Situated Mapping of Sequential Instructions to Actions with Single-step Reward Observation

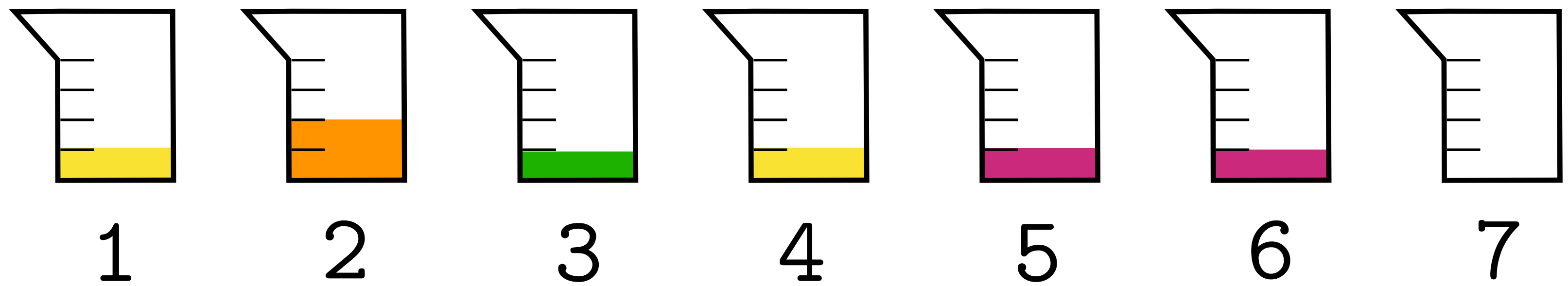
Alane Suhr and Yoav Artzi

Executing Context-Dependent Instructions

Task: map a sequence of instructions to actions

Existing Work	**Today**
Symbolic Representations	System Actions
Modeling Context	Learning from Exploration

Executing a Sequence of Instructions



Empty out the leftmost beaker of purple chemical

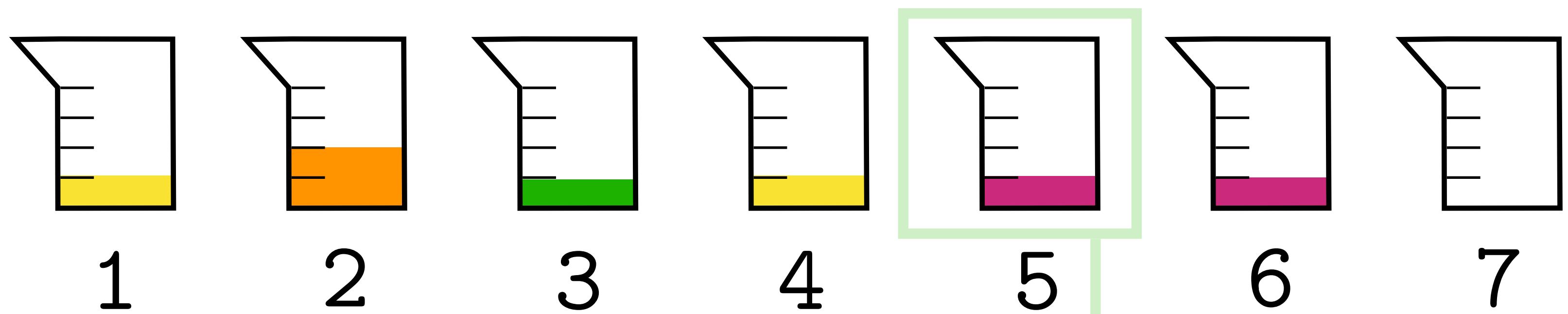
Then, add the contents of the first beaker to the second

Mix it

Then, drain 1 unit from it

Same for 1 more unit

Executing a Sequence of Instructions



→ *Empty out the leftmost beaker of purple chemical*

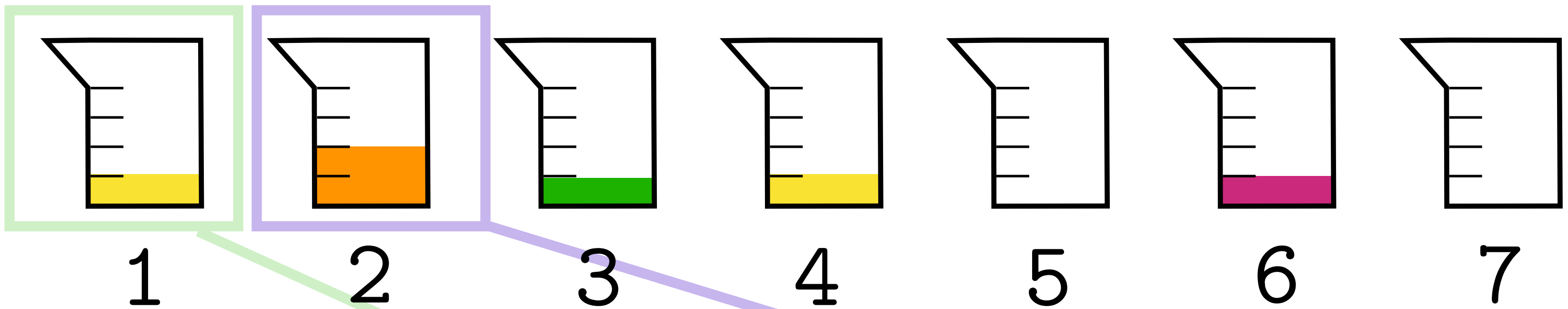
Then, add the contents of the first beaker to the second

Mix it

Then, drain 1 unit from it

Same for 1 more unit

Executing a Sequence of Instructions



Empty out the leftmost beaker of purple chemical

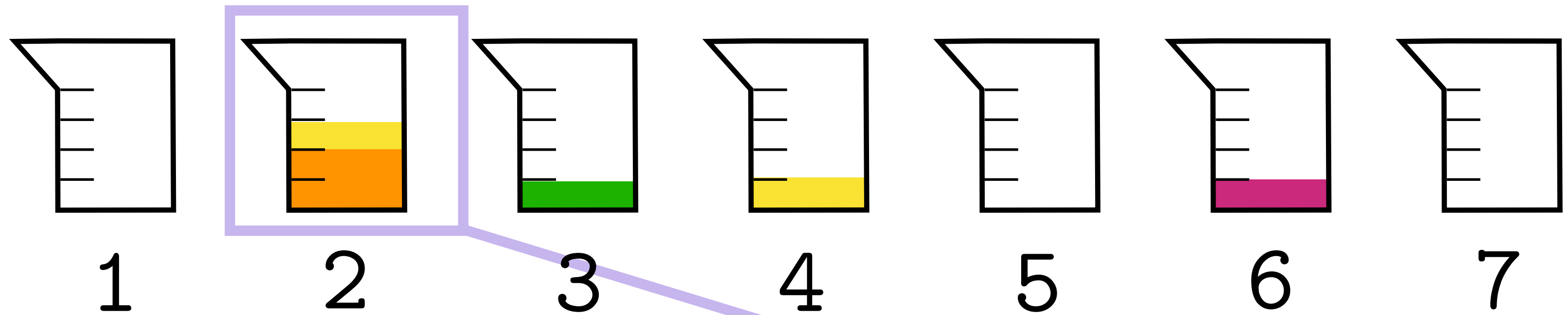
→ *Then, add the contents of the first beaker to the second*

Mix it

Then, drain 1 unit from it

Same for 1 more unit

Executing a Sequence of Instructions



Empty out the leftmost beaker of purple chemical

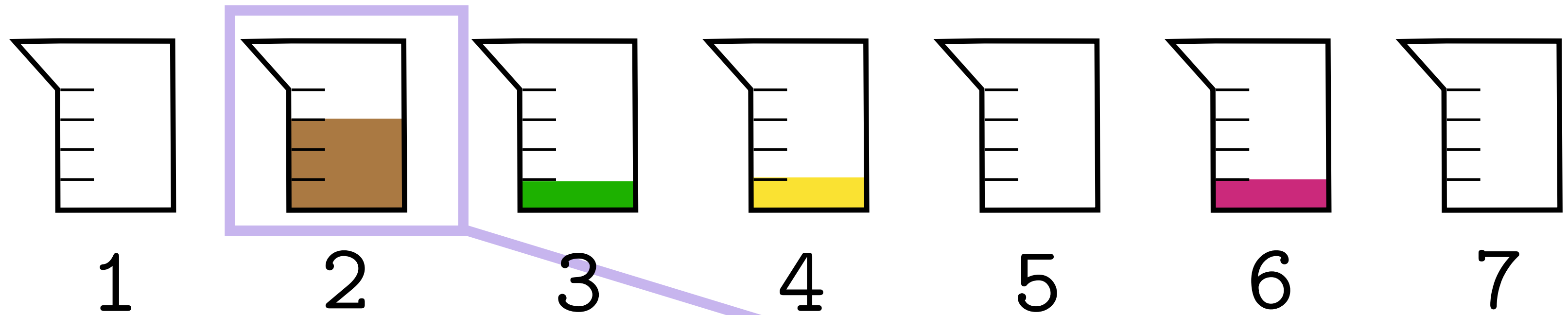
Then, add the contents of the first beaker to the second

Mix it

Then, drain 1 unit from it

Same for 1 more unit

Executing a Sequence of Instructions



Empty out the leftmost beaker of purple chemical

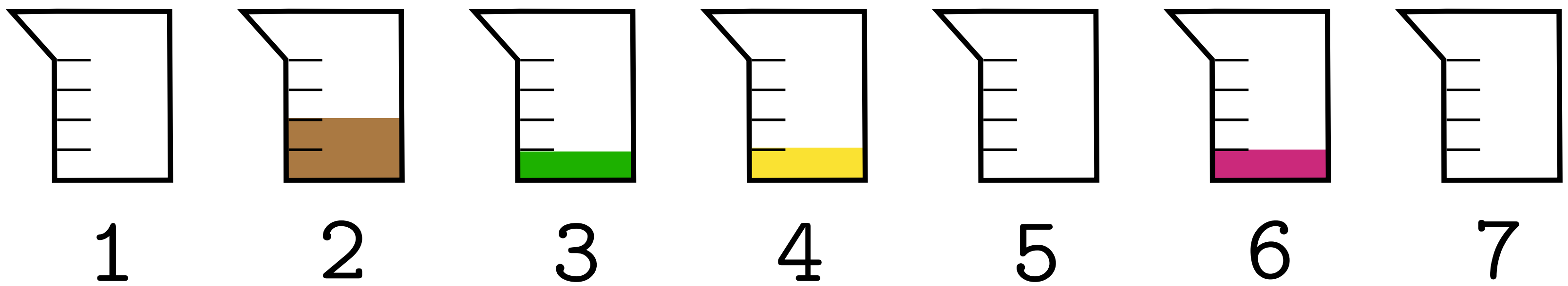
Then, add the contents of the first beaker to the second

Mix it

Then, drain 1 unit from it

Same for 1 more unit

Executing a Sequence of Instructions



Empty out the leftmost beaker of purple chemical

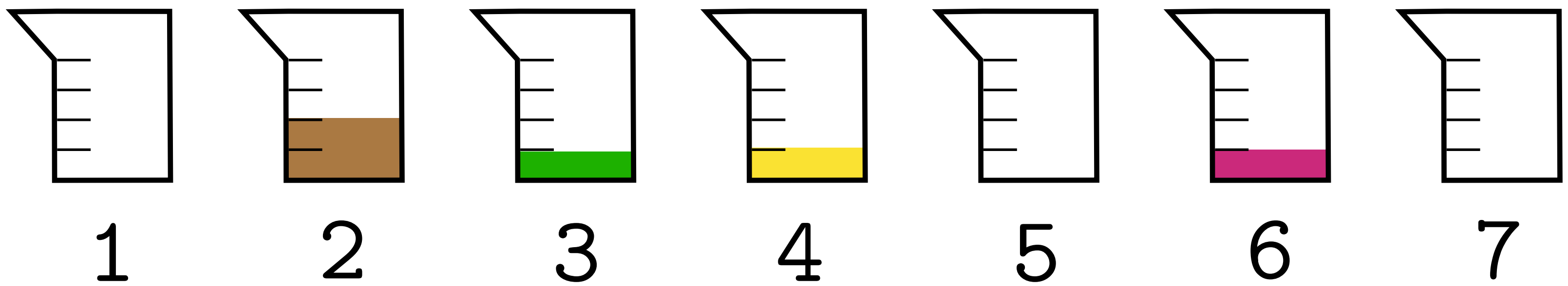
Then, add the contents of the first beaker to the second

Mix it

Then, drain 1 unit from it

Same for 1 more unit

Executing a Sequence of Instructions



Empty out the leftmost beaker of purple chemical

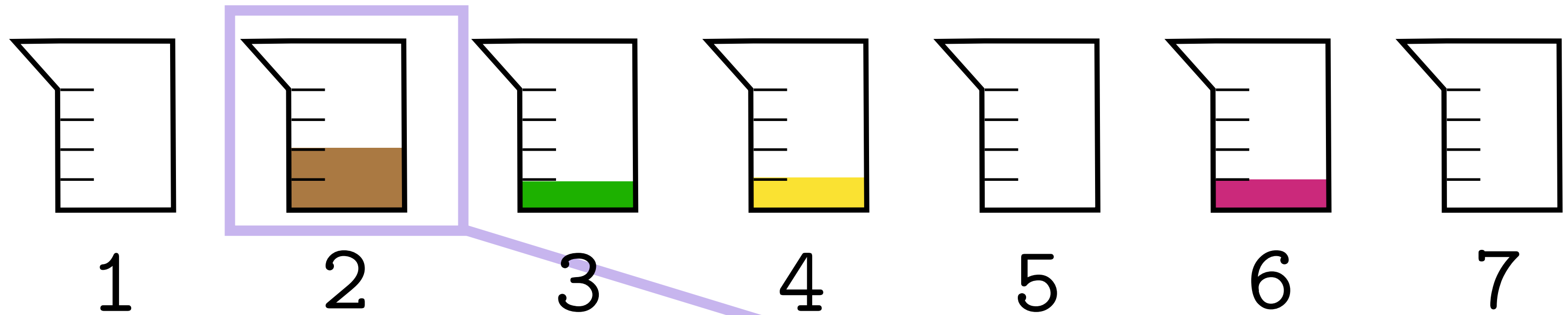
Then, add the contents of the first beaker to the second

Mix it

Then, drain 1 unit from it

Same for 1 more unit

Executing a Sequence of Instructions



Empty out the leftmost beaker of purple chemical

Then, add the contents of the first beaker to the second

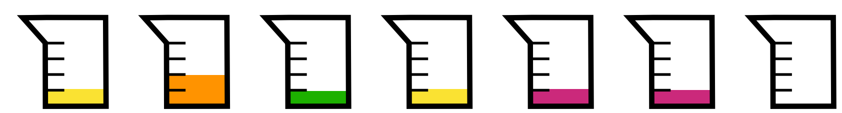
Mix it

Then, drain 1 unit from it

Same for 1 more unit

Problem Setup

- Task: follow sequence of instructions
- Learning from instructions and corresponding world states



Empty out the leftmost beaker of purple chemical

Then, add the contents of the first beaker to the second

Mix it

Then, drain 1 unit from it

Same for 1 more unit

Problem Setup

- Task: follow sequence of instructions
- Learning from instructions and corresponding world states

Empty out the leftmost beaker of purple chemical

Then, add the contents of the first beaker to the second

Mix it

Then, drain 1 unit from it

Same for 1 more unit

Problem Setup

- Task: follow sequence of instructions
- Learning from instructions and corresponding world states

Empty out the leftmost beaker of purple chemical

Then, add the contents of the first beaker to the second

Mix it

Then, drain 1 unit from it

Same for 1 more unit

Problem Setup

- Task: follow sequence of instructions
- Learning from instructions and corresponding world states

Empty out the leftmost beaker of purple chemical

Then, add the contents of the first beaker to the second

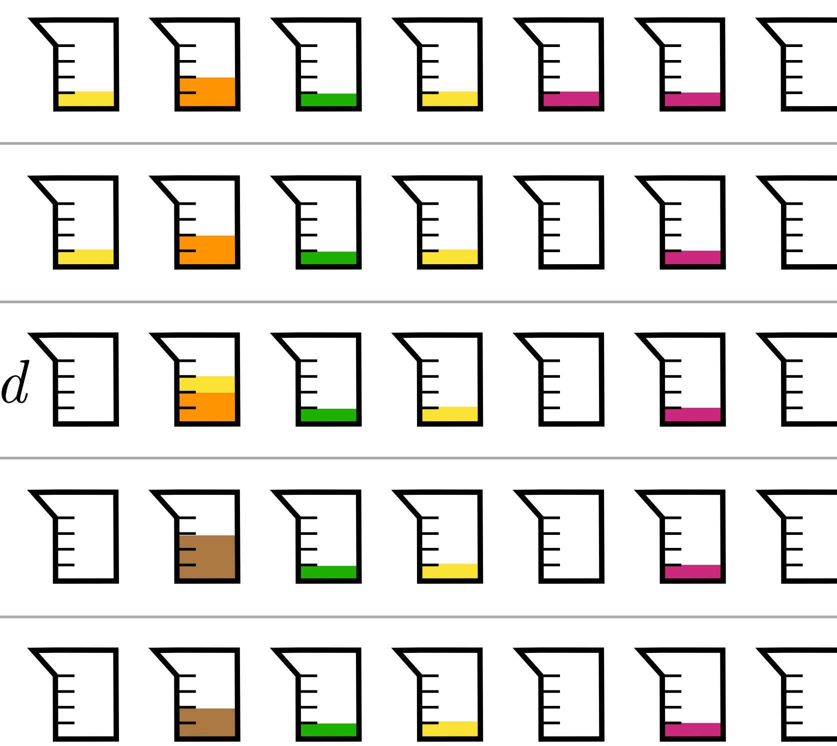
Mix it

Then, drain 1 unit from it

Same for 1 more unit

Problem Setup

- Task: follow sequence of instructions
- Learning from instructions and corresponding world states



