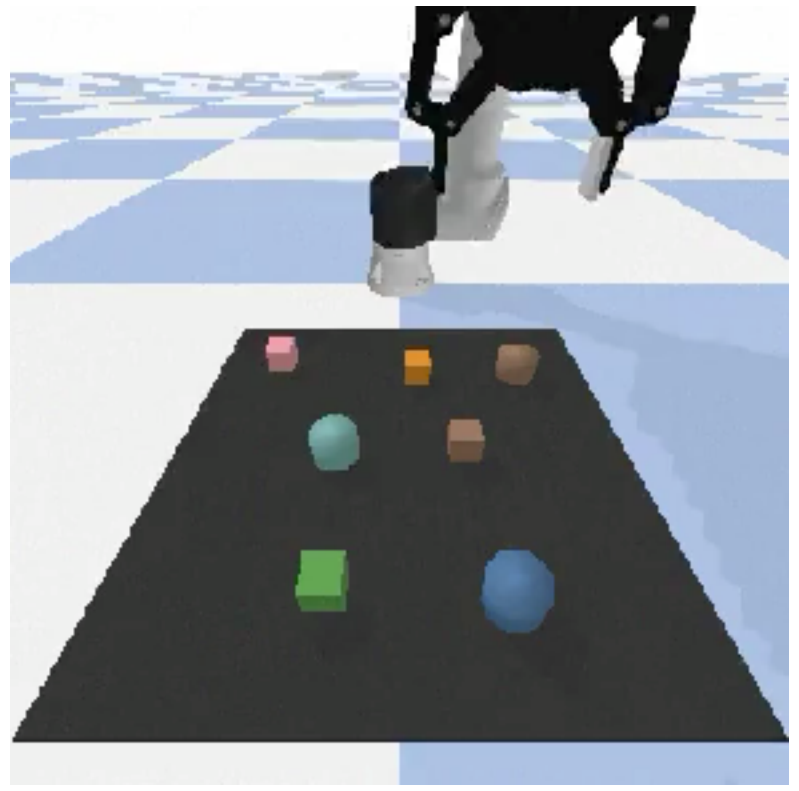
Place the purple cylinder to the left of the green block.



Place the blue block on red cylinder (but blocked by yellow, red)



Stack all objects into two stacks (one stack has only blocks, other cylinder)



	Task	Lang2Code[30]			DemoNoLang2Code			Demo2Code(*ours*)		
		Exec.	Pass.	Match.	Exec.	Pass.	Match.	Exec.	Pass.	Match
Specific	Place A next to B	1.00	0.28	0.47	0.82	0.20	0.33	0.92	0.90	0.90
	Place A at a corner of the table	1.00	0.18	0.05	0.82	0.20	0.33	0.92	0.90	0.90
	Place A at an edge of the table	1.00	0.18	0.03	0.94	0.88	0.87	1.00	0.98	0.96
Hidden	Place A on top of B	1.00	0.20	0.26	0.93	0.00	0.04	1.00	0.73	0.41
	Stack all blocks	0.93	0.00	0.08	0.98	0.73	0.56	1.00	0.97	0.99
	Stack all cylinders	0.80	0.00	0.66	0.88	0.53	0.32	1.00	0.90	0.96
Prefs	Stack all blocks into one stack	0.98	0.00	0.26	1.00	0.40	0.05	0.87	0.97	0.48
	Stack all cylinders into one stack	0.93	0.13	0.01	0.97	0.43	0.11	0.87	0.93	0.42
	Stack all objects into two stacks	0.95	0.30	0.09	0.85	0.40	0.50	0.80	1.00	0.63
	Overall	0.95	0.14	0.21	0.91	0.42	0.34	0.93	0.92	0.74

Demo2Code Learns *Personalized* Tasks


User 1: Prefers lettuce on patty
robot is at ‘table2’,…
'robot1' is holding ‘patty1’,…
`patty 1’ is cooked,…
‘patty1’ is on top of ‘bottom_bun1’,…
‘robot1’ is holding ‘lettuce1’,…
‘lettuce1’ is on top of `patty1’,…
’top_bun1’ is on top of ‘lettuce1’,…
Pick up patty1
Cook patty1
Stack patty1
Pick up lettuce1
Stack lettuce1
Stack top_bun1


