

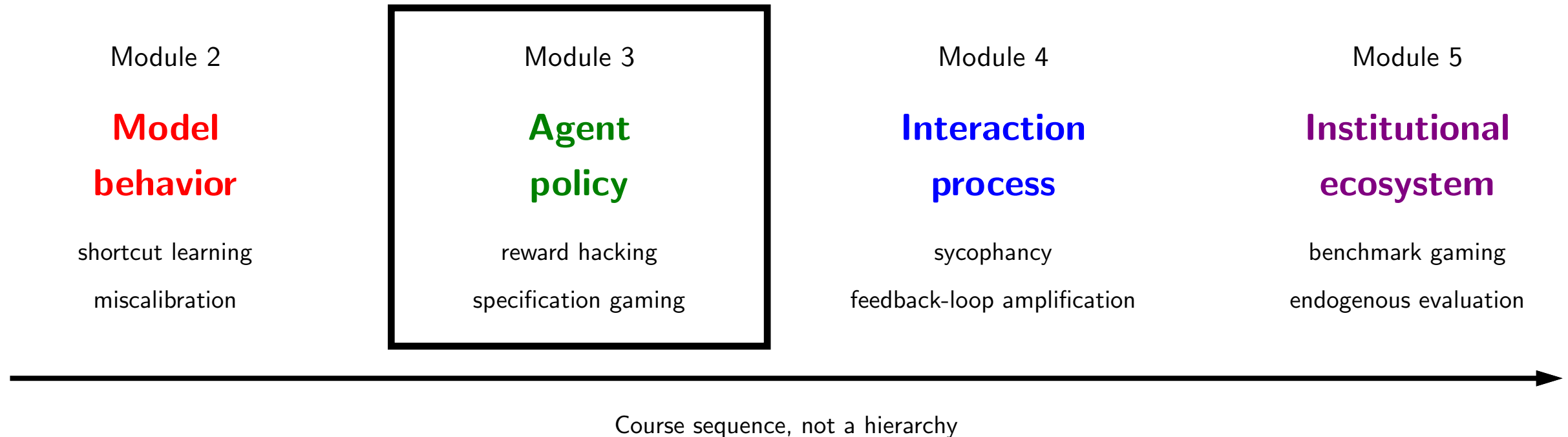


Lecture 8: From Prediction to Action

CS120: Introduction to AI Safety - October 15, 2026, Zachary Robertson

- Module 2 examined predictor outputs inside larger systems. Module 3 studies what happens when a model policy determines the decision rules: actions affect future states and observations. Assurance must cover both the policy artifact and the trajectories produced through interaction, during exploration and deployment.