Empty out the leftmost beaker of purple chemical

Then, add the contents of the first beaker to the second

Mix it

Then, drain 1 unit from it

Same for 1 more unit

Problem Setup

- Task: follow sequence of instructions
- Learning from instructions and corresponding world states

Empty out the leftmost beaker of purple chemical

Then, add the contents of the first beaker to the second

Mix it

Then, drain 1 unit from it

Same for 1 more unit

Related Work

- Context-dependent language understanding
  - Static environments (e.g., large database)
  - Environments that change over time while instructions are given
- Following instructions in isolation; varying levels of supervision

Miller et al. 1996, Zettlemoyer and Collins 2009, Suhr et al. 2018

Long et al. 2016, Guu et al. 2017, Fried et al. 2018

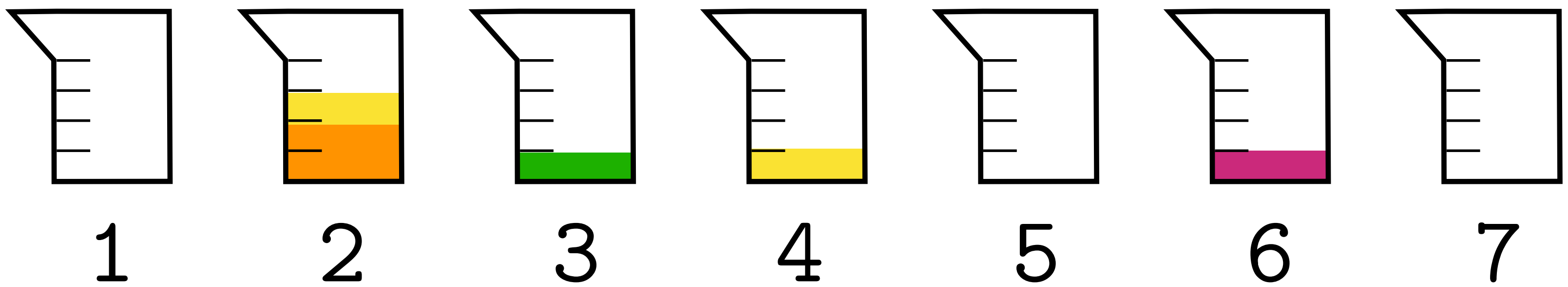
Chen and Mooney 2011, Chen 2012, Artzi and Zettlemoyer 2013, Artzi et al. 2014, Andreas and Klein 2015, Bisk et al. 2016, Misra et al. 2017

Today

1. Attention-based model for generating sequences of system actions that modify the environment

2. Exploration-based learning procedure that avoids biases learned early in training

System Actions

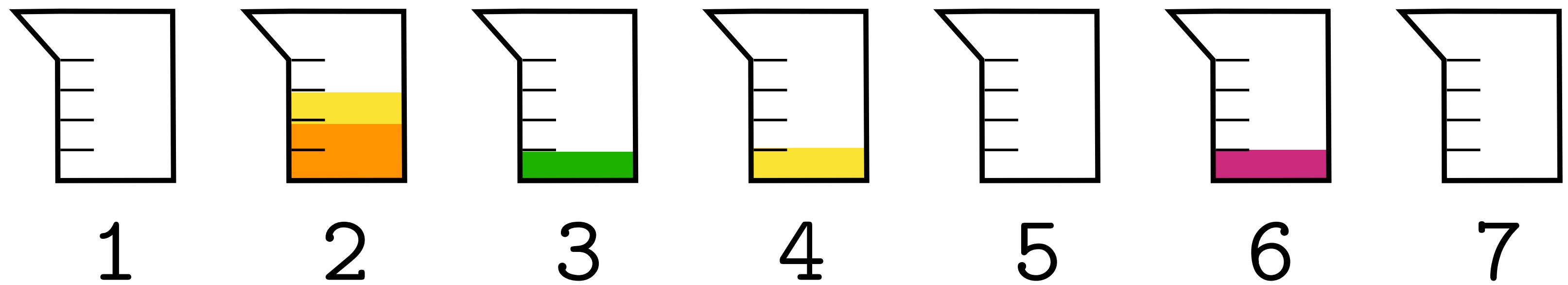


Mix it

- Each beaker is a stack
- Actions are **`pop`** and **`push`**

```
pop  2;
pop  2;
pop  2;
push 2 brown;
push 2 brown;
push 2 brown;
```

Meaning Representation



Mix it

High-level Program

```
mix(prevArg2(2))
```

Representation Engineering

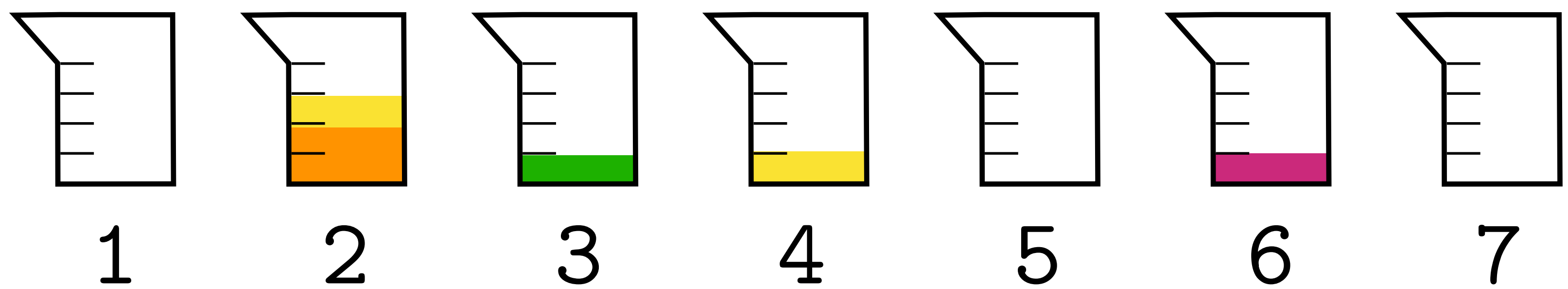
vs.

System Actions

```
pop 2; pop 2; pop 2;
push 2 brown;
push 2 brown;
push 2 brown;
```

Learning Abstractions

Meaning Representation



Mix it

High-level Program `mix(prevArg2(2))` Representation Engineering

vs.

System Actions

```
pop 2; pop 2; pop 2;
    push 2 brown;
    push 2 brown;
    push 2 brown;
```

Learning Abstractions

Model

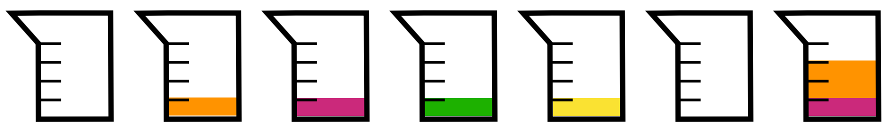
Previous instructions

Throw out first beaker *Pour sixth beaker into last one*

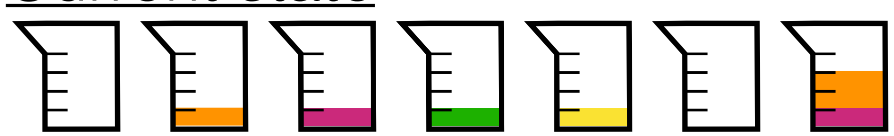
Current instruction

It turns brown

Initial state

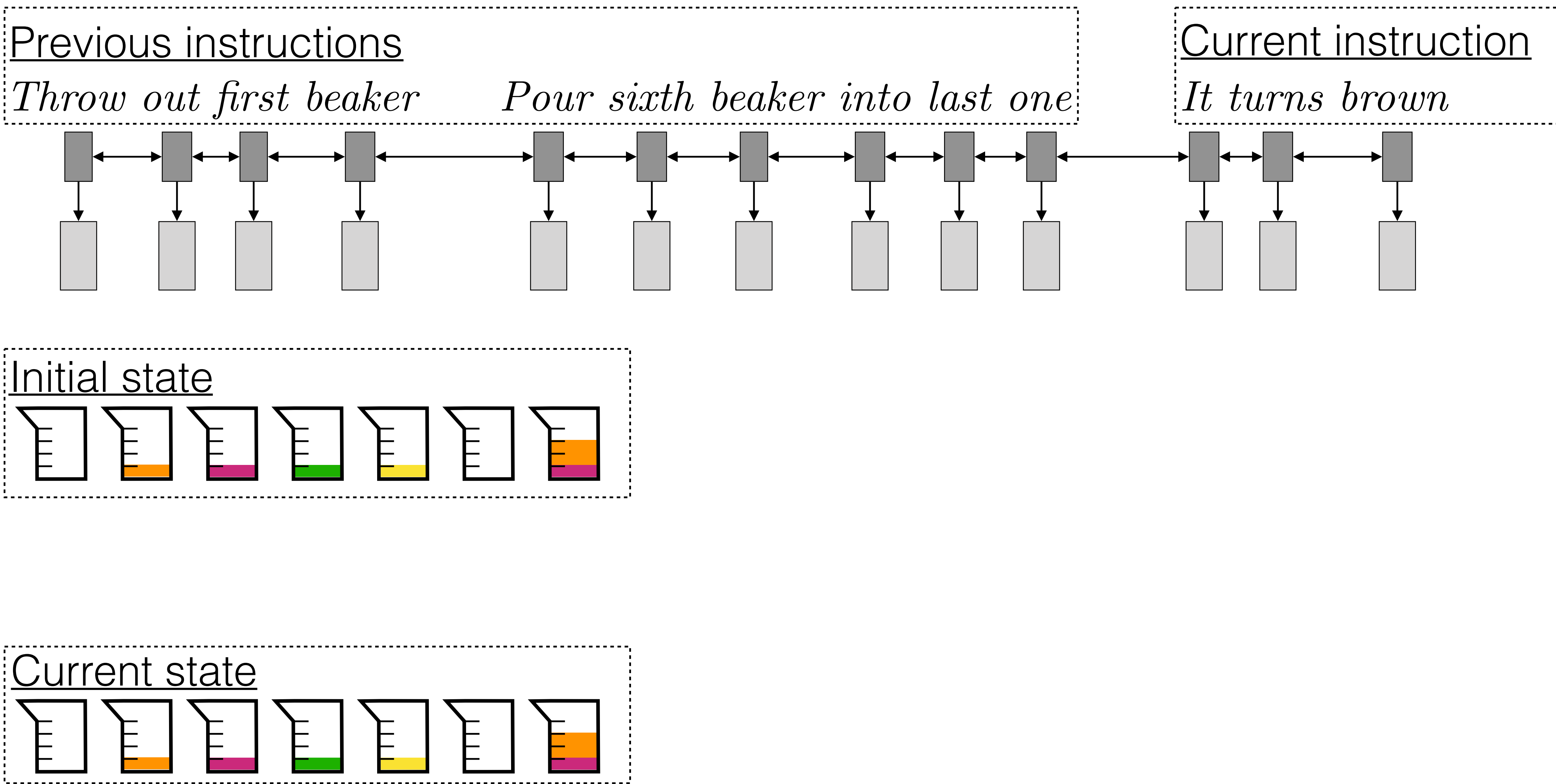


Current state



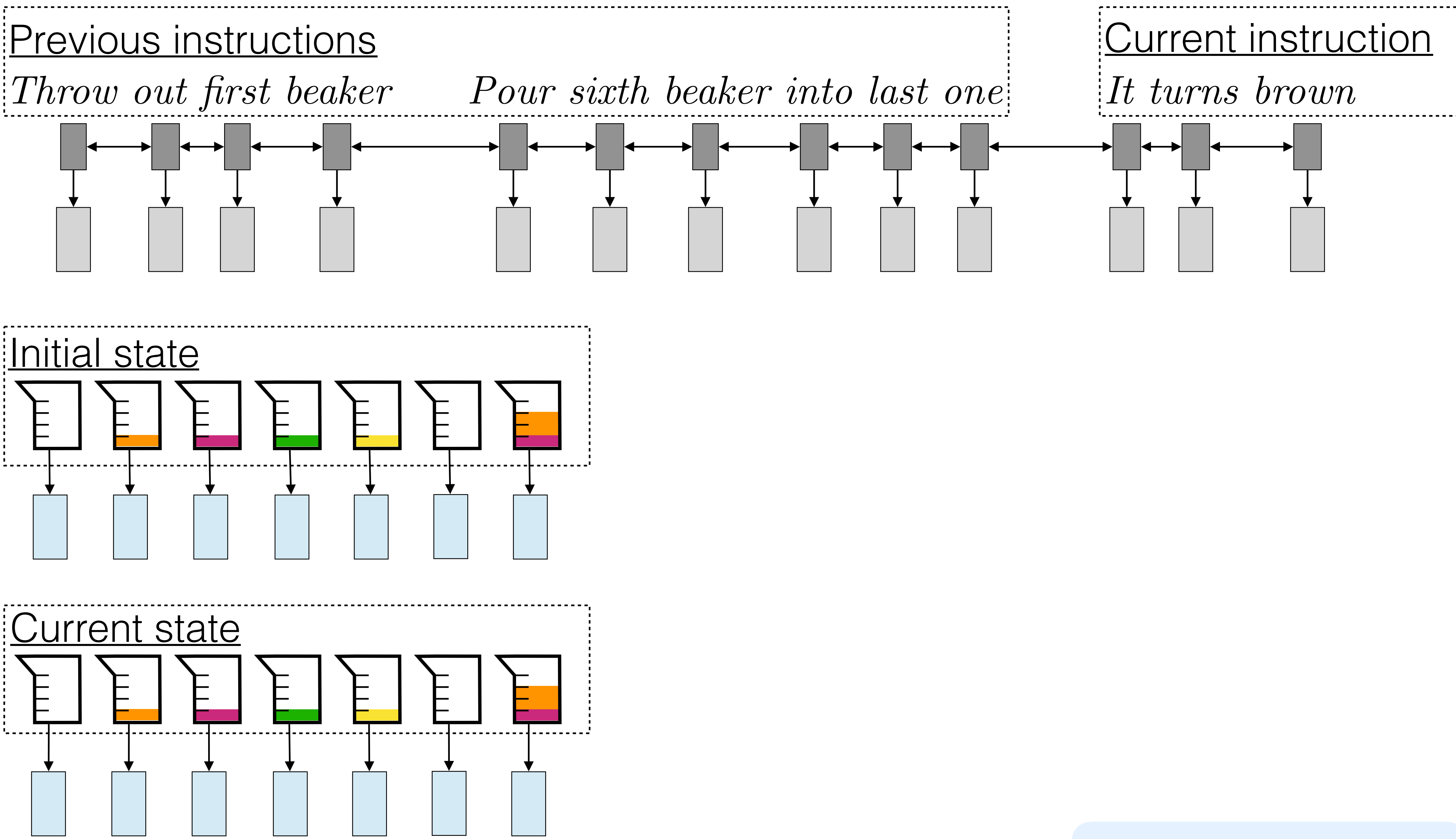
- Four inputs
- Output: a sequence of actions
- Attend over each input when generating actions