User 2: Prefers cheese on patty
robot is at ‘table1’,…
‘robot1’ is holding ‘patty1’,…
‘patty1’ is cooked
‘patty1’ is on top of ‘bottom_bun1,…
‘robot1’ is holding ‘cheese1’
‘cheese1’ is on top of ‘patty1’,…
‘top_bun1’ is on top of ‘cheese1’,…
Pick up patty1
Cook patty1
Stack patty1
Pick up cheese1
Stack cheese1
Stack top_bun1


User 1: Prefers lettuce on patty
robot is at 'table2',…
'robot1' is holding 'patty1',…
`patty 1' is cooked,…
'patty1' is on top of 'bottom_bun1',…
'robot1' is holding 'lettuce1',…
'lettuce1' is on top of `patty1',…
'top_bun1' is on top of 'lettuce1',…
Pick up patty1
Cook patty1
Stack patty1
Pick up lettuce1
Stack lettuce1
Stack top_bun1
Make a burger.
...
Decide a patty to cook.
Cook that patty at that stove.
...
Decide a lettuce to cut.
Cut that lettuce on that cutting board.
Stack that lettuce on that patty.
...
Stack that top bun on that lettuce.
def main():
...
patty = patties[0]
cook_obj_at_loc(patty, stoves[0])
...
lettuce = lettuces[0]
cut_obj_at_loc(lettuce, boards[0])
stack_obj1_on_obj2(lettuce, patty)
...
stack_obj1_on_obj2(top_bun, lettuce)
User 2: Prefers cheese on patty
robot is at 'table1',…
'robot1' is holding 'patty1',…
'patty1' is cooked
'patty1' is on top of 'bottom_bun1,…
'robot1' is holding 'cheese1'
'cheese1' is on top of 'patty1',…
'top_bun1' is on top of 'cheese1',…
Pick up patty1
Cook patty1
Stack patty1
Pick up cheese1
Stack cheese1
Stack top_bun1


User 1: Prefers lettuce on patty
robot is at 'table2',…
'robot1' is holding 'patty1',…
'patty 1' is cooked,…
'patty1' is on top of 'bottom_bun1',…
'robot1' is holding 'lettuce1',…
'lettuce1' is on top of 'patty1',…
'top_bun1' is on top of 'lettuce1',…
Pick up patty1
Cook patty1
Stack patty1
Pick up lettuce1
Stack lettuce1
Stack top_bun1
Make a burger.
...
Decide a patty to cook.
Cook that patty at that stove.
...
Decide a lettuce to cut.
Cut that lettuce on that cutting board.
Stack that lettuce on that patty.
...
Stack that top bun on that lettuce.
def main():
...
patty = patties[0]
cook_obj_at_loc(patty, stoves[0])
...
lettuce = lettuces[0]
cut_obj_at_loc(lettuce, boards[0])
stack_obj1_on_obj2(lettuce, patty)
...
stack_obj1_on_obj2(top_bun, lettuce)
def main():
...
patty = patties[0]
cook_obj_at_loc(patty, stoves[0])
...
cheese = cheeses[0]
stack_obj1_on_obj2(cheese, patty)
...
stack_obj1_on_obj2(top_bun, cheese)
Make a burger.
...
Decide a patty to cook.
Cook that patty at that stove.
...
Decide a cheese to use.
Stack that cheese on that patty.
...
Stack that top bun on that cheese.
User 2: Prefers cheese on patty
robot is at 'table1',…
'robot1' is holding 'patty1',…
'patty1' is cooked
'patty1' is on top of 'bottom_bun1,…
'robot1' is holding 'cheese1'
'cheese1' is on top of 'patty1',…
'top_bun1' is on top of 'cheese1',…
Pick up patty1
Cook patty1
Stack patty1
Pick up cheese1
Stack cheese1
Stack top_bun1