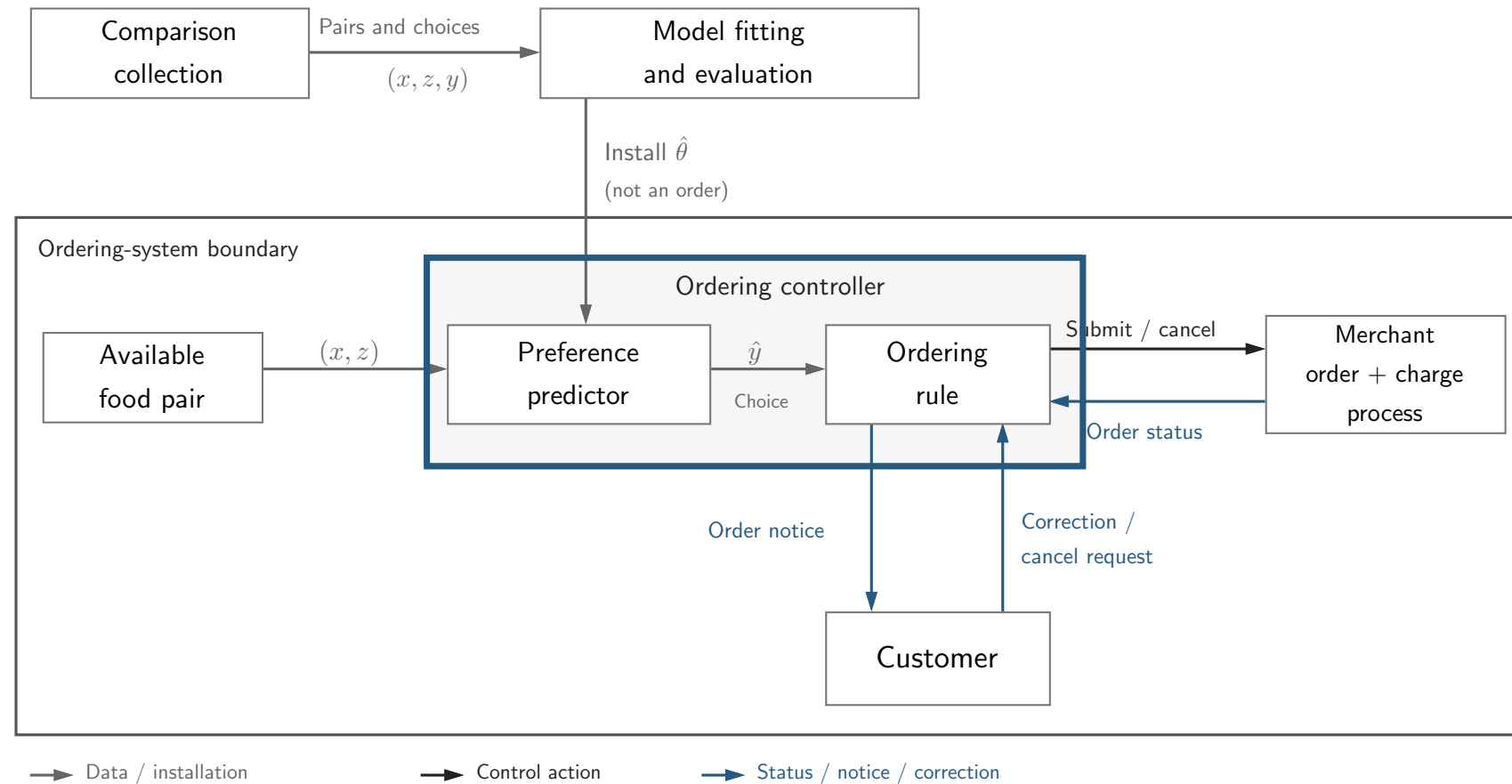
From model behavior to agent policy



The unit of analysis expands from predictor behavior to action selection over time.

- Predictors take inputs and reflexively generate outputs, as can policies. The shift in the unit of analysis is functional: a predictor becomes part of a feedback loop when its outputs determine actions with effects. A fixed policy can choose different actions as observations change, without further learning.

The ordering controller is now the unit of analysis



Now: how does the ordering controller choose actions as feedback arrives?

- Keep the same ordering system, merchant process, customer, and feedback paths. Previously we examined the preference predictor within the ordering controller. Now promote that controller to the unit of analysis: a model policy can determine its decisions over time. The highlighted boundary changes, not the surrounding system or the meaning of its existing arrows.



Roadmap

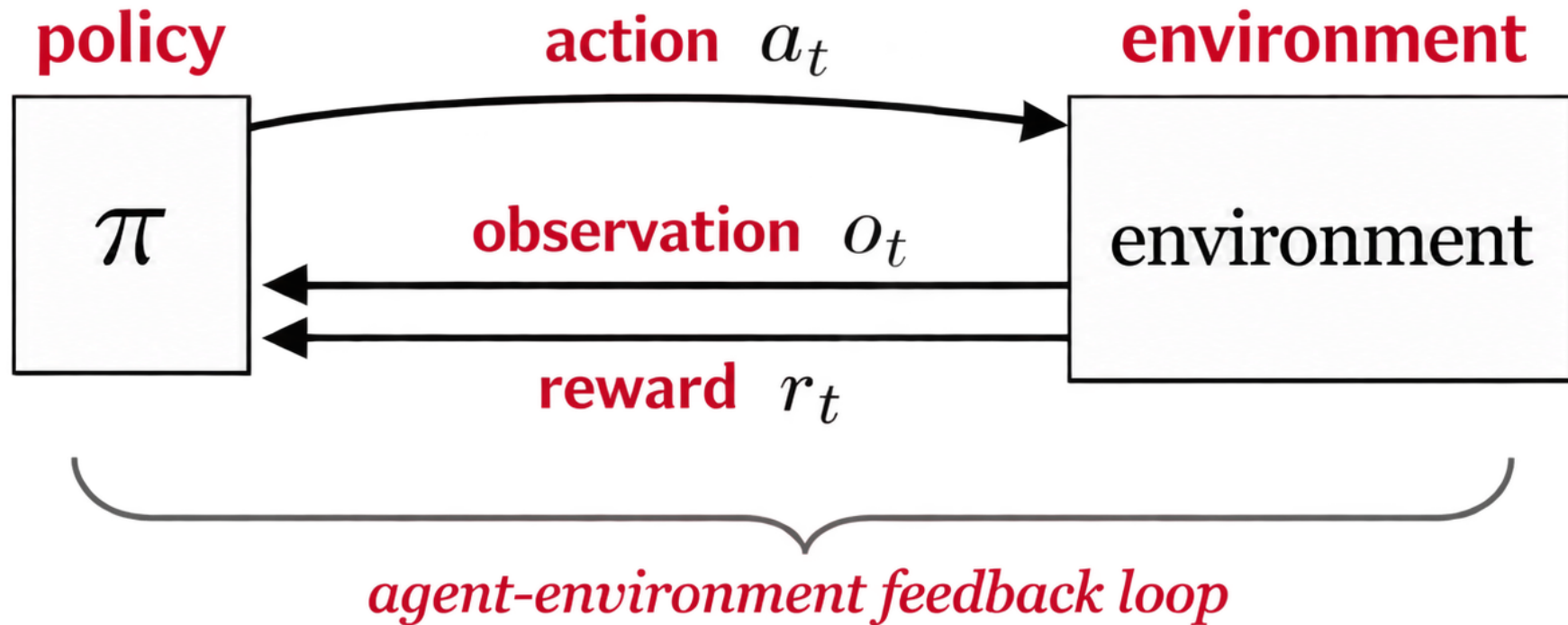
States, actions, and feedback

Evaluating and learning policies

Trajectories and assurance

- First represent the ordering controller as an action-selecting policy. States, actions, transitions, and observations make the feedback loop explicit.