Model



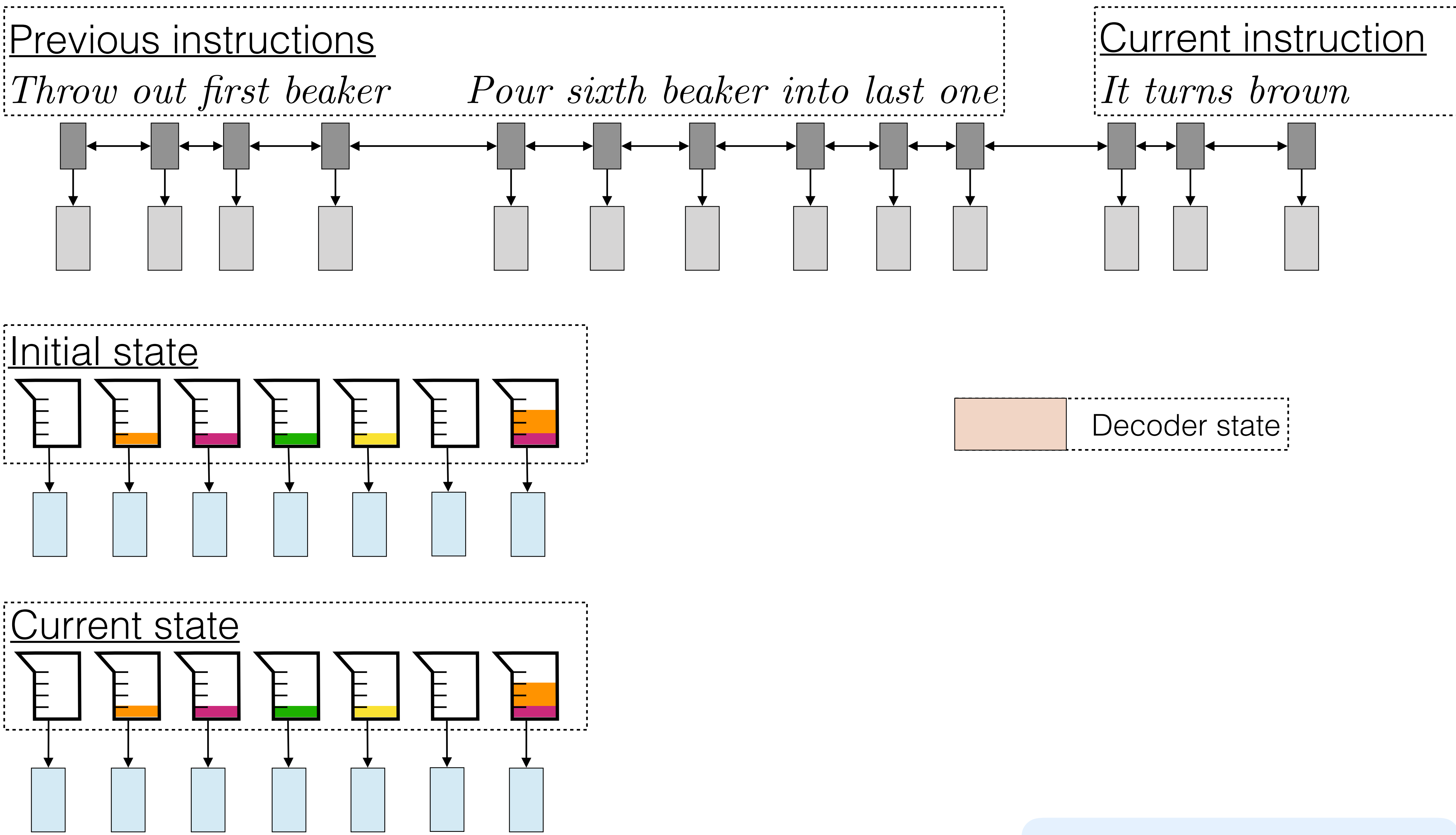
Encode instructions

Model



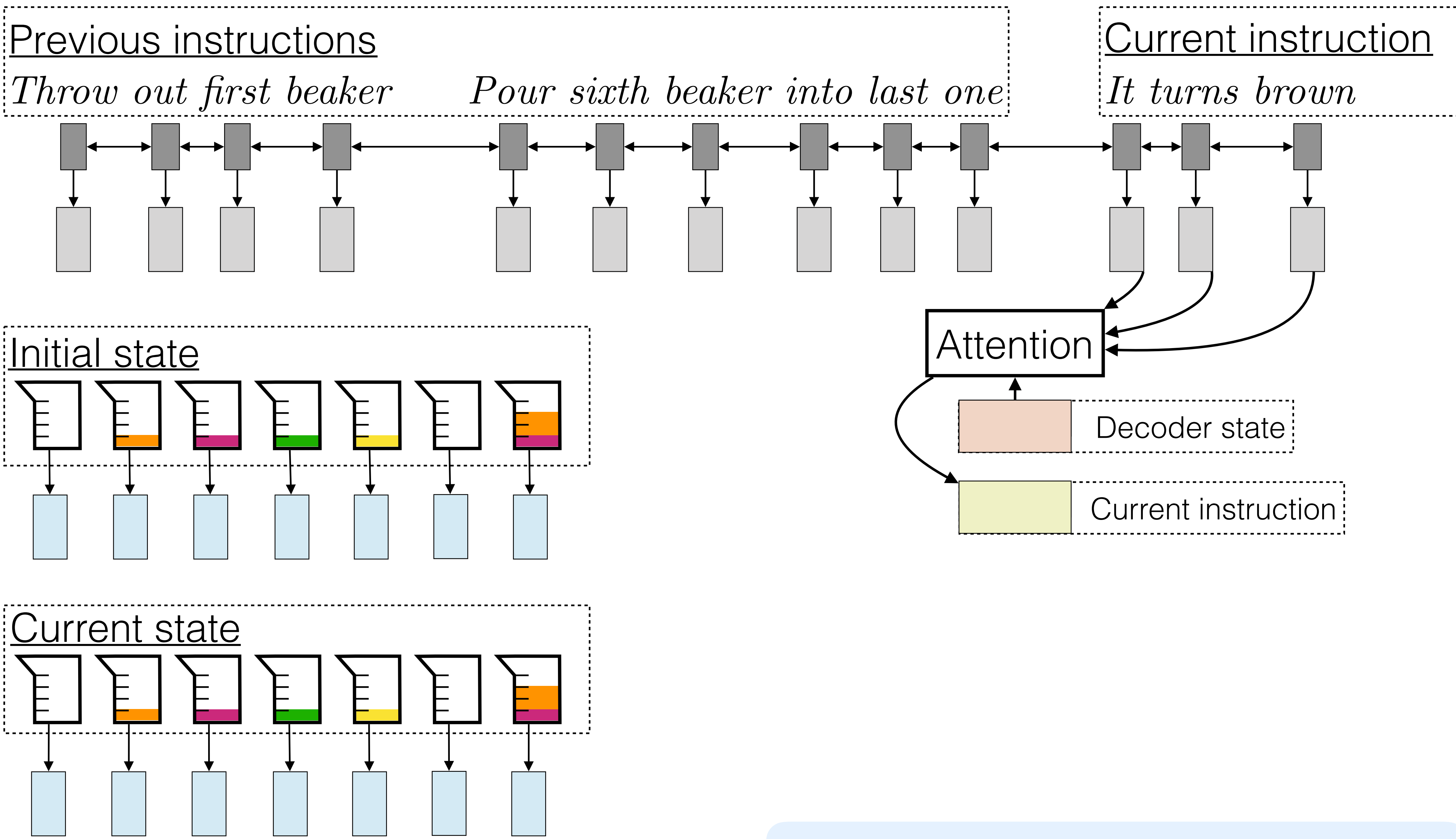
Encode states

Model



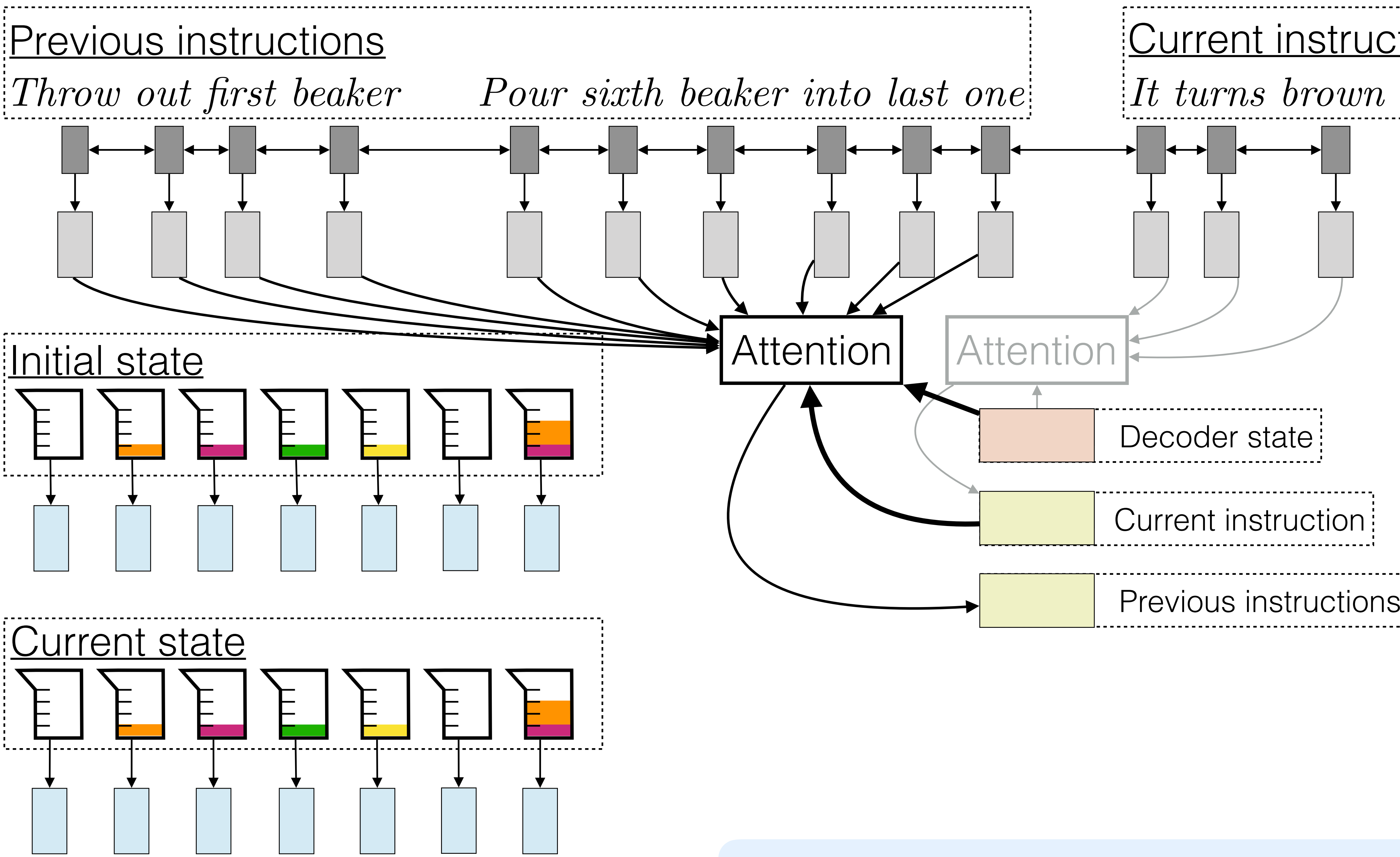
Initialize decoder

Model



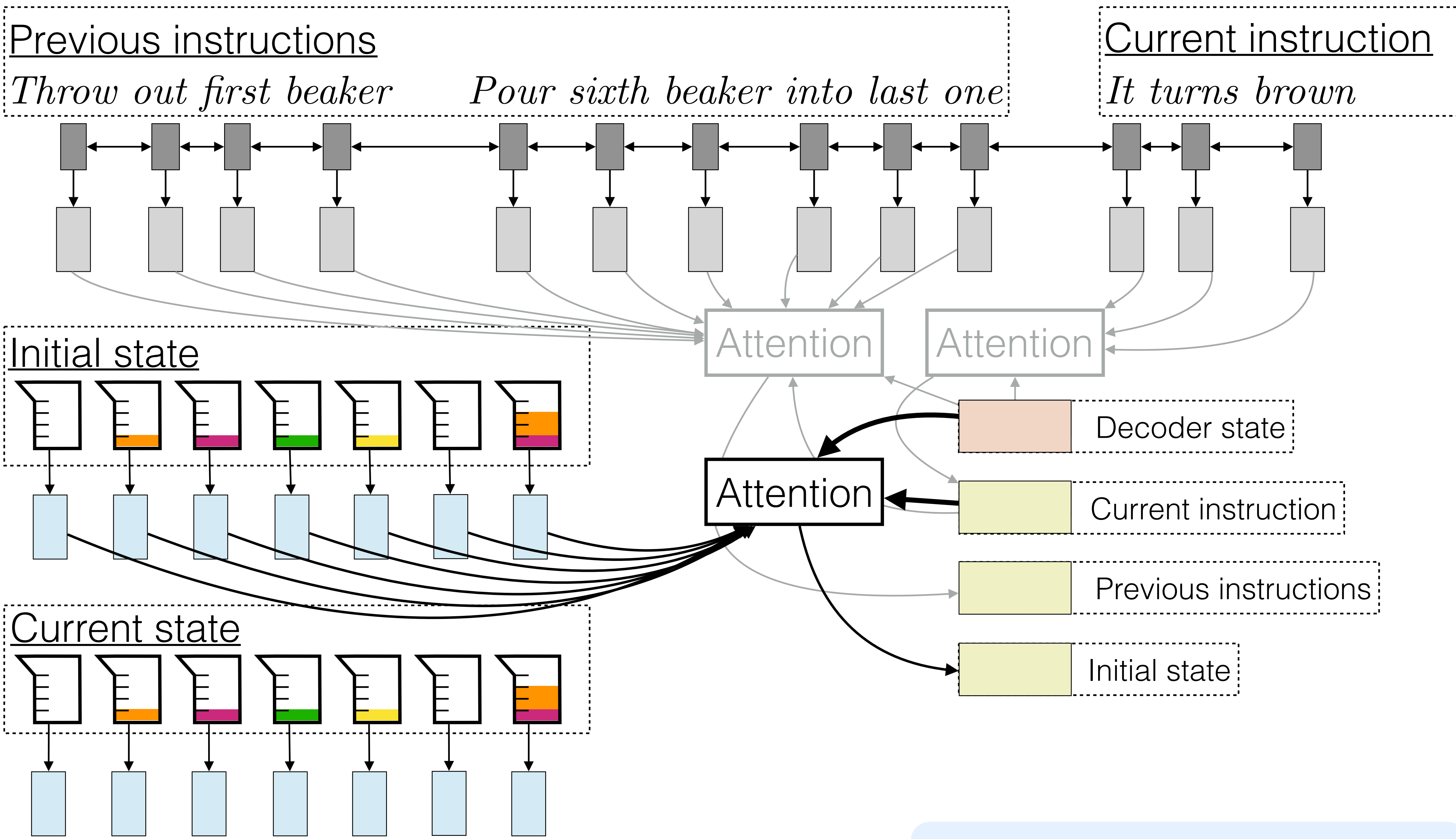
Attend over current instruction

Model



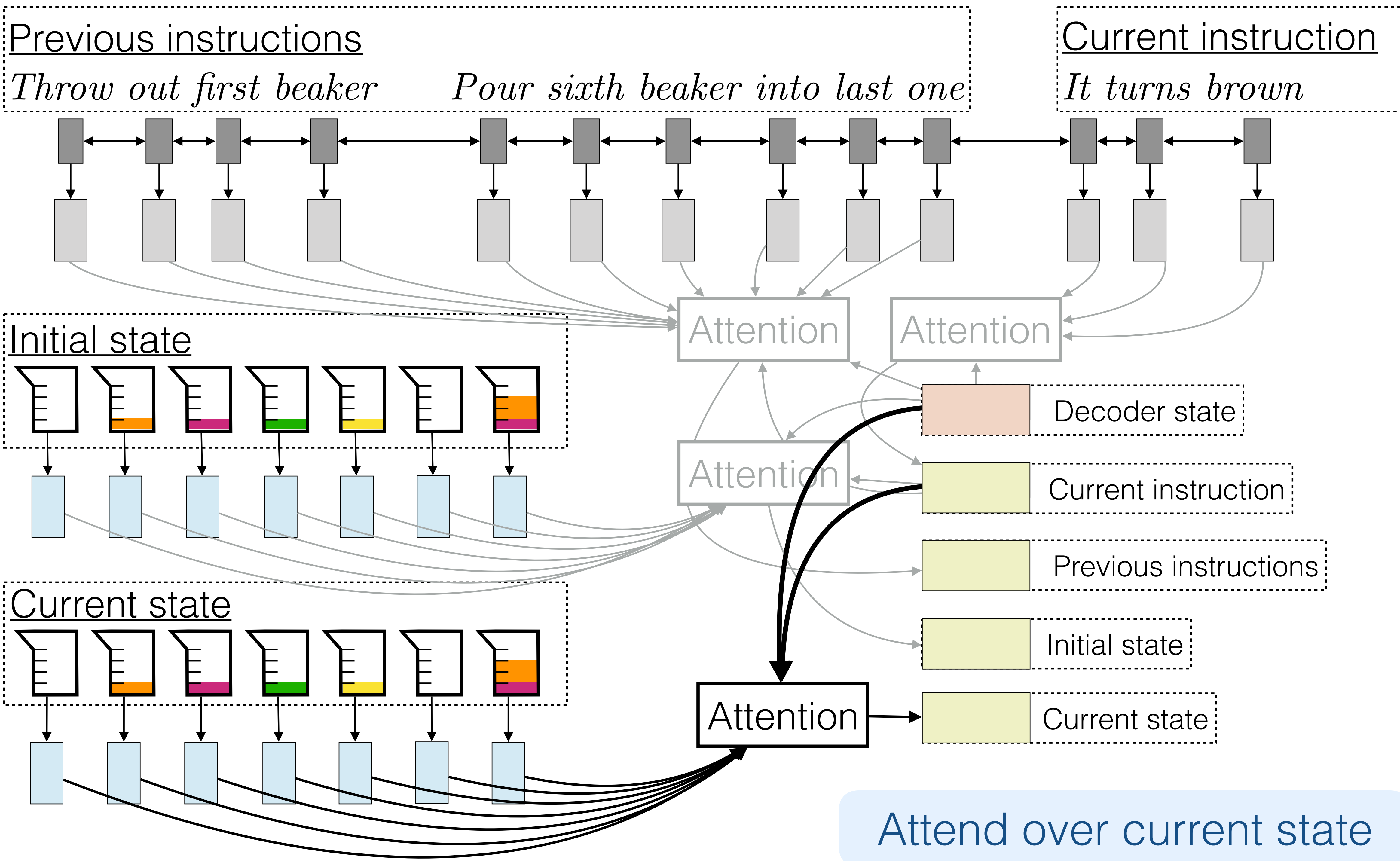
Attend over previous instructions

Model



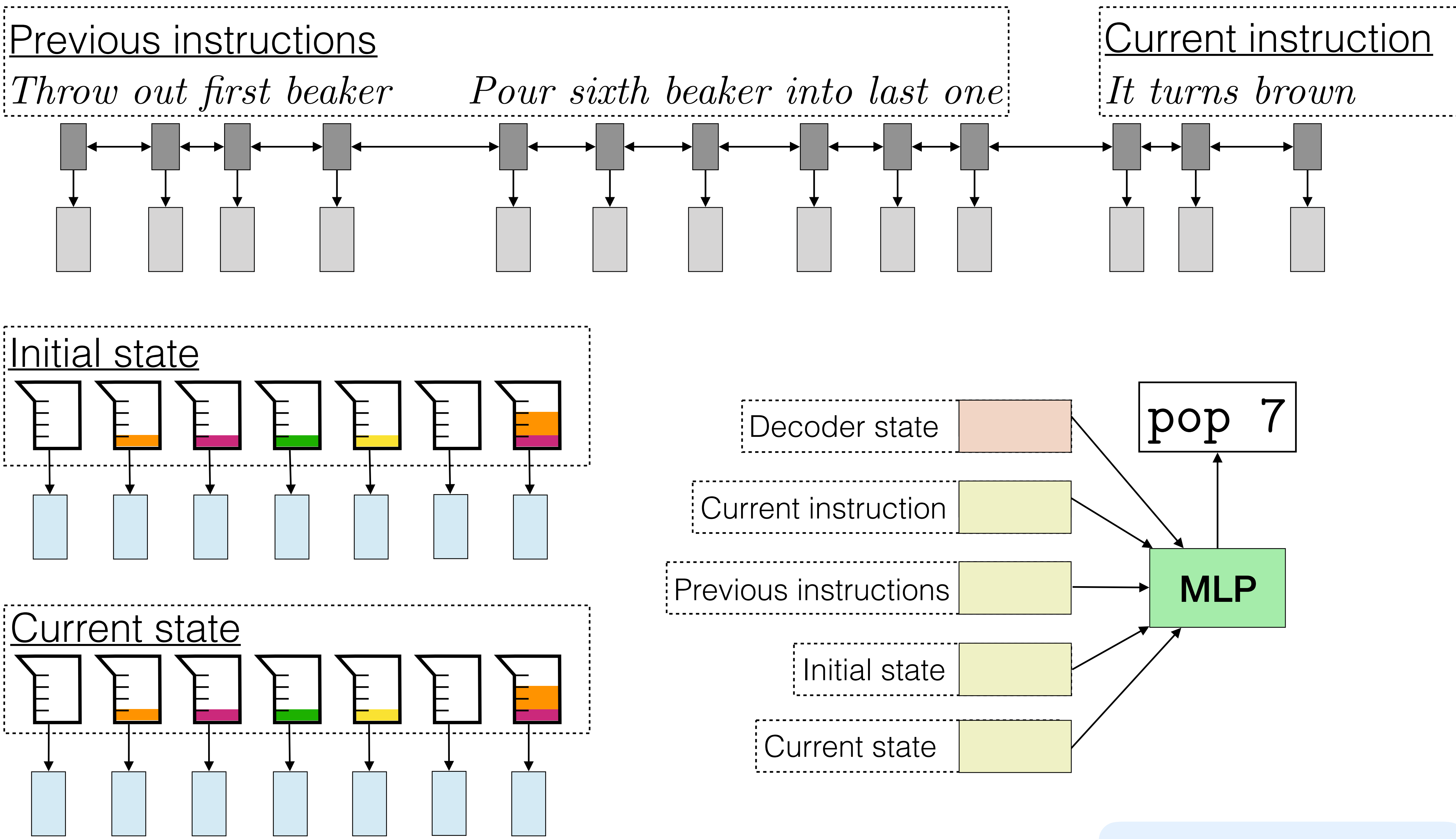
Attend over initial state