User 22: Prefers to first scrub all objects and then rinse
inhand(`bowl_1`); isdirty(`bowl_1`); ...
scrub('bowl_1')
inhand(`glass_1`); isdirty(`glass_1`); ...
scrub('glass_1')
ison(`tap_1`);inhand(`bowl_1`); soapy(`bowl_1`); ...
rinse('bowl_1')
inhand(`bowl_1`); clean(`bowl_1`); ...
place('bowl_1')
inhand(`glass_1`); soapy(`glass_1`); ...
rinse('glass_1')
inhand(`glass_1`);clean(`glass_1`); ...
place('glass_1') ...


User 30: Prefers to scrub and rinse each object
inhand(`mug_1`); **isdirty(`mug_1`)**; ...
scrub('mug_1')
ison(`tap_1`); inhand(`mug_1`); soapy(`mug_1`); ...
rinse('mug_1')
clean(`mug_1`); at(`countertop_1`); ...
place('mug_1')
at(`countertop_1`);on(`jug_1`, countertop_1`); ...
pickup('jug_1')
ison(`tap_1`); inhand(`jug_1`); **isnotdirty(`jug_1`); ...**
rinse('jug_1')
clean(`jug_1`);at(`countertop_1`); ...
place('jug_1')

User 22: Prefers to first scrub all objects and then rinse

inhand(`bowl_1`); isdirty(`bowl_1`); ... | inhand(`glass_1`); isdirty(`glass_1`); ... | ison(`tap_1`);inhand(`bowl_1`); soapy(`bowl_1`); ... | inhand(`bowl_1`); clean(`bowl_1`); ... | inhand(`glass_1`); soapy(`glass_1`); ... | inhand(`glass_1`);clean(`glass_1`); ...

scrub('bowl_1') | scrub('glass_1') | rinse('bowl_1') | place('bowl_1') | rinse('glass_1') | place('glass_1') ...

```
Wash objects at the sink.
...
Get a list of all objects to wash
Pick up scrub_1
For each object in all objects:
    Scrub object
    Place object in sink_2
Turn on tap_1
For each object in all objects:
    Rinse object
    Place object in dishrack_1
Turn off tap_1
```

```
objs = get_all_objs()
pick_up("scrub_1")
for obj in objs:
    scrub(obj)
    place(obj, "sink_2")
turn_on("tap_1")
for obj in objs:
    rinse(obj)
    place(obj, "dishrack_1")
turn_off("tap_1")
```

```
objs = get_all_objs()
for obj in objs:
    bring_objs_to_loc([obj],
     "sink_1")
    if check_if_dirty(obj):
        scrub(obj)
    while check_if_dirty(object):
        rinse(obj)
    turn_off("tap_1")
    place(obj, "counter_1")
```

```
Wash objects at the sink.
...
Get a list of all objects to wash
For each object in all objects:
    Bring object to sink_1
    Scrub object if object is
dirty
    Rinse object till clean
    Turn off tap_1
    Place object on counter_1
```

User 30: Prefers to scrub and rinse each object

inhand(`mug_1`); isdirty(`mug_1`); ... | ison(`tap_1`); inhand(`mug_1`); soapy(`mug_1`); ... | clean(`mug_1`); at(`countertop_1`); ... | at(`countertop_1`);on(`jug_1`, countertop_1`); ... | ison(`tap_1`); inhand(`jug_1`); isnotdirty(`jug_1`); ... | clean(`jug_1`);at(`countertop_1`); ...

scrub('mug_1') | rinse('mug_1') | place('mug_1') | pickup('jug_1') | rinse('jug_1') | place('jug_1')


Many open research questions!

What is the right level of abstraction for LLMs to generate?

(Growing support for LLMs generating reward functions)

Huang et al. VoxPoser

Can language help for non-language tasks?

(Growing evidence that language captures useful invariances)

Mirchandani et al.
Large Language Models as General Pattern Machines

Can LLMs solve planning problems?

(Growing evidence that says No)

Valmeekam et al.
Large Language Models Still Can't Plan