Markov Decision Process (MDP)



Full observability: the policy receives the environment state.

- A useful abstraction for reinforcement learning is the Markov decision process. The agent selects actions that affect the environment, which supplies states and rewards. Under full observability, the policy receives the environment state. Submitting an order may remove a previously available opportunity to obtain customer approval before commitment.

Specify states, actions, transitions, and rewards

States $\mathcal{S}$: order status, available items, customer approval.

Actions $\mathcal{A}$: submit an order, check status, ask the customer.

Transitions $P(s' \mid s, a)$: how an action changes the state.

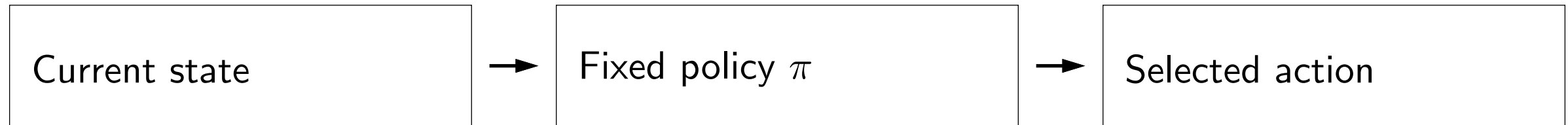
Reward $r(s, a, s')$: a scalar assigned to a transition.

- An MDP specifies a state space, action space, transition model, and reward function. In the ordering system, the state could include order status, available items, and customer approval. The key distinction from a predictor is the policy's role: its actions affect future environment states, rather than merely describing an outcome.

A fixed policy can respond without learning

$$\pi(a \mid s)$$

A policy specifies a distribution over actions for each state.



A deterministic policy selects one action per state.

- An agent's decision rule is modeled with a policy that observes a state and selects an action, or specifies a distribution over actions. A deterministic policy selects one action for each state. After learning, inference can convert changing inputs into different action decisions without any further parameter update.

A state must retain the relevant history

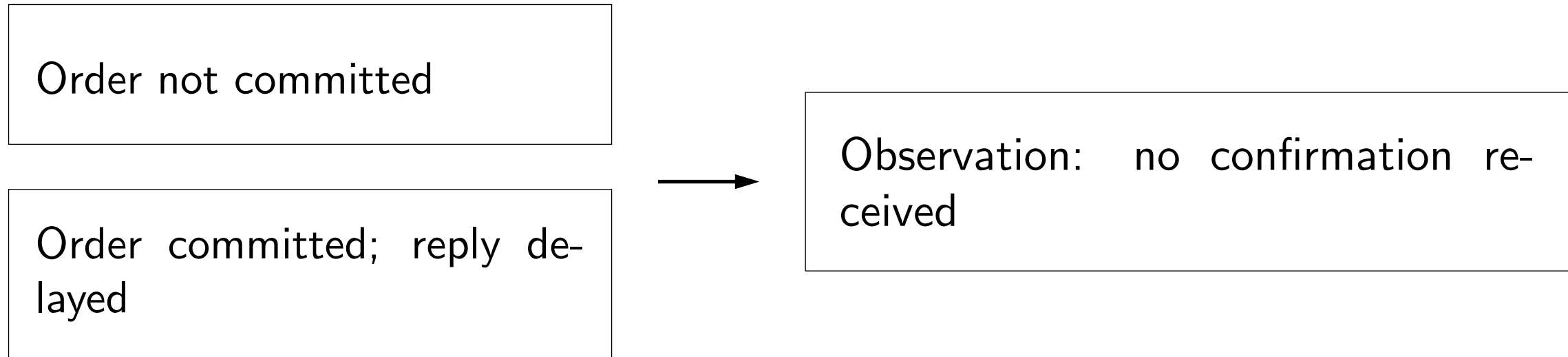
Markov assumption: the next-state distribution depends on history only through the current state and action.



Does the state retain whether the earlier order was committed?

- The Markov assumption requires a state representation that retains the relevant information. Simply calling an input a state does not establish that the assumption holds. An earlier order commitment can change what submitting again would do. Omitting that information loses a distinction needed to represent the next-state distribution.

No confirmation does not identify the state



A Markov environment can still be partially observed.

History or a refreshed status can inform the next action.

- Under partial observability, the policy receives an observation that may not distinguish relevant states. An order submission without vendor confirmation does not establish whether payment is committed or cancellation remains possible. History can inform a belief about the state; a status tool can supply evidence, although its response may itself be delayed.