Model



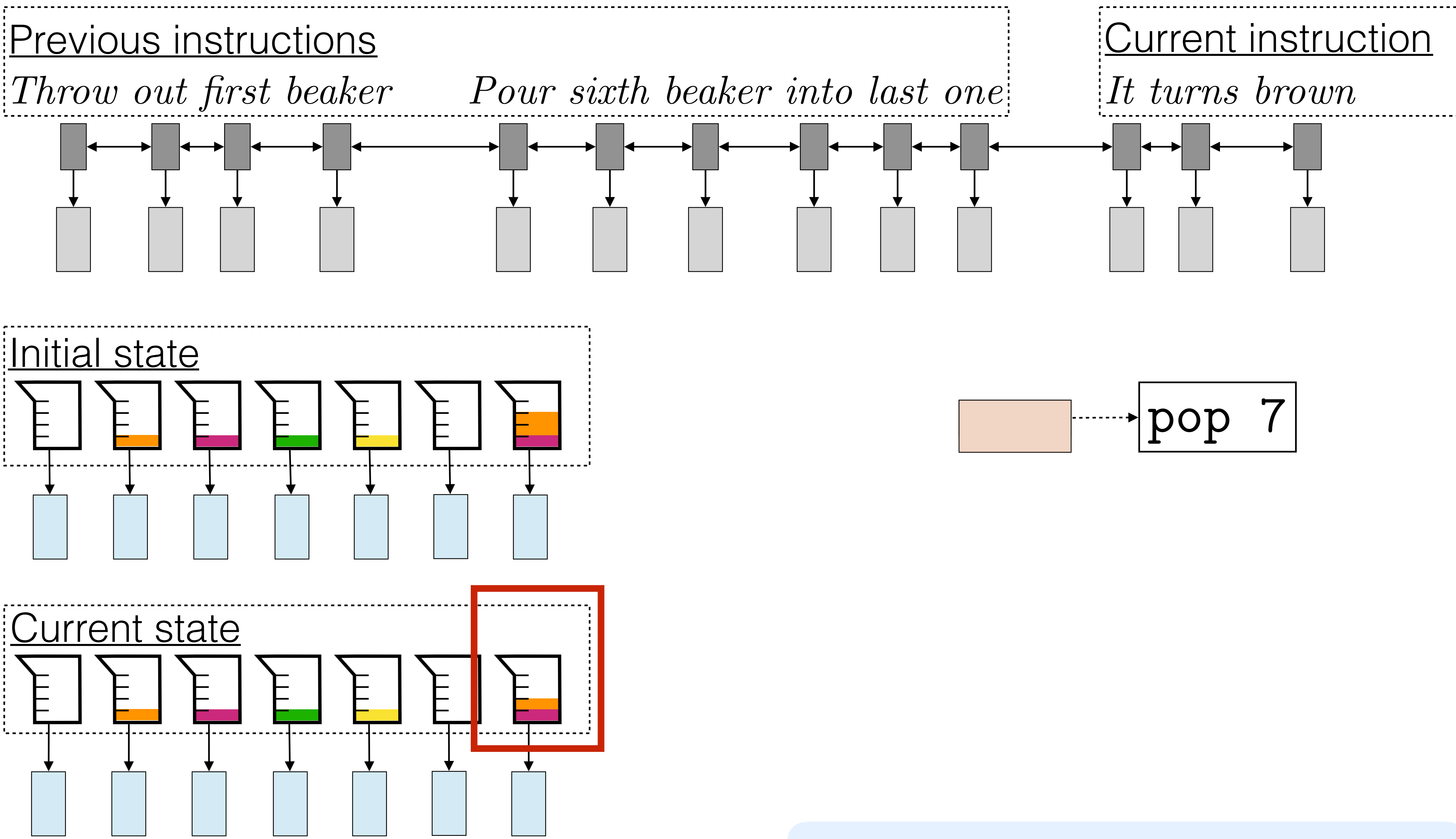
Attend over current state

Model



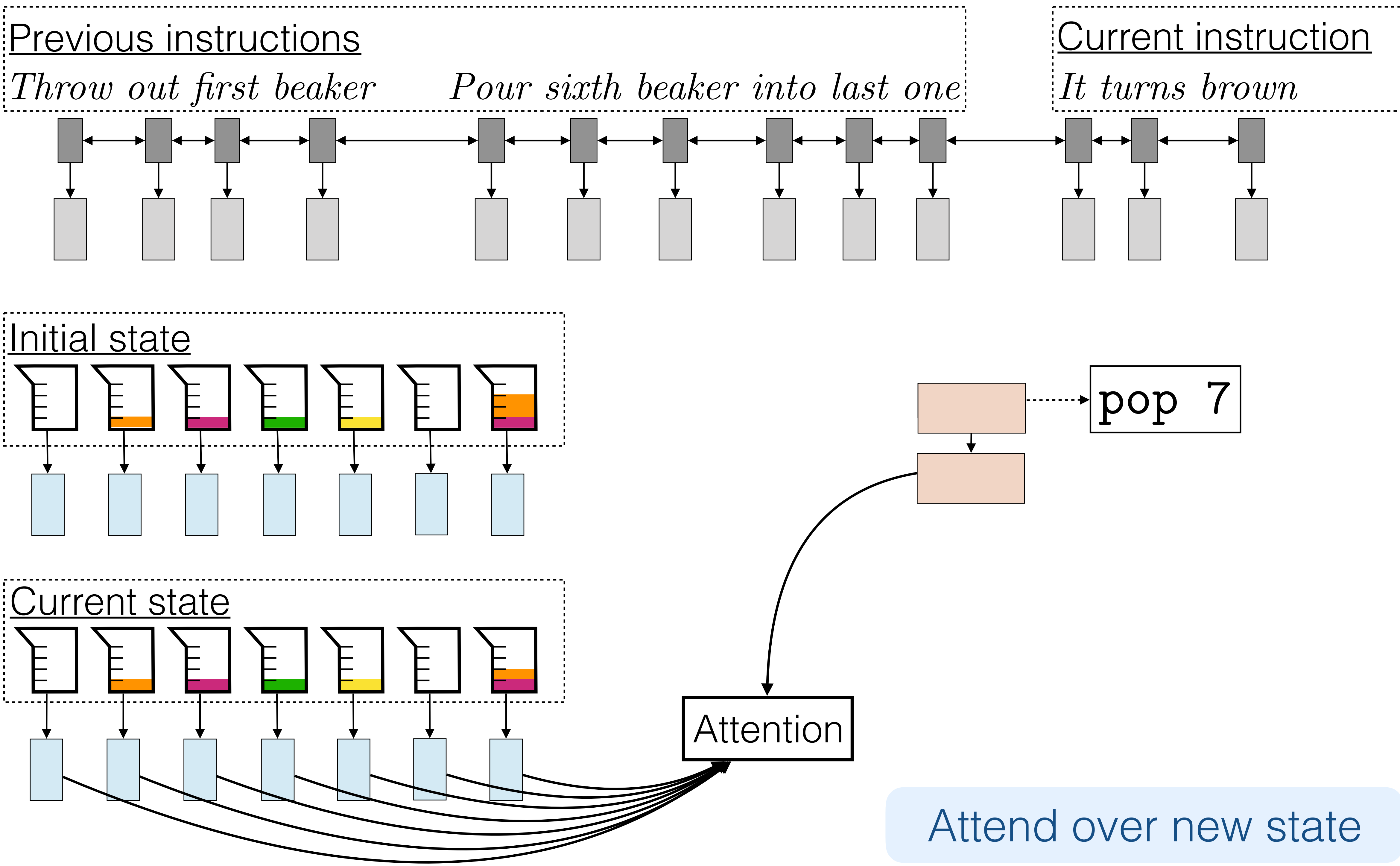
Predict action

Model



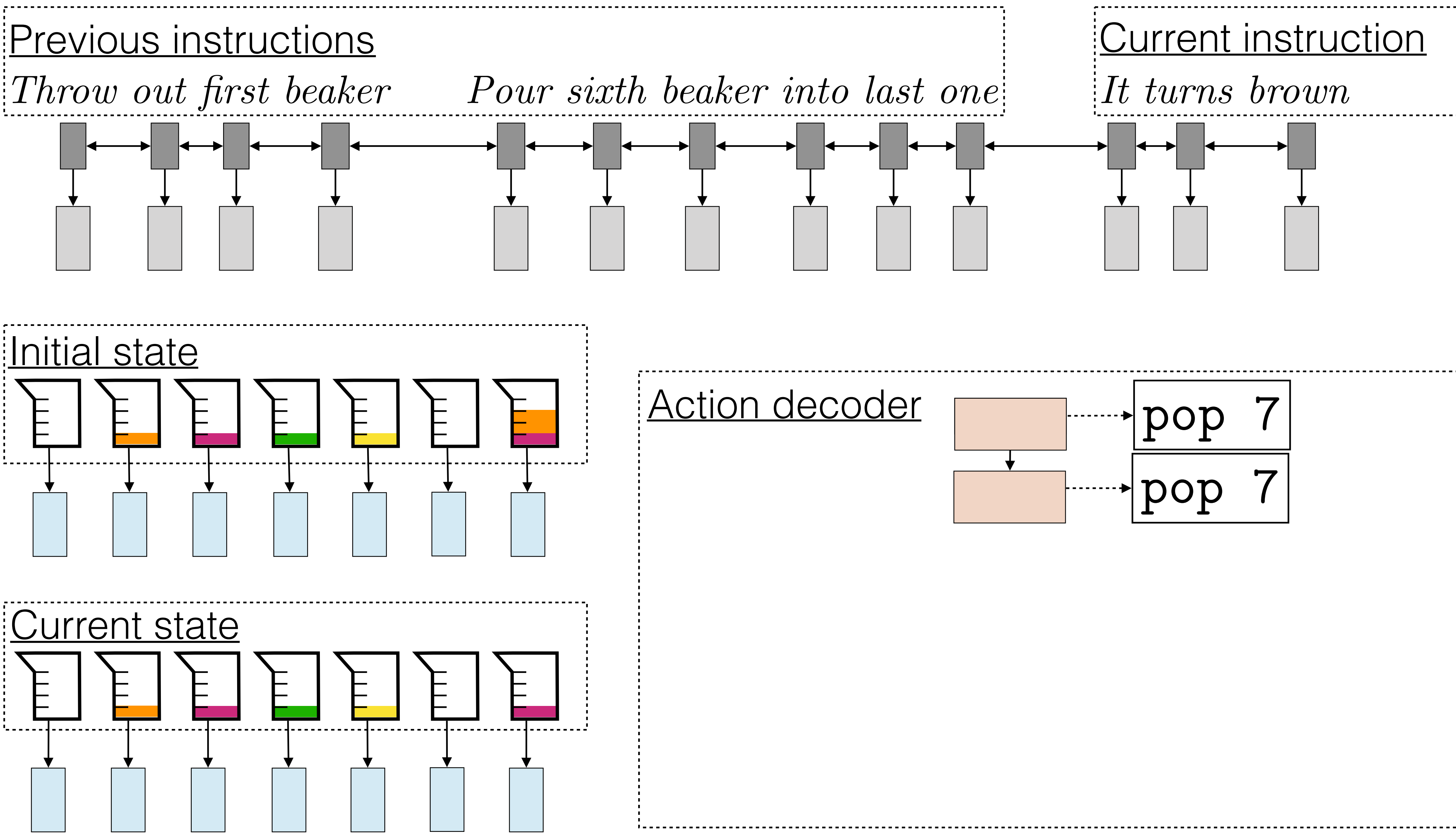
Execute action, update state

Model



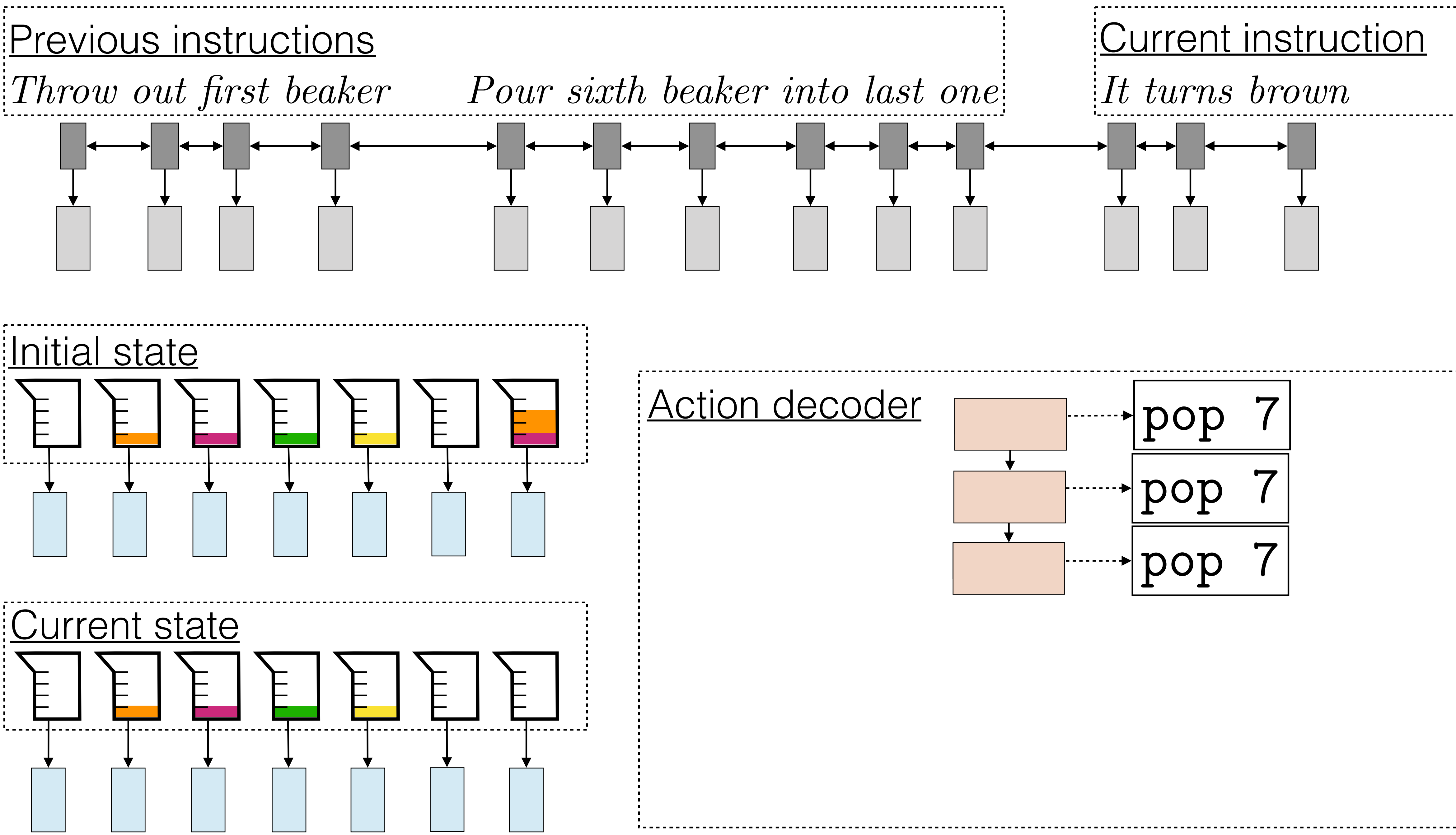
Previous instructions
Throw out first beaker
Pour sixth beaker into last one
Current instruction
It turns brown
Initial state
Current state
pop 7
Attention
Attend over new state

Model

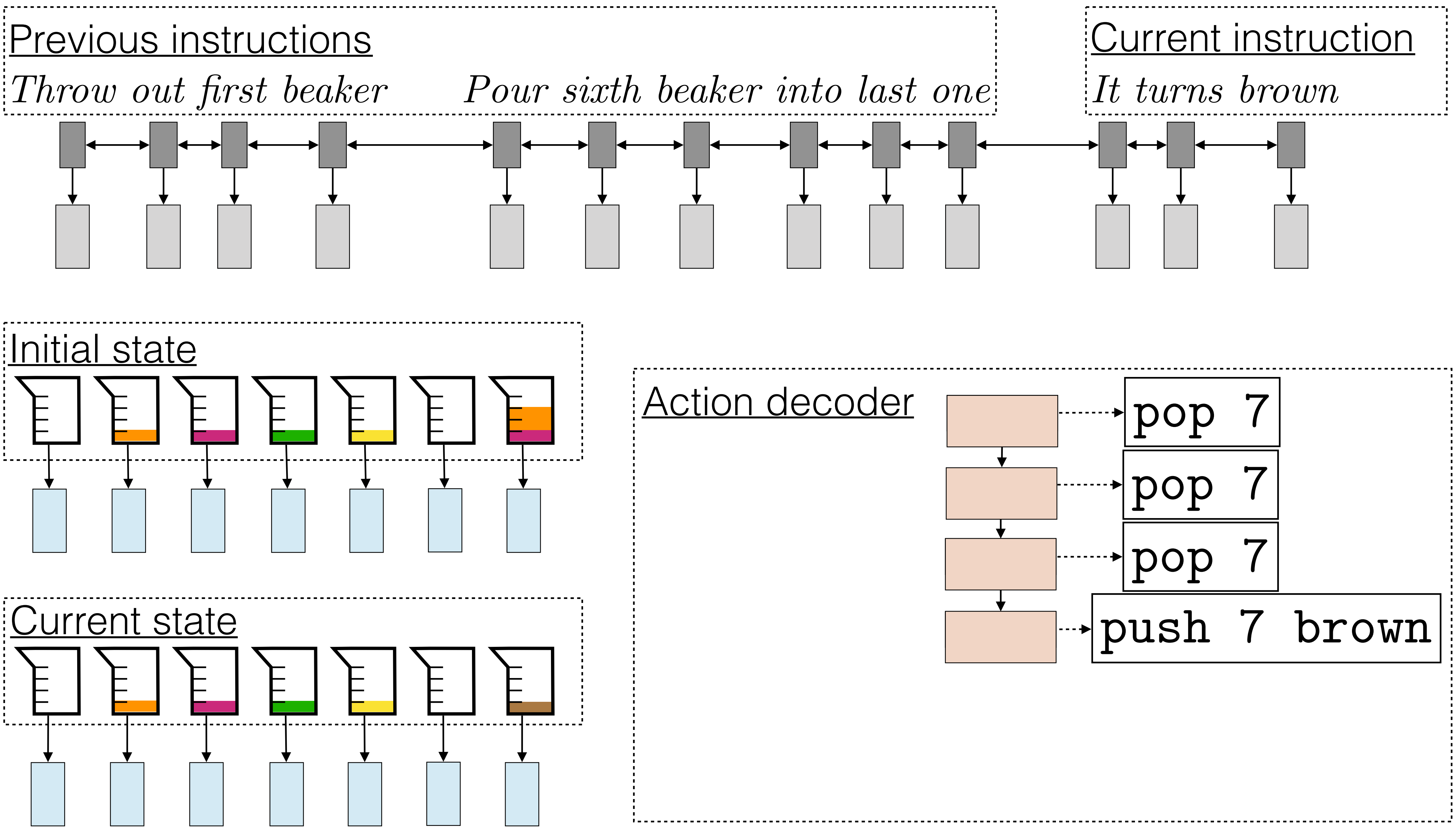


Previous instructions
Throw out first beaker
Pour sixth beaker into last one
Current instruction
It turns brown
Initial state
Current state
Action decoder
pop 7
pop 7

Model

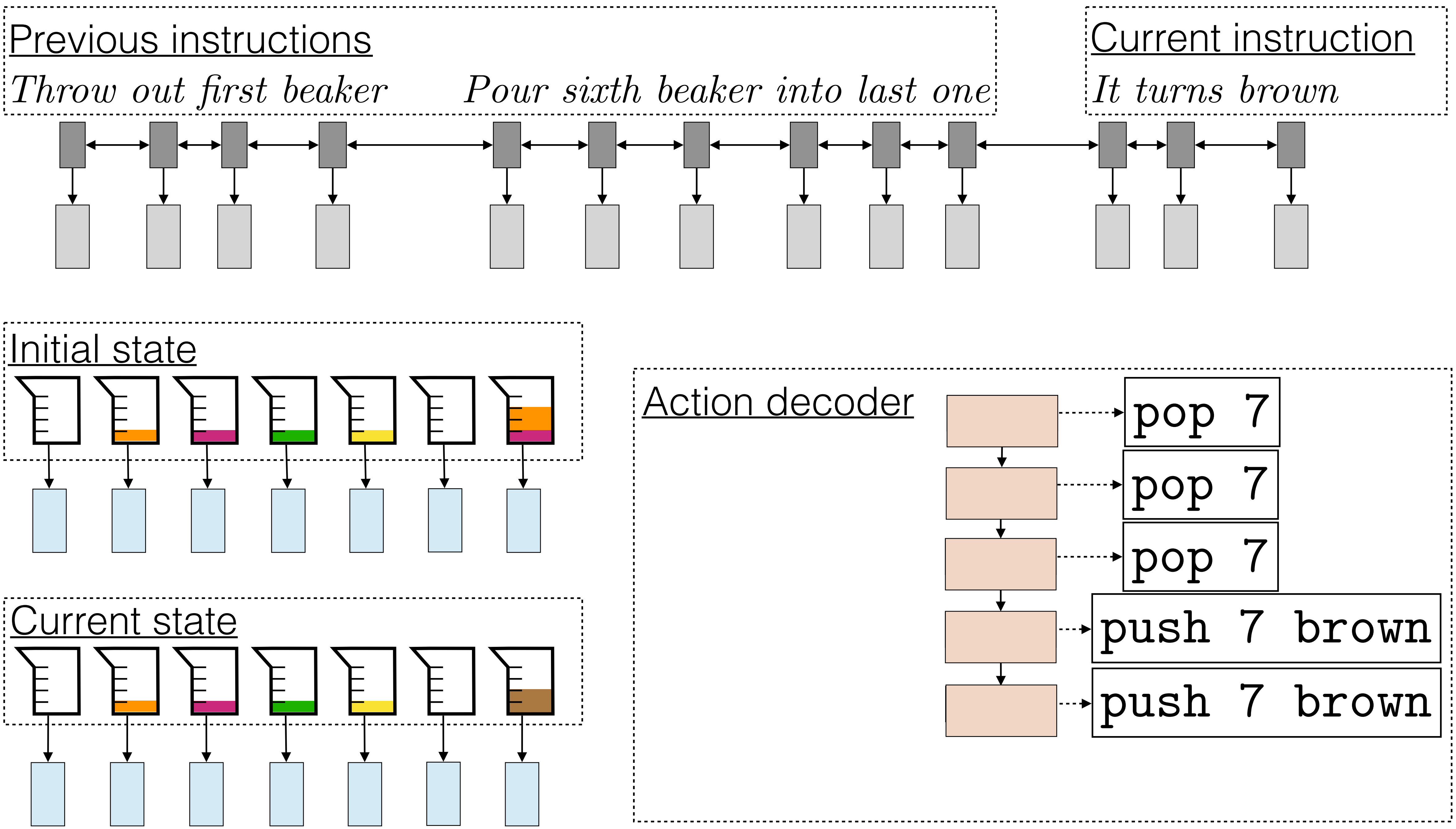

Previous instructions
Throw out first beaker
Pour sixth beaker into last one
Current instruction
It turns brown
Initial state
Current state
Action decoder
pop 7
pop 7
pop 7

Model



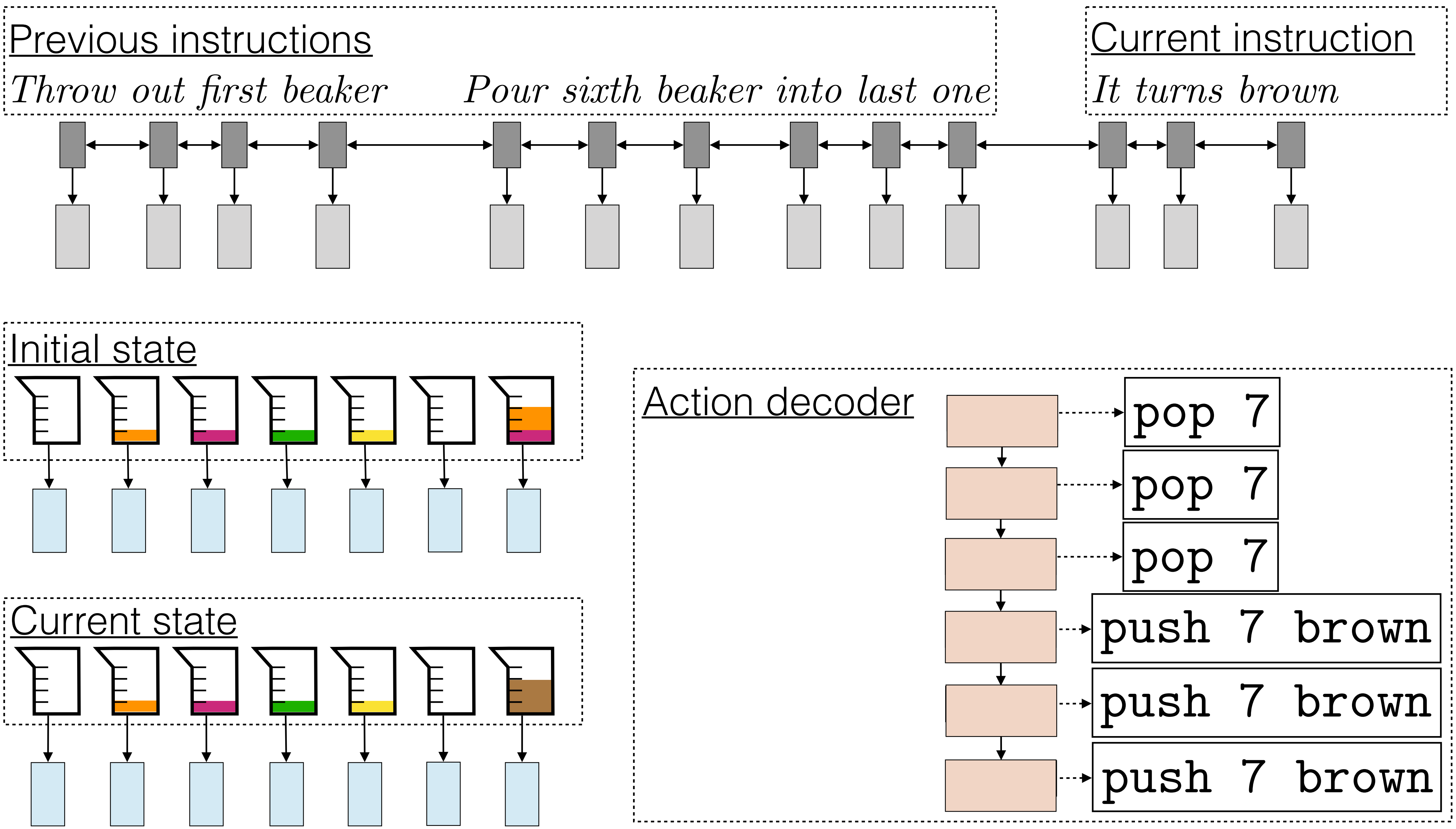
Previous instructions
Throw out first beaker
Pour sixth beaker into last one
Current instruction
It turns brown
Initial state
Current state
Action decoder
pop 7
pop 7
pop 7
push 7 brown

Model



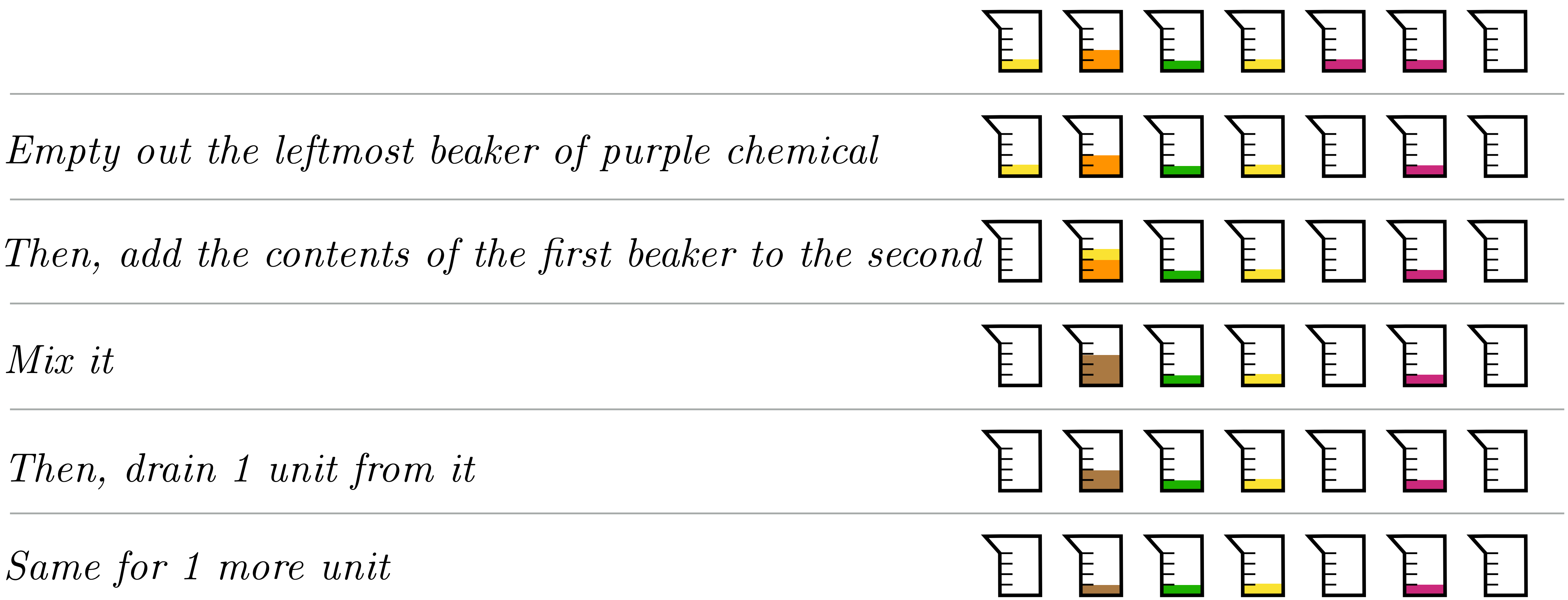
Previous instructions
Throw out first beaker
Pour sixth beaker into last one
Current instruction
It turns brown
Initial state
Current state
Action decoder
pop 7
pop 7
pop 7
push 7 brown
push 7 brown