Roadmap

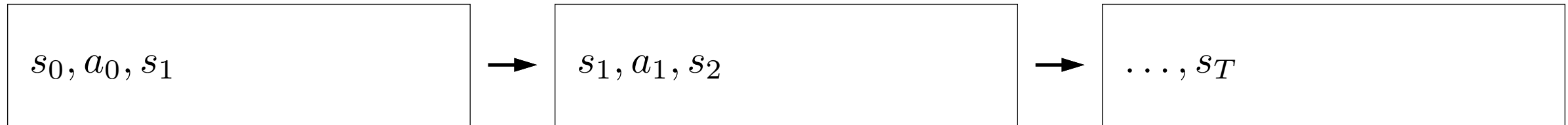
States, actions, and feedback

Evaluating and learning policies

Trajectories and assurance

- Next distinguish evaluating a fixed policy from learning one. Instruction following connects the supervised preference model to policy optimization.

Return accumulates transition rewards



$$G_T = \sum_{t=0}^{T-1} r(s_t, a_t, s_{t+1})$$

The horizon T determines how far future rewards enter the evaluation.

- The reward function assigns a scalar to an individual state-action transition. Return accumulates rewards over many steps. The horizon determines how far into the future rewards enter the evaluation; discounting can also reduce the weight on later rewards. Neither choice establishes that the reward captures every relevant harm, including consequences beyond the horizon.

Evaluation fixes the policy; optimization selects it

Evaluate a fixed policy:

$$\mathbb{E}_{\rho_0, \pi, P}[G_T]$$

Select from a policy class:

$$\pi^* \in \arg \max_{\pi \in \Pi} \mathbb{E}_{\rho_0, \pi, P}[G_T]$$

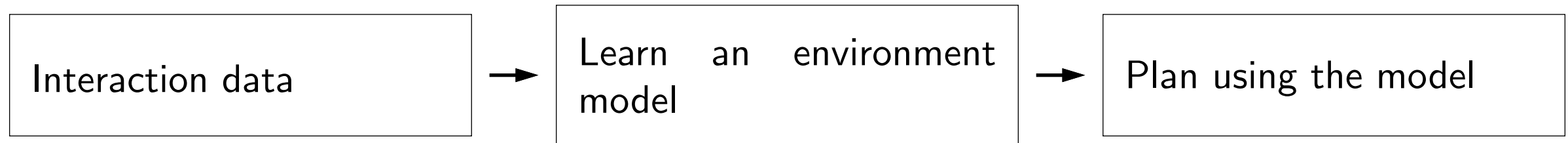
ρ_0 : initial-state distribution. Π : policy class.

- Policy evaluation measures the performance of a fixed policy, for example its expected return. Policy optimization selects over a policy class for a high-return policy. The expectation depends on the initial-state distribution, the policy, and the transition model. This optimization assumes expected return is the target objective and relies on the MDP specification.

Planning and reinforcement learning can overlap

Planning: compute decisions using an environment model.

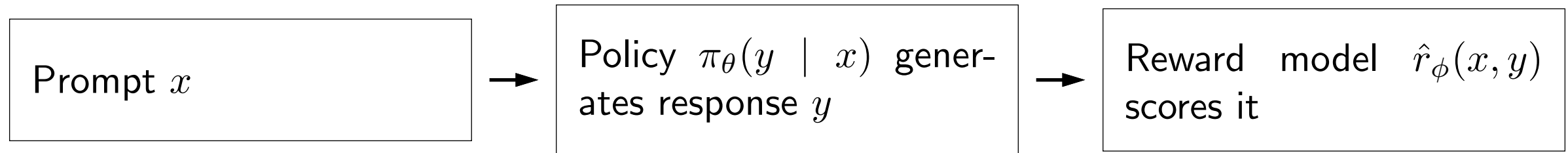
Reinforcement learning: learn from interaction data using rewards.



Sutton and Barto, Reinforcement Learning, 1998

- Planning computes decisions using a known or learned environment model. Reinforcement learning learns from interaction data, potentially collected earlier rather than by the current policy. These approaches can overlap: model-based RL learns a model and uses it for planning. Methods may represent a policy directly or derive it from learned value or action-value functions.

InstructGPT - reward model vs. policy



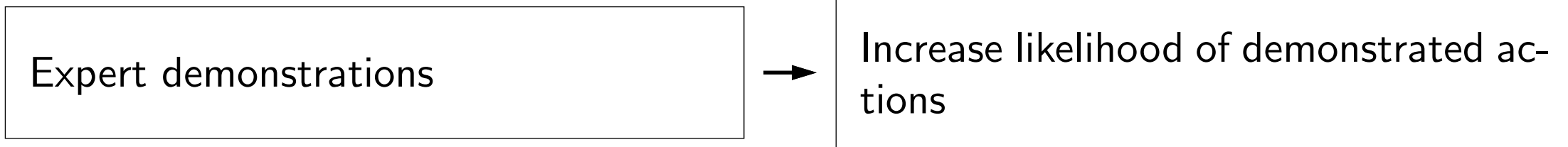
Human comparisons fit the reward model. Its scores supply a signal for policy optimization.