Model


Previous instructions
Throw out first beaker
Pour sixth beaker into last one
Current instruction
It turns brown
Initial state
Current state
Action decoder
pop 7
pop 7
pop 7
push 7 brown
push 7 brown
push 7 brown

Learning from World State Annotation

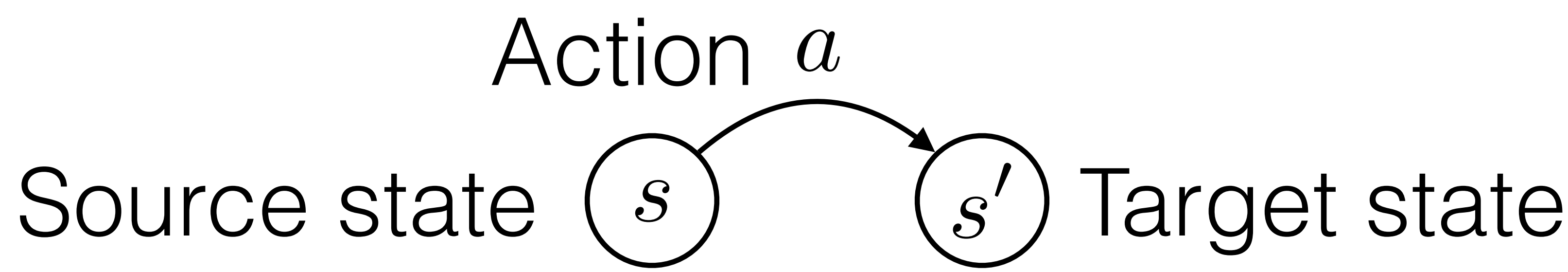
- Goal: learn a policy that maps from instructions and environment states to actions



Learning from World State Annotation

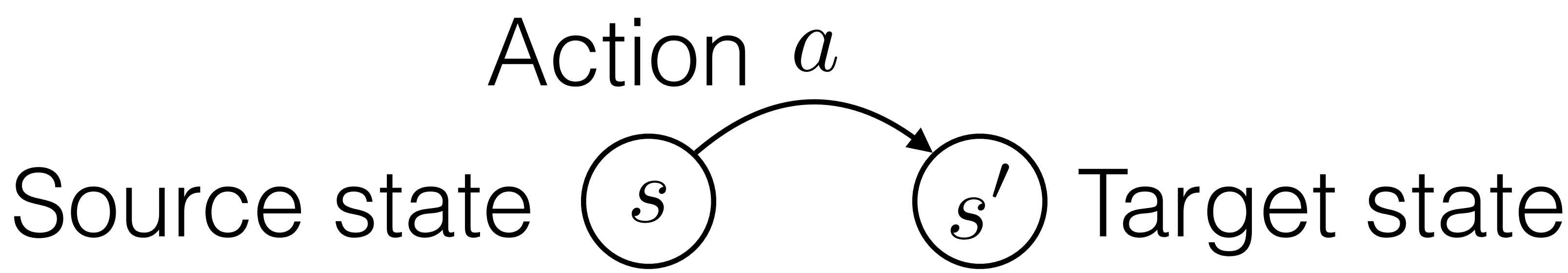
- Goal: learn a policy that maps from instructions and environment states to actions
- Approach
  - Learn through exploring the environment and observing rewards
  - Policy gradient with contextual bandit
- Challenge: overcome biases acquired early during learning

Reward Function



$$R(s, a, s') = P(s, a, s') + \phi(s') - \phi(s)$$

Reward Function



$$R(s,a,s') = \underbrace{P(s,a,s')}_{\text{Problem Reward}} + \phi(s') - \phi(s)$$

$+1$ if a stops the sequence and s' is the goal state
-1 if a stops the sequence and s' is **not** the goal state

Reward Function

$$\text{Source state } \; s \xrightarrow{\text{Action } a} s' \; \text{ Target state}$$

$$R(s, a, s') = P(s, a, s') + \underbrace{\phi(s') - \phi(s)}_{\textbf{Shaping Term}}$$

$+1$ if s' is closer to the goal state than s (moved closer)
-1 if s is closer to the goal state than s' (moved further)

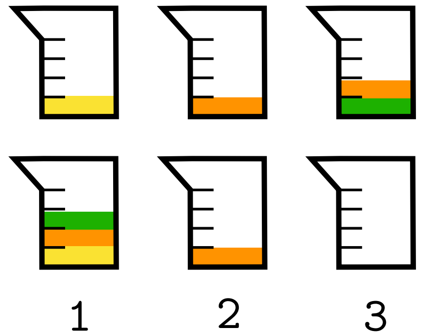
Learning Example

Iteration #1

Add the third beaker to the first

Start:

Goal:



Action Rewards:

	+	−
`pop`	0	0
`push`	0	0

Rollout:

Rewards:

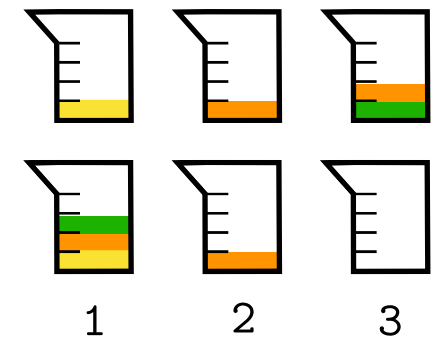
Learning Example

Iteration #1

Add the third beaker to the first

Start:

Goal:



Action Rewards:

	+	-
`pop`	0	0
`push`	0	0

Rollout:	**Rewards:**
`pop 2;`	-1
`push 1 green;`	-1
`pop 3;`	+1
`push 1 yellow;`	-1

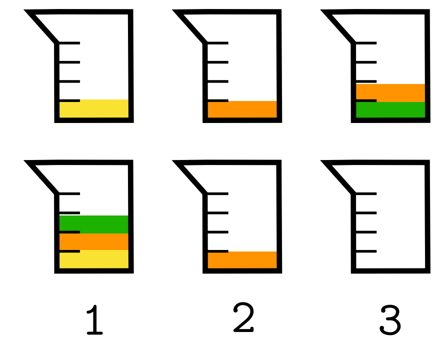
Learning Example

Iteration #1

Add the third beaker to the first

Start:

Goal:



Action Rewards:

	+	-
`pop`	1	1
`push`	0	2

No positive reward for push actions

Rollout:	**Rewards:**
`pop 2;`	-1
`push 1 green;`	-1
`pop 3;`	+1
`push 1 yellow;`	-1

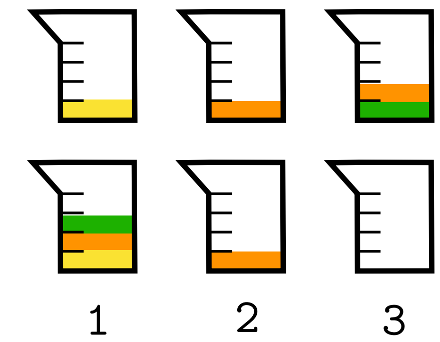
Learning Example

Iteration #2

Add the third beaker to the first

Start:

Goal:



Action Rewards:	+	-
`pop`	1	1
`push`	0	2

Rollout:

Rewards:

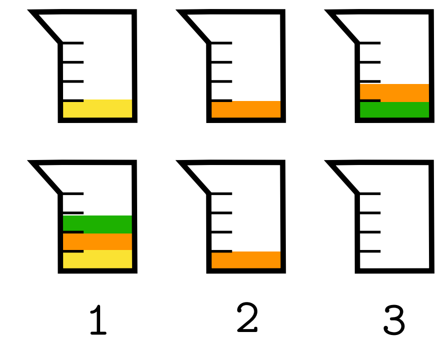
Learning Example

Iteration #2

Add the third beaker to the first

Start:

Goal:



Action Rewards:

	+	-
`pop`	1	1
`push`	0	2

Rollout:	**Rewards:**
`pop 3;`	+1
`push 1 green;`	-1
`pop 1;`	+1
`push 1 green;`	-1

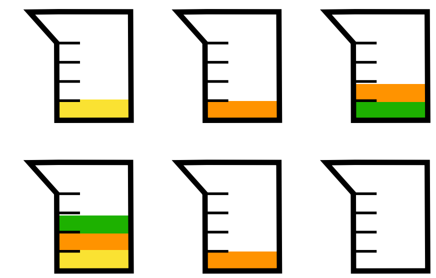
Learning Example

Iteration #2

Add the third beaker to the first

Start:

Goal:



1 2 3

Action Rewards:

	+	-
`pop`	3	1
`push`	0	4

No positive reward for push actions

Rollout:	Rewards:
`pop 3;`	+1
`push 1 green;`	-1
`pop 1;`	+1
`push 1 green;`	-1

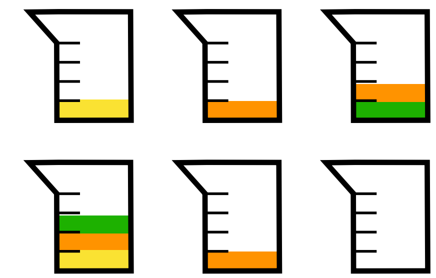
Learning Example

Iteration #3

Add the third beaker to the first

Start:

Goal:



1 2 3

Action Rewards:

	+	-
`pop`	3	1
`push`	0	4

Quickly learned a strong bias against push actions

Rollout:	**Rewards:**
`pop 3;`	+1
`pop 3;`	+1
`pop 1;`	-1

Learned Biases

- Early during learning, model learns it can get positive reward by predicting the `pop` actions
- Less likely to get positive reward with `push` action
- Becomes biased against `push` — during later exploration, `push` is never sampled!
- Compounding effect: never learns to generate `push` actions

Single-step Reward Observation

- **Our approach:** observe reward of <u>all</u> actions by looking one step ahead during exploration
- Observe reward for actions like `push`

Learning Algorithm

For each training example:

1. Rollout: sample sequence of state-action pairs from the current policy

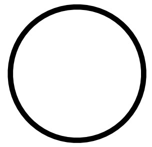
2. For each state visited in the rollout,

   A. For each possible action, execute action and observe reward

Single-step Observation

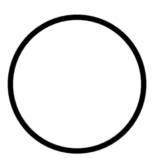
3. Update parameters based on observed rewards for all states and actions

Simple Exploration

Start state 

Only observe states along sampled trajectory

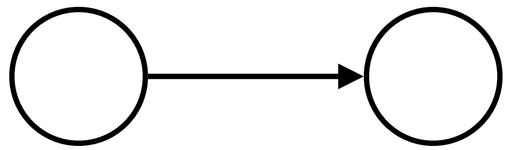
Single-step Reward Observation

Start state 

Observe sampled states **and** single-step ahead

Simple Exploration

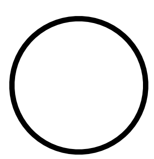
Start state



Only observe states along sampled trajectory

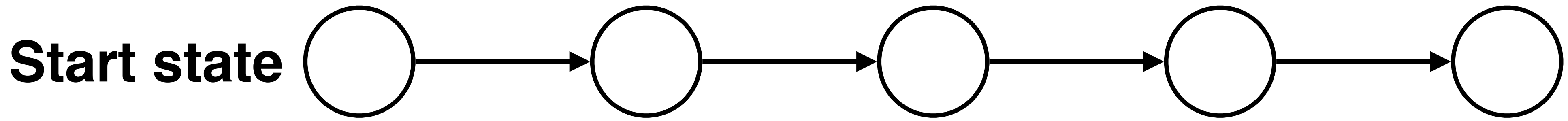
Single-step Reward Observation

Start state



Observe sampled states **and** single-step ahead

Simple Exploration

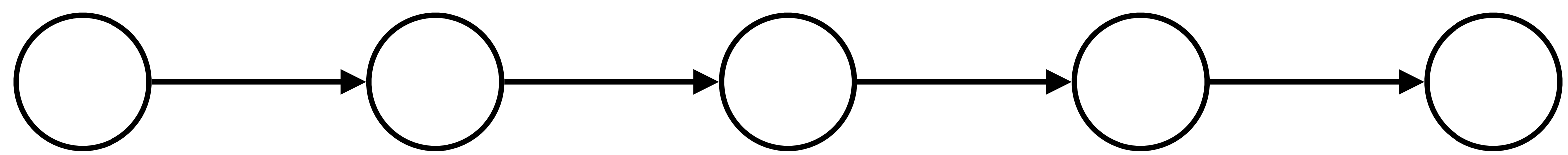


Only observe states along sampled trajectory

Single-step Reward Observation

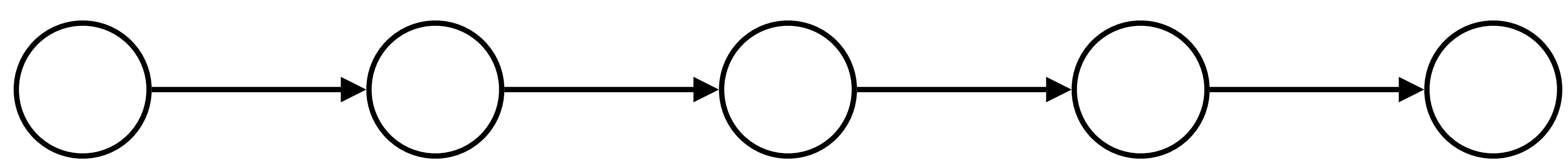
Observe sampled states **and** single-step ahead

Simple Exploration

Start state 

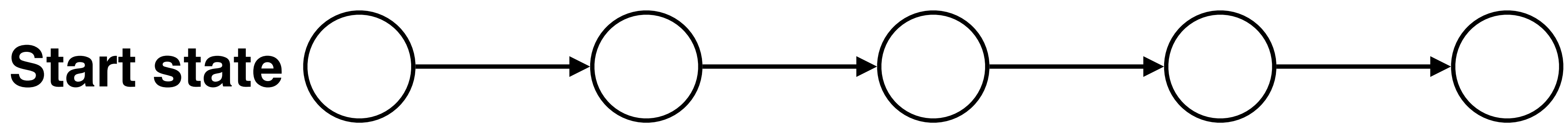
Only observe states along sampled trajectory

Single-step Reward Observation

Start state 

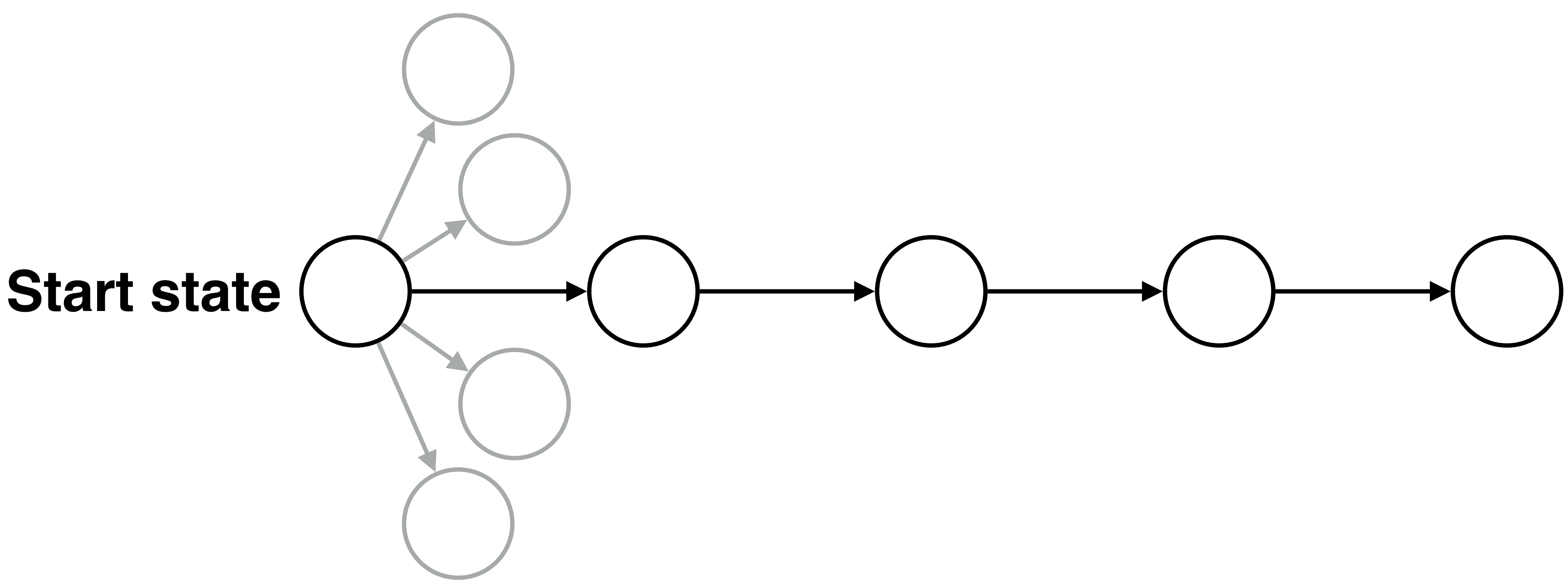
Observe sampled states **and** single-step ahead

Simple Exploration



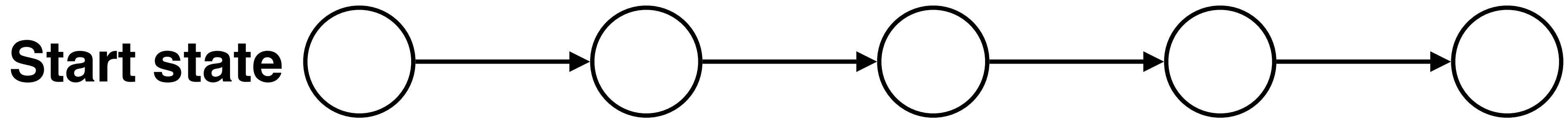
Only observe states along sampled trajectory

Single-step Reward Observation



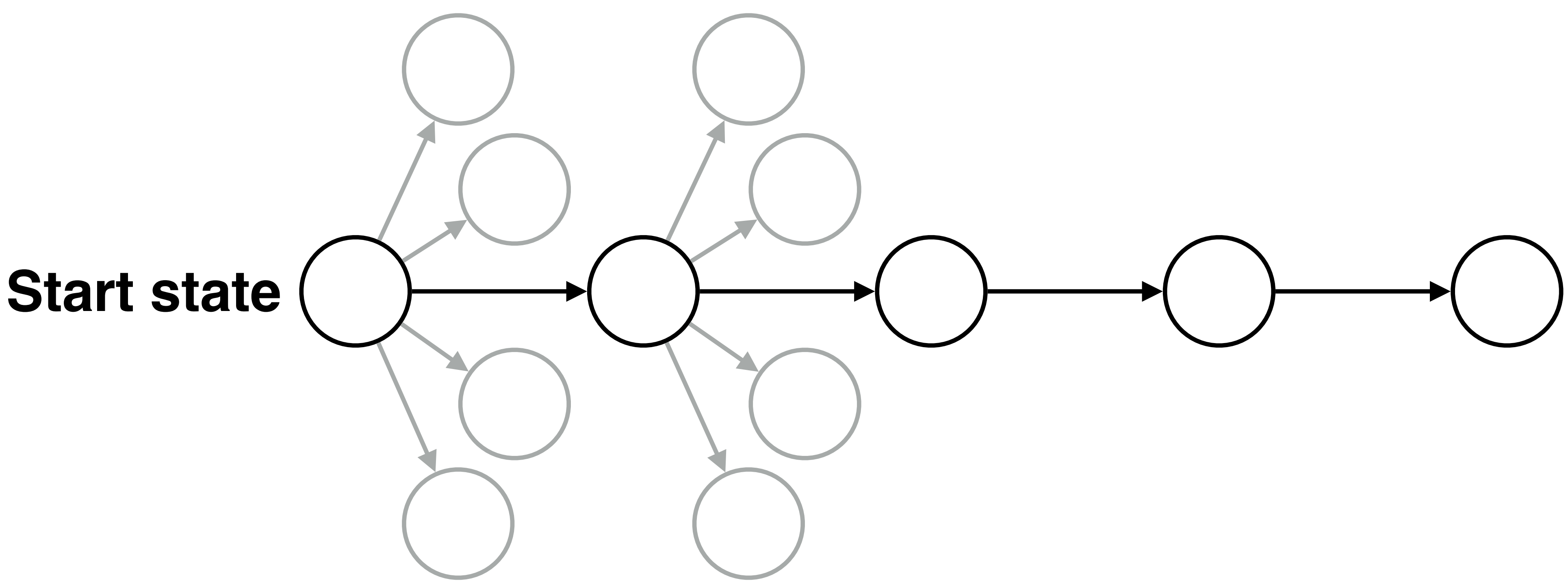
Observe sampled states **and** single-step ahead

Simple Exploration



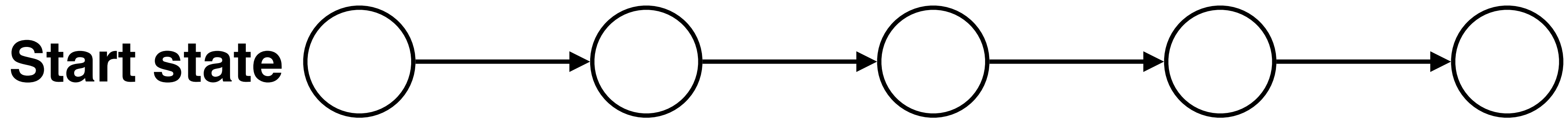
Only observe states along sampled trajectory

Single-step Reward Observation



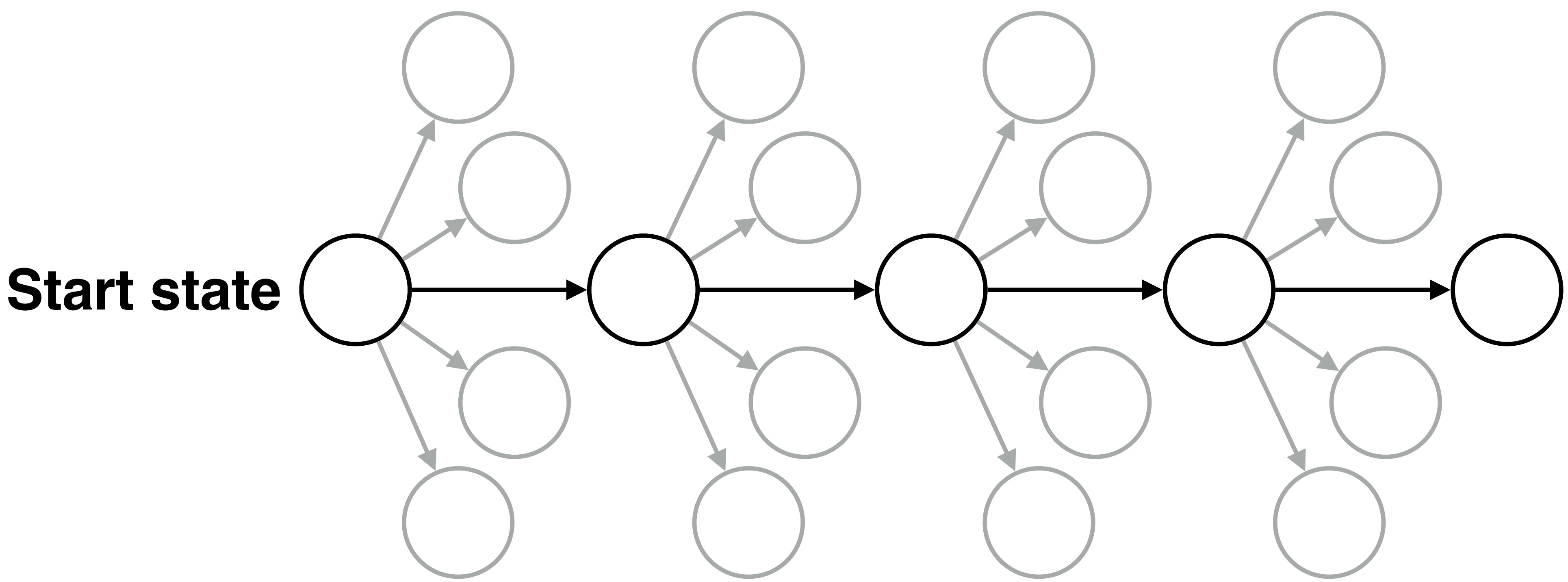
Observe sampled states **and** single-step ahead

Simple Exploration



Only observe states along sampled trajectory

Single-step Reward Observation



Observe sampled states **and** single-step ahead

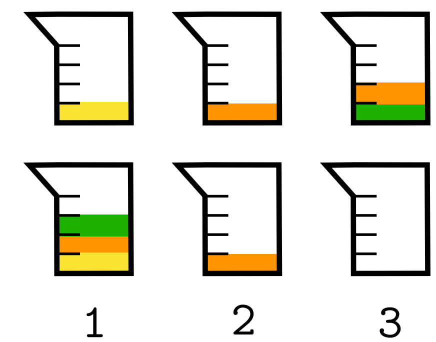
Single-step Observation

Iteration #4

Add the third beaker to the first

Start:

Goal:



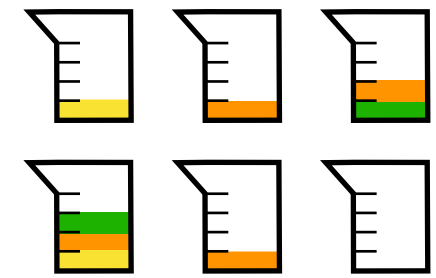
Single-step Observation

Iteration #4

Add the third beaker to the first

Start:

Goal:



1 2 3

Rollout:

```
pop 3;
pop 3;
pop 1;
```

Current State:

Single-step Observation:

Single-Step Actions:

Single-step Observation

Iteration #4

Add the third beaker to the first

Start: 

Goal: 

1 2 3

Rollout:

```
pop 3;
pop 3;
pop 1;
```

Current State:

Single-step Observation:

Single-Step Actions:

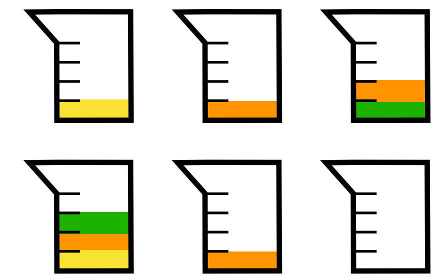
-1 `pop 1;`

Single-step Observation

Iteration #4

Add the third beaker to the first

Start:
Goal:



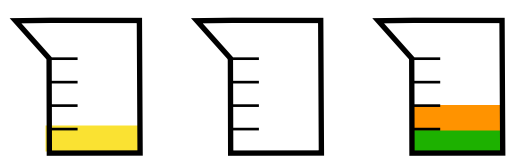
1 2 3

Rollout:

```
pop 3;
pop 3;
pop 1;
```

Current State:

Single-step Observation:



Single-Step Actions:

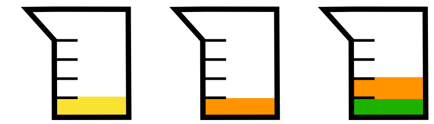
```
-1  pop 1;
-1  pop 2;
```

Single-step Observation

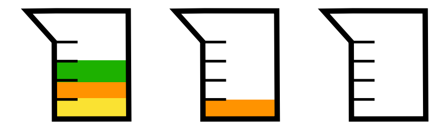
Iteration #4

Add the third beaker to the first

Start:



Goal:



1 2 3

Rollout:

```
pop 3;
pop 3;
pop 1;
```

Current State:

Single-step Observation:

Single-Step Actions:

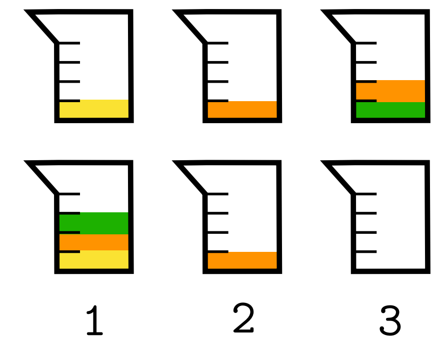
-1	`pop 1;`
-1	`pop 2;`
+1	`pop 3;`

Single-step Observation

Iteration #4

Add the third beaker to the first

Start: **Goal:**



Rollout:

```
pop 3;
pop 3;
pop 1;
```

Current State:

Single-step Observation:

Single-Step Actions:

```
-1   pop 1;
-1   pop 2;
+1   pop 3;
        ⋮
+1   push 1 orange;
```

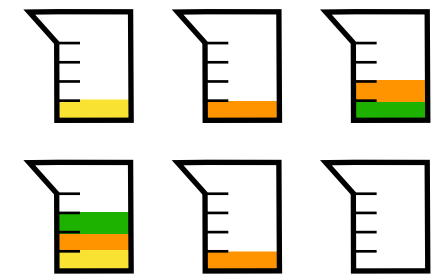
Single-step Observation

Iteration #4

Add the third beaker to the first

Start:

Goal:



1 2 3

Rollout:

```
pop 3;
pop 3;
pop 1;
```

Current State:

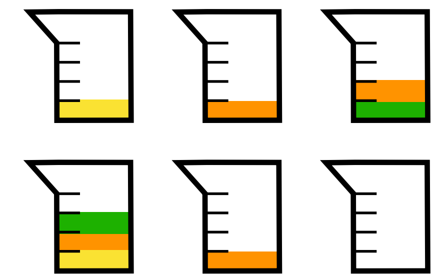
Single-Step Actions:

Single-step Observation

Iteration #4

Add the third beaker to the first

Start:
Goal:



1 2 3

Rollout:

```
pop 3;
pop 3;
pop 1;
```

Current State:

Single-Step Actions:

-1	`pop 1;`
-1	`pop 2;`
+1	`pop 3;`
	⋮
+1	`push 1 orange`

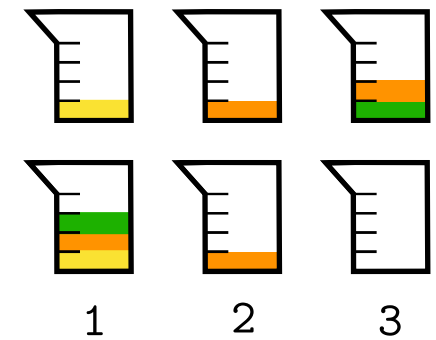
Single-step Observation

Iteration #4

Add the third beaker to the first

Start:

Goal:



Rollout:

```
pop 3;
pop 3;
pop 1;
```

Current State:

Single-Step Actions:

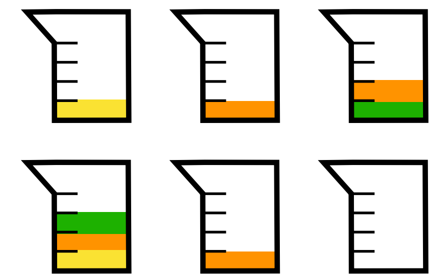
Single-step Observation

Iteration #4

Add the third beaker to the first

Start:

Goal:



1 2 3

Rollout:

```
pop 3;
→ pop 3;
pop 1;
```

Current State:

Single-Step Actions:

```
-1   pop 1;
-1   pop 2;
-1   pop 3;
        ⋮
+1   push 1 orange
```

Single-step Observation

Iteration #4

Add the third beaker to the first

Start: 

Goal: 

Rollout:

```
pop 3;
pop 3;
→ pop 1;
```

Current State:

Single-Step Actions:

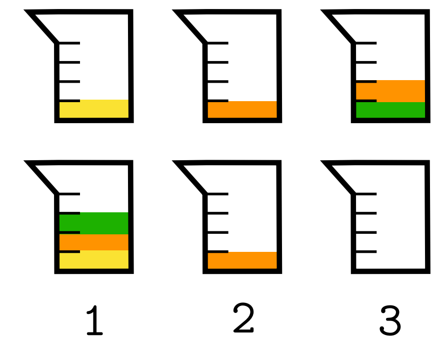
Single-step Observation

Iteration #4

Add the third beaker to the first

Start:

Goal:



Rollout:

```
  pop 3;
  pop 3;
→ pop 1;
```

Current State:

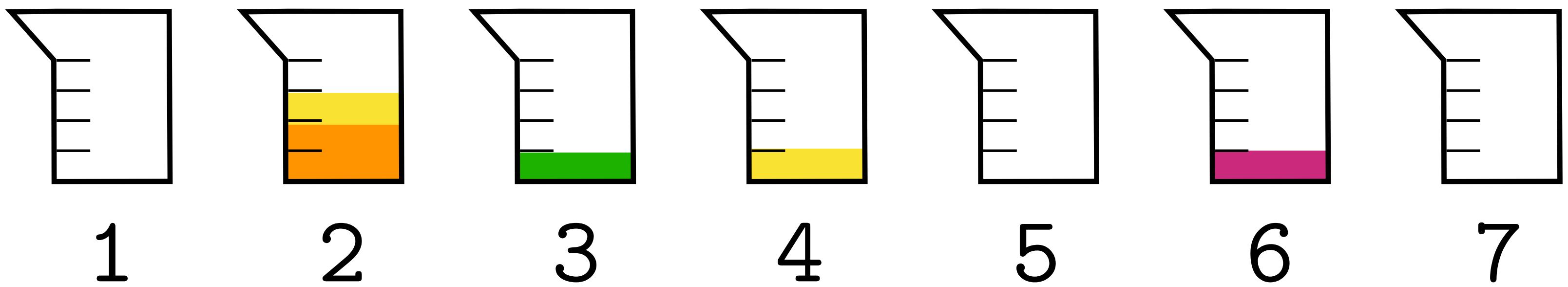
Single-Step Actions:

-1	`pop 1;`
-1	`pop 2;`
-1	`pop 3;`
	⋮
-1	`push 1 orange`
+1	`push 1 yellow`

Experimental Setup

- SCONE (Long et al. 2016): Alchemy, Scene, Tangrams
- Training data: start state and a sequence of instructions and goal states
- Standard evaluation metric: after following a sequence of instructions, is the world state correct?