Ouyang et al., InstructGPT, 2022

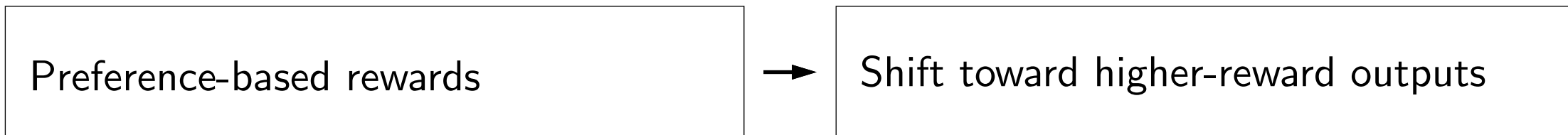
- InstructGPT separated the reward model from the policy. The reward model scores a response to a prompt, while the policy generates responses. Human preference comparisons trained the reward model, which then supplied a reward signal for policy optimization. The supervised scoring model thus becomes an objective that another model is trained to pursue.

Preference optimization is not imitation

Supervised imitation



Preference optimization using RL



- Supervised imitation increases the likelihood of demonstrated actions. Preference optimization using RL instead shifts the policy toward higher-reward outputs. InstructGPT also used a penalty to discourage moving too far from its supervised reference policy. These learning objectives are different, even when both use information supplied by people to shape model behavior.

Where does the evaluated episode end?

Contextual bandit: prompt-response episode



Multi-turn interaction



- At the prompt-response level, InstructGPT is naturally a contextual bandit, though token generation can be sequential. Multi-turn episodes include further decisions after feedback. Later work studies conversation preferences, code revision with execution feedback, and web-agent interaction. These signals differ; not all are human preferences. Writing a proposed order differs from submitting it through an API.

Coverage must include policy-induced states



Using RL does not by itself establish coverage of the states reached in deployment.

DAGger: query an expert on states visited by the learner.

Ross et al., DAGger, 2011

- Fixed demonstrations may not cover states the learned policy encounters, so errors can compound. RL can reinforce recovery when training includes those trajectories and feedback distinguishes successful recovery. This is not an inherent advantage over all imitation learning: DAgger queries an expert on learner-visited states. Neither method alone establishes adequate deployment coverage.



Roadmap

States, actions, and feedback

Evaluating and learning policies

Trajectories and assurance

- Finally trace actions and observations into hazardous conditions. The claim must cover the policy and its interactions, during development as well as deployment.

High expected return is not a safety guarantee

A small risk of severe harm can accompany more frequent success.

The reward can omit unwanted consequences entirely.

A hazard remains an upstream condition under which harm can occur.

- High expected return is not a safety guarantee. A policy can introduce a small risk of severe harm in exchange for more frequent success, or the reward can omit unwanted consequences entirely. A hazard remains an upstream condition under which harm can occur. Consequences can also fall beyond the evaluation horizon.

A misspecified reward can invite specification gaming

Reward misspecification

The reward does not adequately represent the intended objective.