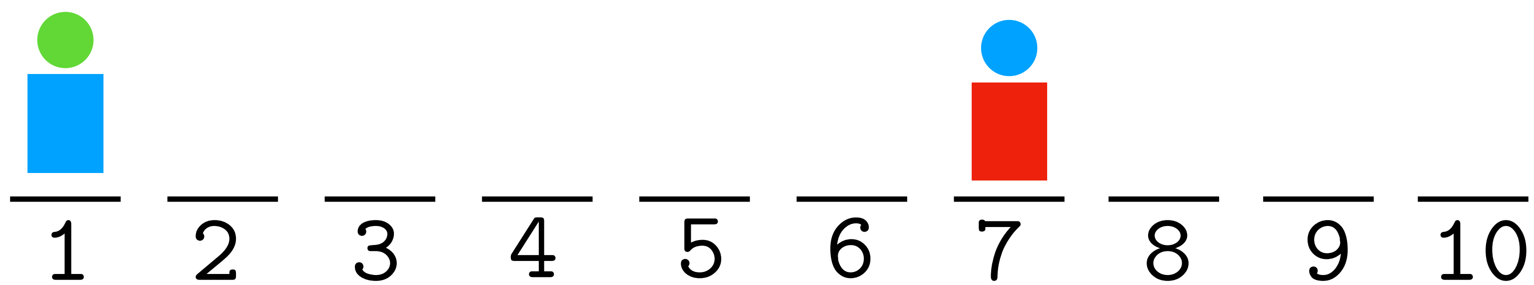
Alchemy



Mix it

```
pop  2;
pop  2;
pop  2;
push 2 brown;
push 2 brown;
push 2 brown;
```

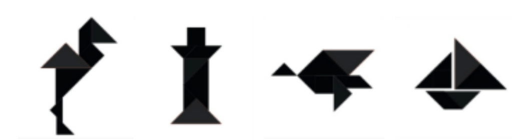
Scene



The person with a red shirt and a blue hat moves to the right end

```
remove_person 7
remove_hat    7
add_person    10 red
add_hat       10 blue
```

Tangrams

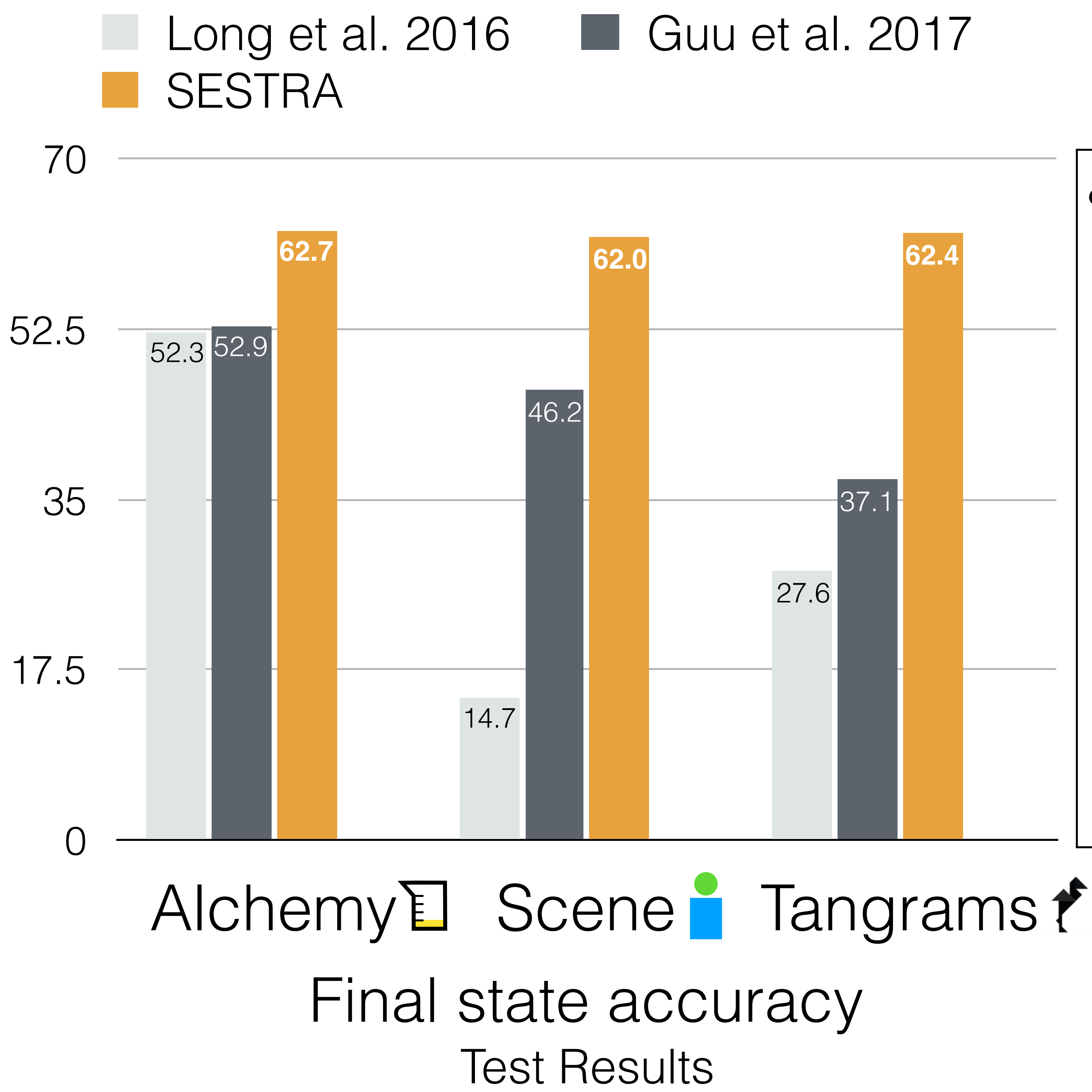


`1 2 3 4`

Swap the third and fourth figures

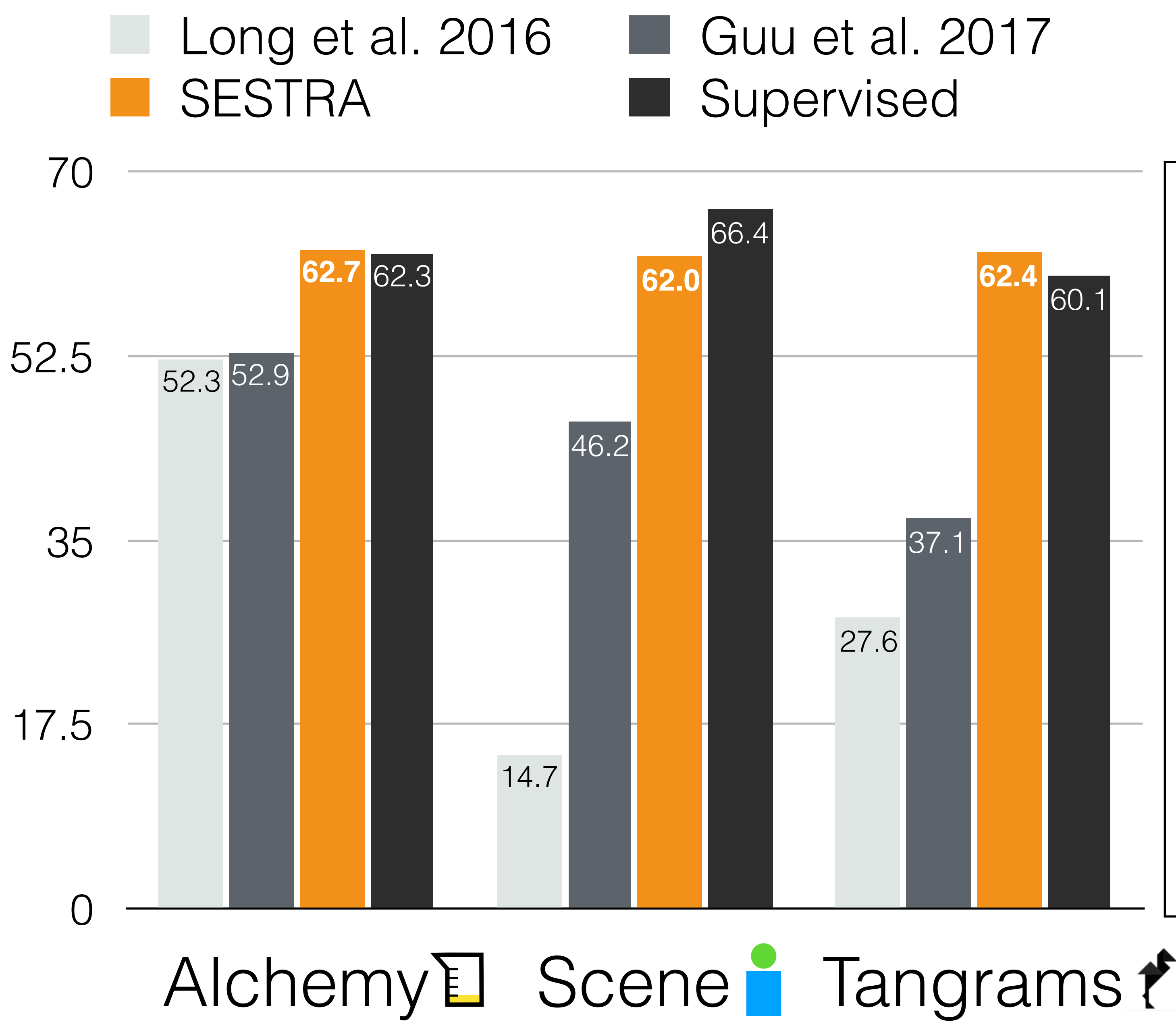
```
remove 4
insert 3 boat
```

Results



- Outperform previous methods by up to 25%, while mapping directly to system actions

Results

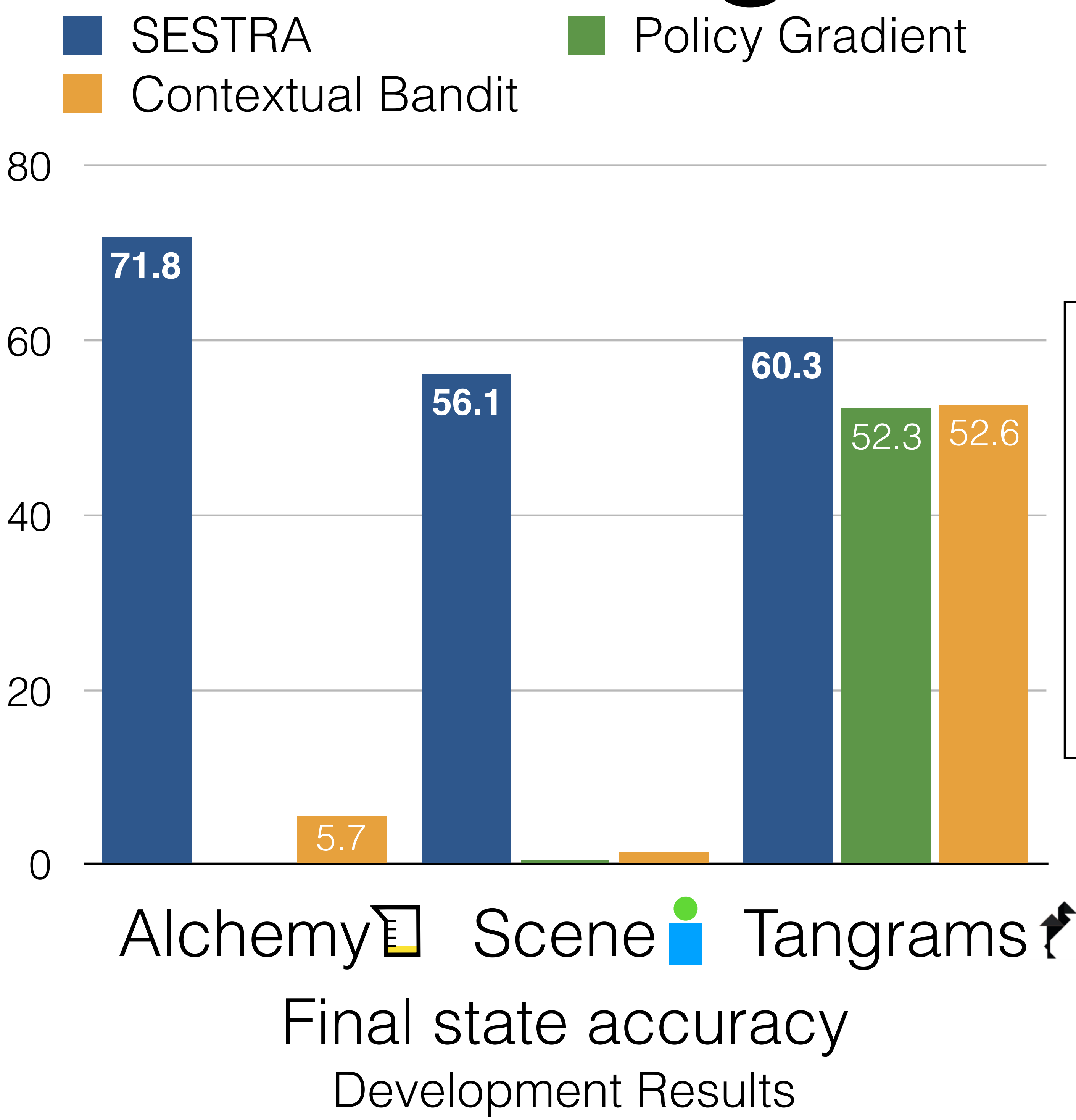


Final state accuracy

Test Results

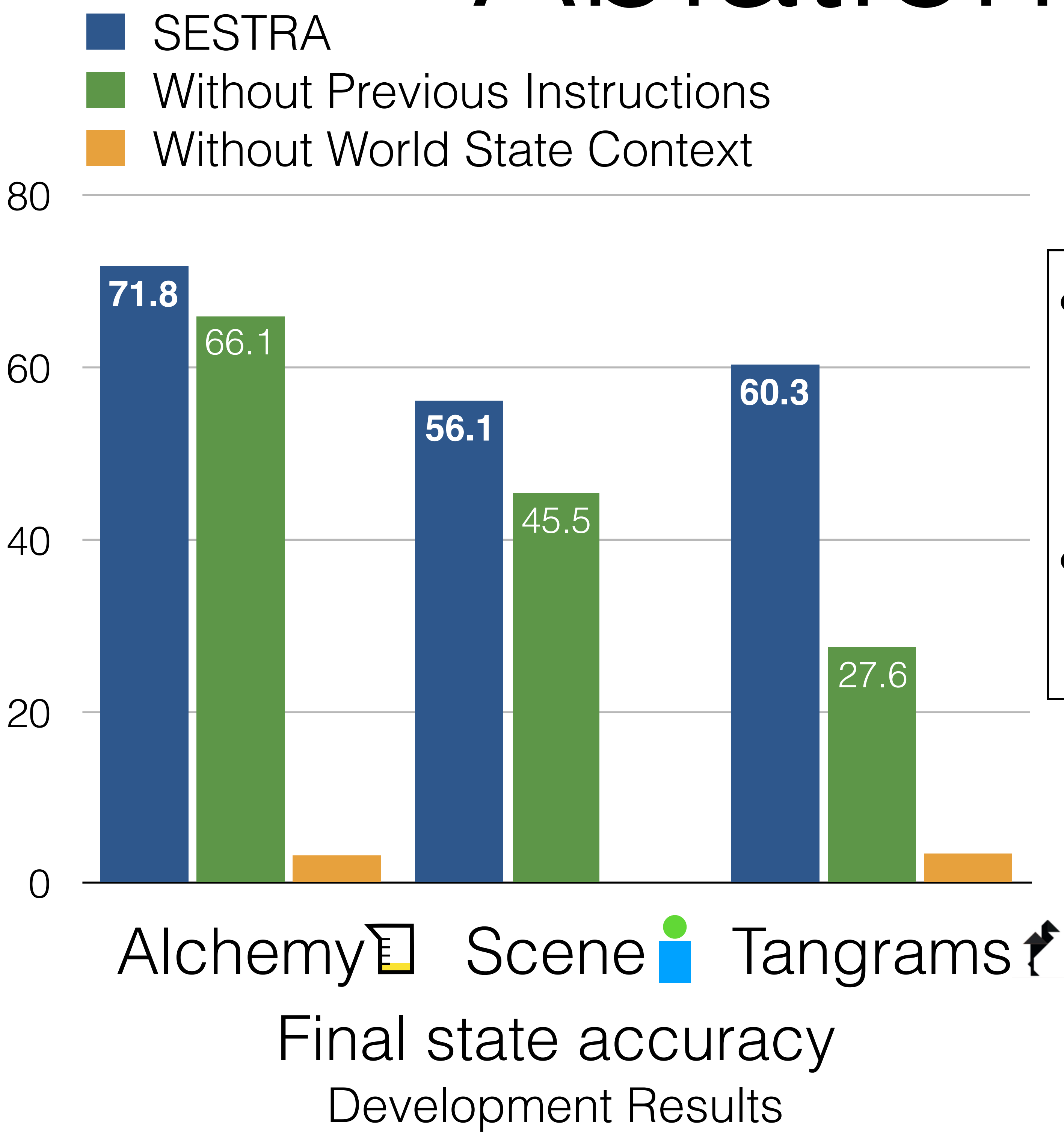
- Outperform previous methods by up to 25%, while mapping directly to system actions
- Performance is comparable to direct supervision

Learning Methods



- Single-step observations overcome biases that get model stuck

Ablations



- Need access to previous instructions
- Need access to world state

- Attention-based model for generating sequences of atomic actions that modify the environment

- Exploration-based learning procedure that avoids biases learned early in training

https://github.com/clic-lab/scone