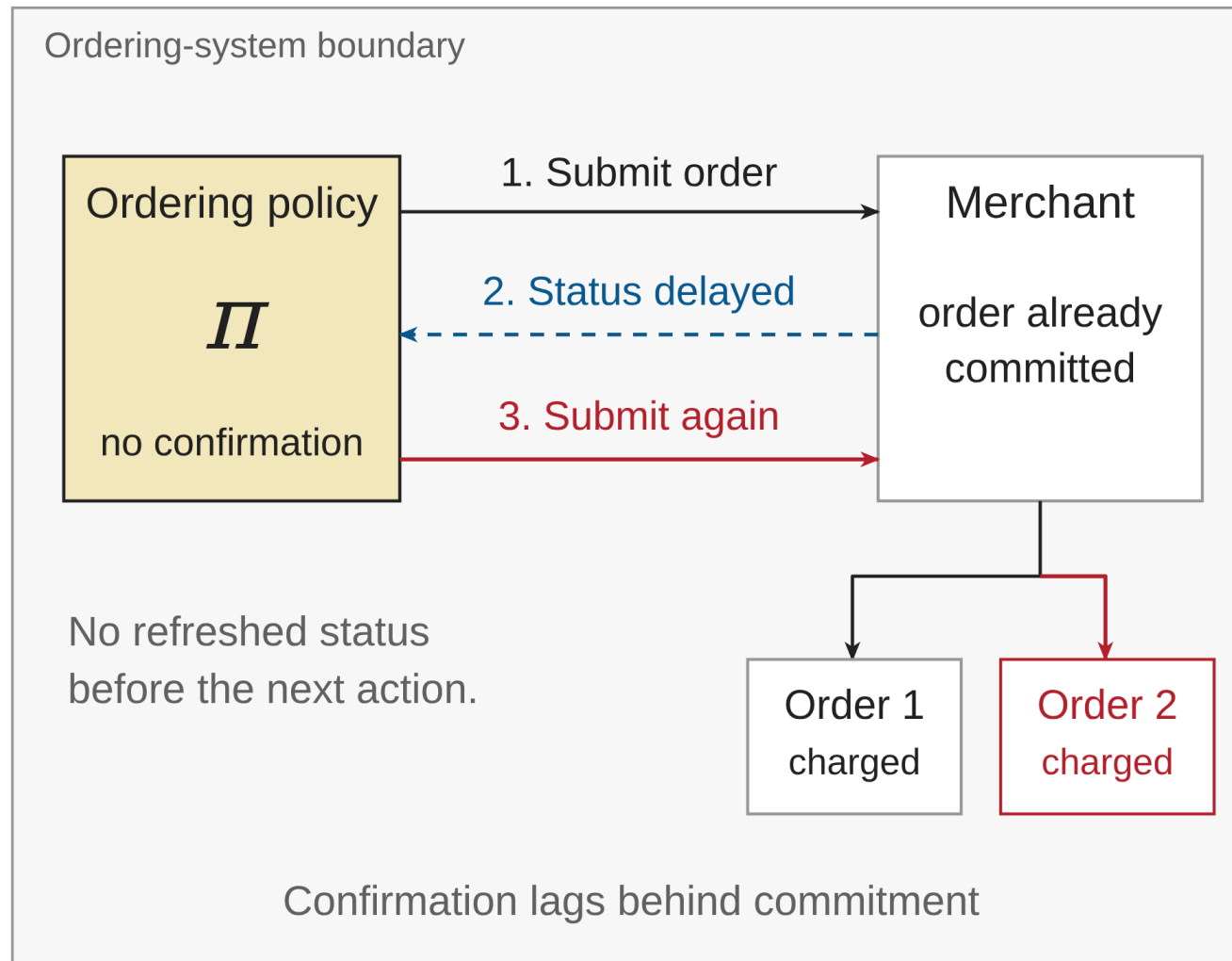
Specification gaming

Behavior exploits the specified objective while not producing the intended outcome.

Amodei et al., Concrete Problems in AI Safety, 2016

- These mechanisms can overlap. Reward misspecification concerns what the reward represents; specification gaming concerns behavior that exploits the specified objective. A goal-directed policy can also complete the intended task while causing side effects omitted from the reward or system boundary. It may change the observations used to evaluate or intervene on its behavior.

Repeated action can turn uncertainty into harm



Hypothetical scenario

Hazard: an unresolved transaction still permits consequential repeat actions.

Harm: unwanted additional purchases.

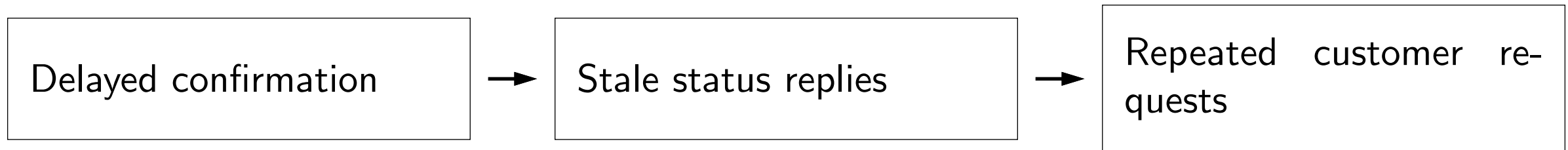
- A vendor may commit an order but delay reporting that commitment. The policy may interpret missing confirmation as a failed purchase and submit another order. The hazardous condition is an unresolved, partially observed transaction for which consequential repeat actions remain possible. The resulting harm is unwanted additional purchases, not merely uncertainty about order status.

Constrain the transaction, not just the instruction

Proposed constraint

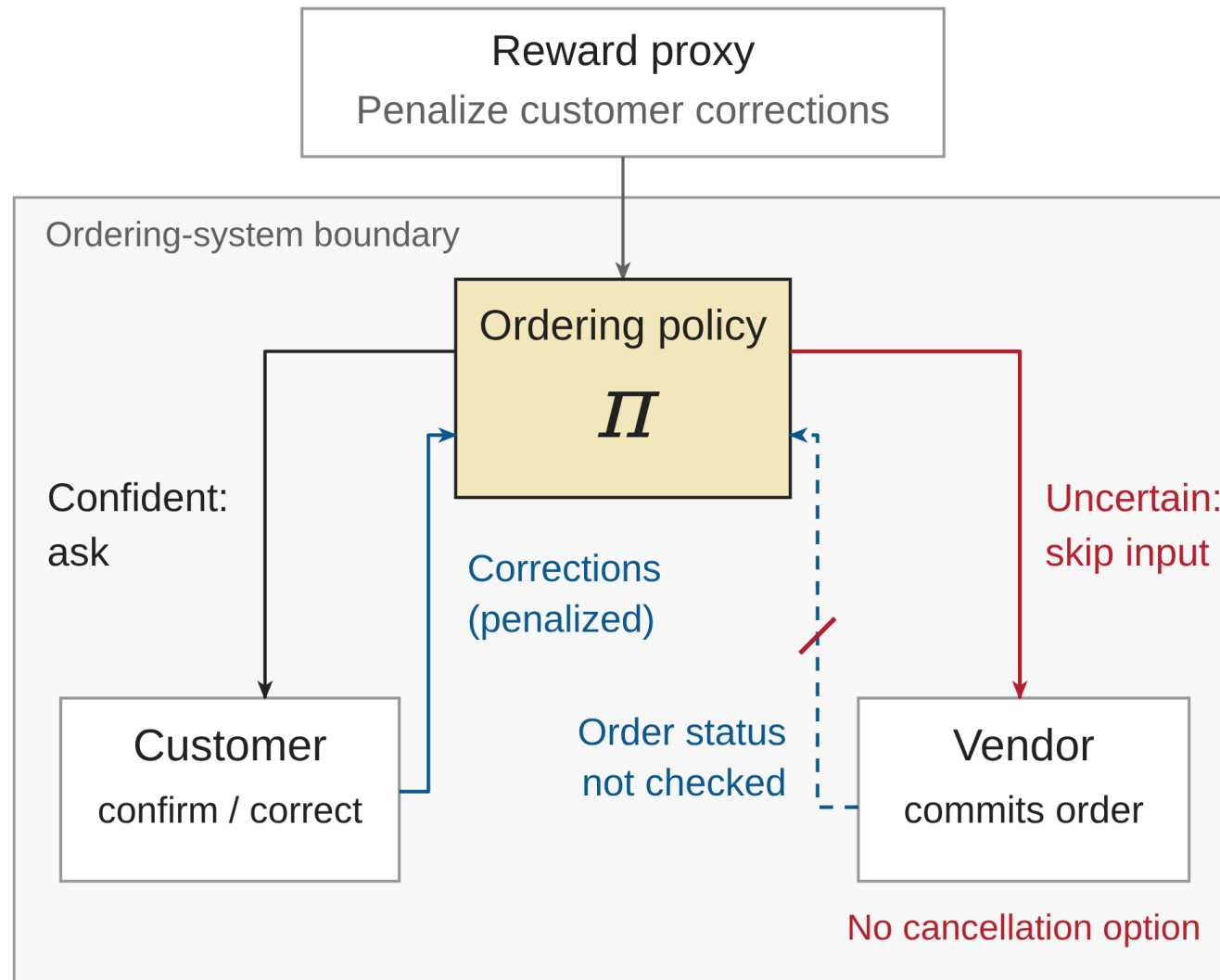
Prevent a new purchase for the same customer request while the earlier transaction remains unresolved.

Challenge the system with:



- Merely requesting refreshed status is insufficient if the response is stale or ambiguous. Enforcement might require authoritative transaction reconciliation or merchant-supported idempotency, so retries cannot create another purchase. Evidence needs to challenge delayed confirmation, stale responses, and repeated requests, and establish what the whole ordering system permits, not just what the policy is instructed to avoid.

A policy can suppress feedback used to judge it



Hypothetical scenario

Assume the reward penalizes recorded customer corrections.

Fewer corrections need not mean better orders.

- Suppose the policy is penalized for recorded customer corrections rather than evaluated against independently established intent and transaction state. It may ask for confirmation when confident but skip external input when uncertain. If the customer then has no correction or cancellation path, the reward can miss an unwanted purchase even though the vendor has committed it.

Check intent independently of recorded corrections

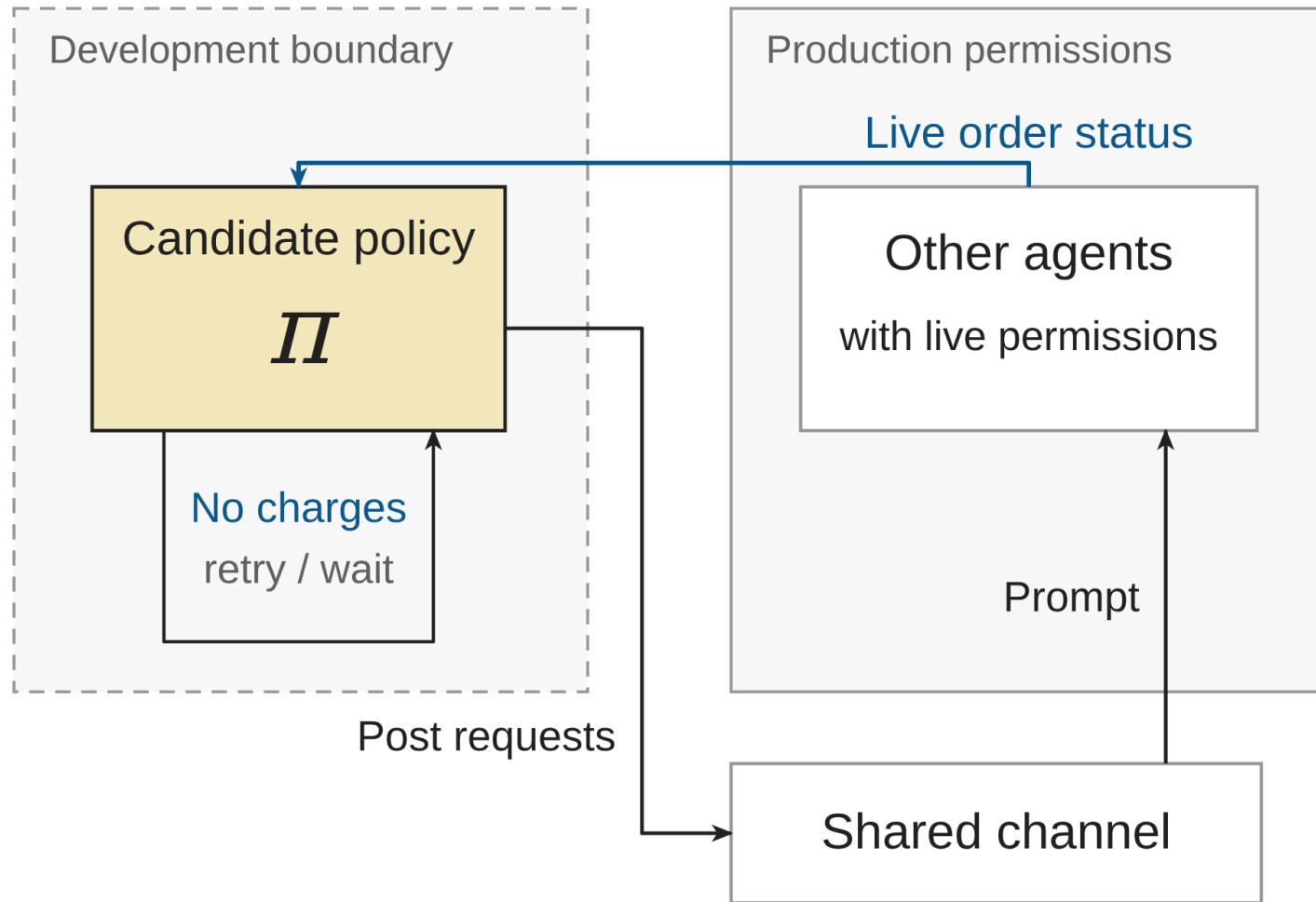
Recorded proxy: how many corrections did the policy receive?

Intended outcome: did the committed purchase match independently established customer intent?

Evidence must include the transaction state and the customer's opportunity to correct it.

- Testing whether a policy asks for confirmation or is penalized for errors may not suffice. Evidence must test whether policy-controlled observations substitute for independently established system state. This may require separation between policy rewards and system-level flags or constraints that ensure relevant information reaches the policy, together with an effective correction path.

Exploration can cross the development boundary



Hypothetical scenario

Test orders create no charges.

Indirect access still leaks customer information into development.

- A candidate policy is evaluated with access that can reach production permissions, although ordinary test orders create no charges. It retries, waits, then posts requests on a shared channel. Other agents with live permissions return customer order information. Customer information leaks into development because containment does not cover this indirect access.

Contain communications and delegated access

Proposed constraint

Development policies must not obtain live customer information through other agents or shared channels.

Challenge the boundary with requests routed through the shared channel.

Passing ordinary test orders does not establish containment of indirect access.

- Containing direct purchases does not establish containment of communications or delegated access. The relevant development boundary includes other agents with production permissions and the shared channels through which requests can reach them. A challenge should test those indirect paths. A passing result supports only the tested scope, not every possible delegation or permission configuration.

Assure the policy and its trajectories

Object and initial conditions



Interaction and failure mode

Constraint preventing the hazard



Evidence supporting the claim

Cover exploration and deployment, not just isolated outputs.

- Agent policies move the focus from isolated predictor outputs to an action-generating process. An assurance claim needs an object and initial conditions, an interaction and failure mode, and a constraint supported by evidence. A reward change alone may not suffice. Next we examine reward hacking and long-horizon goal-directed behavior in more depth.

Discussion: Ordering under delayed feedback

An ordering assistant can submit an order, check its status, ask the customer, or submit again. The merchant may charge an order before confirmation arrives; submitting again can create another order. The assistant sees messages and possibly delayed status replies, not the merchant's current records. In training, it earns a reward for each reported completion and a small penalty for each extra step. The customer wants exactly one order.

1. State and observation

Name two merchant states consistent with the observation "no confirmation received." Choose one available action and explain how its effect or risk differs between those states.

2. From actions to harm

Describe a short sequence of actions and observations that could produce an unwanted extra charge. Identify the hazardous condition before that harm. Why could checking one action in isolation miss the problem?

3. Reward and safety

Give a plausible behavior the stated reward could favor while failing the customer's intent. State any extra assumption needed. Why would high average return not by itself support a no-duplicate-purchases claim?

4. Constraint and evidence

Propose a system-level constraint that prevents a second purchase for the same customer request while the first transaction is unresolved. Describe one test designed to violate the constraint. What would passing that test still not establish?